# Gerenciamento de Memória

LICENCIATURA EM COMPUTAÇÃO

# Memória Primária x Memória Secundária

- Memória real: conhecida por vários outros termos:
  - Memória principal.
  - Memória física.
  - Memória primária.
  - Memória RAM (*Random Access Memory*).
- Para ser executado, um programa tem que estar na RAM.
- Executar a partir do disco seria muito lento (inaceitável).
- Diferença entre tempos de acesso é de 1/1.000.000.
  - RAM = 10ns
  - Disco = 10ms

# Gerenciador de Memória

- RAM: Sempre foi um recurso caro e escasso.
  - Programas eram pequenos: para caber inteiros na memória.
- SO monotarefa: gerenciadores de memória eram simplificados.
- SO multitarefa: gerenciadores de memória são sofisticados.
  - Têm grande impacto no desempenho global do sistema.
- Hoje: combinação de *Hardware* e *Software* para ganhar desempenho.

# Estratégias de Gerenciamento de Memória

- Se novos processos são alocados na RAM, considera-se:
- Estratégias de busca:
  - QUANDO carregar um novo processo na RAM?
- Estratégias de posicionamento:
  - ONDE carregar um novo processo na RAM?
  - Primeiro, melhor ou pior encaixe.
- Estratégias de substituição:
  - QUAIS processos deverão ser substituídos.
  - Qualquer um, o mais antigo, o mais ocioso, o mais usado.

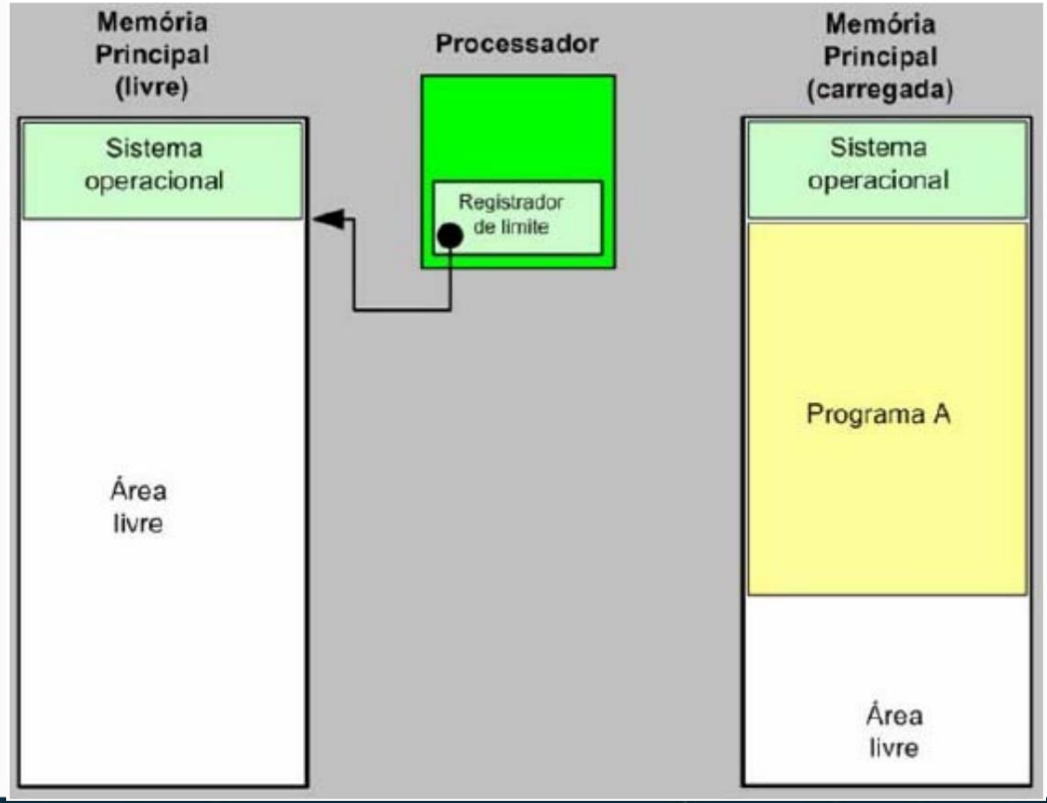
# Política de Gerenciamento de Memória

- Varia em cada SO, mas, de modo geral, objetiva:
  - Manter na RAM o maior número de processos residentes.
  - Minimizar operações de I/O em disco.
  - Maximizar o uso do processador (*throughput*).
  - Carregar novos processos em espaço livre na RAM: *swap*.
  - Executar programas maiores na RAM (*overlay* e memória virtual).
  - Proteger áreas do SO e áreas dos processos na RAM.
  - Ser “transparente” para os programas.

# Formas de alocação (particionamento da RAM)

- **Alocação contígua simples** (monoprogramação).
  - Usa toda a memória livre como um bloco único.
- **Alocação particionada** (multiprogramação).
  - **Estática** (ou de partições fixas).
    - Vários blocos de tamanhos predefinidos.
  - **Dinâmica** (ou de partições variáveis).
    - Não usa o conceito de blocos.
    - Aloca processos de forma adjacente.
- **Gerenciador de memória:** controla particionamento da RAM.
  - Quais partes estão alocadas para processos em execução?
  - Quais partes estão livres para novos processos?

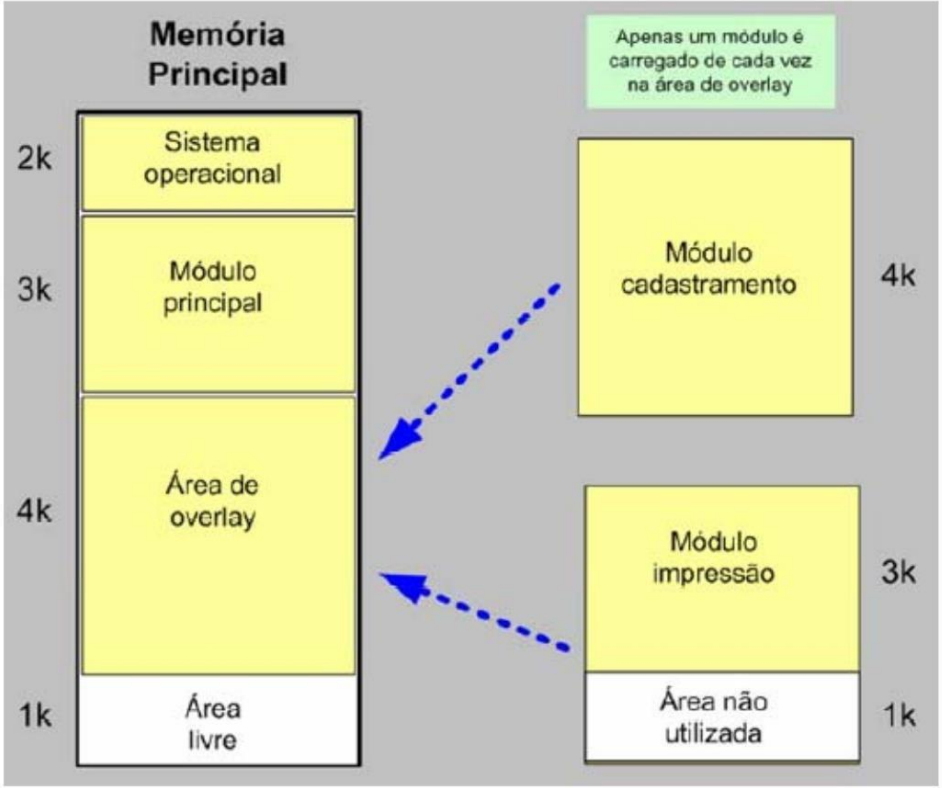
# Alocação contígua simples (monoprogramação)



# Alocação contígua simples (monoprogramação)

- Implementada nos primeiros SO monousuário/monotarefa.
- RAM dividida em duas partições: SO (protegida) e programa.
- Registrador de fronteira (ou de limite) estabelece a divisão.
- Um processo em execução por vez.
  - Menor ou igual ao bloco de programa.
  - Pode acessar toda a memória.
- *Overlay*: uma forma de ultrapassar o limite de memória.
- **Vantagem:** fácil implementação.
- **Desvantagem:** mau aproveitamento da RAM.

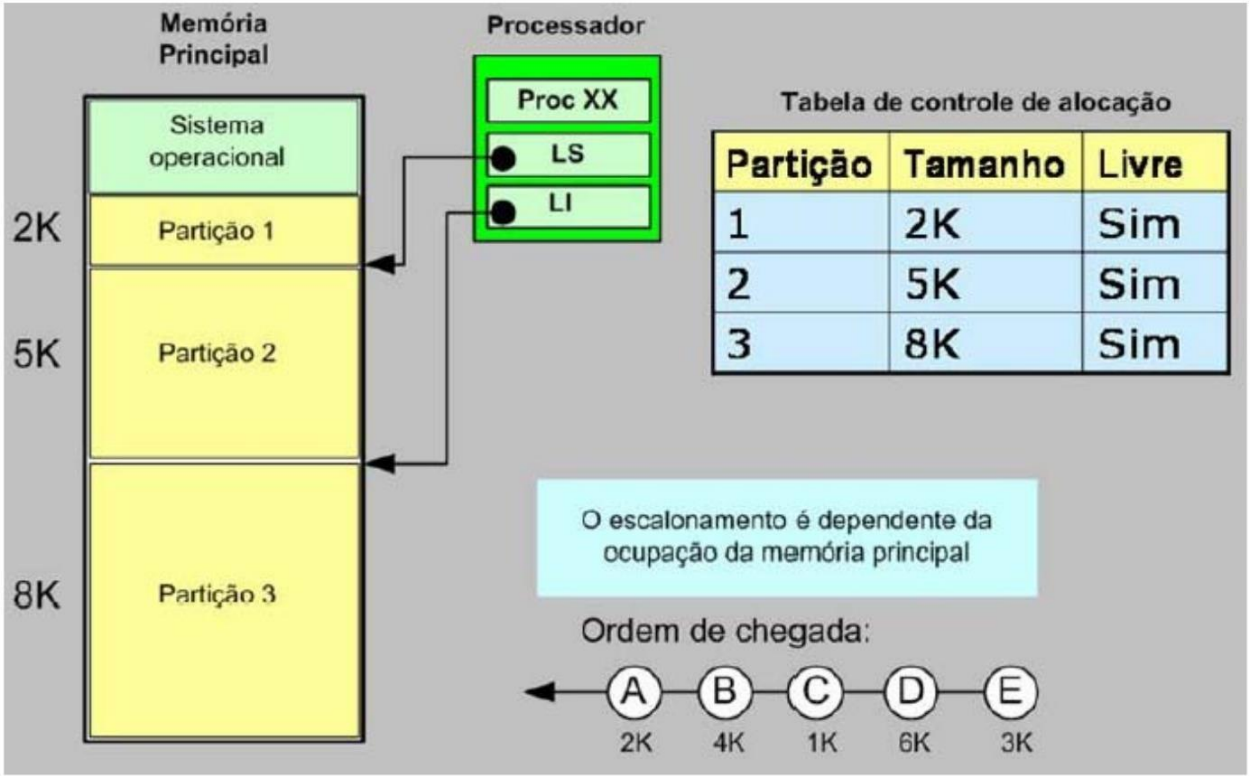
# *Overlay* (Sobreposição)



# *Overlay* (Sobreposição)

- Permite executar um **processo maior** que a RAM disponível.
- **Divide processo:** um bloco principal e *n* blocos secundários.
  - Bloco principal fica na RAM o tempo (chama os secundários).
  - Blocos secundários.
    - São carregados na RAM, um de cada vez.
    - Independem uns dos outros.
    - Dependem do bloco principal para carga e execução.
- Troca de blocos gera sobrecarga no SO.
- Programador decide sobre a divisão em blocos.
- Blocos secundários ≤ área de *overlay*.

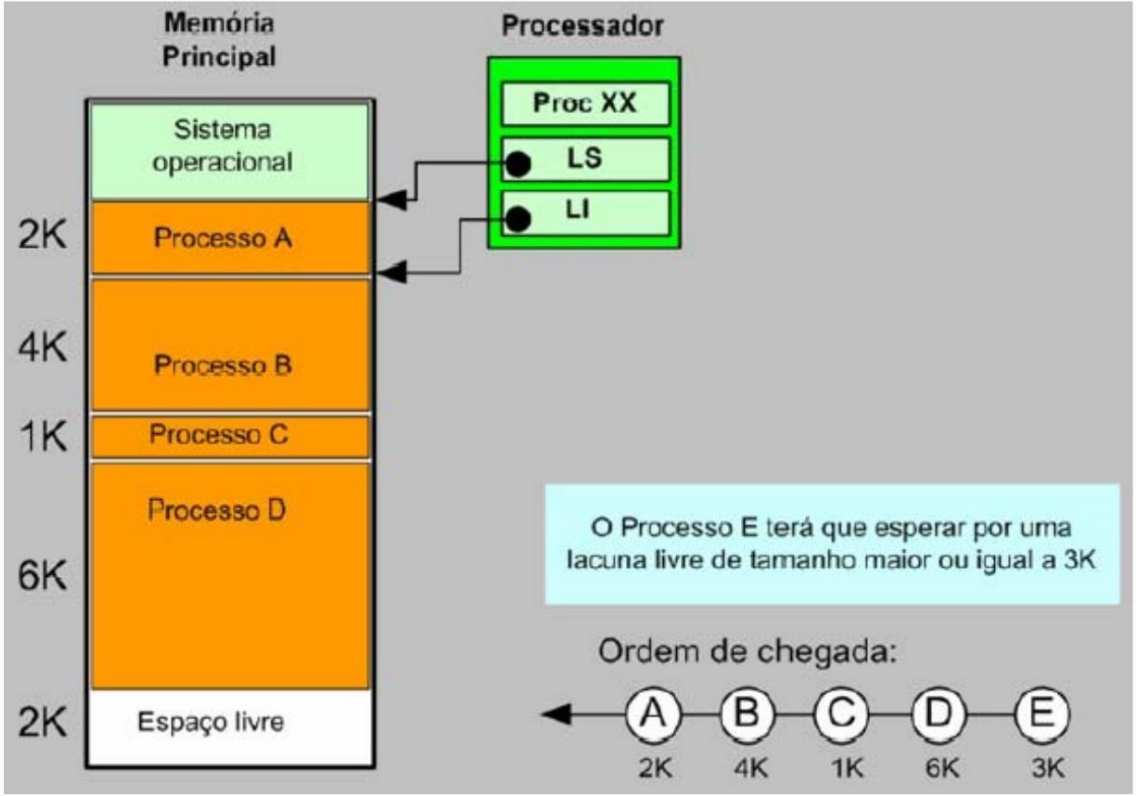
# Alocação Particionada Estática



# Alocação Particionada Estática

- Também conhecida como alocação por partição fixa.
- Memória é dividida em partições de tamanhos fixos.
  - Estabelecidas durante a inicialização do sistema.
  - Em cada partição se encaixa um único processo.
- Alterar tamanho das partições: só reinicializando o sistema.
- **Tabela de controle**: para gerir os espaços alocados/livres.

# Alocação Particionada Dinâmica



# Alocação Particionada Dinâmica

- Também conhecida como alocação por partição variável.
- RAM dividida em 2 partições: SO (protegida) e programas.
- Busca **eliminar desperdícios** dos métodos anteriores.
- Processos carregados sequencialmente, sem espaços livres.
- Processos **encerrados**:
  - Deixam lacunas (espaços livres) entre os blocos.
  - É a fragmentação externa (área livre, não alocável).
- Processo **novo**:
  - Pode encaixar em qualquer lacuna.
  - Se a lacuna>processo → permanece a fragmentação.

# Alocação Particionada Dinâmica

- **Como resolver o problema das lacunas?**

- Compactação das lacunas (relocação)
  - Gera uma enorme sobrecarga no sistema.
  - **Elimina** a fragmentação externa.
- Fusão das lacunas adjacentes (coalescência)
  - Gera menos sobrecarga no sistema.
  - **Diminui** a fragmentação externa.

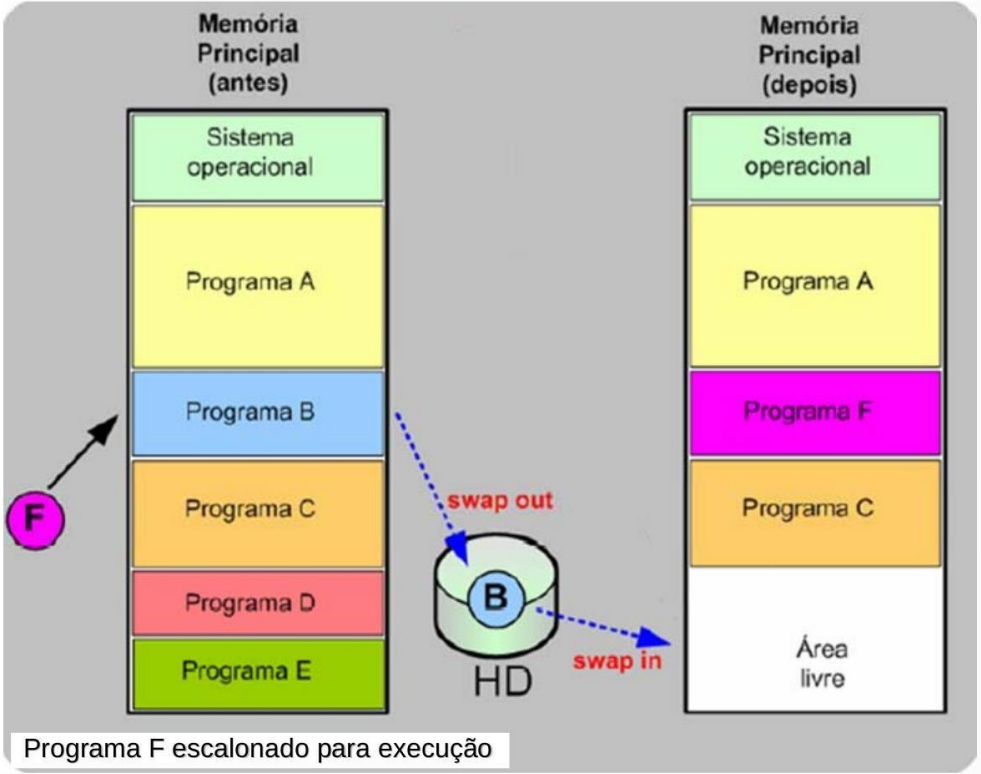
# Alocação Particionada Dinâmica

- **Estratégias de Alocação:**
  - Como **alocar novos processos** que vão chegando?
  - Existem 3 estratégias básicas para alocação das lacunas:
    - *First-fit* (primeiro encaixe)
    - *Best-fit* (melhor encaixe)
    - *Worst-fit* (pior encaixe)

# Alocação Particionada Dinâmica

- ***First-fit:*** varre a Tabela de Endereços Livres (TEL) e encaixa na primeira lacuna possível.
  - Baseado na ordem de endereçamento.
  - TEL não fica ordenada pelos tamanhos das lacunas.
- ***Best-fit:*** varre a TEL toda e encaixa na lacuna que gera menor desperdício de espaço.
  - TEL fica ordenada por **tamanho crescente.**
- ***Worst-fit:*** varre a TEL toda e encaixa na lacuna que gera o maior desperdício de espaço.
  - TEL fica ordenada por **tamanho decrescente.**

# *Swapping* (Permuta)




Programa F escalonado para execução

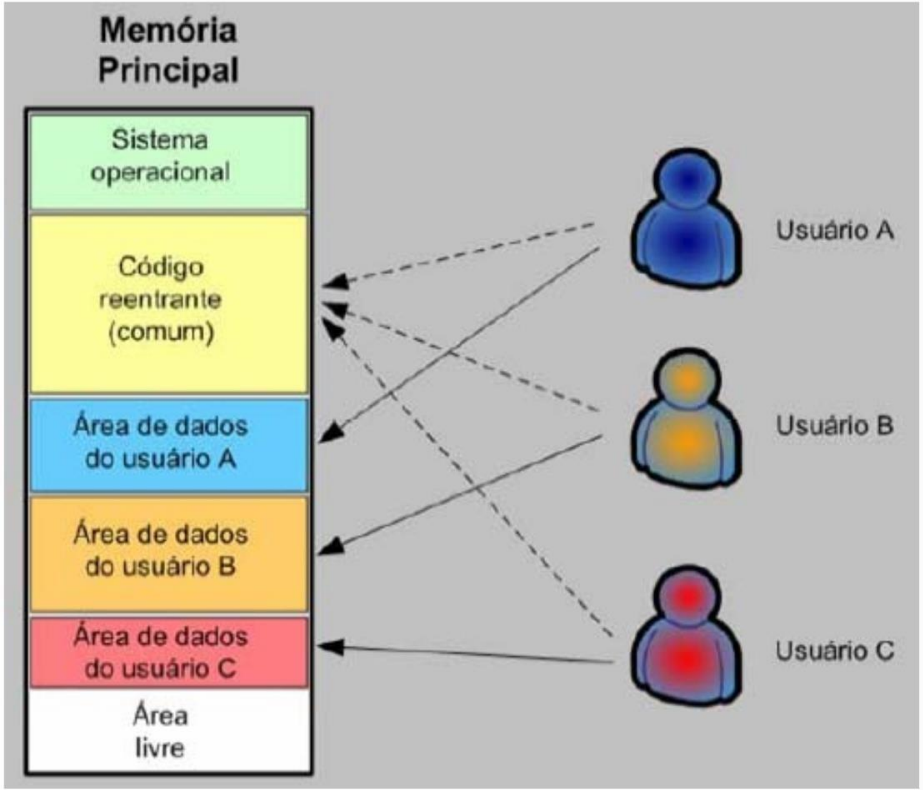
# *Swapping* (Permuta)

- Vários processos em **execução simultânea**:
  - Consomem vários recursos.
  - Tornam a execução muito lenta (sistema degradado).
- ***Swap out***:
  - Interrompe processo e armazena temporariamente em disco.
  - Sobra mais espaço na RAM e a degradação geral diminui.
  - Escolhe processo com menor chance de execução imediata.
    - Para evitar novo *swapping* em seguida.
    - Para dar preferência a processos no estado de espera.

# *Swapping* (Permuta)

- ***Swap in:***
  - Depois, processo interrompido é trazido de volta.
  - Poderá executar com um melhor desempenho.
  - É necessário um *loader* que implemente relocação dinâmica.
  - Pois, o processo pode voltar para uma partição diferente.

# Reentrância (Código Reentrante)



# Reentrância (Código Reentrante)

- Compartilhamento de código: vários usuários/aplicativos.
- Carga do código na RAM é feita apenas uma vez.
- Código deve ser executável e somente leitura.
- Implementada pelos programadores e suportada pelo SO.
- SO detecta código já carregado e o compartilha outra vez.
- Código executável é comum, mas dados são independentes.
- Presente em utilitários, compiladores, *linkers*, aplicativos...
- Evita **desperdícios** quando:
  - Vários usuários invocam um mesmo programa.
  - Um usuário abre várias instâncias do mesmo programa.

# Ler!!!

- **Capítulo 3 do Livro Texto.**

# Atividade

- **Atividade no AVA.**