Arquiteturas dos sistemas

Prof. Dr. José de Jesús Pérez Alcázar

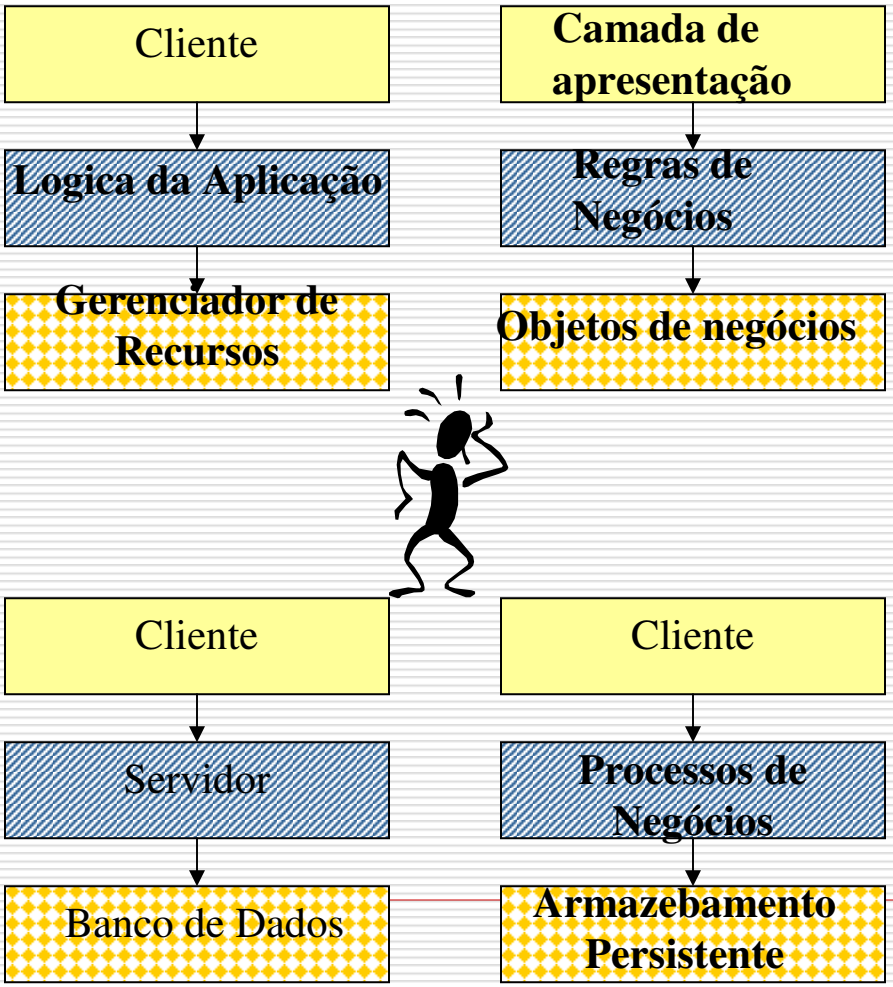
Evolução Tecnológica

- Importância da computação distribuída, especificamente a arquitetura C/S que tem-se sobressaído como alternativa de implementação de solução de sistema para suprir as necessidades de negócios atuais.
- Este crescimento está associado a:
  - Constante aumento do poder de processamento dos micros, aliado ao barateamento do seu preço. Desenvolvimento da computação gráfica, da web, etc.
  - Computador pessoal → conflito da relação entre indivíduos e organização. Deve ser conciliado.
- Camadas.

Evolução Tecnológica

- Arquiteturas mais utilizadas na implementação da solução de sistemas são até hoje:
  - Multiusuária centralizada;
  - Distribuída de usuário único;
  - Arquitetura C/S

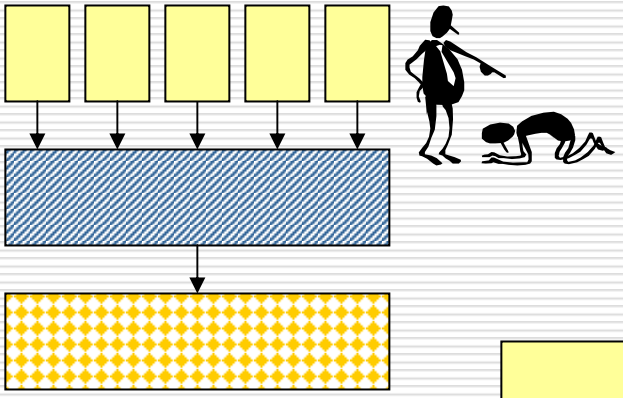
Conceitos básicos e notação



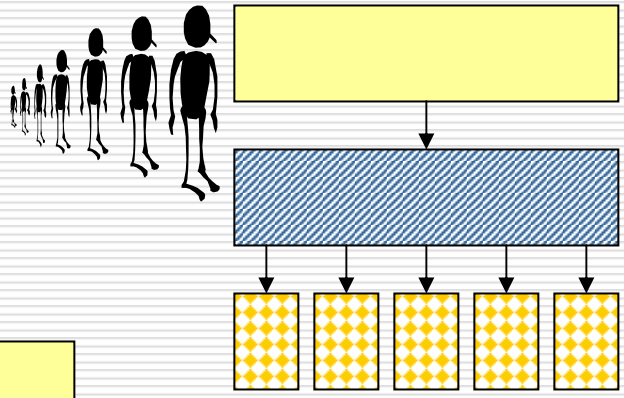
- **Cliente é qualquer usuário ou programa que deseja desenvolver uma operação no sistema, através de uma camada de apresentação.**
- **A lógica da aplicação estabelece que operações podem ser desenvolvidas no sistema e como elas acontecem. Ela trata de reforçar as regras de negócios e estabelece os processos de negócios. A lógica de aplicação pode ser expressada e implementada de várias formas: restrições, processos de negócios, servidor com lógica codificada ...**

O gerenciador de recursos trata com a organização (armazenamento, indexação e recuperação) dos dados necessários para suportar a lógica da aplicação. Isto é tipicamente um banco de dados mas ele pode ser também um sistema de recuperação de textos ou qualquer outro sistema de gerenciamento de dados que fornece facilidades de consulta e persistência

Distribuição nas diferentes camadas



Suporte para múltiplos clientes



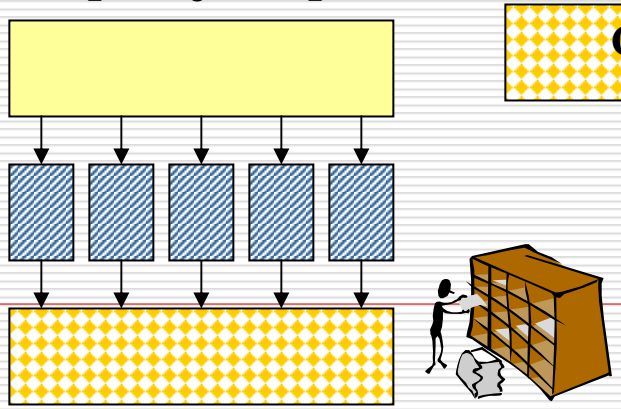
Distribuição ou replicação de dados

Lógica da apresentação

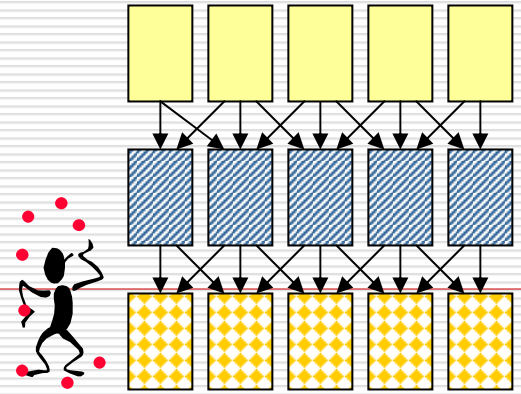
Lógica da Aplicação

Gerenciador de Recursos

Lógica de aplicação separada



Qualquer combinação



Estrutura das aplicações

- Lógica da apresentação: realiza a interação com o usuário. Funções:
  - Garantir a segurança de acesso ao sistema;
  - Permitir a navegação sobre o sistema para a chegada do usuário ao ponto desejado;
  - Realizar a apresentação das informações;
  - Manipular as informações imputadas no sistema;
  - Realizar algumas análises destas informações.
- Sua responsabilidade é fornecer uma interface compreensível e eficiente ao usuário. Formato claro, realização das tarefas com rapidez.

Estrutura das aplicações

As principais atividades realizadas por esta camada são:

- manipulação de menus de acesso;
- formatação e apresentação de telas;
- gerenciamento de diálogos;
- apresentação de gráficos ou caracteres de dados;
- interpretação de mensagens do meio externo, como os cliques do *mouse* ou caracteres enviados pelo teclado;
- controle da segurança de acesso ao sistema;
- validação de tipos de dados;
- edição de campos cruzados;
- edição de ajuda ao usuário;
- muitas vezes, validação de lotes de dados.

- Comunicação com interface GUI. Escrita em: Linguagens visuais, Java Javascript, etc..

Estrutura das aplicações

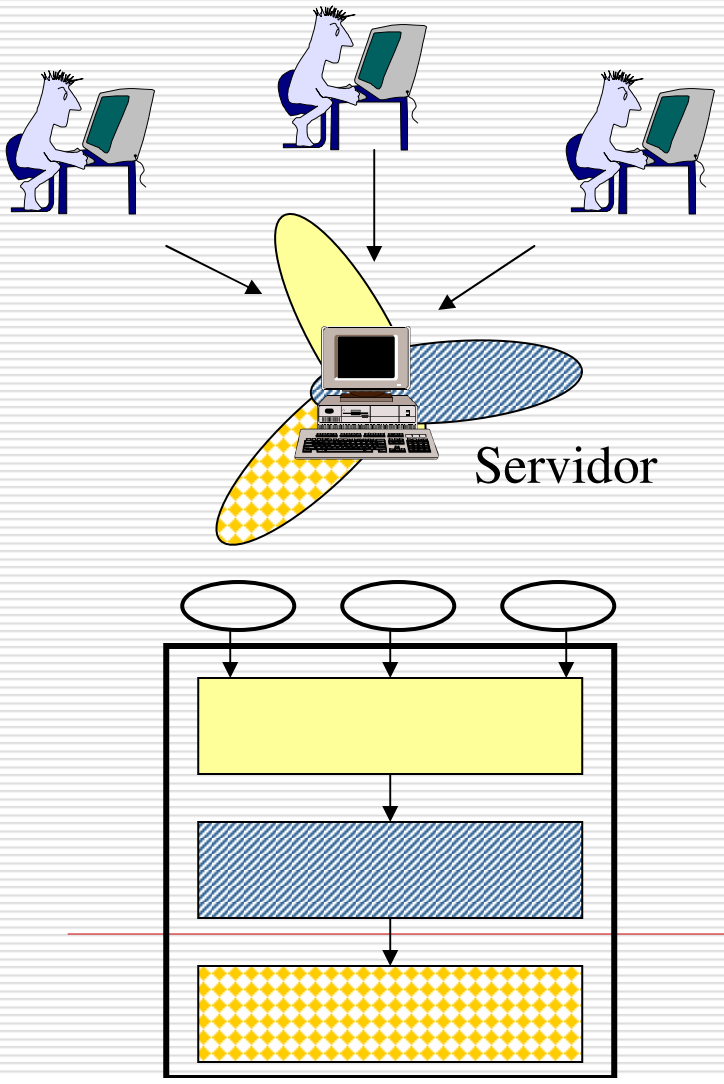
- Função da lógica de regras de negócio ou aplicação: responsável pela imposição da política de negócio da organização, comandando a tomada de decisão do processo, conforme os requisitos e regras que determinam o comportamento do negócio.
- Escrita em linguagens de 3a. Geração (C, Java, etc.) ou de 4a geração.

Estrutura das aplicações

- A lógica de gerenciamento de dados ou recursos: garante o armazenamento, a manipulação, a consistência, a integridade e a segurança dos dados. Um SGBD.
- Funções:
  - Fazer a leitura e a atualização dos dados
  - Controlar a segurança do acesso aos dados
  - Garantir a integridade dos dados
  - Realizar a validação das operações solicitadas.
- A interface de comunicação com a camada de gerenciamento de dados é representada pelas transações e consultas.

Arquitetura multiusuária centralizada

Arquitetura de uma camada

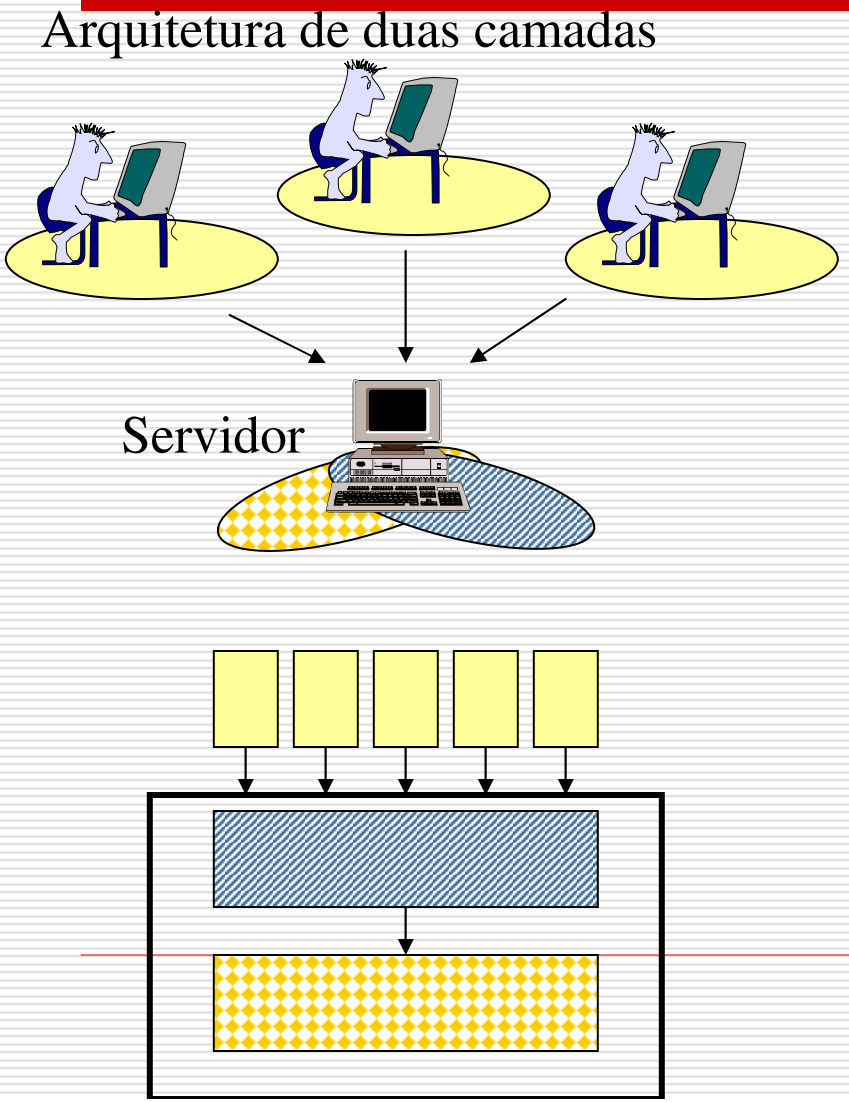


•A camada de apresentação, a lógica da aplicação e o gerenciador de recursos são construídos como uma entidade monolítica.
•Usuários/programas acessam o sistema através de terminais "burros", o que é apresentado e como ele aparece é controlado pelo servidor.
•Esta foi a arquitetura típica das aplicações de mainframe, oferecendo várias vantagens:

- **nenhuma comutação de contexto no fluxo de controle (tudo acontece dentro do sistema),**
- **tudo é centralizado;**

gerenciamento e controle de recursos é fácil, o projeto pode ser muito otimizado atenuando a separação entre camadas.

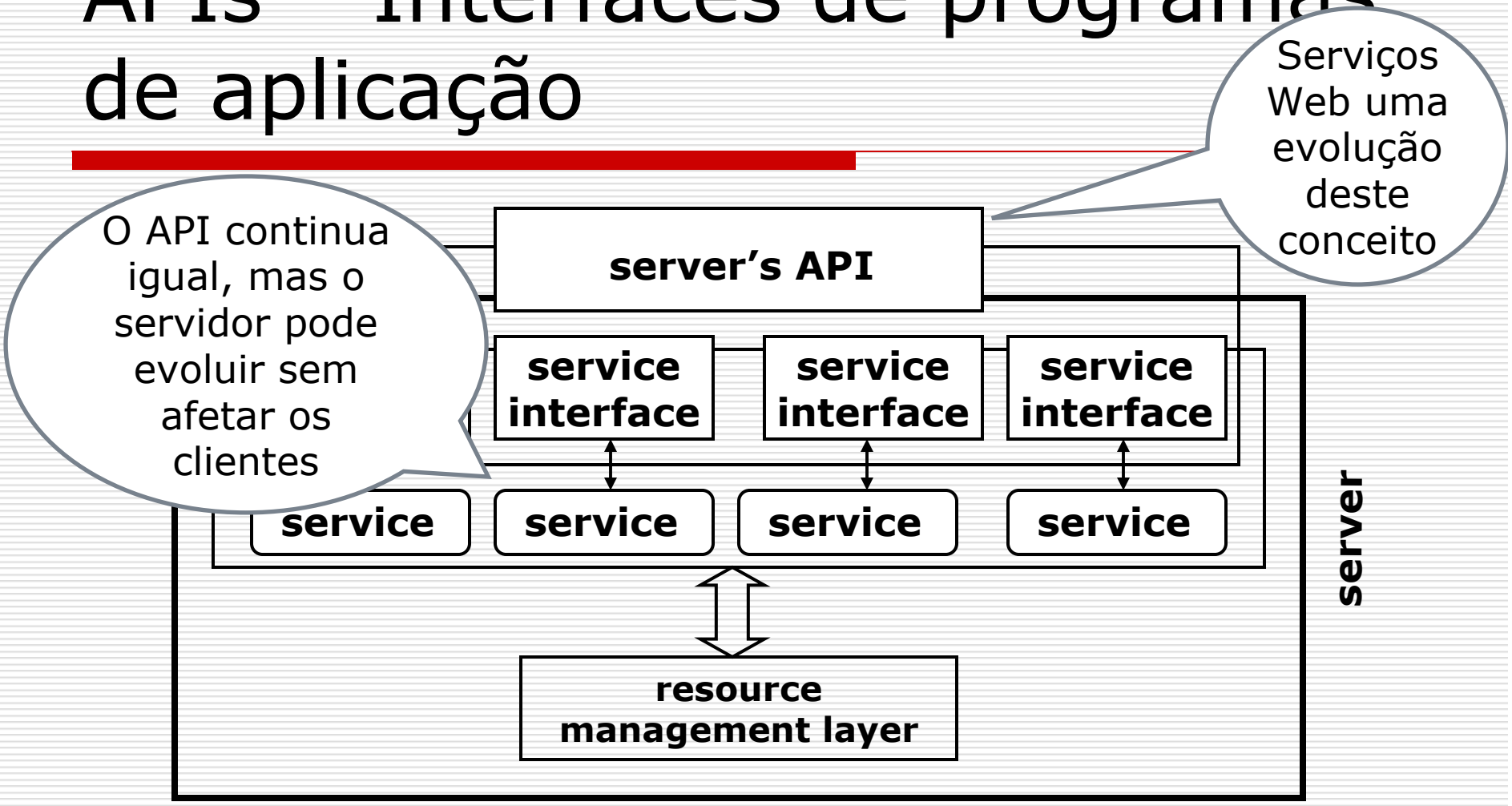
Arquitetura de duas camadas → C/S



• Na medida que os computadores ficam mais poderosos, a camada de apresentação movimenta-se ao cliente. Isto tem várias vantagens:

- **Clientes são independentes entre eles: um poderia ter várias camadas de apresentação dependendo do que cada cliente deseja fazer.**
- **Pode ser tirada vantagem da potência computacional na máquina cliente para ter camadas de apresentação mais sofisticadas. O que poupa recursos computacionais na máquina servidora.**
- **Ela introduz o conceito de API (Application Program Interface). Uma interface para invocar o sistema desde fora.**
- **O gerenciador de recursos somente enxerga um cliente: a lógica da aplicação. Isto ajuda bastante ao desempenho já que não existem connexões/sessões para manter.**

APIs – Interfaces de programas de aplicação



Organização interna da camada de lógica da aplicação em um sistema de duas camadas

Características da Arquitetura C/S

- **Uma arquitetura C/S consiste em um processo cliente e um processo de servidor, que podem ser distinguidos um do outro, embora possam interagir totalmente.**
- **A parte cliente e as partes servidor podem operar em diferentes plataformas, mas isto não é uma regra.**
- **Tanto a plataforma do cliente como a do servidor podem ser atualizadas, sem que se tenha de necessariamente atualizar a outra.**
- **O servidor pode atender a vários clientes simultaneamente; em alguns sistemas C / S, os clientes podem acessar vários servidores.**
- **Os sistemas C/S incluem algum tipo de capacidade de operar em rede.**
- Uma parte da lógica da aplicação pode residir no cliente.
- A ação, em geral, é iniciada no cliente, e não no servidor. No entanto, servidores de BDs podem "iniciar a ação", baseados em gatilhos, bem como através de regras do negócio ou procedimentos armazenados.
- Uma interface gráfica amigável geralmente reside no cliente.
- Um recurso SQL é característico.

Aplicação Cliente/Servidor

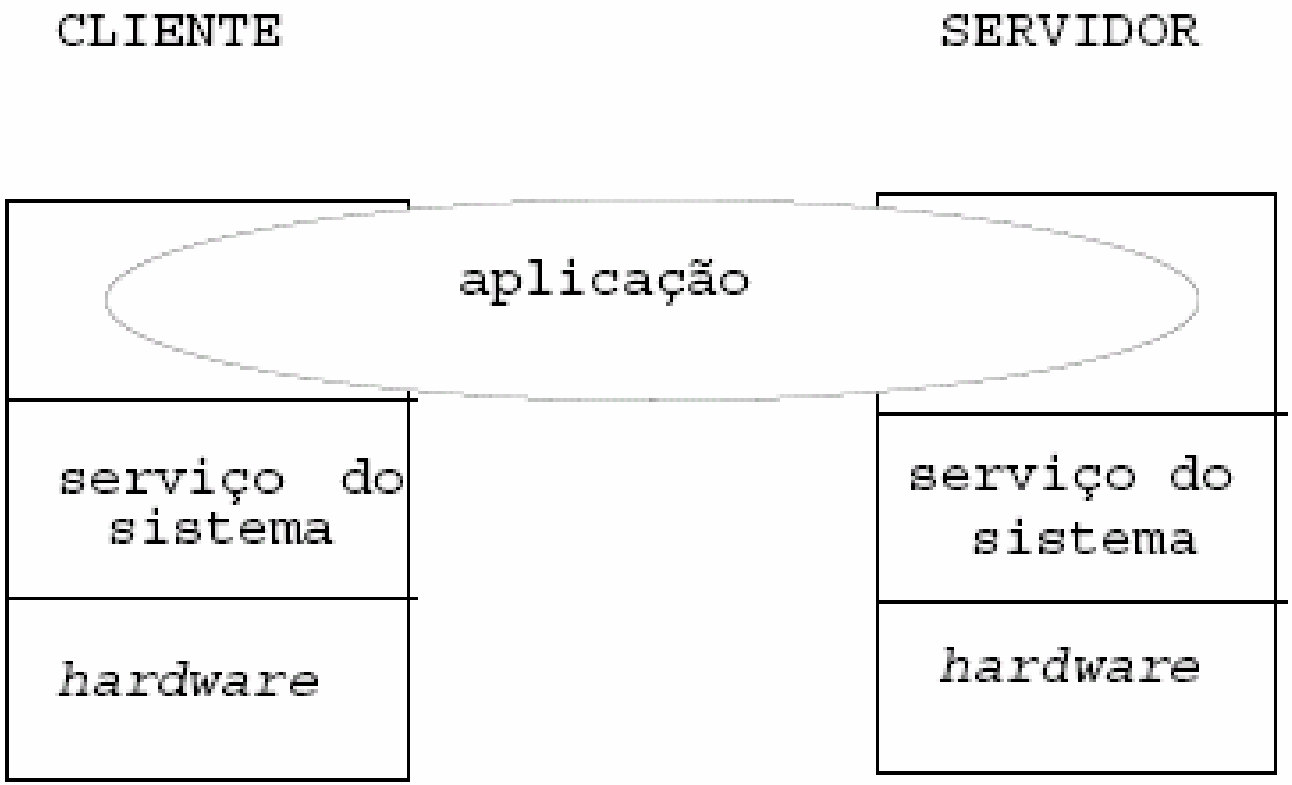


Figura 02.1.2 - Aplicação cliente/servidor

Atributos de cliente

- Responsável pela interação com o usuário. Funções principais:
  - Interpretação de menus ou comandos;
  - Formatação de tela;
  - Entrada de dados;
  - Apresentação de dados.
- Aplicações gráficas:
  - Manipulação de janelas
  - Gerenciamento de som e vídeo
  - Controle do quadro de diálogo
- Dependendo do particionamento da aplicação:
  - Validação de dados de entrada
  - Processamento de ajuda
  - Seleção de dados
  - Classificação de dados
- Não se contempla comunicação cliente-cliente

Atributos do servidor

- Ele roda ininterruptamente, oferecendo serviços a seus clientes. Funções:
  - Compartilhamento de dados na aplicação;
  - Compartilhamento de comunicação;
  - Compartilhamento de arquivos;
  - Compartilhamento de impressora;
  - Compartilhamento de CPU;
  - Compartilhamento de vídeo.
- Ele recebe pedidos de um ou mais clientes, tem modo de processamento reativo e funções específicas. Ele oculta detalhes de implementação dos serviços e não contempla a comunicação servidor-servidor.

Atributos de comunicação

- Os sistemas operacionais de multitarefa ou multiprocessamento, na sua maioria, oferecem facilidades de comunicação entre processos (IPC). Processos concorrentes.
- Os processos se comunicam compartilhando memória ou passando mensagens. As técnicas de compartilhamento de memória → mesma máquina.
- Processos C-S estão baseados em técnicas de passagens de mensagens → chamada remota de procedimentos (RPC)

Comunicação via RPC

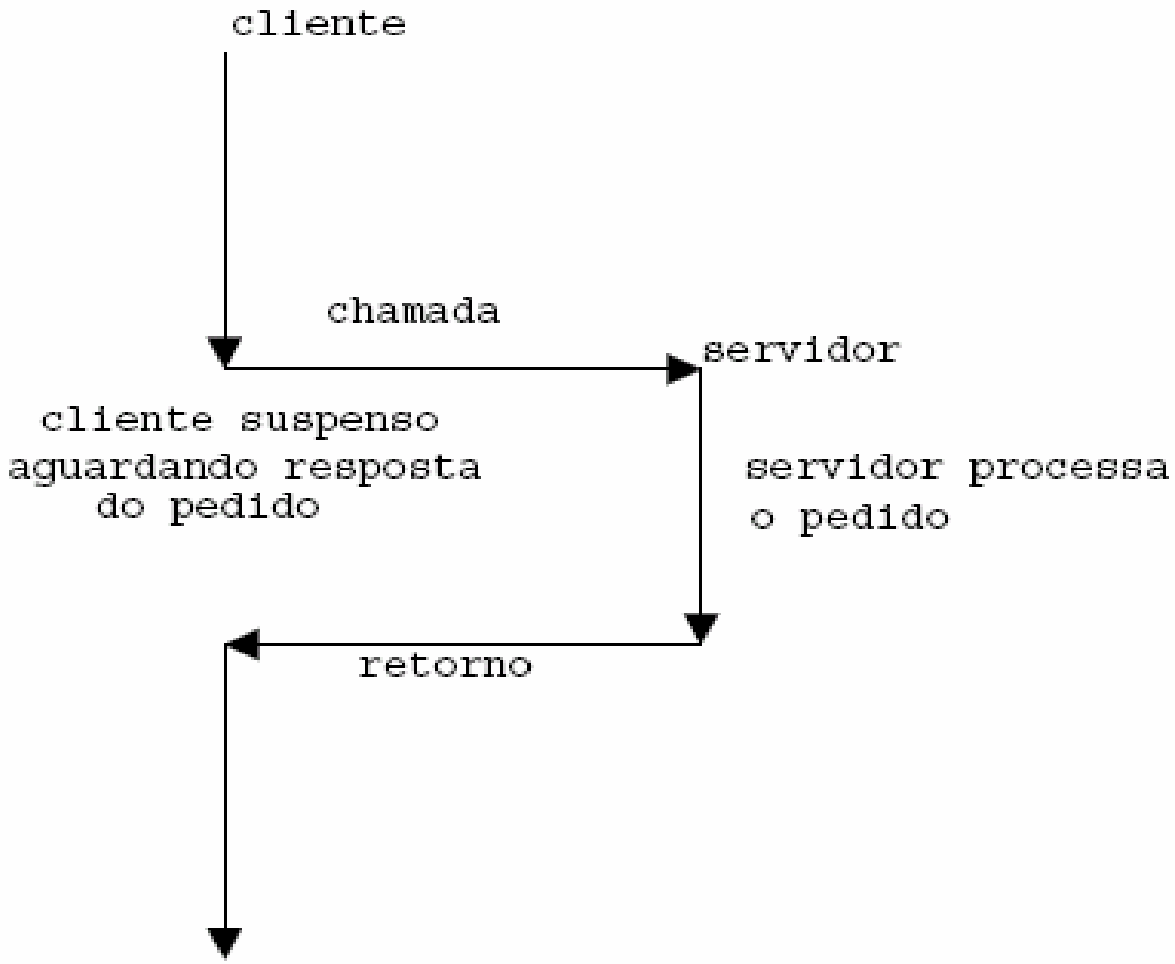


Figura 02.1.3.1 - Comunicação via RPC

Atributos de comunicação

- Nas aplicações C-S orientadas a bancos de dados, normalmente a utilização de RPCs não ocorre diretamente. As aplicações empacotam as operações SQL e as ativam através de APIs da SQL fornecida pelo vendedor. Veja Figura.

SGBD Cliente - Servidor

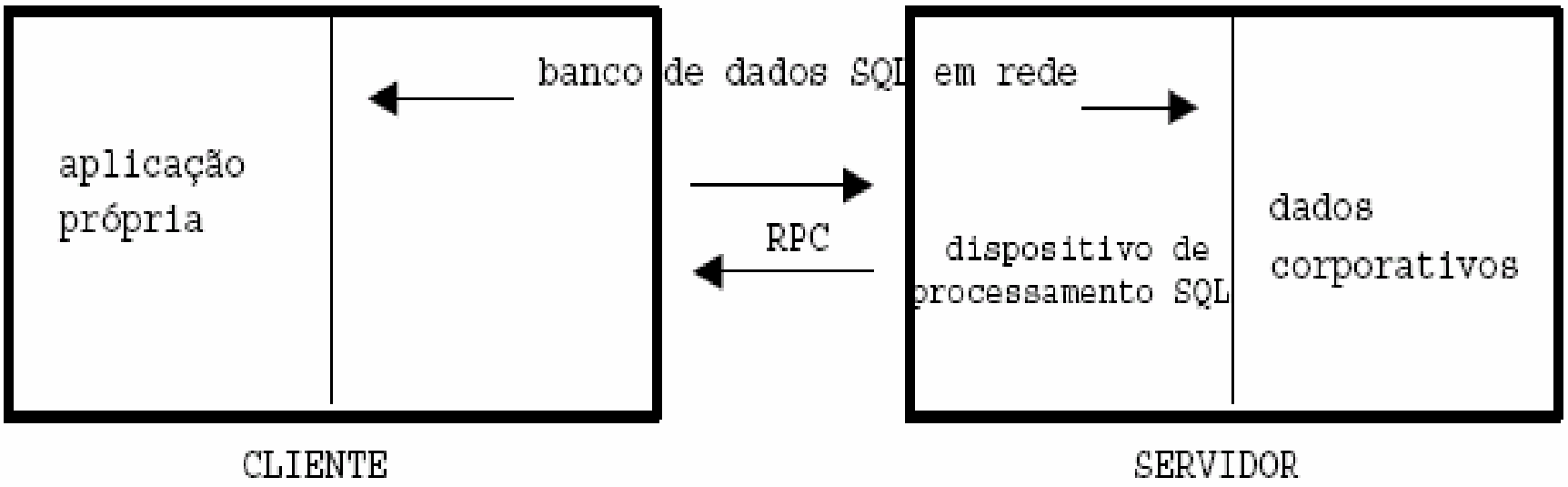


Figura 02.1.3.2 - Banco de dados SQL em rede

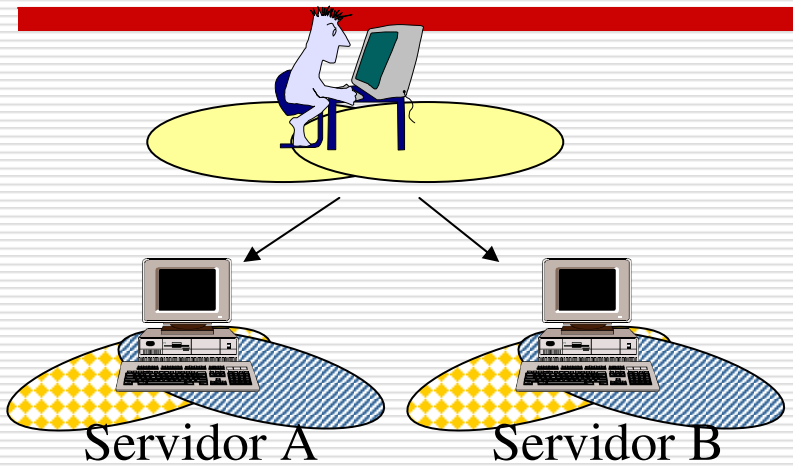
Vantagens da Arquitetura C-S

- Aceitação de tecnologia não proprietária → encoraja o uso de sistemas abertos.
- Disponibilidade de produtos.
- Aproveitamento do poder dos micro-computadores, estações de trabalho.
- Facilidade de utilização de GUIs
- Balanceamento de processamento → Pode ser reduzido o tráfego da rede.
- Gerenciamento centralizado → assegurar a confiabilidade do sistema distribuído. Monitoramento e suporte centralizado.
- Integração de serviços
- Flexibilidade → Os limites do aumento na carga de trabalho estão na capacidade do servidor.
- Escalabilidade → Os limites de incrementos de capacidade estão no tráfego suportado pela rede.
- Custos: Padronização dos sistemas abertos → maior competitividade → menores preços.

Desvantagens da arquitetura C-S

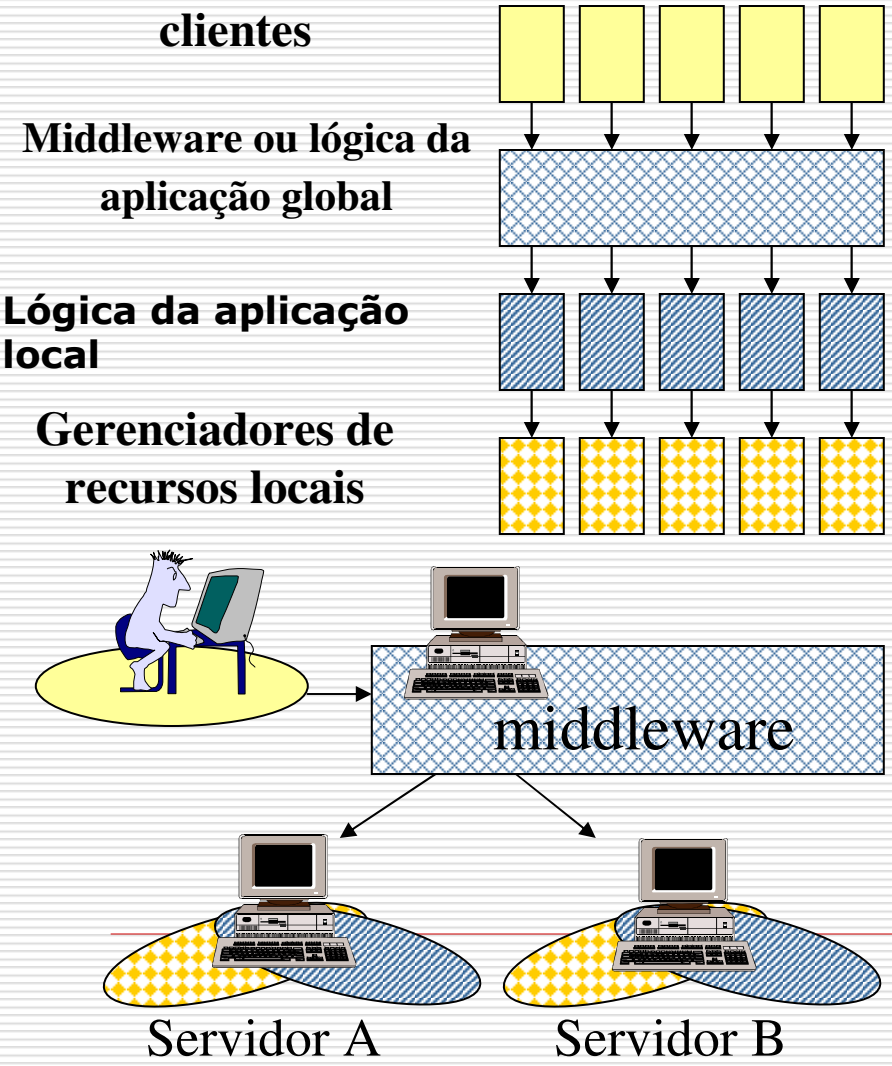
- Entretanto, nem tudo são vantagens:
  - O sistema tem que tratar com todas as possíveis conexões. s. O número máximo de clientes é dado pelo número de conexões suportadas pelo servidor.
  - Os clientes estão "ligados" ao sistema já que não existe uma camada de apresentação padrão. Se alguém deseja se ligar a dois sistemas, então precisa duas camadas de apresentação.
  - Não existe encapsulamento de falhas ou de cargas. Se o sistema falhar, ninguém pode trabalhar. Da mesma forma, a carga criada pelo cliente diretamente afetará o trabalho de outros já que eles todos estão competindo pelos mesmo recursos.
  - O projeto da lógica de aplicação e o gerenciador de recursos estão fortemente ligados, dificultando a mudança ou separação no caso de melhorar a eficiência.
  - O projeto continua sendo complexo e difícil para portar a outras plataformas.

O cliente sempre está certo



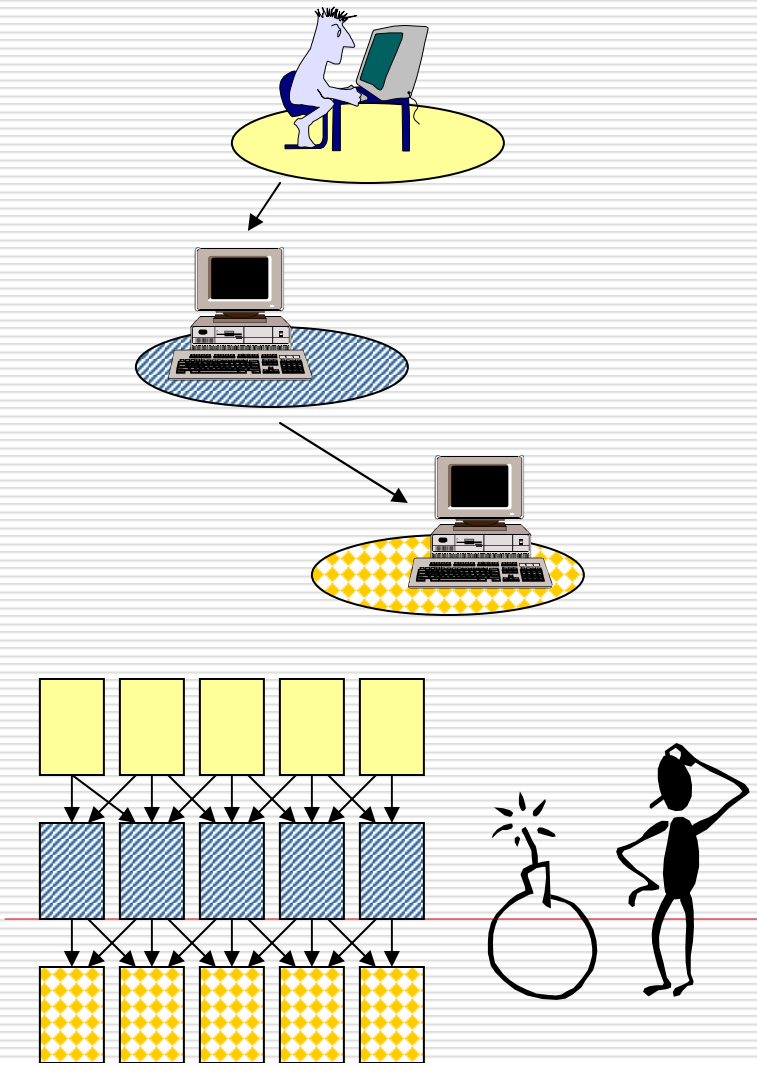
- Os Clientes finalizam desejando acessar dois ou mais sistemas. Com a arquitetura de 2 camadas, criam-se vários problemas:
  - Os sistemas não se conhecem entre eles; não existe uma lógica de negócios comum. Se ela for necessária, ela precisa ser implementada no cliente.
  - Os sistemas a serem tratados são provavelmente diferentes. A complexidade de tratar com dois sistemas heterogêneos precisa ser tratada pelo cliente.
  - O cliente fica responsável pelo conhecimento de onde as coisa estão, como obtê-las e como assegurar consistência!
- Isto é muito ineficiente desde qualquer ponto de vista (muitos clientes pesados não são uma solução).
- Há muito pouco que pode ser feito para solucionar estes problemas dentro do modelo de 2 camadas. Isto pode ser resolvido adicionando um nível de indireção: **MIDDLEWARE**

Middleware



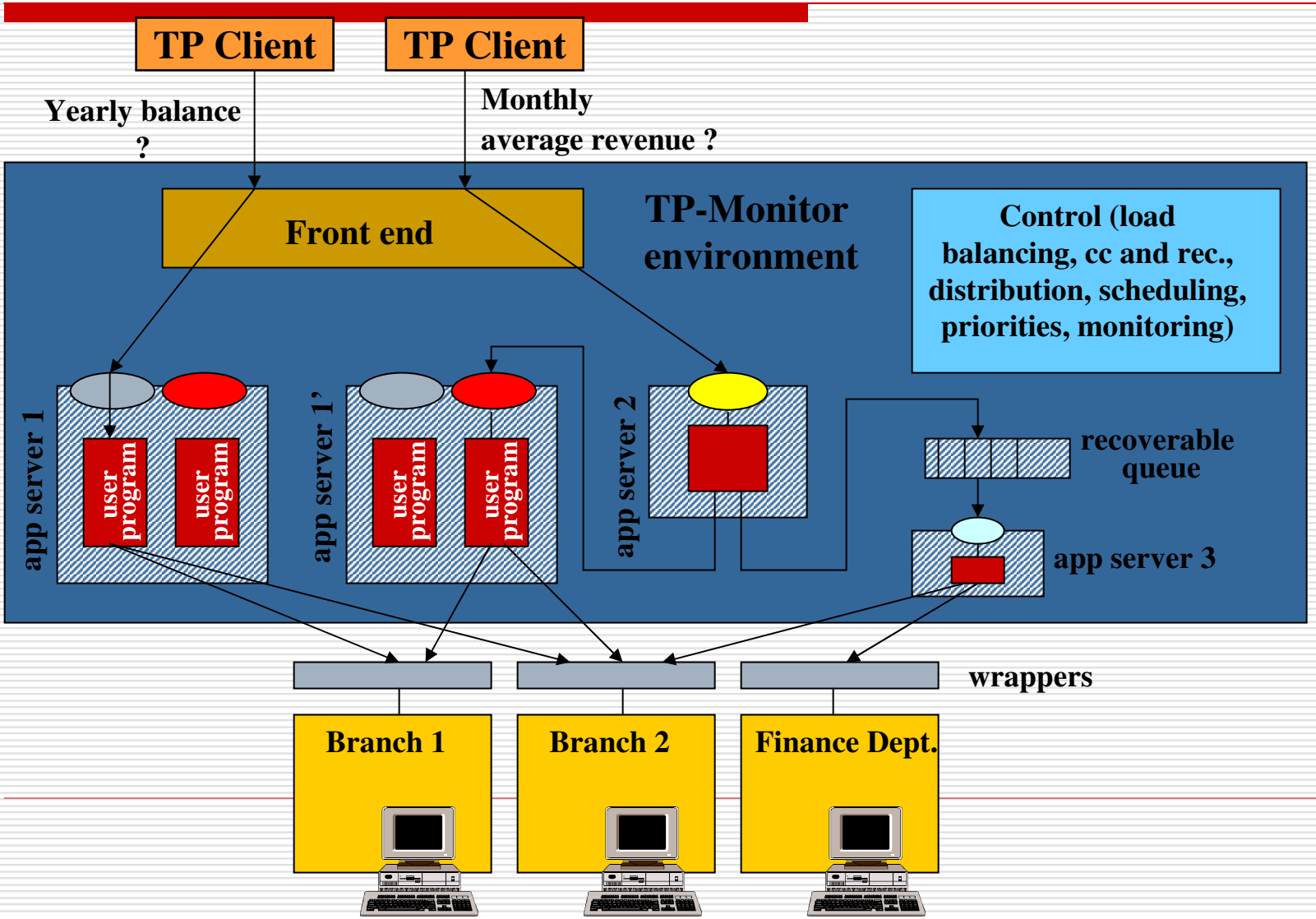
- Middleware é justo um nível de indireção entre os clientes e outras camadas do sistema.
- Ele introduz uma camada adicional da lógica de negócios comum a todos os sistemas.
- Fazendo isto, um sistema de middleware:
  - simplifica o projeto dos clientes reduzindo o número de interfaces,
  - Oferece acesso transparente aos sistemas definidos,
  - atua como a plataforma para a funcionalidade entre sistemas e como uma lógica de aplicação de alto nível, e
  - Cuida da localização de recursos, de seu acesso, e da coleta de resultados.

Arquitetura de 3 camadas



- Em um sistema de 3-camadas, as três são totalmente separadas.
- Para alguns, um sistema baseado em middlewares é um arquitetura de 3-camadas. Isto é uma simplificação mas é conceitualmente correto.
- Veremos alguns exemplos de middlewares..
- Na prática, as coisas não são tão simples como elas parecem … existem várias camadas escondidas que não são necessariamente triviais: os wrappers ou extratores.

Um exemplo


TP Client
TP Client
Yearly balance ?
Monthly average revenue ?
Front end
TP-Monitor environment
Control (load balancing, cc and rec., distribution, scheduling, priorities, monitoring)
app server 1
user program
user program
app server 1'
user program
user program
app server 2
recoverable queue
app server 3
wrappers
Branch 1
Branch 2
Finance Dept.

Tipos de Middleware

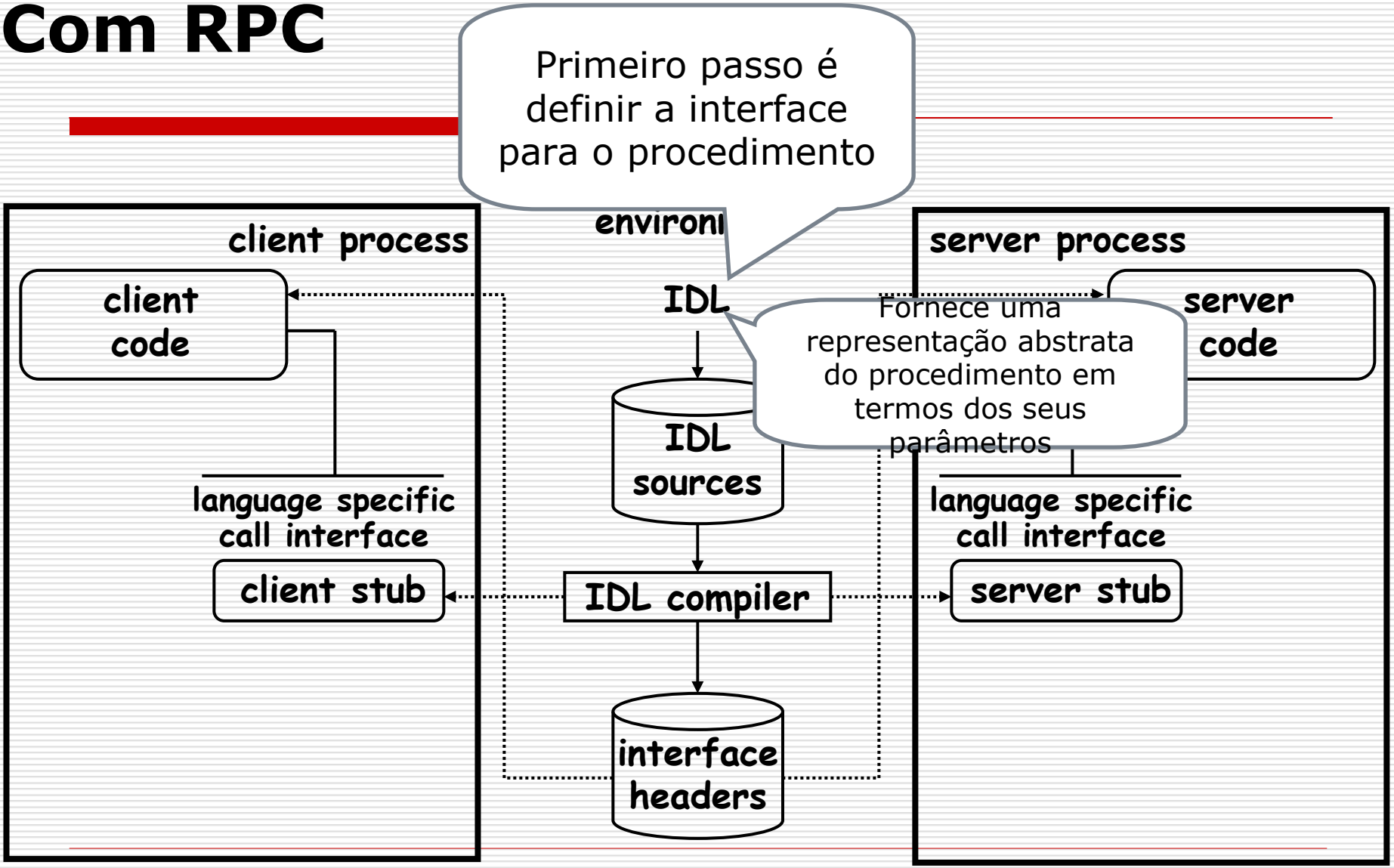
- Sistemas baseados em RPC
- Monitores de processamento de Transações
- Brokers de Objetos → CORBA (Common Object Request Broker Architecture)
- Monitores de Objetos
- Middleware orientado a mensagens.

RPC e Linguagens de Programação

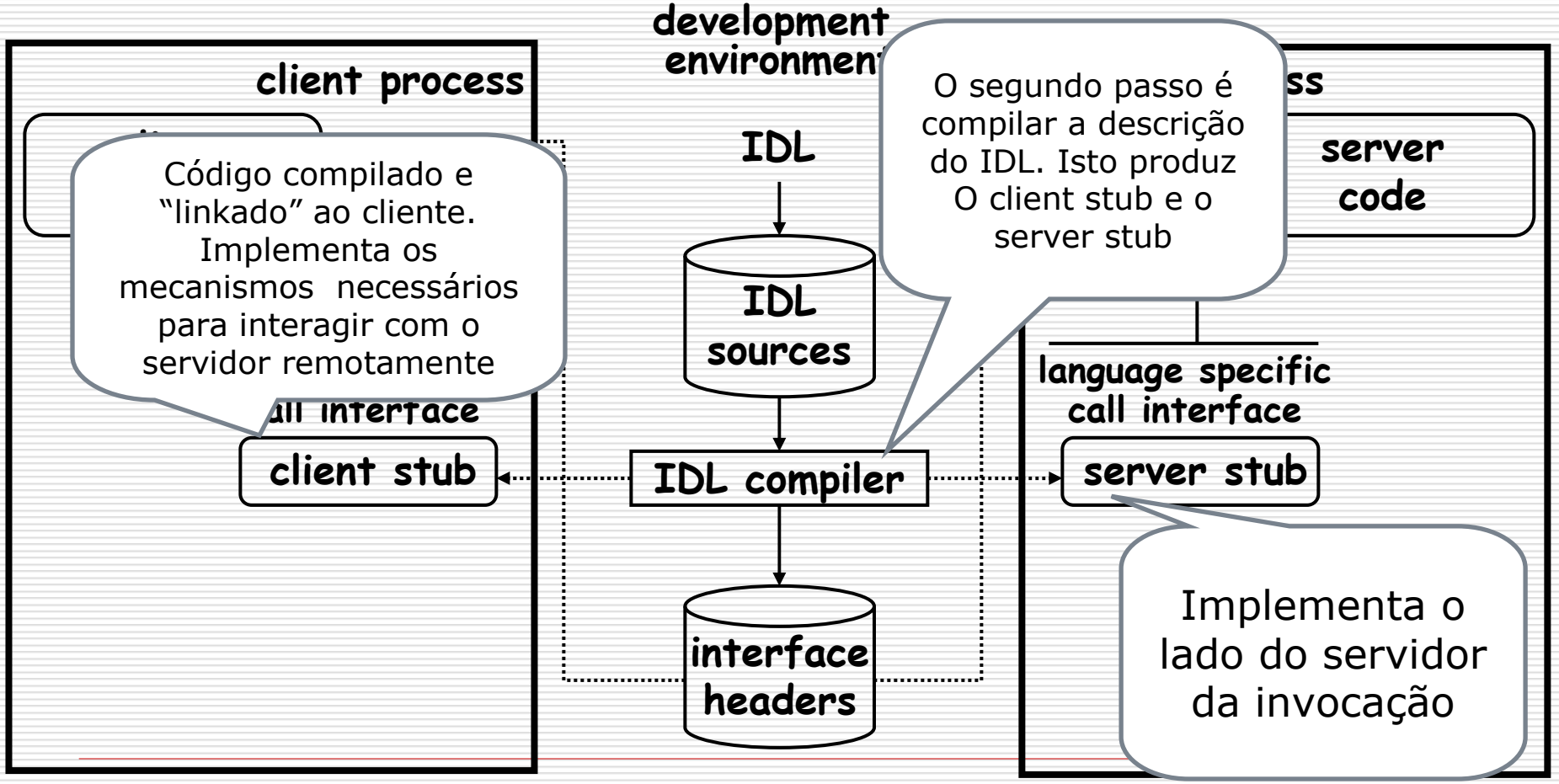
- A noção de invocação distribuída de serviços virou uma realidade no começo dos 80's quando linguagens procedimentais (princip. C) foram dominantes.
- Nas linguagens procedimentais, o módulo básico é o procedimento. Um procedimento implementa uma função ou serviço específico que pode ser chamado em qualquer lugar do programa.
- Parece natural mainter esta mesma noção quando se fala de distribuição: o cliente chama um procedimento que é implementado pelo servidor. Já que o cliente e o servidor podem estar em diferentes máquinas, o procedimento é remoto.
- Arquiteturas C/Server architectures are based on Remote Procedure Calls (RPC)

- Uma vez estejamos trabalhando com procedimentos remotos em mente, existem vários pontos que são determinados imediatamente:
  - Troca de dados é feita como parâmetros de entrada e saída da chamada de procedimento
  - pointers não podem ser passados como parâmetros na RPC, referências opacas são necessárias no lugar para que o cliente possa usar esta referência para se referir à mesma estrutura de dados ou entidade no servidor através de diferentes chamadas. *O servidor é responsável por fornecer estas referências opacas*.

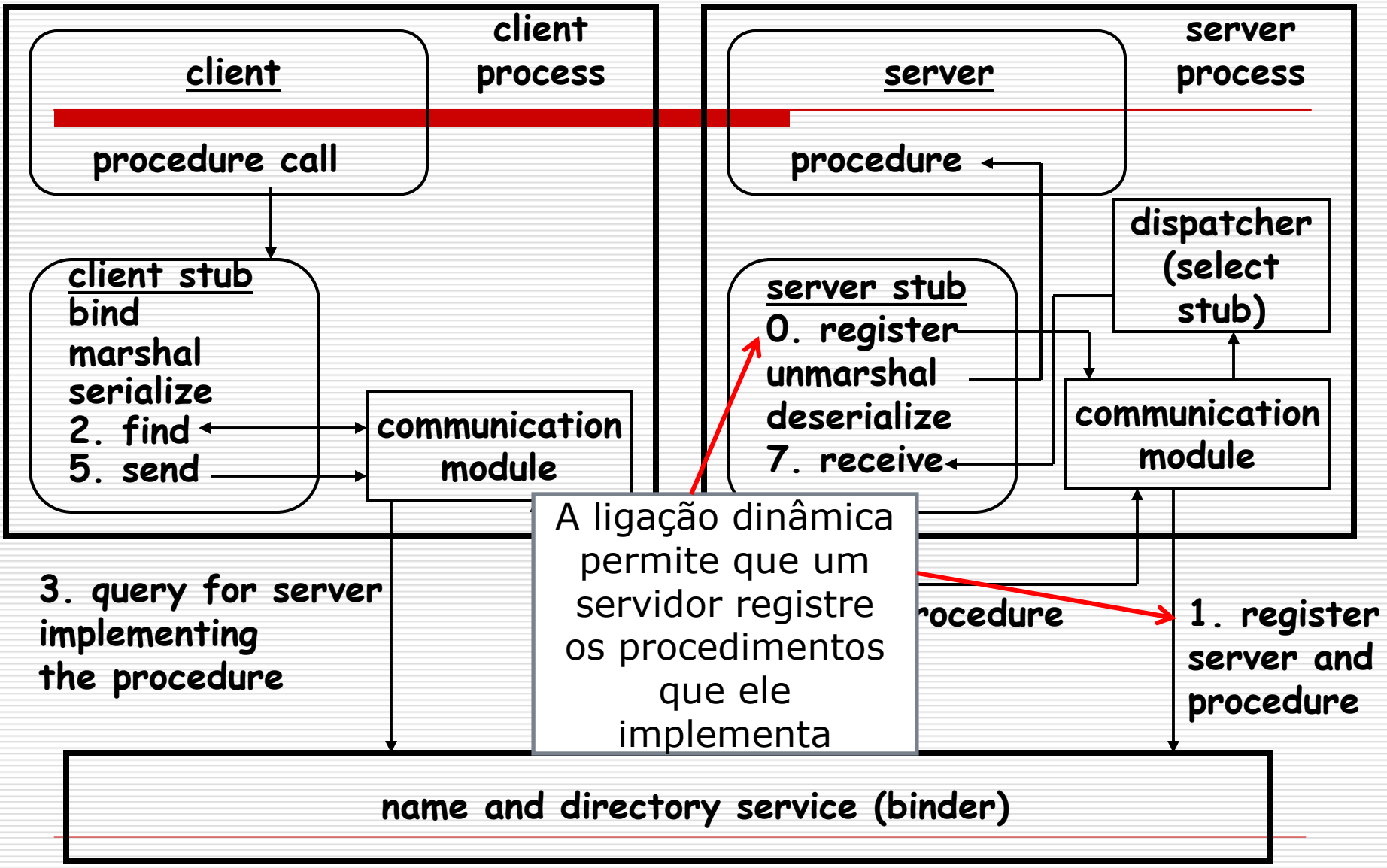
Desenvolvendo aplicações distribuídas Com RPC



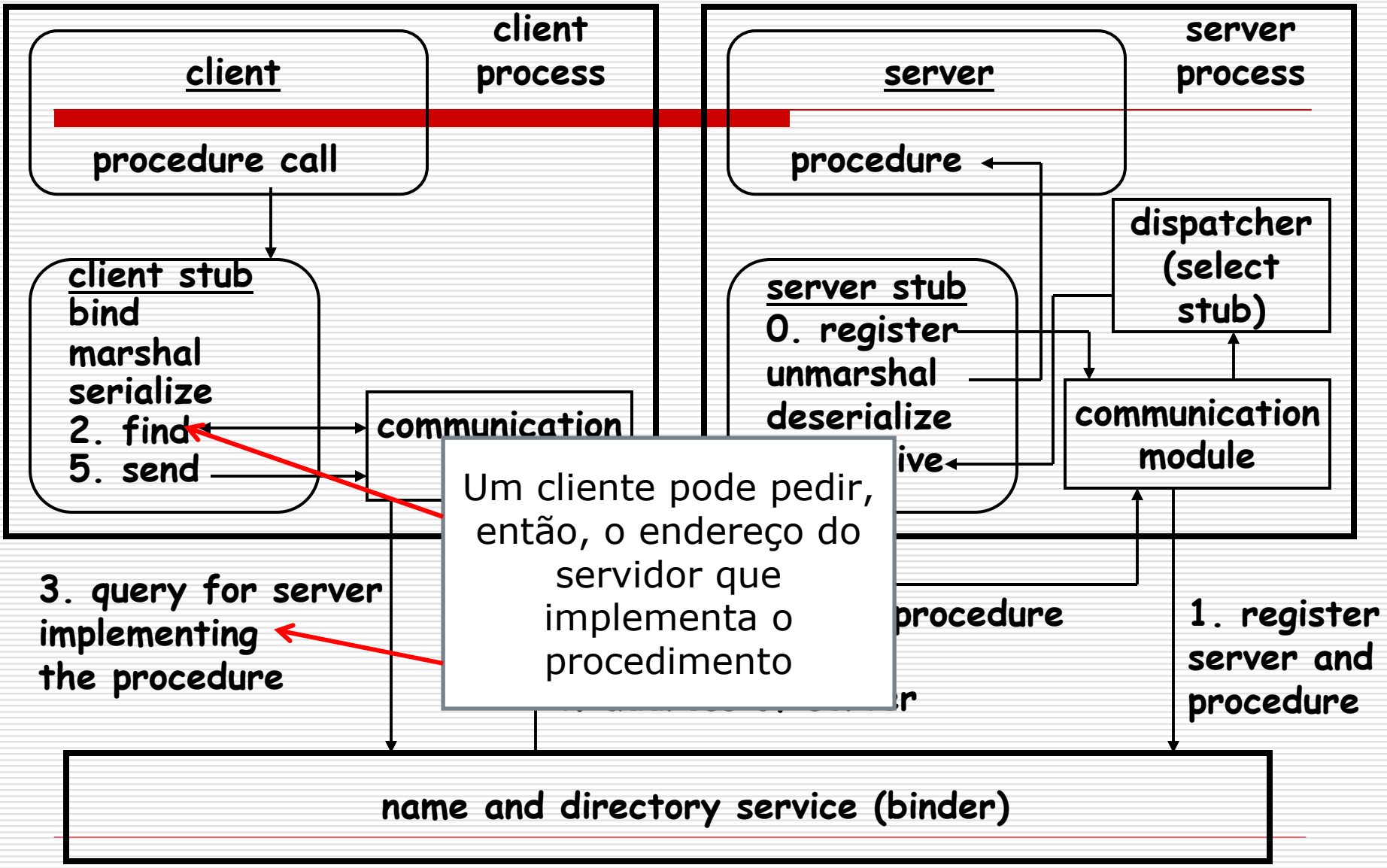
Desenvolvendo aplicações distribuídas Com RPC



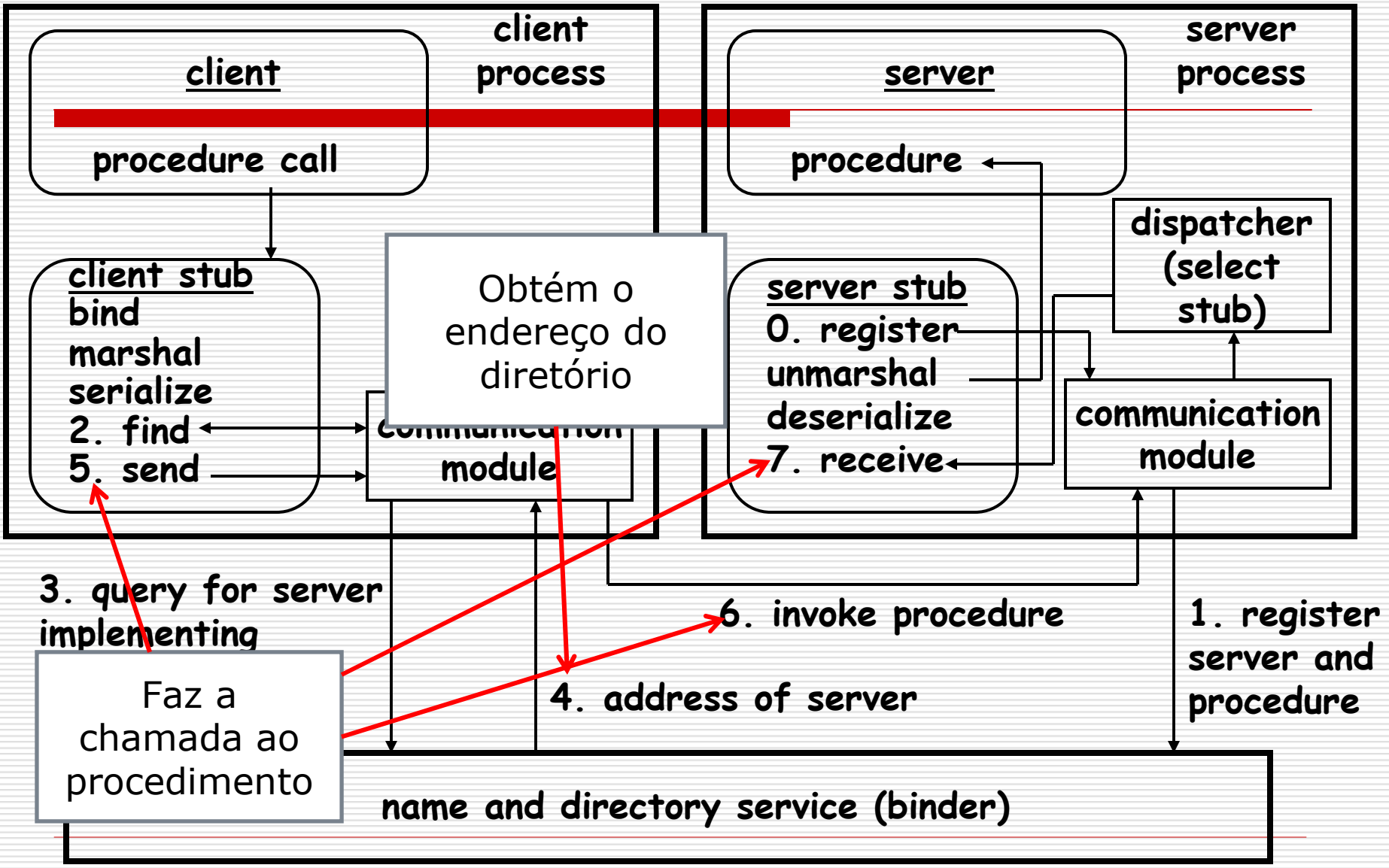
Ligação RPC


client process
client
procedure call
client stub
bind
marshal
serialize
2. find
5. send
communication module
server process
server
procedure
dispatcher (select stub)
server stub
0. register
unmarshal
deserialize
7. receive
communication module
A ligação dinâmica permite que um servidor registre os procedimentos que ele implementa
3. query for server implementing the procedure
1. register server and procedure
name and directory service (binder)

Ligação RPC

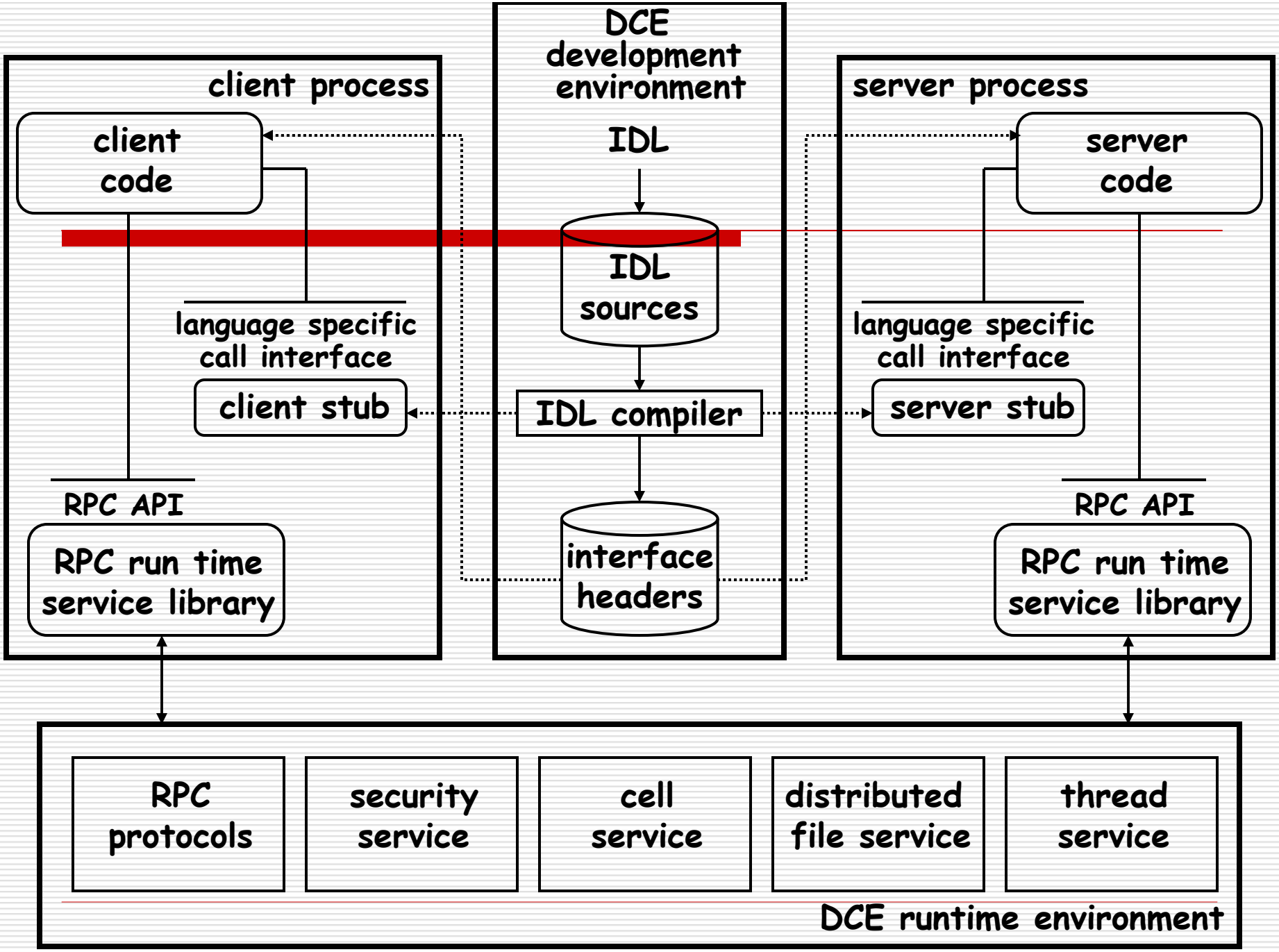

client process
client
procedure call
client stub
bind
marshal
serialize
2. find
5. send
communication
server process
server
procedure
server stub
0. register
unmarshal
deserialize
dispatcher (select stub)
communication module
Um cliente pode pedir, então, o endereço do servidor que implementa o procedimento
3. query for server implementing the procedure
1. register server and procedure
name and directory service (binder)

Ligação RPC


client process
client
procedure call
client stub
bind
marshal
serialize
2. find
5. send
Obtém o endereço do diretório
module
server process
server
procedure
server stub
0. register
unmarshal
deserialize
7. receive
dispatcher (select stub)
communication module
3. query for server implementing
6. invoke procedure
1. register server and procedure
4. address of server
Faz a chamada ao procedimento
name and directory service (binder)

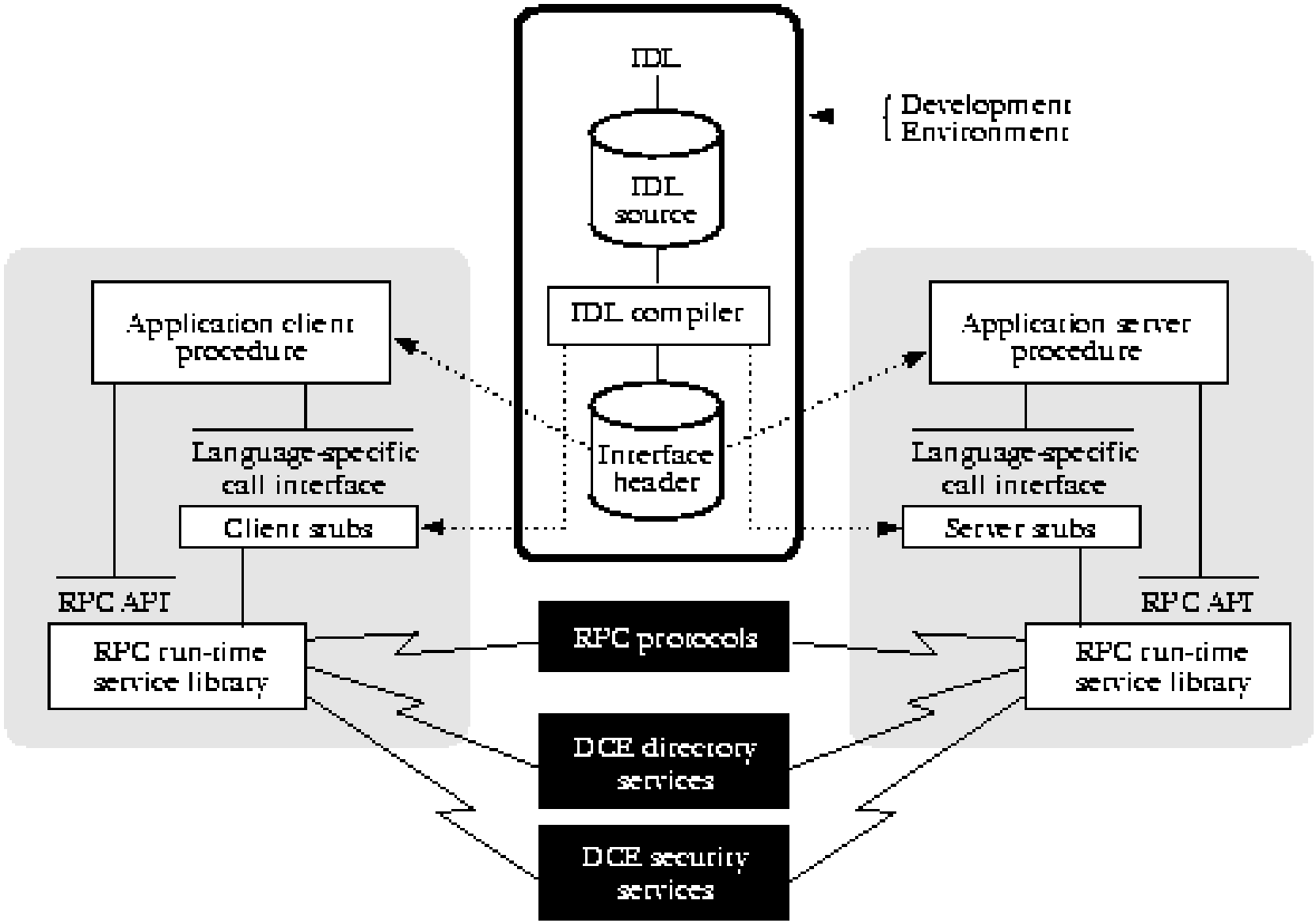
RPC e heterogeneidade

- Objetivos originais do projeto de RPC, servir como mecanismo para ligar sistemas heterogêneos
- Os stubs podem facilitar a comunicação entre múltiplas plataformas.
- RPC usa IDL não só para definir interfaces mas também para definir o mapeamento desde LPs concretos a representações intermediárias usadas na RPC. IDL serve também para definir a representação intermediária para a troca de dados entre clientes e servidores.



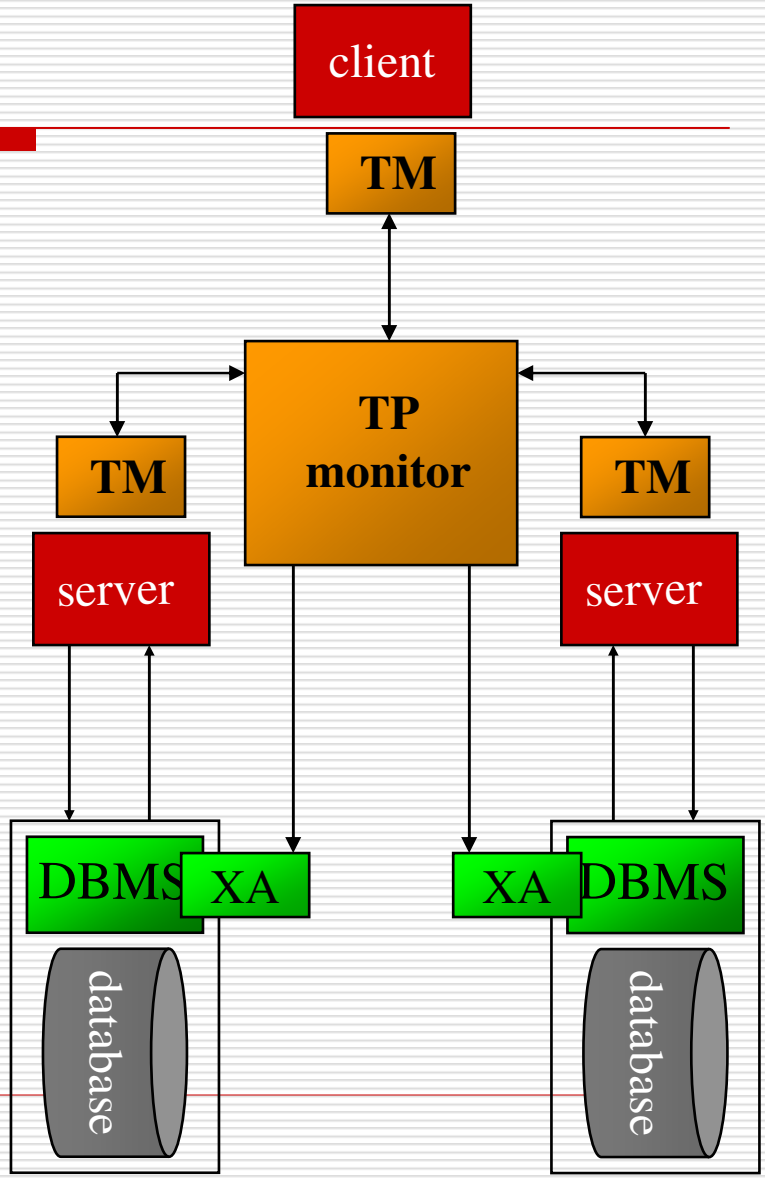
Infraestrutura Middleware RPC: DCE

Sistemas baseados em RPC

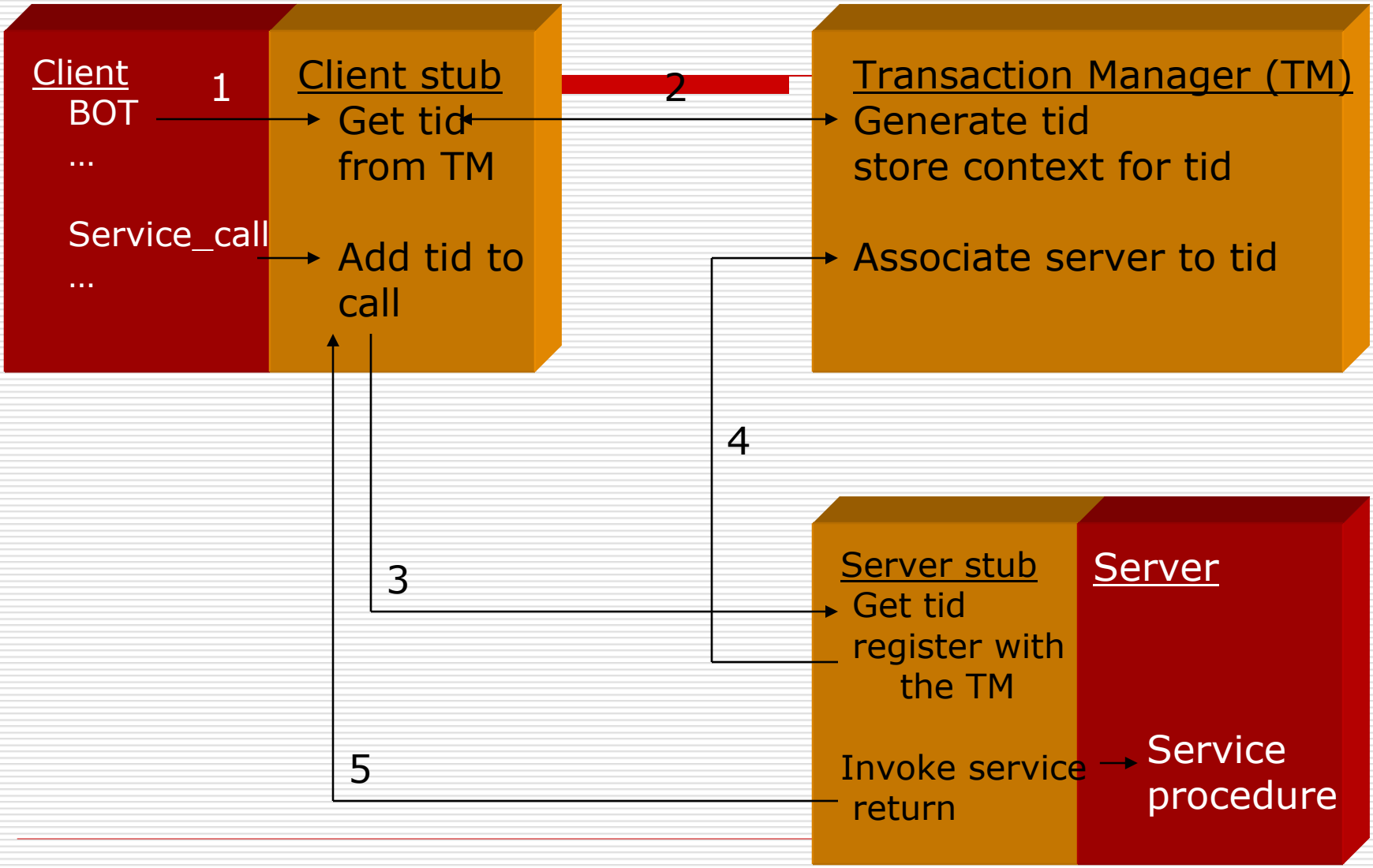


Monitores de processamento de transações

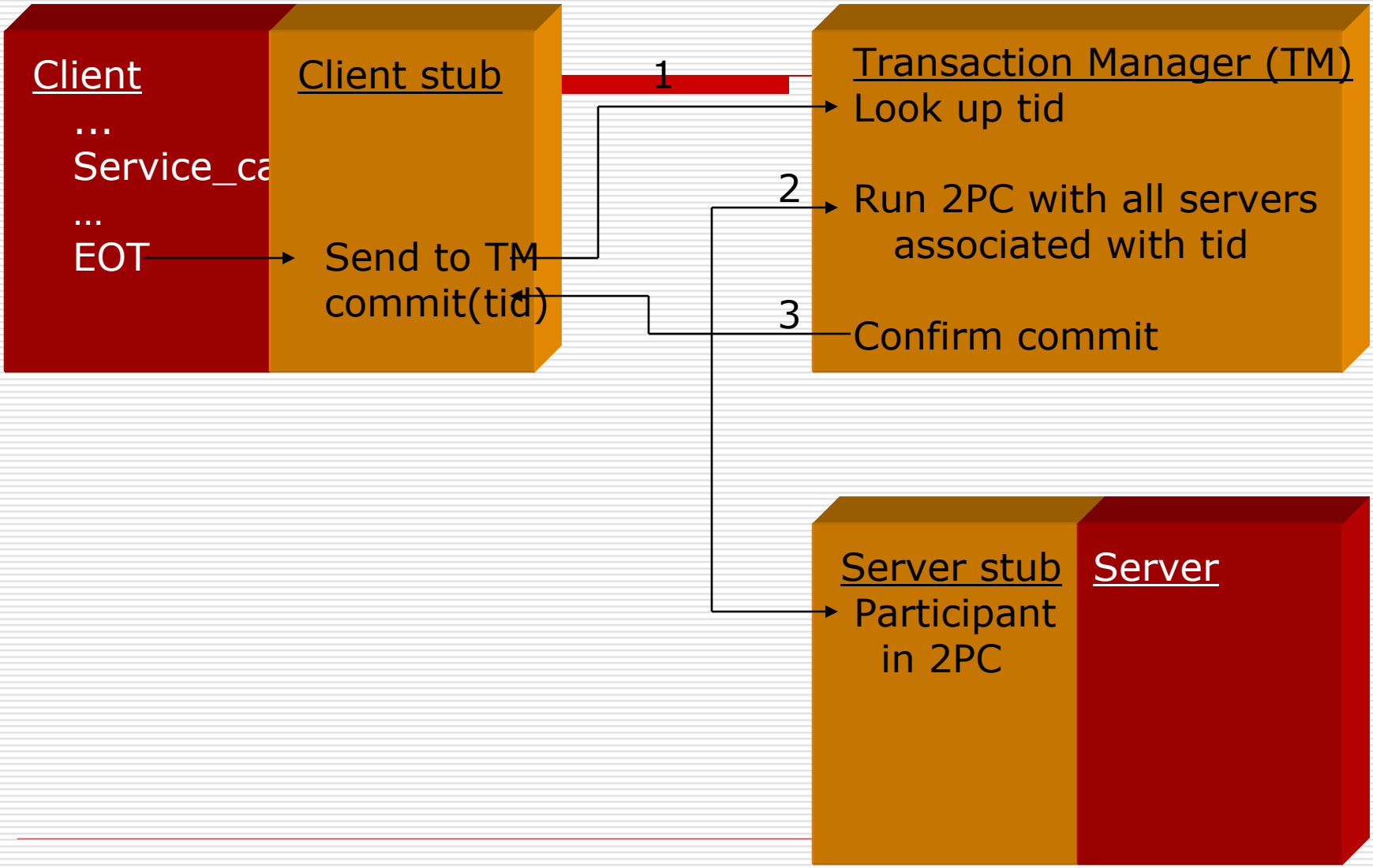
- As limitações da RPC em termos de confiabilidade podem ser resolvidas fazendo chamadas RPC transacionais. Na prática, Isto significa que elas são controladas pelo protocolo 2PC
- Uma entidade intermediária é necessária para executar 2PC (o cliente e o servidor poderiam fazer isto eles mesmos mas não é prático nem genérico)
- Esta entidade intermediária é chamada usualmente de gerenciador de transações (TM) e atua como intermediário em todas as interações entre os clientes, e gerenciadores de recursos
- Quando todos os serviços necessários para suportar RPC, RPC transacional, e características adicionais são adicionadas à camada intermediária, o resultado é um Monitor TP

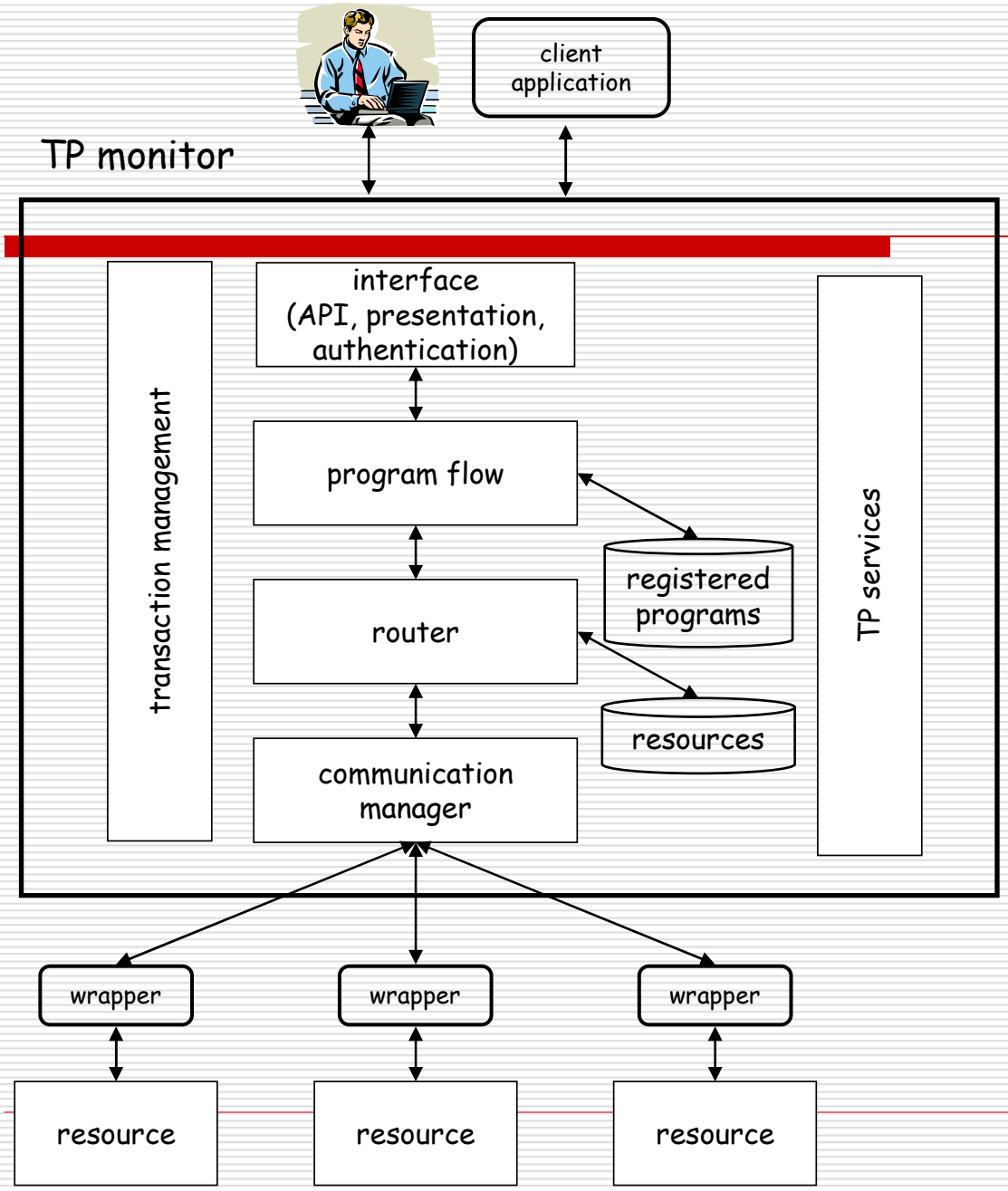


TRPC Básico (Fazendo as chamadas)



TRPC Básico (Confirmando as chamadas)

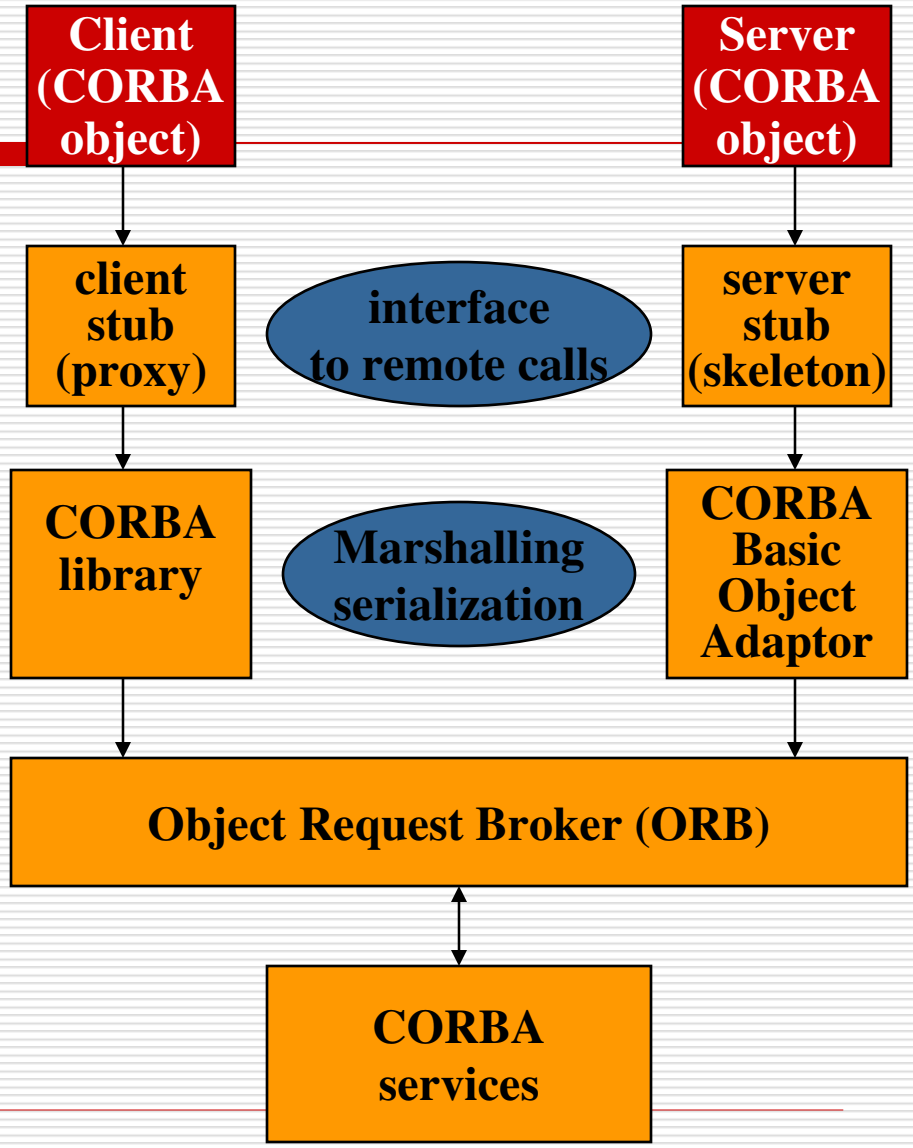




Arquitetura de um monitor TP

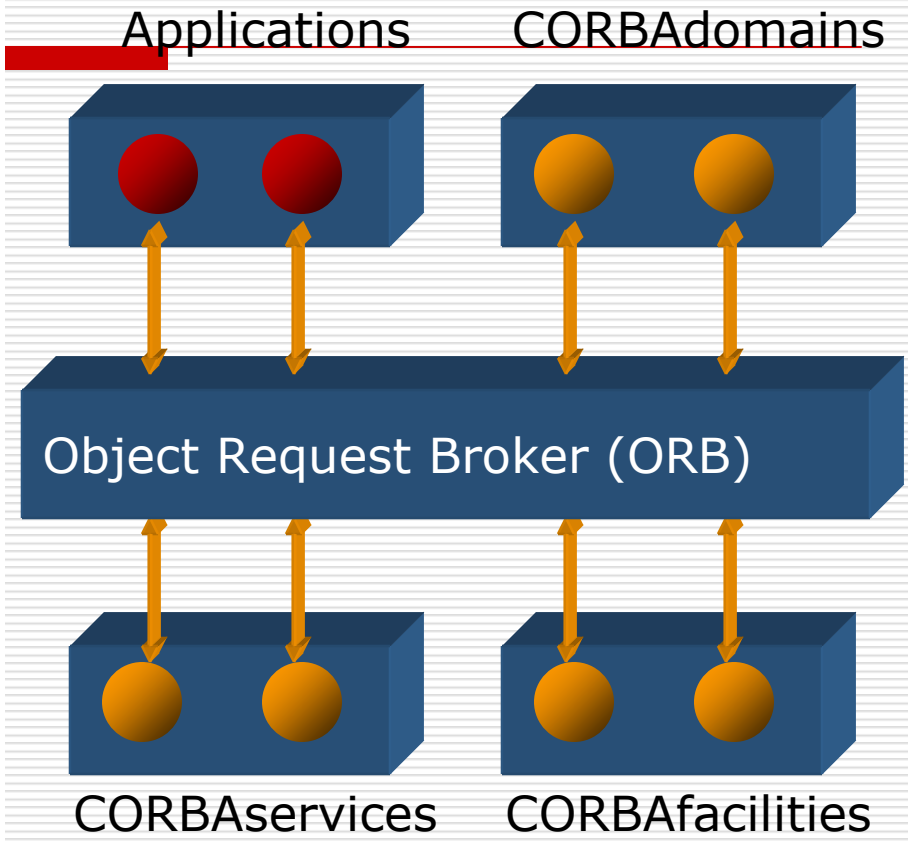
CORBA

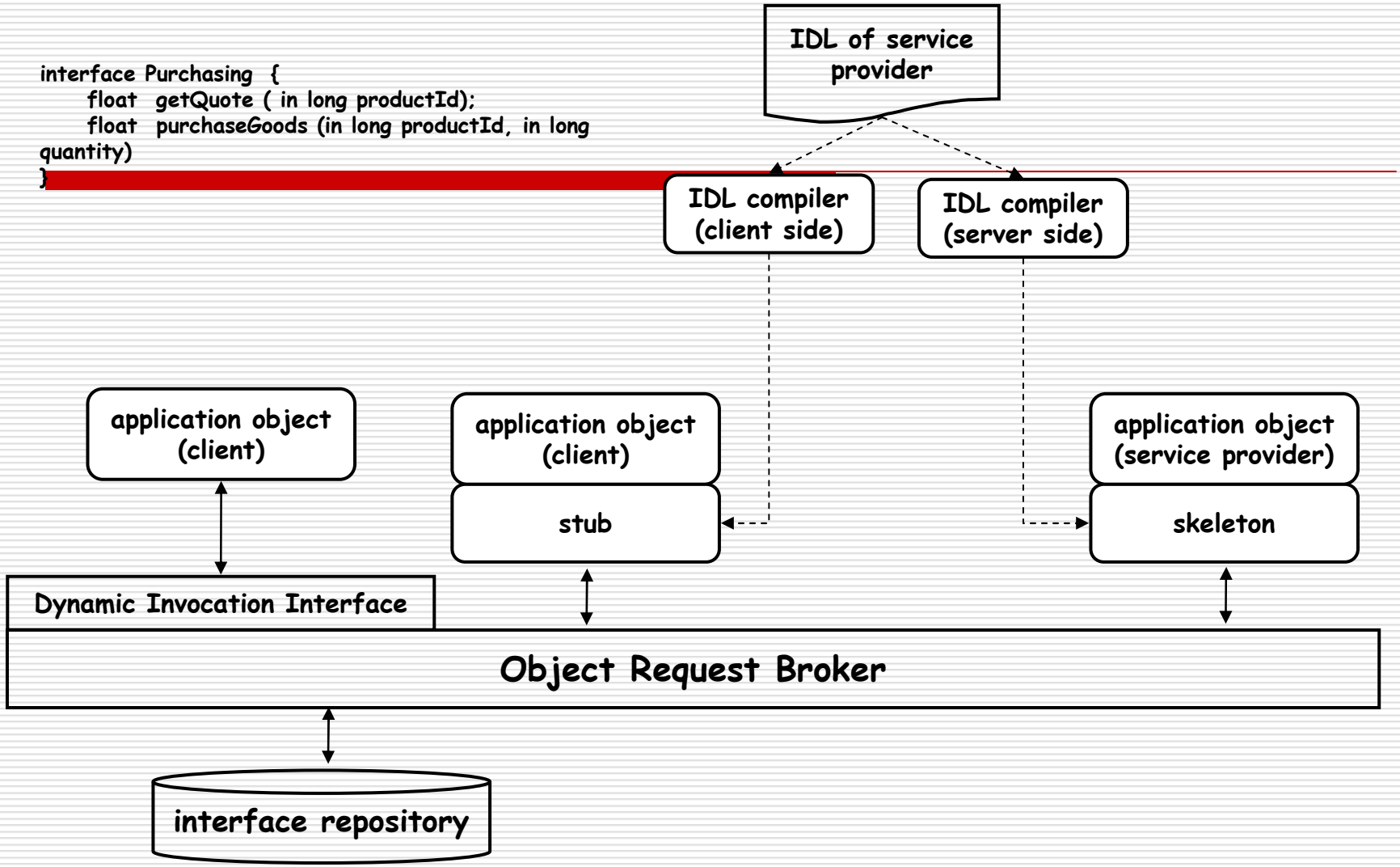
- O Common Object Request Broker Architecture (CORBA) é parte da Arquitetura padrão de Gerenciamento de Objetos (OMA), uma arquitetura de referência para sistemas baseados em componentes
- As partes chaves de CORBA são:
  - Object Request Broker (ORB): encarregado da interação entre componentes
  - Serviços CORBA: definição padrão de serviços de sistemas
  - Uma linguagem padrão IDL para a publicação de interfaces
  - Protocolos que permitem ORBs conversar entre eles
- Estende o paradigma RPC ao mundo orientado a objetos.
- Não se invocam procedimentos senão métodos.



Arquitetura orientada a objetos (OMA)

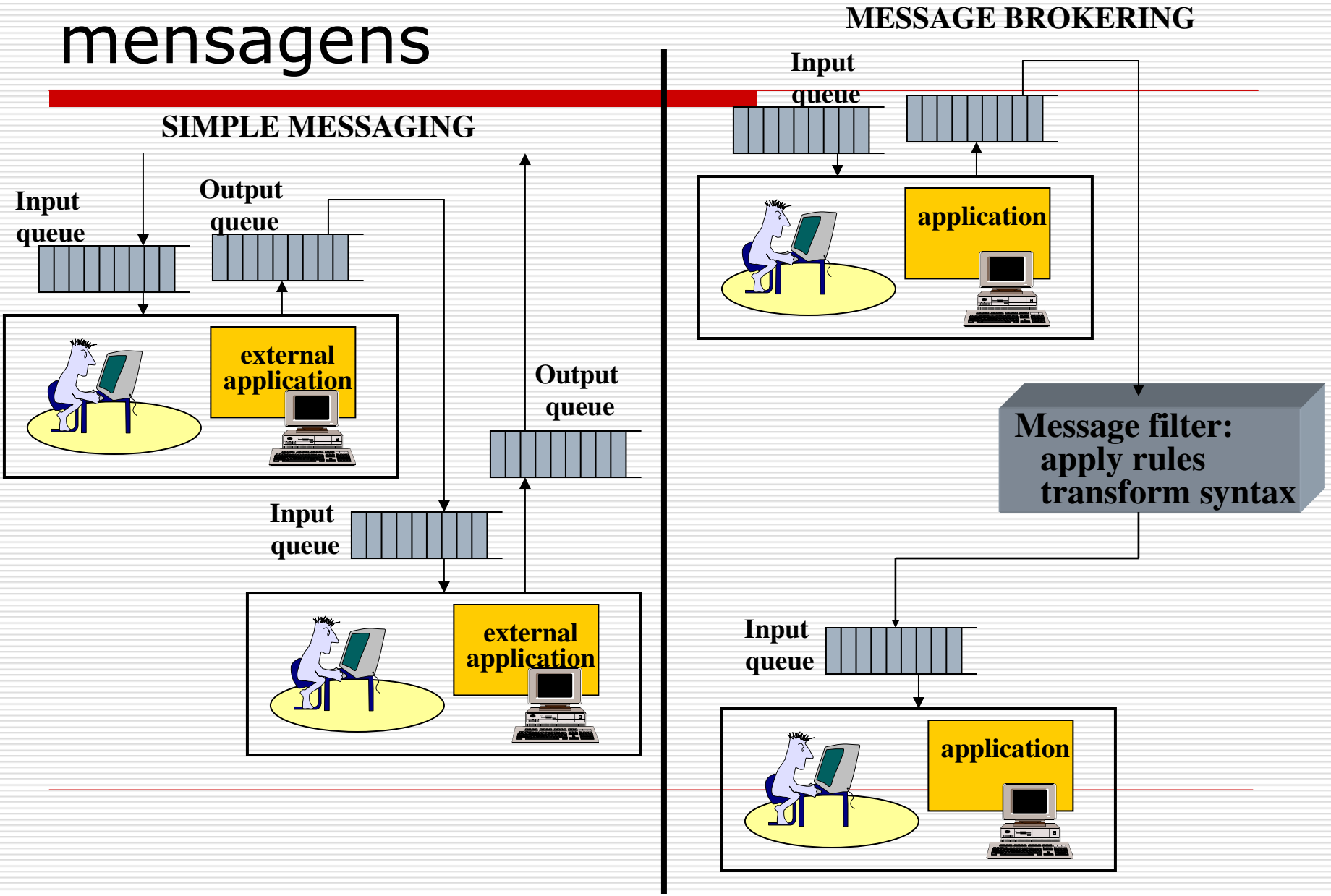
- A OMA define a arquitetura e o conjunto de serviços que suportam a computação distribuída heterogênea baseada em objetos. Seus componentes são:
  - ORB: infraestrutura de comunicação, gerenciamento e monitoração de servidores, marshalling, localização de serviços, etc.
  - CORBAservices: funcionalidade básica de qualquer sistema de middleware (transações, segurança)
  - CORBAdomains: domínios de negócios verticais (finanças, saúde, produção, etc.)
  - CORBAfacilities: serviços que são comuns às aplicações (serviços de impressão, internacionalização, controle de erros)





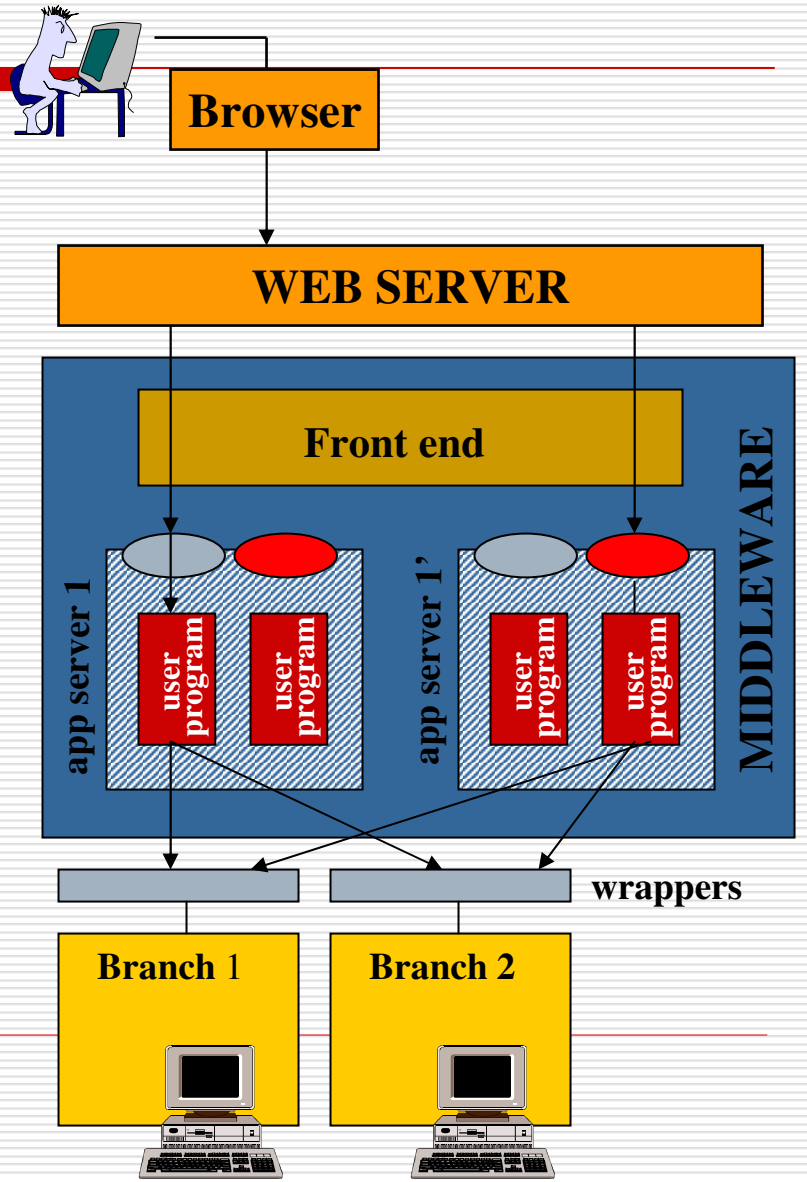
Compilação da especificação IDL

Middlewares orientados às mensagens



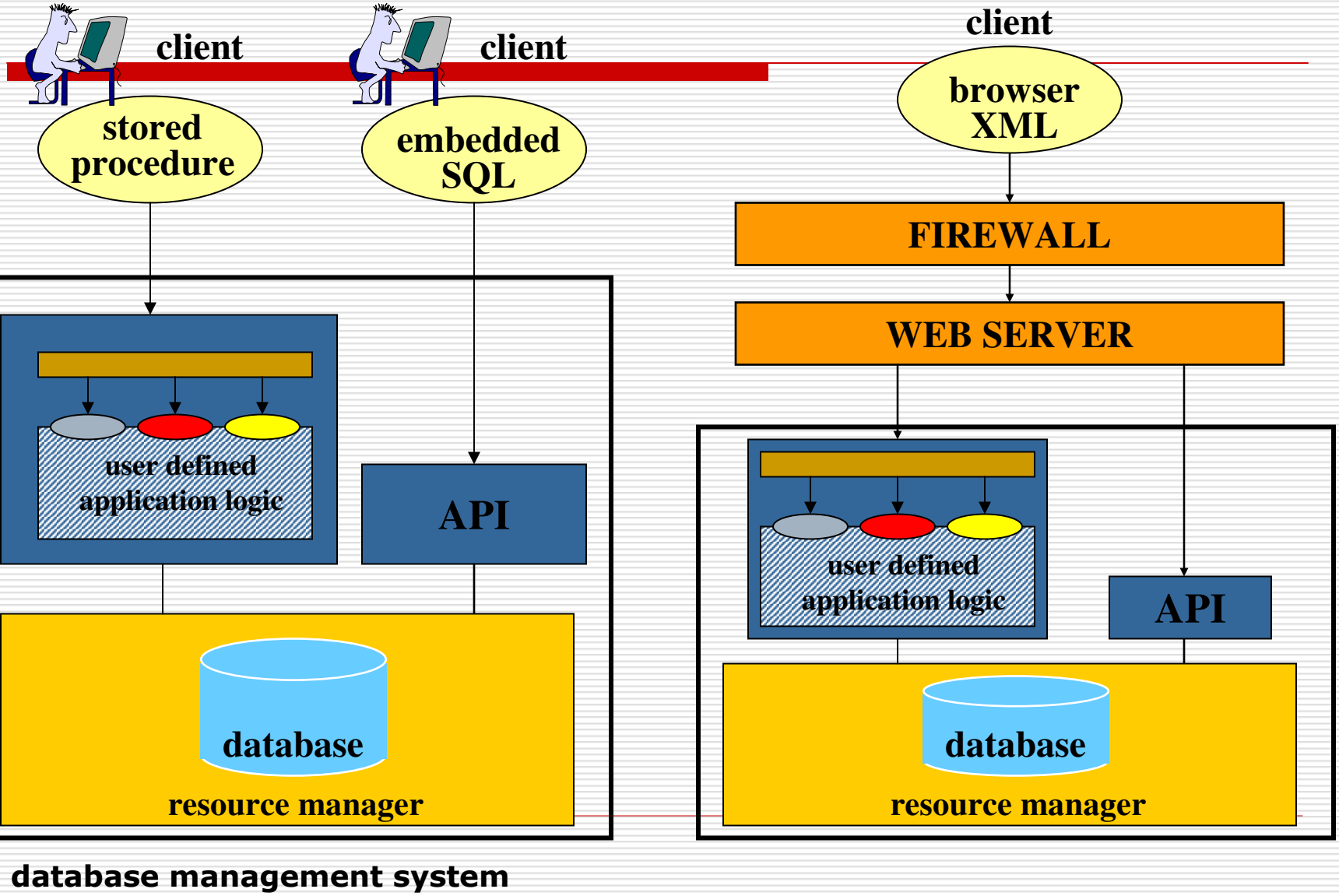
O mundo dos serviços

- A Web de repente abriu uma série de softwares que tinham permanecido escondidos dentro da organização de TI de uma companhia
- A natureza da interação não mudou. Por trás da web existe o mesmo modelo cliente/servidor com o básico RPC. Entretanto, a Web fez as coisas serem mais fáceis, baratas e eficientes
  - Integração na interface do usuário ficou possível
  - serviços poderiam ser acessados desde qualquer lugar do mundo
  - Os clientes poderiam agora ser agora qualquer um com um browser



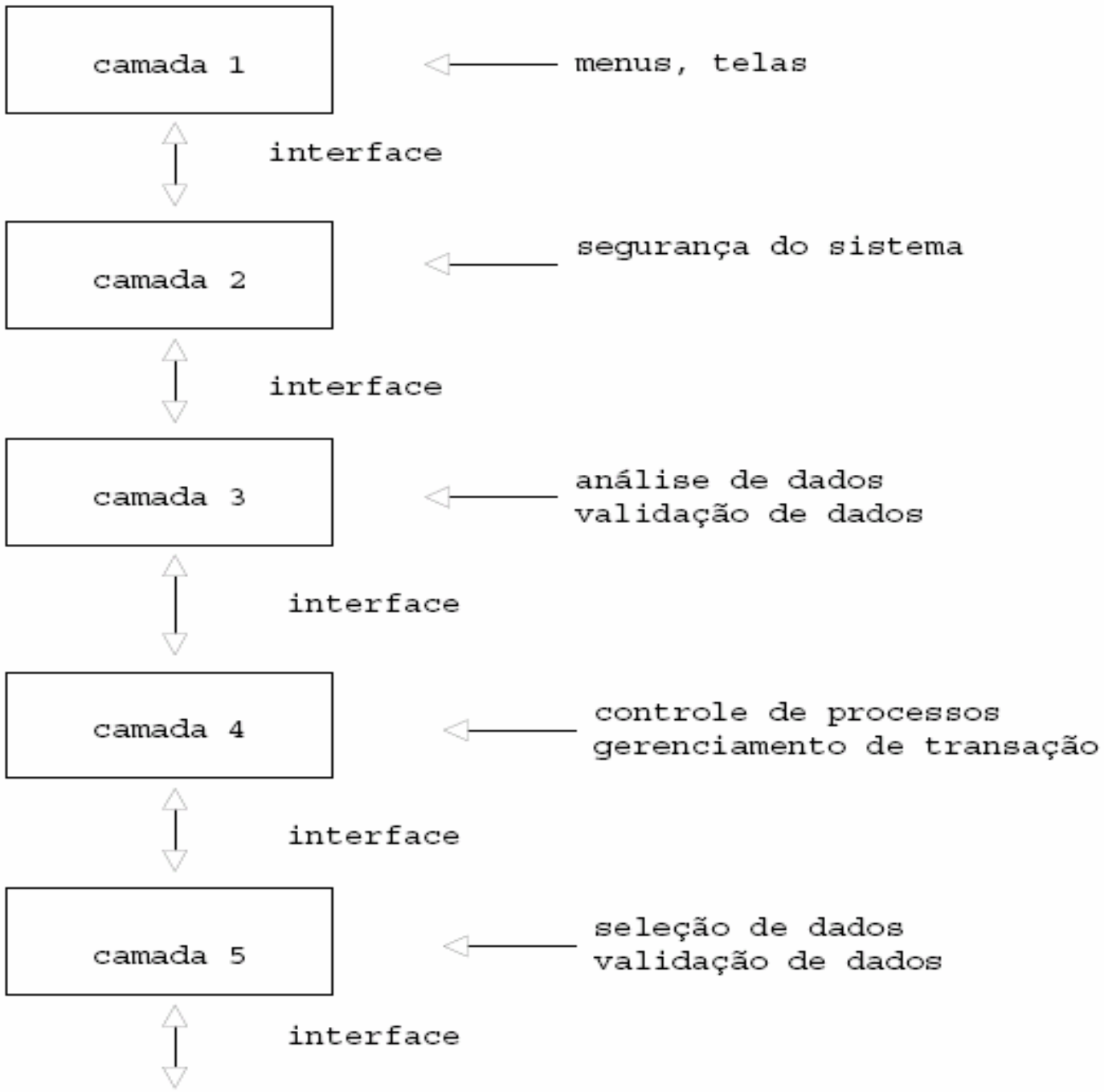
Clientes remotos

Falemos um pouco de serviços



Arquitetura de camadas

- Então, num sistema distribuído o código da aplicação pode ser dividido em camadas, onde cada camada é responsável pelo desempenho de uma ou mais funções no sistema. Veja Fig.
- Uma camada, na realidade, representa um conjunto de componentes de software, os quais são agrupados num determinado nível, tendo estas camadas uma responsabilidade determinada no contexto da aplicação e funções específicas exercidas por seus componentes.


camada 1
menus, telas
interface
camada 2
segurança do sistema
interface
camada 3
análise de dados
validação de dados
interface
camada 4
controle de processos
gerenciamento de transação
interface
camada 5
seleção de dados
validação de dados
interface

Continuação

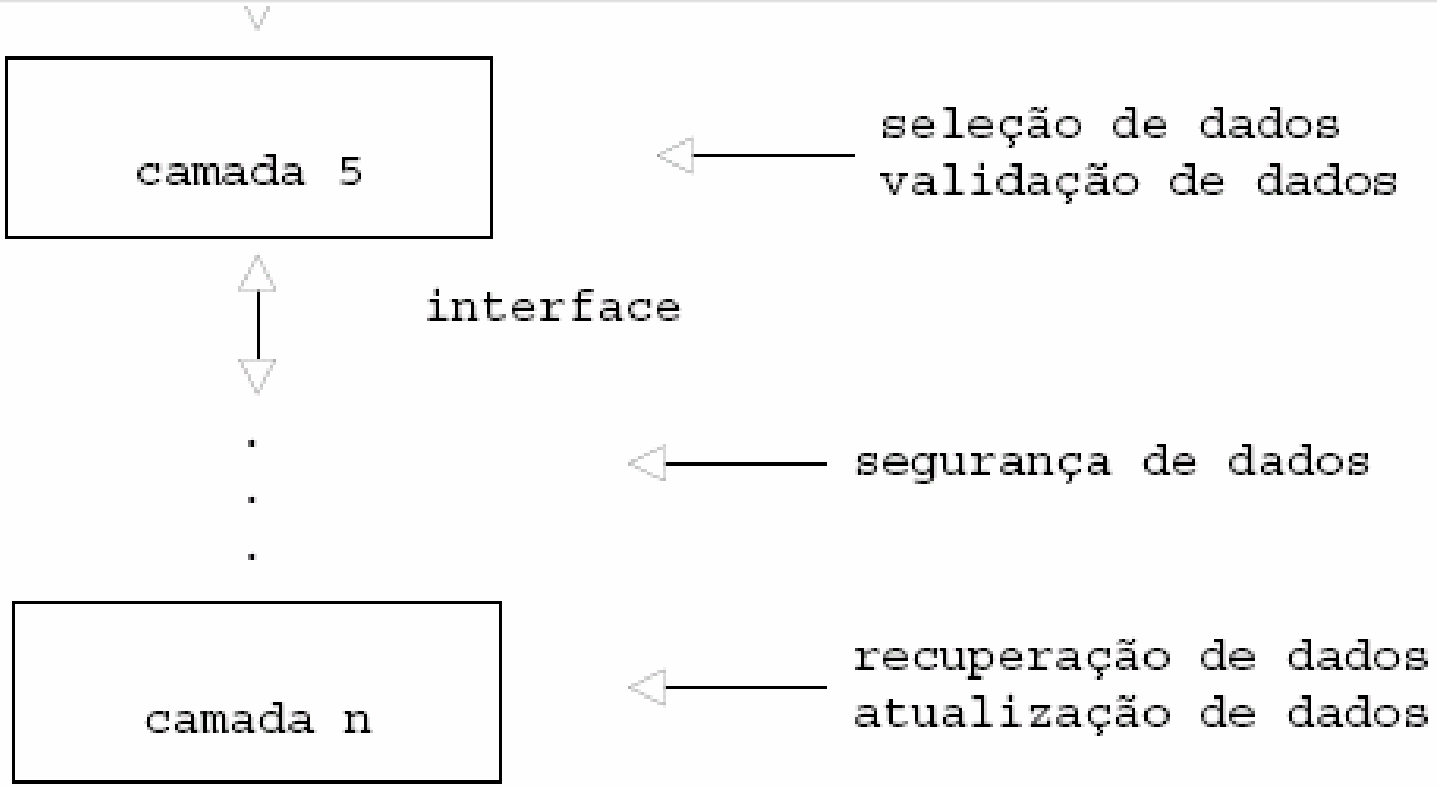


Figura 6.1.2 - Sistema distribuído em camadas

Elementos da arquitetura em camadas

- Três elementos: a camada, a mensagem e a interface. A interação entre camadas se dá por meio da troca de mensagens, as quais trafegam através da interface de comunicação.
- Camadas: têm funções autônomas, não dependendo das outras da aplicação. Elas se comunicam somente com as camadas imediatamente acima ou abaixo.

Elementos da arquitetura em camadas

- Interfaces: Têm formato padronizado para a condução das mensagens entre duas camadas. Componente sintático e semântico. A comunicação através da interface se dá a nível de componente.
- Mensagens: Podem ser geradas dentro do sistema ou externamente (eventos). Geralmente, trafegam em pares (solicitação-resposta).

Benefícios da arquitetura em camadas

- Modularidade: facilita a manutenção.
- Independência tecnológica: a adoção de novas tecnologias de software e hardware pode ser feita em uma camada sem afetar as outras.
- Estabilidade: Interfaces padronizadas proporcionam uniformidade e estabilidade à aplicação.
- Reutilização: Uma camada com interface bem documentada poderá ser usada novamente em outra aplicação
- Distribuição: Facilita a distribuição de dados e processamento.

Voltar