RPCs e Sockets

```
// quase Java....
class HelloWorld {
    // método local
    public void sayHello() {
        print("Hello World!");
    }
    // programa principal
    public static void main(String[] args) {
        new HelloWorld().sayHello();
    }
}
```

```
public class HelloWorld {
  public HelloWorld ( String name ) {
    Naming.rebind( name, this );
  }
  // método remoto
  public String sayHello () {
    return "Hello World!";
  }
}
public class HelloWorldServer {
  public static void main(String[] args) {
    HelloWorld object = new HelloWorld( "Hello" );
  }
}
```

```
public class HelloWorldClient {

  public static void main(String[] args) {
      // conexão
      HelloWorld hello_server = (HelloWorld)Naming.lookup("rmi://hostB/Hello");
      // chamada remota
      print( hello_server.sayHello() );
  }
}
```

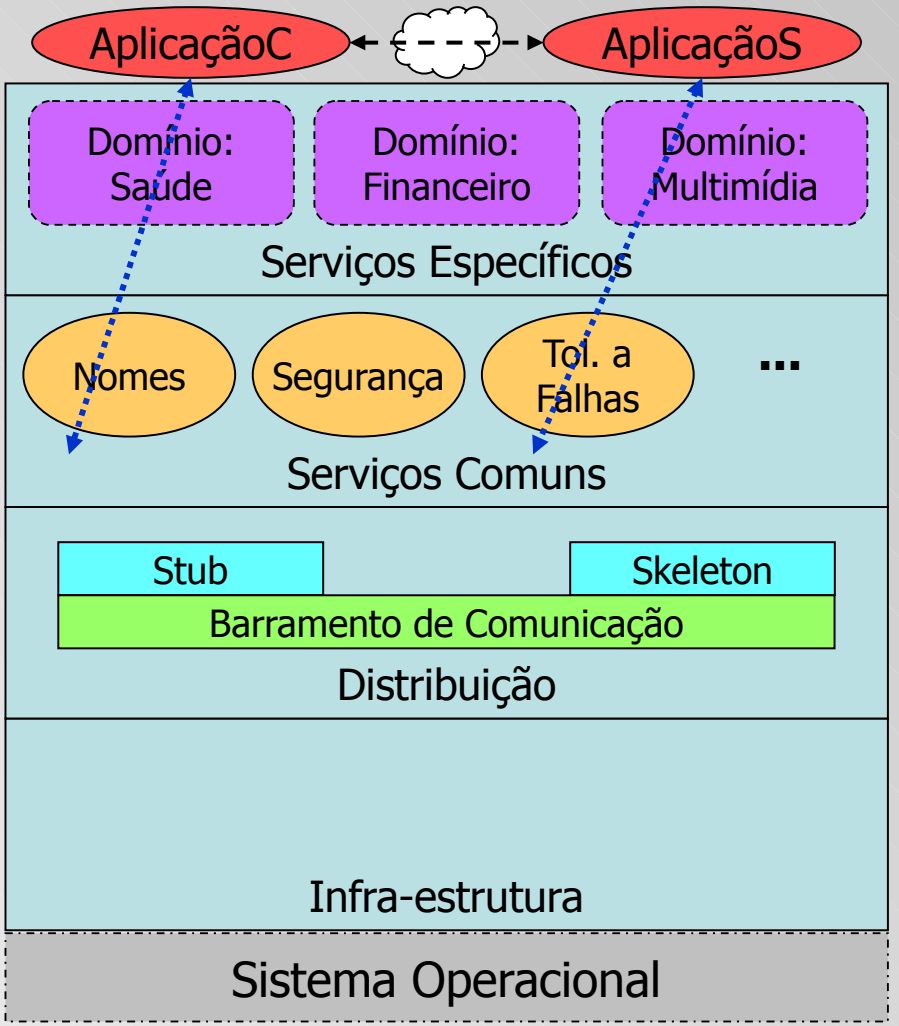
Infra-Estrutura de SW para SDs

Middleware: Arquitetura Básica em Camadas

Hoje!!!

modelos de programação, onde a comunicação é abstraída, por ex. – ORB CORBA, incluindo RPC

abstrai as peculiaridades dos sistemas operacionais, encapsulando e melhorando os mecanismos de concorrência, por ex. – ex. JVM



Tópicos

- Elementos básicos de comunicação
  - Sockets

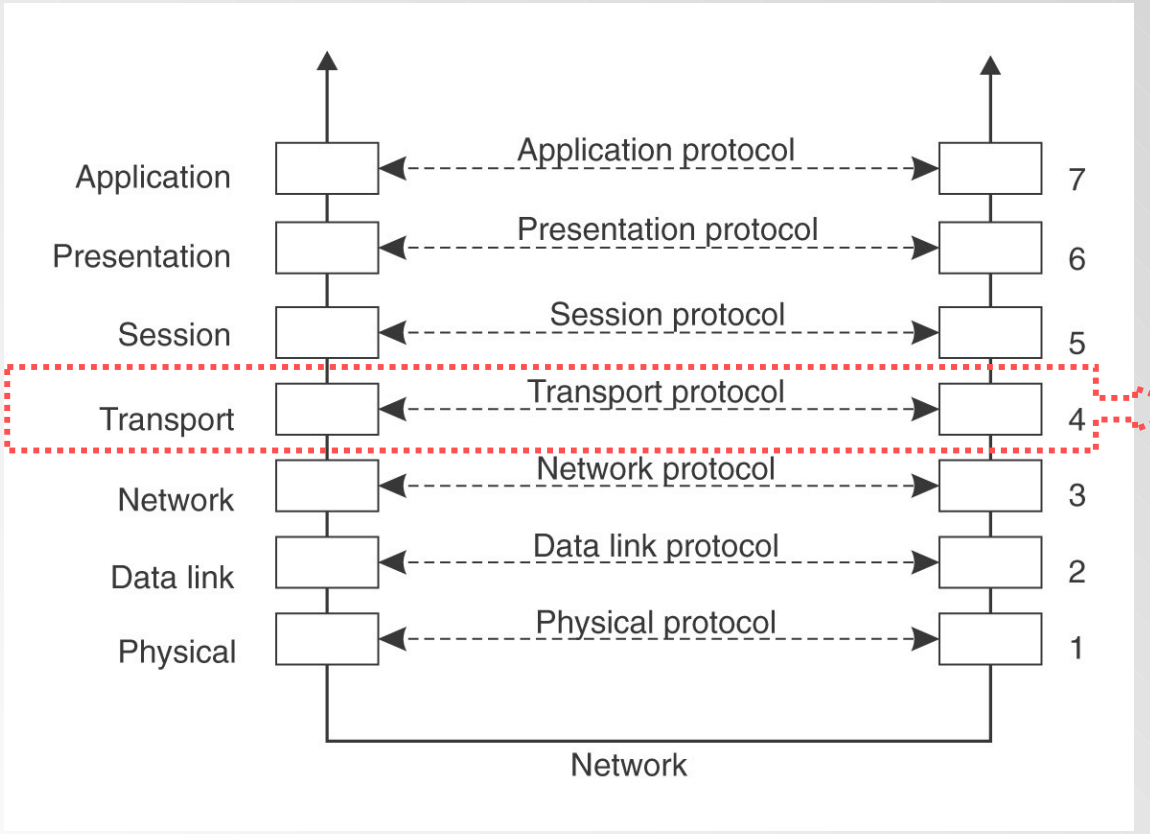
- RPC: *Remote Procedure Call*
  - Conceito geral → no UNIX é homônimo
  - Para Java, usa-se RMI (*Remote Method Invocation*)

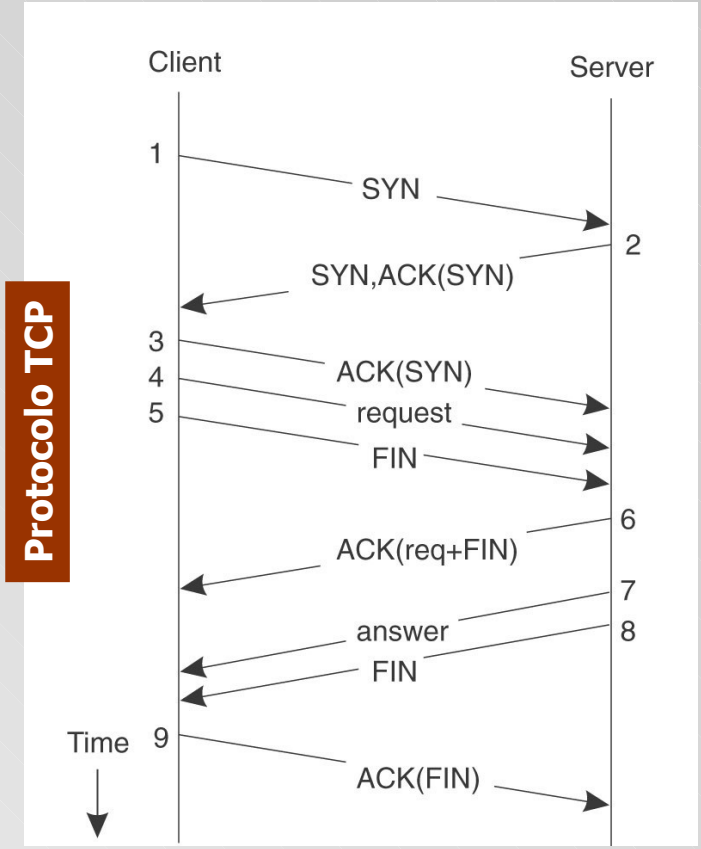
Protocolos

- Ausência de memória compartilhada => **troca de mensagens**
  - Para comunicação entre processos
- Depois executa um *system call*
  - SO envia a mensagem pela rede para B
- A e B têm que concordar sobre um **conjunto de regras** – **protocolo**
- Necessários em vários níveis. Exs.:
  - transmissão de bits
  - detalhes de alto nível sobre como a informação é expressa

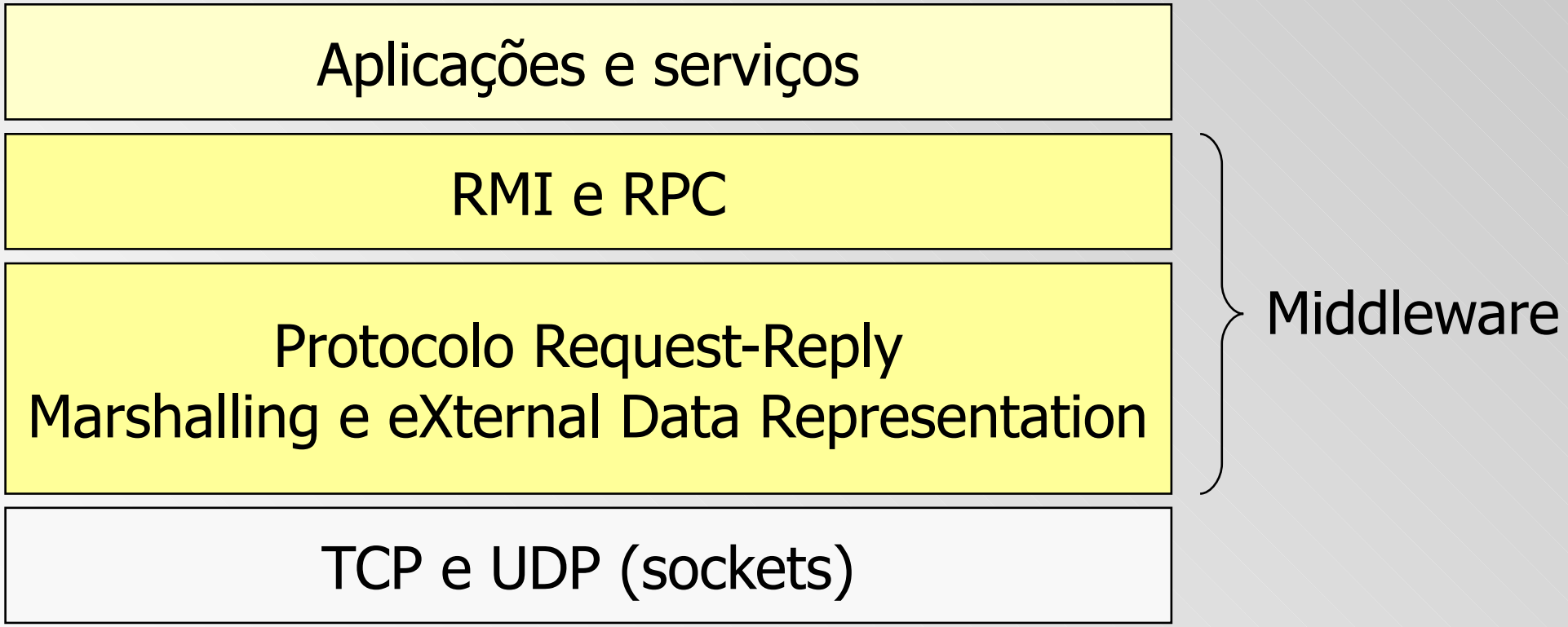
O Conceito de Protocolo

- Conjunto de operações e regras para que duas entidades possam se comunicar





Camadas



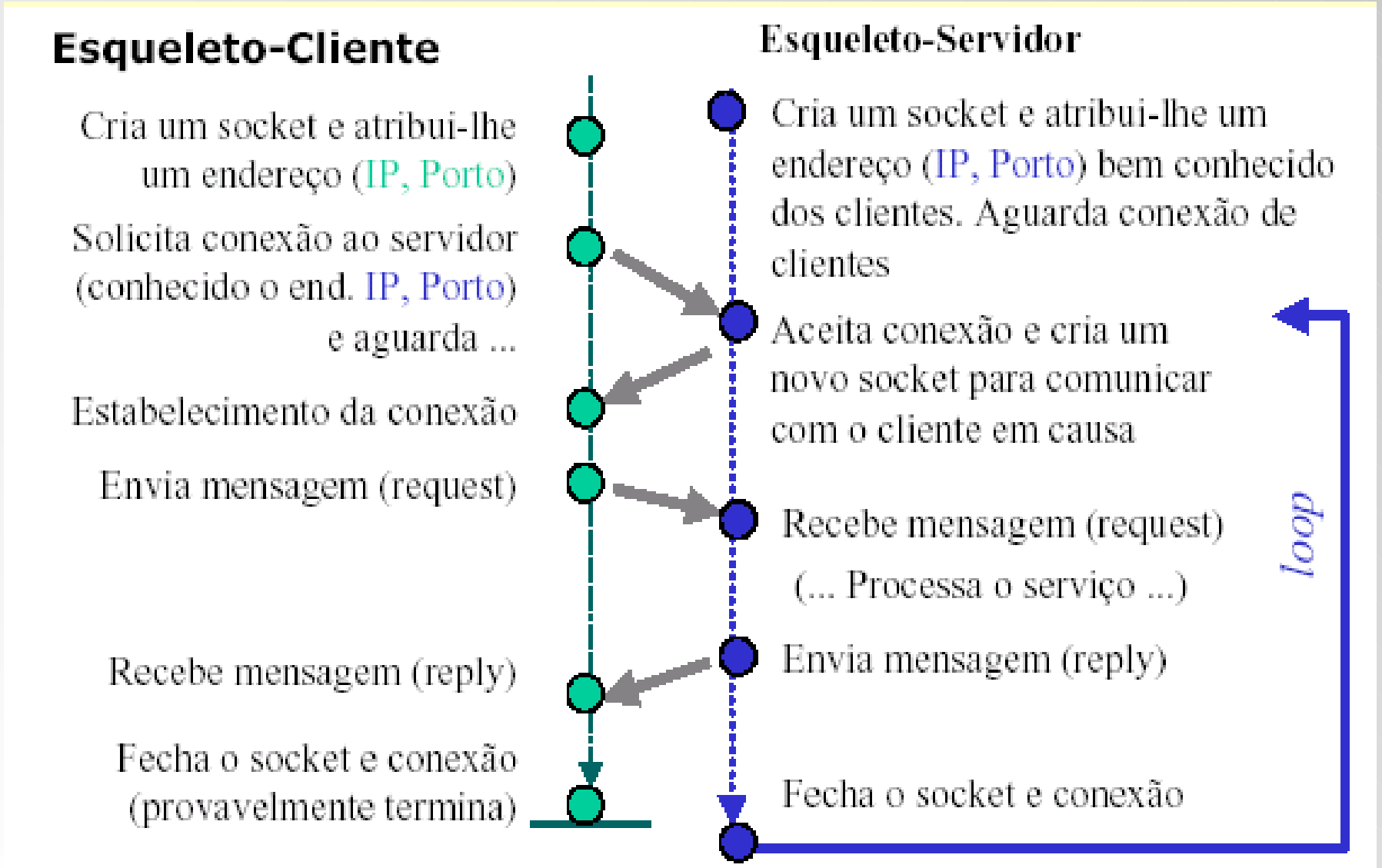
Sockets

Socket

- Uma porta de um canal de comunicação
- Permite a um processo executando num computador enviar/receber mensagens para/de outro processo
  - Através de uma API bem conhecida

- Usado para representar uma conexão entre um cliente e um servidor

Descrição do fluxo (sockets)



Transmissão de dados

- Dados em programas são estruturados
  - Enquanto isso, mensagens carregam informação sequencial:

    Linearização/Restauração de dados

- Heterogeneidade na representação de dados em computadores
  - Uso de um formato externo comum
  - Inclusão de uma identificação de arquitetura na mensagem

Marshalling/Unmarshalling

- *Marshalling*:
  - Linearização de uma coleção de itens de dados estruturados
  - Tradução dos dados em formato externo (ex: XDR – eXternal Data Representation)

- *Unmarshalling*:
  - Tradução do formato externo para o local
  - Restauração dos itens de dados de acordo com sua estrutura

Remote Procedure Calls (RPCs)

Protocolo Pedido-Resposta

- Request-Reply (RR)
- Típico na interação cliente-servidor
- Primitivas:
  - DoOperation - clientes invocam operações remotas
  - GetRequest - servidor adquire os pedidos de serviços
  - SendReply

Chamadas de Procedimentos Remotos

- Ideal: programar um sistema distribuído como se fosse centralizado

- RPC objetiva permitir chamada de procedimento remoto como se fosse local
  - ocultando entrada/saída de mensagens
  - uso análogo ao de um procedimento local

RPCs: considerações em função da distribuição

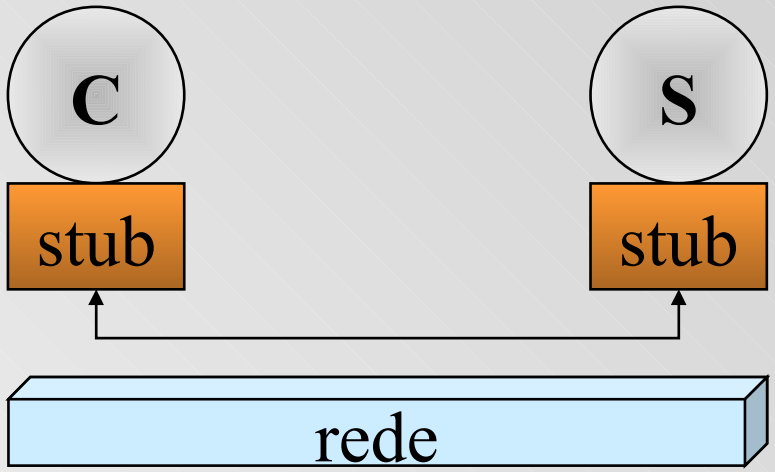
- Evitar passagem de endereço e variáveis globais
  - Sem espaço de endereçamento compartilhado
- Novos tipos de erros
  - Falhas de nós ou da rede
  - Atrasos
- Concorrência não é tanto um problema
  - Por quê?

RPC: processamento das interfaces

- **Objetivo**: integração dos mecanismos de RPC com os programas cliente e servidor escritos em uma linguagem de programação convencional

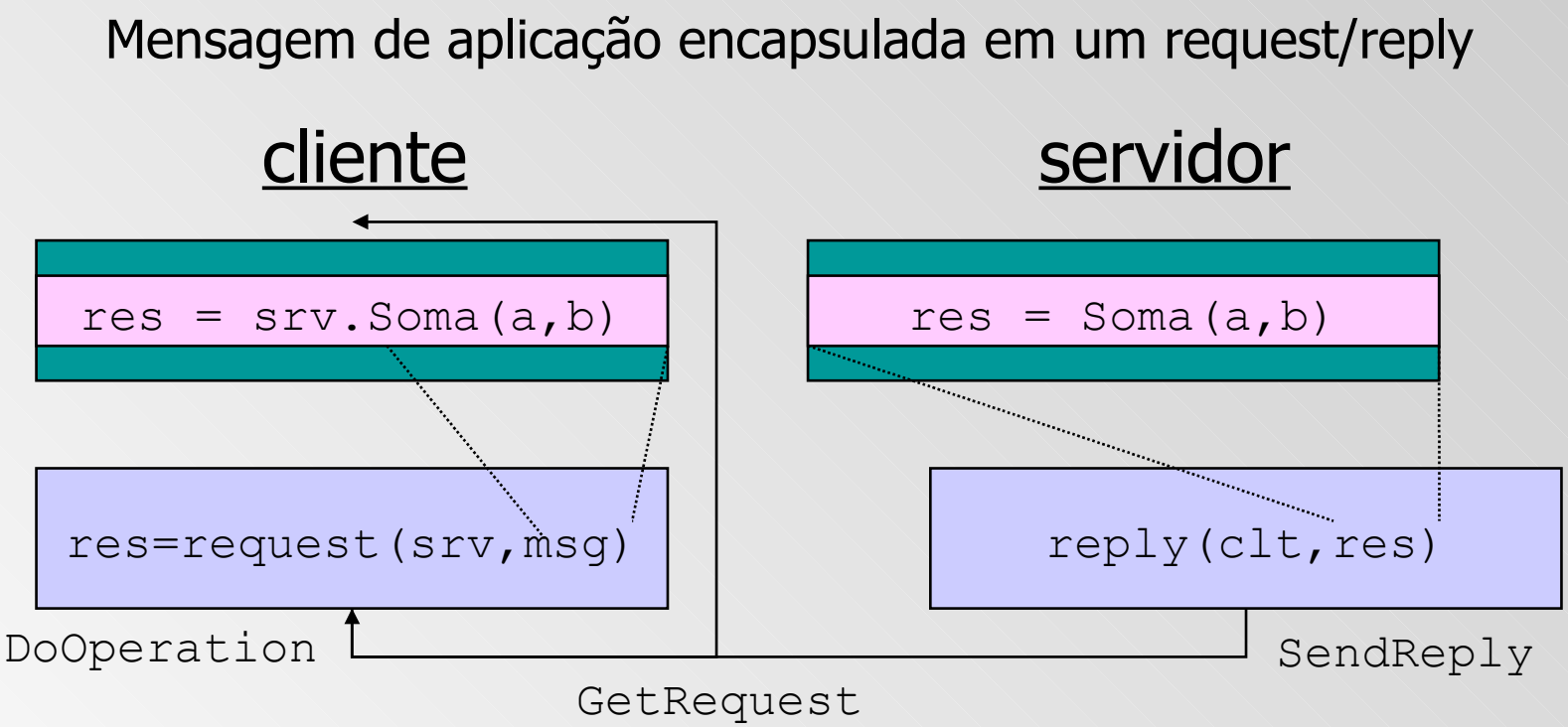
RPC: processamento de interface

- ***Stubs*** (no cliente e no servidor): **transparência de acesso** – forma de chamada remota semelhante à chamada local
  - tratamento de algumas exceções no local
  - *marshalling*
  - *unmarshalling*



RPC: tratamento da comunicação

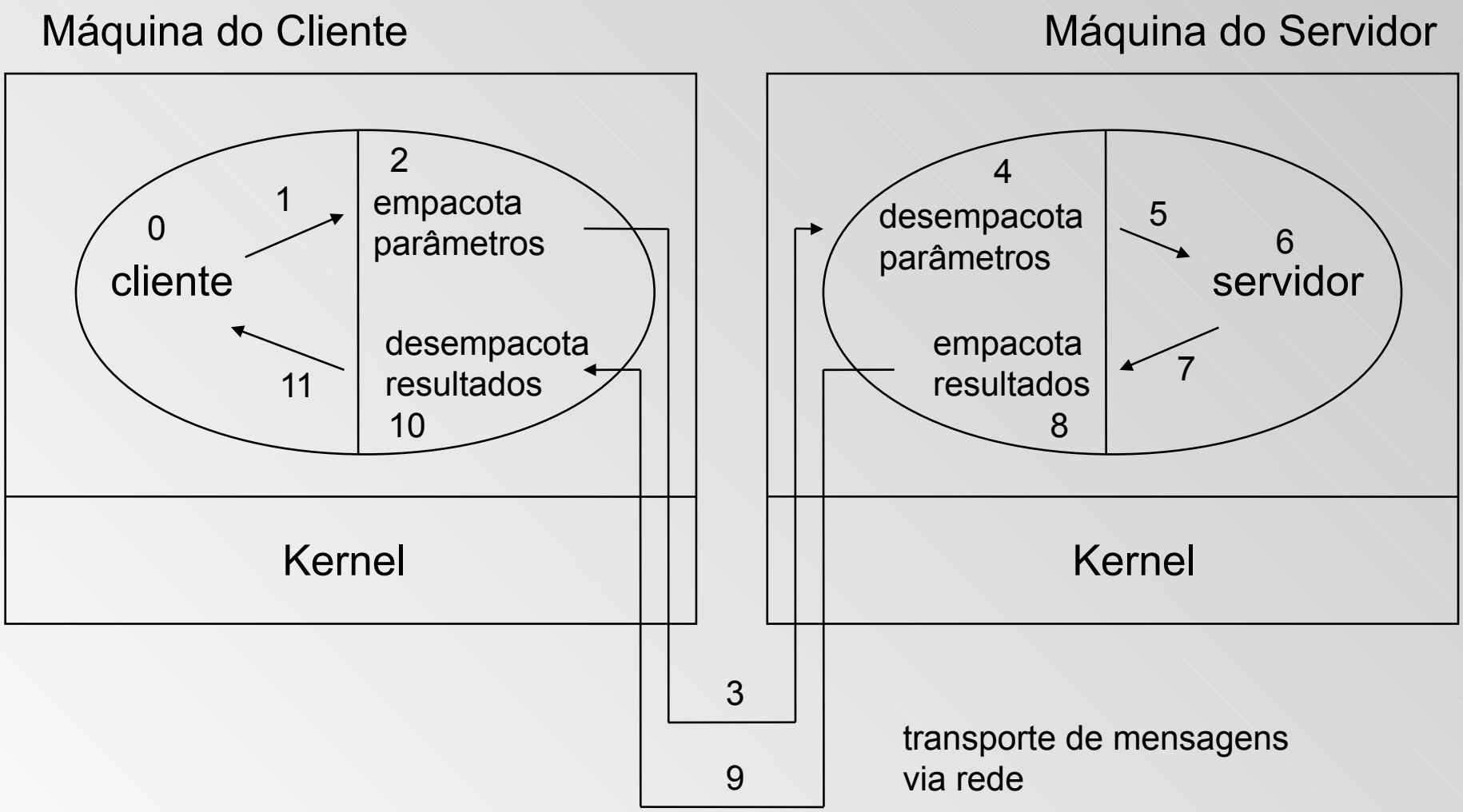
- Módulo de comunicação usa protocolo *pedido-resposta* para troca de mensagens entre cliente e servidor



RPC: ligação

- O mecanismo possui um *binder* para resolução de nomes, permitindo
  - Ligação dinâmica
    - Transparência de localização

Chamadas e mensagens em RPC



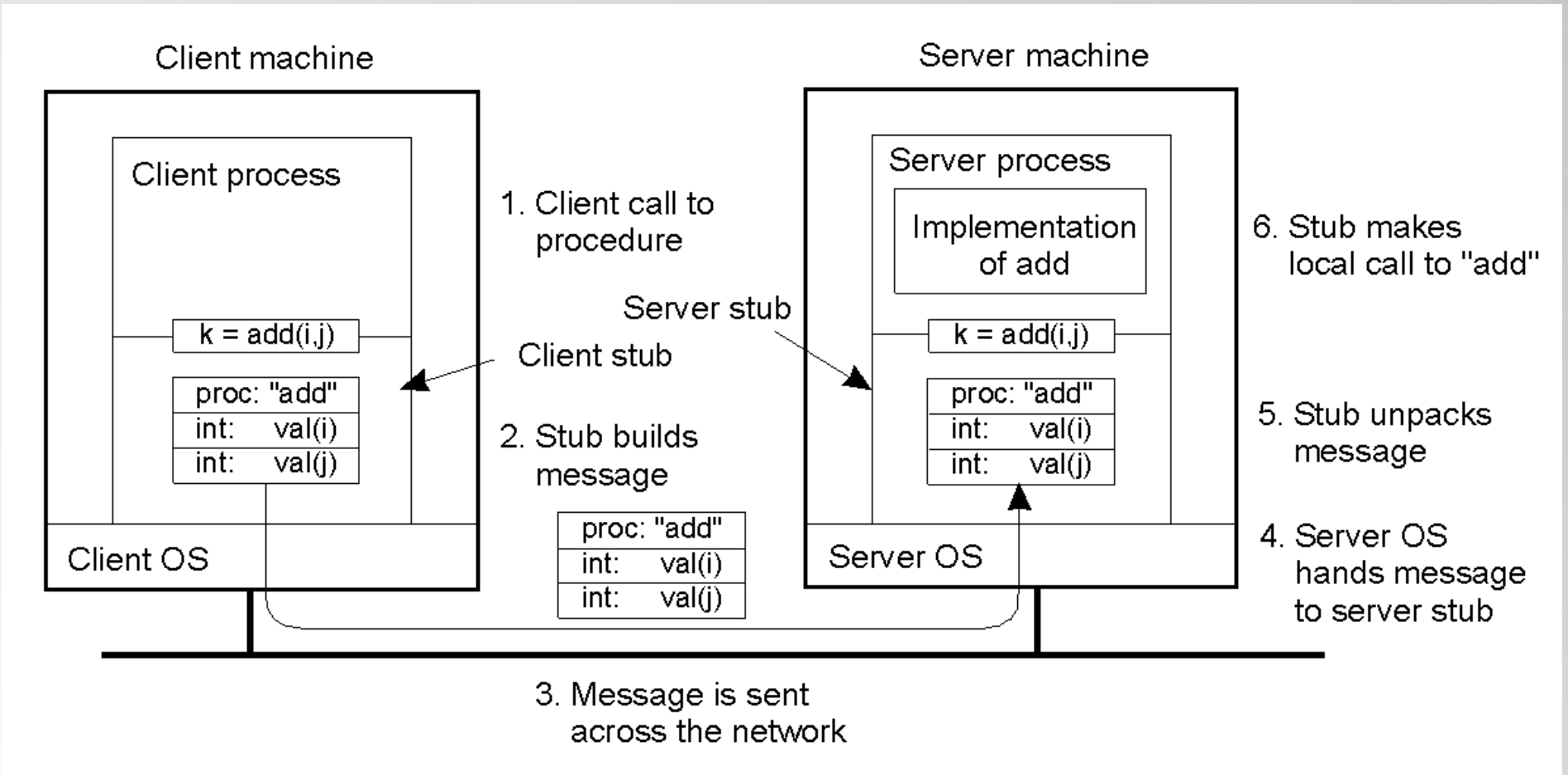
Funções dos Stubs

- **Client stub**
  1. intercepta a chamada
  2. empacota os parâmetros (marshalling)
  3. envia mensagem de request ao servidor (através do núcleo)
- **Server stub** (a.k.a. **skeleton**)
  1. recebe a mensagem de request (através do núcleo)
  2. desempacota os parâmetros (unmarshalling)
  3. chama o procedimento, passando os parâmetros
  4. empacota o resultado
  5. envia mensagem de resposta ao cliente (através do núcleo)
- **Client stub**
  1. recebe a mensagem de resposta (através do núcleo)
  2. desempacota o resultado
  3. passa o resultado para o cliente

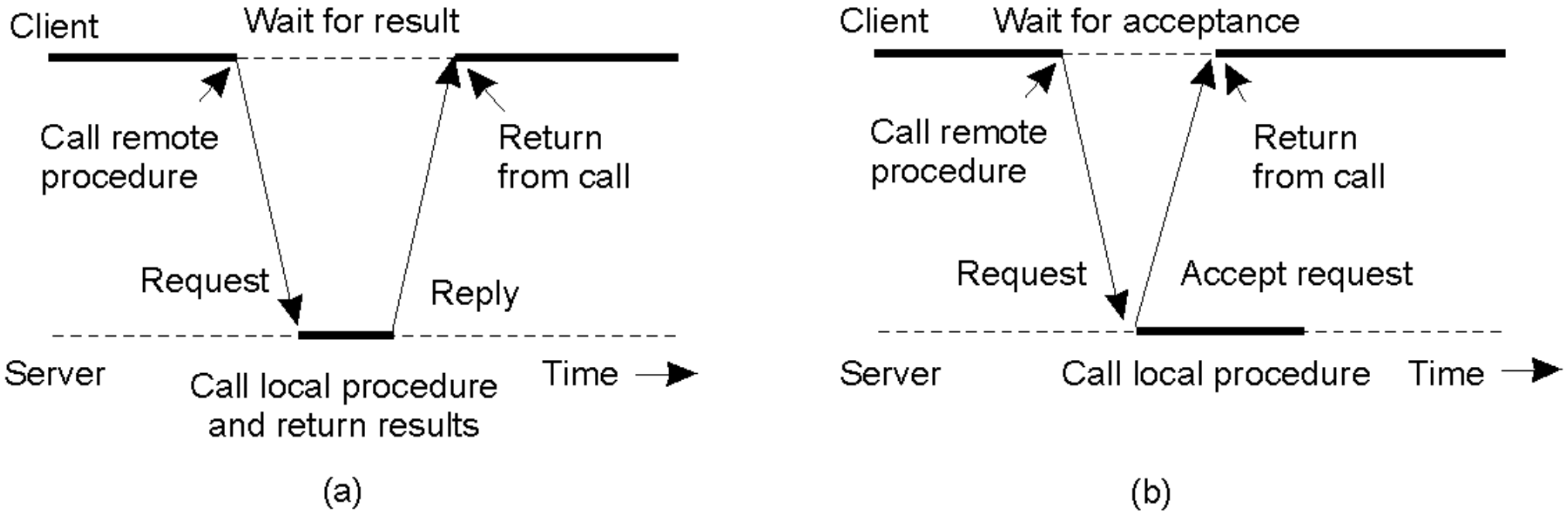
Funções dos Stubs

- **Client stub**
  1. intercepta a chamada
  2. empacota os parâmetros (marshalling)
  3. envia mensagem de request ao servidor (através do núcleo)
- **Server stub** (a.k.a. **skeleton**)
  1. recebe a mensagem de request (através do núcleo)
  2. desempacota os parâmetros (unmarshalling)
  3. chama o procedimento, passando os parâmetros
  4. empacota o resultado
  5. envia mensagem de resposta ao cliente (através do núcleo)
- **Client stub**
  1. recebe a mensagem de resposta (através do núcleo)
  2. desempacota o resultado
  3. passa o resultado para o cliente

RPC: Passagem de Parâmetros



RPC Assíncrono



a) Interação cliente-servidor em um RPC tradicional
b) Interação usando RPC assíncrono