Unidade 3 - Escalonamento da CPU

3.1 Conceitos Básicos

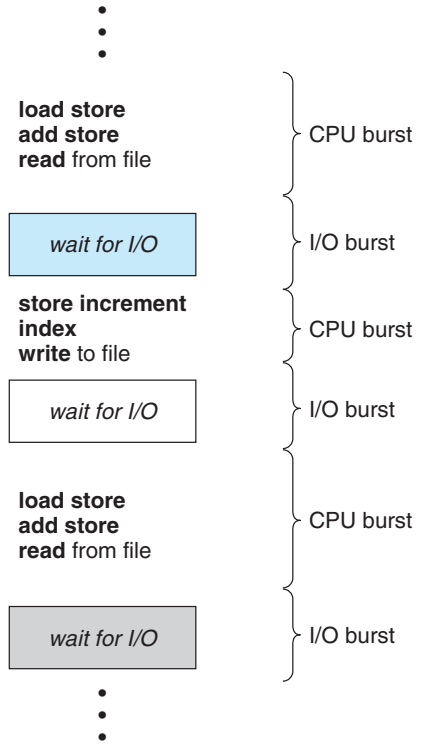
Conceitos Básicos (1/7)

- **Multiprogramação**
  - o SO mantém em RAM vários processos, alguns **prontos** a executar (aguardando por atribuição de CPU), alguns em **execução**, outros **bloqueados** (aguardando a conclusão de operações privilegiadas)
  - com vários processos em RAM, torna-se possível maximizar a utilização dos recursos: enquanto um ou mais processos executam (ocupando a(s) CPU(s)), outros podem estar a ser servidos por outros recursos de HW
  - com um só núcleo de execução (**single-core/mono-processamento**), haverá no máximo um só processo em execução, em cada instante; neste caso, a multiprogramação permite a execução **concorrente** de vários processos, suportando apenas **pseudo-paralelismo**
  - com vários núcleos de execução (**multi-core/multi-processamento**), é possível a execução simultânea de vários processos; nestas condições, a multiprogramação suporta **verdadeiro paralelismo**
  - qd um processo liberta um núcleo, o SO atribui-lhe um dos processos prontos, escolhido usando **critérios e algoritmos de escalonamento**

Conceitos Básicos (2/7)

Surtos de CPU e **Surtos de E/S**

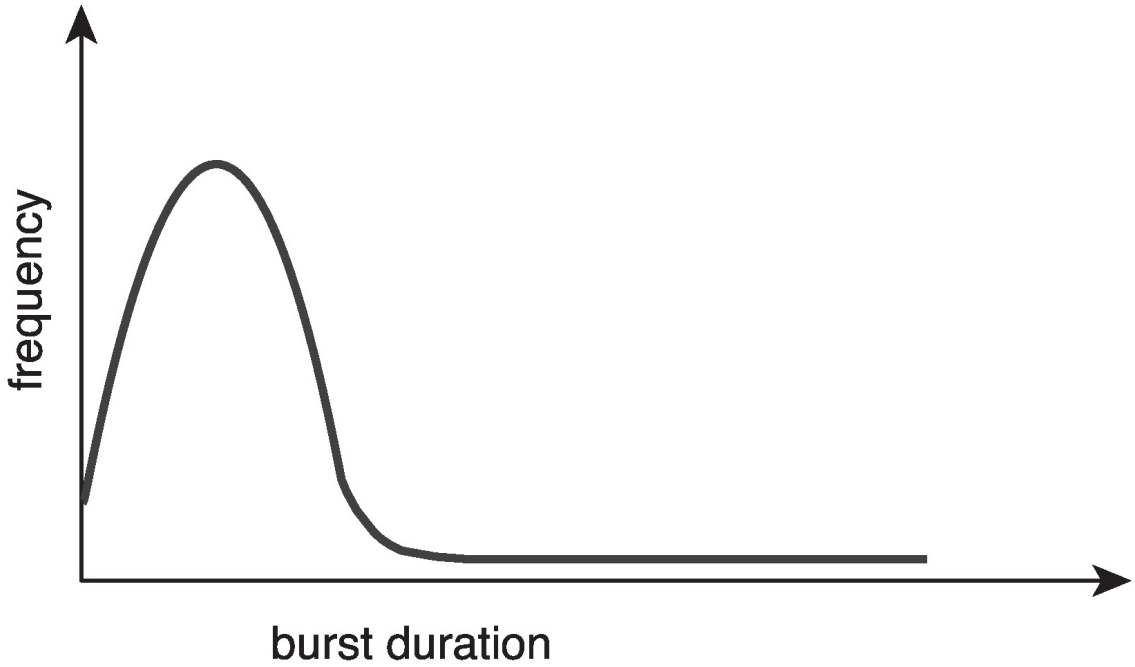
- na maior parte dos processos, o ciclo de vida consiste na alternância entre
  - períodos de execução (**surtos de CPU** ou **CPU bursts**)
  - períodos de bloqueio (**surtos de E/S** ou **I/O bursts**)



Conceitos Básicos (3/7)

Surtos de CPU e **Surtos de E/S** (cont.)

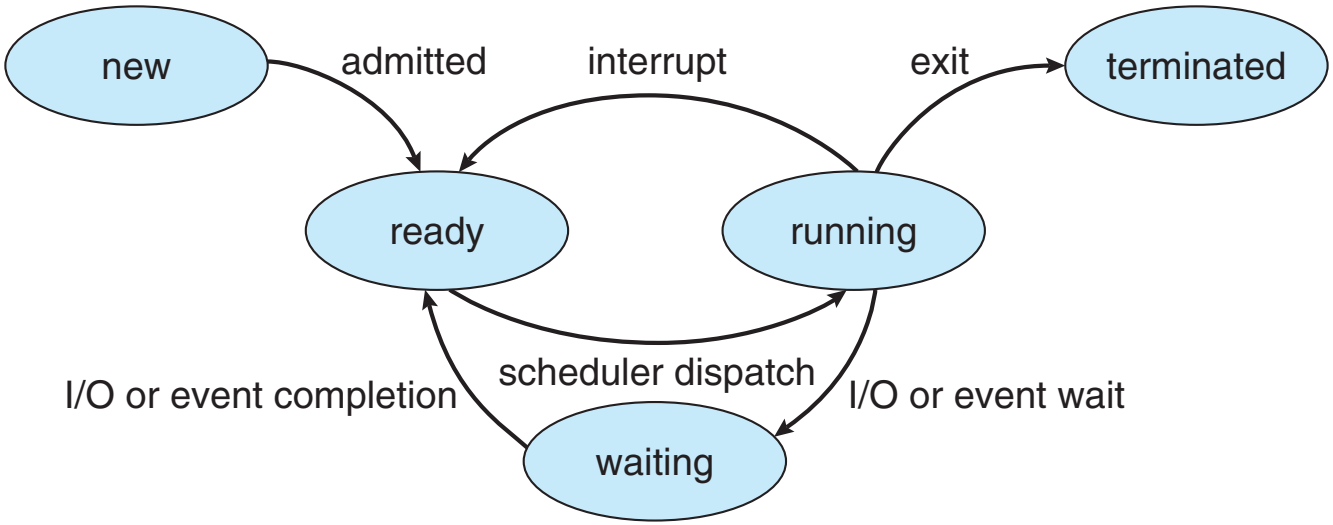
- embora os surtos de CPU possam variar bastante entre diferentes processos e diferentes sistemas, a sua distribuição segue um padrão
  - surtos de curta duração tendem a ser bastante frequentes
  - surtos de longa duração tendem a ser pouco frequentes
- processo **I/O bound**: muitos surtos de CPU, de curta duração
- processo **CPU bound**: poucos surtos de CPU, de longa duração



Conceitos Básicos (4/7)

Escalonador da CPU / Escalonador de Curto Termo

- escolhe um dos processos no estado *pronto*, para execução
- o escalonamento da CPU tem lugar quando um processo:
  1. muda do estado *em execução* para *bloqueado* (liberta a CPU)
  2. muda do estado *em execução* para *pronto* (liberta a CPU)
  3. muda do estado *bloqueado* para *pronto* (o serviço pedido pelo processo concluiu, sendo oportuno escolher um para execução, que pode ser ele)
  4. termina (liberta a CPU)



Conceitos Básicos (5/7)

Escalonador da CPU (cont.)

- o escalonamento nas situações 1 e 4 é **não–preemptivo** (**cooperativo**)
  - os processos libertam a CPU por iniciativa própria e é obrigatório o SO escolher um outro processo (de entre os *prontos*) para executar
  - notar que escalonamento apenas cooperativo dispensa um *timer*
  - exemplos: MS-DOS, Netware, Windows 3.x, Mac OS ≤ 9
- o restante escalonamento é **preemptivo** (**forçado**)
  - exemplos: UNIX, VMS, BSD,Windows $\geq$ 95/NT, Linux, Mac OS $\geq$ X
  - implica cuidados especiais no acesso concorrente a dados partilhados
    - um processo A não deve ceder a CPU a um processo B enquanto A estiver a aceder a recursos que possam ser modificados por A e/ou B
  - implica suportar preempção durante a execução de uma primitiva
    - kernel-não preemptivo: executa até ao fim a primitiva em curso, ou espera pela invocação de uma operação E/S, antes de comutar o contexto (ambas as soluções incompatíveis com SO de tempo real)
  - implica considerar interrupções que acedem a recursos partilhados
    - para evitar acesso concorrente, desligar as interrupções antes de uma secção crítica (desde que tenha poucas instruções) e religá-las após

Conceitos Básicos (6/7)

Escalonador da CPU (cont.)

- qual é a frequência da comutação de contexto num sistema real ?
- exemplo: comando `vmstat` e sistema de ficheiros `/proc` em Linux
  - o comando `vmstat` fornece informação de caráter geral

```
$ vmstat #sem parâmetros, mostra valores médios desde o arranque
... -system--   ------cpu-----
...  in   cs   us sy id wa st
...  184  428   2  1 97  0  0
```

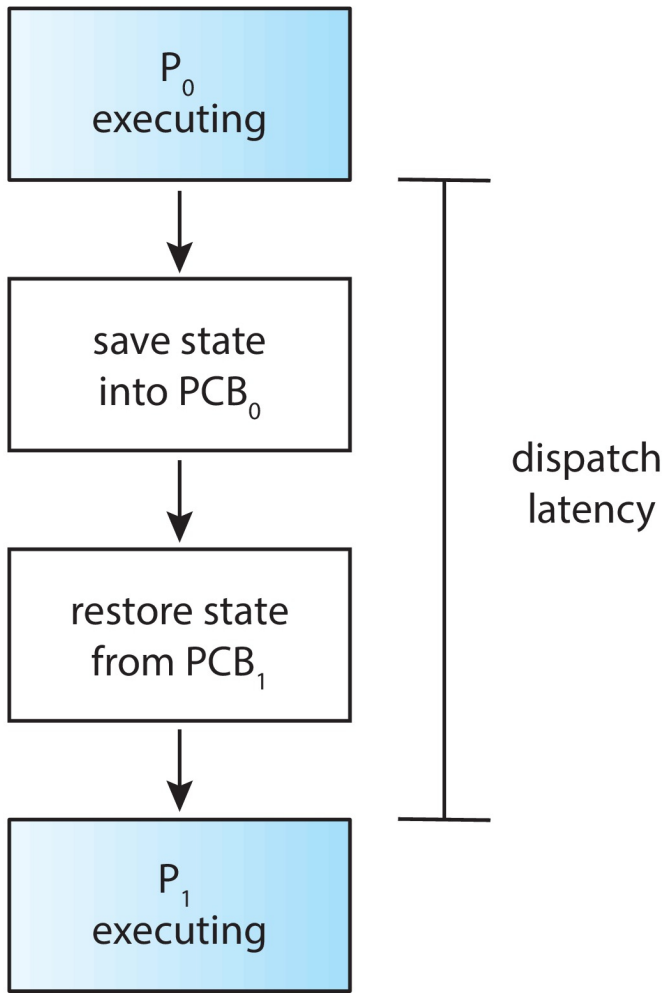
    - `in = number of interrupts per second, including the clock`
    - **`cs = number of context switches per second`**
    - `us = time running non-kernel code (user time); sy = time running kernel code (system time); id= time idle; wa= time waiting for IO`
  - o sistema de ficheiros `/proc` fornece informação particular de um processo

```
$ cat /proc/1/status
Name:   systemd
State:  S (sleeping)
Pid:    1
PPid:   0
...
voluntary_ctxt_switches:  94086
nonvoluntary_ctxt_switches:  662
```

Conceitos Básicos (7/7)

Dispatcher (Despachante)

- invocado após expirar a fatia de tempo do processo que estava em execução
  - modo supervisor ativado automaticamente pela interrupção do timer
  - guarda o estado interno da CPU no PCB do processo que foi parado
  - invoca o Escalonador da CPU
- invocado pelo Escalonador da CPU
  - redefine o estado da CPU com base no PCB do processo seleccionado
  - ativa o modo utilizador
  - salta para a instrução apontada pelo PC do processo selecionado
- *latência de despacho* (*dispatch latency*):
  - tempo que o *dispatcher* demora a parar um processo e arrancar outro



3.2 Critérios de Escalonamento

Critérios de Escalonamento (1/2): Métricas de Comparação

- Métricas Globais (sob o ponto de vista do sistema):
  - **Utilização da CPU** – manter a CPU tão ocupada quanto possível
  - **Throughput** – nº de processos que concluem a execução por unidade de tempo; processos longos/curtos ⇒ baixo/elevado *throughput*
- Métricas Locais (sob o ponto de vista de cada processo):
  - **Tempo de Retorno** (*turnaround*) – tempo que decorre desde a submissão do job/processo à conclusão da sua execução (tempo consumido na fila de jobs + fila de prontos + execução + filas E/S)
  - **Tempo de Espera** (*waiting*) – tempo consumido na fila dos prontos
    - este tempo é diretamente afetado pelo algoritmo de escalonamento (este algoritmo não afeta o tempo gasto em execução ou em E/S)
  - **Tempo de Resposta** (*response*) – tempo desde a submissão do job/processo até ao disparo do seu 1º *output* (não inclui o tempo gasto no *output*, com o processo bloqueado na fila do dispositivo em causa)
    - é a métrica mais relevante em ambientes interativos

Critérios de Escalonamento (2/2): Objetivos Gerais

- Max utilização da CPU
- Max throughput
- Min tempo de retorno
- Min tempo de espera
- Min tempo de resposta
- Uma outra hipótese será otimizar a média destes valores (insensível a extremos)
- Em sistemas interativos será mais importante minimizar a variância do tempo de resposta: um sistema com tempo de resposta previsível mas razoável, será preferível a outro com tempo de resposta médio baixo, mas muito variável

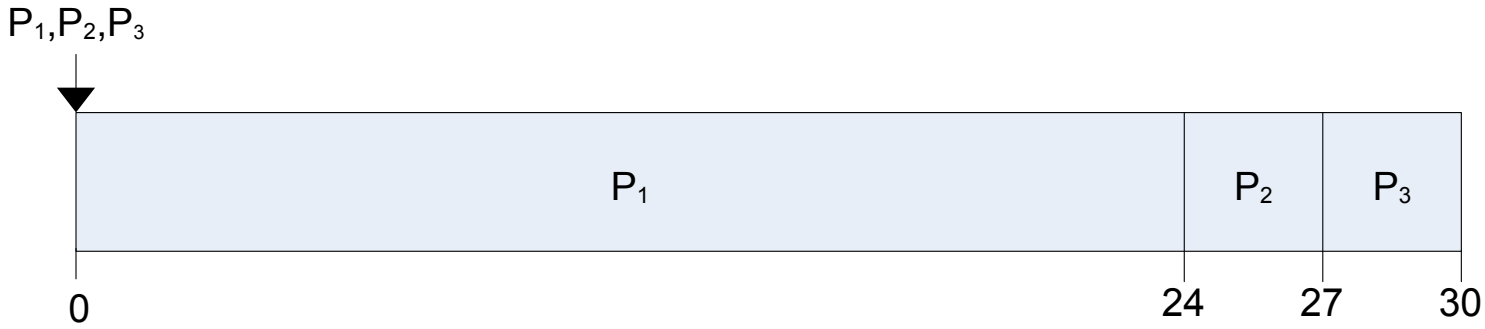
3.3 Algoritmos de Escalonamento

Algoritmos de Escalonamento (1/19): Escalonamento FCFS

- **FCFS** = *First–Come, First–Served*
- os processos ocupam a CPU pela ordem de entrada na fila *ready*
- a duração dos surtos de CPU dos processos é desconhecida à partida
  - salvo exceções (*), os processos ocupam a CPU enquanto quiserem
- é um algoritmo **não–preemptivo**, inadequado a sistemas interativos
  - os processos não são obrigados a dividir o tempo de CPU com outros
- um processo liberta a CPU
  - de forma **voluntária**: termina, ou invoca uma operação privilegiada (bloqueia e após a conclusão da operação regressa à fila *ready*)
  - (*) de forma **forçada**: devido a uma interrupção (e.g., conclusão de operação privilegiada de outro processo), regressando à fila *ready*
- resulta em tempos de espera médios elevados, sendo muito sensível à ordem de entrada dos processos na fila *ready* (ver exemplo a seguir)

Algoritmos de Escalonamento (2/19): Escalonamento FCFS

- **Exemplo**: fila *ready* com 3 processos e ordem de chegada P_1, P_2, P_3



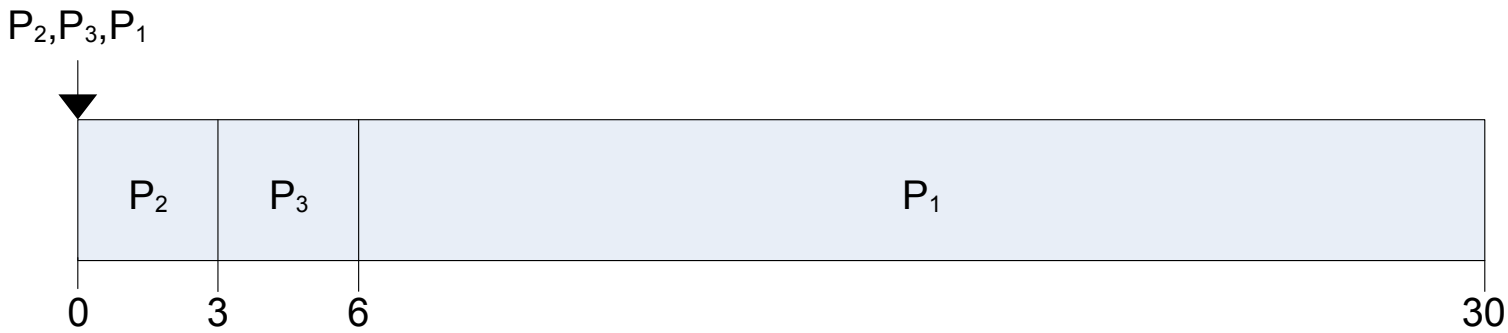
Process	Arrival Order	Total CPU Time
P_1	1	24
P_2	2	3
P_3	3	3

- tempos de **espera**: P_1=0 ms; P_2=24 ms; P_3=27 ms; média=**17 ms**
- tempos de **retorno**: P_1=24 ms; P_2=27 ms; P_3=30 ms: média=**27 ms**

Neste exemplo e no seguinte: i) todos os processos já estão na fila *ready* no instante 0 (mas entraram por uma certa ordem), ii) cada processo necessita de um só surto de CPU, e iii) a duração desse surto só se sabe *após* ter saído da CPU.

Algoritmos de Escalonamento (3/19): Escalonamento FCFS

- **Exemplo**: = anterior, mas com ordem de chegada P_2, P_3, P_1



Process	Arrival Order	Total CPU Time
P_1	3	24
P_2	1	3
P_3	2	3

- tempos de **espera**: P_1=6 ms; P_2=0 ms; P_3=3 ms; média=**3 ms**
- tempos de **retorno**: P_1=30 ms; P_2=3 ms; P_3=6 ms; média=**13 ms**
- bastante melhor que no cenário anterior !

Algoritmos de Escalonamento (4/19): Escalonamento FCFS

Efeito de Comboio:

- o FCFS é propenso ao chamado **efeito de comboio**, que resulta na ausência de sobreposição entre computação e E/S e portanto numa subutilização de recursos
- cenário propício:
  - sejam n processos *IO bound* e 1 processo *CPU bound*
  - com o processo *CPU bound* em execução, os n processos *IO bound* acabam por se concentrar na fila *ready* esperando por CPU (efeito de comboio) $\Rightarrow$ CPU plenamente ocupada e dispositivos E/S ociosos
  - quando o processo *CPU bound* bloqueia numa operação E/S, os *IO bound* ocupam a CPU e libertam-na rapidamente, bloqueando todos em operações E/S $\Rightarrow$ CPU ociosa e dispositivos E/S todos ocupados
  - o processo *CPU bound* conclui a operação E/S, transita para a fila *ready* e ocupa a CPU; quando os processos *IO bound* concluirem as suas operações E/S, concentram-se de novo todos na fila *ready*

Algoritmos de Escalonamento (5/19): Escalonamento SJF

- **SJF** = *Shortest–Job–First* (ou *Shortest–Next–CPU–Burst*)
- associa a cada processo a duração prevista do próximo surto de CPU
  - mas uma vez em execução, o processo *não* tem de respeitar essa previsão, podendo abandonar a CPU antes/depois do tempo previsto
- atribui a CPU ao processo que necessita do menor surto de CPU
  - empate: usar ordem de chegada ou outro critério secundário definido
- um processo liberta a CPU
  - pelas mesmas razões que no algoritmo FCFS (ver slide 14)
  - caso surja na fila *ready* um processo mais curto (SJF preemptivo)
- **SJF preemptivo** (ou SRTF: *Shortest–Remaining–Time–First*)
  - se surgir um processo com surto de CPU previsto inferior ao que resta ao processo em execução, a CPU é dada ao processo recém chegado
- o SJF é ótimo – resulta no tempo de espera médio mínimo: movendo processos curtos para antes dos longos, o tempo de espera de um processo curto decresce mais do que cresce o de um processo longo

Algoritmos de Escalonamento (6/19): Escalonamento SJF

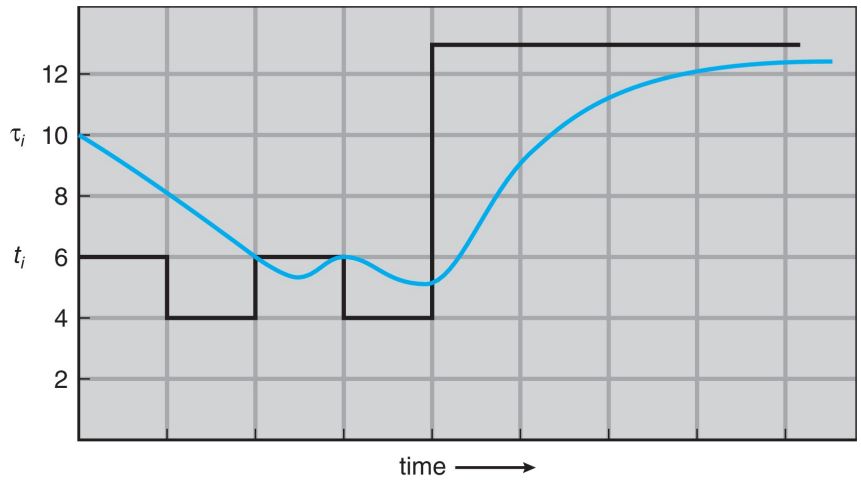
Previsão do próximo surto de CPU

- previsão inicial: fornecida pelo utilizador (jobs) ou definida pelo SO
- previsão seguinte: apenas necessária se o processo regressar à fila *ready*
  - função da previsão e duração efetiva do anterior surto de CPU
  - τ_{i+1} = duração prevista do próximo surto de CPU
  - $\tau_{i+1} = \alpha t_i + (1 - \alpha)\tau_i$
  - t_i = duração efetiva (medida) do anterior surto de CPU
  - τ_i = duração prevista do anterior surto de CPU
  - α = peso da última medida ($0 \leq \alpha \leq 1$)
  - $1 - \alpha$ = peso da última previsão ($0 \leq \alpha \leq 1$)
- previsão seguinte simplificada (exemplos práticos e exercícios):
  - $\tau_{i+1} = \tau_i - t_i$ se $t_i \leq \tau_i$ (se sobrou tempo da anterior previsão, a próxima previsão é o tempo restante da anterior previsão)
  - $\tau_{i+1} = t_i$ se $t_i > \tau_i$ (se a anterior previsão foi excedida, a próxima previsão é o tempo efetivo do anterior surto de CPU)

Algoritmos de Escalonamento (7/19): Escalonamento SJF

Previsão do próximo surto de CPU

- $\tau_{i+1} = \alpha t_i + (1 - \alpha)\tau_i$
  - média móvel exponencial
- $\alpha = 0$
  - $\tau_{i+1} = \tau_i$
  - a história recente não conta
- $\alpha = 1$
  - $\tau_{i+1} = t_i$
  - só conta a história recente
- $\alpha = 0.5$: ver exemplo da figura



CPU burst (t_i)	6	4	6	4	13	13	13	...
"guess" (τ_i)	10	8	6	6	5	9	11	12	...

- note-se que, expandido-se a expressão $\tau_{i+1} = \alpha t_i + (1 - \alpha)\tau_i$ obtém-se

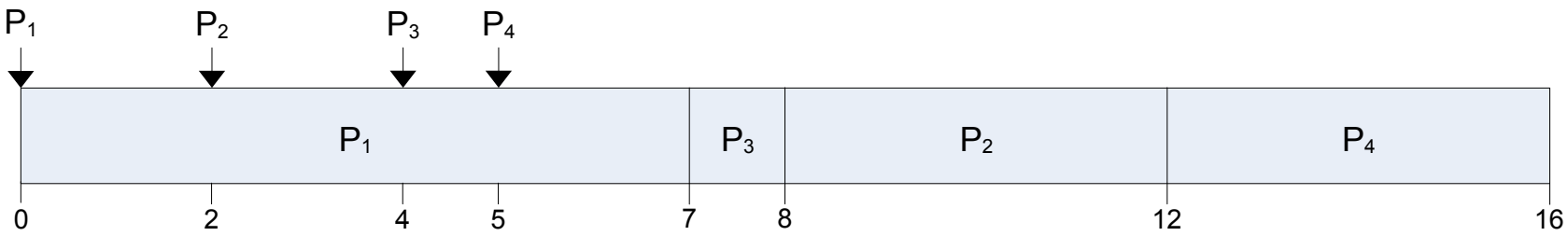
$$\tau_{i+1} = \alpha t_i + (1 - \alpha)\alpha t_{i-1} + ... + (1 - \alpha)^j \alpha t_{i-j} + ... + (1 - \alpha)^{i+1} \tau_0$$

  - sendo $\alpha \leq 1$ e $(1 - \alpha) \leq 1$, cada termo tem menos peso que o anterior (a história passada tem influência, embora cada vez menos relevante)

Algoritmos de Escalonamento (8/19): Escalonamento SJF

- Exemplo de SJF não–preemptivo:

		CPU Bursts(ms)		
Process	Arrival Instants	Predicted	Effective	Total
P_1	0	7	7	7
P_2	2	4	4	4
P_3	4	1	1	1
P_4	5	4	4	4



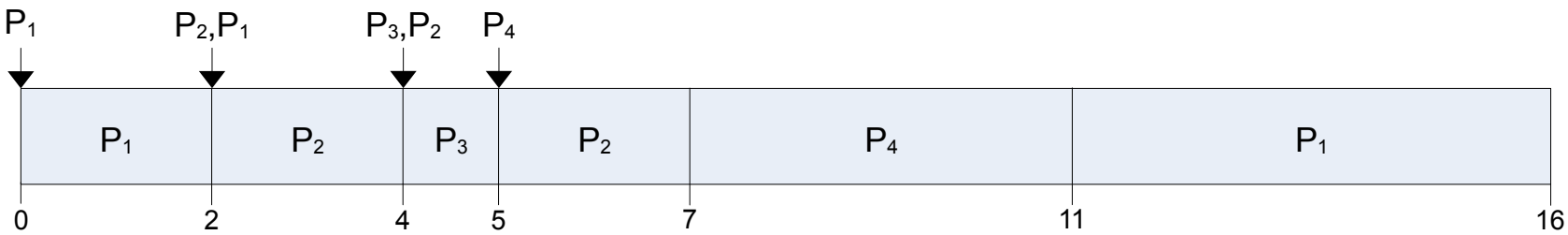
- tmp. **espera médio**=(0+(8-2)+(7-4)+(12-5))/4=**4 ms** (FCFS=4.75 ms)
- tmp. **retorno médio**=((7-0)+(12-2)+(8-4)+(16-5))/4=**8 ms** (FCFS=8.75 ms)

Neste exemplo e no seguinte: i) os processos vão chegando à fila *ready* em diferentes instantes, ii) cada processo traz consigo uma previsão inicial de um só surto de CPU, iii) nenhum processo excede a previsão inicial.

Algoritmos de Escalonamento (9/19): Escalonamento SJF

- Exemplo de SJF preemptivo (SRTF) com previsão simplificada:

		CPU Bursts (ms)		
Process	Arrival Instants	Predicted	Effective	Total = $\sum$ Effective
P_1	0,2	7,5=7-2	2,5	2+5=7
P_2	2,4	4,2=4-2	2,2	2+2=4
P_3	4	1	1	1
P_4	5	4	4	4



- tempo de **espera médio** = ((0+11-2)+(0+5-4)+0+(7-5))/4 = **3 ms**
- tempo de **retorno médio** = ((16-0)+(7-2)+(5-4)+(11-5))/4 = **7 ms**
- melhoria ligeira em relação ao cenário anterior

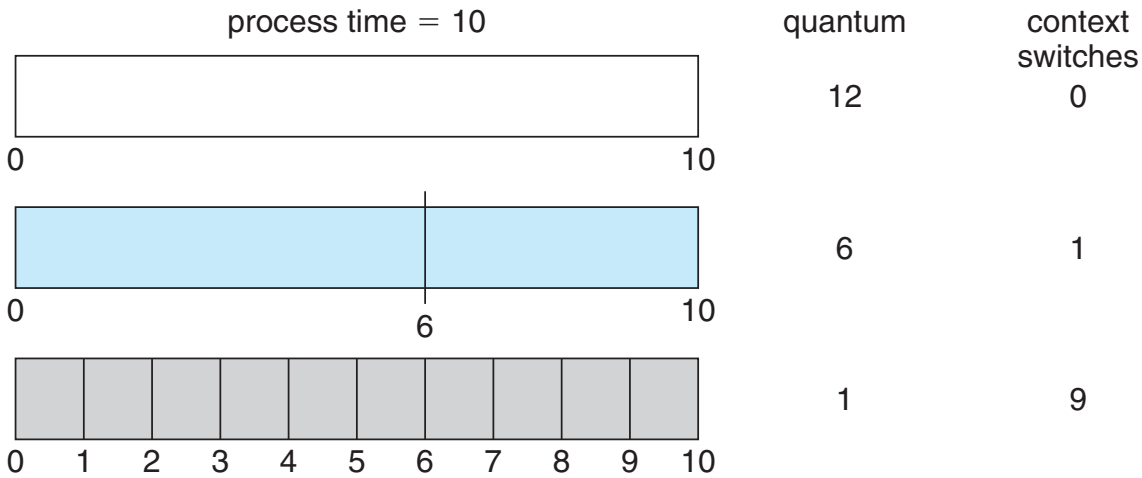
Algoritmos de Escalonamento (10/19): Round Robin (RR)

- **RR** $\Leftrightarrow$ **FCFS** com um limite máximo à duração do surto de CPU
- os processos ganham a CPU pela ordem de entrada na fila *ready*
- define-se uma duração máxima dos surtos de CPU: *quantum* (q)
- um processo liberta a CPU
  - pelas mesmas razões que no algoritmo FCFS (ver slide 14)
  - via interrupção do *timer*, se for atingida a duração máxima do surto
- é um algoritmo **preemptivo**, adequado a sistemas *time-sharing*
  - os processos são obrigados a dividir o tempo de CPU com outros
  - a duração máxima dos surtos de CPU é tipicamente pequena
- o RR traduz-se em *tempos de resposta* reduzidos (beneficiando a interatividade), e *tempos de retorno* e *tempos de espera* acrescidos

Algoritmos de Escalonamento (11/19): Round Robin (RR)

Fatia de Tempo (quantum)

- com n processos na fila *ready* e definido um certo *quantum* q, cada processo recebe $1/n$ do tempo de CPU, em rondas de até q unidades de tempo cada
- antes de ganhar a CPU, um processo aguarda na fila *ready* até $(n-1)q$ unidades de tempo (os outros processos podem ou não consumir todo o seu *quantum*)
- o desempenho do algoritmo RR depende basicamente do valor do *quantum*:
  - q demasiado grande $\Rightarrow$ RR não se diferencia do FCFS
  - q demasiado pequeno $\Rightarrow$ elevada sobrecarga da comutação de contexto
  - dimensionamento de q: a) tempo de comutação de contexto deve ser $\lessapprox$ 10% de q; b) 80% dos surtos de CPU devem ser inferiores a q

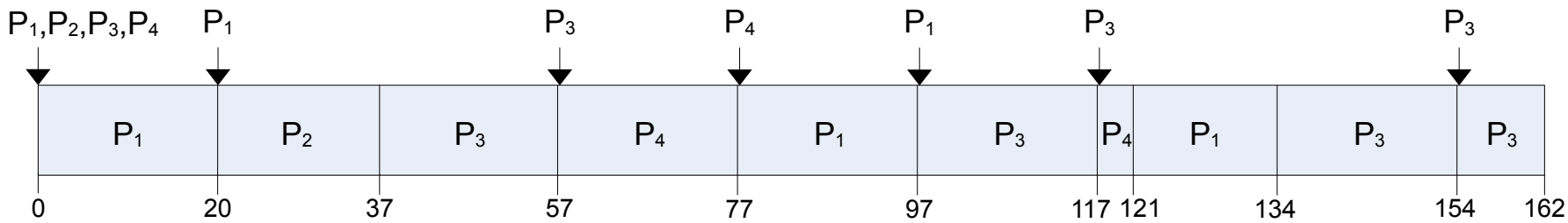


Algoritmos de Escalonamento (12/19): Round Robin (RR)

Exemplo (q=20)

Todos os processos prontos no instante 0, mas entraram na fila por uma certa ordem. A duração do tempo total de CPU necessário a cada processo só se sabe no fim da sua execução, a qual poderá envolver vários surtos de CPU com duração máxima de q.

Process	Arrival Instants	CPU Bursts (ms)	Total CPU Time (ms)
P_1	0, 20, 97	20, 20, 13	$\sum = 53$
P_2	0	17	$\sum = 17$
P_3	0, 57, 117, 154	20, 20, 20, 8	$\sum = 68$
P_4	0, 77	20, 4	$\sum = 24$



Notar que sobrando um só processo (P_3), o tempo continuará a dividir-se em *quantums*.

- tmp. **espera médio**: ((0+77-20+121-97)+(20-0)+(37-0+97-57+134-117)+(57-0+117-77))/4 = **73 ms**
- tmp. **retorno médio**: ((134-0)+(37-0)+(162-0)+(121-0))/4 = **113.5 ms**
- tmp. de **espera médio do SJF não preemptivo**: ((17+24) + 0 + (17+24+53) + 17)/4 = **38 ms**

Algoritmos de Escalonamento (13/19): Prioridades Genéricas

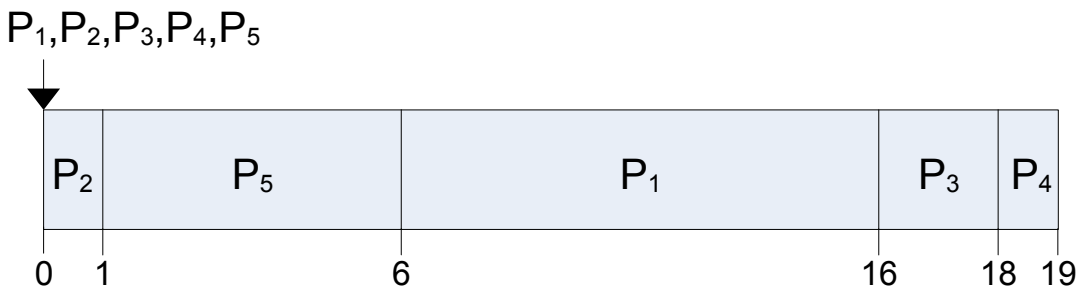
- associa a cada processo um **número** representativo da sua **prioridade**
  - **prioridades internas**: definidas pelo SO; deadlines de execução, surtos de CPU, requisitos de memória RAM, natureza *CPU bound / IO bound*, etc.
  - **prioridades externas**: externas ao SO; organizacionais, monetárias, etc.
- atribui a CPU ao processo mais prioritário
  - empate: usar ordem de chegada ou outro critério secundário fornecido
- FCFS e SJF: casos particulares de escalonamento por prioridades
  - FCFS: prioridade definida pela ordem de chegada à *fila ready*
    - maior prioridade dada aos processos quem chega primeiro
  - SJF: prioridade definida pelo próximo surto de CPU previsto
    - maior prioridade dada aos processos quem requerem menos tempo
- variantes: **preemptiva** e **não–premptiva**
- um processo liberta a CPU:
  - pelas mesmas razões que no SJF, adaptadas ao tipo de prioridade

Algoritmos de Escalonamento (14/19): Prioridades Genéricas

- Problema: processos de reduzida prioridade podem nunca executar (**Starvation**)
- Solução: incrementar, periodicamente, a prioridade desses processos (**Aging**)

- **Exemplo** (sem preempção; todos os processos na fila ready no instante 0; um só surto de CPU por processo, de duração conhecida só após executar)

Process	Priority	Total CPU Time (ms)
P_1	3	10
P_2	1	1
P_3	4	2
P_4	5	1
P_5	2	5

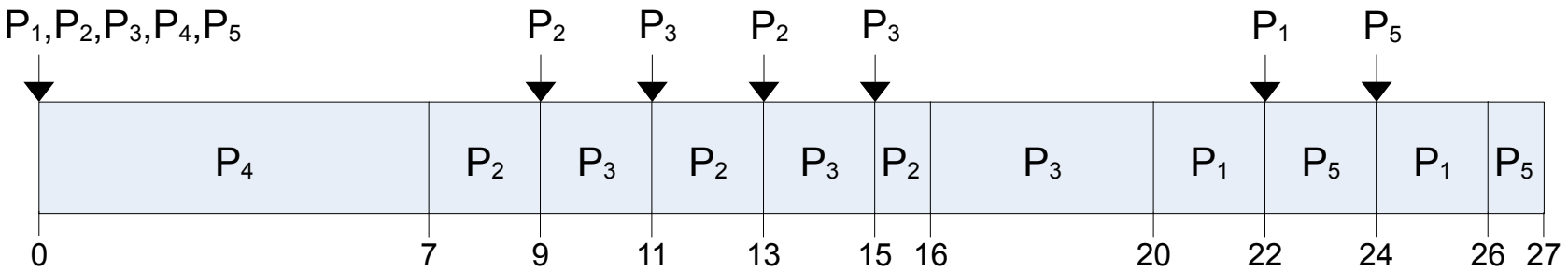


- tempo de **espera médio**: $(6+0+16+18+1)/5 = \mathbf{8.2\ ms}$
- tempo de **retorno médio**: $(16+1+18+19+6)/5 = \mathbf{12\ ms}$

RR combinado com Prioridades Genéricas

- processos de maior prioridade são executados sem preempção
- processos de igual prioridade são executados com preempção sob RR
- **Exemplo** (q=2 ms; todos os processos prontos no instante 0):

Process	Arrival Instants	Priority	CPU Bursts (ms)	Total CPU Time (ms)
P_1	0, 22	3	2, 2	$\sum$=4
P_2	0, 9, 13	2	2, 2, 1	$\sum$=5
P_3	0, 11, 15	2	2, 2, 4	$\sum$=8
P_4	0	1	7	$\sum$=7
P_5	0, 24	3	2, 1	$\sum$=3

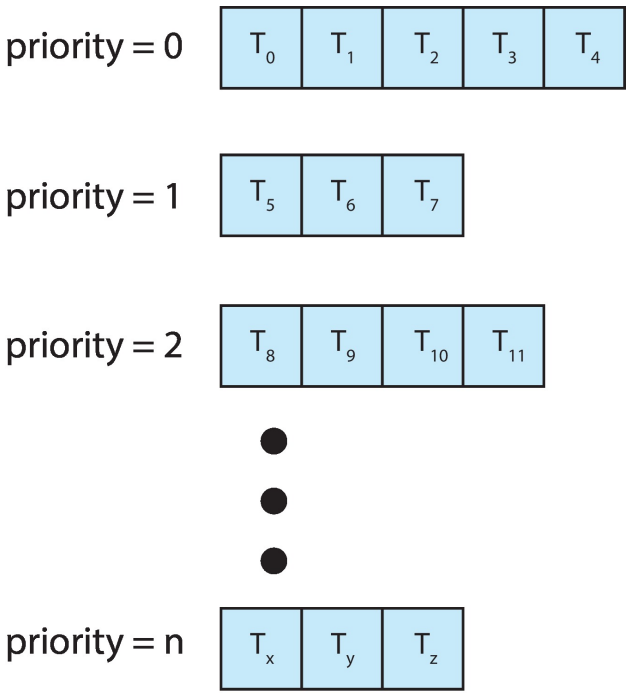


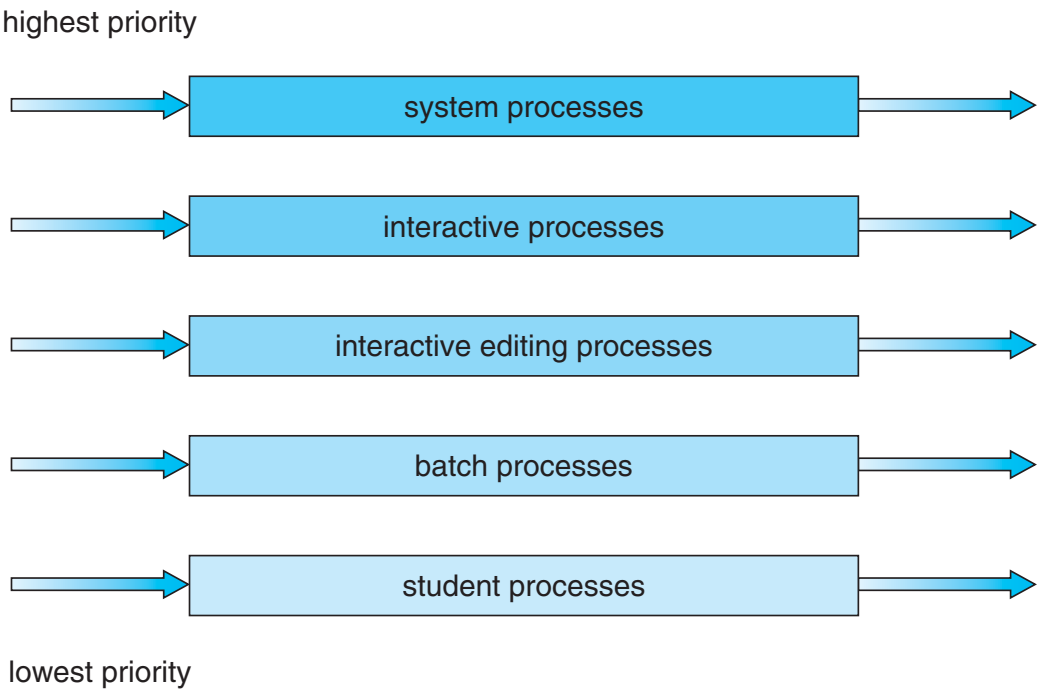
- tempo de **espera médio**: $((20+2)+(7+2+2)+(9+2+1)+0+(22+2))/5 = \mathbf{13.8\ ms}$
- tempo de **retorno médio**: $(26+16+20+7+27)/5 = \mathbf{19.2\ ms}$

Algoritmos de Escalonamento (16/19): Filas Multinível (FM)

- a fila dos processos prontos (fila ready) é dividida em filas separadas:
  - fila(s) para processos interactivos, que executam em 1º plano (*foreground*)
  - fila(s) para não-interactivos, que executam em 2º plano (*background*)
- um processo não pode mudar de uma fila ready para outra fila ready
- cada fila ready tem o seu próprio algoritmo de escalonamento:
  - RR para processos em *foreground*
  - FCFS para processos em *background*
- é feito escalonamento entre as filas:
  - escalonamento de **prioridade fixa**: servir todos os processos em *foreground* e só depois os processos em *background* $\Rightarrow$ possibilidade de *starvation*
  - escalonamento por **fatias de tempo**: cada fila recebe um certo tempo de CPU, a distribuir por todos os processos da fila (*e.g.*, 80% para processos em *foreground* sob RR e 20% para processos em *background* sob FCFS)

Algoritmos de Escalonamento (17/19): Filas Multinível (FM)



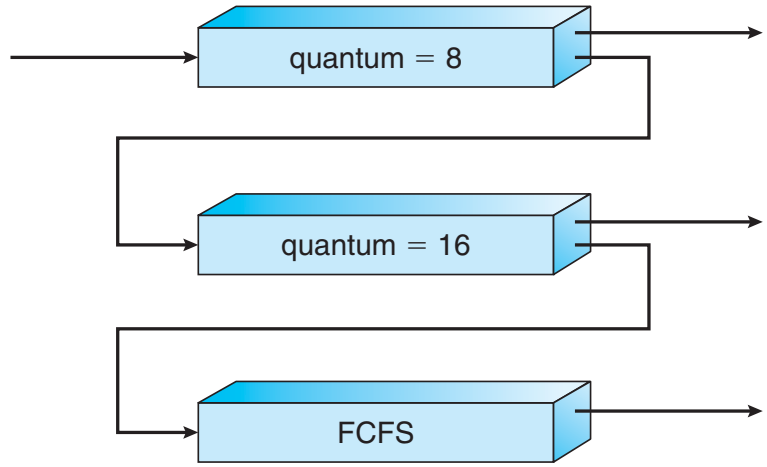


Algoritmos de Escalonamento (18/19): FM com Feedback

- permite-se que um processo transite entre diferentes filas de escalonamento
- as filas de maior prioridade serão para processos interativos ou *IO–bound*
  - esses processos libertam rapidamente a CPU ...
- processos *CPU–bound* tenderão a ser despromovidos para filas de menor prioridade
  - *Aging*: promove processos *CPU-bound* antigos para filas de maior prioridade
- um escalonador de Filas Multinível com Feedback tem como parâmetros:
  - número de filas
  - algoritmo de escalonamento para cada fila
  - método usado para determinar quando promover um processo
  - método usado para determinar quando despromover um processo
  - método usado para determinar em que fila entra um novo processo

Algoritmos de Escalonamento (19/19): FM com Feedback

- exemplo com três filas:
  - Q_0 – FCFS com q=8 ms
  - Q_1 – FCFS com q=16 ms
  - Q_2 – FCFS

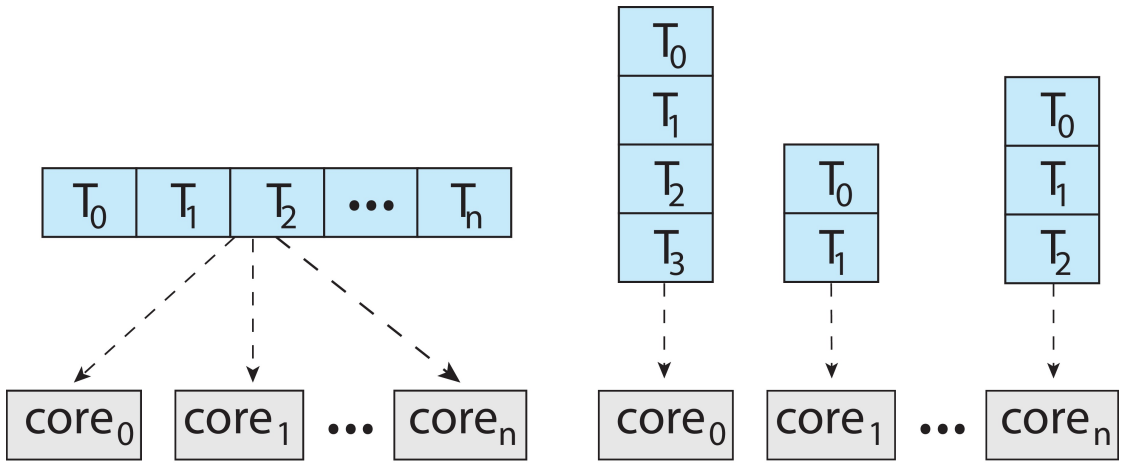


- um novo processo entra sempre na fila Q_0; quando chegar a sua vez de ganhar CPU, o processo recebe 8 ms; se não terminar em 8 ms, transita para a fila Q_1
- os processos em Q_1 só são escalonados se Q_0 estiver vazia; quando escalonado, um processo em Q_1 recebe 16 ms; se não conseguir terminar, transita para a fila Q_2
- em Q_2, os processos são escalonados sob FCFS puro, executando até ao fim
- problema: *starvation* – os processos de uma fila só serão escalonados se as filas de maior prioridade estiverem vazias; solução: *aging* (promoção) desses processos
- variantes com Q_0 e Q_1 geridos com RR: a) um processo é despromovido se regressar à fila atual mais do que n vezes; b) um processo é atribuído à fila com o quantum mais próximo do valor (médio) dos seus surtos de CPU

3.4 Escalonamento em Multiprocessadores

Escalonamento em Multiprocessadores (1/6)

- o escalonamento é mais complexo na presença de múltiplas CPUs ...
  - exemplo: processos que necessitam de um certo dispositivo E/S só podem executar no CPU cujo barramento foi ligado ao dispositivo
- c/ **multiprocess. assimétrico**: apenas o CPU mestre faz escalonamento
- c/ **multiprocess. simétrico**: cada CPU faz o seu auto-escalonamento
  - uma fila por processador (b): possibilidade de filas vazias
  - uma única fila, partilhada (a): necessidade de evitar i) escalonamento duplo do mesmo processo, ii) não-escalonamento de um processo

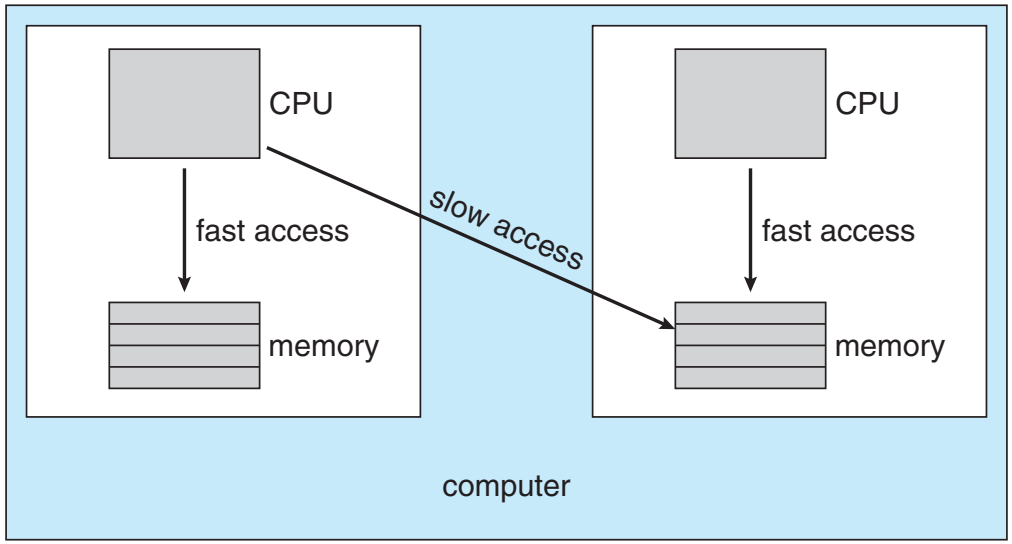


Escalonamento em Multiprocessadores (2/6)

- **balanceamento de carga**
  - em sistemas SMP é importante distribuir a carga uniformemente pelas CPUs; caso contrário, haverá CPUs sobrecarregadas e outras ociosas
    - necessário com filas de escalonamento separadas (uma fila por CPU)
  - **push migration**: um processo especial, de monitorização, vigia a carga dos CPUs e, se sobrecarregados, migra processos para outras CPUs
  - **pull migration**: CPUs ociosas "roubam" processos às sobrecarregadas
  - ambas as técnicas podem ser usadas de forma combinada
    - exemplo, em Linux: *push migration* de 200 ms em 200 ms, mais *pull migration* sempre que a fila de escalonamento de uma CPU fica vazia
  - o balanceamento de carga colide com as políticas de afinidade de CPU

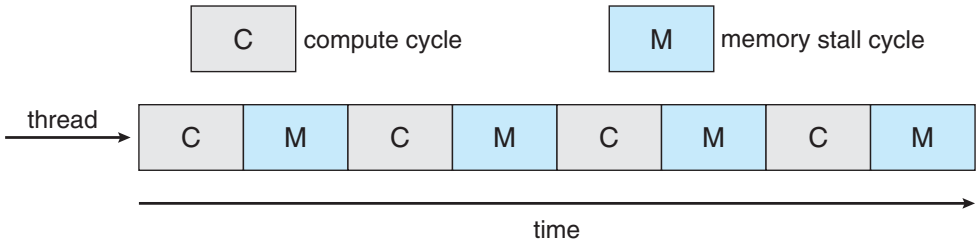
Escalonamento em Multiprocessadores (3/6)

- **afinidade ao processador** (*processor afinity*): o processo "prefere" uma certa CPU
  - objetivo: evitar a penalização envolvida na migração entre CPUs (essa migração implica invalidar as caches da 1a CPU e repovoar as da 2a CPU)
    - *soft affinity* / *hard affinity* : toleram / proibem migração
    - *process sets* (Solaris) : migração entre CPUs do mesmo conjunto
  - a afinidade é influenciada pela arquitetura de memória do sistema
    - numa arquitetura NUMA (Non-Uniform Memory Access), uma CPU acede mais rapidamente à memória RAM próxima; se um processo tem afinidade a uma CPU, a memória dada ao processo deve ser memória próxima dessa CPU

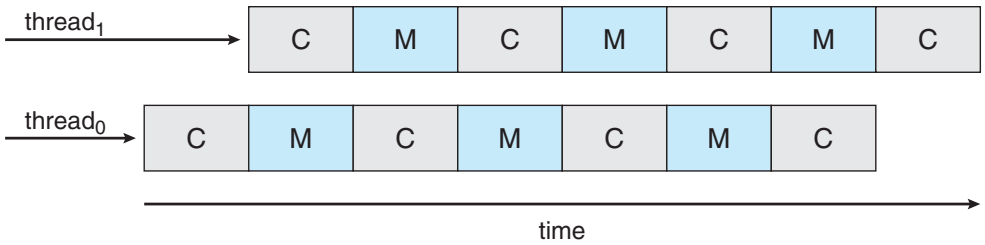


Escalonamento em Multiprocessadores (4/6)

- **impacto do** *hyper-threading*
  - sistemas SMP tradicionais: um núcleo (*core*) por CPU, um *thread* por CPU
  - sistemas multicore: vários núcleos por CPU, vários *threads* por CPU
    - com 1 só thread por núcleo há sub-utilização, devido a situações de *memory stall* (espera por dados/instruções da RAM, após *cache miss*)

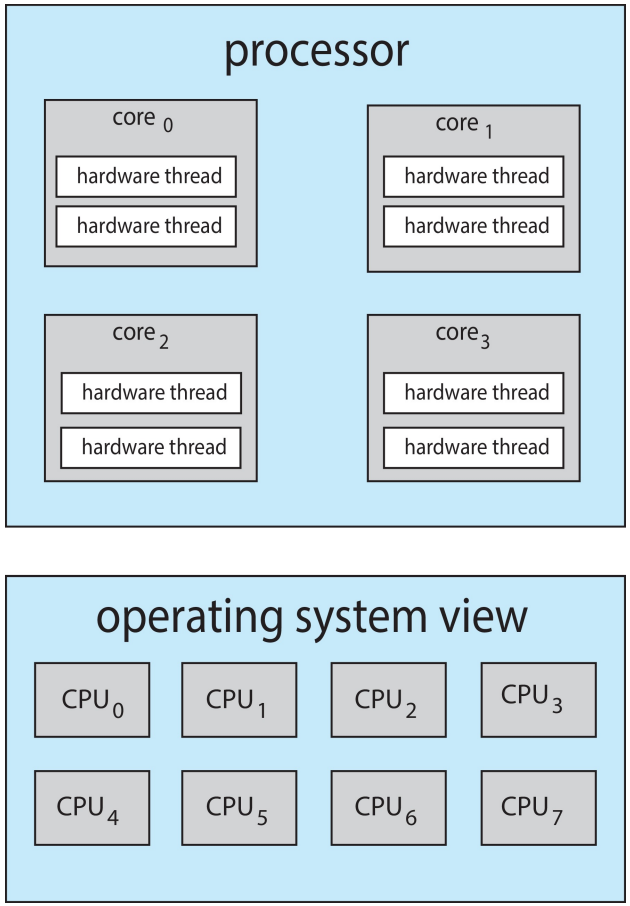


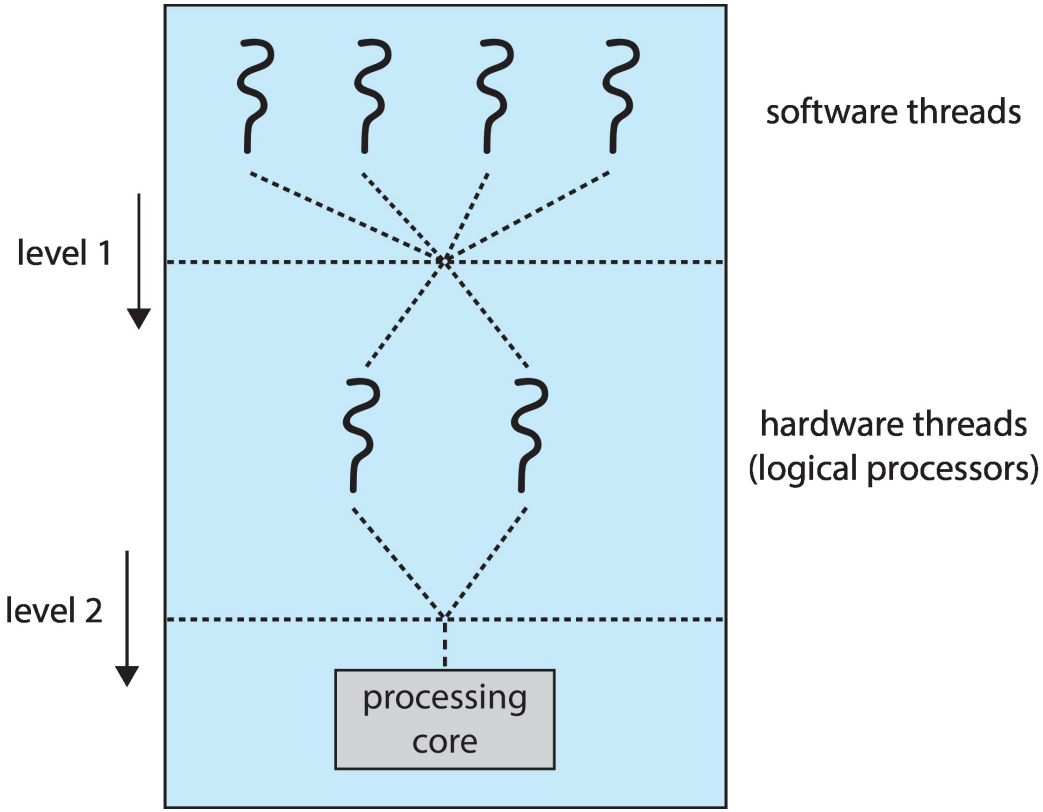
    - solução: vários *hardware threads* por cada núcleo (*hyperthreading*)



    - o SO vê cada *hardware thread* como um núcleo lógico, ao qual atribui um *kernel thread*, usando um algoritmo de escalonamento; mas cada núcleo decide qual a *hardware thread* que executa de cada vez !

Escalonamento em Multiprocessadores (5/6)





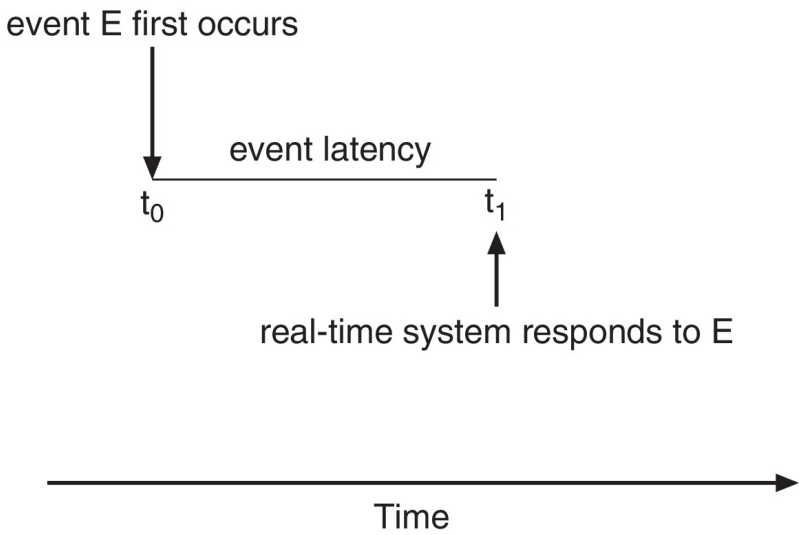
Escalonamento em Multiprocessadores (6/6)

- **impacto da virtualização**
  - um *hypervisor* partilha uma ou mais CPUs reais por vários *guests*
  - mesmo que o sistema tenha uma só CPU real, a camada de virtualização multiplexa-o, fornecendo a ilusão de um sistema SMP (o *hypervisor* apresenta a cada *guest* um ou mais CPUs virtuais)
  - o *hypervisor* escalona as CPUs reais e cada *guest* escalona as CPUs virtuais sem saber que não são reais (excepto em paravirtualização)
  - pode resultar em fracos tempos de resposta (decisões contraditórias); exemplo: num SO time-sharing virtual, um quantum virtual de 100ms pode consumir mais de 100 ms reais, degradando a interatividade
  - pode afectar a medição do tempo nos *guest* (o relógio virtual – baseado num contador – atrasa por sobrecarga da CPU real)
  - em suma: virtualização e escalonamento nos *guests* estão em conflito !

3.5 Escalonamento em Tempo Real

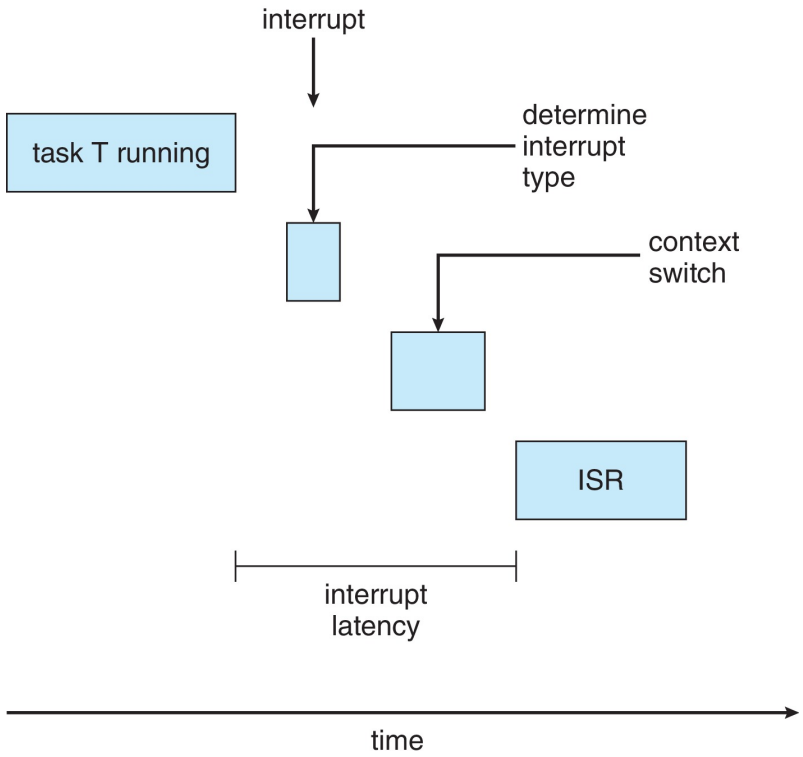
Escalonamento em Tempo Real (1/4)

- sistemas de tempo real colocam exigências especiais ao escalonamento
- sistemas de tipo *Soft Real-Time* escalonam os processos críticos sempre antes dos outros, mas não garantem quando é que serão escalonados
- sistema de tipo *Hard Real–Time* acrescentam a garantia de que a execução da tarefa ocorre num intervalo de tempo limitado, antes de uma *deadline*
- num sistema de tempo real, aguarda-se tipicamente por interrupções, que têm de ser atendidas rapidamente (ou seja, é necessário minimizar a *latência do evento*: tempo desde que o evento ocorre até que é tratado)



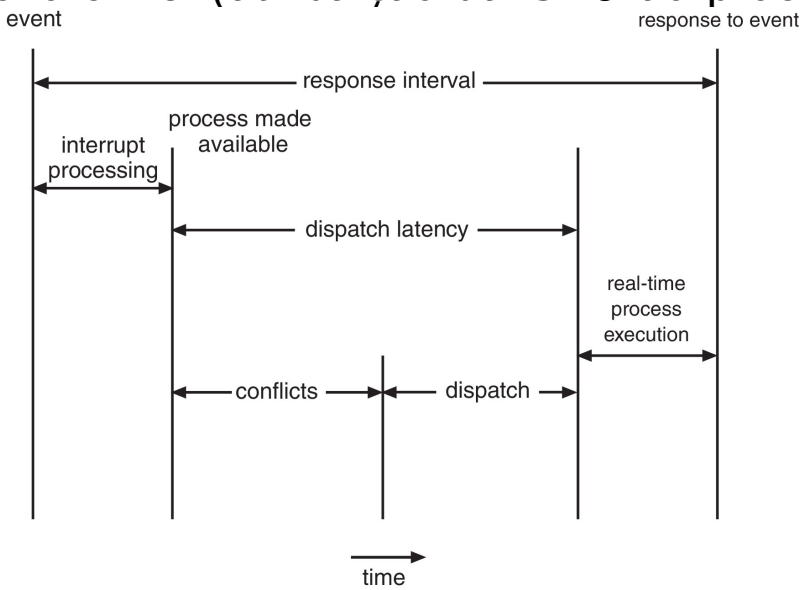
Escalonamento em Tempo Real (2/4)

- o desempenho dos sistema de tempo real é afetado por 2 tipos de latência
- **latência da interrupção**:
  - tempo desde a chegada da interrupção ao início da rotina de serviço
  - em sistemas de tipo *Hard-Real time* não basta minimizar este tempo; este tempo tem obrigatoriamente de ser inferior a um certo limite
  - se precisar, o kernel só desabilita interrupções de forma muito breve



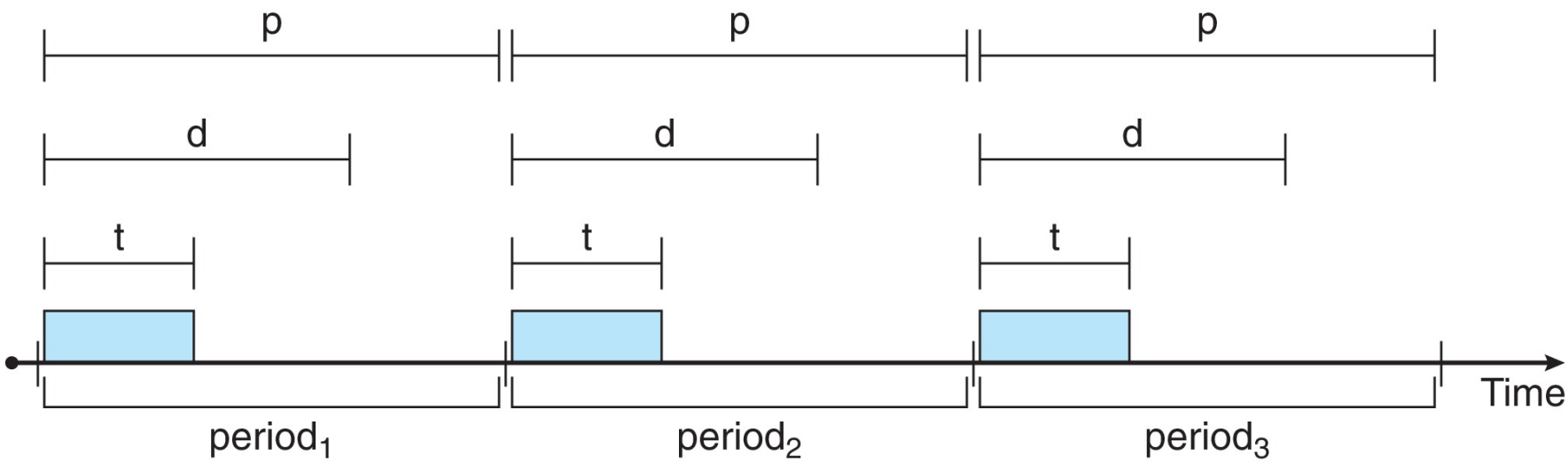
Escalonamento em Tempo Real (3/4)

- **latência de despacho**:
  - tempo de comutação de contexto (gasto em parar um processo e arrancar o próximo); parte integrante da *latência da interrupção*
  - uma forma de minimizar este tempo é tornar o kernel preemptível (senão, para dar lugar a outro processo é necessário aguardar que uma primitiva em curso termine ou que seja invocada uma operação E/S)
  - a latência de despacho inclui: i) fase de resolução de conflitos (preempção do processo atual + libertação de recursos detidos por processos < prioritários); ii) fase do despacho em si (atribuição da CPU ao processo de tempo real)



Escalonamento em Tempo Real (4/4)

- **escalonamento por prioridades**:
  - para suportar escalonamento em tempo real de tipo *soft* basta que o escalonador suporte escalonamento preemptivo baseado em prioridades
  - escalonamento em tempo real de tipo *hard* requer suporte a *deadlines*
    - exemplo: processos *periódicos*, que necessitam CPU com uma certa regularidade ou período constante p, têm deadline d e surto de CPU t
    - os processos têm de anunciar d ao escalonador; este, com base num algoritmo de *controlo de admissão*, decide se é ou não viável aceitar o processo para uma execução que garanta o cumprimento da *deadline*



3.6 Exercícios

Exercício 3.1 (Algoritmos de Escalonamento) (1/7)

Considere o seguinte conjunto de processos e respetivos atributos de escalonamento:

Process	Arrival Instant	Priority	Total CPU Time (ms)
P_1	0	2	5
P_2	1	1	4
P_3	2	3	2

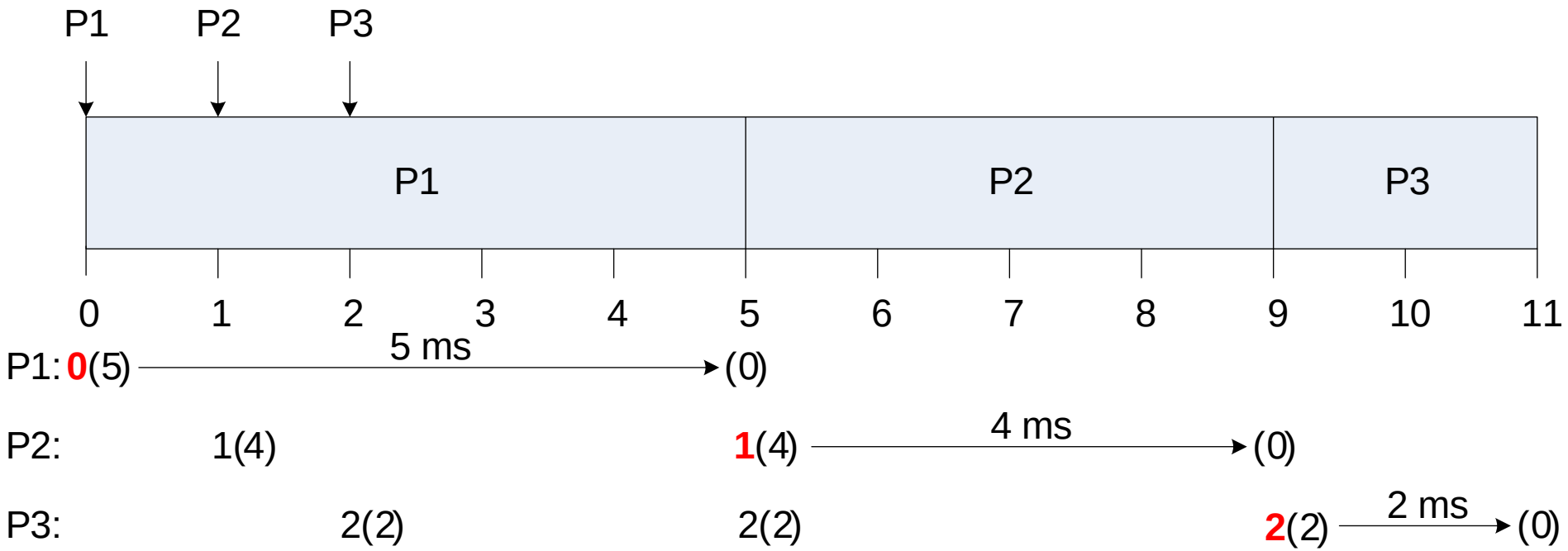
a) Desenhe gráficos de Gantt ilustrando a execução destes processos com as políticas de escalonamento FCFS, SJF não-preemptivo, SJF preemptivo, Prioridades genéricas não-preemptivo, Prioridades genéricas preemptivo, e RR (q=2).
Para as políticas SJF* assuma que a previsão inicial corresponde ao Tempo de CPU Total da tabela. Para as políticas baseadas em Prioridades genéricas assuma que índices de Prioridade menores correspondem a prioridades maiores.

b) Determine o tempo de espera médio (average waiting time), e o tempo de retorno médio (average turnaround time), para cada um dos cenários anteriores.

Exercício 3.1 (Algoritmos de Escalonamento) (2/7)

- FCFS

Process	**Arrival Instant**	Total CPU Time (ms)
P_1	**0**	5
P_2	**1**	4
P_3	**2**	2

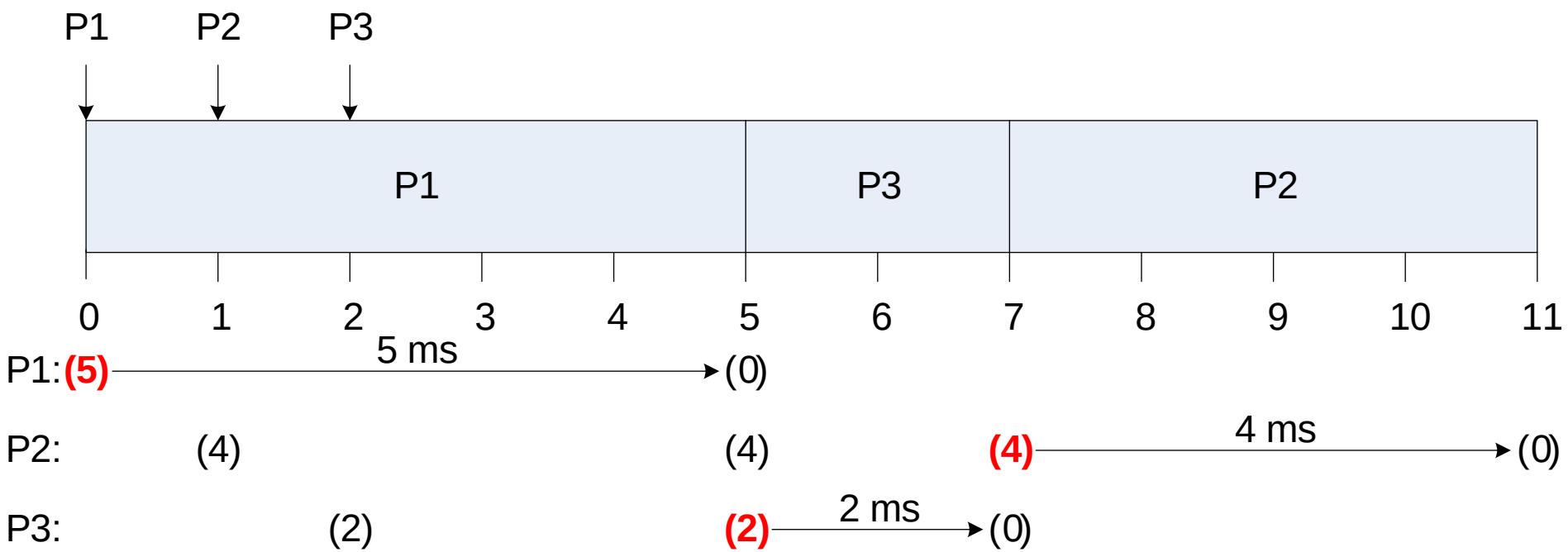


- average waiting time: (0 + (5-1) + (9-2)) / 3 = 11/3 = 3.66(6) ms
- average turnaround time: [(5-0) + (9-1) + (11-2)] / 3 = 22/3 = 7.33(3) ms

Exercício 3.1 (Algoritmos de Escalonamento) (3/7)

- SJF non-preemptive

		CPU Bursts (ms)		
Process	Arrival Instant	**Predicted**	Effective	Total
P_1	0	**5**	5	5
P_2	1	**4**	4	4
P_3	2	**2**	2	2

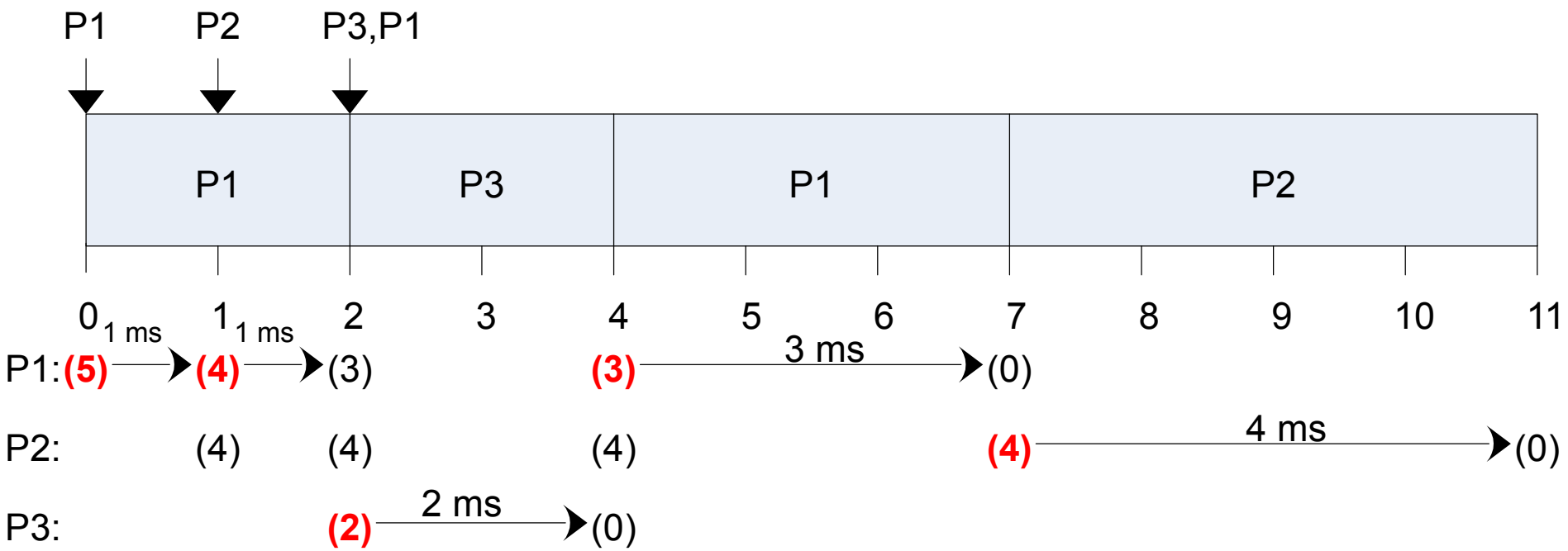


- average waiting time: (0 + (7-1) + (5-2)) / 3 = 9/3 = 3 ms
- average turnaround time: [(5-0) + (11-1) + (7-2)] / 3 = 20/3 = 6.66(6) ms

Exercício 3.1 (Algoritmos de Escalonamento) (4/7)

- SJF preemptive

Process	Arrival Instants	CPU Bursts (ms) Predicted	Effective	Total
P_1	0, 2	**5,4=5**-1(*),**3=4**-1(**)	1(*),1(**),3	1+1+3=5
P_2	1	**4**	4	4
P_3	2	**2**	2	2

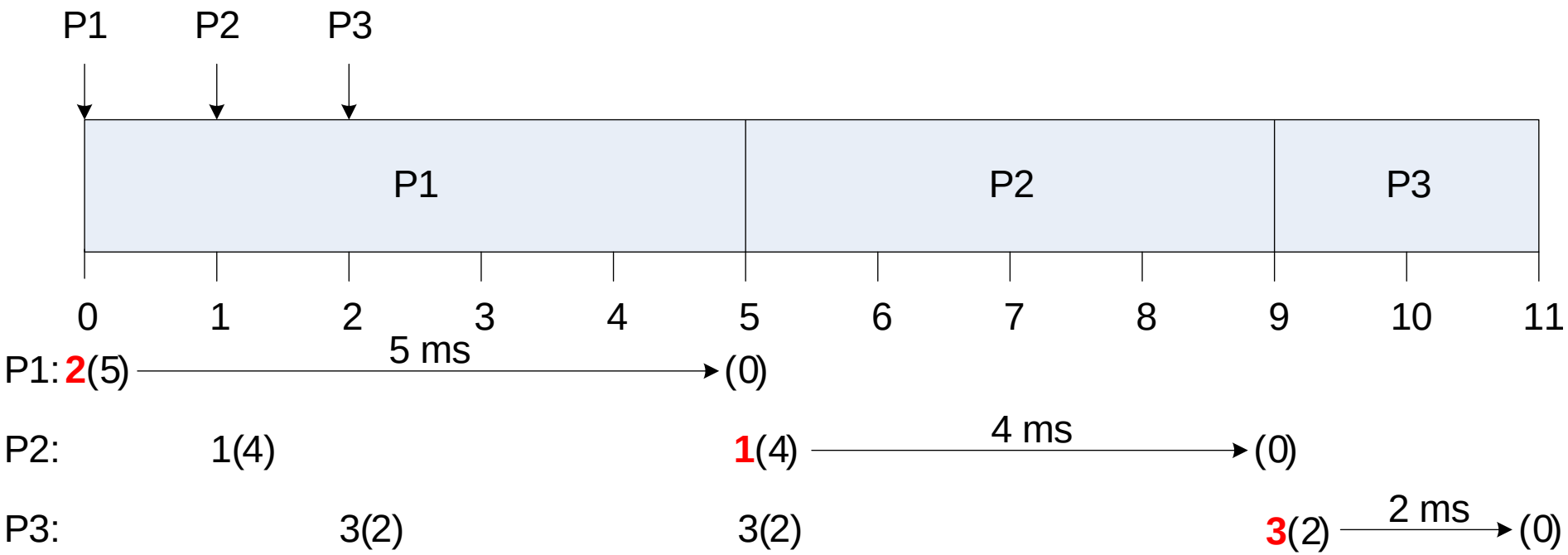


- average waiting time: ((4-2) + (7-1) + 0) / 3 = 8/3 = 2.66(6) ms
- average turnaround time: [(7-0) + (11-1) + (4-2)] / 3 = 19/3 = 6.33(3) ms

Exercício 3.1 (Algoritmos de Escalonamento) (5/7)

- Priorities (generic) non-preemptive

Process	Arrival Instant	**Priority**	Total CPU Time (ms)
P_1	0	**2**	5
P_2	1	**1**	4
P_3	2	**3**	2

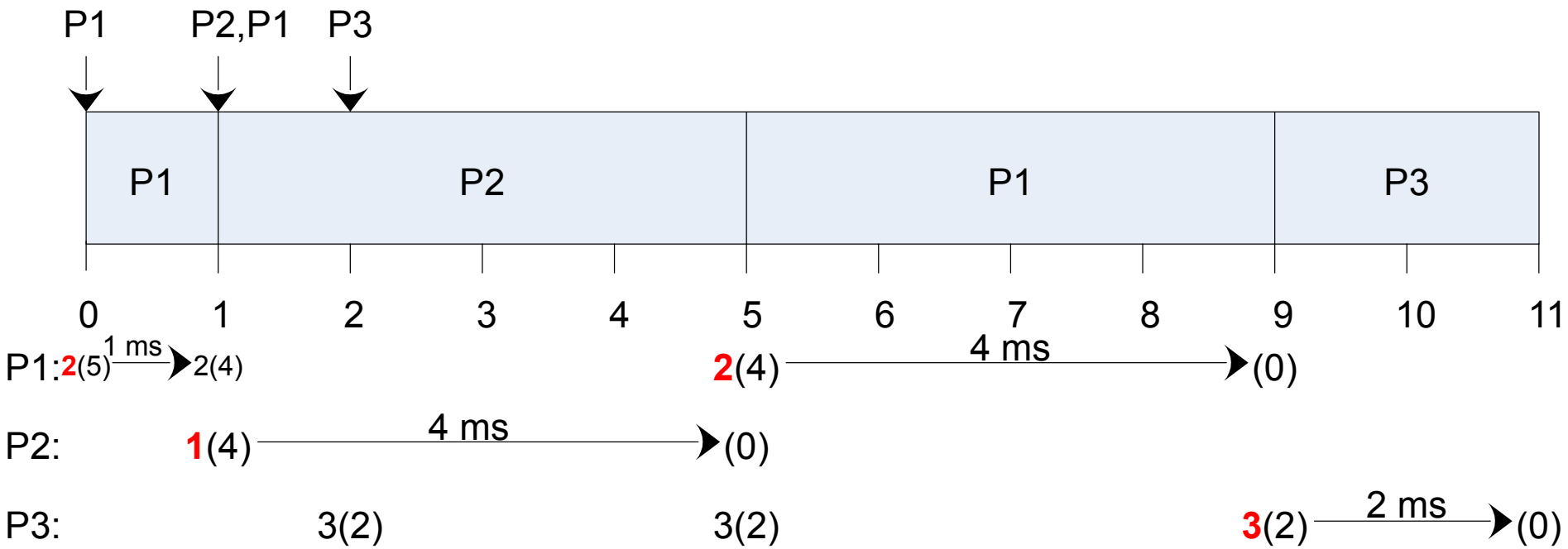


- average waiting time: (0 + (5-1) + (9-2)) / 3 = 11/3 = 3.66(6) ms
- average turnaround time: [(5-0) + (9-1) + (11-2)] / 3 = 22/3 = 7.33(3) ms

Exercício 3.1 (Algoritmos de Escalonamento) (6/7)

- Priorities (generic) preemptive

Process	Arrival Instants	Priority	CPU Bursts (ms)	Total CPU Time (ms)
P_1	0,1	2	1,4	5
P_2	1	1	4	4
P_3	2	3	2	2

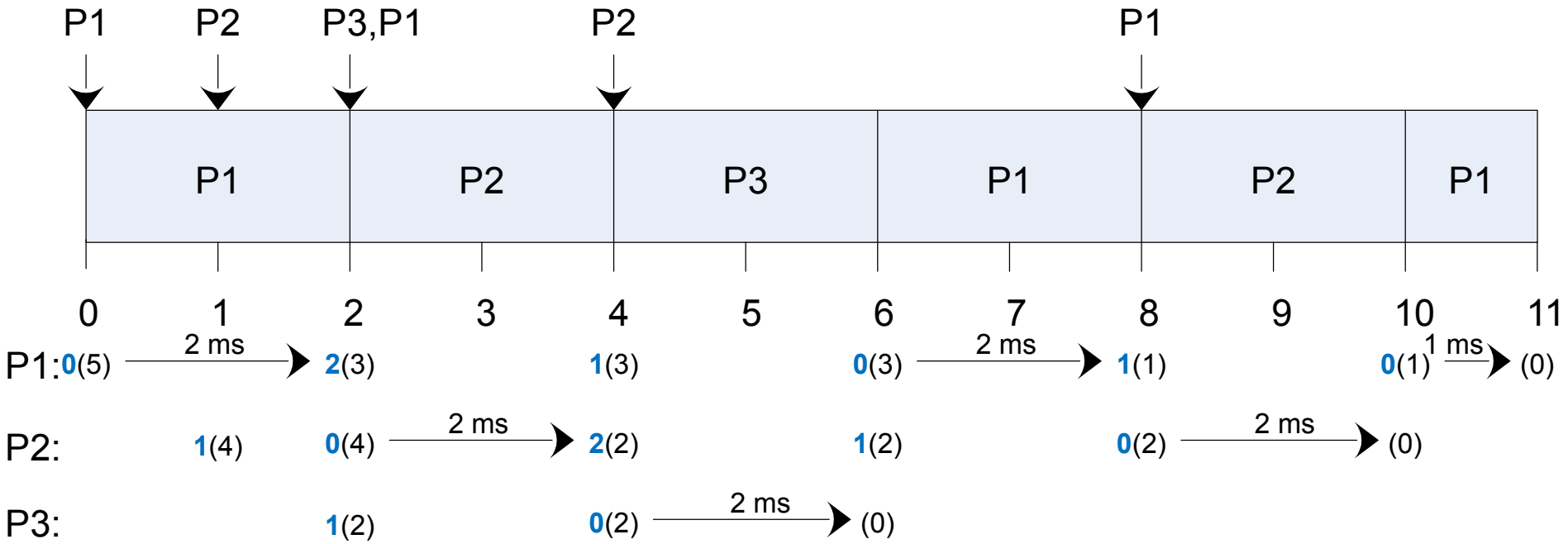


- average waiting time: [(0 + (5-1)) + 0 + (9-2)] / 3 = 11/3 = 3.66(6) ms
- average turnaround time: [(9-0) + (5-1) + (11-2)] / 3 = 22/3 = 7.33(3) ms

Exercício 3.1 (Algoritmos de Escalonamento) (7/7)

- RR (**q=2**)

Process	**Arrival Instants**	CPU Bursts (ms)	Total CPU Time (ms)
P_1	0,2,8	2,2,1	$\sum$=5
P_2	1,4	2,2	$\sum$=4
P_3	2	2	$\sum$=2



- números a azul: posição na fila *ready* (0 para o processo escolhido p/ executar)
- notar que no instante 2, P3 (recém-chegado) entra na fila *ready* à frente de P1
- t. espera médio: [(0+6-2+10-8) + (2-1+8-4) + (4-2)] / 3 = 13/3 = 4.33(3) ms
- t. retorno médio: [(11-0) + (10-1) + (6-2)] / 3 = 24/3 = 8 ms

Exercício 3.2 (Algoritmos de Escalonamento) (1/2)

Considere o seguinte conjunto de processos e respetivos atributos de escalonamento:

Process	Arrival Instant	Priority	Total CPU Time (ms)
P_1	0	4	6
P_2	1	3	4
P_3	2	2	2
P_4	3	1	1

1.1 Desenhe gráficos de Gantt ilustrando a execução destes processos com as políticas de escalonamento FCFS, SJF não-preemptivo, SJF preemptivo, Prioridades genéricas não-preemptivo, Prioridades genéricas preemptivo, e RR (q=2).
Para as políticas SJF* assuma que a previsão inicial corresponde ao Tempo de CPU Total da tabela. Para as políticas baseadas em Prioridades genéricas assuma que índices de Prioridade menores correspondem a prioridades maiores.

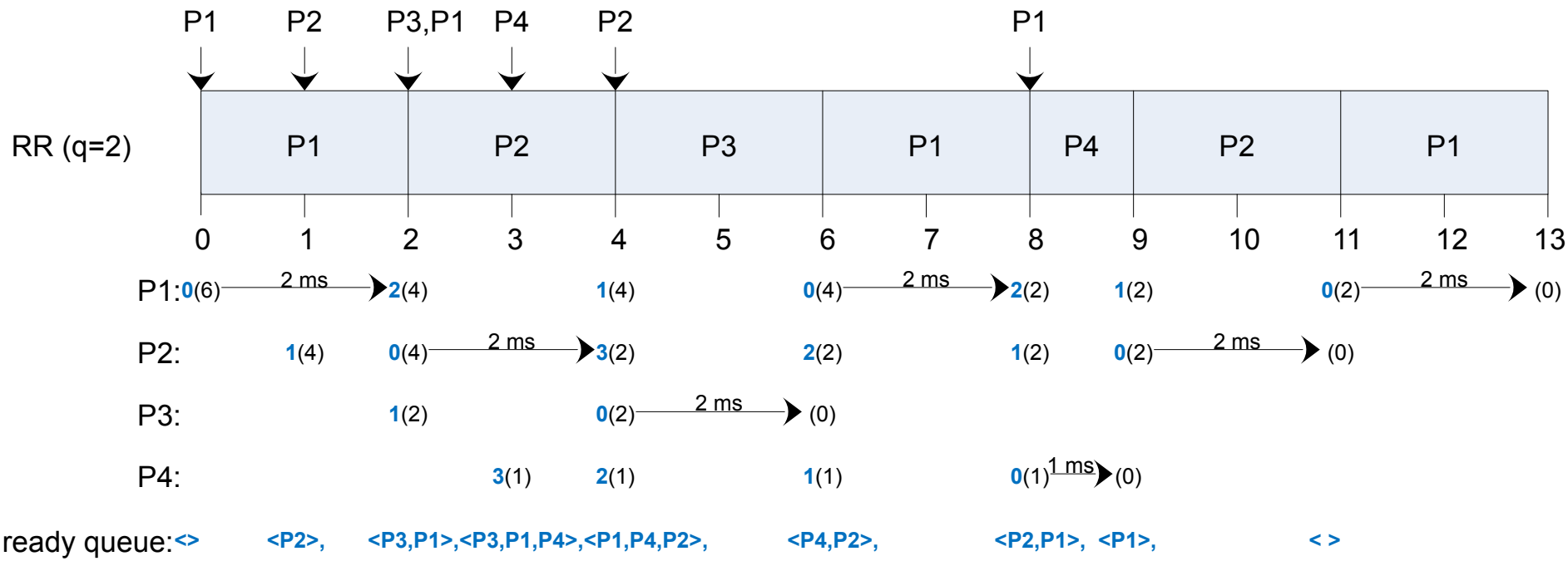
1.2 Determine o tempo de espera médio, e o tempo de retorno médio, para cada um dos cenários anteriores.

Nota: no slide seguinte é fornecida a solução para o algoritmo RR, por tratar-se de um caso especial; para os restantes algoritmos, usar o Exercício 3.1 como referência

Exercício 3.2 (Algoritmos de Escalonamento) (2/2)

- no Exercício 3.1 não há processos novos a entrar na fila *ready* após o regresso de outros processos a essa fila; isso simplifica bastante a análise, mas é irrealista
- **RR(q=2)**: P1 sai da CPU e regressa à fila ready antes do P4 lá entrar pela 1a vez

Process	Arrival Instants	CPU Bursts (ms)	Total CPU Time (ms)
P_1	0,6,11	2,2,2	$\sum$=6
P_2	1,4	2,2	$\sum$=4
P_3	2	2	$\sum$=2
P_4	3	1	$\sum$=1



Exercício 3.3 (Algoritmos de Escalonamento) (1/1)

Considere o seguinte conjunto de processos e respetivos atributos de escalonamento:

Process	Arrival Instant	Priority	CPU Time for 1st Output (ms)	Total CPU Time (ms)
P_1	0	4	1	5
P_2	1	3	2	4
P_3	2	1	2	3
P_4	3	2	1	2

1.1 Desenhe gráficos de Gantt ilustrando a execução destes processos com as políticas de escalonamento FCFS, SJF não-preemptivo, SJF preemptivo, Prioridades genéricas não-preemptivo, Prioridades genéricas preemptivo, e RR (q=2).
Para as políticas SJF* assuma que a previsão inicial corresponde ao Tempo de CPU Total da tabela. Para as políticas baseadas em Prioridades genéricas assuma que índices de Prioridade menores correspondem a prioridades maiores.

1.2 Determine o tempo de espera médio, o tempo de retorno médio, e o tempo de resposta médio, para cada um dos cenários anteriores.

Nota: recorde que um pedido de output faz o processo bloquear numa fila E/S; assuma que o processo regressa rapidamente à fila *ready* já depois de ter sido escolhido o seu sucessor na CPU (mas ainda a tempo de participar na decisão de escalonamento seguinte); no contexto dos algoritmos FCFS e RR, assuma que o processo vai para o fim da fila *ready*

REFERÊNCIAS

- "Operating System Concepts, 10th Ed.", Silberschatz & Galvin, Addison-Wesley, 2018: Capítulo 5
- "Fundamentos de Sistemas Operacionais, 9a Ed.", Silberschatz, Galvin & Gagne, LTC, 2015: Capítulo 6
- "The Fancy Algorithms That Make Your Computer Feel Smoother" (`https://www.youtube.com/watch?v=O2tV9q6784k`)