# Gerenciamento de Memória

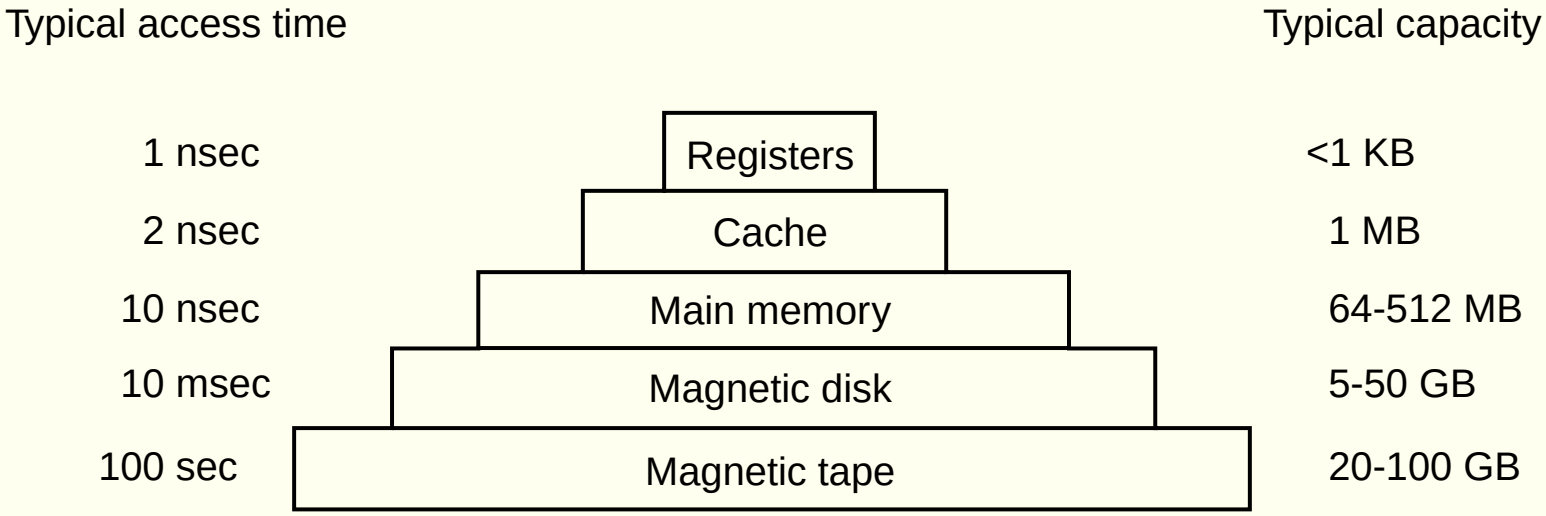
# Sonho dos usuários

Idealmente, a memória deveria ser

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

# Realidade para os usuários

## Hierarquia de Memória



# 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;
```

# Hierarquia de Memória

## Cache

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

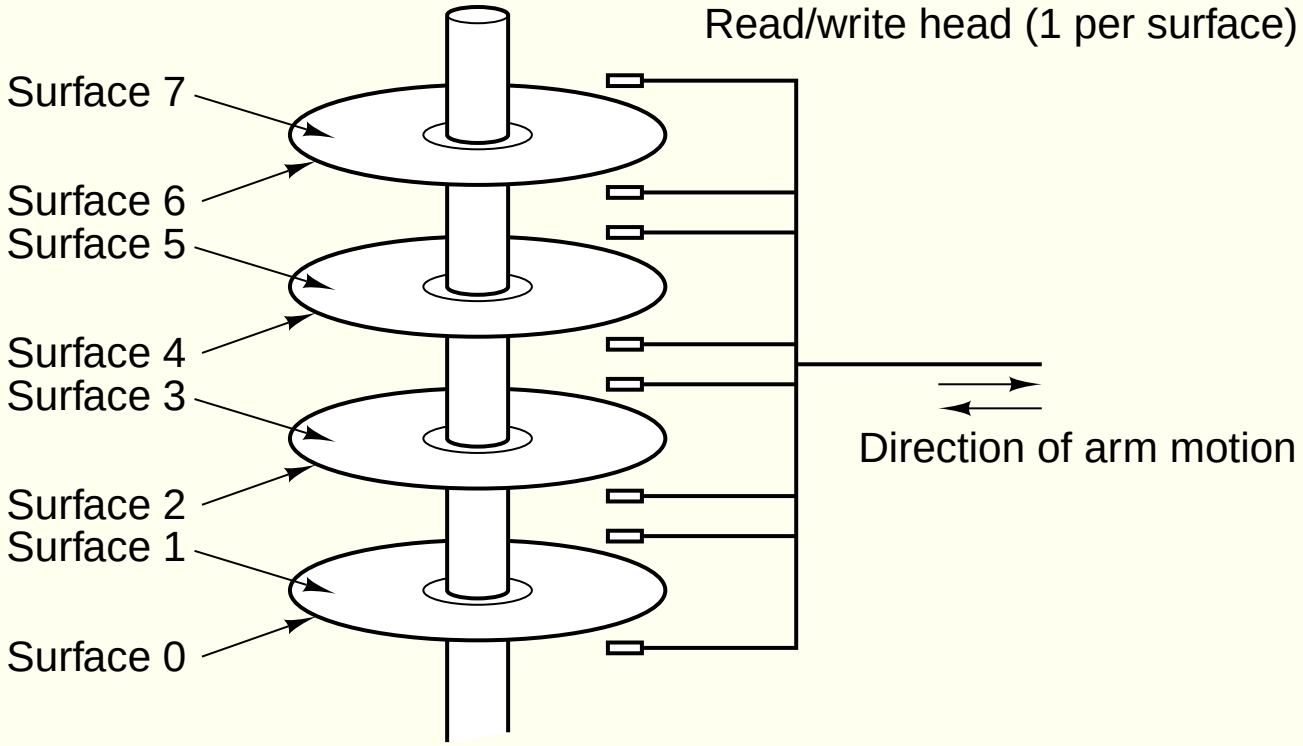
# Hierarquia de Memória

## Memória Principal

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

# Hierarquia de Memória

## Disco



# Hierarquia de Memória

## Fitas magnéticas

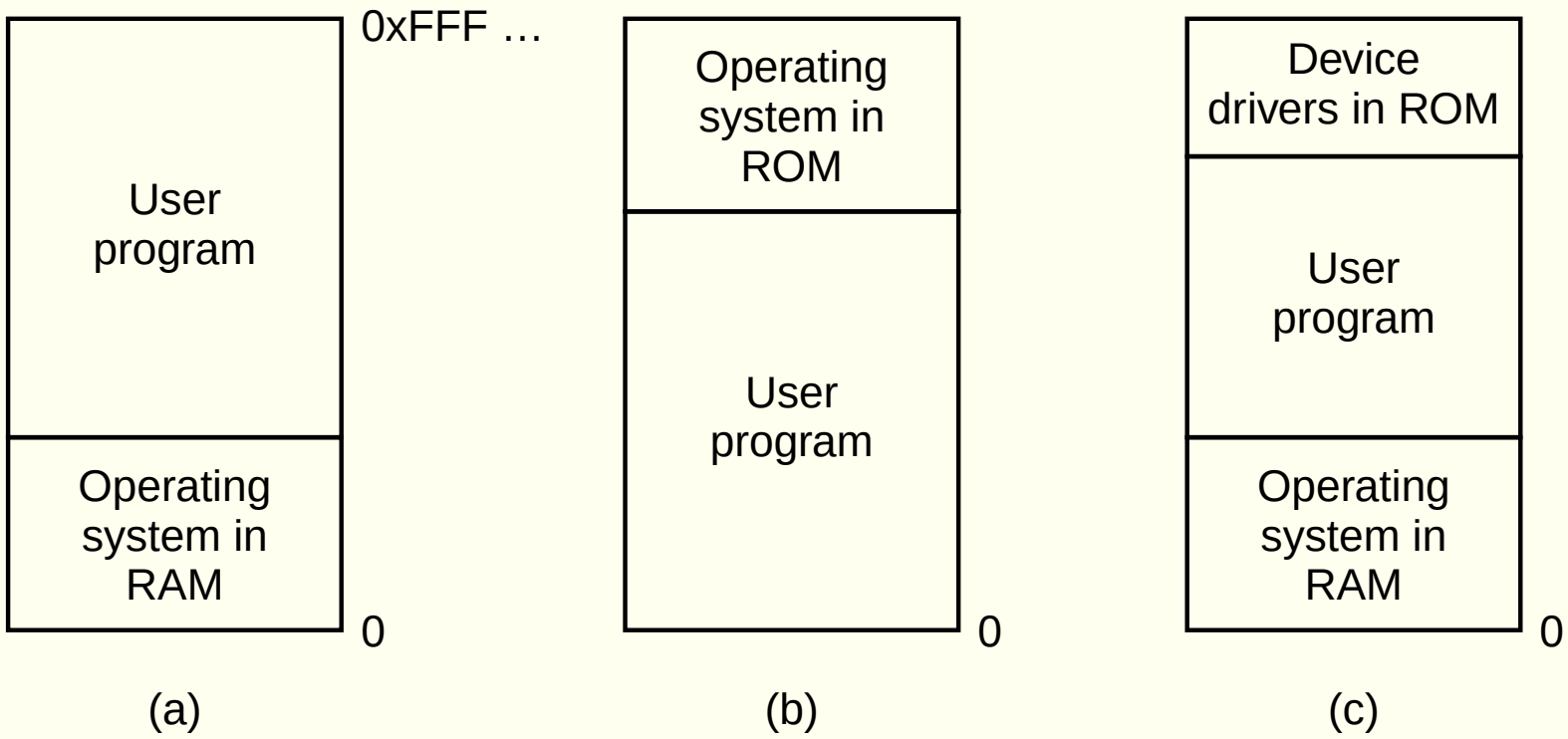
- Utilizadas para cópias de segurança (backups)
- Armazenamento de grandes quantidades de dados
- Acesso seqüencial

# 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

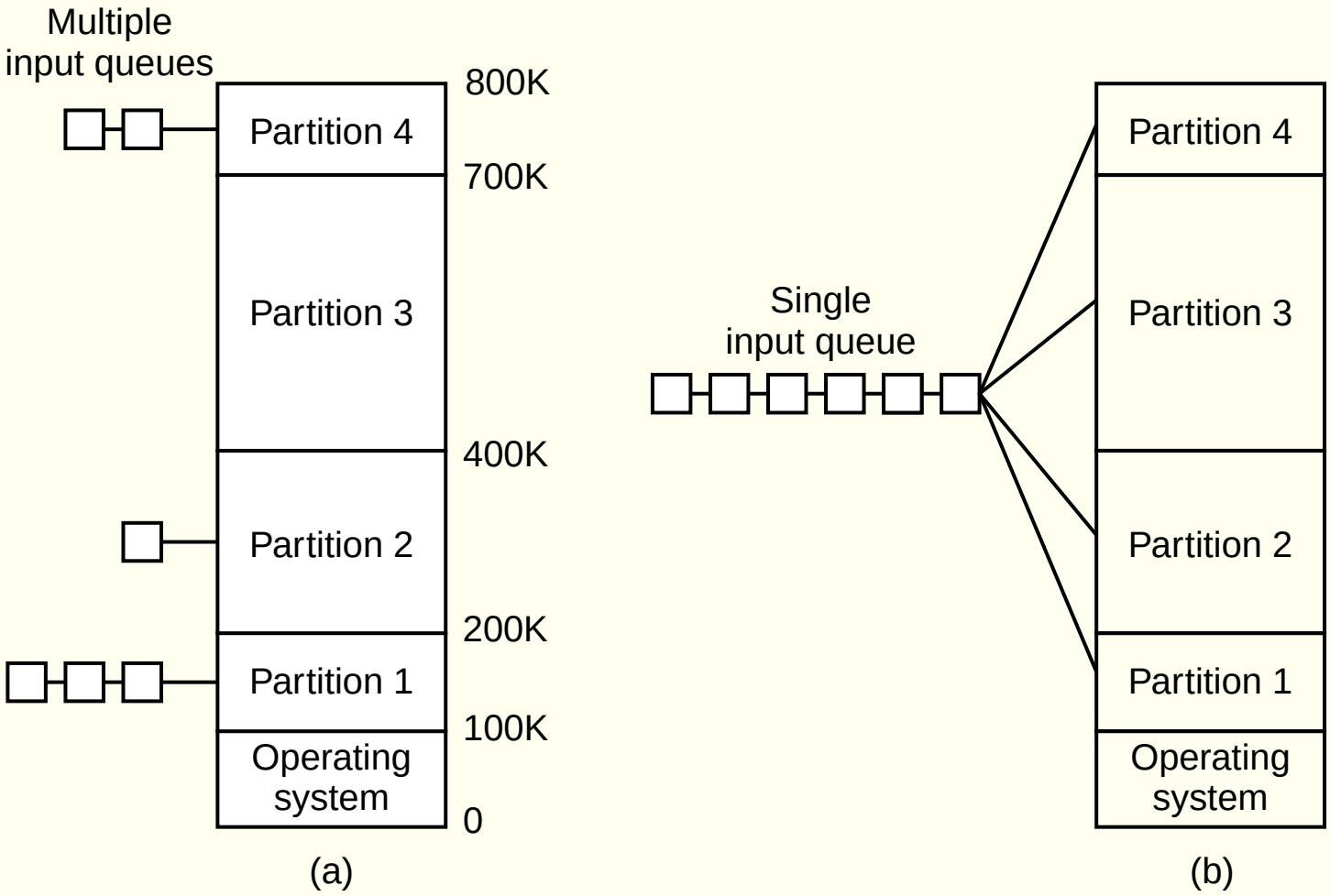



mainframes

palmtops

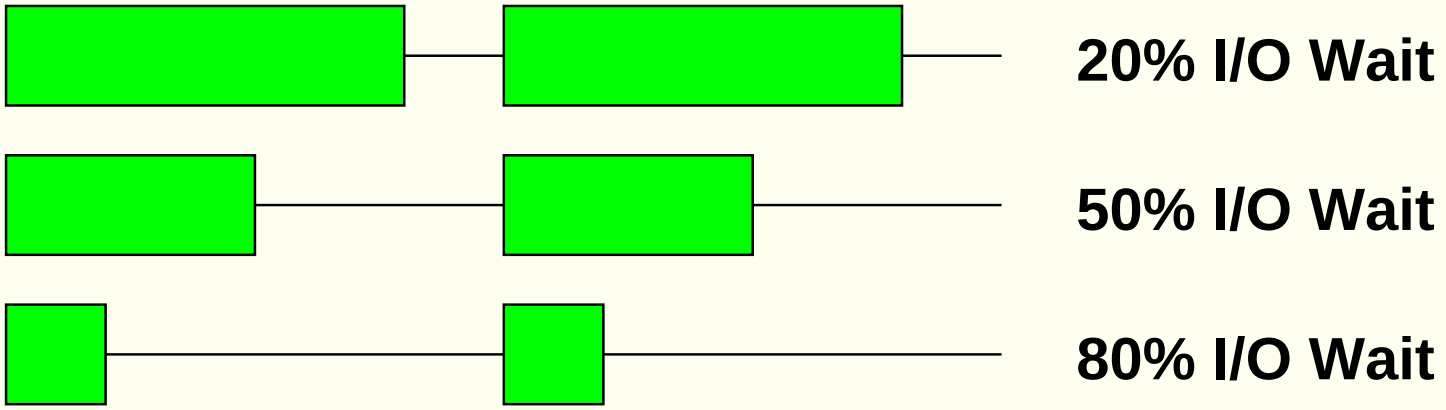
MS-DOS

# Multiprogramação e partições fixas



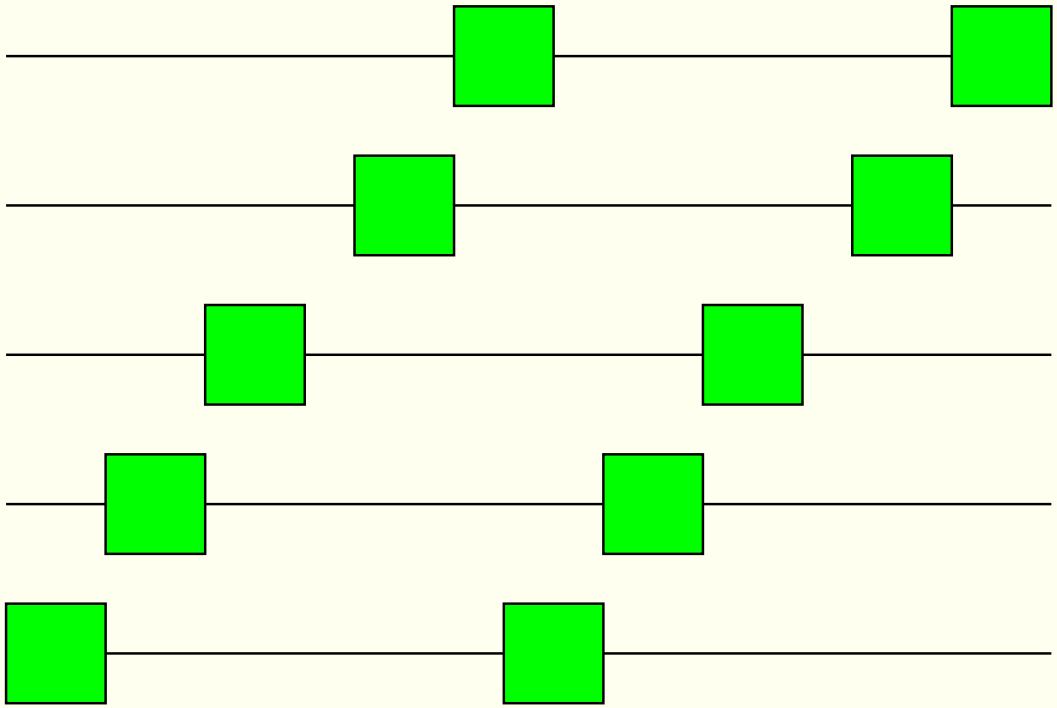
# Modelagem de multiprogramação

- Quantos processos colocar na memória?
- Estimativa de uso da CPU



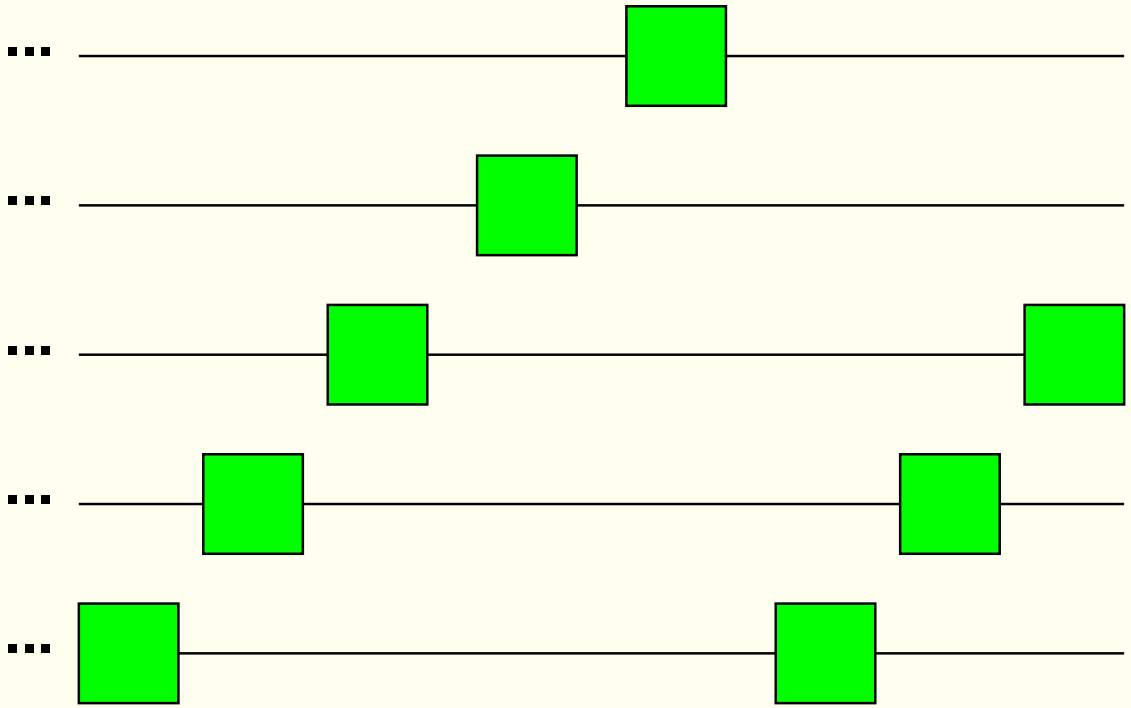
# Modelagem de multiprogramação

- Quantos processos colocar na memória?
- Aposta otimista: 5 processos 80

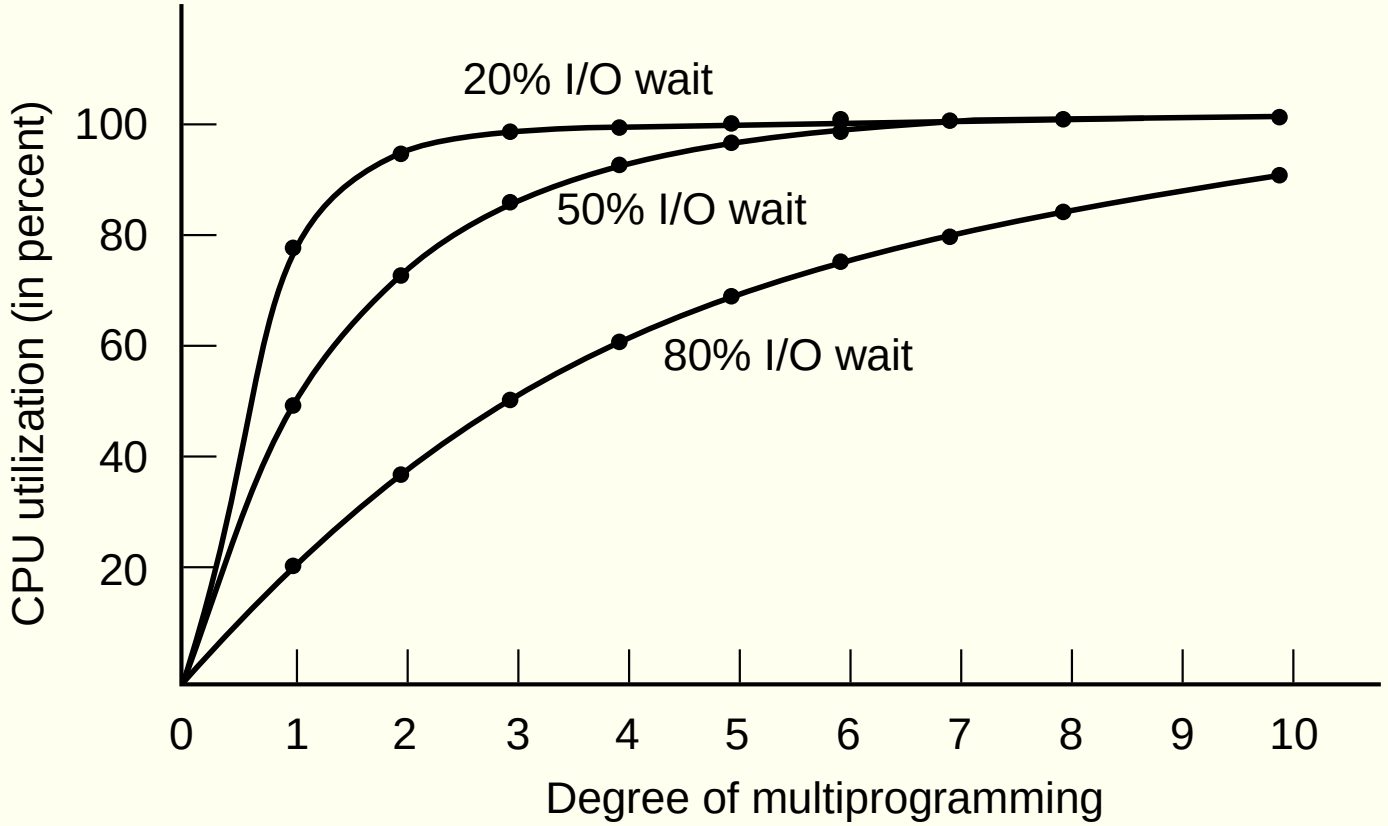


# Modelagem de multiprogramação

- Visão probabilística
- Taxa de utilização da CPU = 1 - $(0.8)^n$



# Modelagem de multiprogramação




Utilização da CPU = 1 - $p^n$

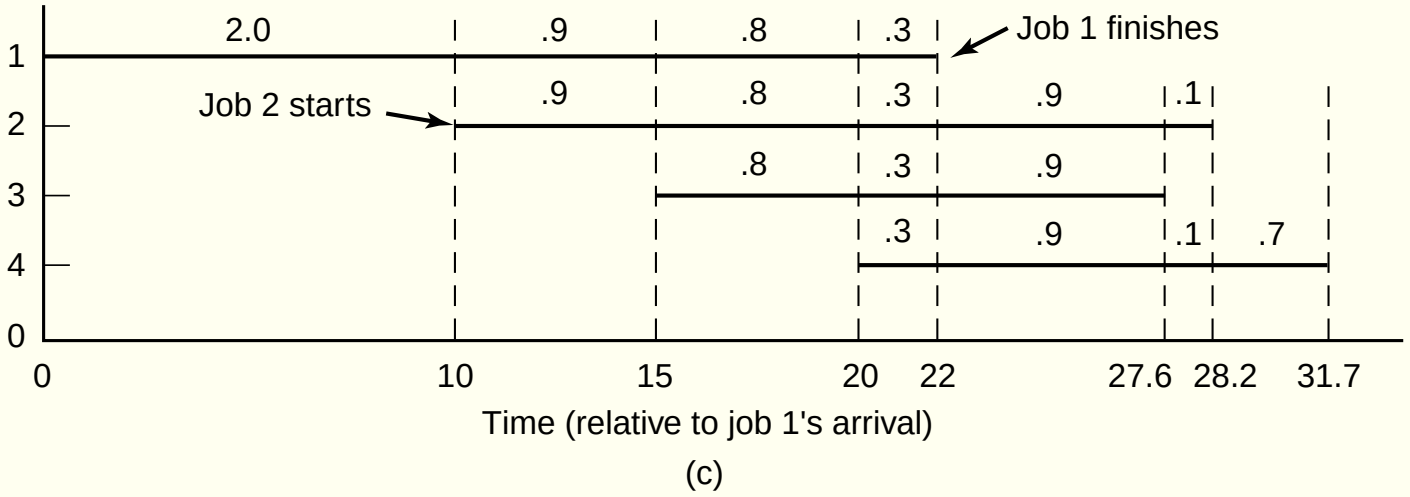
# Análise de Desempenho

| Job | Arrival time | CPU minutes needed |
|---|---|---|
| 1 | 10:00 | 4 |
| 2 | 10:10 | 3 |
| 3 | 10:15 | 2 |
| 4 | 10:20 | 2 |

(a)

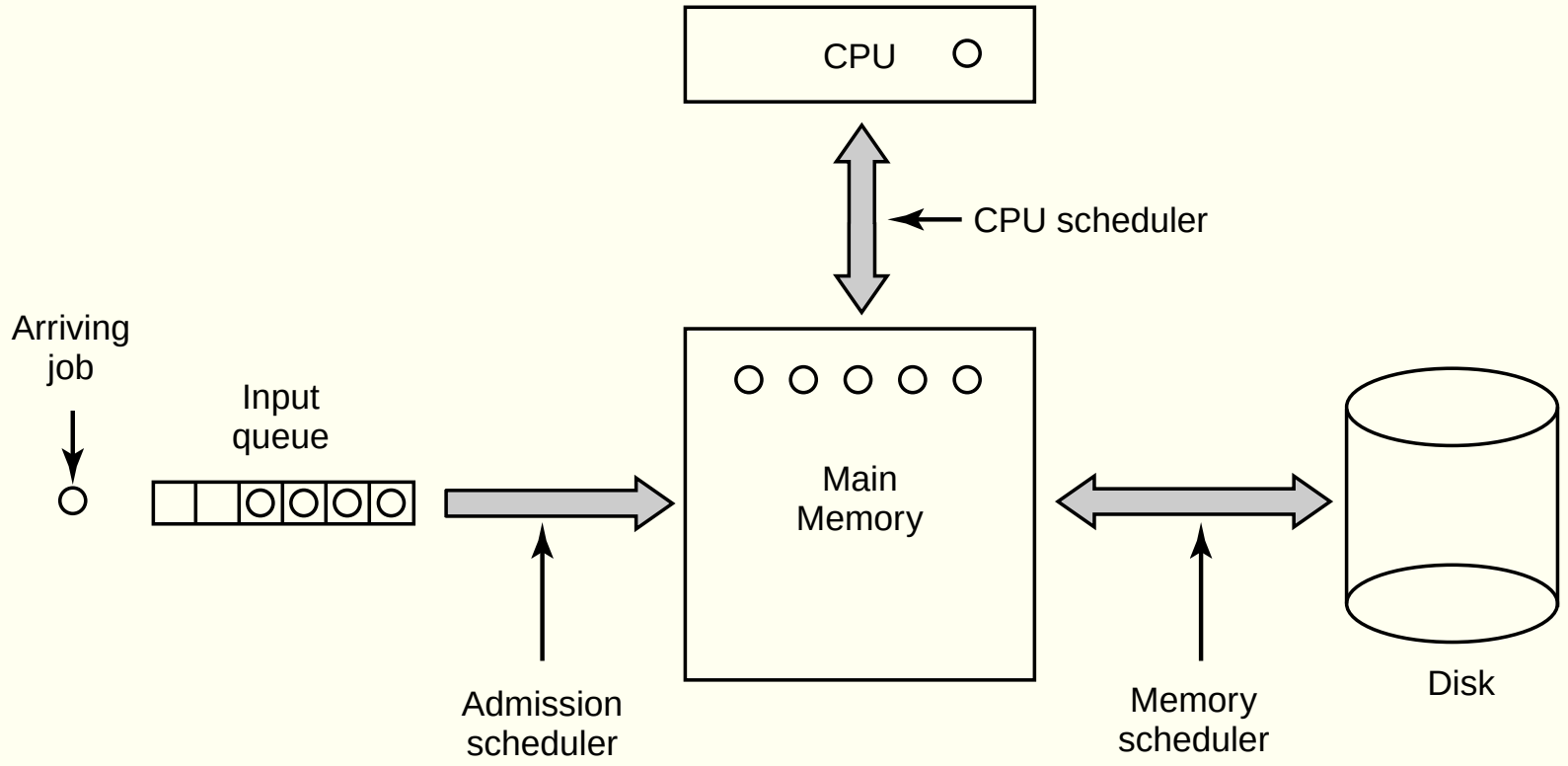
| | # Processes 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| CPU idle | .80 | .64 | .51 | .41 |
| CPU busy | .20 | .36 | .49 | .59 |
| CPU/process | .20 | .18 | .16 | .15 |

(b)

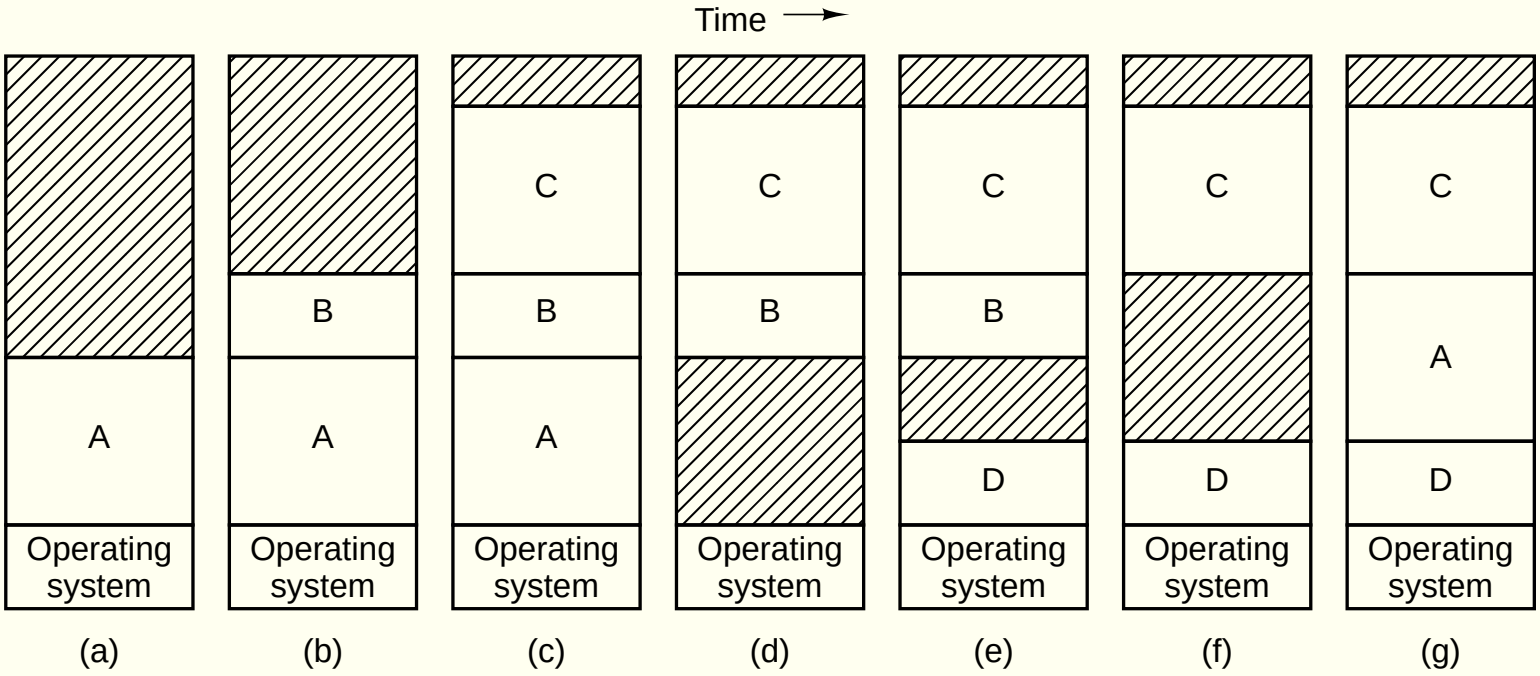



(c)

# 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

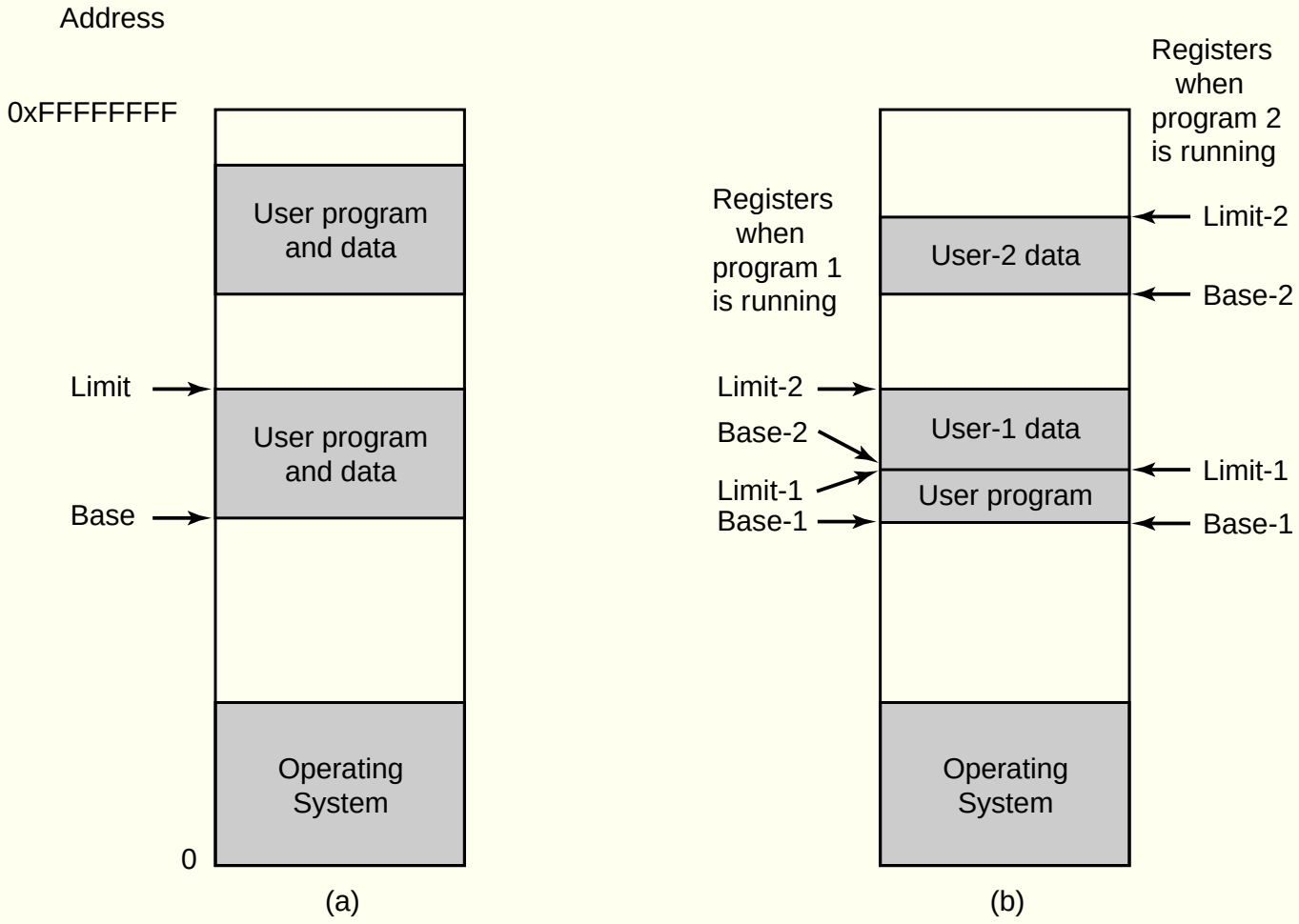
- Todos os endereços precisam ser identificados e alterados
- Não resolve o problema da proteção

# Bits de proteção

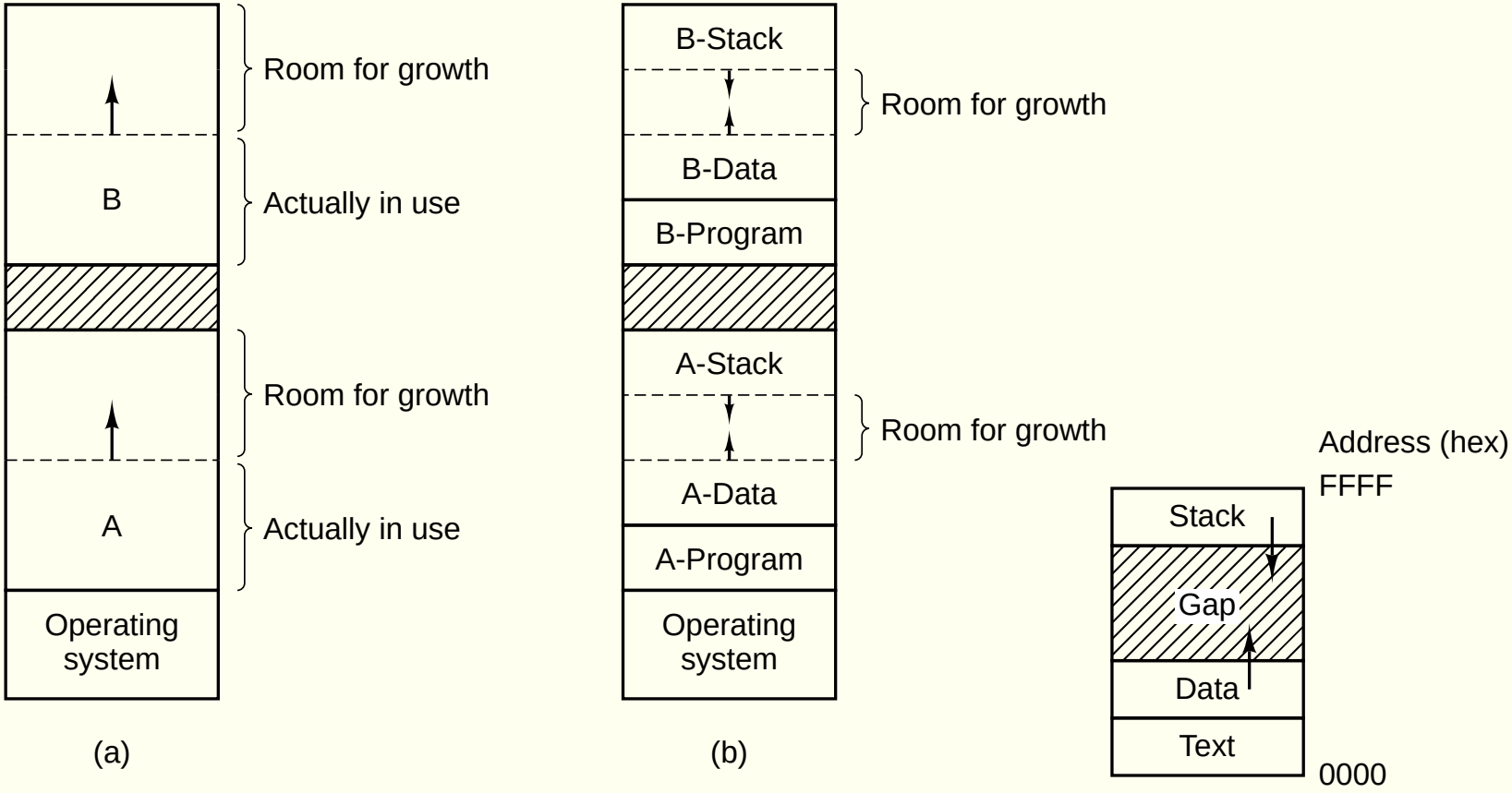
| | |
|---|---|
| 1010 | |
| 1010 | |
| 1010 | |
| 0011 | |
| 0011 | |

O PSW de cada processo deve conter os bits de proteção

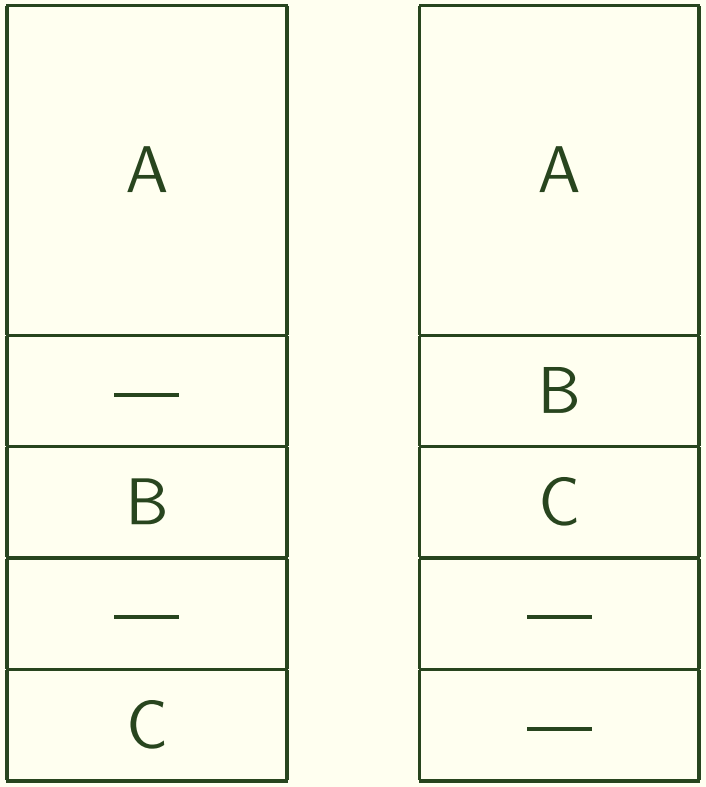
# Registradores base e limite



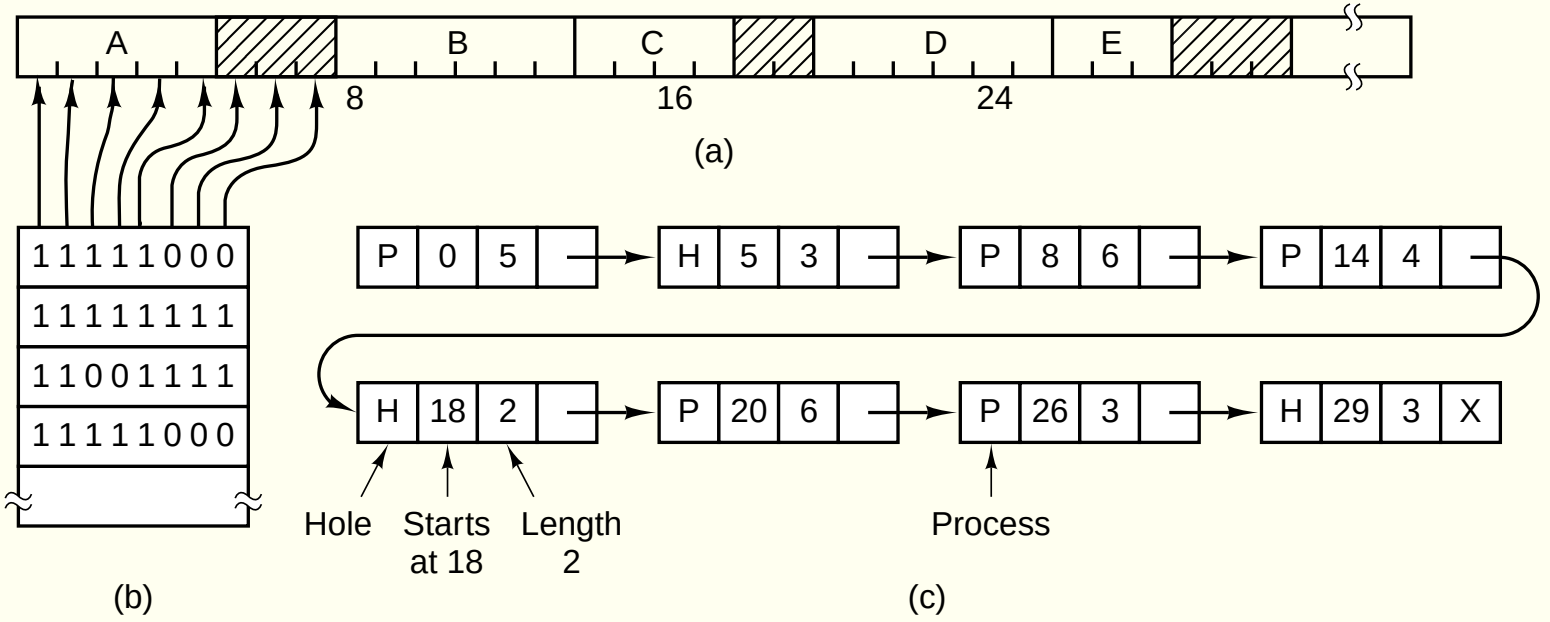
# Espaço para crescimento



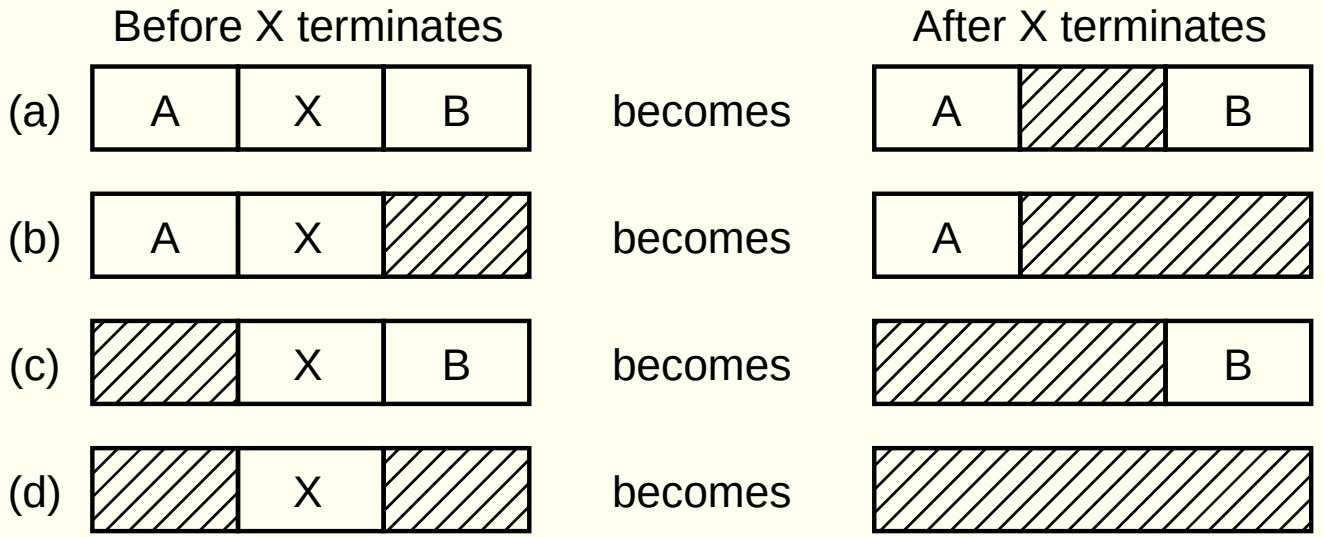
# Compactação de memória



# Bitmaps e lista de livres



# Atualização da lista



# Algoritmos para alocação de memória

- *First fit*
- *Next fit*
- *Best fit*
- *Worst fit*

# Problema relacionado

## Malloc, free e realloc

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

A alocação de registros dentro de um processo tem problemas semelhantes:

- administração de uma lista de livres
- fragmentação

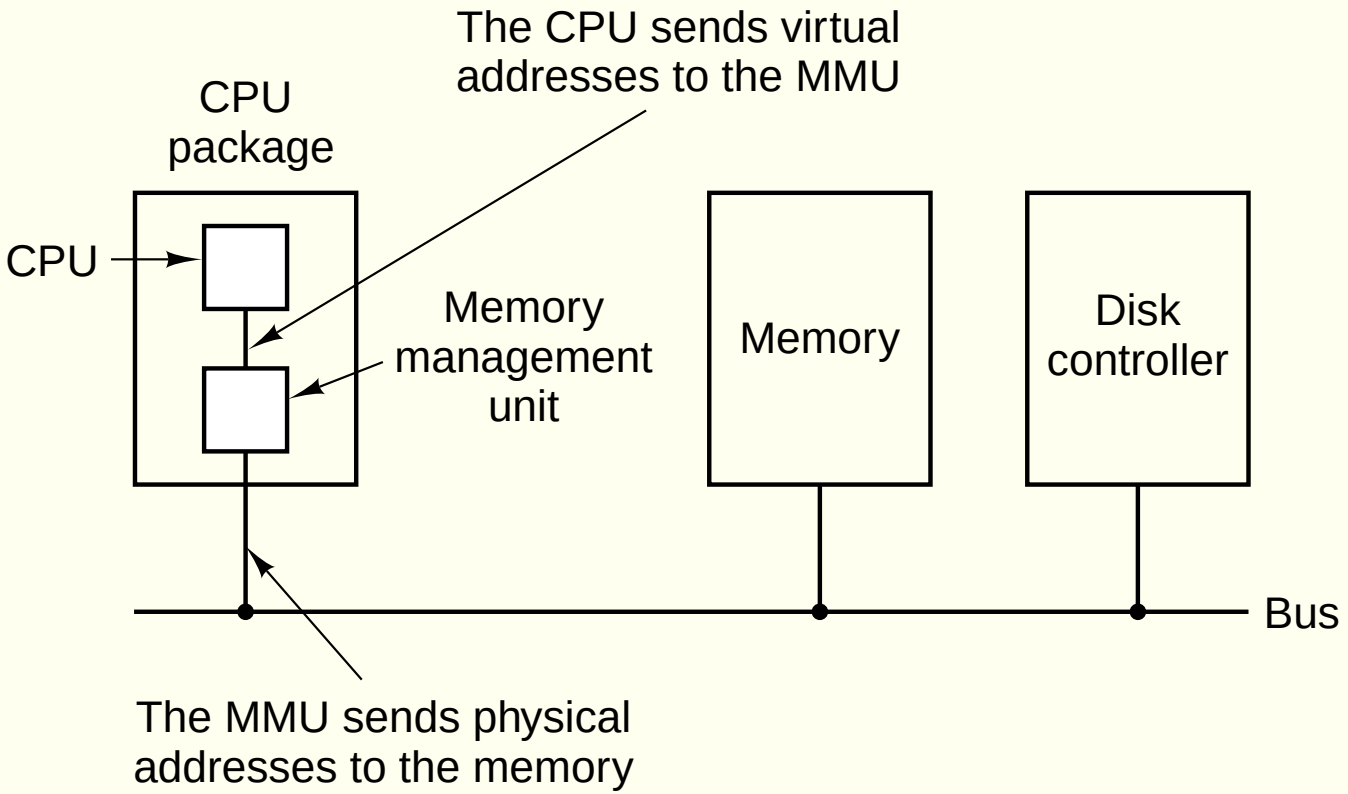
# Programas muito grandes

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

# Overlays

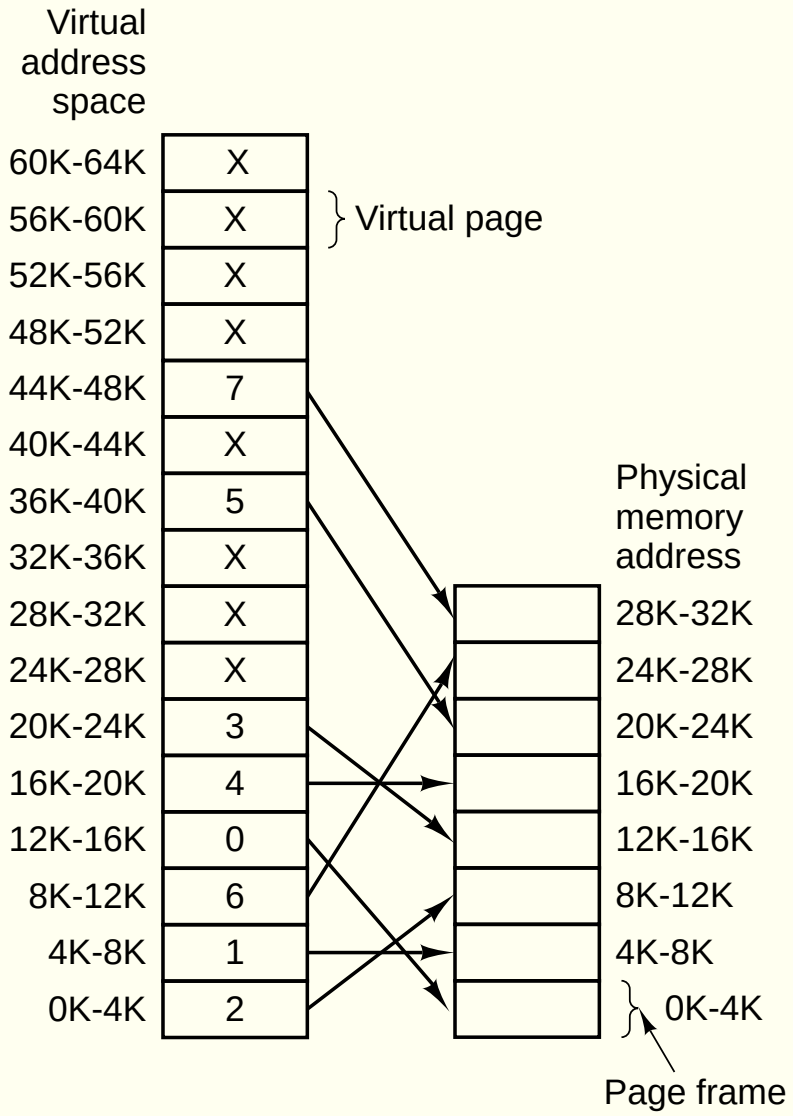
```
dados d1, d2, d3, d4, d5;

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

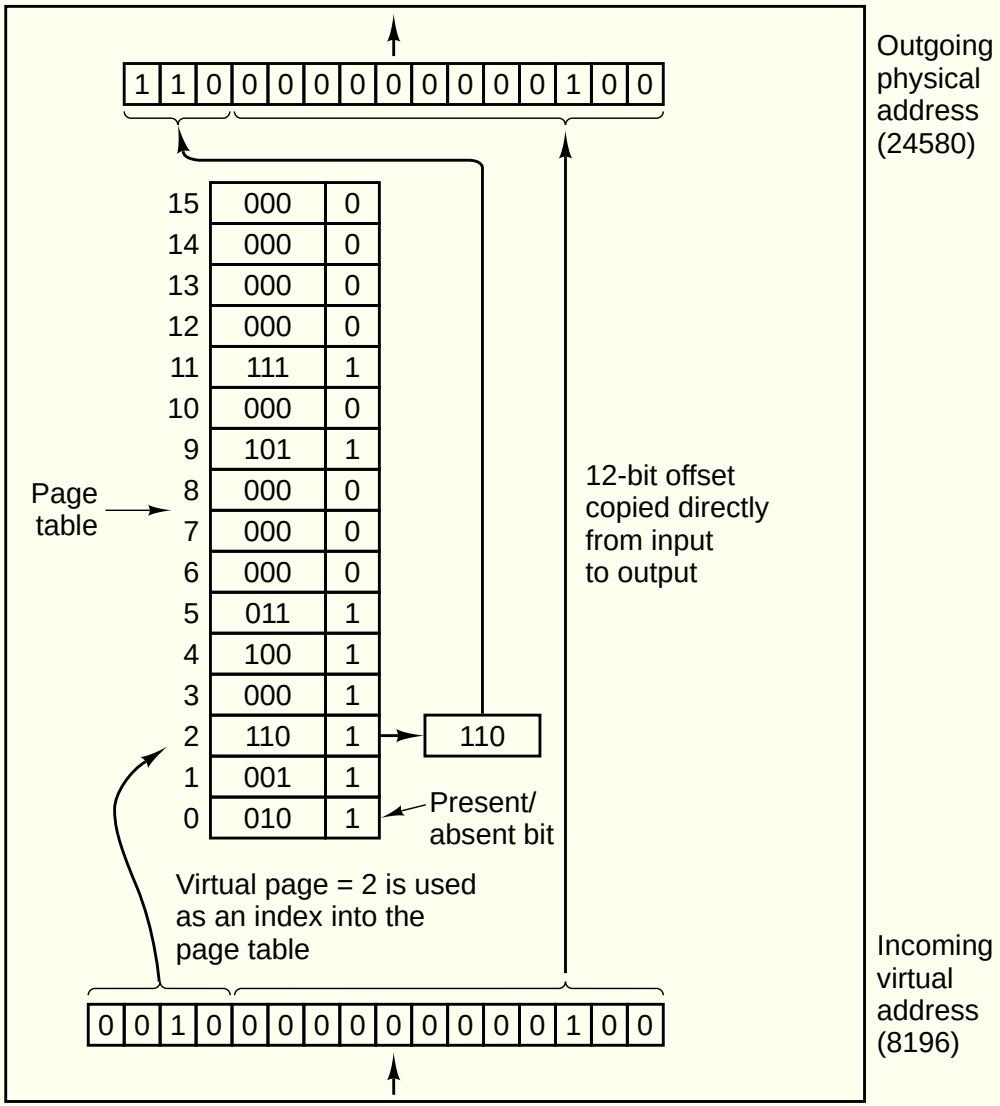



Memory Management Unit (MMU)

# Paginação

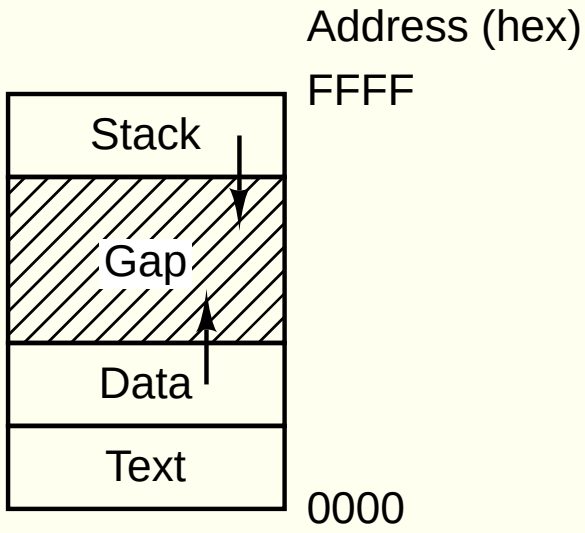

Virtual address space
60K-64K
56K-60K
52K-56K
48K-52K
44K-48K
40K-44K
36K-40K
32K-36K
28K-32K
24K-28K
20K-24K
16K-20K
12K-16K
8K-12K
4K-8K
0K-4K
X
X
X
X
7
X
5
X
X
X
3
4
0
6
1
2
Virtual page
Physical memory address
28K-32K
24K-28K
20K-24K
16K-20K
12K-16K
8K-12K
4K-8K
0K-4K
Page frame

# Mapeamento dos endereços

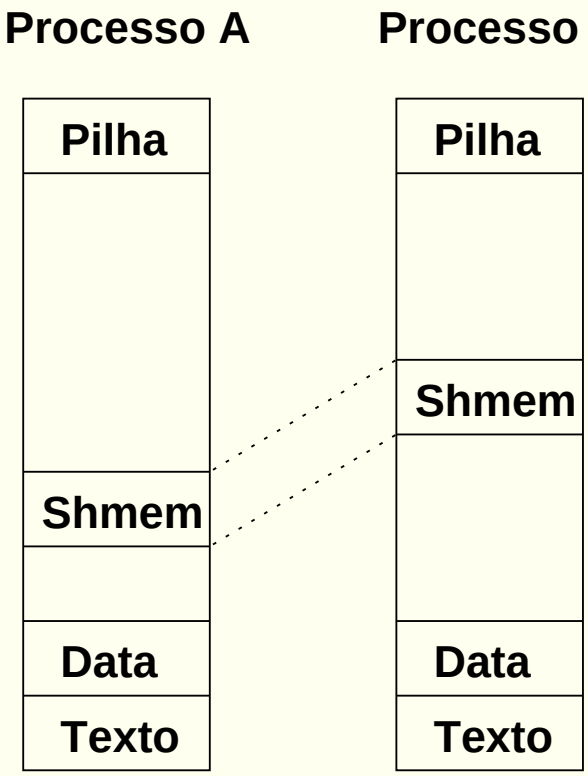


# Espaço de endereçamento

- Apenas as páginas ocupadas precisam ser mapeadas



# 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