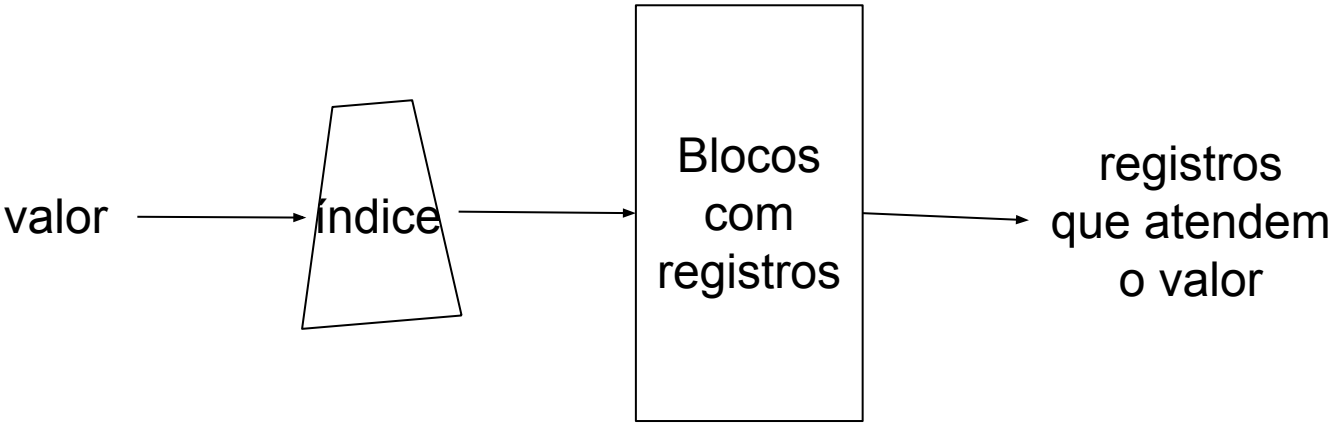
Índices

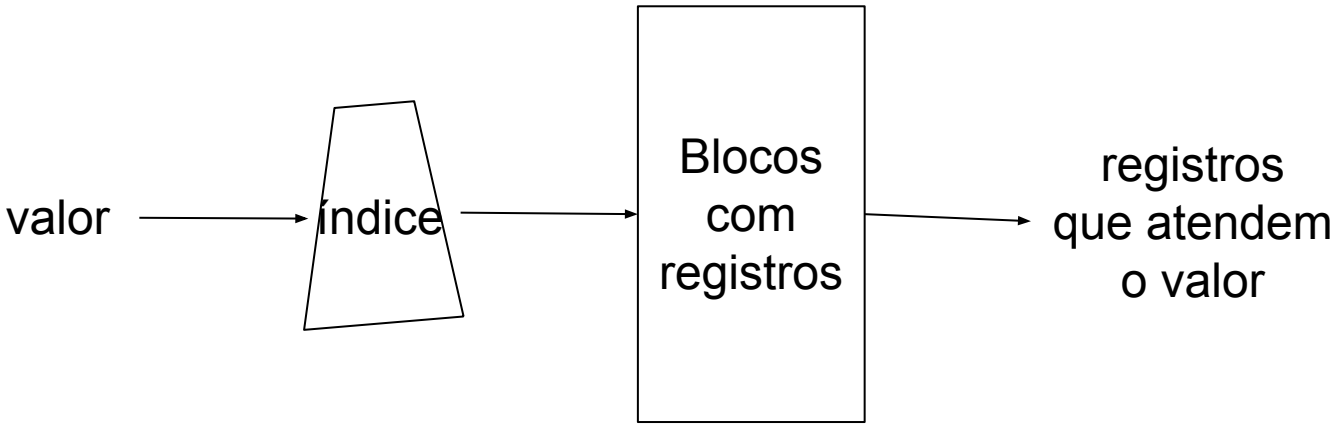
- Estrutura auxiliar projetada para agilizar operações de busca, inserção e supressão



- Alteração nos dados pode levar à alteração no índice
- Espaço extra de armazenamento

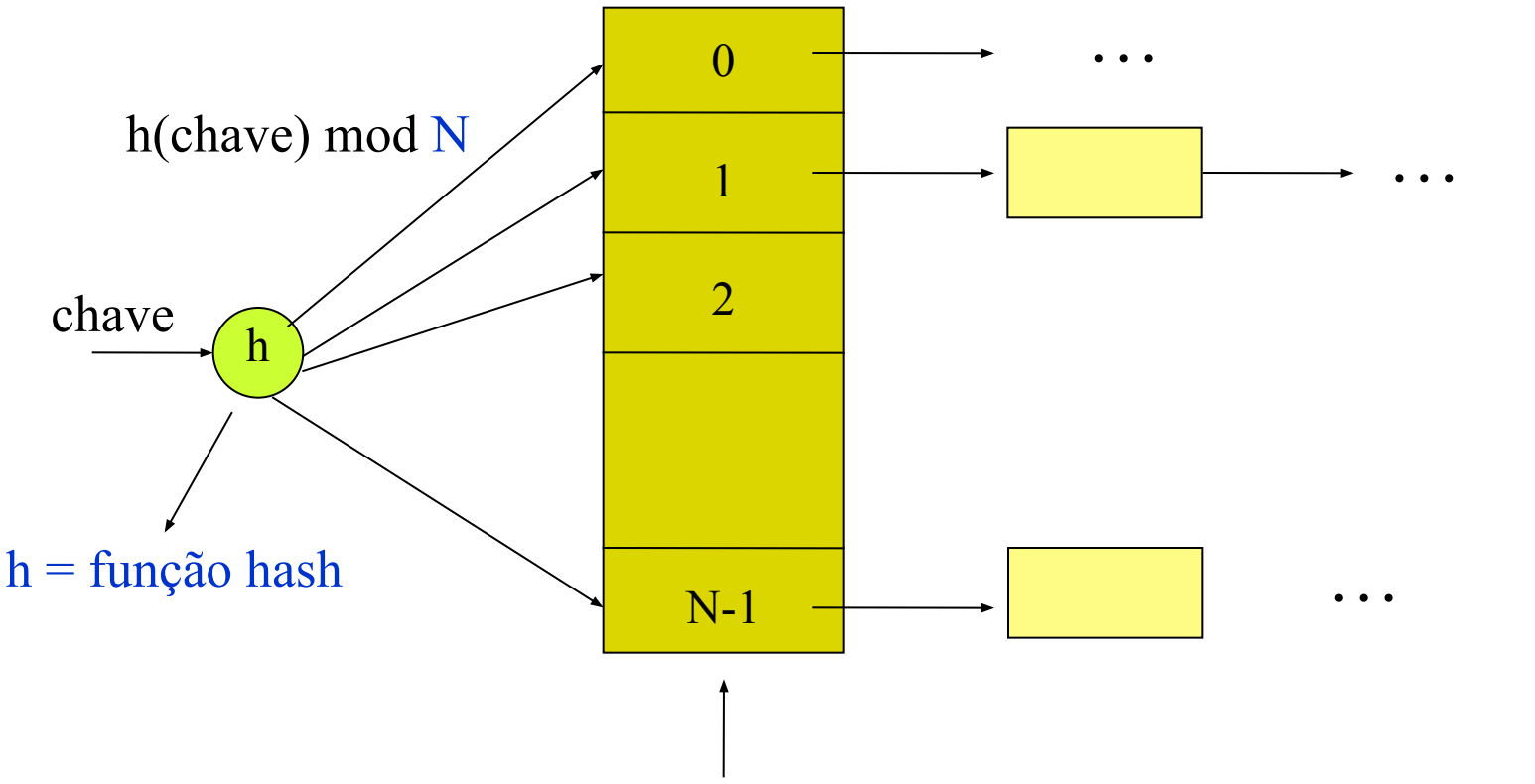
Índices

- Estrutura auxiliar projetada para agilizar operações de busca, inserção e supressão



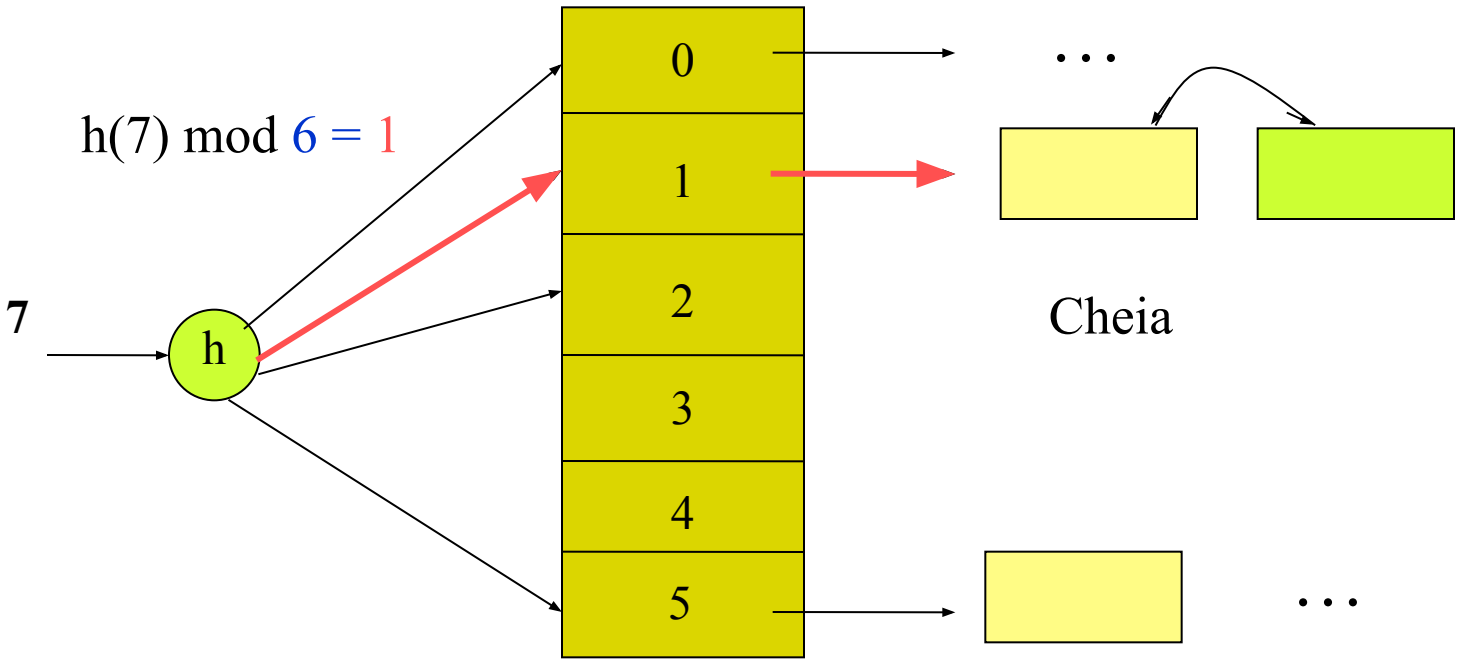
- CREATE INDEX name ON table USING hash(att);
- CREATE INDEX name ON table USING btree(att);

Hash Estático

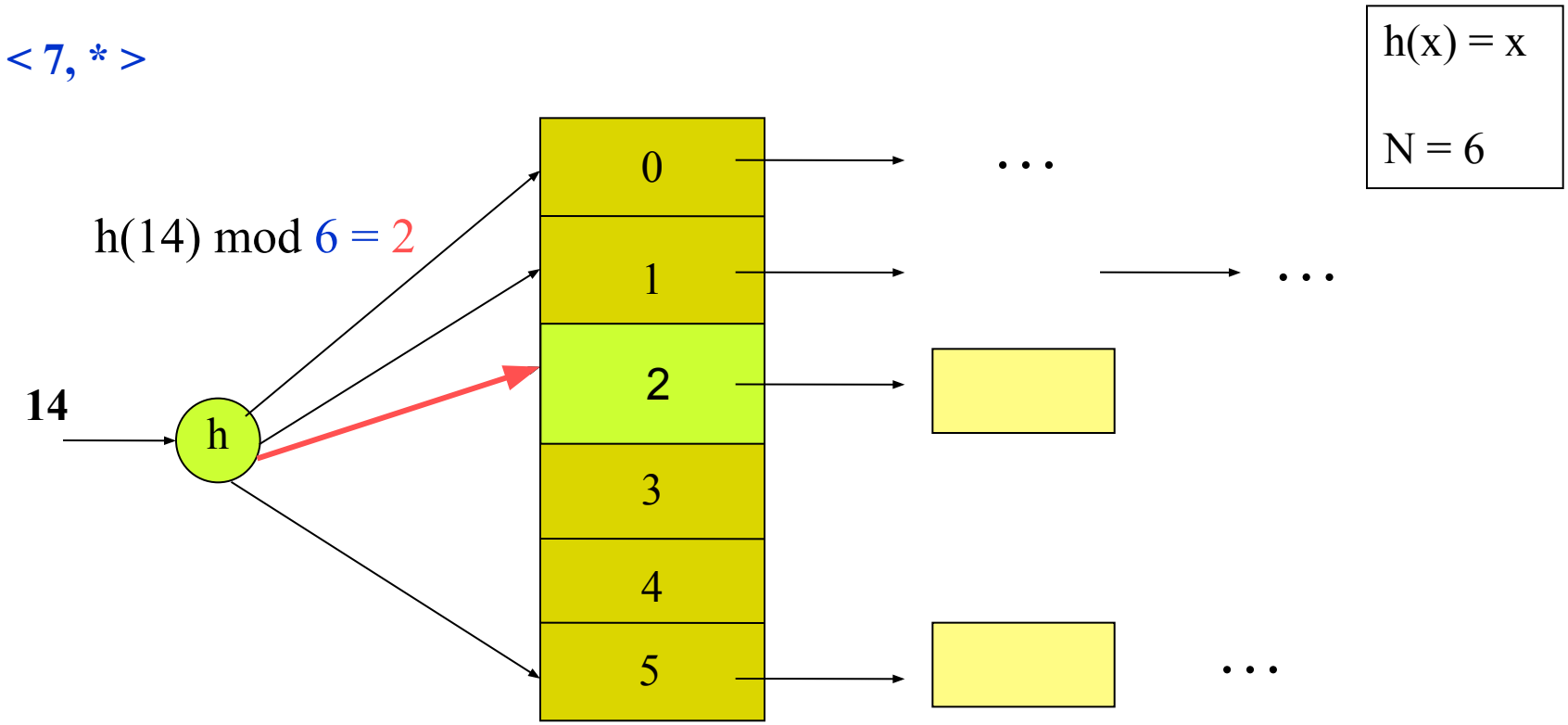


Inserção

Inserindo < 7, * >



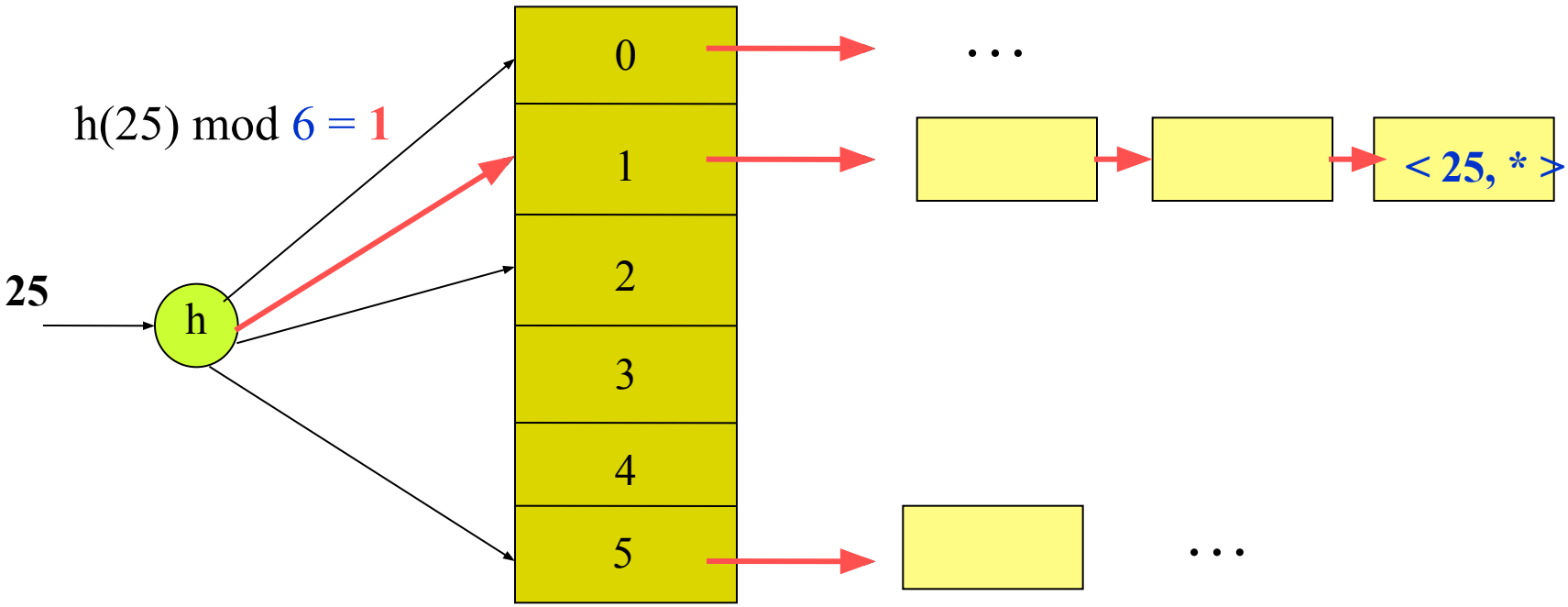
Busca



Página primárias =6

Inserção

Inserindo < 25, * >



Função Hash

Componente importante da técnica hash

Deve distribuir valores das chaves de maneira uniforme nos *buckets*.

Número de buckets = N = parâmetro

Custos

Páginas primárias podem ser armazenadas em páginas de disco sucessivas.

Caso não haja overflow

- Busca requer **1** I/O
- Inserção e Supressão requerem **2** I/O

Custo pode ser alto se existem muitas páginas de overflow.

Estratégia : criar inicialmente os buckets deixando 20% de área livre em cada um.

Atividade

- Construa uma estrutura de hash estático para a lista de valores abaixo.Considere que o bucket tenha espaço para dois valores, e que o número de buckets seja 8.

  Lista: 9, 101, 14, 18, 800, 3, 1, 7, 35, 92.

Desvantagens do Hash Estático

- Número de buckets é fixo
- Se o arquivo encolhe muito, o espaço é desperdiçado, já que os buckets são fixos.
- Crescimento do arquivo produz longas cadeias de páginas de overflow, prejudicando a performance da busca.

Alternativas

Alternativa 1 :

- Periodicamente, modificar a função hash e reestruturar todo o arquivo de modo a evitar páginas de overflow.
- « Rehash » toma muito tempo
- Índice não pode ser utilizado durante o processo de « rehash ».

Alternativa 2 : Hash **dinâmicos**

- Extensível
- Linear

Hash Extensível

Quando algum *bucket* ficar cheio

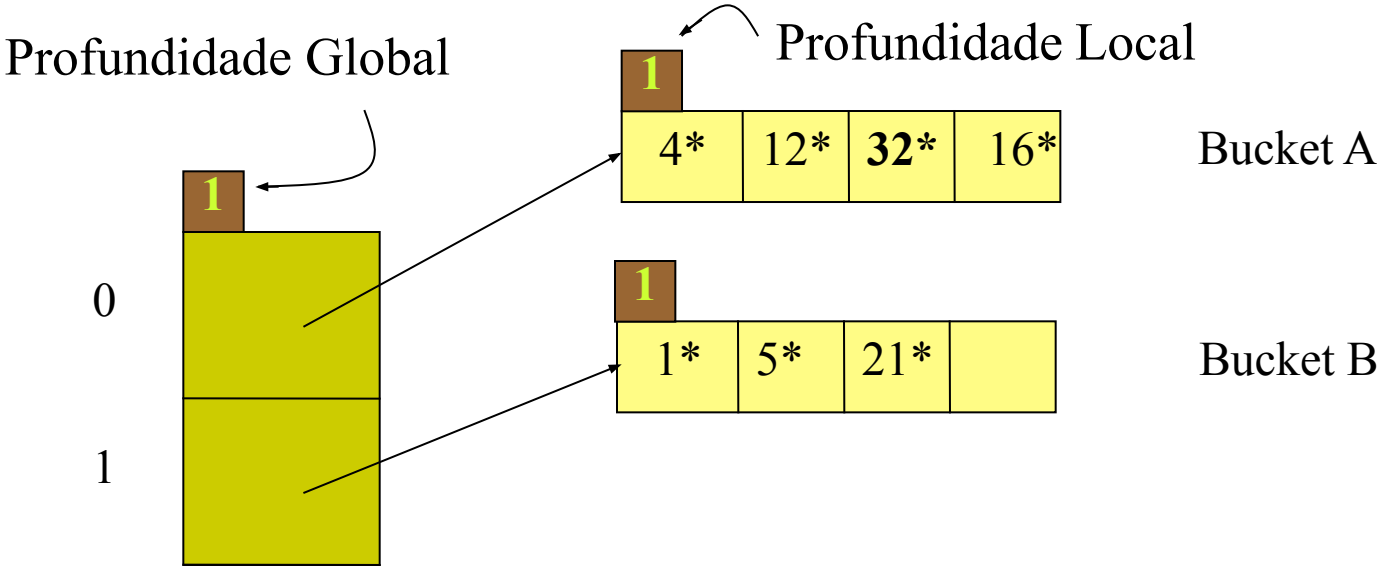
- Dobrar o número de ***buckets***
- Distribuir as entradas nos novos *buckets*

Notação-Hash Extensível

Para procurar, inserir ou suprimir entrada k*

- Aplicar h(k)
- h(k) identifica um bucket
- Duas chaves k1 e k2 podem ter h(k1) = h(k2)

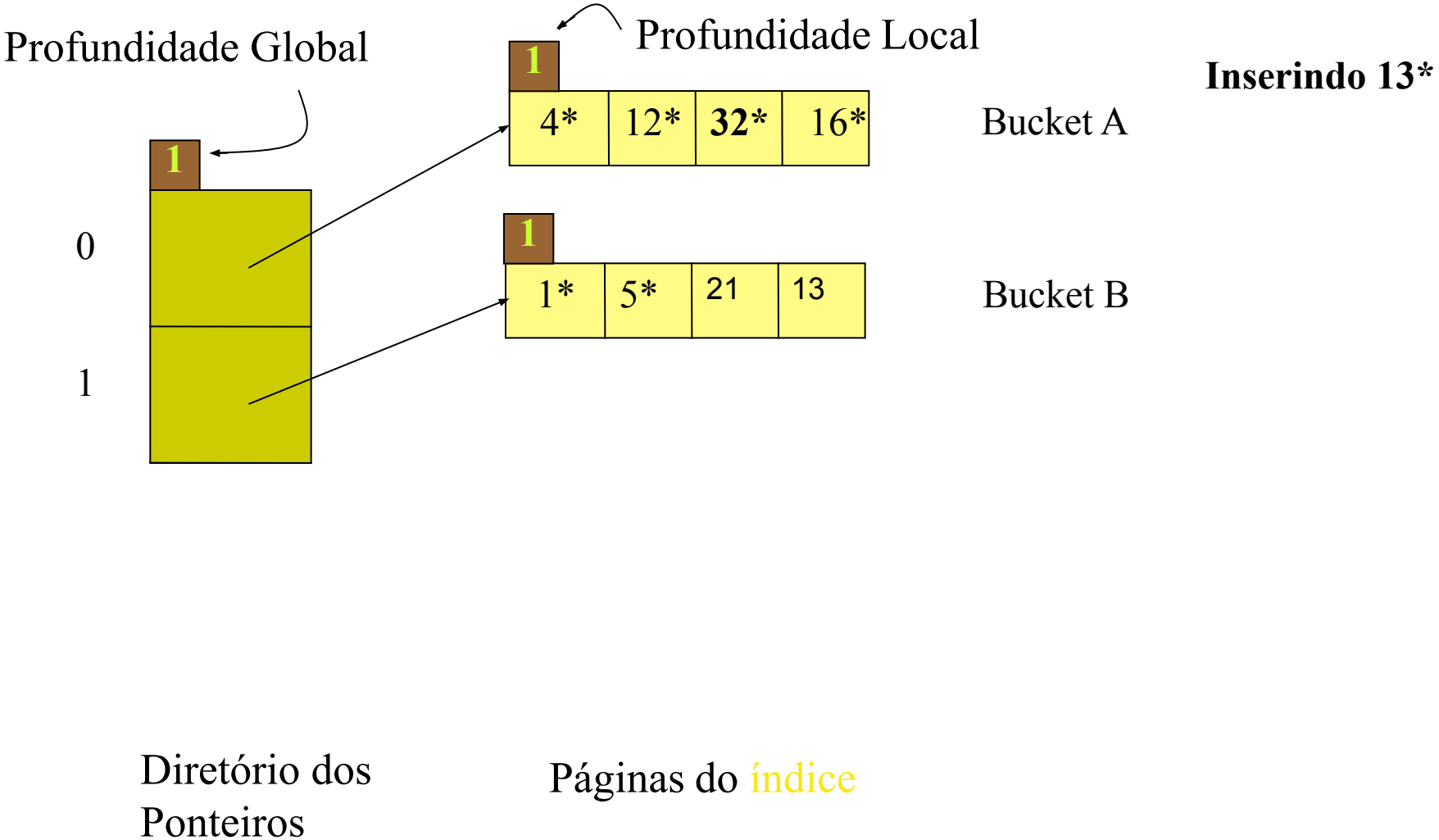
Exemplo-Hash Extensível



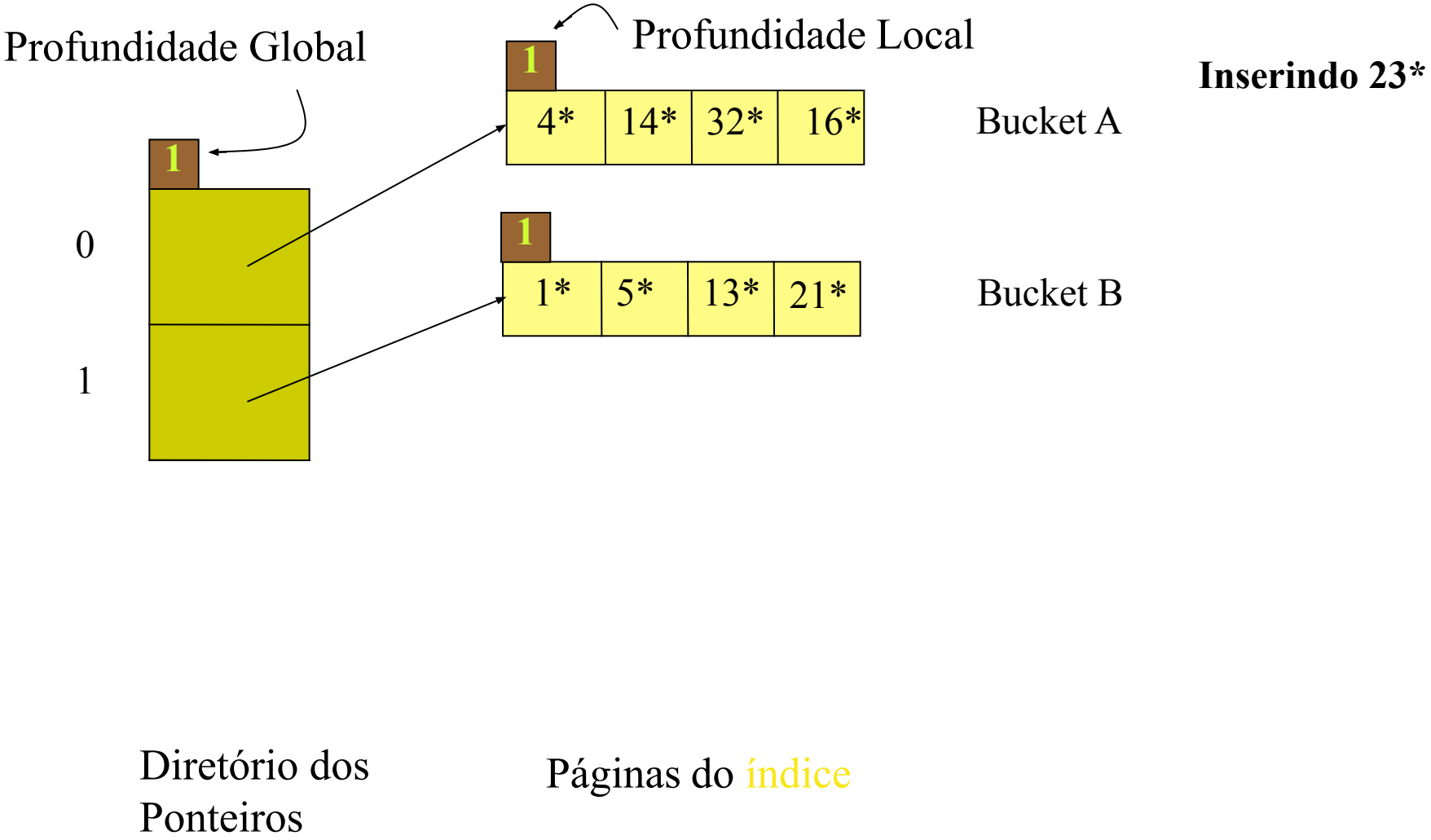
Diretório dos Ponteiros

Páginas do índice

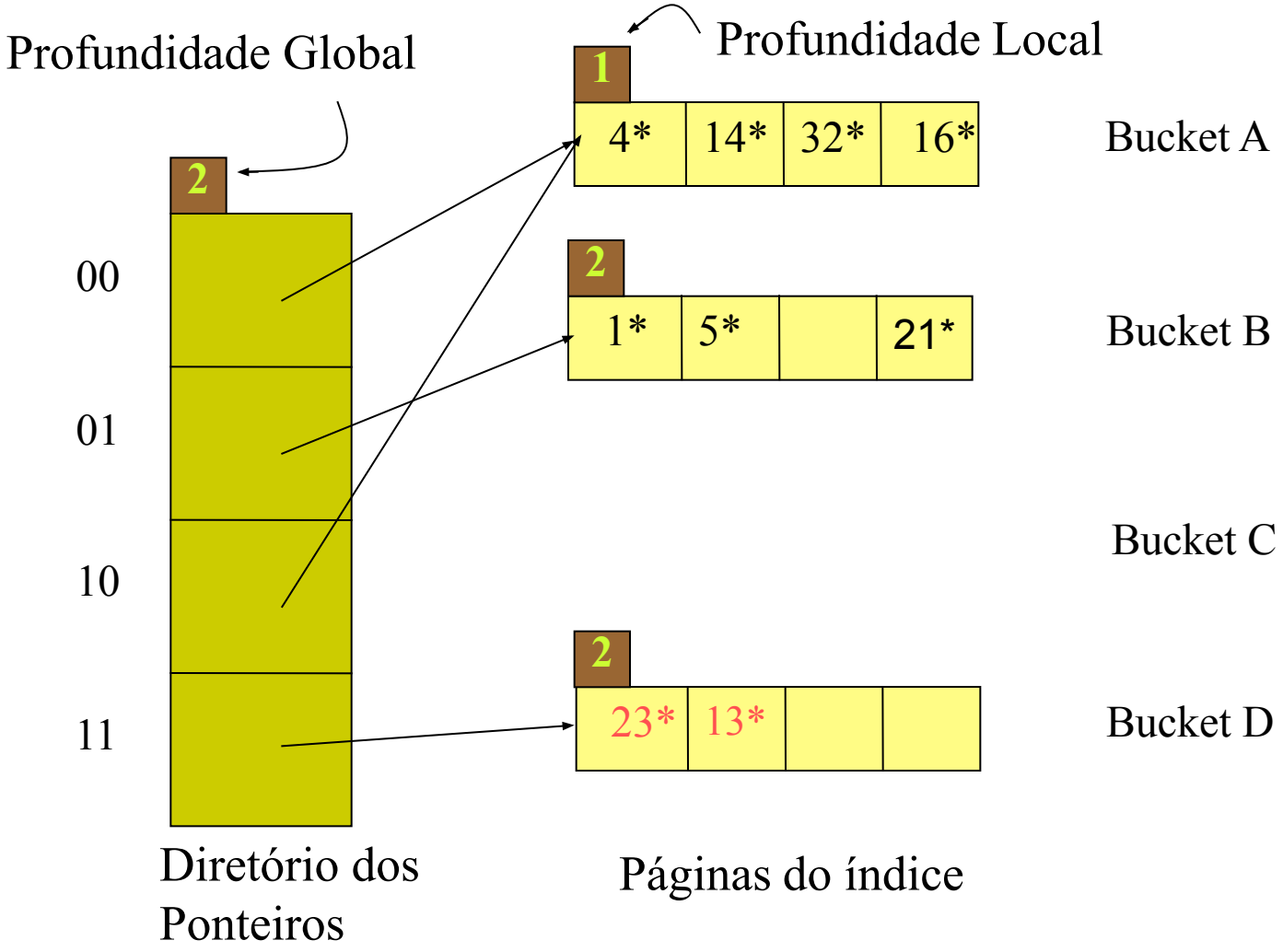
Exemplo-Hash Extensível



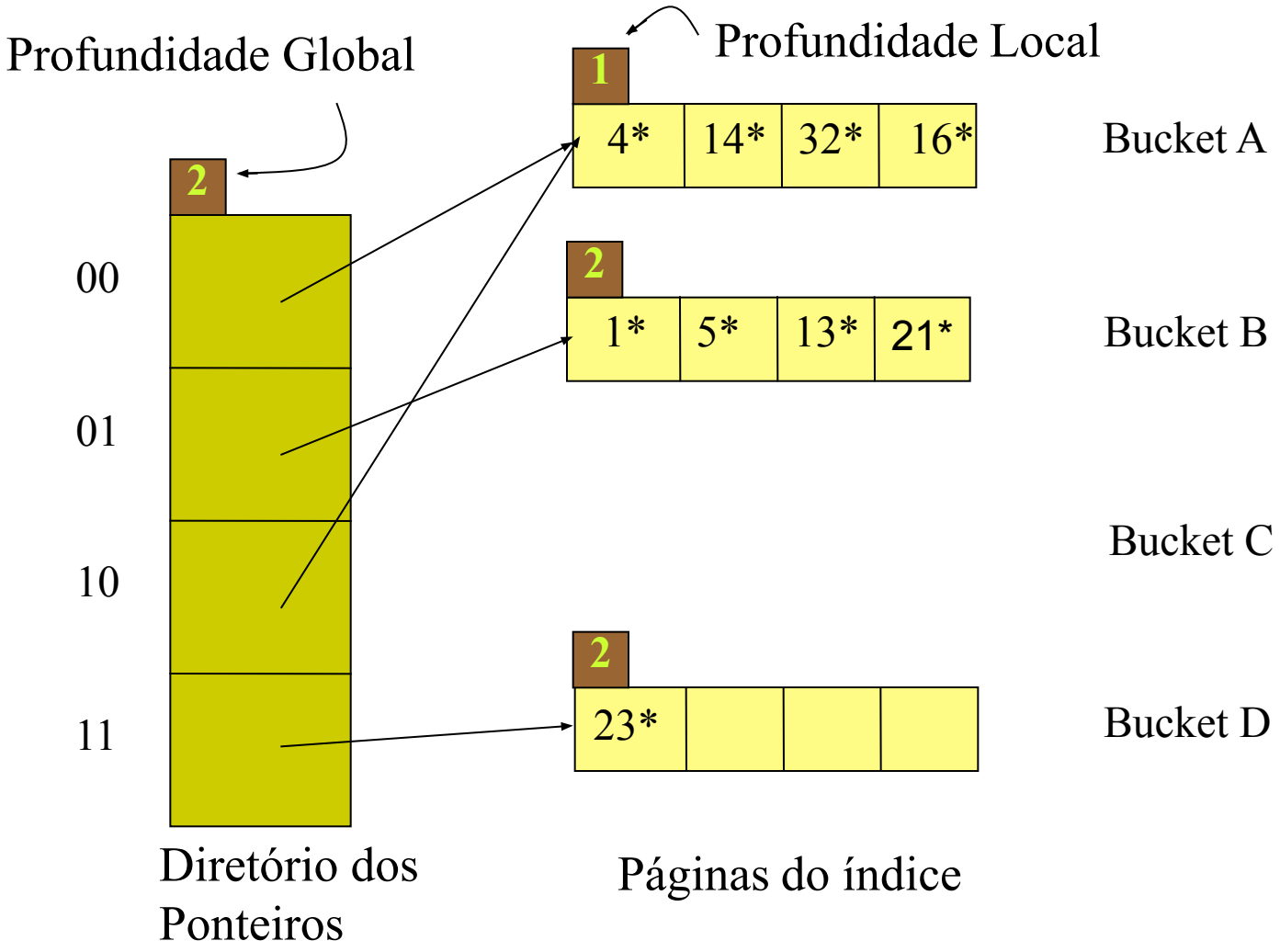
Exemplo-Hash Extensível



Exemplo-Hash Extensível



Exemplo-Hash Extensível

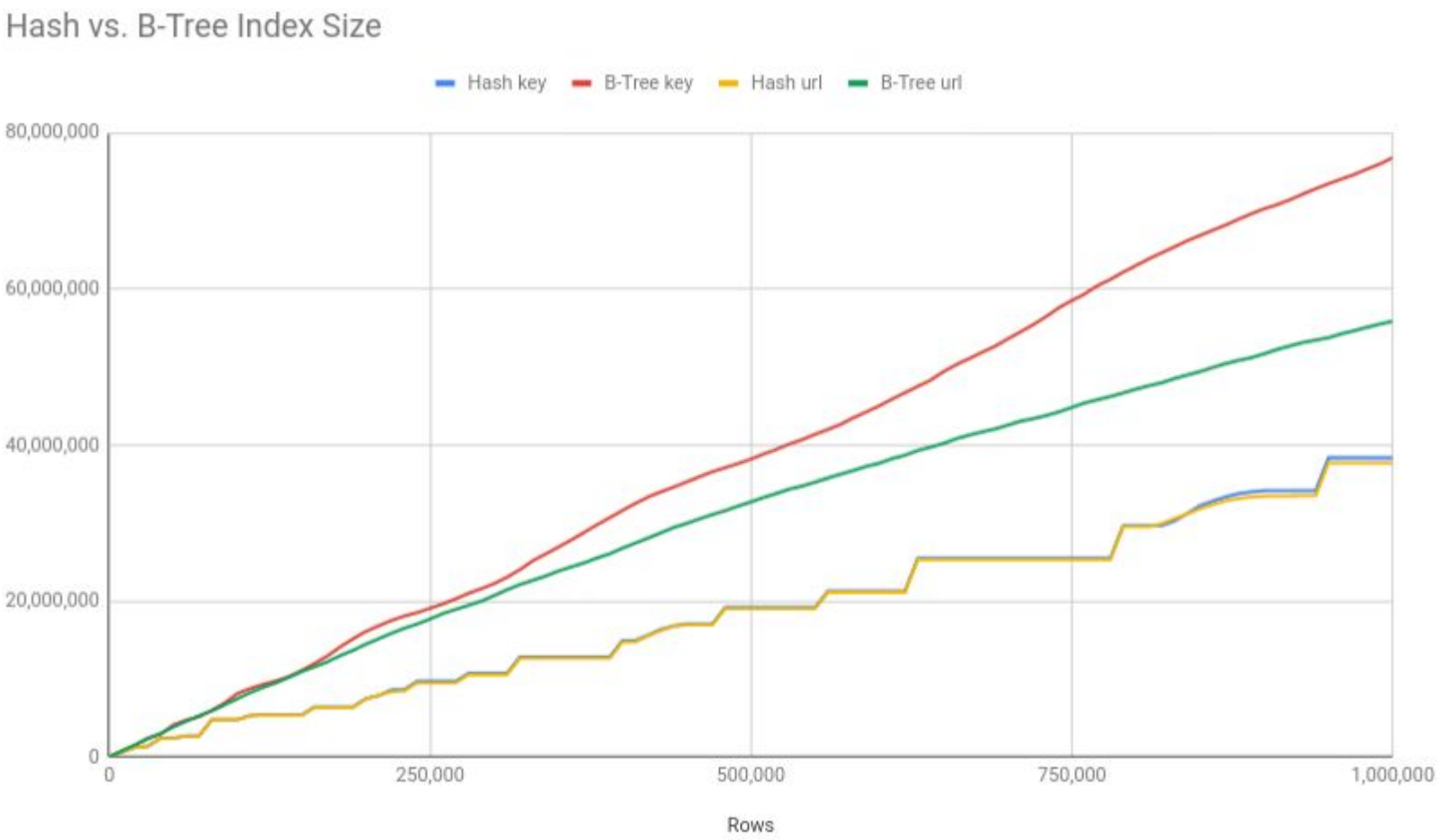


Análise

- Se o diretório couber na memória
  - Seleção com igualdade : **1** I/0

- Se o diretório tiver que ser armazenado em disco
  - Seleção com igualdade : **2** I/0

Custos



https://hakibenita.com/postgresql-hash-index

Atividade 1

1- Construa uma estrutura de hash extensível para a lista de valores abaixo.

Considere que o bucket tenha espaço para dois valores, e que a função de hash usada seja (valor f= N mod 4).

Lista: 9, 101, 14, 18, 800, 3, 1, 7, 35, 92,102,103,101,20,30

Atividade 2

Construa uma estrutura de hash extensível para a lista de valores abaixo.

Considere que o bucket tenha espaço para dois valores, e que a função de hash usada seja (valor f= N mod 2).

Lista: 1,2,3,4,5,7,9,11,13,15,17,19,21,23

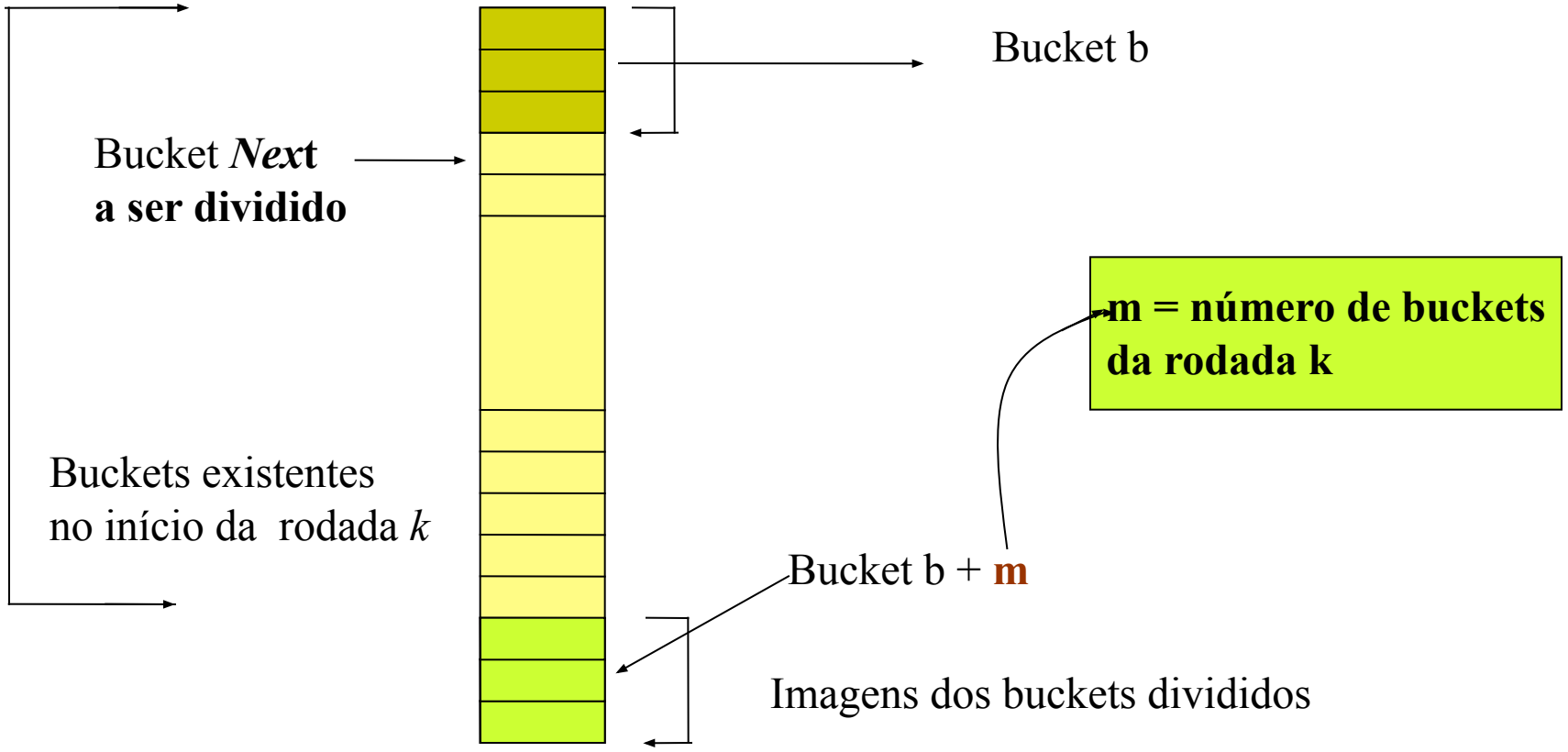
Hash Linear

- ***Nível*** = indica a rodada atual
  - Inicializado com 0
- ***Next*** = bucket que deve ser dividido, **se necessário**
  - Inicializado com 0
- N_m = número de buckets na rodada m

Somente o bucket com número ***Next*** é dividido.

- Usa-se páginas de overflow para os outros buckets, se ficarem cheios.
- Após divisão, ***Next*** é incrementado.

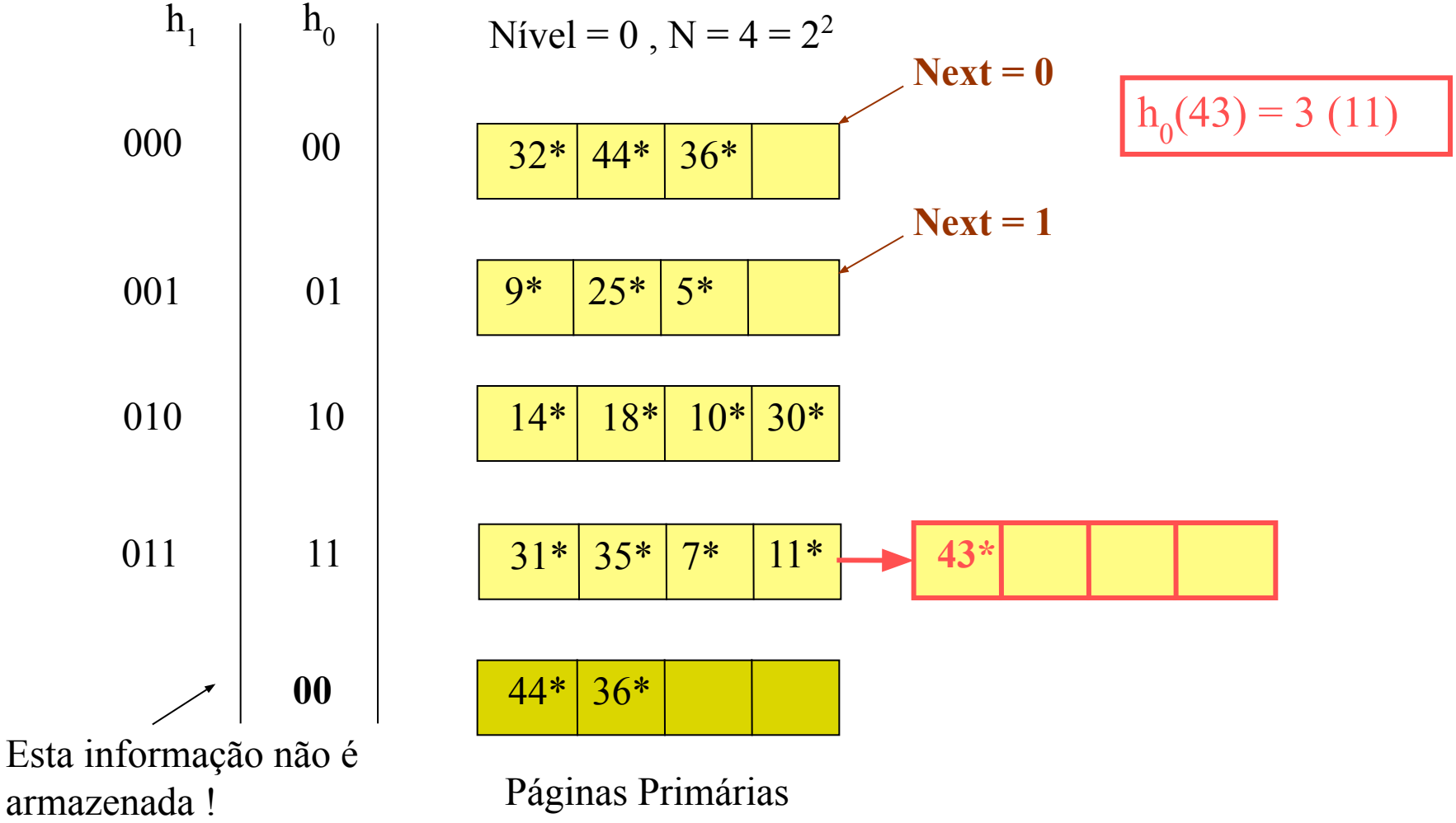
Esquema Geral : rodada *k*



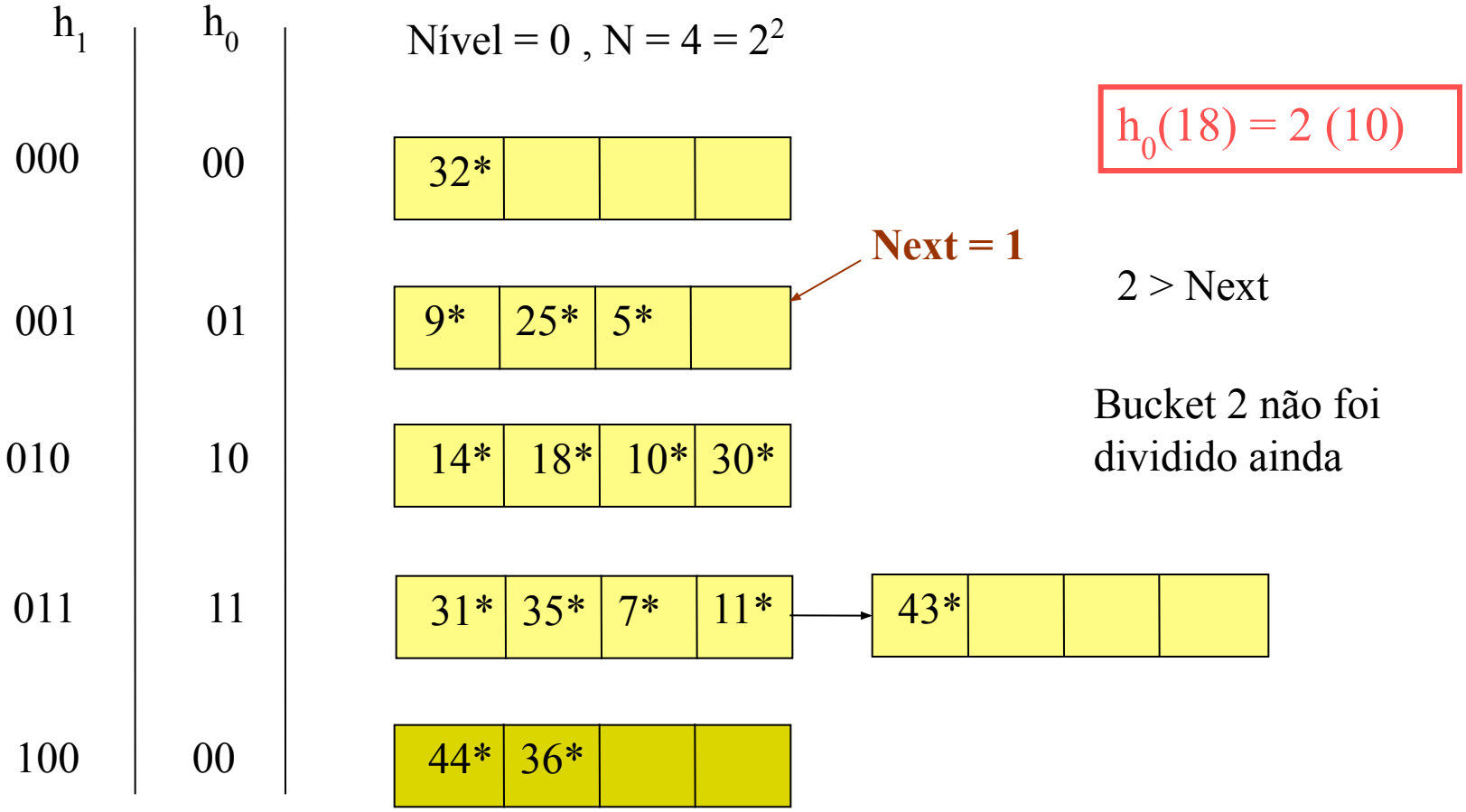
Hash Linear

- Assim como o Hash Extensível, o Hash Linear é ideal para inserções e supressões;
- Vantagem sobre extensível
  - Lida muito bem com colisões
  - Oferece muita flexibilidade
- Cadeias de overflow podem tornar o hash linear inferior em performance ao hash extensível

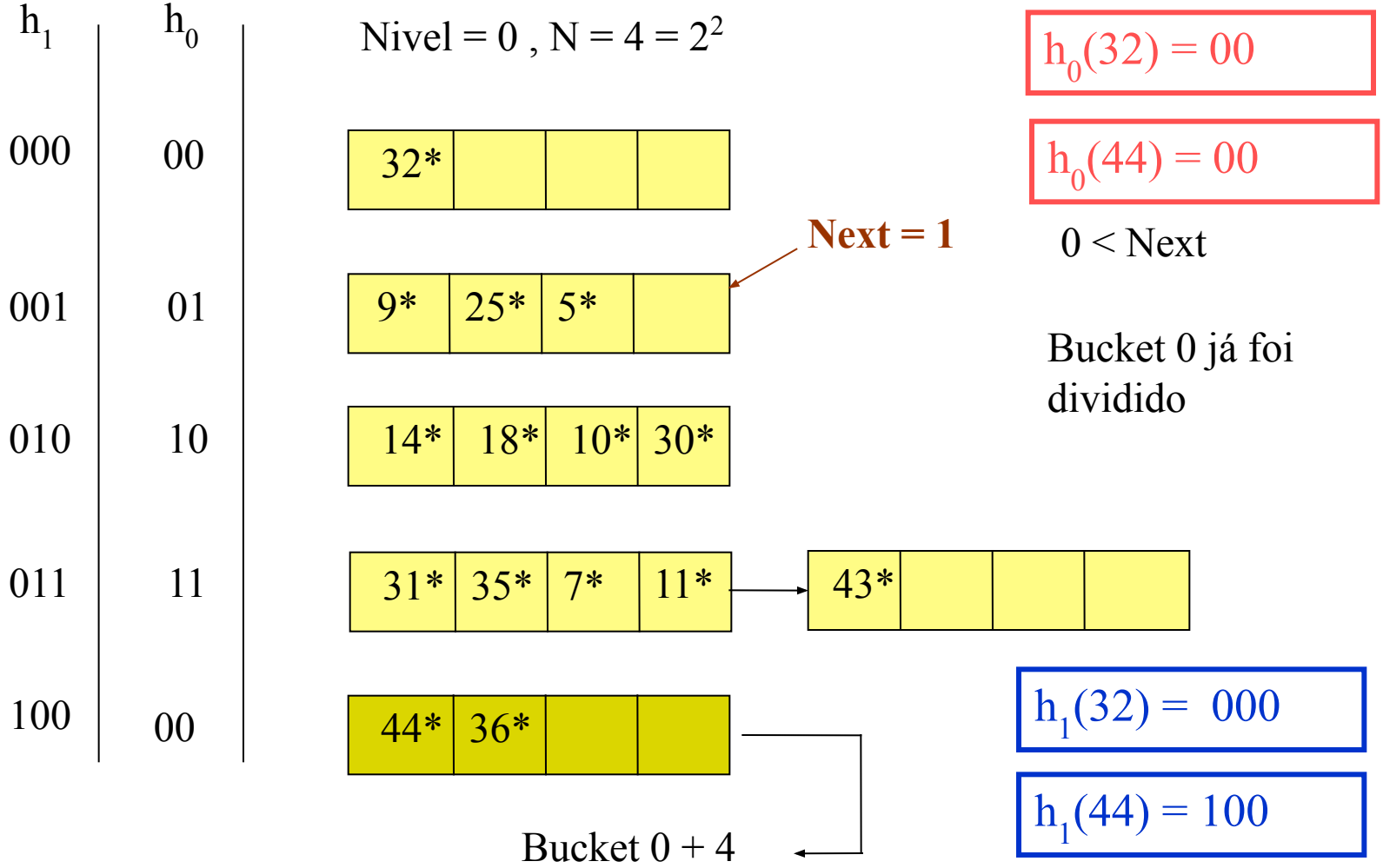
Inserção 43*


h1
h0
Nível = 0 , N = 4 = 2^2
Next = 0
h0(43) = 3 (11)
000
00
32*
44*
36*
Next = 1
001
01
9*
25*
5*
010
10
14*
18*
10*
30*
011
11
31*
35*
7*
11*
43*
00
44*
36*
Esta informação não é armazenada !
Páginas Primárias

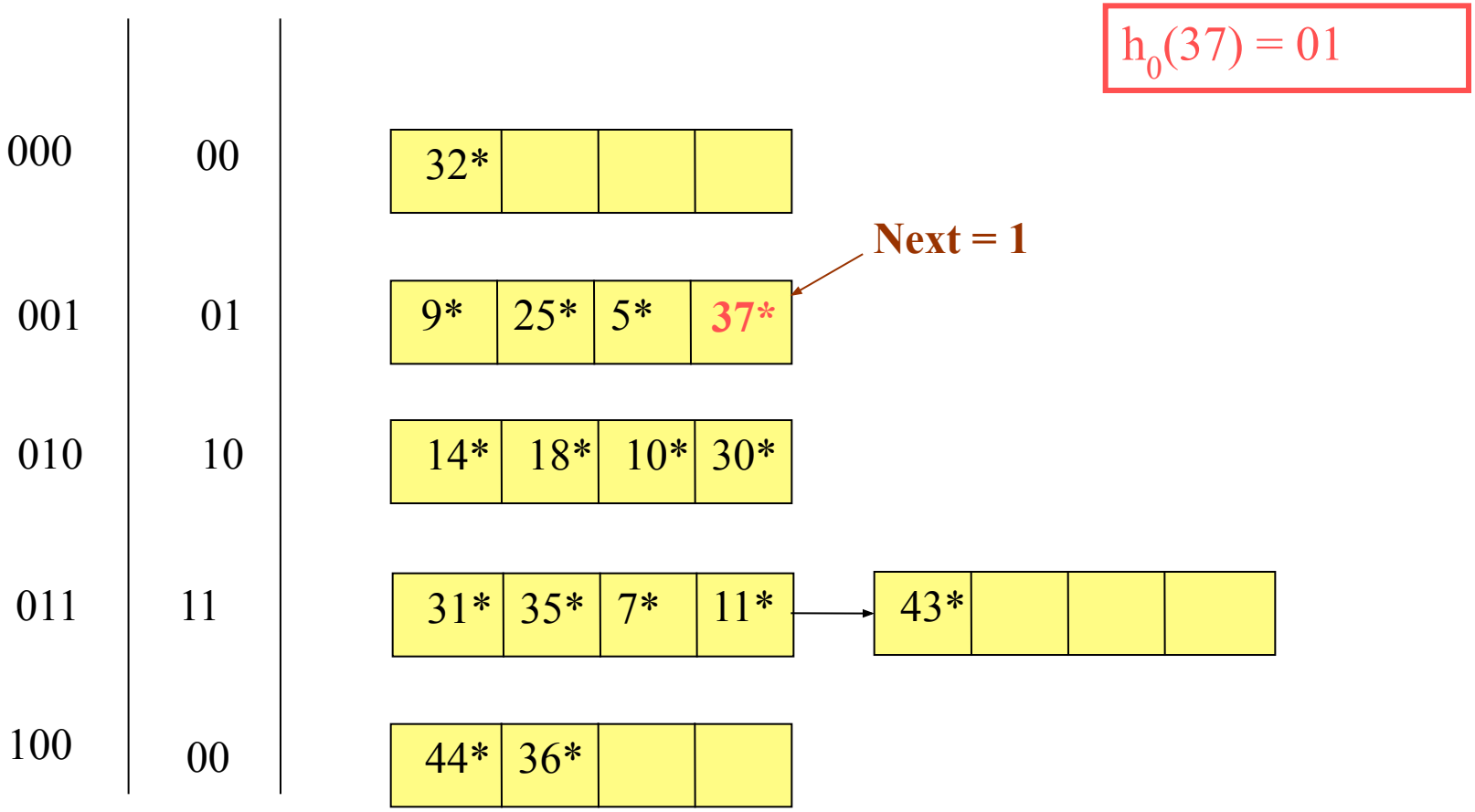
Busca de 18*


h1
h0
Nível = 0 , N = 4 = 2^2
h0(18) = 2 (10)
000
00
32*
Next = 1
001
01
9*
25*
5*
2 > Next
010
10
14*
18*
10*
30*
Bucket 2 não foi dividido ainda
011
11
31*
35*
7*
11*
43*
100
00
44*
36*

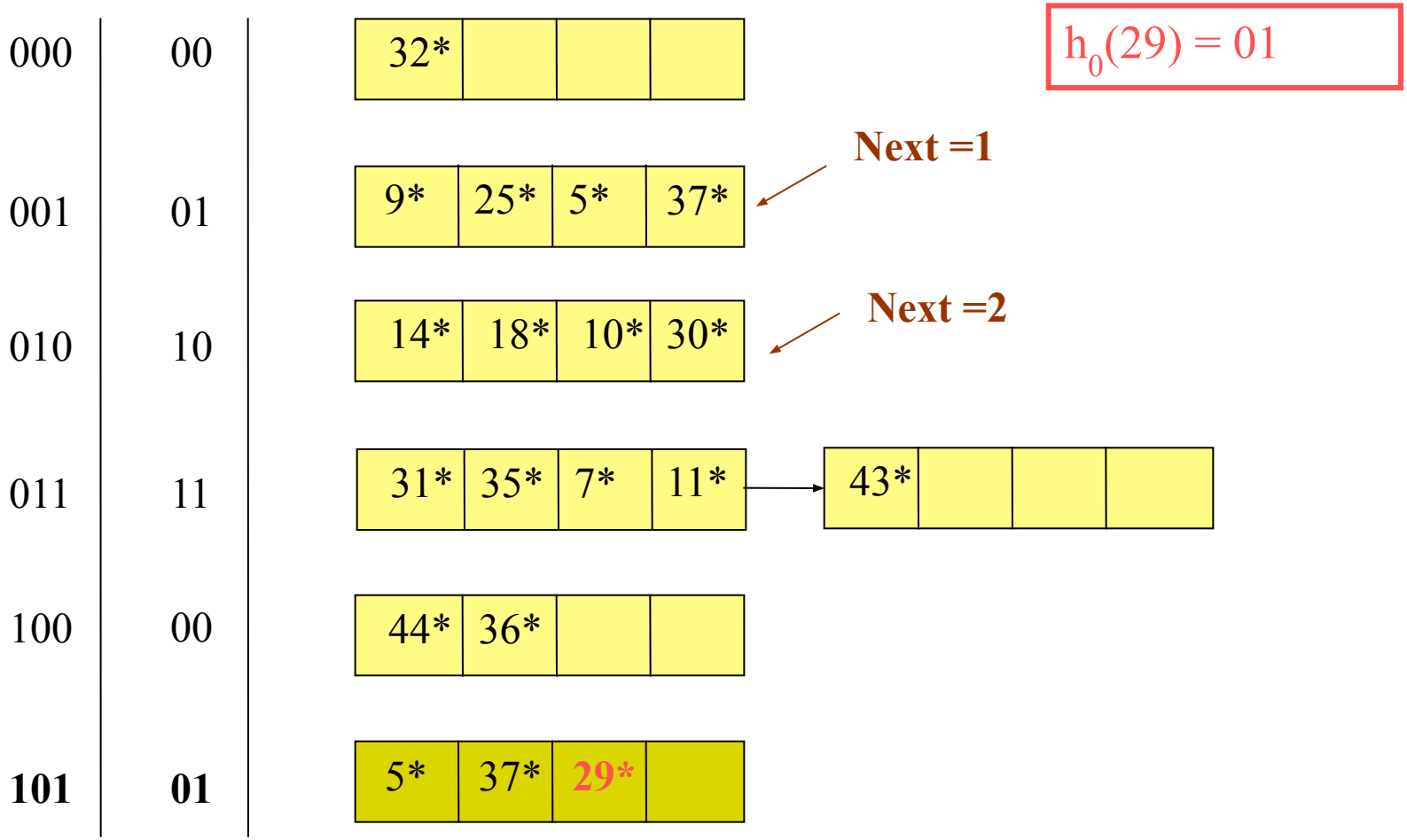
Busca de 32* e 44*



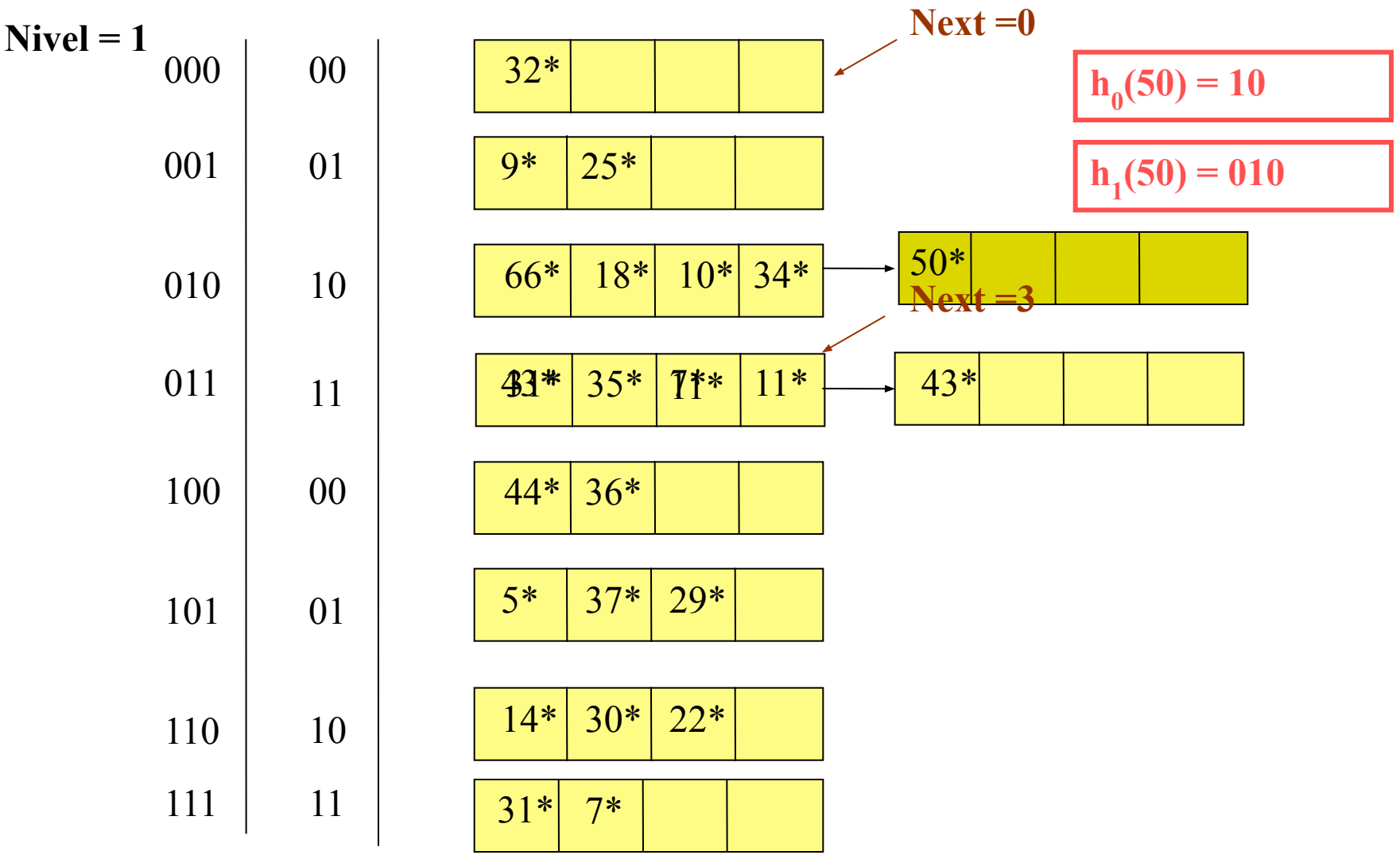
Inserção do 37*



Inserção de 29*



Incremento de Nível



Atividade

- Construa uma estrutura de hash linear para a lista de valores abaixo.
- Considere que o bucket tenha espaço para dois valores, e que a função de hash usada seja (valor f= N mod 8).
- Lista: 9, 101, 14, 18, 800, 3, 1, 7, 35, 92.

Vantagens e Limitações

- Hash : excelente para seleção por *igualdade na chave*.
- Não suporta seleção *range (>, <)*
- B-Trees suportam seleção *range* e são quase tão boas quanto Hash para *igualdade*.
- Muitos SGBDs só implementam índices estruturados por B-Trees
- Técnica de indexação Hash é muito útil na implementação do operador *Junção,* que inclui diversas seleções por igualdade
  - Diferença de custo entre B-Tree e Hash é significativa neste caso.