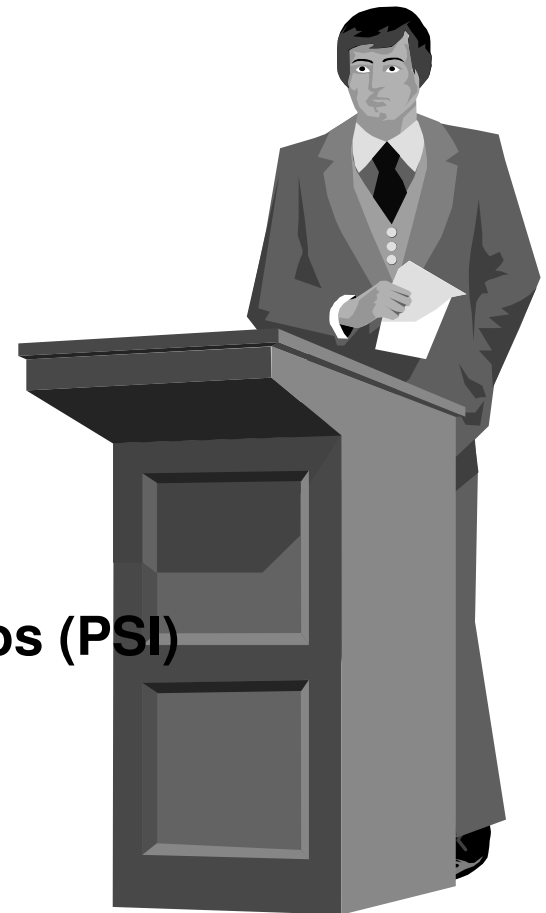


---

# *Threads*

**Volnys Borges Bernal**  
**volnys@lsi.usp.br**

**Depto. de Eng. de Sistemas Eletrônicos (PSI)**  
**Escola Politécnica da USP**



# Agenda

---

- ❑ **Processo**
- ❑ ***Threads***
  - ❖ ***Interface de threads***
  - ❖ ***Uso de threads***
- ❑ ***Interfaces de threads***
- ❑ ***Uso de threads***
- ❑ ***Threads de usuário x threads de núcleo***
  - ❖ ***Threads de usuário***
  - ❖ ***Threads de núcleo***
  - ❖ ***Soluções híbridas***

---

# Processo



# Processo

---

## ❑ Composto por:

### ❖ Contexto de software

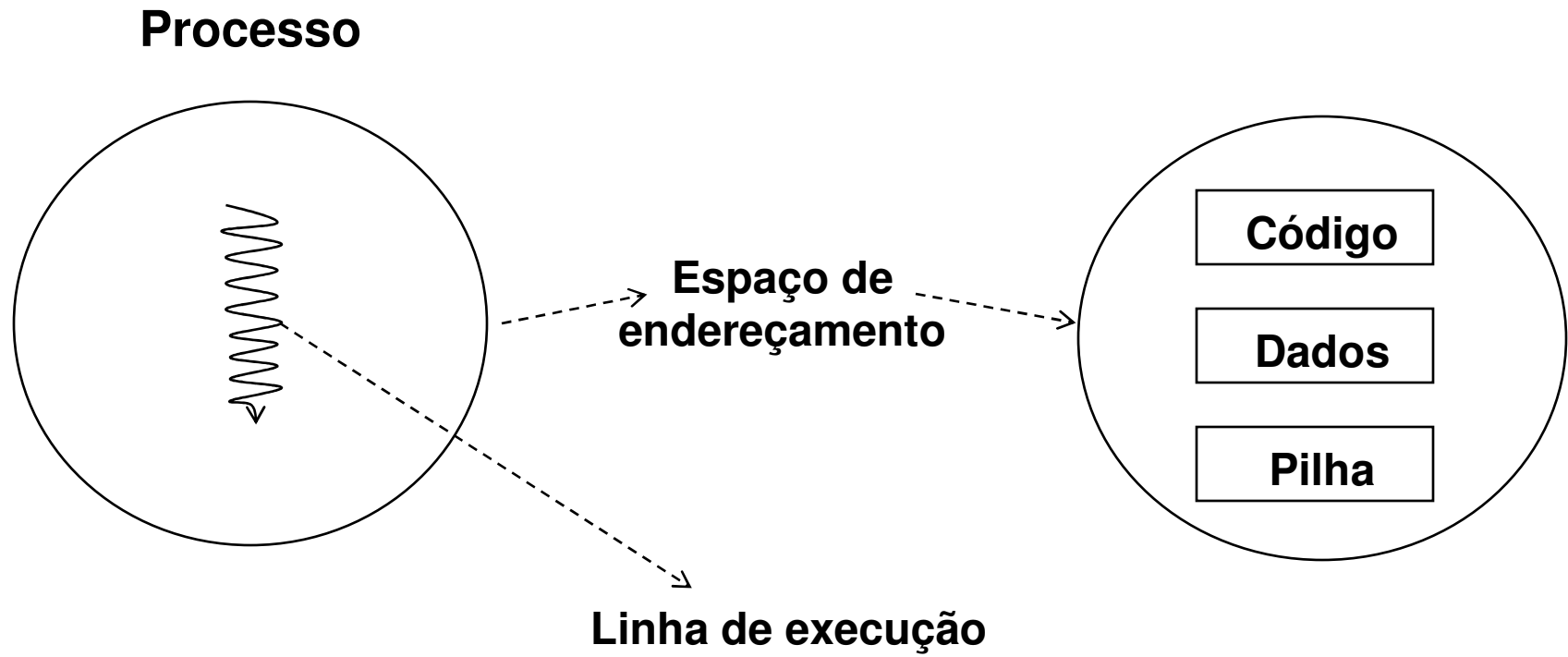
- Espaço de endereçamento
  - Área de código
  - Área de dados
  - Área da pilha de execução
- Informações de controle mantidas pelo S.O.
  - Identificação do processo (pid)
  - Identificação do usuário dono do processo
  - Identificação do terminal do qual foi disparado
  - Estado do processo
  - Arquivos abertos
  - ...

### ❖ Contexto de hardware (valores dos registradores)

- PC (program counter - contador de programa)
- SP (stack pointer – ponteiro para a pilha de execução)
- ST (status – estado)
- Registradores de números inteiros e ponto flutuante

# Processo

---



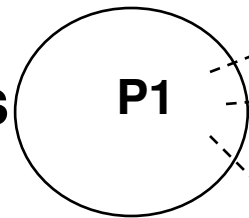
# Processo

---

- ❑ **Um processo tradicional ...**
  - ❖ Contém um único *thread* (linha de execução)
  - ❖ Que executa sobre um espaço de endereçamento próprio
  
- ❑ **Espaço de endereçamento**
  - ❖ Área de código
  - ❖ Área de dados
  - ❖ Área da pilha de execução
  
- ❑ **Linha de execução**
  - ❖ Seqüência de instruções executadas
  - ❖ Controlada pelo registrador PC (*Program Counter*)
  - ❖ Em decorrência do estado do contexto
    - áreas de memória, valores de registradores, etc

# Processo

- ❑ Duplicação de processos



Memória Virtual P1

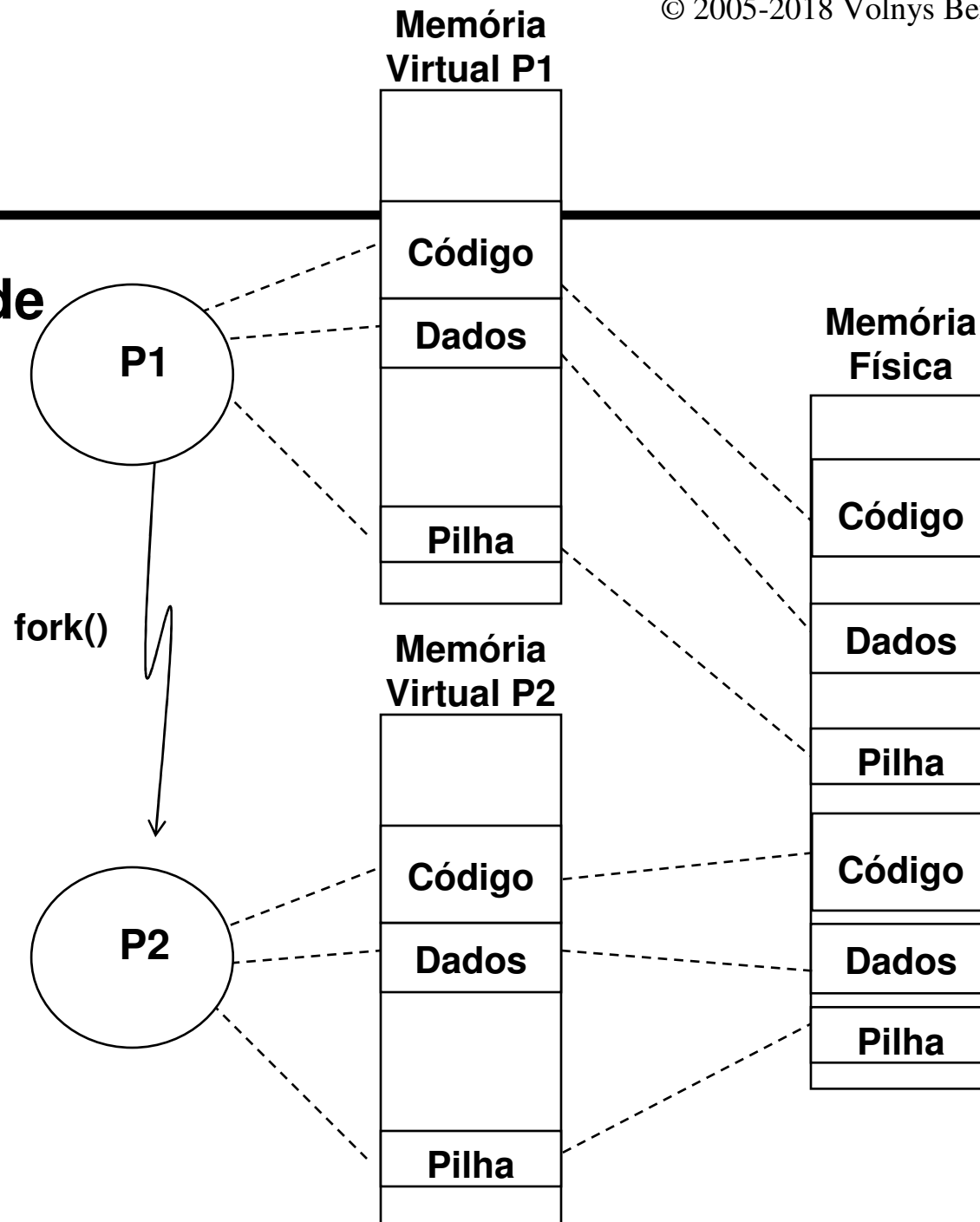


Memória Física



# Processo

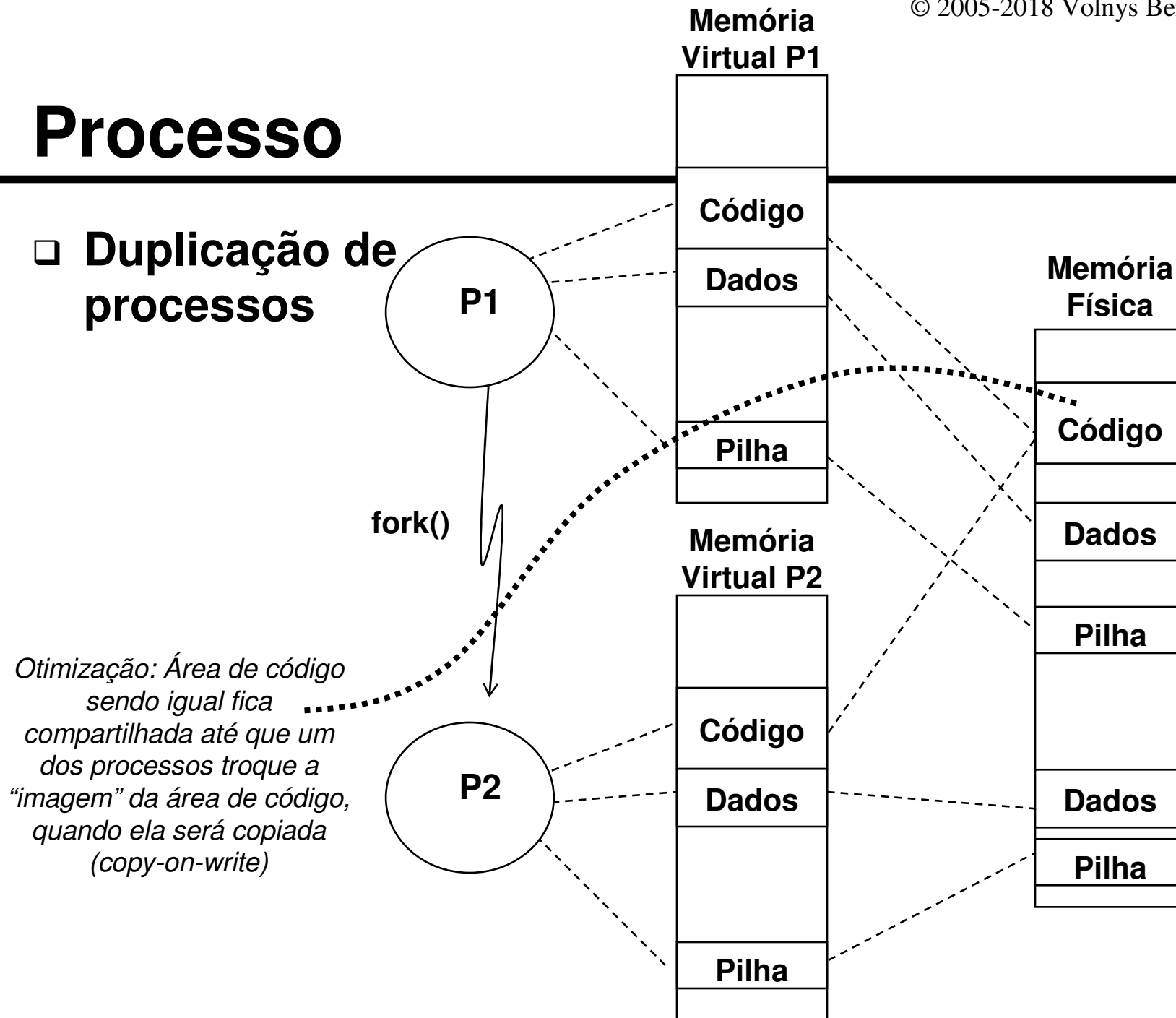
## ❑ Duplicação de processos





# Processo

## ❑ Duplicação de processos



---

# *Thread*



# Thread

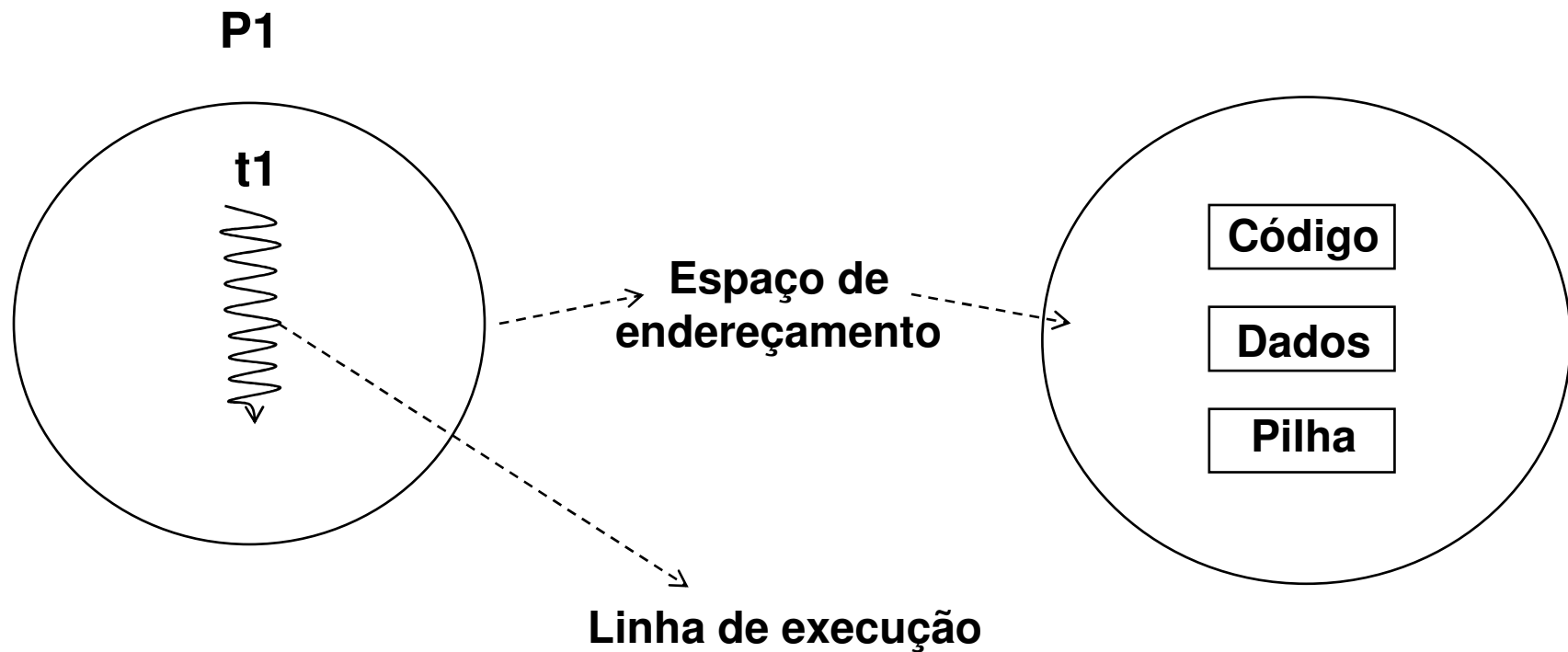
---

- ❑ **Thread = Linha de execução**
  
- ❑ **O que é?**
  - ❖ **Componente do processo relacionado ao fluxo de execução**
  - ❖ **Thread permite separar os componentes relacionados ao fluxo de execução dos outros componentes de um processo.**
  
- ❑ **Em sistemas operacionais modernos:**
  - ❖ **Um processo pode ser composto por um ou mais *threads***
  - ❖ **Alguns recursos ficam associados ao processo, outros a cada *thread***
  
- ❑ **Os recursos dependentes diretamente da execução ficam associados ao *thread*:**
  - ❖ **Informações de controle do *thread***
    - **Identificação do *thread***
    - **Área para salvamento de registradores**
    - **Estado do *thread***
  - ❖ **Registradores de CPU**
  - ❖ **Área da pilha de execução**

# Thread

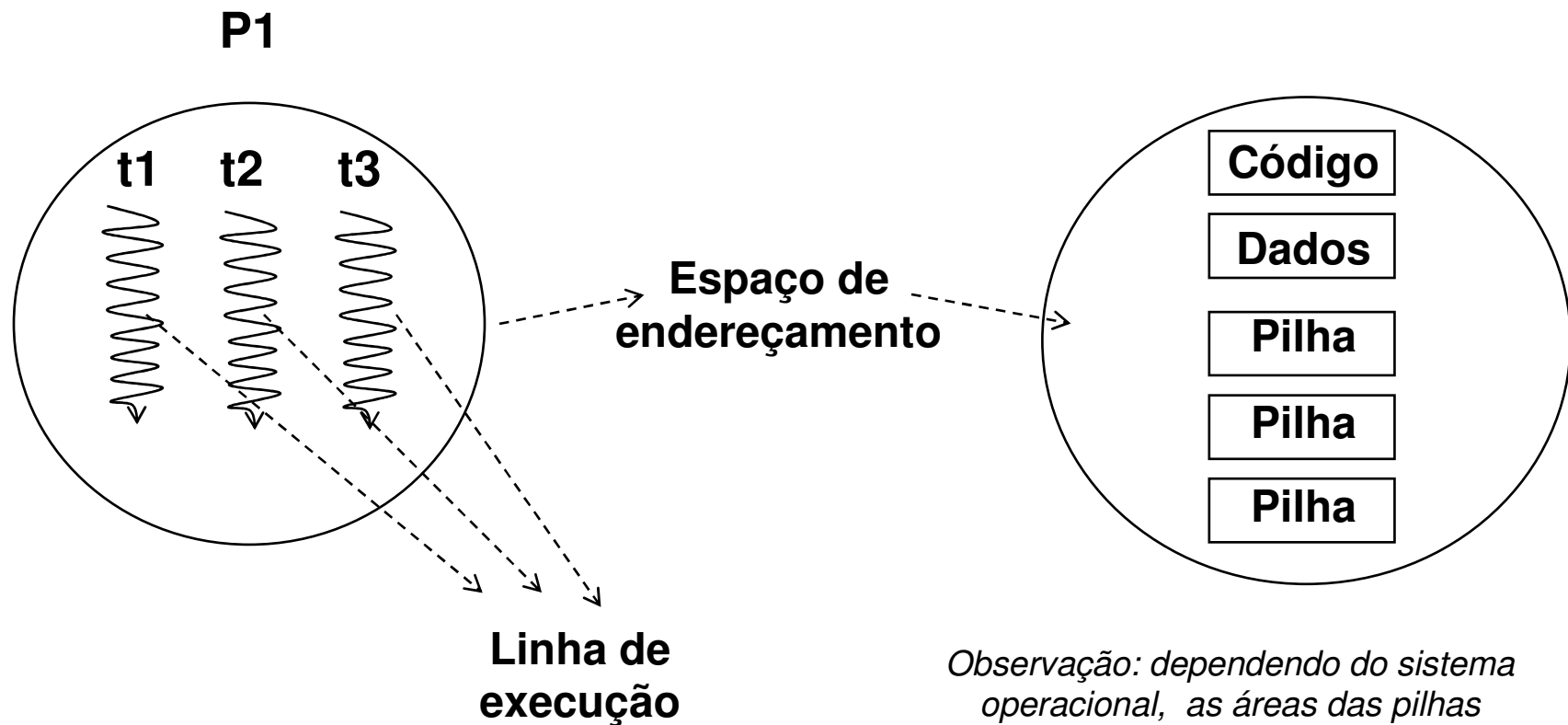
---

- Um processo com um único *thread*



# Thread

## □ Um processo com vários *threads*



*Observação: dependendo do sistema operacional, as áreas das pilhas adicionais associadas aos outros threads podem ser alocadas na área de dados*

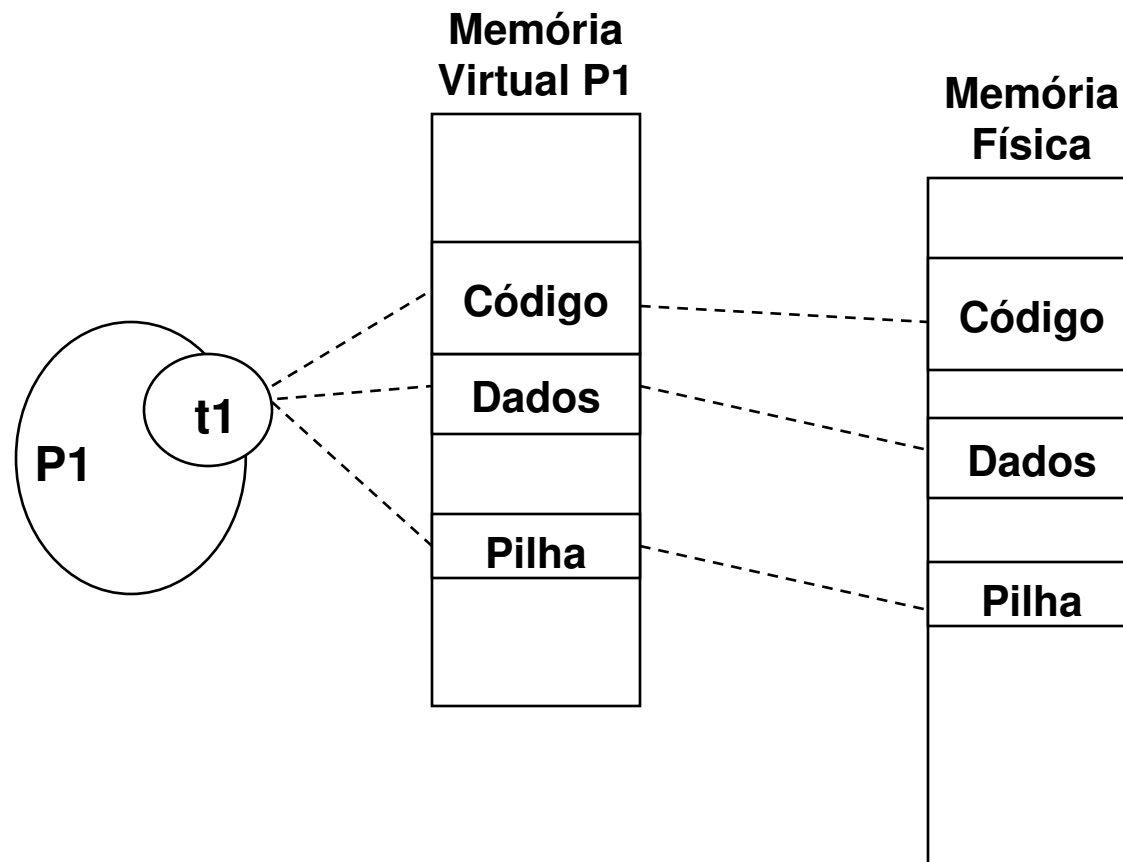
# ***Thread***

---

- ❑ **Ítems asociados ao processo**
  - ❖ **Identificação do processo**
  - ❖ **Espaço de endereçamento**
  - ❖ ***Threads***
  - ❖ **Arquivos abertos**
  - ❖ **Processos filhos**
  - ❖ **Alarmes**
  - ❖ **Sinais e tratadores de sinais**
  - ❖ **Informações de contabilidade**
  
- ❑ **Itens associados ao *thread***
  - ❖ **Identificação do thread**
  - ❖ **Registradores da CPU**
  - ❖ **Área da pilha de execução**
  - ❖ **Estado**

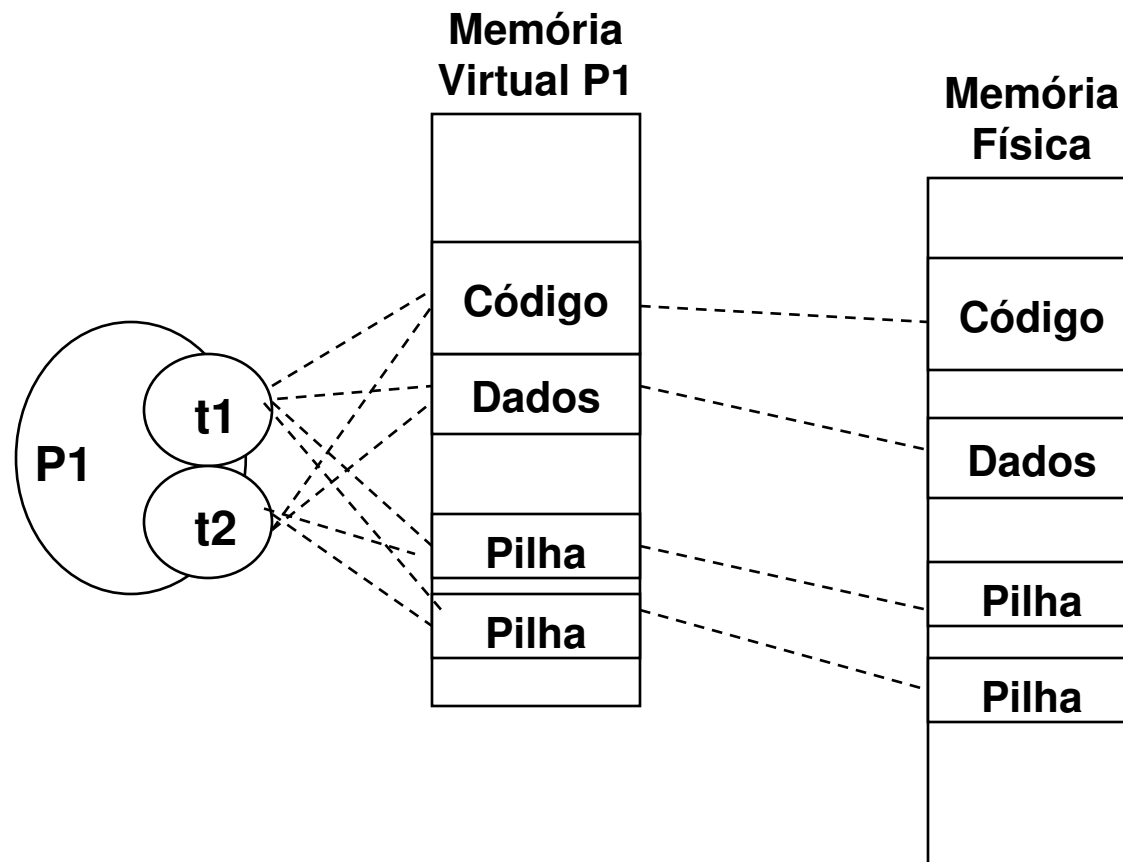
# Thread

## ❑ Criação de *threads*



# Thread

## ❑ Duplicação de *threads*





# *Thread*

---

- ❑ O estado de um processo é, na verdade, o estado do *thread* principal
  
- ❑ Estados de um thread:
  - ❖ Pronto
  - ❖ Executando
  - ❖ Bloqueado
  - ❖ Terminado
  
- ❑ Hardware
  - ❖ Monoprocessador
    - Concorrência entre *threads*
  - ❖ Multiprocessador com memória compartilhada
    - Concorrência e paralelismo entre *threads*

# ***Thread***

---

❑ ***Threads* compartilham todo o espaço de endereçamento do processo!**

❖ **Compartilham**

- Área de código
- Área de dados
- Áreas da pilha de execução (de todos os threads)

❖ **Importante observar:**

- Área de dados é compartilhada
- Área da pilha de execução é compartilhada: apesar do thread possuir sua própria pilha de execução ele tem acesso às pilhas de todos os threads

---

# Exercício



# Exercicio

---

**(1) Compile o programa “mythread.c” utilizando a biblioteca libpthread:**

```
cc -o mythread mythread.c -lpthread
```

**(2) Execute o programa mythread e verifique o resultado da execução:**  
**./mythread**

**(3) Responda:**

**(a) Quantos threads existem executando simultaneamente após o disparo dos threads?**

**(b) A variavel “i” e’ local ou global?**

**(4) Altere o programa e redefina a variavel “i” como sendo global. Qual o comportamento do programa? Explique!**

# Exercício

---

```
//Programa mythread.c

imprimir_msg(char *nome)
{
    int i=0;
    while (i<10)
    {
        printf("Thread %s - %d\n", nome, i);
        i++;
        sleep(2);
    }
    printf("Thread %s terminado \n", nome);
}

int main()
{
    pthread_t thread1;
    pthread_t thread2;
    printf("Programa de teste de pthreads \n");
    printf("Disparando primeiro thread\n");
    pthread_create(&thread1, NULL, (void*) imprimir_msg, "thread_1");
    printf("Disparando segundo thread\n");
    pthread_create(&thread2, NULL, (void*) imprimir_msg, "thread_2");
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    printf("Terminando processo");
}
```

# Exercício

---

```
//Programa mythread.c modificado

int i = 0;

imprimir_msg(char *nome)
{
    int i = 0; ← REMOVER !
    while (i<10)
    {
        printf("Thread %s - %d\n", nome, i);
        i++;
        sleep(2);
    }
    printf("Thread %s terminado \n", nome);
}

int main()
{
    pthread_t thread1;
    pthread_t thread2;
    printf("Programa de teste de pthreads \n");
    printf("Disparando primeiro thread\n");
    pthread_create(&thread1, NULL, (void*) imprimir_msg, "thread_1");
    printf("Disparando segundo thread\n");
    pthread_create(&thread2, NULL, (void*) imprimir_msg, "thread_2");
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    printf("Terminando processo");
}
```

---

# Interfaces de threads



# Interfaces de Threads

---

- **Existem várias interfaces para manipulação de threads**
  - ❖ **Pthreads (IEEE POSIX 1003.1c)**
  - ❖ **Cthreads (OSF/Mach)**
  - ❖ **User Level Threads (Solaris)**
  - ❖ **Light Weight Process (Solaris)**
  - ❖ **etc**



# Interfaces de Threads

---

## ❑ Interface pthreads

### ❖ Chamadas básicas para controles de threads:

- Pthread\_create()
  - Cria um novo thread para o mesmo processo
- Pthread\_exit()
  - Termina o thread
- Pthread\_join()
  - Aguarda um thread terminar
- Pthread\_yield()
  - Libera o processador para outro thread

---

# Uso de threads



# Uso de threads

---

## □ Principais usos

- ❖ **Processamento numérico paralelo**
  - (sistemas multiprocessadores)
  
- ❖ **Sistemas que controlam “interfaces” simultâneas**
  - Interface com usuário
  - Interface de rede (transmissão e recepção)
  - Interface com dispositivos
  
- ❖ **Sistemas que oferecem serviços**
  - Ex: servidor WEB

# Uso de threads

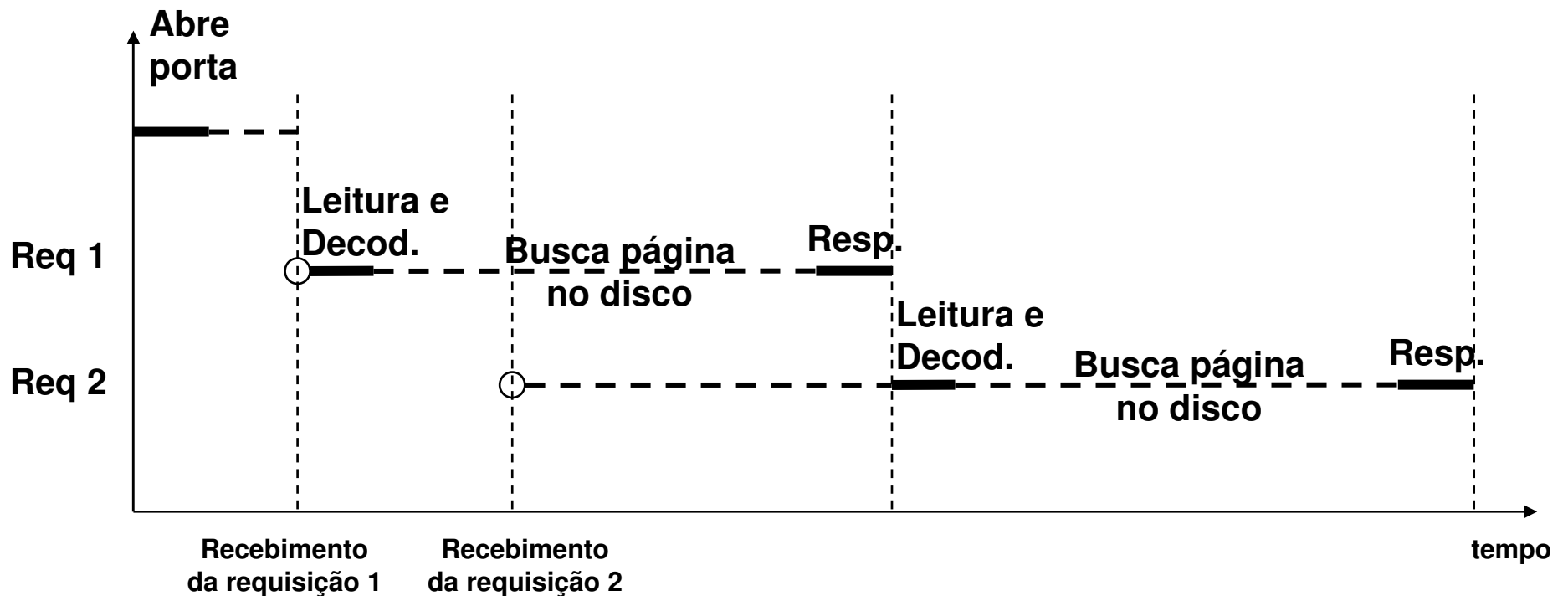
---

## □ Exemplo de servidor WEB monothread:

```
ServidorWEB ()
{
  Abre porta TCP/80
  While (TRUE)
    Aguarda requisição
    Leitura da mensagem HTTP
    Decodifica requisição HTTP
    Busca página no disco
    Responde requisição
  }
}
```

# Uso de threads

- ❑ Exemplo de servidor WEB monothread:



# Uso de threads

---

- ❑ Exemplo de servidor WEB multithread:

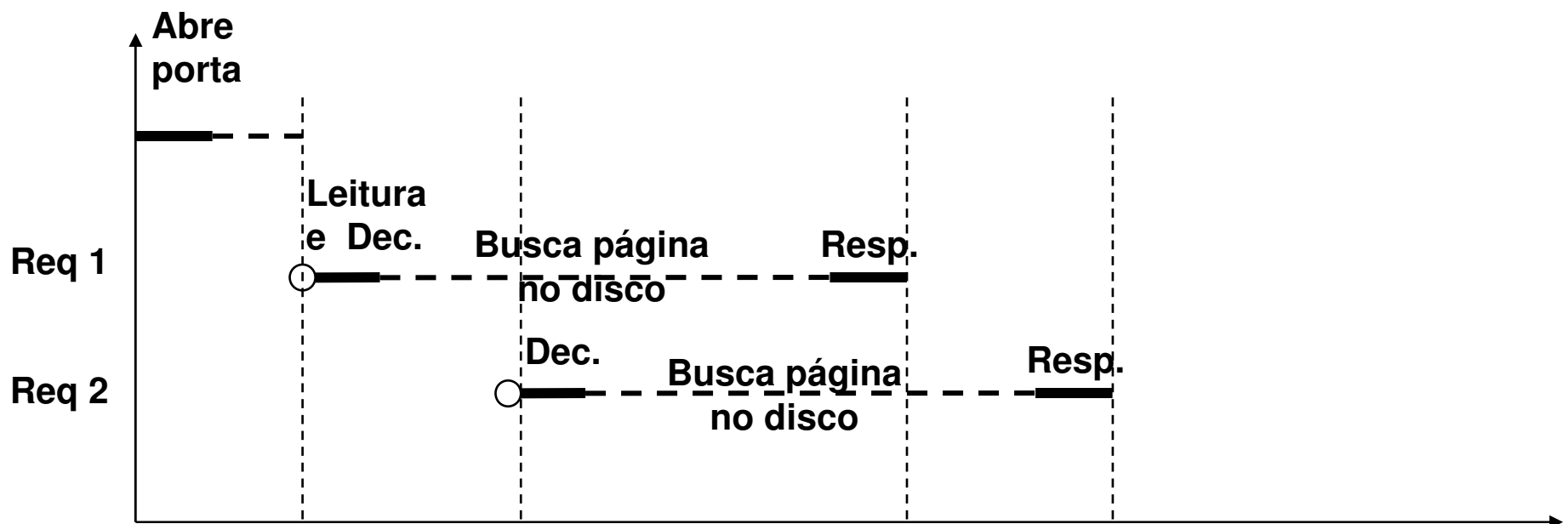
```
ServidorWEB()  
{  
  Abre porta TCP/80  
  While (TRUE)  
    Aguarda requisição  
    Dispara thread operário  
  }  
}
```

```
ThreadOperário()  
{  
  Leitura da mensagem HTTP  
  Decodifica requisição HTTP  
  Busca página no disco  
  Responde requisição  
}
```

# Uso de threads

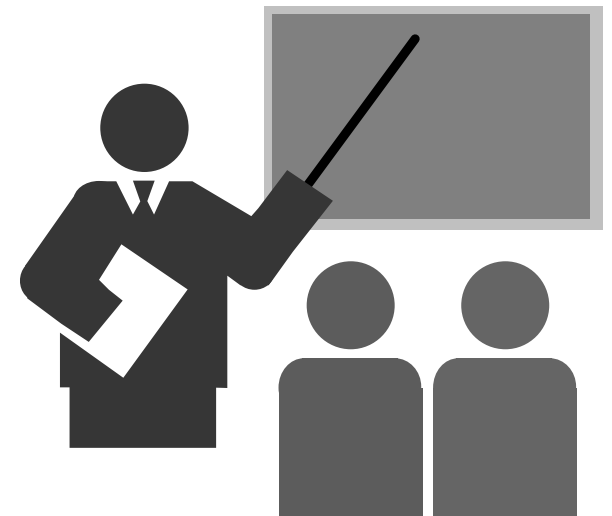
---

## □ Exemplo de servidor WEB multithread:



---

# ***Threads de usuário*** **X** ***Threads de núcleo***





# ***Threads de usuário x threads de núcleo***

---

## □ **Existem 2 formas de implementação de *threads*:**

### ❖ ***Threads de usuário***

- A abstração de threads é criada por um conjunto de rotinas de biblioteca utilizada pelo próprio processo

### ❖ ***Threads de núcleo***

- A abstração de threads é criada pelo núcleo do sistema operacional

# ***Threads* de usuário x *threads* de núcleo**

---

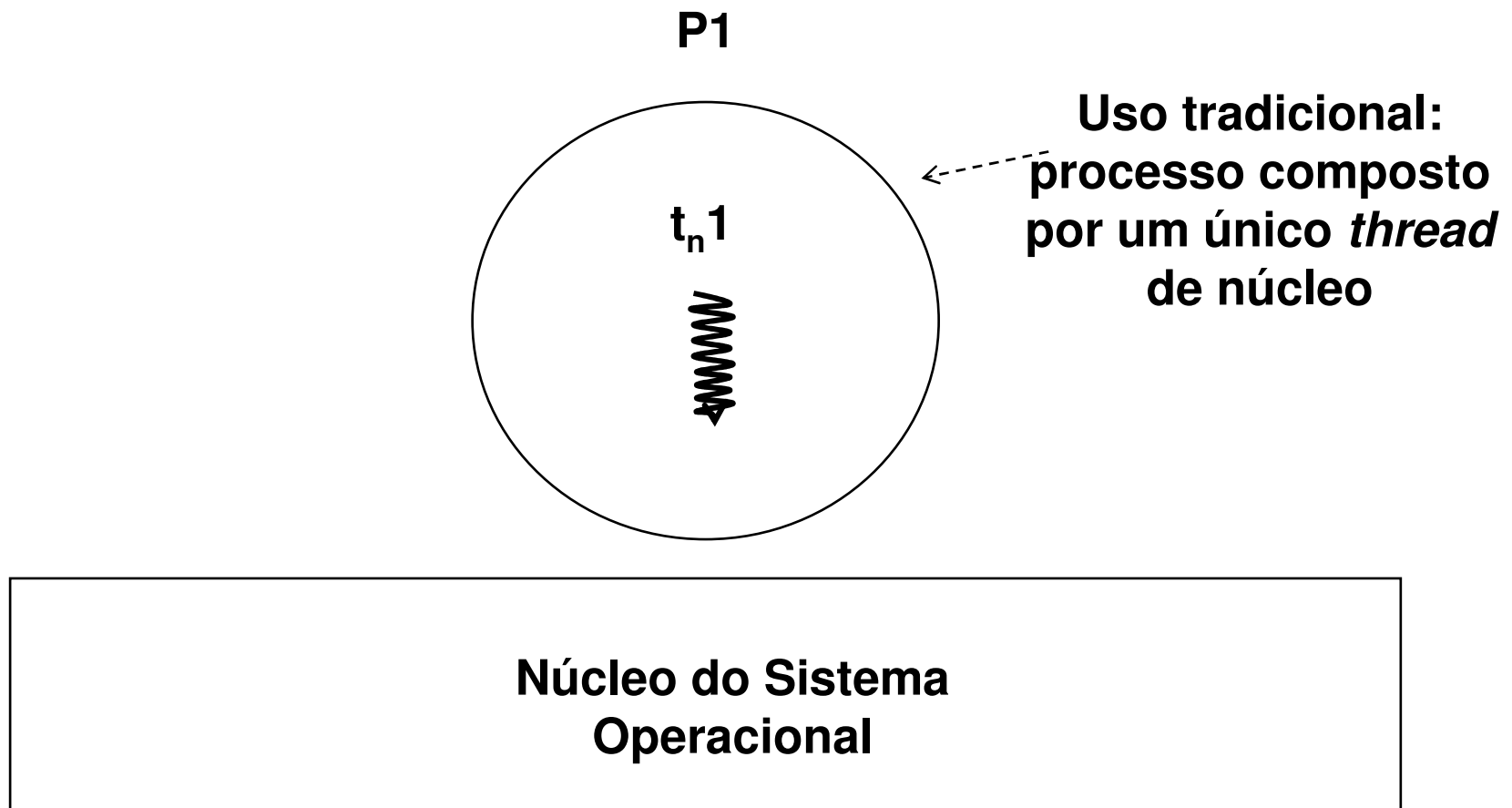
## □ ***Threads* de usuário**

- ❖ **Abstração de *threads* é criada pelo próprio processo (que executa em modo usuário, daí o nome de *thread* de usuário)**
- ❖ **O sistema operacional não tem conhecimento da existência dos *threads* de usuário**
- ❖ **O núcleo do sistema operacional considera o processo monothread**
- ❖ **Sistema supervisor**
  - Biblioteca que fornece a abstração de *threads* de usuário
  - Contém um conjunto de funções de biblioteca que possibilita gerenciar os *threads* de usuário
  - Contém uma tabela de gerenciamento de *threads*
  - Implementa uma troca de contexto entre os *threads* do processo (salvamento e restauração de alguns registradores)

# *Threads* de usuário x *threads* de núcleo

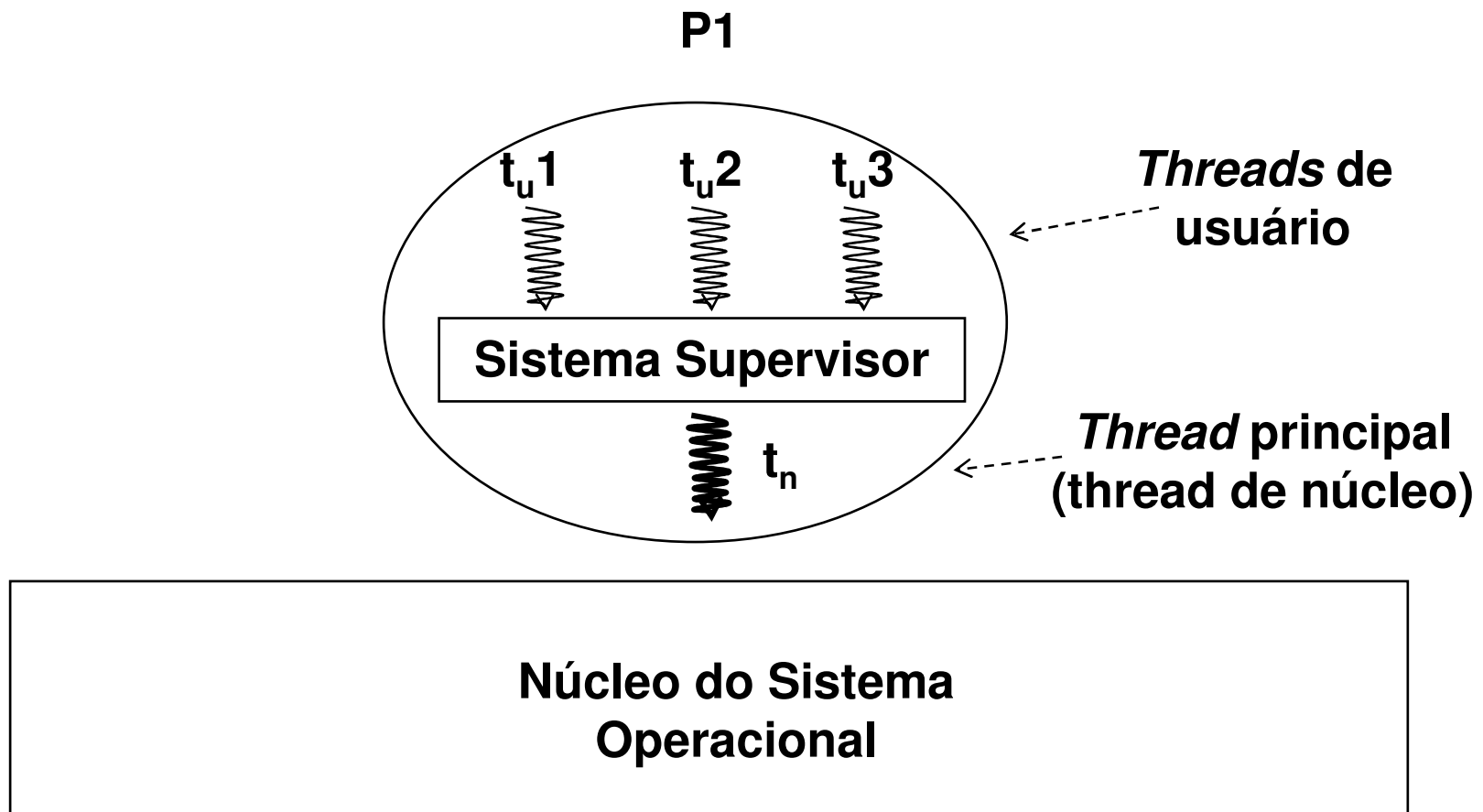
---

## □ *Threads* de usuário



# *Threads* de usuário x *threads* de núcleo

## □ *Threads* de usuário



# ***Threads* de usuário x *threads* de núcleo**

---

## □ ***Threads* de usuário**

### ❖ **Vantagens**

- Pode ser implementado por um sistema operacional que não suporte a abstração de *threads*
  - (atualmente a maior parte do sistemas operacionais fornecem a abstração de *threads*)
- Troca de contexto entre os *threads* de usuário não envolve a passagem de modo usuário para modo supervisor (chamada ao sistema), sendo muito mais rápida
- Possibilita customizar o algoritmo de escalonamento entre os *threads* de usuário do próprio processo

# ***Threads de usuário x threads de núcleo***

---

## □ ***Threads de usuário***

### ❖ **Desvantagens**

- Chamadas ao sistema bloqueantes
- *Thread* de usuário não é interrompido pelo escalonador
  - Sistema supervisor não atende interrupções do temporizador
  - Cada *thread* de usuário deve ceder a CPU (`thread_yield`), de tempos em tempos, para os outros *threads* de usuário
  - Poderia estar requisitando um alarme do S.O periódico
    - → Problema: sobrecarga de processamento

# ***Threads* de usuário x *threads* de núcleo**

---

## □ ***Threads* de núcleo**

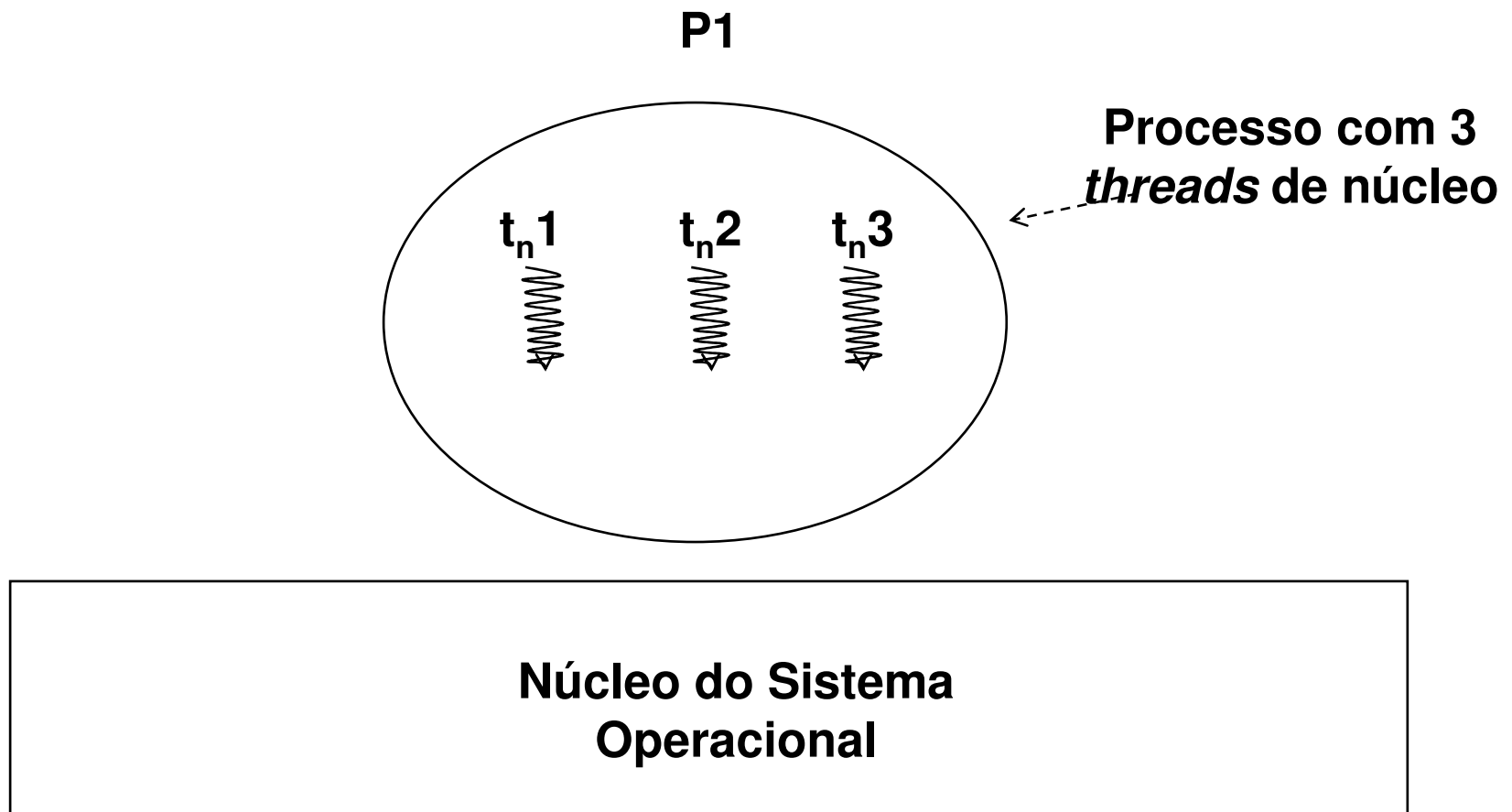
### ❖ O núcleo do sistema operacional :

- Fornece interfaces de gerenciamento de *threads*
  - Através de chamadas ao sistema e rotinas bibliotecas
- Possui conhecimento dos *threads* do processo
- Gerencia os *threads* de núcleo do processo:
  - Mantém a tabela de *threads* de núcleo
  - Realiza o escalonamento dos *threads* de núcleo

# *Threads* de usuário x *threads* de núcleo

---

## □ *Threads* de núcleo





# ***Threads de usuário x threads de núcleo***

---

## □ ***Threads de núcleo***

### ❖ **Desvantagem**

- Sobrecarga de gerenciamento: Cada chamada de função de gerenciamento de *threads* é uma chamada ao sistema

# ***Threads* de usuário x *threads* de núcleo**

---

- Mapeamento de *threads* de usuário em *threads* de núcleo
  - ❖ Solução híbrida

# ***Threads de usuário x threads de núcleo***

---

## **□ Outros problemas**

### **❖ Variáveis globais privadas**

- Exemplo: variável interna da linguagem C “errno”