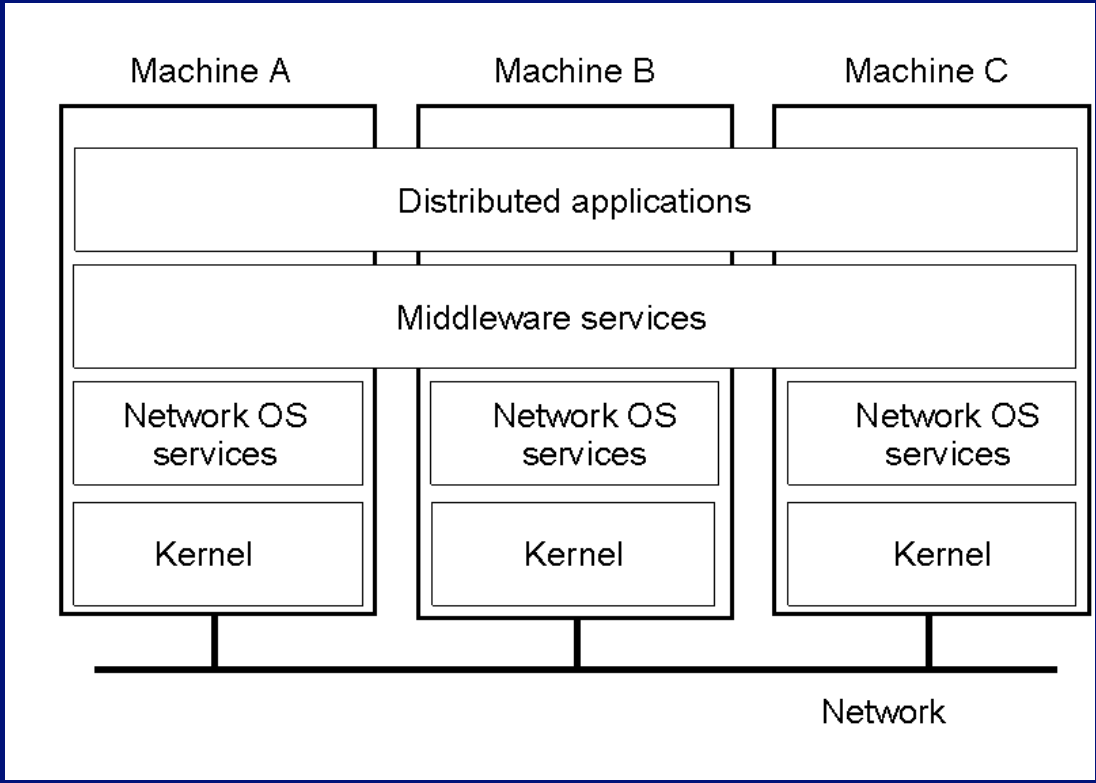
Comunicação Interprocessos (IPC)
RPC

Arquitetura de SD Modernos



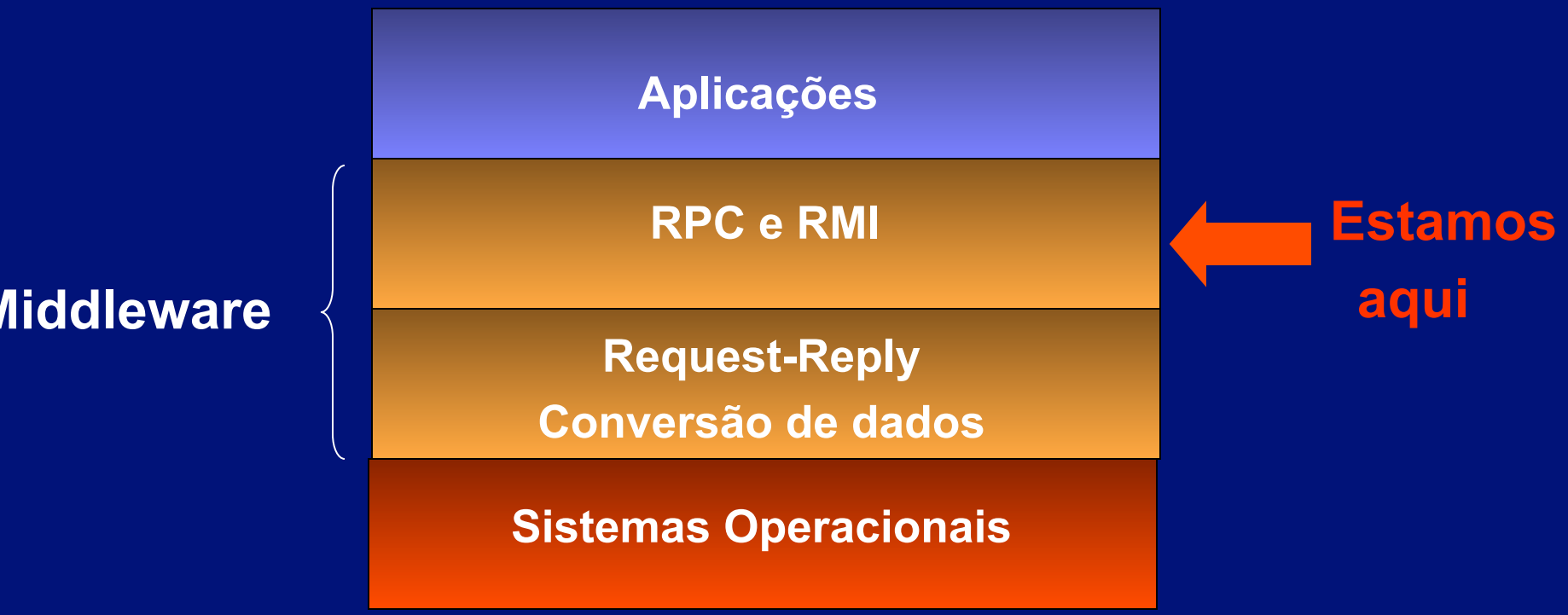
- Funções do middleware
  - Esconder heterogeneidade
  - Prover transparência

Comunicação em Sistemas Distribuídos

- Tem que ser através de troca de mensagens
  - Não há memória compartilhada
- Muito mais complicado do que usar memória compartilhada
  - Quantos volts devem ser usados para sinalizar o bit 0 e o bit 1?
  - Como o receptor sabe qual o último bit de uma mensagem?
  - Como ele detecta se uma mensagem foi corrompida?
  - Quais são os formatos dos dados (inteiros, strings etc)?
  - ???

Comunicação em Sistemas Distribuídos

- O middleware facilita a comunicação em SD



Request-Reply

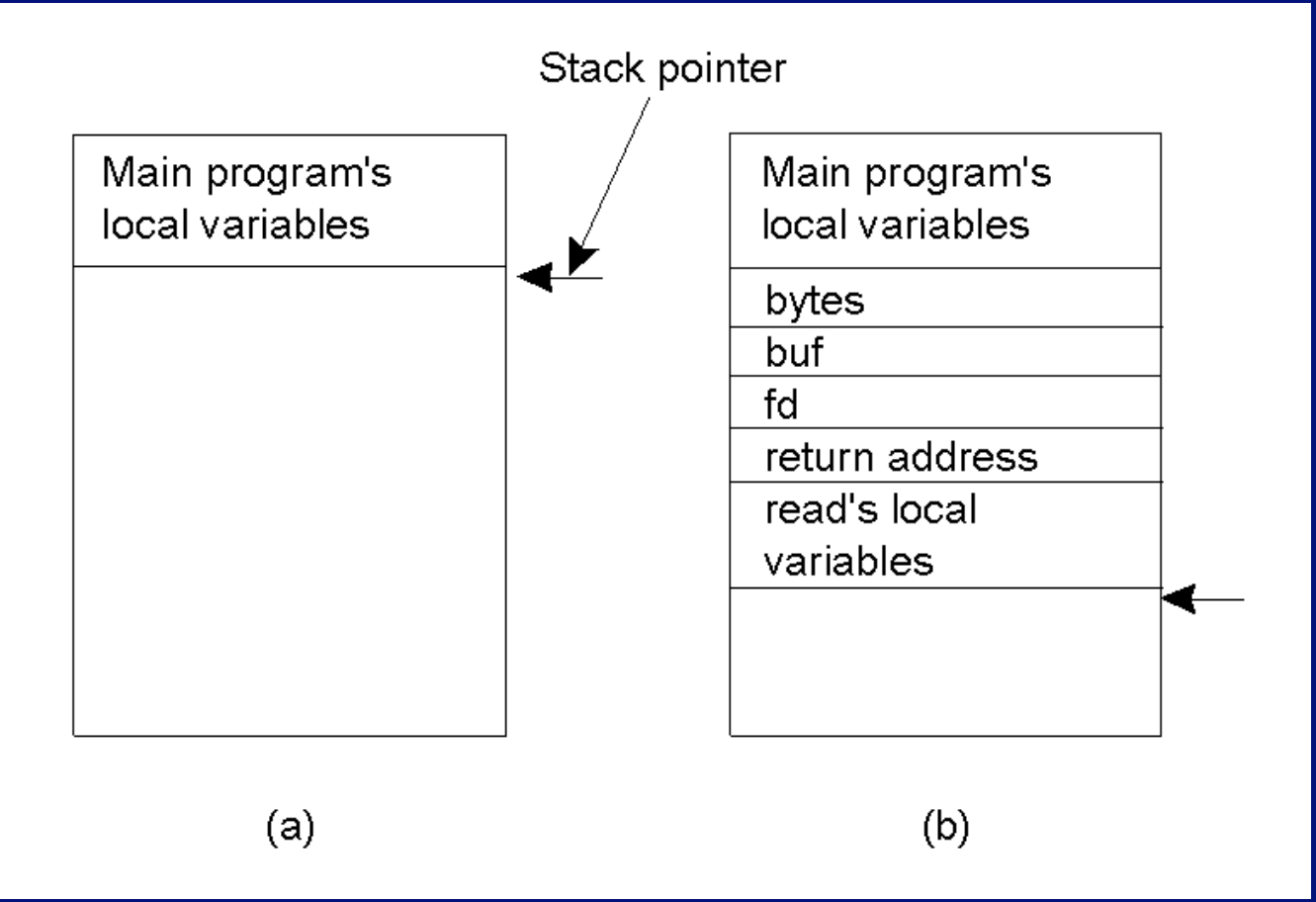
- Como é possível implementar uma aplicação distribuída Cliente/Servidor diretamente em cima desse nível?
  - Há várias APIs disponíveis: API Socket, Windows Socket, Java Socket
- Primitivas de comunicação send e receive
  - Não provêem transparência

RPC

- Forma totalmente diferente de implementar IPC
  - Birrel e Nelson (1984)
- Programas chamam procedimentos localizados em outras máquinas da mesma forma como chamam procedimento locais

Chamada a procedimento local

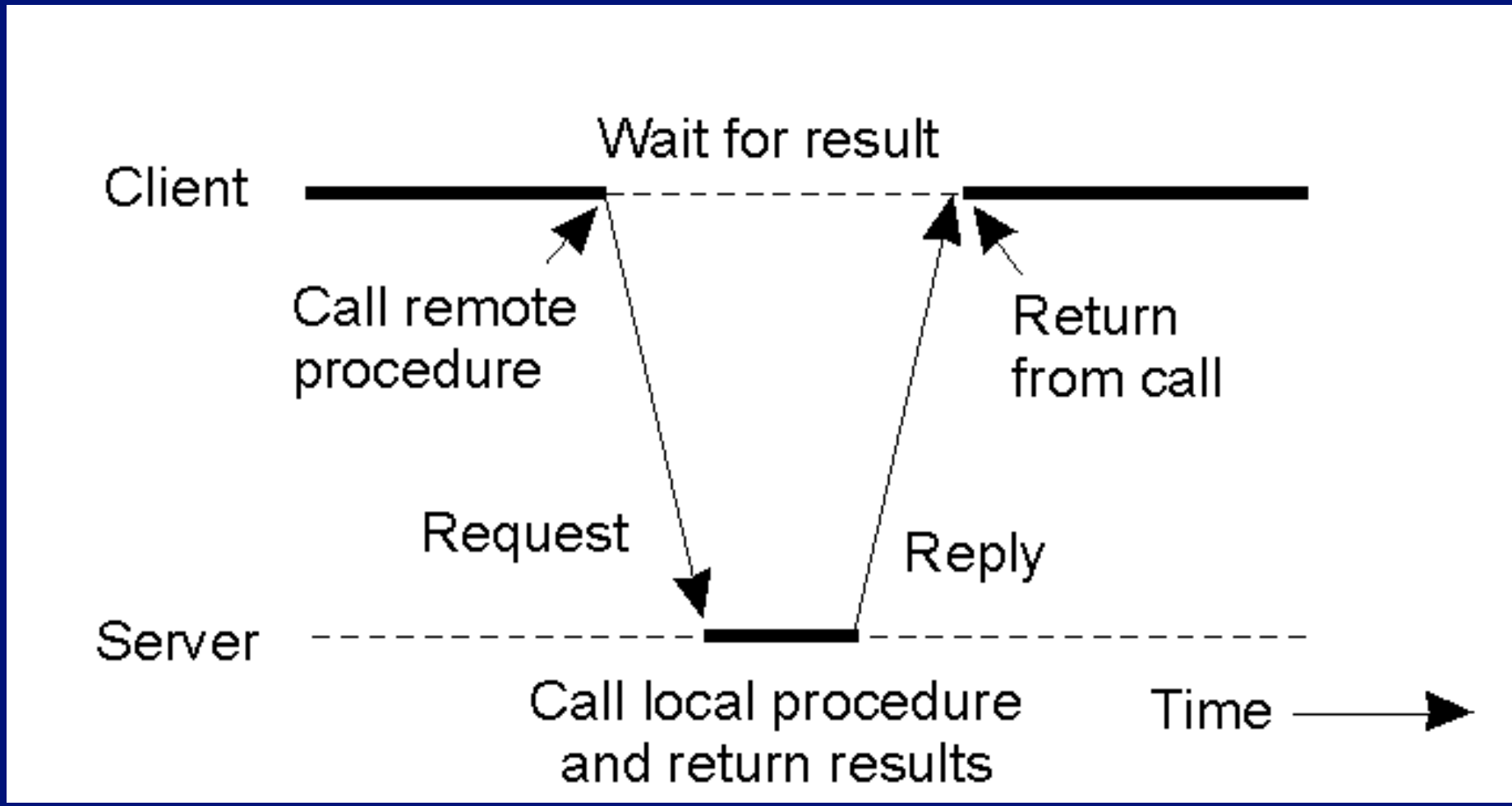
- count = read(fd, buf, nbytes)



Chamada a procedimento remoto

- Objetivo: ser tão parecida quanto possível a uma chamada local
- Quem chama não precisa saber que o procedimento chamado está executando em outra máquina
- Quem é chamado não precisa saber que quem chamou está executando em outra máquina
- Como fazer isso?

- O que acontece quando um read é feito localmente?
- Que tal trocar a versão do read por stubs?
- Stub cliente
  - Não pede os dados ao sistema operacional local
  - Mas empacota os parâmetros em uma mensagem e a envia para o servidor remoto


Wait for result
Client
Call remote procedure
Return from call
Request
Reply
Server
Call local procedure and return results
Time

- **Stub servidor**
  - Transforma uma requisição que chega pela rede em uma chamada local
  - Desempacota os parâmetros contidos na mensagem
  - Chama o servidor da maneira usual (chamada local)
  - Quando obtém o controle de volta (servidor terminou de executar), o stub do servidor empacota o resultado em uma mensagem
  - Envia a mensagem de volta para o cliente

- Stub cliente
  - Recebe a mensagem pela rede
  - Desempacota os resultados e retorna para quem chamou (o cliente de fato)
- Todos os detalhes da troca de mensagens estão escondidos nos stubs
  - send/receive são usados entre stub cliente e stub servidor

RPC

1. O cliente chama o stub cliente da forma usual
2. O stub cliente constróe uma mensagem e chama o SO local
3. O SO do cliente envia uma mensagem para o SO remoto
4. O SO remoto entrega a mensagem para o stub servidor
5. O stub servidor desempacota os parâmetros e chama o servidor
6. O servidor faz seu trabalho e retorna o resultado para o stub servidor

7. O stub servidor empacota o resultado em uma mensagem e chama o SO local
8. O SO do servidor envia a mensagem para o SO do cliente
9. O SO do cliente entrega a mensagem para o stub cliente
10. O stub cliente desempacota o resultado e retorna para o cliente

Nem o cliente nem o servidor sabem dos passos intermediários.

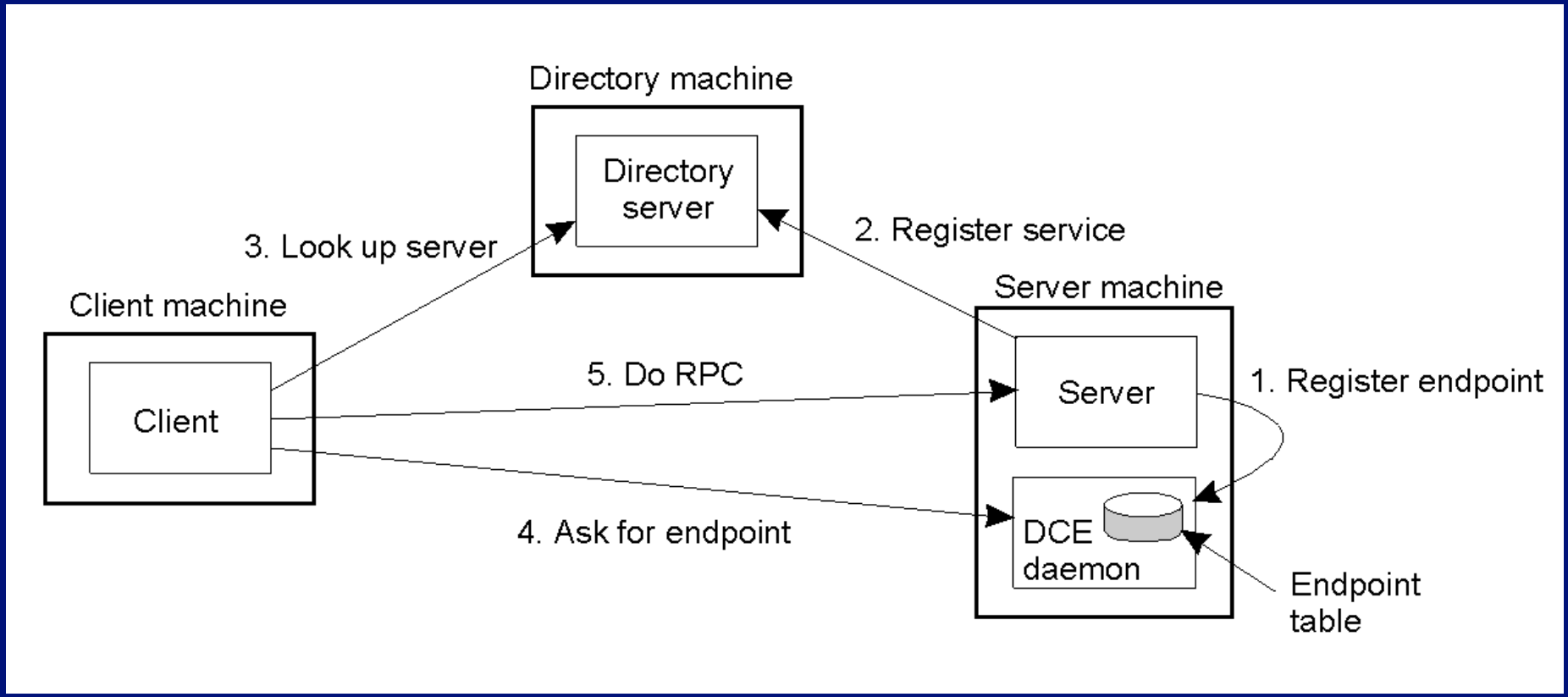
Passagem de Parâmetros

- Parâmetros por valor
  - Problemas com formato dos dados
    - EBCDIC X ASCII
    - Complemento de 1 X Complemento de 2
    - Little endian X big endian

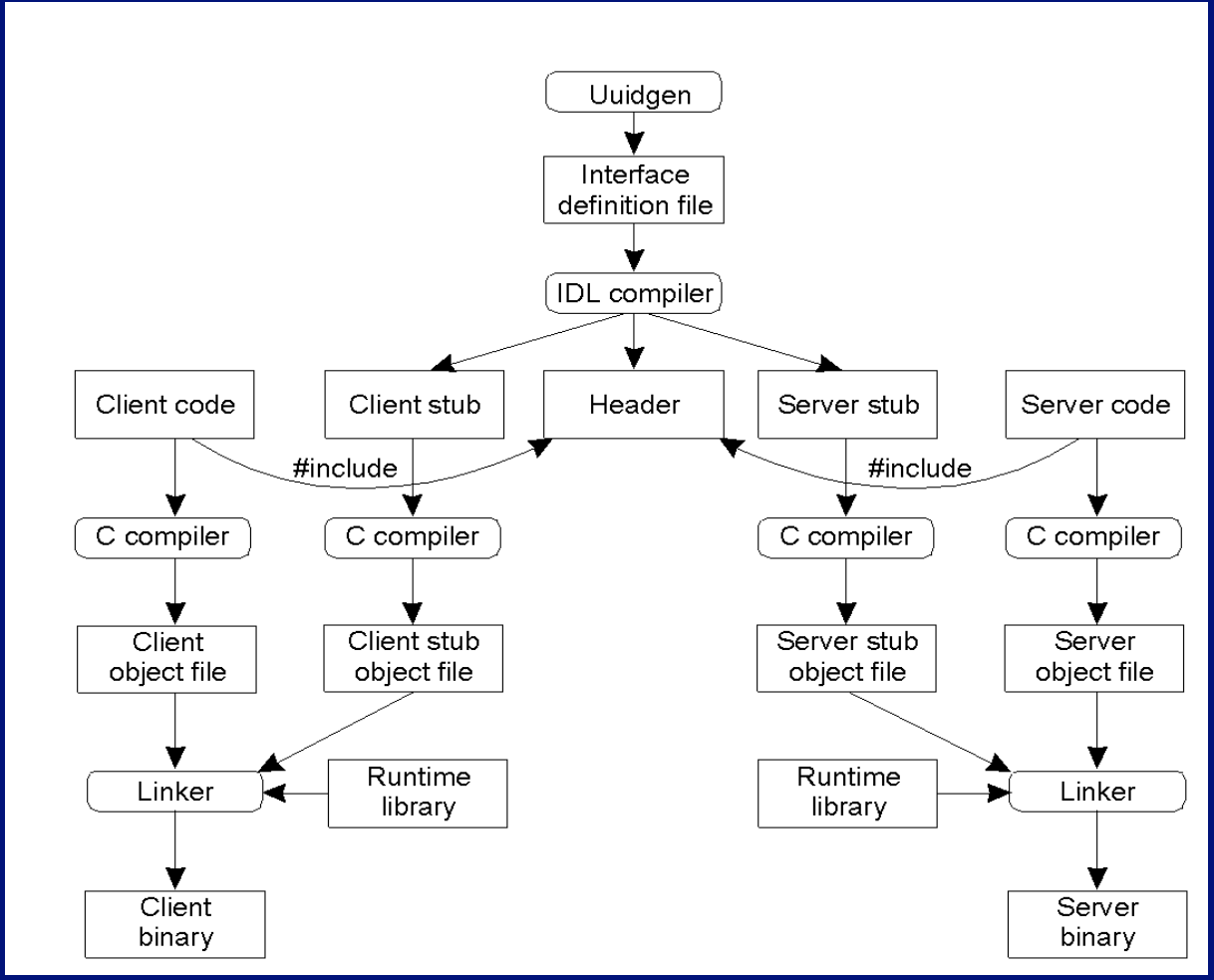
Passagem de Parâmetros

- Parâmetros por referência
  - O procedimento chamado ocupa um espaço de endereçamento totalmente diferente do espaço de quem chamou
  - Trocar chamada por referência por copy/restore

Localizando um Servidor RPC



rpcgen


Uuidgen
Interface definition file
IDL compiler
Client code
Client stub
Header
Server stub
Server code
#include
#include
C compiler
C compiler
C compiler
C compiler
Client object file
Client stub object file
Server stub object file
Server object file
Linker
Runtime library
Runtime library
Linker
Client binary
Server binary