**MC514**

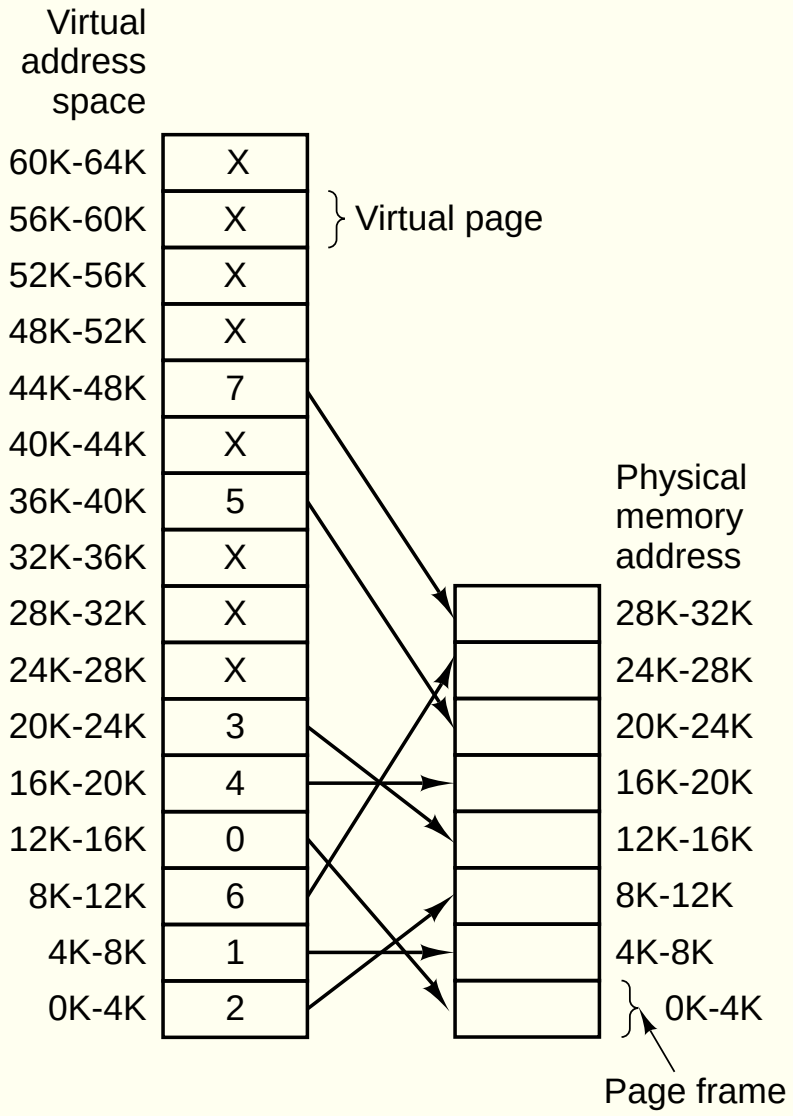
**Sistemas Operacionais:**

**Teoria e Prática**

1s2006

# Gerenciamento de Memória III

# Paginação


Virtual address space
60K-64K
56K-60K
52K-56K
48K-52K
44K-48K
40K-44K
36K-40K
32K-36K
28K-32K
24K-28K
20K-24K
16K-20K
12K-16K
8K-12K
4K-8K
0K-4K
X
X
X
X
7
X
5
X
X
X
3
4
0
6
1
2
Virtual page
Physical memory address
28K-32K
24K-28K
20K-24K
16K-20K
12K-16K
8K-12K
4K-8K
0K-4K
Page frame

# Algoritmos para substituição de páginas

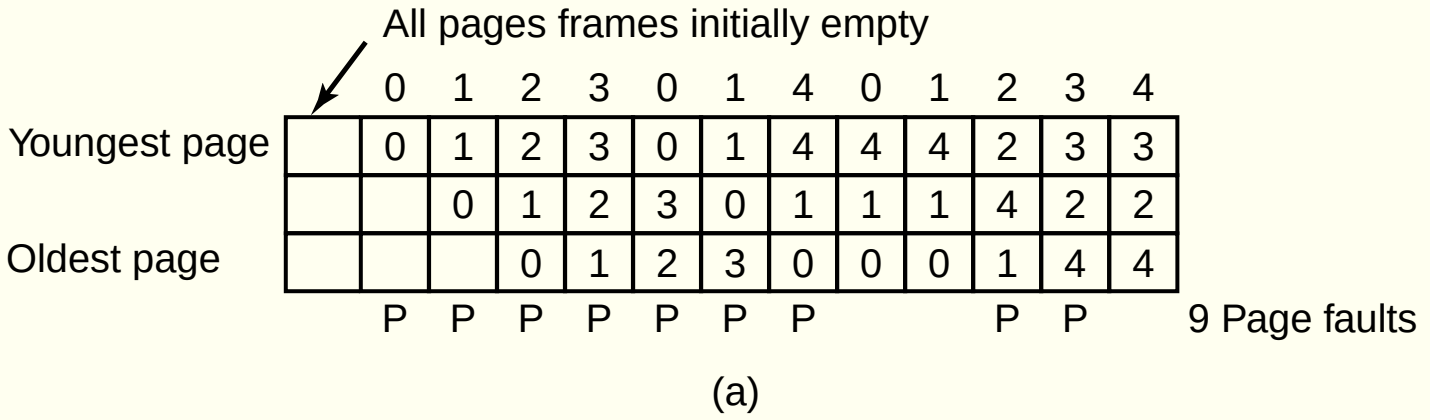
| Algorithm | Comment |
|---|---|
| Optimal | Not implementable, but useful as a benchmark |
| NRU (Not Recently Used) | Very crude |
| FIFO (First-In, First-Out) | Might throw out important pages |
| Second chance | Big improvement over FIFO |
| Clock | Realistic |
| LRU (Least Recently Used) | Excellent, but difficult to implement exactly |
| NFU (Not Frequently Used) | Fairly crude approximation to LRU |
| Aging | Efficient algorithm that approximates LRU well |
| Working set | Somewhat expensive to implement |
| WSClock | Good efficient algorithm |

# Modelagem de algoritmos

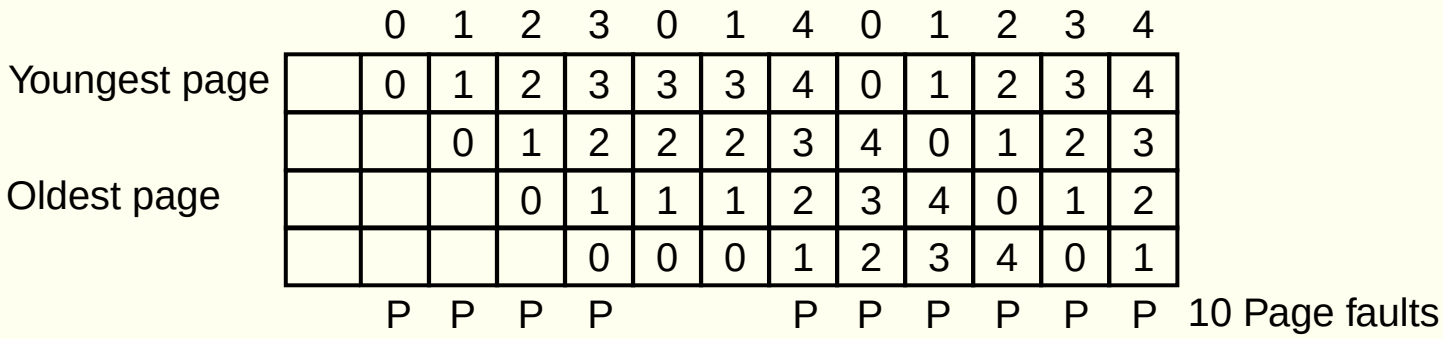
- Análise teórica
- Como avaliar quais algoritmos são bons?
- Como prever o número de faltas de páginas?

# Anomalia de Belady

## Algoritmo FIFO e número de page faults




(a)




(b)

# Algoritmos de pilha

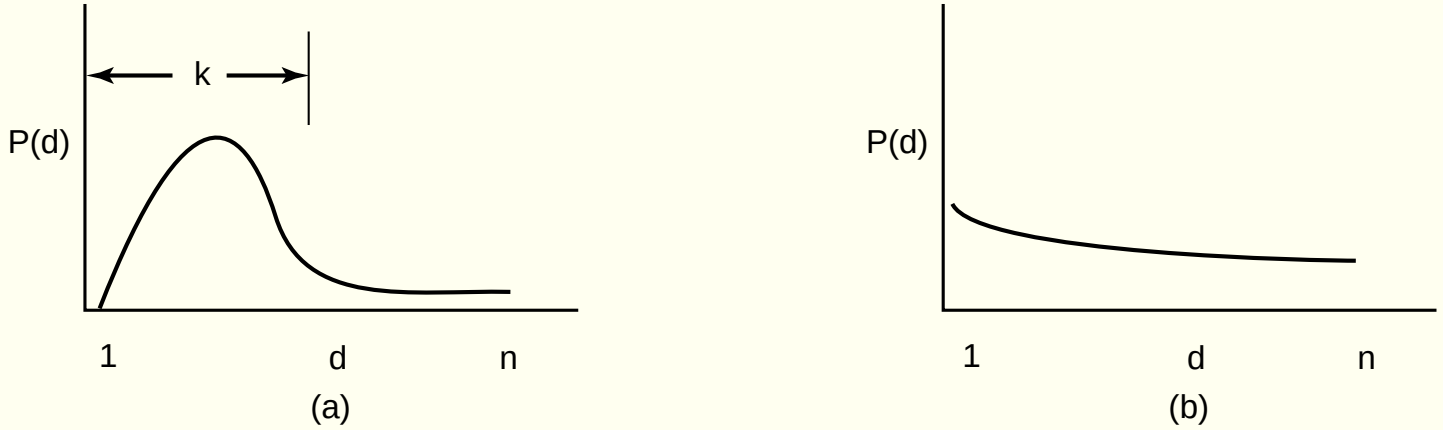
## Não apresentam a anomalia

| Reference string | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 7 | 4 | 7 | 3 | 3 | 5 | 5 | 3 | 1 | 1 | 1 | 7 | 1 | 3 | 4 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 7 | 4 | 7 | 3 | 3 | 5 | 5 | 3 | 1 | 1 | 1 | 7 | 1 | 3 | 4 | 1 |
| | | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 7 | 4 | 7 | 7 | 3 | 3 | 5 | 3 | 3 | 3 | 1 | 7 | 1 | 3 | 4 |
| | | | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 3 | 4 | 4 | 7 | 7 | 7 | 5 | 5 | 5 | 3 | 3 | 7 | 1 | 3 |
| | | | | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 6 | 6 | 6 | 4 | 4 | 4 | 7 | 7 | 7 | 5 | 5 | 5 | 7 | 7 |
| | | | | | 0 | 2 | 1 | 1 | 5 | 5 | 5 | 5 | 5 | 6 | 6 | 6 | 4 | 4 | 4 | 4 | 4 | 4 | 5 | 5 |
| | | | | | | 0 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| | | | | | | | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Page faults | P | P | P | P | P | P | P | | P | | | | | P | | | P | | | | | | P | |
| Distance string | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 4 | ∞ | 4 | 2 | 3 | 1 | 5 | 1 | 2 | 6 | 1 | 1 | 4 | 2 | 3 | 5 | 3 |

$$M(m, r) \subseteq M(m + 1, r)$$

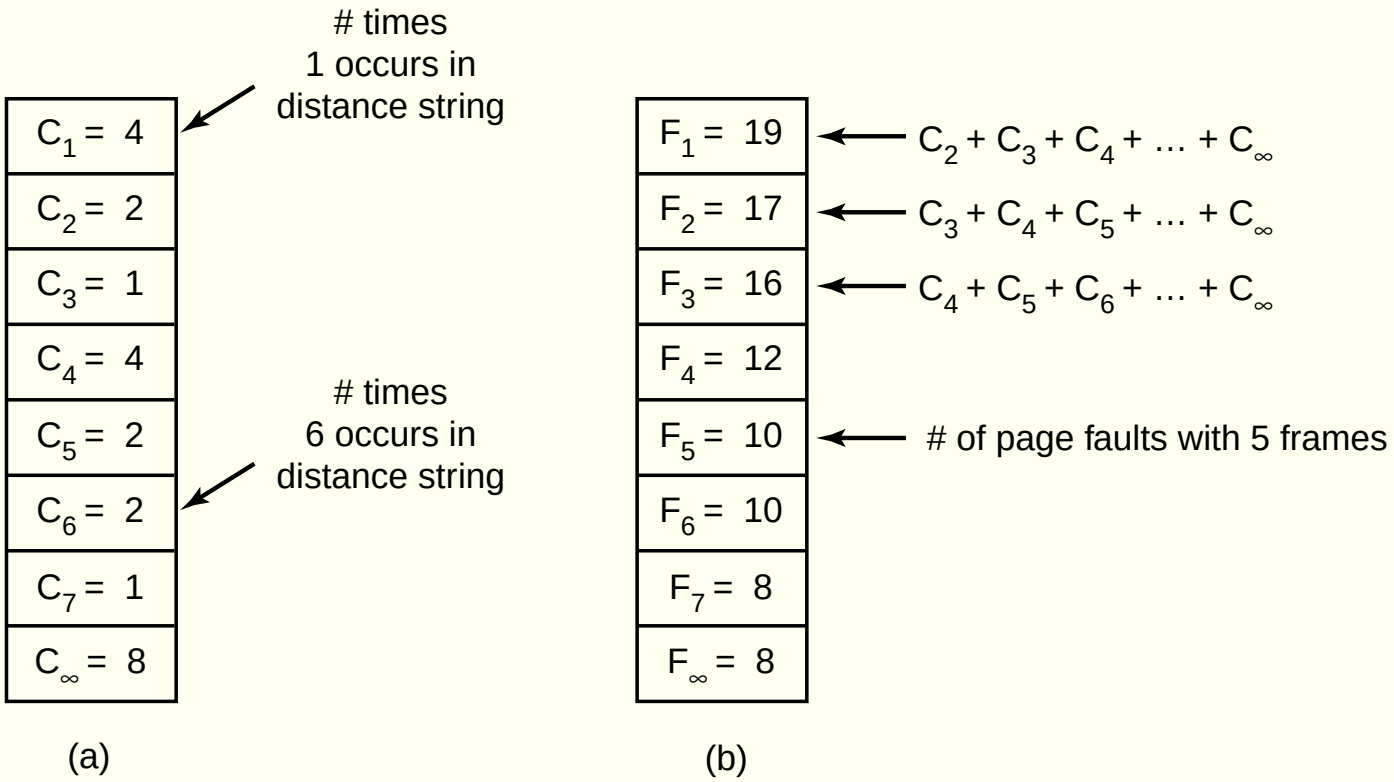
Exemplo: LRU

# Cadeia de Distâncias




- $d =$ distância da página ao topo da pilha antes da referência
- $k =$ boa escolha do número de páginas

# Previsão da freqüência de faltas de página




Números errados?

# Questões de projeto

- Políticas de alocação
- Compartilhamento

# Política Local ou Global

| | Age |
|---|---|
| A0 | 10 |
| A1 | 7 |
| A2 | 5 |
| A3 | 4 |
| A4 | 6 |
| A5 | 3 |
| B0 | 9 |
| B1 | 4 |
| B2 | 6 |
| B3 | 2 |
| B4 | 5 |
| B5 | 6 |
| B6 | 12 |
| C1 | 3 |
| C2 | 5 |
| C3 | 6 |

(a)

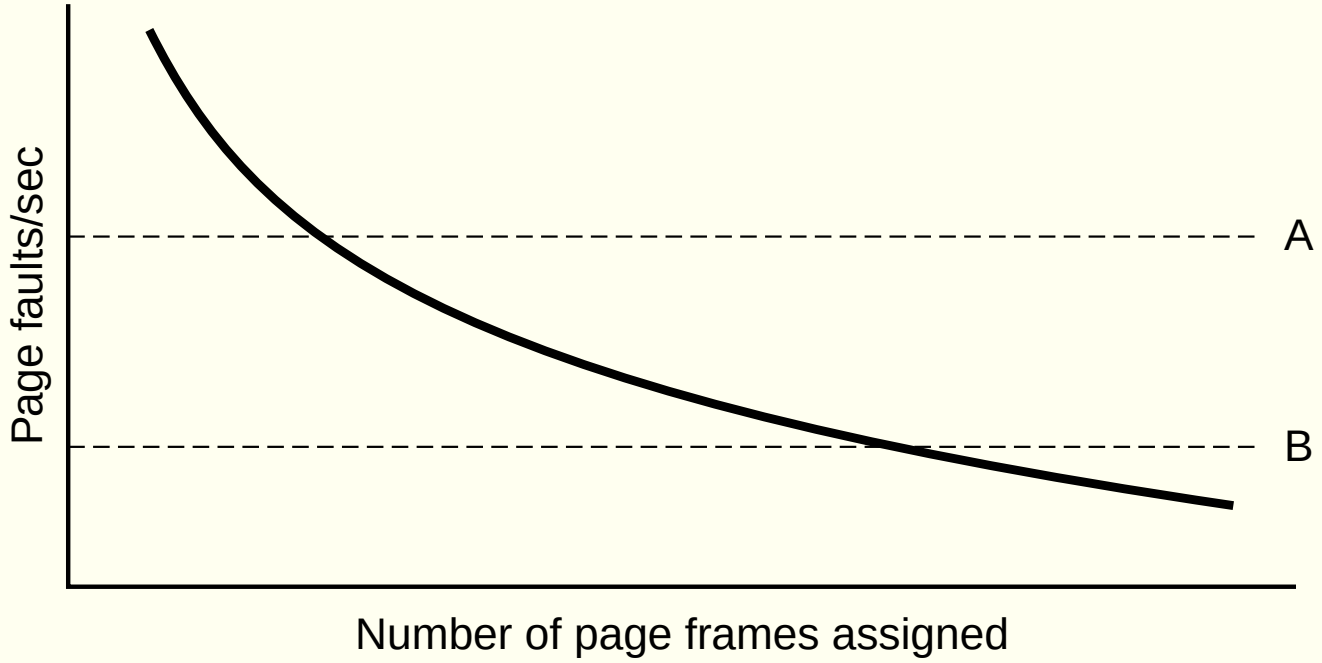
| |
|---|
| A0 |
| A1 |
| A2 |
| A3 |
| A4 |
| A6 |
| B0 |
| B1 |
| B2 |
| B3 |
| B4 |
| B5 |
| B6 |
| C1 |
| C2 |
| C3 |

(b)

| |
|---|
| A0 |
| A1 |
| A2 |
| A3 |
| A4 |
| A5 |
| B0 |
| B1 |
| B2 |
| A6 |
| B4 |
| B5 |
| B6 |
| C1 |
| C2 |
| C3 |

(c)

# Freqüência de falta de páginas




Serve como parâmetro para um alocador global

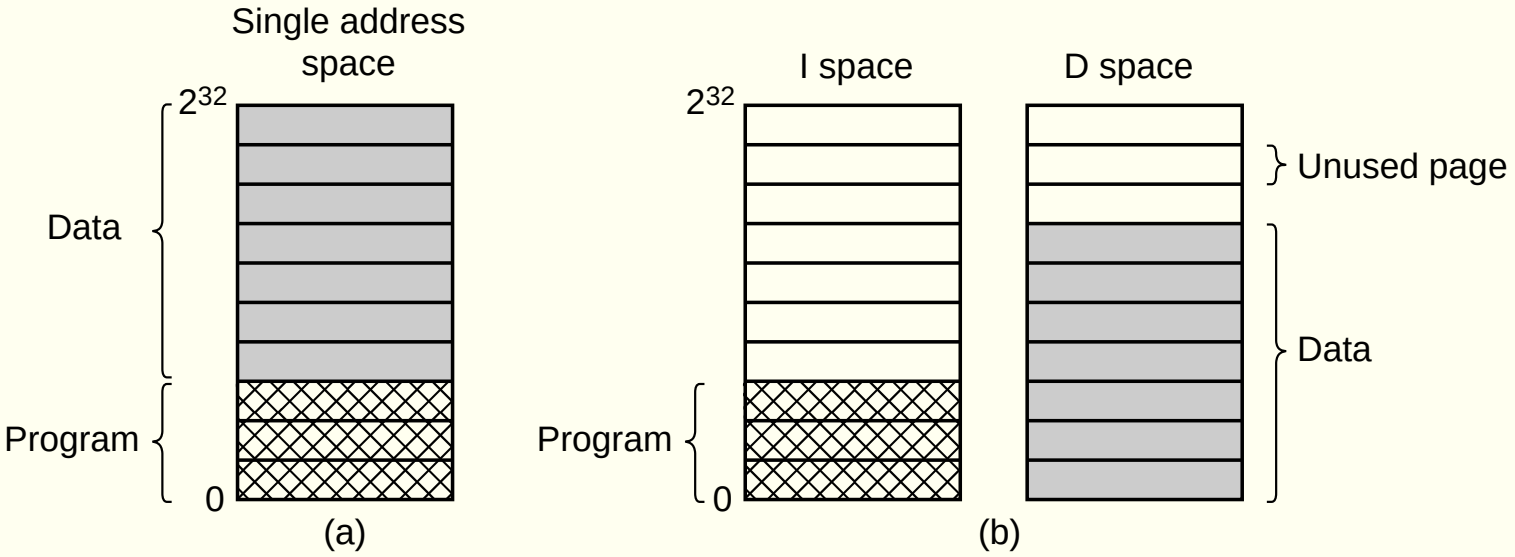
# Controle da carga

- *Thrashing*
  - O Sistema Operacional só se ocupa das tarefas de paginação e escalonamento
  - Todos os processos precisam de mais memória
  - Swap (como escolher quais processos vão para o disco?)

# Tamanho da página

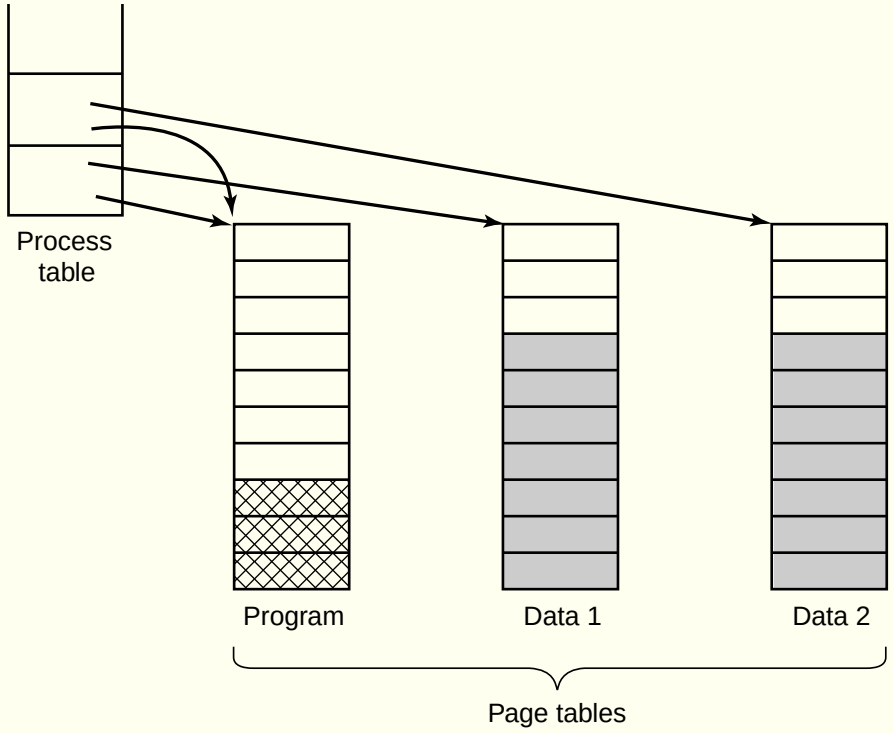
- Compromisso
  - Páginas grandes $\longrightarrow$ fragmentação interna
  - Páginas pequenas $\longrightarrow$ overhead de gerenciamento
- Tamanho nesta máquina: 4KB

# Espaços separados de endereçamento




- Simplificam o compartilhamento do código
- Permitem programas maiores

# Páginas compartilhadas




- Sistema Operacional deve cuidar da alocação e proteção
- fork e *copy on write*

# Política de limpeza

- Gravar as páginas modificadas na última hora pode ser pouco eficiente
- *Paging Daemon*
  - Varre periodicamente a memória
  - Tenta manter um número de *frames* livres

# Interface de memória virtual

- Permite aos usuários maior flexibilidade (mas exige responsabilidade!)
- Compartilhamento de páginas (shmem e mmap)
- Memória compartilhada distribuída

# Questões de implementação

- Decisões de mais baixo nível
  - Tratamento e recuperação das faltas de páginas
  - Memória secundária

# Tratamento de falta de página

- O hardware gera uma interrupção (trap) e desvia a execução para o núcleo.
- Registradores são salvos.
- SO identifica página virtual necessária
- SO verifica se o acesso é válido
- ...

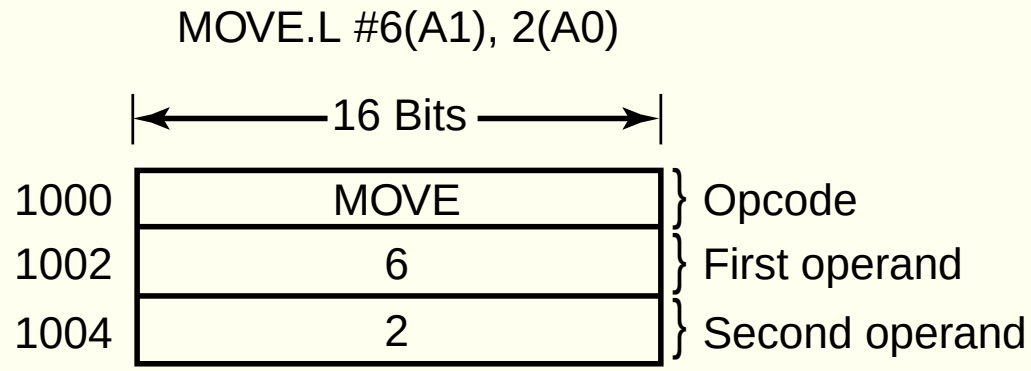
# Tratamento de falta de página

- SO verifica se há um page frame disponível, carrega a nova página e atualiza as tabelas
  - caso não haja, o algoritmo de substituição de páginas é executado
  - caso a página para substituição tenha sido modificada, esta deve ser transferida para o disco. Após esta operação, é que a nova página será carregada neste page frame. Neste intervalo, outro processo pode executar.
- ...

# Tratamento de falta de página

- Instrução causadora da falta de página é recuperada
- Processo causador da falta é escalonado
- Execução é retomada como se nada tivesse acontecido

# Recuperação de instrução




Qual parte da instrução causou a falta de página?

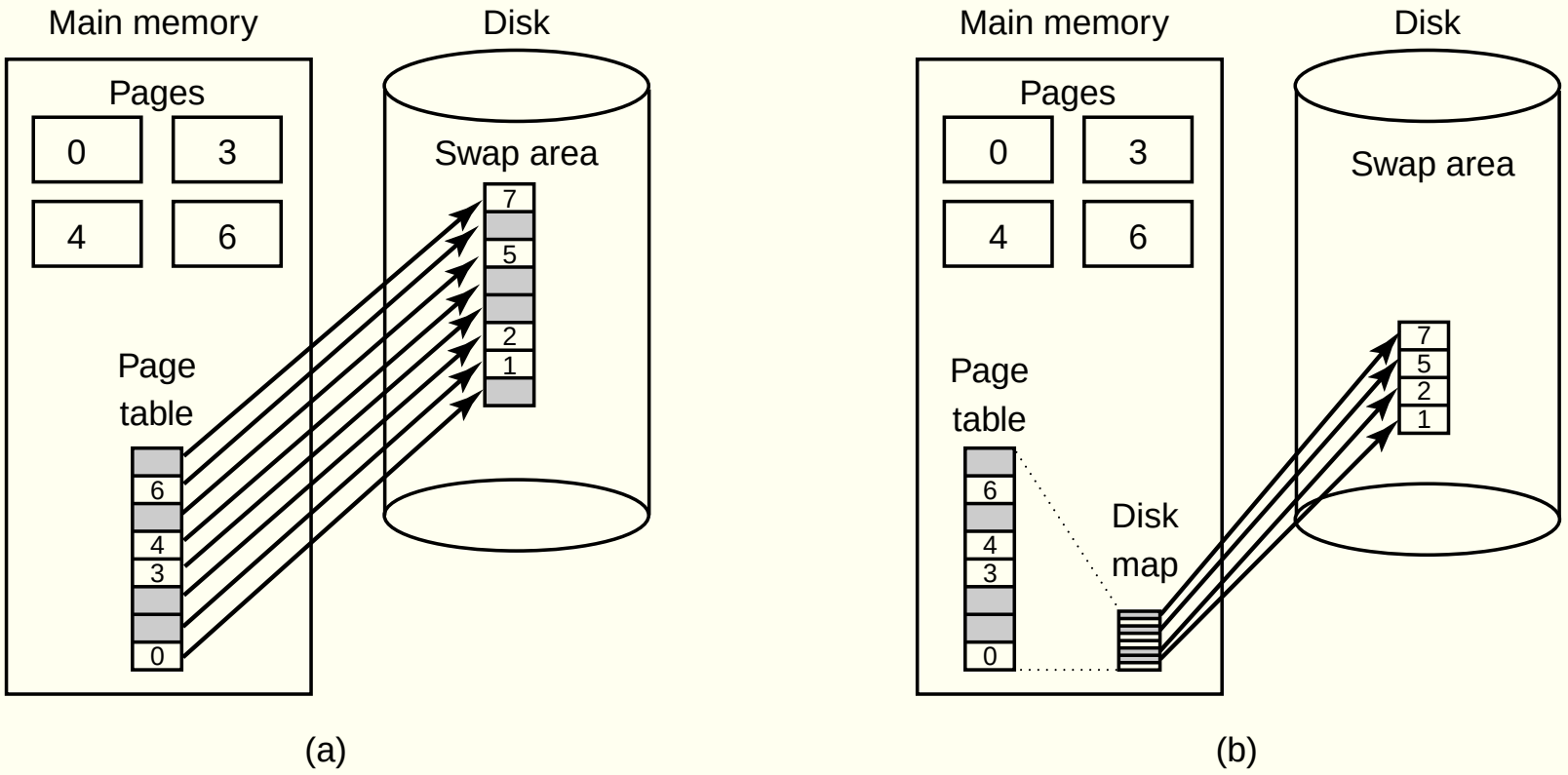
# Fixação de páginas na memória

- Melhora do desempenho para processos com características especiais

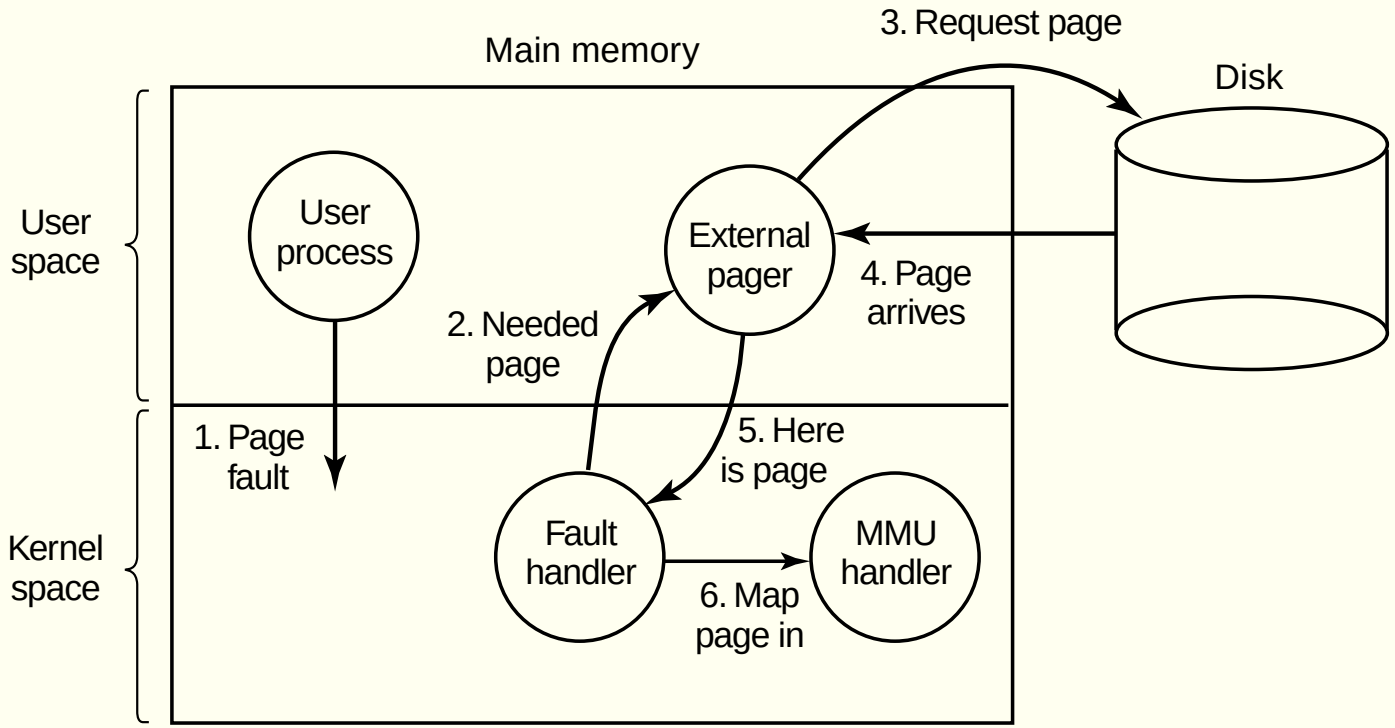
```
int mlock (const void *ADDR, size_t LEN);
int munlock (const void *ADDR, size_t LEN);
```

- Só para super-usuários? :-(

# Memória secundária



# Paginador externo

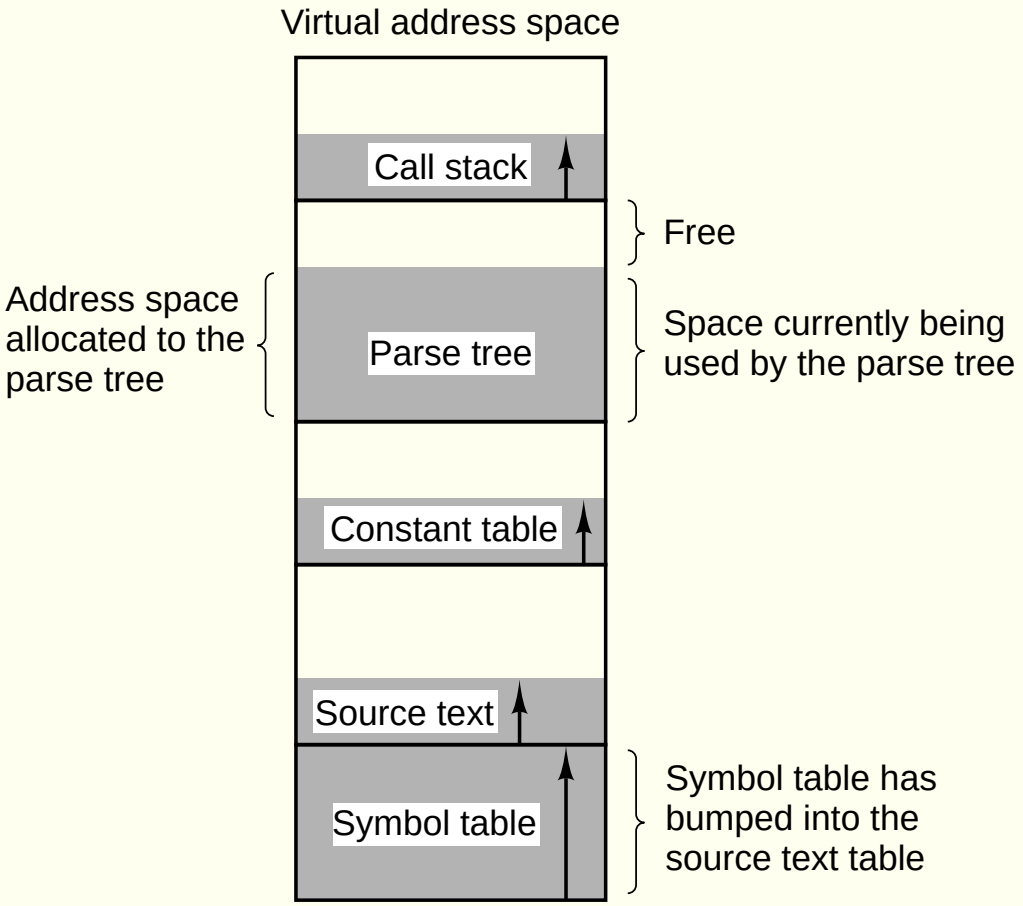



Flexibilidade versus Overhead

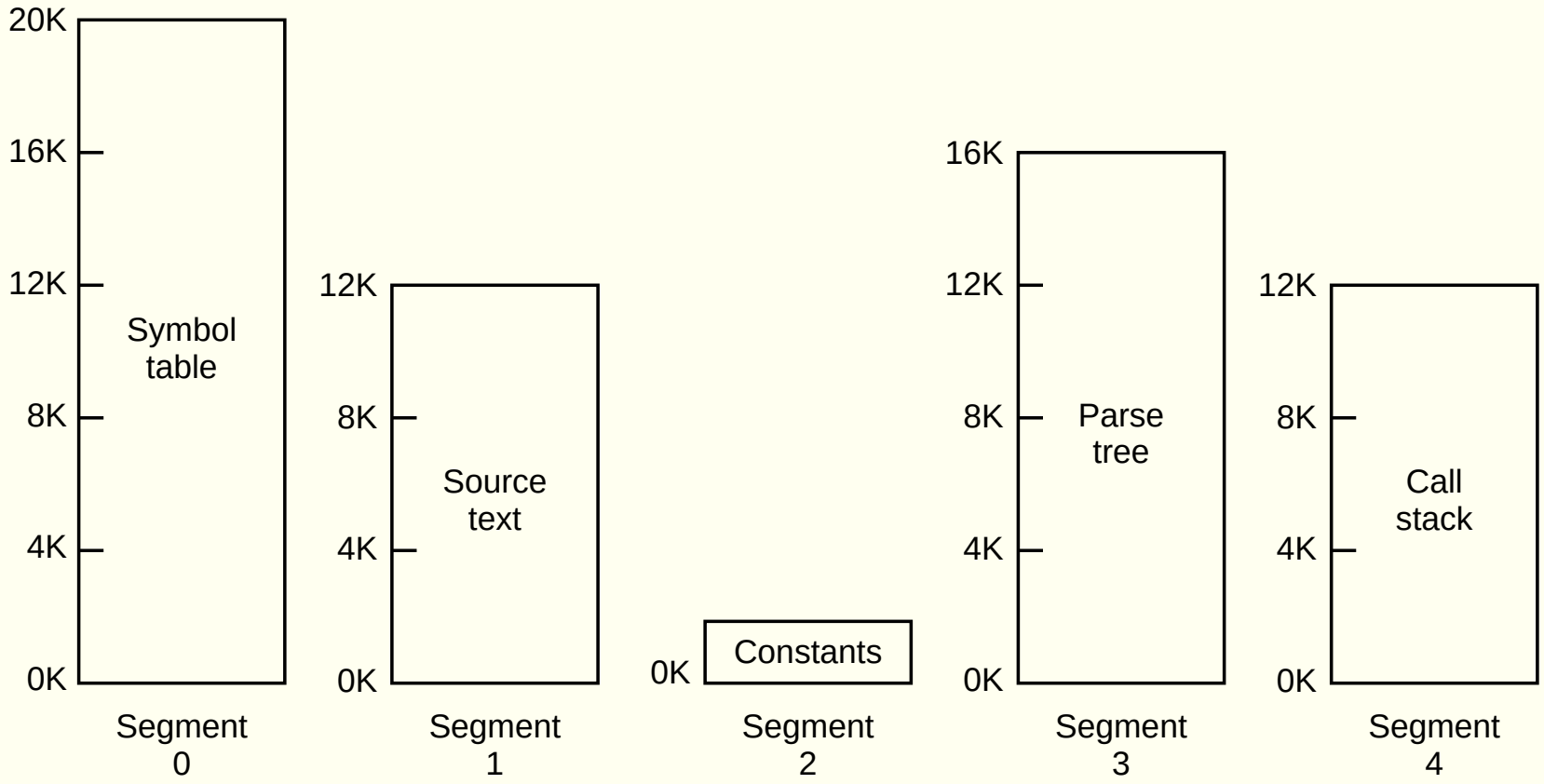
# Segmentação

- Vários espaços de endereçamento
- Maior flexibilidade

# Limites de um espaço único

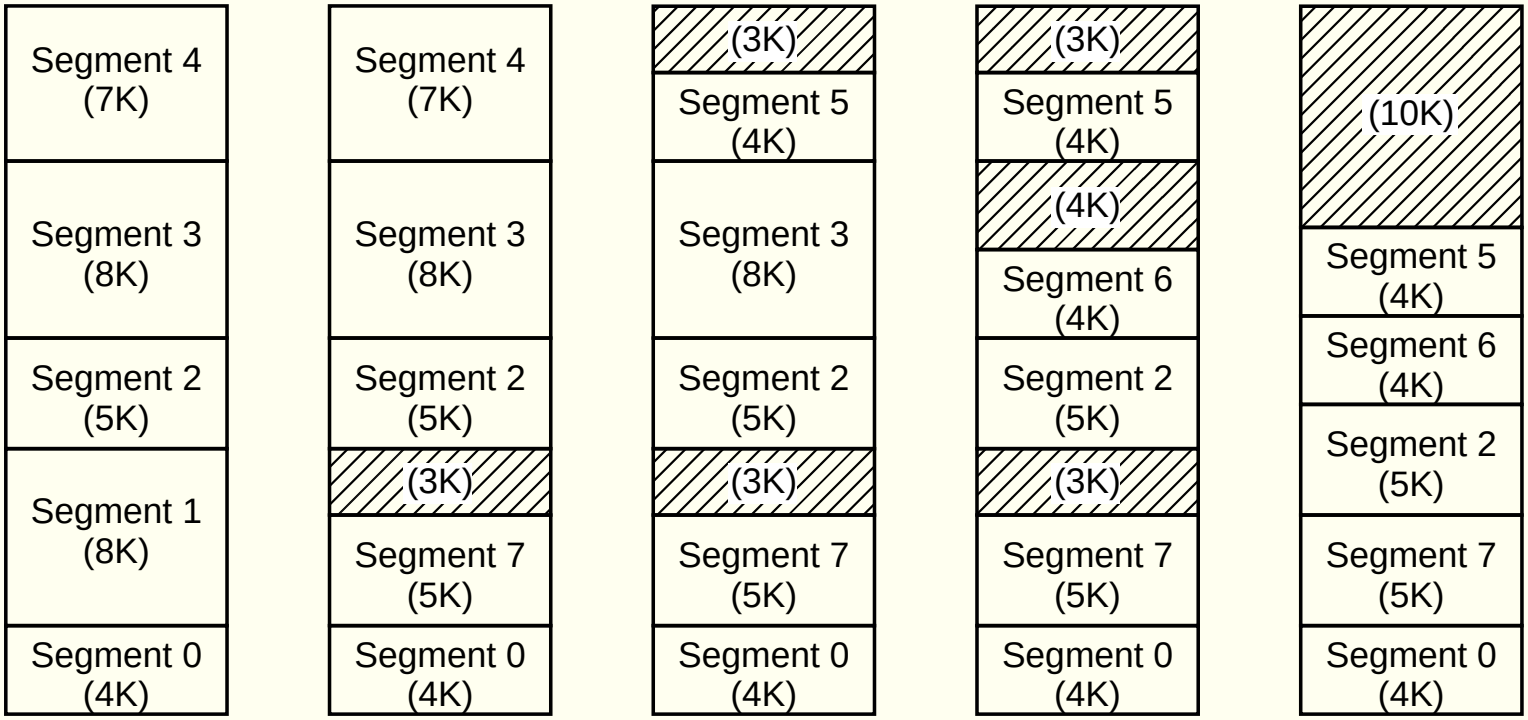


# Segmentação

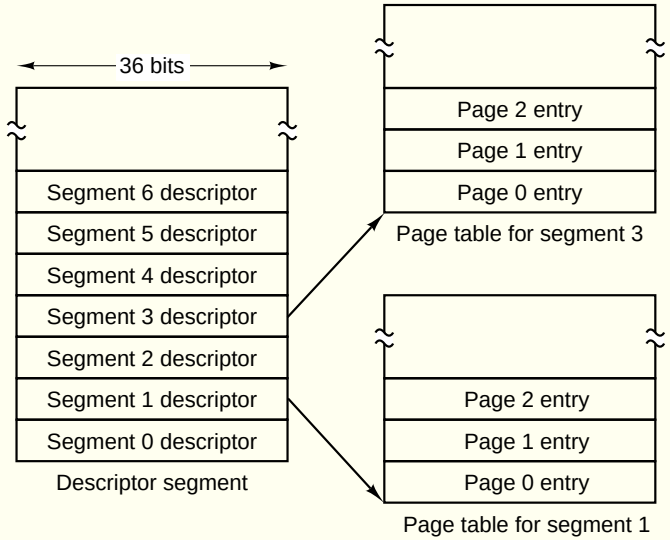


| Consideration | Paging | Segmentation |
|---|---|---|
| Need the programmer be aware that this technique is being used? | No | Yes |
| How many linear address spaces are there? | 1 | Many |
| Can the total address space exceed the size of physical memory? | Yes | Yes |
| Can procedures and data be distinguished and separately protected? | No | Yes |
| Can tables whose size fluctuates be accommodated easily? | No | Yes |
| Is sharing of procedures between users facilitated? | No | Yes |
| Why was this technique invented? | To get a large linear address space without having to buy more physical memory | To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection |

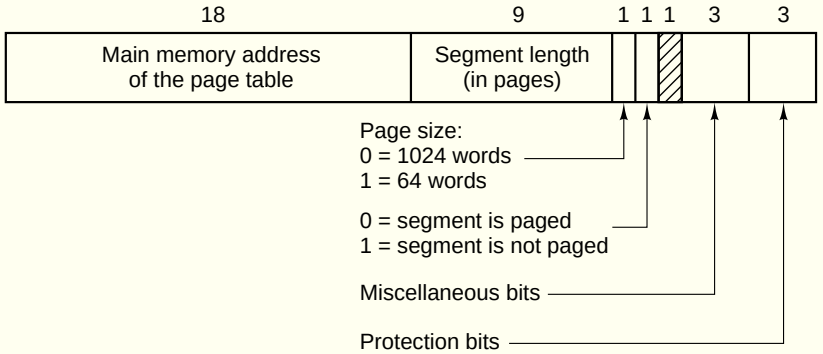
# Segmentação pura



# MULTICS: segmentação e paginação

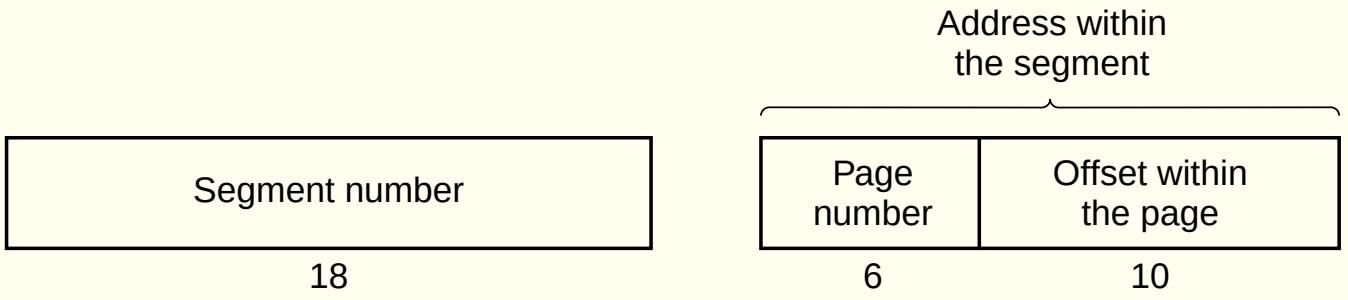



(a)

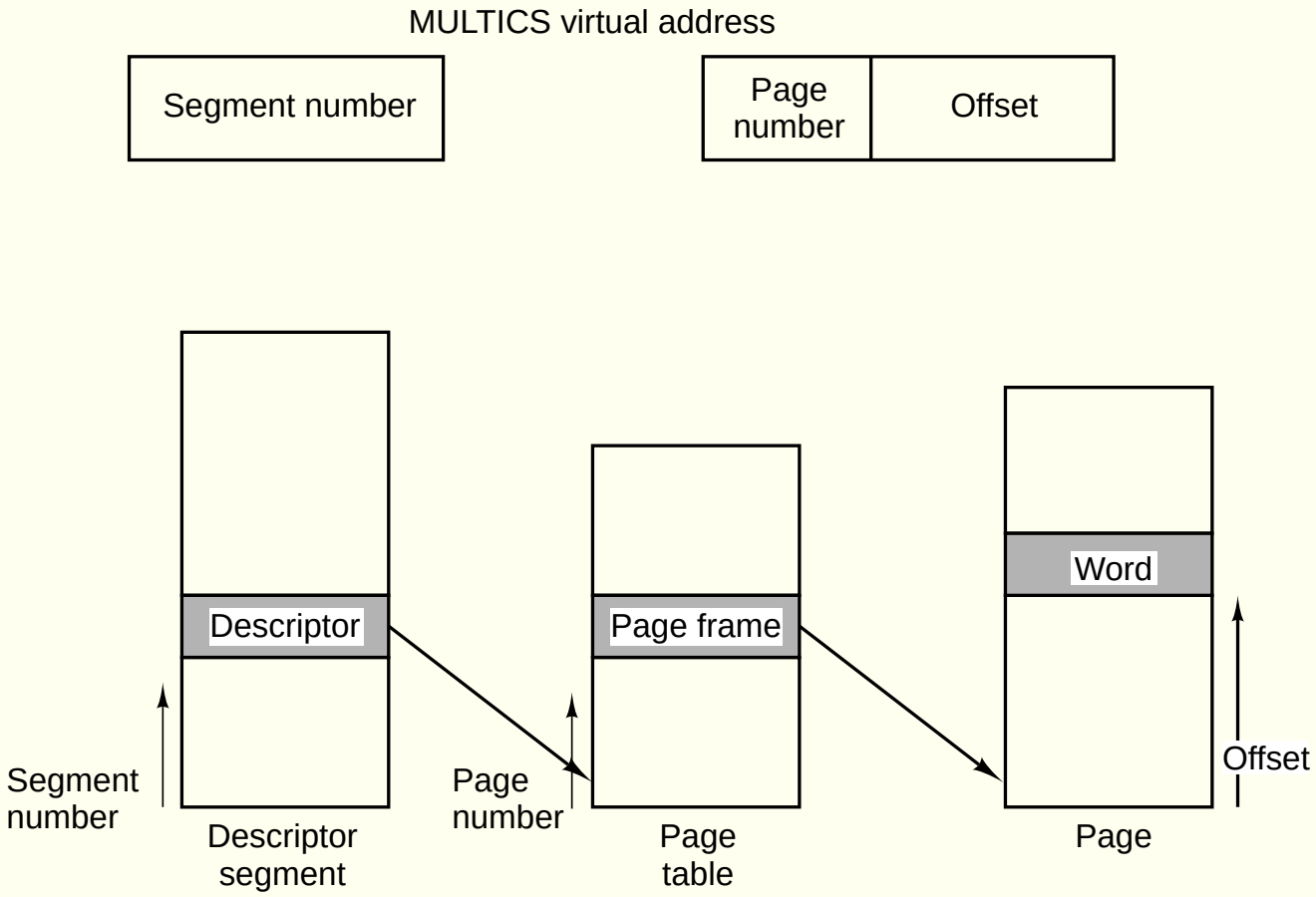



(b)

# MULTICS: endereçamento virtual

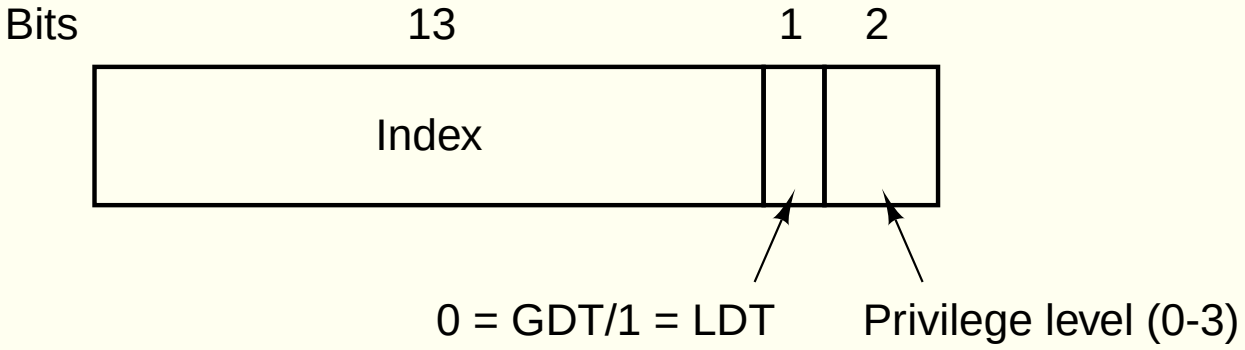


# MULTICS: Conversão de endereços



# Intel Pentium: segmentação e paginação

- Seletor de segmento



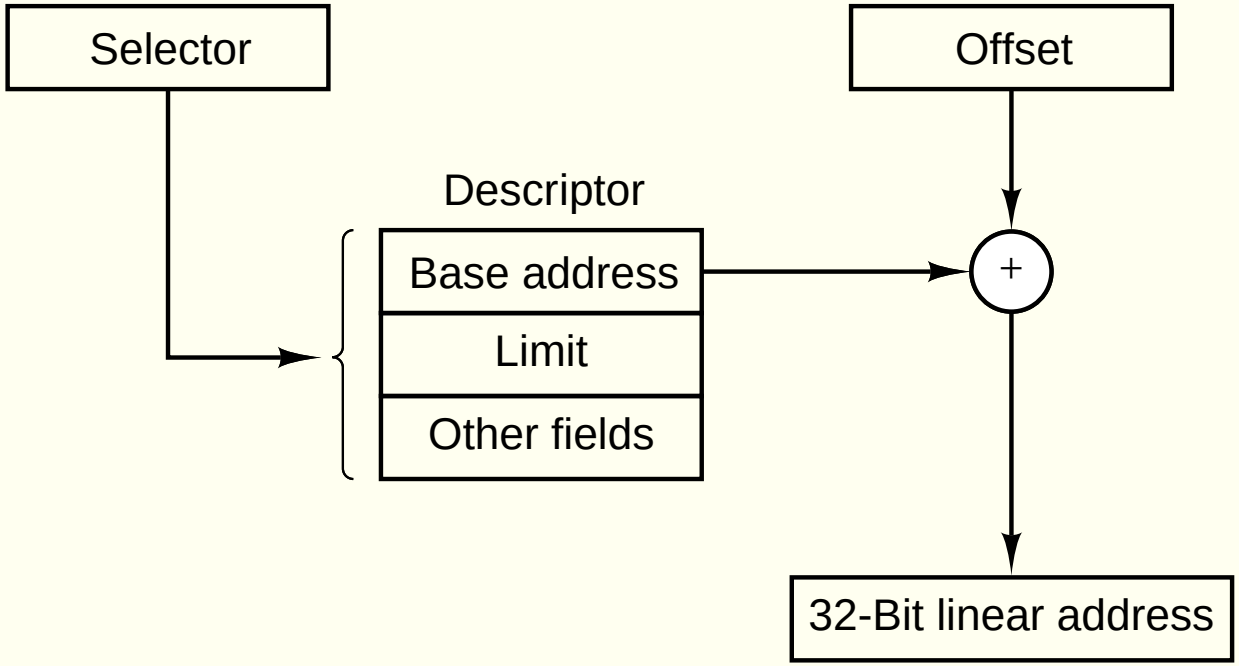

- LDT (Local Descriptor Table)
- GDT (Global Descriptor Table)

# Registradores de segmento

- CS - Code Segment
- DS - Data Segment
- SS - Stack Segment
- ES, FS, GS - Extra Segments

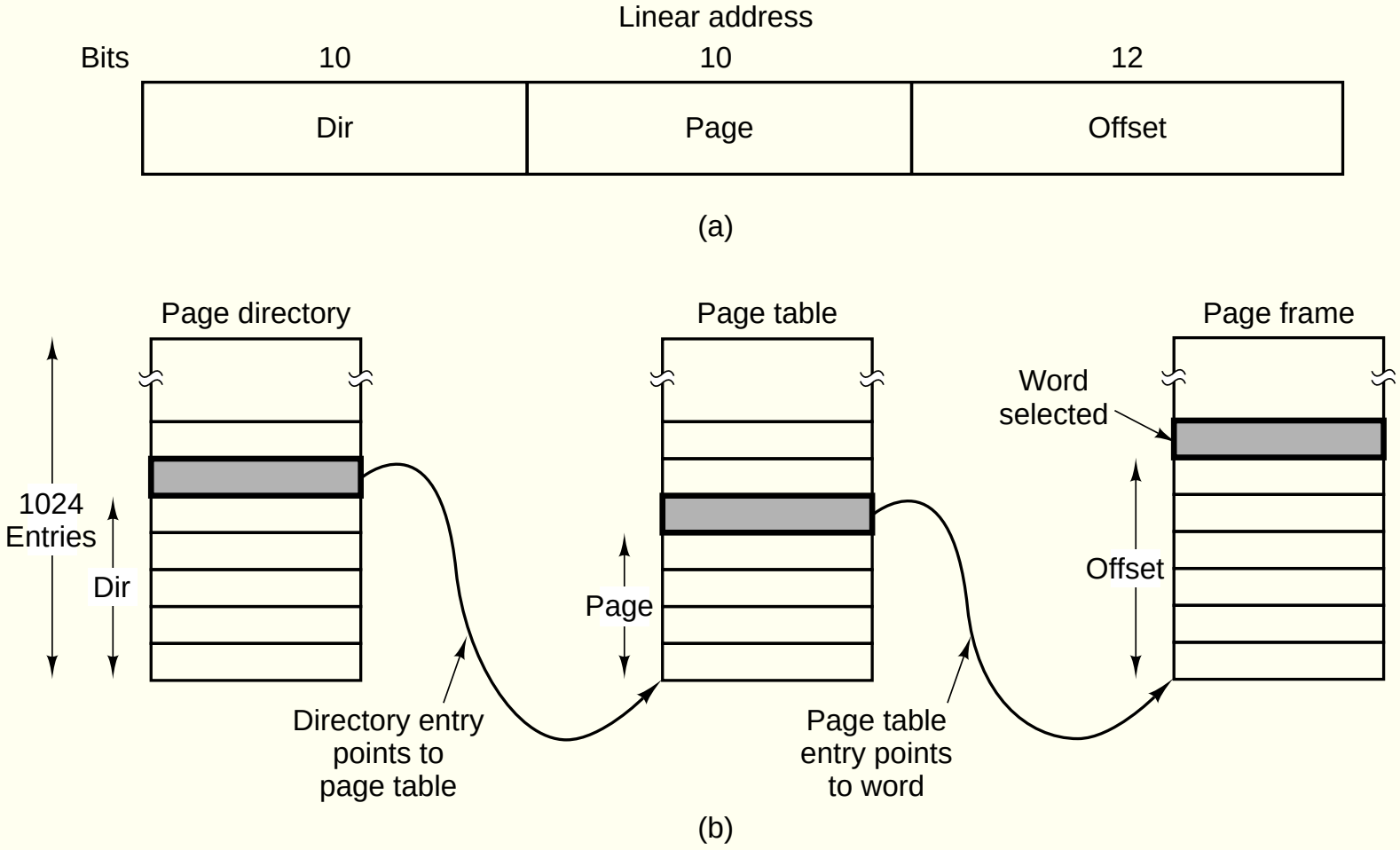
# Conversão de endereços

## (seletor, deslocamento) $\longrightarrow$ linear



# Conversão de endereços

## linear → físico



# Resumo

- Sistemas simples
- Swapping
- Memória virtual
- Paginação
- Segmentação
- Segmentação/paginação