AULA 04

Árvore-B: Introdução (Parte 1)

Prof. Tiago A. Almeida
talmeida@ufscar.br

Cenário

✓ Acesso a disco é caro (lento)

✓ Pesquisa binária é útil em índices ordenados...

✓ mas com índice grande que não cabe em memória principal, pesquisa binária exige muitos acessos a disco

- **Exemplo:** uma busca em um índice com 100.000 chaves podem requerer até 17 acessos ao disco!!!

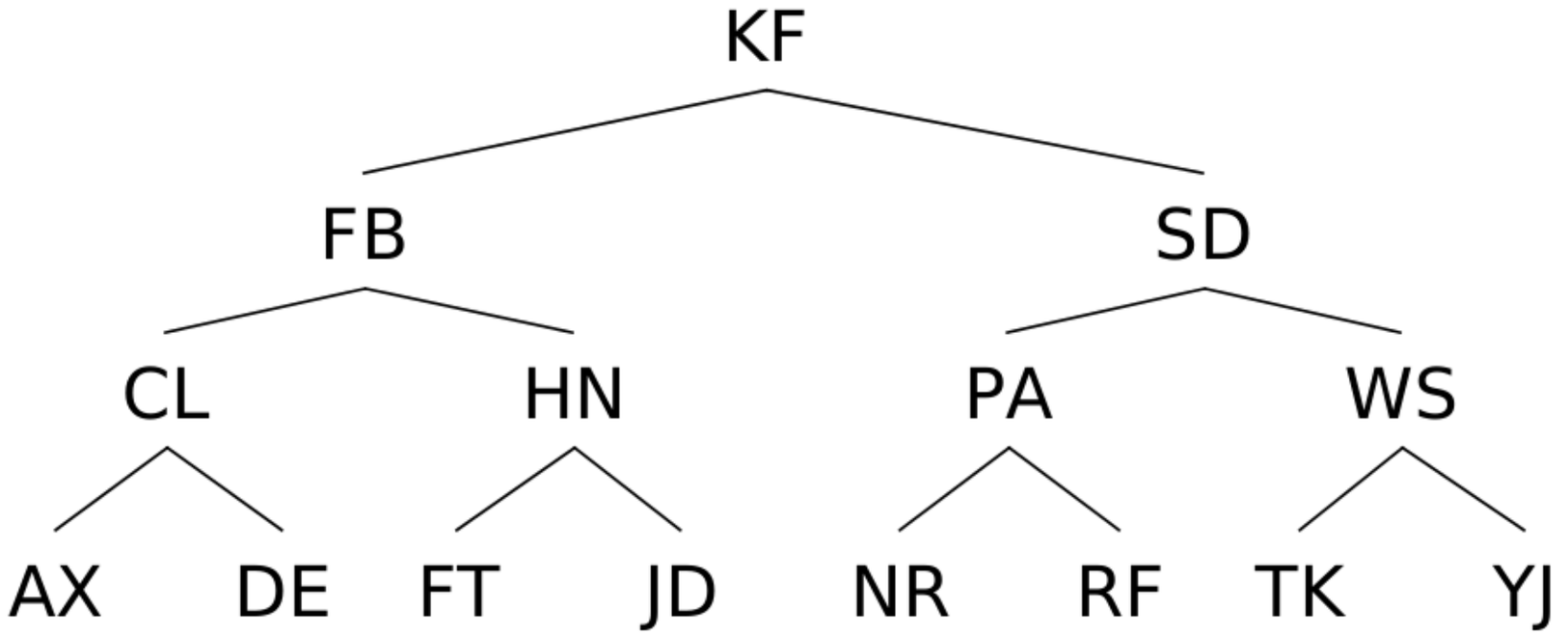
Cenário

- ✓ Manter em disco um índice ordenado para busca binária tem custo proibitivo

- ✓ Necessidade de método com inserção e eliminação com apenas efeitos locais, isto é, que não exija a reorganização total do índice

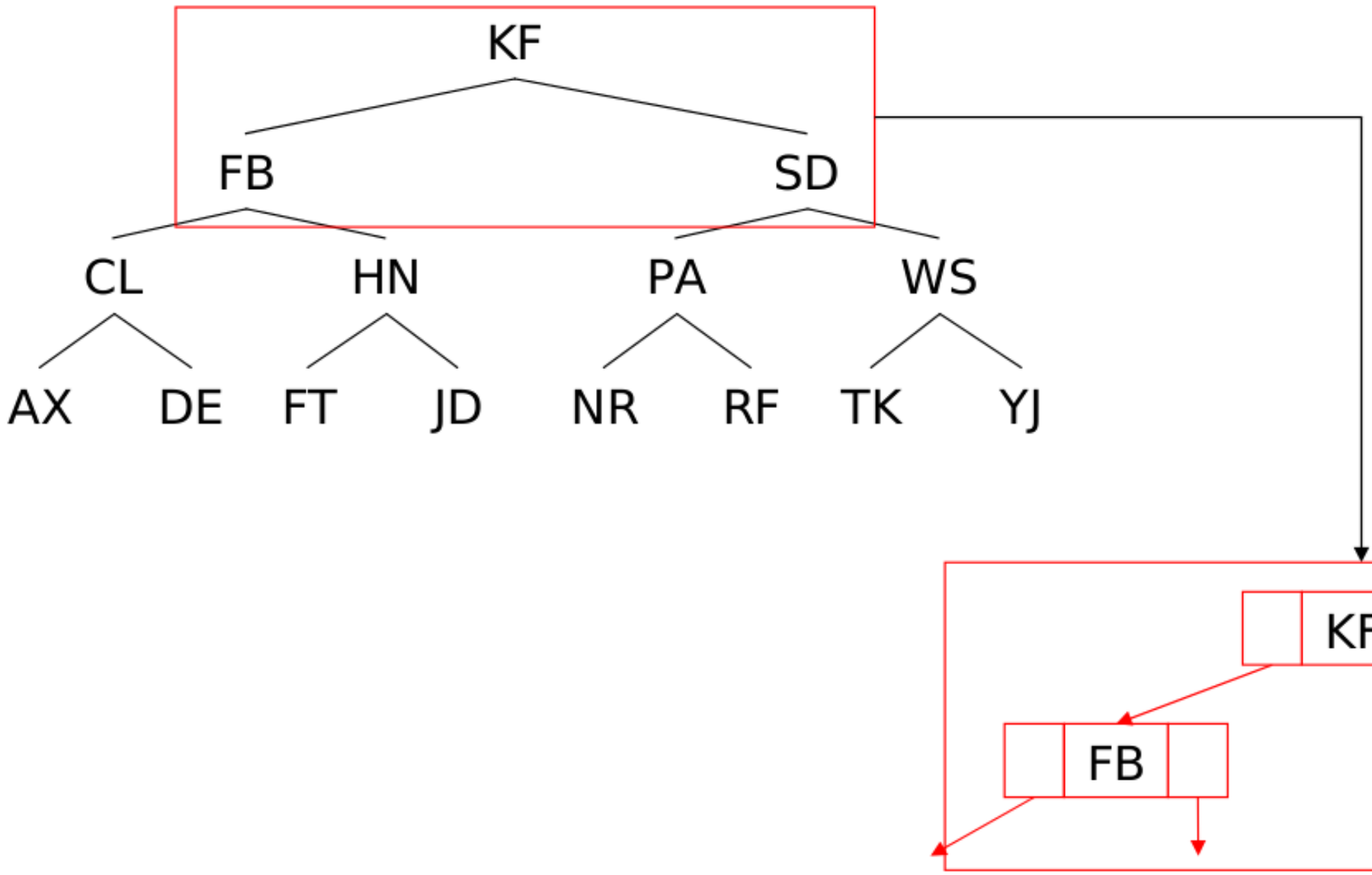
Solução: Árvores Binárias de Busca?

AX, CL, DE, FB, FT, HN, JD, KF, NR, PA, RF, SD, TK, WS, YJ

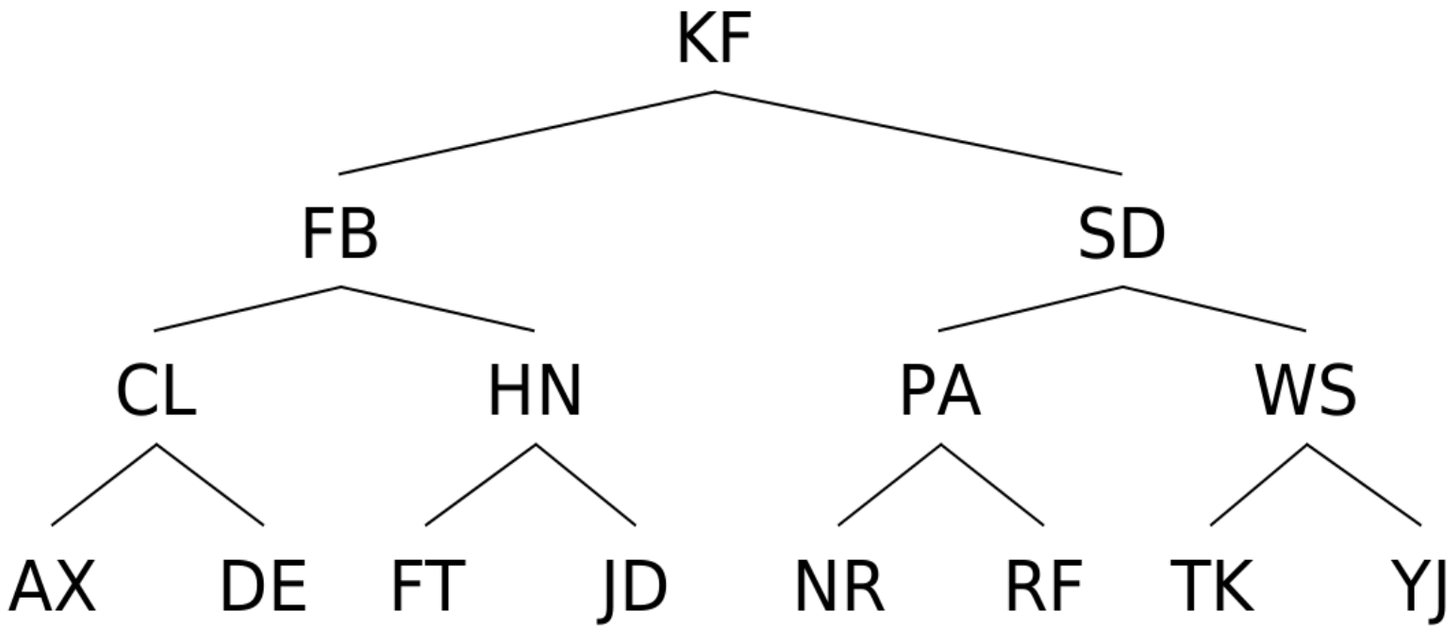


Vetor ordenado e representação por árvore binária

Solução: Árvores Binárias de Busca?



Representação da árvore no arquivo



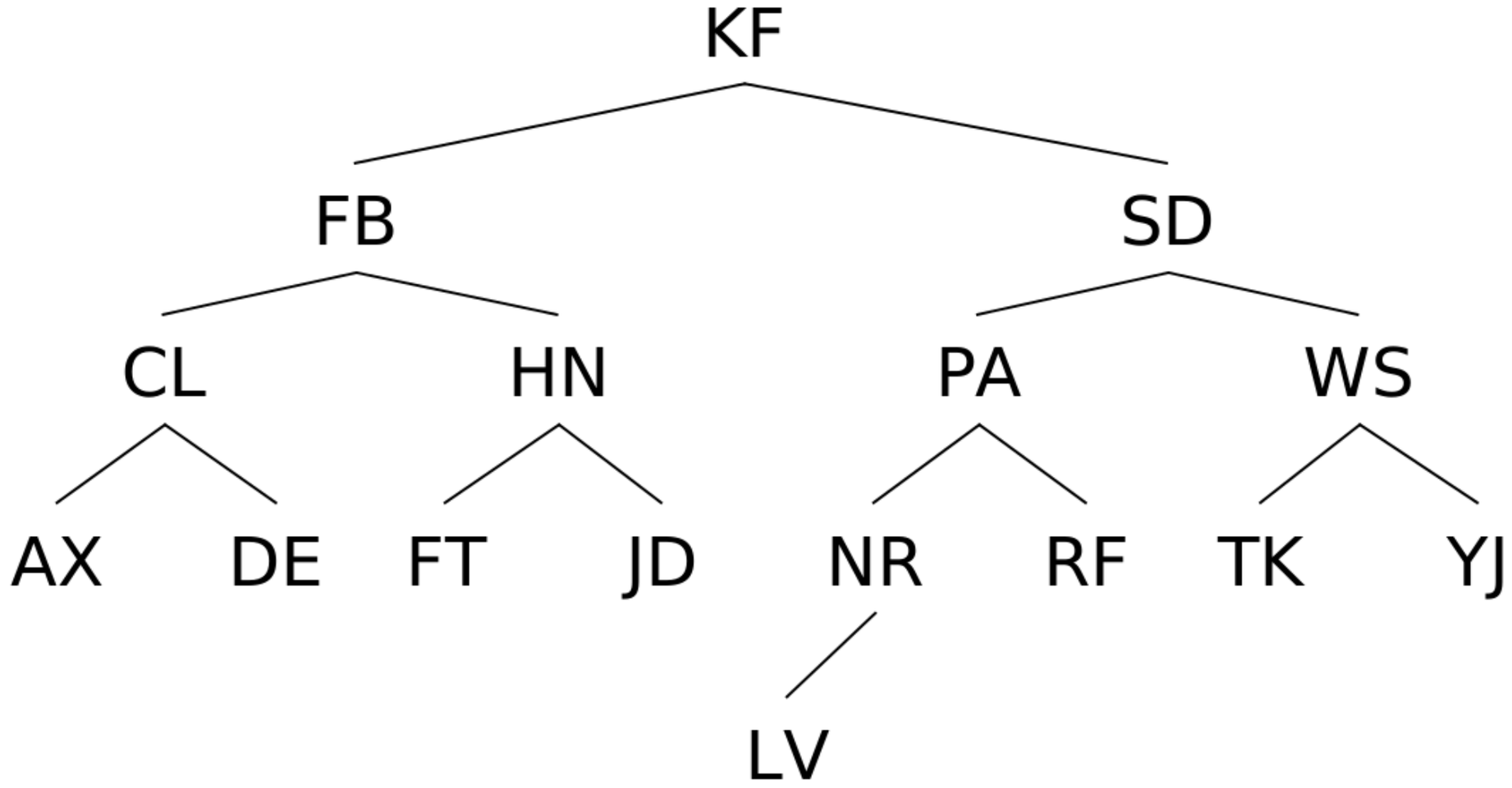
Registros são mantidos em arquivo e ponteiros (`esq` e `dir`) indicam onde estão os registros filhos

		Chave	Filho esq.	Filho dir.
	0	FB	10	8
	1	JD		
	2	RF		
	3	SD	6	13
	4	AX		
	5	YJ		
	6	PA	11	2
	7	FT		
	8	HN	7	1
raiz →	9	KF	0	3
	10	CL	4	12
	11	NR		
	12	DE		
	13	WS	14	5
	14	TK		

Vantagens das ABBs

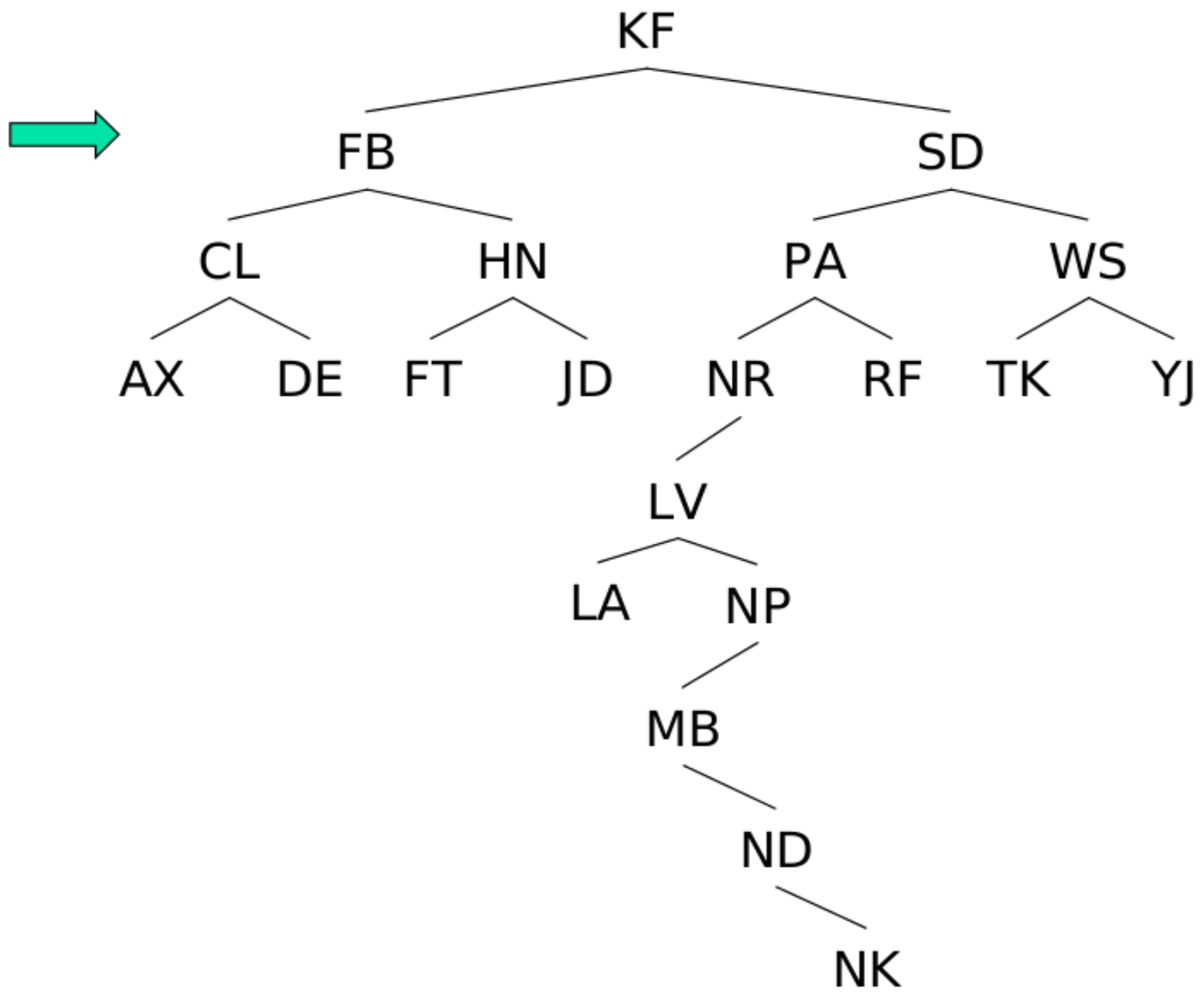
- ✓ Registros não precisam estar fisicamente ordenados
  - Ordem lógica: dada por ponteiros `esq` e `dir`
- ✓ Inserção de uma nova chave no arquivo
  - É necessário saber onde inserir
  - Busca pelo registro é necessária, mas reorganização do arquivo não

Inserção da chave LV



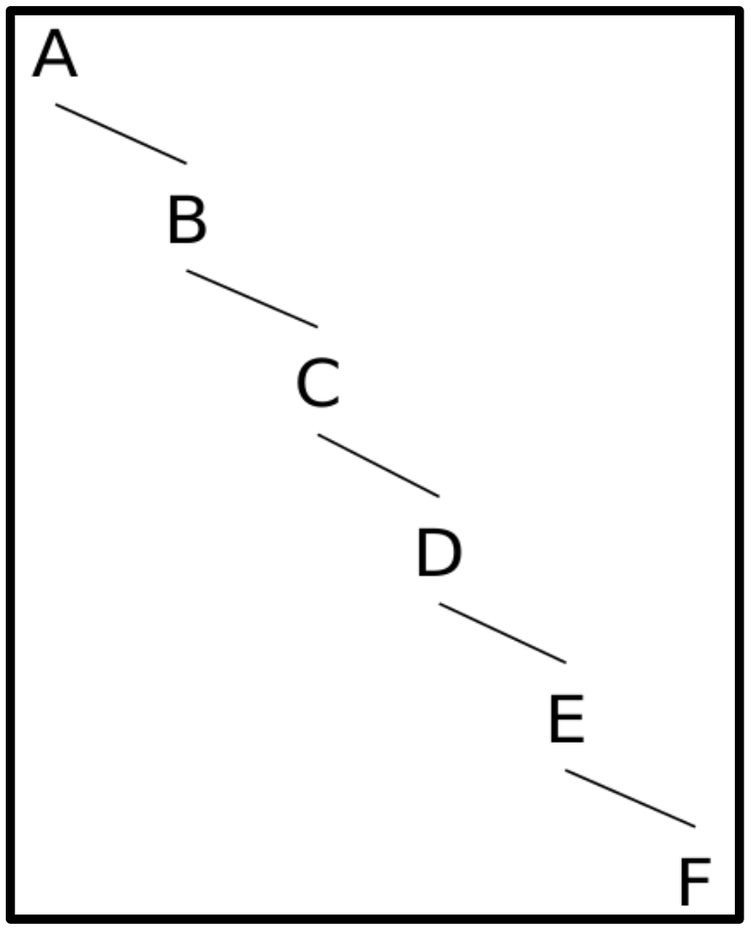
Problema: desbalanceamento

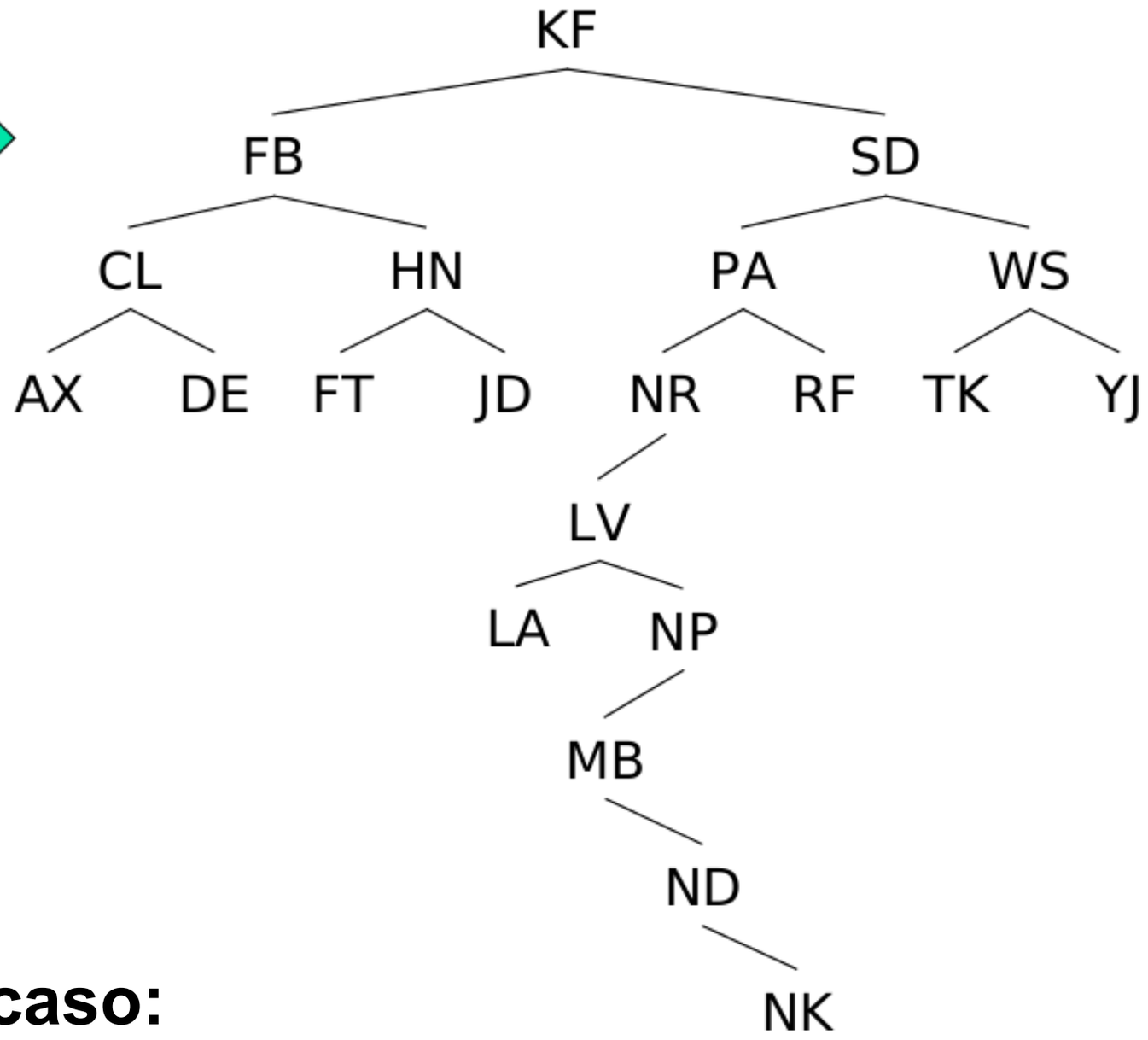
Inserção das chaves NP, MB, TM, LA, UF, ND, TS e NK



Problema: desbalanceamento

Inserção das chaves NP, MB, TM, LA, UF, ND, TS e NK

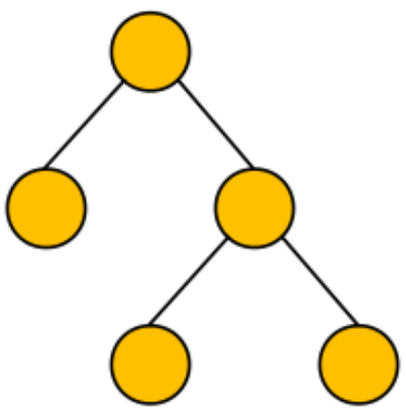


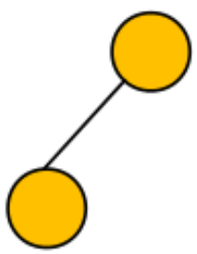


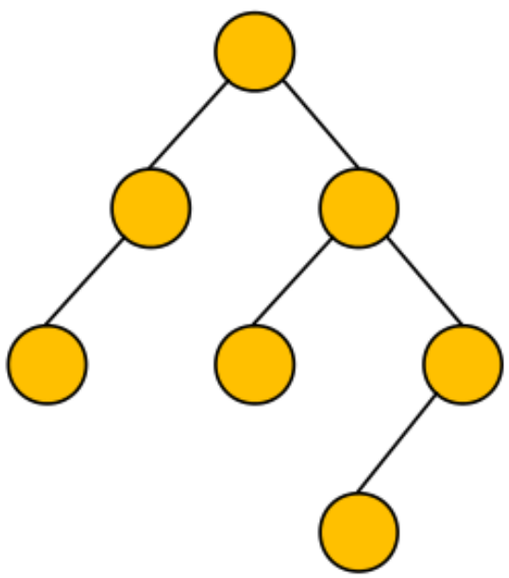
Pior caso:
inserção de chaves em ordem alfabética

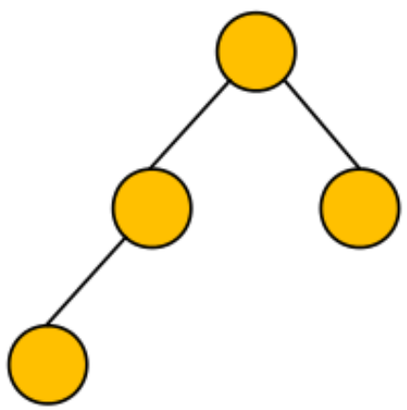
Solução por árvores-AVL

- ✓ Diferença limitada entre níveis
  - Garante performance aproximada de uma árvore completamente balanceada
  - Tradicionalmente, 1 nível de diferença
    - ★ Procedimentos específicos de inserção e remoção
    - ★ Manutenção (complexa) feita por rotações



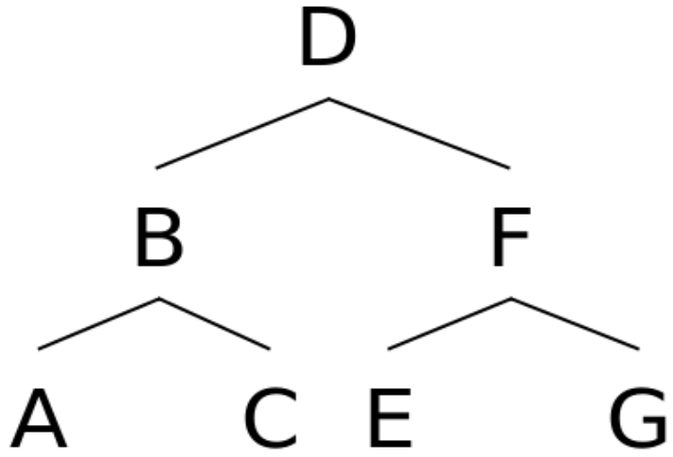




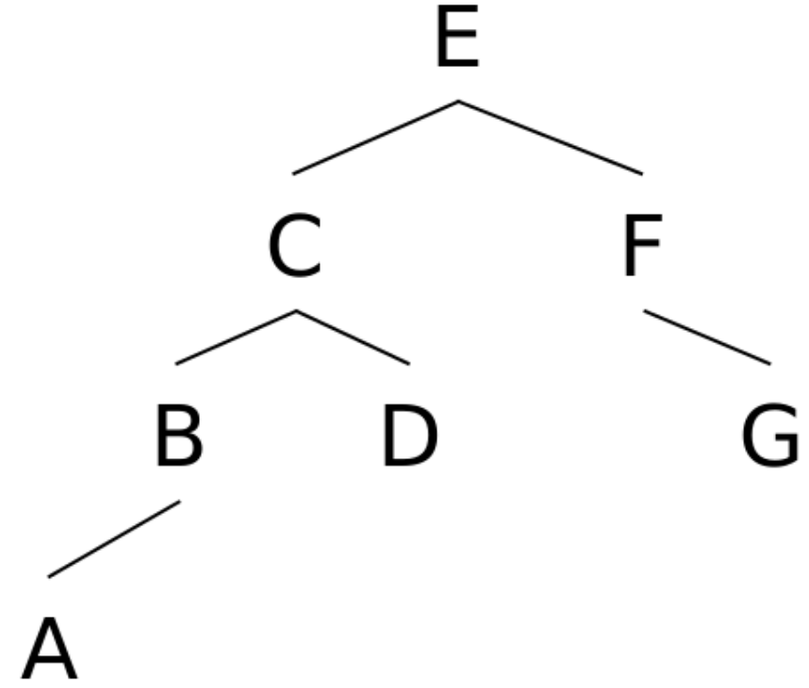


ABBs balanceadas e AVL

Chaves de entrada: B C G E F D A



Árvore perfeitamente balanceada



AVL (aceitável)

Solução por árvores-AVL

✓ Árvores binárias de busca balanceadas garantem eficiência

✓ Busca no pior caso

- Árvore binária balanceada: altura da árvore, ou seja, $\log_2 (n + 1)$
- AVL: $1{,}44 * \log_2 (n + 2)$
- Exemplo: com 1.000.000 chaves

★ Árvore binária perfeitamente balanceada: busca em até 20 níveis

★ AVL: busca em até 28 níveis

Solução por árvores-AVL

Problema

- ✓ Se as chaves estiverem em memória secundária, ainda é necessário muitos acessos!
  - 20 ou 28 `SEEKS` ainda é muito custoso!

Cenário atual

- ✓ Árvores binárias de busca dispensam ordenação dos registros
- ✓ Necessitam de número alto de acessos

Paginação

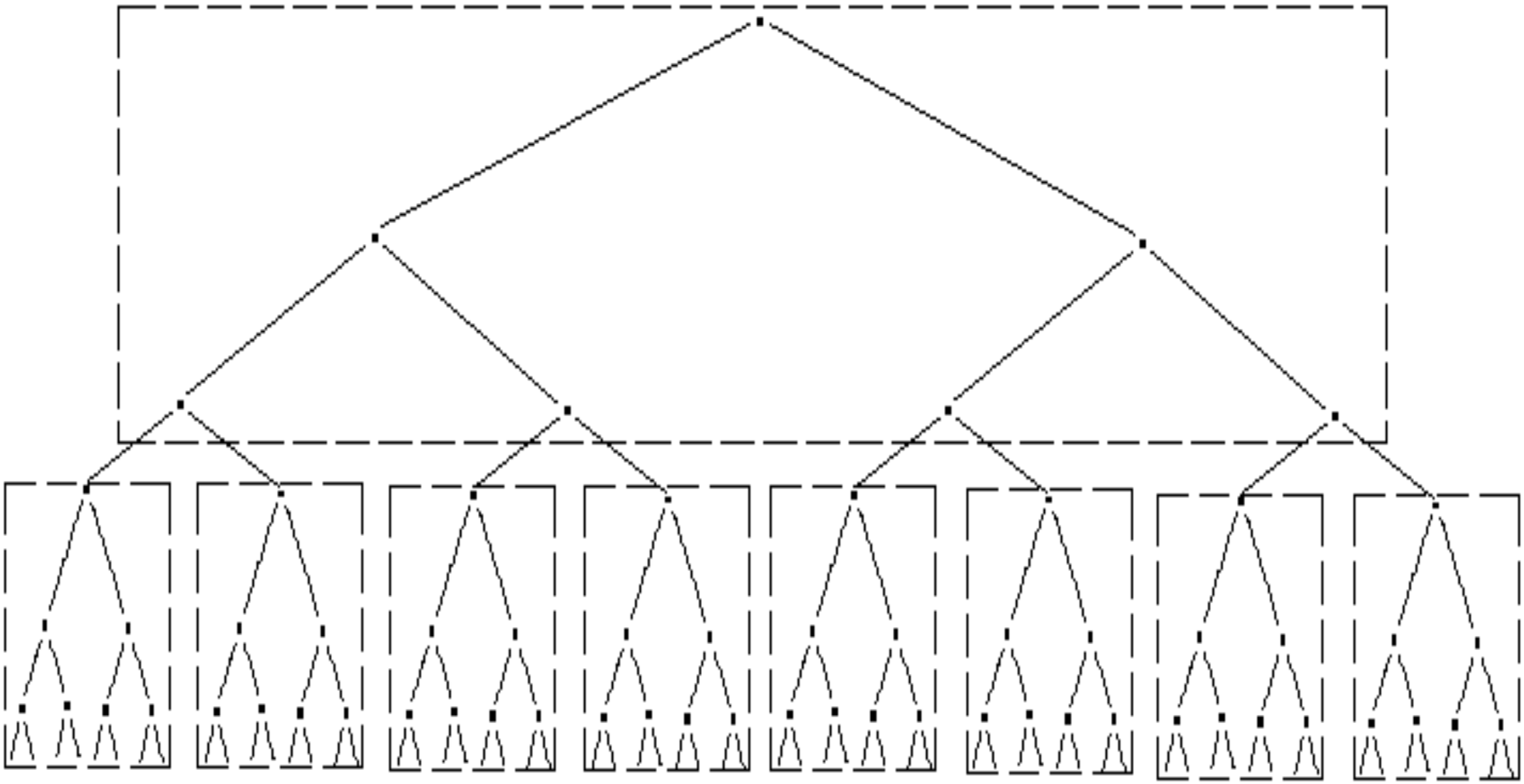
- ✓ A busca (`SEEK`) por uma posição específica do disco é muito lenta
- ✓ Porém, uma vez na posição, pode se ler uma grande quantidade de registros sequencialmente a um custo relativamente baixo

Árvores Binárias Paginadas

Noção de **página** em sistemas paginados

- ✓ Feito um `SEEK`, todos os registros de uma mesma "página" do arquivo (ex: 2 KB de um setor) são lidos
- ✓ Esta página pode conter um número grande de registros
  - Se o próximo registro a ser recuperado estiver na mesma página já lida, evita-se novo acesso

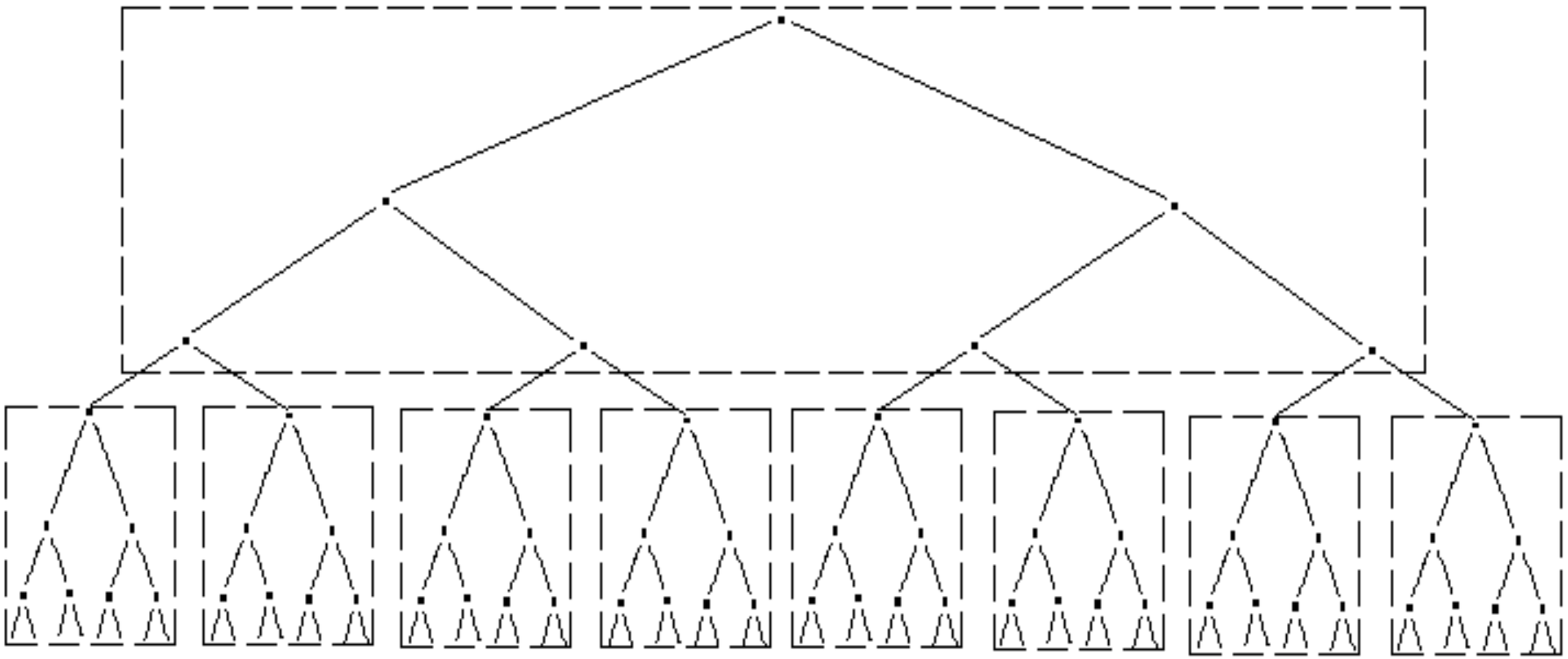
Árvores Binárias Paginadas



Alocar múltiplos nós nas mesmas páginas

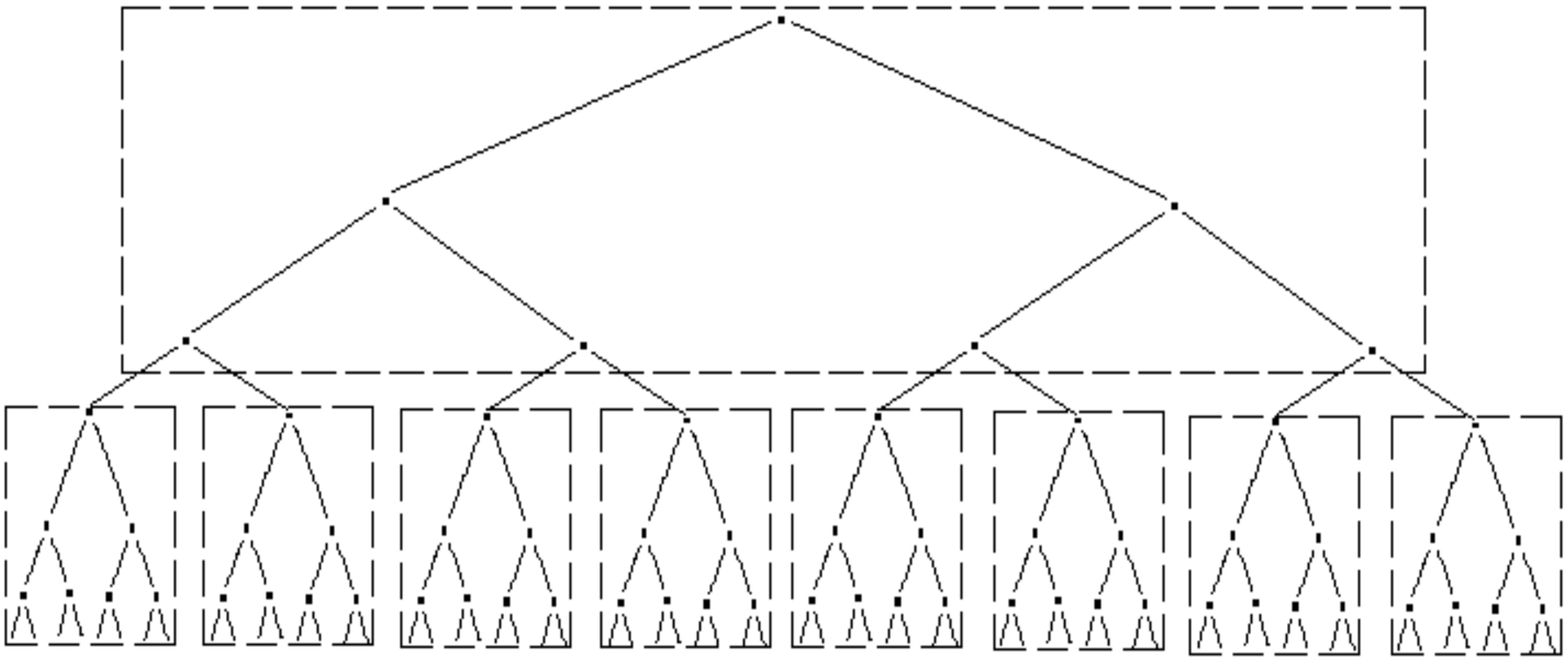
Árvores Binárias Paginadas

- ✓ No exemplo, cada página aloca 7 nós e permite acesso a 8 páginas
- ✓ Assim, qualquer um dos 63 registros (9x7 nós) pode ser acessado com, no máximo, 2 acessos



Árvores Binárias Paginadas

- ✓ Se a árvore for estendida com um nível de paginação adicional, criam-se 64 novas páginas
- ✓ Podemos encontrar qualquer uma das 511 (64 x 7 + 63) chaves com no máximo 3 `SEEKS` (contra 9 de uma AVL)



Eficiência da árvore paginada

Supondo que

- ✓ Cada página de uma árvore ocupa 4KB e armazena 511 pares chave/referencia

- ✓ Cada página contém uma árvore completa perfeitamente balanceada

- ✓ Uma árvore de 3 níveis pode armazenar 134.217.727 chaves
  - Encontra-se qualquer uma das chaves com **no máximo 3 `SEEKS`**

Eficiência da árvore paginada

Pior caso de busca

- ✓ ABB completa, perfeitamente balanceada: $\log_2 (n + 1)$
- ✓ Versão paginada: $\log_{k+1} (n + 1)$
  - onde ***n*** é o número total de **chaves**, e ***k*** é o número de **chaves armazenadas em uma página**
  - Note que, na ABB tradicional, base do $\log_2$ nada mais é do que 1 chave por página + 1

- ✓ Exemplo
  - ABB: $\log_2 (134.217.727) = 27$ acessos
  - Versão paginada: $\log_{511+1} (134.217.727) = 3$ acessos

Árvores Binárias Paginadas

Desvantagens

- ✓ Maior tempo na transmissão de dados
- ✓ Necessidade de manter a organização da árvore

Surgimento das árvores-B

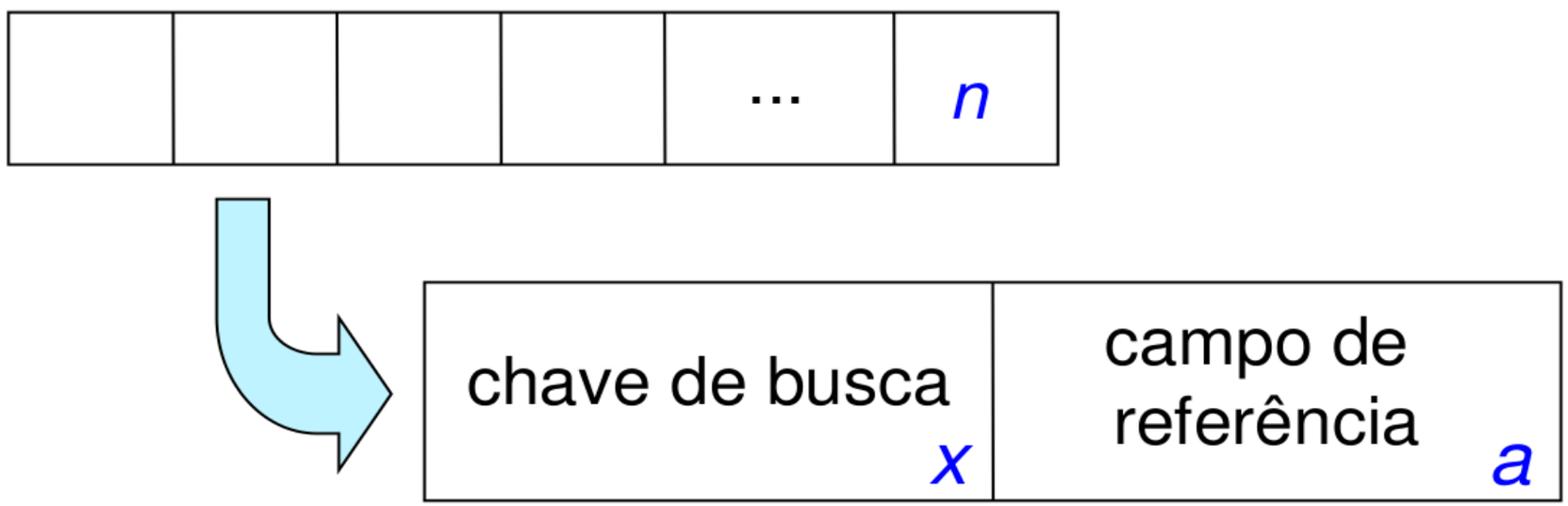
- **Árvores-B:** generalização de uma ABB paginada
  - Não binárias, com conteúdo de uma página não mantido como árvore

- **História:**
  - **1960s:** competição entre fabricantes e pesquisadores
  - **1972:** Bayer e McGreight (trabalhando pela Boeing) publicam o artigo *Organization and Maintenance of Large Ordered Indexes*
  - **1979:** árvores-B viram padrão em sistemas de arquivos de propósito geral
    - De onde vem o “B” do nome?

Características gerais

- ✓ Organizar e manter um índice para um arquivo de acesso aleatório altamente dinâmico

- ✓ **Índice**
  - **n** elementos **(x,a)** de tamanho fixo



Características gerais

Índice

- ✓ extremamente volumoso

Pool de *buffers* é pequeno

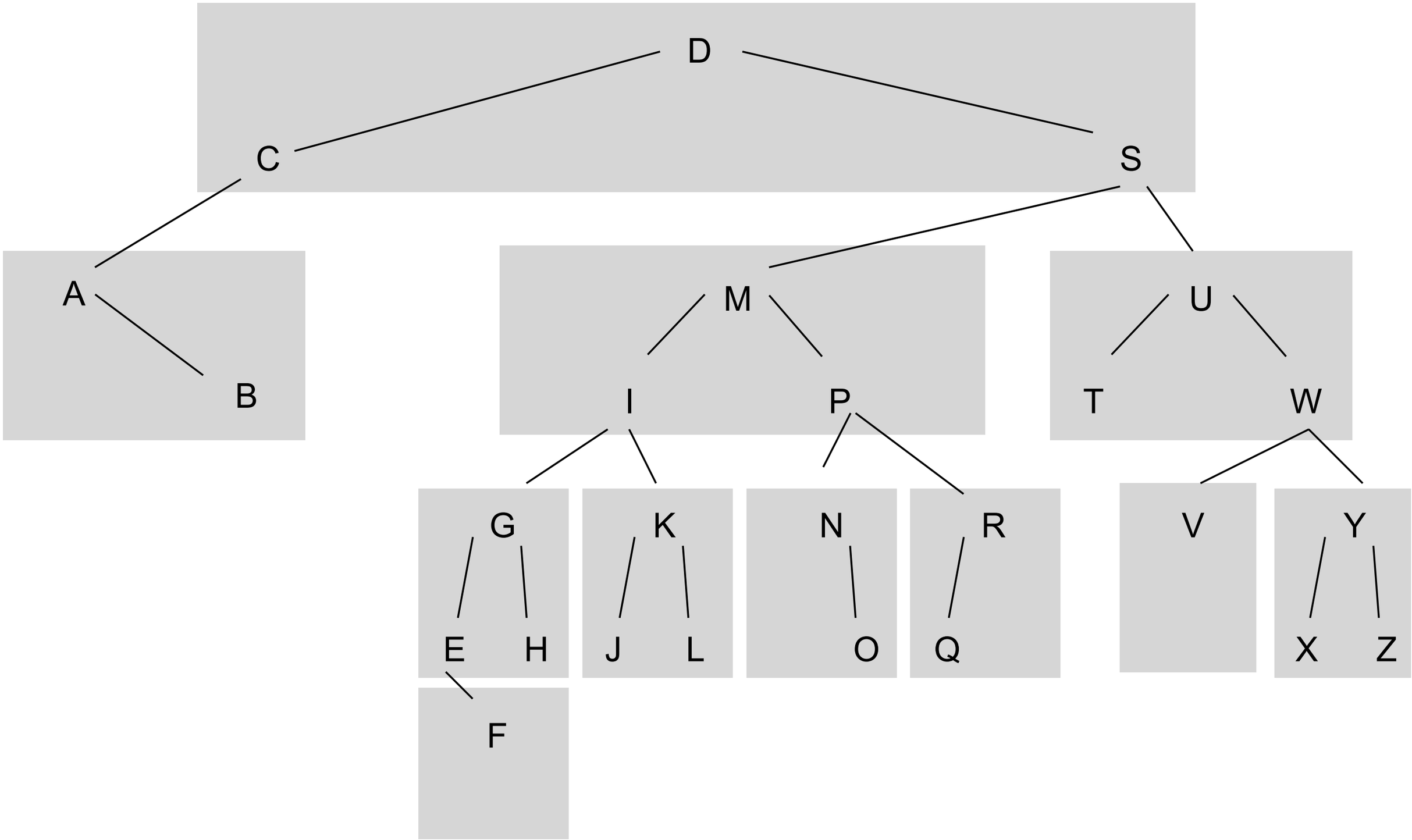
- ✓ apenas uma parcela do índice pode ser carregada em memória principal
- ✓ operações baseadas em disco

Construção Top-Down de árvores paginadas

- ✓ Se conjunto de chaves é conhecido, construção da árvore é simples
  - Inicia-se pela chave do meio para obter uma árvore balanceada
- ✓ Porém, é complicado se as chaves são recebidas em uma sequência aleatória

Construção Top-Down de árvores paginadas

Ordem: 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



Construção Top-Down de árvores paginadas

- ✓ Figura anterior: a construção foi feita ***top-down***, a partir da raiz
- ✓ Quando uma chave é inserida, a árvore dentro da página pode sofrer rotações para manter o balanceamento
- ✓ Construção a partir da raiz implica em que as chaves iniciais tendem a ficar na raiz
  - As chaves C e D não deveriam estar no topo, pois acabam desbalanceando a árvore de forma definitiva

Construção Top-Down de árvores paginadas

Questões

- ✓ Como garantir que as chaves na página raiz são boas separadoras, *i.e.*, dividem o conjunto de chaves de maneira balanceada?
- ✓ Como impedir o agrupamento de chaves que não deveriam estar na mesma página (como **C**, **D** e **S**, por exemplo)?
- ✓ Como garantir que cada página contenha um número mínimo de chaves?

Características

- ✓ Balanceada
- ✓ *bottom-up* para a criação (em disco)
  - nós folhas → nó raiz

Inovação

- ✓ Não é necessário construir a árvore a partir do nó raiz, como é feito para árvores em memória principal e para as árvores anteriores

Construção Bottom-up

Consequências

✓ Chaves indevidas não são mais alocadas na raiz

- Elimina as questões em aberto de chaves separadoras e de chaves extremas

✓ Não é necessário tratar o problema de desbalanceamento

Construção Bottom-up

Consequências

✓ Chaves indevidas não são mais alocadas na raiz

- Elimina as questões em aberto de chaves separadoras e de chaves extremas

✓ Não é necessário tratar o problema de desbalanceamento

Em uma árvore-B, as chaves na raiz da árvore emergem naturalmente

Continua na próxima aula...