

SISTEMAS OPERATIVOS



ANDROID



INFORMÁTICA DE GESTÃO

Professor Doutor Ricardo Vardasca, PhD, ASIS, FRPS
ricardo.vardasca@islasantarem.pt

ISLA

Santarém

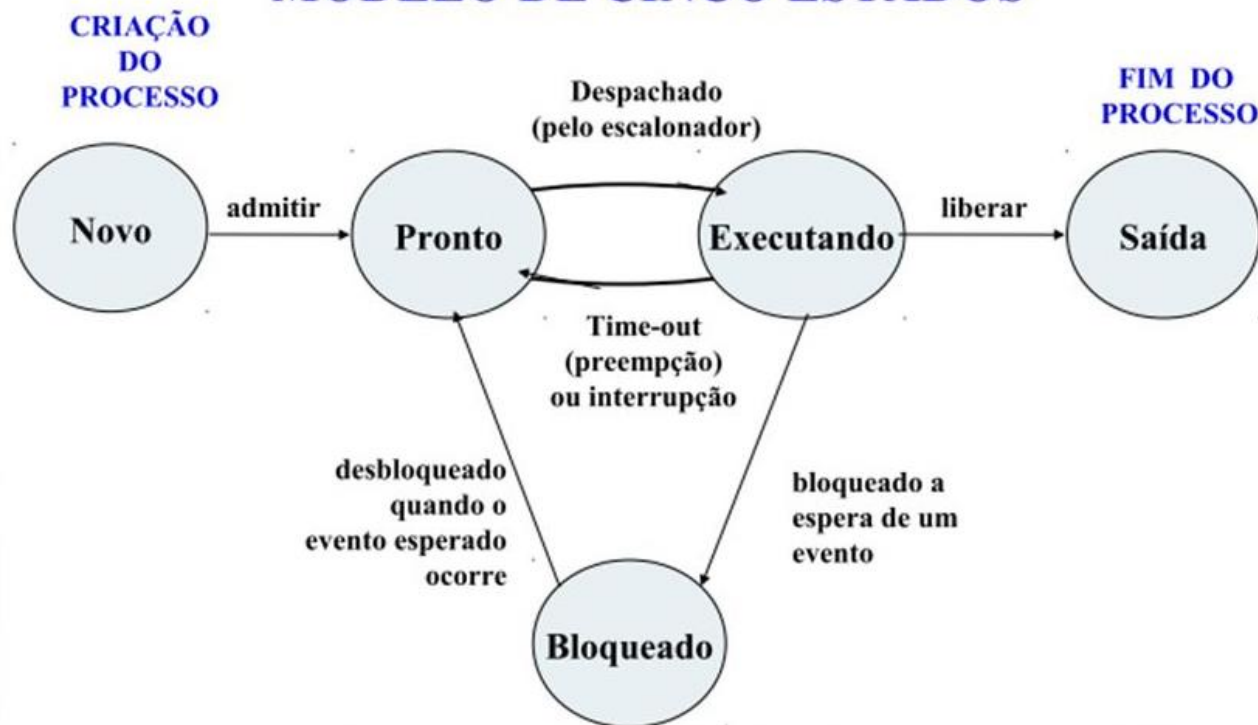
04 – Gestão de Processos

- Estruturas de um SO
- Arquitecturas de SO:
 - Sistemas monolíticos, Sistemas em camadas, Sistemas micro-núcleo, Modelo Cliente-Servidor, Máquinas virtuais, *Containers*, Exonúcleo e Uninúcleo.
- Introdução a Processos
 - Threads vs Processos
 - Características dos Processos
 - Estados dos Processos

- Gestão de processos
 - Estratégias de escalonamento
 - Execução concorrente: problemas e soluções

Estados de um processo

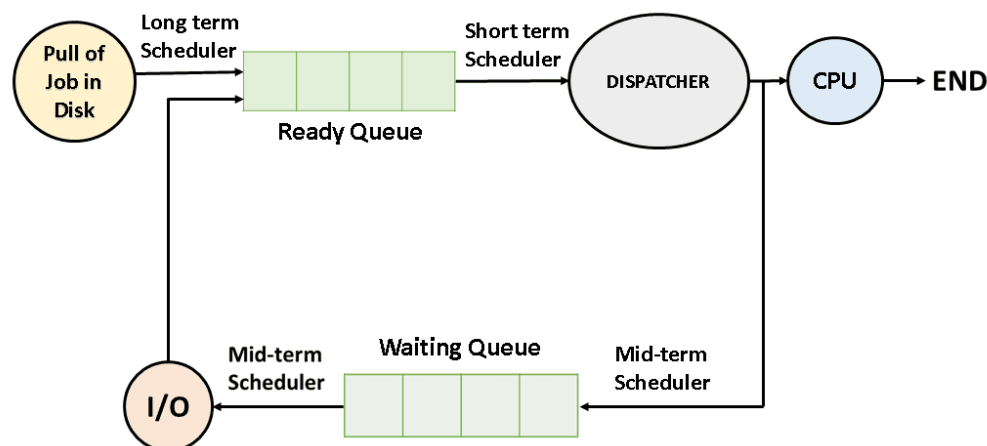
MODELO DE CINCO ESTADOS



Notas:

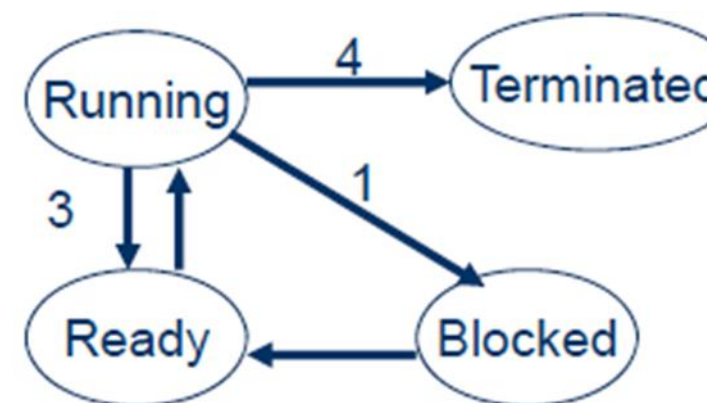
- “**Waiting**” é também conhecido como “**Blocked**”
- Os processos nos estados “**ready**” (pronto) e “**blocked**” (bloqueado) não consomem processador (CPU)

- O escalonador de Processos escolhe o processo que será executado pelo processador
- O escalonamento é realizado com o auxílio do hardware
- O escalonador deve-se preocupar com a eficiência do processador, pois a forma como é definido a execução dos processos é complexa e de grande custo: afeta desempenho do sistema e satisfação do utilizador



- Escalonamento não preemptivo

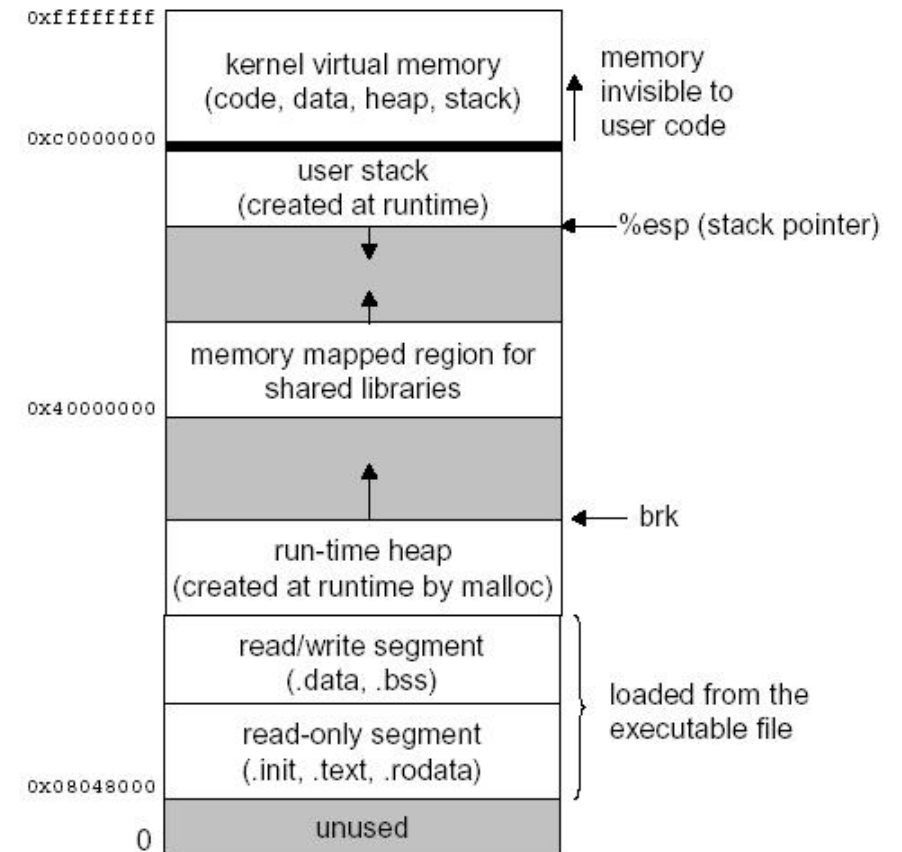
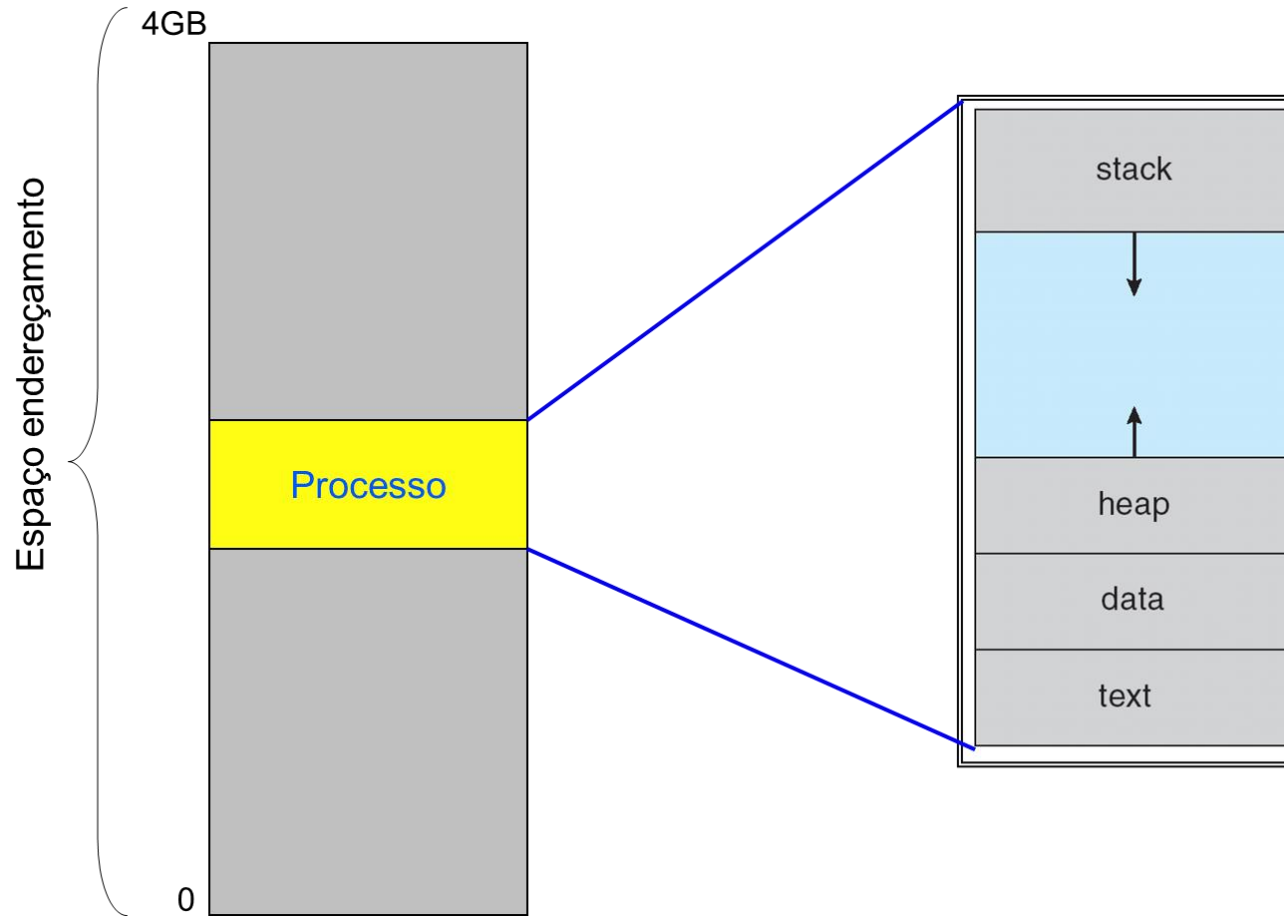
- O processo em execução mantém o processador até que **voluntariamente** o liberta
 - O processo sai
 - Muda para o estado bloqueado
 - 1 e 4 apenas (só iria para 3 se acontecerem chamadas)



- Escalonamento preemptivo

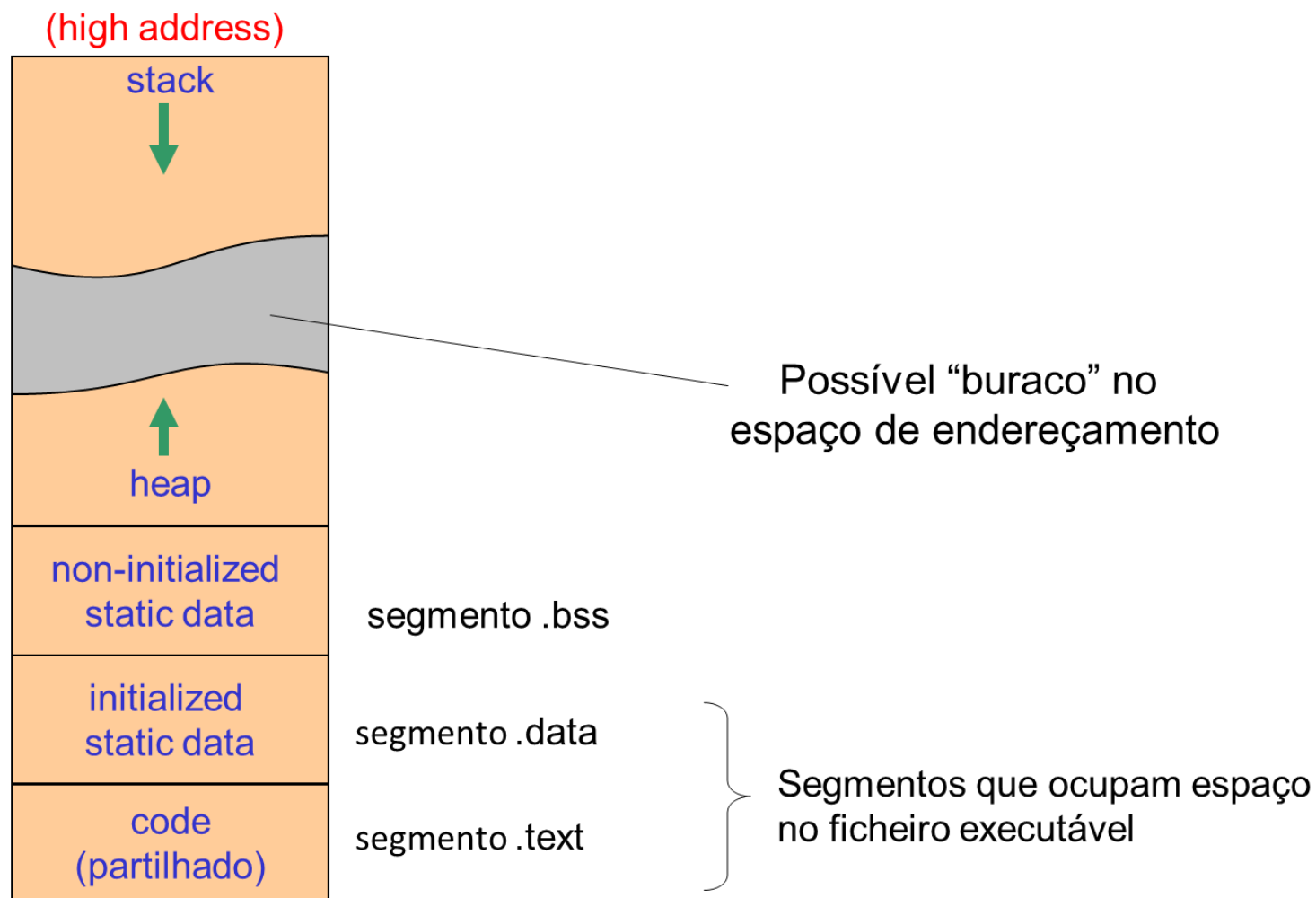
- O processo em execução pode ser interrompido e libertar o processador

Processo em memória



- Imagem do processo
 - Representação do processo na memória
 - Divide-se em vários segmentos
- Segmento “text”
 - Contém o código do processos (instruções que são executadas)
 - Identificada com a secção “.text”
- Segmento “data”
 - Contém variáveis, dividindo-se em duas secções
 - .data: variáveis globais e estáticas (“static”) iniciadas valor diferente de zero
 - .bss: variáveis globais e estáticas não iniciadas (valor zero)
- Segmento “heap”
 - Zona da memória dinâmica (ou alocada, e.g. através do “malloc()”)
- Segmento “stack”
 - Zona das variáveis “automáticas” (ou “locais”)
 - Variáveis que são automaticamente criadas e destruídas nas funções e métodos
 - Cresce “para baixo” (endereços mais altos para os endereços mais baixos)

Imagem de um processo



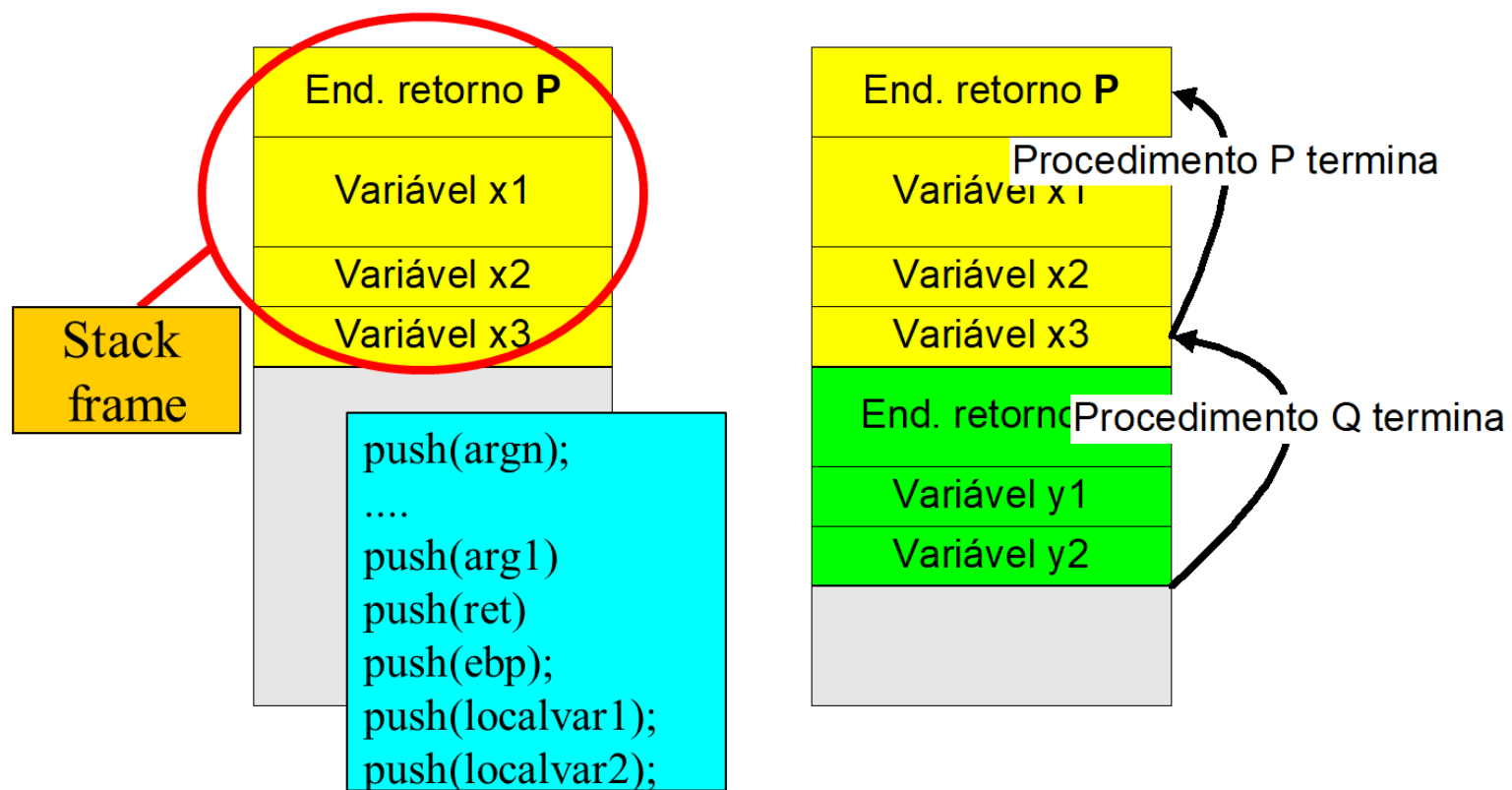
O segmento de pilha serve para:

- Passagem de parâmetros em procedimentos e funções
- Endereço de retorno de procedimentos e funções
- Variáveis automáticas
 - As variáveis locais de um procedimento ou função são:
 - criadas no segmento de pilha no início do procedimento ou função
 - destruídas no fim do procedimento ou função

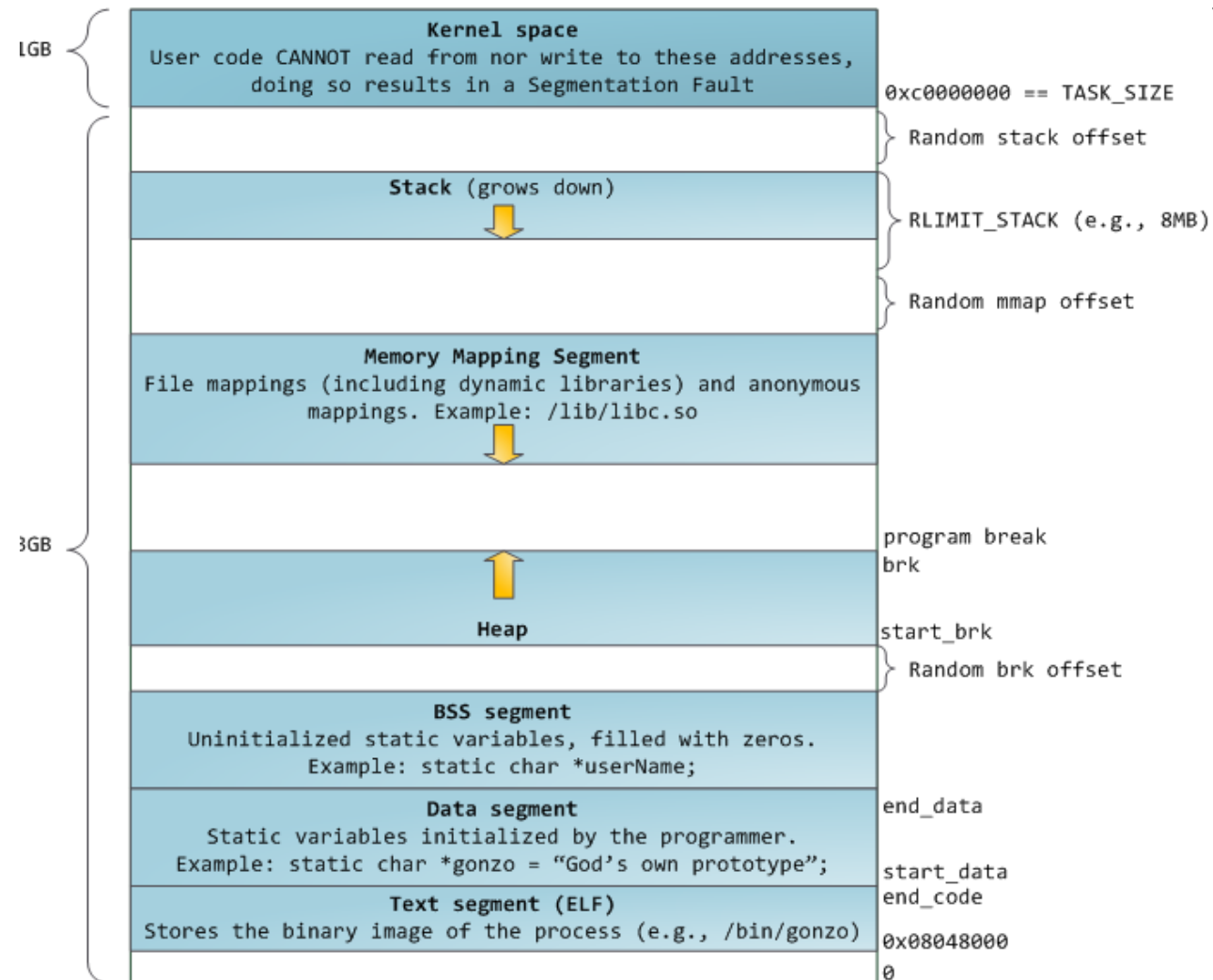
Utilidade do segmento de pilha

Exemplo

- Chamada dos procedimentos **P** e **Q**

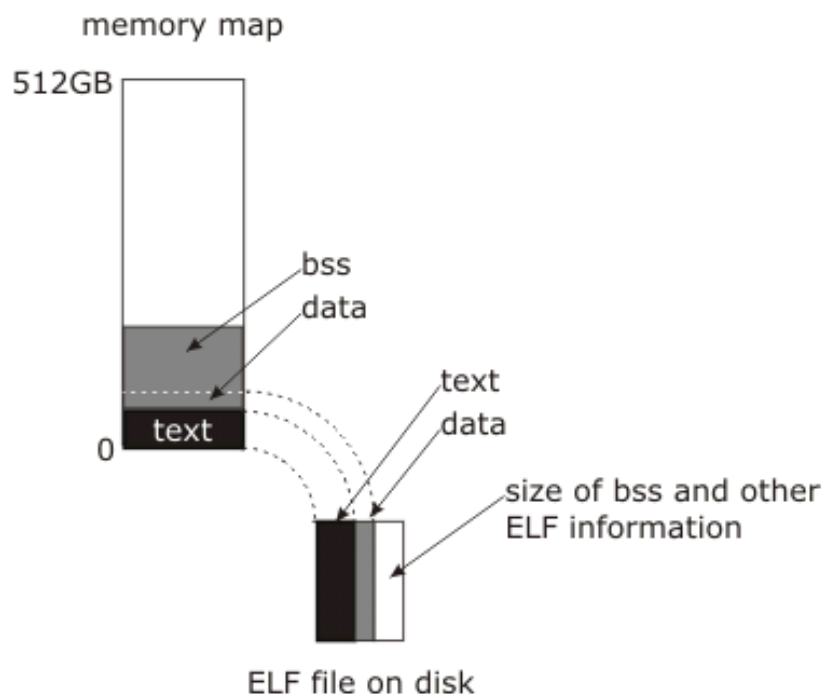


Código mal comportado... (*stack overflow*)



Mapeamento código fonte -> imagem

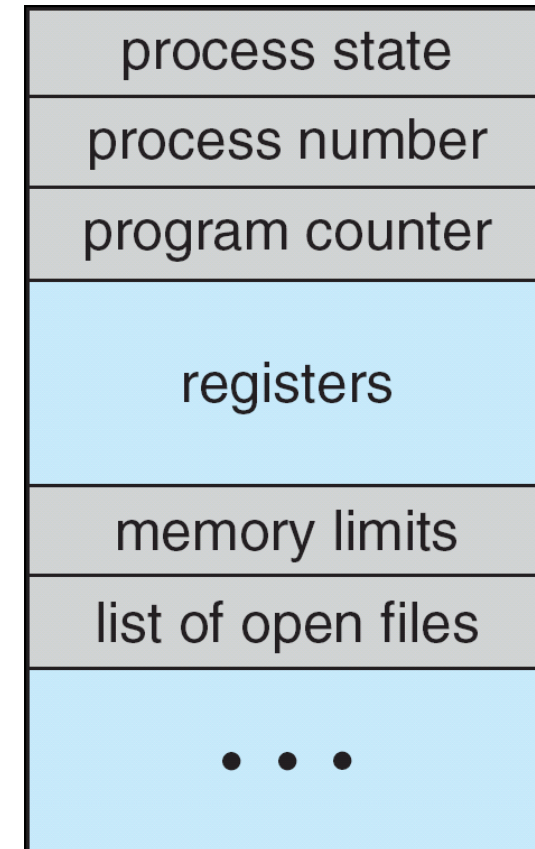
- BSS: Block Started by Symbol
 - Segmento de dados que contém as variáveis “static” e “globais” iniciadas a zero
 - Não ocupa espaço no ficheiro executável (EXE)



- Segmento “data”
 - Variáveis “static” e “globais” iniciadas com valor diferente de zero
 - Existe o segmento “data” no ficheiro executável

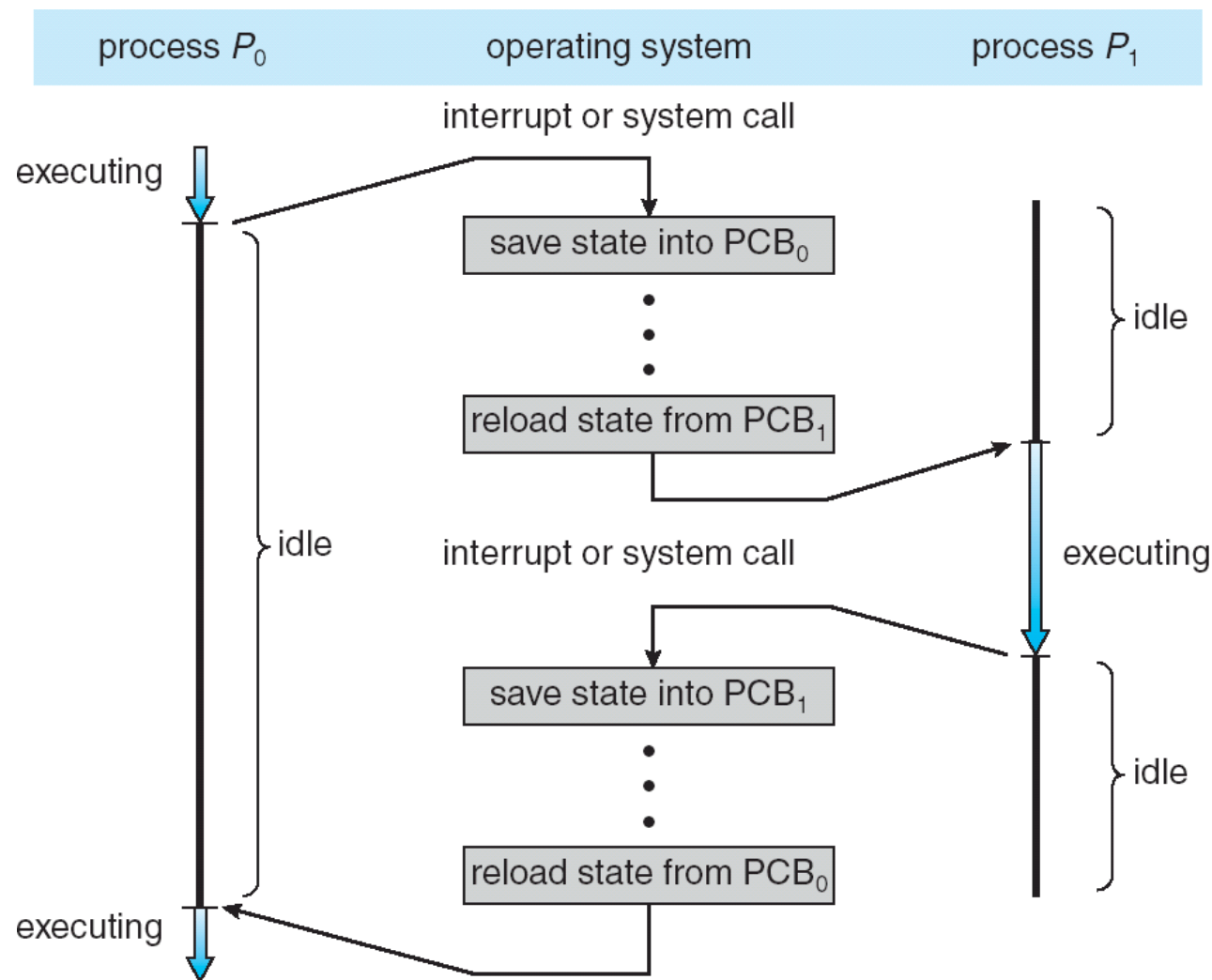
PCB – Process Control Block

- Uma das mais importantes estruturas do SO
 - Representa um processo
- O PCB contém:
 - Identificador processo (PID)
 - Estado processo
 - Contador programa (“PC”)
 - Registos da CPU
 - Informação escalonamento CPU
 - Informação gestão memória
 - Dados contabilísticos
 - Informação estado E/S
 - Tabela ficheiros abertos
 - ...



Comutação de processos

- Comutação do processo “P0” para o processo “P1”
 - “P0” passa de execução para “ready”
 - “P1” passa de “ready” para execução



- **Dependentes de E/S (“I/O bound”)**

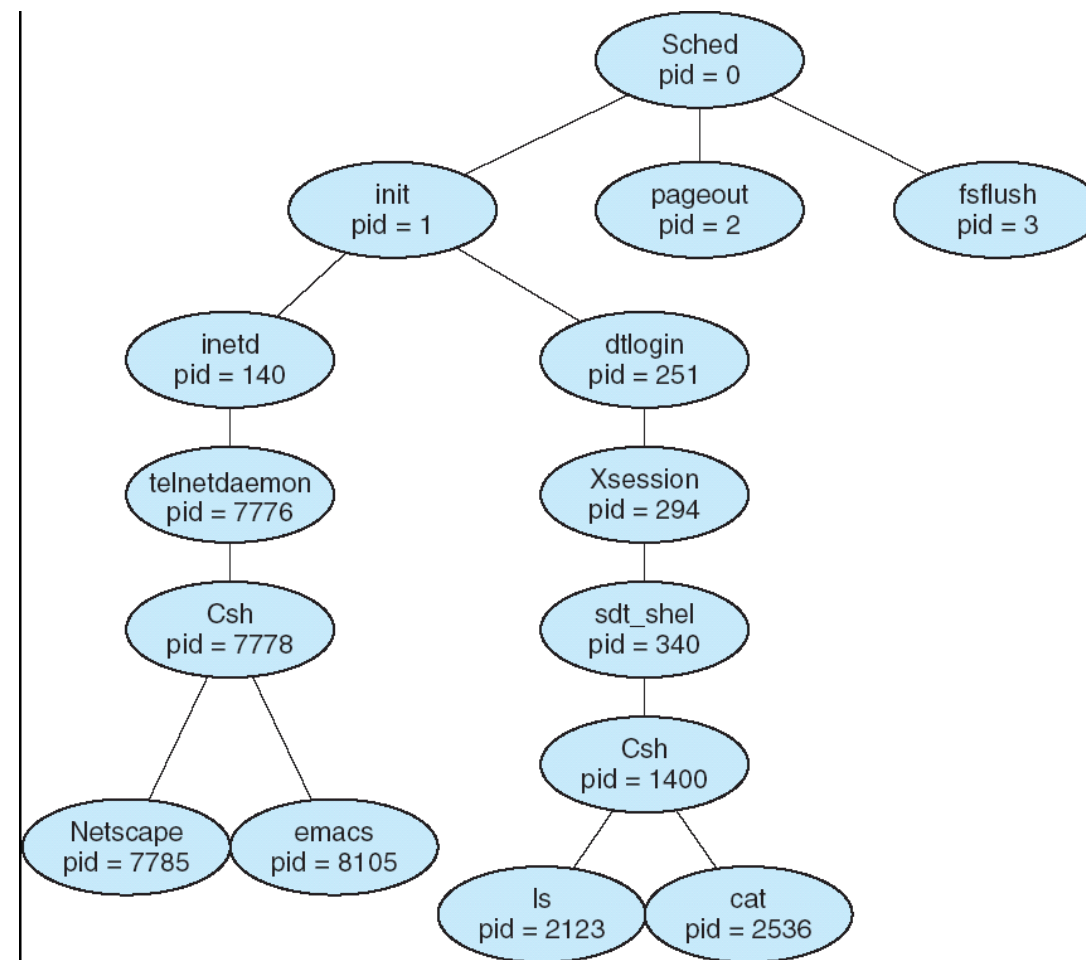
- Executam muitas operações de E/S e pouca computação
 - Processos E/S passam a maior parte do tempo na fila de processos bloqueados
 - Exemplos (aplicações q manipulam ficheiros/rede, etc.)
 - “find”, base de dados (tb podem ser CPU bound – sort), servidor de ficheiros, servidor WEB, etc.

- **Dependentes de CPU (“CPU bound”)**

- Executam muitas operações de computação “pura”
 - Processos CPU passam a maior parte do tempo à espera da CPU
- O tempo de execução do processo está dependente da velocidade da CPU (e tb da memória RAM)
 - Com uma CPU mais rápida o processo executará mais rapidamente
 - Exemplos: cálculo científico, renderização, etc.

Criação de processos

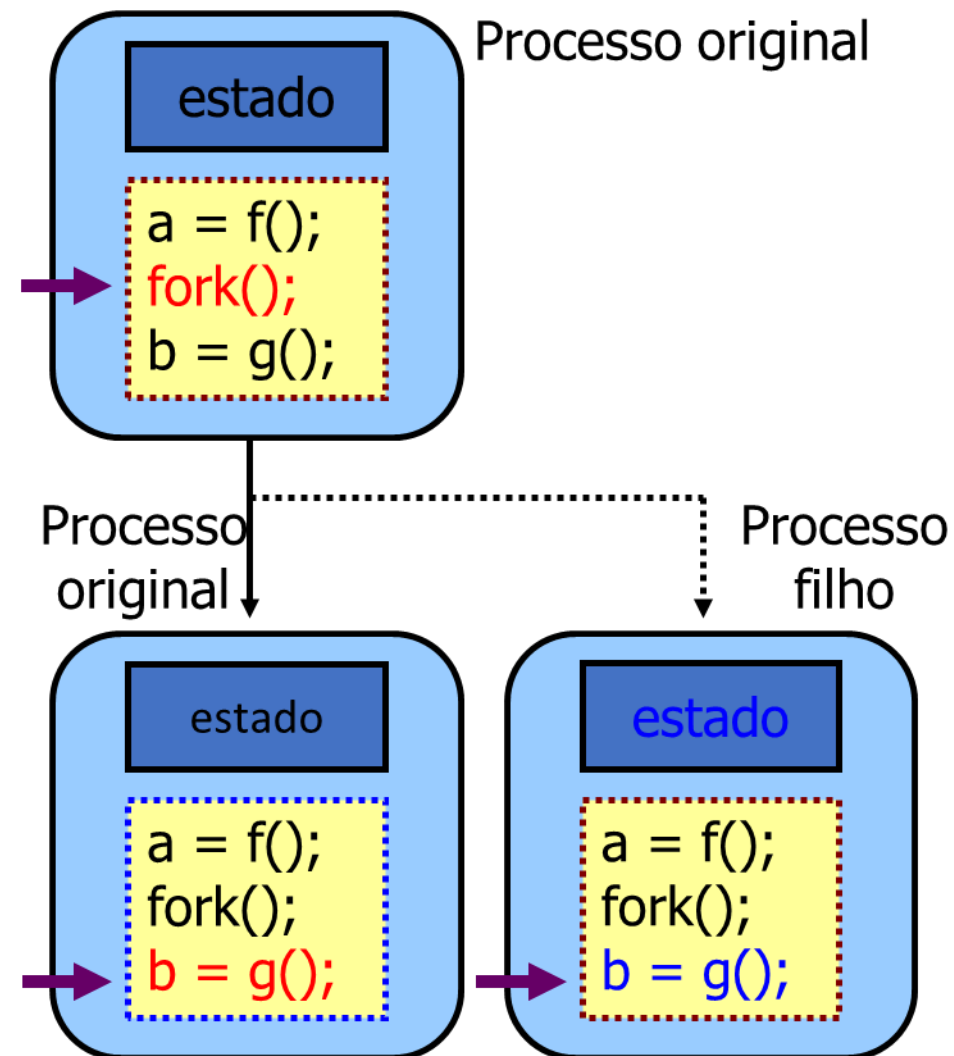
- Conceito de processo “pai”/”filho”
- Um processo (“pai”) cria um outro processo (“filho”) que por sua vez pode dar origem a outros processos
- Forma-se uma hierarquia (árvore) de processos



- A forma de partilha de recursos entre processos pai e filho(s) varia de SO para SO
- Partilha de recursos pode ser um dos seguintes...
 - Processo pai e filho(s) partilham todos os recursos
 - Filhos partilham parte dos recursos do processo pai
 - Processo pai e filho(s) não partilham recursos
- Modelo de execução pode ser...
 - Pai e filho executam concorrentemente
 - Pai espera até que os filhos terminem
 - Filho terminam se o processo pai terminar
 - Término em cascata

Modelo de processos no UNIX

- “fork()”
 - Chamada ao sistema do Unix para criação de um processo filho
- Processos filhos “herdam” todas as características do processo pai
 - Variáveis, contador de programa, ficheiros abertos, etc.
- Após a criação do processo filho (“fork”), cada processo executa com diferentes variáveis e estados
- Alteração de uma variável no processo filho não afeta o processo pai (e vice-versa)



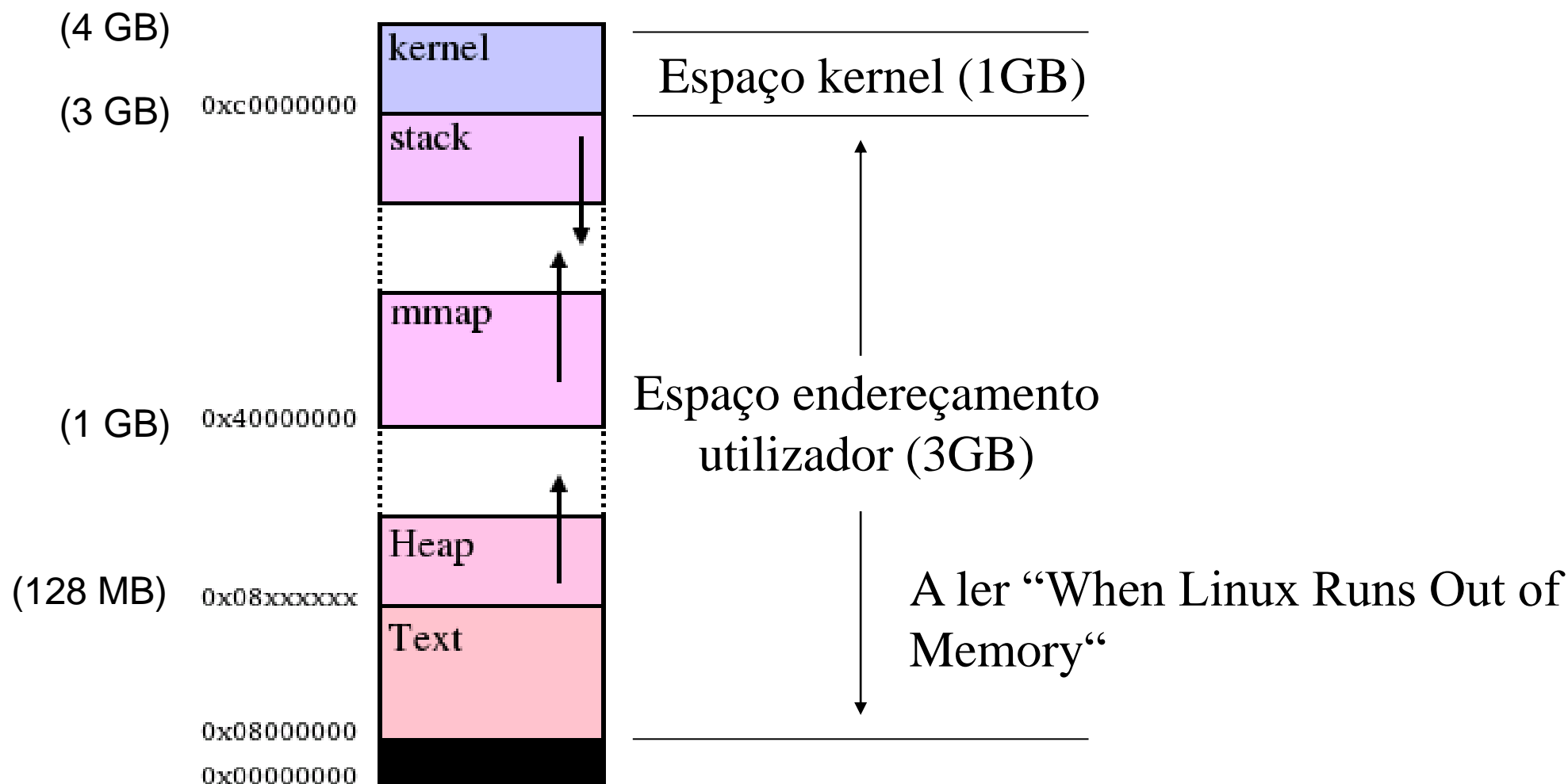
Mecanismo de “Copy-on-Write”

- Tipicamente, a memória encontra-se organizada em “páginas” de tamanho fixo (geralmente, 4 KB)
 - Memória paginada
- Após uma chamada “fork()”, os processos pai e filho partilham as mesmas páginas
 - Deste modo não há lugar à copia do “espaço de memória” do processo pai para o processo filho – evitam-se cópias desnecessárias!
- Quando uma página é escrita por um dos processos (e.g., o valor de uma variável é alterado), o SO procede à sua replicação
 - Passam a existir duas páginas daquela zona (uma para o pai, outra para o filho)

- “chamadas ao sistema” e “traps” (interrupções por software)
 - Representam saltos para código do “kernel”
 - O código a executar está no kernel
 - Exemplo: chamada ao sistema “write()” ou “fork()”
- Como evitar o remapeamento de endereços?
 - Isto é, como evitar que seja necessário converter os endereços do processo para os endereços do kernel?
 - NOTA: dada a elevada frequência de chamadas ao sistemas e “interrupções por software” é ****fundamental**** evitar essa tradução!

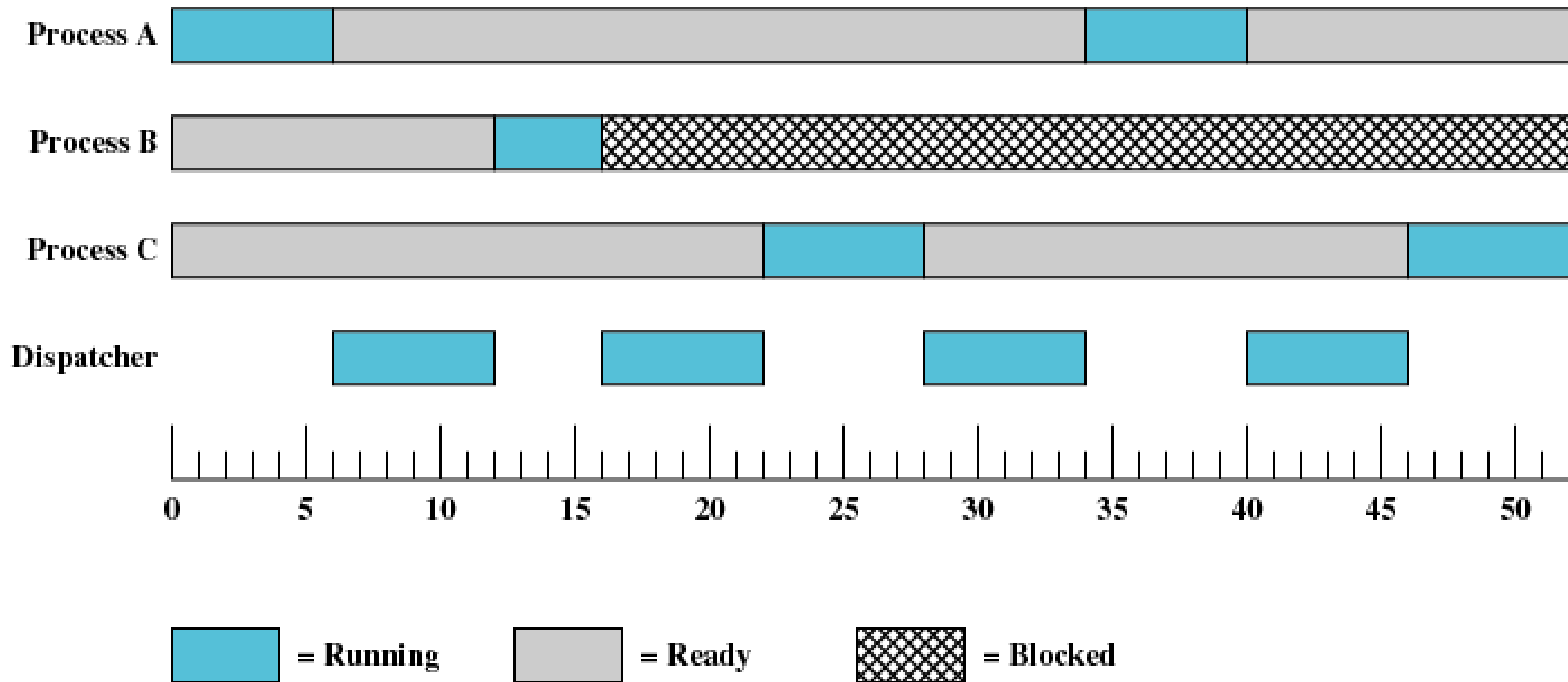
- Resposta: o espaço de endereçamento de cada processo é dividido em duas partes
 - Uma parte privada do processo: “**user space**”
 - Outra parte que apenas pode ser acedida quando a execução está em modo *kernel*: “**kernel space**”
- Como funciona?
 - O processo utilizador não tem acesso direto à memória do *kernel* (não pode nem ler, nem escrever)
 - Quando ocorre uma chamada ao sistema/*trap*, o processo é comutado para “modo *kernel*” acedendo à memória “*kernel*” mapeada

Espaço de endereçamento / Linux (32-bit)



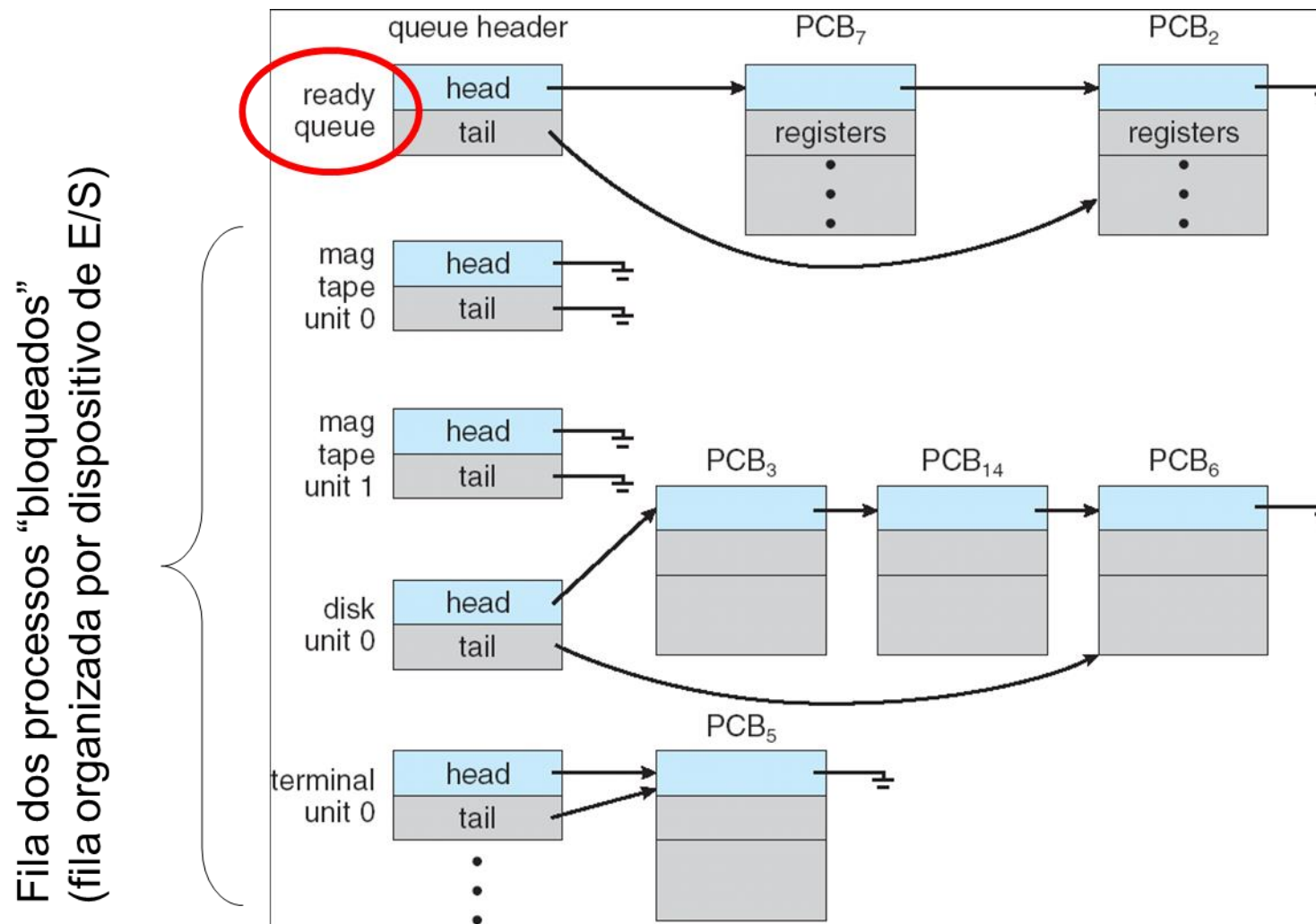
- Um processo é completamente eliminado pelo SO apenas quando o processo pai chama “wait()/waitpid”
 - Isso permite ao processo pai obter o código de término do processo filho
 - Código término: o inteiro que é devolvido pelo “main” (ou através do “exit”)
- **Processo “zombie”**
 - Processo que terminou, mas cujo processo pai não chamou o “wait()”
 - Atenção
 - Processos zombies ainda consomem recursos (PCBs e afins!)
- **Processo “orfão”**
 - Processo cujo processo pai já terminou. Neste caso o “init” (PID 1) assume o papel de processo pai

Execução simultânea de 3 processos

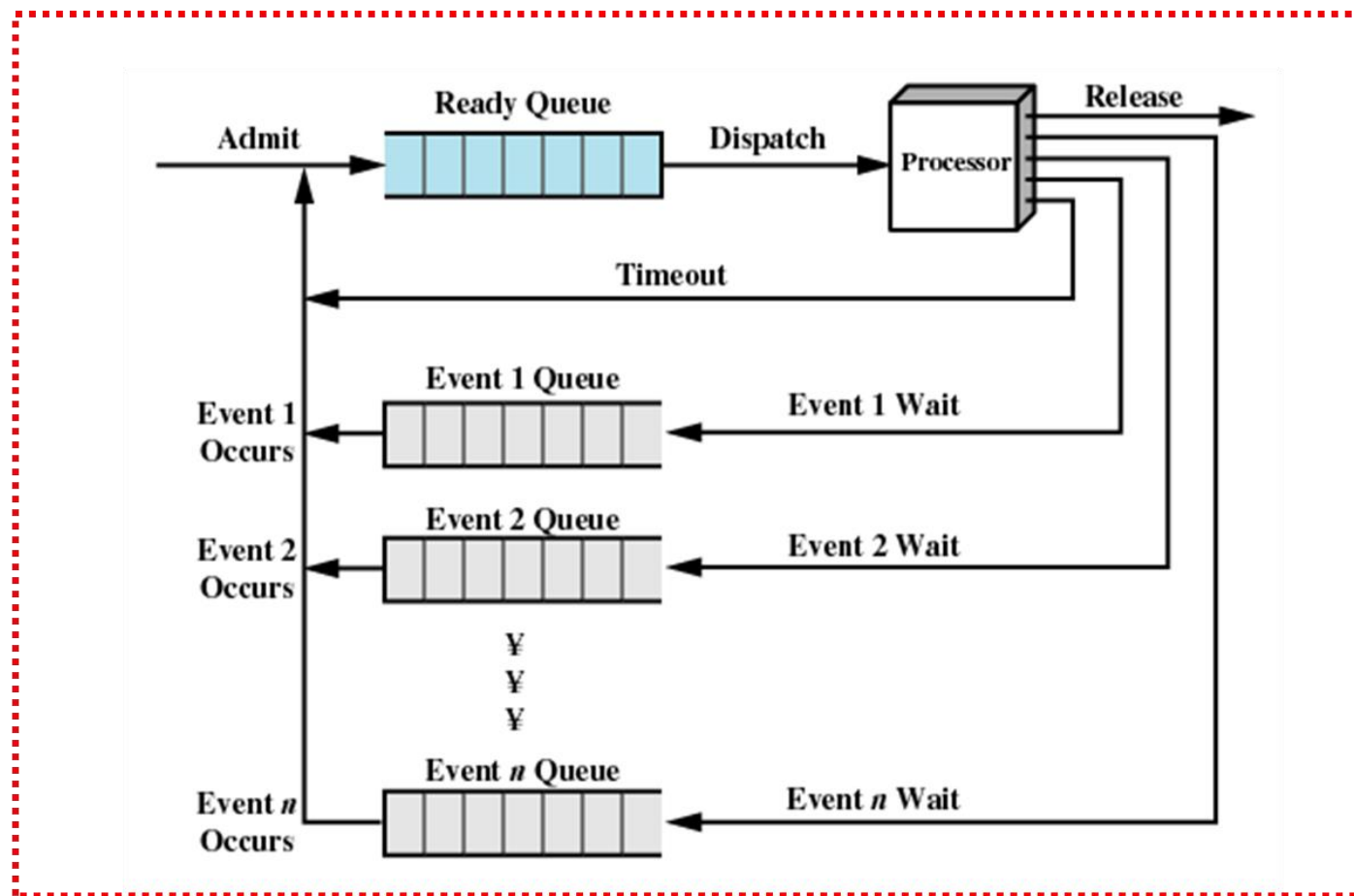
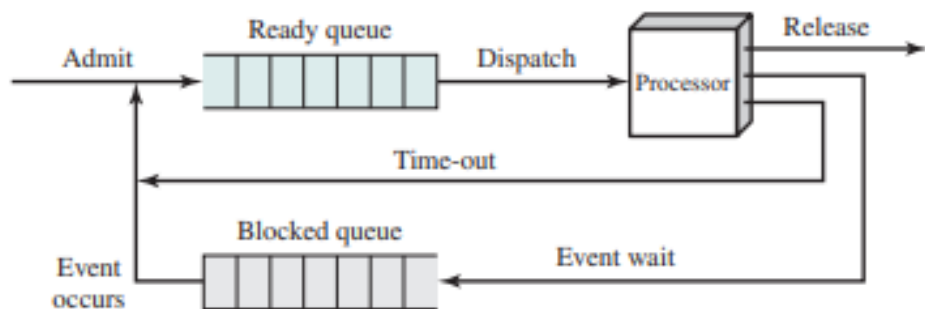


- Os PCBs são mantidos em várias filas de esperas (“queues”)
- Consoante o estado do processo, os PCBs migram de uma fila para a outra
- **Processos “prontos”**
 - Conjunto dos processos que estão “prontos” para serem executados
 - Aguardam apenas pela CPU
- **Processos à espera de E/S**
 - Conjunto de processos que aguardam dispositivos de E/S (também conhecidos como processos “bloqueados”)

Fila de Processos



Os PCBs vão comutando de Filas de Processos



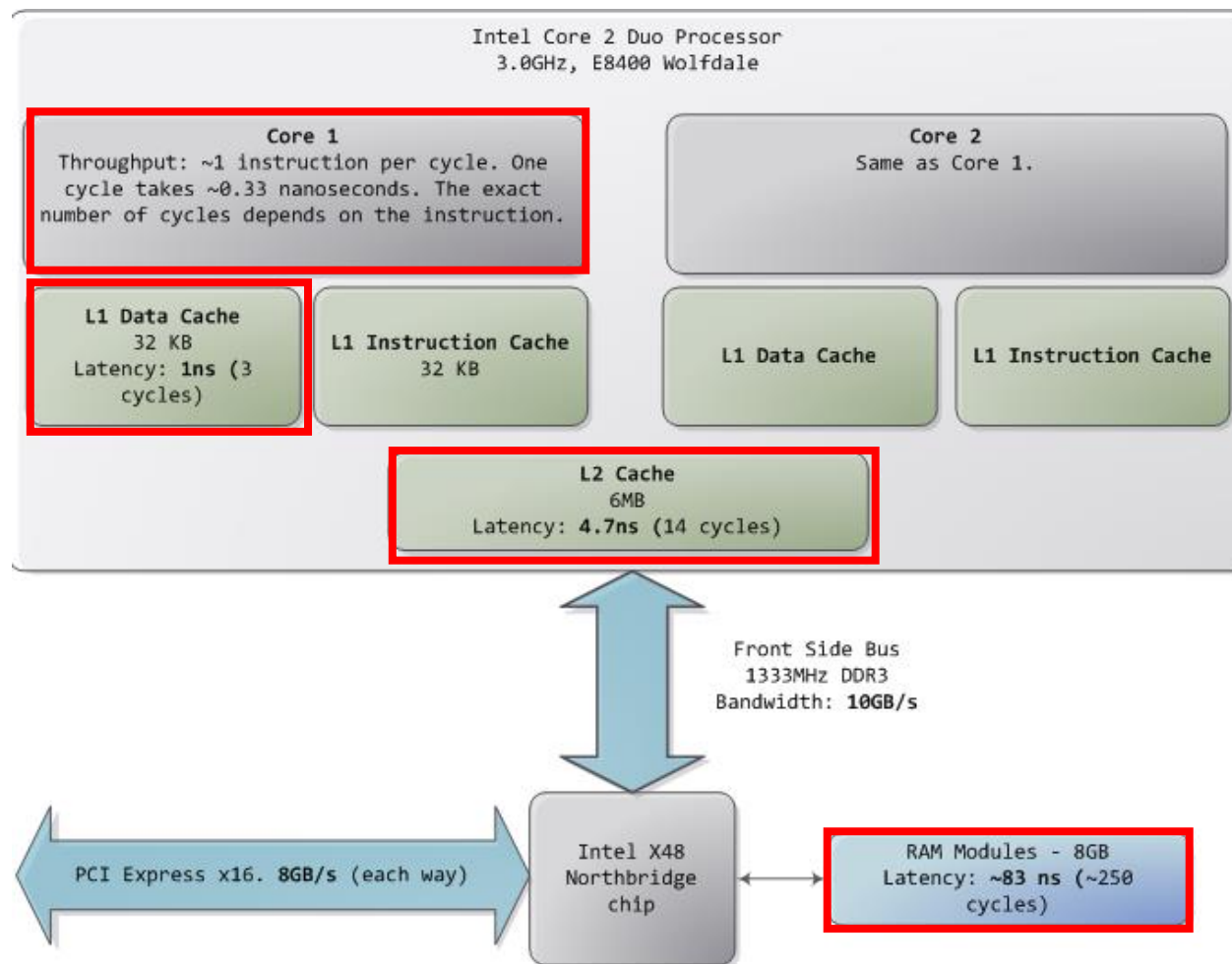
Escalonador de processos

Porquê uma abordagem “event driven”?

- Tem a ver com as velocidades relativas
 - CPU muito mais rápido do que os componentes de E/S

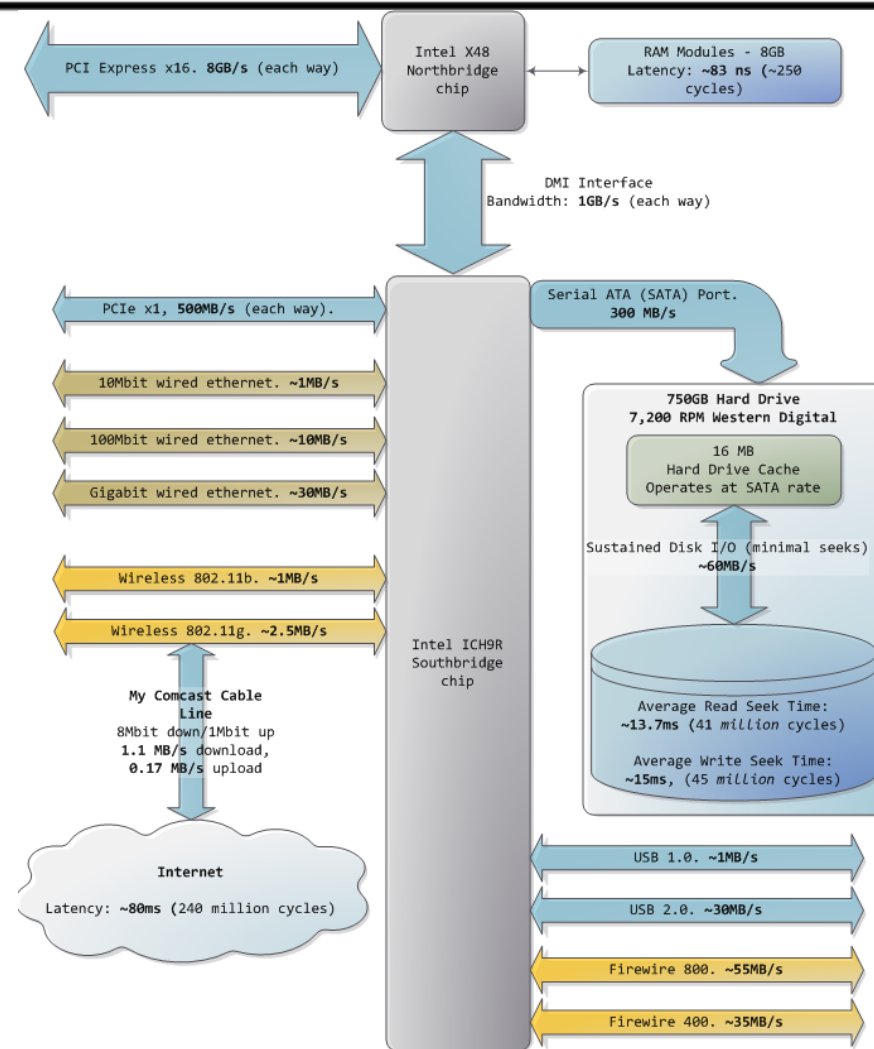
Características do CPU			Escaladas a tempo dos humanos	
<i>CPU frequency</i>	<i>2GHz</i>			
Processor Cycle Time	0,5 ns		1 s	
L2 cache access	10 ns		20 s	
Memory access	80 ns		160 s	(2.6 mins)
Thread context switch	5000 ns (5us)		10000 s	(2.7 horas)
Disk access	8000000 ns (8ms)		16000000 s	(185 dias)
Process quantum	100000000 ns (100ms)		200000000 s	(6.3 anos)
Azul ► Coisas melhorando muito rápido				
Laranja ► Coisa melhorando até um grau				
Vermelho ► Coisas que raramente melhoram				

Latências no acesso à memória (core2Duo)



<http://duartes.org/gustavo/blog>

Latência (fora do CPU)



<http://duartes.org/gustavo/blog>

Qual é o melhor “escalador”?

Qual é o melhor avião?

Avião	Capacidade (passageiros)	Autonomia (Km)	Velocidade(Km/h)	<i>throughput</i> (passag. x Km/h)
Boeing 777	375	7408	976	366000
Boeing 747	470	6640	976	458720
Concorde	132	6400	2160	285120
Douglas DC8	146	13952	870	127020

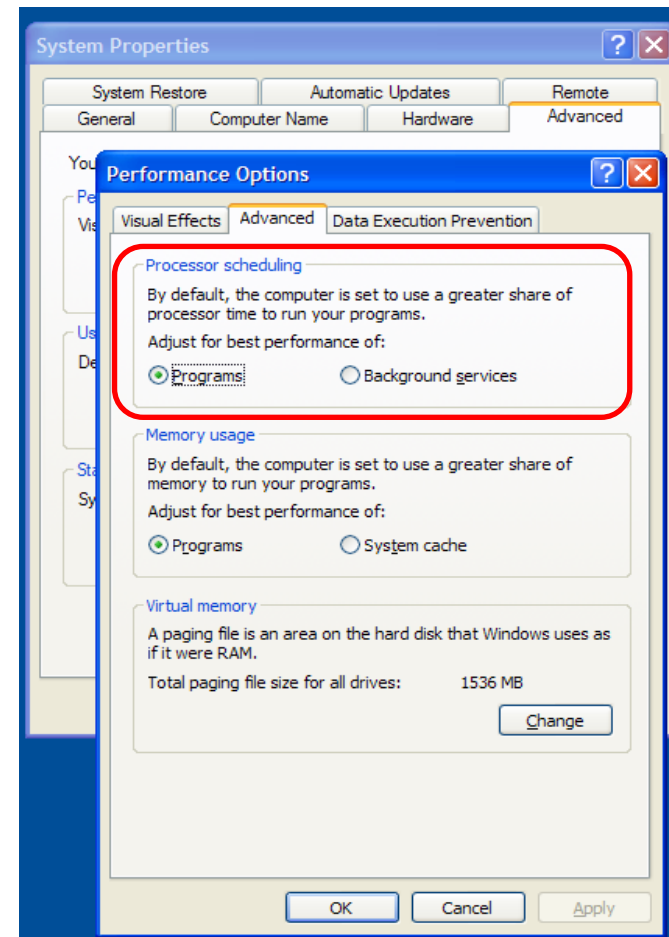
Resposta: **depende do objectivo!**

Qual é o melhor “escalador”?

Depende da finalidade...

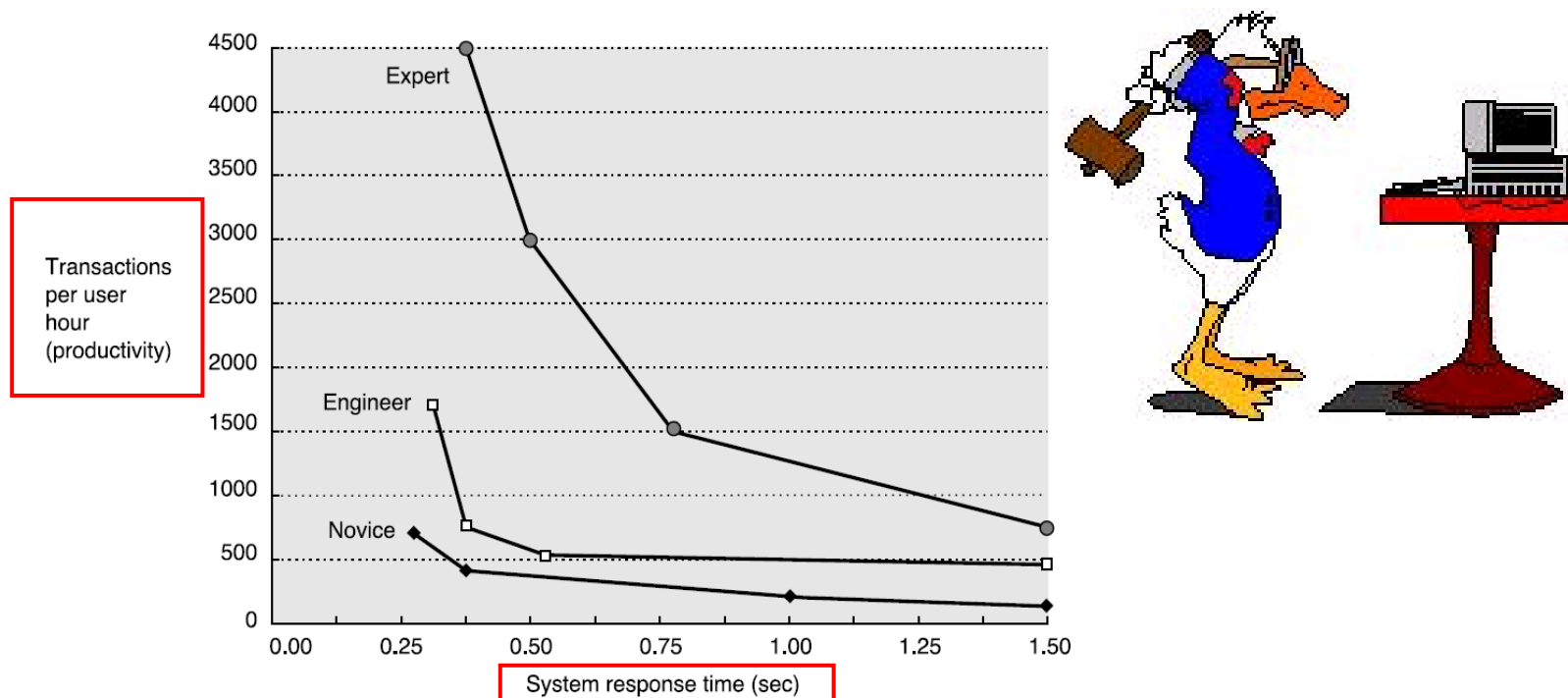
- SO multitarefas / servidores
- SO multitarefas / utilizadores interactivos (e.g. Windows)
- SO para processamentos em lotes
- SO de tempo real
- SO para multimédia
- ...

Até no windows 😊



Sistemas interactivos

- SO multitarefas devem ter cuidado no nível de multitarefa a suportar pelo sistema
 - O tempo de resposta é muito importante para o utilizador



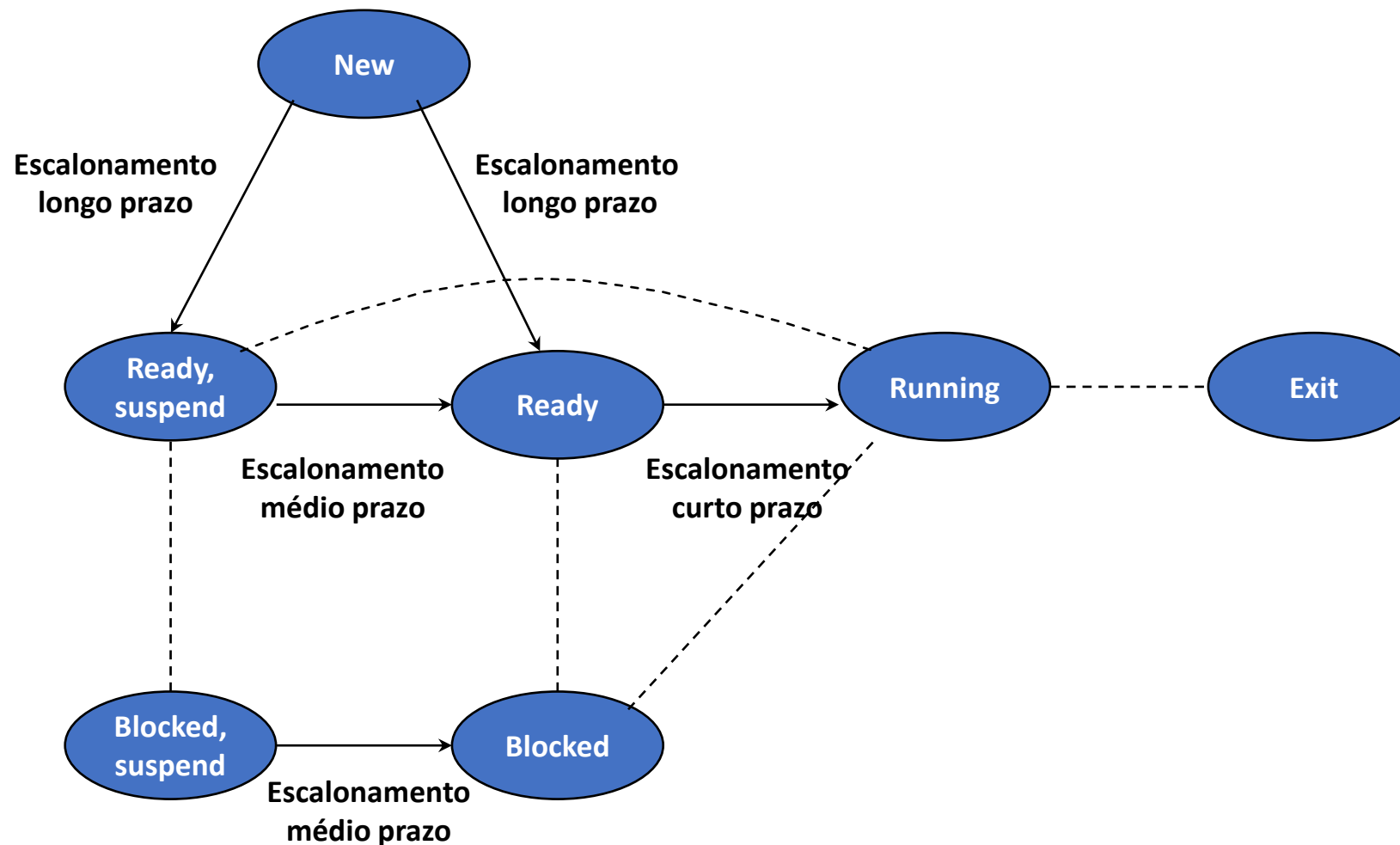
Objetivos do escalonamento

- Podem ser vários os objetivos do escalonamento...
 - Resposta rápida
 - Tempo que medeia entre a entrada do pedido de execução e o início da resposta
 - Execução rápida (*turnaround* time)
 - Tempo que medeia entre a entrada do pedido de execução e o término da execução
 - Elevado número de processos servidos
 - Nº de processos completados por unidade de tempo (*throughput*)
 - Eficiência da utilização do processador
 - Maximizar a utilização do CPU



Objetivos contraditórios: impossível agradar a todos!

- Longo prazo
 - Estabelecido quando o processo é criado
 - Que processo admitir no sistema?
- Médio prazo
 - Permuta do processo de memória principal para memória secundária (mecanismo de swapping, associado à memória virtual)
 - SO: que processo devo enviar para o SWAP ?
 - swap out (RAM >> SWAP)
 - swap in (SWAP >> RAM)
- Curto prazo
 - Qual o próximo processo a ser executado ?
 - Executado regularmente (e.g., todo os 100 ms)



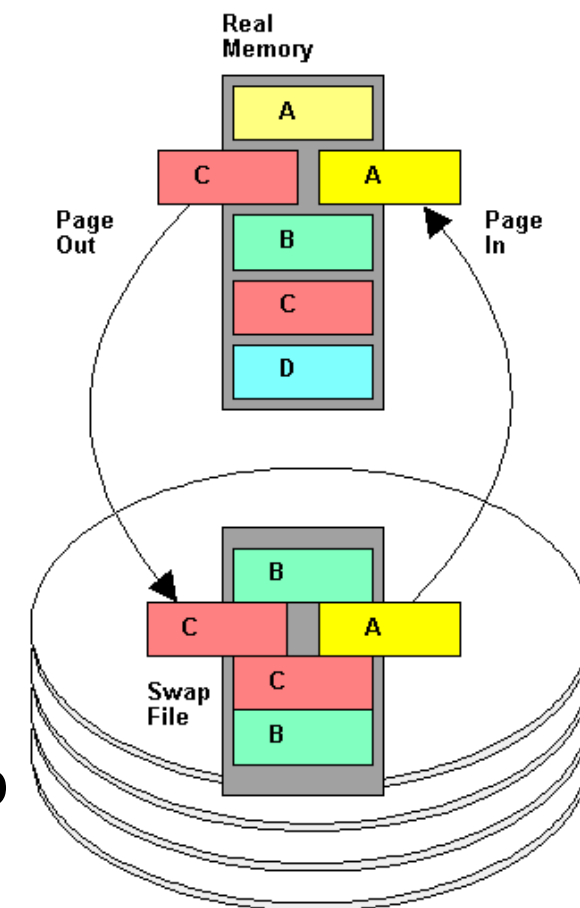
- Determina quais os programas a serem admitidos no sistema para processamento
- Controla o nível de “multiprogramação”
- Quanto mais processos, menor é a percentagem de CPU (e outros recursos) afecta a cada processo...
- O escalonamento de longo prazo é relevante nos seguintes sistemas:
- Processamento em lote
 - Agregados de computadores (clusters)
 - Supercomputadores
 - Rendering farms
- Nestes casos existe um computador de acesso, no qual são submetidos os programas a executar



Farm da PIXAR

- “Swapping”
 - Associado ao mecanismo de memória virtual
 - comutação de processo de:
 - memória primária (RAM) para memória secundária (disco)
 - swap out
 - Exemplo: processo inactivo há algum tempo é retirado para swap
 - memória secundária para memória primária
 - swap in
 - Exemplo: processo no swap voltou a ser preciso
- Baseado na necessidade de gestão da multiprogramação

From Computer Desktop Encyclopedia
© 2008 The Computer Language Co. Inc.

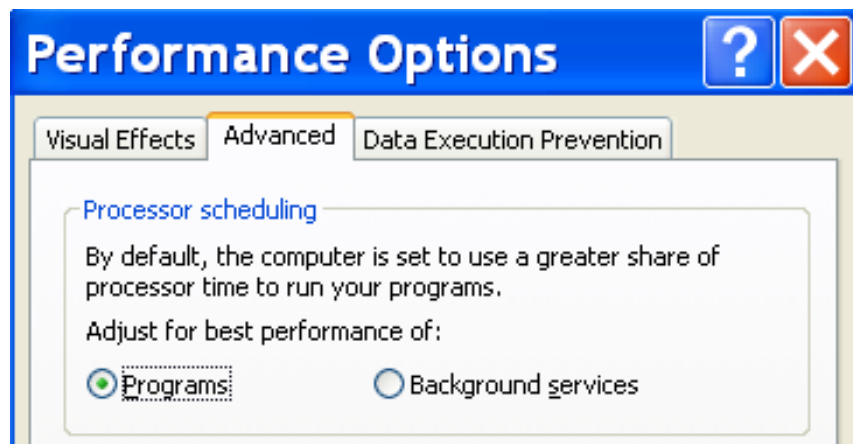


Escalonamento de curto prazo

- 4 eventos que desencadeiam o escalonamento:
 - A. Processo em execução transita para estado espera (espera I/O, resultante de chamada ao sistema -- e.g. **chamada *read* num ficheiro ou *socket***)
 - B. Processo em execução transita para o estado pronto (esgotou a sua fatia de tempo -- *time slice* – **interrupção do relógio**)
 - C. Processo transita do estado de espera para o estado pronto (pedido de I/O foi completado – **interrupção I/O**)
 - D. Término do processo (**chamada ao sistema**)
- Eventos B e C são exclusivos dos SO “preemptivos”
- O escalonador de curto prazo é executado muitos mais frequentemente que os escalonadores de médio/longo prazo
- Resumindo
 - SO retoma o controlo para “escalonar” quando ocorre:
 - Uma interrupção
 - Chamada ao sistema por parte do processo

Critérios de Escalonamento Curto Prazo

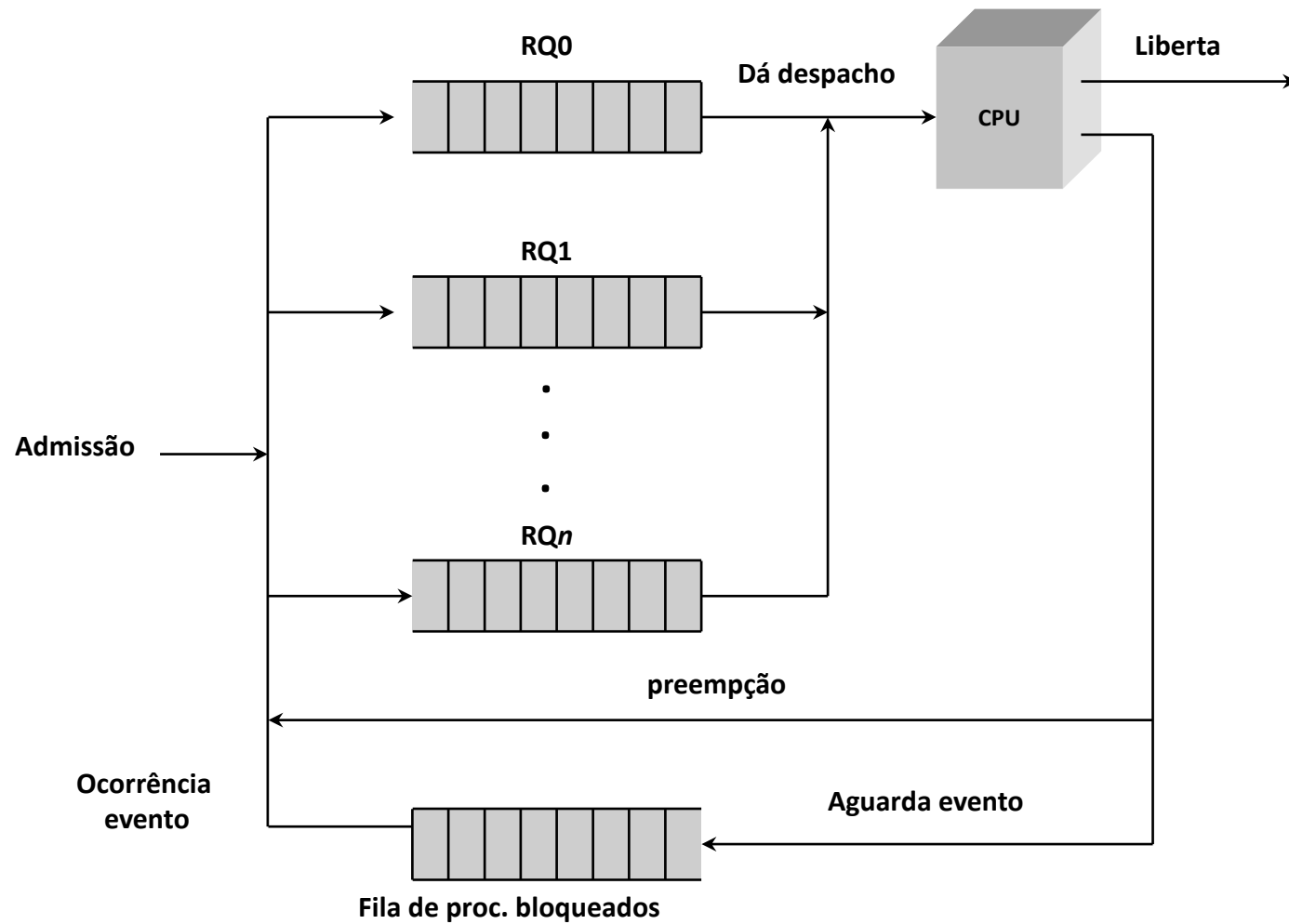
- Orientado para a qualidade de serviço ao utilizador
 - Tempo de resposta
 - Tempo que medeia entre a entrada de um pedido e o aparecer de “output”
- Orientado para o servidor
 - Maximizar a utilização (eficiente) do processador



← Windows

- O escalonador escolhe sempre um processo com prioridade superior
- Existem várias filas de espera, cada uma afeta a um nível de prioridade
- Os processos de baixa prioridade estão sujeitos a nunca acederem ao processador
 - Situação de “fome” (*starvation*)
 - Solução:
 - Permitir que a prioridade de um processo se altere dinamicamente consoante o seu tempo de vida e o seu perfil de execução

Prioridades – filas de espera



Prioridades (Windows)

- No Windows, como iniciar um processo com prioridade diferente de normal?
 - Comando “start” (existe no Windows)
 - `start /LOW programa.exe` -- prioridade LOW
 - `start /HIGH programa.exe` -- prioridade HIGH
 - `start /ABOVENORMAL programa.exe` – prior. >Normal
 - `start /BELOWNORMAL programa.exe` – prior. <Normal
 - Mais informação
 - `start /?`
- Alternativa “RunPrio” (utilitário freeware)
 - `runprio -x low cmd /c echo Hello World!`

- Indicação das prioridades através dos comandos “nice” e “renice”
 - `nice [opções] [comando [argumentos]...]`
 - `-n, --adjustment=N`
 - Adiciona N ao “niceness” do processo
 - Prioridade varia de -19 (mais elevada prioridade!) até +20 (menor prioridade)
 - Um utilizador normal apenas pode ****diminuir**** a prioridade dos ****seus**** processos (i.e. acrescentar um valor positivo com o “nice”)
 - O administrador de sistema pode aumentar/diminuir a prioridade de qualquer processo

Classificação de processos

- Processo “CPU-dependentes” (CPU-bound), cujo tempo de execução é determinado pela velocidade do CPU
 - Exemplo - programa de construção (“*rendering*”) de imagens
- Processo “IO-dependentes” (IO-bound) cujo tempo de execução é determinado pela velocidade do sistema de E/S (discos, etc.)
 - Exemplo – servidor de base de dados, programa de pesquisa de ficheiros (“procurar” do MS Windows)
- Processos Interativos
 - Existe interação com o utilizador
 - Exemplo – editor de texto, aplicações gráficas, shell,...
 - A frequência de “input” é baixa (ser humano muito mais lento que o computador)
 - O processo é frequentemente suspenso
 - Quando é recebido “input” (e.g. utilizador carregou numa tecla) o processo deve ser rapidamente escalonado

- Processos “*batch*”
 - Não requerem interação com o utilizador
 - Frequentemente associados a “serviços”
 - Correm tipicamente em “*background*”
 - Exemplos - compiladores, servidores de base de dados, servidores de web, etc.
- Processos “tempo-real”
 - Grandes exigências de escalonamento
 - Não devem ser bloqueados por processos de menor prioridade
 - Curto tempo de resposta, com uma variância mínima
 - Exemplos – aplicações vídeo e áudio, controladores de “robot” e programas de aquisição de dados

Critérios de avaliação do escalonador

- tempo de execução (*turnaround*) → mede o tempo decorrido entre a criação e o encerrar da tarefa, contabilizando todos os tempos de processamento e de espera.
- tempo de espera (*waiting time*) → tempo total perdido pela tarefa na fila de processos, aguardando o processador.
- tempo de resposta → tempo decorrido entre a chegada de um evento ao sistema e o resultado imediato de seu processamento.
- Eficiência → Indica o grau de utilização do processador na execução das tarefas do utilizador.
- Justiça → Distribuição do processador entre as tarefas prontas.

Cálculo do tempo médio de execução

- A soma dos tempos de execução de cada processo (tempo final – tempo de ingresso) dividido pela quantidade de processos.

$$T_t = \frac{t_e(t_1) + t_e(t_2) + t_e(t_3) + t_e(t_4)}{4}$$

$$T_t = \frac{(5 - 0) + (7 - 0) + (11 - 1) + (14 - 3)}{4}$$

$$T_t = \frac{5 + 7 + 10 + 11}{4} = \frac{33}{4} = 8.25s$$

Cálculo do tempo médio de espera

- A soma dos tempos de espera de cada processo (tempo de início da execução – tempo de ingresso) dividida pela quantidade de processos.

$$T_w = \frac{t_w(t_1) + t_w(t_2) + t_w(t_3) + t_w(t_4)}{4}$$

$$T_w = \frac{(0 - 0) + (5 - 0) + (7 - 1) + (11 - 3)}{4}$$

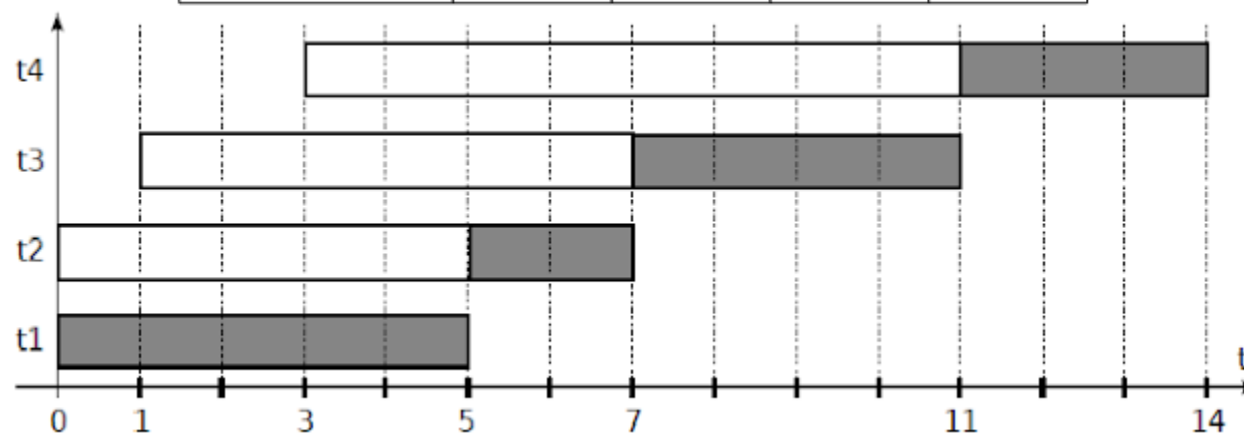
$$T_w = \frac{0 + 5 + 6 + 8}{4} = \frac{19}{4} = 4.75s$$

- Algoritmo de escalonamento
 - Seleciona o próximo processo a ocupar a CPU
- Algoritmos:
 - First-Come-First-Serve (FCFS)
 - Round-Robin
 - Processo mais Curto (SJF - Shortest Job First)
 - Por prioridades
 - Menor tempo de CPU
 - Escalonamento multinível de Fila
 - Escalonamento multinível de Feedback de Fila

First-Come-First-Serve (FCFS)

- É a forma mais elementar de escalonamento. Utiliza um algoritmo simples que atende as tarefas em sequência assim que ficam prontas. Ou seja, de acordo com sua chegada na fila de processos (FIFO). Não utiliza preempção.

Tarefa	t_1	t_2	t_3	t_4
Ingresso	0	0	1	3
Duração	5	2	4	3



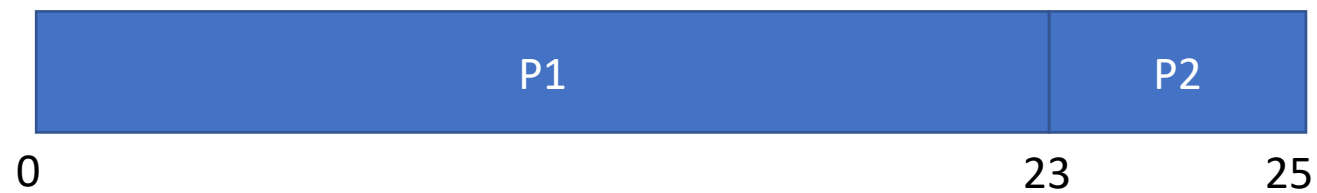
FCFS: Example 1

Process	Burst Time
P1	23
P2	2
P3	5



FCFS: Example 1

Process	Burst Time
P1	23
P2	2
P3	5



FCFS: Example 1

Process	Burst Time
P1	23
P2	2
P3	5



FCFS: Example 1



Waiting time :

P1 → 0

P2 → 23


P3 → 25

Average waiting time = $(0+23+25)/3 = 16$

Turnaround time: Waiting time + Burst Time

Average Turnaround time = $[(0+23) + (23+2) + (25+5)]/3 = 26$

FCFS: Example 2


process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0 
P5	4	4

0

P4

0

FCFS: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1 
P4	3	0
P5	4	4


1

P3

P4

0

FCFS: Example 2

process	Burst time	Arrival time
P1	6	2 
P2	2	5
P3	8	1
P4	3	0
P5	4	4

2

P3 P1

P4

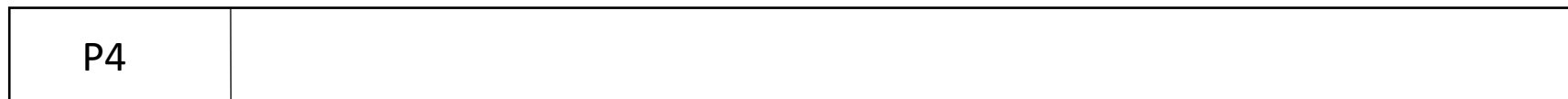
0

FCFS: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

3

P3 P1



0

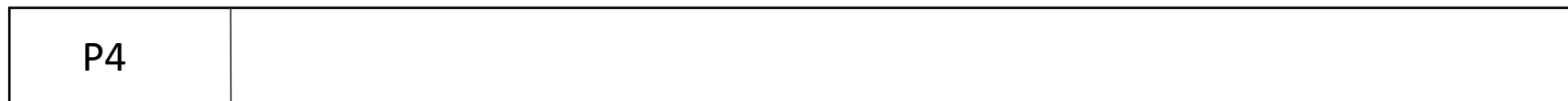
3

FCFS: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

3

P3 P1



0

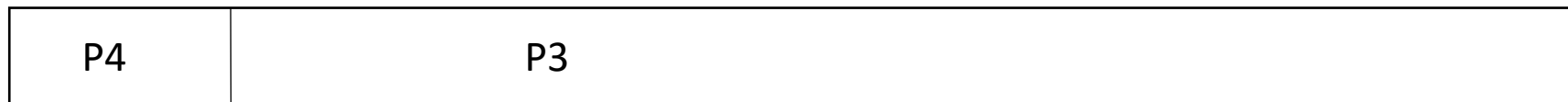
3

FCFS: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

4


P1 P5



0

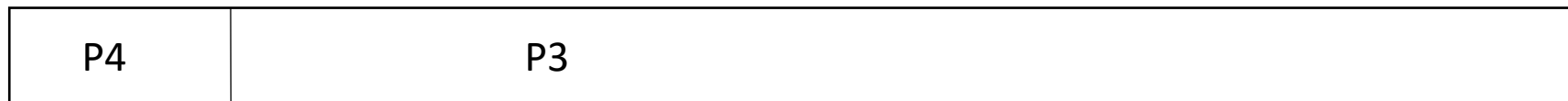
3

FCFS: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5 
P3	8	1
P4	3	0
P5	4	4

5

P1 P5 P2



0 3

FCFS: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

6

P1 P5 P2

P4

P3

0

3

FCFS: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

7

P1 P5 P2

P4

P3

0

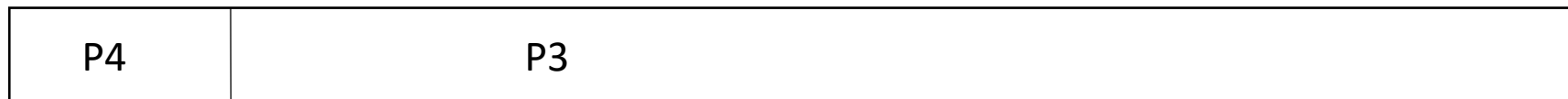
3

FCFS: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

8

P1 P5 P2



0

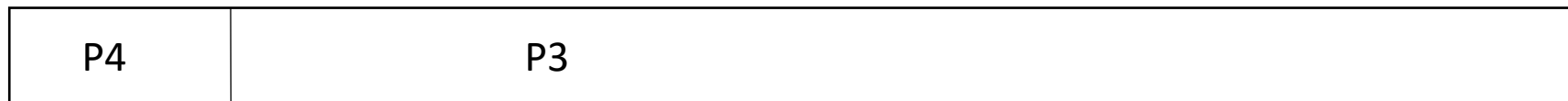
3

FCFS: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

9

P1 P5 P2



0

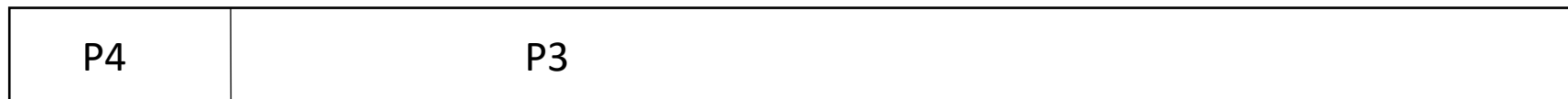
3

FCFS: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

10

P1 P5 P2



0

3

FCFS: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

11

P1 P5 P2



0

3

11

FCFS: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

11

P1 P5 P2



0

3

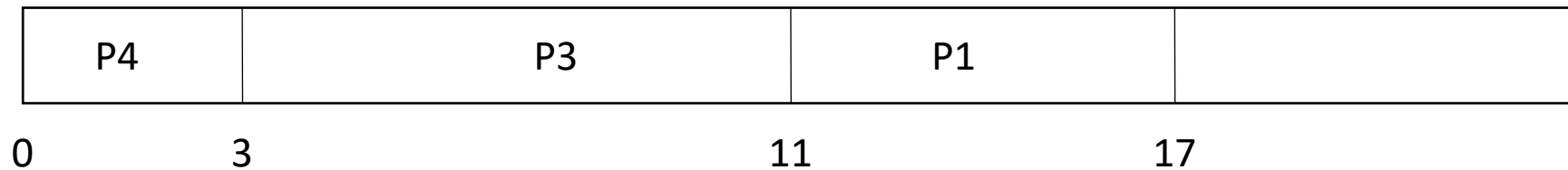
11

FCFS: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

17

P5 P2

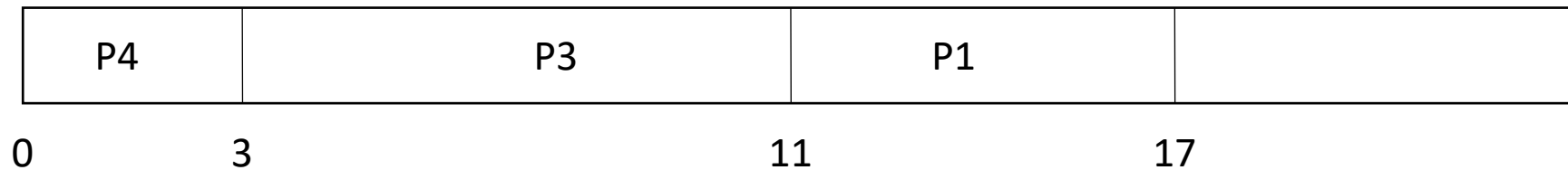


FCFS: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

17

P5 P2

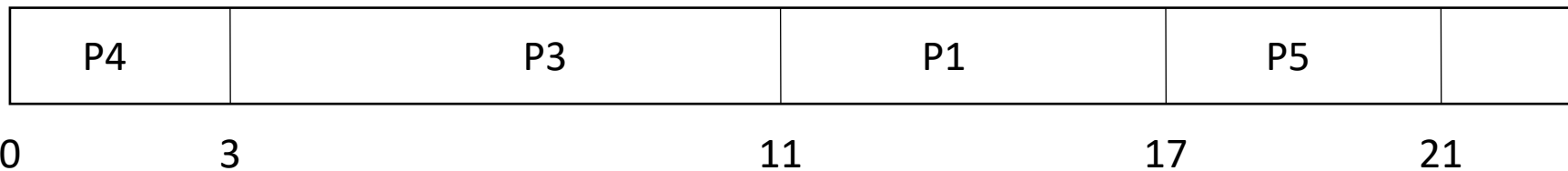


FCFS: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

21

P2



FCFS: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

21

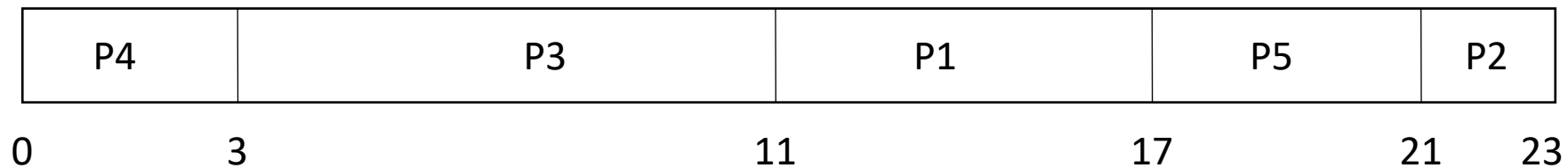
P2



FCFS: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

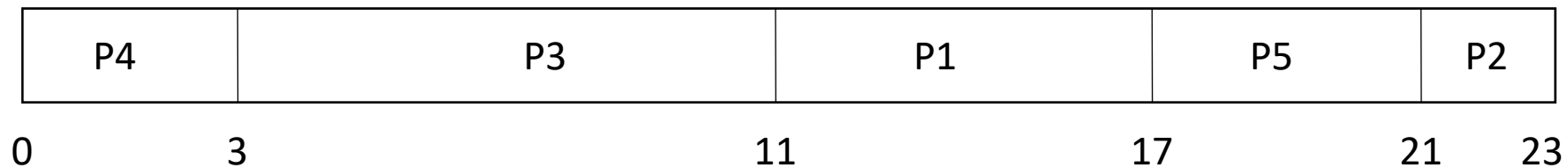
23



FCFS: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

23



FCFS: Example 2

process	Burst time	Arrival time
P1	6	1
P2	2	0
P3	8	4
P4	3	
P5	4	

P4	P3	P1	P5	P2
----	----	----	----	----

0	3	11	17	21	23
---	---	----	----	----	----

$$P4 = 0 - 0 = 0 \quad \text{Waiting time} = \text{Start time} - \text{Arrival time}$$

$$P3 = 3 - 1 = 2$$

$$P1 = 11 - 2 = 9$$

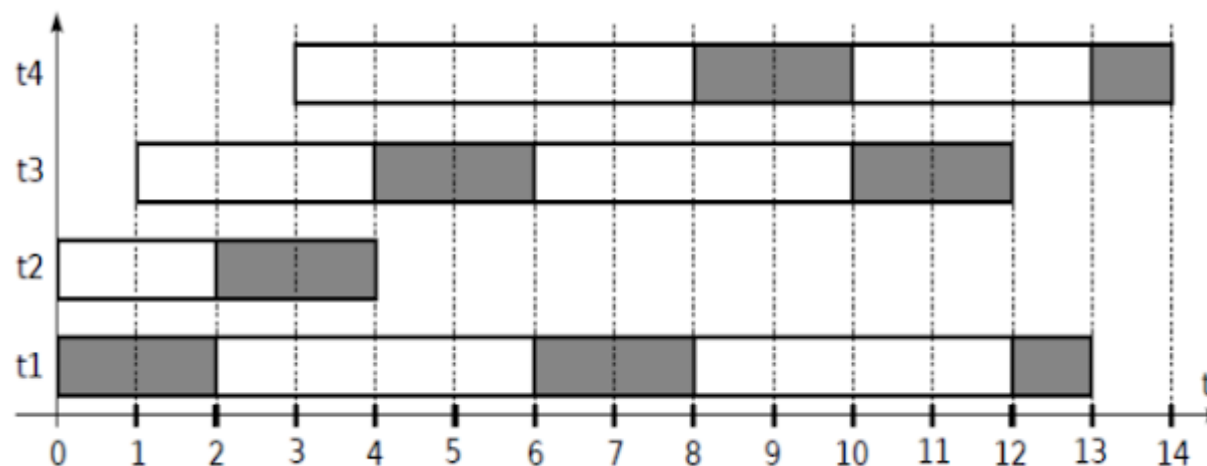
$$P5 = 17 - 4 = 13$$

$$P2 = 21 - 5 = 16$$


Av. Waiting Time =

Escalonamento Round-Robin

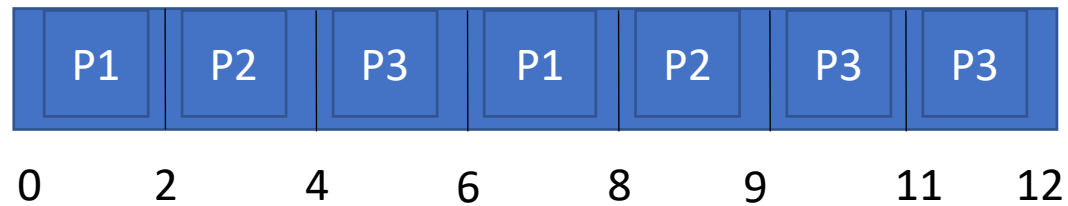
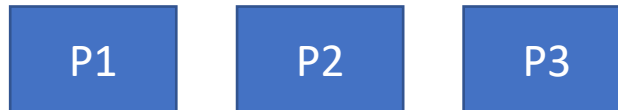
- O FCFS não leva em conta a importância das tarefas nem o seu comportamento em relação aos recursos. É o algoritmo FCFS com a adição de preempção por tempo ao escalonamento. Define um tempo de quantum.
- Exemplo: Tempo de quantum = 2



RR: Example 1

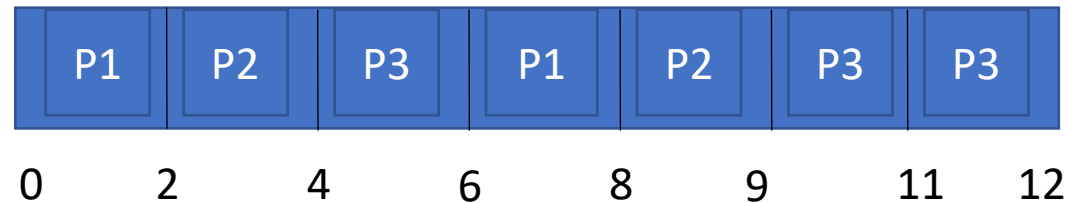
Process	Burst time
P1	4 
P2	3
P3	5

Time Quantum = 2



RR: Example 1

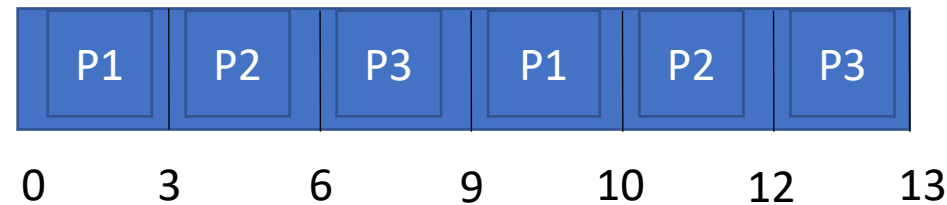
Process	Burst time
P1	4
P2	3
P3	5



RR with arrival time

Process	Burst Time	Arrival time
P1	4	0
P2	5	1
P3	4	2

Time quantum: 3



RR: Example 2

Time quantum = 2

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4




0

P4

0

RR: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1 
P4	3	0
P5	4	4

1

P3

P4

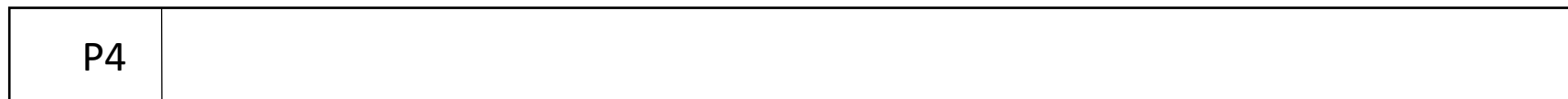
0

RR: Example 2

process	Burst time	Arrival time
P1	6	2 ←
P2	2	5
P3	8	1
P4	3	0
P5	4	4

2

P3 P1



0 2

RR: Example 2

process	Burst time	Arrival time
P1	6	2 ←
P2	2	5
P3	8	1
P4	3 / 1	0
P5	4	4

2

P1 P4

P4	P3
----	----

0 2

RR: Example 2

process	Burst time	Arrival time
P1	6	2 ←
P2	2	5
P3	8	1
P4	3 / 1	0
P5	4	4

3

P1 P4

P4	P3
----	----

0 2

RR: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3 / 1	0
P5	4	4



4

P1 P4 P5

P4	P3	
----	----	--

0 2 4

RR: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8 / 6	1
P4	3 / 1	0
P5	4	4

4

P4 P5 P3

P4	P3	P1
----	----	----

0 2 4

RR: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5 
P3	8 / 6	1
P4	3 / 1	0
P5	4	4

5

P4 P5 P3 P2

P4	P3	P1
----	----	----

0 2 4

RR: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8 / 6	1
P4	3 / 1	0
P5	4	4

6

P4 P5 P3 P2

P4	P3	P1	
----	----	----	--

0 2 4 6

RR: Example 2

process	Burst time	Arrival time
P1	6 / 4	2
P2	2	5
P3	8 / 6	1
P4	3 / 1	0
P5	4	4

6

P5 P3 P2 P1

P4	P3	P1	P4
----	----	----	----

0 2 4 6

RR: Example 2

process	Burst time	Arrival time
P1	6 / 4	2
P2	2	5
P3	8 / 6	1
P4	3 / 1	0
P5	4	4

7

P5 P3 P2 P1

P4	P3	P1	P4	
----	----	----	----	--

0 2 4 6 7

RR: Example 2

process	Burst time	Arrival time
P1	6 / 4	2
P2	2	5
P3	8 / 6	1
P4	3	0
P5	4	4

7

P5 P3 P2 P1

P4	P3	P1	P4	
----	----	----	----	--

0 2 4 6 7

RR: Example 2

process	Burst time	Arrival time
P1	6 / 4	2
P2	2	5
P3	8 / 6	1
P4	3	0
P5	4	4

8

P3 P2 P1

P4	P3	P1	P4	P5
----	----	----	----	----

0 2 4 6 7

RR: Example 2

process	Burst time	Arrival time
P1	6 / 4	2
P2	2	5
P3	8 / 6	1
P4	3	0
P5	4	4

9

P3 P2 P1

P4	P3	P1	P4	P5	
----	----	----	----	----	--

0 2 4 6 7 9

RR: Example 2

process	Burst time	Arrival time
P1	6 / 4	2
P2	2	5
P3	8 / 6	1
P4	3	0
P5	4 / 2	4

9

P2 P1 P5

P4	P3	P1	P4	P5	P3
----	----	----	----	----	----

0 2 4 6 7 9

RR: Example 2

process	Burst time	Arrival time
P1	6 / 4	2
P2	2	5
P3	8 / 6	1
P4	3	0
P5	4 / 2	4

10

P2 P1 P5

P4	P3	P1	P4	P5	P3
----	----	----	----	----	----

0 2 4 6 7 9

RR: Example 2

process	Burst time	Arrival time
P1	6 / 4	2
P2	2	5
P3	8 / 6	1
P4	3	0
P5	4 / 2	4

11

P2 P1 P5

P4	P3	P1	P4	P5	P3	
----	----	----	----	----	----	--

0 2 4 6 7 9 11

RR: Example 2

process	Burst time	Arrival time
P1	6 / 4	2
P2	2	5
P3	8 / 6 / 4	1
P4	3	0
P5	4 / 2	4

11

P1 P5 P3

P4	P3	P1	P4	P5	P3	P2
----	----	----	----	----	----	----

0 2 4 6 7 9 11

RR: Example 2

process	Burst time	Arrival time
P1	6 / 4	2
P2	2	5
P3	8 / 6 / 4	1
P4	3	0
P5	4 / 2	4

12

P1 P5 P3

P4	P3	P1	P4	P5	P3	P2
----	----	----	----	----	----	----

0 2 4 6 7 9 11

RR: Example 2

process	Burst time	Arrival time
P1	6 / 4	2
P2	2	5
P3	8 / 6 / 4	1
P4	3	0
P5	4 / 2	4

13

P1 P5 P3

P4	P3	P1	P4	P5	P3	P2	
----	----	----	----	----	----	----	--

0 2 4 6 7 9 11 13

RR: Example 2

process	Burst time	Arrival time
P1	6 / 4	2
P2	2	5
P3	8 / 6 / 4	1
P4	3	0
P5	4 / 2	4

13

P5 P3

P4	P3	P1	P4	P5	P3	P2	P1
----	----	----	----	----	----	----	----

0 2 4 6 7 9 11 13

RR: Example 2

process	Burst time	Arrival time
P1	6 / 4	2
P2	2	5
P3	8 / 6 / 4	1
P4	3	0
P5	4 / 2	4

14

P5 P3

P4	P3	P1	P4	P5	P3	P2	P1
----	----	----	----	----	----	----	----

0 2 4 6 7 9 11 13

RR: Example 2

process	Burst time	Arrival time
P1	6 / 4	2
P2	2	5
P3	8 / 6 / 4	1
P4	3	0
P5	4 / 2	4

15

P5 P3

P4	P3	P1	P4	P5	P3	P2	P1	
0	2	4	6	7	9	11	13	15

RR: Example 2

process	Burst time	Arrival time
P1	6 / 4 / 2	2
P2	2	5
P3	8 / 6 / 4	1
P4	3	0
P5	4 / 2	4

15

P3 P1

P4	P3	P1	P4	P5	P3	P2	P1	P5
----	----	----	----	----	----	----	----	----

0 2 4 6 7 9 11 13 15

RR: Example 2

process	Burst time	Arrival time
P1	6 / 4 / 2	2
P2	2	5
P3	8 / 6 / 4	1
P4	3	0
P5	4 / 2	4

16

P3 P1

P4	P3	P1	P4	P5	P3	P2	P1	P5
----	----	----	----	----	----	----	----	----

0 2 4 6 7 9 11 13 15

RR: Example 2

process	Burst time	Arrival time
P1	6 / 4 / 2	2
P2	2	5
P3	8 / 6 / 4	1
P4	3	0
P5	4 / 2	4

17

P3 P1

P4	P3	P1	P4	P5	P3	P2	P1	P5	
0	2	4	6	7	9	11	13	15	17

RR: Example 2

process	Burst time	Arrival time
P1	6 / 4 / 2	2
P2	2	5
P3	8 / 6 / 4	1
P4	3	0
P5	4	4

17

P1

P4	P3	P1	P4	P5	P3	P2	P1	P5	P3
0	2	4	6	7	9	11	13	15	17

RR: Example 2

process	Burst time	Arrival time
P1	6 / 4 / 2	2
P2	2	5
P3	8 / 6 / 4	1
P4	3	0
P5	4	4

18

P1

P4	P3	P1	P4	P5	P3	P2	P1	P5	P3
----	----	----	----	----	----	----	----	----	----

0 2 4 6 7 9 11 13 15 17

RR: Example 2

process	Burst time	Arrival time
P1	6 / 4 / 2	2
P2	2	5
P3	8 / 6 / 4	1
P4	3	0
P5	4	4

19

P1

P4	P3	P1	P4	P5	P3	P2	P1	P5	P3	
----	----	----	----	----	----	----	----	----	----	--

0 2 4 6 7 9 11 13 15 17 19

RR: Example 2

process	Burst time	Arrival time
P1	6 / 4 / 2	2
P2	2	5
P3	8 / 6 / 4 / 2	1
P4	3	0
P5	4	4

19

P3

P4	P3	P1	P4	P5	P3	P2	P1	P5	P3	P1
0	2	4	6	7	9	11	13	15	17	19

RR: Example 2

process	Burst time	Arrival time
P1	6 / 4 / 2	2
P2	2	5
P3	8 / 6 / 4 / 2	1
P4	3	0
P5	4	4

20

P3

P4	P3	P1	P4	P5	P3	P2	P1	P5	P3	P1
0	2	4	6	7	9	11	13	15	17	19

RR: Example 2

process	Burst time	Arrival time
P1	6 / 4 / 2	2
P2	2	5
P3	8 / 6 / 4 / 2	1
P4	3	0
P5	4	4

21

P3

P4	P3	P1	P4	P5	P3	P2	P1	P5	P3	P1	
0	2	4	6	7	9	11	13	15	17	19	21

RR: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8 / 6 / 4 / 2	1
P4	3	0
P5	4	4

21

P4	P3	P1	P4	P5	P3	P2	P1	P5	P3	P1	P3
----	----	----	----	----	----	----	----	----	----	----	----

0 2 4 6 7 9 11 13 15 17 19 21

RR: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8 / 6 / 4 / 2	1
P4	3	0
P5	4	4

22

P4	P3	P1	P4	P5	P3	P2	P1	P5	P3	P1	P3
0	2	4	6	7	9	11	13	15	17	19	21

RR: Example 2

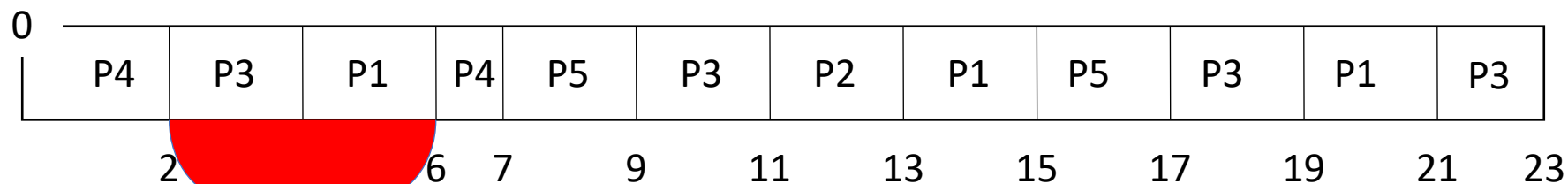
process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

23

P4	P3	P1	P4	P5	P3	P2	P1	P5	P3	P1	P3	
0	2	4	6	7	9	11	13	15	17	19	21	23

RR: Example 2

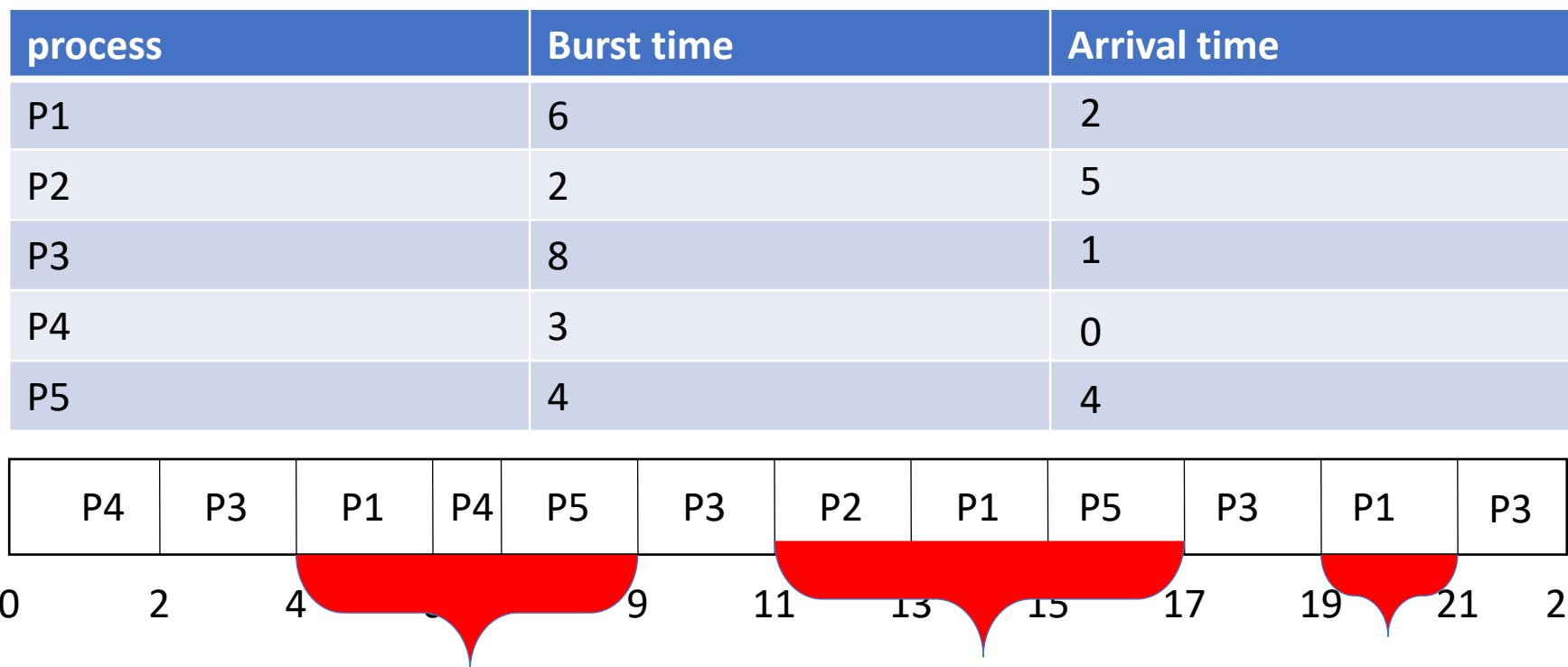
process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4



Waiting time = Start time – Arrival time

$$P4 = 0 - 0 + 4 = 4$$

RR: Example 2



Waiting time = Start time – Arrival time

$$P4 = 0 - 0 + 4 = 4$$

$$P3 = 2 - 1 + 5 + 6 + 2 = 14$$

RR: Example 2

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

P4	P3	P1	P4	P5	P3	P2	P1	P5	P3	P1	P3	
0	2	4	6	7	9	11	13	15	17	19	21	2

$$P4 = 0 - 0 + 4 = 4$$

$$P3 = 2 - 1 + 5 + 6 + 2 = 14$$

$$P1 = 4 - 2 + 7 + 4 = 13$$

$$P5 = 7 - 4 + 6 = 9$$

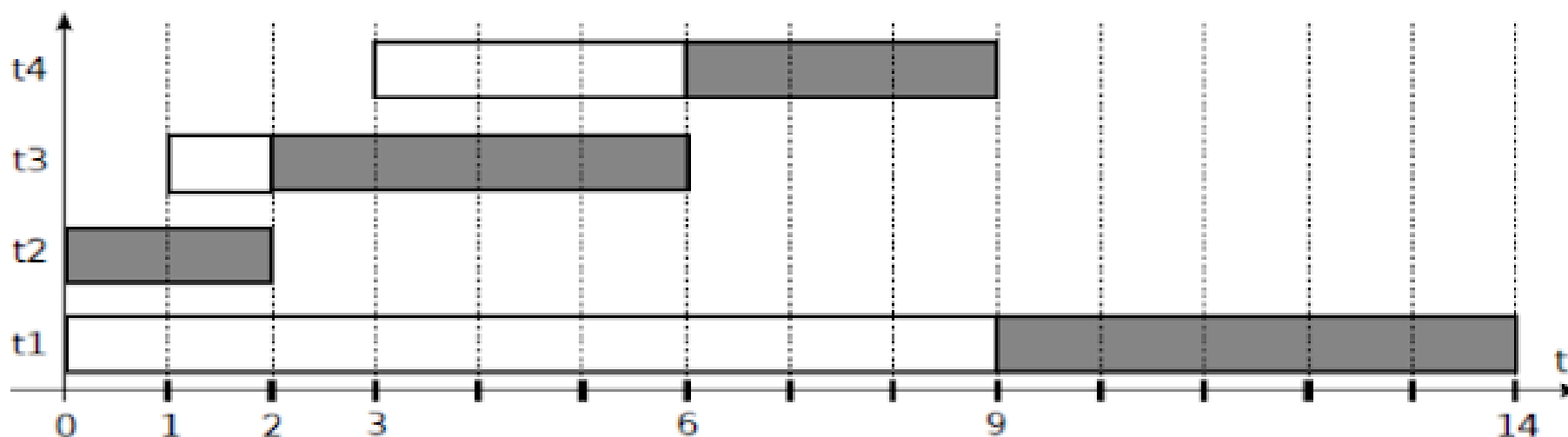
$$P2 = 11 - 5 = 6$$

Waiting time = Start time – Arrival time

Av. Waiting Time =

Processo mais Curto (SJF - Shortest Job First)

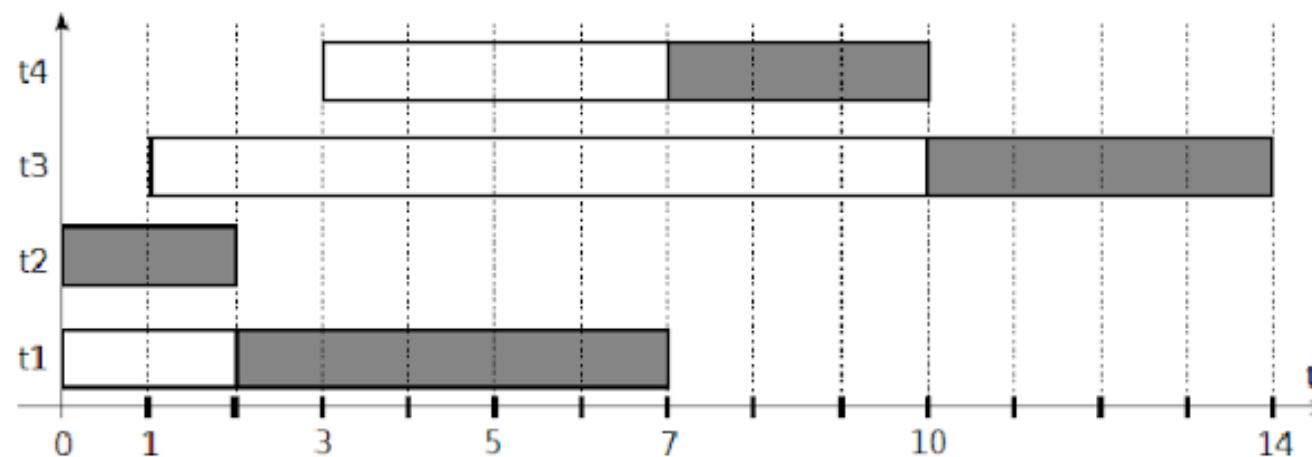
- O algoritmo de escalonamento que proporciona os menores tempos médios de execução e de espera é conhecido como menor tarefa primeiro, ou SJF (Shortest Job First). Consiste em atribuir o processador à menor (mais curta) tarefa da fila de tarefas prontas.



Escalonamento por Prioridades

- No escalonamento por prioridades, a cada tarefa é associada uma prioridade (número inteiro) usada para escolher a próxima tarefa a receber o processador, a cada troca de contexto.

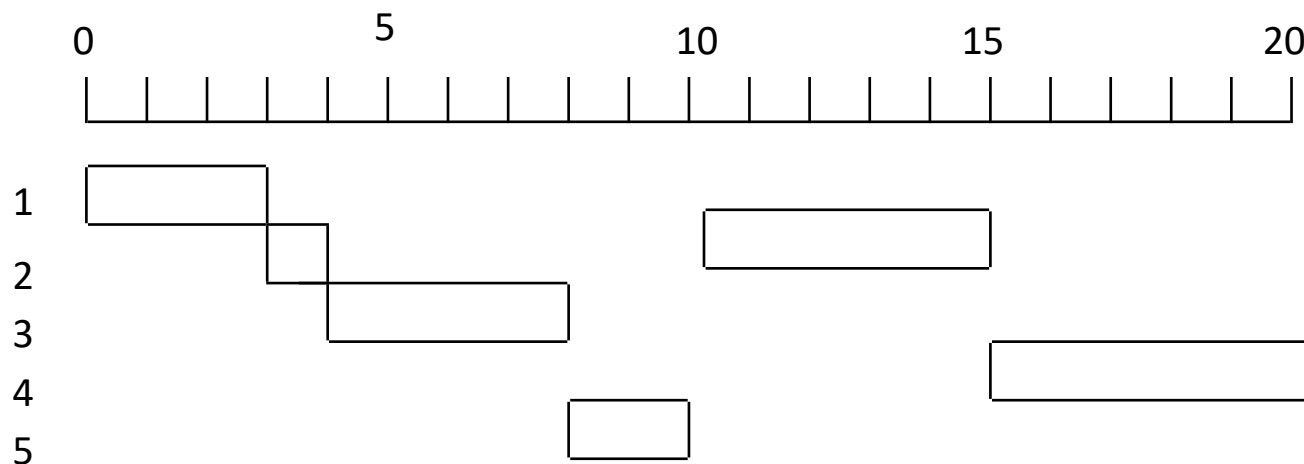
Tarefa	t_1	t_2	t_3	t_4
Ingresso	0	0	1	3
Duração	5	2	4	3
Prioridade	2	3	1	4



Menor tempo de CPU

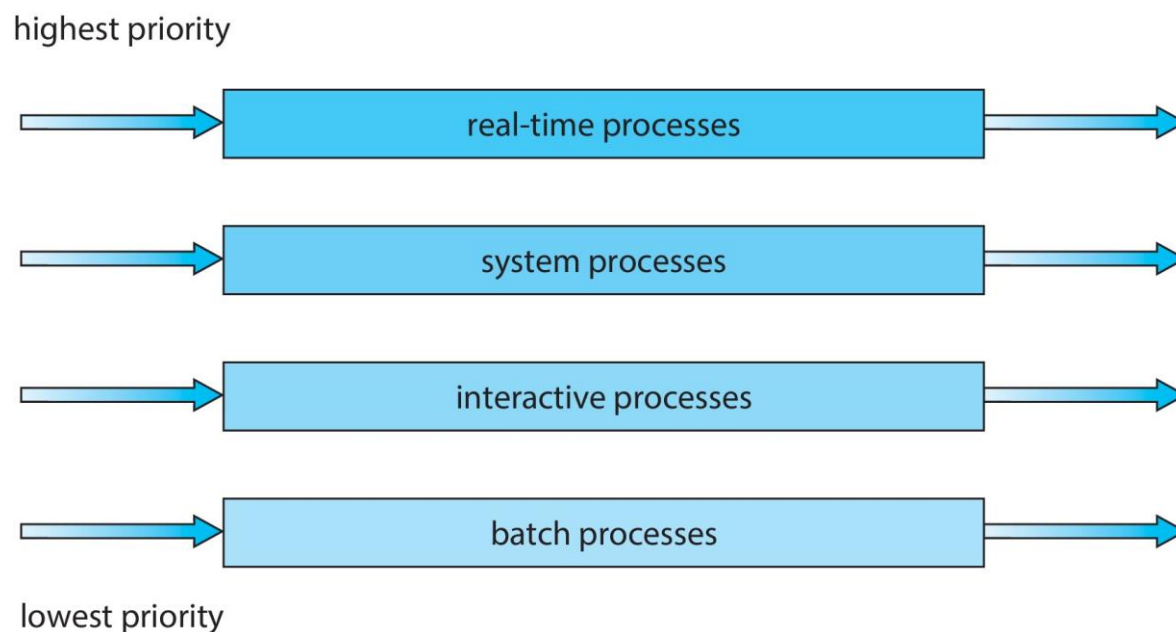
- Selecção do processo que permanecerá menor tempo no CPU
- Privilegia o processo E/S-bound
- Versão “preemptiva” da metodologia “processo mais curto”
- Requer estimativa do tempo de CPU a consumir pelo processo na próxima instância (CPU burst)

Processo	Tempo de entrada no sistema	Tempo de serviço
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



Escalonamento multinível de Fila

- Um algoritmo de escalonamento de fila multinível divide a fila pronta em várias filas separadas. Os processos são atribuídos permanentemente a uma fila, geralmente com base em alguma propriedade do processo, como tamanho da memória, prioridade do processo ou tipo de processo. Cada fila possui seu próprio algoritmo de agendamento.



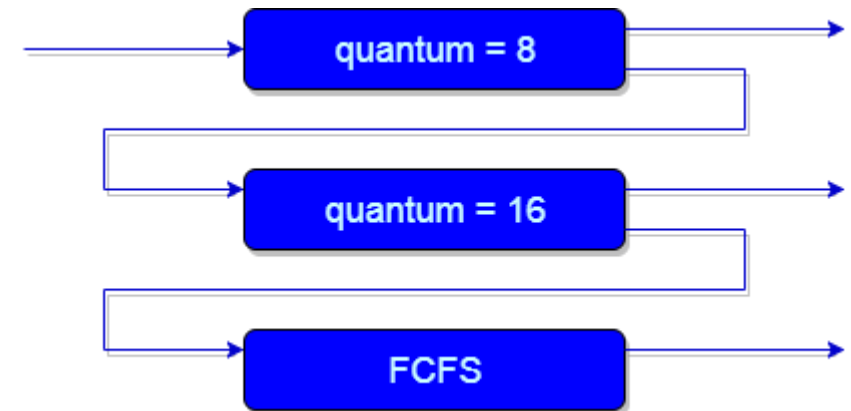
- Vantagens do escalonamento multinível de fila
 - Para processos do sistema: Escalonamento First Come First Serve(FCFS).
 - Para processos interativos: Escalonamento Shortest Job First (SJF)
 - Para Processos em Batch: Escalonamento Round Robin (RR)
- A principal desvantagem do Agendamento multinível de fila é o problema de starvation para processos de nível inferior.

Escalonamento multinível de Feedback de Fila

Permite que um processo se mova entre as filas. A ideia é separar processos com diferentes características de pico de CPU. Se um processo usar muito tempo de CPU, ele será movido para uma fila de prioridade mais baixa. Da mesma forma, um processo que espera muito tempo em uma fila de prioridade mais baixa pode ser movido para uma fila de prioridade mais alta. Essa forma de envelhecimento evita a starvation.

Sendo definido pelos seguintes parâmetros:

- O número de filas.
- O algoritmo de escalonamento para cada fila.
- O método usado para determinar quando atualizar um processo para uma fila de prioridade mais alta.
- O método usado para determinar quando rebaixar um processo para uma fila de prioridade mais baixa.
- O método usado para determinar em qual fila um processo entrará quando esse processo precisar de serviço.



Pontos para entender a necessidade de tal programação complexa:

- Este agendamento é mais flexível do que o agendamento de fila multinível.
- Este algoritmo ajuda a reduzir o tempo de resposta.
- Para otimizar o tempo de resposta, é necessário o algoritmo SJF, que basicamente requer o tempo de execução dos processos para escaloná-los. O tempo de execução dos processos não é conhecido com antecedência. Além disso, este escalonamento executa principalmente um processo para um quantum de tempo e, depois disso, pode alterar a prioridade do processo se o processo for longo. Assim, este algoritmo de escalonamento aprende principalmente com o comportamento passado dos processos e, então, pode prever o comportamento futuro dos processos. Desta forma, tenta executar primeiro um processo mais curto que, em troca, leva a otimizar o tempo de resposta.

- Vantagens
 - Este é um algoritmo de escalonamento flexível.
 - Este algoritmo de escalonamento permite que diferentes processos se movimente entre diferentes filas.
 - Neste algoritmo, um processo que espera muito tempo em uma fila de prioridade mais baixa pode ser movido para uma fila de prioridade mais alta, o que ajuda a prevenir a starvation.
- Desvantagens
 - Este algoritmo é muito complexo.
 - À medida que os processos se movem em diferentes filas, isso leva à produção de mais sobrecargas de CPU.
 - A fim de selecionar o melhor escalonador, este algoritmo requer alguns outros meios para selecionar os valores necessários.

Escalonamento multinível de Feedback de Fila

Process	Arrival time	CPU burst time	I/O	CPU burst time
A	0	5	6	0
B	3	4	2	3
C	4	2	3	4
D	7	5	2	7
E	14	3	2	4

Assuma Queue 1 com Quantum =4 ms e Queue 2 com Quantum = 7ms, ambos usam Round Robin.

Queue_1 : ABCDBCEED

Queue_2 : AD

CPU	A	B	C	D	B	C	E	A	D	E	idle	D
	0	4	8	10	14	17	21	24	25	26	30	34
I/O	idle		B	C	idle			E	A		D	
	0	8	10	13				24	26	32	34	

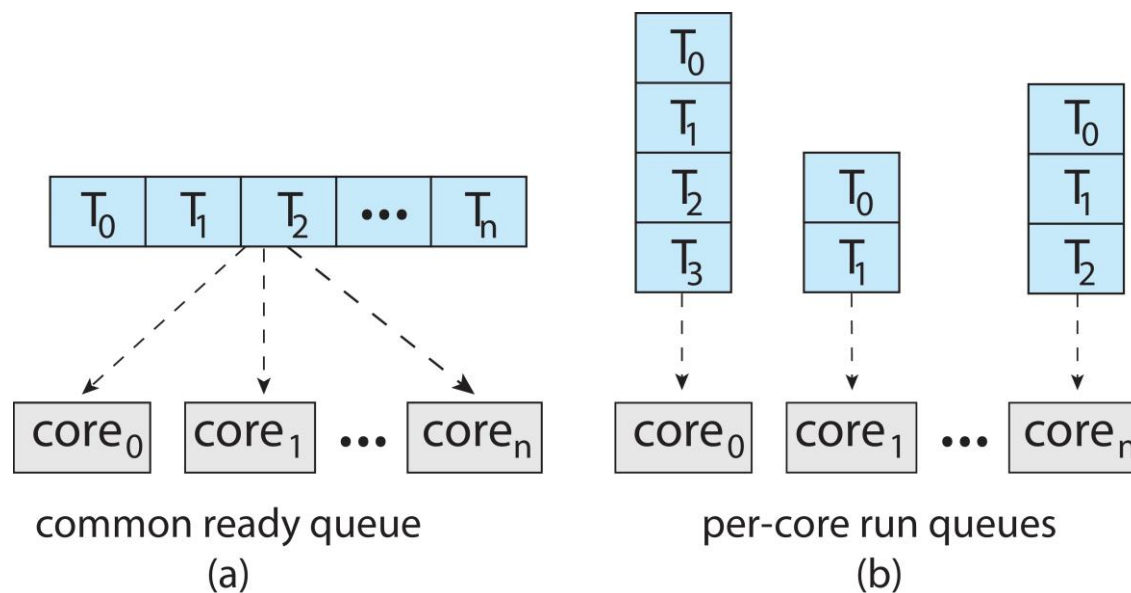
	FCFS	Round Robin	SPN	SRT	HRRN	Feedback
Selection Function	$\max[w]$	constant	$\min[s]$	$\min[s - e]$	$\max\left(\frac{w + s}{s}\right)$	(see text)
Decision Mode	Non-preemptive	Preemptive (at time quantum)	Non-preemptive	Preemptive (at arrival)	Non-preemptive	Preemptive (at time quantum)
Throughput	Not emphasized	May be low if quantum is too small	High	High	High	Not emphasized
Response Time	May be high, especially if there is a large variance in process execution times	Provides good response time for short processes	Provides good response time for short processes	Provides good response time	Provides good response time	Not emphasized
Overhead	Minimum	Minimum	Can be high	Can be high	Can be high	Can be high
Effect on Processes	Penalizes short processes; penalizes I/O-bound processes	Fair treatment	Penalizes long processes	Penalizes long processes	Good balance	May favor I/O-bound processes
Starvation	No	No	Possible	Possible	No	Possible

- Distinção entre threads no nível do utilizador e no nível do kernel
- Quando as threads são suportadas, threads são escalonadas e não processos
- Modelos muitos para um e muitos para muitos, a biblioteca de threads escala encadeamentos de nível de utilizador para execução em LWP
- Conhecido como âmbito de contenção de processo (PCS), uma vez que a competição de escalonamento está dentro do processo
- Normalmente feito por meio de prioridade definida pelo programador
- A thread do kernel escalonada no CPU disponível é o âmbito de contenção do sistema (SCS) - competição entre todas as threads no sistema.

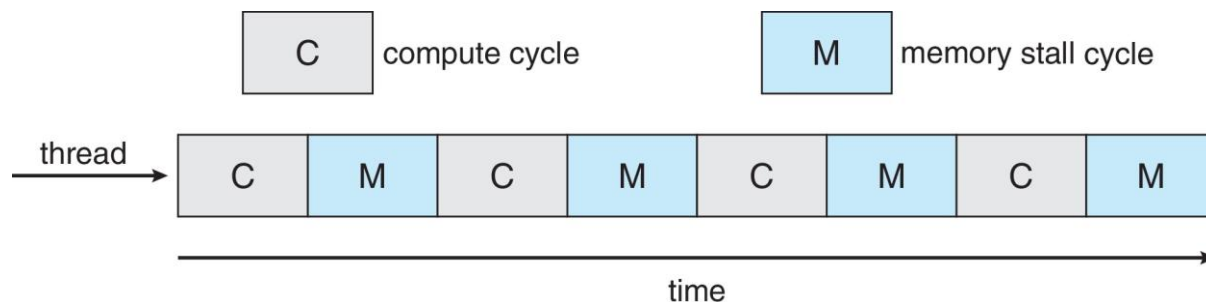
- O escalonamento de CPU é mais complexo quando múltiplos CPUs estão disponíveis.
- Multiprocessamento pode acontecer em qualquer uma das seguintes arquiteturas:
 - CPUs multicore
 - Cores Multithread
 - Sistemas NUMA
 - Multiprocessamento heterogéneo

Escalonamento de Múltiplo Processador

- Multiprocessamento simétrico (SMP) é quando cada processador faz auto-escalonamento
- Todas as threads podem estar numa fila de “pronto” comum (a)
- Cada processador pode ter a sua fila privada de threads (b)

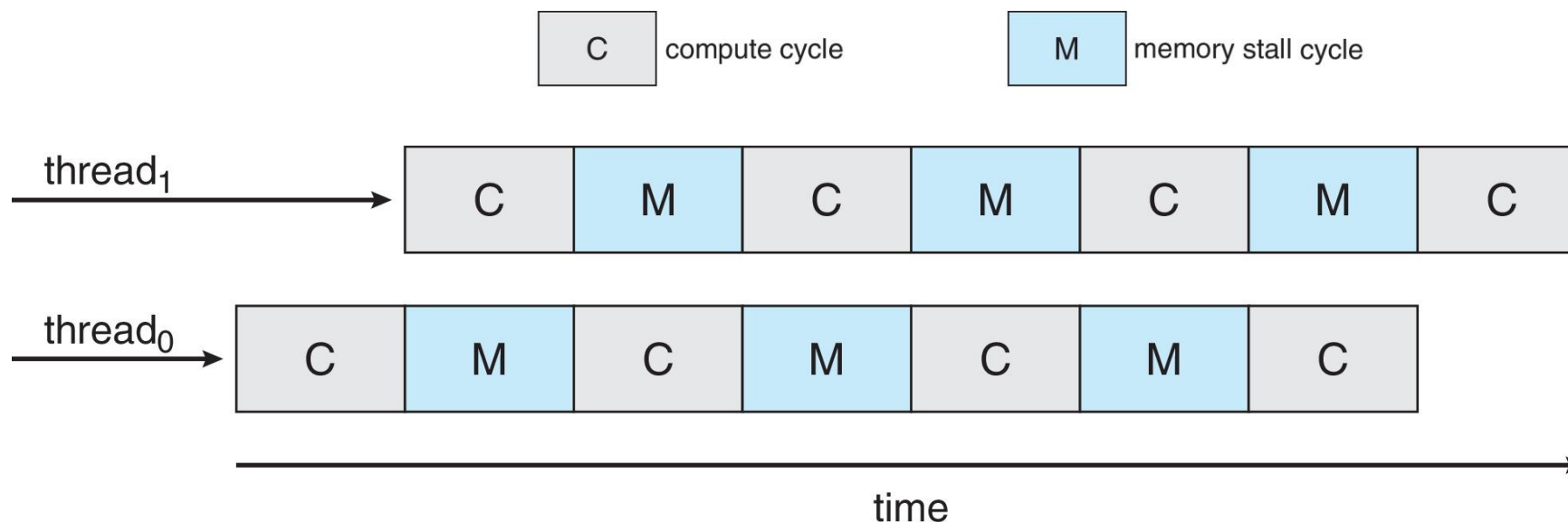


- Tendência recente para colocar múltiplos cores processadores no mesmo chip físico
- Mais rápido e consome menos energia
- Threads múltiplas por core também em crescimento:
- Aproveita o desacelerar de memória para progredir em outro segmento enquanto ocorre a recuperação de memória

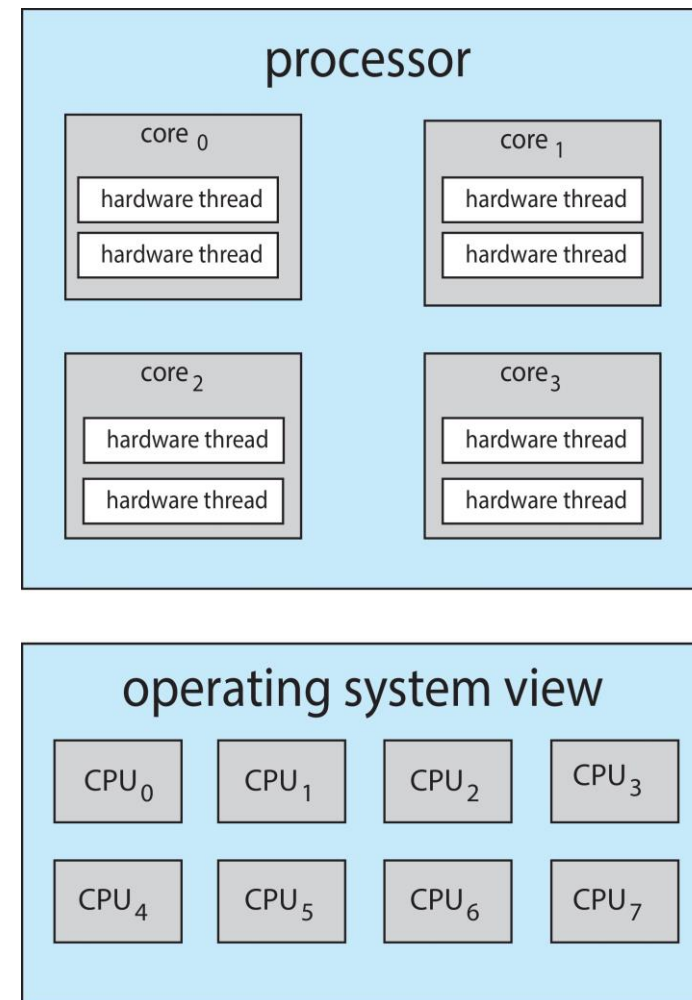


Sistema Multicore Multithread

- Cada core tem > 1 thread de hardware
- Se uma thread tem desacelerar de memória, muda para a outra thread

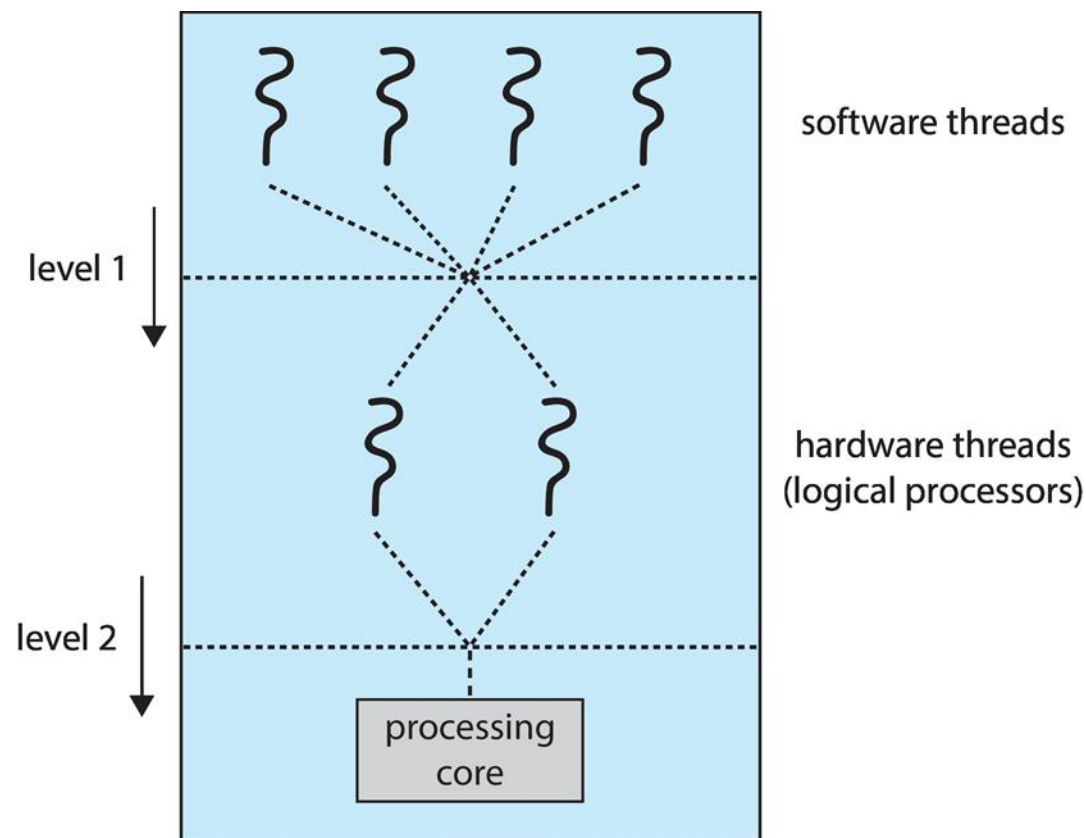


- Chip-multithreading (CMT) atribui cada thread de hardware de multiplo core (Intel chama-lhe hyperthreading)
- Num sistema quad-core system com 2 threads de hardware por core, o operating system considera 8 processadores lógicos



Dois níveis de escalonamento:

- O SO decide que thread de software corre em que CPU lógico
- Como cada core decide que thread de hardware corre no core físico



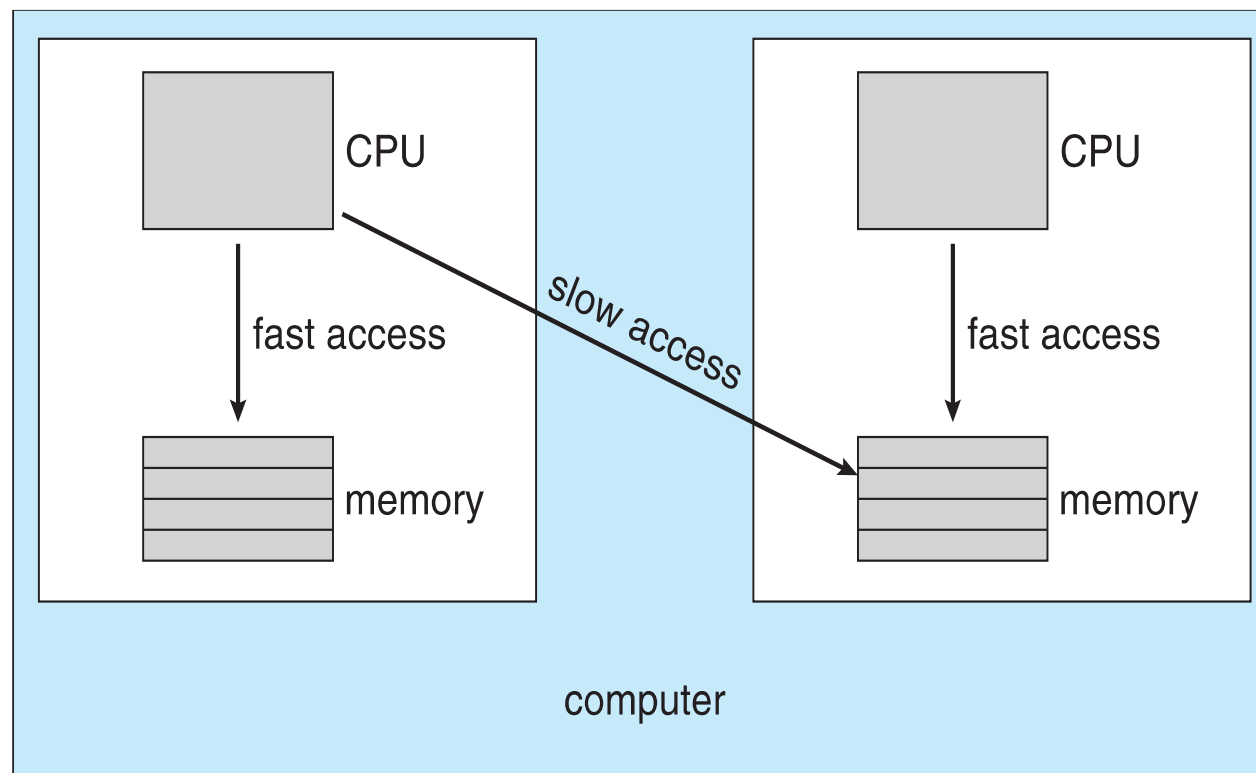
- Se estiver a usar SMP, é necessário manter os CPUs com carga eficiente
- **Balanceamento de carga** tenta manter a carga de trabalho uniformemente distribuída
- **Migração push** - verifica a carga de tarefas periódicas em cada processador e, se encontrada, empurra a tarefa da CPU sobrecarregada para outras CPUs
- **Migração pull** - processadores ociosos puxam tarefas de espera do processador ocupado

Sistema Multicore Multithread – Afinidade do processador

- Quando uma thread está a ser executada num processador, o conteúdo da cache desse processador armazena os acessos à memória dessa thread
- Referimo-nos a isso como uma thread com afinidade por um processador (ou seja, "afinidade do processador")
- O balanceamento de carga pode afetar a afinidade do processador, pois uma thread pode ser movida de um processador para outro para equilibrar as cargas, mas essa thread perde o conteúdo que tinha no cache do processador do qual foi movida
- **Afinidade suave (soft)** - o SO tenta manter uma thread em execução no mesmo processador, mas sem garantias
- **Afinidade rígida (hard)** - permite que um processo especifique um conjunto de processadores nos quais ele pode ser executado

NUMA e escalonamento do CPU

- Se o SO for compatível com NUMA, ele atribuirá bloqueios de memória à CPU em que a thread está sendo executada



Prioridades de processos no SO Windows

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

Algoritmo de escalonamento por SO

Sistema Operativo	Preempção	Algoritmo de escalonamento
FreeBSD	SIM	Escalonamento multinível de Fila
Linux Kernel > 2.6.23	SIM	Escalonamento completamente justo
Mac OS X	SIM	Escalonamento multinível de Fila
NetBSD	SIM	Escalonamento multinível de Fila
Windows 95, 98 e Me	METADE	Escalonamento preemptivo e cooperativo
Windows NT, Server, 10	SIM	Escalonamento multinível de Fila
Android	SIM	Escalonamento multinível de Fila

- O que é um processo?
- Quais são os vários tipos de processos e como se caracterizam?
- Como é feito o escalonamento de processos?
- Qual é o ciclo de vida de um processo?
- Que tipos de escalonamento existem?
- Como distinguir os algoritmos de escalonamento existentes?