ORGANIZAÇÃO FÍSICA DE UM BANCO DE DADOS PARTE 1

Sérgio Mergen

Sumário

- Camadas de Abstração de um SGBD
- Meio de Armazenamento
- Transferência de dados
  - Alguns cenários

Camadas de Abstração de um SGBD

modelo conceitual

modelo lógico

modelo físico

Camadas de Abstração de um SGBD

modelo conceitual

modelo lógico

modelo físico

Camadas de Abstração de um SGBD

- Modelo conceitual
  - Independente de tipo de SGBD
  - Registra
    - Quais tipos de dados importam e como eles são relacionados
  - Não registra
    - Como estes dados são armazenados a nível de SGBD

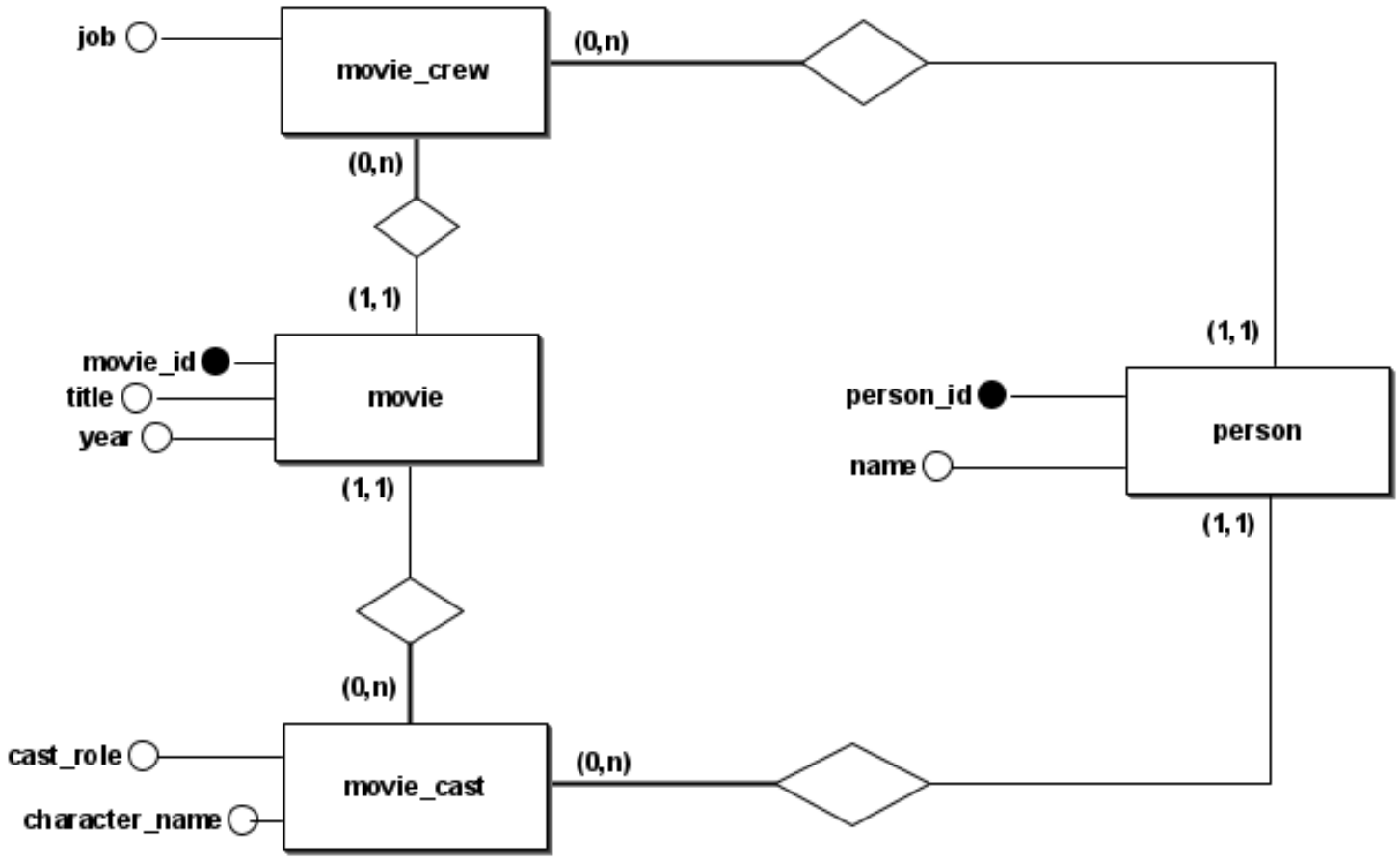
Camadas de Abstração de um SGBD

- Técnica mais difundida de modelagem conceitual
  - Abordagem entidade-relacionamento (ER)

- Principais construtores da modelagem ER
  - Entidades
  - Relacionamentos
  - Atributos

Camadas de Abstração de um SGBD

Diagrama entidade-relacionamento



Camadas de Abstração de um SGBD

modelo conceitual

modelo lógico

modelo físico

Camadas de Abstração de um SGBD

- Modelo Lógico
  - Nível de abstração visto pelo usuário do SGBD
  - Dependente do tipo particular de SGBD que está sendo usado

Camadas de Abstração de um SGBD

```
create table movie
(
    movie_id      int    not null,
    title         varchar(255),
    year          int,
    primary key(movie_id)
);
```

```
create table person
(
    person_id     int    not null,
    name          varchar(255),
    primary key(person_id )
);
```

```
create table movie_cast
(
    movie_id        int    not null,
    person_id       int    not null,
    cast_order      int,
    character_name  varchar(255),
    primary key(movie_id, person_id),
    foreign key movie_id references movie(movie_id ),
    foreign key person_id references person(person_id)
);
```

```
create table movie_crew
(
    movie_id        int    not null,
    person_id       int    not null,
    job             varchar(255),
    primary key(movie_id, person_id),
    foreign key movie_id references movie(movie_id ),
    foreign key person_id references person(person_id)
);
```

Camadas de Abstração de um SGBD

Tabela: movie

movie_id	title	year
13	Forrest Gump	1994
680	Pulp Fiction	1994

Tabela: person

person_id	title
62	Bruce Willis
138	Quentin Tarantino
518	Danny DeVito

Tabela: movie_cast

movie_id	person_id	character_name	cast_order
680	62	Butch Coolidge	3
680	138	Jimmie Dimmick	10

Tabela: movie_crew

movie_id	person_id	job
680	138	Director
680	518	Executive Producer

Camadas de Abstração de um SGBD

modelo conceitual

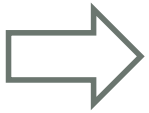
modelo lógico

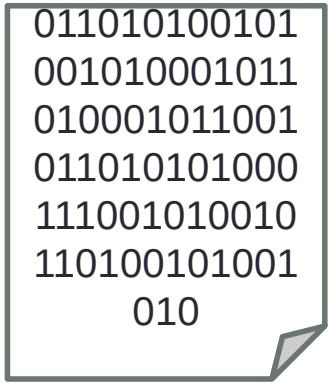
modelo físico

Camadas de Abstração de um SGBD

- Modelo físico
  - Se preocupa com o armazenamento dos dados propriamente dito

movie_id	title	year
13	Forrest Gump	1994
680	Pulp Fiction	1994





Modelo lógico

Modelo físico

Camadas de Abstração de um SGBD

- Como as tabelas são armazenadas em memória secundária?
  - Em um arquivo?
  - Em vários arquivos?
  - Qual o formato desses arquivos?
  - Como é feito o acesso aos dados?

- Para responder essas perguntas, é necessário estudar o modelo físico do banco de dados

Camadas de Abstração de um SGBD

- O modelo físico contém detalhes de armazenamento interno de informações
- Detalhes que
  - não têm influência sobre a programação de aplicações no SGBD
  - Mas têm influência no desempenho da aplicações
- Usados por profissionais que fazem *sintonia* de performance em banco de dados
- Uma das principais preocupações de quem estuda o modelo físico é entender e reduzir o custo no acesso aos dados

Sumário

- Camadas de Abstração de um SGBD
- Meio de Armazenamento
- Transferência de dados
  - Alguns cenários

Meio de Armazenamento

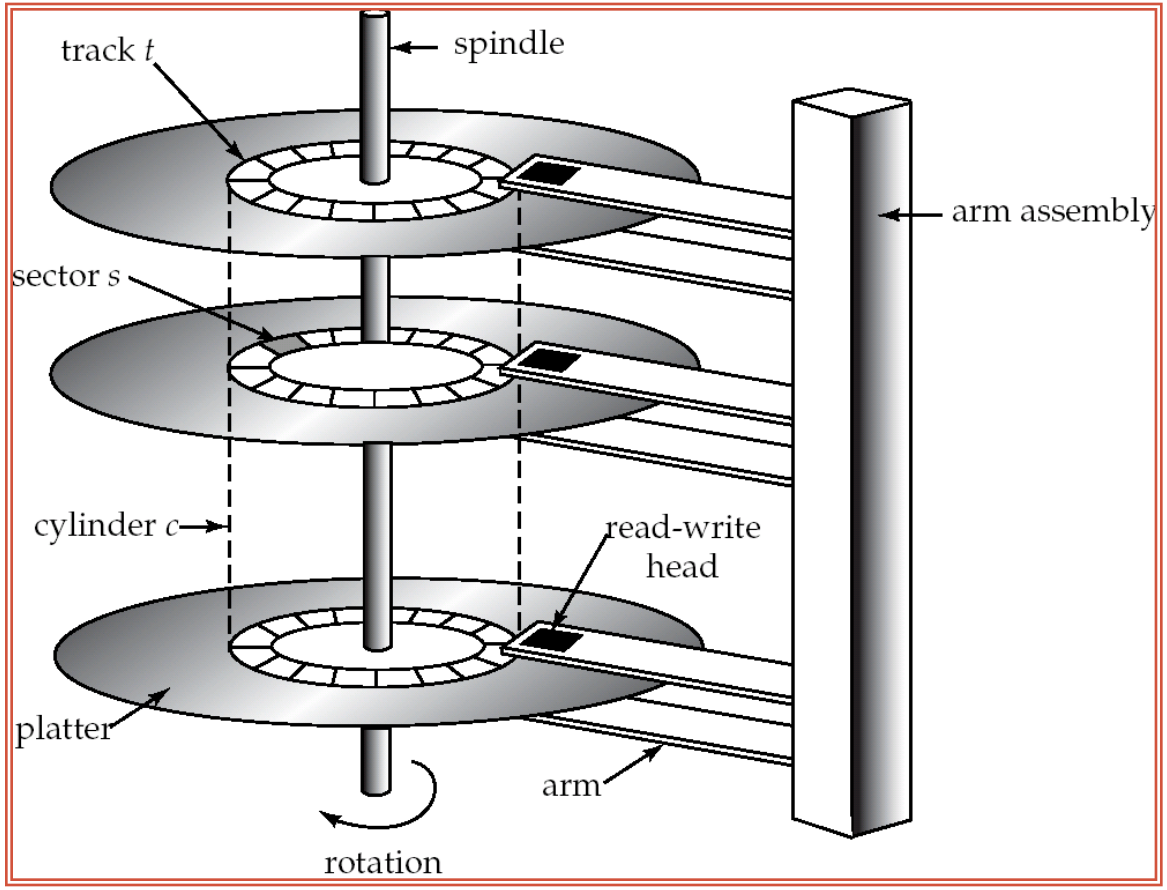
- Os dados podem ser armazenados em discos magnéticos
  - Meio preferido para o armazenamento não volátil de bancos de dados de longo prazo.

- Os próximos slides explicam como funciona o processo de leitura/gravação em discos magnéticos

Disco magnético

- **Características**
  - Requer transferência para a memória
    - Dados precisam ser transferidos para a memória principal para manipulação, e escritos de volta em disco para armazenamento
      - Muito mais lento do que o acesso a memória principal
  - Acesso direto
    - possibilita ler dados do disco em qualquer posição

Disco magnético



Disco magnético

- Cabeçote de escrita-leitura
  - Posicionada bem próxima da superfície do prato (quase o tocando)
  - Lê e escreve dados codificados magneticamente.

- Superfície dos pratos divididos em trilhas circulares
  - Cerca de 50K-100K trilhas por prato em discos rígidos convencionais

Disco magnético

- Cada trilha é dividida em setores
  - Um setor é a menor unidade de dados que pode ser lida ou escrita.
  - O tamanho de um setor é tipicamente de **512 bytes**
  - Número típico de setores por trilha: 500 (em trilhas internas) até 1000 (em trilhas mais externas)

Disco magnético

- Para ler/escrever em um setor
  - O braço do disco se move para posicionar o cabeçote na trilha correta
  - O prato gira; dados são lidos/escritos a medida que o setor passa pelo cabeçote
- Montagem dos cabeçotes do disco
  - Múltiplos pratos em um único mecanismo (usualmente de 1 até 5)
  - Um cabeçote por prato, montado em um braço comum.
- Cilindro *i* consiste na *i*-ésima trilha em todos os pratos

Disco magnético

- Como visto, discos magnéticos utilizam um processo mecânico para acessar informação
  - lento
- Existem tecnologias mais eficientes, baseadas em processos totalmente eletrônicos
  - Ex. SSDs
- Empresas estão passando a migrar a mídia de armazenamento para SSDs
- No entanto, SSDs ainda são mais caros e tem um tempo de vida útil menor.
  - Por isso, a transição de tecnologia é lenta

Sumário

- Camadas de Abstração de um SGBD
- Meio de Armazenamento
- Transferência de dados
  - Alguns cenários

Transferência de dados

- O código abaixo mostra uma das formas de ler/gravar dados do disco
- Chamadas envolvidas
  - **seek:** localiza uma parte do arquivo com base no numero de bytes que se deseja deslocar a partir do início
  - **read:** lê uma sequência de bytes
  - **write:** escreve uma sequência de bytes

```
RandomAccessFile raf = new RandomAccessFile(fileName, "rw");
...
byte[] bytes = new byte[5];
raf.seek(2000);
raf.read(bytes);
raf.seek(5000);
raf.write(bytes);
```

Transferência de dados

- O que acontece por baixo dos panos quando ocorre uma leitura/escrita?

```
RandomAccessFile raf = new RandomAccessFile(fileName, "rw");
...
byte[] bytes = new byte[5];
raf.seek(2000);
raf.read(bytes);
raf.seek(5000);
raf.write(bytes);
```

Transferência de dados

- A leitura/escrita a partir de um disco passa por uma cadeia de operações, envolvendo
  - Sistema operacional
  - Sistema de arquivos
  - Controlador do disco

Transferência de dados - Sistema de arquivos

- Um disco é formado por partições
- Cada partição possui um sistema de arquivos
- O sistema de arquivos gerencia os arquivos armazenados em uma partição, indicando em que partes do disco o arquivo está localizado
  - O arquivo pode estar fragmentado em várias partes

Transferência de dados - Sistema de arquivos

- Entre outras coisas, o sistema de arquivos determina o tamanho do **cluster**
  - Uma sequência contígua de setores da unidade de armazenamento
- O tamanho de um cluster vai de 512 bytes até muitos kb
  - Determinado no momento da formatação do meio físico (ntfs, fat32, ...)
- Um arquivo pode ocupar vários clusters
  - Mas um cluster só pode ser ocupado por um arquivo
  - Por isso, o tamanho do arquivo sempre é um múltiplo do tamanho do cluster

Transferência de dados - Sistema de arquivos

- A escolha do tamanho do cluster deve ser feita de forma criteriosa

- Clusters pequenos (maior fragmentação)
  - O mesmo arquivo pode estar fragmentado em clusters espalhados pelo disco
  - Requer mais transferências de disco

- Clusters grandes (menor fragmentação)
  - maior desperdício de espaço devido ao preenchimento parcial dos clusters
  - Um arquivo de 2kb em um cluster de 16kb leva a um desperdício de 14kb de espaço que não podem ser usados para outra coisa

- Clusters típicos são de **4** kb

Transferência de dados - Sistema operacional

- As requisições a dados são efetuadas a partir de chamadas de sistema
  - Chamadas para funções do kernel do sistema operacional
    - Seek, read, write, ...
- O SO usa o sistema de arquivos para localizar o(s) cluster(s) onde estão os dados solicitados

Transferência de dados - Sistema operacional

- Os dados do arquivo são carregados para páginas da memória RAM
- O tamanho de cada página é predominantemente definido pela arquitetura do processador
- A escolha do tamanho da página também é criteriosa
  - Páginas muito pequenas = mais páginas
    - Maior o overhead de controle das páginas em uso
  - Páginas muito grandes = menos páginas
    - Menos processos podem estar em execução simultaneamente
- Tamanho típico: 4kb

Transferência de dados - Sistema operacional

- As páginas podem residir em dois espaços
  - Espaço do kernel: usado pelo kernel do sistema operacional
  - Espaço do usuário: usado por um processo executado por um usuário

- As páginas guardam vários tipos de conteúdo
  - Código sendo executado
  - Dados sendo processados
  - Dados carregados de arquivo (caso que nos interessa)

Transferência de dados - Sistema operacional

- Os dados carregados do arquivo são primeiro copiados para páginas no espaço do kernel
  - Para então serem copiados para o espaço do usuário
- Cópia para o espaço do kernel
  - Normalmente, o SO copia dados do arquivo até encher uma página, mesmo que menos dados tenham sido solicitados
    - Isso visa otimizar o uso das páginas
- Tipicamente, teremos páginas de 4kb e clusters de 4kb
  - se a leitura for alinhada a esse tamanho, haverá um mapeamento de um para um entre página e cluster

Transferência de dados - Controlador do disco

- Controlador do disco
  - Hardware que faz a interface entre o dispositivo de armazenamento e o Sistema Operacional

- Caso os setores solicitados pelo SO não estejam em memória, eles são requisitados ao controlador do disco

Transferência de dados - Controlador do disco

- O tempo total envolvido no envio de dados para o SO é dividido em tempo de acesso e tempo de transferência

- **Tempo de acesso:** o tempo que leva desde a solicitação de leitura/escrita até o momento em que a transferência começa de fato.
  - **Tempo de busca (seek):** tempo que leva para reposicionar o braço na trilha correta.
  - **Latência rotacional:** tempo que leva para o setor a ser acessado aparecer debaixo do cabeçote
- **Taxa de transferência:** a taxa com que os dados são recuperados ou armazenados no disco.
- Custo para transferir um bloco é bem menor do que o custo para acessá-lo

Transferência de dados - Controlador do disco

- O tempo para acessar setores do disco é muito alto
  - Associado a forma mecânica como esses dados são localizados (seek)
- Para reduzir esse custo, os controladores de disco usam um cache
- Esse cache armazena informações que serão lidas para a memória ou efetivamente gravadas no disco
- O tamanho varia
  - De 8Mb a 256Mb (em discos magnéticos)
  - Até 4gb (em SSDs)

Transferência de dados - Controlador do disco

- Usa Read-ahead (ou prefetching) para agilizar a leitura
  - Setores vizinhos são guardados no cache mesmo que não tenham sido solicitados
  - Caso sejam solicitados no futuro, estarão disponíveis sem que seja necessário fazer um seek no disco

- Obs.
  - O SO também implementa um mecanismo de read-ahead para armazenar clusters vizinhos na memória principal

Transferência de dados

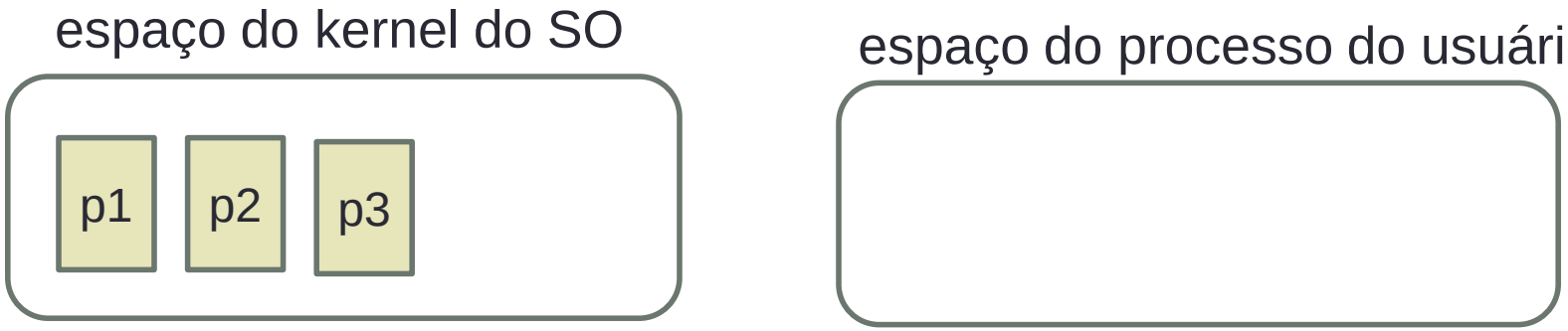
Visão simplista de como funciona a transferência de dados

- Carga dos dados
  - Programa solicita dados (bytes) para o SO
  - O SO usa o sistema de arquivos para localizar o(s) cluster(s) correspondente(s)
  - Caso os dados já estejam em alguma página do kernel, são recuperados
  - Caso contrário, uma nova página deve ser carregada
- Carga da página
  - SO solicita um (ou mais) cluster para o controlador do disco
  - Caso o cluster já esteja no buffer do disco, ele é recuperado
  - Caso contrário, um novo seek deve ser feito (custoso)
- Obs. os dados podem estar espalhados em diversos clusters
  - Isso exigiria um esforço maior para localização dos clusters

Sumário

- Camadas de Abstração de um SGBD
- Meio de Armazenamento
- Transferência de dados
  - Alguns cenários

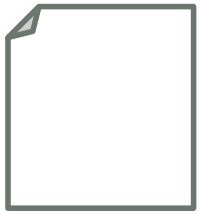
MEMÓRIA VIRTUAL



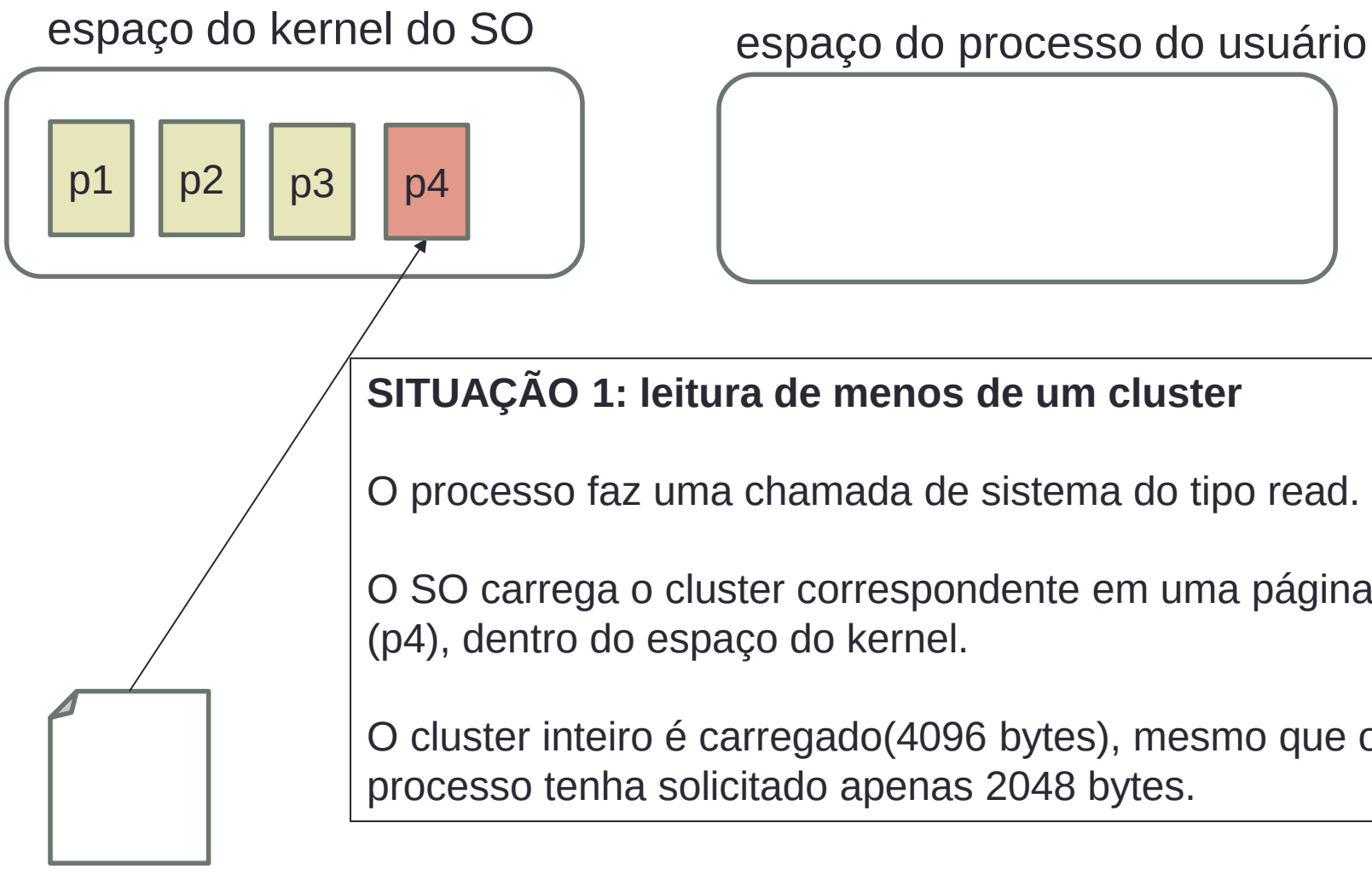
SITUAÇÃO 1: leitura de menos de um cluster

O processo deseja ler 2048 bytes a partir da posição 131.072.

Essa parte do arquivo está em um cluster que ainda não foi carregado para a memória.



MEMÓRIA VIRTUAL



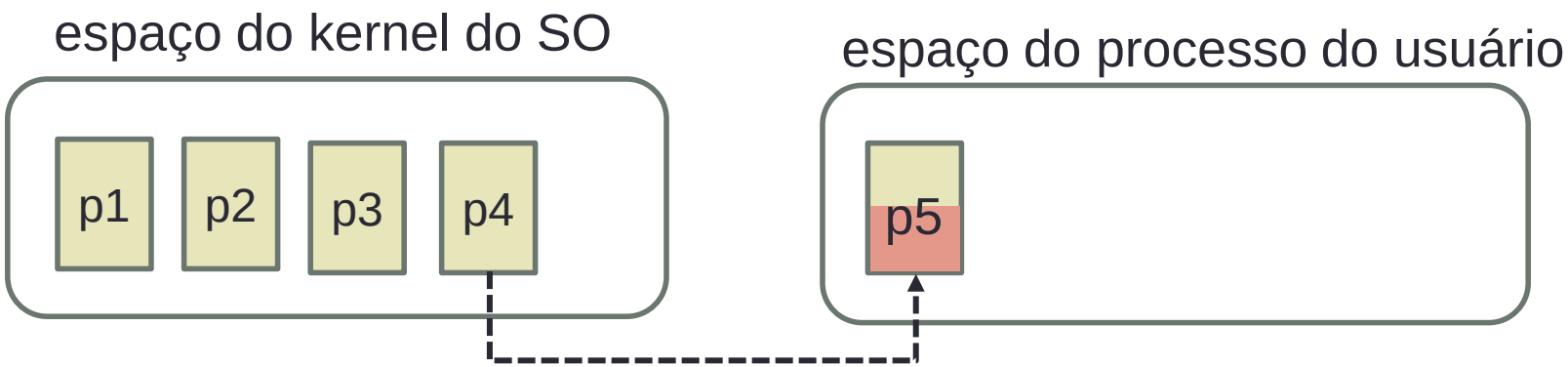
SITUAÇÃO 1: leitura de menos de um cluster

O processo faz uma chamada de sistema do tipo read.

O SO carrega o cluster correspondente em uma página (p4), dentro do espaço do kernel.

O cluster inteiro é carregado(4096 bytes), mesmo que o processo tenha solicitado apenas 2048 bytes.

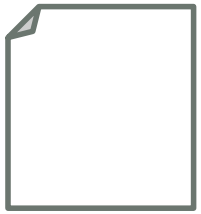
MEMÓRIA VIRTUAL



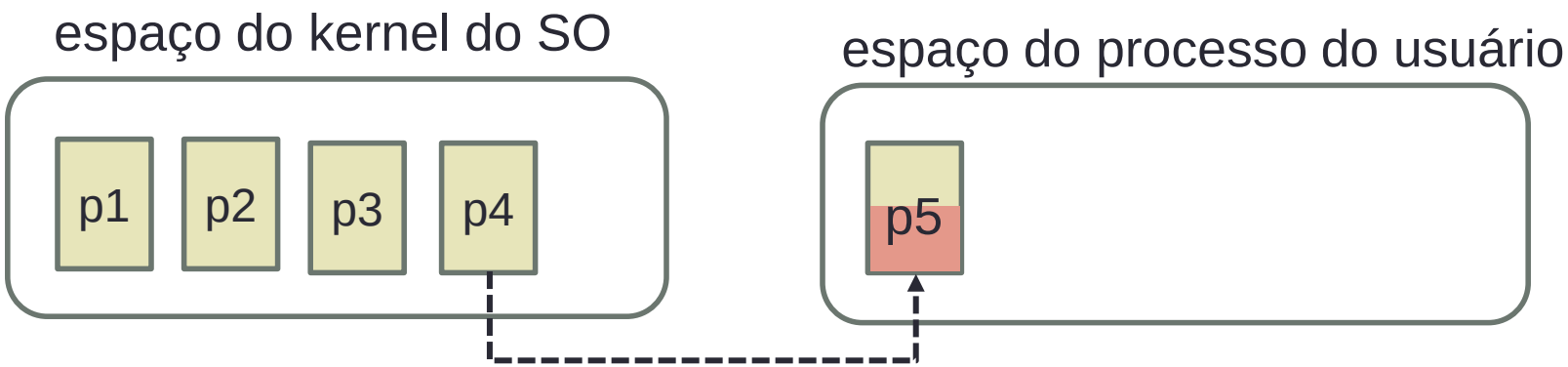
SITUAÇÃO 1: leitura de menos de um cluster

O SO copia os 2048 bytes requisitados para o espaço do processo.

Observe que apenas metade da página foi utilizada.

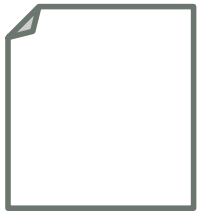


MEMÓRIA VIRTUAL

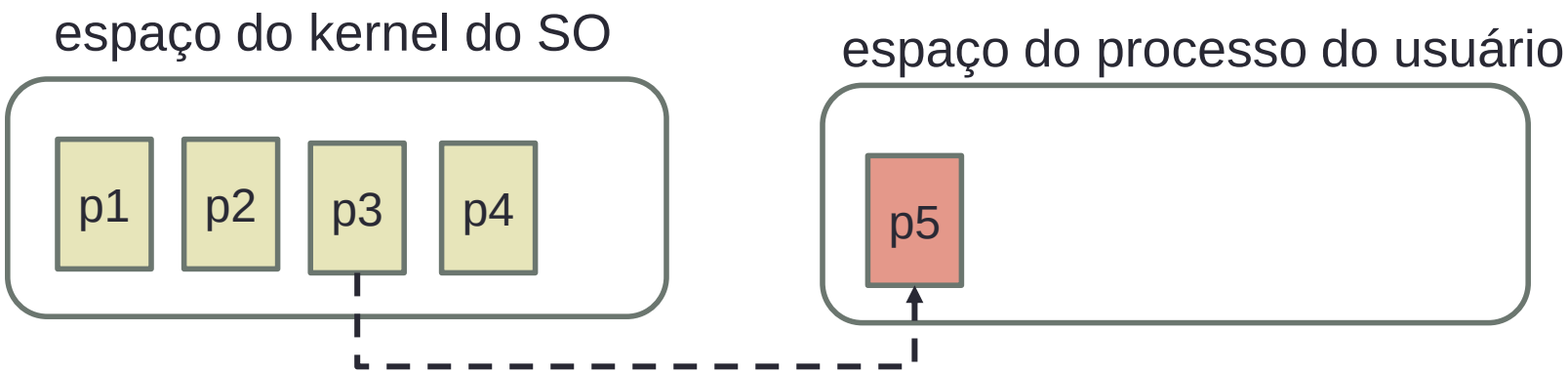


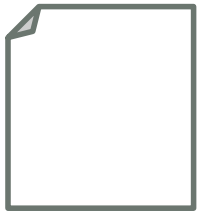
SITUAÇÃO 1: leitura de menos de um cluster

DICA: Como todo o conteúdo do cluster é carregado, é melhor estruturar o programa de modo que ele consuma 4096 bytes, para aproveitar ao máximo os dados em memória.



MEMÓRIA VIRTUAL





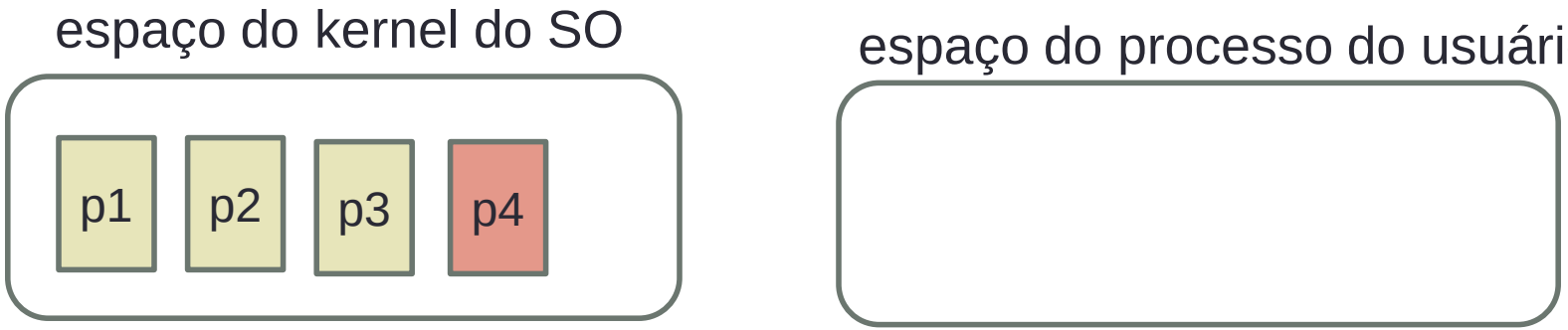
SITUAÇÃO 2: Uso de dados já presentes no kernel

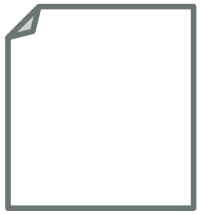
Neste outro exemplo, o processo solicitou 4096 bytes de um cluster que já está na memória (p3).

Nesse caso, ocorre apenas uma cópia dos dados para a área do processo.

É o melhor cenário, pois não foi necessário carregar dados do disco.

MEMÓRIA VIRTUAL





SITUAÇÃO 2: Uso de dados já presente no kernel

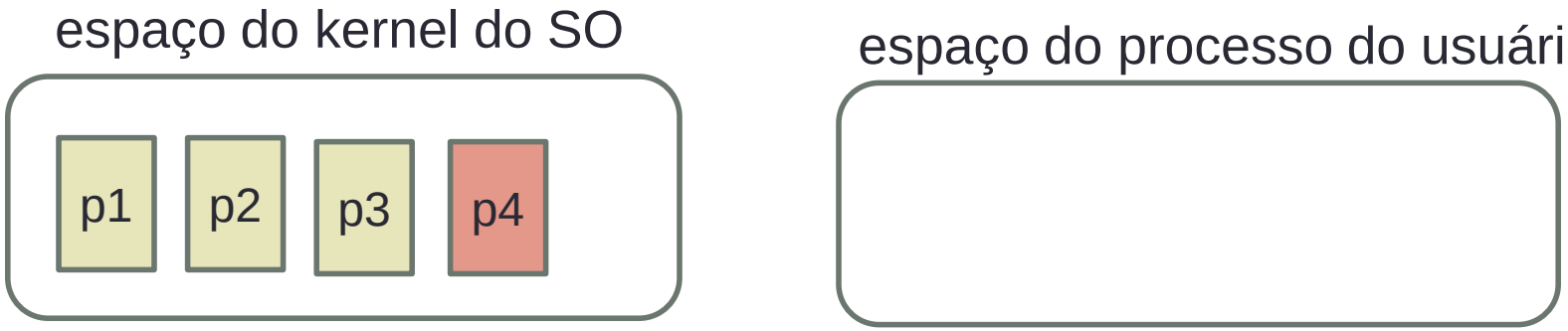
Após uso, a memória usada pelo processo foi descartada.

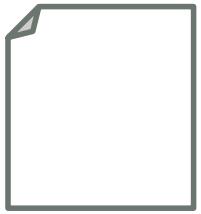
Mas a página já carregada no espaço do kernel continua disponível.

- desde que o SO não tenha decidido descartá-la.

Caso o processo precise dos mesmos dados novamente, será possível obtê-los sem recorrer ao disco.

MEMÓRIA VIRTUAL



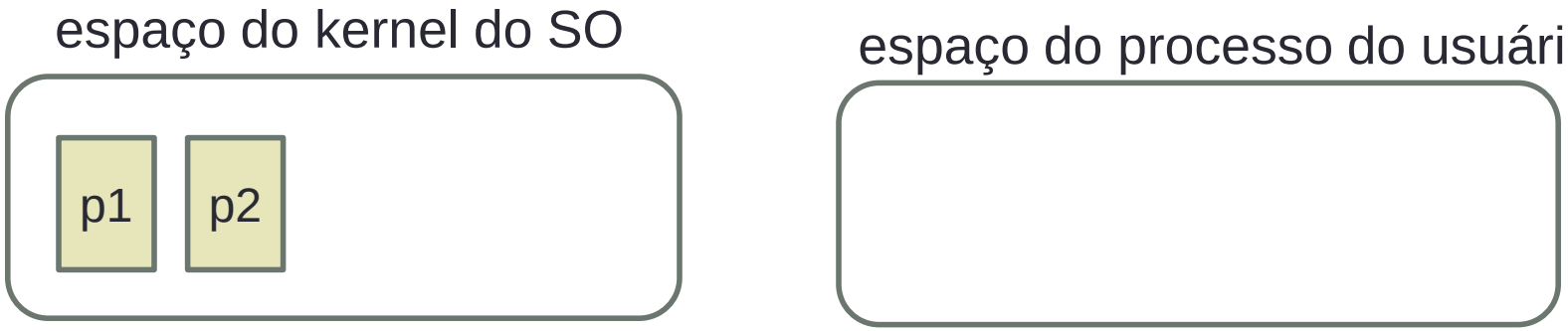


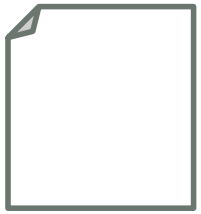
SITUAÇÃO 2: Uso de dados já presente no kernel

DICA: Procure manter os dados mais requisitados próximos, de modo a que eles possam ser encontrados no mesmo cluster.

Isso aumente as chances de que esses dados já estejam na memória.

MEMÓRIA VIRTUAL





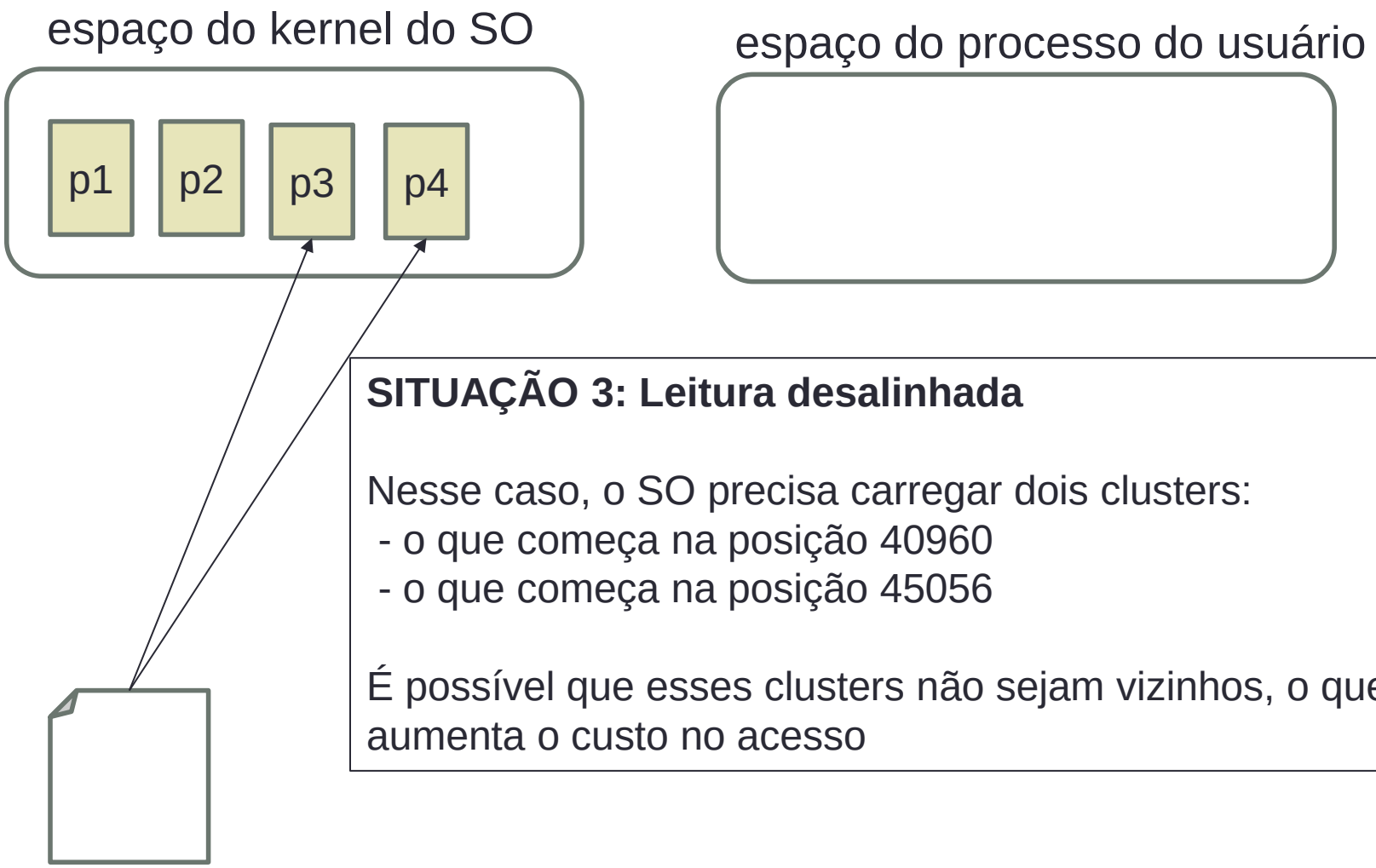
SITUAÇÃO 3: Leitura desalinhada

Agora o processo gerou um read de 4096 bytes a partir da posição 43.800

Essa posição está na metade de um cluster.

Cluster 100: 40960
Cluster 101: 45050

MEMÓRIA VIRTUAL

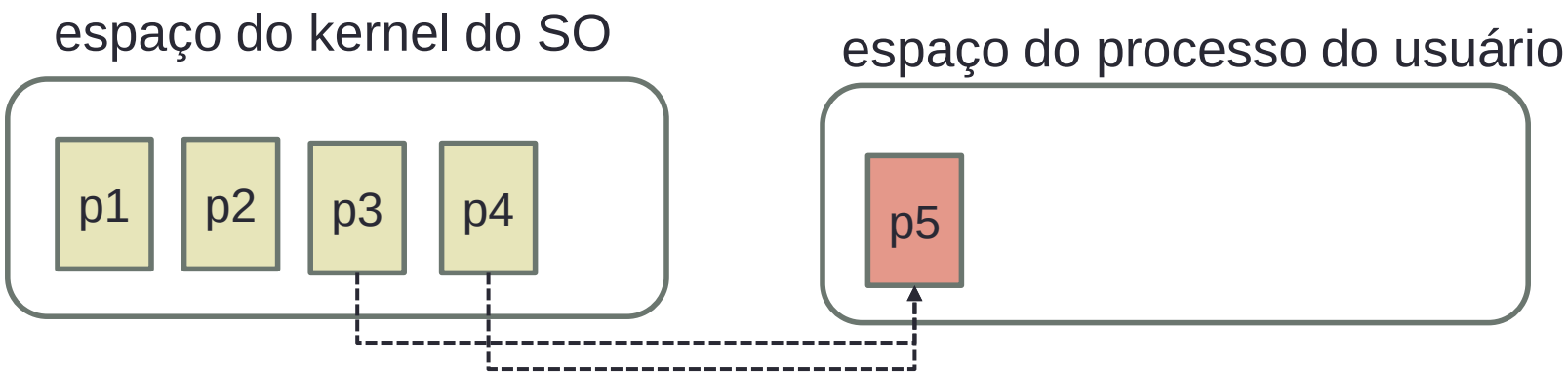


SITUAÇÃO 3: Leitura desalinhada

Nesse caso, o SO precisa carregar dois clusters:
- o que começa na posição 40960
- o que começa na posição 45056

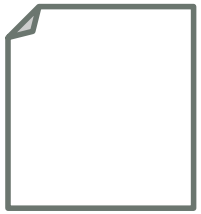
É possível que esses clusters não sejam vizinhos, o que aumenta o custo no acesso

MEMÓRIA VIRTUAL

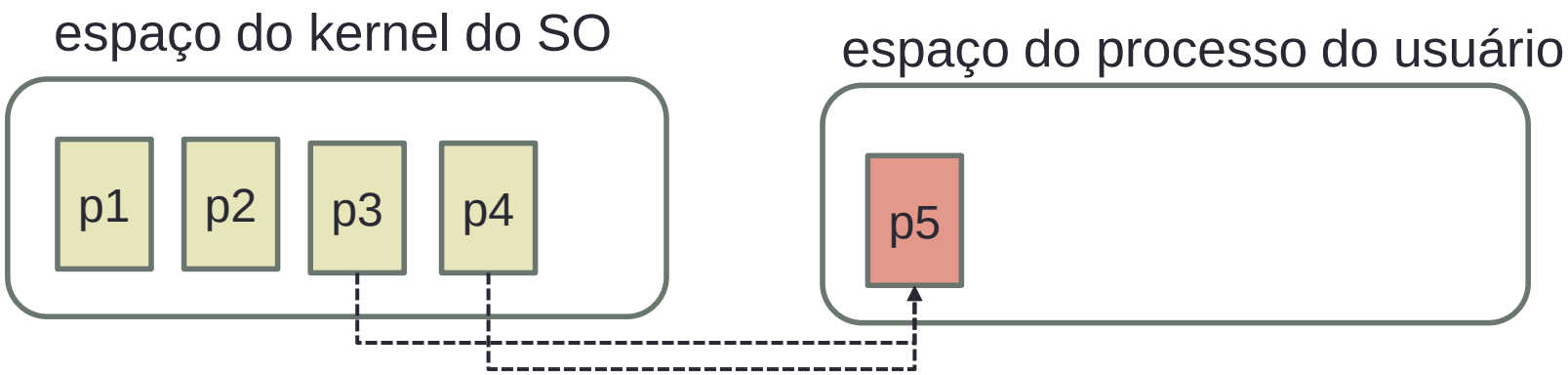


SITUAÇÃO 3: Leitura desalinhada

As porções solicitadas são carregadas na página do usuário, preenchendo todos os 4096 bytes da página.

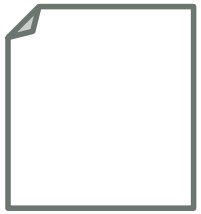


MEMÓRIA VIRTUAL



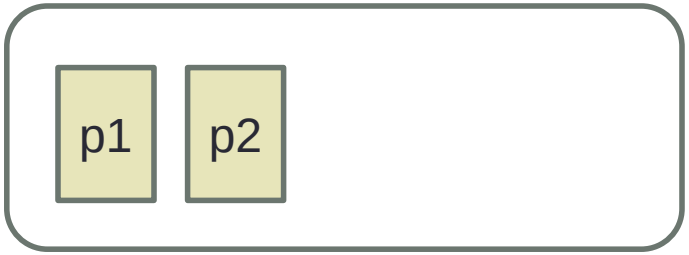
SITUAÇÃO 3: Leitura desalinhada

DICA: para evitar o carregamento de clusters adicionais, certifique-se de que as solicitações estejam alinhadas com os clusters.

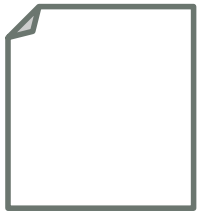


MEMÓRIA VIRTUAL

espaço do kernel do SO



espaço do processo do usuário



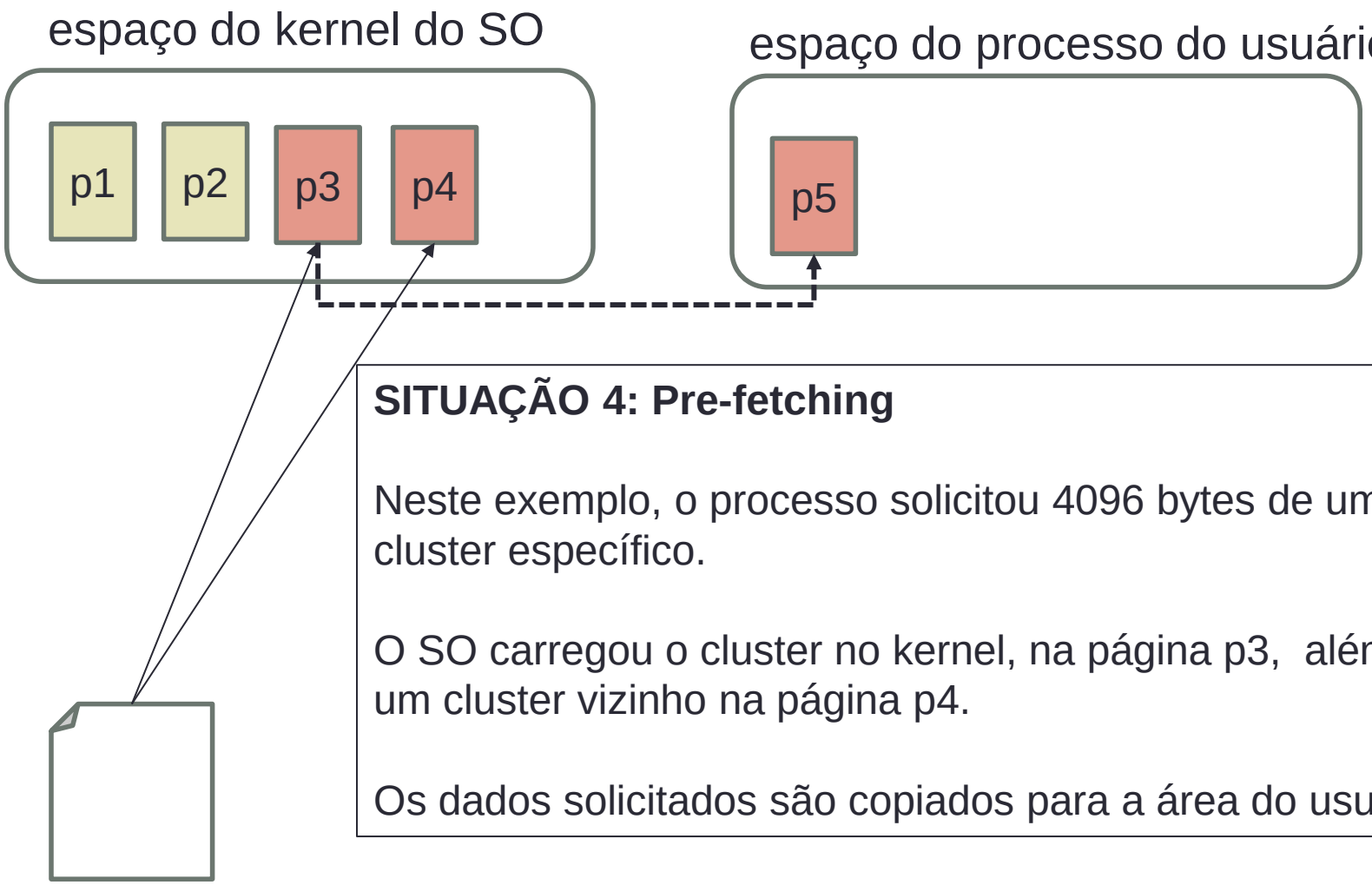
SITUAÇÃO 4: Pre-fetching

Quando o SO carrega um cluster requisitado pelo processo, ele também pode carregar clusters vizinhos que não tenham sido solicitados (pre-fetching).

Como os clusters são vizinhos no disco, a carga deles é mais barata.

O pre-fetching é a aposta do SO de que as próximas requisições irão solicitar esses clusters vizinhos.

MEMÓRIA VIRTUAL



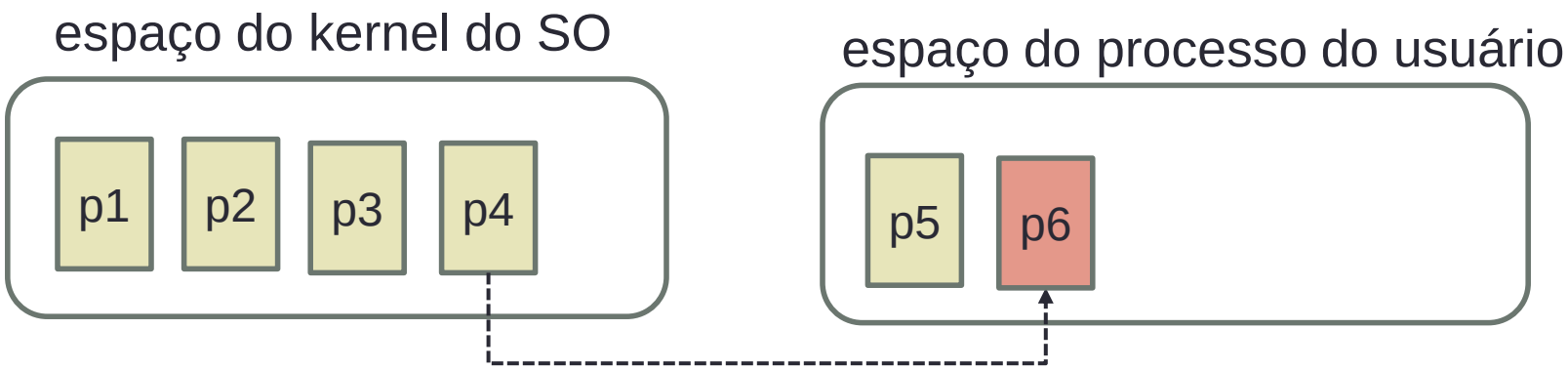
SITUAÇÃO 4: Pre-fetching

Neste exemplo, o processo solicitou 4096 bytes de um cluster específico.

O SO carregou o cluster no kernel, na página p3, além de um cluster vizinho na página p4.

Os dados solicitados são copiados para a área do usuário.

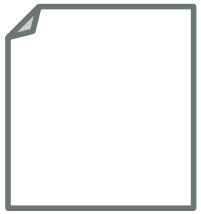
MEMÓRIA VIRTUAL



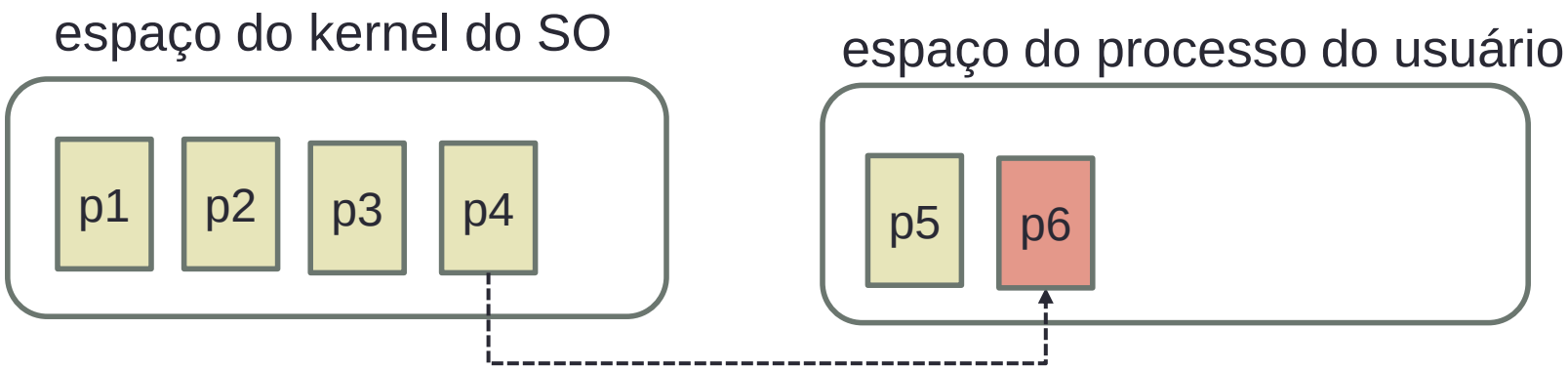
SITUAÇÃO 4: Pre-fetching

Caso o cluster vizinho ao anterior seja solicitado, será possível obtê-lo a partir da memória.

Mesmo que o kernel não possua a página solicitada, ainda há a possibilidade de que o controlador de disco tenha o cluster salvo em sua cache



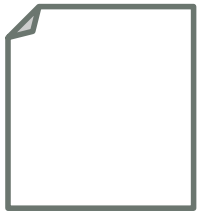
MEMÓRIA VIRTUAL



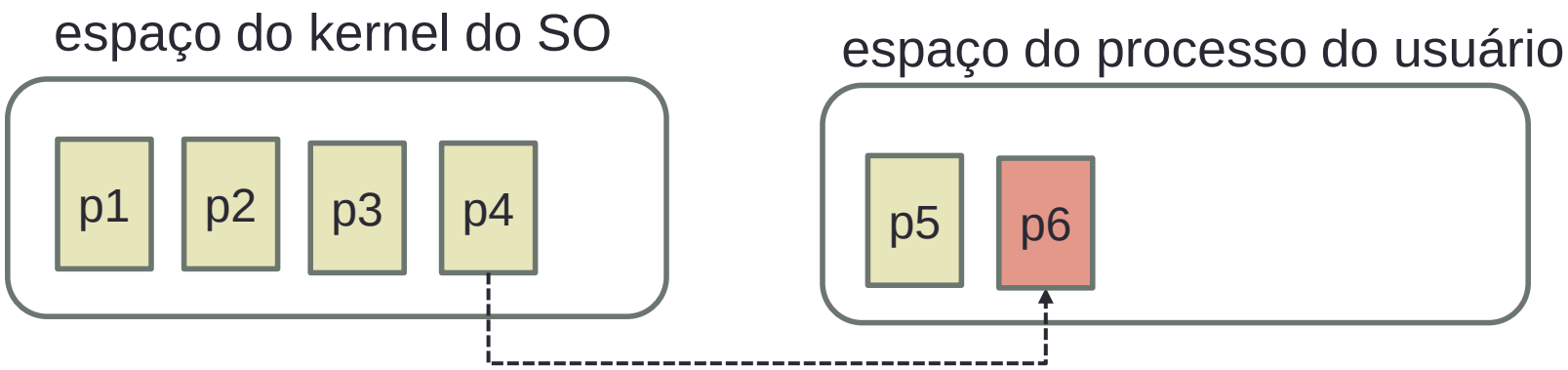
SITUAÇÃO 4: Pre-fetching

DICA: Procure manter os dados mais requisitados próximos, de modo que eles possam ser encontrados em clusters vizinhos.

Isso aumenta as chances de que esses dados já estejam na memória.

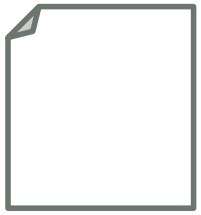


MEMÓRIA VIRTUAL

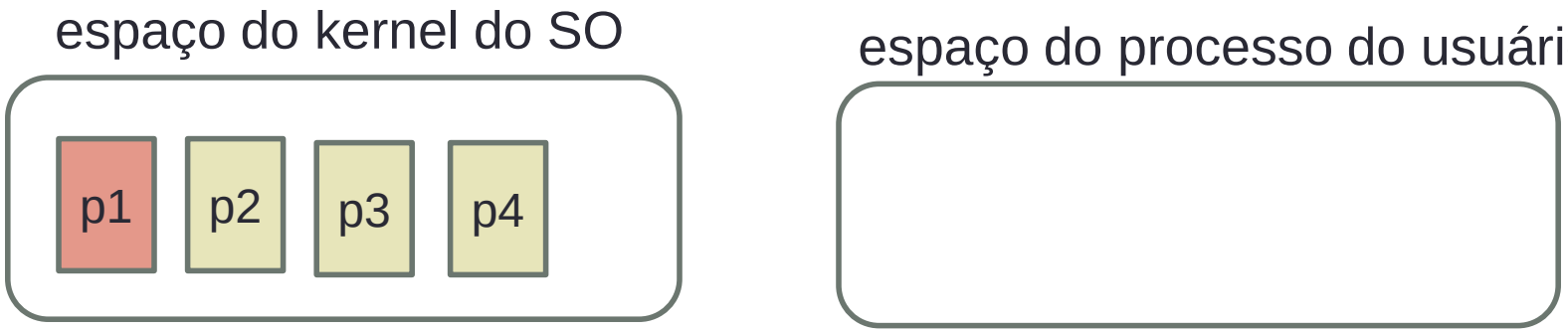


SITUAÇÃO 4: Pre-fetching

DICA: Além disso, convém desfragmentar o disco de tempos em tempos, para que partes próximas do arquivo sejam realmente vizinhos no disco



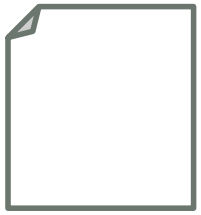
MEMÓRIA VIRTUAL



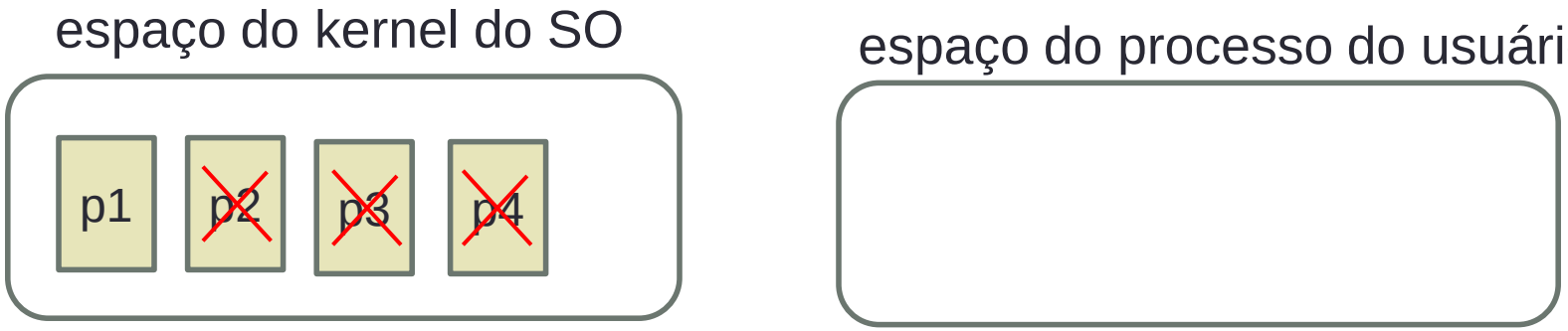
SITUAÇÃO 5: Leitura de múltiplos clusters

Neste exemplo, o processo solicitou 16384 bytes de quatro clusters específicos.

Um cluster já está na memória (p1). Mas os demais precisam ser carregados.



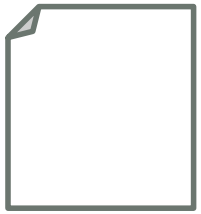
MEMÓRIA VIRTUAL



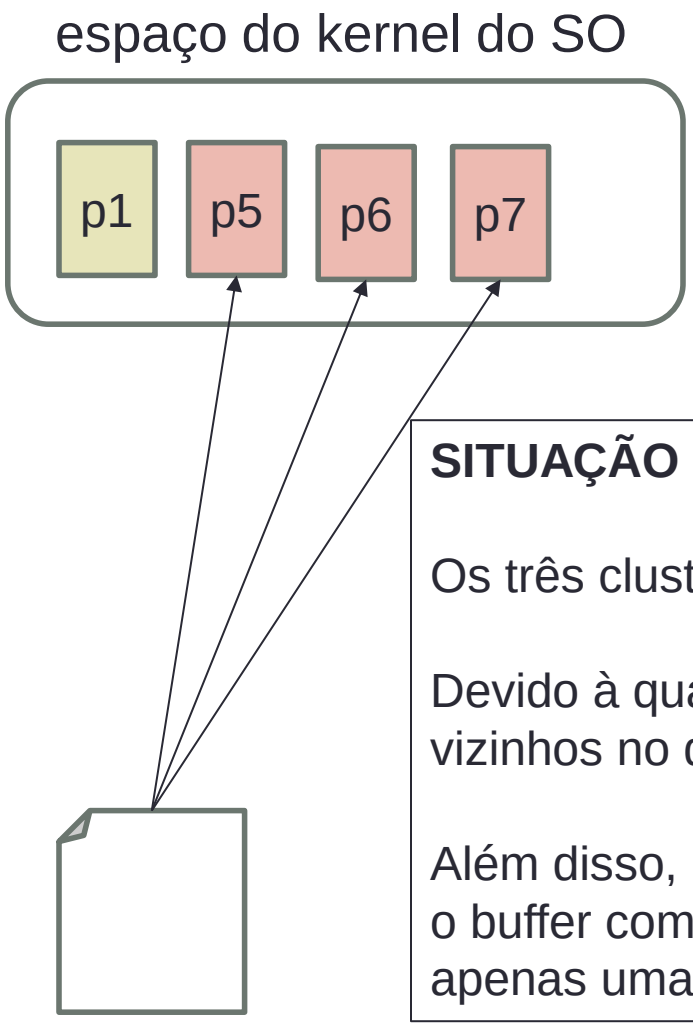
SITUAÇÃO 5: Leitura de múltiplos clusters

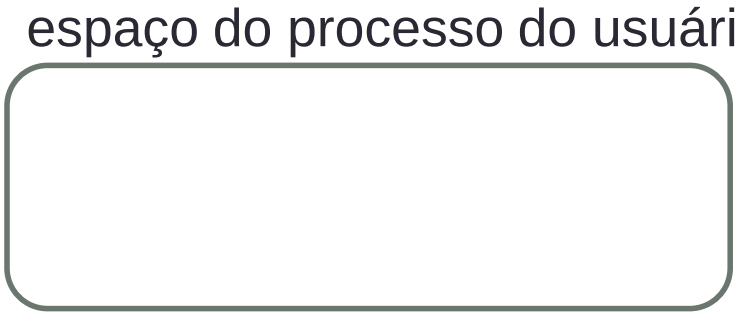
Como o buffer está cheio, é necessário abrir espaço removendo páginas.

Esse processo de remoção de páginas tem um custo associado.



MEMÓRIA VIRTUAL





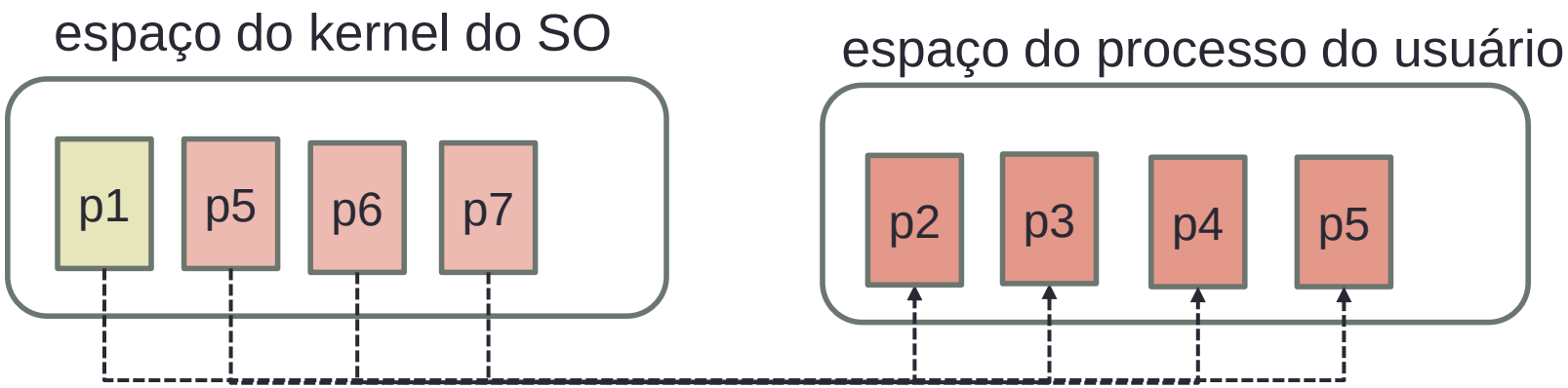
SITUAÇÃO 5: Leitura de múltiplos clusters

Os três clusters faltantes são carregados.

Devido à quantidade, é possível que não sejam todos vizinhos no disco, o que aumenta o custo de leitura.

Além disso, a grande quantidade de clusters lidos encheu o buffer com páginas que possivelmente sejam usadas apenas uma vez.

MEMÓRIA VIRTUAL

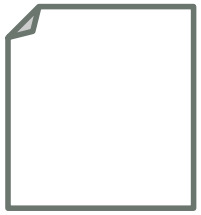


SITUAÇÃO 5: Leitura de múltiplos clusters

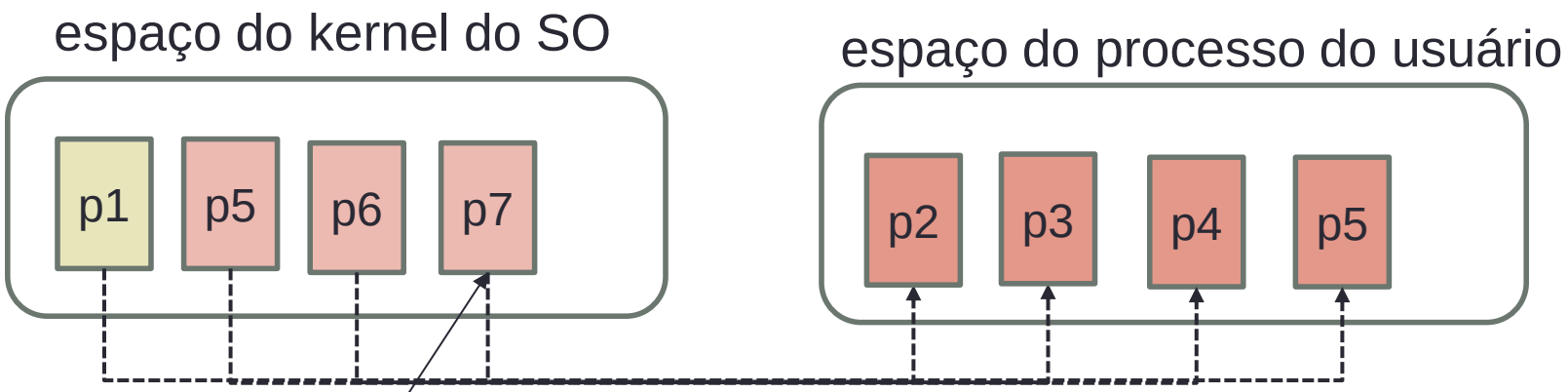
Por fim, as páginas carregadas são enviadas ao espaço do usuário.

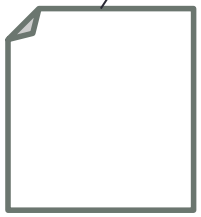
O processo do usuário precisa esperar que todas as páginas seja lidas para iniciar o processamento.

Ou seja, muito tempo gasto com IO pode tornar o CPU ocioso.



MEMÓRIA VIRTUAL





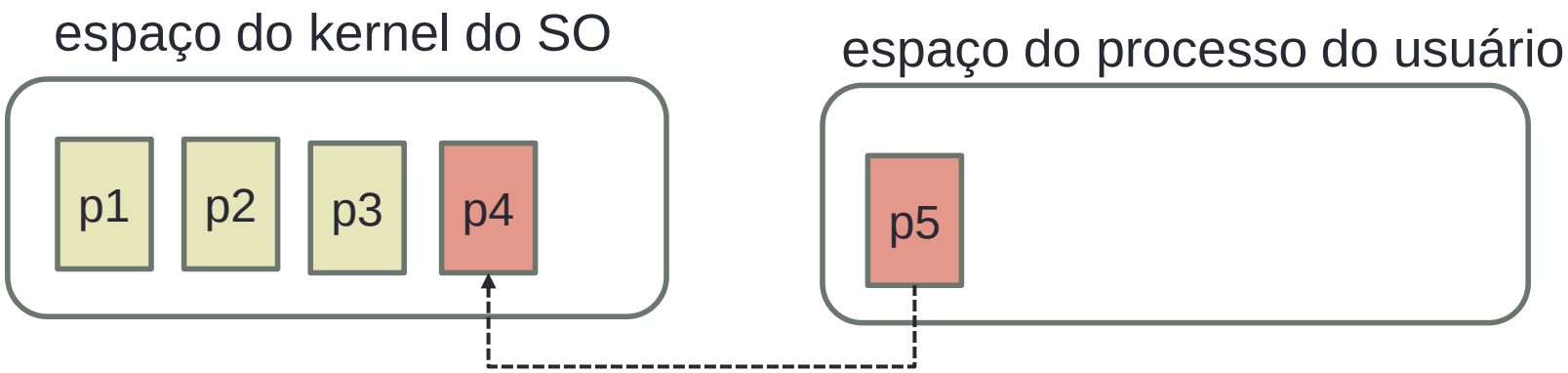
SITUAÇÃO 5: Leitura de múltiplos clusters

DICA: Leia apenas a quantidade de clusters que seja necessária.

Isso:

- reduz o custo de leitura (caso os clusters não sejam contínuos)
- não atrapalha o gerenciamento de memória
- reduz ociosidade da CPU

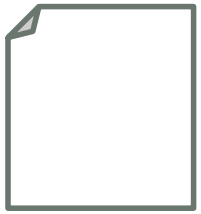
MEMÓRIA VIRTUAL



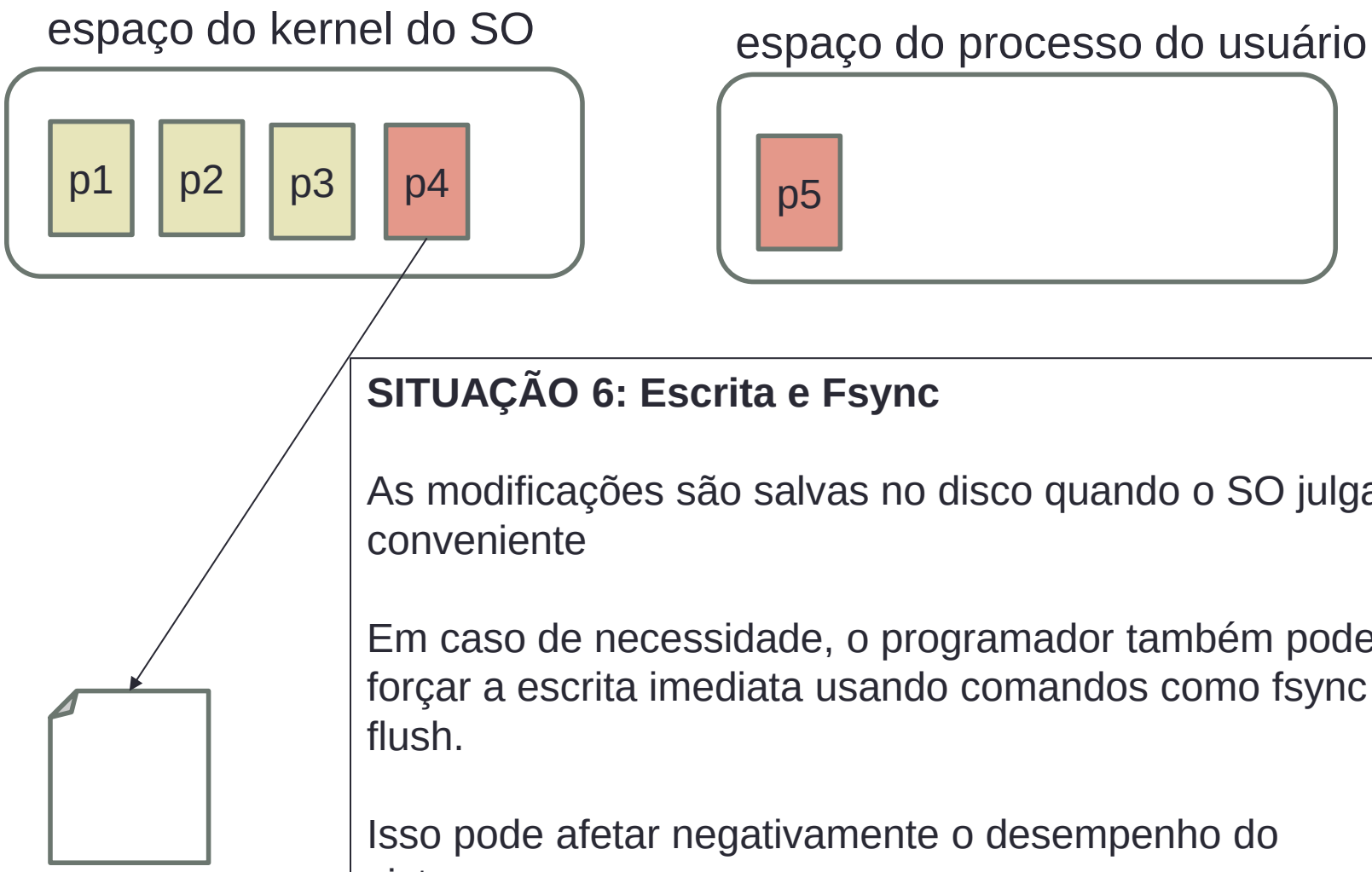
SITUAÇÃO 6: Escrita e Fsync

O processo usa uma chamada de sistema **write** caso queira gravar modificações em um arquivo.

Nesse caso, as modificações são enviadas para uma página no espaço do kernel.



MEMÓRIA VIRTUAL



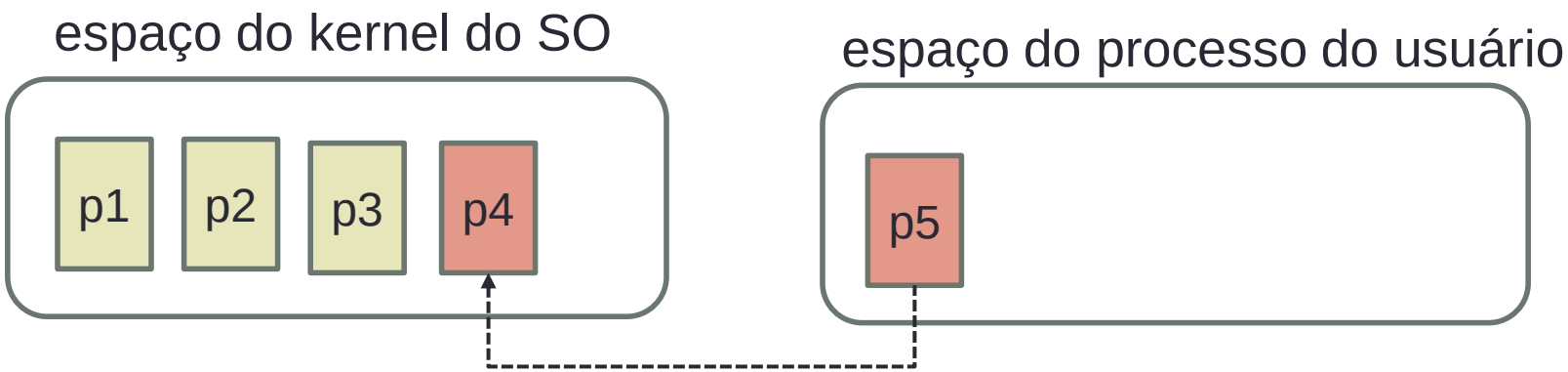
SITUAÇÃO 6: Escrita e Fsync

As modificações são salvas no disco quando o SO julgar conveniente

Em caso de necessidade, o programador também pode forçar a escrita imediata usando comandos como fsync ou flush.

Isso pode afetar negativamente o desempenho do sistema.

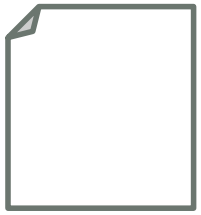
MEMÓRIA VIRTUAL



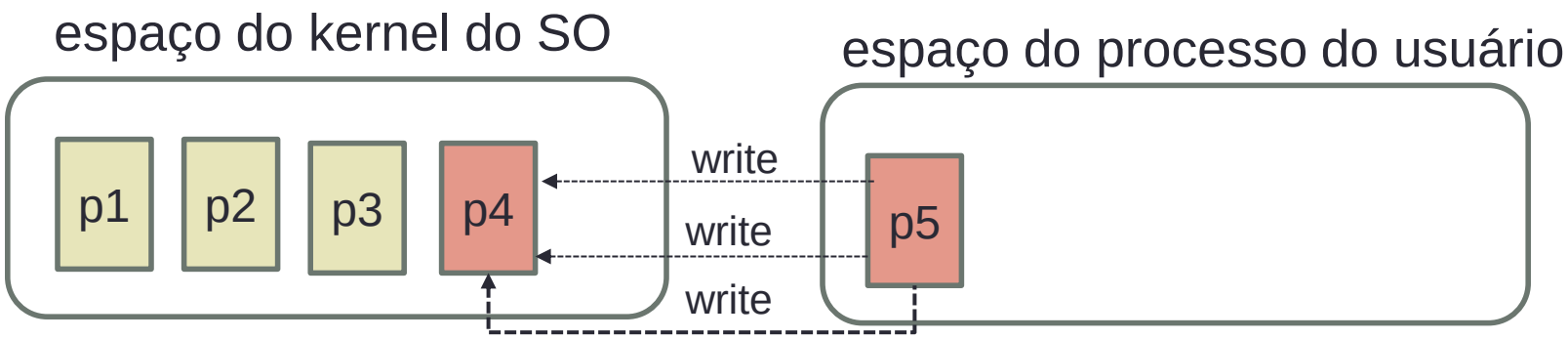
SITUAÇÃO 6: Escrita e Fsync

DICA: Quando são necessárias várias modificações em um arquivo, é melhor realizá-las todas antes de forçar a escrita em disco.

Isso evita que varias operações de escrita sejam realizadas.



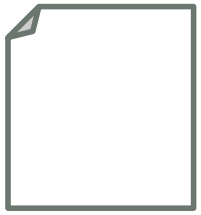
MEMÓRIA VIRTUAL



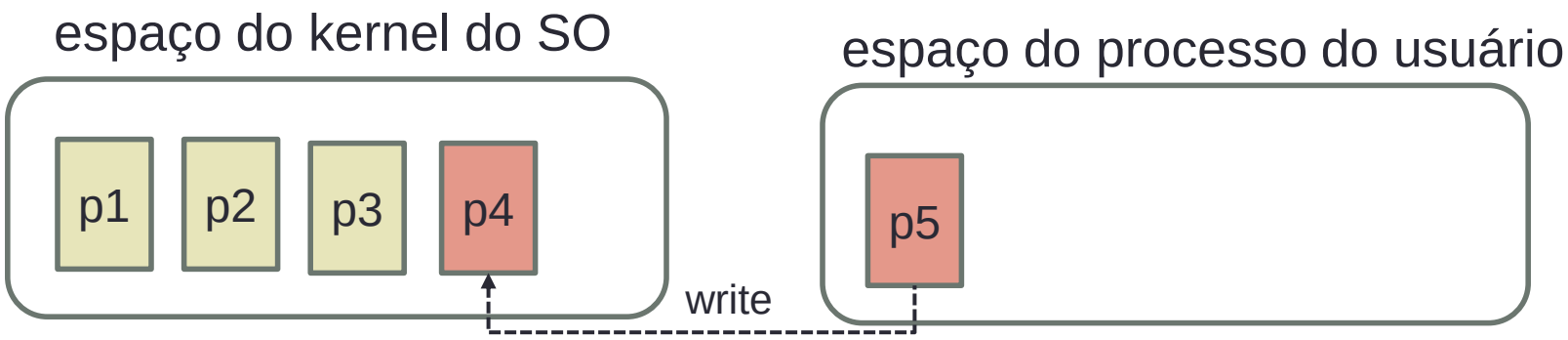
SITUAÇÃO 7: Write buffer

O processo usa várias chamadas de sistema **write** para gravar modificações em um arquivo

Cada write requer uma chamada de sistema, que são relativamente custosas, pois envolvem troca de contexto, do modo usuário para o modo kernel.



MEMÓRIA VIRTUAL



SITUAÇÃO 7: Write buffer

DICA: Junte em um buffer local todas as modificações em uma única operação de write.

Essa bufferização pode ser feita manualmente ou usando funções que possuam essa finalidade. Ex. BufferedOutputStream do java.