Sistemas Operacionais
Escalonamento

Carlos Ferraz (cagf@cin.ufpe.br)
Jorge Cavalcanti Fonsêca (jcbf@cin.ufpe.br)

Copyright

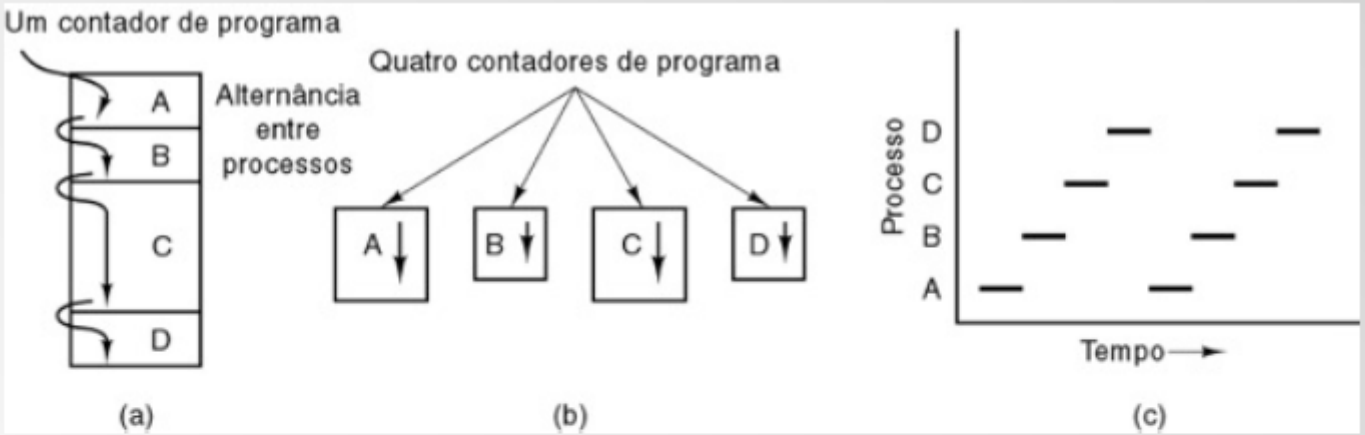
Carlos Ferraz – Cin/UFPE

Escalonamento

Escalonamento

Decidindo qual processo vai executar

Processos Concorrentes
O modelo de multiprogramação



- Multiprogramação de quatro programas
- Modelo conceitual de 4 processos sequenciais, independentes
- Somente um programa está ativo a cada momento

Escalonamento

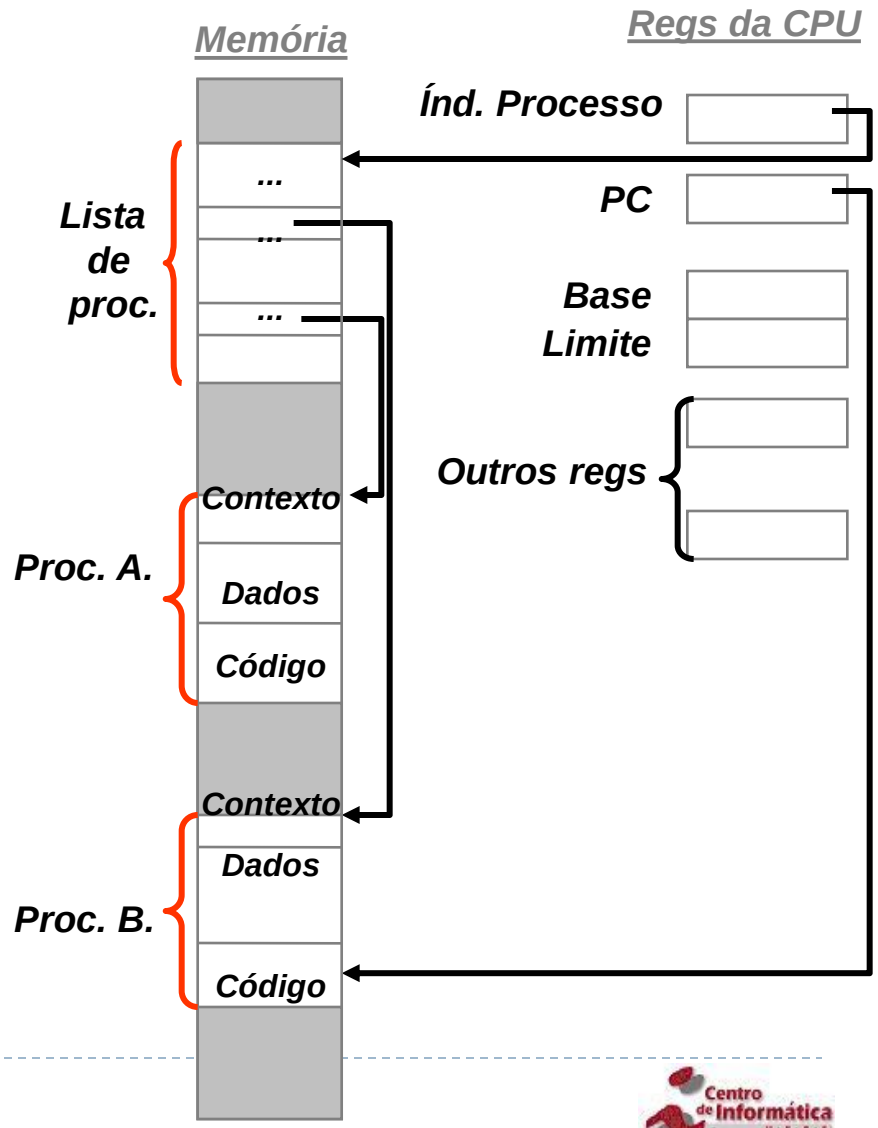
- Como evitar que um processo monopolize o sistema?

Sistemas de tempo compartilhado (Time Sharing Systems)

- Permite sistemas interativos (entrada/saída)
- Requer temporizadores (timers)
- Interrupções

Multiprocessamento

- O índice do processo contém o apontador para a lista de processos
- PC (Program Counter) = contador de programas
- Uma troca de processos consiste em trocar o valor dos registradores de contexto da CPU



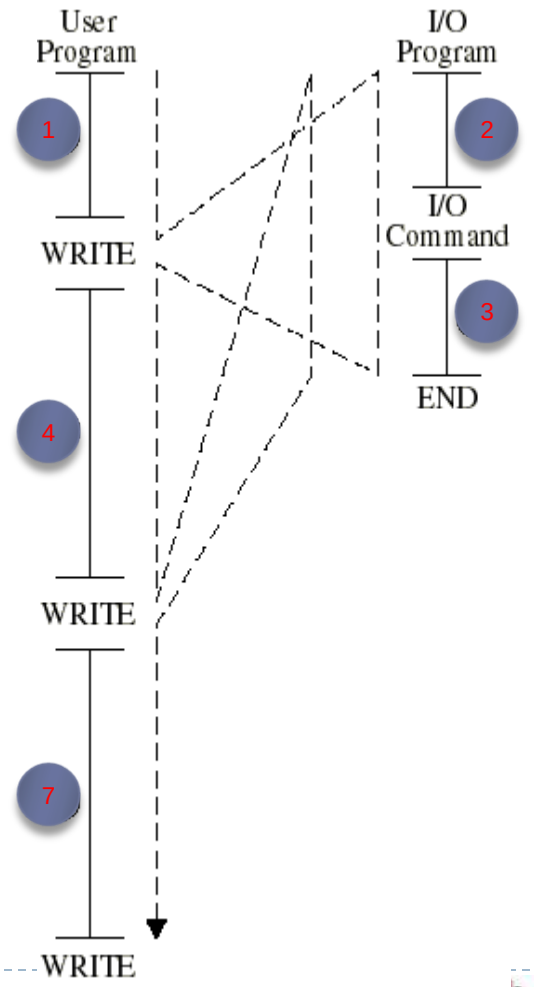
Multiprocessamento

- O que é necessário para haver multiprocessamento?
  - Suporte do Hardware
    - Temporizadores (timers)
    - Interrupções
    - Proteção de memória
  - Suporte do S.O.
    - Escalonamento dos processos
    - Alocação de memória
    - Gerenciamento dos periféricos

A importância da Interrupção

- Num sistema simples, CPU deve esperar a execução do comando de E/S
  - A cada chamada do comando *write* a CPU fica esperando o dispositivo executar o comando.

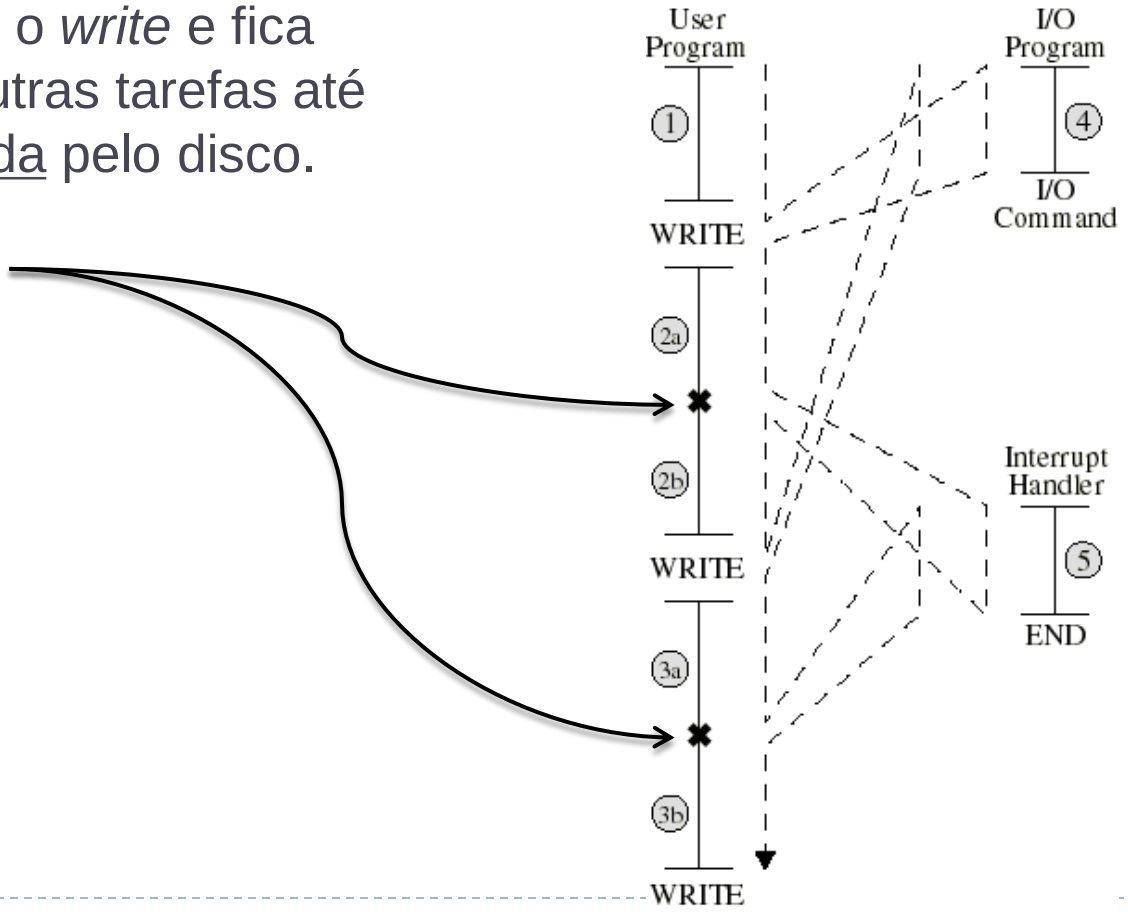
Ex: escrita em disco



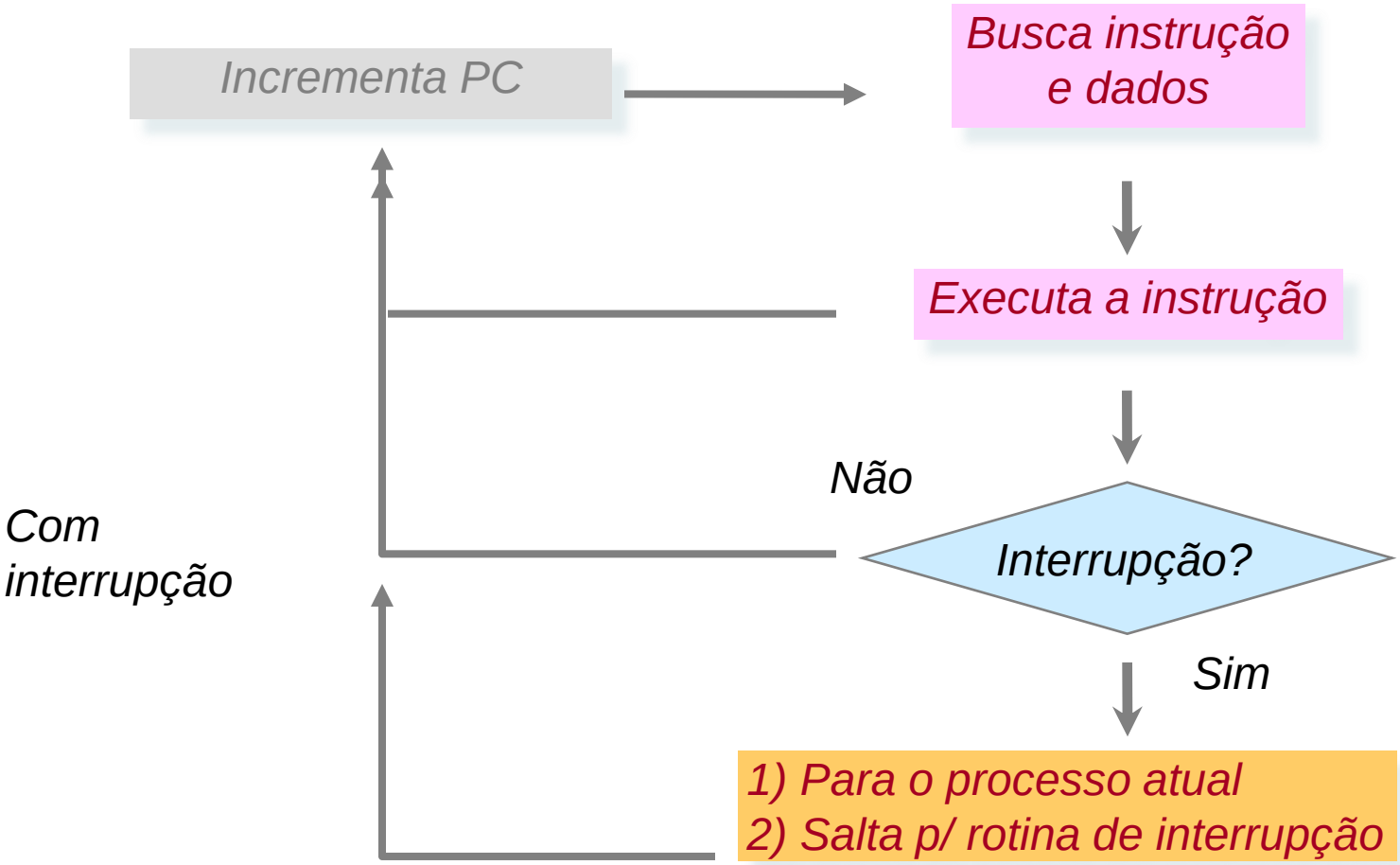
A importância da Interrupção

- Um sistema com interrupção não fica esperando
  - A CPU solicita o *write* e fica executando outras tarefas até ser interrompida pelo disco.

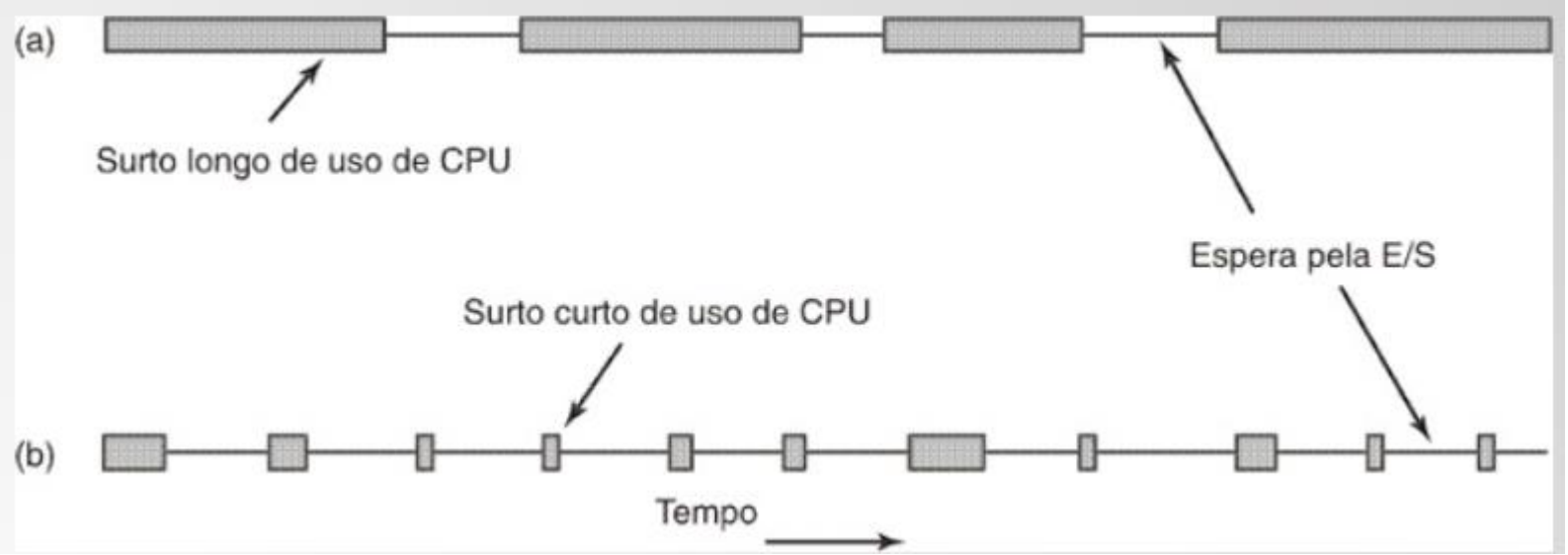
Ex: escrita em disco



Operação Básica da CPU



Processos



Surtos de uso da CPU alternam-se com períodos de espera por E/S

a) um processo orientado à CPU
b) um processo orientado à E/S

Processos

- CPU-bound:
  - Se o processo gasta a maior parte do seu tempo usando a CPU ele é dito orientado à computação (compute-bound ou CPU-bound)
  - processos com longos tempos de execução e baixo volume de comunicação entre processos
    - ex: aplicações científicas, engenharia e outras aplicações que demandam alto desempenho de computação

- I/O-bound:
  - Se um processo passa a maior parte do tempo esperando por dispositivos de E/S, diz-se que o processo é orientado à E/S (I/O-bound)

processos I/O-bound devem ter prioridade sobre processos CPU-bound

Batch (lote) x Interativos

Interrupção do Programa

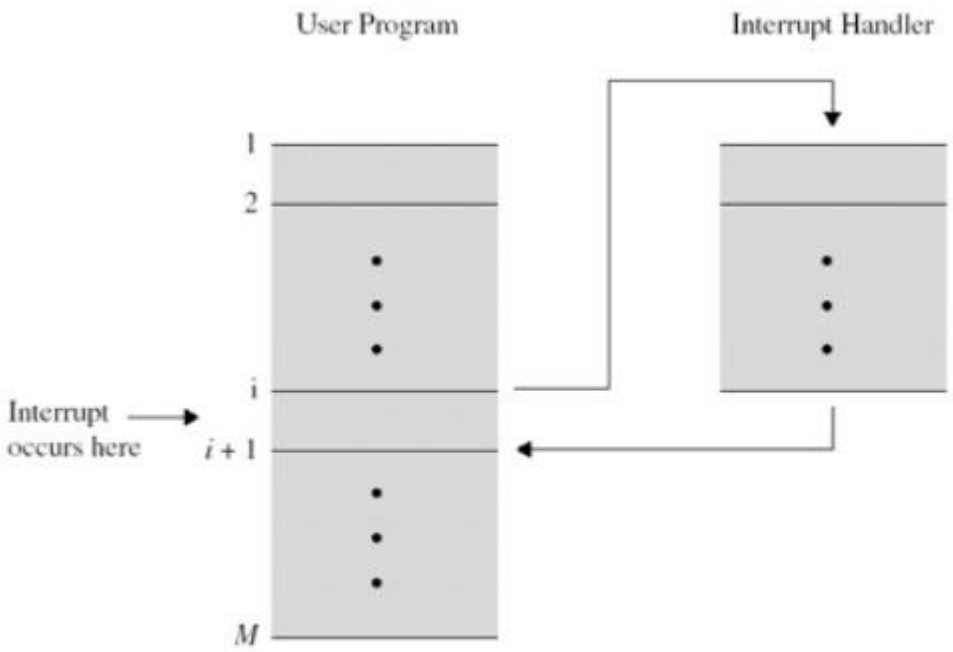
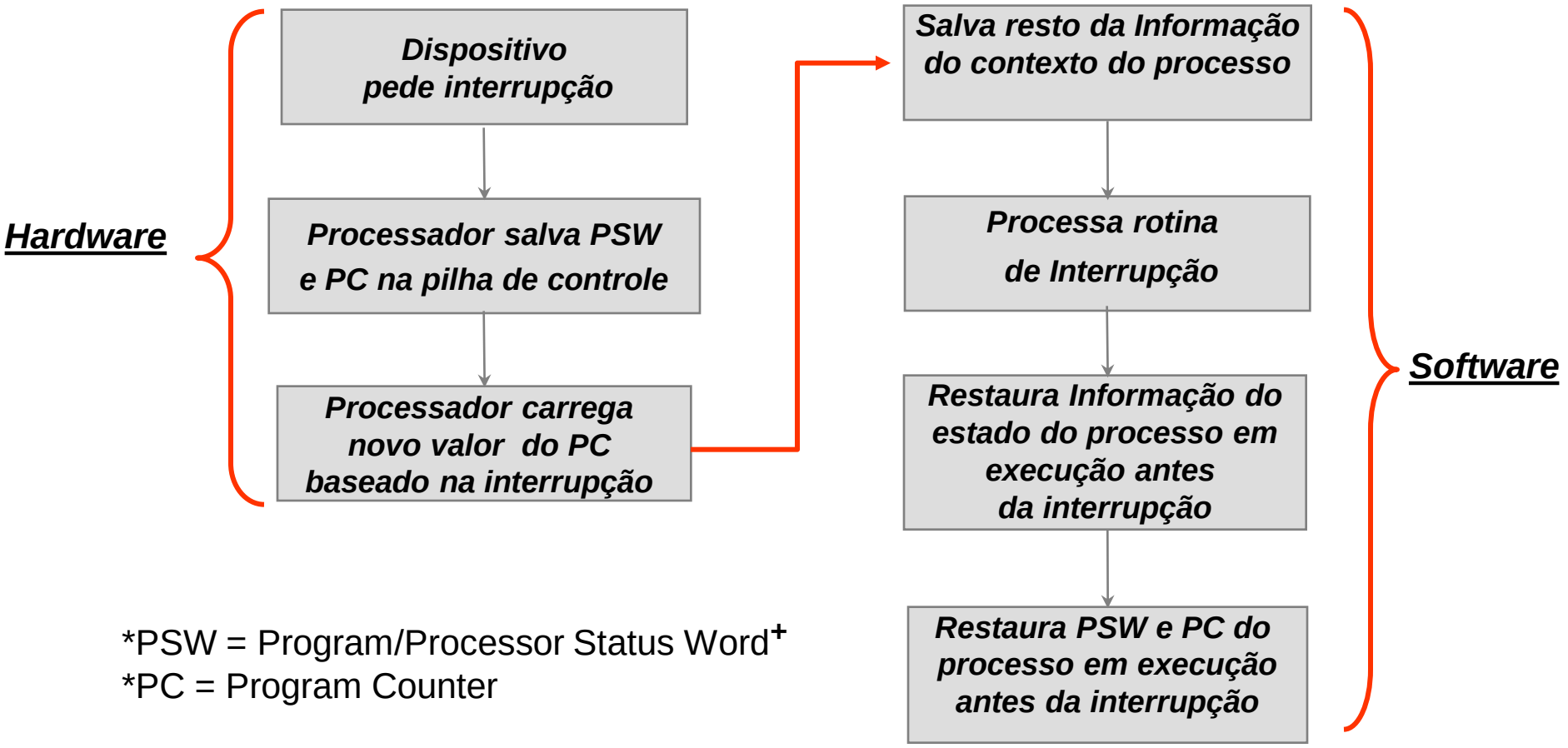


Figure 1.6 Transfer of Control via Interrupts

Processo de Interrupção



*PSW = Program/Processor Status Word+
*PC = Program Counter

+O registrador PSW (palavra de estado do processador) é um "registrador de estado" e passou a ser necessário quando os computadores começaram a utilizar sistemas de interrupção

Algoritmos de Escalonamento

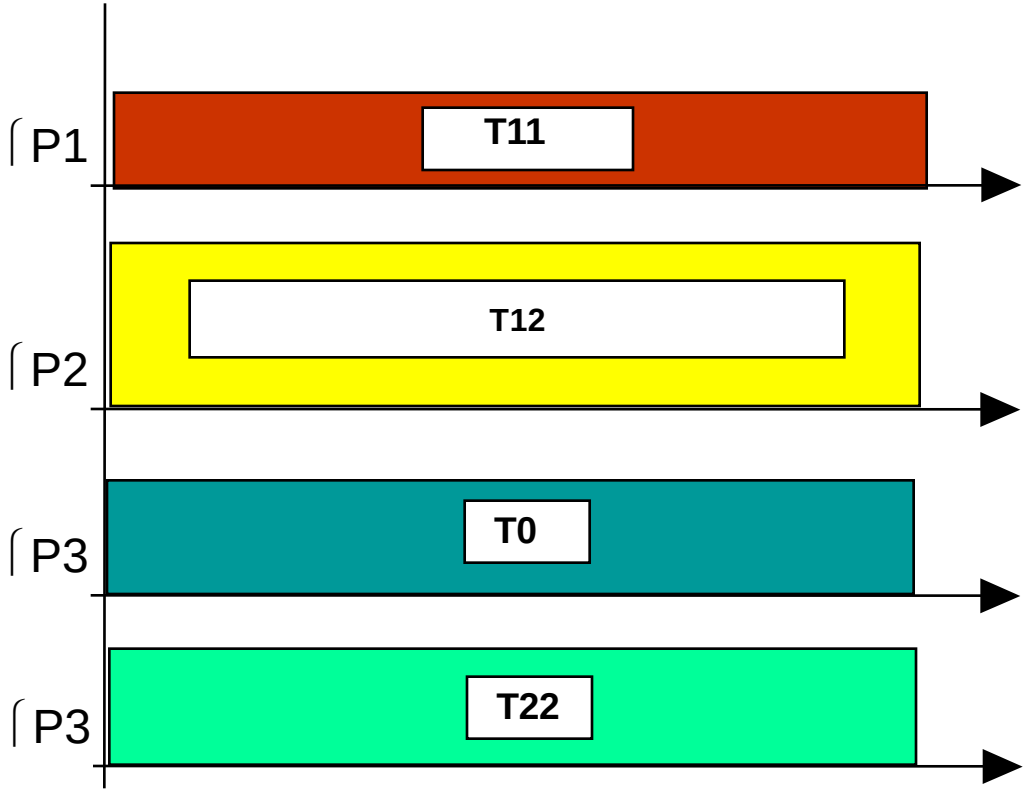
Escalonamento de Processos

- Quando um ou mais processos estão prontos para serem executados, o sistema operacional deve decidir qual deles vai ser executado primeiro

- A parte do sistema operacional responsável por essa decisão é chamada escalonador, e o algoritmo usado para tal é chamado de algoritmo de escalonamento

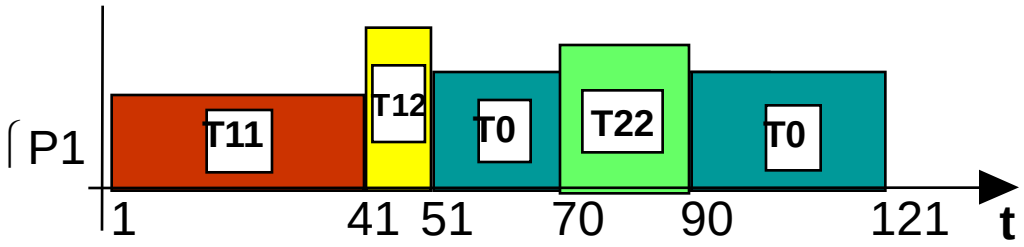
Escalonamento (Abstração)

- Uma máquina para cada processo
- Paralelismo real



Escalonamento (Realidade)

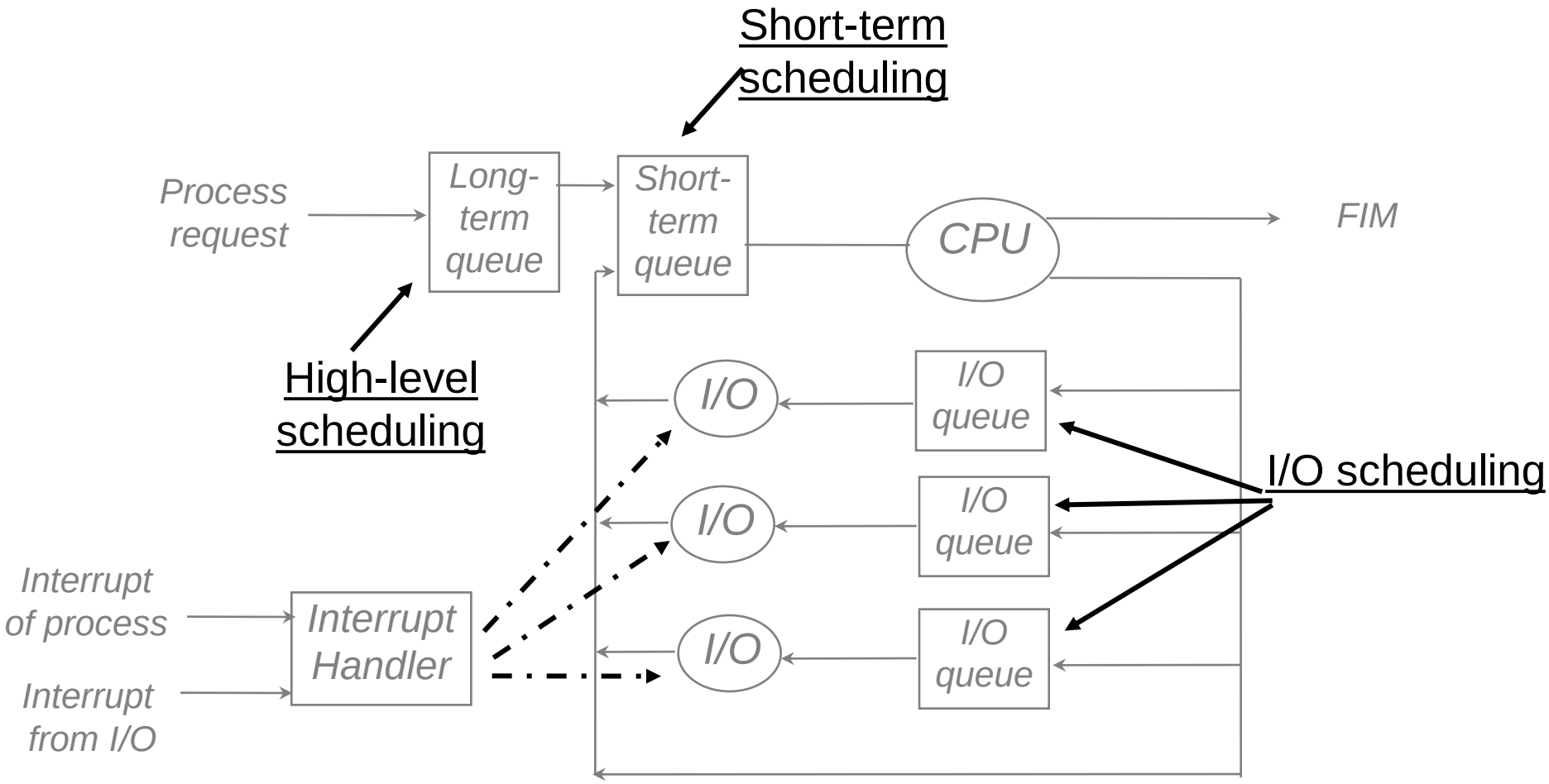
- Compartilhamento do tempo
- Pseudo-paralelismo



Filas/Níveis de Escalonamento

- High-level
  - Decide quantos programas são admitidos no sistema
  - Aloca memória e cria um processo
  - Controla a *long-term queue*
- Short-term
  - Decide qual processo deve ser executado
  - Controla a *short-term queue*
- I/O
  - Decide qual processo (com I/O) pendente deve ser tratado pelo dispositivo de I/O
  - Controla a *I/O queue*

Filas de Escalonamento



Categorias de Escalonamento

- Em lote (batch)
- Interativo
- Tempo-real

Escalonamento (Objetivos)

Todos os sistemas

Justiça — dar a cada processo uma porção justa da UCP
Aplicação da política — verificar se a política estabelecida é cumprida
Equilíbrio — manter ocupadas todas as partes do sistema

Sistemas em lote

Vazão (throughput) — maximizar o número de jobs por hora
Tempo de retorno — minimizar o tempo entre a submissão e o término
Utilização de UCP — manter a UCP ocupada o tempo todo

Sistemas interativos

Tempo de resposta — responder rapidamente às requisições
Proporcionalidade — satisfazer as expectativas dos usuários

Sistemas de tempo real

Cumprimento dos prazos — evitar a perda de dados
Previsibilidade — evitar a degradação da qualidade em sistemas multimídia

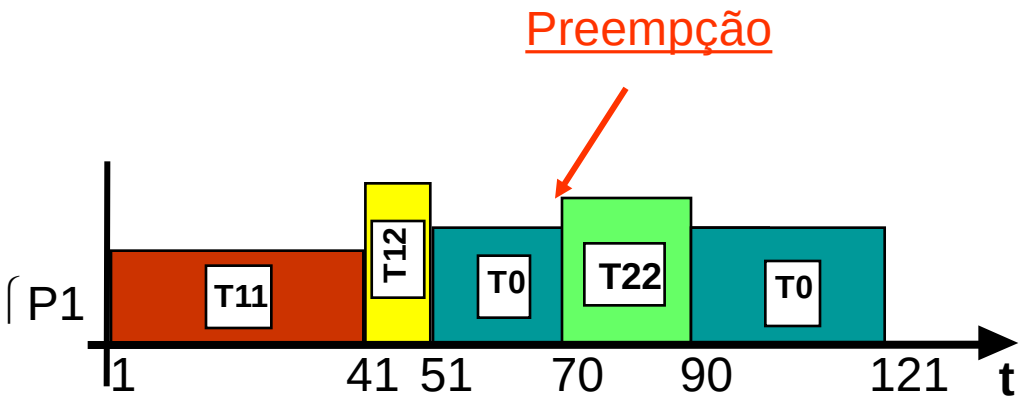
Tipos de Escalonamento

- Mecanismos de Escalonamento
  - Preemptivo x Não-preemptivo
- Políticas de Escalonamento
  - Round-Robin
  - FIFO (First-In First-Out)
  - Híbridos
    - Partições de Lote (Batch)
    - MFQ - Multiple Feedback Queue
  - SJF – Shortest Job First
  - SRJN – Shortest Remaining Job Next

Diz-se que um algoritmo/sistema operacional é preemptivo quando um processo entra na CPU e o mesmo pode ser retirado (da CPU) antes do término da sua execução

Escalonamento Preemptivo

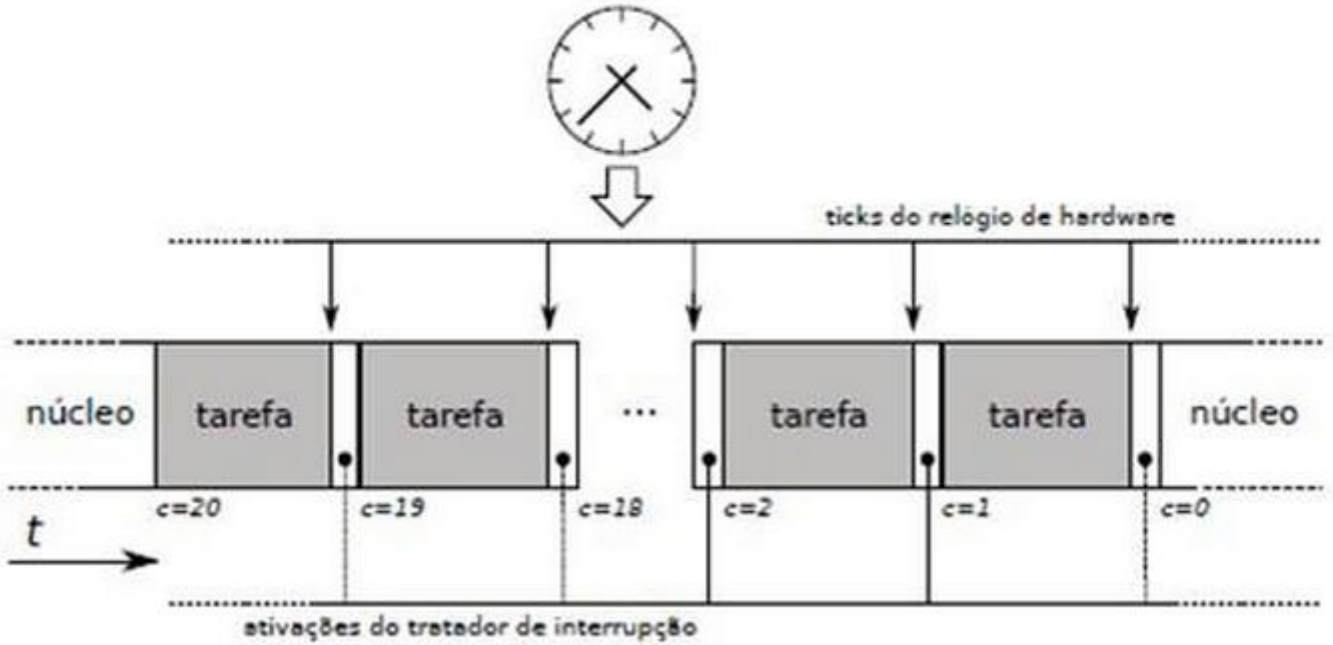
- Permite a suspensão temporária de processos
- *Quantum* ou *time-slice*: período de tempo durante o qual um processo usa o processador a cada vez



Escalonamento Preemptivo (*Quantun*)

Quando uma tarefa recebe o processador, o núcleo ajusta um contador de *ticks* que essa tarefa pode usar, ou seja, seu *quantum* é definido em número de *ticks*.

A cada *tick*, o contador é decrementado; quando ele chegar a zero, a tarefa perde o processador e volta à fila de prontas.



Dinâmica da preempção por tempo (*quantum* igual a 20 *ticks*).

Problema das trocas de processos

- Mudar de um processo para outro requer um certo tempo para a administração — salvar e carregar registradores e mapas de memória, atualizar tabelas e listas do SO, etc
- Isto se chama troca de contexto

- Suponha que esta troca dure 5 ms
- Suponha também que o *quantum* está ajustado em 20 ms
- Com esses parâmetros, após fazer 20 ms de trabalho útil, a CPU terá que gastar 5 ms com troca de contexto. Assim, 20% do tempo de CPU (5 ms a cada 25 ms) é gasto com o *overhead* administrativo...

Solução?

- Para melhorar a eficiência da CPU, poderíamos ajustar o *quantum* para 500 ms
  - Agora o tempo gasto com troca de contexto é menos do que 1% - “desprezível”...

- Considere o que aconteceria se dez usuários apertassem a tecla <ENTER> exatamente ao mesmo tempo, disparando cada um processo:
  - Dez processos serão colocados na lista de processo aptos a executar
  - Se a CPU estiver ociosa, o primeiro começará imediatamente, o segundo não começará cerca de ½ segundo depois, e assim por diante
  - O “azarado” do último processo somente começará a executar 5 segundos depois do usuário ter apertado <ENTER>, isto se todos os outros processos tiverem utilizado todo o seu *quantum*
  - Muitos usuários vão achar que o tempo de resposta de 5 segundos para um comando simples é “muita” coisa

“Moral da estória”

- Ajustar um *quantum* muito pequeno causa muitas trocas de contexto e diminui a eficiência da CPU, ...
- mas ajustá-lo para um valor muito alto causa um tempo de resposta inaceitável para pequenas tarefas interativas

Quantum grande:
Diminui número de mudanças de contexto e *overhead* do S.O., mas...
Ruim para processos interativos

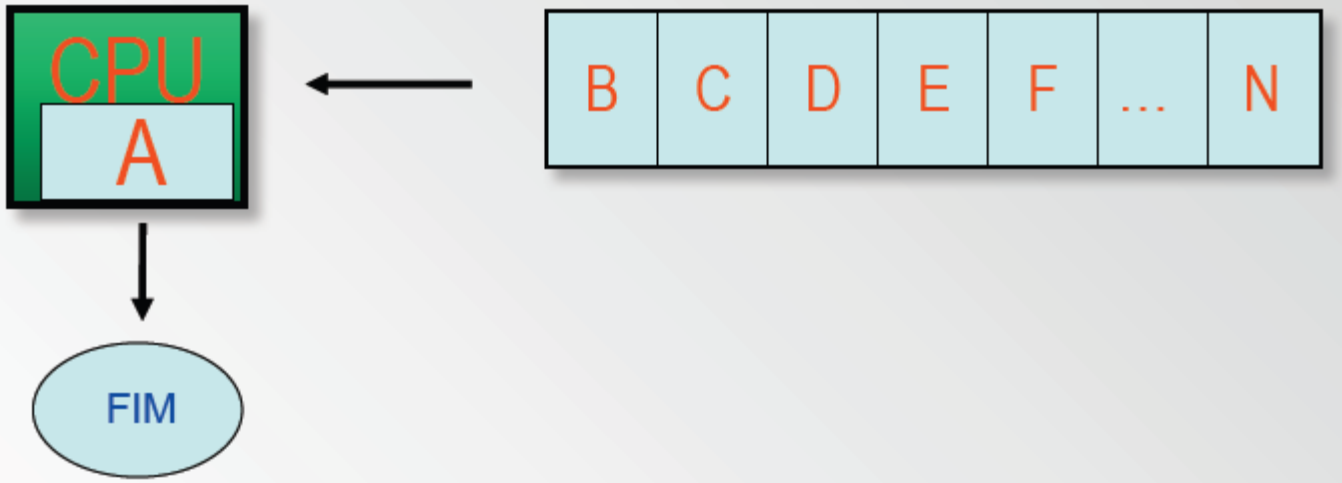
Escalonamento **Não-**Preemptivo

Escalonamento em Sistemas em Lote

- First-come first-served (ou FIFO)
- Shortest Job First (*job mais curto primeiro*) - SJF
- Shortest Remaining Time/Job First - SRTF

Algoritmos (FIFO)

- Uso de uma lista de processos sem prioridade
- Escalonamento não-preemptivo
- Simples e justo
- Bom para sistemas em batch (lote)



Algoritmos (Tempo de Execução)

- Shortest Job First (não-preemptivo)
- Shortest Remaining Job Next (preemptivo)

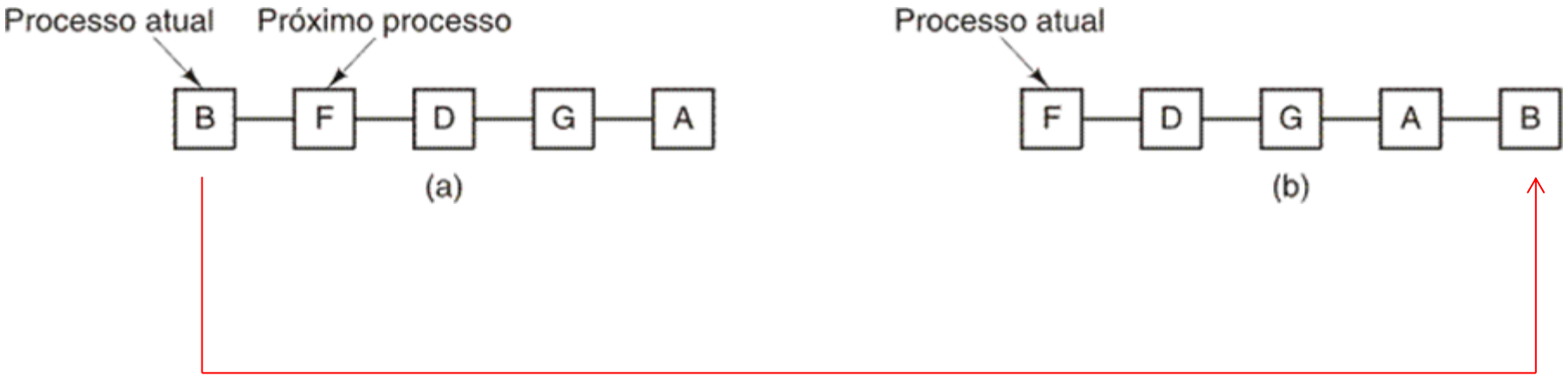
- Melhora o tempo de resposta
- Não é justo: pode causar estagnação (*starvation*)
  - Pode ser resolvida alterando a prioridade dinamicamente

Exemplo de escalonamento *job mais curto primeiro* (*Shortest Job First – SJF*)

Escalonamento em Sistemas Interativos

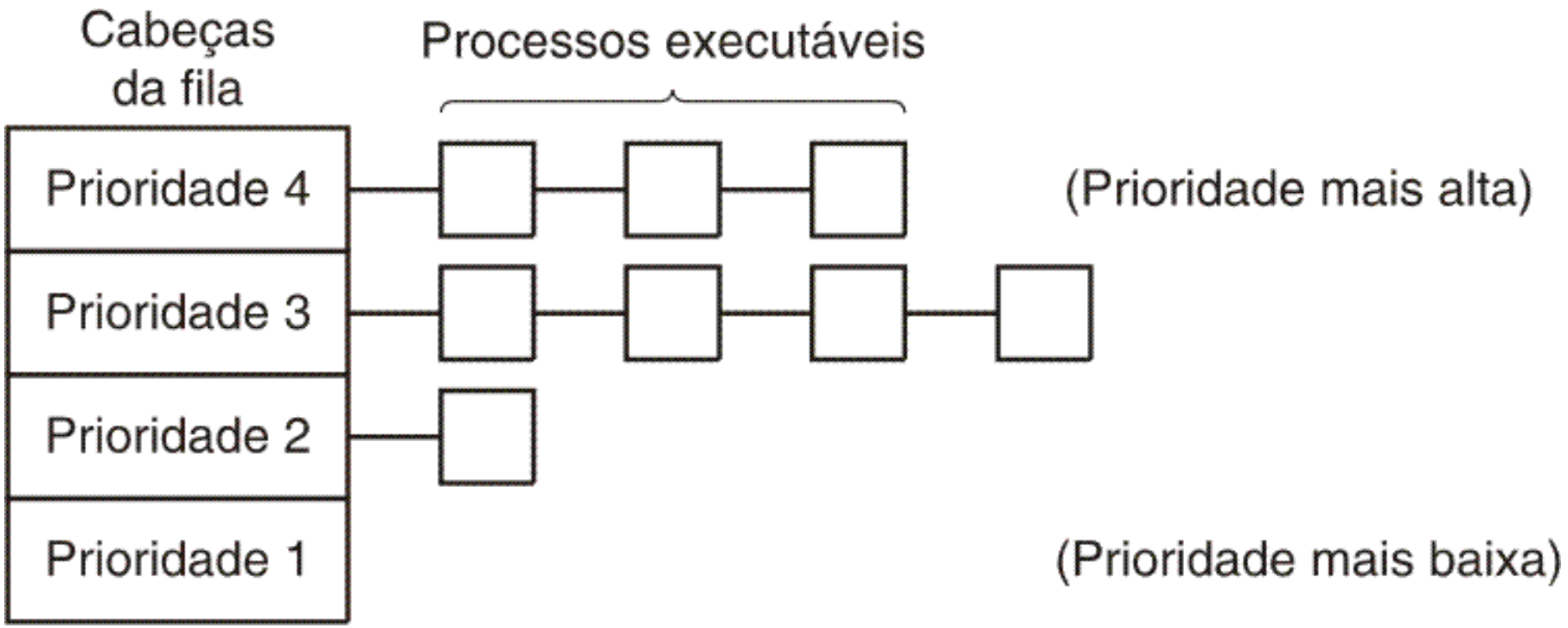
- Round-robin
- Prioridade
- Multiple queues
- Shortest process next
- Guaranteed scheduling
- Lottery scheduling
- Fair-share scheduling (fração justa)

Algoritmos (Round-robin)



- Escalonamento por alternância circular (*round-robin*)
  a) lista de processos executáveis
  b) lista de processos executáveis depois que B usou todo o seu quantum

Algoritmos (Prioridade)



Um algoritmo de escalonamento com quatro classes de prioridade

Bom para sistemas interativos

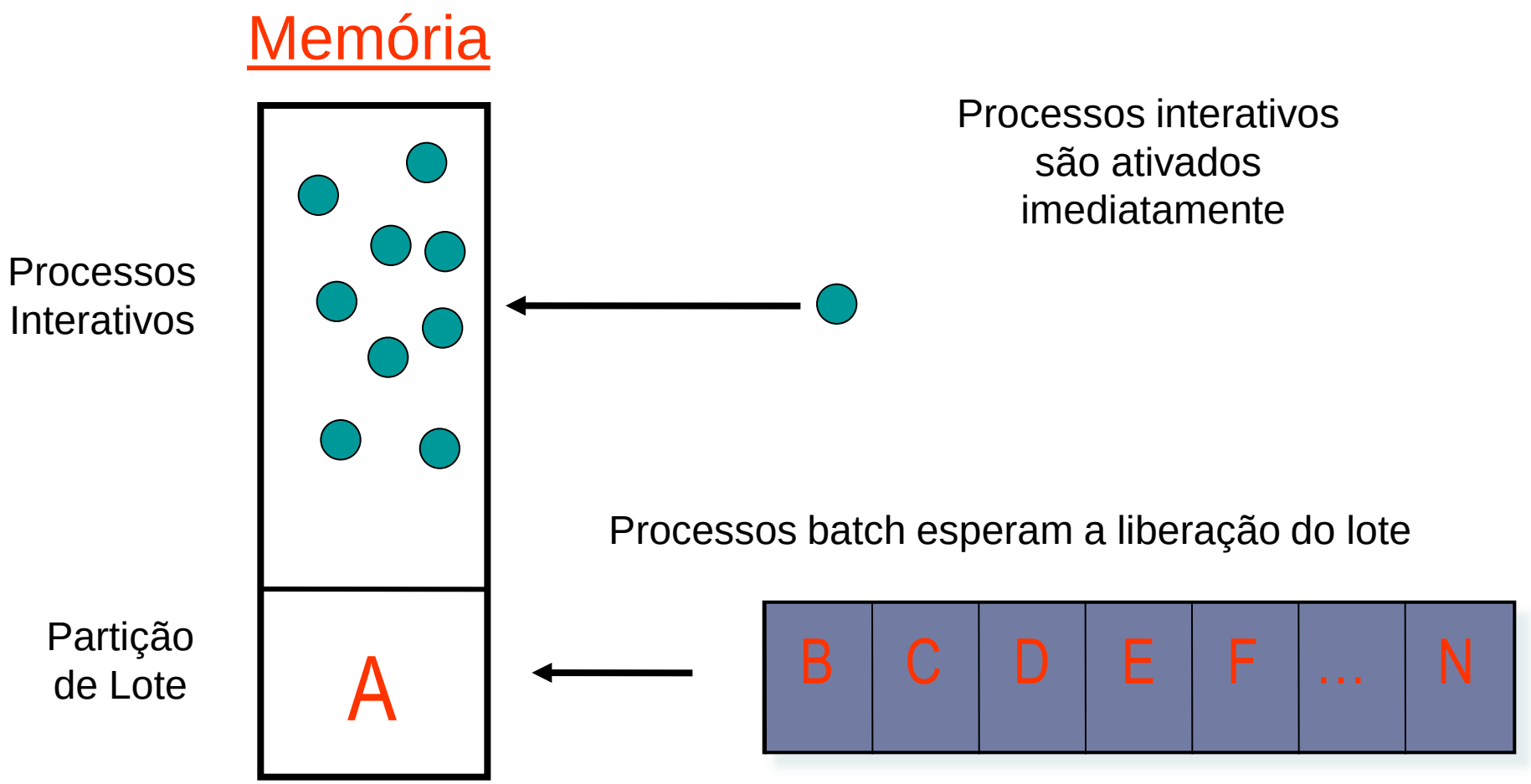
Escalonamento
Híbridos

- Como combinar processos batch com interativos?
- Uso de Partições de Lote (batch)
  - O sistema aceita tantos processos batch quantas forem as partições de lote
  - O sistema aceita todos os processos interativos
  - Escalonamento em dois níveis

Segue...

Centro de Informática UFPE

Escalonamentos Híbridos
Partições de Lote



Escalonamentos Híbridos
Multiple Feedback Queue

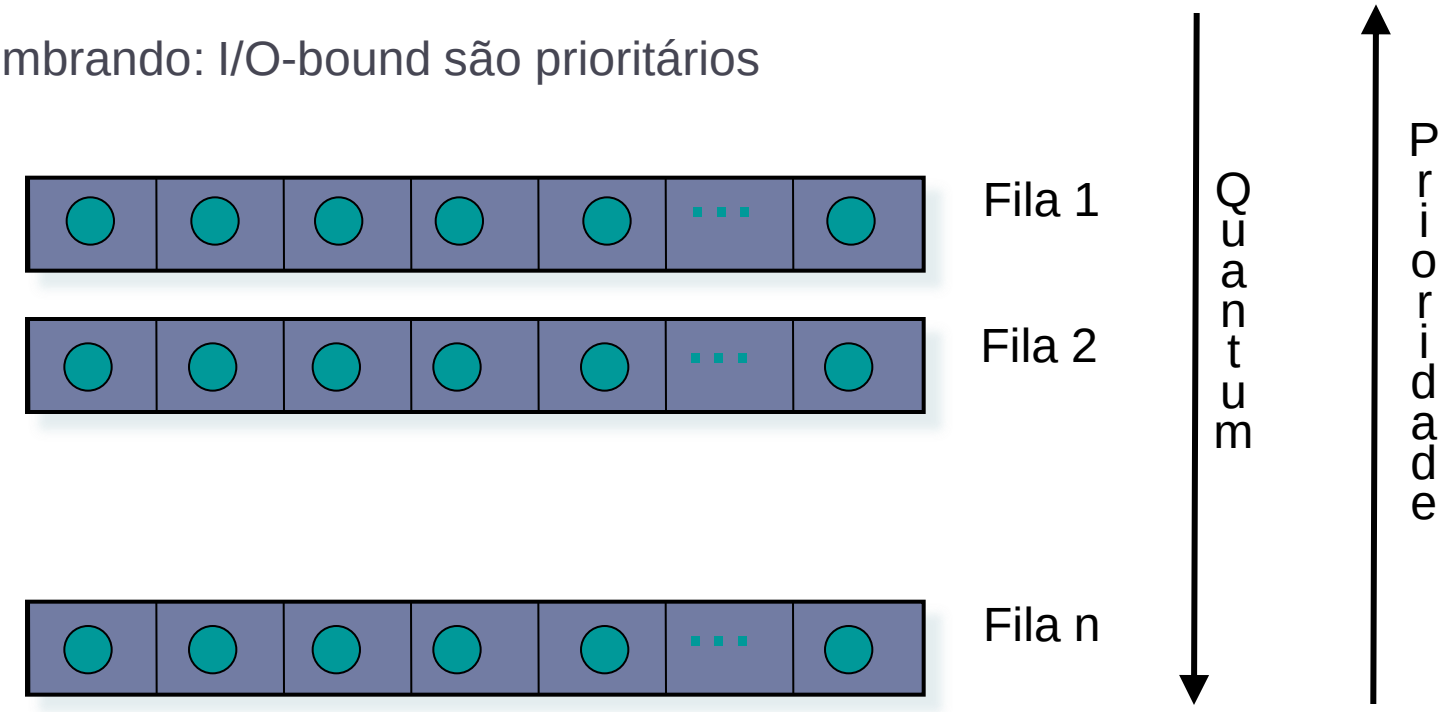
- Como saber *a priori* se o processo é CPU-bound ou I/O-bound?
- MFQ usa abordagem de prioridades dinâmicas
- Adaptação baseada no comportamento de cada processo

Segue…

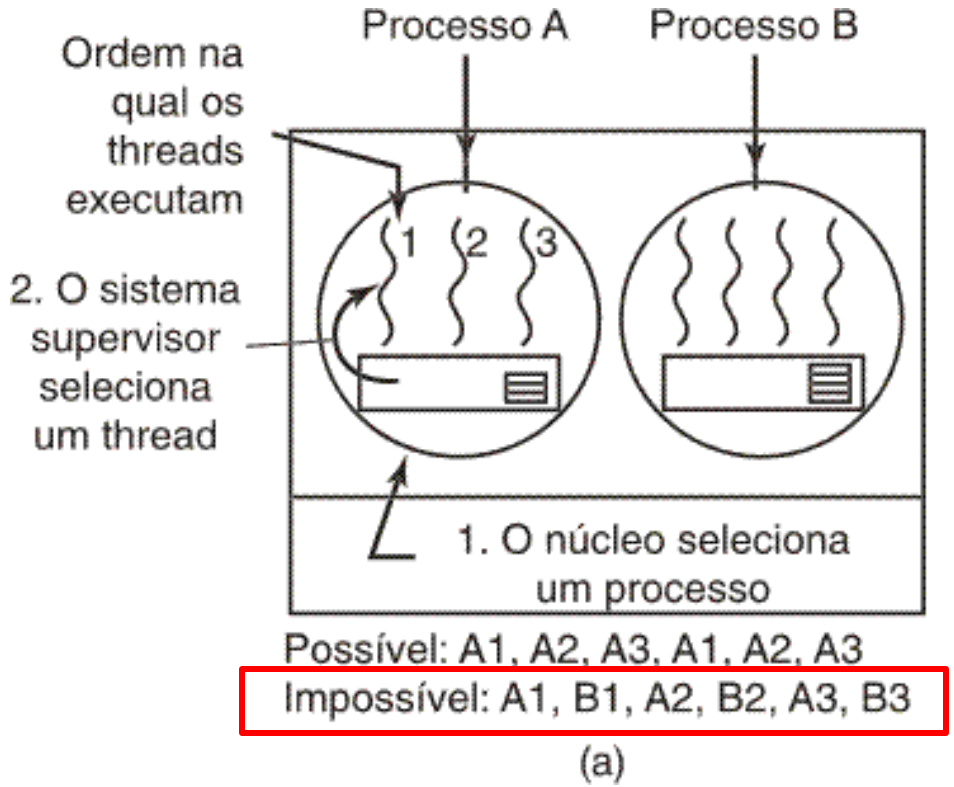
Centro de Informática UFPE

Escalonamentos Híbridos
Multiple Feedback Queue

- Novos processos entram na primeira fila (prioridade mais alta)
- Se acabar o quantum desce um nível
- Se requisitar E/S sobe um nível
  - Lembrando: I/O-bound são prioritários



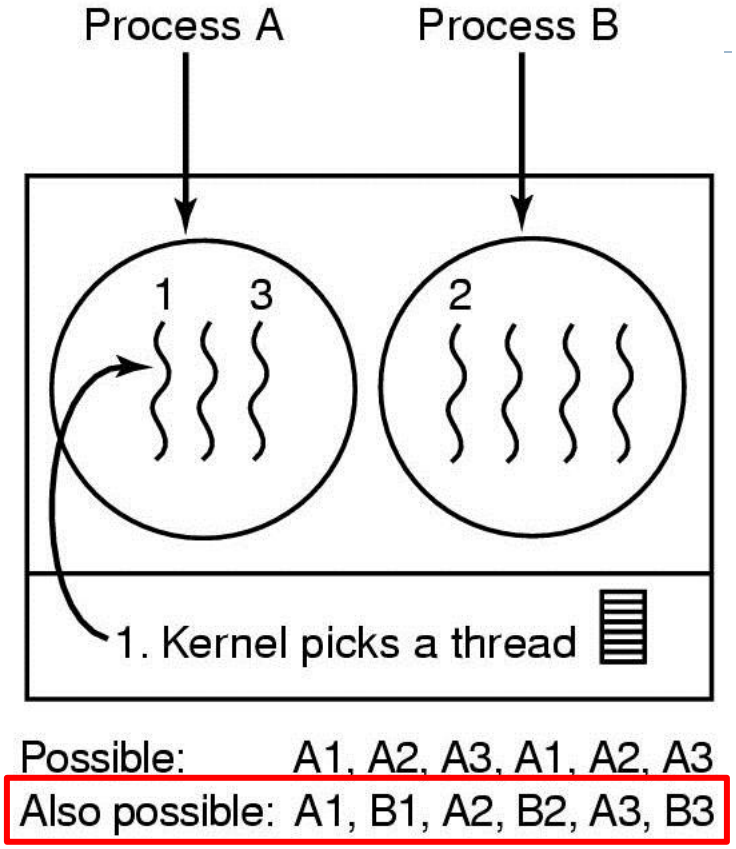
Escalonamento de Threads (1)



Possível escalonamento de threads de usuário

- processo com quantum de 50-mseg
- threads executam 5 mseg por surto de CPU

Escalonamento de Threads (2)



Possível escalonamento de threads de núcleo

- processo com quantum de 50-mseg
- threads executam 5 mseg por surto de CPU

Gerenciamento de Threads

Gerenciamento de Threads

- Retira a responsabilidade do programador de controlar o ciclo de vida de várias threads em execução concorrente.

- ThreadPoolExecutor é a implementação básica desse gerenciador.

Gerenciamento de Threads

- ThreadPoolExecutor
  - É extremamente flexível em relação a sua implementação.
  - Existem 3 configurações padrão obtidas a partir de métodos da Factory Executors.
    - CachedThreadPool
    - FixedThreadPool
    - SingleThreadExecutor
  - Também pode ser configurado manualmente para cenários específicos.

Gerenciamento de Threads

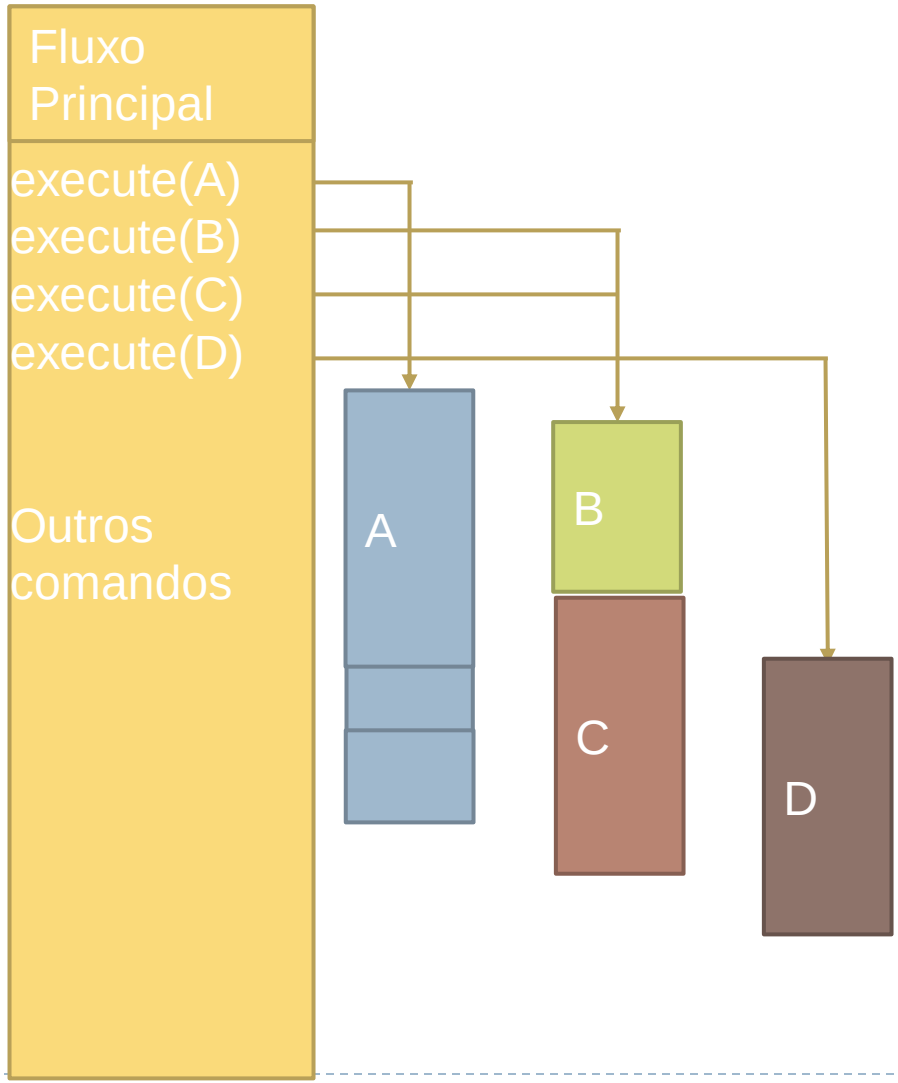
- CachedThreadPool

```
//Criação de um Executor do tipo Cached Thread Pool
ExecutorService cachedPool= Executors.newCachedThreadPool();

//Registra a execução de vários objetos que implementam Runnable
cachedPool.execute( new MessageRunnable("A")  );//leva 20s
cachedPool.execute( new MessageRunnable("B")  );//leva 10s
Thread.sleep(11s);
cachedPool.execute( new MessageRunnable("C")  ); //leva 10s
cachedPool.execute( new MessageRunnable("D")  ); //leva 10s

//Libera as instâncias de thread do pool (Quando terminarem a execução)
cachedPool.shutdown();
```

CachedThreadPool



Gerenciamento de Threads

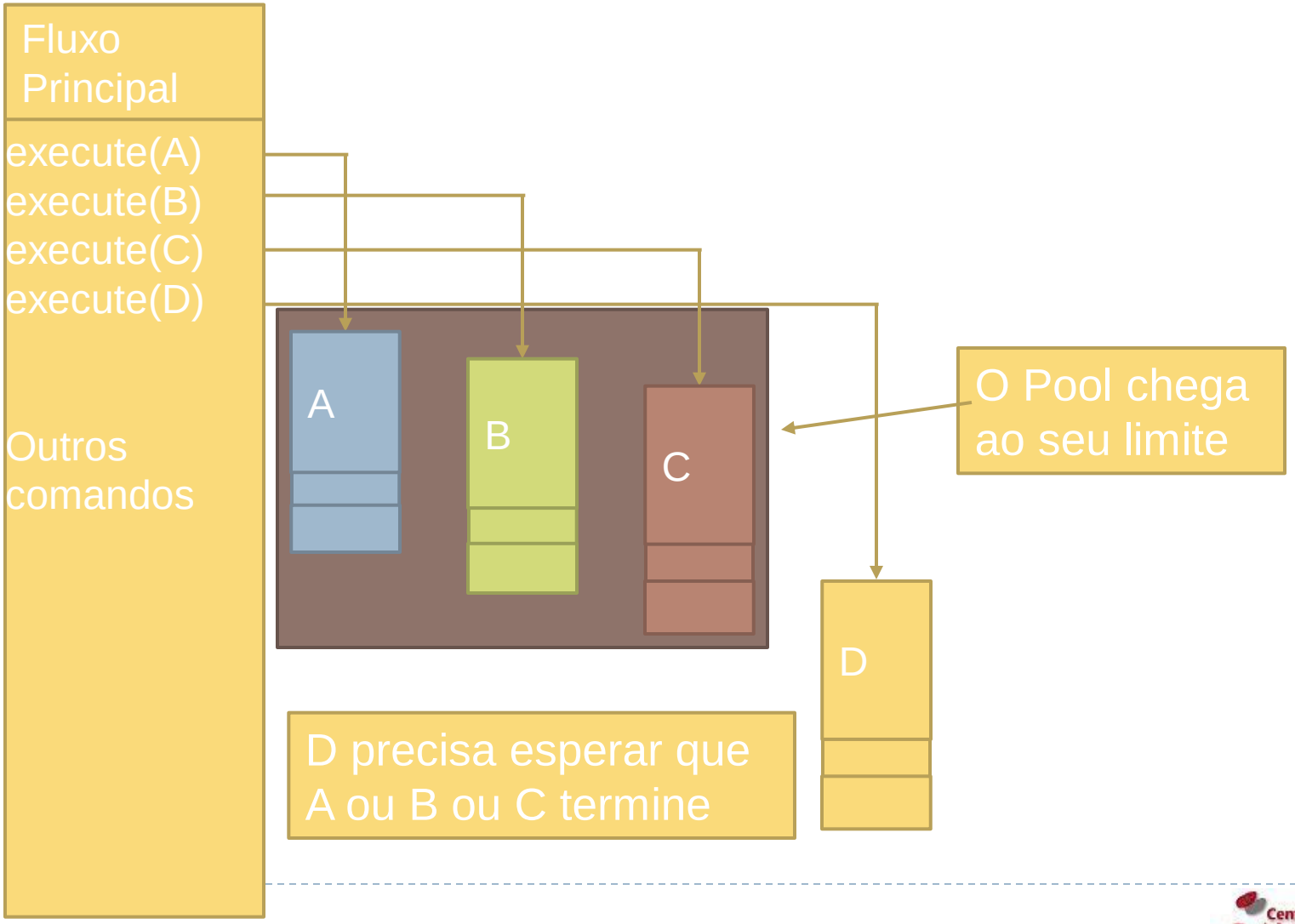
- FixedThreadPool

```
//Criação de um Executor do tipo Fixed Thread Pool
ExecutorService fixedPool= Executors.newFixedThreadPool(3);

//Registra a execução de vários objetos que implementam Runnable
fixedPool.execute( new MessageRunnable("A")  );
fixedPool.execute( new MessageRunnable("B")  );
fixedPool.execute( new MessageRunnable("C")  );
fixedPool.execute( new MessageRunnable("D")  );

//Libera as instâncias de thread do pool (Quando terminarem a execução)
fixedPool.shutdown();
```

FixedThreadPool



Gerenciamento de Threads

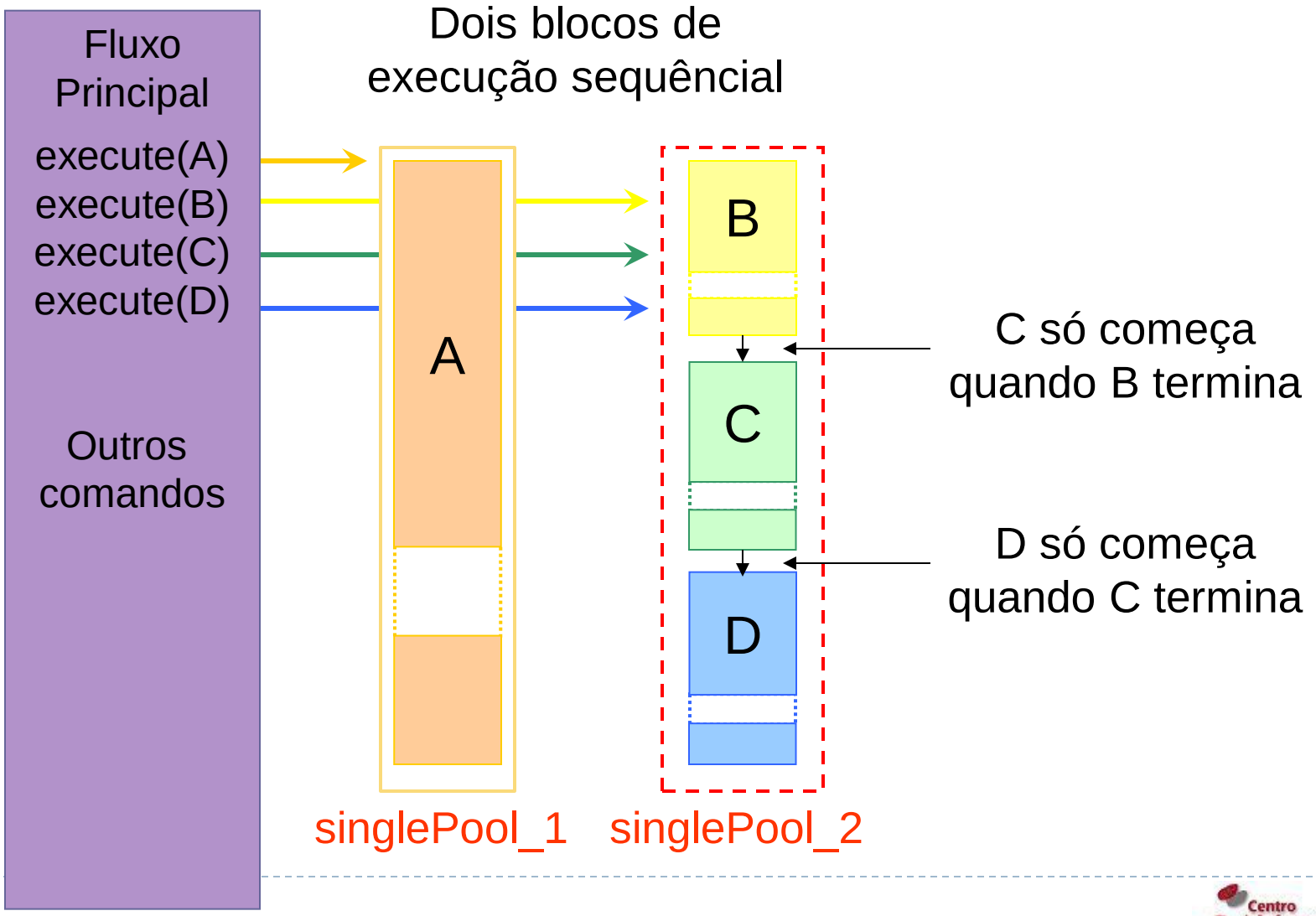
- SingleThreadPool

```
//Criação de um Executor do tipo Fixed Thread Pool
ExecutorService singlePool_1 = Executors. newSingleThreadExecutor();
ExecutorService singlePool_2 = Executors. newSingleThreadExecutor();

//Registra a execução de vários objetos que implementam Runnable
singlePool_1.execute( new MessageRunnable("A") );
singlePool_2.execute( new MessageRunnable("B") );
singlePool_2.execute( new MessageRunnable("C") );
singlePool_2.execute( new MessageRunnable("D") );

//Libera as instâncias de thread do pool (Quando terminarem a execução)
singlePool_1.shutdown();
singlePool_2.shutdown();
```

SingleThreadPool



Sistemas Operacionais
Escalonamento

Carlos Ferraz (cagf@cin.ufpe.br)
Jorge Cavalcanti Fonsêca (jcbf@cin.ufpe.br)