# Computação Concorrente (DCC/UFRJ)

## Aula 3: Visão geral dos sistemas de computação

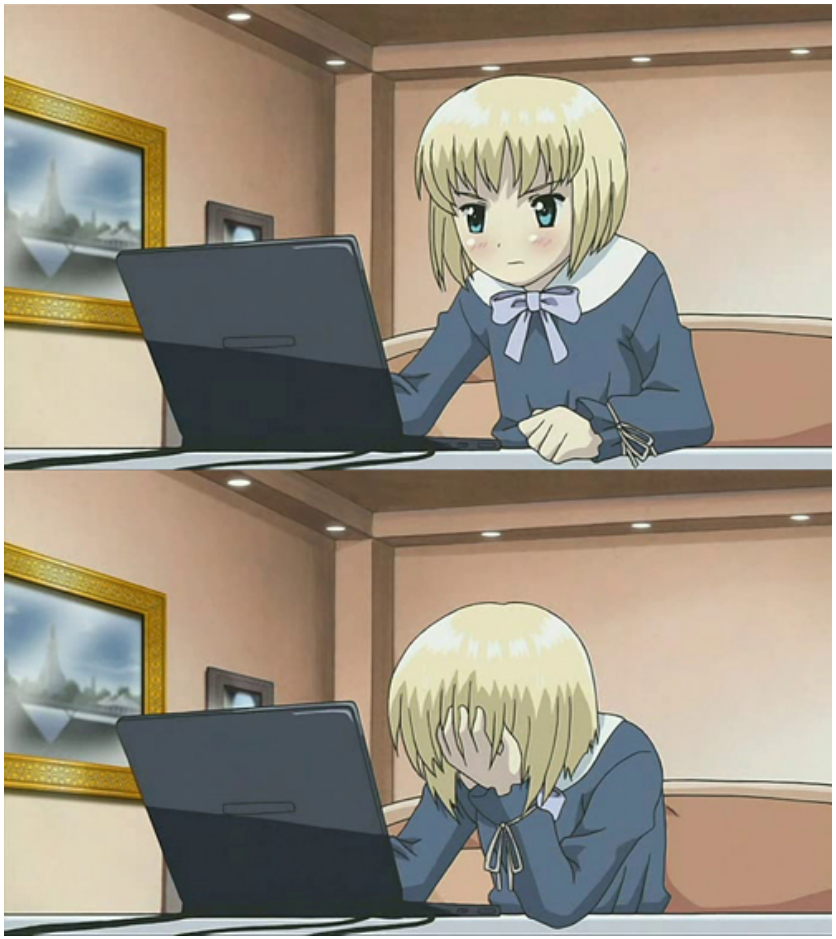
Prof. Silvana Rossetto

20 de agosto de 2019

# Sistemas de computação


Um "sistema de computação" consiste de **hardware** (computador, máquina) e **sistemas de software** que funcionam juntos para executar aplicações do usuário

# Sistemas de computação



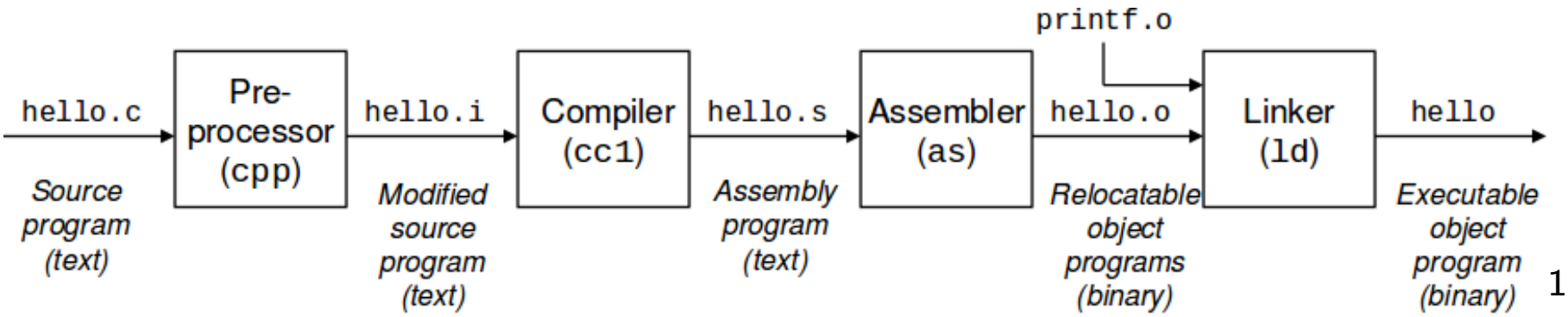
- Há diferentes implementações dos sistemas de computação (Hw e Sw), mas os **conceitos básicos são os mesmos**
- Queremos compreender como esses **componentes funcionam** e como afetam a corretude e desempenho dos nossos programas

# Trajetória de um programa básico (hello.c)

```
#include <stdio.h>
int i=10;
int main(void) {
  i++;
  printf("%d\n", i);
  return 0;
}
```

O programa `hello.c` é armazenado em um **arquivo texto** (representação em ASCII) como uma sequência de bytes, cada byte sendo um valor inteiro que representa um caracter

# Trajetória de um programa básico




[1]

Para que o programa `hello.c` execute, as sentenças C precisam ser traduzidas em uma sequência de **instruções de máquina** (`gcc -o hello hello.c`)

[1] *Fonte: http://csapp.cs.cmu.edu*

# Trajetória de um programa básico

## Programa executável

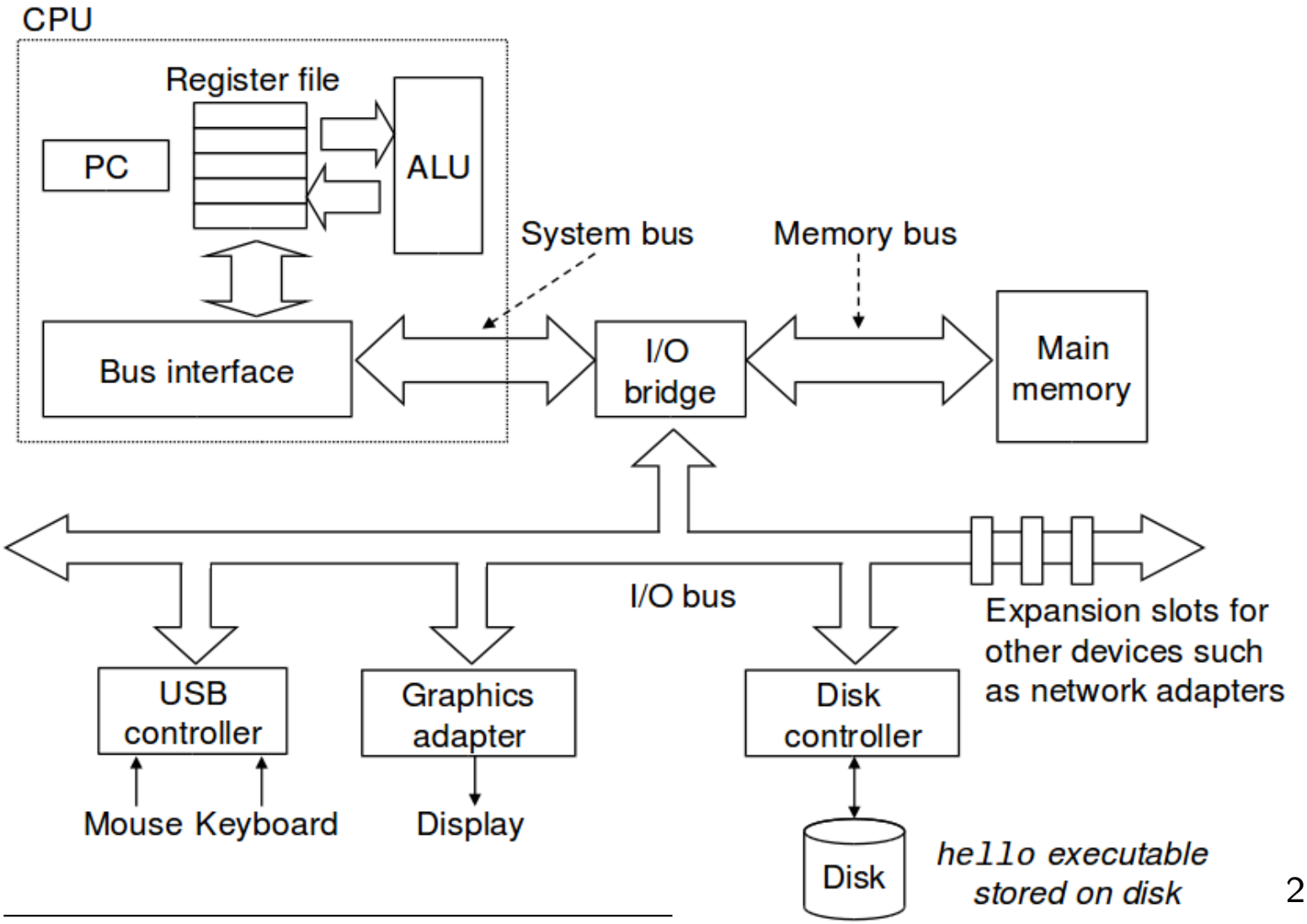
- O programa executável gerado (`hello`) é armazenado no disco do computador
- Para executá-lo, podemos usar a aplicação `shell` fazendo: `./hello`
- O `shell` é um **interpretador de linha de comando** que exibe um prompt e espera por comandos ou arquivos executáveis
- O programa executável será carregado para a memória principal e executado pelo processador **instrução por instrução**

# Programa na linguagem de montagem

(**gcc -S hello.c**)

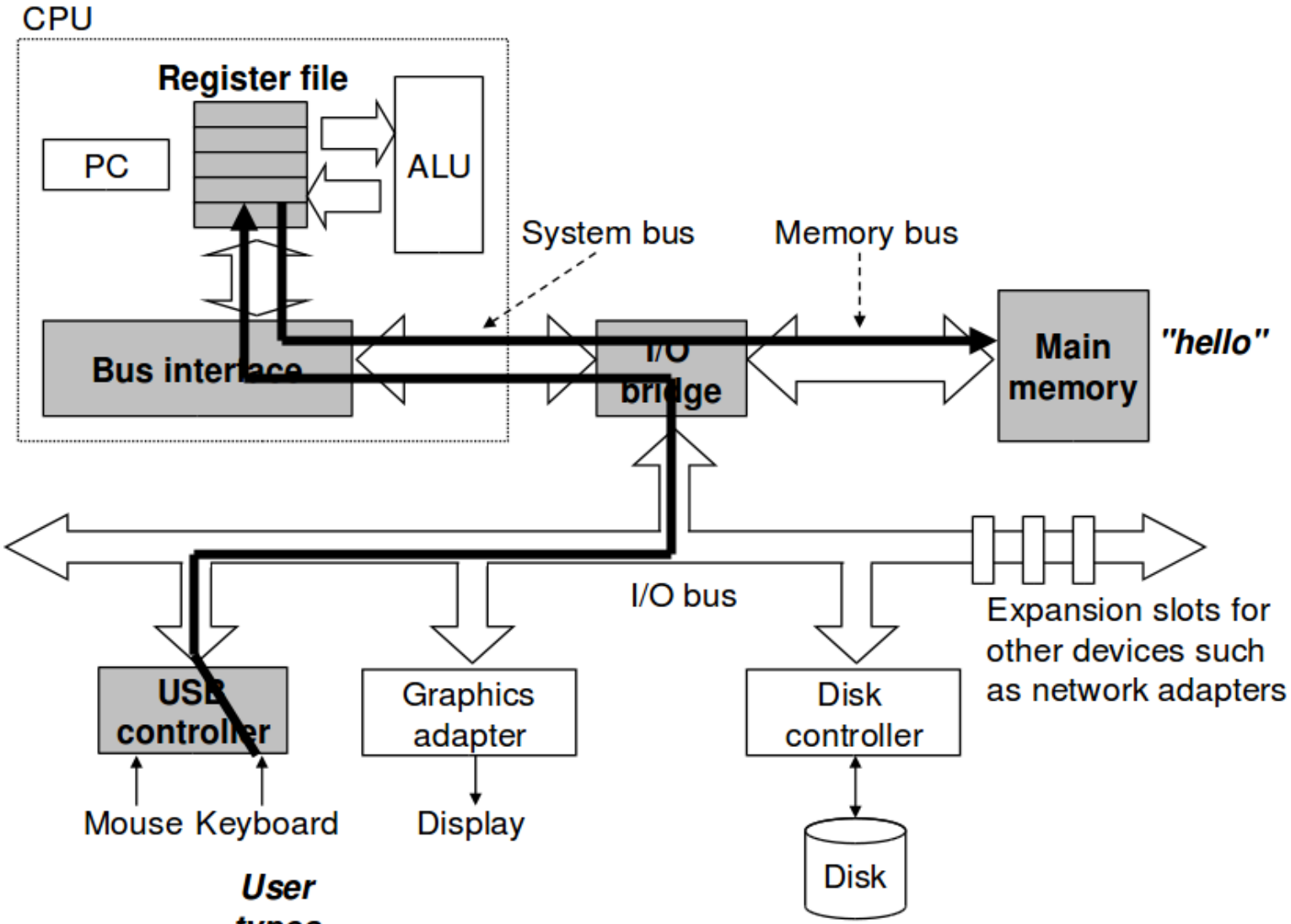
```
main:   pushl   %ebp
        movl    %esp, %ebp
        movl    i, %eax
        addl    $1, %eax
        movl    %eax, i
        movl    i, %edx
        movl    $.LC0, %eax
        movl    %edx, 4(%esp)
        movl    %eax, (%esp)
        call    printf
        movl    $0, %eax
        ret
```

# Organização do hardware de um sistema

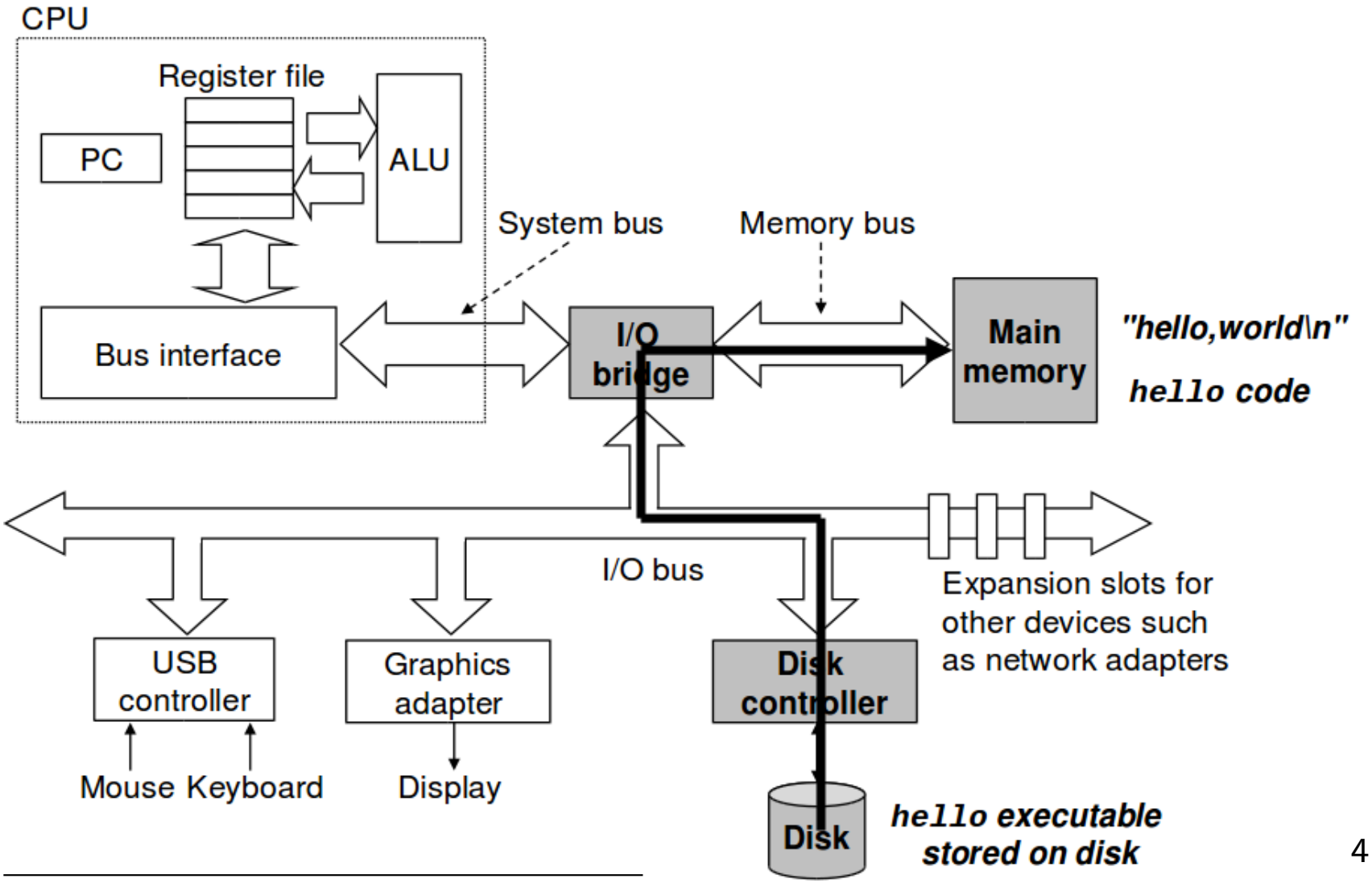



[2] *Fonte: http://csapp.cs.cmu.edu*

# Execução do programa "hello"

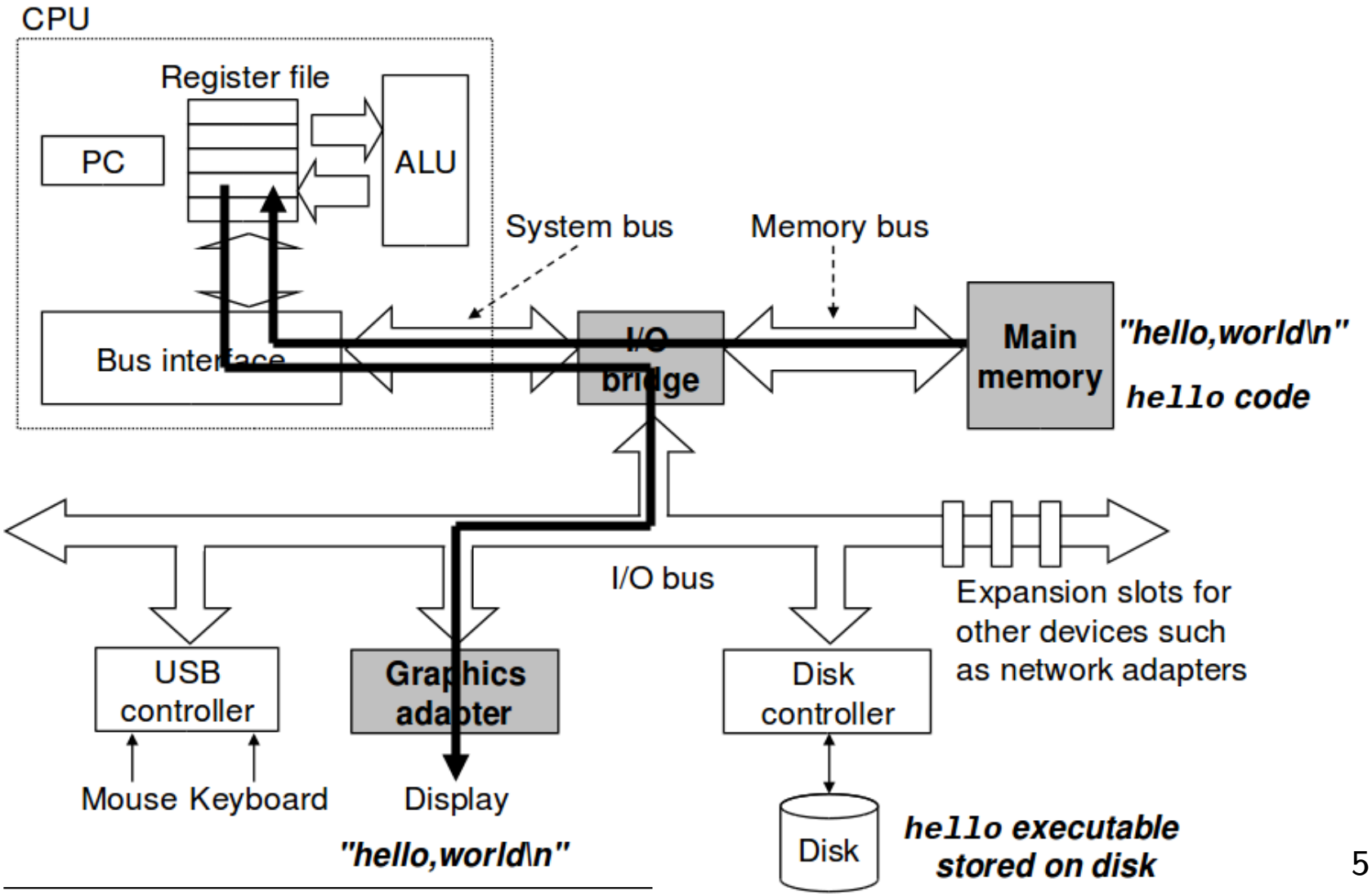


# Execução do programa "hello"



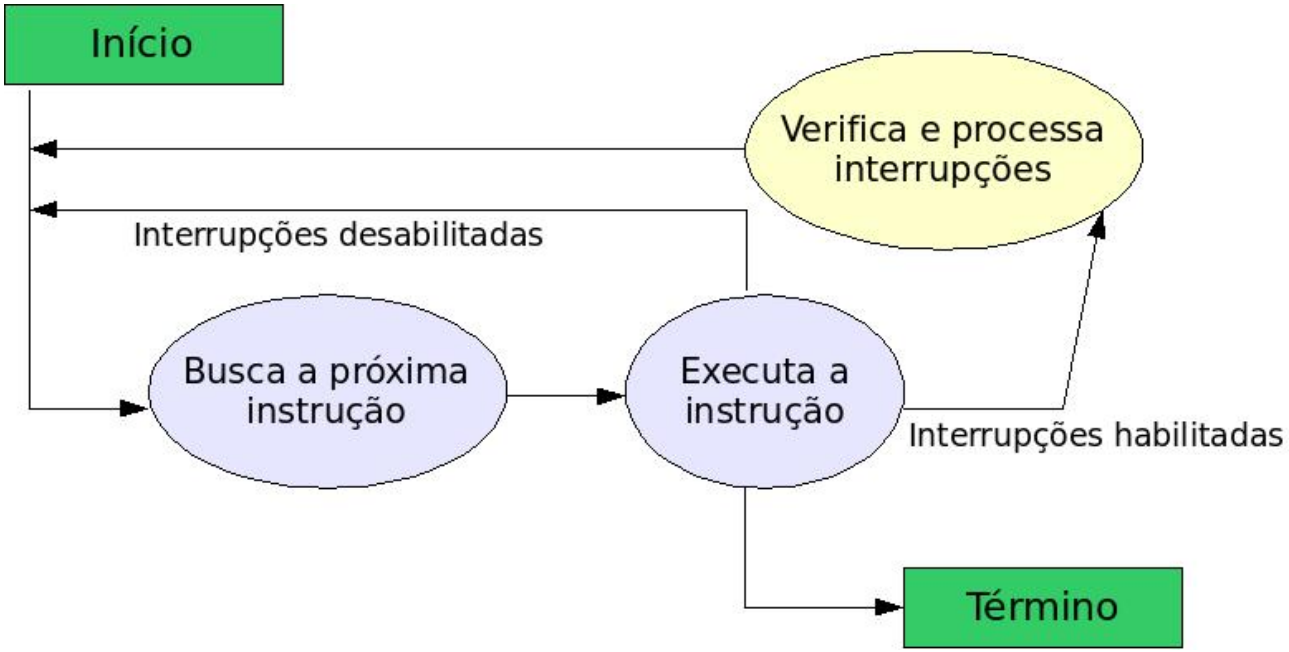

[4] *Fonte: http://csapp.cs.cmu.edu*

# Execução do programa "hello"




[5] *Fonte: http://csapp.cs.cmu.edu*

# O papel do processador (CPU)




- A CPU **carrega, interpreta** e **executa** as instruções de máquina armazenadas na memória principal
- O PC (Ponteiro de Programa) contém o **endereço da instrução corrente**
- A CPU repetidamente executa a instrução apontada pelo PC e o atualiza para apontar para a próxima instrução

# Interrupção do tempo

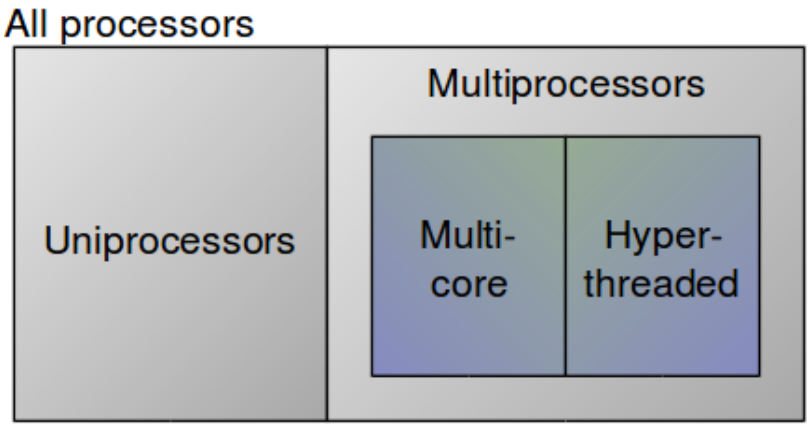
```
int s;

void soma() {
      s++;
}
```

```
.comm   s,4,4
soma: (...)
      movl    s, %eax

      addl    $1, %eax

      movl    %eax, s
      (...)
```

Interrupção do tempo

Interrupção do tempo

# Evolução dos processadores




1. **Uniprocessador**: computador com UM único processador, pode alternar entre várias tarefas (*timesharing*)
2. Multiprocessador: computador com VÁRIOS processadores, sob o controle do mesmo Sist. Oper.
   - **multicore**: várias CPUs (ou "cores") integrados no mesmo chip
   - **hyperthread**: uma CPU com cópias/réplicas de parte do HW (ex., registradores)

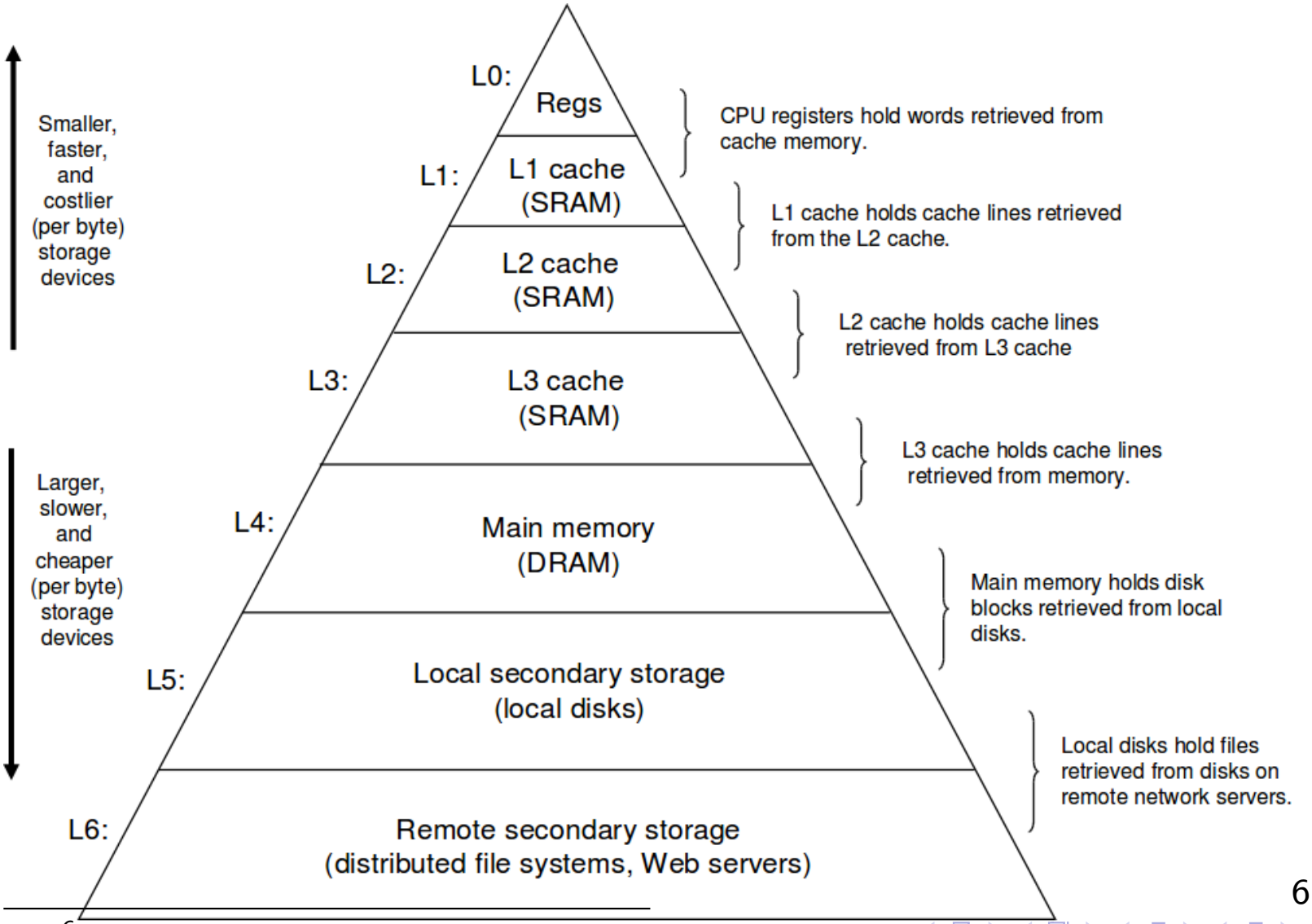
# Gargalo processador-memória

- O sistema gasta boa parte do tempo movendo informação de um lugar para outro
- Boa parte do tempo de execução do programa é gasto com acesso à memória

## Solução: hierarquia de memória

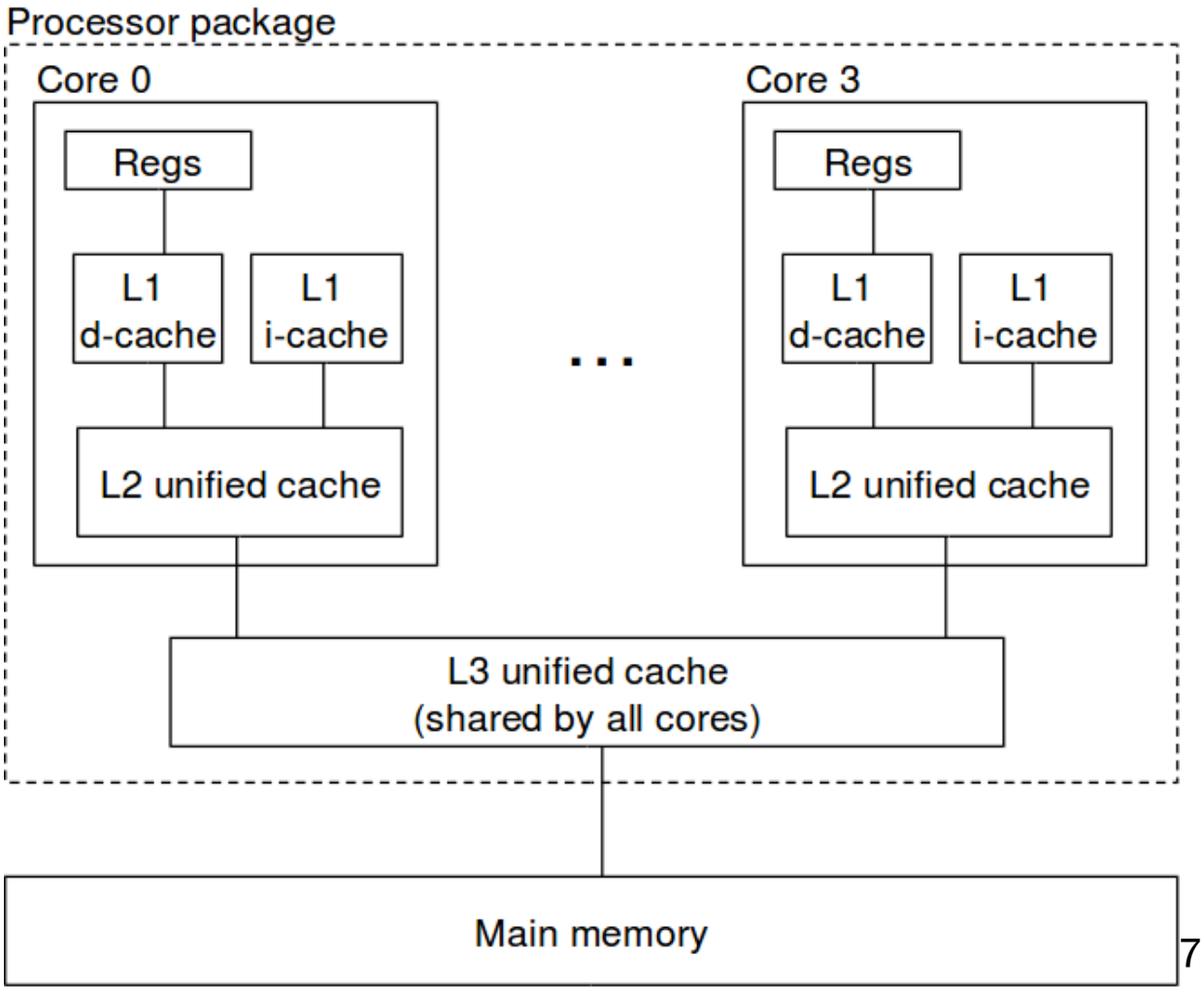
- Insere **dispositivos de armazenamento mais rápidos e baratos entre o processador e a memória principal**

# Hierarquia de memória



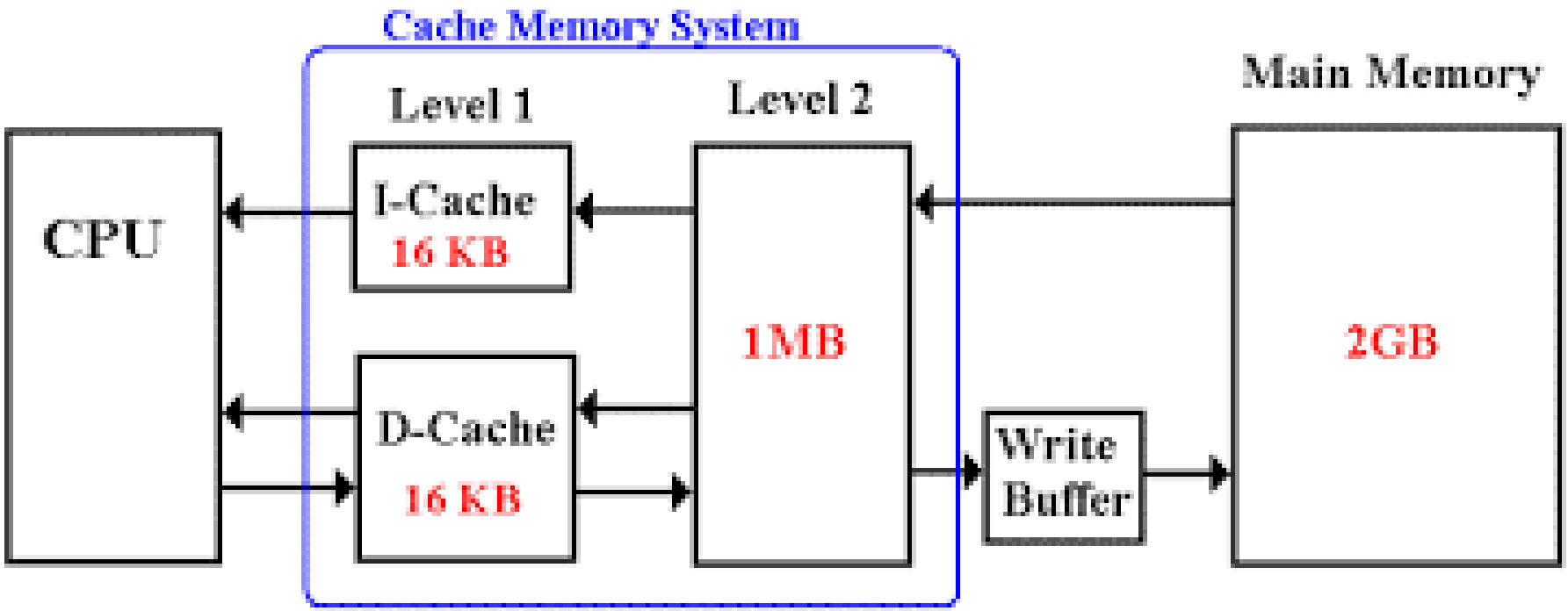

[6] Fonte: http://csapp.cs.cmu.edu

# Organização do IntelCore i7

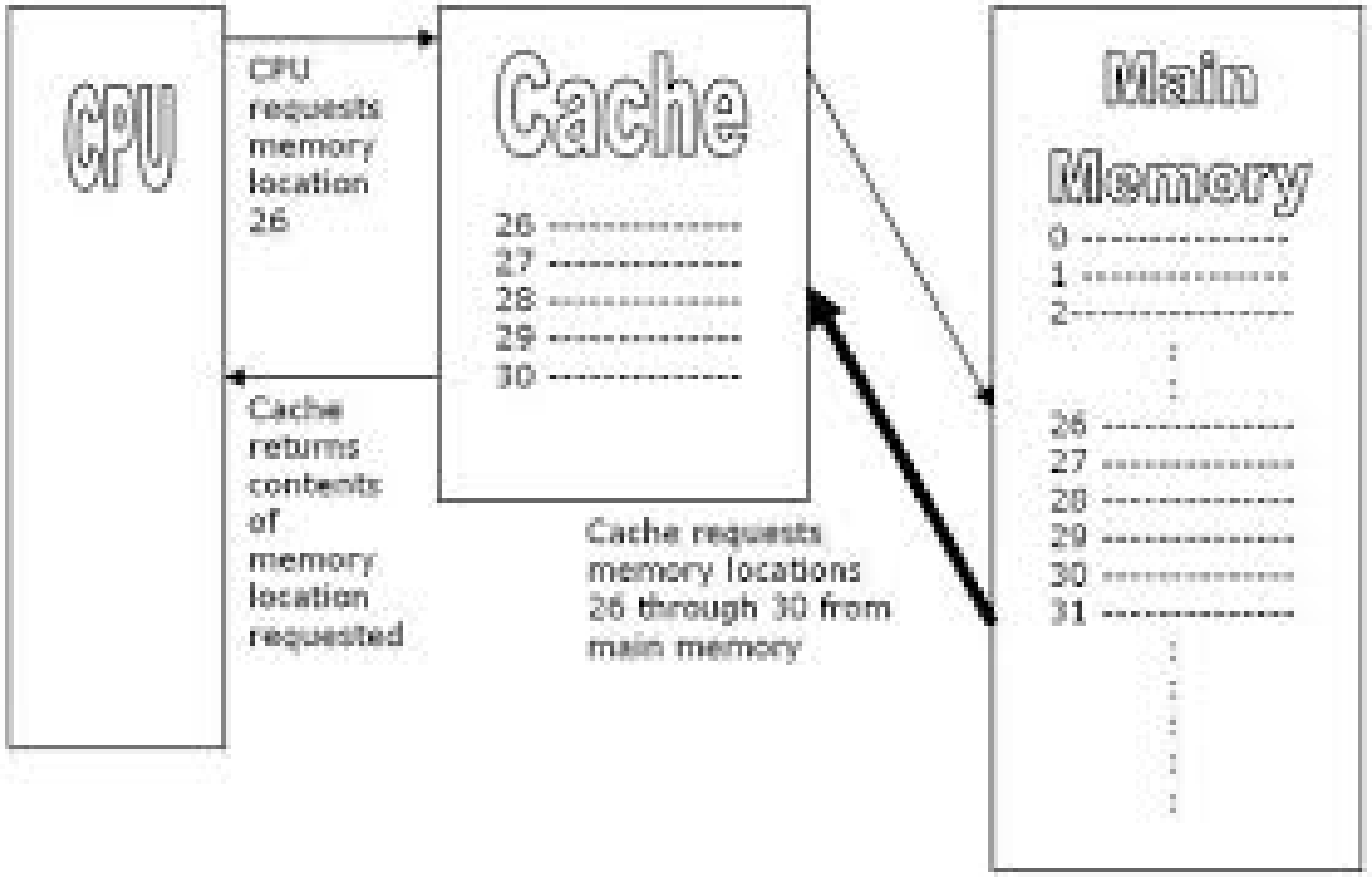



[7] *Fonte: http://csapp.cs.cmu.edu*

# Memória cache



# Memória cache



# Conceitos básicos

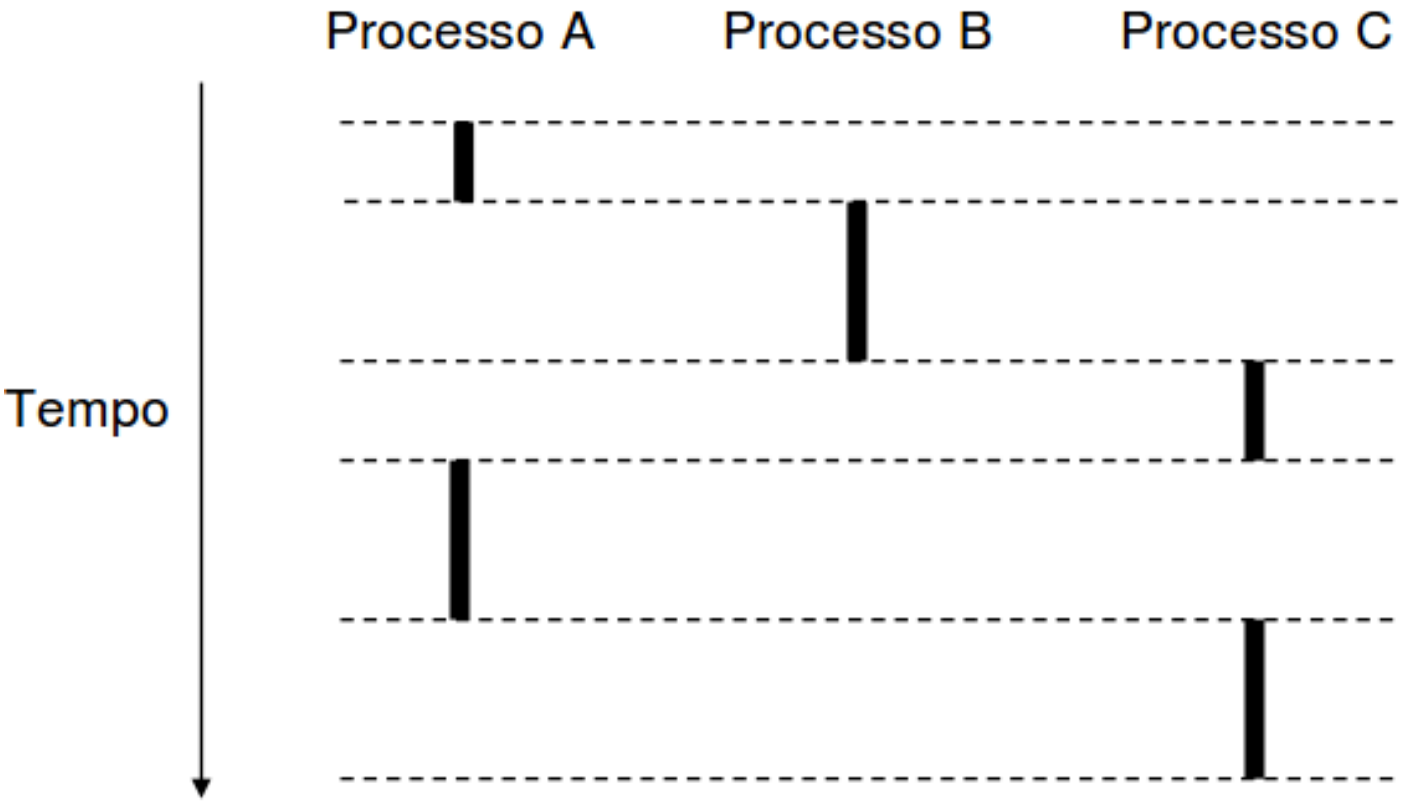
**Processo**: programa em execução

1. **Coleção de recursos**: espaço de memória virtual para armazenar a imagem do processo (código, dados, pilha), dispositivos de E/S alocados, descritores de arquivos abertos, etc.
2. **Escalonamento/execução**: um processo tem um estado de execução e é a entidade que é escalonada pelo Sistema Operacional para execução pelo processador

# Ilusão de execução isolada

- Quando um programa executa, o SO provê a "ilusão" de que o programa dispõe de uso exclusivo do processador, da memória e dos dispositivos de E/S
- Essa "ilusão" é criada pelo conceito de **processo**
- Assim, vários processos podem executar concorrentemente no mesmo sistema de computação e **cada processo parece ter uso exclusivo do hardware**

# Altenância de execução entre processos




Um único processador aparenta executar vários processos ao mesmo tempo, executando um pedaço de cada até terminarem

8

[8] *Fonte: http://csapp.cs.cmu.edu*

# Multiprocessadores X desempenho

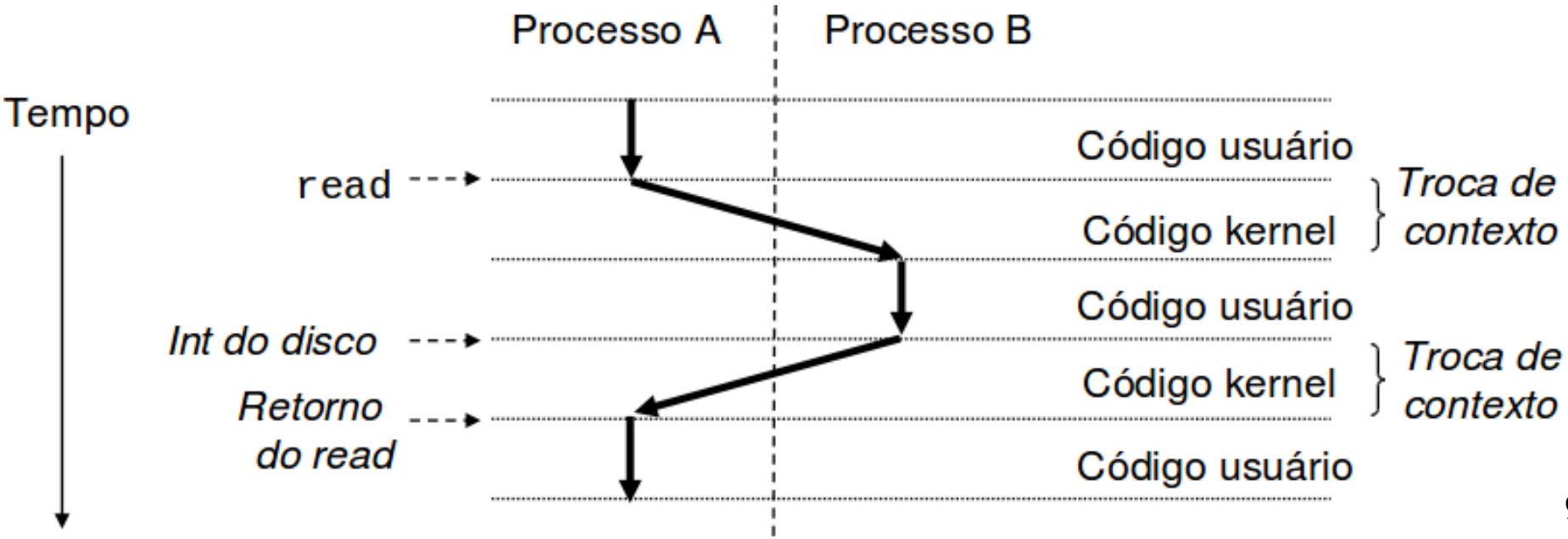
**Multiprocessadores** podem melhorar o **desempenho** em duas frentes:

1. **reduzindo a necessidade de simular concorrência**, quando executando várias aplicações
2. **executando uma mesma aplicação em menos tempo**, se o programa é implementado com várias threads que podem efetivamente executar em paralelo

# Troca de contexto

- Quando o SO transfere o controle do processo atual para algum novo processo, ele executa uma **troca de contexto**:
  1. salva o contexto do processo atual
  2. restaura o contexto do novo processo
  3. passa o controle para o novo processo
- O novo processo retoma a sua execução exatamente do ponto onde ele parou anteriormente

# Troca de contexto



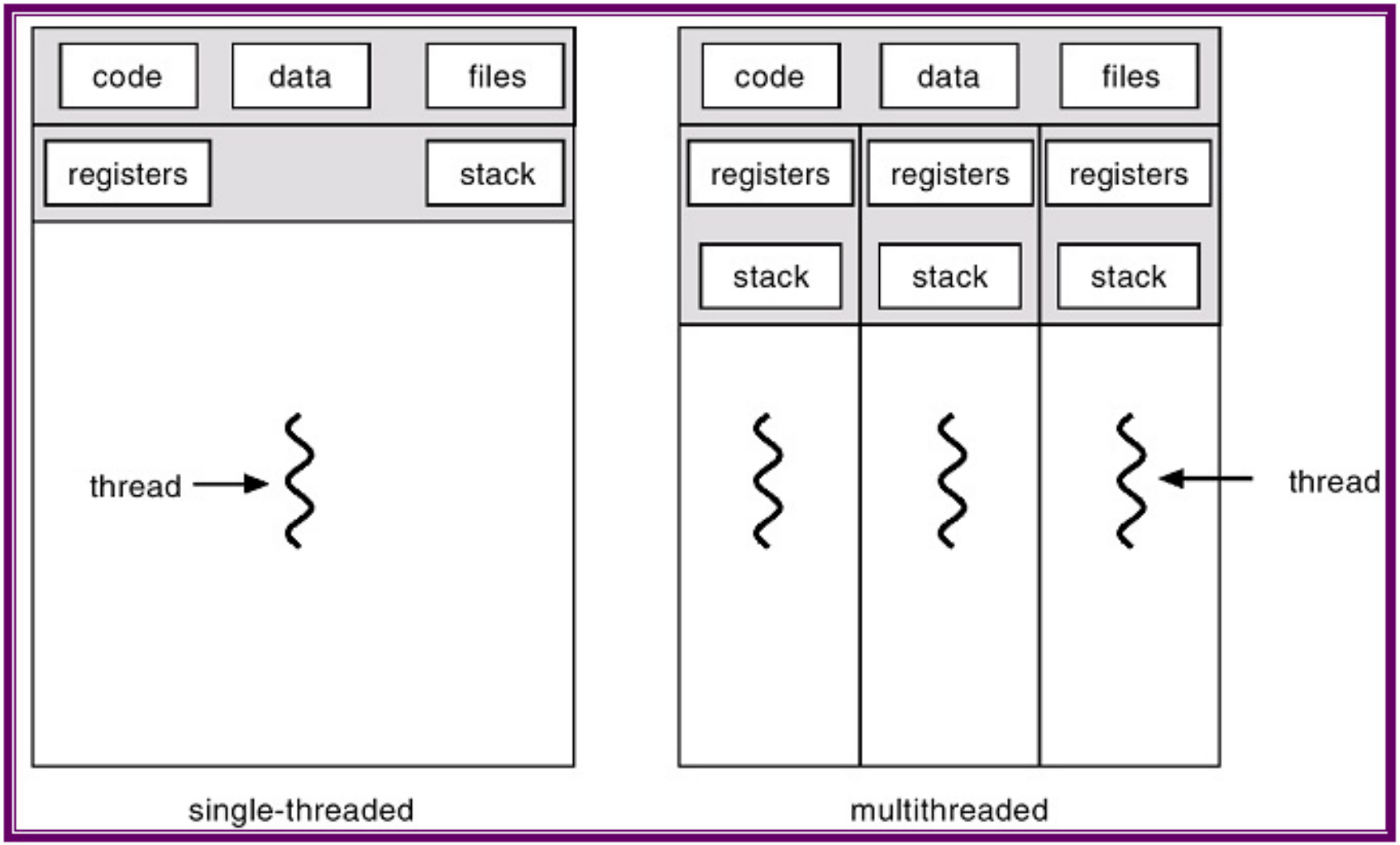

[9] *Fonte: http://csapp.cs.cmu.edu*

# Threads

- Um processo pode consistir de **várias unidades de execução** chamadas *threads*
- Cada uma executa dentro do contexto do processo e **compartilha o mesmo código e dados globais com as outras threads**

É mais fácil compartilhar dados entre threads do que entre processos, e a troca de contexto entre elas é menos custosa do que a troca de contexto entre processos

# Threads dentro de processos

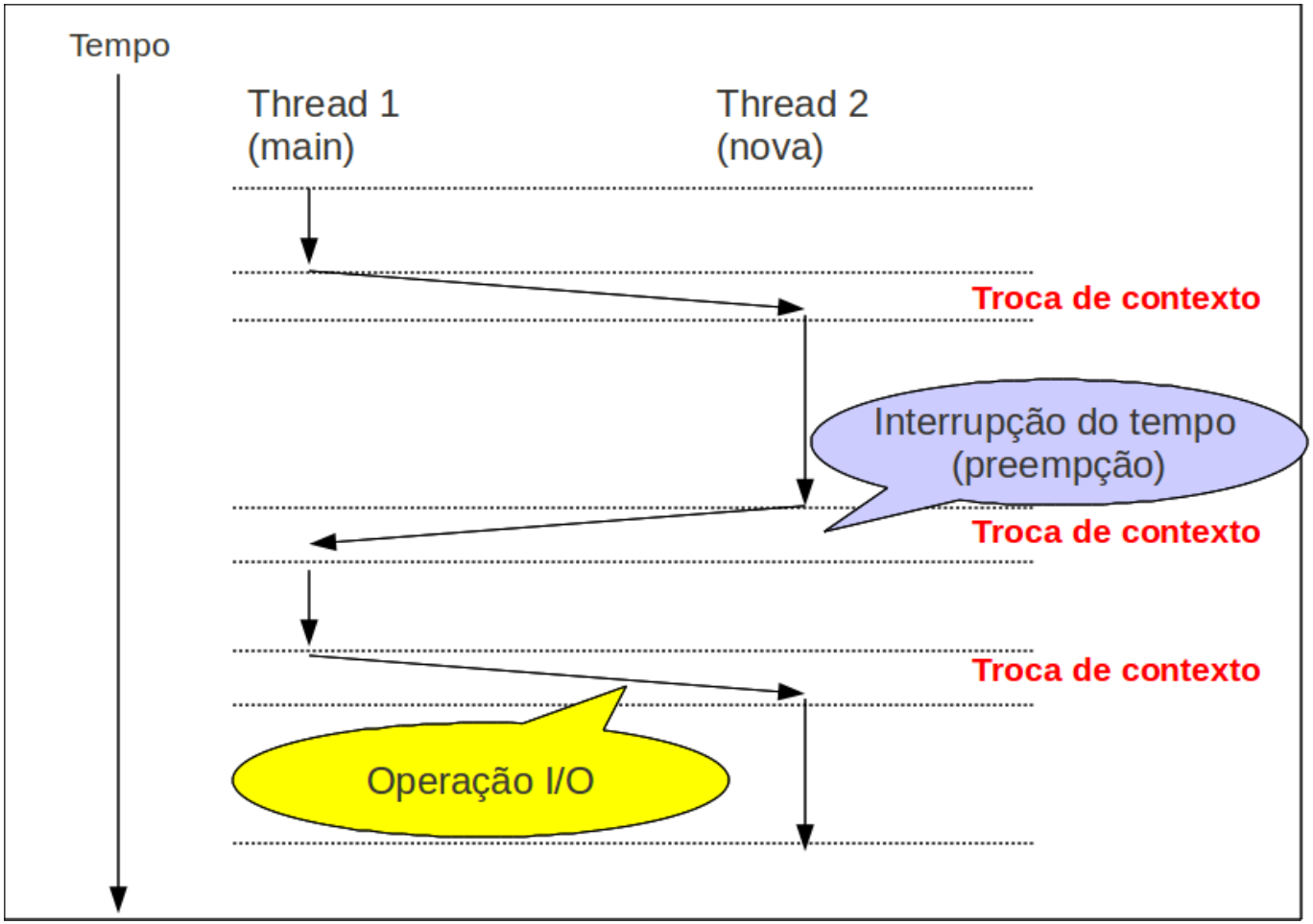



[10] *Fonte: www.csc.villanova.edu*

# Threads

- Uma **thread** é uma unidade básica de uso da CPU (escalonada pelo processador) e compreende:
  - **um identificador da thread, um conjunto de registradores e uma pilha**
- Compartilha com outras threads do mesmo processo:
  - **seção de código e de dados, arquivos abertos, conexões de rede e sinais**

# Modelo de execução de threads compartilhando o mesmo processador



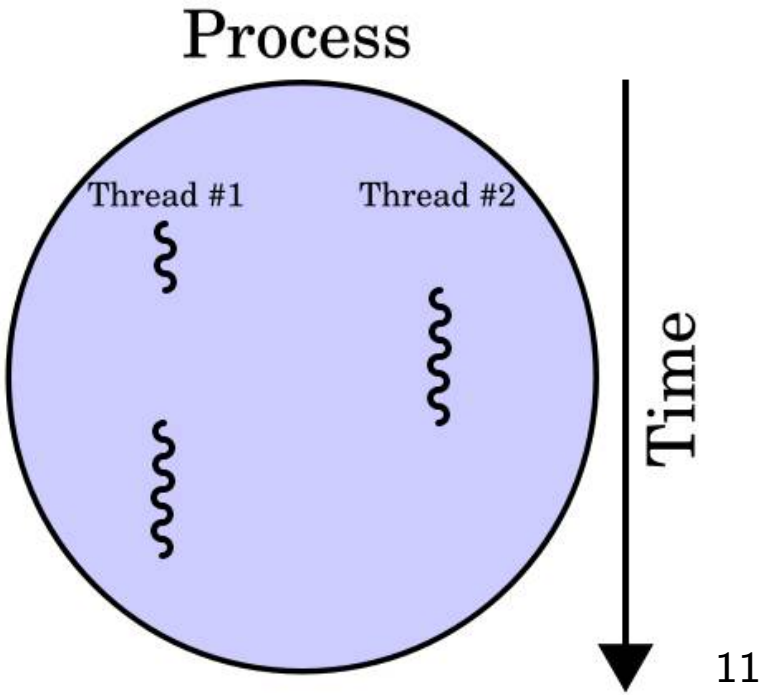
# Processos X Threads

Com base na **abstração de processo**, podemos ter sistemas onde vários programas executam ao mesmo tempo, criando a noção de concorrência

Com base na **abstração de threads**, podemos ter **vários fluxos de controle executando dentro de um único processo**, ampliação da concorrência

# Multithreading

Quando o SO permite várias linhas de execução independentes (threads) dentro do mesmo processo, usa-se o termo **multithreading**




[11] *Fonte: http://wapedia.mobi*

# Benefícios de threads

- Menos tempo para criar uma thread do que um processo filho
- Menos tempo para terminar uma thread do que um processo
- Menos tempo para trocar o contexto entre threads do mesmo processo
- Mais eficiência de comunicação através do uso de memória compartilhada dentro de um mesmo processo

# Exemplos de aplicações multithreading

- **Execução em background**: em aplicações com interface visual, uma thread pode ser responsável por exibir os menus e capturar os eventos de entrada e outra pode ser responsável por executar os comandos e atualizar a interface
- Normalmente **melhora a percepção de velocidade da aplicação**, permitindo que o programa apresente os próximos comandos enquanto o comando anterior ainda está sendo executado

# Exemplos de aplicações multithreading

**Processamento assíncrono**: elementos assíncronos do programa em threads distintas, *ex., uma thread é responsável por periodicamente fazer um backup da aplicação enquanto outra thread é responsável pelo programa principal*

# Exemplos de aplicações multithreading

**Estrutura modular**: Programas que envolvem uma variedade de atividades ou uma **variedade de fontes e destinos de entrada e saída** podem ser mais fáceis de projetar e manter usando threads

# Exemplos de aplicações multithreading

**Sobreposição de processamento e comunicação**: um processo com várias threads pode computar um lote de dados enquanto lê o próximo lote de um dispositivo

**Aplicações Web**: o uso de várias linhas de execução garante que operações rápidas (ex., exibição de texto) não precisem esperar por operações mais lentas (ex., exibição de imagens)

# Exercícios

1. O que é um "sistema de computação"?
2. Por que o sistema de memória é organizado em diferentes níveis e dispositivos? (custos e conceito de *localidade*)
3. De que forma essa forma de organização da memória pode afetar a **programação concorrente**? (armazenamento local de valores globais, ex., registradores, cache)

# Exercícios

1. O que é um "processo"?
2. Como o sistema de computação provê a ilusão de execução simultânea de vários processos usando um único processador?
3. O que é uma "thread"?
4. Quais as vantagens da programação "multithreading"? Em que situações ela pode ser usada?

# Referências bibliográficas

- *Computer Systems - A Programmer's Perspective, Bryant and O'Hallaron, 2ed* (**Cap.1**)
(http://csapp.cs.cmu.edu/public/ch1-preview.pdf)