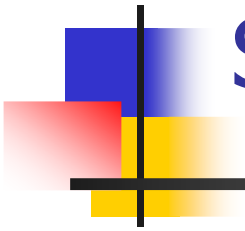
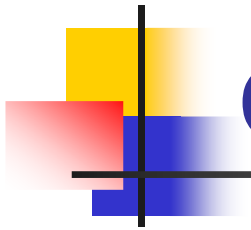


Sistemas Operacionais

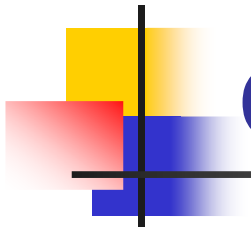
Unidade Dois

Prof. Flávio Márcio de Moraes e Silva



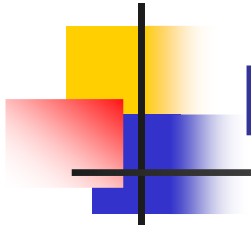
Conceitos de Processo

- Processo – um programa em execução
  - Organizados em vários processos seqüenciais
- Um processo inclui:
  - Contador de programa
  - Registradores (pilha)
  - Variáveis (seção de dados)



Conceitos de Processo

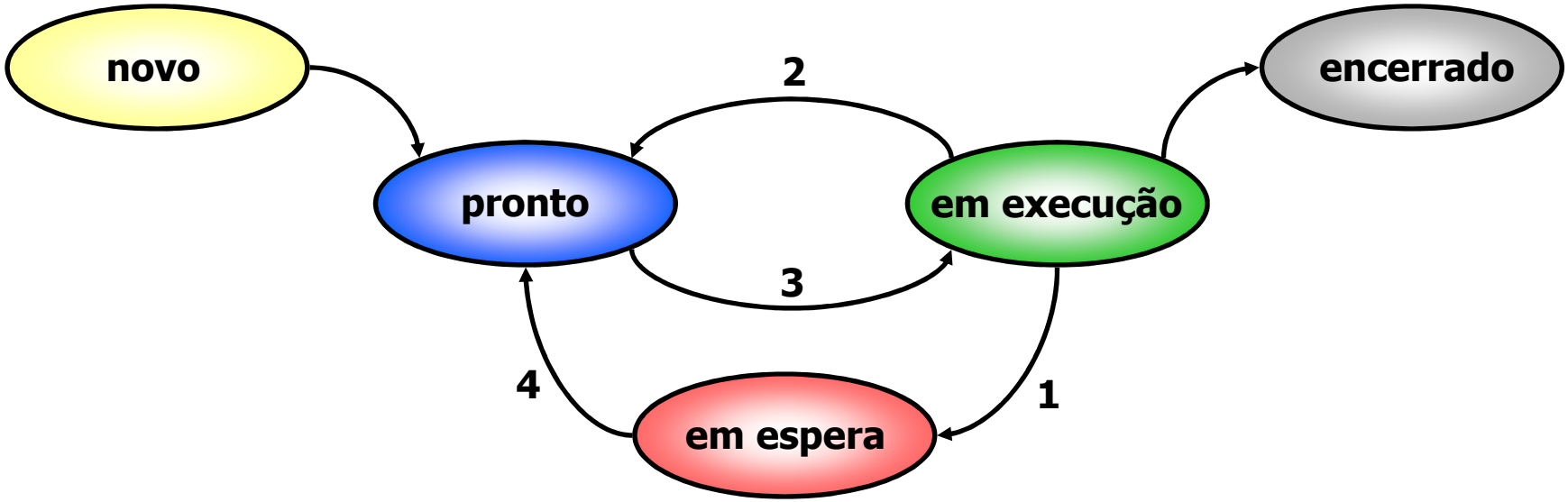
- Ciclo de vida de um processo
  - Tempo entre a criação do mesmo e sua destruição no SO
  - SO's lidam de formas diferentes com a criação de processos (numero fixo x variável de processos, processos soltos x vinculados a seções de usuários, etc)
- Relacionamento entre processos
  - Independência total de processos
  - Grupos de processos
  - Hierarquia de processos

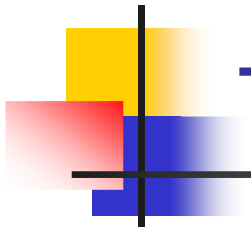


Estados de Processos

- A medida em que o programa executa, seu estado muda:
  - Novo: O processo está sendo criado
  - Pronto: O processo está esperando para ser atribuído a um processador
  - Em execução: Instruções estão sendo executadas
  - Em espera: O processo espera por um evento
  - Encerrado: O processo terminou sua execução

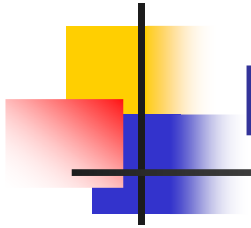
Estados de Processos





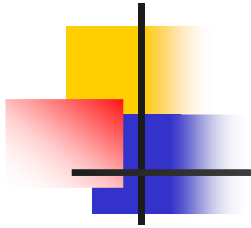
Transição de Estado

- **Transição 1**
  - Ocorre quando um processo descobre que não pode prosseguir
- **Transições 2 e 3**
  - São causadas pelo escalonador de processo
  - O processo não precisa tomar conhecimento que foi interrompido
  - Dispatcher = responsável pela reposição de contexto (voltar valores dos registradores na transição 3)
- **Transição 4**
  - Ocorre quando acontece um evento externo pelo qual um processo estava aguardando



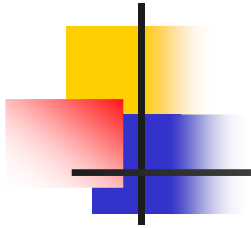
Bloco de Controle de Processos

- Informação associada com cada processo
  - Contador de Programa
  - Pilha de execução
  - Registradores da CPU
  - Estado do processo (apto, executando, bloqueado,etc)
  - Informações de escalonamento de CPU (tempo de processador, prioridade)
  - Informações de gerência de memória
  - Informações de status de I/O
  - Informações de arquivos abertos



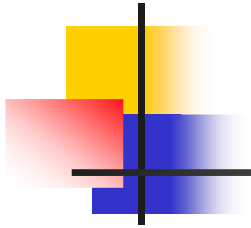
Threads

- Processos são baseados em dois conceitos independentes: agrupamento de recursos e execução
  - Threads são úteis quando há a necessidade de separá-los
- Um *thread* (ou *processo leve*) é uma unidade básica de utilização da CPU. Consiste em:
  - Contador de programa: para manter o controle da próxima instrução a ser executada
  - Conjunto de registradores: contêm suas variáveis atuais de trabalho
  - Pilha: traz o histórico de execução, e uma estrutura para cada processo chamado e não retornado



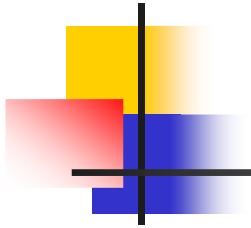
Threads

- Um *thread* compartilha com outros *threads* do mesmo processo:
  - Seção de código
  - Seção de dados
  - Recursos do SO
- Um processo tradicional ou *pesado* é igual a uma tarefa com apenas um *thread*



Threads

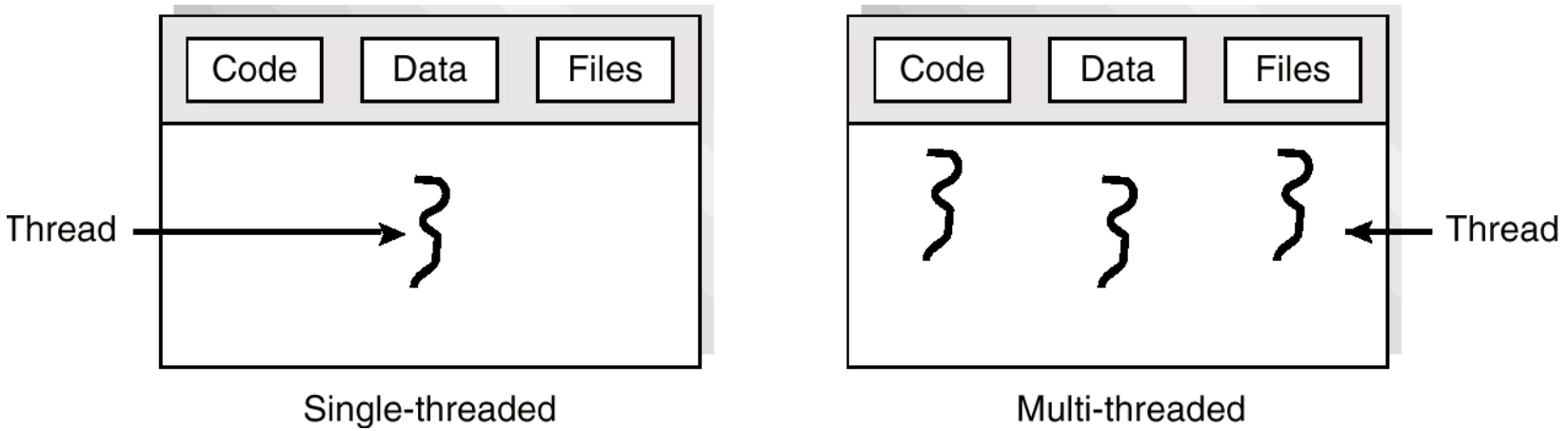
- Em uma tarefa com múltiplos *threads*, enquanto um *thread* servidor está bloqueado e esperando, um segundo *thread* na mesma tarefa pode executar
  - Cooperação entre múltiplos *threads* na mesma tarefa confere maior produção (*throughput*) e performance
  - Aplicações que requerem o compartilhamento de um buffer comum se beneficiam da utilização de *threads*
- Exemplos:
  - Um navegador *Web* pode ter um *thread* exibindo uma imagem enquanto outro recupera dados na rede
  - Um editor de textos pode ter um *thread* lendo caracteres do teclado enquanto outro faz a verificação ortográfica

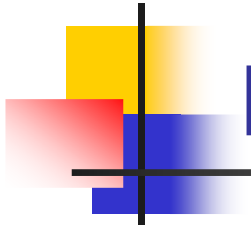


Threads

- **Vantagens**
  - Capacidade de Resposta: tarefas podem ser executadas enquanto outras esperam por recurso
  - Compartilhamento de Recursos: facilita acesso
  - Economia: alocar memória e recursos para a criação de um processo novo é caro
  - Utilização de arquiteturas paralelas

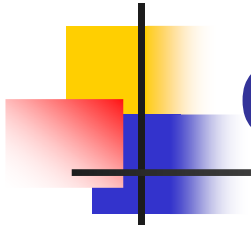
Processos com um ou Vários Threads





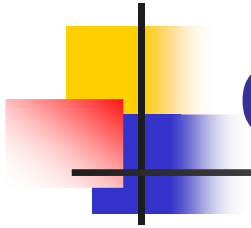
Escalonamento: Conceitos Básicos

- A utilização máxima de CPU é obtida através de multiprogramação. Com um só processador, nunca haverá mais de um processo em execução
- Quando dois ou mais processos estão na situação de pronto, o SO deverá fazer a escolha de qual processo será executado
  - Escalonador é a parte do SO que escolhe processo
  - Utiliza algoritmos de escalonamento
  - A troca de processo é "cara" para a CPU



Comportamento do Processo

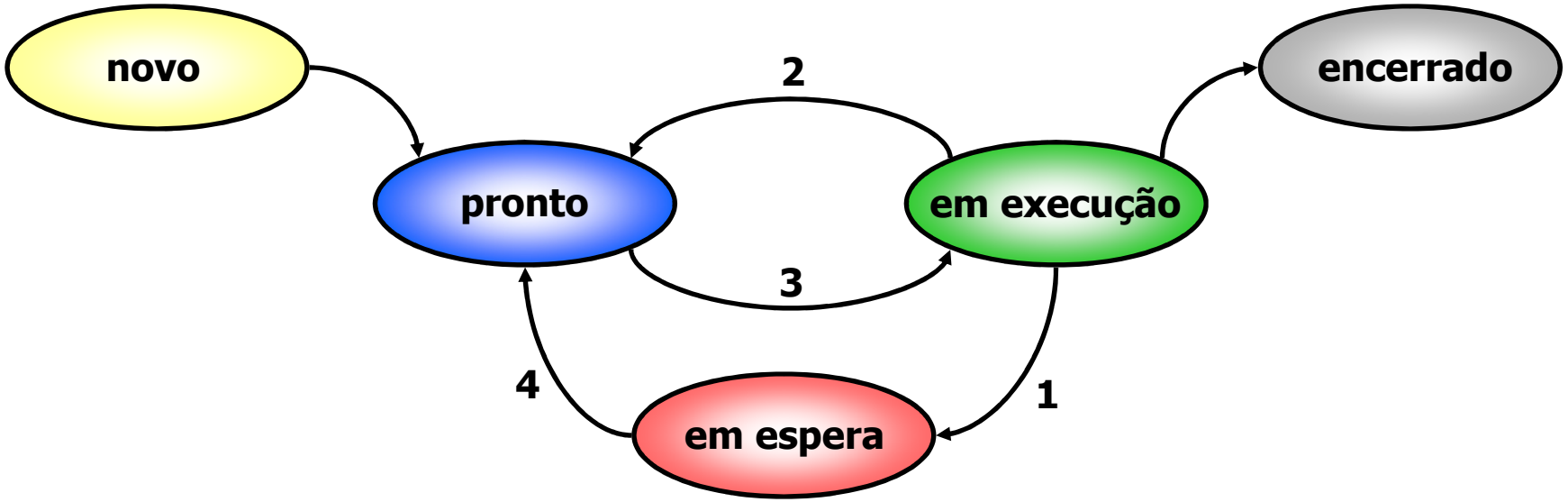
- Todo processo alterna surtos de uso da CPU com requisições de E/S
- Tipos de surtos
  - Orientado a CPU
    - Passam a maior parte do tempo utilizando a CPU
  - Orientado a E/S
    - Passam a maior parte do tempo realizando E/S
- A medida que a velocidade dos processadores aumentam os processos tendem a ficar orientados a E/S

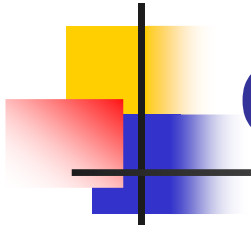


Critérios de Escalonamento

- É fundamental o SO saber o momento certo de escalonar um processo
- Decisões de escalonamento de CPU ocorrem quando o processo:
  1. Muda do estado de execução para espera
  2. Muda do estado de execução para pronto
  3. Muda do estado de espera para pronto
  4. Termina
- Escalonamento *não-preemptivo* um processo só perde a CPU por vontade própria (término ou chamada de sistema 1 e 4).
- Escalonamento *preemptivo* o processo em execução pode perder a CPU para outro e maior prioridade (2 e 3).

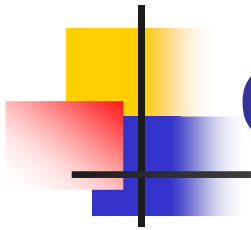
Estados de Processos





Critérios de Escalonamento

- Características para comparação de algoritmos:
  - Utilização de CPU: manter a CPU o mais ocupada possível
  - *Throughput*: quantidade de processos que completam sua execução por unidade de tempo
  - Tempo de retorno: quantidade de tempo para executar um processo
  - Tempo de espera: quantidade de tempo que um processo gasta na fila de processos prontos
  - Tempo de resposta: quantidade de tempo gasto entre a requisição e a produção da primeira resposta



Critérios de Escalonamento

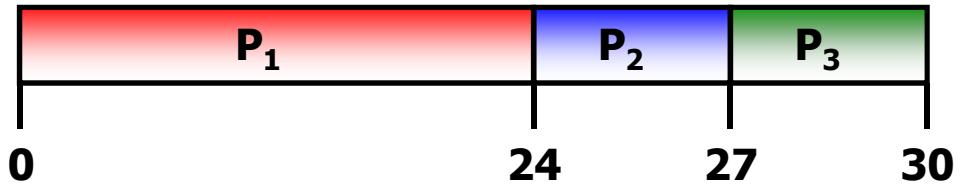
- Máxima utilização de CPU
- Máximo *throughput*
- Mínimo tempo de retorno
- Mínimo tempo de espera
- Mínimo tempo de resposta

Algoritmos de Escalonamento

- **Primeiro a chegar é servido (FIFO)**
  - Exemplo:

Processo	Duração de Surto
P_1	24
P_2	3
P_3	3

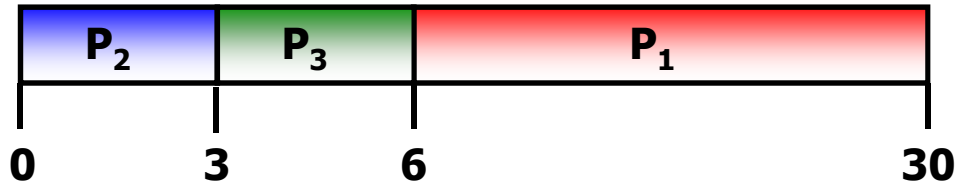
  - Suponha que os processos chegam na ordem: P_1 , P_2 , P_3
    O diagrama para o escalonamento é:



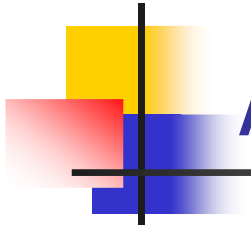
  - O tempo de espera para P_1 = 0; P_2 = 24; P_3 = 27
  - O tempo de espera médio: (0 + 24 + 27)/3 = 17

Algoritmos de Escalonamento

- **Primeiro a chegar é servido (FIFO)**
  - Suponha que os processos cheguem na ordem: P_2 , P_3 , P_1
  - O diagrama para o escalonamento é:



  - O tempo de espera para $P_1 = 6; P_2 = 0, P_3 = 3$
  - O tempo de espera médio : (6 + 0 + 3)/3 = 3
  - Muito melhor que o anterior
  - *Efeito Comboio:* pequenos processos atrás de longos processos



Algoritmos de Escalonamento

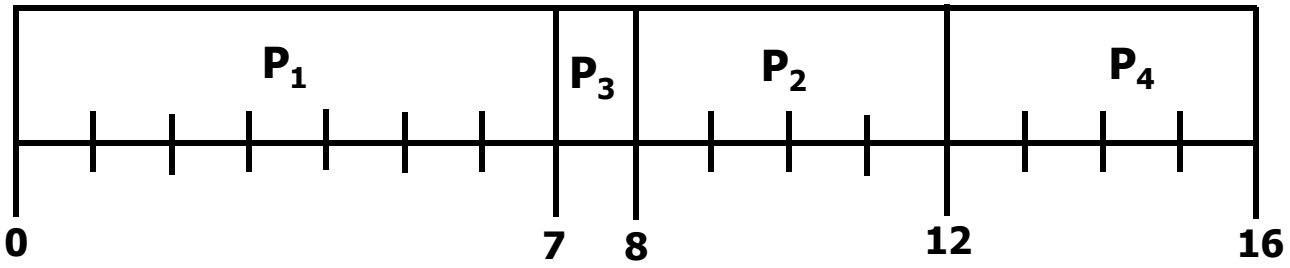
- **Escalonamento job mais curto primeiro (SJF)**
  - Associa a cada processo o tamanho do seu próximo surto de CPU. Usa estes valores para escalonar o processo com o menor tempo
  - Dois esquemas:
    - Não-preemptivo – uma vez que a CPU é dada ao processo, não pode ser retirada até completar seu surto
    - Preemptiva – se um novo processo chegar com um tempo de surto de CPU menor que o tempo restante do processo sendo executado, é feita a troca. Este esquema é conhecido como Menor-tempo-restante-primeiro (SRTF)
  - SJF preemptivo é ótimo – sempre dá o mínimo tempo médio de espera para o conjunto de processos

Algoritmos de Escalonamento

- **Escalonamento job mais curto primeiro (SJF)**
- Exemplo

Processo	Chegada	Tempo Surto
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (não-preemptivo)



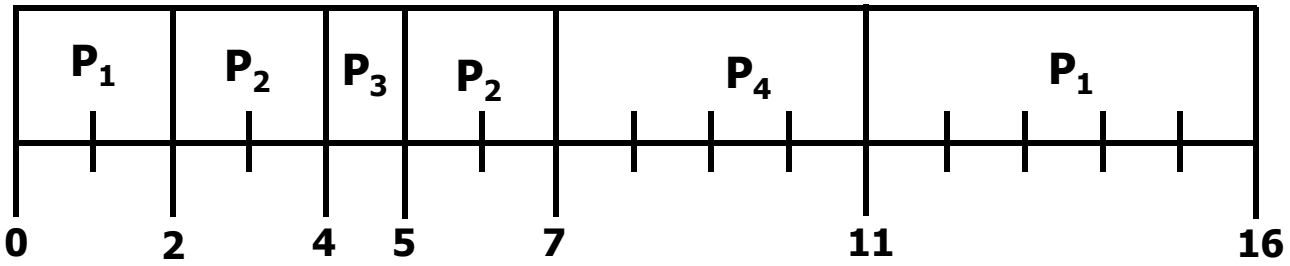
- Tempo médio de espera = (0 + 3 + 6 + 7)/4 = 4

Algoritmos de Escalonamento

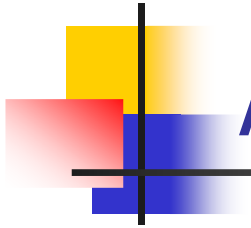
- **Escalonamento job mais curto primeiro (SJF)**
- Exemplo

Processo	Chegada	Tempo Surto
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptivo)

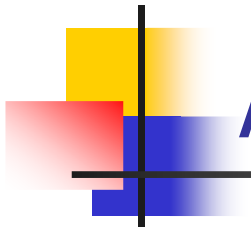


- Tempo médio de espera = (0 + 0 + 0 + 2)/4 = 0,5



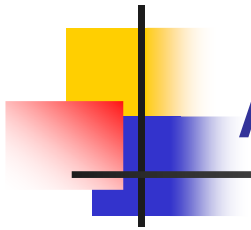
Algoritmos de Escalonamento

- **Escalonamento por prioridade**
- Um número de prioridade (inteiro) é associado com cada processo
- A CPU é alocada para o processo com maior prioridade (menor valor ≡ maior prioridade)
- SJF é um escalonamento por prioridade, onde a prioridade é dada pelo tempo de surto
- Problema: *Starvation* – processos com baixa prioridade podem nunca serem executados
- Solução: Envelhecimento (*Aging*) – a prioridade aumenta com o tempo



Algoritmos de Escalonamento

- **Round Robin (RR)**
  - Cada processo recebe uma pequena unidade de tempo de CPU (*quantum*), geralmente 10-100 ms. Após este tempo, o processo é retirado e inserido no fim da fila de prontos
  - Se existirem *n* processos na fila de prontos e o quantum for *q*, cada processo terá 1/*n* de tempo de CPU em parcelas de no máximo *q* unidades de tempo por vez. Nenhum processo esperará mais que (*n*-1)*q* unidades de tempo
  - Não precisa esperar até o final do quantum
    - Quando um processo termina outro processo utiliza a CPU imediatamente



Algoritmos de Escalonamento

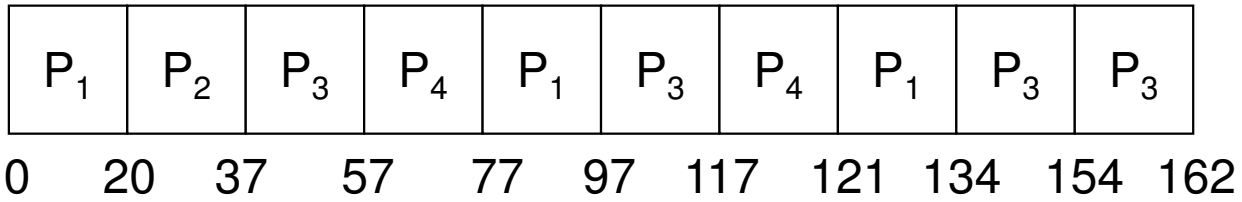
- **Round Robin (RR)**
  - Desempenho
    - q grande $\Rightarrow$ FIFO
    - q pequeno $\Rightarrow$ q deve ser grande em relação ao tempo de troca de contexto, ou o overhead será muito alto
      - Se q possui 4ms e o tempo de chaveamento é 1ms, o resultado será péssimo

Algoritmos de Escalonamento

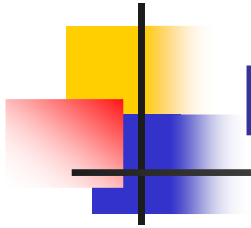
- **Round Robin (RR) -> q = 20**
- Exemplo

Processo	Tempo de Surto
P_1	53
P_2	17
P_3	68
P_4	24

- O diagrama é:



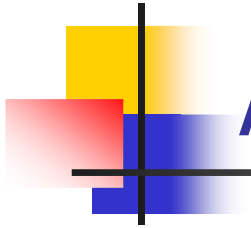
- Tipicamente, maior média de retorno que SJF, mas melhor resposta



Exercício

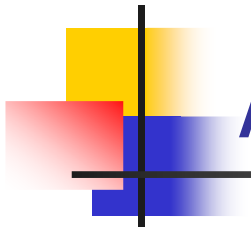
- Com base nos dados abaixo mostre os diagramas de escalonamento para os algoritmos: FIFO, SJF Preemptivo, SJF Não-Preemptivo, por Prioridade Preemptivo, por Prioridade Não-Preemptivo e Round-Robim (quantum = 2).

Prioridade	Processo	Chegada	Tempo Surto
3	*P4*	0.0	8
2	*P3*	2.0	2
1	*P2*	4.0	4
2	*P1*	5.0	6



Algoritmos de Escalonamento

- **Filas Multiplas**
  - A fila de prontos é particionada em filas separadas:
    - primeiro plano (interativo)
    - segundo plano (batch)
  - Cada fila tem seu próprio algoritmo de escalonamento, por exemplo:
    - primeiro plano – RR
    - segundo plano – FIFO
  - Deve haver escalonamento entre as filas
    - Escalonamento de prioridade fixa: primeiro plano pode ter prioridade sobre o segundo plano. Possibilidade de *starvation*
    - Fatia de tempo: cada fila recebe uma certa quantidade de tempo de CPU que pode ser escalonada entre seus processos. Por exemplo, 80% para primeiro plano em RR e 20% para o segundo plano em FIFO

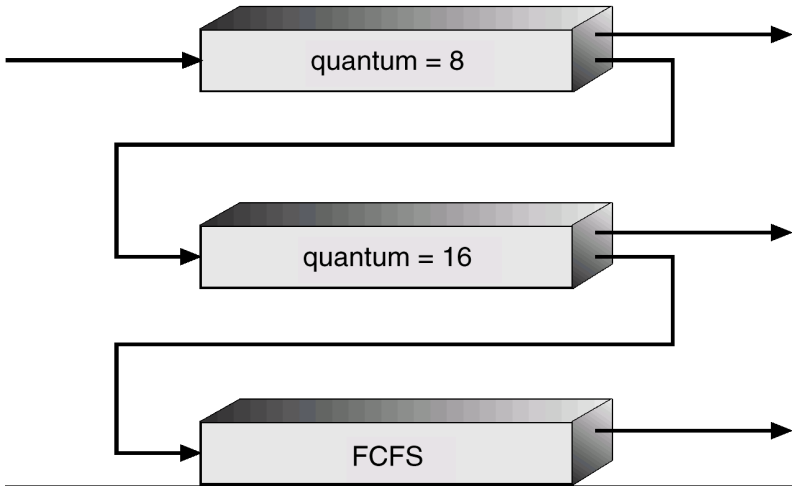


Algoritmos de Escalonamento

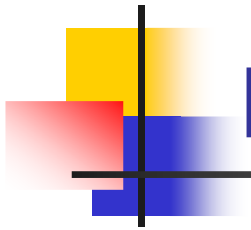
- **Filas Multiplas com Realimentação**
  - Um processo pode passar de uma fila para outra. O envelhecimento pode ser implementado desta maneira
  - O escalonador de Filas Múltiplas com realimentação é definido pelos seguintes parâmetros:
    - número de filas
    - algoritmos de escalonamento para cada fila
    - algoritmo de escalonamento entre filas
    - método usado para determinar a promoção de um processo a uma fila de maior prioridade
    - método usado para determinar quando rebaixar um processo
    - método usado para determinar em qual fila o processo deve entrar quando precisar de serviço

Algoritmos de Escalonamento

- **SJF - Filas Multiplas com Realimentação**
  - Três filas:
    - Q_0 – quantum 8 ms
    - Q_1 – quantum 16 ms
    - Q_2 – FCFS = FIFO



  - Escalonador
    - Um novo job entra na fila Q_0 servido por FIFO. Quando recebe CPU, o job tem 8 ms. Se não terminar em 8 ms, o job é removido para Q_1.
    - Em Q_1 o job é servido por FIFO e recebe 16 ms adicionais. Se ainda não terminar, é transferido para Q_2.
    - Algoritmo entre filas por PRIORIDADE: $Q_0 > Q_1 > Q_2$



Exercício

- Proponha uma solução de escalonamento de processos, através de filas múltiplas com ou sem re-alimentação, para o seguinte problema de uma empresa, existem:
  - Processos lentos, acima de 3hs/processo, sem restrição de data e hora para executarem.
  - Processos de alta prioridade para a empresa.
  - Processos dos usuários de tempo variável, sendo interessante executar os orientados a E/S primeiro.