Infra-Estrutura de Software

Entrada / Saída

Diversidade de dispositivos

Hardware de E/S

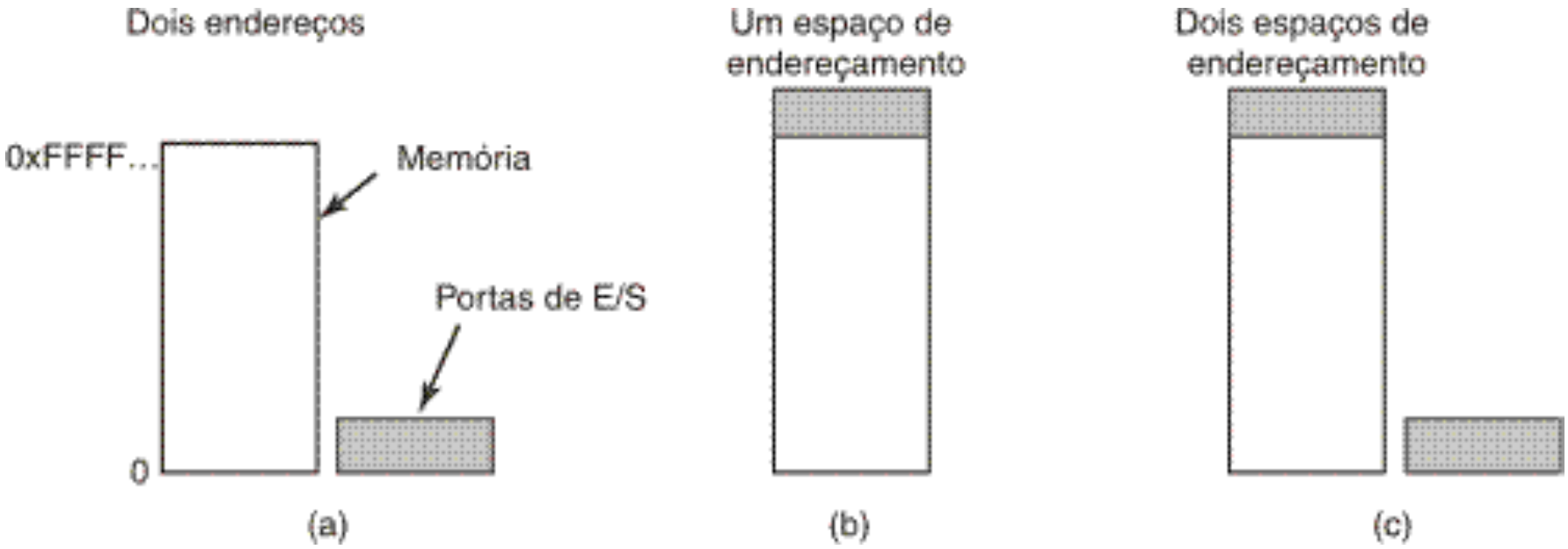
Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Scanner at 300 dpi	1 MB/sec
Digital camcorder	3.5 MB/sec
4x Blu-ray disc	18 MB/sec
802.11n Wireless	37.5 MB/sec
USB 2.0	60 MB/sec
FireWire 800	100 MB/sec
Gigabit Ethernet	125 MB/sec
SATA 3 disk drive	600 MB/sec
USB 3.0	625 MB/sec
SCSI Ultra 5 bus	640 MB/sec
Single-lane PCIe 3.0 bus	985 MB/sec
Thunderbolt 2 bus	2.5 GB/sec
SONET OC-768 network	5 GB/sec

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)

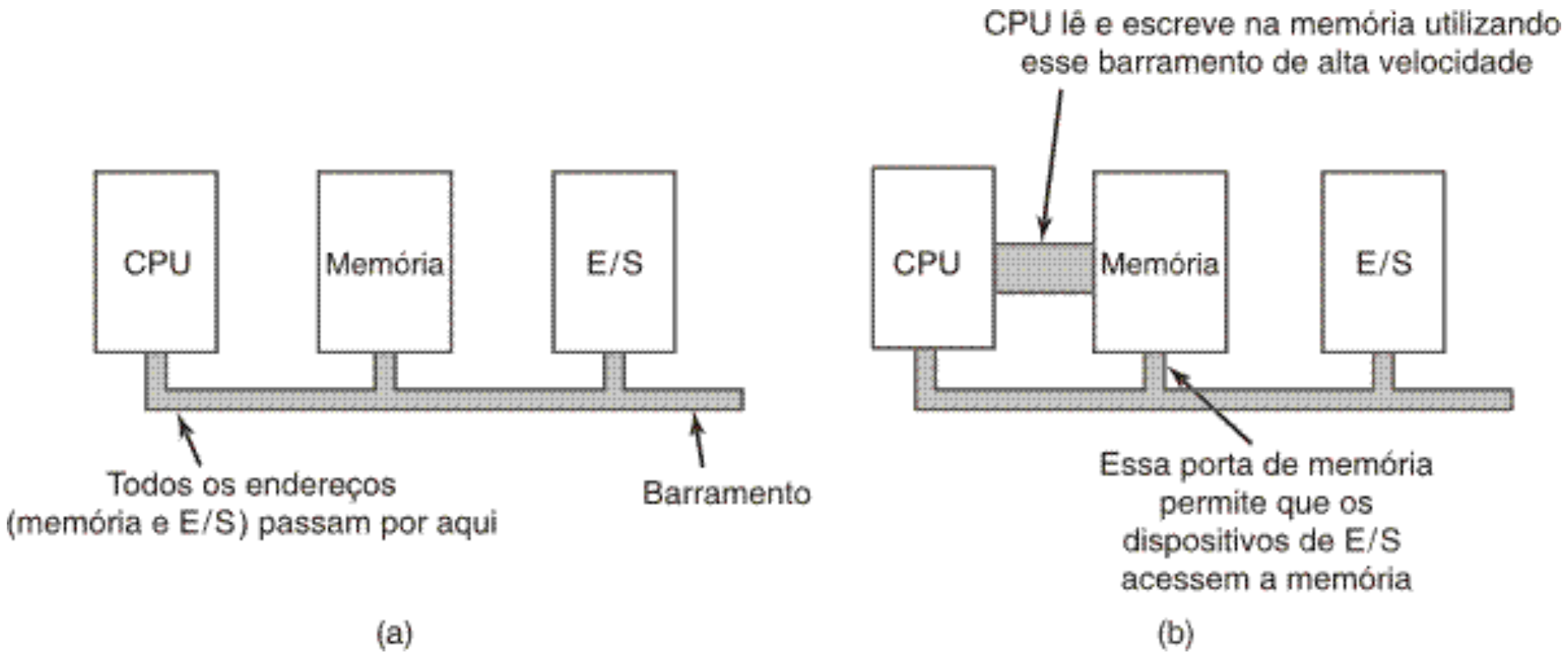
- **E/S isolada**
  - Através de instruções especiais de E/S
  - Especifica a leitura/escrita de dados numa porta de E/S
- **E/S mapeada em memória**
  - Através de instruções de leitura/escrita na memória
- **Híbrido** (ex. IBM-PC):
  - E/S mapeada em memória: memória de vídeo
  - E/S isolada: dispositivos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

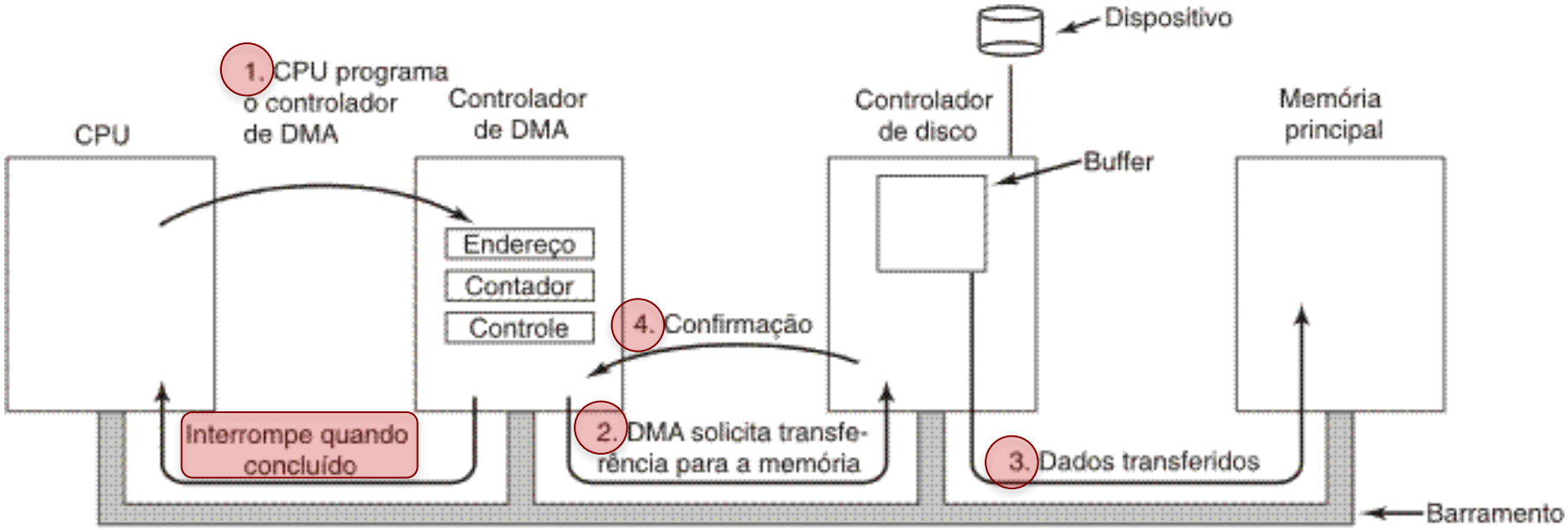
Como o processador "enxerga" a memória e os demais dispositivos ou como o processador se comunica com o seu exterior

- 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'

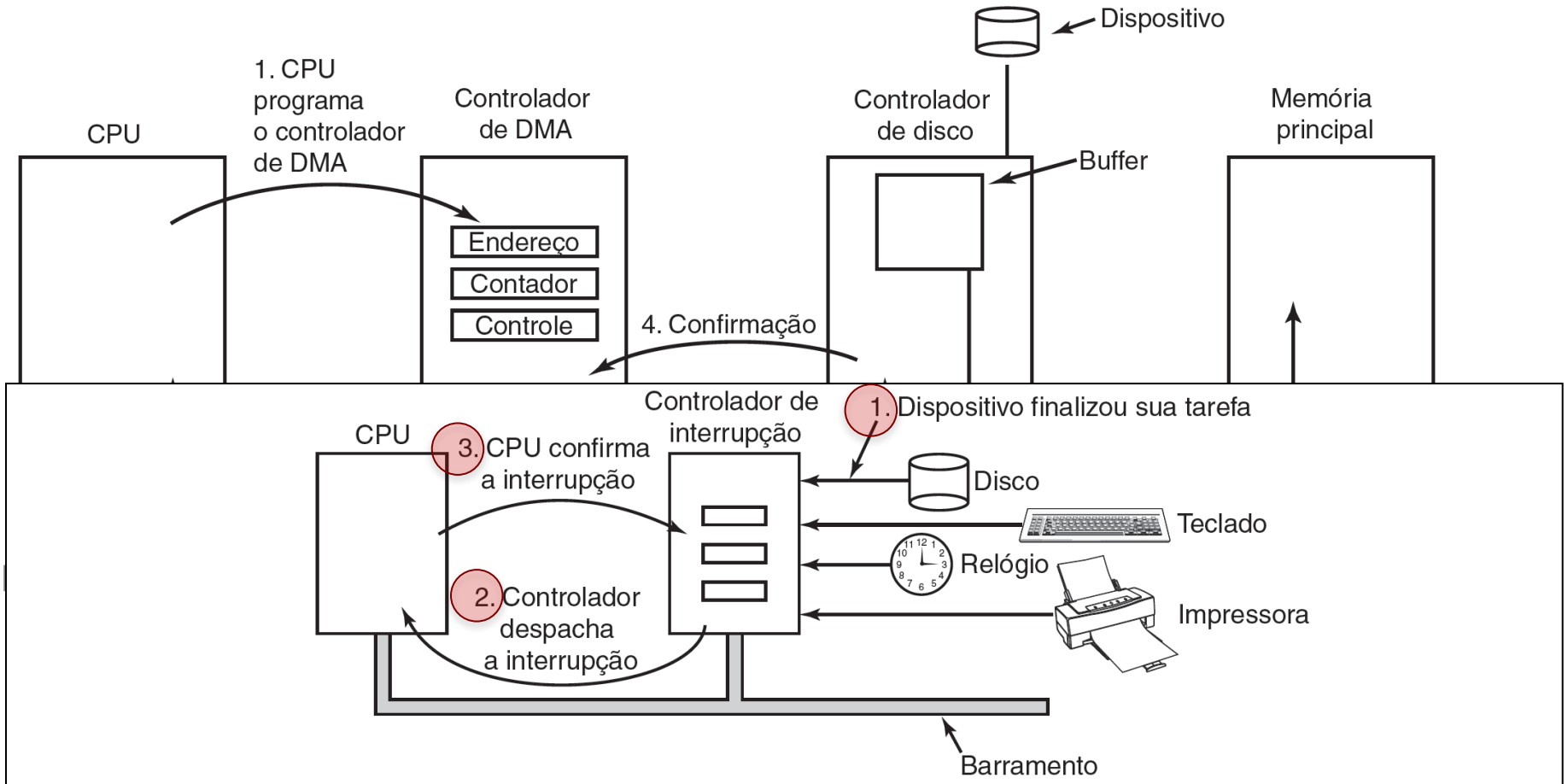


Figura 5.4 Como ocorre uma interrupção. As conexões entre os dispositivos e o controlador de interrupção atualmente utilizam linhas de interrupção no barramento, em vez de cabos dedicados.

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]

Comunicação S.O.(CPU) – Controlador

Exemplo de comunicação com dispositivo

Secretária, por favor escreva o contrato agora para que eu possa assiná-lo; estou esperando.

Pois não Sr., um momento

Programa sem interrupção

Contrato pronto, estou enviando

Ok, Obrigado

Secretária, por favor escreva o contrato agora, quando estiver pronto me avise.

Pois não Sr.

Programa com interrupção

Contrato pronto, posso enviá-lo?

Um momento, estou terminado de escrever um ofício. ... agora pode enviar o contrato, obrigado. (Após receber o contrato, o chefe continua fazendo as outras atividades.....)

Acesso Direto à Memória

CPU

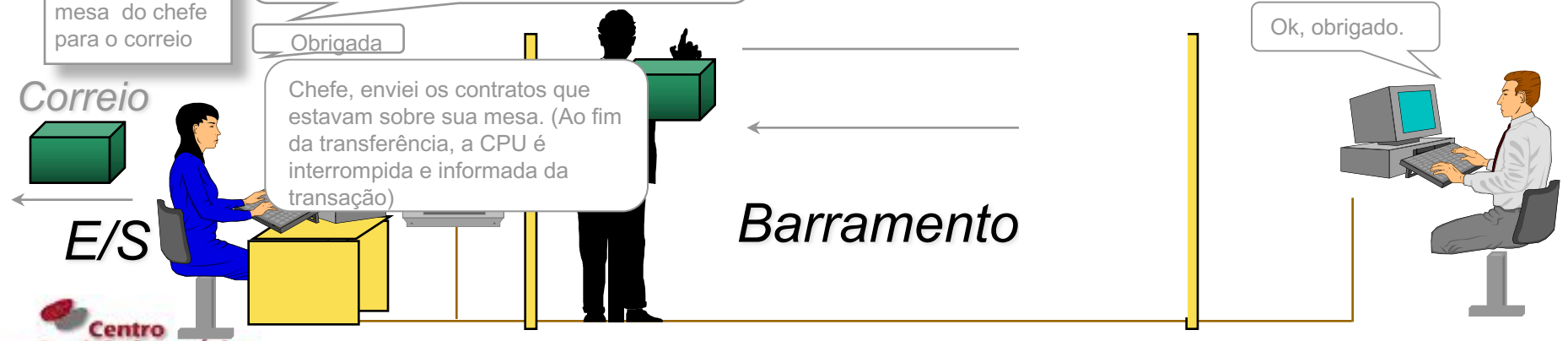
Preciso enviar os contratos que estão sobre a mesa do chefe para o correio

Auxiliar, por favor apanhe os contratos que estão na pasta sobre a mesa do chefe para mim. Por favor não incomode o chefe.

Obrigada

Chefe, enviei os contratos que estavam sobre sua mesa. (Ao fim da transferência, a CPU é interrompida e informada da transação)

Ok, obrigado.



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, diante da alta diversidade, associada a heterogeneidade):
  - 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 (*System Calls*)

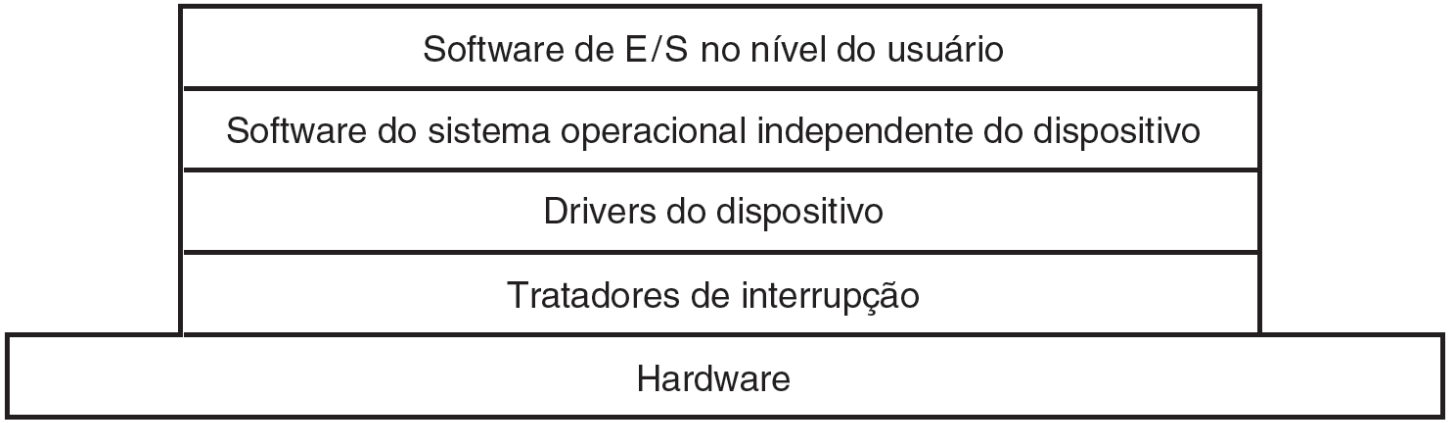
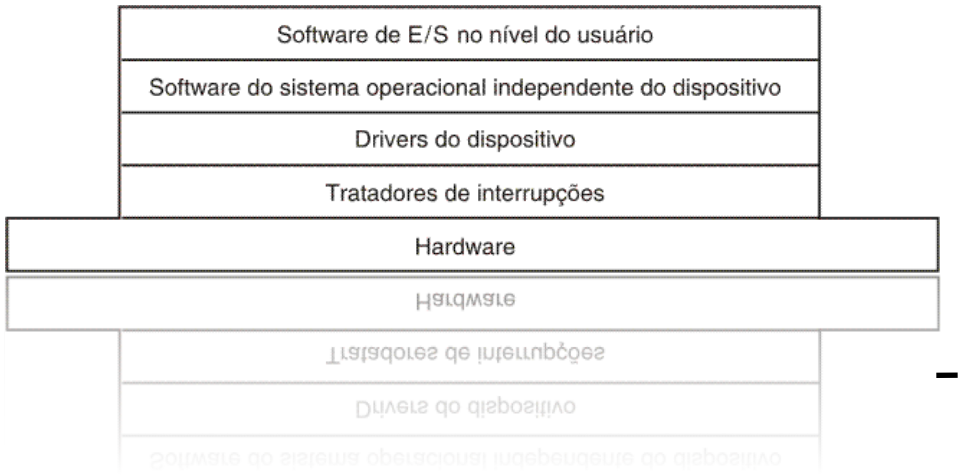
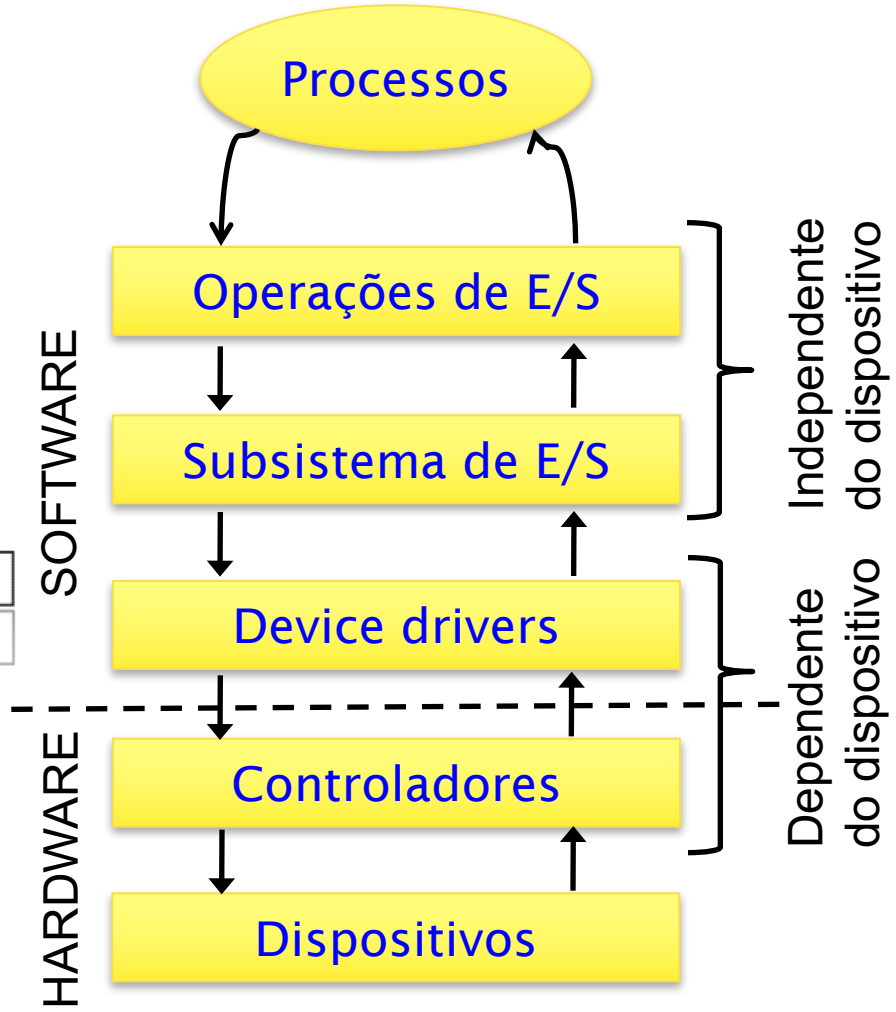


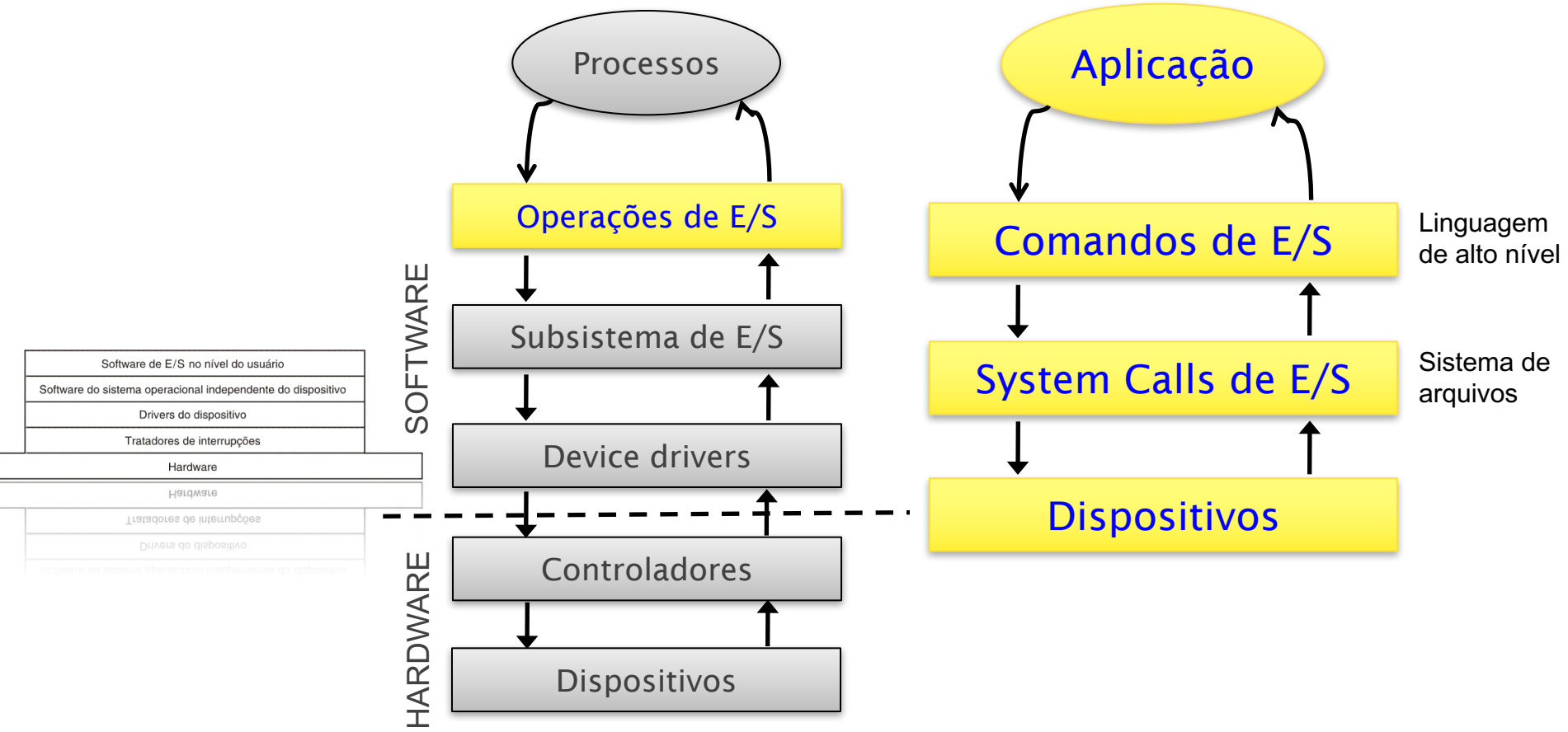
Figura 5.10 Camadas do software de E/S.

Camadas do Software de E/S





Camadas do Software de E/S

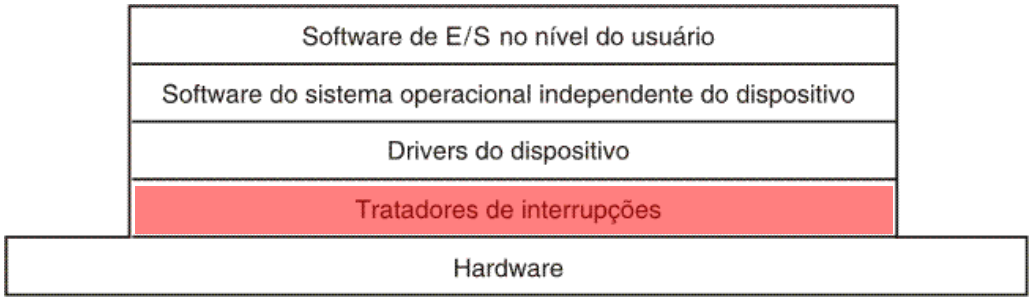


Tratador(es) de Interrupção

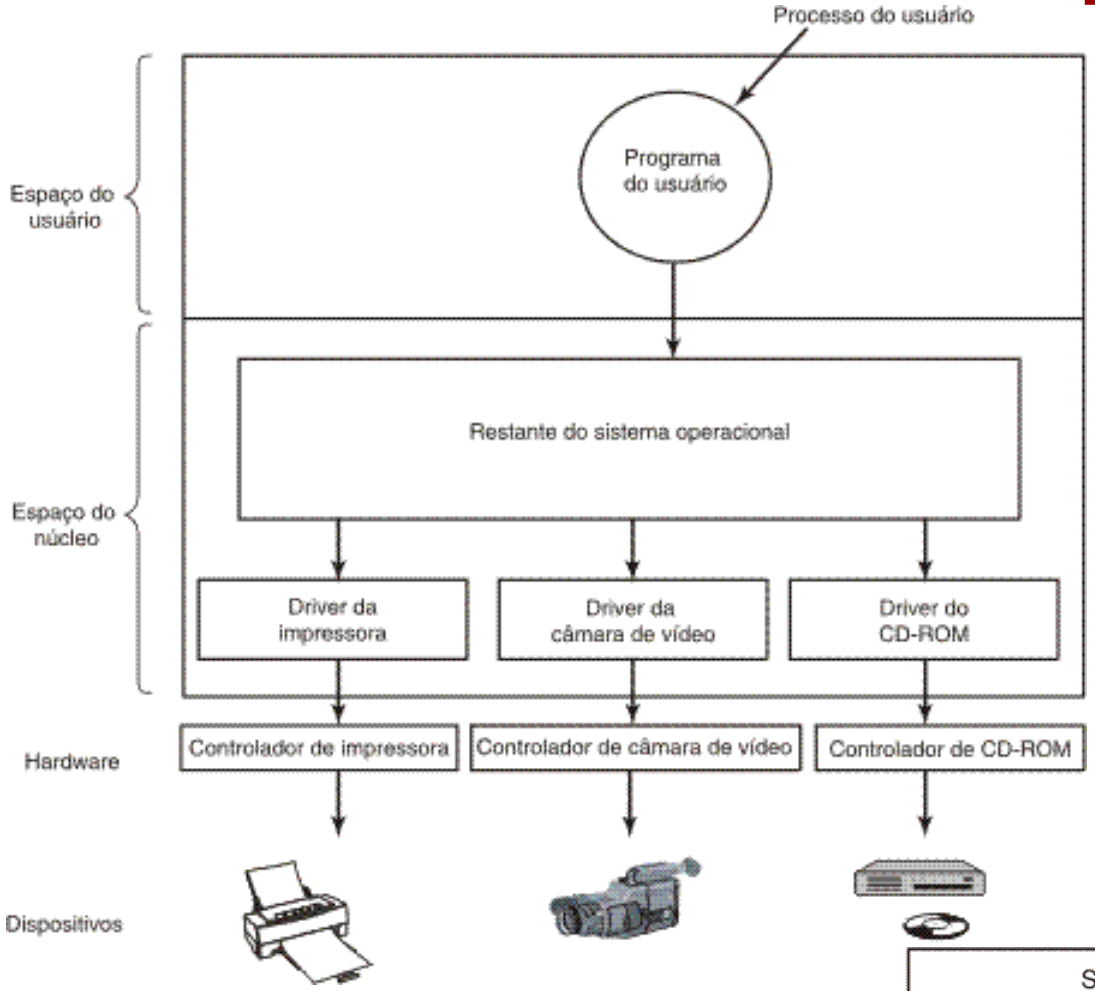
- As interrupções devem ser escondidas (transparentes) o máximo possível

  Para tanto:

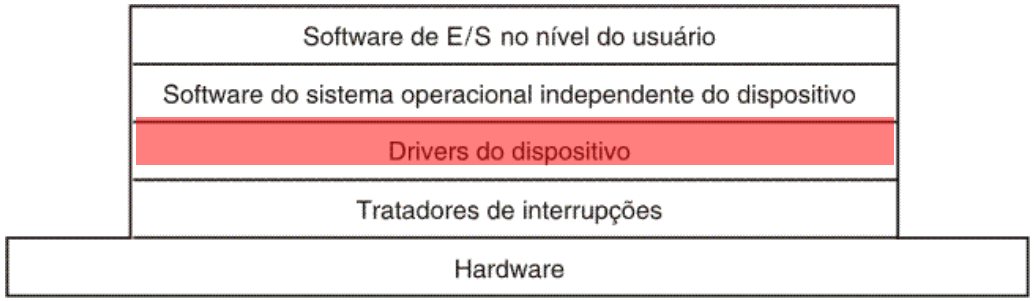
  - bloqueia o *driver* que iniciou uma operação de E/S
  - rotina de tratamento de interrupção cumpre sua tarefa
  - notifica que a E/S foi completada
  - e então desbloqueia o *driver* que a chamou



Drivers dos Dispositivos



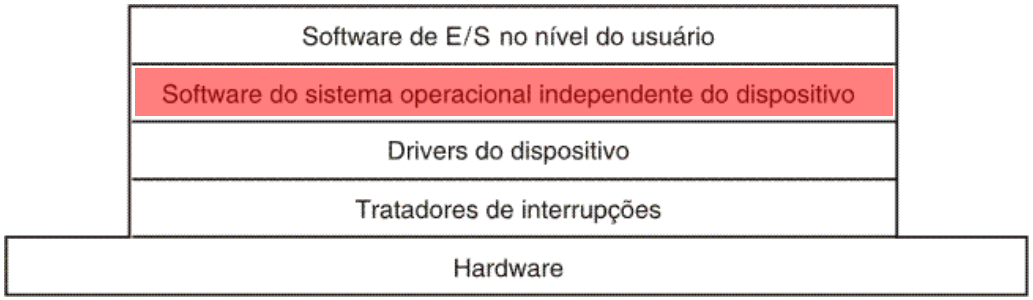
- Posição lógica dos drivers dos dispositivos
- A comunicação entre os drivers e os controladores de dispositivos é feita por meio do barramento



Software de E/S Independente de Dispositivo

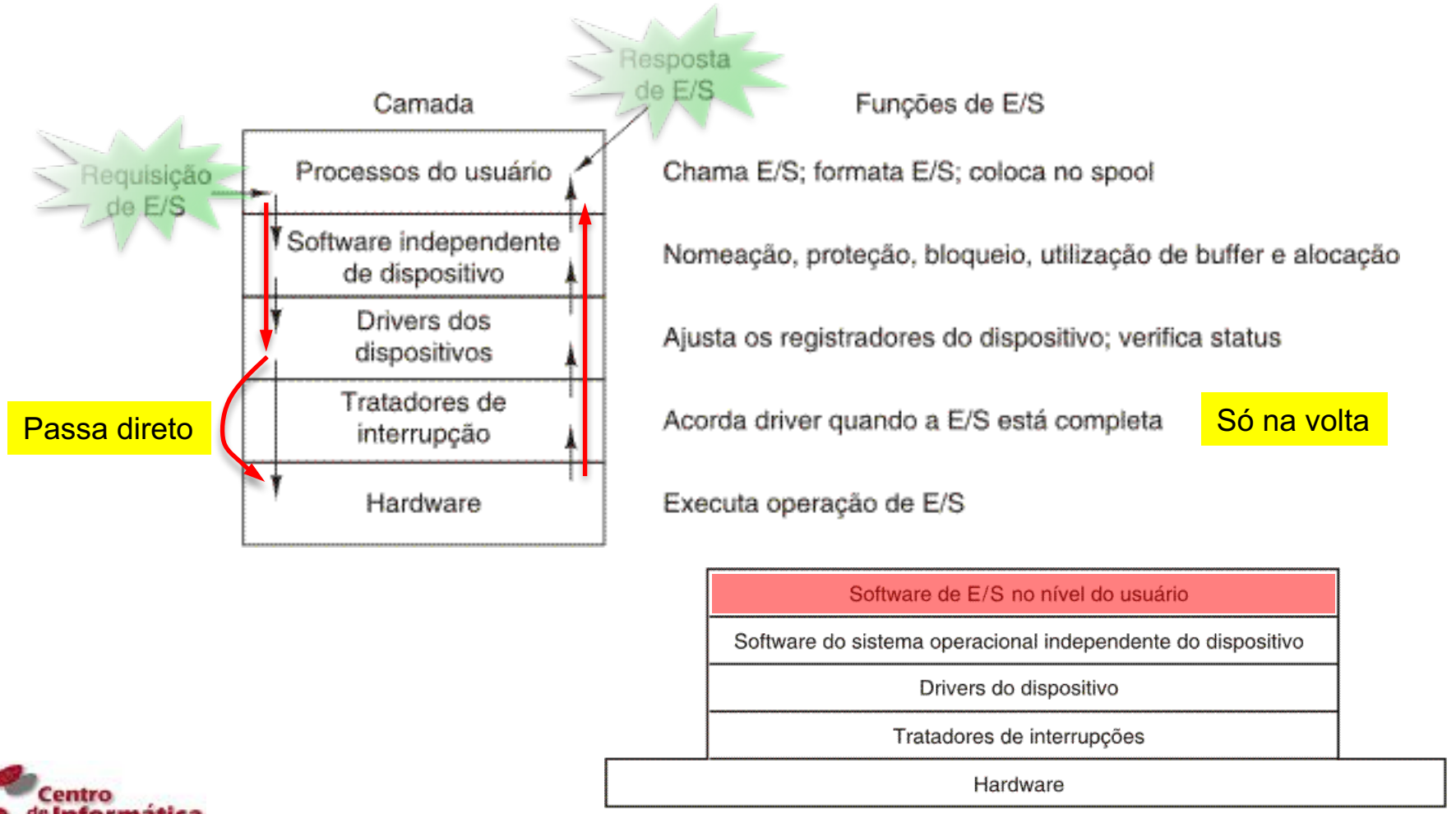
Funções do software de E/S independente de dipositivo

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



Entrada/Saída
(Curiosidades)

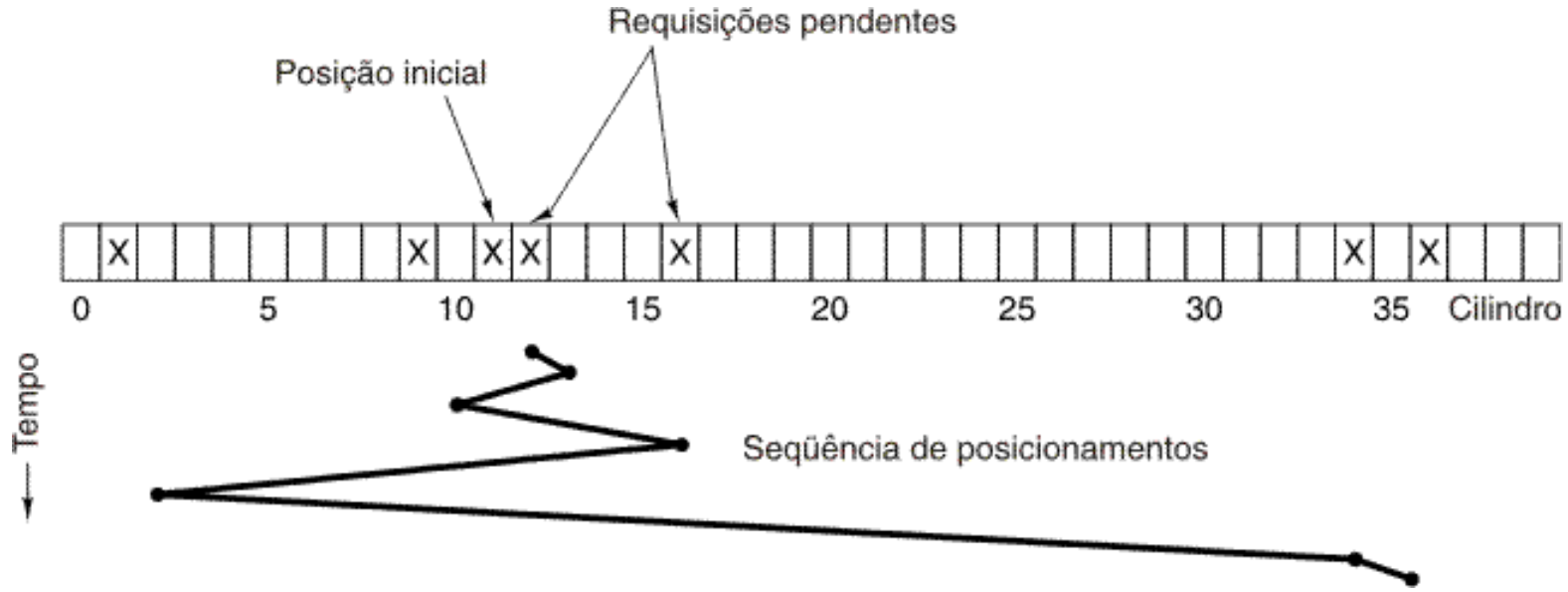
- E/S em disco
- Gerenciamento de energia

Centro de Informática UFPE

Algoritmos de Escalonamento de Braço de Disco (1)

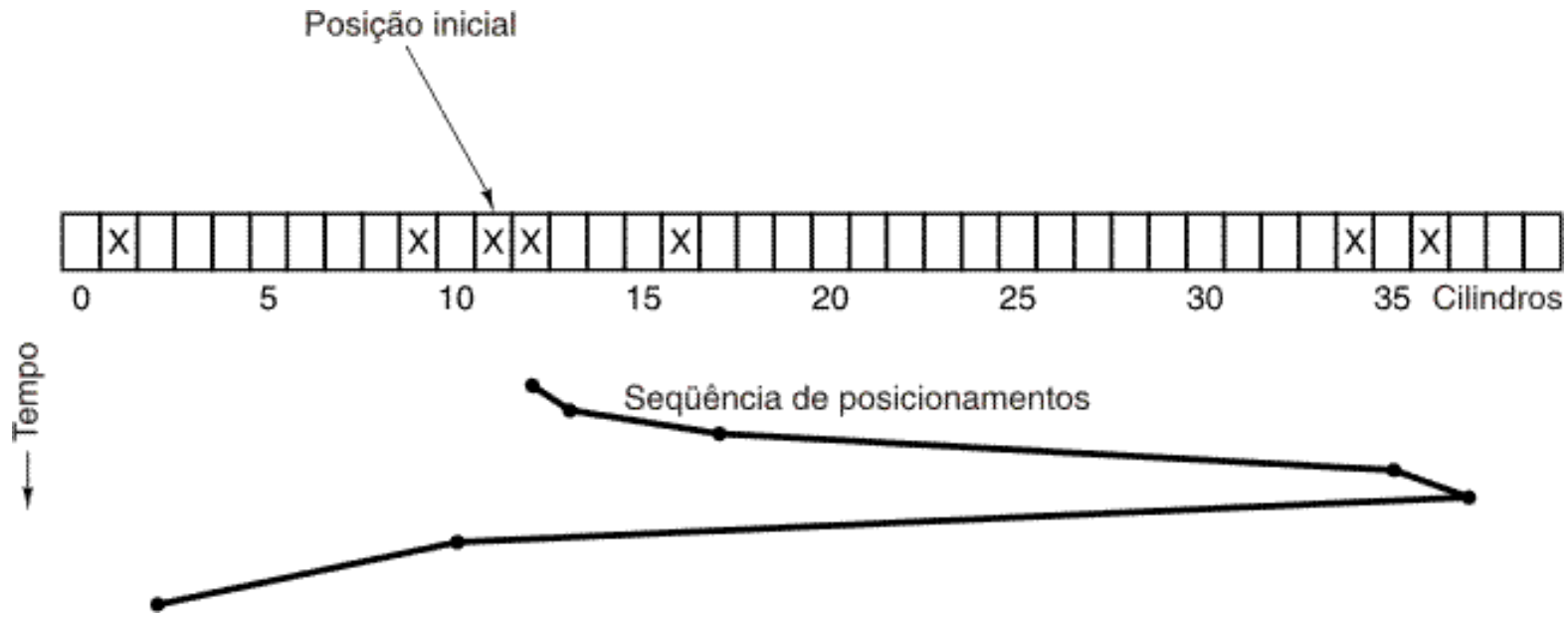
- Tempo necessário para ler ou escrever um bloco de disco é determinado por 3 fatores
  1. tempo de posicionamento
  2. atraso de rotação
  3. tempo de transferência do dado
- Tempo de posicionamento domina
- Checagem de erro é feita por controladores

Algoritmos de Escalonamento de Braço de Disco (2)



Algoritmo de escalonamento de disco *Posicionamento Mais Curto Primeiro* (SSF)

Algoritmos de Escalonamento de Braço de Disco (3)



O algoritmo do elevador para o escalonamento das requisições do disco

Entrada/Saída
(Curiosidades)

- ✓ E/S em disco
- Gerenciamento de energia

Gerenciamento de Energia (1)

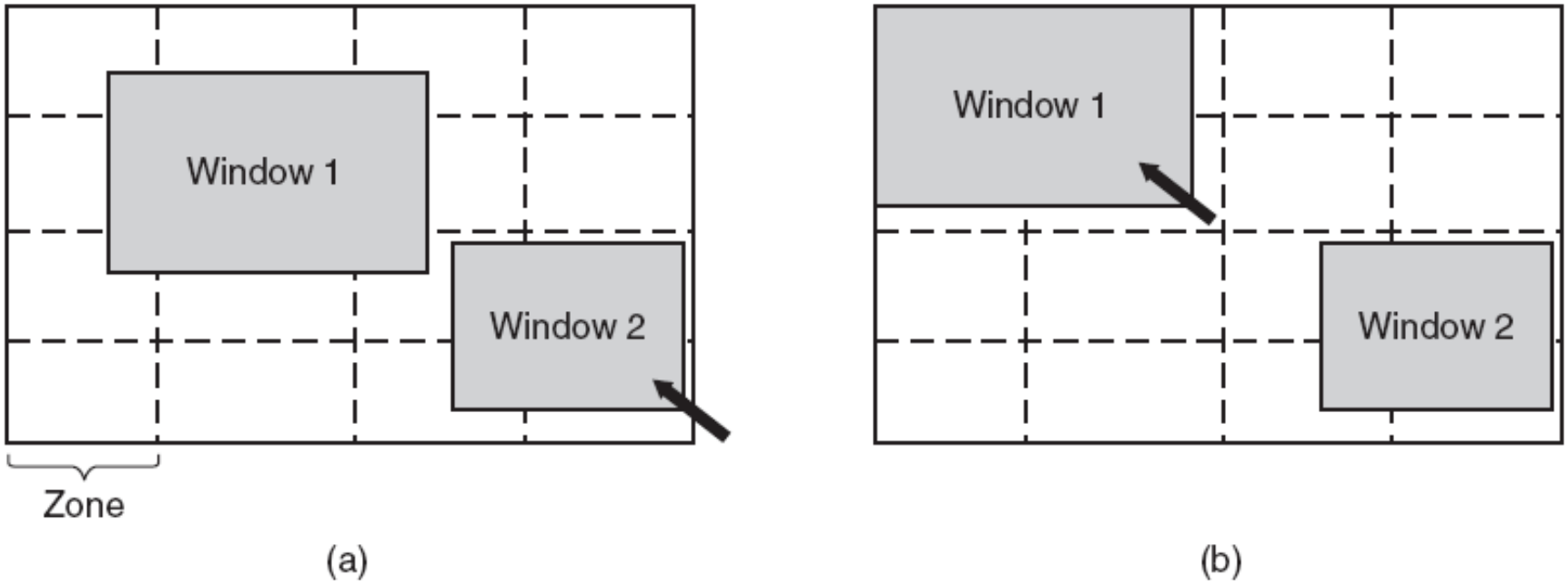
Dispositivo	Li *et al.* (1994)	Lorch e Smith (1998)
Monitor de vídeo	68%	39%
CPU	12%	18%
Disco rígido	20%	12%
Modem		6%
Som		2%
Memória	0,5%	1%
Outros		22%

Tecnologia mais avançada →

Aumento do desempenho →

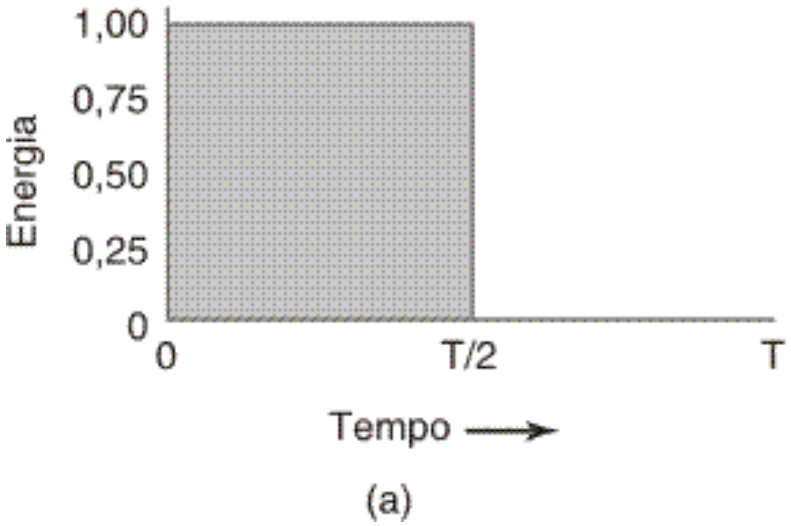
Consumo de energia de várias partes de um laptop

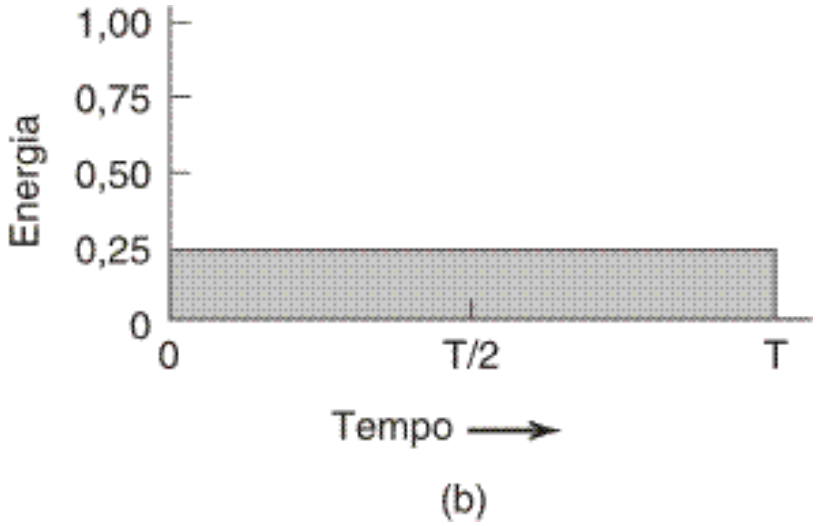
Gerenciamento de Energia (2): O uso de zonas para iluminação do monitor de vídeo



(a) Janela 2 selecionada e não se move.
(b) Janela 1 selecionada e se move para reduzir o número de zonas iluminadas.

Gerenciamento de Energia (3)





a. Execução em velocidade máxima do relógio
b. Cortando a voltagem pela metade
   - corta a velocidade do relógio também pela metade,
   - consumo de energia cai para 4 vezes menos

Gerenciamento de Energia (4): Impactos na Entrada/Saída

- Dizer aos programas para usar menos energia
  - pode significar experiências mais pobres para o usuário

- Exemplos
  - muda de saída colorida para preto e branco
  - reconhecimento de fala com vocabulário reduzido
  - menos resolução ou detalhe em uma imagem

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

Sistema de Software

- É composto por uma **sequência** de instruções, que é interpretada e executada por um processador

Sistema de Software Distribuído

- É composto por uma **sequência** de instruções, que é interpretada e executada por um processador
- É composto por instruções concorrentes ou paralelas, que são interpretadas e executadas por um ou mais processadores

- **Por que distribuir?**

Por que distribuir?

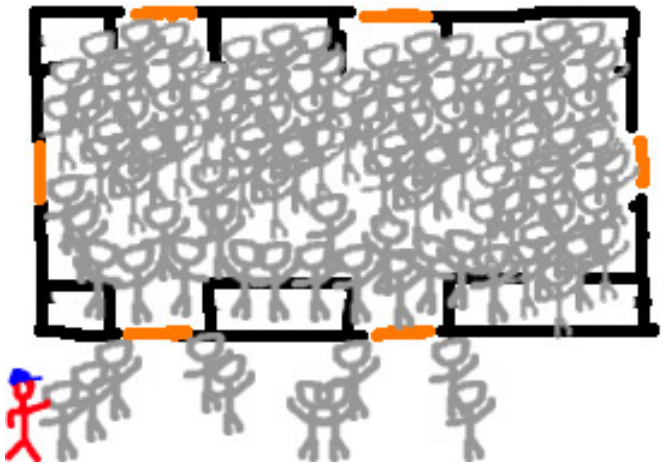
Manutenção

Dividir para conquistar

Por que distribuir?

Acomodação às capacidades das máquinas disponíveis

Pode ser “impossível” instalar e executar um sistema de grande porte em uma única máquina



Melhor desempenho e Escalabilidade
Concorrência, paralelismo

- Manutenção
  - Dividir para conquistar
- Acomodação às capacidades das máquinas disponíveis
  - Pode ser "impossível" instalar e executar um sistema de grande porte em uma única máquina

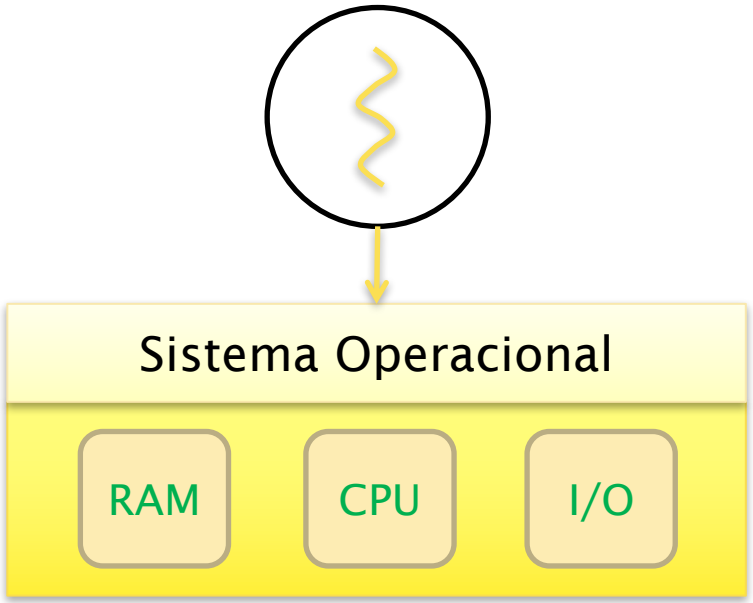
Por que distribuir?

Tolerância a falhas

Analogia-exemplo: pesquisas na Guerra Fria ➡ Internet

- Manutenção
  - Dividir para conquistar
- Acomodação às capacidades das máquinas disponíveis
  - Pode ser "impossível" instalar e executar um sistema de grande porte em uma única máquina
- Melhor desempenho e Escalabilidade
  - Concorrência, paralelismo

Evolução



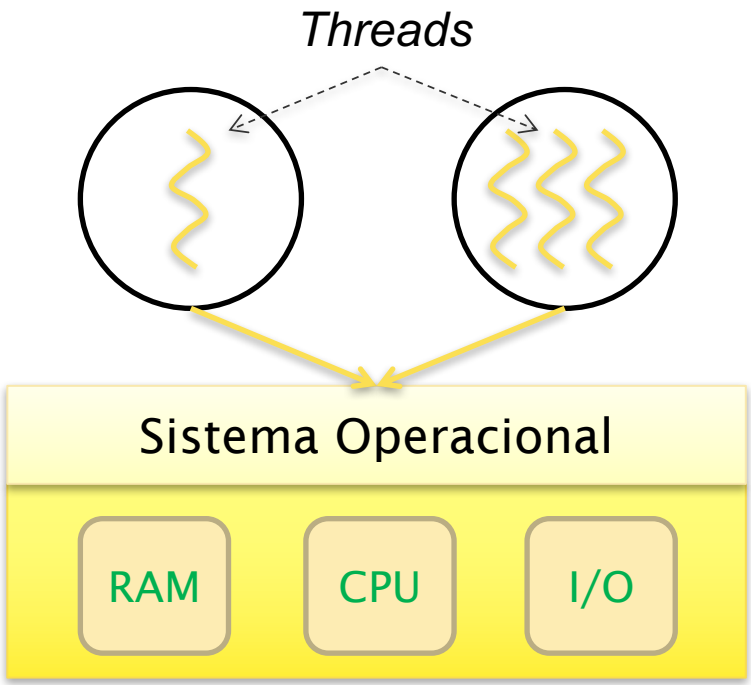
Processo

Máquina Abstrata

Computador

Computação Centralizada

Evolução



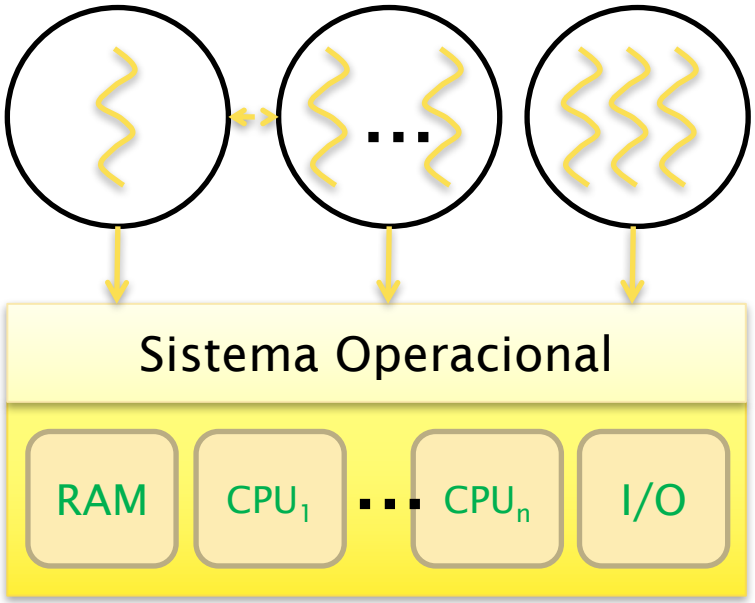
Processos

Máquina Abstrata

Computador

Concorrência

Evolução



Processos

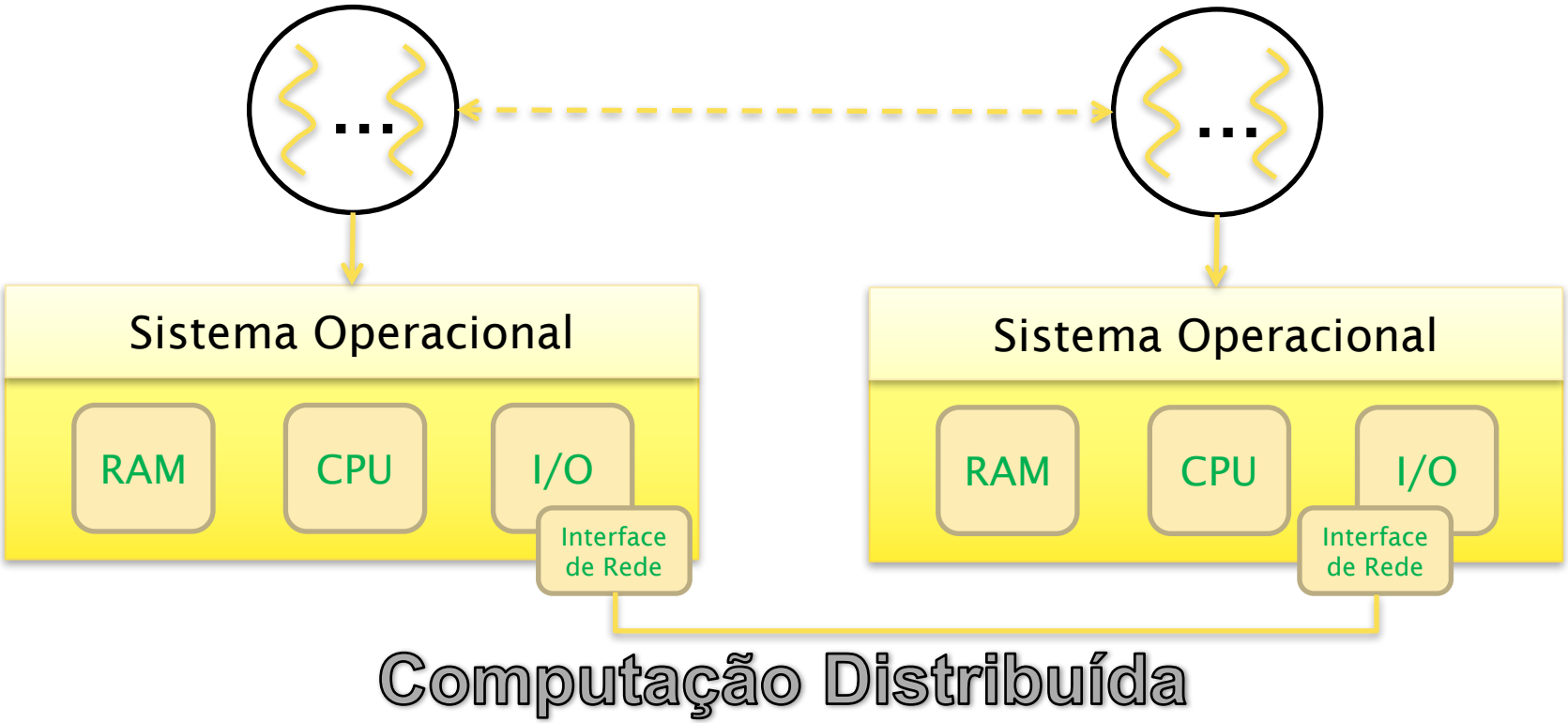
Máquina Abstrata

Computador

Computação Paralela

(e comunicação entre processos (IPC) concorrentes/paralelos)

Evolução

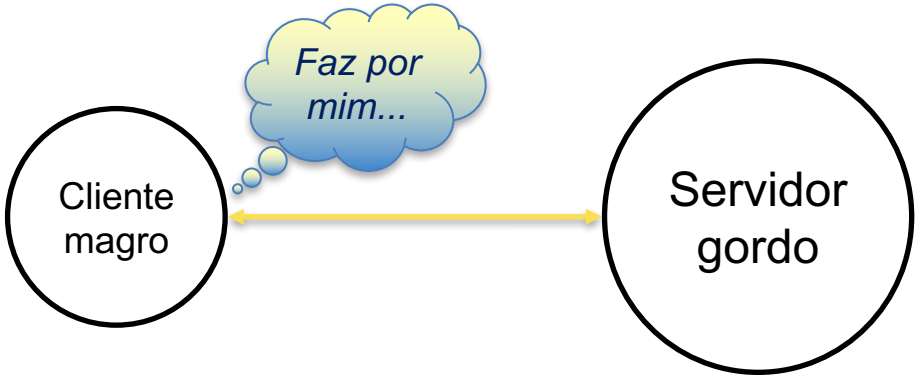

...
...
Sistema Operacional
RAM
CPU
I/O
Interface de Rede
Sistema Operacional
RAM
CPU
I/O
Interface de Rede
Computação Distribuída

Tendências-chave

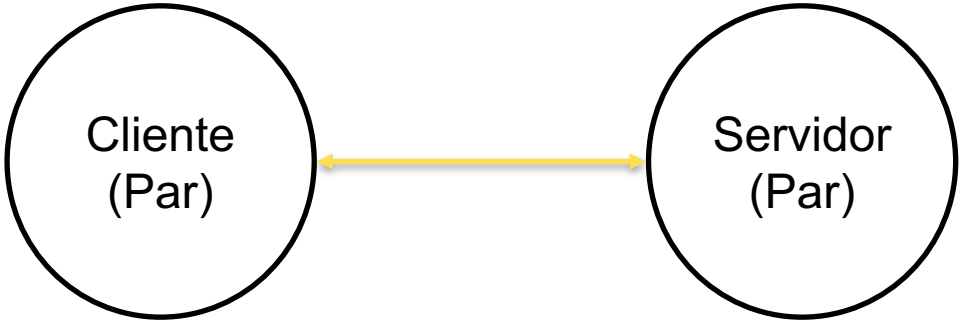
- O que move os **Sistemas Distribuídos** hoje?
  - Pervasividade das redes
  - Computação móvel e ubíqua
  - Importância crescente de sistemas multimídia (distribuídos)
  - Sistemas distribuídos como utilidade (serviço) – *Cloud Computing* e seus modelos (IaaS, PaaS, SaaS)
- Principal motivação: **compartilhamento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

Compartilhamento de Recursos

- Passado



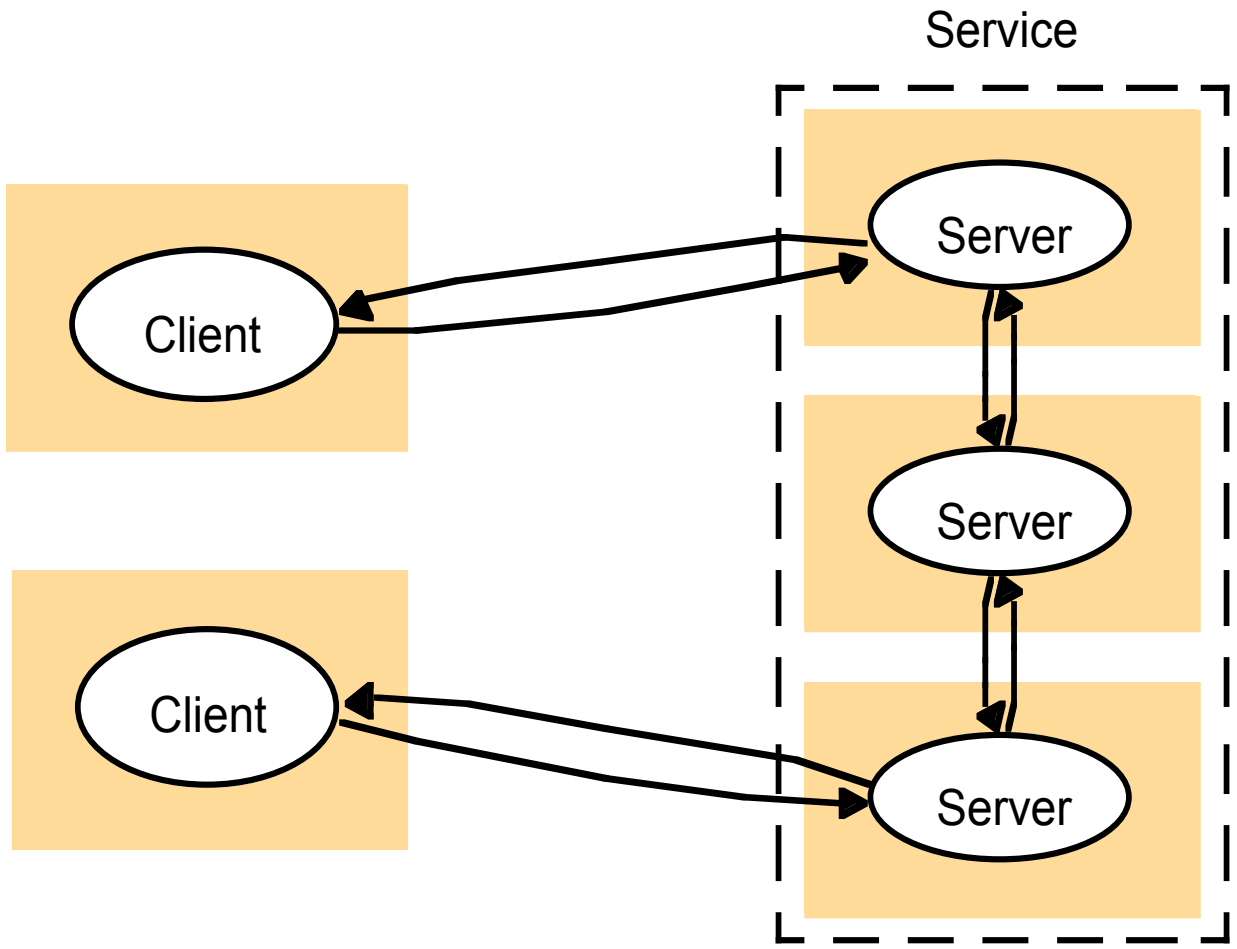
- Presente



"Equilíbrio" de recursos (compartilhamento mais "verdadeiro" e justo)

Arquitetura Cliente-Servidor

Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5
© Pearson Education 2012



Um **serviço** fornecido por múltiplos **servidores**

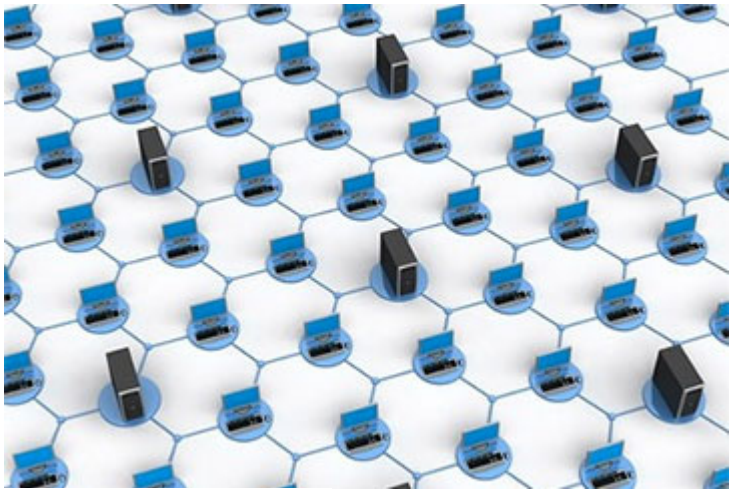
O maior problema

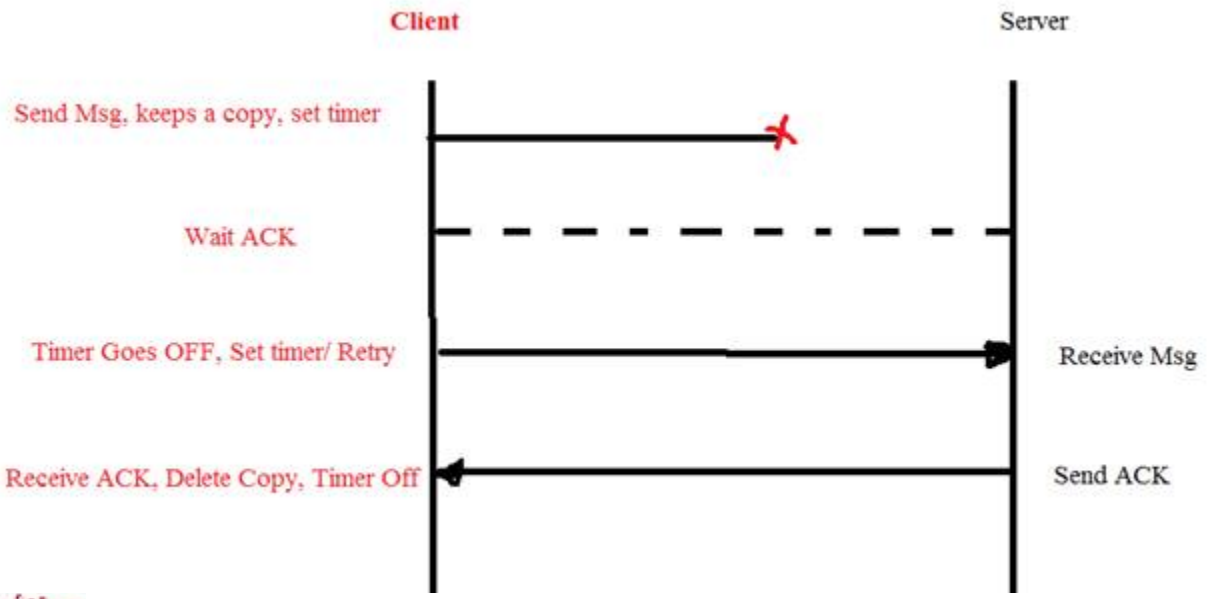
Mensagens se perdem, máquinas param de funcionar, discos corrompem dados…
Mas os sistemas (distribuídos) parecem funcionar a maior parte do tempo: Google, Facebook, Amazon etc.

Como lidar com falhas?

Algums técnicas:

- Replicação
- Repetição de tentativa de comunicação...





Definições

"*Um sistema distribuído é aquele que faz você parar de ter o trabalho realizado quando uma máquina da qual você nunca ouviu falar* ***falha***".
Leslie Lamport

Um **sistema distribuído** é aquele no qual componentes localizados em uma rede de computadores se comunicam e coordenam suas ações através da passagem (troca) de mensagens

George Coulouris et al., 2012

Características marcantes

- Concorrência de componentes
- Falta de um relógio global
- Componentes falham de forma independente – falha parcial

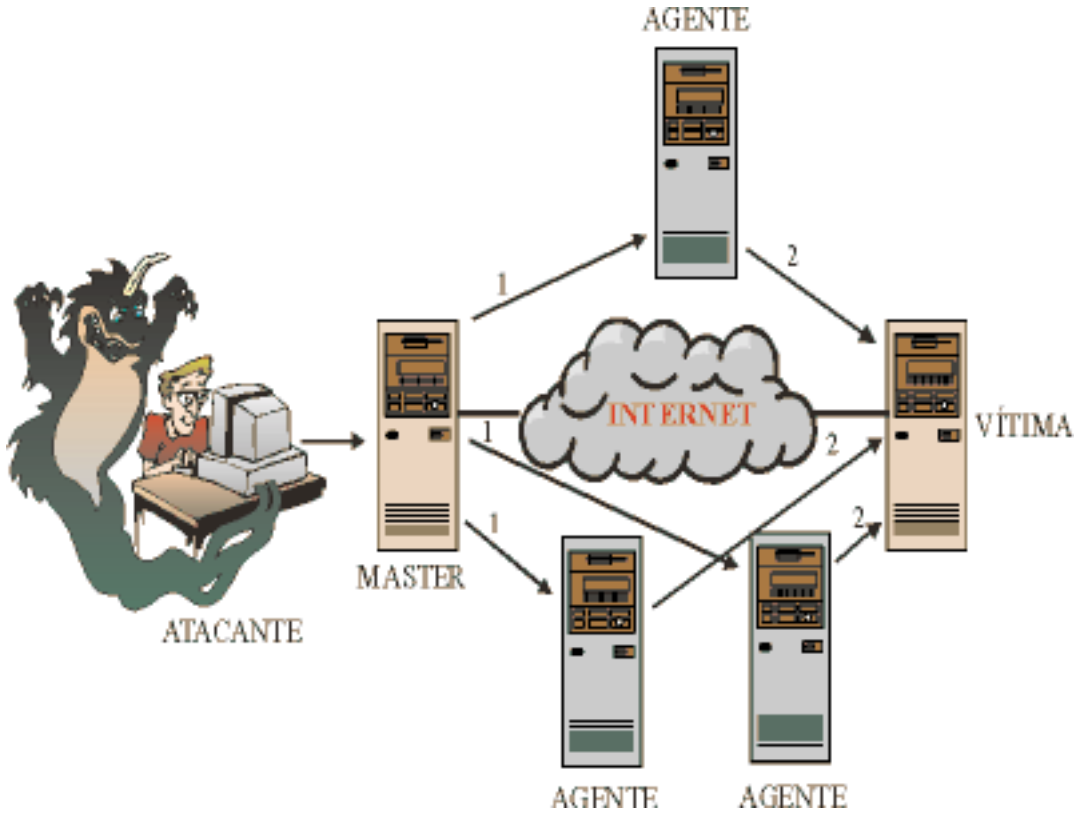
Objetivos de um projeto de SD

- Eficiência – desempenho produtivo
- Robustez – resistência a falha
  - Disponibilidade: o sistema está no ar quando preciso (instante de tempo)
  - Confiabilidade: o sistema não falha por um longo período de tempo
    - E quando falha, a falha não provoca uma "catástrofe"
- Consistência – mesma visão de dados

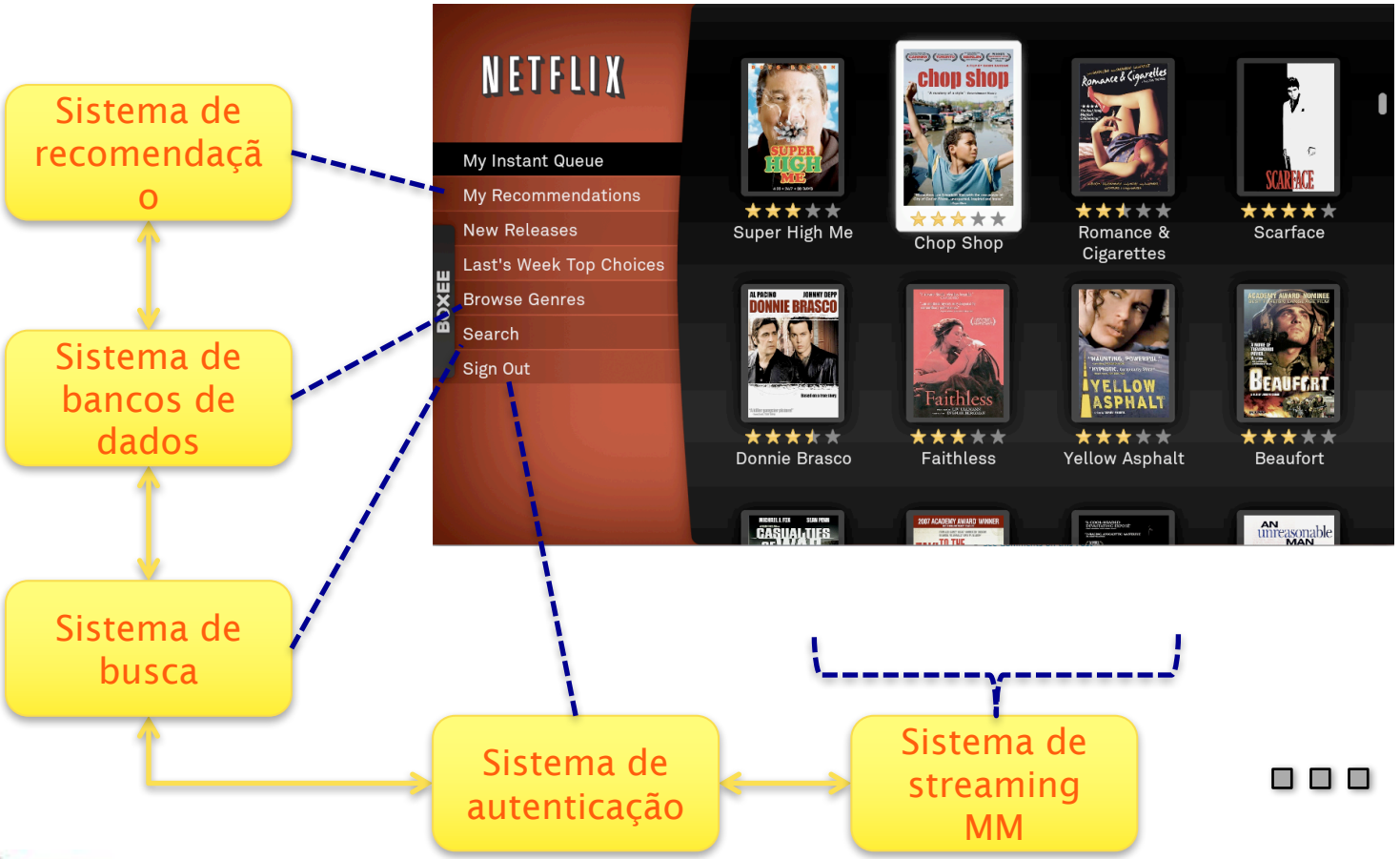
Objetivo: Eficiência

Para o bem, baseado em paralelismo

Para o mal, DDoS...



Sistemas multimídia distribuídos



Conceito-chave

- Distribuição
- Comunicação
- Complexidade
- Heterogeneidade

- **Transparência**

Tipos de Transparência

- **Localização**: esconde onde o recurso está localizado
- **Acesso**: operações idênticas para acesso local e remoto
- Migração: esconde que um recurso pode se mover para outra localização
- Relocação: esconde que um recurso pode ser movido para outra localização enquanto está em uso
- Concorrência: compartilhamento de recursos sem interferência entre processos concorrentes
- Falha: esconde a falha e recuperação de um recurso
- Replicação: esconde de usuários ou programadores de aplicação a existência de réplicas de recursos

- Localização + acesso = transparência de rede

Últimas Datas Importantes

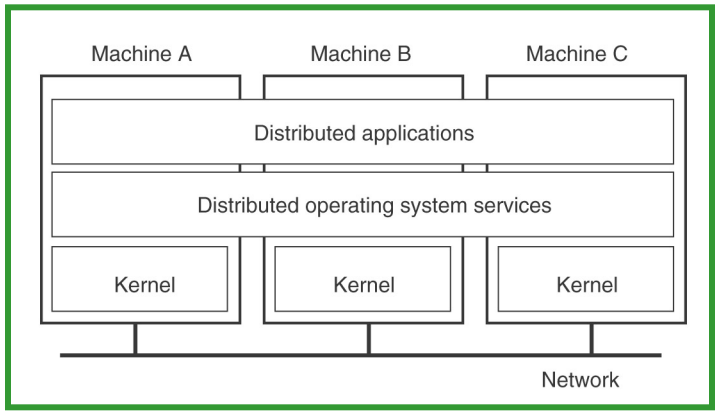
- 30/jun: **2o. EE** - Prova teórica
- 05/jul: **3o. EE** - Lista de concorrência – apresentação no lab G5 (10-12h)
- 12/jul: **Revisão** de notas
- 14/jul: **Prova Final**

http://www.cin.ufpe.br/~cagf/if677/2017-1/slides/

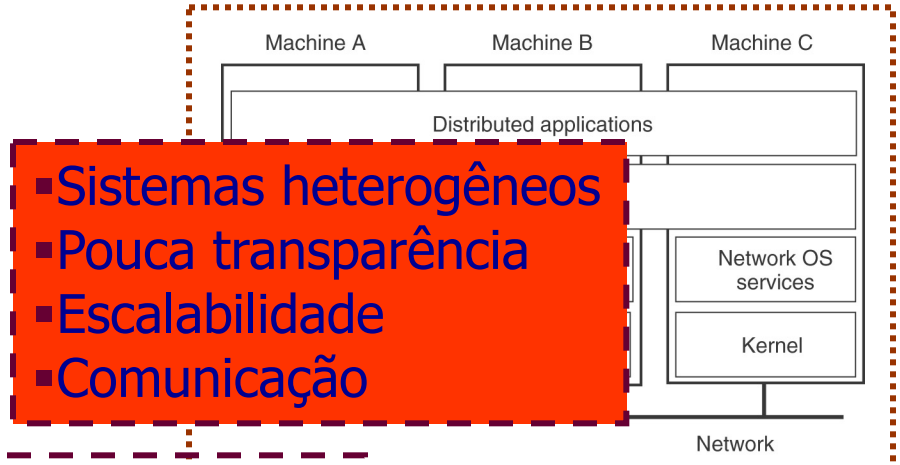
Infra-estruturas de software para Sistemas Distribuídos

- 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

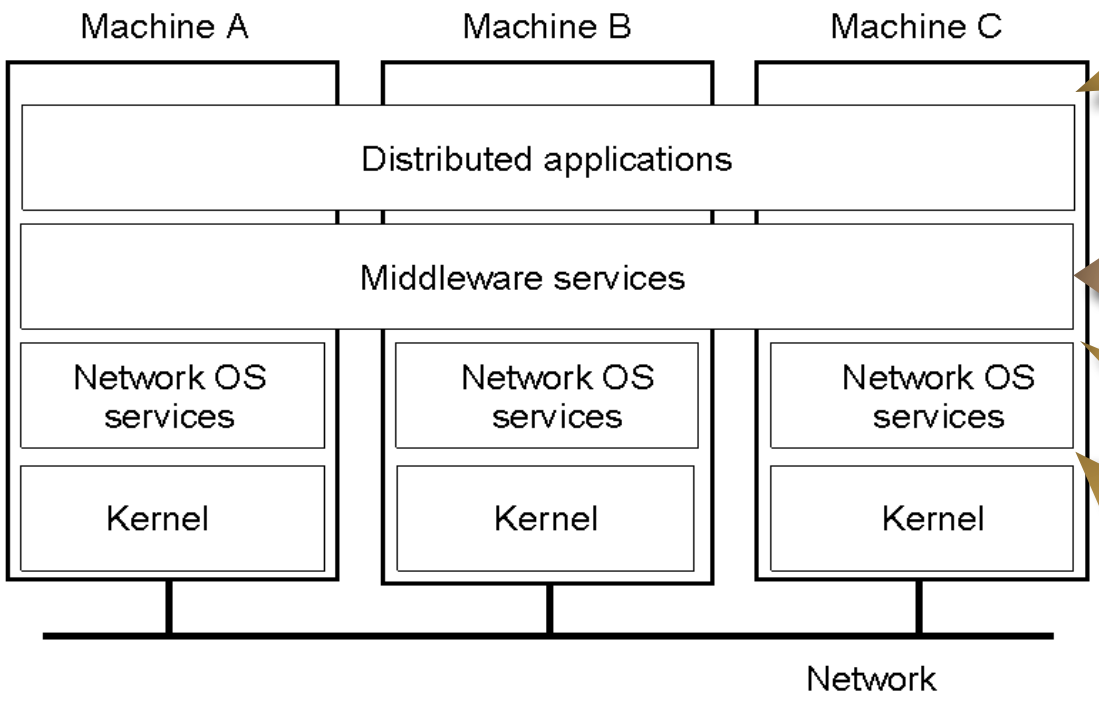


- Sistemas heterogêneos
- Pouca transparência
- Escalabilidade
- Comunicação

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

Middleware

Necessidades: quem atende o quê?



- Lógica do negócio

- Integração de Sistemas Heterogêneos
- Interoperabilidade destes sistemas
- Portabilidade

- Comunicação

Uso e gerência de recursos de hardware:

CPU
Memória
I/O

Middleware: definição

- um conjunto reusável, expansível de serviços e funções …
- que são comumente necessários por parte de várias aplicações distribuídas …
- para funcionarem bem em um ambiente de rede

Infra-Estrutura de SW para SDs

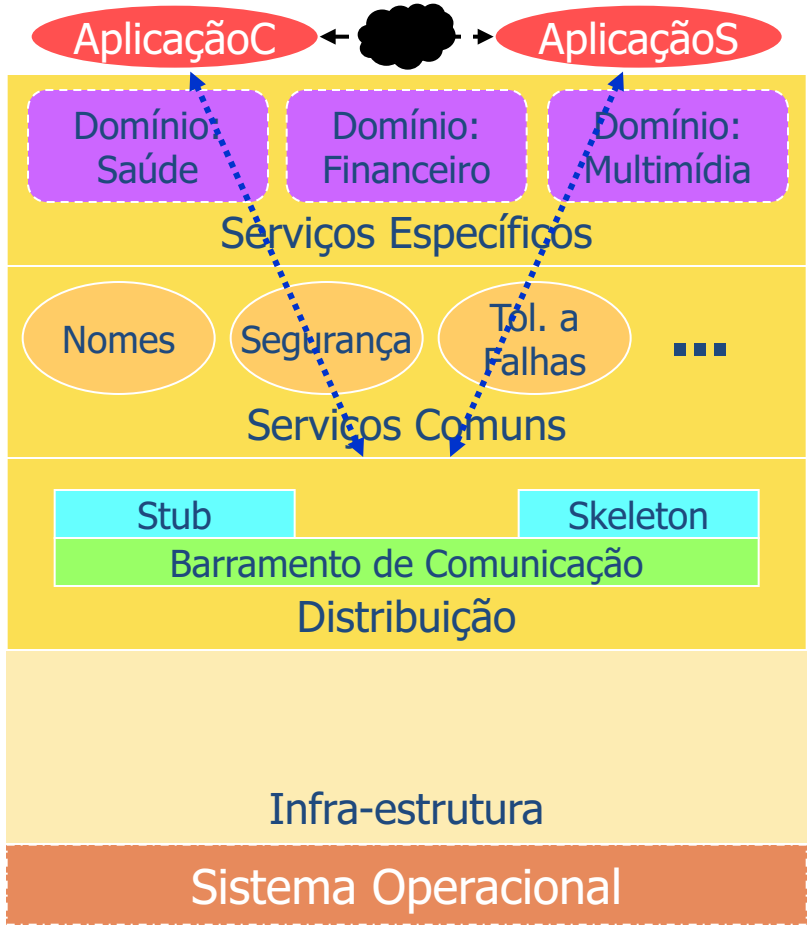
Middleware: Arquitetura Básica em Camadas

Middleware: coleção de serviços (serviços de middleware) fornecidos através de interfaces padrões (APIs) – visão "unificada" de redes e engenharia de software, respectivamente

+ formado sobre camadas de infra-estrutura

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



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() );
  }
}
```

Comunicação forma o *coração* de um sistema distribuído

Uma **abstração** de comunicação se encontra em RPC, que permite que clientes façam chamadas remotas em servidores

O *pacote* RPC envolve o uso de *timeout/retry* e *acknowledgment* (ACK) para entregar um serviço que se assemelha a uma chamada de procedimento local

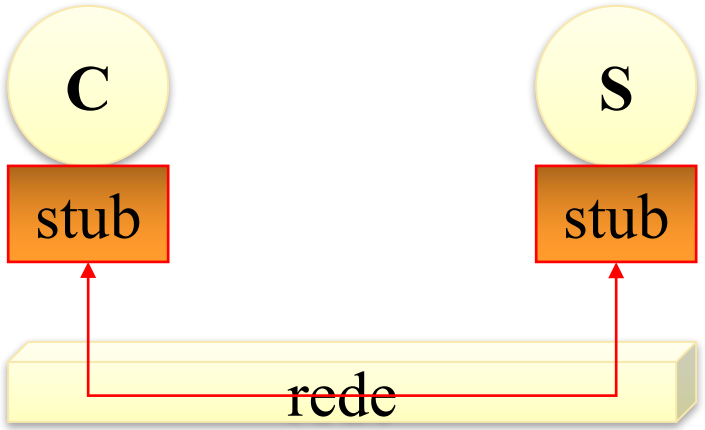
Chamada de Procedimentos Remotos

- 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

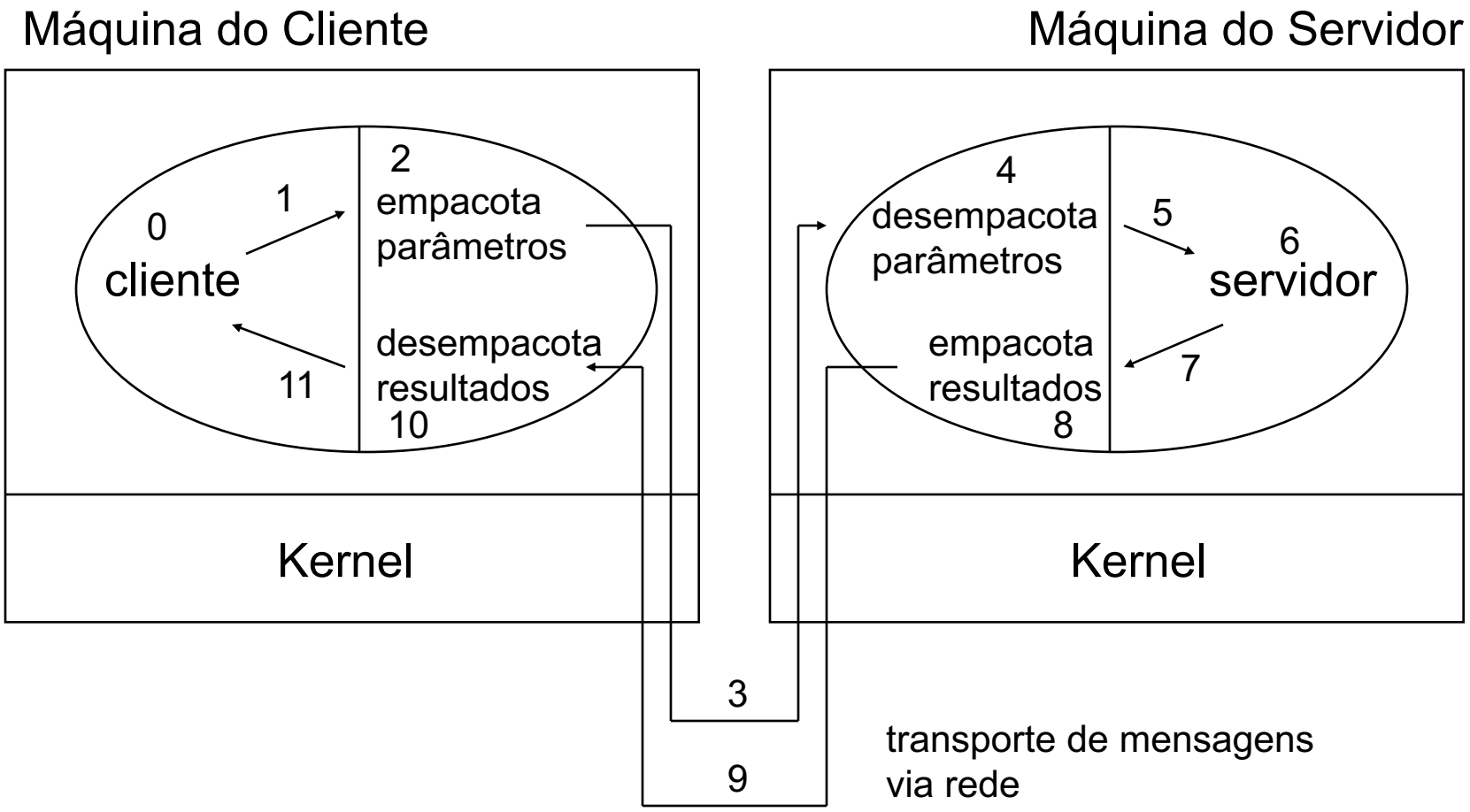
RPC: processamento de interface

- **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

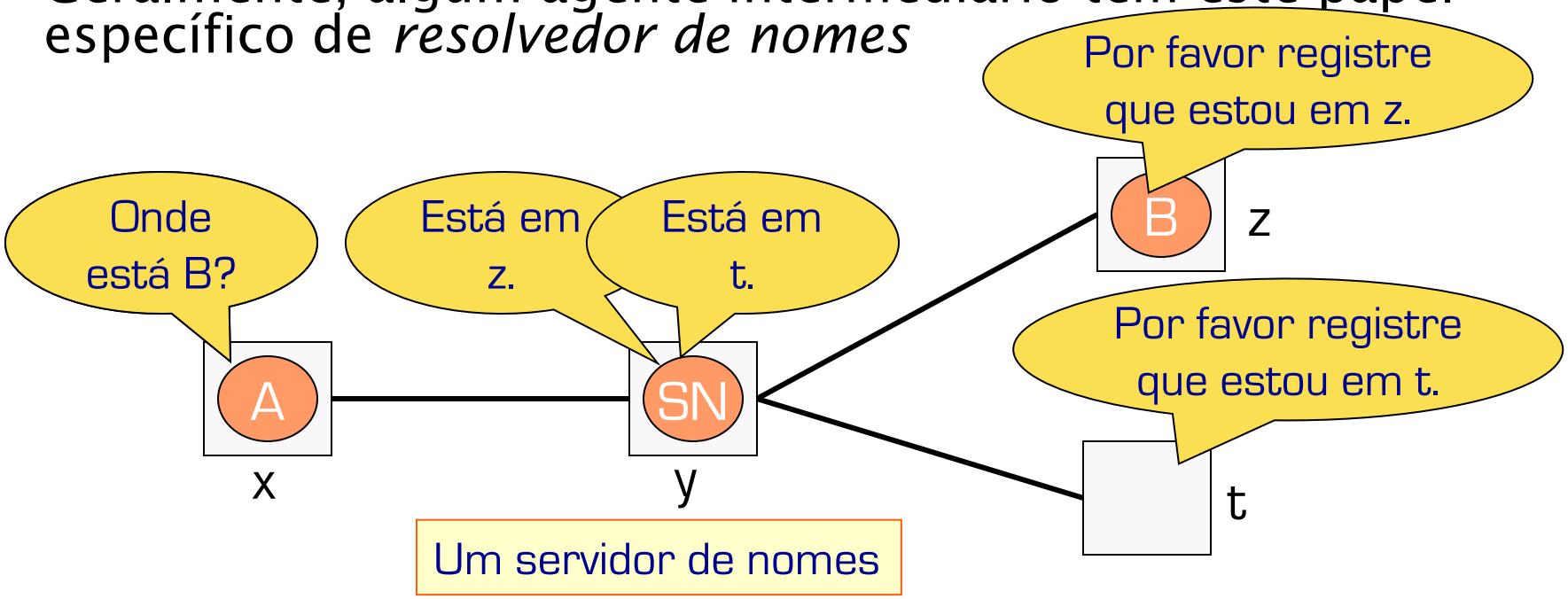


RPC: ligação (antes da comunicação...)

- 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)

Últimas Datas Importantes

- 30/jun: **2o. EE** - Prova teórica
- 05/jul: **3o. EE** - Lista de concorrência – apresentação no lab G5 (10-12h)
- 12/jul: **Revisão** de notas
- 14/jul: **Prova Final**

http://www.cin.ufpe.br/~cagf/if677/2017-1/slides/