Gerenciamento de Memória

Sistemas Operacionais

Prof. Pedro Ramos
pramos.costar@gmail.com

Pontifícia Universidade Católica de Minas Gerais
ICEI - Departamento de Ciência da Computação

ONDE ESTAMOS NO CURSO

Já discutimos:

- Processos e Threads
- Escalonamento de CPU
- Sincronização e Deadlock

Próximo:

- Gerenciamento de Memória

Ainda por vir:

- Sistemas de Arquivos e Armazenamento (I/O)
- Sistemas Distribuídos
- Segurança

GERENCIAMENTO DE MEMÓRIA

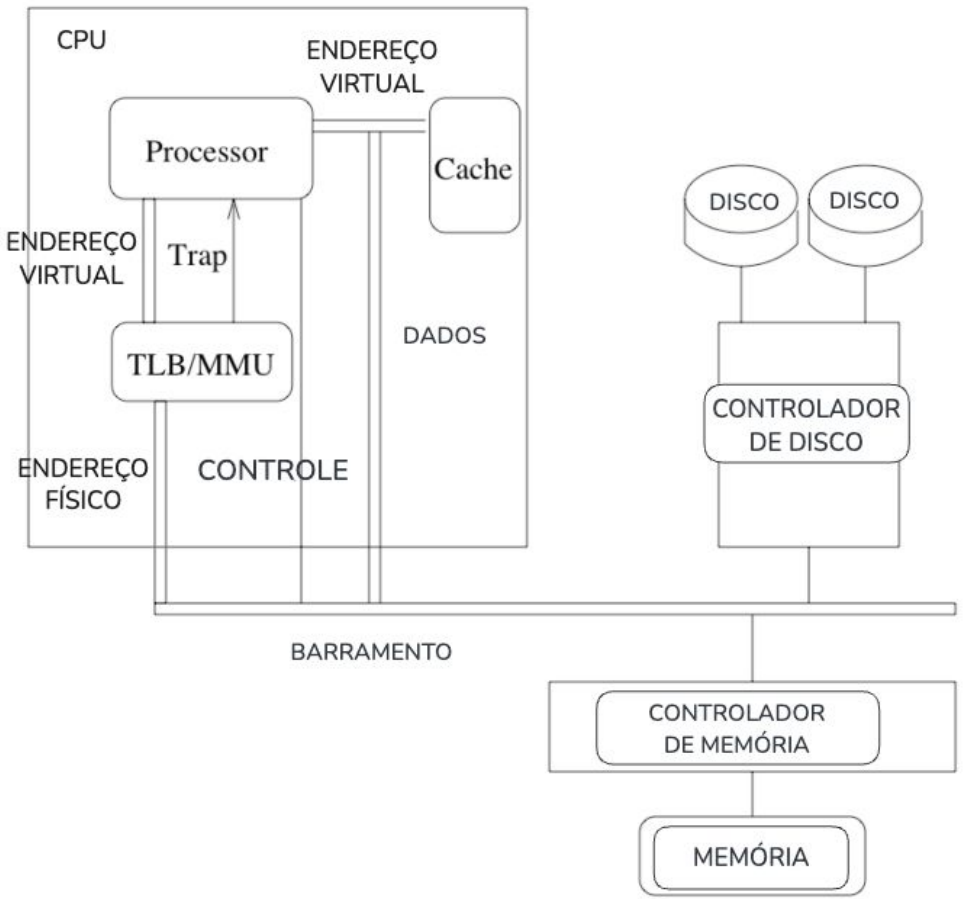
- Onde está o processo em execução?
- Como permitimos que múltiplos processos usem a memória principal simultaneamente?
- O que é um endereço e como ele é interpretado?

BACKGROUND: ARQUITETURA

- O executável do programa começa no disco.
- O sistema operacional carrega o programa na memória.
- A CPU busca instruções e dados da memória enquanto executa o programa.

BACKGROUND: ARQUITETURA

- O executável do programa começa no disco.
- O sistema operacional carrega o programa na memória.
- A CPU busca instruções e dados da memória enquanto executa o programa.

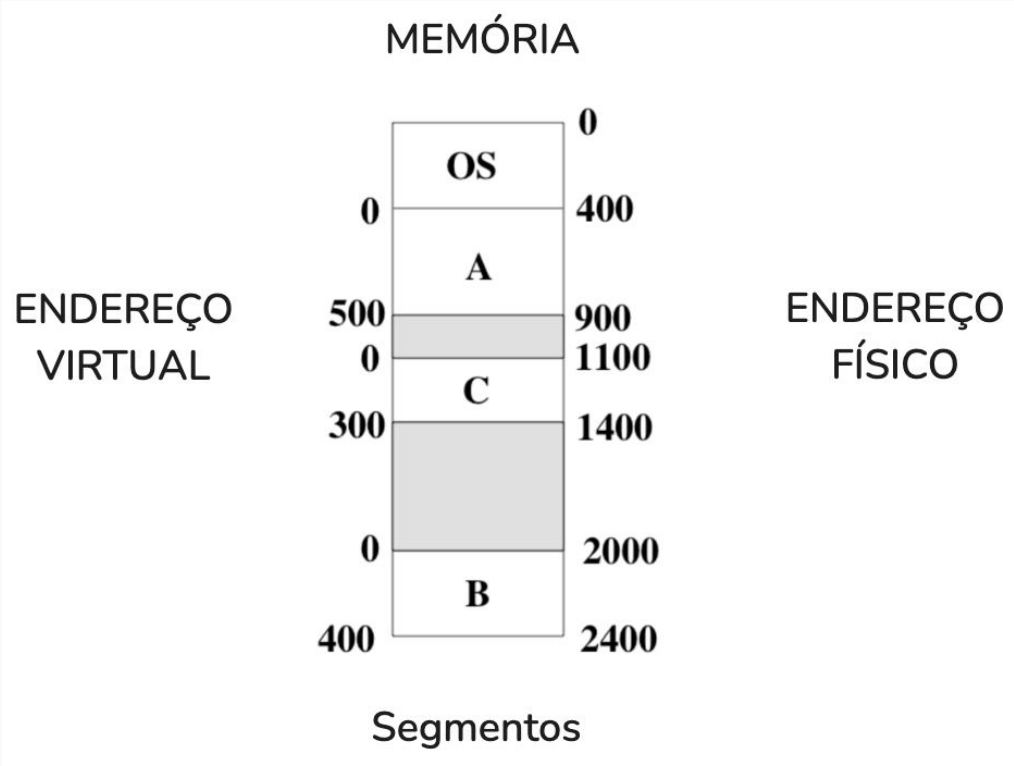


GERENCIAMENTO DE MEMÓRIA: TERMINOLOGIA

Segmento: Um pedaço de memória atribuído a um processo.

Endereço Físico: um endereço real na memória.

Endereço Virtual: um endereço relativo ao início do espaço de endereçamento de um processo.

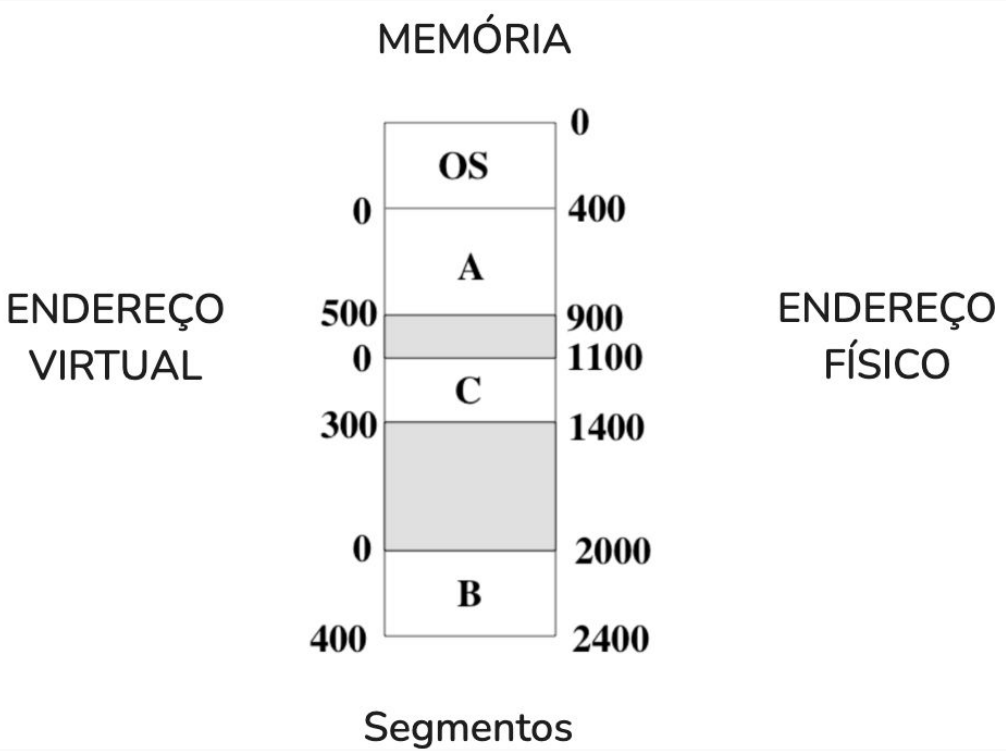


GERENCIAMENTO DE MEMÓRIA: TERMINOLOGIA

Segmento: Um pedaço de memória atribuído a um processo.

Endereço Físico: um endereço real na memória.

Endereço Virtual: um endereço relativo ao início do espaço de endereçamento de um processo.



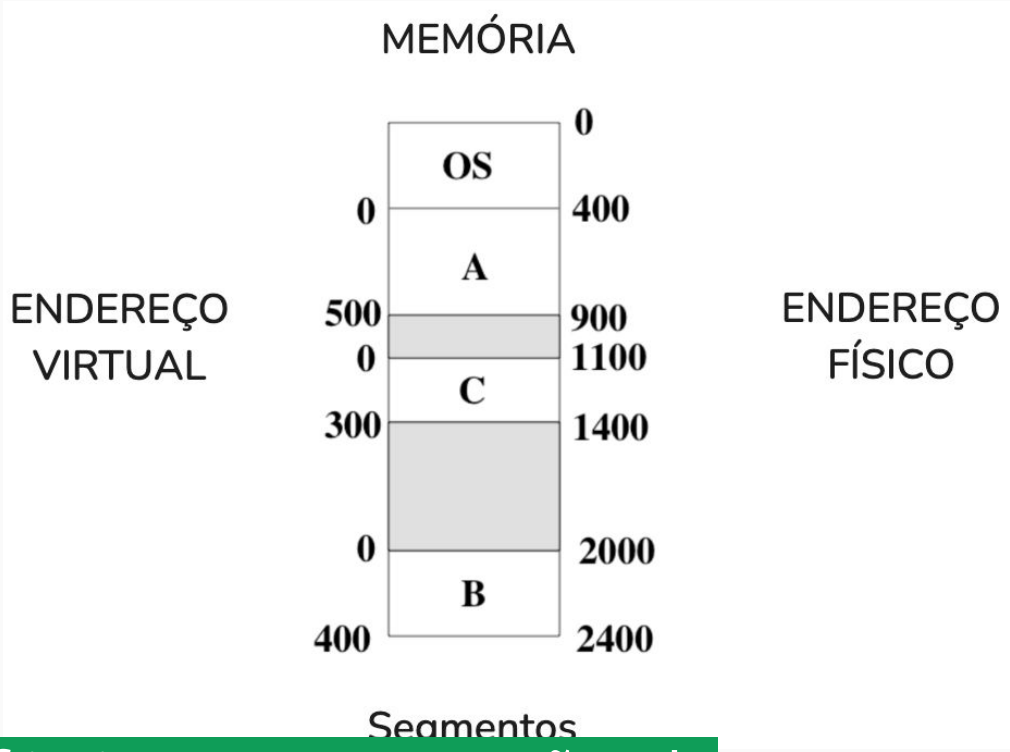
PORQUÊ NÃO USAR SOMENTE ENDEREÇOS FÍSICOS?

GERENCIAMENTO DE MEMÓRIA: TERMINOLOGIA

Segmento: Um pedaço de memória atribuído a um processo.

Endereço Físico: um endereço real na memória.

Endereço Virtual: um endereço relativo ao início do espaço de endereçamento de um processo.



=> memória virtual infinita + separação de espaços de endereço por processo

DE ONDE VÊM OS ENDEREÇOS?

Como os programas geram endereços de instruções e dados?

3 ABORDAGENS POSSÍVEIS:

- Tempo de **compilação**: O compilador gera a localização física exata na memória a partir de **uma posição inicial fixa k**. *O SO não faz nada.*

- Tempo de **carregamento**: O compilador gera um endereço ESTÁTICO, mas **no tempo de carregamento o SO determina a posição inicial do processo**. Uma vez que o processo é carregado, ele não se move na memória.

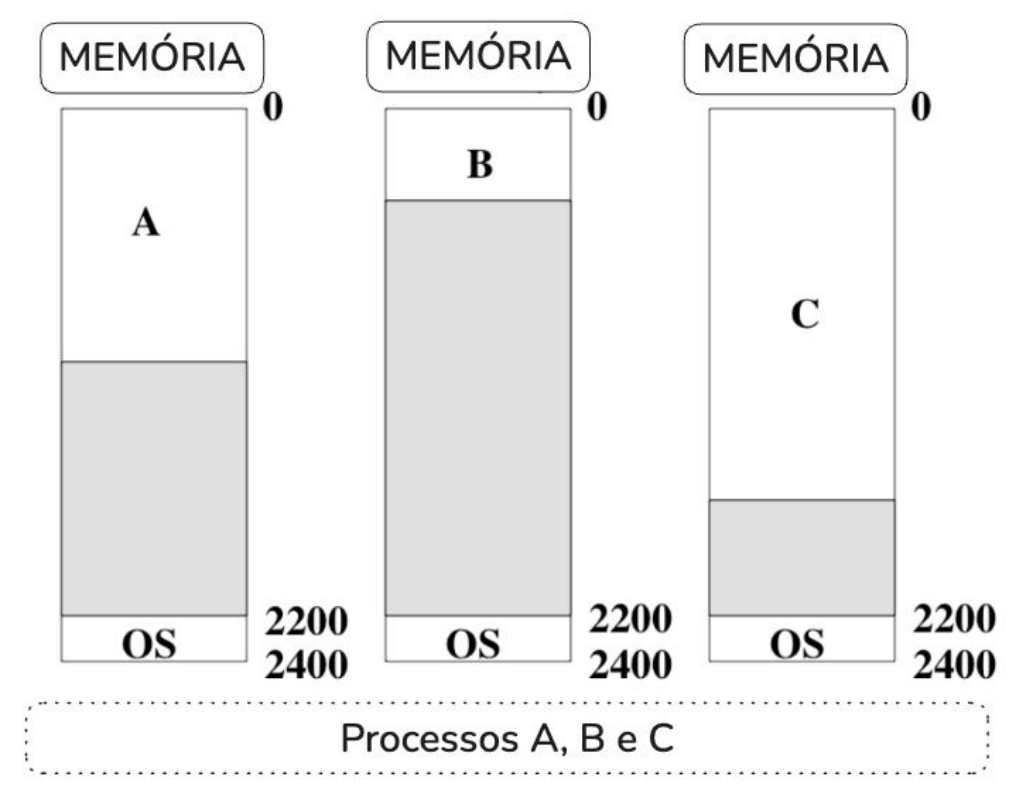
- Tempo de **execução**: O compilador gera um endereço, e o SO pode colocá-lo em qualquer lugar que quiser na memória (*endereço virtual*).

UNIPROGRAMAÇÃO

- O SO obtém uma parte fixa da memória (*em DOS, são os endereços mais altos da memória*).
- 1 processo é executado por vez.
- O processo é sempre carregado começando no endereço 0.
- O processo é executado em uma seção contígua da memória.
- O compilador pode gerar endereços físicos.
- Endereço máximo = [Tamanho da Memória] - [Tamanho do SO]
- O SO é protegido do processo: o SO verifica os endereços usados pelo processo.

UNIPROGRAMAÇÃO

- Simples, mas **não permite sobreposição de I/O e computação.**



VÁRIOS PROGRAMAS COMPARTILHAM A MESMA MEMÓRIA AO MESMO TEMPO

Transparência:

- Queremos que múltiplos processos coexistam na memória.
- Nenhum processo sabe que a memória é compartilhada.
- Os processos não devem se importar com a parte física da memória que são atribuídos.

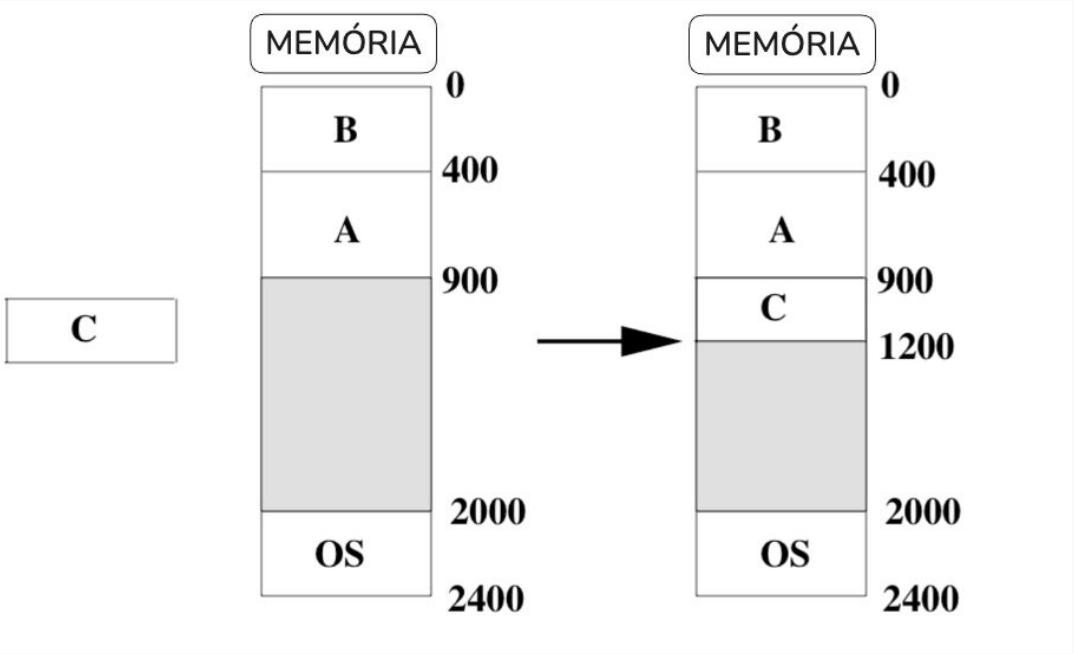
Segurança:

- Os processos não devem ser capazes de corromper um ao outro.
- Os processos não devem ser capazes de corromper o SO.

Eficiência:

- O desempenho da CPU e da memória não deve ser degradado significativamente devido ao compartilhamento.

RELOCAÇÃO



Coloque o SO na memória mais alta.

Em tempo de compilação/ligação **o processo começa em 0 com um endereço máximo = [tamanho da memória] - [tamanho do SO].**

Novo processo é carregado alocando um segmento contíguo de memória no qual o processo cabe.

O primeiro (menor) endereço físico do processo é o endereço base e o maior endereço físico que o processo pode acessar é o endereço limite.

RELOCAÇÃO

Relocação Estática:

- No tempo de carregamento, o SO **ajusta os endereços em um processo para refletir sua posição na memória**.
- Uma vez que um processo é atribuído a um lugar na memória e começa a executar, ***o SO não pode movê-lo***. **(Por quê?)**

RELOCAÇÃO

Relocação Estática:

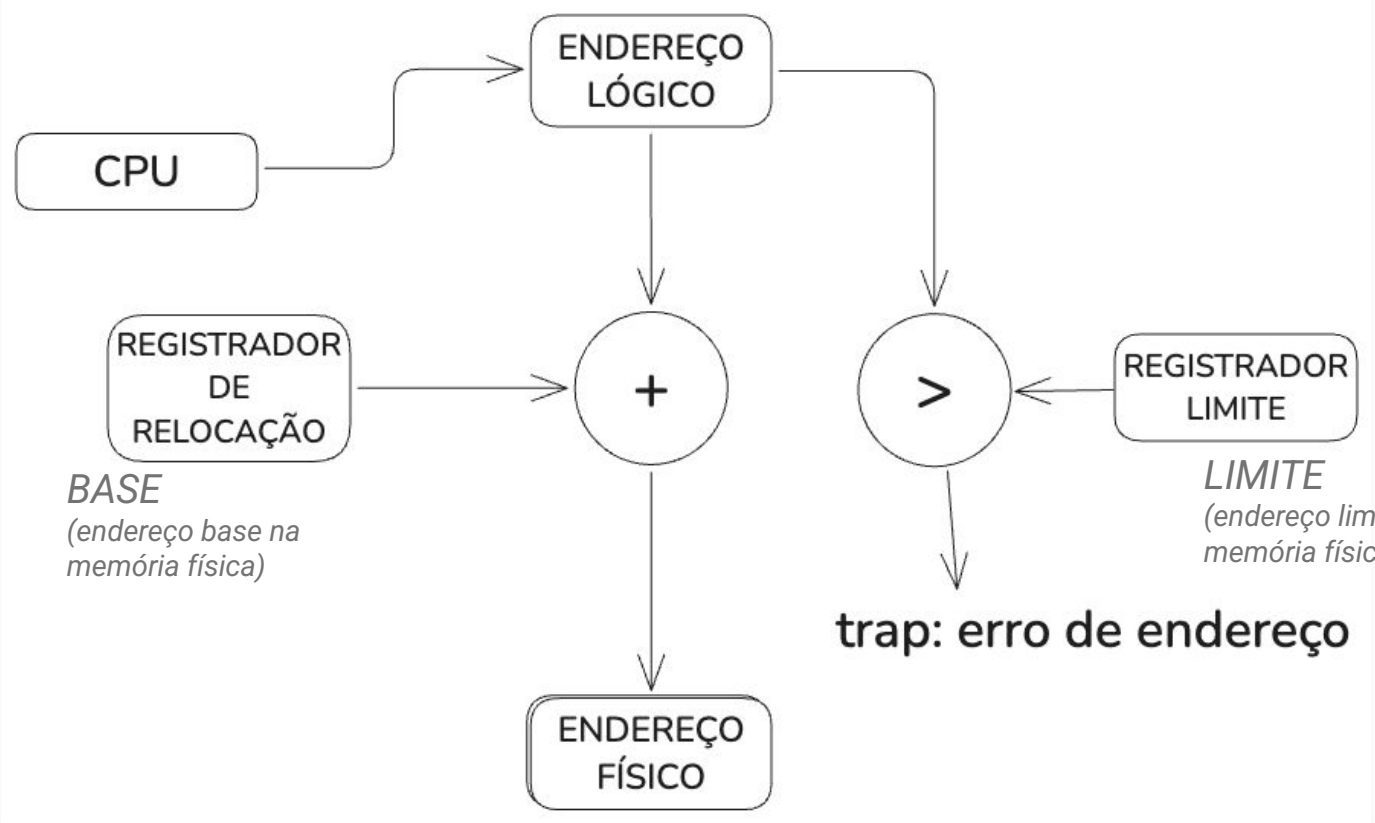
- No tempo de carregamento, o SO ajusta os endereços em um processo para refletir sua posição na memória.
- Uma vez que um processo é atribuído a um lugar na memória e começa a executar, o SO não pode movê-lo.

Relocação Dinâmica:

- O hardware adiciona um **registro de relocação** (base) ao endereço virtual para obter um endereço físico;
- O hardware **compara** o **endereço com o registro limite** (o endereço deve ser menor que o limite).
- Se o teste falhar, o processador dispara uma armadilha (*trap -> um tipo de interrupção*) de endereço e ignora o endereço físico.

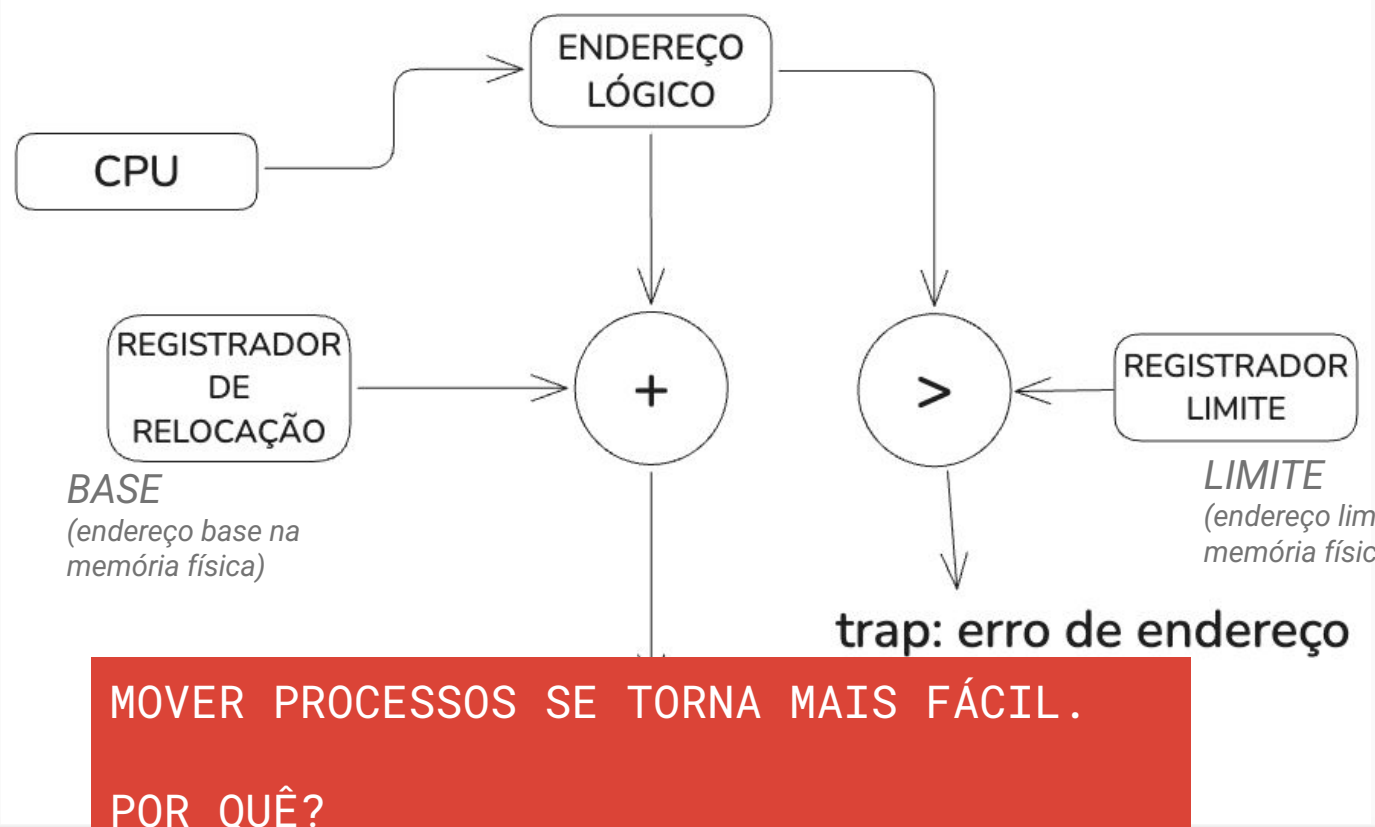
RELOCAÇÃO

Relocação Dinâmica



RELOCAÇÃO

Relocação Dinâmica



MOVER PROCESSOS SE TORNA MAIS FÁCIL.

POR QUÊ?

RELOCAÇÃO DINÂMICA

Vantagens:

- O SO pode **mover facilmente um processo** durante a execução.
- O SO pode permitir que um **processo cresça ao longo do tempo**.
- **Hardware** simples e rápido: **dois registradores especiais**, uma adição e uma comparação.

Desvantagens:

- *Desacelera o hardware* devido à adição em cada referência de memória.
- *Não pode compartilhar memória* (como por exemplo, o texto do programa) entre processos.
- O processo ainda é *limitado pelo tamanho da memória física.*
- O grau de *multiprogramação é muito limitado, pois toda a memória de todos os processos ativos deve caber na memória*.
- *Complica o gerenciamento de memória.*

RELOCAÇÃO: PROPRIEDADES

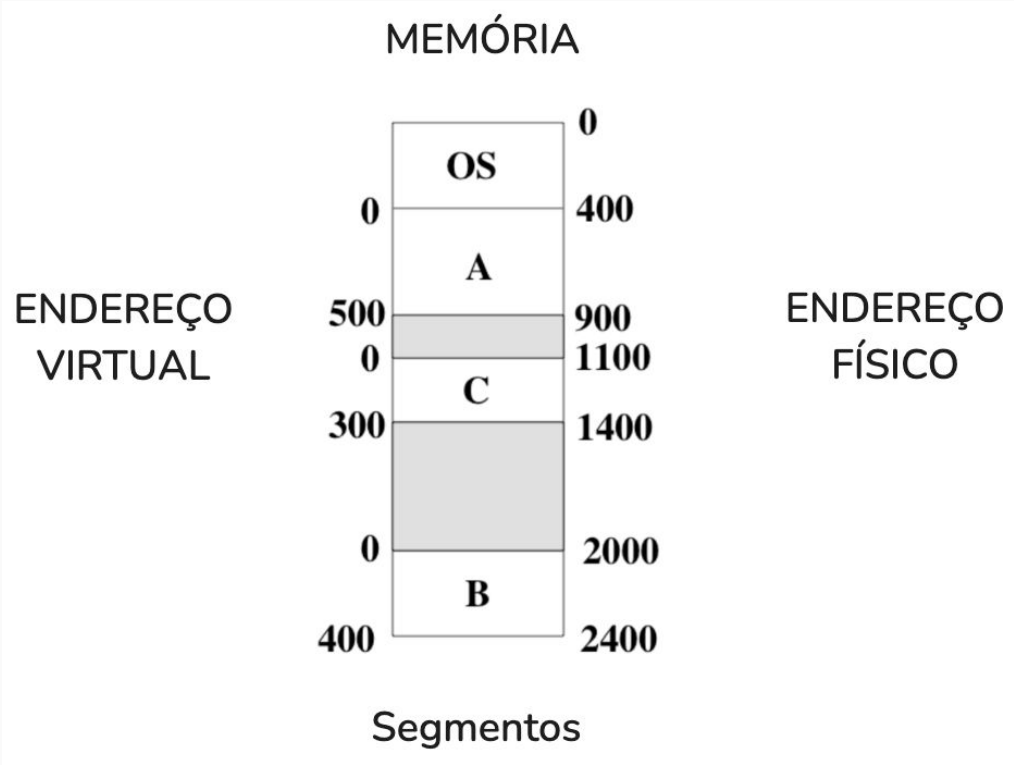
Transparência: os processos estão 100% alheios ao compartilhamento.

Segurança: cada referência de memória é verificada.

Eficiência: as verificações de memória e a tradução de endereço virtual para físico são rápidas, pois são feitas em hardware, **MAS se um processo crescer, pode ser necessário movê-lo, o que é muito lento.**

RECAPITULANDO ATÉ AGORA:

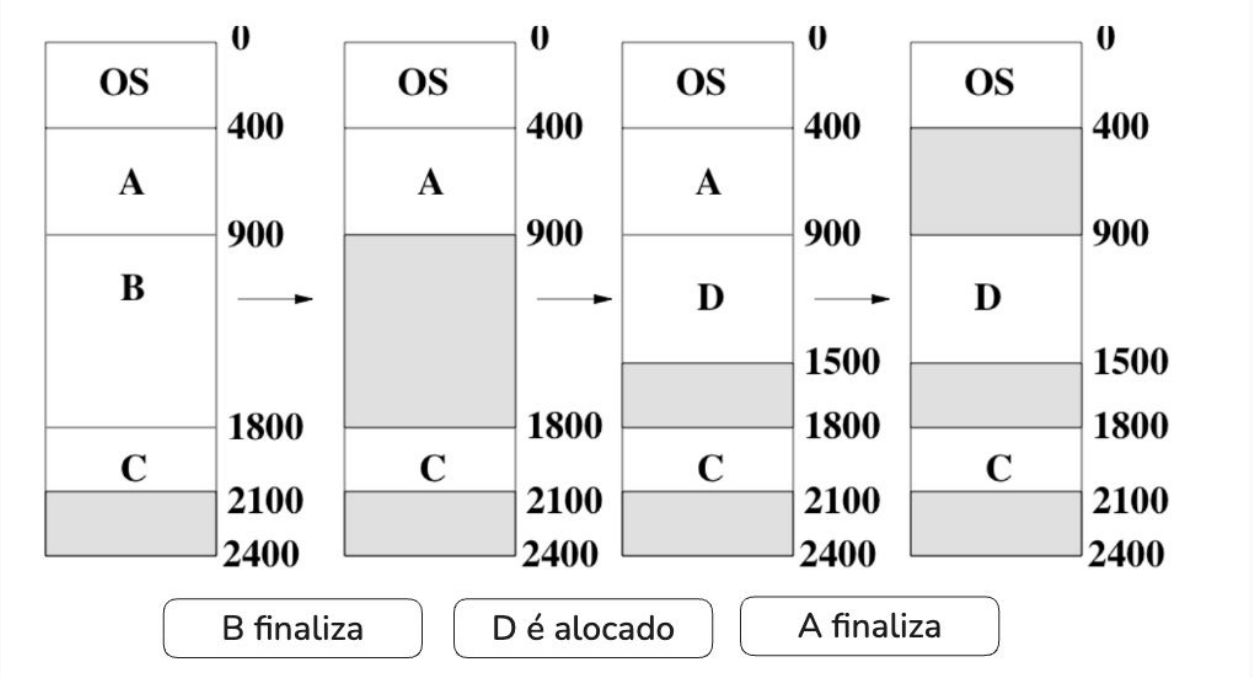
- Uniprogramação
- Relocação estática
- Relocação dinâmica
- Alocação contígua



GERENCIAMENTO DE MEMÓRIA: ALOCAÇÃO DE MEMÓRIA

O SO deve manter registro de qual memória está disponível e qual está sendo utilizada.

FRAGMENTOS: Pedaços livres de memória.



Com a chegada de um novo PROCESSO, o SO deve decidir em qual FRAGMENTO colocar o processo.

POLÍTICAS DE ALOCAÇÃO DE MEMÓRIA

First-Fit: **aloca o primeiro da lista no qual o processo cabe**. A busca pode começar com o primeiro fragmento ou onde a busca First-Fit anterior terminou.

Best-Fit: aloca o menor fragmento que seja grande o suficiente para conter o processo. **O SO deve pesquisar toda a lista** ou armazenar a lista ordenada por tamanho de fragmento.

Worst-Fit: **aloca o maior fragmento para o processo**. Novamente, o SO deve pesquisar toda a lista ou manter a lista ordenada.

Simulações mostram que First-Fit e Best-Fit geralmente produzem melhor utilização de armazenamento do que Worst-Fit; First-Fit é geralmente mais rápido que Best-Fit.

FRAGMENTAÇÃO

Fragmentação Externa

- O carregamento e descarregamento frequente de programas faz com que o espaço livre seja dividido em pequenos pedaços.

- A fragmentação externa existe quando há memória suficiente para caber um processo na memória, mas o espaço não é contíguo.

- **Regra de 50%:** Simulações mostram que para cada 2N blocos alocados, N blocos são perdidos devido à fragmentação (ou seja, *1/3 do espaço de memória é desperdiçado*).

- Queremos uma política de alocação que **minimize o espaço desperdiçado.**

FRAGMENTAÇÃO

Fragmentação Interna:

- Considere um processo de tamanho 8846 bytes e um bloco de tamanho 8848 bytes.
  ⇒ É melhor alocar o processo inteiro de 8848 bytes do que rastrear 2 bytes livres.

- A fragmentação interna existe quando a memória interna de um processo alocado em uma partição/fragmento é desperdiçada.

COMPACTAÇÃO

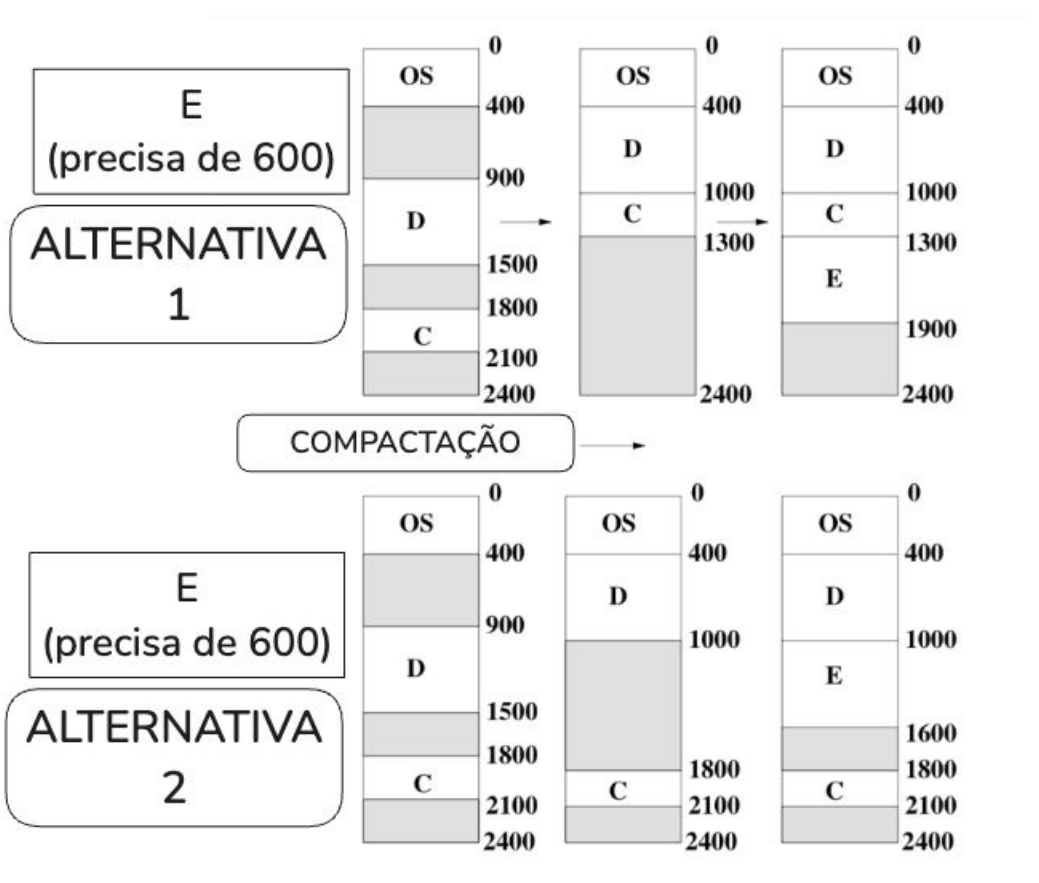
O QUANTO DE MEMÓRIA É MOVIDO?

QUAL O TAMANHO MÁXIMO DO BLOCO QUE É CRIADO?

... HÁ OUTRA ALTERNATIVA?

O PROCESSO QUE FAZ I/O, PRECISA FICAR NA MEMÓRIA?

***QUANDO** COMPACTAR?*



SWAP

- Retirar um processo para o disco, liberando toda a memória que ele ocupa.

- Quando o processo se torna ativo novamente, o SO recarrega ele na memória.
  - Com relocação estática, o processo deve ser colocado na mesma posição.
  - Com relocação dinâmica, o SO encontra uma **nova posição na memória para o processo e atualiza os registradores de relocação e limite**.

- Como o *swap* se relaciona com o escalonamento de CPU? *O swap atrapalha o escalonamento?*

- Faça compactação somente quando um processo entrar do swap para a memória.

RESUMO DOS PROBLEMAS ATÉ AGORA

- Fragmentação
  - Compactação frequente necessária (alta complexidade de tempo - cópia dos processos).

- Alocação contígua
  - Dificuldade em aumentar e diminuir processos na memória.

- Processos devem estar todos na memória ao mesmo tempo para o escalonamento e o SO poder gerenciá-los.
  - *Swap* ajuda mas não é perfeito (alta latência)

PAGINAÇÃO: MOTIVAÇÃO E FUNCIONALIDADES

- Regra 90/10: Processos gastam 90% do tempo acessando 10% do seu espaço na memória.
  => Mantenha apenas as partes de um processo na memória que estão sendo realmente utilizadas.

- Páginas simplificam o problema de ajuste de fragmentos.

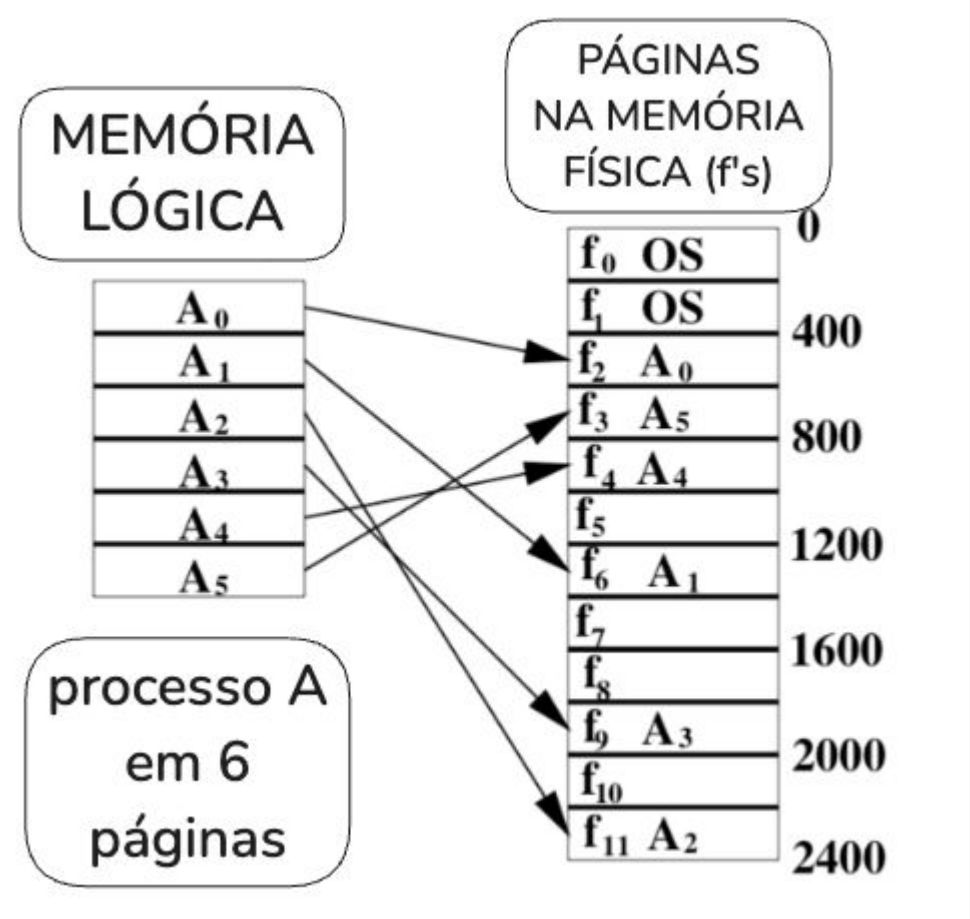
- A memória lógica (virtual) do processo é contígua, mas **as páginas não precisam ser alocadas contiguamente na memória**.

- Dividindo a memória em páginas de tamanho fixo, podemos ***eliminar a fragmentação externa***.

- A paginação não elimina a fragmentação interna (*1/2 página por processo, em média*).

PAGINAÇÃO - EXEMPLO

Qual o papel do SO?

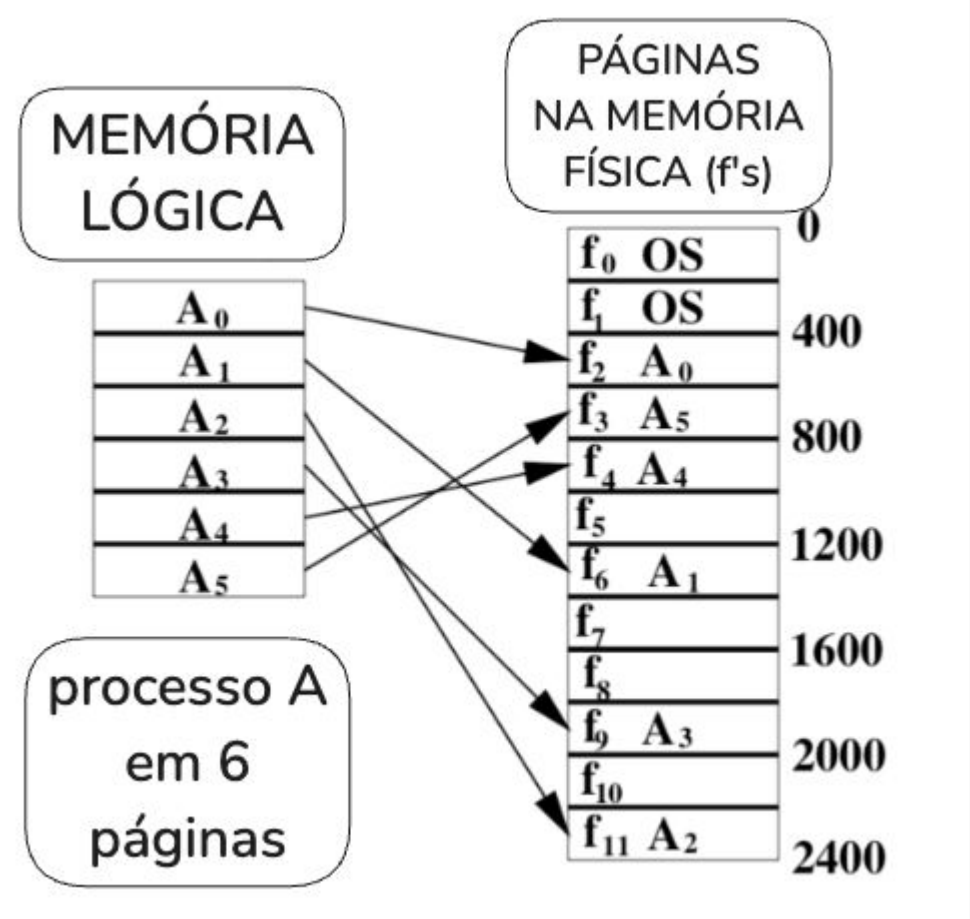


PAGINAÇÃO - EXEMPLO

Qual o papel do SO?

GERENCIAR A TABELA DE PÁGINAS.

TRADUZIR ENDEREÇOS DAS PÁGINAS PARA ENDEREÇOS FÍSICOS.



HARDWARE DE PAGINAÇÃO

COMO ENCONTRAR ENDEREÇOS QUANDO AS PÁGINAS NÃO ESTÃO ALOCADAS CONTIGUAMENTE EM MEMÓRIA?

Próxima aula: **Paginação**

PERGUNTAS?

REFERÊNCIAS

- **TANENBAUM, Andrew.** Sistemas operacionais modernos.
- **SILBERSCHATZ, Abraham et al.** Fundamentos de sistemas operacionais: princípios básicos.
- **MACHADO, Francis; MAIA, Luiz Paulo.** Arquitetura de Sistemas Operacionais.
- **CARISSIMI, Alexandre et al.** Sistemas operacionais.