Sistemas Operacionais

Aula 11 – Gestão de Memória – Alocação de Memória

Prof. Igor da Penha Natal

Material baseado no livro: MAZIERO, C. Sistemas Operacionais: Conceitos e Mecanismos. Editora da UFPR, 2019. 456 p. ISBN 978-85-7335-340-2.

Conteúdo

Alocação de memória

Alocar memória

Reservar áreas de memória para uso do SO e dos processos

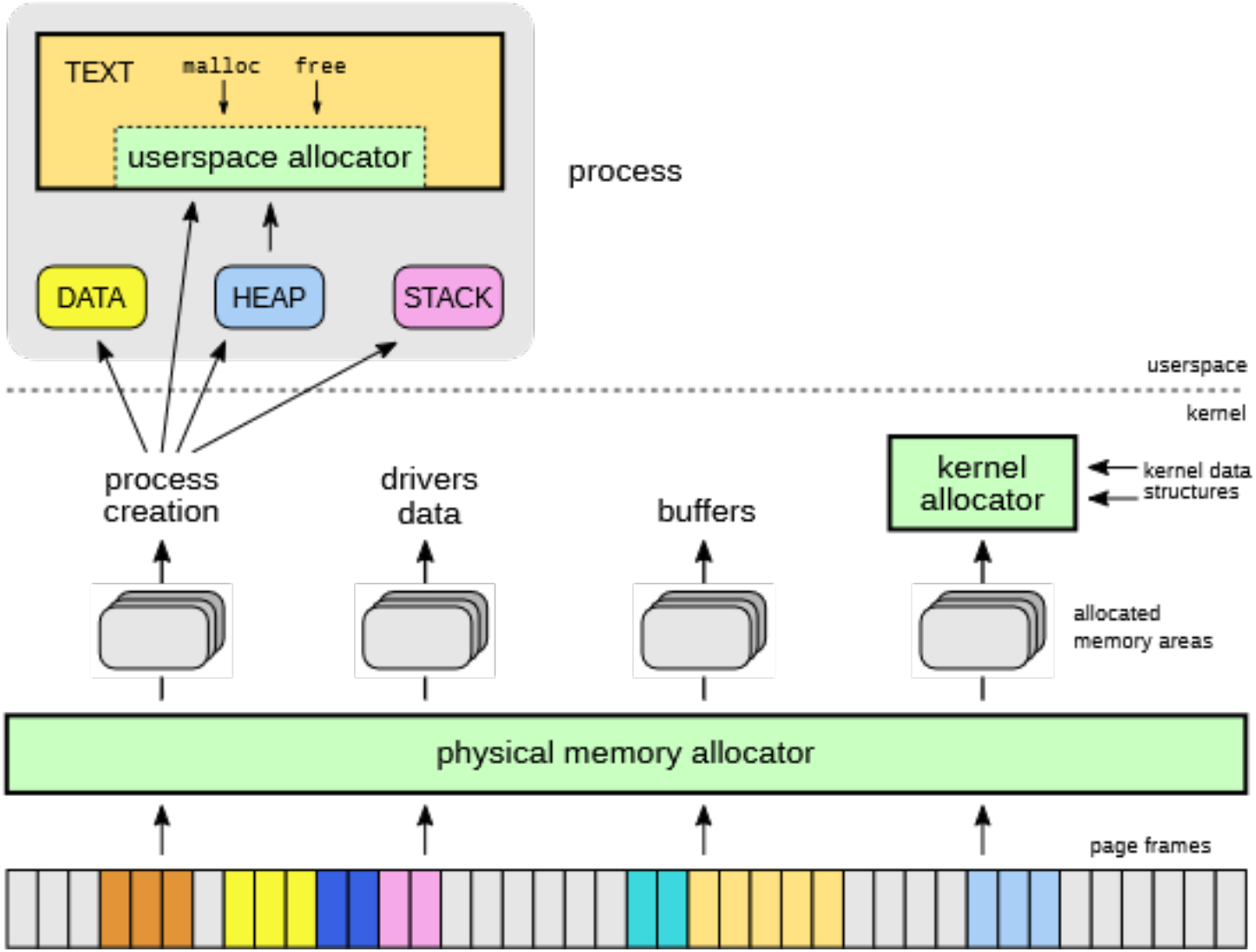
O **alocador** de memória:

- Atende solicitações do núcleo ou de processos
- Aloca e libera áreas de memória
- Gerencia que áreas estão livre ou ocupadas
- Deve ser rápido e eficiente (baixo desperdício de memória)

Níveis de alocação

- **Da memória física**:
  - Organiza a memória RAM (endereços físicos)
  - Separa as áreas alocáveis das áreas reservadas
  - Aloca áreas para carregar processos e para o núcleo
- **No espaço de núcleo**:
  - Obtém áreas do alocador físico
  - Aloca/libera estruturas de dados do núcleo
  - Sockets, semáforos, descritores de processos, ...
- **No espaço de usuário**:
  - Implementado em bibliotecas fora do núcleo
  - Gerencia a área *Heap* do processo
  - Atende requisições dos processos (`malloc`, `free`)

Alocação de memória



Alocação básica

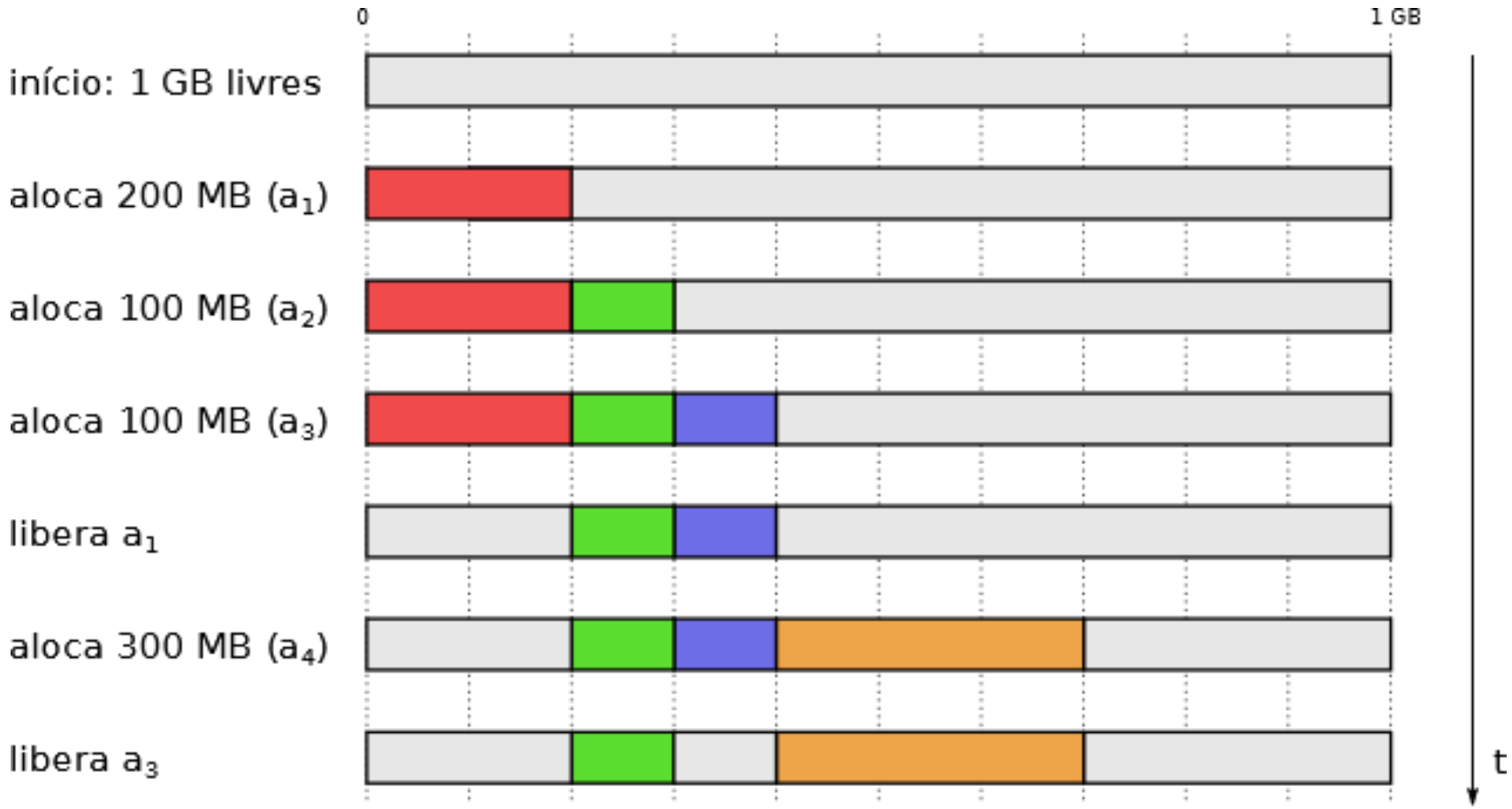
Problema:

- Gerenciar uma ou mais áreas de memória
- Atender pedidos de alocação e liberação de blocos
- Otimizar o uso da memória
- Evitar/minimizar a fragmentação

Estratégia:

- Manter o registro das áreas livres e ocupadas
- Listas, árvores, bitmaps

Alocação de memória



Fragmentação externa da memória

Fragmentação **externa**:

- “Buracos” entre áreas de memória alocadas
- Gerados pelas alocações e liberações de memória
- Reduz a capacidade de alocação
- Exige mais esforço de gerência

Resolução:

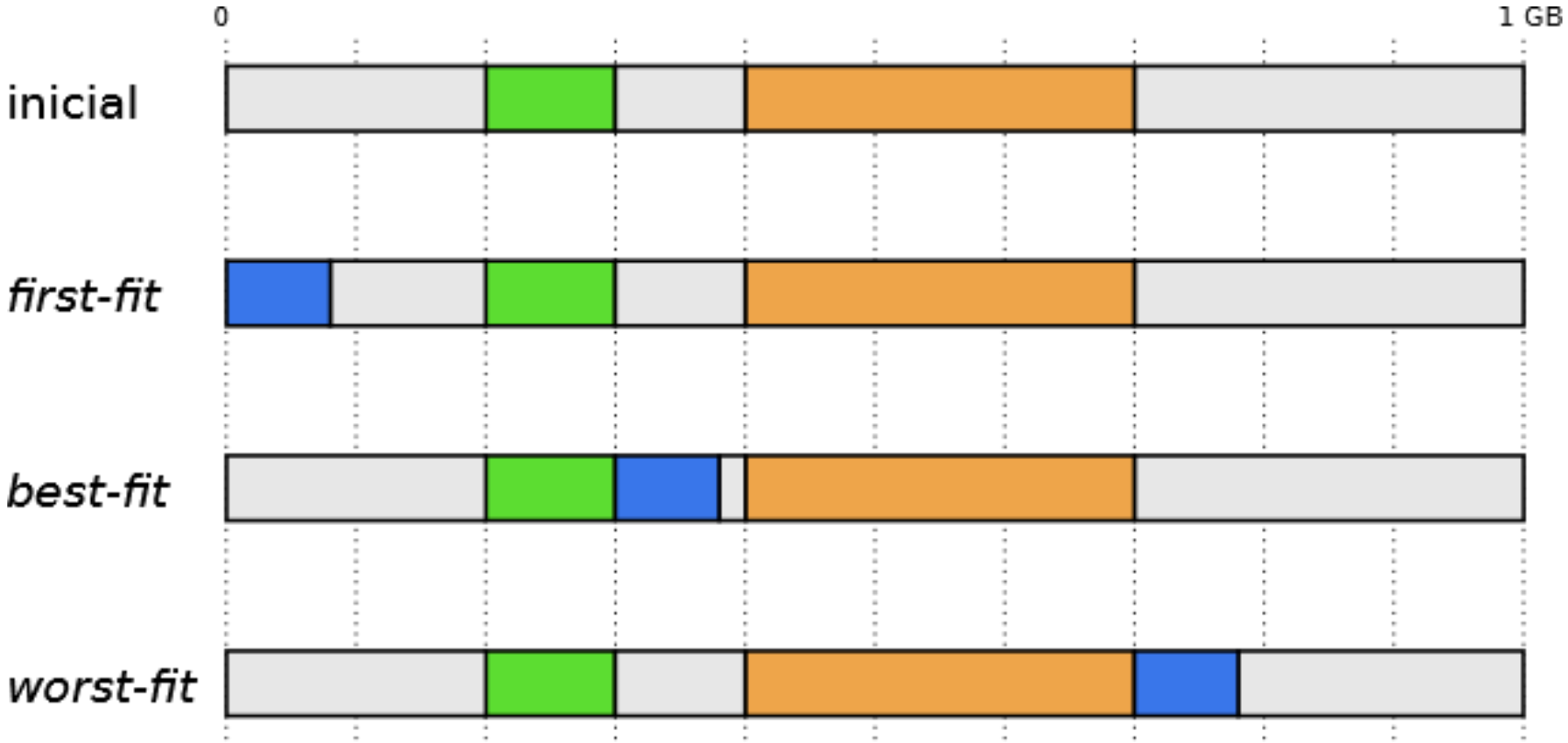
- Estratégias para minimizar a fragmentação
- Desfragmentação periódica

Estratégias de alocação

- ***Best-Fit***:
  - Escolher a **menor área** livre onde o bloco couber
  - Pode gerar fragmentos pequenos e inúteis
- ***Worst-Fit***:
  - Escolher sempre a **maior área** livre
  - Pode gerar escassez de áreas grandes
- ***First-Fit***:
  - Escolher a **primeira área** livre onde o bloco couber
  - Solução rápida
- ***Next-Fit***:
  - Variante da *first-fit*, para distribuir as alocações

Fragmentação externa

Precisamos alocar um novo bloco de 80 MB:



Qual seria a melhor estratégia?

Desfragmentação

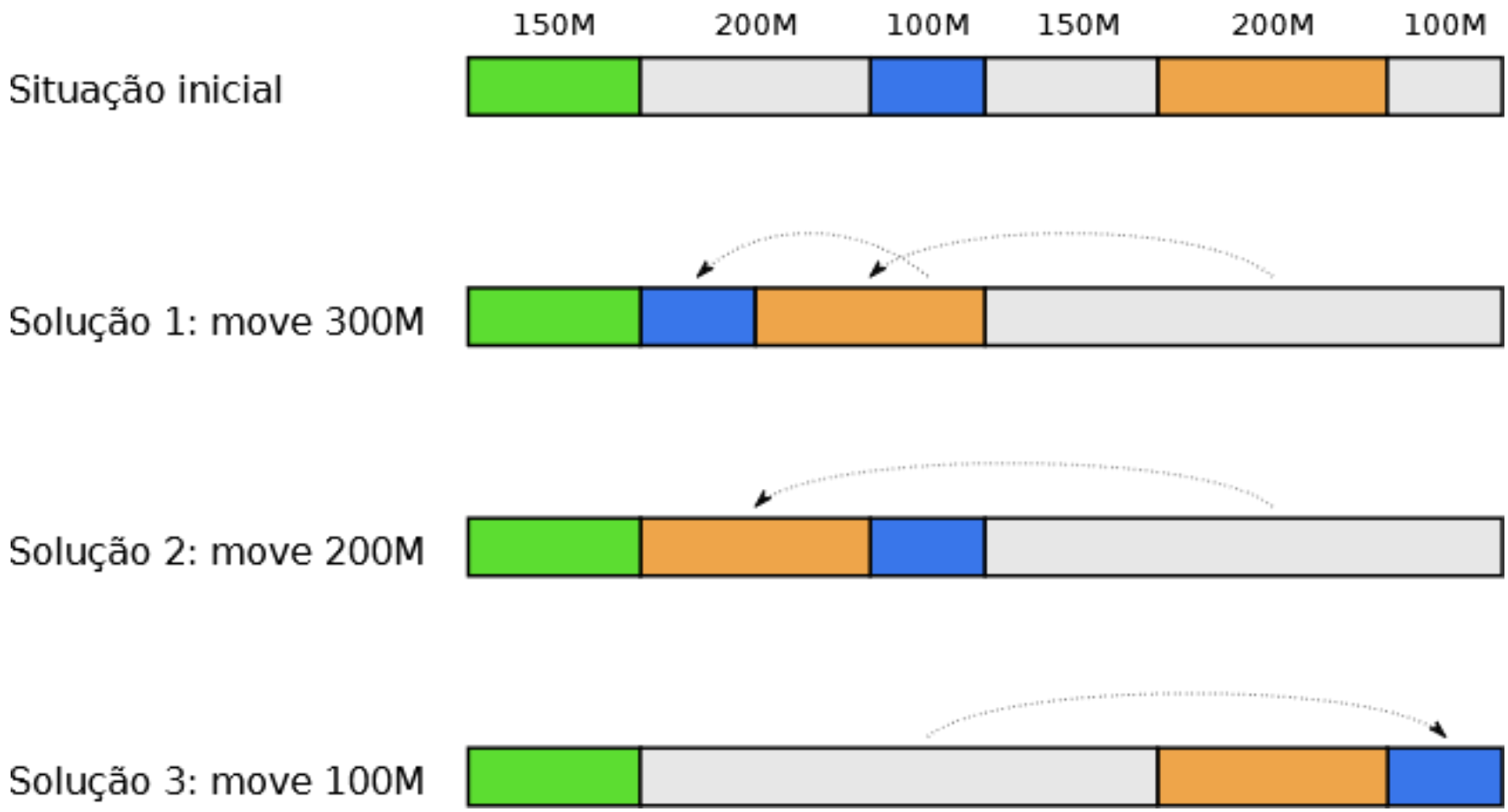
Princípio:

- Periodicamente, desfragmentar a memória
- Mover os blocos para concatenar fragmentos livres
- Liberar áreas maiores de memória

Problemas:

- Posição dos blocos na memória é alterada
  - Só pode ser aplicado à memória física (RAM)
- Deve ser rápido para não atrapalhar o sistema
- Algoritmos de otimização combinatória

Desfragmentação



Fragmentação interna

Arredondamento das solicitações de blocos de memória:

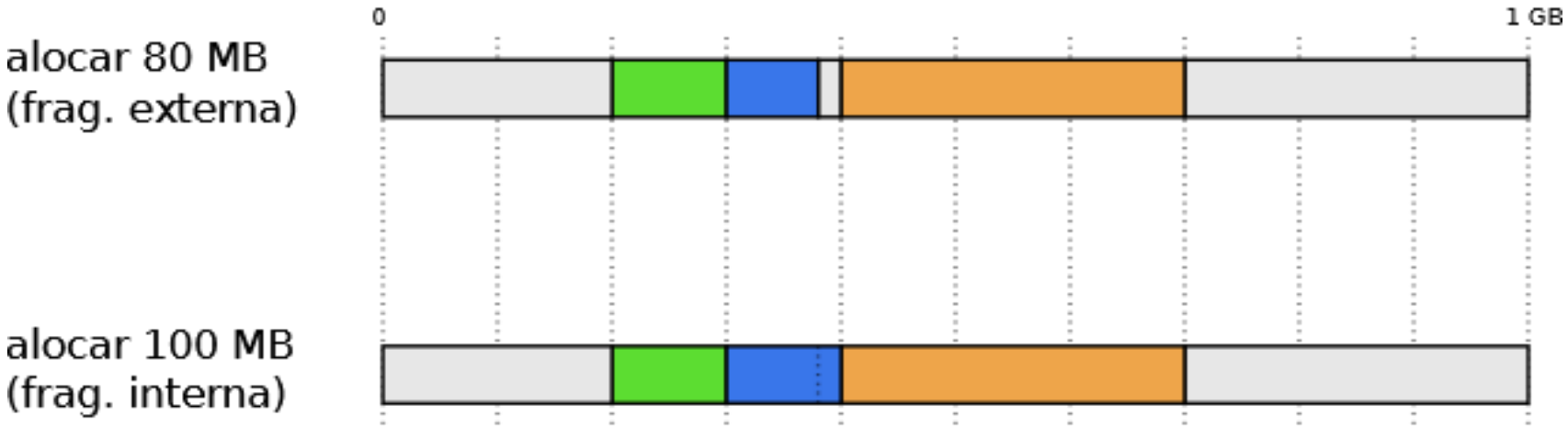
- Evitar o surgimento de fragmentos muito pequenos
- Diminuir custo de gerência
- Inevitável em sistemas paginados

Processos recebem mais memória que o solicitado:

- Essa memória adicional não será usada
- Perda média: 1/2 página por área alocada

Fragmentação interna

Exemplo: dada uma requisição de 80 MB de memória:



O alocador Buddy

Organiza os blocos alocados em uma árvore binária

Geralmente usado para alocação da memória física

Princípio:

- Alocar blocos de memória de tamanho 2^n
  - Exemplo: p/ pedido de 85 KB, alocar 128 KB (2^7 KB)
- Bloco mínimo: entre 1 KB e 64 KB
- Bloco máximo: a memória disponível

Funcionamento do alocador Buddy

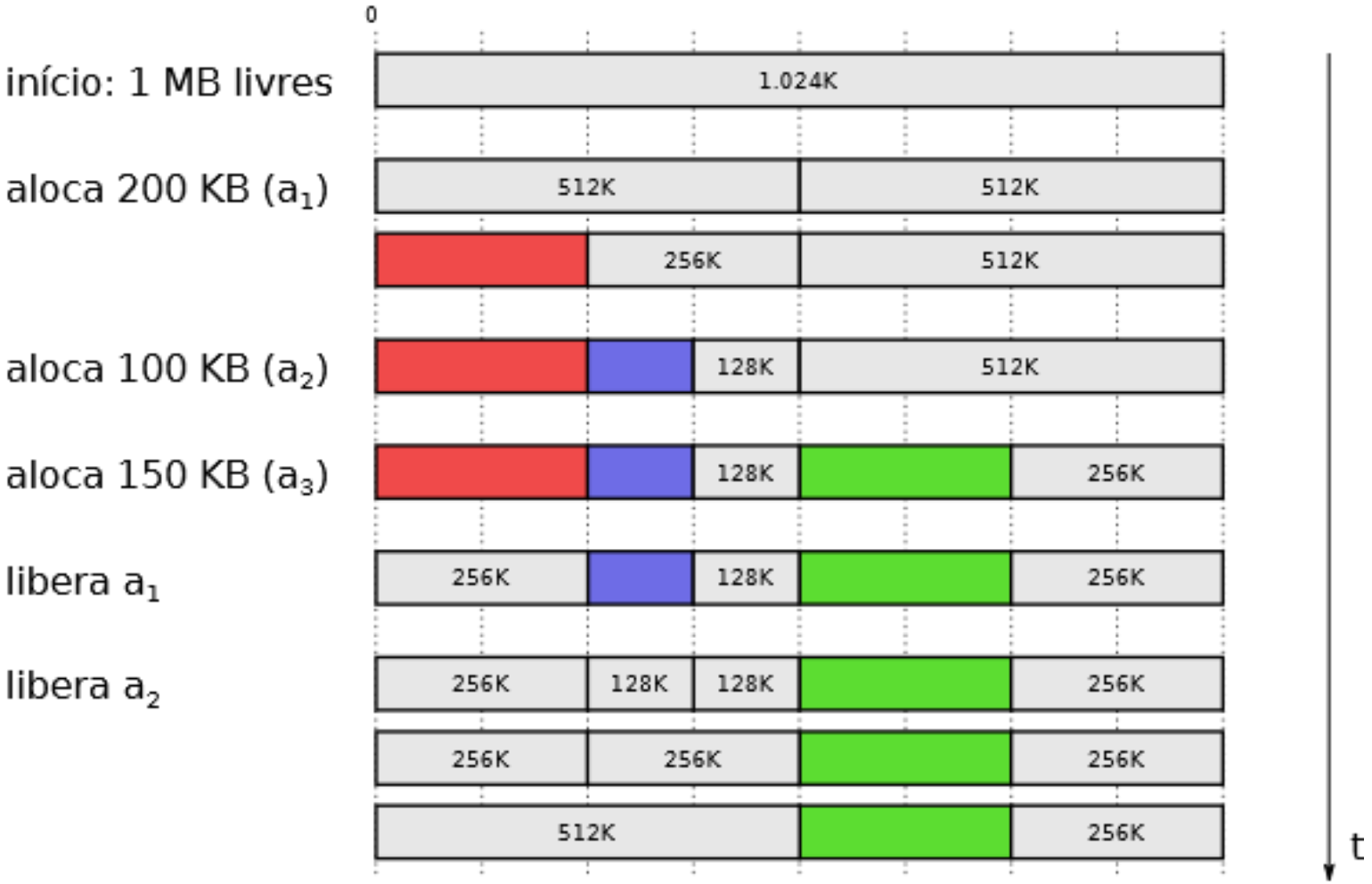
Ao alocar um bloco de 40 KB:

1. 2^n maior e mais próximo de 40 KB é 64 KB
2. procura bloco de 64 KB e o aloca
3. se não houver, procura bloco de 128 KB e divide em dois
4. se não houver, procura bloco de 256 KB e divide em dois
5. se não houver, ...

Ao liberar um bloco:

1. se o par (*buddy*) do bloco estiver livre, funde-os
2. se o par do novo bloco estiver livre, funde-os
3. se o par ...

O alocador Buddy



O alocador Buddy

Variantes do alocador Buddy:

- Buddy binário (sempre dois buddies iguais)
- Buddy binário com pesos (64 KB = 48KB + 16 KB)
- Buddy com Fibonacci (0,1, 1, 2, 3, 5, 8, 13, ...)

Exemplo: alocador de memória física no Linux:

```
# cat /proc/buddyinfo
Chunk size              4K     8K    16K   32K 64K 128K 256K 512K 1M 2M   4M
Node 0, zone     DMA     1      0      1     2   3     2     0     0  1  1    3
Node 0, zone   DMA32    12      8      9     7   6    10     6     6  4  3  551
Node 0, zone Normal   237  15443  11954  3798 984   444   289   133 70 37  126
```

O alocador Slab

Proposto para o sistema SunOS 5.4, amplamente usado hoje.

Especializado na alocação de "objetos" do núcleo:

- descritores de arquivos, processos, sockets, etc
- pequenos (10s-100s bytes) e com tamanhos padronizados
- continuamente criados/destruídos na operação do núcleo
- podem ser reutilizados ao invés de alocados/liberados

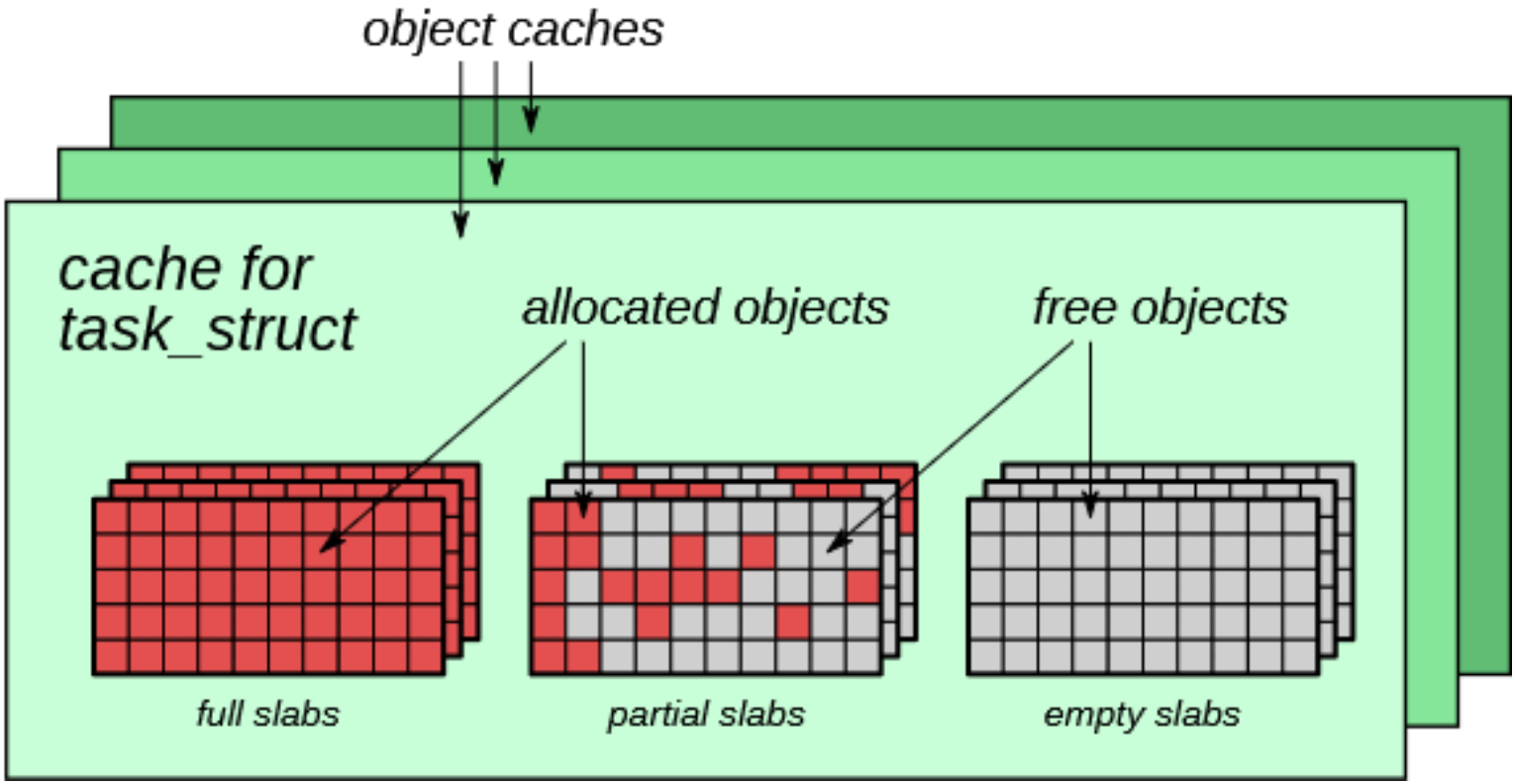
Estratégia básica: caching de objetos

- Cada tipo de objeto tem um cache
- Cada cache é dividido em *slabs* (placas)
- Cada *slab* pode estar cheio, vazio ou parcial

O alocador Slab

Composto de caches de objetos, cada um com vários *slabs*.

Cada *slab* contém objetos em uso ou livres.



Funcionamento do alocador Slab

1. Ao receber um pedido de objeto, o alocador analisa o cache adequado e entrega um objeto livre de um *slab* parcial;
2. se não houver *slab* parcial, entrega um objeto livre de um *slab* vazio e muda o status dele para parcial;
3. se não houver *slab* vazio, solicita RAM ao alocador físico para um novo *slab*, ajusta seu status para vazio, inicializa seus objetos e os marca como livres;
4. ao liberar um objeto, este é marcado como livre;
5. se todos os objetos de um *slab* estiverem livres, ele é marcado como vazio;
6. *slabs* vazios podem ser devolvidos ao alocador físico se houver necessidade de RAM no sistema.

O alocador Slab no Linux

```
# cat /proc/slabinfo

# name            <active_objs> <num_objs> <objsize> <obj/slab> <pg/slab>
RAWv6                   252          252        1152          28          8
UDPv6                   104          104        1216          26          8
fat_inode_cache          44           44         744          22          4
fat_cache                 0            0          40         102          1
pid_namespace            76           76         208          19          1
posix_timers_cache      374          374         240          17          1
request_sock_TCP          0            0         304          26          2
TCP                     224          224        2048          16          8
sock_inode_cache       2783         2783         704          23          4
file_lock_cache         160          160         200          20          1
inode_cache           18902        18902         608          26          4
mm_struct               207          225        2112          15          8
files_cache             230          230         704          23          4
signal_cache            346          368        1024          16          4
task_struct            1141         1185        5824           5          8
...
```

Alocação no espaço de usuário

Userspace allocator:

- Gerencia o uso da área *Heap* do processo
- Alocação dinâmica de variáveis da aplicação
- Implementa operações como `malloc` e `free`
- Implementado por biblioteca (LibC)

Abordagens:

- A mais simples: *first-fit* ou *best-fit*
- *DLmalloc* e *PTmalloc* (usados no Linux)
- Soluções ad-hoc para aplicações específicas
  - *Memory pool*: prealocar vetor de blocos iguais