# Aula 05 – Escalonamento de Processos

Norton Trevisan Roman
Clodoaldo Aparecido de Moraes Lima

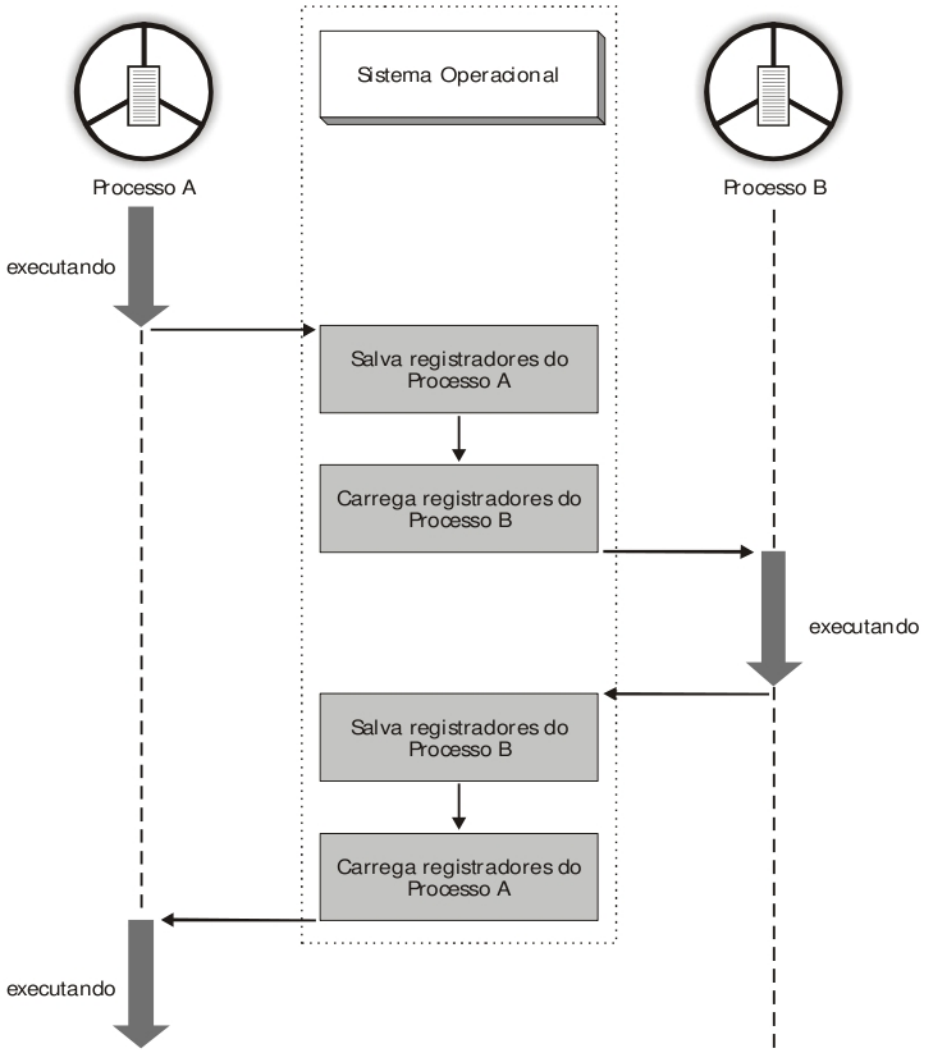
5 de setembro de 2014

# Mudança de Contexto

- À mudança de um processo a outro dá-se o nome de mudança de Contexto
  - O escalonador deve se preocupar com a eficiência da CPU, pois o chaveamento de processos é complexo e custoso:
    - Afeta desempenho do sistema e satisfação do usuário;
    - De nada adianta ocupar 100% da CPU se 90% é gasto com escalonamento
- Linux:
  - As operações de troca de contexto para Intel x86 estão definidas no arquivo arch/ia64/kernel/process.c (fontes)

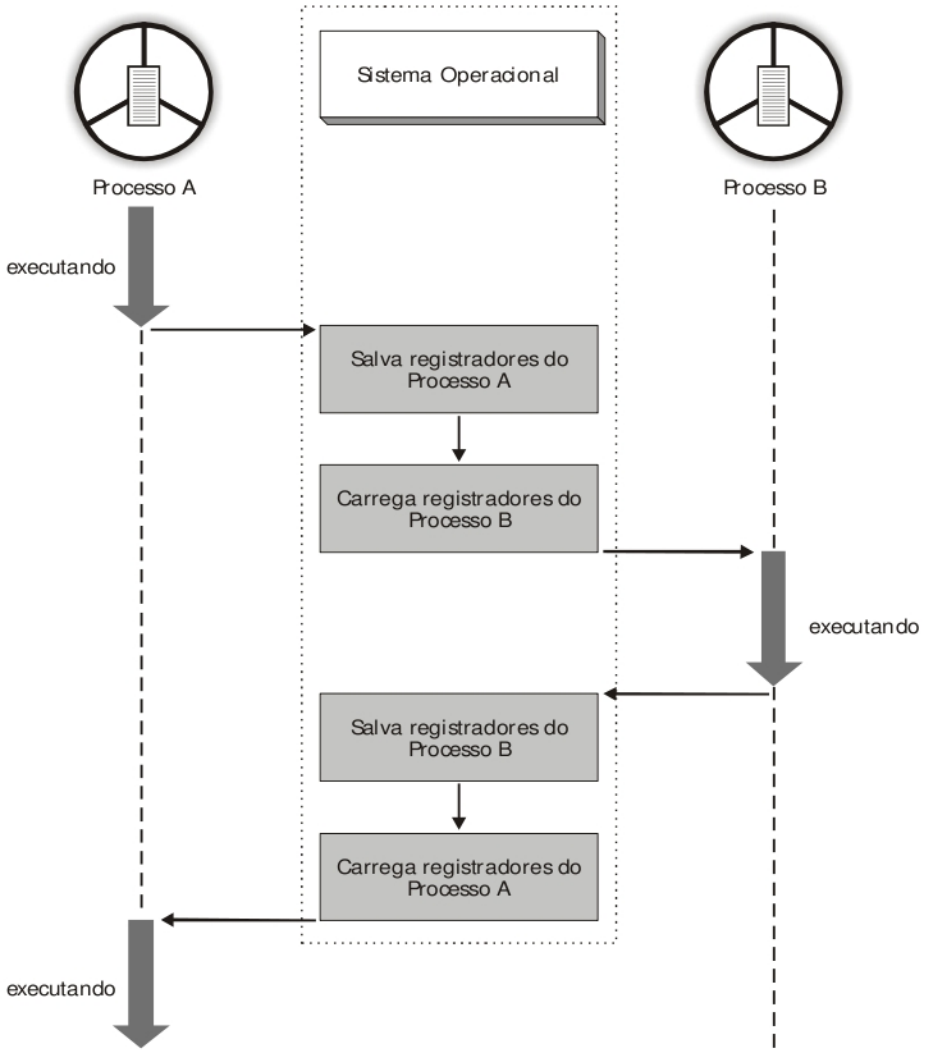
# Mudança de Contexto

- A mudança de contexto leva a um overhead de tempo
- Tarefa cara:
  - Deve-se salvar as informações do processo que está deixando a CPU em seu BCP
    - Salvar o conteúdo dos registradores



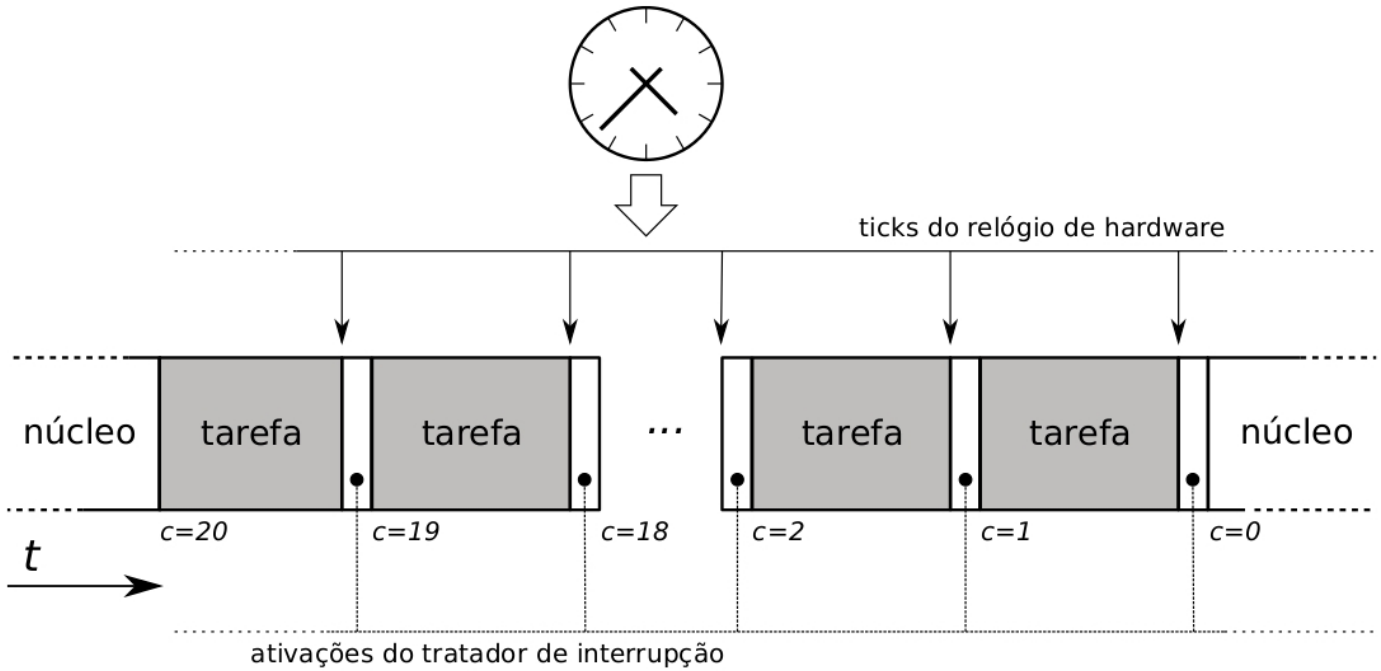
# Mudança de Contexto

- Tarefa cara:
  - Carregar as informações do processo que será colocado na CPU
    - Copiar do BCP o conteúdo dos registradores;



# Mudança de Contexto

- Feita a cada vez que se interrompe a CPU
  - Ativando o tratador de interrupções (*interrupt handler*)
  - As ativações periódicas do tratador de interrupção, pelo clock, são chamadas de ticks do relógio

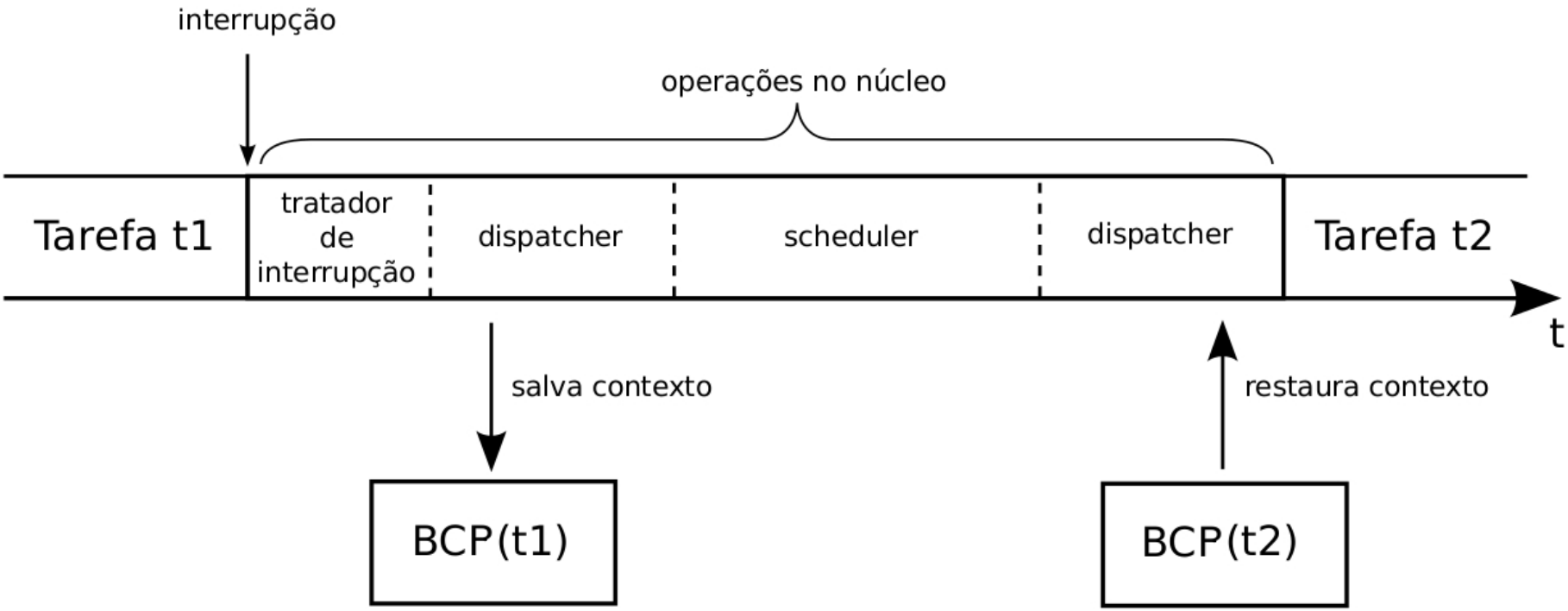


# Mudança de Contexto

## Componentes envolvidos

- Despachante (Dispatcher):
  - Armazena e recupera o contexto
  - Atualiza as informações no BCP
  - Processo relativamente rápido (0,1ms)
- Escalonador (Scheduler):
  - Escolhe a próxima tarefa a receber o processador
  - Parte mais demorada
    - Por isso muitos SOs executam o escalonador apenas quando há necessidade de reordenar a fila de processos prontos. Nesse caso, o despachante sempre ativa o primeiro processo da fila.

# Mudança de Contexto




Passos de uma mudança de contexto

# Escalonamento de Processos

## Situações nas quais escalonamento é necessário:

- Um novo processo é criado;
  - Quando um processo cria outro, qual executar? Pai ou filho?
- Um processo chegou ao fim e um processo pronto deve ser executado
  - Se nenhum estiver pronto, é executado um processo ocioso gerado pelo sistema (no Unix isso não ocorre, pois sempre há o init rodando)

# Escalonamento de Processos

## Situações nas quais escalonamento é necessário:

- Quando um processo é bloqueado (dependência de E/S), outro deve ser executado;
  - O bloqueio é feito via trap – chamada ao SO
  - O processo bloqueado é colocado em uma fila de bloqueados
- Quando uma interrupção de E/S ocorre o escalonador deve decidir por:
  - Executar o processo que estava esperando esse evento;
  - Continuar executando o processo que já estava sendo executado; ou
  - Executar um terceiro processo que esteja pronto para ser executado.

# Escalonamento de Processos

- Preempção:
  - Quando um processo pode, por algum motivo, perder seu uso da CPU.
- Tempo de execução de um processo é imprevisível:
  - Se o hardware do clock gera interrupções em intervalos entre 50 a 60 hz (ocorrências por segundo), uma decisão de escalonamento pode ser tomada a cada interrupção ou a cada k interrupções – veremos mais tarde
- A decisão depende do algoritmo de escalonamento

# Escalonamento de Processos

- Algoritmos de escalonamento podem ser divididos em duas categorias dependendo de como essas interrupções são tratadas:
  - Preemptivo:
    - Suspende o processo sendo executado;
    - Contempla a preempção das tarefas (provoca uma interrupção forçada de um processo para que outro, com a preempção, possa usar a CPU)
    - A preempção pode surgir tão somente porque a fatia de tempo do primeiro processo acabou
    - Requer a existência de uma interrupção de relógio ao fim da fatia de tempo, para que o controle sobre a CPU seja devolvido ao escalonador

# Escalonamento de Processos

- Algoritmos de escalonamento podem ser divididos em duas categorias dependendo de como essas interrupções são tratadas:
  - Não-Preemptivo:
    - Permite que o processo sendo executado continue executando, só parando caso termine de executar, solicite uma operação de entrada/saída ou libere explicitamente o processador, voltando à fila de tarefas prontas
    - Não contempla as preempções
    - O escalonador não interrompe os processos que estão em execução
    - Nenhuma decisão de escalonamento é tomada durante as interrupções do relógio

# Algoritmos de Escalonamento: Categorias

- Sistemas em Batch (Lote):
  - Usuários não esperam por respostas rápidas
  - Algoritmos preemptivos ou não-preemptivos
  - Pouca alternância entre processos $\rightarrow$ melhor desempenho
- Sistemas Interativos:
  - Interação constante do usuário
    - Processo interativo $\rightarrow$ espera comando e executa comando
  - Algoritmos preemptivos – evitam que um processo se aposse da CPU

# Algoritmos de Escalonamento: Categorias

- Sistemas em Tempo Real:
  - Processos são executados mais rapidamente
  - Tempo é crucial $\rightarrow$ sistemas críticos
- Com graus diferentes, comunicação e sincronismo entre processos é importante
  - Especialmente em sistemas interativos
  - Obtidos por meio de:
    - Pipes, semáforos, monitores
    - Veremos mais adiante detalhes

# Algoritmos de Escalonamento: Objetivos

- Comuns a todos os sistemas:
  - Justiça (Fairness): cada processo deve receber uma parcela justa de tempo da CPU
    - Processos da mesma categoria devem receber a mesma fatia de tempo
  - Balanceamento: diminuir a ociosidade do sistema
    - Manter todas as partes do sistema ocupadas
  - Cumprimento das políticas do sistema – prioridade de processos

# Algoritmos de Escalonamento: Objetivos

- Sistemas em Batch (Lote):
  - Vazão (throughput): maximizar o número de jobs executados por unidade de tempo (e.g. hora)
  - Turnaround time (tempo de retorno): minimizar o tempo médio entre a entrada e o término
    - Indica quanto tempo, em média, o usuário espera pelo fim de um trabalho
    - Tempo entre a criação do processo e seu encerramento, incluindo processamento e espera
    - Pode ocorrer que tenha grande vazão, e pequeno tempo de retorno – quando o algoritmo de escalonamento prioriza jobs pequenos, os grandes podem nunca ser executados
  - Eficiência: CPU deve estar 100% do tempo ocupada;

# Algoritmos de Escalonamento: Objetivos

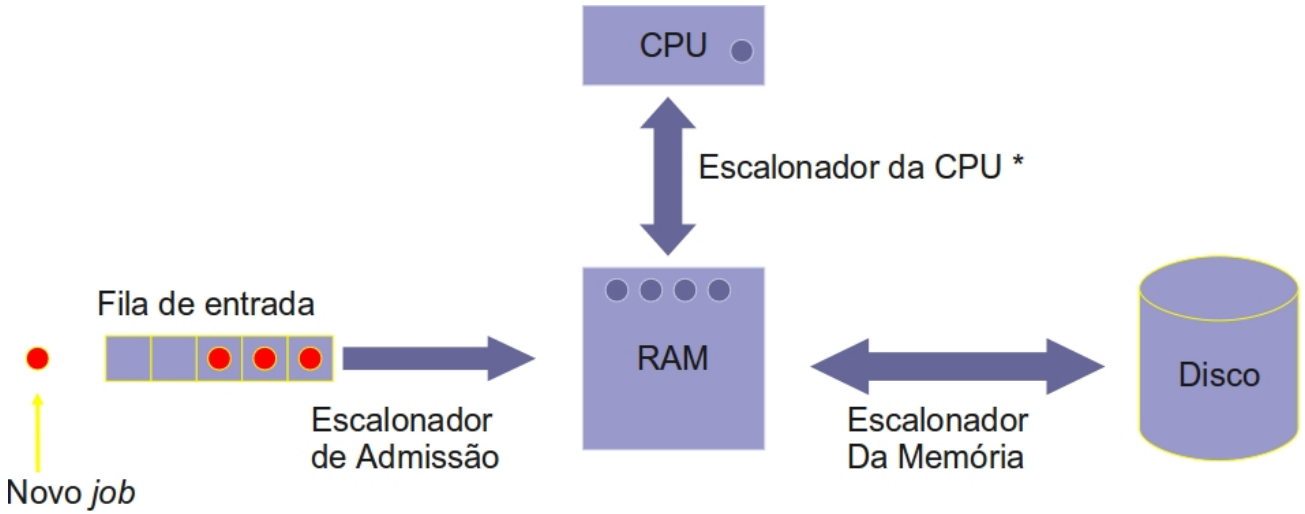
- Sistemas Interativos:
  - Tempo de resposta: minimizar o tempo entre a emissão de um comando e a obtenção do resultado
    - Requisições do usuário devem ter precedência sobre trabalho em segundo plano
  - Proporcionalidade: satisfazer às expectativas do usuário
    - Satisfazer a intuição que o usuário tem sobre quanto tempo as coisas devem durar
    - Nem sempre é possível

# Algoritmos de Escalonamento: Objetivos

- Sistemas em Tempo Real:
  - Cumprimento dos prazos, evitando assim perda de dados
  - Previsibilidade: Prevenir perda da qualidade dos serviços oferecidos
    - Importante em alguns sistemas
    - Ex: evitar a degradação da qualidade em sistemas multimídia (por algum atraso no cumprimento de prazo de algum processo)

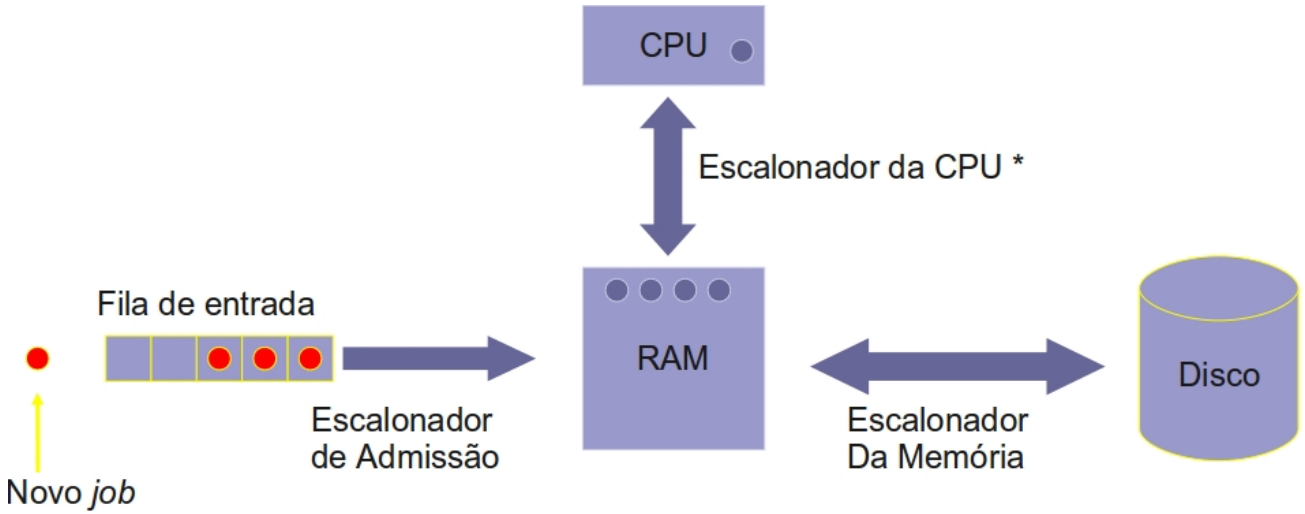
# Escalonamento em Batch

- Sistemas em lote permitem escalonamento em três níveis
  - Escalonador de admissão: decide qual job da fila de entrada será admitido no sistema
    - Processos menores primeiro;
    - Processos com menor tempo de acesso à CPU e maior tempo de interação com dispositivos de E/S;



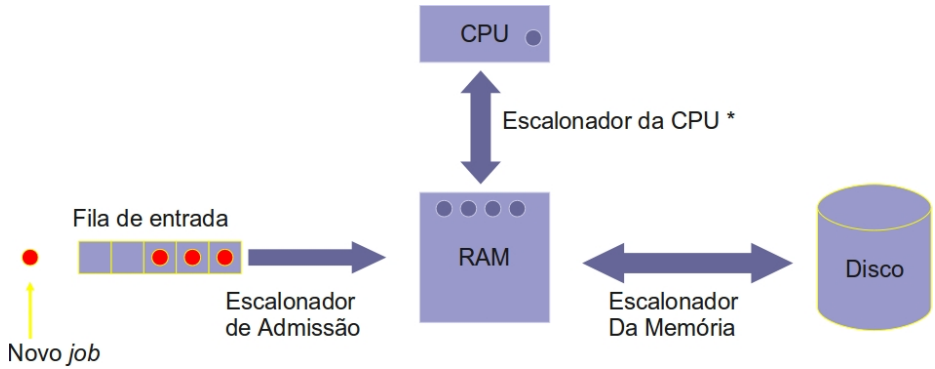
# Escalonamento em Batch

- Sistemas em lote permitem escalonamento em três níveis
  - Escalonador de memória: decide quais processos serão mantidos na memória ou no disco
    - Necessário quando o número de processos (referentes a jobs) é grande demais para a memória
    - Alguns devem ser levados ao disco (swap)



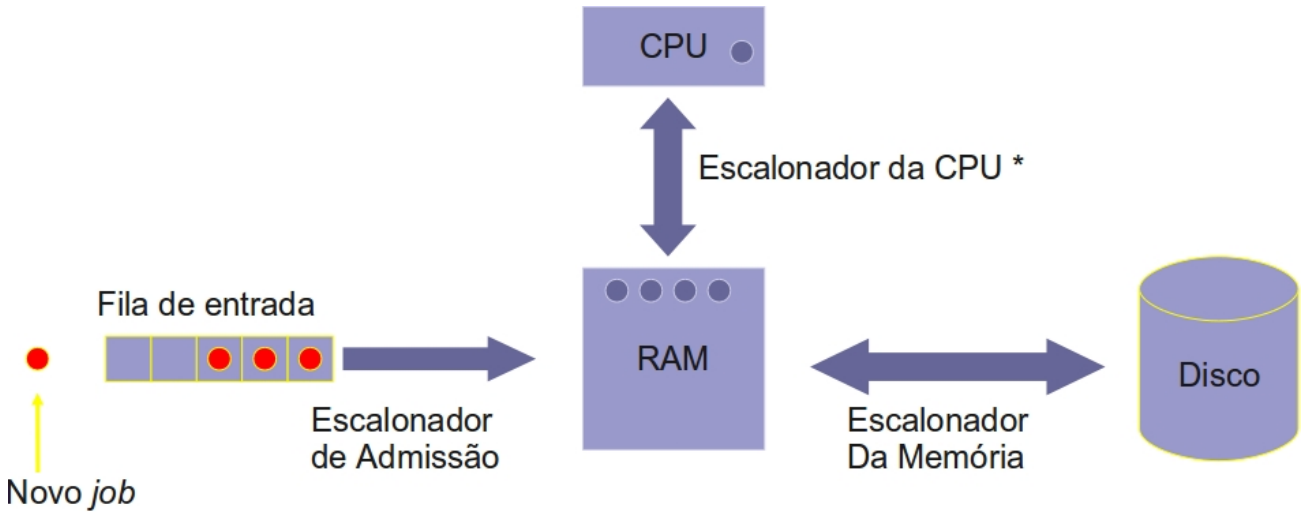
# Escalonamento em Batch

- Sistemas em lote permitem escalonamento em três níveis
  - Escalonador de memória: critérios para tomada de decisões sobre quais processos vão para a memória:
    - Quanto tempo da CPU o processo utilizou da última vez?
    - Qual o tamanho do processo? (pequenos não vão ao disco $\rightarrow$ operação cara)
    - Há quanto tempo o processo está esperando?
    - Qual a importância do processo?



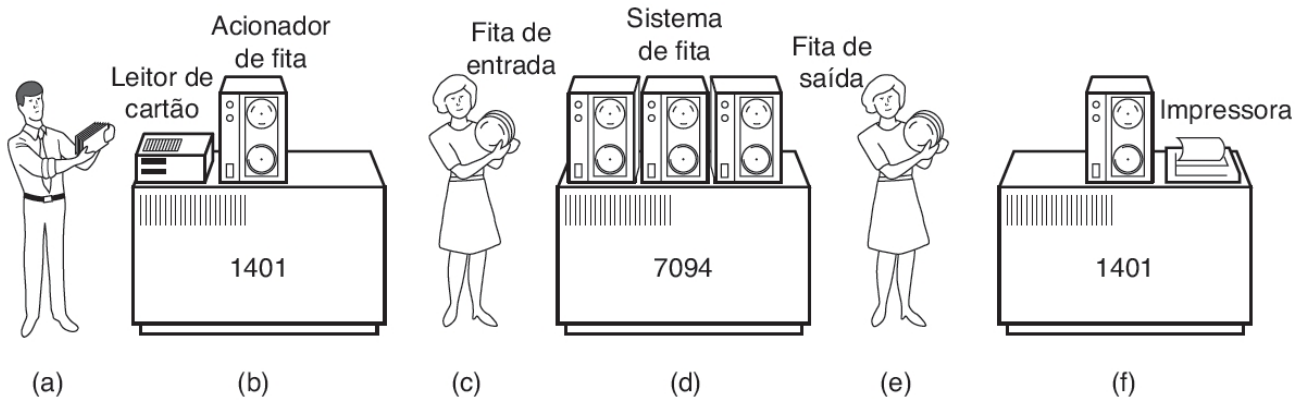
# Escalonamento em Batch

- Sistemas em lote permitem escalonamento em três níveis
  - Escalonador de CPU: seleciona qual o próximo processo (da memória) a ser executado



# Escalonamento em Batch: Algoritmos

- First-Come First-Served (ou FIFO)
- Shortest Job First (SJF)
- Shortest Remaining Time Next (SRTN)



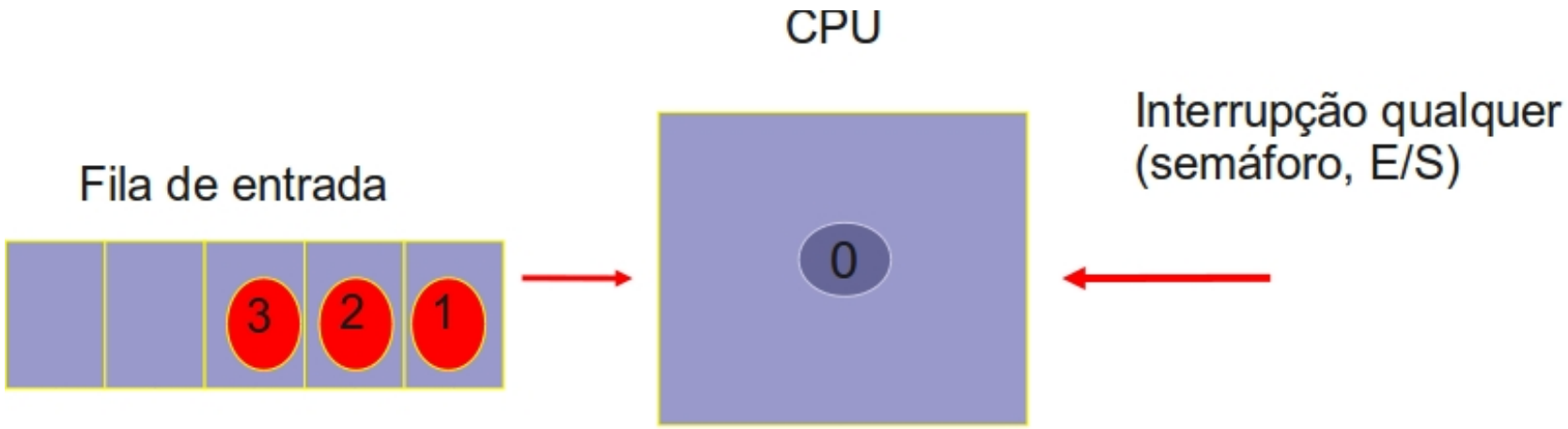
# Escalonamento em Batch: Algoritmos

## First-Come First-Served

- Processos são executados na CPU seguindo a ordem de requisição
  - Há fila única de processos prontos
  - Cada processo roda o tempo que quiser
  - Novos processos entram no final da fila
  - Se um processo que está rodando é bloqueado, o primeiro na fila é o próximo a executar
  - Quando um processo bloqueado fica pronto, é recolocado no final da fila

# Escalonamento em Batch: Algoritmos

## First-Come First-Served



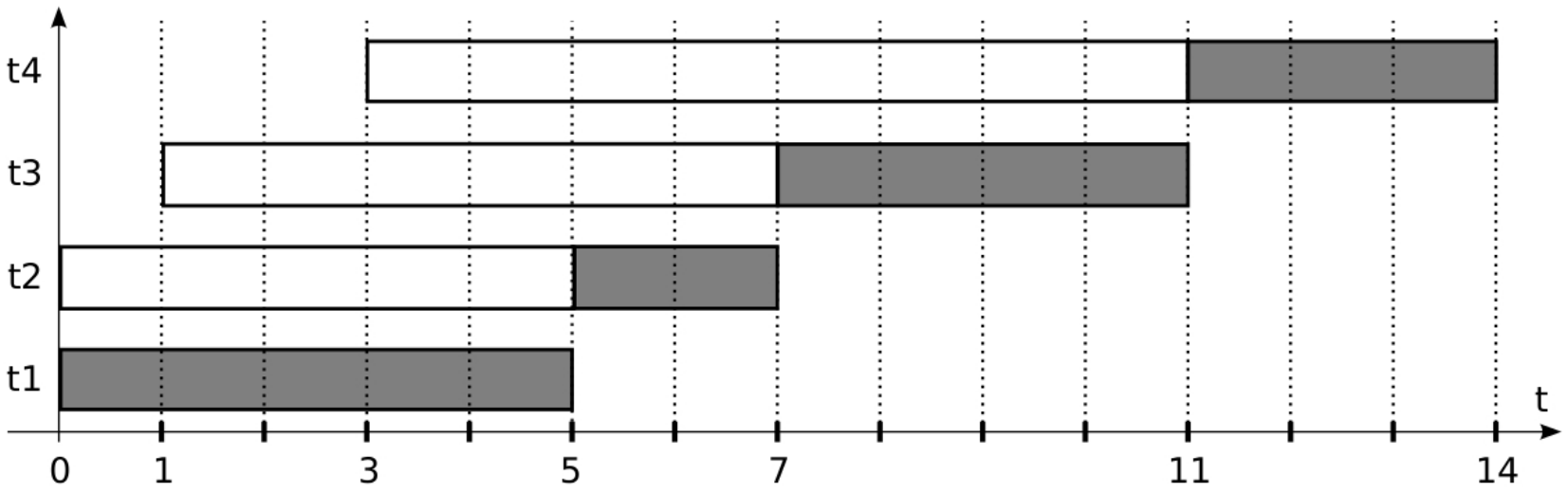

CPU não controla o tempo dos processos!
(não-preemptivo)

# Escalonamento em Batch: Algoritmos

## First-Come First-Served: Exemplo

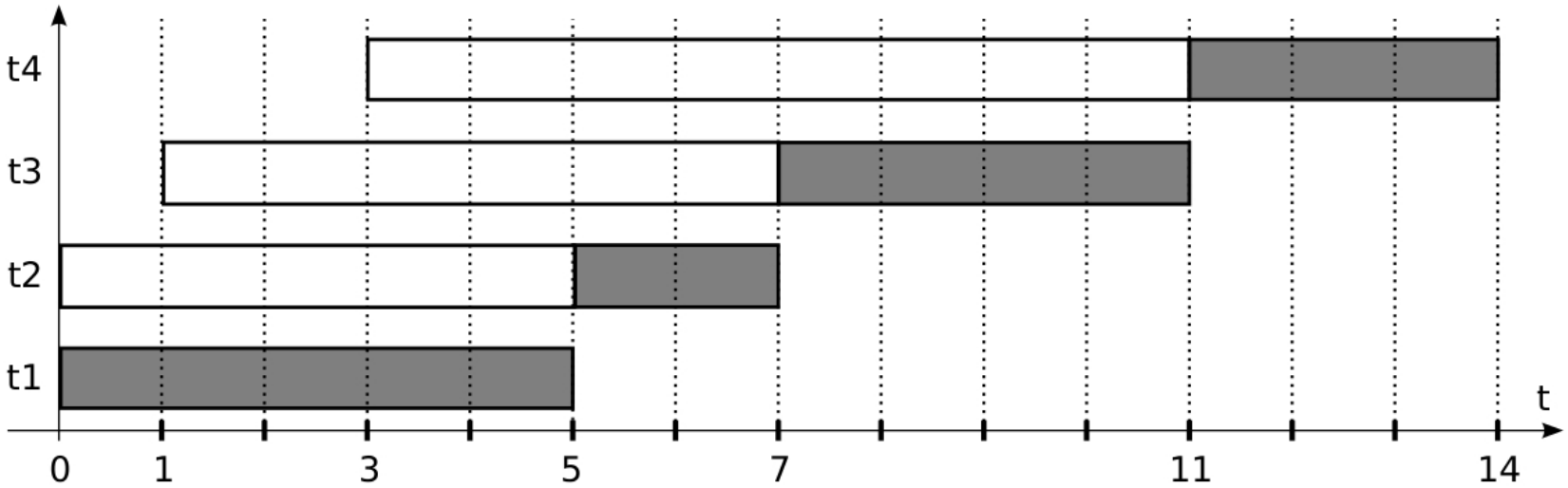
| tarefa | $t_1$ | $t_2$ | $t_3$ | $t_4$ |
|---|---|---|---|---|
| ingresso | 0 | 0 | 1 | 3 |
| duração | 5 | 2 | 4 | 3 |

Fila de processos prontos (chegada ao sistema e duração prevista)



# Escalonamento em Batch: Algoritmos

## First-Come First-Served: Exemplo




- Tempo médio de execução (*turnaround* médio):

$$T = \frac{t_1 + t_2 + t_3 + t_4}{4} = \frac{(5-0) + (7-0) + (11-1) + (14-3)}{4} = \frac{33}{4} = 8,25s$$

# Escalonamento em Batch: Algoritmos

## First-Come First-Served

- Não-preemptivo
  - Não se interrompe processos (no máximo bloqueiam em E/S)
- Vantagem: Fácil de entender e programar
- Desvantagem:
  - Não leva em conta a importância das tarefas nem seu comportamento em relação aos recursos
  - Ineficiente quando se tem processos que demoram na sua execução

# Escalonamento em Batch: Algoritmos

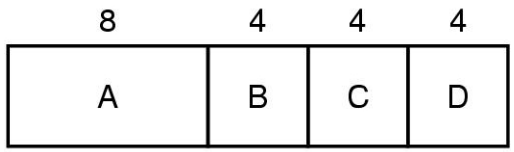
## First-Come First-Served

- Desvantagem: Ex
  - Fila com um processo que demora 1s por vez, e faz E/S, e outros rápidos que executem, cada um, 1000 E/S
  - O primeiro executa 1s e bloqueia (E/S). Os demais, executam rapidamente e bloqueiam (mantendo a fila)
  - O primeiro desbloqueia, e roda mais 1s, seguido pelos demais
  - Os outros estão de fato fazendo 1 E/S por segundo $\rightarrow$ cada um termina em 1000s. Se houvesse preempção do inicial, a cada 10ms, eles levariam $10ms \times 1000 = 10s$ somente.

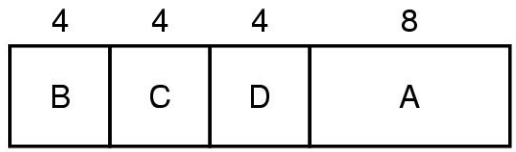
# Escalonamento em Batch: Algoritmos

## Shortest Job First

- Não-preemptivo
- Supõe que se sabe de antemão o tempo de execução de todos os processos
- Menor processo da fila de processos prontos é executado primeiro
  - Se algum bloquear por E/S, voltará à fila ordenado




Execução na ordem normal




Execução na ordem shortest job first

# Escalonamento em Batch: Algoritmos

## Shortest Job First

- Menor turnaround (médio):
  - Ex:

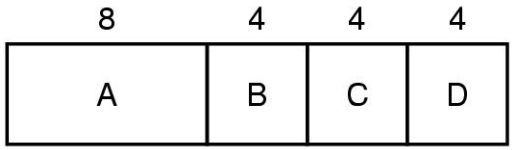
$$\begin{array}{lll} A & \rightarrow & \text{(tempo) } a \\ B & \rightarrow & b + a \\ C & \rightarrow & c + b + a \\ D & \rightarrow & d + c + b + a \end{array}$$

  - Tempo médio-turnaround $(4a+3b+2c+d)/4$
  - Se $a<b<c<d$ tem-se o mínimo tempo médio – fazemos quem contribui mais para a média ser o menor
  - O menor tempo médio esperado para um job terminar

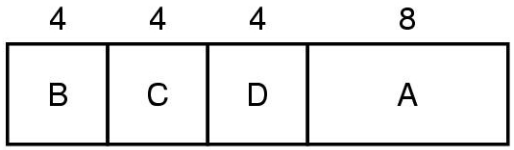
# Escalonamento em Batch: Algoritmos

## Shortest Job First

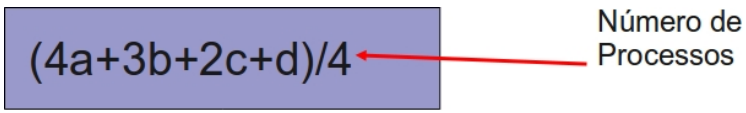
- Menor turnaround (médio):
- Ex:




*Em ordem:*
Turnaround A = 8
Turnaround B = 12
Turnaround C = 16
Turnaround D = 20
Média → 56/4 = 14




*Menor job primeiro:*
Turnaround B = 4
Turnaround C = 8
Turnaround D = 12
Turnaround A = 20
Média → 44/4 = 11



# Escalonamento em Batch: Algoritmos

## Shortest Job First

- Desvantagem:
  - Todos os processos precisam ser conhecidos de antemão
  - Se processos curtos começarem a chegar, os longos podem demorar a serem executados (após seu bloqueio voluntário) – inanição (*starvation*)
  - Processos precisam estar disponíveis simultaneamente $\rightarrow$ pode ocasionar problemas
  - Ex: 5 processos: A $\rightarrow$ E
  - Com tempo de execução $\{2,4,1,1,1\}$
  - Tempos de chegada $\{0,0,3,3,3\}$

# Escalonamento em Batch: Algoritmos

## Shortest Job First

- Desvantagem:
  - Como somente A e B estão disponíveis no início, a ordem fica {A,B}+{C,D,E}

$$
\begin{array}{lll}
A & \rightarrow & \text{(tempo) } 2 = 2 \\
B & \rightarrow & 4 + 2 = 6 \\
C & \rightarrow & 1 + (4 + 2 - 3) = 4 \\
D & \rightarrow & 1 + 1 + (4 + 2 - 3) = 5 \\
E & \rightarrow & 1 + 1 + 1 + (4 + 2 - 3) = 6
\end{array}
$$

$$(2+6+4+5+6)/5 = 4{,}6$$

  - Se todos estivessem disponíveis simultaneamente, a ordem seria {C,D,E,A,B} $\rightarrow (1\times5 + 1\times4 + 1\times3 + 2\times2 + 4)/5 = 4$

# Escalonamento em Batch

## Shortest Job First

- Utilidades Inusitadas: Filas de bancos
  - Por vezes, além da distinção prioritário/não prioritário, encontramos outra:
    - Até 3 contas e mais de 3 contas
  - A ideia é passar mais rapidamente os clientes que têm pouco tempo de uso de caixa
    - Objetivo:

# Escalonamento em Batch

## Shortest Job First

- Utilidades Inusitadas: Filas de bancos
  - Por vezes, além da distinção prioritário/não prioritário, encontramos outra:
    - Até 3 contas e mais de 3 contas
  - A ideia é passar mais rapidamente os clientes que têm pouco tempo de uso de caixa
    - Objetivo: Minimizar o *turnaround* médio
    - Algoritmo:

# Escalonamento em Batch

## Shortest Job First

- Utilidades Inusitadas: Filas de bancos
  - Por vezes, além da distinção prioritário/não prioritário, encontramos outra:
    - Até 3 contas e mais de 3 contas
  - A ideia é passar mais rapidamente os clientes que têm pouco tempo de uso de caixa
    - Objetivo: Minimizar o *turnaround* médio
    - Algoritmo: Shortest Job First

# Escalonamento em Batch

## Shortest Job First

- Utilidades Inusitadas: Filas de bancos
  - Por vezes, além da distinção prioritário/não prioritário, encontramos outra:
    - Até 3 contas e mais de 3 contas
  - A ideia é passar mais rapidamente os clientes que têm pouco tempo de uso de caixa
    - Objetivo: Minimizar o *turnaround* médio
    - Algoritmo: Shortest Job First
    - Deve-se, de tempos em tempos, liberar os maiores, para evitar inanição

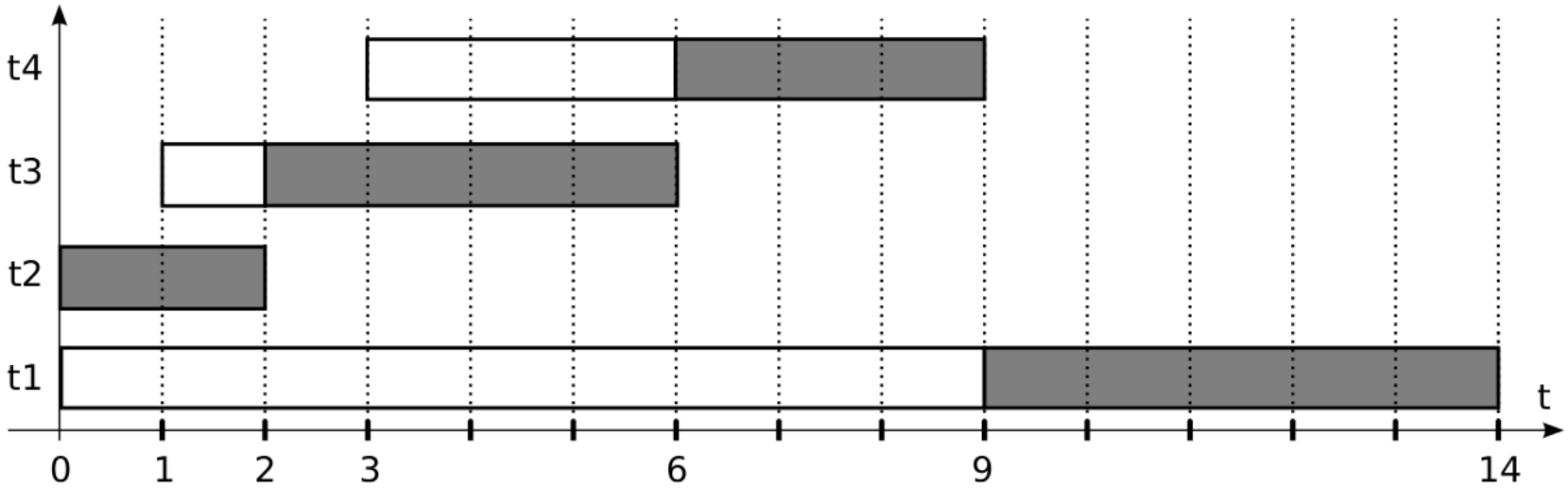
# Escalonamento em Batch: Algoritmos

## Shortest Remaining Time Next

- Versão preemptiva do Shortest Job First
- Processos com menor tempo restante são executados primeiro
  - Precisamos saber previamente seus tempos
  - Se um processo novo chega e seu tempo de execução (total) é menor do que do processo corrente na CPU, a CPU suspende o processo corrente e executa o processo que acabou de chegar
    - Senão insere-o no local apropriado na fila de prontos

# Escalonamento em Batch: Algoritmos

## Shortest Remaining Time Next: Exemplo




- Tempo médio de execução (*turnaround* médio):

$$T = \frac{t_1 + t_2 + t_3 + t_4}{4} = \frac{(14-0) + (2-0) + (6-1) + (9-3)}{4} = \frac{27}{4} = 6,75s$$

# Escalonamento em Batch: Algoritmos

## Shortest Remaining Time Next

- Desvantagem:
  - Processos que consomem mais tempo podem demorar muito para serem finalizados se muitos processos pequenos chegarem! – inanição (*starvation*)