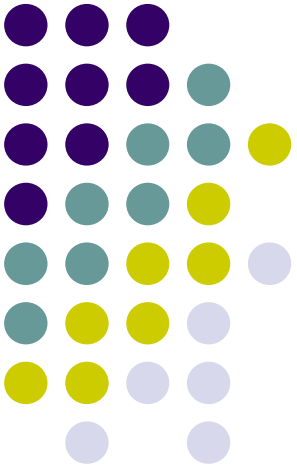
Sistemas Distribuídos

Universidade Federal do ABC

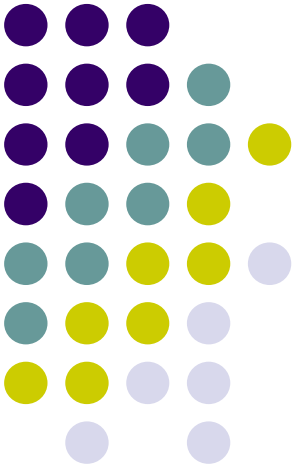
Turma:
Ciência da Computação

Prof. Dr. Francisco **Isidro** Massetto

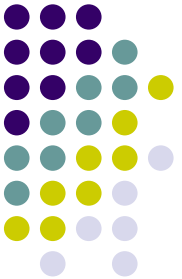


Introdução

Comunicação em Sistemas Distribuídos

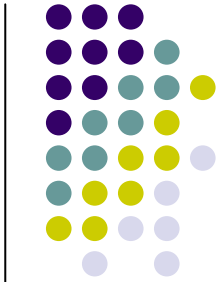


Introdução: Comunicação em Sistemas Distribuídos



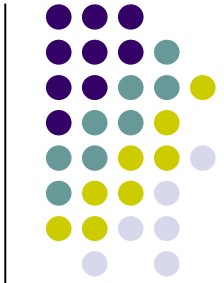
- A diferença mais importante entre Sistemas Distribuídos e Sistemas Centralizados é a Comunicação Interprocesso (Interprocess Communication – IPC);
- Sistemas Centralizados
  - memória partilhada;
- Sistemas Distribuídos
  - não existe memória partilhada;
  - Comunicação Interprocesso reformulada
  - Permitir que os processos se comuniquem para troca de dados ou acessos a recursos ou serviços em processadores remotos.

Introdução: Comunicação Interprocessos



- Sistema Distribuído é baseado na Troca de Mensagens (Message Passing);
- Exemplo:
  - Quando um Processo-A quer se comunicar com um Processo-B, o Processo-A constrói uma mensagem no seu próprio espaço de endereçamento (address space);
  - Em seguida, o Processo-A executa uma chamada de sistema (system call) que faz com que o sistema operacional pegue essa mensagem e envie pela rede para o Processo-B;
- Quais são os desafios (acordos e regras) necessários que permitem que os Processos A e B se comuniquem?

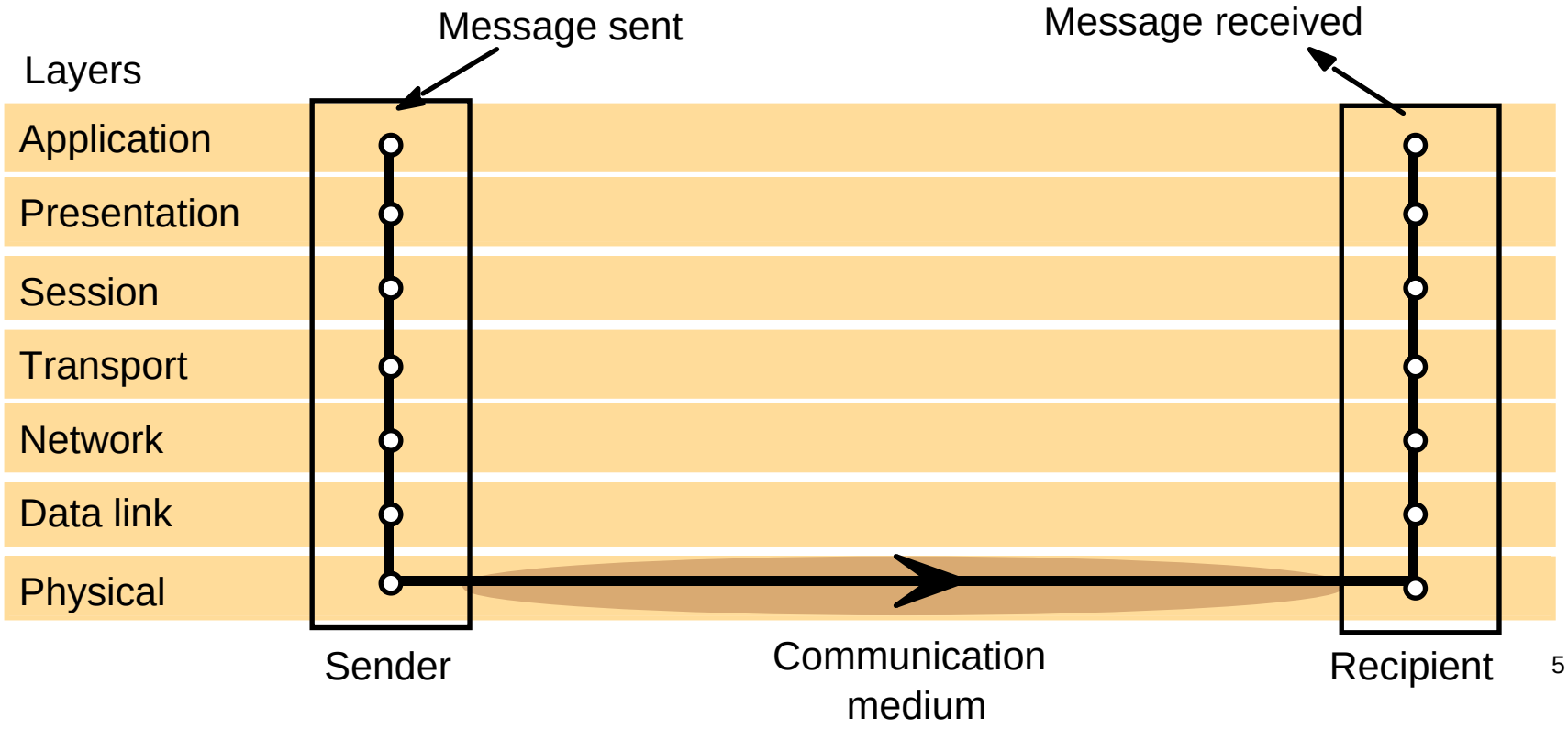
Protocolos: Conceitos Básicos



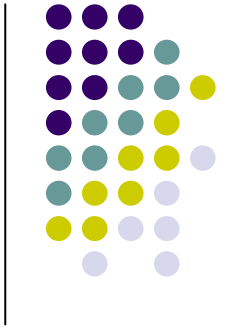
- Conjunto de regras, procedimentos e formatos para garantir a comunicação entre duas entidades geograficamente distintas
  - A seqüência de mensagens que devem ser trocadas;
  - O formato dos dados nas mensagens.
  - ISO (International Standard Organization), desenvolveu um modelo de referência
    - diversos níveis envolvidos
    - o que deve ser tratado em cada nível;
    - Open Systems Interconnection Reference Model, ISO OSI ou, simplesmente, Modelo OSI.

Protocolos: Camadas

- Comunicação é dividida em sete níveis ou camadas, onde cada camada trata de um aspecto específico da comunicação.

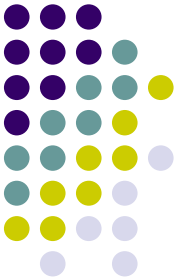


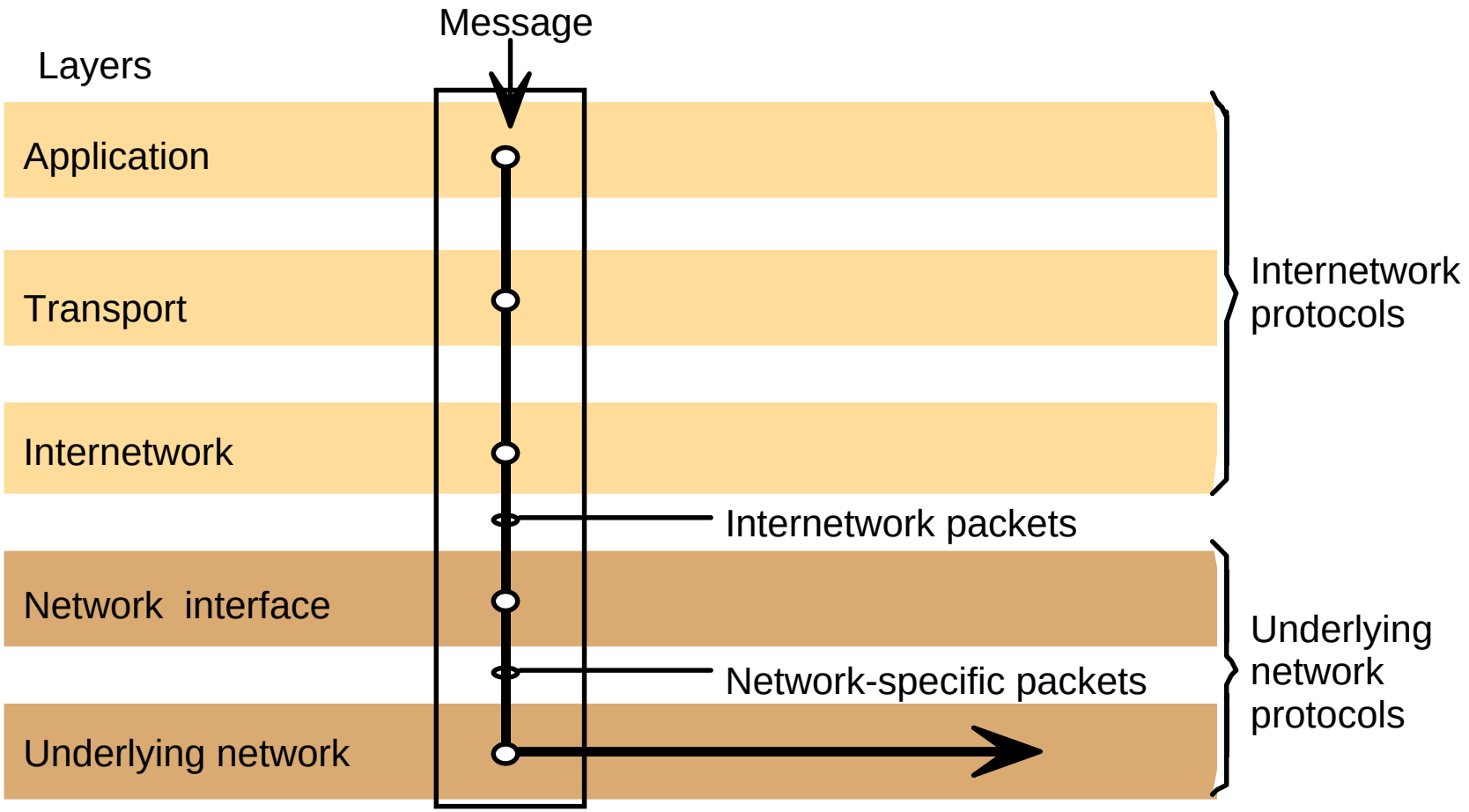
Protocolos: Desafios



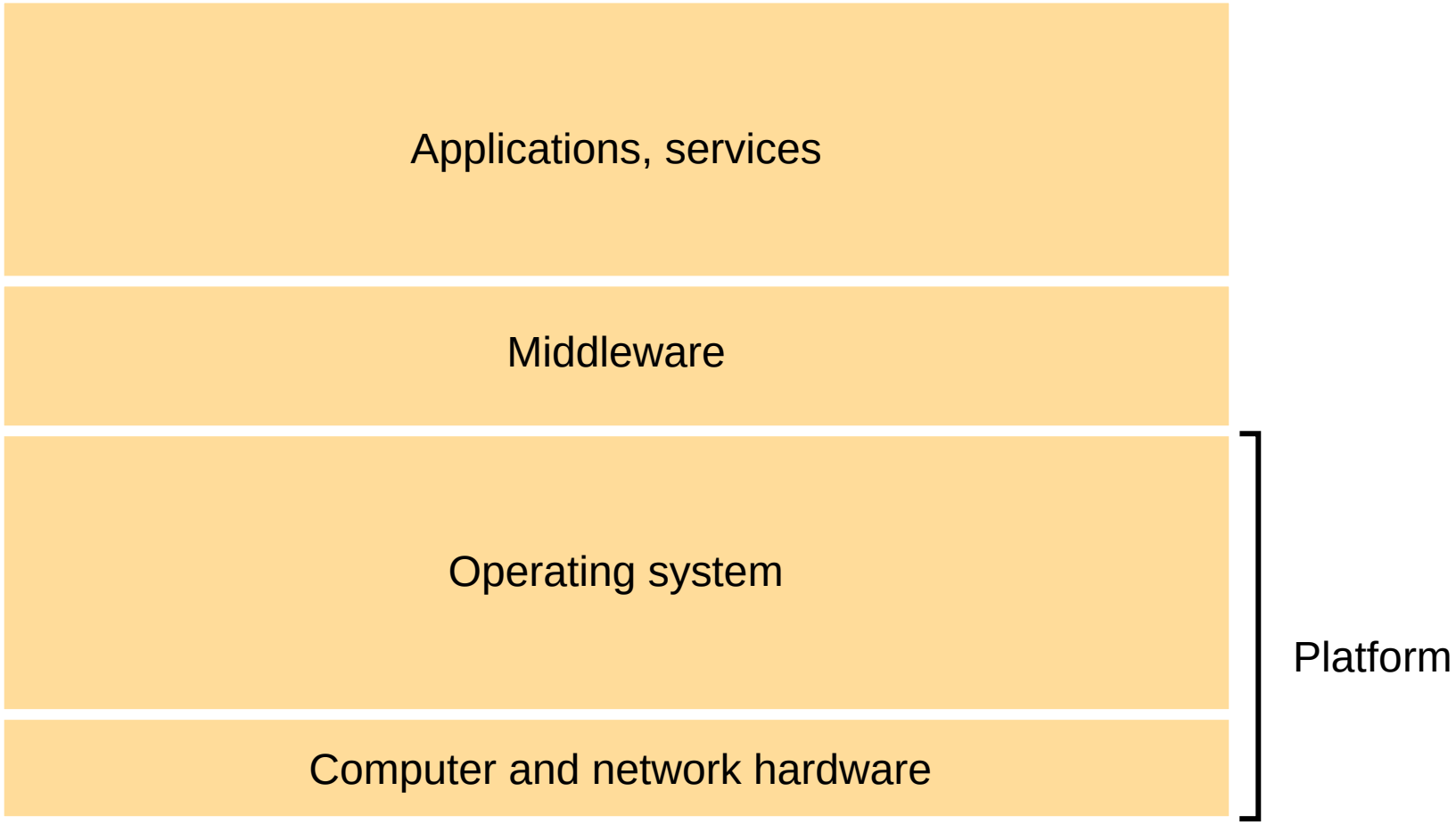
- Roteamento:
  - Prover o caminho mais eficiente para um pacote, através da aplicação de algoritmos de roteamento
- Controle de Congestionamento:
  - Evitar a degradação na vazão da rede através de atrasos no envio de pacotes;
  - Informar aos participantes da rota do pacote sobre o congestionamento.
- Internetworking:
  - Integrar diversos tipos de redes, endereçamento, protocolos, componentes de ligação (roteadores, bridges, hubs, switches, tunneling).

Protocolos: Camadas Internetworking

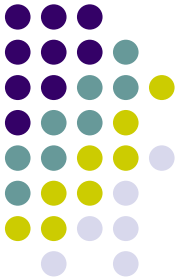




Comunicação Interprocessos: Camadas de Serviços

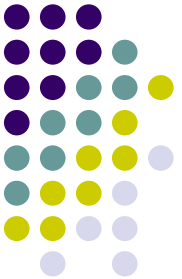


Comunicação Interprocessos: Modelo Cliente-Servidor



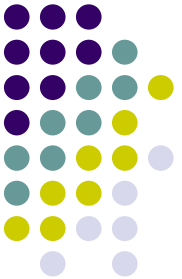
- A idéia é estruturar o sistema operacional como um grupo de processos cooperativos chamados:
  - Servidores: Oferecem serviços ao usuários;
  - Clientes: Usam os serviços provido pelos Servidores.
- Uma máquina pode executar:
  - Um único ou múltiplos processos clientes;
  - Um único ou múltiplos processos servidores;
  - Ou uma combinação das alternativas anteriores.

Comunicação Interprocessos: Modelo Cliente-Servidor

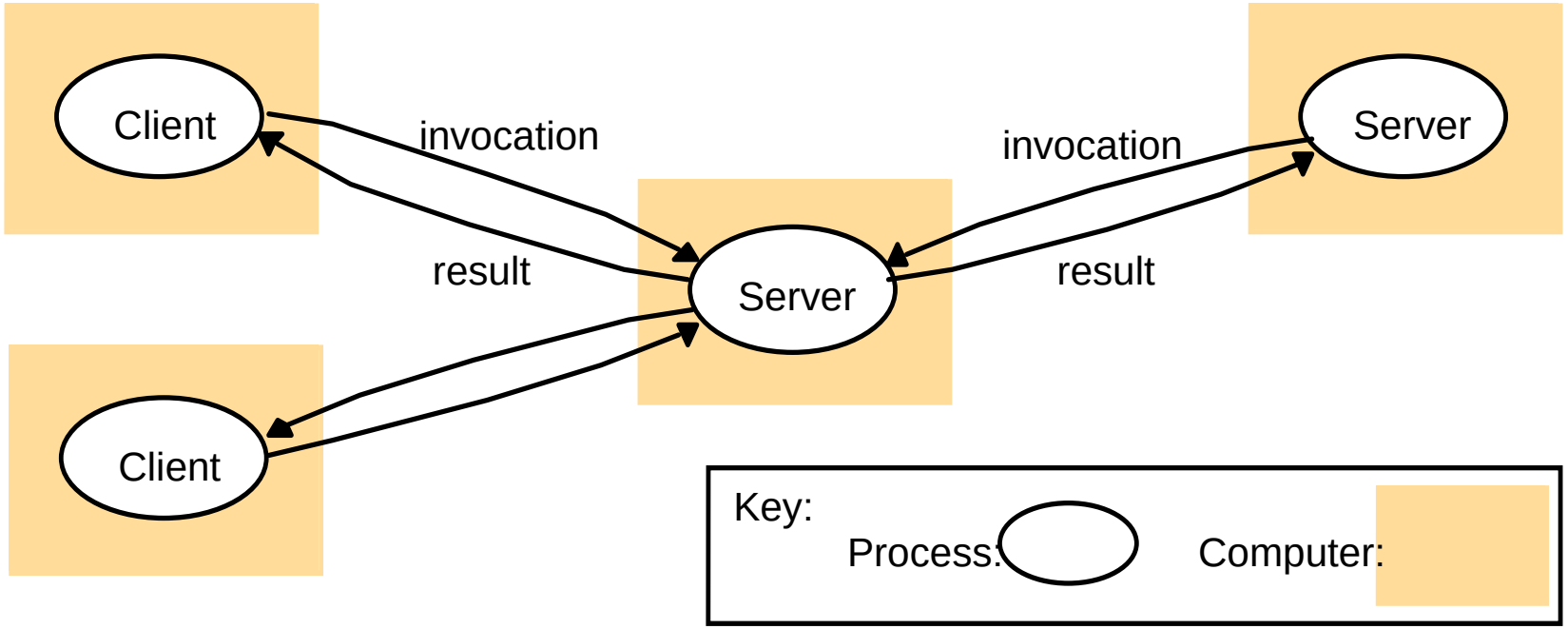


- Principais Vantagens:
  - Simplicidade;
  - Não tem necessidade de estabelecer e finalizar a conexão;
  - A mensagem de resposta de um servidor pode funcionar como confirmação de entrega de uma requisição
  - Eficiência;
  - Se as máquinas forem idênticas são necessários somente três níveis de protocolo:
    - Físico, Data Link e Request/Reply.

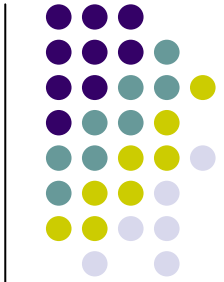
Comunicação Interprocessos: Modelo Cliente-Servidor



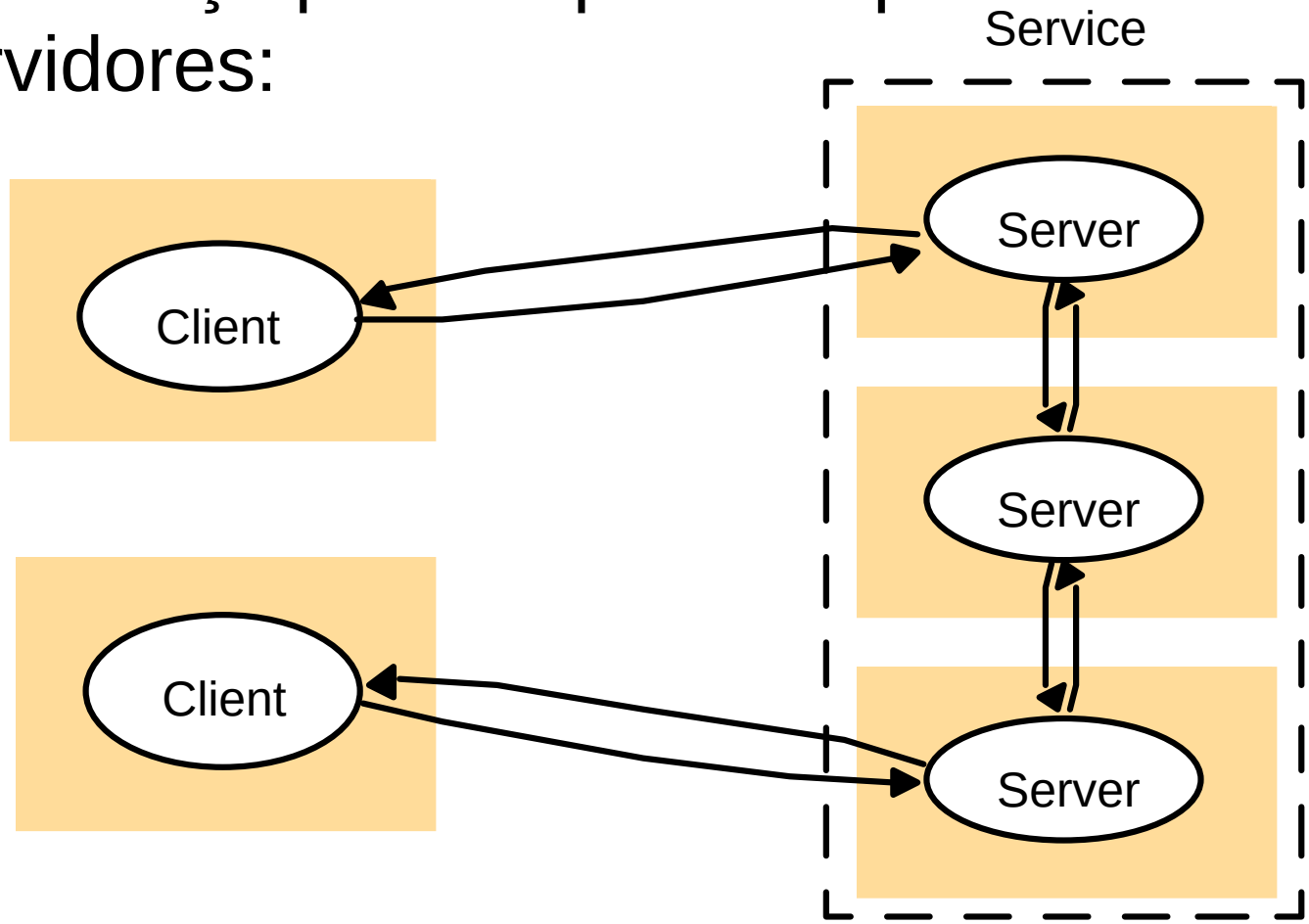
- Processos Clientes acionando individualmente Processos Servidores:



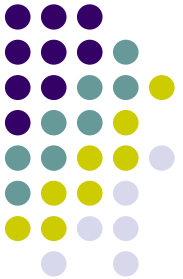
Comunicação Interprocessos: Modelo Cliente-Servidor



- Um serviço provido por Múltiplos Processos Servidores:

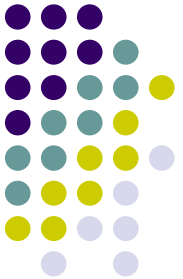


Modelo Cliente-Servidor: Troca de Mensagem



- Duas chamadas ao Sistema Operacional:
  - **send** (*destino*, *&ptmsg*)
    - Envia uma mensagem apontada por ptmsg para o processo identificado por destino, bloqueando o processo que executou o send até que a mensagem tenha sido enviada;
  - **receive** (*endereço*, *&ptmsg*)
    - Bloqueia o processo que executou o receive até que uma mensagem chegue;
    - Quando ela chega é copiada para o buffer apontado por ptmsg;
    - O parâmetro endereço especifica o endereço no qual o receptor está esperando a mensagem.

Troca de Mensagem: Exemplo de Processos C/S



- header.h

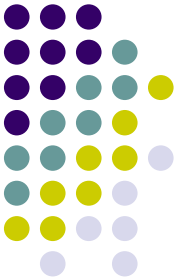
```
/*  Definições  dos  Clientes  e
  Servidores  */
#define MAX_PATH        255
#define BUF_SIZE       1024
#define FILE_SERVER     243

/*  Definições  das  operações
  permitidas */
#define CREATE          1
#define READ            2
#define WRITE           3
#define DELETE          4

/*  Códigos de Erros   */
#define OK              0
#define E_BAD_OPCODE   -1
#define E_BAD_PARAM    -2
#define E_IO           -3
```

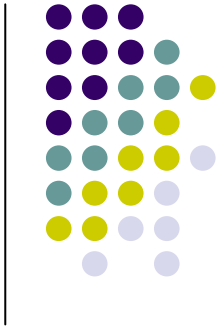
```
/*  Definições  do  formato  de
  mensagem */
struct message {
  long source;
  long dest;
  long opcode;
  long count;
  long offset;
  long extra1;
  long extra2;
  long result;
  char name [MAX_PATH];
  char data[BUF_SIZE];
 };
```

Processo Servidor



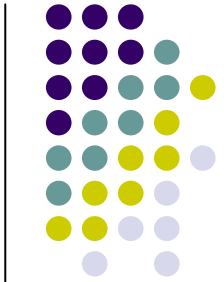
```
#include <header.h>
void main (void) {
     struct message m1, m2; int r;
     while (1) {
          receive (FILE_SERVER, &m1);
          switch (m1.opcode) {
           case CREATE:
             r=do_create(&m1,&m2);break;
           case READ:
             r=do_read(&m1,&m2); break;
           case WRITE:
             r=do_write(&m1,&m2); break;
           case DELETE:
             r=do_delete(&m1,&m2); break;
           default:
             r=E_BAD_OPCODE;  }
           m2.result = r;
        send (m1.source, &m2);  }}
```

Processo Cliente

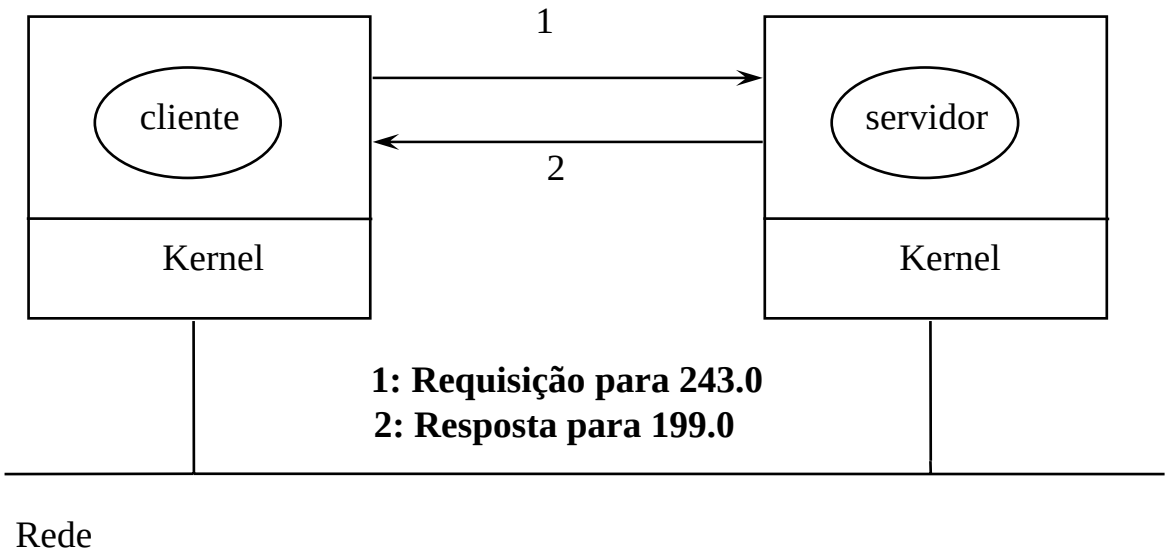


```
#include <header.h>
int copy(char *src, char *dst) {
 struct message m1; long
 position, client = 110;
 initialize();
 position = 0;
    do {
   /* Pega um bloco de dado do
 arquivo
       fonte */
      m1.opcode = READ;
      m1.offset = position;
      m1.count = BUF_SIZE;
      strcpy(&m1.name, src);
      send (FILE_SERVER, &m1);
      receive (client, &m1);
 /* Escreve o bloco de dado
recebido no arquivo destino */
  m1.opcode = WRITE;
  m1.offset = position;
  m1.count = m1.result;
   strcpy(&m1.name, dst);
   send (FILE_SERVER, &m1);
    receive (client, &m1);
   position += m1.result;
 } while (m1.result > 0);
 return
     (m1.result>=0?ok:m1.result);
}
```

Troca de Mensagem: Endereçamento

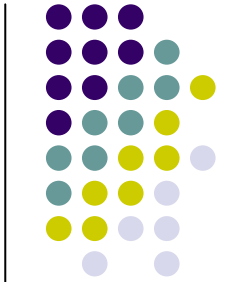


- Endereçamento Máquina Processo:

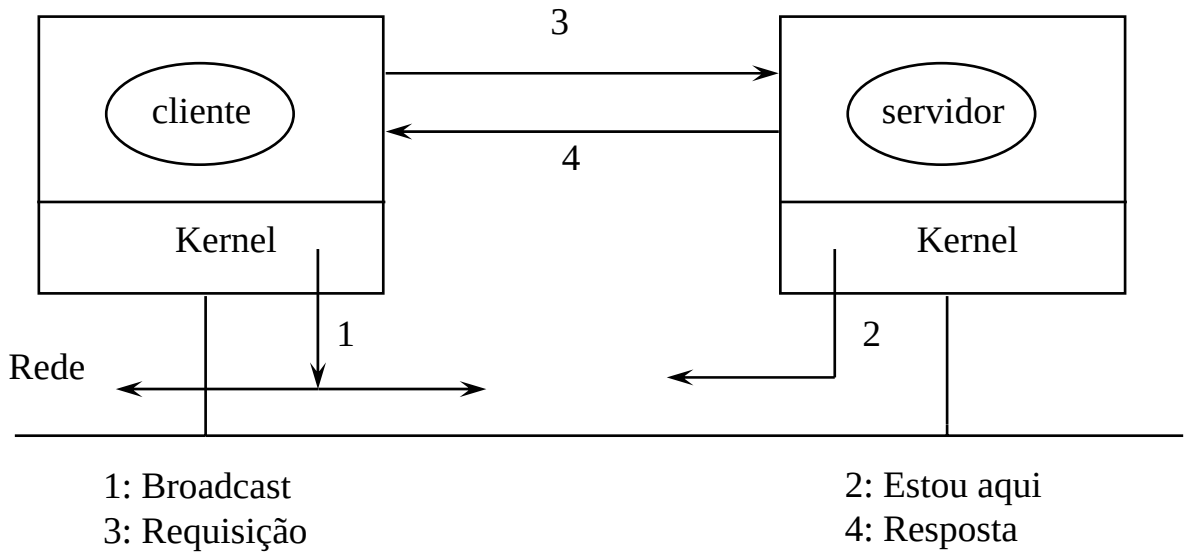


- Não é transparente
  - se um servidor não estiver disponível teremos recompilação para poder realizar o serviço em outro servidor.

Troca de Mensagem: Endereçamento

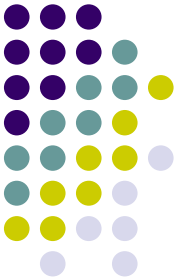


- Endereçamento Aleatório:

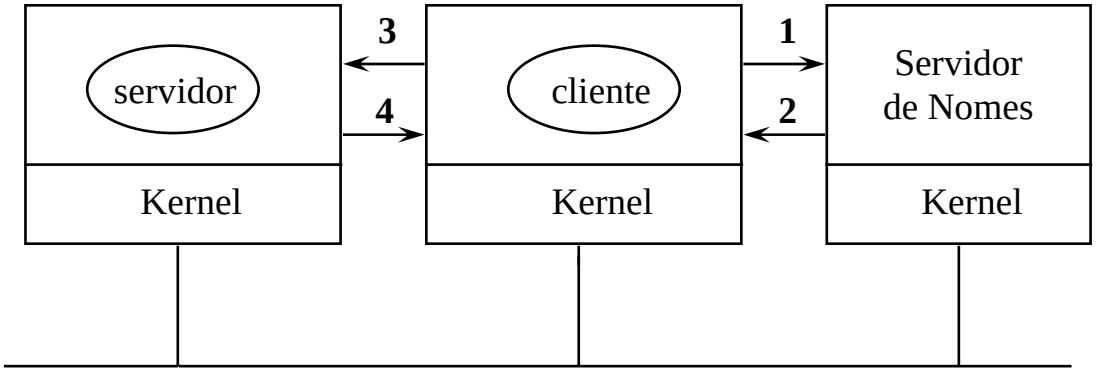


- Broadcast gera carga extra de comunicação no sistema.

Troca de Mensagem: Endereçamento

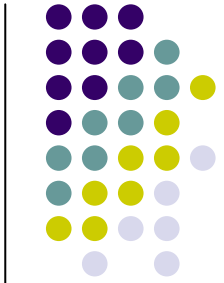


- Endereçamento usando um Servidor de Nomes:

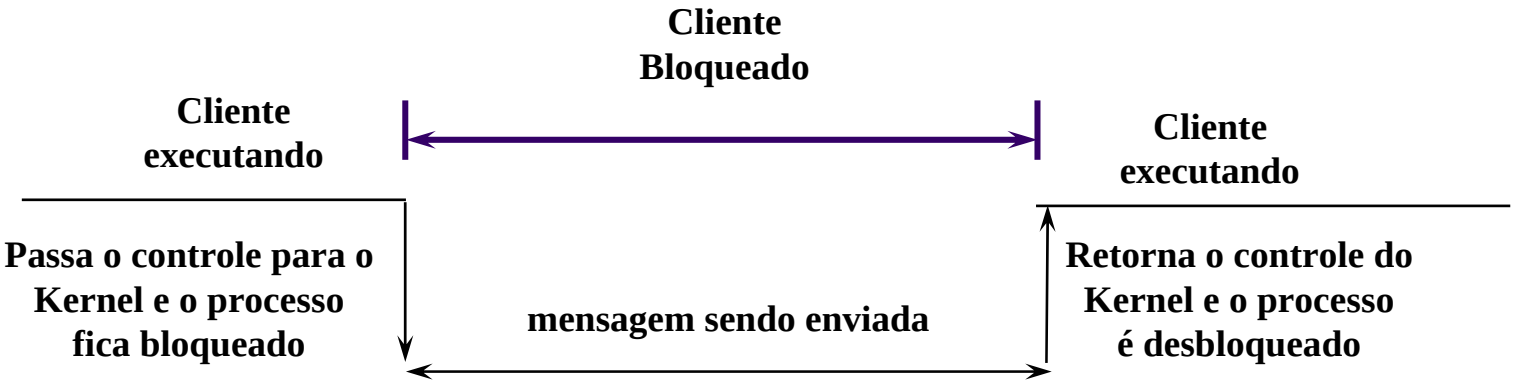


- Problemas:????

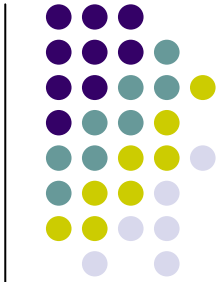
Troca de Mensagem: Primitiva Send



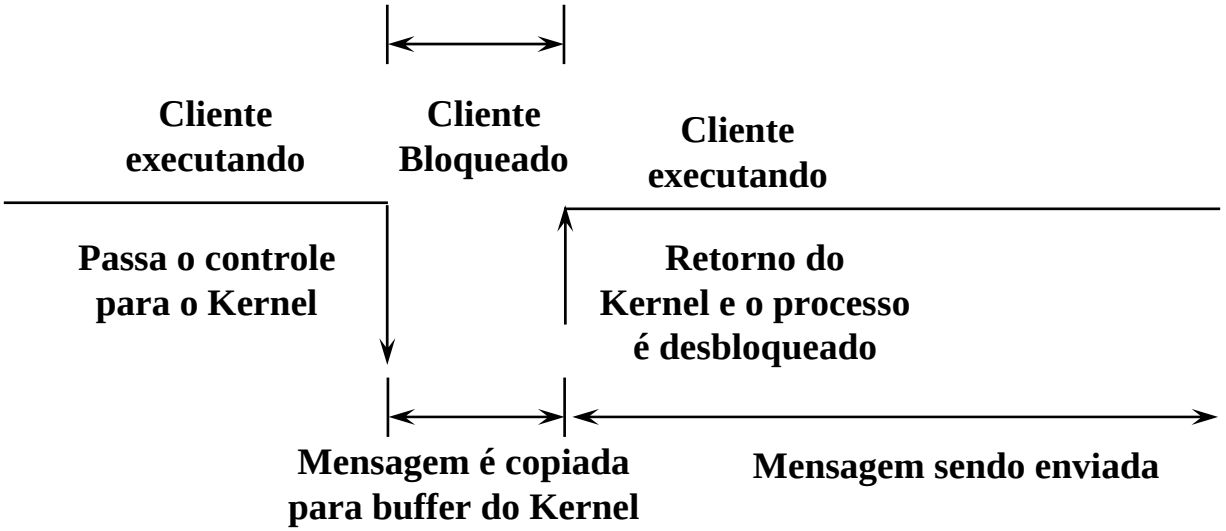
- Primitivas Bloqueadas (Síncronas):
  - Primitivas vistas até agora (send e receive) são chamadas primitivas bloqueadas (bloqueantes).
  - Enquanto a mensagem está sendo enviada ou recebida, o processo permanece bloqueado (suspenso).



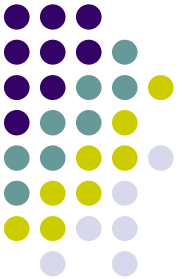
Troca de Mensagem: Primitiva Send



- Primitivas Não Bloqueadas (Assíncronas):
  - Quando um send é executado o controle retorna ao processo antes da mensagem ser enviada;
  - O processo que executa o send pode continuar processando enquanto a mensagem é enviada.

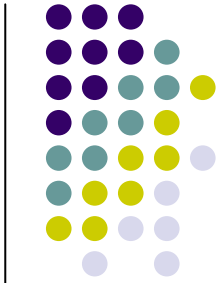


Troca de Mensagem: Primitiva Send



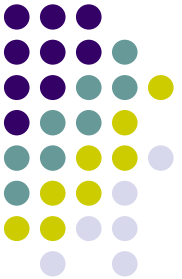
- Algumas Considerações:
  - A escolha do tipo de primitiva a ser usado (bloqueada ou não bloqueada) é feita pelo projetista do sistema;
  - O processo que executa o send não pode modificar o buffer de mensagem até a mensagem ter sido totalmente enviada;
  - Existem dois modos para o processo ficar sabendo que a mensagem já foi enviada:
    - Send com cópia - desperdício do tempo de CPU com a cópia extra;
    - Send com interrupção – torna a programação mais difícil e não portável

Troca de Mensagem: Primitiva Receive



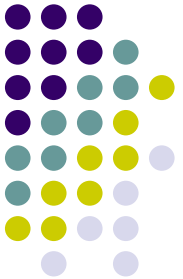
- Primitivas Não Bloqueadas:
  - Informa ao Kernel onde está o buffer para receber a mensagem e retorna o controle ao processo que o executou **receive**;
  - Existem três modos do **receive** saber que a recepção da mensagem ocorreu:
    - Primitiva wait;
    - Receive condicional;
    - Interrupção.

Troca de Mensagem: Primitiva Receive

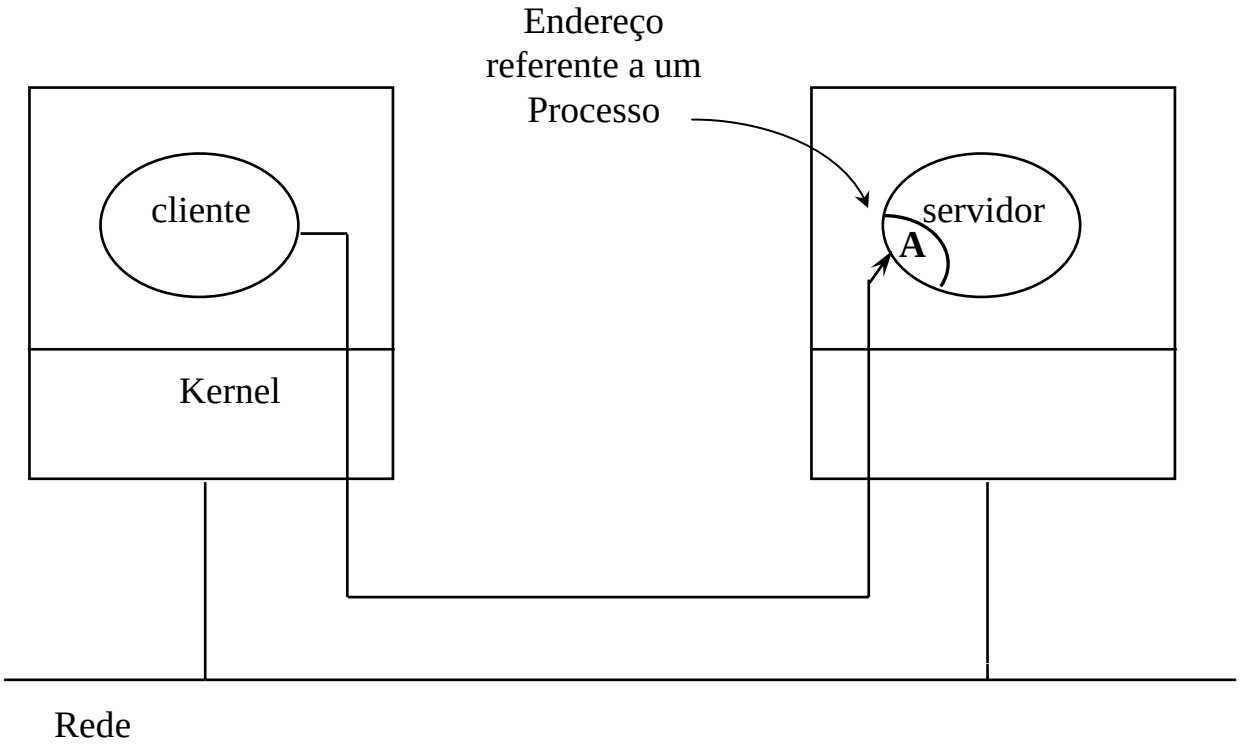


- Primitivas “Não Bufferizadas”:
  - As primitivas descritas até agora são Não Bufferizadas;
  - O que acontece com um processo que executa **receive (endereço, &m)** após a execução de um send ?
  - Como o Kernel sabe onde colocar a mensagem que venha chegar? Não sabe!
  - Desta forma, a mensagem é descartada e espera o *timeout* ocorrer. A mensagem é então retransmitida.

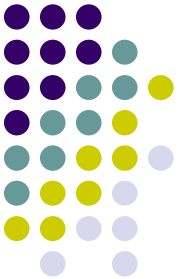
Troca de Mensagem: Primitiva Receive



- Primitivas “Não Bufferizadas”:

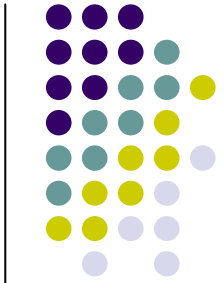


Troca de Mensagem: Primitiva Receive



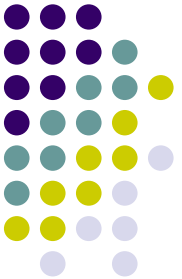
- Primitivas “Bufferizadas”:
  - O Kernel receptor mantém as mensagens armazenadas por um período de tempo, esperando que primitivas receive sejam executadas;
  - Reduz a chance da mensagem ser descartada;
  - Problema de armazenamento e gerenciamento de mensagens;
  - Buffers precisam ser alocados e gerenciados.

Troca de Mensagem: Primitiva Receive

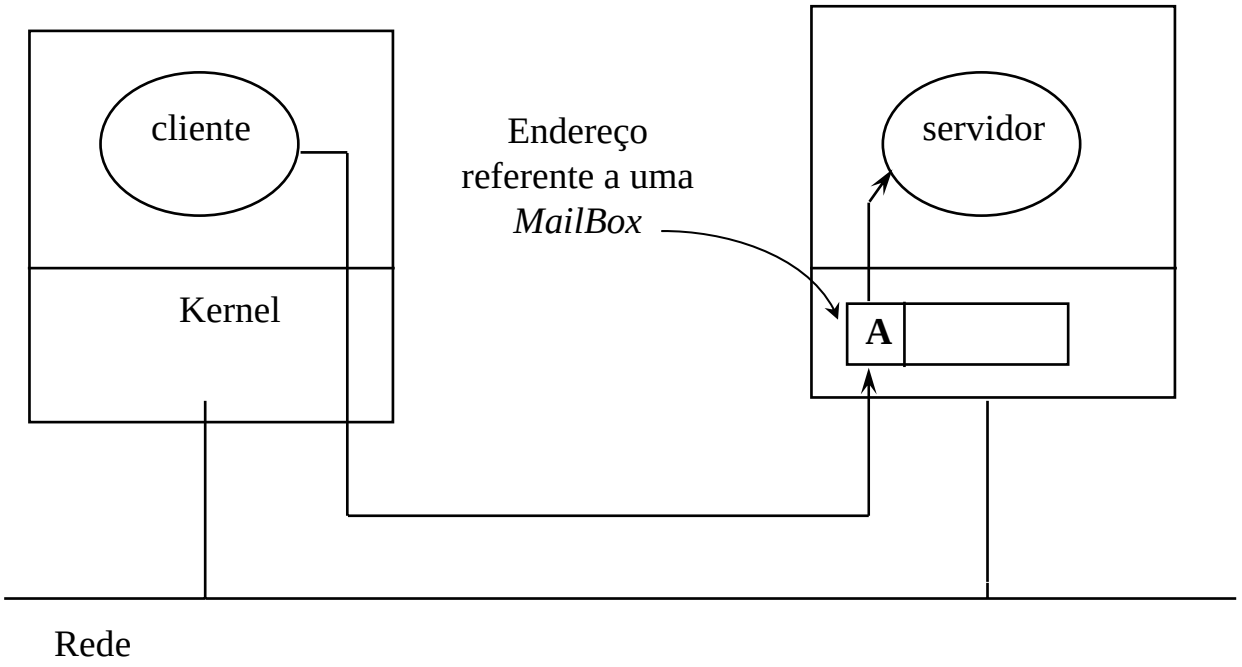


- Primitivas “Bufferizadas” - Implementação (Mailbox):
  - Definição de uma estrutura de dados chamada mailbox;
  - Um processo interessado em receber mensagens pede ao Kernel para criar um mailbox para ele;
  - O mailbox recebe um endereço para poder ser manipulado;
  - Toda mensagem que chega com aquele endereço é colocada no mailbox;
  - Receive agora simplesmente remove uma mensagem do mailbox ou fica bloqueado se não tem mensagem.

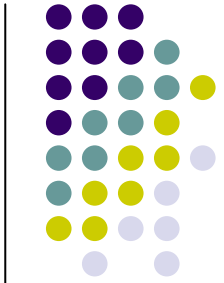
Troca de Mensagem: Primitiva Receive



- Primitivas “Bufferizadas” – Implementação (Mailbox):

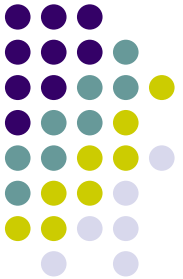


Troca de Mensagem: Confiabilidade das Primitivas



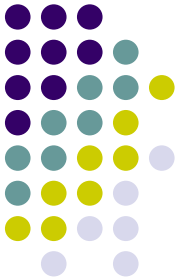
- Primitivas Confiáveis e Não Confiáveis
  - Até agora temos assumido que quando um cliente envia uma mensagem o servidor a receberá;
  - O modelo real é mais complicado que o modelo abstrato;
  - Mensagens podem ser perdidas, afetando desta forma a semântica do modelo de comunicação;
  - No caso de uma primitiva bloqueada ser usada, quando um cliente envia uma mensagem, o processo é suspenso até que a mensagem seja efetivamente enviada;
  - Entretanto, não existe garantia de que quando o processo for reativado a mensagem realmente foi entregue.

Troca de Mensagem: Confiabilidade das Primitivas



- **1** Redefinir a semântica do send para ser não confiável:
  - O sistema não dá garantias sobre uma mensagem que sendo enviada
  - Tornar a comunicação confiável uma tarefa do usuário (hummmm - complicado!!).
- **2** Requerer que o Kernel do receptor envie uma mensagem acknowledgement (ack - confirmação) para o Kernel do transmissor
  - O Kernel só libera o cliente quando o ack for recebido;
  - O ack é uma operação realizada pelos dois Kernels, sem o conhecimento do cliente e servidor

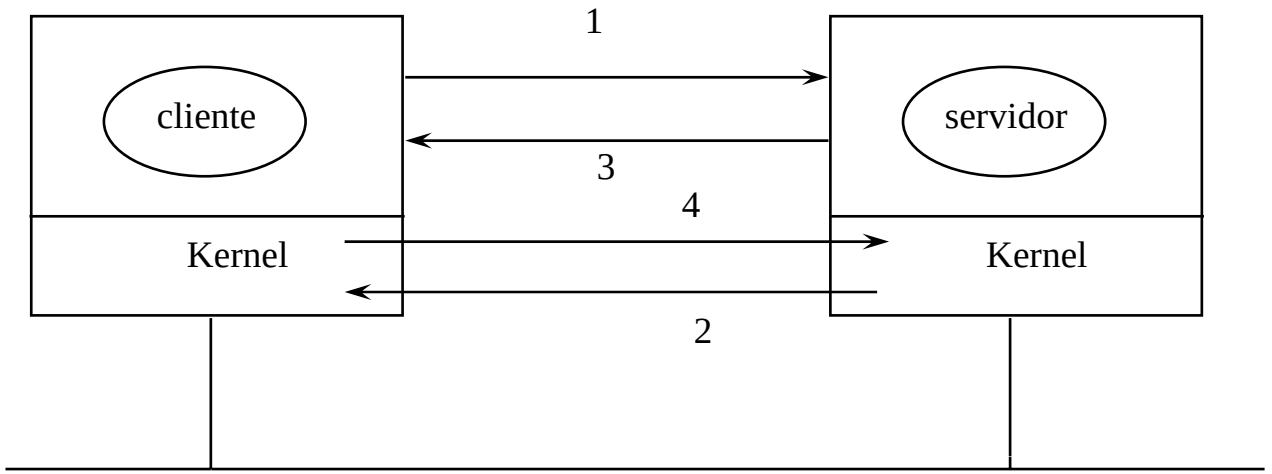
Troca de Mensagem: Confiabilidade das Primitivas



- **3** Aproveitar o fato que a comunicação cliente-servidor é estruturada como uma requisição do cliente para o servidor, seguido de uma resposta do servidor para o cliente:
  - O cliente é bloqueado depois do envio da mensagem;
  - O Kernel do servidor não envia um ack, em vez disto a resposta serve de ack;
  - Desta forma o processo de envio da mensagem permanece bloqueado até a resposta chegar;
  - Se isto demorar muito o Kernel pode reenviar a requisição, protegendo-se contra a perda de mensagens.

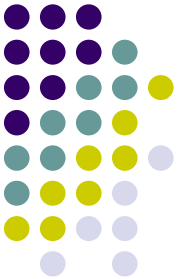
Troca de Mensagem: Confiabilidade das Primitivas

Mensagens de ACK individual

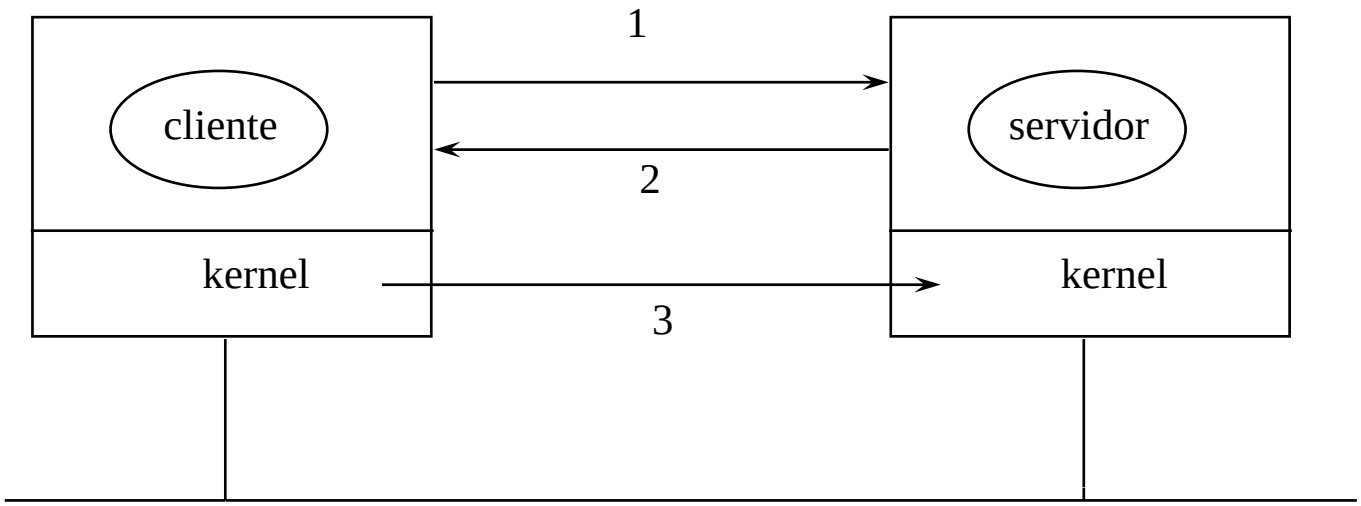


1: Requisição (Cliente para Servidor)
2: ACK (Kernel para Kernel)
3: Resposta (Servidor para Cliente)
4: ACK (Kernel para Kernel)

Troca de Mensagem: ***Confiabilidade das Primitivas***

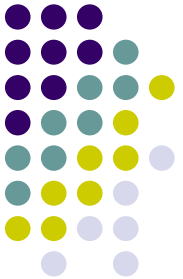


Resposta sendo usada como ACK



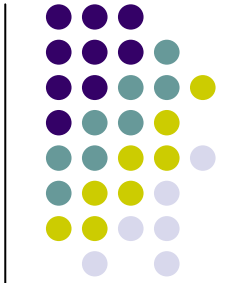
1: Requisição (Cliente para Servidor)
2: Resposta (Servidor para Cliente)
3: ACK (Kernel para Kernel)

Troca de Mensagem: Resumo



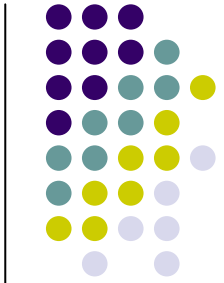
	Opção 1	**Opção 2**	**Opção 3**
Endereçamento	maq:processo	Aleatório	Servidor de Nomes
Bloqueio	Bloqueadas	Não Bloqueadas com cópia	Não Bloqueadas com interrupção
Bufferização	Não Bufferizadas (descarte)	Não Bufferizadas (armazenamento temporário)	Mailbox
Confiabilidade	Não Confiável	REQ-ACK-REP-ACK	REQ-REP-ACK

Troca de Mensagem: Implementação



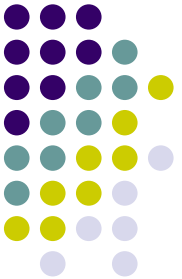
- Detalhes de como a passagem de mensagem é implementada depende das escolhas feitas durante o projeto;
- Algumas considerações:
  - Há um tamanho máximo do pacote transmitido pela rede de comunicação;
  - Mensagens maiores precisam ser divididas em múltiplos pacotes que são enviados separadamente;
  - Alguns dos pacotes podem ser perdidos ou chegar na ordem errada;
  - Solução: Atribuir a cada mensagem o número da mensagem e um número de seqüência.

Troca de Mensagem: Implementação



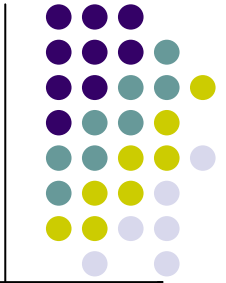
- O acknowledgment pode ser para cada pacote individual ou para a mensagem como um todo;
  - No primeiro caso na perda de mensagem, somente um pacote precisa ser retransmitido, mas na situação normal requer mais pacotes na rede de comunicação;
  - No segundo caso há a vantagem de menos pacotes na rede mas a desvantagem da recuperação no caso de perda de mensagem é mais complicada;
- Conclusão: A escolha de um dos dois métodos depende da taxa de perdas na rede.

Implementações do Modelo



- Apesar de simples, o modelo necessita de regras
  - Definição do protocolo de aplicação
  - Tanto cliente quanto servidor necessitam entender o que cada um transmite
  - A eficiência no modelo depende da eficiência do protocolo

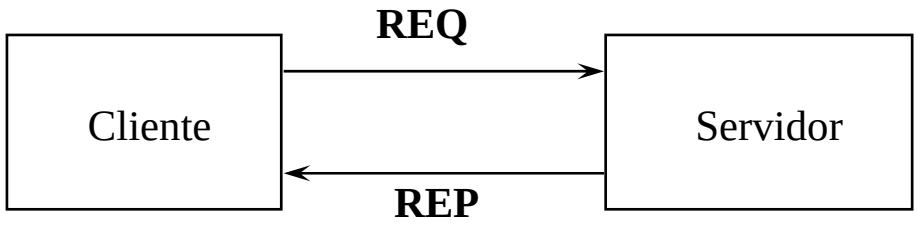
Protocolo Cliente/Servidor



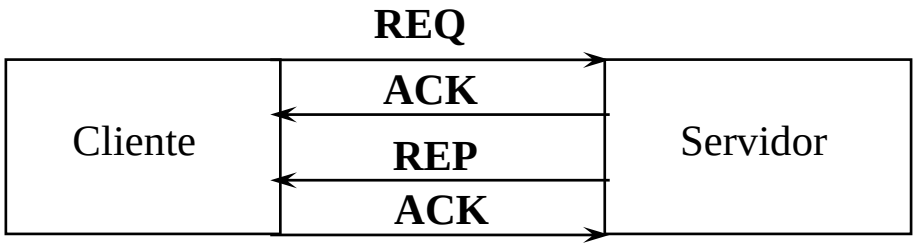
Código	Tipo	De	Para	Significado
REQ	Requisição	Cliente	Servidor	Solicitação de Serviço
REP	Resposta	Servidor	Cliente	Resposta à solicitação
ACK	Confirmação	Cliente Servidor	Servidor Cliente	A última mensagem chegou
AYA	Are You Alive?	Cliente	Servidor	Testa se o servidor está ativo
IAA	I Am Alive	Servidor	Cliente	Servidor ativo
TA	Try Again	Servidor	Cliente	Não posso atender
AU	Address Unknown	Servidor	Cliente	Não há processos com este endereço

Protocolo Cliente/Servidor

Protocolo Usado na Comunicação Cliente-Servidor (a)

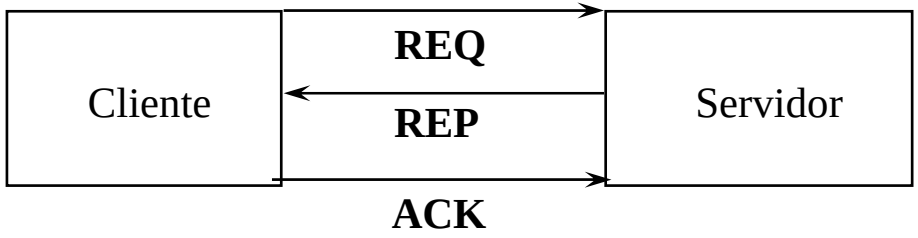


Protocolo Usado na Comunicação Cliente-Servidor (b)



Protocolo Cliente/Servidor

Protocolo Usado na Comunicação Cliente-Servidor (c)



Protocolo Usado na Comunicação Cliente-Servidor (c)