FSO
03 de Dezembro 2018

Processos cooperantes

- Sincronização
- Comunicação

  Exemplo: *Pipes* no UNIX

Bibliografia: OSTEP 5.4 (pipes)

Processos cooperantes

- **Os processos são independentes** o que se passa num processo não afecta nada o que se passa noutra
- **Processos independentes podem cooperar** Um programa constituído por vários processos
- Vantagens da cooperação entre processos
  - Partilha da informação
  - Aceleração das computações
  - Facilidade no desenvolvimento (Modularidade)
  - Natureza do problema a resolver

Os processos cooperantes têm de

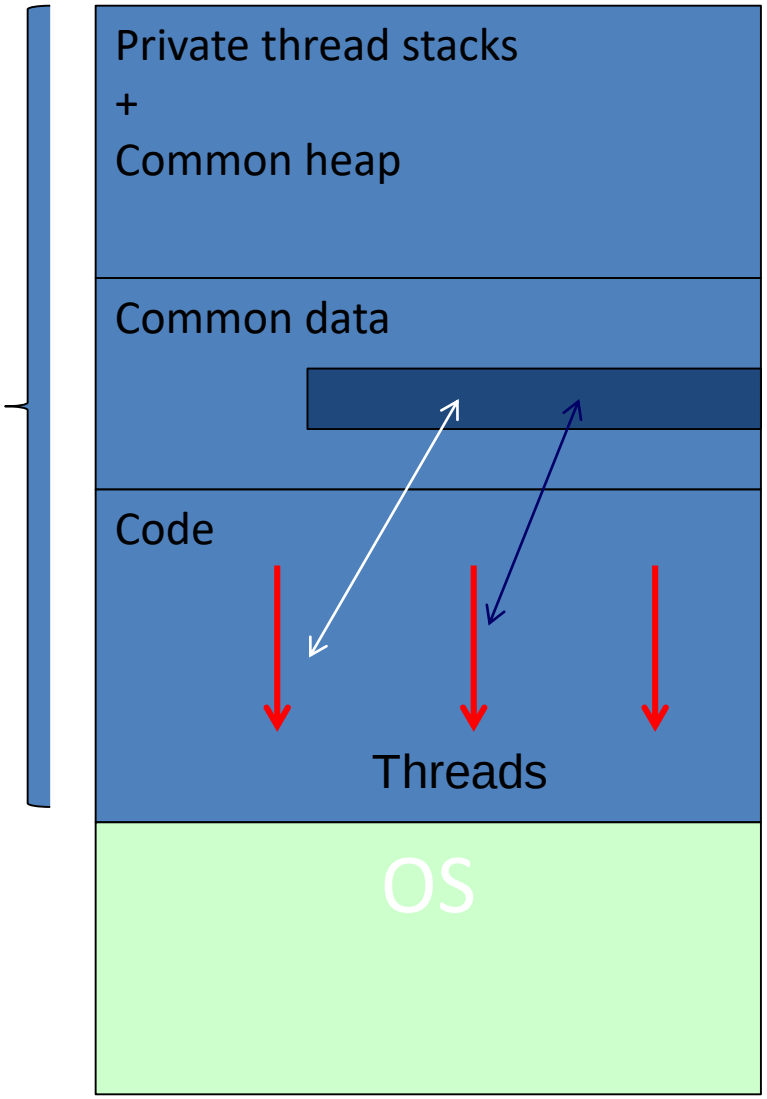
- **Transferir informação entre si:** transferir bytes entre processos. Exemplos: browser / servidor web
- **Sincronizar acções:** Garantir que a acção B no processo P2 só acontece depois da acção A no processo P1. Exemplo: garantir actulização correcta de variáveis comuns (mutexes)

Comunicação entre processos *Interprocess Communication* (IPC)

- Mecanimos que permitem aos processos
  - Sincronizar as suas acções
  - Transferir informação
  - Utilizando memória comum Exemplo: Nos Pthreads, processos que lêm e escrevem variáveis partilhadas
  - Sem partilha de memória
    - Exemplo 1: um processo UNIX criado por fork() não partilha memória com outros processos no sistema
    - Exemplo 2: um processo na máquina H1 e outro que executa na máquina H2

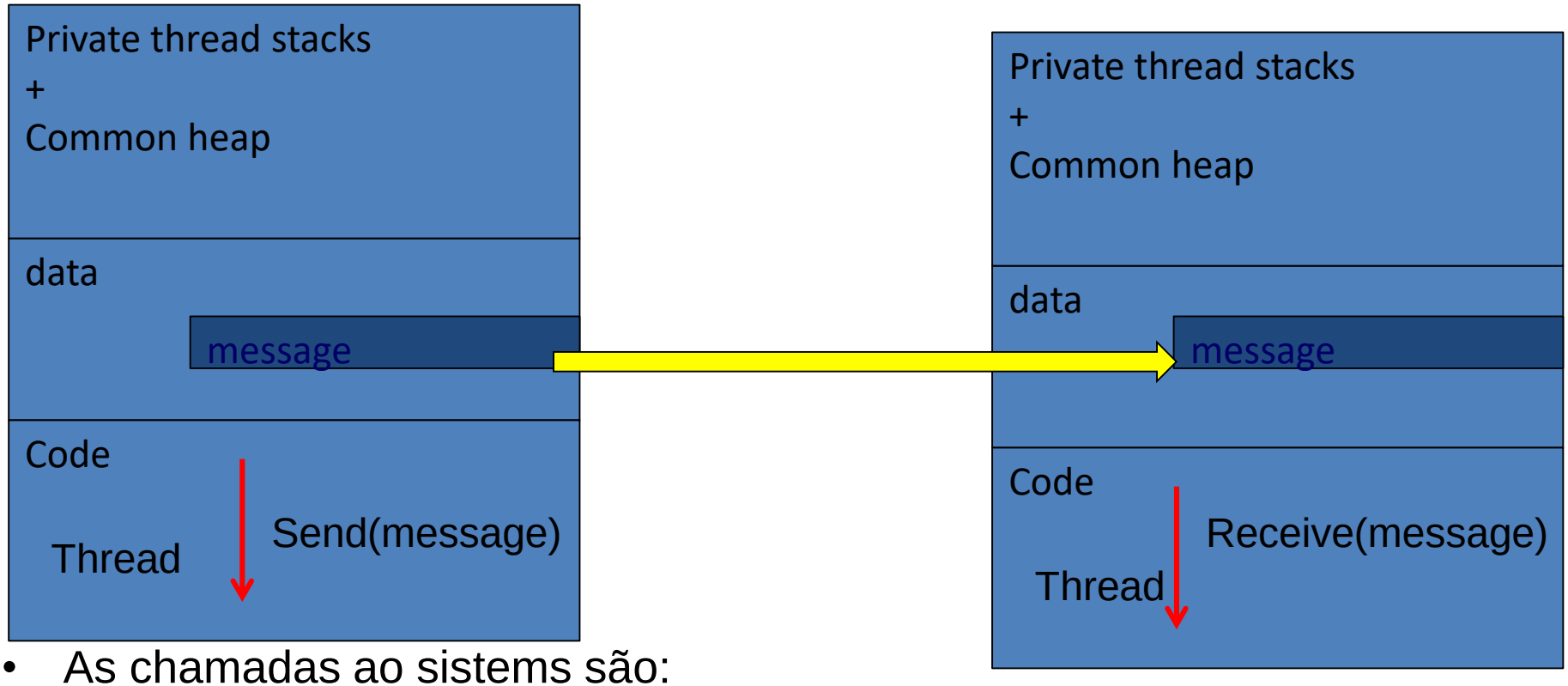
IPC com memória partilhada

Imagem do processo em RAM



- Threads comunicando através da leitura e escrita em variáveis
- Não é precisa a intervenção do SO (rápido)
- Ler e escrever em variáveis comuns tem de ser feita com regras (perigoso)

IPC sem memória partilhada: envio e recepção de mensagens

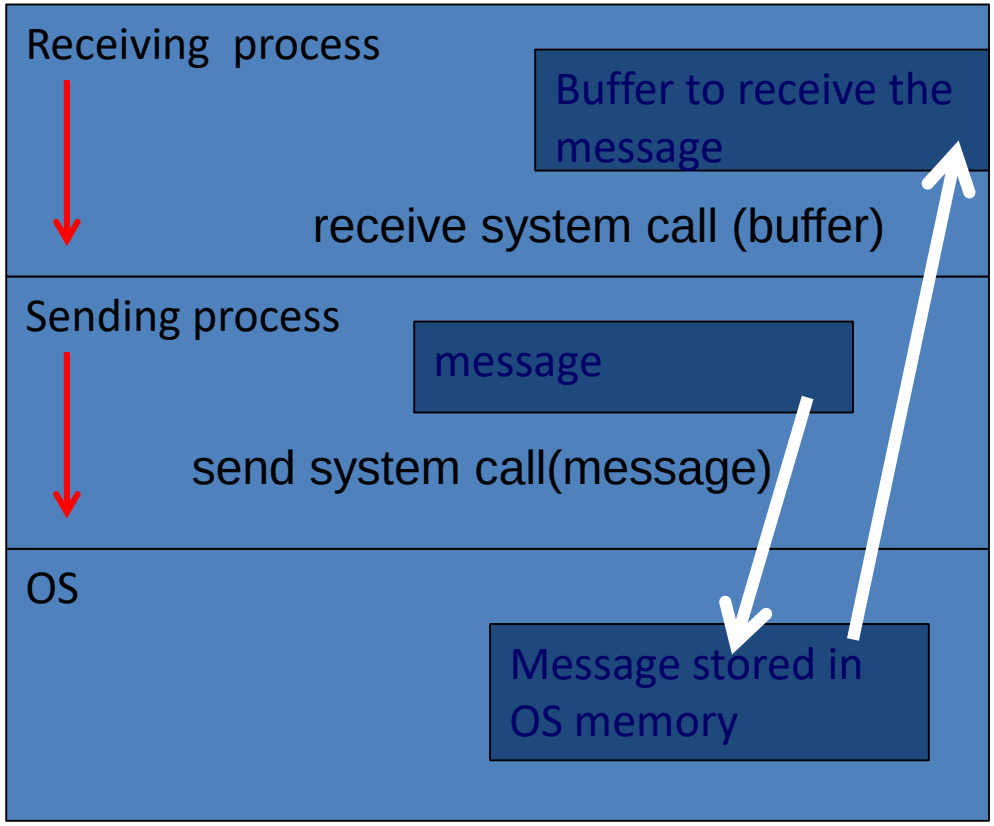


- As chamadas ao sistems são:
  - **send**(destino, *sequência de bytes*) – um conjunto de bytes que estão na memória do processo emissor é enviado para um destino
  - **receive**(*espaço em memória*) –bytes enviados são copiados para um espaço previamente reservado na memória do processo receptor.

Como especificar o destinatário

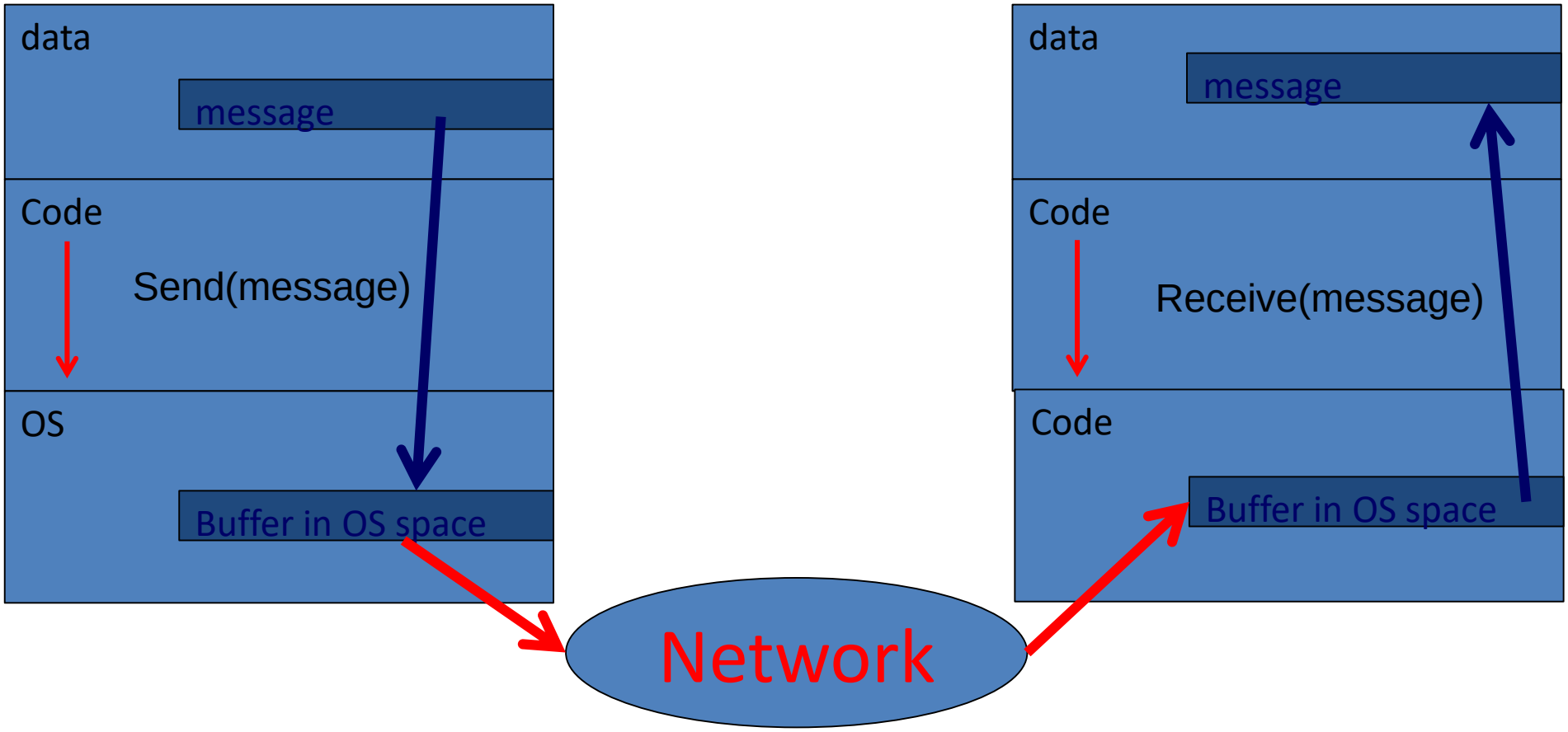
- Comunicação directa – o receptor da mensagem é especificado pelo identificador do processo pid
  - Os processos designam-se explicitamente:
    - **send** (*P, message*) – enviar uma mensagem ao processo P
    - **receive**(*Q, message*) – receber uma mensagem do processo Q
- Comunicação indirecta – As mensagens são enviadas para entidades externas aos processos(mailboxes or ports)
  - Cada mailbox / port tem um identificador distinto
    - **send**(*A, message*) – enviar mensagem para a mailbox /port A
    - **receive**(*B, message*) – receber mensagem da mailbox / port B

IPC sem memória partilhada precisa de suporte do SO



- Na mesma máquina, o SO tem de transferir os bytes, uma vez que um processo não pode ler / escrever na memória de outro processo

Comunicação entre processos em máquinas distintas



- O SO pode enviar e receber bytes através da rede
- O emissor tem de especificar: o endereço da máquina onde os bytes são entregues, e uma port / mailbox na máquina remota

Sincronização associada ao envio / recepção de mensagens

- **Recepção de mensagens** é usualmente **bloqueante**
  - O receptor espera até que haja uma mensagem disponível. O SO coloca o processo no estado Bloqueado (Waiting) até que a mensagem chegue
- **Envio de mensagens** é usualmente **não bloqueante**
  - O emissor copia os bytes a enviar para a memória do SO e continua

Pipes no UNIX (OSTEP Ch4, Sec 5)

- Redirecção de Input / Output

  wc –l xpto.c > count.txt redirige o output de *wc* para o ficheiro *count.txt*

- **Combinação de comandos**
  - **ls –al | sort** o output do 1º comando é o input of segundo comando

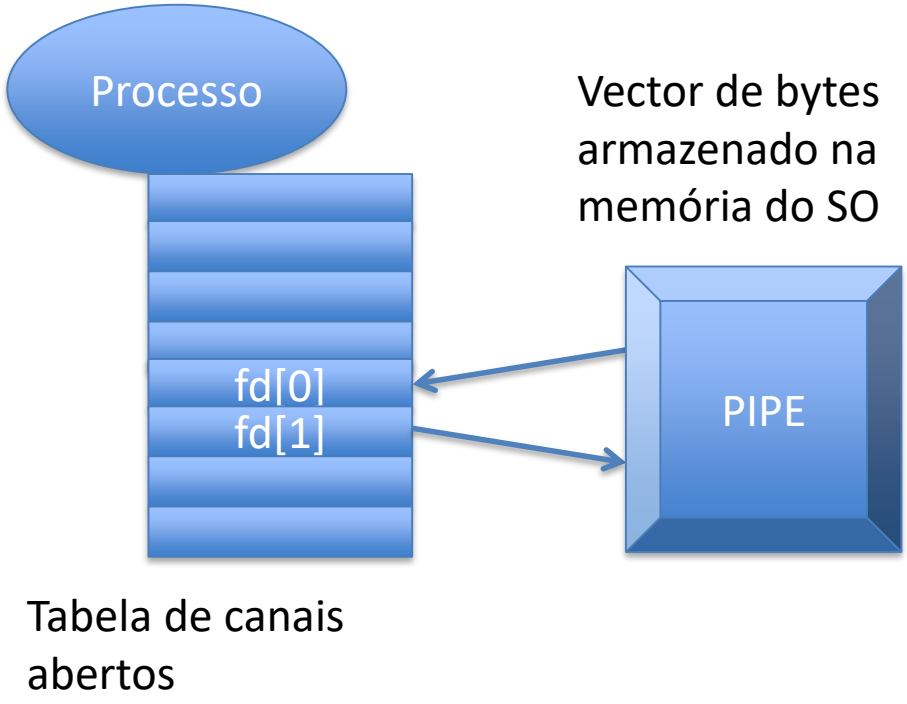
Chamada ao sistema *pipe*

int pipe(int fd[2]) cria um par de canais

- fd[0] for reading
- fd[1] for writing

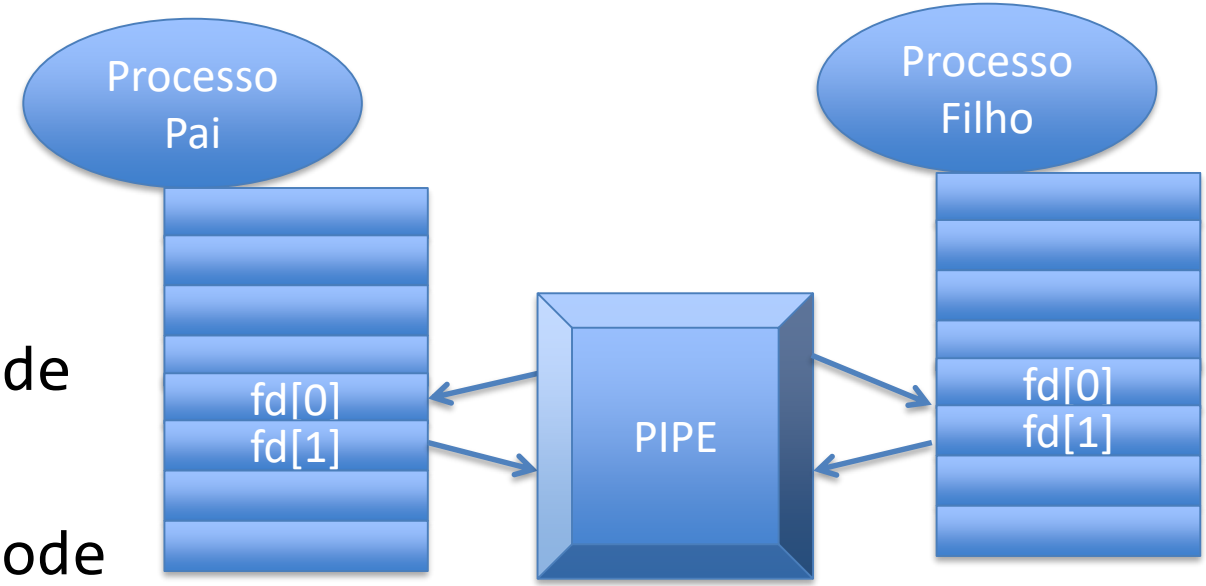
returns 0 on success, -1 on error

```
#include <stdio.h>
#include <unistd.h>
#include<sys/types.h>
main(){
   int fd[2];
   pipe(fd);
      . . .
```



Pipes e fork( )

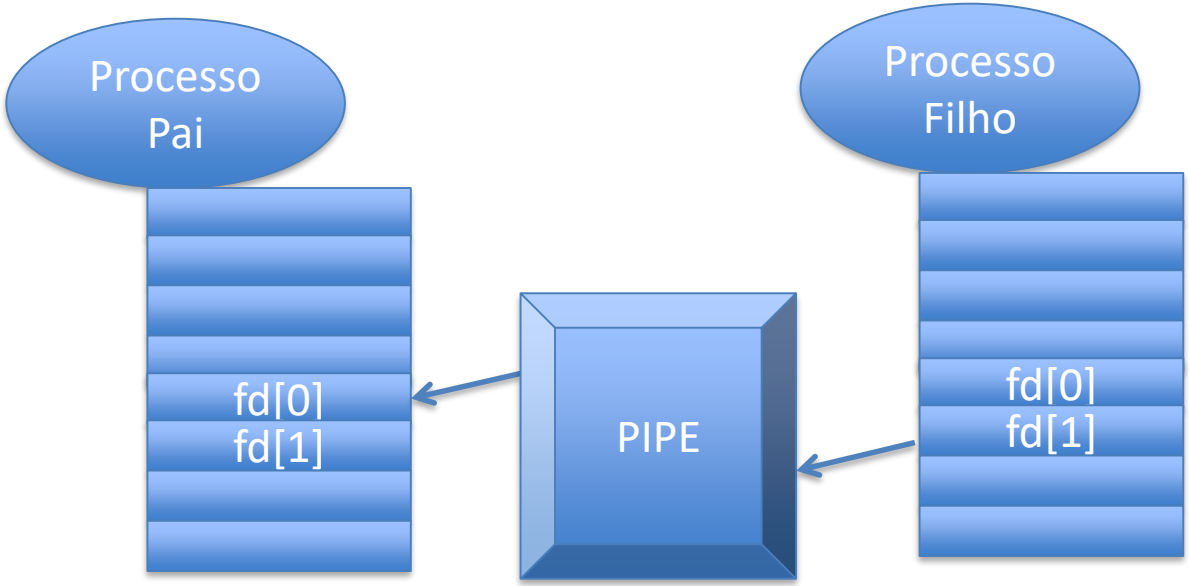
```
#include <stdio.h>
#include <unistd.h>
#include<sys/types.h>
main(){
   int fd[2];
   pipe(fd);
      p = fork();
      if( p==0){
         // child code
      } else{
         // parent code
      }
```



Pai e filho comunicando através de um pipe [1]

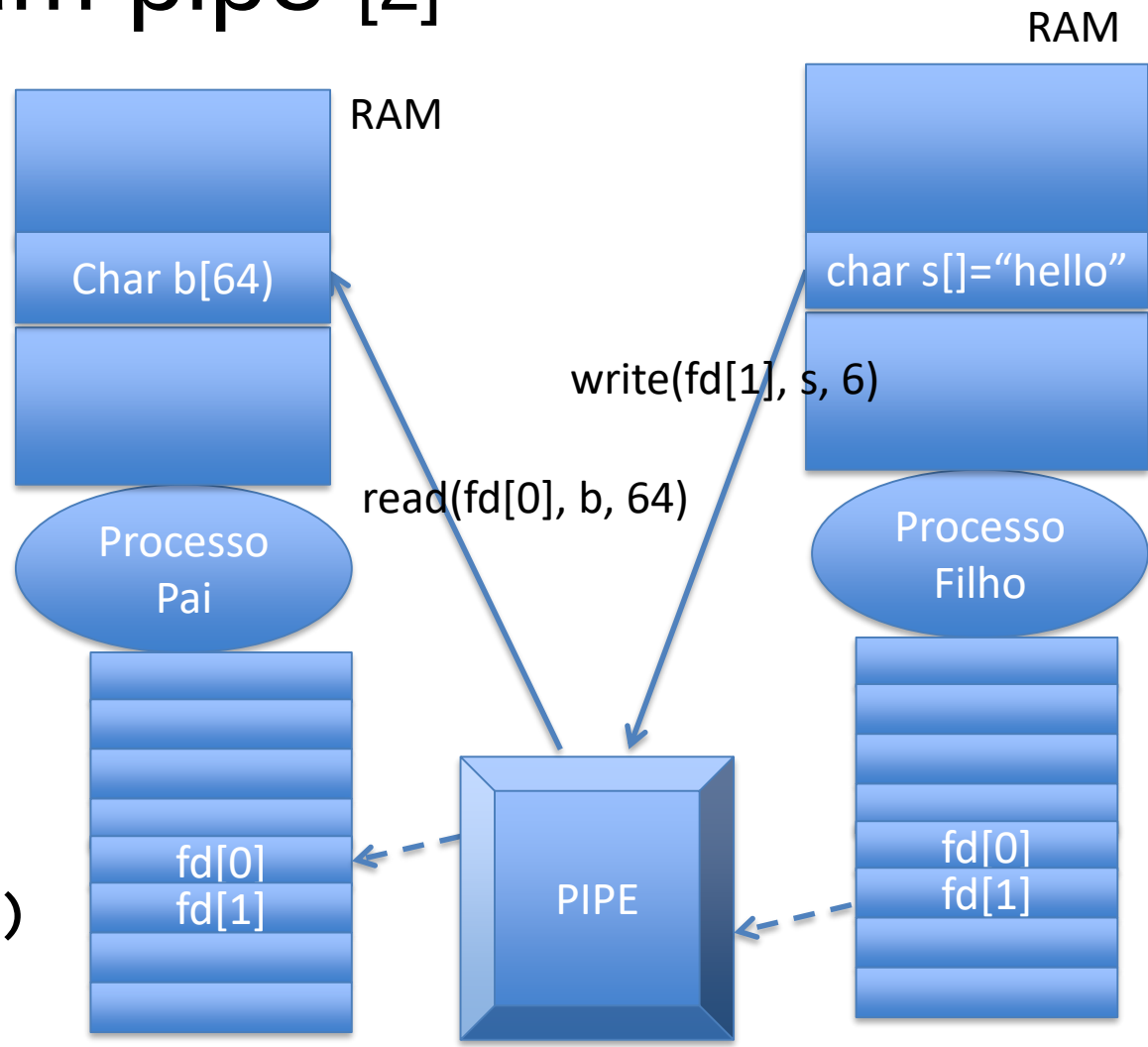
```
.. .
main(){
    int fd[2];
    pipe(fd);
      p = fork();
      if( p==0){
         // child code
         close(fd[0]);
        // child closes
        //input channel
         ...
      } else{
         // parent code
         close(fd[1]);
       // parent closes
       //output channel

      }
```



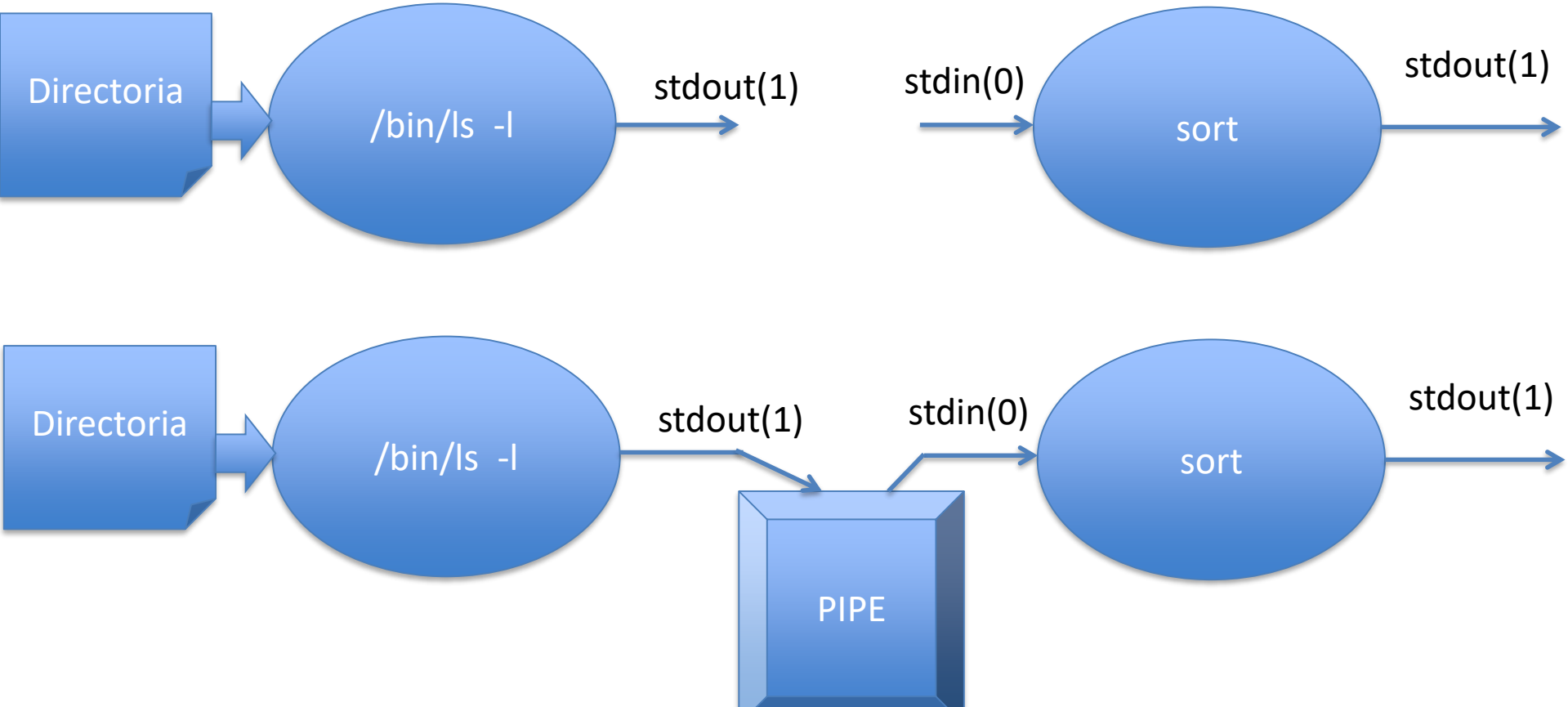
Pai e filho comunicando através de um pipe [2]

```
main(){
      char s[]="Hello\n";
      char b[64];
   . . .
     if( p==0){
        // child code
        close(fd[0]);
        write( fd[1], s,
          strlen(s)+1);
     } else{
        // parent code
        close(fd[1]);
        read( fd[0],b, 64)
     }
```



Uso de *Pipes* para combinar processos

- **Combinação de comandos**
  - **ls –al | sort** o output do 1º comando é o input of segundo comando



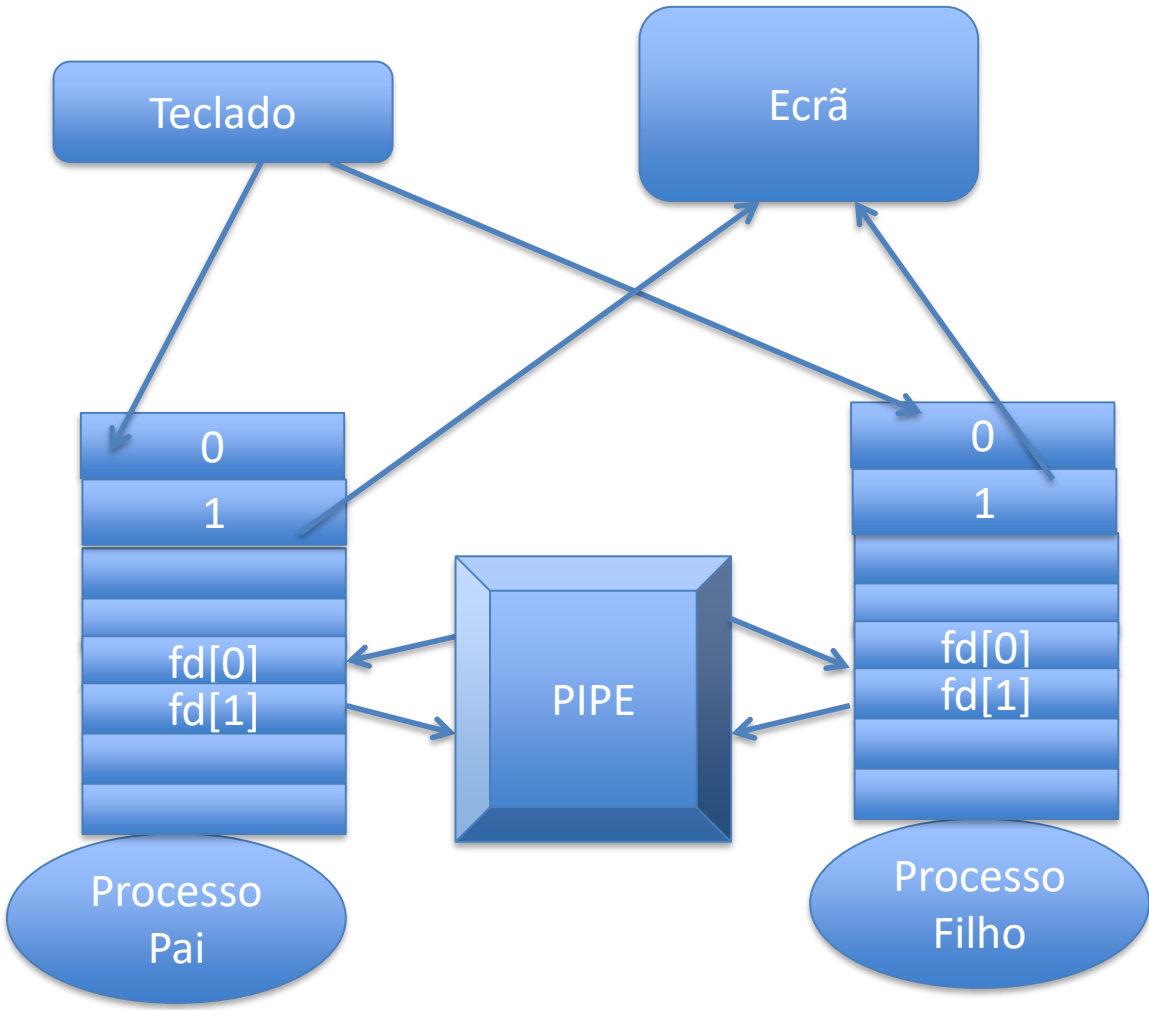
Chamada ao sistema Dup

int dup(int existingFd) duplica o canal *existingFd* retornado o nº do novo canal; retorna -1 se há erro. O novo canal tem a menor entrada livre na tabela de canais

```
int fd[2];
pipe(fd);
p = fork(); // pipe exists in parent and child
if(p = 0)
  close(0); // closes stdin
  dup(fd[0]); // input part of pipe is channel 0
  execvp( ... ); // inherits open channels
```

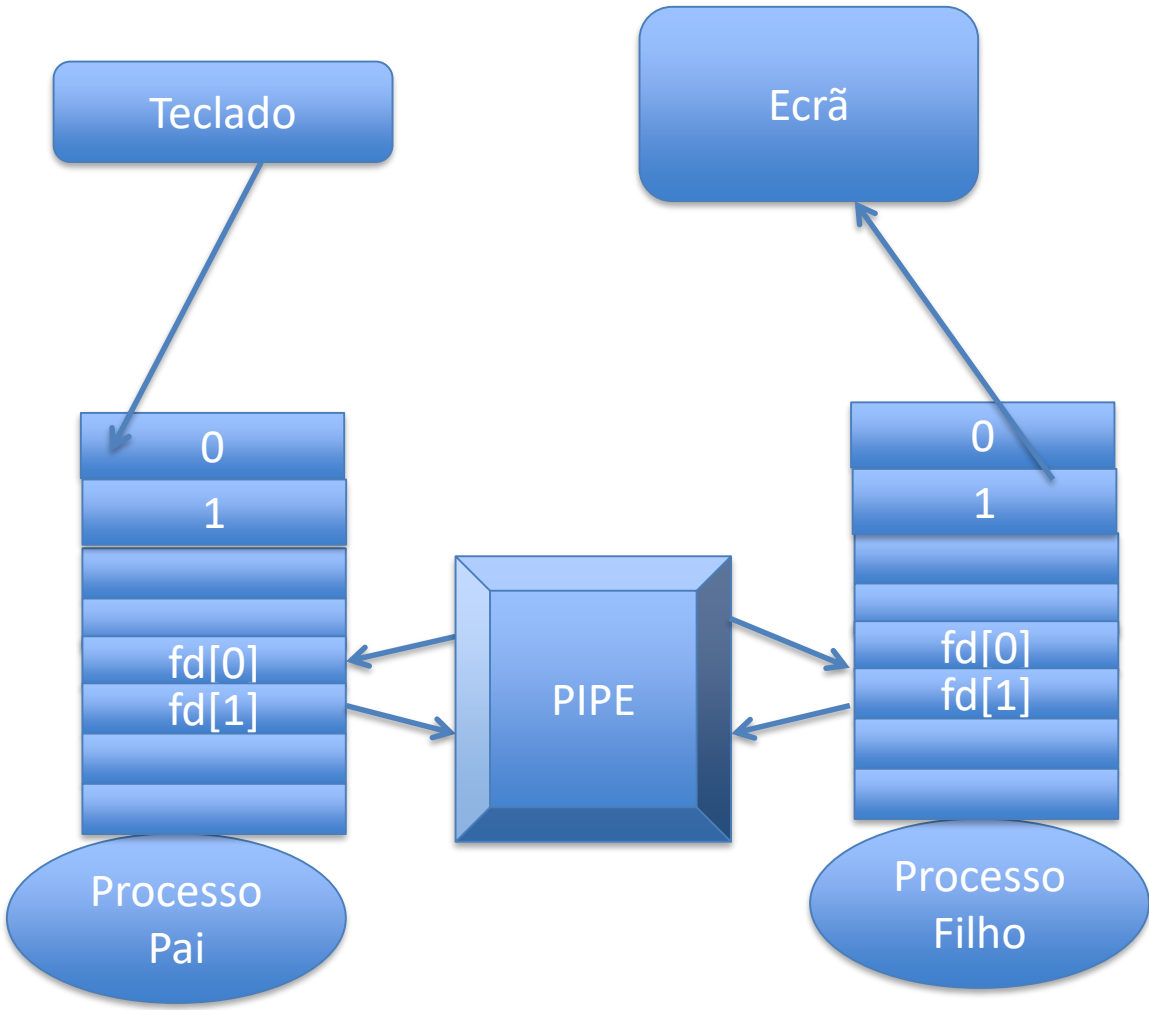
Dup system call [2]

```
int fd[2];
pipe(fd);
p = fork();
if(p != 0){
          . . .
}else{
          . . .
```



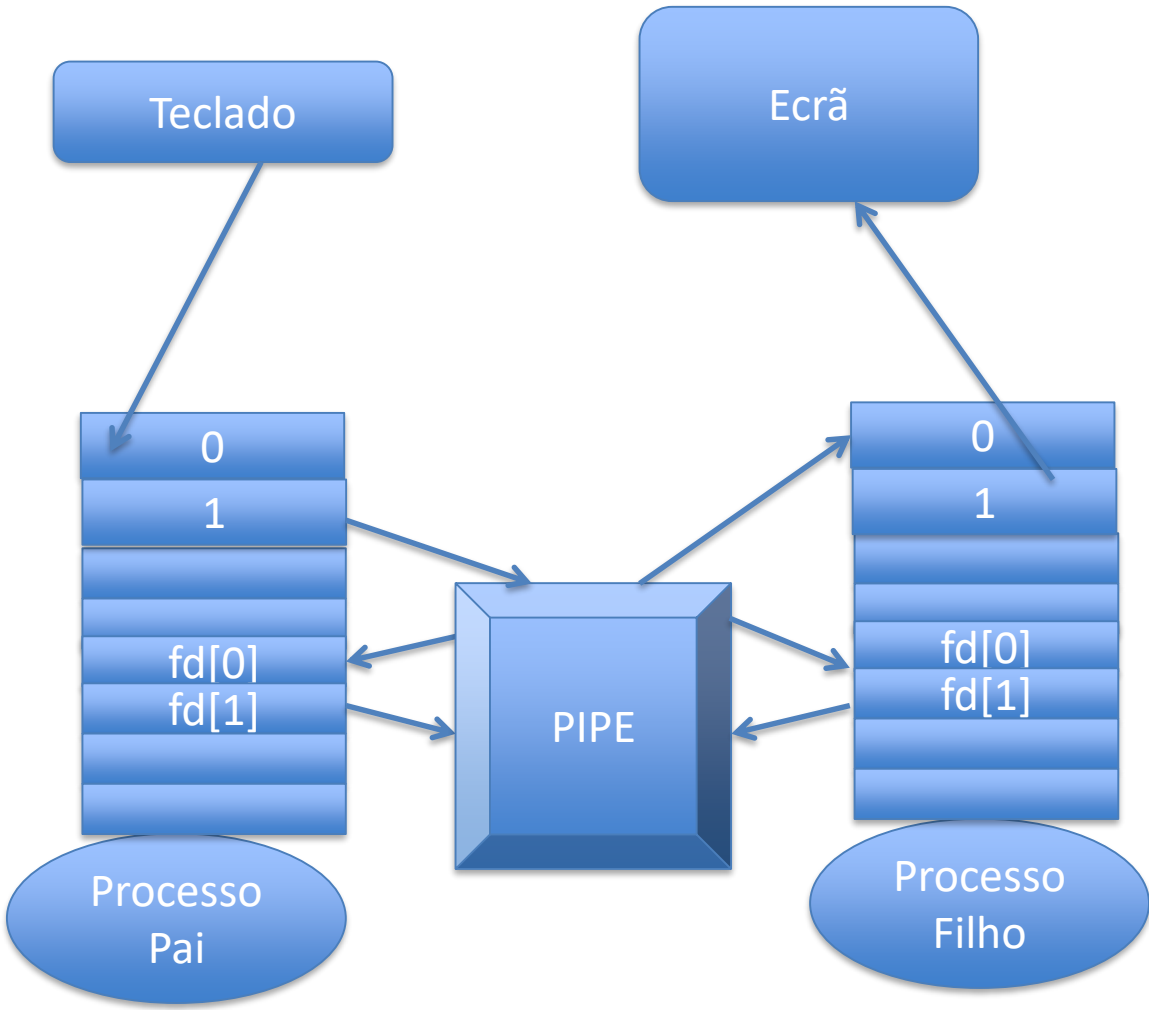
Dup system call [2]

```
int fd[2];
pipe(fd);
p = fork();
if(p != 0){
    close(1);
          . . .
}else{
     close(0);
          . . .
```

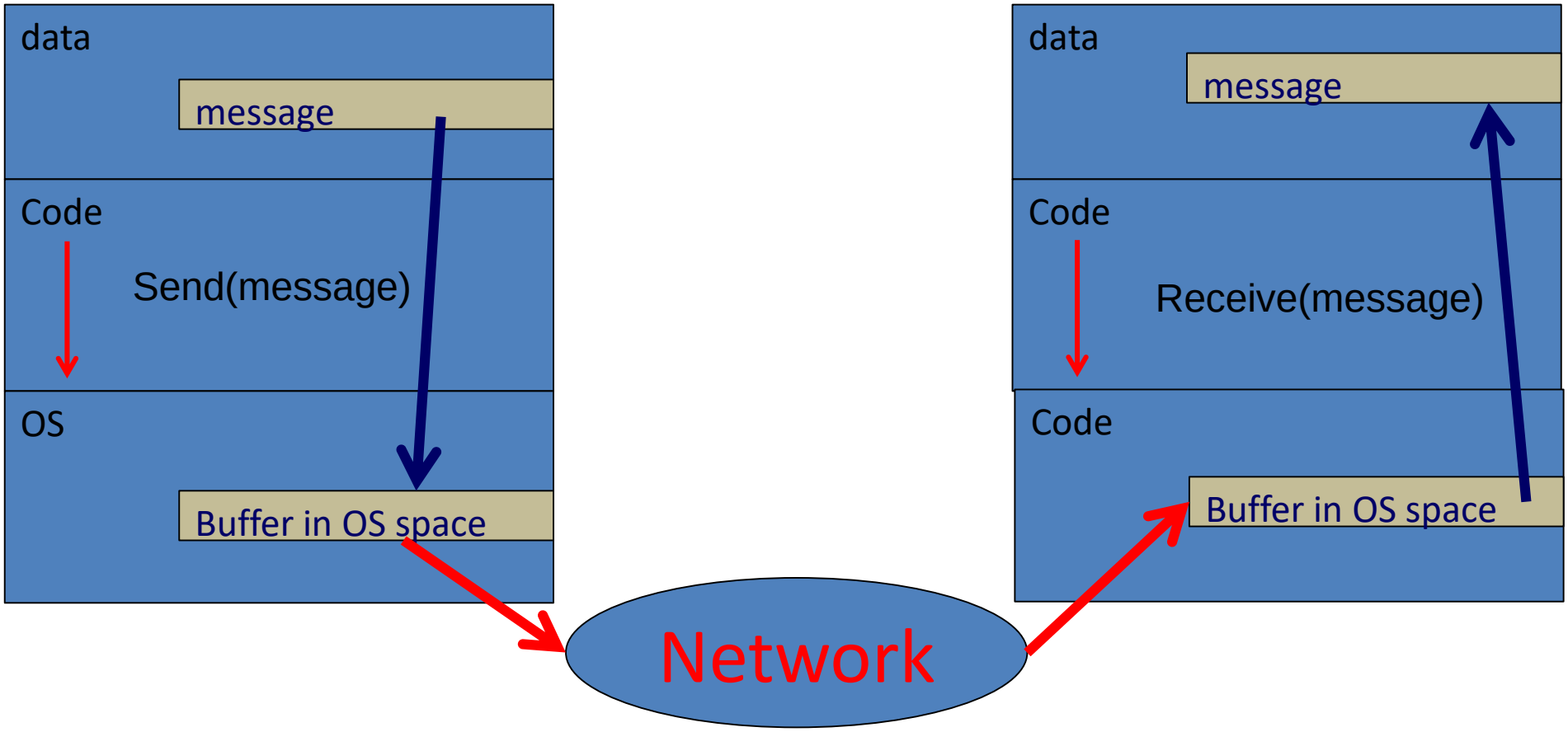


Dup system call [2]

```
int fd[2];
pipe(fd);
p = fork();
if(p != 0){
    close(1);
    dup(fd[1]);
          . . .
}else{
     close(0);
     dup(fd[0]);
          . . .
```



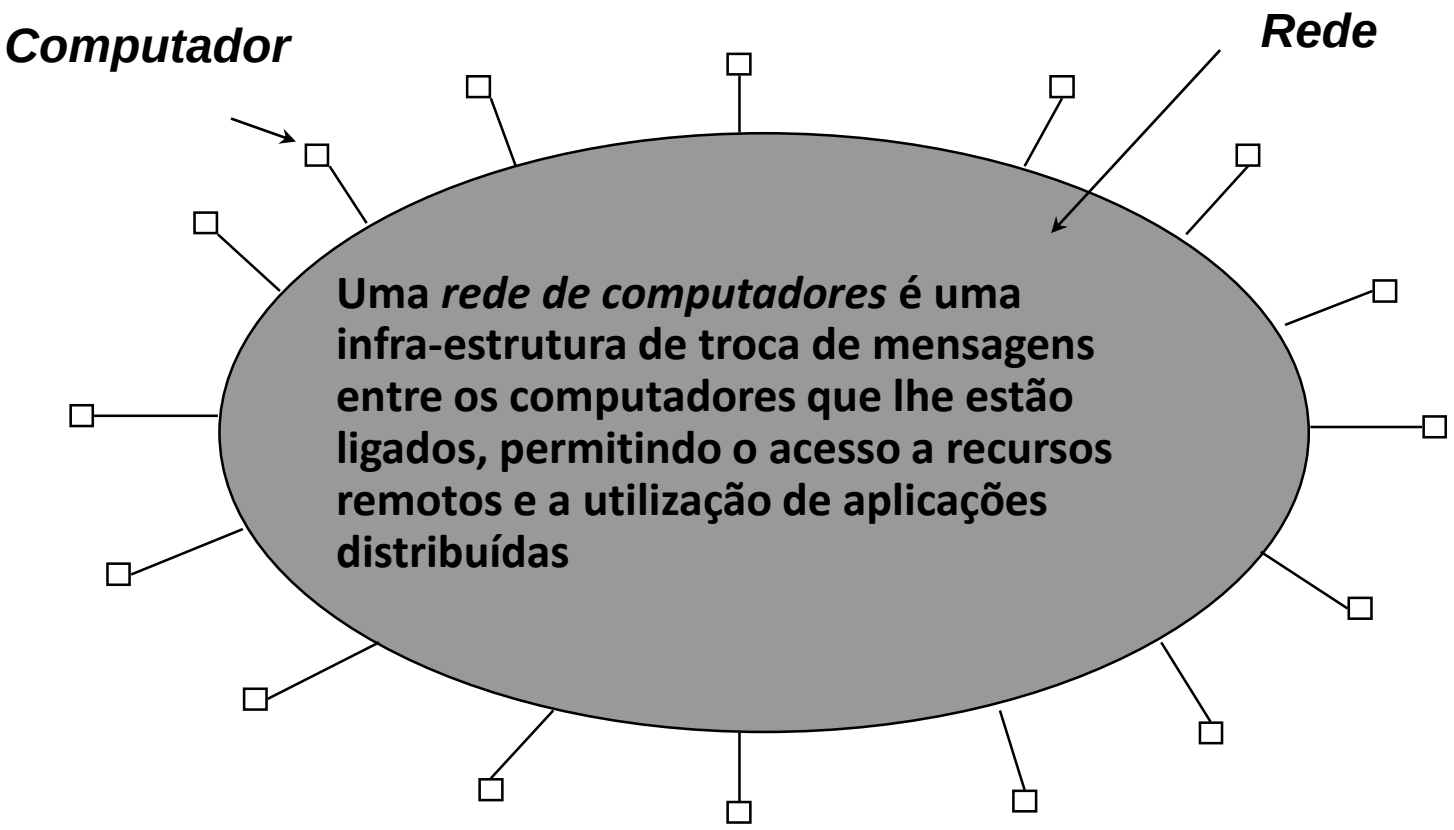
Comunicação entre processos em máquinas distintas



- O SO pode enviar e receber bytes através da rede
- O emissor tem de especificar: o endereço da máquina onde os bytes são entregues, e uma port / mailbox na máquina remota

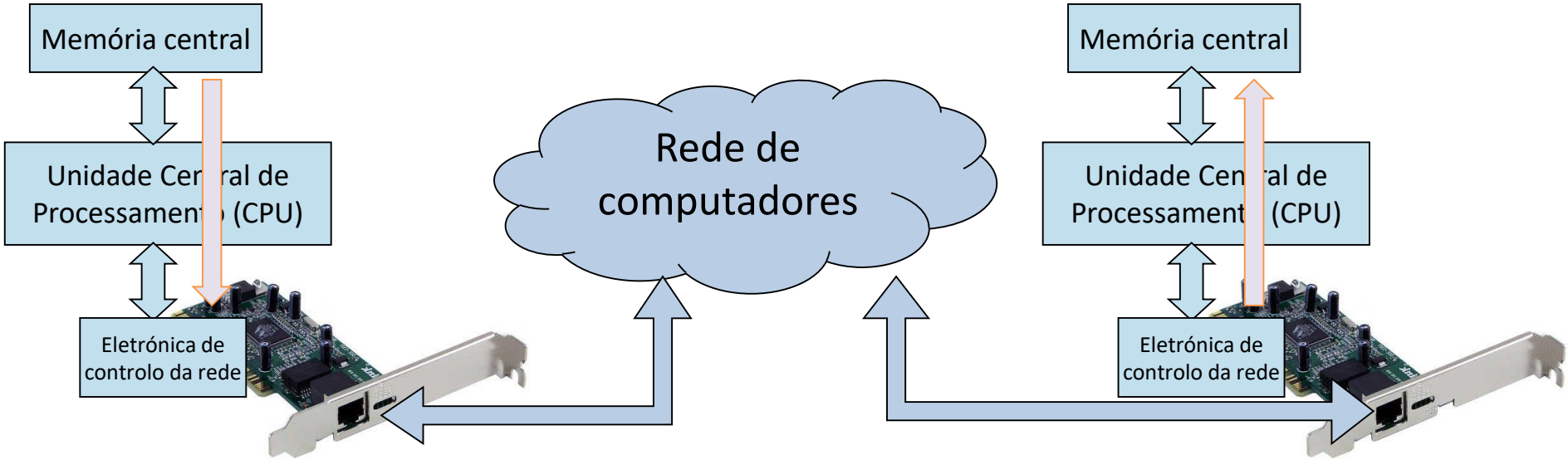
Rede de computadores

Um conjunto de computadores que estão interligados com o objetivo de trocar informação e partilhar recursos.



Numa transferência intervêm dois computadores e a rede

- Dois computadores
  - Nó emissor: produz uma sequência de bytes (mensagem).
  - Nó recetor ou destinatário: recebe essa sequência de bytes.
- Rede
  - Meios de interligação: cabos, atmosfera, …
  - Equipamentos dedicados a assegurar que a mensagem é transportada do nó emissor ao nó recetor.



Internet e o encaminhamento de pacotes

- A Internet é uma interligação de redes locais de acordo com normas próprias.
- Na Internet todos os computadores (ou nós) têm um endereço único (normalmente um número com 32 bits) chamado endereço IP (Internet Protocol).
- Quando uma computador emite um pacote destinado a outro, noutra LAN, entrega-o ao *router* mais próximo (que está na sua rede local).
- Os *routers* propagam o pacote até este chegar ao *router* da rede local do computador de destino, que entrega o pacote ao destinatário.

Endereços dos nós

Endereço IP do nó (numérico / *machine friendly*)

- Constituído por 4 bytes (32 bits)
  - representado por 4 números entre 0 e 255, separados por pontos.
- Identifica univocamente o computador na Internet.
- Usado no encaminhamento das mensagens na rede.
- **Exemplo:** 193.136.122.33

Endereço simbólico do nó (cadeia de caracteres / *user friendly*)

- São aqueles nomes que o utilizador usa.
- **Exemplo:** www.google.com

Nome do nó
asc.di.fct.unl.pt

Domain Name System (DNS)

- Serviço de conversão de nomes “user friendly” em nomes “machine friendly”.
  - Este serviço reside num nó da LAN (ou do fornecedor de Internet).
- Invocado quando um nó tenta obter o endereço IP correspondente a um dado nome simbólico.

TLDs do Domain Name System [1]

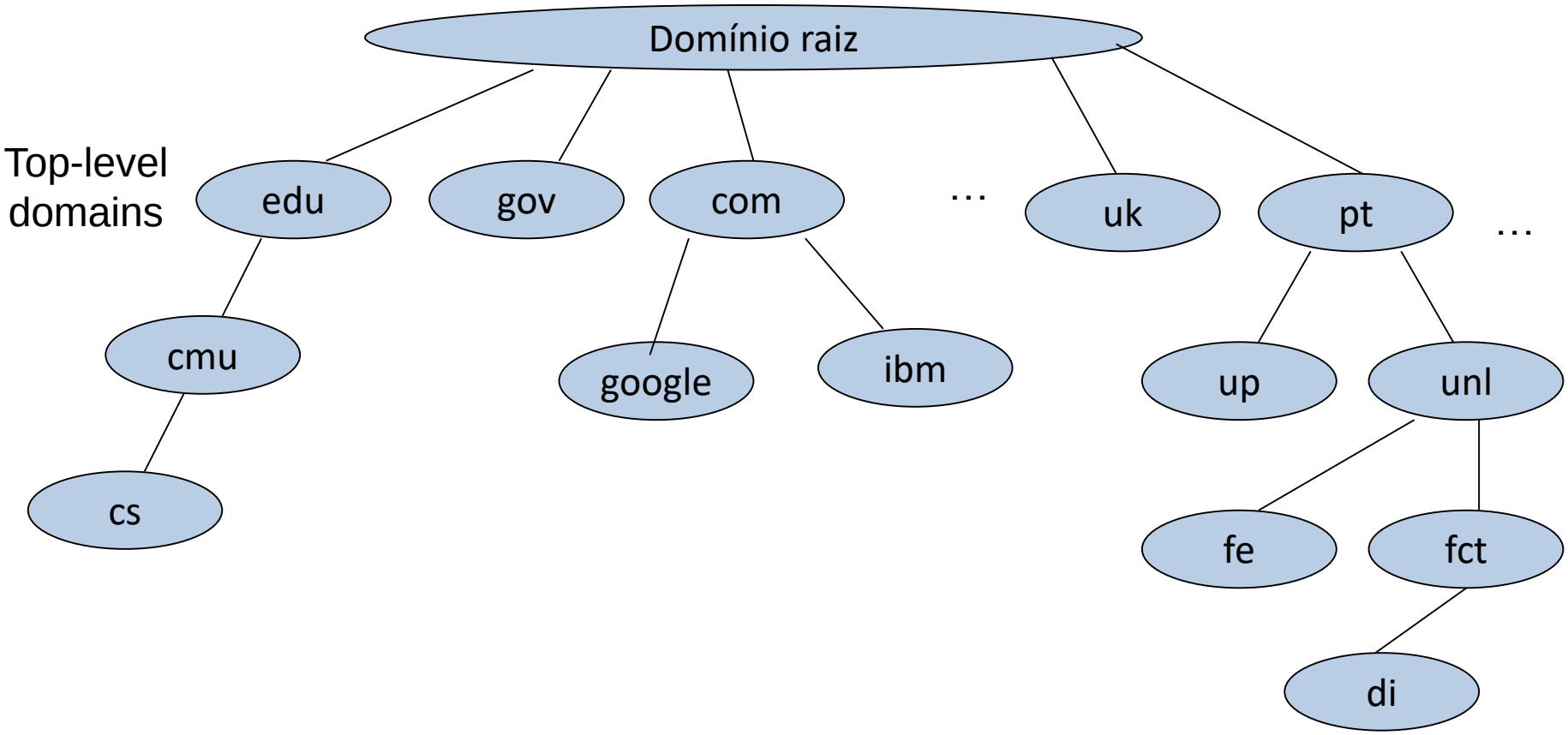
Top-Level Domain	Utilização
biz	Negócios
com	Comercial (EUA)
edu	Educação (EUA)
info	Informação
gov	Governo (EUA)
mil	Militar (EUA)
net	Rede
org	Sem fins lucrativos

TLDs do Domain Name System [2]

As organizações com sede fora dos Estados Unidos usam um top-level domain que é o código do país (com 2 letras).

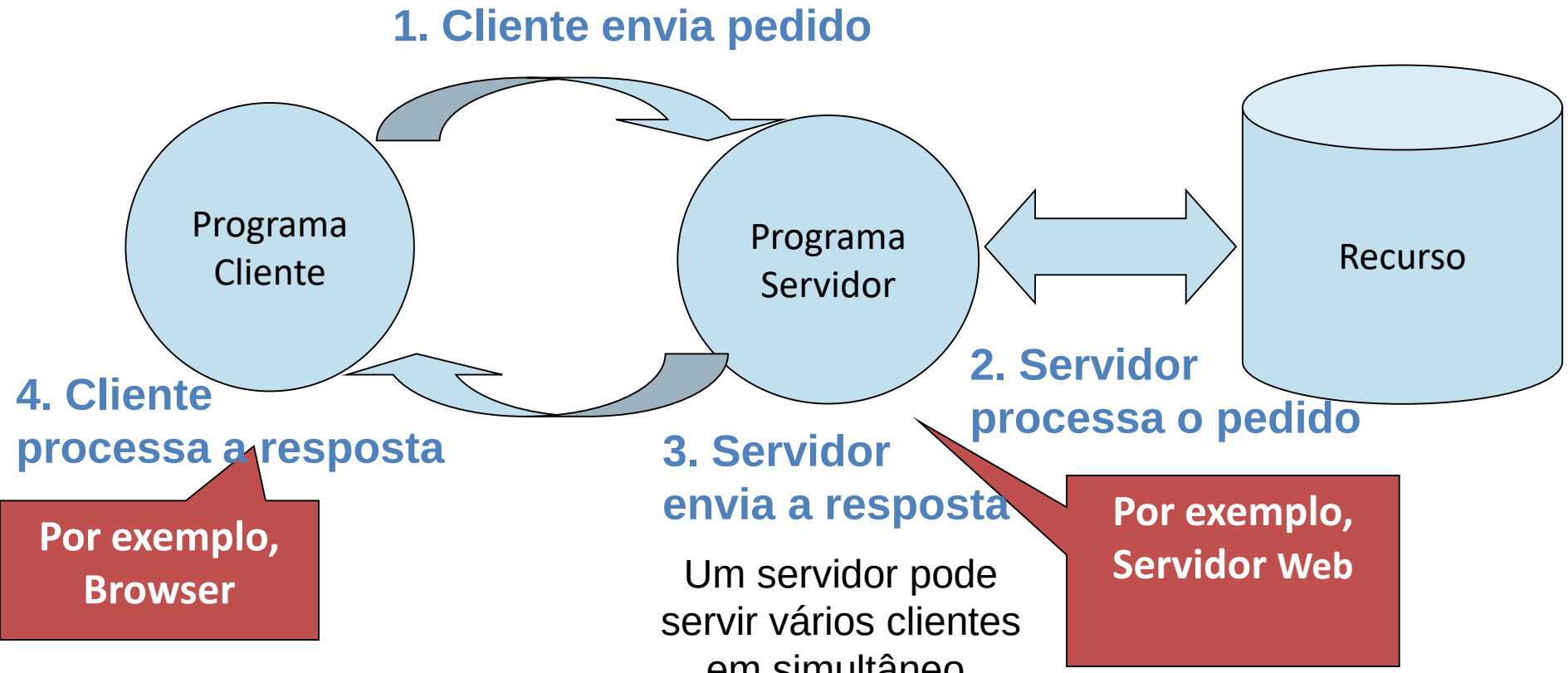
País	TLD
pt	Portugal
uk	Reino Unido
es	Espanha
fr	França
nz	Nova Zelândia
cn	China
…	…

Árvore de domínios do DNS



Relação cliente-servidor

- Na maioria das situações, as aplicações que usam a rede são constituídas por dois programas.



Exemplos de servidores

- Servidor Web
  - Recursos: ficheiros e programas.
  - Serviços: obter ficheiros e gerar páginas dinâmicas executando programas a pedido do cliente.
- Servidor de Mail
  - Recurso: ficheiros com mensagens.
  - Serviços: armazena mensagens ou encaminha-as para outros servidores de Mail.
- Servidor Dropbox
  - Recursos: ficheiros.
  - Serviços: leitura e escrita de ficheiros.

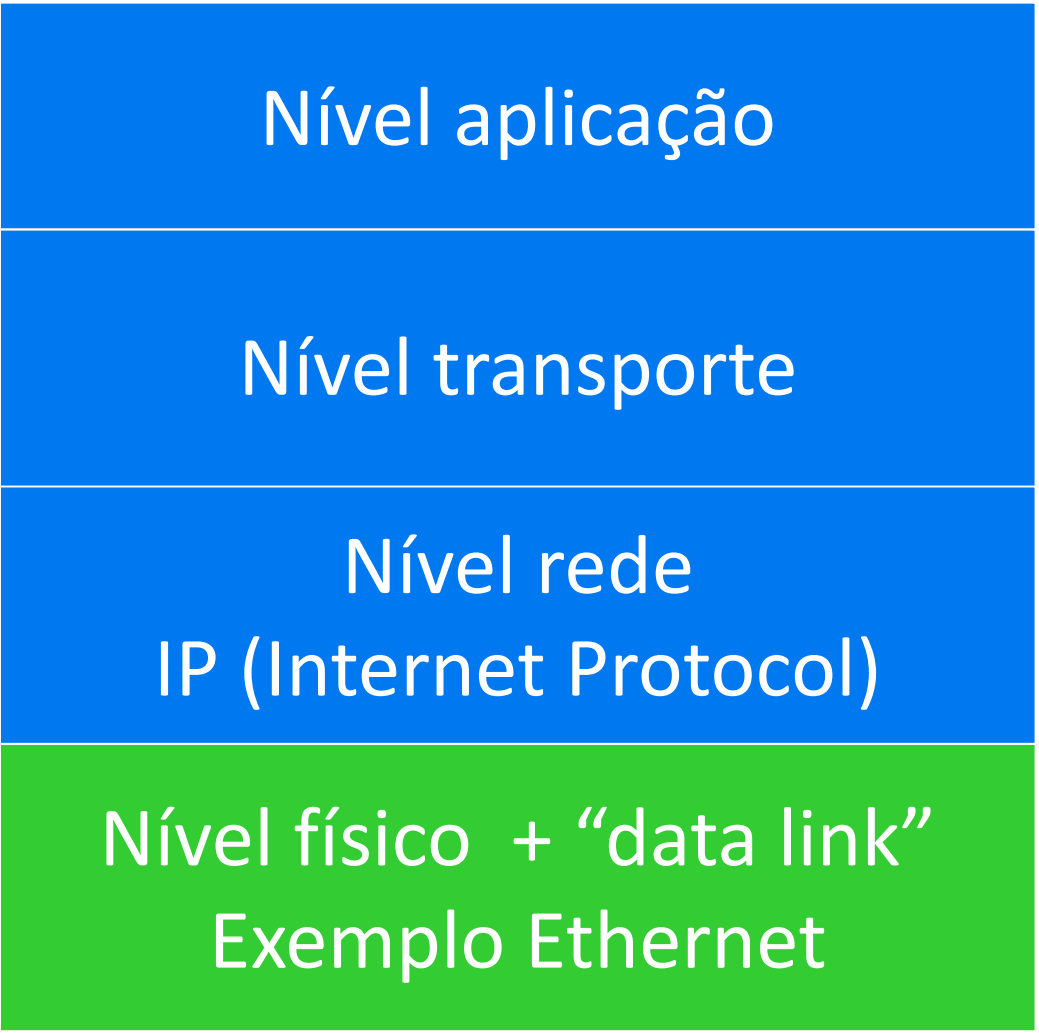
Exemplos de clientes

- Cliente Web
  - Firefox, Internet Explorer, Safari, etc.
- Ciente de Mail
  - eM Client, Mozilla Thunderbird, Opera, Mail, Live Mail, etc.

Protocolos de aplicação

- Conjunto de regras que definem como é que os bytes que circulam na rede são interpretados.
  - O cliente e o servidor têm de acordar previamente os formatos do pedido e da resposta;
  - Existem normas para imensos serviços na Internet.
  - **Exemplos:**
    - Protocolos Internet para consultar o DNS e obter o endereço IP para um nome;
    - Protocolos de aplicações de email (SMTP) ou de Web (HTTP).

A pilha de protocolos TCP/IP [1]



HTTP, FTP, SMTP, …
Cliente/servidor

UDP (Unreliable Datagram Protocol)
TCP (Transmission Contol Protocol)

Define a forma de designar os nós e o mecanismo de entrega de pacotes; não fiável

O nível transporte

- A camada de *Transporte*:
  - Disponibiliza todos serviços necessários à comunicação "*end-to-end*" (e.g. entre dois *hosts / end-points* na periferia da rede) incluindo a noção de *porta (ver à frente)*
  - Se necessário, divide a mensagem que recebe do nível da aplicação (e.g. uma página html) no número de pacotes IP necessários.

Nível transporte: o protocolo TCP [1]

- Objectivo: transferência de dados entre dois computadores de forma a simplificar o trabalho dos programadores de aplicações
  - Modelo oferecido é de um canal de bytes; não há perda de bytes nem chegadas fora de ordem
- Numa primeira fase é estabelecido um "*handshake*" (aperto de mão), isto é, uma ligação ou conexão entre os dois computadores
- Esta fase termina com o estabelecimento da conexão que fica conhecida e é monitorizada pelos dois computadores
- TCP é usado por: HTTP, FTP, SMTP, Telnet, …

Nível transporte: o protocolo TCP [2]

- **TCP – Transmission Control Protocol** [RFC 793] é o serviço de “transmissão fiável de dados, orientado à conexão” usado na Internet
  - Transferência de sequências de bytes de forma ordenada e fiável: *acknowledgements* (confirmações de recepção) e retransmissões se necessário
  - Controlo de fluxo: o emissor não afoga o receptor
  - Controlo da saturação: o emissor abranda o ritmo de emissão quando a rede está saturada

Nível transporte: o protocolo TCP [2]

- **TCP – Transmission Control Protocol** [RFC 793] é o serviço de “transmissão fiável de dados, orientado à conexão” usado na Internet
  - Transferência de sequências de bytes de forma ordenada e fiável: *acknowledgements* (confirmações de recepção) e retransmissões se necessário
  - Controlo de fluxo: o emissor não afoga o receptor
  - Controlo da saturação: o emissor abranda o ritmo de emissão quando a rede está saturada

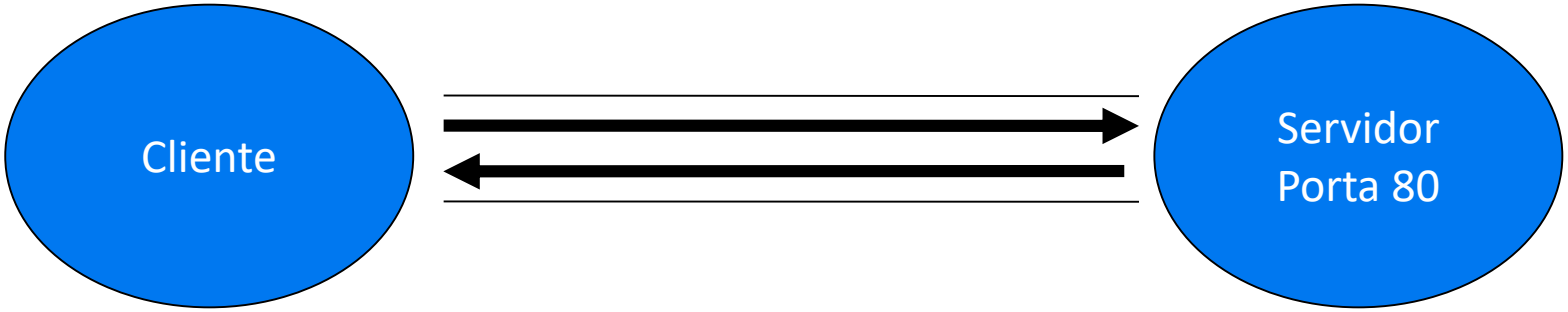
A noção de conexão [1]

- Clientes e servidores que usam o protocolo TCP comunicam enviando e recebendo sequências (streams) de bytes através de canais (connections):
  - É Ponto-a-Ponto porque dois programas são interligados
  - É Full-Duplex, porque os dados podem fluir nos dois sentidos (cliente / servidor e servidor / cliente)
  - É fiável porque (a menos de algo catastrófico) a sequência de bytes enviada pelo emissor é recebido pelo receptor, sem perdas e pela mesma ordem da emissão

A noção de conexão [2]

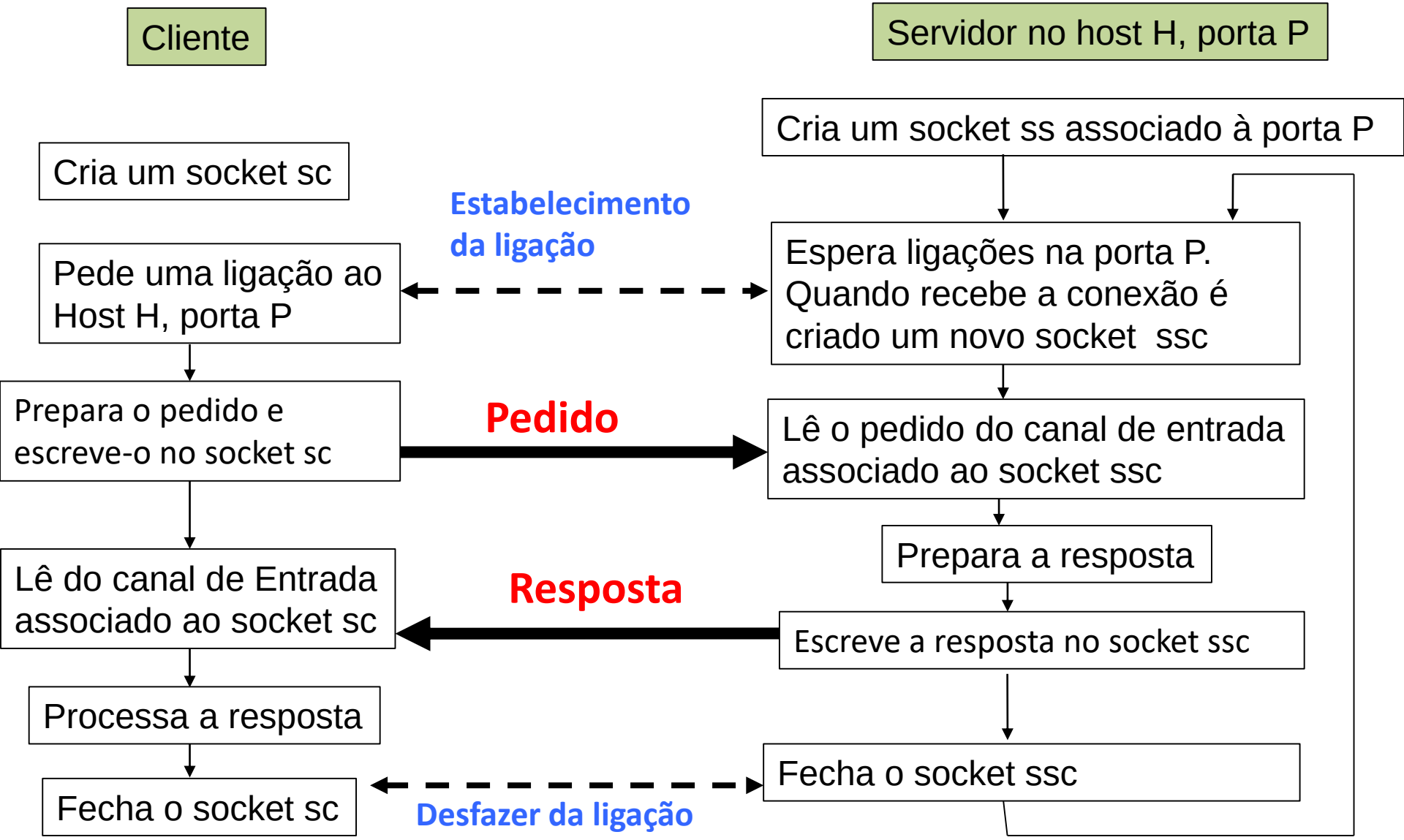
Endereço do nó do cliente: 193.136.122.33

Endereço do nó do servidor: 208.216.181.15



Porta do servidor: 80

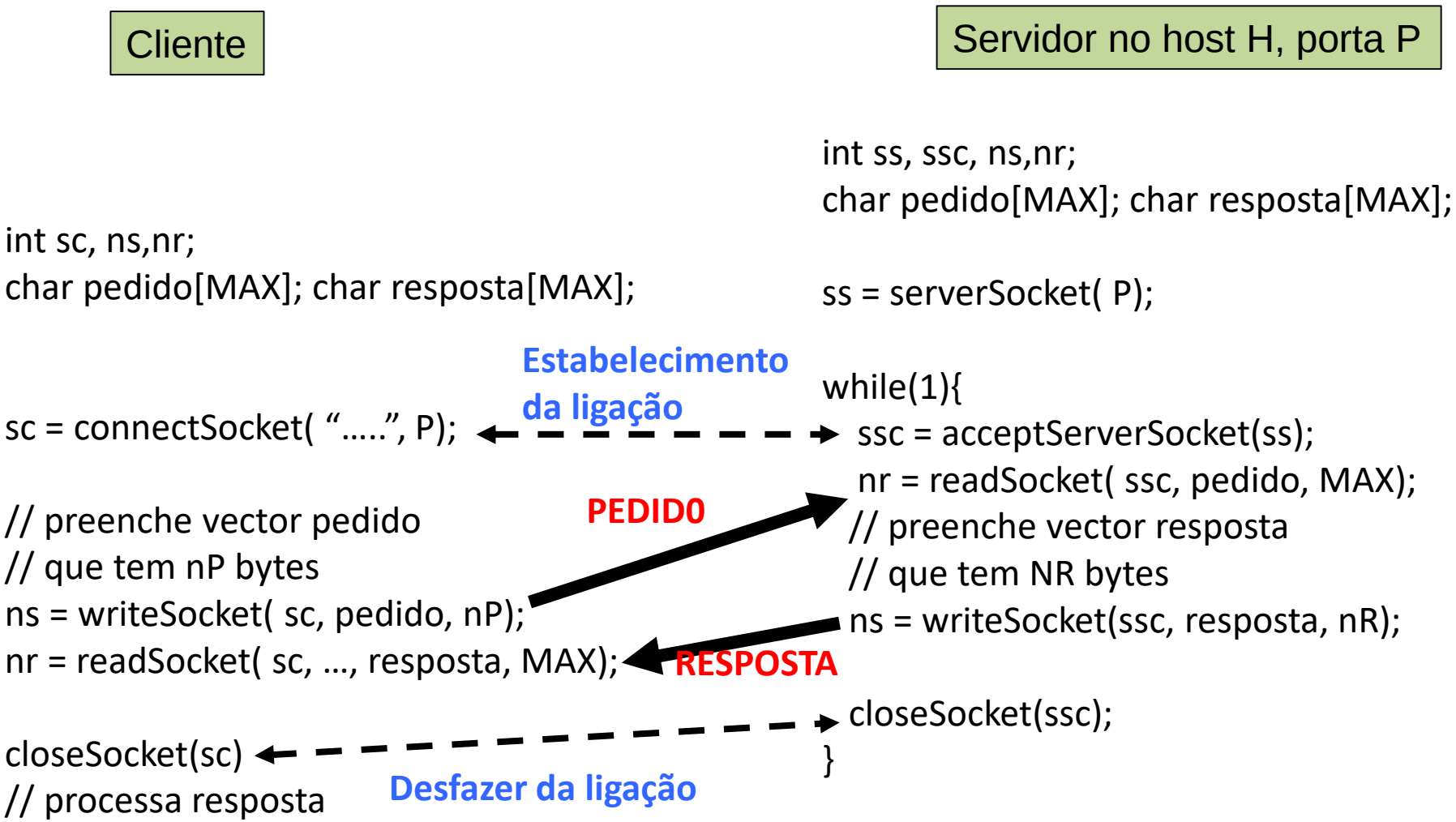
Cliente/Servidor com Sockets TCP



Uma biblioteca de sockets TCP

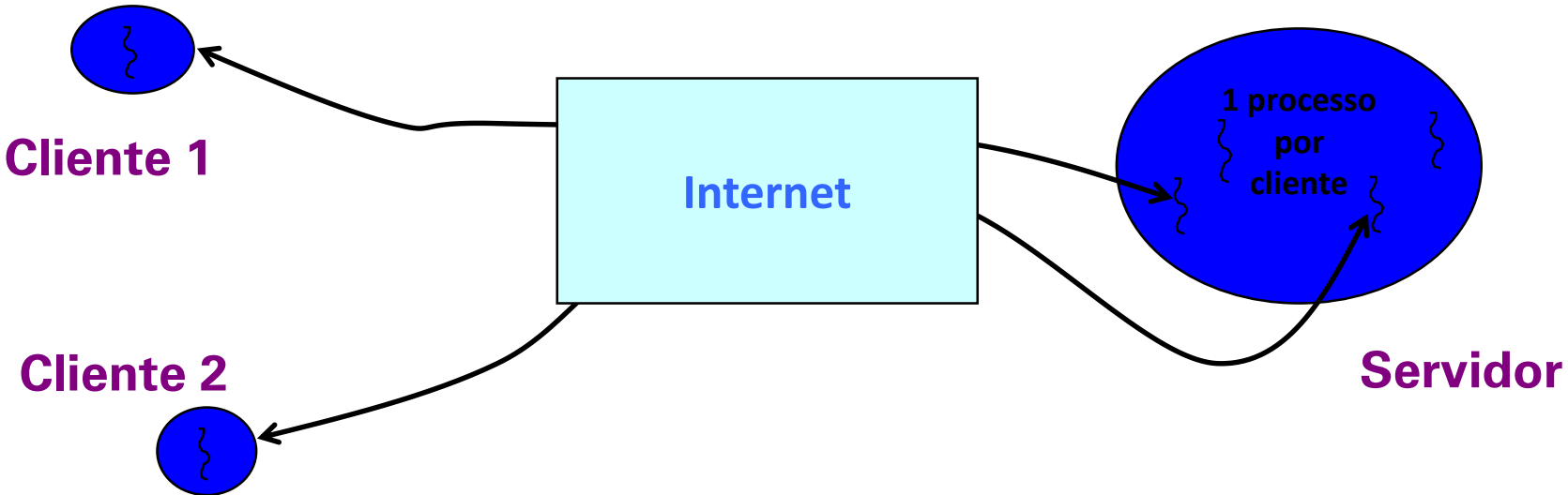
Operação	Parâmetros de entrada	Retorno
serverSocket (só no servidor)	Server Port	> 0 canal de entrada/saída (E/S) associado ao socket; -1 se erro
acceptServerSocket (só no servidor)	Canal retornado pela operação serverSocket	> 0 canal de E/S que permite ler e escrever bytes da/na conexão com o cliente
connectSocket (só no cliente)	String com o nome da máquina onde está o servidor, inteiro com o nº da porta do servidor	> 0 canal de E/S que permite ler e escrever bytes na/da conexão com o servidor
writeSocket (cliente e servidor)	canal de E/S, endereço inicial dos bytes a escrever, número de bytes a escrever	> 0, nº de bytes enviados < 0 erro
readSocket (cliente e servidor)	canal de E/S, endereço inicial do buffer para onde vão os bytes a receber, nº máximo de bytes a receber	> 0 nº de bytes recebidos < 0, erro == 0, o outro extremo da conexão fechou o canal de escrita
closeSocket (cliente e servidor)	Canal de E/S associado ao socket	0 OK; -1 erro

Cliente/Servidor com sockets TCP

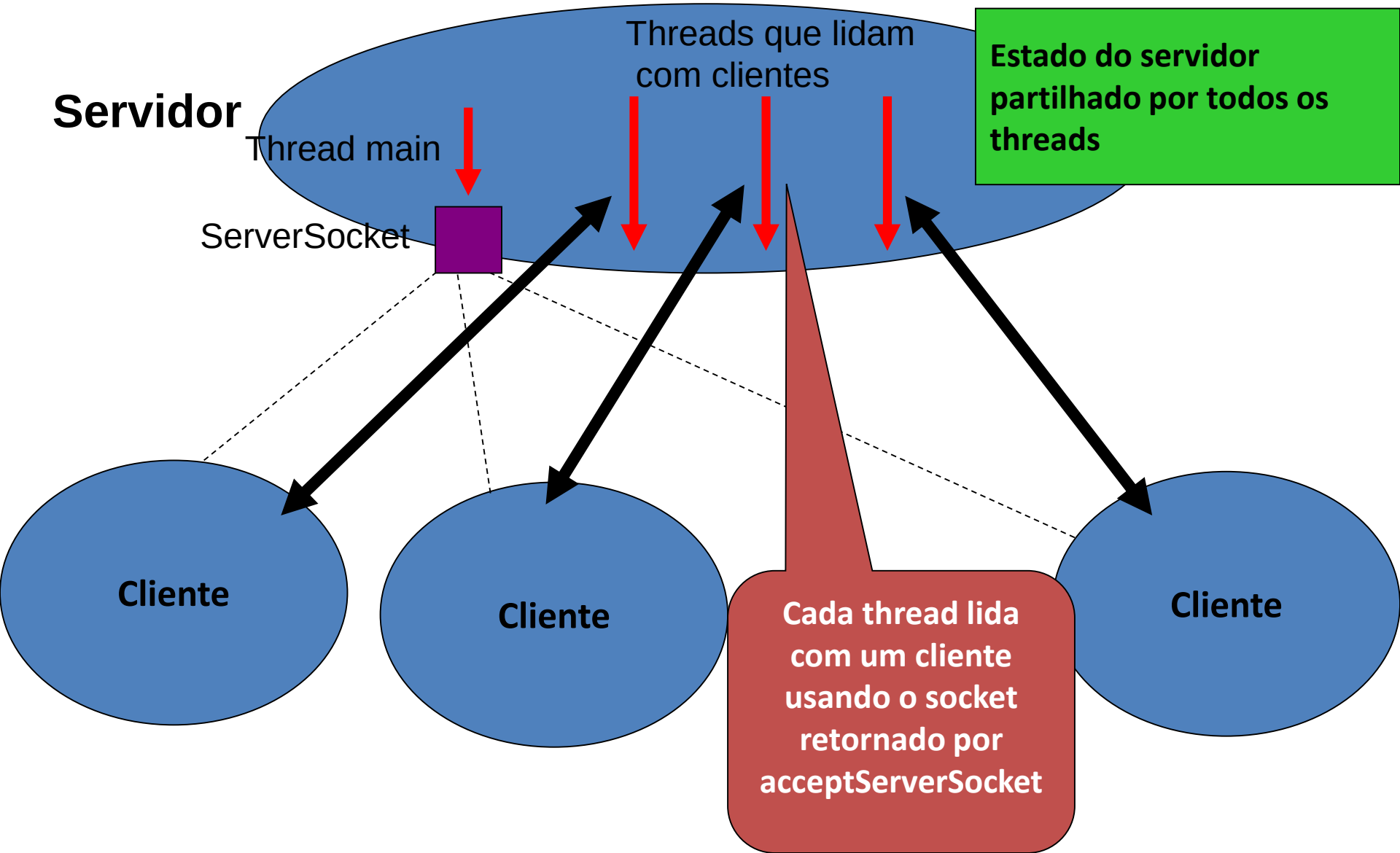


Servidores que atendem vários pedidos ao mesmo tempo

- Os servidores têm de ser capazes de dialogar com vários clientes em simultâneo
  - Os servidores que tratam vários pedidos ao meso tempo chamam-se servidores concorrentes.
- Quando é pedida uma nova conexão, o servidor aceita-a e lança um novo processo para responder ao cliente.

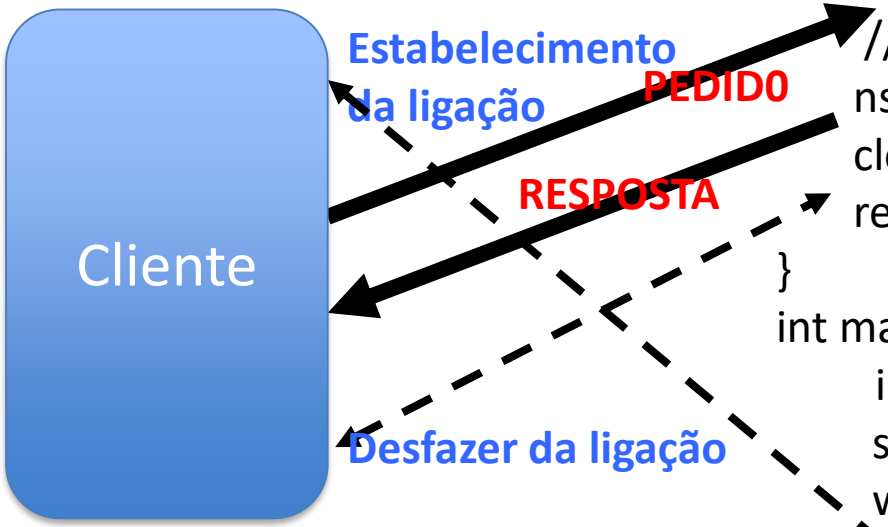


Servidor com múltiplos threads



Servidor com múltiplos threads

Servidor no host H, porta P




```
void * handleClient( void *arg){
        int ssc, ns, nr;
        char pedido[MAX]; char resposta[MAX];
        nr = readSocket( ssc, pedido, MAX);
        // preenche vector resposta  que tem NR bytes
        ns = writeSocket(ssc, resposta, nR);
        closeSocket(ssc);
        return NULL;
}
int main( ){
        int ss, ssc;
        ss = serverSocket( P);
        while(1){
            ssc = acceptServerSocket(ss);
            pthread_create( …,…, handleClient,
              (void *) ssc);
        }
}
```

Nível transporte: o protocolo UDP [1]

- Objectivo: transferência de dados entre dois (ou mais) computadores – troca de mensagens pura e simples, sem muitas garantias
- Não necessita de "*handshake*" (estabelecimento prévio de conexão) entre os *hosts*

Nível transporte: o protocolo UDP [2]

- **UDP – User Datagram Protocol** [RFC 768] é o serviço de “transmissão de dados sem conexão ou orientado a datagramas” usado na Internet
  - Transferência de mensagens de forma não fiável; não detecta a perda das mesmas
  - Sem Controlo de fluxo nem Controlo da saturação
- UDP é usado por: *streaming media*, tele-conferência, vídeo, telefonia

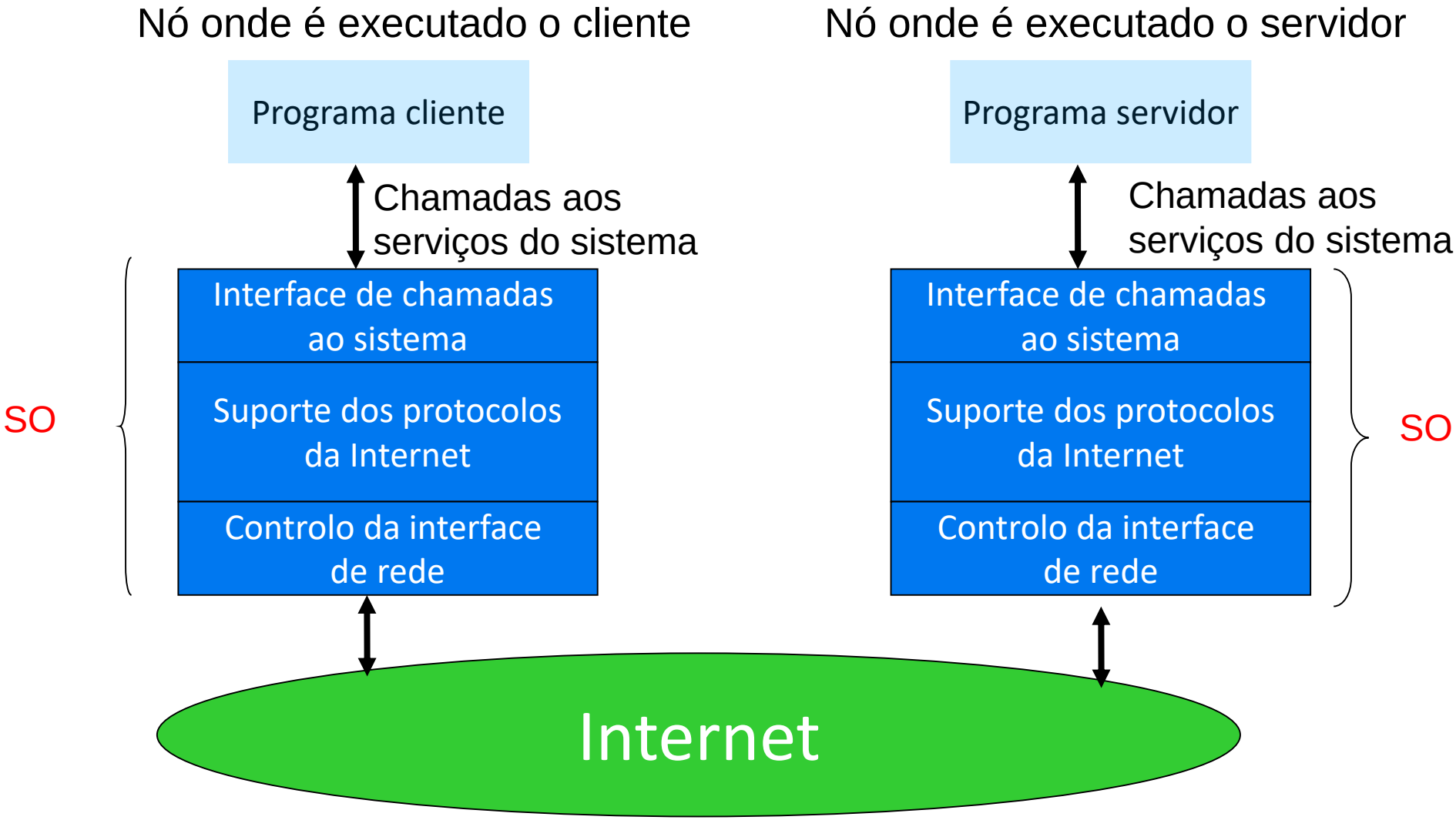
A noção de porta nos protocolos de transporte TCP/IP

- Endereço de portos/portas *“Port Addresses” (port numbers)*
  - A camada de transporte usa-os para identificar a aplicação a quem é dirigida uma mensagem
  - Analogia: número do apartamento num prédio (num determinado endereço)
  - Exemplo: a porta 80 é geralmente usado for serviços Web por omissão

Portas normalizadas para alguns serviços

Protocolo de aplicação	Porta	Utilização
ftp	20	Transferência de ficheiros
ssh	22	Login remoto seguro
smtp	25	Envio e recepção de email
http	80	Web
pop3	110	Protocolo de correio alternativo

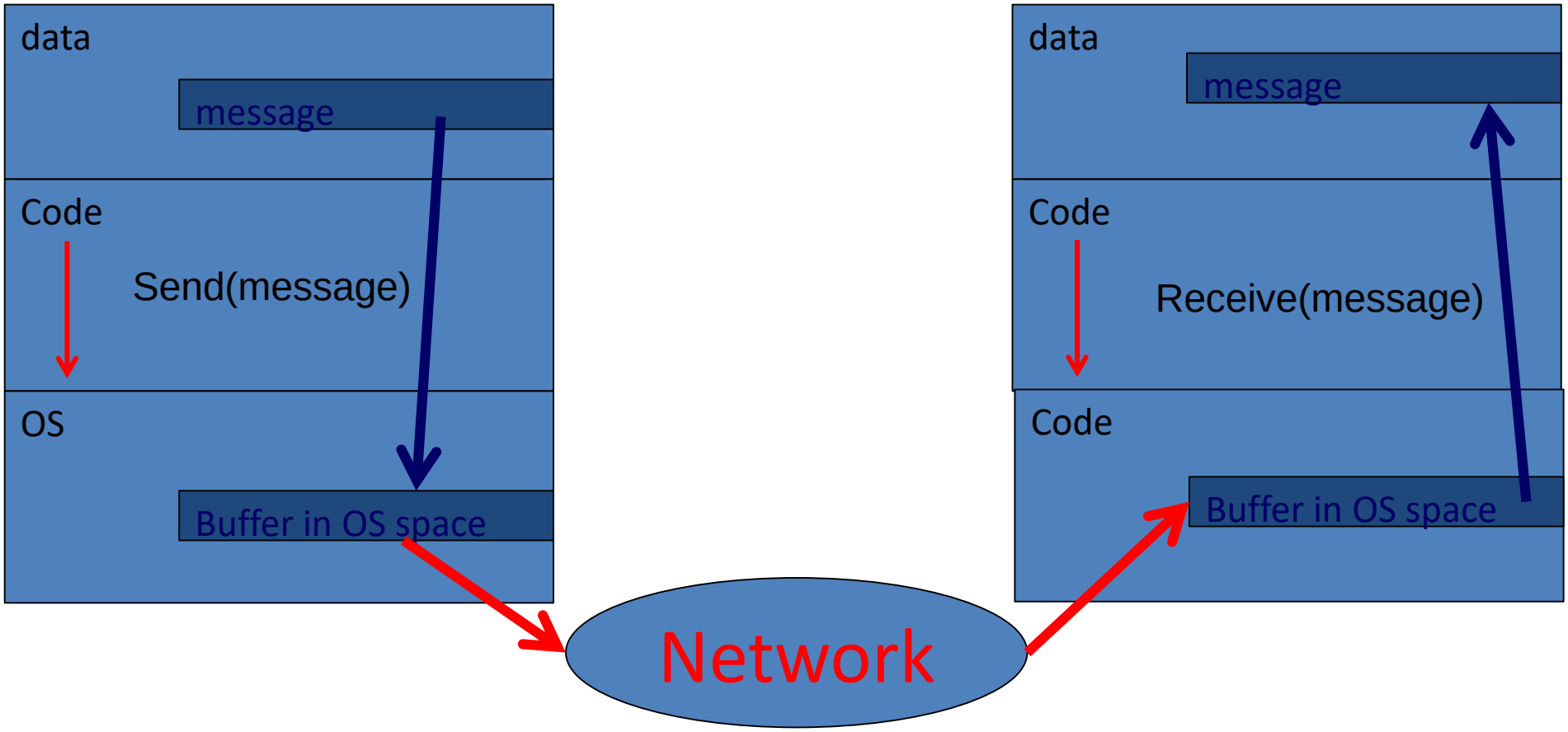
Aplicação cliente-servidor na Internet



Sockets TCP e UDP

- É um dispositivo que permite o acesso a portas TCP e a portas UDP
- Conjunto de chamadas ao sistema que permitem
  - criar conexões, destruir conexões, enviar bytes e receber bytes por uma canal (TCP)
  - Enviar e receber bytes contidos datagramas em portas UDP
- Conceito que provém do BSD UNIX (1982), mas todos os sistemas operativos modernos suportam a noção de *socket*

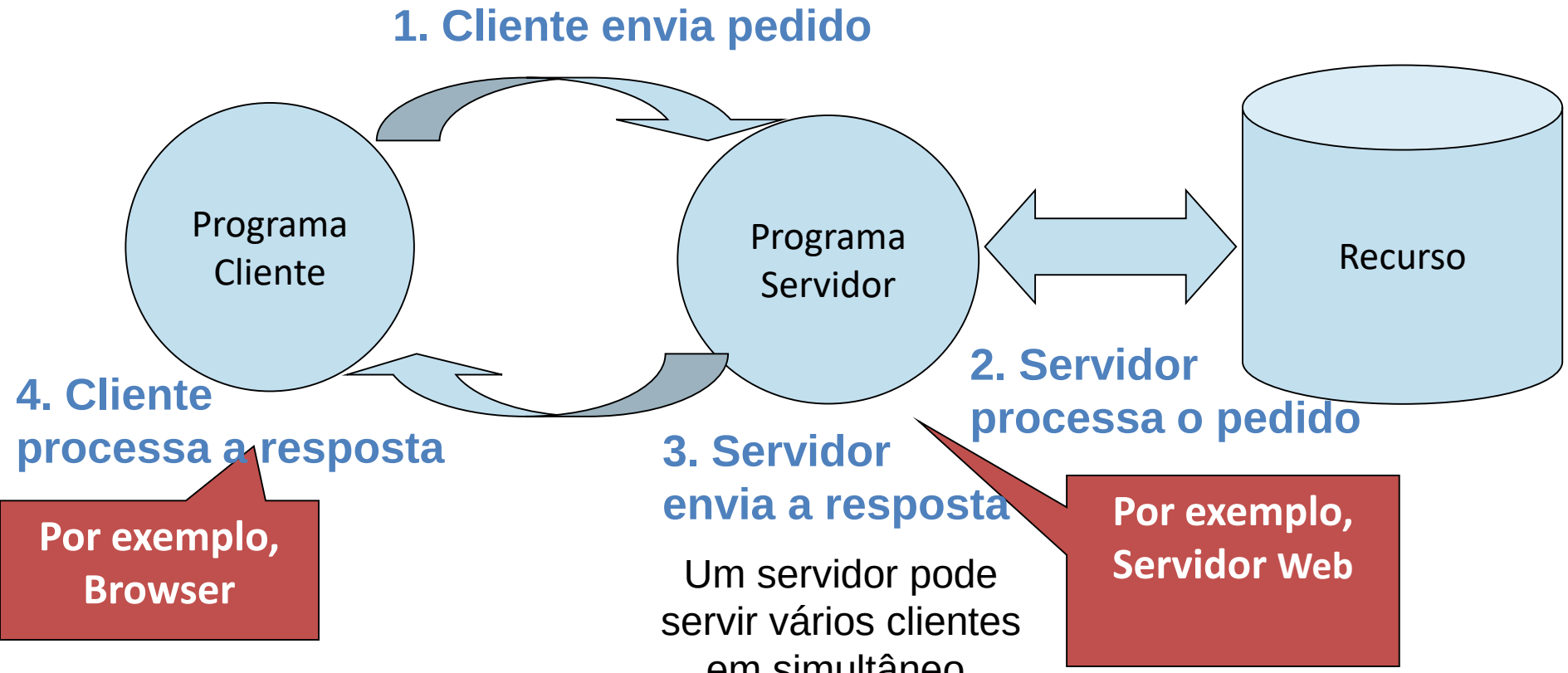
Comunicação entre processos em máquinas distintas



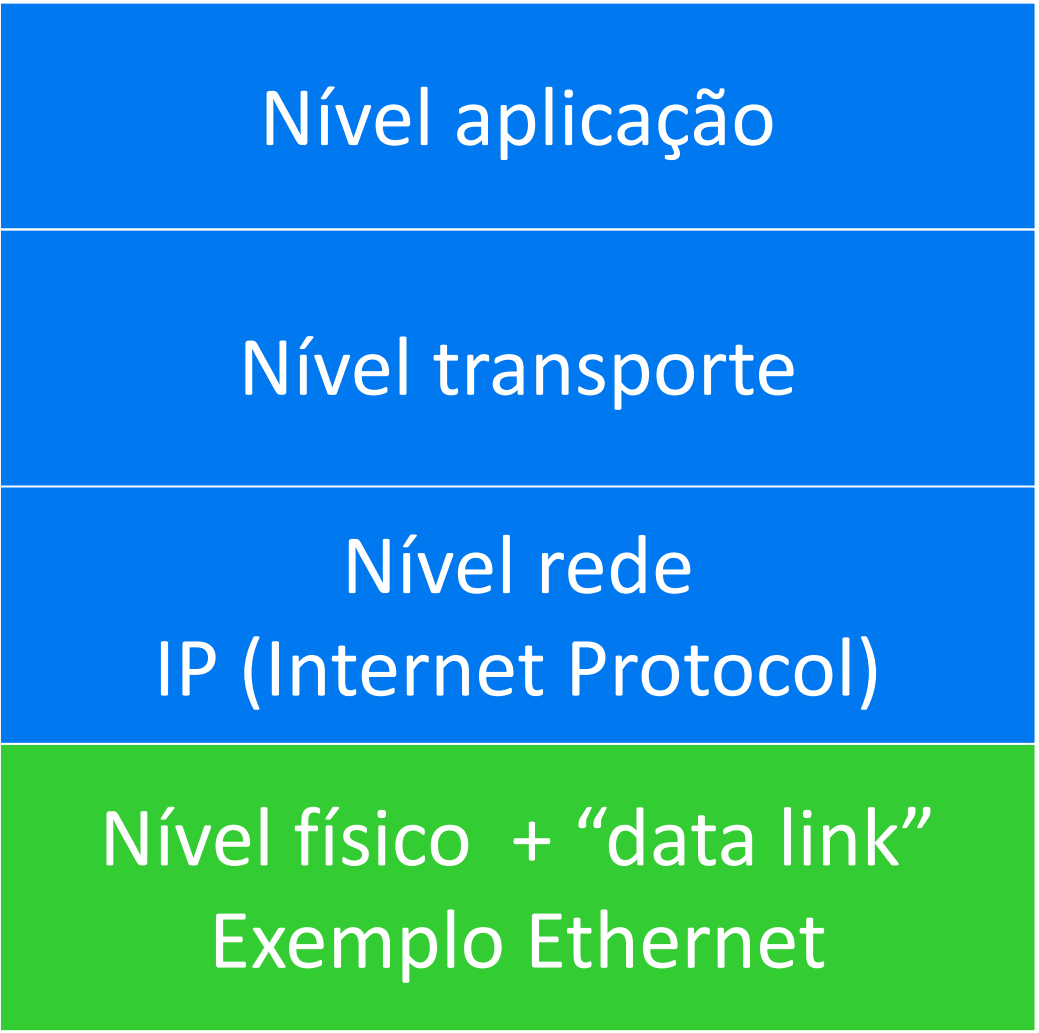
- O SO pode enviar e receber bytes através da rede
- O emissor tem de especificar: o endereço da máquina onde os bytes são entregues, e uma port / mailbox na máquina remota

Relação cliente-servidor

- Na maioria das situações, as aplicações que usam a rede são constituídas por dois programas.



A pilha de protocolos TCP/IP [1]



HTTP, FTP, SMTP, …
Cliente/servidor

UDP (Unreliable Datagram Protocol)
TCP (Transmission Contol Protocol)

Define a forma de designar os nós e o mecanismo de entrega de pacotes; não fiável

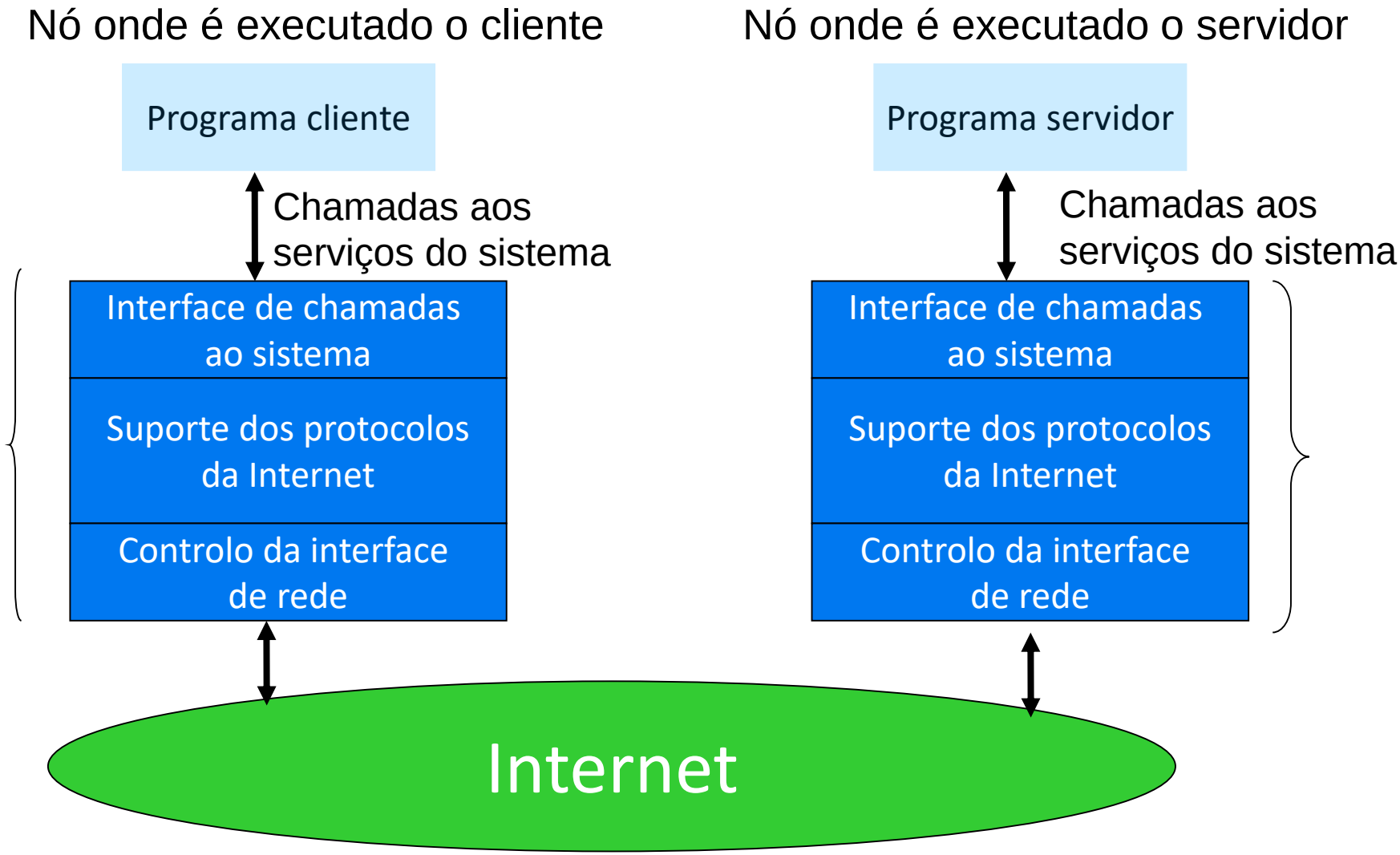
Nível transporte: o protocolo UDP [2]

- **UDP – User Datagram Protocol** [RFC 768] é o serviço de “transmissão de dados sem conexão ou orientado a datagramas” usado na Internet
  - Transferência de mensagens de forma não fiável; não detecta a perda das mesmas
  - Sem Controlo de fluxo nem Controlo da saturação
- UDP é usado por: *streaming media*, tele-conferência, vídeo, telefonia

A noção de porta nos protocolos de transporte TCP/IP

- Endereço de portos/portas *“Port Addresses” (port numbers)*
  - A camada de transporte usa-os para identificar a aplicação a quem é dirigida uma mensagem
  - Analogia: número do apartamento num prédio (num determinado endereço)
  - Exemplo: a porta 80 é geralmente usado for serviços Web por omissão

Aplicação cliente-servidor na Internet



Acesso ao protocolo de transporte UDP usando sockets [1]

- **Sockets UDP** são acessíveis através de um conjunto de chamadas ao sistema
- Livro OSTEP (cap 47) define uma biblioteca UDP construída sobre essas chamadas ao sistema com as seguintes funções:

  UDP_Open

  UDP_FillSockAddr

  UDP_Read

  UDP_Write

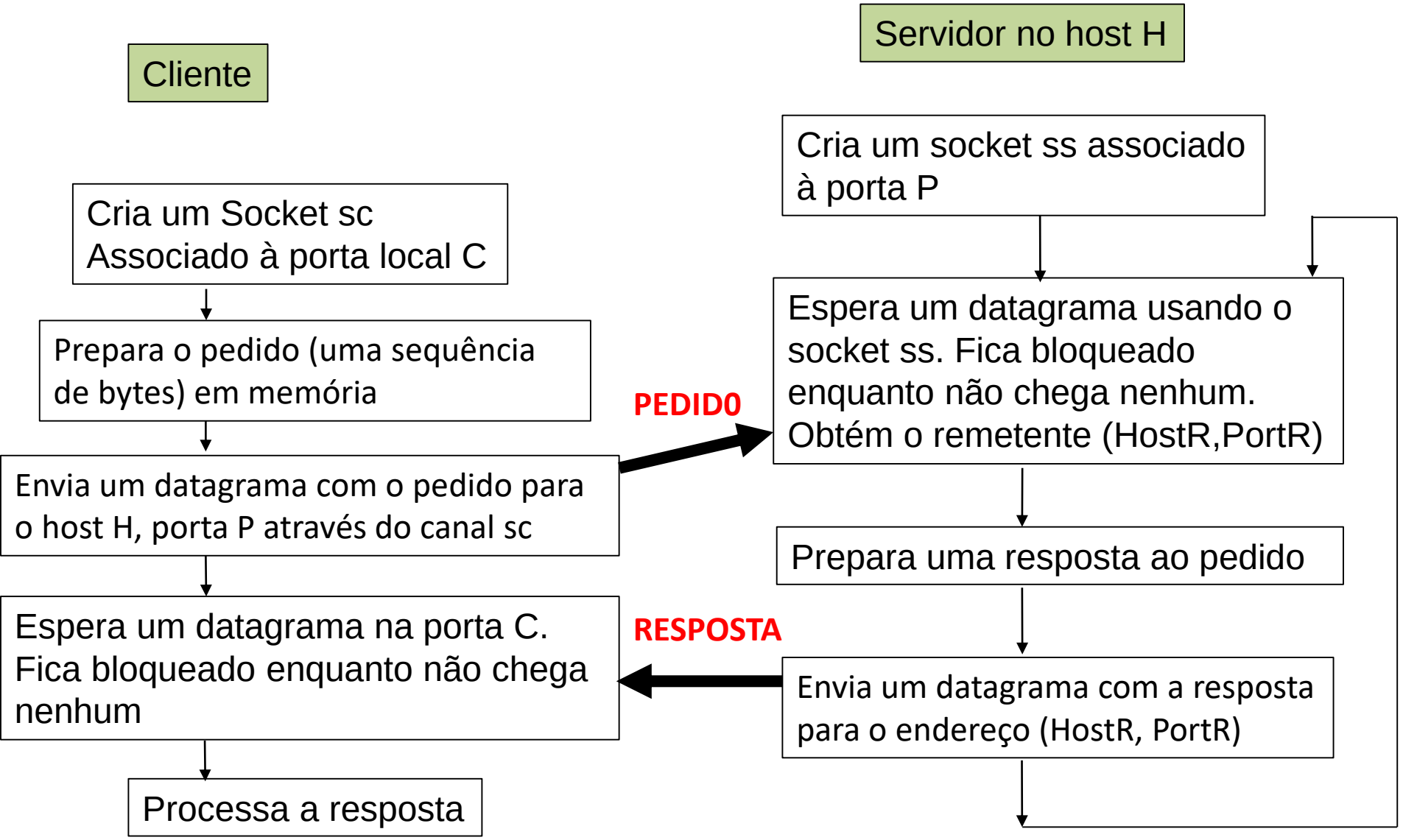
Acesso ao protocolo de transporte UDP usando sockets [2]

Função	Assinatura	Descrição
UDP_Open	int UDP_Open(int port)	*Cria um socket UDP associado a uma porta* **Entrada**: nº da porta local associada ao socket **Saída**: em caso de sucesso, nº do canal que permite o acesso ao socket, -1 se houve erro
UDP_FillSockAddr	int UDP_FillSockAddr (struct sockaddr_in *ss, char *hostname, int port)	**Entrada**: nome simbólico da máquina para a qual se pretende enviar um datagrama; número da porta na máquina de destino **Saída**: é preenchida a estrutura *ss* para ser usada na função UDP_Write. Retorna 0 se OK, -1 se há erro

Acesso ao protocolo de transporte UDP usando sockets [3]

Função	Assinatura	Descrição
UDP_Read	int UDP_Read(int s, struct sockaddr_in *sr, char *buffer, int maxb)	*Recebe um datagrama UDP* **Entrada**: nº canal *s* associado à porta criada, estrutura *sr* que irá conter o endereço do remetente do datagrama, espaço em RAM onde é recebido o datagrama definido pelo endereço inicial *buffer* e pelo tamanho máximo *maxb* **Saída**: nº de bytes recebidos, -1 em caso de erro
UDP_Write	Int UDP_Write(int s, struct sockaddr_in *sd, char *buffer, int nb)	*Envia um datagrama UDP* **Entrada**: nº canal *s* associado à porta criada, estrutura *sr* com o endereço do destinatário, espaço em RAM onde foi preenchido o datagrama a enviar definido pelo endereço inicial *buffer* e pelo tamanho *nb* **Saída**: nº de bytes enviados, -1 em caso de erro

Cliente/Servidor com Sockets UDP [1]

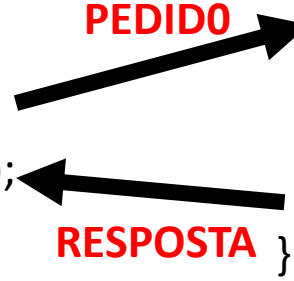


Cliente/Servidor com Sockets UDP [2]

Cliente usa porta PC

```
int sc, ns,nr;
char pedido[MAX]; char resposta[MAX];
struct sockaddr_in serv;

sc = UDP_Open( PC);
UDP_FillAddrSock( serv, "…", PS);
// preenche vector pedido
// que tem nP bytes
ns = UDP_Write( sc, serv, pedido, nP);
nr = UDP_Read( sc, …, resposta, MAX);
// processa resposta
```



Servidor no host H, porta PS

```
int ss, ns,nr;
char pedido[MAX]; char resposta[MAX];
struct sockaddr_in client;

ss = UDP_Open( PS);
while(1){
  nr = UDP_Read( ss, &client, pedido, MAX);
  // preenche vector resposta
  // que tem NR bytes
  ns = UDP_Write( ss, resposta, pedido, nR);
}
```