Indexação & Hashing

Bianca Maria Pedrosa

Índice

Organização de Arquivos

Conceitos Básicos

- Arquivo Lógico
  - é o arquivo visto do ponto de vista da aplicação. Cada arquivo manipulado pela aplicação é tratado como se um canal de comunicação estivesse estabelecido entre a aplicação e o arquivo em disco

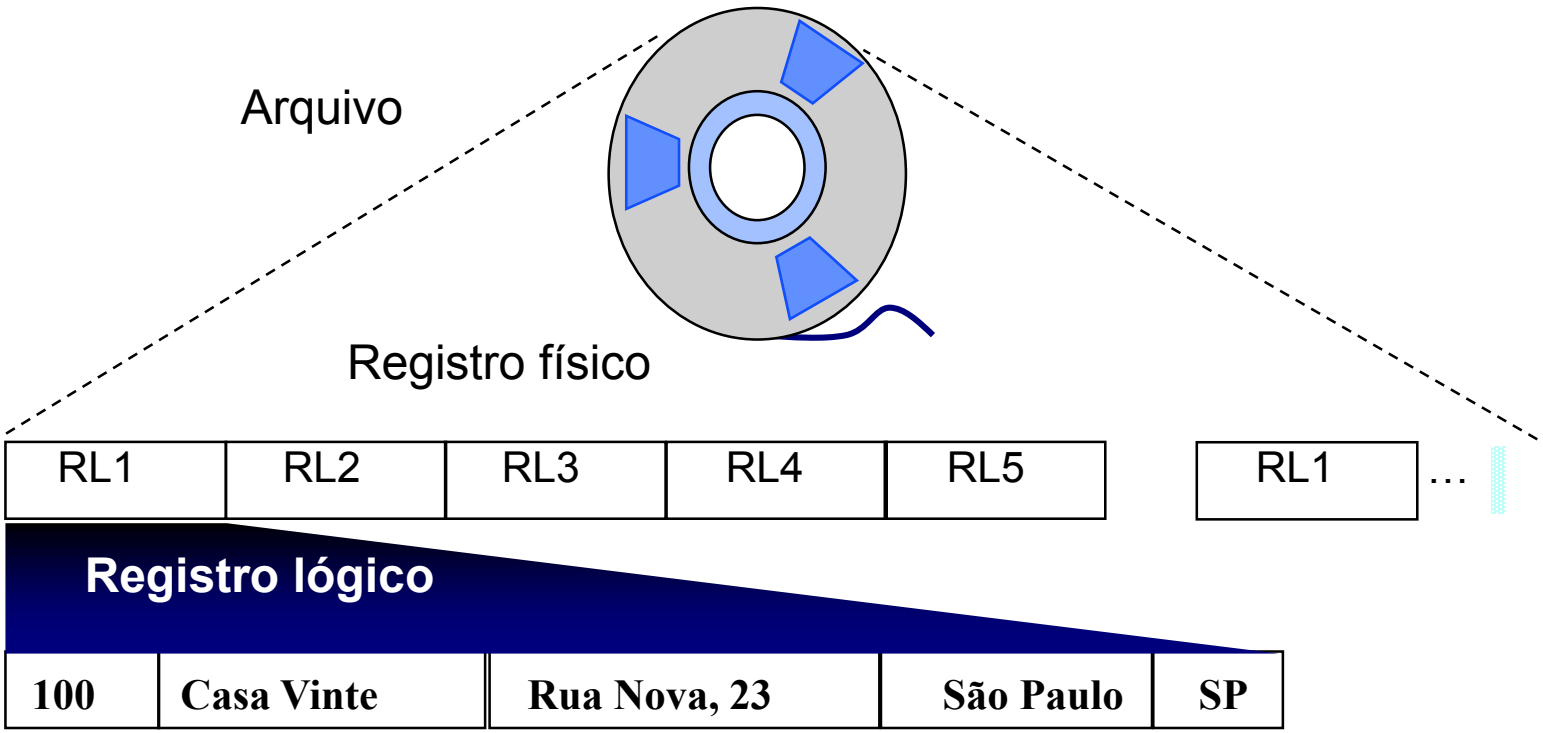
Conceitos Básicos

- Arquivo Físico
  - Refere-se ao modo como a estrutura Lógica é armazenada no meio físico
  - Quando nos referimos a um dado arquivo no meio físico, nos referimos a uma particular seqüência de bytes armazenada neste meio

Conceitos Básicos

- Arquivo Lógico X Arquivo Físico
  - cada arquivo manipulado pela aplicação é tratado como se um canal de comunicação estivesse estabelecido entre a aplicação e o arquivo no meio Físico
  - o espaço necessário para o armazenamento do arquivo depende de seu tamanho Lógico (a soma dos tamanhos de todos os registros) e do modo de armazenamento no meio Físico

Arquivo de Dados



Meios Físicos de Armazenamento

- Primários:
  - operados pela CPU
  - memória principal e memórias cache
  - rápidos
  - capacidade limitada de memória
  - caros

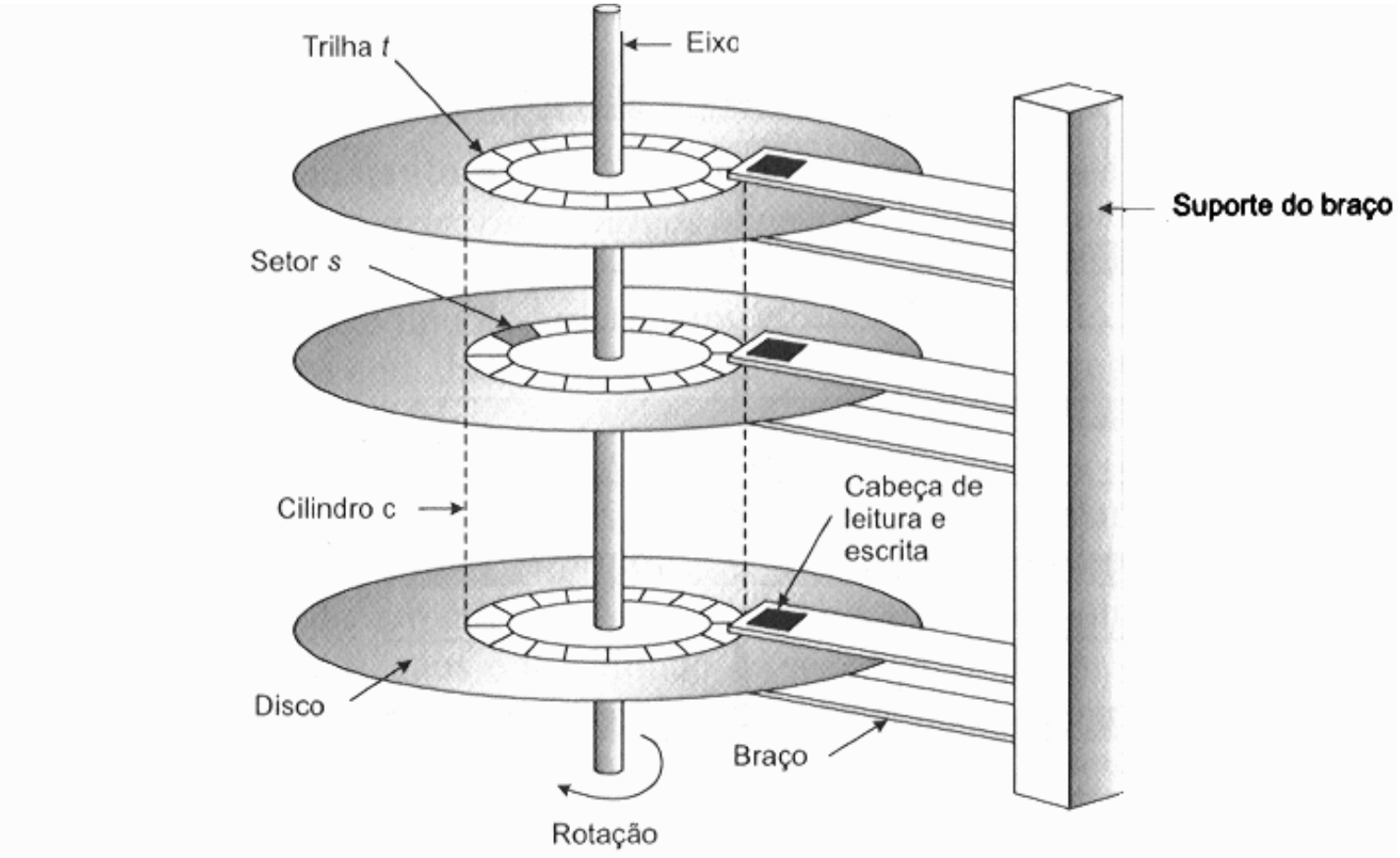
Meios Físicos de Armazenamento

- Secundários e Terciários
  - discos, CDs, etc.
  - grande capacidade de memória
  - lentos
  - não são operados diretamente pela CPU; devem ser copiados para o dispositivo primário.

Discos Magnéticos

- Dispositivos de acesso direto
- Informação é gravada em uma ou mais superfícies e armazenada em trilhas na superfície.
- Cada trilha é normalmente dividida em setores.
- Setor é a menor porção endereçável do disco.
- Quando temos um disk pack (vários pratos), um cilindro é formado pelo conjunto de trilhas na mesma direção. Uma vantagem é que toda a informação de um cilindro pode ser obtida sem movimentação da cabeça de leitura e escrita.

Mecanismo de Movimentação da cabeça de leitura e gravação



Jornada de um byte

O que ocorre quando um programa escreve um byte para um arquivo em um disco?

- O programa pede ao SO para escrever o conteúdo da variável **c** na próxima posição do arquivo **arq.**
- O SO repassa a tarefa para o **gerenciador de arquivos (GA)**, que é um subconjunto do SO.
- O GA busca em uma tabela informações sobre o arquivo.
- O GA busca em uma FAT a localização física do setor que deve conter o byte (cilindro, trilha, setor).
- O GA instrui o processador de I/O sobre a posição do byte na RAM, e onde ele deve ser colocado no disco.
- O processador de I/O formata o dado apropriadamente, e decide o melhor momento de escreve-ló no disco.
- O processador de I/O envia o dado para o controlador de disco.
- O controlador de disco instrui o drive de disco para mover a cabeça de R/W para a trilha correta, espera o setor desejado ficar sob a cabeça, e então envia o byte para o drive de disco o qual deposita, bit-a-bit, o byte na superfície do disco.

Medidas de Desempenho de Discos

- Tempo de acesso
  - tempo gasto desde o pedido de leitura/escrita até o início da transferência de dados
- Tempo de procura
  - tempo para reposicionamento do braço
- Tempo de latência
  - tempo de espera para que o setor a ser acessado fique sob a cabeça de leitura/gravação
- Taxa de Transferência de dados
- Tempo de falha

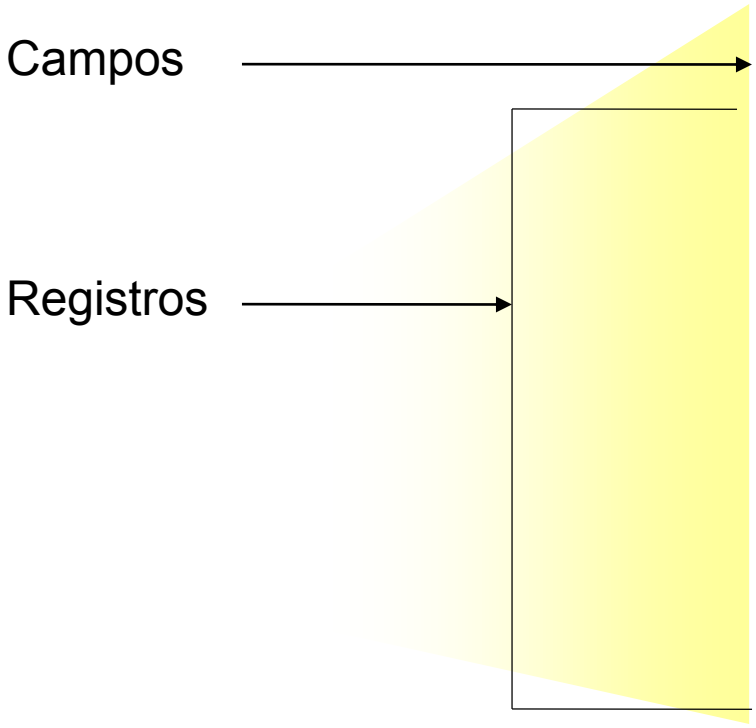
Organização de Arquivos

- A organização de arquivo trata do arranjo ou a forma de distribuição dos registros dentro do arquivo, objetivando agilizar o processo de armazenamento e recuperação de dados.
- Um arquivo é organizado, logicamente, como uma seqüência de registros que são mapeados nos dispositivos de memória auxiliar

Organização de arquivos

- Registros
  - estrutura que agrega dados de vários tipos
  - pode ser usada para representar uma entidade
  - todos os atributos que descrevem a entidade devem estar representados
- Campo
  - é a unidade de informação do registro
  - cada campo representa um atributo
  - cada campo tem um nome, um tipo e um tamanho

Organização básica de arquivos



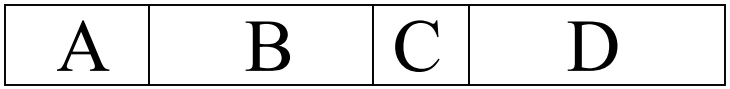
E	Ch	Nome	Curso
1	100	Ana	Bcc
2	101	João	Adm
3	105	Maria	Bcc
4	106	José	Eco
5	110	Pedro	Bio
6	111	Rosa	Ped
7	112	Carlos	Med
8	113	Valdir	Eco
9	115	Paulo	Med

Organização básica de arquivos

- Os registros podem assumir tamanhos fixos e variáveis:
  - Tamanho fixo – todos os registros do arquivo possuem o mesmo comprimento em bytes;
  - Tamanho variável – os registros podem assumir um comprimento entre dois limites especificados, máximo e mínimo.

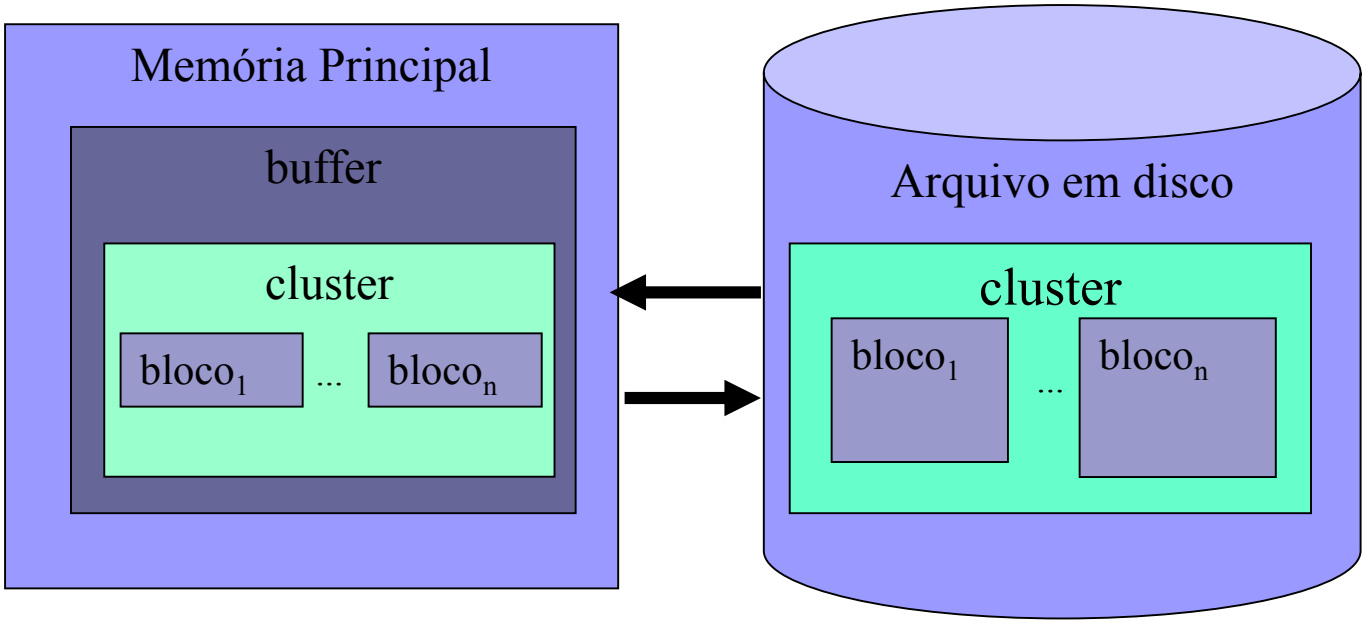
Registro de tamanho variável

Registro com campo de tamanho variável



Registro com n° variável de ocorrências

Acesso a Discos



Organização Seqüencial Desordenada

- Arquivos Desordenados (Heap Files)
  - Inclusão
    - Inserir novos registros no final do arquivo
  - Pesquisa
    - Busca Seqüêncial
  - Exclusão
    - Encontrar o bloco do registro a ser apagado, copiá-lo em buffer, excluir o registro, reescrever o bloco

Organização Seqüencial Ordenada

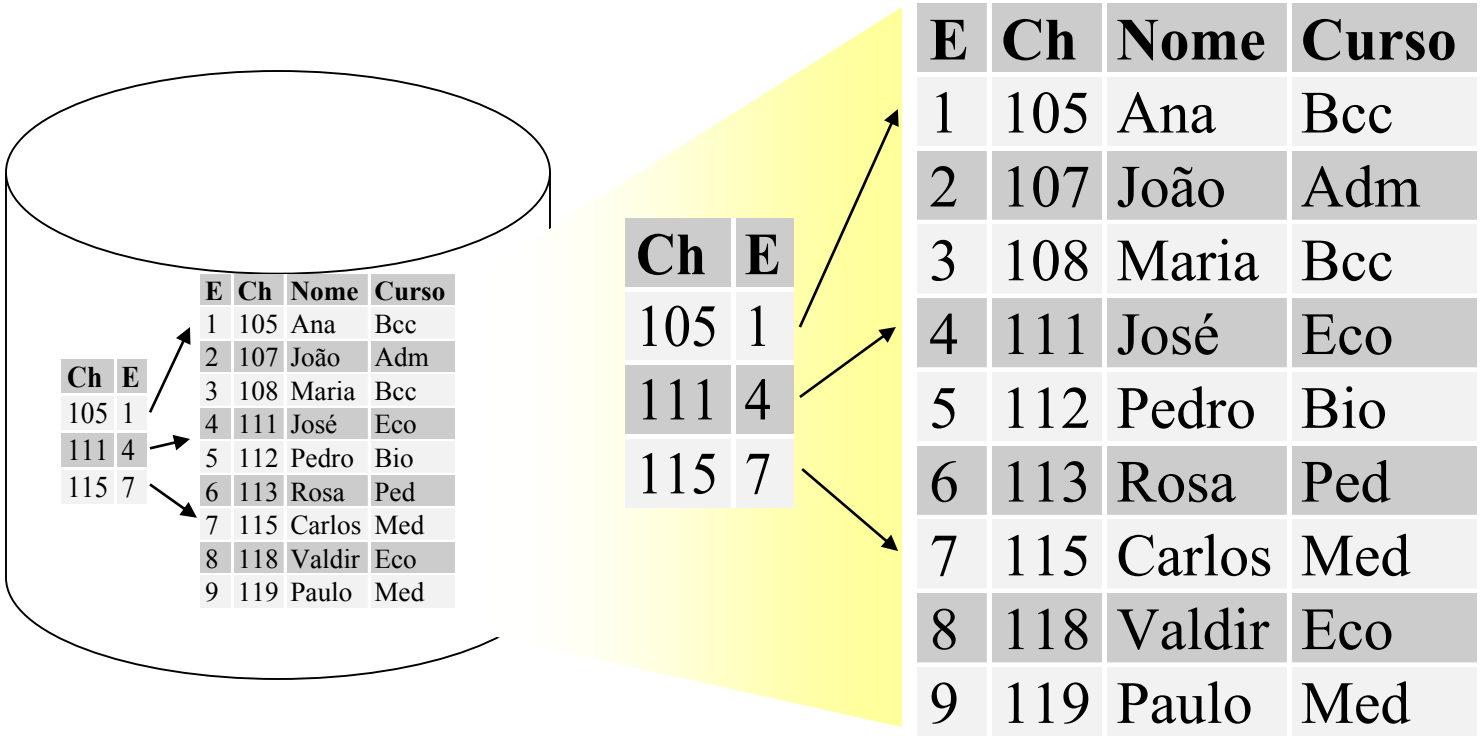
- Arquivos Ordenados (*Sorted Files*)
  - Inclusão de um registro
    - Localizar a posição no arquivo em que o registro deve ser inserido, abrir espaço, inserir o registro na posição.
  - Pesquisa
    - Busca Binária
  - Exclusão de um registro
    - Marcar o registro logicamente para exclusão localizar o registro e apagá-lo, deixando a posição livre. Ambos requerem uma posterior reorganização do arquivo

Indexação

Índices

- O que são índices?
  - Estruturas adicionais que associamos ao arquivo de dados para localizar mais diretamente os registros desejados

Índices



Ch	E
105	1
111	4
115	7

E	Ch	Nome	Curso
1	105	Ana	Bcc
2	107	João	Adm
3	108	Maria	Bcc
4	111	José	Eco
5	112	Pedro	Bio
6	113	Rosa	Ped
7	115	Carlos	Med
8	118	Valdir	Eco
9	119	Paulo	Med

Ch	E
105	1
111	4
115	7

E	Ch	Nome	Curso
1	105	Ana	Bcc
2	107	João	Adm
3	108	Maria	Bcc
4	111	José	Eco
5	112	Pedro	Bio
6	113	Rosa	Ped
7	115	Carlos	Med
8	118	Valdir	Eco
9	119	Paulo	Med

Índices

- Tipos?
  - Ordenados
    - baseiam-se na ordenação de valores
  - Hash
    - baseiam-se na distribuição uniforme de valores por meio de uma faixa de buckets

Critérios para escolhas de Índices

- tempo de acesso
- tempo de inserção
- tempo de exclusão
- sobrecarga de espaço

Índices ordenados

- Vantagens:
  - Índices possuem registros de tamanho fixo
  - Índices são menores do que arquivo de dados.
  - Podem ser percorridos por **Busca Binária**

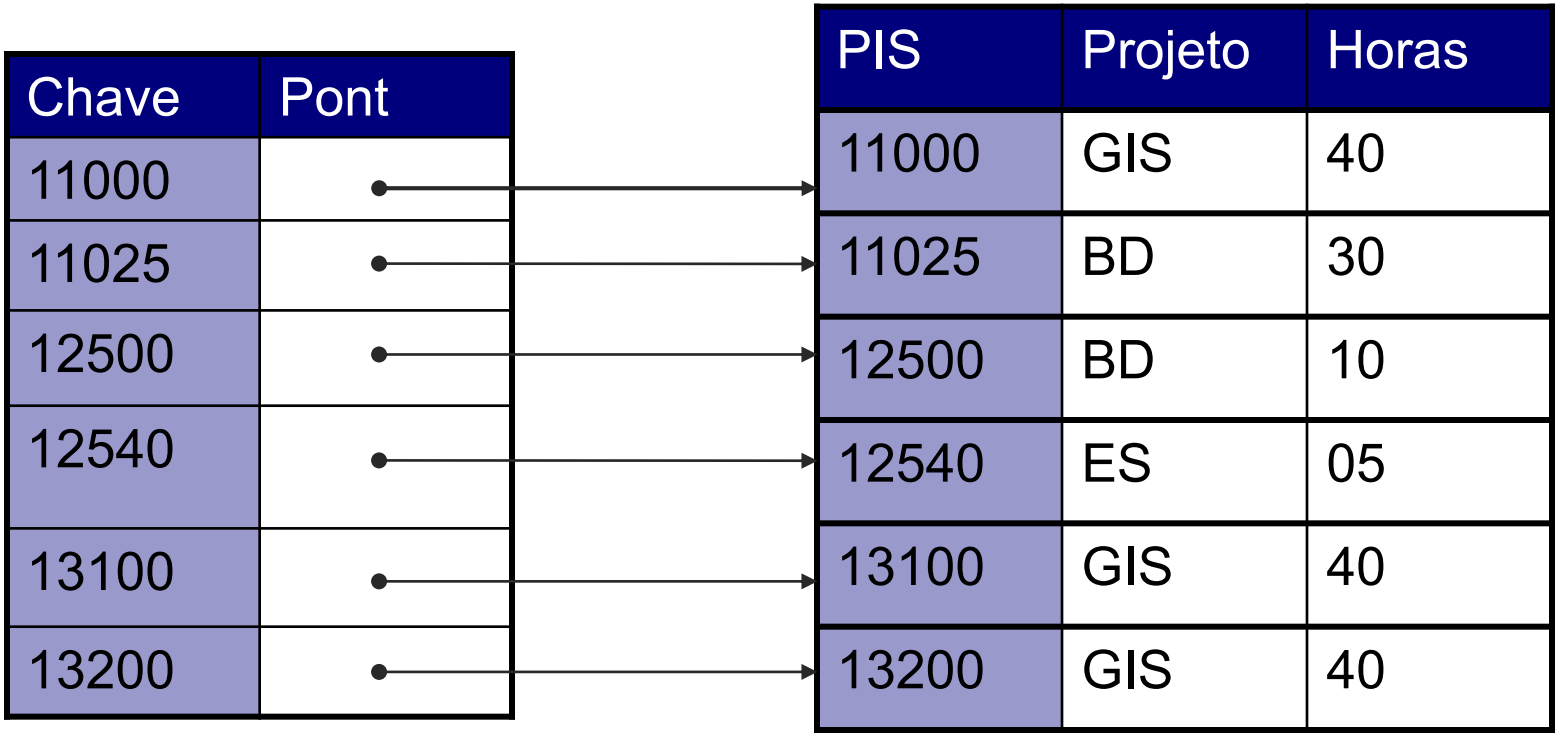
Algoritmo Busca Binária

```
inicio<- 1; fim <- nro_blocos;
Enquanto  ( inicio <= fim )
{  corrente <- ( inicio+ fim) div 2;
   se (k = campo_chave)
         então retorne campo_chave
   se (K< campo_chave)
         então fim  <- corrente – 1
   senão inicio <- corrente+1
}
```

Tipos de Índices ordenados

- Densos
  - Aplicam-se a arquivos não ordenados
  - Possuem um registro para cada valor diferente no campo chave
- Esparsos
  - Aplicam-se apenas a arquivos ordenados
  - Possuem um registro para cada bloco

Índices Ordenados Densos

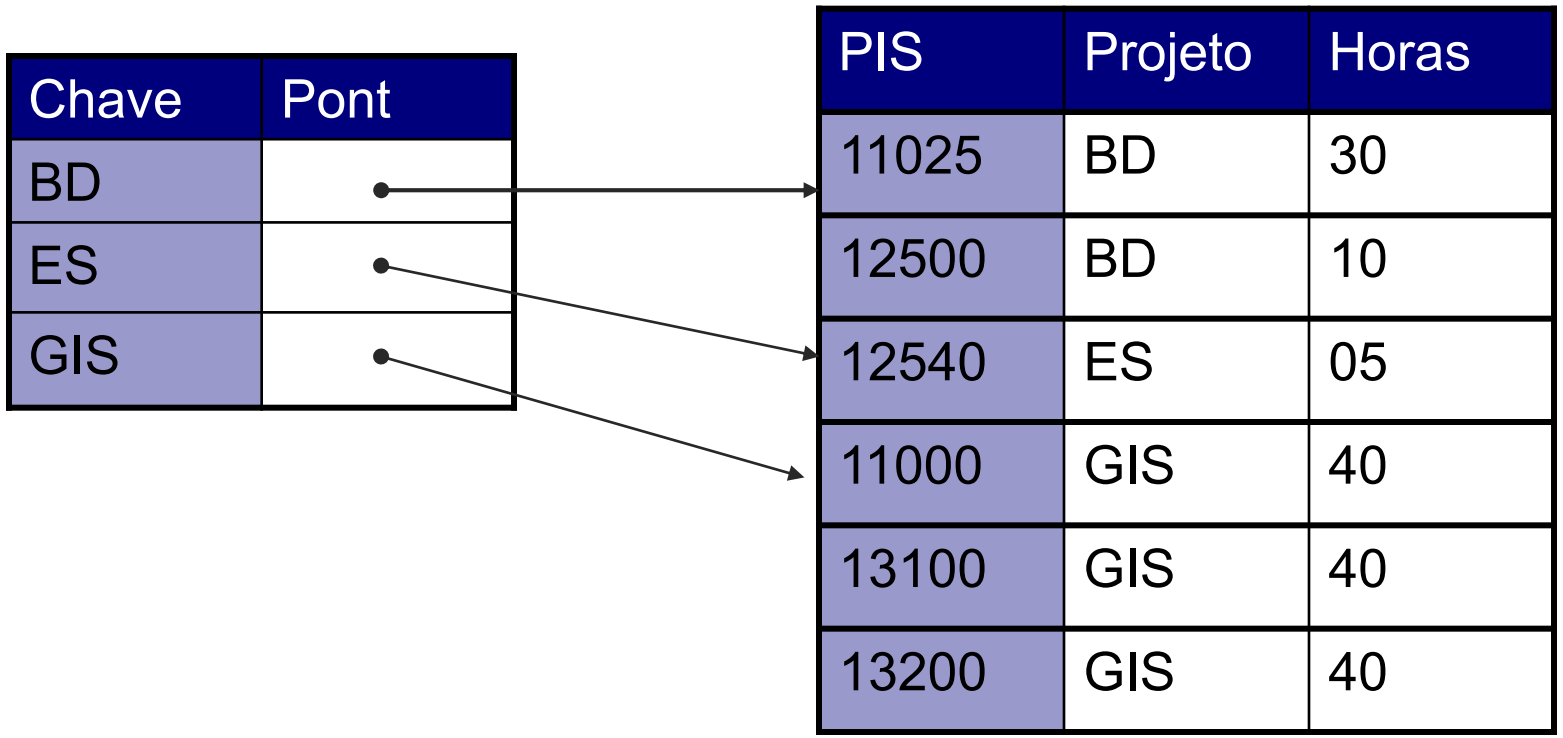


Chave	Pont
11000	
11025	
12500	
12540	
13100	
13200	

PIS	Projeto	Horas
11000	GIS	40
11025	BD	30
12500	BD	10
12540	ES	05
13100	GIS	40
13200	GIS	40

Existe um registro de índice para cada registro de dado.
Geralmente, índices densos são definidos sobre atributos chave.
O arquivo de dados não precisa estar ordenado.

Índices Ordenados Esparsos



Chave	Pont
BD	
ES	
GIS	

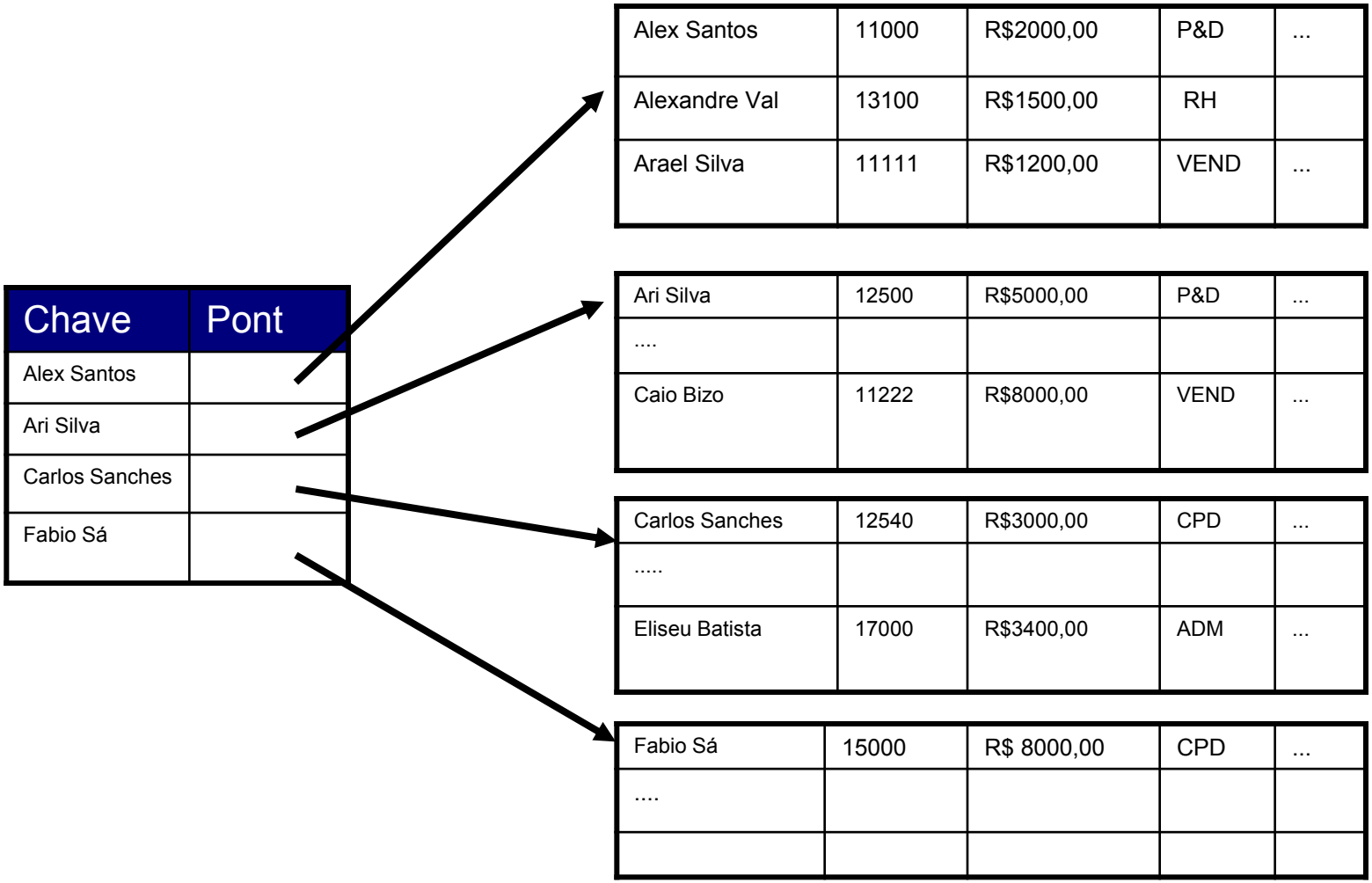
PIS	Projeto	Horas
11025	BD	30
12500	BD	10
12540	ES	05
11000	GIS	40
13100	GIS	40
13200	GIS	40

Não existe um registro de índice para cada registro de dado.
Geralmente, índices esparsos são definidos sobre atributos não chave e requerem que o arquivo de dados esteja ordenado

Índices Primários

- Definido sobre a chave primária de um arquivo de dados ordenado por esta mesma chave
- Índice esparso (um registro p/ cada bloco)
- Cada registro tem o valor do campo da chave primária do 1º registro do bloco e um ponteiro para este bloco.
- Para inserir um registro na sua posição correta no arquivo de dados é necessário, além de mover os registros para abrir espaço para o novo registro, alterar alguns registros âncora (primeiro registro de um bloco)

Índices Primários



Se o arquivo de dados já está ordenado, para que utilizar um índice primário ?

Para que usar índice primário?

- Características do arquivo de dados:
  - Número total de registro = 30000
  - Tamanho do registro = 100 bytes
  - Bloco de disco = 1024 bytes
  - Fator de bloco = nro de registros existente em cada bloco de disco
    - FB= tam. do bloco/tam. do registro
    - FB= 1024/100 =10 registros/bloco
  - Número de blocos necessários para armazenar o arquivo todo
    - NB= número total de registros / fator de bloco
    - NB= 30000/10= 3000 blocos.
  - Busca binária = 12 acessos a blocos (log2 3000=12)

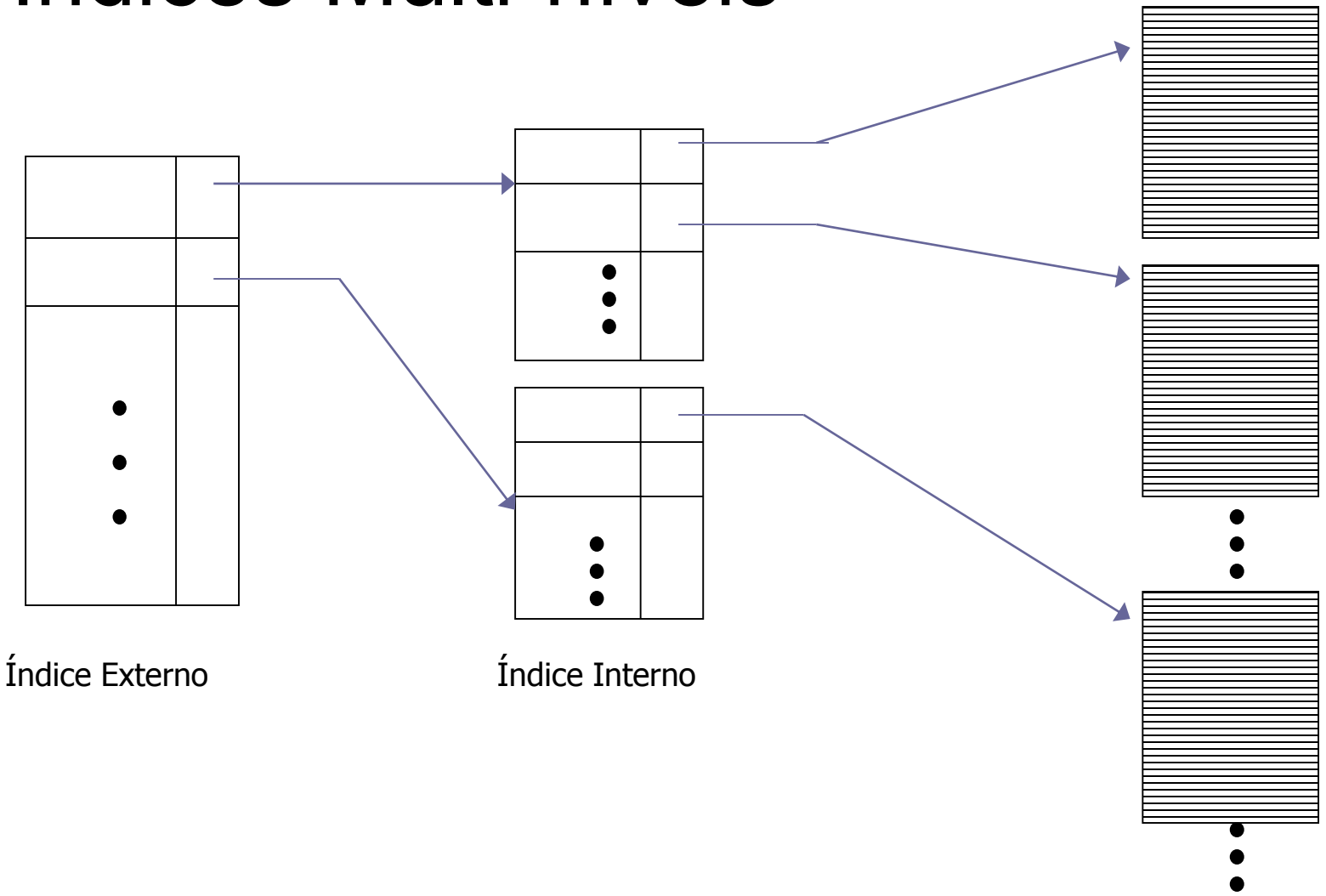
Para que usar índice primário?

- Características do arquivo de índice primário:
  - Número total de registros = 3000 (1 POR BLOCO)
  - Bloco de disco = 1024 bytes
  - Tamanho do registro do arquivo de índice = 15 bytes, por exemplo (campo chave de 9 bytes e ponteiro para bloco de disco de 6 bytes)
  - Fator de bloco do arquivo de índice é de 1024/15=68 registros/bloco.
  - Número de blocos necessários para armazenar o arquivo todo
  - Número total de registros / fator de bloco = 3000/68= 45 blocos
  - Busca binária = log 2 45 = 6 acessos a blocos

Vantagem de Índices Primários

- Redução expressiva do número de acessos a disco
- No exemplo anterior, sem a utilização do índice seriam necessários 12 acessos a bloco, com o índice é possível resolver com 7 acessos (6 acessos no índice e 1 no arquivo de dados).

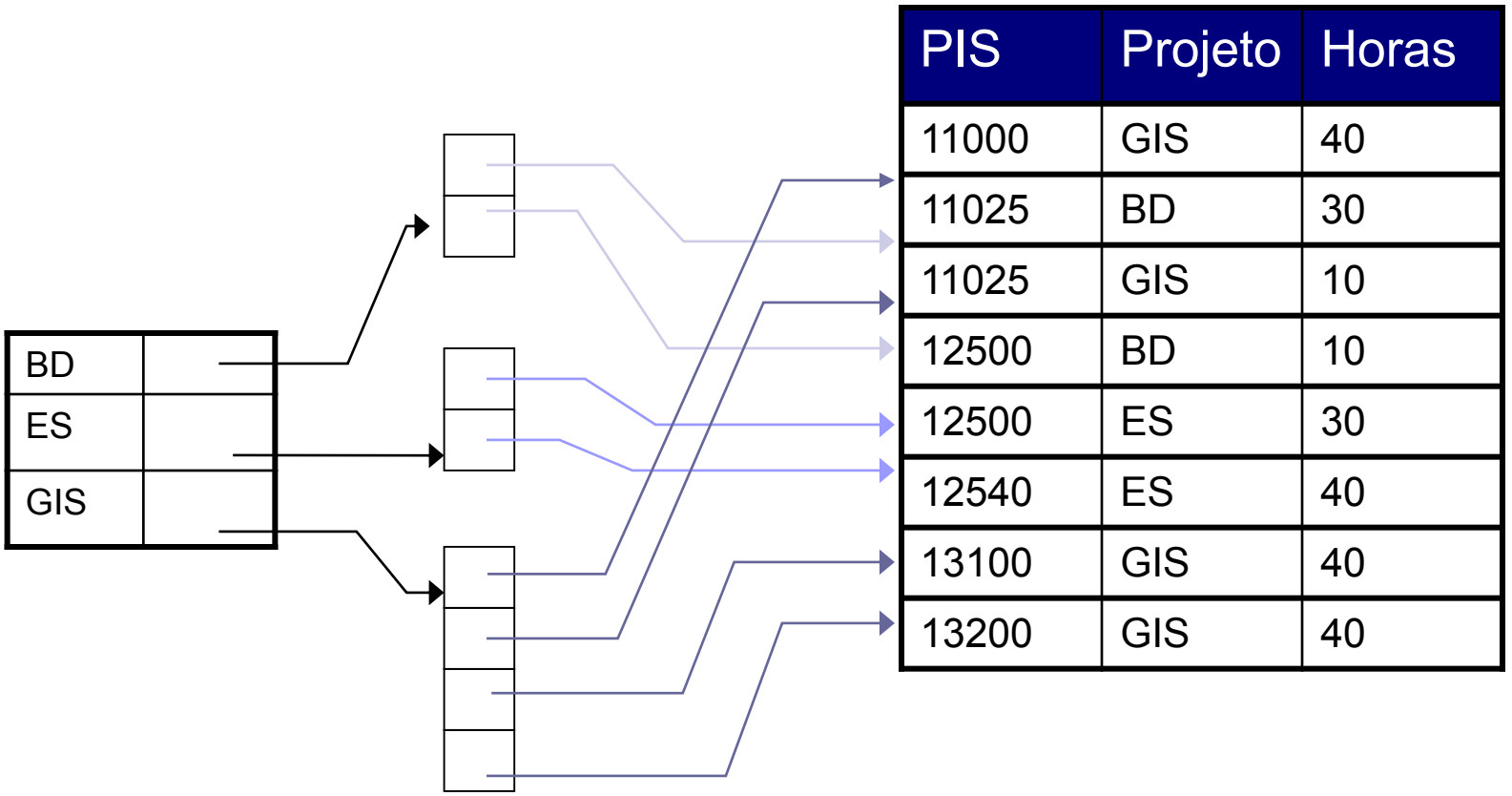
Índices Multi-níveis



Índices Secundários

- composto por 2 campos:
  - 1º consiste em um campo não ordenado do arquivo de dados, chamado **campo de indexação**, e o
  - 2º consiste em um ponteiro para um bloco de disco ou registro.
- Se o índice secundário é definido em um **campo chave - chave secundária -** haverá um registro de índice para cada registro do arquivo, o que o caracterizará como **índice denso**.

Índices Secundários



PIS	Projeto	Horas
11000	GIS	40
11025	BD	30
11025	GIS	10
12500	BD	10
12500	ES	30
12540	ES	40
13100	GIS	40
13200	GIS	40

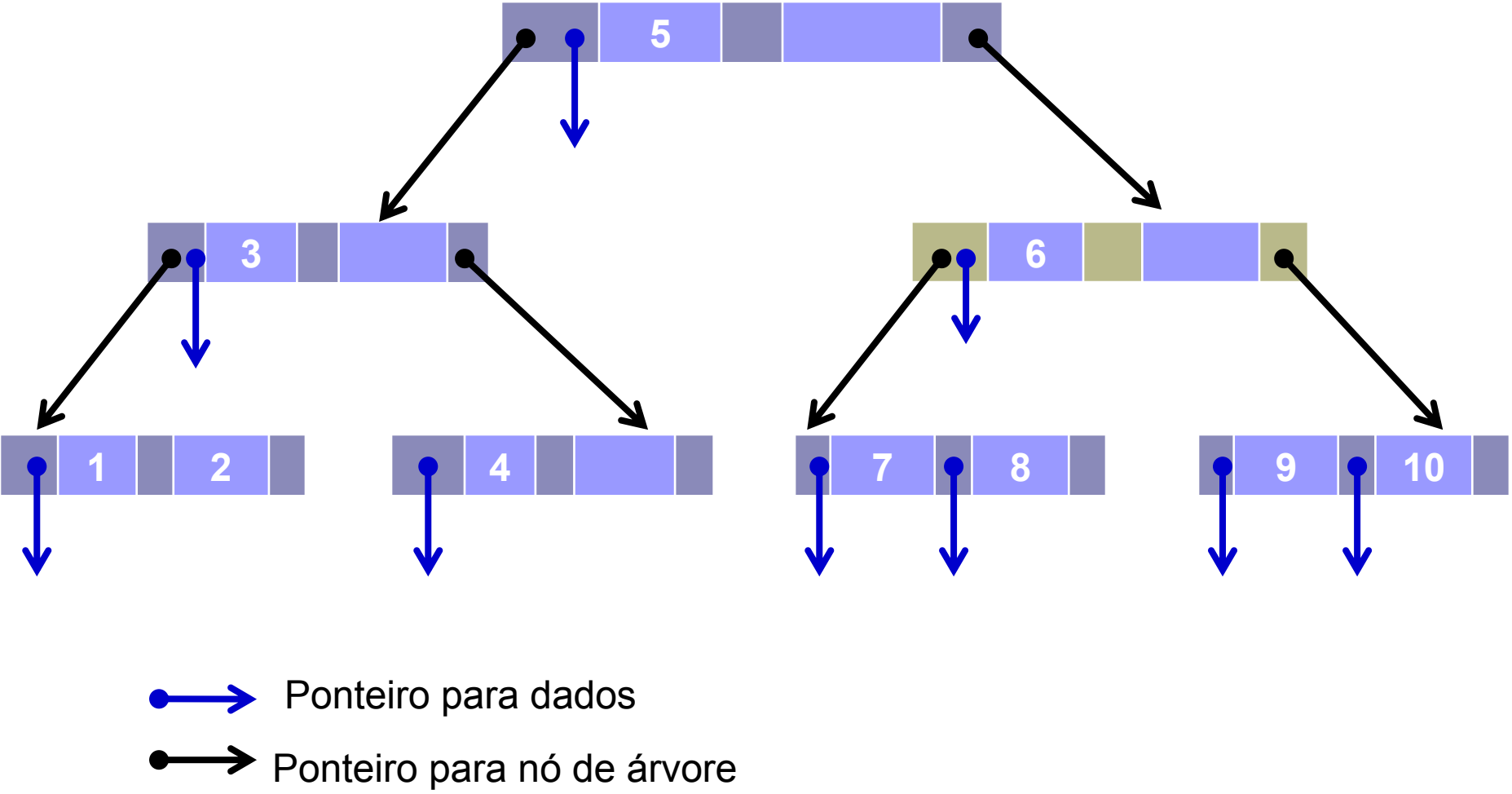
Índices Secundários

PIS	Projeto	H
11000	GIS	4
11025	BD	3
11025	GIS	1
12500	BD	1
12500	ES	3
12540	ES	4
13100	GIS	4
13200	GIS	4

Implementação de Índices Secundários

Árvores B

Árvore B

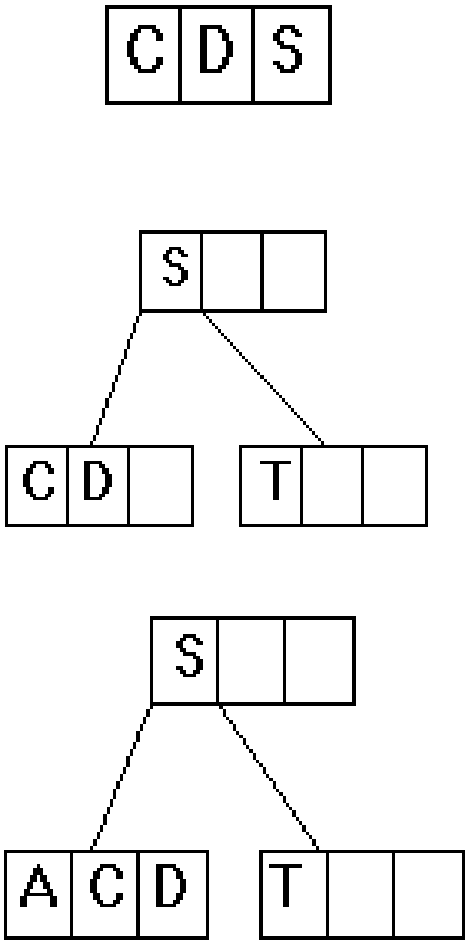

5
3
6
1
2
4
7
8
9
10
Ponteiro para dados
Ponteiro para nó de árvore

Árvore B

- É uma árvore homogênea (nós iguais)
- Todos os nós armazenam ponteiros para os dados
- Valores Chaves aparecem 1 única vez

B-tree - Inserção

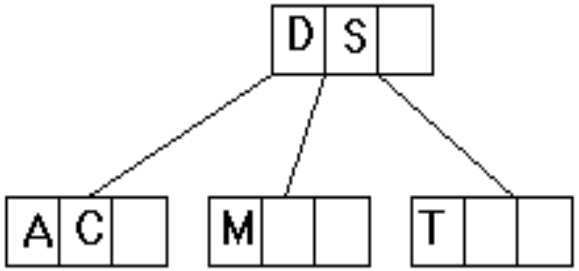
C S D T A M P I B W N G U R K E H O L J Y Q Z F X V



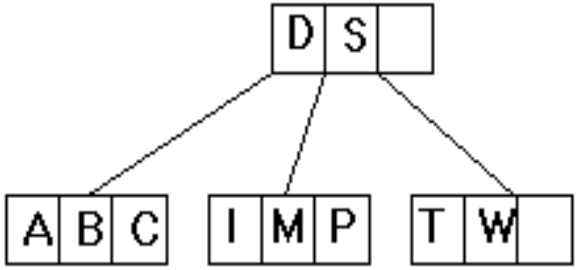
- Inserção de C,S e D dentro da página inicial.)
- Inserção de T força o *split* e o *promotion* de S.
- Adição de A.

B-tree - Inserção

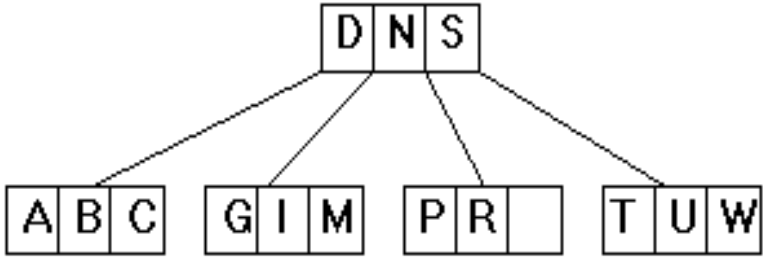
C S D T A M P I B W N G U R K E H O L J Y Q Z F X V



- Inserção de M força outro split e o promotion de D.



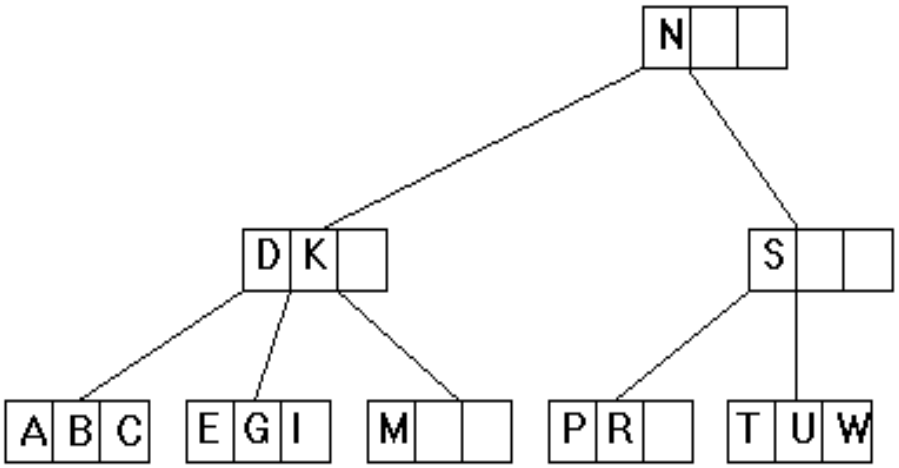
- Inserção de P, I, B,e W nas páginas existentes.



- Inserção de N precisa de outro split seguido por promotion de N (G, U e R).

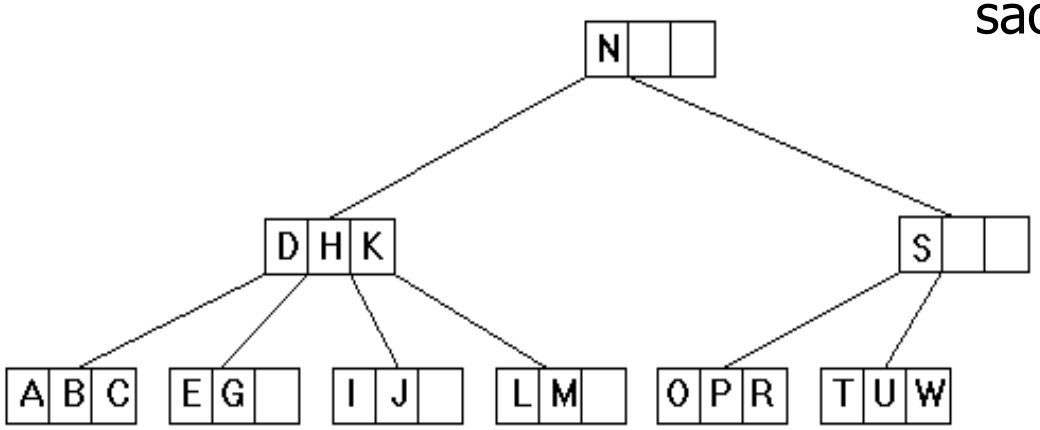
B-tree - Inserção

C S D T A M P I B W N G U R K E H O L J Y Q Z F X V



- Inserção de k resulta no spliting no nível folha, seguido pelo promotion de k. Isto resulta no split da raiz. N é promovido para ser a nova raiz. E é posto como nó folha

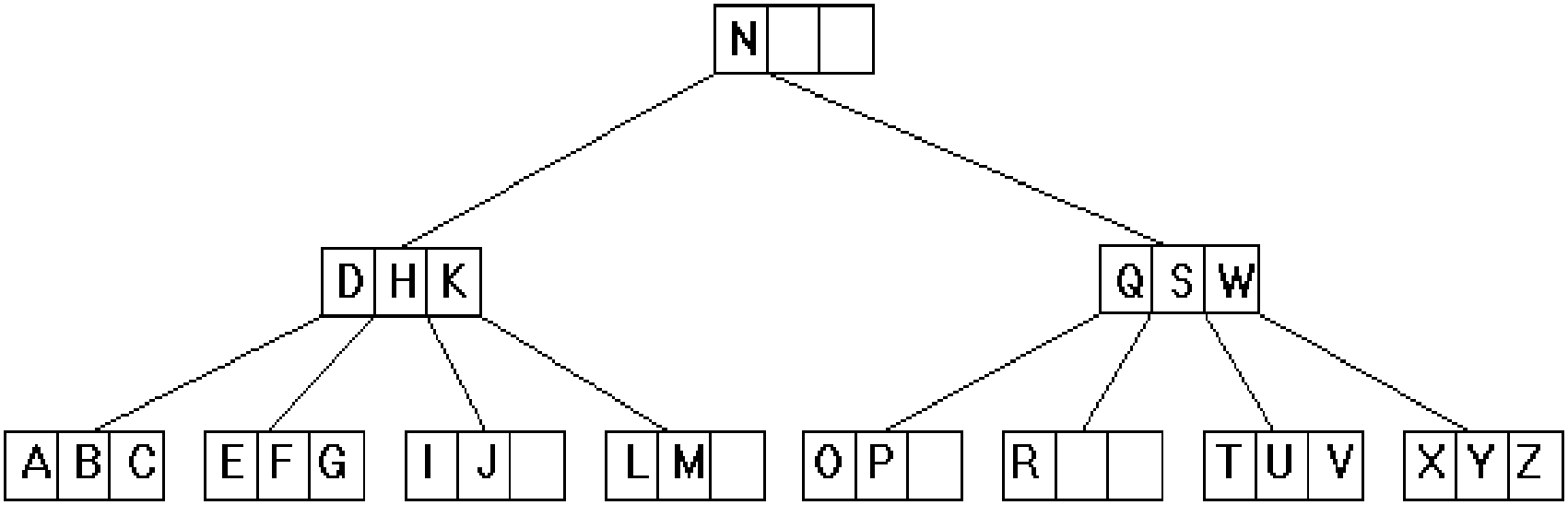
- Inserção de H resulta em split no nó folha. H é promovido. O, L e J são adicionados.



B-tree - Inserção

C S D T A M P I B W N G U R K E H O L J Y Q Z F X V

- Inserção de Y e Q força mais dois splits nos nós folhas. O restante das letras são adicionados.



Qual a vantagem índice secundário?

- Características do arquivo de dados:
  - Número total de registro = 30000
  - Tamanho do registro = 100 bytes
  - Bloco de disco = 1024 bytes
  - Fator de bloco = nro de registros existente em cada bloco de disco
  - FB= tam. do bloco/tam. do registro = 1024/100 =10 registros/bloco
  - Número de blocos necessários para armazenar o arquivo todo
  - Número total de registros / fator de bloco = 30000/10= 3000 blocos.
  - Busca seqüencial = 1500 acessos a blocos

Qual a vantagem do índice secundário?

- Características do arquivo índice secundário:
  - Número total de registros = 30000
  - Bloco de disco = 1024 bytes
  - Tamanho de uma chave de procura (valor+ponteiro) do nó da árvore b+ = 20 bytes
  - Nro de chaves no nó = 1024/20 = 51 chaves/no
  - Pesquisa árvore B^+ = $\log_{25}$ 300 = 3 acessos na árvore + 1 acesso no disco

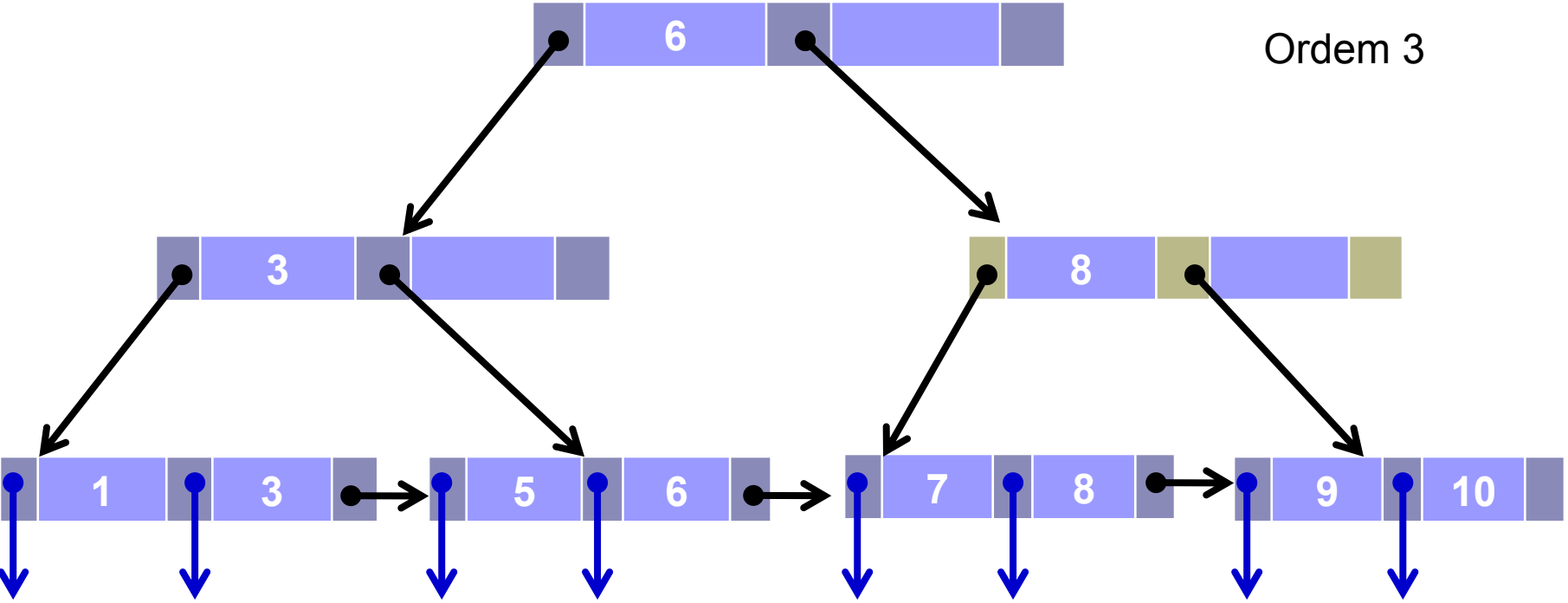
Implementação de Índices Secundários

Árvores B^+

Árvores B^+

- Difere da Btree nos seguintes aspectos:
  - É uma árvore heterogênea (nós diferentes)
  - Apenas os nós folha armazenam ponteiros para os dados
  - Chaves estão duplicadas na árvore.
  - Cada nó folha possui um ponteiro para o próximo nó folha na seqüência. Por isto, é possível a travessia pelo índice de folha p/ folha

Árvores B+

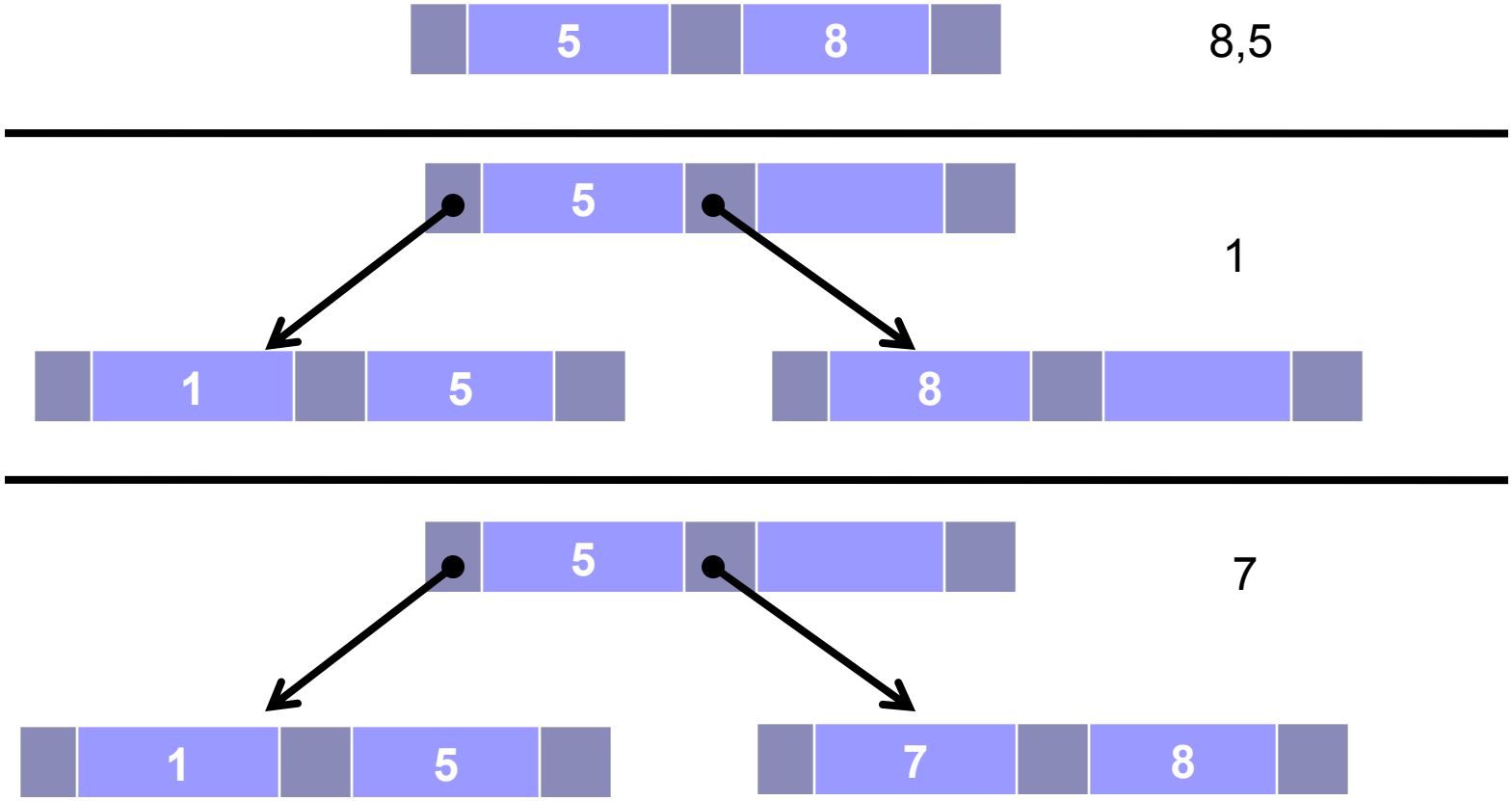


Ponteiro para dados

Ponteiro para nó de árvore

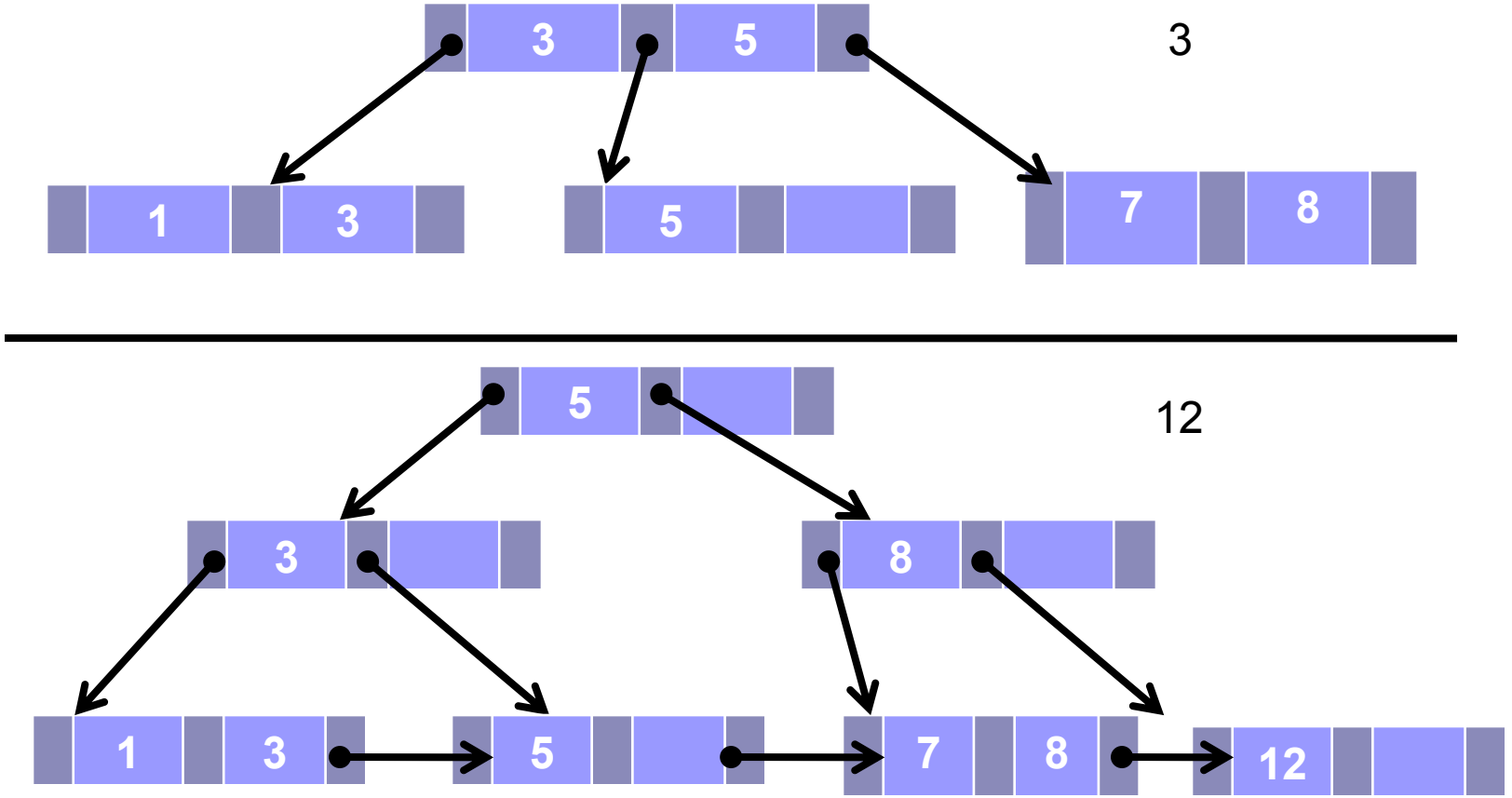
Árvores B^+

Seqüência de inclusão: 8,5,1,7,3,12,9,6

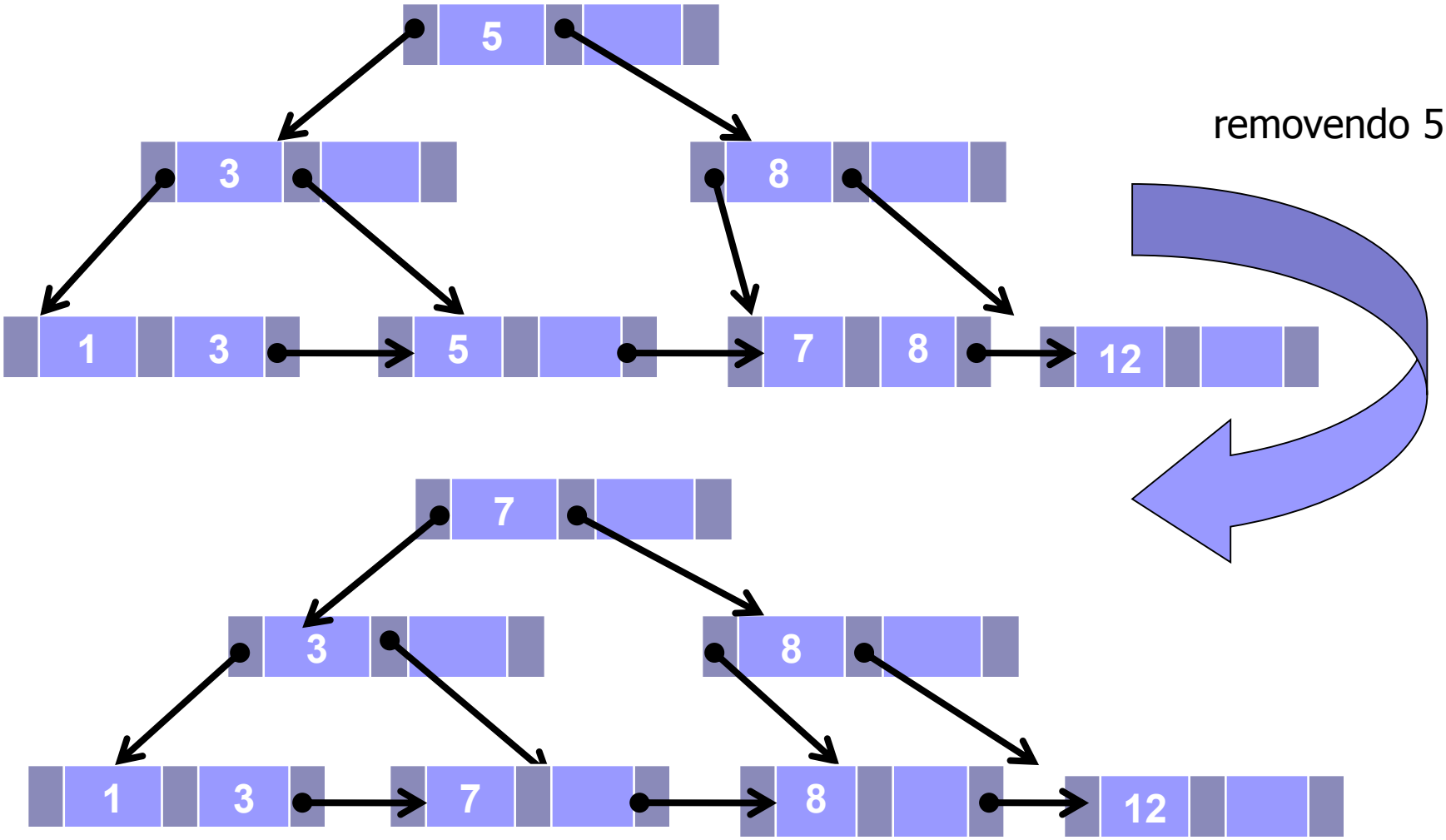


Árvores B+

Seqüência de inclusão: 8,5,1,7,3,12



Remoção em Árvore B+



Operações Básicas de Indexação

- Criação dos arquivos de índice e dados
- Trazer índice para a memória.
- Atualizar índice no disco.
  - Obs 1: Deve haver um mecanismo que permita saber se o índice está atualizado. Exemplo: uso de *status flag* no arquivo para indicar quando foi carregado e atualizado.
  - Obs2 : Todo programa, antes de usar o índice, verifica sua validade. Se inválido deve ser requisitado uma função que reconstrua o índice a partir do arquivo.

Operações Básicas de Indexação

- Adição de Registro.
  - Aumenta o *indexfile*.
  - Atualização somente na memória.
  - Não envolve acesso a disco.

- Eliminação de Registro.
  - No *indexfile*, é necessário rearranjo para que a ordem seja mantida

HASHING

HASH

- O que é *Hashing* ?
  - A função *hash*, h(k), transforma uma chave k num endereço.
  - É similar a Indexação pois associa a chave ao endereço relativo do registro.
- Diferenças:
  - com *hashing* os endereços parecem ser aleatórios;
  - com *hashing* duas chaves podem levar ao mesmo endereço (a colisão deve ser tratada)

Tratamento de Colisões

- Solução Ideal: algoritmo que não produz colisão. Muito difícil para mais de 500 chaves.
- Espalhamento dos registros: Busca de um algoritmo que distribui os registros relativamente por igual.
- Utilização de mais memória: é relativamente fácil achar um bom algoritmo quando se pode desperdiçar muito espaço.
- Utilização de mais de um registro por endereço: técnica chamada de *bucket*.

Algoritmo Simples de *Hashing*

1. Represente a chave numericamente.
2. Pique o número em pedaços e some;
3. Ajuste a soma dividindo por um número primo;
3. Divida pelo espaço de endereçamento para obter endereço final.

Algoritmo Simples de *Hashing*

- Passo 1. Tome a chave LOWELL, forme a string de caracteres, com o código ASCII
  - LOWELL = 76 79 87 69 76 76 32 32 32 32 32 32
  - L O W E L L <-------branco------->
- Passo 2. Pique e some.
  - Pique ---> 7679!8769!7676!3232!3232!3232!
  - Adição--> 7679+8769+7676+3232+3232=30588
  - Nro de “endereços” do índice = 19937 (um valor definido)
- Passo 3. Para não ultrapassar o valor 19937 utilizamos o operador *mod* (*n).*
  - a = s mod n
    - s = representa a soma do passo 2.
    - n = representa o divisor (# endereços no arq.).
    - a = o endereço a produzir.
  - Ex : 100 endereços 0-99 para o nosso arq. Pela fórmula a = 13883 mod 100 = 20

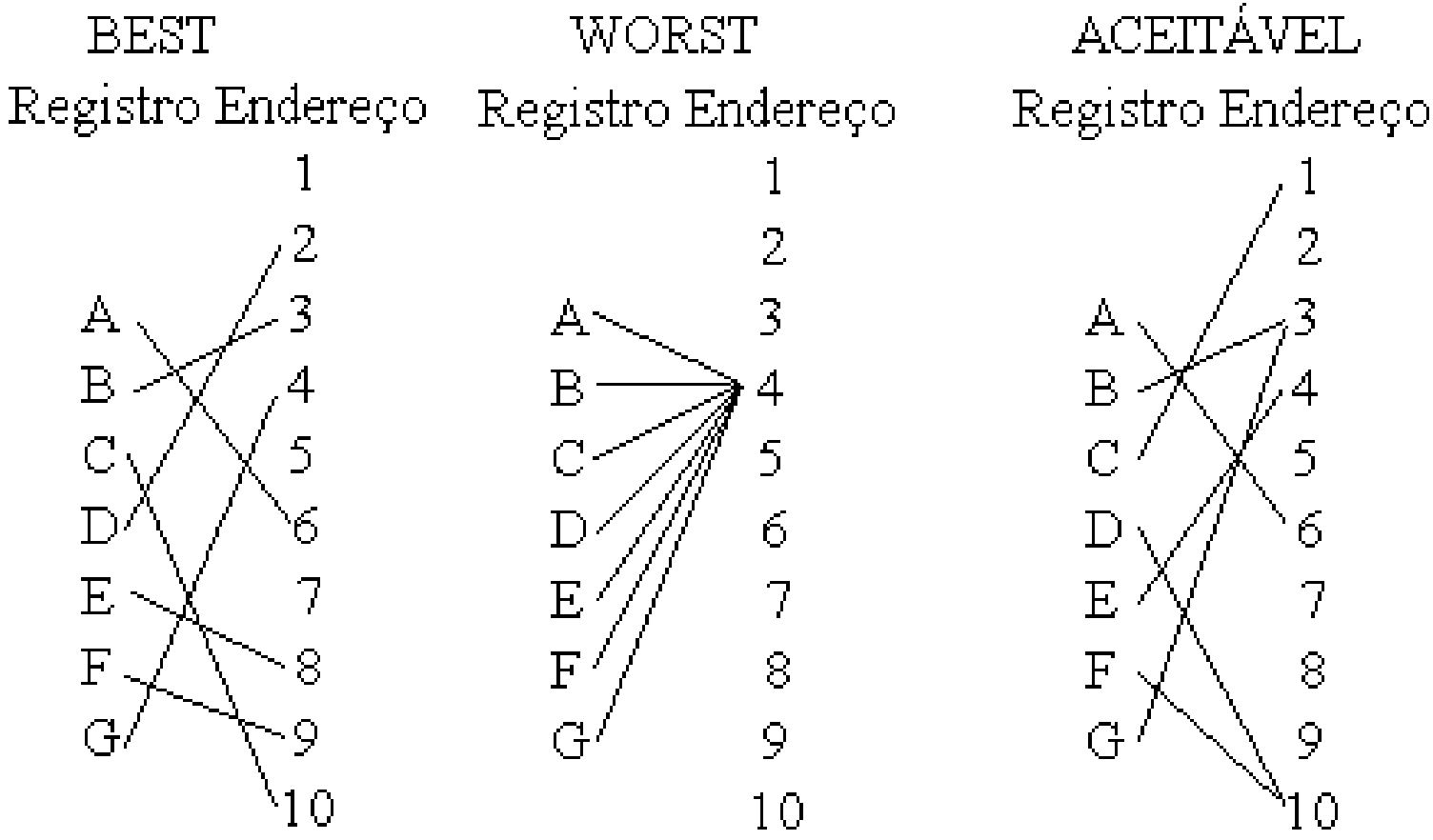
Algoritmo Simples de *Hashing*

```
FUNCTION hash (KEY,MAXAD)
  set SUM to 0
  set J to 0
  while (J < 12)
    set SUM to (SUM+100*KEY[J]+KEY[J+1]) mod 19937
    increment J by 2
  endwhile
  return (SUM mod MAYAD)
end FUNCTION
```

Distribuição dos registros entre endereços

- A função hash perfeita seria aquela que não produzisse nenhum sinônimo para um dado conjunto de chaves, em um dado endereçamento.
- A pior função seria aquela que , para qualquer chave, geraria sempre o mesmo endereço (todas as chaves seriam sinônimas).
- Uma função aceitável gera poucos sinônimos.

Distribuição de registros nos endereços



Alguns outros métodos *hashing*

- examinar as chaves ?
- separar e somar partes da chave apenas
- dividir a chave por primos
- elevar a chave ao quadrado e considerar os dígitos do meio
- mudar de base e dividir pelo espaço de endereçamento (considerando o módulo)

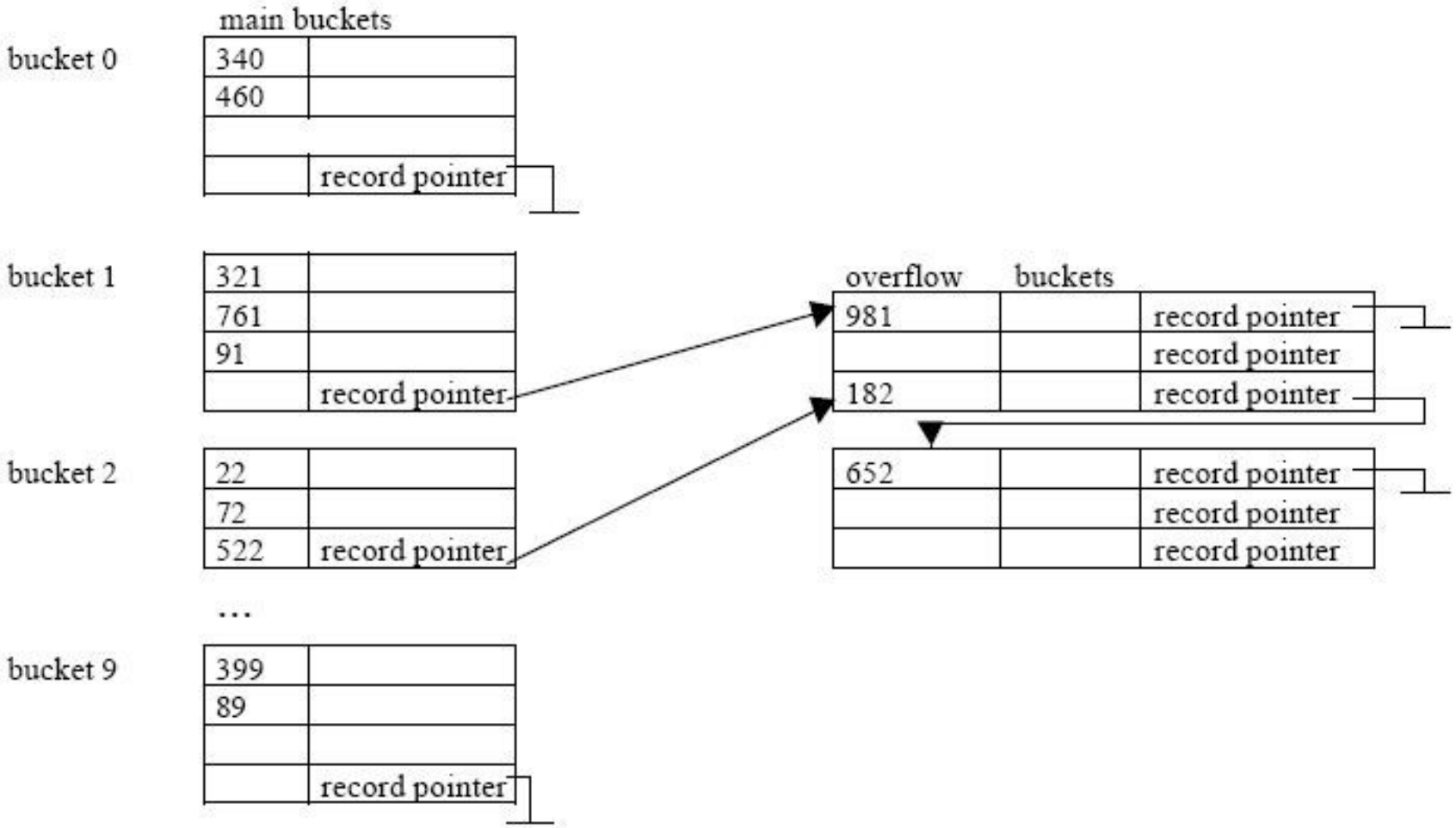
Tipos de *Hashing*

- Estático
- Dinâmico

Terminologia

- *Buckets*: Lista de tamanho fixo de chaves com a mesma codificação *hashing*
- *Overflow*: é necessário inserir mais chaves do que um *bucket* suporta

Tratamento de *overflow* de *buckets*



Manipulando overflow de *buckets* através de *chaining*

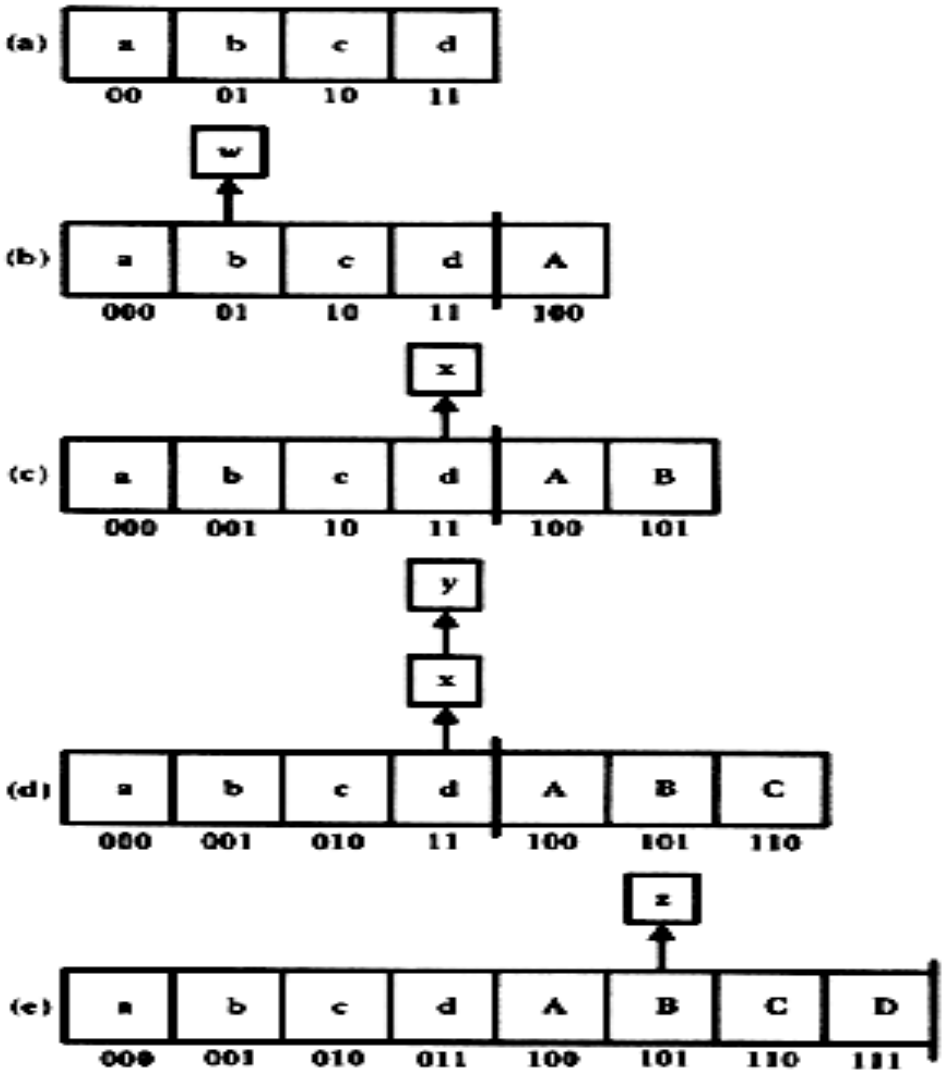
Hashing Linear

- *Hashing* linear é um tipo de *hashing* que se difere dos outros tipos dinâmicos pela não-utilização de diretórios para acesso aos *buckets*.
- Utiliza uma tabela principal e um tabela de *overflow*. Quando a tabela principal está cheia, a tabela de *overflow* recebe o *bucket* com a chave e um número relacionado com a chave da qual ela é sinônima. Quando a tabela de overflow fica cheia, a tabela principal aumenta de tamanho, recebe os *buckets* da tabela de overflow e esta é esvaziada, aumentando de tamanho também.

Hashing linear

- Vantagens
  - O número de acessos a disco é menor que em outros tipos de hashing. No hashing estendido e no hashing dinâmico, dois ou mais ponteiros podem apontar para o mesmo *bucket*, causando mais de uma busca no disco. O hashing linear diminui o nº de leituras do disco, fazendo o acesso mais rápido
- Desvantagens
  - Ele utiliza menos espaço, em média, que os outros tipos de hashing. No hashing linear, a média é de ±60%. Nos outros dois tipos de hashing, a média gira em torno de 69%.

Hashing Linear



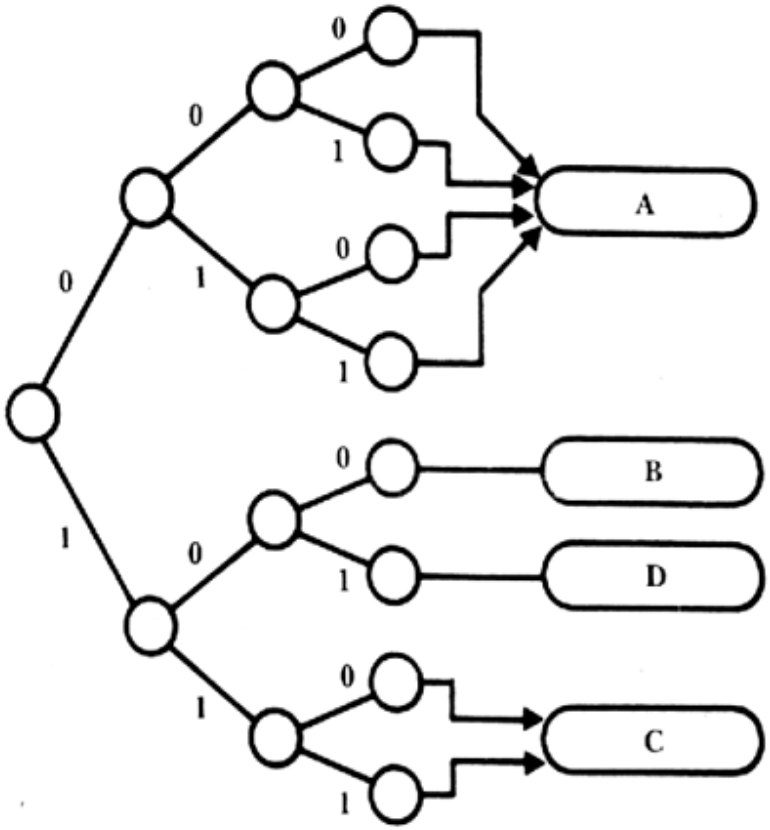
Hashing Extensível

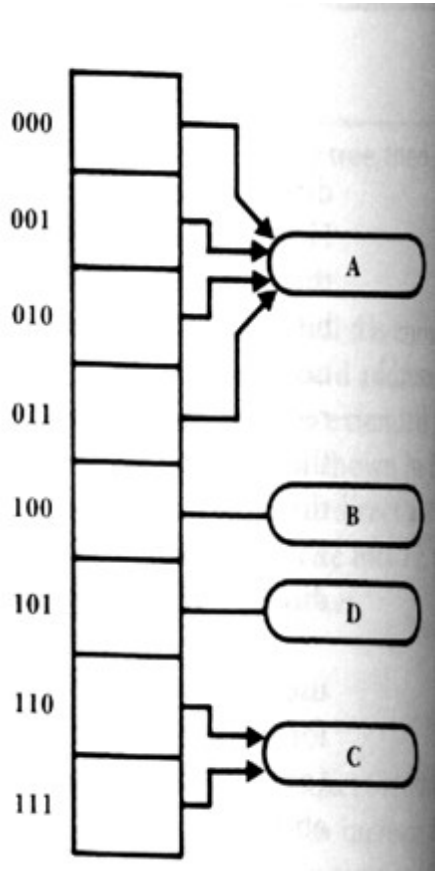
- Neste tipo de *hashing, o número de buckets não é fixo, crescendo e diminuindo quando necessário.*
- *O* arquivo pode começar com um único *bucket, quando este bucket encher, um novo bucket deve ser criado.*
- Os registros são divididos entre os *2 buckets baseados no valor do 1o bit de seus valores hash.*
- *Registros* cujos valores *hash começam com um bit 0 são armazenados em um bucket, aqueles cujos valores hash* começam com bit 1 são armazenados em outro *bucket.*
- *Neste ponto, uma árvore binária chamada* diretório é construída.

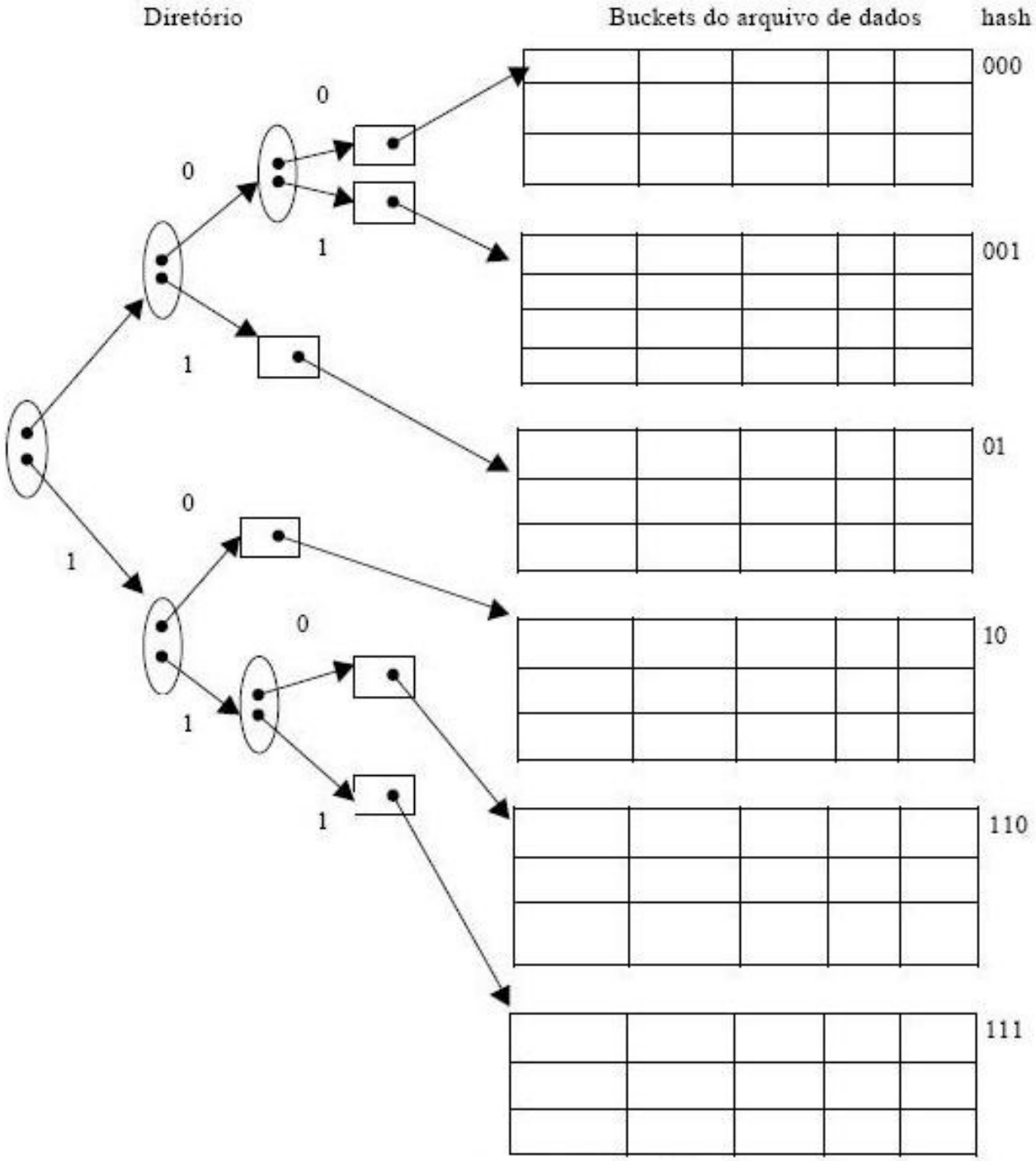
Diretórios

- Utiliza diretório, um vetor $2^{\text{profundidade}}$ de endereços de *buckets*
- A profundidade do diretório indica o número de bits no qual o conteúdo dos *buckets* são baseados

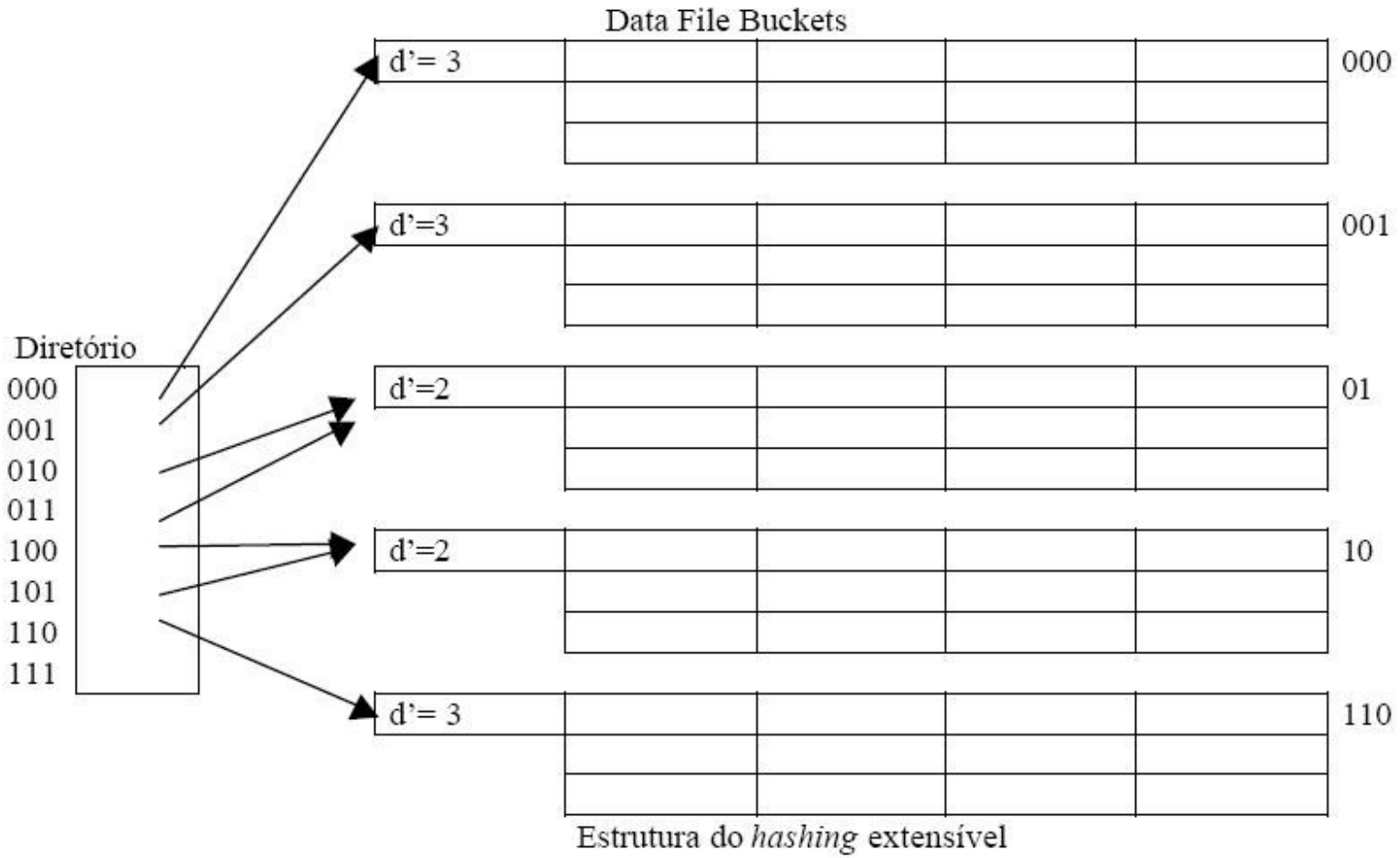
Diretório de profundidade 3






Diretório
Buckets do arquivo de dados
hash value
000
001
01
10
110
111
0
1
0
1
0
1
0
1

Hashing Extensível



Performance

- **Quanto ao tempo:**
  - A maioria das recuperações de registro exige dois acessos a blocos: um para o diretório e outro para o *bucket*.