Infra-Estrutura de Software

Entrada / Saída

Próximas Datas

- 2º. EE: 21/06
  - http://www.cin.ufpe.br/~cagf/if677/2016-1/slides/
- Revisão de notas: 28/06
- FINAL: 30/06

Diversidade de dispositivos

Hardware de E/S

Dispositivo	Taxa de dados
Teclado	10 bytes/s
Mouse	100 bytes/s
Modem 56 K	7 KB/s
Scanner	400 KB/s
Filmadora *camcorder* digital	3,5 MB/s
Rede sem fio 802,11g	6,75 MB/s
CD-ROM 52x	7,8 MB/s
Fast Ethernet	12,5 MB/s
Cartão flash compacto	40 MB/s
FireWire (IEEE 1394)	50 MB/s
USB 2.0	60 MB/s
Padrão SONET OC-12	78 MB/s
Disco SCSI Ultra 2	80 MB/s
Gigabit Ethernet	125 MB/s
Drive de disco SATA	300 MB/s
Fita Ultrium	320 MB/s
Barramento PCI	528 MB/s

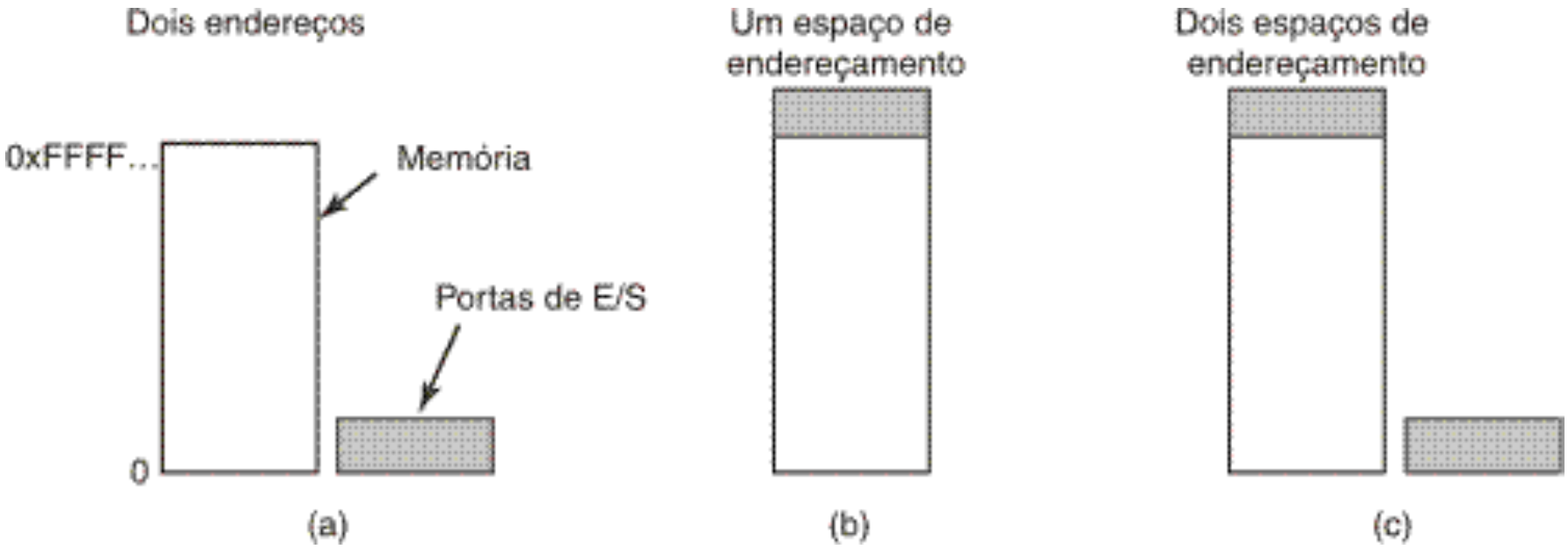
Tabela 5.1 Algumas taxas de dados típicas de dispositivos, placas de redes e barramentos.

E/S: Como a CPU acessa a informação?

- **Espaço de endereçamento**: conjunto de endereços de memória que o processador consegue acessar diretamente
- A forma de acessar os **registradores** (das interfaces) **dos periféricos** é definida no projeto do processador:
  - Espaço único
  - Dois espaços, um deles dedicado à E/S (isolada)

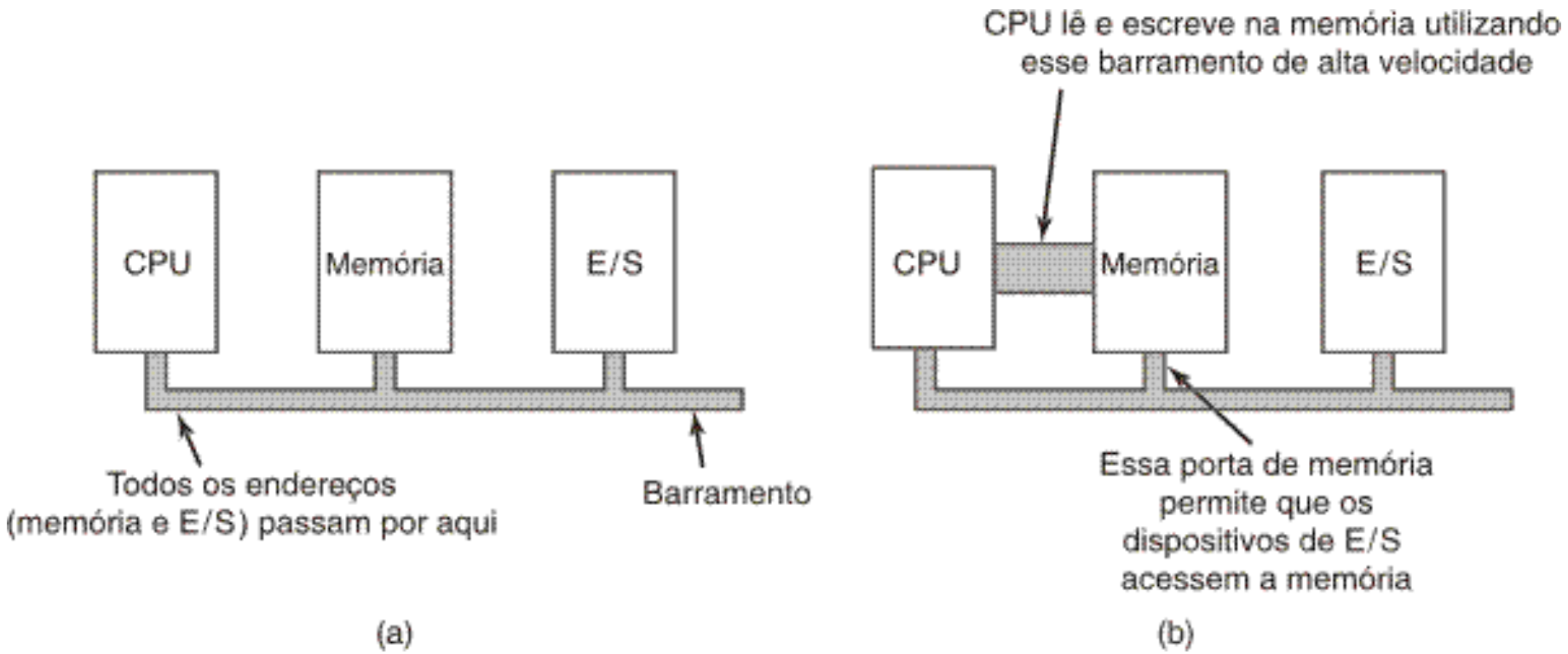
- **E/S isolada**
  - Através de instruções especiais de E/S
  - Especifica a leitura/escrita de dados numa porta de E/S
- **E/S mapeada em memória**
  - Através de instruções de leitura/escrita na memória
- **Híbrido** (ex. IBM-PC):
  - E/S mapeada em memória: memória de vídeo
  - E/S isolada: dispositivos em geral

Espaços de Memória e E/S



a) Espaços de memória e E/S separados - E/S isolada
b) E/S mapeada na memória
c) Híbrido

E/S mapeada na memória



(a) Arquitetura com barramento único
(b) Arquitetura com barramento duplo (dual)

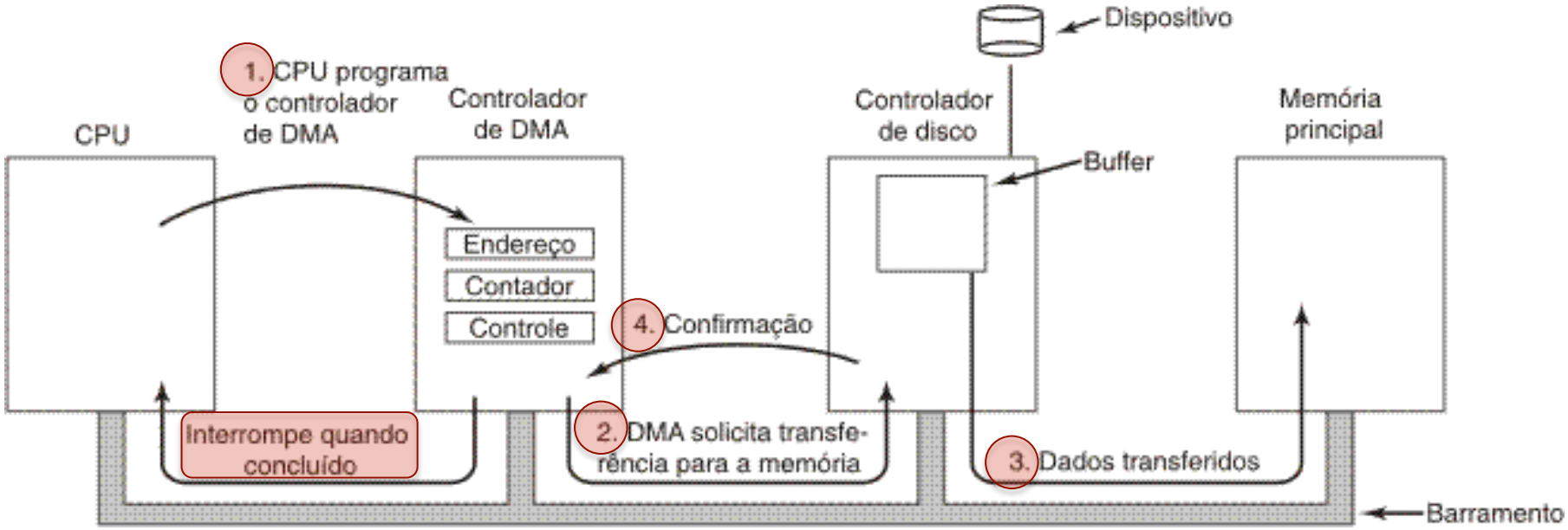
Como o processador "enxerga" a memória e os demais dispositivos ou como o processador se comunica com o seu exterior

- O processador realiza operações como:
  - Ler um dado da memória
  - Escrever um dado na memória
  - Receber (ler) um dado de dispositivos de E/S
  - Enviar (escrever) dados para dispositivos de E/S

- Nas operações de acesso à memória, o processador escreve e lê dados, praticamente sem intermediários

- Nos acessos a dispositivos de E/S, existem circuitos intermediários, que são as *interfaces*

Acesso Direto à Memória (DMA)



Operação de uma transferência com DMA

Revisitando 'interrupções'

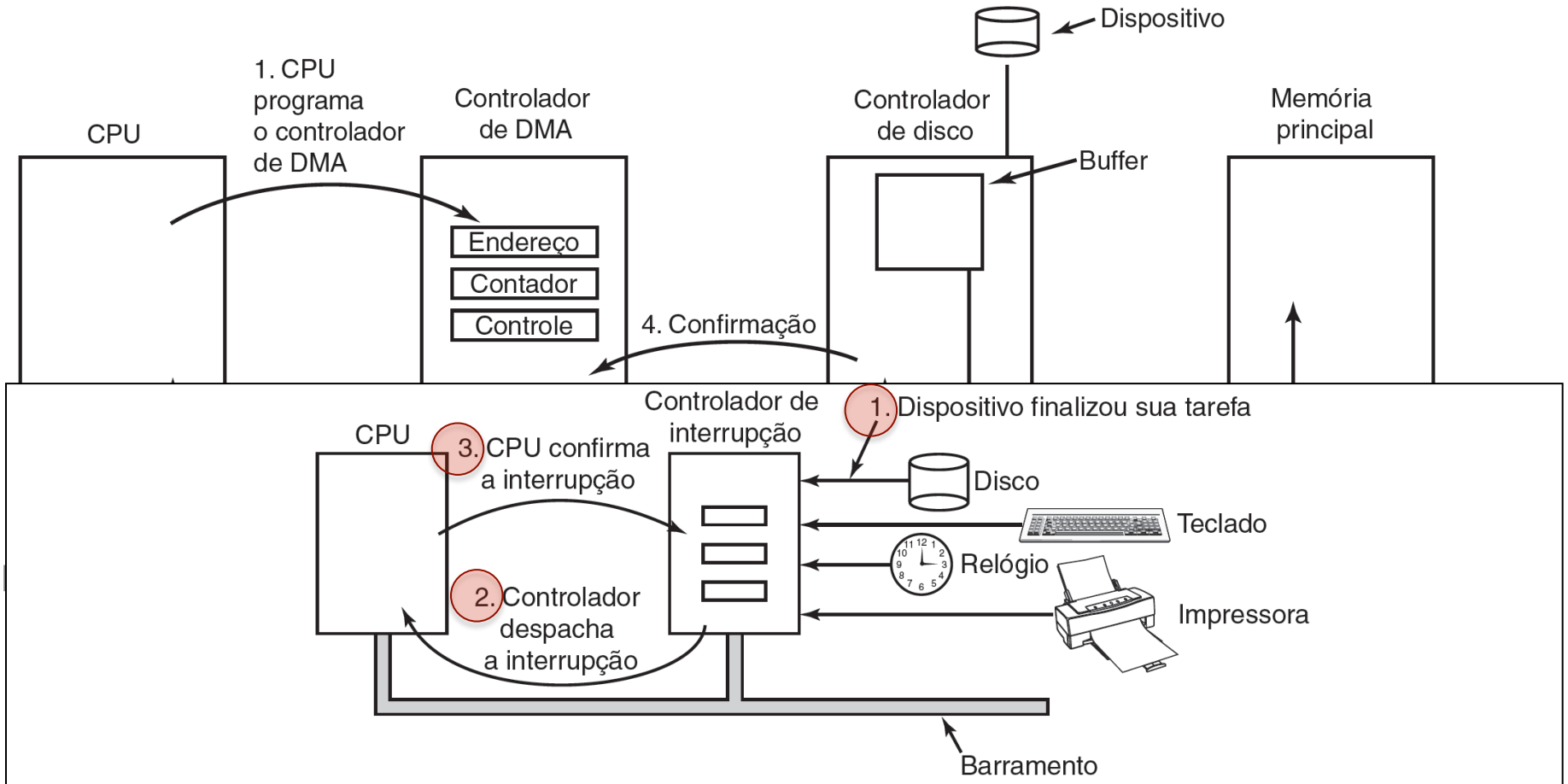


Figura 5.4 Como ocorre uma interrupção. As conexões entre os dispositivos e o controlador de interrupção atualmente utilizam linhas de interrupção no barramento, em vez de cabos dedicados.

Como a CPU sabe que o dispositivo já executou o comando?

- E/S Programada
  - CPU lê constantemente o status do controlador e verifica se já acabou (Polling ou Busy-waiting)
  - Desvantagem: Espera até o fim da operação
- E/S por Interrupção
  - CPU é interrompida pelo módulo de E/S e ocorre transferência de dados
  - CPU continua a executar outras operações
  - Desvantagem: toda palavra lida do (ou escrita no) periférico passa pela CPU
- E/S por DMA - Acesso Direto à Memória
  - Quando necessário, o controlador de E/S solicita ao controlador de DMA a transferência de dados de/para a memória
  - Nesta fase de transferência não há envolvimento da CPU
  - Ao fim da transferência, a CPU é interrompida e informada da transação [figura anterior]

Entrada/Saída

✓Princípios do hardware de E/S
- Princípios do software de E/S
- Camadas do software de E/S
- Gerenciamento de energia

Objetivos da gerência de E/S

- Eficiência
- Uniformidade (desejável):
  - Todos dispositivos enxergados da forma mais uniforme possível
- Esconder os detalhes (estes são tratados pelas camadas de mais baixo nível)
- Fornecer abstrações genéricas: read, write, open, close etc.

Princípios básicos do software de E/S

- Subsistema de E/S é complexo, dada a diversidade de periféricos
- Padronizar ao máximo para reduzir número de rotinas
  - Novos dispositivos não alteram a visão do usuário em relação ao SO
- Organizado em camadas

Visão Geral do software de E/S

- Tratador de interrupção
  - É acionado ao final da operação de transferência
  - Aciona driver
- Driver de dispositivo
  - Recebe requisições
  - Configura (aciona) o controlador
- E/S independente de dispositivo
  - Nomes e proteção
  - bufferização
- E/S em nível de usuário
  - Chamadas de E/S

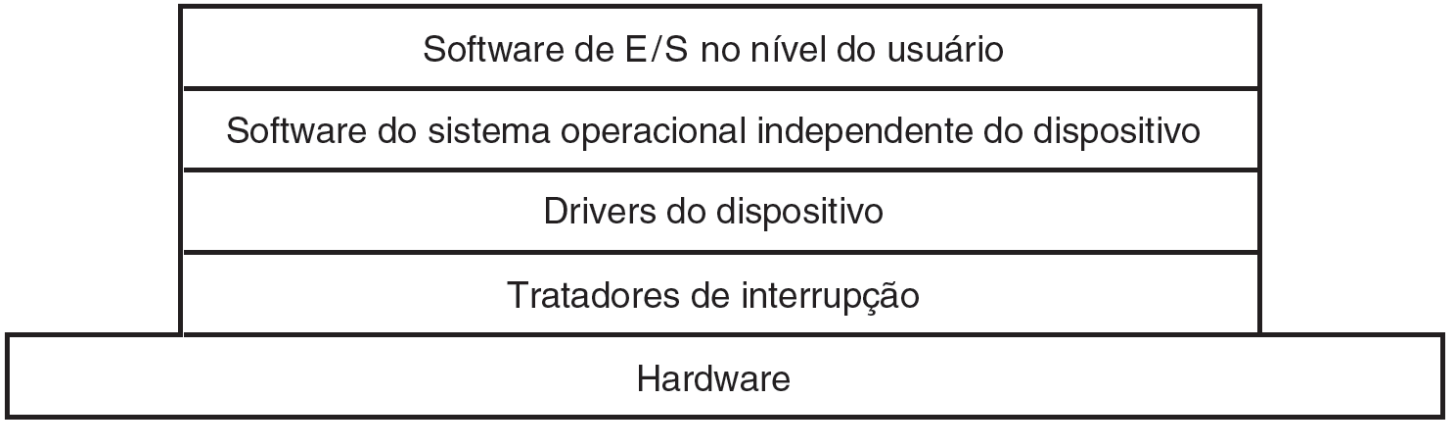
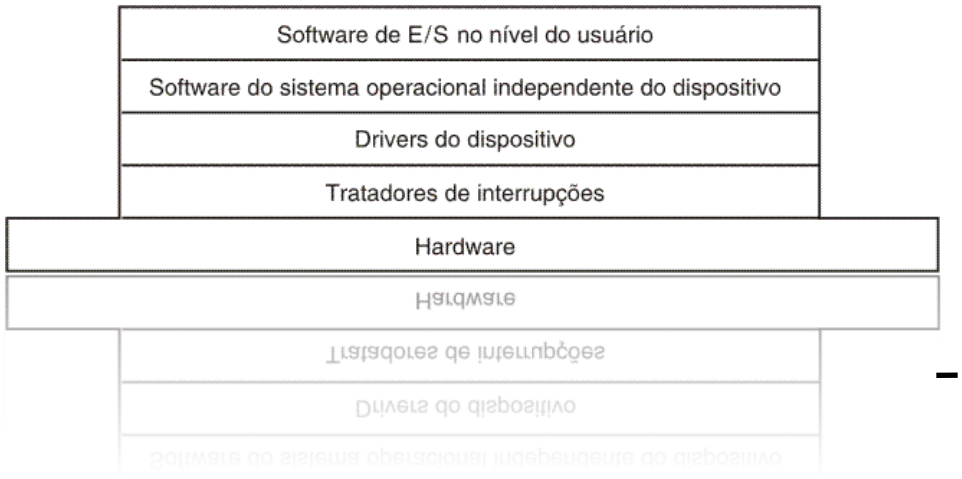
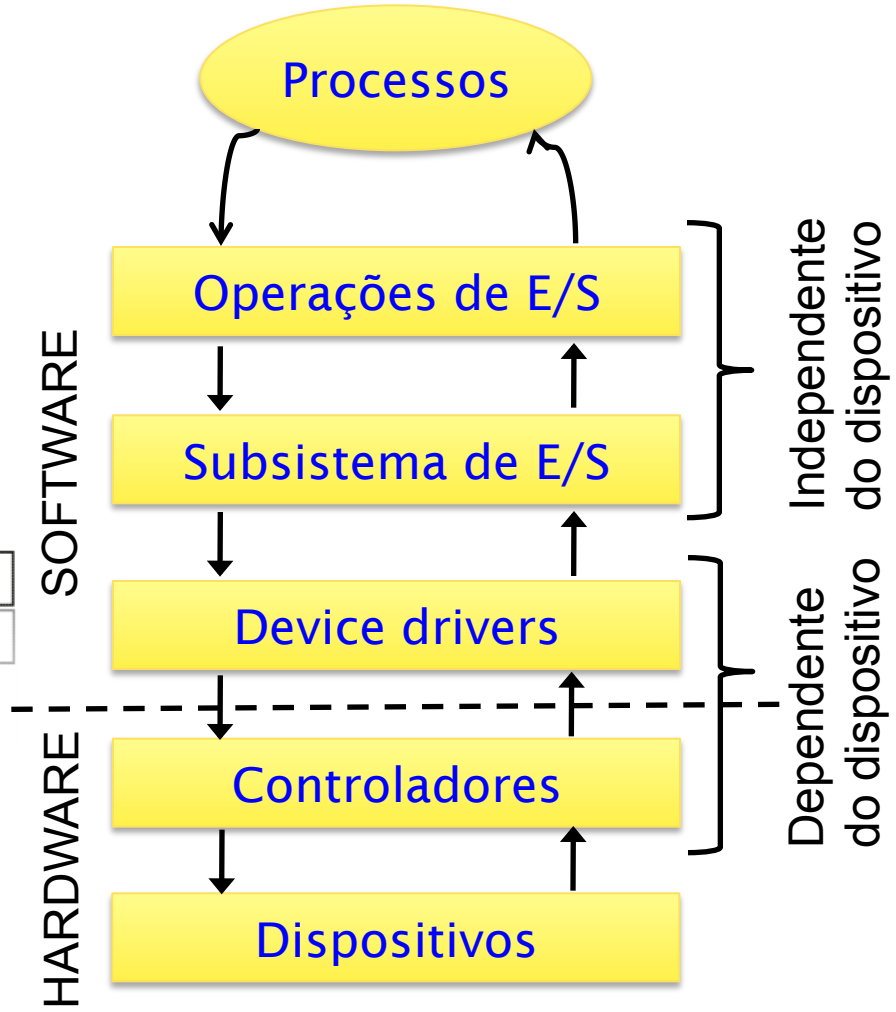


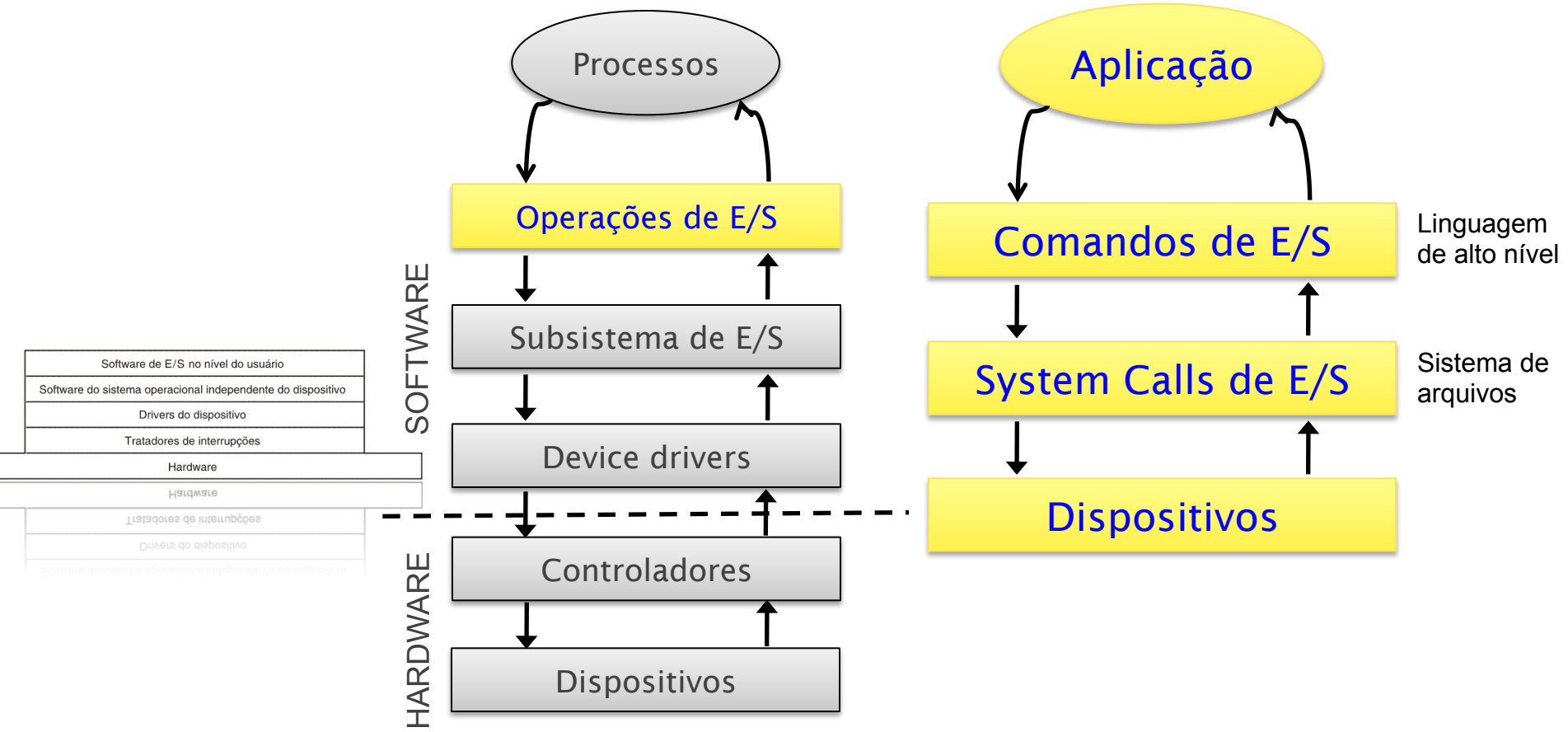
Figura 5.10 Camadas do software de E/S.

Camadas do Software de E/S





Camadas do Software de E/S

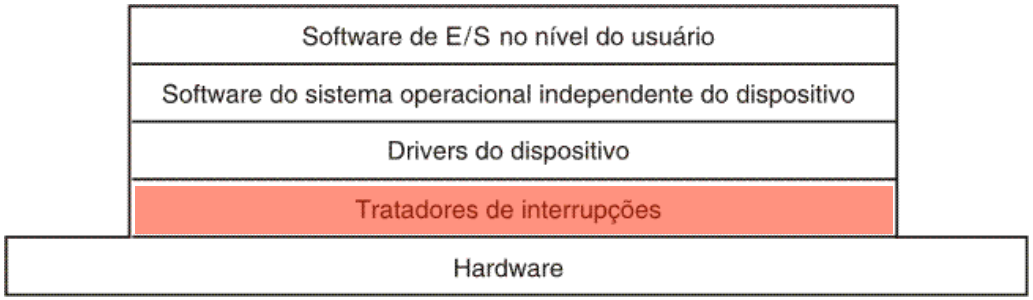


Tratador(es) de Interrupção

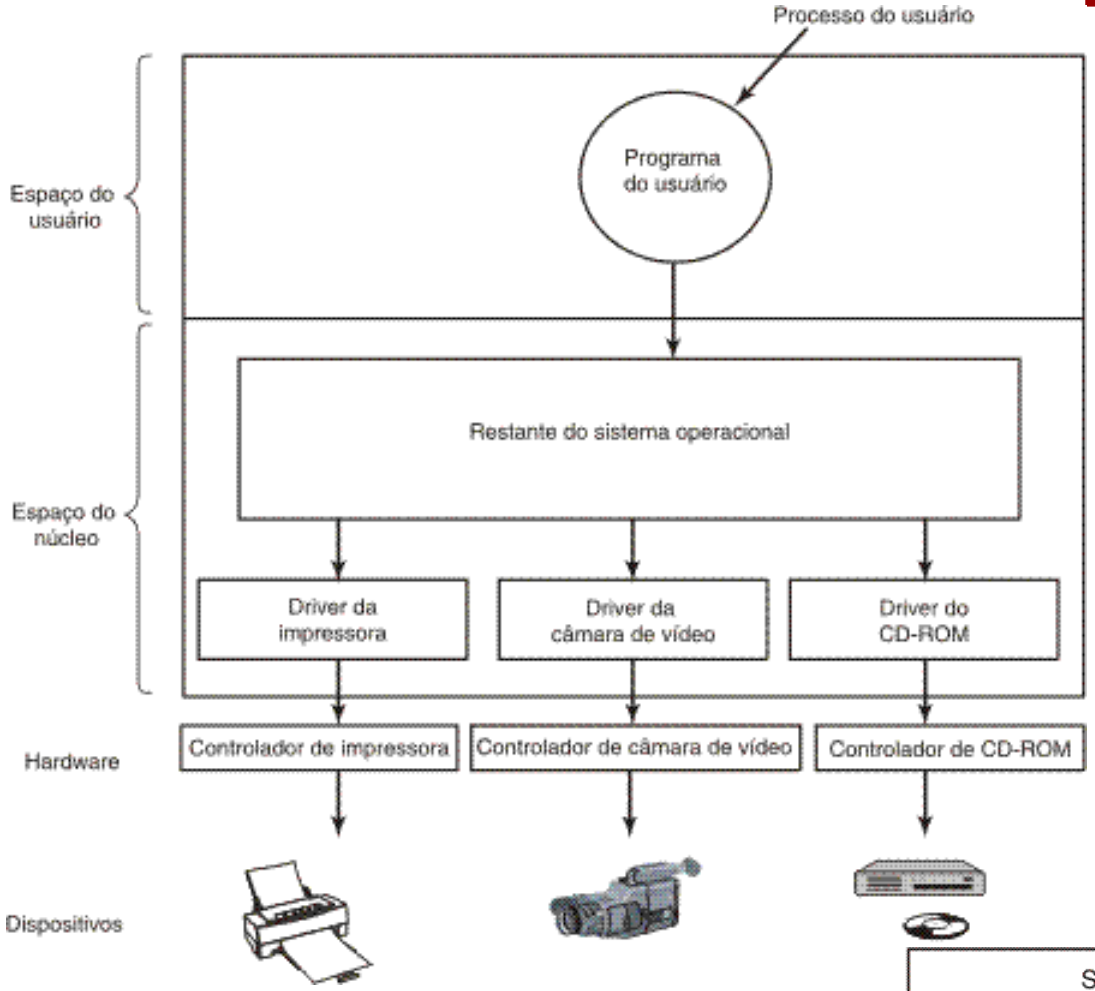
- As interrupções devem ser escondidas (transparentes) o máximo possível

  Para tanto:

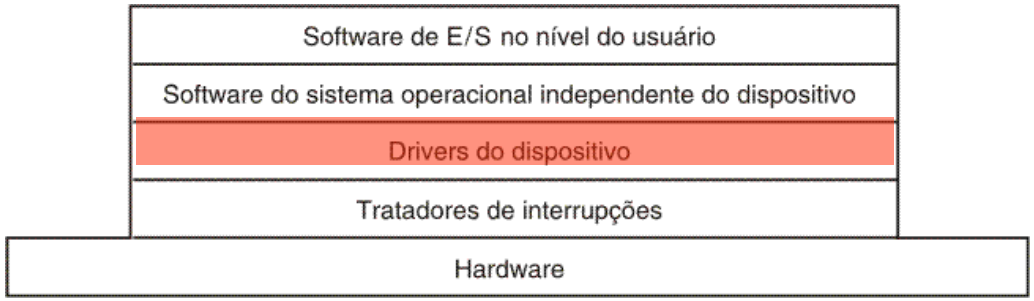
  - bloqueia o *driver* que iniciou uma operação de E/S
  - rotina de tratamento de interrupção cumpre sua tarefa
  - notifique que a E/S foi completada
  - e então desbloqueia o *driver* que a chamou



Drivers dos Dispositivos



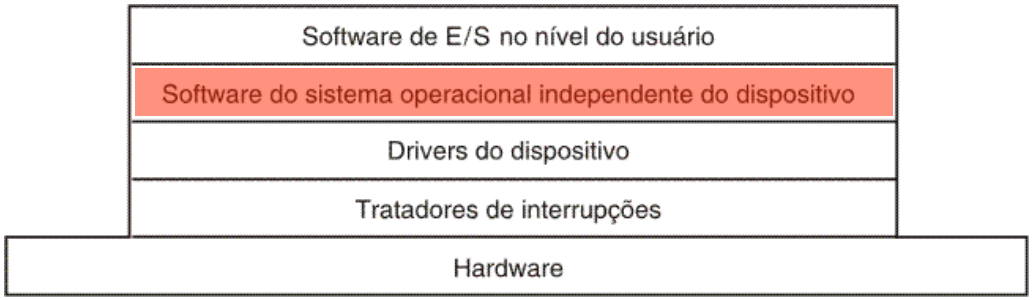
- Posição lógica dos drivers dos dispositivos
- A comunicação entre os drivers e os controladores de dispositivos é feita por meio do barramento



Software de E/S Independente de Dispositivo

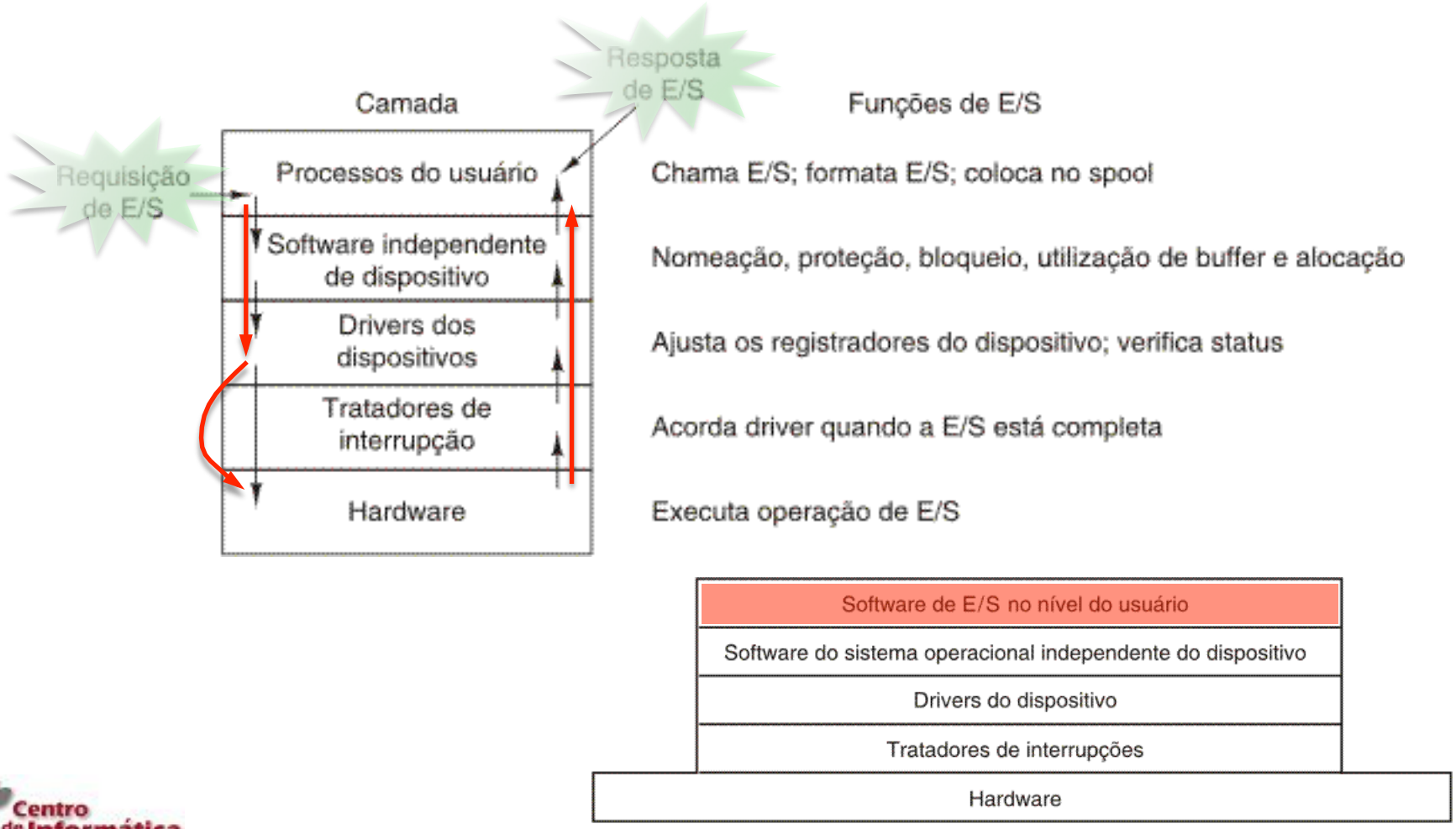
Funções do software de E/S independente de dipositivo

Interface uniforme para os drivers dos dispositivos
Armazenamento em buffer
Relatório de erros
Alocação e liberação de dispositivos dedicados
Fornecimento de tamanho de bloco independente de dispositivo



Software de E/S do Espaço do Usuário

Camadas do sistema de E/S e as principais funções de cada camada



Conclusões: princípios básicos do software de E/S

- Subsistema de E/S é complexo dada a diversidade de periféricos
- Padronizar ao máximo para reduzir número de rotinas
  - Novos dispositivos não alteram a visão do usuário em relação ao SO
- Organizado em camadas

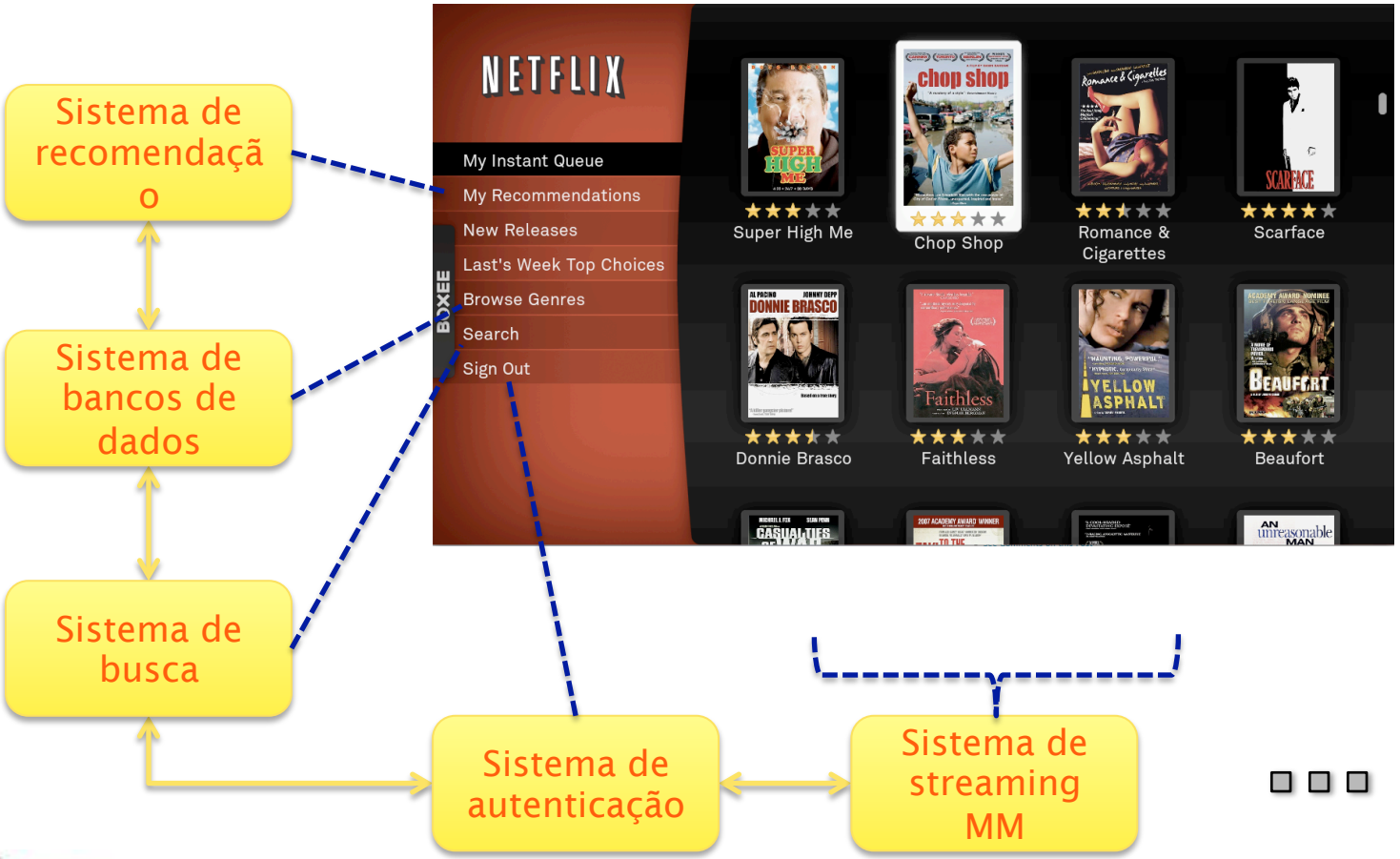
Para concluir...

SISTEMAS DISTRIBUÍDOS

Tendências-chave

- O que move os **Sistemas Distribuídos** hoje?
  - Pervasividade das redes
  - Computação móvel e ubíqua
  - Importância crescente de sistemas multimídia (distribuídos)
  - Sistemas distribuídos como utilidade (serviço) – *Cloud Computing* e seus modelos (IaaS, PaaS, SaaS)
- Principal motivação: **compartilhamento de recursos**

Sistemas multimídia distribuídos



Conceito-chave

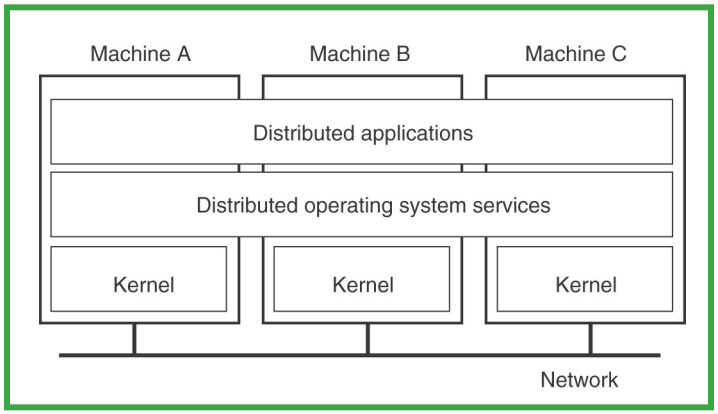
- Distribuição
- Comunicação
- Complexidade
- Heterogeneidade

- **Transparência**

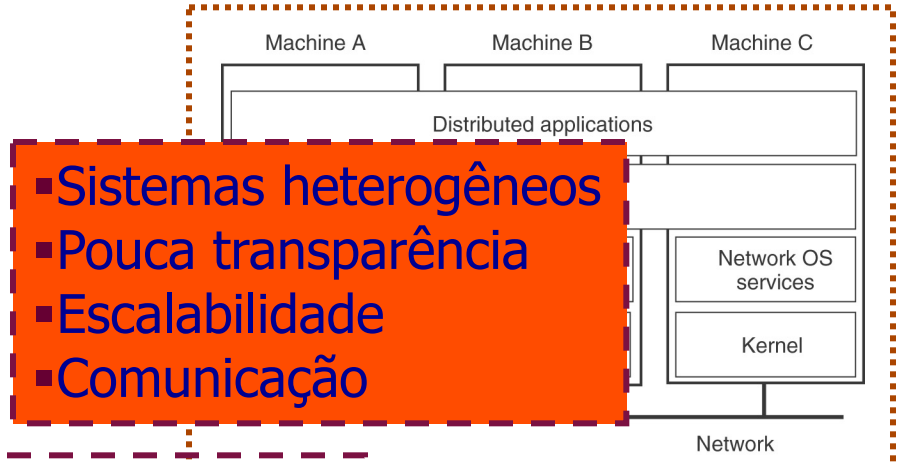
Infra-estruturas de software para Sistemas Distribuídos

- Sistemas Operacionais Distribuídos
- Sistemas Operacionais de Rede
- Middleware

Infra-estruturas para SDs: diferenças de objetivos e abstrações



- Sistemas homogêneos
- Transparência de distribuição
- Alto desempenho
- Memória compartilhada
- Controle de concorrência



- Sistemas heterogêneos
- Pouca transparência
- Escalabilidade
- Comunicação

- Sistemas heterogêneos
- Transparência de distribuição e comunicação
- Serviços
- Abertura

Middleware "Tradicional"

- Message-Oriented Middleware (MOM)
- Transaction Processing Monitors (TPMON)
  - Forte associação com acesso distribuído a BD
  - Propriedades "ACID"
- Remote Procedure Calls (RPC)
- Object-Oriented Middleware (ORB, ...)

```
// 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() );
  }
}
```

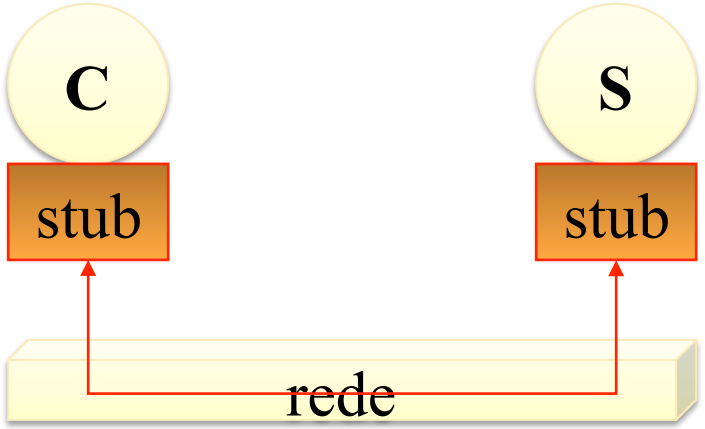
Chamada de Procedimentos Remotos

- Remote Procedure Call (RPC)
- 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

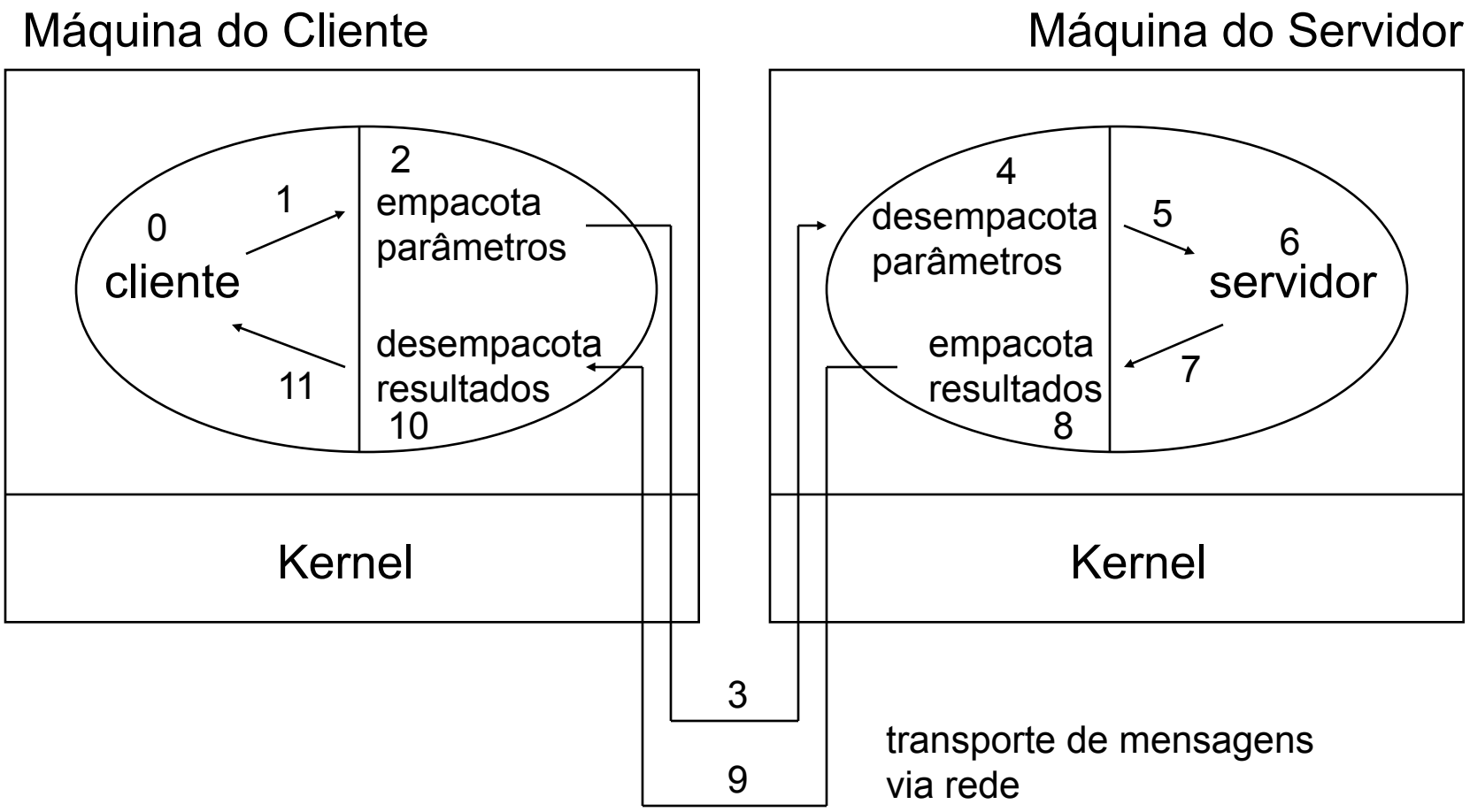
RPC: processamento de interface

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

- ***Stubs*** (no cliente e no servidor): **transparência de acesso**
  - tratamento de algumas exceções no local
  - *marshalling*
  - *unmarshalling*



Chamadas e mensagens em RPC

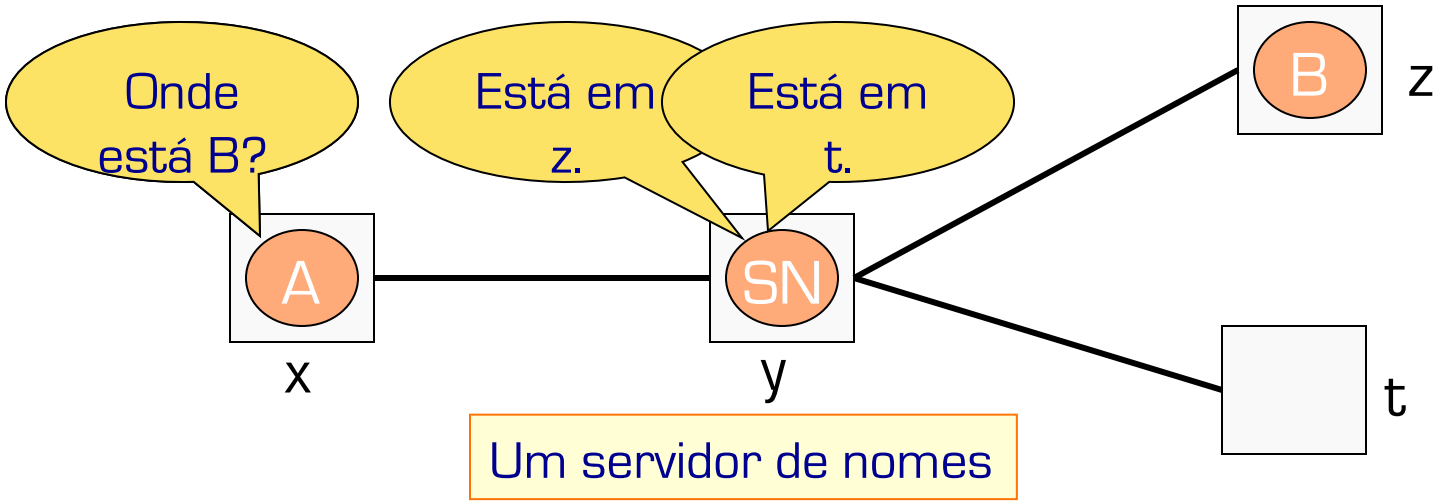


RPC: ligação (antes da comunicação...)

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

Ligação Dinâmica e Nomes

- Nomes podem ser a solução, mas na realidade os endereços físicos é que são necessários...
- Daí, faz-se necessário mapear nomes em endereços... como serviço
- Geralmente, algum agente intermediário tem este papel específico de *resolvedor de nomes*



Computação Distribuída

Conclusões

Características marcantes

- Concorrência de componentes – possibilidade de paralelismo
- Compartilhamento de recursos
- Componentes falham de forma independente – falha parcial

Sistemas Distribuídos

- Muito mais do que comunicação entre processos
- É preciso abstrações (serviços) para, entre outros:
  - Ligação (por nome, e não por endereço)
  - Segurança
  - Controle de transações distribuídas
  - Sincronização de relógios
- É preciso ter infra-estrutura de software
  - Middleware (ex. RPC, RMI) – sobre SOR
  - SOD (ex. Chorus)

Próximas Datas

- 2º. EE: 21/06
  - http://www.cin.ufpe.br/~cagf/if677/2016-1/slides/
- Revisão de notas: 28/06
- FINAL: 30/06