Estruturas baseadas em Listas

Prof. Joaquim Uchôa
Profa. Juliana Greghi
Prof. Renato Ramos

- Deques
- Vetores Expansíveis
- Sequence Set

Deques - Double Queues

Deque - Fila Dupla (Double Queue)

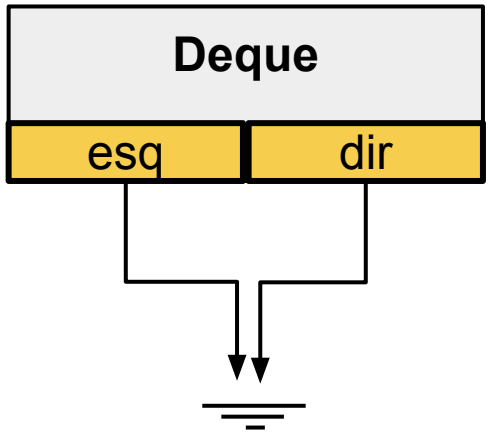
Uma deque é uma estrutura de dados em que a remoção e inserção só é permitida em suas pontas (ou seja: no início e no final da lista).

Em diversas implementações e utilizada aqui, o início é definido como esquerdo (**esq**) e o fim como direito (**dir**).

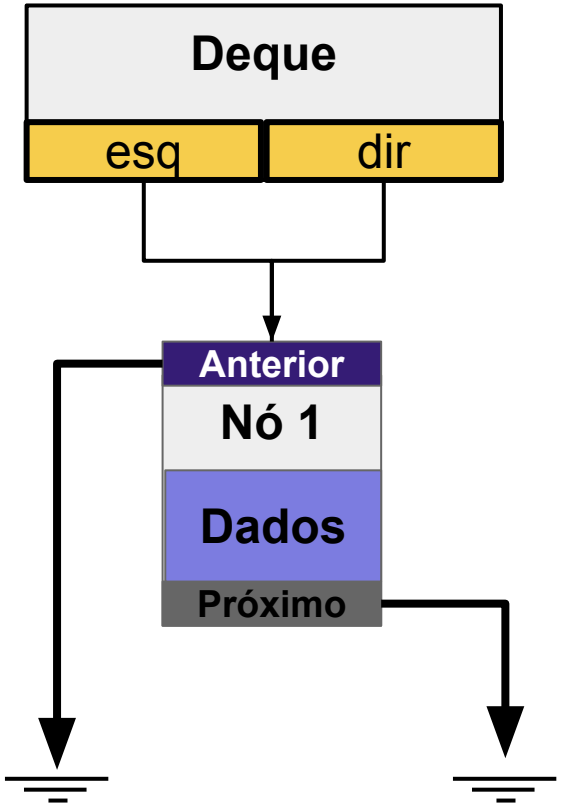
Deque - Fila Dupla (Double Queue)

Em geral, são implementadas como listas duplamente encadeadas, sem adicionar suporte para busca, percorrimento, bem como inserção e remoção em posições intermediárias.

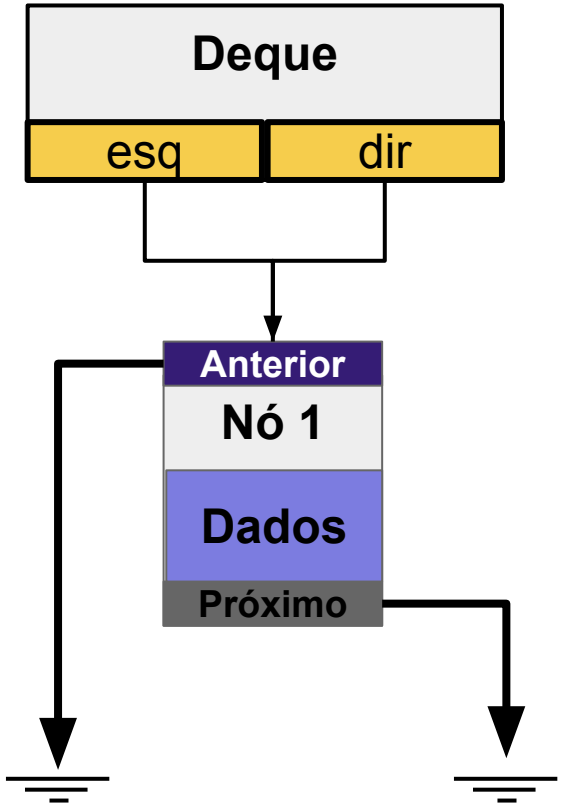
Inserção em deque vazia - I



Inserção em deque vazia - II



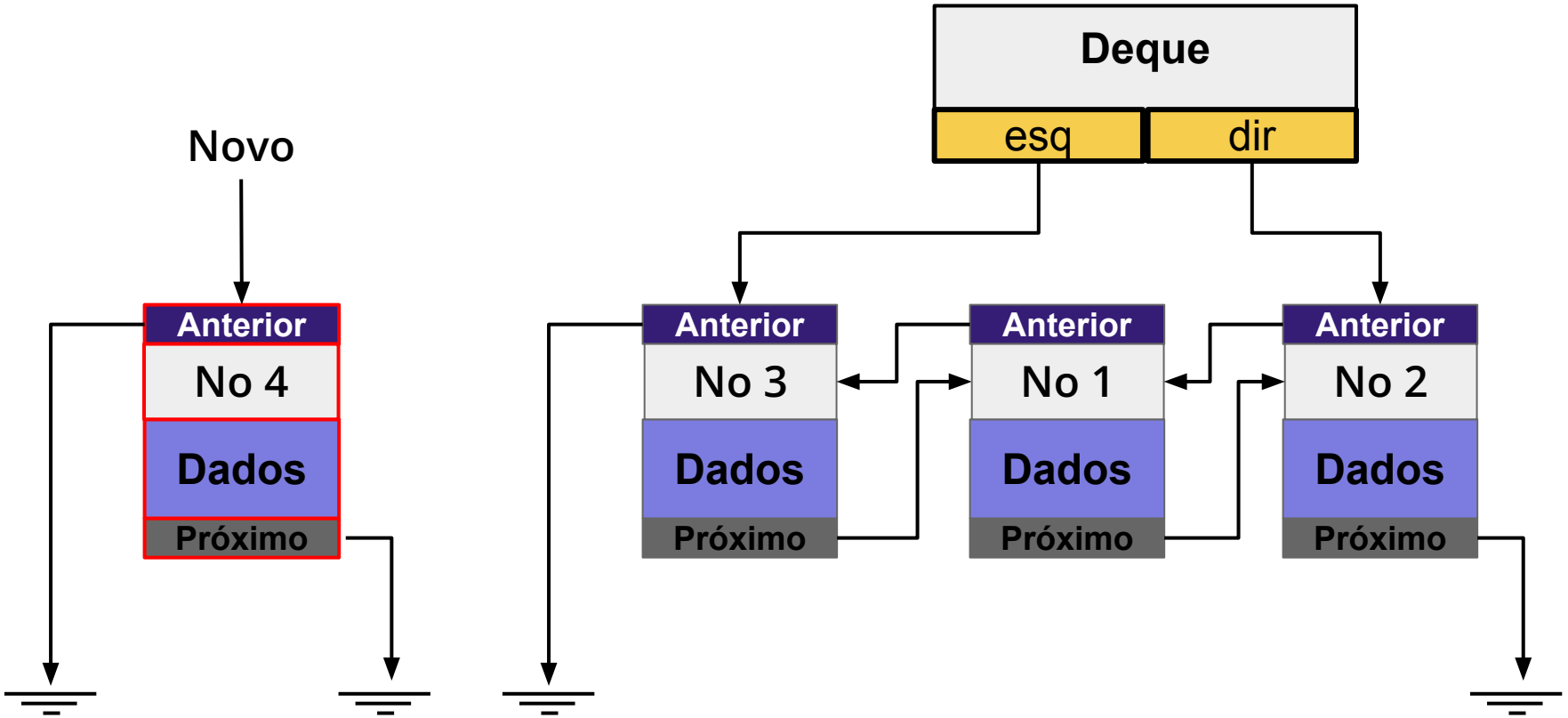
Inserção em deque vazia - II



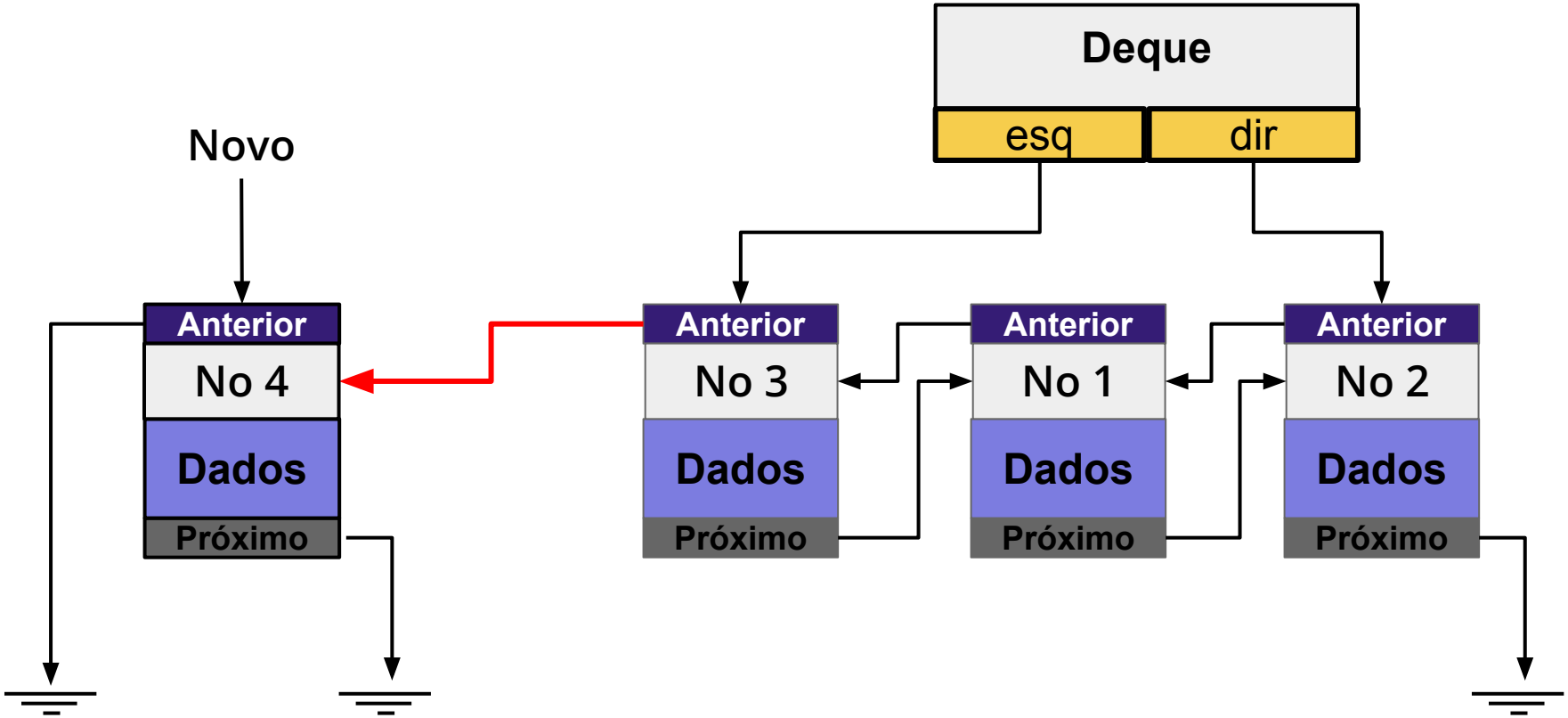
A inserção em deque vazia é utilizada tanto para inserção à esquerda ou à direita, desde que se constate que a estrutura não possui ainda elementos.

Método é implementado como auxiliar, usado se necessário.

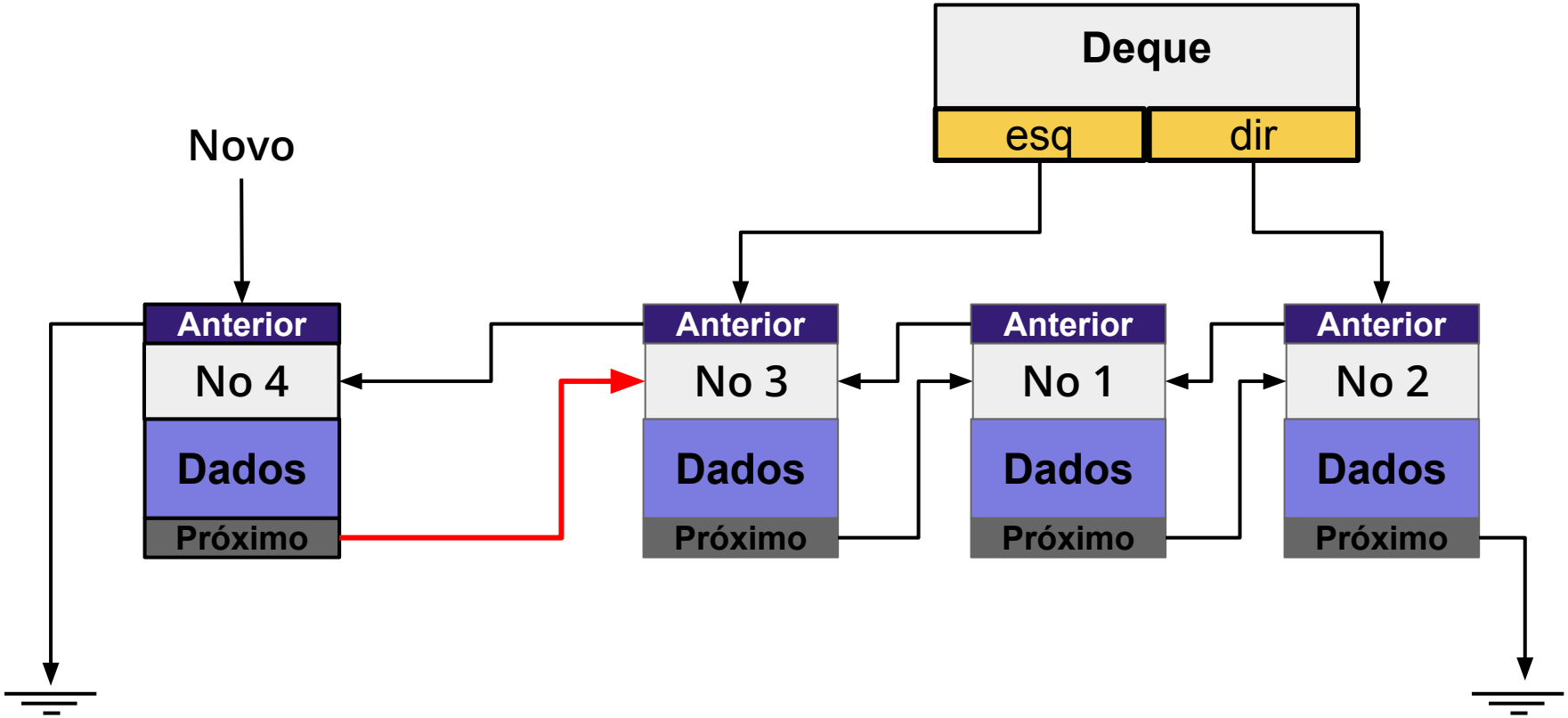
Inserção à esquerda - I



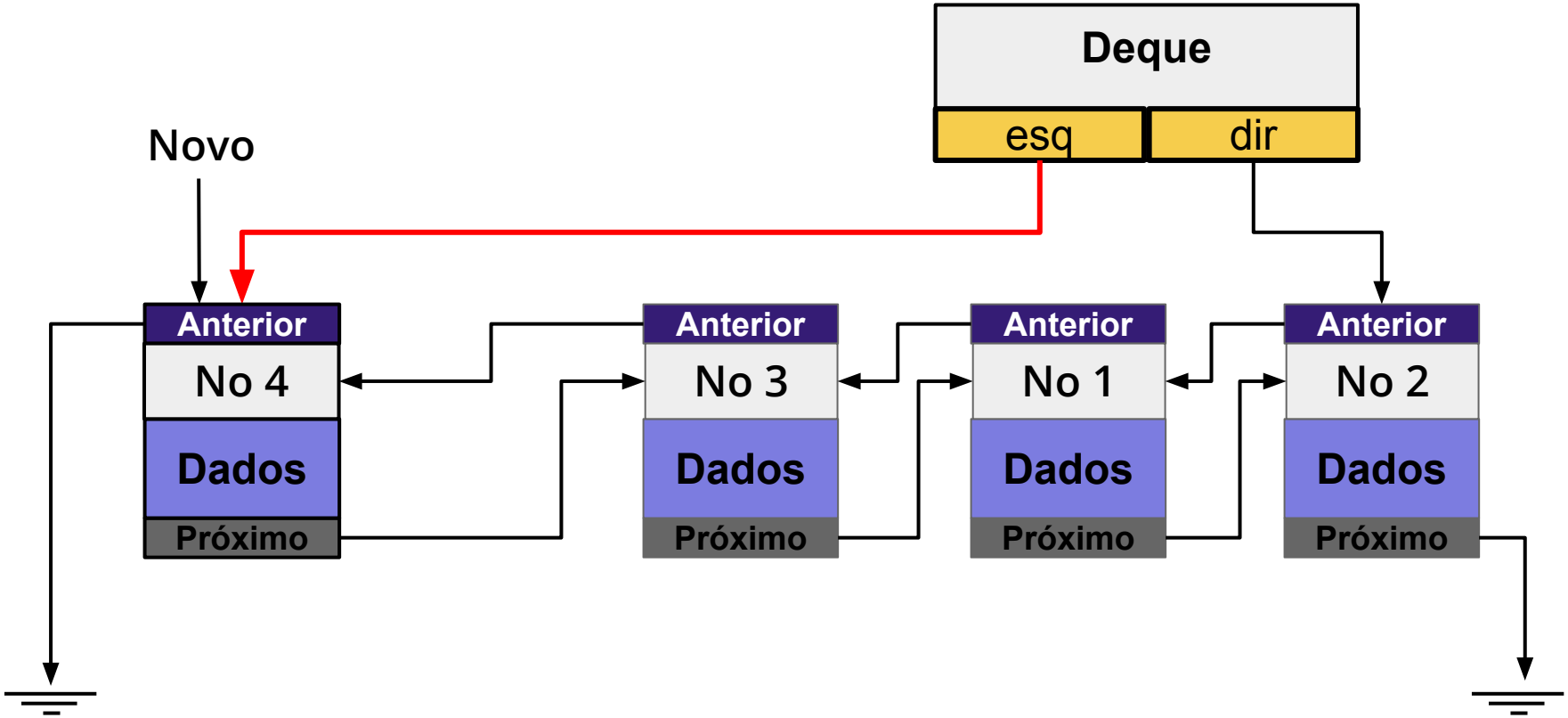
Inserção à esquerda - II



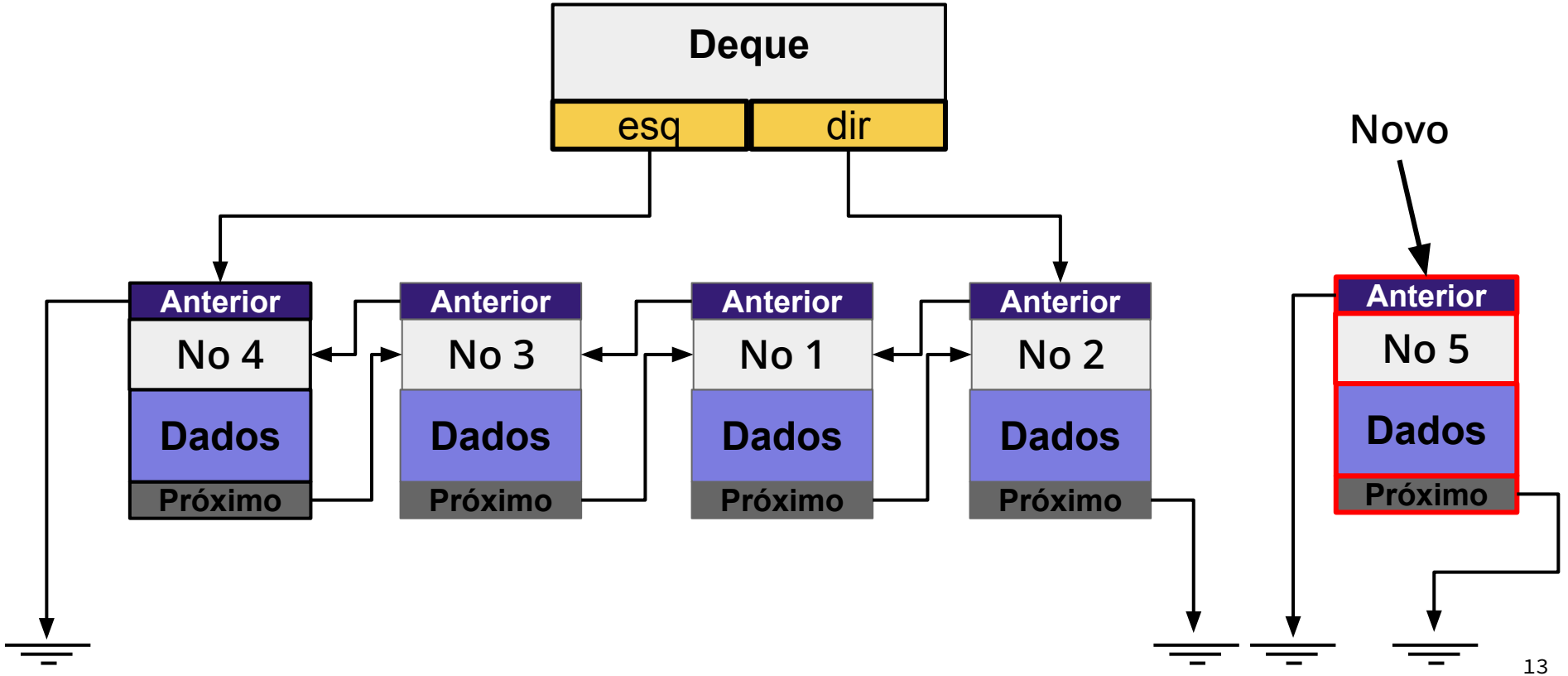
Inserção à esquerda - III



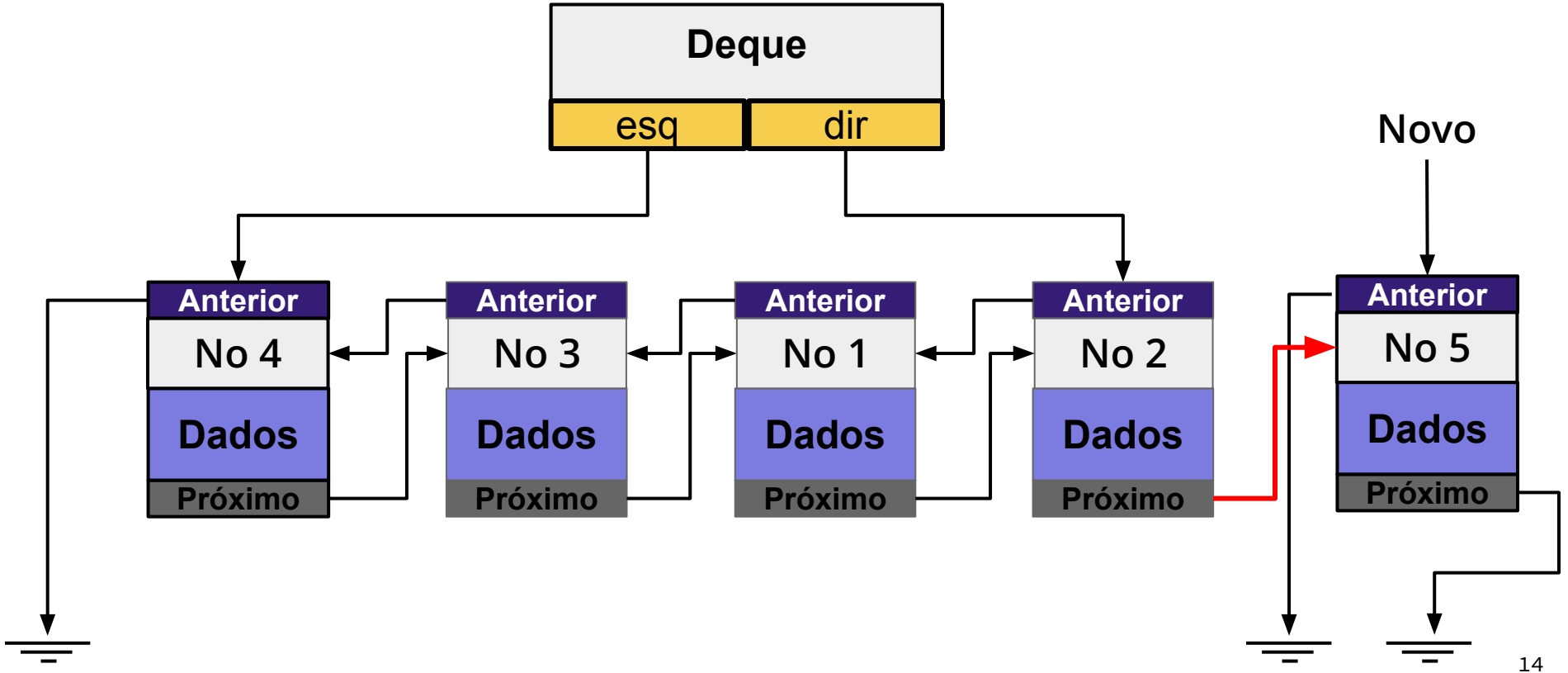
Inserção à esquerda - iv



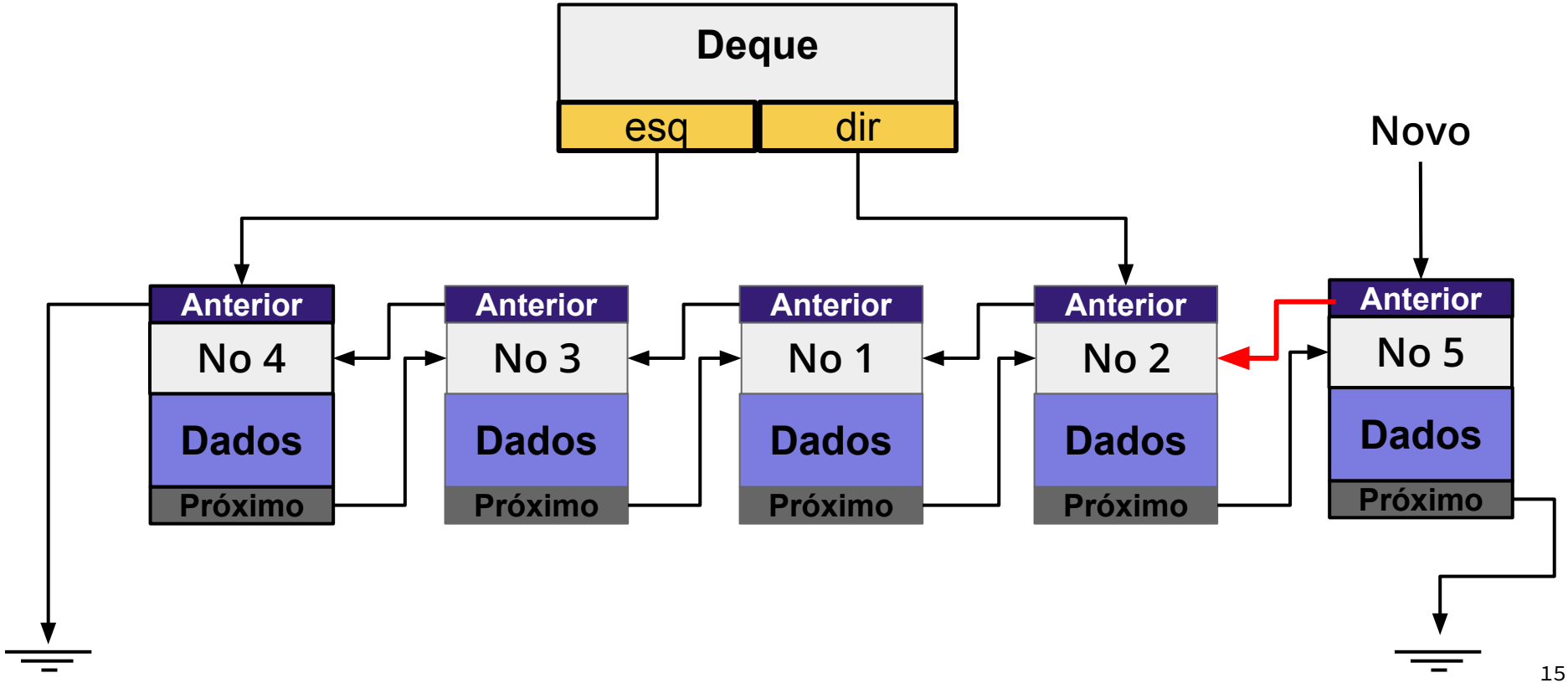
Inserção à direita - I



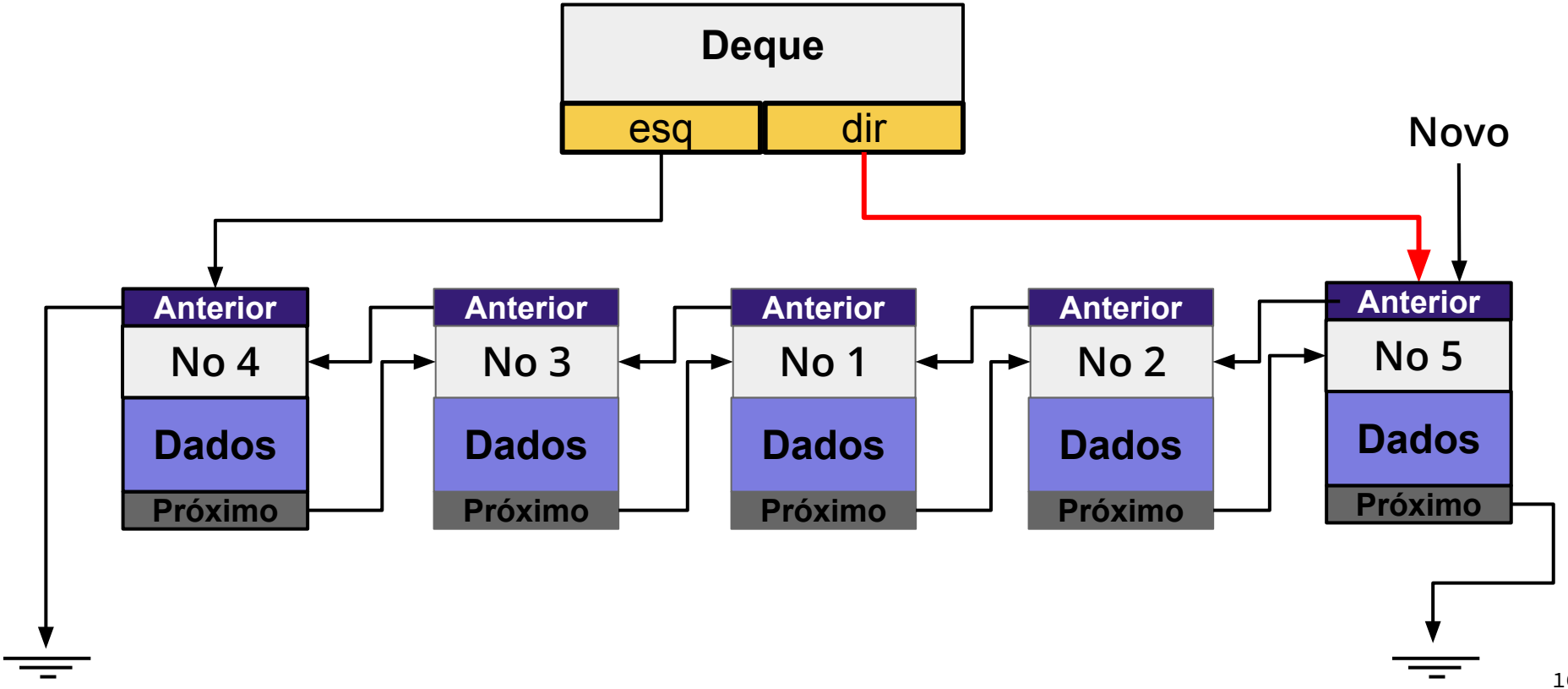
Inserção à direita - II



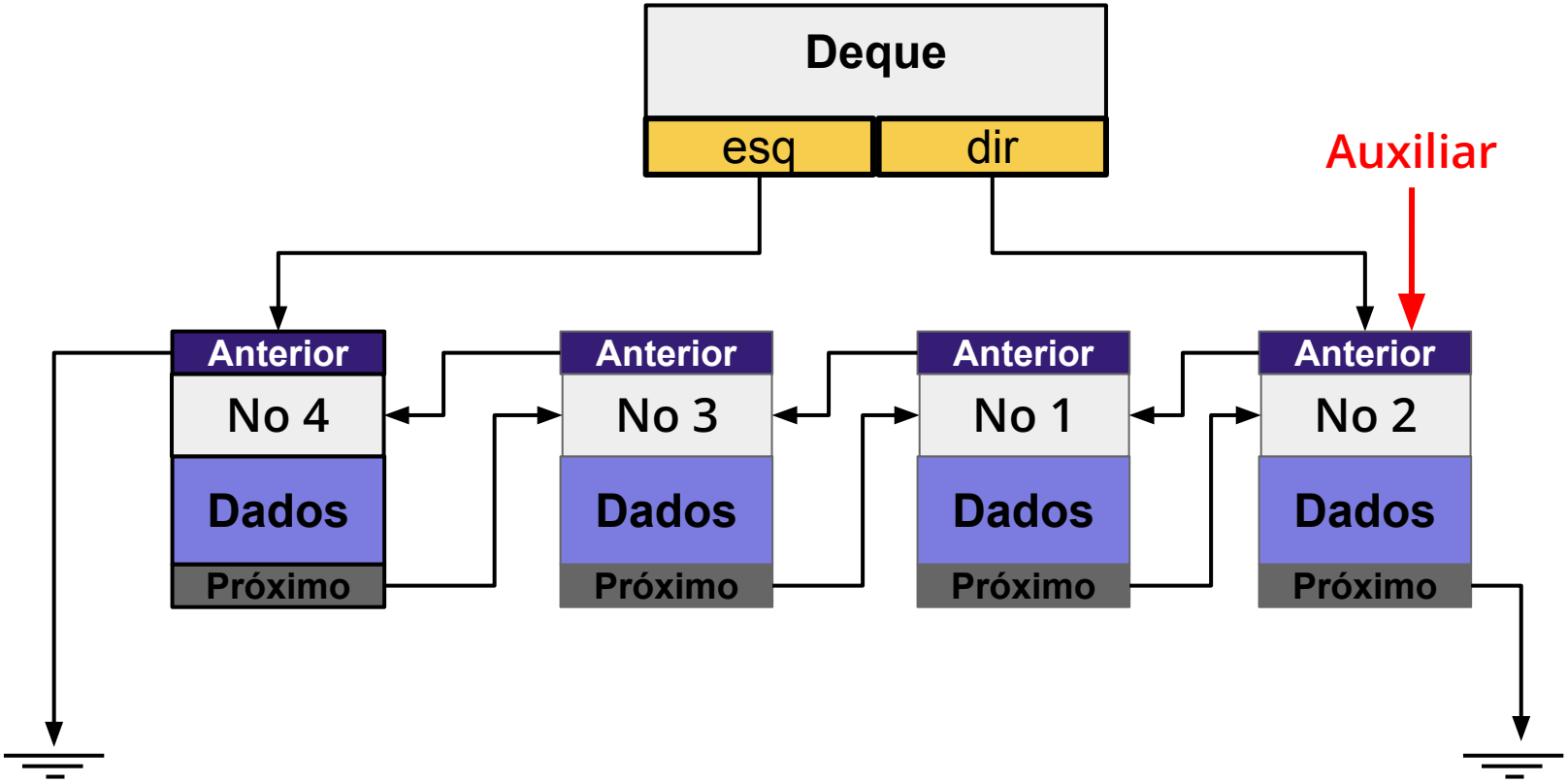
Inserção à direita - III



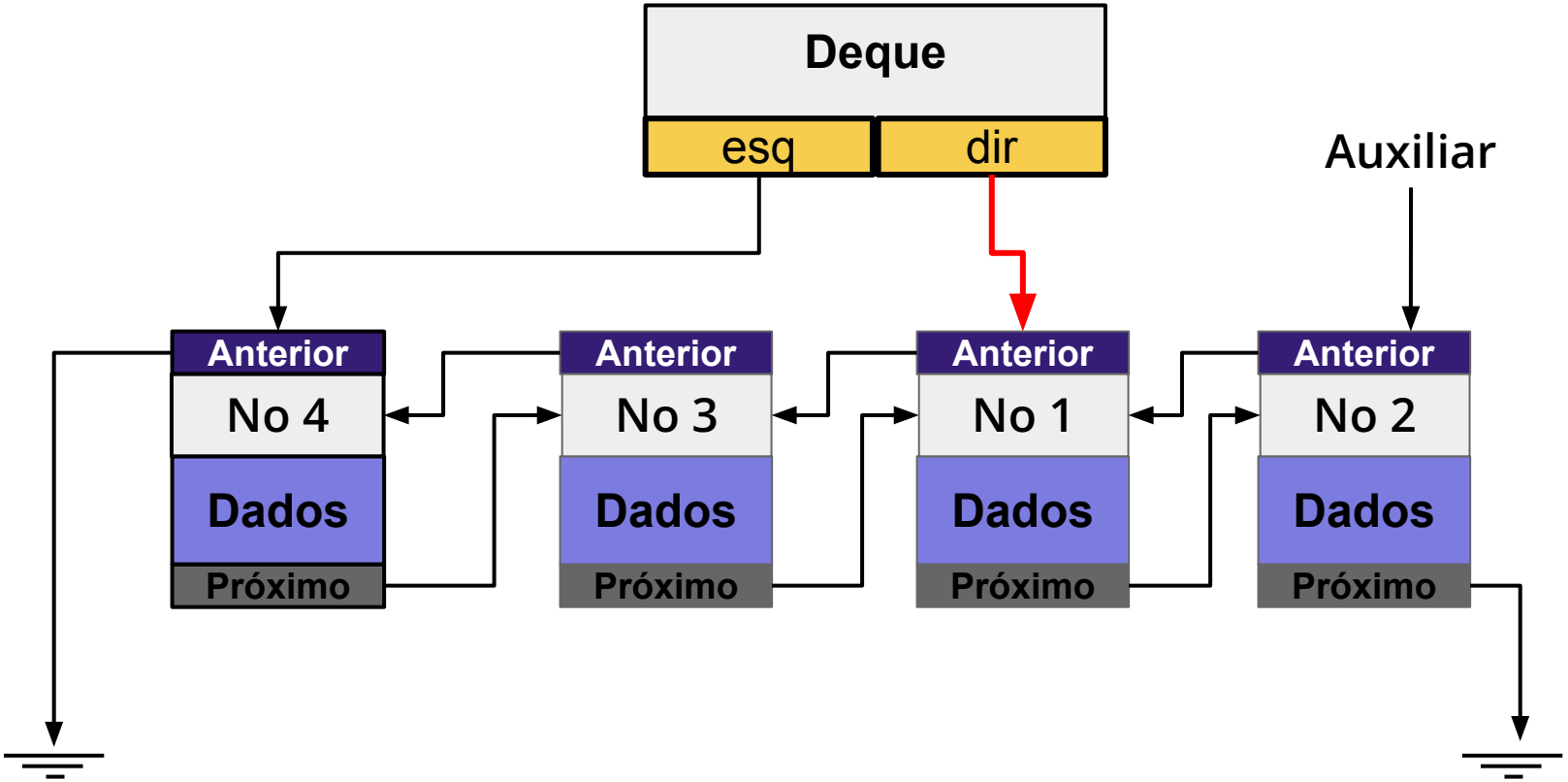
Inserção à direita - IV



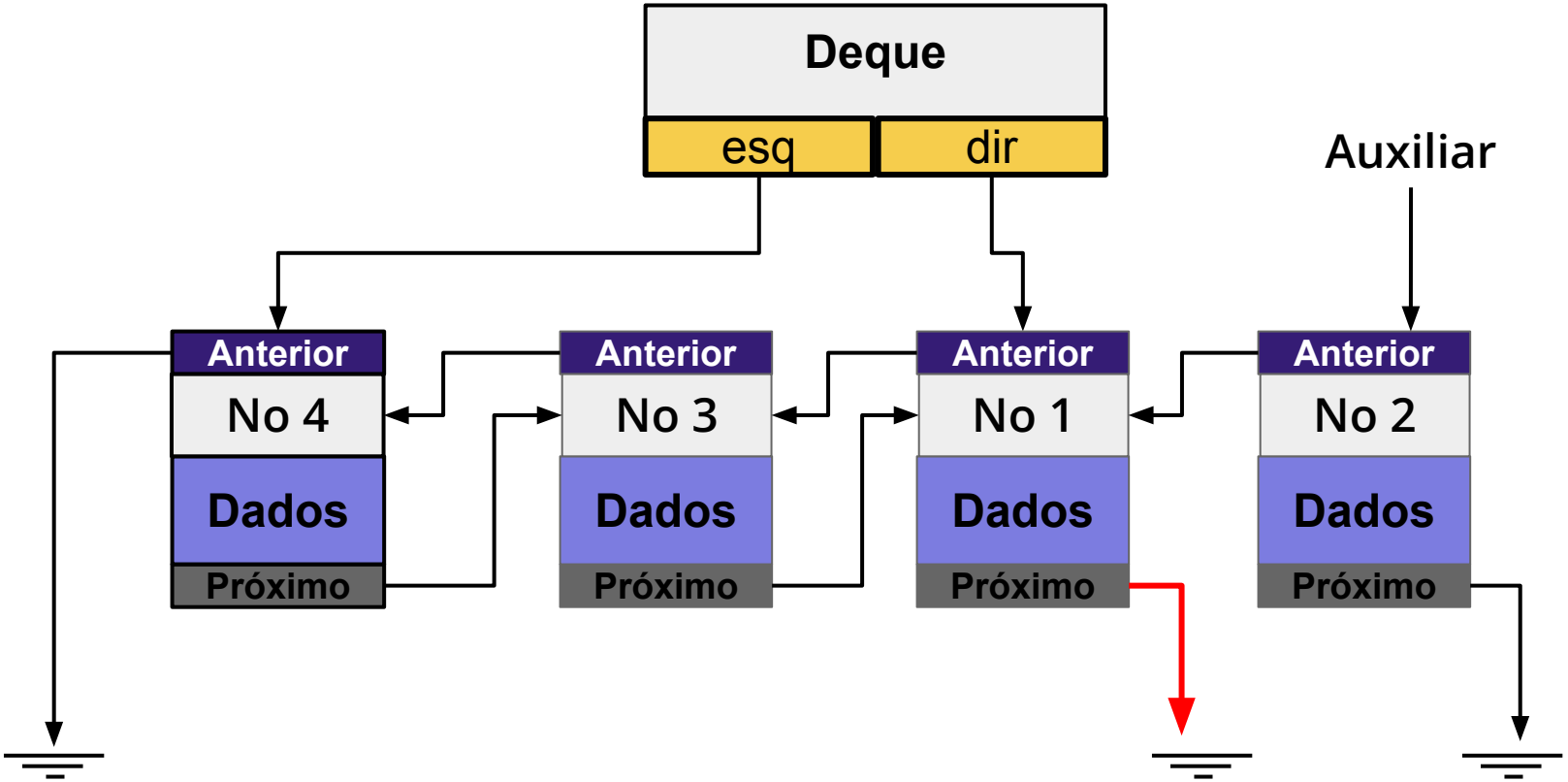
Remoção à direita - 1



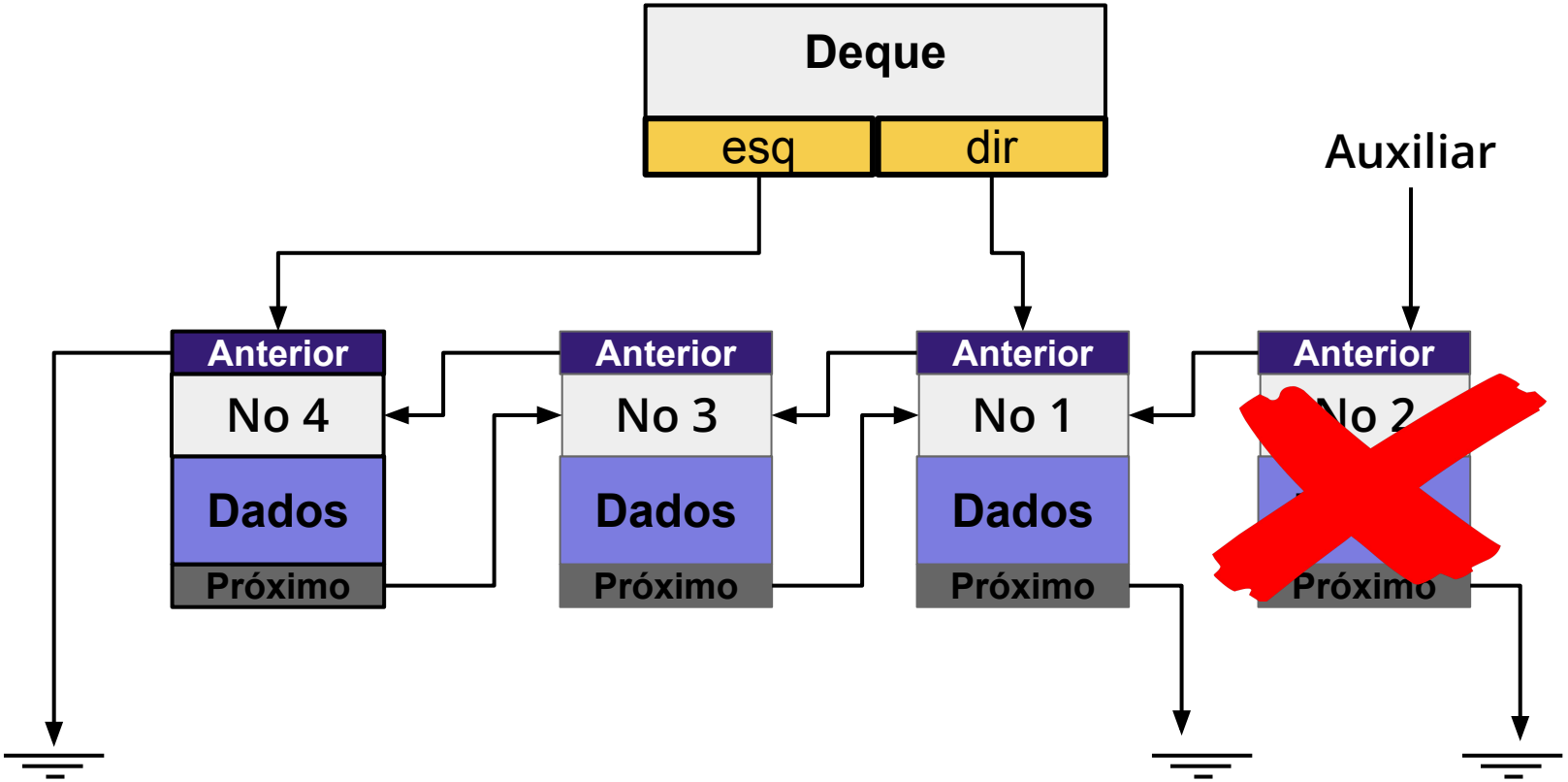
Remoção à direita - II



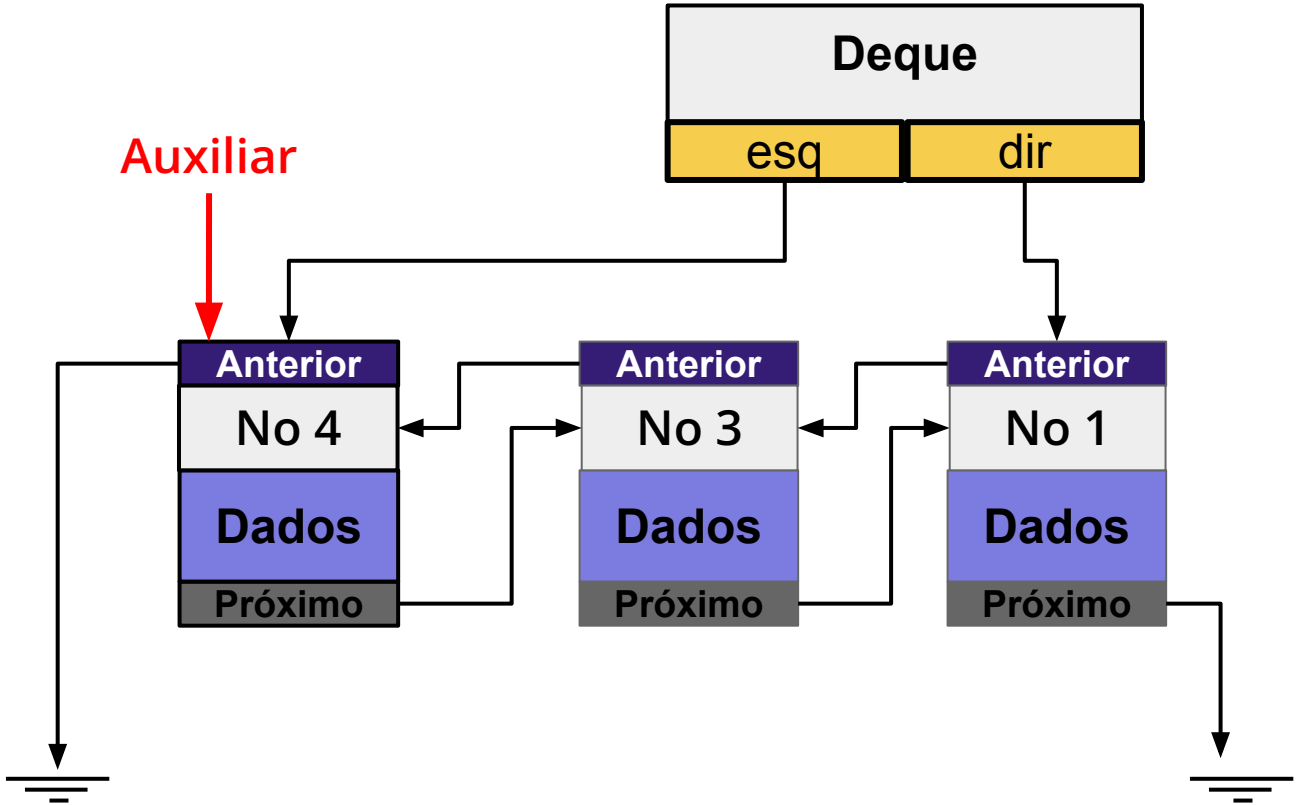
Remoção à direita - III



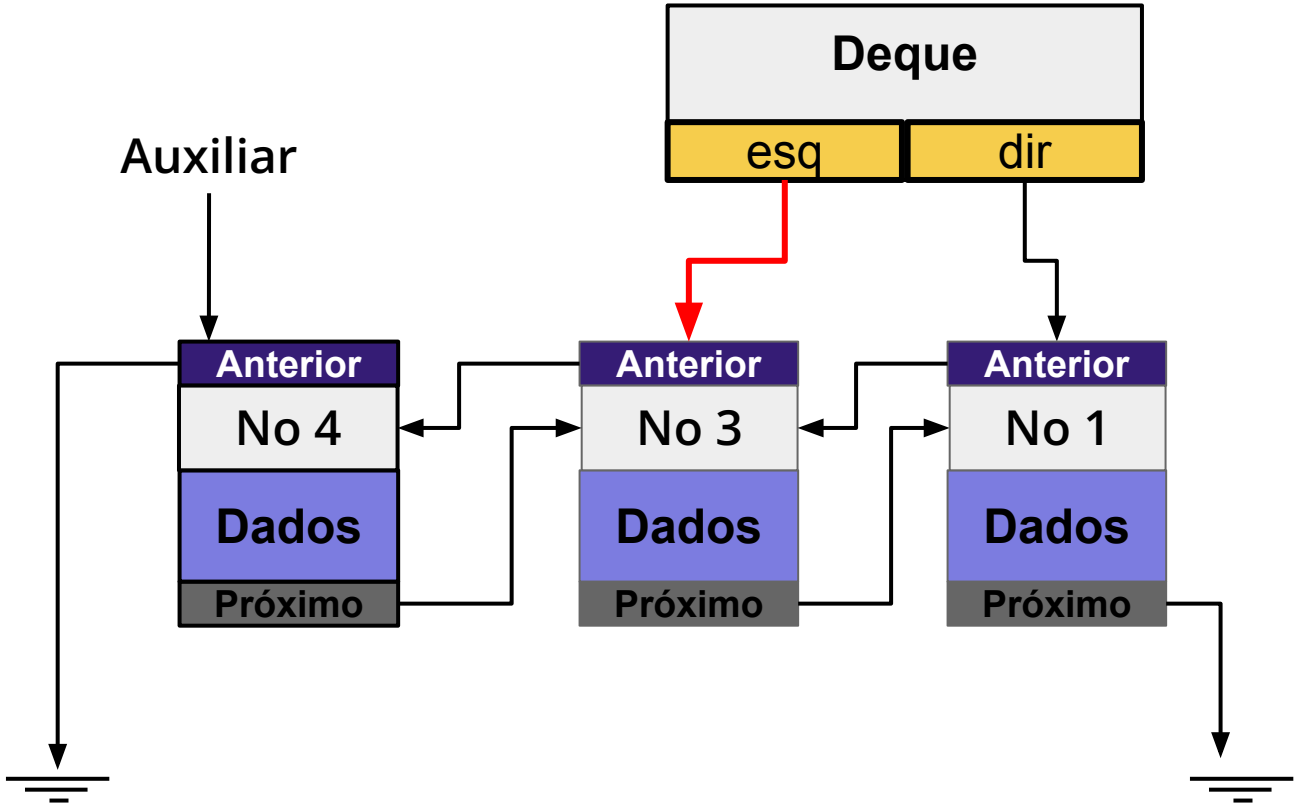
Remoção à direita - IV



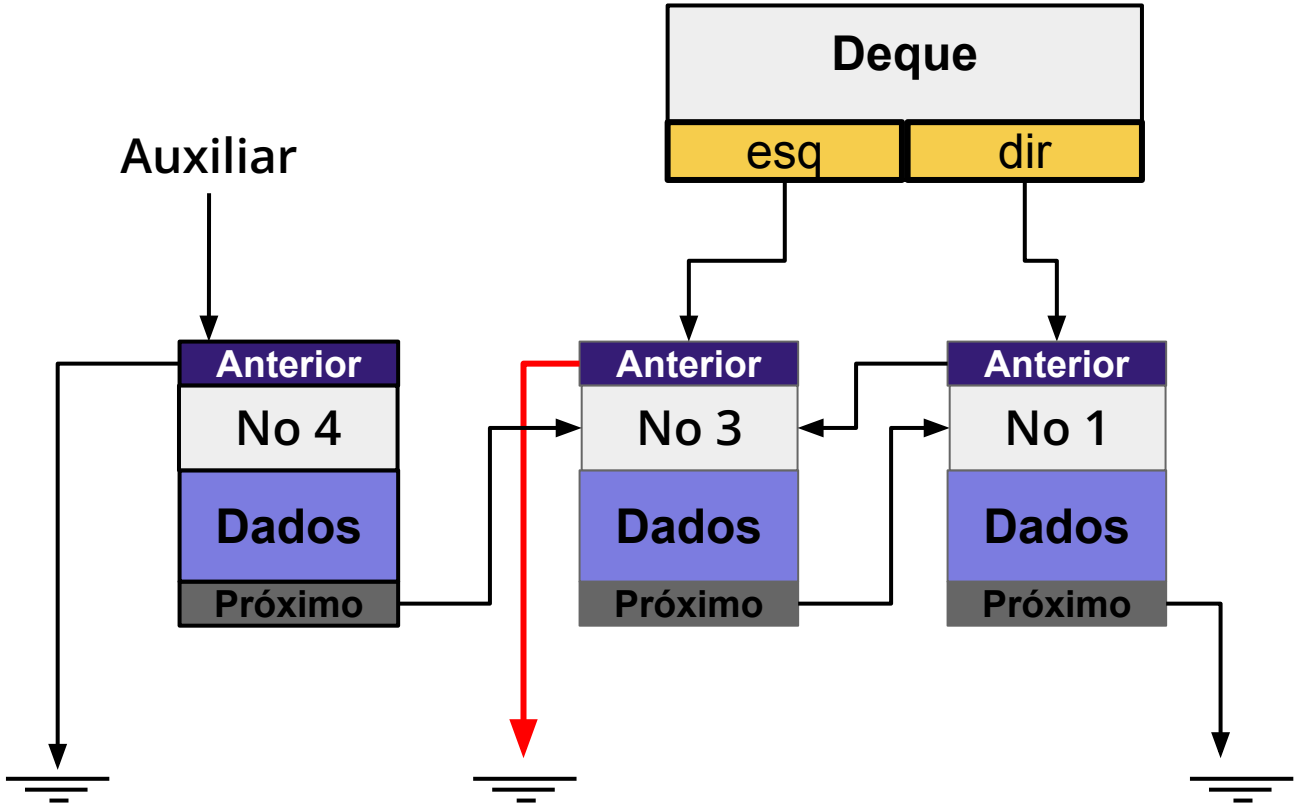
Remoção à esquerda -I



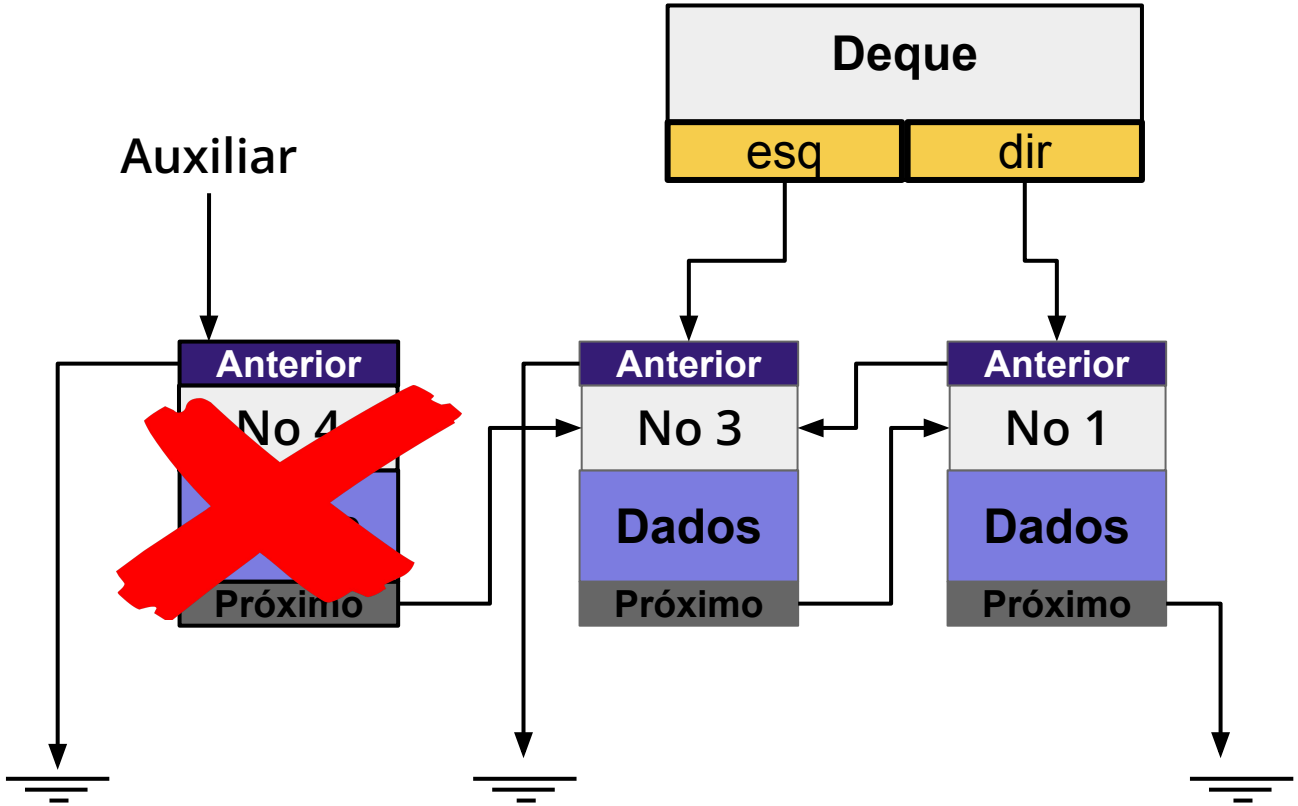
Remoção à esquerda -II



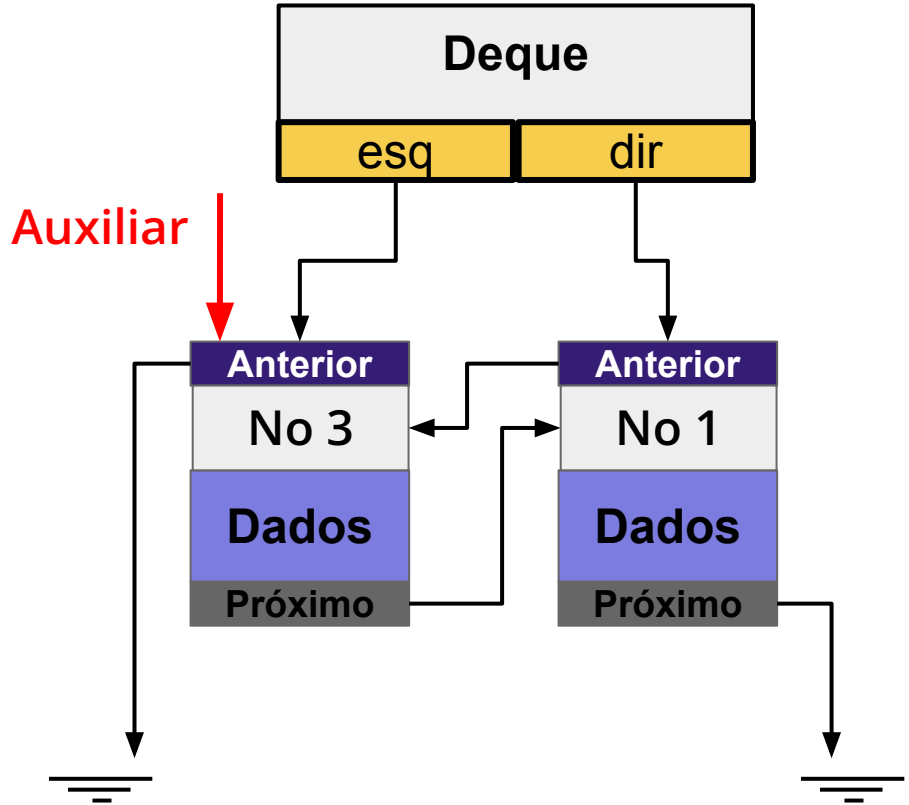
Remoção à esquerda -IV



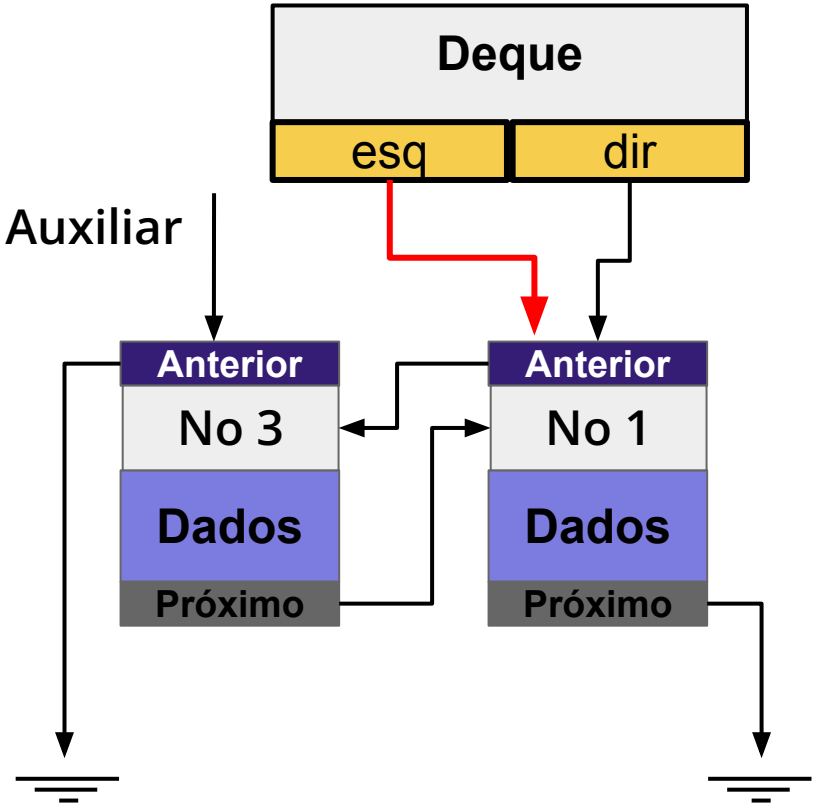
Remoção à esquerda -v



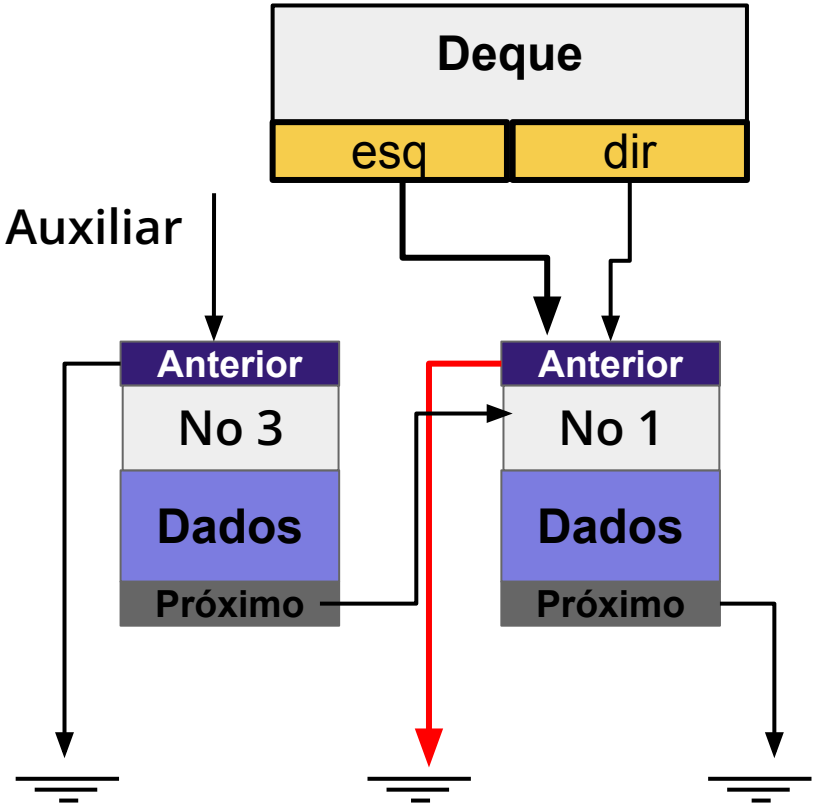
Nova Remoção à esquerda -I



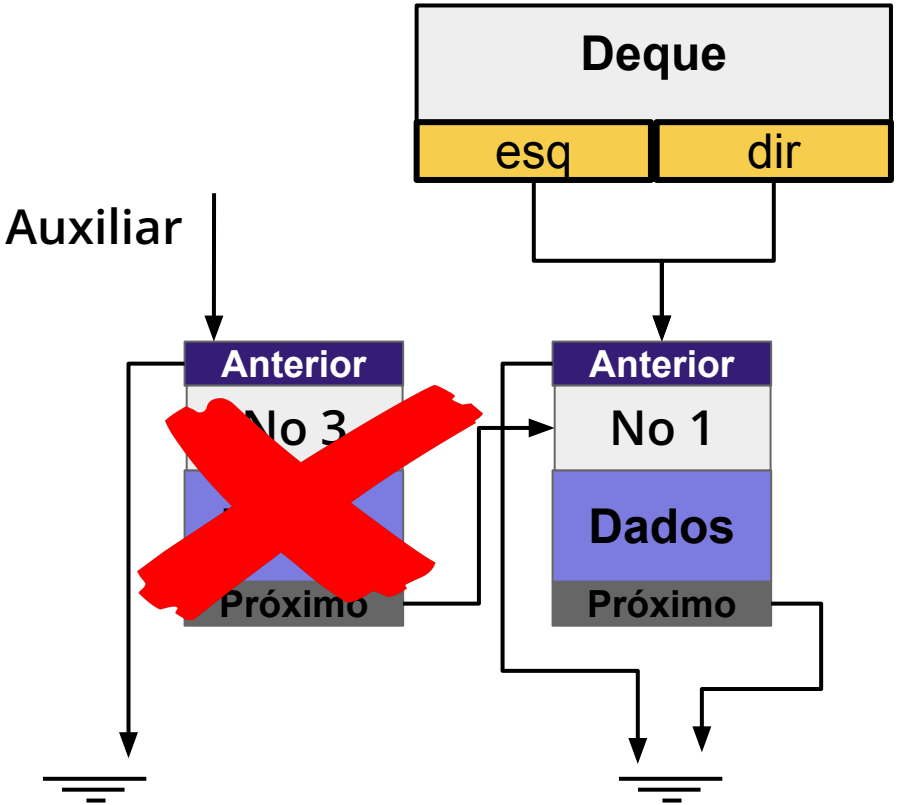
Nova Remoção à esquerda -II



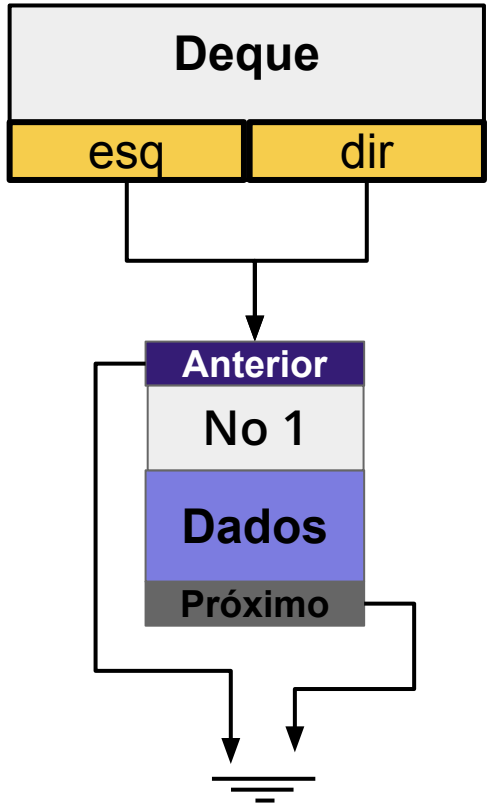
Nova Remoção à esquerda -IV



Nova Remoção à esquerda -V

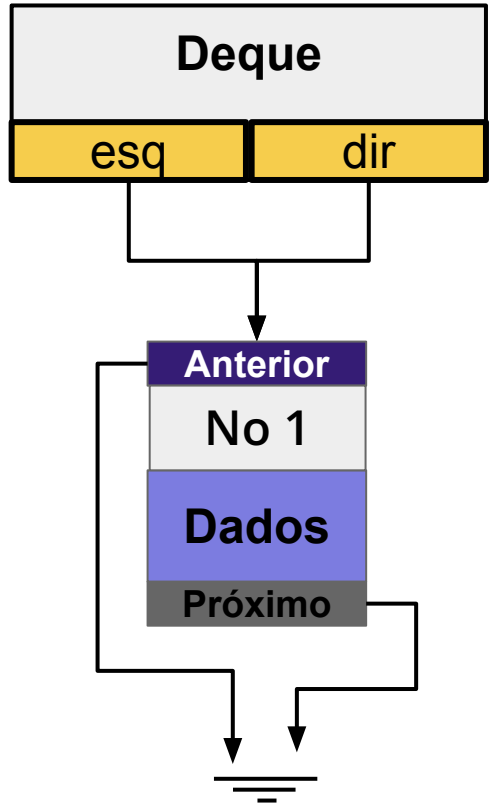


Nova Remoção à esquerda -VI

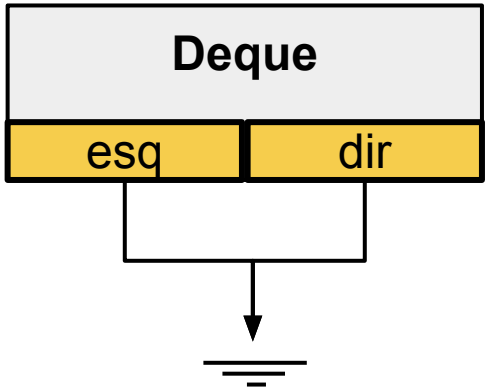


Remoção do último elemento

Caso seja feita a remoção do último elemento da deque, não importa se à esquerda ou à direita, será necessário ajustar os ponteiros esq e dir para NULO



Remoção do último elemento



Vetores Expansíveis

Comentários iniciais

Vetores possuem como principal característica o fácil acesso a qualquer um de seus elementos.

Em contrapartida, o redimensionamento é uma tarefa complicada.

Em listas, ocorre o inverso.

Comentários iniciais

Vetores p terística
o fácil a
elementos

Em contra o é uma
tarefa co

Em listas, ocorre o inverso.

E quanto precisamos de uma solução intermediária???

Precisamos de redimensionamento, mas também de rápido acesso...

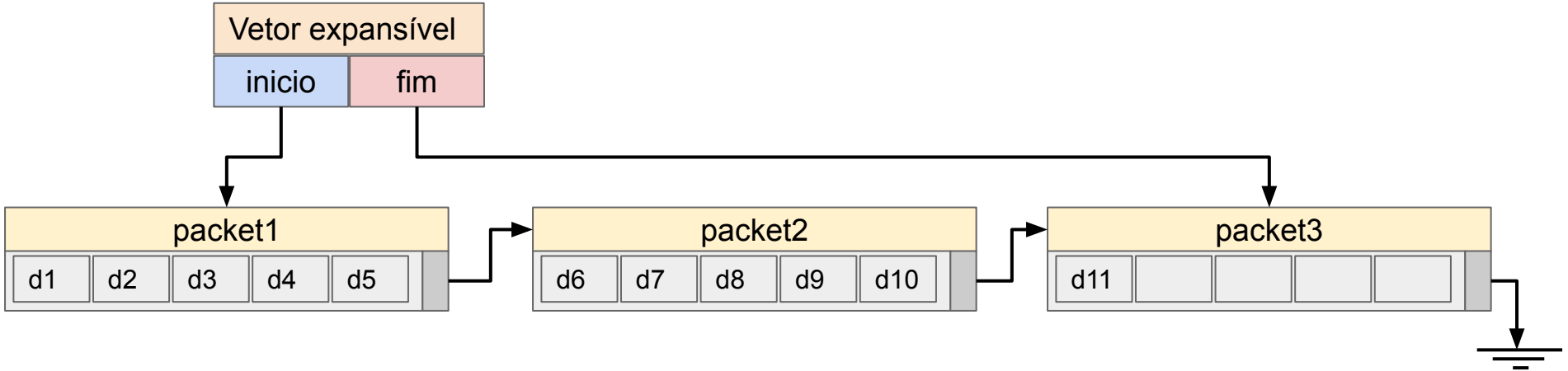
Comentários iniciais

Em várias situações é necessário o uso de vetores redimensionáveis, capazes, principalmente, de aumentar o tamanho sem um esforço computacional exagerado.

A melhor forma de implementar é como uma lista de vetores menores (pedaços, *buckets* ou *packets*).

Vetores expansíveis - I

Em geral vetores expansíveis são implementados como uma lista, mas em que seus elementos são vetores menores.



Vetores expansíveis - II

Em implementações mais simples, os pedaços (*packets*) costumam ter o mesmo tamanho. Esse tamanho é definido geralmente como atributo do vetor ou valor constante.

Nesse caso ainda, cada pedaço contém, além do vetor de dados, um ponteiro para o próximo pedaço (e dependendo a aplicação, também para o anterior).

Vetores expansíveis - III

Quando da inserção de um elemento, caso o pedaço esteja cheio, é alocado um novo pedaço para a tarefa.

Pedaços vazios, após remoção do último elemento, são simplesmente desalocados da memória com ajustes dos ponteiros dos pedaços vizinhos.

Vetores expansíveis - IV

Algumas implementações possuem pedaços de tamanho variável, atendendo a demandas específicas de redimensionamento.

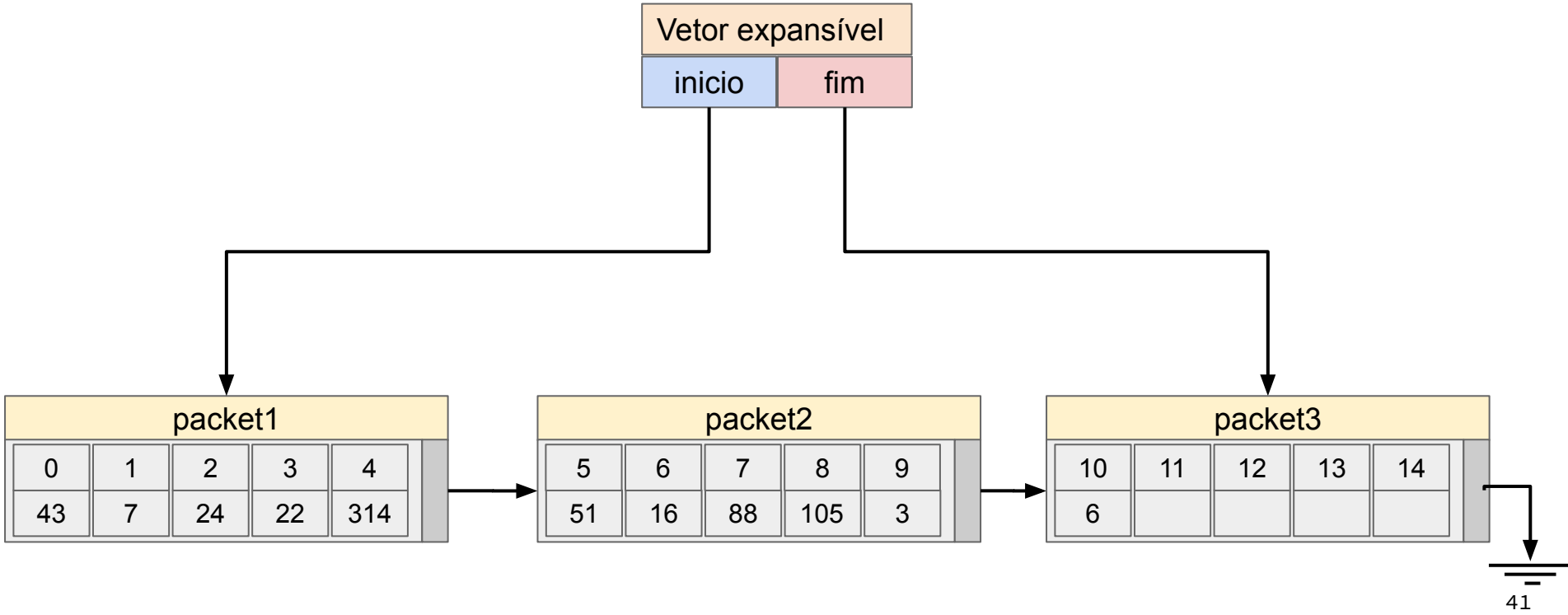
Nesse caso, cada pedaço possui, além dos dados e ponteiros, um atributo para armazenar sua própria capacidade.

Vetores expansíveis - V

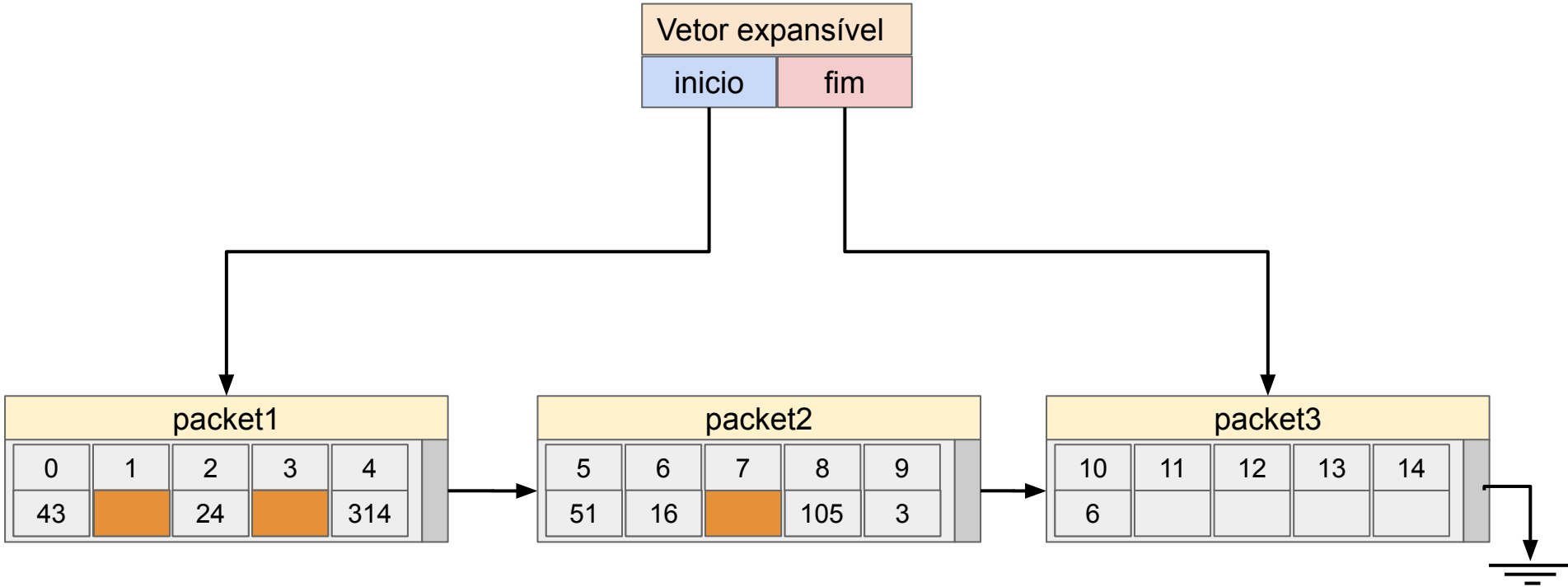
Apesar que conceitualmente não há restrição quanto ao tipo de objeto armazenados, a menos que o problema não precise de remoção, os dados precisam estar ordenados.

Sem a ordenação dos dados, a remoção torna inviável ou ineficiente o uso do vetor expansível.

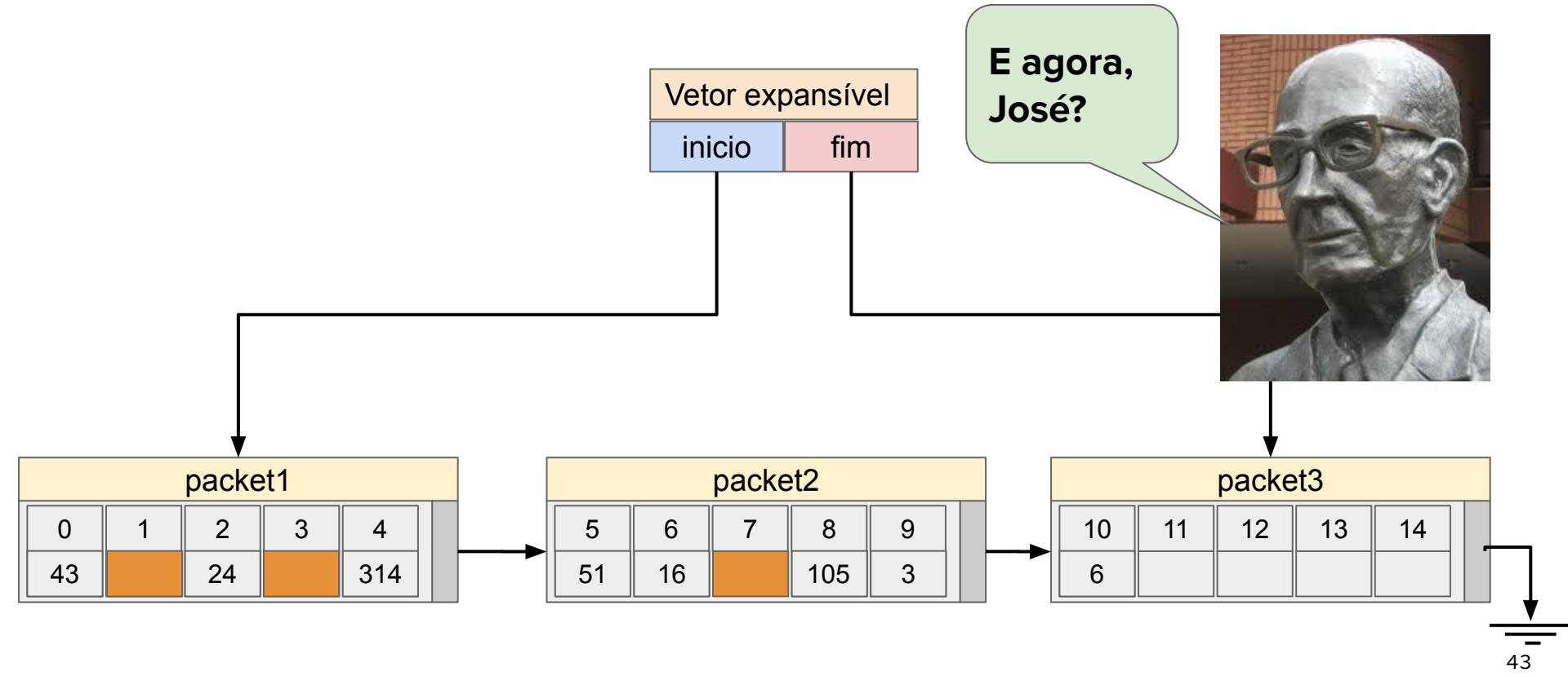
Exemplo- Dados Não-Ordenados - Remoção - 1



Exemplo- Dados Não-Ordenados - Remoção - II



Exemplo- Dados Não-Ordenados - Remoção - II



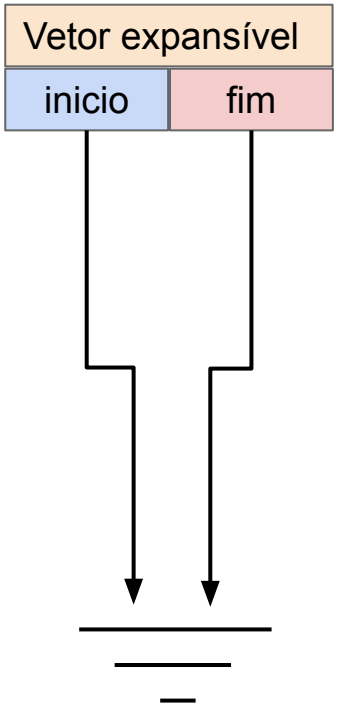
Remoção - Dados Não-Ordenados

Como pôde ser visto no exemplo, o uso de remoção em vetores expansíveis com dados não-ordenados é inviável:

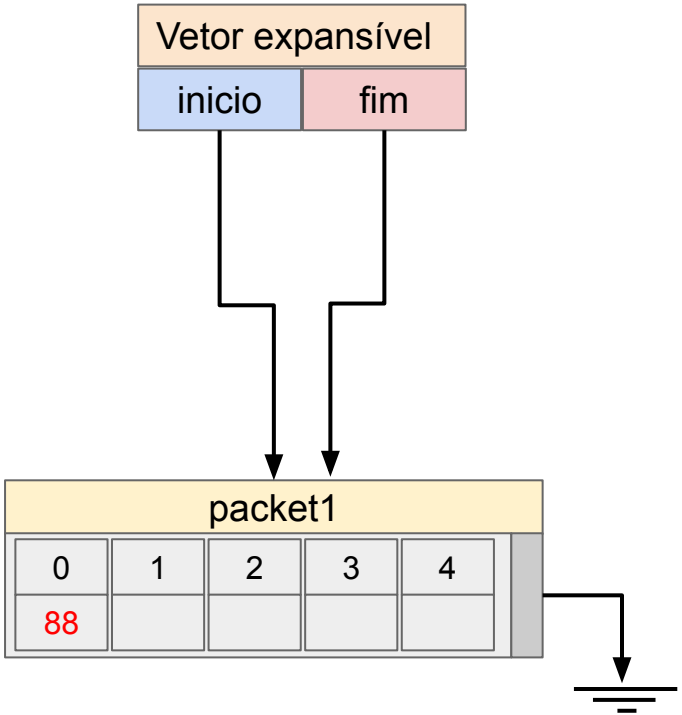
- Ou o percorrimento e busca vão ter que pular nós inválidos, o que torna o processo ineficiente, sendo melhor usar uma lista diretamente.
- Ou será necessário rearranjar os packets a todo momento, sendo melhor usar um vetor diretamente.

Assim, iremos trabalhar com vetores expansíveis com dados ordenados.

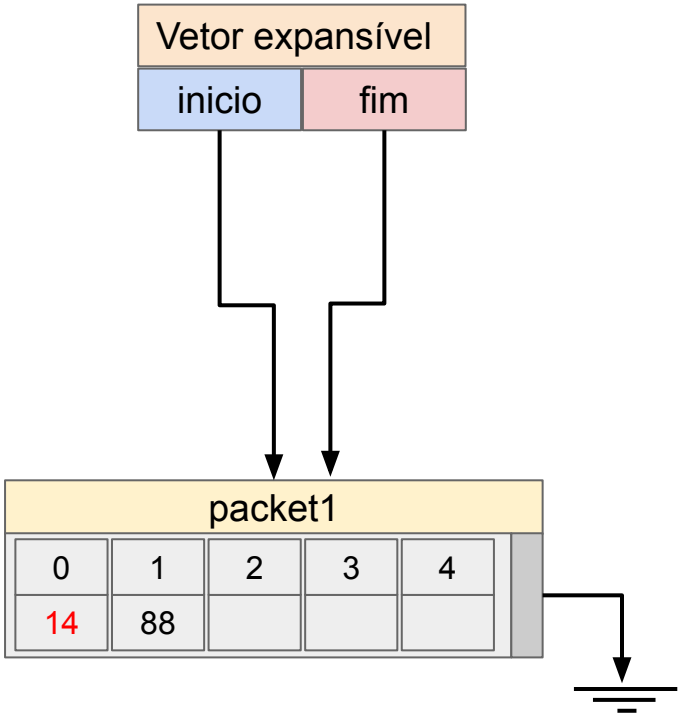
Exemplo - Inserção - I



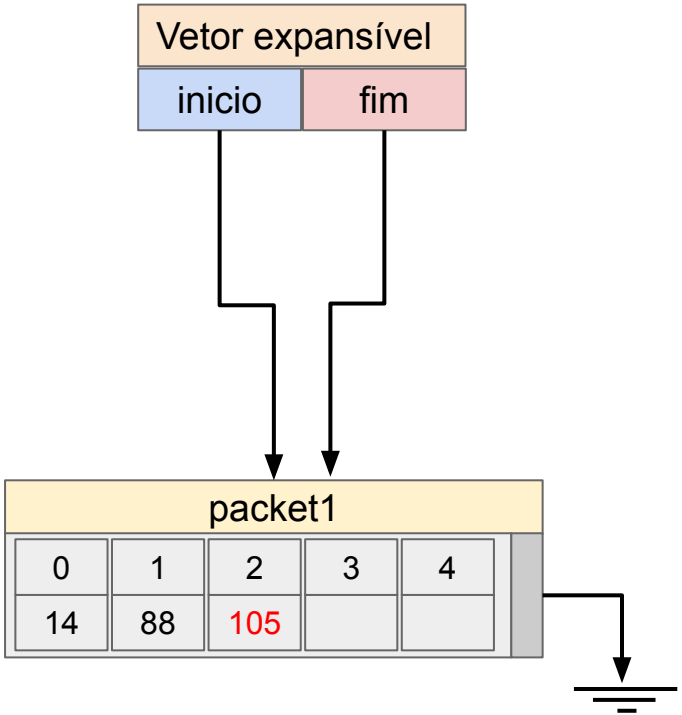
Exemplo - Inserção - ii



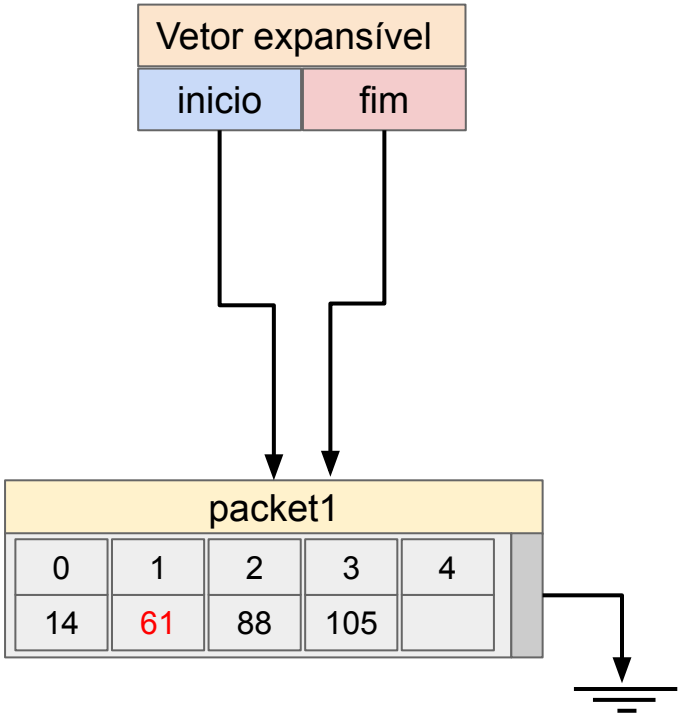
Exemplo - Inserção - III



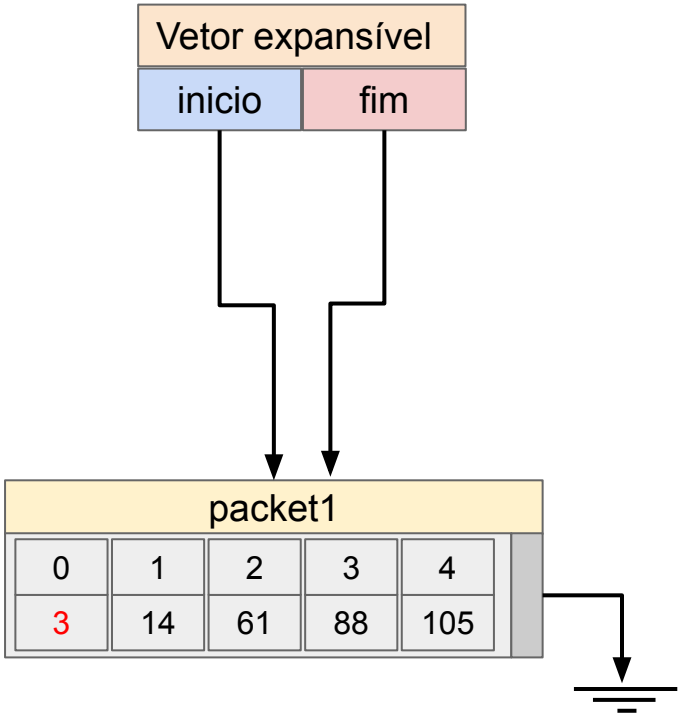
Exemplo - Inserção - iv



Exemplo - Inserção - v

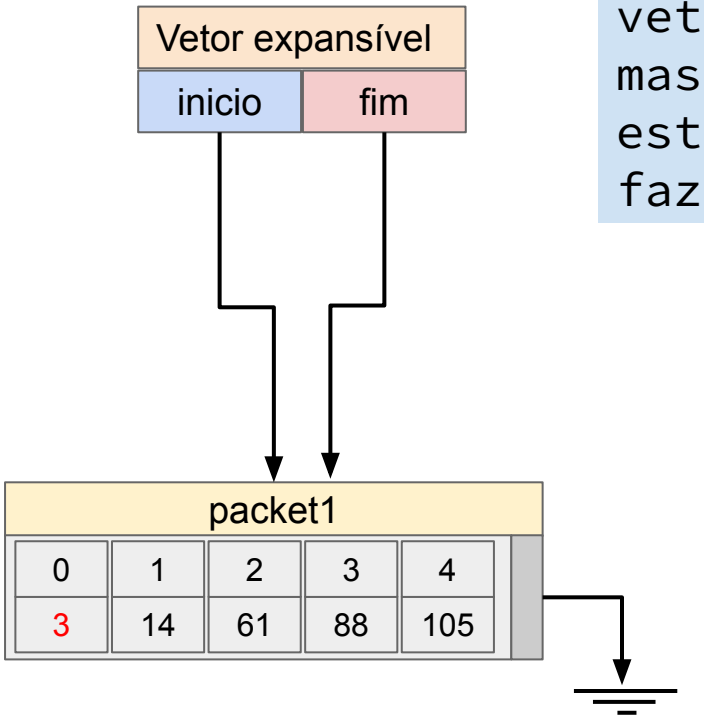


Exemplo - Inserção - VI



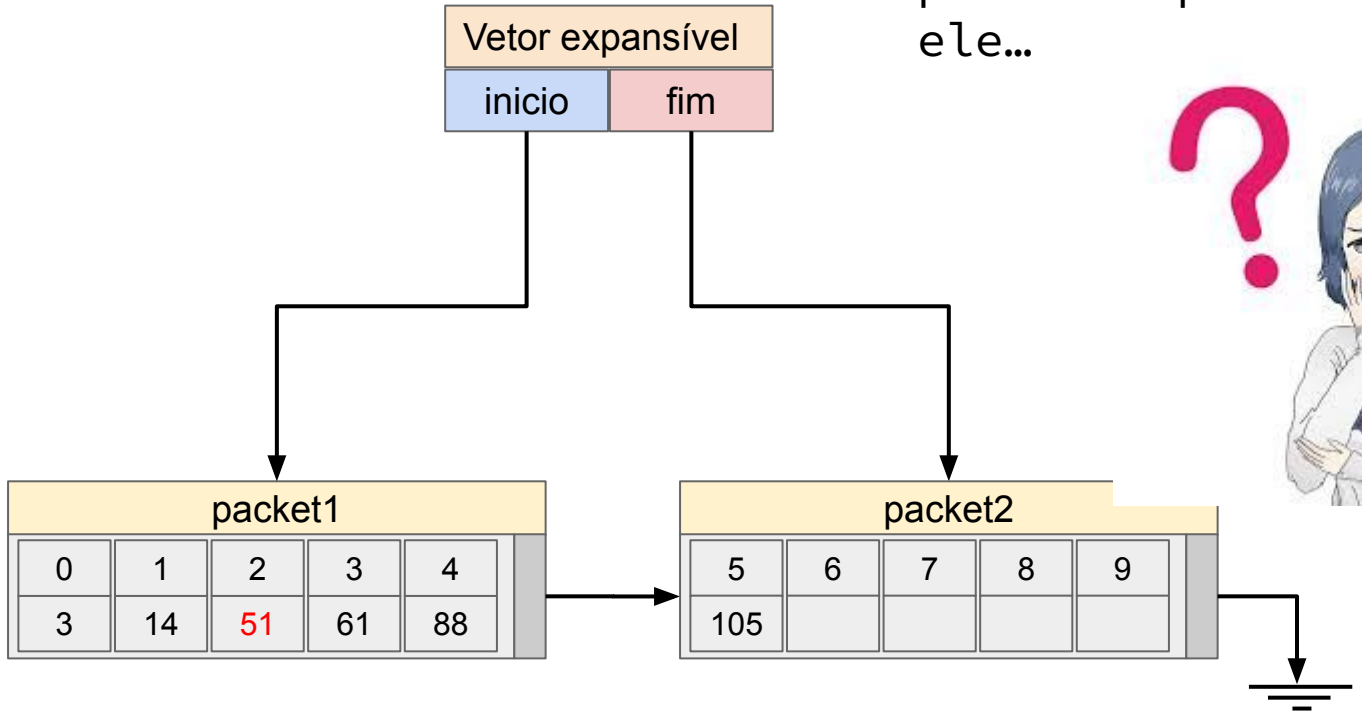
Exemplo - Inserção - VII

Queremos agora inserir o 51 no vetor expansível, mas o packet1 está cheio, como fazer?

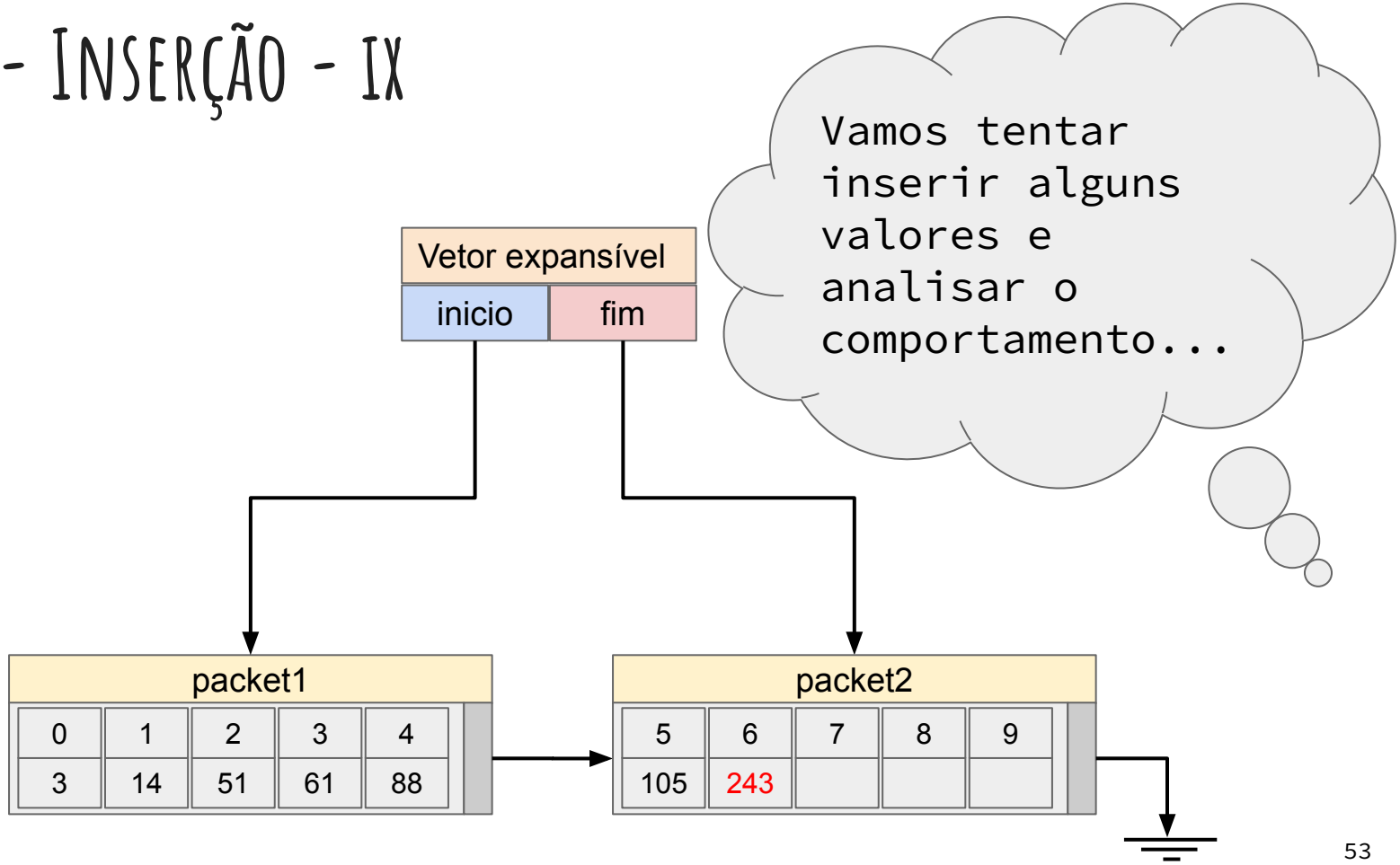


Exemplo - Inserção - VIII

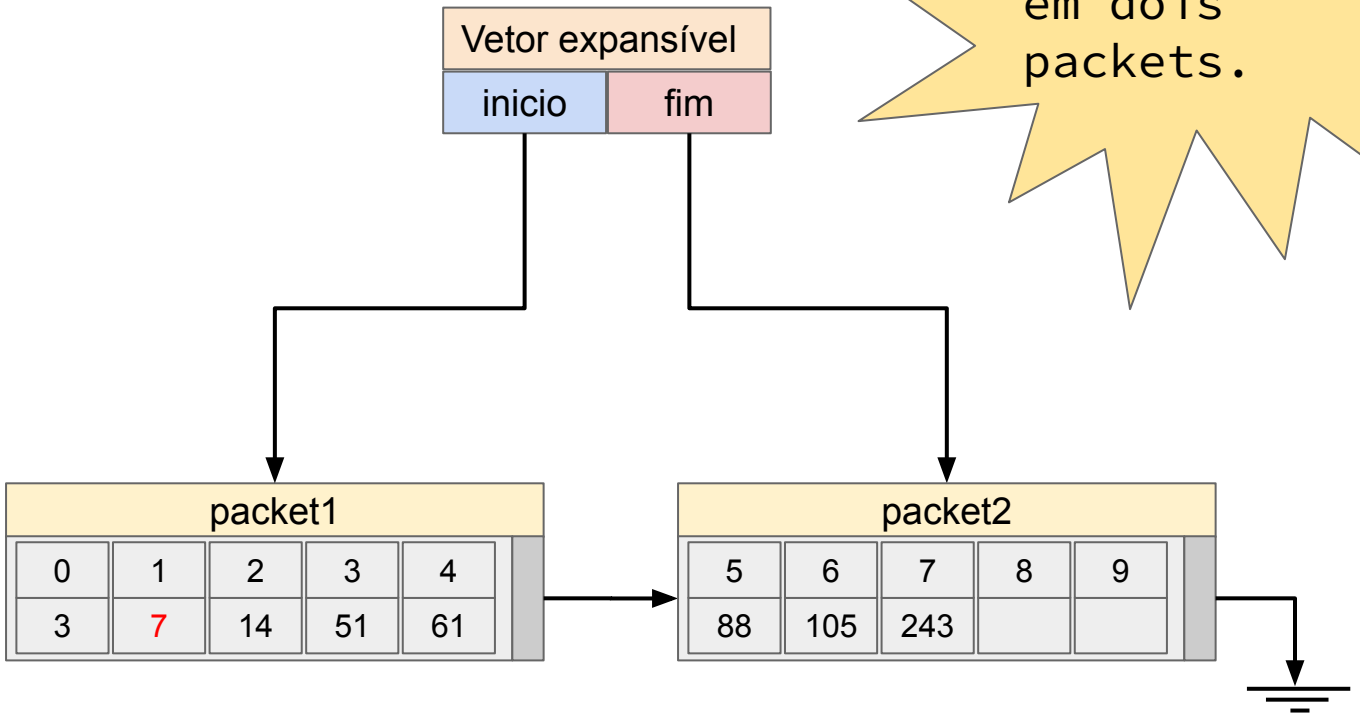
Talvez possamos criar um novo packet e mover o excedente do primeiro packet para ele…



Exemplo - Inserção - ix

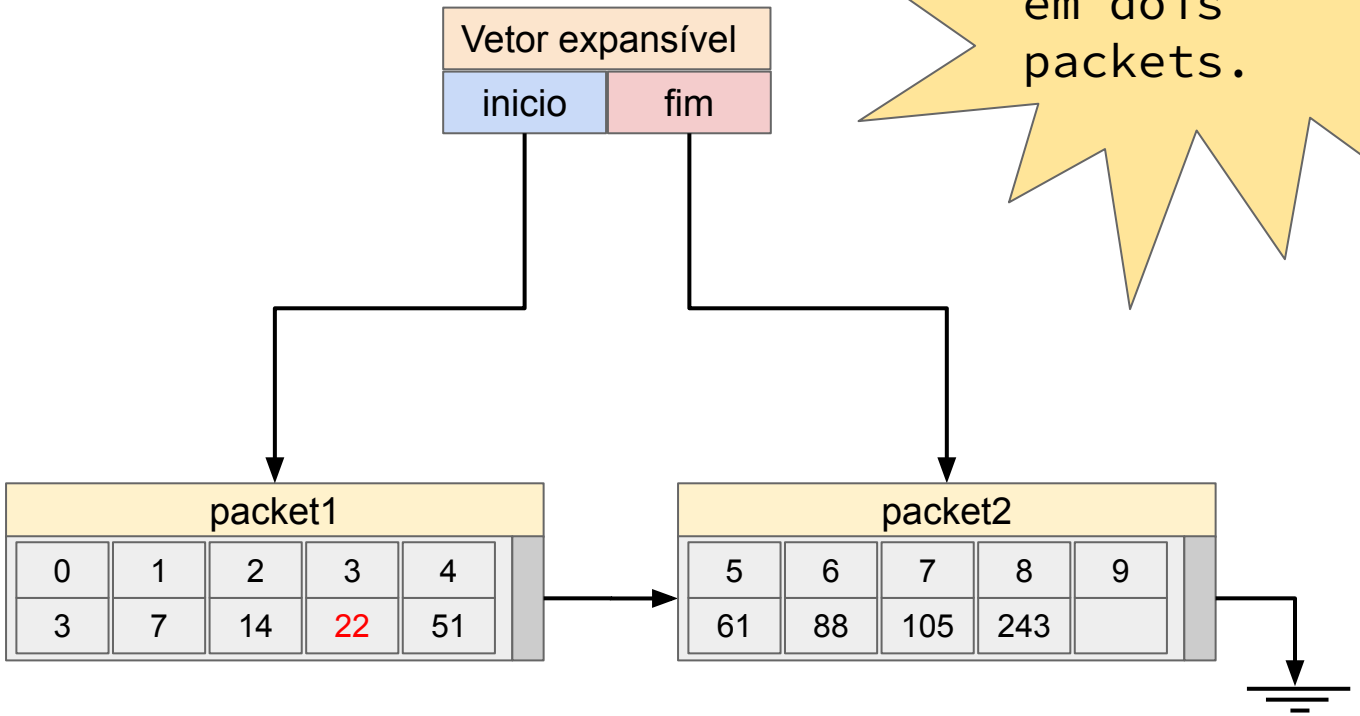


Exemplo - Inserção - x

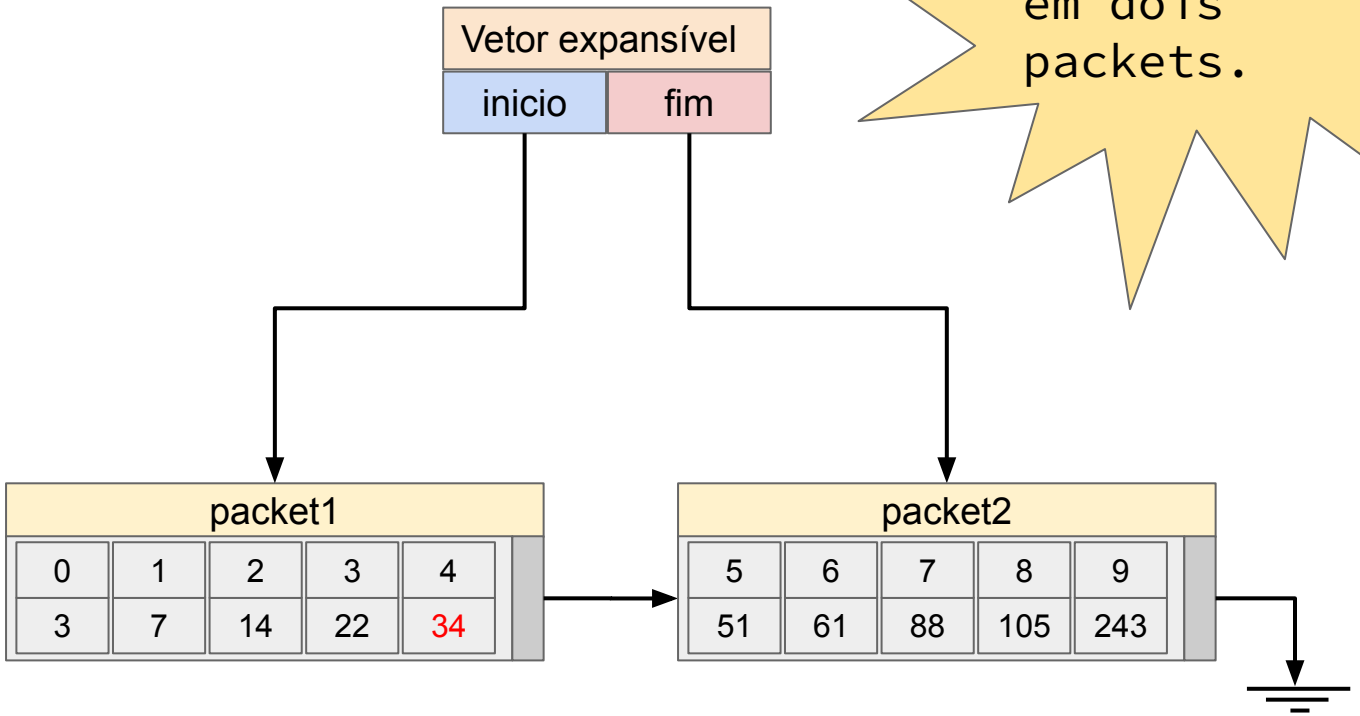


Modificações em dois packets.

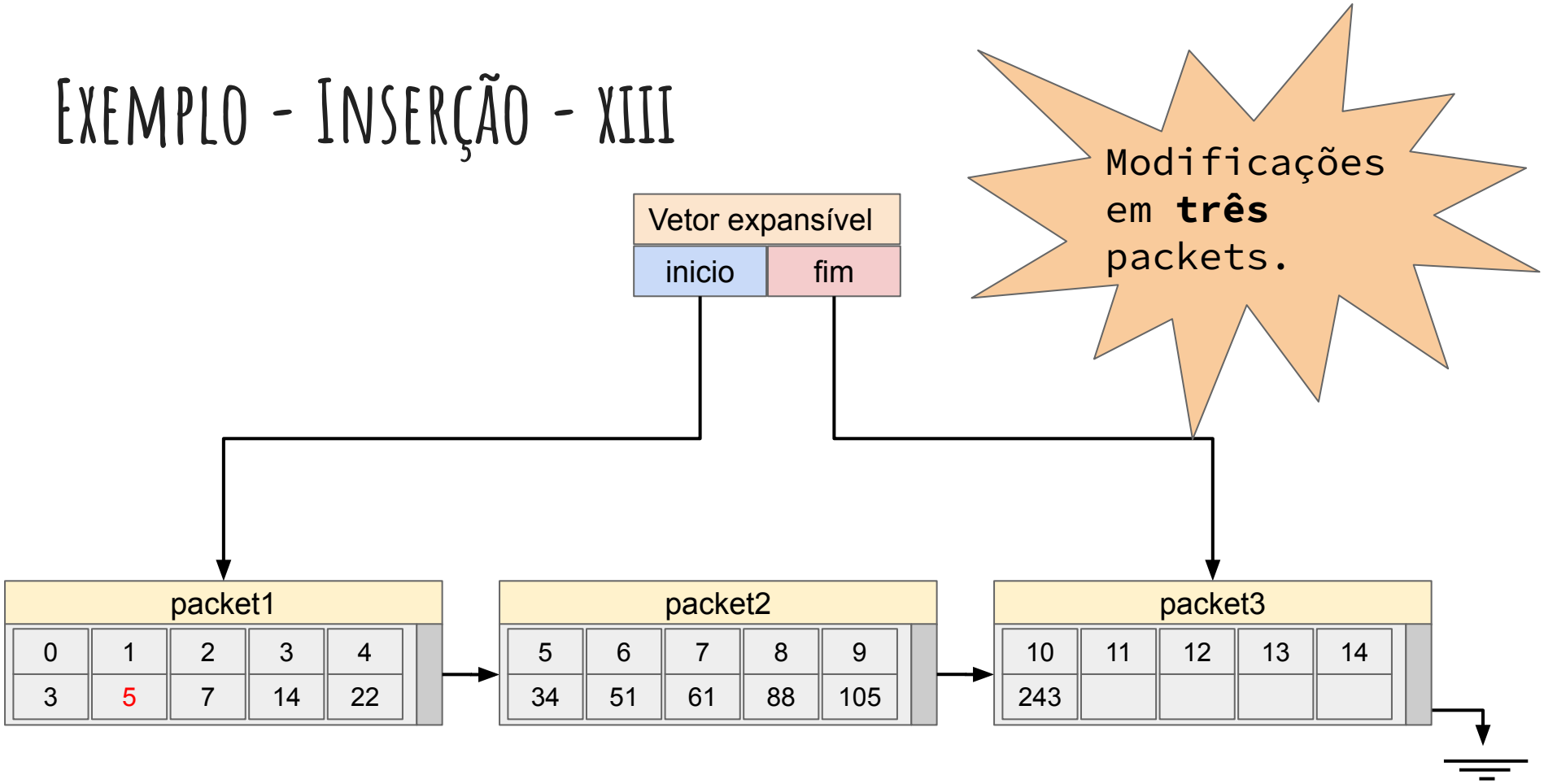
Exemplo - Inserção - xi



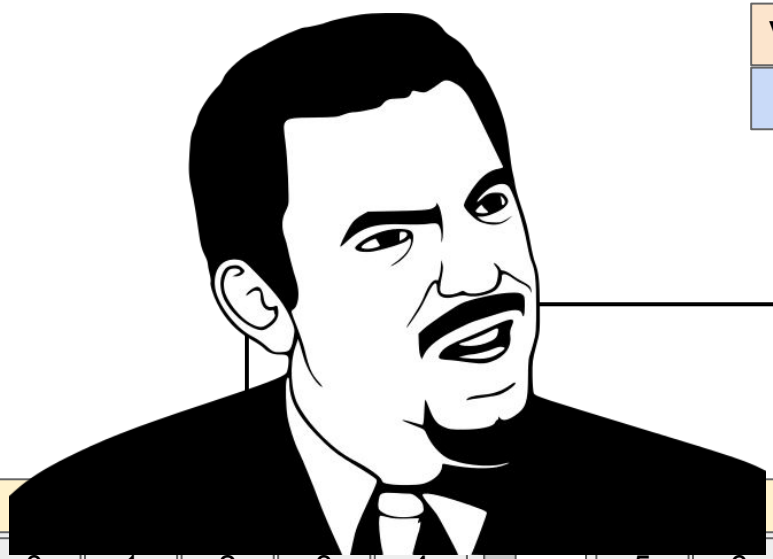
Exemplo - Inserção - xii



Exemplo - Inserção - xiii



Exemplo - Inserção - xiii



Modificações em **três** packets.

Vetor expansível

inicio	fim

packet2				
0	1	2	3	4
3	5	7	14	22

5	6	7	8	9
34	51	61	88	105

packet3				
10	11	12	13	14
243				

Inserção em Packet Cheio - I

Ficar empurrando o dado à frente, vai tirar a eficiência da estrutura, que irá funcionar como um vetor normal.

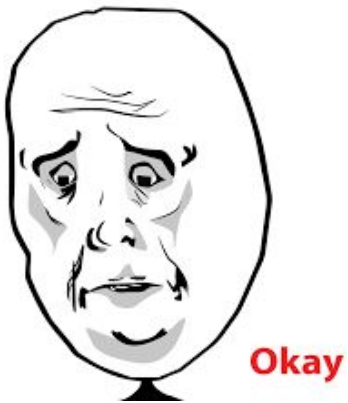
Uma implementação adequada irá dividir o packet em dois quando ele fica cheio.

Isso implica que você pode ter vários packets que não estão completos.

Inserção em Packet Cheio - II

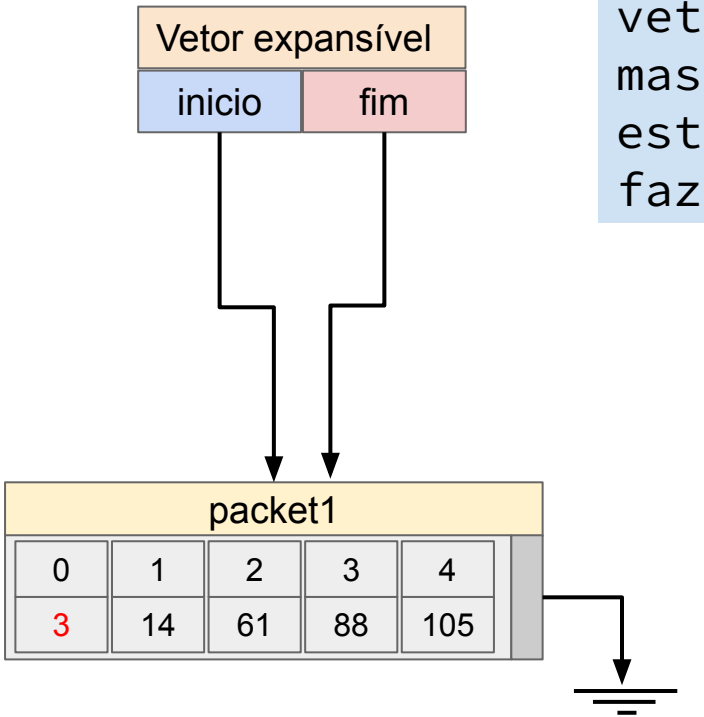
Se os packets não estão cheios, isso tem dois impactos para o vetor expansível:

- Cada packet terá que controlar quantos elementos são válidos;
- Não será possível acessar o vetor expansível como um todo por posição. Só é possível usar posição dentro de cada packet.



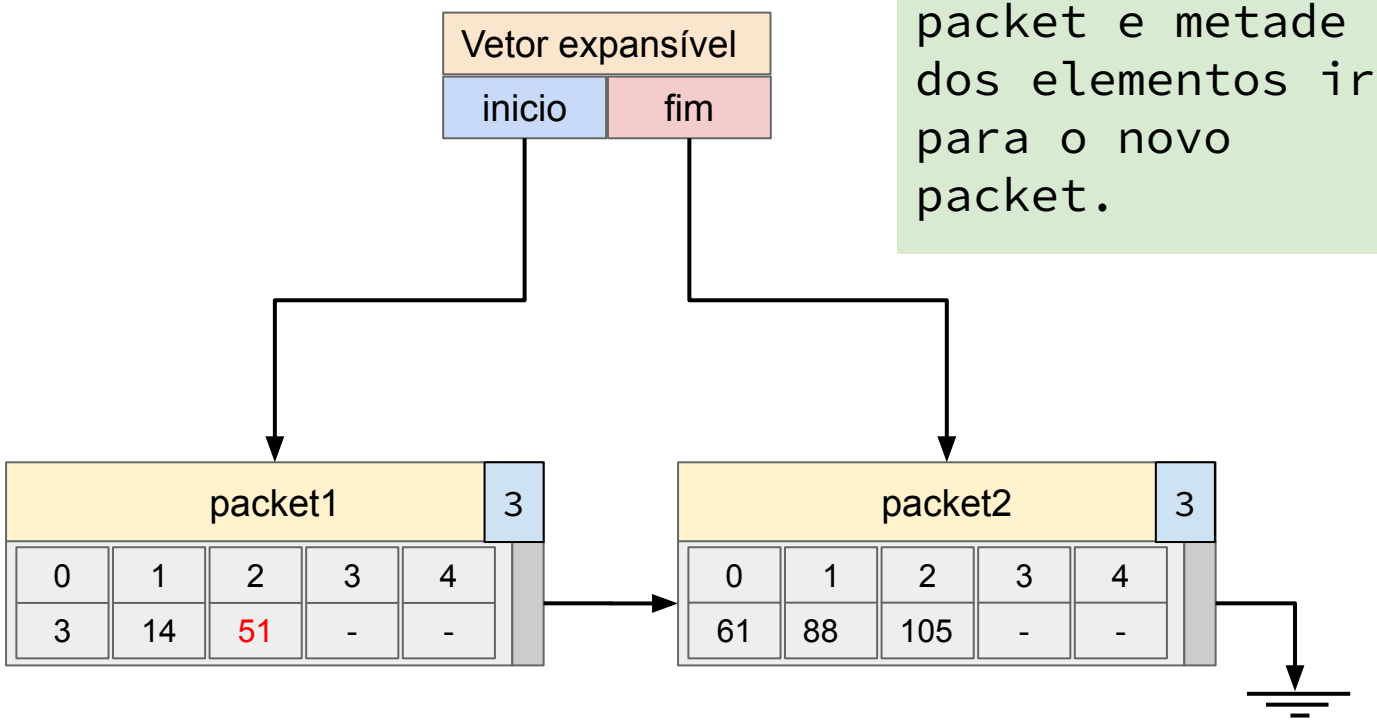
Refazendo o Exemplo- Inserção - 1

Queremos agora inserir o 51 no vetor expansível, mas o packet1 está cheio, como fazer?



Refazendo o Exemplo- Inserção - ii

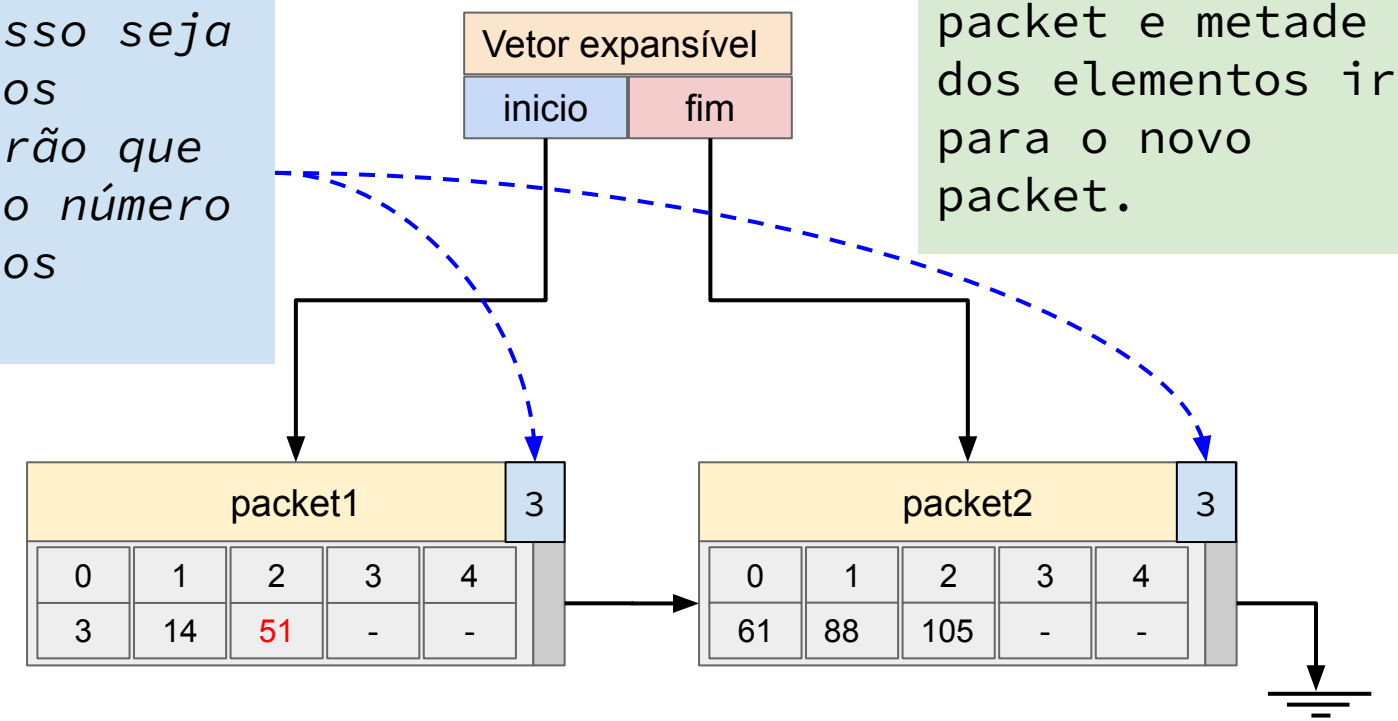
Dividimos o packet e metade dos elementos irá para o novo packet.



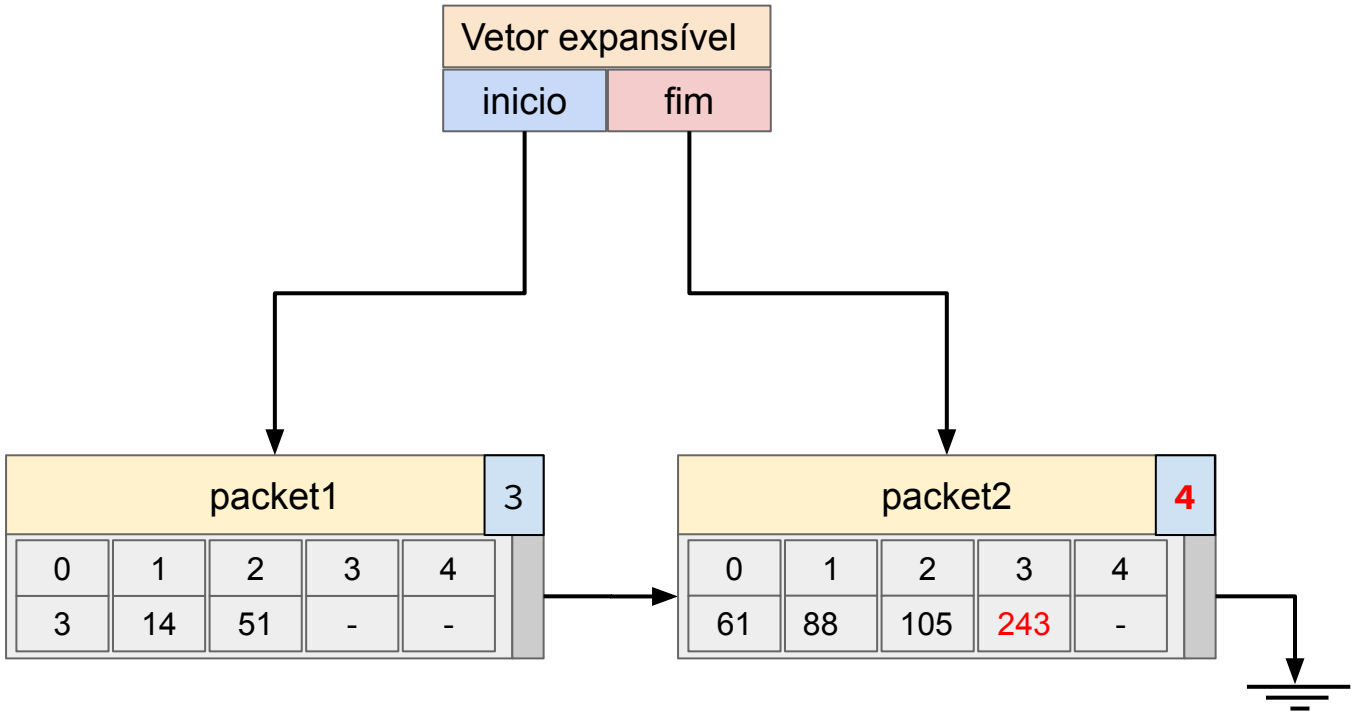
Refazendo o Exemplo- Inserção - II

Para que isso seja possível, os pacotes terão que armazenar o número de elementos válidos!

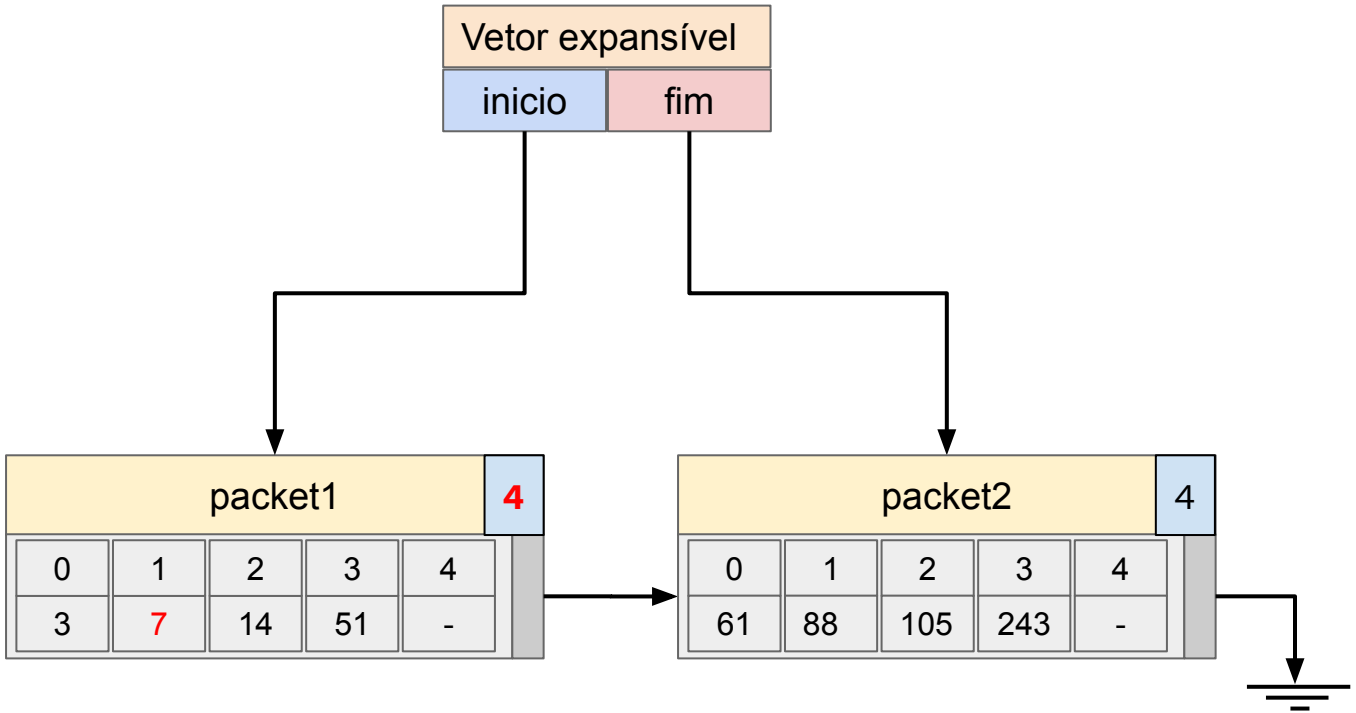
Dividimos o packet e metade dos elementos irá para o novo packet.



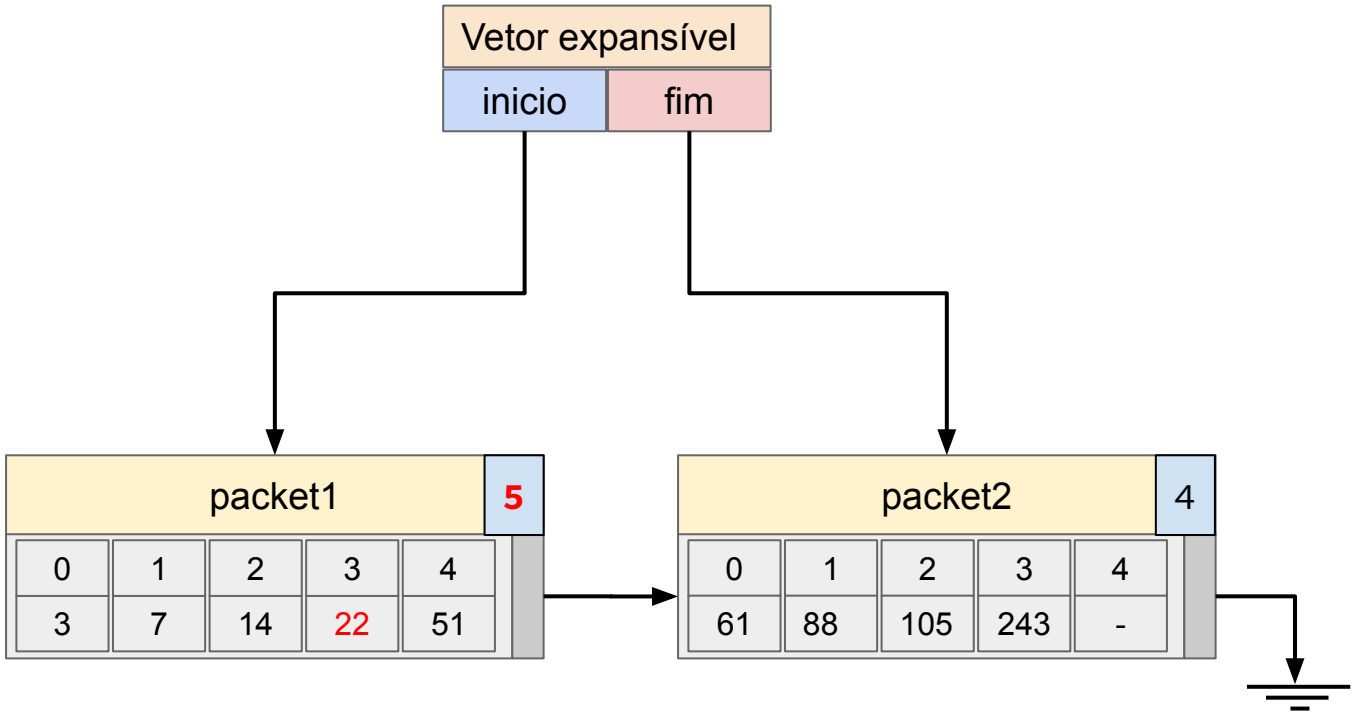
Refazendo o Exemplo- Inserção - iii



Refazendo o Exemplo - Inserção - iv

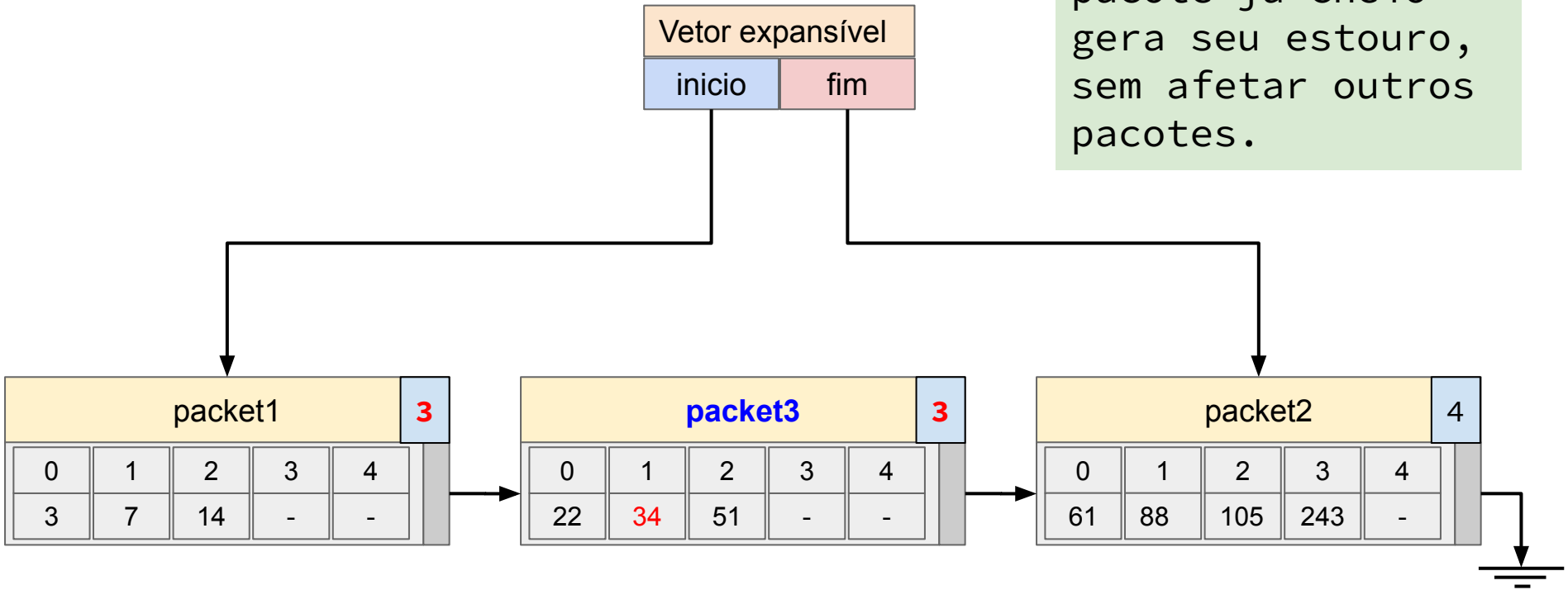


Refazendo o Exemplo- Inserção - v

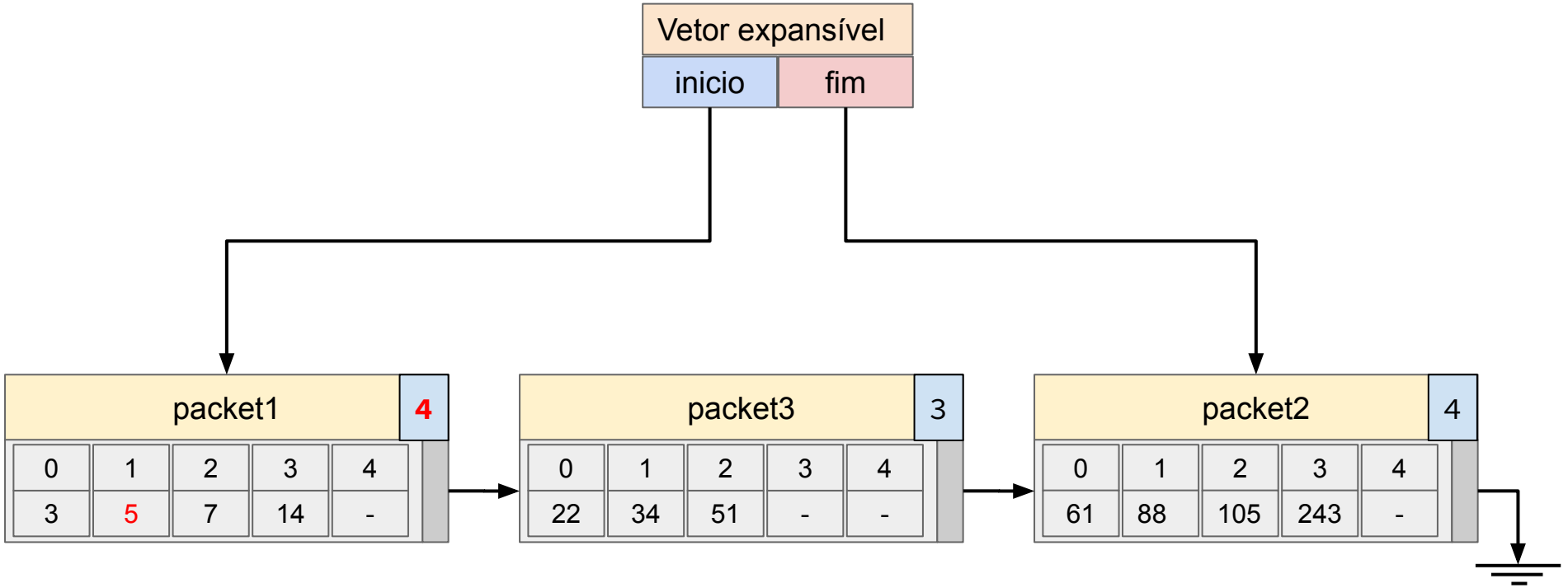


Refazendo o Exemplo- Inserção - VI

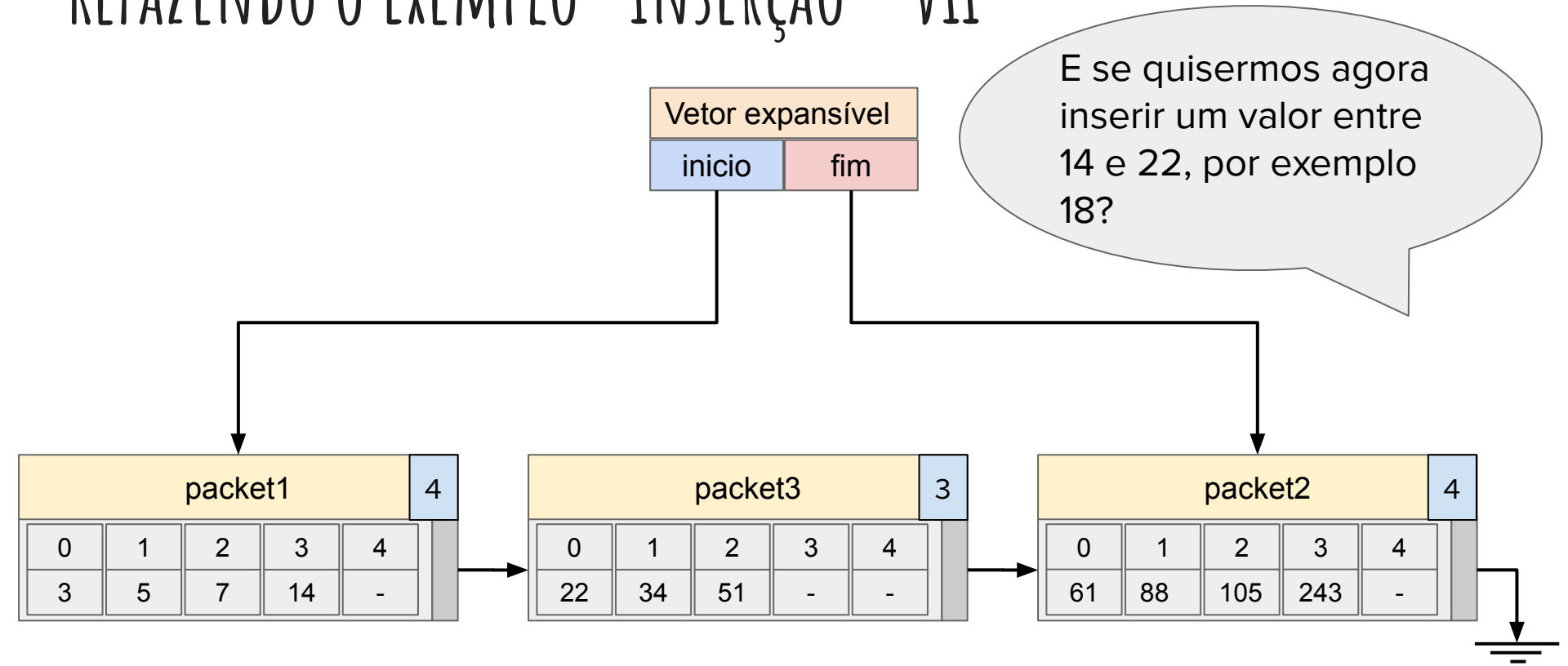
Inserção em um pacote já cheio gera seu estouro, sem afetar outros pacotes.



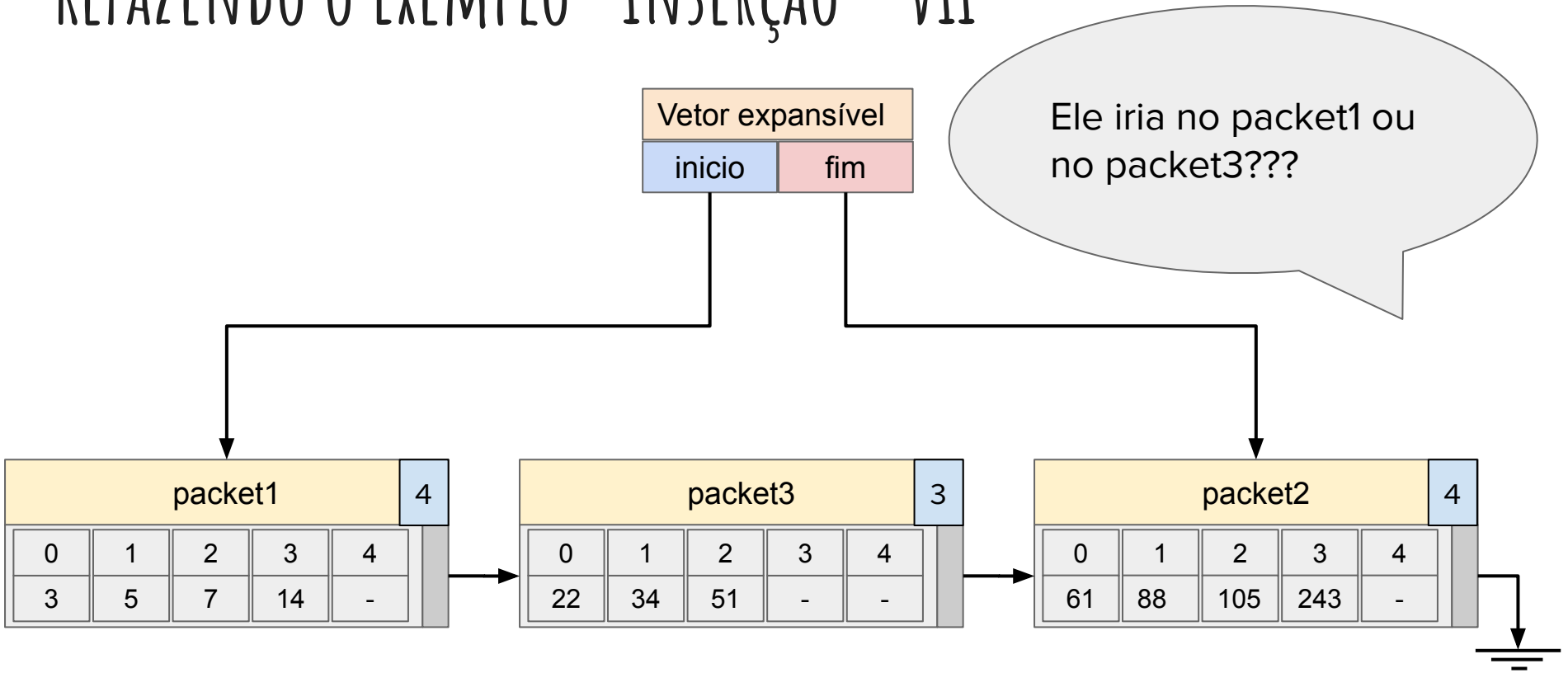
Refazendo o Exemplo- Inserção - vii



Refazendo o Exemplo- Inserção - vii



Refazendo o Exemplo- Inserção - vii



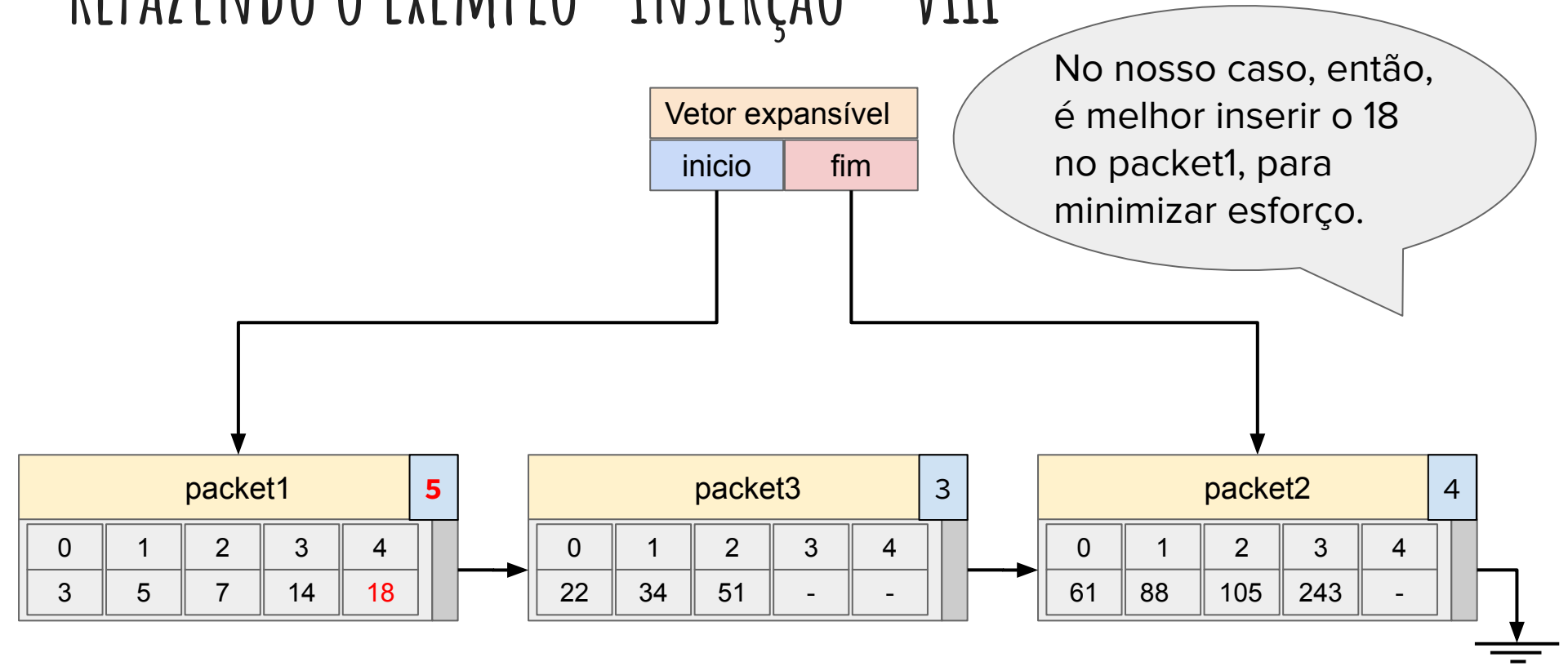
Pensando na Inserção

Quando a inserção é entre elementos de um pacote, não há dúvidas quanto ao melhor caminho.

Quando o dado a ser inserido poderia ser armazenado em um pacote, no próximo ou anterior, é necessário avaliar o que irá gerar menos impacto computacional.

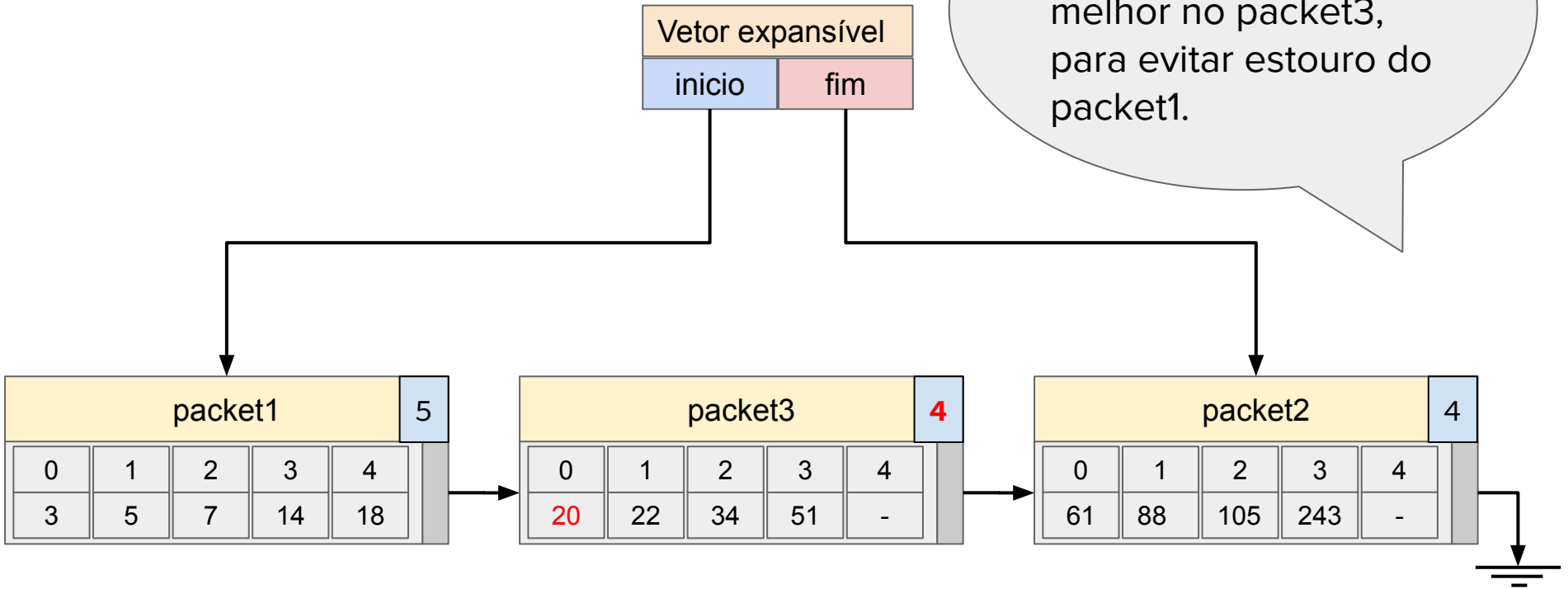
Por exemplo, é melhor inserir no fim de um pacote que empurrar todos os elementos de um próximo pacote. Mas é melhor empurrar que fazer o processo de divisão.

Refazendo o Exemplo- Inserção - VIII



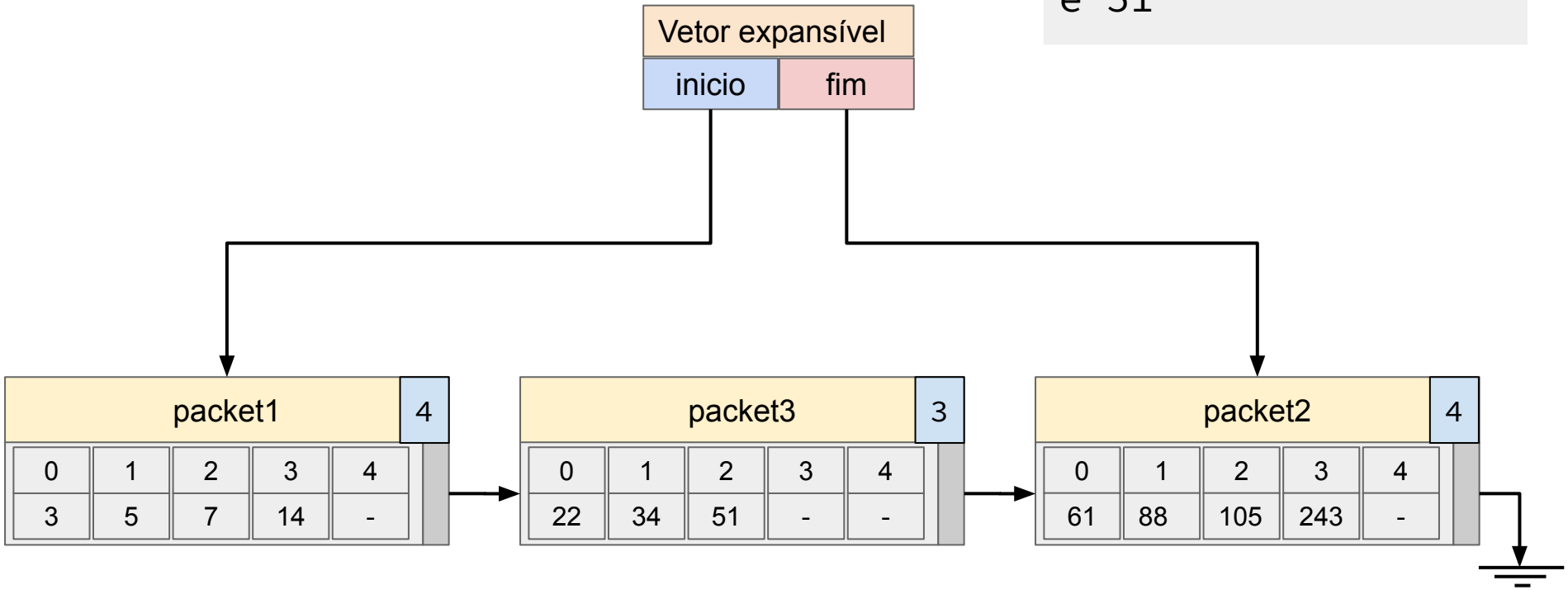
Refazendo o Exemplo- Inserção - ix

A inserção do 20, na sequência, seria melhor no packet3, para evitar estouro do packet1.



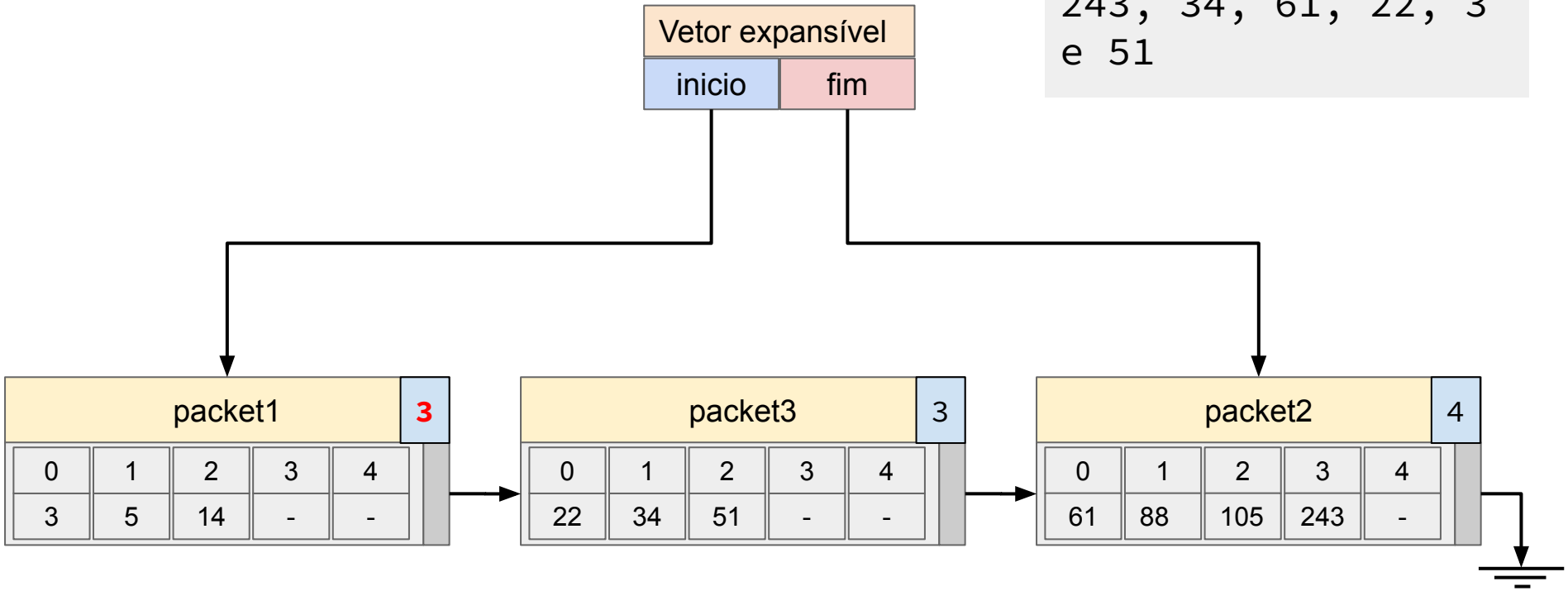
Exemplo- Remoção - I

Considere a remoção dos elementos: 7, 243, 34, 61, 22, 3 e 51



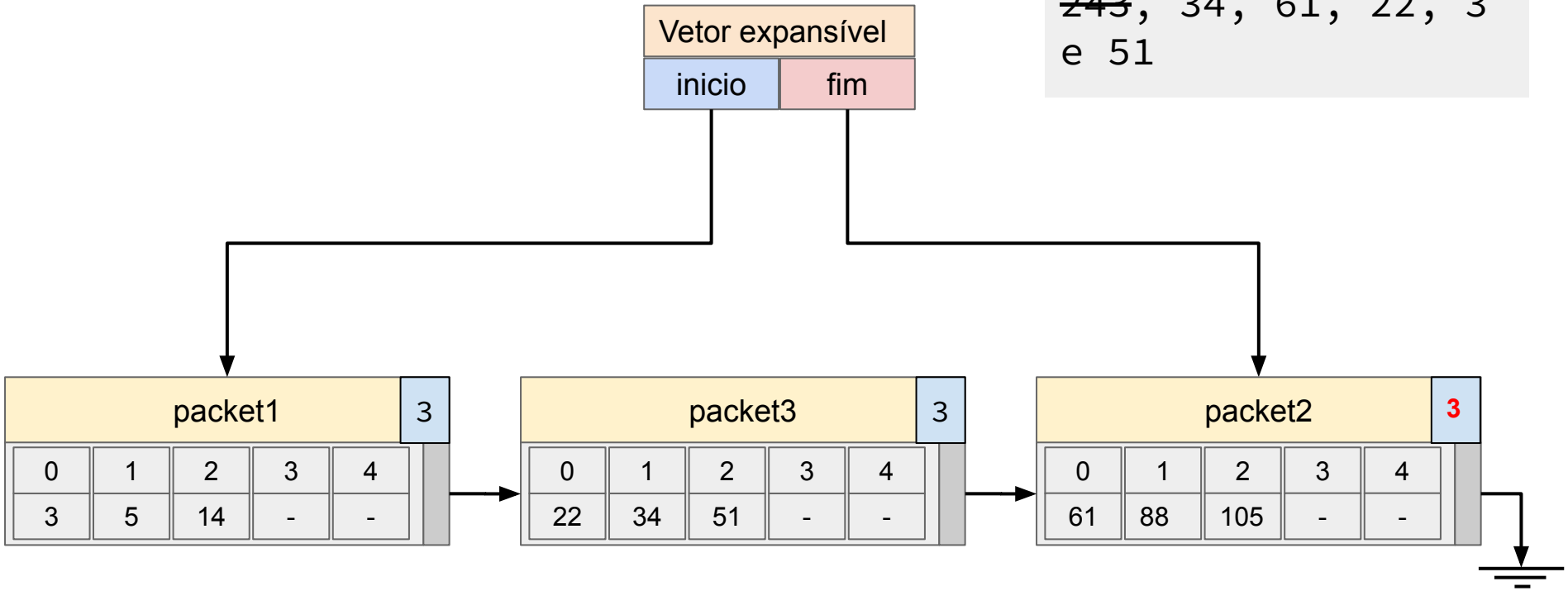
Exemplo- Remoção - II

Considere a remoção dos elementos: ~~7~~, 243, 34, 61, 22, 3 e 51



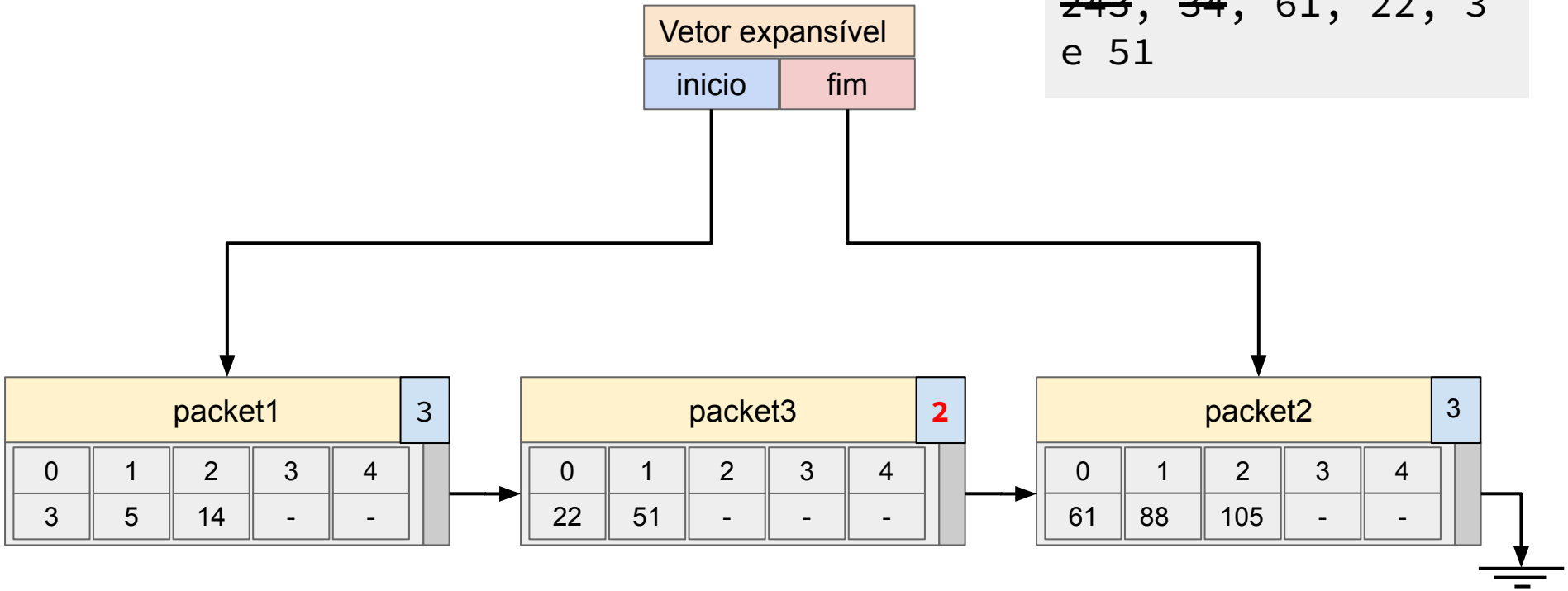
Exemplo- Remoção - III

Considere a remoção dos elementos: ~~7~~, ~~243~~, 34, 61, 22, 3 e 51



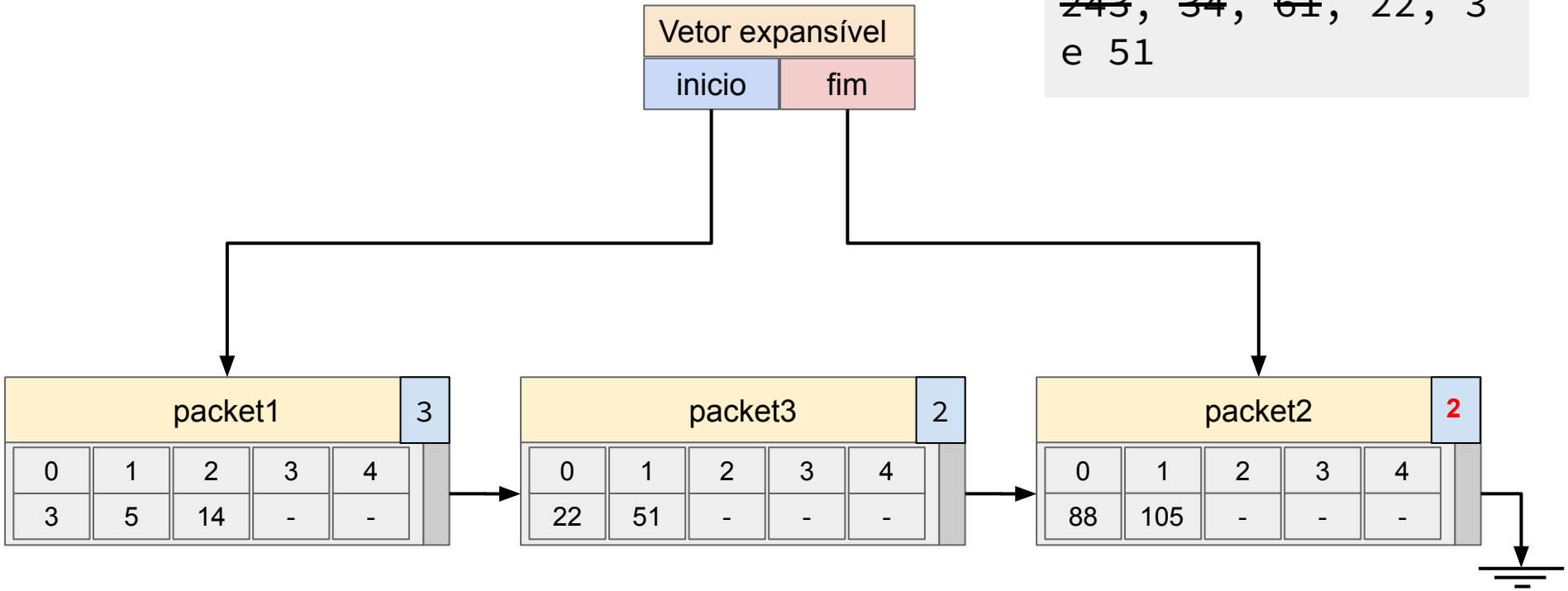
Exemplo- Remoção - IV

Considere a remoção dos elementos: ~~7~~, ~~243~~, ~~34~~, 61, 22, 3 e 51



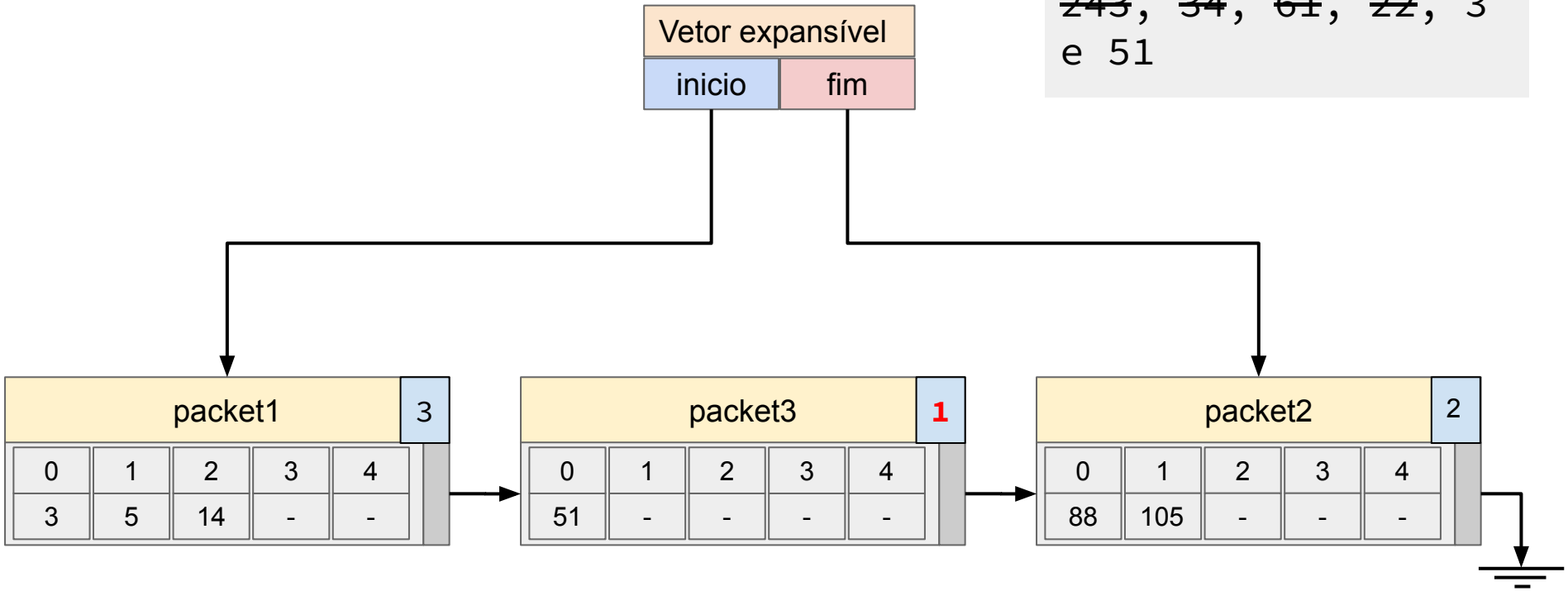
Exemplo- Remoção - V

Considere a remoção dos elementos: ~~7~~, ~~243~~, ~~34~~, ~~61~~, 22, 3 e 51



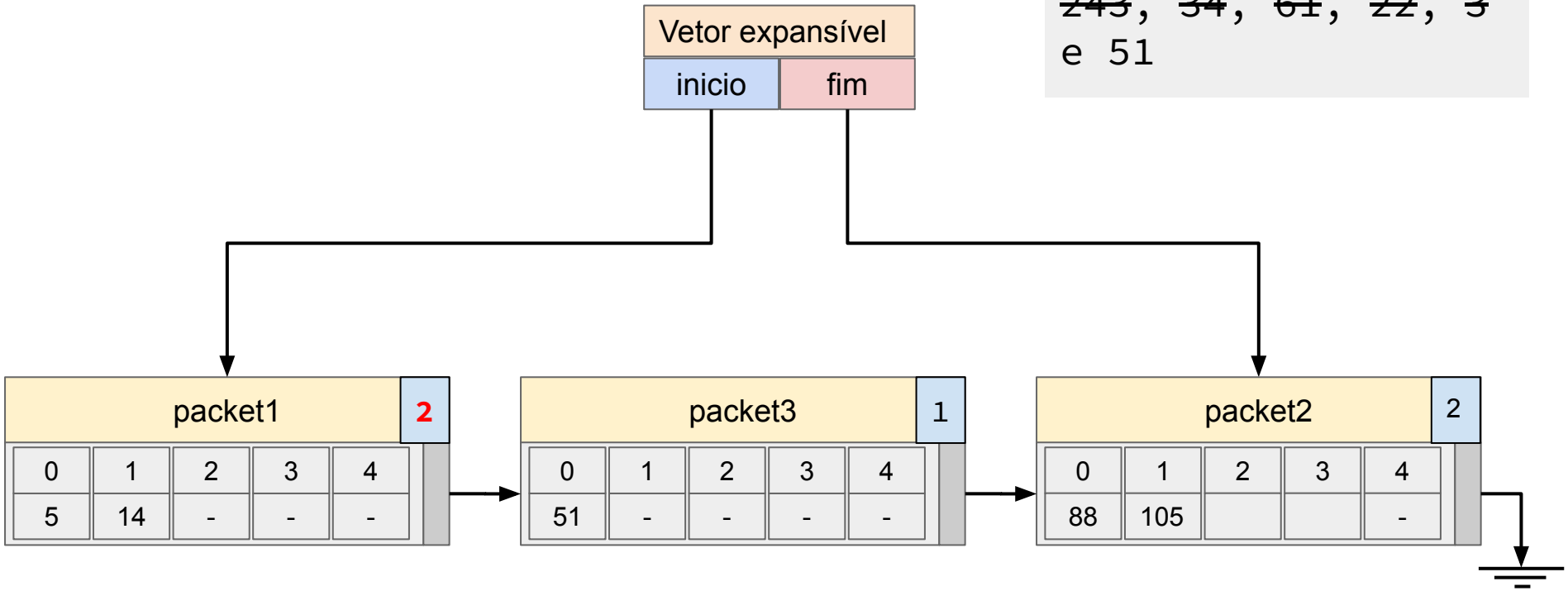
Exemplo- Remoção - VI

Considere a remoção dos elementos: ~~7~~, ~~243~~, ~~34~~, ~~61~~, ~~22~~, 3 e 51



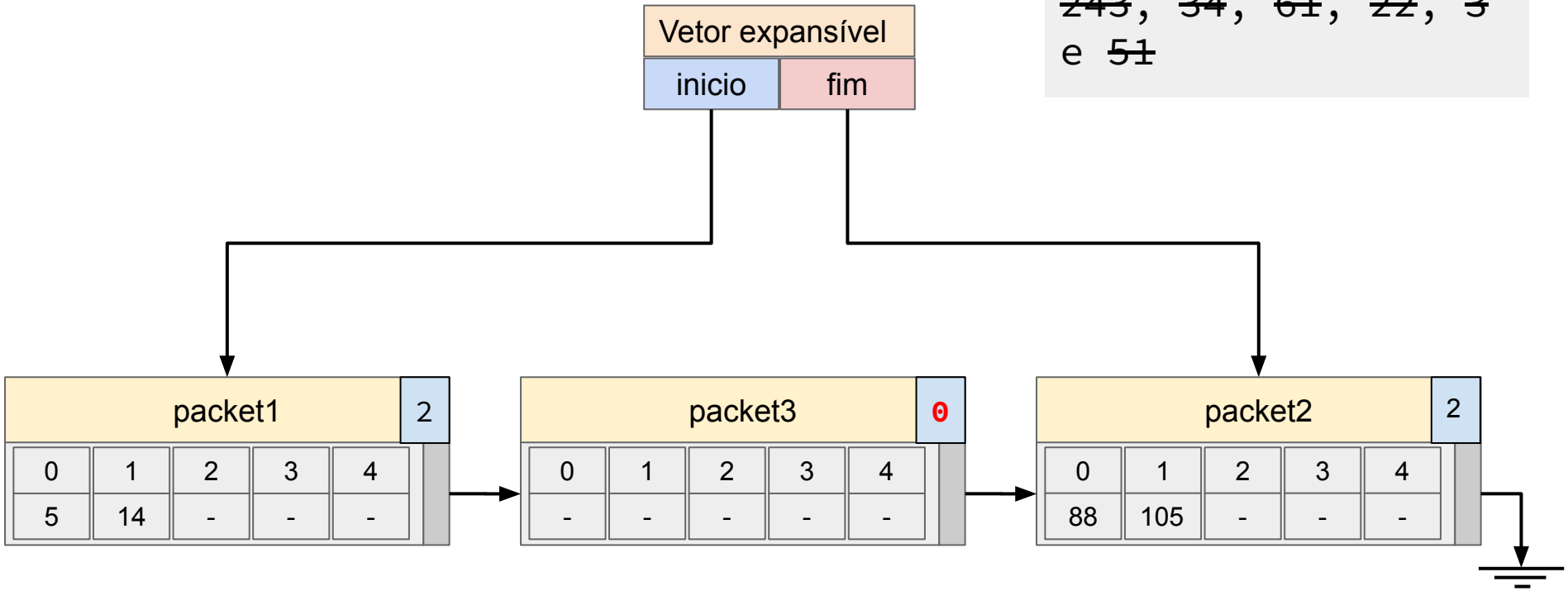
Exemplo- Remoção - VII

Considere a remoção dos elementos: ~~7~~, ~~243~~, ~~34~~, ~~61~~, ~~22~~, ~~3~~ e 51



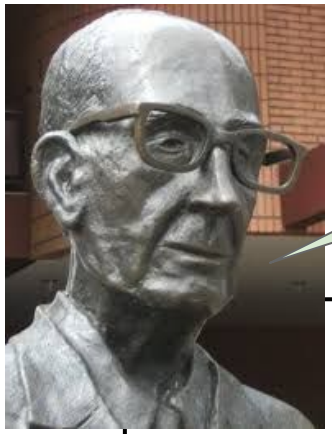
Exemplo- Remoção - VIII

Considere a remoção dos elementos: ~~7~~, ~~243~~, ~~34~~, ~~61~~, ~~22~~, ~~3~~ e ~~51~~



Exemplo- Remoção - VIII

Considere a remoção dos elementos: ~~7~~, ~~243~~, ~~34~~, ~~61~~, ~~22~~, ~~3~~ e ~~51~~



E agora, José?

Vetor expansível

inicio	fim

packet1	2

0	1	2	3	4
5	14	-	-	-

packet3	0

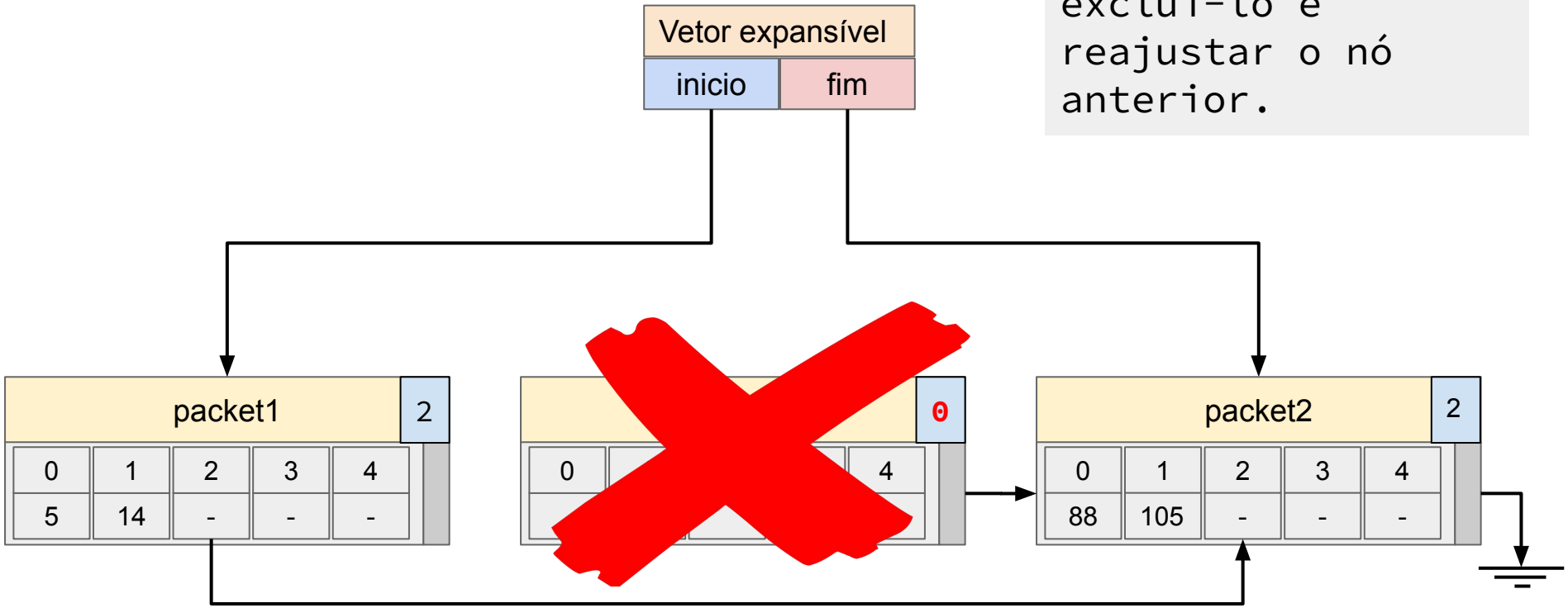
0	1	2	3	4
-	-	-	-	-

packet2	2

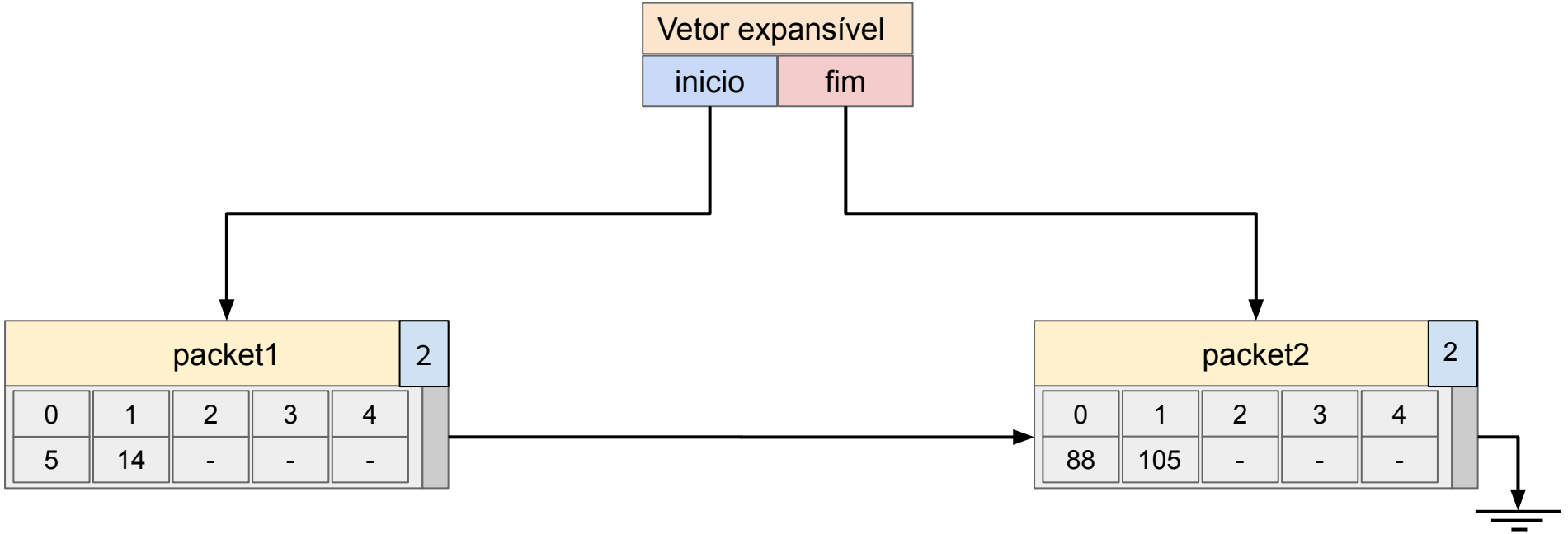
0	1	2	3	4
88	105	-	-	-

Exemplo- Remoção - VIII

Caso um pacote fique vazio, basta excluí-lo e reajustar o nó anterior.

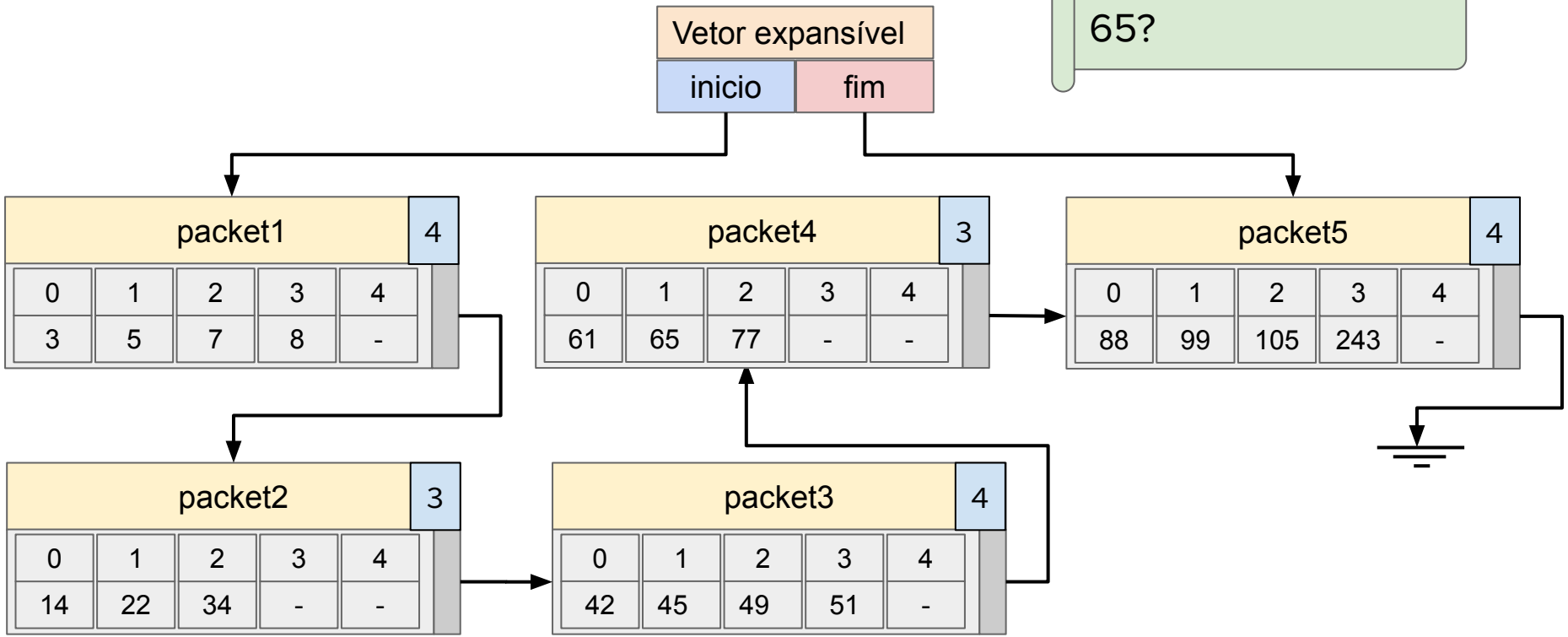


Exemplo- Remoção - VIII

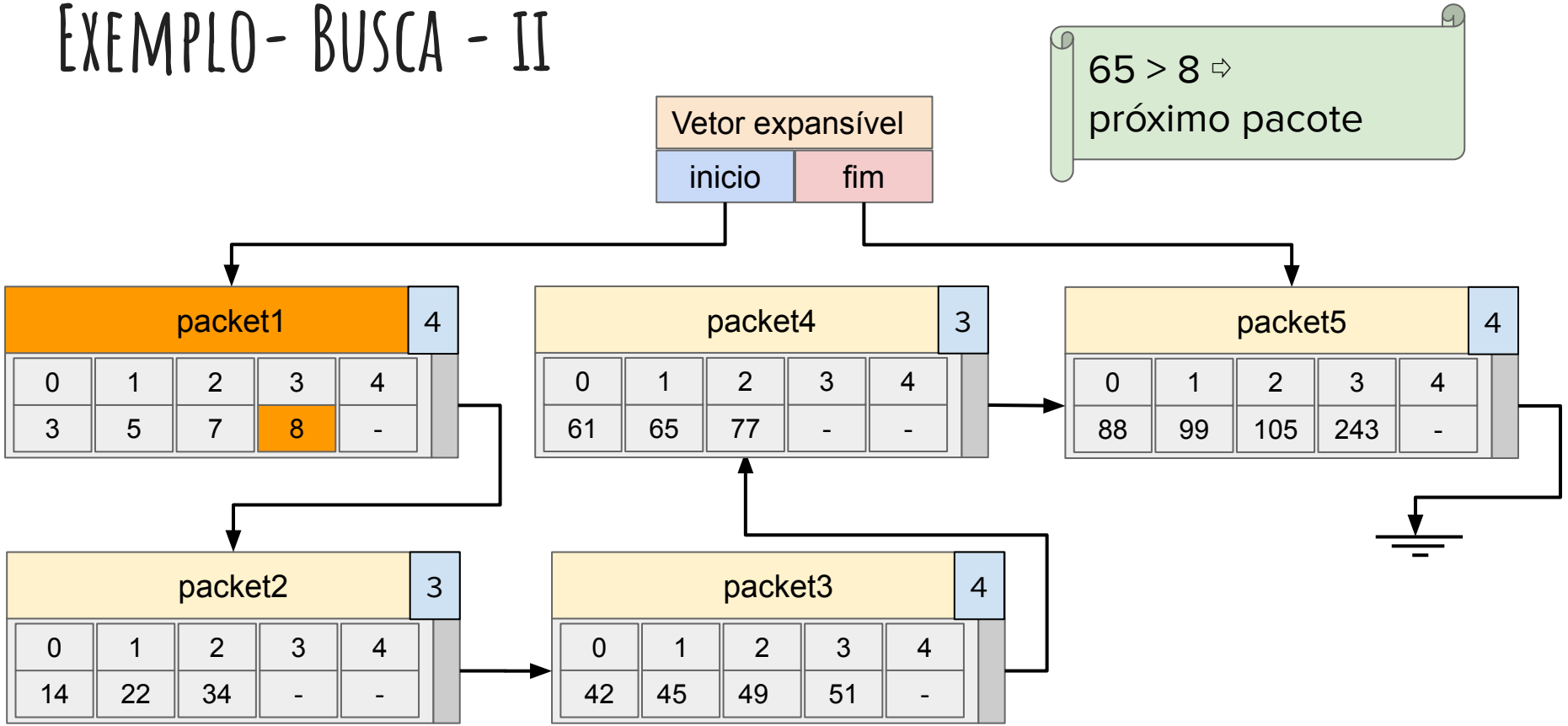


Exemplo- Busca - I

Vamos buscar o 65?

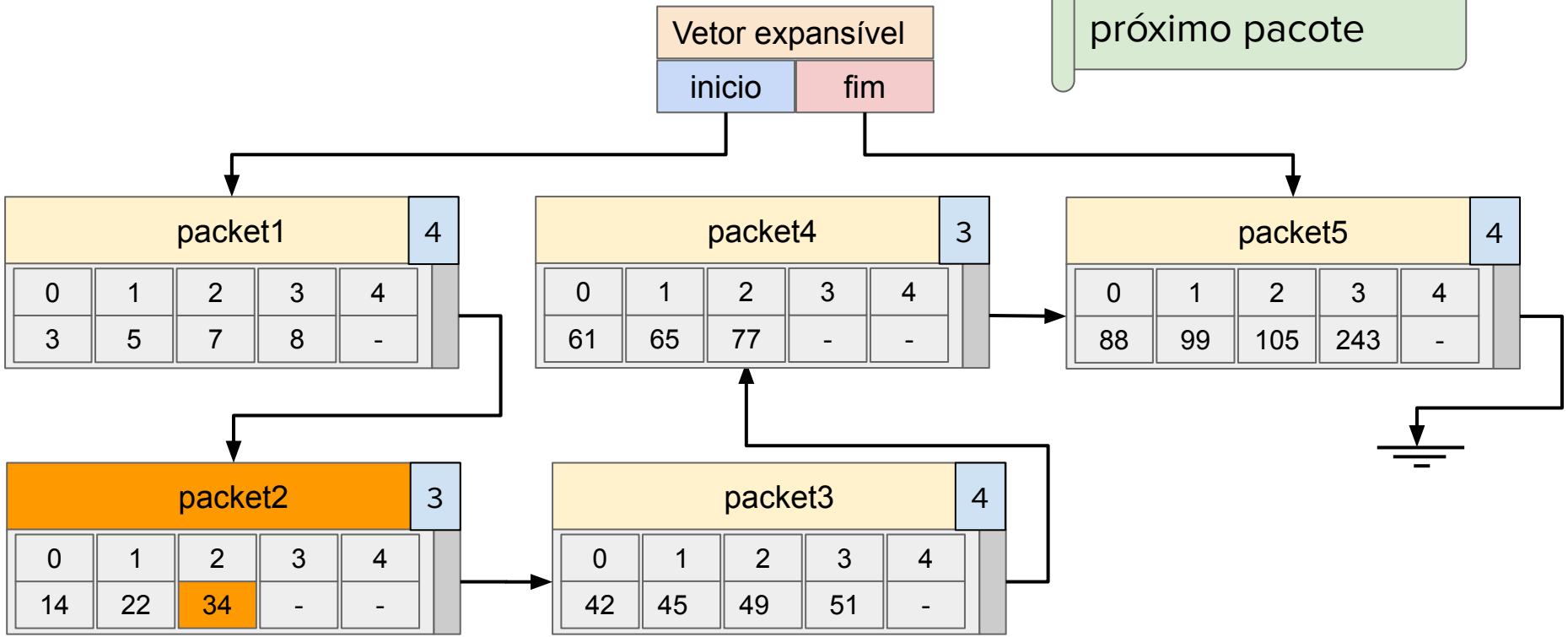


Exemplo- Busca - II



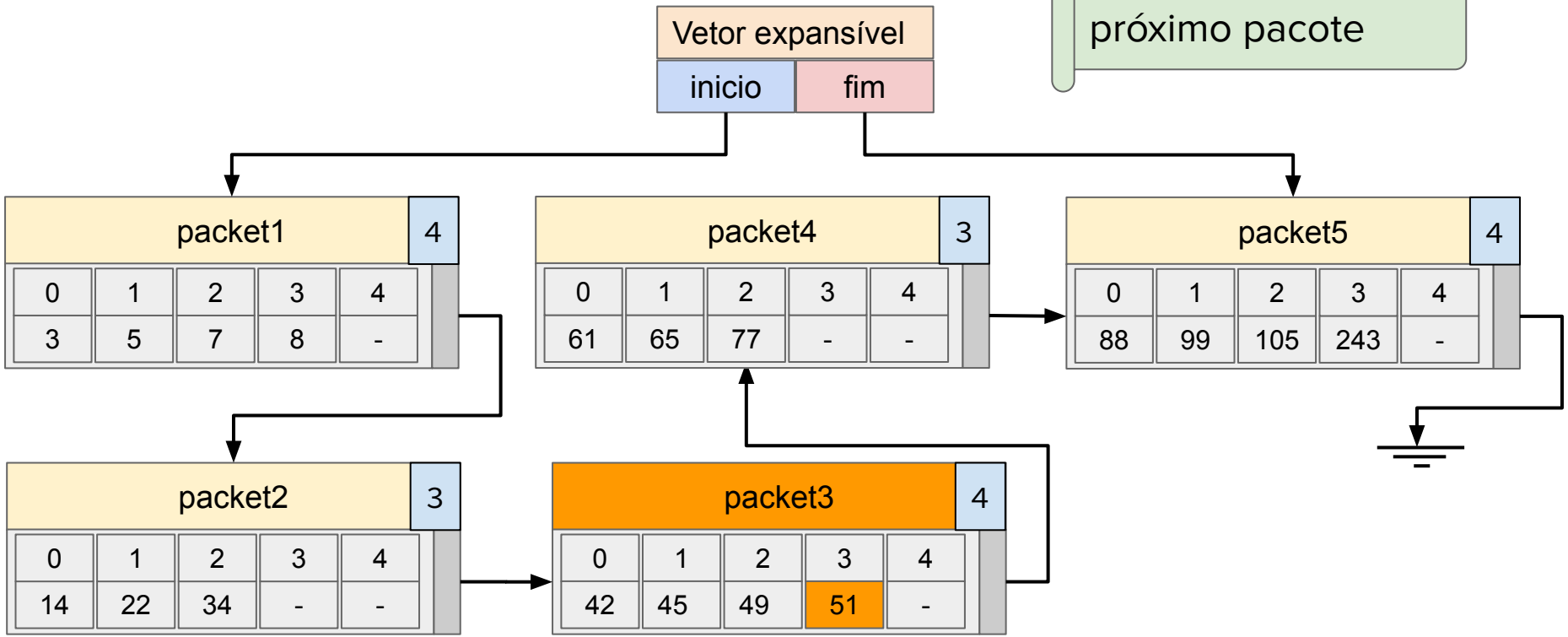
Exemplo- Busca - III

65 > 34 ⇨
próximo pacote

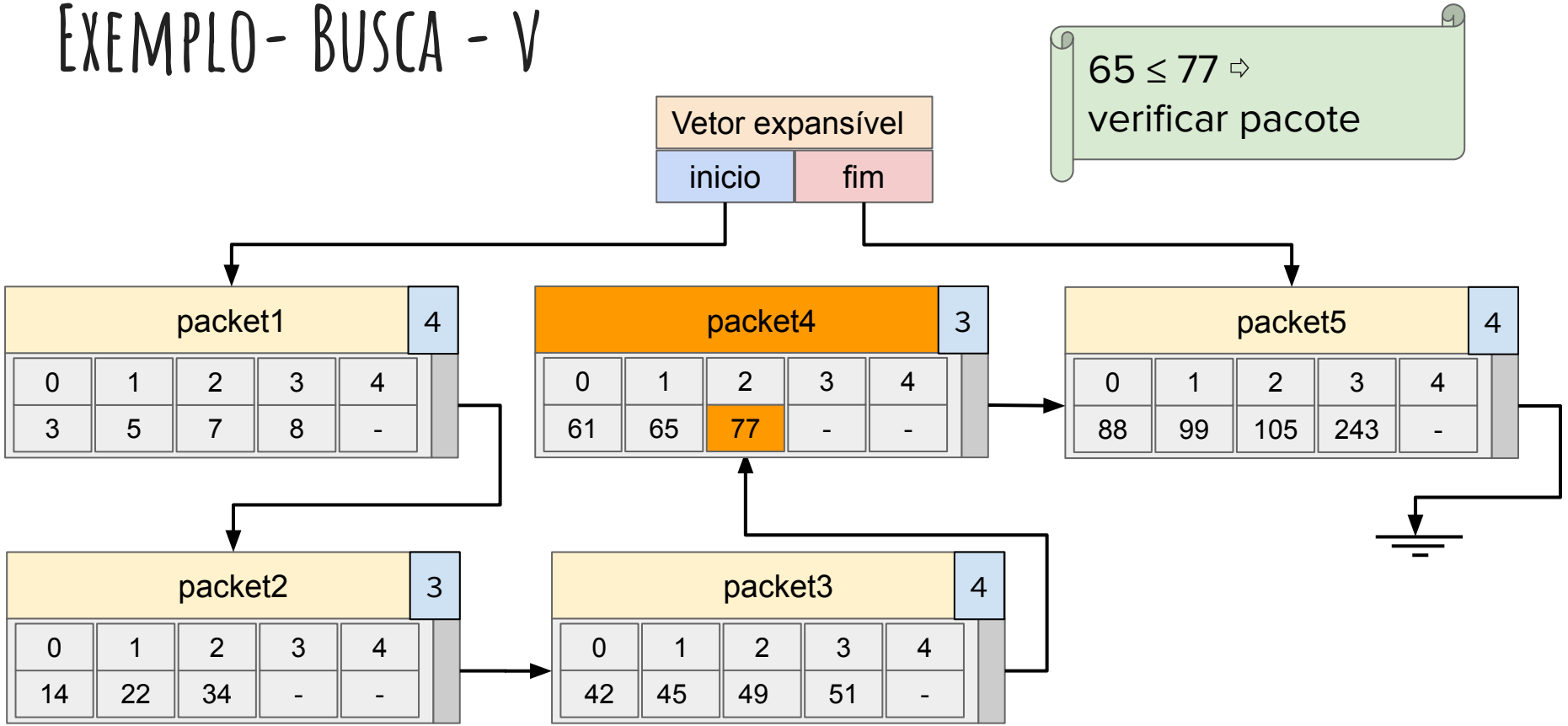


Exemplo- Busca - iv

65 > 51 ⇨
próximo pacote

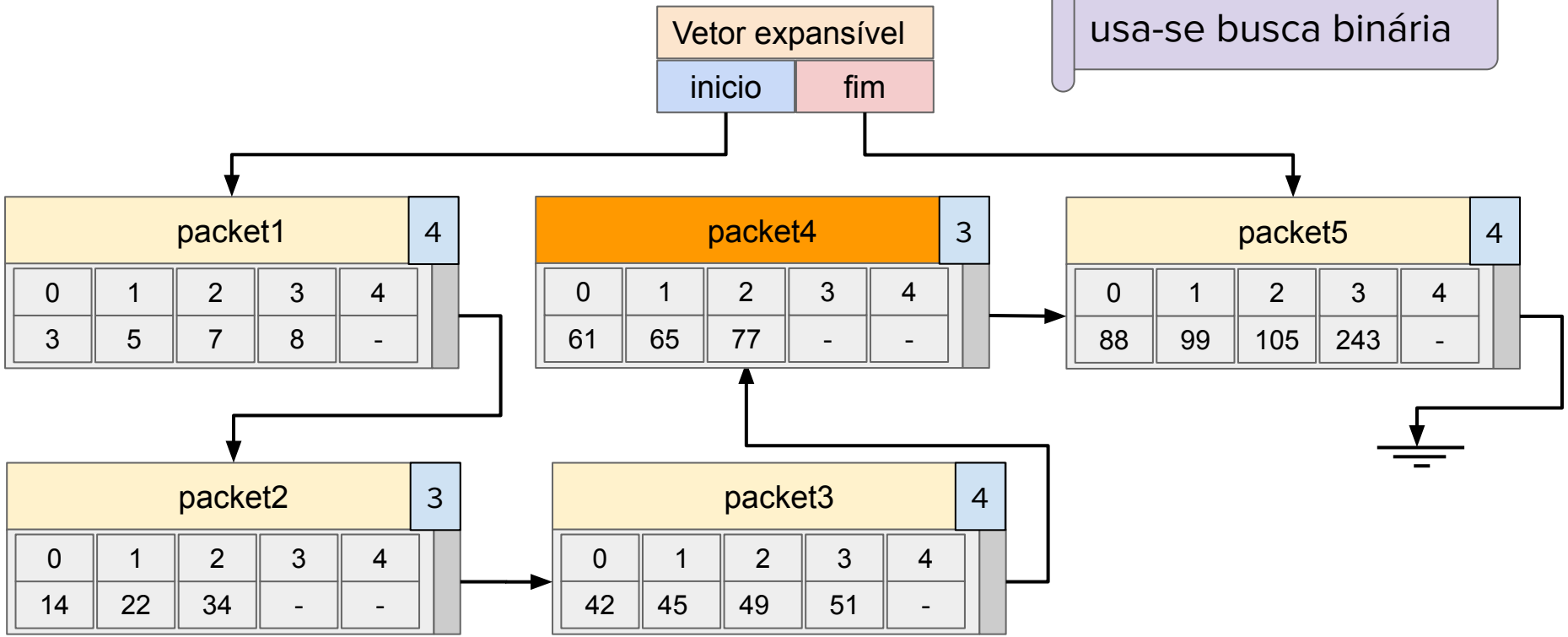


Exemplo- Busca - v

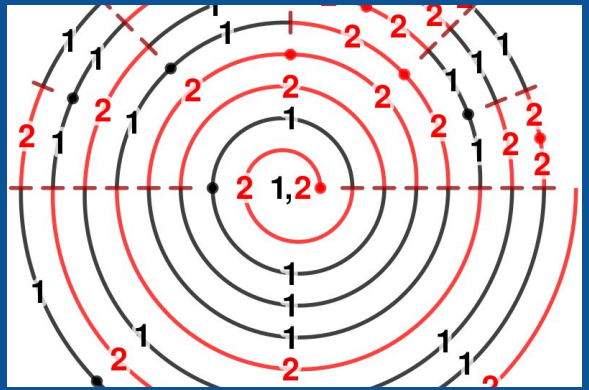


Exemplo- Busca - VI

Em cada pacote,
usa-se busca binária



Sequence sets



Sequence Sets - O Problema

Trabalhar com arquivos binários estruturados geralmente é uma tarefa complexa, quando são necessárias várias modificações (inserções, remoções, alterações, etc.).

Por exemplo, imagine um arquivo com milhares de dados de clientes, onde cada registro ocupa aproximadamente 100KB de dados. Esse arquivo terá obviamente mais de 100MB em seu tamanho.

Sequence Sets - O Problema

Imagine agora a tarefa de percorrer todo esse arquivo para realizar as seguintes ações:

-> Buscar um registro com um dado valor (busca sequencial).
-> Ordenar os registros;
-> Remover um registro no início do arquivo.

Sequence Sets - Conceito

Uma abordagem alternativa e que funciona muito bem em arquivos de tamanhos intermediários é utilizar sequence sets.

Um Sequence Set é um arquivo binário que encontra-se dividido em pedaços de tamanhos iguais, cada uma contendo uma sequência de registros ordenados.

Sequence Sets - Conceito

Cada pedaço do Sequence Set está ordenado em relação aos outros pedaços, assim, os registros de um Sequence Set A possuem chaves que são sempre maiores ou menores que um Sequence Set B. Além disso, cada pedaço indica o próximo pedaço, aquele com os registros com chaves na sequência.

Um cabeçalho de um Sequence Set costuma possuir as seguintes informações: número de pedaços que ele possui e qual a posição relativa do primeiro pedaço.

Sequence Sets - Conceito

????

l ???? n
u e

l ?? Se
c en

c ???

de um Sec ????
Set B. Alé
sequência ??

Um cabeç ???
que ele po

PARA TUDO!

????

e tamanho interme ??

edaços de taman ???

pedacos. assim. os registros
m Sequence
n c ???? n chaves na

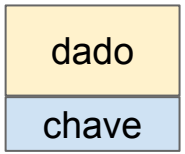
??

or ro de pedaços

???

Calma que eu explico - I

Imagine que você tem um arquivo binário ordenado com 12 registros. Cada registro é composto por um conjunto de dados e uma chave.



Considere um arquivo binário ordenado com 12 registros, com as seguintes chaves:

dado	dado	dado	dado	dado	dado	dado	dado	dado	dado	dado	dado
9	21	32	45	82	83	86	91	96	101	123	204

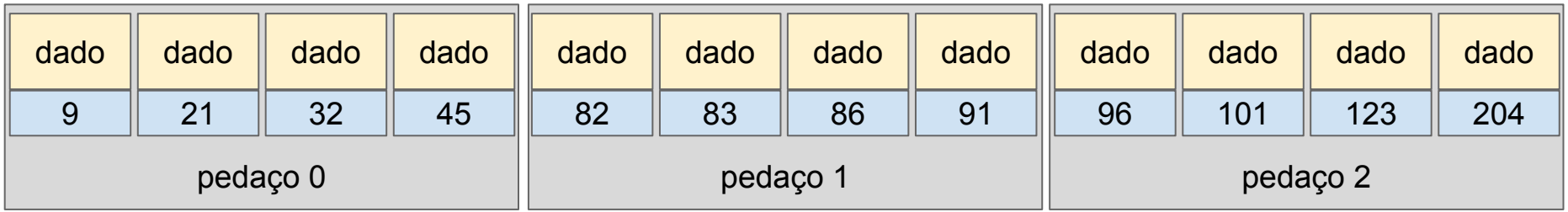
Calma que eu explico - II

dado	dado	dado	dado	dado	dado	dado	dado	dado	dado	dado	dado
9	21	32	45	82	83	86	91	96	101	123	204

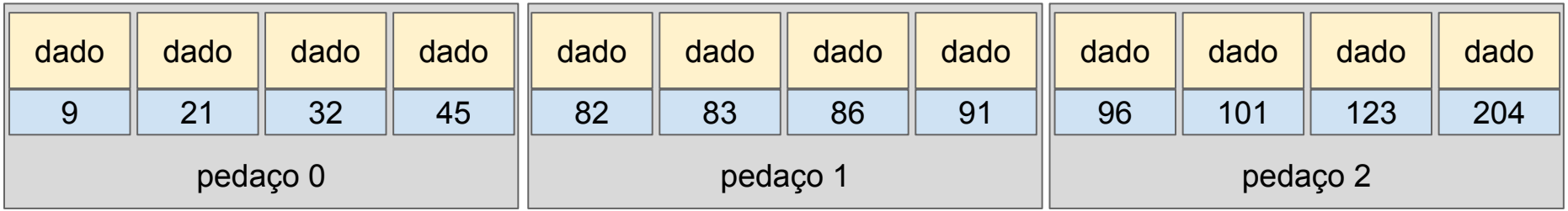
A remoção do registro com chave 83 exigiria o deslocamento dos registros posteriores. A inserção de um elemento com chave 33 exigiria também que registros fossem deslocados.

Calma que eu explico - III

Para diminuir as movimentações necessárias, uma das abordagens é dividir o arquivo em pedaços menores. Em nosso caso, vamos trabalhar com pedaços de tamanho 4:



Calma que eu explico - IV

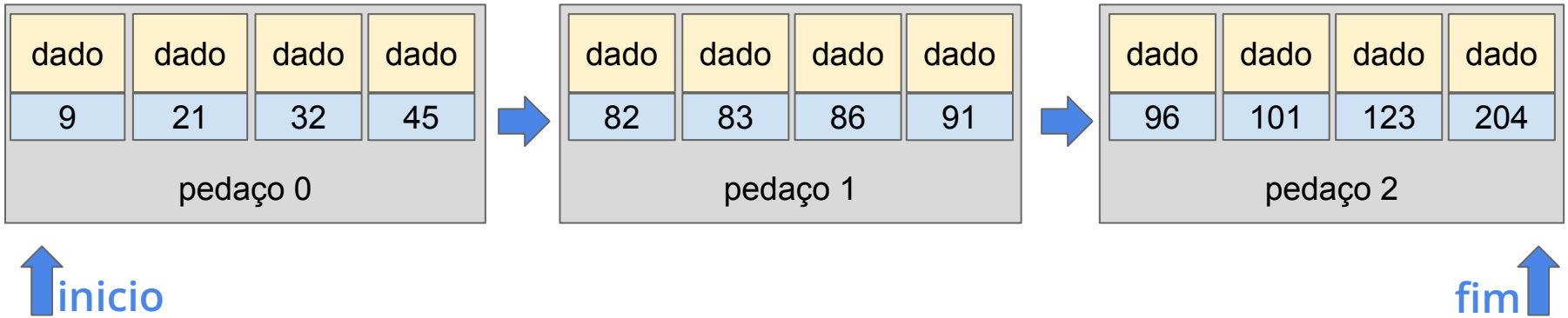


Cada pedaço possui os dados ordenados. Nesse exemplo acima, os pedaços estão ordenados entre si. Para a implementação ficar completa, falta apenas um pedaço apontar o próximo e informar o número de registros válidos.

Isso é mesmo necessário???

Calma que eu explico - V

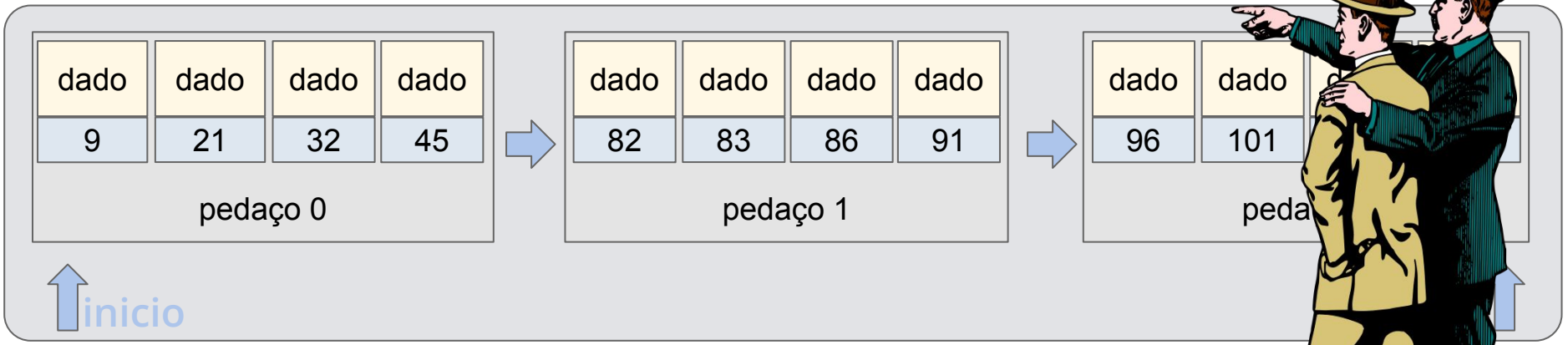
Usaremos então uma indicação para o primeiro pedaço e cada pedaço apontará o próximo:



Isso é feito por meio de dois mecanismos: um cabeçalho no arquivo e um cabeçalho em cada pedaço.

Calma que eu explico - V

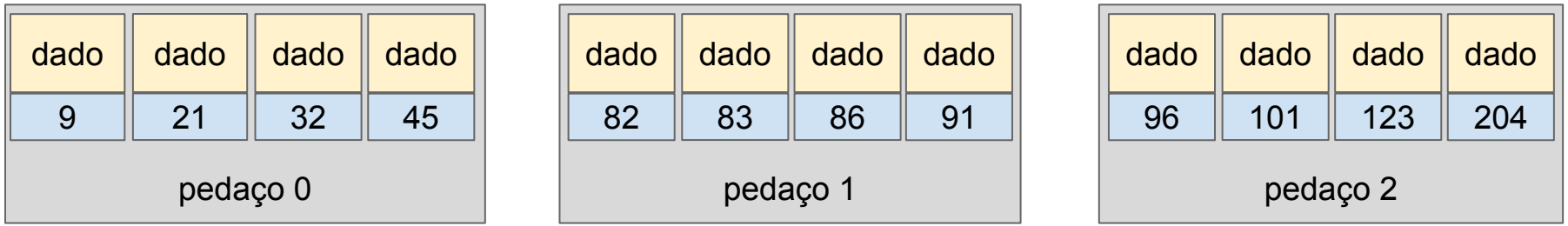
Usaremos então uma indic ço e cada pedaço apontará o próximo:



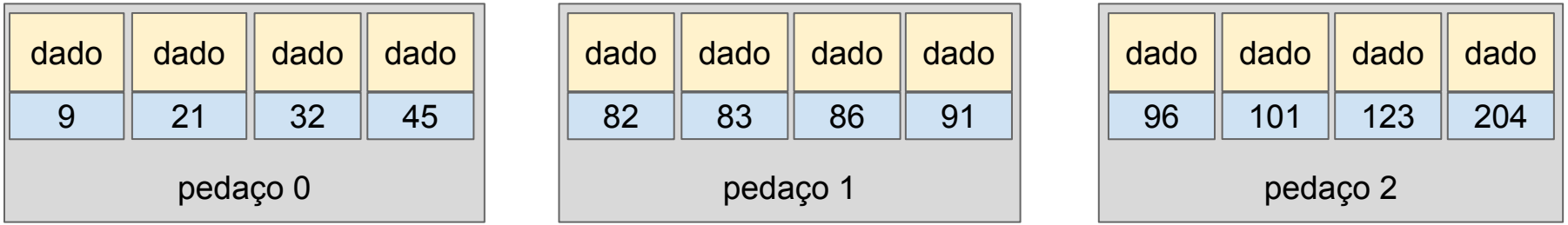
Isso é feito por meio de dois mecanismos: um cabe l no arquivo e um cabeçalho em cada pedaço.

Calma que eu explico - VI

Cada pedaço possui uma posição relativa dentro do arquivo, como já visto antes:



Calma que eu explico - VII



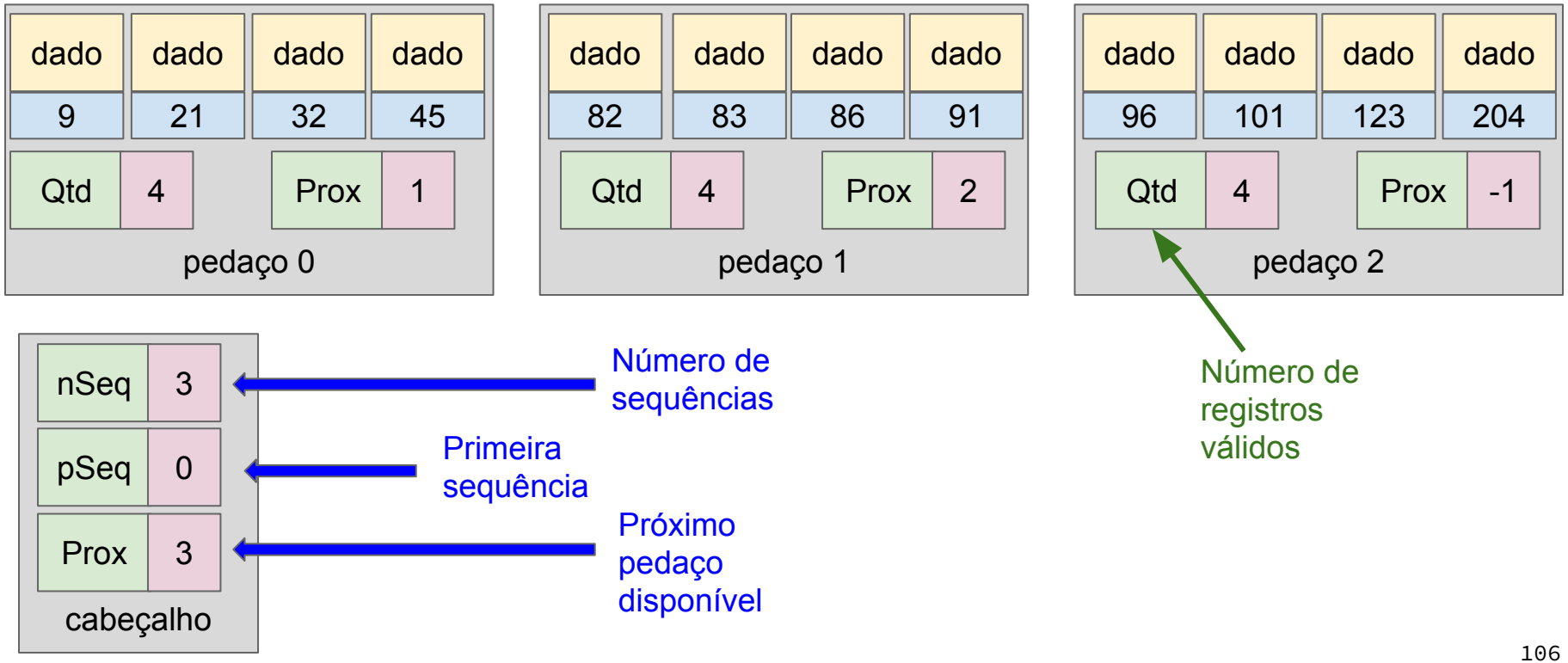
Assim, no cabeçalho do arquivo binário, teremos as seguintes informações:

- ❏ Número de pedaços (sequence): 3
- ❏ Posição relativa do primeiro pedaço: 0
- ❏ Próximo pedaço a ser utilizado: 3

Calma que eu explico - VIII

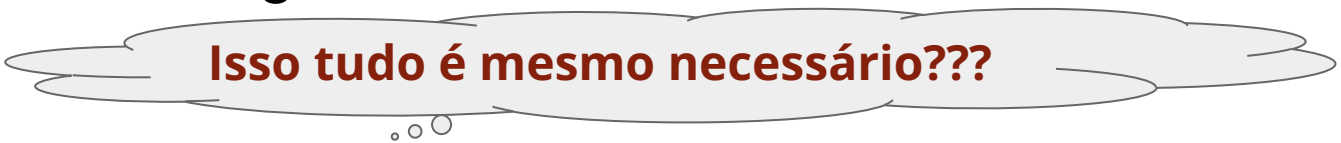
Cada pedaço, por sua vez, terá a posição relativa da próxima sequência. O último pedaço usará -1 para indicar que não há mais sequências. Segue-se a representação do Sequence Set do exemplo:

Calma que eu explico - VIII



Peraí, peraí, ...

Cada pedaço possui os dados ordenados. O cabeçalho do arquivo indica o primeiro pedaço e qual a posição do primeiro pedaço disponível. Em cada pedaço, um cabeçalho local aponta a próxima sequência e informa o número de registros válidos.



Essas informações são necessárias para manter o Sequence Set otimizado após alterações (inserções e remoções).

Pode desenhar???

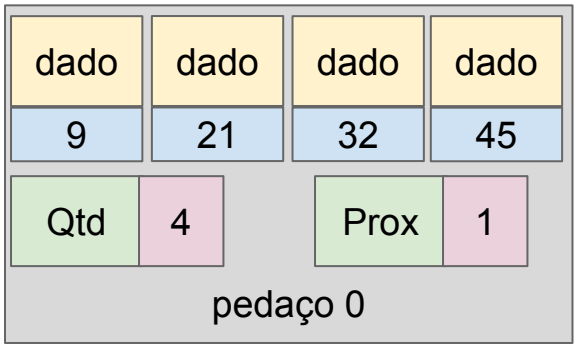
Imagine que serão realizadas as seguintes operações no Sequence Set de exemplo:

- Inserção dos valores 13, 85, 81, 80 e 3, nessa ordem;
- Remoção dos valores 21, 13, 9 e 3, nessa ordem;
- Inserção dos valores 5 e 6, nessa ordem.

Operações com Sequence Sets - I

1) Inserção do 13:

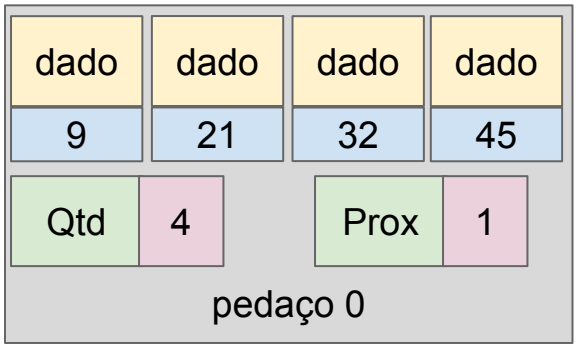
Obviamente o 13 será inserido no pedaço (sequência) 0, que está **cheio**...



Operações com Sequence Sets - 1

1) Inserção do 13:

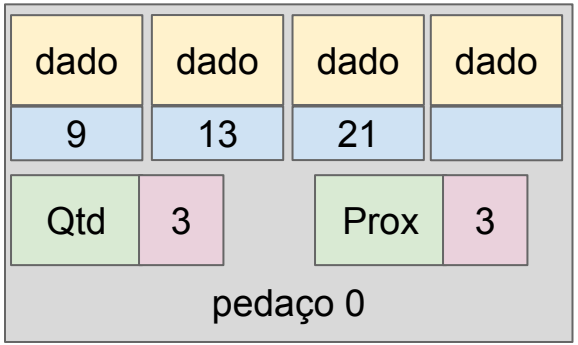
Obviamente o 13 será inserido no pedaço (sequência) 0, que está **cheio**...

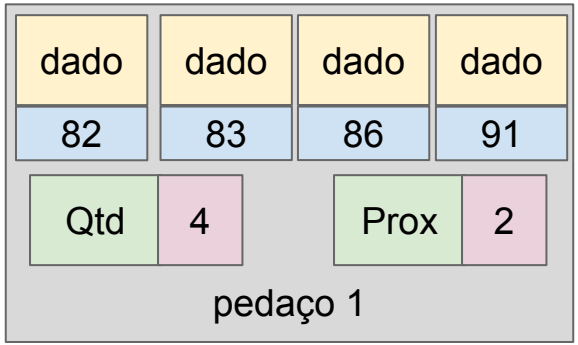


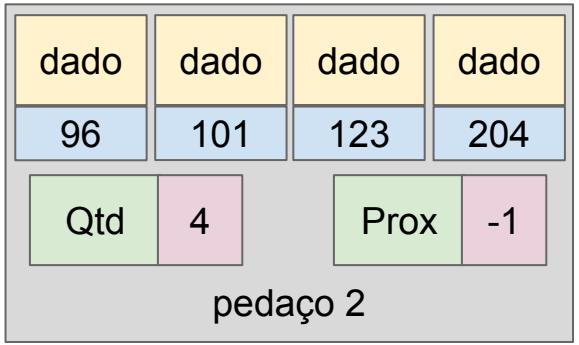
Operações com Sequence Sets - II

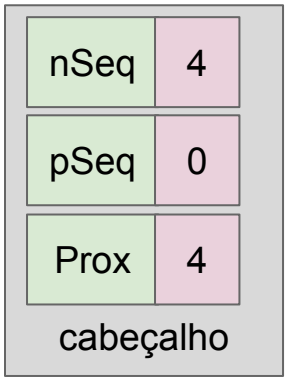
A solução é criar um novo pedaço (no final do arquivo) inicialmente vazio para dividir o Sequence Set cheio. A divisão é feita com a inserção em memória e dividindo a sequência na metade.

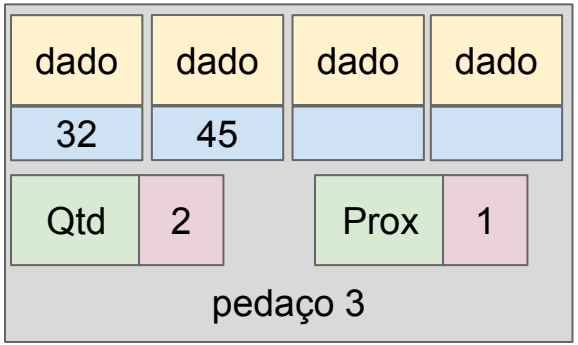
Operações com Sequence Sets - II











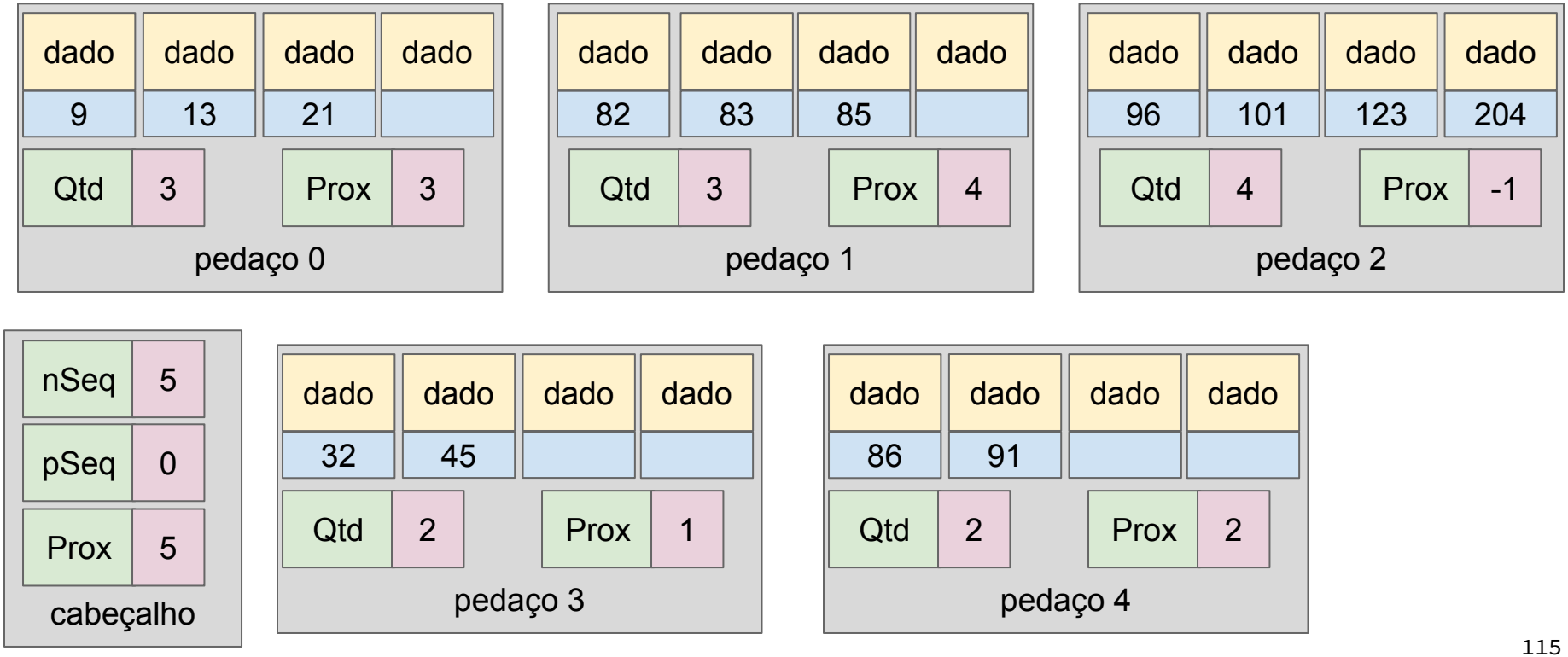
Operações com Sequence Sets - II

Ainda há alterações no arquivo, mas o número de operações diminuiu bastante!

Operações com Sequence Sets - III

A inserção do 85 irá produzir divisão similar:

Operações com Sequence Sets - III

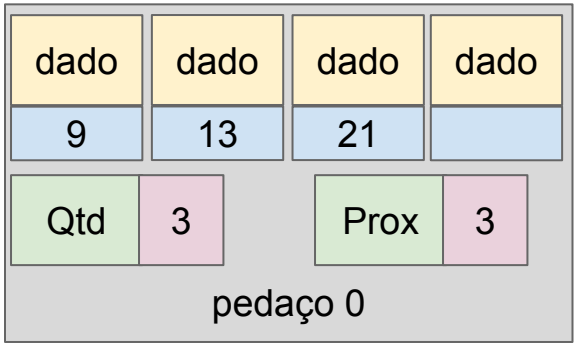


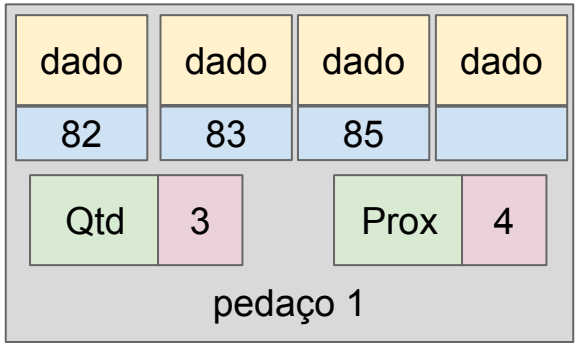
Operações com Sequence Sets - iv

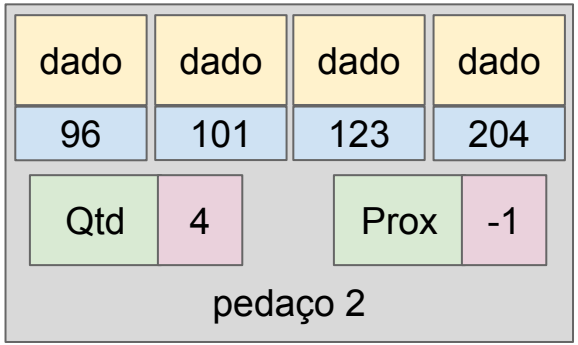
A inserção do 81 e do 80 serão relativamente tranquilas:

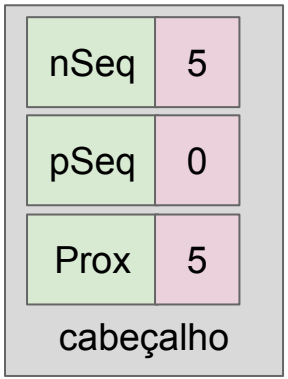
Quase sem alterações!!

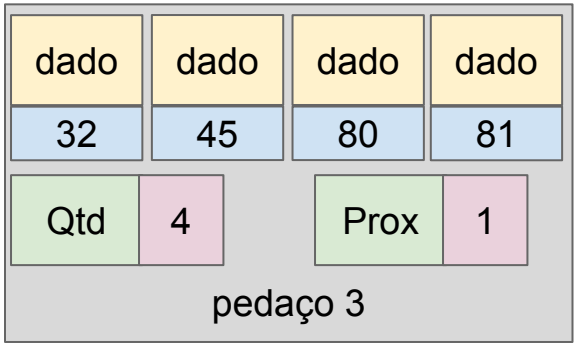
Operações com Sequence Sets - iv

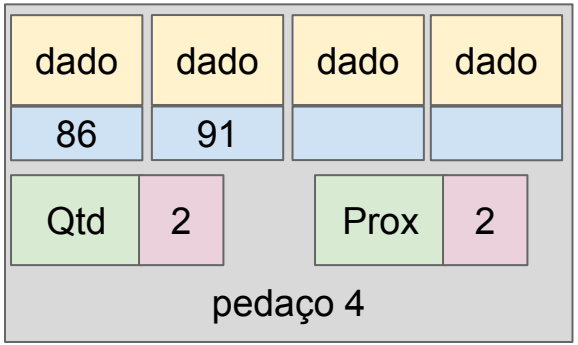










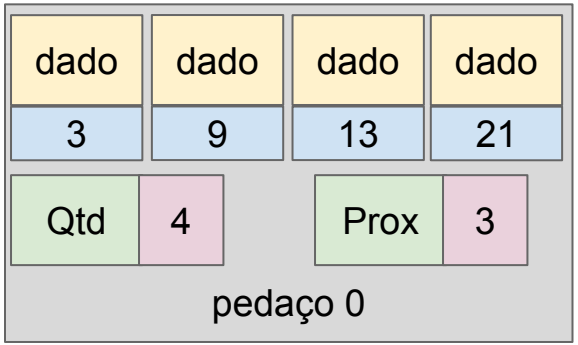


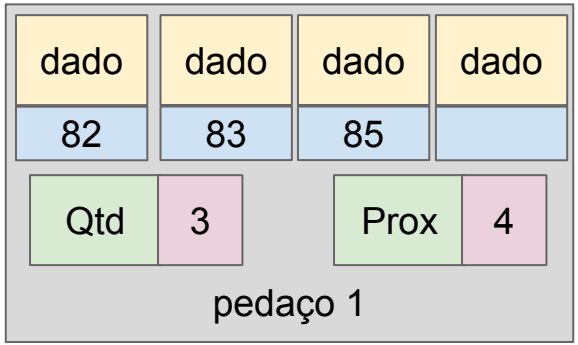
Operações com Sequence Sets - V

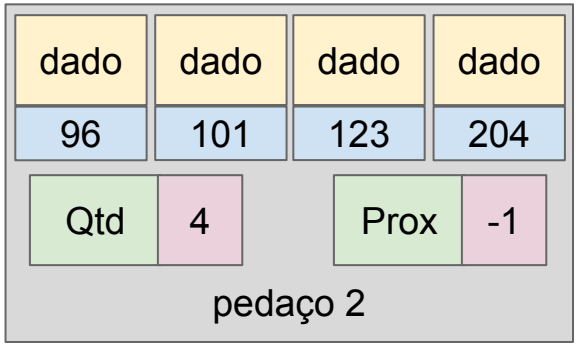
A inserção do 3 só gera impacto na própria sequência:

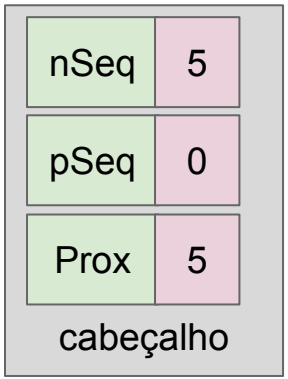
Ainda bom para o arquivo como um todo!

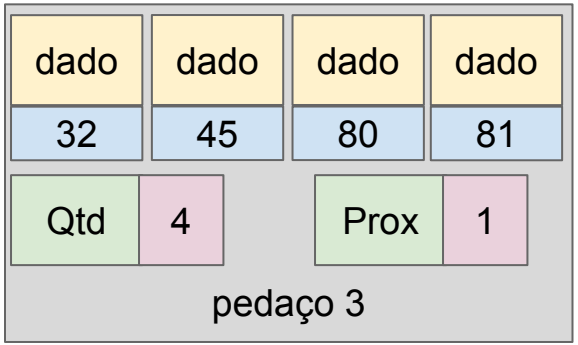
Operações com Sequence Sets - V

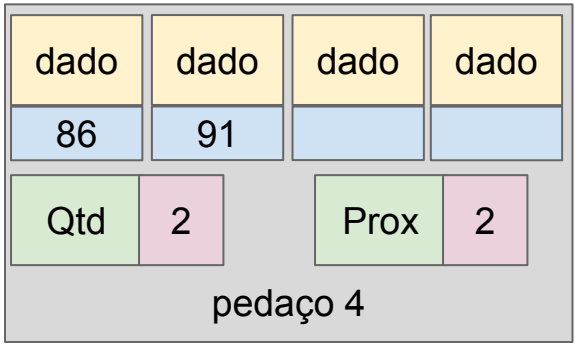








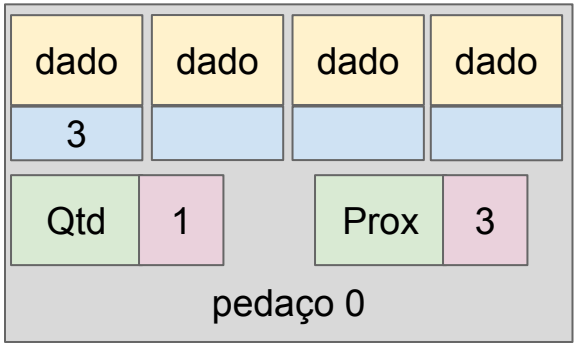


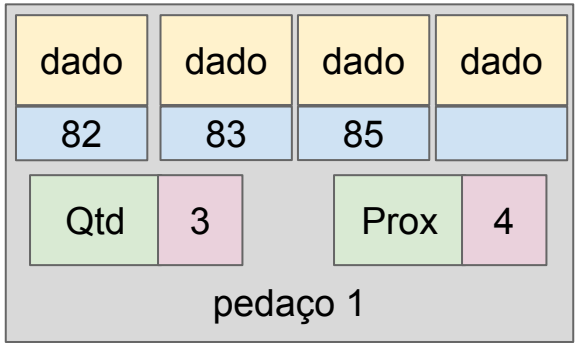


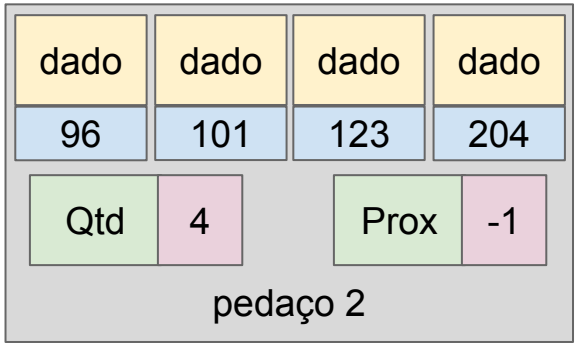
Operações com Sequence Sets - VI

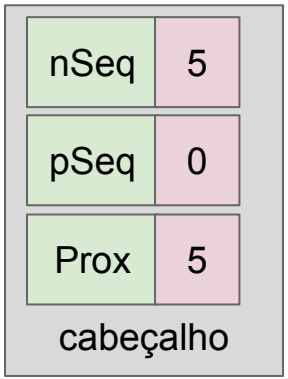
A remoção dos registros com chaves 21, 13, 9 é muito tranquila:

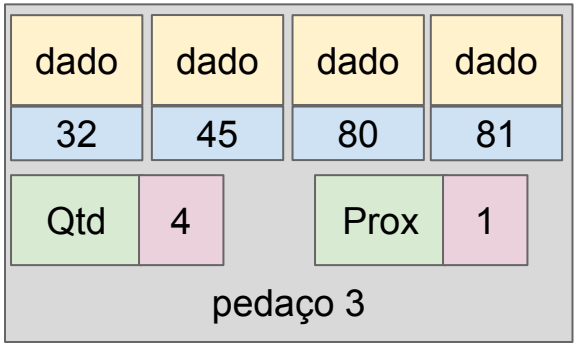
Operações com Sequence Sets - VI

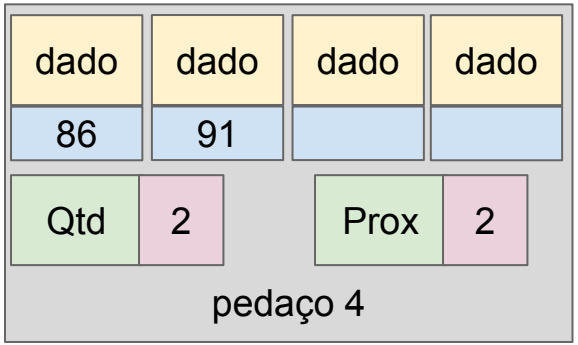












Operações com Sequence Sets - VII

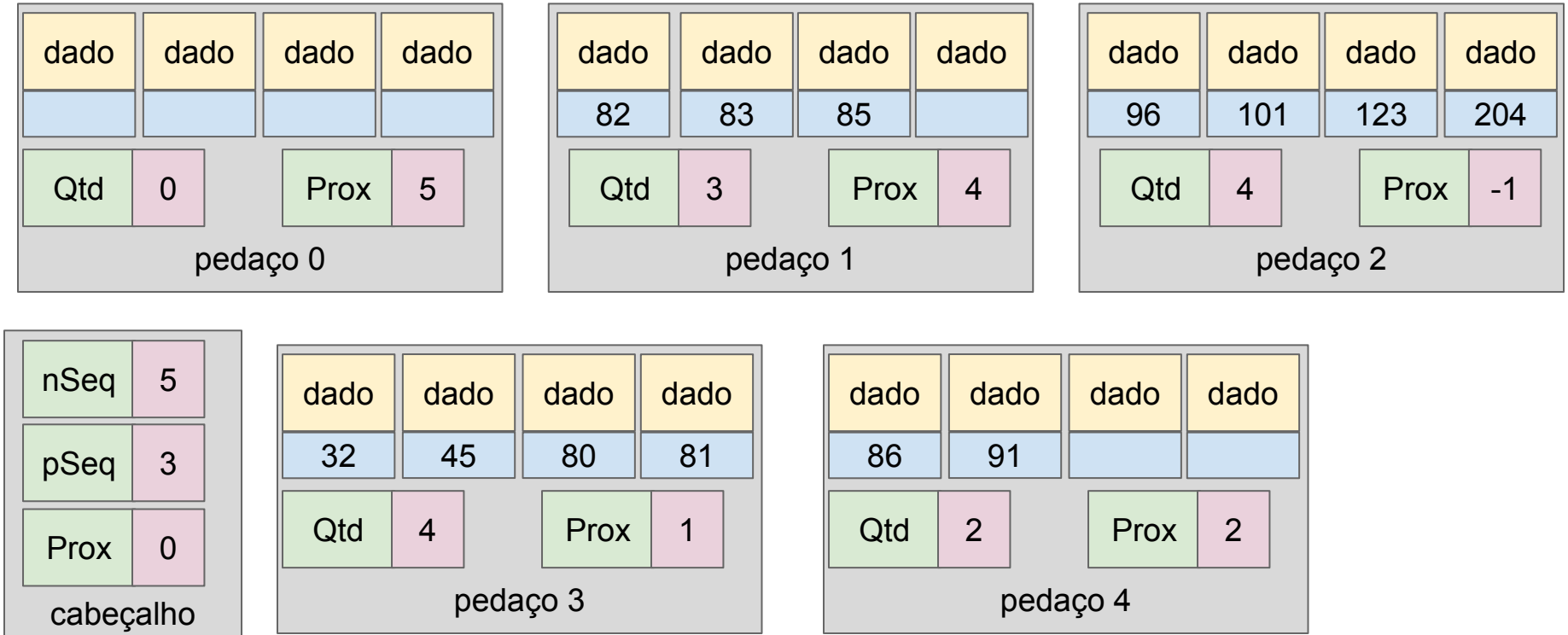
E como remover o 3???

Operações com Sequence Sets - VII

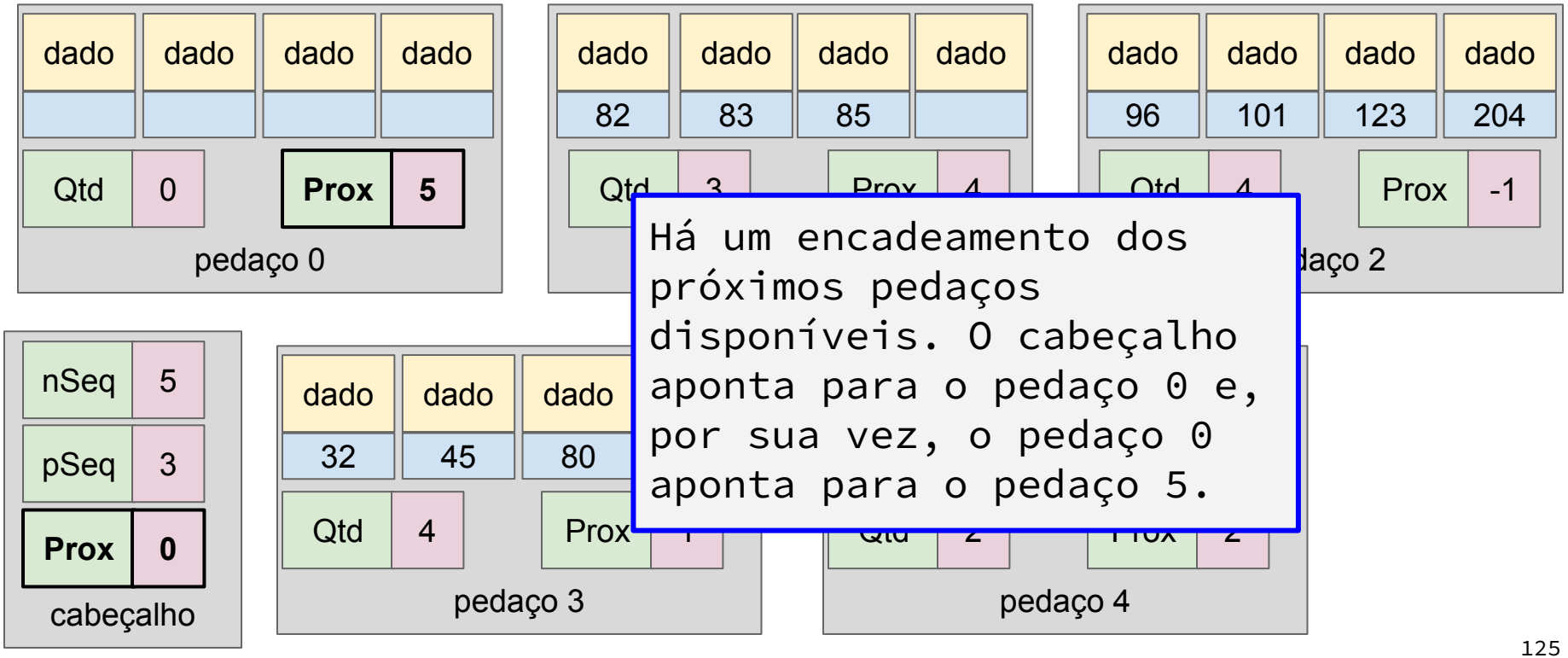
Quando se remove o último elemento de uma sequência em um Sequence Set, o pedaço não é removido! Ele é mantido inutilizado, à espera de uma próxima divisão!

Assim, isso obriga a mudança no cabeçalho do nosso arquivo de exemplo, para indicar o primeiro pedaço a ser utilizado e qual o primeiro pedaço disponível.

Operações com Sequence Sets - VII



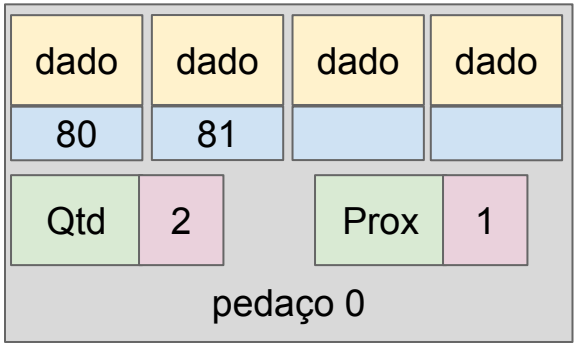
Operações com Sequence Sets - VII

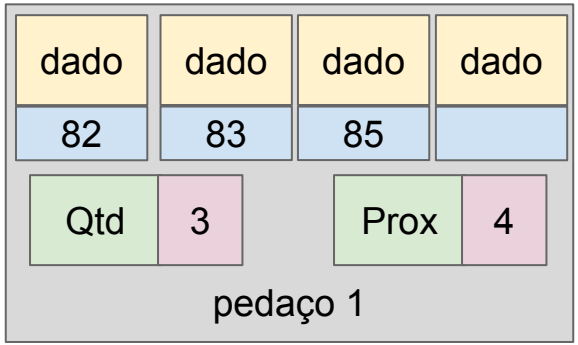


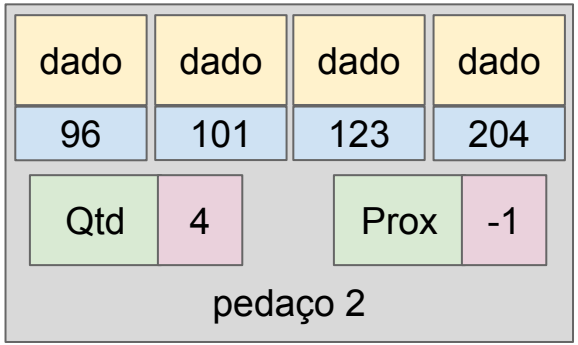
Operações com Sequence Sets - VIII

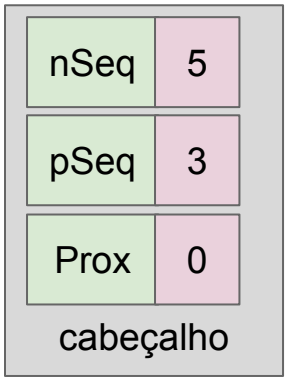
A inserção do 5 irá gerar nova divisão… Só que o pedaço disponível está na posição 0:

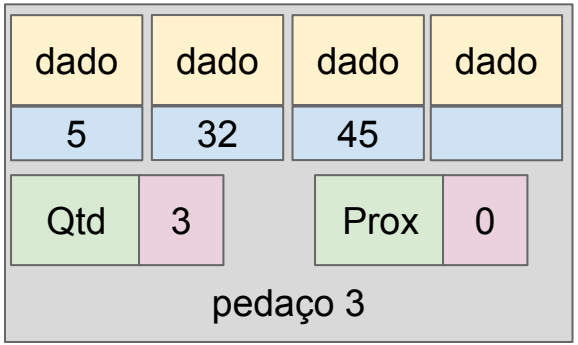
Operações com Sequence Sets - VIII

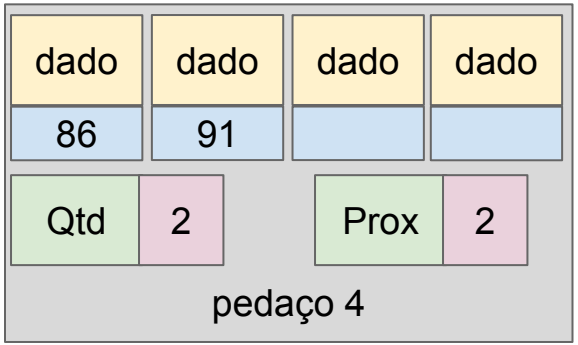








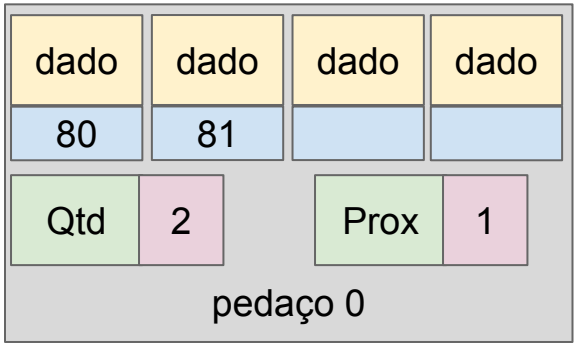


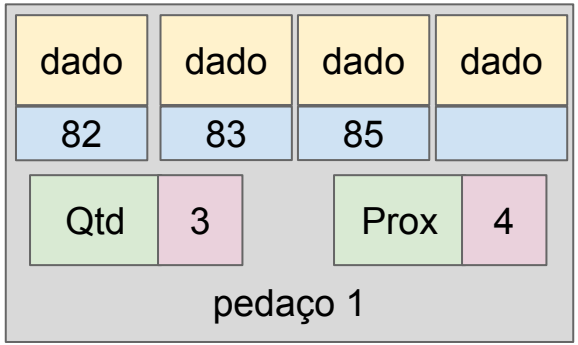


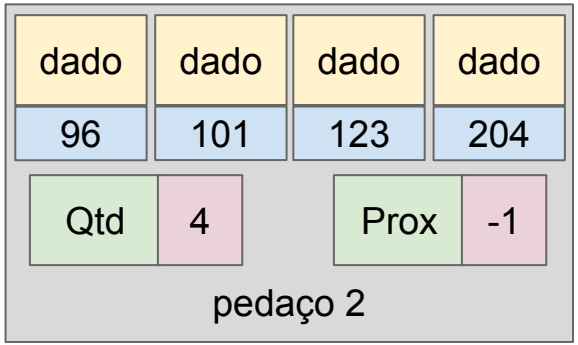
Operações com Sequence Sets - ix

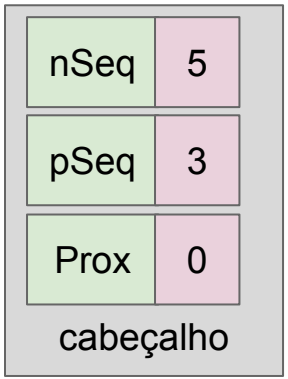
A inserção do 6 será muito tranquila:

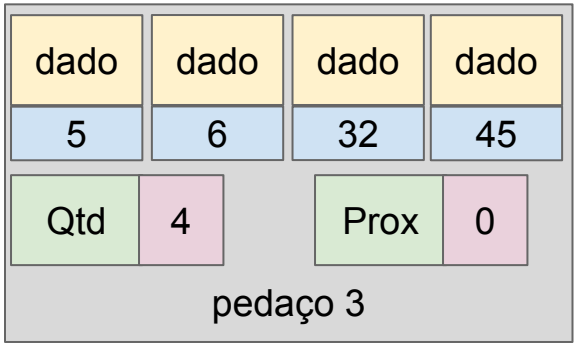
Operações com Sequence Sets - ix

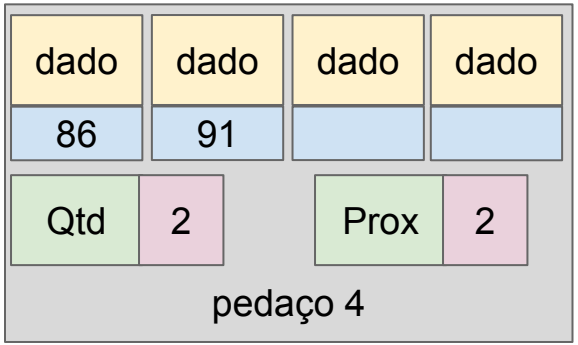












Ainda sobre remoção

Implementações mais eficientes de sequence sets utilizam a ideia de não permitir que cada pedaço possua menos que um número mínimo de elementos (em geral a metade).

Ainda sobre remoção

Nesse caso, quando um pedaço fica com menos que a metade, tenta-se emprestar um elemento do pedaço anterior ou posterior. Caso não seja possível o empréstimo (pelo fato que ambos estão com o número mínimo de elemento), une-se então o pedaço atual com um desses pedaços.

Sequence Sets - Considerações Finais

O uso de Sequence Sets otimiza a manipulação de dados em arquivos binários com vários registros.

Pode ocorrer de, com o passar do tempo, os pedaços ficarem muito subutilizados… No nosso exemplo, o pedaço 0 não irá receber novos valores,... Nesse caso, pode-se reconstruir o Sequence Set, unificando pedaços menores em pedaços maiores.

Sequence Sets - Considerações Finais

Outra abordagem para diminuir a existência de espaços subutilizados é estabelecer um limite mínimo de ocupação de cada pedaço, agrupando dois pedaços subsequentes.

Sequence sets tem seu uso potencializado por meio de índices, como em árvores B+.

Sobre o Material

Sobre este material

Material produzido coletivamente, principalmente pelos seguintes professores do DCC/UFLA:

- Joaquim Quinteiro Uchôa
- Juliana Galvani Greghi
- Renato Ramos da Silva

Inclui contribuições de outros professores do setor de Fundamentos de Programação do DCC/UFLA.