

Prof. Ricardo Inácio Álvares e Silva

Gerenciamento de Memória

Sistemas Operacionais

Rápida Revisão

- ❖ Sistemas operacionais **multiprogramados** permitem múltiplos programas estarem em execução ao mesmo tempo
- ❖ Cada instância de um programa em execução é um **processo**
- ❖ Programas em execução são aqueles que tem **atributos** de execução salvos no **PCB**, para que possam sair e entrar em contexto sem alterar os resultados da execução
- ❖ Tipicamente os processos ficam alocados na **memória principal** do computador

Escopo da Aula

- ❖ Gerenciamento de Memória
 - ❖ Modelo de memória
 - ❖ Problemática
- ❖ Espaços de Endereçamento
 - ❖ Registradores de endereços base e limite
- ❖ Swapping
 - ❖ Gerenciamento de espaços livres
 - ❖ Algoritmos de inclusão

Organização de Memórias

- ❖ A memória ideal para computadores teriam as seguintes características:
 - ❖ Desempenho sempre na velocidade do processador principal
 - ❖ Tamanho tendendo ao infinito
 - ❖ Preço baixo
- ❖ Com a atual tecnologia, só é possível atender a uma dessas características por vez
- ❖ Resultado disso é a hierarquia de memórias
 - ❖ Registradores, Caches, Memória Principal, Discos, Fitas magnéticas

Organização em Monoprogramados

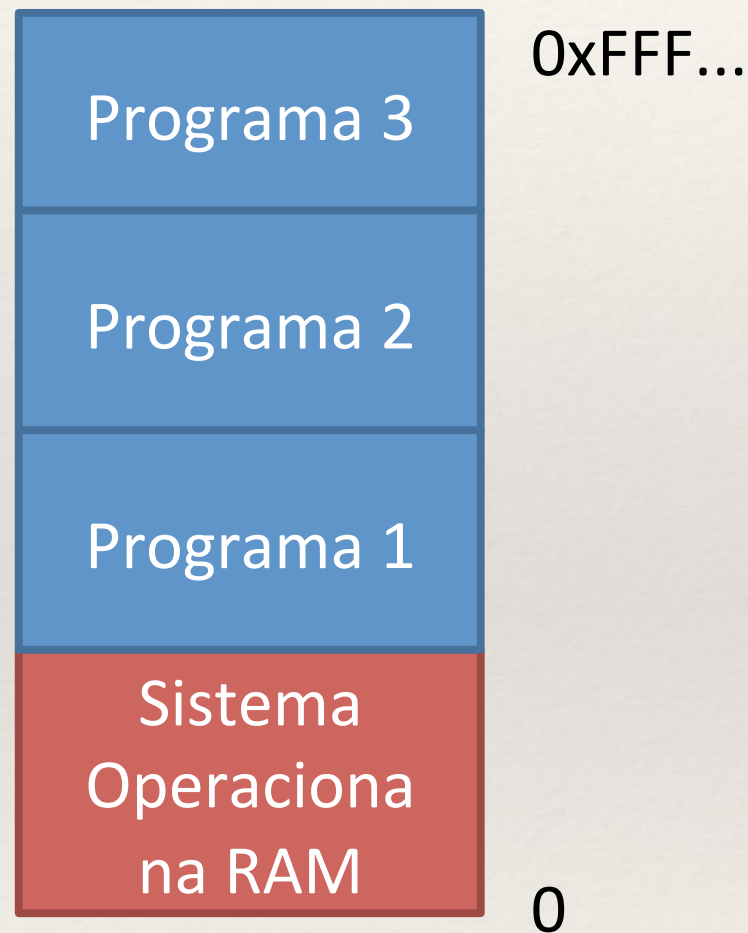


- ❖ Só há um programa por vez na memória
- ❖ Possui acesso total ao espaço de memória

- ❖ Acesso aos dados:

```
lw $s1, 1000($t1)
```

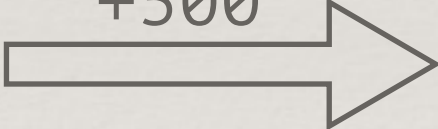
Organização em Multiprogramados



- ❖ Como colocar múltiplos processos na memória?
- ❖ Posicionar um programa após o outro na memória
- ❖ Porém, quando um programa é compilado, ele possui instruções da forma:
$$j \quad 1000$$
- ❖ Se o programa 3 esperasse estar no espaço do 1 no momento de compilação, o que ocorreria?

Troca de endereços

- ❖ Possível solução: modificar todas as instruções com acesso a memória quando o processo for colocado em execução
- ❖ Suponha um programa compilado para iniciar na posição 0 da memória, mas é posicionado a partir da 500 pelo SO:

j 300		j 800
addiu \$t2, \$zero, 800		addiu \$t2, \$zero, 1300
lw \$t1, 0(\$t2)		lw \$t1, 0(\$t2)

- ❖ O sistema operacional modifica todos os acessos diretos à memória de acordo com a posição inicial.

- ❖ Porém, todo processador possui instruções de acesso indireto:

```
addiu $t1,$zero,24350
...
addiu $t1, $t1, 200
...
jr $t1
```

- ❖ O que acontece no código acima:
 - ❖ Carrega o número 24350 em *\$t1*
 - ❖ Em algum momento adiante, soma 200 ao valor anterior de *\$t1*,
 - ❖ Ordena um desvio para a instrução indicada por esse registrador
- ❖ É impossível saber quais valores em registradores serão utilizados em saltos ou desvios indiretos
 - ❖ Requer indicação explícita no código pelo programador
 - ❖ Significa aumento da complexidade de programação

Falta de Segurança

- ❖ Dificuldade no ajuste de endereços não é o único problema
- ❖ Neste modelo simplificado
 - ❖ processos têm acesso direto a toda a memória real
 - ❖ qualquer processo pode modificar toda a memória
- ❖ Um processo pode violar dados de outros
- ❖ Um processo pode modificar dados do próprio sistema operacional!

Gerenciador de Memória

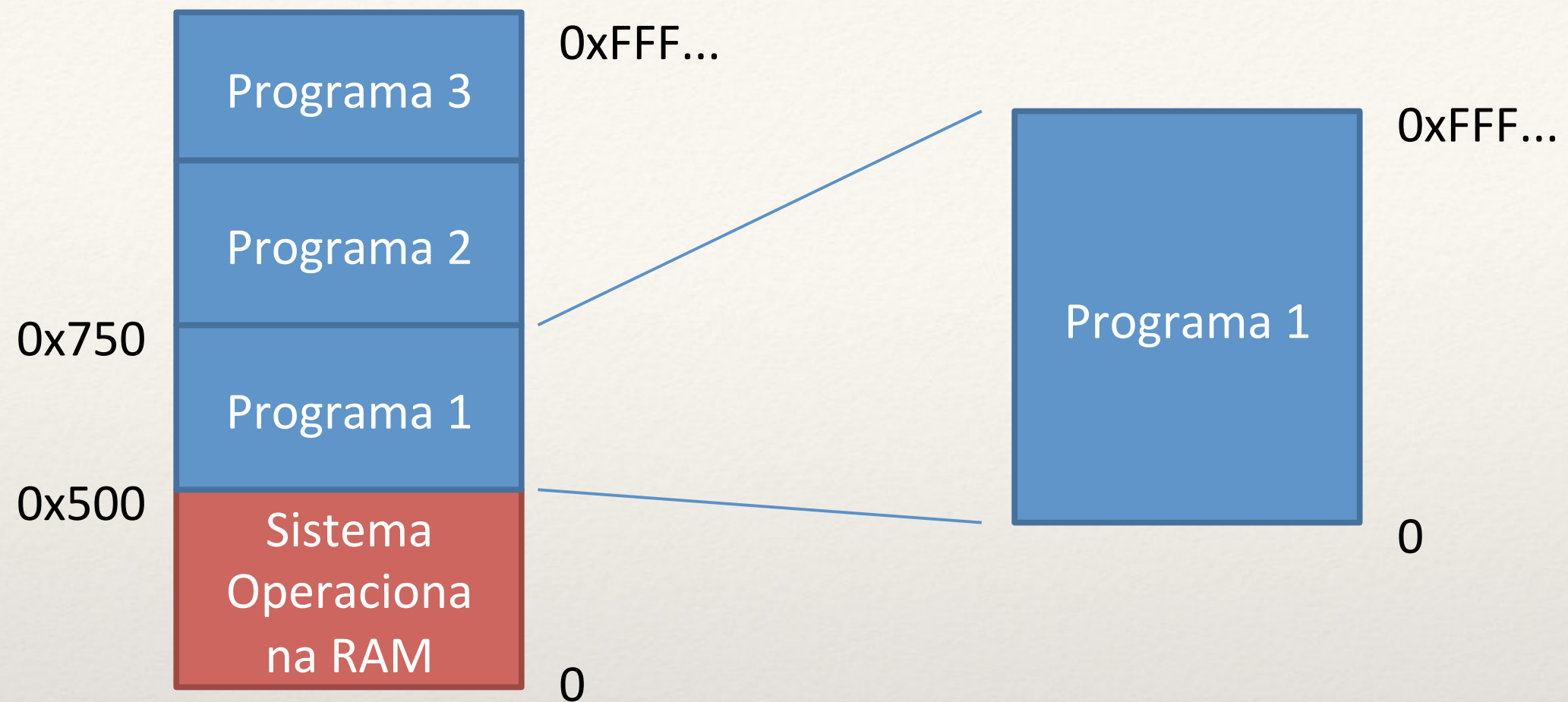
- ❖ Como a memória principal possui limites de tamanho, é necessário gerenciá-la quanto a organização e de processos e seus acessos
- ❖ Por este motivo, sistemas operacionais possuem gerenciadores de memória
- ❖ Questões atribuídas ao **gerente de memória**:
 - ❖ Como impedir que um processo acesse dados de outro processo?
 - ❖ Como alocar memória para um novo processo?
 - ❖ Onde alocar a memória?
 - ❖ Como proceder quando a memória se esgotar?

Abstração da Memória

- ❖ Para resolver os problemas indicados anteriormente, é necessário fazer **abstração** do espaço de memória
- ❖ Assim como todo processo acha que possui um processador só para ele, a ideia é dar a ele a noção de que possui acesso total e irrestrito à memória
- ❖ Dessa forma, todo processo é compilado normalmente, como se fosse sempre posicionado a partir da posição 0 da memória
- ❖ Todo acesso à memória por parte do processo é traduzido em espaço real
- ❖ A tradução de endereços só é possível através de suporte direto do hardware

Registradores Base e Limite

- ❖ A implementação mais simples prevê o uso de dois registradores:
 - ❖ **Base:** indica o endereço real de posicionamento do processo
 - ❖ **Limite:** indica o endereço real limite do processo
- ❖ Todo acesso à memória é adicionado ao valor indicado por base
- ❖ Se esse valor superar o marcado em limite, o hardware indica uma violação de memória e devolve o controle ao sistema operacional

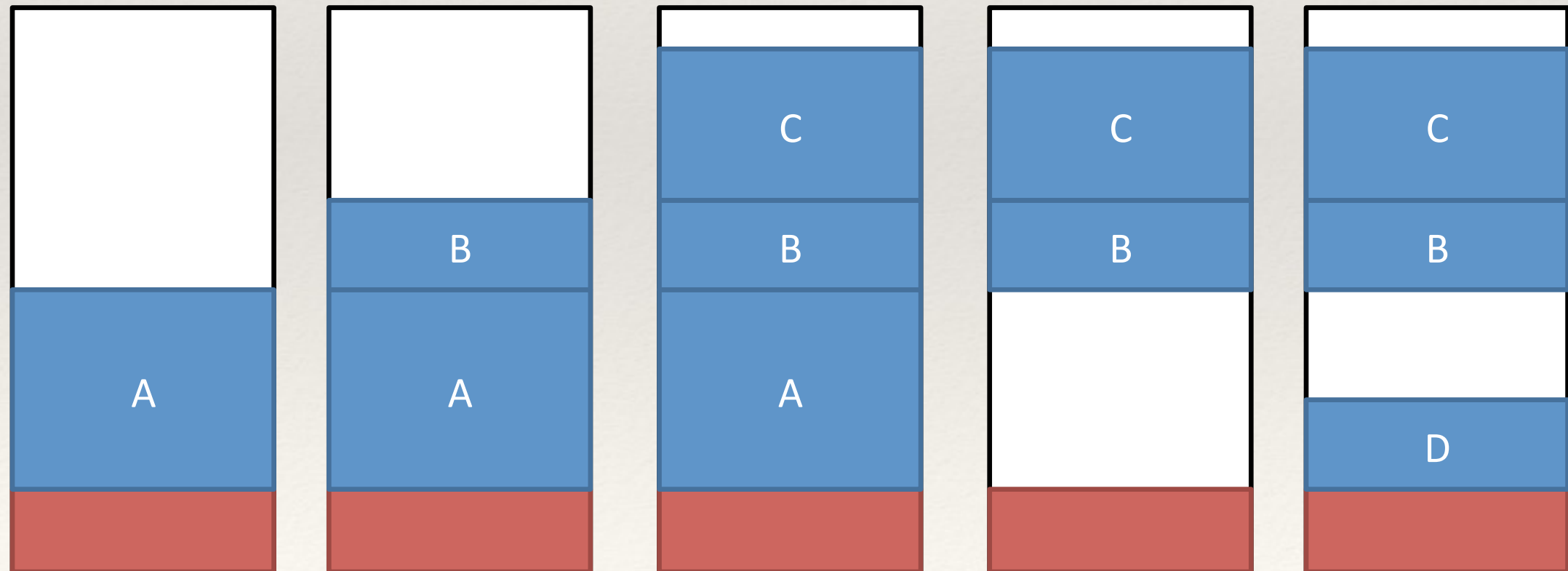


- ❖ Qual o valor do registrador base para o Programa 1?
- ❖ Qual o valor do registrador limite?
- ❖ Se o programa 1 acessar um dado da posição 50, qual a posição real do dado na memória?
- ❖ E um dado na posição 300? O que ocorre?

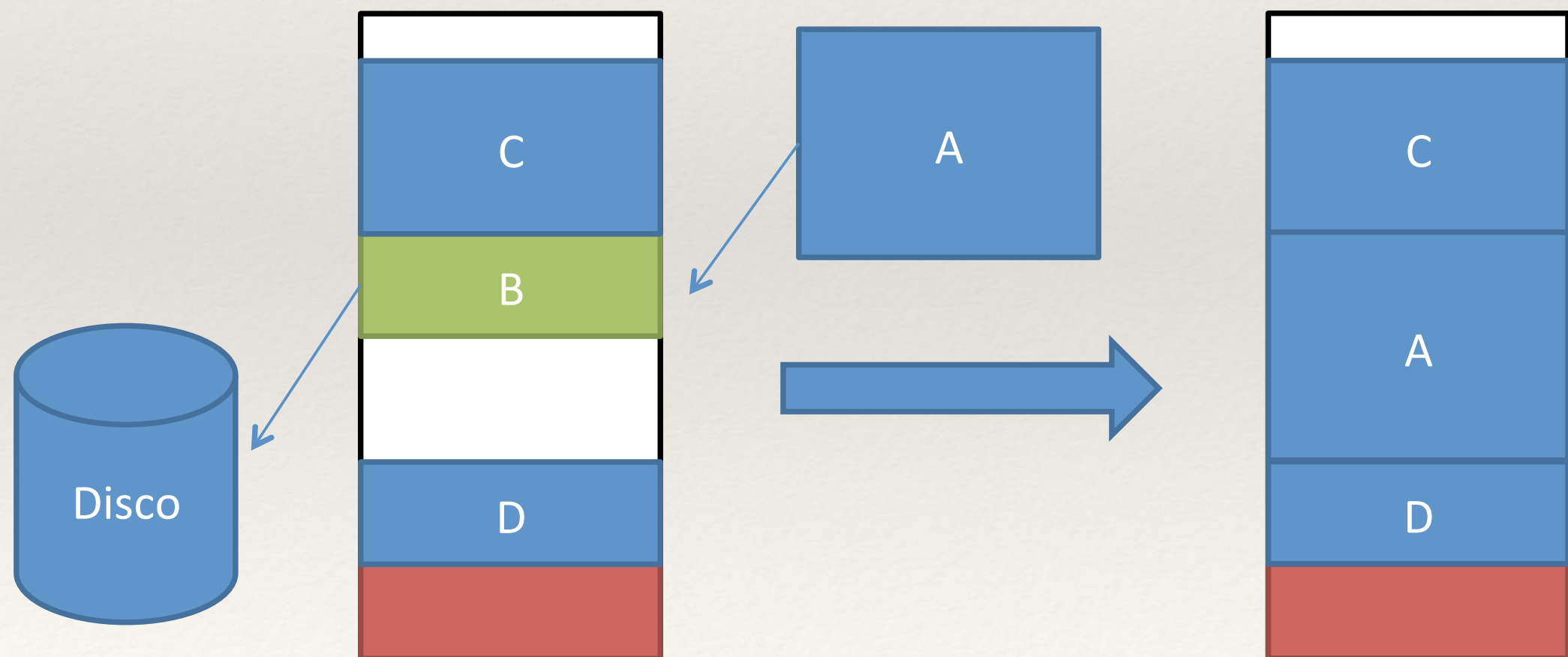
- ❖ Sistemas operacionais modernos dificilmente possuem apenas 3 processos em execução ao mesmo tempo
 - ❖ tipicamente possuem de 40 a 100
 - ❖ eles são lançados e terminados a todo momento
- ❖ Problemas restantes
 - ❖ Como decidir qual processo fica em qual posição?
 - ❖ E se a memória acabar, o que fazer?
- ❖ Principais soluções desenvolvidas ao longo dos anos:
 - ❖ *Swapping*
 - ❖ Memória Virtual

Swapping

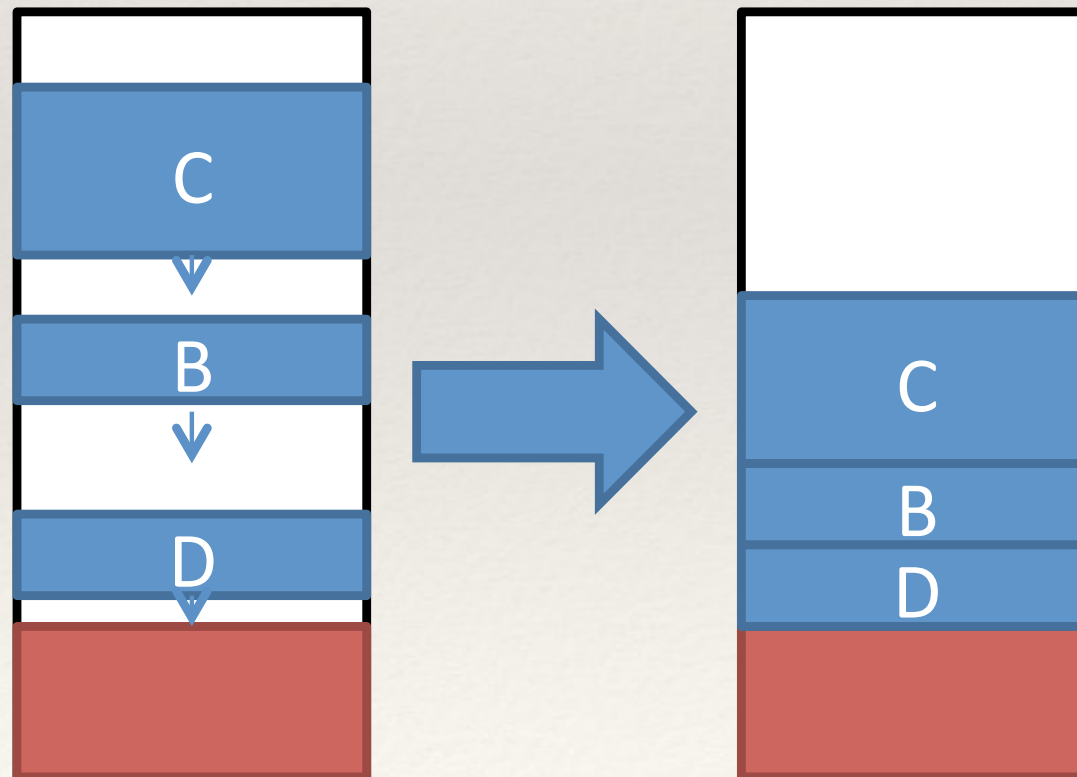
- ❖ Processo mais simples que Memória Virtual, para colocação de vários processos na memória simultaneamente
- ❖ É possível ser implementado apenas com registradores base e limite
- ❖ Os programas são posicionados onde há espaço na memória, como no exemplo abaixo:



- ❖ No *swapping*, um processo sempre está inteiro na memória, ou inteiro no disco (caso esteja dormindo ou bloqueado).
- ❖ Se um processo precisa de espaço que não há disponível na memória
 - ❖ o sistema deve escolher outro processo para ser passado para o disco e abrir espaço suficiente



- ❖ Após muitas entradas e saídas
 - ❖ pode haver espaço total desocupado suficiente para um processo que precisa entrar na memória, mas dividido em vários espaços vazios
 - ❖ Neste caso, diz-se que a memória se encontra **fragmentada**
- ❖ O sistema pode realizar uma **compactação de memória**, que consiste em deslocar todos os processos de forma a remover os espaços livres:



- ❖ Compactação de memória é uma operação que deve ser evitada
 - ❖ Alto custo de desempenho
 - ❖ *Um computador com 16 GB de RAM e com capacidade de transferência de memória de 20 GB/s, como um Intel Core i7 de 4 núcleos, levaria quase 1 segundo para realizá-la*
- ❖ Uso de memória dinamicamente (*heap*) é problema em *swapping*
 - ❖ Ex: operador `new` do Java e C++, ou `malloc()` do C
 - ❖ Solução: reposicionar um ou mais processos, ou até mesmo colocar no disco
 - ❖ risco de um processo crescer e ocupar a memória inteira

- ❖ Requisitos do *swapping*
 - ❖ registradores **base** e **limite**
 - ❖ necessário o gerenciamento da ocupação da memória
 - ❖ *qual processo detém qual região da memória?*

Gerenciamento *bitmapped*

- ❖ Utiliza um mapa de bits para representar os espaços vazios (0) e os ocupados (1):



- ❖ Acima, a ocupação de espaços é: $A = 5$, $B = 6$, $C = 4$, $D = 6$, $E = 3$. Há 8 espaços vazios distribuídos entre 3 buracos.
- ❖ O bitmap correspondente é:

```
11111000
11111111
11001111
11111000
```

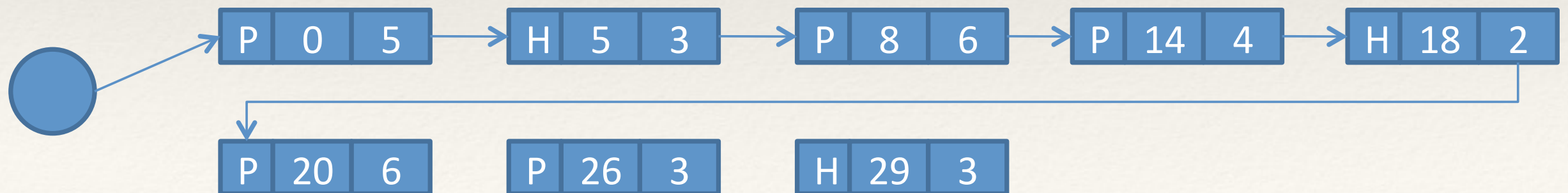

- ❖ Gerenciar palavra por palavra de memória é custoso computacionalmente e gasta muito espaço da própria memória
- ❖ Solução: separar a memória em pequenos pedaços
 - ❖ tamanhos entre 32 a 4096 (4K) palavras
- ❖ Uma memória de 64K palavras dividida em pedaços de 2K terá 32 pedaços

Influência do tamanho do pedaço no gerenciamento de memória

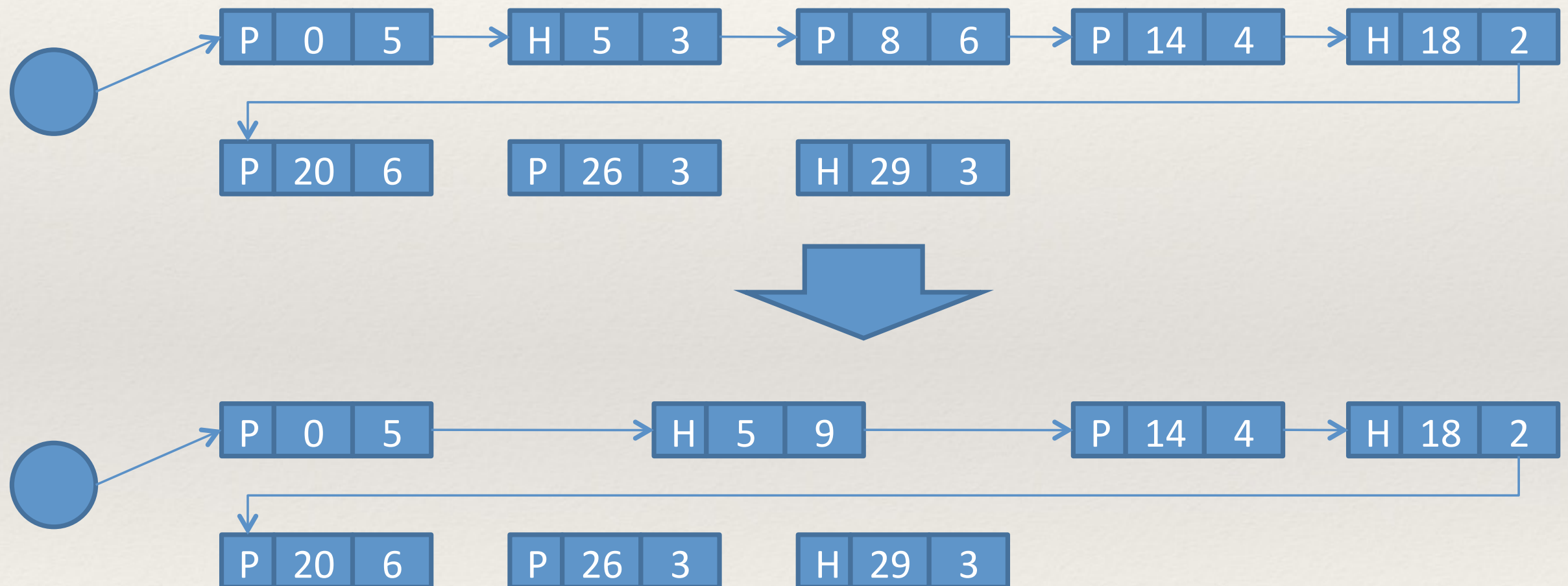
Fator	Menor (32 pedaços)	Maior (4K pedaços)
Desempenho computacional	pior , pois menor	<u>melhor</u> , pois maior
Desperdício de memória pelo processo	<u>melhor</u> , pois menor	pior , pois maior
Quantidade de buracos	pior , pois maior	<u>melhor</u> , pois menor

Gerenciamento com Lista

- ❖ Outra maneira de gerenciar é utilizando uma lista encadeada de espaços ocupados e desocupados
- ❖ Não precisa separar memória em pedaços, pode utilizar endereços absolutos.
- ❖ A lista marca os seguintes atributos:
 - ❖ Ocupado (P) ou desocupado (H)
 - ❖ Posição de início
 - ❖ Comprimento do espaço
- ❖ Considere a distribuição de processos do slide anterior:



- ❖ Ao remover o processo B da memória, une-se os nós de seu espaço e o do buraco anterior a ele:



Algoritmos de posicionamento

- ❖ *First fit*: posiciona processo no primeiro espaço possível
- ❖ *Next fit*: *first fit*, mas busca a partir da última inserção
- ❖ *Best fit*: analisa espaços e posiciona no mais apertado
- ❖ *Worst fit*: analisa espaços e posiciona no mais largo
- ❖ *Quick fit*: separa memória em tamanhos de processos
- ❖ Todos tem desempenho similar, exceto *quick fit*