

ESTRUTURAS BASEADAS EM LISTAS

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



- Deques
- Vetores Expansíveis
- Sequence Set

DEQUES - DOUBLE QUEUES



DEQUE - FILA DUPLA (DOUBLE QUEUE)

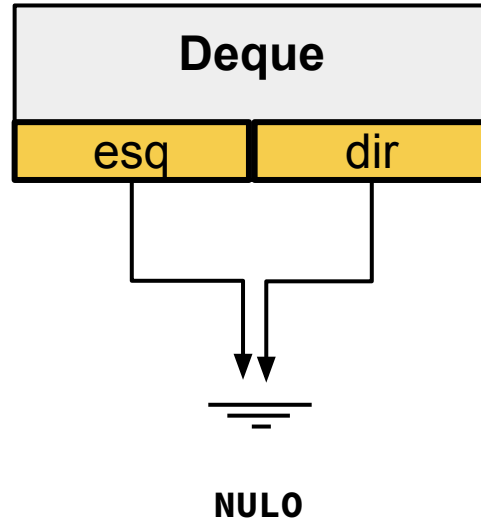
Uma deque é uma estrutura de dados em que a remoção e inserção só é permitida em suas pontas (ou seja: no início e no final da lista).

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

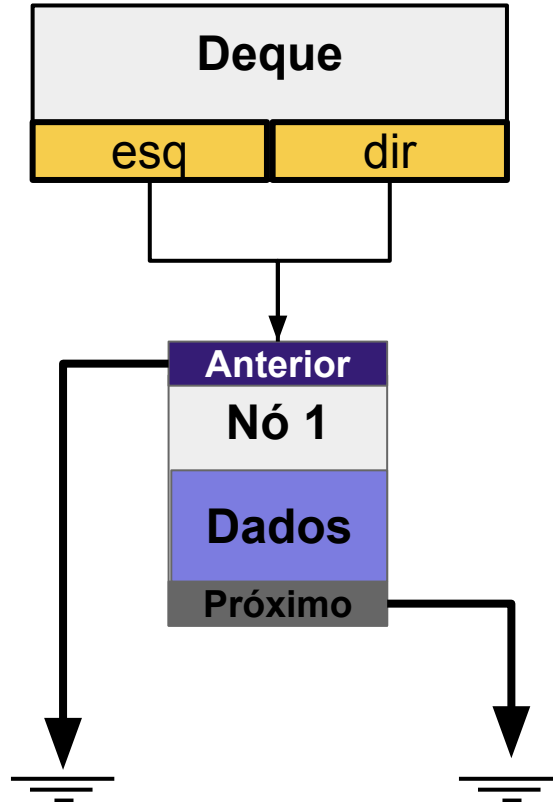
DEQUE - FILA DUPLA (DOUBLE QUEUE)

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

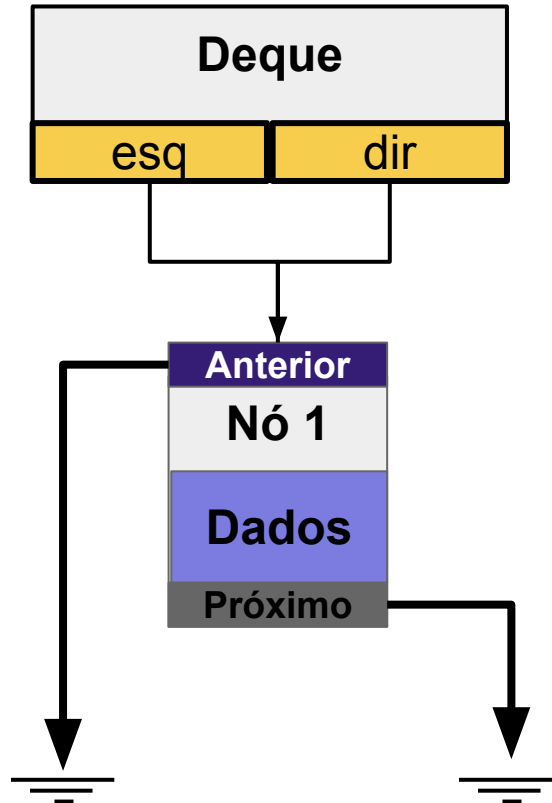
INSERÇÃO EM DEQUE VAZIA - I



INSERÇÃO EM DEQUE VAZIA - II



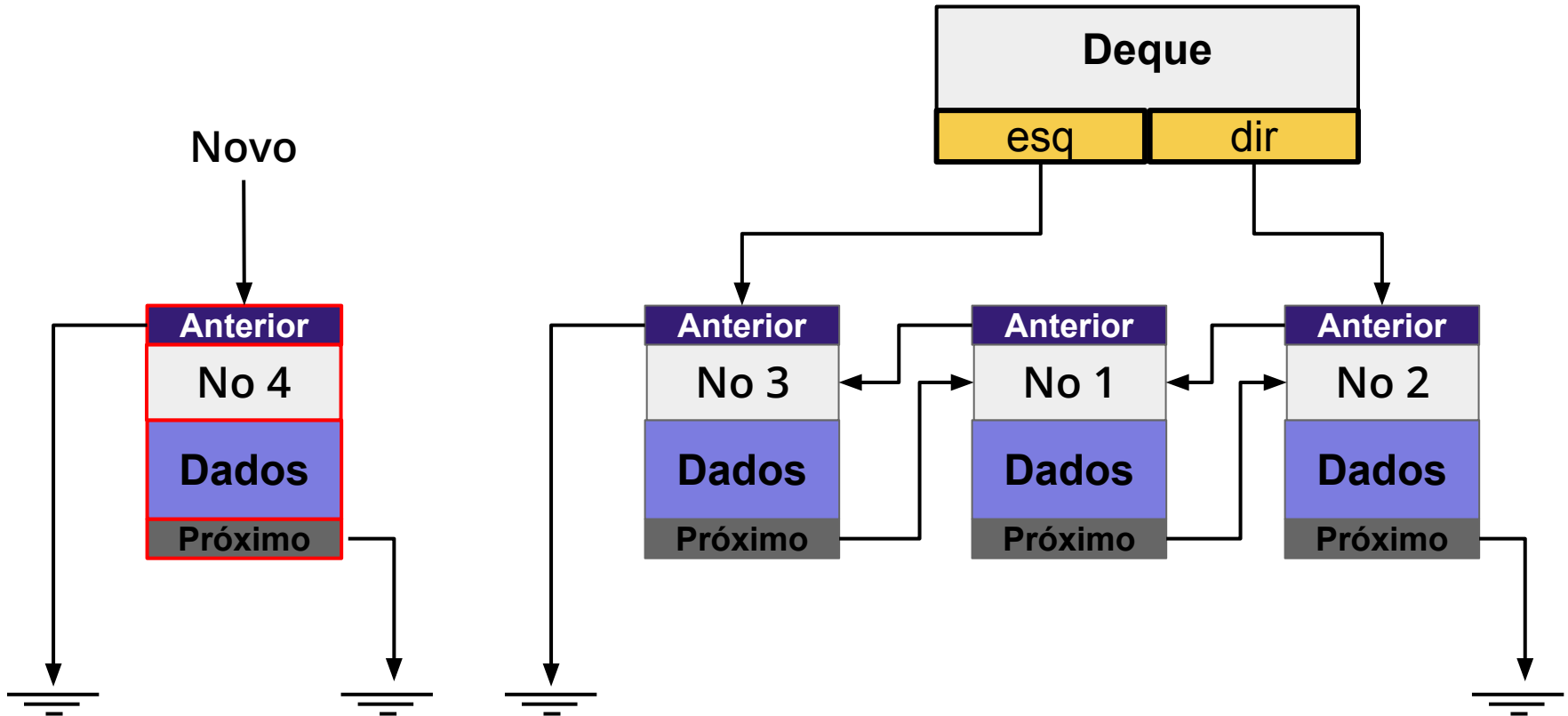
INSERÇÃO EM DEQUE VAZIA - II



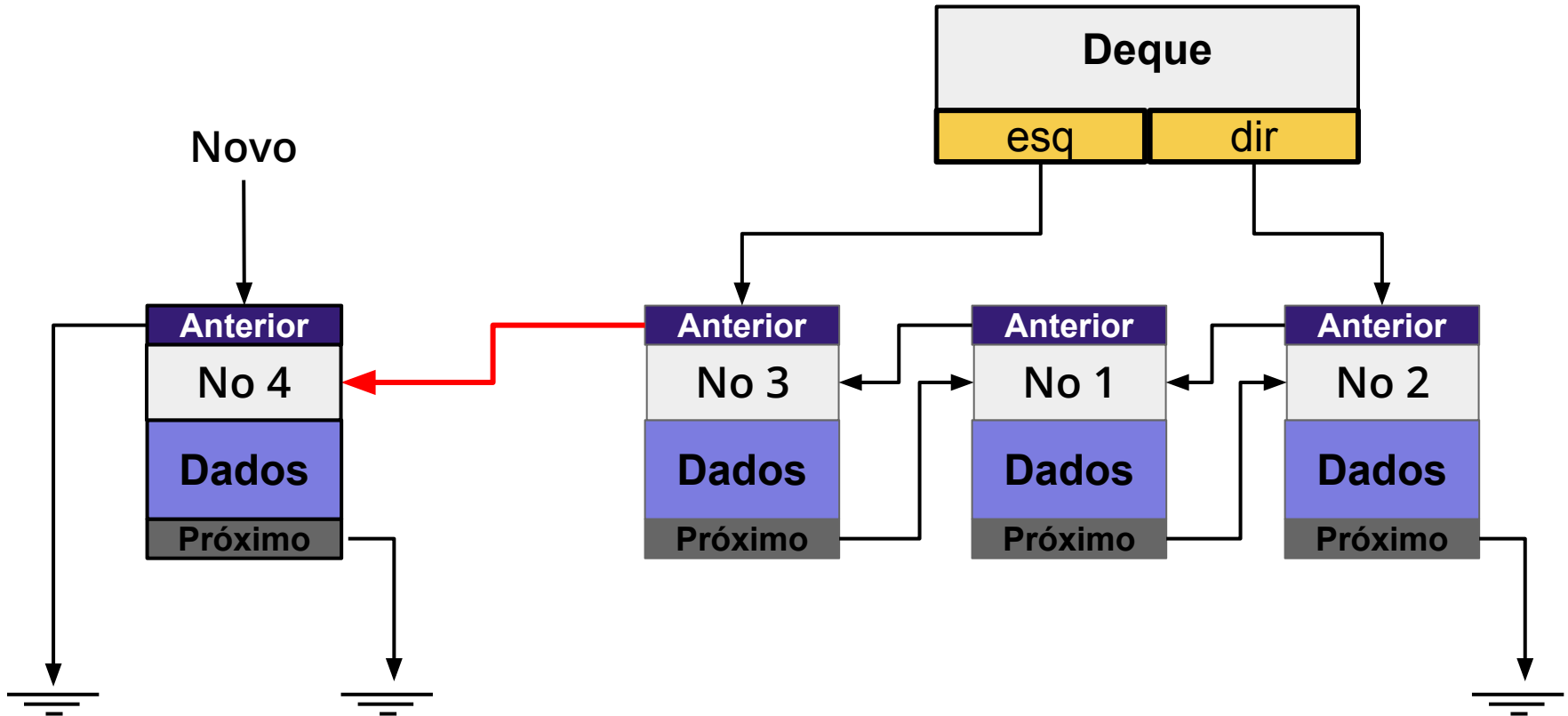
A inserção em deque vazia é utilizada tanto para inserção à esquerda ou à direita, desde que se constate que a estrutura não possui ainda elementos.

Método é implementado como auxiliar, usado se necessário.

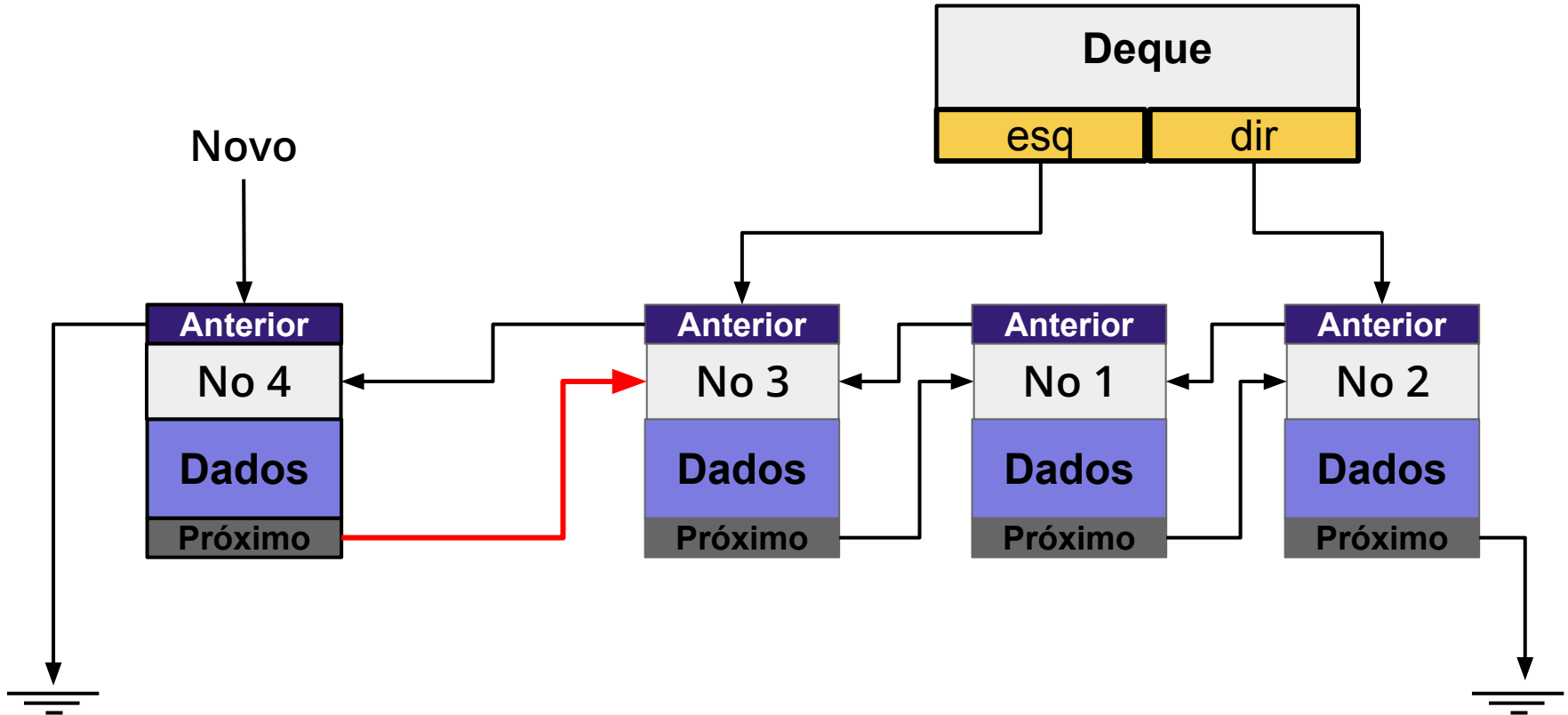
INSERÇÃO À ESQUERDA - I



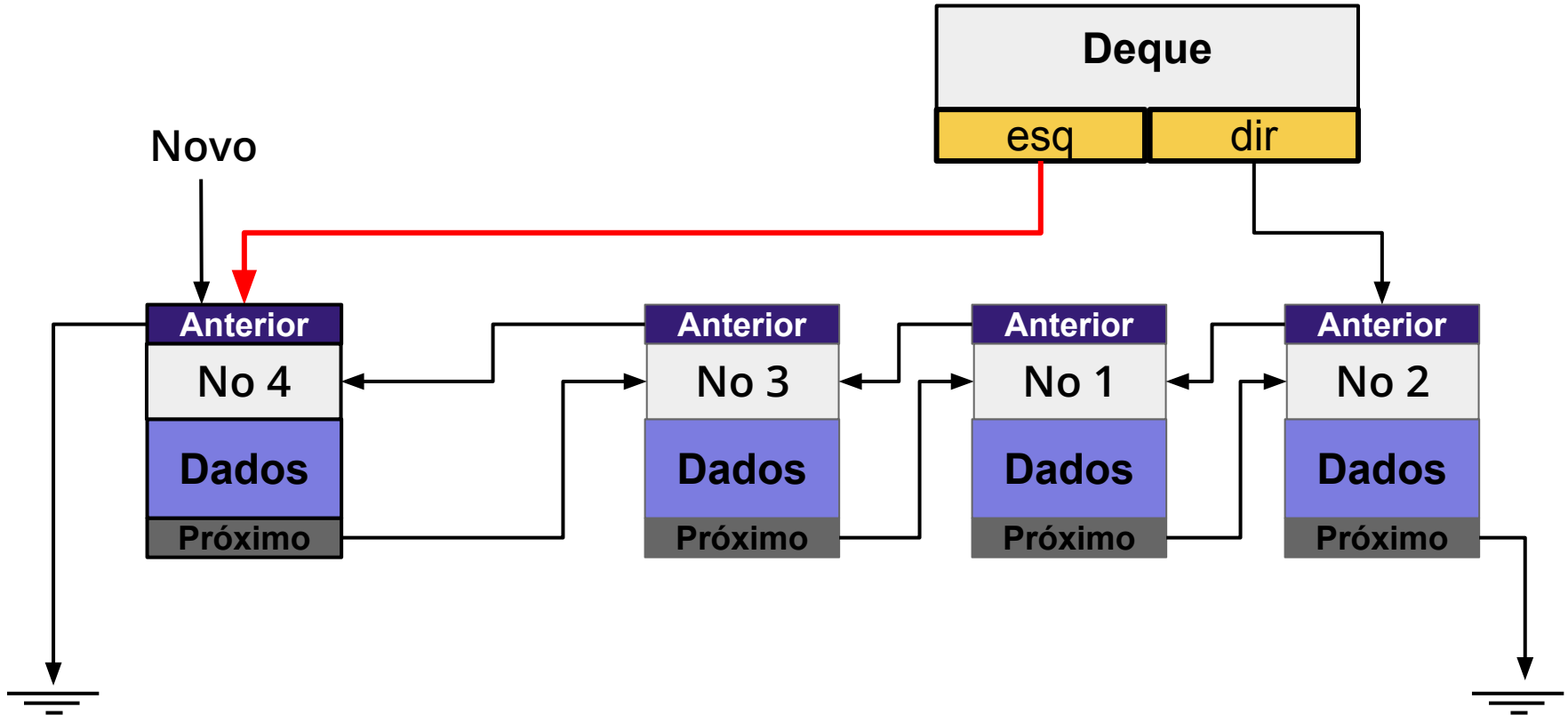
INSERÇÃO À ESQUERDA - II



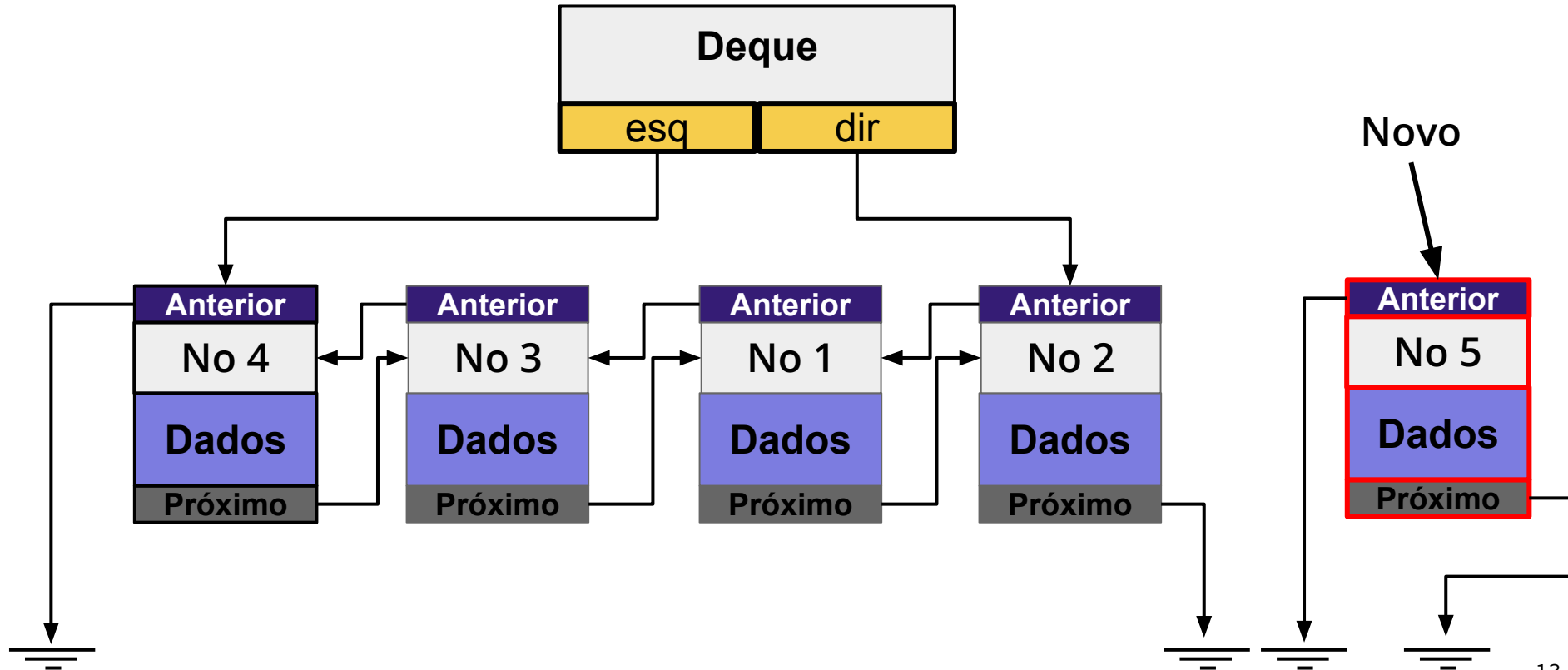
INSERÇÃO À ESQUERDA - III



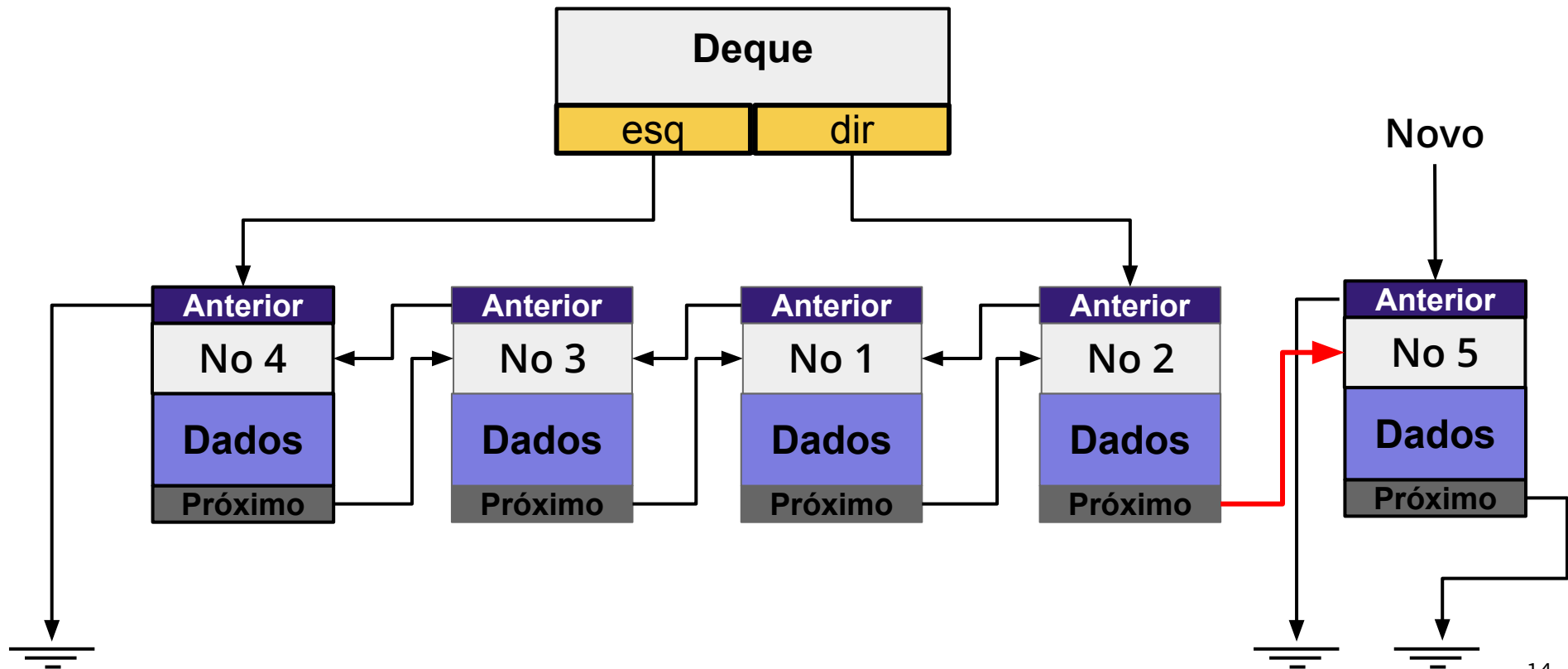
INSERÇÃO À ESQUERDA - IV



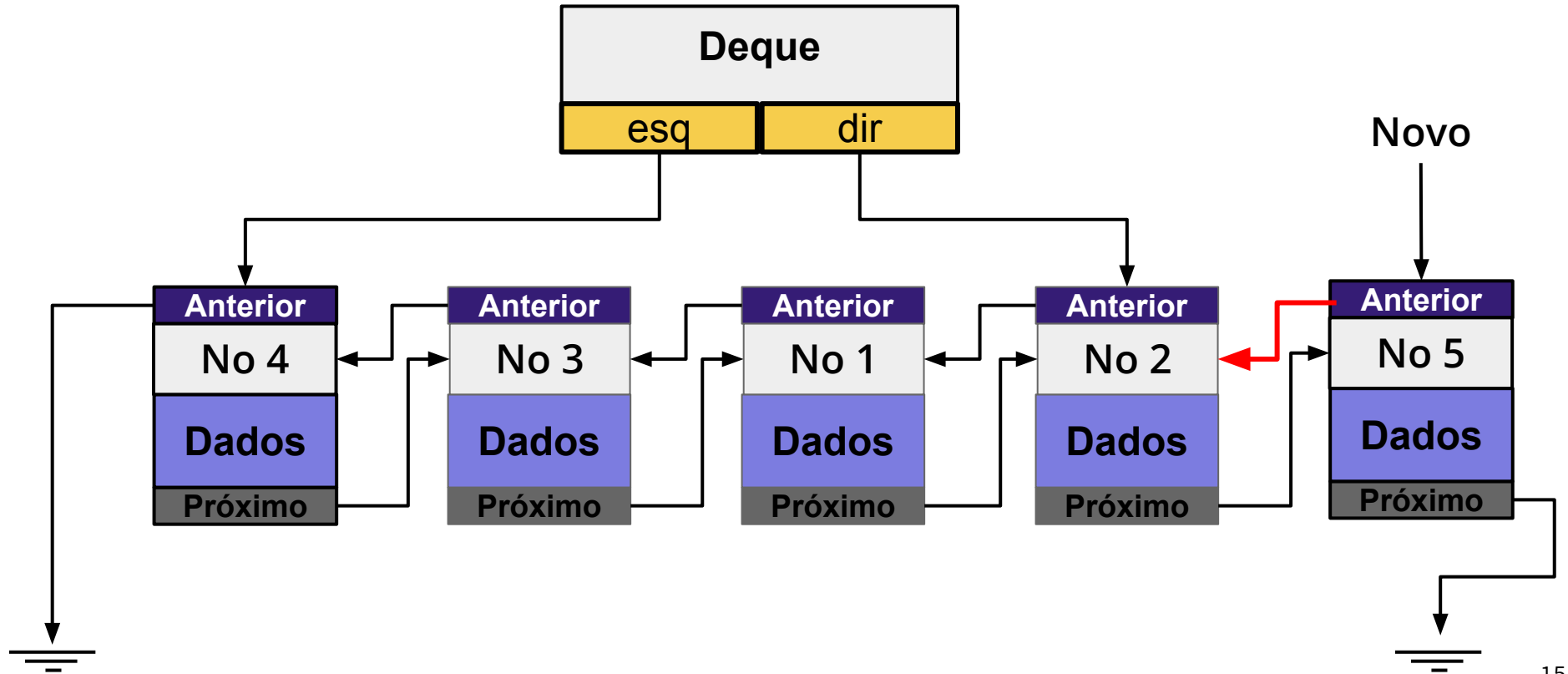
INSERÇÃO À DIREITA - I



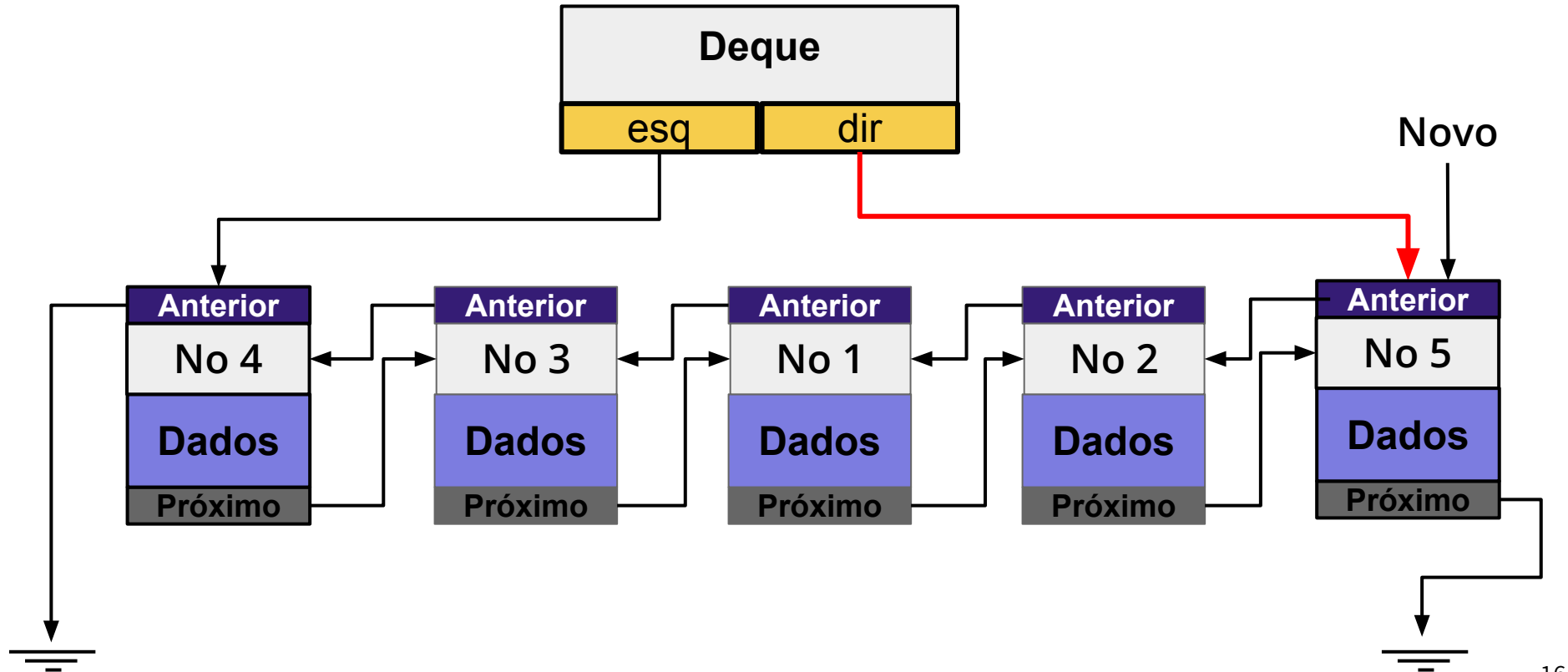
INSERÇÃO À DIREITA - II



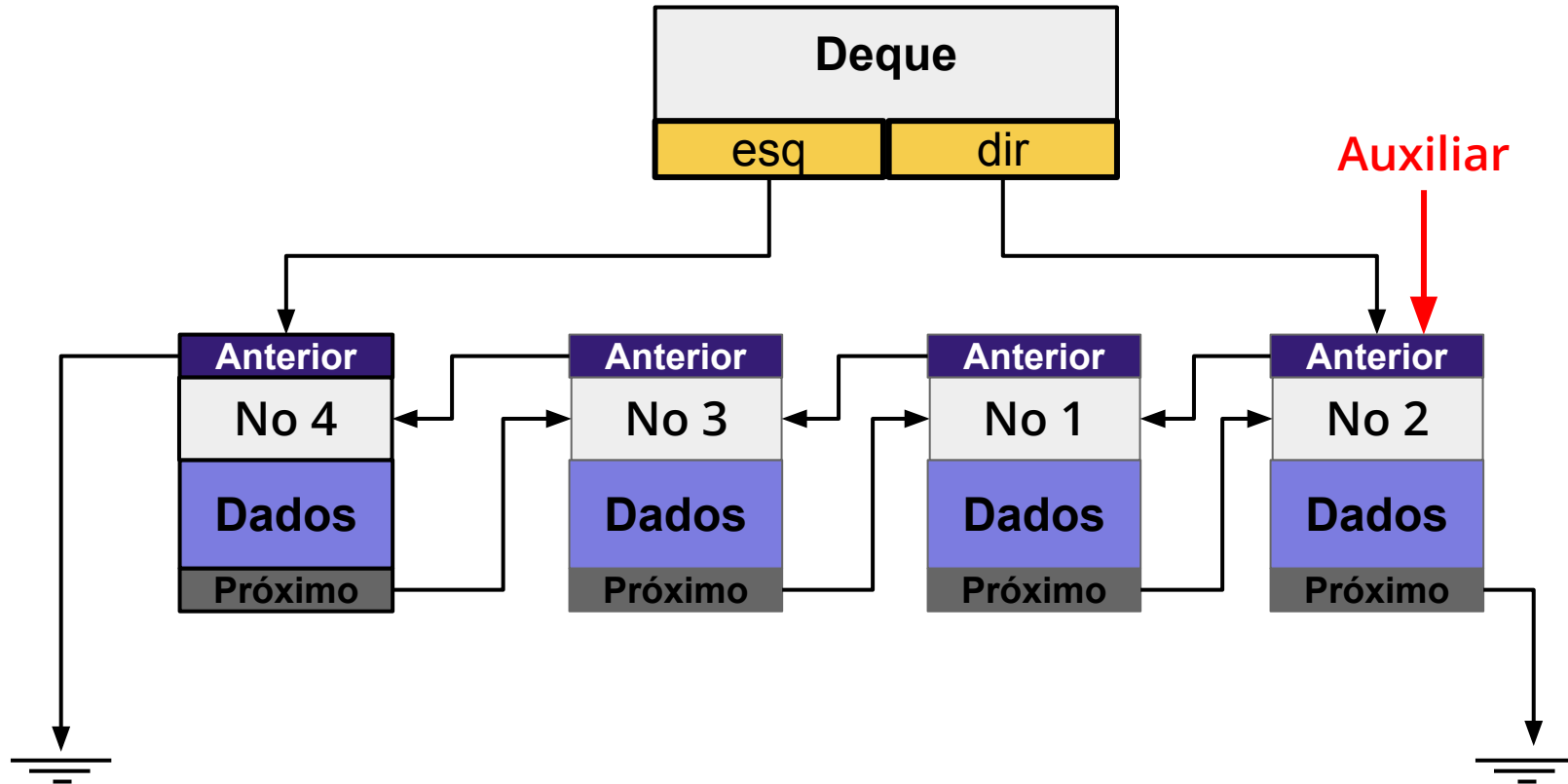
INSERÇÃO À DIREITA - III



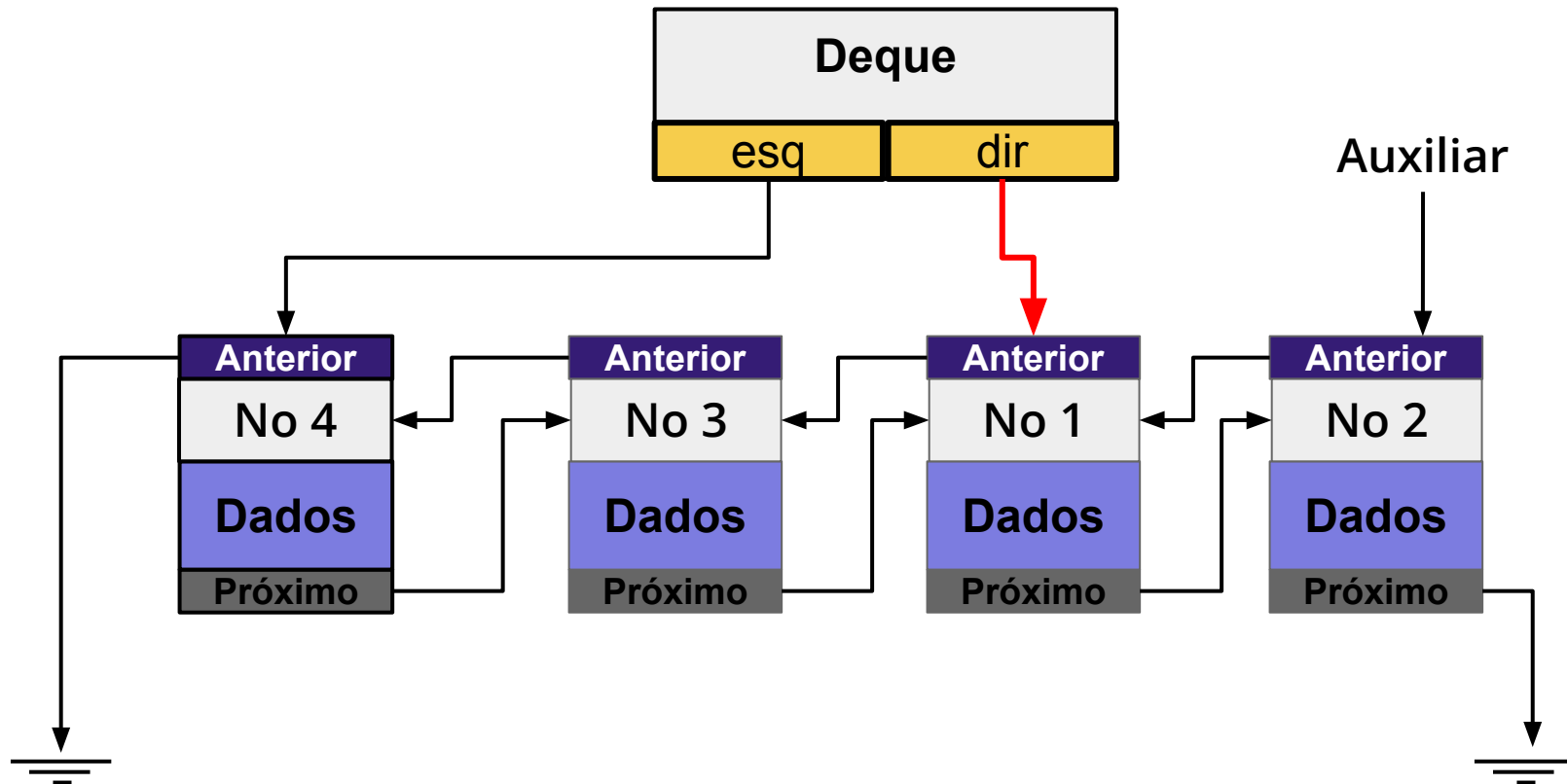
INSERÇÃO À DIREITA - IV



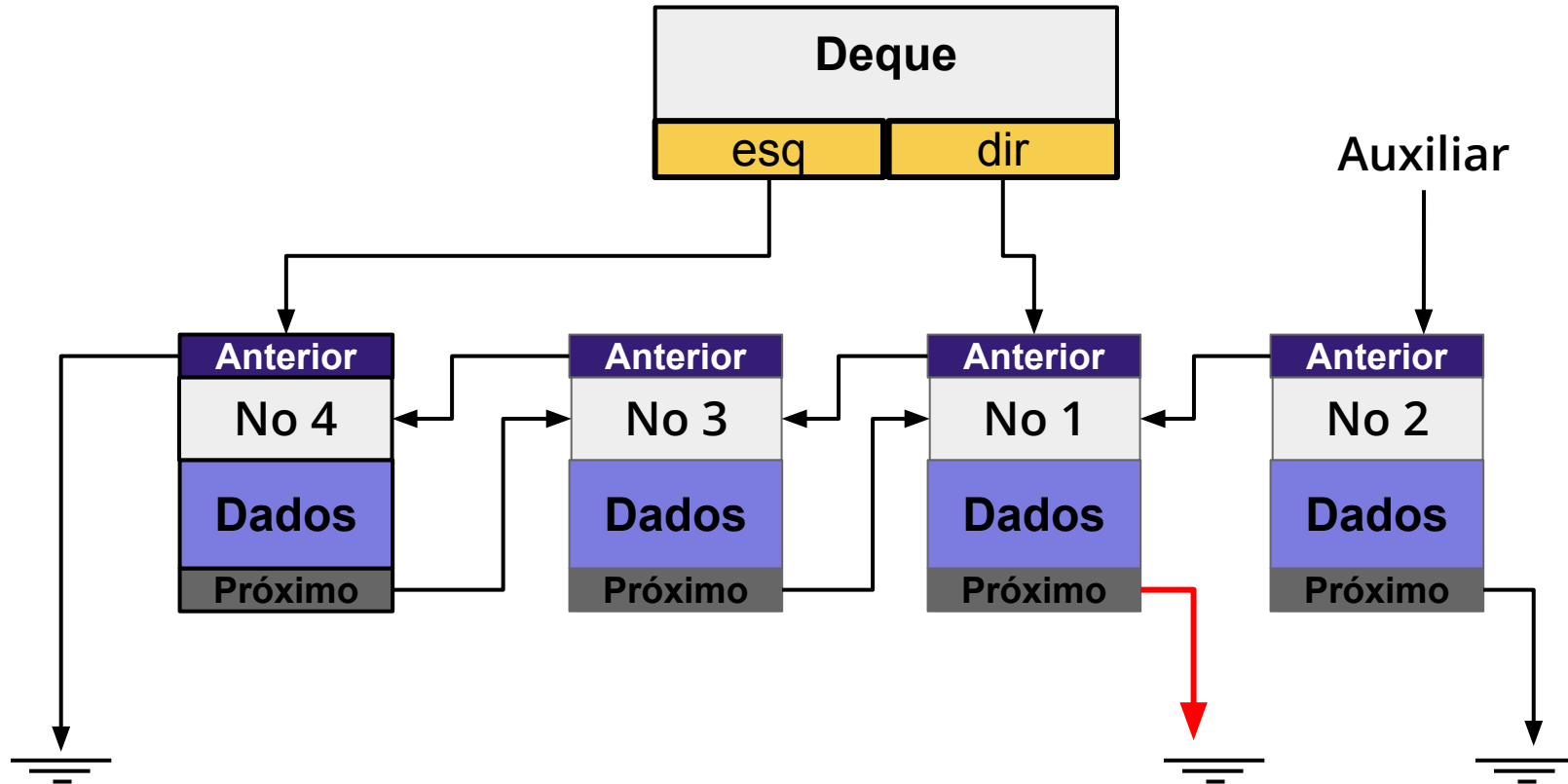
REMOÇÃO À DIREITA - I



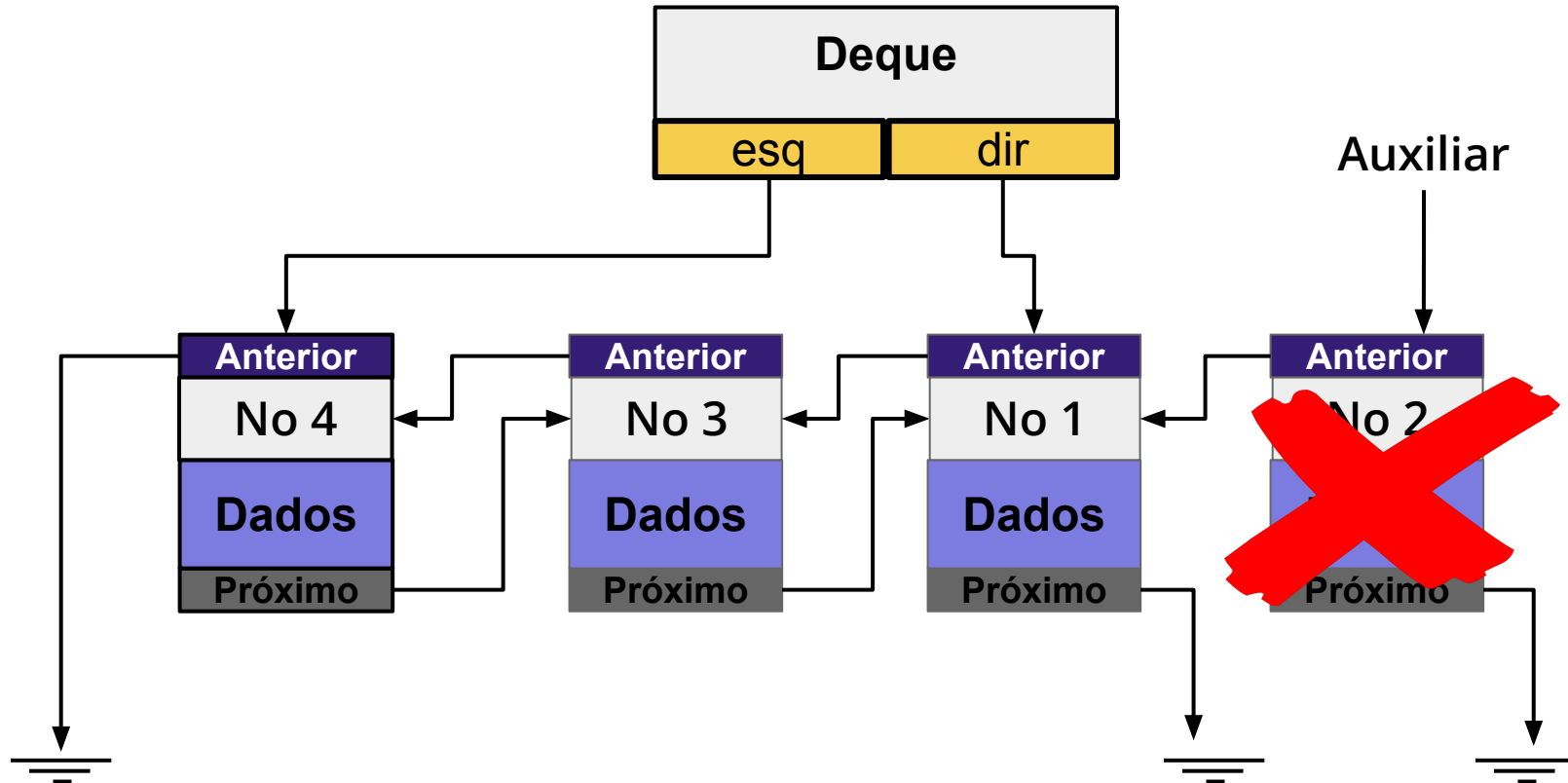
REMOÇÃO À DIREITA - II



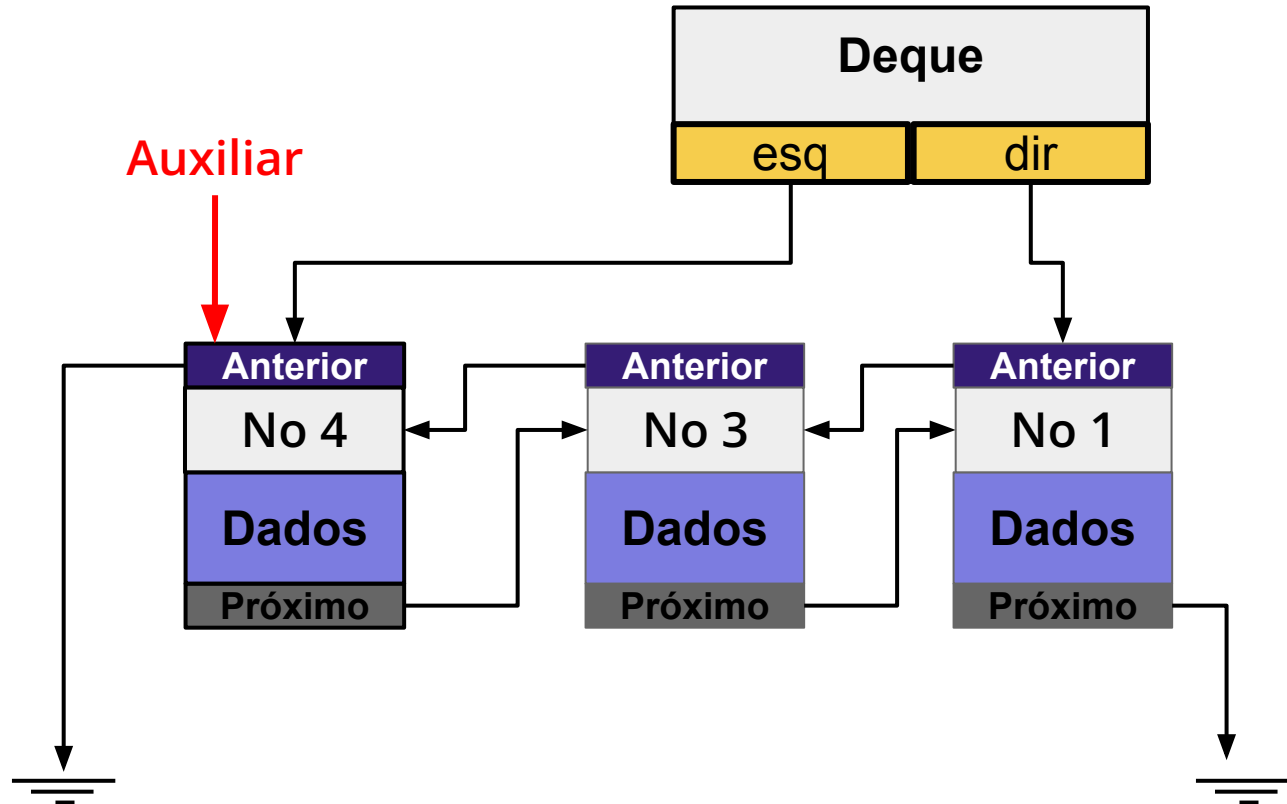
REMOÇÃO À DIREITA - III



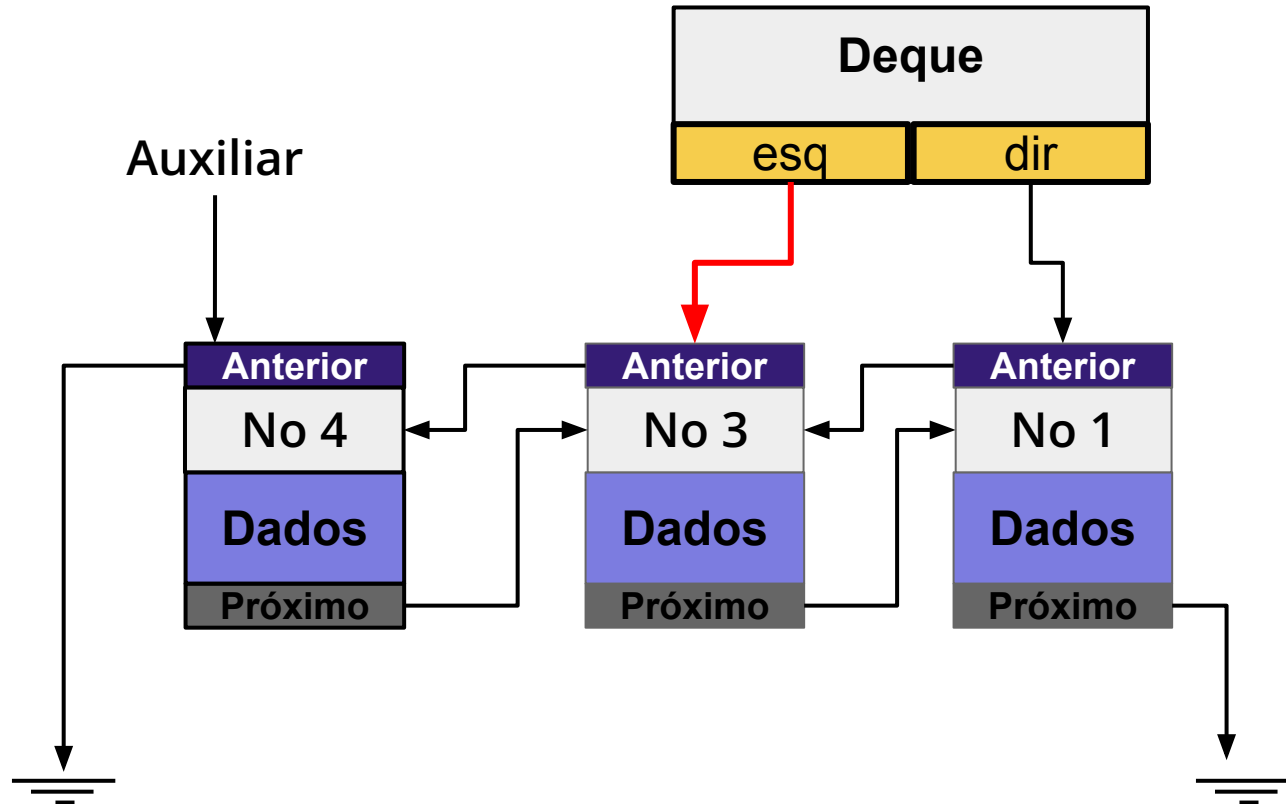
REMOÇÃO À DIREITA - IV



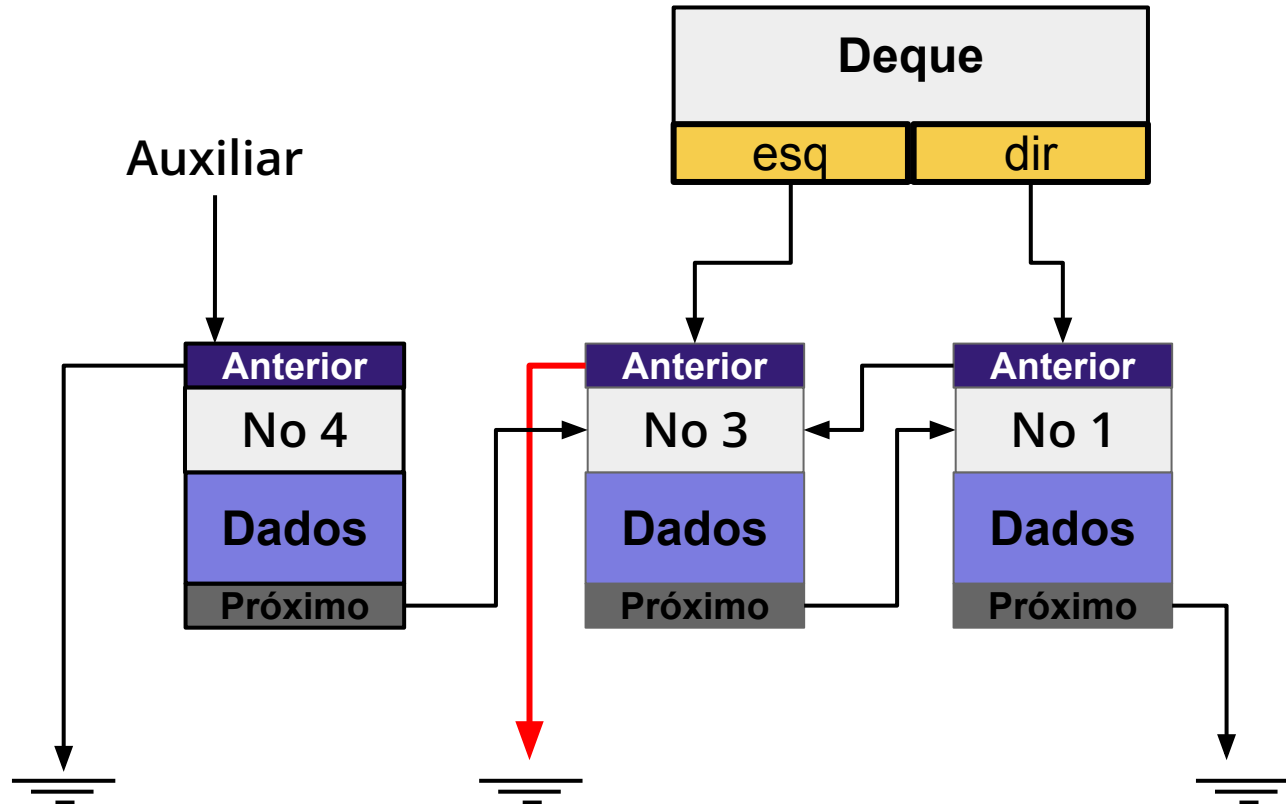
REMOÇÃO À ESQUERDA - I



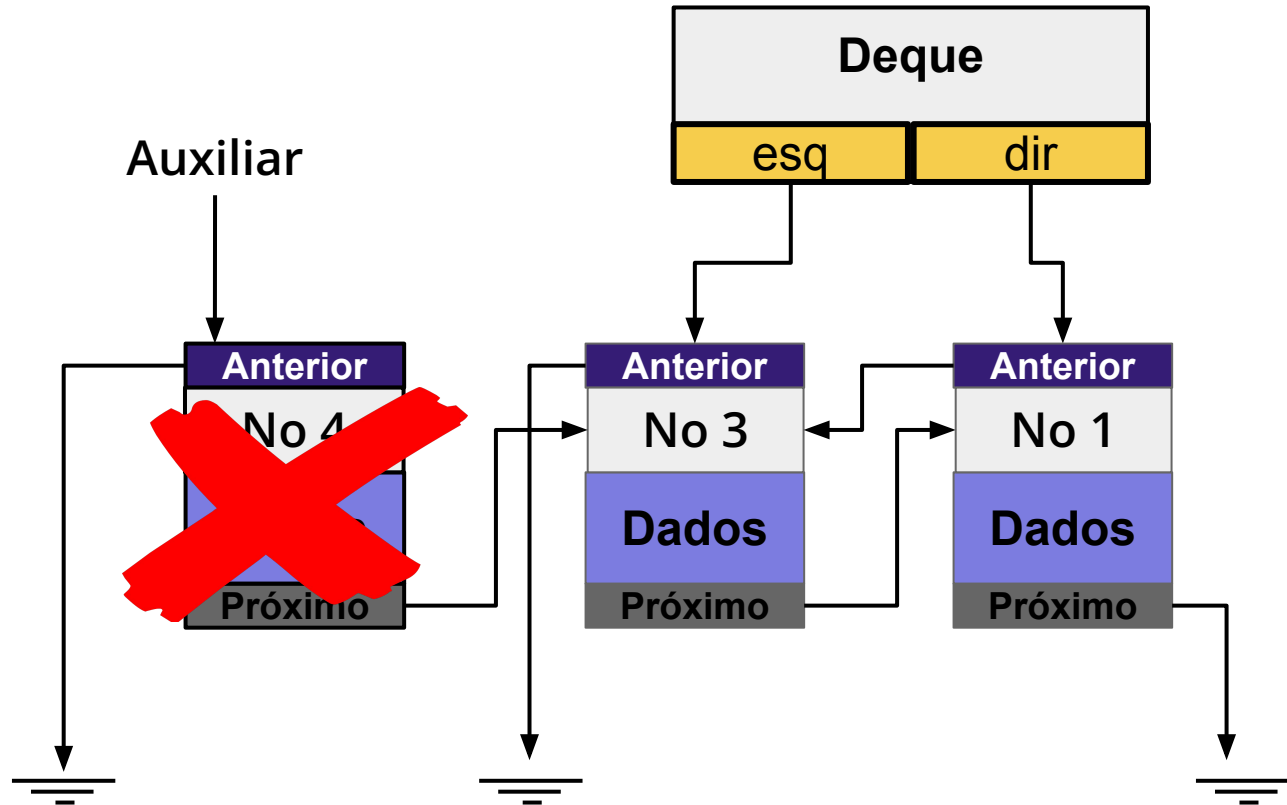
REMOÇÃO À ESQUERDA -II



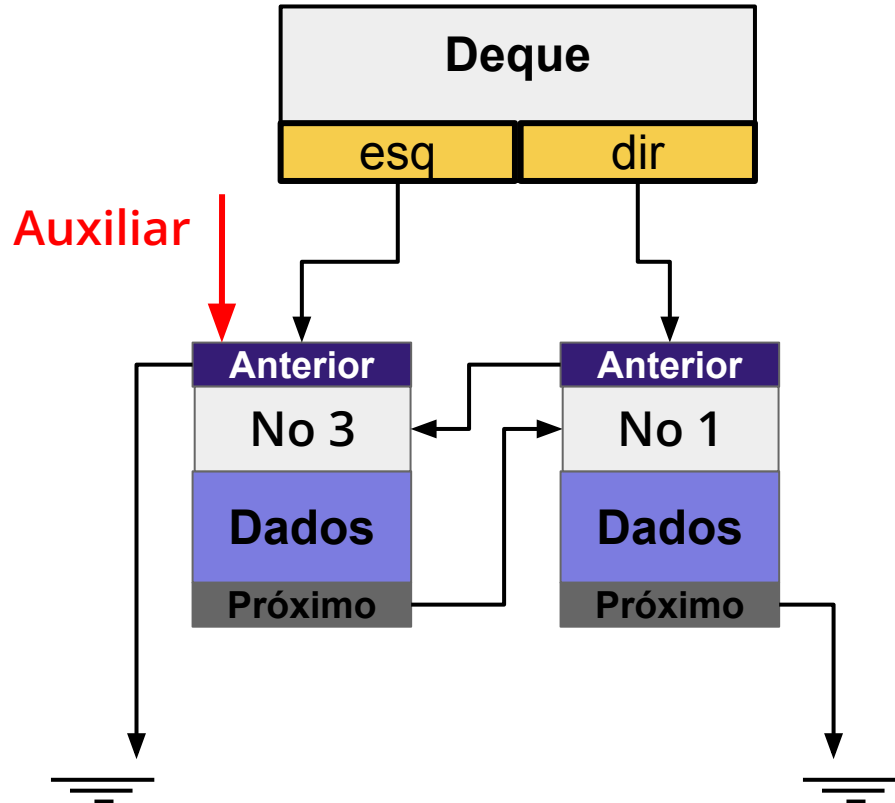
REMOÇÃO À ESQUERDA -IV



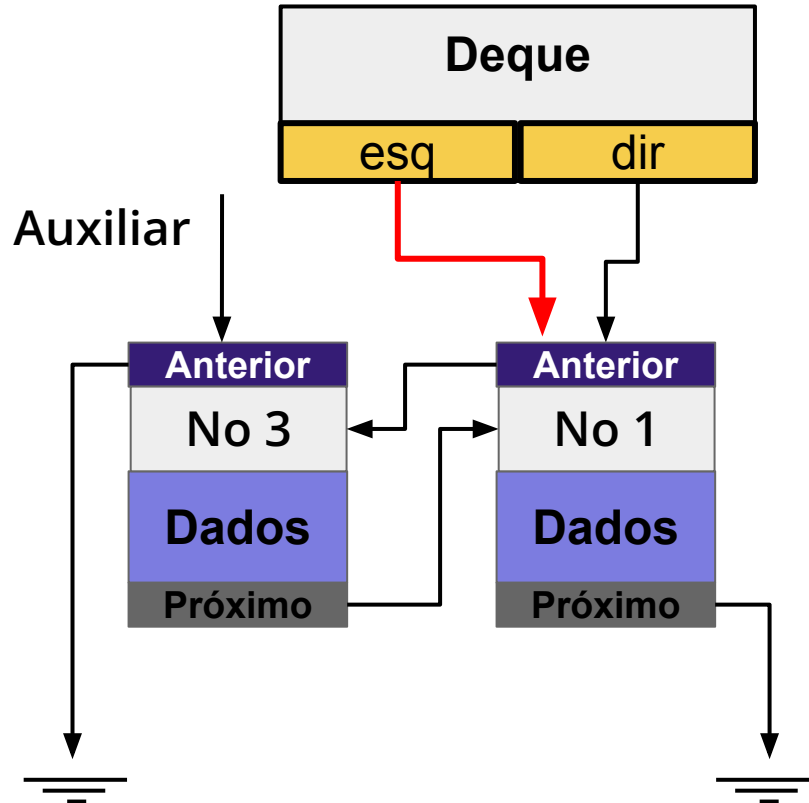
REMOÇÃO À ESQUERDA - V



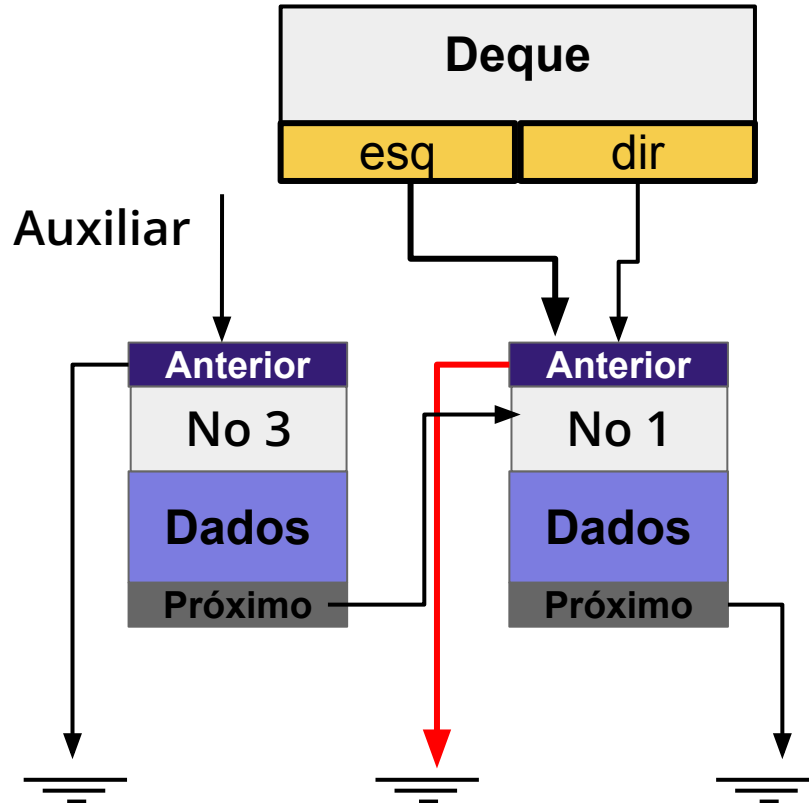
NOVA REMOÇÃO À ESQUERDA -I



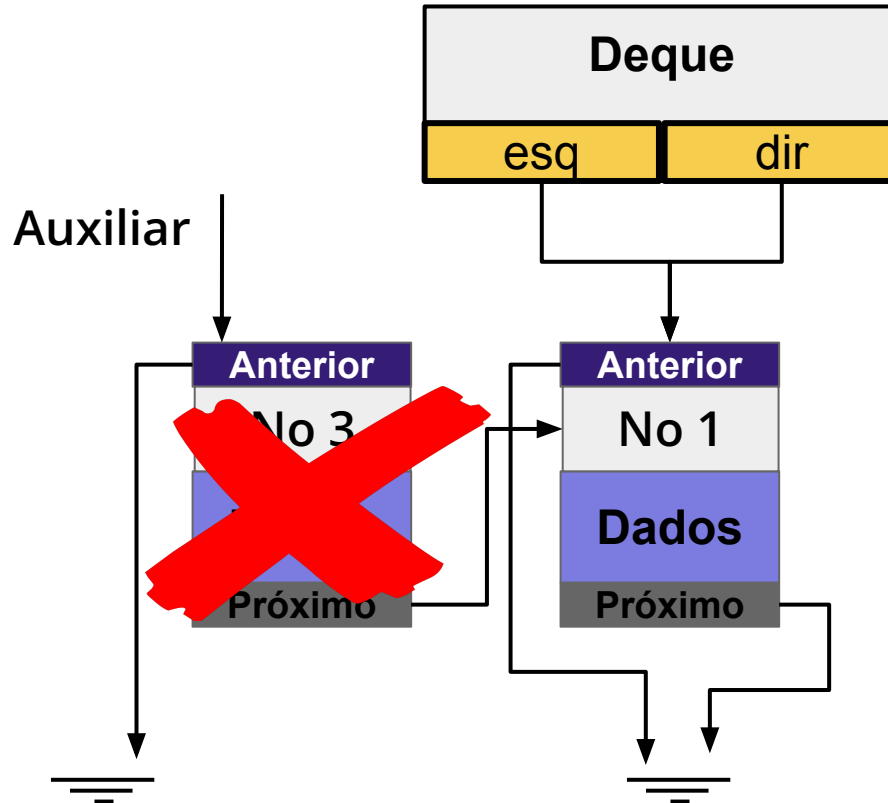
NOVA REMOÇÃO À ESQUERDA -II



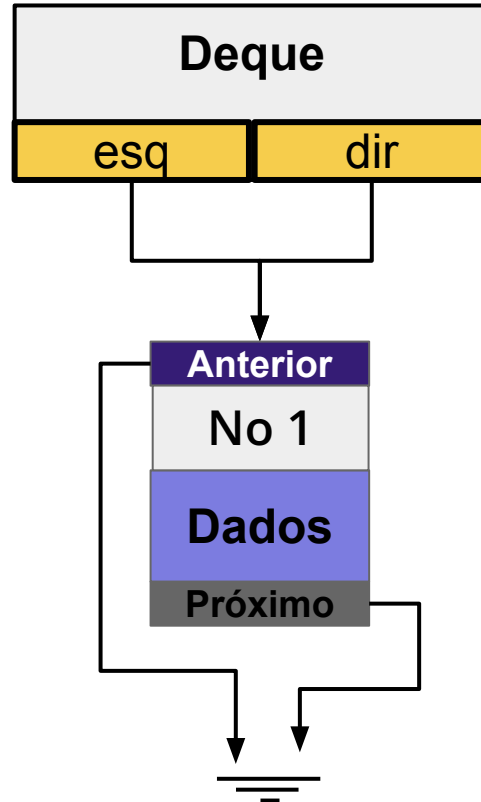
NOVA REMOÇÃO À ESQUERDA -IV



NOVA REMOÇÃO À ESQUERDA - V

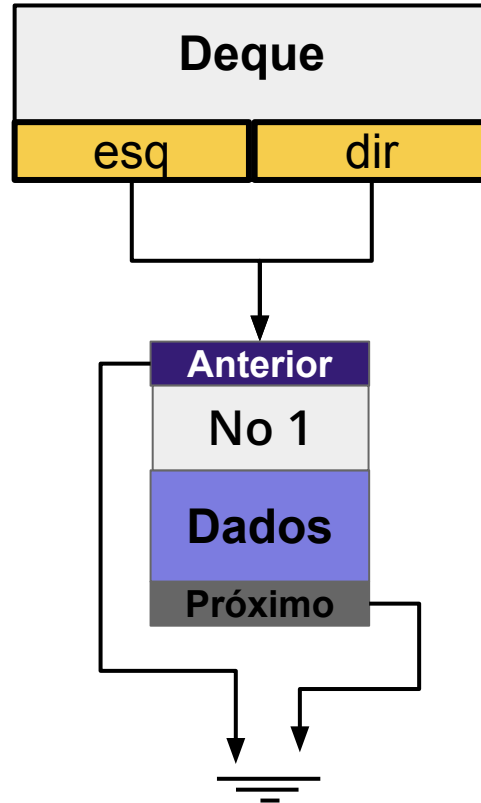


NOVA REMOÇÃO À ESQUERDA - VI

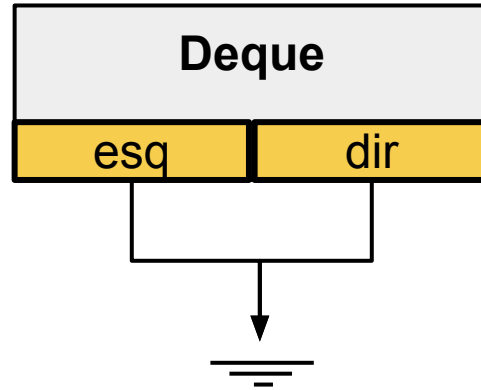


REMOÇÃO DO ÚLTIMO ELEMENTO

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 são uma característica
o fácil acesso a elementos
E quanto precisamos de uma
solução intermediária???

Em contraste, a matriz é uma
tarefa com redimensionamento, mas
também de rápido acesso...

Em listas, ocorre o inverso.

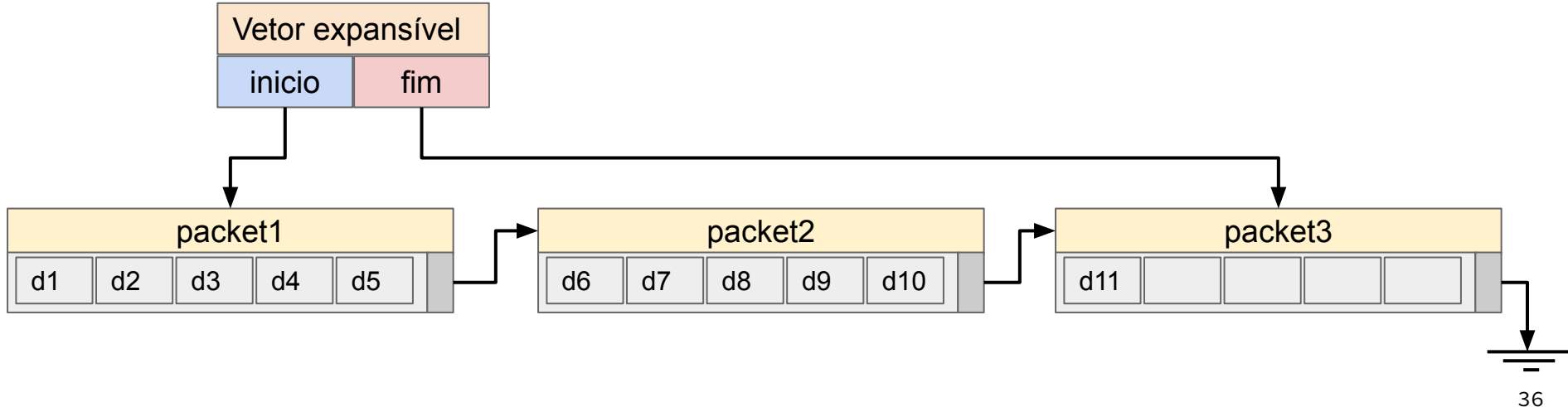
COMENTÁRIOS INICIAIS

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.

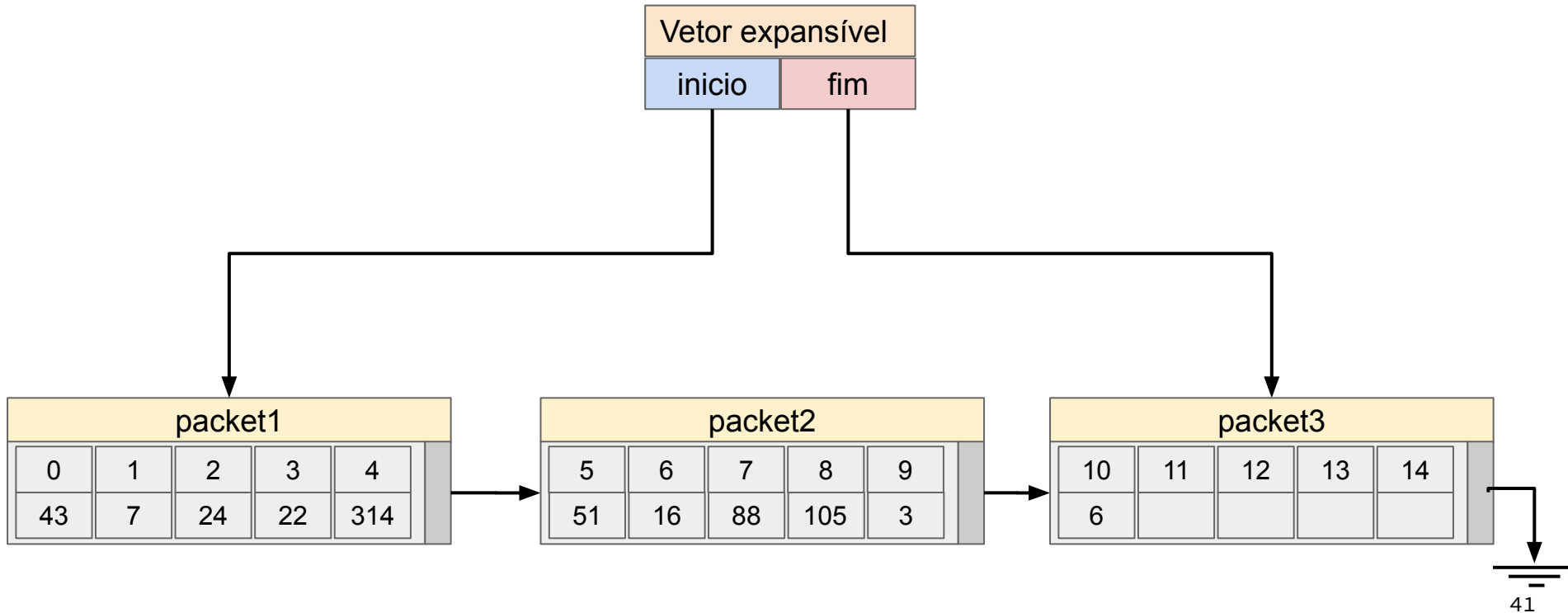
Nesse caso, cada pedaço possui, além dos dados e ponteiros, um atributo para armazenar sua própria capacidade.

VETORES EXPANSÍVEIS - V

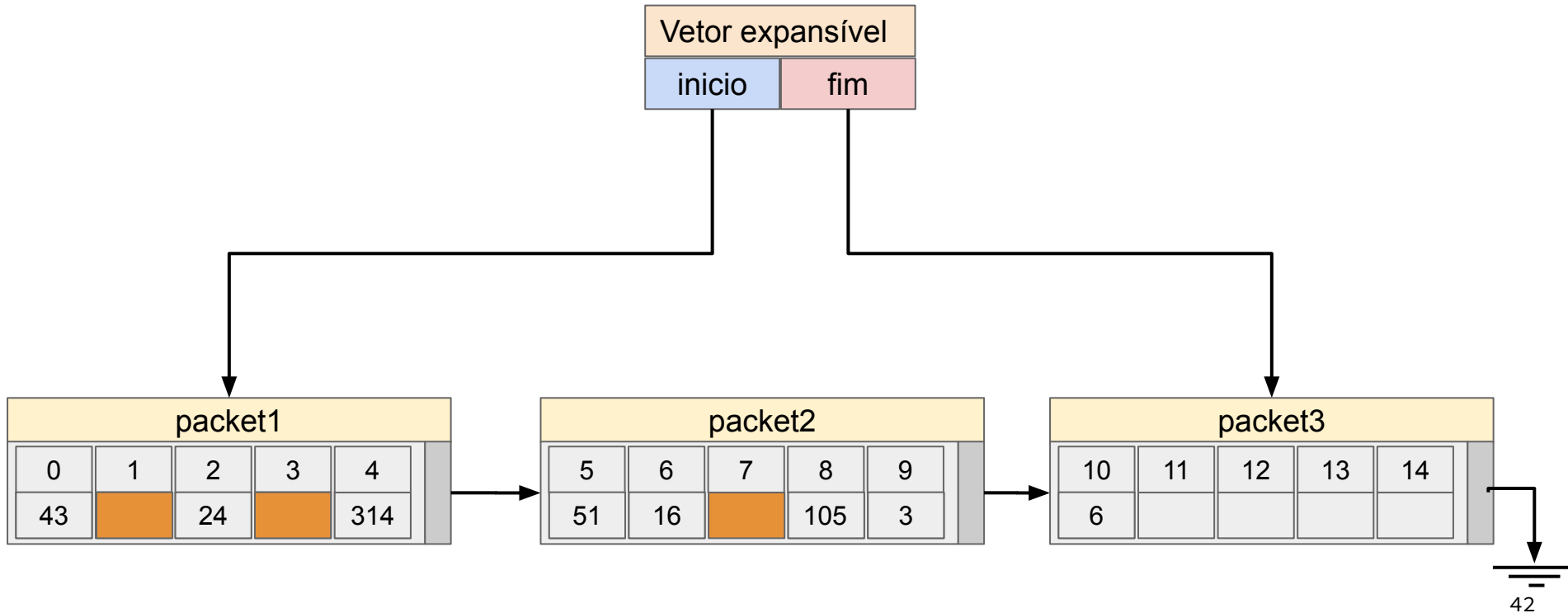
Apesar que conceitualmente não há restrição quanto ao tipo de objeto armazenados, a menos que o problema não precise de remoção, os dados precisam estar ordenados.

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

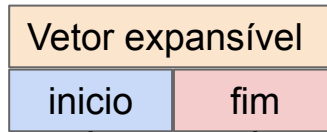
EXEMPLO- DADOS NÃO-ORDENADOS - REMOÇÃO - I



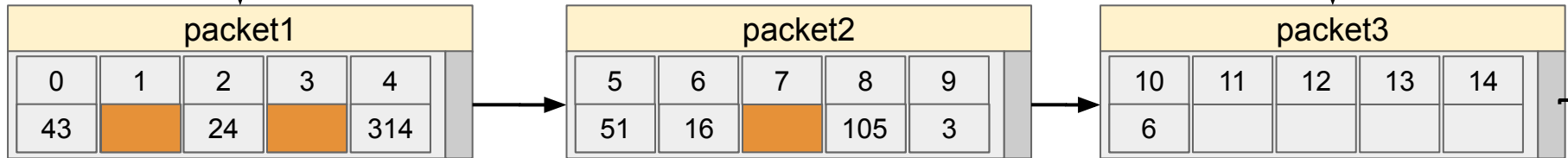
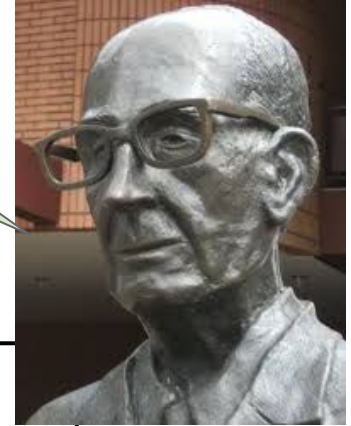
EXEMPLO- DADOS NÃO-ORDENADOS - REMOÇÃO - II



EXEMPLO- DADOS NÃO-ORDENADOS - REMOÇÃO - II



E agora,
José?



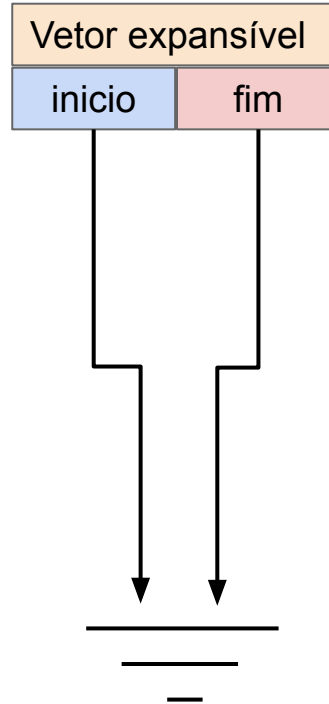
REMOÇÃO - DADOS NÃO-ORDENADOS

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

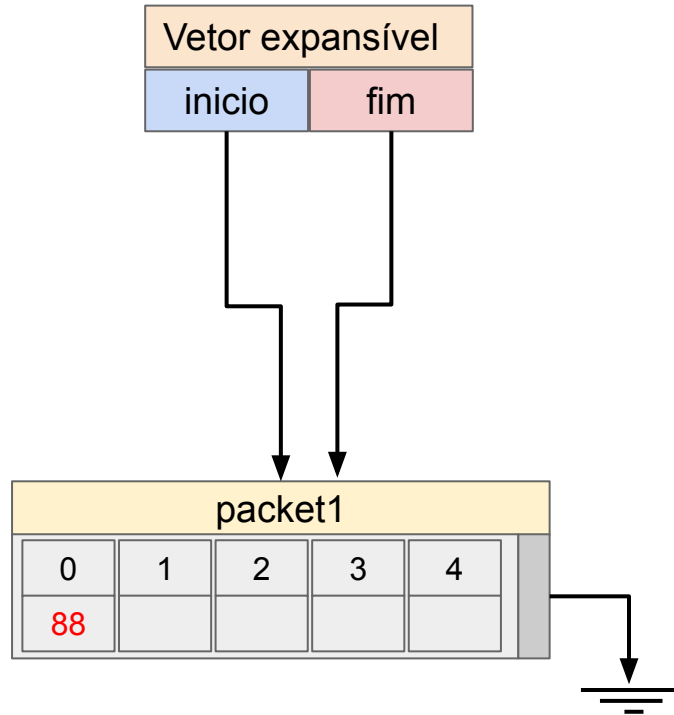
- Ou o percorrimto e busca vão ter que pular nós inválidos, o que torna o processo ineficiente, sendo melhor usar uma lista diretamente.
- Ou será necessário rearranjar os packets a todo momento, sendo melhor usar um vetor diretamente.

Assim, iremos trabalhar com vetores expansíveis com dados ordenados.

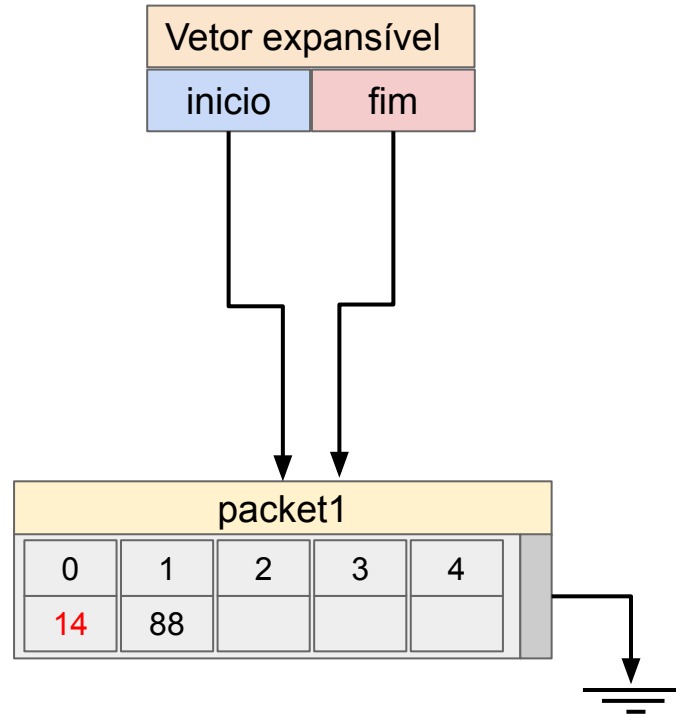
EXEMPLO - INSERÇÃO - I



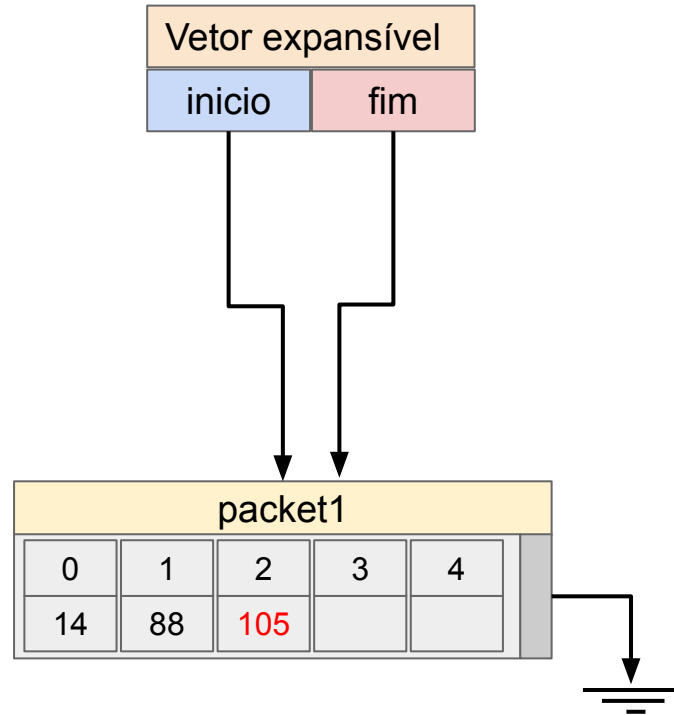
EXEMPLO - INSERÇÃO - II



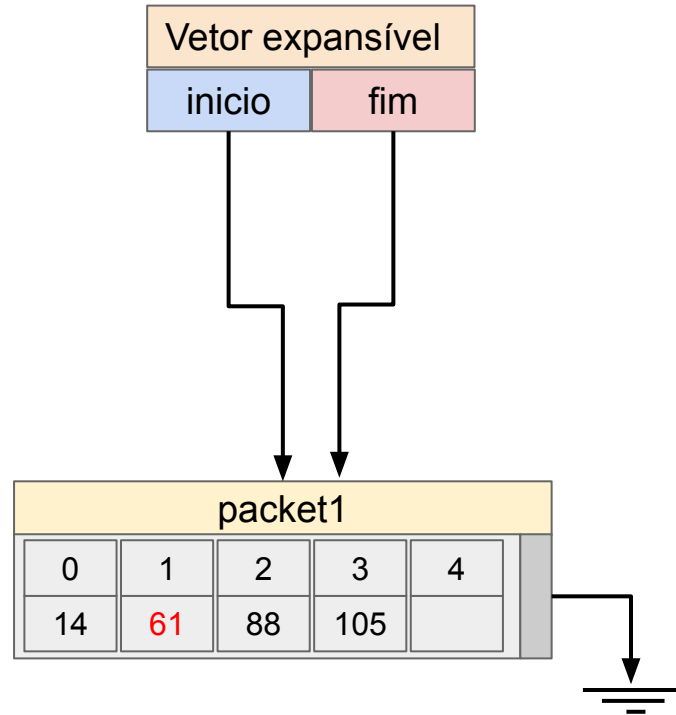
EXEMPLO - INSERÇÃO - III



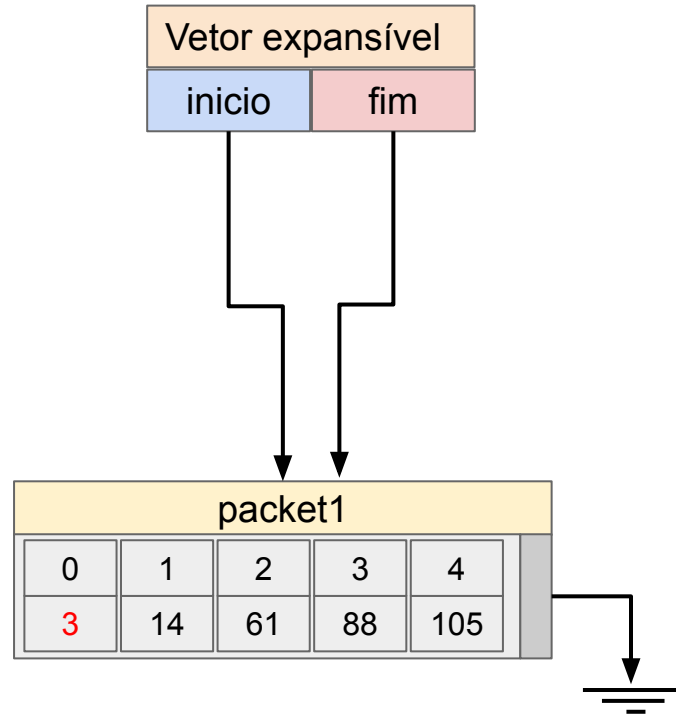
EXEMPLO - INSERÇÃO - IV



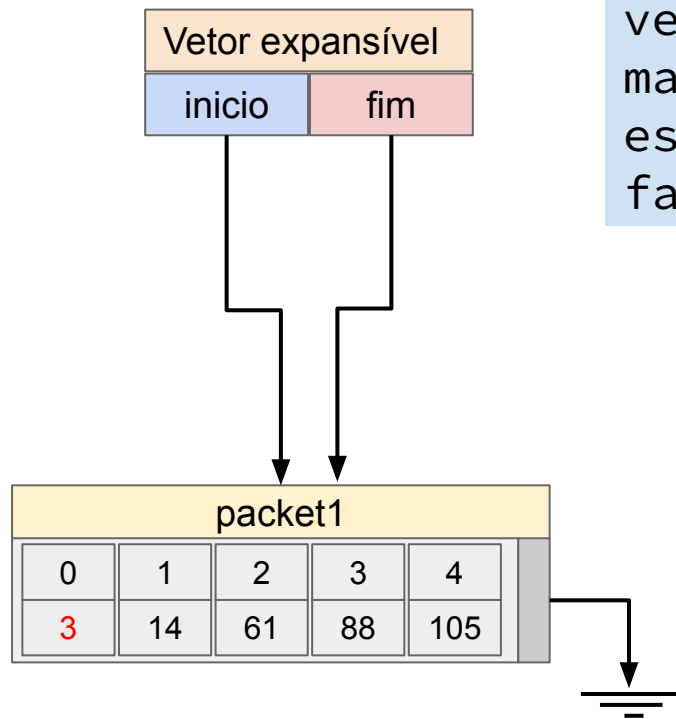
EXEMPLO - INSERÇÃO - V



EXEMPLO - INSERÇÃO - VI



EXEMPLO - INSERÇÃO - VII

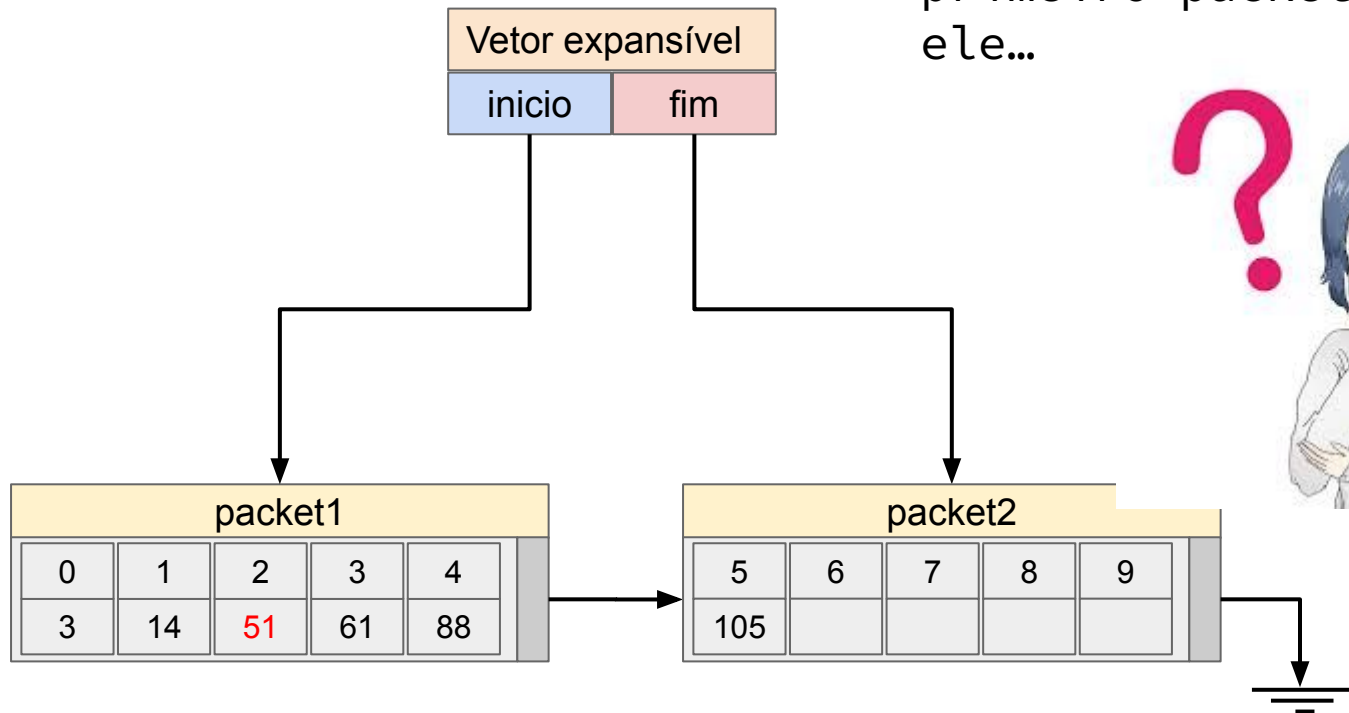


Queremos agora inserir o 51 no vetor expansível, mas o packet1 está cheio, como fazer?

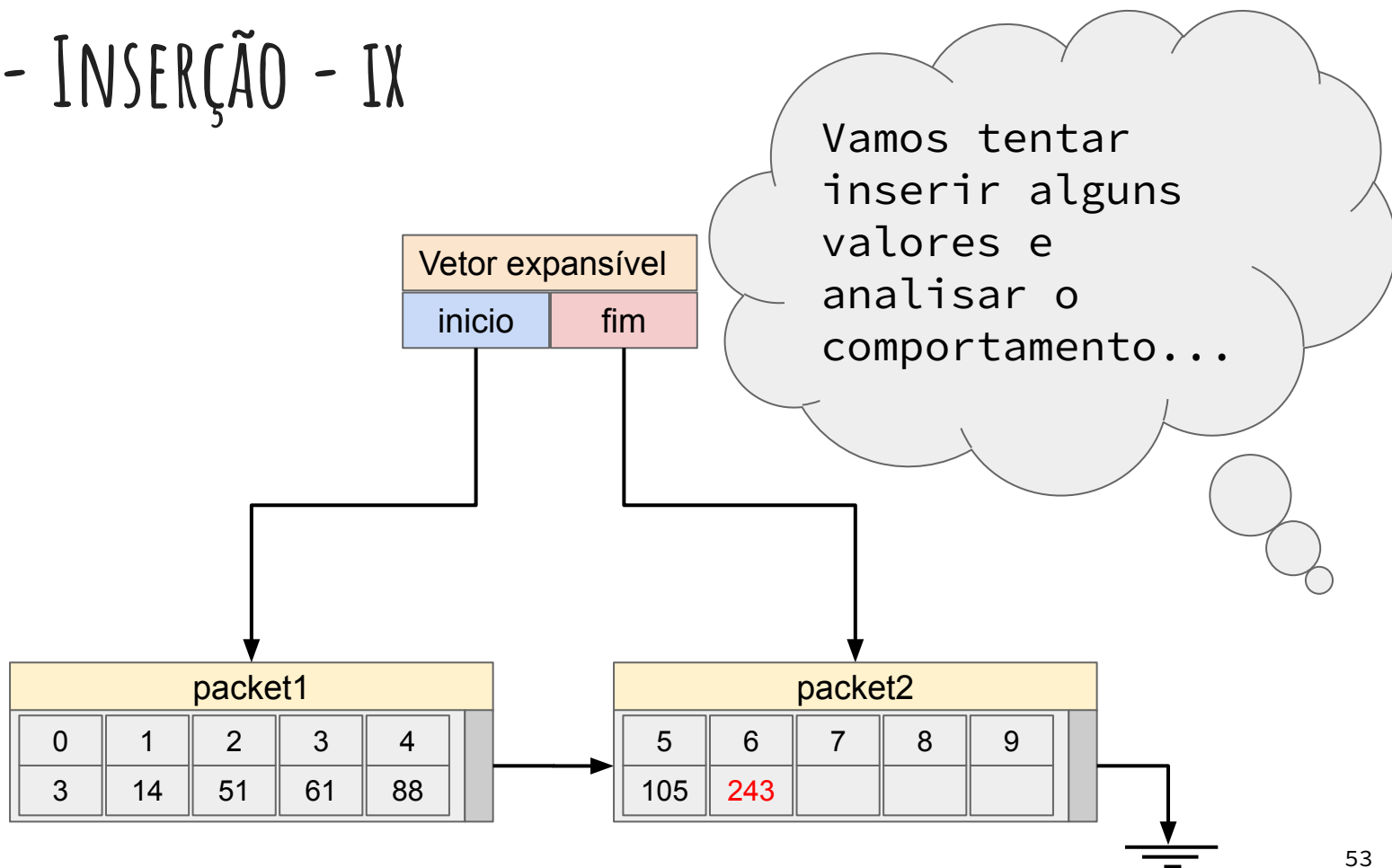


EXEMPLO - INSERÇÃO - VIII

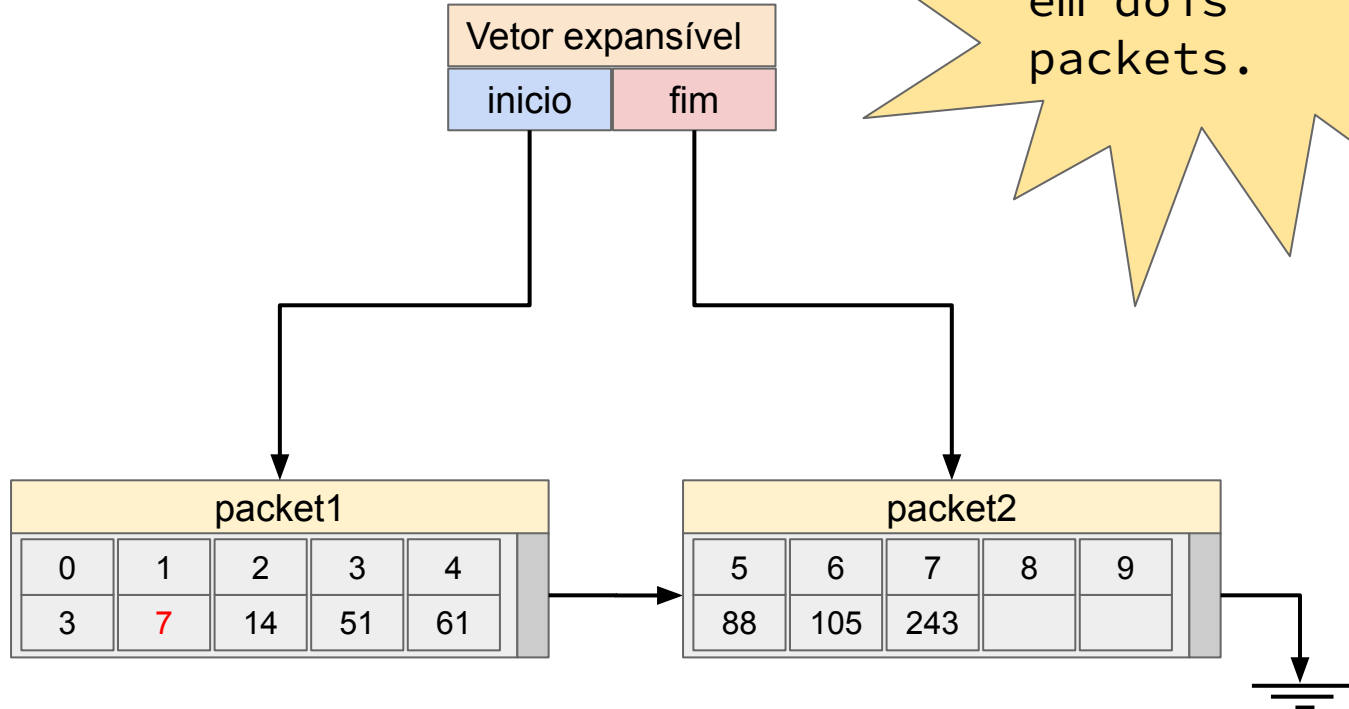
Talvez possamos criar um novo packet e mover o excedente do primeiro packet para ele...



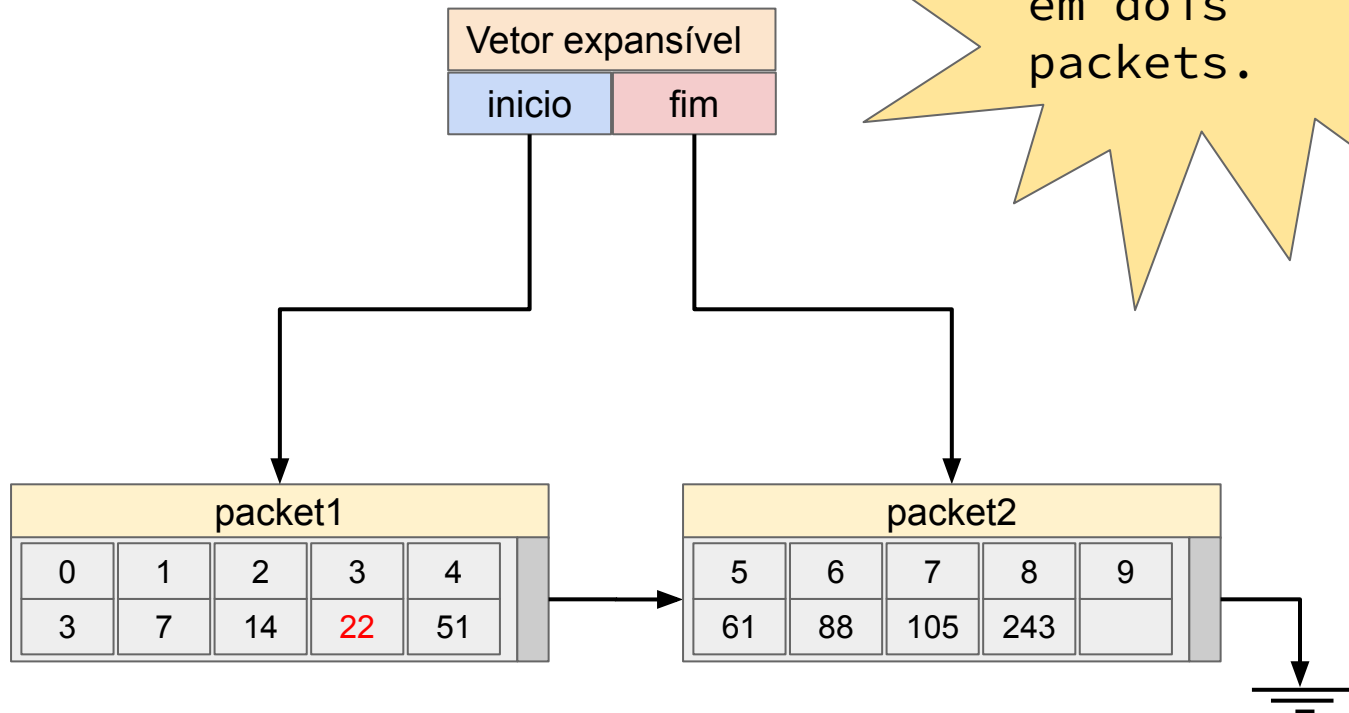
EXEMPLO - INSERÇÃO - IX



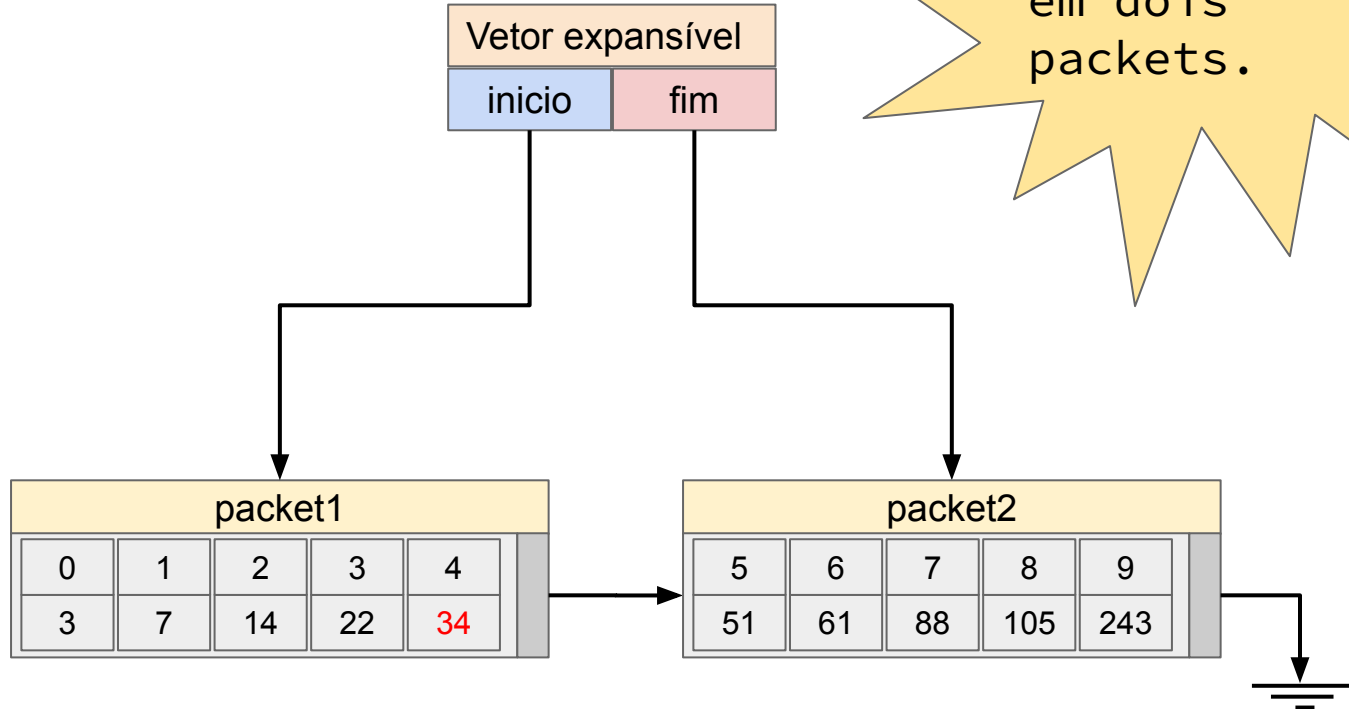
EXEMPLO - INSERÇÃO - X



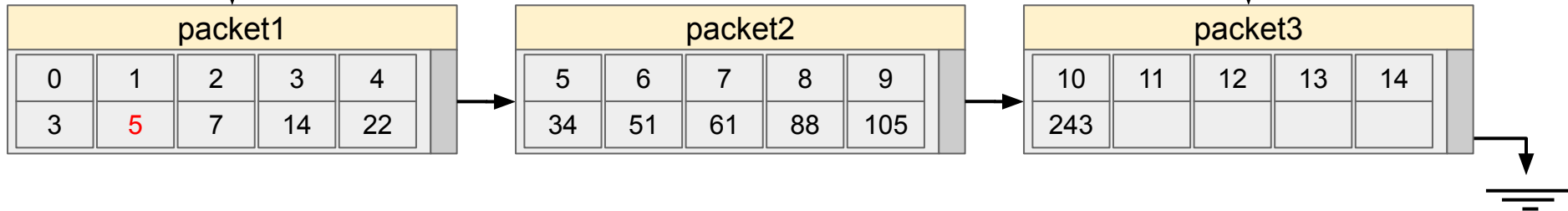
EXEMPLO - INSERÇÃO - XI



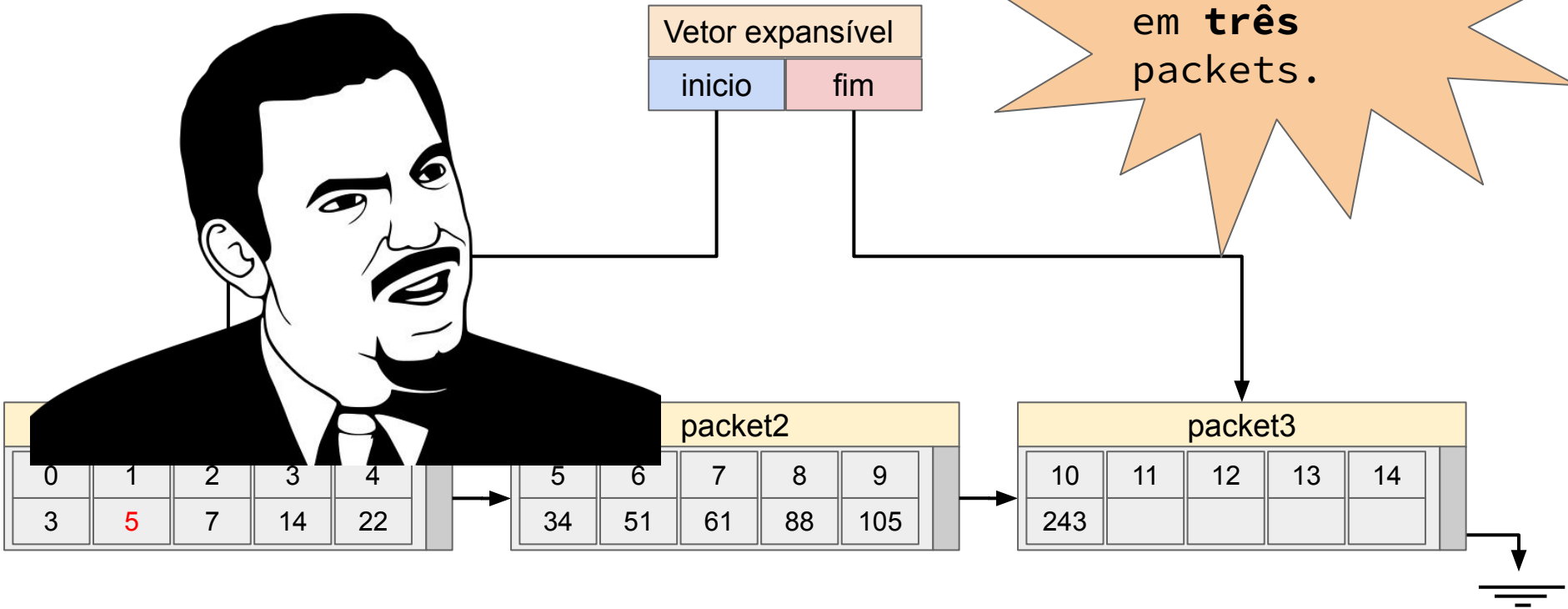
EXEMPLO - INSERÇÃO - XII



EXEMPLO - INSERÇÃO - XIII



EXEMPLO - INSERÇÃO - XIII



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.

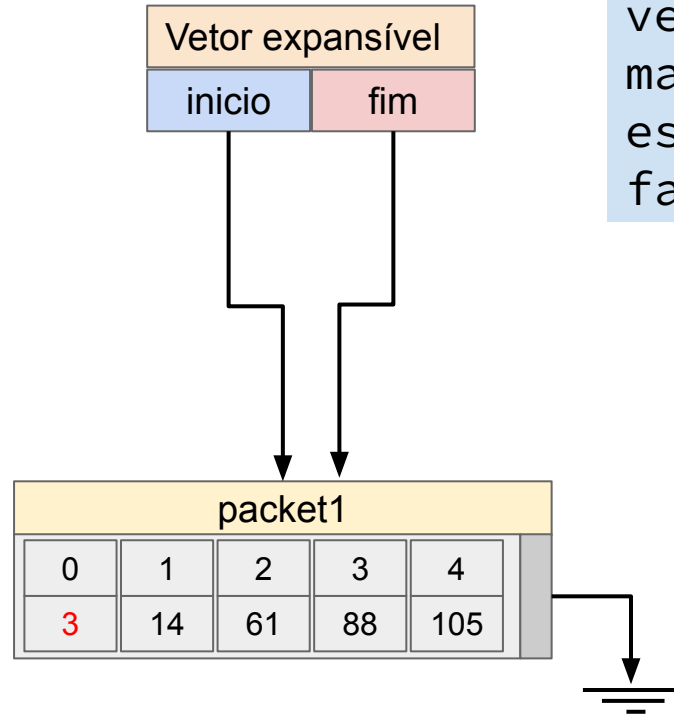
INSERÇÃO EM PACKET CHEIO - II

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.



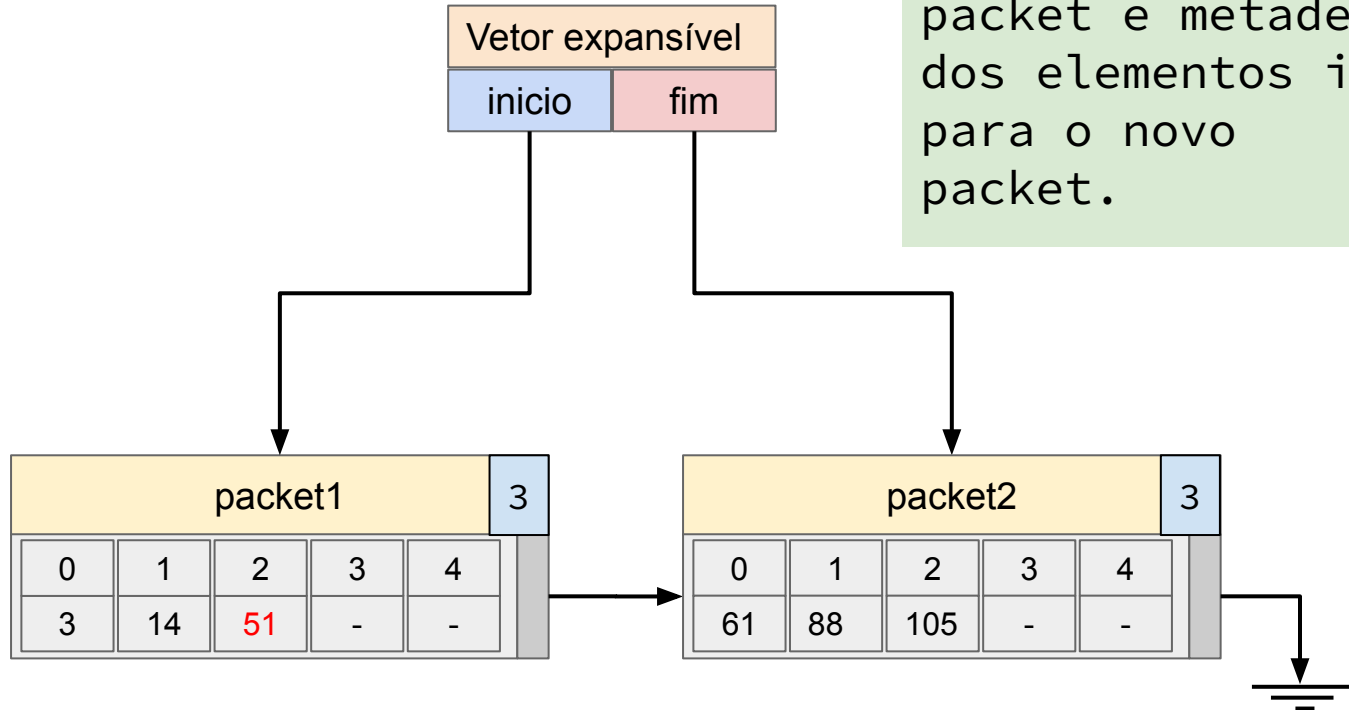
REFAZENDO O EXEMPLO- INSERÇÃO - I



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



REFAZENDO O EXEMPLO- INSERÇÃO - II

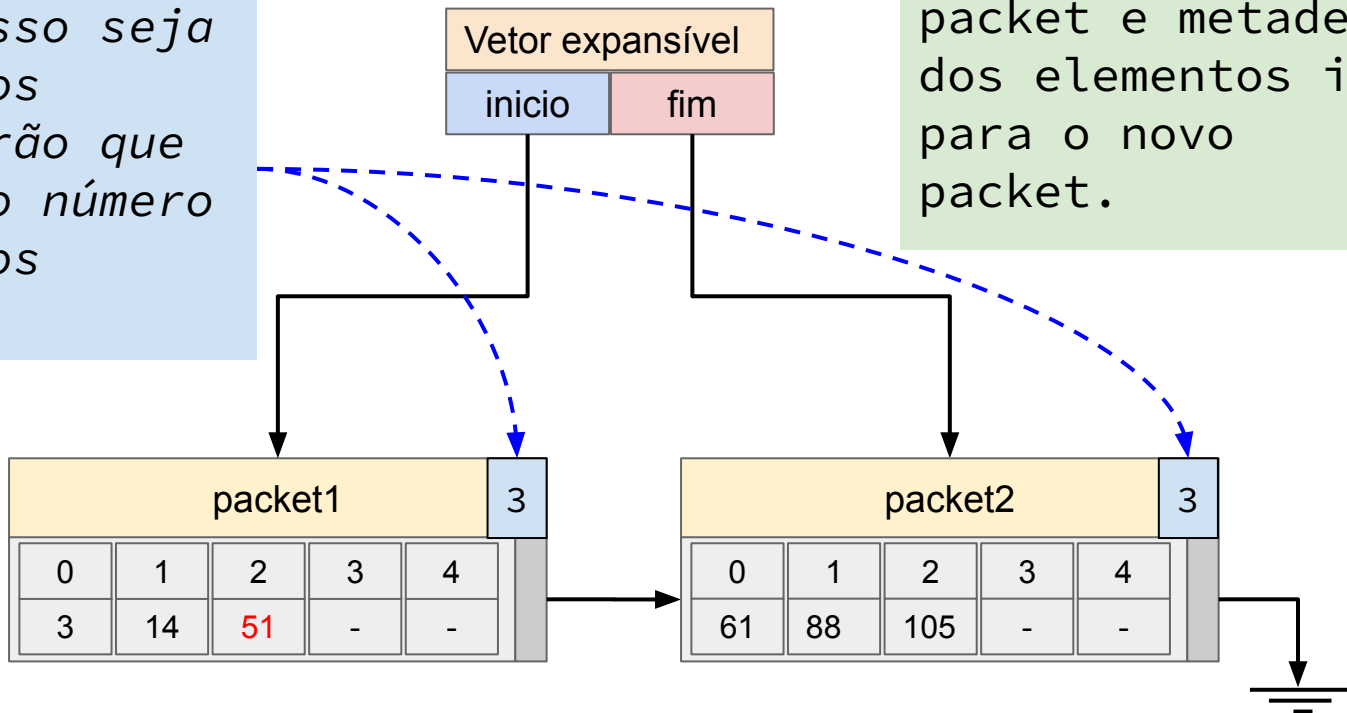


Dividimos o packet e metade dos elementos irá para o novo packet.

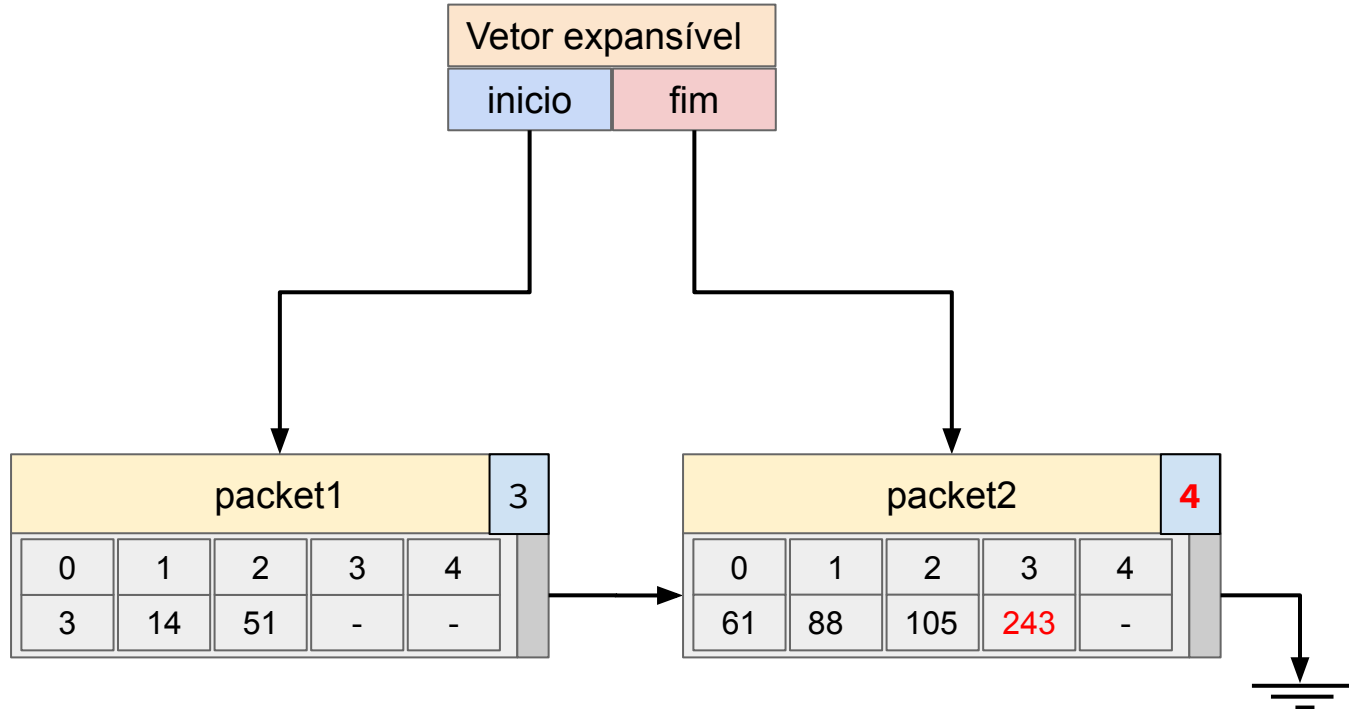
REFAZENDO O EXEMPLO- INSERÇÃO - II

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

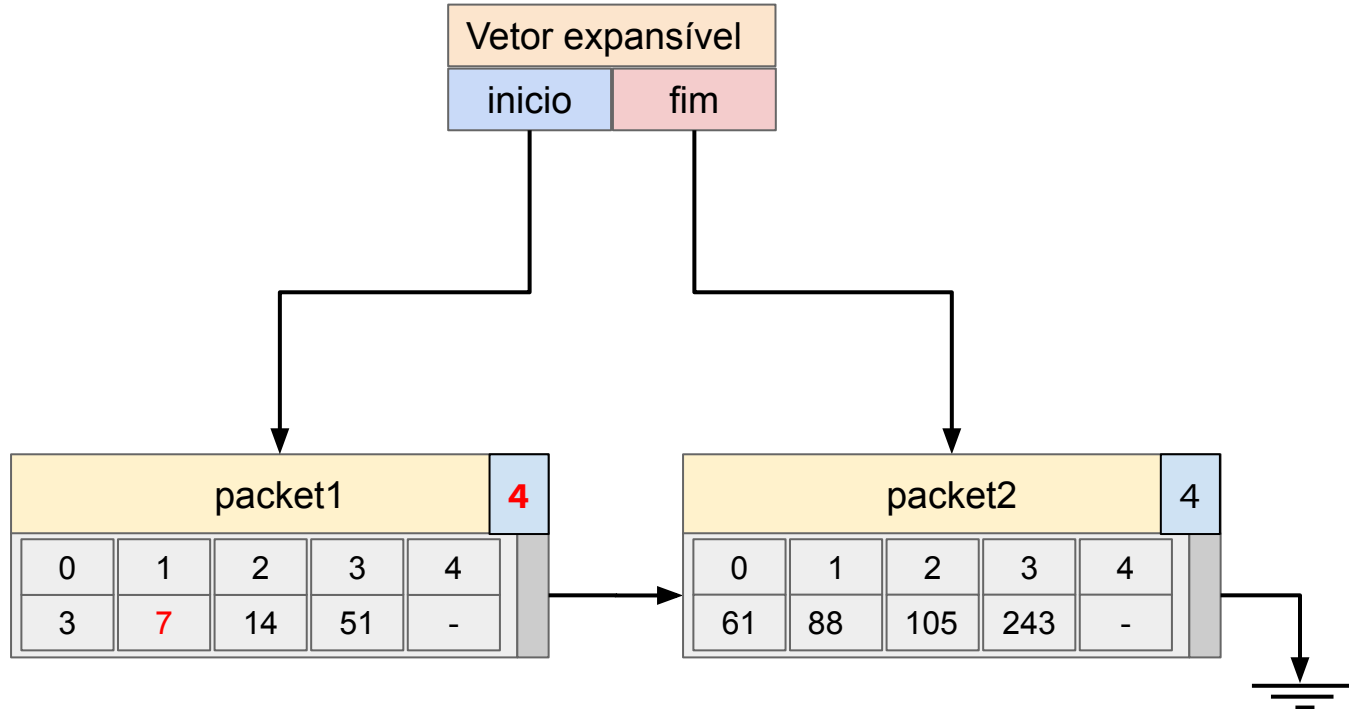
Dividimos o packet e metade dos elementos irá para o novo packet.



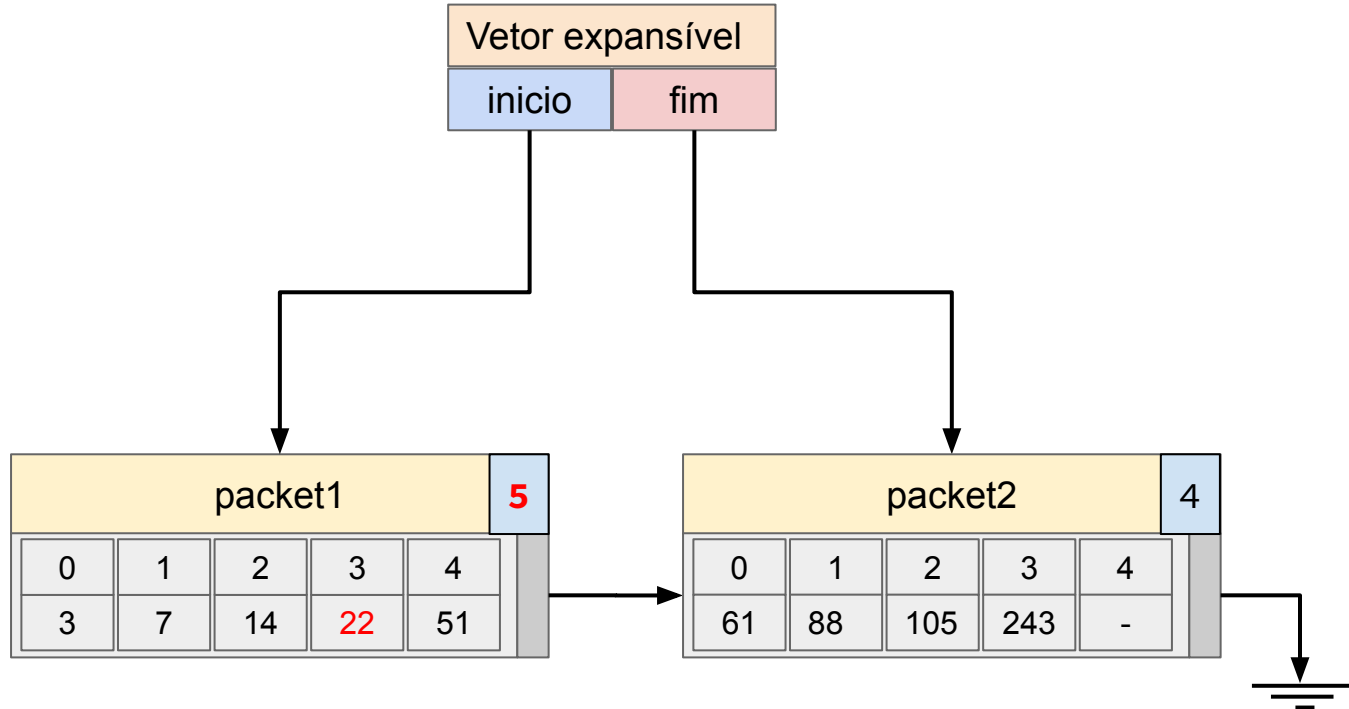
REFAZENDO O EXEMPLO- INSERÇÃO - III



REFAZENDO O EXEMPLO - INSERÇÃO - IV

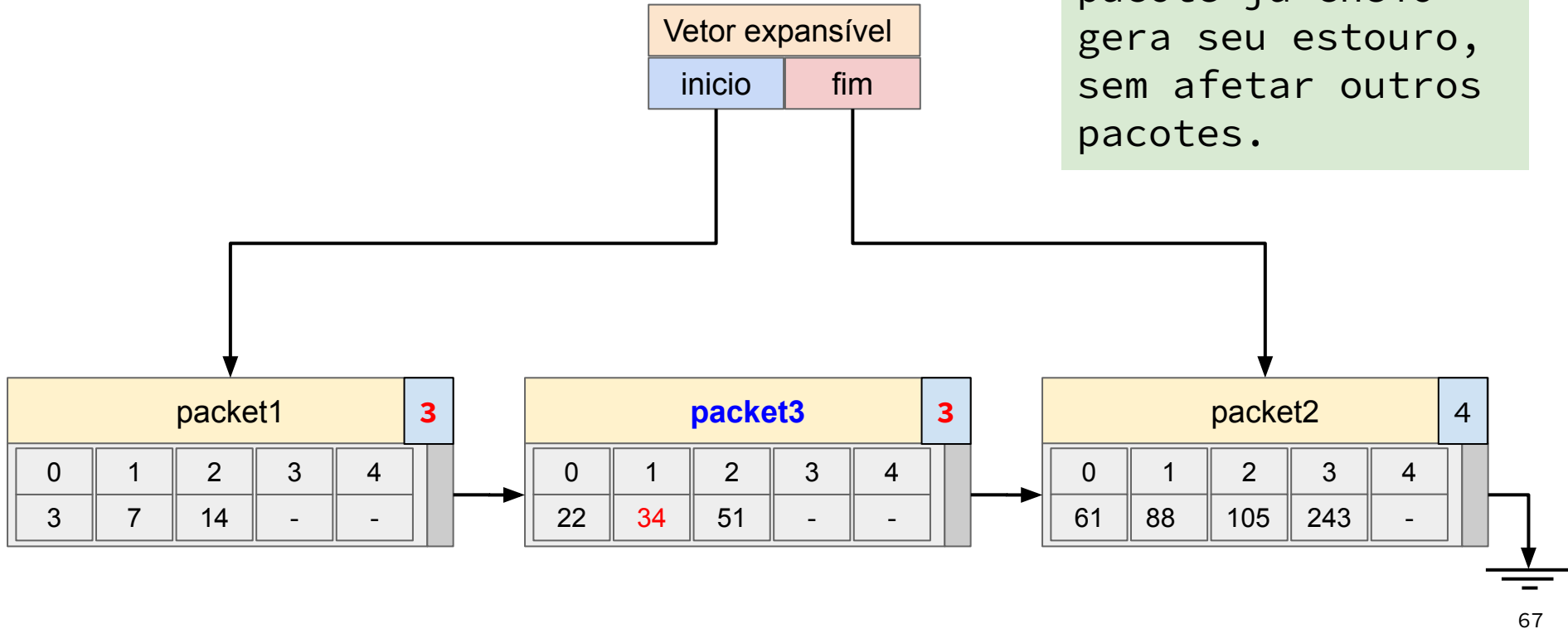


REFAZENDO O EXEMPLO- INSERÇÃO - V

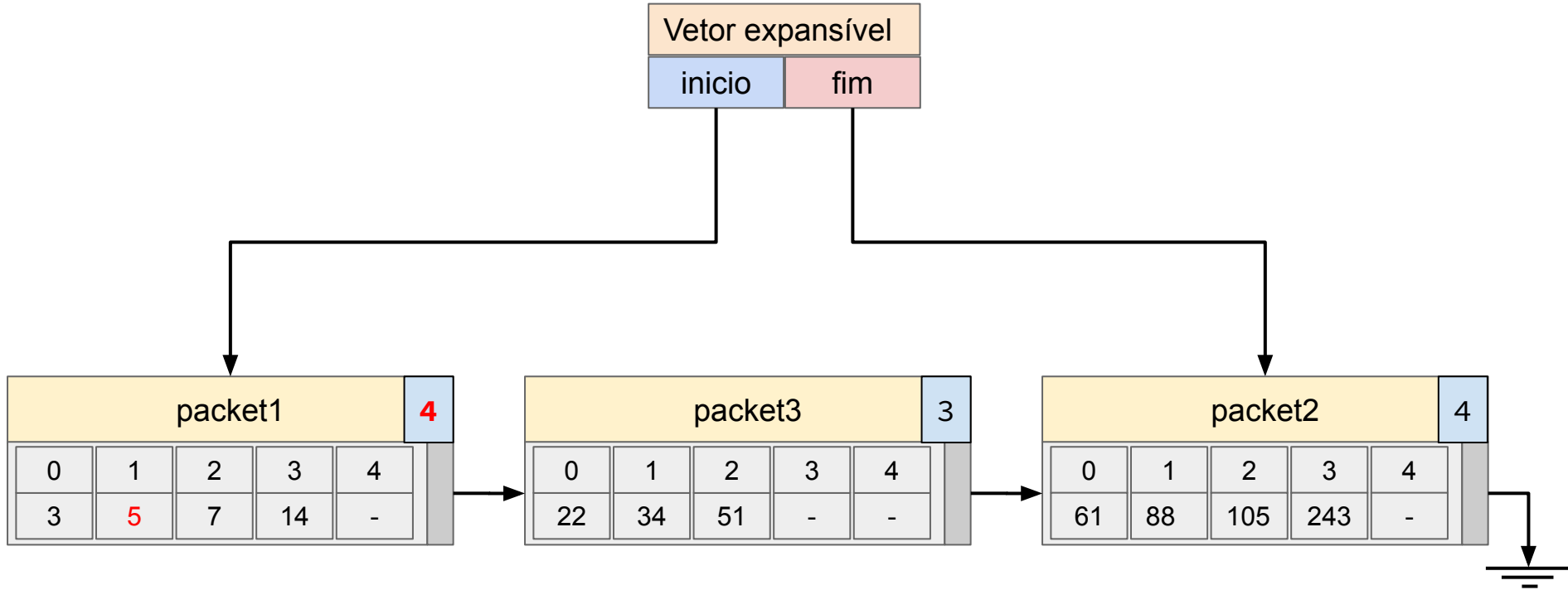


REFAZENDO O EXEMPLO- INSERÇÃO - VI

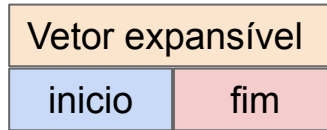
Inserção em um pacote já cheio gera seu estouro, sem afetar outros pacotes.



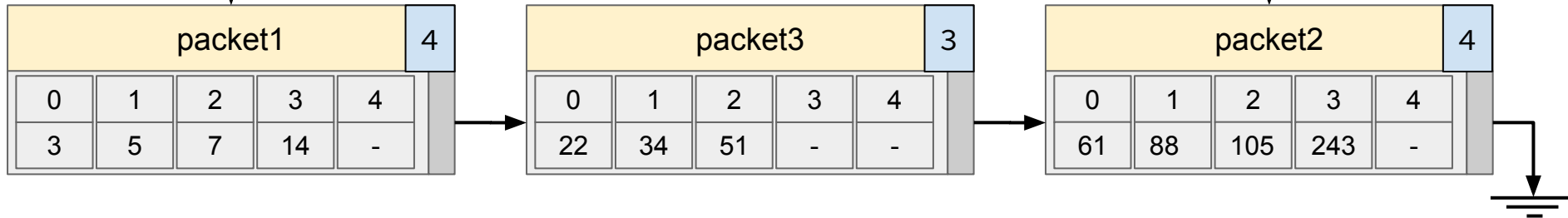
REFAZENDO O EXEMPLO- INSERÇÃO - VII



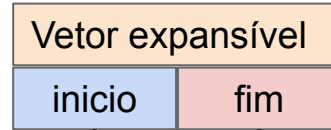
REFAZENDO O EXEMPLO- INSERÇÃO - VII



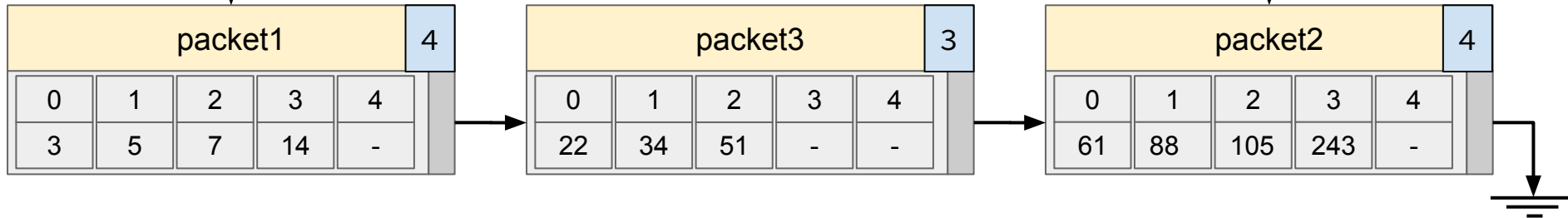
E se quisermos agora inserir um valor entre 14 e 22, por exemplo 18?



REFAZENDO O EXEMPLO- INSERÇÃO - VII



Ele iria no packet1 ou no packet3???



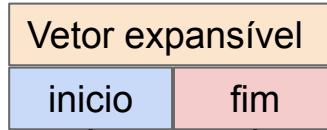
PENSANDO NA INSERÇÃO

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

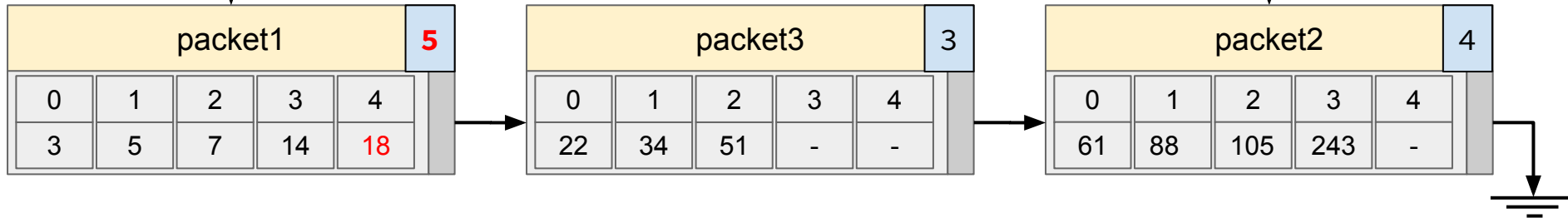
Quando o dado a ser inserido poderia ser armazenado em um pacote, no próximo ou anterior, é necessário avaliar o que irá gerar menos impacto computacional.

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

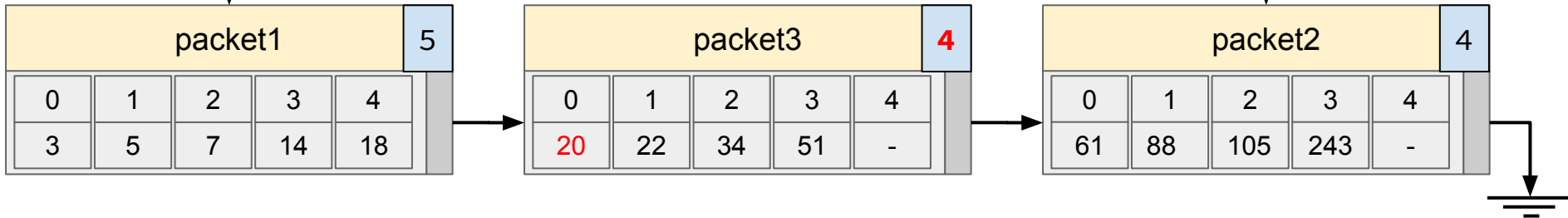
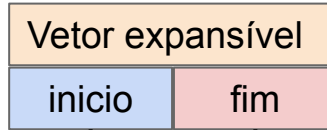


No nosso caso, então, é melhor inserir o 18 no packet1, para minimizar esforço.



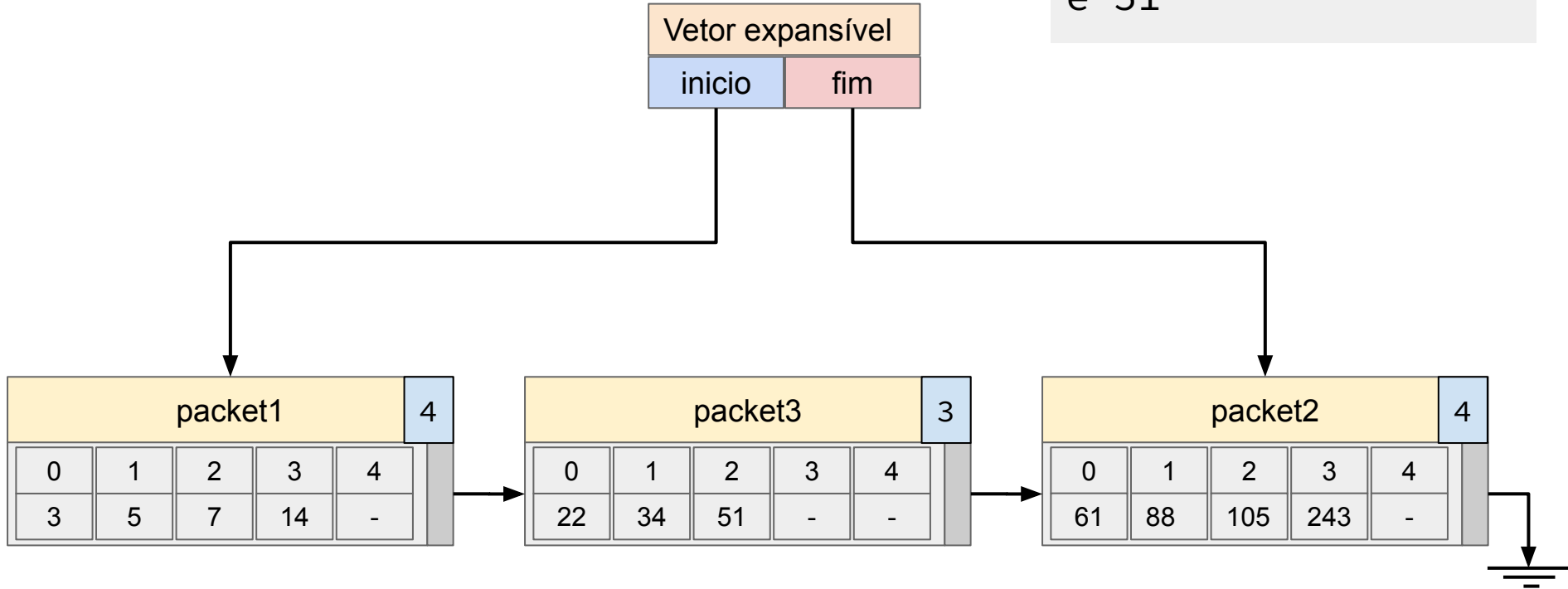
REFAZENDO O EXEMPLO- INSERÇÃO - IX

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



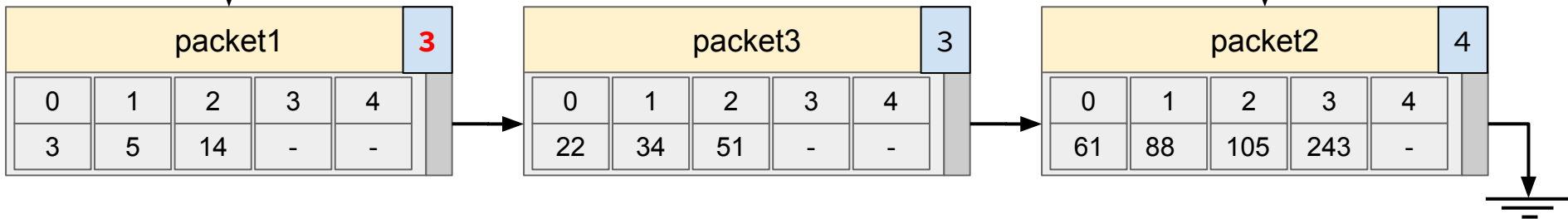
EXEMPLO- REMOÇÃO - I

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



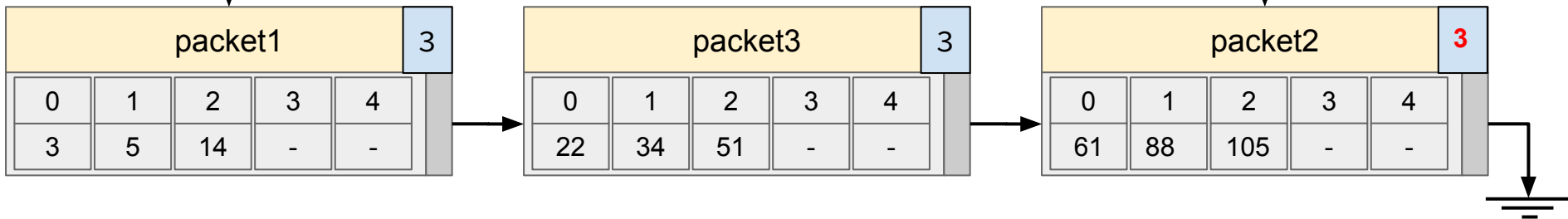
EXEMPLO- REMOÇÃO - II

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



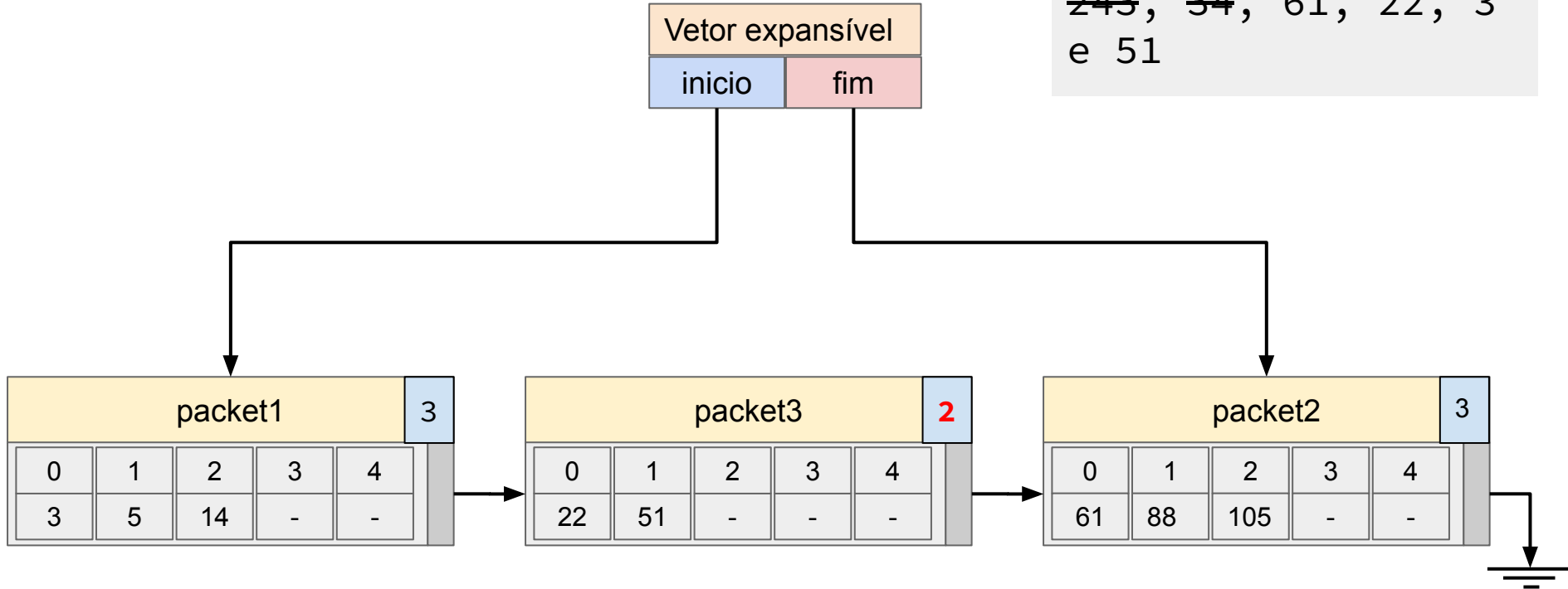
EXEMPLO- REMOÇÃO - III

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



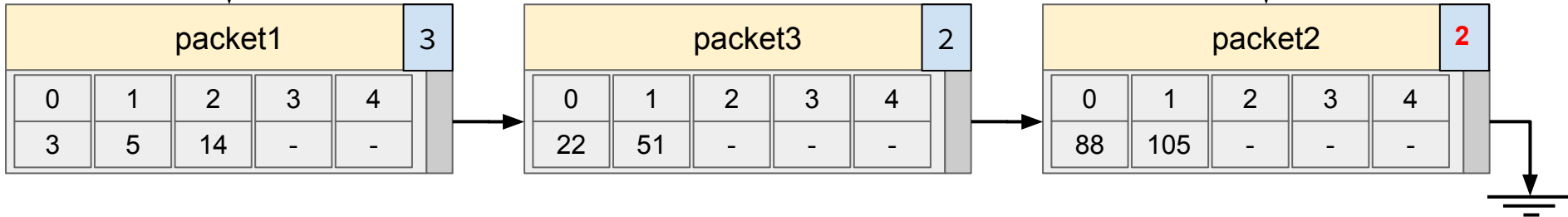
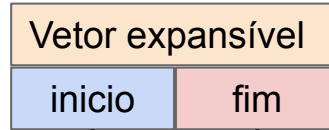
EXEMPLO- REMOÇÃO - IV

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



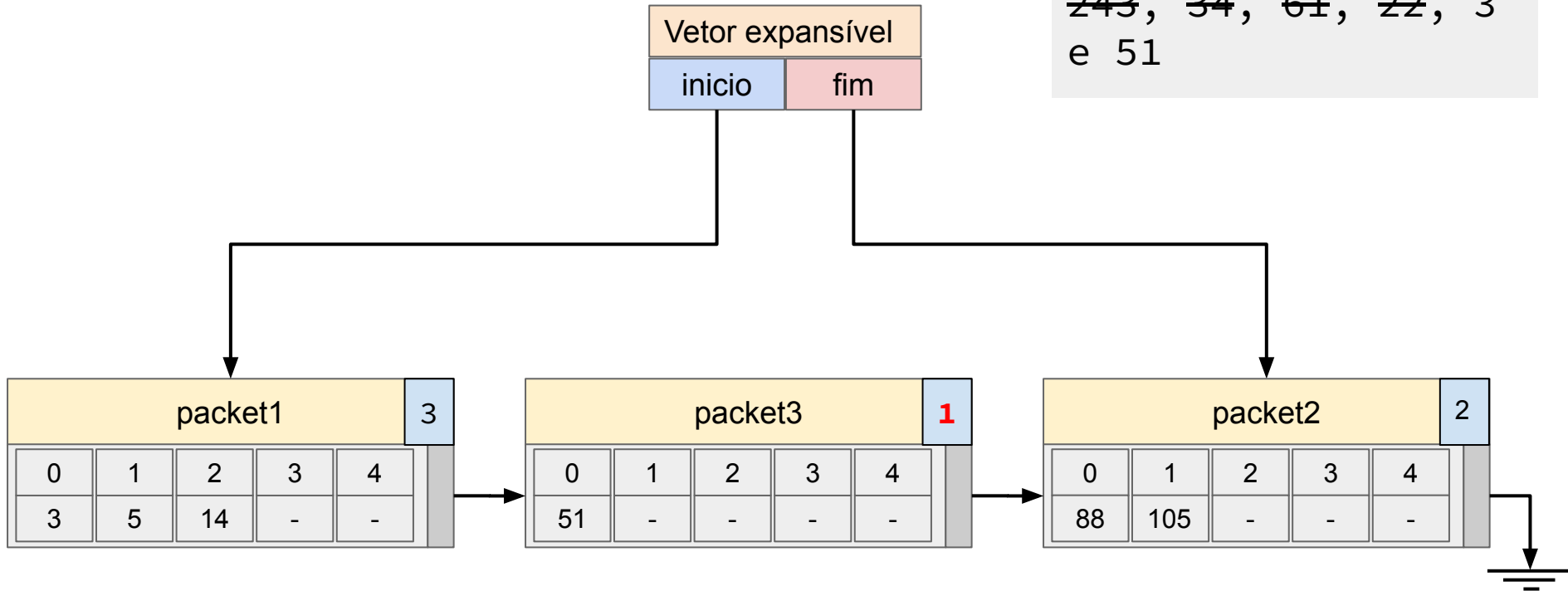
EXEMPLO- REMOÇÃO - V

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



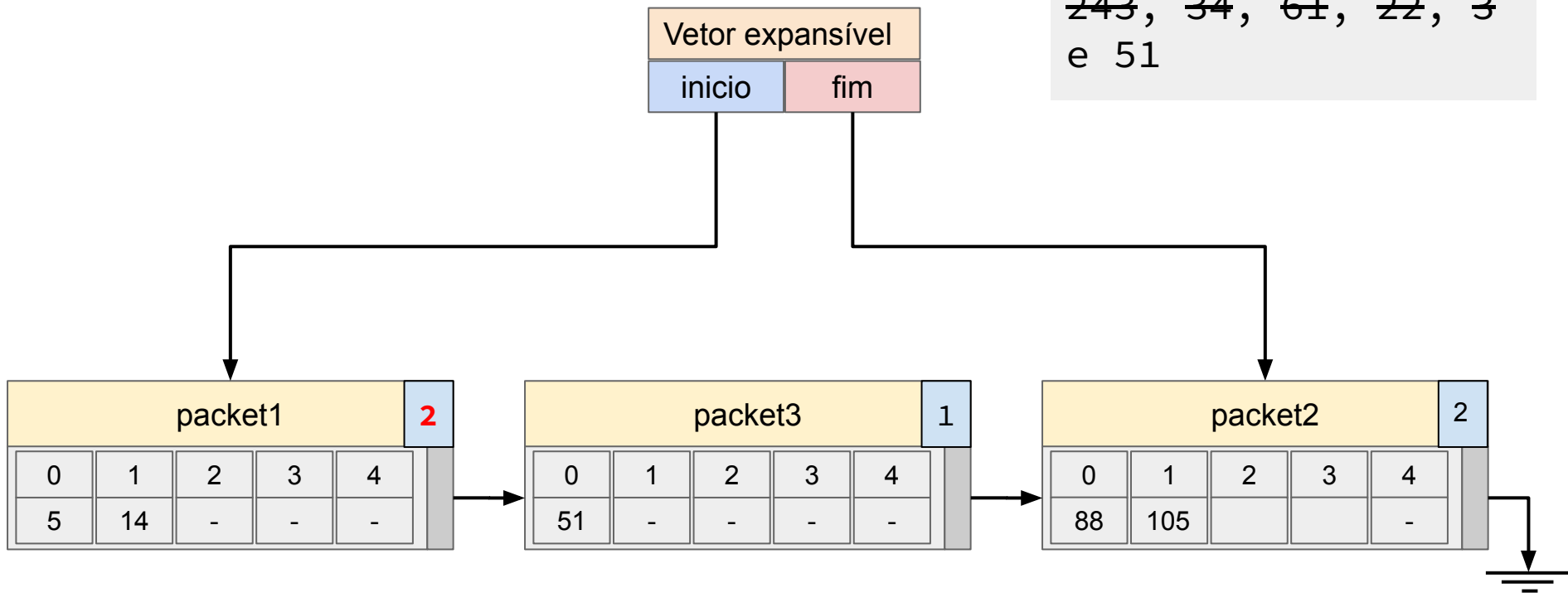
EXEMPLO- REMOÇÃO - VI

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



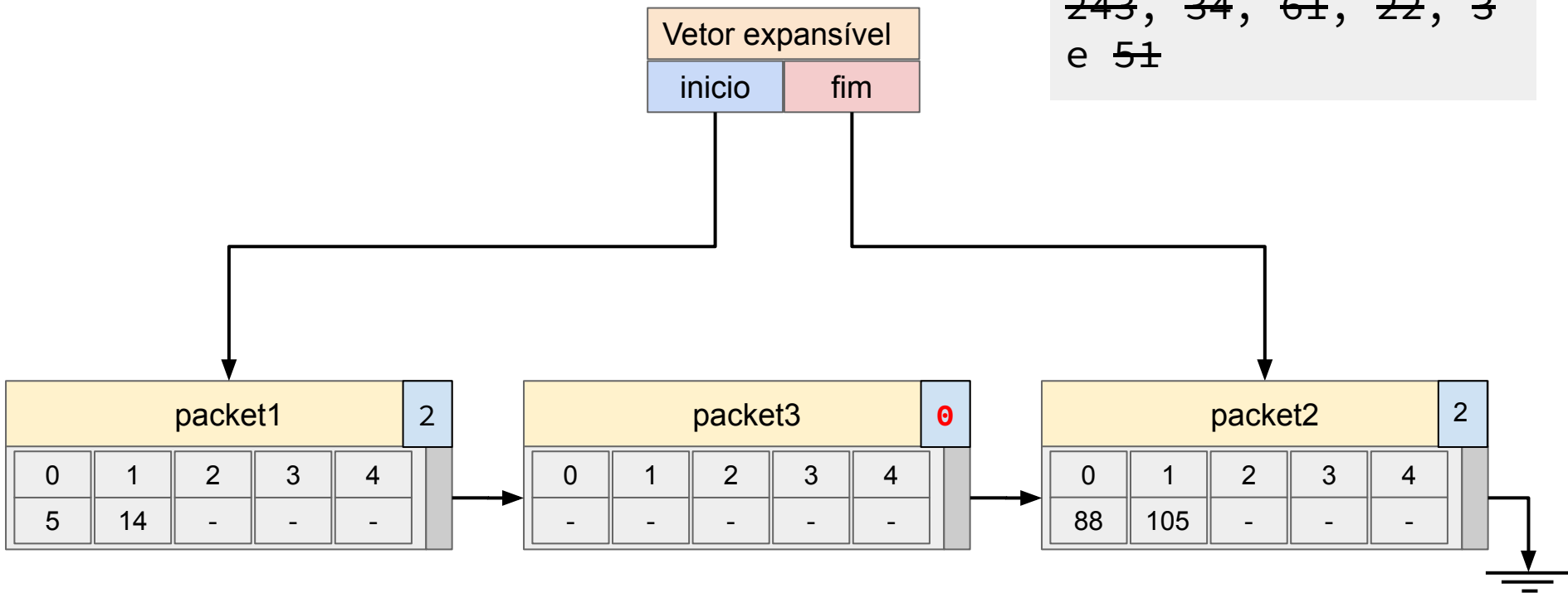
EXEMPLO- REMOÇÃO - VII

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



E agora,
José?

Vetor expansível

inicio

fim

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

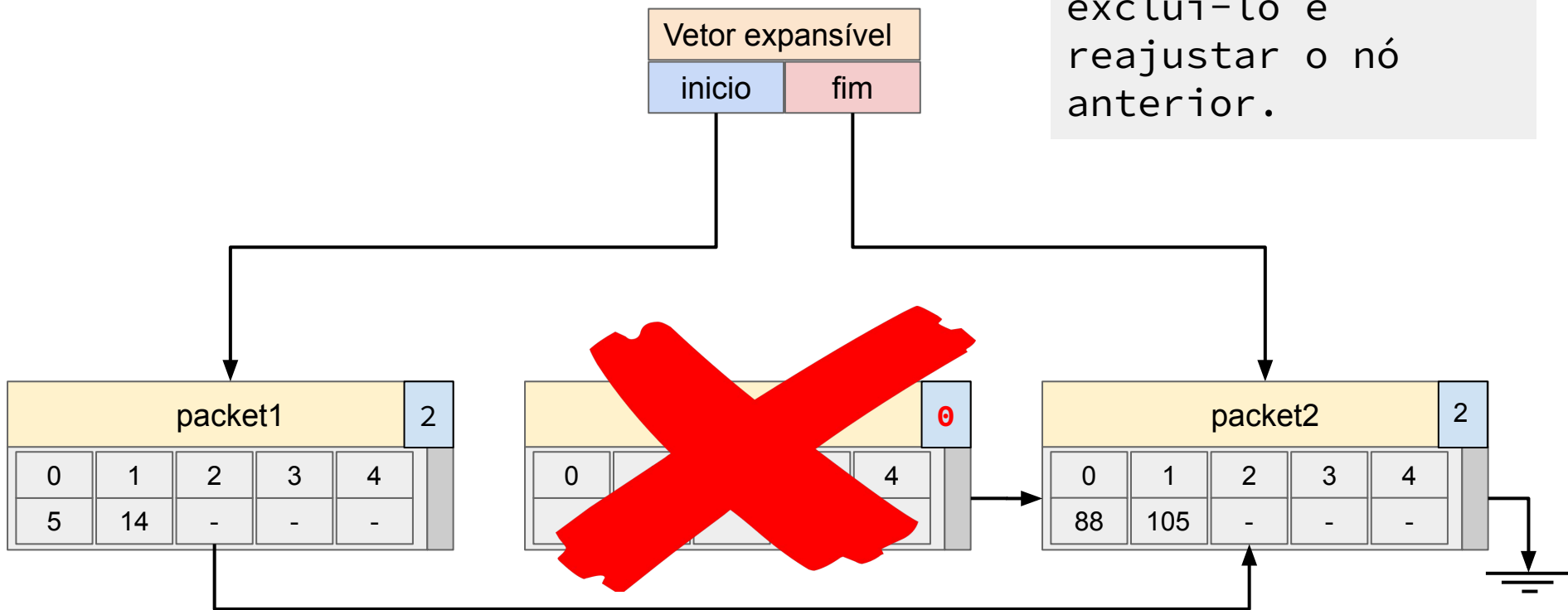
packet1					2
0	1	2	3	4	
5	14	-	-	-	

packet3					0
0	1	2	3	4	
-	-	-	-	-	

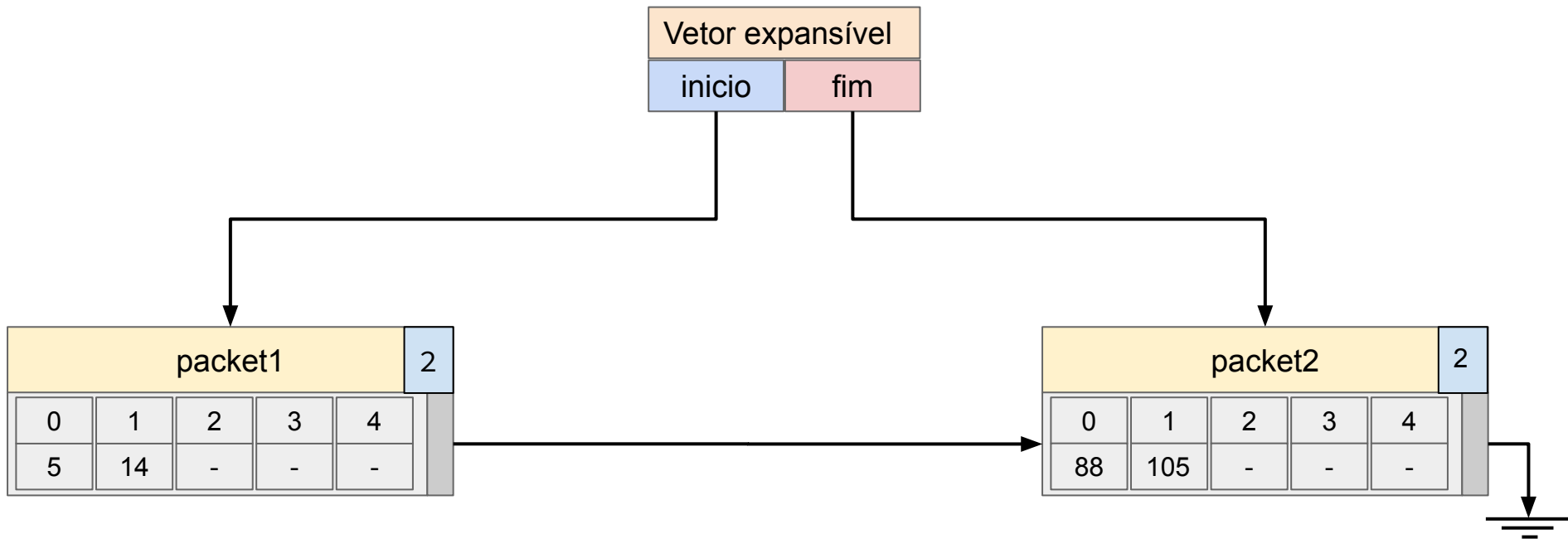
packet2					2
0	1	2	3	4	
88	105	-	-	-	

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?

Vetor expansível

inicio

fim

packet1

4

0	1	2	3	4
3	5	7	8	-

packet2

3

0	1	2	3	4
14	22	34	-	-

packet4

3

0	1	2	3	4
61	65	77	-	-

packet3

4

0	1	2	3	4
42	45	49	51	-

packet5

4

0	1	2	3	4
88	99	105	243	-

EXEMPLO- BUSCA - II

$65 > 8 \Rightarrow$
próximo pacote

Vetor expansível

inicio

fim

packet1

4

0	1	2	3	4
3	5	7	8	-

packet4

3

0	1	2	3	4
61	65	77	-	-

packet5

4

0	1	2	3	4
88	99	105	243	-

packet2

3

0	1	2	3	4
14	22	34	-	-

packet3

4

0	1	2	3	4
42	45	49	51	-

EXEMPLO- BUSCA - III

$65 > 34 \Rightarrow$
próximo pacote

Vetor expansível

inicio

fim

packet1

4

0	1	2	3	4
3	5	7	8	-

packet4

3

0	1	2	3	4
61	65	77	-	-

packet5

4

0	1	2	3	4
88	99	105	243	-

packet2

3

0	1	2	3	4
14	22	34	-	-

packet3

4

0	1	2	3	4
42	45	49	51	-

EXEMPLO- BUSCA - IV

65 > 51 ⇒
próximo pacote

Vetor expansível

inicio

fim

packet1

4

0	1	2	3	4
3	5	7	8	-

packet2

3

0	1	2	3	4
14	22	34	-	-

packet4

3

0	1	2	3	4
61	65	77	-	-

packet3

4

0	1	2	3	4
42	45	49	51	-

packet5

4

0	1	2	3	4
88	99	105	243	-

EXEMPLO- BUSCA - V

$65 \leq 77 \Rightarrow$
verificar pacote

Vetor expansível

inicio

fim

packet1

4

0	1	2	3	4
3	5	7	8	-

packet2

3

0	1	2	3	4
14	22	34	-	-

packet4

3

0	1	2	3	4
61	65	77	-	-

packet3

4

0	1	2	3	4
42	45	49	51	-

packet5

4

0	1	2	3	4
88	99	105	243	-

EXEMPLO- BUSCA - VI

Em cada pacote, usa-se busca binária

Vetor expansível

inicio

fim

packet1

4

0	1	2	3	4
3	5	7	8	-

packet4

3

0	1	2	3	4
61	65	77	-	-

packet5

4

0	1	2	3	4
88	99	105	243	-

packet2

3

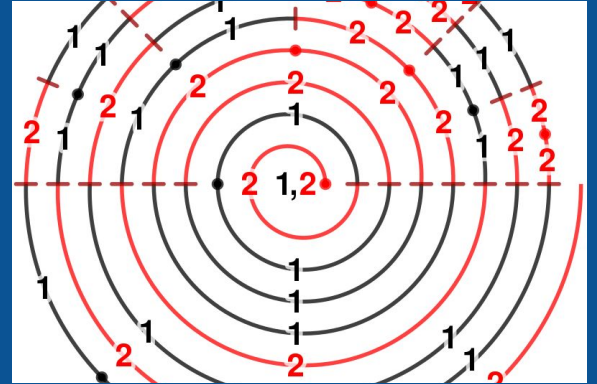
0	1	2	3	4
14	22	34	-	-

packet3

4

0	1	2	3	4
42	45	49	51	-

SEQUENCE SETS



SEQUENCE SETS - O PROBLEMA

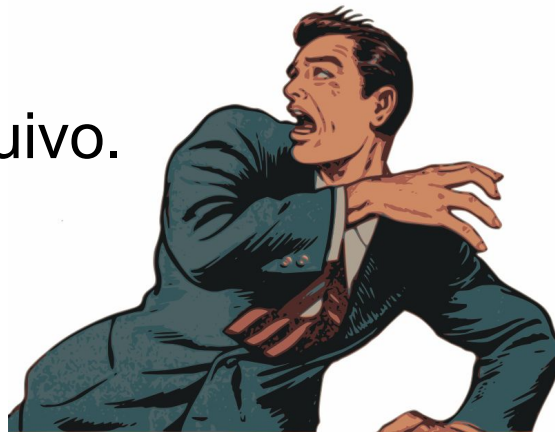
Trabalhar com arquivos binários estruturados geralmente é uma tarefa complexa, quando são necessárias várias modificações (inserções, remoções, alterações, etc.).

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
l ???? e

l ?? Se
c ??? ni

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

Um cabeç ????
que ele po

PARA TUDO!



????

o tamanho interme ??

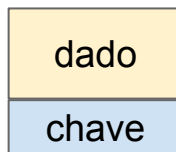
pedaços de taman ???

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

??
ori ??? 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:

dados	dados	dados	dados	dados	dados	dados	dados	dados	dados	dados	dados
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:

dado	dado	dado	dado
9	21	32	45
pedaço 0			

dado	dado	dado	dado
82	83	86	91
pedaço 1			

dado	dado	dado	dado
96	101	123	204
pedaço 2			

CALMA QUE EU EXPLICO - IV

dado	dado	dado	dado
9	21	32	45
pedaço 0			

dado	dado	dado	dado
82	83	86	91
pedaço 1			

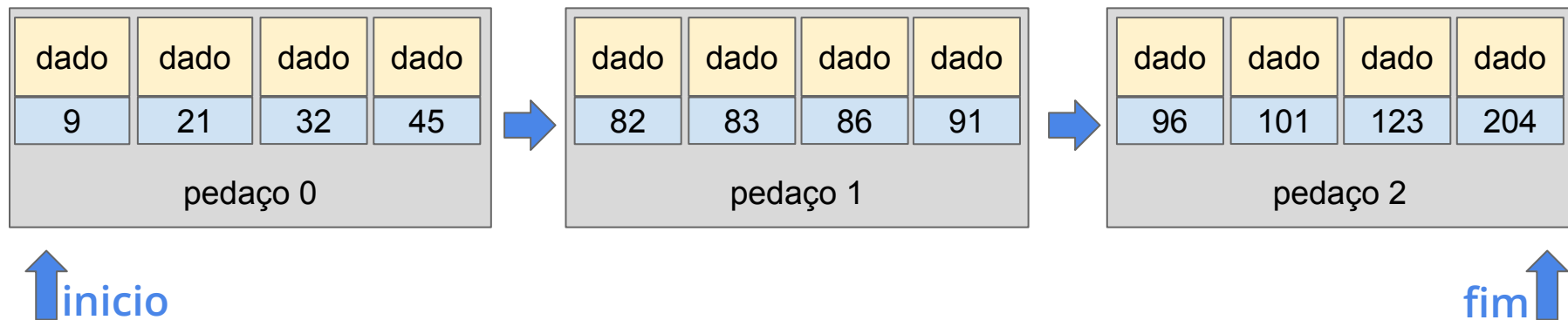
dado	dado	dado	dado
96	101	123	204
pedaço 2			

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:

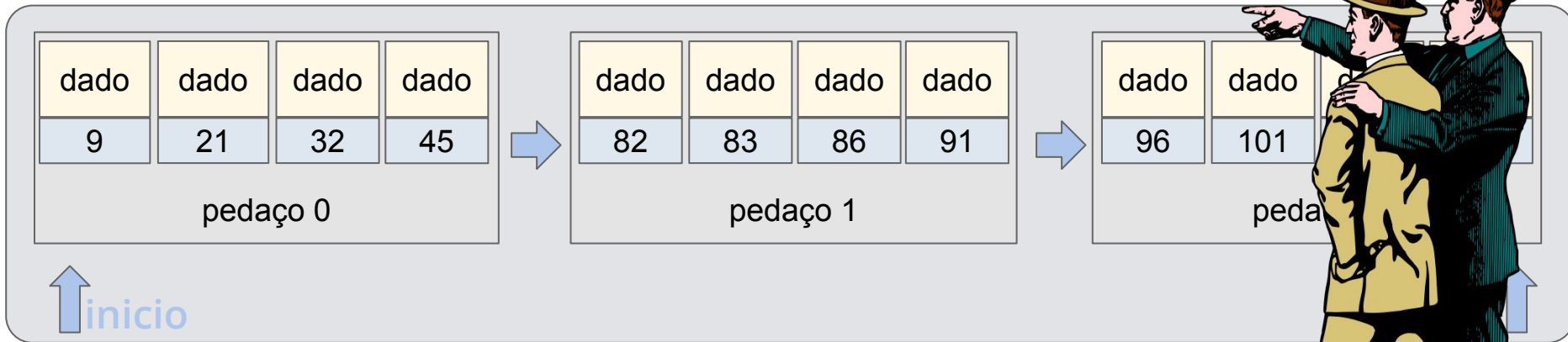


Isso é feito por meio de dois mecanismos: um cabeçalho no arquivo e um cabeçalho em cada pedaço.

CALMA QUE EU EXPLICO - V

Usaremos então uma indicação e cada pedaço apontará o próximo:

*Veja, um vetor expansível
disfarçado de arquivo binário!*



Isso é feito por meio de dois mecanismos: um cabeçalho 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:

dado	dado	dado	dado
9	21	32	45
pedaço 0			

dado	dado	dado	dado
82	83	86	91
pedaço 1			

dado	dado	dado	dado
96	101	123	204
pedaço 2			

CALMA QUE EU EXPLICO - VII

dado	dado	dado	dado
9	21	32	45
pedaço 0			

dado	dado	dado	dado
82	83	86	91
pedaço 1			

dado	dado	dado	dado
96	101	123	204
pedaço 2			

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

dado	dado	dado	dado
9	21	32	45
Qtd	4	Prox	1

pedaço 0

dado	dado	dado	dado
82	83	86	91
Qtd	4	Prox	2

pedaço 1

dado	dado	dado	dado
96	101	123	204
Qtd	4	Prox	-1

pedaço 2

nSeq	3
pSeq	0
Prox	3

cabeçalho

Número de
seqüências

Primeira
seqüência

Próximo
pedaço
disponível

Número de
registros
válidos

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.



Isso tudo é mesmo necessário???

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

PODE DESENHAR???

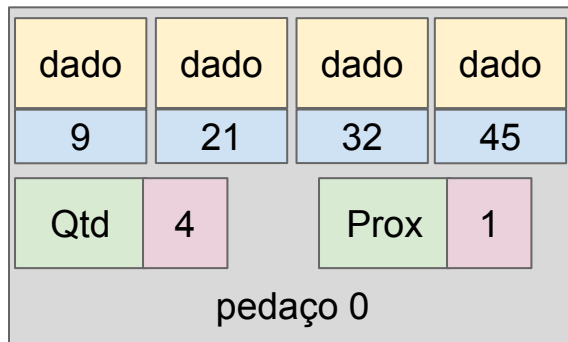
Imagine que serão realizadas as seguintes operações no Sequence Set de exemplo:

- ❖ 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 - I

1) Inserção do 13:

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

dado	dado	dado	dado
9	21	32	45
Qtd	4	Prox	1
pedaço 0			



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

dado	dado	dado	dado
9	13	21	
Qtd	3	Prox	3

pedaço 0

dado	dado	dado	dado
82	83	86	91
Qtd	4	Prox	2

pedaço 1

dado	dado	dado	dado
96	101	123	204
Qtd	4	Prox	-1

pedaço 2

nSeq	4
pSeq	0
Prox	4

cabeçalho

dado	dado	dado	dado
32	45		
Qtd	2	Prox	1

pedaço 3

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

dado	dado	dado	dado
9	13	21	
Qtd	3	Prox	3

pedaço 0

dado	dado	dado	dado
82	83	85	
Qtd	3	Prox	4

pedaço 1

dado	dado	dado	dado
96	101	123	204
Qtd	4	Prox	-1

pedaço 2

nSeq	5
pSeq	0
Prox	5

cabeçalho

dado	dado	dado	dado
32	45		
Qtd	2	Prox	1

pedaço 3

dado	dado	dado	dado
86	91		
Qtd	2	Prox	2

pedaço 4

OPERAÇÕES COM SEQUENCE SETS - IV

A inserção do 81 e do 80 serão relativamente tranquilas:



Quase sem alterações!!

OPERAÇÕES COM SEQUENCE SETS - IV

dado	dado	dado	dado
9	13	21	
Qtd	3	Prox	3

pedaço 0

dado	dado	dado	dado
82	83	85	
Qtd	3	Prox	4

pedaço 1

dado	dado	dado	dado
96	101	123	204
Qtd	4	Prox	-1

pedaço 2

nSeq	5
pSeq	0
Prox	5

cabeçalho

dado	dado	dado	dado
32	45	80	81
Qtd	4	Prox	1

pedaço 3

dado	dado	dado	dado
86	91		
Qtd	2	Prox	2

pedaço 4

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

dado	dado	dado	dado
3	9	13	21
Qtd	4	Prox	3

pedaço 0

dado	dado	dado	dado
82	83	85	
Qtd	3	Prox	4

pedaço 1

dado	dado	dado	dado
96	101	123	204
Qtd	4	Prox	-1

pedaço 2

nSeq	5
pSeq	0
Prox	5

cabeçalho

dado	dado	dado	dado
32	45	80	81
Qtd	4	Prox	1

pedaço 3

dado	dado	dado	dado
86	91		
Qtd	2	Prox	2

pedaço 4

OPERAÇÕES COM SEQUENCE SETS - VI

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

OPERAÇÕES COM SEQUENCE SETS - VI

dado	dado	dado	dado
3			
Qty	1	Prox	3
pedaço 0			

dado	dado	dado	dado
82	83	85	
Qty	3	Prox	4
pedaço 1			

dado	dado	dado	dado
96	101	123	204
Qty	4	Prox	-1
pedaço 2			

nSeq	5
pSeq	0
Prox	5
cabeçalho	

dado	dado	dado	dado
32	45	80	81
Qty	4	Prox	1
pedaço 3			

dado	dado	dado	dado
86	91		
Qty	2	Prox	2
pedaço 4			

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

dado	dado	dado	dado
Qty	0	Prox	5
pedaço 0			

dado	dado	dado	dado
82	83	85	
Qty	3	Prox	4
pedaço 1			

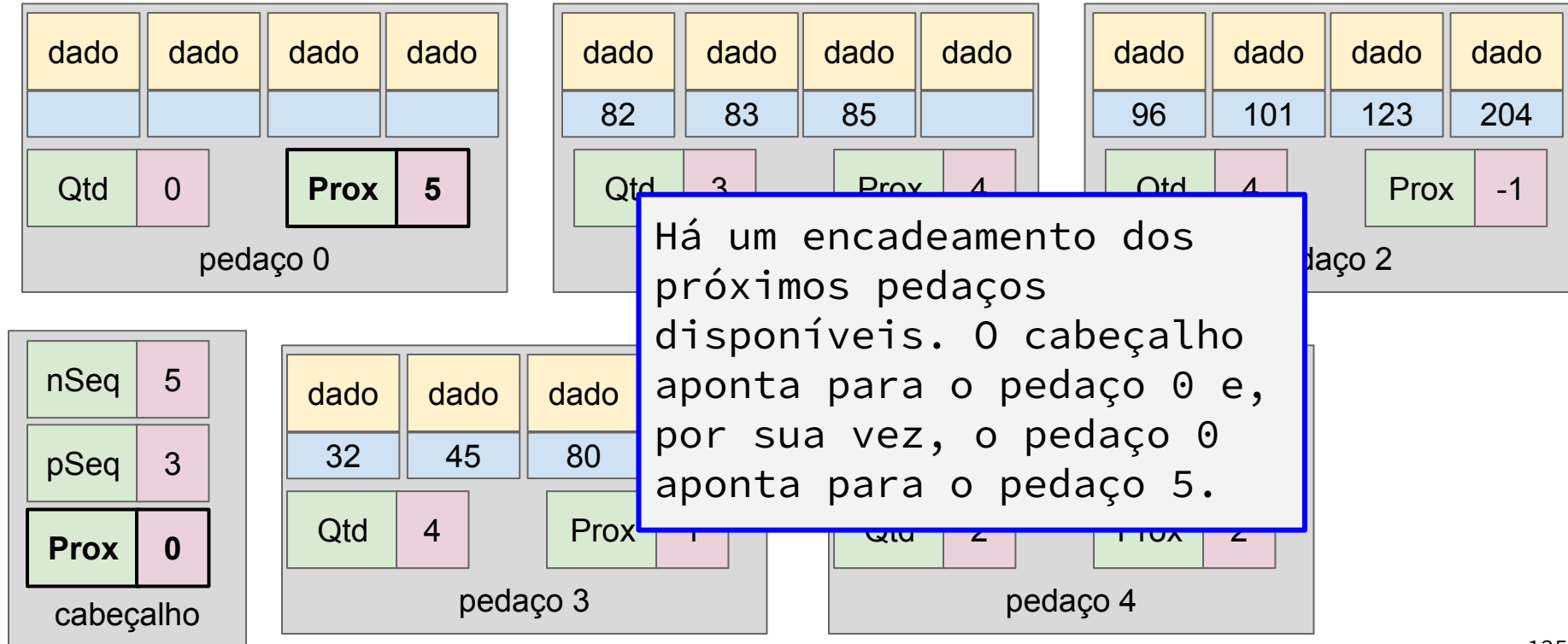
dado	dado	dado	dado
96	101	123	204
Qty	4	Prox	-1
pedaço 2			

nSeq	5
pSeq	3
Prox	0
cabeçalho	

dado	dado	dado	dado
32	45	80	81
Qty	4	Prox	1
pedaço 3			

dado	dado	dado	dado
86	91		
Qty	2	Prox	2
pedaço 4			

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

dado	dado	dado	dado
80	81		
Qty	2	Prox	1
pedaço 0			

dado	dado	dado	dado
82	83	85	
Qty	3	Prox	4
pedaço 1			

dado	dado	dado	dado
96	101	123	204
Qty	4	Prox	-1
pedaço 2			

nSeq	5
pSeq	3
Prox	0
cabeçalho	

dado	dado	dado	dado
5	32	45	
Qty	3	Prox	0
pedaço 3			

dado	dado	dado	dado
86	91		
Qty	2	Prox	2
pedaço 4			

OPERAÇÕES COM SEQUENCE SETS - IX

A inserção do 6 será muito tranquila:

OPERAÇÕES COM SEQUENCE SETS - IX

dado	dado	dado	dado
80	81		
Qty	2	Prox	1
pedaço 0			

dado	dado	dado	dado
82	83	85	
Qty	3	Prox	4
pedaço 1			

dado	dado	dado	dado
96	101	123	204
Qty	4	Prox	-1
pedaço 2			

nSeq	5
pSeq	3
Prox	0
cabeçalho	

dado	dado	dado	dado
5	6	32	45
Qty	4	Prox	0
pedaço 3			

dado	dado	dado	dado
86	91		
Qty	2	Prox	2
pedaço 4			

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 Gregghi
- Renato Ramos da Silva

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

Esta obra está licenciado com uma Licença [Creative Commons Atribuição 4.0 Internacional](https://creativecommons.org/licenses/by/4.0/).