UFRJ – IM - DCC

Sistemas Operacionais I

Unidade II - Threads

ORGANIZAÇÃO DA UNIDADE

- **Processos**
- ***Threads***
  - Conceituação
  - Descrição e Imagem de uma Thread
  - Tipos de thread
  - Modelos Multithread
  - Comunicação entre Threads
- **Concorrência**
- ***Deadlock* e *Starvation***

Conceituação

Unidade de execução em sistemas modernos

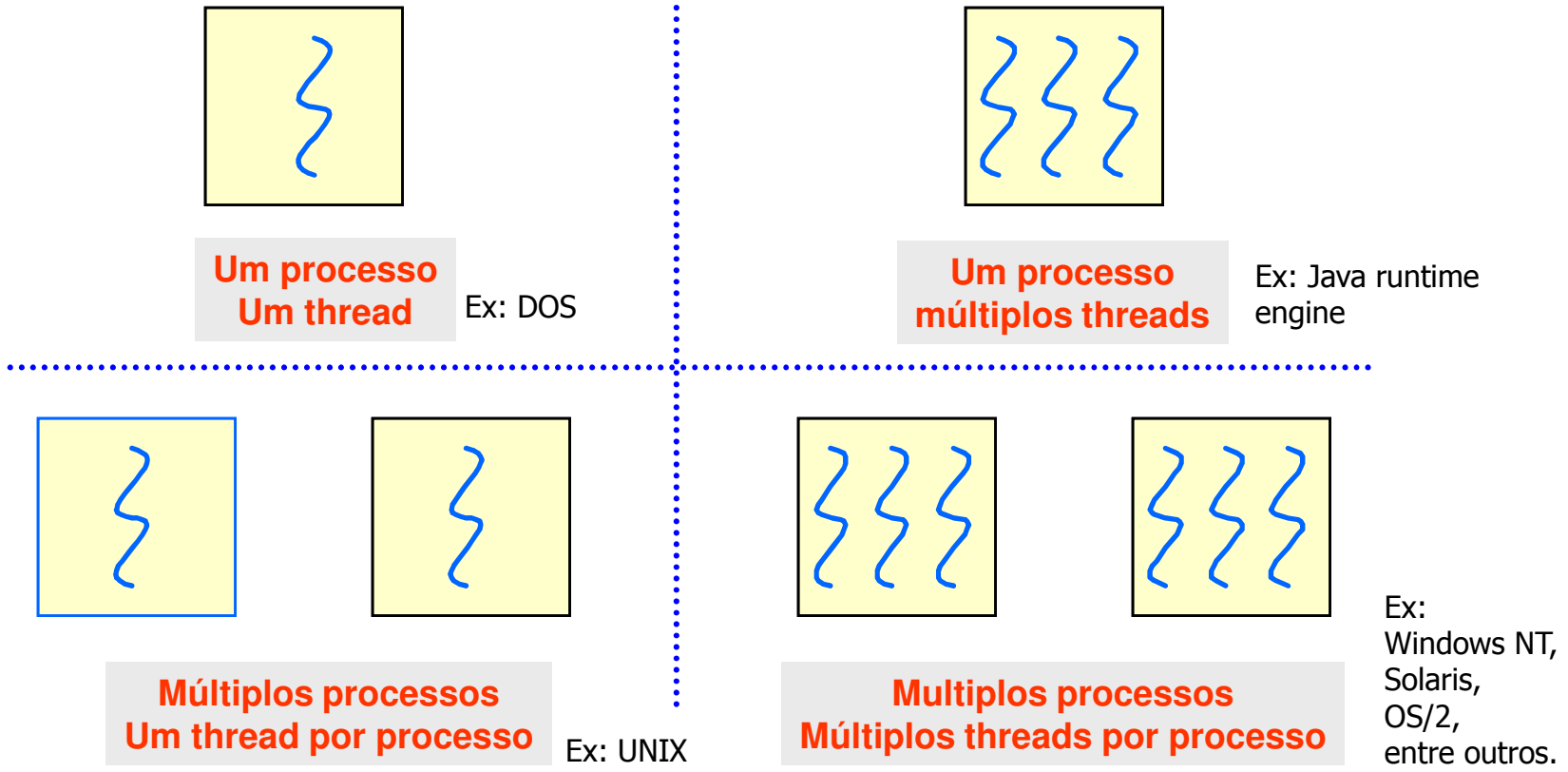
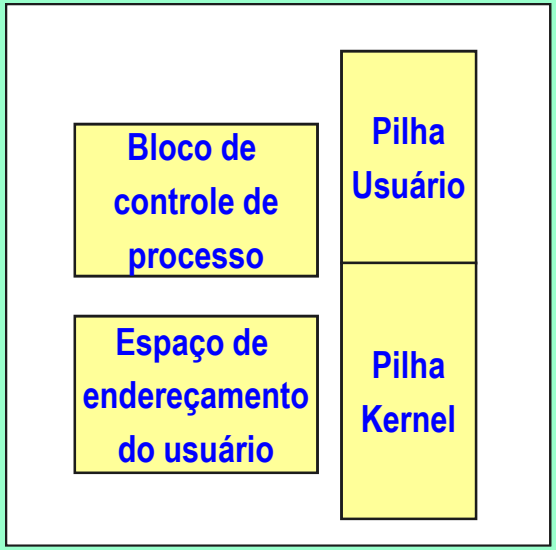
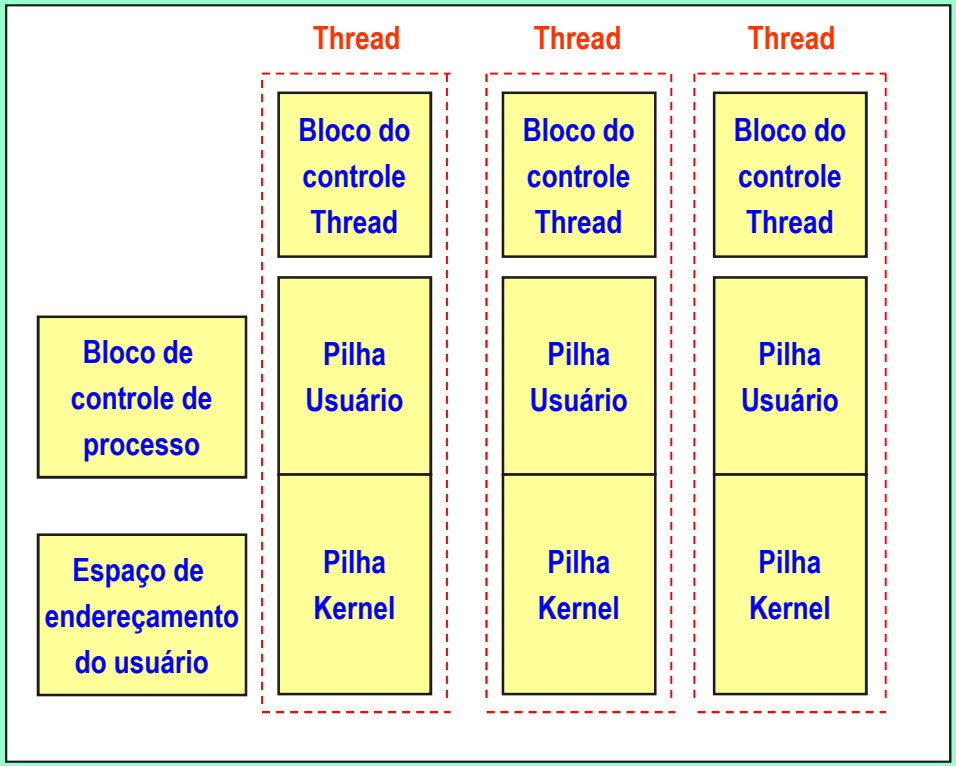


Imagem de um *Thread*

Modelo de Processo
(*únicoThread*)



Modelo
MultiThread



Sistemas *Multithread*

Unidade de Alocação
PROCESSO

Unidade de Execução
THREAD

Unidade de Proteção
PROCESSO

Benefícios

- ***Efetividade***
  - ✓ Permite que a execução do processo continue mesmo que alguns de seus Threads estejam bloqueados.
- ***Compartilhamento de Recursos***
  - ✓ Threads compartilham memória e outros recursos do processo
- ***Economia***
  - ✓ Threads são mais econômicos de serem criados e o custo da troca de contexto é menor
- ***Utilização de Múltiplos Processadores***
  - ✓ Cada Thread pode ser executado em paralelo em um processador distinto

Tipos de *Threads*

Usuário

O kernel não vê os *threads*

Kernel

Implementada pelo Kernel

Processo leve (LWP*)

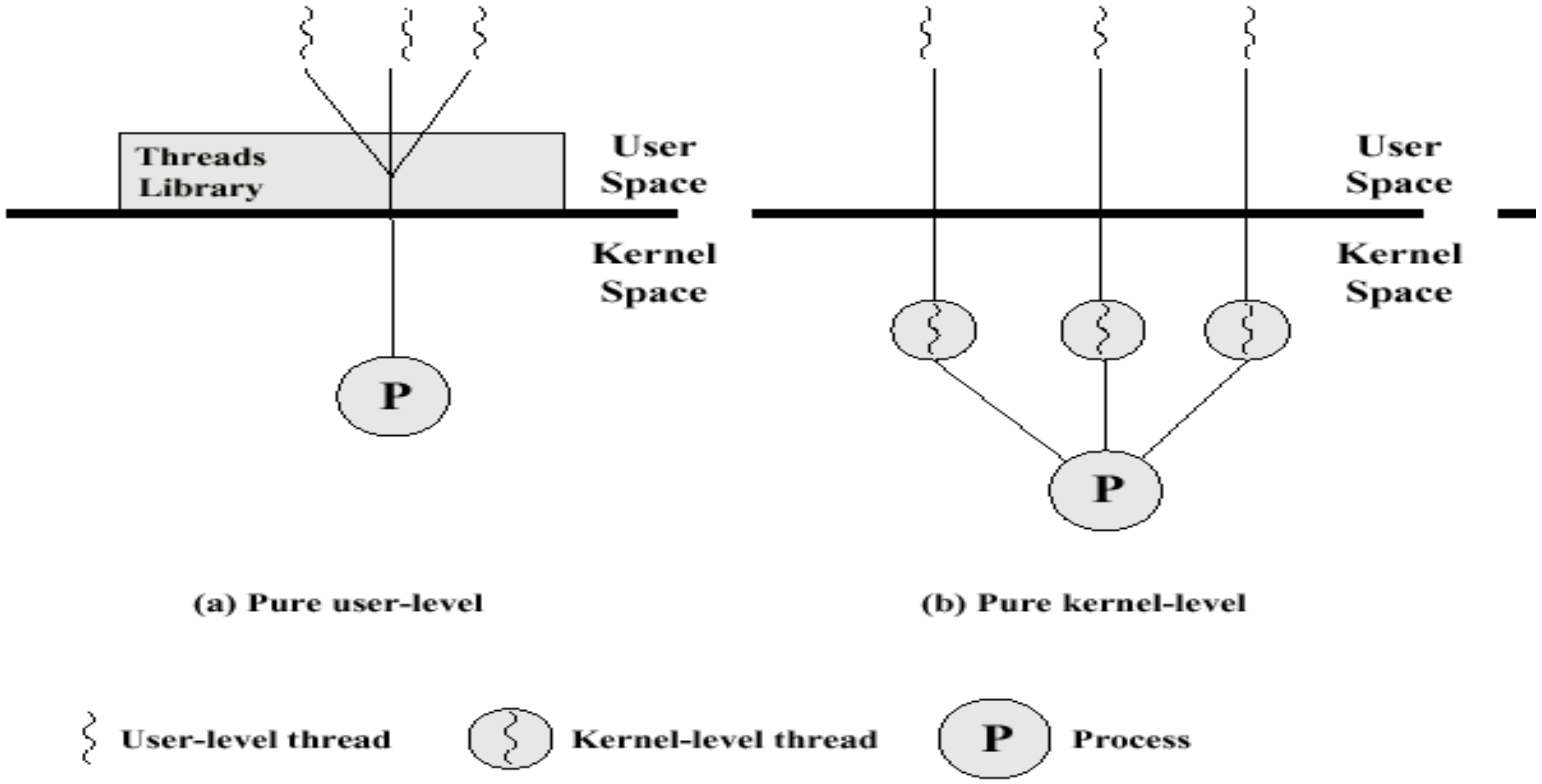
Uma combinação

(*) *Lightweight process*

Tipos de *threads*

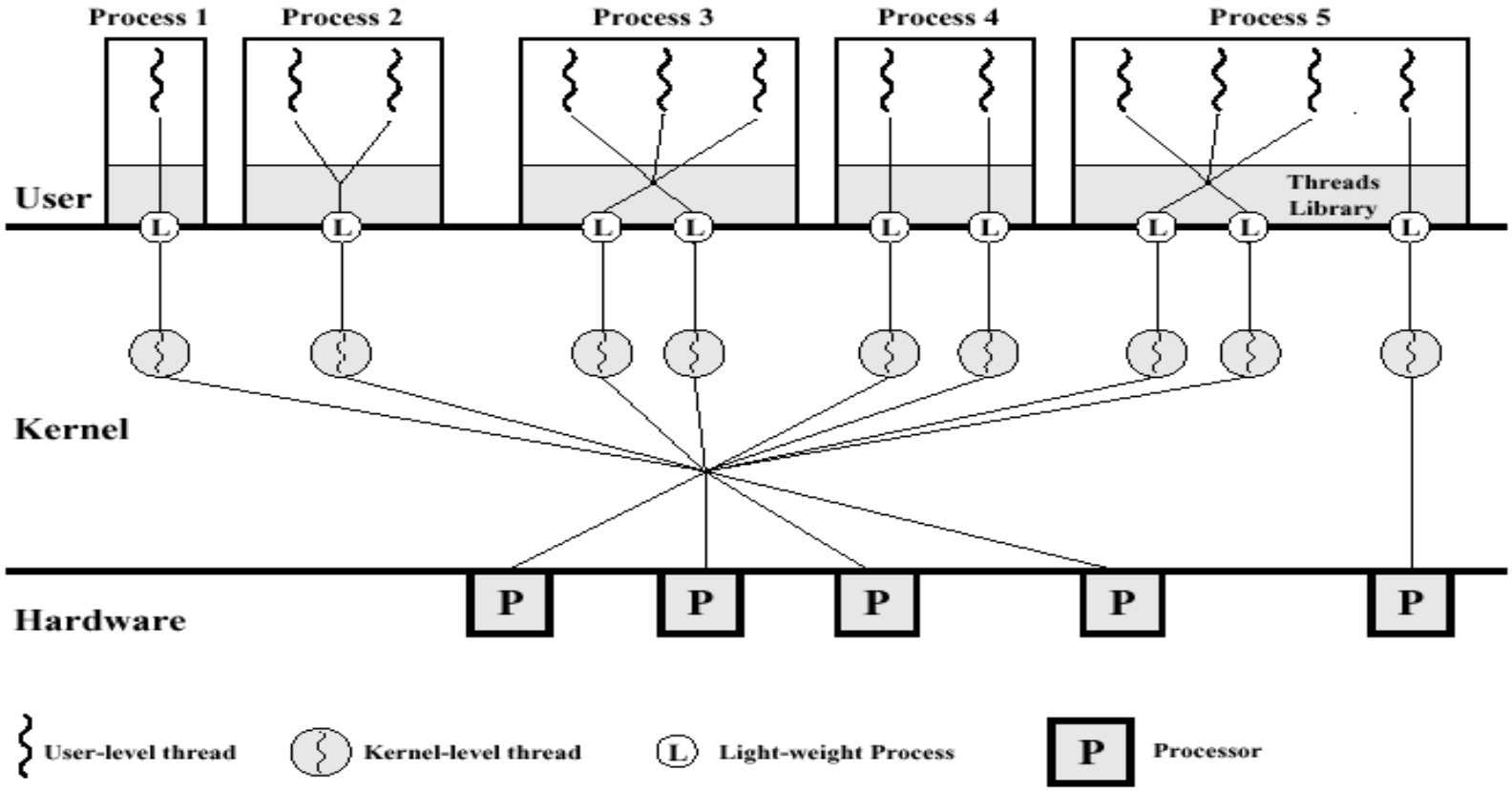
Threads de usuário

Threads de Kernel



Tipos de *Threads*

Processos Leves



Threads de usuário

- ***Suporte***
  - acima do nível de Kernel
  - Implementadas via user-level thread library (RTL*)
  - Criação, escalonamento e suporte via biblioteca
  - Rápida de ser criada e gerenciada
  - ***Não podem fazer uso de mais de um processador***
  - ***Chamadas ao sistema podem bloquear o processo***

- ***Sistemas que suportam user threads***
  - POSIX *Pthreads*
  - Mach *C-threads*
  - Solaris 2 UI-*threads* (UI – Unix International)

(*) *Run-time Library*

Threads de kernel

- ***Suporte***
  - O Kernel cria, escalona e administra os threads
  - Criação e administração mais demorados que as ULTs
  - Podem ser escalonadas para rodar em diferentes processadores
  - Bloqueio de um thread não bloqueia o processo
- ***Sistemas que Suportam kernel threads***
  - Windows 95/98/NT/2000/XP
  - Solaris
  - Linux

LWP (processos leves)

- **Criação de *threads* é feita em modo usuário**
- **A maior parte do escalonamento e sincronização acontece em modo usuário**
- **Os *threads* em modo usuário são mapeados num número possivelmente menor de *threads* do *kernel***

Modelos de Multi-Thread

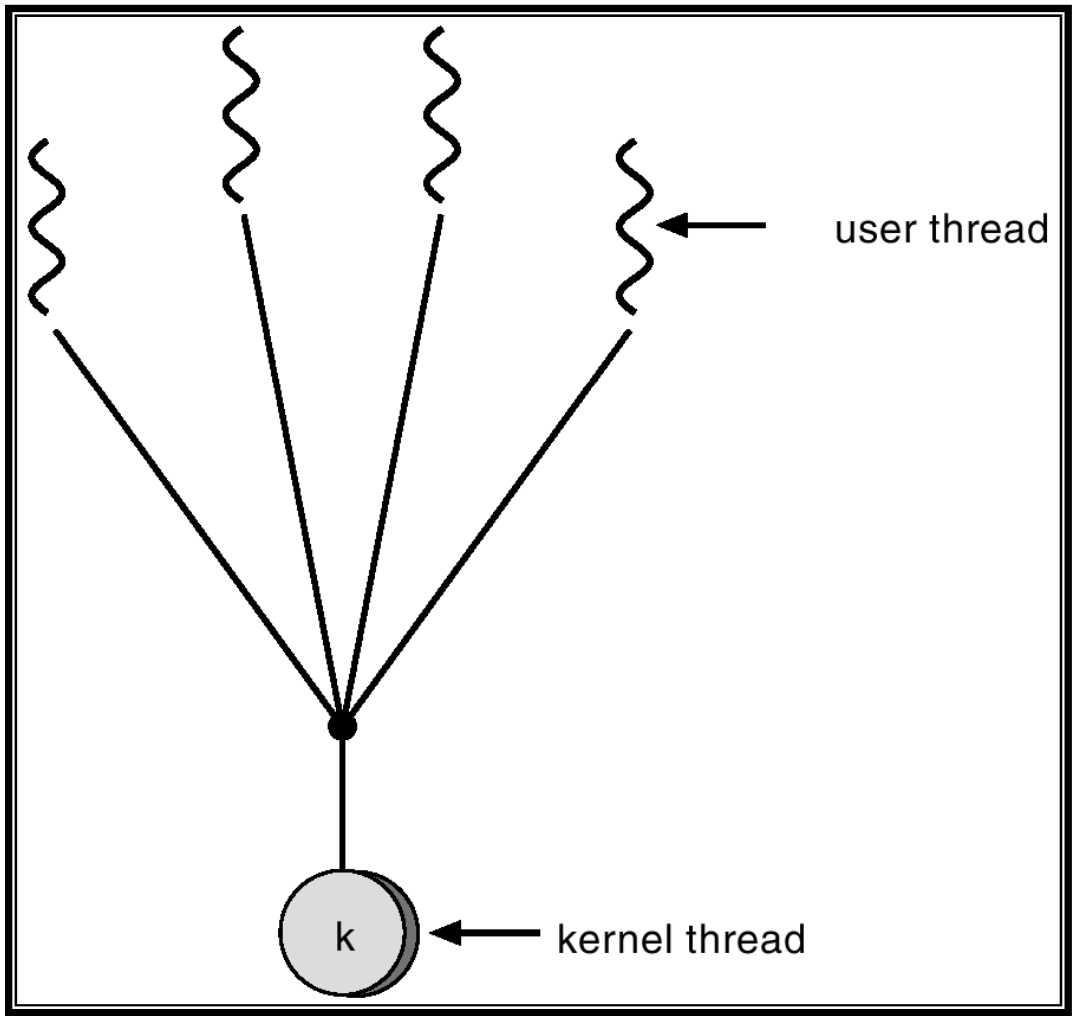
- **Um–para–Um** (Implementações UNIX)
  - Cada thread é um único processo com seu próprio espaço de endereçamento e recursos.
- **Muitos–para–Um** (Windows NT, Solaris, OS/2, dentre outros)
  - Um processo define um espaço de endereçamento e propriedade de recursos dinâmicos. Múltiplos threads podem ser criados e executados dentro deste processo.
- **Um–para–Muitos** (Ra – Clouds, Emerald)
  - Um thread pode migrar de um ambiente de processo para outro. Isto permite mover o thread entre diferentes sistemas,
- **Muitos–para–Muitos** (TRIX)
  - Combina atributos de M:1 e 1:M

Modelo Muitos–para–Um

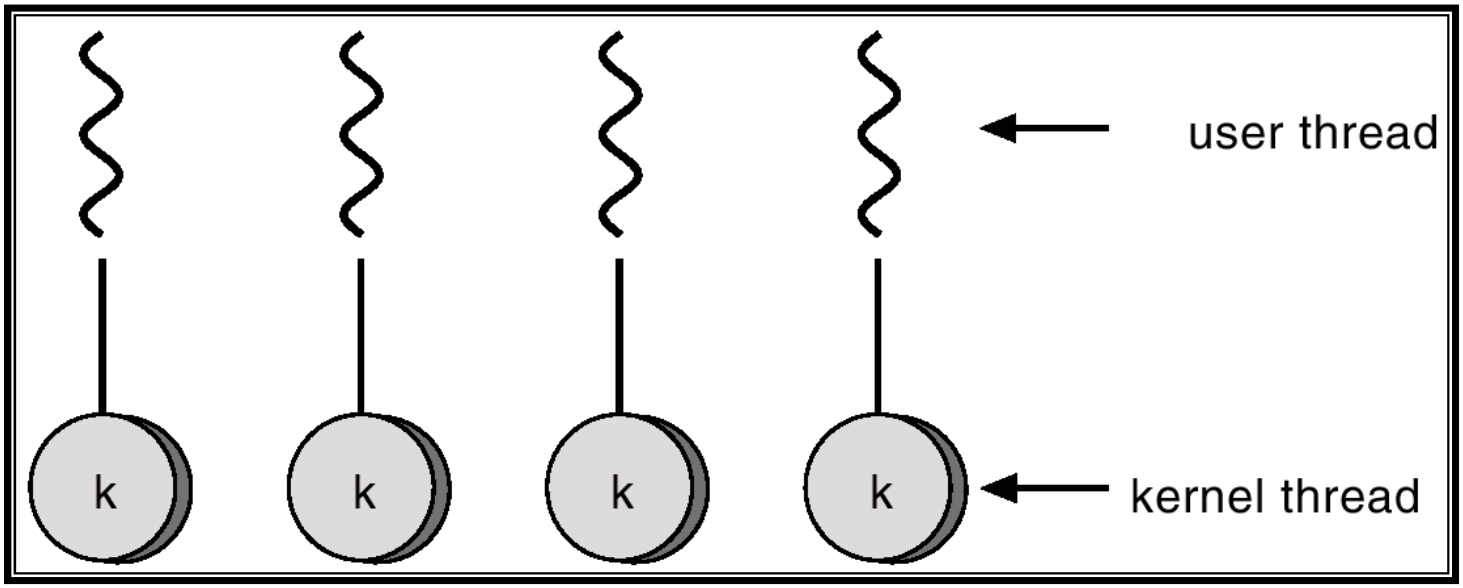
✓ *Gerenciamento do Thread é feito no espaço do usuário (RTL)*

✓ *Usado em sistemas que não suportam Threads*

✓ *Drawback: um thread fazendo uma "blocking system call" bloqueia todo o processo.*



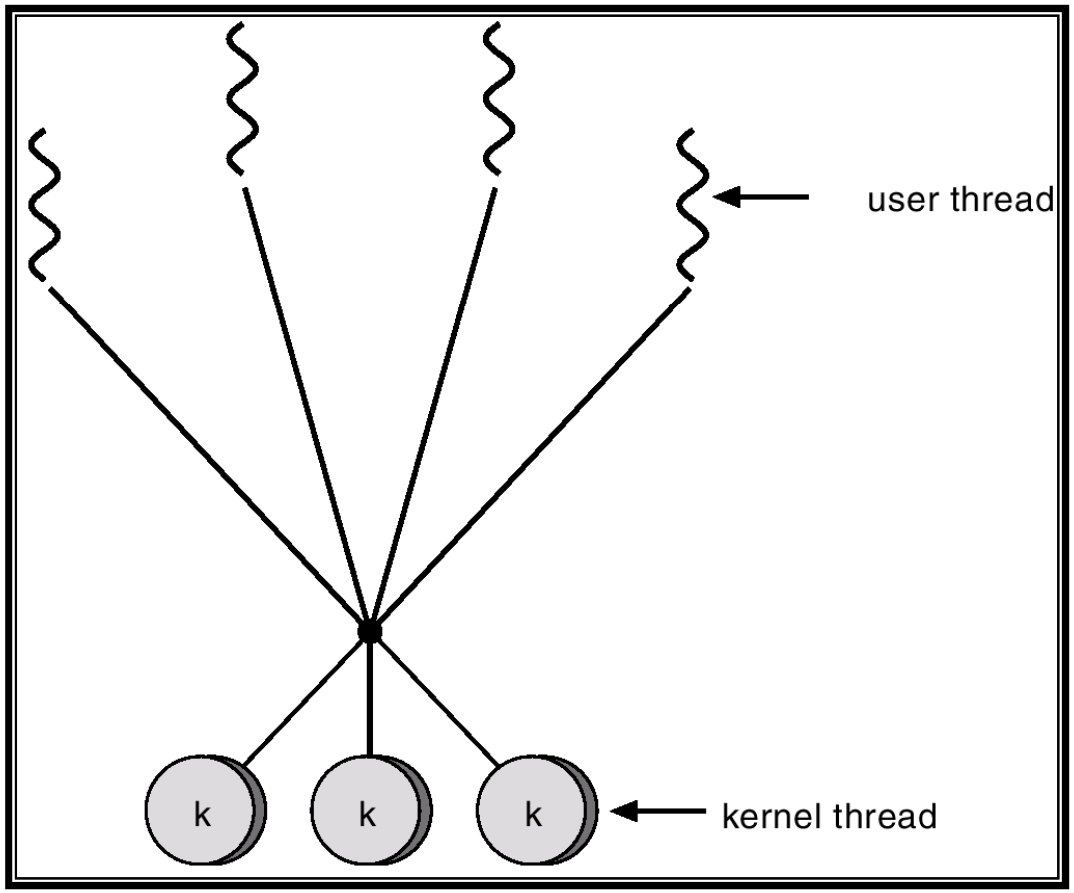
Modelo Um–para–Um



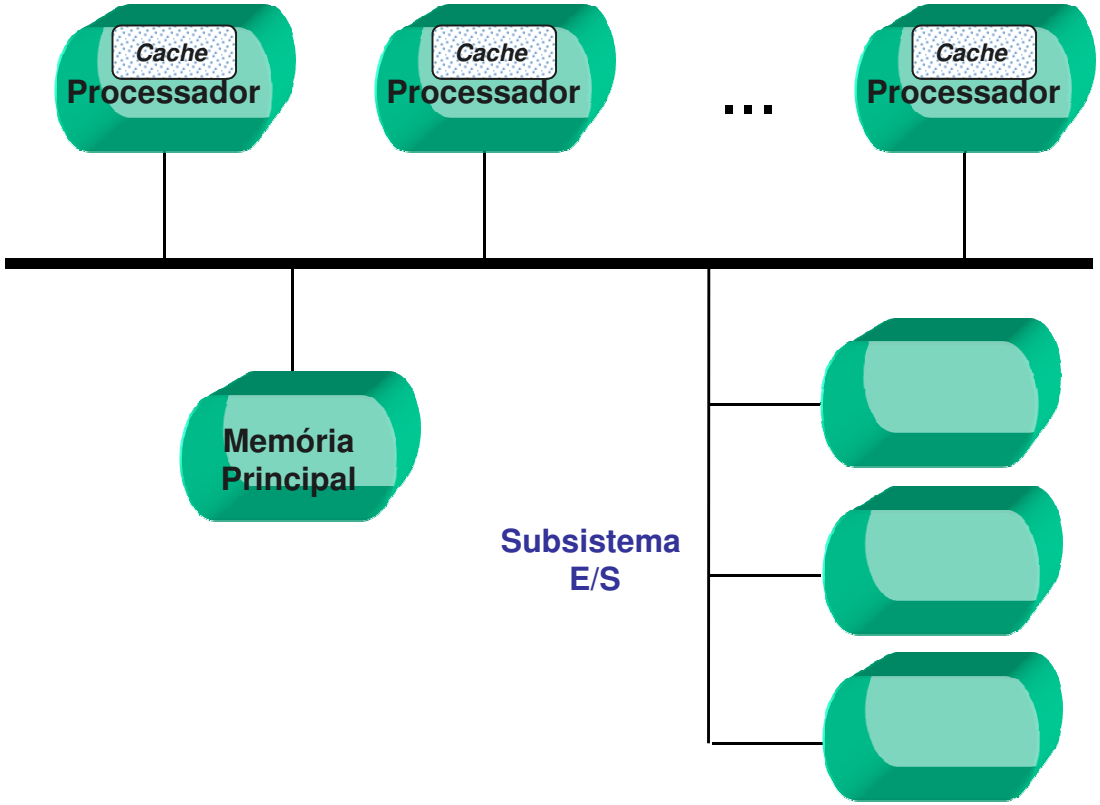
- ✓*Cada user-level thread é mapeado para um kernel thread.*
- ✓*Drawback: criar um user thread requer a criação de um kernel thread – isto pode afetar fortemente o desempenho da aplicação*
- ✓*Muitas implementações restringem o número de Threads suportados pelo sistema*

Modelo Muitos–para–Muitos

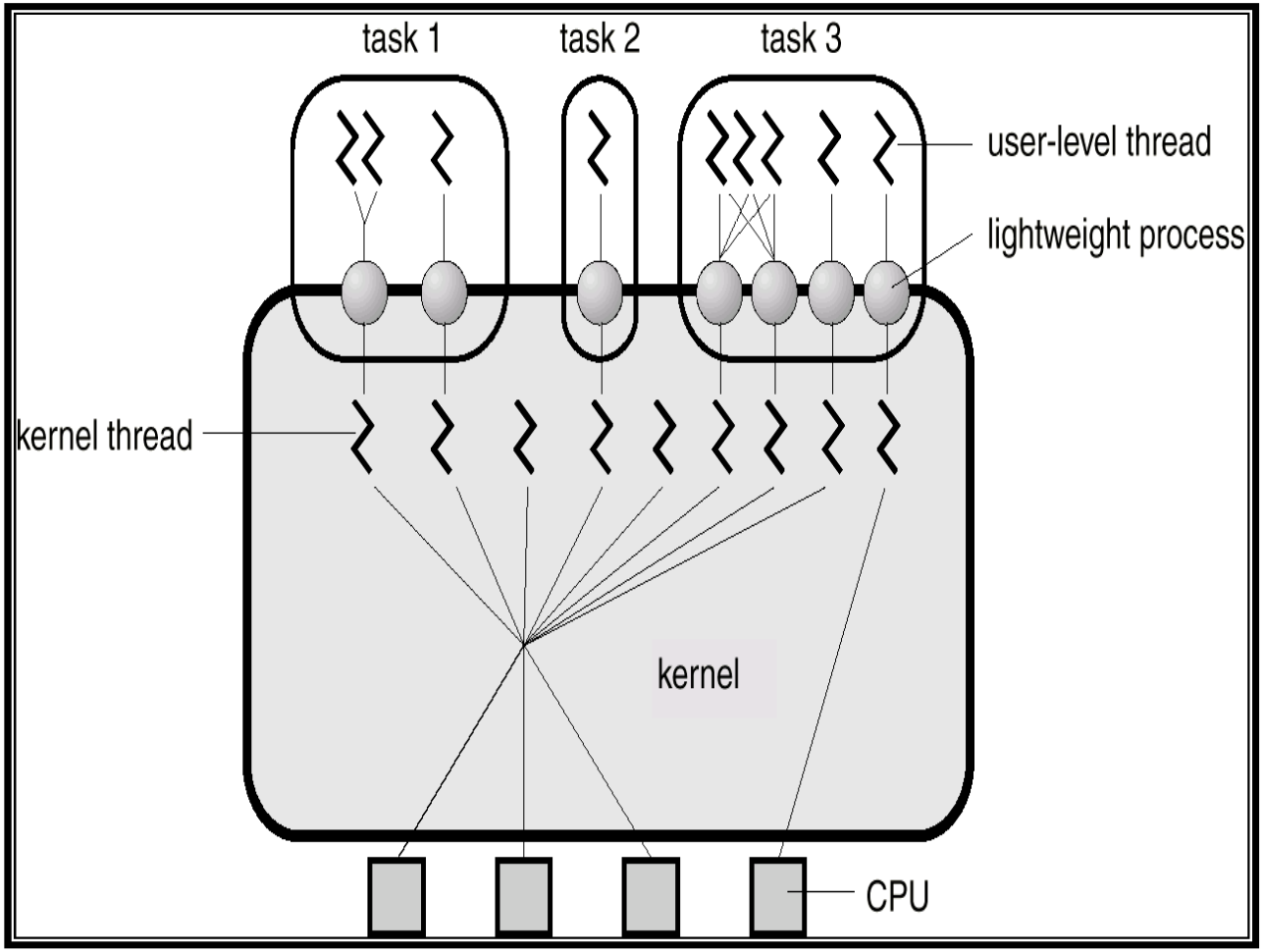
- ✓ *Permite que vários ULTs sejam mapeados para um número menor ou maior de KLTs..*
- ✓ *Permite que o SO crie um número suficiente de KLTs.*
- ✓ *Supera algumas restrições dos modelos: Many-One and One-One models*
  - ✓ ***Examples***
    - ✓ ***Solaris 2***
    - ✓ ***Tru64 UNIX***

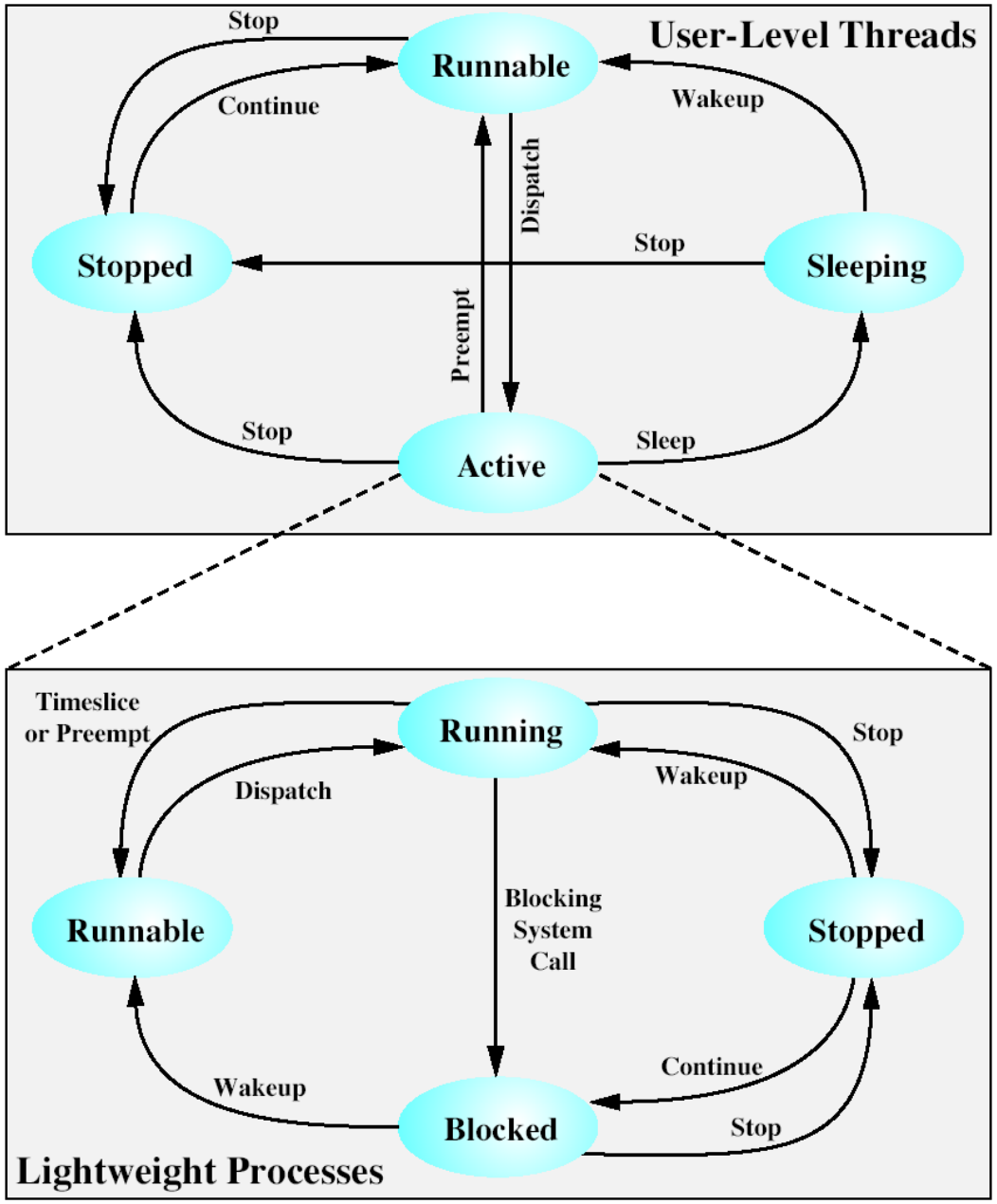


Multiprocessamento Simétrico



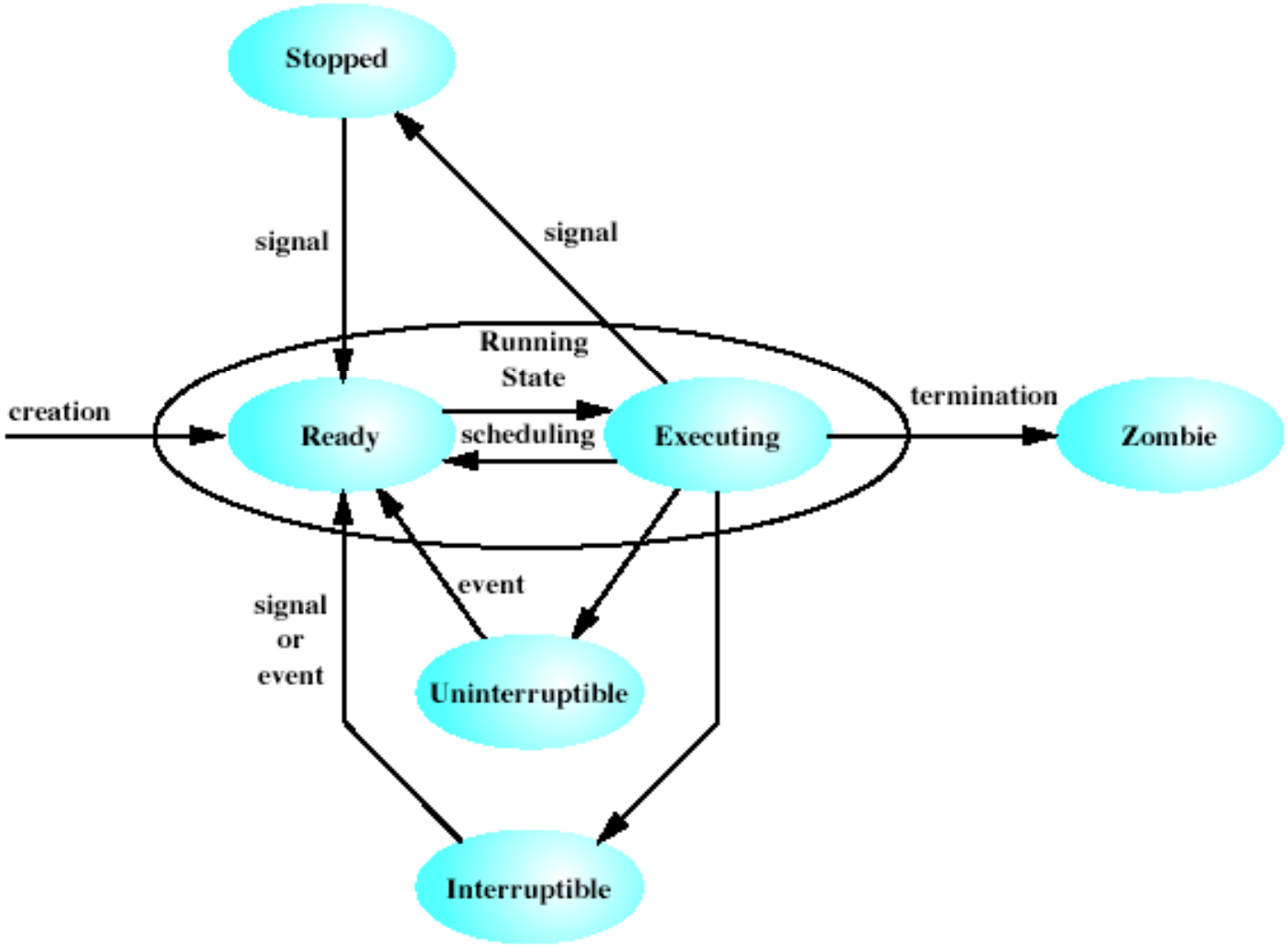
Threads no Solaris





Threads no Solaris

Threads no LINUX



Threads no Ambiente JAVA

- *Java provê suporte para criação e gerência de threads no nível da linguagem*
- *Java threads são gerenciadas pela Java Virtual Machine (JVM)*
- *Java threads podem ser criadas:*
  - *Estendendo-se a classe Thread sobrepondo a execução do método da classe. É executado como uma nova thread pela JVM sempre que o método for chamado*
  - *Um método inicial é chamado, que aloca memória e inicializa uma nova thread na JVM e chama o método responsável pela execução da thread*

Comunicação entre *Threads*

- **compartilhamento de recursos**
- **espaço de endereçamento de memória compartilhado**
- **técnicas de sincronização semelhante as utilizadas em processos**

Biblioteca de *Threads - Pthread*

Algumas funções

```
pthread_create(&id1, NULL, proc, NULL);
pthread_join(id1, NULL);
pthread_mutex_lock(&mut);
pthread_mutex_unlock(&mut);
```

Exemplo

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h

void *print_message_function( void *ptr );

main() {
pthread_t thread1, thread2;
char *message1 = "Thread 1"; char *message2 = "Thread 2";
int iret1, iret2;

iret1 = pthread_create( &thread1, NULL, print_message_function, (void*) message1);
iret2 = pthread_create( &thread2, NULL, print_message_function, (void*) message2);

pthread_join( thread1, NULL);
pthread_join( thread2, NULL);
printf("Thread 1 returns: %d\n",iret1);
printf("Thread 2 returns: %d\n",iret2);
exit(0);
}
```

Exemplo

```
void *print_message_function( void *ptr )
{
char *message;
message = (char *) ptr;
printf("%s \n", message);
 }
```

Compile: cc -lpthread pthread1.

Hyper-threading

A geração de processadores da Intel IA-32 conhecida como arquitetura Prescott, inclui uma novidade chamada SMT (**simultaneous multithreading)**, ou **hyper-threading**.

Para melhor aproveitar a funcionalidade SMT, as aplicações precisam ser escritas com múltiplos Threads e, como numa arquitetura SMP.

Quanto maior o grau de multi-thread maior a performance que a aplicação poderá extrair da arquitetura Prescott.

Hyper-threading

Outra característica da Arquitetura Prescott é ser OOE (Out of Order Execution) em tempo de execução.

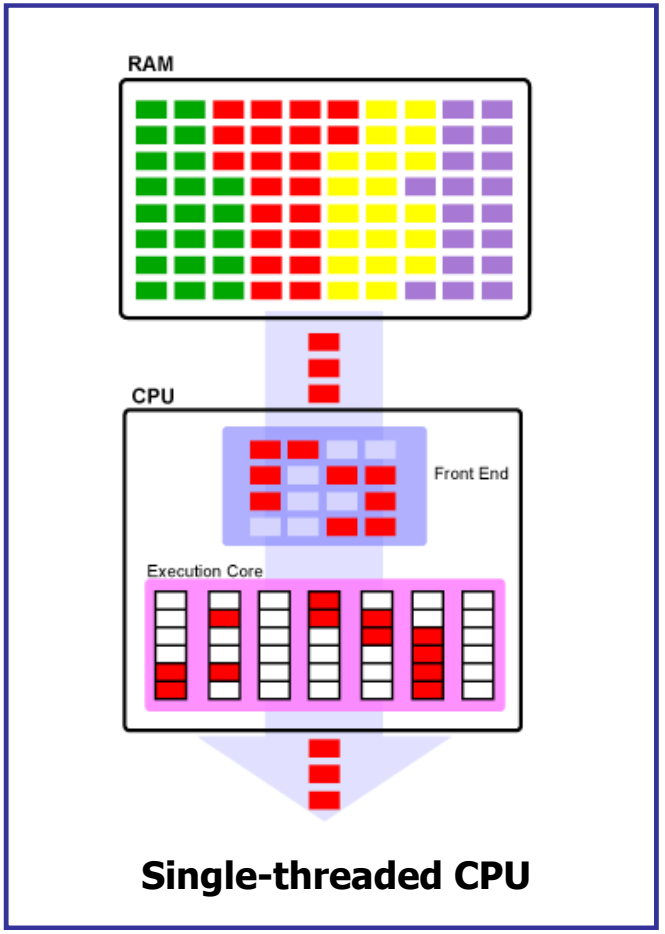
OOE pega o código escrito e compilado para ser executado em sequência e reescalona a ordem de execução (sempre que possível) para otimizar ao máximo o uso do processador. Após a execução restaura a ordem original e assim a validade dos resultados é garantida.

A operação é transparente para o programador, somente a CPU sabe a ordem em que as instruções são realmente executadas.

Hyper-threading

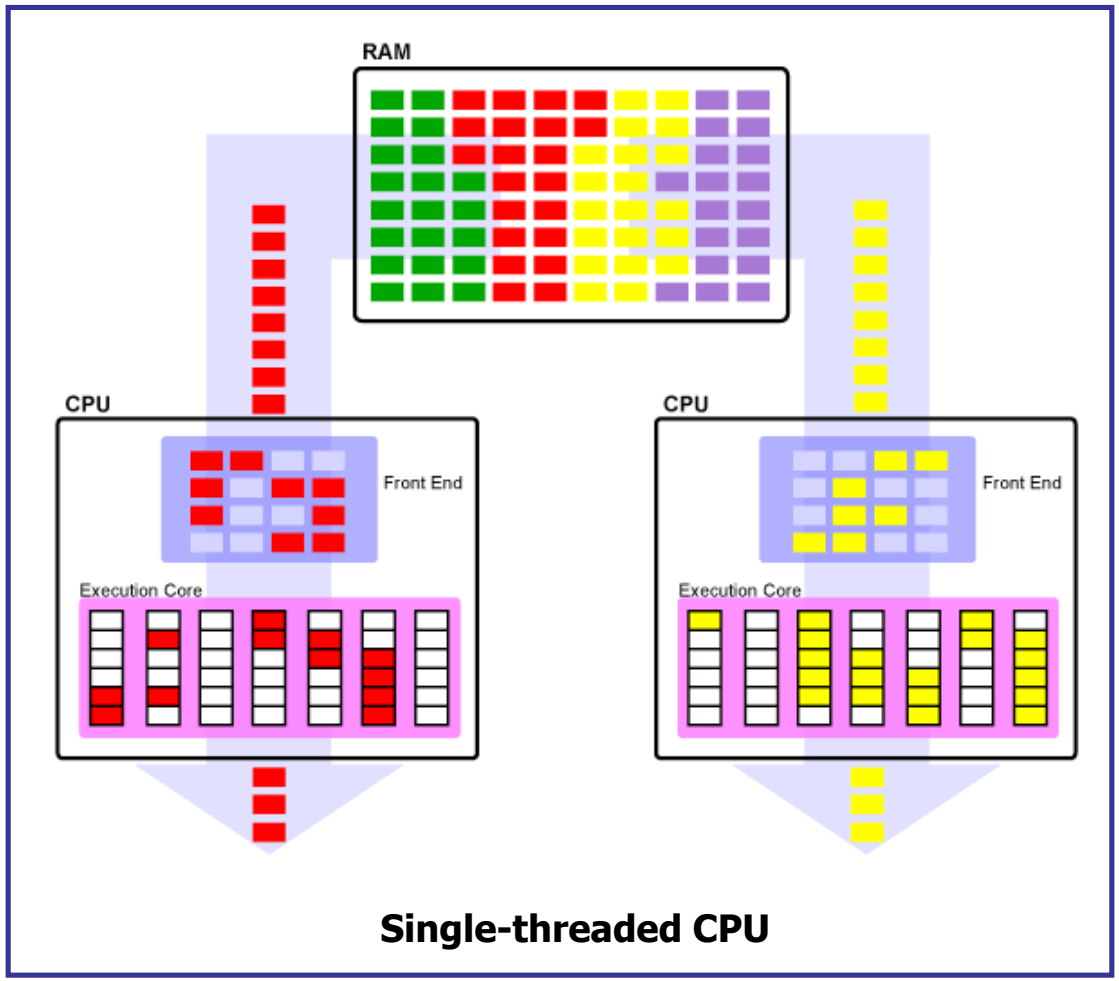
Nesta Arquitetura INTEL:

- "Front end" consiste do instruction fetcher and do decoder.
- "Back end" ou "execution core" consiste de todo o resto das operações: register ename logic, out-of-order scheduling logic.



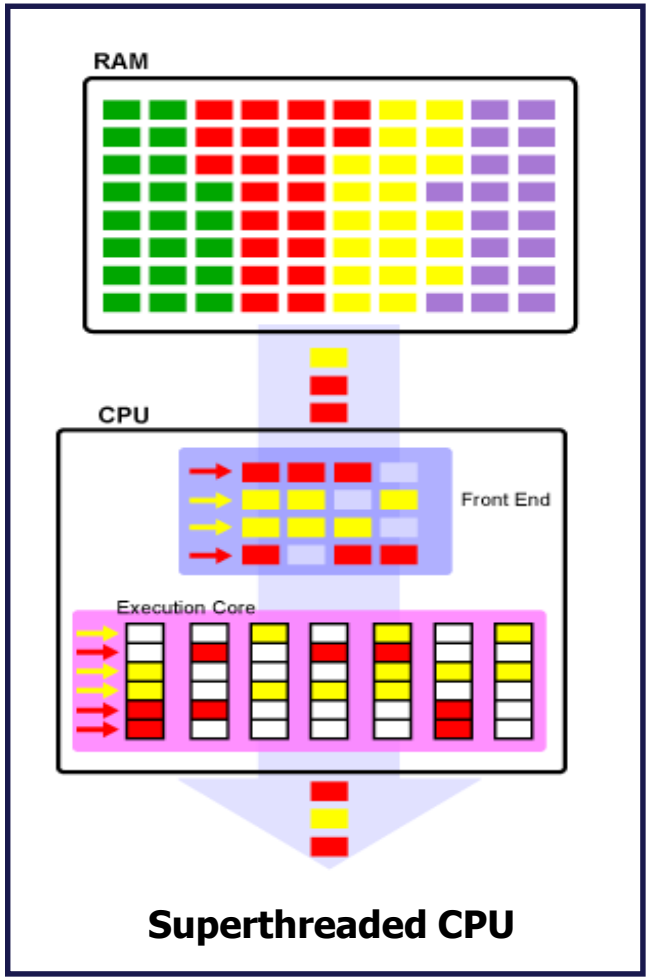
Fonte: http://archive.arstechnica.com/paedia/images/hyperthreading-2.html

Hyper-threading



Hyper-threading

Superthreading com CPUs Multithreads



Hyper-threading

Hyper-threading elimina a restrição onde, a cada clock, apenas instruções de um mesmo thread podem carregar o Front End