MC514–Sistemas Operacionais: Teoria e Prática

1s2010

Gerenciamento de Memória

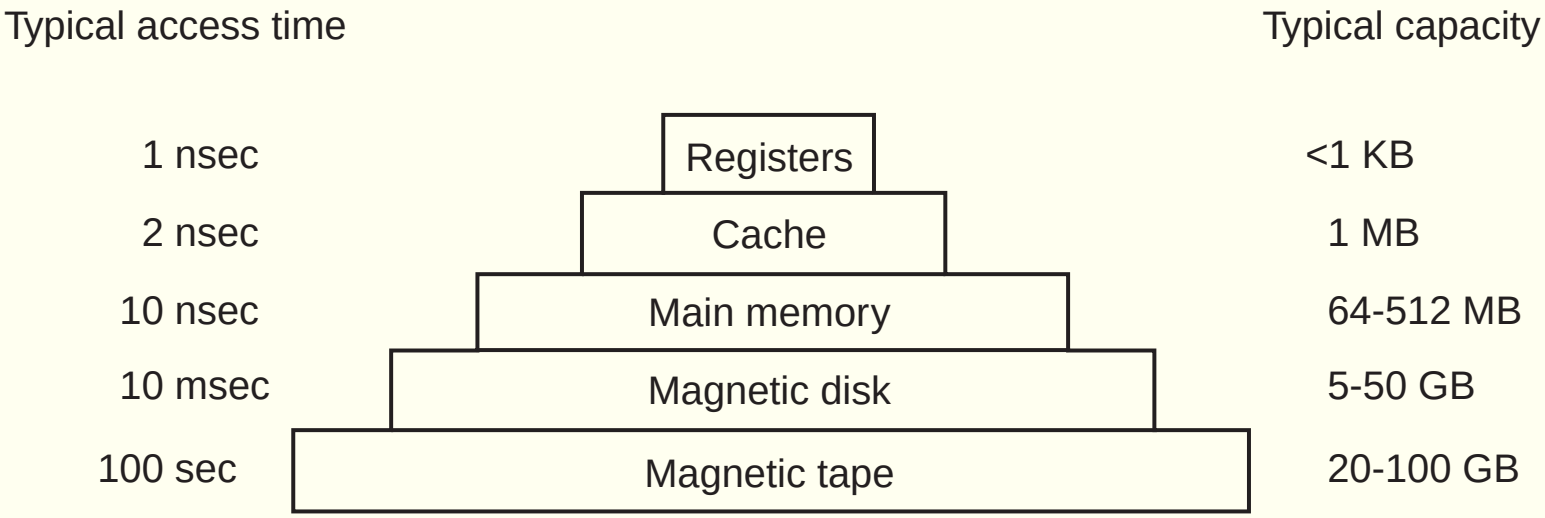
Sonho dos usuários

Idealmente, a memória deveria ser

- rápida,
- de custo baixo,
- imensa e
- não volátil.

Realidade para os usuários

Hierarquia de Memória



Tanenbaum: Figura 1.7

Como estão estes valores hoje?

Hierarquia de Memória

Registradores

- Internos à CPU
- Extremamente rápidos
- Otimizações de código podem mover temporariamente variáveis para registradores.
- Programas podem dar palpites sobre o que deve ficar armazenado nos registradores

  ```
  register int r;
  ```

- Veja o código register.c

Hierarquia de Memória

Cache

- Internos ou muito próximos à CPU
- Divididos em linhas de cache
- Controlados por hardware
- Cache hit
- Cache miss

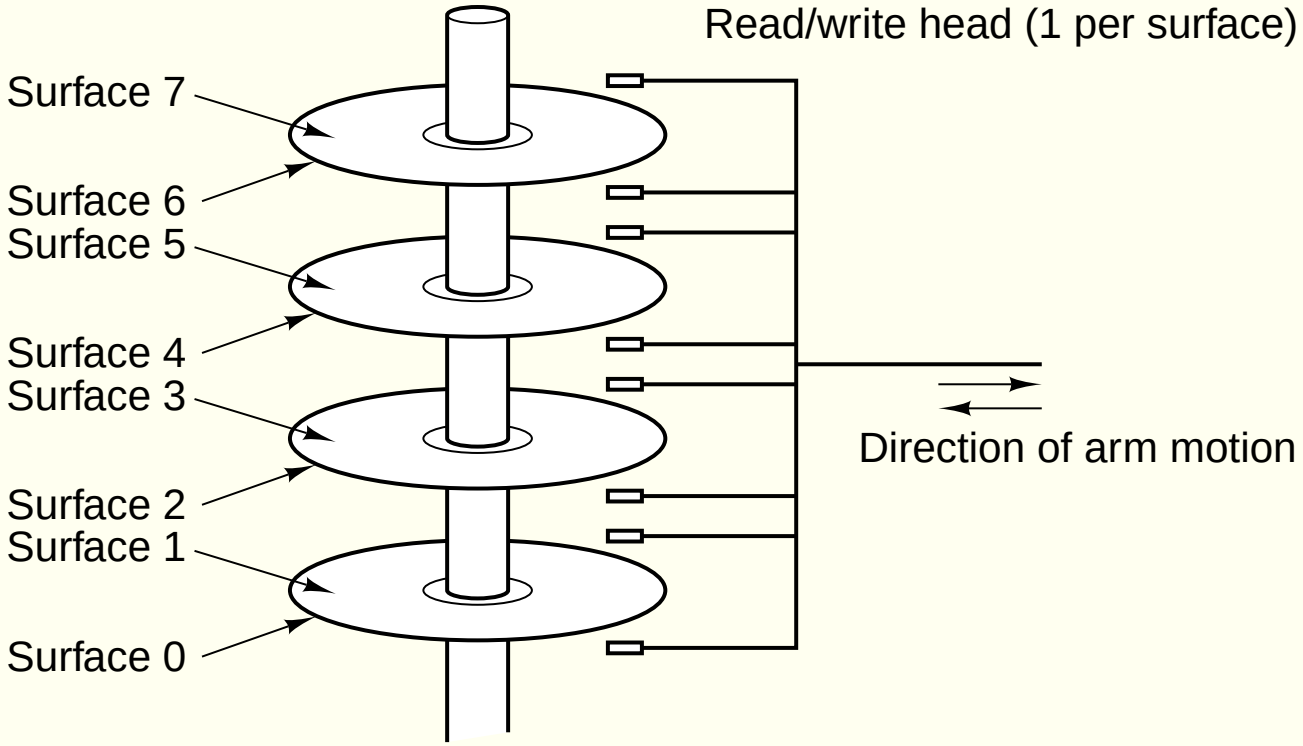
Hierarquia de Memória

Memória Principal

- Random Access Memory (RAM)
- Compromisso entre preço e desempenho
- Armazenamento volátil

Hierarquia de Memória

Disco



Hierarquia de Memória

Fitas magnéticas

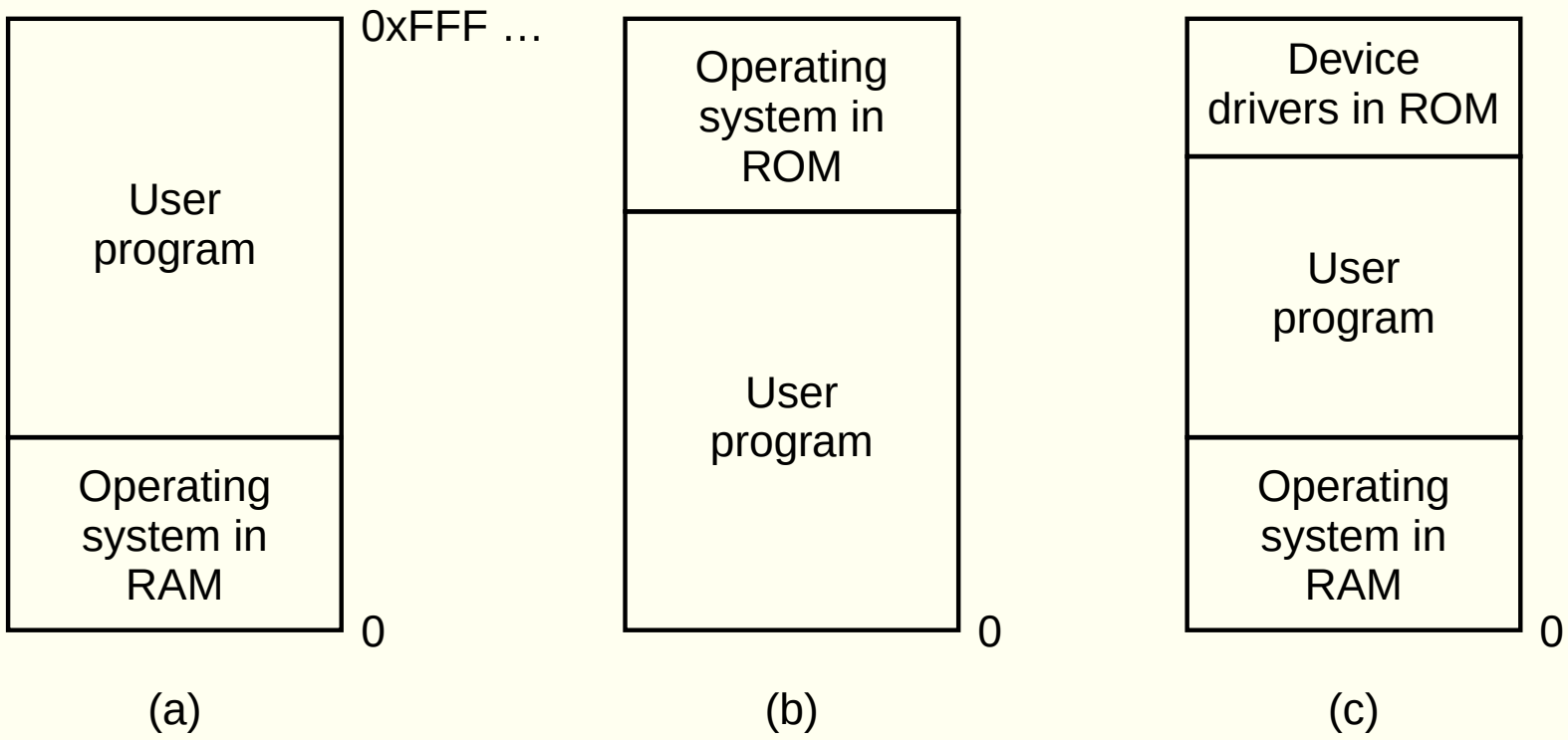
- Utilizadas para cópias de segurança (backups)
- Armazenamento de grandes quantidades de dados
- Acesso seqüencial

Hierarquia de Memória

Outros tipos de memória

- ROM (Read Only Memory)
  - rápida e barata
  - bootstrap loader está gravado em ROM
- EEPROM (Electrically Erasable ROM)
  - podem ser apagadas (erros podem ser corrigidos)
- CMOS
  - dependem de uma bateria
  - armazenam relógio e configurações

Monoprogramação

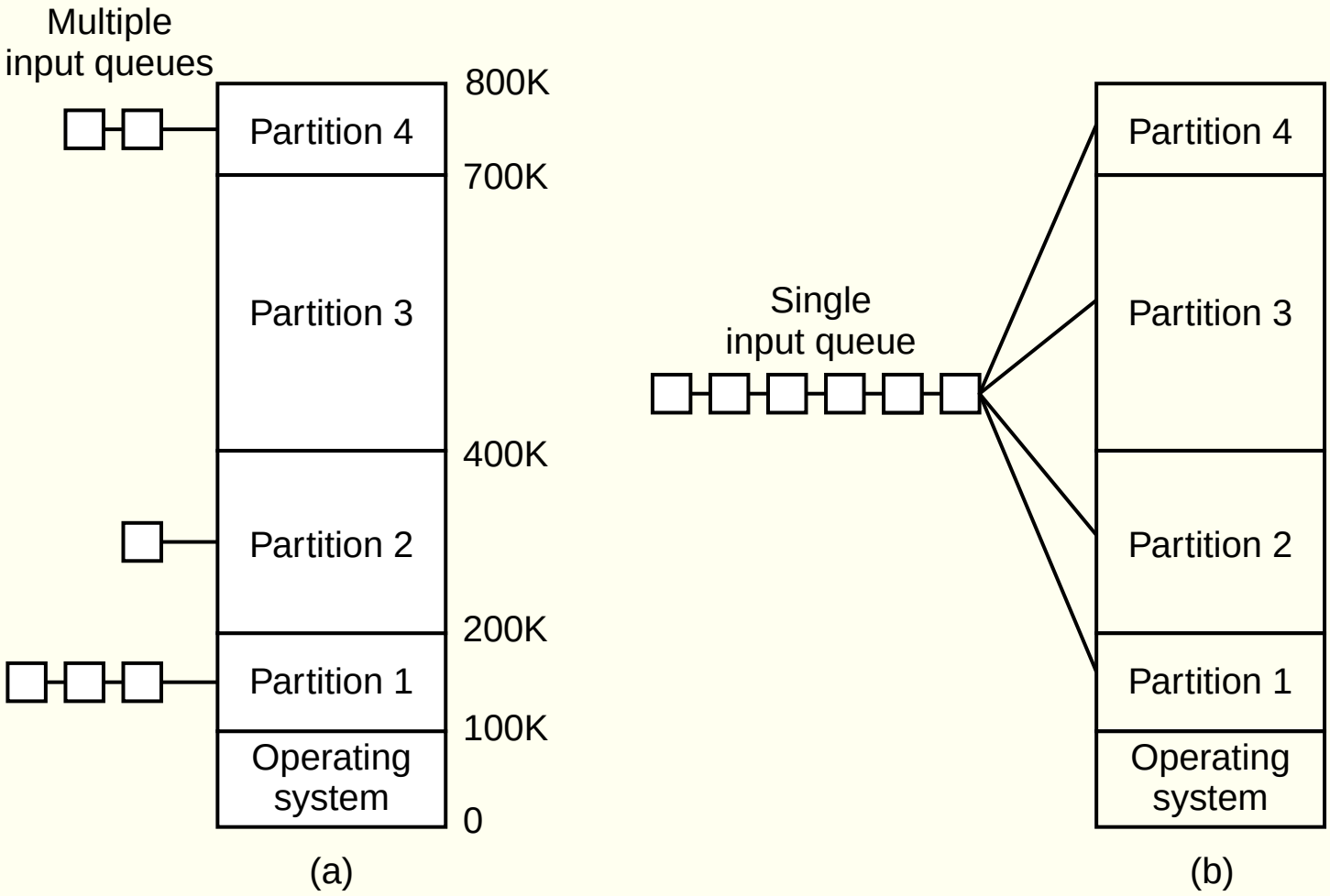


mainframes

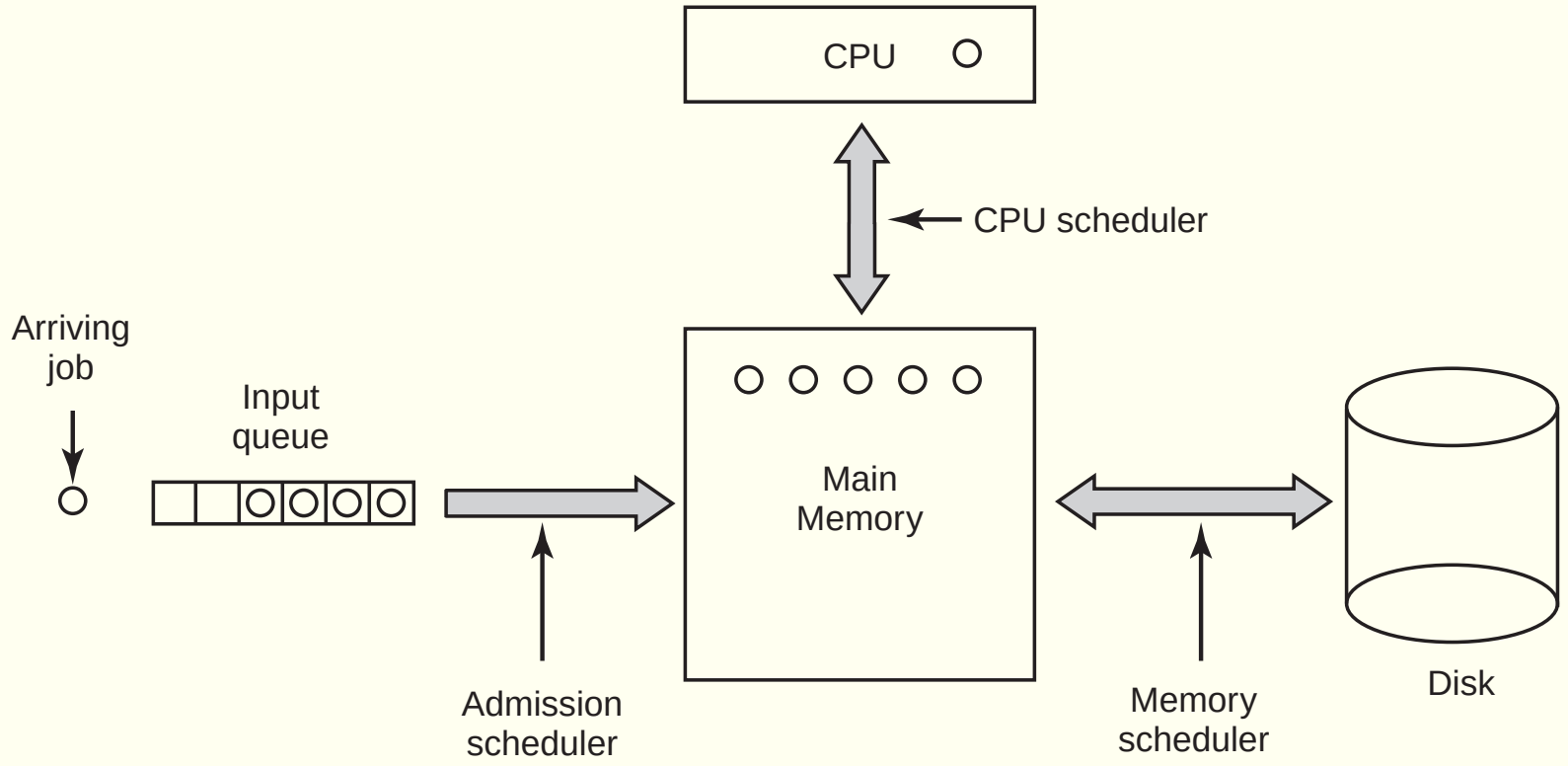
palmtops

MS-DOS

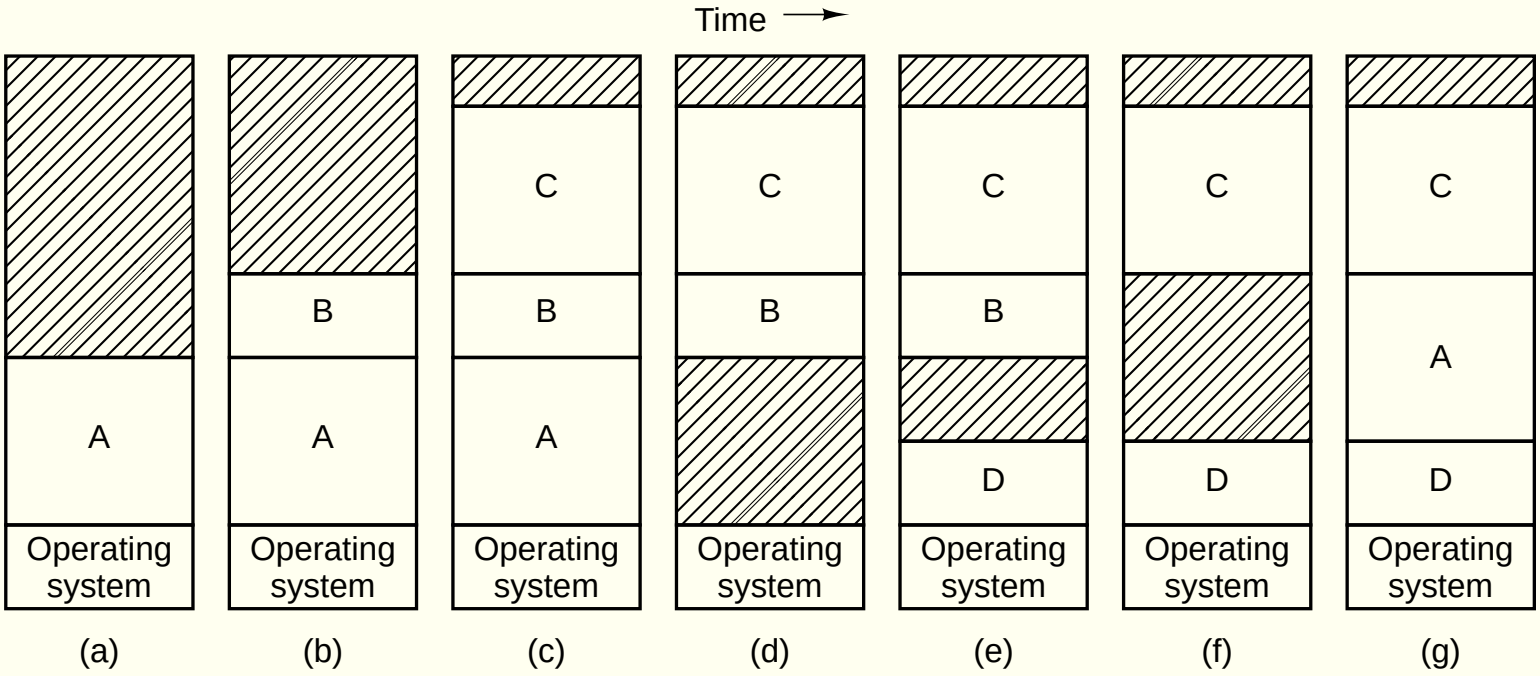
Multiprogramação e partições fixas



Swapping


CPU
CPU scheduler
Arriving job
Input queue
Main Memory
Admission scheduler
Memory scheduler
Disk

Swapping



Relocação e Proteção

- Relocação: um programa deve poder rodar em endereços físicos distintos.
- Proteção: um programa não pode fazer acesso à área de memória reservada a outro programa.

Relocação durante a carga

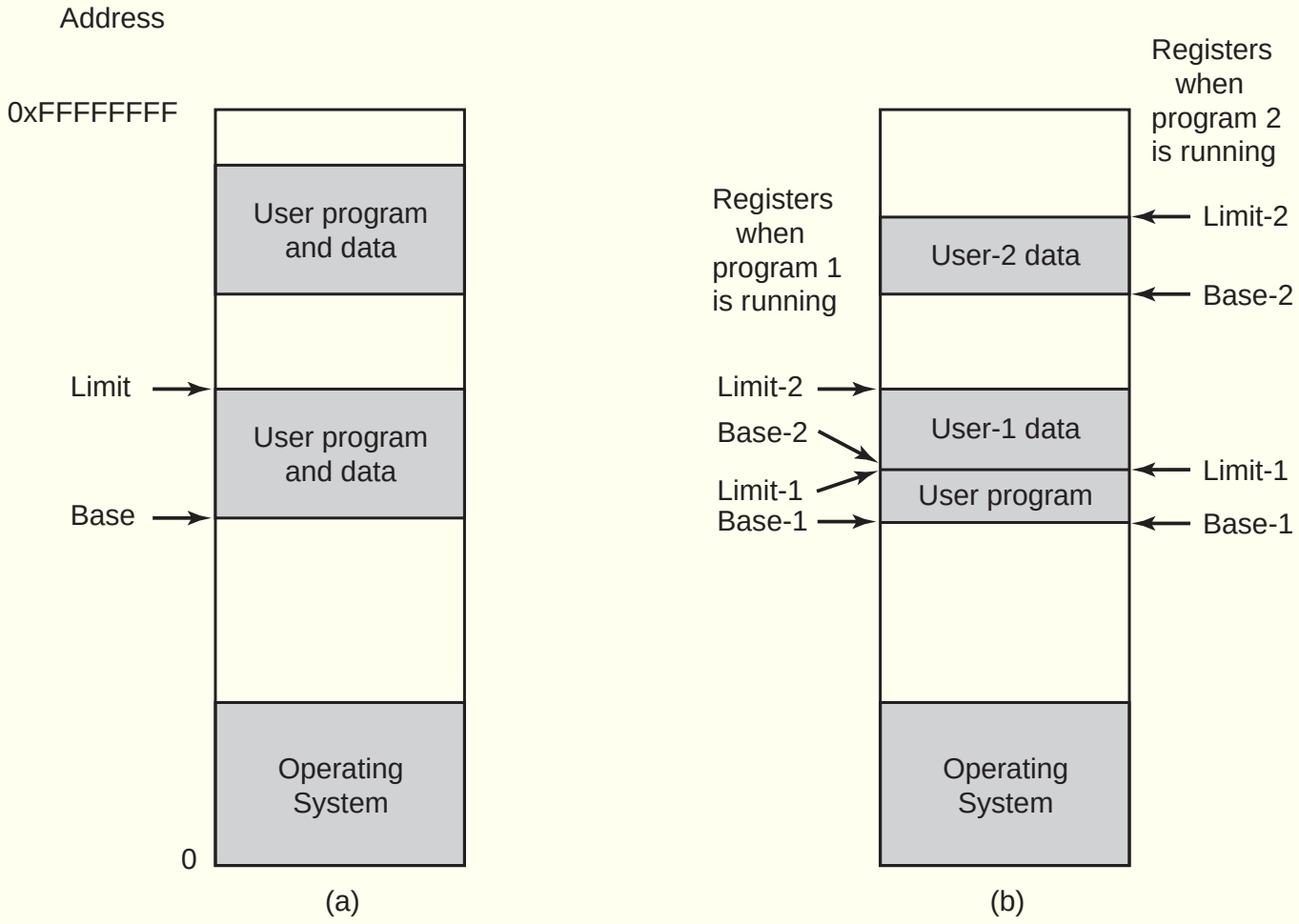
- Todos os endereços precisam ser identificados e alterados
- Não resolve o problema da proteção

Bits de proteção

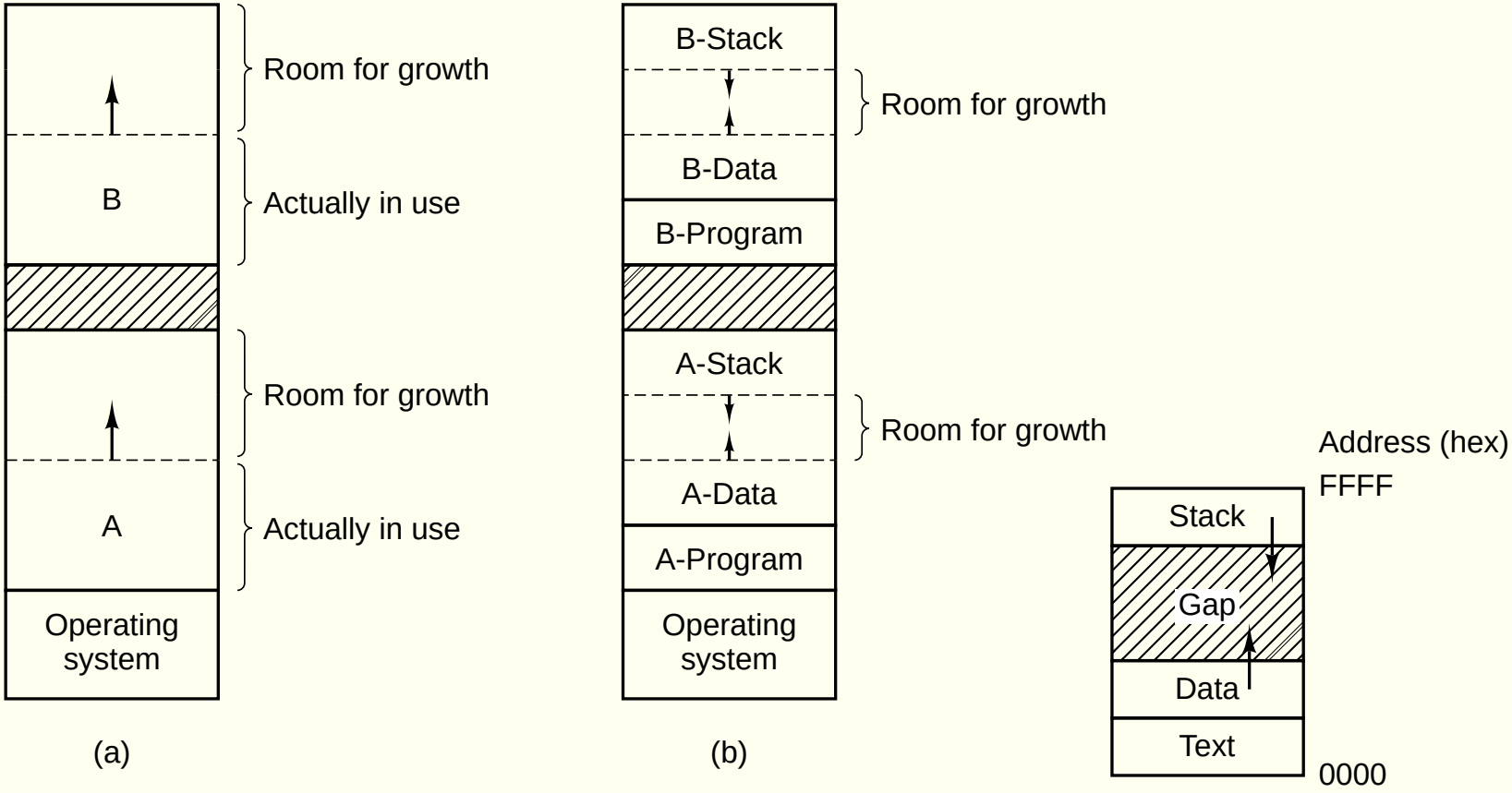
1010	
1010	
1010	
0011	
0011	

O PSW de cada processo deve conter os bits de proteção

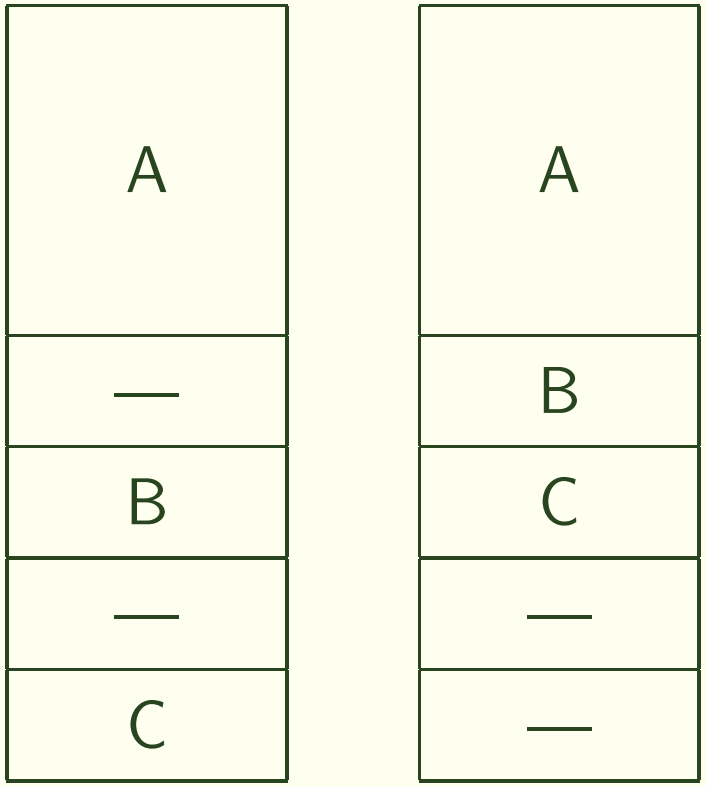
Registradores base e limite



Espaço para crescimento



Compactação de memória



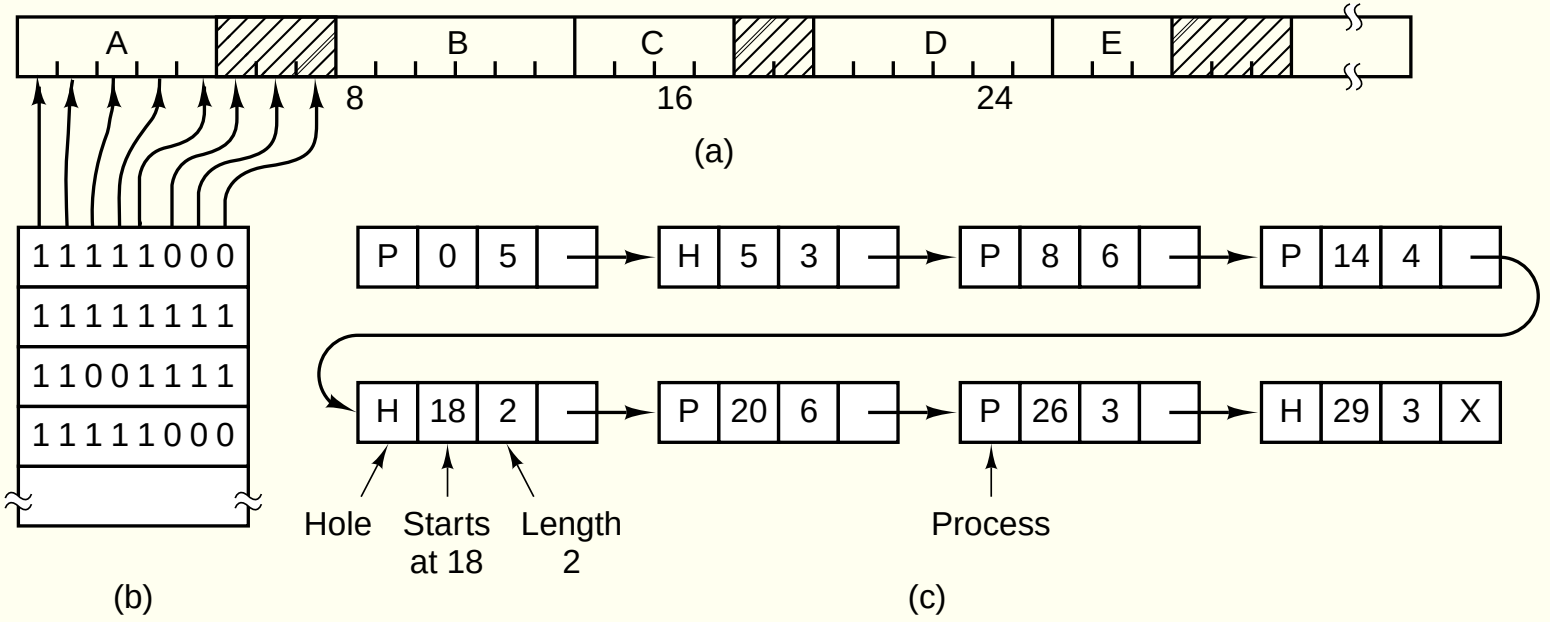
Gerência de grandes blocos de memória

Problema relacionado

Malloc, free e realloc

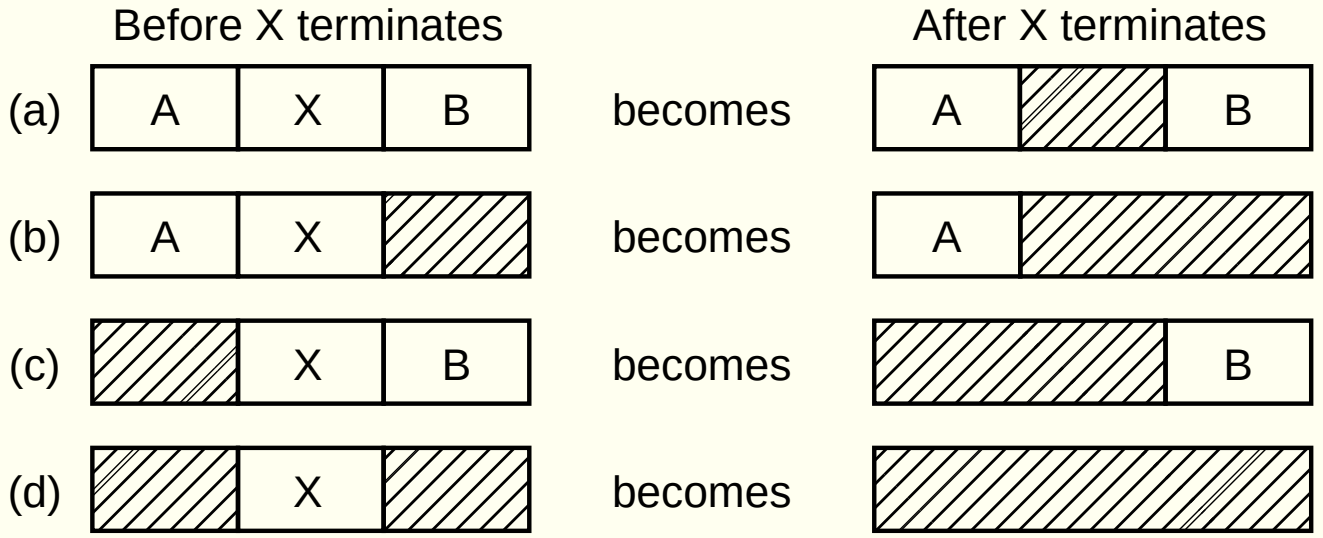
```
void *malloc(size_t size);
void free(void *ptr);
void *realloc(void *ptr, size_t size);
```

Bitmaps e lista de livres



Tanenbaum: Figura 4.7

Atualização da lista



Tanenbaum: Figura 4.8

Veja os códigos: erro_malloc.c erro_free.c

Algoritmos para alocação de memória

- *First fit*
- *Next fit*
- *Best fit*
- *Worst fit*
- Veja o código fit.c

Programas muito grandes

O que fazer se um programa for muito grande para caber na memória?

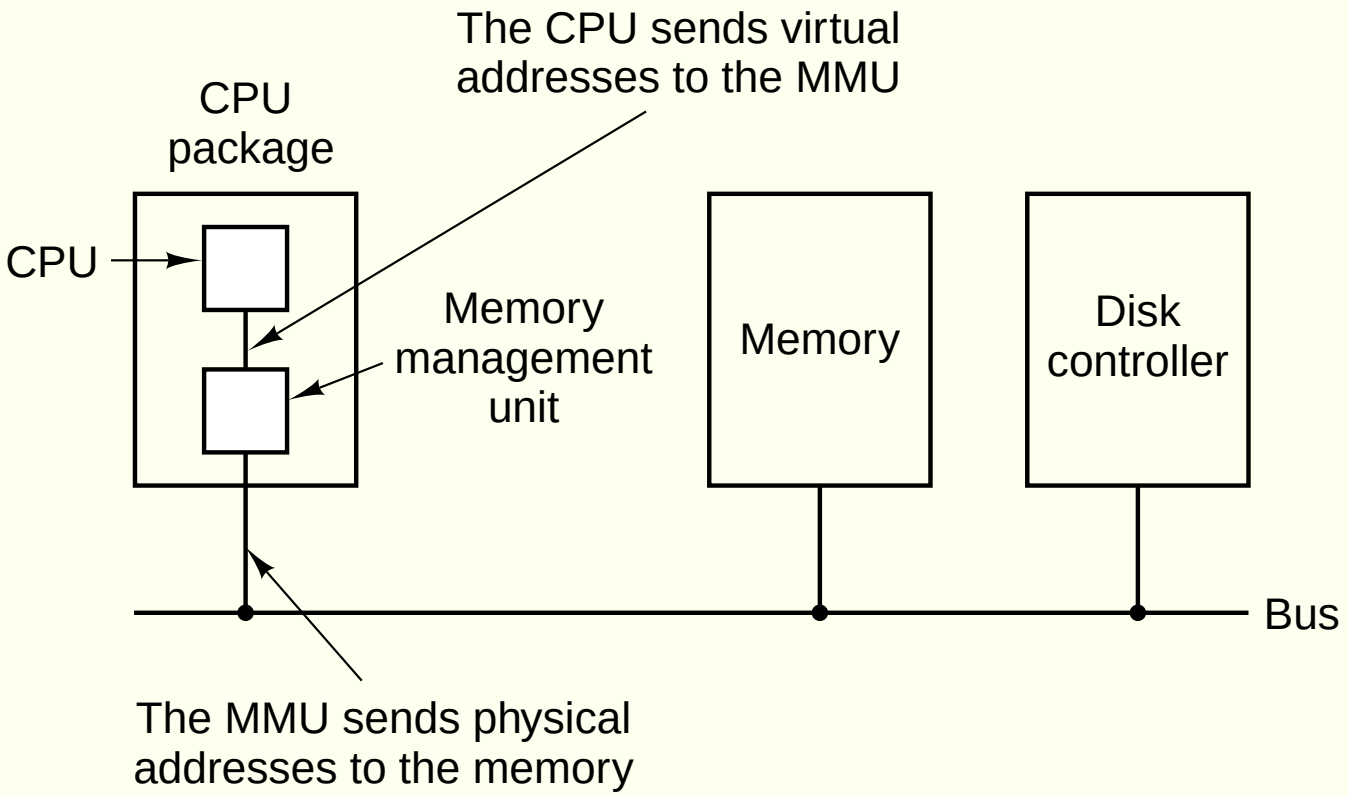
Overlays

```
dados d1, d2, d3, d4, d5;

f1();    g1();   h1();
f2();    g2();   h2();
f3();    g3();   h3();

main() {
  fase_1();  /* funçoes f e dados d1, d2, d3 */
  fase_2();  /* funçoes g e dados d1, d2, d4 */
  fase_3();  /* funçoes h e dados d1, d2, d5 */
  }
```

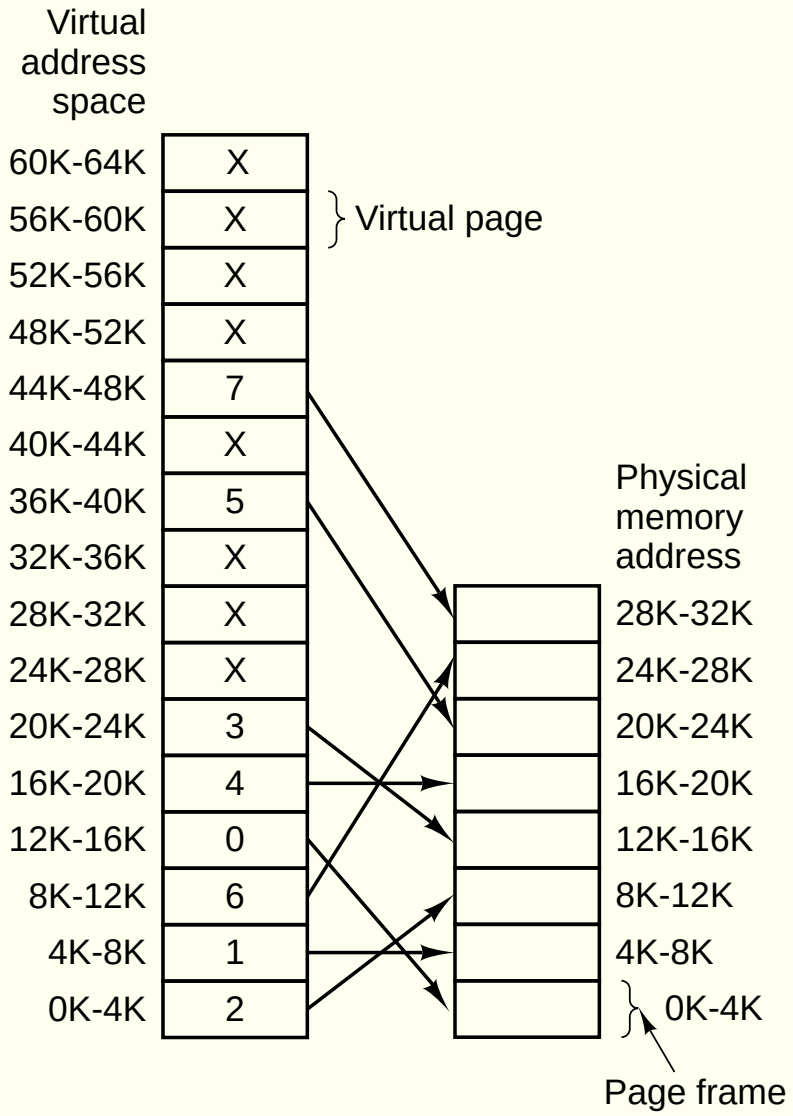
Memória Virtual



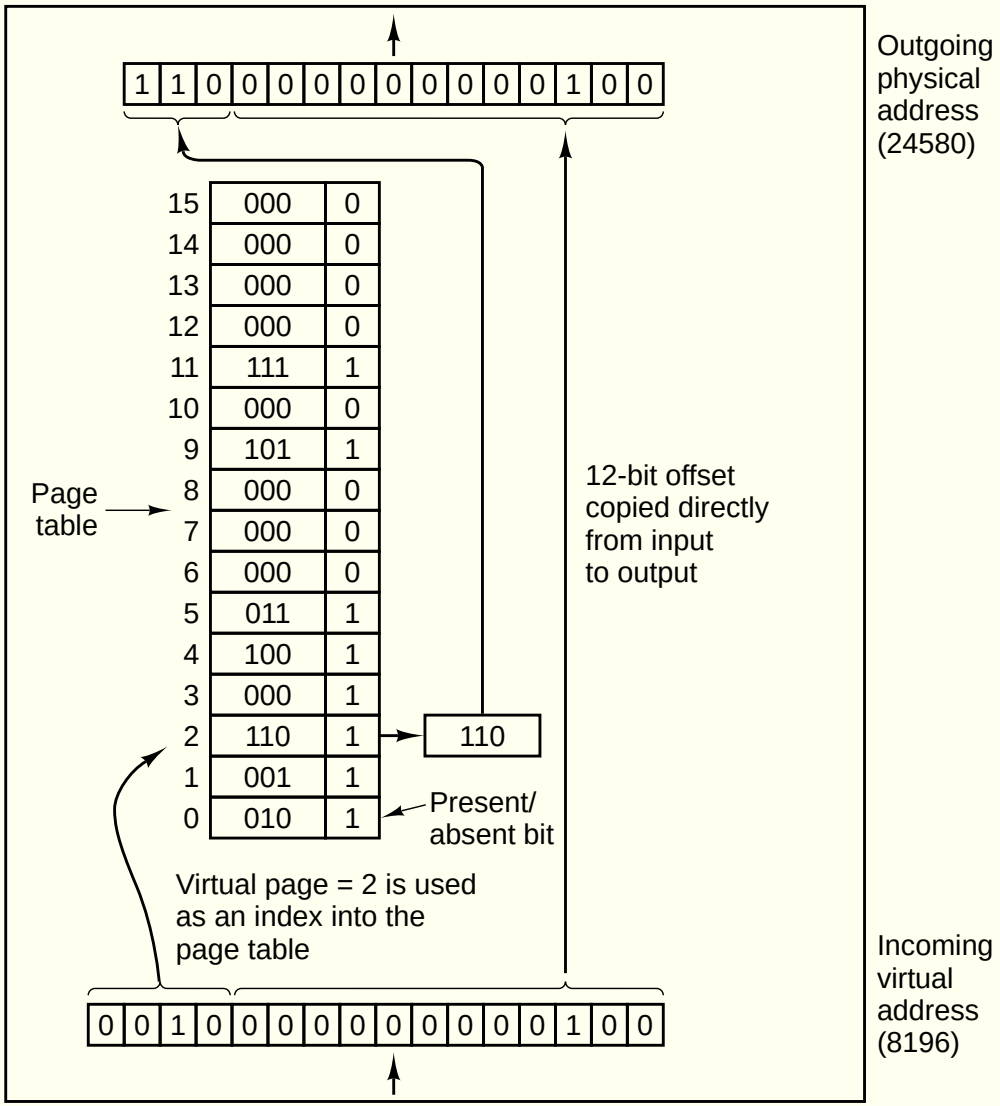
Tanenbaum: Figura 4.9

Memory Management Unit (MMU)

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

Mapeamento dos endereços

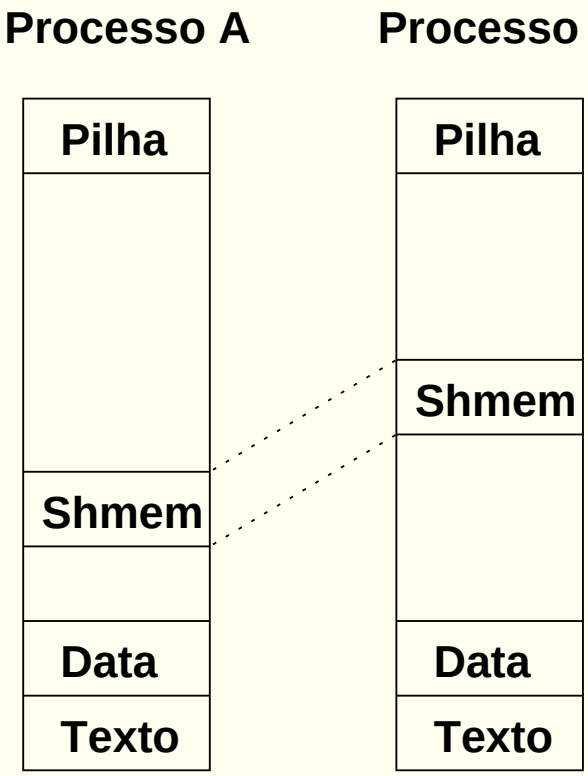


Paginação - Exemplo

- 32 bits de endereço
- páginas de 4k
- 20 primeiros bits indicam a página
- 12 últimos bits indicam o deslocamento dentro da página

- Veja o código pagesize.c

Memória compartilhada



Memória compartilhada

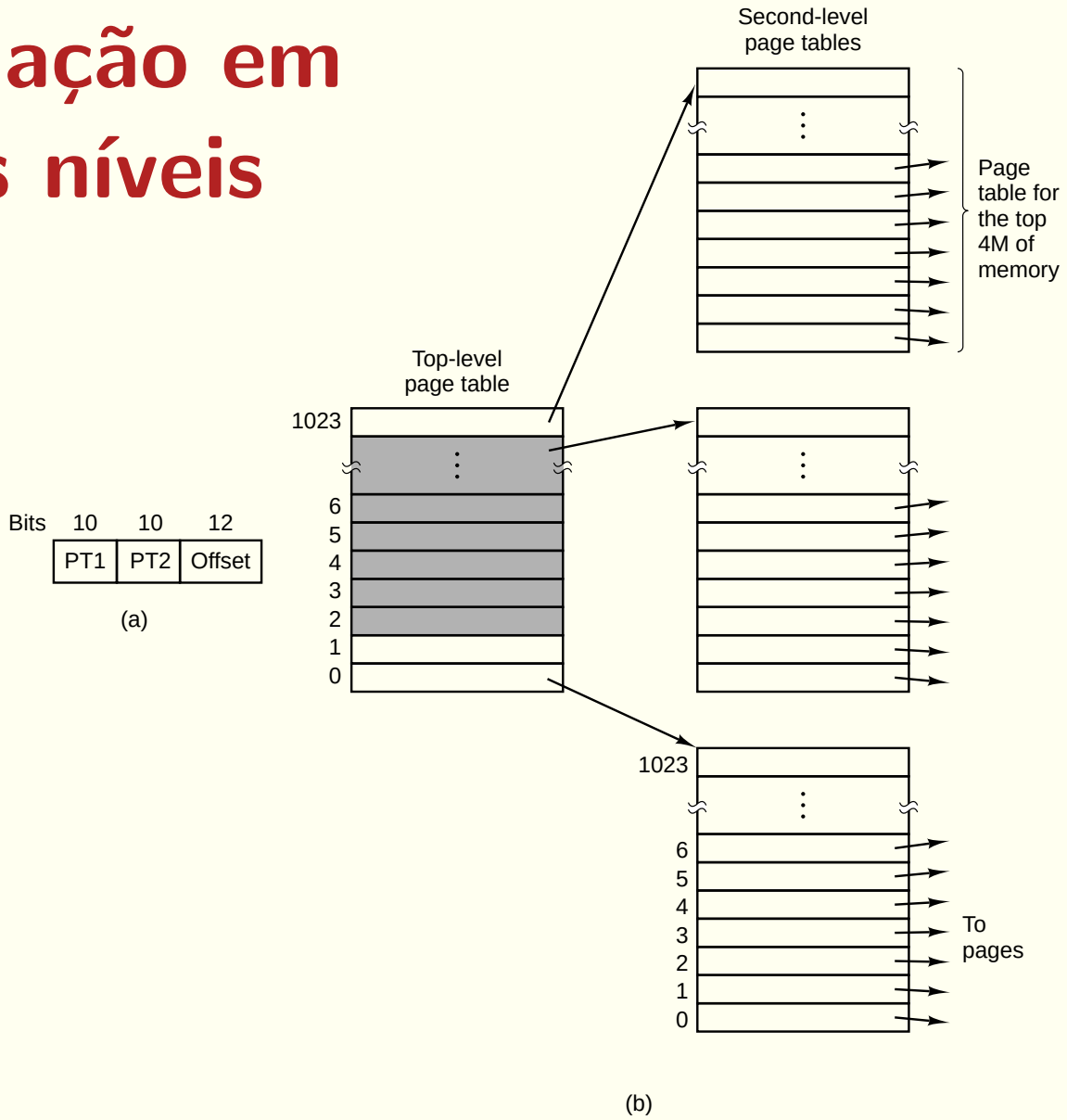
```
int shmget(key_t key, size_t size, int shmflg);
void *shmat(int shmid,
            const void *shmaddr, int shmflg);
```

- Veja os exemplos: sh1.c sh2.c sh_fork.c sh_server.c e sh_client.c

Mmap

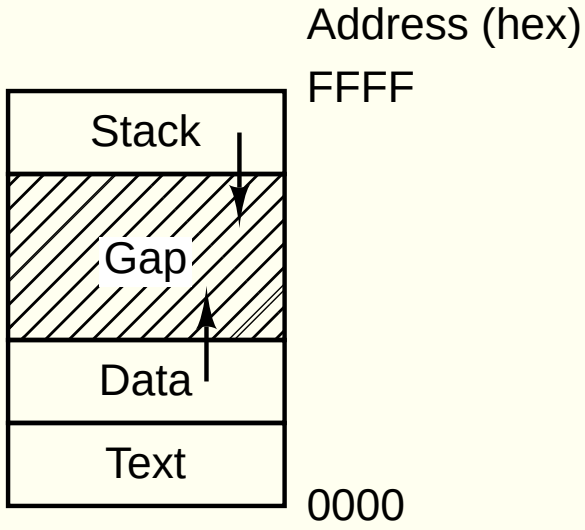
- Um arquivo pode ser mapeado em memória
- Memória compartilhada
- Veja map.c map2.c

Paginação em vários níveis

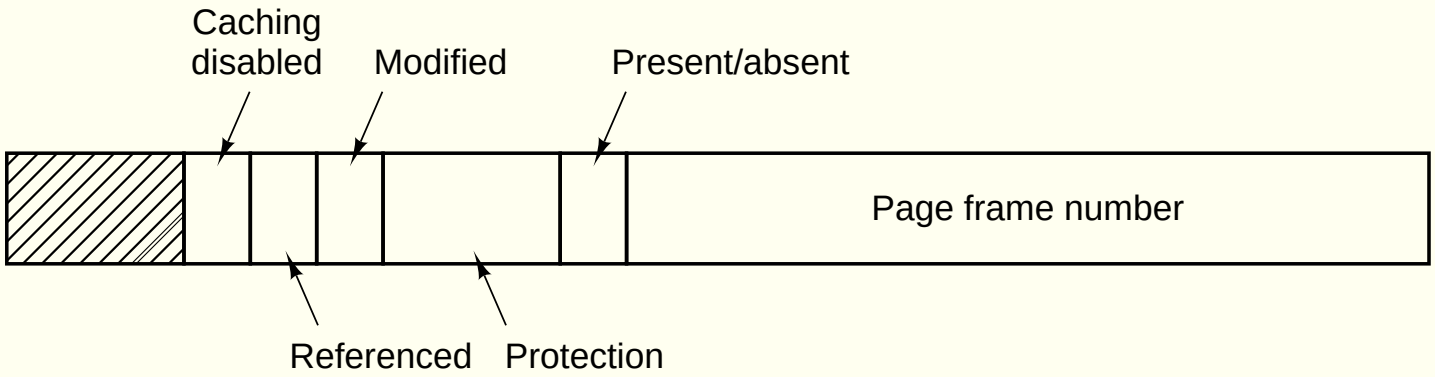


Espaço de endereçamento

- Apenas as páginas ocupadas precisam ser mapeadas
- Veja o código sbrk.c



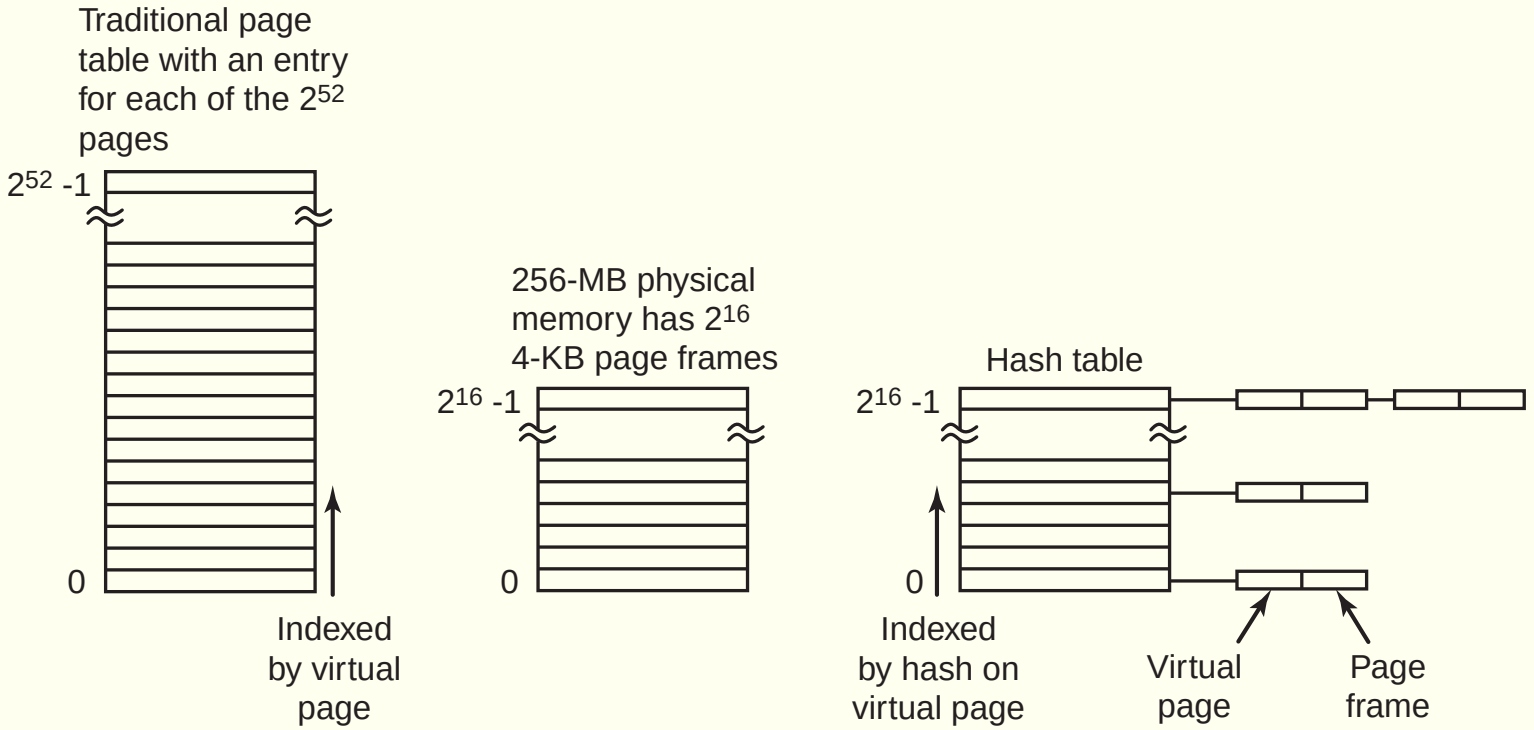
Entrada na tabela



Translation Look Aside Buffers (TLBs)

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

Tabela de páginas invertida



- Uma entrada por página na memória física

Substituição de páginas

- Veja os códigos: pag1.c pag2.c

Algoritmo ótimo:

- Baseado no uso futuro de uma página
- Impossível de ser implementado
- Pode ser simulado (segunda execução do mesmo processo com a mesma entrada)
- Útil para medidas de desempenho

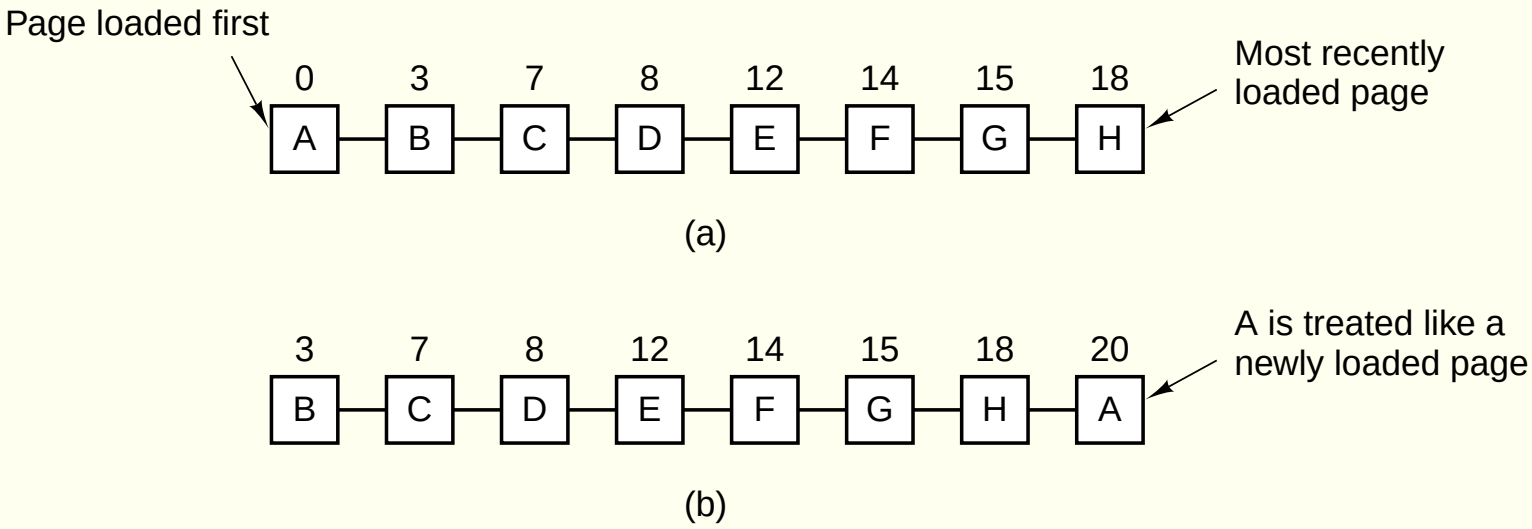
Não usada recentemente

- Classe 0: não referenciada, não modificada
- Classe 1: não referenciada, mas modificada
- Classe 2: referenciada, mas não modificada
- Classe 3: referenciada e modificada

First In, First Out

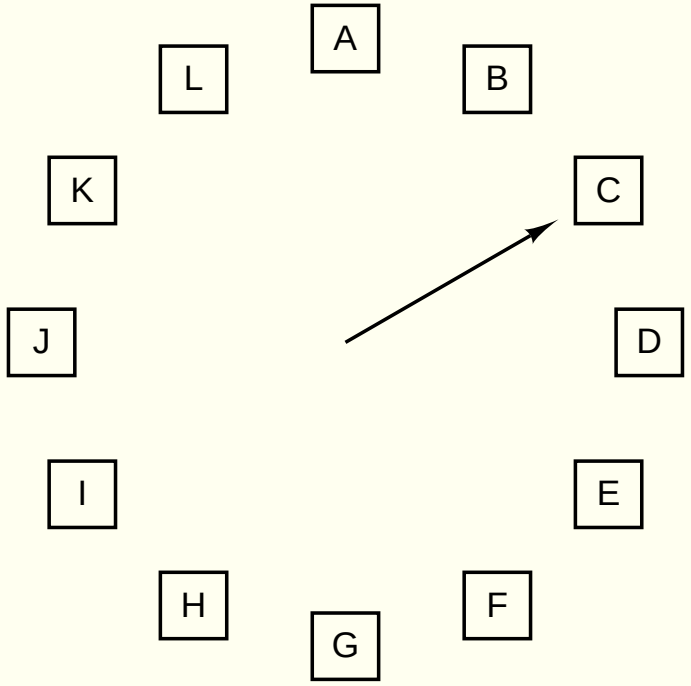
- Simplementes coloca as páginas em uma fila
- Pode remover páginas importantes

Segunda chance



- Se o bit R == 0, a página é substituída, senão
- bit R é limpo e a página é colocada no final da fila

Relógio



When a page fault occurs,
the page the hand is
pointing to is inspected.
The action taken depends
on the R bit:

R = 0: Evict the page

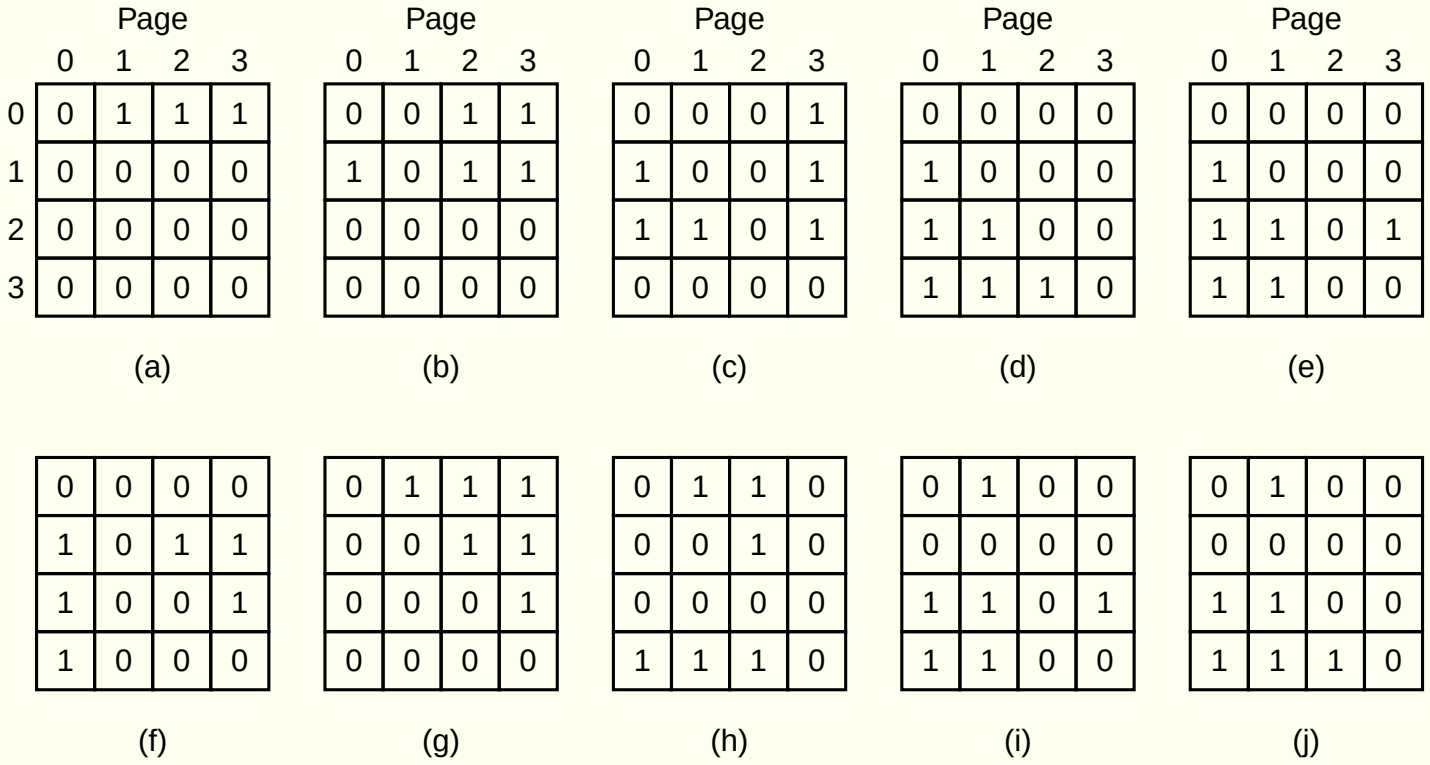
R = 1: Clear R and advance hand

- Implementação circular da segunda chance

Uso recente

- LRU (Least Recently Used)
- Implementação utilizando lista ligada
- Implementação em hardware com contador
  - incrementado a cada instrução
  - entrada na tabela deve armazenar o contador
- Implementação com matriz n x n

LRU em hardware



Acessos: 0 1 2 3 2 1 0 3 2 3

Simulando LRU em software

Página não usada frequentemente

- Contador de uso para cada página (soma o bit R a cada clock tick)
- Não esquece nada...
- Considere um compilador baseado em passos

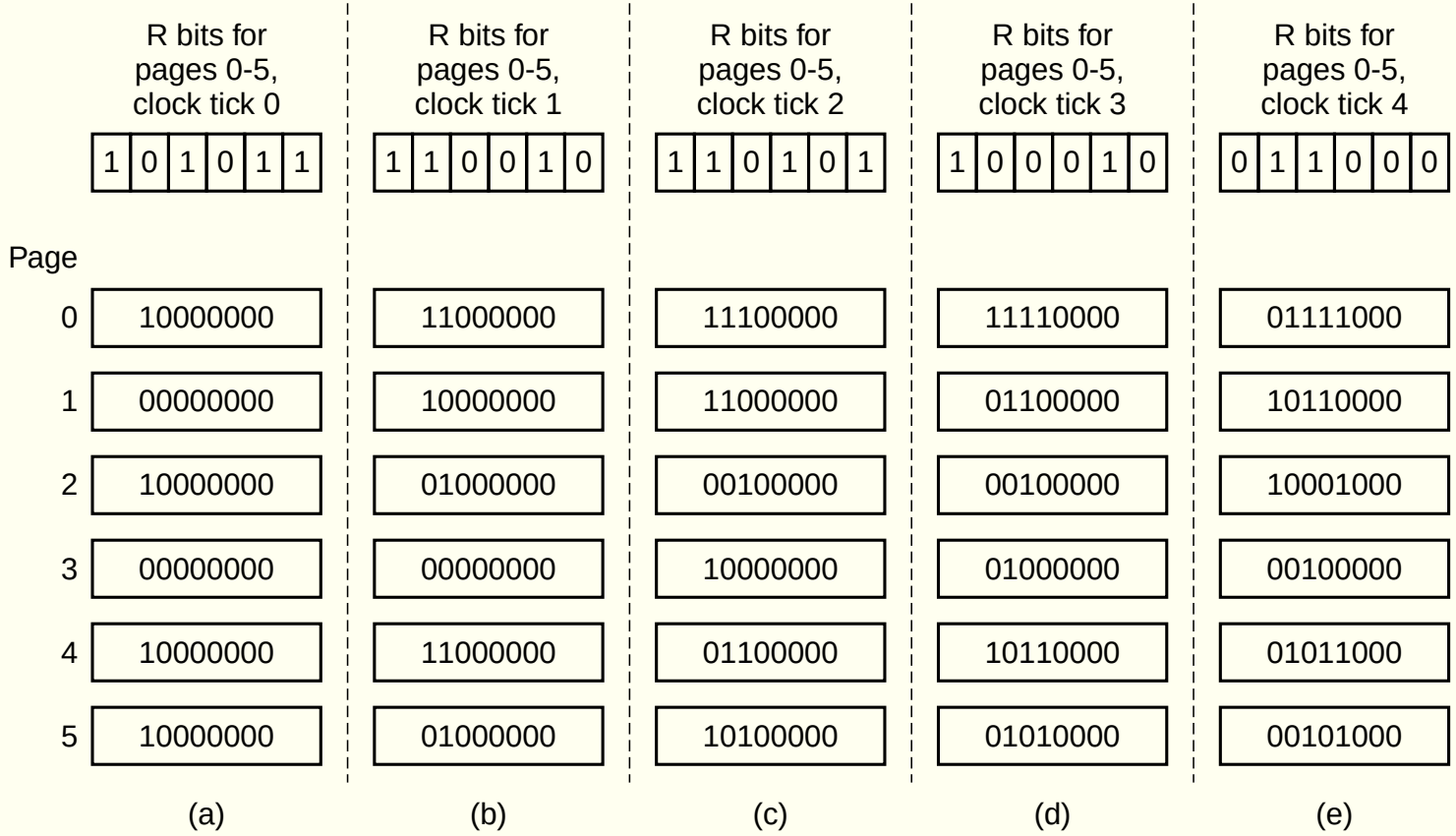
Não usada frequentemente

Aging

- O contador é deslocado à direita
- Bit R é adicionado à esquerda

```
          1 0 1 0 1 1              1 0 1 1 0 1
>> 1        1 0 1 0 1     >> 1       1 0 1 1 0
+  1     1 1 0 1 0 1      +  0    0 1 0 1 1 0
```

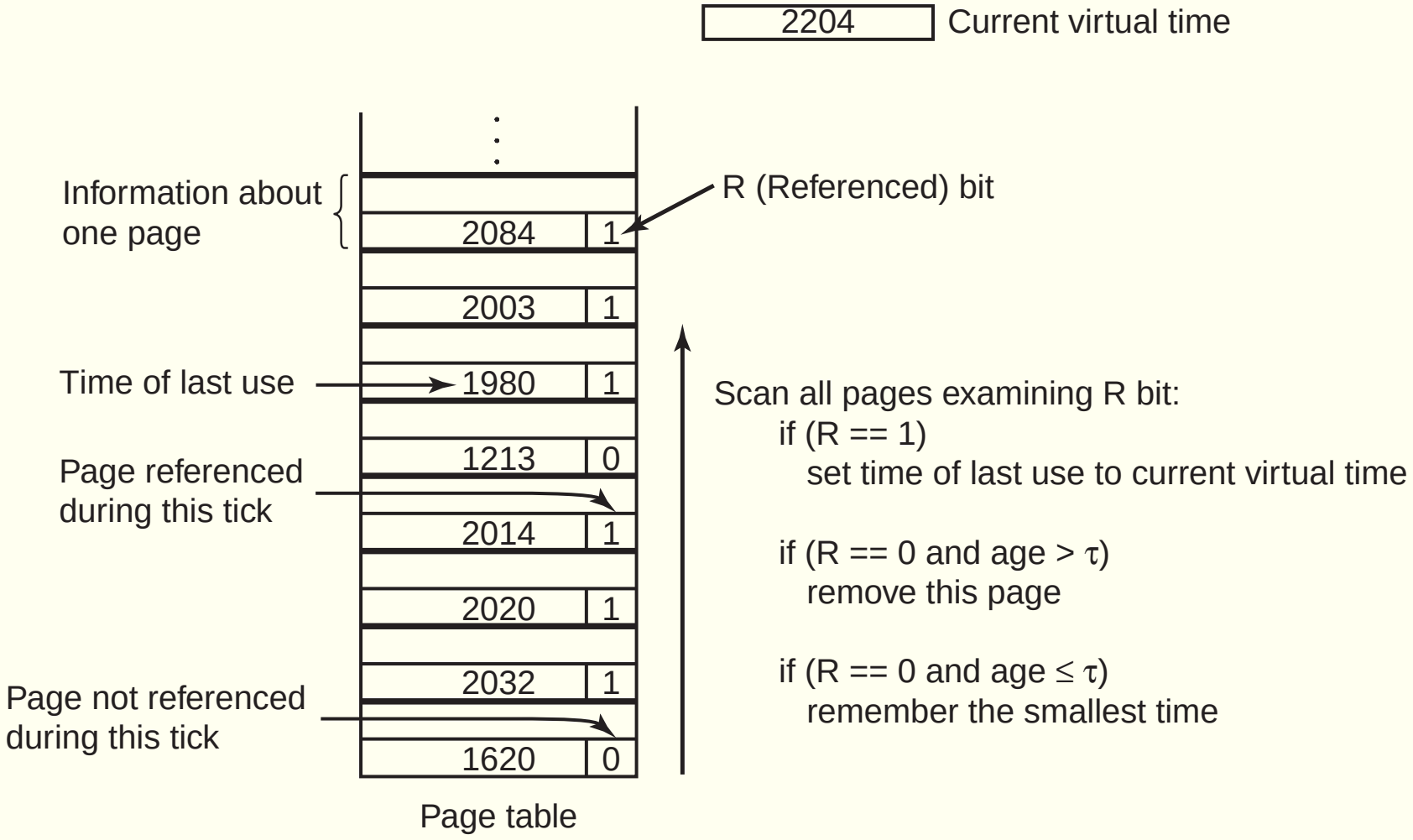
Não usada frequentemente



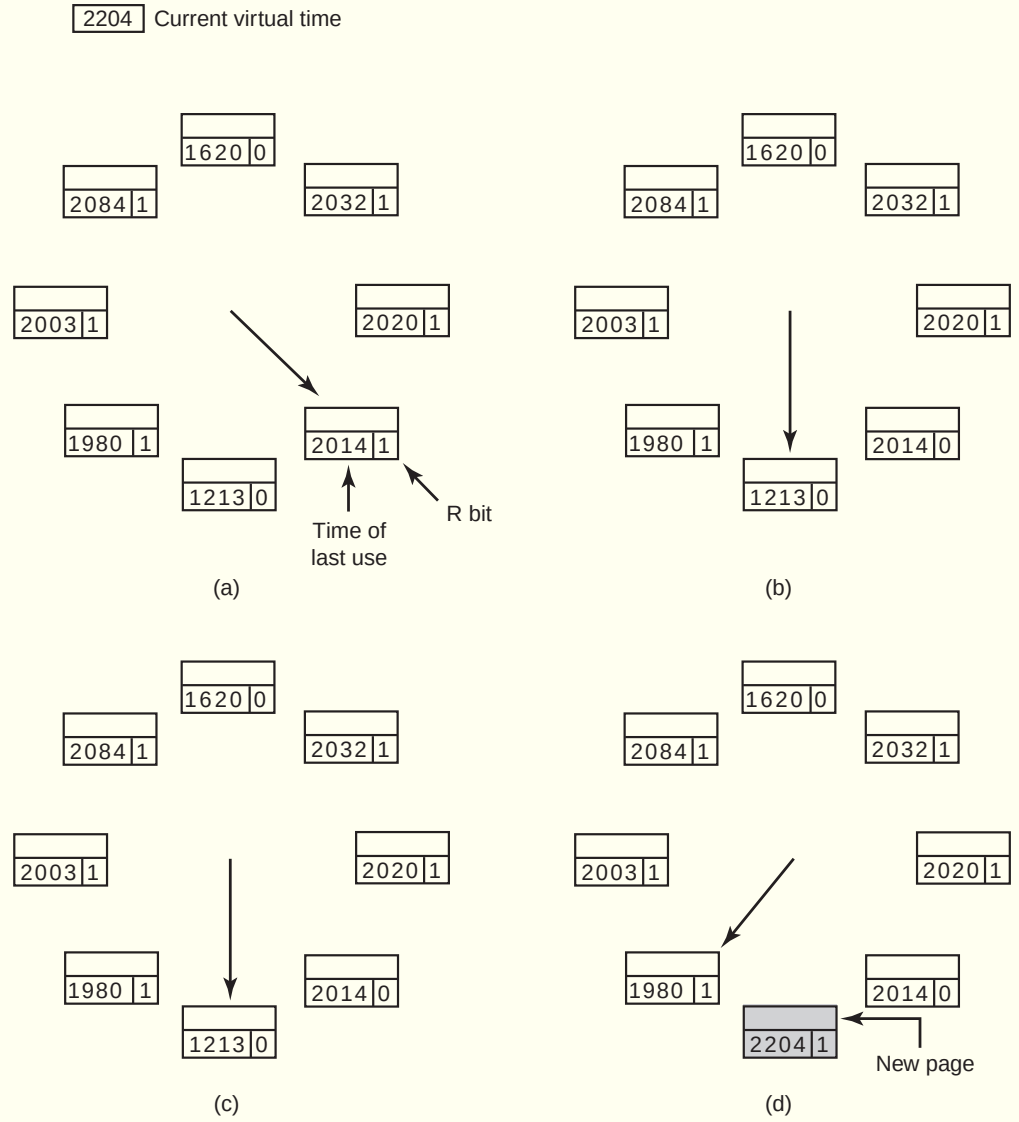
Working Set $w(k, t)$

- Conjunto de páginas utilizadas nas últimas k referências em relação ao instante t.
- Paginação sob demanda
- Prepaging
- Implementação exata é muito cara

Working Set


2204
Current virtual time
Information about one page
R (Referenced) bit
2084
1
2003
1
Time of last use
1980
1
1213
0
Page referenced during this tick
2014
1
2020
1
2032
1
Page not referenced during this tick
1620
0
Page table
Scan all pages examining R bit:
if (R == 1)
set time of last use to current virtual time
if (R == 0 and age > τ)
remove this page
if (R == 0 and age ≤ τ)
remember the smallest time

WSClock


2204 Current virtual time
2084 1
1620 0
2032 1
2003 1
2020 1
1980 1
2014 1
1213 0
Time of last use
R bit
(a)
(b)
(c)
(d)
2014 0
2204 1
New page

Resumo

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?

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)

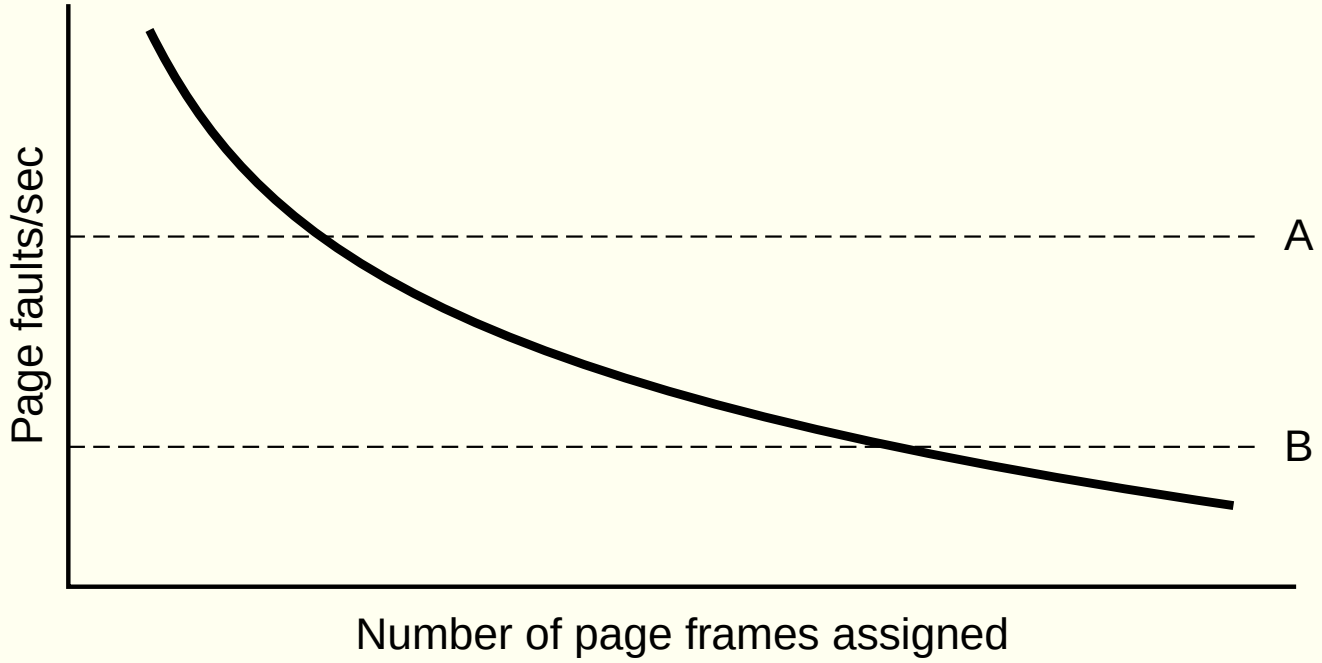
A0
A1
A2
A3
A4
A6
B0
B1
B2
B3
B4
B5
B6
C1
C2
C3

(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

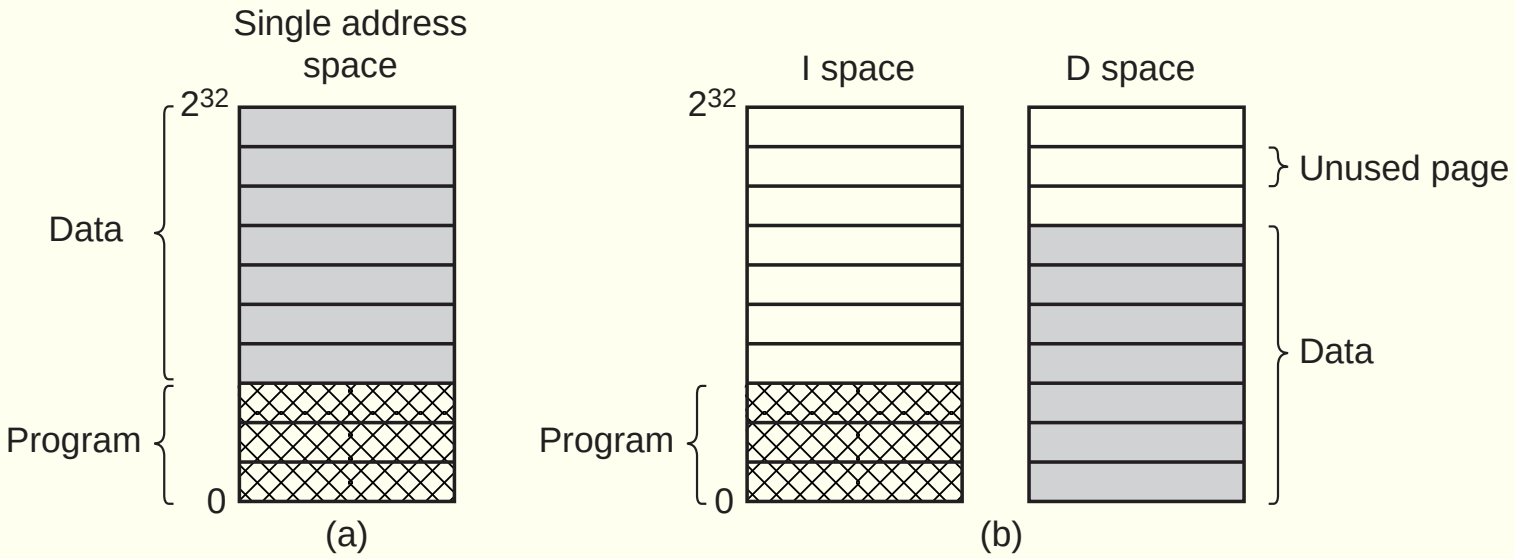
Controle da carga

- *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

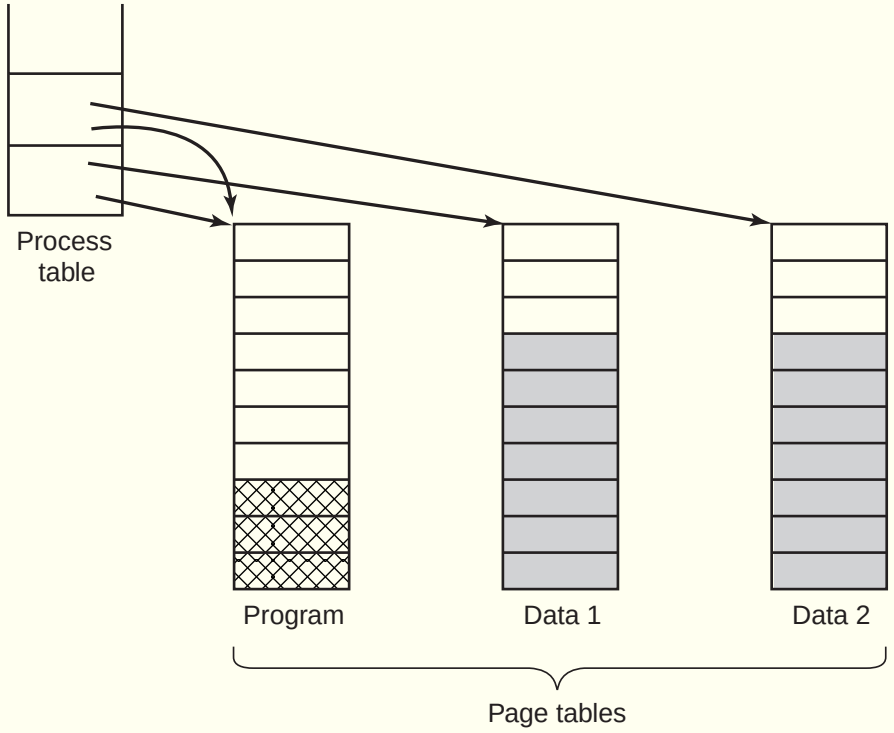
- Compromisso
  - Páginas grandes $\longrightarrow$ fragmentação interna
  - Páginas pequenas $\longrightarrow$ overhead de gerenciamento
- Tamanho nesta máquina: 4KB

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
- ...

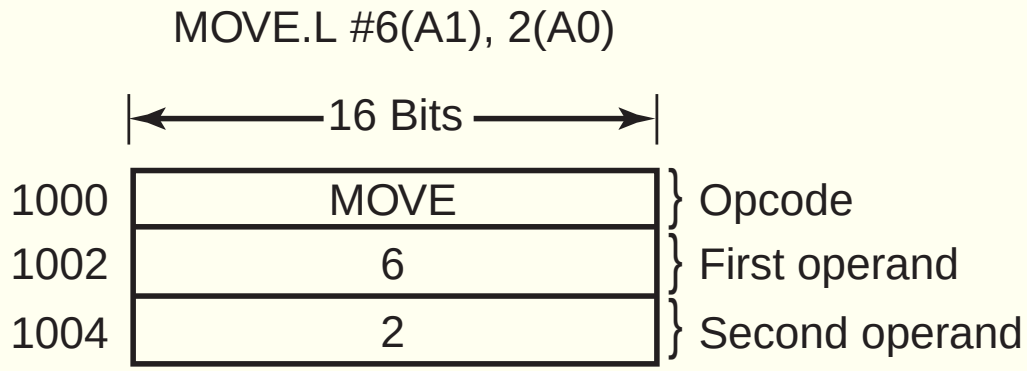
Tratamento de falta de página

- 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?

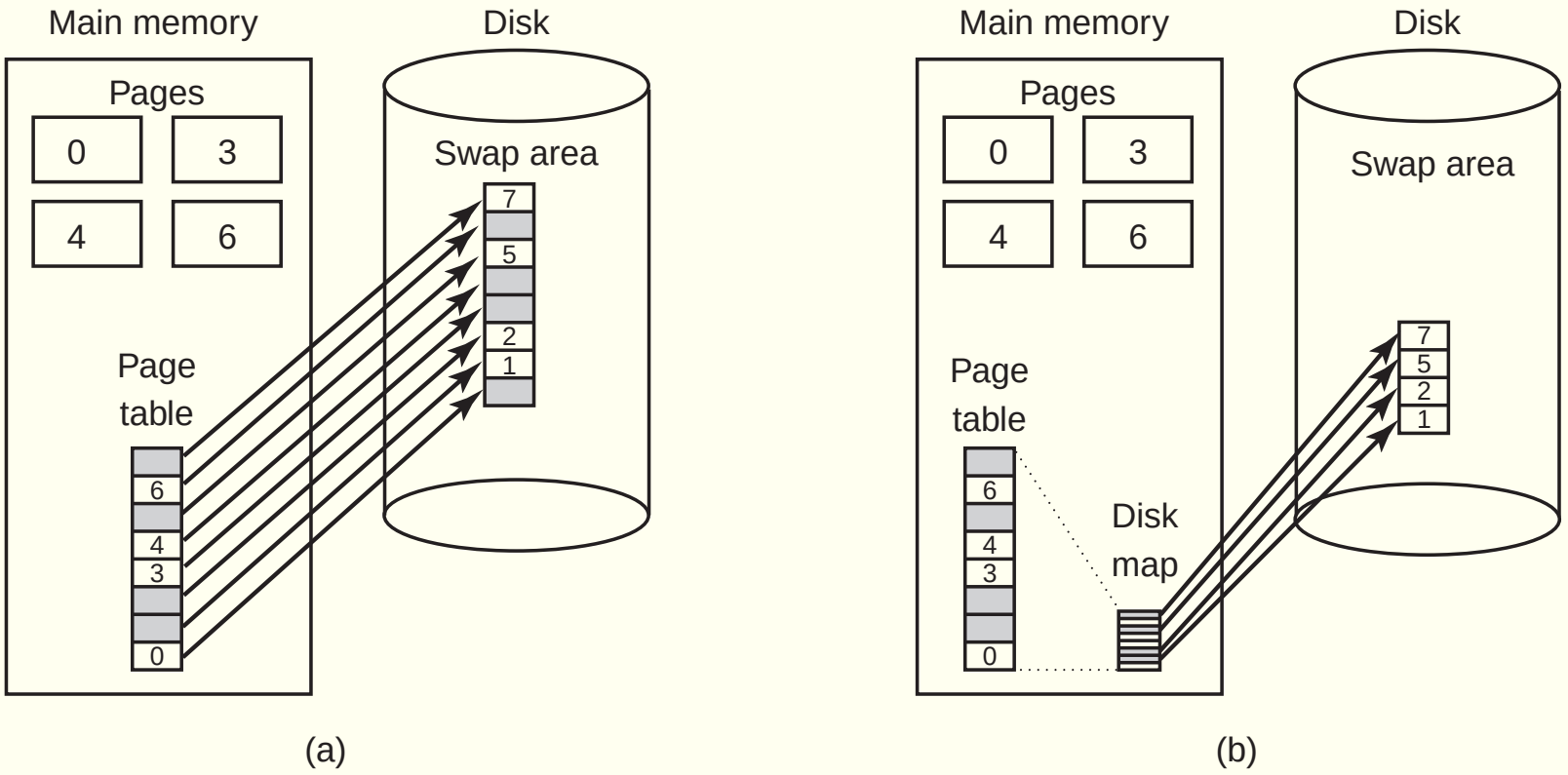
Fixação de páginas na memória

- Melhora do desempenho para processos com características especiais

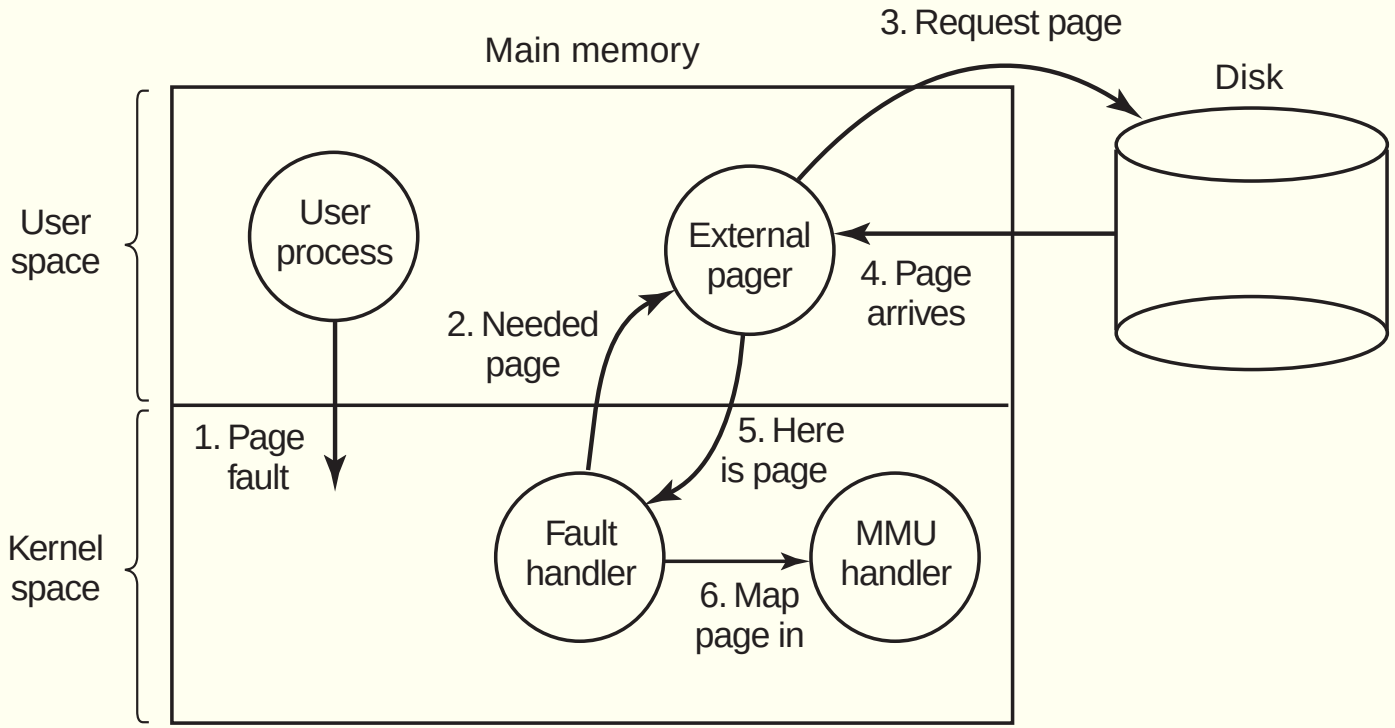
```
int mlock (const void *ADDR, size_t LEN);
int munlock (const void *ADDR, size_t LEN);
```

- Só para super-usuários? :-(

Memória secundária



Paginador externo

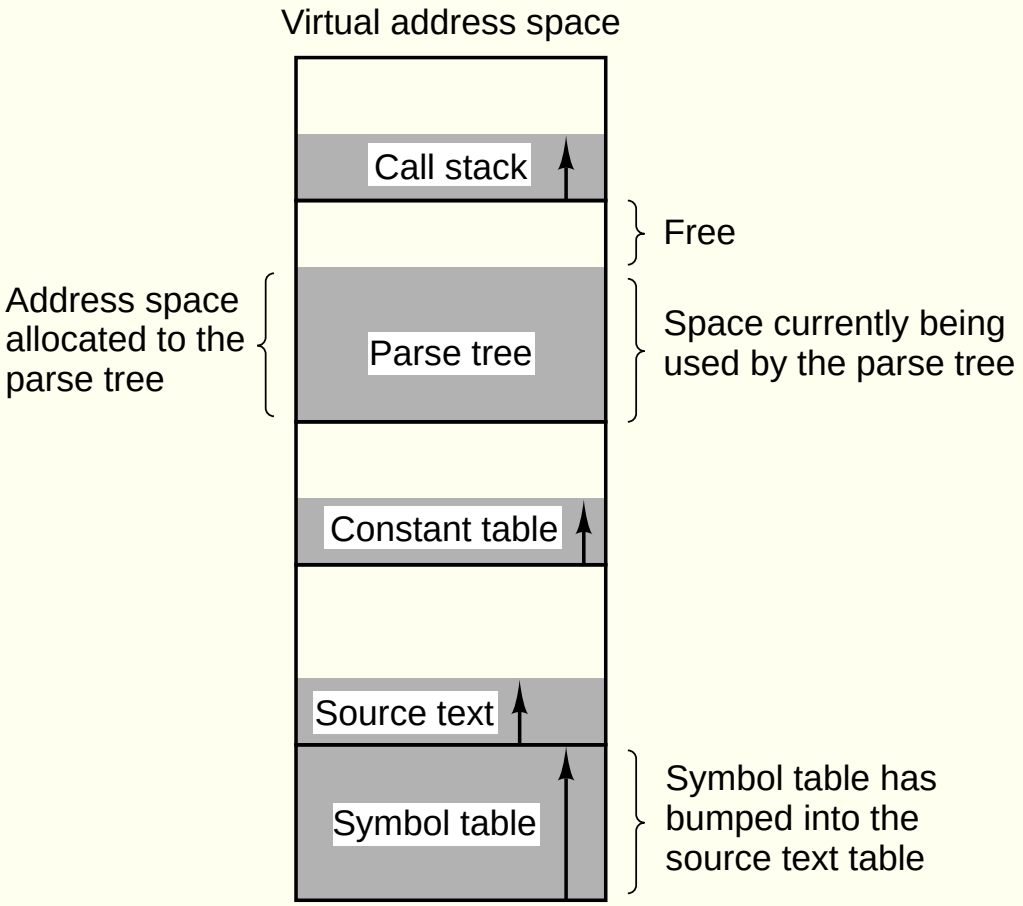


Flexibilidade versus Overhead

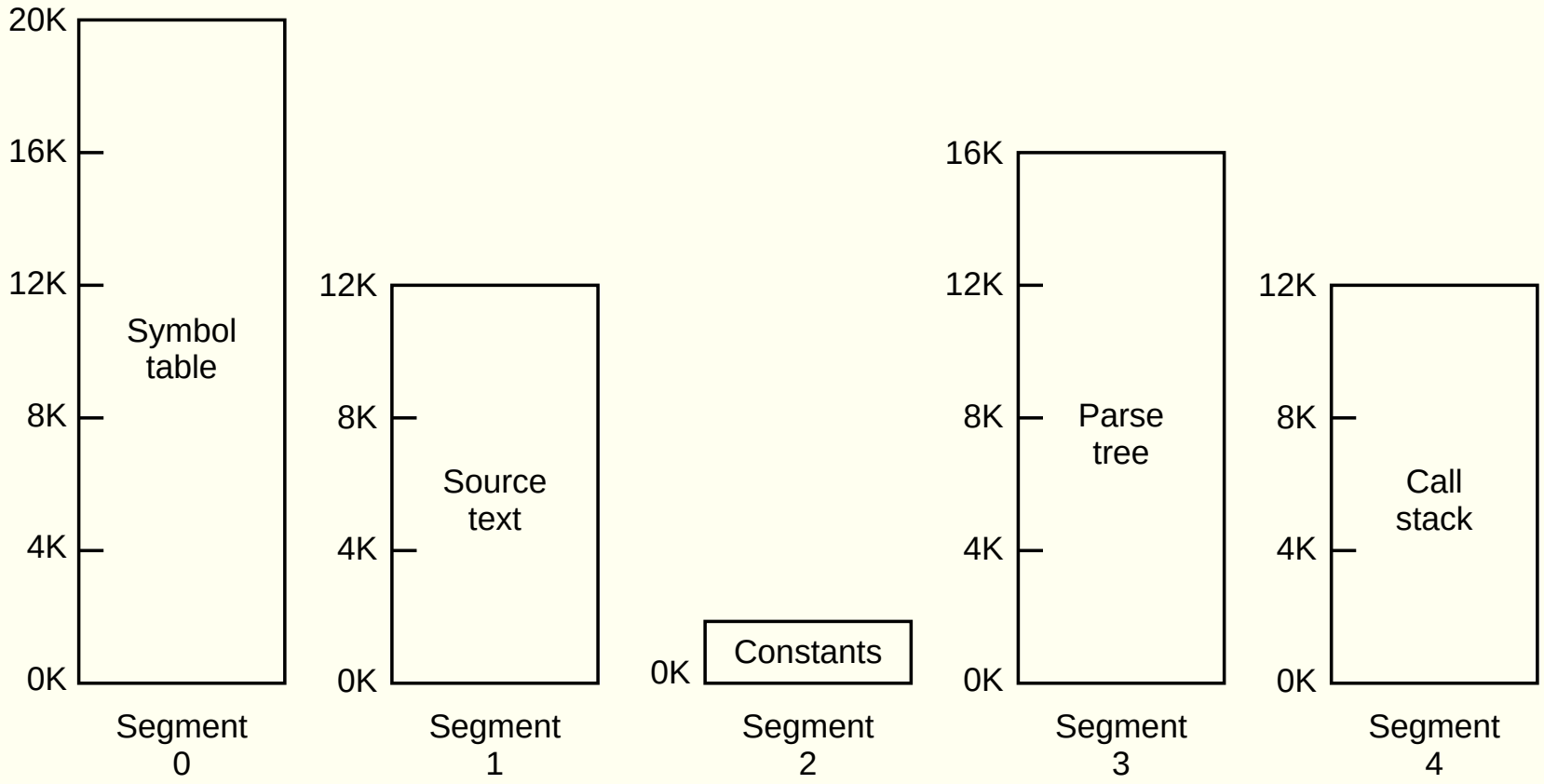
Segmentação

- Vários espaços de endereçamento
- Maior flexibilidade

Limites de um espaço único

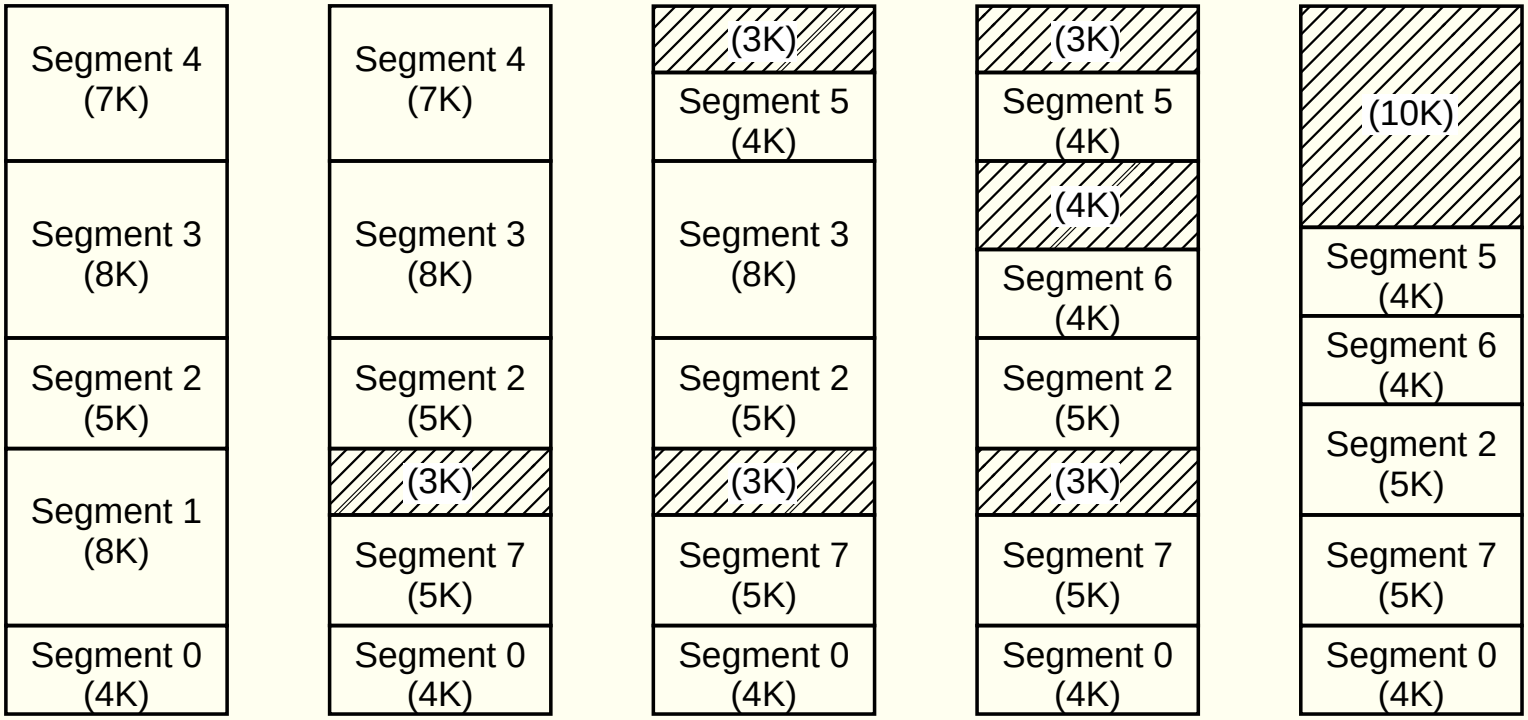


Segmentação

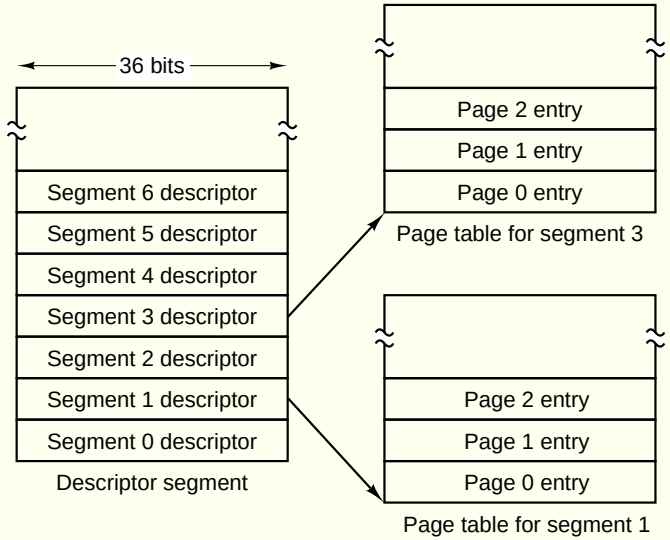


Consideration	Paging	Segmentation
Need the programmer be aware that this technique is being used?	No	Yes
How many linear address spaces are there?	1	Many
Can the total address space exceed the size of physical memory?	Yes	Yes
Can procedures and data be distinguished and separately protected?	No	Yes
Can 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

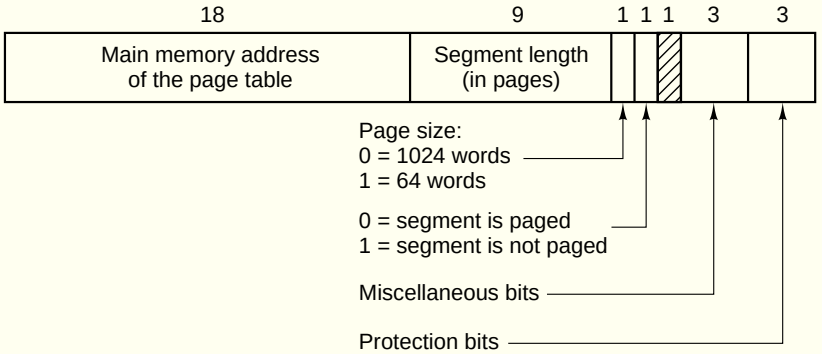
Segmentação pura



MULTICS: segmentação e paginação

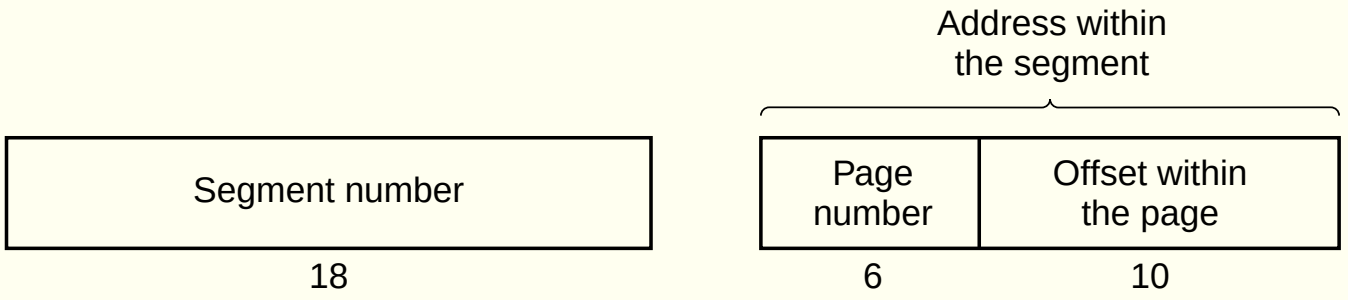


(a)

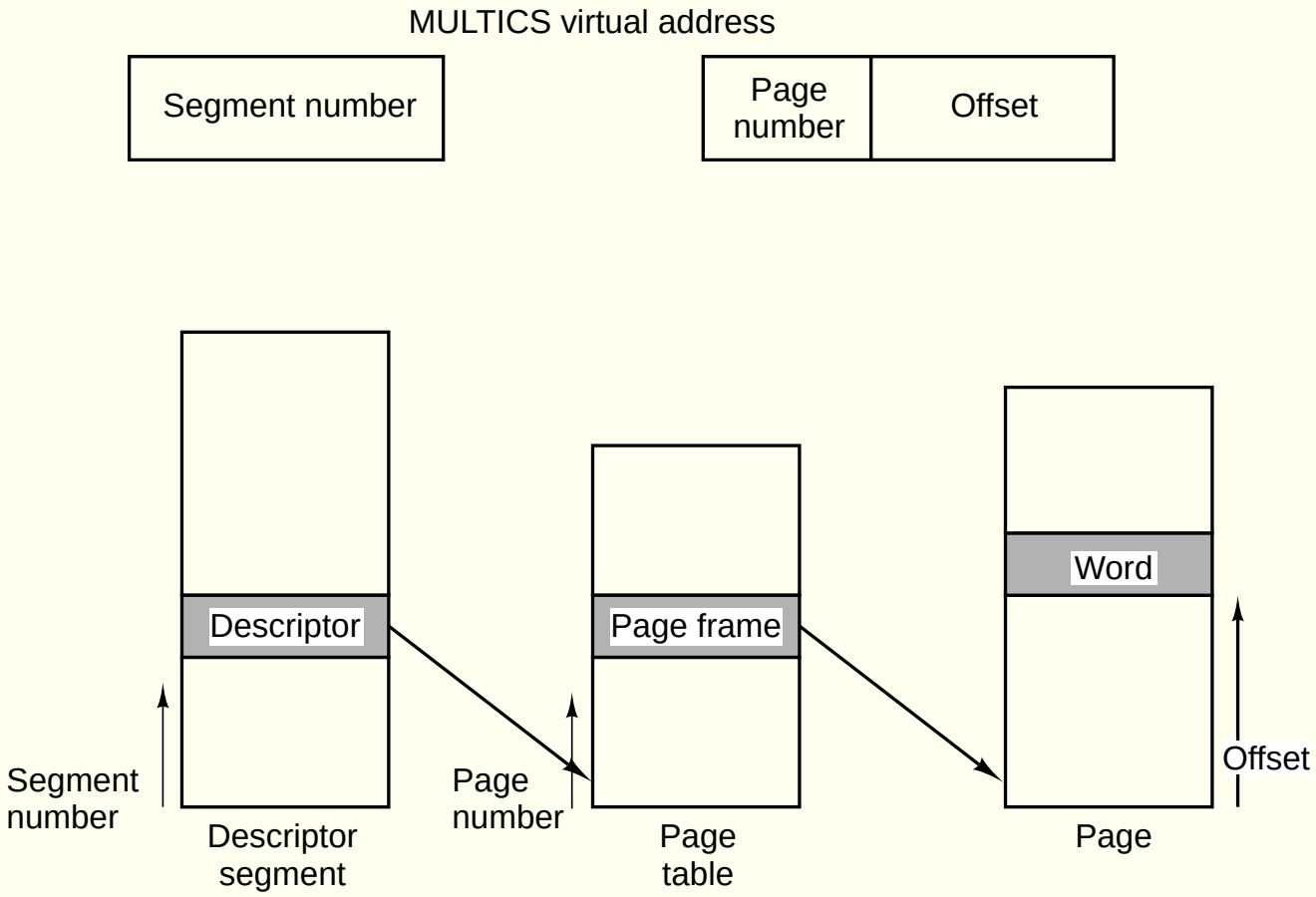


(b)

MULTICS: endereçamento virtual

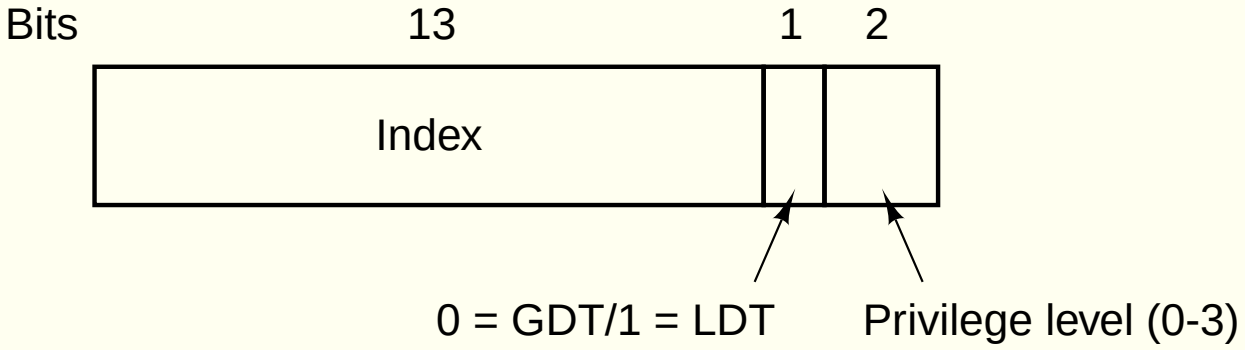


MULTICS: Conversão de endereços



Intel Pentium: segmentação e paginação

- Seletor de segmento



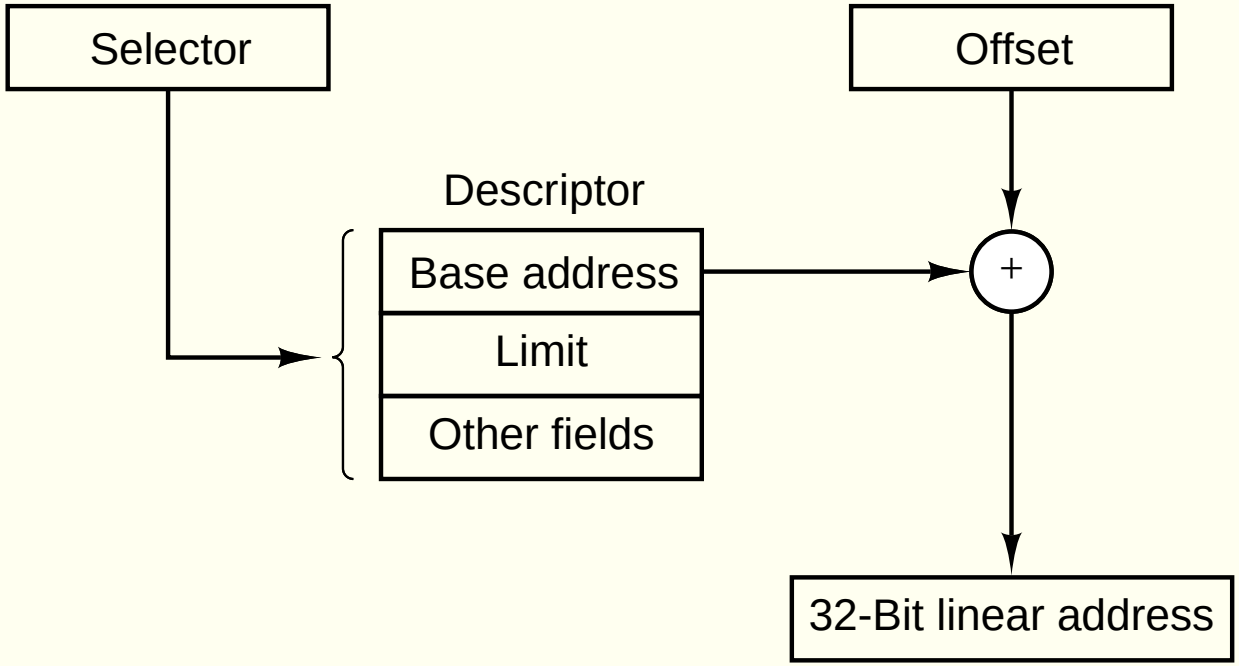
- LDT (Local Descriptor Table)
- GDT (Global Descriptor Table)

Registradores de segmento

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

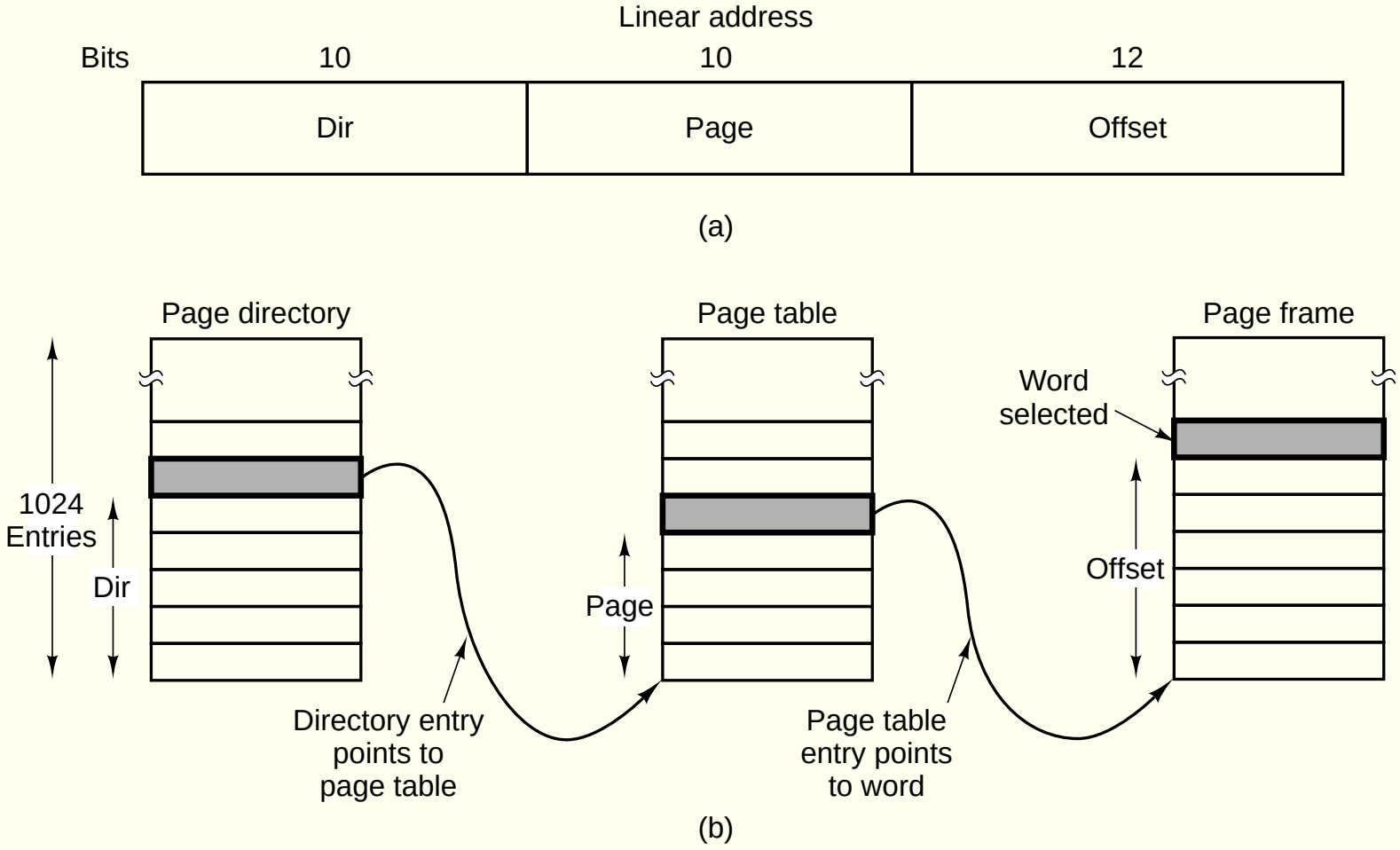
Conversão de endereços

(seletor, deslocamento) $\longrightarrow$ linear



Conversão de endereços

linear → físico



Resumo

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