

Infra-Estrutura de Software

Introdução

Objetivos das duas próximas aulas

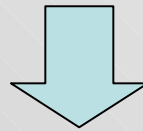
- Ao final da aula de quinta-feira você deverá ser capaz de:
 - **Descrever** os componentes básicos de um computador
 - **Explicar** como o sistema operacional evita que aplicações precisem interagir diretamente com eles
 - **Explicar** sucintamente a história dos sistemas operacionais
 - **Explicar** e **comparar** as principais organizações para sistemas operacionais

Computador Moderno

- Componentes físicos (**hardware**)
 - Um ou mais processadores
 - Memória
 - Discos
 - Impressoras
 - Vários outros dispositivos de E/S (tela, mouse...)

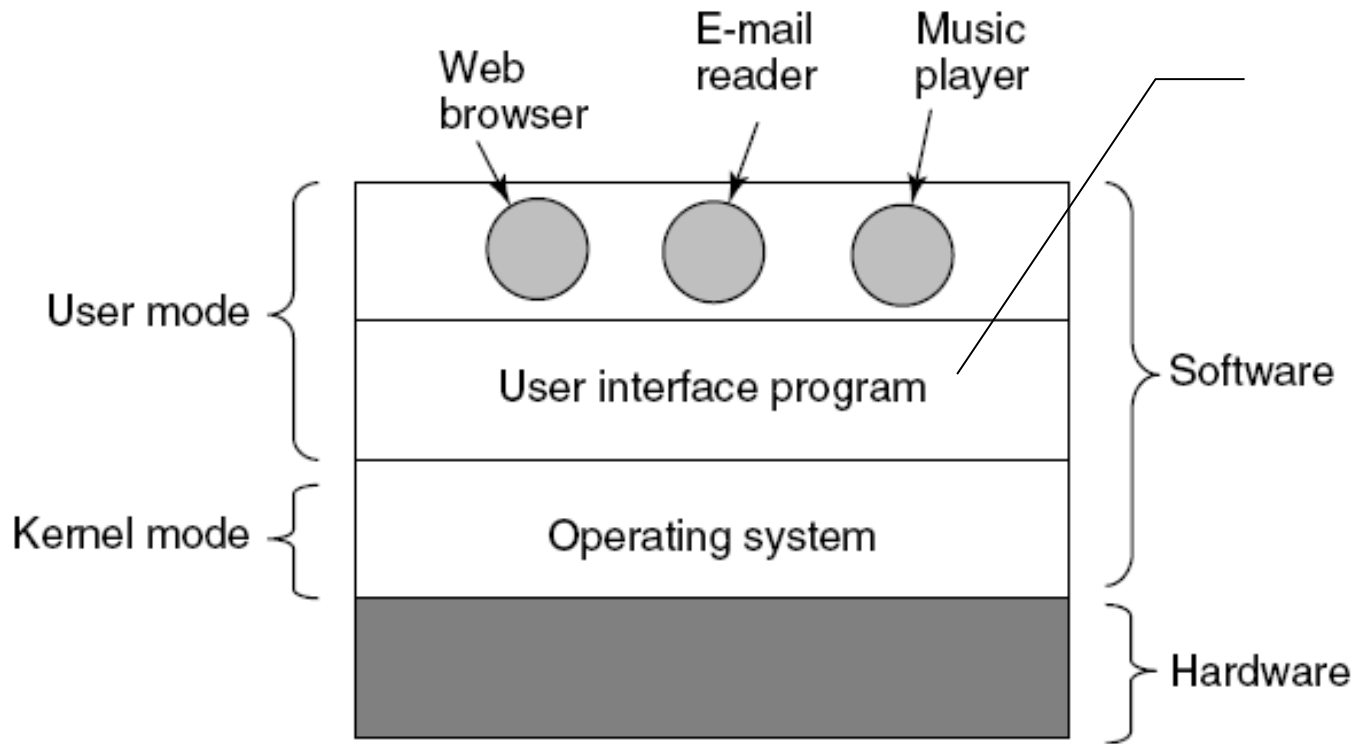
Computador Moderno

- Componentes físicos (**hardware**)
 - Um ou mais processadores
 - Memória
 - Discos
 - Impressoras
 - Vários outros dispositivos de E/S (tela, mouse...)



Gerenciar todos estes componentes requer abstração – *um modelo mais simples do computador* – o **sistema operacional**

Sistema Computacional em Camadas



Sistema Computacional em Camadas

Não pode executar instruções que afetam o controle da máquina ou fazem E/S

User mode

Kernel mode

Acesso completo a todo o hardware e pode executar qualquer instrução que a máquina seja capaz de executar

Web browser

E-mail reader

Music player

GUI ou shell

User interface program

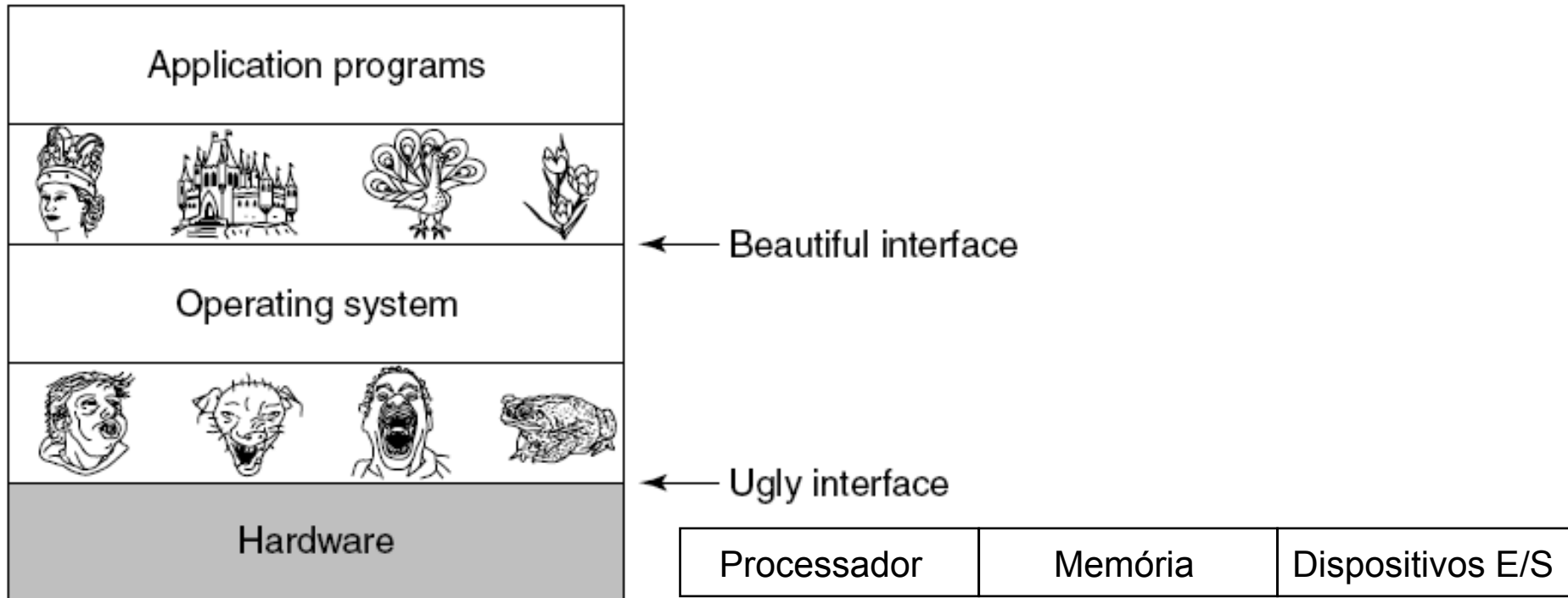
Operating system

Software

Hardware

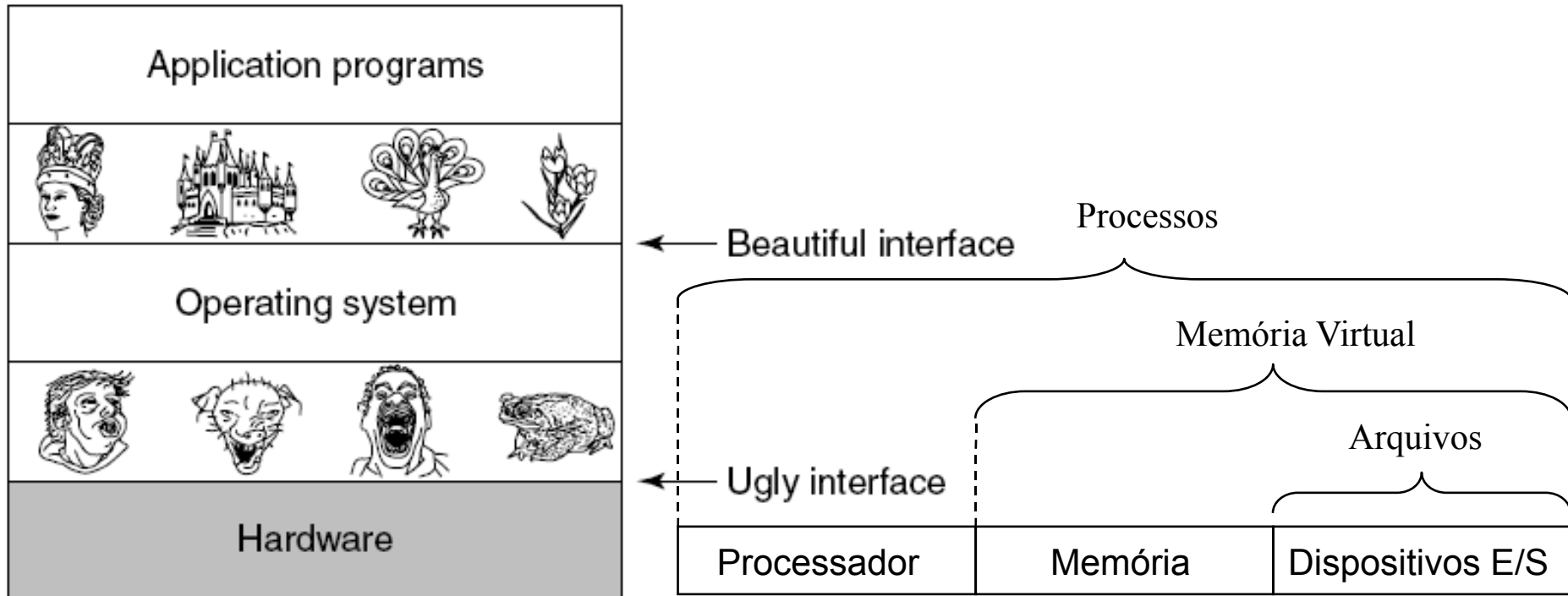
Sistema Operacional como uma Máquina Estendida

- Sistemas operacionais transformam o hardware pouco atraente em abstrações mais interessantes



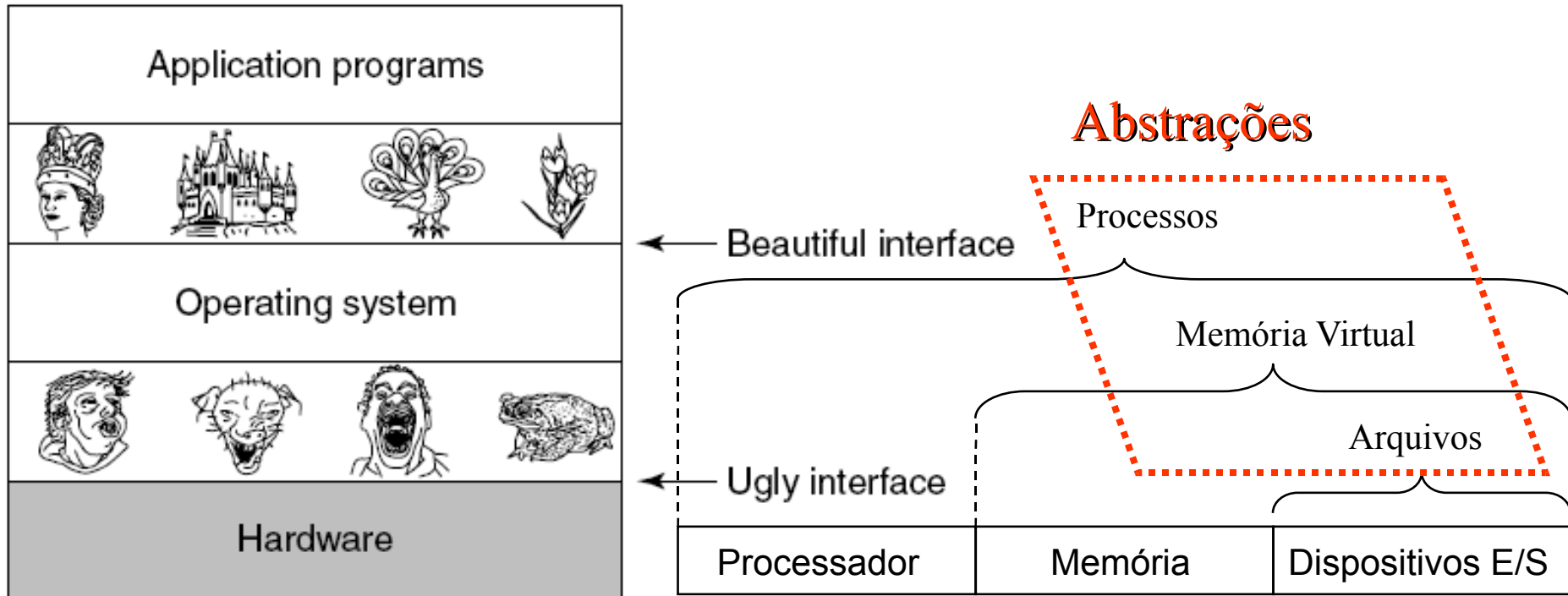
Sistema Operacional como uma Máquina Estendida

- Sistemas operacionais transformam o **hardware pouco atraente** em **abstrações mais interessantes**

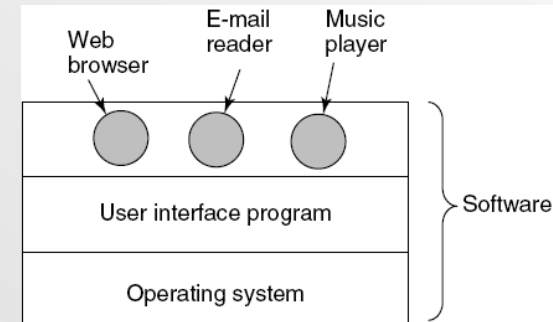


Sistema Operacional como uma Máquina Estendida

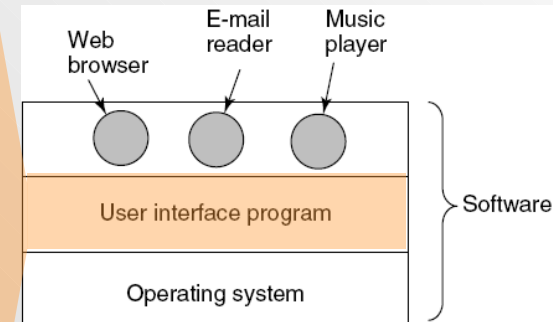
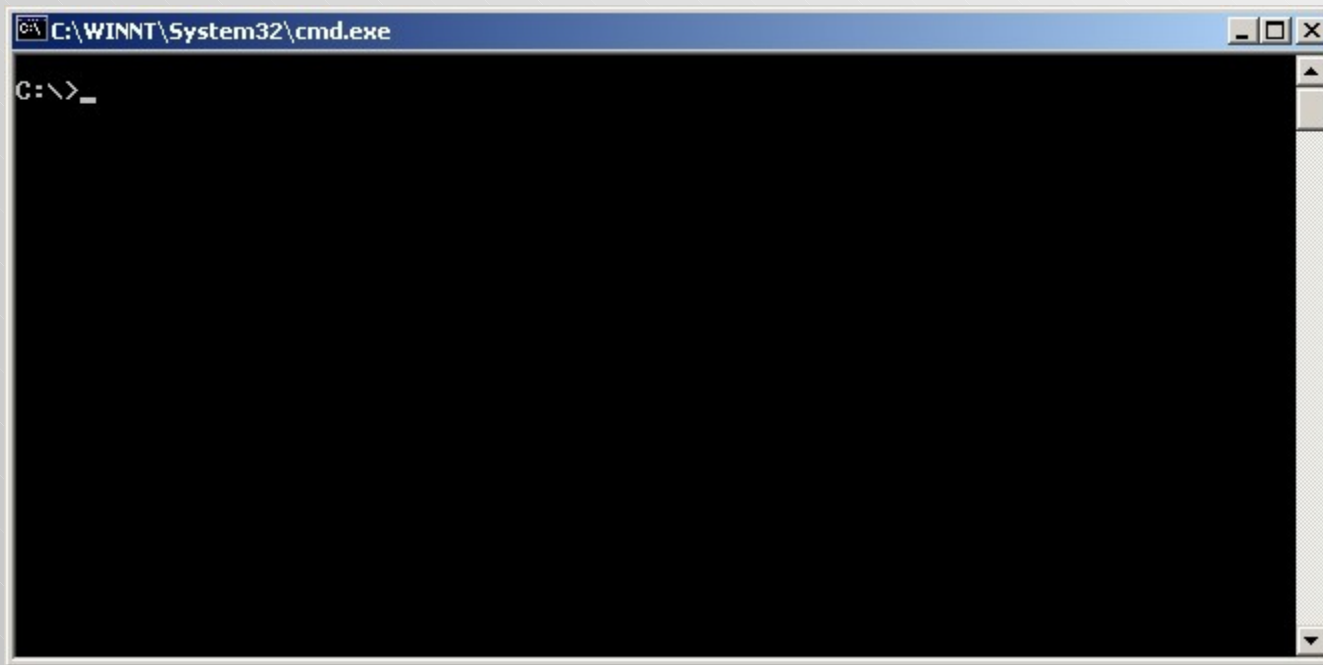
- Sistemas operacionais transformam o **hardware pouco atraente** em **abstrações mais interessantes**



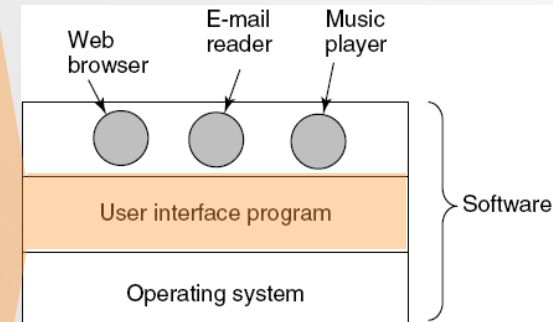
SO: Interface de Usuário



SO: Interface de Usuário - Shell



SO: Interface de Usuário - GUI



Sistema Operacional como um Gerenciador de Recursos

- Gerencia e protege memória, dispositivos de E/S e outros recursos (hardware)
- Permite o compartilhamento (ou multiplexação) de recursos
 - no tempo (*time-sharing*)
 - no espaço

Sistema Operacional como um Gerenciador de Recursos

- **Gerencia** e **protege** memória, dispositivos de E/S e outros recursos (hardware)
- Permite o compartilhamento (ou multiplexação) de recursos
 - no tempo (*time-sharing*)
 - no espaço

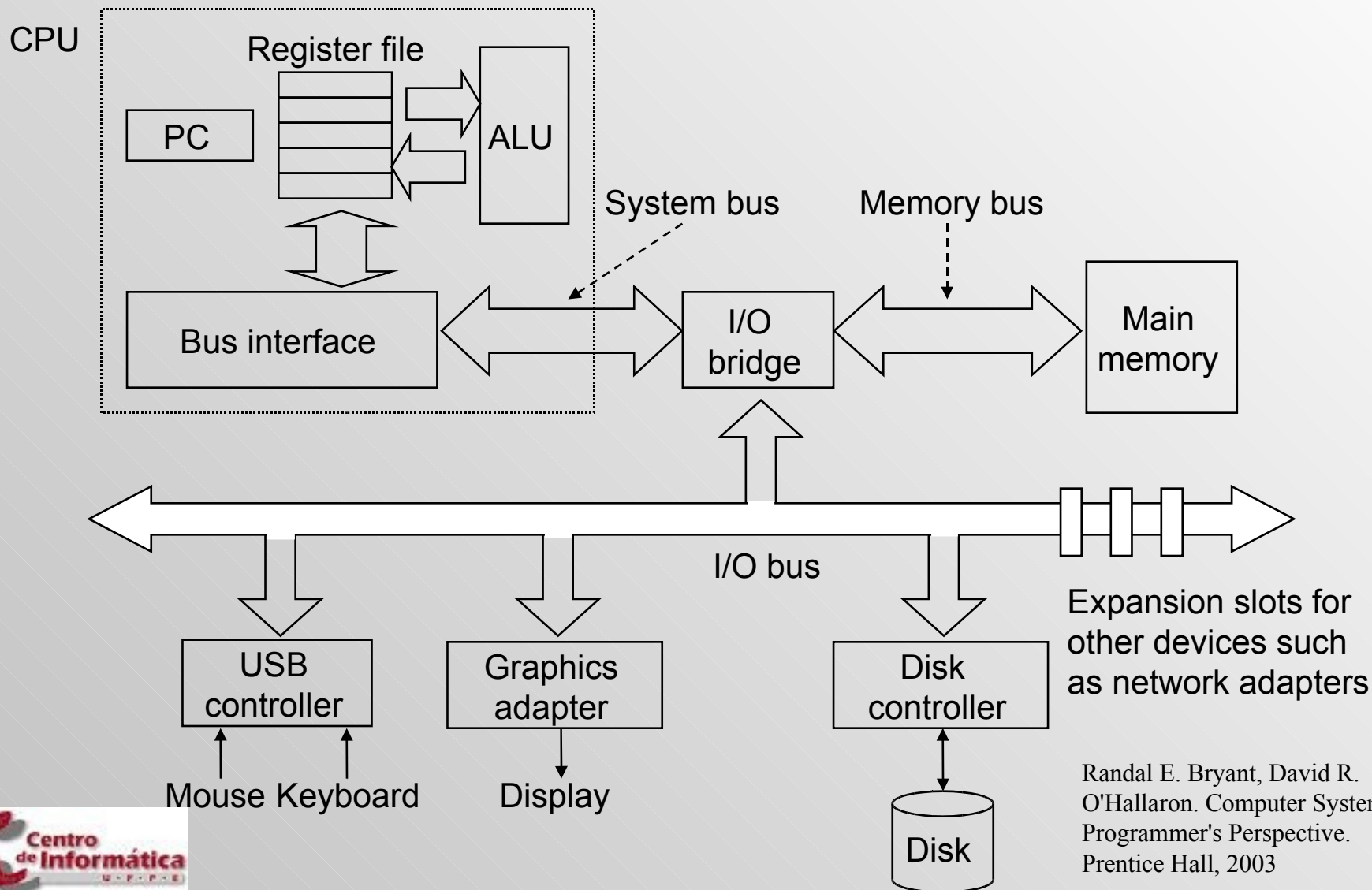
Exemplos?

Sistema Operacional como um Gerenciador de Recursos

- **Gerencia** e **protege** memória, dispositivos de E/S e outros recursos (hardware)
- Permite o compartilhamento (ou multiplexação) de recursos
 - no tempo (*time-sharing*)
 - Ex.: múltiplos programas compartilham o processador (executam) ao mesmo tempo
 - no espaço
 - Ex.: dados de diferentes usuários/arquivos podem compartilhar o espaço em disco

Um pouco de hardware...

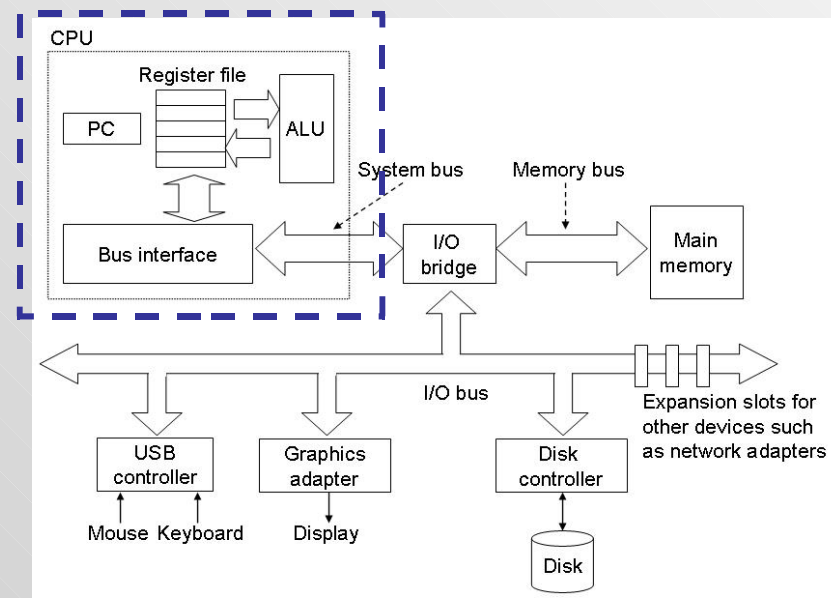
um computador típico



Randal E. Bryant, David R. O'Hallaron. Computer Systems: A Programmer's Perspective. Prentice Hall, 2003

CPU: Central Processing Unit

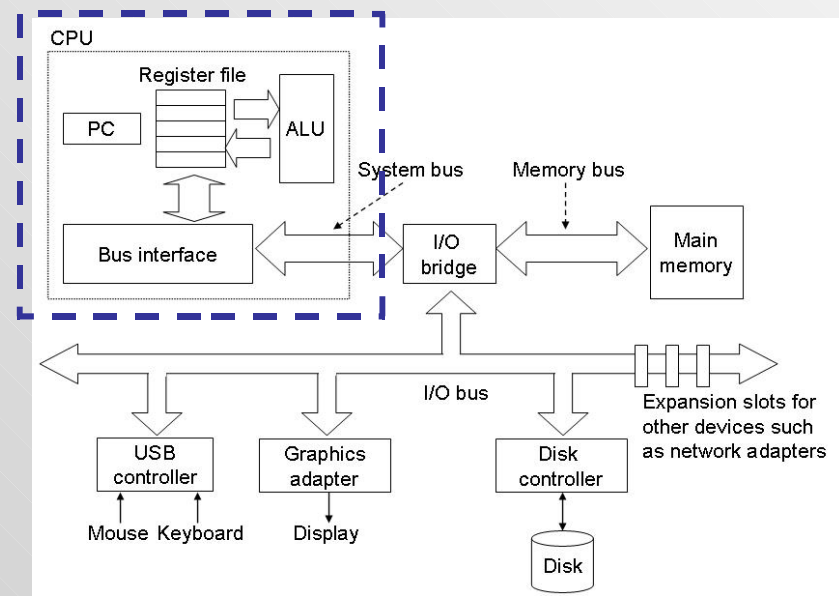
- Unidade de Controle
- **ALU**: Unidade Aritmética e Lógica
- **Registradores**
 - Funcionam como uma memória de acesso **extremamente rápido**
 - **Baixa capacidade**
 - Alguns têm funções especiais
- Exemplos de registradores
 - **PC** (*program counter*): contém o endereço da próxima instrução a ser executada
 - **Instruction register**: onde é copiada cada instrução a ser executada



CPU: Central Processing Unit

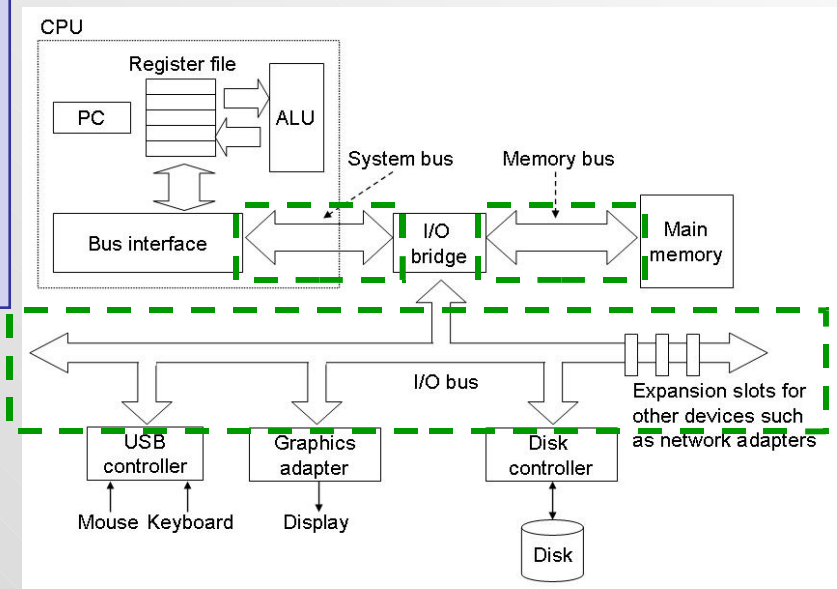
- Unidade de Controle
- **ALU**: Unidade Aritmética e Lógica
- **Registradores**
 - Funcionam como uma memória de acesso **extremamente rápido**
 - **Baixa capacidade**
 - Alguns têm funções especiais
- Exemplos de registradores
 - **PC** (*program counter*): contém o endereço da próxima instrução a ser executada
 - **Instruction register**: onde é copiada cada instrução a ser executada

- A CPU, seguidamente, executa instruções requisitadas à memória
- Ciclo **fetch-decode-execute**:
 1. busca instrução na memória
 2. atualiza PC
 3. decodifica instrução
 4. executa instrução



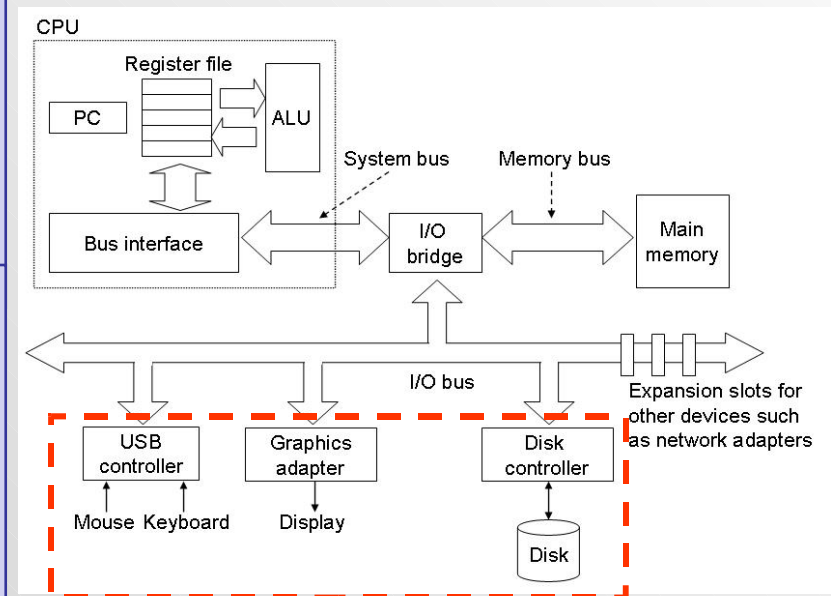
Barramentos e Dispositivos de E/S

- Barramentos: “conduítes” elétricos que **carregam a informação** entre os vários componentes da máquina



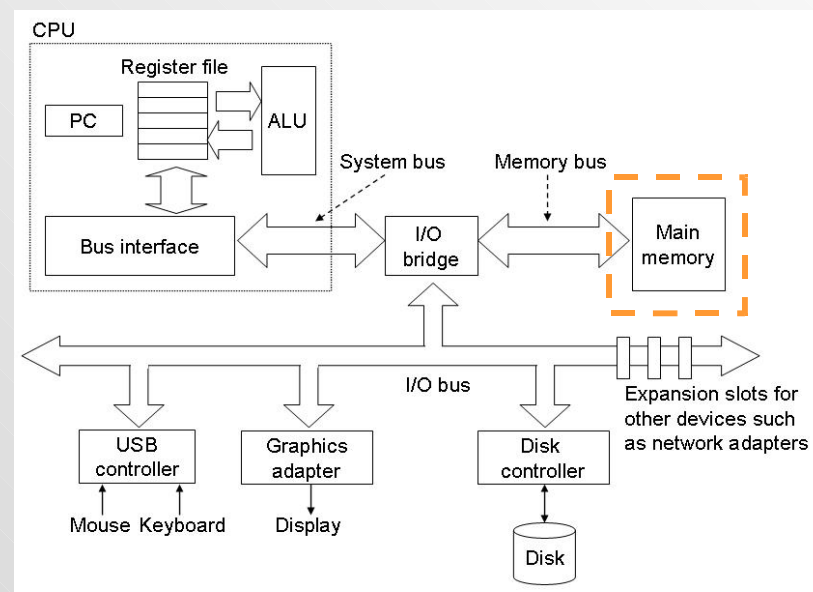
Barramentos e Dispositivos de E/S

- Barramentos: “conduítes” elétricos que **carregam a informação** entre os vários componentes da máquina
- Dispositivos de E/S:
 - Conexão da máquina com o **mundo externo**
 - Conectados ao barramento de E/S por
 - *controladores* (chips no próprio dispositivo ou na placa mãe) ou
 - *adaptadores* (quando placa separada)

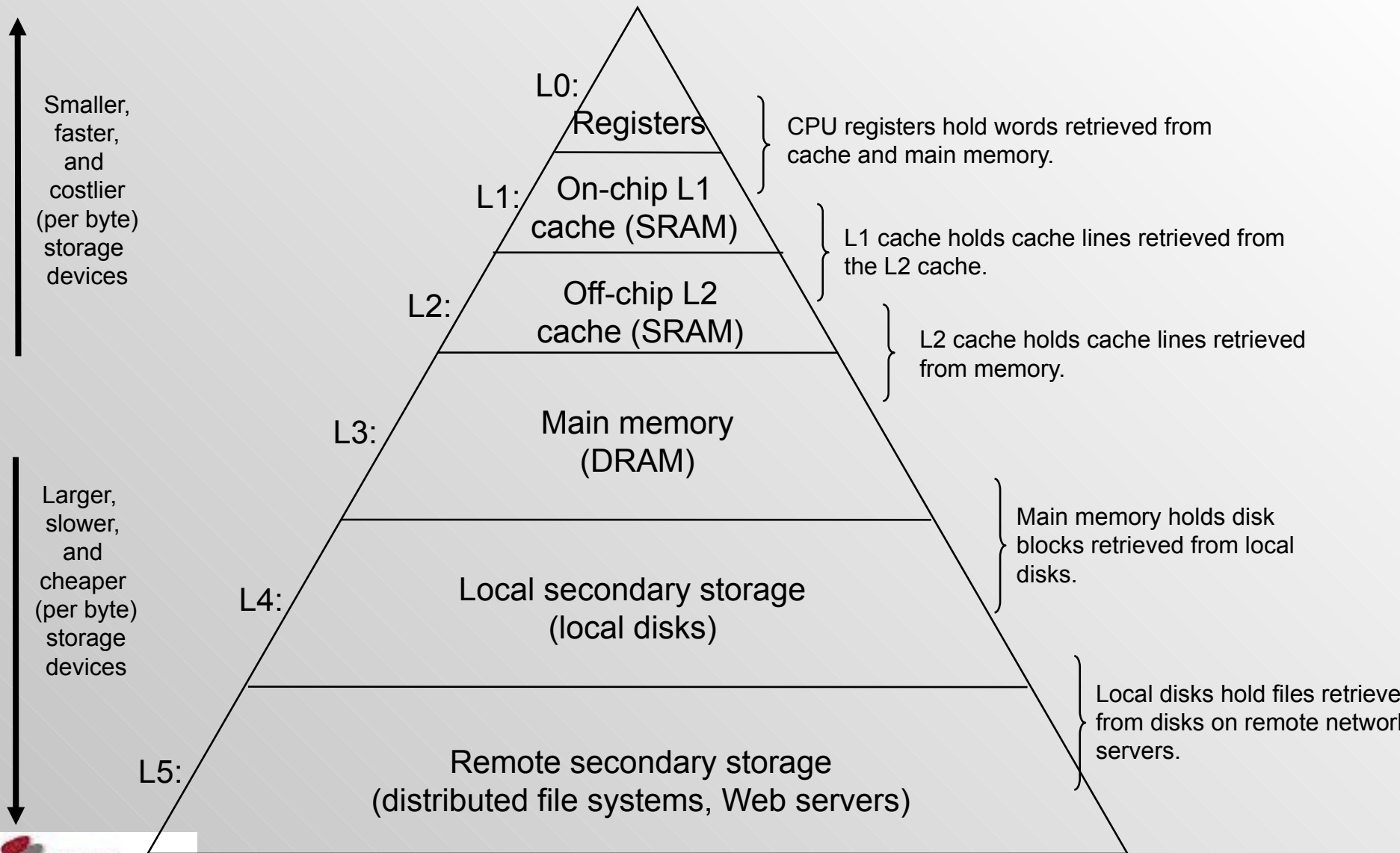


Memória

- Logicamente, a memória principal corresponde a um enorme vetor (*array*) de *bytes*
 - cada posição tem um endereço único (o índice do vetor)
- Os registradores da CPU muitas vezes são usados para armazenar endereços de memória
 - Assim, o número de *bits* em cada registrador **limita** o número de **posições de memória endereçáveis**



Hierarquia de Memória



Sobre hierarquia de memória...

- Nível 1 (L1): dentro de cada núcleo
- Nível 2 (L2): normalmente dentro do mesmo processador (*chip*) e fora dos núcleos
 - No Core 2 Duo e no Core Quad, todos os núcleos compartilham a mesma *cache* L2
- Nível 3 (L3): normalmente fora do processador
 - Alguns exemplos do contrário: Itanium 2 e Xeon MP
- Nível 4 (L4): fora do processador, útil dependendo do padrão de funcionamento da aplicação

Sobre memória *cache*

- Um sistema de computador tem memória *cache* em dois níveis e memória principal (RAM). São necessários 2ns para acessar uma palavra a partir da *cache* L1, 3ns para a *cache* L2 e 10ns para a memória RAM. Se a taxa de acertos da *cache* L1 é 70% e a da *cache* L2 é 95%, qual é o tempo médio de acesso a uma palavra?

Resolução

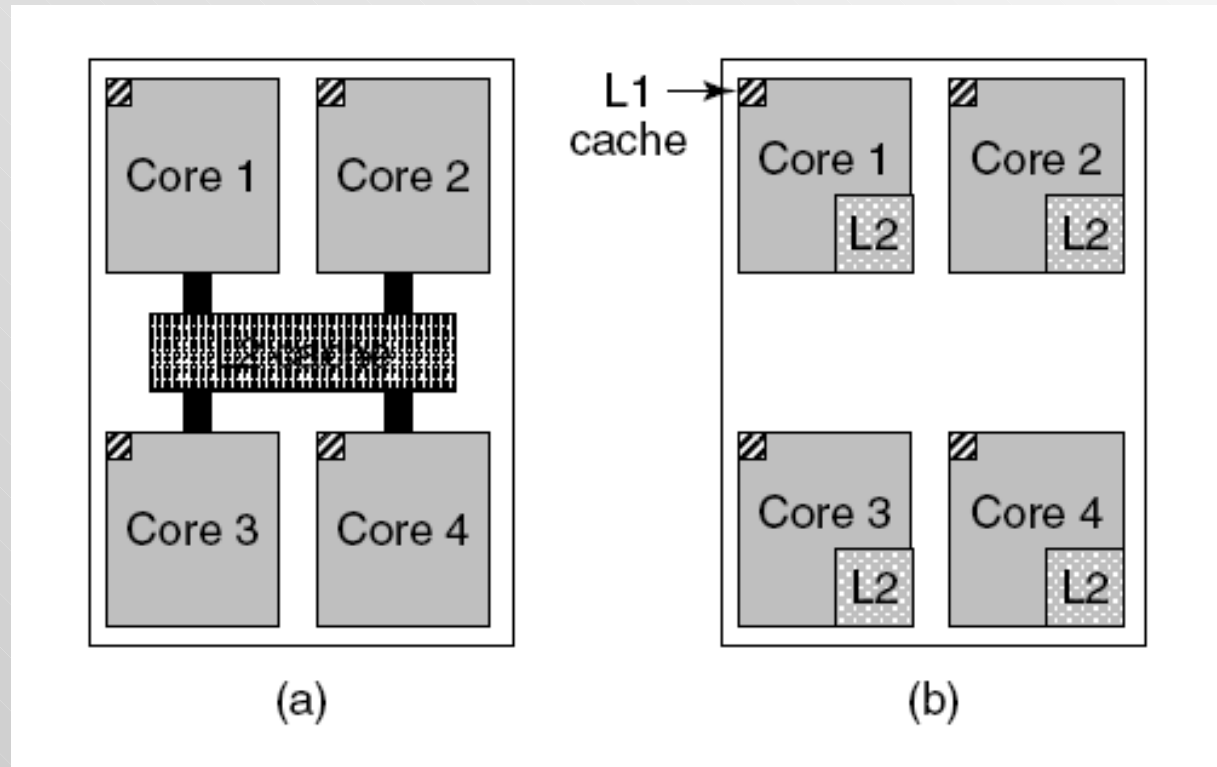
$$T_{\text{médio}} = 0,7*2 + (0,3*0,95)*3 + (0,3*0,05)*10 = 2,405$$

Tempo
médio
para
acesso à
cache L1

Tempo médio
para acesso à
cache L2

Tempo médio para
acesso à memória
RAM

Chips *Multithreaded e Multicore*



(a) A quad-core chip with a shared L2 cache.

(b) A quad-core chip with separate L2 caches.

Software Básico

[A. Raposo e M. Endler, PUC-Rio, 2008]

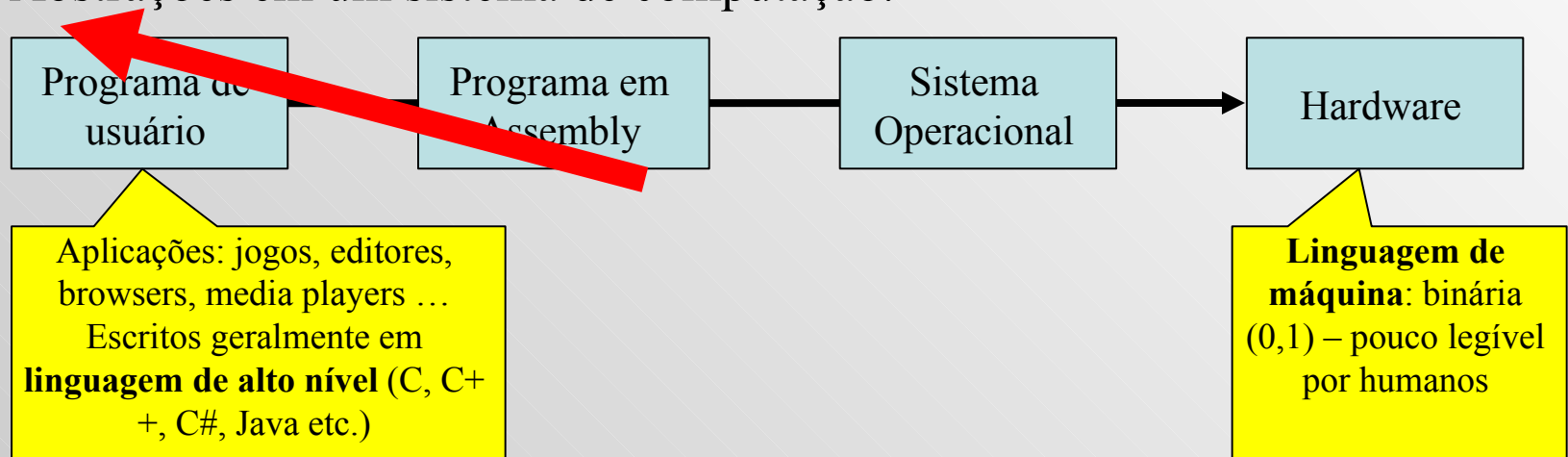
- “Conhecendo mais sobre o que está ‘por baixo’ do programa, você pode escrever programas mais eficientes e confiáveis”
- Abstrações em um sistema de computação:



Software Básico

[A. Raposo e M. Endler, PUC-Rio, 2008]

- “Conhecendo mais sobre o que está ‘por baixo’ do programa, você pode escrever programas mais eficientes e confiáveis”
- Abstrações em um sistema de computação:



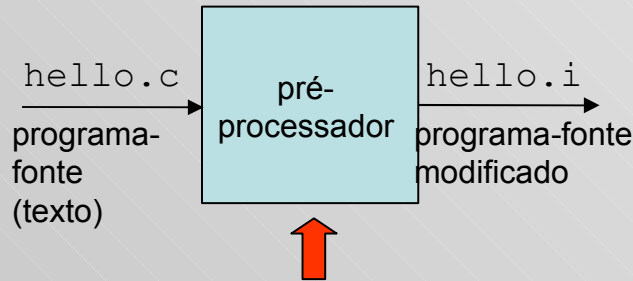
- A linguagem de montagem (**Assembly**) é um **mapeamento direto da linguagem de máquina**, mas que introduz várias “facilidades” para o programador
 - usa "apelidos" das instruções de máquina
 - Ex.: `mov eax, edx`

— move o que está no registrador de dados para o acumulador

Gerando um executável

- `unix> gcc -o hello hello.c`

```
1. #include <stdio.h>
2. int main()
3. {
4.     printf("hello, world\n");
5. }
```

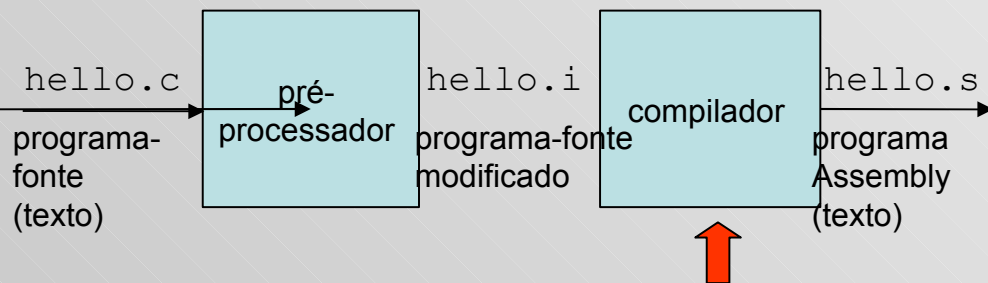


- Modifica o programa em C de acordo com diretivas
 - Ex.: `#include <stdio.h>` diz ao pré-processador para ler o arquivo `stdio.h` e inseri-lo no programa fonte
- Resultado: programa expandido em C (extensão `.i`, em Unix)

Gerando um executável

- `unix> gcc -o hello hello.c`

```
1. #include <stdio.h>
2. int main()
3. {
4.     printf("hello, world\n");
5. }
```

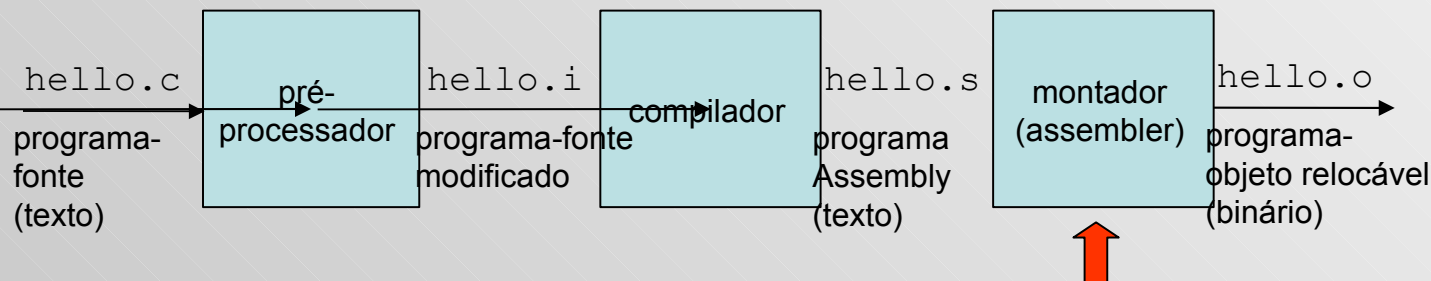


- Compilador traduz o programa .i em um programa em Assembly
 - Opção -S
 - Formato de saída comum para os compiladores em várias LPs de alto nível
 - i.e., programas em C, Java, Fortran, etc. vão ser traduzidos para a mesma linguagem Assembly

Gerando um executável

- `unix> gcc -o hello hello.c`

```
1. #include <stdio.h>
2. int main()
3. {
4.     printf("hello, world\n");
5. }
```

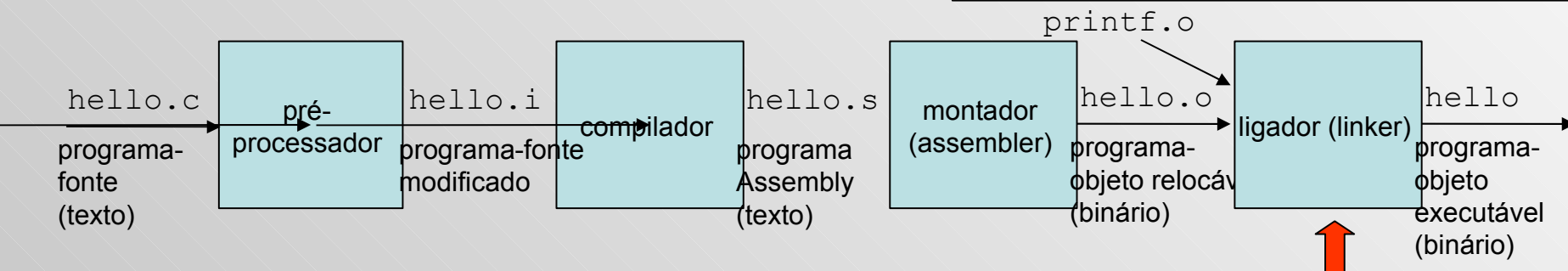


- Montador transforma o programa em Assembly em um binário em linguagem de máquina (“programa-objeto relocável” – extensão .o)
 - Opção -c
 - Os módulos de programas, compilados ou montados, são armazenados em um formato intermediário (“Programa-Objeto Relocável”)
- Endereços de acesso e a posição do programa na memória ficam **indefinidos**

Gerando um executável

- `unix> gcc -o hello hello.c`

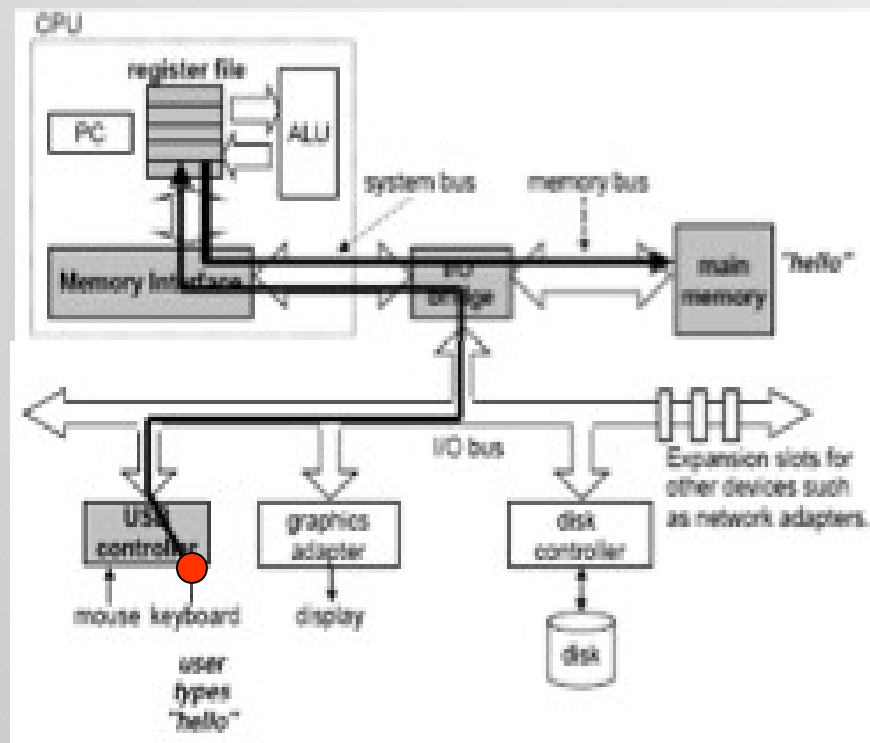
```
1. #include <stdio.h>
2. int main()
3. {
4.     printf("hello, world\n");
5. }
```



- Ligador (*linker*) gera o programa executável a partir do arquivo `.o`
- Pode haver funções-padrão da linguagem (ex., `printf`) definidas em outro arquivo `.o` pré-compilado (`printf.o`)
- O ligador combina os programas-objeto necessários para gerar o executável

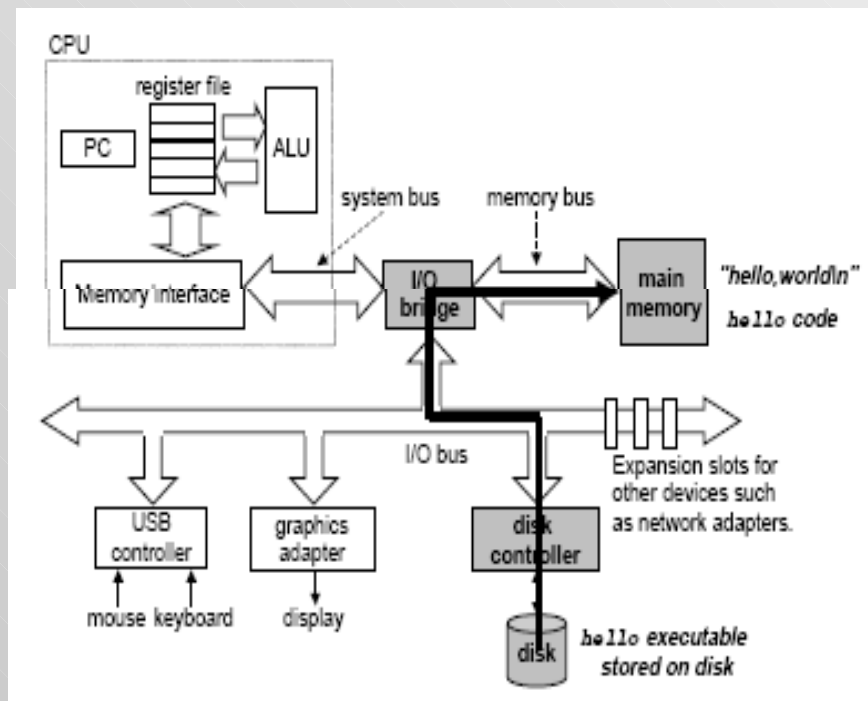
Execução de um programa

- **Processo:** Um programa em execução
 1. Ao digitar “hello”, os caracteres são passados para um registrador e depois para memória principal



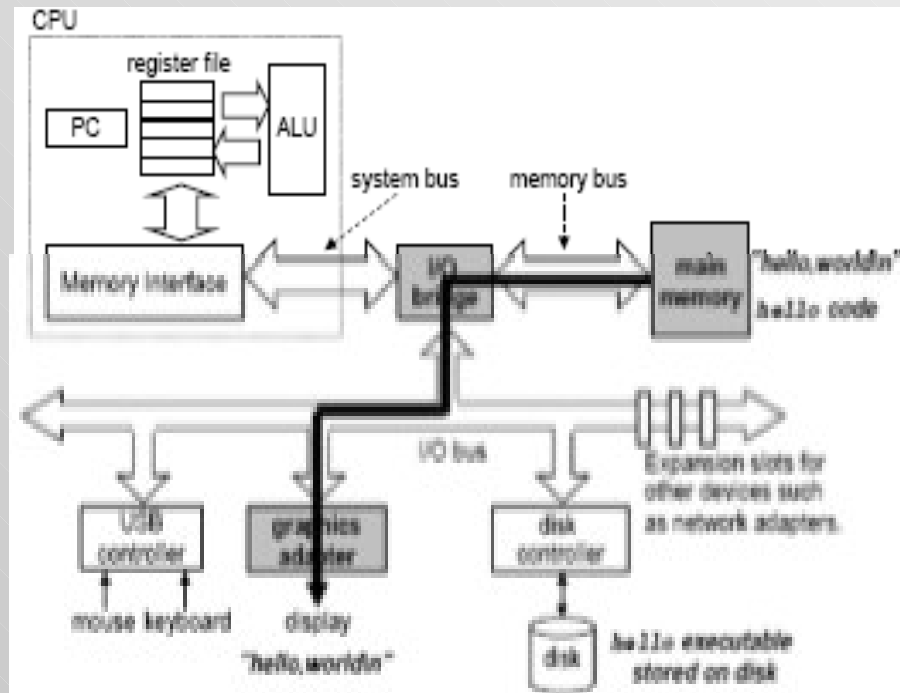
Execução de um programa

1. Ao digitar “Enter”, sabe-se que acabou o comando
 - São executadas instruções para copiar código e dados do programa `hello` do disco para a memória principal



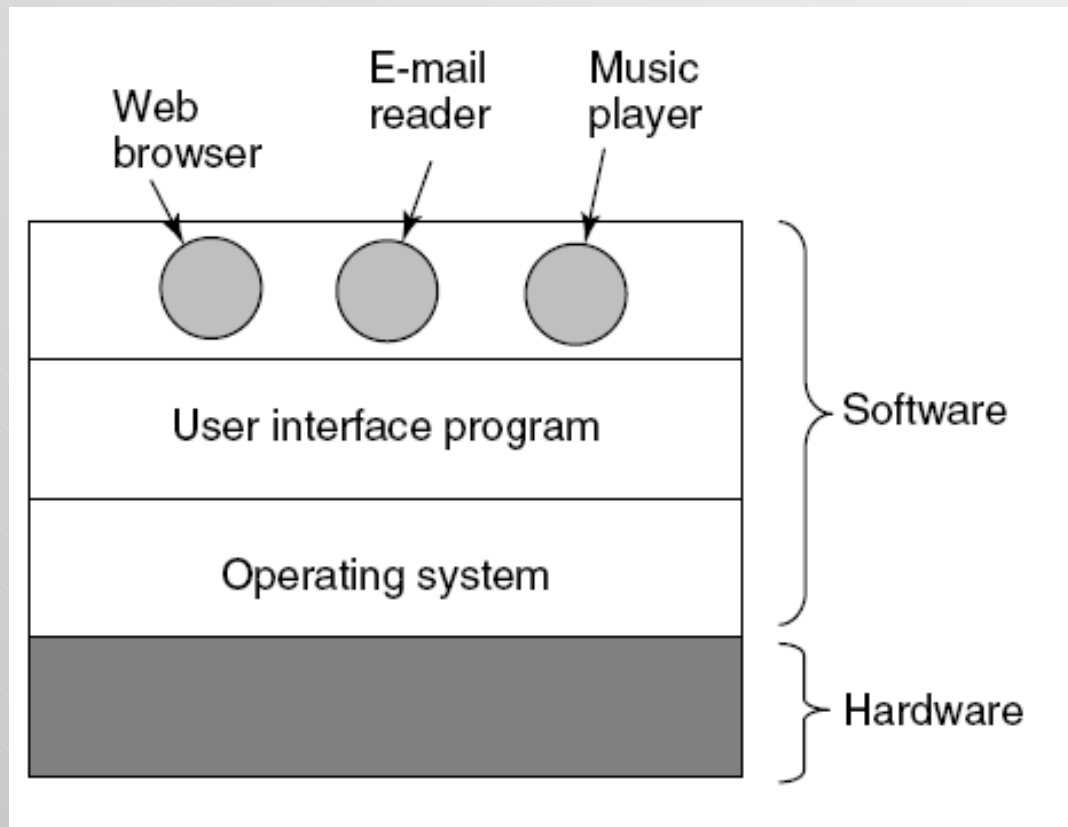
Execução de um programa

1. PC aponta para o endereço de memória onde o programa `hello` está
2. Processador executa instruções em linguagem de máquina da função `main()` do programa



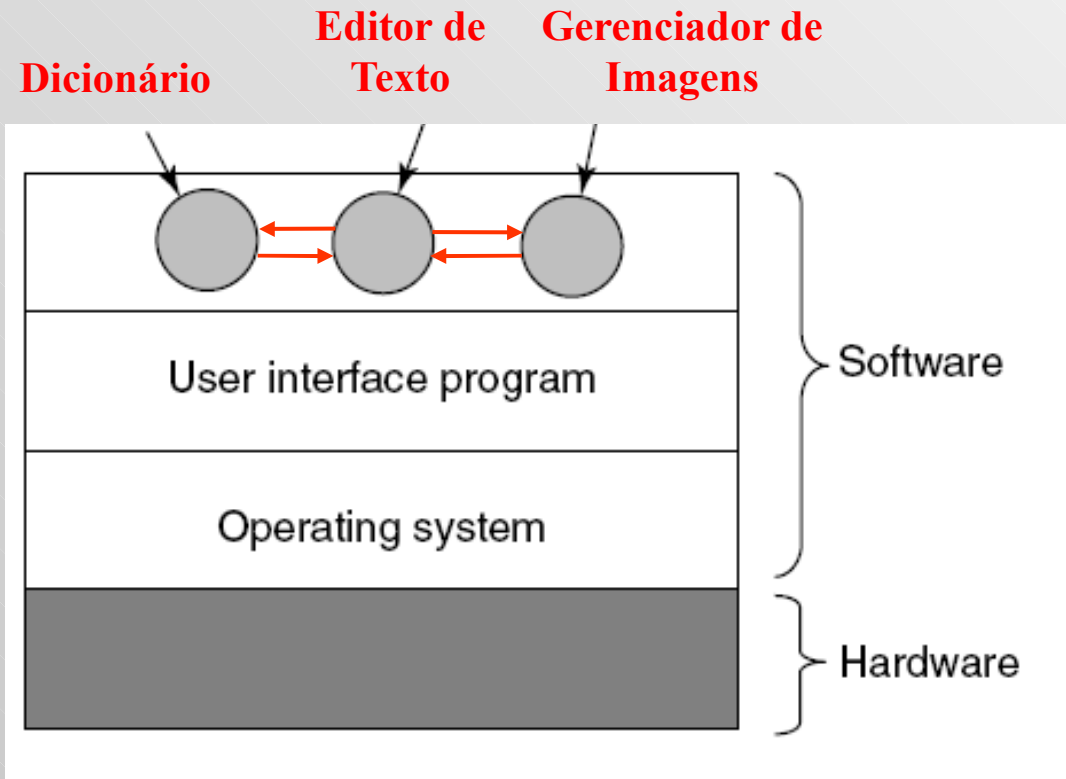
Mais de um programa em execução

- Múltiplos processos vs. um (ou [poucos] mais) processador(es) $\Rightarrow$ **como é possível?**



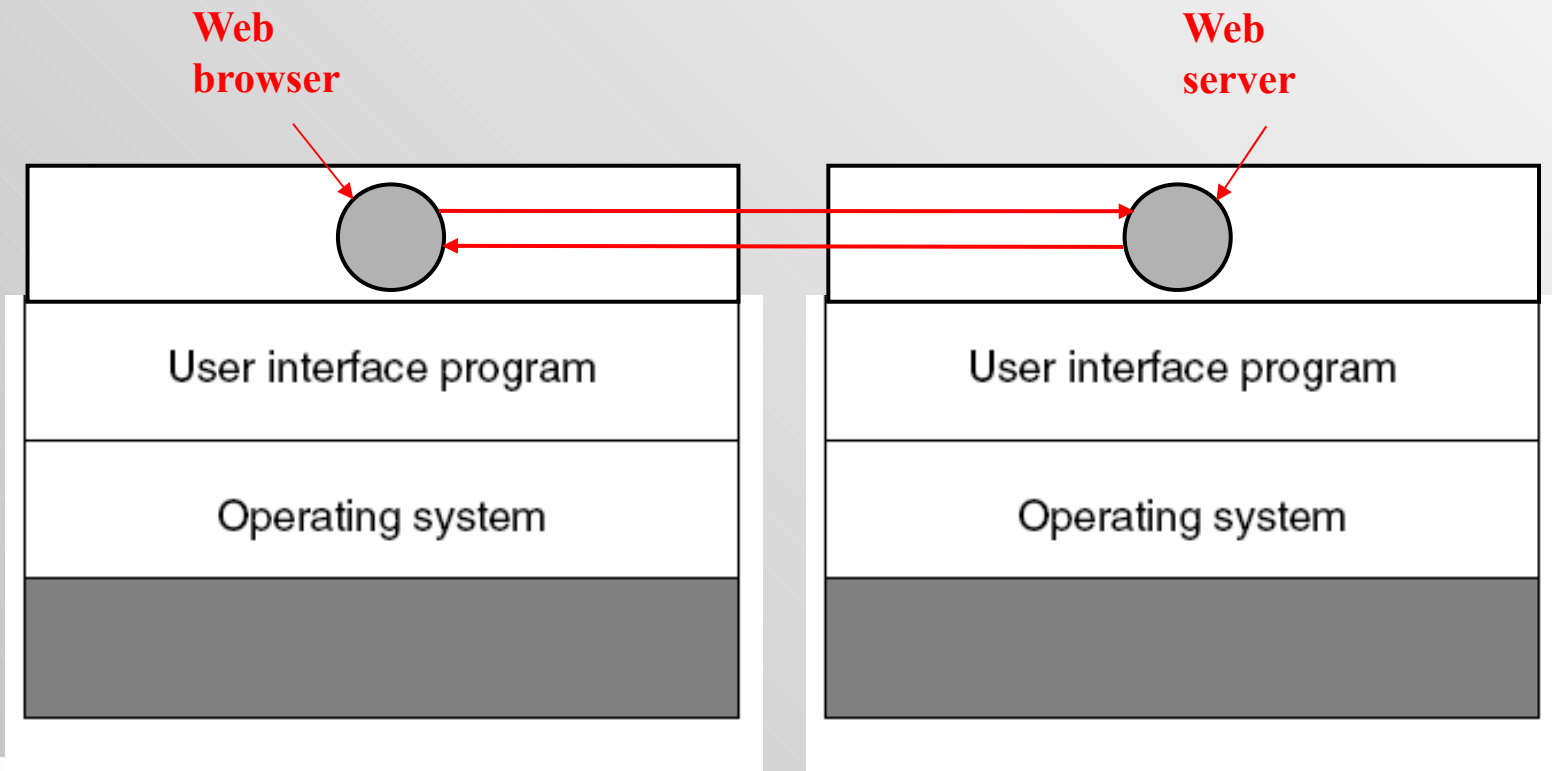
Processos que se Comunicam

- Como é possível?



Sistemas Distribuídos

- Processos em máquinas distintas e que se comunicam



O que vimos

- **Complexidade** do computador moderno, do ponto de vista do hardware
- Necessidade de **abstrações** – software
- Sistema computacional em **camadas**
- SO como uma **máquina estendida** – abstrações
- SO como um **gerenciador de recursos**
 - Gerenciamento
 - Proteção
 - **Compartilhamento**



O que vimos

- **Complexidade** do computador moderno, do ponto de vista do hardware
- Necessidade de **abstrações** – software
- Sistema computacional em **camadas**
- SO como uma **máquina estendida** – abstrações
- SO como um **gerenciador de recursos**
 - Gerenciamento
 - Proteção
 - **Compartilhamento**

Um pouco de hardware...

- CPU: unidade de controle e execução
 - Registradores
 - Ciclo *fetch-decode-execute*
- Barramentos e dispositivos de E/S
- Memória [**hierarquia**]

Software básico

- Linguagem de programação de alto nível $\Rightarrow$ linguagem de montagem (**Assembly**) $\Rightarrow$ linguagem de máquina
- **Processo**: um programa em execução
- Comunicação entre processos
- **Sistemas Distribuídos**

