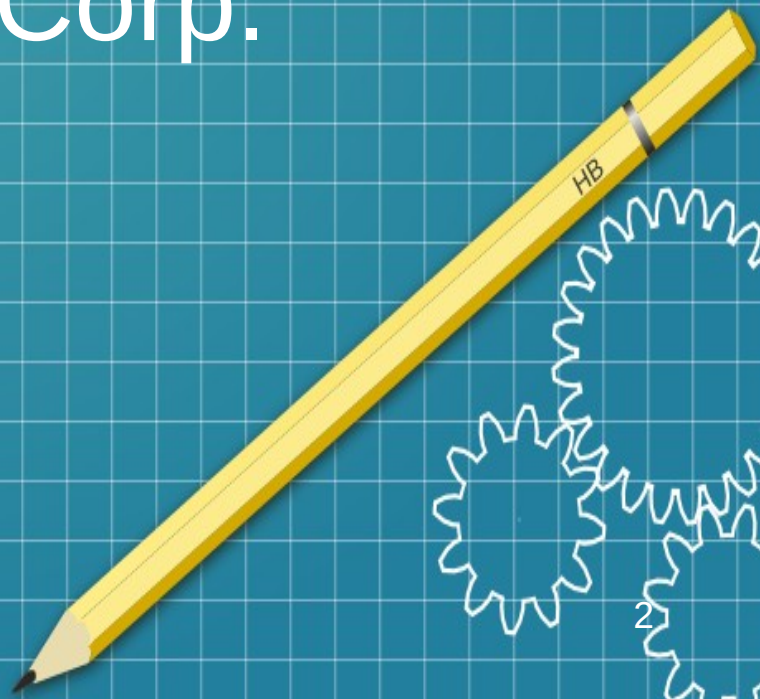
CRSC08 – Tópicos Especiais em Programação

OpenMP

UNIFEI
Universidade Federal de Itajubá
IMC – Instituto de Matemática e Computação

Prof. Carlos Minoru Tamaki
Prof. Roberto Claudino da Silva

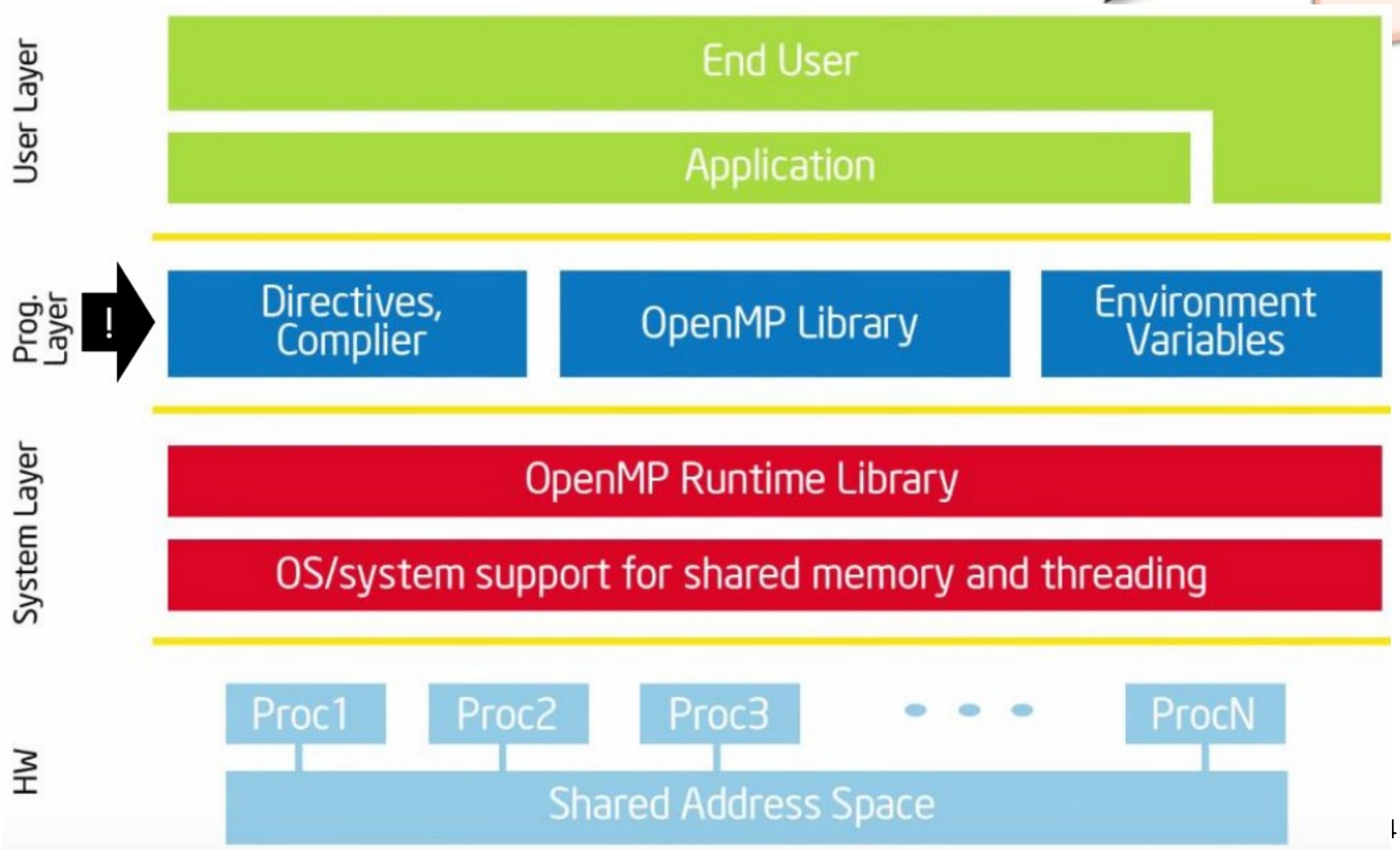
Slides baseados no curso de Tim Mattson da Intel Corp.



VISÃO GERAL OPENMP:

- OpenMP: Uma API para escrever aplicações Multithreaded
- Um conjunto de diretivas do compilador e biblioteca de rotinas para programadores de aplicações paralelas
- Simplifica muito a escrita de programas multi-threaded (MT)
- Padroniza a prática SMP

OPENMP DEFINIÇÕES BÁSICAS: PILHA SW



SINTAXE BÁSICA OPENMP

- Tipos e protótipos de funções no arquivo:

  #include <omp.h>

- A maioria das construções OpenMP são diretivas de compilação.

  #pragma omp construct [clause [clause]...]

- Exemplo:

  #pragma omp parallel num_threads(4)

- A maioria das construções se aplicam a um **“bloco estruturado”** (basic block).
- **Bloco estruturado:** Um bloco com um ou mais declarações com um ponto de entrada no topo e um ponto de saída no final.
- Podemos ter um exit() dentro de um bloco desses.

NOTAS DE COMPILAÇÃO

- Linux e OS X com gcc gcc:

  gcc -fopenmp foo.c

  export OMP_NUM_THREADS=4

  ./a.out

EXERCÍCIO 1, PARTE A: HELLO WORLD

- Verifique se seu ambiente funciona
- Escreva um programa que escreva “hello world”.

```
#include <stdio.h>
int main()
{
        int ID = 0;
        printf(" hello(%d) ", ID);
        printf(" world(%d) \n", ID);
}
```

Compilação:
Gcc hello.c -o hello

EXERCÍCIO 1, PARTE B: HELLO WORLD

- Verifique se seu ambiente funciona
- Escreva um programa multithreaded que escreva “hello world”.

```
#include <stdio.h>
#include <omp.h>
int main() {
        int ID = 0;
        #pragma omp parallel
        {
                printf(" hello(%d) ", ID);
                printf(" world(%d) \n", ID);
        }
}
```

Compilação:
gcc -fopenmp hello1.c -o hello1

EXERCÍCIO 1, PARTE C: HELLO WORLD

- Verifique se seu ambiente funciona
- Vamos adicionar o número da thread ao “hello world”.

```
#include <stdio.h>
#include <omp.h>
int main() {
        #pragma omp parallel
        {
                int ID = omp_get_thread_num();
                printf(" hello(%d) ", ID);
                printf(" world(%d) \n", ID);
        }
}
```

EXERCÍCIO 1: SOLUÇÃO

```
#include <stdio.h>
#include <omp.h>
int main() {
        #pragma omp parallel
        {
                int ID = omp_get_thread_num();
                printf(" hello(%d) ", ID);
                printf(" world(%d) \n", ID);
        }
}
```

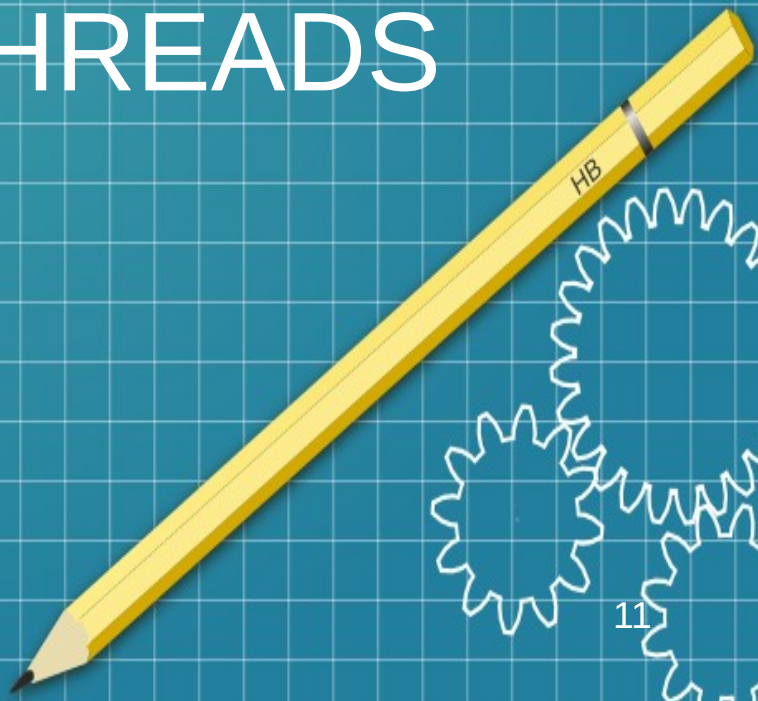
Arquivo OpenMP

Região paralela com um número padrão de threads

Função da biblioteca que retorna o thread ID.

Fim da região paralela

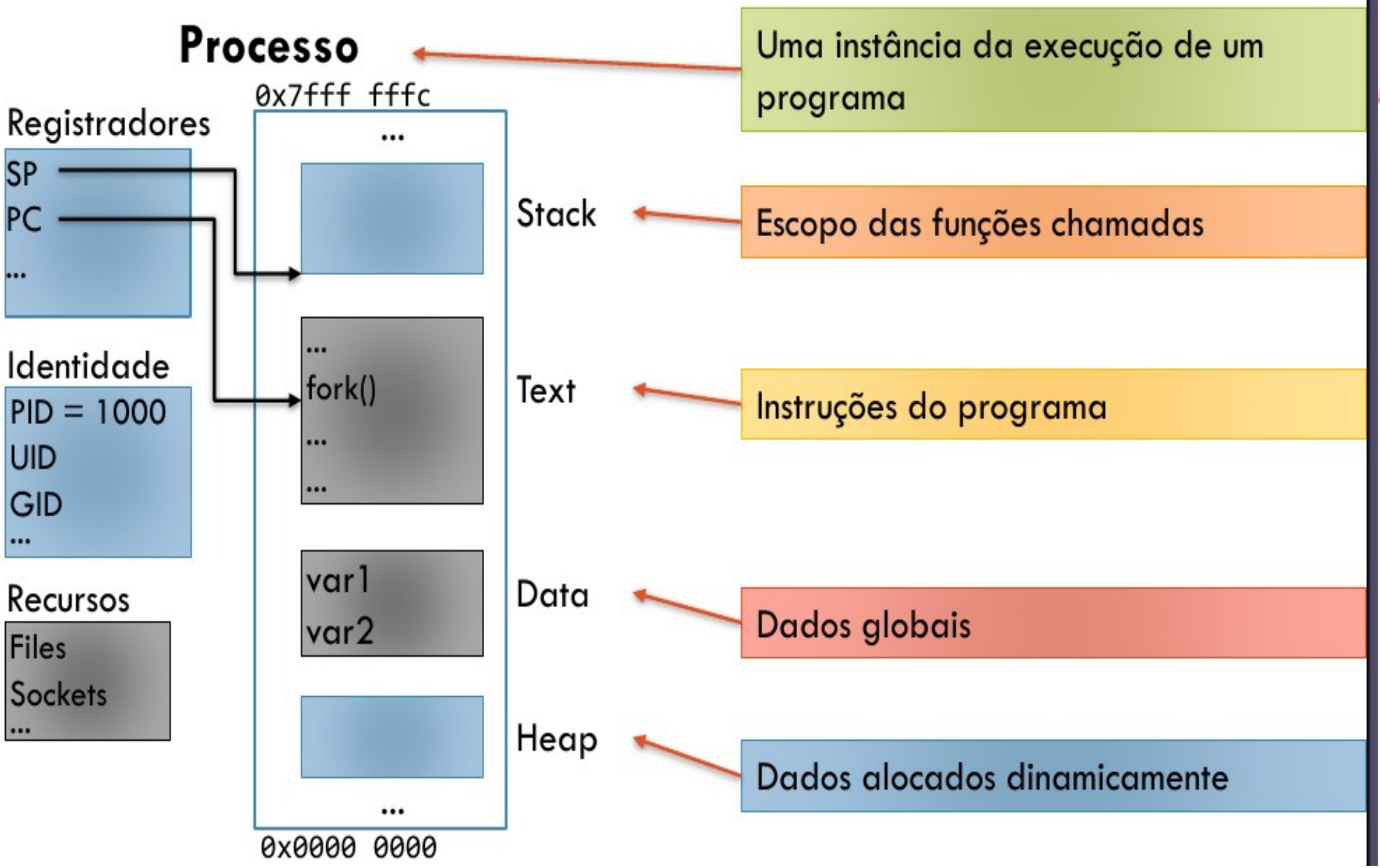
FUNCIONAMENTO DOS PROCESSOS VS. THREADS



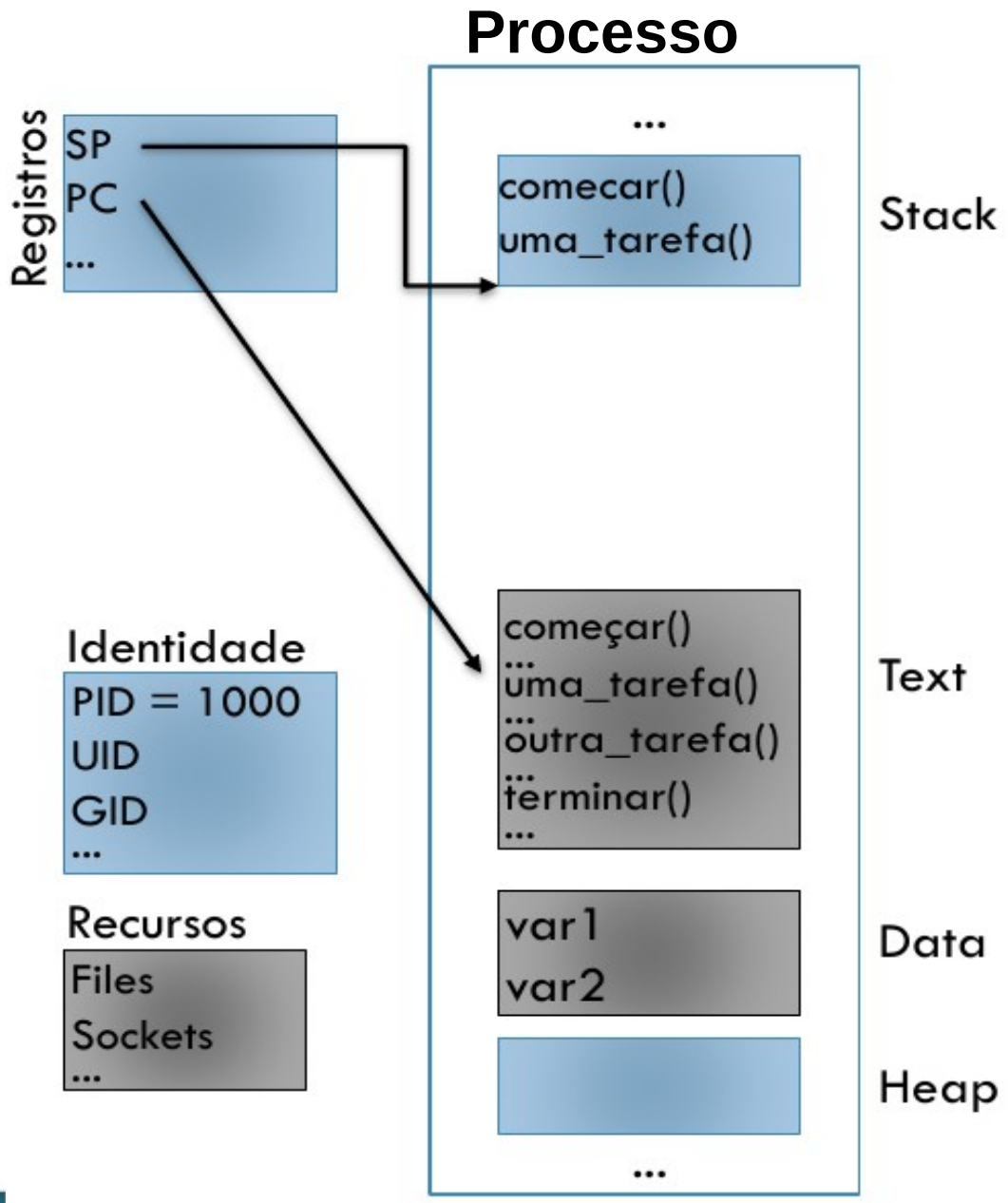
MULTIPROCESSADORES MEMÓRIA COMPARTILHADA

- Um multiprocessadores é um computador em que todos os processadores partilham o acesso à memória física.
- Os processadores executam de forma independente mas o espaço de endereçamento global é partilhado.
- Qualquer alteração sobre uma posição de memória realizada por um determinado processador é igualmente visível por todos os restantes processadores.
- Existem duas grandes classes de multiprocessadores :
  - Uniform Memory Access Multiprocessor (UMA) ou Symmetrical Multiprocessor (SMP)
  - Non-Uniform Memory Access Multiprocessor (NUMA)

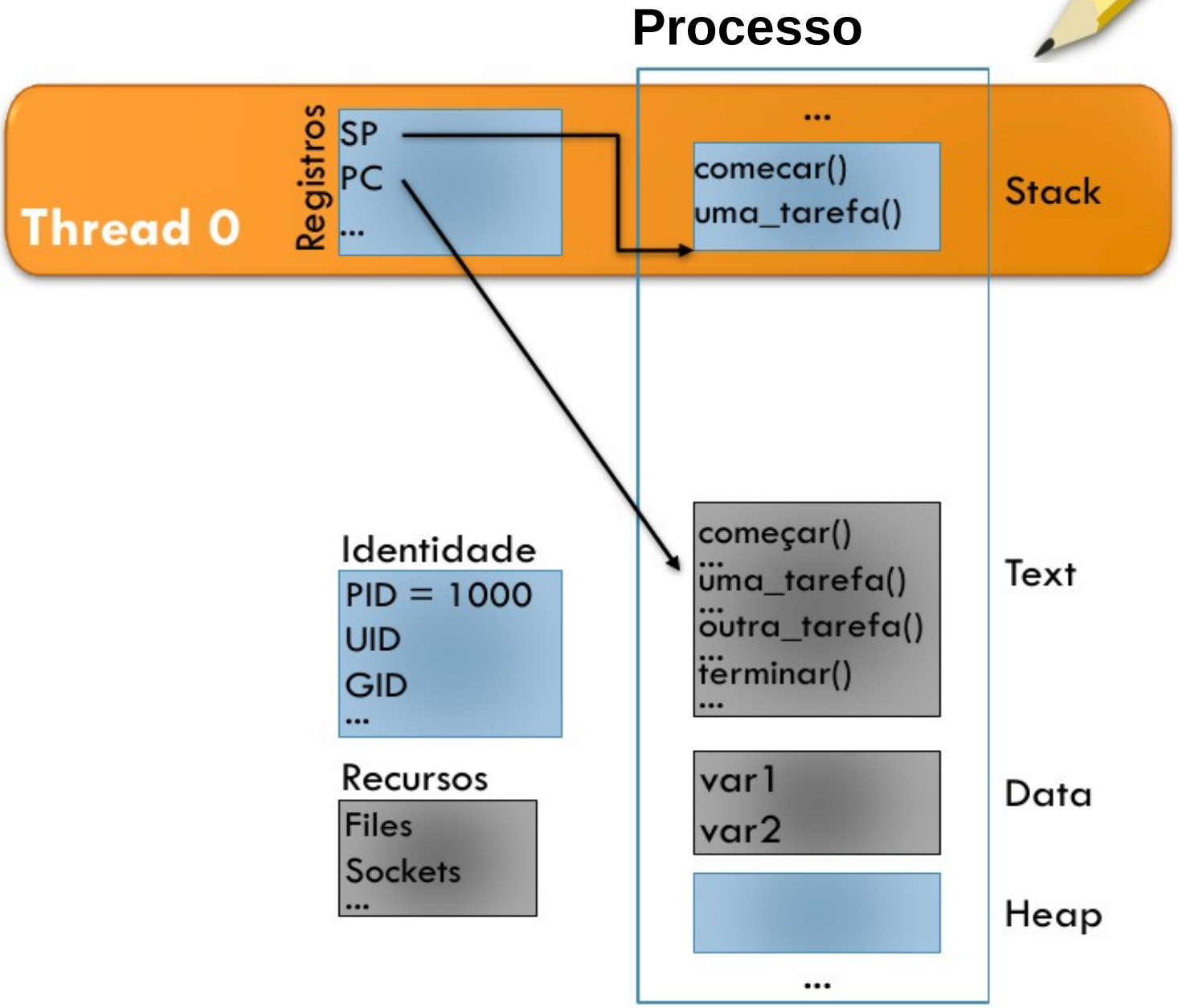
PROCESSOS


Processo
Uma instância da execução de um programa
0x7fff fffc
...
Registradores
SP
PC
...
Stack
Escopo das funções chamadas
...
fork()
...
...
Identidade
PID = 1000
UID
GID
...
Text
Instruções do programa
var1
var2
Recursos
Files
Sockets
...
Data
Dados globais
Heap
Dados alocados dinamicamente
...
0x0000 0000

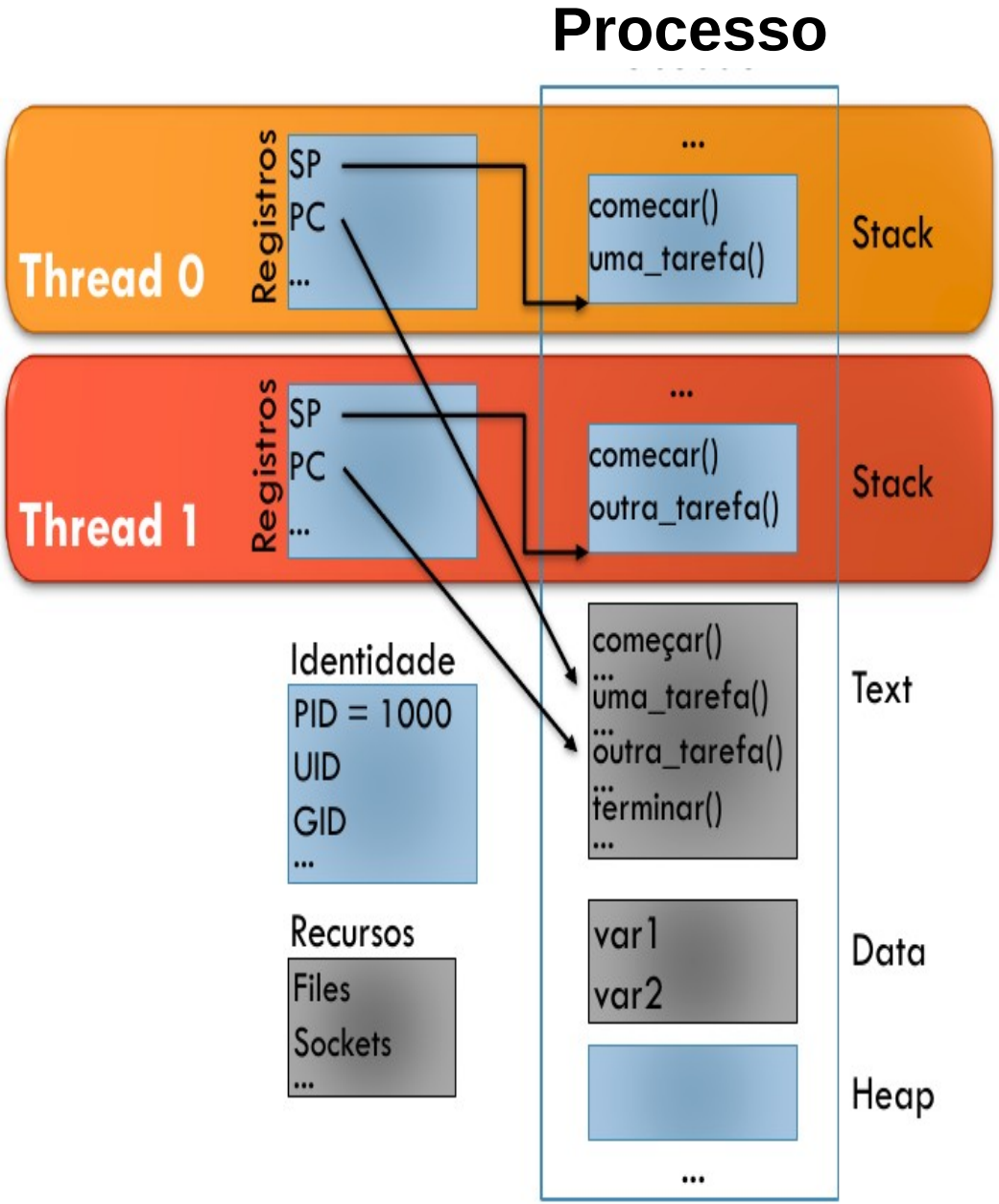
MULTITHREADING



MULTITHREADING



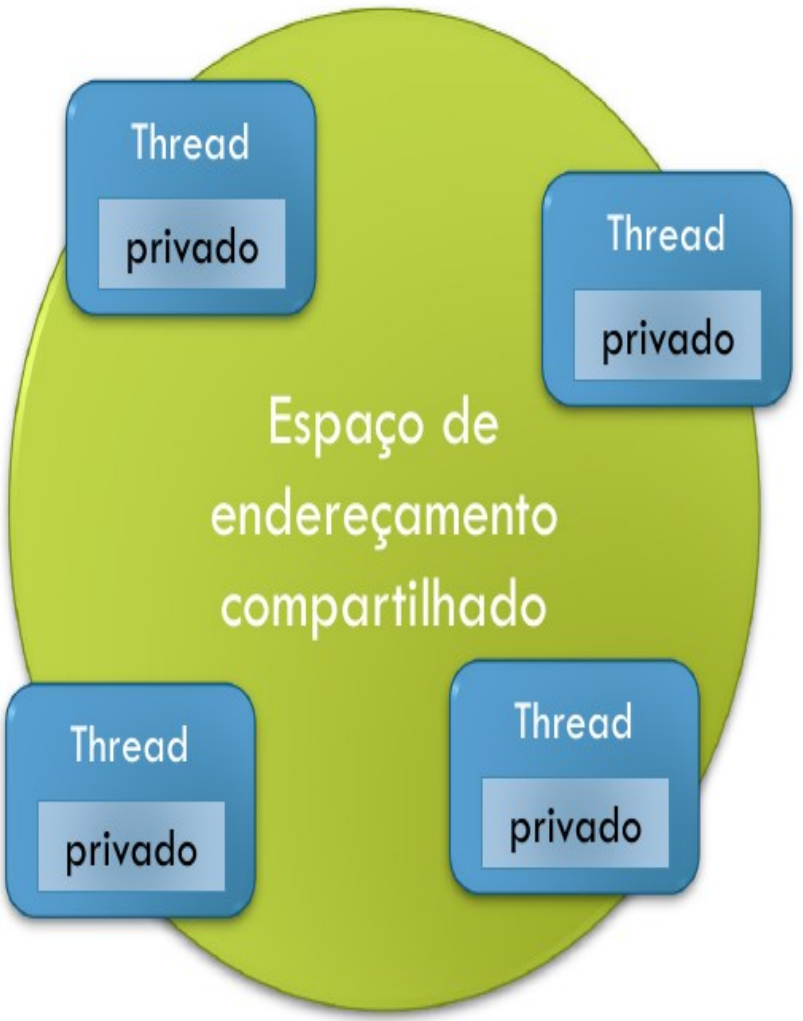
MULTITHREADING



- Cada thread mantém suas chamadas de funções (stack) e suas variáveis locais
- Espaço de endereçamento único
- Variáveis globais/dinâmicas podem ser acessadas por qualquer thread
- Troca de contexto rápida (dados compartilhados)

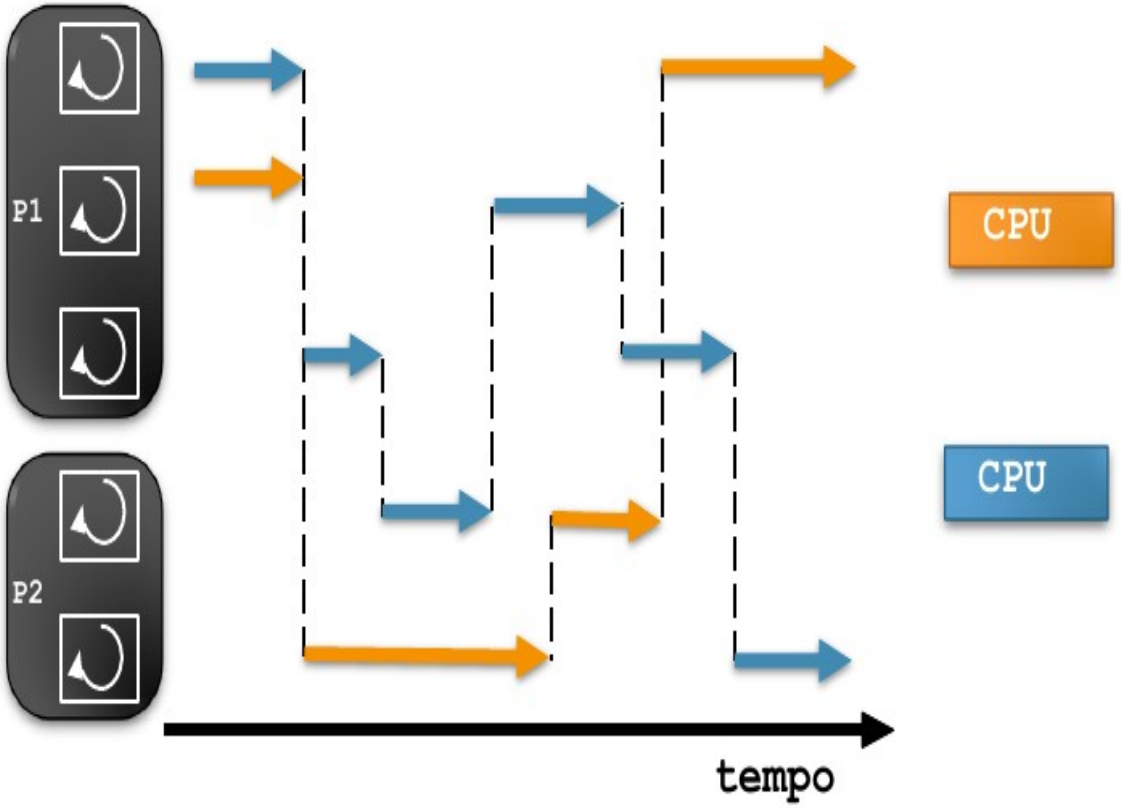
UM PROGRAMA DE MEMÓRIA COMPARTILHADA

- Uma instância do programa:
- Um processo e muitas threads.
- Threads interagem através de leituras/escrita com o espaço de endereçamento compartilhado.
- Escalonador SO decide quando executar cada thread (entrelaçado para ser justo).
- Sincronização garante a ordem correta dos resultados.



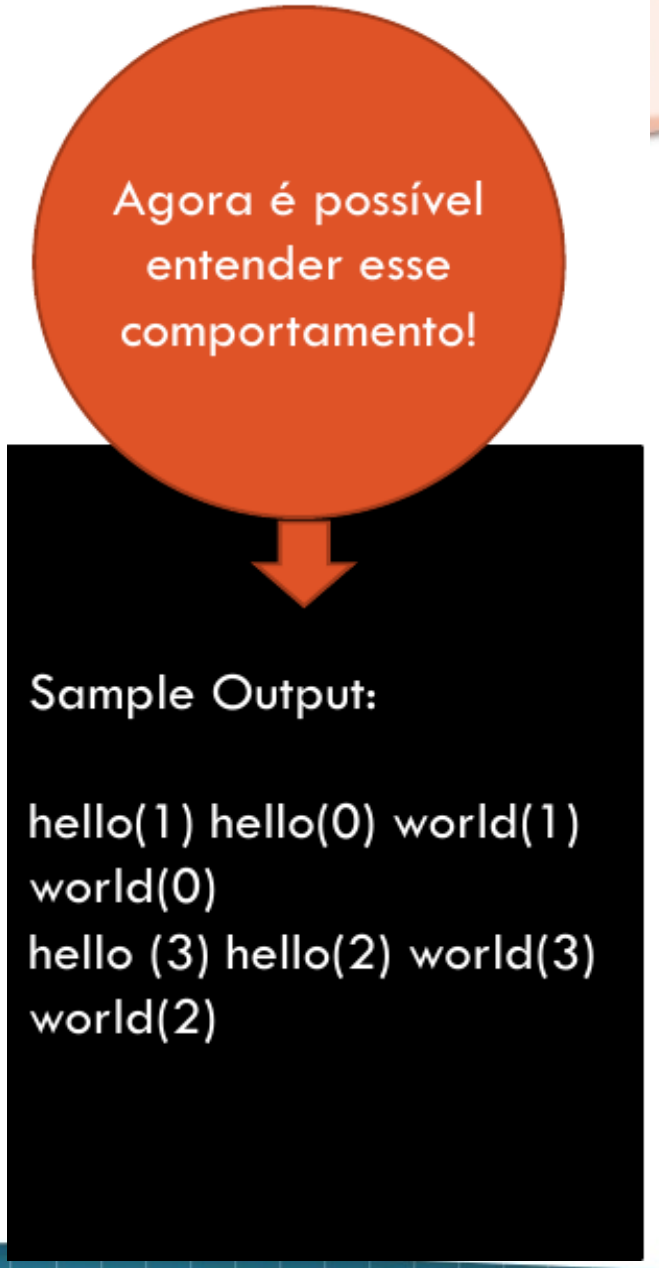
EXECUÇÃO DE PROCESSOS MULTITHREADED

- Todos os threads de um processo podem ser executados concorrentemente e em diferentes processadores, caso existam.



EXERCÍCIO 1: SOLUÇÃO

```
#include <stdio.h>
#include <omp.h>
int main() {
        #pragma omp parallel
        {
                int ID;
                ID  = omp_get_thread_num();
                printf(" hello(%d) ", ID);
                printf(" world(%d) \n", ID);
        }
}
```



VISÃO GERAL DE OPENMP: COMO AS THREADS INTERAGEM?

- OpenMP é um modelo de multithreading de memória compartilhada.
  - Threads se comunicam através de variáveis compartilhadas.
- Compartilhamento não intencional de dados causa condições de corrida.
  - Condições de corrida: quando a saída do programa muda quando a threads são escalonadas de forma diferente.
- Apesar de este ser um aspectos mais poderosos da utilização de threads, também pode ser um dos mais problemáticos.
- O problema existe quando dois ou mais threads tentam acessar/alterar as mesmas estruturas de dados (condições de corrida).
- Para controlar condições de corrida:
  - Usar sincronização para proteger os conflitos por dados
- Sincronização é cara, por isso:
  - Tentaremos mudar a forma de acesso aos dados para minimizar a necessidade de sincronizações.

SINCRONIZAÇÃO E REGIÕES CRÍTICAS: EXEMPLO

Tempo	Th1	Th2	Saldo
T0			$200
T1	Leia Saldo $200		$200
T2		Leia Saldo $200	$200
T3		Some $100 $300	$200
T4	Some $150 $350		$200
T5		Escreva Saldo $300	$300
T6	Escreva Saldo $350		$350

SINCRONIZAÇÃO E REGIÕES CRÍTICAS: EXEMPLO

Tempo	Th1	Th2	Saldo
T0			
T			
T			
T			
T			
T5		Escreva Saldo $300	$300
T6	Escreva Saldo $350		$350

Devemos garantir que **não importa a ordem de execução (escalonamento)**, teremos sempre um resultado consistente!