Universidade Federal de Santa Catarina (UFSC)
Departamento de Informática e Estatística

Computação Distribuída
INE 5625

Prof. Mario Dantas

http://www.inf.ufsc.br/~mario

Objetivo

Neste curso vamos estudar os conceitos relativos a computação distribuída, abordando paradigmas tais:

- *computação obíqua;*
- *computação pervasiva;*
- *computação colaborativa;*
- *ambientes de troca de mensagens;*
- *memória compartilhada; e*
- *objetos distribuídos.*

Objetivo

Vamos iniciar nosso curso com as novas abordagens de computação distribuídas, estas conhecidas como *ubíqua*, *pervasiva e colaborativa.*

O objetivo é introduzir uma visão inovadora de ambientes computacionais altamente distribuídos, que necessitam de um tratamento especial por parte de especialistas e, por outro lado, ser *ignorado* pelos seus usuários.

Objetivo

Quanto as bibliotecas de *troca de mensagem* serão estudadas: *PVM (Parallel Virtual Machine) e MPI (Message Passing Interface).*

Ambientes de *middlewares* orientados à *memória compartilhada (shared memory),* tais como Jiajia e ThreadMark, serão também estudados.

Objetivo

Finalmente, os objetos distribuídos serão abordados através das tecnologias *CORBA*, *DCOM* e *Java RMI*

Conteúdo Programático

Modulo I

. Introdução a computação distribuída

. Ambientes de middleware

Modulo II

. Paradigma de troca de mensagem

. Parallel Virtual Machine (PVM)

. Message Passing Interface (MPI)

. Ambientes de Memória Compartilhada

Conteúdo Programático

Modulo III

. Objetos distribuídos

. Componentes

. CORBA

. Barramento de objetos CORBA

. Adaptador básico de objetos

. Linguagem de definição de interfaces

Conteúdo Programático

. Serviços CORBA

. Objetos COM

. Barramento de objetos COM

. Serviços OLE/COM

. Java/RMI

Bibliografia Recomendada

• Coulouris, Dollimore e Kindberg, Distributed Systems: Concepts and Design, ISBN: 0201619180, Addison Weley, 2006

• Tanenbaum e van Steen, Distributed Systems: Principles and Paradigms (2nd Edition), ISBN: 0132392275, Prentice Hall, 2000

•Dantas, Computação Distribuída de Alto Desempenho: Redes, Clusters e Grids Computacionais, ISBN: 8573232404, Axcel Books 2005. (Download: www.inf.ufsc.br/~mario)

Conteúdo Programático

Modulo I

. **Introdução a computação distribuída**

. Ambientes de middleware

Modulo II

. Paradigma de troca de mensagens

. Parallel Virtual Machine (PVM)

. Message Passing Interface (MPI)

. Ambientes de Memória Compartilhada

1.1 - Introdução a computação distribuída

- Aplicações têm se tornado a cada dia mais complexas. Um exemplo é a utilização de diversos tipos de mídia (voz, dados, imagens);

- O crescimento numa imaginado da Internet e redes;

- A *transparência* provida pela Web;

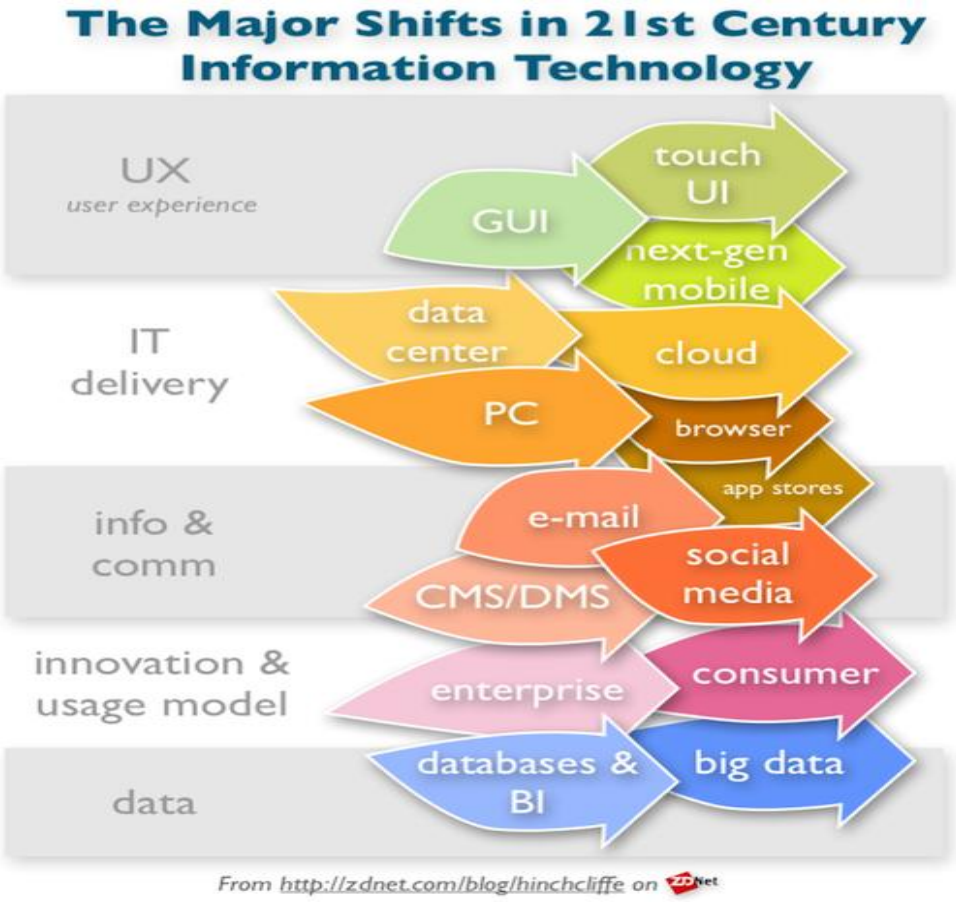
Introdução a computação distribuída

- Computadores pessoais se tornam cada vez mais acessíveis;

- Redes e dispositivos móveis sem fio tornam-se um padrão de comunicação pessoal e empresarial.

Exemplo típico

- The "Big Five" IT trends of the next half decade: Mobile, social, cloud, consumerization, and big data.



[Dion, 2011]

Big Data

O que seria a ***big data*** ????

(***giga, tera ou penta***)

Big Data

Conceitos

Atualmente as empresas estão se deparando com uma ***enorme quantidade*** de dados, nunca antes imaginado.

O desafio é o ***acesso*** a esse "mar de dados".

Os ***ciclos*** de negócios estão ***crescendo*** e cada vez ***mais curtos***. Então, torna necessário a ***observação*** do fluxo de dados de negócios novos e existentes.

Objetiva-se o ***processamento*** rápido o suficiente para auxílio na tomada ***decisões críticas***.

Big Data

Conceitos

O termo ***big data*** foi criado para descrever novas ***tecnologias*** e ***técnicas*** que podem lidar com uma ***ordem de magnitude*** de dados muito ***superior*** aquela suportada pelos gerenciadores de ***banco de dados***.

Objetiva-se o ***acesso*** a essa massa de dados de uma ***forma escalável ou custo-benefício.***

Big Data

Conceitos

A abordagem de ***big data*** oferece um ***melhor ROI*** em conjuntos de dados corporativos importantes.

Capacidade de ***resolver problemas*** inteiramente novos de negócios que antes eram ***impossíveis de resolver*** com as técnicas existentes.

Big Data

Conceitos

Muitas empresas ainda estão no ***estágio*** de preocupação com armazenamento de dados (***data warehouse***), enquanto que o mais produtivo seria conhecer a abordagem do ***big data***.

Big Data *Desafios*

Big data requer novos perfis

Há uma série de tecnologias avançadas e de novas plataformas para aprender a ser eficaz com dados grandes.

Departamentos de TI devem se preocupar com essas novas habilidades que devem adquirir (ou promover internamente) para tirar melhor proveito do novo paradigma.

Big Data

Desafios

Funções apropriadas

O uso significativo de dados grandes requer uma considerável abordagem matricial nativa.

O paradigma de big data requer o acesso a data warehouse, silos e sistemas externos com novas técnicas.

Big Data

Desafios

Funções apropriadas

A abordagem SOA (no passado) teve desafios similares porque tinha que coordenar e alinhar tantas partes do negócio.

Por outro lado, no paradigma big data existe a necessidade de uma grande cooperação em toda a empresa e com fornecedores externos, o que não é uma tarefa fácil.

Big Data

Desafios

A abordagem de ***big data*** requer uma mudança de mentalidade, tanto quanto uma atualização de tecnologia.

Pode-se dizer que dar ***prioridade a colocar os dados abertos*** na empresa, bem como ***melhorar a velocidade operacional*** que anteriormente não tem sido uma prioridade.

Big Data

Desafios

Big data permite resolver problemas de negócios em espaços que ***não eram possíveis antes***.

Adicionalmente a ***infra-estrutura e o desenvolvimento*** devem fazer parte da responsabilidade da mesma equipe, que devem ser acostumados a trabalhar juntos.

Os refinamentos organizacionais devem ser feitos para explorar um maior potencial da organização.

Big Data *Tecnologias*

Exemplos Instituicionais

ORACLE

http://www.oracle.com/us/technologies/big-data/index.html?sckw=srch:big-data3&SC=srch:big-data3

Intel

http://www.intel.com.br/content/www/br/pt/big-data/big-data-analytics-turning-big-data-into-intelligence.html?cid=sem155p6427

MICROSOFT

http://www.microsoft.com/sqlserver/en/us/solutions-technologies/business-intelligence/big-data-solution.aspx

Big Data Tecnologias

Diverse Data Sets

Big Data:

Decisions based on all your data

Video and Images 

Documents 

Social Data 

Machine-Generated Data 

Information Architectures Today:

Decisions based on database data

Transactions

ORACLE FUSION APPLICATIONS

ORACLE SIEBEL

SAP

Big Data Tecnologias

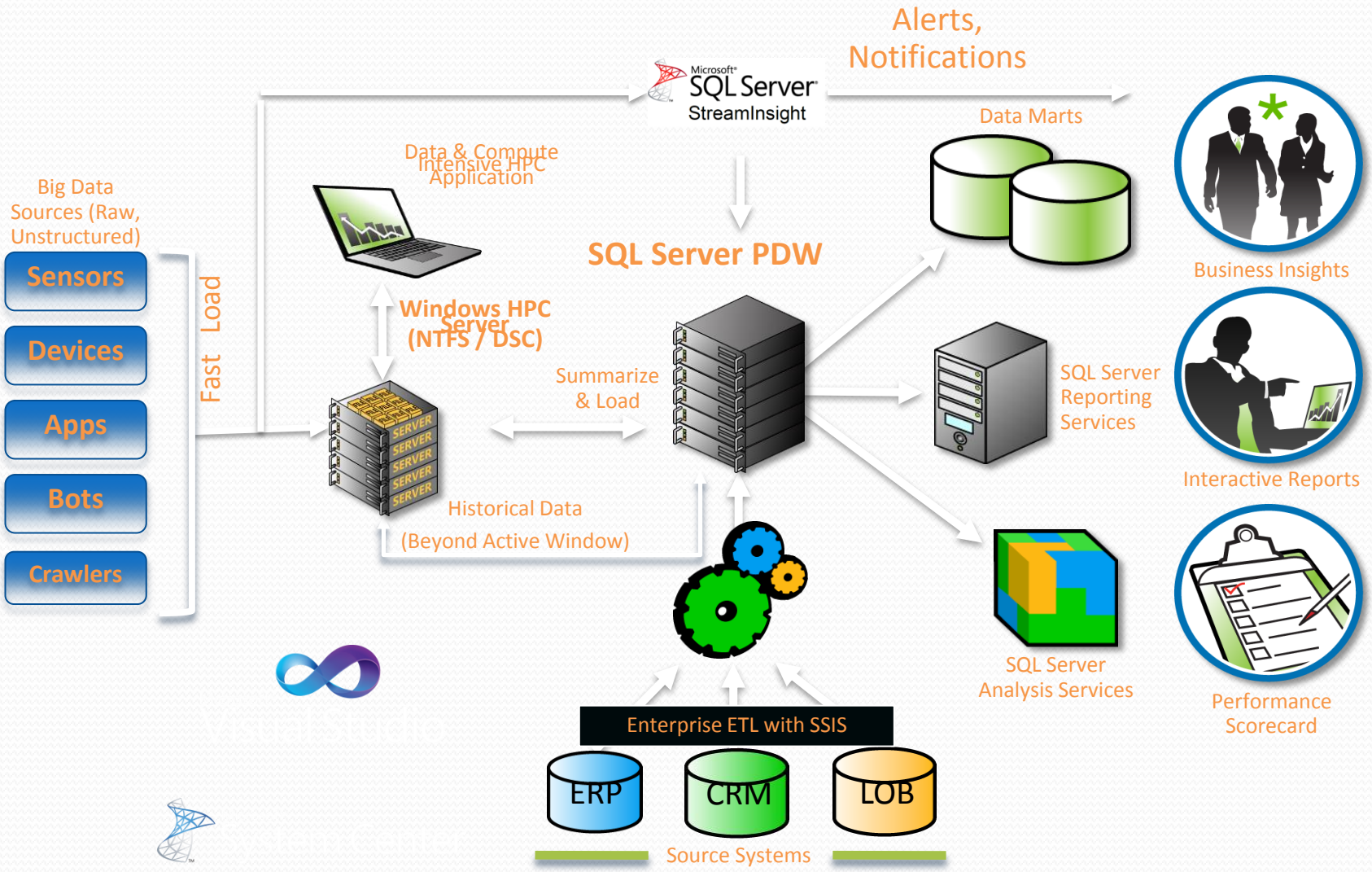
Why Is Big Data Important?

US HEALTH CARE	MANUFACTURING	GLOBAL PERSONAL LOCATION DATA	EUROPE PUBLIC SECTOR ADMIN	US RETAIL
Increase industry value per year by	Decrease dev. assembly costs by	Increase service provider revenue by	Increase industry value per year by	Increase net margin by
$300 B	**–50%**	**$100 B**	**€250 B**	**60+%**

Source: * McKinsey Global Institute: Big Data – The next frontier for innovation, competition and productivity (May 2011)

Big Data Tecnologias

Microsoft Big Data Solution



Big Data *Tecnologias*

Exemplos Pacotes de Software

Internet

Google File System
S3 (Amazon)
Hadoop

Big Data

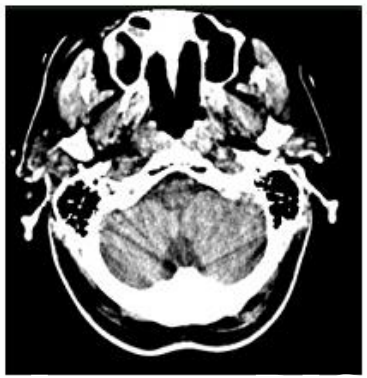
Tecnologias

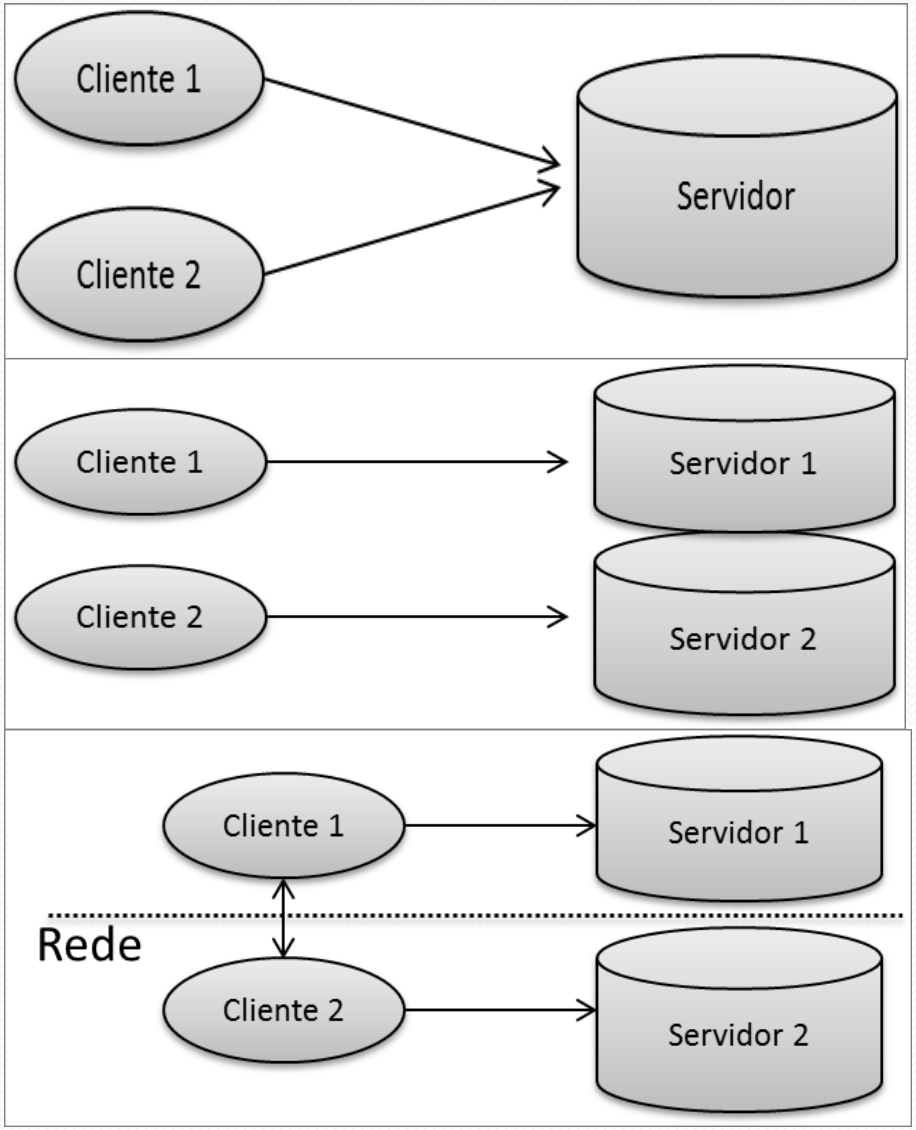
Exemplos Pacotes de Software

Intranet

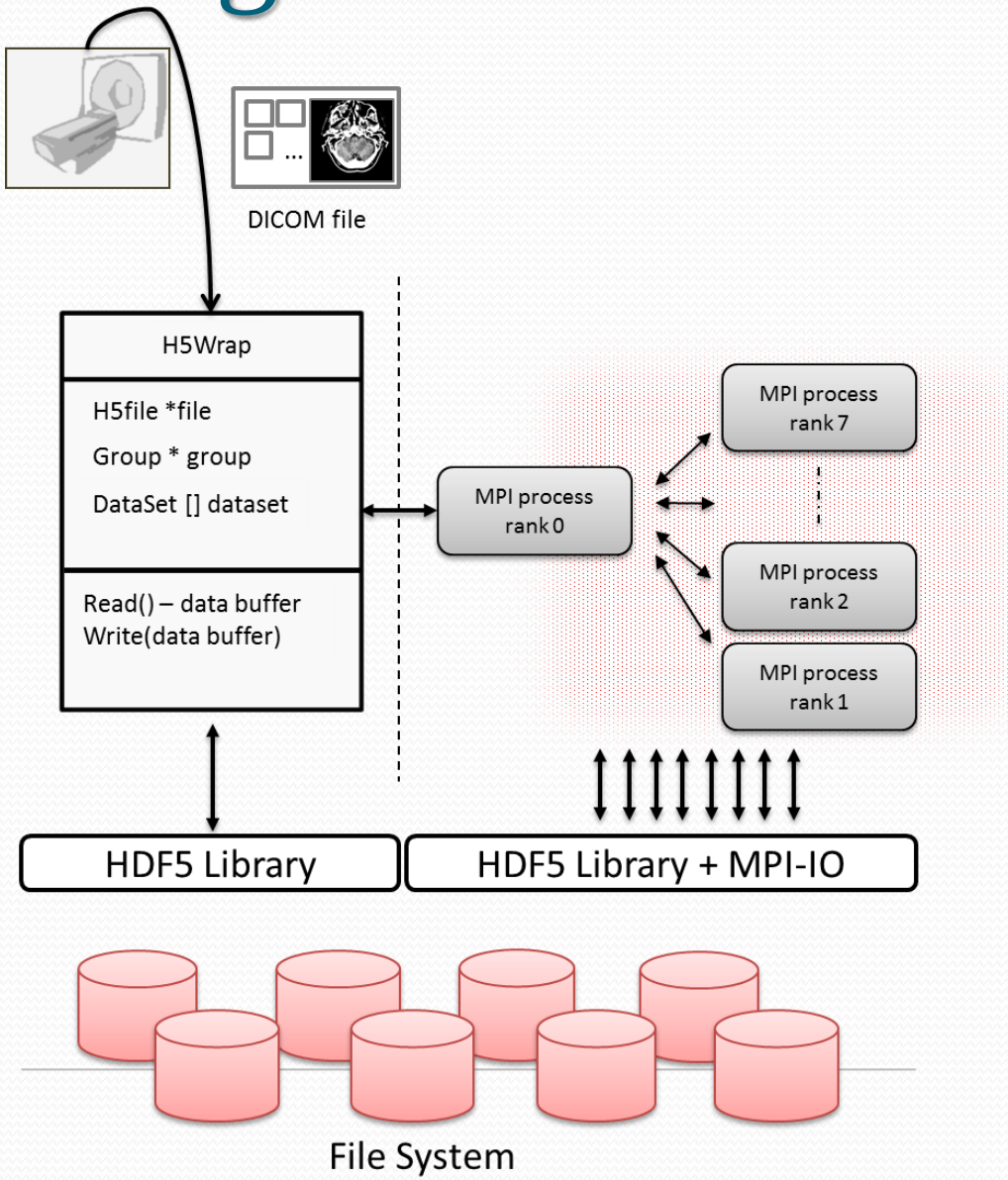
HDF5
+
PVFS
FhGFS
Lustre
Ceph

Big Data *Estudo de Caso*



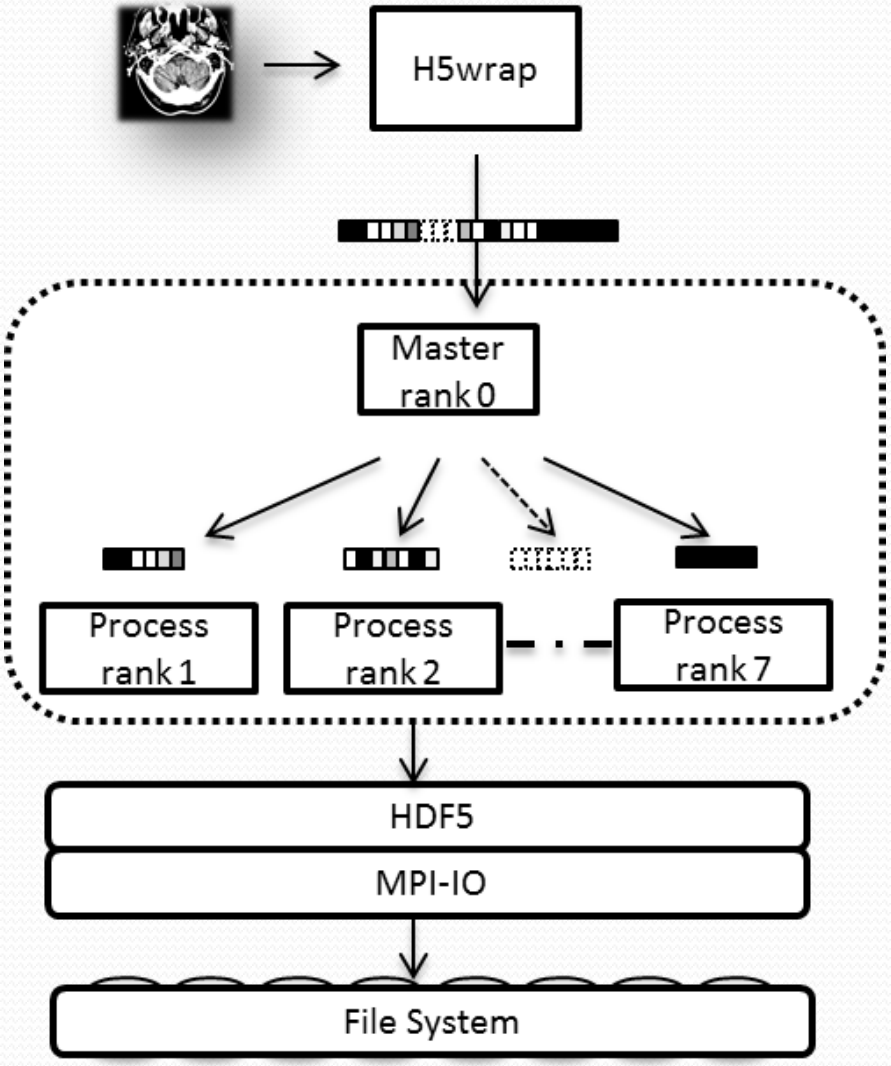


Big Data *Estudo de Caso*



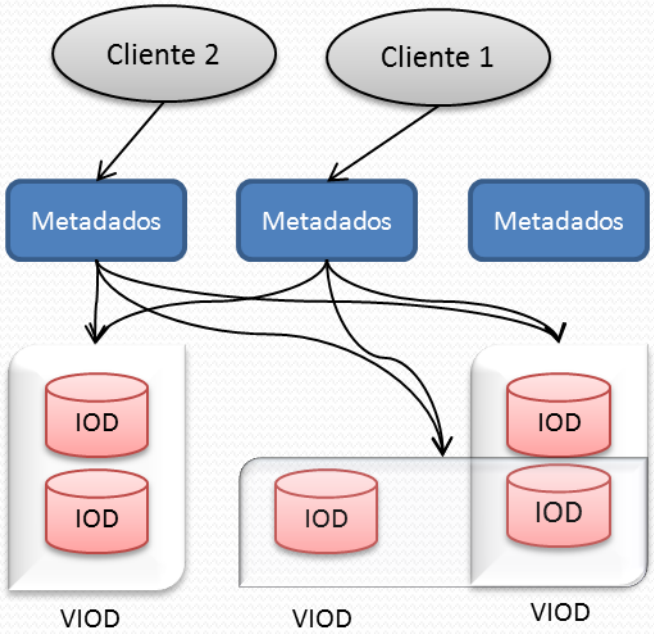
Big Data

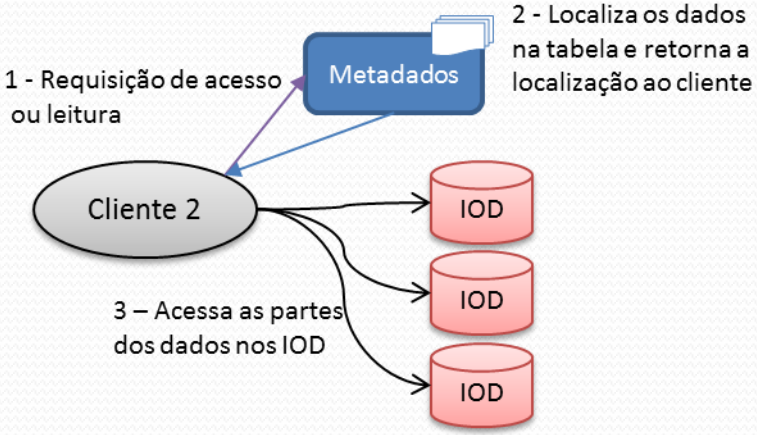
Estudo de Caso

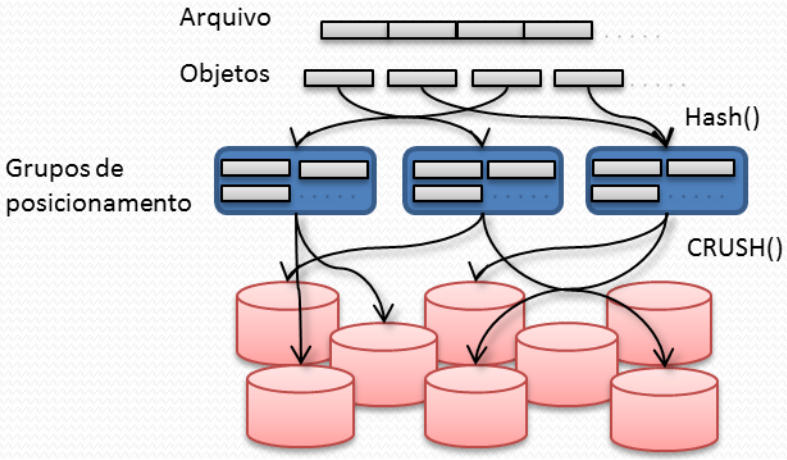


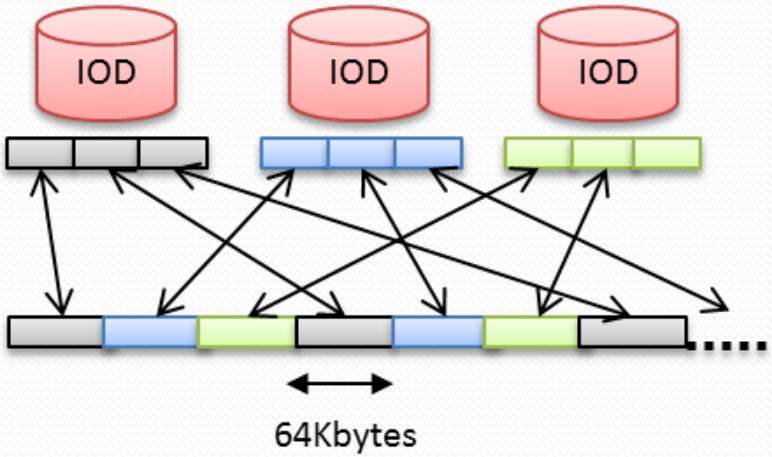
Big Data

Estudo de Caso



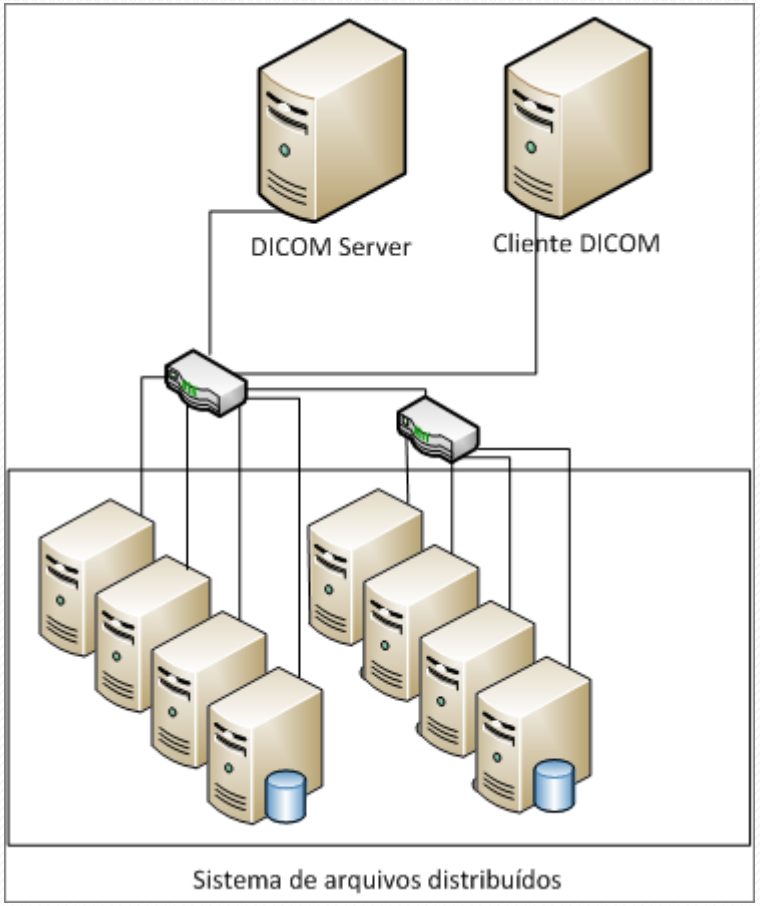




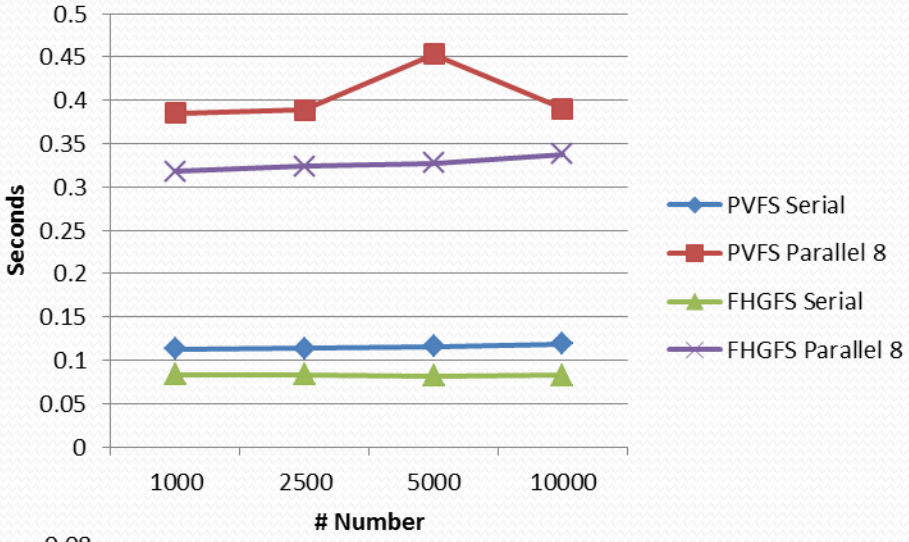


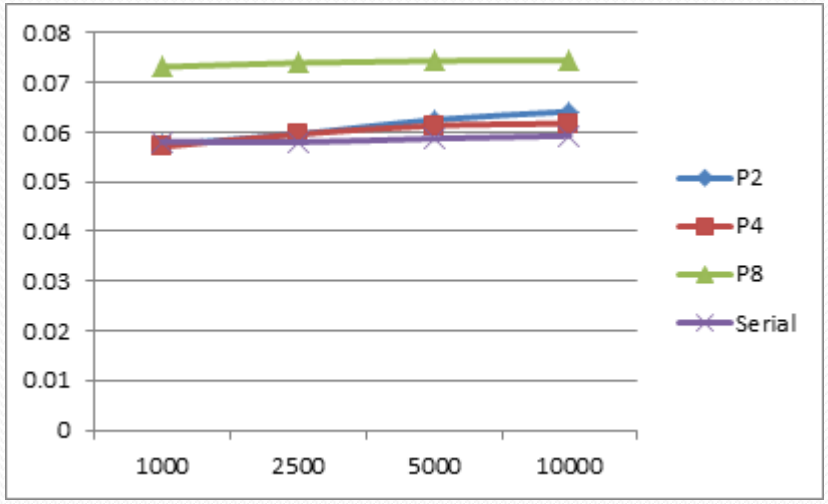
Big Data

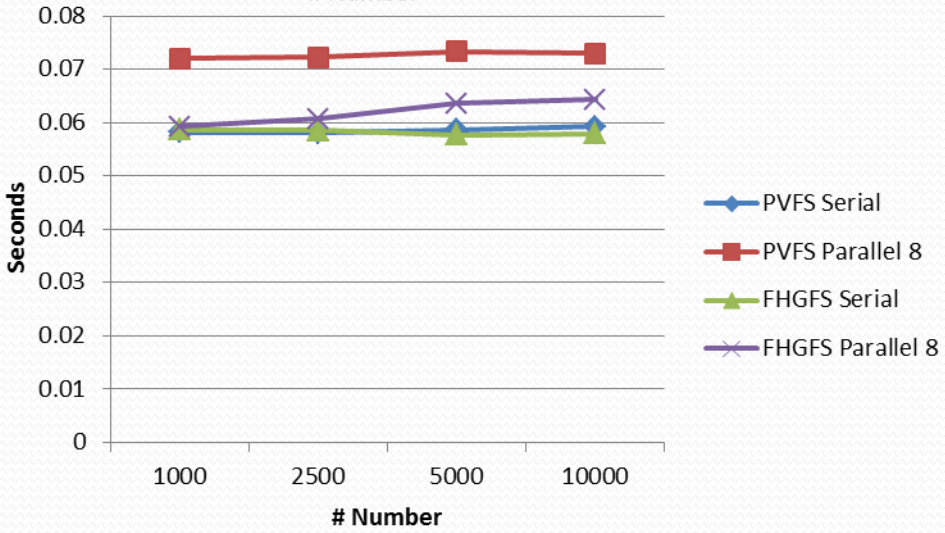
Estudo de Caso

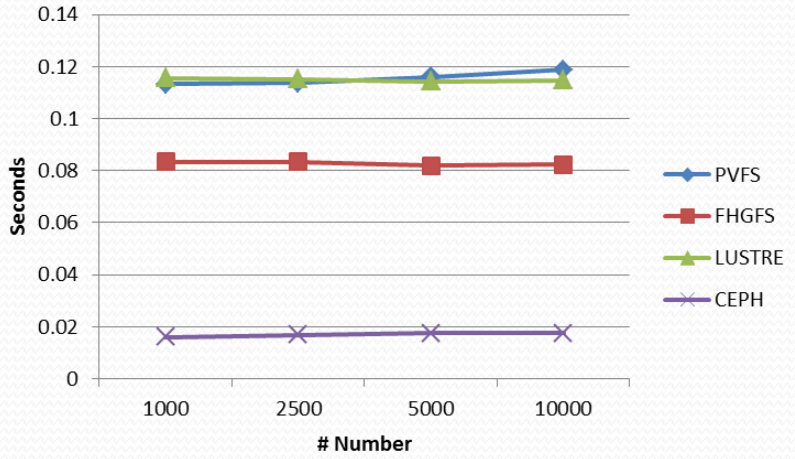


Big Data *Estudo de Caso*









Fatores a serem considerados

- Eficiência
  - desenvolvimento da solução;
  - tempo de execução.
- Confiabilidade
- Transparência
  - flexibilidade;
  - tolerância à falha X alta disponibilidade.

Fatores a serem considerados

- Codificação da Aplicação
  - ✔ fácil utilização
- Computação da Aplicação
  - ✔ precisa;
  - ✔ segura;
  - ✔ eficiente.

Fatores a serem considerados

- Aplicação Web
  - ✔ flexível;
  - ✔ Poder ser replicada;
  - ✔ eficiente.

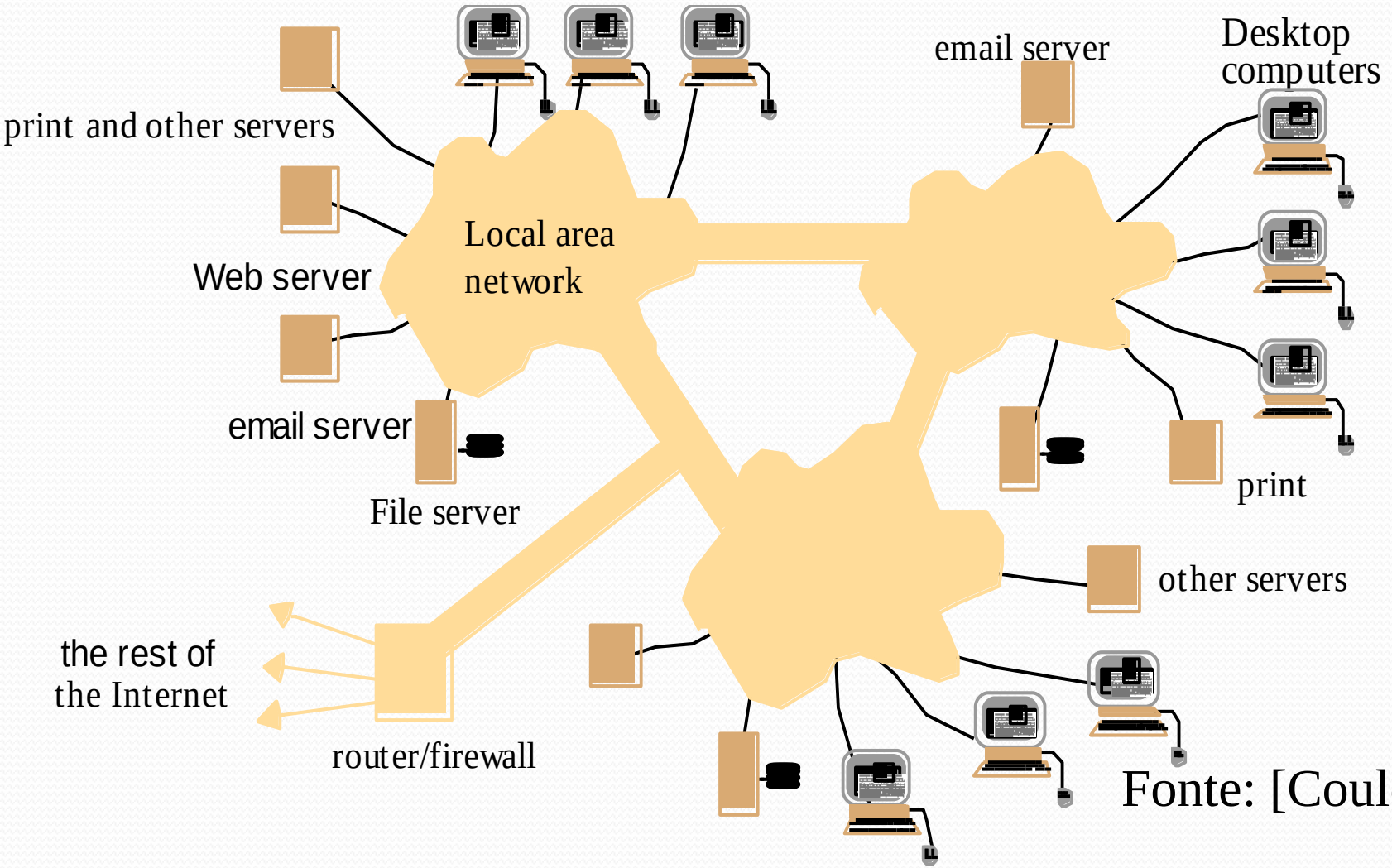
Possíveis soluções

- C/C++ aplicações + Sockets
  - não possui uma maior escalabilidade
  - problemas de empacotamento/desempacotamento
  - muitos protocolos temporários
- DCE (remote procedure invocation)
  - não existe polimorfismo

Possíveis soluções

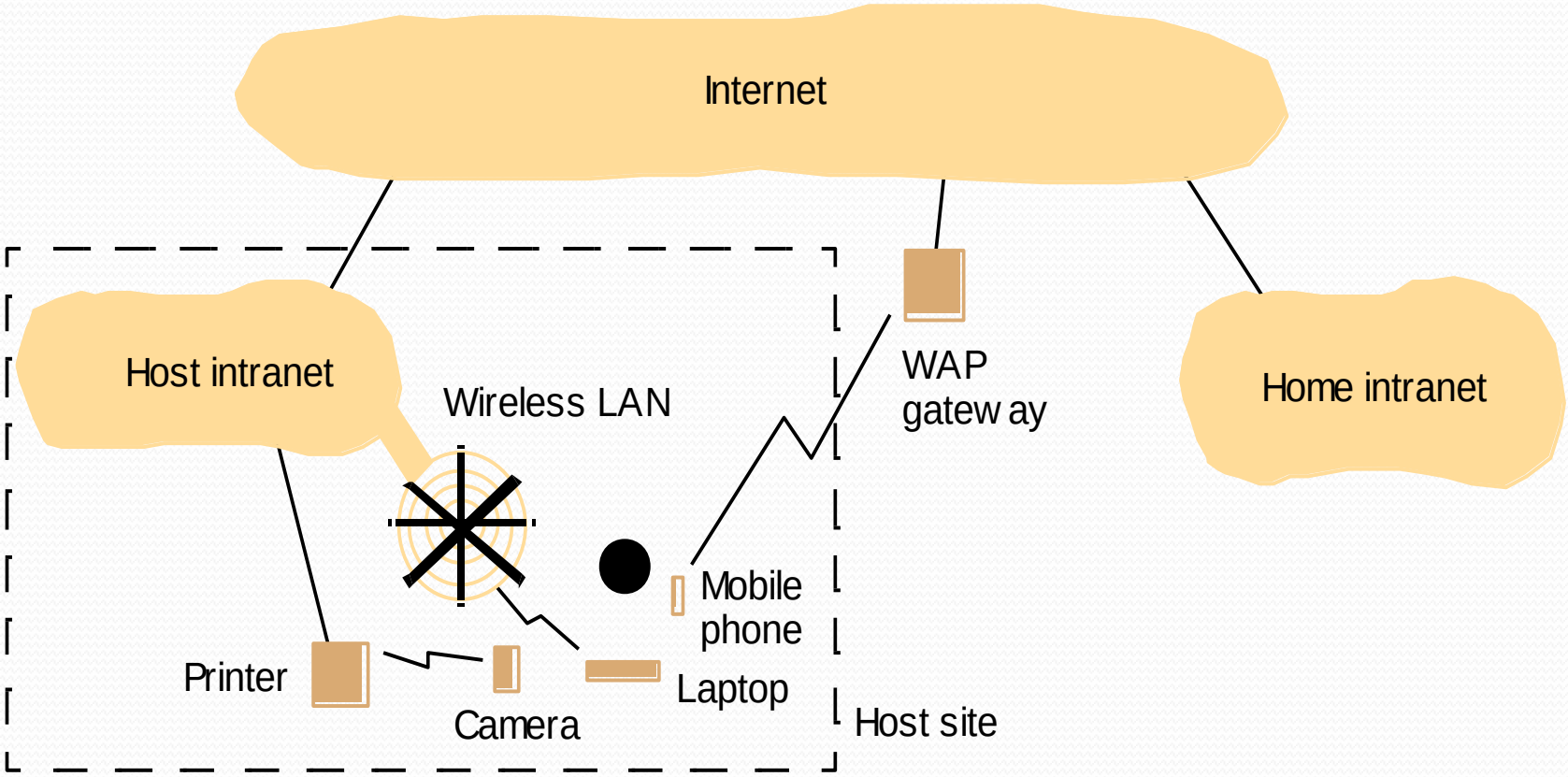
- Ambientes de troca de mensagem;
- Pacotes de memória compartilhada distribuída;
- Objetos distribuídos.

Uma típica Intranet



Fonte: [Coulouris]

Dispositivos móveis e portáveis em sistemas distribuídos



Fonte: [Coulouris]

Computadores vs. Web servers na Internet

Date	*Computers*	*Web servers*	*Percentage*
1993, July	1,776,000	130	0.008
1995, July	6,642,000	23,500	0.4
1997, July	19,540,000	1,203,096	6
1999, July	56,218,000	6,598,697	12
2001, July	125,888,197	31,299,592	25
		42,298,371	

Fonte: [Coulouris]

Introdução a computação distribuída

Como podemos definir um sistema distribuído ?

Introdução a computação distribuída

Como podemos definir um sistema distribuído ?

No mundo real os sistemas distribuídos, sob o aspecto de arquitetura de máquinas para execução de aplicativos, devem ser vistos como configurações com grande poder de escala pela agregação de computadores existentes nas redes.

Fonte: [Dantas]

Introdução a computação distribuída

Como podemos definir um sistema distribuído ?

Nos ambientes distribuídos, a homogeneidade ou heterogeneidade de

um conjunto de máquinas, onde cada qual possui sua arquitetura de software-hardware executando sua própria cópia do SO, permite a formação de interessantes configurações de SMPs, de MPPs, de clusters e de grids computacionais.

Fonte: [Dantas]

Tipos de Transparências

1. Access transparency: enables local and remote resources to be accessed using identical operations;

2. Location transparency: enables resources to be accessed without knowledge of their physical or network location (for example, which building or IP address);

3. Concurrency transparency: enables several processes to operate concurrently using shared resources without interference between them;

Fonte: [Coulouris]

Tipos de Transparências

4. Replication transparency: enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers;

5. Failure transparency: enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components;

Fonte: [Coulouris]

Tipos de Transparências

6. Mobility transparency: allows the movement of resources and clients within a system without affecting the operation of users or programs;

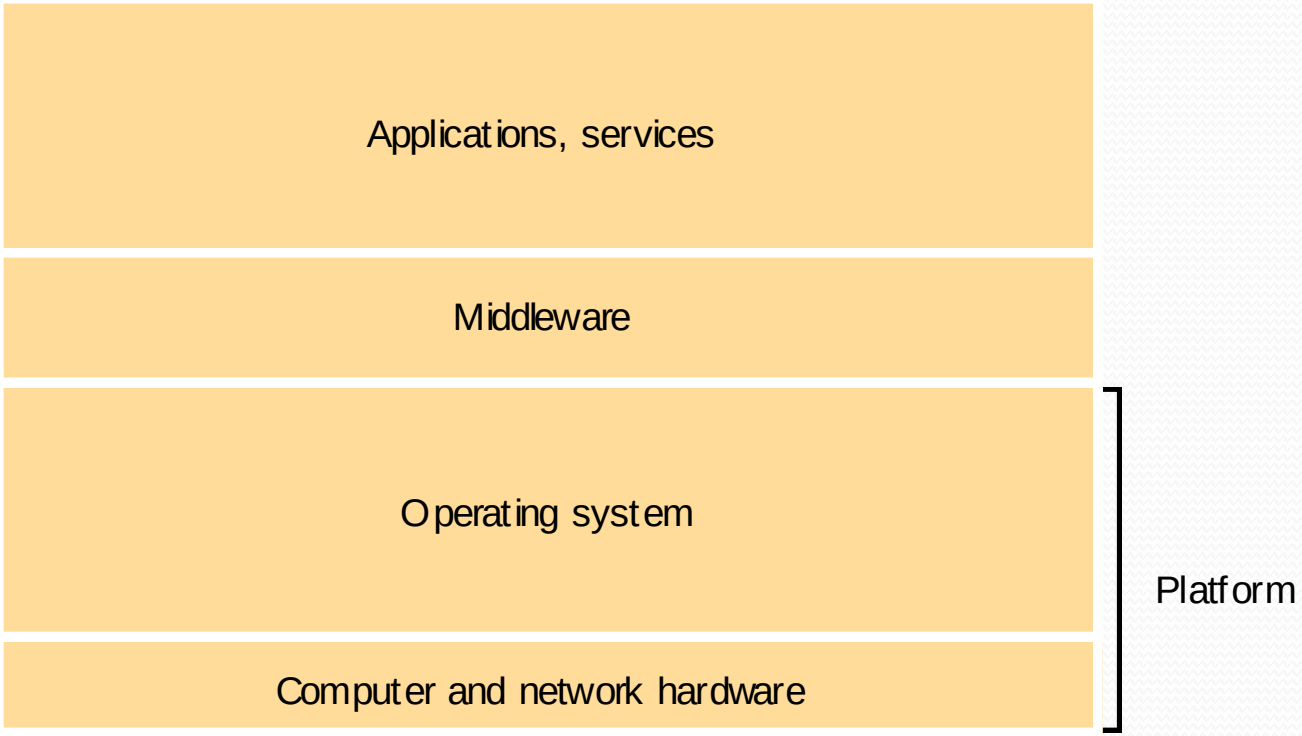
7. Performance transparency: allows the system to be reconfigured to improve performance as loads vary;

8. Scaling transparency: allows the system and applications to expand in scale without change to the system structure or the application algorithms.

Fonte: [Coulouris]

Sistemas Distribuídos: Modelos de Arquitetutras

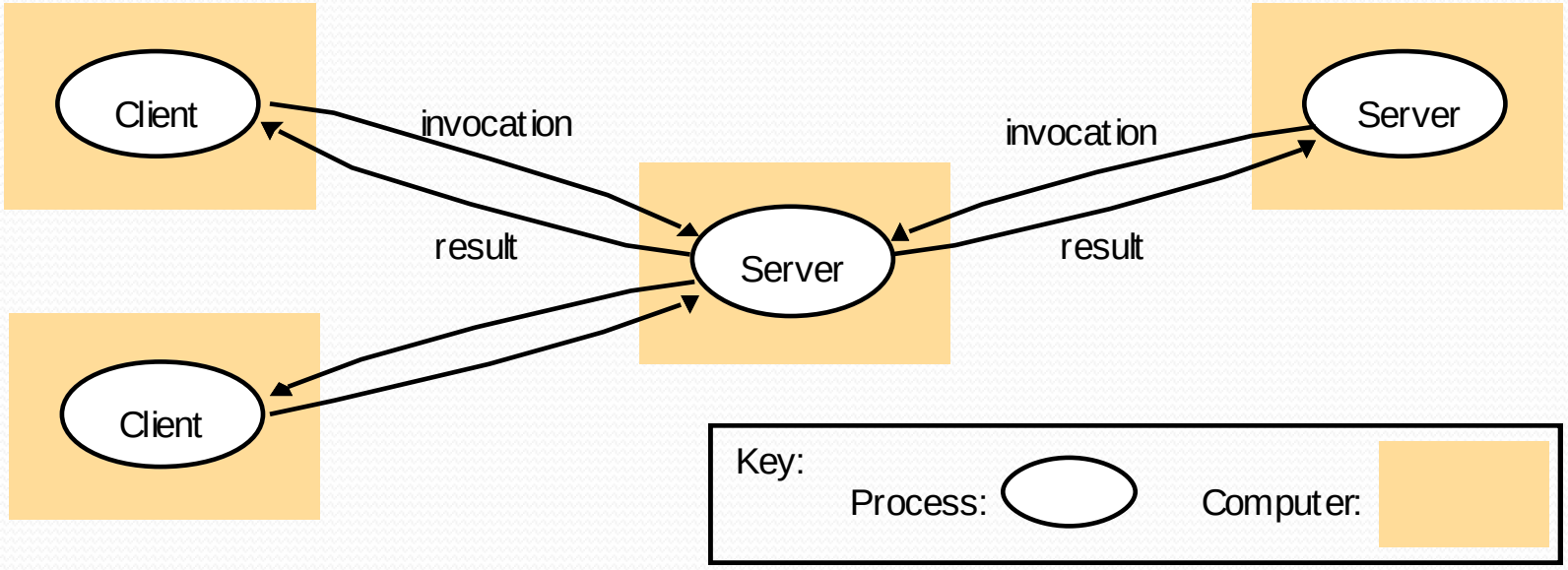
Modelos de Arquiteturas



Fonte: [Coulouris]

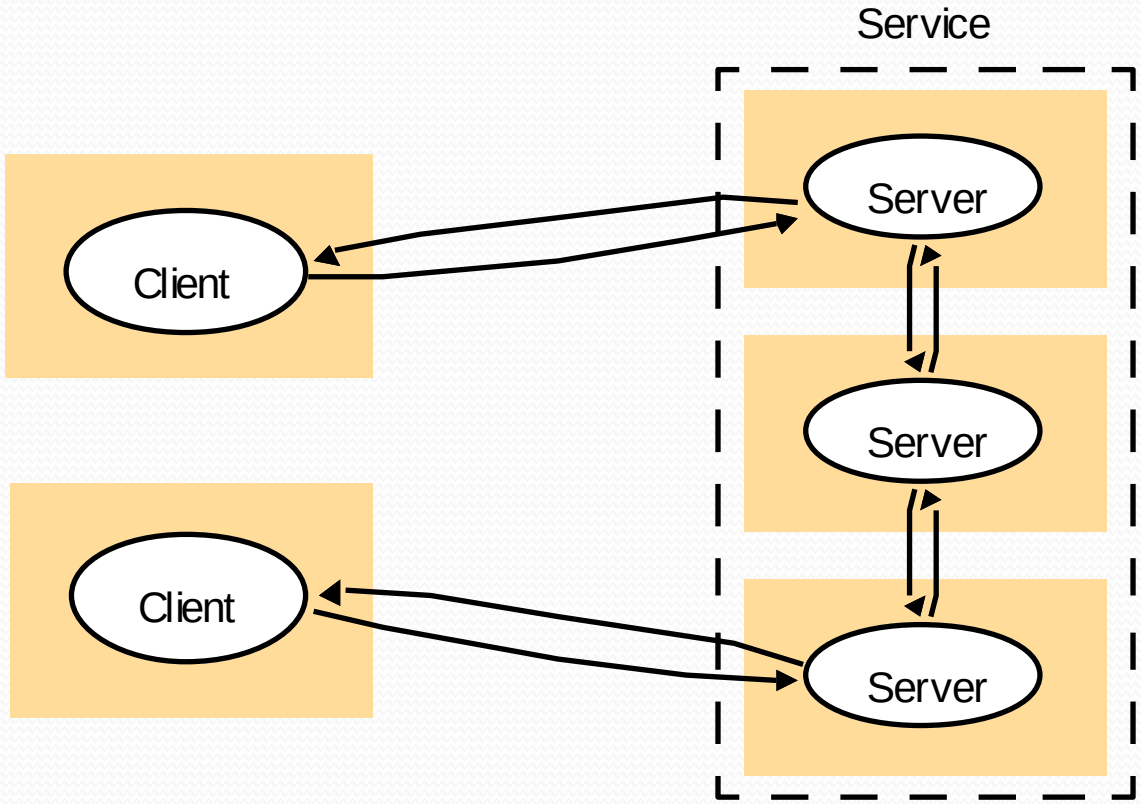
Sistemas Distribuídos: Modelos de Arquitetutras

Clientes invocando servidores individuais



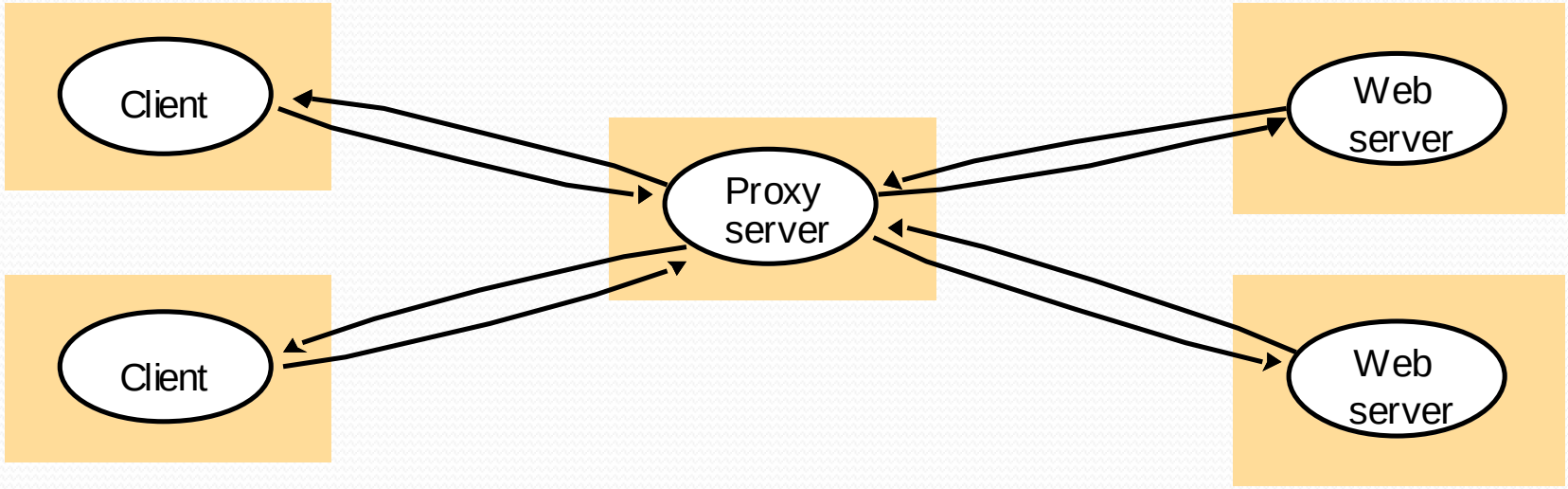
Sistemas Distribuídos: Modelos de Arquitetutras

Um serviço provido por múltiplos servers



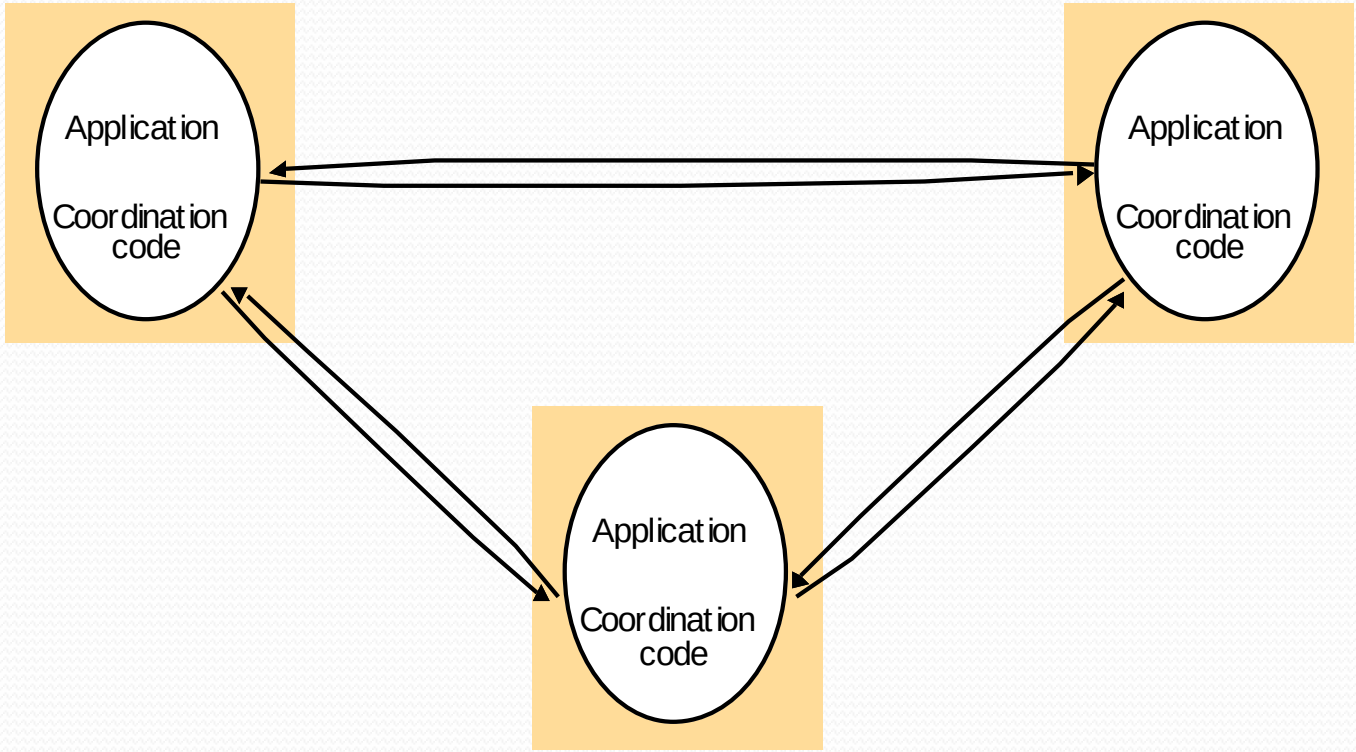
Sistemas Distribuídos: Modelos de Arquitetutras

Web proxy server



Sistemas Distribuídos: Modelos de Arquitetutras

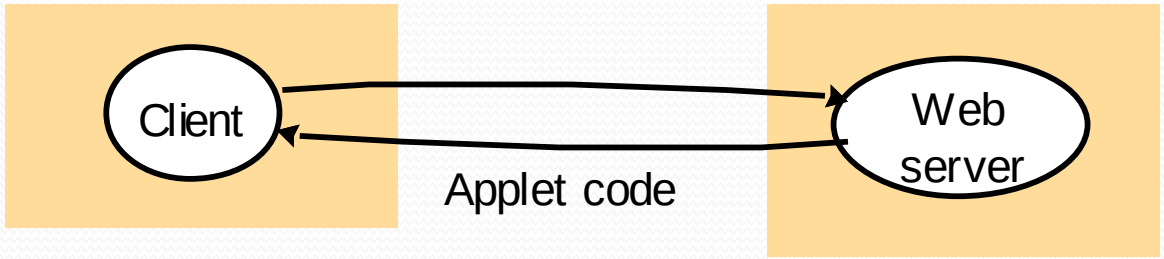
Uma aplicação distribuída baseada em processos fim (peer processes)



Sistemas Distribuídos: Modelos de Arquitetutras

Web applets

a) client request results in the downloading of applet code



b) client interacts with the applet

Sistemas Distribuídos: Modelos de Arquitetutras

Thin clients e servidores

Conceito de Sistemas Distribuídos:

São sistemas compostos por diversas partes cooperantes que são executadas em máquinas diferentes interconectadas por uma rede

Cooperação → Máquinas trabalham em conjunto para chegar a um determinado objetivo; para cooperar as máquinas precisam se comunicar de alguma forma

Interconexão → Máquinas dispõem de um meio de comunicação para trocar dados com o intuito de cooperar de modo a alcançar um objetivo comum

Computação Distribuída:

- Consiste em executar processos / aplicações cooperantes em máquinas diferentes
- Tornou-se bastante comum a partir da popularização das redes de computadores

- Processos são executados em máquinas diferentes interligadas por uma rede
  - Redes locais
  - Internet
  - Outras redes públicas ou privadas

Sistemas distribuídos x centralizados

- Sistemas centralizados têm um ponto único de falha;
- Sistemas distribuídos podem *aumentar a robustez* do sistema e a disponibilidade dos dados;
- Sistemas centralizados são limitados pela capacidade de processamento e armazenamento da máquina;

Sistemas distribuídos x centralizados

- Sistemas distribuídos podem crescer em escala adicionando novos servidores ao sistema;
- Sistemas distribuídos são mais sujeitos a apresentar falhas parciais de funcionamento e de segurança;
- Sistemas distribuídos são mais difíceis de administrar pois os servidores estão em locais diferentes;

Sistemas distribuídos x paralelos

- Ambos podem executar processamento paralelo usando os vários processadores disponíveis;
- Em máquinas paralelas os processadores podem trocar dados usando discos ou memória;
- Em sistemas distribuídos dados são trocados pela rede, o que pode prejudicar o seu desempenho;

Sistemas distribuídos x paralelos

- Sistemas paralelos são mais vulneráveis a alguns tipos de falha (ex.: falha na alimentação ou na rede);
- Sistemas paralelos não são tão escaláveis quanto sistemas distribuídos.

Aplicações

- Outros exemplos de aplicações distribuídas:
  - Messenger, ICQ;
  - Redes Peer-to-Peer (P2P);
  - Sistema de Reserva de Passagens;
  - Sistema de Operadoras de Cartão de Crédito;

Aplicações

- Sistemas bancários;
- Sistemas de gerenciamento de redes de telecomunicações, transmissão de energia;
- Sistemas de informação de grandes empresas;

- Nova abordagem de TV digital.

Características

- Classificação:
  - Sistemas Homogêneos
    - Todos os *sites* têm o mesmo software e hardware
  - Sistemas Heterogêneos
    - Hardware, sistema operacional e aplicativos podem ser diferentes em cada máquina
    - Podem representar os dados de modo diferente
    - Protocolos de comunicação precisam ser os mesmos para que as máquinas possam interagir

Características

- Vantagens
  - Dispõem de maior poder de processamento;
  - Se bem construídos, podem apresentar um melhor desempenho que sistemas centralizados;
  - Podem apresentar maior confiabilidade ;

Características

- Vantagens
  - Permitem reutilizar serviços já disponíveis;
  - Possuem maior capacidade para armazenamento de dados que sistemas centralizados;
  - Podem compartilhar dados e recursos;
  - Podem atender um maior número de usuários;

Características

- Dificuldades
  - Desenvolver, gerenciar e manter o sistema;
  - Controlar o acesso concorrente a dados e a recursos compartilhados;
  - Controlar a consistência dos dados replicados;

Características

- Dificuldades
  - Evitar que falhas de máquinas ou da rede comprometam o funcionamento do sistema
    - Mais hardware → mais falhas
  - Garantir a segurança do sistema e o sigilo dos dados trocados entre máquinas;
  - Lidar com a heterogeneidade do ambiente;

Características

- Desempenho
  - Mais máquinas → maior poder de processamento;
  - Uso da rede → pior desempenho;
  - É importante utilizar adequadamente o poder de processamento disponível e limitar a comunicação da rede para que o desempenho da aplicação distribuída seja satisfatório;

Características

- Compartilhamento de Dados e de Recursos
  - É possível acessar dados e recursos disponíveis em outras máquinas do sistema
    Ex.: impressoras, câmeras, bancos de dados, filmes.
  - Recursos podem estar indisponíveis devido a falhas;
  - Acesso concorrente deve ser controlado;
  - Deadlocks precisam ser evitados;

Características

- Uso da Rede
  - Acoplamento fraco → máquinas trocam dados através da rede
  - Tempo de comunicação ilimitado → pode comprometer o funcionamento do sistema
  - Como reduzir o tráfego na rede?

Características

- Não-deterministas
  - Comportamento pouco previsível devido ao uso da rede, que leva à possibilidade de ocorrerem retardos inesperados ou mesmo falhas na comunicação
  - Como evitar congestionamento/sobrecarga?

Características

- Influência do Tempo
  - Sistemas distribuídos são bastante influenciados pelo tempo de comunicação pela rede
  - Em geral não há uma referência de tempo global, pois os relógios das máquinas não estão sincronizados
    - Como ordenar os eventos no sistema?

Características

- Controle
  - Os sistemas distribuídos tendem a empregar recursos remotos, sobre os quais não se tem total controle
  - Estes recursos podem estar indisponíveis por já estarem sendo usados ou por falha na máquina ou programa que os administra, na rede ou no próprio recurso

Características

- Sujeito a Falhas
  - As máquinas e a rede podem falhar
  - Como evitar que estas falhas comprometam o funcionamento do sistema?
- Segurança
  - Mais difícil controlar o acesso a dados e recursos em sistemas distribuídos
  - Como garantir a segurança do sistema e o sigilo dos dados trocados pela rede?

Características

- Difícil ter uma visão global do sistema
  - Difícil saber o estado de cada parte do sistema em um determinado instante devido à latência da rede
  - Transações envolvem várias máquinas
  - Como conhecer o estado global do sistema?

Características

- Gerenciamento e manutenção do sistema
  - Máquinas dispersas pela rede → difícil gerenciar e manter as máquinas e o sistema funcionando
  - Há maior dificuldade se o sistema for heterogêneo
  - Administração pode ser independente

Características

- Replicação de Dados
  - Os dados do sistema podem estar replicados;
  - É necessário garantir a consistência de réplicas → réplicas com o mesmo valor!
  - Alterações nos dados devem ser propagadas para as máquinas com réplicas do dado;

Características

- Replicação de Dados
  - Replicação dificulta a obtenção do estado global do sistema;
  - Cada réplica deve atualizar seu estado ao reintegrar-se ao sistema; qualquer inconsistência nos dados deve ser resolvida.

Desenvolvimento de Aplicações

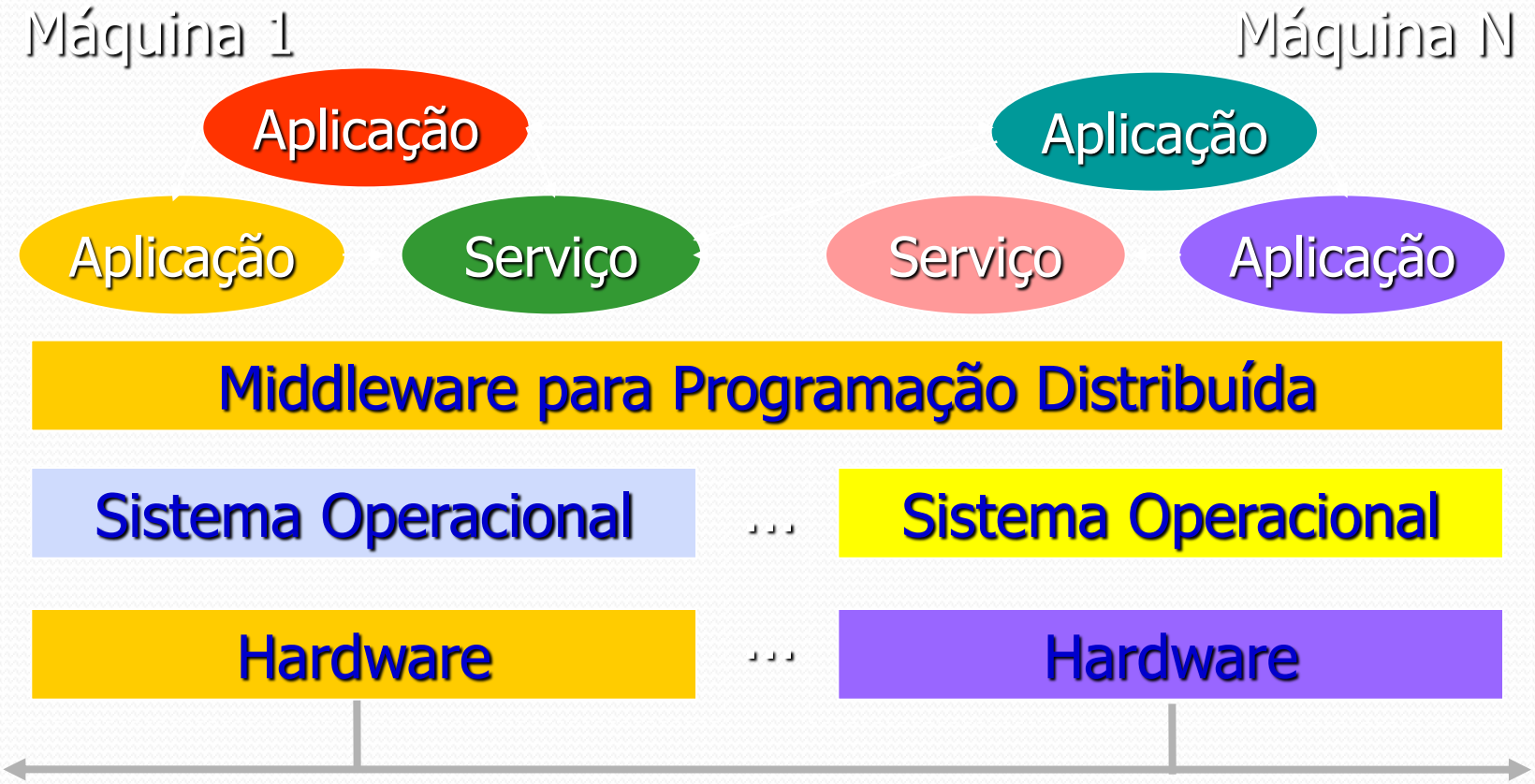
- Aplicações Distribuídas são criadas usando:
  - APIs e Bibliotecas: fornecem rotinas para comunicação entre processos, controle de concorrência, monitorar transações, etc.
    Ex.: Sockets, WinSock, java.net, etc.

Desenvolvimento de Aplicações

- Middleware para Programação Distribuída: fornece suporte para criar / executar programas distribuídos.
  Ex.: CORBA, COM, .NET, etc.
- Servidores de Aplicação: hospedam aplicações que provêem serviços e dados a clientes remotos
  Ex.: JBoss, WebSphere, etc.
- Linguagens e sistemas operacionais distribuídos vem caindo em desuso

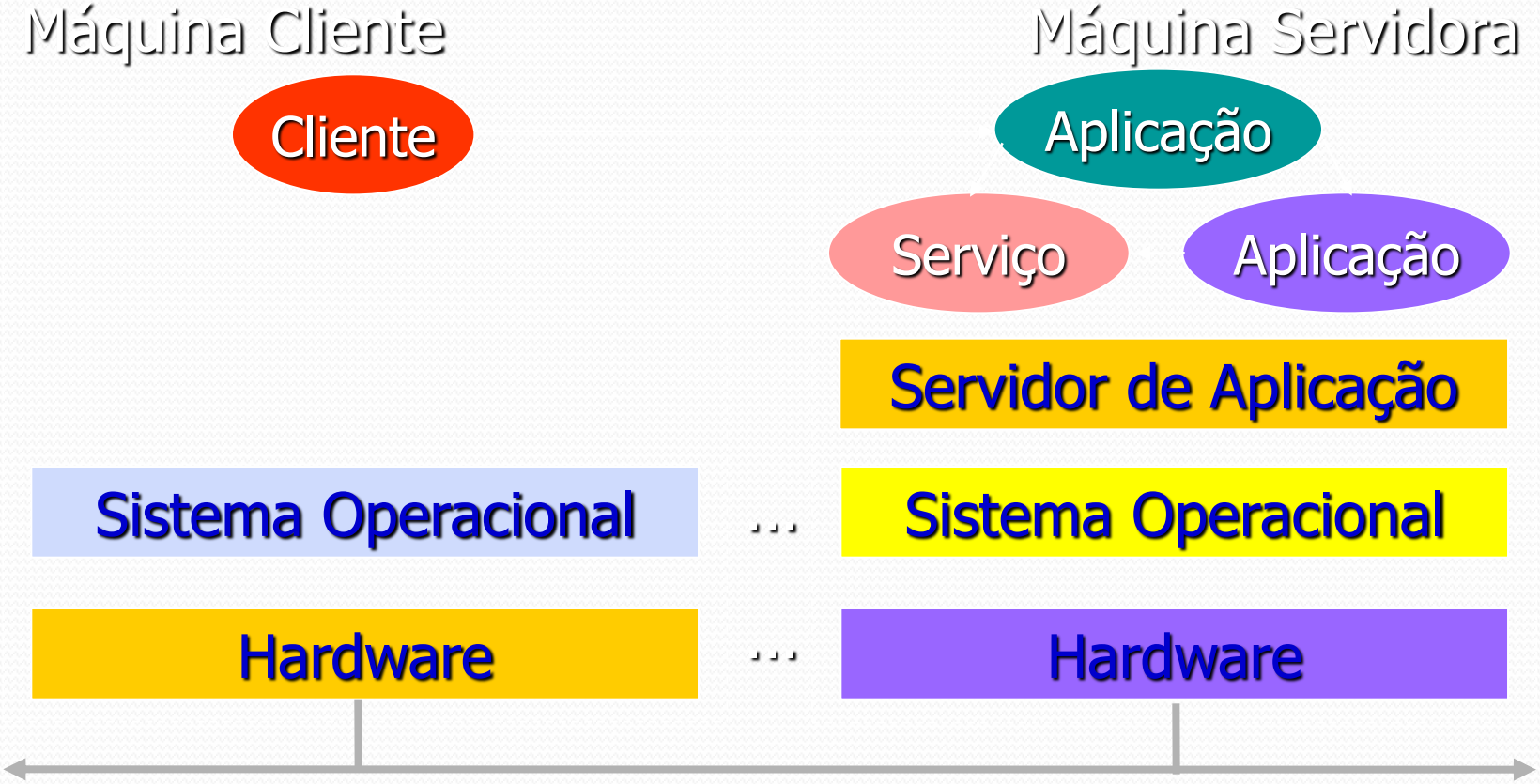
Desenvolvimento de Aplicações

- **Middleware para Programação Distribuída**



Desenvolvimento de Aplicações

- **Servidores de Aplicação**



Desenvolvimento de Aplicações

- Projeto de Aplicações Distribuídas
  - Abordagens de projeto
    - *Top-Down*: usada em sistemas construídos do zero, possivelmente homogêneos
    - *Bottom-Up*: usada quando os sistemas locais já estão instalados e precisam ser integrados

Desenvolvimento de Aplicações

- Questões a serem respondidas no projeto
  - Onde colocar os *sites*?
  - Como distribuir dados e serviços pelos *sites*?
  - Replicação é necessária?
  - Como integrar sistemas já existentes?

Introdução a computação distribuída

O que vem a ser computação ubíqua ?

Introdução a computação distribuída

O que vem a ser computação ubíqua ?

O termo computação ubíqua foi primeiramente sugerido por Mark Weiser em 1988. Este descreve sua idéia de tornar os computadores *onipresentes e invisíveis.*

O esforço é tirar o computador do caminho entre o usuário e seu trabalho. Em outras palavras, o objetivo é ir além da *interface amigável* e longe da *realidade virtual*.

Introdução a computação distribuída

O que vem a ser computação ubíqua ?

Ao invés de usar ao máximo todas os canais de entrada e saída do corpo, semelhante à *realidade virtual*, a idéia é permitir que o indivíduo faça seu trabalho com o auxílio de computadores sem nunca ter que se preocupar em trabalhar nos computadores.

Melhorando-se as interfaces fazem do obstáculo (o computador) um elemento mais fácil de usar.

Introdução a computação distribuída

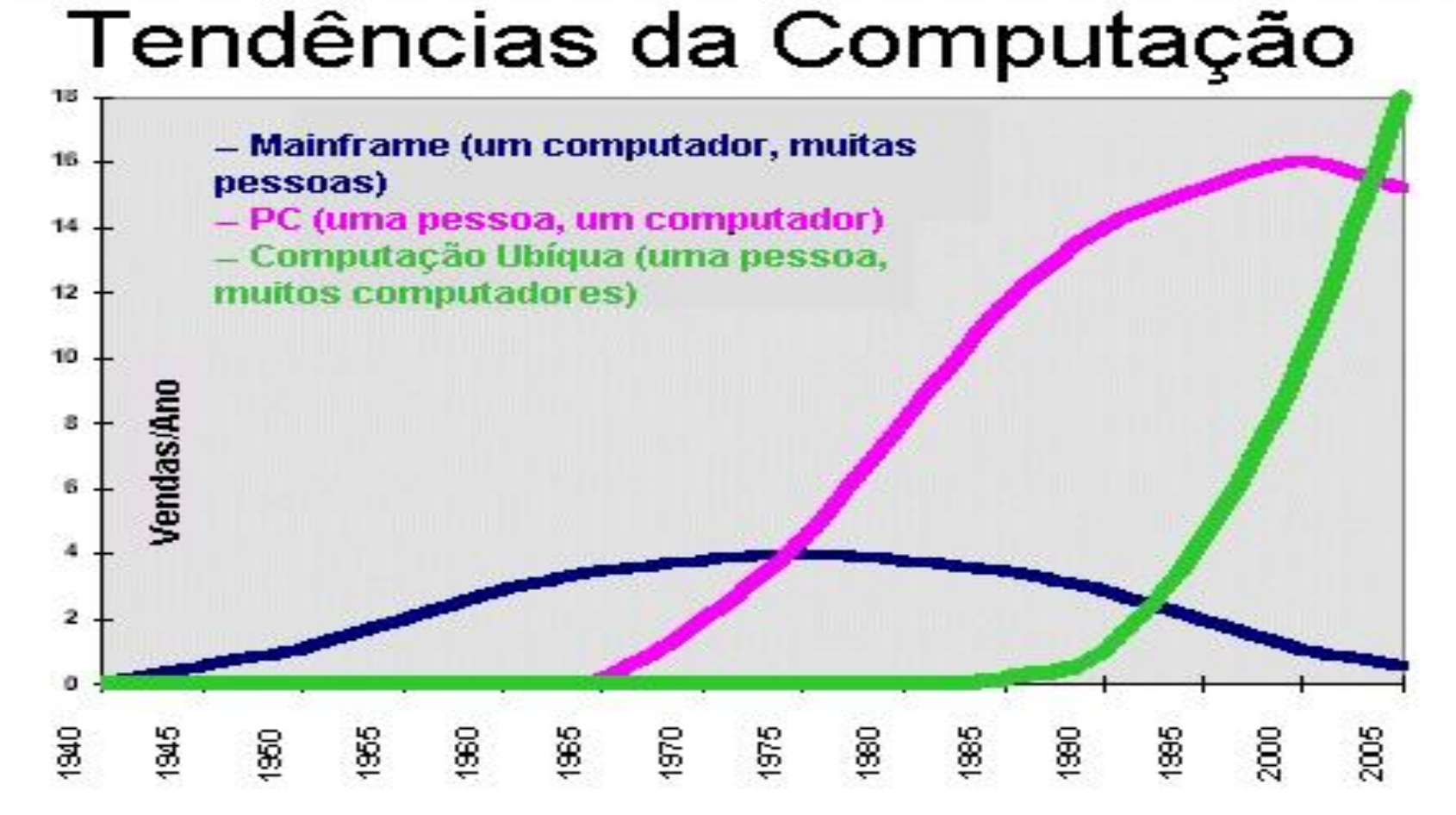
O que vem a ser computação ubíqua ?

Este tipo de computação cria a *era da tecnologia calma* (*calm technology*).

A computação passa a ser subjacente às nossas vidas. Os computadores passam a ser tão naturais, tão sob medida e tão *embutidos* em todos os locais, que eles se tornam praticamente invisíveis, isto é, nós os utilizamos quase sem pensar.

Introdução a computação distribuída

O que vem a ser computação ubíqua? Fonte: [Mark Weiser]



Introdução a computação distribuída

O que vem a ser computação pervasiva ?

Mark Weiser e seus colegas no Lab. PARC, utilizaram os termos *computação ubíqua e pervasiva de maneira intercambiável para descrever como a computação iria migrar do ambiente desktop, para um ambiente pessoal e depois para uma configuração mais distribuída – móvel e embarcada*.

Introdução a computação distribuída

O que vem a ser computação pervasiva ?

Todavia, existe uma diferença nestes tipos de computação.

O termo *computação ubíqua significa o estado de estar presente em todos os lugares.*

Por outro lado, computação pervasiva representa passar através, ou ser difundida através de.

Introdução a computação distribuída

O que vem a ser computação pervasiva ?

No mundo real, a computação ubíqua está relacionada aos frameworks, sistemas embarcados, redes e *displays que estão transparentes e em qualquer lugar, permitindo aos usuários operações de plug-and-play em dispositivos e ferramentas.*

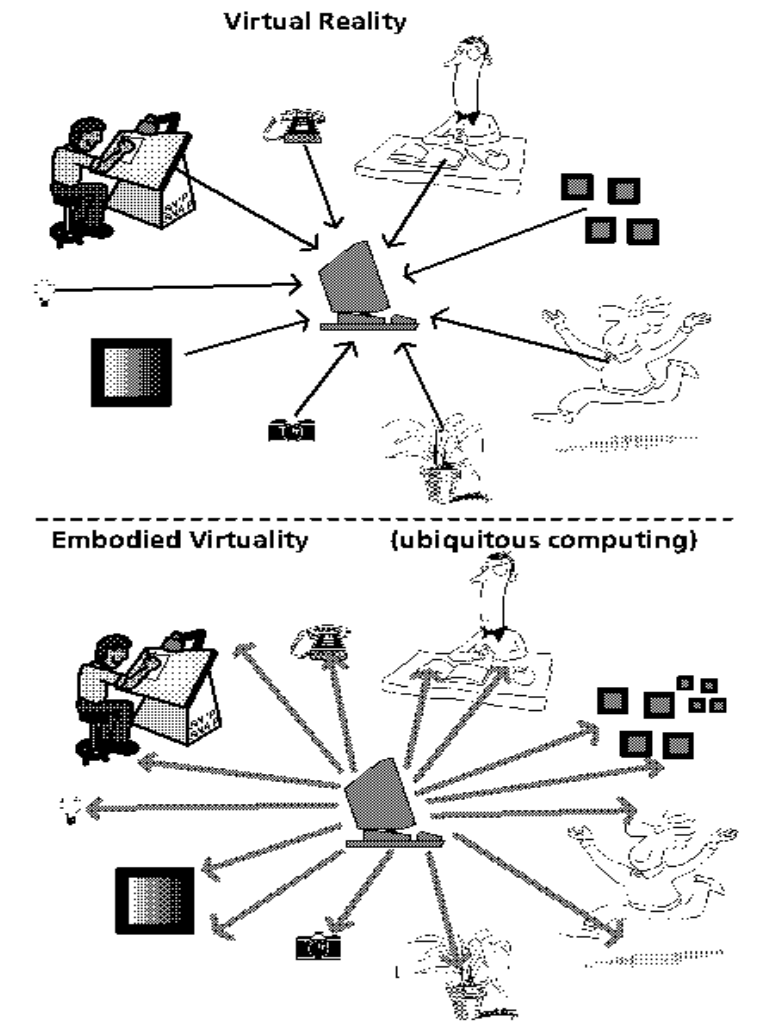
Introdução a computação distribuída

O que vem a ser computação pervasiva ?

Por outro lado, a *computação pervasiva* refere-se a dispositivos físicos tais como seu telefone celular, computador de mão, uma jaqueta inteligente que se modifica com a temperatura do corpo.

Assim, podemos entender a computação pervasiva como um conjunto de ferramentas dentro de um determinado ambiente, no qual podemos acessar as informações a qualquer hora e a qualquer momento.

Introdução a computação distribuída



Fonte:[Mark Weiser]

Tolerância à Faltas

- ***Conceitos***
- ***Tipos de Faltas***
- ***Replicação***
- ***Detecção de Falhas***
- ***Recuperação de Falhas***

Conceitos

- Confiança de Funcionamento (*Dependability*)
  - Representa a confiança depositada em um determinado sistema em relação ao seu correto funcionamento
  - Um sistema dito confiável (*dependable*) possui uma grande probabilidade de se comportar da maneira esperada

Conceitos

- Confiança de Funcionamento (*Dependability*)
  - A confiança de funcionamento é um importante requisito de Qualidade de Serviço (QoS) em sistemas computacionais críticos

Conceitos

- Estados de um Sistema:
  - Sistema próprio: serviço é fornecido pelo sistema como foi especificado
  - Sistema interrompido ou impróprio: serviço não é fornecido conforme especificado

Conceitos

- A Confiança de Funcionamento de um sistema é medida pelos seguintes fatores:
  - Confiabilidade (*Reliability*): tempo de funcio-namento contínuo (sem falhas) do sistema
  - Manutenibilidade (*Maintainability*): tempo gasto para restaurar o sistema após uma falha

Conceitos

- A Confiança de Funcionamento de um sistema é medida pelos seguintes fatores:
  - Disponibilidade (*Availability*): tempo de funcionamento em relação ao tempo de falha
  - Seguridade (*Safety*): prejuízo causado pela falha do sistema

Conceitos

- Confiabilidade pode ser representada por:
  - Tempo Médio Para a Falha (MTTF): indica o tempo médio que o sistema fica sem falhar
    - Ex.: X horas ou dias de funcionamento
    - Quanto maior, melhor
  - Médio Entre Falhas (MTBF): representa o tempo entre falhas sucessivas
    - Ex.: Y horas ou dias entre falhas
    - Quanto maior, melhor

Conceitos

- Confiabilidade pode ser representada por:
  - Probabilidade (taxa) de Falha
    - Ex.: 10^{-Z} falhas/hora ou falhas/dia
    - Quanto maior o expoente, melhor

Conceitos

- Manutenibilidade é representada por:
  - Tempo Médio Para Reparo (MTTR): indica o tempo necessário para que o sistema volte a funcionar corretamente
    - Ex.: X segundos, minutos ou horas para o sistema voltar a funcionar corretamente
    - Quanto menor, melhor

Conceitos

- Seguridade é representada por:
  - Grau de Seguridade: probabilidade do sistema ser recuperável (de não se tornar impróprio) em caso de falha, ou seja, a chance de uma falha não ser catastrófica

$$\frac{\text{Falhas Benignas}}{\text{Falhas Benignas + Catastróficas}}$$

    - Ex.: sistema recuperável em 98% das falhas
    - Quanto maior, melhor

Conceitos

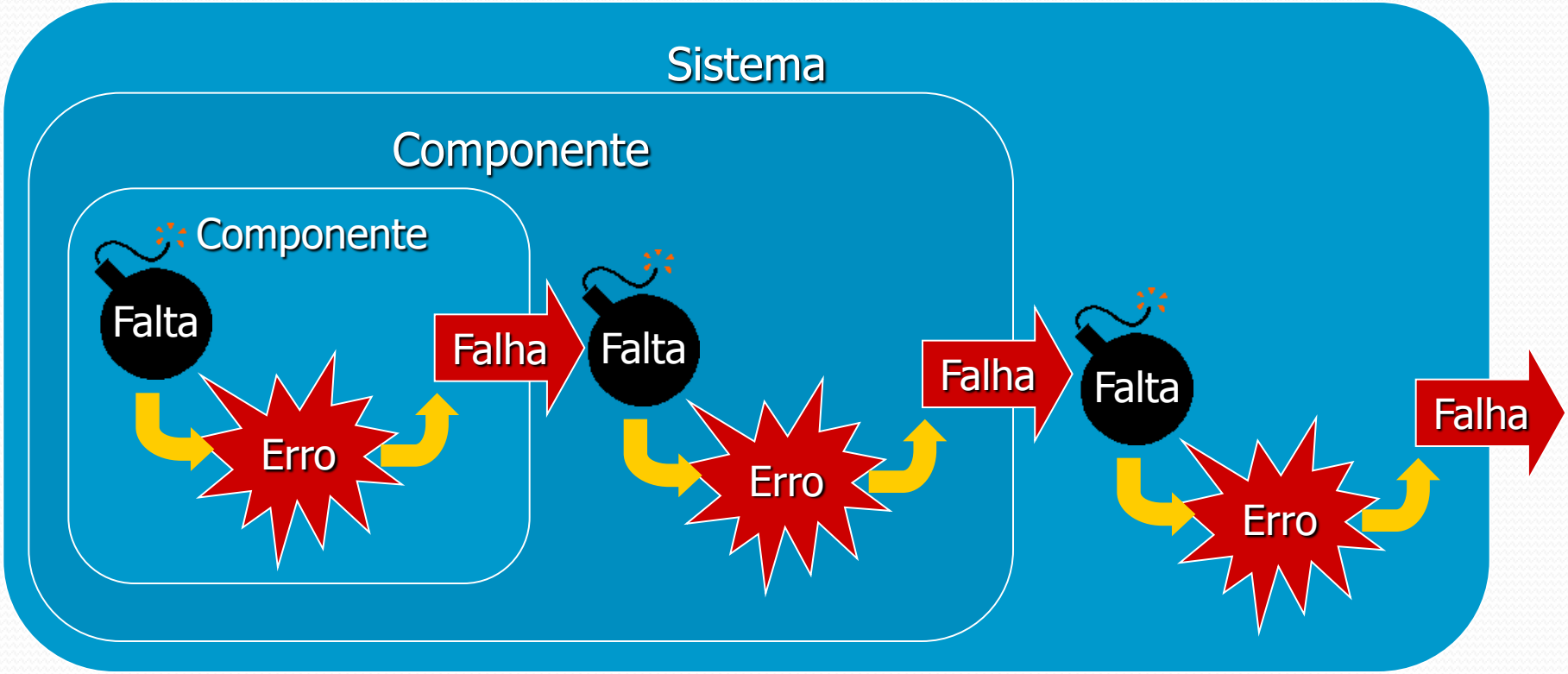
- Falta, Erro e Falha
  - Faltas são defeitos que ocorrem em sistemas
    - Originadas por fatores internos ou externos
    - Podem ficar dormentes até serem notadas
  - Erros são resultantes de faltas no sistema
    - Ocorrem quando faltas impedem o funcionamento normal do sistema
    - Diferentes faltas podem causar o mesmo erro
  - Falhas podem ocorrer devido a erros no sistema
    - A falha é o efeito observável do erro
    - Diversos erros podem levar à mesma falha

Conceito

- Falta, Erro e Falha
  - Exemplo: HD
    - Um setor do disco pode estar com defeito (falta)
    - Um erro de leitura pode ocorrer se um programa tentar ler ou escrever neste setor
    - Pode ocorrer uma falha em um sistema que tente acessar este setor e não consiga
    - Se os dados gravados neste setor estiverem replicados em outro local, o sistema pode tolerar a falta e não apresentar falha

Conceitos

- Falhas em Cascata



Conceitos

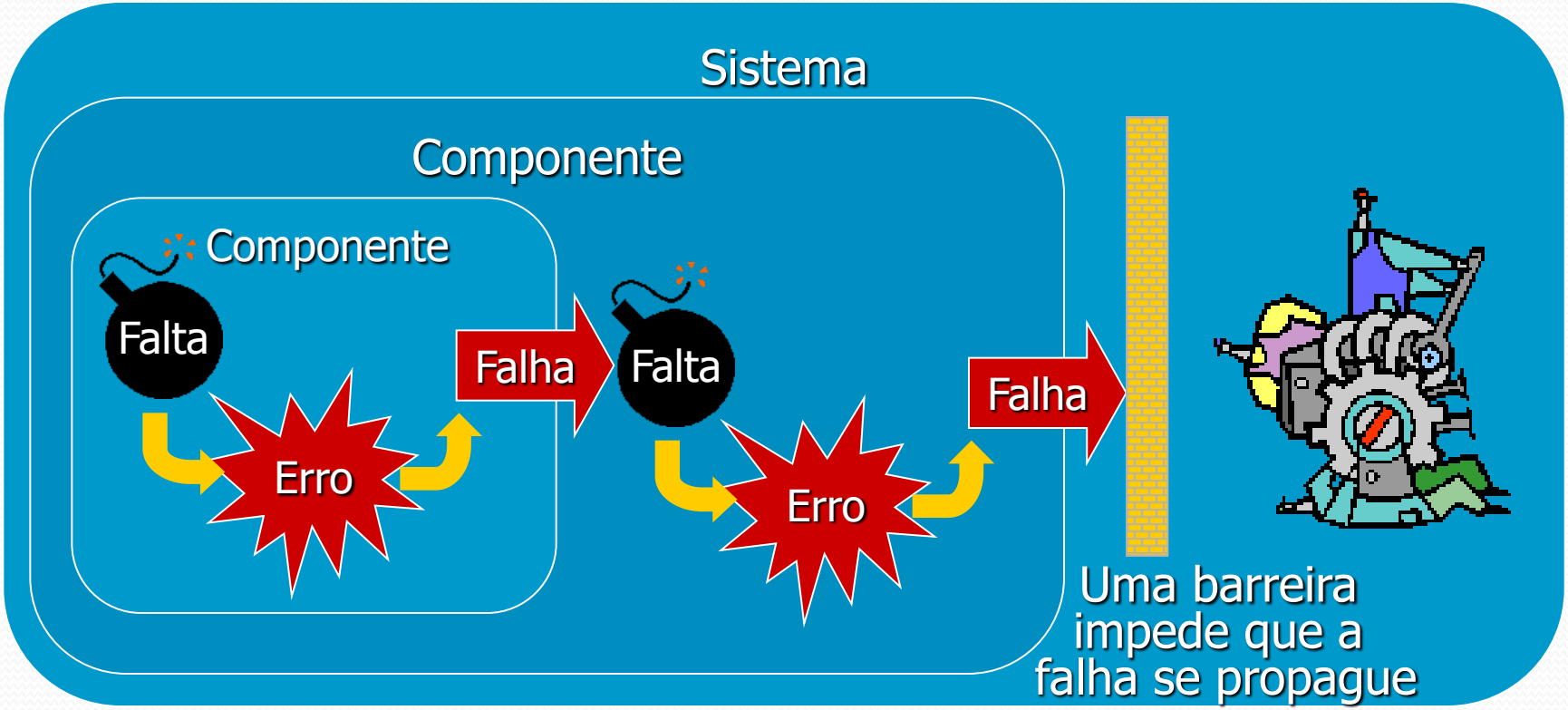
- Falhas em Cascata
  - A falha de um componente pode ocasionar a falha de outro que necessita dos serviços do primeiro, e assim sucessivamente, podendo levar o sistema como um todo a falhar
    - Exemplo:
      - Uma falta no disco pode causar uma falha no sistema de arquivos
      - Os servidores Web e de e-mail, que usam o sist. de arquivos, podem falhar
      - Uma aplicação de comércio eletrônico baseada na Web pode também falhar

Conceitos

- Previsão de Faltas
  - Estima a probabilidade de que faltas ocorram
  - Permite que se avalie os riscos de falha
- Remoção de Faltas
  - Consiste em detectar e remover as faltas antes que causem erros e falhas
  - Usar ferramentas como debugger, scandisk, …
- Prevenção de Faltas
  - Elimina as condições que fazem com que faltas ocorram durante a operação do sistema
  - Usa replicação interna, técnicas de validação, …

Conceitos

- Tolerância a Faltas



Conceitos

- Tolerância a Faltas
  - Propriedade de sistemas que não falham necessariamente ao se deparar com uma falta
- Sistemas Tolerantes a Faltas
  - São sistemas capazes de tolerar faltas encontradas durante a sua execução
- Técnicas de Tolerância a Faltas
  - Permitem prevenir falhas contornando as faltas que os sistemas podem vir a apresentar

Tipos de Faltas

- Classificação em relação à sua origem:
  - Física: causada pelo hardware
  - De projeto: introduzida durante a fase de projeto do sistema
  - De interação: ocorrida nas interfaces entre componentes do sistema ou na interação com o mundo exterior
- Classificação em relação à sua natureza:
  - Acidental ou Intencional
  - Maliciosa ou Não

Tipos de Faltas

- Classificação em relação ao seu surgimento:
  - Na fase de desenvolvimento do sistema
  - Na fase de operação do sistema
- Classificação em relação à sua localização:
  - Interna
  - Externa
- Classificação em relação à persistência:
  - Temporária
    - Transiente ou Intermitente
  - Permanente

Tipos de Faltas

- Classificação com base no modelo de faltas:
  - Faltas Omissivas
    - Crash: deixa de funcionar permanentemente
    - Omissão: sistema deixou de fazer o que deveria em um determinado instante
    - Temporal: sistema atrasou-se para executar uma determinada ação
  - Faltas Assertivas
    - Sintática: formato da saída é inadequado
    - Semântica: saída apresenta valor incorreto
  - Faltas Arbitrárias: omissivas + assertivas

Tipos de Faltas

- Os tipos de faltas mais freqüentes são:
  - Faltas de operação e administração: 42%
  - Faltas de software: 25%
  - Faltas de hardware: 18%
  - Faltas de ambiente: 14%
  - Fonte: Jim Gray. *Why do Computers Stop and What Can Be Done About It?* IEEE SRDS'85.
  - Estudos mais recentes confirmam estes dados

Replicação

- Tolerância a faltas pode ser obtida através do uso de recursos redundantes
- Redundância pode ser aplicada das seguintes maneiras:
  - Redundância Temporal: repetir uma mesma tarefa até que um resultado válido seja obtido
  - Redundância de Valores: replicar um dado armazenado ou enviado pela rede
  - Redundância Espacial: usar várias réplicas de um componente de hardware ou software

Replicação

- Uso de réplicas aumenta a disponibilidade
  - Exemplo: se a probabilidade de perda de uma mensagem na rede é de 2% (disponibilidade de 0,98), se duplicarmos todas as mensagens, a chance de se perder as duas cópias será de 0,04% (disponibilidade de $1-0{,}02^2=0{,}9996$)
  - Exemplo 2: se um servidor fica indisponível durante 8 horas a cada ano (disponibilidade de 0,999), se criarmos 3 réplicas teremos uma parada total de 3,15 segundos em um século (disponibilidade de $1-0{,}001^3= 0{,}999999999$)

Replicação

- O acesso a serviços ou dados replicados deve ser transparente para o usuário
  - Usuário deve acessar o dado ou serviço replicado da mesma forma que o faria se não houvesse replicação
  - Se for preciso manter a consistência dos dados replicados, este processo deve ser efetuado automaticamente pelas réplicas
  - Mesmo que mais de uma réplica responda a uma requisição, apenas uma resposta deve ser entregue ao usuário

Replicação

- Técnicas de Replicação
  - Definem como as réplicas se comportarão durante o funcionamento normal do sistema e sob a presença de faltas
- Principais Técnicas de Replicação
  - Replicação Passiva (Primário-Backup)
  - Replicação Ativa
  - Replicação Semi-Ativa (Líder-Seguidores)
  - Replicação Preguiçosa (*Lazy*)

Replicação

- Replicação Passiva (Primário-Backup)
  - São criados um ou mais backups de um componente (primário) com o objetivo de substituí-lo em caso de falha
  - Funcionamento com propagação de estado instantânea:
    - Primário recebe requisições, as executa, atualiza o estado dos backups e retorna o resultado ao cliente

Replicação

- Replicação Passiva (Primário-Backup)
  - Em caso de falha do primário, um backup será escolhido para assumir o seu lugar
  - O backup escolhido terá o mesmo estado do primário até a última requisição executada
  - Uma requisição em execução durante a falha pode ser recuperada pelo cliente reenviando a requisição ao novo primário
  - Atualização de estado a cada requisição causa uma sobrecarga considerável no primário

Replicação

- Replicação Passiva (Primário-Backup)
  - Funcionamento com *log* e *checkpoints* :
    - As requisições de clientes são enviadas ao primário e ao(s) backup(s)
      - Primário executa as requisições e responde aos clientes
      - Backup recebe as requisições mas não as executa – apenas as registra em um *log*
    - O estado do primário é transferido para o(s) backup(s) em instantes predeterminados – chamados de *checkpoints*

Replicação

- Replicação Passiva (Primário-Backup)
  - Funcionamento com *log* e *checkpoints* (cont.):
    - O backup limpa o *log* a cada *checkpoint*
    - Em caso de falha do primário, o backup escolhido para assumir o seu lugar terá o estado do primário no último *checkpoint*
    - Para chegar ao mesmo estado do primário no instante da falha, o backup escolhido executa as requisições registradas no *log*

Replicação

- Replicação Passiva (Primário-Backup)
  - Considerações:
    - O(s) backup(s) consomem muito pouco poder de processamento, pois não precisam processar as requisições
    - O primário tem a obrigação de salvar seu estado e enviar ao(s) backup(s), o que consome processamento e largura de banda
    - Transferência de estado pode ser incremental
    - Quanto maior o intervalo entre *checkpoints*, menor a sobrecarga no primário, e maior o tempo de recuperação de falhas

Replicação

- Replicação Ativa
  - Um grupo de réplicas de um componente recebe uma requisição de um cliente
  - Todas as réplicas processam a requisição concorrentemente e enviam as suas respostas ao cliente
  - Não é preciso sincronizar o estado das réplicas, pois todas executam os mesmos procedimentos

Replicação

- Replicação Ativa
  - O cliente precisa de apenas uma resposta
  - A resposta válida para o cliente pode ser:
    - A mais freqüente (votação)
    - A primeira recebida
    - Uma média
    - etc.
  - Com isso, a replicação ativa pode tolerar faltas de valor por meio de votação
    - Em 2N+1 réplicas podemos ter N respostas erradas sem ocasionar falha do sistema

Replicação

- Replicação Ativa
  - Considerações:
    - Alto custo para execução das réplicas ativas
    - As requisições devem ser entregues na mesma ordem para todas as réplicas → usar protocolo de difusão atômica
    - Ordenação de mensagens tem custo alto
    - Recuperação é mais rápida que na replicação passiva, pois caso uma réplica falhe, as demais continuam funcionando normalmente

Replicação

- Replicação Semi-Ativa (Líder-Seguidores)
  - Um componente (líder) possui uma ou mais réplicas (seguidores)
  - Cada requisição é enviada a todos, que as executam na ordem definida pelo líder
  - Apenas o líder responde ao cliente que efetuou a requisição
  - Não é preciso sincronizar o estado das réplicas, pois todas executam os mesmos procedimentos

Replicação

- Replicação Semi-Ativa (Líder-Seguidores)
  - Considerações:
    - Mesmo que as requisições dos clientes sejam entregues fora de ordem, todas as réplicas chegarão ao mesmo estado, já que a ordem de execução é arbitrada pelo líder
    - O tempo gasto para processamento nas réplicas seguidoras é grande, já que elas também têm que processar a chamada
    - Caso o líder falhe, um seguidor é escolhido para assumir o seu lugar

Replicação

- Replicação Preguiçosa (*Lazy*)
  - Operações que precisam de ordenação total (ou seja, que alteram o estado) são executadas como na replicação ativa
  - Operações que podem ser executadas em ordens diferentes em cada réplica podem ser enviadas para qualquer réplica, que deve:
    - Atender a requisição
    - Enviar a resposta ao cliente
    - Difundir em *background* a requisição para as outras réplicas

Replicação

- Replicação Preguiçosa (*Lazy*)
  - Considerações:
    - Reduz a sobrecarga na execução de algumas operações, já que nem sempre é necessário usar um protocolo de difusão atômica
    - Exige que a semântica das operações seja conhecida para diferenciar as operações que exigem ordenação total daquelas nas quais ordem causal é suficiente

Replicação

- Programação com Múltiplas Versões
  - Réplicas não precisam ser idênticas
  - São criadas diferentes implementações de um mesmo componente de software
    - Linguagem de programação, compilador e suporte de execução diferentes
    - Algoritmos diferentes
    - Times de desenvolvimento diferentes
  - Resultados são obtidos através de votação
  - Com múltiplas versões, reduz-se a chance de todas as réplicas falharem no mesmo instante

Detecção de Falhas

- A falha de um componente de um sistema pode levar todo o sistema a falhar
- Mesmo que o sistema consiga tolerar a falha do componente, este deve ser recuperado para restaurar a capacidade do sistema de tolerar faltas
  - Ex.: na replicação passiva, se o único backup existente assume o lugar do primário, é preciso criar um novo backup
- É preciso detectar as faltas sofridas pelos componentes para poder recuperá-los

Detecção de Falhas

- Detecção Local de Falhas
  - Podem ser usados diversos métodos:
    - Rotinas de auto-verificação (*self-check*)
    - Guardiões: verificam constantemente as saídas geradas por um componente
    - *Watchdogs* : componente deve constante-mente reiniciar um temporizador antes que ele se esgote, indicando uma falha
  - Problema: mesmo para um observador local, processos lentos podem parecer falhos

Detecção de Falhas

- Detecção Distribuída de Falhas
  - Um componente do sistema envia mensagens periodicamente aos seus pares e avisa que está vivo (*I am alive*) ou pergunta se eles estão vivos (*Are you alive* ?)
  - Se um componente não se manifestar por um determinado tempo, ele é suspeito de falha
  - Suspeitas infundadas podem ser causadas por atraso, particionamento ou falha da rede

Detecção de Falhas

- Diagnóstico do Sistema
  - Componentes faltosos podem reportar erroneamente o estado dos seus pares
  - Em um sistema com f componentes faltosos, cada componente deve ser testado por pelo menos f outros, e precisamos de $n \geq 2f+1$ elementos para detectar corretamente a falta
  - Para diagnosticar falhas de componentes do sistema, um elemento deve coletar e analisar os dados obtidos dos demais componentes

Detecção de Falhas

- Detector de Falhas
  - Serviço ou módulo que verifica a ocorrência de falhas em componentes do sistema
  - Implantado junto ao componente
  - Executa um algoritmo de detecção de falhas
  - Interage com detectores de outros componentes do sistema
  - O componente pode requisitar ao seu detector informações sobre o estado de outros componentes do sistema

Detecção de Falhas

- Tipos de Detectores de Falhas
  - Perfeitos
    - Determinam precisamente se um componente do sistema falhou ou não
    - Todos os componentes têm a mesma visão
  - Imperfeitos
    - Detectores determinam se um processo é suspeito de falha ou não
    - Diferentes componentes podem ter visões distintas de um mesmo componente

Detecção de Falhas

- Tipos de Detectores de Falhas
  - Detectores perfeitos são difíceis de obter, principalmente em sistemas distribuídos
  - Detectores quase-perfeitos podem ser obtidos usando *crash* controlado
    - Se um componente é suspeito de falha, ele é removido do sistema
    - O componente passa a ser ignorado por todos os demais componentes
    - Pode levar a descartar componentes que estão funcionando corretamente

Recuperação de Falhas

- Recuperação de Erros
  - Ao perceber um erro, o componente pode tentar recuperar-se automaticamente
  - Recuperação de erro por retrocesso (*backward error recovery*): componente volta a um estado anterior ao erro e continua ativo
    - Exemplos: reinicia a execução de um método, retransmite pacotes perdidos, etc.
    - Operações posteriores ao instante de retrocesso são perdidas, mas seu efeito pode ainda ser sentido no sistema, levando possivelmente a inconsistências

Recuperação de Falhas

- Recuperação de Erros (cont.)
  - Recuperação por avanço (*forward error recovery*): componente toma medidas que anulem ou aliviem o efeito do erro e continua a operar normalmente
    - Exemplo: descartar pacotes, substituir um valor inválido pelo valor válido anterior, etc.
    - Usada quando não há tempo para voltar para estado anterior e retomar execução, ou quando ações não podem ser desfeitas

Recuperação de Falhas

- Recuperação de Falhas
  - Se ocorrer a falha de um componente, um sistema tolerante a faltas deve mascará-la usando as réplicas disponíveis
  - Na replicação passiva, substituir o primário por um backup e criar um novo backup
  - Na replicação semi-ativa, substituir o líder por um de seus seguidores e criar um seguidor
  - Nas replicações ativa e *lazy*, criar uma nova réplica para manter a capacidade do sistema de tolerar faltas (ou falhas de componentes)

Conteúdo Programático

Modulo I

. **Introdução a computação distribuída**

. **Ambientes de middleware**

Modulo II

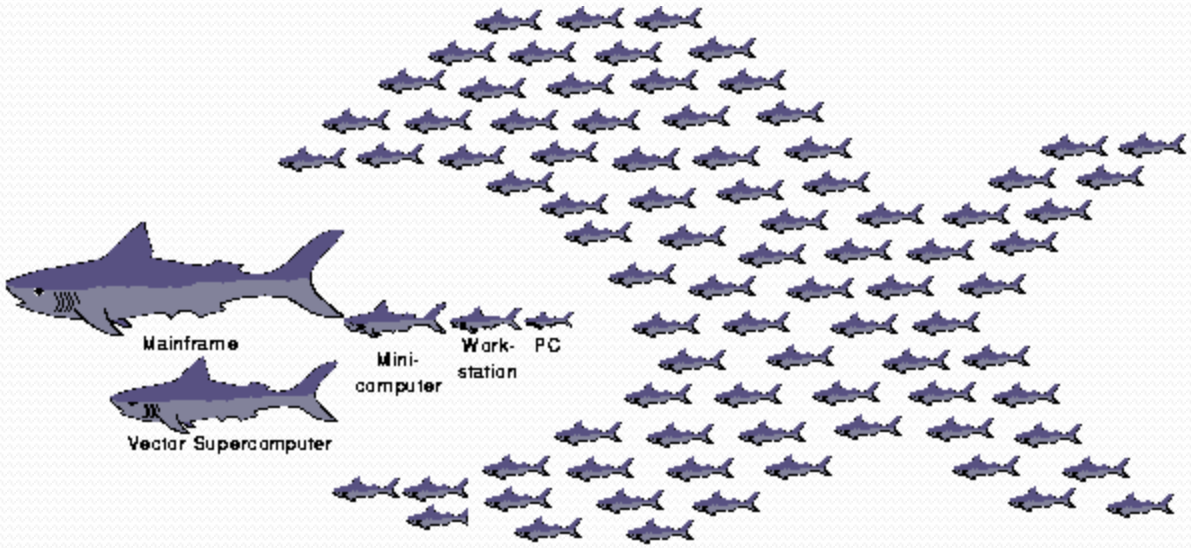
. Paradigma de troca de mensagens

. Parallel Virtual Machine (PVM)

. Message Passing Interface (MPI)

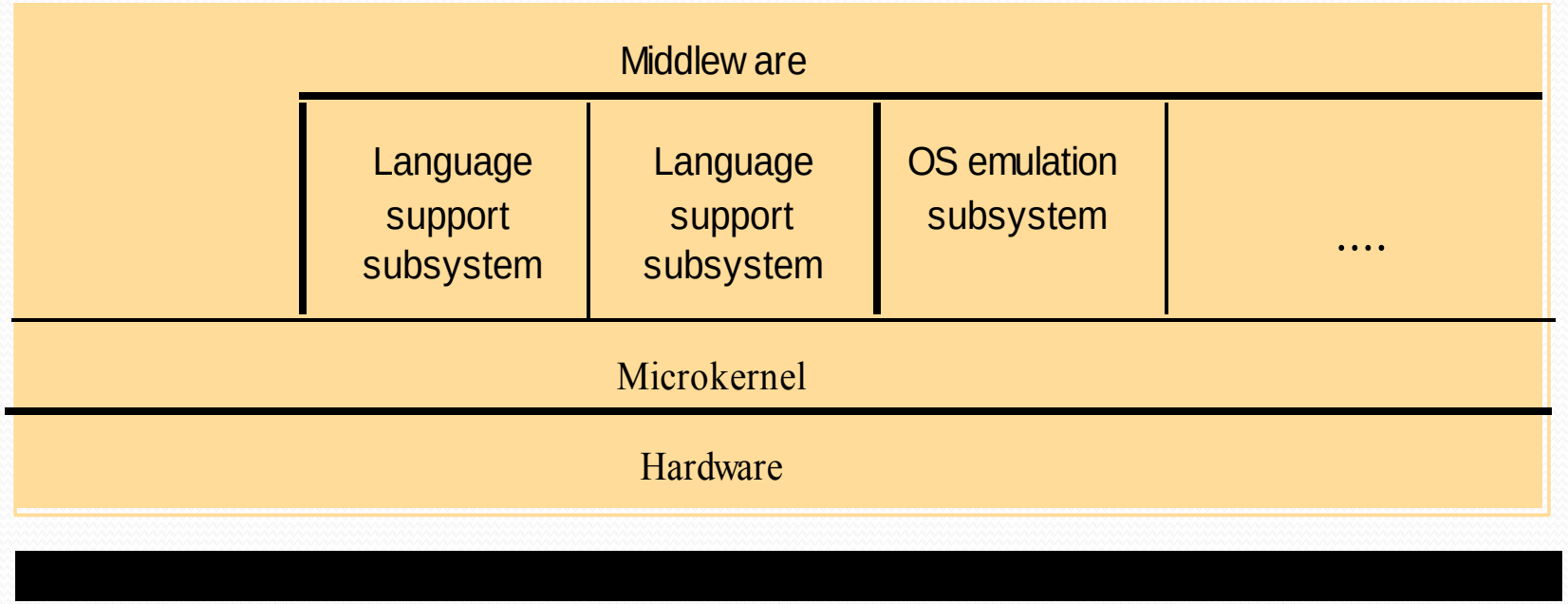
. Ambientes de Memória Compartilhada

1.2- Ambientes de Middleware



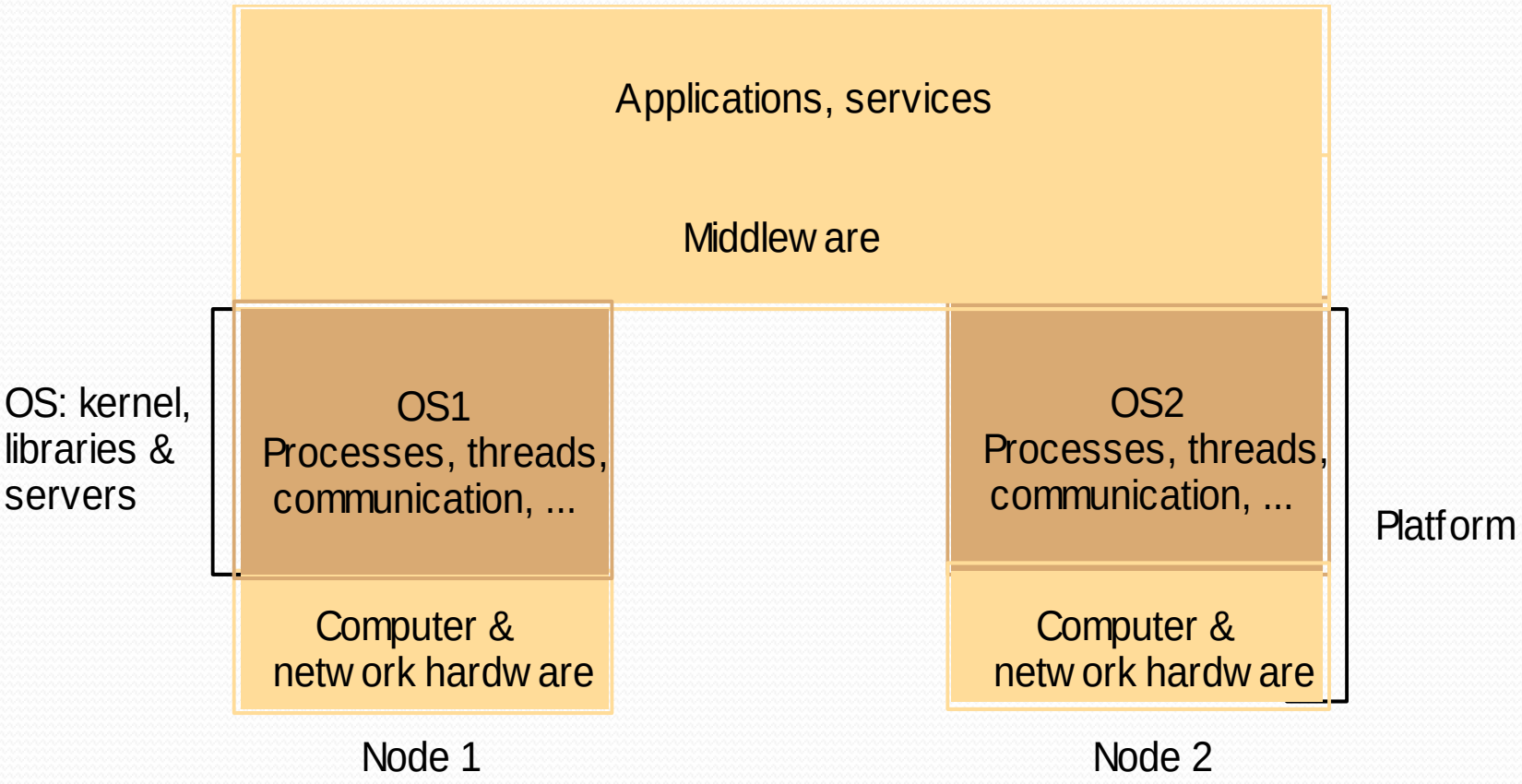
NOW

1.2- Ambientes de Middleware



Fonte: [Coulouris]

1.2- Ambientes de Middleware



Fonte: [Coulouris]

Integração com Sistemas Legados

- JDBC
- Conectores Java

Integração com Sistemas Legados

- Integração é necessária pois:
  - Empresas não substituem todos os seus sistemas de uma vez só;
  - Novos sistemas muitas vezes precisam interagir com sistemas legados;

Integração com Sistemas Legados

- Integração é necessária pois:
  - Sistemas de diferentes empresas precisam ser integrados quando estas passam a trabalhar juntas;
  - Processos de aquisição e junção de empresas exigem a integração de seus sistemas;

Integração com Sistemas Legados

- Ocorrem dificuldades ao integrar sistemas escritos em linguagens diferentes e rodando em plataformas diferentes

Integração com Sistemas Legados

- Tecnologias de Integração de Sistemas
  - CORBA: permite a integração de software desenvolvido em diferentes linguagens e plataformas;

Integração com Sistemas Legados

- Web Services: integram software utilizando padrões da Web, como XML e HTTP;

- ODBC (*Open DataBase Connectivity*): API usada por aplicações cliente para contactar servidores de banco de dados; o SGBD deve fornecer um driver ODBC; através das rotinas da API, o cliente envia comandos SQL ao servidor, que os executa e retorna resultado;

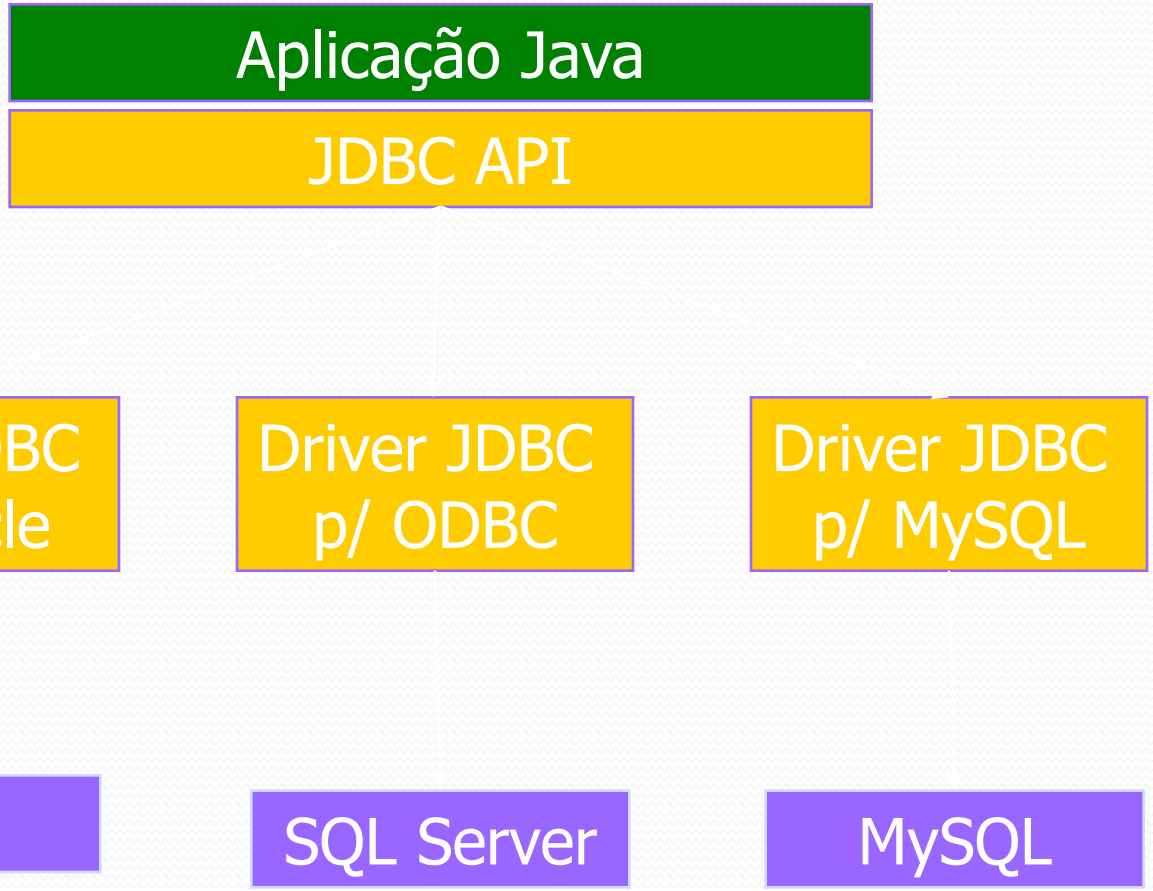
Integração com Sistemas Legados

- Integração de sistemas com a arquitetura Java
  - JDBC: interconexão com bancos de dados
  - Conectores Java: integração com outros sistemas

JDBC

- JDBC permite que aplicações Java acessem bases de dados
  - É necessário utilizar um driver
  - O driver deve ser fornecido pelo fabricante do SGBD
- Acessar uma base de dados implica em:
  - Estabelecer uma conexão com a base de dados
  - Executar comandos SQL
  - Processar os resultados obtidos

JDBC


Aplicação Java
JDBC API
Driver JDBC p/ Oracle
Driver JDBC p/ ODBC
Driver JDBC p/ MySQL
Oracle
SQL Server
MySQL

JDBC

- Drivers JDBC
  - JDBC-ODBC Bridge
    - É uma ponte do driver ODBC para JDBC
    - Implementado pela classe JdbcOdbcDriver
    - Permite que a aplicação Java veja o driver ODBC como um driver JDBC
  - JDBC – Native Bridge
    - Converte as chamadas JDBC da aplicação para chamadas de um driver nativo já instalado na máquina
    - O driver nativo se comunica com a base de dados através de um protocolo proprietário

JDBC

- Drivers JDBC (cont.)
  - JDBC – Net Bridge
    - Componente Java que converte chamadas JDBC para um protocolo de rede
    - A aplicação se comunica com um servidor intermediário que gerencia qualquer conexão solicitada à base de dados
  - All Java JDBC Driver
    - Escrito 100% em Java
    - Opera em qualquer plataforma com uma JVM

JDBC

- Para acessar a base de dados é preciso:
  - Carregar o driver JDBC:
    - Instanciar um objeto da classe java.sql.Driver:
      Class.forName(*driver_path*);
    - Forma alternativa:
      java –Djdbc.drivers=*driver_path application*

JDBC

- Instanciar um objeto da classe java.sql.Connection: Connection conn = DriverManager.getConnection(*URL*, *login*, *password*);
  - URL: localização da base de dados
  - Login: Nome do usuário da base de dados
  - Password: senha do usuário da base de dados

JDBC

- O envio de comandos SQL para a base de dados é feita através das classes:
  - java.sql.Statement
  - Java.sql.PreparedStatement //pré compilada

- Métodos das classes Statement
  - .execute() → executa um comando qualquer.
  - .executeQuery(String SQL) → executa consulta.
  - .executeUpdate(String SQL) → update, insert, delete

JDBC

- O resultados de um comando SQL é recebido em um ResultSet
  - É retornado pelo método executeQuery():
    Java.sql.ResultSet rs =stmt.executeQuery(SQL);
  - Os resultados são armazenados em uma lista

JDBC

- O método next() retorna um valor booleano indicando se há mais valores na lista
- Se houver um novo valor, este pode ser lido com um método rs.get*Tipo* (*id*), onde *Tipo* é o tipo correspondente ao dado lido (ex.: String, Int, etc.) e *id* pode ser a posição ou o nome do campo

Conectores

- Integração através de Conectores Java
  - Integram diversos sistemas à arquitetura Java
    - ERP
    - CRM
    - BI
    - etc.
  - Fornecidos pelo fabricante do sistema legado ou por terceiros
  - Para desenvolver um conector geralmente é necessário escrever código nativo para a plataforma do sistema legado e integrar ao Java usando JNI (Java Native Interface), CORBA IIOP ou sockets

Conteúdo Programático

Modulo II

. **Paradigma de troca de mensagens**

. Parallel Virtual Machine (PVM)

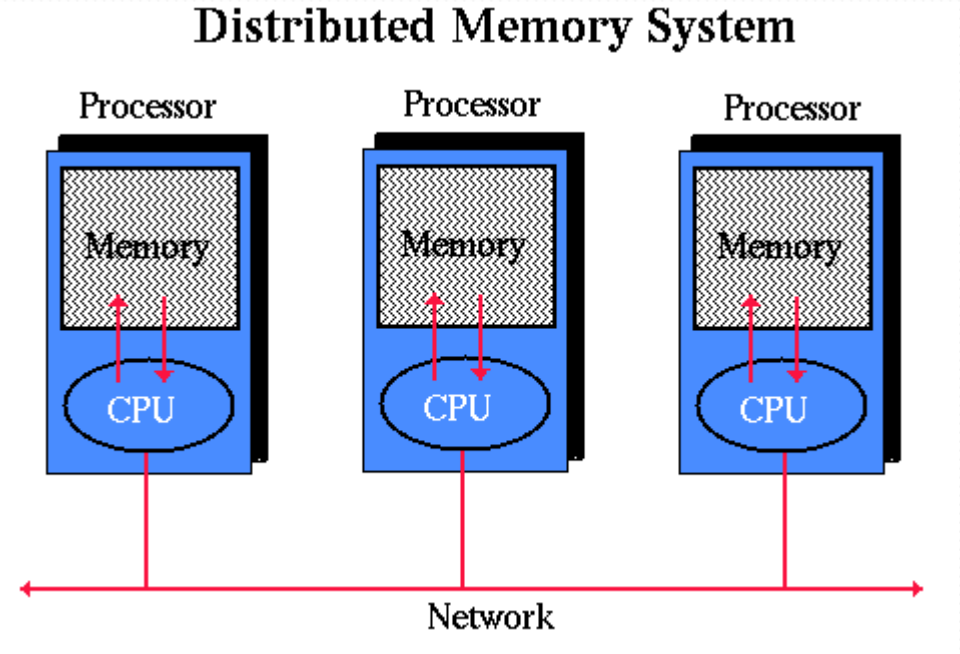
. Message Passing Interface (MPI)

. Ambientes de Memória Compartilhada

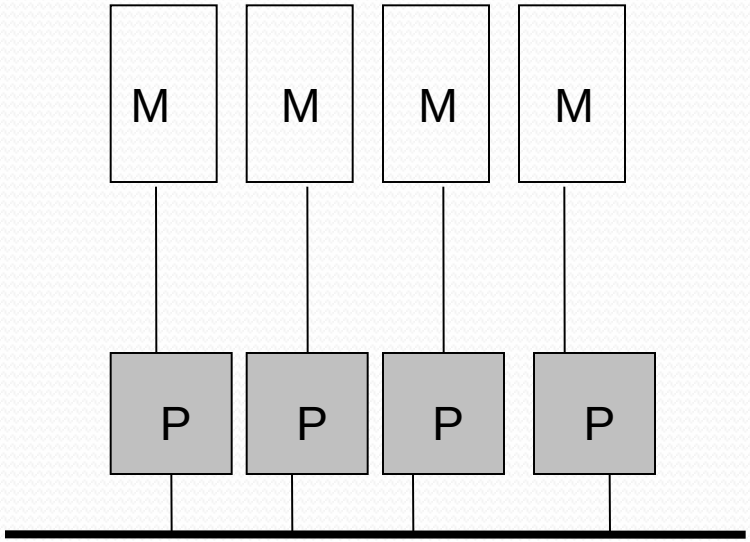
Troca de Mensagens

O paradigma de troca de mensagem é uma abordagem de *computação paralela distribuída* que considera a troca de informação entre *memórias distribuídas* em uma configuração computacional.

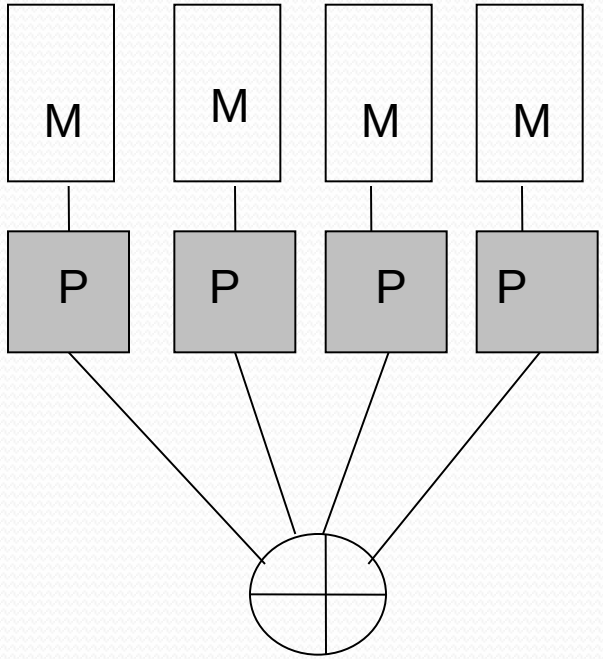
Troca de Mensagens



Configurações Típicas de Multicomputadores



Configuração Compartilhada



Configuração Comutada

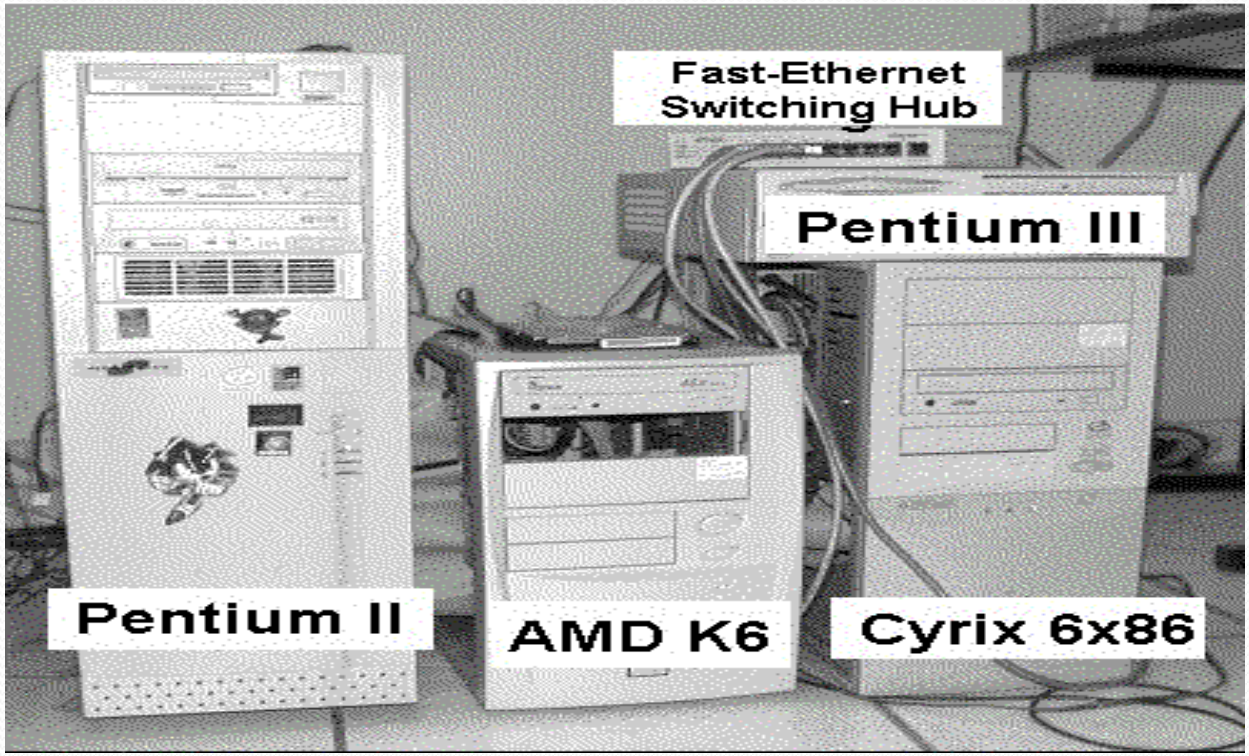
P – Processador

M - Memória

[Dantas, 2005]

Troca de Mensagens

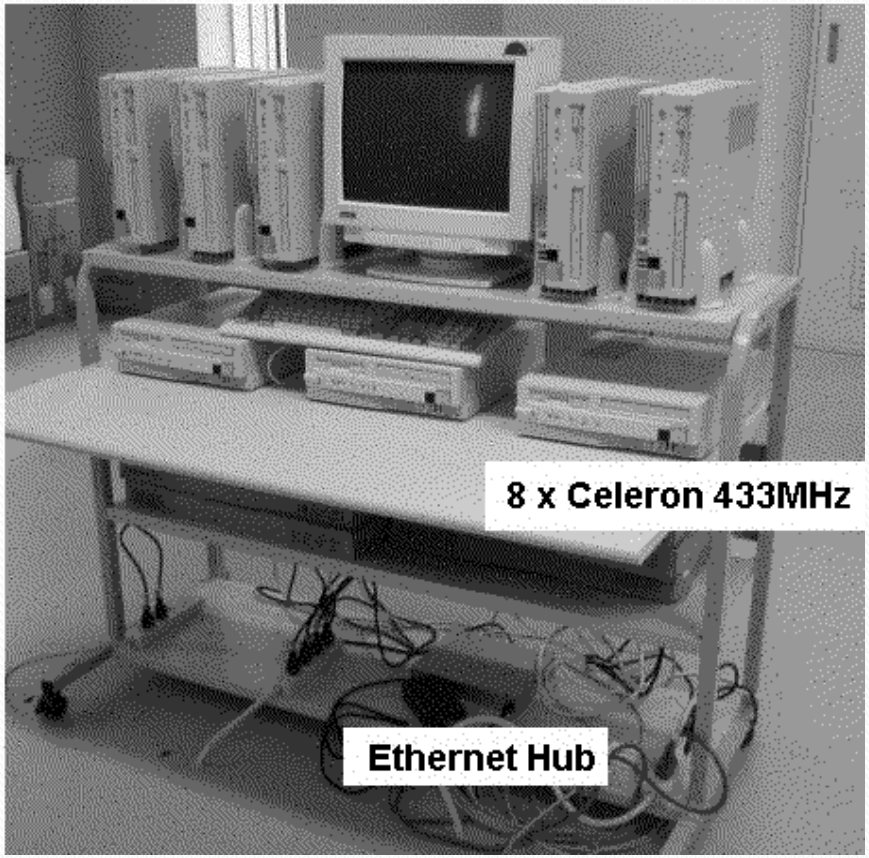
Exemplo de Caso 1: máquinas com configuração heterogêneas



[Dantas, 2005]

Troca de Mensagens

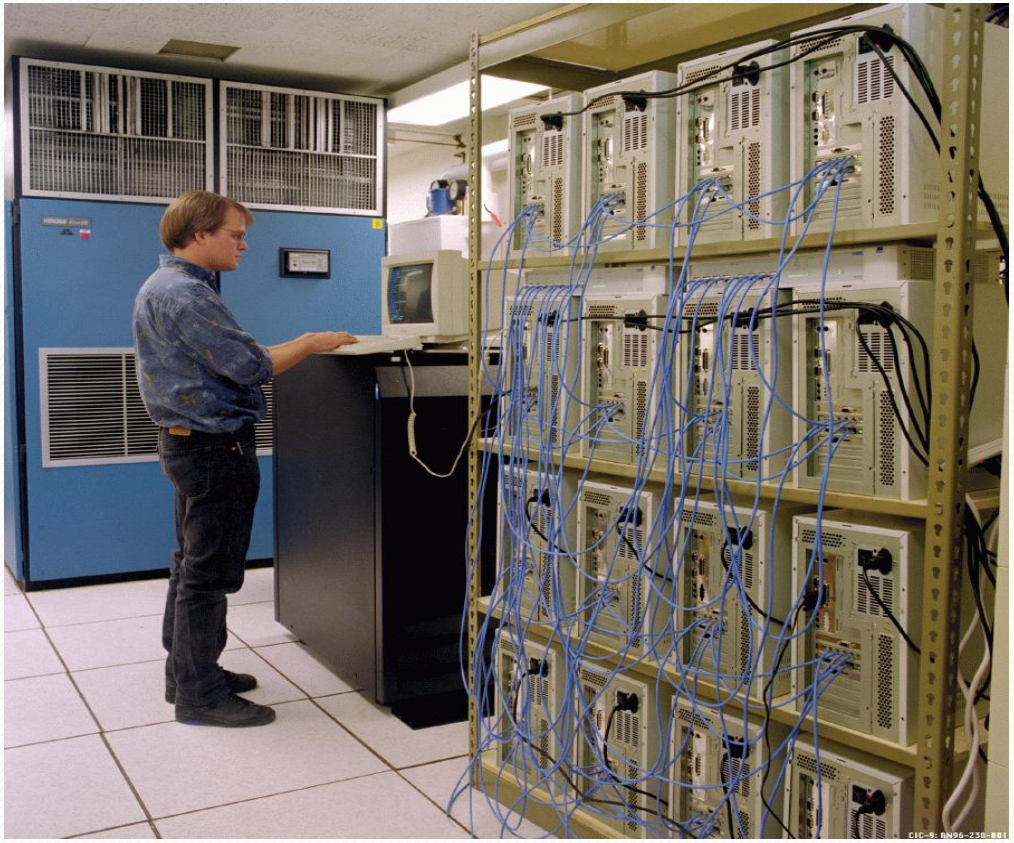
Exemplo de Caso 2: máquinas com configuração homogêneas



[Dantas, 2005]

Troca de Mensagens

Exemplo de Caso 2: máquinas com configuração homogêneas



Troca de Mensagens

Exemplo de Caso 2: máquinas com configuração homogêneas

Troca de Mensagens

Exemplo de Caso 2: máquinas com configuração homogêneas

Troca de Mensagens

Exemplo de Caso 3: máquinas distribuídas em uma rede.

Troca de Mensagens

Definições Básicas

(a) Troca de Mensagens (Message Passing):

Método que permite que o contéudo de uma memória, de um determinado processador, possa ser copiado para outra(s) memória(s). Uma mensagem consiste de um pacote, ou um conjunto de pacotes, que contém o conteúdo que deseja-se enviar/receber.

Troca de Mensagens

Definições Básicas

(b) Processo:

Conjunto de instruções executáveis que rodam em um processador. Um ou vários processos podem estar executando em um processador. Em um sistema de troca de mensagens todos os processos se comunicam através do envio e recebimento de mensagens.

Observe que processos podem estar, ou não, executando sob um mesmo processador.

Troca de Mensagens

Definições Básicas

(c) Biblioteca de Troca de Mensagens:

Coleção de rotinas que são embutidas em uma aplicação para o envio, a recepção e outras operações de troca de informação.

Exemplos são:

- *PVM (Parallel Virtual Machine);*
- *MPI (Message Passing Interface);*
- *p4;*
- *Chameleon;*
- *PARMACS.*

Troca de Mensagens

Definições Básicas

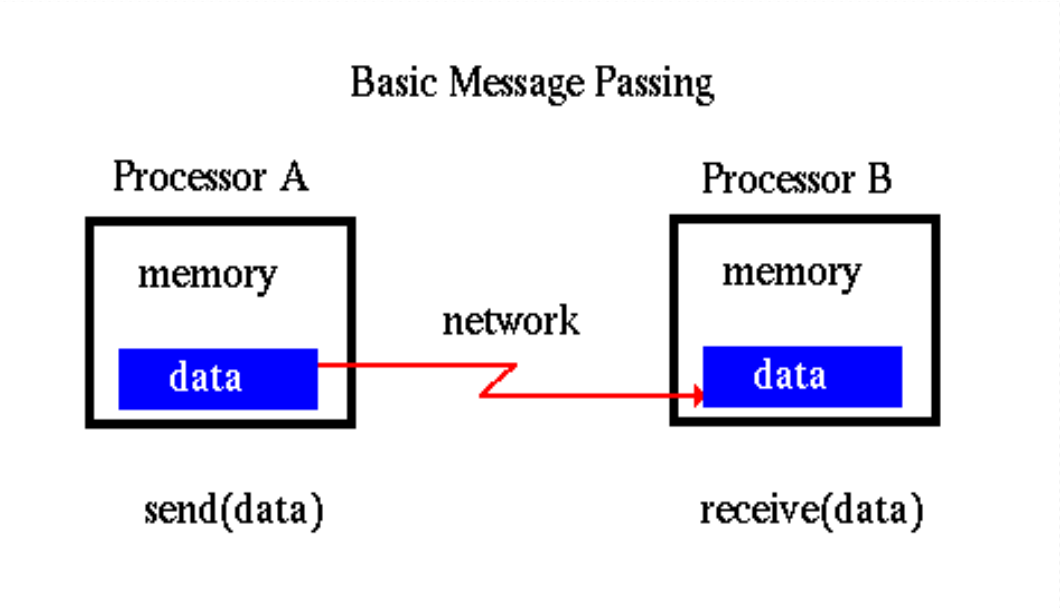
(d) Send/Receive:

A abordagem de troca de mensagem promove a transferência de dados de um processo (send) para outro (receive). Desta forma, a cada send deve existir uma operação receive equivalente.

Em uma operação de send, o processo especifica a localização dos dados, o seu tamanho, o seu tipo e o destino. A operação receive de forma análoga deve possuir os mesmos campos.

Troca de Mensagens

Exemplo de uma comunicação utilizando send/receive



http://www.mhpcc.edu/

Troca de Mensagens

Exemplo de uma comunicação utilizando send/receive

P0	P1
a = 100;	receive(&a,1,0)
send(&a, 1,1);	printf(“%d\n”,a);
a = 0;	

Troca de Mensagens

As operações de *send* e *receive* têm os seguintes tipos:

- send (void *sendbuf, int nelems, int dest)

- receive (void *receivebuf, int nelems, int source)

Troca de Mensagens

Definições Básicas

(e) Comunicação Síncrona-Assíncrona:

Um send síncrono será completado somente quando receber um aviso de recebimento (ack) do processo destino (operação receive).

Por outro lado, um send assíncrono é completado mesmo que a operação receive não tenha sido executada.

Troca de Mensagens

Definições Básicas

(f) Buffer da Aplicação:

O espaço de endereço que armazena os dados que devem ser enviados ou recebidos.

Imagine um programa que utiliza uma variável chamada de "mensagem". O buffer de aplicação para "mensagem" é o local de memória do programa aonde o valor de "mensagem" está armazenado.

Troca de Mensagens

Definições Básicas

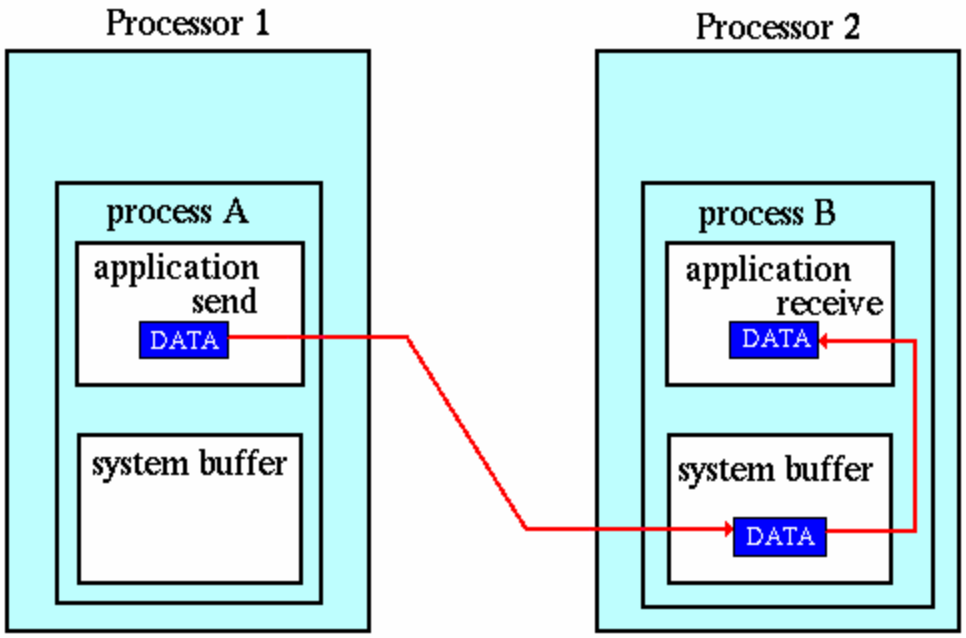
(g) Buffer do Sistema:

É uma área do sistema para armazenamento das mensagens. Dependendo do tipo de operação de send-receive, dados da área do buffer da aplicação podem solicitar a copia de-para o sistem de buffer do sistema.

O buffer do sistema permita a comunicação assíncrona.

Troca de Mensagens

Exemplo de um buffer de sistema:



Path of a message buffered at the receiving process

http://www.mhpcc.edu/

Troca de Mensagens

Definições Básicas

(h) Comunicação Bloqueante :

Uma rotina de comunicação é bloqueante se para terminar esta depende de certos eventos. No caso de operações do tipo send, os dados devem ter sido enviados com sucesso, ou armazenados com sucesso no buffer do sistema, assim permitindo que o buffer da aplicação possa reusar estas informações.

Em uma operação de receive, os dados devem estar armazenados com sucesso no buffer de recepção para que possam ser usados.

Troca de Mensagens

Definições Básicas

(i) Comunicação Não-Bloqueante:

Rotinas de comunicações desse tipo são caracterizadas por não precisar esperar nenhum evento para serem completadas.
Um exemplo seria a cópia de uma mensagem de um usuário para o sistema de memória ou chegada de mensagem.

Troca de Mensagens

Definições Básicas

(i) Comunicação Não-Bloqueante (cont.):

Não é seguro a modificação ou uso do buffer de aplicação depois do término de uma operação de send não-bloquenate.

Fica a cargo do programador da aplicação que o buffer da aplicação poderá estará livre para ser reusado.

Usualmente comunicações não-bloqueantes são usadas para melhoria de desempenho, pois podem sobrepor computação e comunicação.

Troca de Mensagens

Definições Básicas

(j) Comunicadores e Grupos:

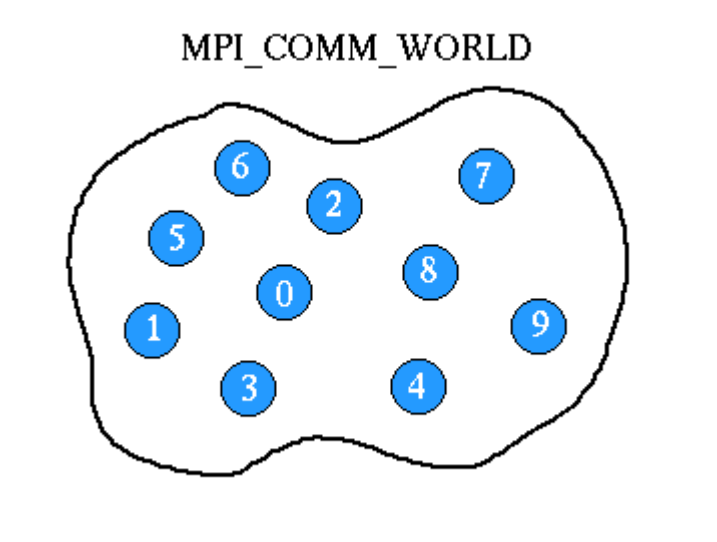
São componentes empregados por ambientes de troca de mensagem para determinar o tipo de comunicação entre os processos.

Um exemplo é a biblioteca MPI que requer a especificação um comunicador como um argumento.

O componente de comunicação MPI_COMM_WORLD é empregado para se referir a comunicação entre todos os processos MPI em uma determinda aplicação.

Troca de Mensagens

Exemplo de escopo do comunicador MPI_COMM_WORLD



http://www.mhpcc.edu/

Troca de Mensagens

Definições Básicas

(k) Rank:

Dentro de um componente comunicador, cada processo possui seu identificador único que é um número inteiro fornecido pelo sistema de troca de mensagem assim que o processo é inicializado.

O rank usualmente é chamado de "process id", tem como característica ser contíguo e iniciar com zero.

Troca de Mensagens

Definições Básicas

(k) Rank (cont...):

O rank é empregado pelo programdor para especificar a origem e destino das mensagens.

O rank, também, é usado em condicionais nas aplicações para contolar a execução dos programas.

```
Exemplo: if rank= 0
              then .....
              else if rank=1
                        then ...
                        else ...
                   endif
         endif
```

Conteúdo Programático

Modulo I

. Introdução a computação distribuída

. Ambientes de middleware

Modulo II

. Paradigma de troca de mensagens

. **Parallel Virtual Machine (PVM)**

. Message Passing Interface (MPI)

. Ambientes de Memória Compartilhada

PVM

PVM (Parallel Virtual Machine) :

Pacote de troca de mensagens que permite que uma coleção de sistemas computacionais (serial, paralelo ou vetorial) em uma rede possam ser agregados e gerenciados com um *grande recurso computacional*.

A abordagem do modelo computacional é conhecida como *metacomputing*. Esta permite que seja possível alcançar:

- alto desempenho;
- elementos computacionais adaptados para subproblemas
- utilização de recursos para visualização

PVM

Programando em PVM

(a) ambiente de troca de mensagem simples: fácil adição de hosts, submissão de tarefas, envio de mensagens, não existe topologia fixa, a máquina virtual pode ser formada por diferentes tipos de máquinas;
(b) Controle de processo: processos podem ser submetidos/terminados em qualquer lugar da máquina virtual;

(c) Comunicação: qualquer tarefa pode ser comunicar com qualquer outra, a troca de formatos é efetuado pelo PVM;

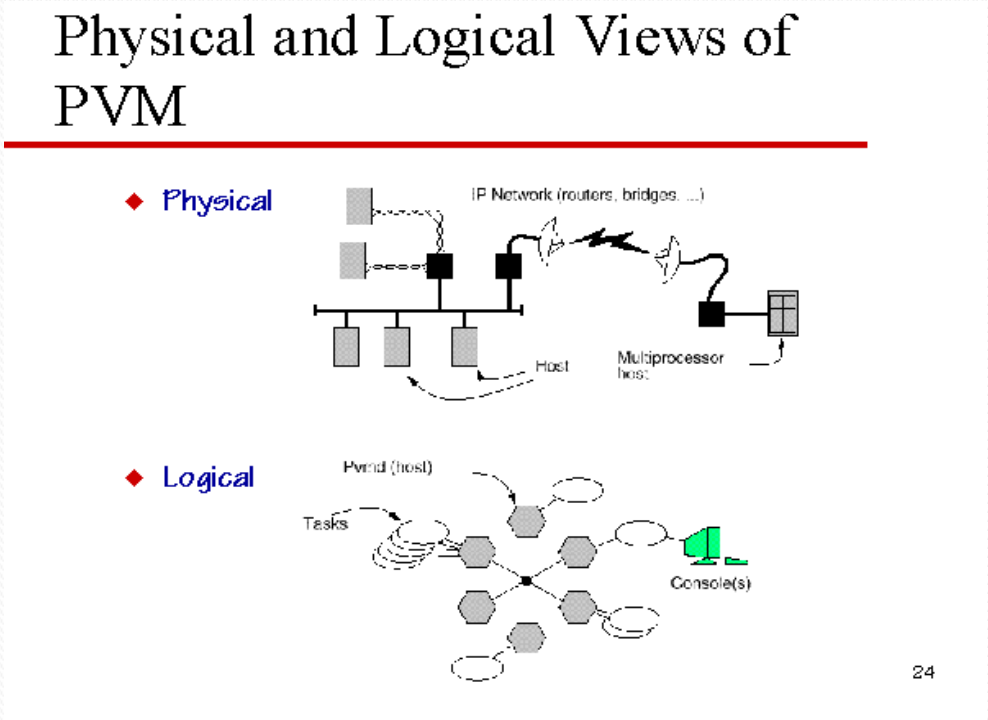
PVM

Programando em PVM

(d) Grupos dinâmicos de processos: tarefas podem serem anexadas ou partir de um ou mais grupos a qualquer momento;
(e) Tolerância a falha: uma tarefa pode requisitar notificação de perda ou ganho de recursos;
(f) Sistema operacional : executa sob uma grande gama de sistemas;
(g) Liguagens: C, C++, Fortran ou outra linguagem que possa ser “linkada” a linguagem C

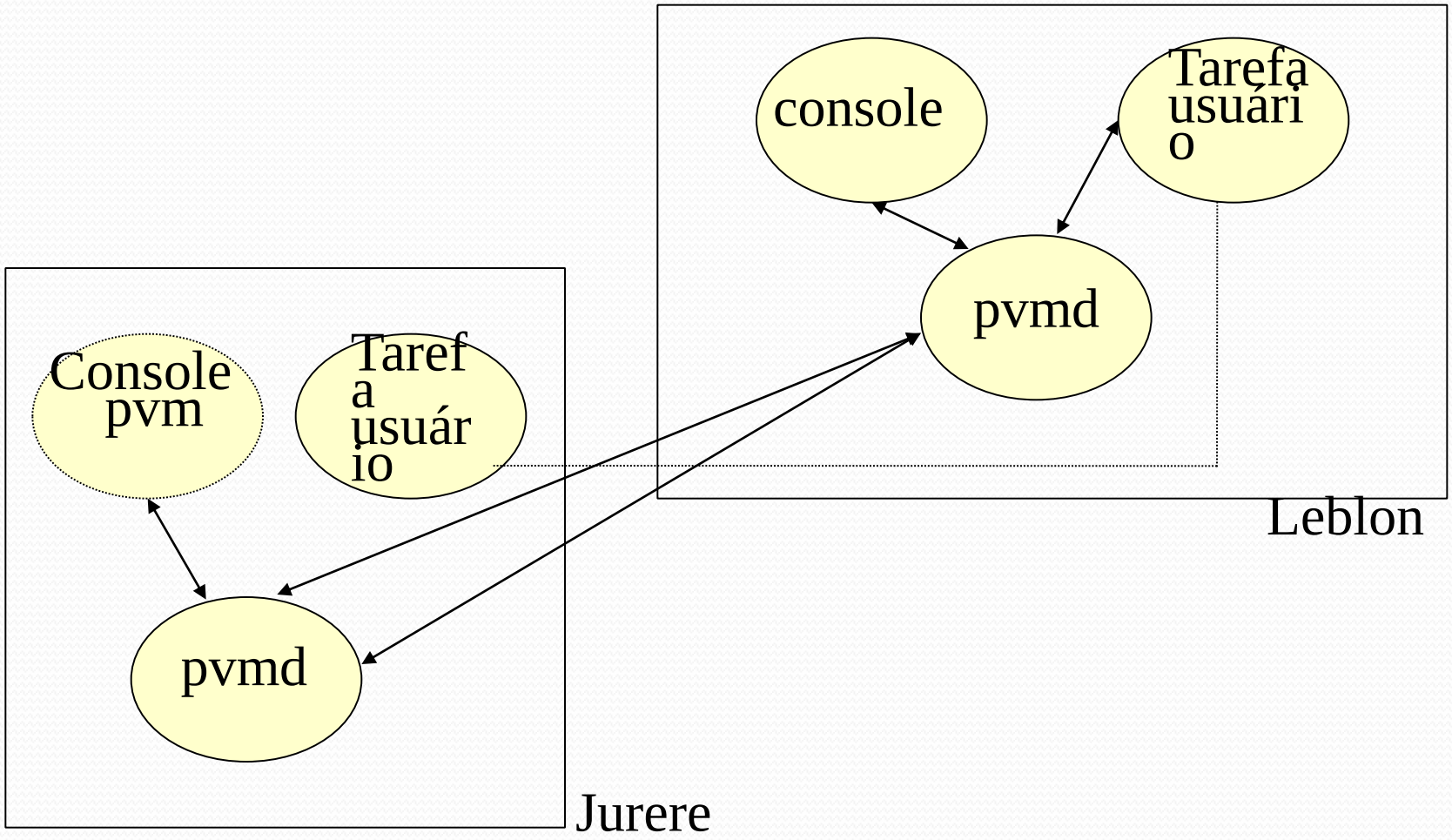
PVM

O PVM permite que tenhamos uma visão lógica de uma configuração física de um ambiente computacional.



[http://www.netlib.org/]

PVM


console
Tarefa usuário
pvmd
Leblon
Console pvm
Tarefa usuário
pvmd
Jurere

PVM

O ambiente PVM é composto de quatro compomentes:

. pvmd - um daemon

. libpvm - a biblioteca de funções do pvm

. console - interface entre o usuário e o sistema

. aplicação - programa do usuário

PVM

Componente pvmd:

. Roda em cada hospedeiro da máquina virtual
. Autentica tarefas
. Executa processos no hospedeiro
. Provê detecção de falhas
. Roteia mensagens
. Mais robusto que as aplicações
. Contato com os hospedeiros

PVM

Conceitos do PVM

- Hospedeiro: o computador, também chamado nó

- Máquina Virtual: uma meta máquina composta de um ou mais hosts

- Processo: um programa: dados, pilha

PVM

Conceitos do PVM

- Tarefa: um processo pvm
- pvmd: o daemon do pvm
- Mensagem: uma lista ordenada de dados enviada de uma tarefa para outra
- Grupo: uma lista ordenada de tarefas que recebem um nome simbólico

PVM

A inicialização e operação de funções de conexão de hosts na ambiente PVM são executadas de maneira simples, como mostramos abaixo:

% pvm (chamada de inicialização do pacote de software)
pvm> add hostname (adicionar um hosts no ambiente)
pvm> delete hostname (remover um host do ambiente)
pvm> conf (visualização da máquina virtual)
pvm> halt (finaliza a execução da máquina virtual)
pvm> ps -a (visualização de tarefas sendo executadas)

PVM

Operações Básicas de Troca de Mensagens

Comando *send*

Sintaxe: send (address, length, destination, tag)

send: operação de envio de mensagem

address: endereço inicial do buffer contendo os dados a serem enviados

length: comprimento em bytes da mensagem

destination: identificador do processo para o qual a mensagem deve ser enviada

tag: um identificador arbitrariamente escolhido para restringir o recebimento da mensagem

PVM

Operações Básicas de Troca de Mensagens

Comando *receive*

Sintaxe: receive (address, maxlength, source, tag,actlen)

recv:operação de recepção de mensagem

address: endereço inicial do buffer onde os dados serão recebidos

maxlength: comprimento máximo em bytes da mensagem

source: identificador do processo do qual a mensagem deve ser recebida

tag: um identificador arbitrariamente escolhido para restringir o recebimento da mensagem

actlen: número de bytes realmente recebidos o recebimento da mensagem

PVM

Exemplo de configuração de uma máquina virtual

```
pvm> conf
1 host, 1 data format
                HOST    DTID    ARCH  SPEED
                lapesd1   40000    LINUX    1000
pvm> add jurere
1 successful
                HOST    DTID
                 jurere    80000
pvm> add leblon
0 successful
                HOST    DTID
                leblon  can't start pvmd
```

PVM

Exemplo de configuração de uma máquina virtual

```
pvm> conf
2 hosts, 2 data formats
 HOST     DTID     ARCH   SPEED
 lapesd1    40000    LINUX    1000
 jurere     80000    SUN4     1000
pvm>
```

PVM

```
#include <stdio.h>
#include "pvm3.h"

main()
{
        int cc, tid;
        char buf[100];

        printf("i'm t%x\n", pvm_mytid());

        cc = pvm_spawn("hello_other", (char**)0, 0, "", 1, &tid);

        if (cc == 1) {
                cc = pvm_recv(-1, -1);
                pvm_bufinfo(cc, (int*)0, (int*)0, &tid);
                pvm_upkstr(buf);
                printf("from t%x: %s\n", tid, buf);

        } else
                printf("can't start hello_other\n");

        pvm_exit();
        exit(0);
}
```

Código do mestre

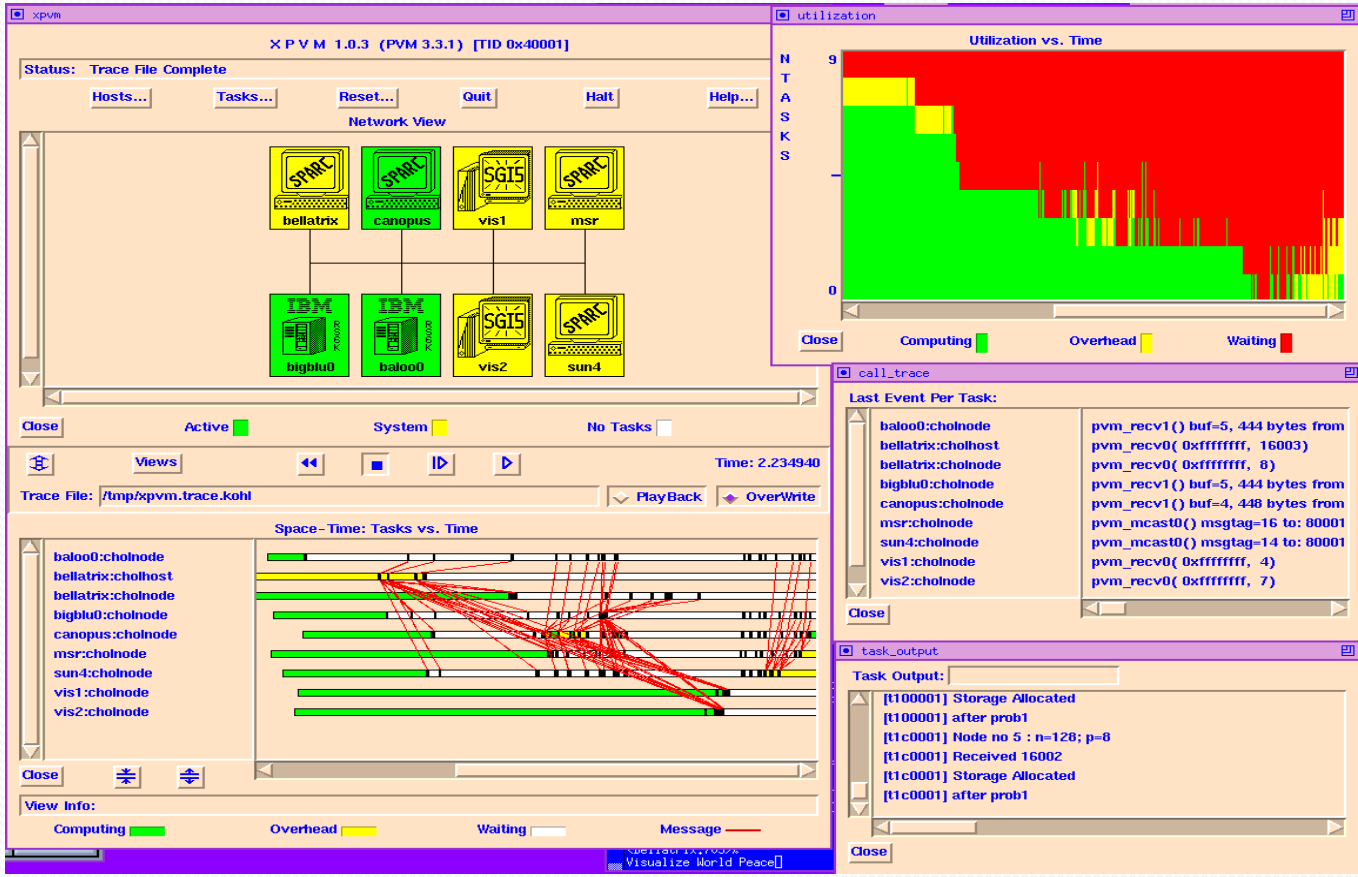
PVM

```
#include "pvm3.h"

main()
{
        int ptid;
        char buf[100];

        ptid = pvm_parent();

        strcpy(buf, "hello, world from ");
        gethostname(buf + strlen(buf), 64);

        pvm_initsend(PvmDataDefault);
        pvm_pkstr(buf);
        pvm_send(ptid, 1);

        pvm_exit();
        exit(0);
}
```

Código do *escravo*

PVM

Exemplo da interface gráfica XPVM



Conteúdo Programático

Modulo I

. **Introdução a computação distribuída**

. **Ambientes de middleware**

Modulo II

. Paradigma de troca de mensagens

. **Parallel Virtual Machine (PVM)**

. **Message Passing Interface (MPI)**

. Ambientes de Memória Compartilhada

Computação Paralela com Troca de Mensagem

- Uma tarefa é dividida em várias, executadas por vários *processos*, em *um* (ou *mais*) *processador*(es);

- Processos interagem trocando informação

- Qual é fundamento básico necessário para funcionamento?

  - Executar um procedimento para inicializar as tarefas;
  - Prover uma maneira de comunicação entre as tarefas.

Comunicação

- Cooperativa
  - Todas as parte entram em acordo para troca de dados
  - Um exemplo é a troca de mensagens
    - Dados são explicitamente enviados (send) e recebidos (receive)
    - Qualquer troca na memória destino é efetuada com a participação do processo destino
- Num sentido
  - Um *worker* efetua a troca de dados
  - Dado por ser acessado sem esperar por outro processo.

O que é MPI?

- Uma especificação de ***biblioteca*** para troca de mensagens
  - Modelo de *Message-passing*
  - Não é uma especificação de compilador
  - Não é uma linguagem
  - Não é um produto específico
- Projetada para computadores paralelos, clusters e redes heterogeneas.

DESENVOLVIMENTO DO MPI

- Desenvolvimento iniciou-se em 1992
- Participação aberta no esforço
  - IBM,Intel, TMC, Meiko, Cray, Convex, Ncube
  - PVM, p4, Express, Linda, ...
  - Laboratórios, Universidades, Orgãos Governamentais
- Versão final finalizada em Maio de1994
- Implementações abertas e comerciais estão disponíveis.

Comunicação Ponto-a-Ponto

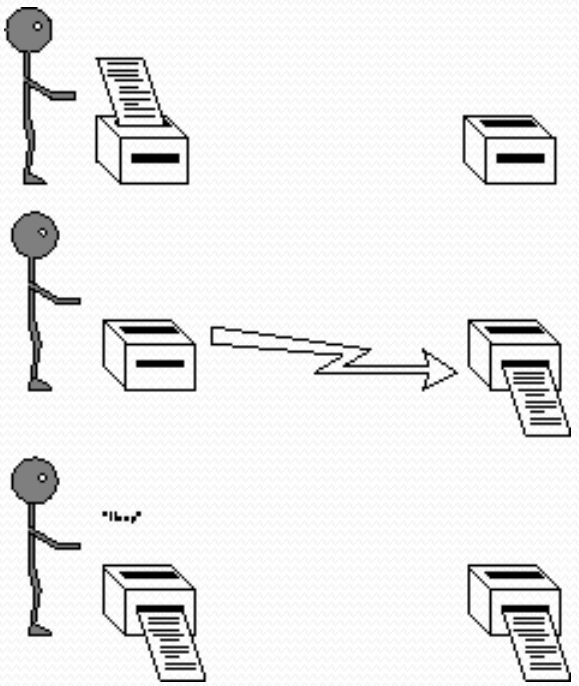
- Mensagens são enviadas pelo remetente para um (ou mais) destinário(s)

- Existem vários *opções* de como uma chamada de envio de mensagem pode interagir com um programa

Comunicação Síncrona

- Uma comunicação síncrona é caracterizada por seu término somente após a mensagem ser recebida.

  Exemplos para visualização seriam
  - Um FAX ou um e-mail registrado

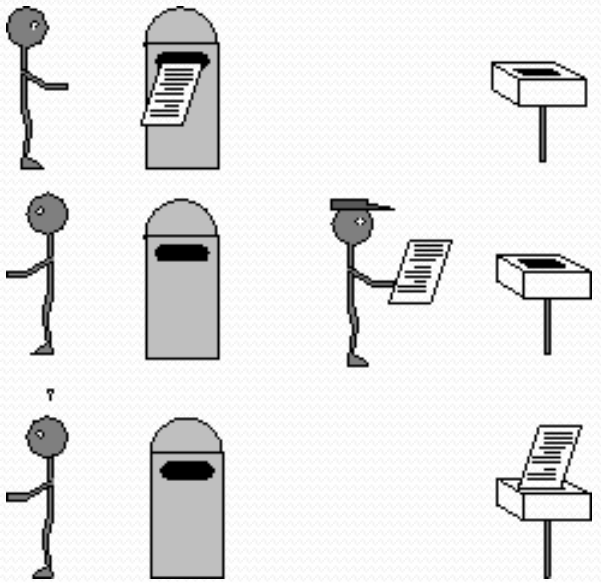


Comunicação Assíncrona

- Na comunicação assíncrona é finalizada assim que a mensagem foi enviada.

  Exemplos seriam:

  - O envio de um cartão ou e-mail convencional



Bloqueante e Não-bloqueante

- Operações bloqueantes somente retornam depois que a operação foi completada
  - Exemplo seria uma maquina de FAX
- Operações não -bloquenates não ficam no aguardo, permitem que após o envio o processo execute outra tarefa

  - Recebimento de um FAX

Comunicação Coletiva

- Comunicação ponto-a-ponto envolve um par de processos.
- Muitos sistemas de troca de mensagens provêm operações que permitem que um grande número de processos possam participar.

Tipos de Transferências Coletivas

- Barreira
  - Sincroniza processadores
  - Nenhum dado é trocado, a operação de barreira bloqueia todos os processos até que todos tenham chamado a rotina de barreira
- Broadcast (algumas vezes *multicast*)
  - Uma comunicação *broadcast* é to tipo *um-para-muitos*
  - Um processador envia uma mensagem para vários destinos
- Redução
  - Usualmente ágil para a comunicação *vários-para-um*

O que é uma Mensagem?

- Uma mensagem MPI é um vetor de elementos de um tipo particular MPI
- Todas as mensagens em MPI são *tipadas*
  - O tipo de conteúdo deve ser especificado nas operações de *send* e *receive*

Parâmetros de Controle do MPI

- O MPI mantem estruturas de dados internos que são referenciados pelo usuário através dos parâmetros de controle
- Os parâmetros de controle podem ser passados e retornados por procedimentos do MPI
- Parâmetros de controle podem ser cópiados por operações comuns de atribuição

Erros no MPI

- As rotinas MPI retornam um valor *int* que contem um código de erro
- A ação natural na detecção de um erro é causar uma interrupção na operação paralela
  - É possível efetuar uma troca para retornar um código de erro

Inicializando o MPI

- A primeira rotina chamada em um programa MPI é MPI_INIT
- Todos os processos chamam a rotina MPI_INIT, antes da execução de outras rotinas

```
int mpi_Init( int *argc, char **argv );
```

Esqueleto de Programa MPI

```
#include <mpi.h>

main( int argc, char** argv ) {
    MPI_Init( &argc, &argv );

    /* main part of the program */

    MPI_Finalize();
}
```

Comunicadores

- Um parâmetro de controle de comunicação (*comunicador*) define para quais processos específicos deverão ser aplicados a comunicação
- Todos as comunicações em MPI consideram um comunicador como parâmetro. De uma outra forma, o comunicador é que efetivamente guia o contexto no qual a comunicação será efetuada.
- MPI_INIT define um comunicador denominado de MPI_COMM_WORLD para cada processo que o chamam

Comunicadores

- Cada comunicador é caracterizado por um grupo, que é uma lista de processos
- Todos os processos são ordenados a partir de o
- O número de cada processo é chamado de *rank*

  - O *rank* identifica cada processo dentro de um comunicador
- O grupo do MPI_COMM_WORLD é o conjunto de todos os processos MPI

Comunicação Ponto-a-ponto

- Comunicação comum entre dois processos
- Nó destino é identificado por seu *rank* no identificador
- Existem quatro modos de comunicação providas pelo MPI (relativas a operação *send*)
  - Buffered
  - Synchronous
  - Standard
  - Ready

Send Padrão

- Quando empregando o *send padrão:*

  - Fica a cargo do MPI decidir quando as mensagens serão *bufferizadas*

  - É considerada uma operação com sucesso quando a mensagem for enviada, não implicando que a mesma tenha chegado ao destino

  - Pode ser iniciado sem que tenha uma batimento com um receive

  - Não existe uma verificação local sobre a semantica.

Send Padrão

```
MPI_Send( buf, count, datatype, dest, tag, comm )
```

Where

- `buf` is the address of the data to be sent
- `count` is the number of elements of the MPI datatype which `buf` contains
- `datatype` is the MPI datatype
- `dest` is the destination process for the message. This is specified by the rank of the destination within the group associated with the communicator `comm`
- `tag` is a marker used by the sender to distinguish between different types of messages
- `comm` is the communicator shared by the sender and the receiver

Send Síncrono

- `MPI_Ssend( buf, count, datatype, dest, tag, comm )`
  - Pode ser iniciado sem existir um *receive equivalente*
  - Somente será bem sucessido se existir um *receive postado, e a operação de receive foi iniciada para receber a mensagem da operação de send síncrono*

Send Síncrono

- `MPI_Ssend( buf, count, datatype, dest, tag, comm )`
  - Prove uma semântica de comunicação síncrona: a comunicação não se realiza até ambos os processos que estiverem realizando uma comunicação tenha um ponto de *rendezvous*
  - Não possui uma semântica de *verificação de completação local.*

Send Bufferizado

- O modo de *send bufferizado é characterizado por:*
  - Pode ser iniciado se existir (ou não) um *receive postado. Deverá ser completado antes que um receive seja postado.*
  - Possui uma semântica local para ser completado: todavia não depende do *receive postado*
  - Uma condição para completar satisfatoriamente, é possível que seja necessário *bufferizar as mensagens localmente. Desta forma, um espaço de buffer é provido pela aplicação*

Modo ReadySend

- O modo ready send
  - Completa imediatamente
  - Só se inicia se existir um *receive postado*
  - Possui uma semântica semelhante ao modo de *send padrão*
  - Economiza em *overhead, evitando handshaking e bufferização*

Conteúdo Programático

Modulo I

. **Introdução a computação distribuída**

. **Ambientes de middleware**

Modulo II

. Paradigma de troca de mensagens

. **Parallel Virtual Machine (PVM)**

. Message Passing Interface (MPI)

. **Ambientes de Memória Compartilhada**

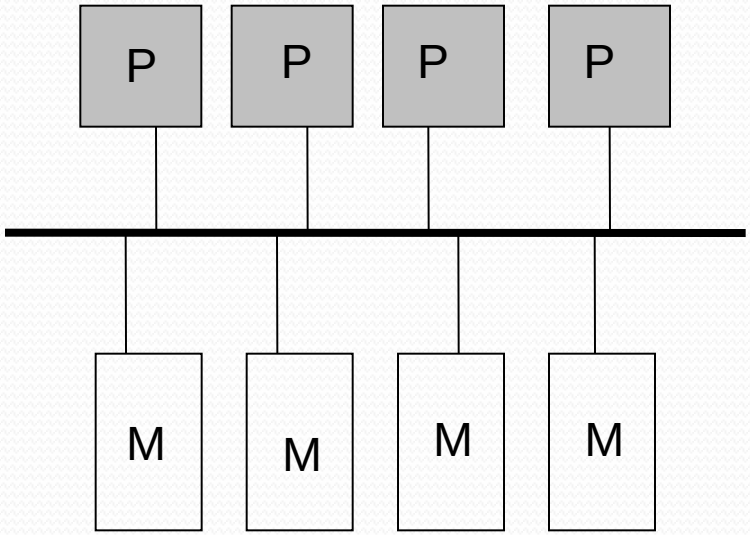
Memória Compartilhada Distribuída

Aspecto de Hardware

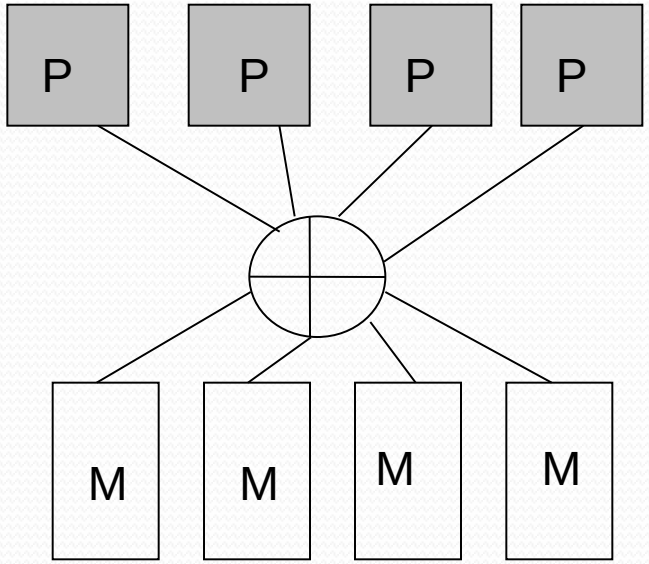
Em arquiteturas computacionais (*hardware*),

memória compartilhada (*shared memory)*

se refere usualmente a um grande *bloco de memória RAM (Random Access Memory)*, que pode ser acessado por um número grande de diferentes processadores em uma configuração ***Multiprocessada.***



Configuração Compartilhada

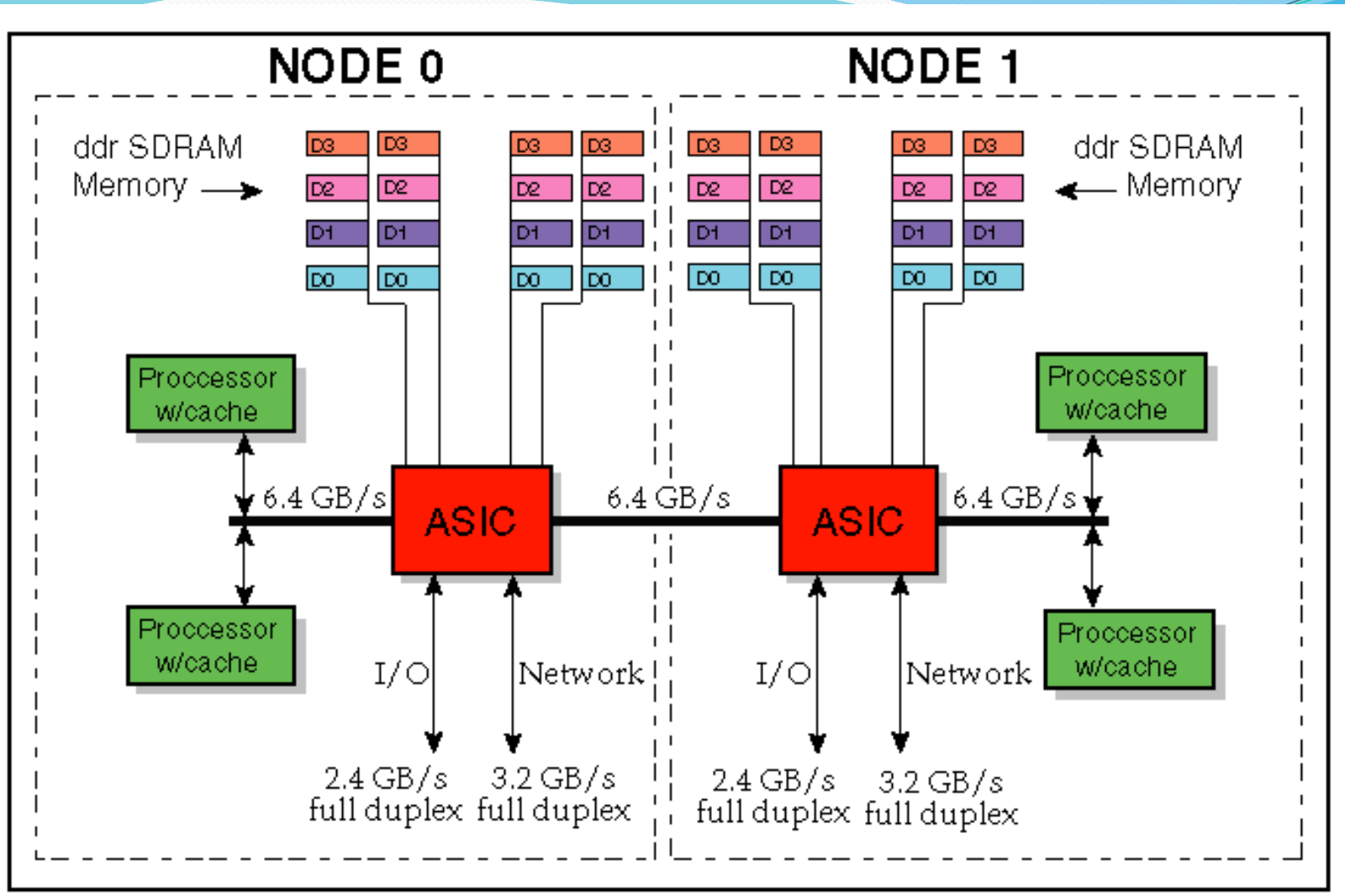


Configuração Comutada

P – Processador M - Memória

Exemplos de Multiprocessadores

[Dantas, 2005]



Altix SGI

[Dantas, 2005]

O problema conhecido com relação aos *multiprocessadores* é que os processadores devem ter acesso rápido a memória semelhante ao nível de *cache*.

Devido a problemas de conteção é usual que as configurações multiprocessadas utilizem *caches locais* para melhorar o desempenho da arquitetura.

O desafio dessa solução é que sempre que ocorra uma alteração dos *caches dos* processadores devam ser notificados.

Caso contrário, os processadores estarão trabalhando em dados incoerentes.

O problema de coerência de *cache* e *memória* é um desafio antigo e bastante conhecido em arquiteturas distribuídas.

Existem vários protocolos que auxiliam no melhor desempenho de arquiteturas multiprocessadas.

Os paradigmas de ***memórias distribuídas*** e ***memória compartilhadas distribuídas*** representam dois grandes grupos de pesquisas relativos ao uso de memória com eficiência.

Na abordagem de memórias distribuídas, temos uma arquitetura de ***multicomputadores***.

Nesta o acesso a informação é efetuado através de troca de mensagem (exemplos são PVM e MPI)

Na abordagem de ***memória compartilhada distribuída*** temos um arquitetura multicomputada.

Todavia, para os usuários das aplicações o modelo de programação se apresenta como de *memória compartilhada*, ou seja *multiprocessado*.

Observação:

Tanto a abordagem de ***troca de mensagem*** quanto ao ***compartilhamento de memória*** são abordagens básicas de comunicação entre processos (*inter-process communication- IPC)*

Aspecto de Software

Para que um ambiente *multicomputado* pareça para o usuário final como *multiprocessado*, se faz necessário que uma porção de software efetue a transparência necessária.

Pacotes de DSM (Distributed Shared Memory), tais como Threadmarks, Jiajia e OpenMosix provêem a transparência necessária para o usuário final.

Conteúdo Programático

Modulo III

. Objetos distribuídos

. Componentes

. CORBA

. Barramento de objetos CORBA

. Adaptador básico de objetos

. Linguagem de definição de interfaces

Introdução aos Objetos Distribuídos

A comunicação nos sistemas distribuídos pode ser efetuada utilizando-se técnicas tais como :

- troca de mensagem;

- memória compartilhada;

- chamada remota de procedimento;

- ***objetos distribuídos.***

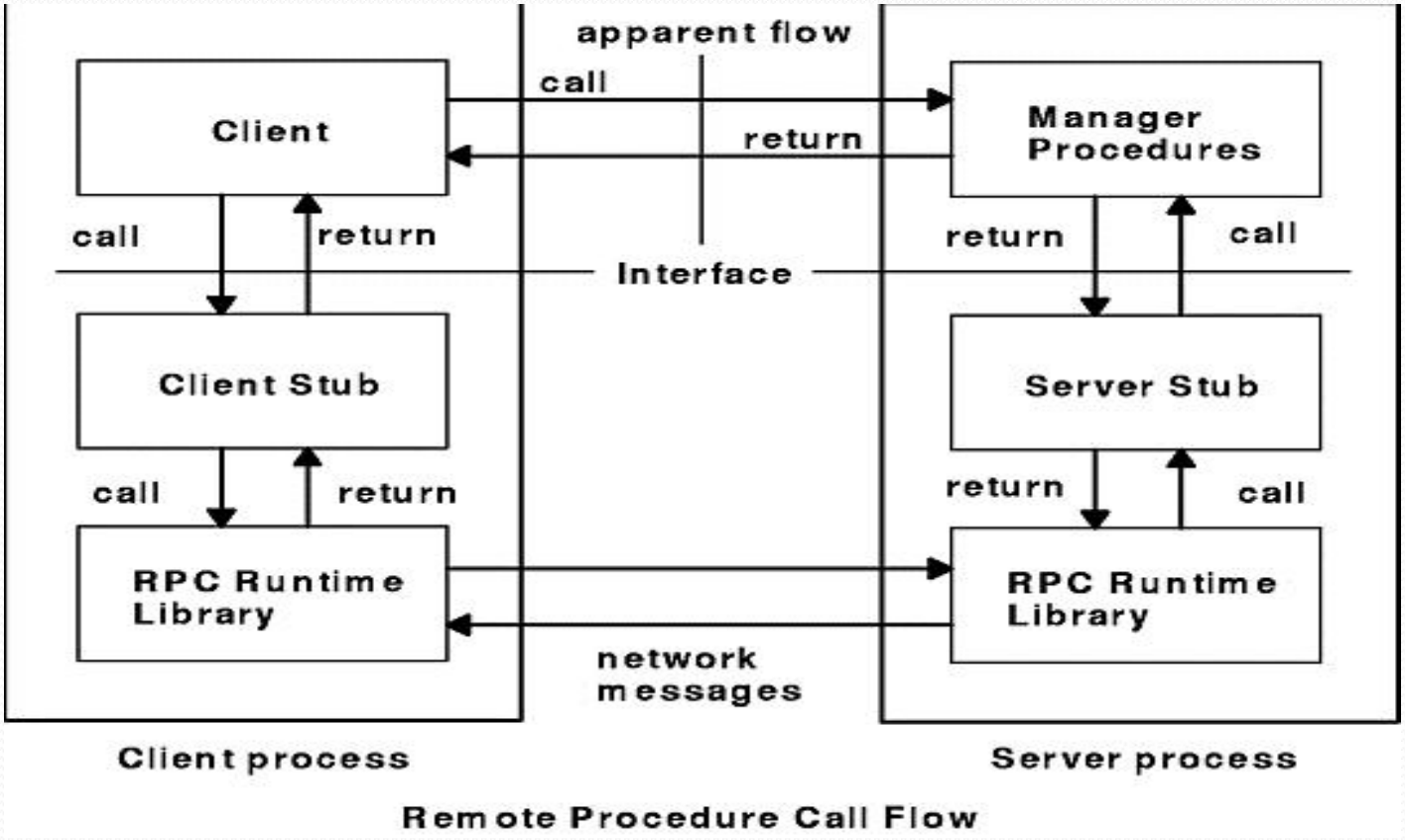
A idéia básica dos ***objetos distribuídos*** é uma extensão do conceito de *chamada remota de procedimentos* (*RPC).*

Na abordagem dos sistemas baseados em

remote procedure call

um código é executado remotamente por intermédio de uma chamada RPC.

Fluxo de uma Chamada RPC



[www.ibm.com]

Exemplos comerciais de sistemas RPC(*Remote Procedure Call*) *temos :*

- *SUN- RPC;*
- *DCE-RPC;*
- *JAVA-RMI.*

SUN- RPC

Foi a RFC 1050 proposta pela SUN em 1988, cujo objetivo é a chamada remota de um procedimento e a passagem de um parâmetro.

SUN- RPC

O funcionamento convencional de uma RPC tem o seguinte padrão:

1. Ao iniciar-se um servidor RPC registra-se como um serviço denominado de *portmapper*, ou *rpcbind.* Este serviço é local;

2. O servidor RPC não se conecta diretamente a porta. O servidor solicita ao serviço *portmapper* uma porta. O serviço *portmapper* reside na porta 111.

SUN- RPC

O funcionamento convencional de uma RPC tem o seguinte padrão (cont.):

3. O *portmapper* acha uma porta apropriada e associa o Servidor RPC a essa porta;

4. A partir desse momento, qualquer solicitação de serviços clientes ao Servidor RPC é respondido pelo *portmapper* que indica a localização do Servidor RPC.

SUN- RPC

A abordagem que apresentamos de funcionamento do RPC é a maneira convencional de operação.

DCE-RPC

O OSF (Open Software Foundation) DCE(*Distributed Computing Environment)* é um padrão da indústria, independente de fabricante, cuja meta é o estabelecimento de tecnologias padronizadas para a computação distribuída.

Desta forma, o *DCE-RPC* é um padrão para a interoperabilidade entre ambientes de fabricantes distintos, empregando a abordagem de chamadas remotas de procedimentos [www.opengroup.org].

JAVA-RMI

A técnica conhecida como RMI (*Remote Method Invocation*) é uma solução existente para o *mundo* Java.

Nesta abordagem um programador pode criar rotina que pode invocar um outra rotina remota em máquinas com diferente arquitetura.

JAVA-RMI

O RMI utiliza a serialização de objetos para *empacotar* e *desempacotar* parâmetros, não fazendo um truncamento, ou seja suportando um real poliformismo orientado a objeto [java.sun.com].

Considerações 1/3

. O protocolo RPC é independente dos protocolos de transporte;

. A forma através da qual uma mensagem é passada de um processo para outro não faz diferença para operações RPC;

. O protocolo RPC apenas se preocupa com a especificação e interpretação das mensagens;

Considerações 2/3

. A RPC não implementa nenhum esquema de confiabilidade;

. A aplicação deve estar consciente do protocolo sob o qual a RPC está baseada;

. Se a RPC está sob um protocolo confiável, tipo TCP, não deverá muita preocupação por parte da aplicação com a comunicação;

Considerações 3/3

. Por outro lado, se o protocolo de transporte utilizado não for confiável, exemplo UDP, a aplicação deverá implementar políticas de retransmissão e temporizadores, uma vez que a RPC não disponibiliza tais facilidades.

Em suma :

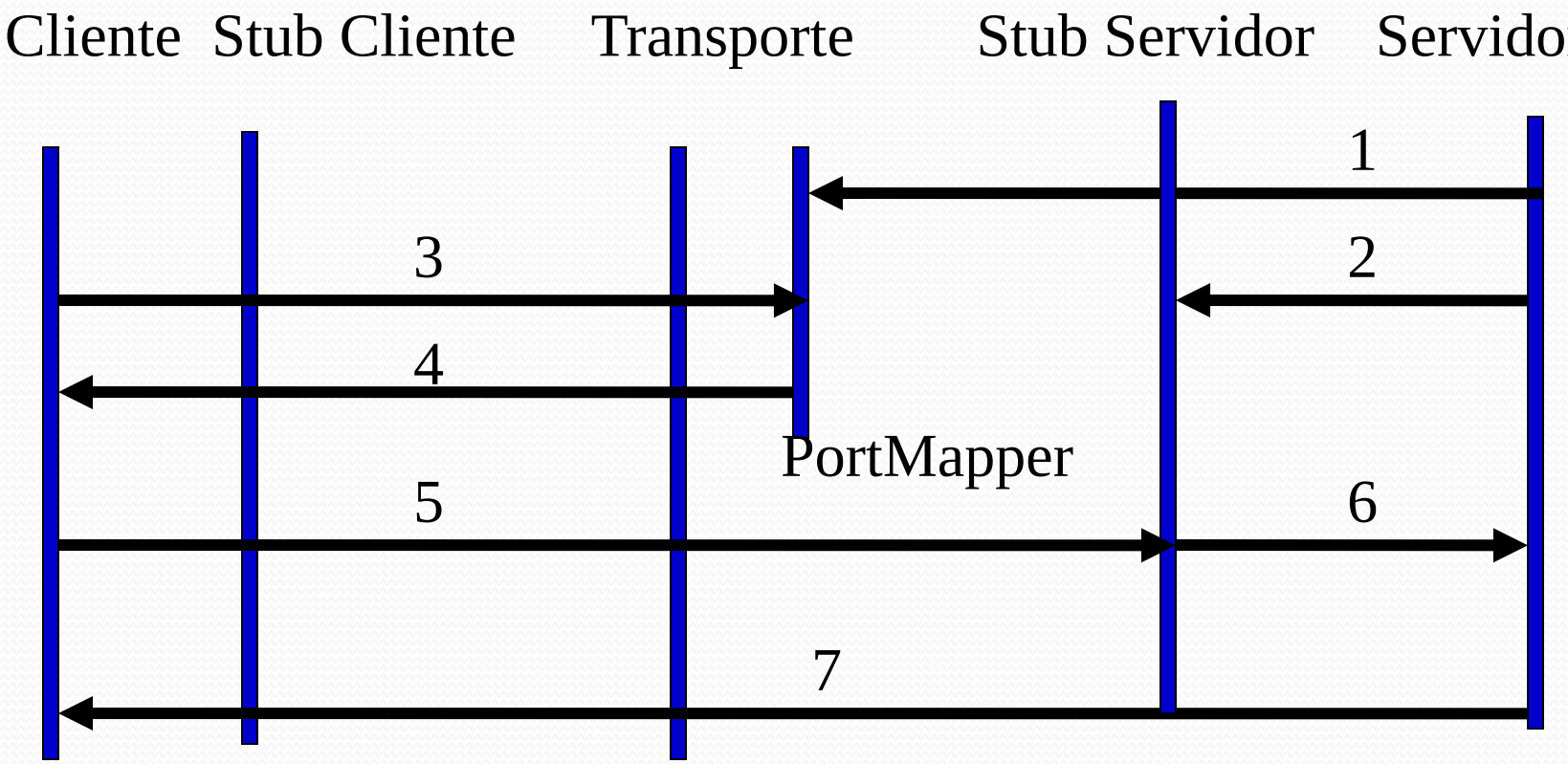
-No paradigma RPC a unidade de distribuição é um procedimento (uma função ou método). Um cliente deve importar um *stub* para poder efetuar uma conexão com um servidor que oferece uma chamada remota.

Remote Procedure Call

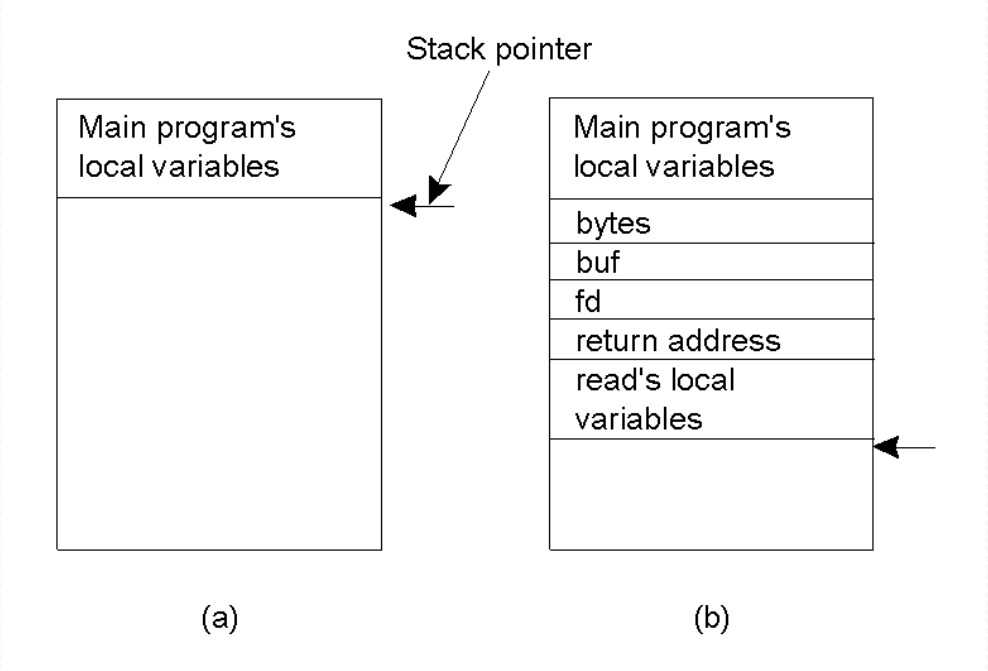
O mecanismo de RPC integra o protocolo RR usado para comunicação Cliente/Servidor com as linguagens de programação convencionais permitindo clientes comunicar com servidores através de chamadas de procedimentos.

A chamada remota segue o modelo da chamada local sendo que o procedimento chamado executa em um processo diferente normalmente em um computador diferente.

Diagrama de Interação em RPC



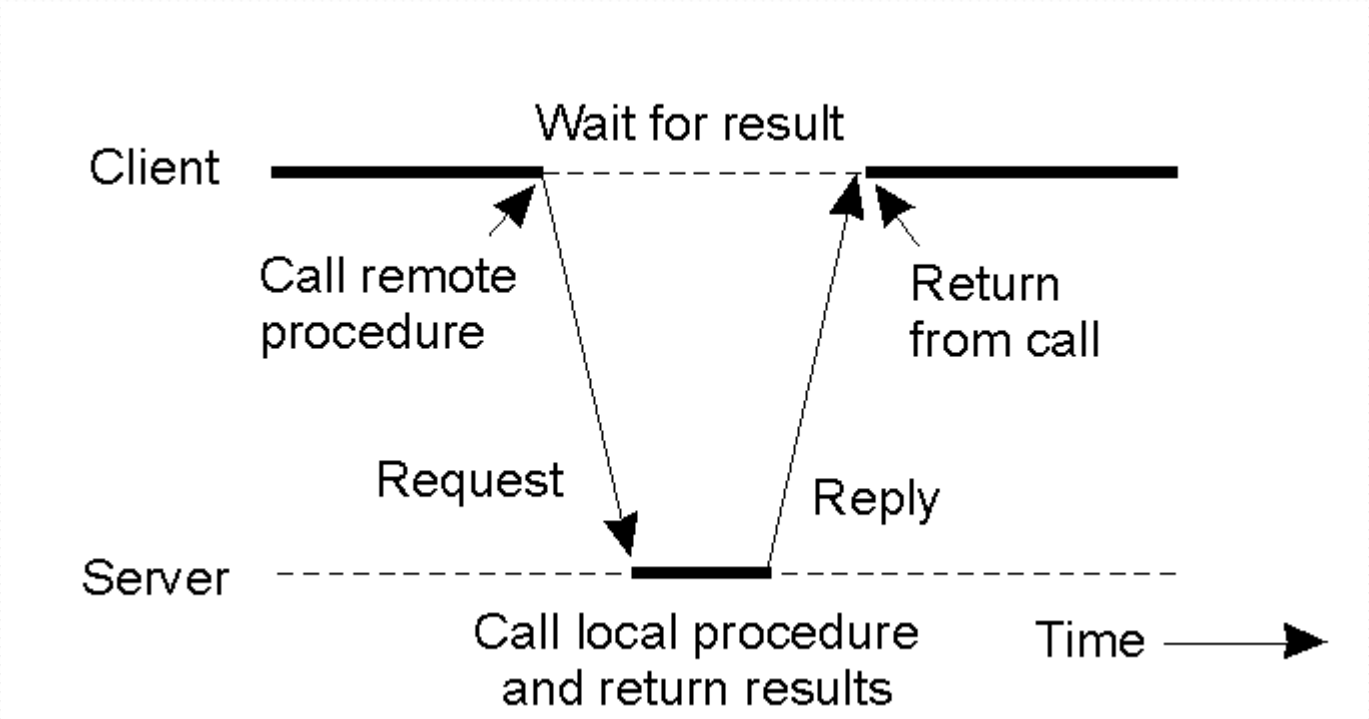
Chamada de Procedimento Convencional



a) Passagem de parâmetro chamada local: pilha antes da chamada
b) pilha enquanto procedimento está ativo

Stubs Cliente e Servidor

⌘ Princípio do RPC entre um programa cliente e servidor.



Passos em RPC

1. Cliente chama stub cliente

1. Stub client constrói mensagem, chama SO local
    1. SO cliente envia mensagem para SO remoto
    2. SO remoto entrega mensagem para stub servidor

1. Stub servidor retira parametros, chama servidor

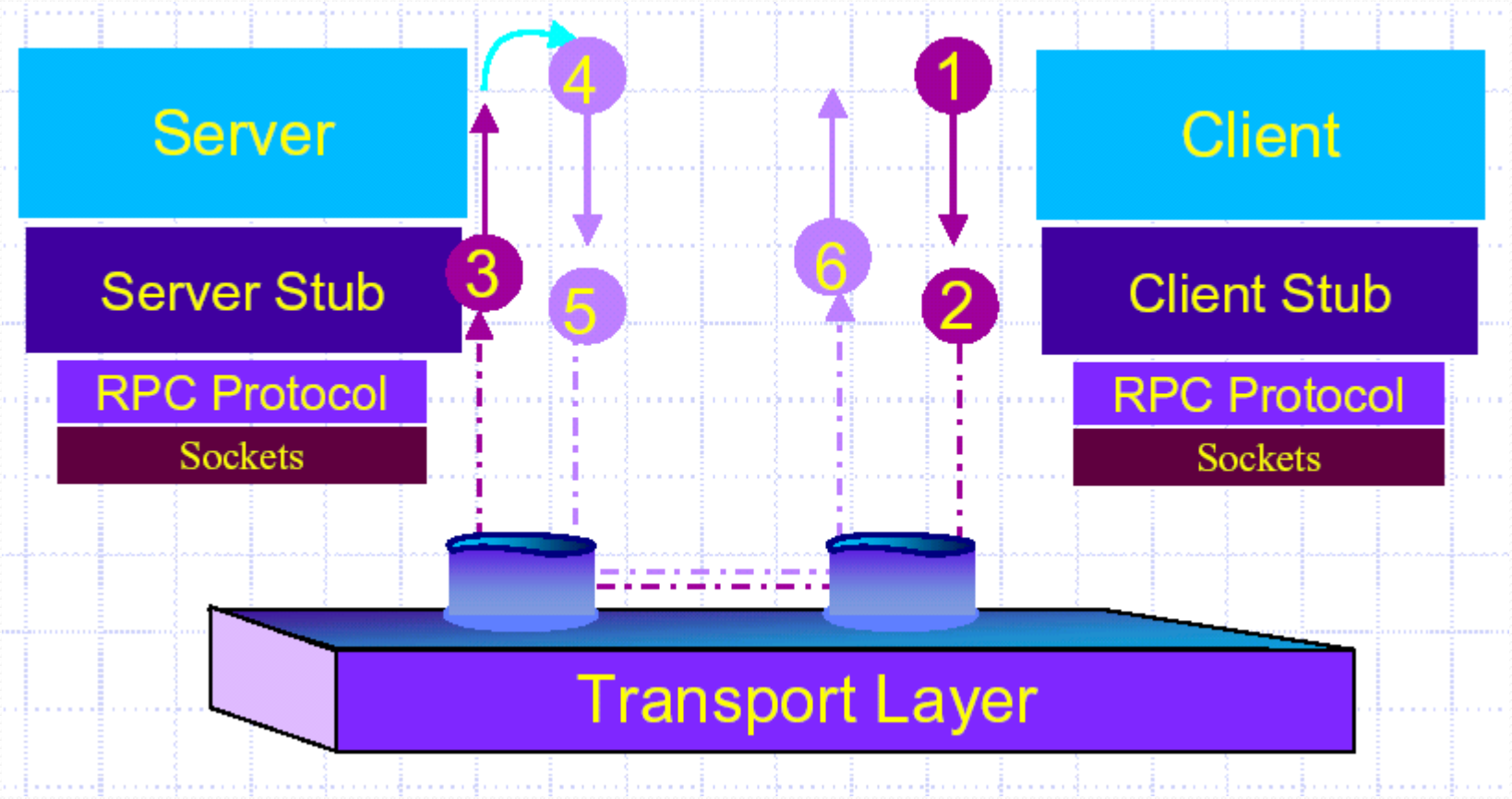
Passos em RPC

4. Servidor realiza trabalho, retorna resultado para stub

5. Stub servidor coloca na mensagem, chama SO local
   - SO servidor envia mensagem para SO cliente
   - SO cliente entrega mensagem para stub cliente

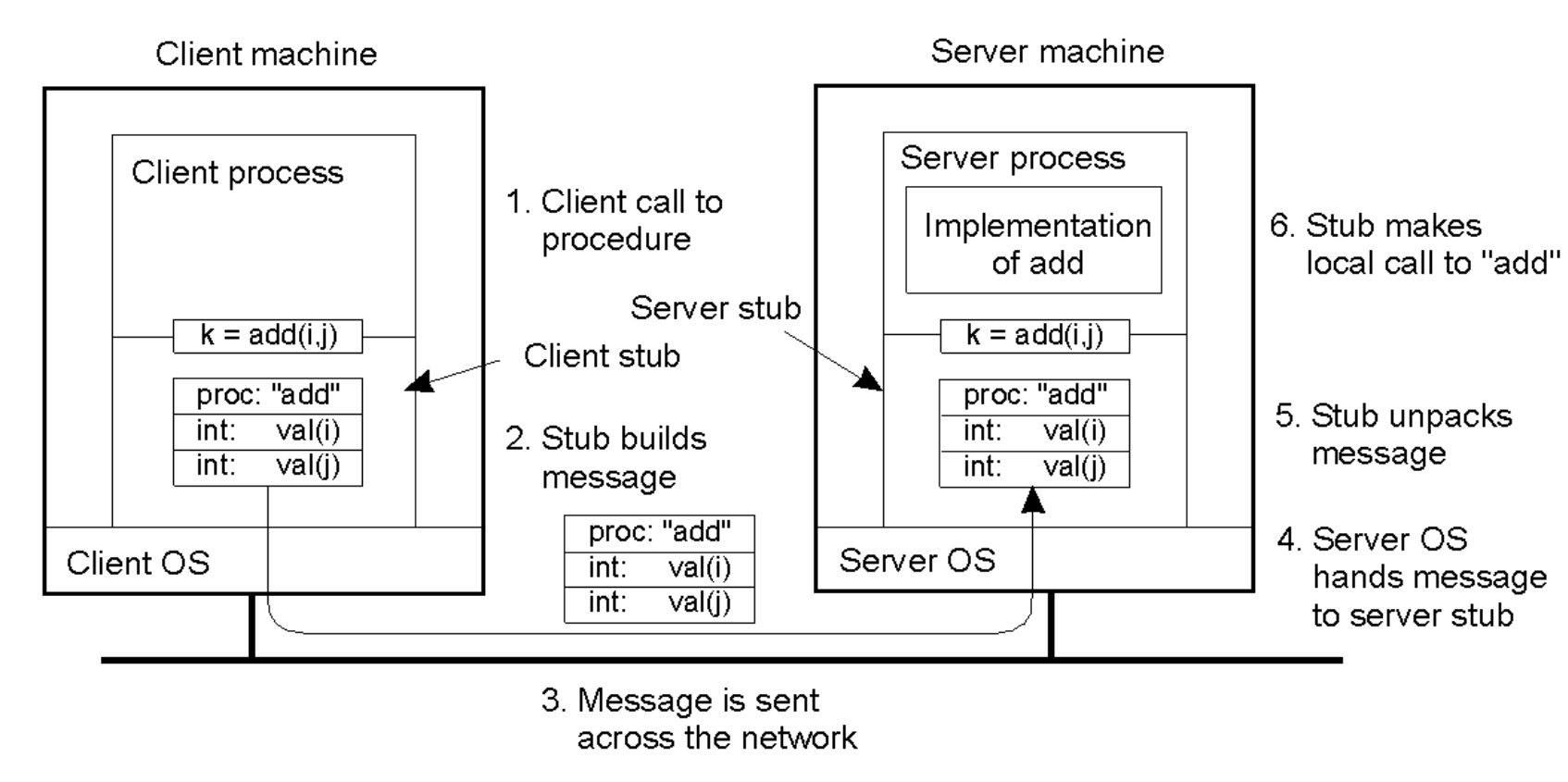
6. Stub retira resultado, retorna para cliente

Passos em RPC



Passando Parâmetros (1)

- Trocando parâmetros em RPC



Passando Parâmetros (2)

a) mensagem original no Pentium
b) mensagem depois de recebida em SPARC
c) mensagem depois de invertida

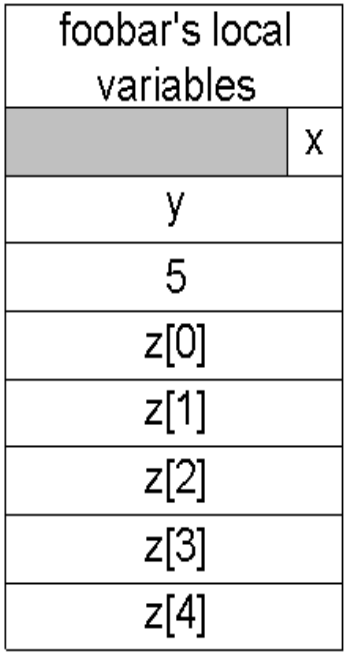
Especificação de parâmetros e geração de stubs

a) Procedimento

b) Mensagem correspondente

```
foobar( char x; float y; int z[5] )
{
   ....
}
```

(a)



(b)

Chamada Remota de Procedimento (RPC)

Parâmetros

A escolha da semântica de passagem de parâmetros é crucial para o projeto de um mecanismo de RPC :

- ***Call-by-Value*** : cópia dos valores na mensagem

- ***Call-by-Reference*** , Pointers: cópia da estrutura de dados (array) na mensagem e refaz na volta, no cliente (call-by-copy/restore).

- Passagem de Pointers é um problema

- proibir

- copiar só quando necessário

Chamada Remota de Procedimento (RPC)

Especificação de um servidor de arquivos

Specification of file_server *version* 2.0

long read (*in char* fname[n_size], *out char* buffer [b_size],
in long bytes, *in long* position);

long write(*in char* fname[n_size], *in char* buffer [b_size],
in long bytes, *in long* position);

int create(*in char* fname[n_size], *in int* mode);

int delete(*in char* fname[n_size]);

end_specification;

Chamada Remota de Procedimento (RPC)

Modelos Estendidos

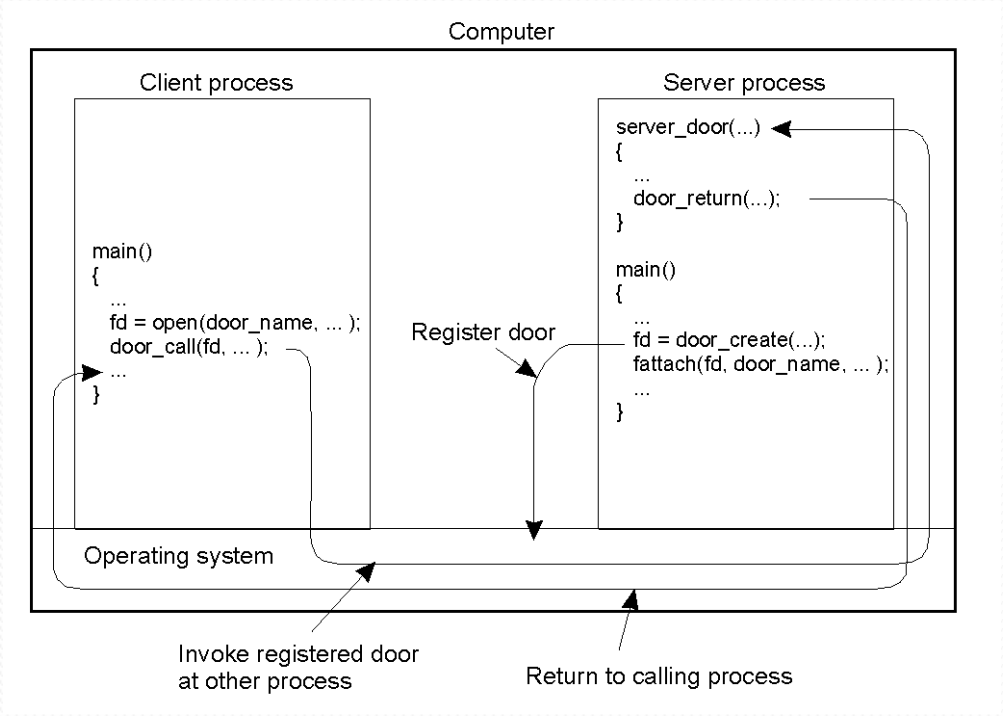
RPC se tornou um padrão de fato para comunicação em sistemas distribuídos. Extensões ao modelo original foram propostas para solucionar alguns problemas:

- RPC leve

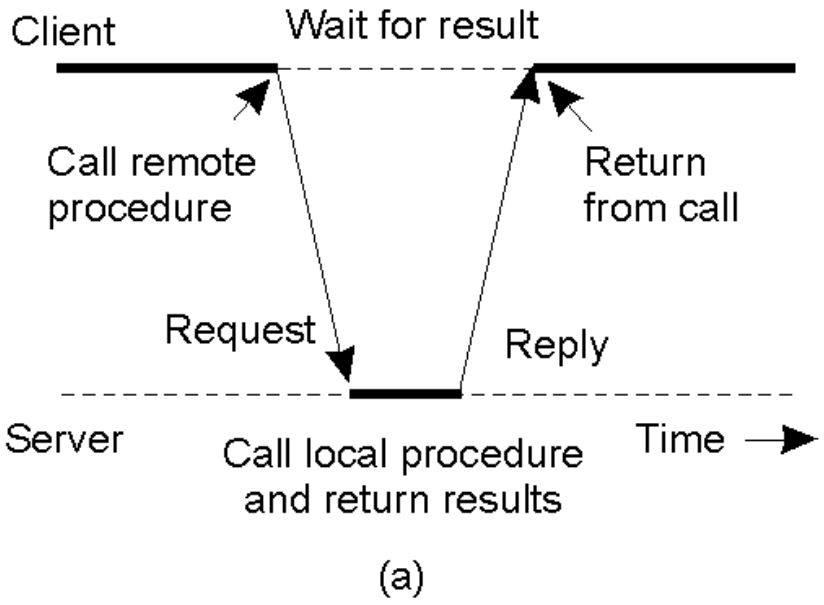
- RPC Assíncrona

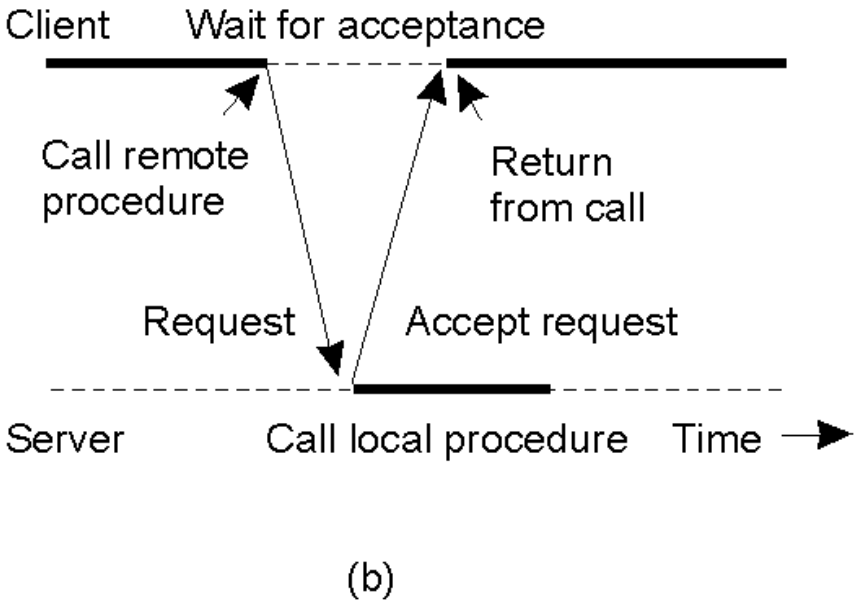
Doors

- O principio de uso de ***doors*** como mecanismo de IPC.



RPC Assíncrona (1)

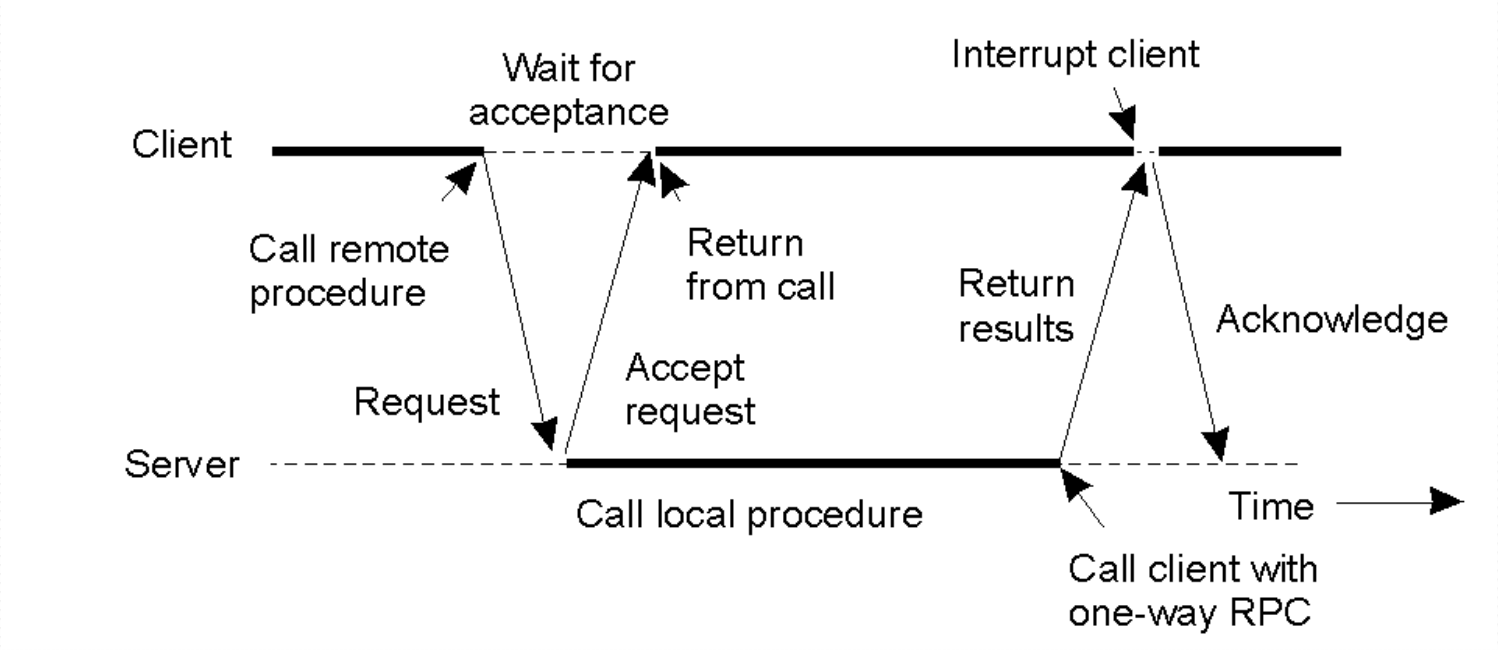




a) interconexão entre cliente e servidor RPC tradicional
b) interação usando RPC assíncrona

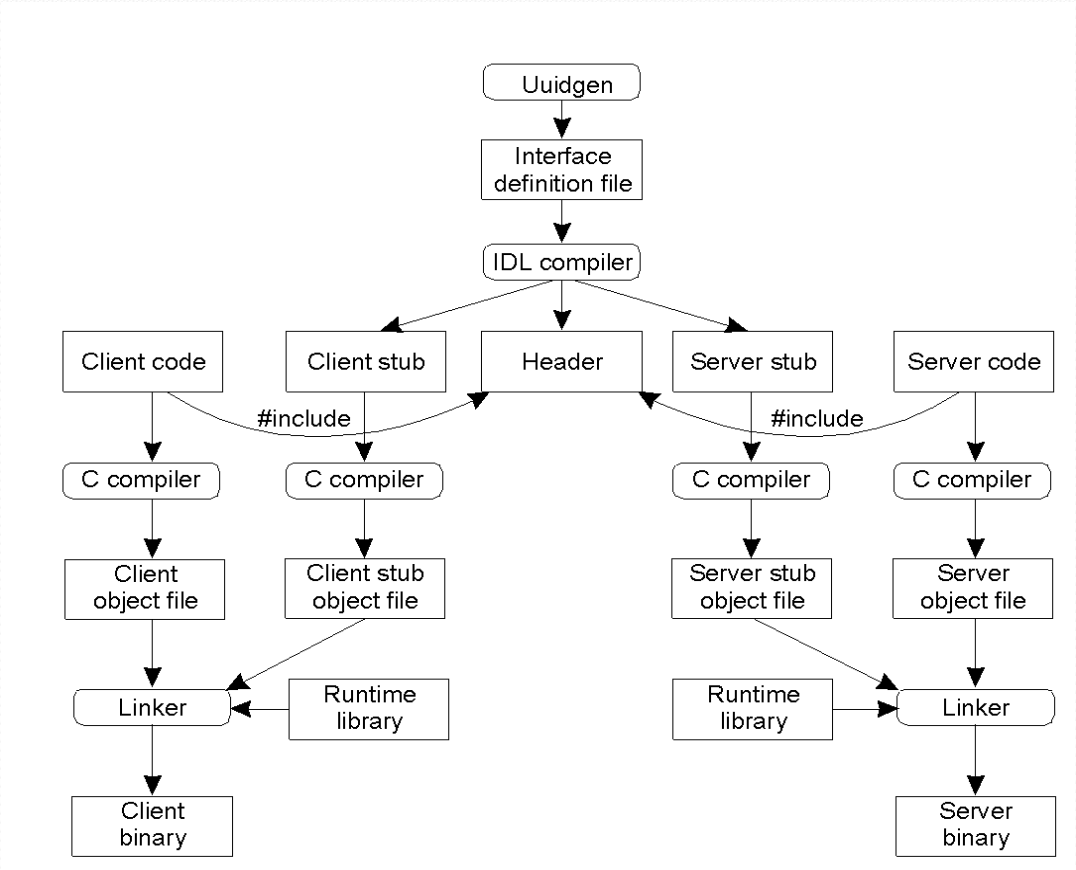
RPC Assíncrona (2)

- cliente e servidor interagindo com RPC assíncrona



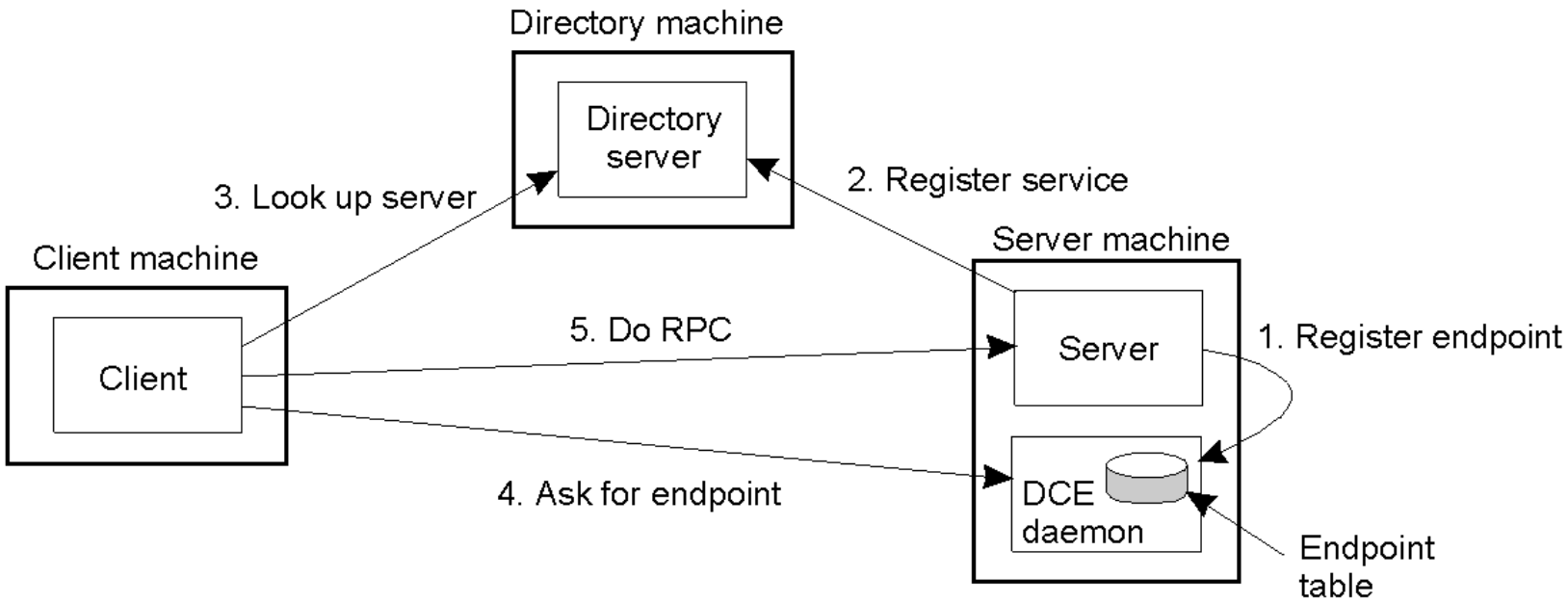
Programa Cliente / Servidor

- Passos escrevendo cliente e servidor em DCE RPC.



Ligação Cliente Servidor

- Cliente-Servidor binding em DCE.



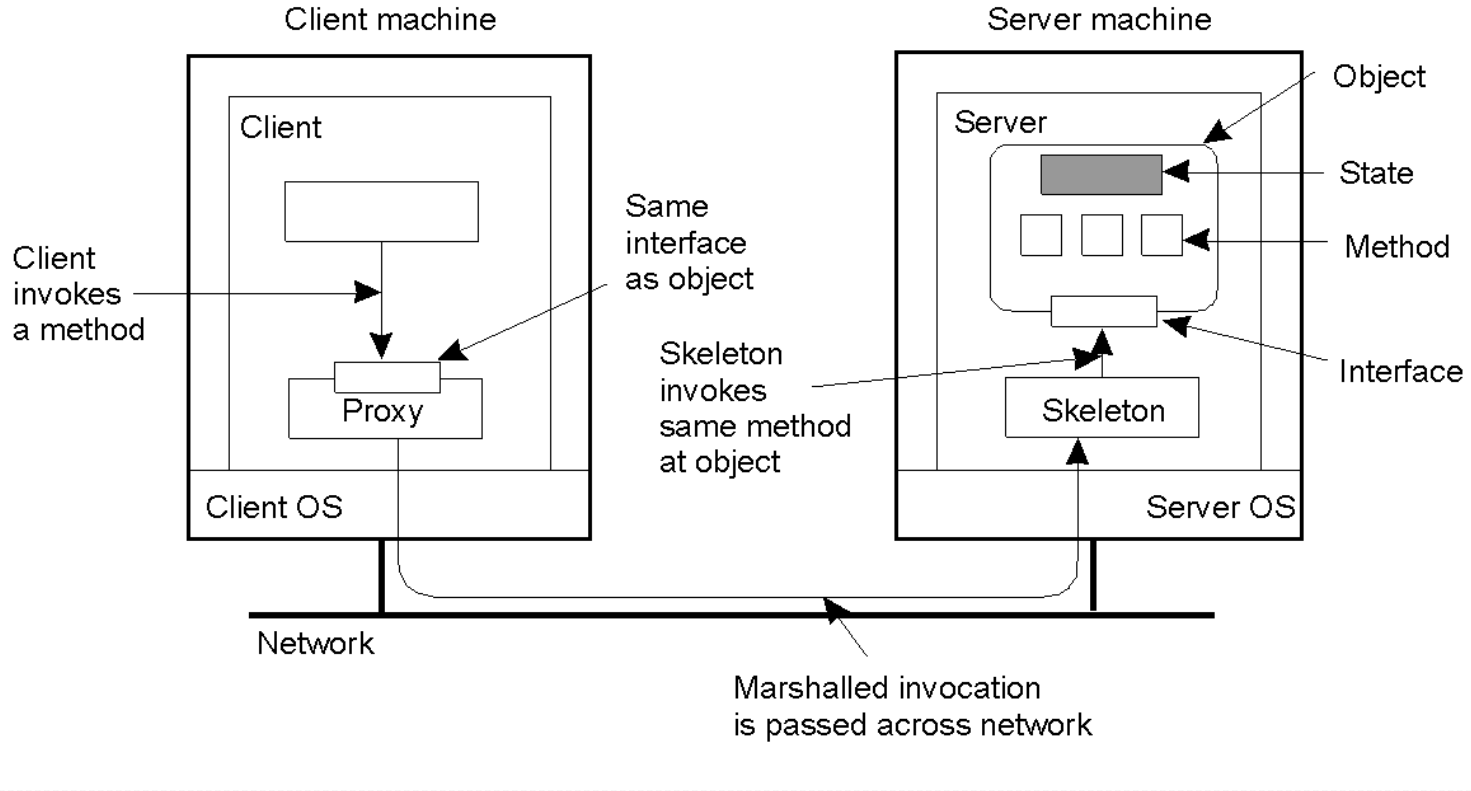
Invocação Remota de Métodos

RPC tornando-se padrão de comunicação e tecnologia baseada em objetos apresentando aspectos importantes com relação a adaptabilidade, porque não aplicar o princípio RPC em objetos.

- encapsulamento de operações e dados (objetos)
- operações (métodos) acessadas via interfaces
- objeto servidor (coleção de objetos)
- stub cliente (proxy) implementa interface
- stub servidor (skeleton)

Objetos Distribuídos

- Organização comum de um objeto remoto com proxy no lado cliente.



Objetos Distribuídos

- Objetos tempo de compilação: objetos nível de linguagem, a partir dos quais proxy e skeleton são automaticamente gerados.
- Objetos tempo de execução: implementados em qualquer linguagem, necessitam utilização de um adaptador de objeto que faz com que a implementação pareça como um objeto.

Objetos Distribuídos

- Objeto transiente: existe apenas em virtude de um servidor, se o servidor termina o objeto também.

- Objeto persistente: existe independente de um servidor, se o servidor termina o estado do objeto e código permanecem (passivos) no disco.

Ligação Cliente-Objeto

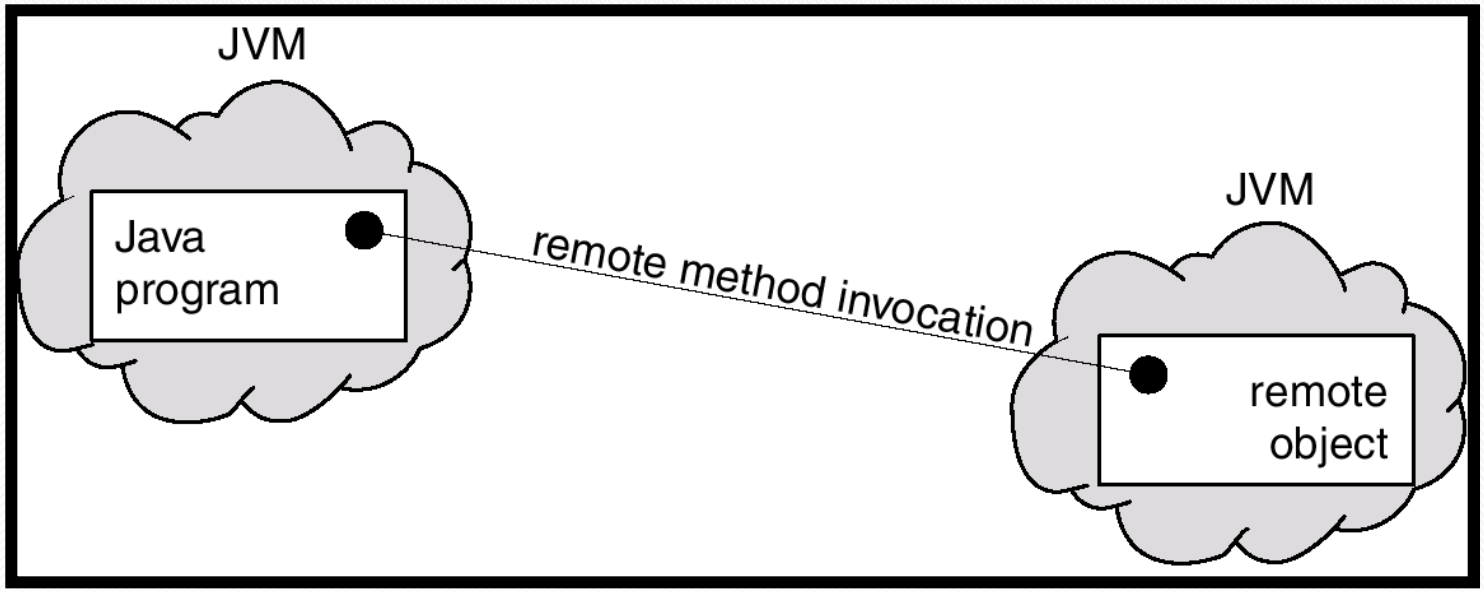
- Referência a Objeto : ter uma referência ao objeto permite um cliente se ligar a um objeto:
  - Referência denota servidor, objeto e protocolo de comunicação
  - Cliente carrega código stub associado
  - Stub é instanciado e inicializado para objeto específico

Ligação Cliente-Objeto

- Formas de Ligação:
  - Implícito: invoca métodos diretamente no objeto referenciado.
  - Explícito: cliente deve primeiro explicitamente ligar-se ao objeto antes de invocá-lo.

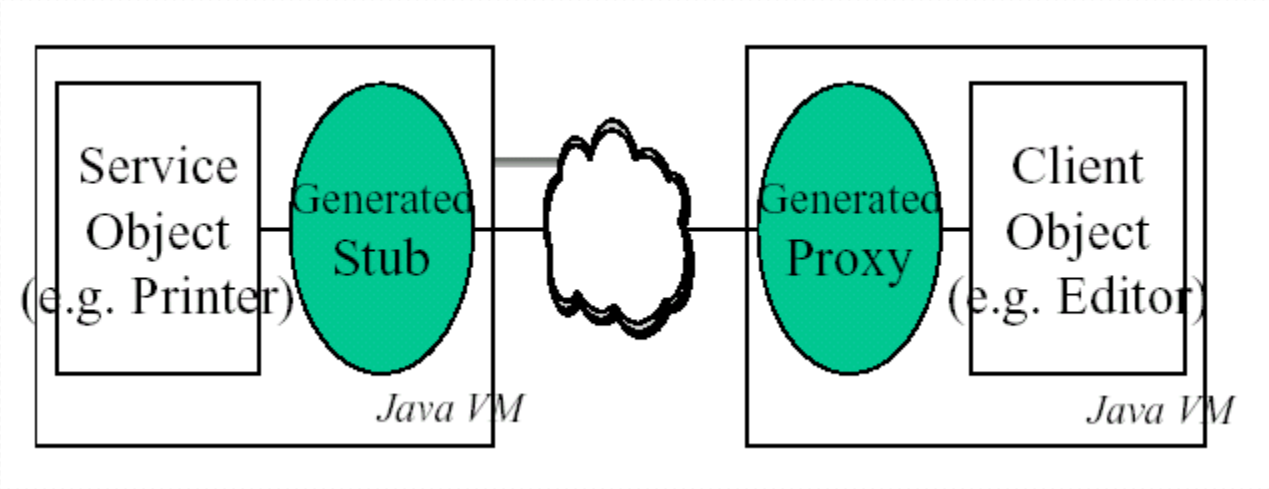
Remote Method Invocation

- Remote Method Invocation (RMI) é um mecanismo próprio de Java, similar a RPC.
- RMI permite um programa Java em uma máquina invocar um método remoto em um objeto remoto.

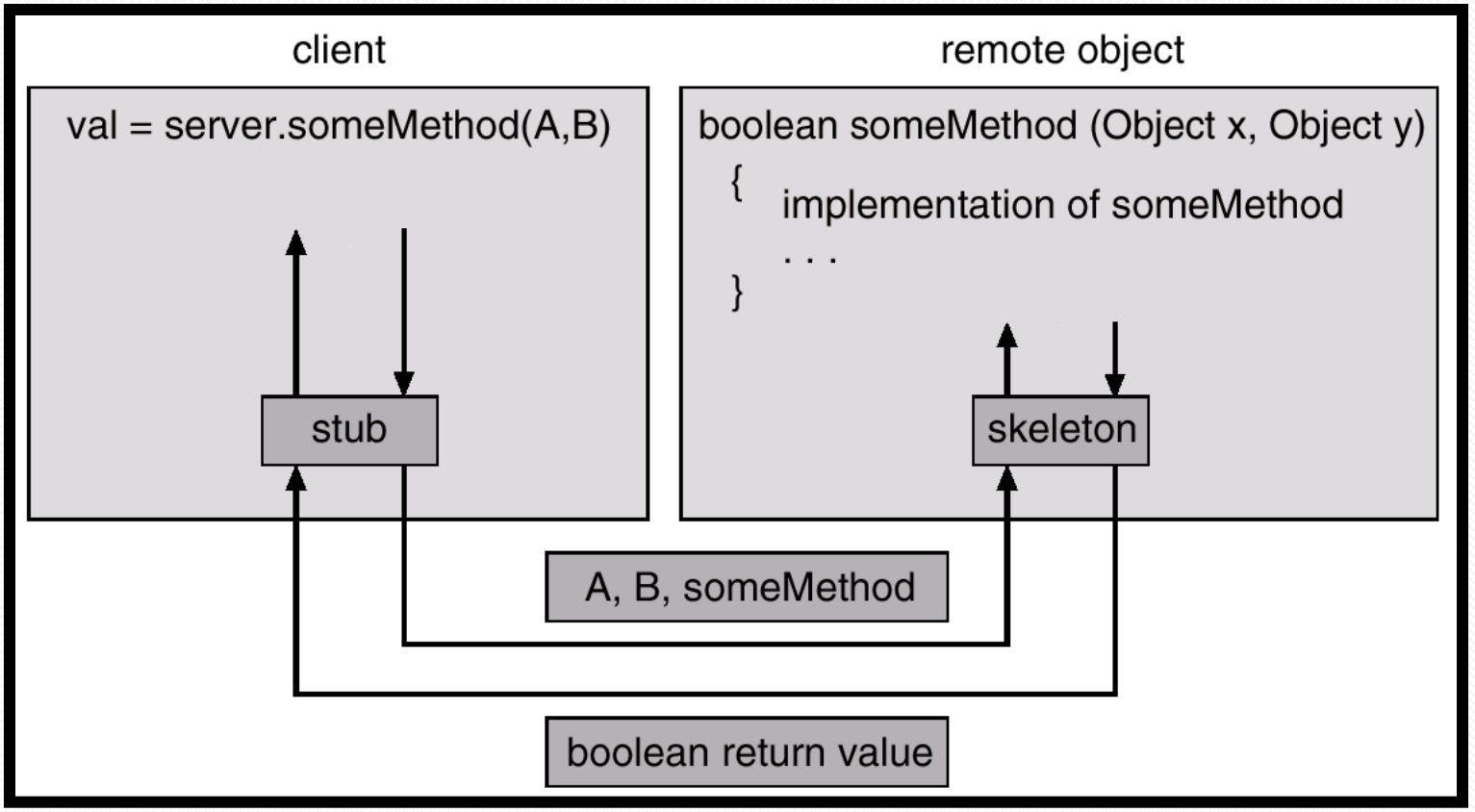


Remote Method Invocation

- Remote Method Invocation (RMI) é o mecanismo Java, que gera as classes proxy de forma automática.
- O usuário codifica os objetos cliente e serviço.
- O compilador RMI gera o código responsável pela comunicação na rede.



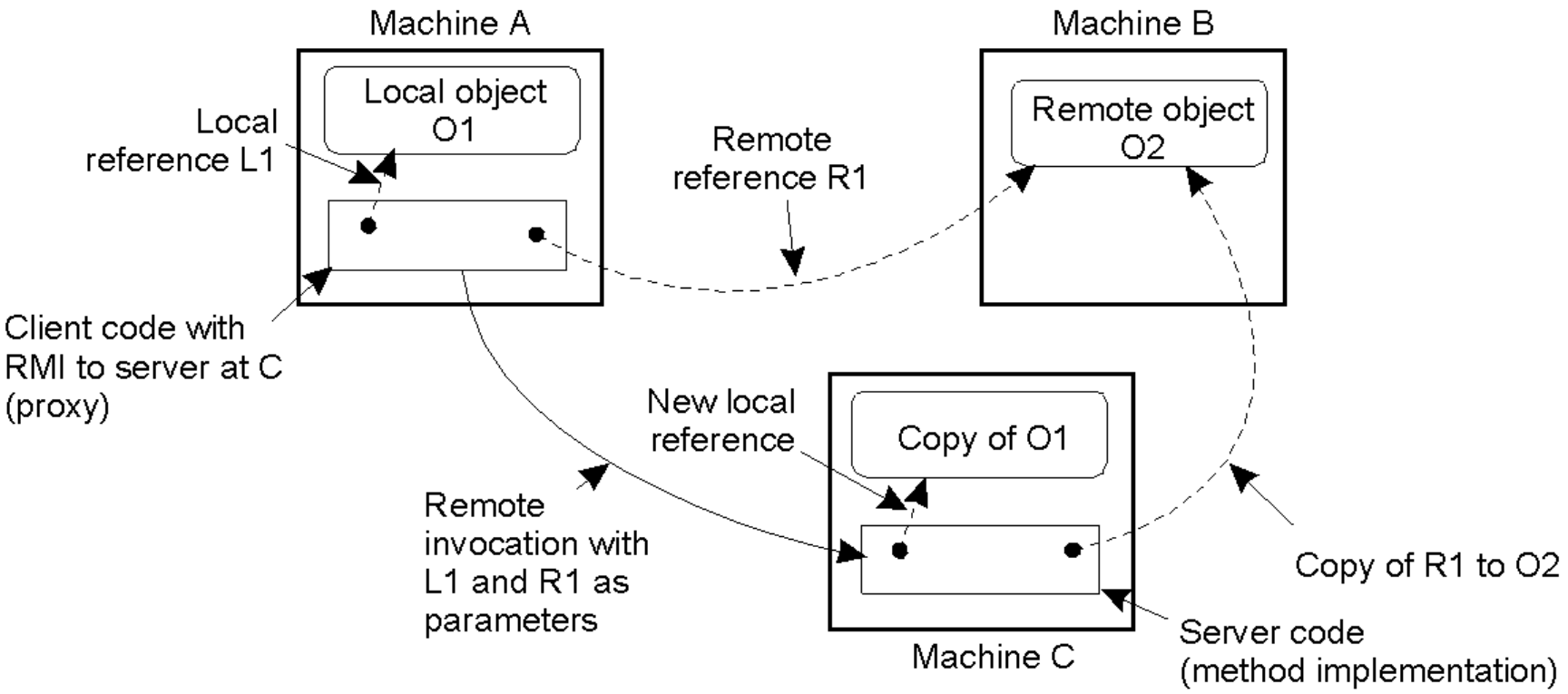
Marshalling dos Parâmetros



[http://java.sun.com/docs/books/tutorial/rmi/overview.html]

Passagem de Parâmetros

- Passando um objeto por referência ou por valor.



Objetos Distribuídos

Na abordagem de ***objetos distribuídos*** a unidade de distribuição são objetos.

Os clientes importam ***algo***, que permite a um cliente acessar um objeto remoto como se o mesmo fizesse parte do programa

Este acesso é semelhante ao efetuado através dos procedimentos do RPC e RMI.

Objetos Distribuídos

Os sistemas de objetos distribuídos permitem ainda que serviços tais como:

- descoberta: permite que clientes localizem objetos que eles necessitam ;

- segurança;

- confiabilidade.

Objetos Distribuídos

Exemplos de tecnologias de objetos distribuídos são:

- ***Corba;***
- ***DCOM;***
- ***JAVA/RMI***

Objetos Distribuídos

Uma breve descrição dessas tecnologias de objetos distribuídos pode ser entendida como:

- ***Corba*** (*Common Object Request Broker Architecture)* – modelo definido por um consórcio da indústria, conhecido como *Object Management Group* [www.omg.org]

  O Corba é disponível para a maioria dos sistema operacionais;

Objetos Distribuídos

- ***DCOM*** (Distributed Component Object Model) – desenvolvido pela Microsoft, também utilizado por outras empresas, foi construído no topo do DCE-RPC, interagindo com a tecnologia COM (*Component Object Model*) [www.microsoft.com]*;*

Objetos Distribuídos

O *Distributed Component Object Model* permite que componentes se comuniquem considerando os limites de uma rede, pois o COM foi projeto para os limites locais de processos em uma máquina.

Objetos Distribuídos

O DCOM utiliza o mecanismo de RPC para que ocorra de uma forma transparente o *envio* e *recebimento* entre componentes COM, ou seja entre clientes e servidores.

O DCOM foi disponibilizado em 1995 na versão original do sistema operacional Windows NT4.

Objetos Distribuídos

A abordagem da tecnologia Microsoft COM foi projetada visando que componentes de software utilizando a ***família*** de sistemas operacionais Windows pudesse se comunicar.

Objetos Distribuídos

O COM é usado por desenvolvedores para criar componentes de software reusáveis, agregando componentes para a construção de aplicações. Assim, também, tirando proveito dos serviços do Windows.

Objetos Distribuídos

A família da tecnologia COM da Microsoft inclui:

- COM+;

- DCOM;

- ActiveX® Controls.

Conteúdo Programático

. Objetos distribuídos

. Componentes

Contextualização

. CORBA

. Barramento de objetos CORBA

. Adaptador básico de objetos

. Linguagem de definição de interfaces

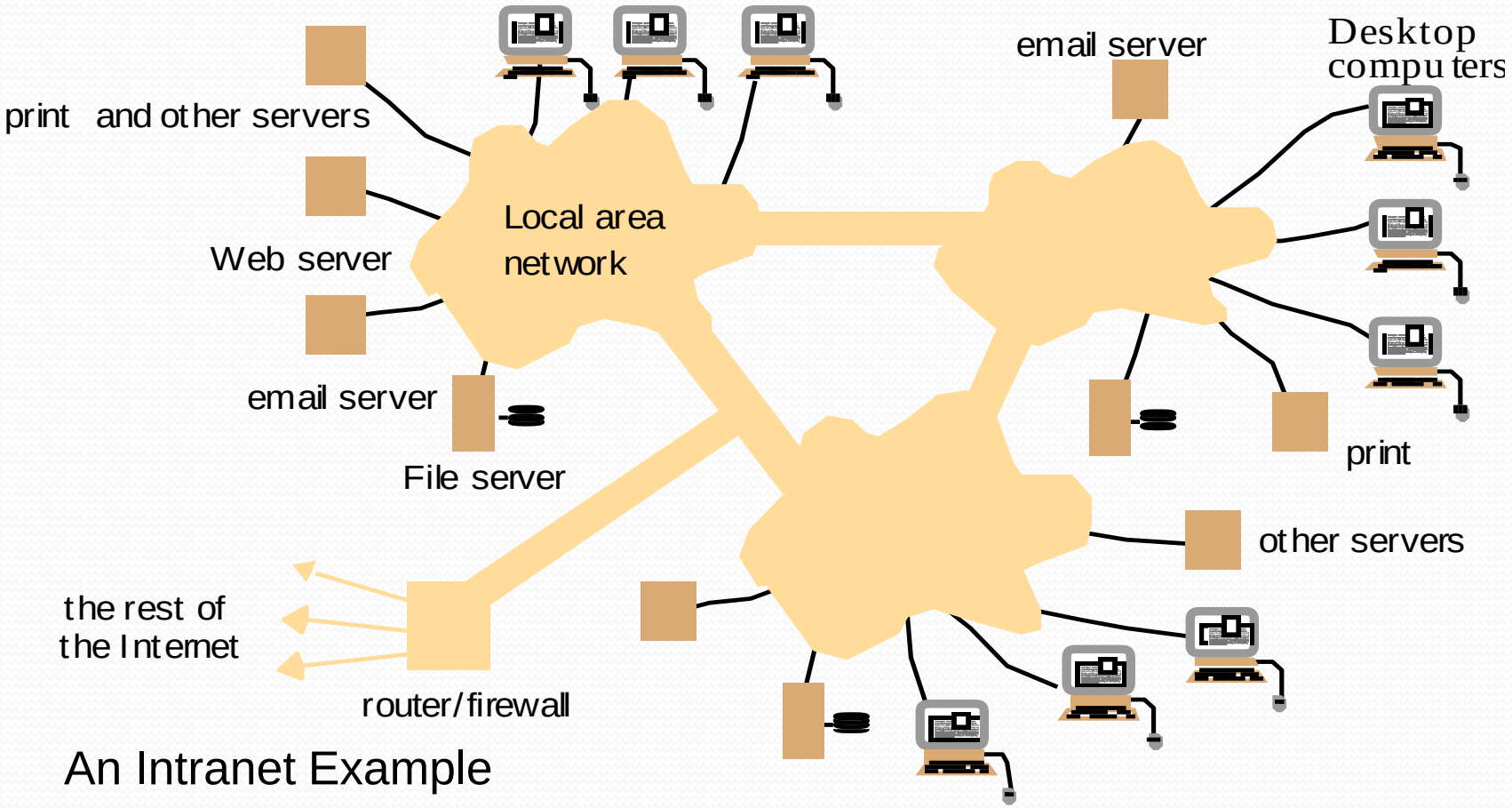
Contextualização

Os *sistemas distribuídos* são caracterizados por um conjunto de computadores interligados através de uma rede, provendo uma

imagem global única

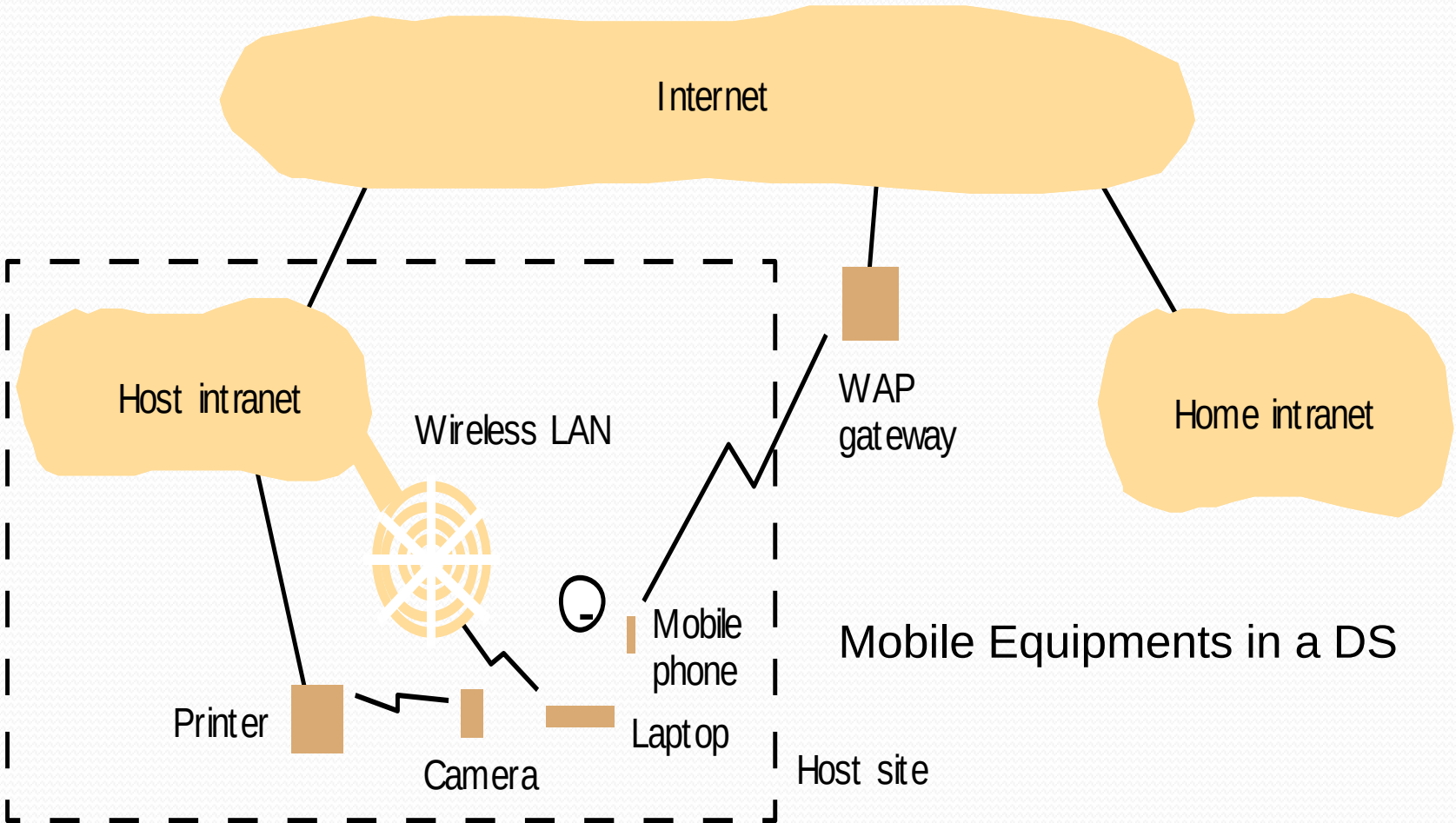
para o compartilhamento de recursos e serviços.

Sistemas Distribuídos - Características



An Intranet Example

Sistemas Distribuídos - Características



Contextualização

Os *sistemas distribuídos* são diferentes das *redes de computadores*, pois as redes não são focadas ao provimento de uma imagem única.

Contextualização

As abordagens mais comuns nos ambientes distribuídos são:

- Cliente-Servidor: nesta abordagem temos um nó cliente solicitando serviços para um outro nó servidor;

Contextualização

As abordagens mais comuns nos ambientes distribuídos são (cont.):

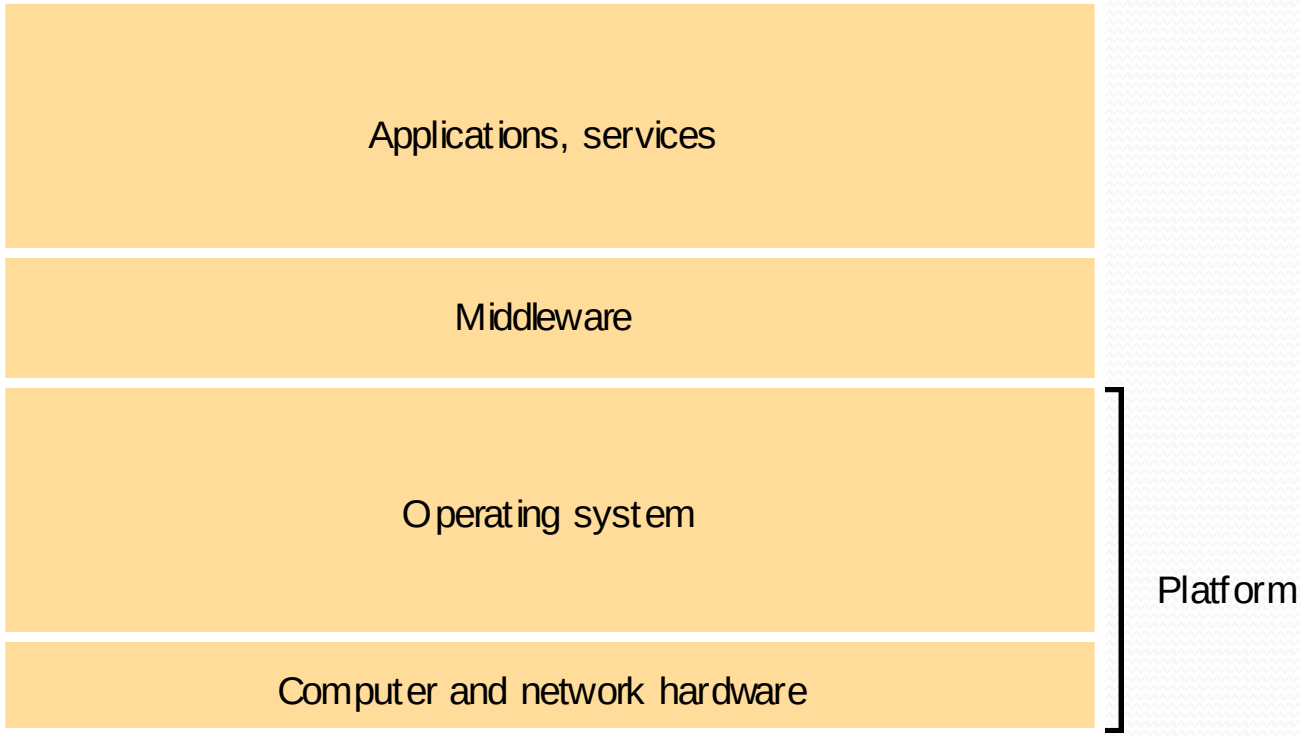
- Fim-a-Fim (Peer-to-Peer): neste paradigma temos *nós* atuando como clientes e servidores. Em um determinado momento um nó pode ser cliente e em outro instante ser um servidor.

Contextualização

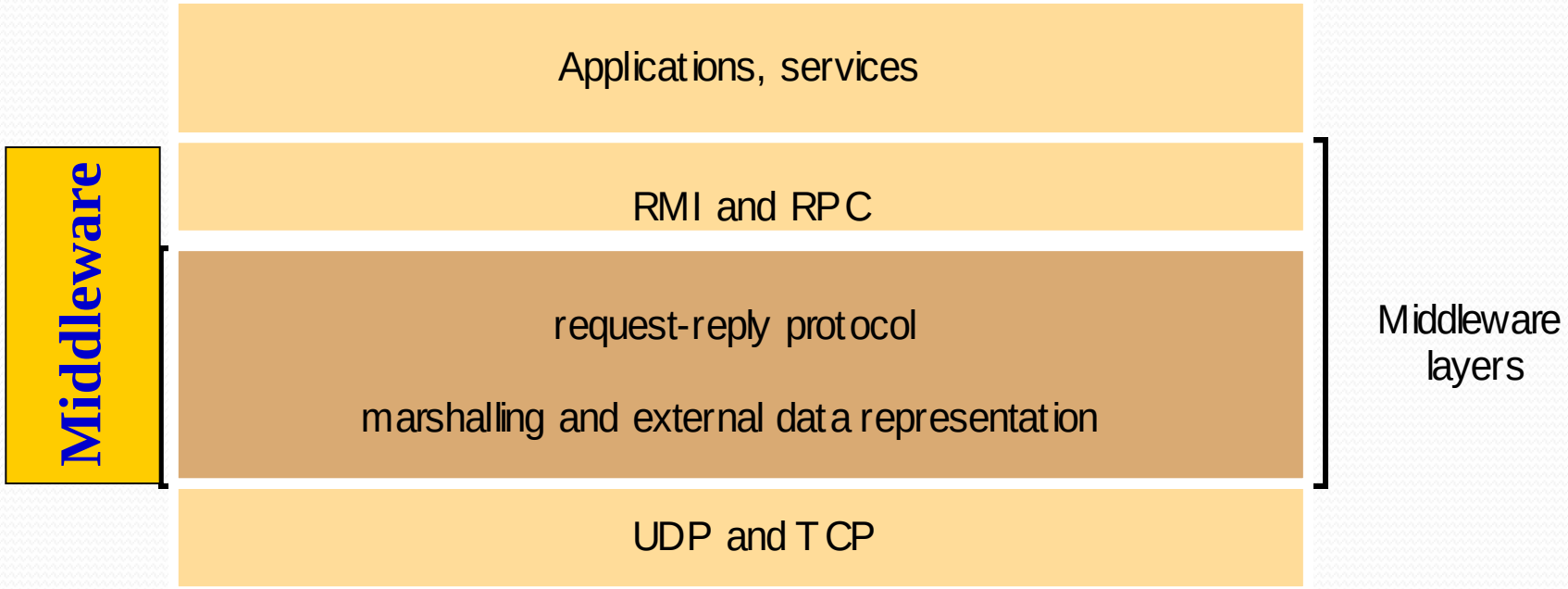
Um aspecto muito importante para prover uma imagem global única, nos sistemas distribuídos é a transparência.

Sistemas Distribuídos – Modelo

Modelo



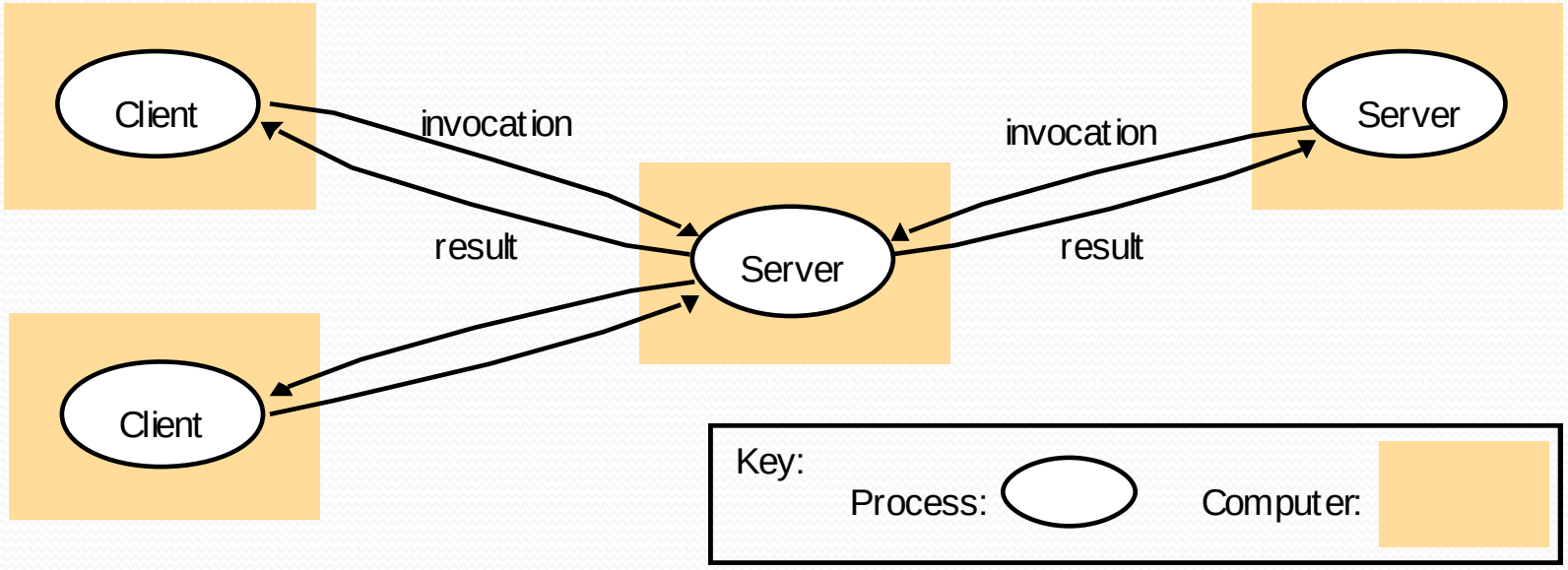
Sistemas Distribuídos



Middleware layers

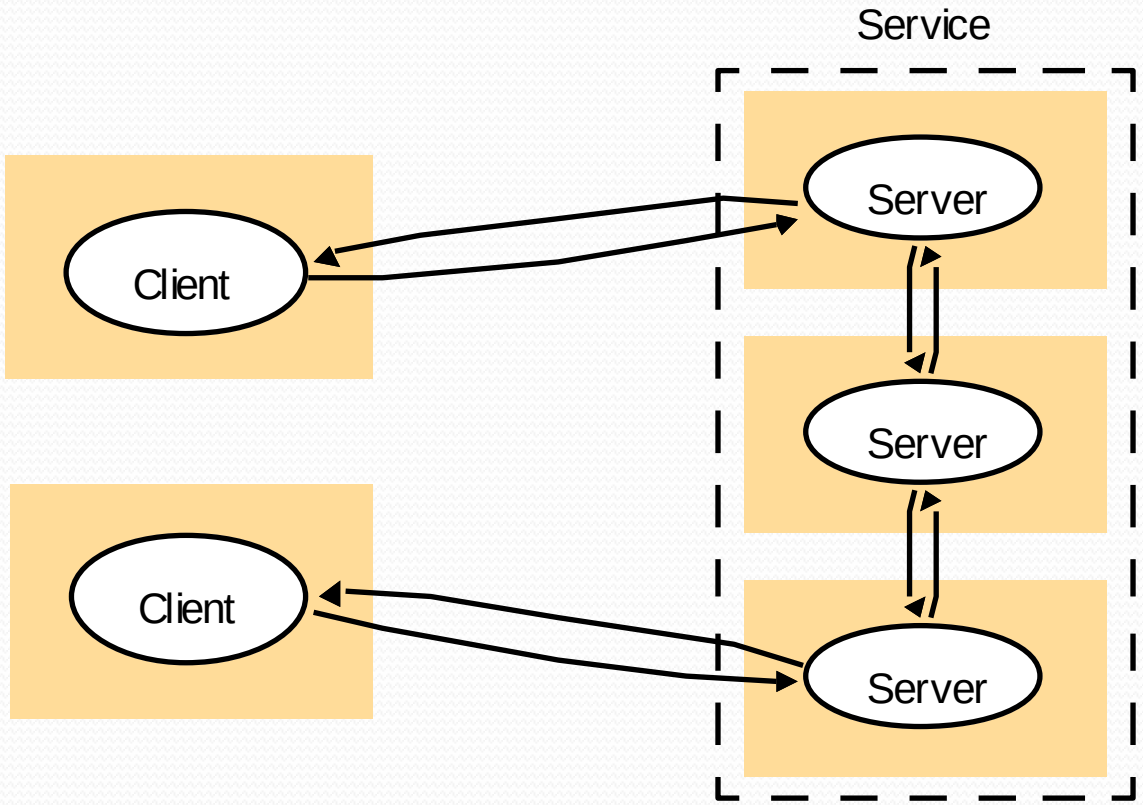
Sistemas Distribuídos

Clientes chamando servidores individuais

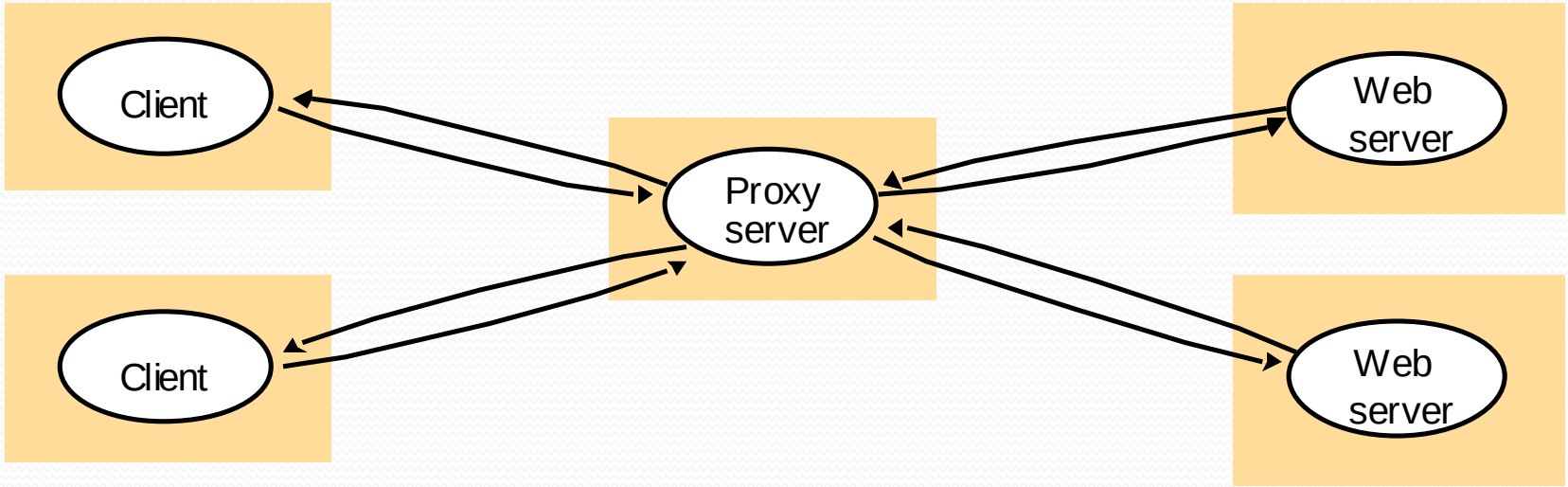


Sistemas Distribuídos

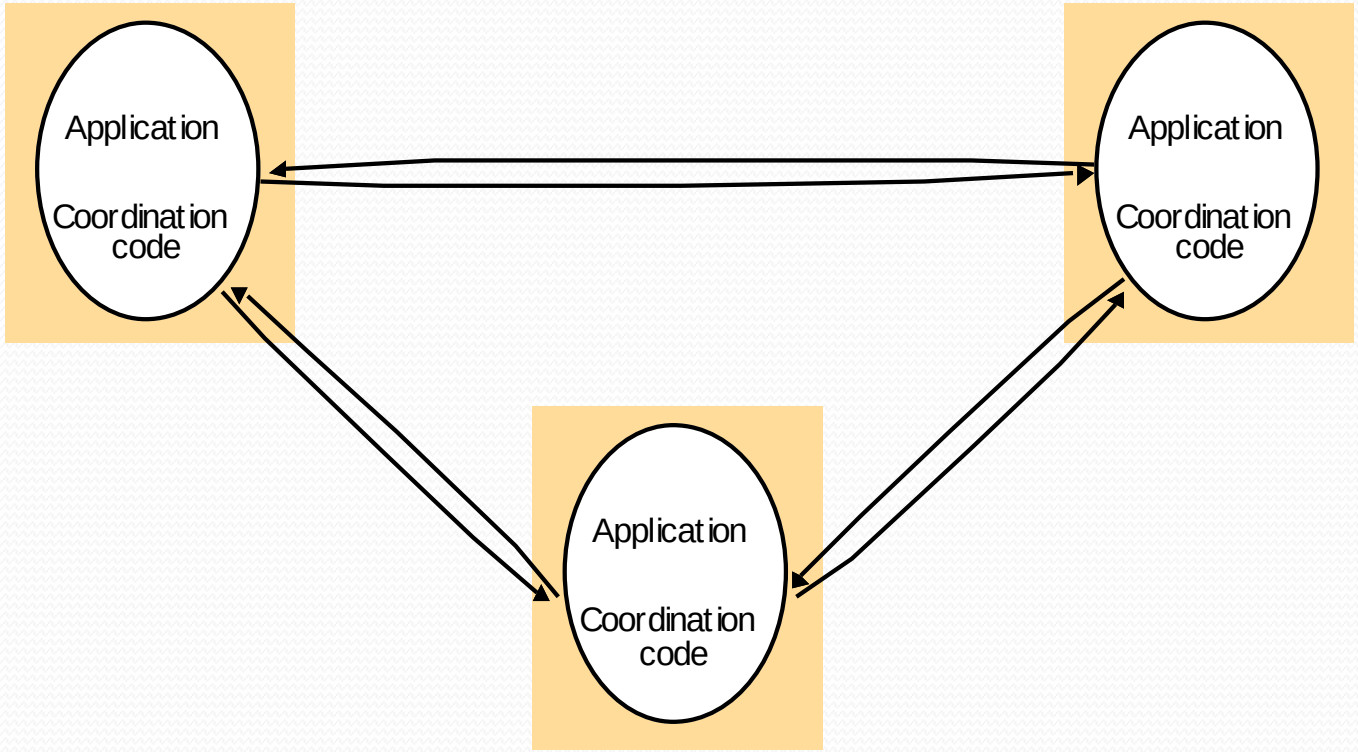
Um serviço provido por multiplos servidores



Servidor Web proxy

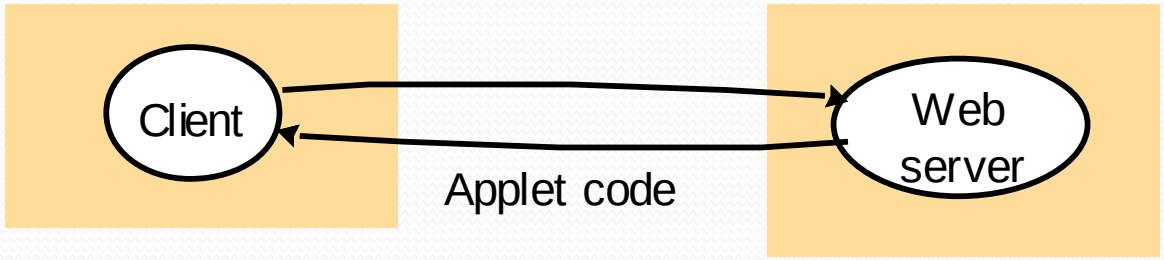


Uma aplicação distribuída baseada em processos fim-a-fim



Web applets

a) client request results in the downloading of applet code



b) client interacts with the applet

Contextualização

Os tipos de transparência considerados nos SDs, são :

- nome
- localização
- acesso

Contextualização

Os tipos de transparência considerados nos SDs, são (cont.):

- migração
- replicação
- *concorrência e paralelismo*
- falha

Contextualização

A tecnologia de software utilizada para auxiliar na gerência da complexidade de aplicações sob os ambientes heterogêneos distribuídos é chamada de *middleware*.

Exemplos de *middlewares* são o CORBA, COM/DCOM e JAVA RMI.

Conteúdo Programático

. Objetos distribuídos

. Componentes

. CORBA

. Barramento de objetos CORBA

. Adaptador básico de objetos

. Linguagem de definição de interfaces

Componentes

O grupo conhecido como OMG estabeleceu quatro componentes básicos para a formação do CORBA:

- ORB (Object Request Broker): análogo a um barramento de computador, esse componente prove um função de concentrador de comunicação entre os demais componentes;

- Object Services: prove um conjunto de funções padrões para a criação, controle e referência dos objetos.

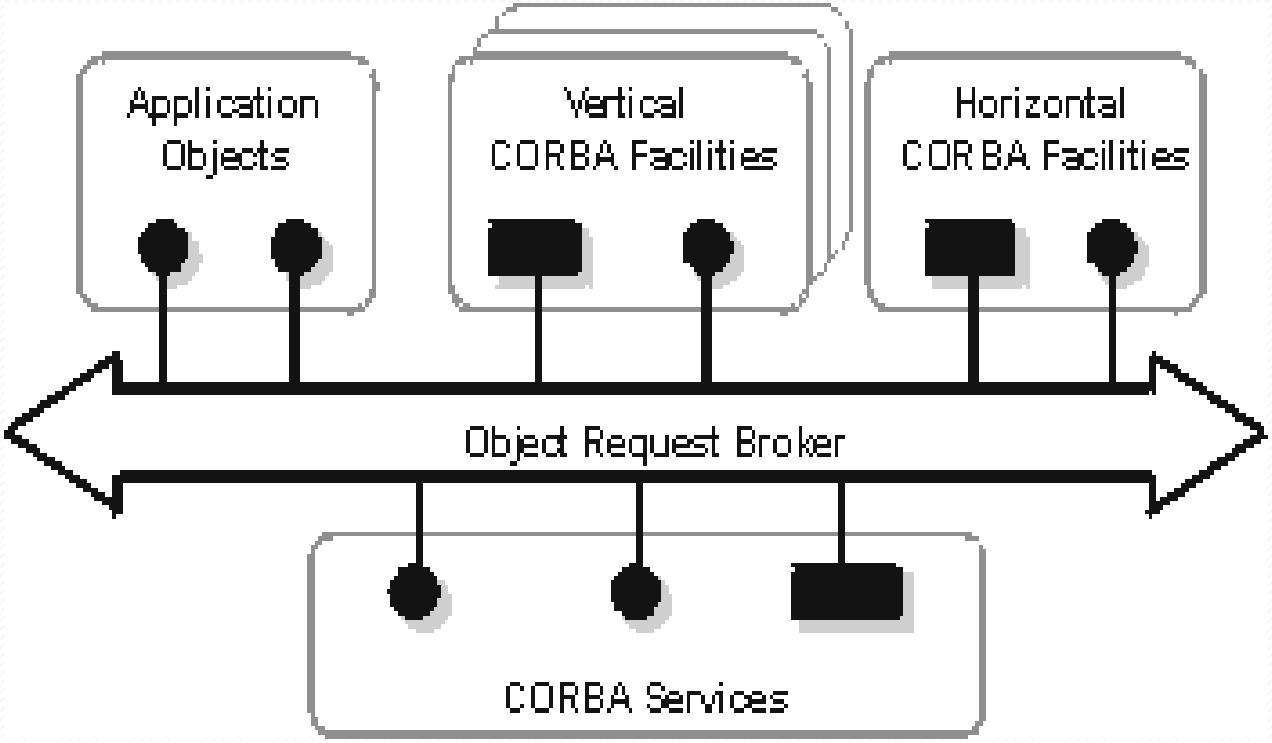
Componentes

O grupo conhecido como OMG estabeleceu quatro componentes básicos para a formação do CORBA (cont.):

- Common Facilities: prove um conjunto genérico de funções com capacidade de utilização em diferentes tipos de aplicação;

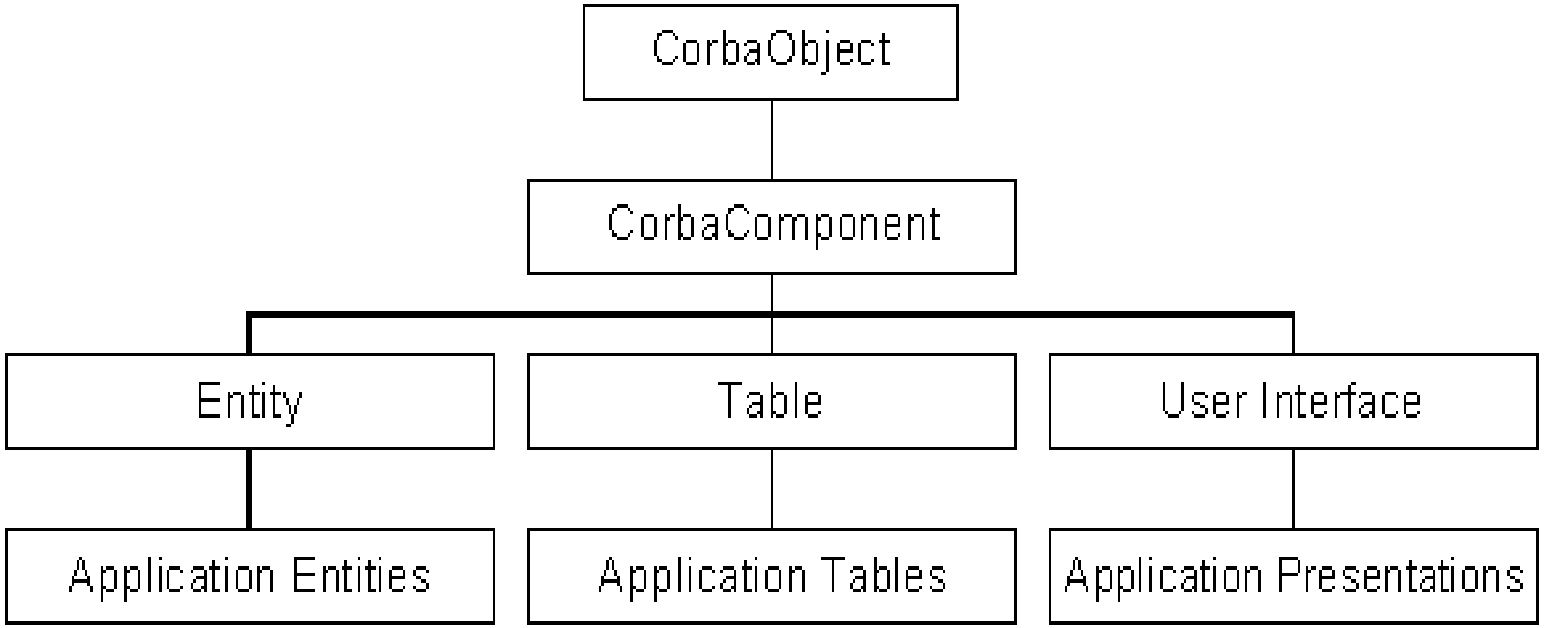
- Application Object: prove um conjunto de objetos que podem efetuar funções específicas para usuários finais. Essas aplicações são essencialmente aplicações orientadas à objetos.

Object Management Architecture (OMA)

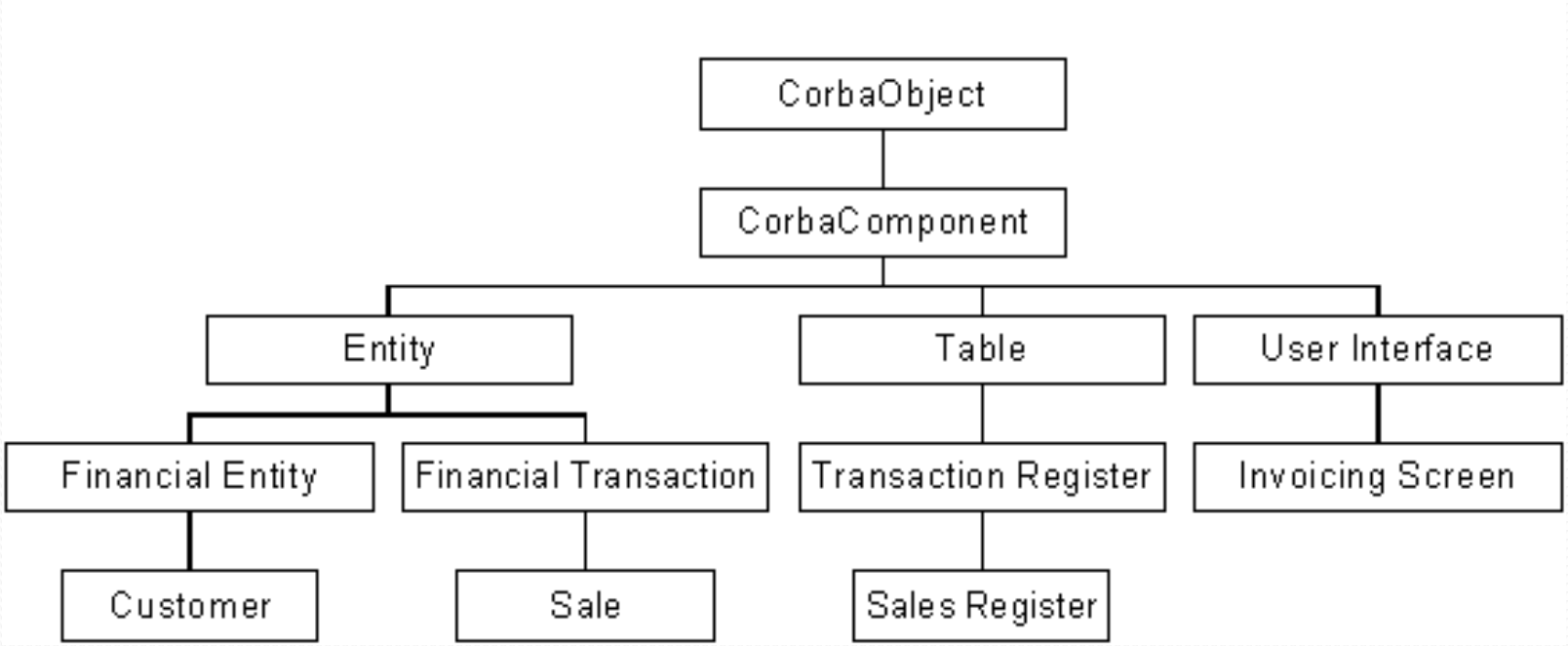


[www.omg.org]

Corba


CorbaObject
CorbaComponent
Entity
Table
User Interface
Application Entities
Application Tables
Application Presentations

Corba


CorbaObject
CorbaComponent
Entity
Table
User Interface
Financial Entity
Financial Transaction
Transaction Register
Invoicing Screen
Customer
Sale
Sales Register

Conteúdo Programático

. Objetos distribuídos

. Componentes

. CORBA

. Barramento de objetos CORBA

. Adaptador básico de objetos

. Linguagem de definição de interfaces

Corba

O modelo do *middleware* conhecido como

CORBA

(*Common Object Request Broker Architecture)* –
foi definido por um consórcio da indústria, denominado como *Object Management Group* [www.omg.org]

Corba

O *middleware* ***CORBA*** permite que:

- Aplicações comerciais de desktop convencionais possam fazer acesso a um conjunto de informações distribuídas e recursos;

- Os sistemas de dados de negócios possam ser disponibilizados como recursos de rede;

Na próxima figura é ilustrado uma chamada de um cliente no ambiente CORBA a um objeto distribuído.

Corba

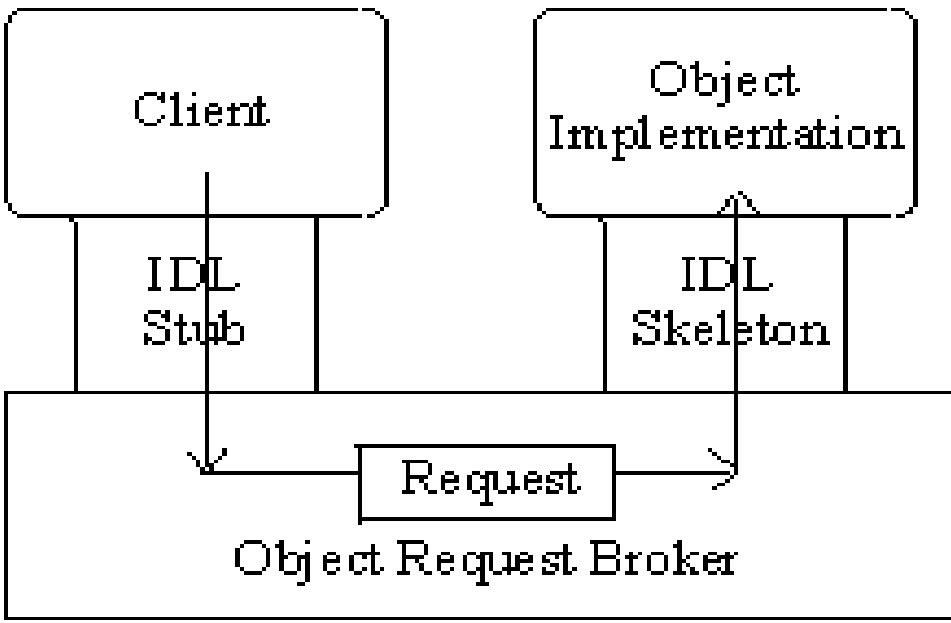


Figure 1: A request passing from client to object implementation

[www.omg.org]

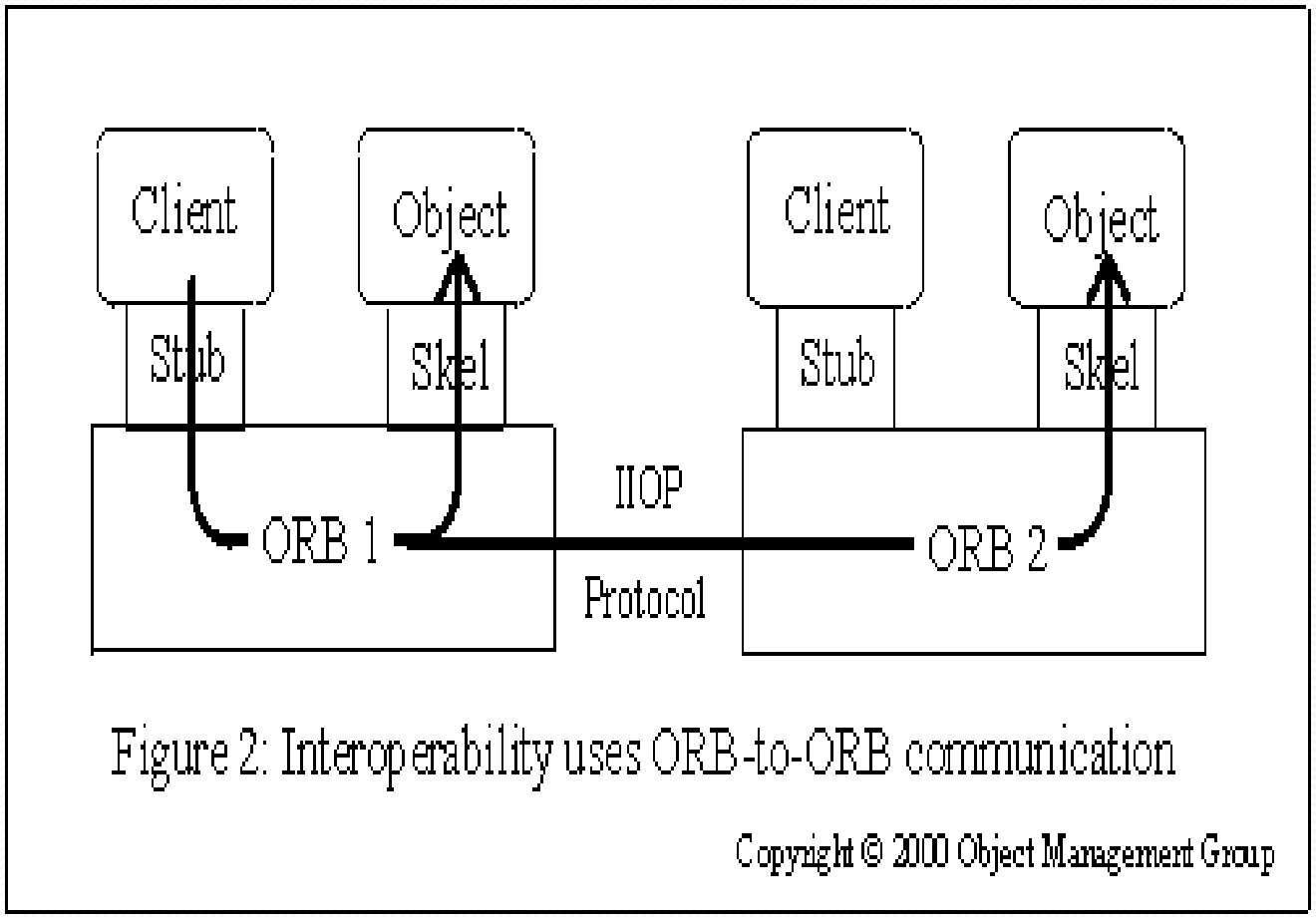
Corba

O *middleware* ***CORBA*** permite que as solicitações sejam solicitadas entre diferentes ambientes, provendo uma imagem global única de consulta para o cliente solicitante.

Na próxima figura é ilustrado esse conceito, na comunicação entre dois diferentes ORBs.

Observe que entre dois ORBs é utilizado o protocolo IIOP (Internet Inter ORB Protocol)

Corba



[www.omg.org]

Corba

O modelo ***CORBA*** foi projetado para prover uma interoperabilidade entre aplicações em sistema distribuídos heterogêneos com orientação à objeto.

Corba

A utilização de OO no projeto, analise e desenvolvimento usando ***CORBA*** permite o reuso através de sistemas.

Corba

A programação orientada à objetos simplifica o desenvolvimento de software através de abstrações de alto nível e padrões.

Exemplos de facilidades da programação OO são:

- associação entre dados e operações; e
- decomposição de interfaces e implementações

Corba

As vantagens da OO como:

- Herança;
- Encapsulamento;
- Redefinição; e
- Agregação dinâmica

estas são características existentes no CORBA

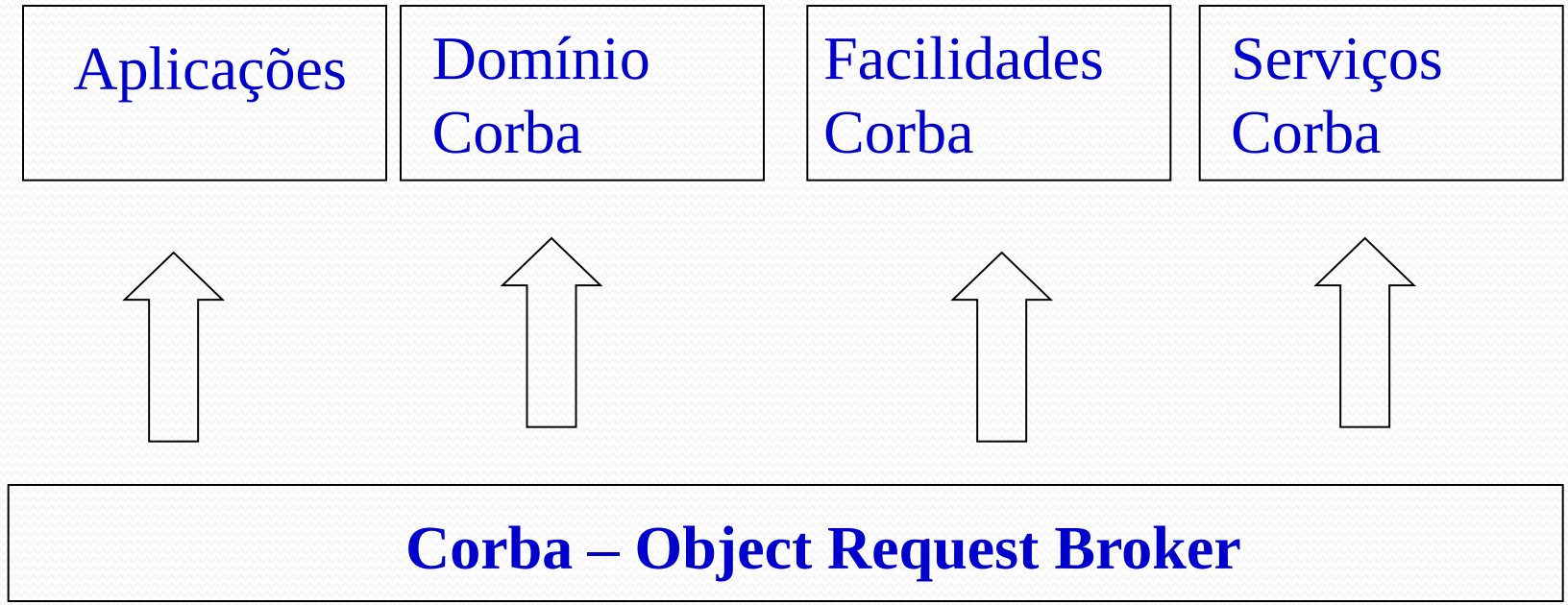
Corba

Transparência em Corba:

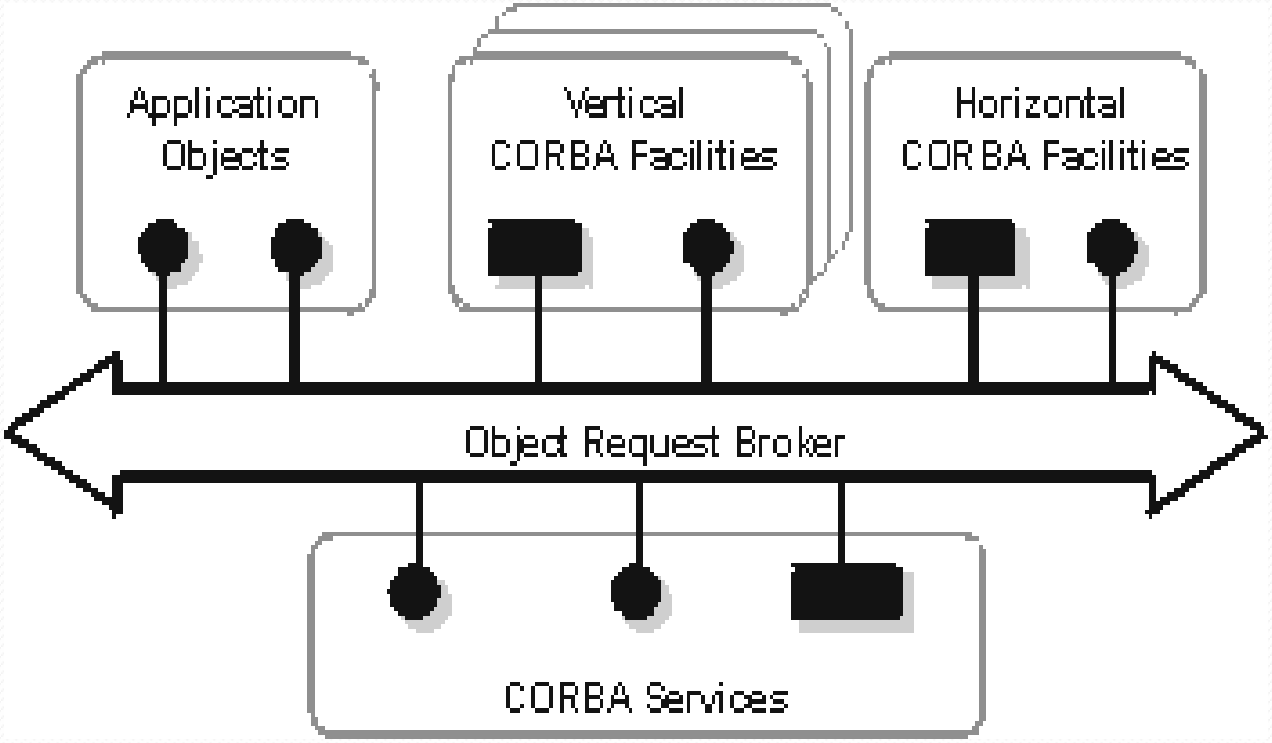
- Localização : objetos clientes podem estar localizados e acessados no cliente ou servidor remotos;
- Linguagem de programação: Java, C/C++, Ada 95, Smaltalk e Cobol;
- S.O.: Família Windows, Unix´s e outros;

Corba

Object Management Architecture (OMA)



Object Management Architecture (OMA)



[www.omg.org]

Corba

A arquitetura Corba pode ser categorizada em quatro interfaces de software:

- Object Request Broker (ORB) é o principal elemento d arquitetura. O ORB é a interface que permite um client fazer acesso a objetos distribuídos residentes em servidores.

  O cliente faz uma chamada ao ORB que localiza o objet solicitado e identifica o servidor aonde se encontra o objeto;

Corba

A arquitetura Corba pode ser categorizada em quatro interfaces de software (cont.):

O ORB faz a tradução das chamadas do cliente para o servidor. O ORB então faz o mapeamento do IDL (Interface Definition Language) para a linguagem para a qual o objeto foi implementado.

Corba

A arquitetura Corba pode ser categorizada em quatro interfaces de software (cont.):

- Serviços: ciclos de vida dos objetos, notificação de eventos e transações distribuídas;

Corba

A arquitetura Corba pode ser categorizada em quatro interfaces de software (cont.):

- Facilidades: são interfaces horizontais que têm foco na interoperabilidade entre aplicações distribuídas;

Corba

A arquitetura Corba pode ser categorizada em quatro interfaces de software (cont.):

- Domínios: inúmeros segmentos, tais como financeiro, farmacêutico, administração de negócios são exemplos de segmentos que usam da arquitetura Corba.

Corba

As interfaces do CORBA são caracterizadas como uma coleção de três elementos:

- Operações;

- Atributos;

- Exceções.

Corba

O acesso aos atributos são efetuados por intermédio dos :

- Accessor (get operation);

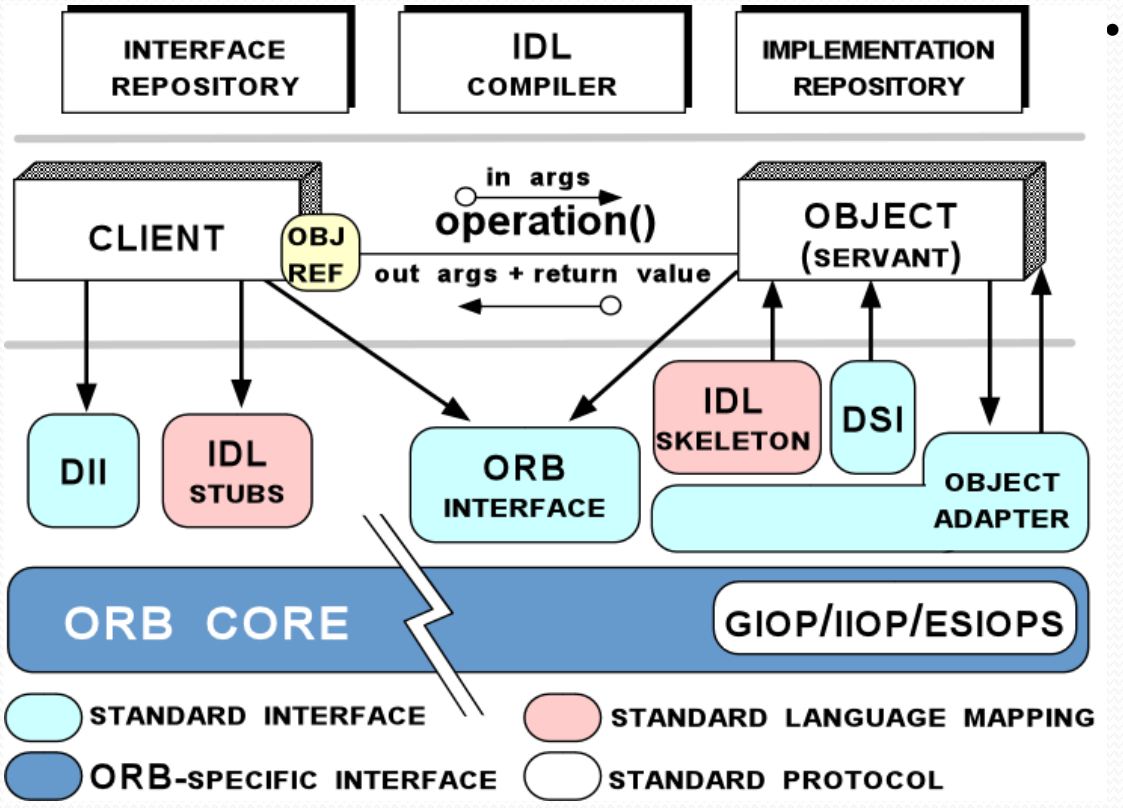
- Mutator (a set of operation);

Corba

CORBA é um middleware com a abordagem COD que não agrega as aplicações as dependências das plataformas heterogêneas distribuídas.

Exemplo*:* linguagens, sistemas operacionais, protocolos de rede, hardware

Corba



- CORBA automatiza
  - Localização de objetos
  - Conexões e gerência de memória.
  - (De) *Envelopamento* de parametros
  - Demultiplexação de eventos e solicitações
  - Tratamento de erro e tolerância a falta
  - Ativação de objeto-servidor
  - Concorrencia e sincronização
  - Segurança

CORBA define interfaces e políticas, mas *não* implementações

Corba

Componentes da Arquitetura ORB:

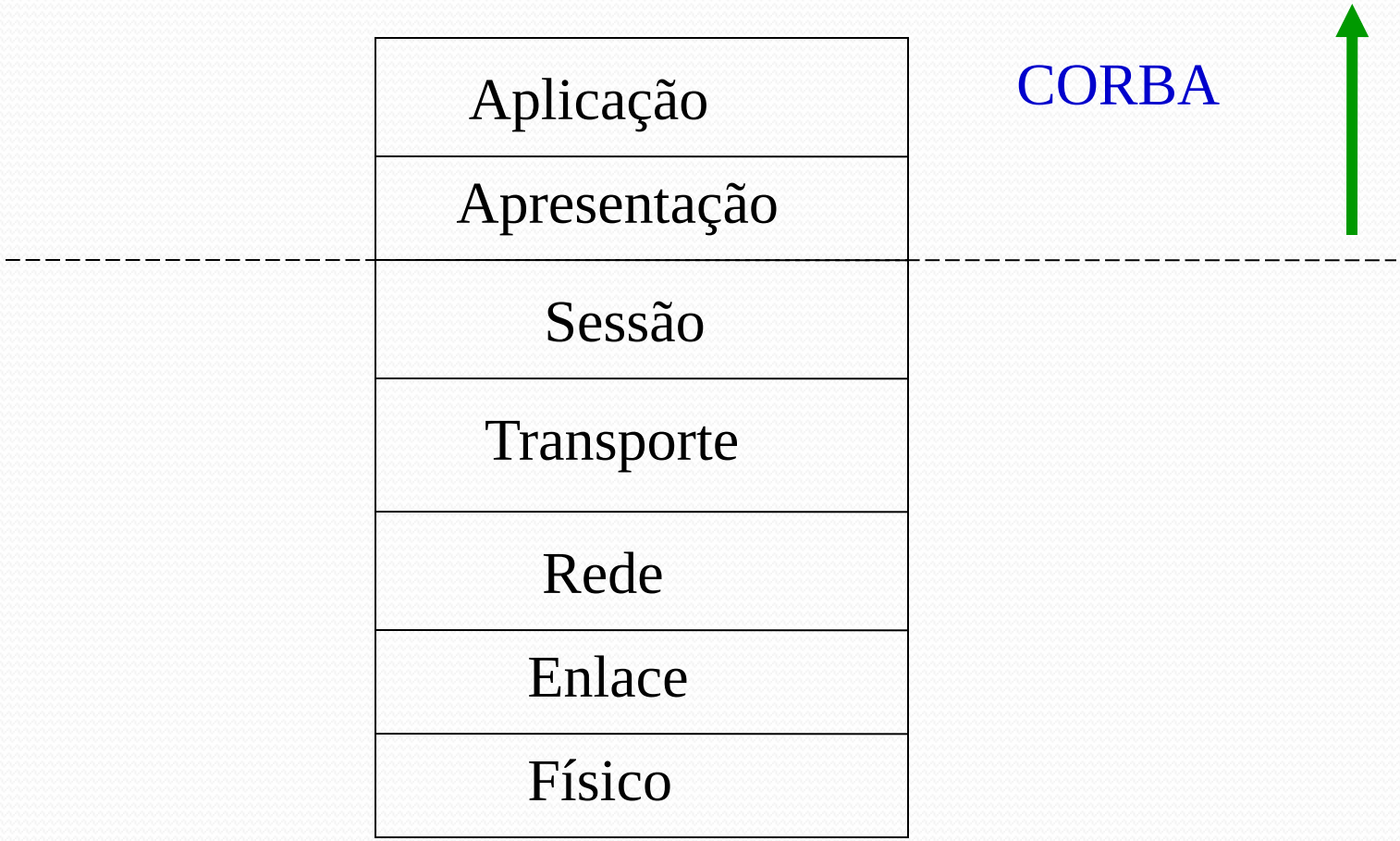
- Interface Definition Language (IDL)

- Internet Inter ORB Protocol (IIOP)

- Client Stub / Server Skeleton

- Client side : . Dynamic Invocation Interface (DII)
  . Interface Repository
- Server side:. Dynamic Skeleton Interface (DSI)
  . Implementation Repository

Corba - Interoperabilidade



Corba - Interoperabilidade

O IIOP requer quatro elementos:

- Maneira de representar referências nulas;

- Forma de conhecer o protocolo em uso;

- Maneira de prover significado para tipos que preservem sua identidade;

- Prover forma que cada correspondente possa obter sessões de chave privadas, ou seja maneira para cifrar e descifrar os protocolos que chegam ou saem.

Corba - Interoperabilidade

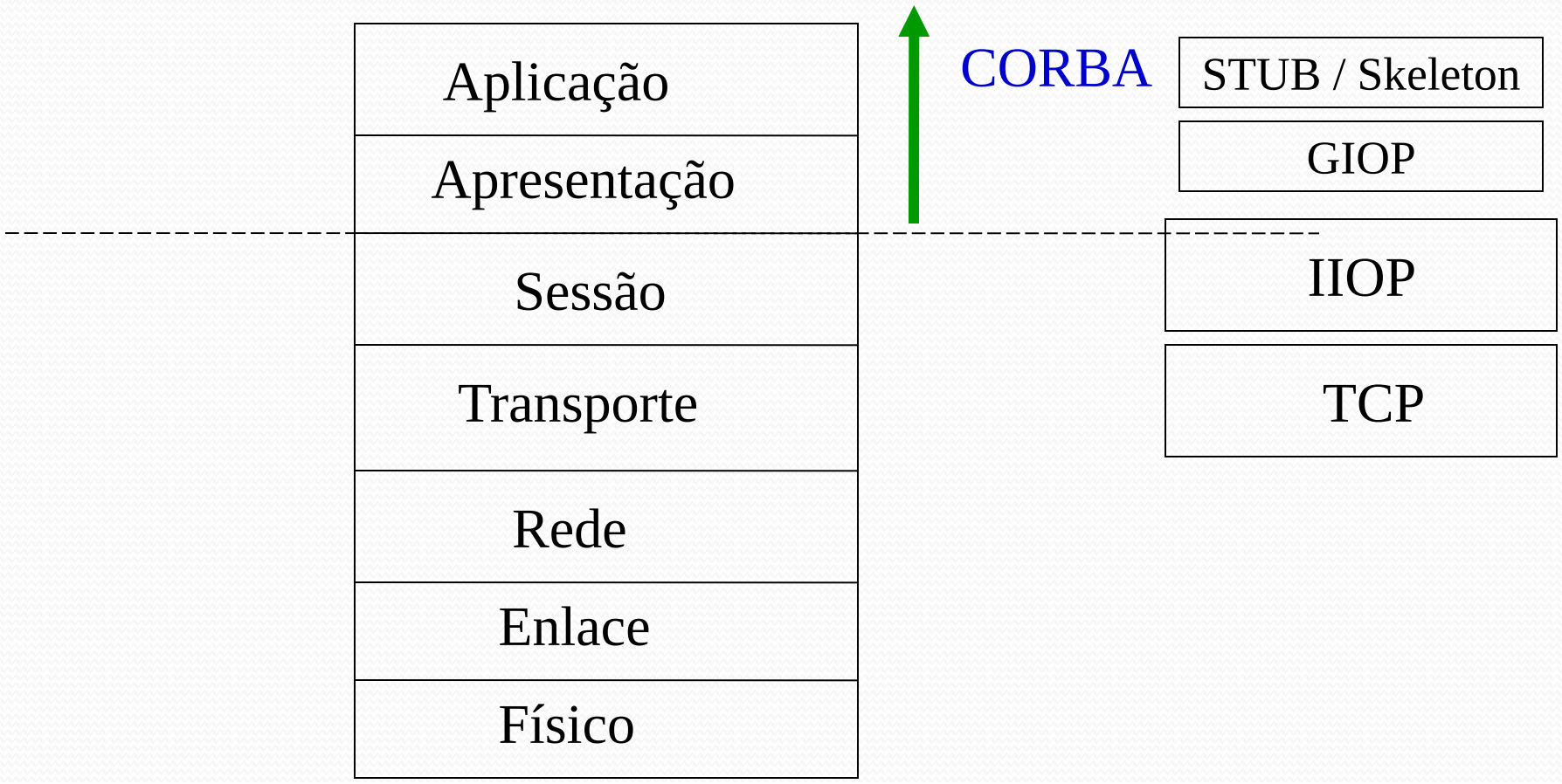
- O IIOP faz uma correlação entre GIOP para as sessões TCP;

- O IIOP especifica um *profile* que orienta o GIOP para usá-lo para uma conexão remota.

Corba - Interoperabilidade

Os protocolos GIOP (General Inter ORB Protocol) e IIOP provêem as facilidades anteriores através dos mecanismos internos do IOR (Interoperable Object Reference) que existem no ORB BUILDER.

GIOP


Aplicação
Apresentação
Sessão
Transporte
Rede
Enlace
Físico
CORBA
STUB / Skeleton
GIOP
IIOP
TCP

Conteúdo Programático

. Objetos distribuídos

. Componentes

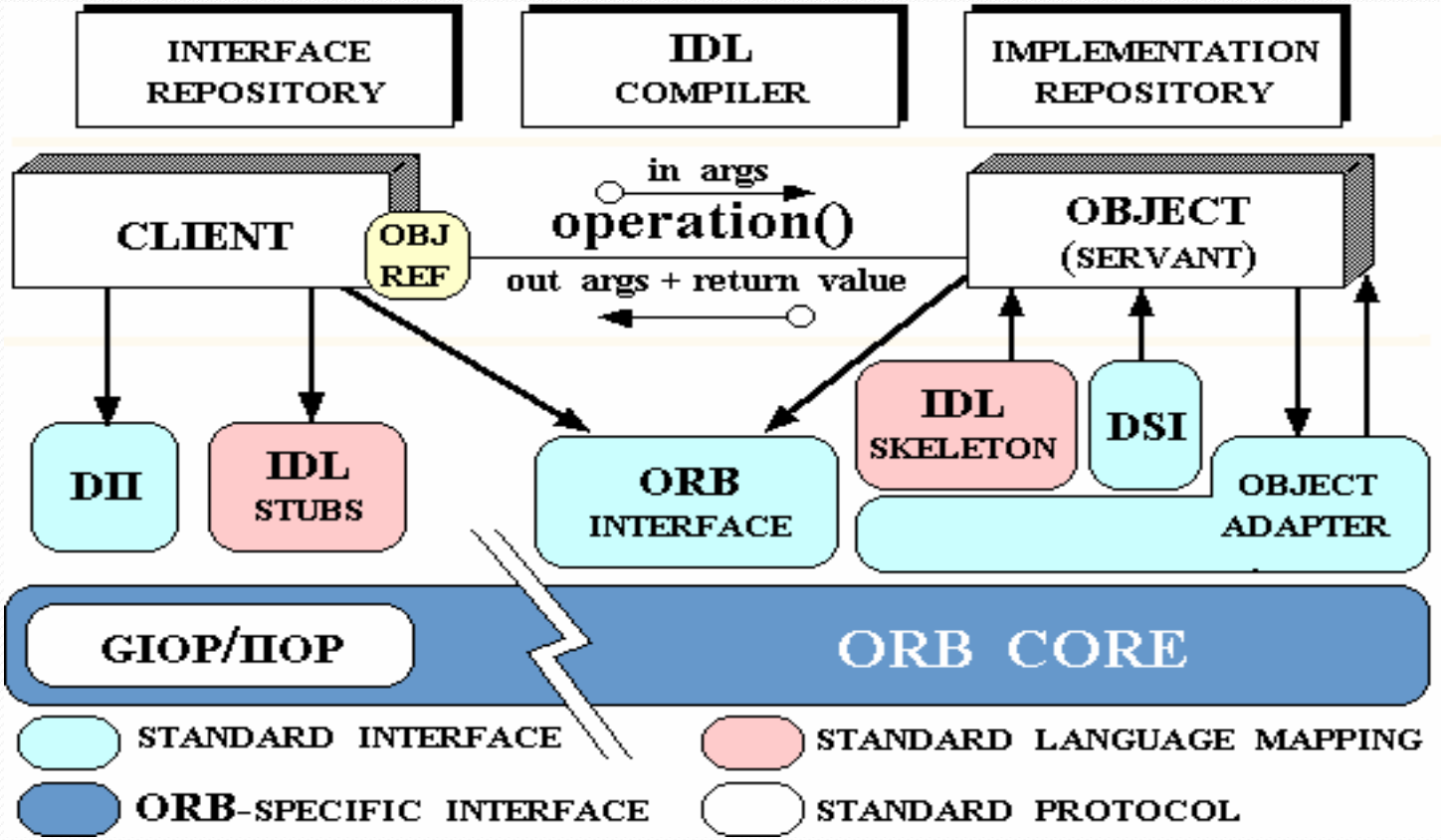
. CORBA

. Barramento de objetos CORBA

. Adaptador básico de objetos

. Linguagem de definição de interfaces

Corba


INTERFACE REPOSITORY
IDL COMPILER
IMPLEMENTATION REPOSITORY
CLIENT
OBJ REF
in args
operation()
out args + return value
OBJECT (SERVANT)
DII
IDL STUBS
ORB INTERFACE
IDL SKELETON
DSI
OBJECT ADAPTER
GIOP/IIOP
ORB CORE
STANDARD INTERFACE
STANDARD LANGUAGE MAPPING
ORB-SPECIFIC INTERFACE
STANDARD PROTOCOL

Conteúdo Programático

. Objetos distribuídos

. Componentes

. CORBA

. Barramento de objetos CORBA

. Adaptador básico de objetos

. Linguagem de definição de interfaces

Adaptador de Objetos

O Adaptador básico de objetos tem as seguintes características:

- Sugestão de estilo de implementação do objeto;

- O CORBA tem um adaptador básico genérico, conhecido como BOA (Basic Object Adapter);

- A BOA aceita diferentes estilos de ações fortemente acopladas ao ORB;

- A BOA é invocada durante várias fases do ciclos do objeto, tais como criação, destruição, ativação e desativação.

Adaptador de Objetos

A BOA (Basic Object Adapter) possui 5 estilos:

- Shared Server: Objetos compartilham um objeto;

- Unshared Server: Processos separados por objeto;

- Per Method Server: processos separados por solicitação;

- Persistent Server: ativado durante o *startup* não necessita de uma ação da BOA.

Adaptador de Objetos

Outros adaptadores de objetos Corba são:

- Library Object Adapter: utilizado para que possa haver um compartilhamento de uma biblioteca local em um primeiro instante. Utilizado para desenvolvimento de aplicações em WAN;

- Load Balancing Object Adapter: OAs por possuírem mais conhecimento de recursos disponíveis podem auxiliar na utilização de locais que possuem mais facilidades para processamento/armazenamento.

Adaptador de Objetos

Outros adaptadores de objetos Corba são (cont.):

- Mobile Object Adapter: conhece a localização do objeto melhor do que a BOA. Pode enfileira solicitações para objetos móveis.

Conteúdo Programático

. Objetos distribuídos

. Componentes

. CORBA

. Barramento de objetos CORBA

. Adaptador básico de objetos

. Linguagem de definição de interfaces

Componentes

- Object Request Broker (ORB)
- Interface Definition Language (IDL)
- Static Invocation Interface (SII)
- Dynamic Invocation Interface (DII)
- Dynamic Skeleton Interface (DSI)

Componentes: IDL

A IDL é a linguagem utilizada para descrever interfaces de chamadas dos objetos clientes e também que tipo de interface de implementação o objeto prove.

- A IDL é puramente uma linguagem descritiva, ou de definição;

- A IDL é a linguagem de ***contrato*** orientada a objeto do padrão Corba.

Componentes: IDL

Um compilador IDL é composto de :

- FE (Front End ou Compilador IDL): faz o pré-processamento ;

- BE (Back End): efetua a tradução, ou mapeamento, é orientado a uma linguagem específica que cria o código em linguagens tipo C, C++, Rexx, Cobol.

Utilizando OO em programação de rede

- O IDL do CORBA especifica as *interfaces* com as operações
  - O mapeamento faz interface entre objetos para uma linguagem OO.
    - Exemplos: C++, Java, Ada95.

```
interface Foo
{
  void bar (in long arg);
};
```

IDL

C++

```
class Foo : public virtual CORBA::Object
{
  virtual void bar (CORBA::Long arg);
};
```

  - Operações definidas na interface podem fazer chamadas a objetos locais ou remotos.

Componentes: IDL

- Interface Description Langage
  - Mapping: C, C++, Ada, COBOL, Java, Smalltalk
- Syntaxe: baseada em C++
- Facilidades:
  - Modules, Interfaces
  - Operations, Attributes
  - Inheritance
  - Basic types (e.g., double, long, char, etc)
  - Arrays, sequence
  - Struct, enum, union, typedef
  - Consts
  - Exceptions

IDL: Módulos e Interfaces

```
module SportEvent {
    interface People {
        attribute string name;
        };
    const short MaxTeamPeople = 100;
    interface Team {
        attribute sequence<People, MaxTeamPeople> Members;
        };
    }
```

IDL: Tipos Construídos

- Union
  - union Result switch (ResultType) {
  - case Length: float value;
  - case Weight: short value;
  - default: long value; };
- Enums
  - enum ResultType {Length, Time, Weight};
  - enum SpringStatus {Valid, NotValid};
- Structures
  - struct Sport { string Name;
  - string Category; };
- Exceptions
  - exception NoScore{string Explanation};

IDL: Interface e operações

```
interface Competition;

interface Score {
   attribute Competition TheCompetition;

   unsigned long NbPeople();
   unsigned long GetPosition(in People APeople)
                                raises (NoScore);
   void GetResult(in People APeople, out Result TheResult);
   };
```

- oneway operations
- parameters: in, inout, out.

IDL: Interface Inheritance

- interface Competition {
- attribute Sport TheSport;
- attribute Score TheResults;
- };
- interface IndividualCompetition: Competition {
- readonly attribute sequence<People> Participants;
- void AddParticipant(in People APeople);
- void AddResult(in People APeople, in Result AResult)
- };

- Multiple Inheritance

Componentes

- Object Request Broker (ORB)
- Interface Definition Language (IDL)
- Static Invocation Interface (SII)
- Dynamic Invocation Interface (DII)
- Dynamic Skeleton Interface (DSI)

Static Invocation Interface

- lado do cliente
- tipos são conhecido previamente
- tipos confiáveis
- necessita “empacotamento”

Componentes

- Object Request Broker (ORB)
- Interface Definition Language (IDL)
- Static Invocation Interface (SII)
- Dynamic Invocation Interface (DII)
- Dynamic Skeleton Interface (DSI)

Dynamic Invocation Interface (DII)

- lado do cliente
- Baseado em ‘Interface Repository’
- sem “empacotamento”
- novas interfaces podem ser criadas;
- Ferramentas CASE

Dynamic Invocation Interface (DII)

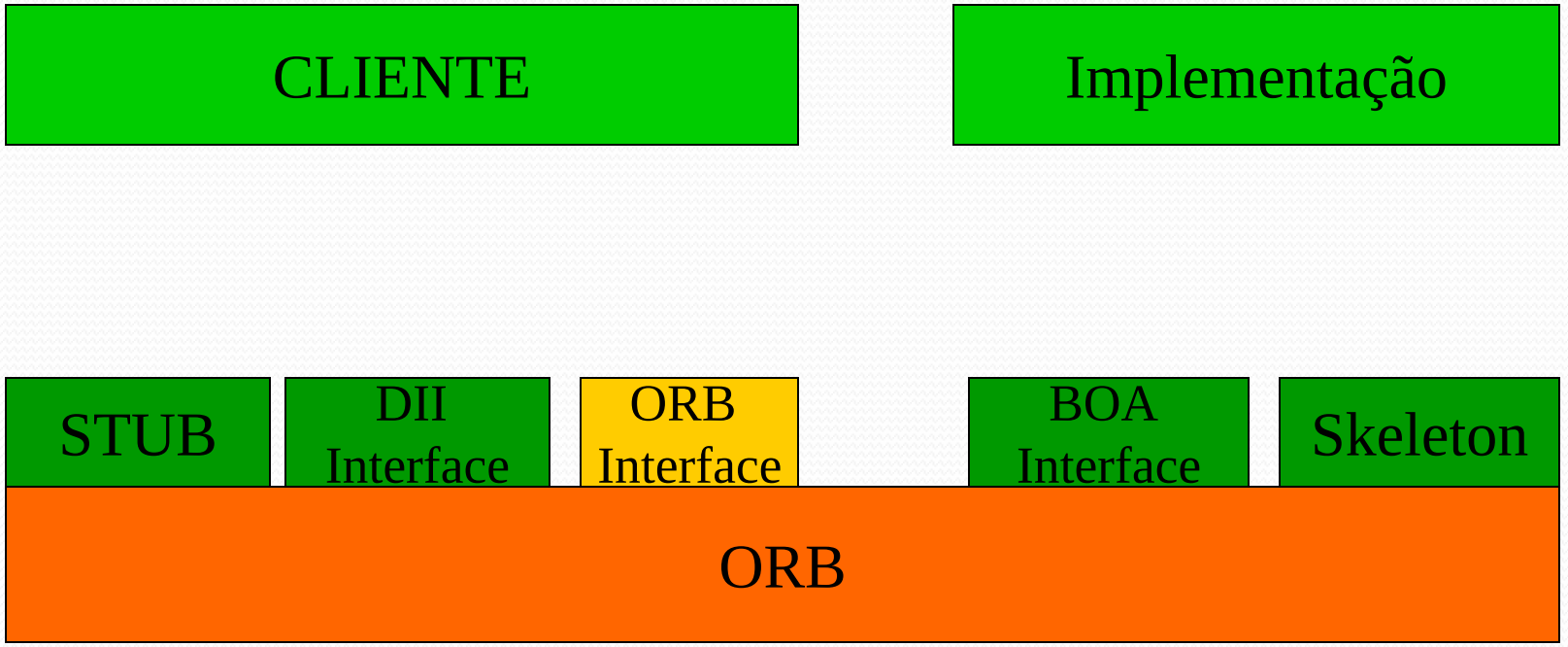
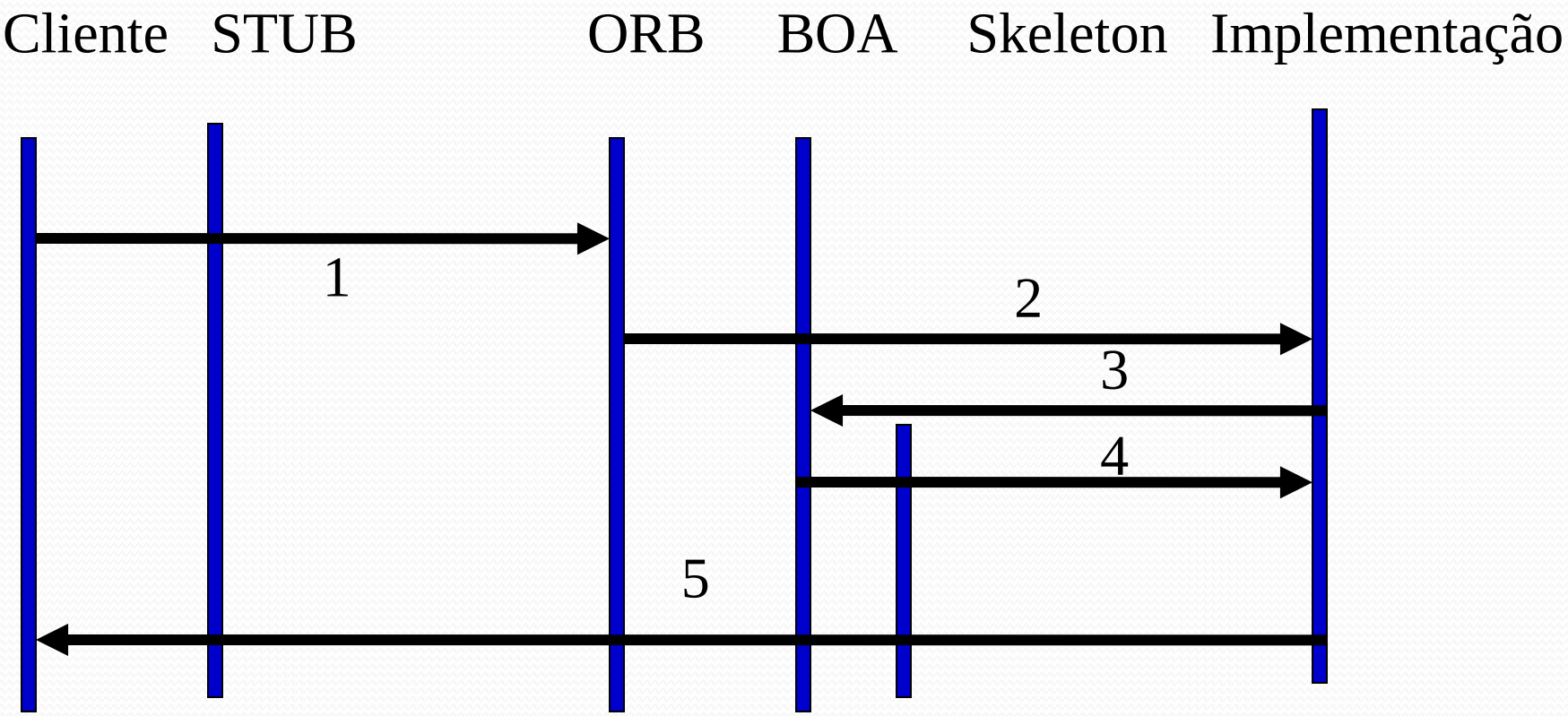


Diagrama de Interação em Corba



Static IDL Skeleton

- Lado do servidor
- Existe um enlace estático com a implementação do objeto.

Componentes

- Object Request Broker (ORB)
- Interface Definition Language (IDL)
- Static Invocation Interface (SII)
- Dynamic Invocation Interface (DII)
- Dynamic Skeleton Interface (DSI)

Dynamic Skeleton Interface (DSI)

- Lado do servidor
- Um ponto de entrada é requerido
- Existe um disparo interno

Módulo III

. Serviços CORBA

. Objetos COM

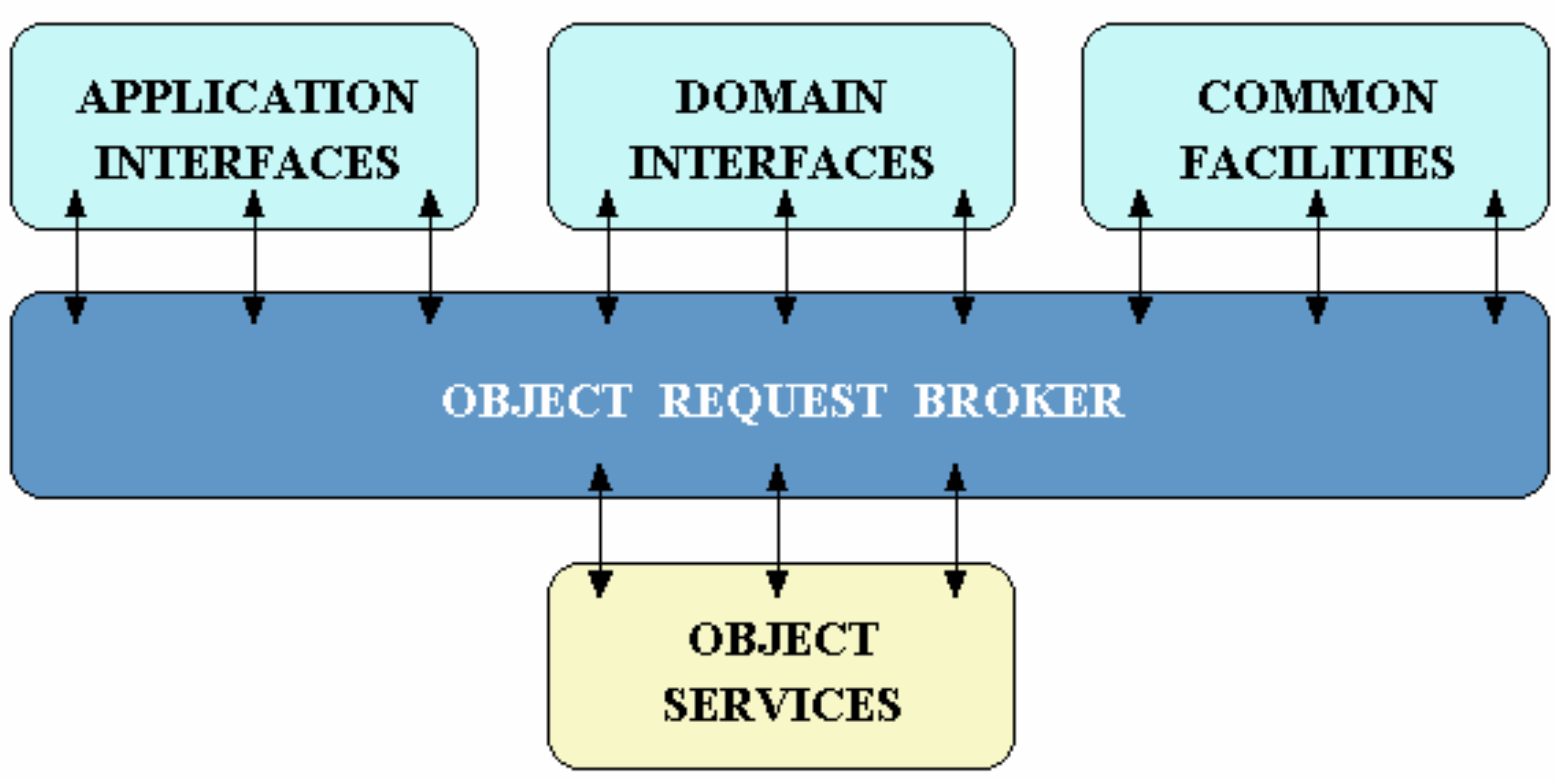
. Barramento de objetos COM

. Serviços OLE/COM

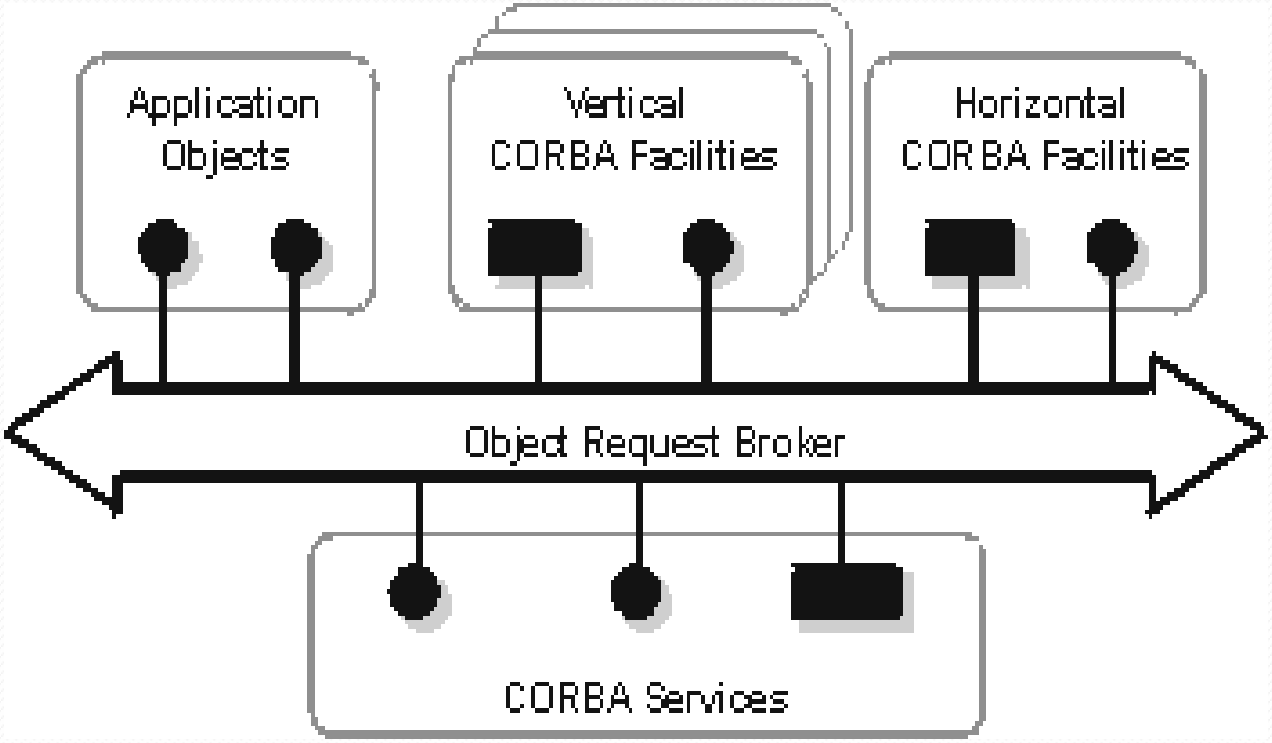
CORBA

Serviços e Facilidades

Revisão


APPLICATION INTERFACES
DOMAIN INTERFACES
COMMON FACILITIES
OBJECT REQUEST BROKER
OBJECT SERVICES

Object Management Architecture (OMA)



[www.omg.org]

Common Facilities

- Orientada para aplicações de usuários finais
- Alto nível
- **Facilidades Horizontal**
  - User Interface (Rendering, Compound Presentation, User Support, Desktop, Scripting)
  - Information Management (Inf. Modeling, Inf. Storage, Compound Interchange, Data Int., Information Int., Time operations)
  - System Management (Management Tools, Collection Management, Control)
  - Task Management (Workflow, Agent, Rule Management, Automation)

Common Facilities

- Facilidades de Domínio Vertical

Imagery	Provides interoperability between imagery objects, image-related information, and imagery application services.
Information Superhighways	Supports multi-user information service applications across wide area networks.
Manufacturing	Supports interoperability between manufacturing objects.
Distributed Simulation	Supports the interaction of multiple simulation objects in virtual environments.
Oil and Gas Industry Exploitation and Production	Supports interoperability in the petroleum vertical market.
Accounting	Supports commercial transactions.
Application Development	Supports interoperability between application development objects.
Mapping	Supports interoperability between mapping objects

Object Services

- Independente domínio
- Exemplos:
  - Naming,
  - Event,
  - Life Cycle,
  - Persistent Object,
  - Transaction,
  - Concurrency Control,
  - Relationship,

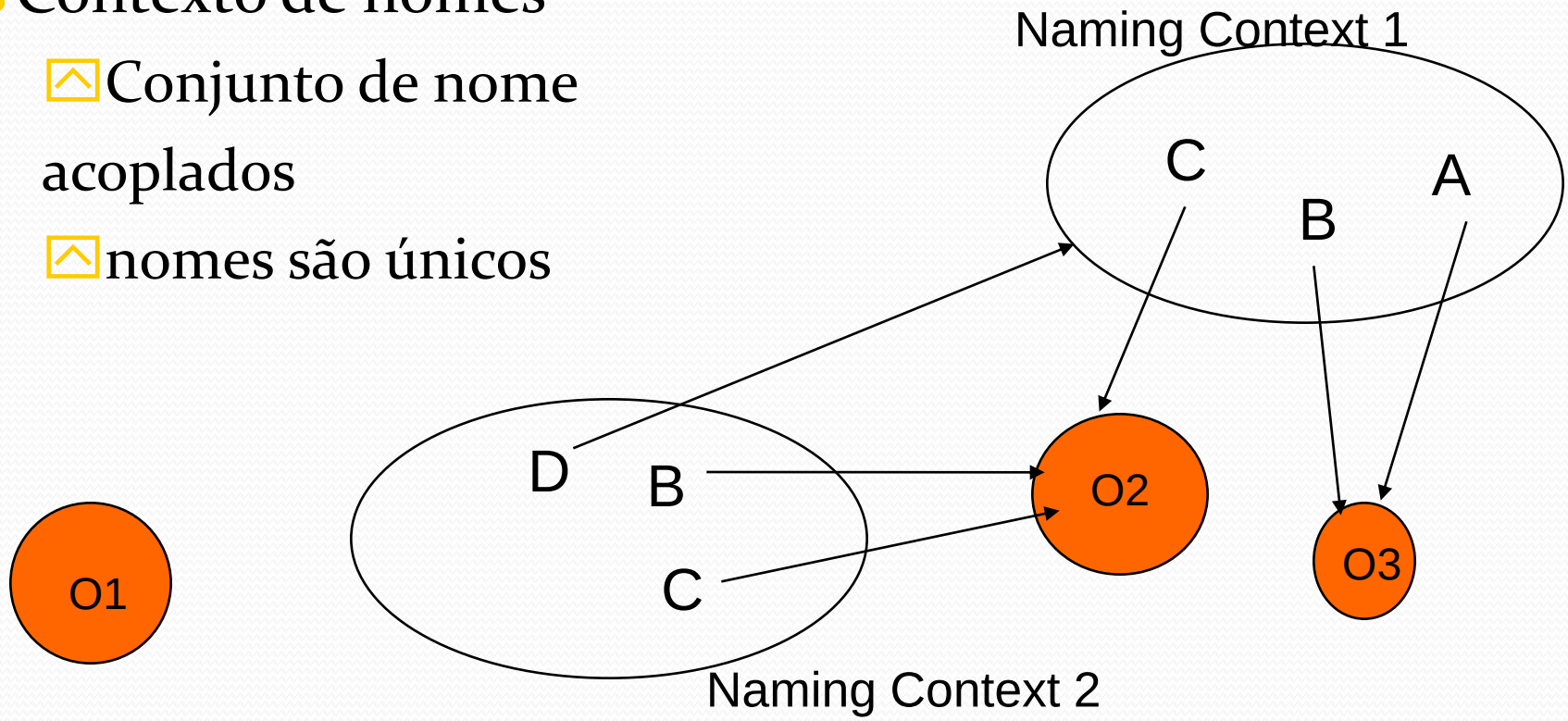
Object Services

Exemplos (cont.):

- Externalization,
- Query,
- Licensing,
- Property,
- Time,
- Security,
- Object Trader.

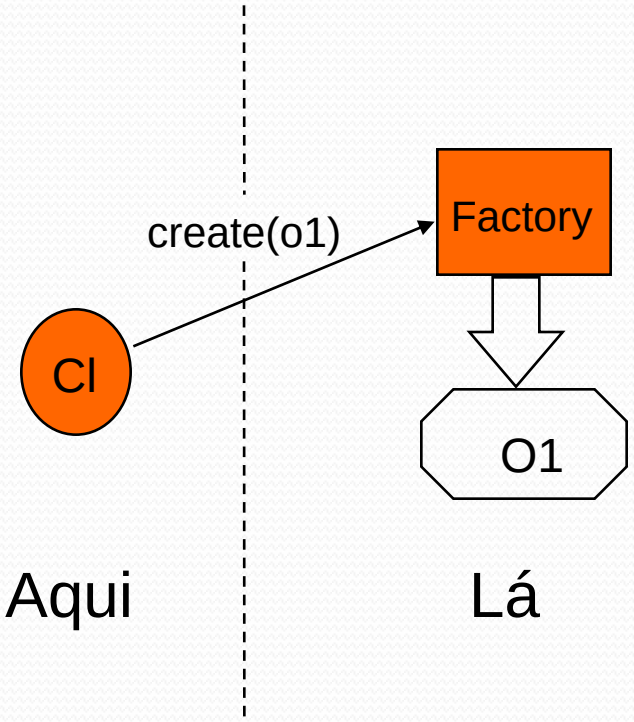
Naming Service

- Prove serviço de acoplamento de nome
- Contexto de nomes
  - Conjunto de nome acoplados
  - nomes são únicos



Life Cycle Service

- Criar, Deletar, Copia, Mover objetos
- Client / Factories
- Factories Finder (move)



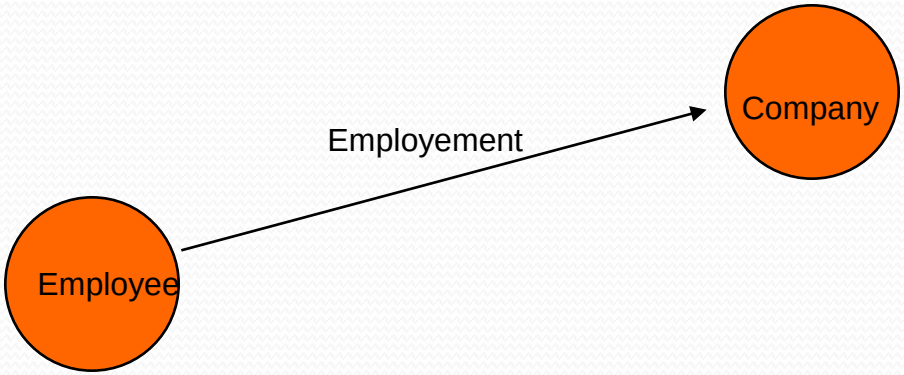
Concurrency Control Service

- Modos Transacional e não Transacional
- Locks: leitura (r), escrita (w), update , +intention

Granted Mode	Requested Mode				
	IR	R	U	IW	W
IR					*
R				*	*
U			*	*	*
IW		*	*		*
W	*	*	*	*	*

Relationship Service

- Relacionamento / Regras
- Grau, Cardinalidade, atributos



Query Service

- Utilização de SQL e OQL-like consultas em coleção de objetos.
- Seleção / atualização.
- Pode invocar operações IDL nos objetos

Time Service

- Usa o **UTC** (Universal Time Coordinated) do X/Open DCE Time Service
- Serviço provido:
  - Informação da hora atual (+ estimativa de erro),
  - Gera eventos baseados em tempo usando temporizadores e alarmes,
  - Efetua o calculo entre dois intervalos,
  - Especifica a ordem na qual os eventos ocorrem.

Security Service

- Confidencialidade, Integridade, Contabilidade, Disponibilidade
  - Identificação, Autenticação,
  - Autorização e controle de acesso,
  - Auditoria de segurança,
  - Segurança de comunicação,
  - Não repudio,
  - Administração.

Licensing Service

- Não impõe uma política de negócios e prática,
- Licença
  - Tempo (início/duração, expiração)
  - Mapeamento de valores (alocação, consumo de recursos)
  - Consumidor (entidades específicas, máquinas específicas)

Conteúdo Programático

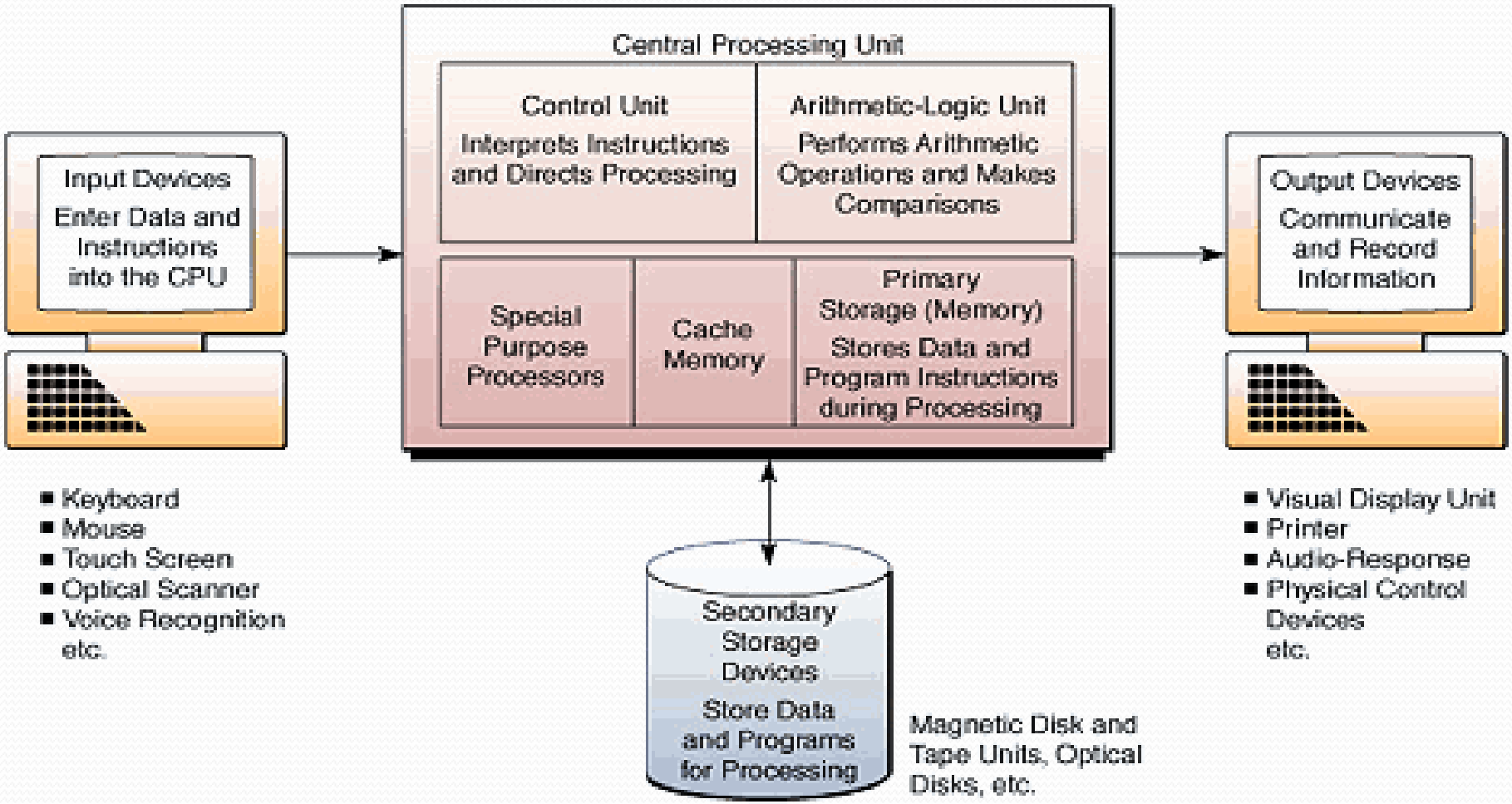
. Serviços CORBA

. Objetos COM

. Barramento de objetos COM

. Serviços OLE/COM

Arquitetura Centralizada em Mainframe



Paradigma Cliente-Servidor

O paradigma cliente-servidor é um paradigma muito comum e usado praticamente em todos os processos distribuídos em que a aplicação servidora (aquela que aguarda a conexão) aguarda mensagens, executa serviços e retorna resultados.

A aplicação cliente, pelo contrário, é a que estabelece a ligação, envia mensagens para o servidor e aguarda mensagens de resposta.

Paradigma Cliente-Servidor

Os clientes e servidores trocam mensagens através dos protocolos de transporte (exemplos são o TCP e o UDP), o cliente e o servidor devem ter a mesma pilha de protocolos e ambos interagem com a camada de transporte do TCP/IP.

Ambiente Cliente Servidor

Uma aplicação cliente-servidor é aquela na qual o cliente, que instancia a interface do usuário da aplicação, liga-se a um servidor de aplicação ou sistema de base de dados (também chamado de banco de dados).

Ambiente Cliente Servidor

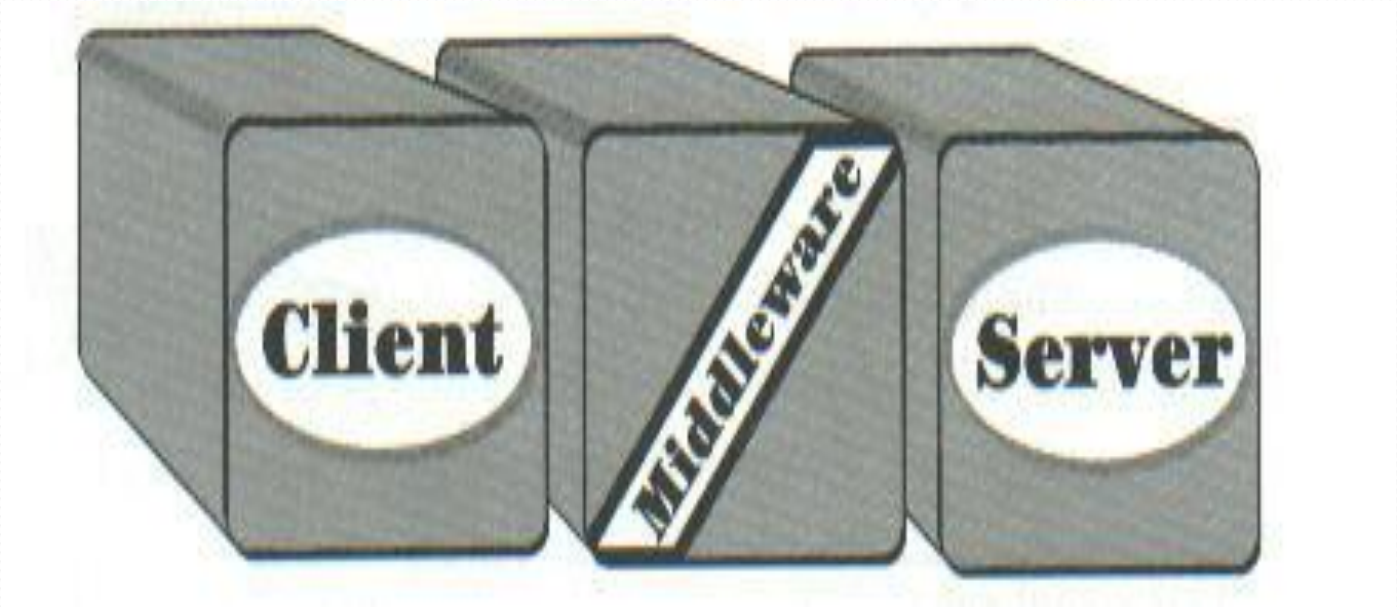
Quando um cliente se conecta diretamente com um sistema de base de dados, ou com uma aplicação servidora monolítica, a arquitetura da aplicação é uma arquitetura duplamente ligada.

Ambiente Cliente Servidor

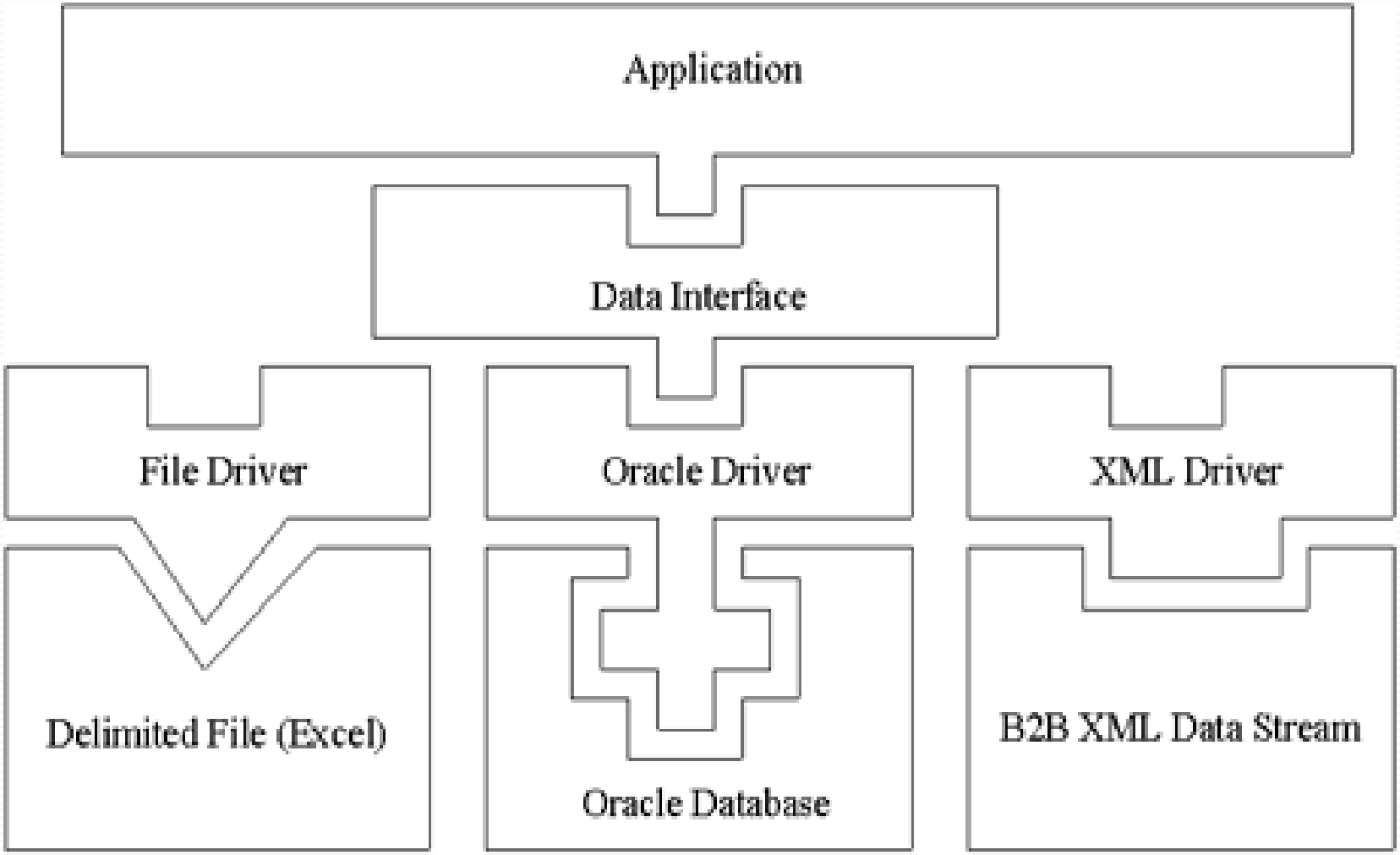
Os protocolos da camada de Aplicação fornecem serviços de mais alto nível tais como a Web, servidores de nomes, correio eletrônico, login remoto e a transferência de arquivos, por exemplo.

O que estas aplicações têm em comum é o uso da arquitetura do tipo 'cliente-servidor' [en.wikipedia.org]

Ambiente Cliente Servidor



Middleware


Application
Data Interface
File Driver
Oracle Driver
XML Driver
Delimited File (Excel)
Oracle Database
B2B XML Data Stream

Paradigmas de Middleware

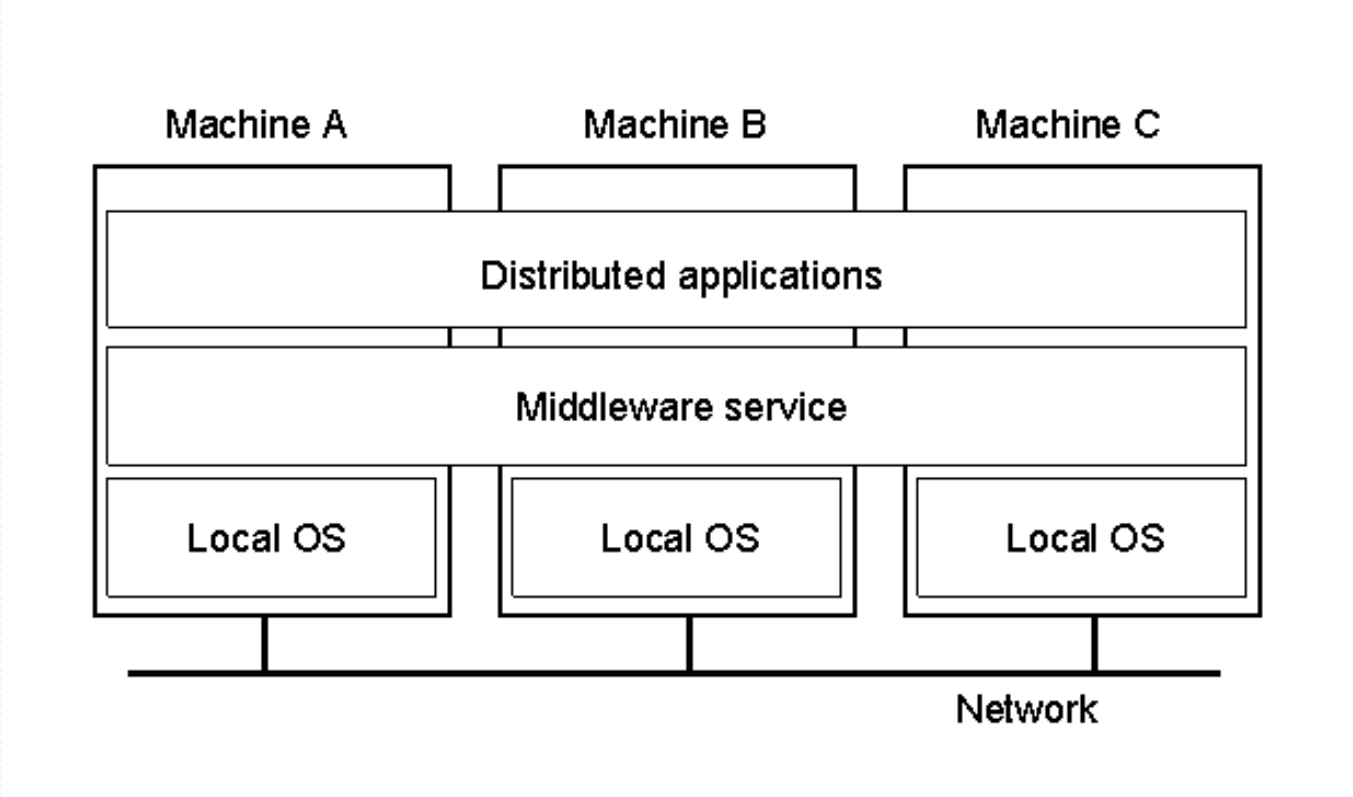
No projeto sistemas distribuídos o hardware tem grande importância, mas na verdade o software é o fator determinante da aparência do sistema.

Neste sentido os ambientes de *middleware* têm uma grande importância.

Paradigmas de Middleware

O *middleware* é uma camada adicional de software que é colocada entre as aplicações e o sistema operacional, considerando uma arquitetura de protocolo de rede, oferecendo um alto nível de abstração.

Paradigmas de Middleware



Um sistema distribuído organizado como middleware, o nível de middleware cria a abstração de um único ambiente, todavia usando multiplas maquinas.

Paradigmas de Middleware

Os quatro paradigmas de *middleware* mais utilizados são:

1. Transaction Processing Monitors (TPs): esse controlam aplicações de transações e executam computações de negócios e atualizações em bancos de dados;

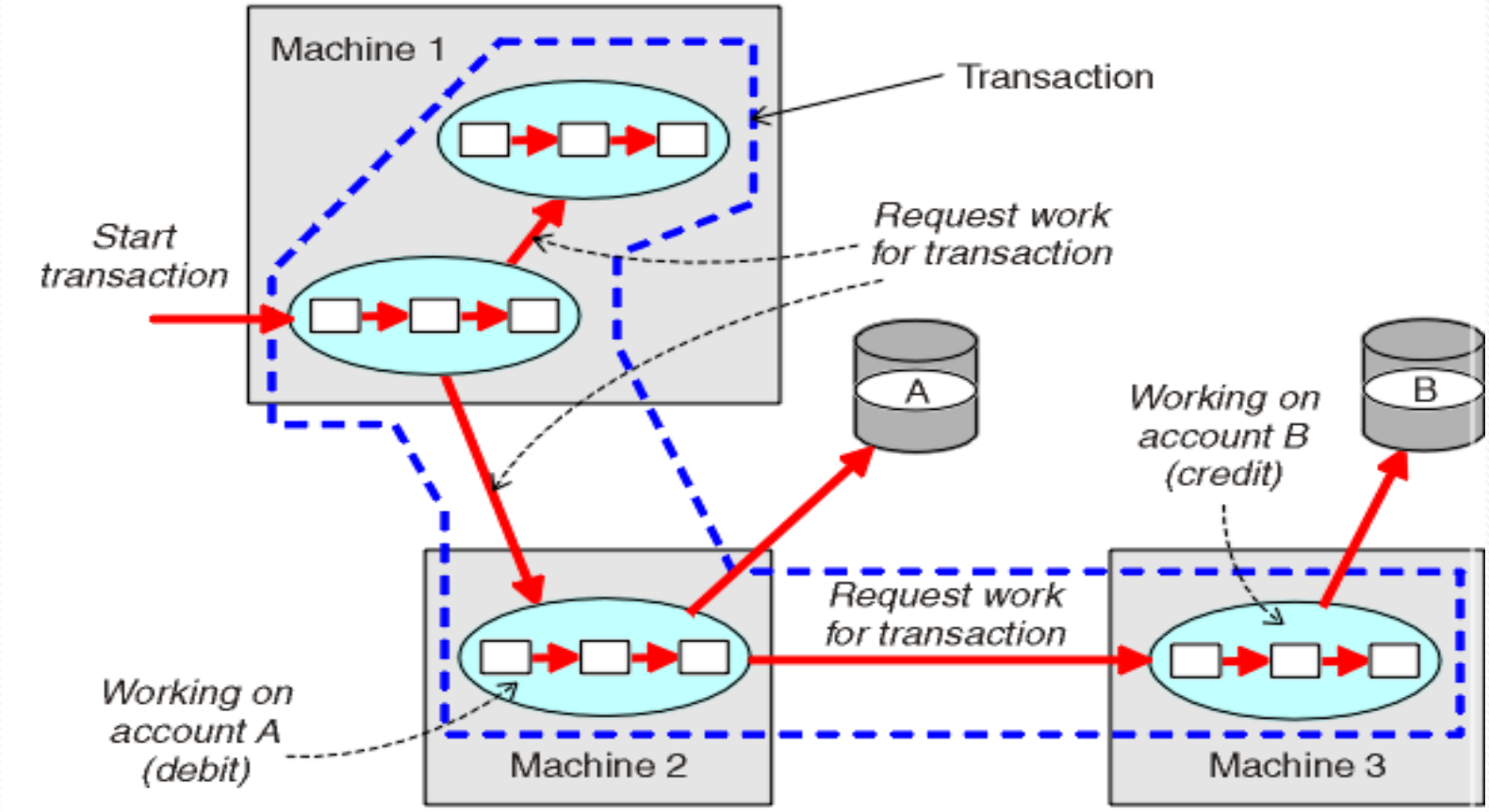
2. Remote Procedure Calls (RPCs): permitem que processos clientes em uma aplicação chamem funções para acessar servidores em sistemas remotos;

Monitores de Transações de Processamento

A tecnologia *Transaction Processing* (TP) *Monitor* provê ao ambiente cliente/servidor distribuído a capacidade para desenvolver, rodar e gerenciar aplicações com transações de forma eficiente e confiável.

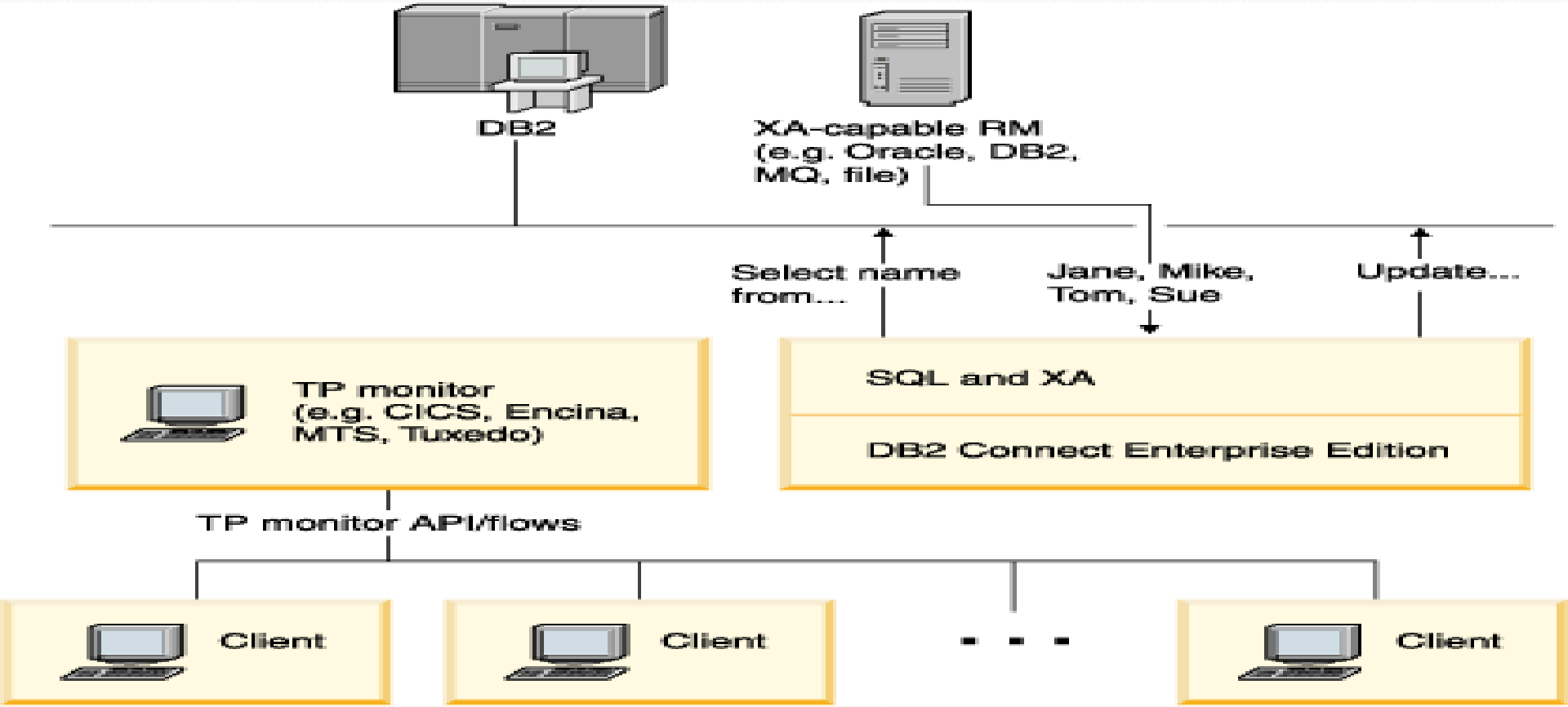
Paradigmas de Middleware

1. Monitores de Transações de Processamento



Paradigmas de Middleware

1. Monitores de Transações de Processamento



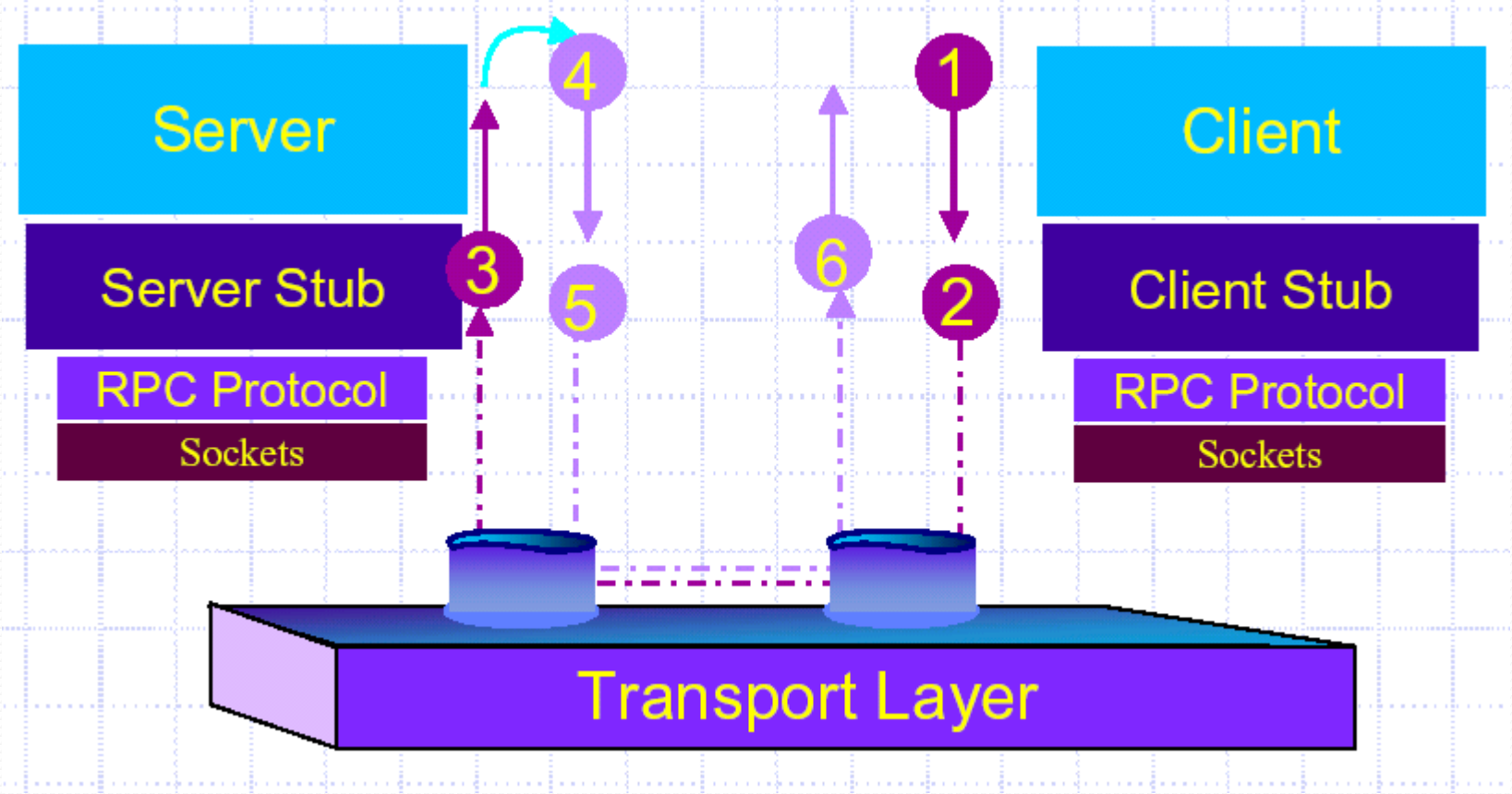
Monitores de Transações de Processamento

A tecnologia TP monitor controla aplicações de transação e realiza computações de negócios e atualizações em banco de dados.

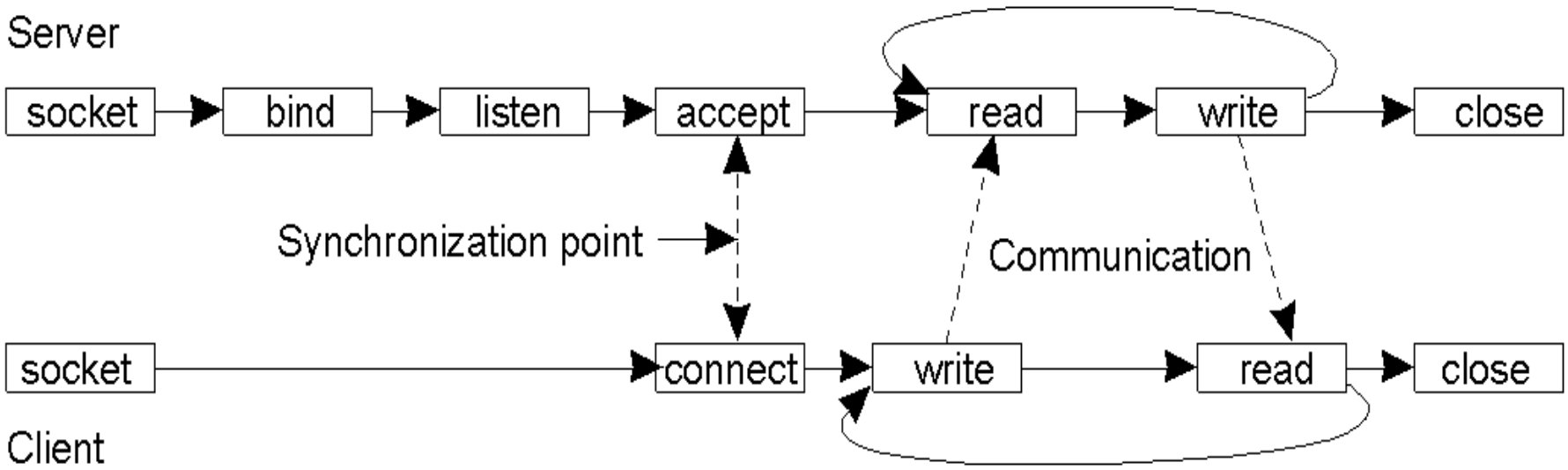
A tecnologia TP monitor surgiu há 25 anos atrás quando a *Atlantic Power and Light* criou um ambiente de suporte online para compartilhar serviços de aplicações concorrentes e recursos de informação através de um ambiente de sistemas operacionais.

RPC

2. Remote Procedure Call



Sockets



Comunicação orientada a conexão usando sockets.

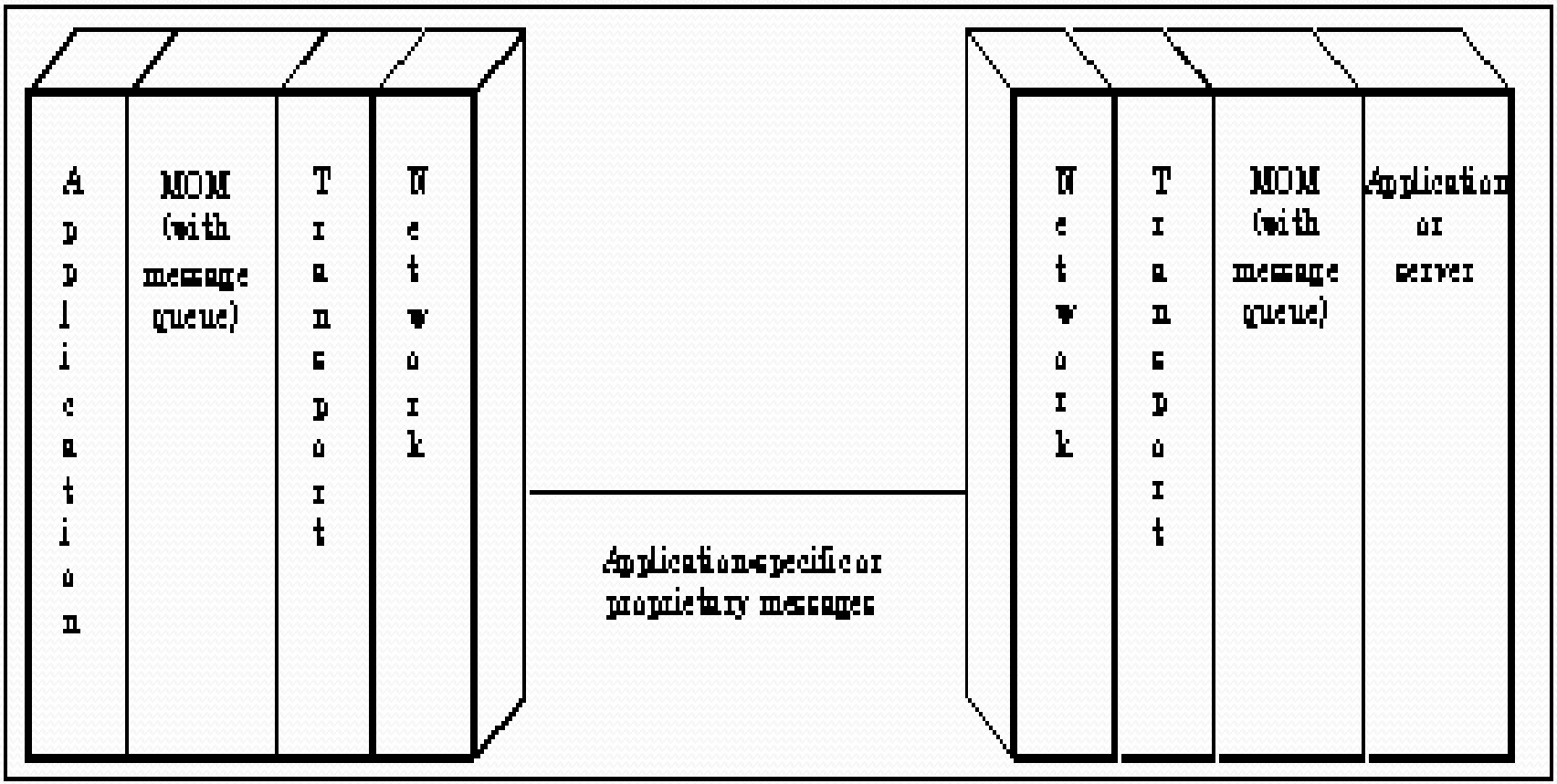
Paradigmas de Middleware

3. Message-Oriented Middlewares (MOMs): residem em ambas partes de uma arquitetura cliente/servidor e suportam chamadas assíncronas entre as aplicações;

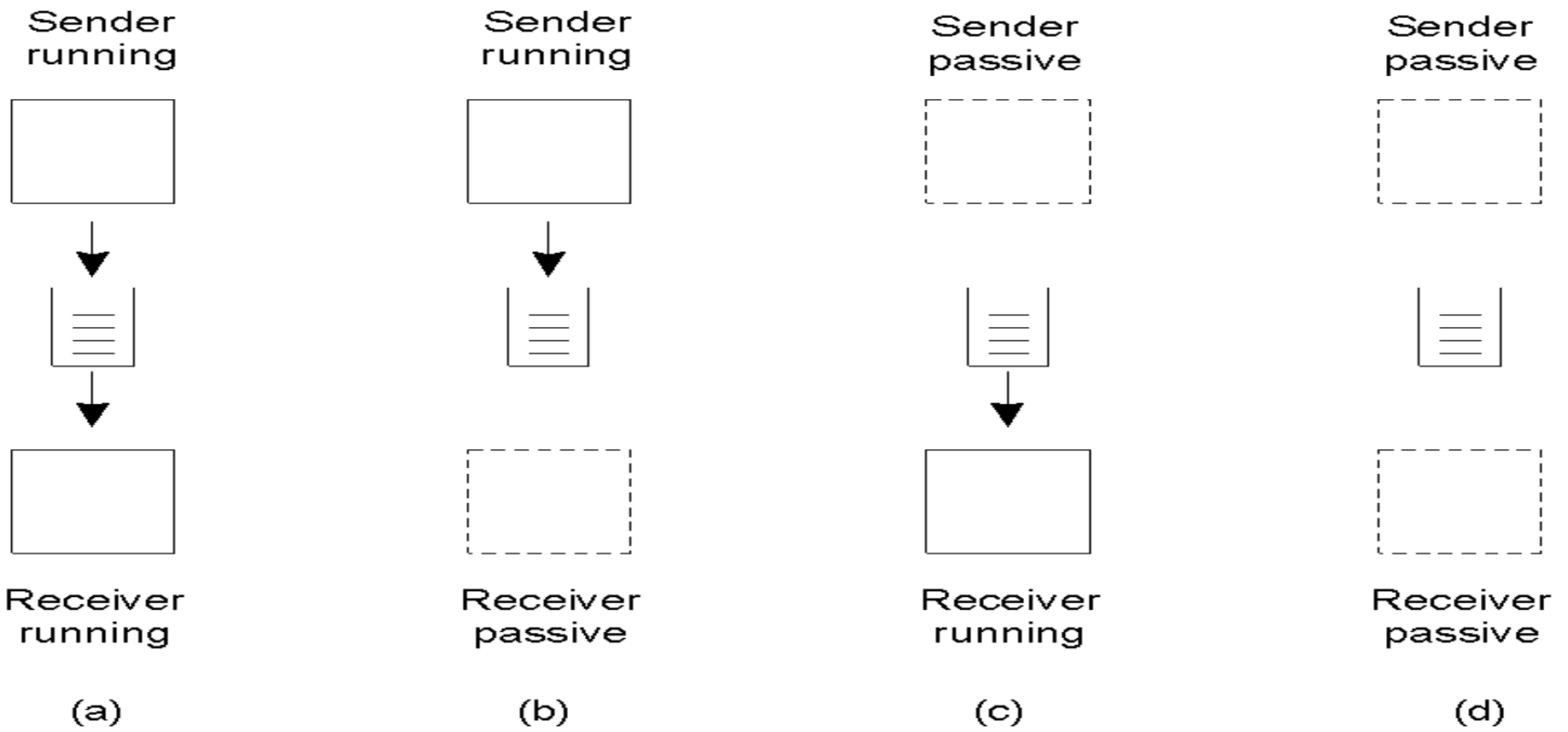
4. Object Request Brokers (ORBs): são caracterizados por objetos que compõem uma aplicação e que podem estar distribuídos em redes heterogêneas.

Middleware Orientado a Fila de Mensagem

3. Message-Oriented Middlewares



Modelo Fila de Mensagens



Quatro combinaçoes para comunicaçao fracamente acoplada usando filas.

Paradigmas de Middleware

4. Object Request Brokers (ORBs)

A tecnologia ORB permite a comunicação de objetos entre diferentes máquinas, diferentes softwares e diferentes fornecedores.

Um ORB provê um diretório de serviços e auxilia a estabelecer conexões entre clientes e estes serviços. Esta é a primeira etapa na abordagem ORB.

Paradigmas de Middleware

4. Object Request Brokers (ORBs)

O próximo passo é a comunicação de objetos entre plataformas.

Um ORB permite aos objetos esconder seus detalhes de implementação dos clientes. Isto inclui linguagens de programação, sistemas operacionais, hardware, e localização de objetos.

Paradigmas de Middleware

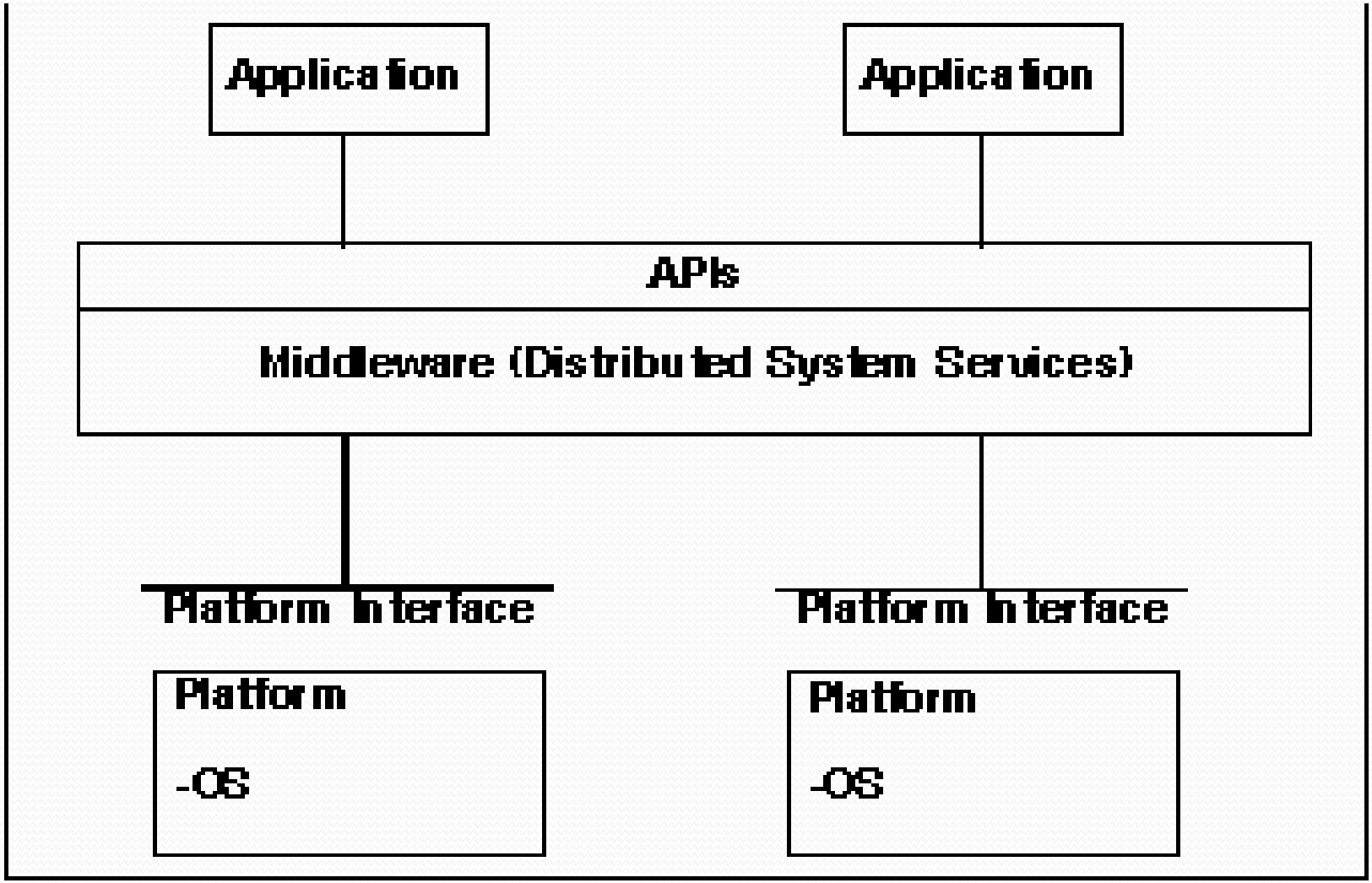
4. Object Request Brokers (ORBs)

Um ORB pode ser implementado de diversas maneiras.

As funções de um ORB podem, por exemplo, estar compiladas dentro dos programas clientes; podem ser processos separados ou daemons; ou podem ainda ser partes de um kernel de um sistema operacional.

Exemplos clássicos de ORB são o CORBA, COM/DCOM e Java RMI

Middleware


Application
Application
APIs
Middleware (Distributed System Services)
Platform Interface
Platform Interface
Platform
-OS
Platform
-OS

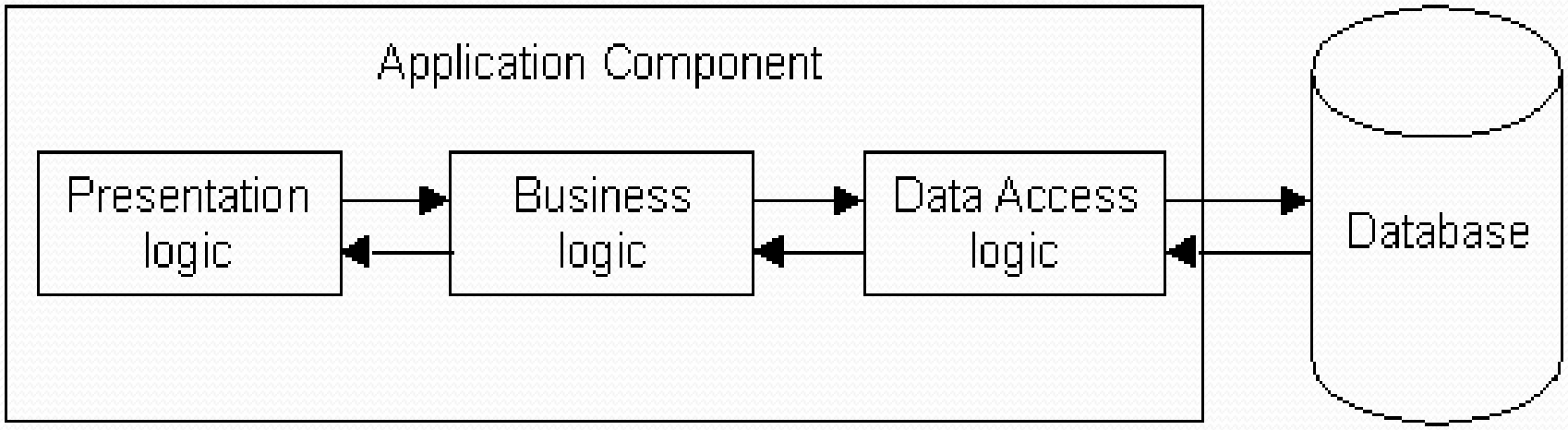
Arquitetura *Multi-tier*

Multitier Architecture

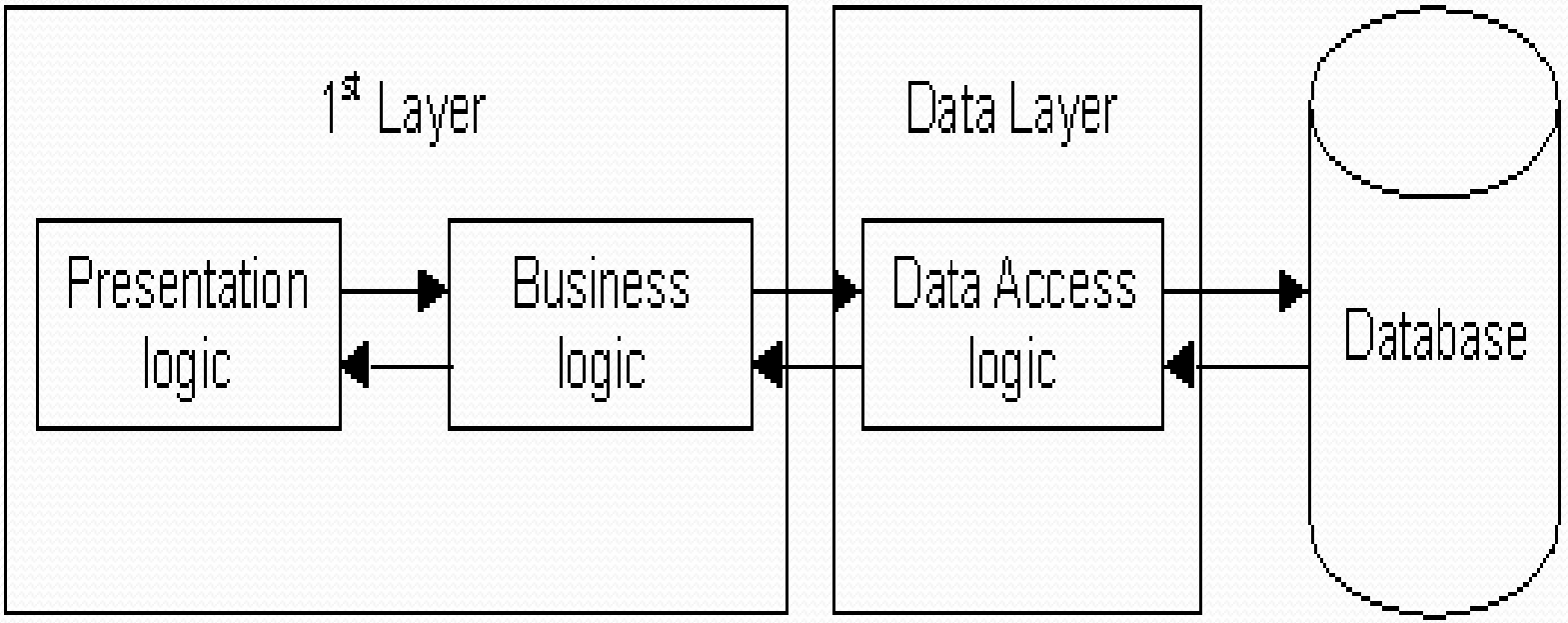
In software engineering, multi-tier architecture (often referred to as n-tier architecture) is a client-server architecture in which an application is executed by more than one distinct software agent.

For example, an application that uses middleware to service data requests between a user and a database employs multi-tier architecture. The most widespread use of "multi-tier architecture" refers to three-tier architecture [en.Wikipedia.org].

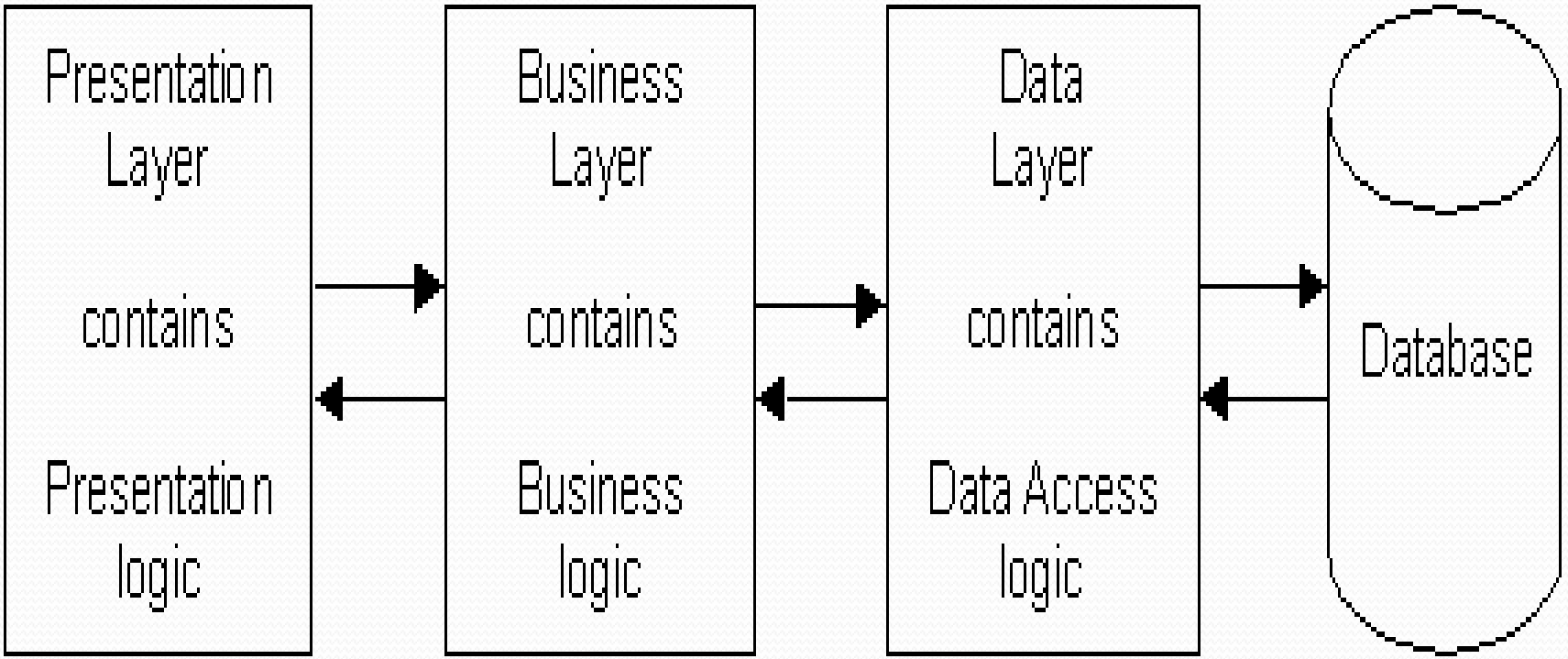
Arquitetura *Multi-tier*



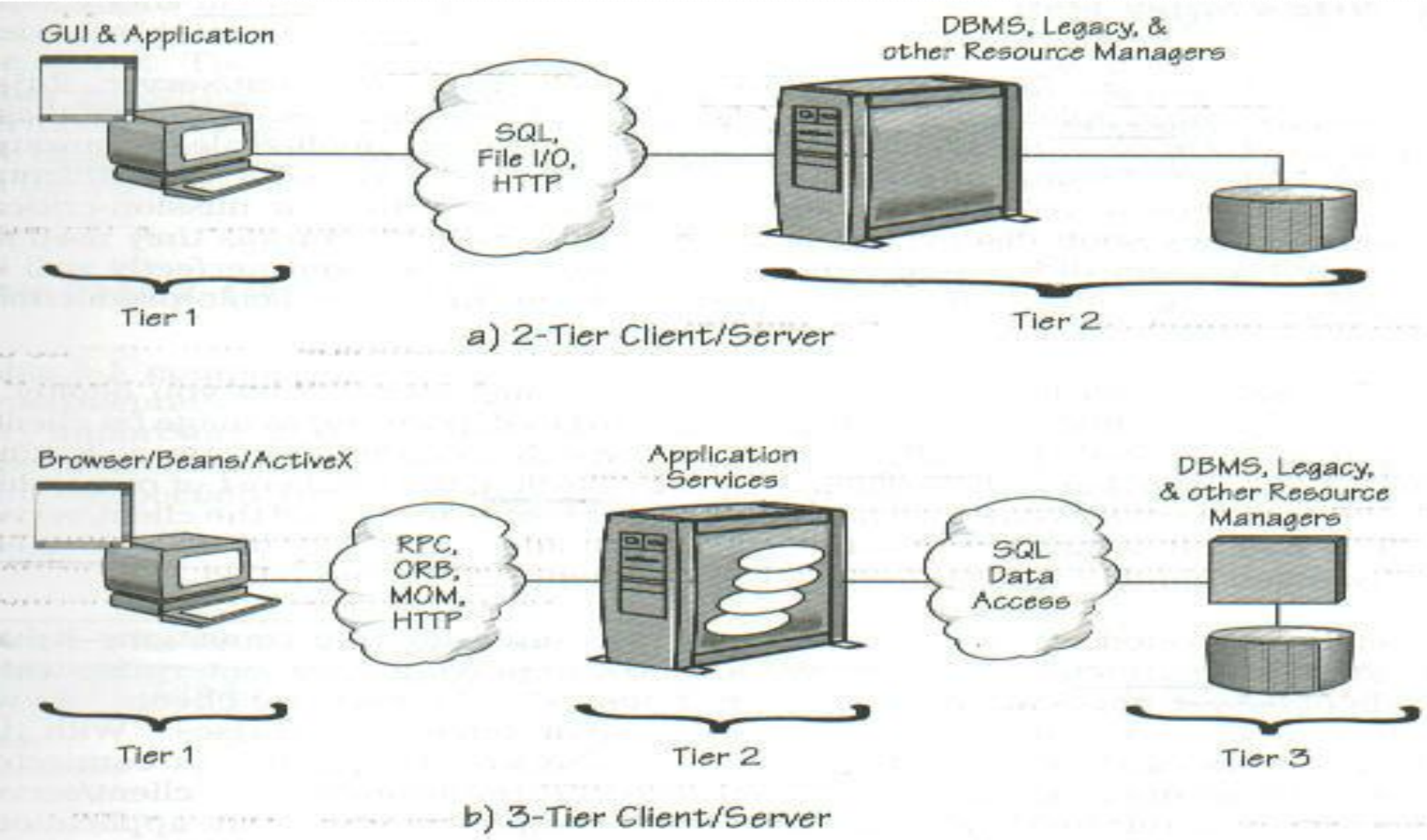
Arquitetura *Multi-tier*



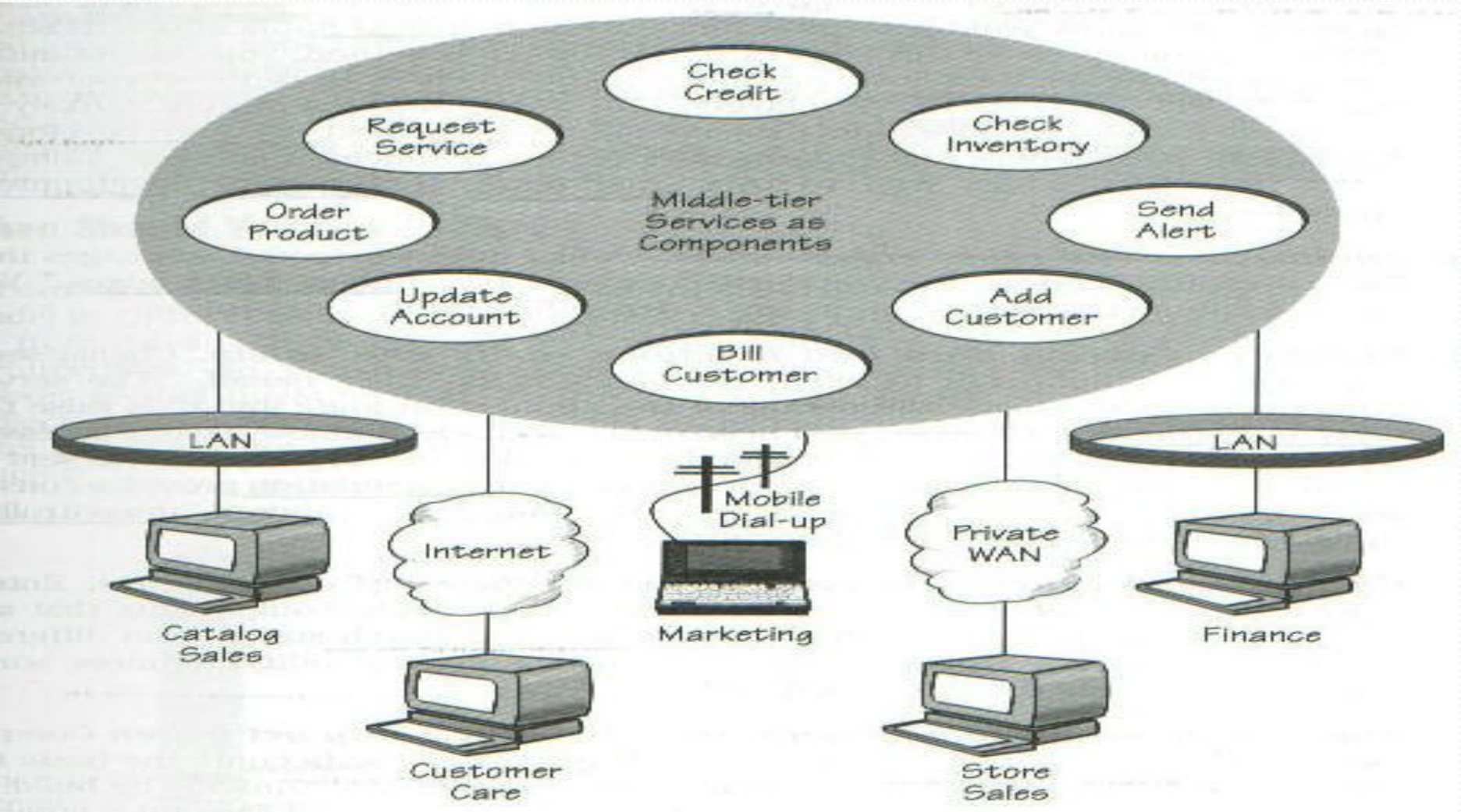
Arquitetura *Multi-tier*



Componentes Objetos em uma Arquitetura Cliente-Servidor N-Tier



Componentes Objetos em uma Arquitetura Cliente-Servidor N-Tier



COM

Visão crítica
(Mercado)

DCOM
(Distributed Component Object Model)

Distributed Component Object Model (DCOM) é a proposta da Microsoft para fazer frente a tecnologia CORBA de *middleware.*

Essa abordagem se iniciou como uma tecnologia de estruturação de documentos, denominada de *Object Linking and Embedding* (OLE). OLE está presente na maioria das aplicações Windows, mais tarde foi transformada em uma tecnologia mais genérica e orientada a objetos chamada de *Component Object Model* (COM).

COM
(Component Object Model)

COM está longe de ser um *middleware*, pois não define um protocolo de interoperabilidade, embora possua uma linguagem genérica de definição de interfaces.

Dessa forma, COM pode ser usada para definir interfaces comuns entre componentes de software, mas com a restrição de que ela só pode vincular os componentes que residem na mesma máquina.

COM
(Component Object Model)

Com o objetivo de criar um produto *middleware*, em 1996, a Microsoft estendeu e aperfeiçoou COM resultando na tecnologia DCOM.

A idéia inicial era fornecer uma tecnologia para plataformas Windows e não-Windows, mas apenas algumas implementações não-Windows apareceram.

Em meados de 1990, a Microsoft criou alguns produtos para fornecer serviços de *middleware* para a plataforma DCOM.

DCOM
(Distributed Component Object Model)

Entre os produtos, o *Microsoft Message Queue Server* (MSMQ) para comunicação assíncrona, e o *Microsoft Transaction Server* (MTS), um monitor de transações.

Em 1997, a Microsoft novamente anunciou um novo modelo de componentes conhecido como COM+

COM+

O modelo de componentes COM+ estende o modelo COM adicionando diversas melhorias, entre elas o conceito de programação baseada em atributos, ou *attribute-based programming*.

Em meados do ano de 2000 foi anunciada uma nova estratégia para o desenvolvimento de sistemas distribuídos, conhecida como *Distributed interNetwork Architecture* (DNA) que utiliza o modelo de componentes COM+ e que está incorporada ao sistema operacional Windows 2000.

DNA e .NET

Atualmente, a Microsoft está migrando sua arquitetura DNA para incluir o framework .NET, uma nova especificação para o desenvolvimento de aplicações distribuídas para Internet que possui fortes fundamentos em XML e SOAP (*Simple Object Access Protocol*) .

Resumo sobre o COM

De uma forma simplificada, pode-se visualizar o DCOM como a especificação de um ORB específico para a plataforma Windows.

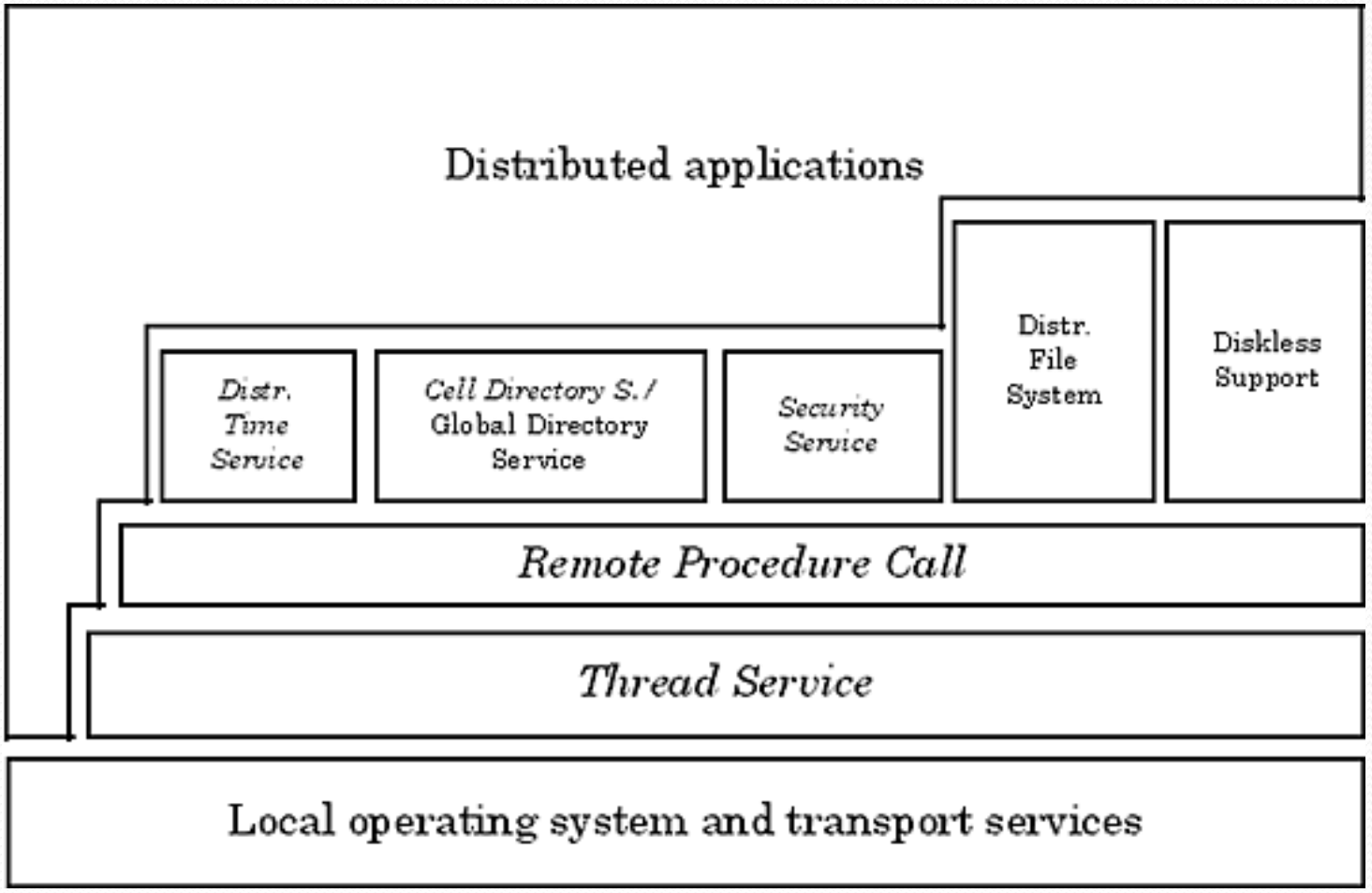
Tal como CORBA, o padrão DCOM separa a interface de um objeto de sua implementação e requer que todas as interfaces sejam declaradas utilizando uma IDL.

Resumo sobre o COM

No caso do DCOM o IDL denomina-se MIDL (Microsoft Interface Definition Language), que especifica os requisitos mínimos dos objetos para que estes possam se comunicar através do barramento DCOM.

A MIDL é baseada no padrão DCE (Distributed Computing Environment) e, portanto, não é compatível com CORBA.

Distributed Computing Environment (DCE)



DCOM
(Distributed Component Object Model)

Da mesma forma que CORBA, DCOM também provê invocação de métodos através de interfaces estáticas e dinâmicas.

O *Type Library* é uma versão do Repositório de Interfaces de CORBA. Os clientes consultam o *Type Library* para localizar quais interfaces um objeto suporta e quais parâmetros são necessários para invocar um método em particular.

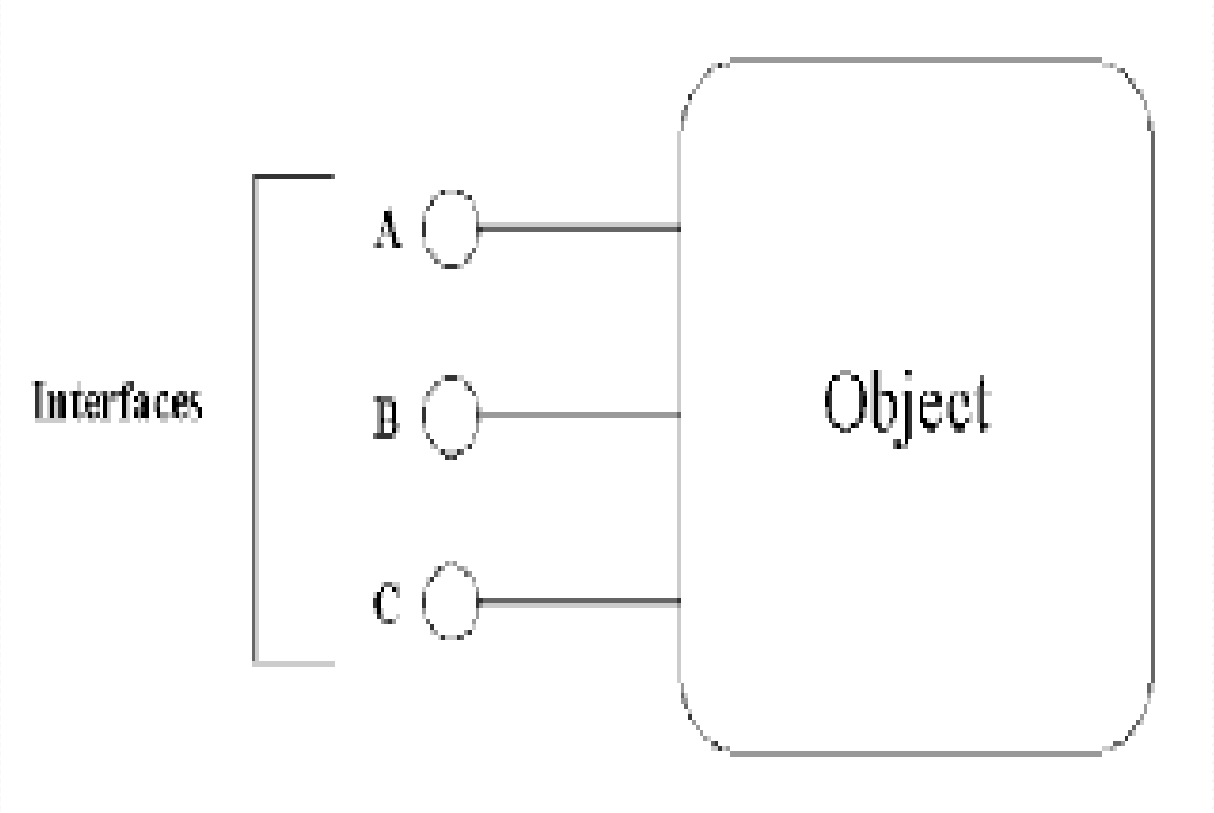
DCOM também oferece um registro e alguns serviços de localização que são similares aos de CORBA.

DCOM
(Distributed Component Object Model)

Em DCOM, uma interface é simplesmente um conjunto de funções, métodos ou funções membros, que serve como um contrato entre a aplicação cliente e o servidor de objetos, através do qual podem trocar mensagens.

Na interface, os métodos são definidos independentemente de sua implementação (e de linguagem), numa API binária, acessível por qualquer linguagem de programação que suporte a especificação.

DCOM
(Distributed Component Object Model)

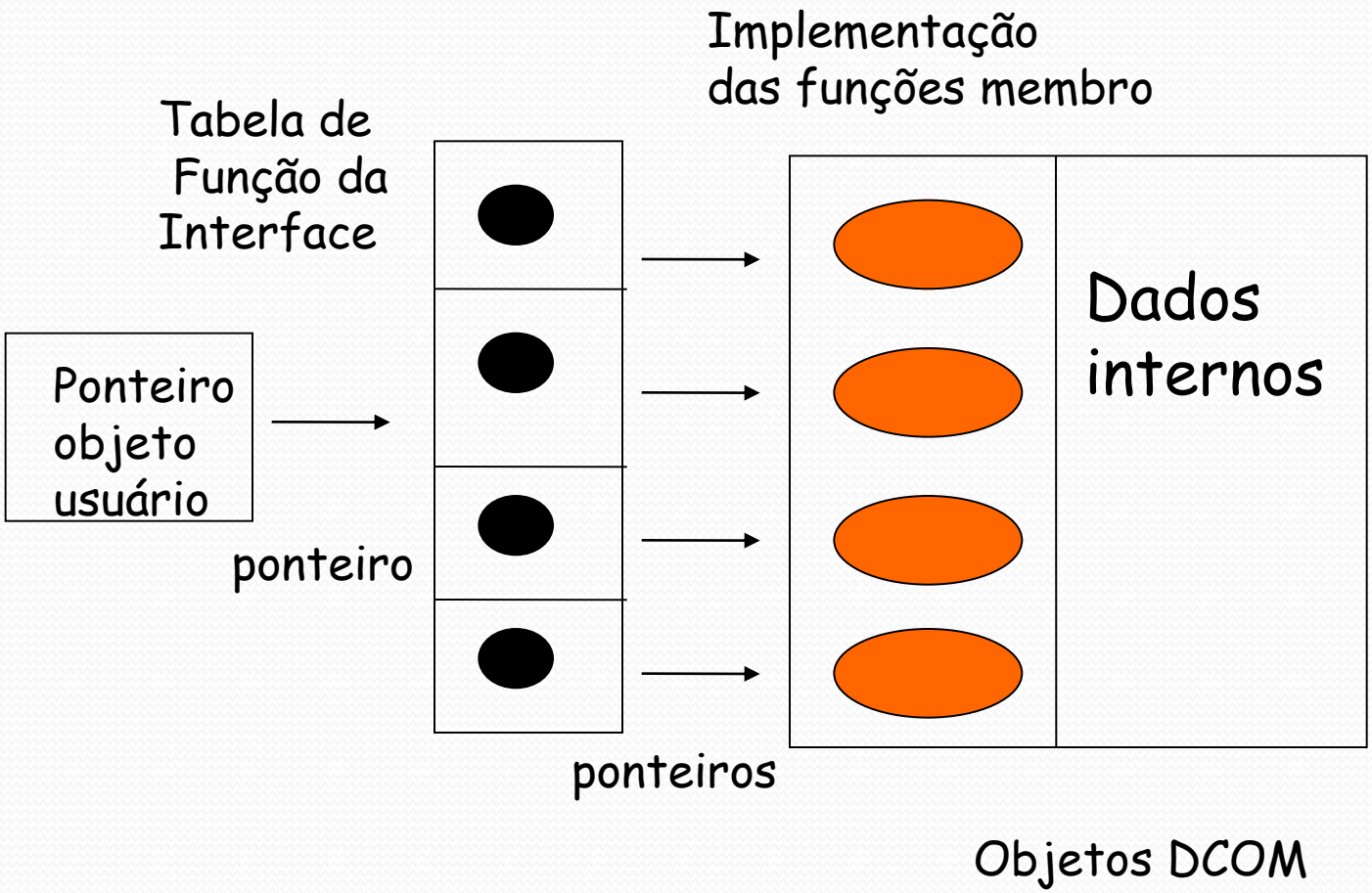


DCOM
(Distributed Component Object Model)

Para acessar uma interface, um cliente DCOM utiliza um ponteiro para uma tabela de ponteiros conhecida como tabela virtual (vtable).

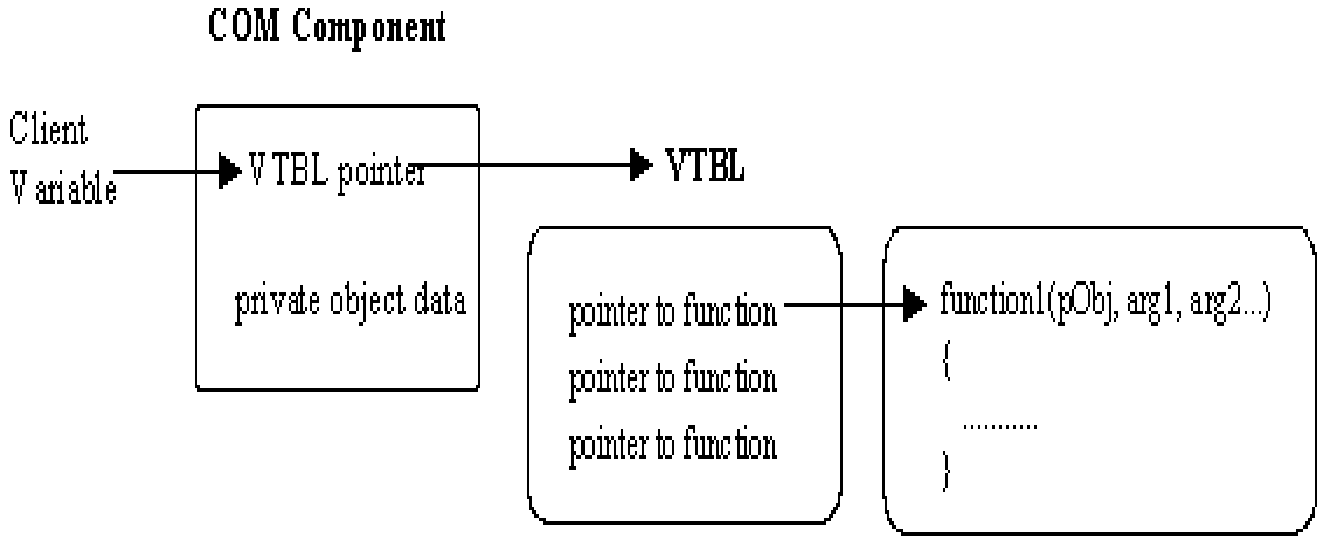
As funções membros apontadas pela vtable são as implementações dos métodos do objeto. Cada objeto DCOM possui uma ou mais vtables que definem o contrato entre a implementação do objeto e seus clientes.

DCOM
(Distributed Component Object Model)



Interface DCOM

DCOM
(Distributed Component Object Model)



DCOM
(Distributed Component Object Model)

No modelo, cada interface é identificada por um identificador único conhecido como IID (Interface IDentifier), um número de 128 bits gerado pela API DCOM.

DCOM
(Distributed Component Object Model)

Um objeto DCOM – também conhecido como objeto ActiveX – é definido como um componente que suporta uma ou mais interfaces definidas por sua classe.

DCOM
(Distributed Component Object Model)

Uma interface DCOM refere-se a um grupo pré-definido de funções relacionadas.

Uma classe DCOM implementa sempre pelo menos uma interface e é identificada por um número único de 128 bits, chamado de Class ID ou CLSID.

Um objeto DCOM é uma instância de uma classe. Um objeto implementará todas as funções definidas na interface que sua classe suporta.

DCOM
(Distributed Component Object Model)

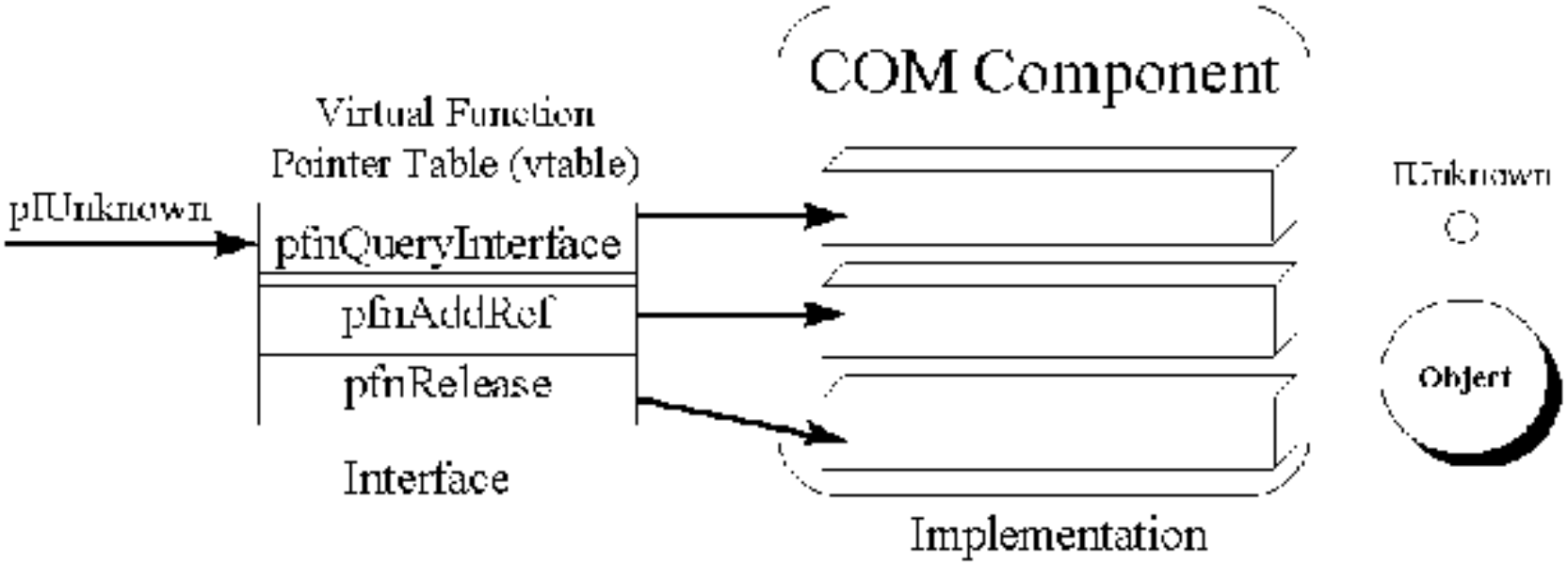
Os clientes sempre se comunicam com os objetos DCOM através de ponteiros para as interfaces; nunca através de acesso direto.

Os objetos DCOM não suportam identificadores únicos de objetos (ou referências persistentes de objetos no modelo CORBA).

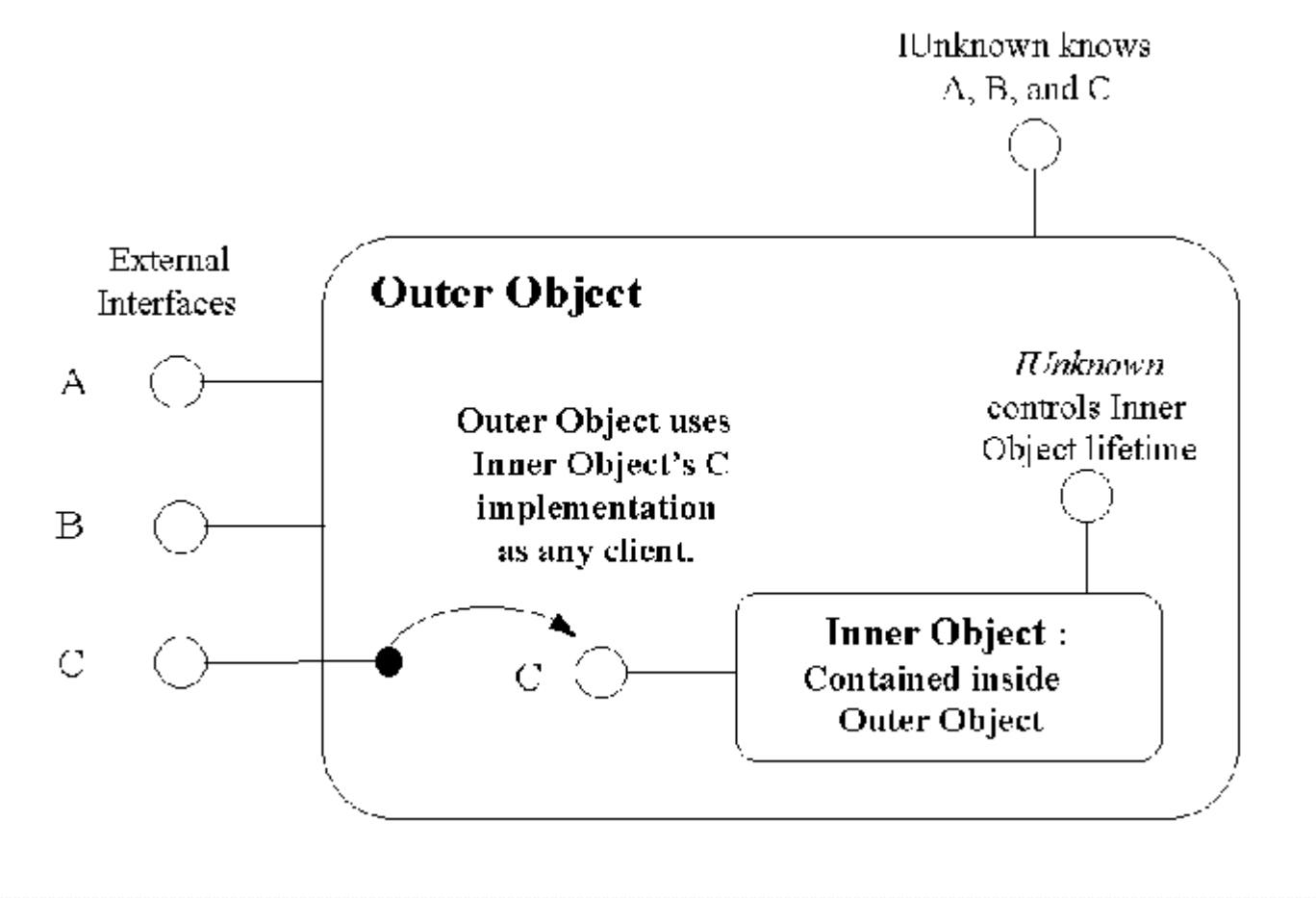
DCOM
(Distributed Component Object Model)

No modelo, todas as classes devem implementar, obrigatoriamente, a interface pré-definida IUnknown, que é responsável pela criação e remoção dos objetos, além de permitir acesso aos demais membros da interface específica da classe.

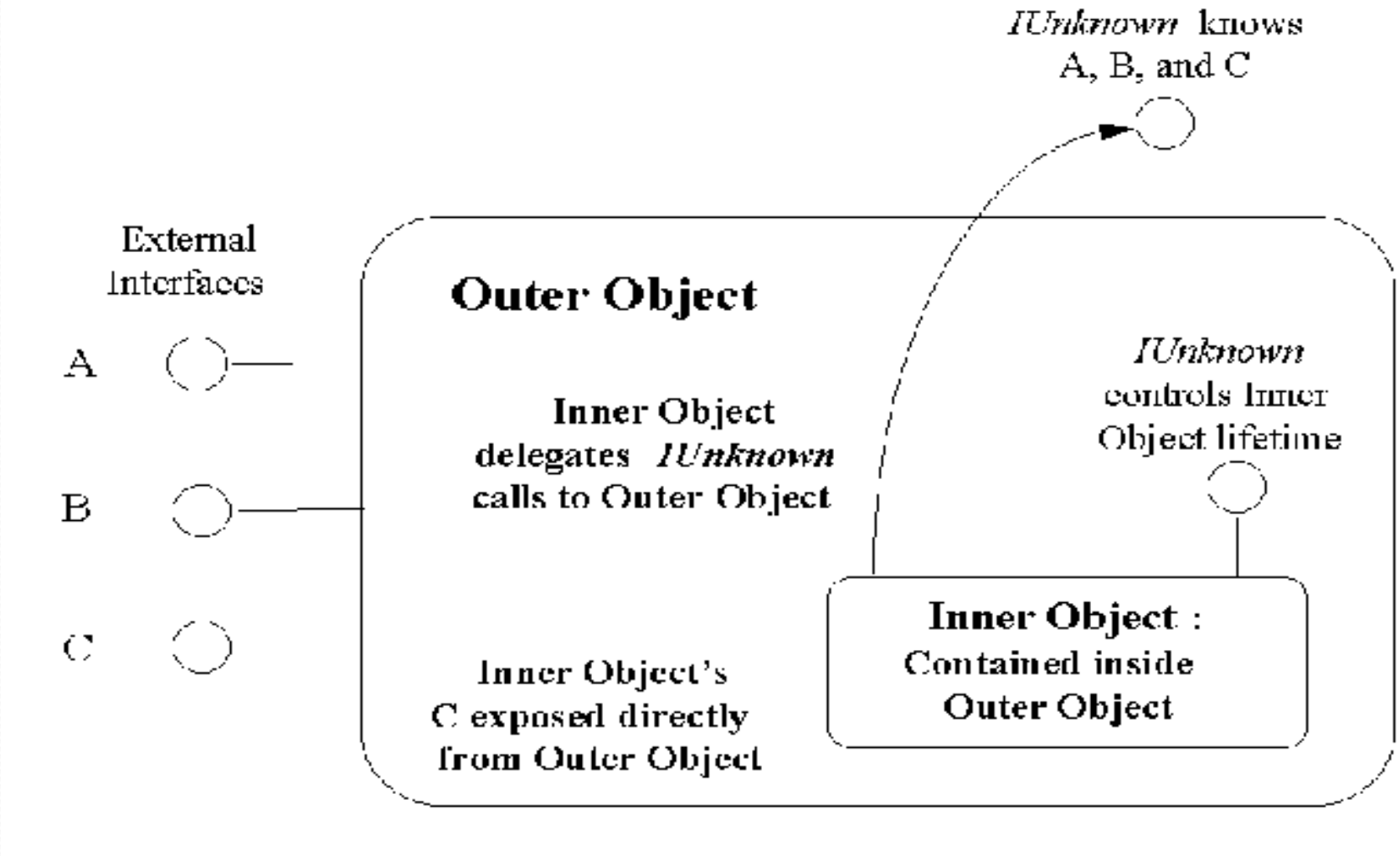
IUnknown


COM Component
Virtual Function
Pointer Table (vtable)
pIUnknown
pfnQueryInterface
pfnAddRef
pfnRelease
Interface
IUnknown
Object
Implementation

DCOM
(Distributed Component Object Model)



DCOM
(Distributed Component Object Model)



DCOM
(Distributed Component Object Model)

Ao contrário do modelo CORBA, o modelo DCOM não suporta o conceito de identificador de objetos (object ID).

Os clientes obtêm um ponteiro para uma interface e não um ponteiro para um objeto com estado.

Um cliente DCOM não pode acessar exatamente o mesmo objeto posteriormente. Os clientes possuem apenas ponteiros transientes para as interfaces.

DCOM
(Distributed Component Object Model)

Em outras palavras, objetos DCOM não mantêm o estado entre conexões ou acessos.

E este pode se tornar um grande problema em ambientes onde existem falhas de comunicação, como por exemplo na Internet.

DCOM
(Distributed Component Object Model)

Para contornar os problemas com persistência de objetos no ambiente distribuído, DCOM criou os monikers.

Um moniker DCOM é um objeto que age como um nome alternativo (alias) persistente para outro objeto.

DCOM
(Distributed Component Object Model)

Os monikers podem prover nomes alternativos, ou aliases, para sistemas de arquivos distribuídos, consultas em bancos de dados, células de uma planilha eletrônica, computadores remotos, entre outros.

Um moniker, no entanto, é uma solução paliativa adotada para a falta de suporte de DCOM para persistência de objetos.

DCOM
(Distributed Component Object Model)

Se o componente projetado se destinar a ser utilizado como um servidor (local ou remoto) de objetos, deverá implementar também a interface IClassFactory, que fornece mecanismos para instanciar objetos.

DCOM
(Distributed Component Object Model)

No modelo, ao contrário de implementações como Java, por exemplo, que executam automática e periodicamente o *recolhimento de lixo* (garbage collection) do sistema, DCOM obriga o servidor de objetos a cooperar com o sistema, devolvendo a ele a memória não utilizada, tão logo o último cliente deixe de referenciar um objeto.

DCOM
(Distributed Component Object Model)

Se o componente projetado se destinar a ser utilizado como um servidor (local ou remoto) de objetos, deverá implementar também a interface IClassFactory, que fornece mecanismos para instanciar objetos.

DCOM
(Distributed Component Object Model)

No modelo, ao contrário de implementações como Java, por exemplo, que executam automática e periodicamente o *recolhimento de lixo* (garbage collection) do sistema, DCOM obriga o servidor de objetos a cooperar com o sistema, devolvendo a ele a memória não utilizada, tão logo o último cliente deixe de referenciar um objeto.

DCOM
(Distributed Component Object Model)

O padrão especifica servidores de objetos de três tipos diferentes:

a) Servidores in-process: executam no mesmo processo de seus clientes. No sistema operacional Windows 9x e Windows NT/2000 estes servidores são implementados como bibliotecas do tipo DLL (Dynamic Linked Library);

DCOM
(Distributed Component Object Model)

O padrão especifica servidores de objetos de três tipos diferentes (cont.):

b) Servidores locais: executam em processos distintos de seus clientes, mas na mesma máquina. Os clientes utilizam o mecanismo LRPC (Lightweigth Remote Procedure Call) para se comunicar com o servidor local.

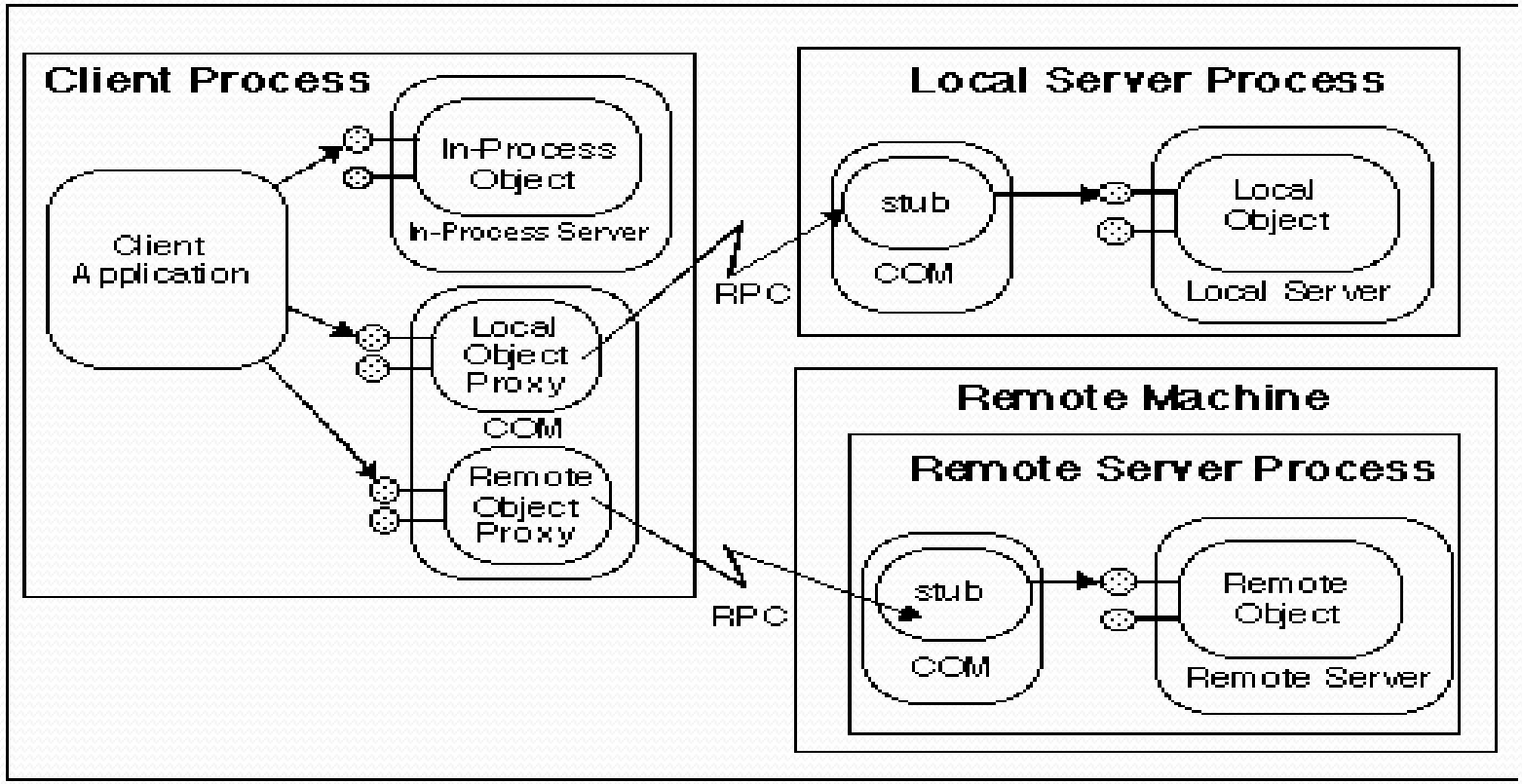
São implementados como arquivos executáveis (.EXE);

DCOM
(Distributed Component Object Model)

O padrão especifica servidores de objetos de três tipos diferentes (cont.):

c) Servidores remotos: executam em processos separados, residentes em máquinas remotas, possivelmente com outro sistema operacional. Os clientes utilizam o mecanismo de RPC para se comunicar com os servidores remotos.

DCOM
(Distributed Component Object Model)



Servidores do Modelo DCOM.

DCOM
(Distributed Component Object Model)

Com essa abordagem a especificação DCOM garante comunicação transparente entre a aplicação cliente e o servidor de objetos (local ou remoto).

Do ponto de vista do cliente, todo o acesso aos membros da Classe é feito através de ponteiros para os elementos da interface.

Por definição, um ponteiro deve ser inprocess. Assim, a comunicação com um servidor in-process local é feita diretamente.

DCOM
(Distributed Component Object Model)

Para a comunicação out-of-process (local ou remota) o barramento DCOM gera um objeto de comunicação, conhecido como proxy object, que recebe a chamada e converte numa chamada RPC apropriada, enviando-a ao processo externo (local ou remoto).

O elemento DCOM responsável pela localização dos servidores e pela comunicação RPC entre clientes e servidores é chamado de *Service Control Manager* (SCM) – ou simplesmente *Scum*.

DCOM
(Distributed Component Object Model)

Na outra ponta da mensagem, o barramento DCOM gera um stub object, que recebe a chamada do proxy object e a converte numa chamada local à interface solicitada.

Assim, tudo se passa transparentemente no barramento DCOM. Os clientes e os servidores sempre se comunicam Utilizando algum código local ou in-process.

Os objetos proxy e o mecanismo de stubs são muito similares ao modo como CORBA implementa a transparência local/remota utilizando stubs estáticos no cliente e interfaces de skeletons no servidor.

DCOM
(Distributed Component Object Model)

O modelo DCOM também provê facilidades de invocações dinâmicas e obtenção de metadados. Essas facilidades são conhecidas como automatizações (automations).

O Type Library de DCOM permite que clientes descubram dinamicamente métodos e propriedades de um servidor DCOM.

Para que um servidor disponibilize metadados na Type Library, ele precisa ter suas interfaces descritas através da ODL (Object Definition Language).

DCOM
(Distributed Component Object Model)

DCOM ODL é um subconjunto da Microsoft IDL e é utilizada para se definir metadados de uma classe.

Um servidor automatizado é um objeto DCOM que implementa uma interface específica chamada de IDispatch.

Essa interface fornece mecanismos de late-binding através dos quais um objeto expõe suas funções – incluindo seus métodos e propriedades. Essas funções de late-binding são conhecidas como dispatch interfaces, ou simplesmente dispinterfaces (em CORBA são equivalentes às dynamic skeleton interfaces).

DCOM
(Distributed Component Object Model)

As interfaces, métodos, e propriedades dos servidores são armazenadas em type libraries que agem como repositórios de execução.

Do mesmo modo que CORBA, DCOM não tem sido encontrado em larga escala. Pelo fato de ser uma arquitetura dependente da plataforma Windows, poucas aplicações com caráter mais sério, utilizando DCOM, estão sendo implementadas.

DCOM
Resumo Crítico

Os objetos DCOM são ponteiros para interfaces sem estado persistente e os monikers são uma solução alternativa.

Em adição, DCOM requer serviços adicionais como o Microsoft Transaction Server (MTS) para permitir alta escalabilidade no servidor.

Com o surgimento de novas tecnologias, como o framework .NET e tecnologias já existentes como DNA e COM+, espera-se uma maior portabilidade em relação a DCOM e uma robustez comparável a da arquitetura CORBA.

Conteúdo Programático

. Serviços CORBA

. Objetos COM

. Barramento de objetos COM

. Serviços OLE/COM

COM

Visão Tecnológica (Microsoft)

COM (Component Object Model)

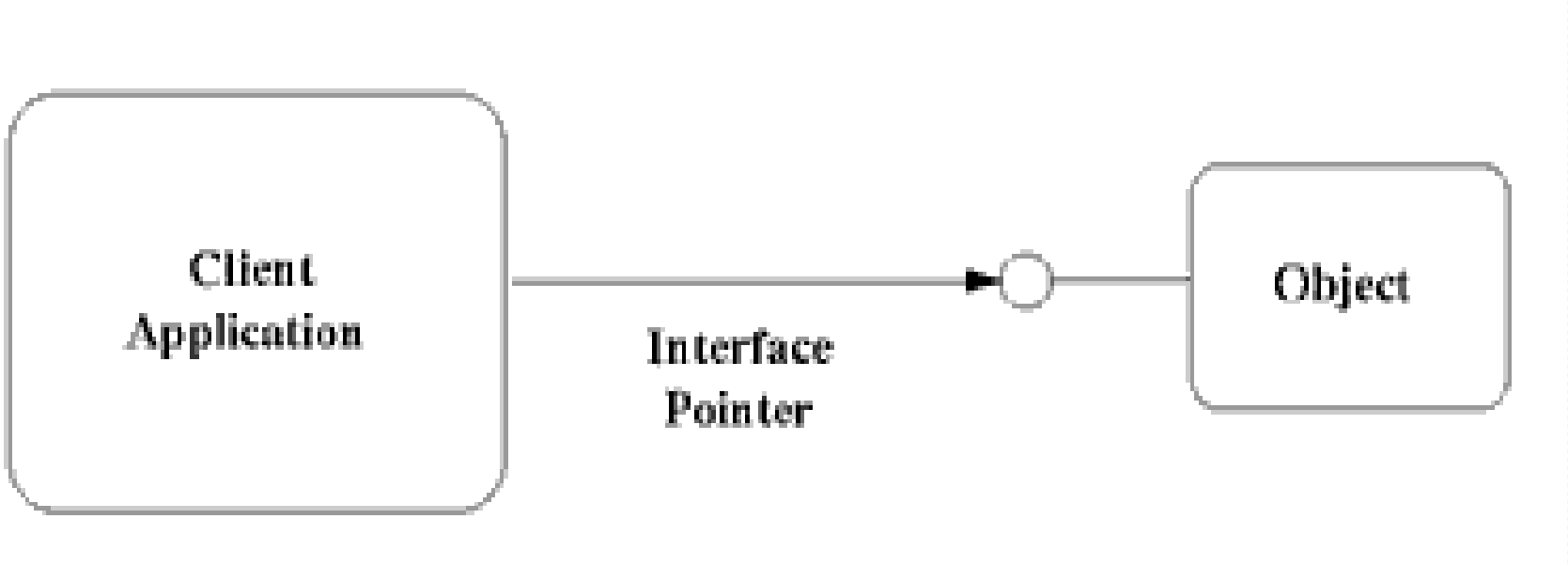
COM refere-se a *especificação* e a *implementação* desenvolvidas pela Microsoft que prove uma abordagem de integração de componentes.

A abordagem suporta a *interoperabilidade* e *reusabilidade* de objetos distribuídos permitindo desenvolvedores construir sistemas montando componentes de diferentes fabricantes, desde que utilizem o *COM*. Assim é possível a construção de sistemas com componentes já existentes.

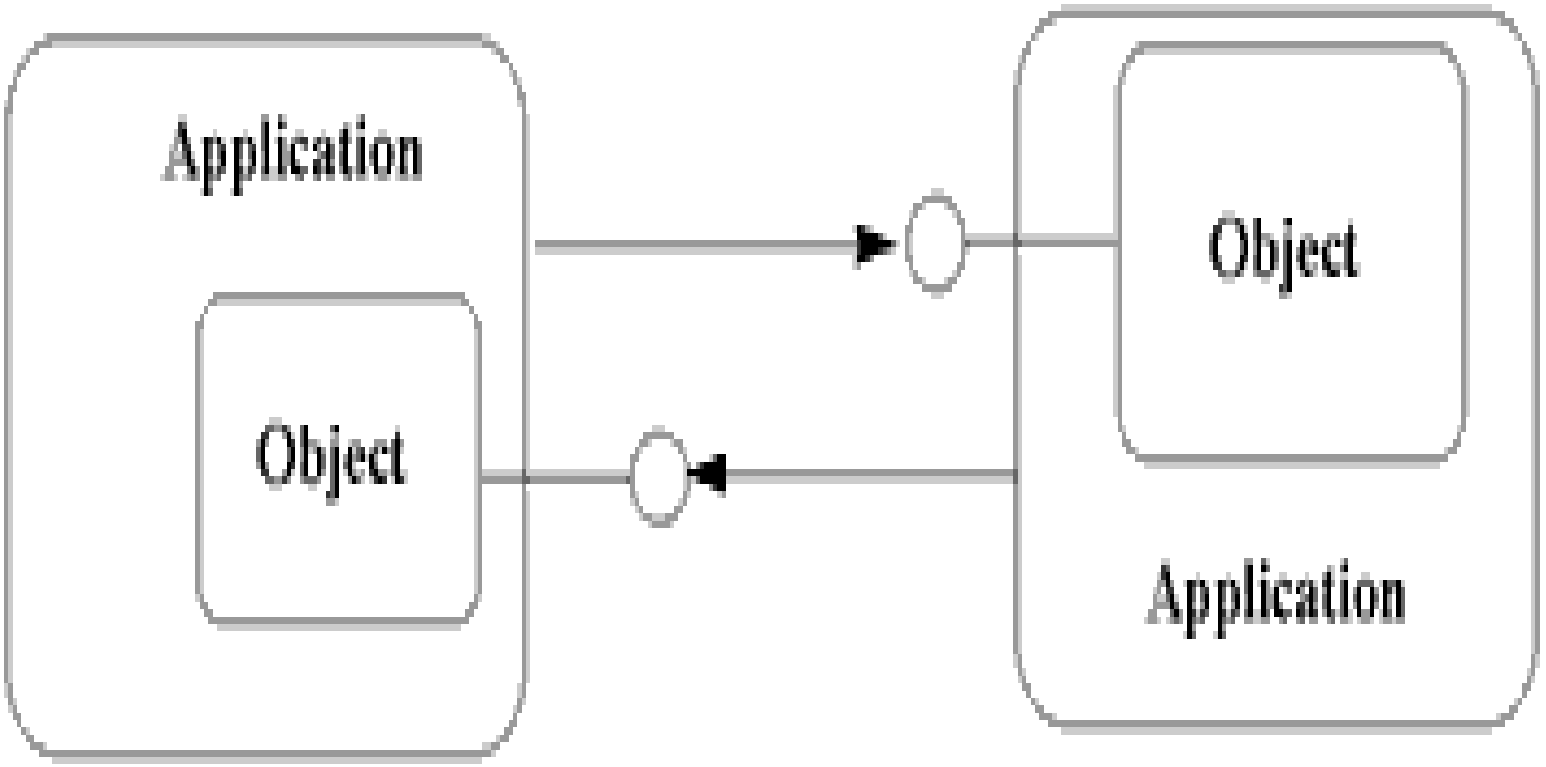
COM (Component Object Model)

COM define uma interface de programação de aplicação (API) para permitir a criação de componentes para o uso de integração de aplicações de usuários ou para permitir que componentes possam interagir.

COM (Component Object Model)



COM (Component Object Model)



COM (Component Object Model)

Todavia, para que ocorra a interação no *COM*, os componentes têm que aderir a estrutura binária especificada pela Microsoft.

Desta forma, os componentes escritos em diferentes linguagens podem interoperar.

COM (Component Object Model)

Segundo a Microsoft [www.microsoft.com/com],a tecnologia COM, dentro da família de sistemas operacionais Windows, permite que a comunicação entre componentes de software.

A família COM inclui COM+, DCOM e o ActiveX® Controls.

COM (Component Object Model)

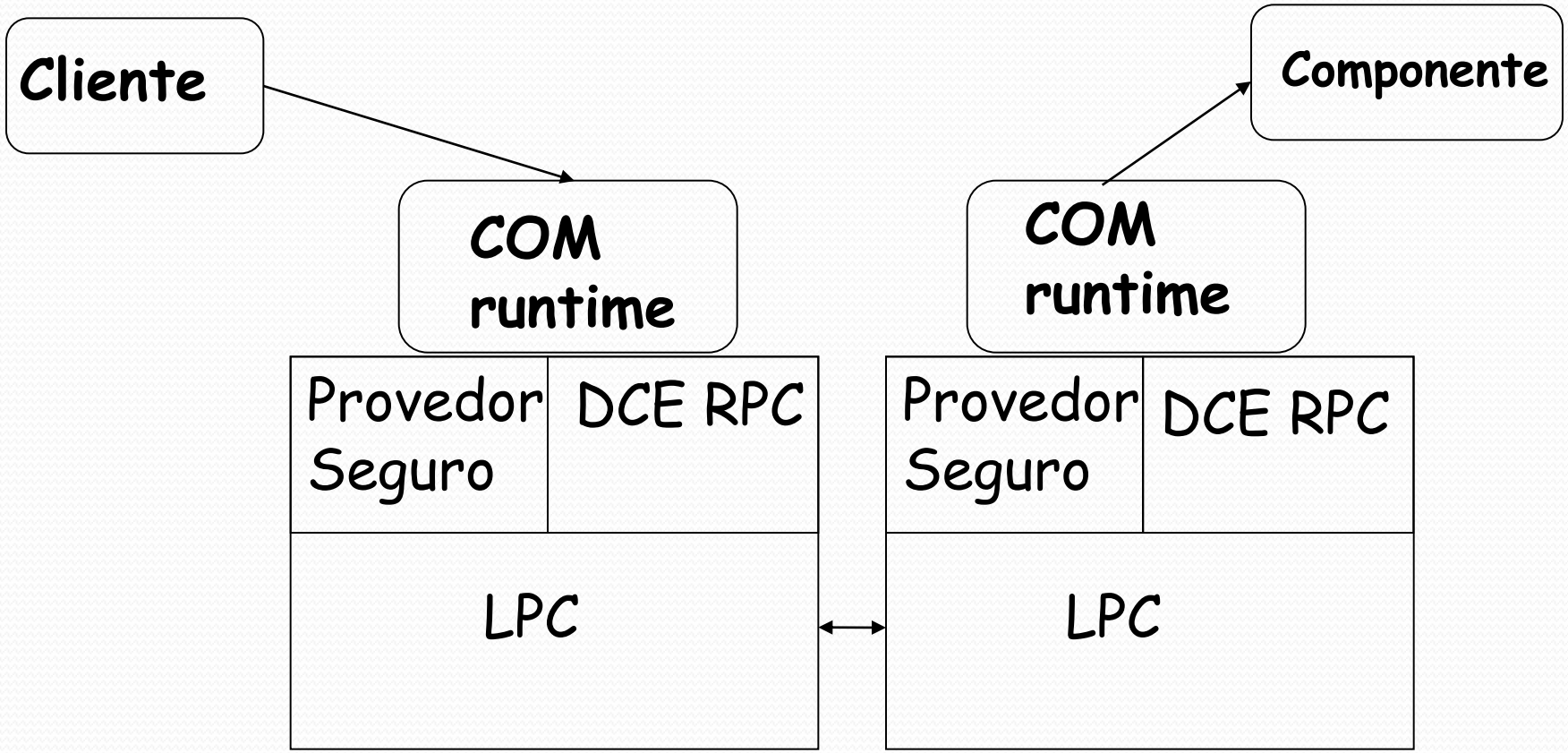
Um exemplo de uso do COM é a interação entre aplicações dos produtos do ambiente Microsoft Office.

O COM OLE (Object Linking and Embedding) permite que um documento WORD seja dinamicamente relacionado a uma planilha Excel.

COM (Component Object Model)

Um outro exemplo seria a automatização permitida pelo COM para que usuários construam scripts em suas aplicações para efetuar tarefas repetidas ou controle de uma aplicação através de outra.

COM (Component Object Model)



OLE (Object Linking and Embedding)

O OLE *(Object Linking and Embedding)* é construído com COM, prove serviços tais como *linking* e *embedding* que são usados na criação de documentos Compostos (documentos gerados por múltiplas fontes de ferramentas);

COM+

O COM+ é uma evolução do COM que integra os serviços do MTS e fila de mensagem. Assim, torna o COM mais fácil de ser programado através de linguagem de programação da Microsoft tais como Visual Basic, Visual C++, e J++.

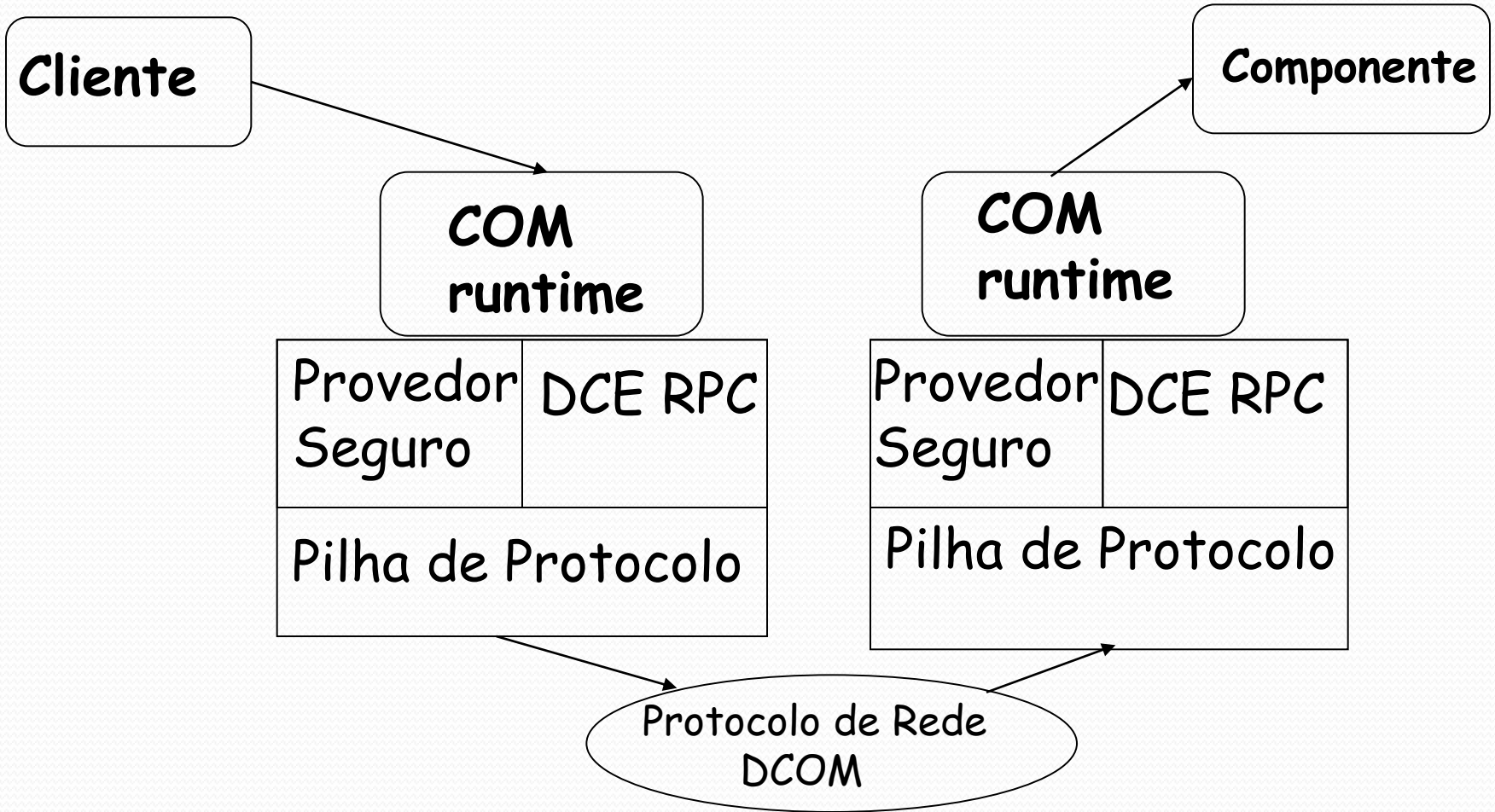
COM+ também esconde algumas complexidades da codificação COM.

DCOM
(Distributed Component Object Model)

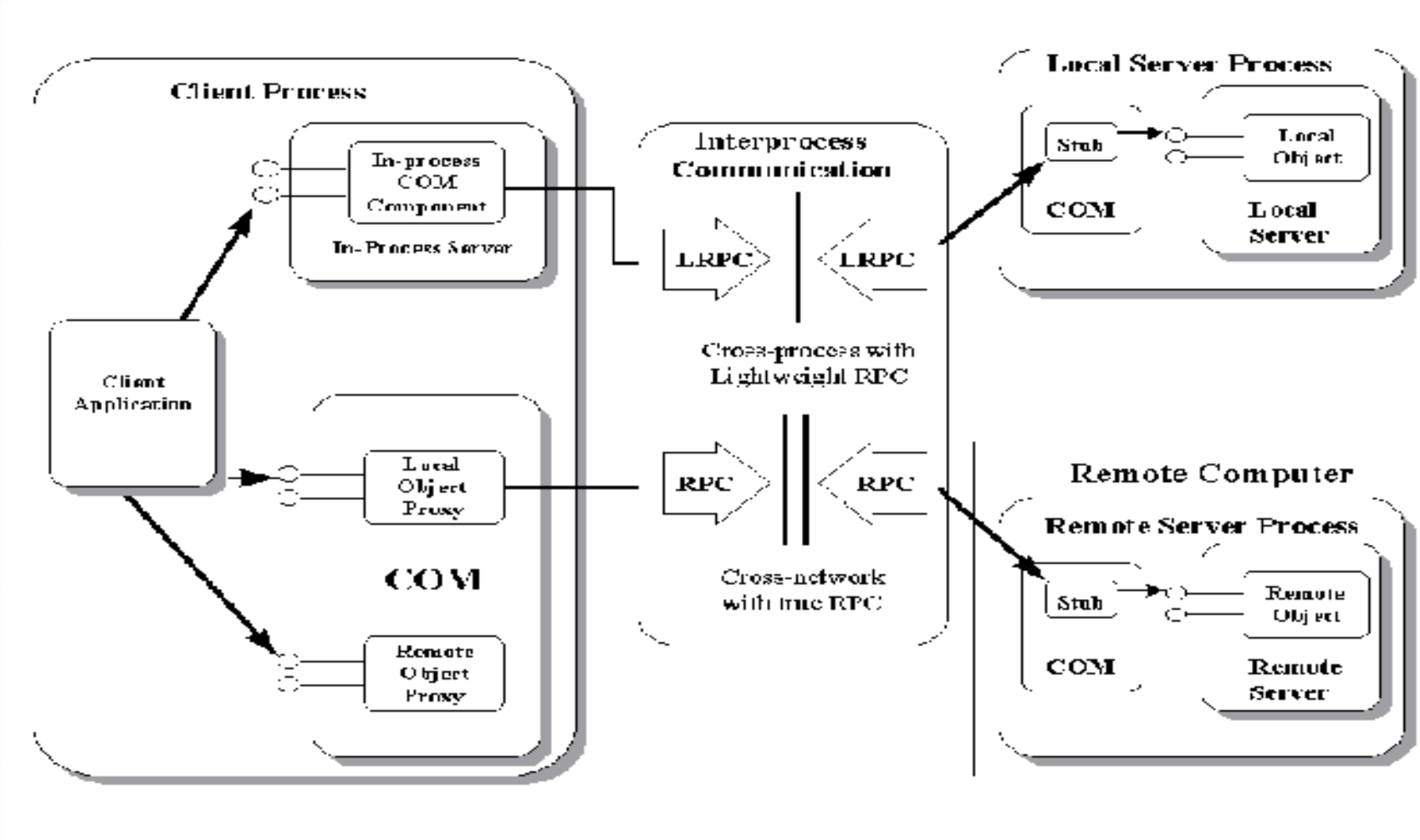
DCOM é o acrônimo para *Distributed Component Object Model* desenvolvido pela Microsoft.

DCOM suporta objetos remotos através da utilização do protocolo conhecido como ORPC (*Object Remote Procedure Call*), é independente de linguagem, MAS não é compatível entre qualquer plataforma.

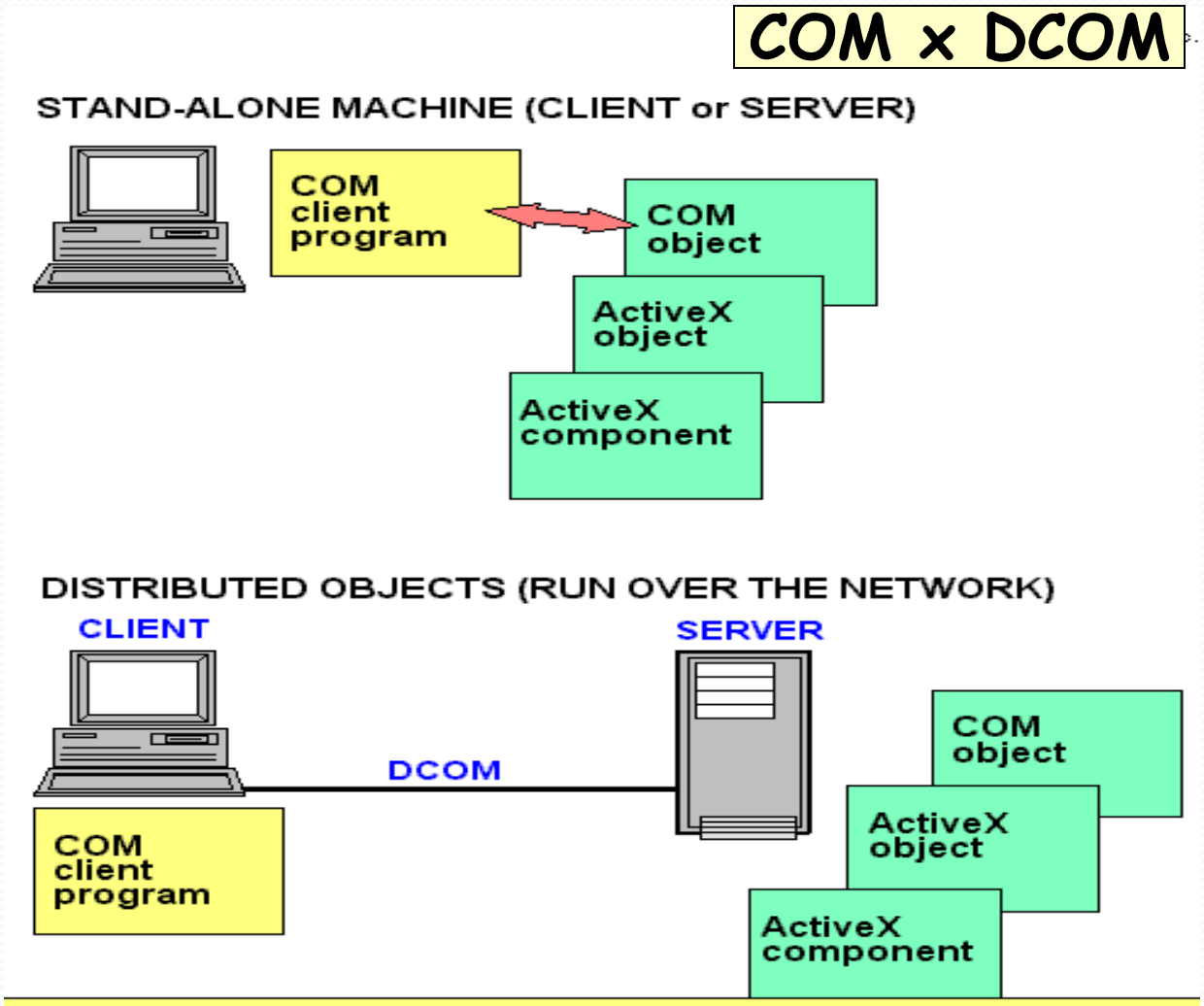
DCOM (Distributed Component Object Model)



DCOM (Distributed Component Object Model)



COM vs DCOM


COM x DCOM
STAND-ALONE MACHINE (CLIENT or SERVER)
COM client program
COM object
ActiveX object
ActiveX component
DISTRIBUTED OBJECTS (RUN OVER THE NETWORK)
CLIENT
SERVER
DCOM
COM client program
COM object
ActiveX object
ActiveX component

DCOM
(Distributed Component Object Model)

DCOM é uma extensão do COM que permite componentes baseados na abordagem de rede possam interagir. O COM deve ser executado na mesma máquina e pode considerar diferentes endereços de memória local.

Por outro lado, o DCOM estende o uso de diferentes espaços de memória dispersos em uma rede.

DCOM
(Distributed Component Object Model)

Com o DCOM, componentes de uma grande gama de plataformas podem interagir, desde que o DCOM exista dentro do ambiente.

Pode-se considerar que COM e DCOM são uma *tecnologia única*, que fornece uma série de serviços para a interação de componentes em uma mesma plataforma e até entre *diferentes* plataformas em uma rede.

Em outras palavras, os paradigmas COM e DCOM existem em um mesmo módulo de *runtime*, esse prove acesso a componentes locais e remotos.

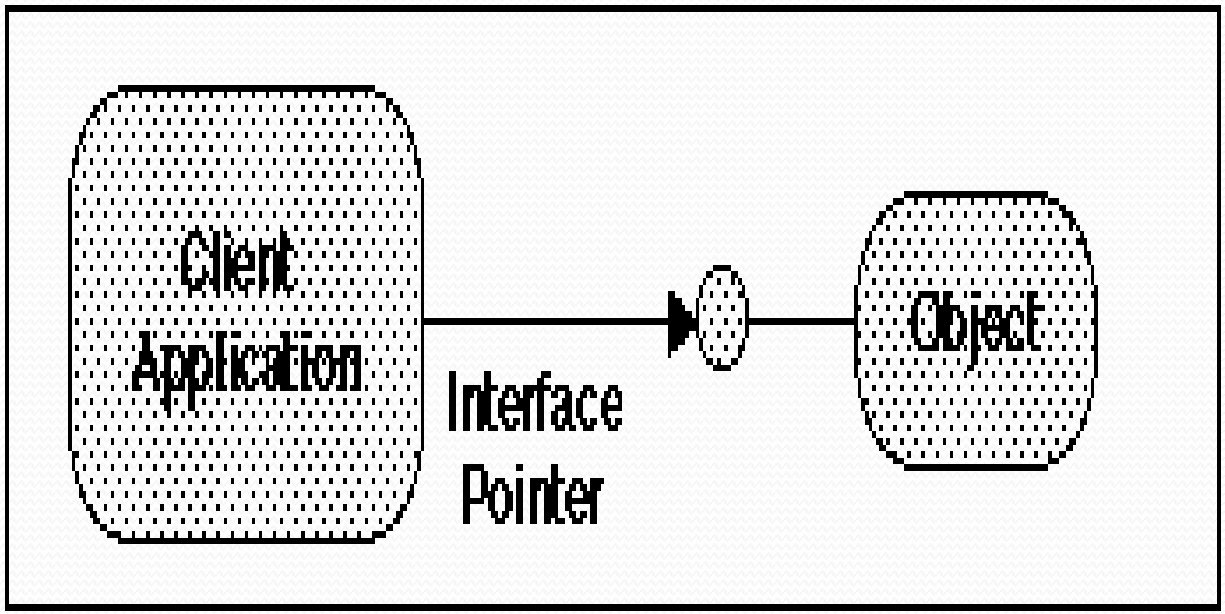
ActiveX

O *ActiveX* expande capacidades básicas para permitir que componentes sejam embutidos em sítios Web;

MTS (Microsoft Transaction Server)

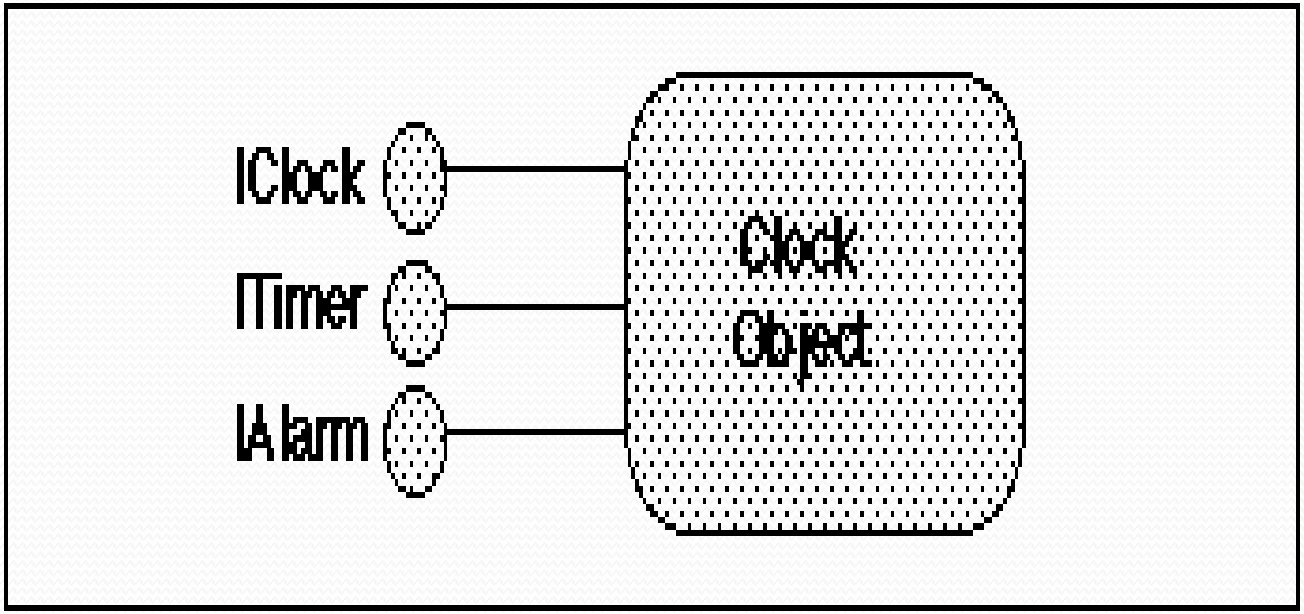
O MTS expande a capacidade do COM com serviços Corporativos, tais como transações e segurança utilizados em sistema EIS (*Enterprise Information Systems*), permitindo o uso de componentes COM;

COM (Component Object Model)



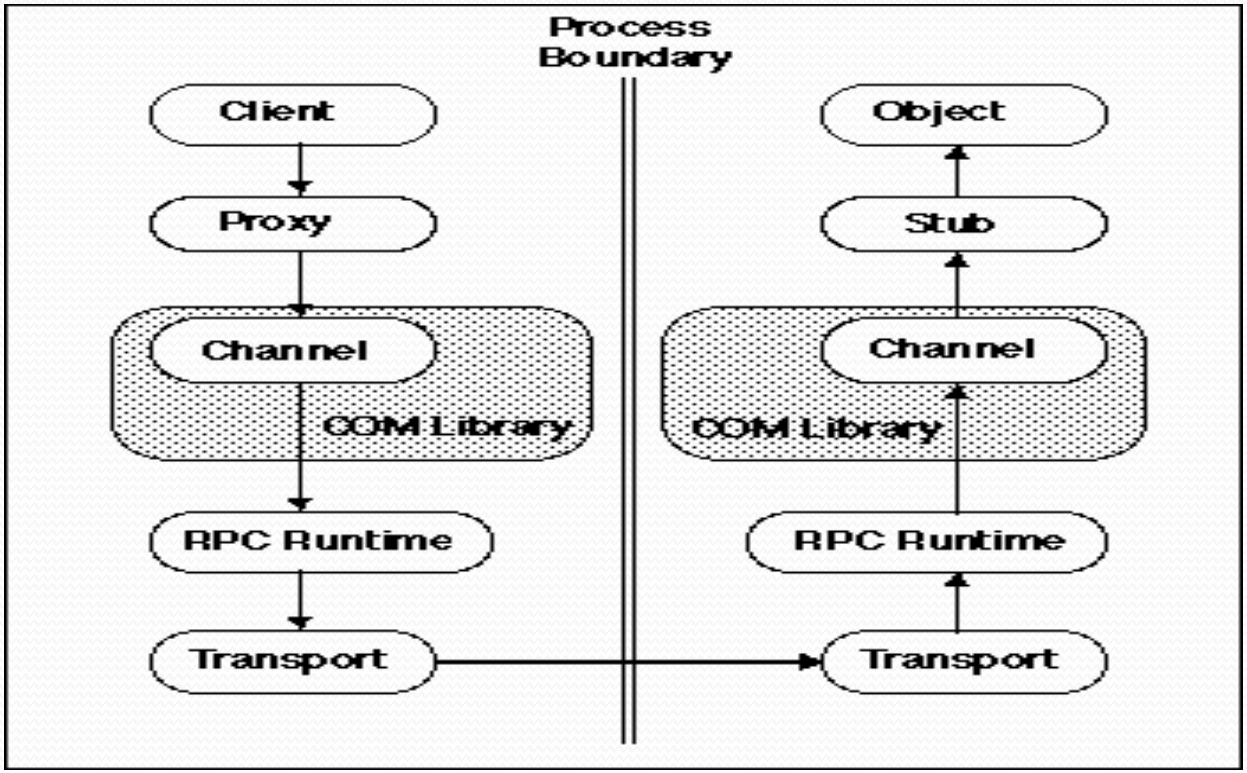
Cliente usando *COM Objeto* através de uma interface ponteiro.

COM (Component Object Model)



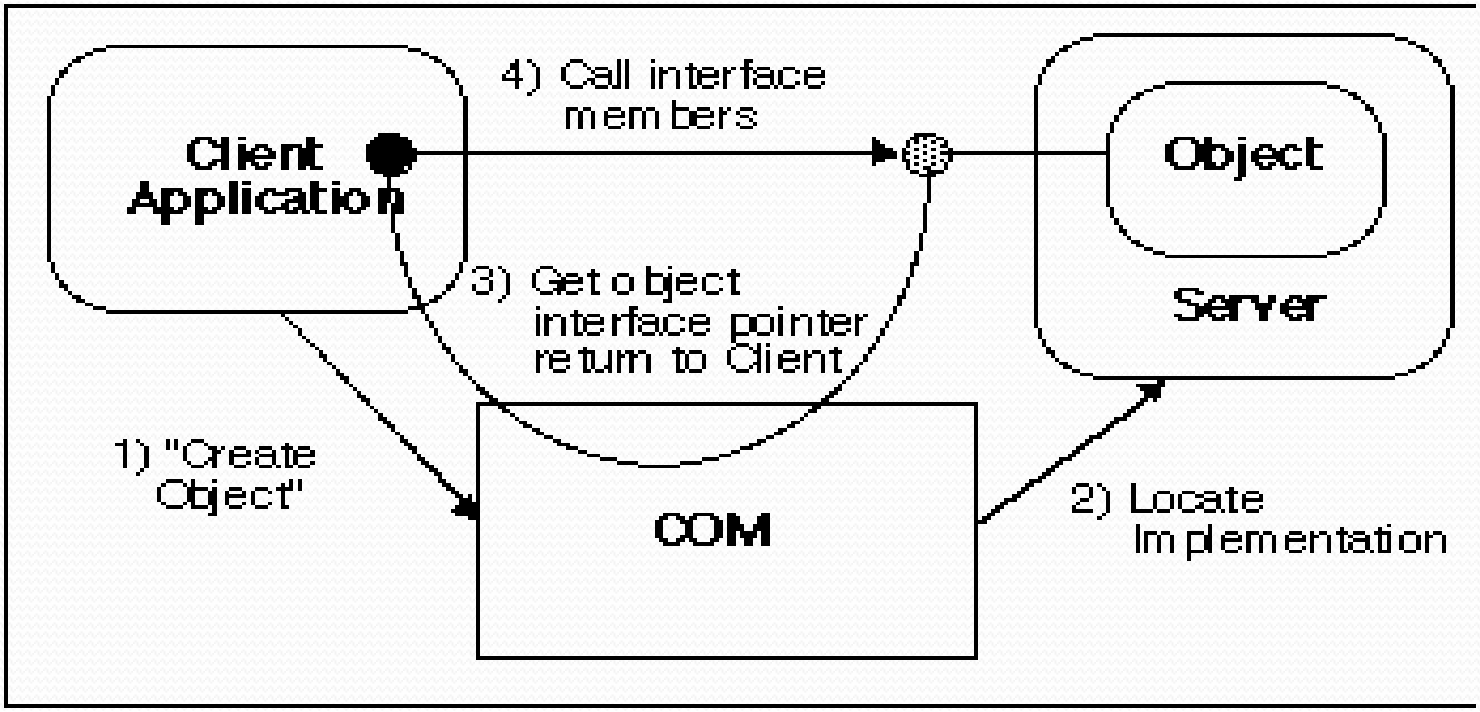
Clock COM objeto

COM (Component Object Model)



Comunicação entre processos em COM

COM (Component Object Model)



Criando um ponteiro objeto COM

COM e DCOM representam técnicas de *baixo nível* que permitem a interação entre componentes.

Por outro lado, OLE, ActiveX e MTS representam serviços de aplicação de alto nível que são construídos no topo do COM e DCOM.

COM e .NET

COM e .NET são tecnologias complementares.

O .NET Common Language Runtime prove uma forma bi-direcional e de integração transparente com o COM.

Isto significa que as aplicações e os componentes COM e .NET podem utilizar funcionalidade uma da outra [www.microsoft.com].

COM e .NET

COM e .NET podem atingir resultados semelhantes.

O framework .NET prove para desenvolvedores um número significante de facilidades, tais como:

- Robustez;
- Gerenciamento de memória automática;
- Modelo de segurança;
- Serviço nativo de serviços Web.

COM e .NET

For new development, Microsoft recommends .NET as a preferred technology because of its powerful managed runtime environment and services [www.microsoft.com].

COM+

COM+ é o nome dos serviços e tecnologias providas primeiramente no Windows 2000.

COM+ traz consigo a tecnologia de componentes COM e aplicações do Microsoft Transaction Server (MTS).

COM+

COM+ pode automaticamente lidar com tarefas complexas tais como agregação de recursos, desconexão de aplicações, publicação de eventos e subscrição/distribuição de transações.

A infra-estrutura COM+ também prove serviços para aplicações e desenvolvedores do .NET, através do *System.EnterpriseServices namespace* do .NET Framework.

Construindo aplicações com COM ou COM+

Para o desenvolvimento de aplicações com COM ou COM+, aonde devo encontrar um maior suporte ?

Construindo aplicações com COM ou COM+

Uma boa fonte de informação para desenvolvedores COM e o sítio Microsoft Developer Network (MSDN) [http://msdn.microsoft.com/].

A biblioteca MSDN contem informação para desenvolvedores da plataforma incluindo um revisão sobre os componente [overview of component development] usando tecnologias baseadas em COM.

Futuro do COM

O COM continua a ser suportado pela Microsoft?

Vai continuar a ser ?

COM é uma facilidade do Windows. a feature of Windows. Cada versão do Windows tem uma política denominada de *Windows Product Lifecycle.*

Assim, o COM continua a ser suportado como parte do Windows. A nova versão do Windows, o Windows Vista, é uma facilidade encontrada no sistema operacional [www.microsoft.com]

.NET

O .NET é a estratégia de Web Service da Microsoft para a conexão de informação, pessoas, sistemas e dispositivos através de uma abordagem de software.

O .NET é integrado através da plataforma Microsoft e prove a habilidade de fácil construção, utilização, gerência e soluções seguras considerando-se serviços web.

.NET

As soluções .NET permitem que aplicações comerciais sejam integradas de maneira rápida, provendo a informação onde é necessária.

Serviços Web

O que são os Serviços WEB?

Para um desenvolvedor os serviços web são:

" Módulos de software auto descritos, semanticamente encapsulados em uma função discreta, disponível e acessível através de protocolos Internet padrão, tais como o XML e SOAP"

Conteúdo Programático

. Serviços CORBA

. Objetos COM

. Barramento de objetos COM

. Serviços OLE/COM

OLE (Object Linkinh and Embedding)

Object Linking and Embedding (OLE) is a distributed object system and protocol developed by Microsoft.

OLE allows an editor to "farm out" part of a document to another editor and then reimport it.

For example, a desktop publishing system might send some text to a word processor or a picture to a bitmap editor using OLE. The main benefit of using OLE, next to reduced file size is the ability to create a master file.

OLE (Object Linkinh and Embedding)

References to data in this file can be made and the master file can then have changed data which will then take effect in the referenced document.

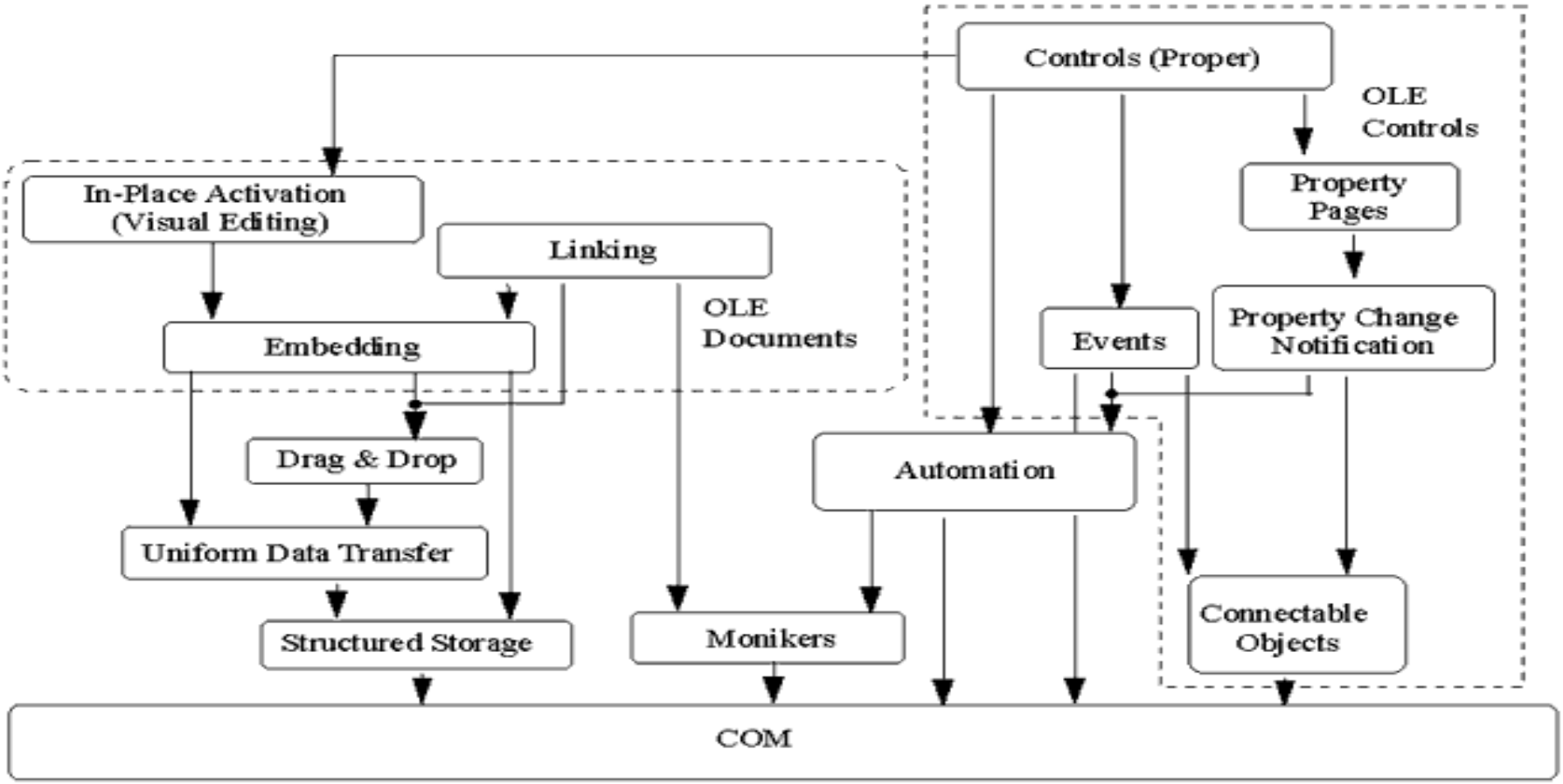
It was initially used primarily for copying and pasting data between different applications, especially using drag and drop, as well as for managing compound documents.

It later evolved to become an architecture for software components known as the component object model (COM), and later DCOM. OLE, Version 1.0 was released in 1990 [en.wikipedia.org].

OLE (Object Linkinh and Embedding)

A tecnologia OLE é uma abordagem de estruturação de documentos presente na maioria das aplicações Windows, que tornou-se com o passar do tempo foi generalizada e em uma abordagem orientada à objetos denominada de COM.

Tecnologia OLE



COM/DCOM
Custos e Limitações

• Baixo custo de ferramentas de desenvolvimento da Microsoft, tal como Visual C++ ou Visual Basic;

• Ferramentas de outros fabricantes, também, provêm facilidades para a construção e acesso a componentes COM para as plataformas Windows;

• Usualmente a construção de clientes e servidores é fácil;

COM/DCOM
Custos e Limitações

• O custo inicial para o COM e DCOM na plataforma Windows é relativamente baixo, quando comparado com outras plataformas;

• Para plataformas, como por exemplo dos *Mainframes,* o custo aproximado do DCOM em 1999 era da ordem de duzentos mil dolares;

COM/DCOM
Custos e Limitações

- Além de se considerar do uso da tecnologia, o desenvolvimento de um pacote de software utilizando COM e DCOM deve considerar a utilização de um programador experiente, devido a dificuldade de construção de aplicações distribuídas;

COM/DCOM
Custos e Limitações

• É um erro de projeto assumir que o uso do paradigma de objetos distribuídos com COM e DCOM reduz o requerimento de conhecimento de tópicos tais como:

- Projeto de sistemas distribuídos;

- Aplicações que se utilização de *multi-thread*;

- Conhecimento de redes.

COM/DCOM
Custos e Limitações

- A Microsoft possui um grande suporte para auxílio aos programadores que desenvolvem aplicações usando COM e DCOM. Existe uma grande quantidade de documentação referente a exemplos de desenvolvimento.

COM/DCOM
Outras Alternativas

• COM e DCOM representam uma alternativa em um universo onde existem outras tecnologias que dão suporte a computação distribuída. Exemplos de outros paradigmas são:

- RPC;
- Troca de mensagem;
- Memória compartilhada distribuída;
- Monitores de transações computacionais ;

COM/DCOM
Outras Alternativas

- Common Object Request Broker Architecture (CORBA) e Java RMI são abordagens que também podem ser consideradas como alternativas.

COM/DCOM
Outras Alternativas

Um comentário comum é a utilização comum de COM, DCOM, OLE, ActiveX, MTS e COM+.

Essas tecnologias constituem a estratégia distribuída e orientada da Microsoft, denominada de *Distributed interNet Architecture* (DNA), compreendendo um conjunto de produtos e especificações orientadas à rede.

Conteúdo Programático

. Serviços CORBA

. Objetos COM

. Barramento de objetos COM

. Serviços OLE/COM  **JAVA RMI**

JAVA RMI

A técnica conhecida como RMI (*Remote Method Invocation*) é uma solução existente para o *mundo* Java.

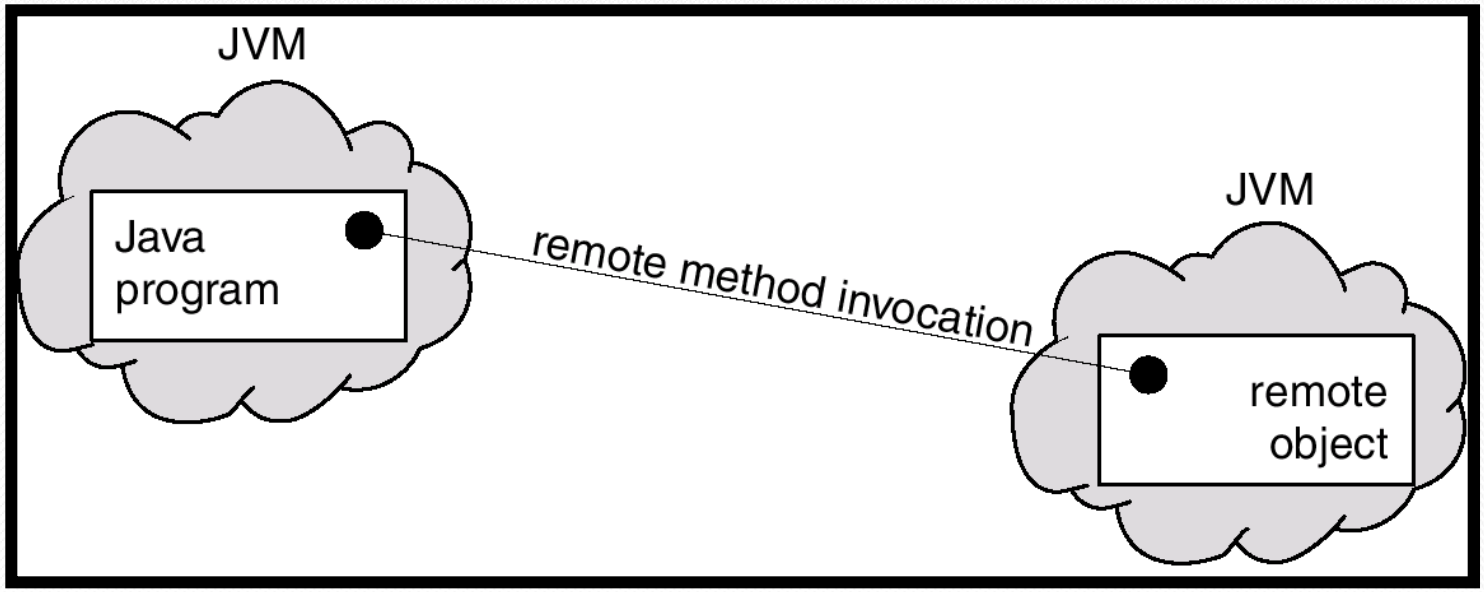
Nesta abordagem um programador pode criar uma rotina que pode invocar um outra rotina remota em máquinas com diferente arquitetura.

JAVA RMI

O RMI utiliza a serialização de objetos para *empacotar* e *desempacotar* parâmetros, não fazendo um truncamento, ou seja suportando um real poliformismo orientado a objeto [java.sun.com].

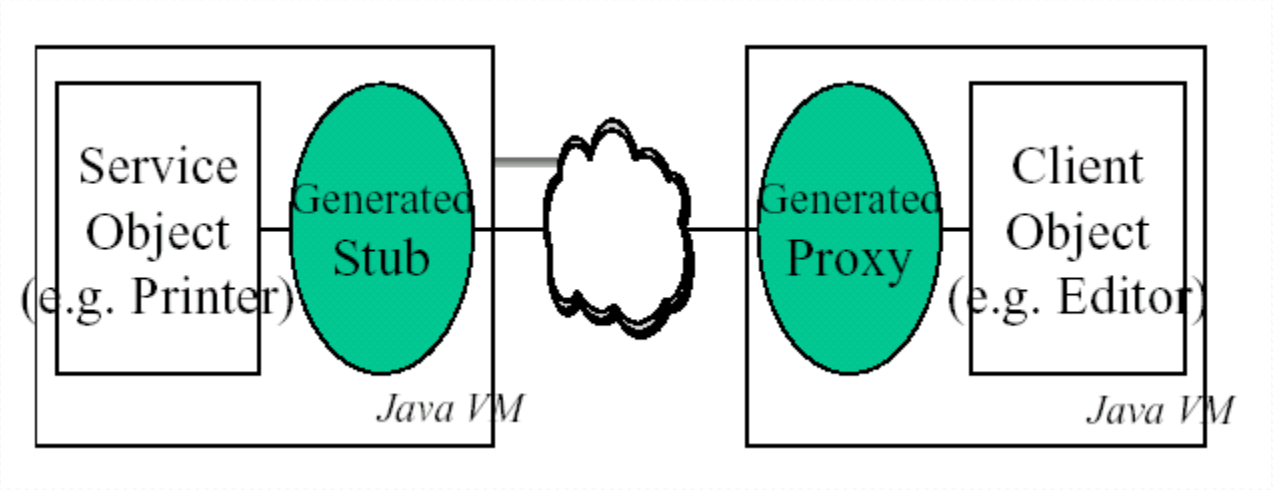
JAVA RMI

- Remote Method Invocation (RMI) é um mecanismo próprio de Java, similar a RPC.
- RMI permite um programa Java em uma máquina invocar um método remoto em um objeto remoto.



JAVA RMI

- Remote Method Invocation (RMI) é o mecanismo Java, que gera as classes proxy de forma automática.
- O usuário codifica os objetos cliente e serviço.
- O compilador RMI gera o código responsável pela comunicação na rede.



JAVA RMI

A principal funcionalidade do modelo JAVA RMI é permitir a invocação de um método de uma interface remota em um objeto remoto.

JAVA RMI

A especificação Java RMI define objeto remoto como um objeto no qual seus métodos podem ser invocados por outra Máquina Virtual Java (Virtual Machine), geralmente localizada em uma máquina diferente.

JAVA RMI

Um objeto deste tipo é descrito por uma ou mais interfaces remotas, que são interfaces Java que declaram os métodos de objeto remoto.

Importante observar que :

-Uma invocação de um método de um objeto remoto possui a mesma sintaxe de uma invocação de um método de um objeto local.

JAVA RMI

Esta característica, faz com que o RMI seja muito parecido com CORBA.

Os clientes RMI interagem com objetos remotos através de interfaces remotas bem definidas, nunca diretamente com as classes que implementam as interfaces.

O modelo RMI utiliza a serialização de objetos para converter objetos em streams de bytes para a transmissão.

JAVA RMI

Qualquer tipo de objeto pode ser passado durante a invocação, incluindo tipos primitivos, classes básicas, classes definidas pelo usuário, e JavaBeans.

O modelo Java RMI pode ser descrito como uma evolução natural do modelo RPC, adaptado para o paradigma da orientação a objetos.

JAVA RMI

O modelo RMI pode ser considerado como um ORB que suporta invocações em objetos remotos.

O RMI torna o ORB quase totalmente transparente para o cliente adicionando alguns requerimentos de implementação no servidor.

Na essência, o RMI é uma extensão da linguagem Java.

JAVA RMI

Os clientes RMI invocam os métodos remotos utilizando stubs específicos para cada serviço.

Esses stubs são gerados pelo compilador de stubs rmic e estendem as funções da classe RemoteStub.

No lado do servidor, as classes RMI precisam estender a classe UnicastRemoteObject.

JAVA RMI

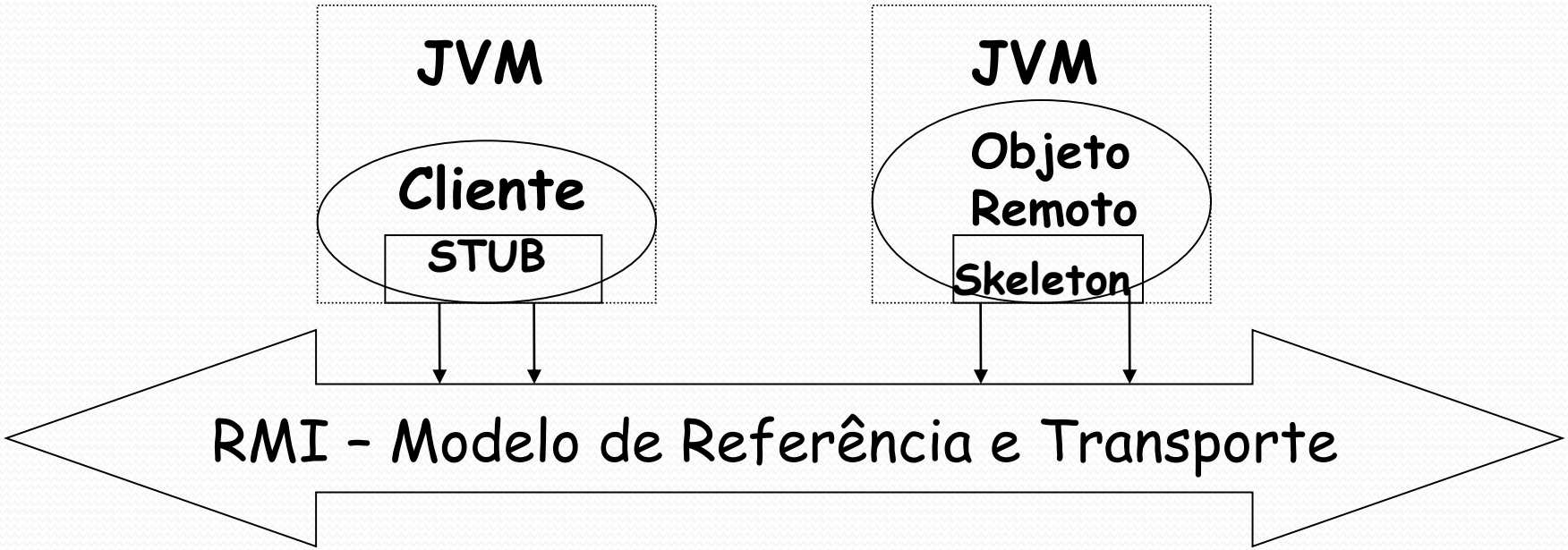
O modelo RMI envia e recebe as chamadas dos clientes para os objetos remotos do servidor utilizando skeletons gerados pelo compilador rmic.

O compilador rmic implementa uma classe skeleton para cada interface definida no servidor.

JAVA RMI

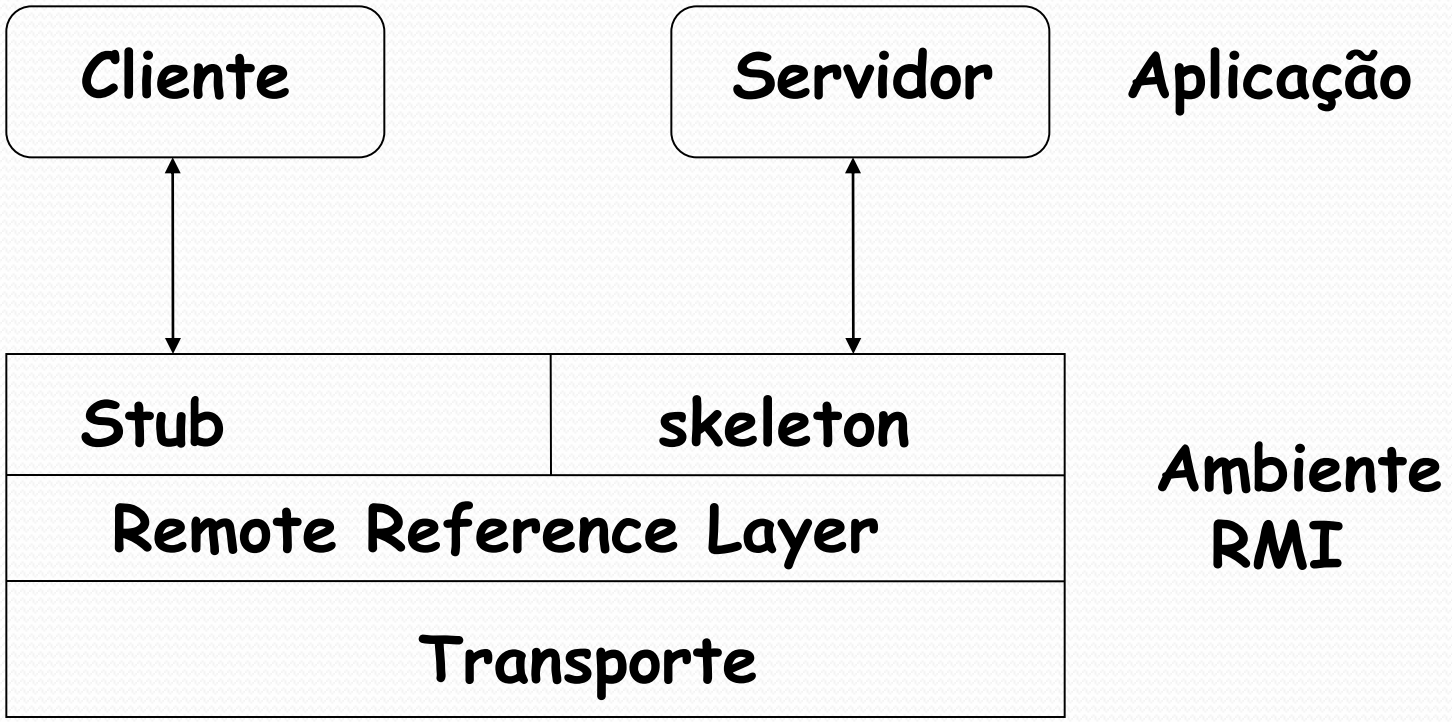
A próxima figura ilustra o mecanismo:

JAVA RMI

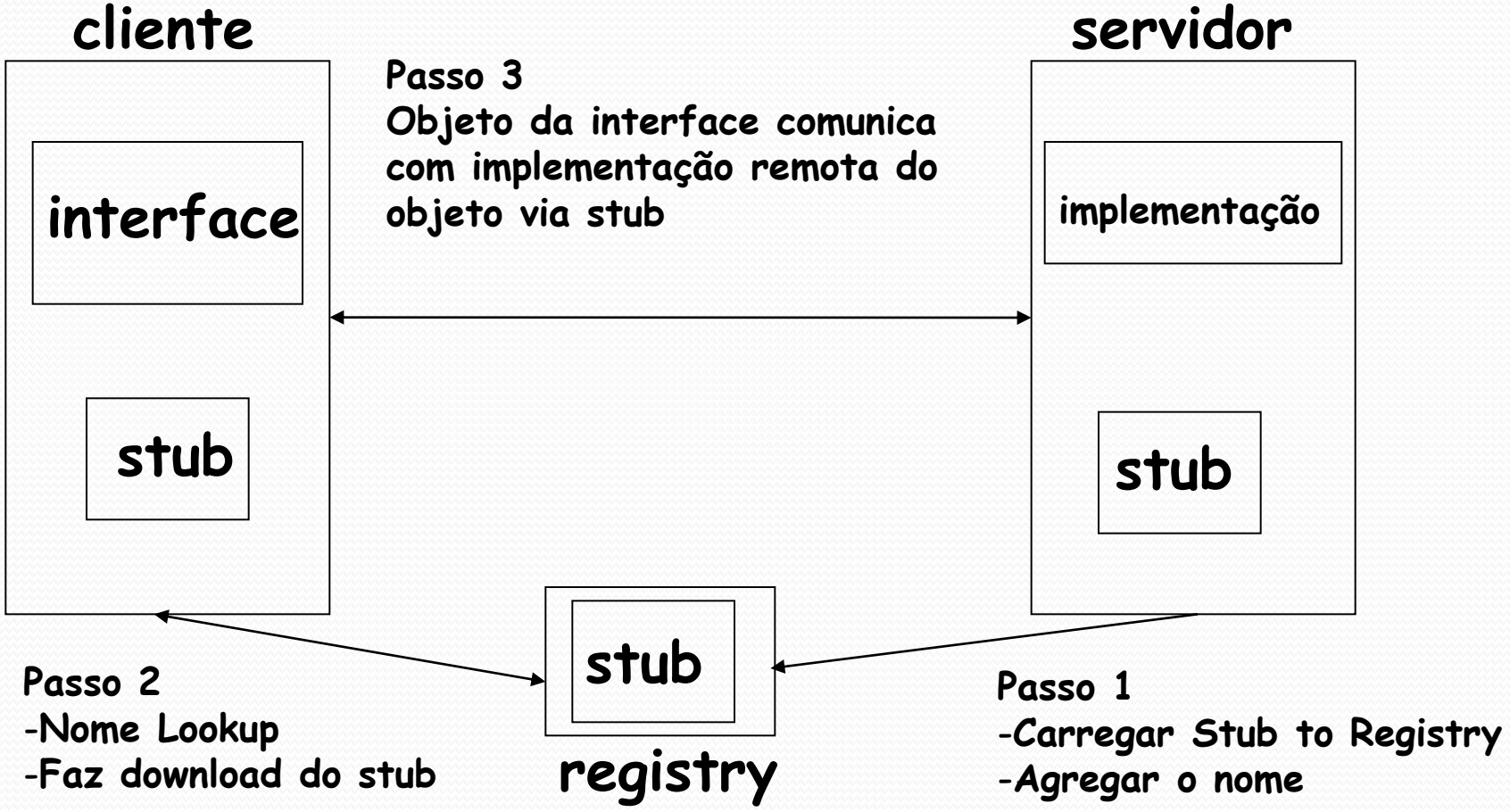

JVM
Cliente
STUB
JVM
Objeto
Remoto
Skeleton
RMI – Modelo de Referência e Transporte

JAVA RMI

Três níveis da Arquitetura JAVA RMI



JAVA RMI
Exemplo de Uso



JAVA RMI

Diferentemente de uma chamada local em Java, uma invocação RMI utiliza a passagem de parâmetros por cópia (*pass-by-value*), ao invés da passagem por referência (*pass-by-reference*), exceto com relação a objetos remotos que são passados por referência ao invés de se copiar a implementação remota.

JAVA RMI

O modelo RMI fornece classes e interfaces que possibilitam a localização, o carregamento, e a execução confiável de objetos remotos.

Atualmente, o modelo RMI, provê um serviço bastante primitivo e não persistente, de nomeação. Ele também fornece um carregador de classes (*class loader*) que carrega stubs do servidor automaticamente.

JAVA RMI

Um *stub* RMI funciona como um *proxy* no cliente para o Objeto remoto. Além disso, o modelo também provê mecanismos extras de segurança para garantir que estes *proxies* tenham o comportamento esperado.

JAVA RMI

Em um sistema distribuído, um coletor de lixo deve ser apto a automaticamente remover objetos remotos que não são mais referenciados por nenhum cliente.

O modelo RMI utiliza um mecanismo de coleta de lixo por contagem de referências que mantém um histórico de todas as referências externas ativas dentro de uma máquina virtual.

JAVA RMI

Neste contexto, uma referência externa ativa é apenas uma conexão TCP/IP entre um cliente e um servidor.

Cada vez que o cliente obtém uma referência, o contador de referências daquele objeto é incrementado de 1; e é decrementado de 1 quando o cliente pára de referenciar aquele objeto, ou seja, quando a conexão é finalizada.

JAVA RMI

Quando o contador de referência alcança zero, o RMI coloca o objeto servidor na lista de referências fracas (*weak reference list*). O coletor de lixo pode então remover o objeto.

JAVA RMI

No modelo RMI, uma referência para um objeto remoto é "alugada" (*leased*) por um período de tempo para o cliente.

O período de aluguel se inicia quando a chamada remota é recebida. É de responsabilidade do cliente renovar o período de aluguel através de chamadas adicionais no servidor antes que o prazo se expire.

JAVA RMI

Pelo fato do modelo RMI poder resolver dinamicamente invocações de métodos entre Máquinas Virtuais, ele fornece um ambiente totalmente distribuído e orientado a objetos.

JAVA RMI

Também pelo fato do modelo RMI ser nativo da linguagem Java, os *desenvolvedores* trabalham dentro de um único modelo de objetos (o modelo Java), ao invés de trabalhar com múltiplos modelos de objetos (Java, CORBA IDL e outros).

JAVA RMI

Isto remove bastante a complexidade de programação.

Diferente de modelos de objetos com linguagens neutras, o RMI não requer mapeamento para linguagens de definição de interfaces, como a IDL.

JAVA RMI

Antes da invocação de um método em um objeto remoto, o cliente deve primeiro obter uma referência para este objeto.

Geralmente, o cliente obtém essa referência como retorno da invocação de um método.

JAVA RMI

O modelo RMI também fornece um serviço simples de nomeação, conhecido como RMI registry, ou simplesmente registro RMI, que permite a obtenção das referências dos objetos através de nomes.

O RMI define o serviço de nomeação utilizando duas classes e uma interface.

JAVA RMI

Tipicamente, o cliente interage com a classe Naming para obter os serviços do registro do RMI.

O registro RMI é definido pela interface Registry. É importante notar que este registro é um objeto servidor remoto. Isto significa que se pode acessar seus métodos através da rede utilizando invocações remotas.

JAVA RMI

A classe LocateRegistry auxilia na localização do registro RMI. A JavaSoft fornece uma implementação da interface Registry.

Esta implementação não-persistente apenas suporta nomes simples. Nesta implementação, os clientes utilizam apenas os métodos *lookup* e *invoke*.

Toda essa comunicação entre objetos é realizada através do protocolo Java Remote Method Protocol (JRMP).

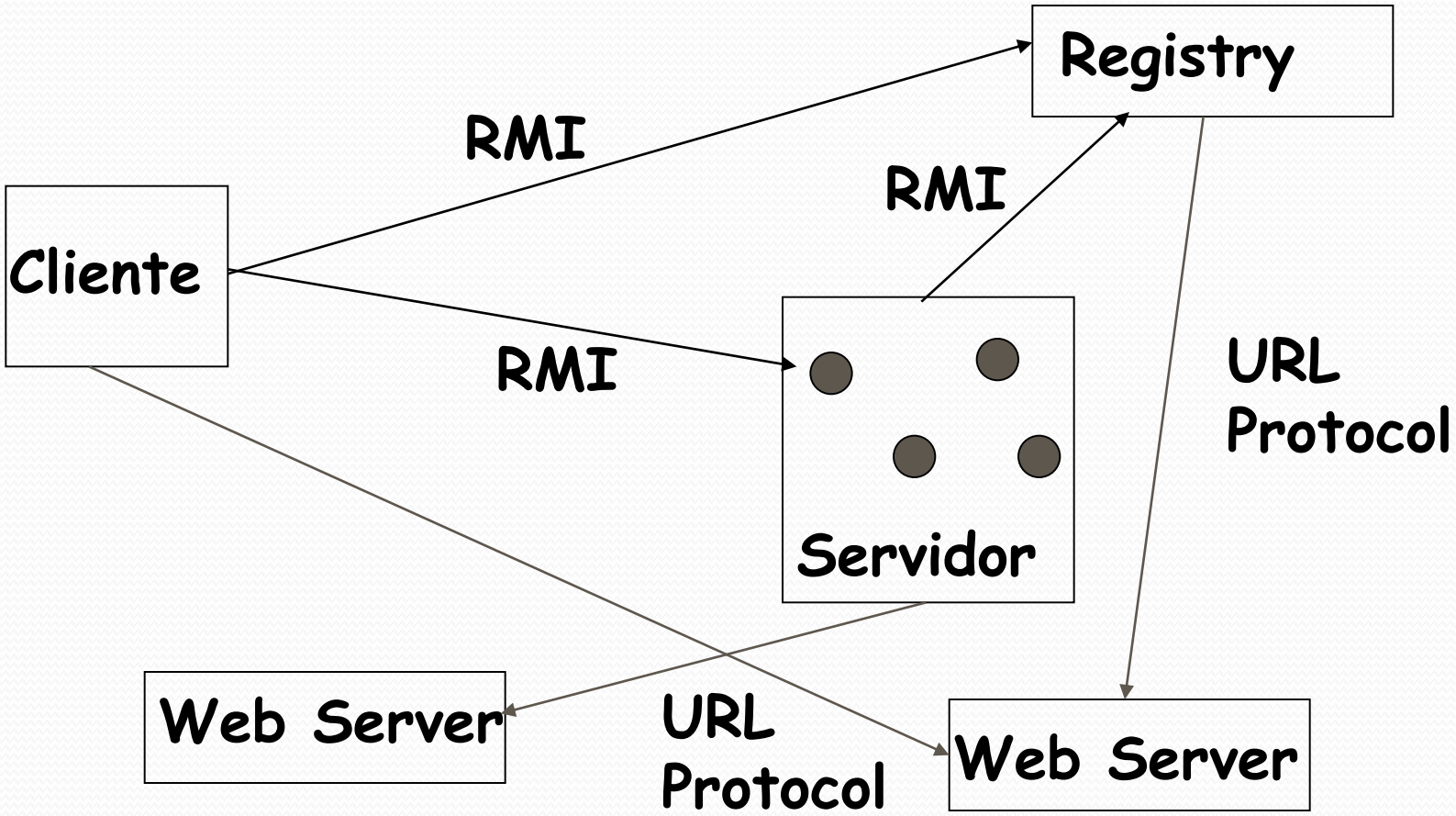
JAVA RMI

A próxima figura ilustra uma chamada típica de um método remoto utilizando a arquitetura RMI.

A figura mostra o servidor registrando seus objetos no registro RMI e o cliente obtendo a referência do objeto requisitado através do mesmo.

JAVA RMI

Exemplo de chamada RMI



JAVA RMI

Com a referência retornada, o cliente pode então realizar a invocação do método remoto.

Os dois servidores *Web* da figura exemplo funcionam como repositórios de classes e *stubs*.

A idéia principal é que os clientes, e mesmo os servidores, possam carregar dinamicamente classes e *stubs* necessários para a aplicação e que não se encontram localmente.

JAVA RMI

Esse procedimento é conhecido como *dynamic stub-loading*.

O registro RMI armazena, juntamente com a referência de cada objeto, a URL que permite a localização das classes e *stubs*.

É importante lembrar que sem os *stubs* apropriados, um cliente não pode fazer uma invocação remota em um objeto.

JAVA RMI

A localização das classes e *stubs* é definida pela propriedade *java.rmi.server.codebase* informada para a Máquina Virtual na inicialização do programa.

Além de servidores *Web*, servidores FTP também podem ser utilizados como repositórios das classes e dos *stubs*.

JAVA RMI

Atualmente, há um esforço grande em especificar um modelo padrão entre RMI e CORBA.

Duas especificações estão em desenvolvimento, mas já podem ser encontradas para utilização:

JAVA RMI

a) RMI-over-IIOP: é uma versão do modelo RMI que funciona no topo do protocolo IIOP.

O protocolo IIOP provê alguns benefícios, entre eles: interoperabilidade com outros objetos escritos em outras linguagens, e um padrão aberto para objetos distribuídos;

JAVA RMI

b) RMI/IDL: Através desse padrão é possível especificar interfaces CORBA utilizando a semântica do RMI ao invés da IDL de CORBA.

O compilador gera automaticamente CORBA IDL, stubs e skeletons. *Enterprise JavaBeans* utiliza este subconjunto RMI/IDL.

JAVA RMI

A especificação EJB define um modelo de componentes-servidores para JavaBeans.

Um EJB é um JavaBean especializado e não-visual que é executado no servidor.

JAVA RMI

Resumidamente, pode-se concluir que o modelo RMI oferece alguns dos elementos críticos de um sistema de objetos distribuídos em Java pelo fato do RMI ser nativo da linguagem Java.

JAVA RMI

Possui facilidades de comunicação de objetos análogas ao protocolo IIOP de CORBA, e seu sistema de serialização de objetos fornece uma maneira prática de se transferir ou requisitar uma instância de um processo remoto para outro.

JAVA RMI

Também possui uma forte influência do modelo CORBA e pode ser definido como um novo tipo de ORB construído no topo do modelo de objetos de Java.

E como um ORB, RMI introduz cinco inovações-chave:

JAVA RMI

1. Permite mover código ao invés de somente dados;

1. Garante a integridade do código carregado remotamente;

1. Permite a passagem de objetos por valor (objects-by-value);

4. Utiliza Java como uma linguagem para definição das interfaces e para a implementação dos Métodos;

5. Utiliza um mecanismo baseado em URLs para a nomeação dos objetos.

JAVA RMI

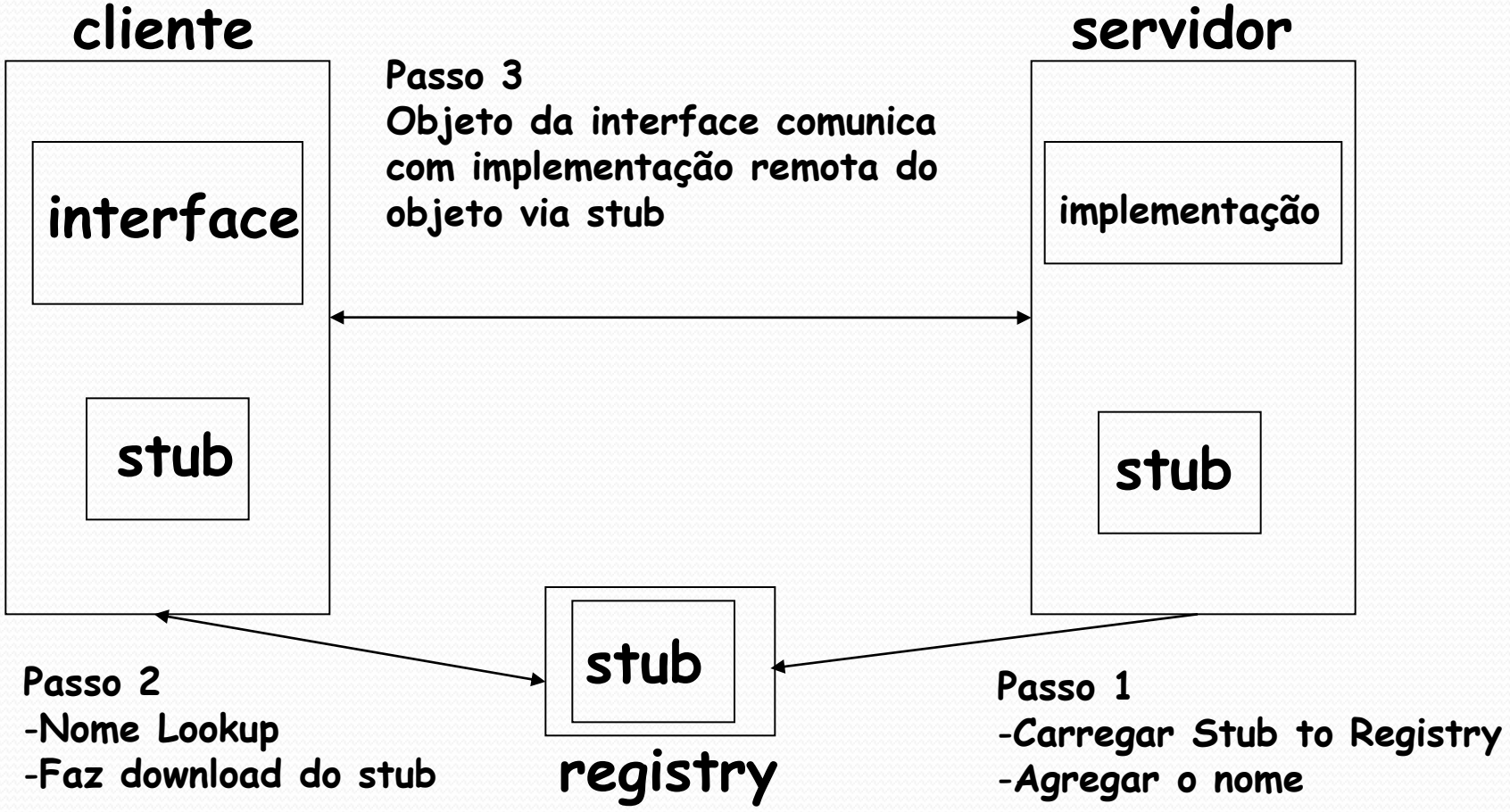
Em adição as essas inovações-chave, o modelo RMI implementado pela plataforma Java 2 introduz características similares ao modelo CORBA, que tornam a implementação com o uso de *RMI-over-IIOP* mais fácil e flexível.

Entre elas: suporte para persistência em referências de objetos, suporte para ativação remota de objetos, e suporte para *sockets* customizados.

JAVA RMI

O nome "Java 2" é uma definição para as implementações JDK 1.2 e superiores, incluindo J2SE, J2EE e J2ME.

JAVA RMI
Exemplo de Uso



JAVA RMI

Iniciando um RMI Registry

Iniciando o RMI registry permite que seja criado e exportado objetos remotos.

> rmiregistry

JAVA RMI

Definindo a interface remota.

```
import java.rmi.*;
public interface RObject extends Remote {
        void  aMethod() throws RemoteException;
}
```

JAVA RMI

Definindo a implementação do objeto remoto.

```
import java.rmi.*;
  import java.rmi.server.UnicastRemoteObject;
   public class RObjectImpl extends UnicastRemoteObject implements RObject {
        public RObjectImpl() throws RemoteException {
             super();
         }
        // All remote methods must throw RemoteException
        public void aMethod() throws RemoteException {
        }
   }
```

JAVA RMI

Compilando a implementação do objeto remoto

› javac *RObject*.java *RObjectImpl*.java

JAVA RMI

Gerando os módulos *skeletons* e *stubs*.

> rmic *RObjectImpl*

JAVA RMI

Criando uma instância do objeto remoto e agregando no RMI registry.

```
try {
    RObject robj = new RObjectImpl();
    Naming.rebind("//localhost/RObjectServer", robj);
} catch (MalformedURLException e) {
} catch (UnknownHostException e) {
} catch (RemoteException e) {
}
```

JAVA RMI

Looking Up, Objeto Remoto e Invocação de Método

```
try {
    // Look up remote object
     RObject robj = (RObject) Naming.lookup("//localhost/RObjectServer");
     // Invoke method on remote object
   robj.aMethod();
} catch (MalformedURLException e) {
} catch (UnknownHostException e) {
} catch (NotBoundException e) {
} catch (RemoteException e) {
}
```

JAVA RMI

O modelo RMI pode ser bem vantajoso para programadores Java. Ele torna o ORB transparente e naturalmente estende o alcance das funcionalidades da linguagem Java. Com a utilização de RMI, não é necessário utilizar a IDL de CORBA ou qualquer tipo de conversão Java/CORBA.

Entretanto, utilizando apenas RMI, perde-se o poder da infraestrutura do modelo CORBA.

JAVA RMI

Para compensar esta deficiência, pode-se utilizar RMI-over-IIOP, garantindo a união entre os dois modelos: RMI e CORBA.

Comparação dos Middlewares

Vamos neste momento traçar alguns pontos para que seja possível uma melhor escolha entre os middlewares Corba, Dcom e Java RMI.

Comparação dos Middlewares

Algumas métricas que devemos considerar são :

- Independência de linguagem;
- Linguagem;
- Independência de plataforma;
- Facilidade de aprendizado;
- Facilidade de execução de tarefas complexas;
- Tempo para desenvolvimento;
- Ambiente Microsoft.

Comparação dos Middlewares

Independência de linguagem

Essa métrica considera se o *middleware* é baseado (ou não) em uma determinada linguagem.

- Corba: é apenas uma especificação, assim a abordagem pode ser implementada em qualquer linguagem;

-Dcom: suporta uma série de linguagens desde que tenham uma compatibilidade binária com ambiente Microsoft;

- Java RMI: Somente usado com a linguagem Java

Comparação dos Middlewares

Linguagem

O aspecto *linguagem* refere-se a que ambiente os middlewares são mais facilmente trabalhados:

- Corba e Dcom: C++ e VB;

- Java RMI: Java.

Comparação dos Middlewares

Independência de plataforma

A métrica independência de plataforma é relativa a possibilidade de sua implementação em diferentes ambientes e o quão fácil pode ser atingido esse objetivo.

-Corba e Java RMI: podem ser implementados em várias configurações e de fácil implementação;

- DCOM: apenas na plataforma Microsoft e em poucas não Microsoft.

Comparação dos Middlewares

Facilidade de aprendizado

Com referência a facilidade de aprendizado podemos afirmar que:

- Corba: por ser uma especificação complexa é de difícil aprendizado;

- Dcom: Embora seja difícil o número de programadores e conceitos do ambiente Microsoft facilitam seu aprendizado;

- Java RMI: Semelhante ao Dcom o grande número de adeptos do ambiente Java proporcionam uma maior facilidade para o entendimento do Java RMI.

Comparação dos Middlewares

Facilidade de execução de tarefas complexas

Quanto a facilidade de execução de tarefas complexas, considerando a passagem de métodos e objetos, pode-se afirmar que:

- CORBA e DCOM: por causa da heterogeneidade de linguagens possíveis de serem usadas não existe um maior uso de funções especificas;

- JAVA RMI: por causa do uso específico de uma linguagem é possível a utilização de complexas funções.

Comparação dos Middlewares

Tempo para desenvolvimento

Quanto ao tempo para desenvolvimento é importante levar em consideração o tempo em si e o quanto será compatível com aplicações já existentes, assim:

• CORBA: é *tido* que o tempo é curto e que ferramentas têm elevado número, devido a grande interoperabilidade

•Dcom e Java RMI: *bom* tempo, todavia ambos ambientes ficam em um segundo plano quando comparados com Corba.

Comparação dos Middlewares

Ambiente Microsoft

O aspecto ambiente Microsoft leva em conta se o middleware é facilmente adaptável a esse ambiente:

- Corba e Java RMI: possuem facilidades de adaptação, mas não tão natural como DCOM;

- DCOM: naturalmente projetado para o ambiente Windows.

Comparação dos Middlewares [Cunnane 2004]

Métricas	Corba	DCOM	JAVA-RMI
Linguagem C++ (6)	6	0	0
Indep. de Plataforma (5)	10	5	5
Temp. Desenv. (4)	8	4	4
Tipo de aplicação: Internet (3)	0	0	3
Tarefas complexas (2)	2	2	4
Linguagem Java (1)	0	0	1
	26	11	22