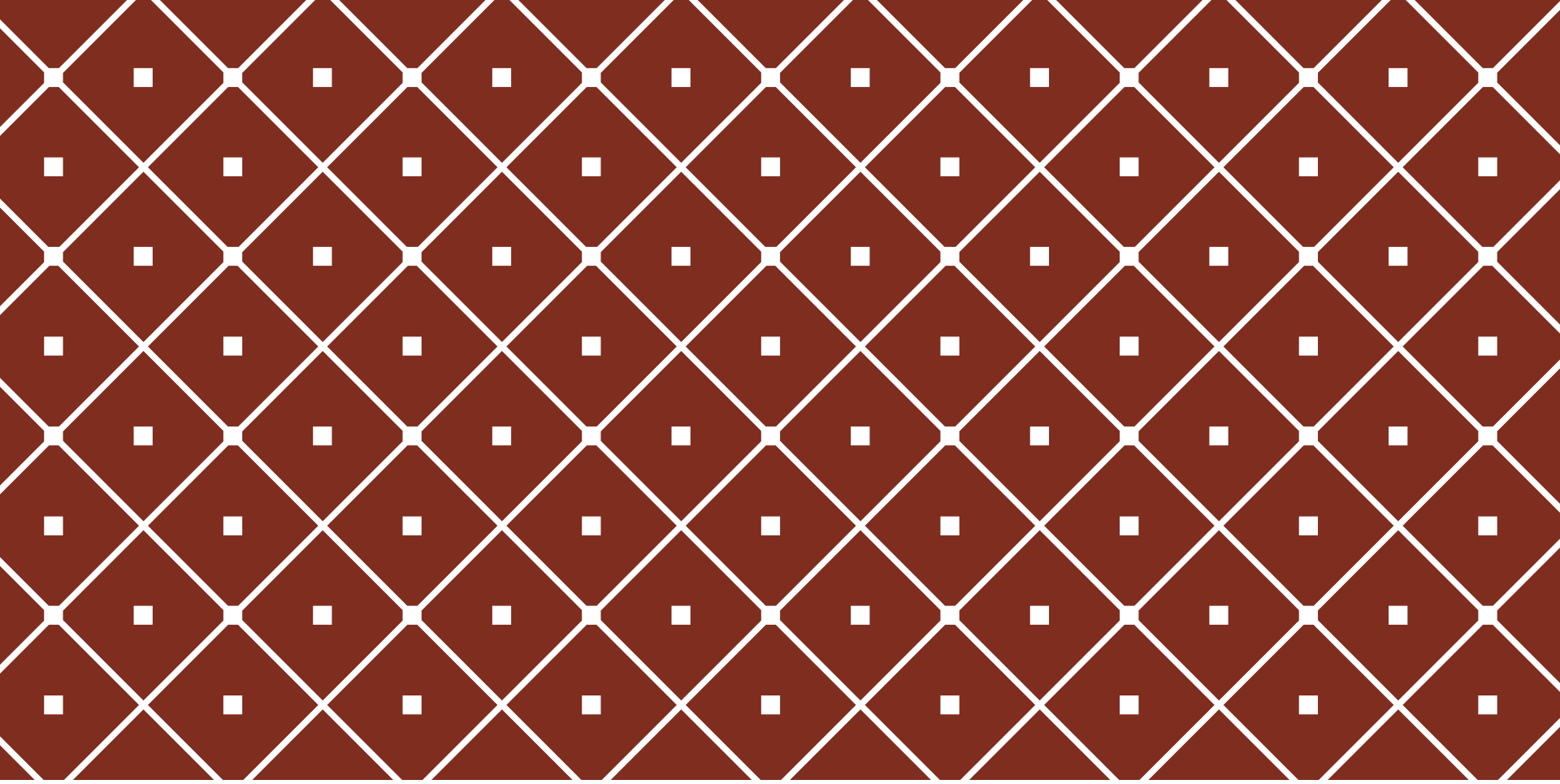


PROGRAMAÇÃO PARALELA OPENMP

Marco A. Zanata Alves

Slides baseados no material de Tim Mattson - Intel Corp. (2010)
A "Hands-on" Introduction to OpenMP *

COMO FAZER ATIVIDADES EM PARALELO? NO MUNDO REAL

Você pode enviar cartas/mensagens aos amigos e pedir ajuda

Você pode criar uma lista de tarefas (pool)

Quando um amigo ficar atoa, pegue uma nova tarefa da lista

Você pode ajudar na tarefa ou então ficar apenas gerenciando

Mais amigos, mais tempo para eles chegarem/ se acomodarem

Tarefas curtas ou longas?

Muitos amigos olhando e riscando a lista?

Precisa gerenciar algo mais?

O QUE É NECESSÁRIO?

Definir...

- O que é uma tarefa
- Atividades por tarefa (granularidade)
- Protocolo de trabalho (divisão de tarefas)
- Como retirar uma tarefa da lista
- Onde colocar o resultado final
- Como reportar eventualidades
- Existe ordenação/sincronia entre as tarefas
- Existe dependência/conflito por recursos

INTRODUÇÃO

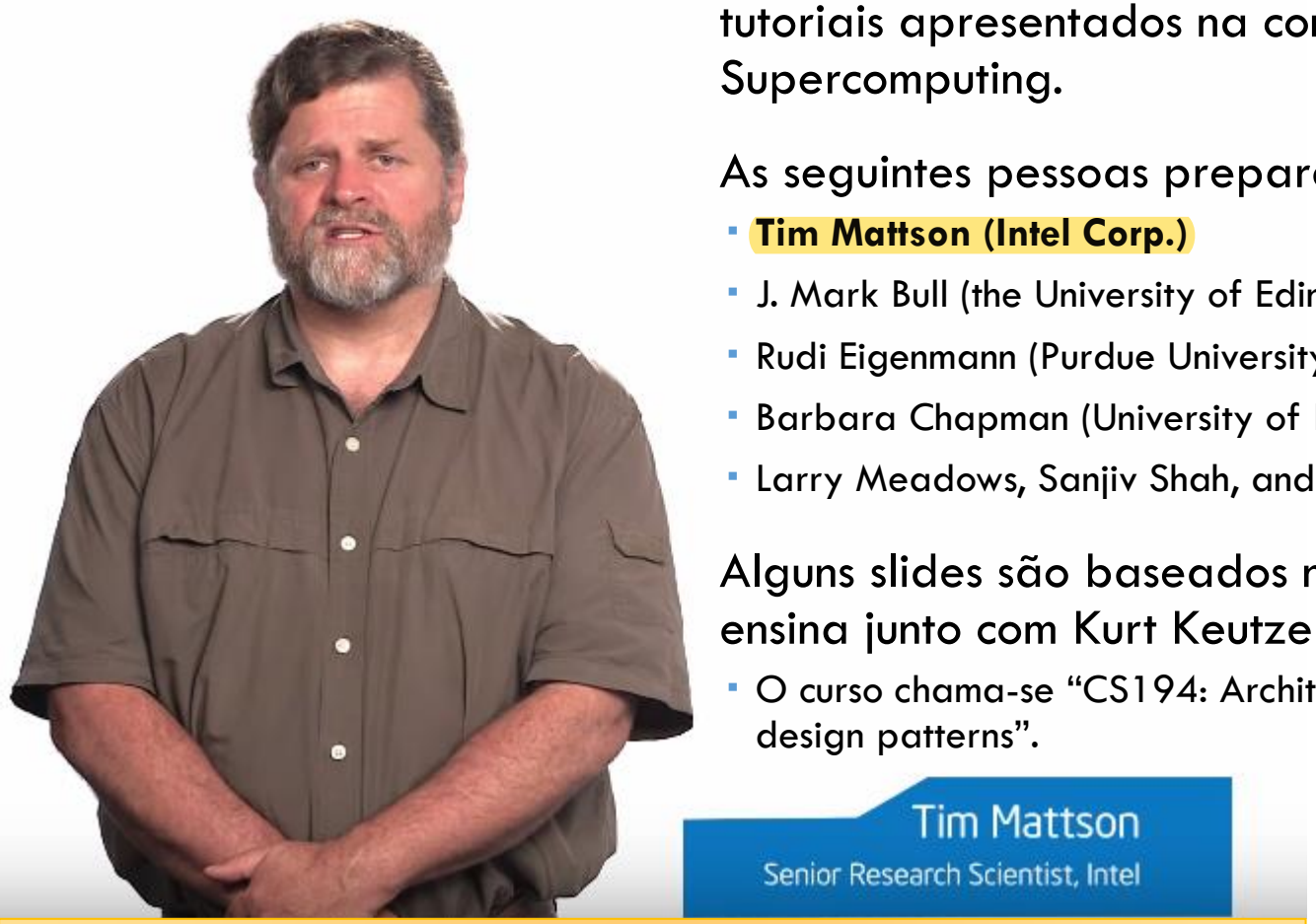
OpenMP é um dos modelos de programação paralelas mais usados hoje em dia.

Esse modelo é relativamente fácil de usar, o que o torna um bom modelo para iniciar o aprendizado sobre escrita de programas paralelos.

Premissas:

- Assumo que todos sabem programar em linguagem C. OpenMP também suporta Fortran e C++, mas vamos nos restringir a C.
- Assumo que todos são novatos em programação paralela.
- Assumo que todos terão acesso a um compilador que suporte OpenMP

AGRADECIMENTOS



Esse curso é baseado em uma longa série de tutoriais apresentados na conferência Supercomputing.

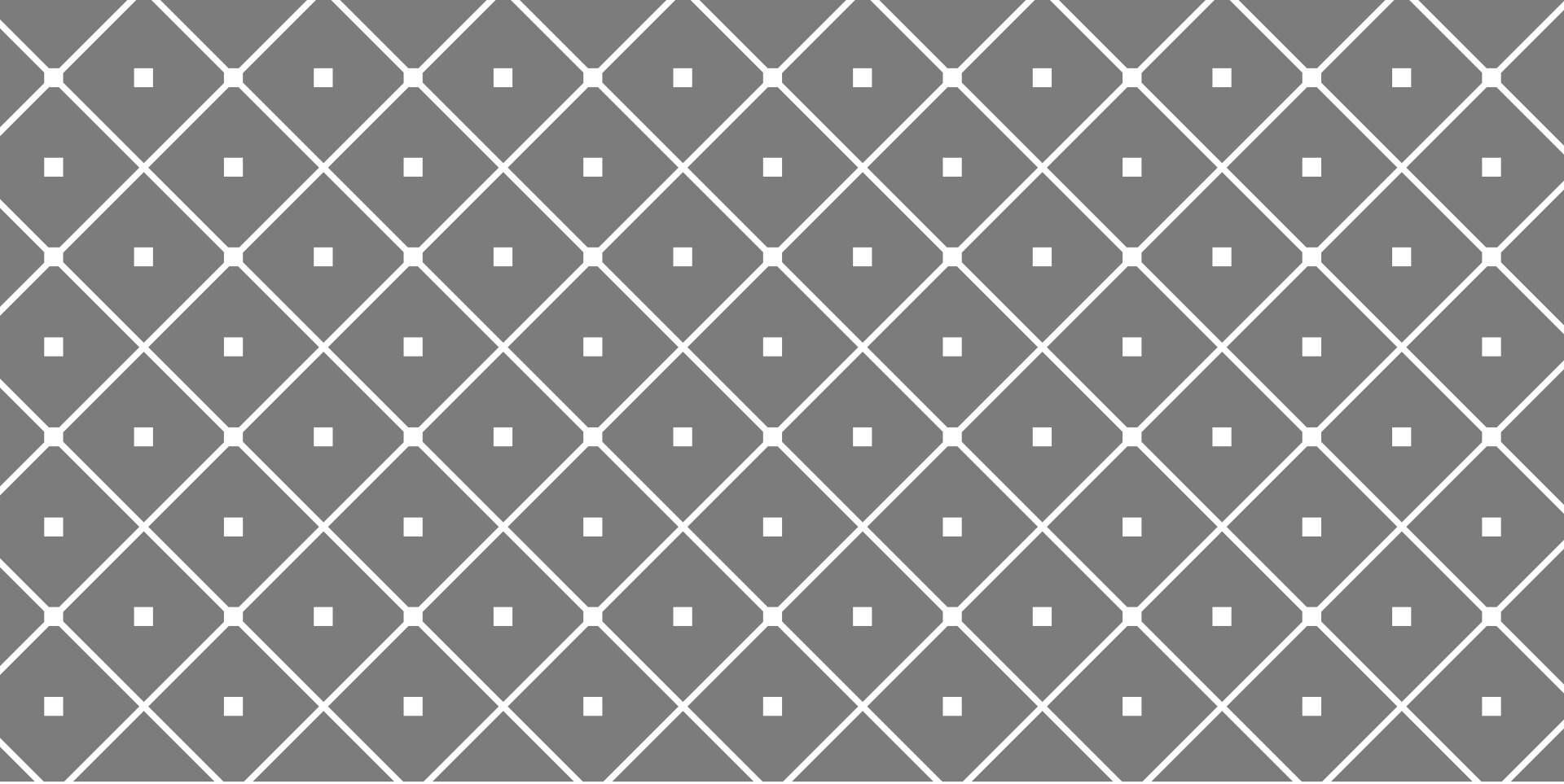
As seguintes pessoas prepararam esse material:

- **Tim Mattson (Intel Corp.)**
- J. Mark Bull (the University of Edinburgh)
- Rudi Eigenmann (Purdue University)
- Barbara Chapman (University of Houston)
- Larry Meadows, Sanjiv Shah, and Clay Breshears (Intel Corp).

Alguns slides são baseados no curso que Tim Mattson ensina junto com Kurt Keutzer na UC Berkeley.

- O curso chama-se "CS194: Architecting parallel applications with design patterns".

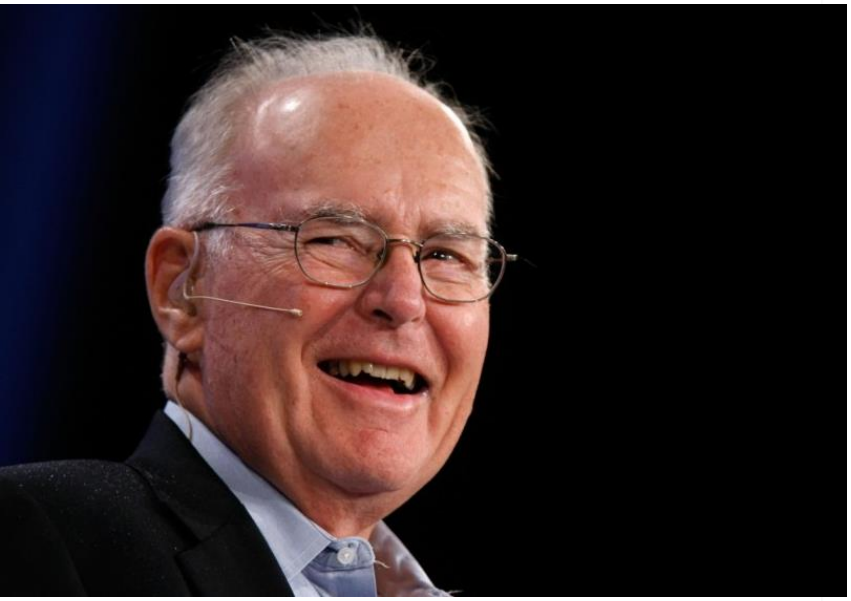
https://www.youtube.com/user/OpenMPARB

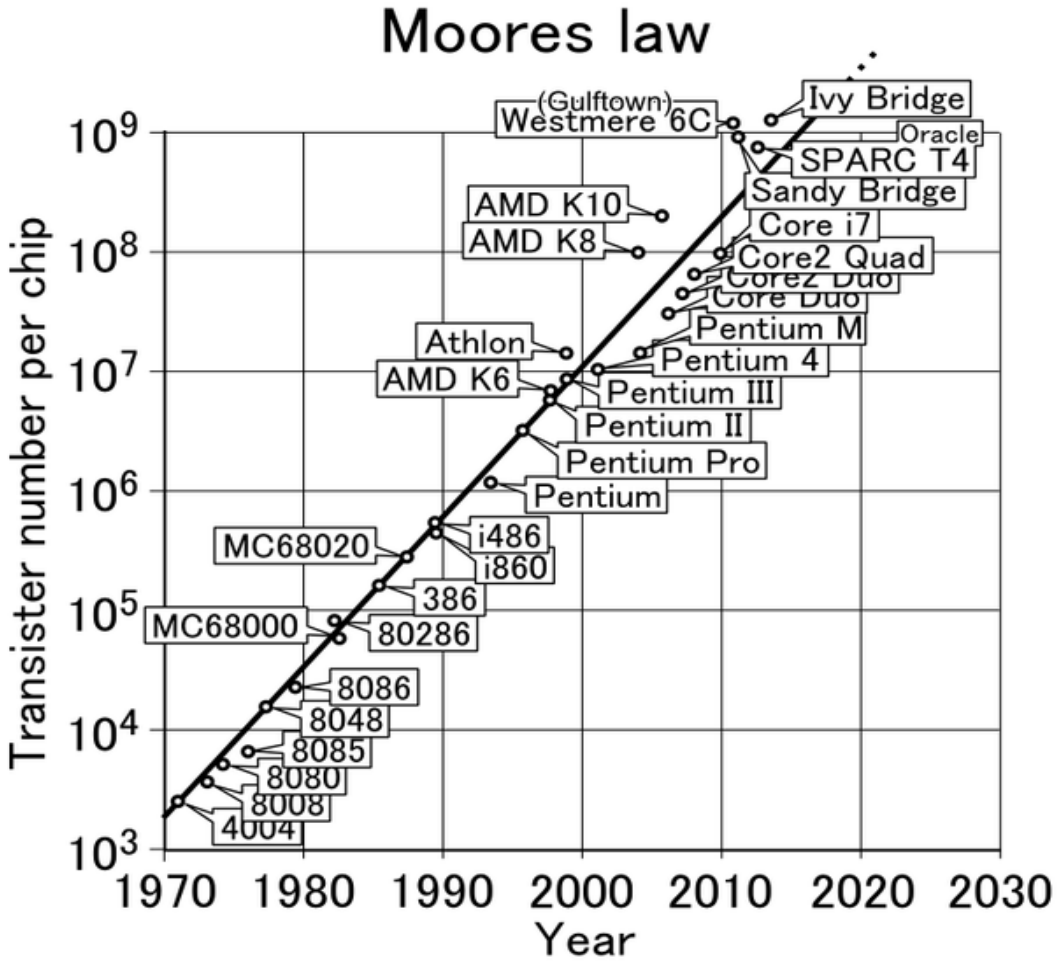


BREVE HISTÓRIA DA PROGRAMAÇÃO PARALELA

Fontes: http://projetodraft.com/selecao-draft-lei-de-moore-2/
http://www.cringely.com/2013/10/15/breaking-moores-law/

LEI DE MOORE



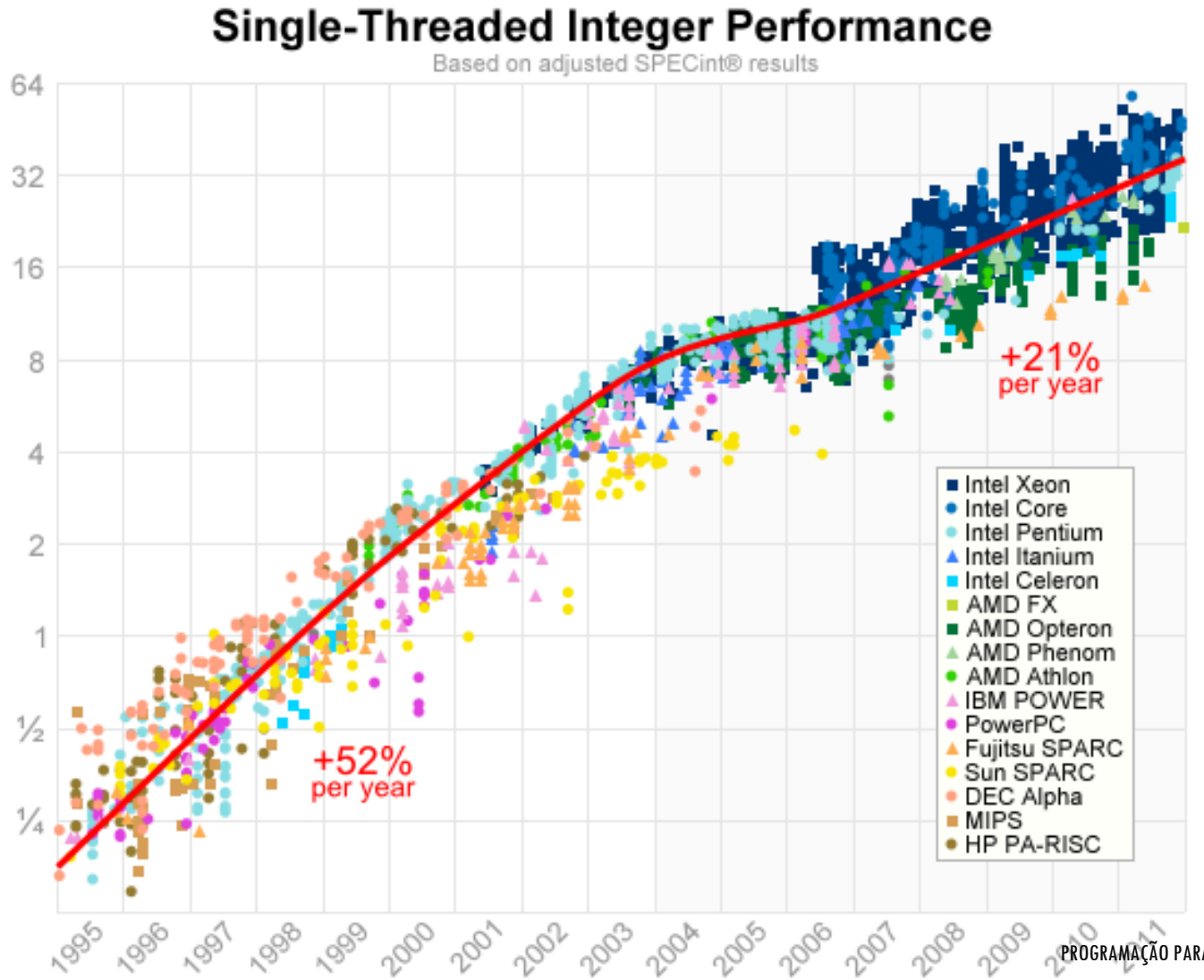


Em 1965, o co-fundador da Intel, Gordon Moore previu (com apenas 3 pontos de dados) que a densidade dos semicondutores iria dobrar a cada 18 meses.

Ele estava certo! Os transistores continuam diminuindo conforme ele projetou

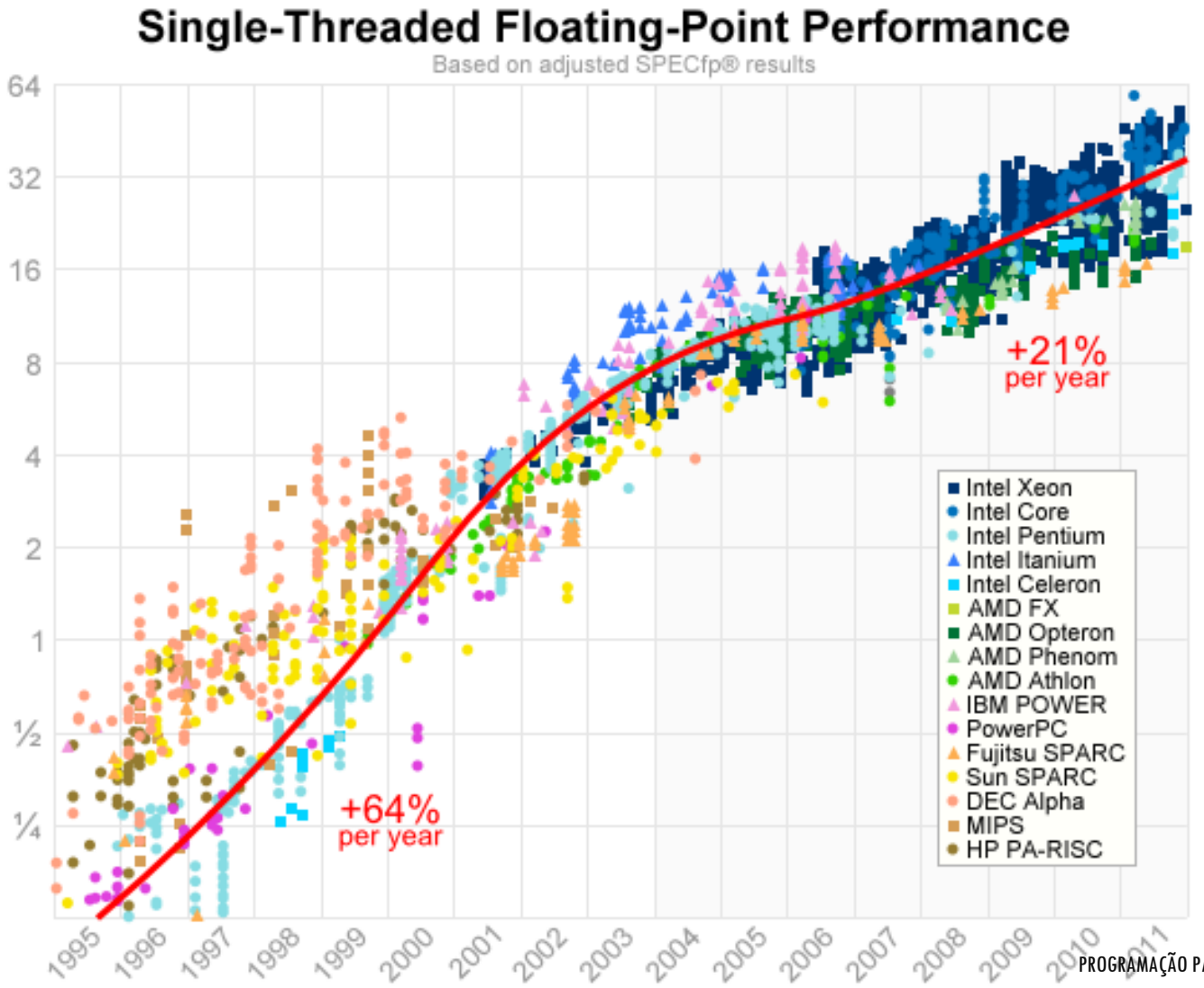
Fonte: http://preshing.com/20120208/a-look-back-at-single-threaded-cpu-performance/

CONSEQUÊNCIAS DA LEI DE MOORE...

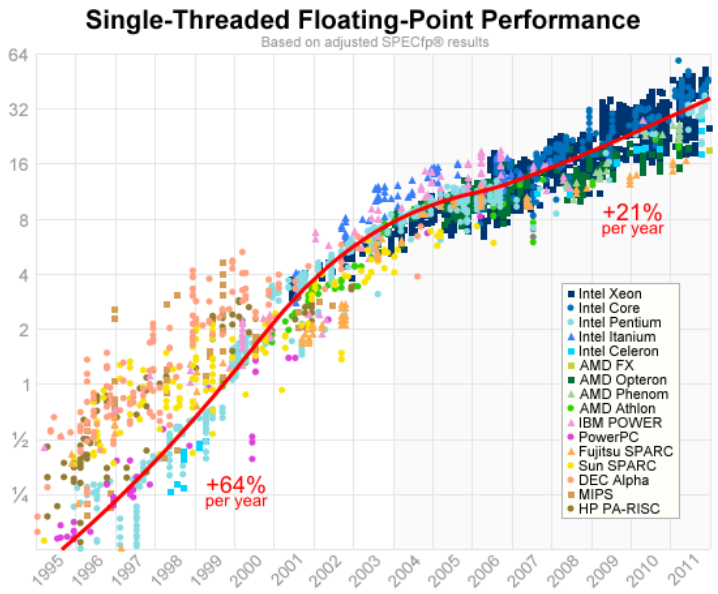


Fonte: http://preshing.com/20120208/a-look-back-at-single-threaded-cpu-performance/

CONSEQUÊNCIAS DA LEI DE MOORE...



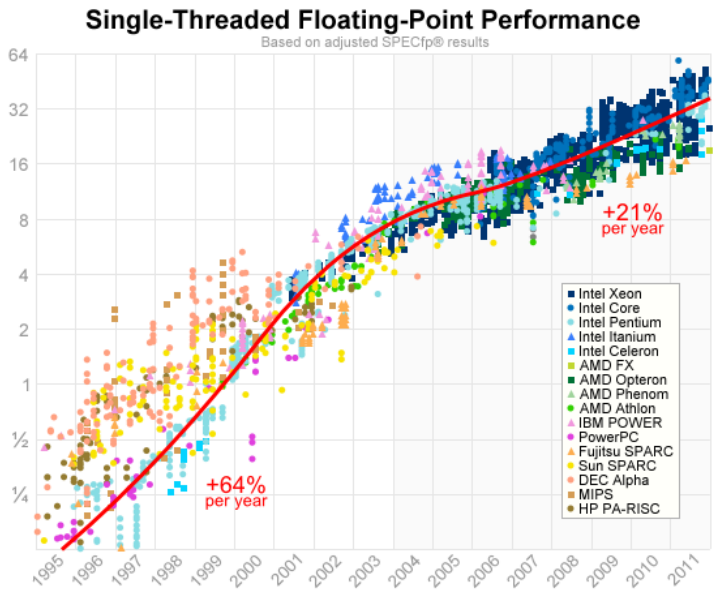
CONSEQUÊNCIAS DA LEI DE MOORE...



Treinamos pessoas dizendo que o desempenho viria do hardware.

Escreva seu programa como desejar e os Gênios-do-Hardware vamos cuidar do desempenho de forma "mágica".

CONSEQUÊNCIAS DA LEI DE MOORE...

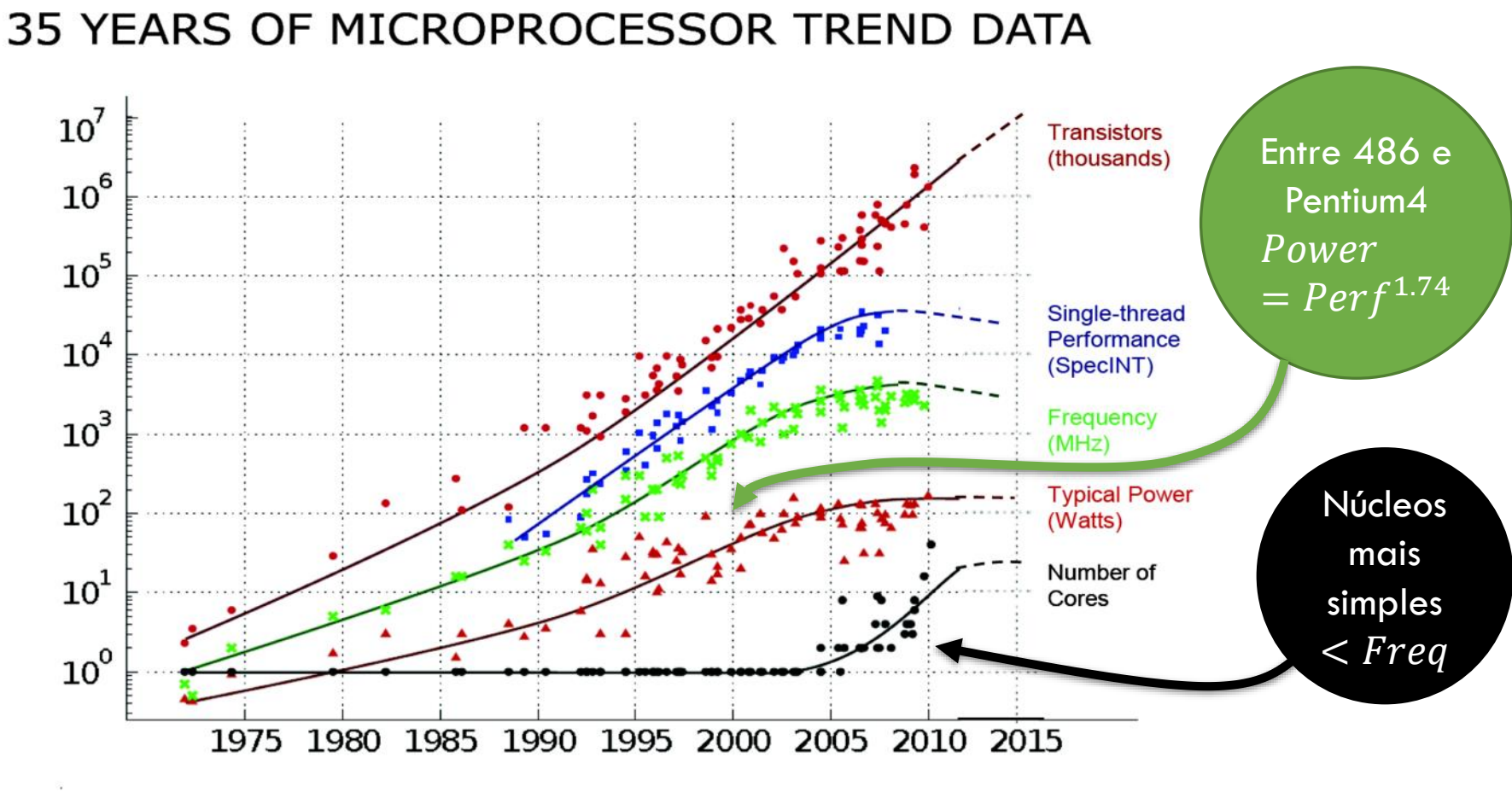


Treinamos pessoas dizendo que o desempenho viria do hardware.

Escreva seu programa como desejar e os Gênios-do-Hardware vamos cuidar do desempenho de forma "mágica".

O resultado: Gerações de engenheiros de software "ignorantes", usando linguagens ineficientes em termos de performance (como Java)... o que era OK, já que o desempenho é trabalho do Hardware.

... ARQUITETURA DE COMPUTADORES E O POWER WALL



. . . O RESULTADO

Um novo contrato **(the free lunch is over):**

...os **especialistas em HW** irão fazer o que é natural para eles (**muitos núcleos de processamento pequenos….**

...os **especialistas de SW** vão ter que se adaptar (**reescrever tudo**)

... O RESULTADO

Um novo contrato **(the free lunch is over):**

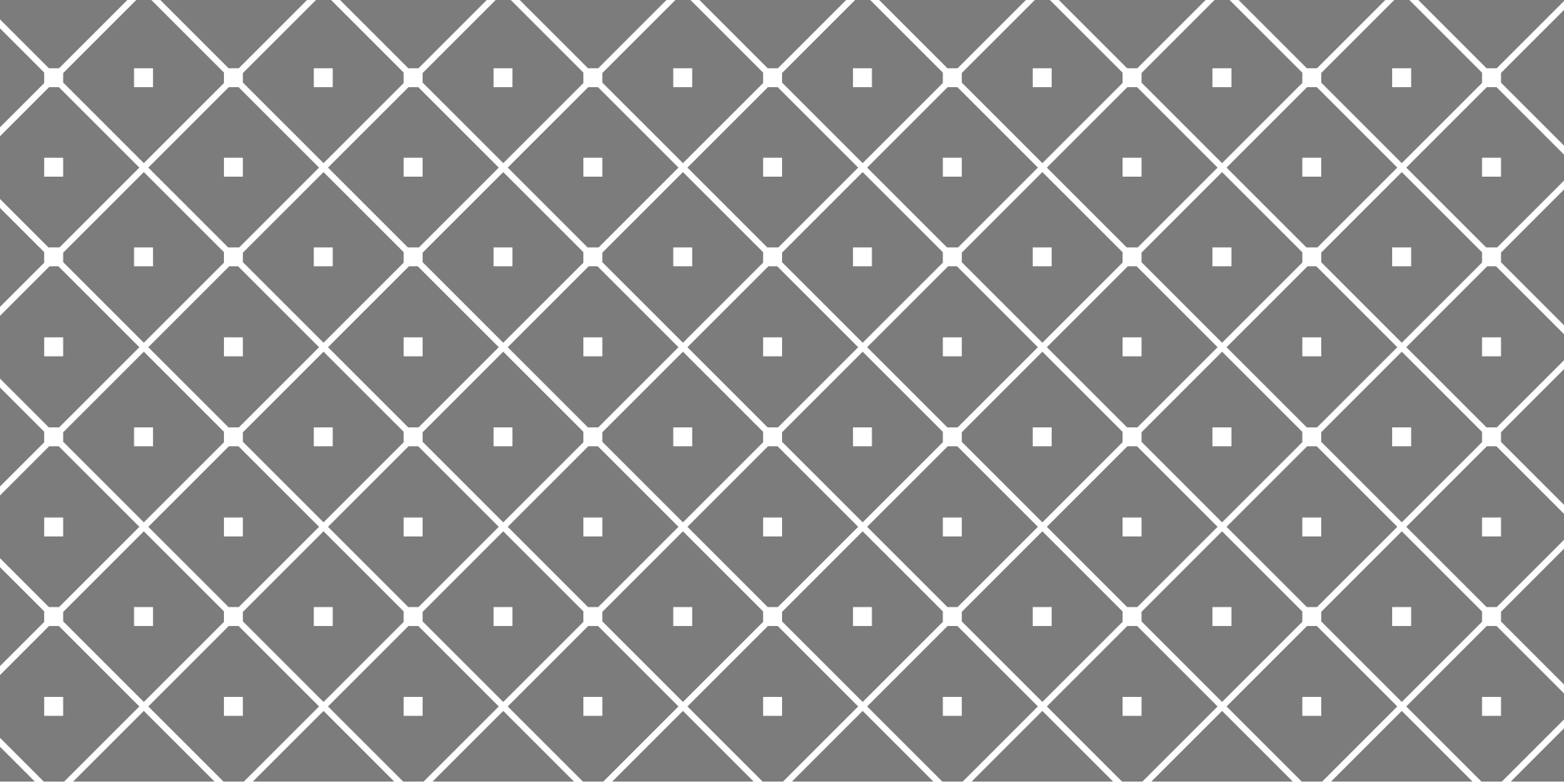
...os **especialistas em HW** irão fazer o que é natural para eles (**muitos núcleos de processamento pequenos**....

...os **especialistas de SW** vão ter que se adaptar (**reescrever tudo**)

O problema é que isso foi apresentado como um ultimato... ninguém perguntou se estaríamos OK com esse novo contrato...

...o que foi bastante rude.

Só nos resta colocar a mão na massa e usar programação paralela!



OPENMP

VISÃO GERAL OPENMP:

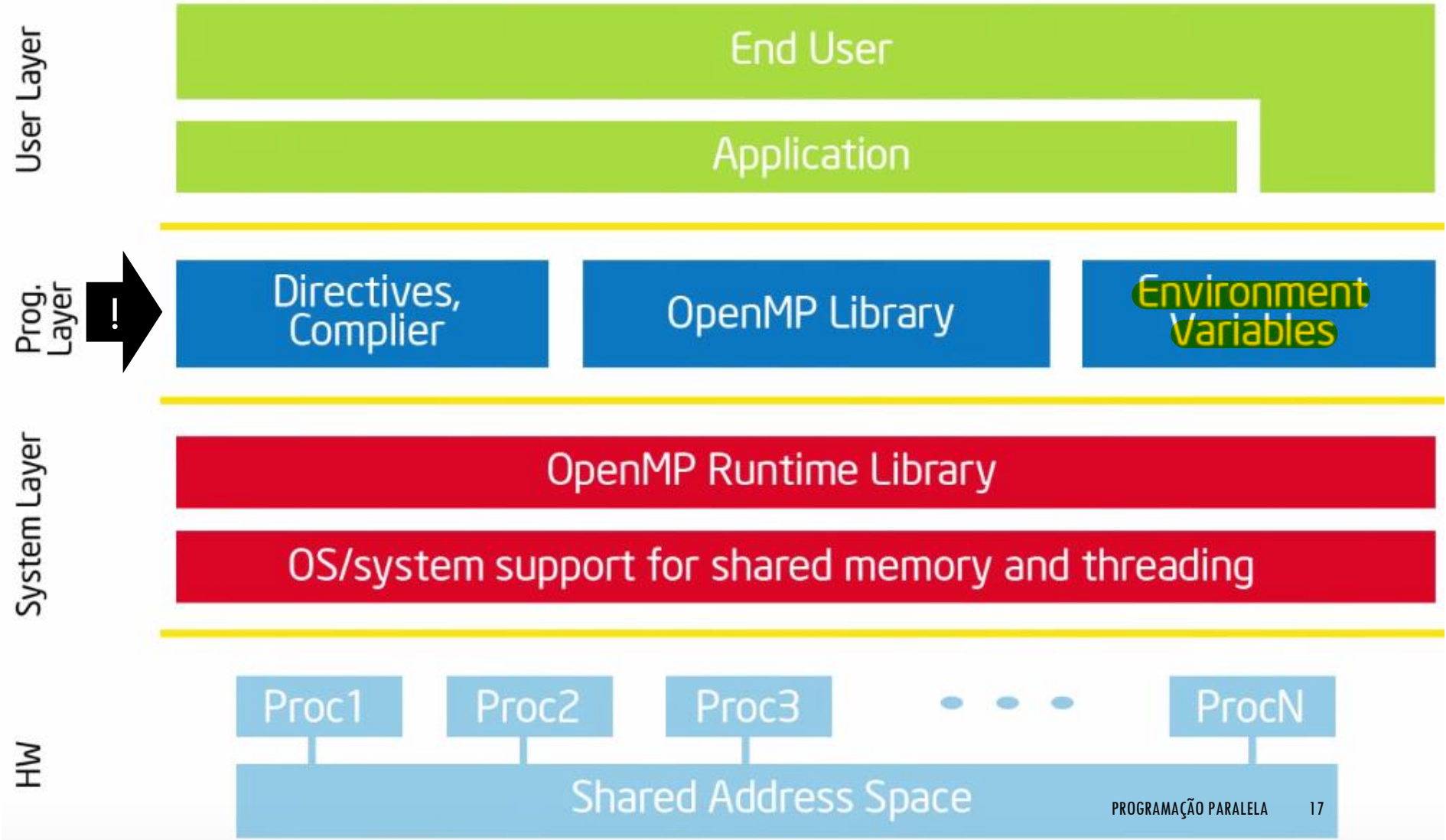
OpenMP: Uma API para escrever aplicações Multithreaded

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 20 anos de 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]...]
```

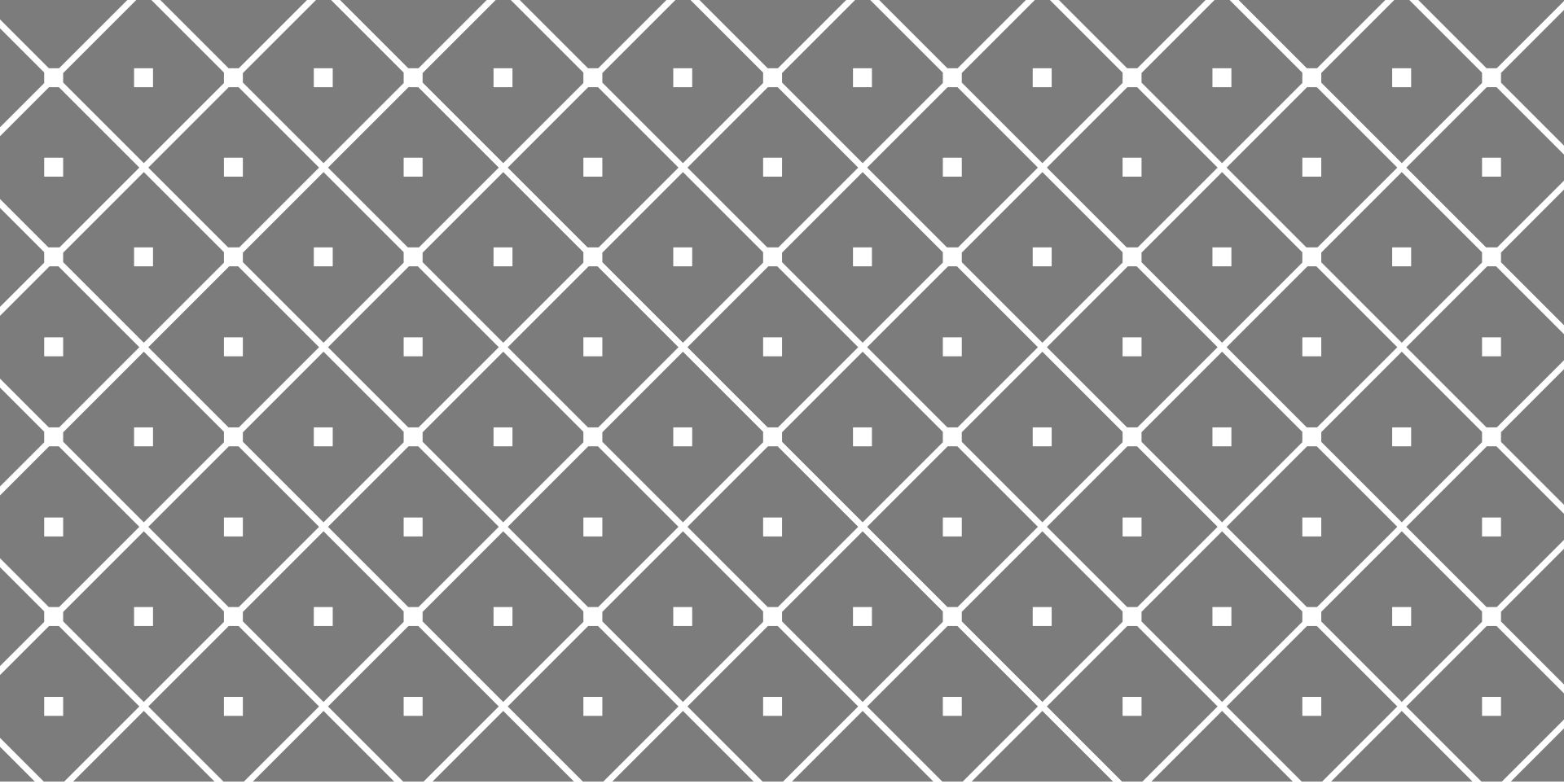
- Exemplo:

```
#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.



ALGUNS BITS CHATOS... USANDO O COMPILADOR OPENMP (HELLO WORLD)

NOTAS DE COMPILAÇÃO

Linux e OS X com gcc:

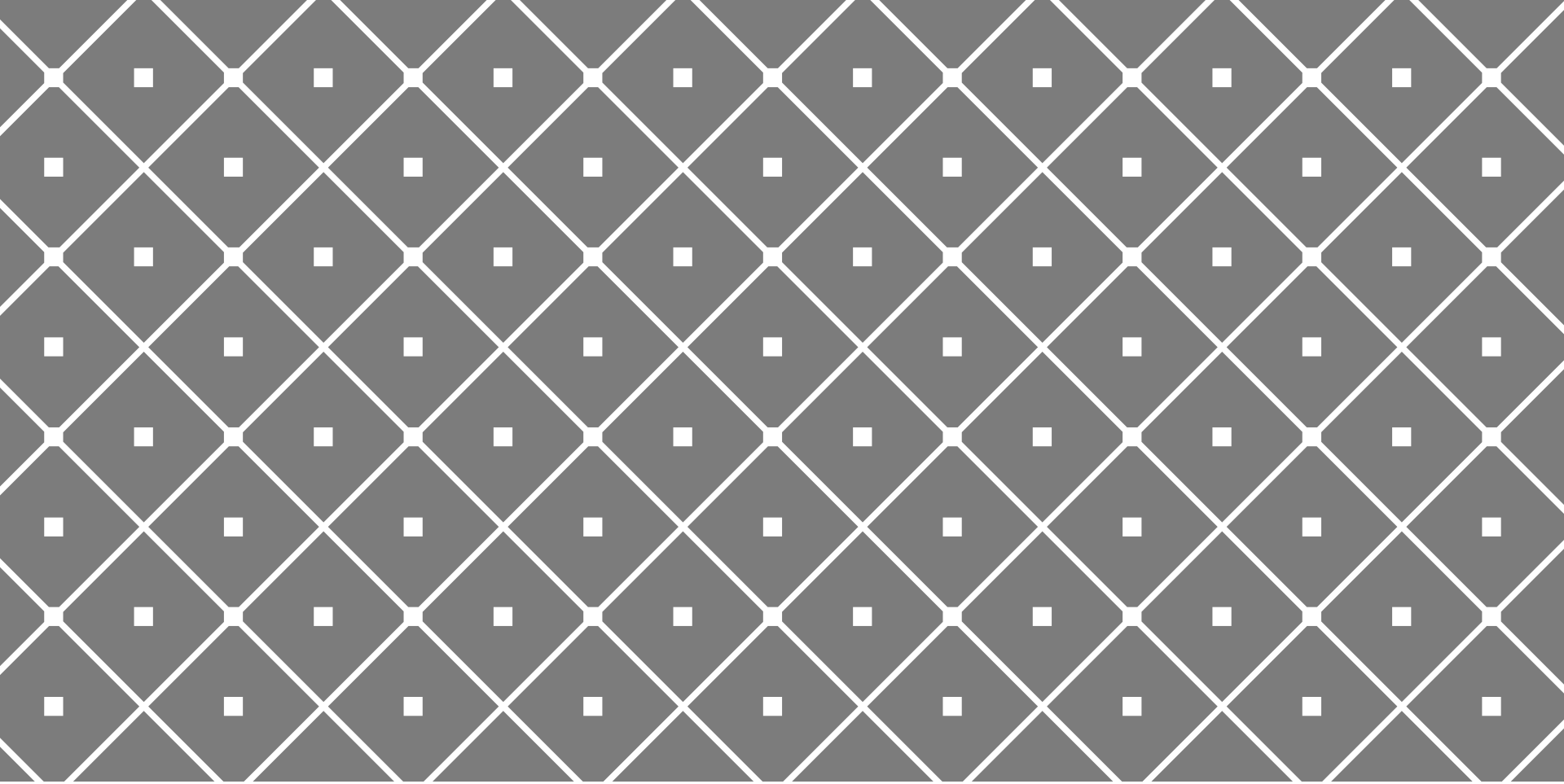
```
gcc -fopenmp foo.c
export OMP_NUM_THREADS=4
./a.out
```

Para shell Bash

Também funciona no Windows!

Até mesmo no VisualStudio!

Mas vamos usar Linux!



EXERCÍCIO

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);

}
```

```
gcc
```

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);
  }
}
```

```
gcc -fopenmp
```

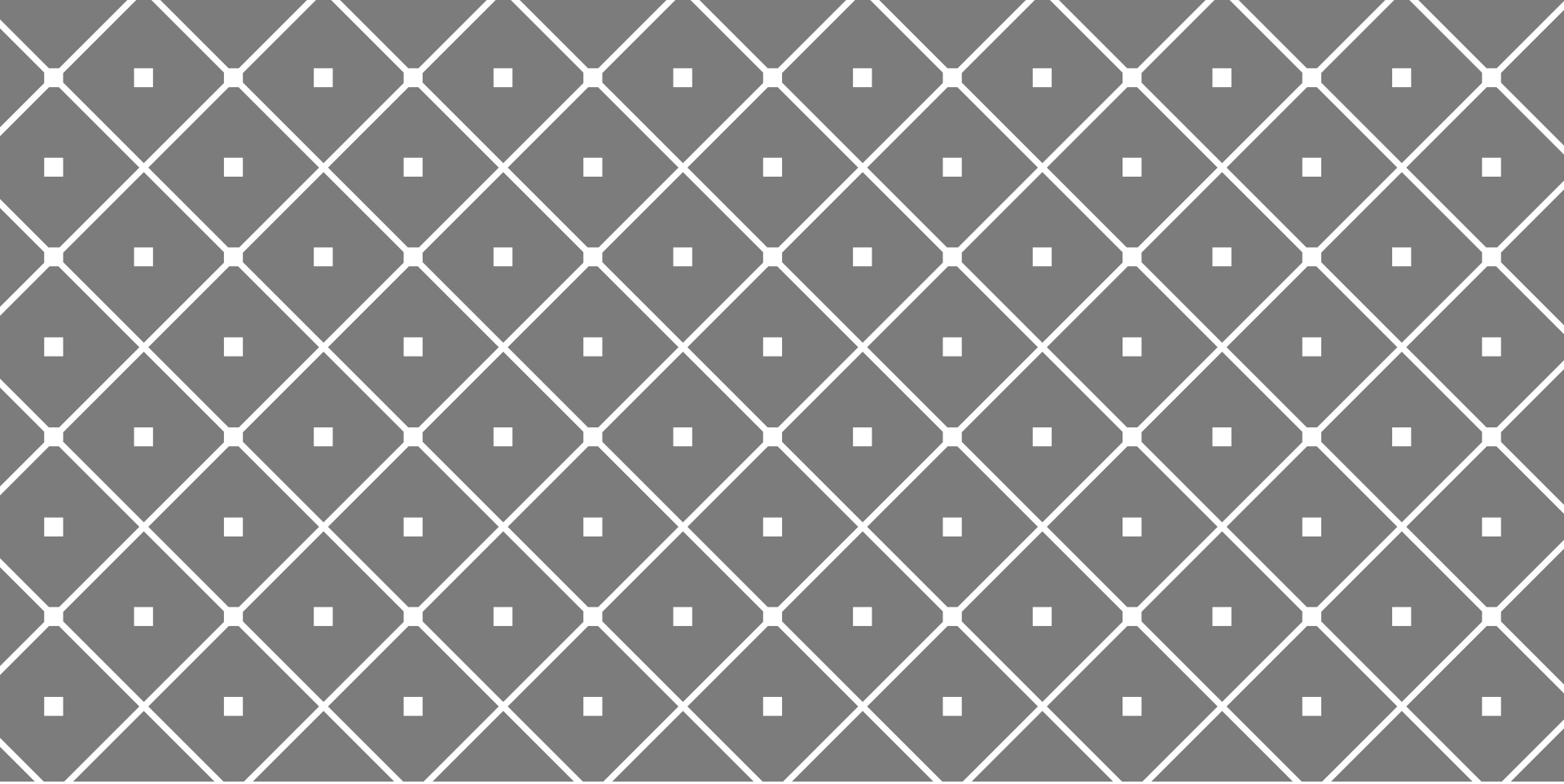
EXERCÍCIO 1, PARTE C: HELLO WORLD

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);
  }
}
```

```
gcc -fopenmp
```



HELLO WORLD E COMO AS THREADS FUNCIONAM

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.

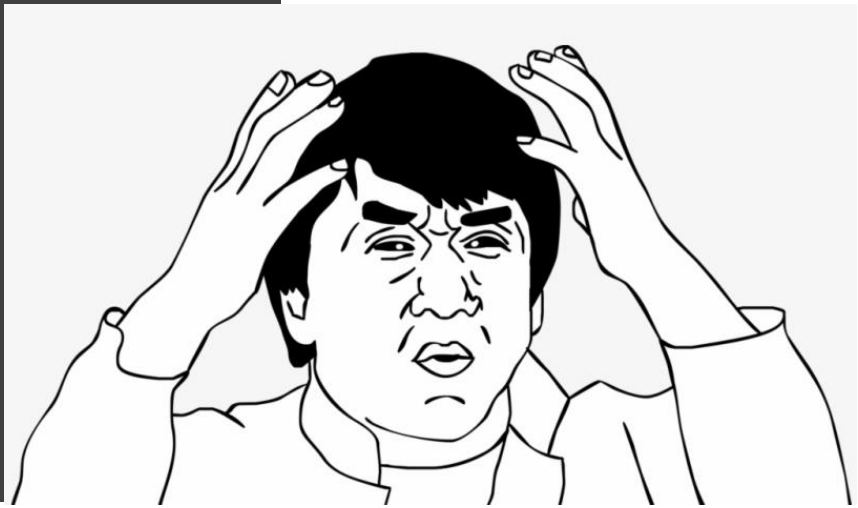
Fim da região paralela

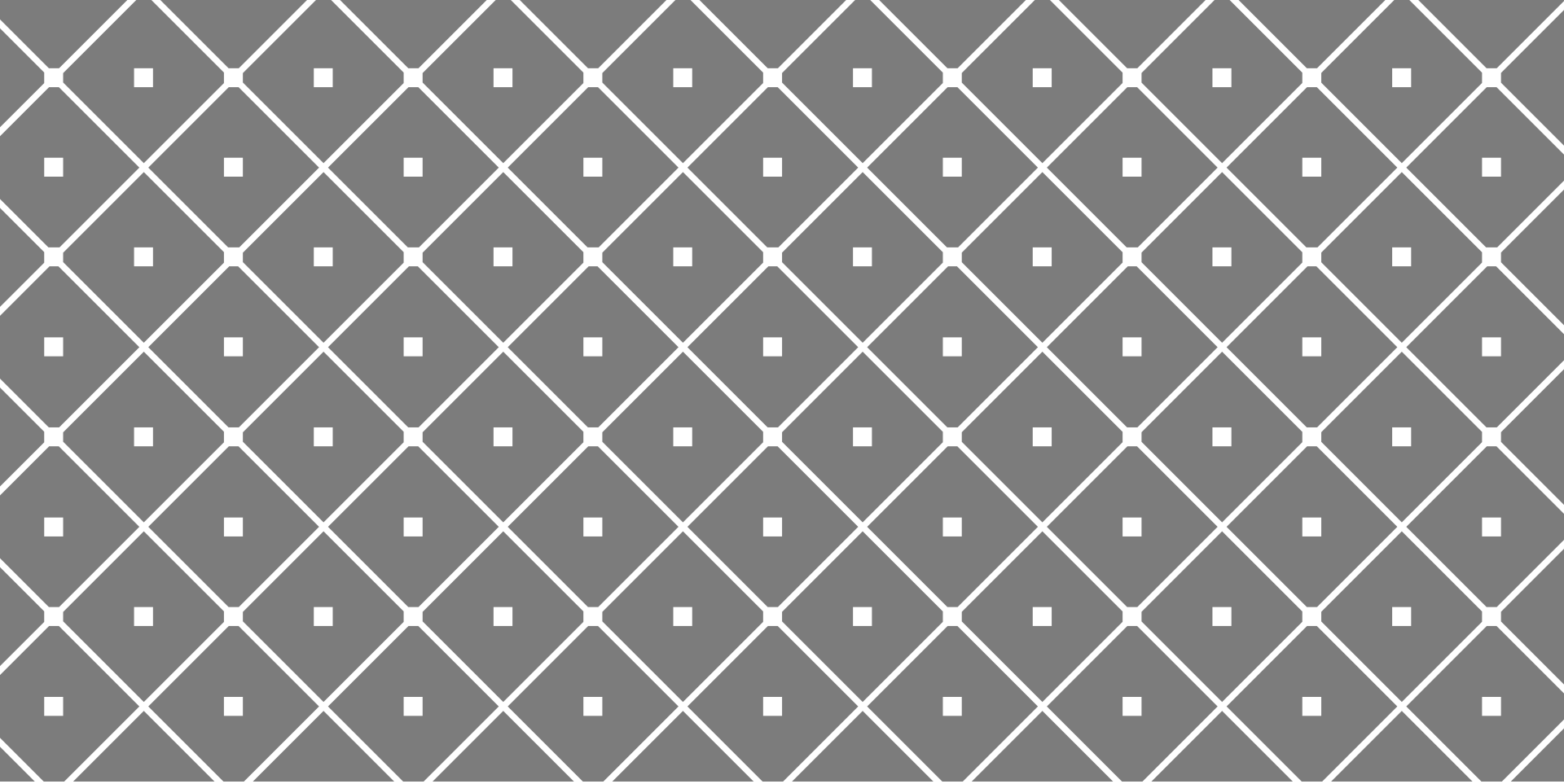
EXERCÍCIO 1: SOLUÇÃO

```
#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);
  }
}
```

Sample Output:

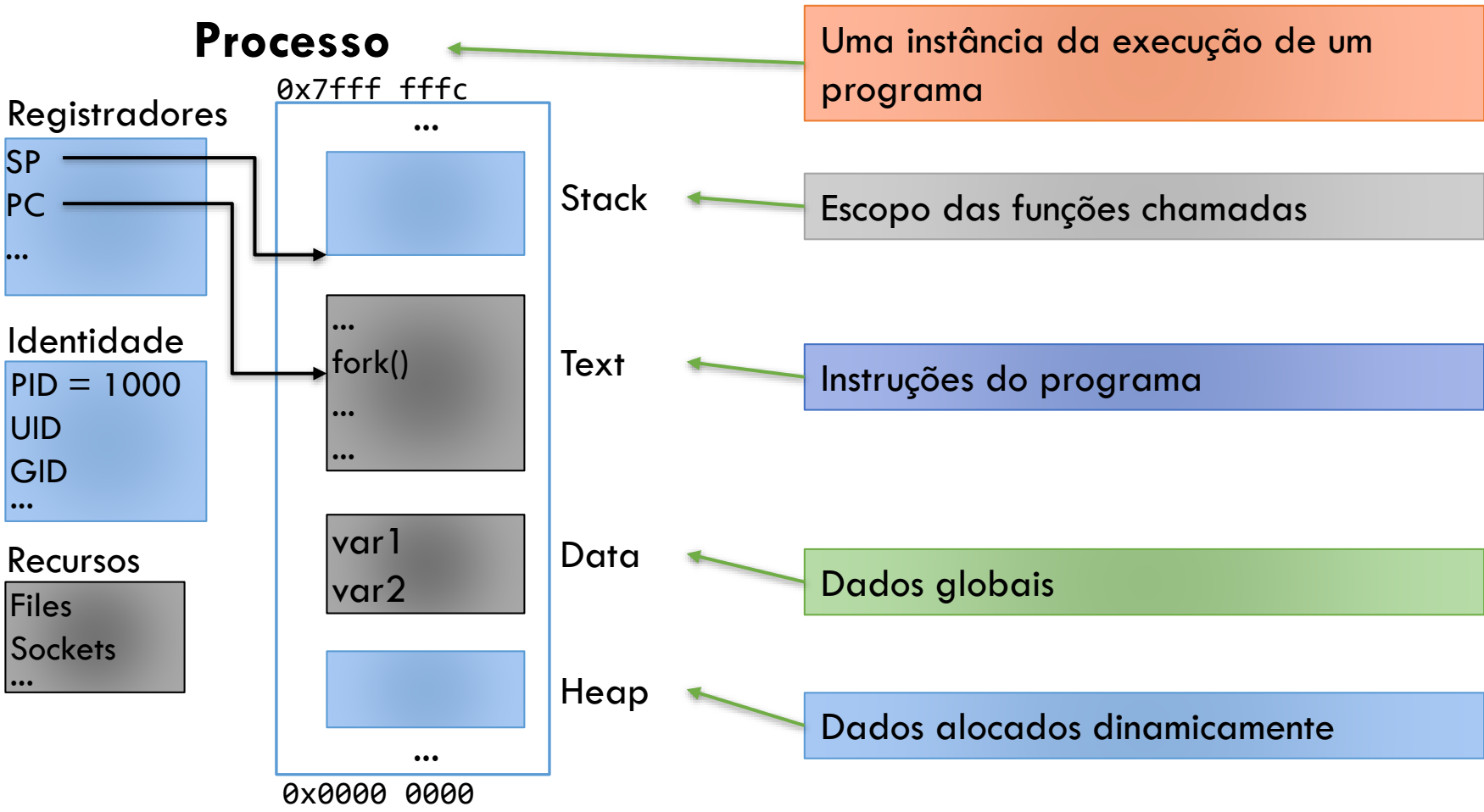
hello(1) hello(0) world(1)
world(0)
hello (3) hello(2) world(3)
world(2)



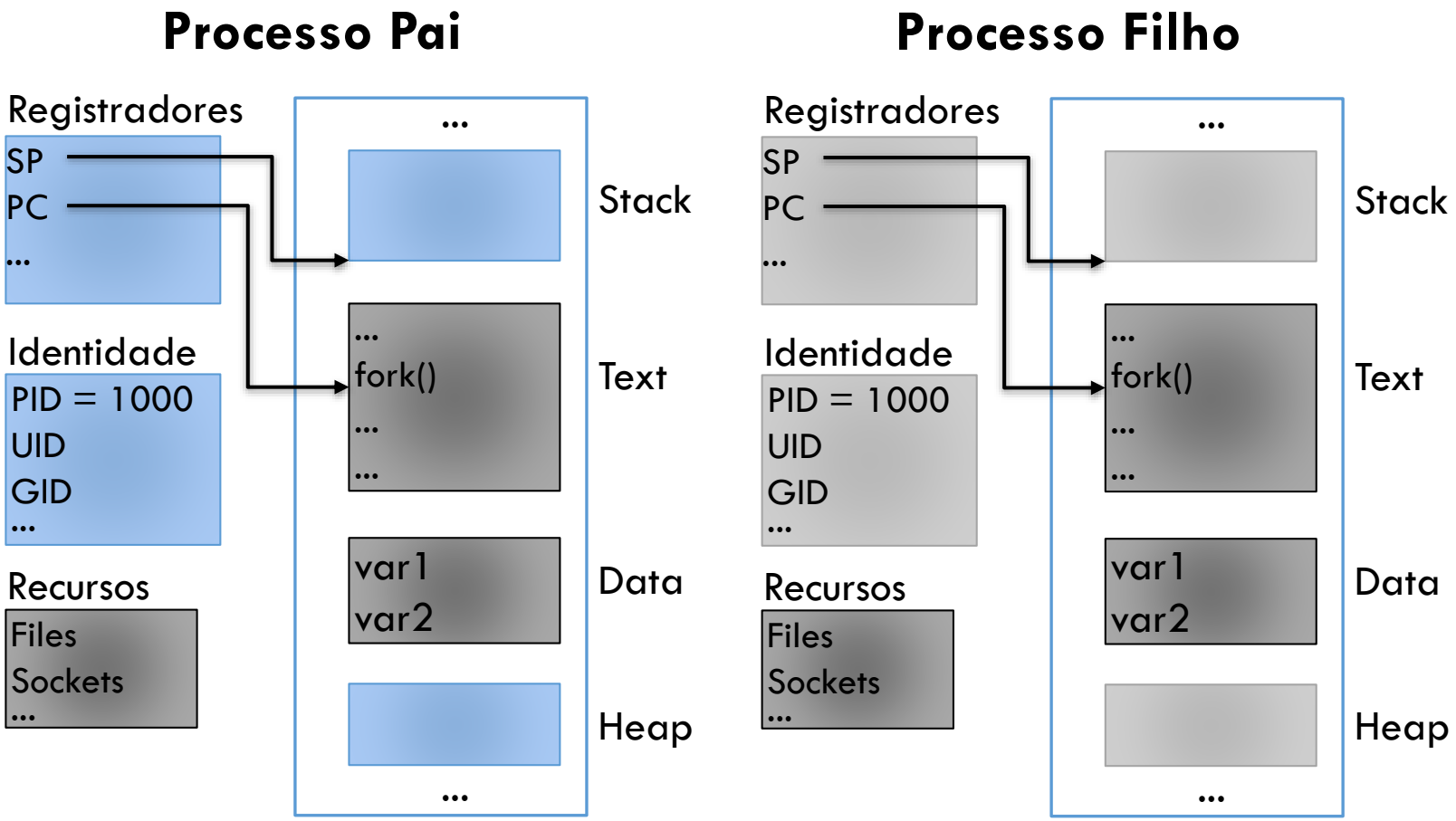


FUNCIONAMENTO DOS PROCESSOS VS. THREADS

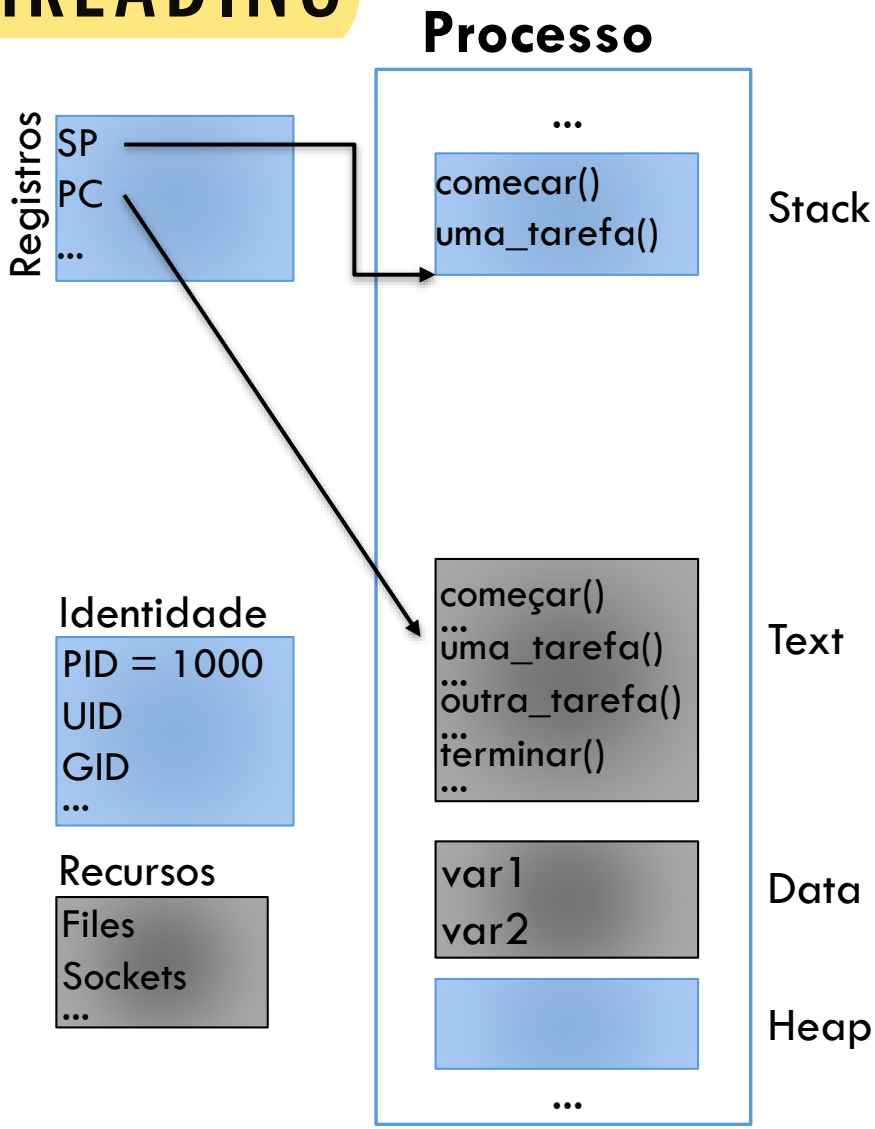
PROCESSOS


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

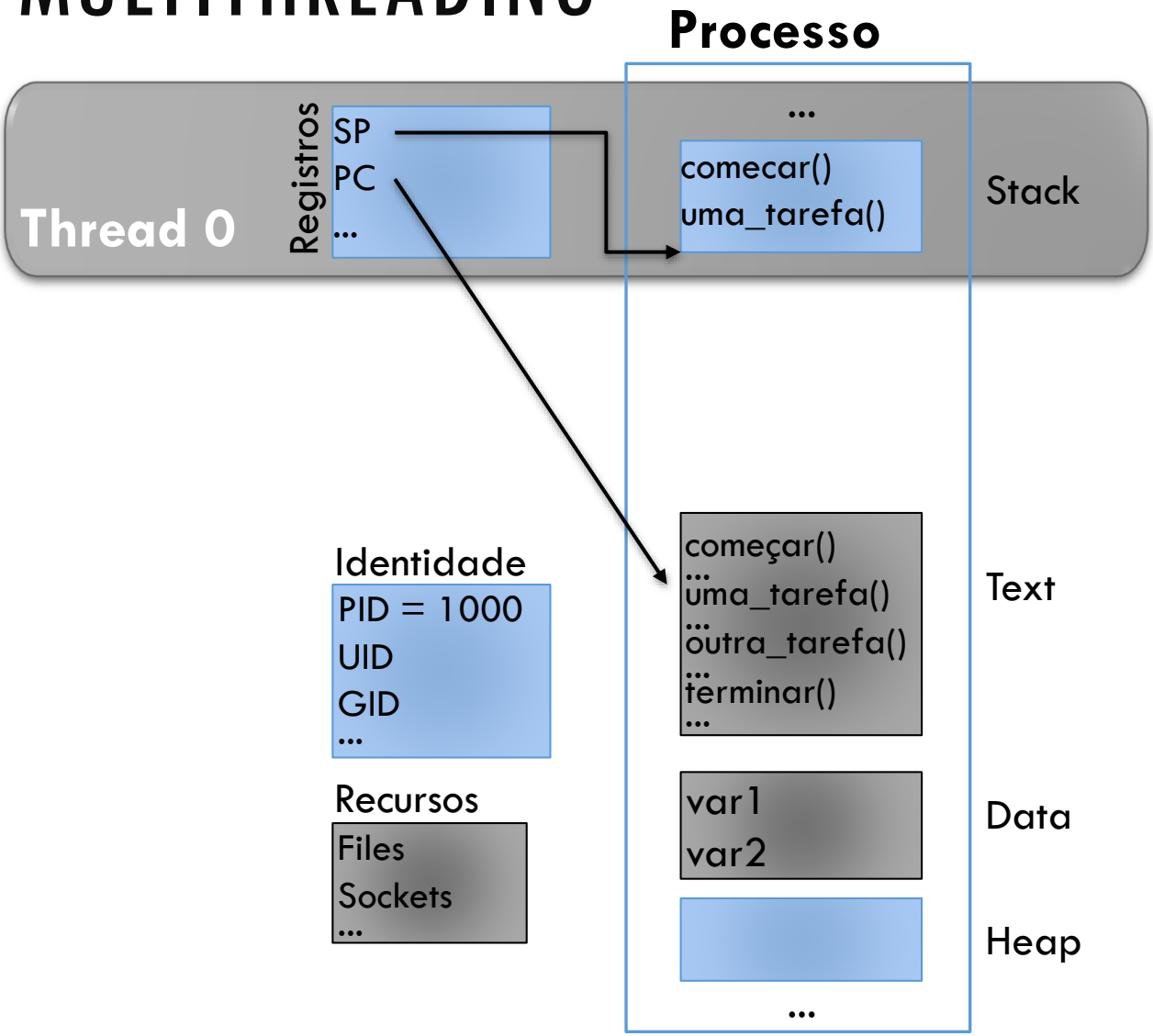
CONCORRÊNCIA COM PROCESSOS



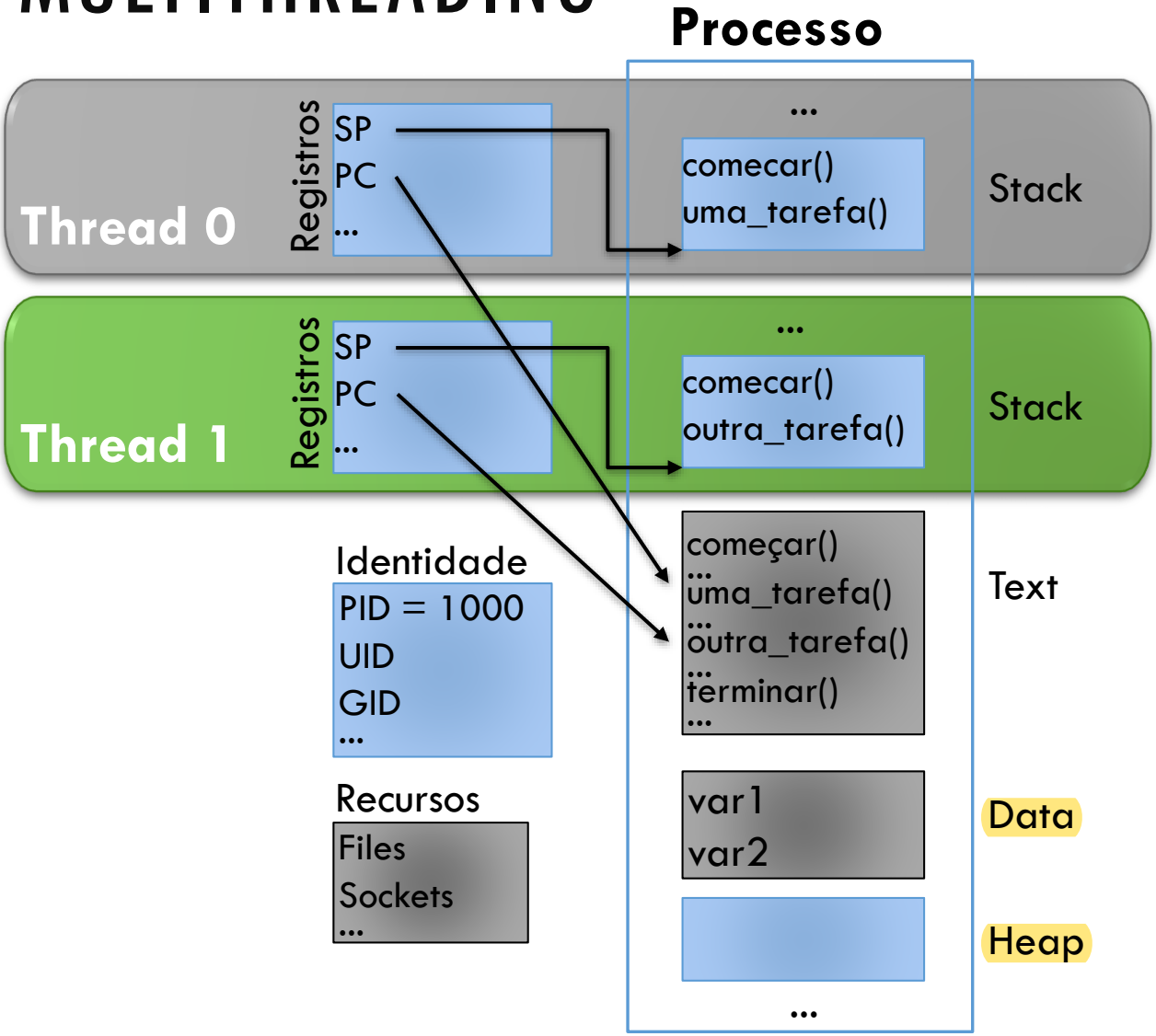
MULTITHREADING



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 na cache)

UM PROGRAMA DE MEMÓRIA COMPARTILHADA

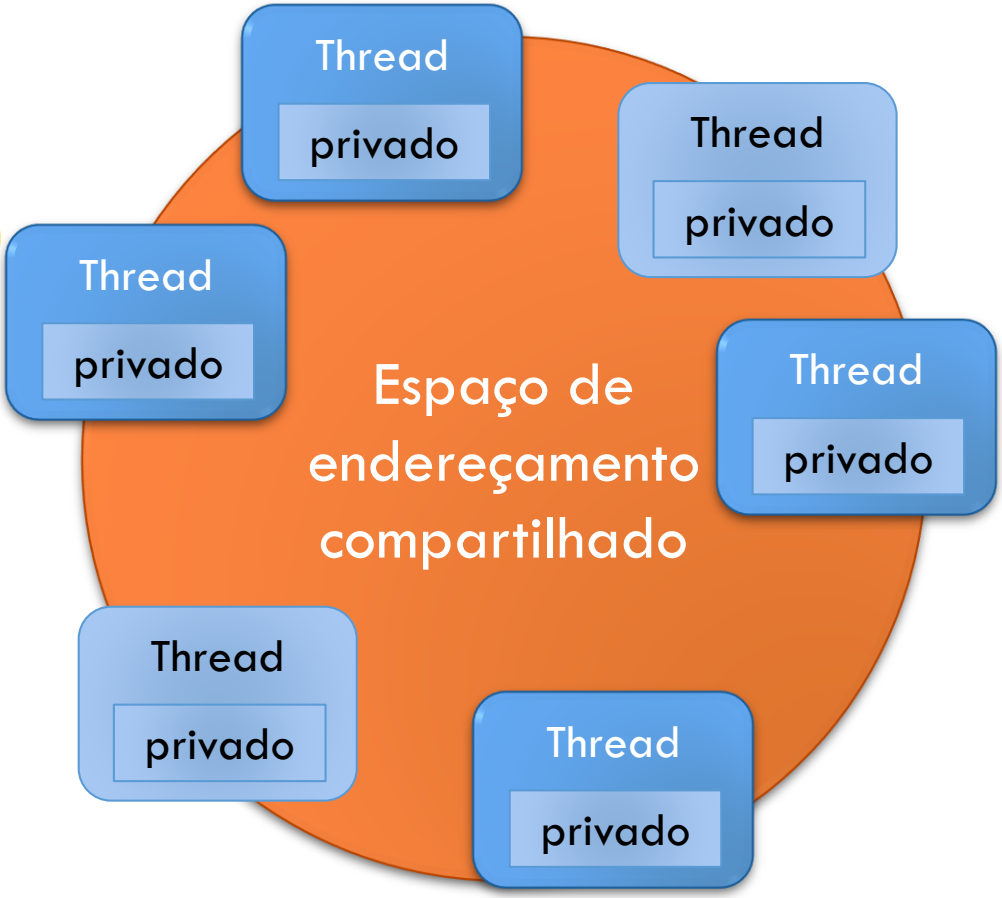
Uma instância do programa:

Um processo e muitas threads.

Threads interagem através de leituras/escrita com o espaço de endereçamento compartilhado.

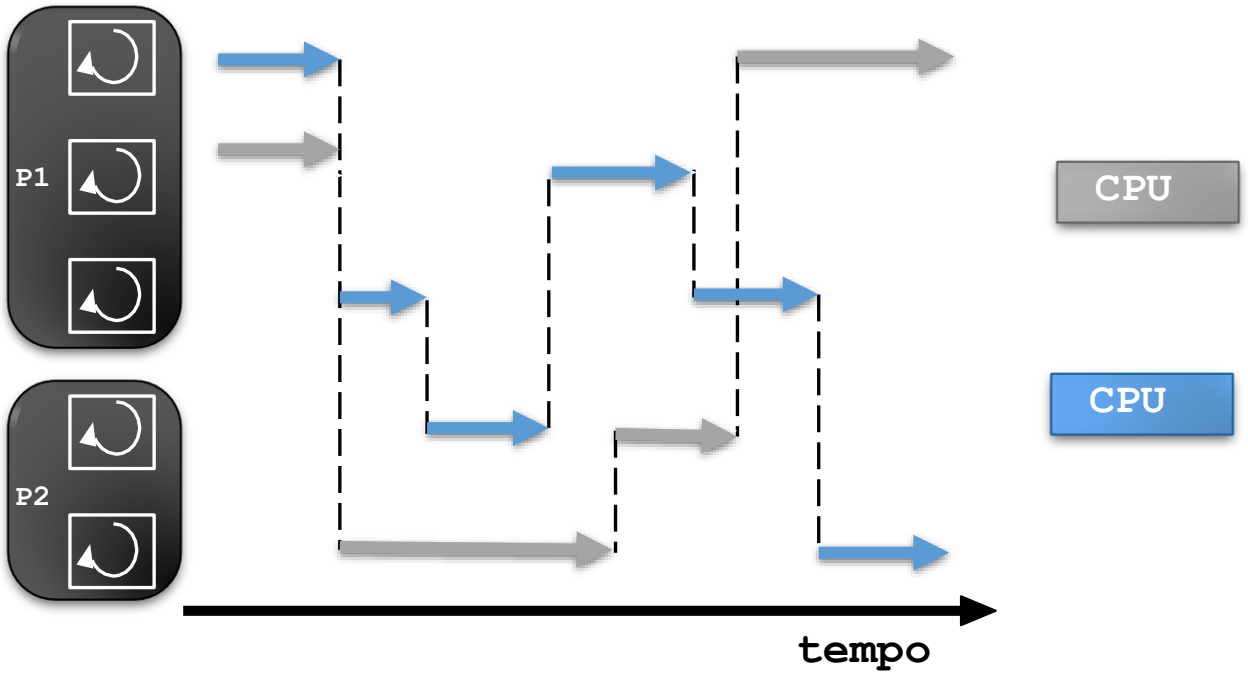
Escalonador SO decide quando executar cada thread (entrelaçado para ser justo).

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

Agora é possível entender esse comportamento!

```
#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);
  }
}
```

Sample Output:

hello(1) hello(0) world(1)
world(0)
hello (3) hello(2) world(3)
world(2)

VISÃO GERAL DE OPENMP: COMO AS THREADS INTERAGEM?

OpenMP é um modelo de multithreading de memória compartilhada.

- Threads se comunicam através de variáveis compartilhadas.

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).

VISÃO GERAL DE OPENMP: COMO AS THREADS INTERAGEM?

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.

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.

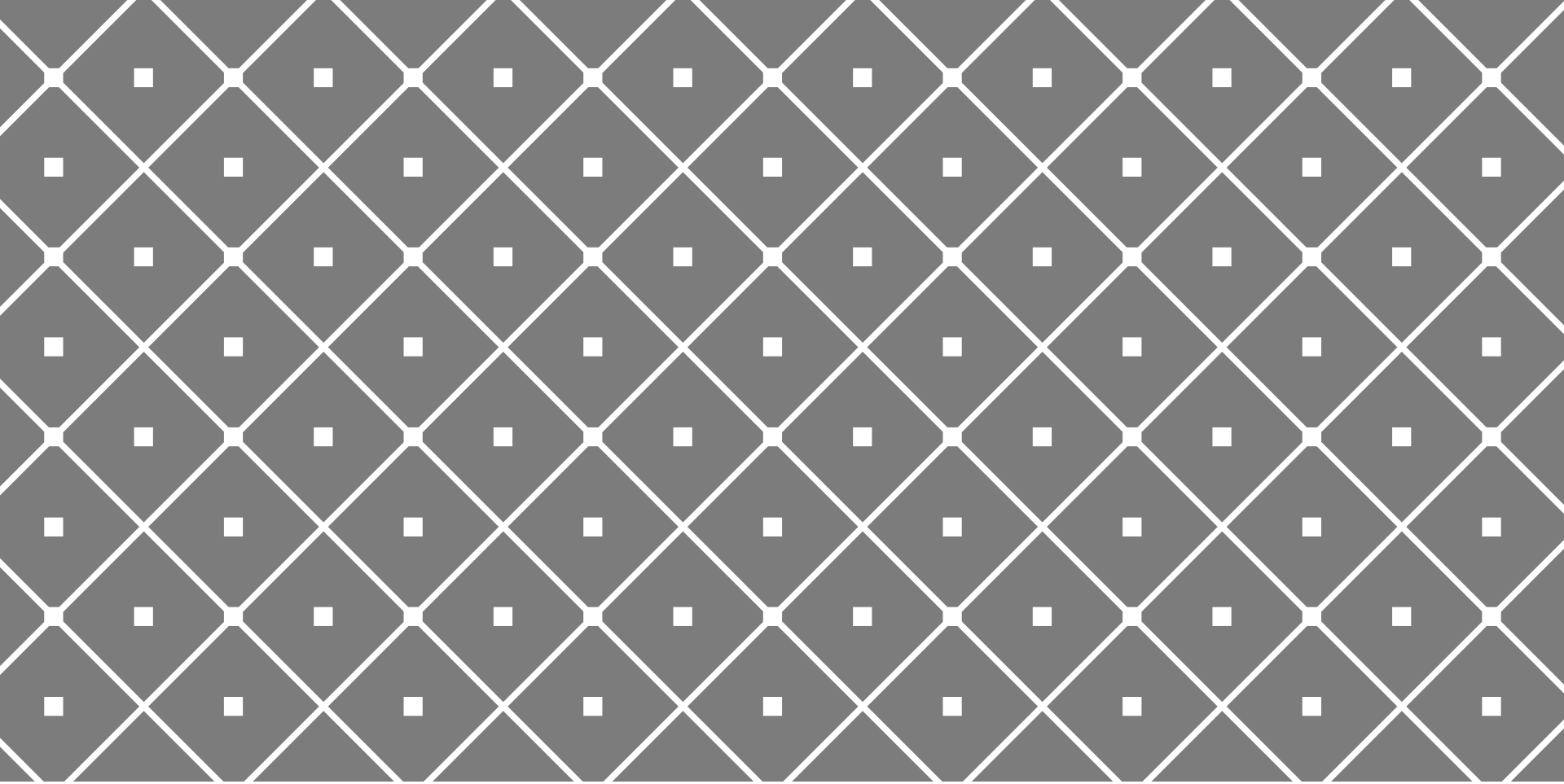
SINCRONIZAÇÃO E REGIÕES CRÍTICAS: EXEMPLO

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!

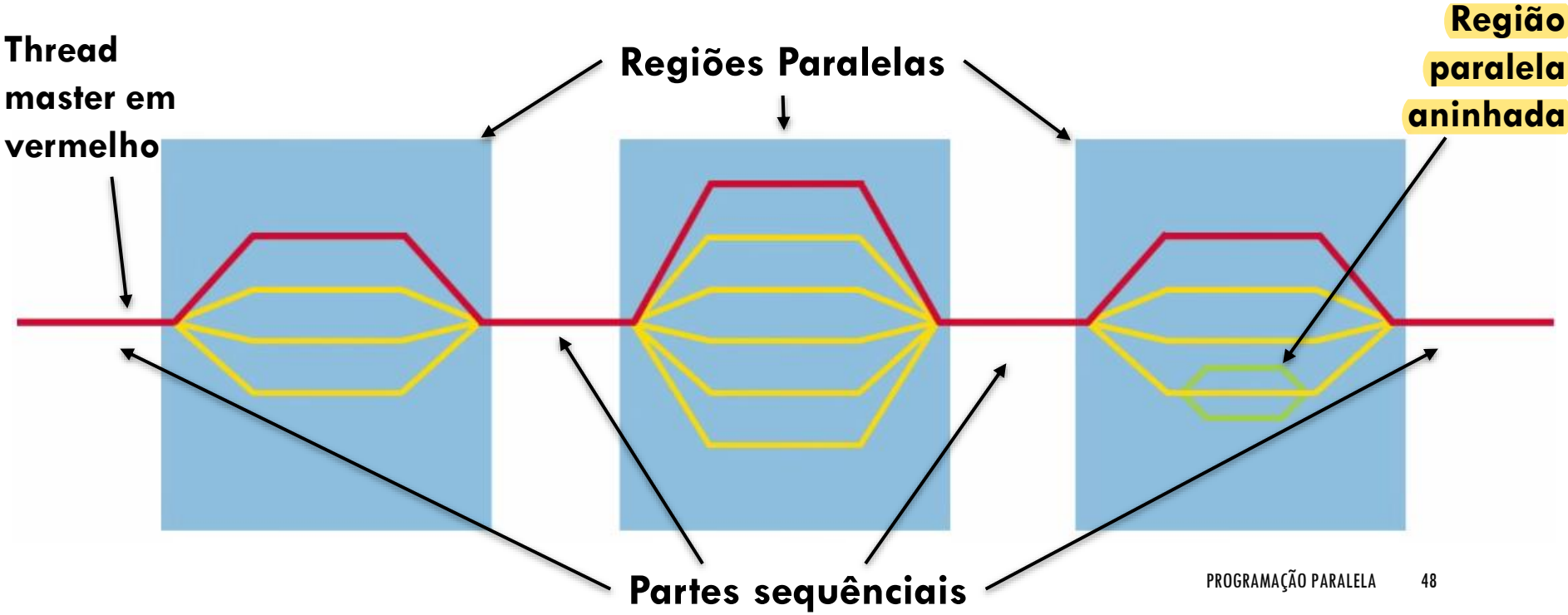


CRIANDO THREADS (O PROGRAMA PI)

MODELO DE PROGRAMAÇÃO OPENMP: PARALELISMO FORK-JOIN

A thread Master despeja um time de threads como necessário

Paralelismo é adicionado aos poucos até que o objetivo de desempenho é alcançado. Ou seja o programa sequencial evolui para um programa paralelo



CRIAÇÃO DE THREADS: REGIÕES PARALELAS

Criamos threads em OpenMP com construções **parallel.**

Por exemplo, para criar uma região paralela com 4 threads:

Cada thread chama pooh(ID,A) para os IDs = 0 até 3

```
double A[1000];
omp_set_num_threads(4);

#pragma omp parallel
{
 int ID = omp_get_thread_num();
 pooh(ID,A);
}
```

Função para requerer uma certa quantidade de threads

Cria threads em OpenMP

Função que retorna o ID de cada thread

Cada thread executa uma cópia do código dentro do bloco estruturado

CRIAÇÃO DE THREADS: REGIÕES PARALELAS

Criamos threads em OpenMP com construções **parallel.**

Por exemplo, para criar uma região paralela com 4 threads:

Cada thread chama pooh(ID,A) para os IDs = 0 até 3

```
double A[1000];


#pragma omp parallel num_threads(4)
{
 int ID = omp_get_thread_num();
 pooh(ID,A);
}
```

Um cópia única de A é **compartilhada** entre todas as threads

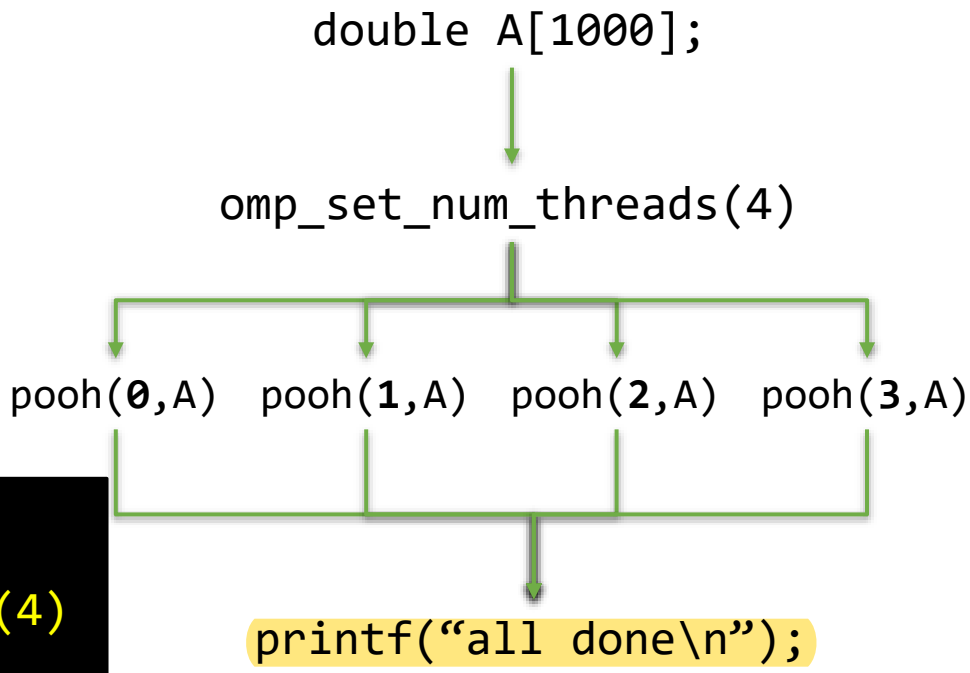
Forma reduzida

O inteiro ID é **privada** para cada thread

CRIAÇÃO DE THREADS: REGIÕES PARALELAS

Cada thread executa o mesmo código de forma redundante.

As threads esperam para que todas as demais terminem antes de prosseguir (i.e. uma barreira)




```
double A[1000];
#pragma omp parallel num_threads(4)
{
  int ID = omp_get_thread_num();
  pooh(ID, A);
}
printf("all done\n");
```

OPENMP: O QUE O COMPILADOR FAZ...

```
#pragma omp parallel num_threads(4)
{
 foobar ();
}
```

Tradução do Compilador

```
void thunk () {
 foobar ();
}
// Implementação Pthread
pthread_t tid[4];
for (int i = 1; i < 4; ++i)
 pthread_create(&tid[i],0,thunk,0);

thunk();
for (int i = 1; i < 4; ++i)
 pthread_join(tid[i]);
```

Todas as implementações de OpenMP conhecidas usam um pool de threads para que o custo de criação e destruição não ocorram para cada região paralela.

Apenas três threads serão criadas porque a última seção será invocada pela thread pai.

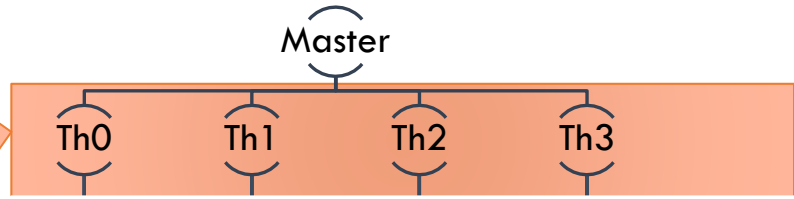
CRIAÇÃO DE THREADS: ERRO COMUM

Criamos threads em OpenMP com construções **parallel**.

Por exemplo, para criar uma região paralela com 4 threads.

Devemos ter cuidado ao criar regiões paralelas aninhadas.

```
double A[1000];
omp_set_num_threads(4);

#pragma omp parallel
{
  int ID = omp_get_thread_num();



}
```



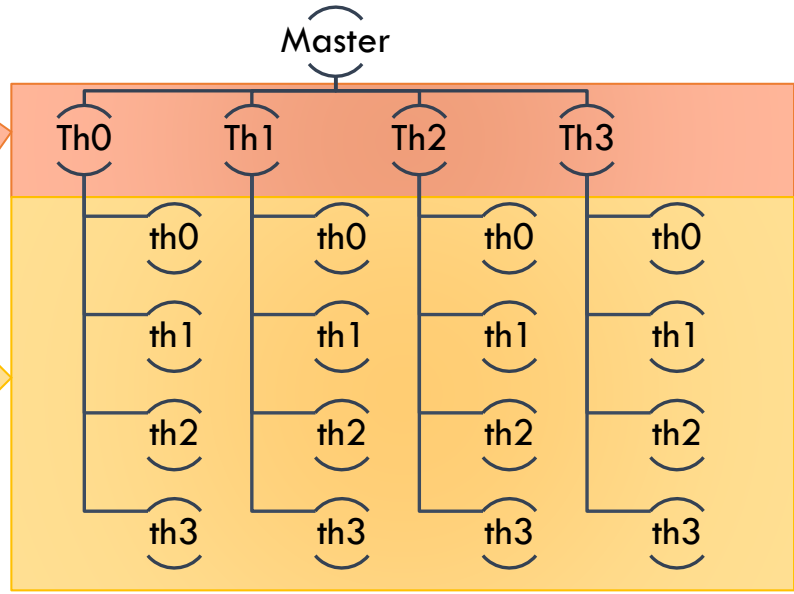
CRIAÇÃO DE THREADS: ERRO COMUM

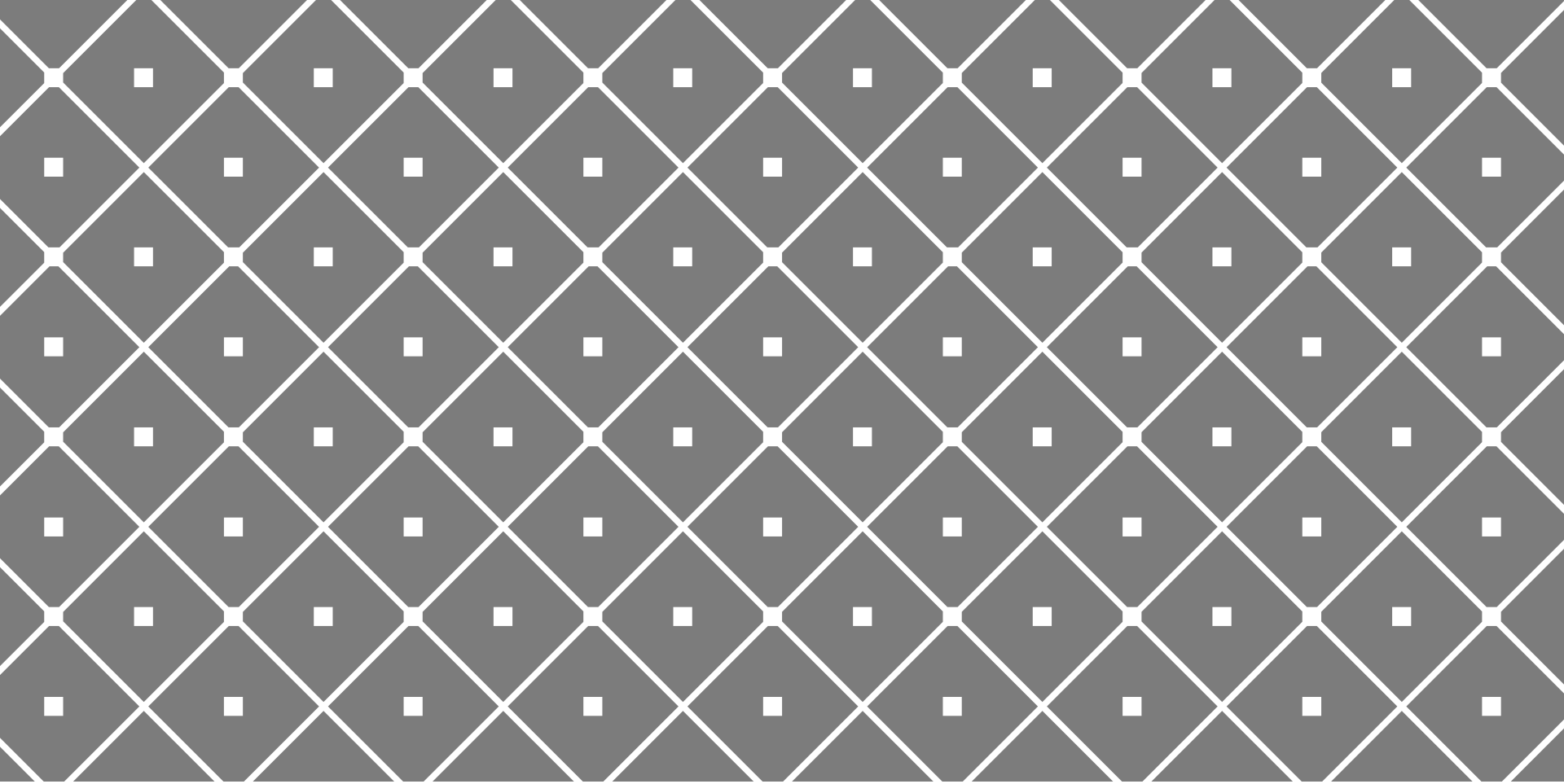
Criamos threads em OpenMP com construções **parallel.**

Por exemplo, para criar uma região paralela com 4 threads.

Devemos ter cuidado ao criar regiões paralelas aninhadas.

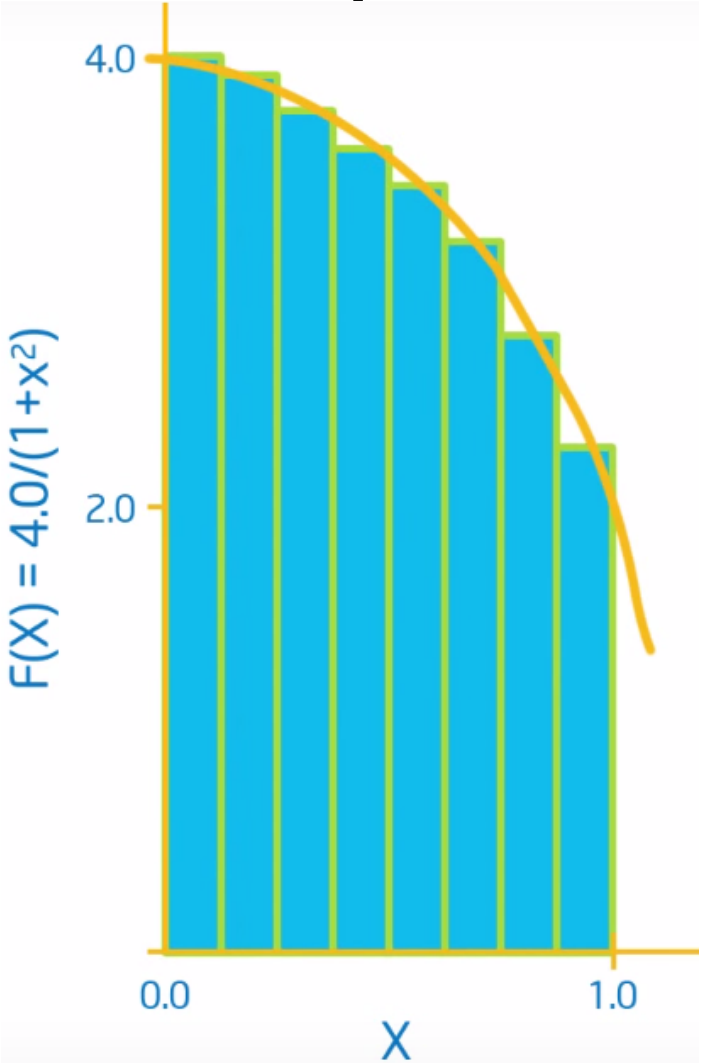
```
double A[1000];
omp_set_num_threads(4);

#pragma omp parallel
{
  int ID = omp_get_thread_num();
    #pragma omp parallel
    {
      pooh(ID,A);
    }
}
```





EXERCÍCIO

EXERCÍCIOS 2 A 4: INTEGRAÇÃO NUMÉRICA



Matematicamente, sabemos que:

$$\int_0^1 \frac{4.0}{1+x^2}\,dx = \pi$$

Podemos aproximar essa integral como a soma de retângulos:

$$\sum_{i=0}^{n} F(x_i)\Delta x \cong \pi$$

Onde cada retângulo tem largura Δx e altura $F(x_i)$ no meio do intervalo i.

EXERCÍCIOS 2 A 4: PROGRAMA PI SERIAL

```
static long num_steps = 100000;
double step;
int main () {
  int i; double x, pi, sum = 0.0;
  step = 1.0/(double) num_steps;
  for (i=0;i< num_steps; i++){
    x = (i + 0.5) * step; // Largura do retângulo
    sum = sum + 4.0 / (1.0 + x*x); // Sum += Área do retângulo
  }
  pi = step * sum;
}
```

EXERCÍCIO 2

Crie uma versão paralela do programa pi usando a construção paralela.

Atenção para variáveis globais vs. privadas.

Cuidado com condições de corrida!

Além da construção paralela, iremos precisar da biblioteca de rotinas.

```
int omp_get_num_threads(); // Número de threads no time

int omp_get_thread_num(); // ID da thread

double omp_get_wtime();  // Tempo em segundos desde um ponto
fixo no passado
```