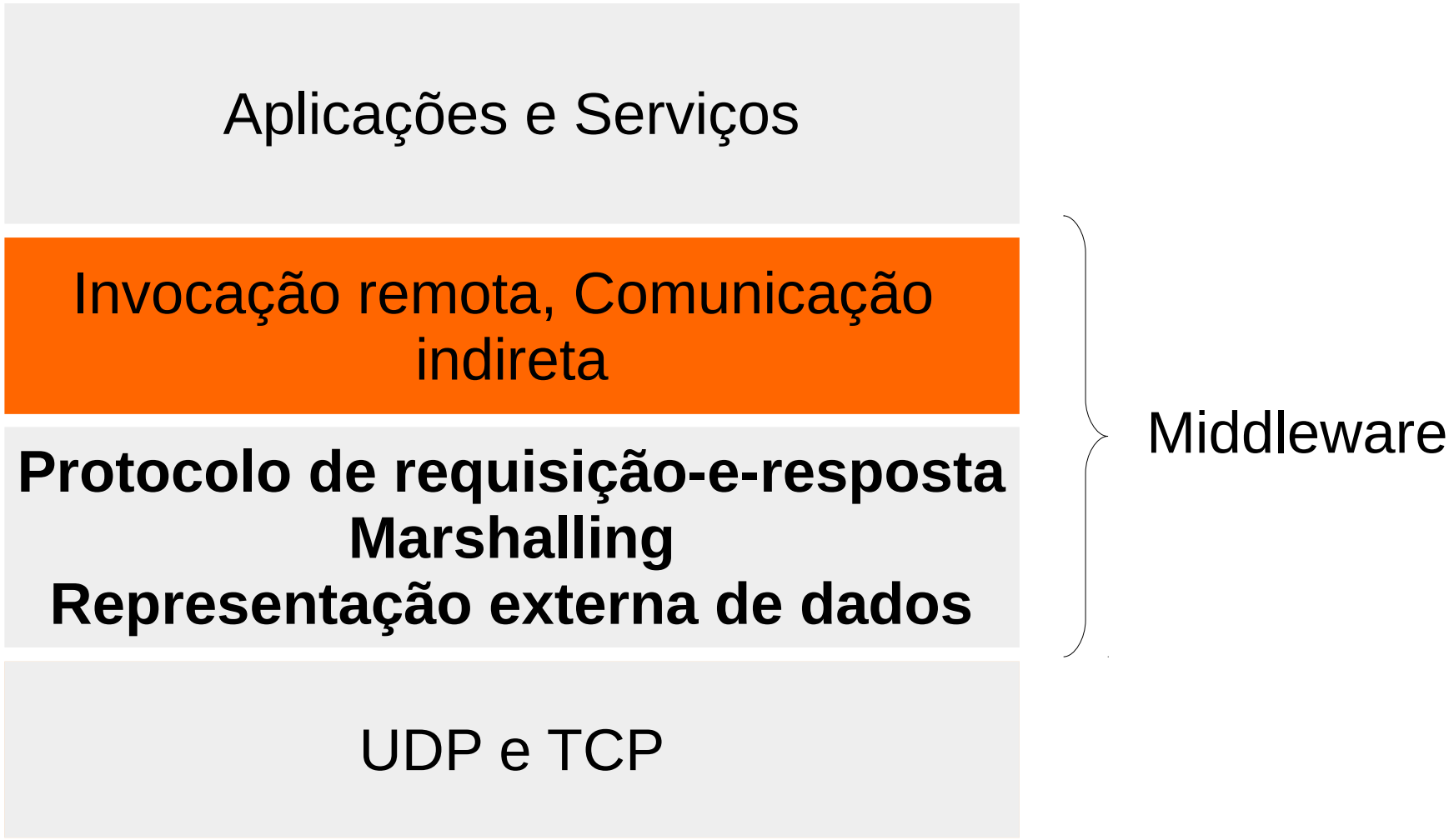
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

Sistemas Distribuídos

Prof. Sergio T. Carvaho
sergio@inf.ufg.br

Invocação Remota, Objetos Distribuídos

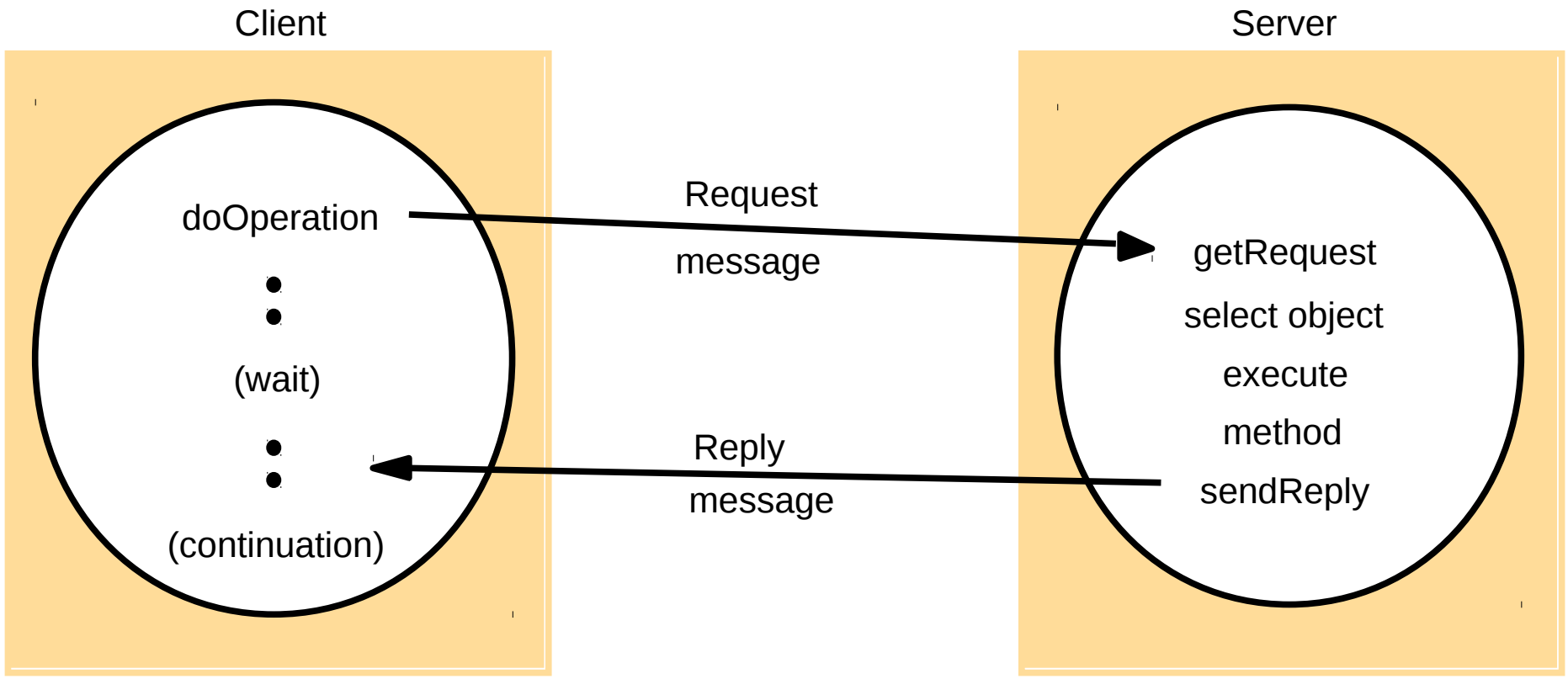
Camadas do middleware



Comunicação Cliente-Servidor

- Modelo geral: requisição-e-resposta
- Variantes:
  - síncrona: cliente bloqueia até receber a resposta
  - assíncrona: cliente recupera (explicitamente) a resposta em um instante posterior
    - não-bloqueante
- Implementação sobre protocolo baseado em datagramas é mais eficiente
  - evita: reconhecimentos redundantes, mensagens de estabelecimento de conexão, controle de fluxo

Protocolo requisição-e-resposta



Operações utilizadas para implementar o protocolo de requisição-e-resposta

public byte[] doOperation (RemoteObjectRef o, int methodId, byte[] arguments)

envia uma mensagem de requisição para o objeto remoto e retorna a resposta. Os argumentos especificam o objeto remoto, o método a ser chamada e os argumentos para aquele método.

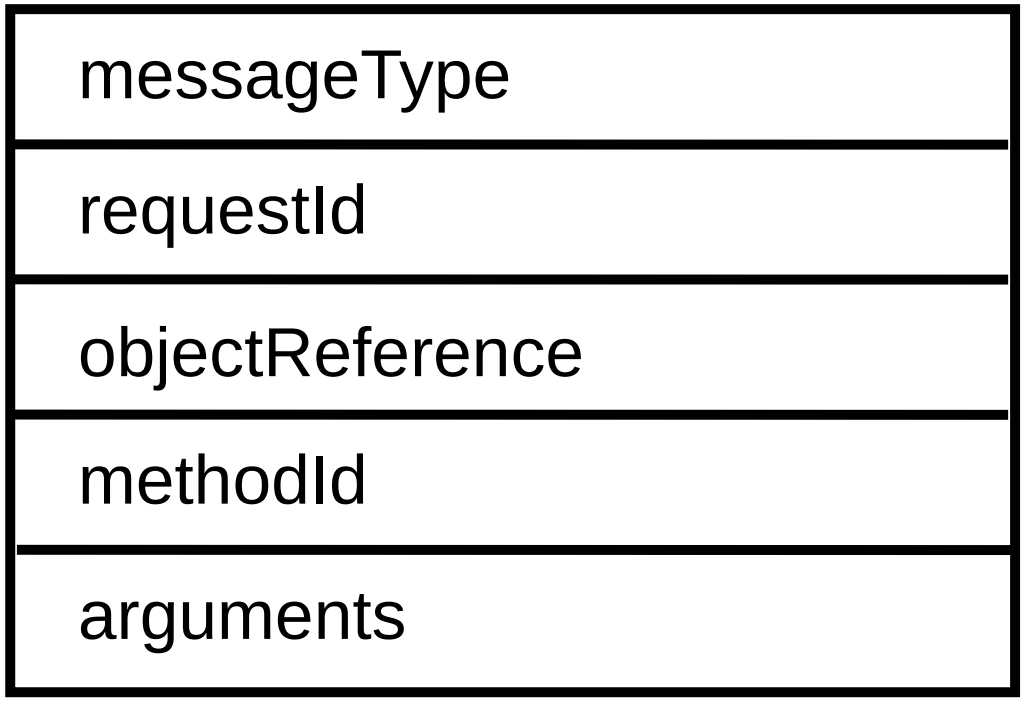
public byte[] getRequest ();

obtém uma requisição de um cliente através de uma porta servidora.

public void sendReply (byte[] reply, InetAddress clientHost, int clientPort);

envia a mensagem de resposta para o cliente, endereçando-a a seu endereço IP e porta.

Estrutura de mensagens de requisição-e-resposta



int (0=Request, 1= Reply)

int

RemoteObjectRef

int or Method

array of bytes

Identificador da mensagem

request ID + ID do processo que fez a requisição

Modelo de falhas

- Suposições
  - processos: param após falha
  - canais: podem omitir mensagens
- Timeouts: em *doOperation*
  - detectar que a resposta (ou a requisição) foi perdida
- alternativas de tratamento:
  - sinalizar a falha para o cliente (uma boa opção?)
  - repetir o envio da requisição (um certo número de vezes)
    - até ter certeza sobre a natureza da falha (quem falhou? O canal ou o processo?)

Modelo de falhas (2)

- Descarte de mensagens duplicadas
  - o servidor deve distinguir requisições novas de requisições retransmitidas
    - apropriado re-executar a requisição?
- Em caso de perda da resposta
  - servidor re-executa a requisição para gerar novamente a resposta?
  - servidor mantém um histórico das respostas geradas para o caso de precisar retransmitir?
    - o histórico precisa conter apenas a última resposta enviada para cada cliente. Por que? Suficiente?

Protocolos de troca de mensagens em RPC

Nome	*Mensagens enviadas pelo*		
	Cliente	***Servidor***	***Cliente***
R	*Request*		
RR	*Request*	*Reply*	
RRA	*Request*	*Reply*	*Acknowledge reply*

- Tipo de garantia (confiabilidade) em cada caso?
- Que tipo de aplicação em cada caso?

Protocolo de transporte utilizado: TCP

- não impõe limites no tamanho das mensagens
  - na verdade, cuida da segmentação de forma transparente
  - permite argumentos de tamanho arbitrário (ex.: listas)
- transferência confiável
  - lida com mensagens perdidas e duplicadas
- controle de fluxo
- overhead reduzido em sessões longas (muitas requisições e respostas sucessivas)

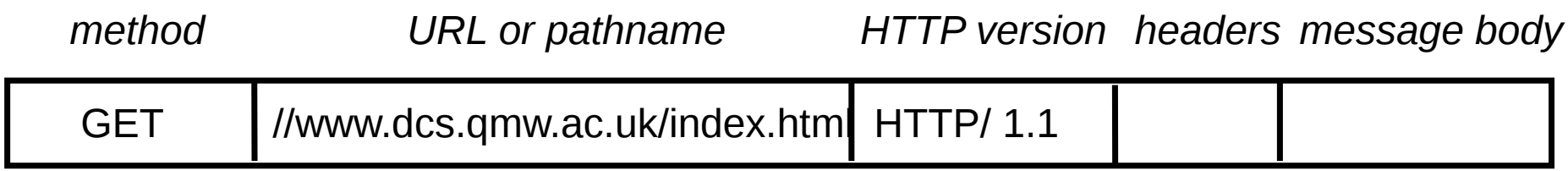
Protocolo de transporte utilizado: UDP

- se mensagens têm tamanho fixo
- se sessões são curtas (apenas uma requisição e sua resposta)
  - não compensa o overhead do handshake TCP
- se operações são idempotentes
  - mensagens duplicadas (retransmissões) podem ser re-processadas sem problema
- otimização do protocolo de confiabilidade para casos específicos

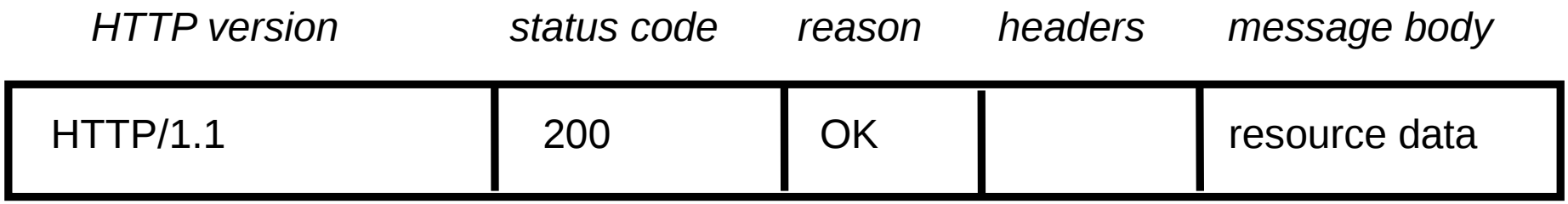
HTTP como protocolo de requisição-e-resposta

- Tipo RR
- Regras para o formato de mensagens
  - Marshalling
- Conjunto fixo de métodos: PUT, GET, POST,...
- Permite negociar o tipo do conteúdo (MIME)
- Autenticação (senhas, credenciais, desafios)
- Conexões persistentes para amortizar o custo de abrir e fechar conexões TCP

Mensagem de requisição HTTP



Mensagem de resposta HTTP



RPC

- Procedimentos remotos chamados como procedimentos locais
- Questões de projeto
  - Programação com interfaces
  - Semântica da chamada
  - Transparência

Interfaces

- Proveem acesso às características externamente visíveis de um objeto ou módulo
  - em geral: métodos e variáveis
- Papel fundamental no encapsulamento
- Em sistemas distribuídos: apenas os métodos são acessíveis através de interfaces
- Passagem de parâmetros: *copy-restore*
  - parâmetros de entrada e saída
  - ponteiros não são permitidos
  - objetos como parâmetros: referência de objeto

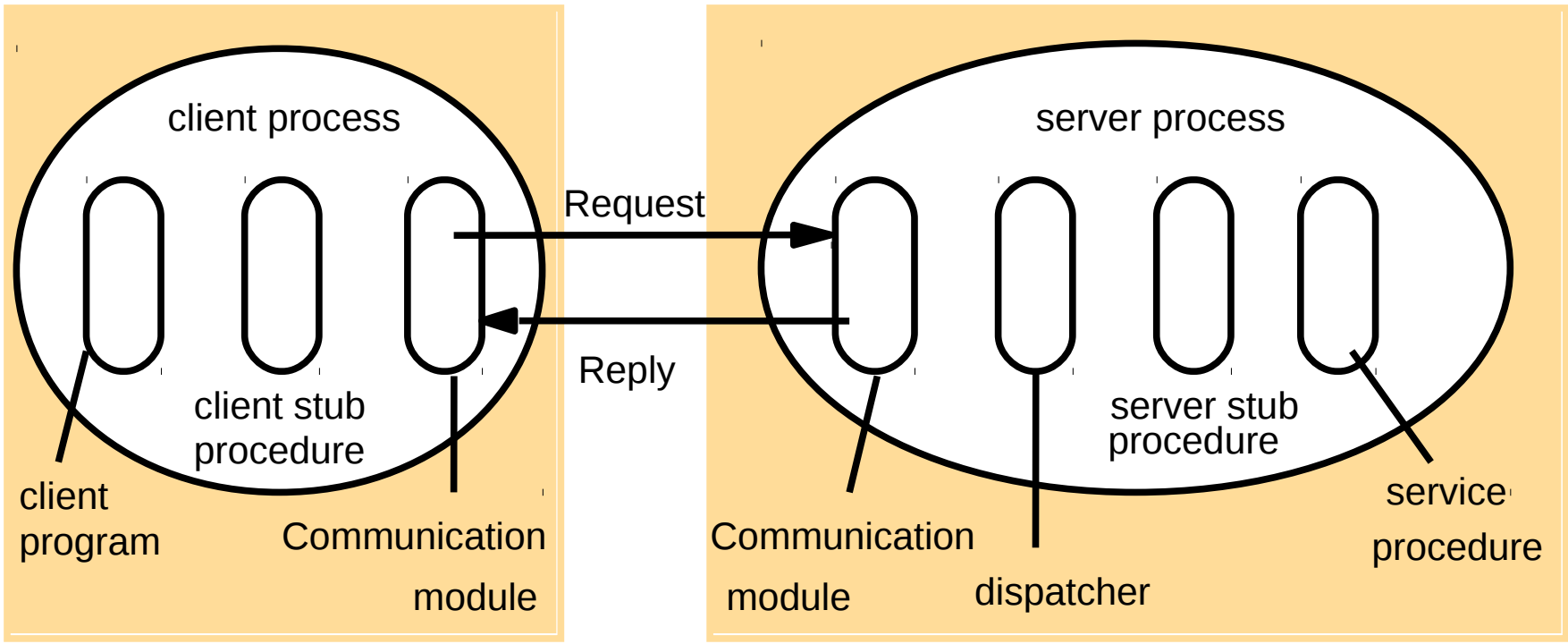
Linguagens de Definição de Interfaces (IDL)

- Sintaxe (e semântica associada) para a definição de:
  - operações: nome, parâmetros e valor de retorno
  - exceções
  - atributos
  - tipos primitivos e construídos (para os parâmetros e valores de retorno)
- Exemplos:
  - CORBA IDL
  - DCOM IDL (Microsoft IDL)

Exemplo de definição em CORBA IDL

```
// In file Person.idl
struct Person {
        string name;
        string place;
        long year;
} ;
interface PersonList {
        readonly attribute string listname;
        void addPerson(in Person p) ;
        void getPerson(in string name, out Person p);
        long number();
};
```

Implementação de RPC



Questões de projeto para RPC

- Semântica de chamadas
  - talvez executada (*maybe, best-effort)*
  - executada pelo menos uma vez (*at least once)*
  - executada no máximo uma vez (*at most once)*
- Transparência
  - ideal, mas não 100% prática
    - falhas parciais
    - latência
  - sintaxe transparente, mas semântica explicitamente distinta

Semântica de chamadas em RPC

Fault tolerance measures			***Invocation semantics***
Retransmit request message	*Duplicate filtering*	*Re-execute procedure or retransmit reply*	
No	Not applicable	Not applicable	***Maybe***
Yes	No	Re-execute procedure	***At-least-once***
Yes	Yes	Retransmit reply	***At-most-once***

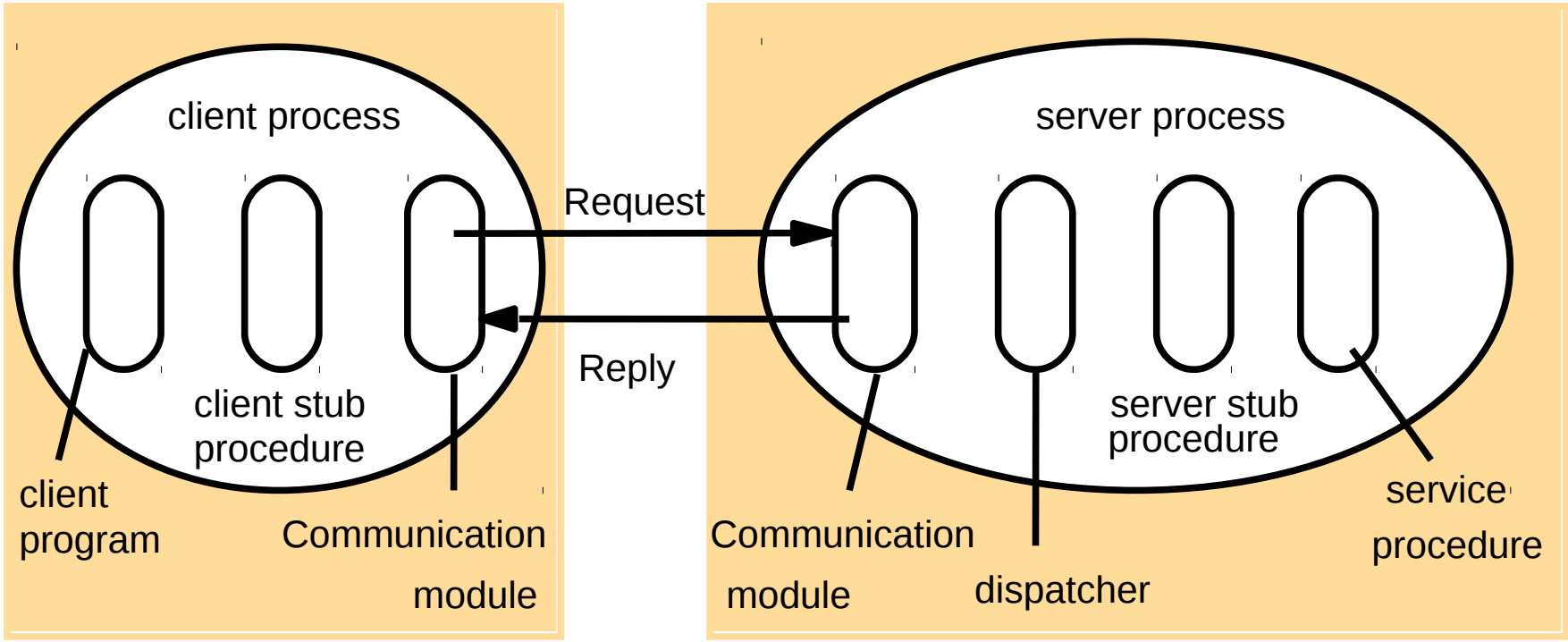
Falhas de omissão e falhas arbitrárias

Class of failure	*Affects*	*Description*
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a *send,* but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.

Semântica de Chamadas em RPC

- talvez executada (*maybe, best-effort)*
  - Nenhuma tolerância a falhas
  - Tipos de falhas: omissão e *crash*
- executada pelo menos uma vez (*at least once)*
  - Mascara falhas de omissão
  - Tipos de falhas: *crash* e falhas arbitrárias
  - Aceitável para operações idempotentes
- executada no máximo uma vez (*at most once)*
  - Mascara falhas de omissão
  - Evita falhas arbitrárias

Implementação de RPC



Qual a diferença principal em relação ao RMI?

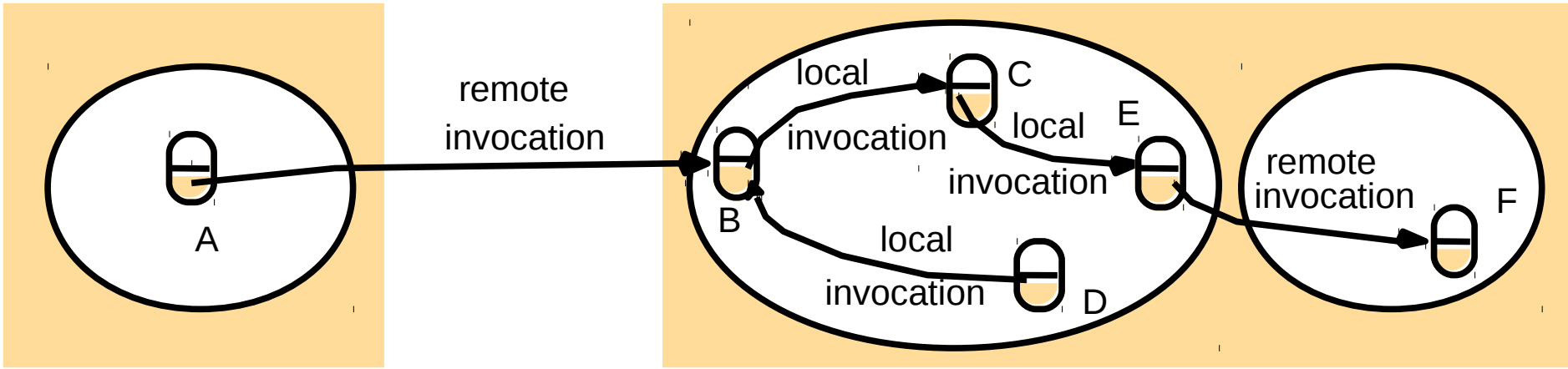
RMI

- Invocação remota de métodos
- Questões de projeto
  - Modelo de objetos básico
  - Conceitos de objetos distribuídos
  - Extensão do modelo de objetos convencional

Modelo de objetos básico

- Referências de objetos
- Interfaces
- Ações
- Exceções
- Coleta de lixo

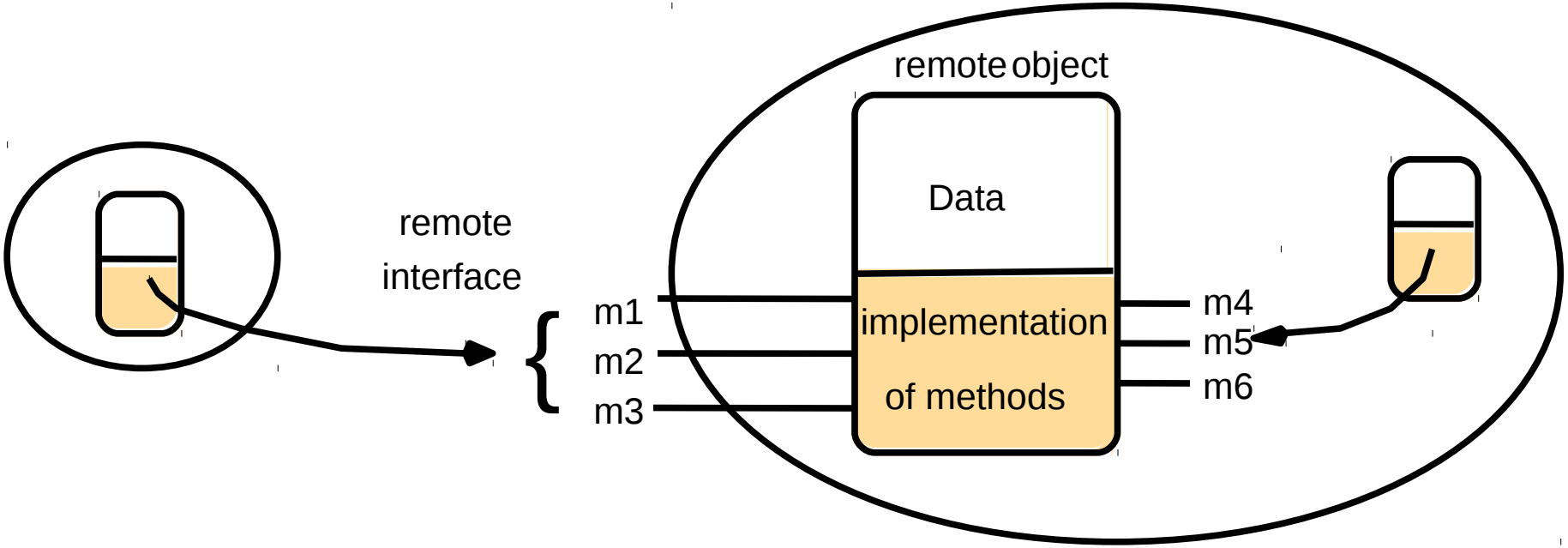
Chamadas locais e remotas



O modelo de objetos distribuídos: uma extensão ao modelo de objetos básico

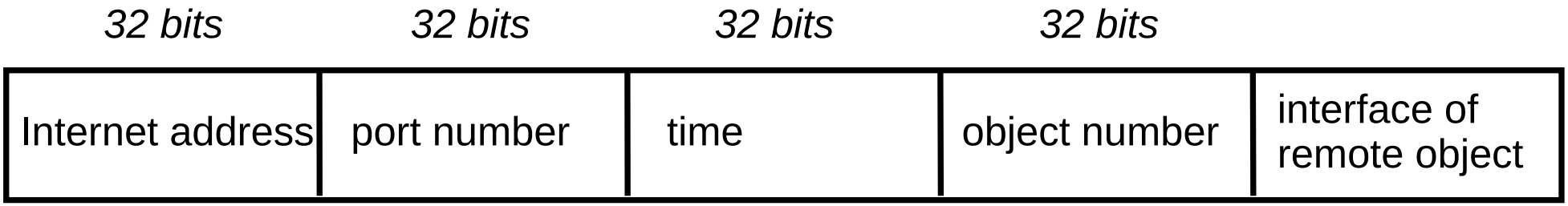
- Objetos remotos vs. objetos locais
- Chamadas de métodos remotos (RMI)
- Referência de objeto remoto
  - funcionalmente semelhante a referências locais
  - estruturalmente diferente: identificador válido em todo o sistema distribuído
- Interface remota
  - define os métodos remotamente acessíveis
  - geralmente independente da linguagem de implementação

Um objeto com interfaces local e remota



Representação de Referências de Objetos

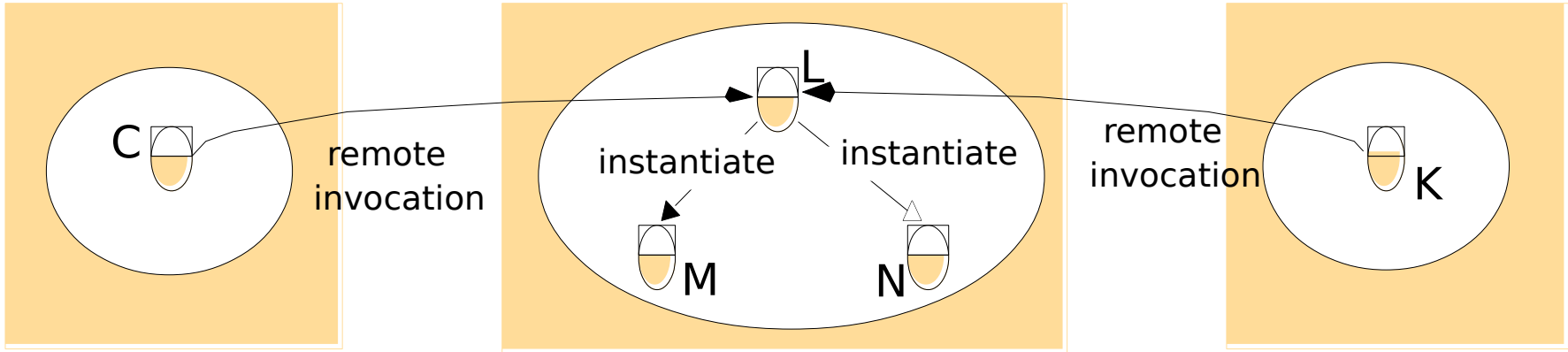
- Necessária quando objetos remotos são passados como parâmetro
- A referência é serializada (não o objeto)
- Contém toda informação necessária para identificar (e endereçar) um objeto unicamente no sistema distribuído



O modelo de objetos distribuídos: uma extensão ao modelo de objetos básico (2)

- Exemplos de ações:
  - requisição para executar alguma operação em um objeto remoto
  - obtenção de referências de objetos remotos
  - instanciação de objetos remotos (através de fábricas)
- Coleta de lixo distribuída
  - requer contagem de referências explícita
  - rastreamento de todas as referências trocadas
  - pouco eficiente ou difícil de ser implementada

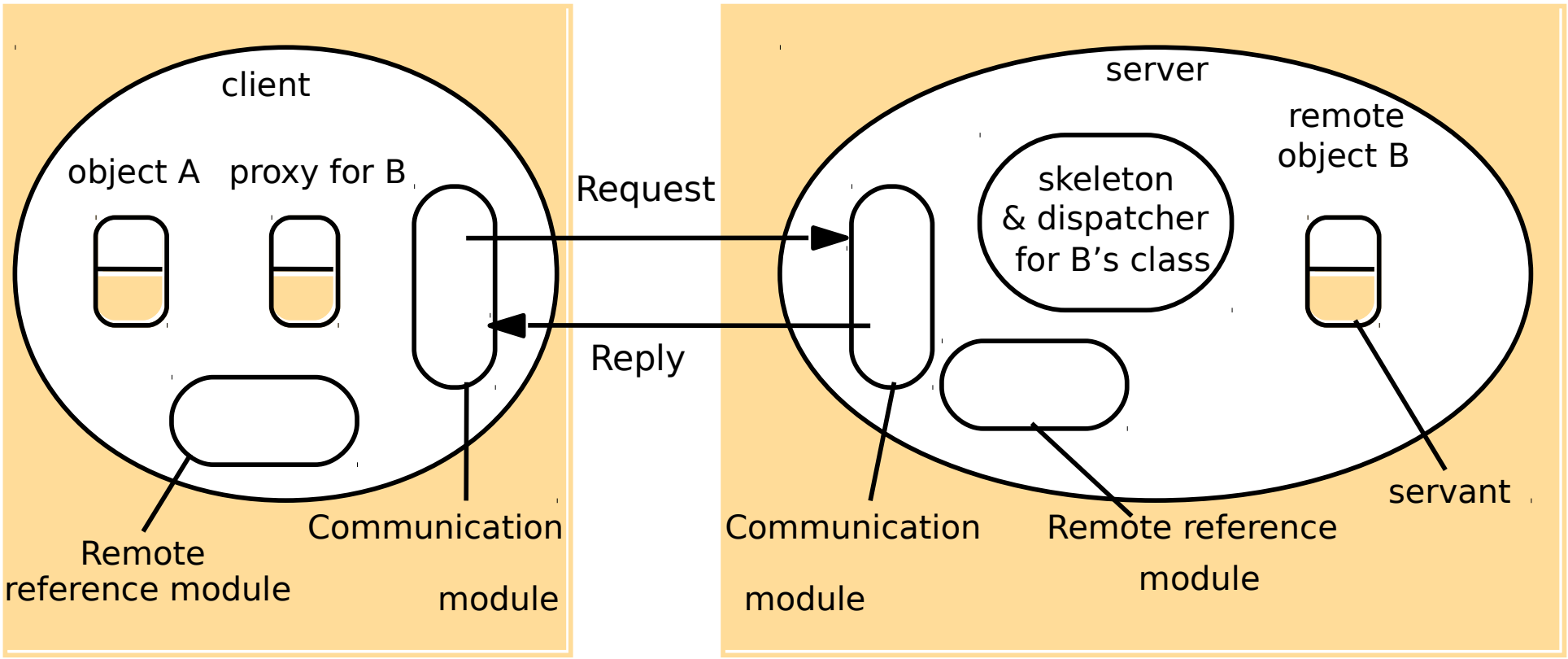
Instanciação de objetos remotos



O modelo de objetos distribuídos: uma extensão ao modelo de objetos básico (3)

- Exceções
  - erros de aplicação: gerados pela lógica do servidor
  - erros de sistema: gerados pelo middleware
  - conduzidos de volta ao cliente sob a forma de mensagens

Implementação de RMI



Módulo de comunicação

- Cuida do protocolo de comunicação de mensagens entre cliente e servidor
- Implementa o protocolo requisição-e-resposta
- Define os tipos de mensagens utilizados
- Provê a identificação das requisições
- Semântica de chamadas (ex.: *at-most-once*)
- Retransmissão e eliminação de duplicatas

Módulo de referências remotas

- Gerencia referências de objetos remotos
- No lado servidor:
  - tabela com as referências a objetos remotos que residem no processo local
  - ponteiro para o skeleton correspondente
- No lado cliente:
  - tabela com as referências a objetos remotos que residem em outros processos e que são utilizadas por clientes locais
  - ponteiro para o proxy local para o objeto remoto

Serventes

- Instância de uma implementação de objeto
- Implementação de objeto = classe
- Hospedados em *processos servidores*

Proxy (ou *Stub*)

- Objeto local que representa o objeto remoto para o cliente
- “Implementa” os métodos definidos na interface do objeto remoto
- Cada implementação de método no *proxy*:
  - *marshalling* de requisições
  - *unmarshalling* de respostas
- Torna a localização e o acesso ao objeto remoto transparentes para o cliente

Despachante

- Recebe requisições do módulo de comunicações e as repassa para o *skeleton*
- Duas variantes:
  - Despachantes específicos gerados para cada classe de objetos remotos
    - utiliza o ID do método para chamar o *skeleton*
  - Despachante genérico
    - utiliza o ID do objeto para chamar um método genérico do *skeleton* associado ao objeto alvo da requisição
    - o próprio *skeleton* se encarrega de chamar o método-alvo
    - abordagem do adaptador de objetos de CORBA

Skeleton

- Implementa os mecanismos de tratamento de requisições recebidas e repasse das mesmas (na forma de chamadas locais) ao objeto alvo:
  - *Unmarshalling* de requisições
  - *Marshalling* de respostas
- Co-localizado com o objeto remoto
- Específico para cada tipo de objeto remoto

Compilador de IDL

- Geração de *proxy* e *skeleton* (e despachante) a partir de uma definição de interface
- Segue o mapeamento da IDL para a linguagem de implementação do cliente e do servidor

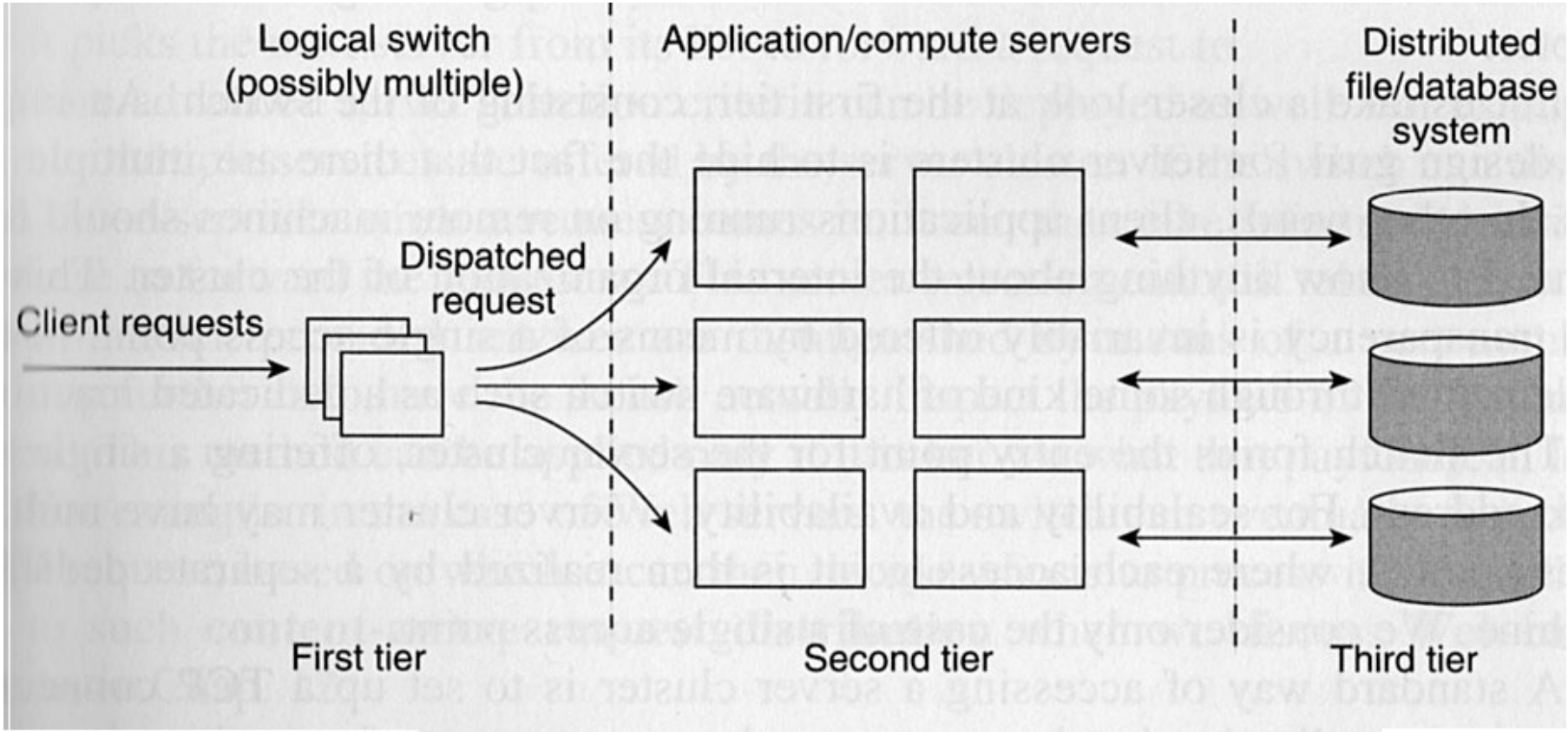
Chamadas dinâmicas

- *Proxy* dinâmico
  - genérico, independente de interface
  - utiliza uma definição de interface acessível dinamicamente para formatar as requisições
    - repositório de interfaces
  - útil quando não se conhece a interface em tempo de programação
- *Skeleton* dinâmico
  - permite receber chamadas destinadas a “qualquer” tipo de objeto/interface
  - útil na construção de servidores “genéricos”

Programas: Servidor e Cliente

- Servidor
  - código para instanciação das implementações de objetos (instância da impl. de objeto = servente)
  - dá origem ao processo servidor que hospeda os objetos remotos
- Cliente
  - código que faz uso de objetos remotos
  - não necessariamente composto de objetos

O Paradigma "Servidor Concorrente"



The general organization of a three-tiered server cluster.

Serviços de suporte

- *Binder*
  - mapeia nomes para referências de objetos

  Ex.: serviço de nomes de CORBA, *Registry* de RMI
- Serviço de localização
  - mapeia referências de objetos para as respectivas (prováveis) localizações físicas dos objetos
  - provê suporte para migração de objetos e redirecionamento
  - elimina a necessidade de guardar endereço IP e porta na referência de objeto!

Serviços de suporte (2)

- *Threads*
  - cada requisição é servida com o uso de uma *thread* separada no processo servidor
  - evita o bloqueio do servidor e permite que o mesmo processe várias requisições "simultaneamente"
- Ativação de objetos
  - objetos podem ser salvos em disco quando não utilizados (estado serializado + metadados)
    - economia de recursos (ex.: memória)
  - ao ser necessário, um objeto pode ser reativado

um novo servente é criado para "materializar" o objeto

Serviços de suporte (3)

Armazenamento persistente de objetos

- serialização do estado dos objetos
- gerenciamento do armazenamento dos objetos
- estratégia para definir
  - quando um objeto deve ser desativado (ir para o estado “passivo”)
  - quais partes do estado do objeto devem ser salvas
- o uso de persistência deve ser transparente para o implementador do objeto e para os clientes

Coleta de lixo distribuída

Ideia geral:

- Um objeto existe enquanto houver alguma referência para ele no sistema distribuído
- Quando a última referência for removida, o objeto pode ser destruído

Mecanismo:

- Interceptação de referências de objetos remotos passadas como parâmetros ou retorno de RMI
- Contagem de referências (no processo servidor)
- Contador incrementado/decrementado como resultado da criação/destruição de *proxies* nos clientes

Créditos

Prof. Sérgio T. Carvalho

sergio@inf.ufg.br