Sistemas Operacionais

Módulo 7 – Gerência de Memória

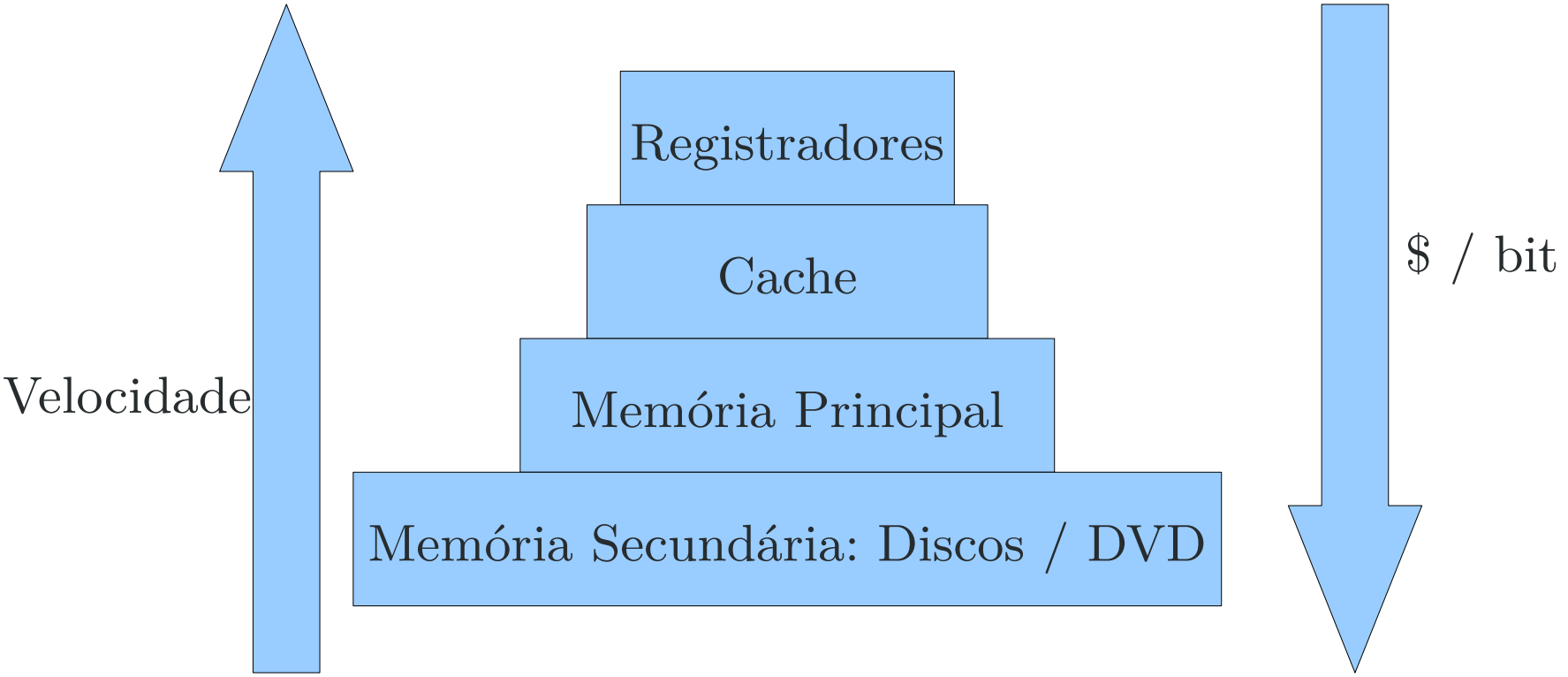
Prof. Daniel Sundfeld
daniel.sundfeld@unb.br

Introdução

- A memória RAM é um dos principais recursos computacionais e precisa ser cuidadosamente gerenciado
- Infelizmente, o recurso não está disponível infinitamente e não existe memória rápida e não-volátil
- Assim, os SOs modernos precisam ser projetados para lidar com essas características

Hierarquia de Memória

- Relembrando:



Introdução

- Os registradores são manipulados diretamente pelos programas ou usados pelo hardware
- Já a memória cache é completamente gerenciada pelo hardware
- O foco da gerência de memória é da **memória principal** (memória RAM)
- Abstrações para **memória secundária** (não-volátil, Discos, DVDs) serão vistas na gerência de arquivos (aula futura)

Roteiro

- Funções básicas
- Esquemas de gerenciamento de memória
  - Sem abstração de memória
  - Com abstração de memória
    - Partições Fixas
    - Partições Variáveis
- Algoritmos de controle de memória
- Swapping e Overlay

Funções Básicas

- Programas são armazenados em memória secundária e precisam ser transferidos para a memória principal
- O sistema operacional deve transferir programas da memória secundária para a principal antes de serem executados
- Devido a diferença de tempo, é desejável que se mantenha em memória os processos em uso e reduzir as operações de E/S para melhor performance do sistema

Funções Básicas

- Desta forma, a gerência de memória deve manter o maior número de processos residentes, buscando maximizar o compartilhamento de recursos
- Mesmo na ausência de espaço livre, deve-se permitir que novos processos sejam criados
- A memória alocada para um programa deve ser protegida de outros programas que estejam em execução

Funções Básicas

- Principal função de um gerente de memória é determinar o esquema de gerência que será utilizado e implementar eficientemente
- Também possui algumas funções secundárias que não dependem do esquema de gerência, como:
  - Controlar a alocação de pedaços de memória a processos
  - Liberar pedaços de memória alocados a processos que não são mais necessários

Sem Abstração de Memória

- Esquema de alocação contígua simples
- Implementada nos primeiros sistemas operacionais
- Cada programa vê uma memória física completa
- O usuário possui controle sobre todo o espaço de memória
- Não é necessário mudanças no hardware e sua implementação é muito simples!

Sem Abstração de Memória

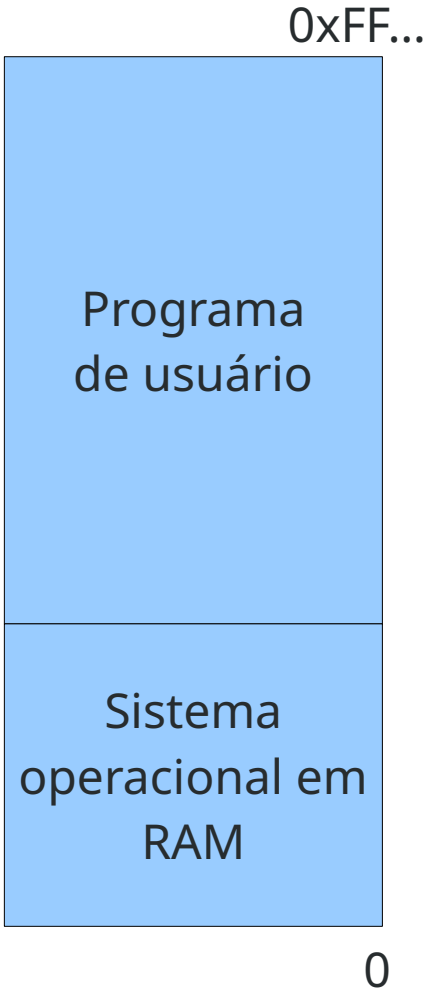
- Nesses sistemas, a instrução:
  `LOAD R1, 1000`
- A CPU move o conteúdo da posição de memória 1000 para o registrador 1
- Normalmente, cada endereço de memória possui 8 bits

Sem Abstração de Memória

- Porém, observe que nesse caso dois programas iguais não podem executar ao mesmo tempo
- Se um programa escreve um valor em uma posição, o outro programa irá sobrescrever esse valor alguma hora
- Porém, o modelo sem abstração de memória permite diferentes organizações na estrutura da memória e SO

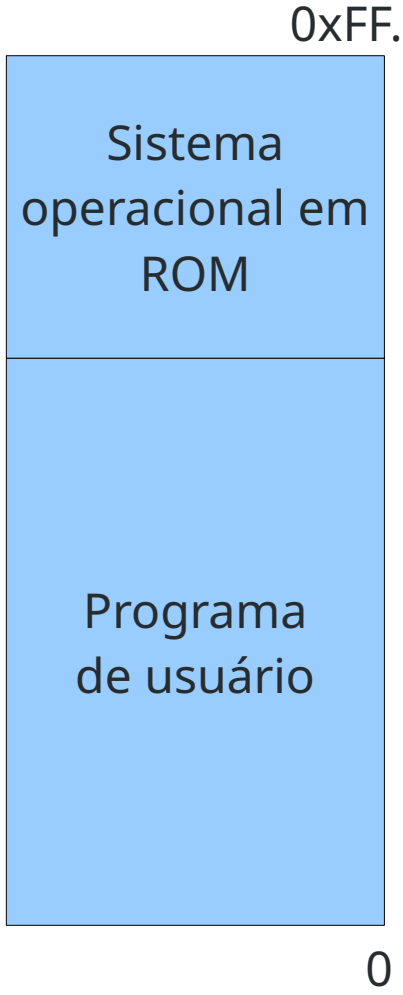
Sem Abstração de Memória

- **Alocação contígua simples**
- Nesse modelo o SO pode estar na parte inferior da memória e o restante está disponível para o usuário
- Erros nos programas podem causar falhas gerais no sistema
- Por isso, hardware pode ser necessário para proteger áreas do SO de programas executando. Para isso, pode usar um registrador para proteção



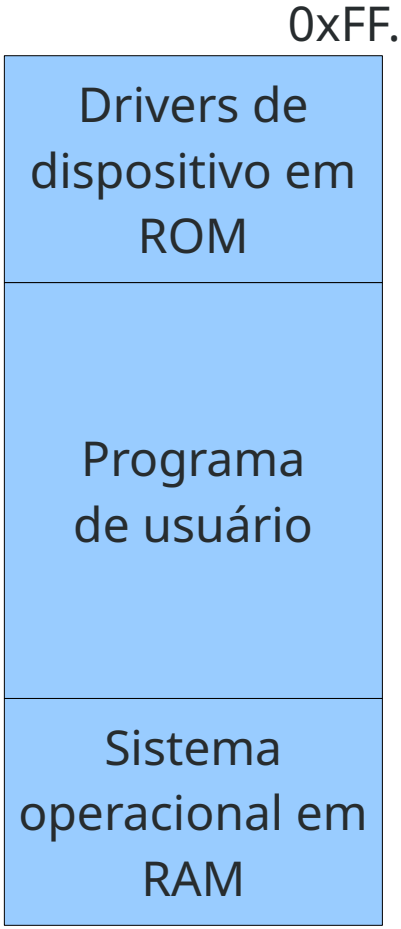
Sem Abstração de Memória

- Uma outra forma de proteção é manter o SO em uma região de memória de apenas leitura



Sem Abstração de Memória

- Por último, temos uma organização onde alguns drivers do dispositivo são armazenadas em memória de leitura
- O sistema operacional é armazenado em memória de leitura e escrita
- Modo utilizado pelos primeiros computadores pessoais com MS-DOS
- A porção ROM é chamada de BIOS

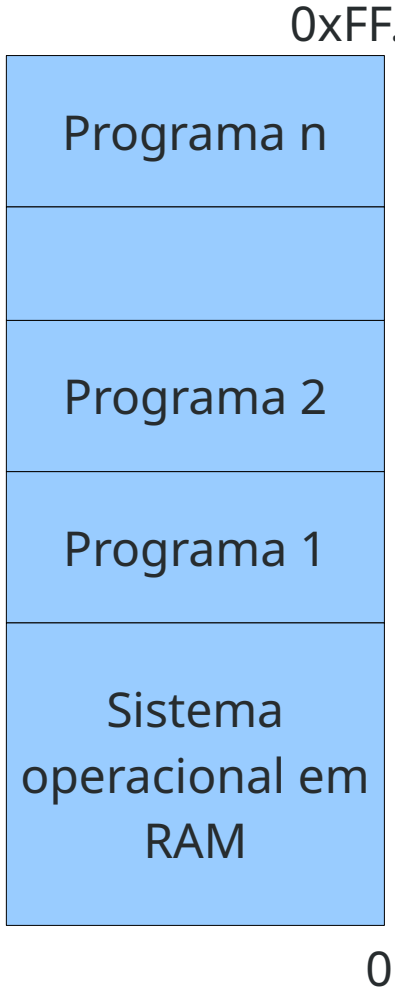


Sem Abstração de Memória

- No entanto, todas essas técnicas não são viáveis para se utilizar múltiplos processos em execução de um programa
- Elas também trazem um grande desperdício computacional: se um programa não utilizar toda a memória RAM, um grande espaço livre é encontrado na estação

Multiprogramação

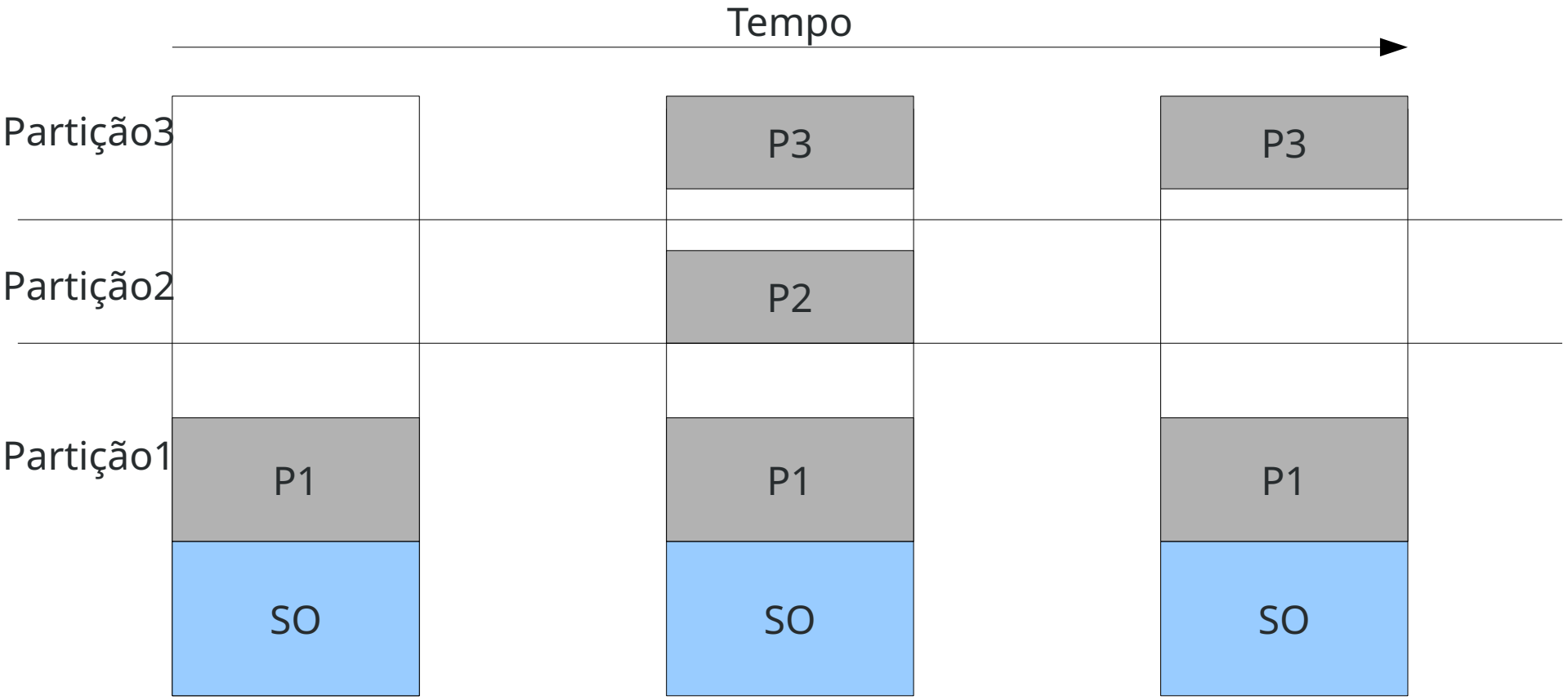
- No entanto, a grande maioria dos SO modernos permitem a execução de múltiplos programas
- Assim, uma região de memória (protegida por hardware) é alocada para o SO e os programas dividem o espaço de usuário
- Essa técnica é conhecida como **Alocação Particionada**



Alocação Particionada

- Nos primeiros computadores multiprogramáveis, a memória era divida em regiões (ou partições) de tamanho fixo. Alterar o tamanho dessas partições exigia o reinicio do sistema operacional
- Cada partição pode ter apenas um programa em execução e um processo sempre roda na mesma partição até terminar
- MFT: Multiprogramming with a fixed number of tasks ou **Alocação Particionada Estática**

Alocação Particionada



Alocação Particionada

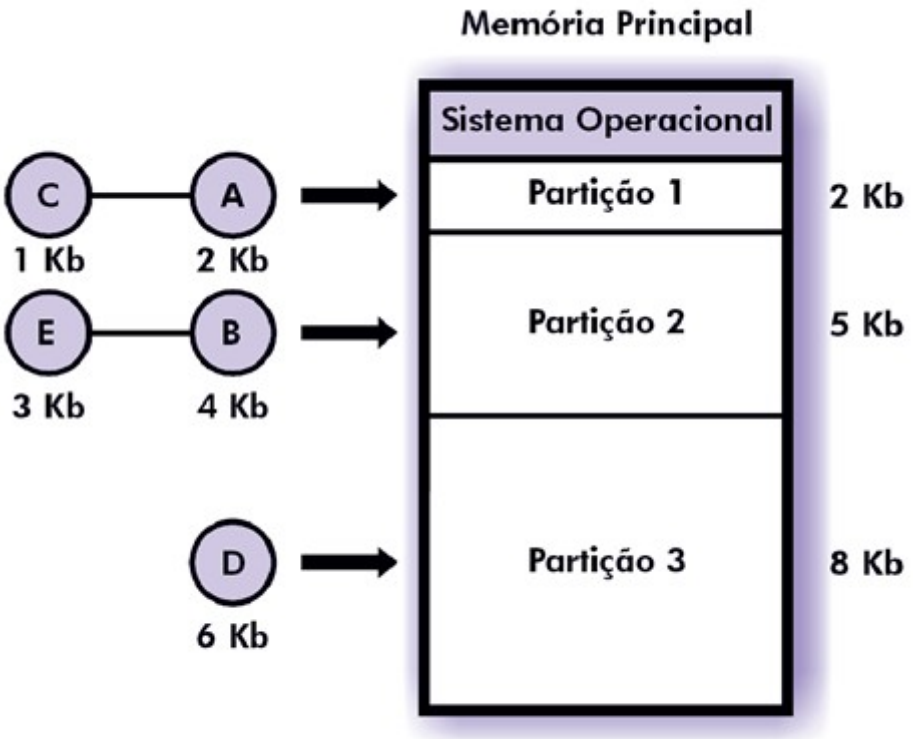


Fig. 9.6 Alocação particionada estática absoluta.

Fonte: Machado e Maia (2017)

Alocação Particionada

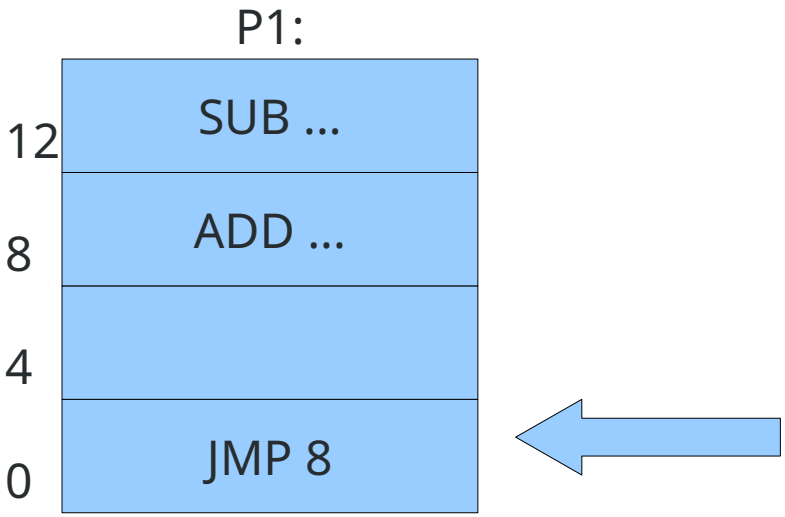
- As principais desvantagens da alocação particionada estática é o desperdício de memória, um processo nem sempre vai utilizar toda memória disponível na partição: **fragmentação interna**
- Além disso, se existem 3 partições de 5 KB, 3 KB e 2 KB e são criados dois processos de 4 KB, apenas um deles vai poder executar, mesmo que exista 5 KB disponíveis em memória

Alocação Particionada

- No início, os computadores podiam usar apenas uma partição, mesmo que outra estivesse disponível
- Isso era por causa das instruções de manipulação da memória e controle de fluxo que usavam endereços de memória absolutos
- Essas funções eram decididas em tempo de compilação e não podiam ser alteradas

Alocação Particionada

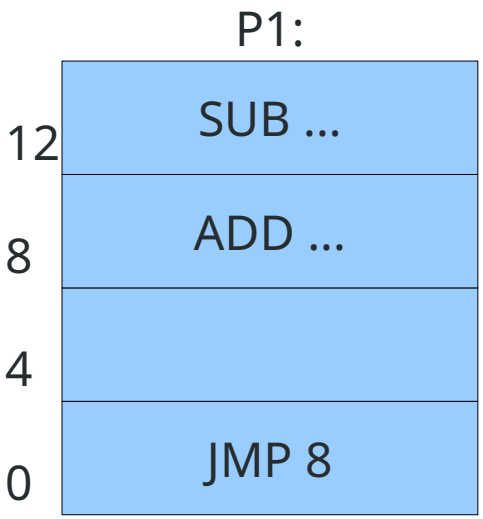
- Imagine dois programa (P1 e P2) que ocupam 1000 bytes em memória cada um.

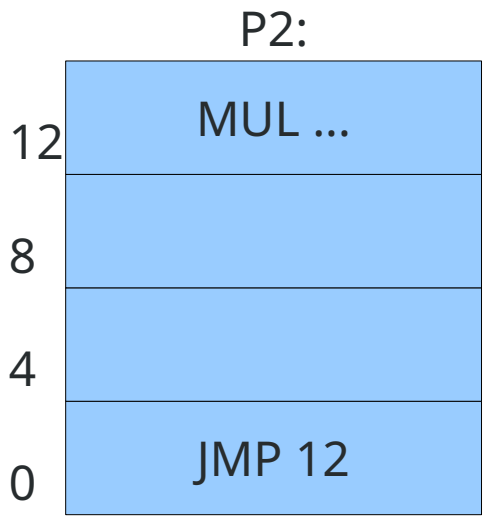


Esta instrução faz com que a próxima instrução a ser executada seja a da posição 8 (ADD)

Alocação Particionada

- Imagine dois programa (P1 e P2) que ocupam 1000 bytes em memória cada um.

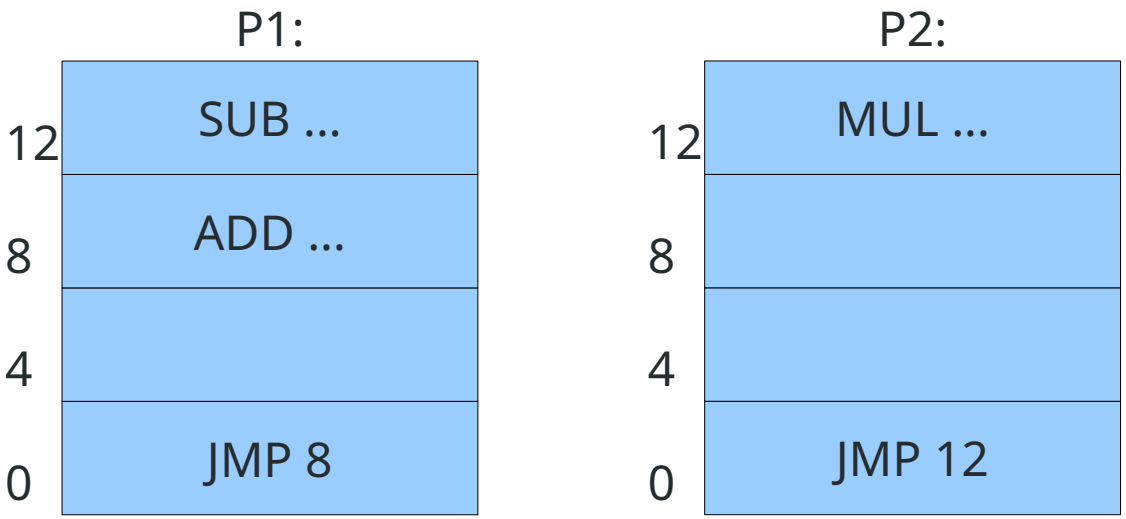




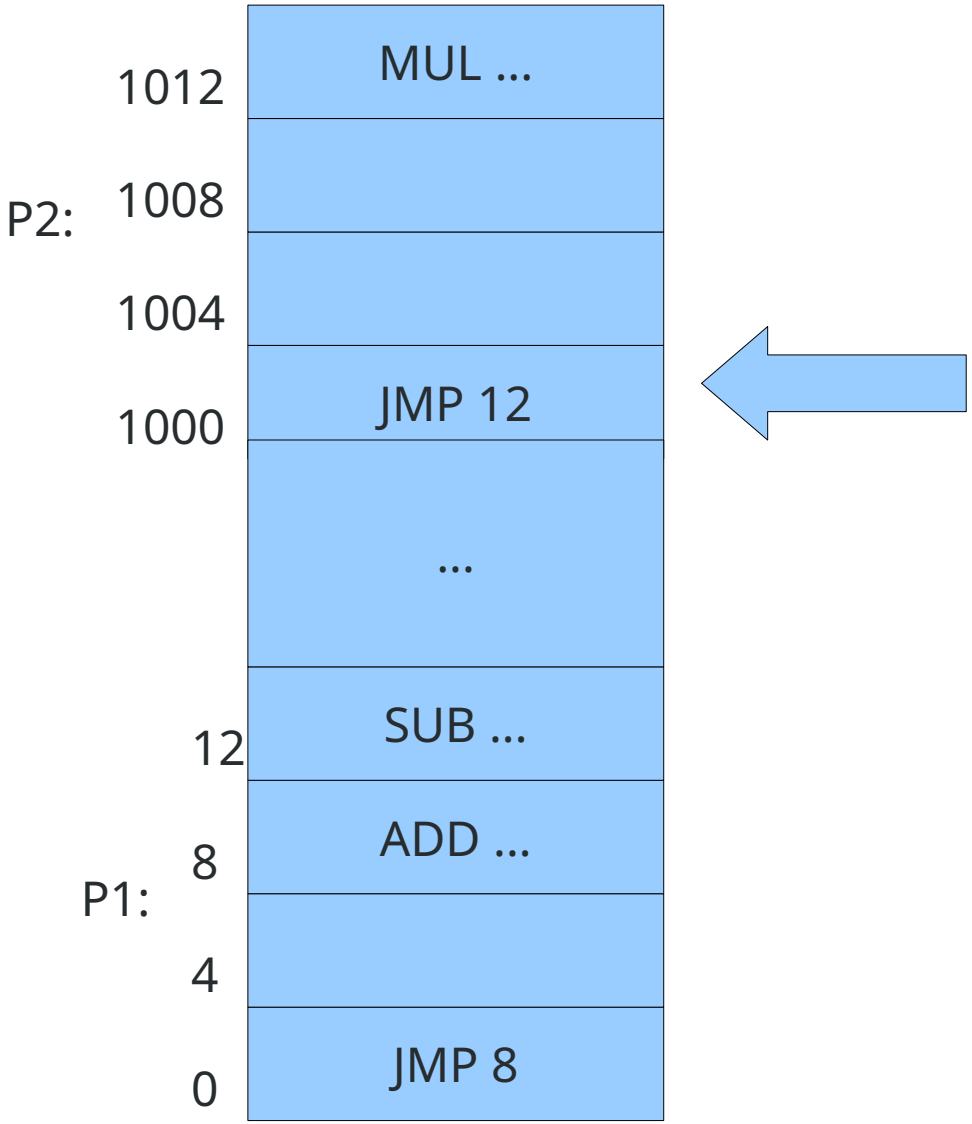
Já o programa 2 decidiu executar instruções da posição 12 (MUL)

Alocação Particionada

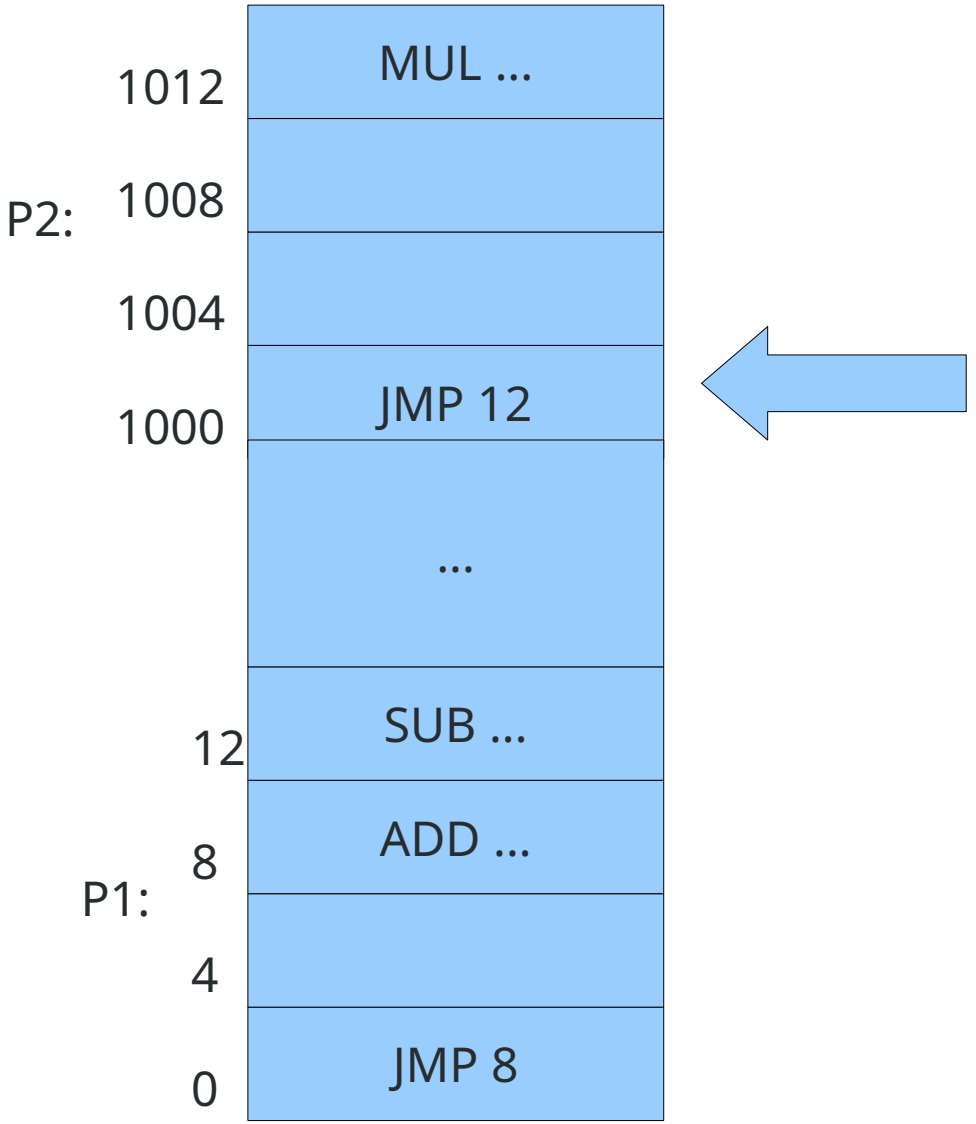
- O que acontece quando os dois programas são carregados na mesma memória? P1 na partição de 0 a 999 e o P2 na 1000 a 1999?



Alocação Particionada



Alocação Particionada



Neste caso P2 irá "pular" para a área de P1 e começar a executar instruções de outro programa

Alocação Particionada

- Por esse motivo, os programas deveriam ser mantidos na mesma partição para executar, mesmo que outra partição estivesse disponível
- **Alocação estática particionada absoluta**

Alocação Particionada

- Outro exemplo: imagine o processo P1 e o processo P2 com as seguintes instruções:
- P1: LOAD R1, 0x2000
- P2: LOAD R1, 0x2100
- Em um sistema com o seguinte particionamento:
- Partição1: 0x1000 – 0x5000
- Partição2: 0x5000 – 0x9000
- O processo P1 e P2 não podem executar simultaneamente

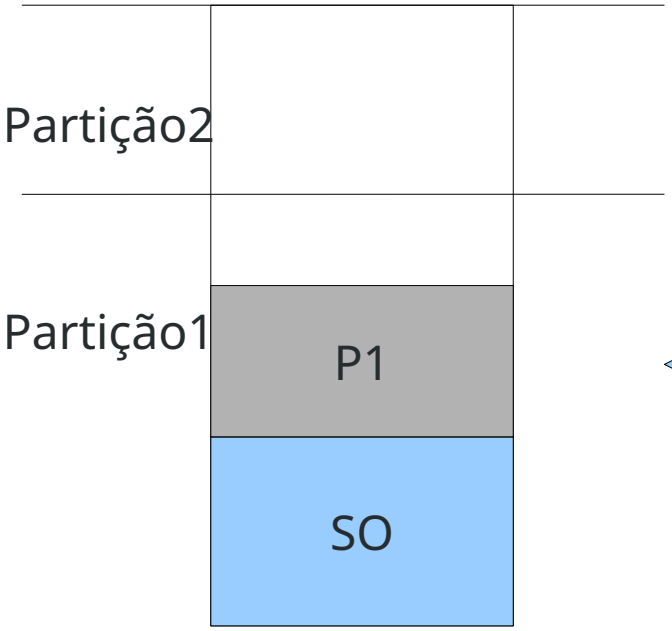
Alocação Particionada

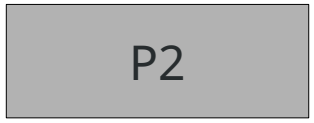
Partição1: 0x1000 – 0x5000

Partição2: 0x5000 – 0x9000

P1: LOAD R1, 0x2000

P2: LOAD R1, 0x2100





P2 utiliza a mesma partição do P1 e por isso não pode executar agora!

Alocação Particionada

- Uma solução temporária para esse problema foi criada no IBM 360 chamada de **realocação estática**
- Quando um programa era carregado no endereço "16384" uma constante era adicionada a todas as funções de manipulação de memória
- Portanto "JMP 12" ao ser carregada para a memória era transformada em "JMP 16396"
- $16396 = 12 + 16384$

Alocação Particionada

- Modificar todas as instruções não é um trabalho trivial, as instruções “JMP 12” deveriam ser alteradas, mas as instruções “MOV R1, 12” significa carregar o valor 12 para o registrador R1 e não deve ser alterada
- Porém, com os avanços dos compiladores, montadores e loaders, essa técnica tornou-se viável e os programas puderam ser executados em múltiplas partições
- **Alocação particionada estática realocável**

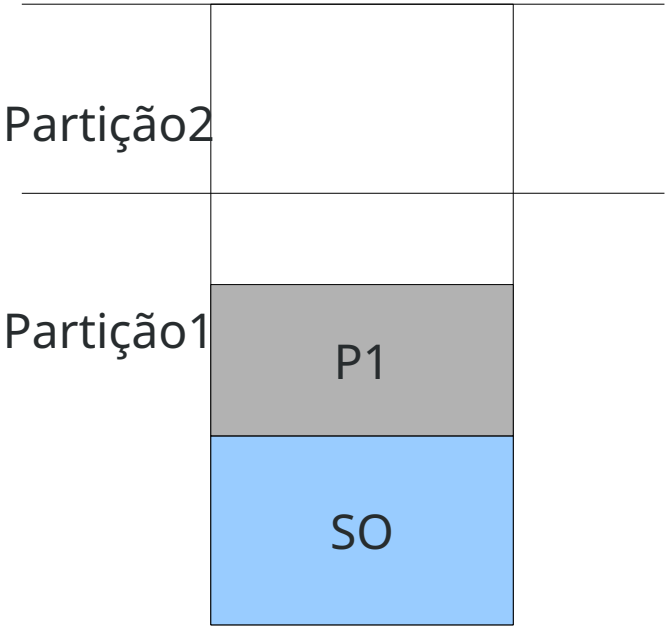
Alocação Particionada

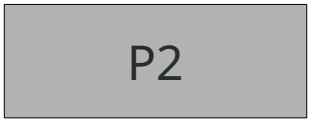
Partição1: 0x1000 – 0x5000

Partição2: 0x5000 – 0x9000

P1: LOAD R1, 0x2000

P2: LOAD R1, 0x2100





Alocação Particionada

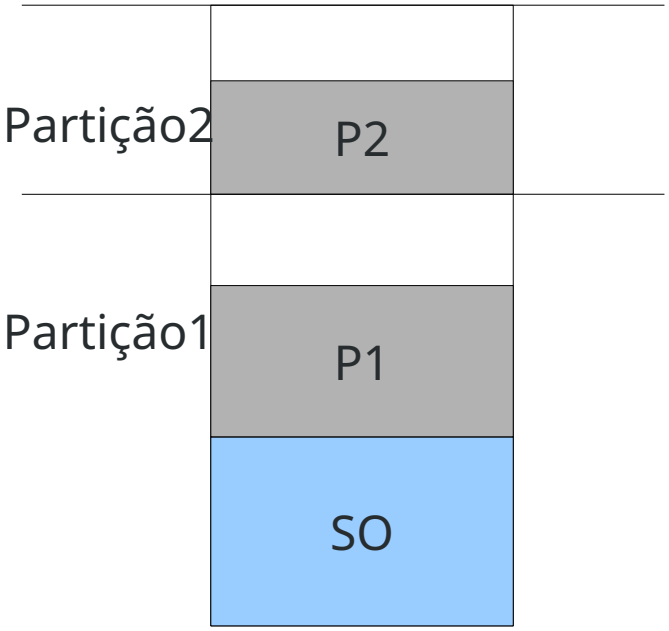
Com a Alocação Particionada Realocável, os endereços acessados pelo programa são alterados durante a criação do processo para a partição que o sistema operacional o alocou:

Partição1: 0x1000 – 0x5000

Partição2: 0x5000 – 0x9000

P1: LOAD R1, 0x2000

P2: LOAD R1, 0x2100



Alterado durante a criação do processo

P1: LOAD R1, 0x3000 (0x2000 + 0x1000)

P2: LOAD R1, 0x7100 (0x2100 + 0x5000)

Alocação Particionada: abstração de memória

- Com isso, soluções de uma abstração de memória surgiu, de forma a tratar a memória como algo relativo dentro do processo, e não um endereço absoluto de memória
- No entanto esse modelo não serve para o problema da proteção contra os outros processos ativos

Abstração de memória: espaço de endereçamento

- Dois problemas devem ser resolvidos na gerência de memória: proteção e realocação
- A realocação estática com modificações de instruções é uma possível solução, mas pode ser um processo lento, veremos outras
- Para solucionar a proteção é criado um novo conceito “espaço de endereçamento”

Abstração de memória: espaço de endereçamento

- Com o espaço de endereçamento, cada processo possui um conjunto de endereços associados a ele e que não podem referenciar outros
- Algumas exceções são feitas para quando os processos desejam compartilhar dados na mesma memória (memória compartilhada)
- Uma das maneiras mais simples de se implementar a proteção e a realocação é com Registrador de base e limite

Registrador de Base e Limite

- Solução simples de **realocação dinâmica**
- Todos os endereços no código binário executável são de 0 a x (onde X é o tamanho do binário)
- A CPU é equipada com dois registradores especiais, base e limite

Registrador de Base e Limite

- Nessa estratégia os programas são sempre carregados em memórias consecutivas
- Quando o processo é carregado em memória, o registrador base armazena o valor do endereço físico de onde o programa começa
- O registrador limite armazena o comprimento do programa

Registrador de Base e Limite

- Toda vez que um processo referencia a memória, seja para buscar uma instrução, ler ou escrever uma palavra de dados, o hardware da CPU automaticamente adiciona o valor base ao endereço gerado pelo processo antes de enviá-lo ao barramento de memória
- Da mesma forma, é verificado se esse valor extrapola os limites definidos para o programa (base + limite), caso sim uma falta é gerada e o programa é abortado

Registrador de Base e Limite

- Uma desvantagem da realocação usando registrador base e limite é a necessidade de uma adição e uma comparação em cada referência de memória
- Comparações são operações rápidas no hardware, mas as somas podem ser lentas, exceto quando um circuito de adição especiais seja usado

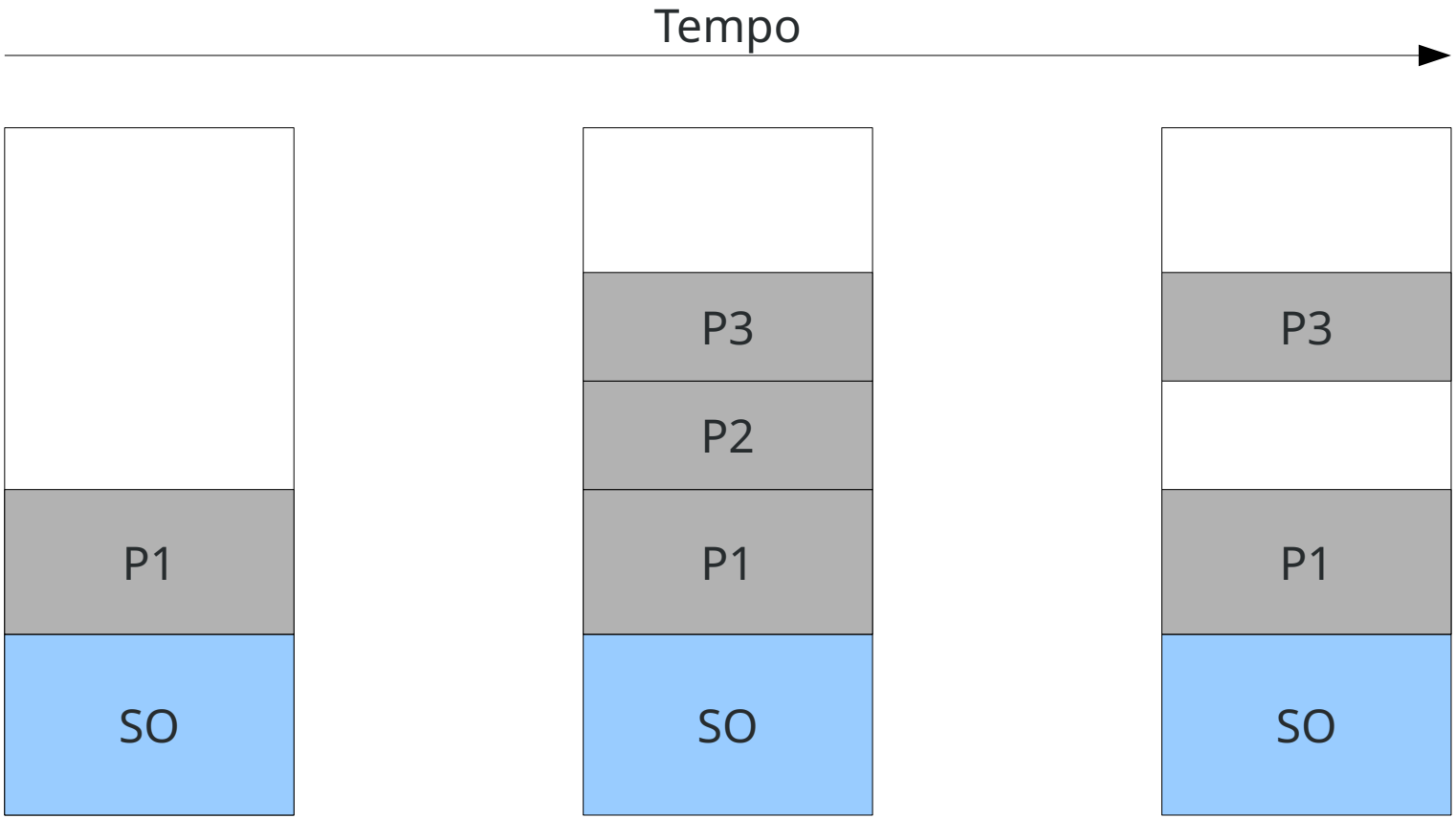
Alocação Particionada Dinâmica

- O uso de registradores base e limite permite o esquema de partições de tamanho variável
- Desta forma, o sistema não aloca previamente espaços de memória, mas a medida que os programas são criados, uma área de memória com o tamanho necessário para o programa é reservada a eles

Alocação Particionada Dinâmica

- Esquema conhecido como Multiprogramação de tamanho variável, MVT, Multiprogramming with a variable number of tasks
- Resolve o problema de desperdício de memória do MFT
- Memória é dividida em um conjunto de partições de tamanho variável
- O número, tamanho e localização dos processos em memória varia com o tempo

Alocação Particionada Dinâmica

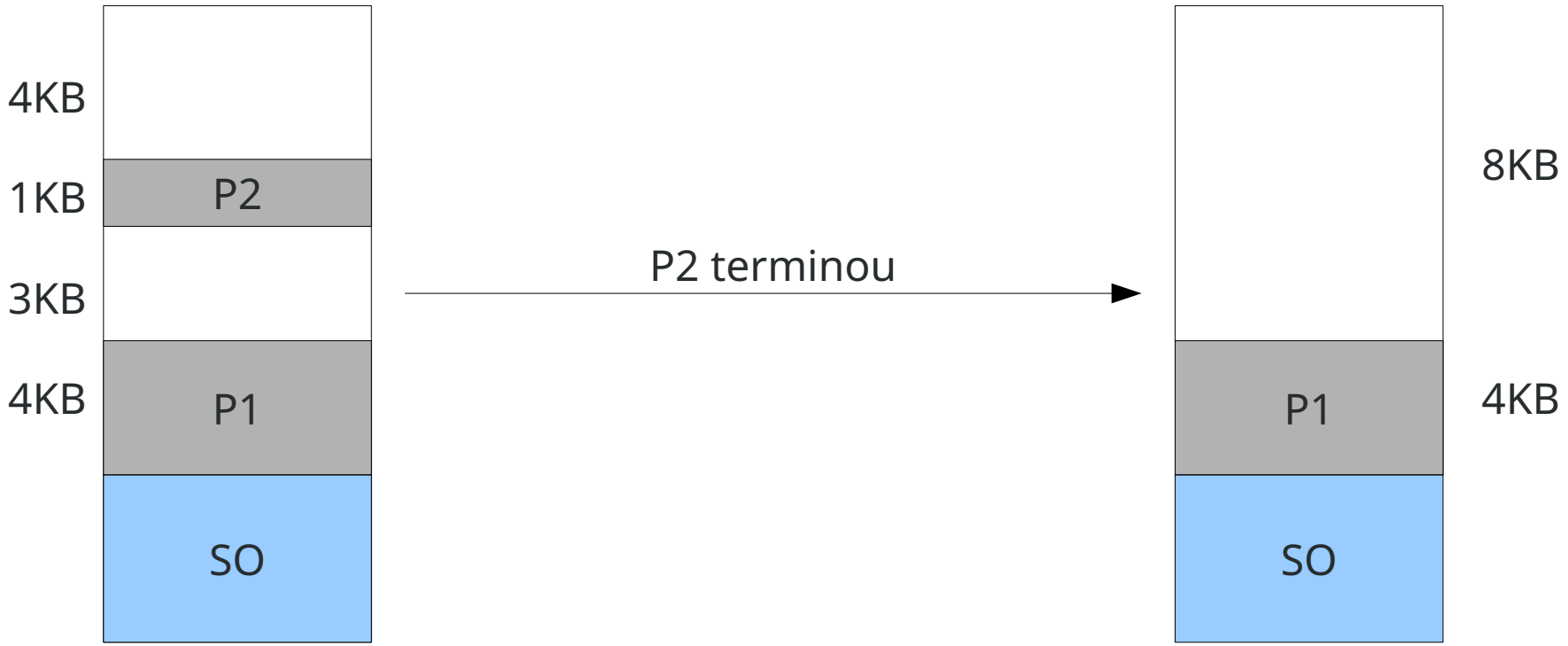


Alocação Particionada Dinâmica

- Apesar de solucionar o problema da fragmentação interna, esses “buracos” que surgem durante a execução são um novo problema: **fragmentação externa**
- Existem algumas estratégias para minimizar os efeitos dessa fragmentação

Alocação Particionada Dinâmica

- 1 - Agrupamento de áreas adjacentes



Compactação de Memória

- A segunda estratégia é a compactação de memória, que pode ser realizada quando a fragmentação de memória atingiu níveis intoleráveis
- Os processos são movidos dentro da memória principal para novas regiões
- Causa um congelamento de todos os processos: nada além do gerenciamento de memória se executa

Compactação de Memória

Compactação de Memória

- A compactação de memória é um processo lento e de grande impacto na performance
- Todos os processos em execução precisam ser parados temporariamente, para que as cópias de memória possam ser realizadas
- Copiar grandes regiões de memória não é rápido

Algoritmo de Alocação de Espaço

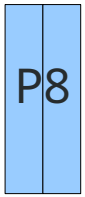
- No entanto, sendo função do SO gerenciar os recursos computacionais, cabe a ele decidir onde encaixar os processos na memória
- Como alocar memória para um processo sendo criado?
- Existem 3 estratégias:
  - First fit: aloca no primeiro espaço grande o suficiente;
  - Best fit: aloca no menor espaço grande o suficiente;
  - Worst fit: aloca no maior espaço grande o suficiente.

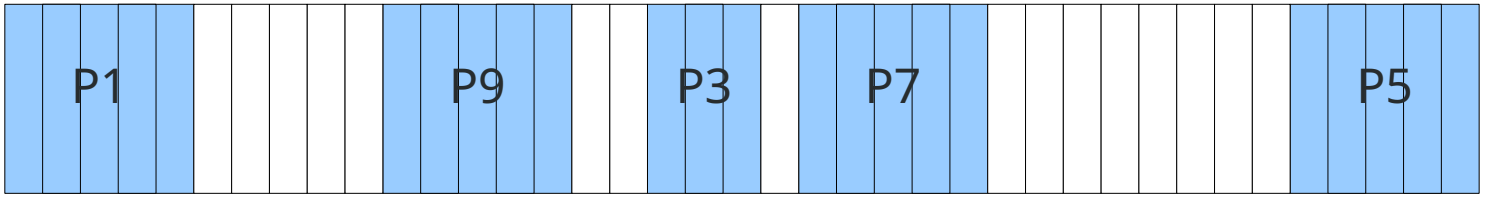
Algoritmos de Alocação de Espaço

- O primeiro algoritmo é o **first fit**(primeiro encaixe)
- O gerenciador de memória verifica a lista de segmentos livres até encontrar um que seja grande o suficiente
- O espaço livre é dividido em duas partes: uma para o processo alocado e outra para memória não utilizado (assumindo que não ocorreu um improvável encaixe exato)

Algoritmos de Alocação de Espaço

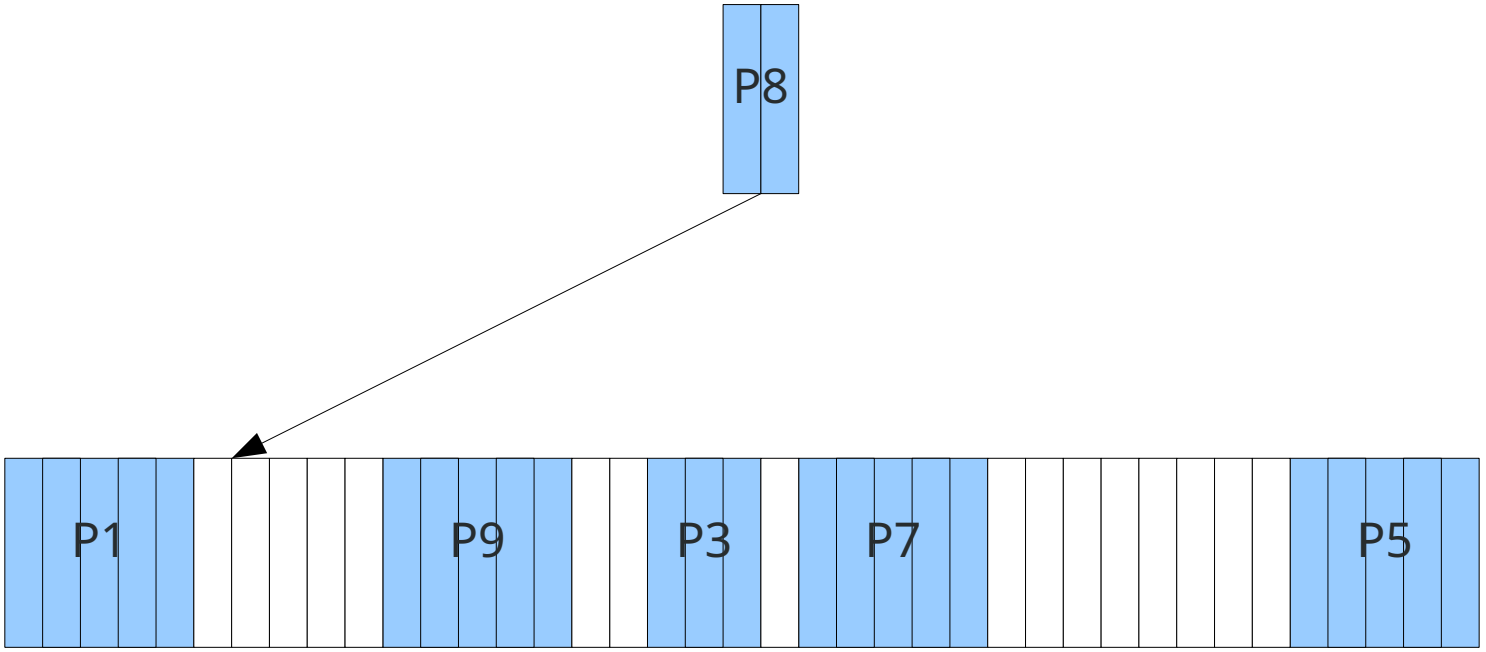
- First fit: onde colocar o processo P8 que precisa de 2 unidades de memória?





Algoritmos de Alocação de Espaço

- First fit: onde colocar o processo P8 que precisa de 2 unidades de memória?

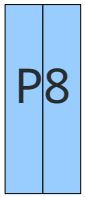


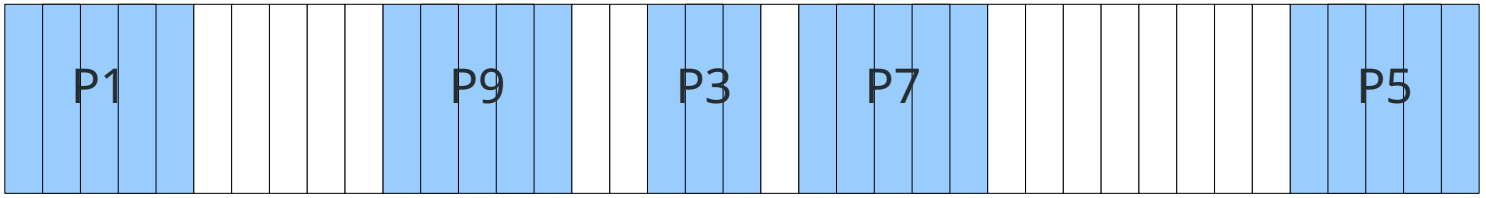
Algoritmos de Alocação de Espaço

- Outra solução é o best fit
- Esse algoritmo percorre **toda a lista**, do início ao fim, procurando encontrar um espaço que seja adequado, ao invés de usar um espaço grande demais
- Esse algoritmo busca diminuir a fragmentação de memória na esperança de encontrar um lugar ótimo para colocar o processo

Algoritmos de Alocação de Espaço

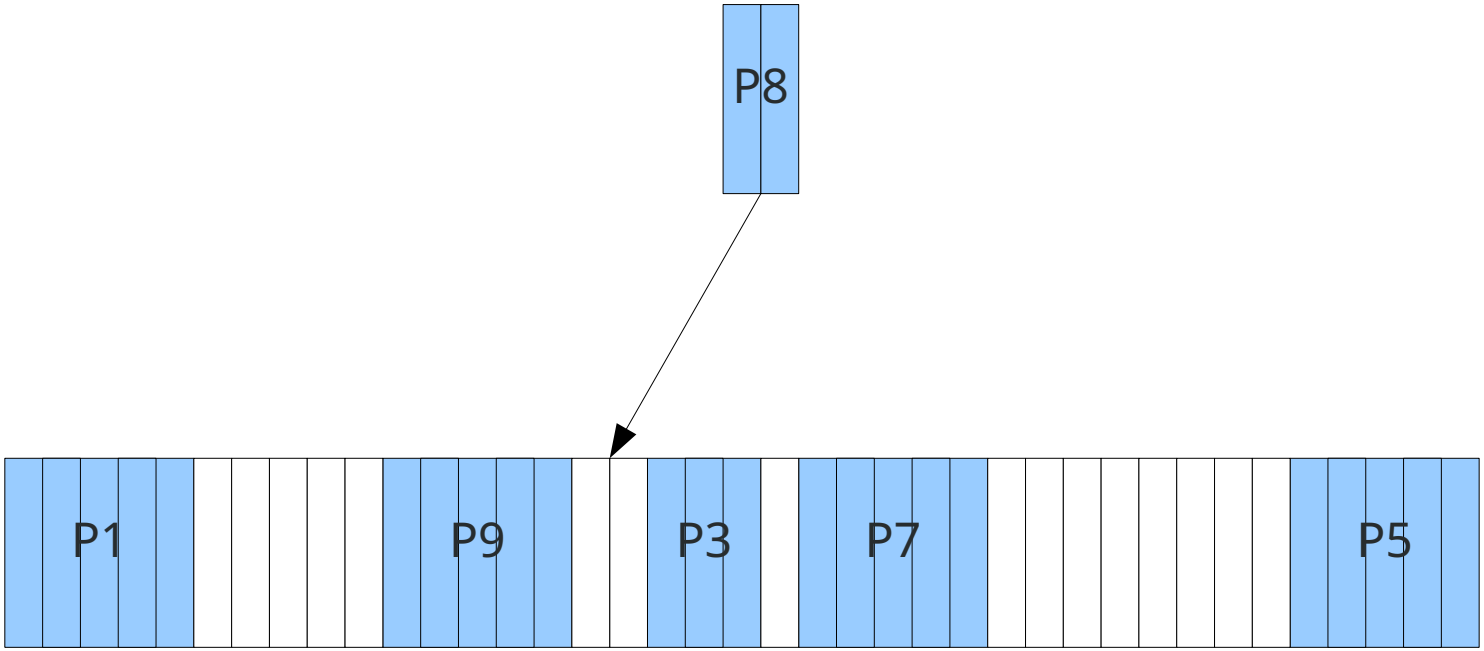
- Best fit: onde colocar o processo P8 que precisa de 2 unidades de memória?





Algoritmos de Alocação de Espaço

- Best fit: onde colocar o processo P8 que precisa de 2 unidades de memória?

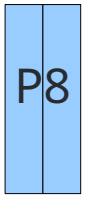


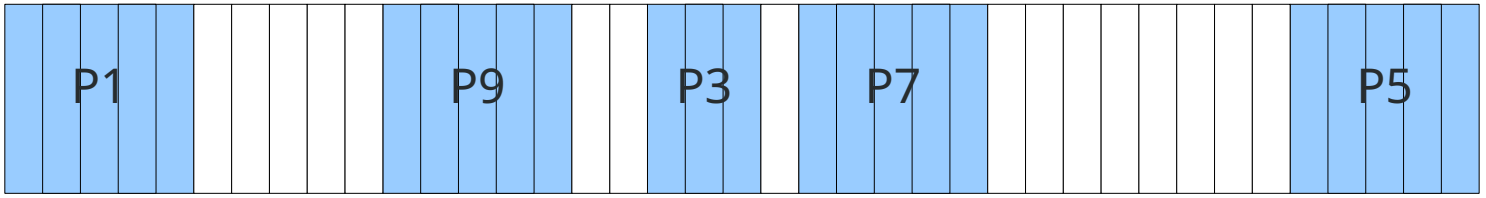
Algoritmos de Alocação de Espaço

- O best fit tende a criar espaços minúsculos inúteis na memória
- Para tentar contornar esse problema, a estratégia **worst fit** tenta colocar o espaço na área onde tem mais espaço livre
- Dessa forma, ele acredita que o espaço restante tem maior chance de ser útil

Algoritmos de Alocação de Espaço

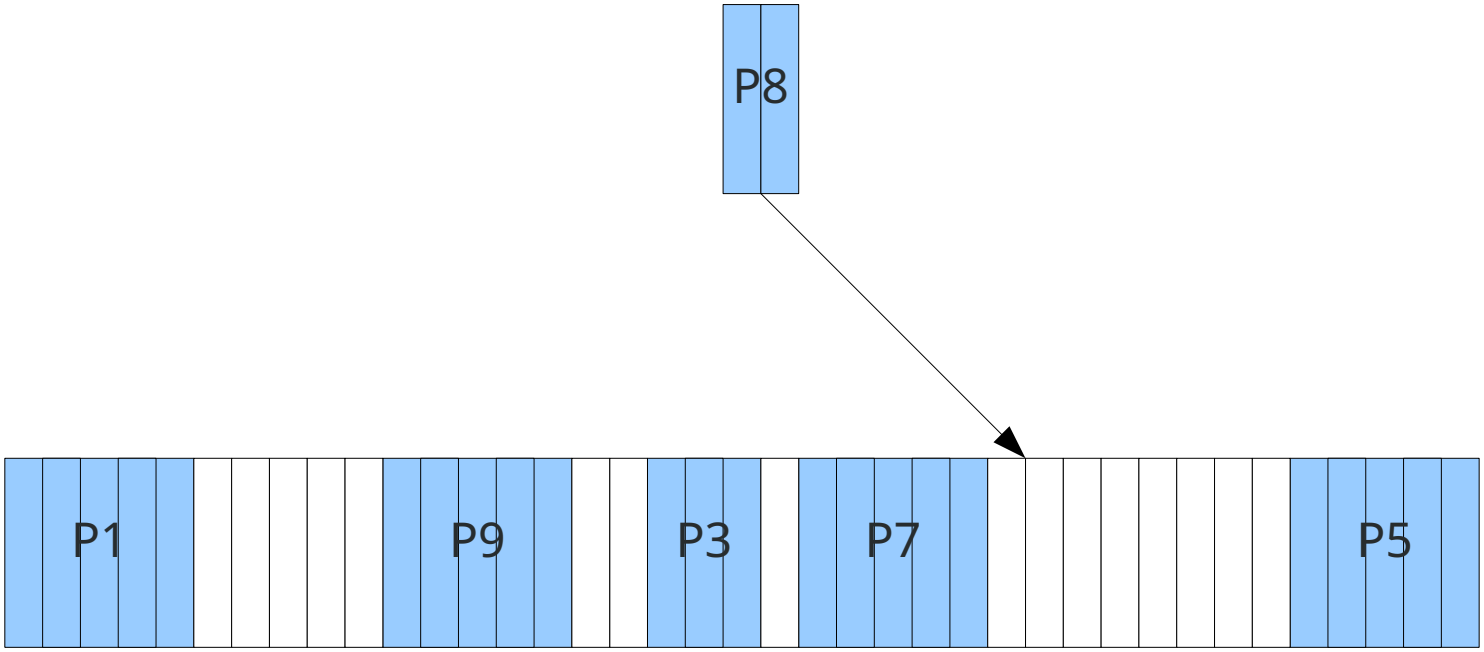
- Worst fit: onde colocar o processo P8 que precisa de 2 unidades de memória?





Algoritmos de Alocação de Espaço

- Worst fit: onde colocar o processo P8 que precisa de 2 unidades de memória?



Algoritmos de Alocação de Espaço

- Qual desses algoritmos apresenta o menor tempo de execução?
- Qual desses algoritmos apresenta o melhor uso de memória?

Algoritmos de Alocação de Espaço

- O first fit termina rapidamente por não precisar analisar toda a lista de memória livre
- Comparando o fist fit e o best fist com simulações, foi mostrado que o first fit com espaços livres maiores na média
- As simulações mostraram que o worst fit não é uma boa estratégia na média
- Por isso, o first fit é frequentemente utilizado

Gerenciando Memória Livre

- Quando um processo é iniciado o sistema reserva uma quantidade de memória para esse processo
- Quando um processo possui um tamanho fixo, é preciso alocar uma quantidade de memória exatamente do tamanho do processo
- No entanto, a maioria dos sistemas permite que o tamanho da área de dados cresça durante a execução, sendo então necessário alocar um espaço de memória maior que o tamanho inicial

Gerenciando Memória Livre

- Em todos os casos, quando a alocação é um processo dinâmico, é necessário manter uma forma de organizar e rastrear o uso de memória
  - Mapa de bits;
  - Lista encadeada.

Controle de Alocação de Memória Mapa de Bits

- Memória divida em em unidade de alocações, tipicamente 1 bytes
- Cada unidade de alocação corresponde a um bit no mapa, sendo 0 se livre e 1 se ocupada

Mapa de bits associado:
1111100000111110

Mapa de bits

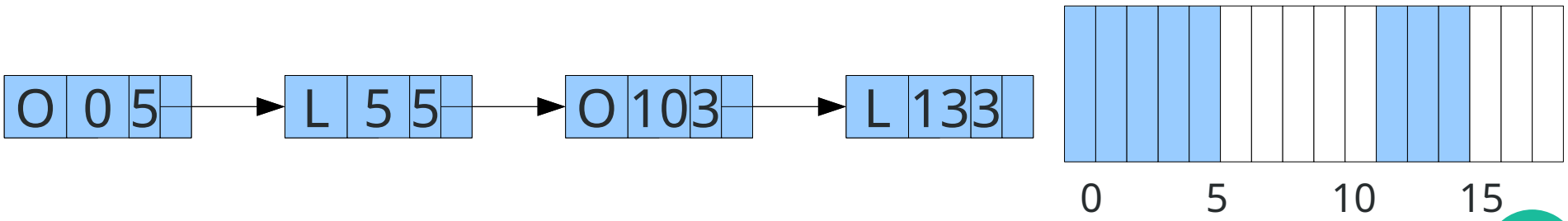
- O tamanho da unidade de alocação é importante
- No mínimo, temos um bit representando o tamanho da palavra em memória (ex: 1 byte)
- Porém é possível aumentar o tamanho dessa representação: 1 bit irá representar 2 bytes
- Essa técnica reduz o tamanho do mapa de bits
- No entanto, ela causa um desperdício de memória quando o tamanho requisitado não é múltiplo da unidade de alocação do mapa de bits

Mapa de bits

- Unidade de alocação grande: desperdício no uso de memória
- Unidade de alocação pequena: mapa de bits grande
- O mapa de bits é uma forma simples de controlar as palavras no entanto ele é muito lento para procurar por regiões vazias dentro do mapa de bits
- Isto torna seu uso prático inviável

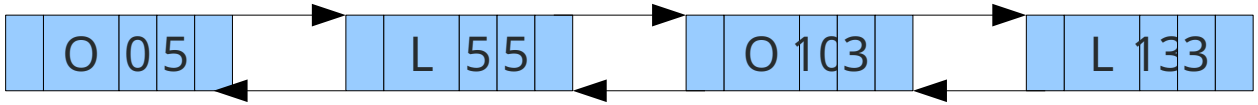
Controle de Alocação de Memória Lista Encadeada

- Outro esquema de gerenciar a memória é uma representação de uma lista encadeada
- Cada membro dessa lista contém o tipo de segmento (livre ou ocupado), o endereço de início, o tamanho do segmento e um ponteiro para o próximo elemento



Controle de Alocação de Memória Lista Encadeada

- A lista encadeada possui a vantagem de ser muito boa para atualizar, ao liberar um espaço, você deve fundir a lista com seus vizinhos
- Para isso, uma variação comum é o uso de lista duplamente encadeadas que permitem operações de obter os dois vizinhos, e não apenas o próximo



Swapping

- Muita vezes um programa pode não ser executado devido à falta de uma partição livre
- No entanto, existem vários processos que ocupam memória mas não são utilizados (daemons que vem pré-instalados no sistema)
- Manter todos esses processos em memória pode ser desnecessário e inviável
- A estratégia mais simples para lidar com esse problema é a swapping

Swapping

- No swapping o SO escolhe um processo residente e transfere esse processo para a memória secundária (swap out)
- Posteriormente, se o processo despertar ou for escolhido pelo escalonamento, ele é movido do disco para a memória principal como se nada tivesse acontecido

Swapping

- Para isso, existe uma área reservada no disco, chamada de área de swap
- No Linux, ao se instalar o SO gera-se explicitamente uma partição de swap
- No Windows, é um arquivo dentro de uma partição conhecida pelo SO e pode ser configurada

Swapping

- Quando um processo é movido do disco para memória (swap in) ele pode ser alocado em uma região diferente da região previamente alocada a ele
- Neste caso, o software pode precisar alterar todas as instruções dela (modificação de instruções) e pode tornar inviável esse processo
- Nesse caso, registradores base e limite funcionam muito bem!

Overlay

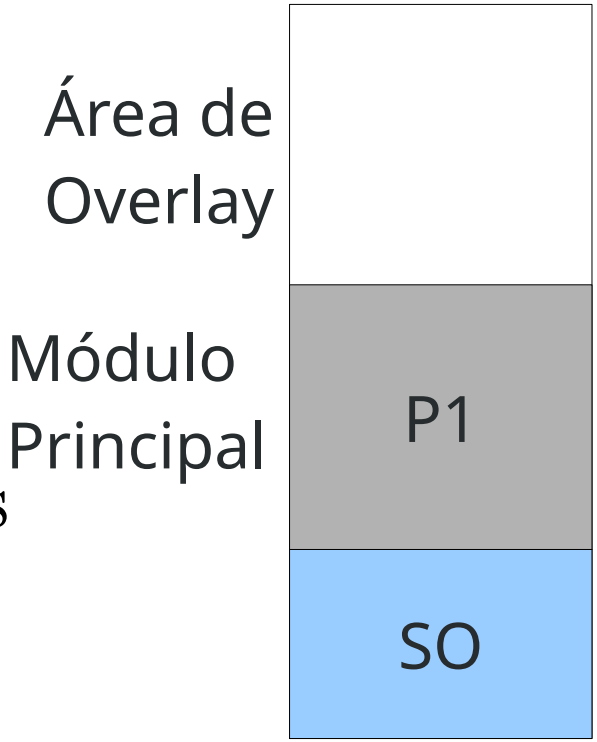
- Uma outra estratégia possível é a Overlay
- Nela o programador divide seu programa em módulos, informando quais módulos devem estar ativos simultaneamente
- O SO é responsável por carregar e liberar os módulos da memória
- Desta forma, módulos independentes não precisam ocupar a memória e o programa não precisa ser todo carregado em memória

Overlay

- Imagine que você tem um programa de computador dividido em três módulos:
- Módulo principal, precisa de 3 KB
- Módulo de cadastro de clientes, precisa de 3 KB
- Módulo de visualização de clientes, precisa de 3 KB
- No entanto, o módulo de cadastro e visualização NUNCA precisam ficar ativos simultaneamente

Overlay

- Nesta técnica, a partição o módulo principal ocupa um pedaço da memória.
- A área de Overlay é utilizada por um dos outros dois módulos, já que eles não precisam estar ativos ao mesmo tempo
- Para isso, o programador deve dividir o programa em módulos



Referências

- Capítulo 3 – TANENBAUM, A. S. *Sistemas Operacionais Modernos*. 4ª ed. Prentice Hall, 2016.
- Capítulo 9 – MACHADO, F. B.; MAIA, L. P. *Arquitetura de Sistemas Operacionais*. 5ª ed. Rio de Janeiro: LTC, 2013.