

# Colisões, Transbordamento, e Hashing

Eduardo Furlan Miranda

2024-02-01

Adaptado do material do Prof. Jairo F. de Souza

# Hashing - tratamento de colisões

Adaptado do material do Prof. Jairo Francisco de Souza

SOUZA, Prof. Jairo Francisco de. Estrutura de Dados II. Hashing - Resolução de colisões. Universidade Federal de Juiz de Fora. 2012.

# Alguns termos

- Tabela de dispersão = hash table
- Função de espalhamento = hash function
- Código hash = hash code
- Encadeamento coalescido = coalesced chaining = coalesced hashing

# Tratamento de colisões

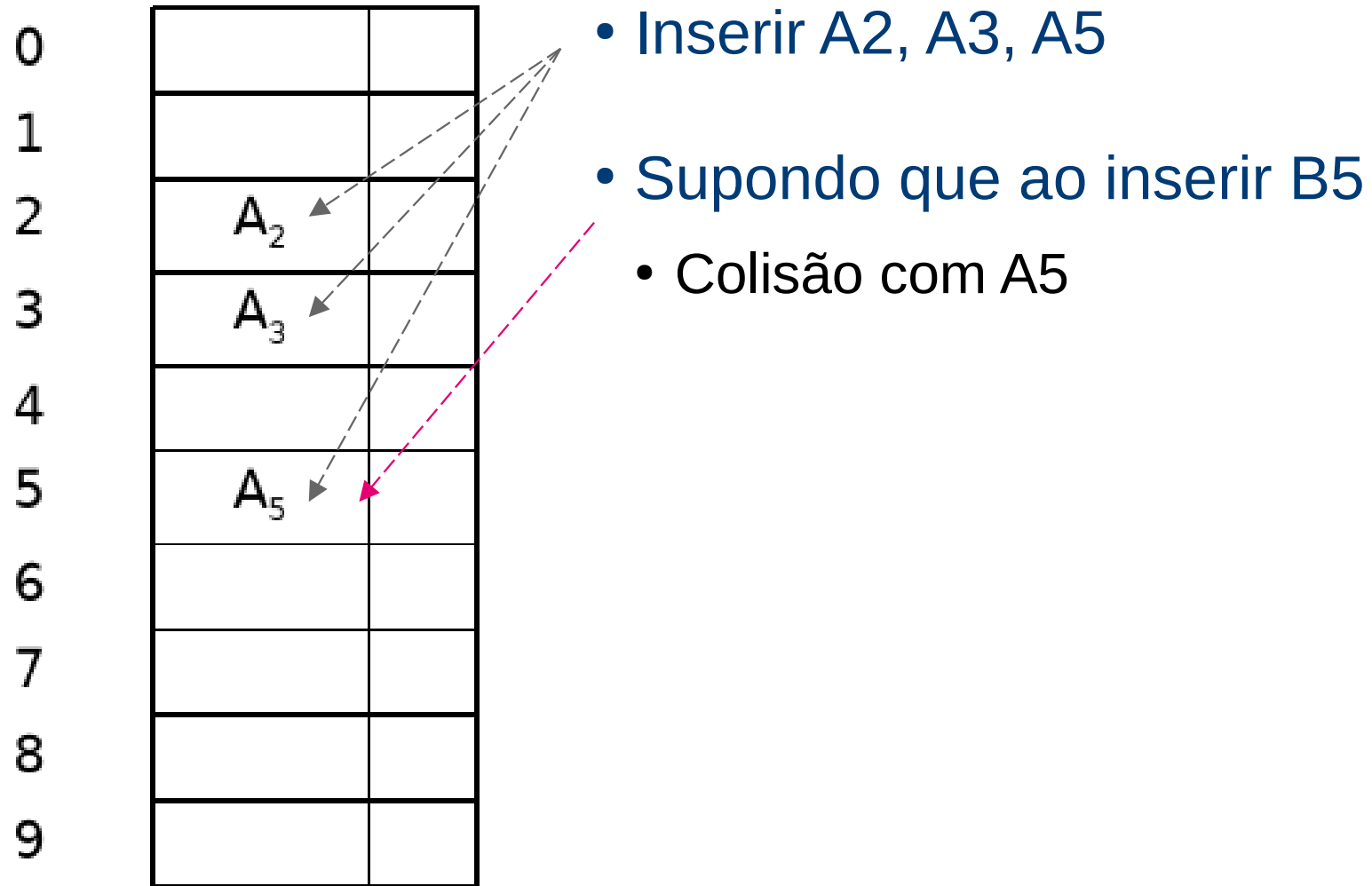
- Endereçamento aberto, fechado, duplo → já visto na U3S1
- Encadeamento
  - Coalescido
  - Coalescido com porão (transbordamento)
- Endereçamento em balde

# Encadeamento coalescido

- Cada posição na tabela possui espaço
  - para a chave
  - para um ponteiro que aponta para a posição da próxima chave
- Quando ocorre colisão
  - Encontrar primeira posição disponível
  - Armazenar o índice dessa posição com a chave que já está na tabela da lista ligada

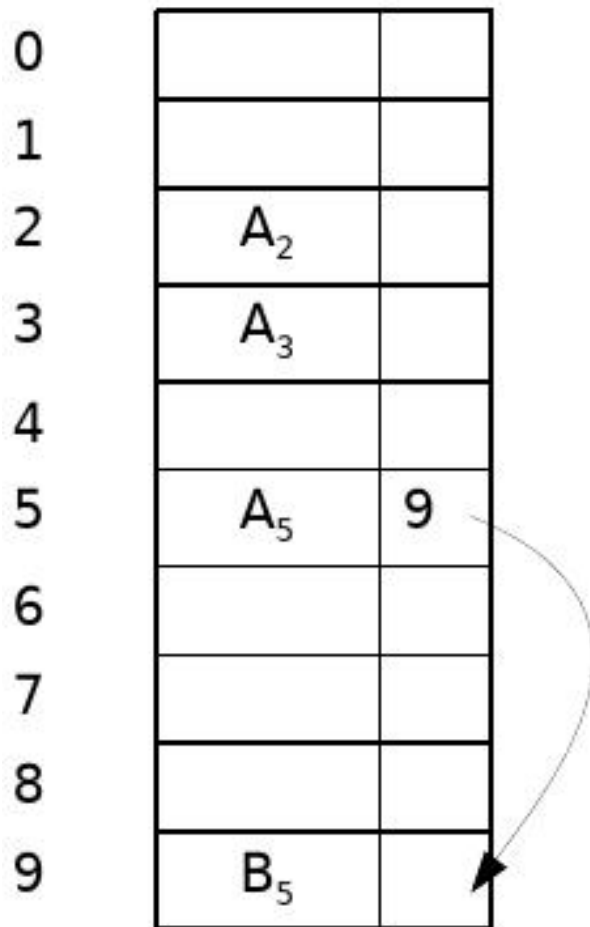
0		
1		
2	$A_2$	
3	$A_3$	
4		
5	$A_5$	
6		
7		
8		
9		

# Exemplo para ilustrar o conceito




- Quando ocorrer a colisão com A5
- Armazena B5 na primeira posição vazia a partir do final da tabela
  - Armazena na posição 9
- Armazena o índice 9 na posição 5

0		
1		
2	A <sub>2</sub>	
3	A <sub>3</sub>	
4		
5	A <sub>5</sub>	9
6		
7		
8		
9	B <sub>5</sub>	



0		
1		
2	$A_2$	
3	$A_3$	
4		
5	$A_5$	9
6		
7		
8	$A_9$	
9	$B_5$	8

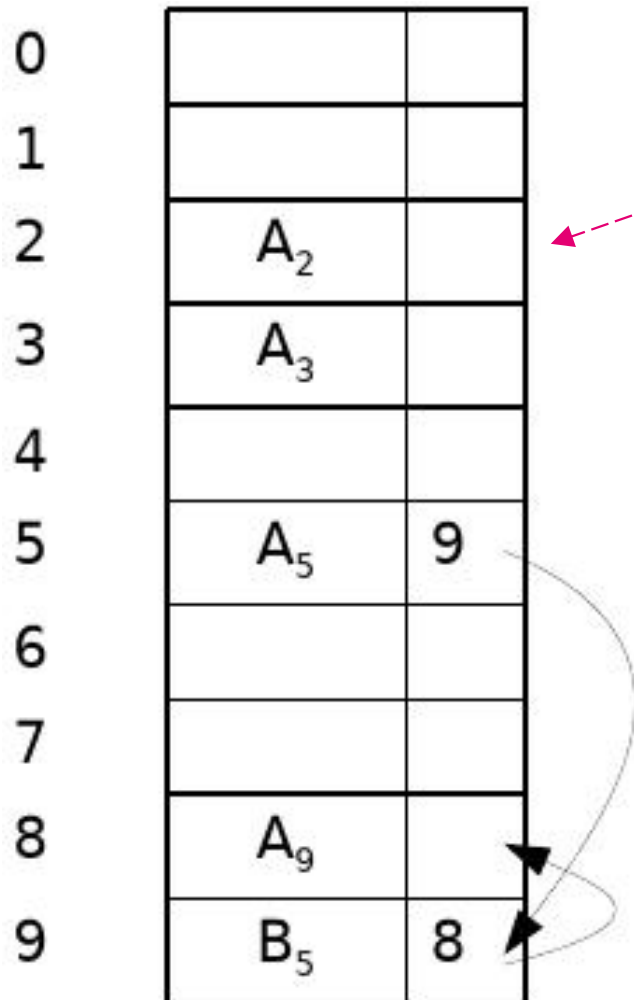


- Agora supondo inserir  $A_9$  e de novo colisão com  $A_5$
- Coloca  $A_9$  na posição vazia no final
  - Armazena em  $A_8$
- O índice 8 armazena na posição 9
  - Pois a posição 5 já está preenchida com a colisão anterior



- Agora supondo inserir B2 e colisão com A2

0		
1		
2	A <sub>2</sub>	
3	A <sub>3</sub>	
4		
5	A <sub>5</sub>	9
6		
7		
8	A <sub>9</sub>	
9	B <sub>5</sub>	8



0		
1		
2	A <sub>2</sub>	6
3	A <sub>3</sub>	
4		
5	A <sub>5</sub>	9
6	B <sub>2</sub>	
7		
8	A <sub>9</sub>	
9	B <sub>5</sub>	8

- Inserir B<sub>2</sub>
- Colisão com A<sub>2</sub>
- Inserir na próxima posição vazia
  - Neste caso, no fim da tabela, porém em ordem alfabética depois do “A<sub>5</sub>”
  - Insere na posição 6
- O índice 6 insere na posição 2

0		-2
1		-2
2	$A_2$	6
3	$A_3$	-1
4		-2
5	$A_5$	9
6	$B_2$	-1
7		-2
8	$A_9$	-1
9	$B_5$	8

- -1 pode ser usado para indicar o final de uma cadeia
- -2 pode ser usado para indicar posições disponíveis

# Outro exemplo

0		-2
1		-2
2	A <sub>2</sub>	6
3	A <sub>3</sub>	-1
4		-2
5	A <sub>5</sub>	9
6	B <sub>2</sub>	-1
7	A <sub>7</sub>	-1
8	A <sub>9</sub>	-1
9	B <sub>5</sub>	8

- Inserir A7
- Inserir C2
  - Colisão com A2
- Na posição 2 o índice armazenado é o 6
- Vamos para a posição 6
  - Já está ocupada
- O -1 indica fim da cadeia

0		-2
1		-2
2	$A_2$	6
3	$A_3$	-1
4		-2
5	$A_5$	9
6	$B_2$	-1
7	$A_7$	-1
8	$A_9$	-1
9	$B_5$	8

- O -1 indica fim da cadeia
- Achar a nova posição vazia
  - A partir do final
  - Posição 4
- O -1 da posição 6 precisa ser atualizado

- Posição 4 vazia

- Insere C2 na posição livre

- Marca como final

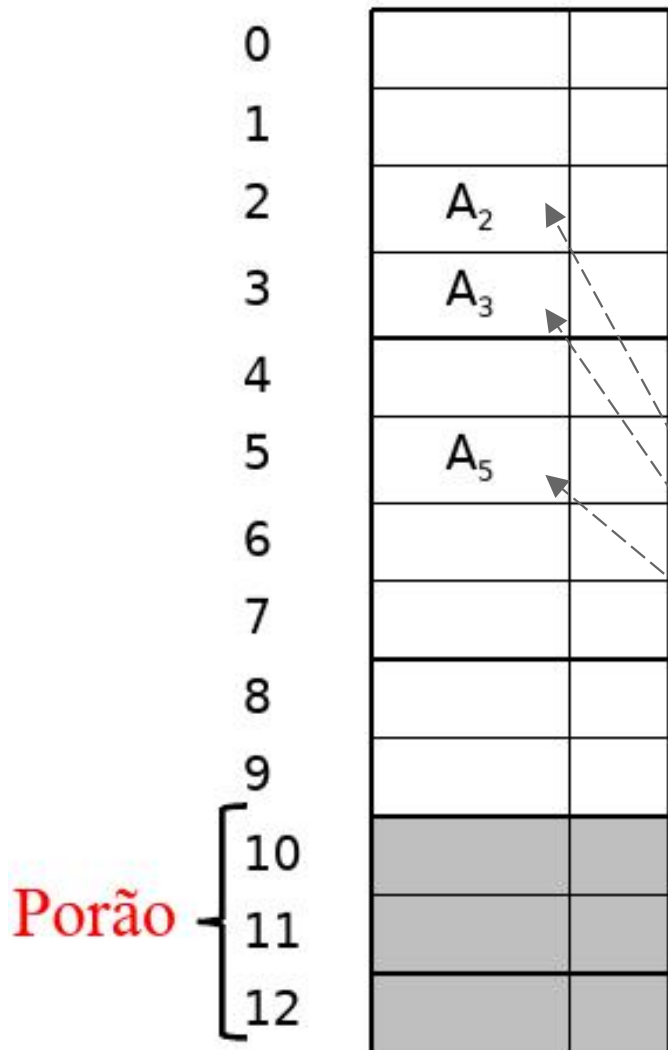
- Montar a cadeia

- A posição 6 é atualizada e aponta para 4

0		-2
1		-2
2	A <sub>2</sub>	6
3	A <sub>3</sub>	-1
4	C <sub>2</sub>	-1
5	A <sub>5</sub>	9
6	B <sub>2</sub>	4
7	A <sub>7</sub>	-1
8	A <sub>9</sub>	-1
9	B <sub>5</sub>	8

Encadeamento coalescido com porão  
(transbordamento)

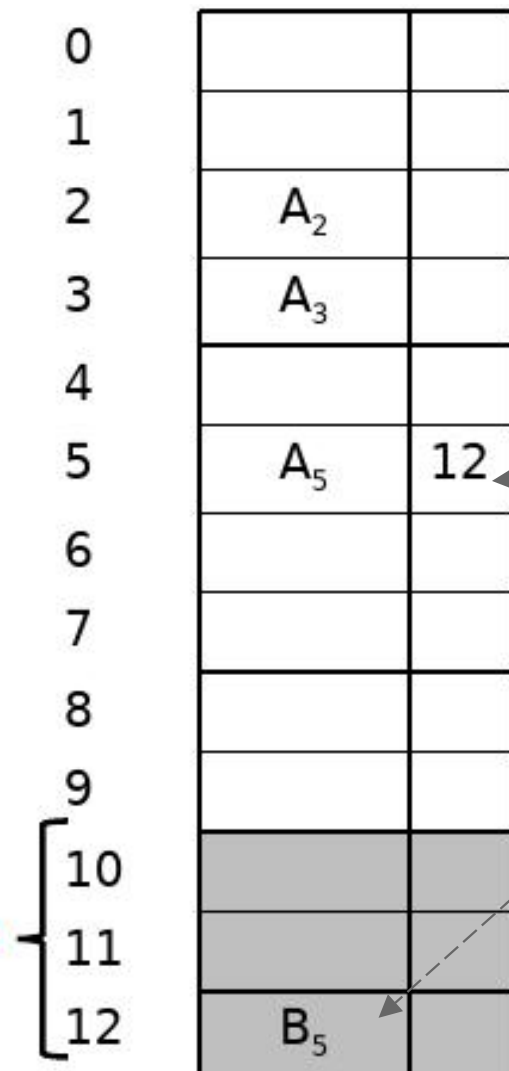
# Área de transbordamento (“porão”)



- Uso de área de transbordamento para armazenar chaves quando não há mais espaço na tabela
- Esta área pode ser alocada dinamicamente
- Exemplo:
  - Inserir  $A_2$ ,  $A_3$  e  $A_5$
  - Supondo inserir  $B_5$  e colisão com  $A_5$
  - Achar posição no porão



# Encadeamento coalescido com porão



- Usa a última posição livre do porão
  - Posição 12
- Armazena  $B_5$  na posição 12
- O índice 12 armazena na posição 5
  - Link entre as posições

# Outro exemplo

0		
1		
2	$A_2$	11
3	$A_3$	
4		
5	$A_5$	12
6		
7		
8		
9		
10		
11	$B_2$	
12	$B_5$	

- Inserir B2 (e colisão com A2)
- Próxima posição livre no porão
  - Posição 11
- Armazena B2 em 11
- Insere o link entre as posições
  - Insere 11 na posição 2

0

1

2

3

4

5

6

7

8

9

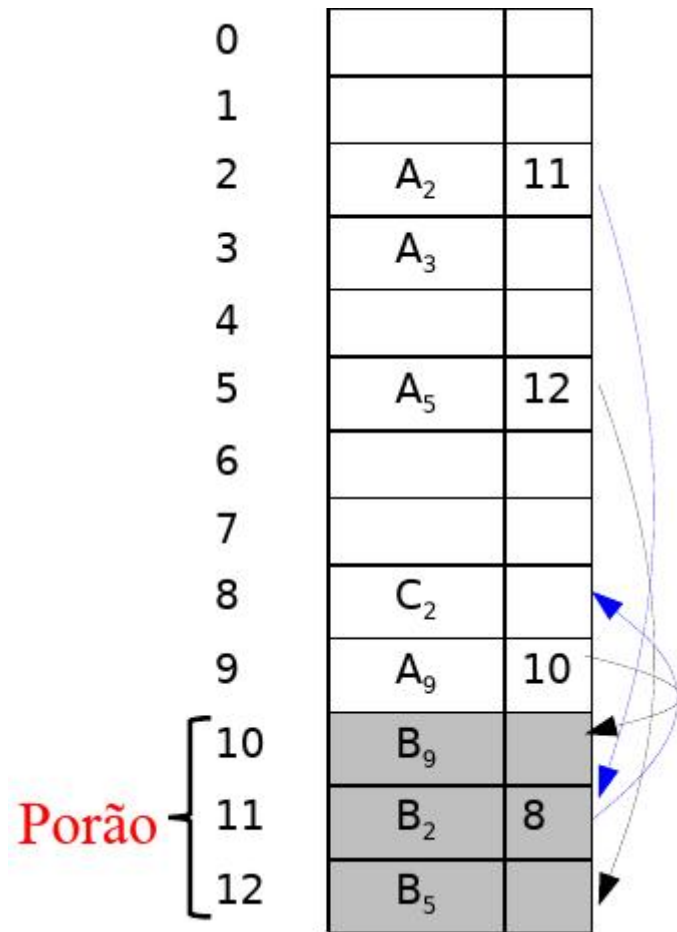
10

11

12

Porão

A <sub>2</sub>	11	
A <sub>3</sub>		
A <sub>5</sub>	12	
C <sub>2</sub>		
A <sub>9</sub>	10	
B <sub>9</sub>		
B <sub>2</sub>	8	
B <sub>5</sub>		



- Inserir A9, B9, e C2

- A9 não tem colisão
  - Insere na 9
- B9 colide com A9
  - Próxima posição livre: 10
  - Insere B9 na posição 10
  - Índice 10 na posição 9
- C2 colide com A2
  - O próximo índice é 11, que colide de novo
  - Próxima posição livre: 8
  - Insere C2 na posição 8
  - Índice 8 na posição 11

0		-2
1		-2
2	$A_2$	11
3	$A_3$	-1
4		-2
5	$A_5$	12
6		-2
7		-2
8	$C_2$	-1
9	$A_9$	10
<b>Porão</b> { <div>10</div> <div>11</div> <div>12</div>	$B_9$	-1
	$B_2$	8
	$B_5$	-1

- -1 para fim de cadeia
- -2 para posição vazia

# Hashing para arquivos extensíveis

Adaptado do material do Prof. Jairo Francisco de Souza

SOUZA, Prof. Jairo Francisco de. Estrutura de Dados II. Hashing para arquivos extensíveis. Universidade Federal de Juiz de Fora. 2012.

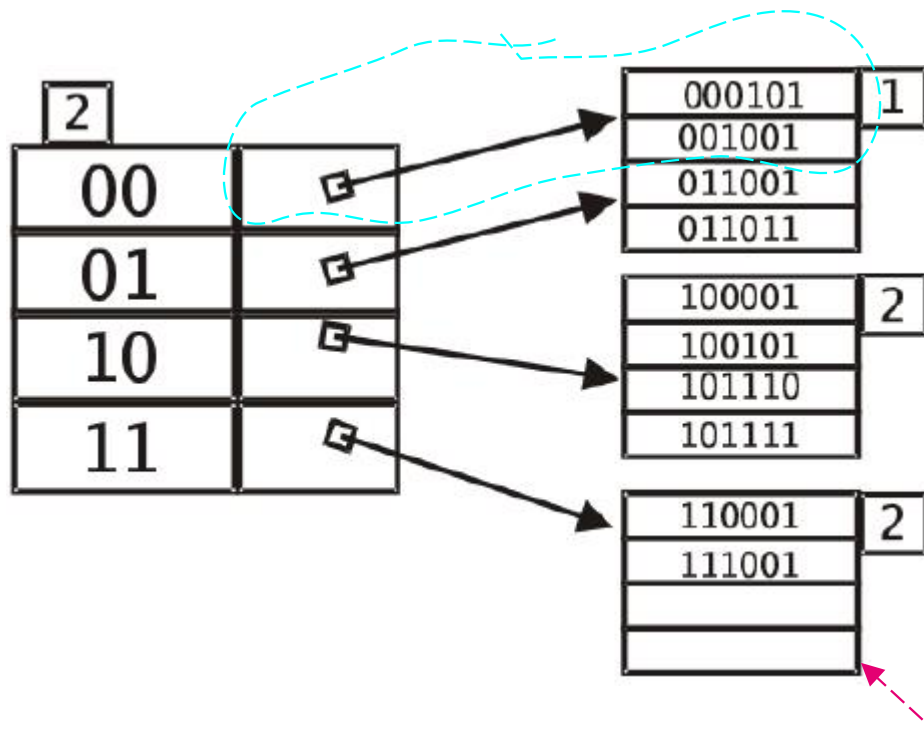
# Hashing para arquivos extensíveis

- Os métodos anteriores usam tamanho fixo para alocação das chaves
- Existem várias técnicas propostas para hashing em arquivos com tamanhos variáveis, com inserções e deleções constantes
- Podem ser divididas em
  - Técnicas que usam diretórios
    - hashing expansível (Knott, 1971)
    - hashing dinâmico (Larson, 1978)
    - hashing extensível (Fagin et al, 1979)
  - Técnicas que não usam diretórios
    - hashing virtual (Litwin, 1978)
    - hashing linear (Litwin, 1980)

# Hashing extensível

- Situação: balde (página primária) fica cheio
- Porque não reorganizar o arquivo, duplicando o número de baldes?
  - Ler e escrever todas as páginas (baldes) é dispendioso
- Ideia: usar um diretório de ponteiros para baldes
  - Duplicar o diretório de ponteiros → duplicar o nº de baldes
  - Particionando justamente o balde que transbordou!
- Um diretório de ponteiros é muito menor que um arquivo
  - Duplicá-lo é menos dispendioso
  - Só uma página de verbetes de dados é particionada
- O truque reside em como a função de hashing é ajustada

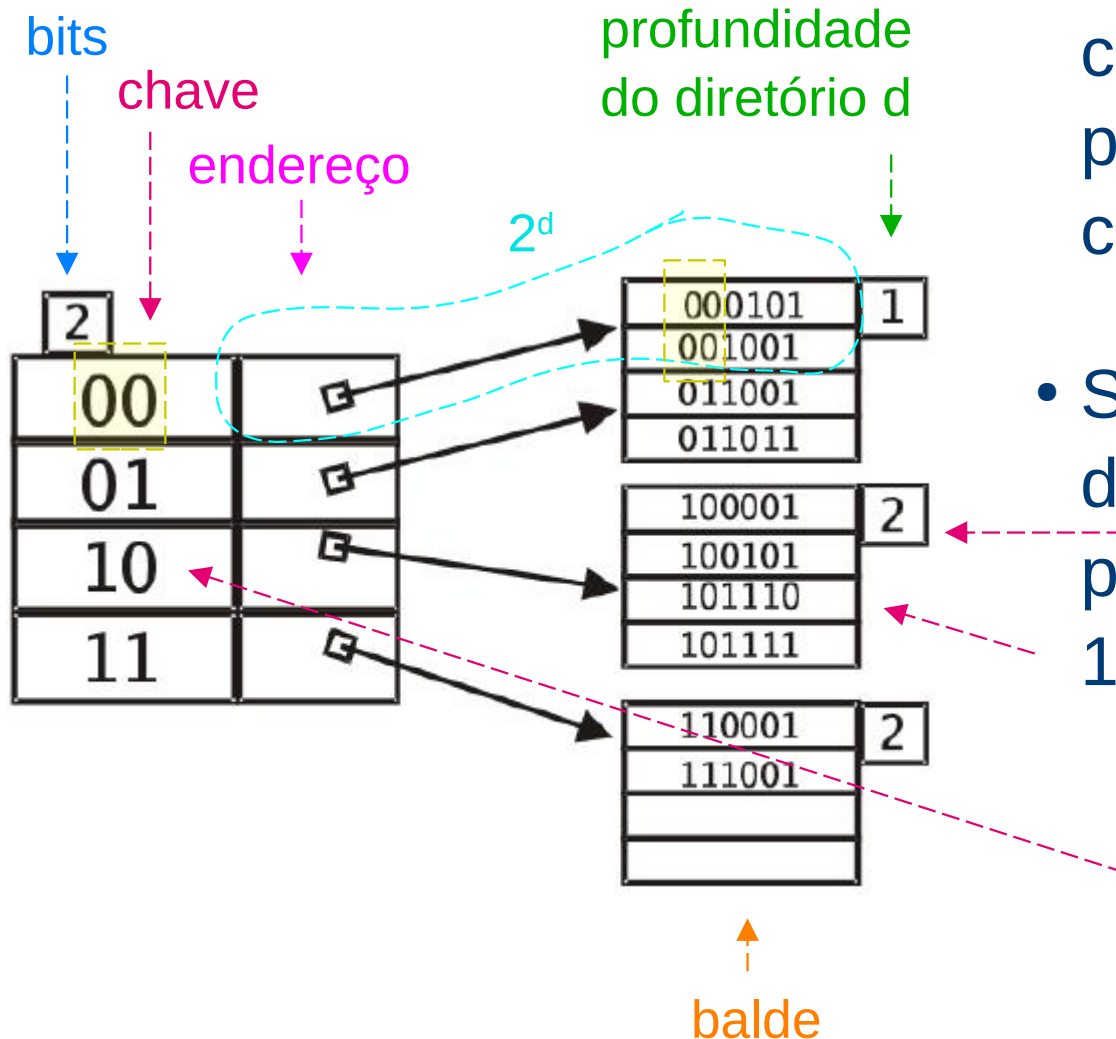
# Hashing extensível



- O diretório contém informações que indicam a localização do balde
- um diretório é um arranjo de  $2^d$  endereços de baldes
- $d$  é a profundidade do diretório
- O tamanho de cada balde é fixo

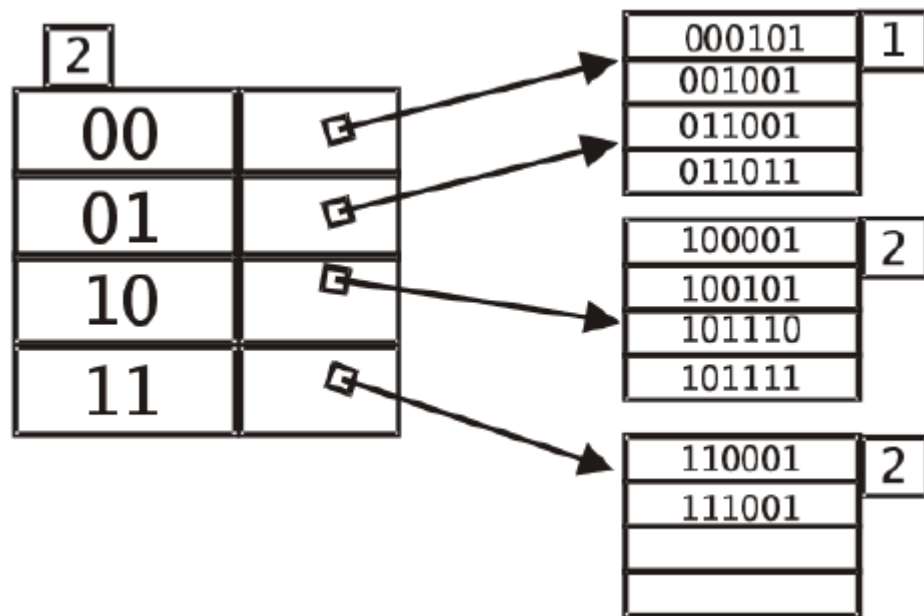


# Hashing extensível



- Utiliza o valor inteiro correspondente aos primeiros  $d$  bits de uma chave
- Supondo a profundidade do diretório igual a 2 e a pseudo-chave igual a 101110
  - Procuraremos na posição 10 pelo endereço do balde que contém a informação desejada

# Hashing extensível



- É possível ter mais de uma entrada no diretório apontando para o mesmo balde
- Cada balde armazena a informação de sua profundidade
- A profundidade do balde é chamada de profundidade local
- A profundidade local indica quantos primeiros bits de cada pseudo-chave são comum a todos os elementos do balde
- A profundidade do balde pode ser menor ou igual à profundidade do diretório

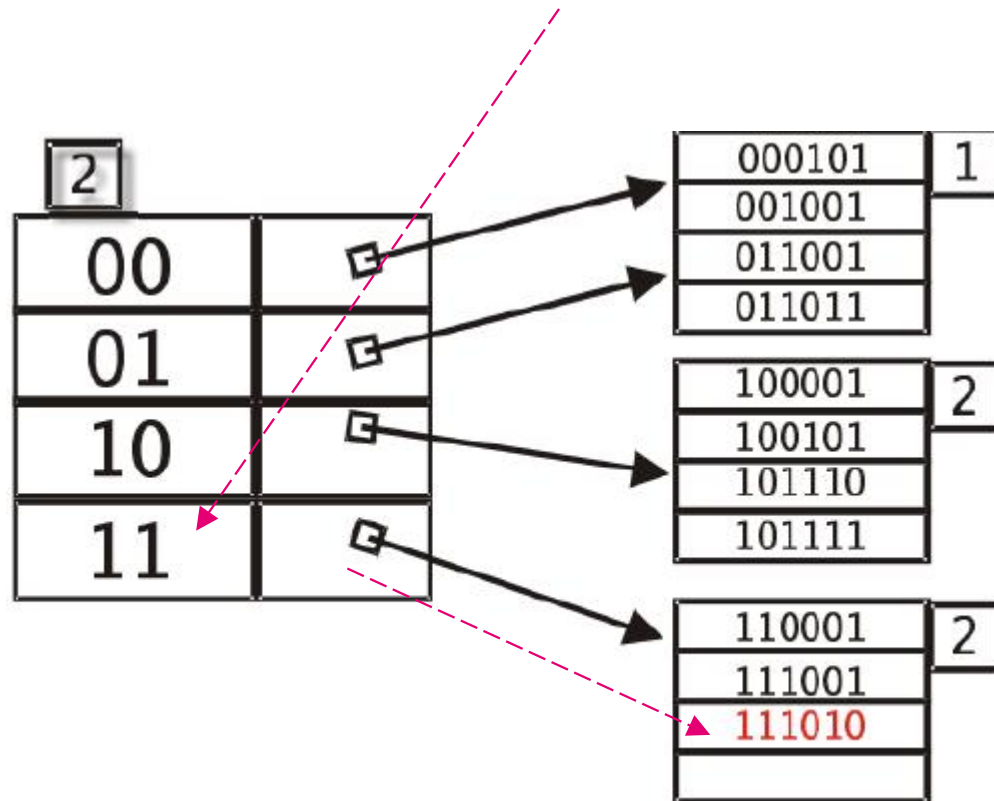
# Hashing extensível

- Quando necessário, um diretório pode ser expandido
  - Isto ocorre quando “estoura” o espaço de endereços do diretório
  - Os baldes estão cheios
- Neste caso, dobramos de tamanho o diretório
  - Ou seja, a profundidade é incrementada em 1
- Inserção:
  - Após obtida a pseudo-chave, pega-se os **p** primeiros dígitos mais significativos da chave como índice para acessar uma posição
  - Por exemplo, se a pseudo-chave é 10100110 e a profundidade é 3, acessaremos a posição 101
  - Daí, teremos 3 casos durante a inserção...

# Hashing extensível - inserção

- Melhor caso

- Há espaço no balde
- A pseudo-chave é simplesmente inserida
- Ex.: inserção da pseudo-chave 111010:



# Hashing extensível - inserção

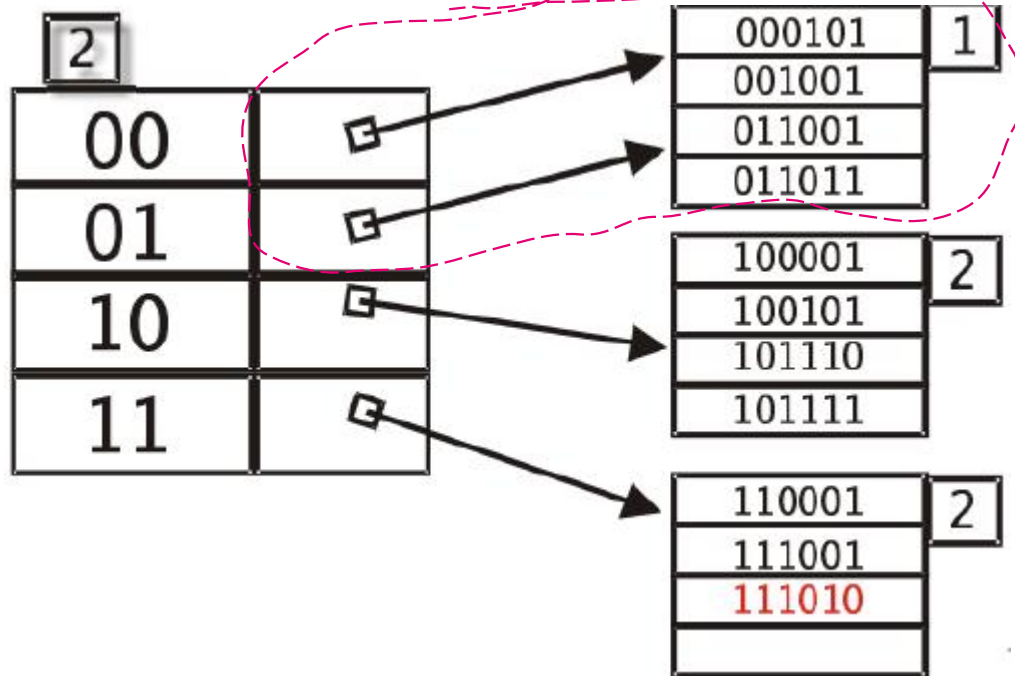
- Caso intermediário

- Não há espaço no balde encontrado
- Mas ele pode ser dividido
  - Ex.: de  $d=1$  passa para  $d=2$  (vide próx. slide)
- Mais de uma entrada no diretório faz referência ao balde no qual estamos tentando inserir
  - Ex.: 00 e 01 fazem referência ao mesmo balde
- O balde é dividido, um fica com 00 e o outro com 01
- As pseudo-chaves são redistribuídas entre os baldes
- A profundidade local dos dois baldes serão iguais à do balde original incrementado de 1

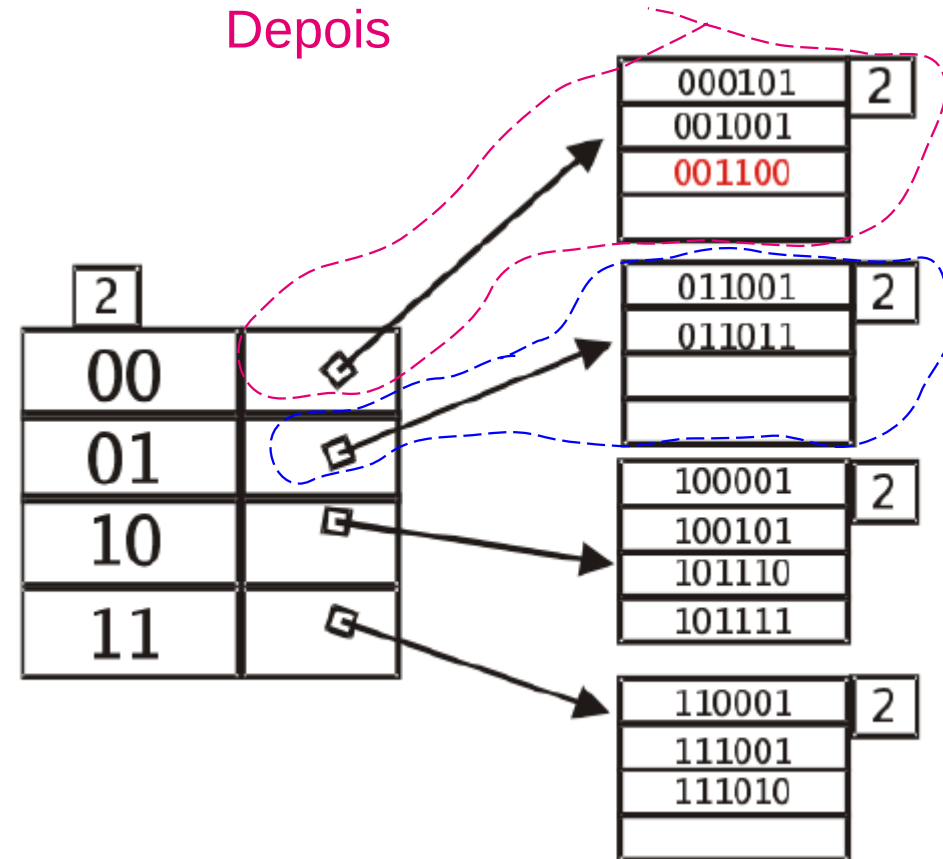
# Hashing extensível - inserção

- Caso intermediário
- Ex.: inserção de 001100

Antes



Depois

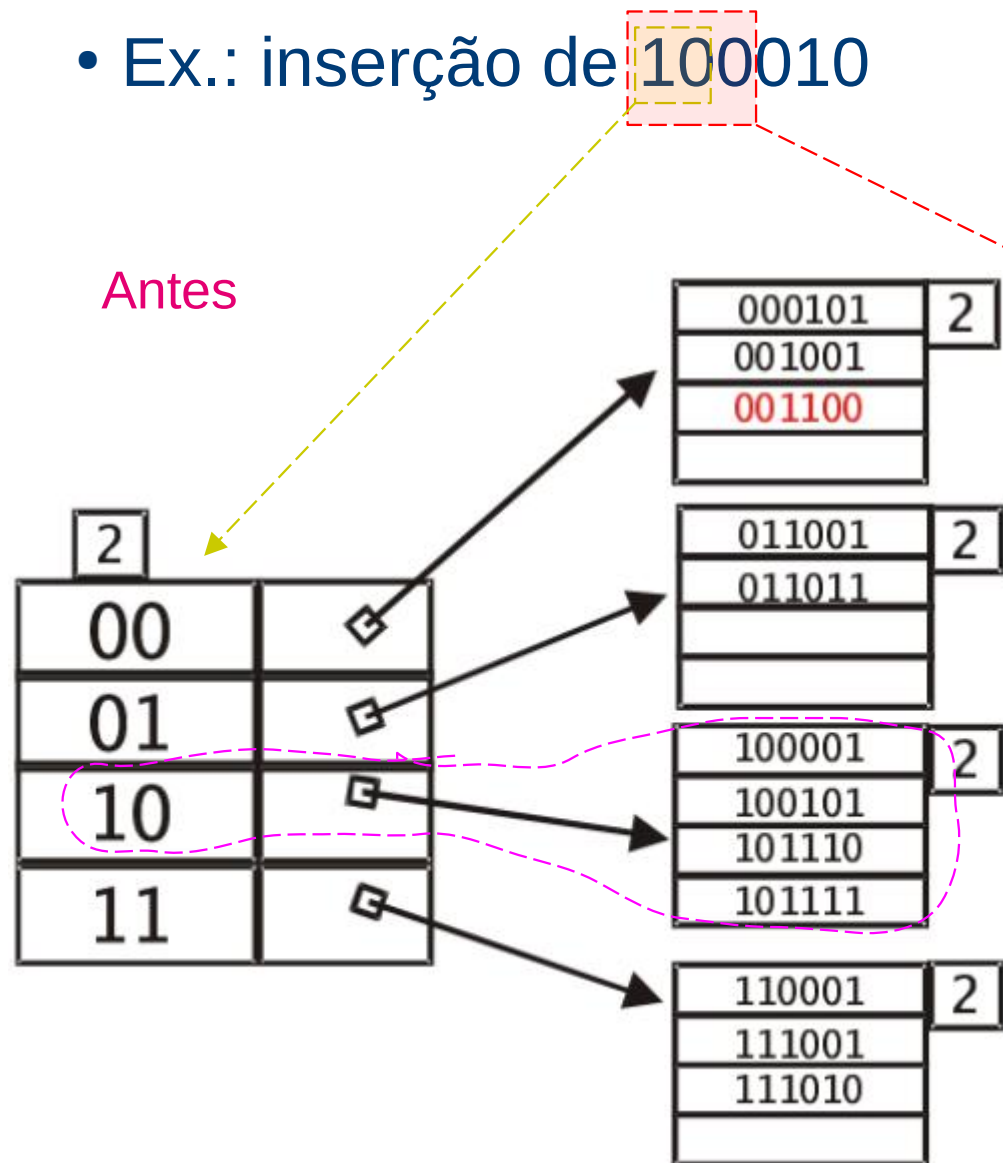


# Hashing extensível - inserção

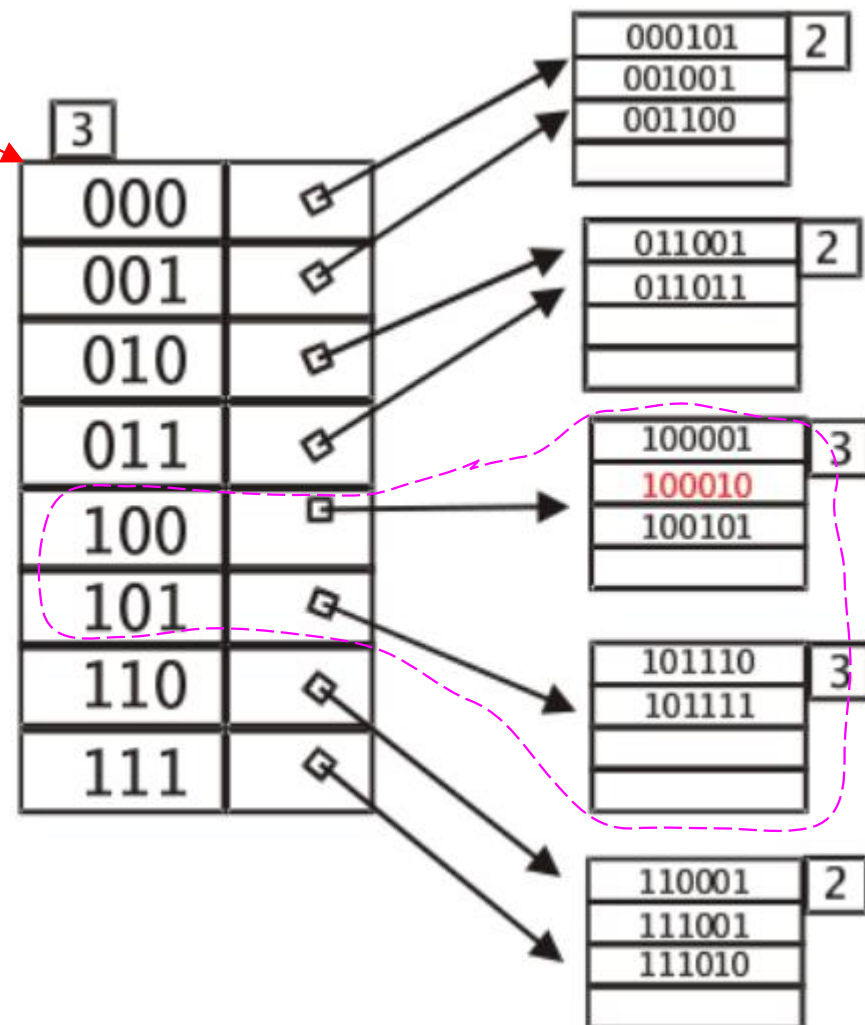
- Pior caso
  - Não há espaço no balde
  - Entrada no diretório faz referência única e exclusivamente a um balde
  - Ou seja, profundidade local é igual a profundidade global
  - O diretório dobra de tamanho
  - Ou seja, profundidade global é incrementada de 1
  - As pseudo-chaves são distribuídas entre os baldes do novo diretório

- Ex.: inserção de 100010

Antes



Depois



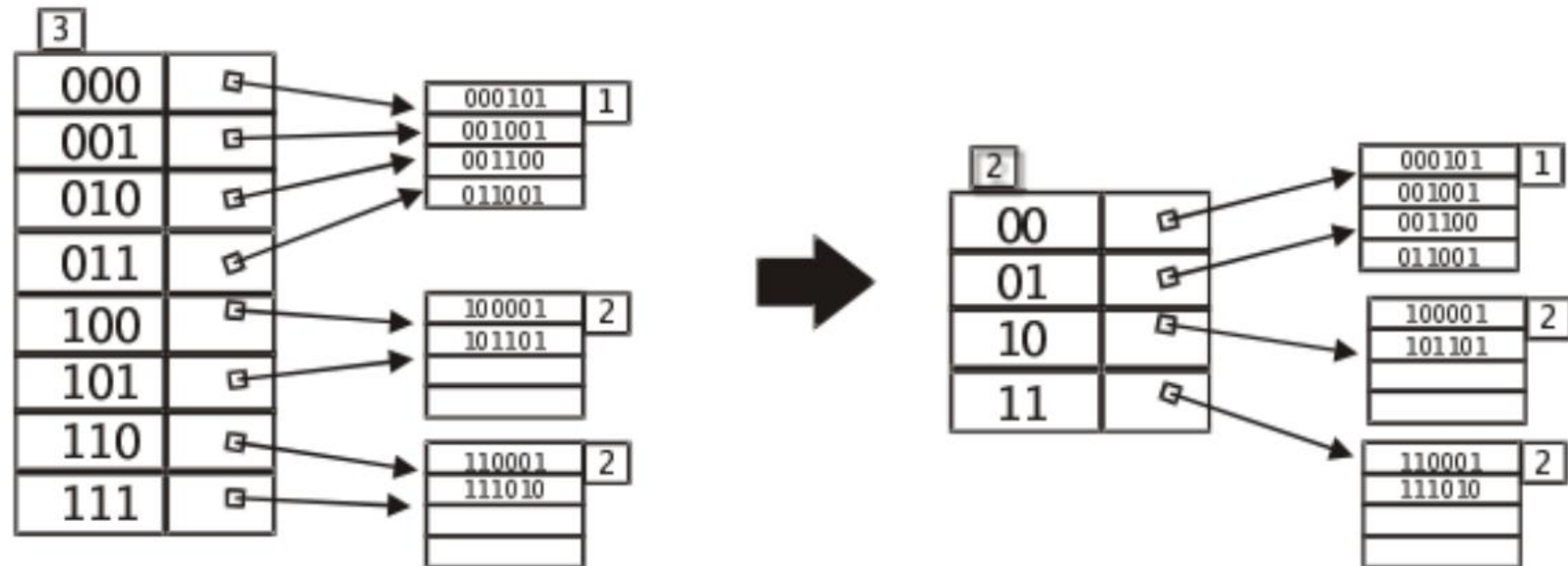


# Hashing extensível - remoção

- Ao remover uma entrada, temos que fazer duas verificações
  - Se é possível fundir o balde com um balde amigo
  - Se o tamanho do diretório pode ser reduzido
- Encontrando um balde amigo
  - Para haver um único balde amigo, a profundidade do balde tem que ser a mesma do diretório
  - Dado um índice para o balde, o balde amigo é aquele que difere apenas no último bit do índice
  - Ex.: se um balde tem índice 000 e sua profundidade é 3 e igual a do diretório, o balde amigo tem índice 001
  - Quando fundir?
    - Critério a cargo do programador
    - Pode-se sempre tentar fundir os baldes
    - Pode-se definir um número mínimo de informação em cada balde

# Hashing extensível - remoção

- Reduzindo o diretório
  - Depois de fundir o balde, talvez o diretório possa ser reduzido
    - É possível quando cada balde é referenciado por pelo menos duas entradas do diretório
      - Isto é, todo  $p$  local é menor que o  $p$  global



# Referências

SOUZA, J. F. de. Hashing - Resolução de Colisões. [S. l.]: Universidade Federal de Juiz de Fora, 2012a.

SOUZA, J. F. de. Hashing para Arquivos Extensíveis. [S. l.]: Universidade Federal de Juiz de Fora, 2012b.