GBC053–Gerenciamento de Banco de Dados
Índices baseados em Hash

Ilmério Reis da Silva
ilmerio@facom.ufu.br
www.facom.ufu.br/~ilmerio/gbd
UFU/FACOM/BCC

Roteiro

- *Fundamentos*
- *Hash Estático*
- *Hash Extensível*
- *Hash Linear*
- *Considerações Finais*

Fundamentos

Características

- *função hash : set of keys → set of bucket ids*
- *melhor desempenho em busca por igualdade*
- *não melhora busca por intervalo (>, <)*
- *ganho significativo em junções EquiJoin*
- *hash ideal: distribuição uniforme das entradas*
- *entrada nos buckets: alternativa (1), (2) ou (3)*
- *alguns SGBDs só implementam Árvore B+*
- *o PostgreSql implementa também Tabelas Hash*

Função Hash

Exemplo:

hash(k) = h(k) mod N,

onde:

- ✓ *h : set of keys → integer*
- ✓ *N : é o número de buckets disponíveis*
- ✓ *SE k é inteiro, use*
  - *h(k) = k ou*
  - *h(k) = a k + b, sendo a e b empíricos.*

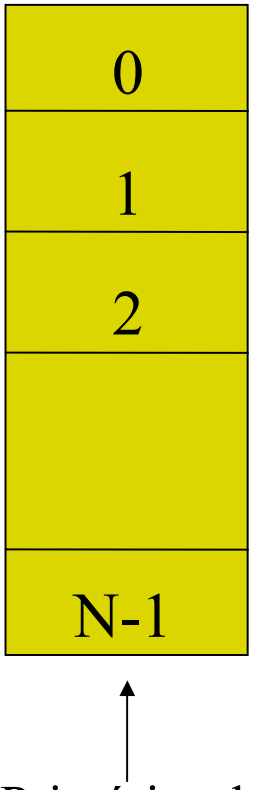
Classificação

- ***Hash Estático:***
  - *conjunto de buckets fixo*
  - *problemas com cadeias de overflow*
- ***Hash Dinâmico:***
  - *cria buckets de forma dinâmica*
  - *evita ou reduz cadeias de overflow*
  - ***Hash Extensível:***
    - *diretório indexado por uma sequência de bits;*
    - *havendo split, a sequência de bits cresce*
  - ***Hash Linear:***
    - *novos buckets são criados linearmente*
    - *ameniza impacto do crescimento, 'tolerando' poucos overflows;*
- ***Compromisso (trade-off) Hash Estático x Hash Dinâmico é análogo ao compromisso ISAM x B-Tree***

Hash Estático

Alocação de área para buckets

Alocação de N páginas sucessivas no disco

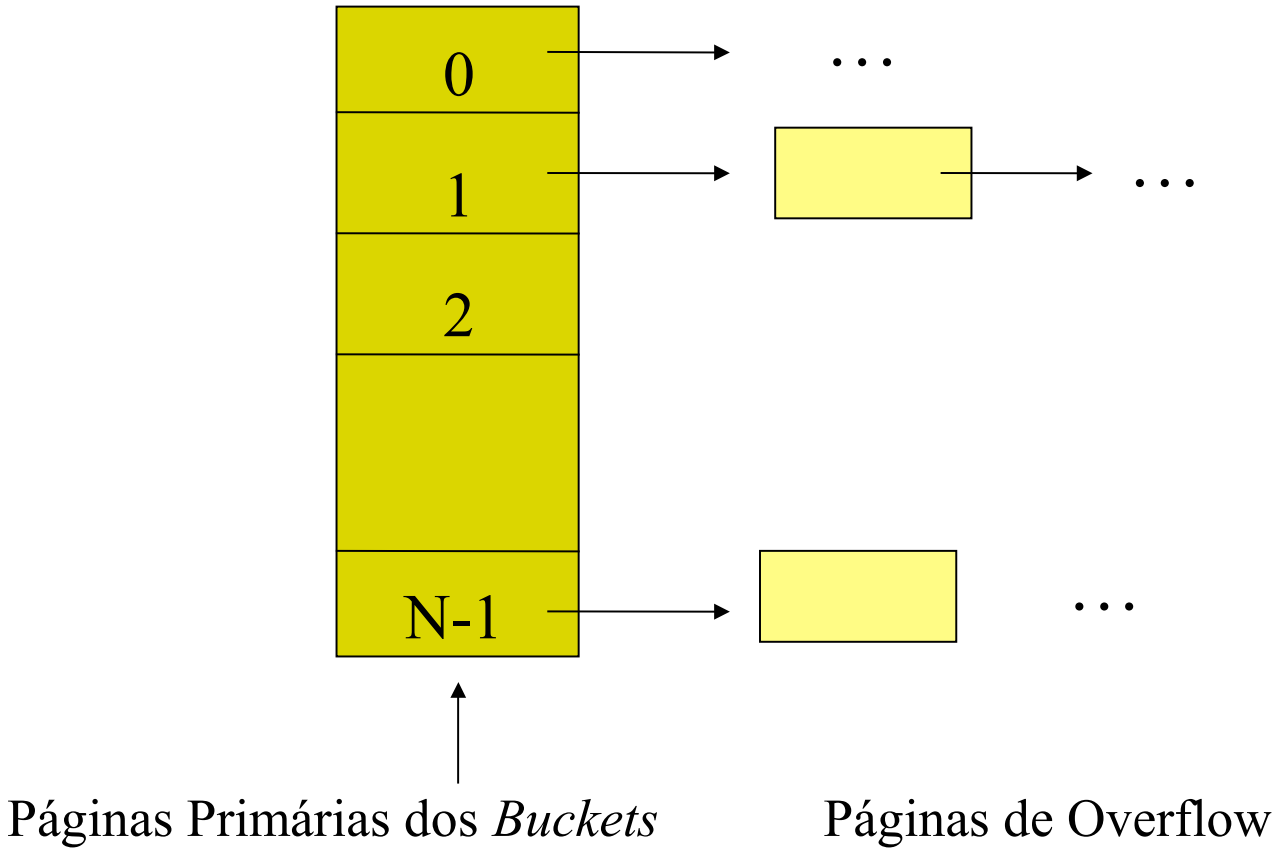


Páginas Primárias dos *Buckets*

Cada bucket terá entradas de dados <chave*> (Alt. 1, 2 ou 3)

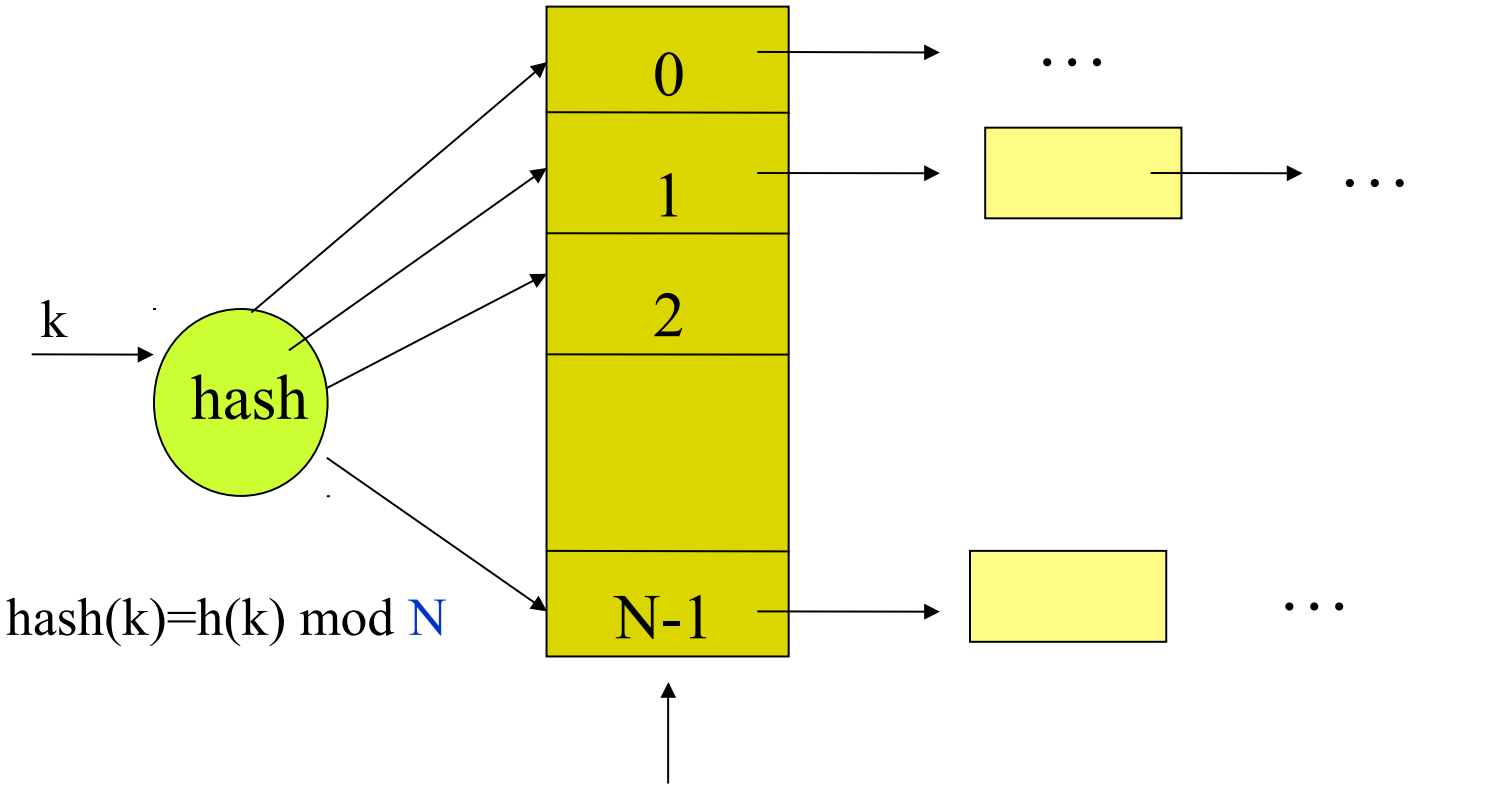
Alocação dinâmica de páginas de overflow

Páginas de overflow são alocadas de forma dinâmica

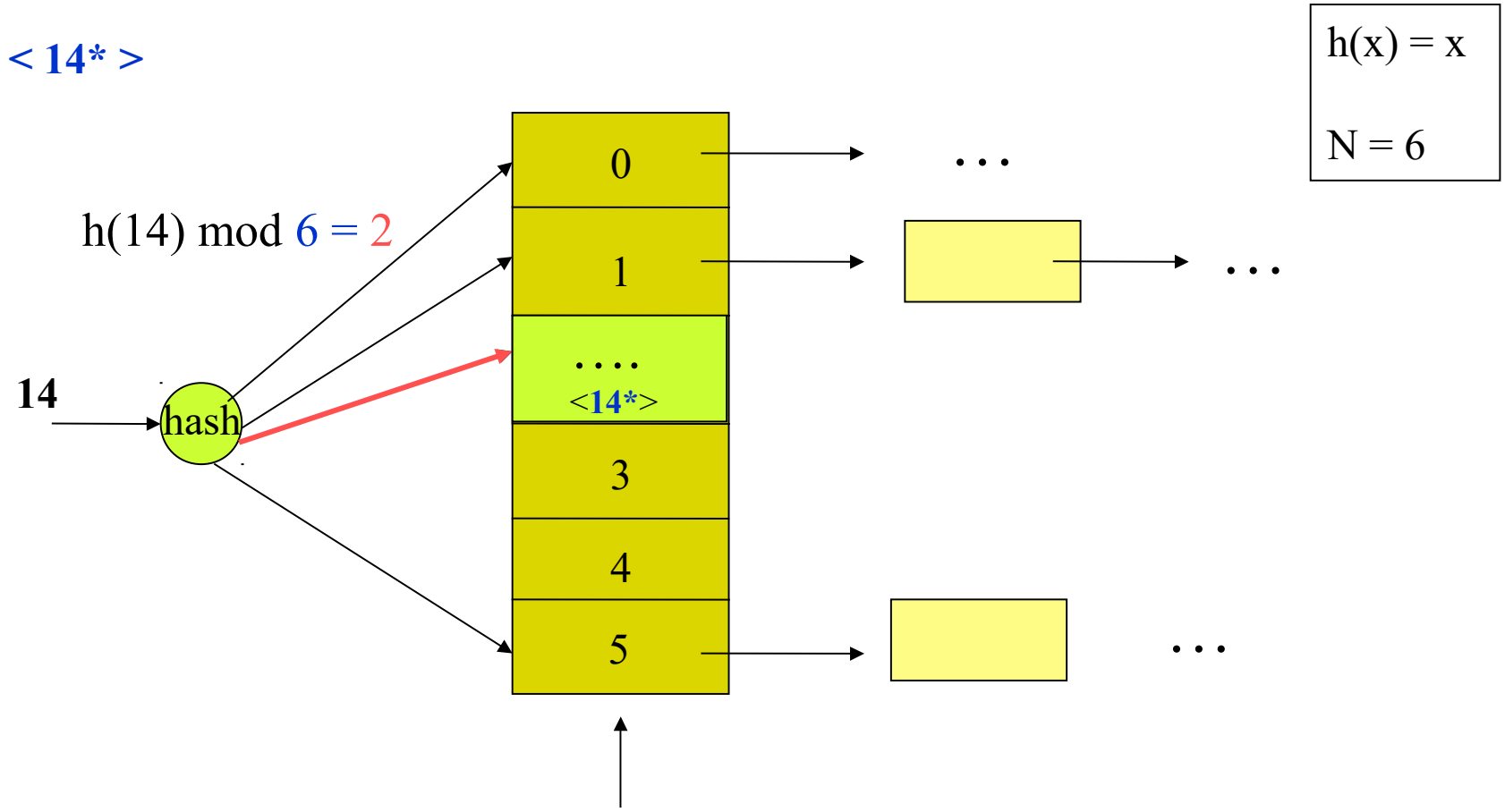


Alocação dinâmica de páginas de overflow

Seja k a chave do hash, então a localização do bucket será por meio da função hash

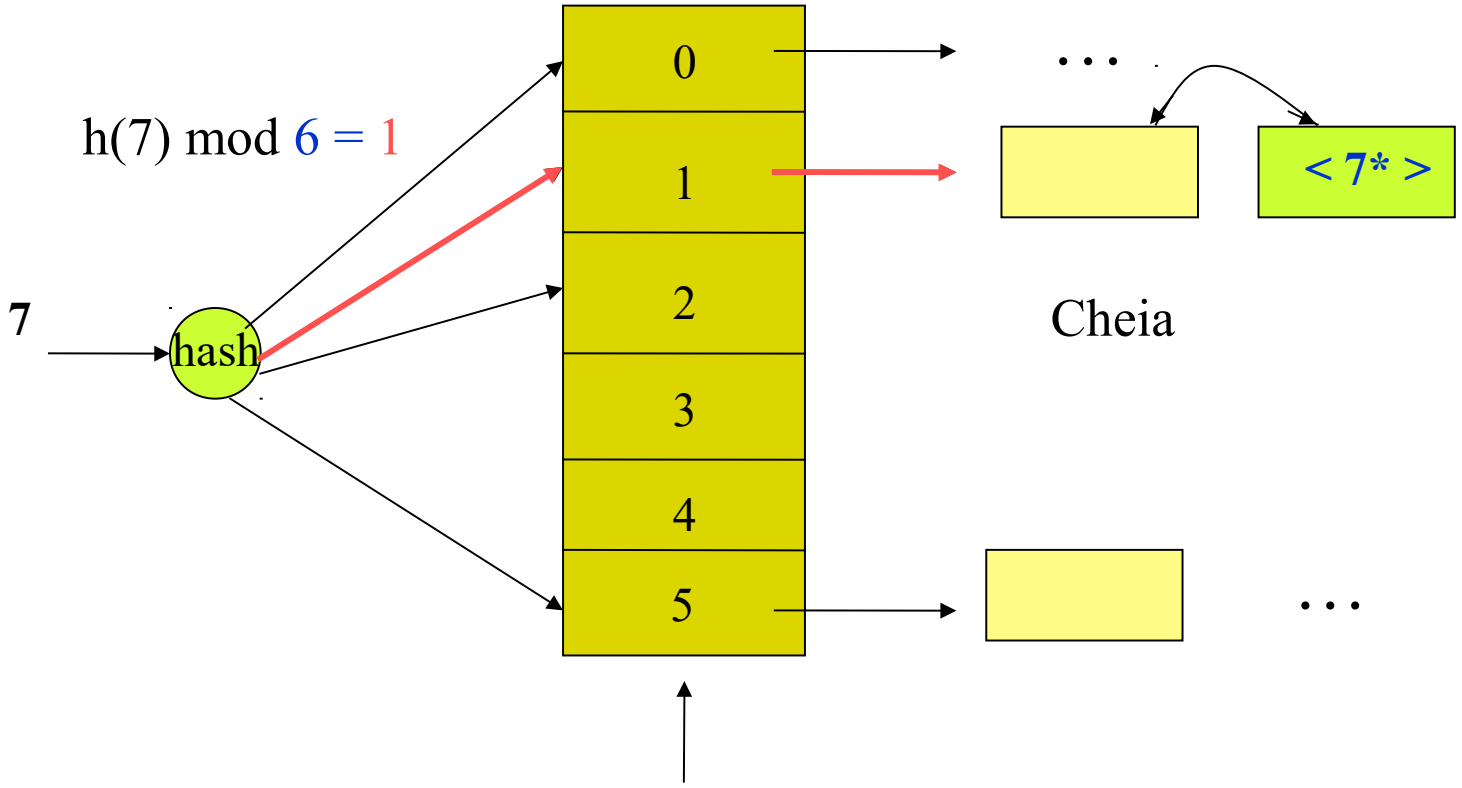


Busca


< 14* >
h(x) = x
N = 6
h(14) mod 6 = 2
14
hash
0
1
....
<14*>
3
4
5
Páginas Primárias dos *Buckets*

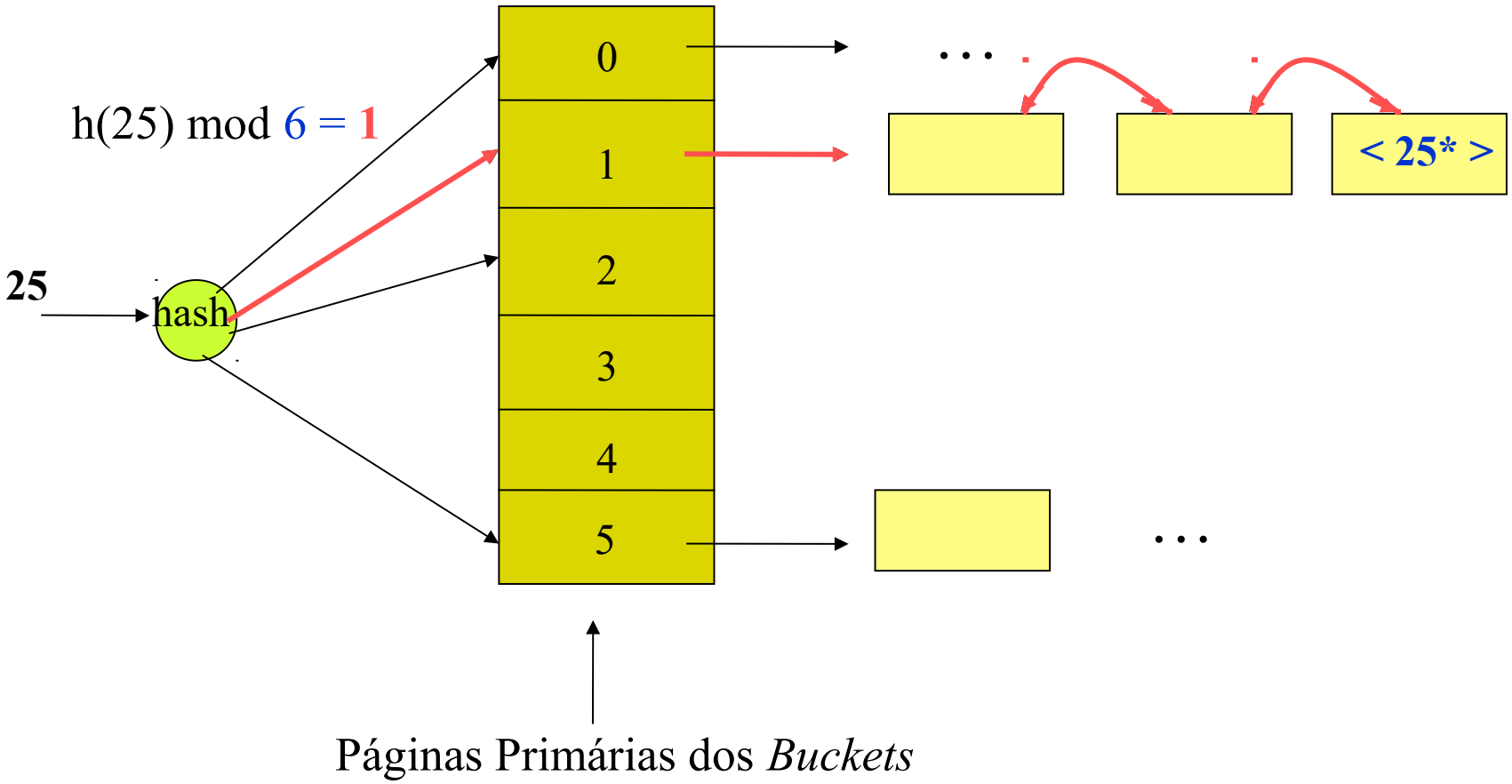
Inserção

Inserindo < 7* >



Remoção

Suprimindo < 25* >



Considerações finais sobre Hash Estático

- *usualmente bucket = h(k) mod N, onde N = 2^L*
- *custo depende do tamanho das cadeias de overflow*
- *páginas primárias geralmente são contíguas no disco*
- *bucket não é removido, mesmo ficando sem entradas*
- *cadeia de overflow é dinâmica e degrada desempenho*
- *uma estratégia é, na construção do índice, calcular o espaço dos buckets deixando em média 20% de área livre nas páginas primárias*
- *o problema da cadeia de overflow pode ser tratado com hash dinâmico...*

Hash Extensível

Evitando cadeias de overflow
Solução ingênua

Seja N o número de páginas primárias

Solução ingênua:

SE inserção em página primária cheia

*faça N=2 * N*

reorganize todos os buckets

Problema: a reorganização de todos os buckets é computacionalmente cara, tornando esta solução inviável.

Evitando cadeias de overflow
Hash Extensível

- *Um diretório de ponteiros para buckets*
- *SE inserção em página primária cheia*

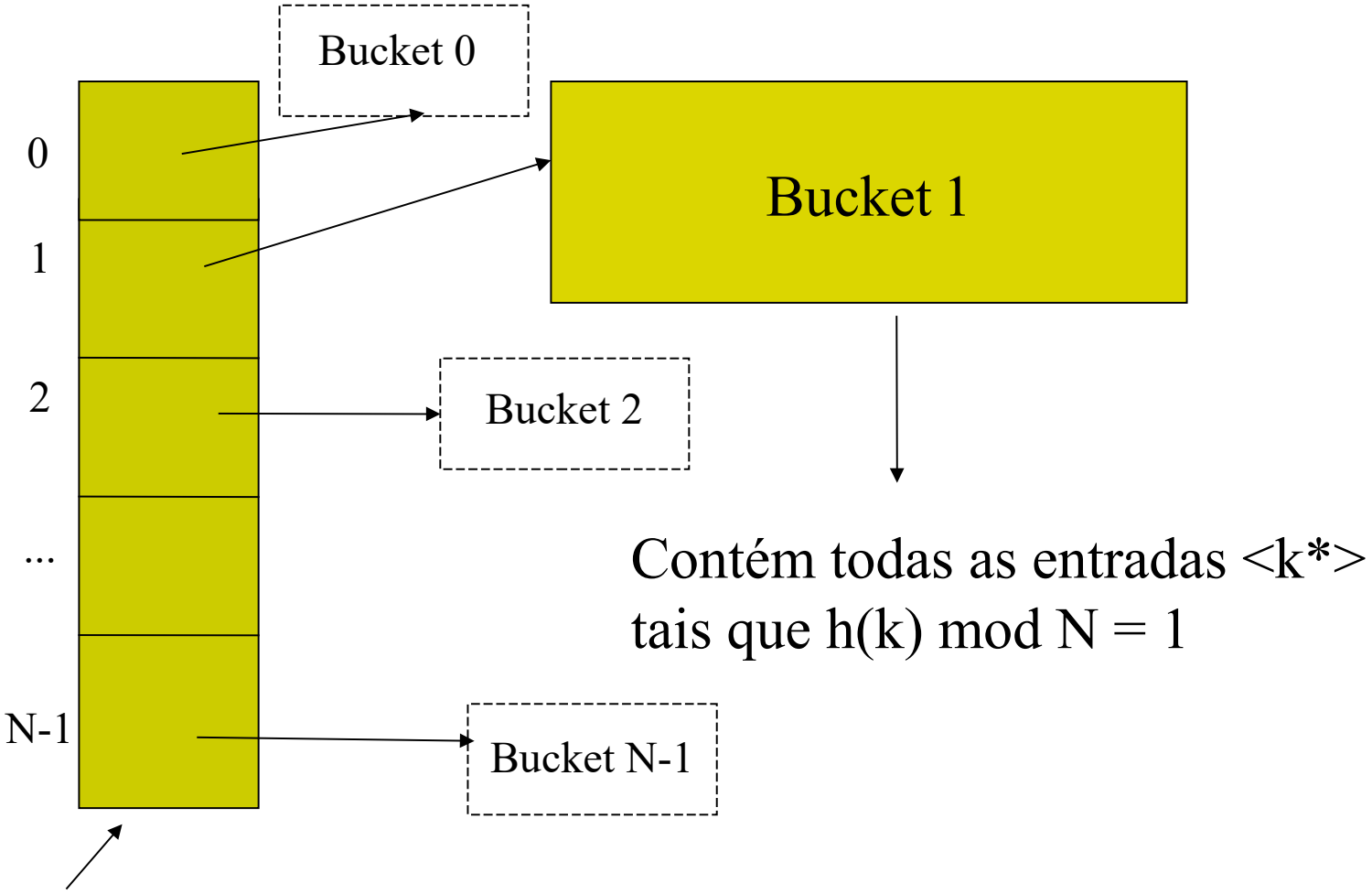
  crie um novo bucket

  dobre o tamanho do diretório

  reorganize a função hash

Problema: a reorganização da função hash é um pouco complexa, mas considerando que o diretório cabe na memória, é computacionalmente viável.

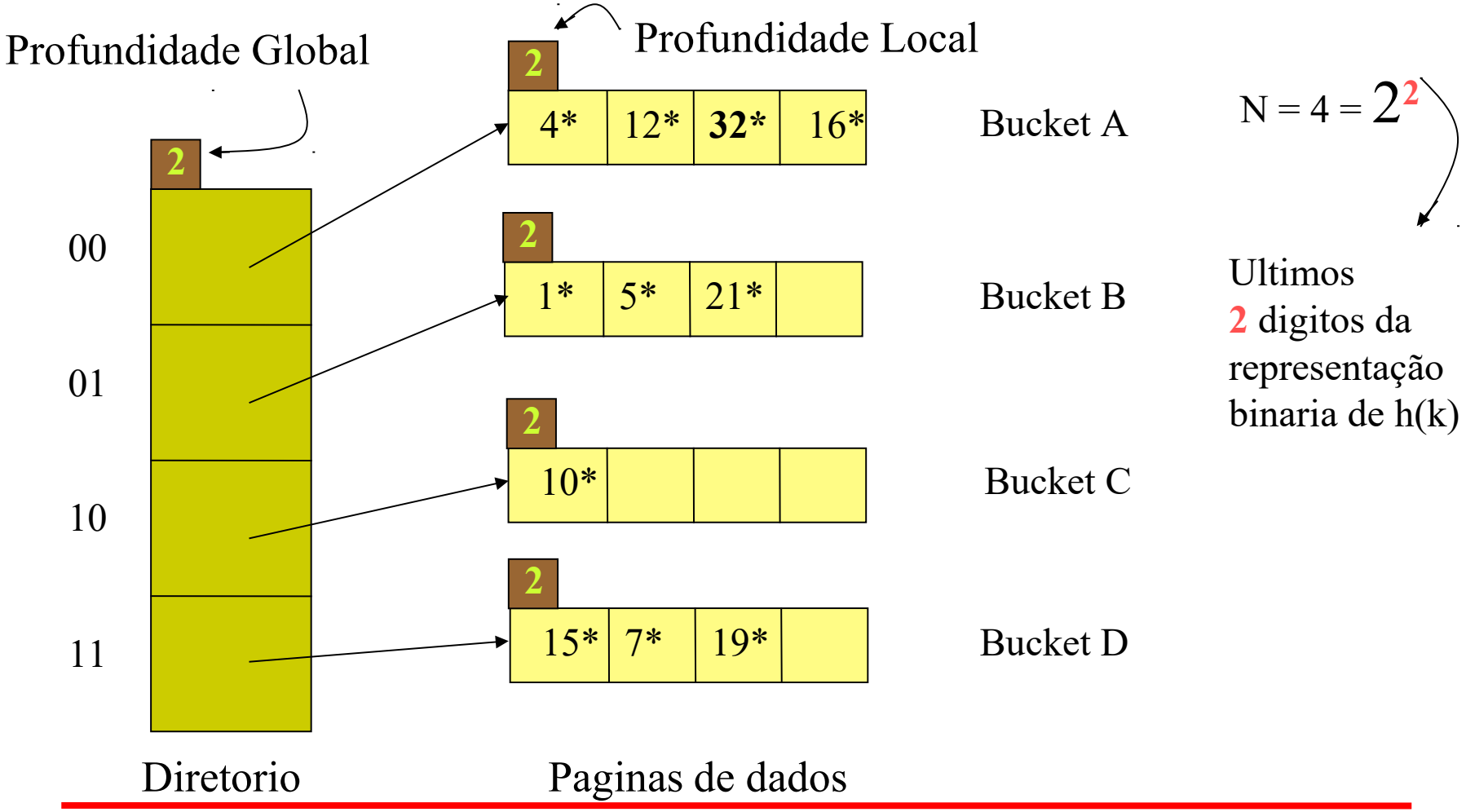
Diretorio de ponteiros para buckets



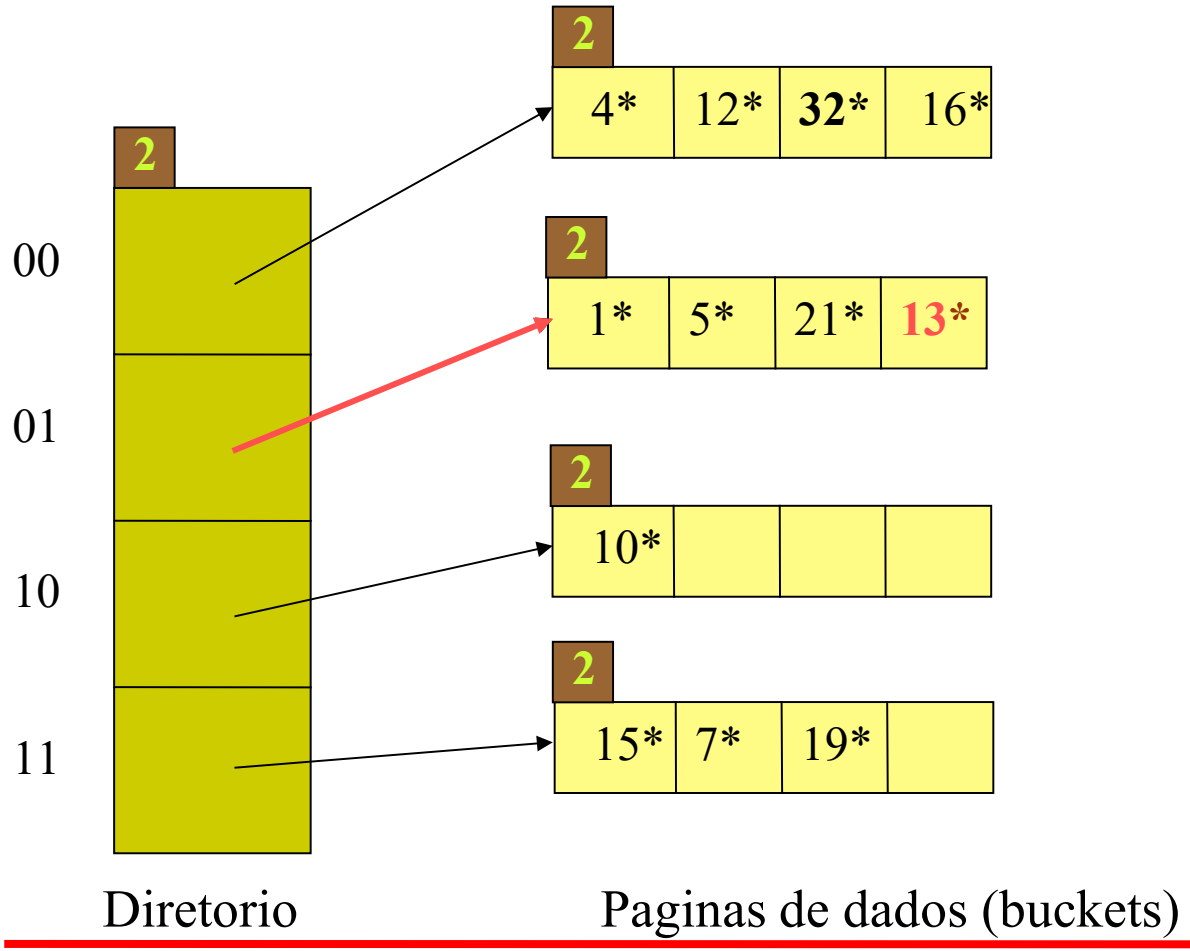
Funcionamento

- *Função hash não varia*
- *O número N de entradas no diretório varia*
- *A medida que os buckets se enchem, estes se duplicam, e o diretório de buckets também pode duplicar. Atenção somente um bucket por vez é duplicado*
- *Resultado:*
  - *se um único bucket duplica, o diretório todo pode duplicar*
  - *vários ponteiros do diretório podem apontar para o mesmo bucket*
  - *só duplicam os buckets que ficam cheios.*
  - *ao contrário do hash estático, registros em buckets duplicados (decorrentes de um overflow) podem ser facilmente localizados através do novo ponteiro no diretório de buckets.*

Exemplo

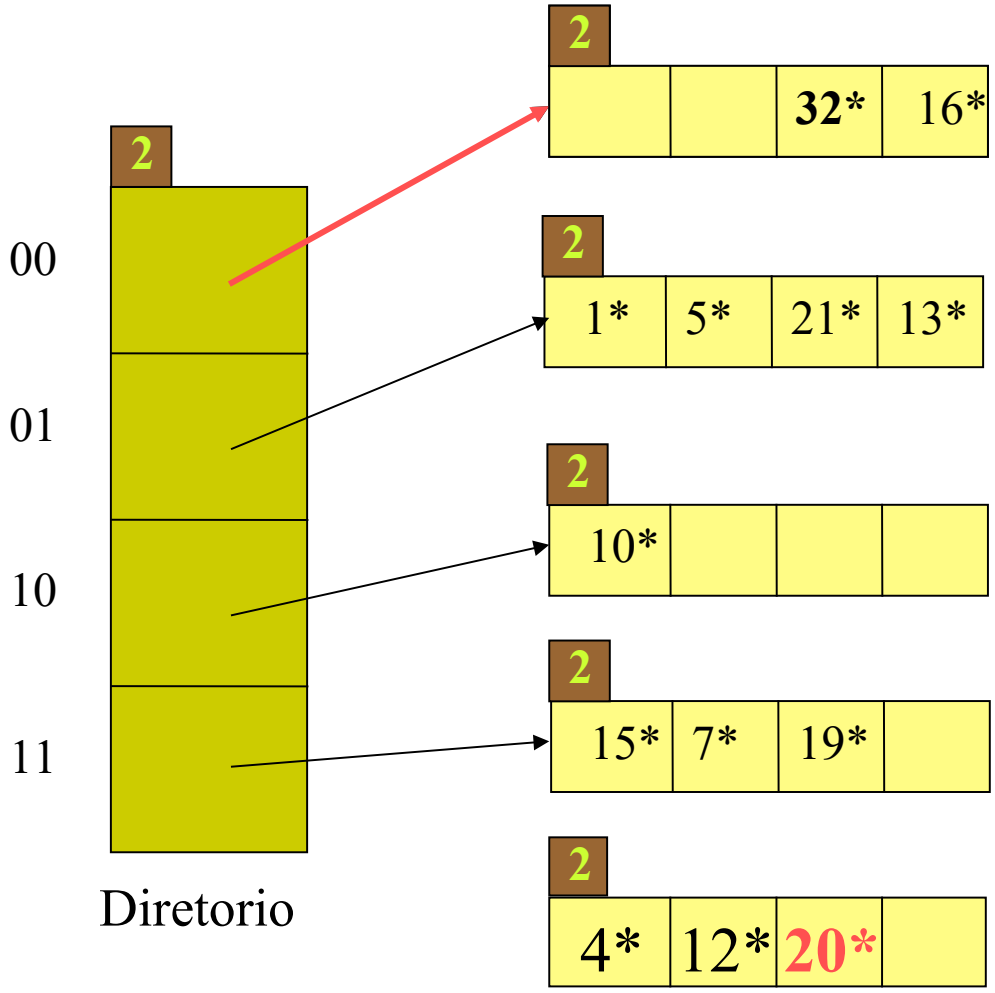

Profundidade Global
Profundidade Local
2
00
01
10
11
4* 12* 32* 16*
Bucket A
1* 5* 21*
Bucket B
10*
Bucket C
15* 7* 19*
Bucket D
N = 4 = 2²
Ultimos
2 digitos da
representação
binaria de h(k)
Diretorio
Paginas de dados

Inserção simples



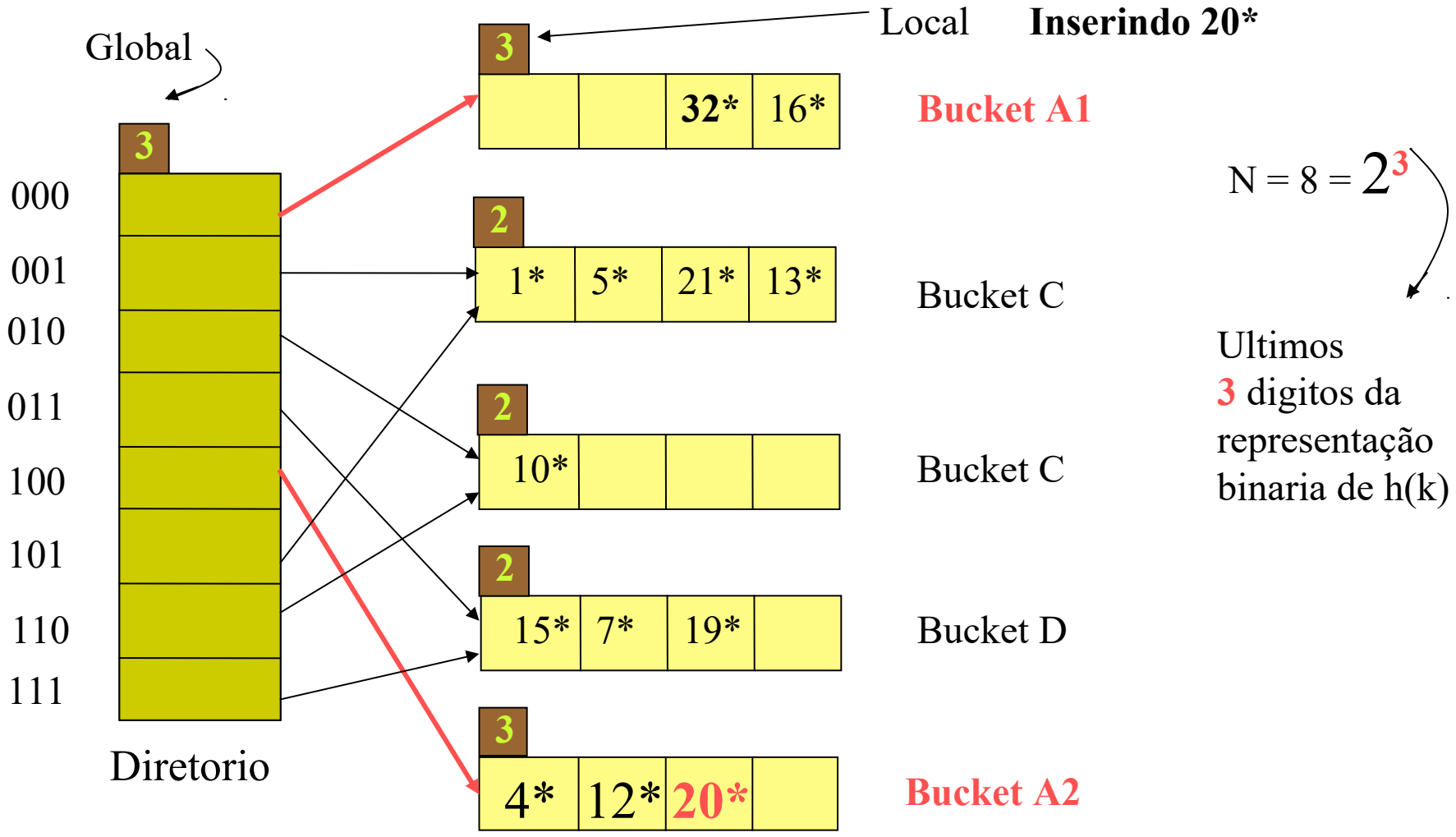
Inserindo 13*

Inserção em bucket cheio
Split do bucket



Inserindo 20*

Inserção em bucket cheio
Duplica diretório e reorganiza da função hash



Regra de inserção com split - Hash Extensível

Sejam: *pl a profundidade local e*

pg a profundidade global;

Incrementar *pl;*

Criar *novo bucket e* ***Redistribuir*** *entradas de acordo com pl-ésimo bit mais significativo de hash(k);*

SE *(pl > pg), duplicar diretório:*

Incrementar *pg;*

Criar *novas entradas no diretório (i.e. duplicar);*

Apontar *novas entradas p/ buckets originais(pl-1 bits);*

Atualizar *ponteiro que apontará para o novo bucket, de acordo com os pl bits mais significativos de hash(k).*

Considerações Finais – Hash Extensível

- *diretório geralmente cabe na memória*
- *Exemplo:*
  - 100MB em páginas de 4K → 25000 entradas no diretório
- *remoção usa processo inverso, podendo reduzir tamanho do diretório*
- *problema:*
  - distribuição não uniforme de chaves geram muitos *splits* localizados e, consequentemente, diretório pode ter crescimento exagerado.

Hash Linear

Hash Linear

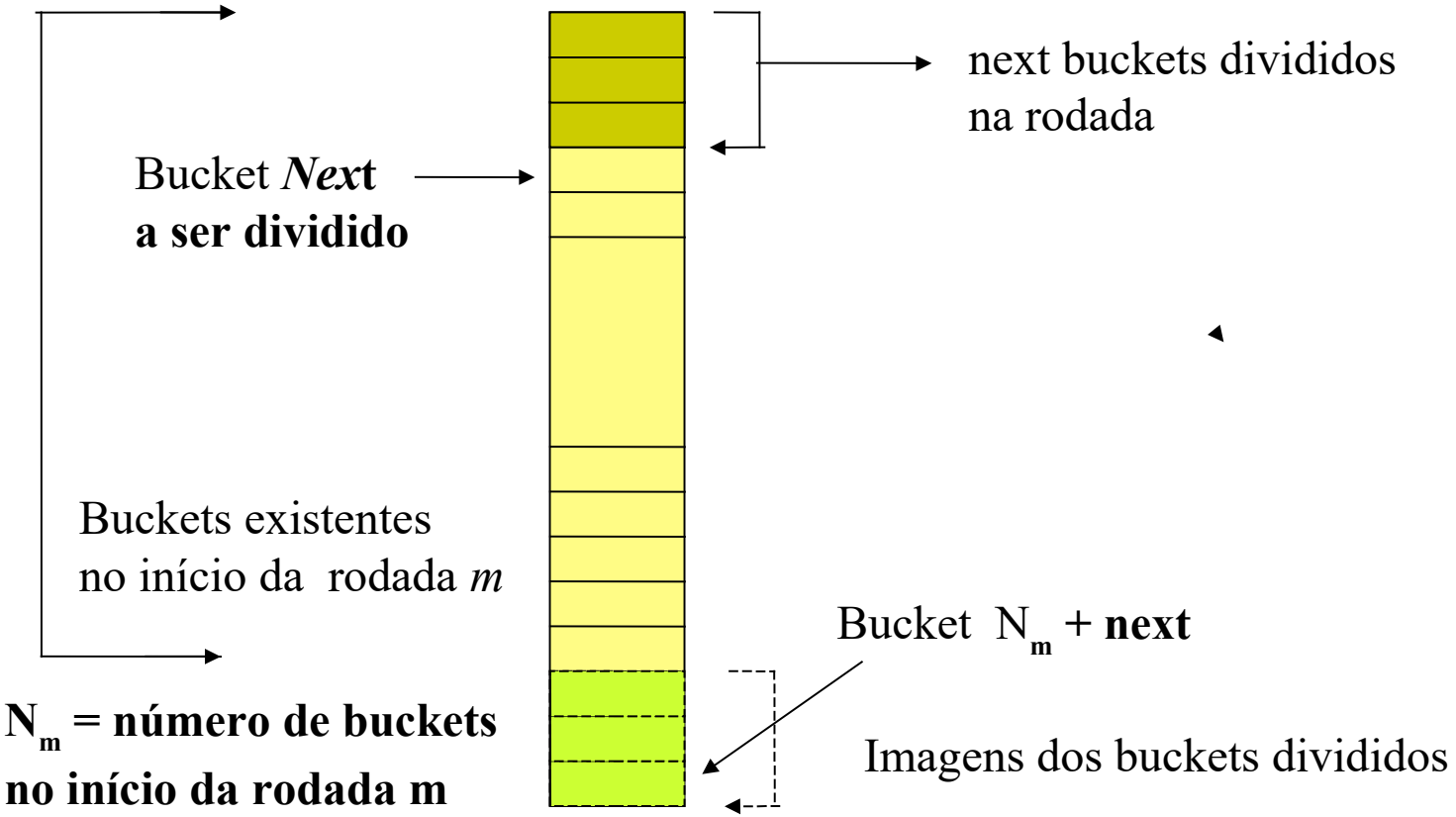
Hash Linear

- *Não ha diretório de buckets*
- *Pode haver páginas de overflow à medida que os buckets se enchem*
- *Regularmente páginas de overflow são transformadas em novos buckets*
- *Regularmente a função hash se modifica, dobrando a capacidade de endereçamento de buckets*

Parâmetros e Contadores

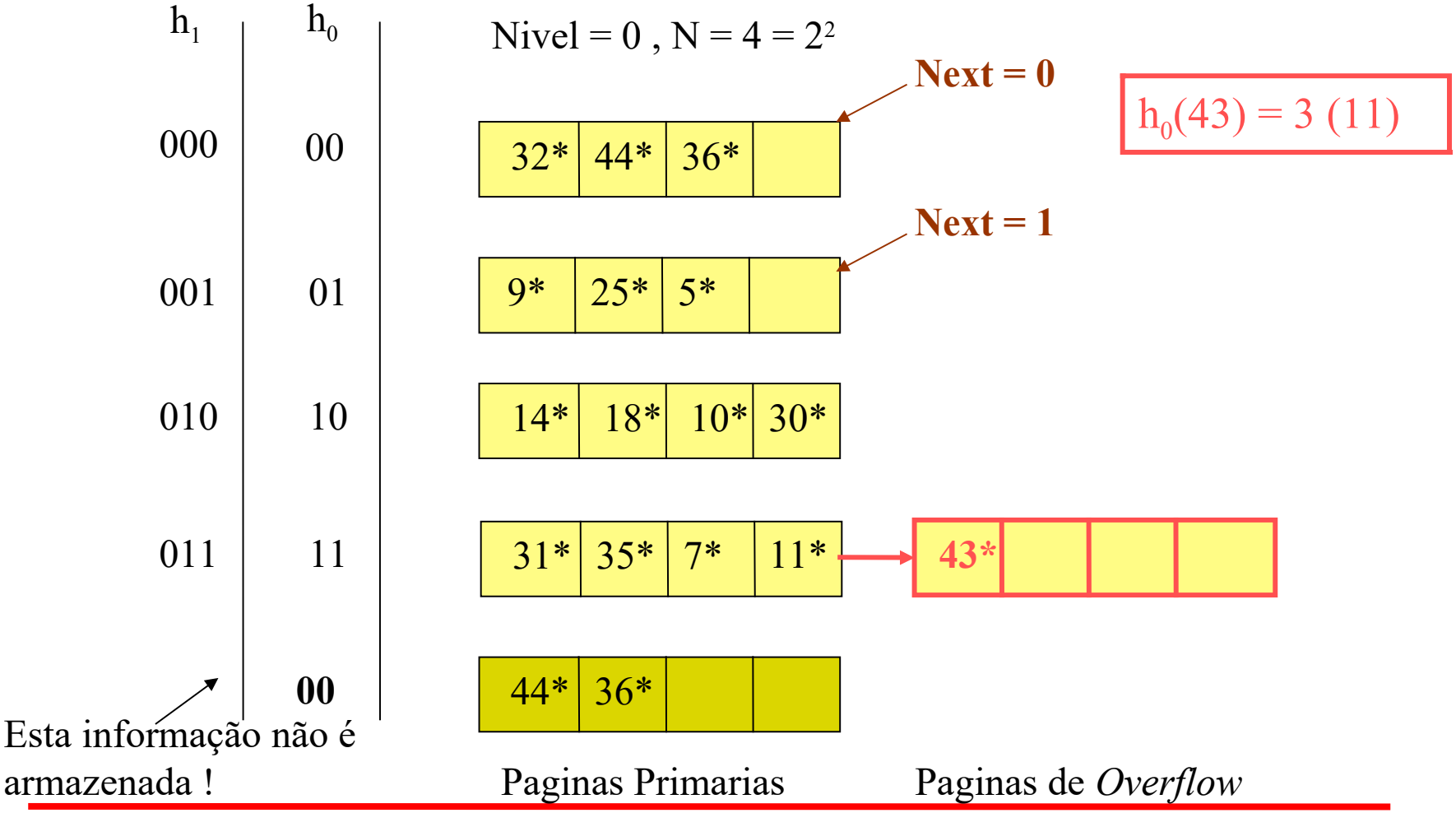
- *Nivel = indica a rodada atual, a função de hash atual*
  - Inicializado com 0
- **Next = bucket que deve ser dividido**
  - Inicializado com 0
- N_m = *número de buckets na rodada m*
  - $N_0 = N$
  - $N_m = N*2^m$
- **Somente o bucket com número Next é dividido**.
  - Usa-se páginas de overflow para os outros buckets, se ficarem cheios.
  - Apos divisão, *Next* é incrementado

Esquema Geral : rodada *m*



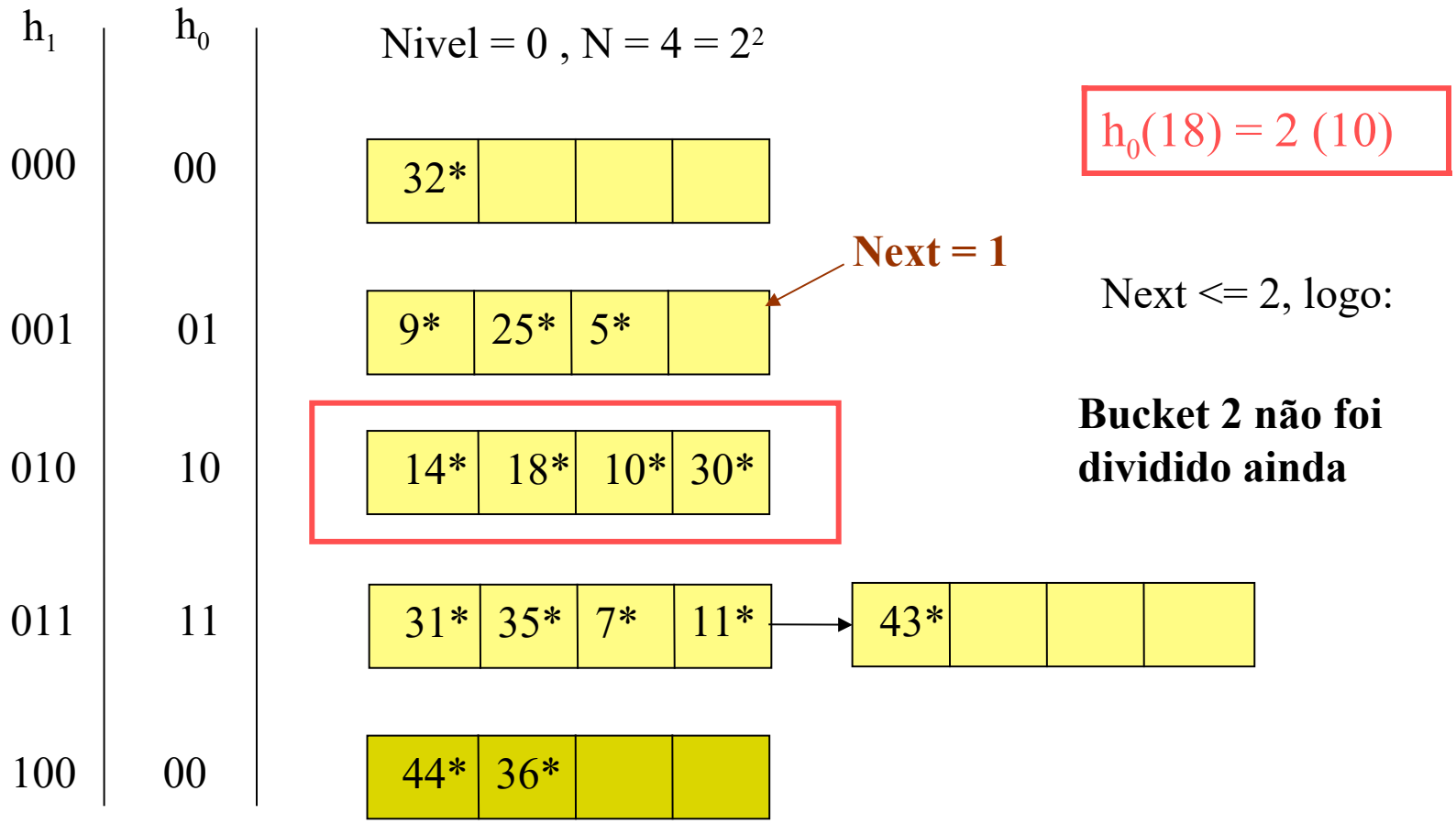
Inserção provocando *split* e avanço do *next*

Inserção de 43*



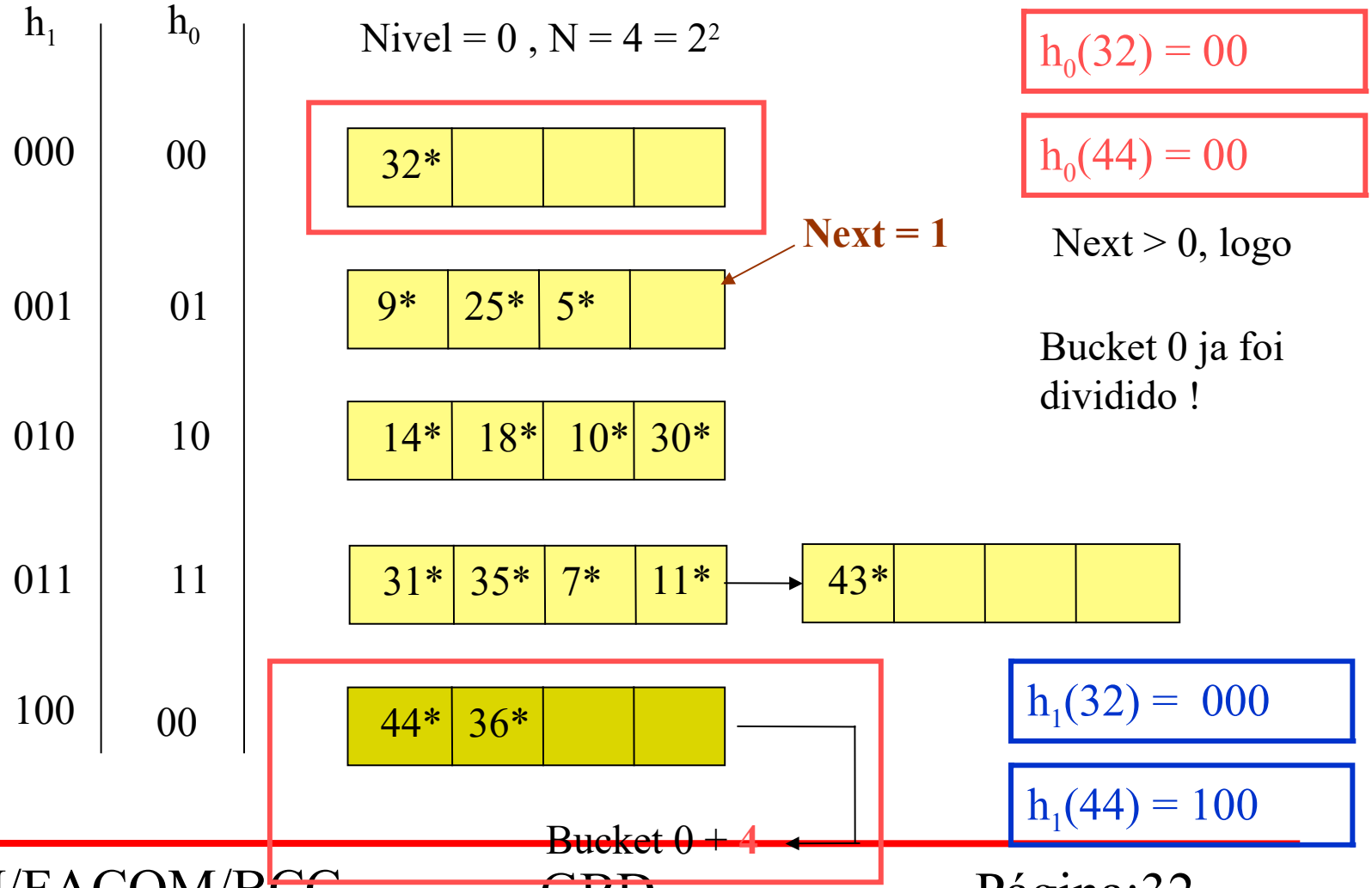
Busca em bucket após o next

Busca de 18*



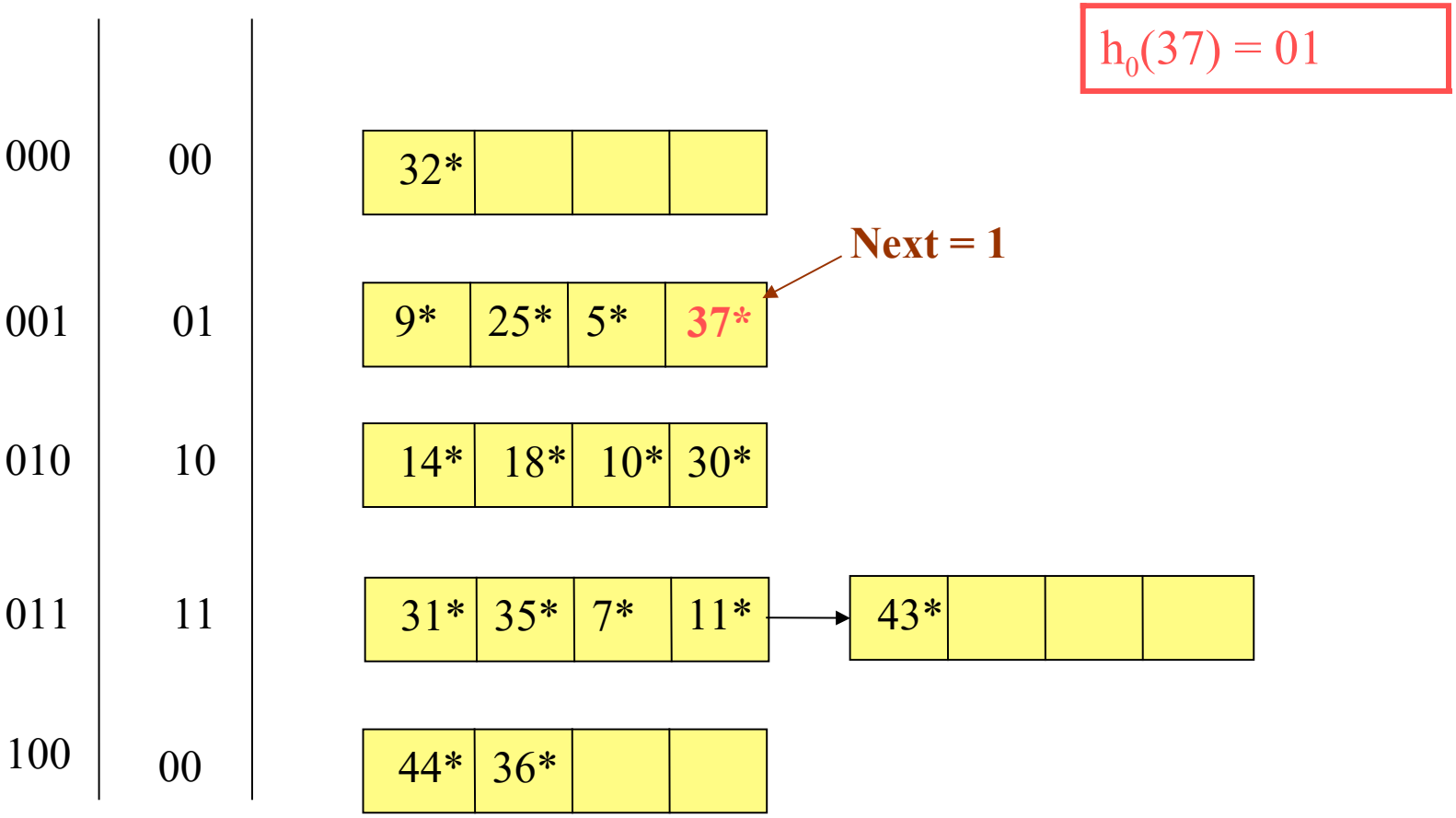
Busca em bucket antes o next

Busca de 32* e 44*



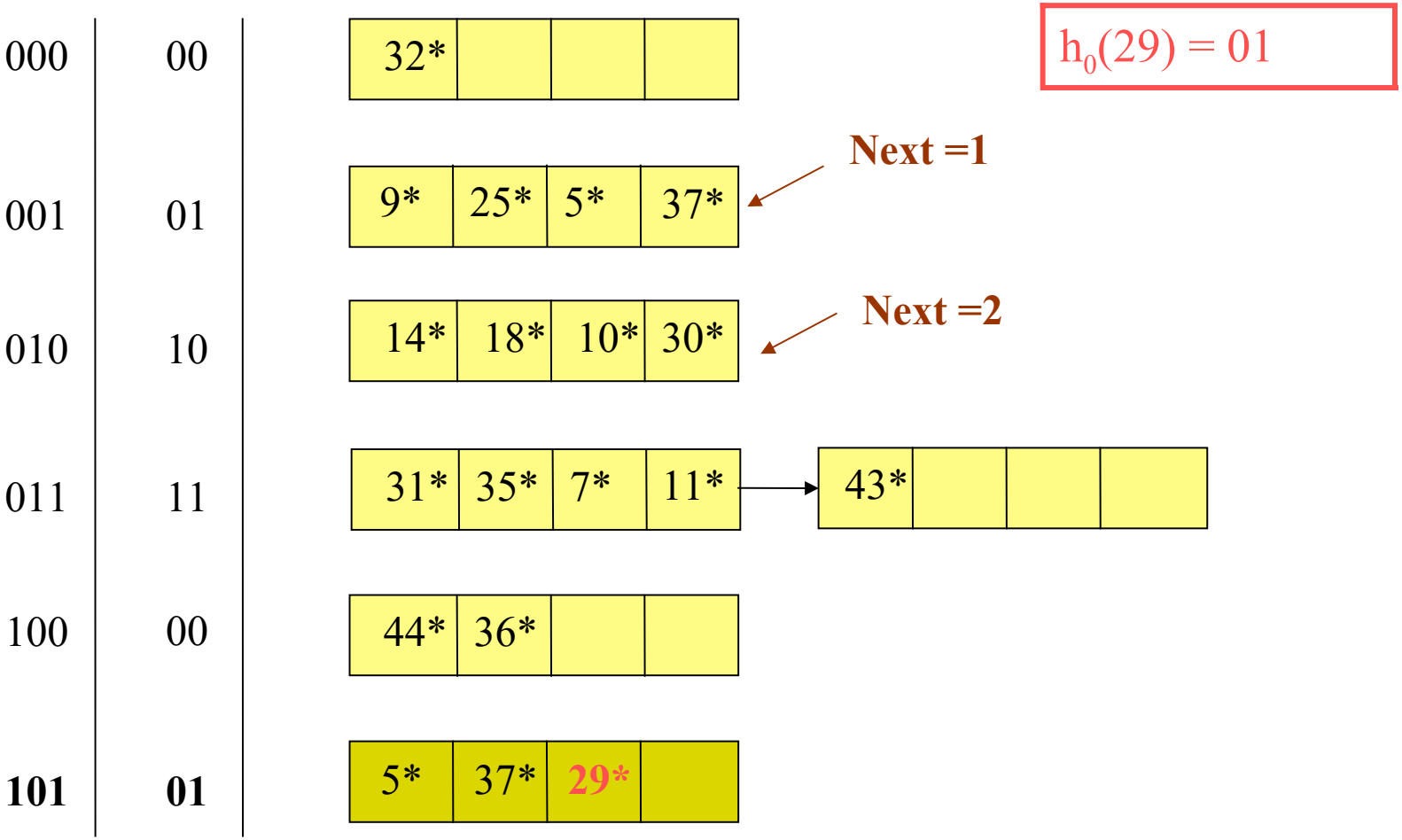
Inserção sem split

Inserção de 37*



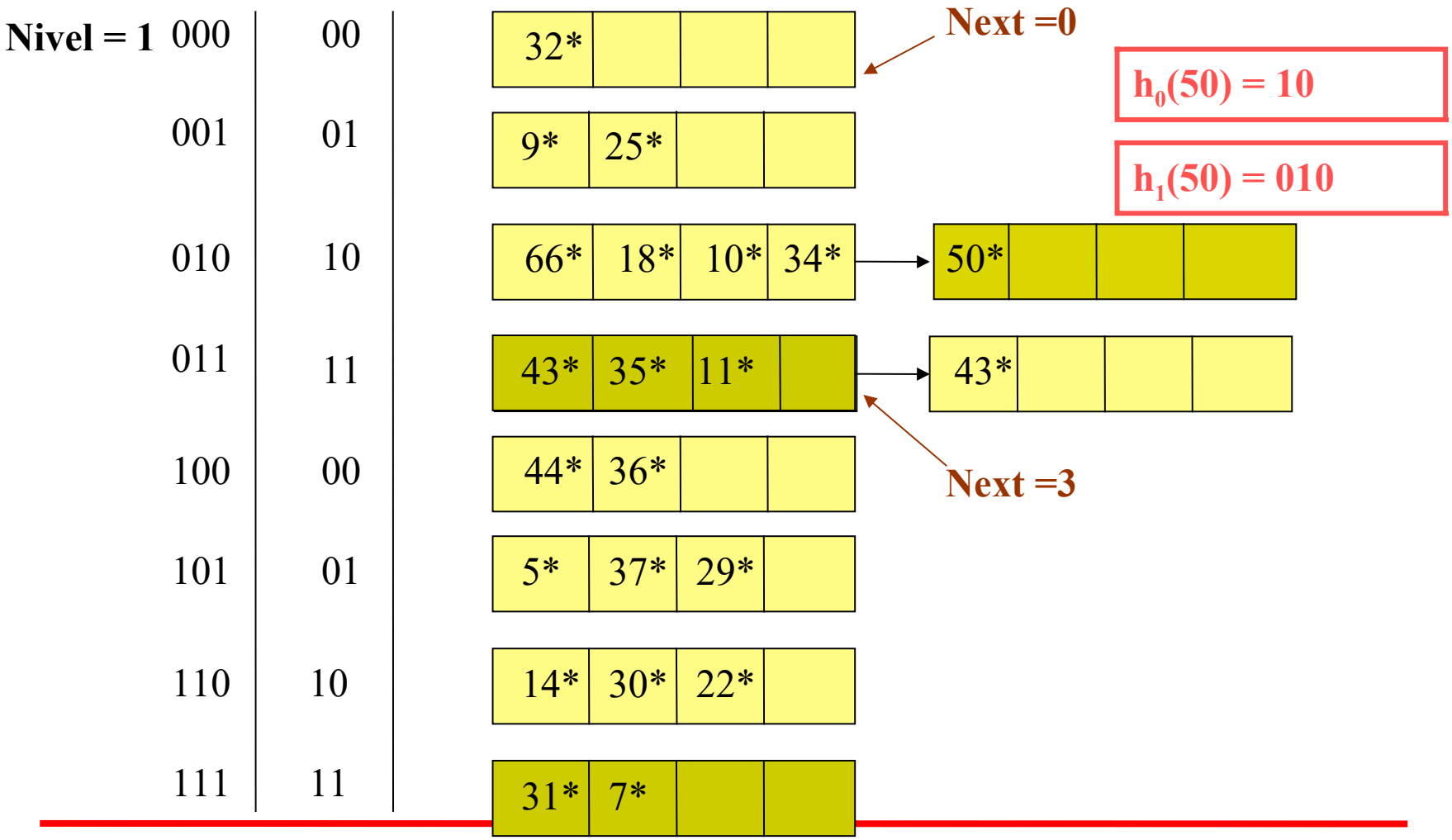
Inserção com split e incremento no next

Inserção de 29*



Inserção com split e mudança de rodada

Inserção do 50*



Comparando
Duplicar diretório x duplicar buckets

- *Passagem de h_i para h_{i+1} no Hash Linear corresponde a duplicar o diretorio no Hash Extensível*
- *Hash Extensível : diretório é duplicado num único passo*
- *Hash Linear : duplicação do número de buckets se faz gradualmente*

Comparando utilização de espaço nos buckets

- *Hash Extensível tem uma melhor ocupação dos buckets pois só se divide o bucket apropriado*

- *Hash Linear não precisa de diretório:*
  - ✓ existe uma maneira precisa de se saber quais buckets foram divididos e quais devem ser divididos e em que circuntâncias
  - ✓ Buckets são alocados consecutivamente: é possível localizar a página do bucket *m* por um simples cálculo de offsets.

Comparando Custo IO Extensível x Linear

Em busca por igualdade

- Hash Linear tem um custo I/O menor em caso de distribuição uniforme
- Hash Extensível tem um custo menor em caso de distribuição não uniforme

Considerações finais sobre Hash

- *índices baseados em hash são os melhores para busca com predicado de igualdade*
- *índices baseados em hash não suportam busca por intervalo*
- *Hash estático pode resultar em longas cadeias de overflow*
- *Hash extensível usa diretórios para controlar crescimento do hash, evitando overflow*
- *Hash linear usa variação de função para diminuir cadeias de overflow, gerando buckets linearmente*
- *Em geral distribuição não uniforme prejudica desempenho de índices baseados em Hash*

Exercícios - Índices baseados em hash

EXERCÍCIOS CAP 11 LIVRO-TEXTO

*FIM - Índices baseados em hash**

** material baseado no livro-texto e slides da Profa. Sandra de Amo*