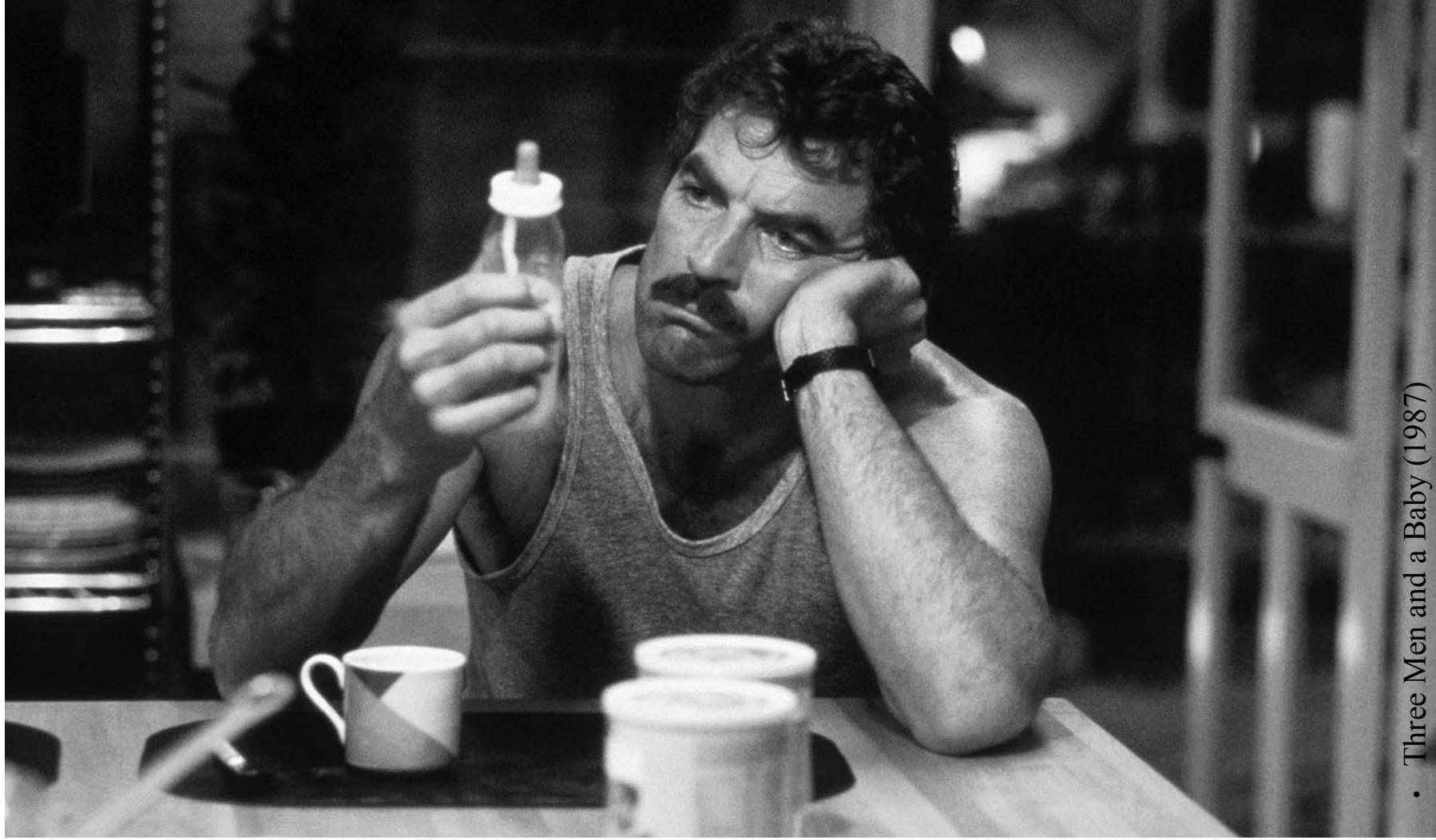
Modulo: BigData



• Three Men and a Baby (1987)

Aula #3 - Estruturas de Dados

EDUARDO CUNHA DE ALMEIDA

Agenda

- Listas e SkipList
- Arvores
- Hash
- Compressão

Agenda

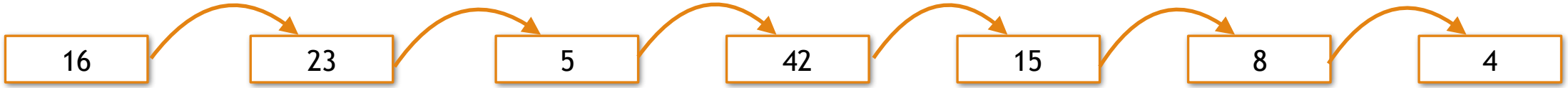
- **Listas e SkipList**
- Arvores
- Hash
- Compressão

Estruturas de Dados

Definição: estruturas para armazenamento e busca eficiente de dados

Lista: vetor (espaço contíguo de memória)

Lista: ponteiros



Estruturas de Dados

Lista: vetor

```
int v[10], tamanho=0;
// aloca memoria
void insere_lista(int valor, int *v) {
    if(tamanho < 10)
        v[tamanho++] = valor;
    else // realoca memoria
}
```

Lista: ponteiros

```
typedef struct nodo{
    int valor;
    struct no *sucessor;
} nodo;

nodo *insere_lista(int valor, no *ultimo) {
    no *novo=malloc(sizeof(no))
    novo->valor = valor;
    ultimo->sucessor= novo;
    …
    }
```

Estruturas de Dados

Lista: vetor

- Economiza memória (ponteiros implicitos)
- Custo alto de inclusão e exclusão (deslocamento de elementos)
- Previsão de crescimento é problemático

Lista: ponteiros

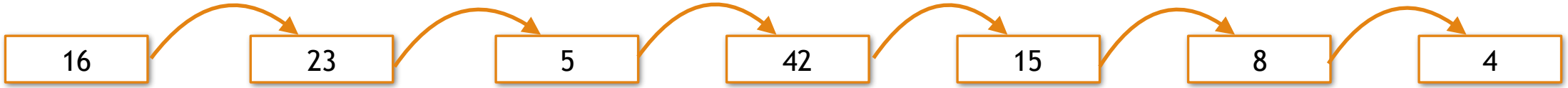
- não precisa alocar/desalocar memória em blocos
- não precisa mover(deslocar) elementos numa inclusão/exclusão
- armazena memória extra (devido aos ponteiros)

Nem sempre a ordem dos dados é mantida

Outro problema: encontre o menor valor

Lista: vetor

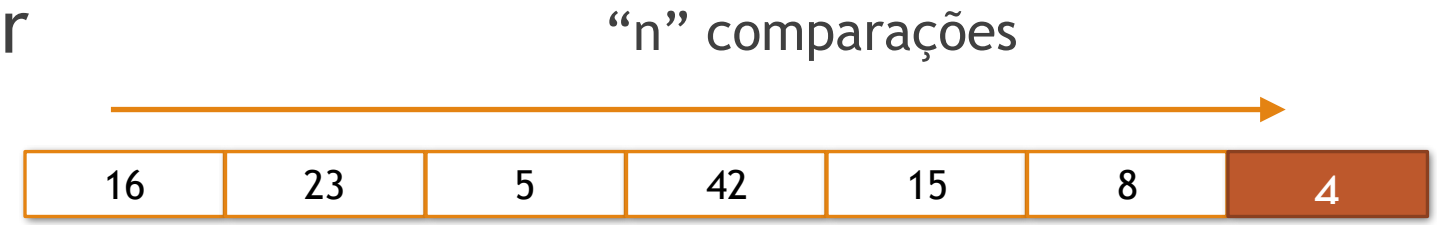
Lista: ponteiros



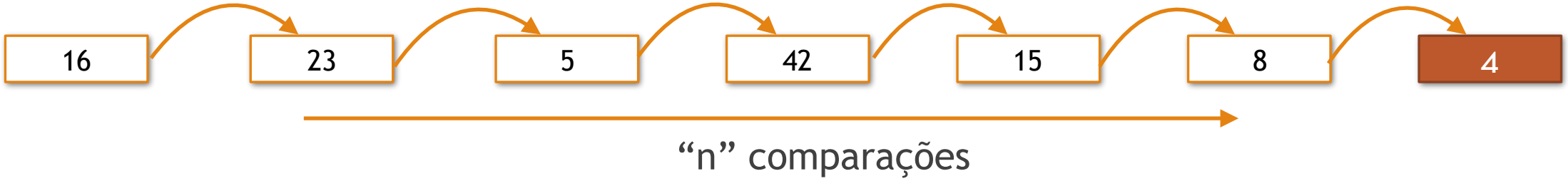
Outro problema: encontre o menor valor

Sem ordenação: O(n) comparações

Lista: vetor



Lista: ponteiros



Solução?

Solução? Ordenar !!!

Solução: Ordenar

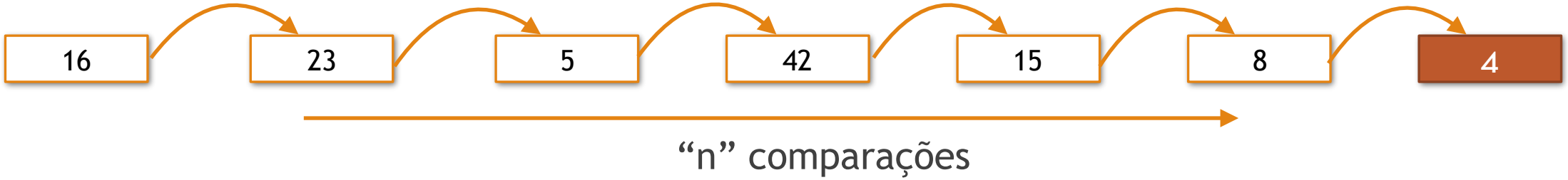
Lista: vetor

Agora posso fazer busca binária!!!!

O(log n) comparações pra achar 4

Solução: Ordenar

Lista: ponteiros … melhor não !!!! Tem coisa melhor!



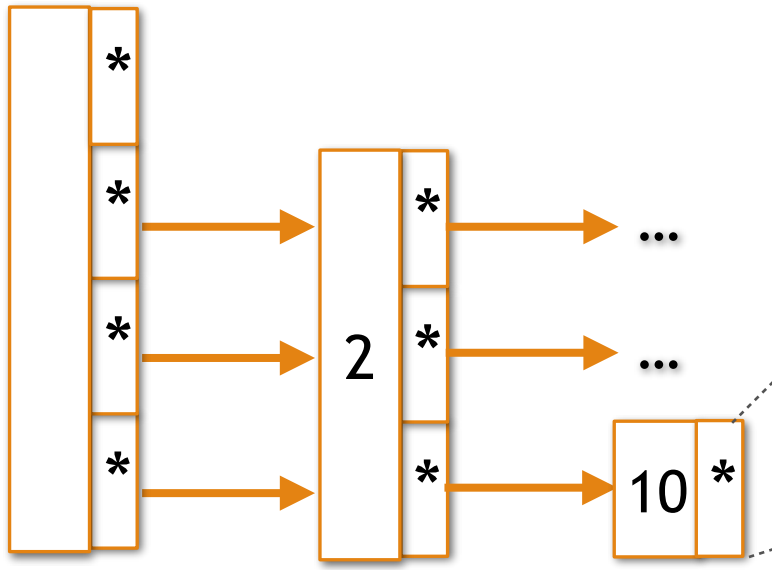
Será possível ter uma lista sem necessidade de um algoritmo de ordenação?

SkipList

- Lista encadeada que mantém a ordenação dos dados
- Busca esperada em O(log n)
- Estrutura aleatorizada (componente aleatório na construção da estrutura)

SkipList: Nodo

- Chaves
- Ponteiros para próximos elementos

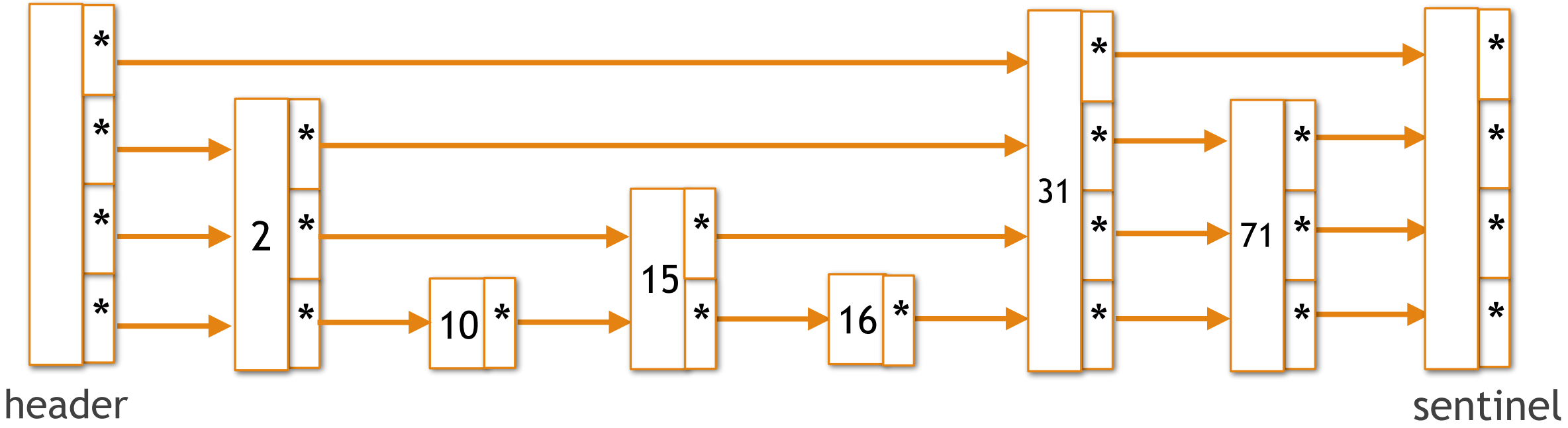



```
struct skiplist {
    int key;        /* chave  */
    int height;     /* # de ponteiros */
    struct skiplist *p[1]; /* ponteiros*/
};
```

header

SkipList

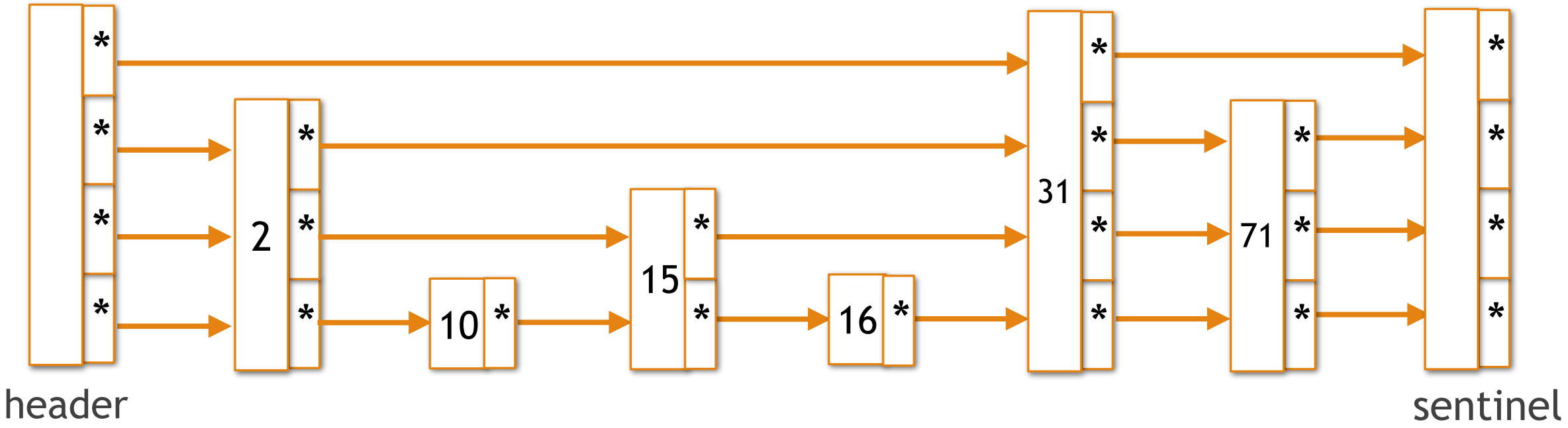
- Chaves ordenadas
- Cada nível contém 1/2 dos elementos do nível inferior
- Header e Sentinel em todos os níveis



SkipList

Exemplo: busca(71)

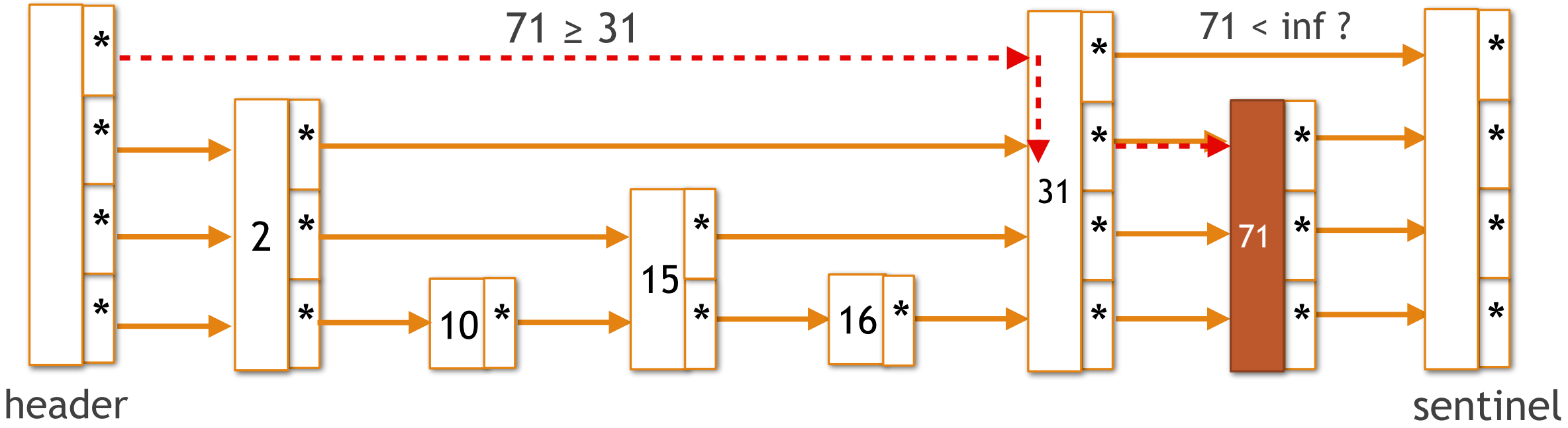
na busca(k):
If k = chave, achou!
If k < prox. chave, desce 1 nível
If k ≥ prox. chave, anda pra direita



SkipList

Exemplo: busca(71)

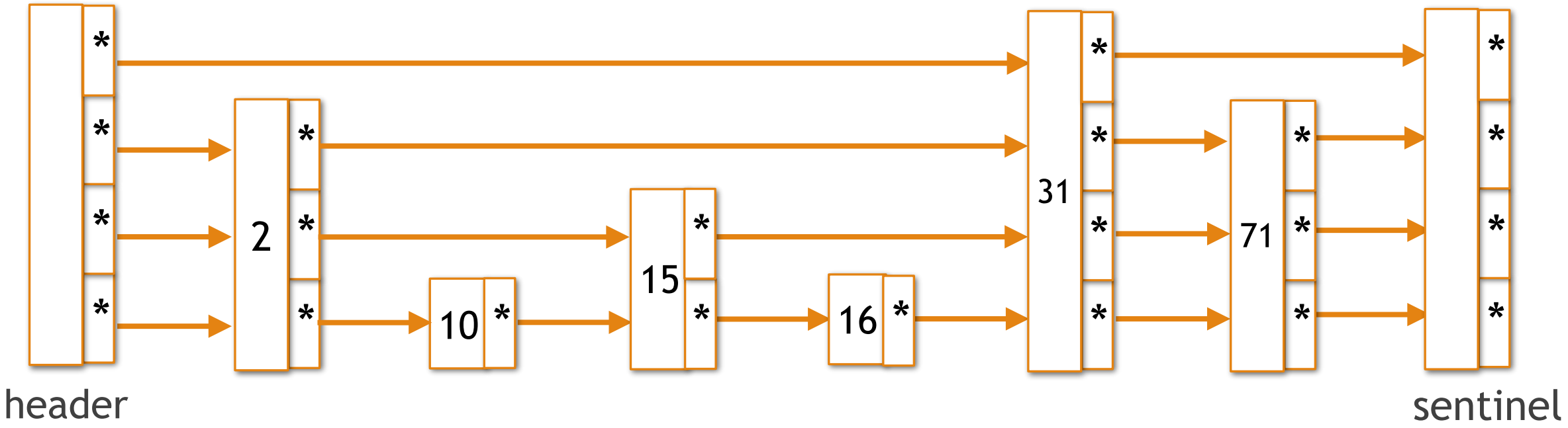
na busca(k):
If k = chave, achou!
If k < prox. chave, desce 1 nível
If k ≥ prox. chave, anda pra direita



SkipList

Outro exemplo: busca(16)

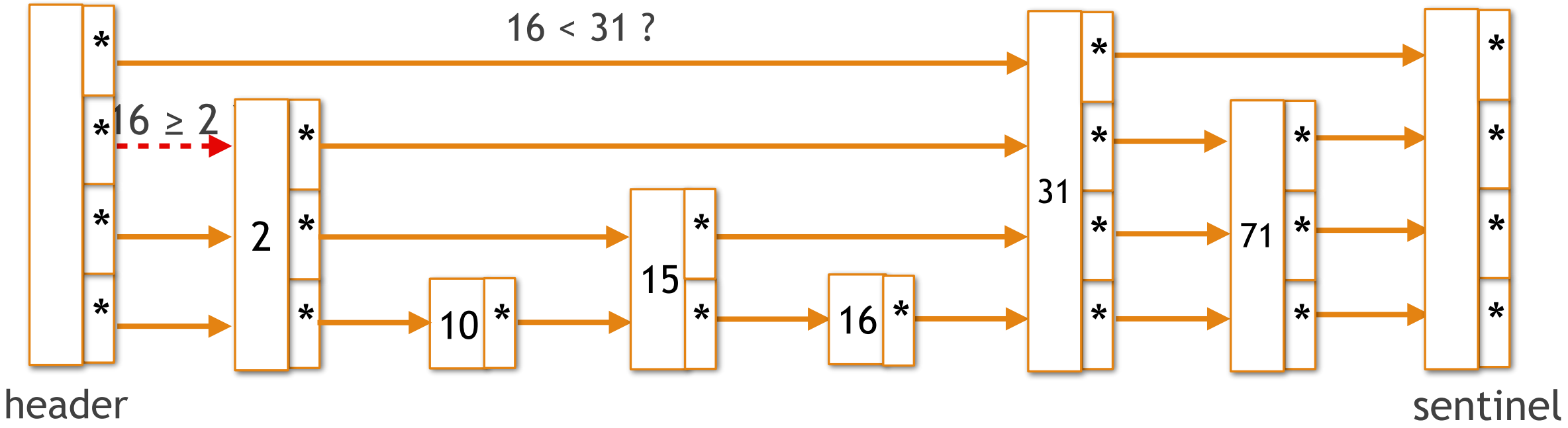
na busca(k):
If k = chave, achou!
If k < prox. chave, desce 1 nível
If k ≥ prox. chave, anda pra direita



SkipList

Outro exemplo: busca(16)

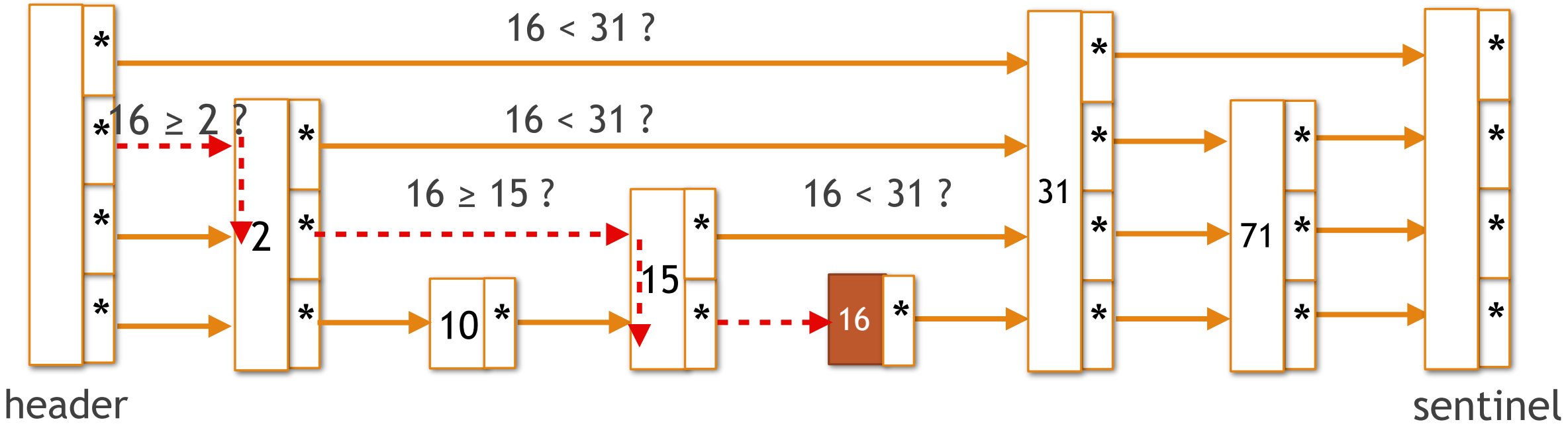
na busca(k):
If k = chave, achou!
If k < prox. chave, desce 1 nível
If k ≥ prox. chave, anda pra direita



SkipList

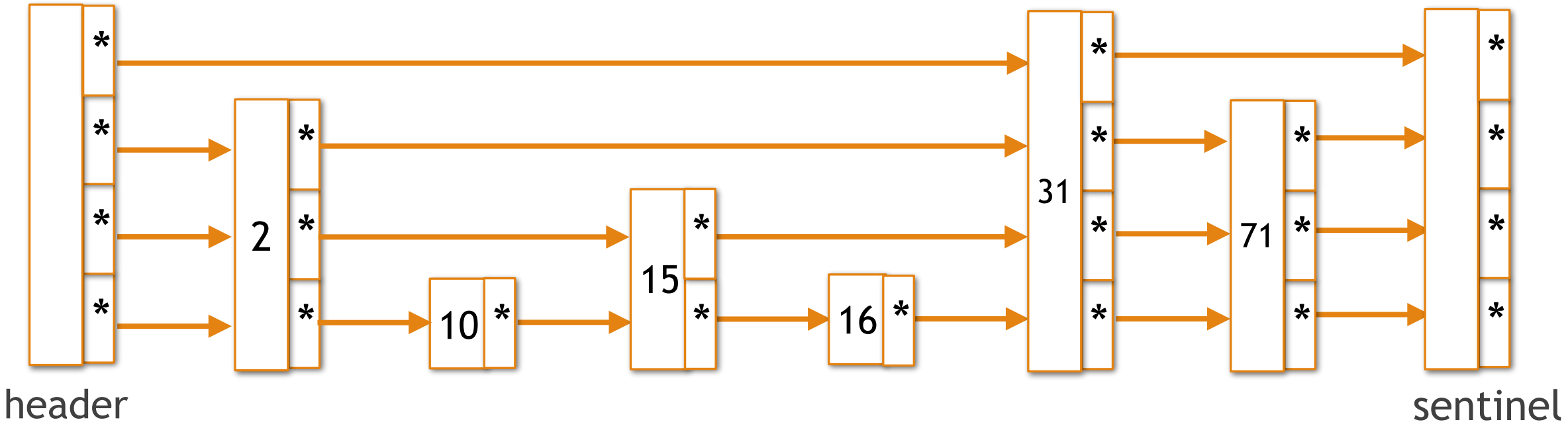
Outro exemplo: busca(16)

na busca(k):
If k = chave, achou!
If k < prox. chave, desce 1 nível
If k ≥ prox. chave, anda pra direita



SkipList: Inclusão

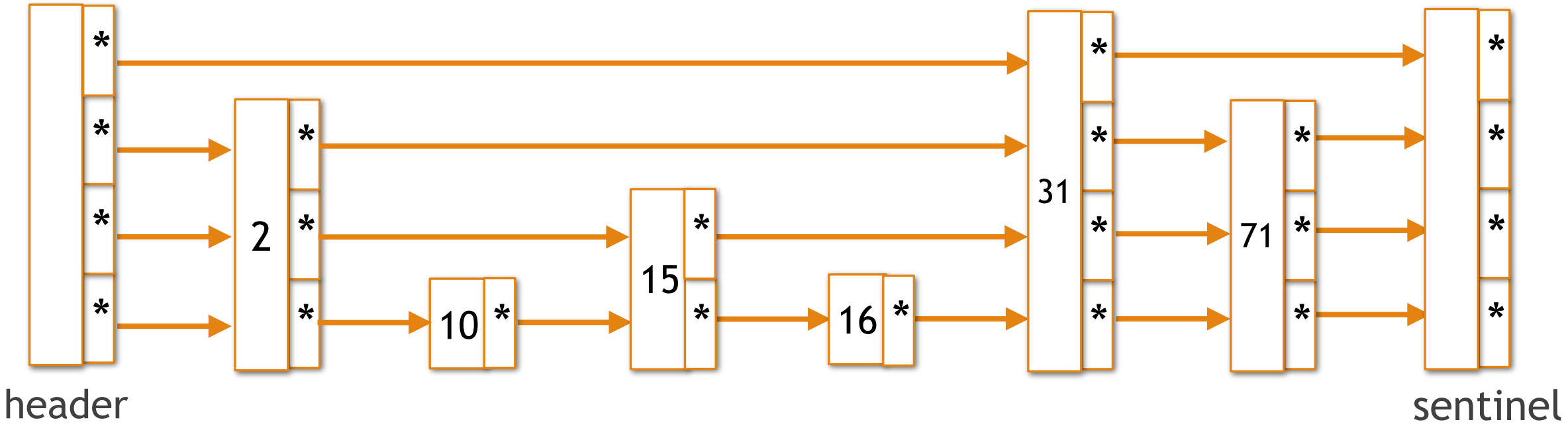
Passo 1 - Realiza a busca até achar a localização do valor na base da lista



SkipList: Inclusão

Exemplo: incluir(13)

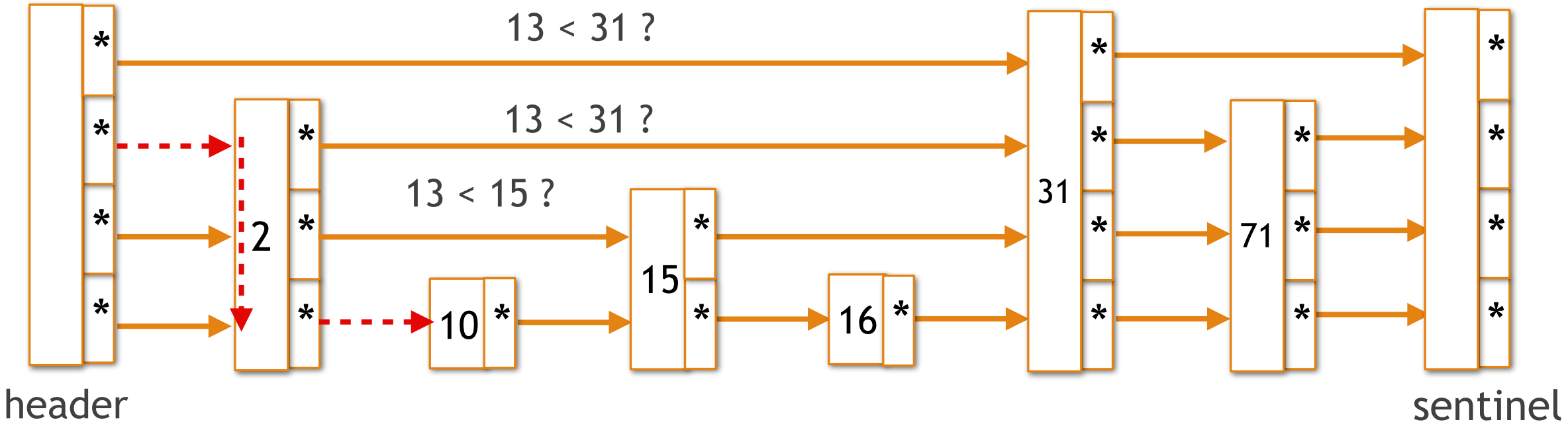
Passo 1 - Realiza a busca até achar a localização do valor na base da lista



SkipList: Inclusão

Exemplo: incluir(13)

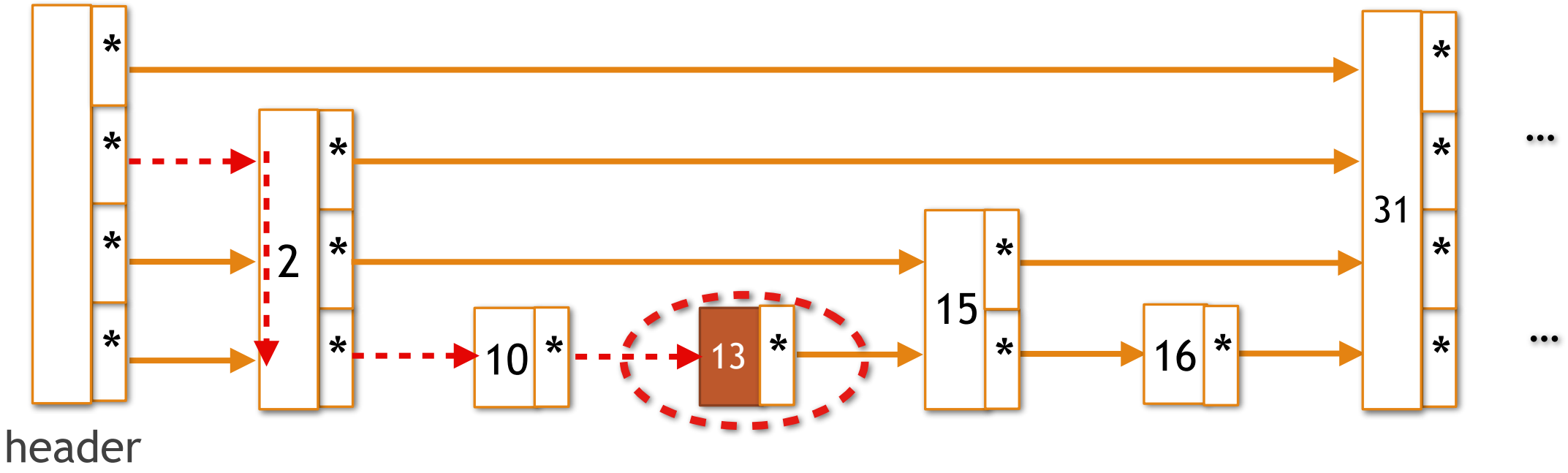
Passo 1 - Realiza a busca até achar a localização do valor na base da lista



SkipList: Inclusão

Exemplo: incluir(13)

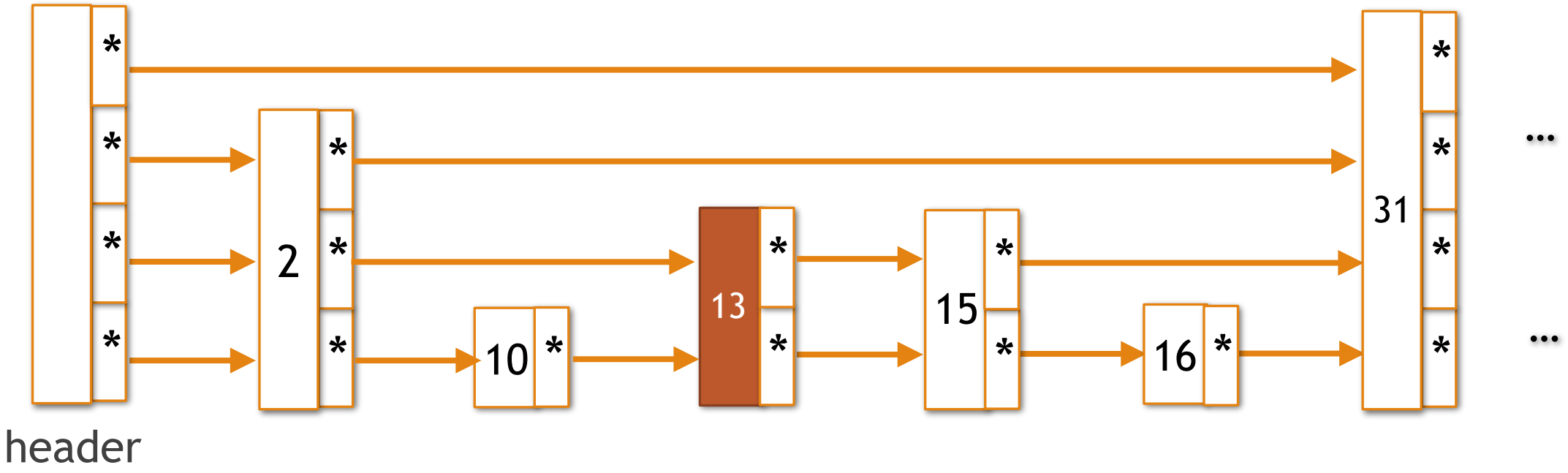
Passo 1 - Realiza a busca até achar a localização do valor na base da lista



SkipList: Inclusão

Exemplo: incluir(13)

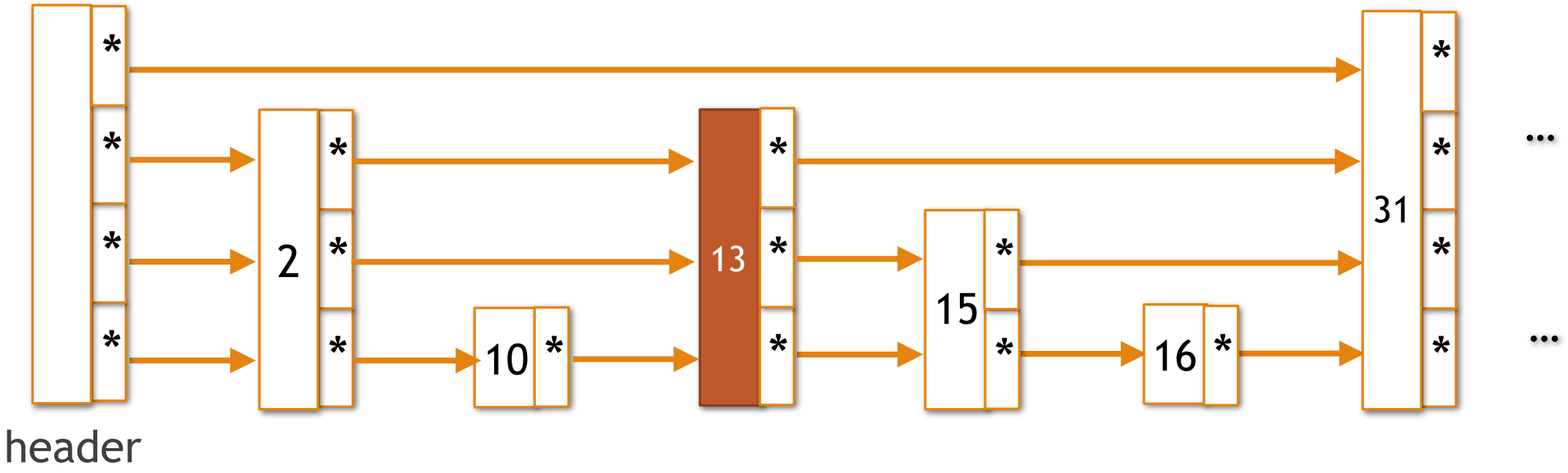
Passo 2 - “joga a moeda” para saber o número de ponteiros “skips”



SkipList: Inclusão

Exemplo: incluir(13)

Passo 2 - “joga a moeda” para saber o número de ponteiros “skips”

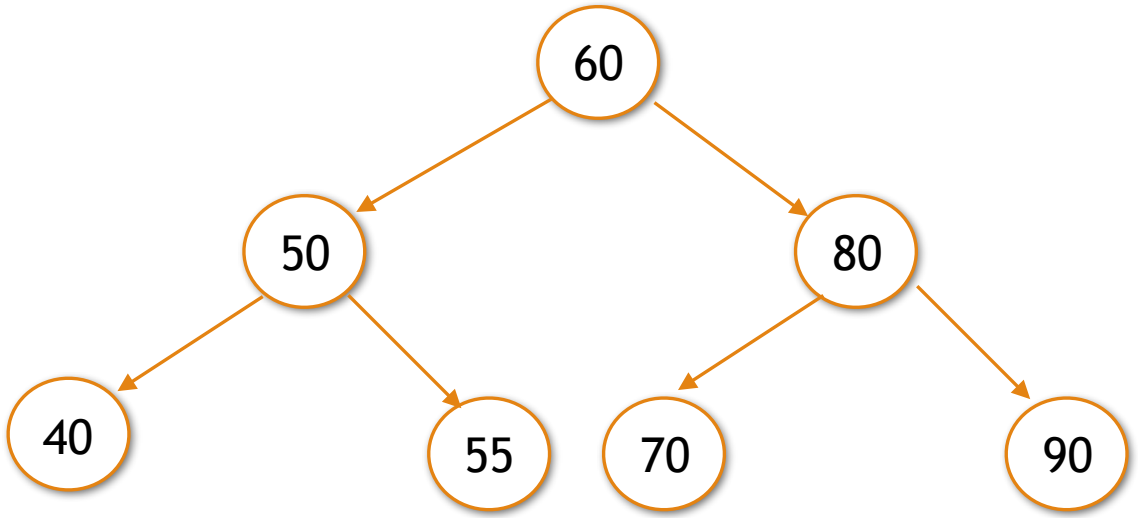


Agenda

- Listas e SkipList
- Arvores
- Hash
- Compressão

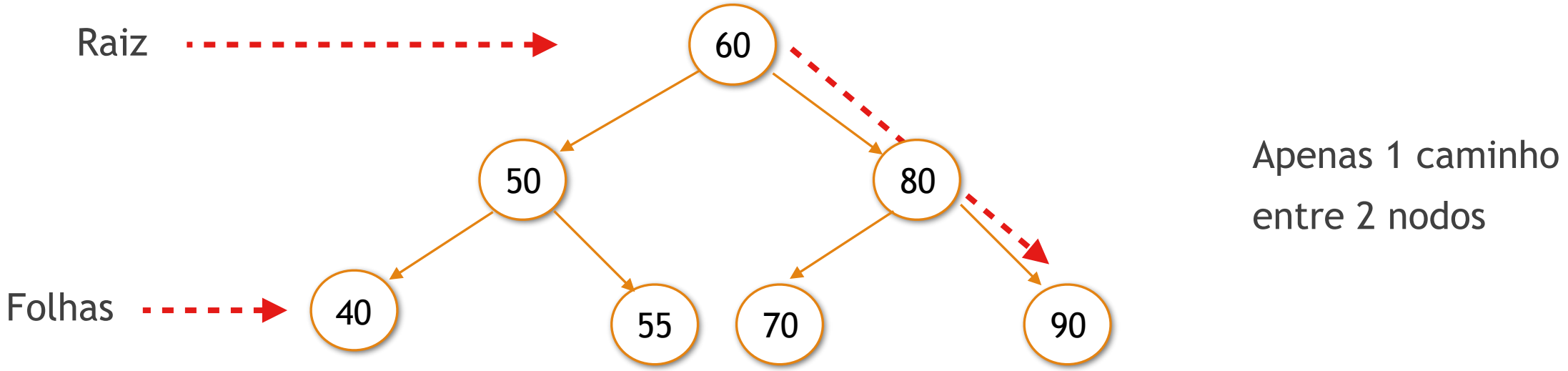
Árvores

Definição: estrutura de dados hierárquica contendo nodos ligados por ponteiros à nodos filhos.



Árvores

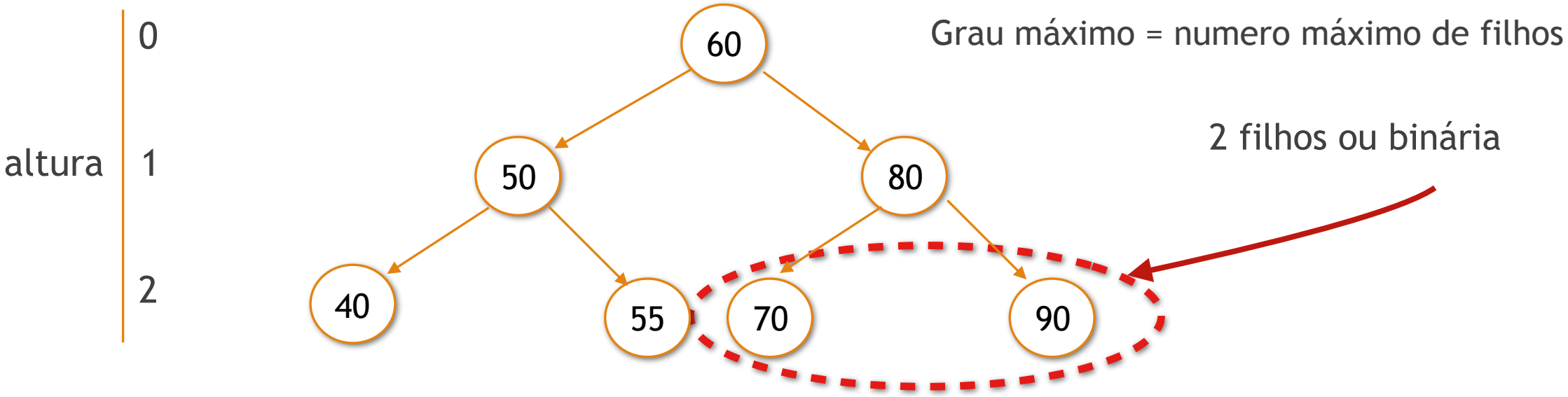
Definição: estrutura de dados hierárquica contendo nodos ligados por ponteiros à nodos filhos.



Apenas 1 caminho entre 2 nodos

Árvores

Definição: estrutura de dados hierárquica contendo nodos ligados por ponteiros à nodos filhos.



Árvores

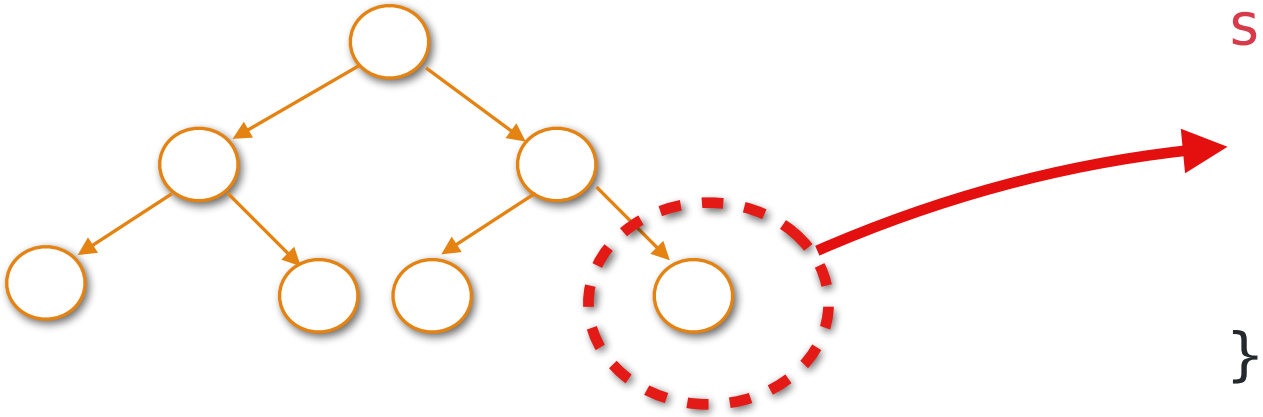
Onde usamos árvores?

R: quase tudo na computação

- Bancos de dados
- Redes de computadores
- Compiladores
- Sistemas operacionais
- Compressão

… BigData

Um nodo de uma árvore

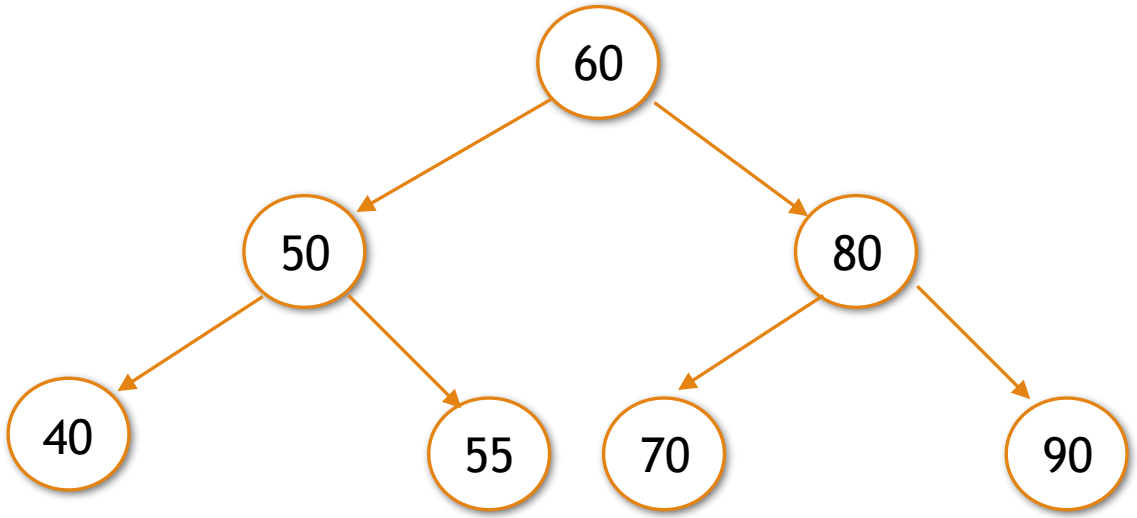


```
struct nodo {
    int             chave;
    struct t_nodo *esq, *dir;
    void            *valor;

};
```

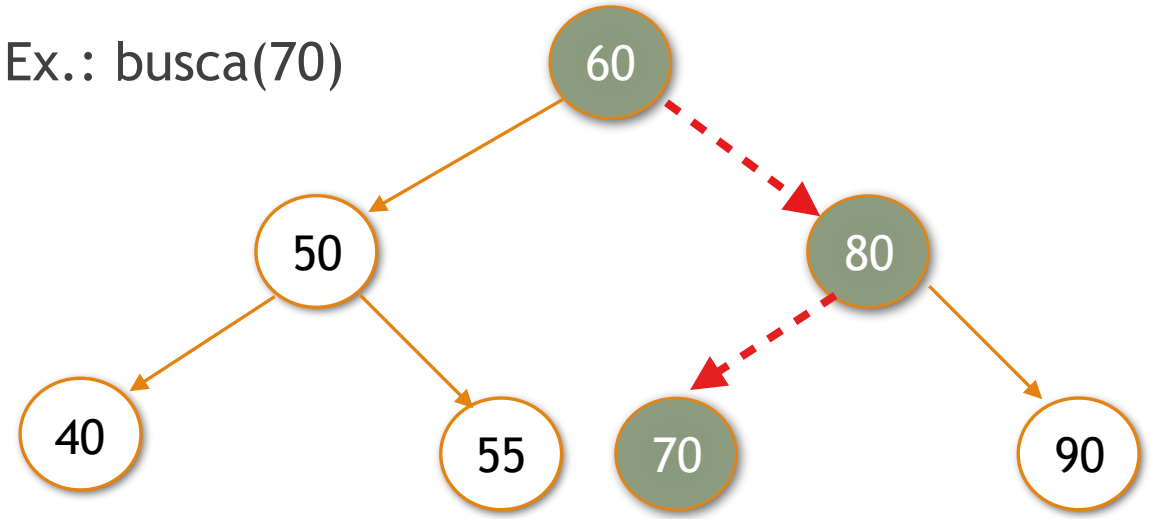
Árvore de busca binária

dado uma chave x: esq.chave <= x <= dir.chave



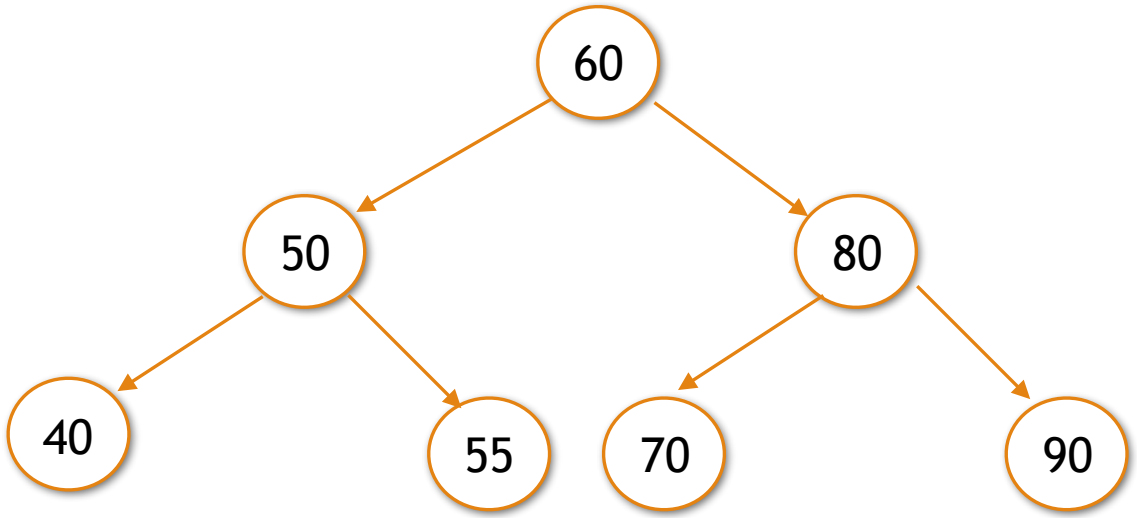
Árvore de busca binária

dado uma chave x: esq.chave <= x <= dir.chave



Árvore de busca binária

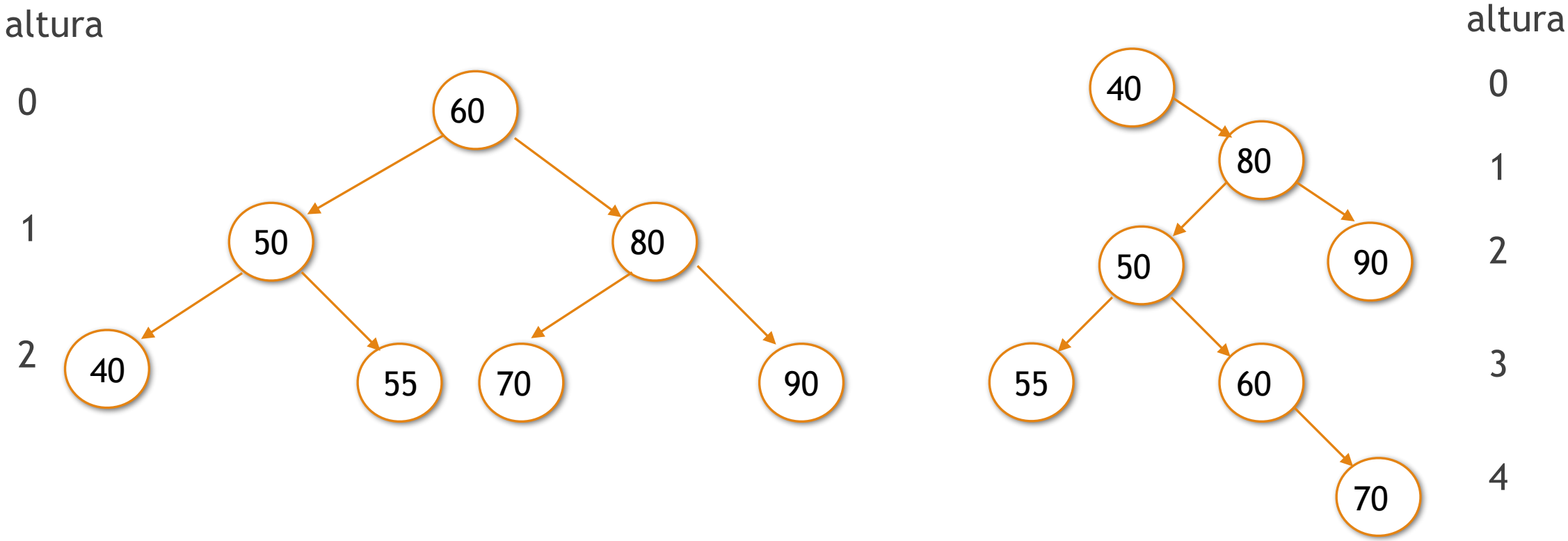
dados podem ser lidos em ordem: 40, 50, 55, 60, 70, 80, 90




```
void em_ordem(nodo x) {
    if(x){
        em_ordem(x.esq);
        printf("%d",  x.chave);
        em_ordem(x.dir);
    }
}
```

Árvore de busca binária

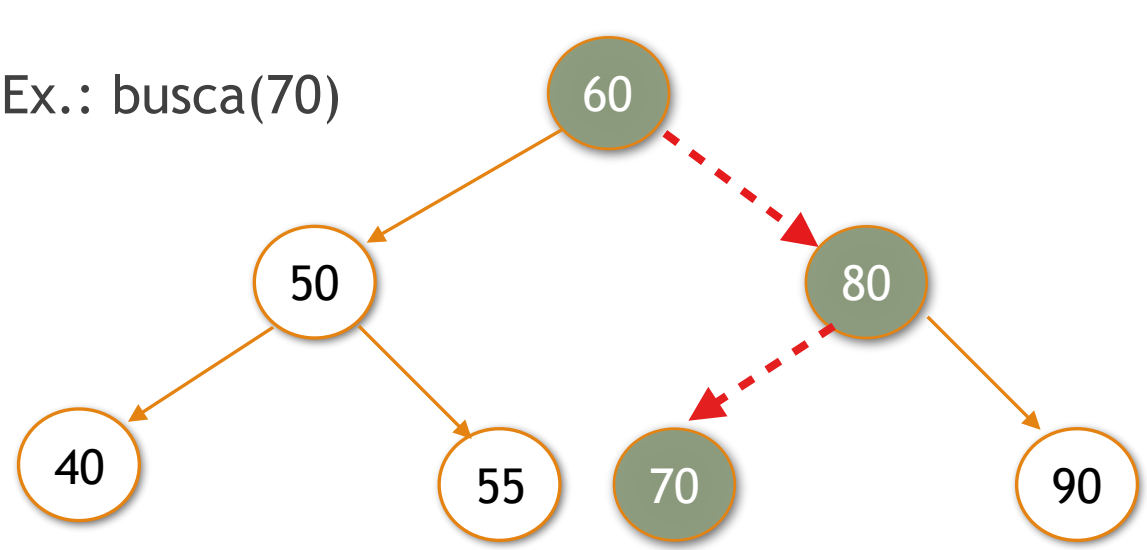
Árvores com as mesmas chaves mas com alturas diferentes

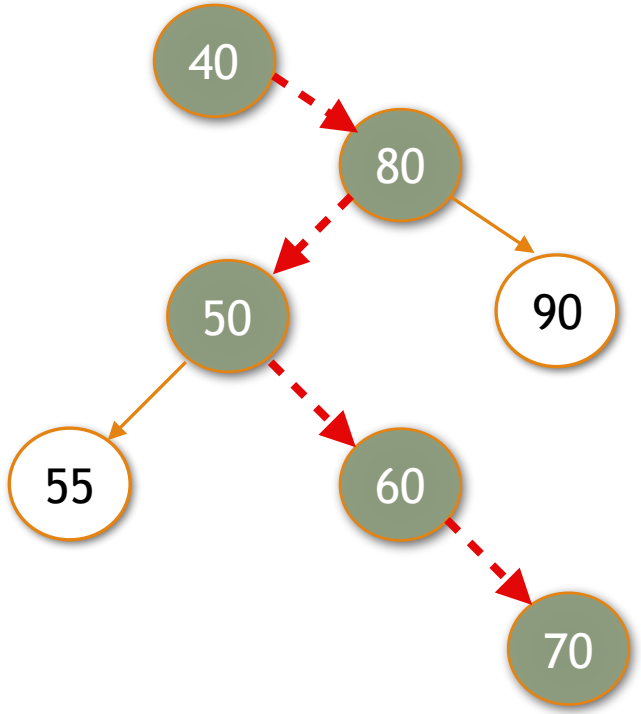


Árvore de busca binária

Árvores com as mesmas chaves mas com alturas diferentes

Ex.: busca(70)

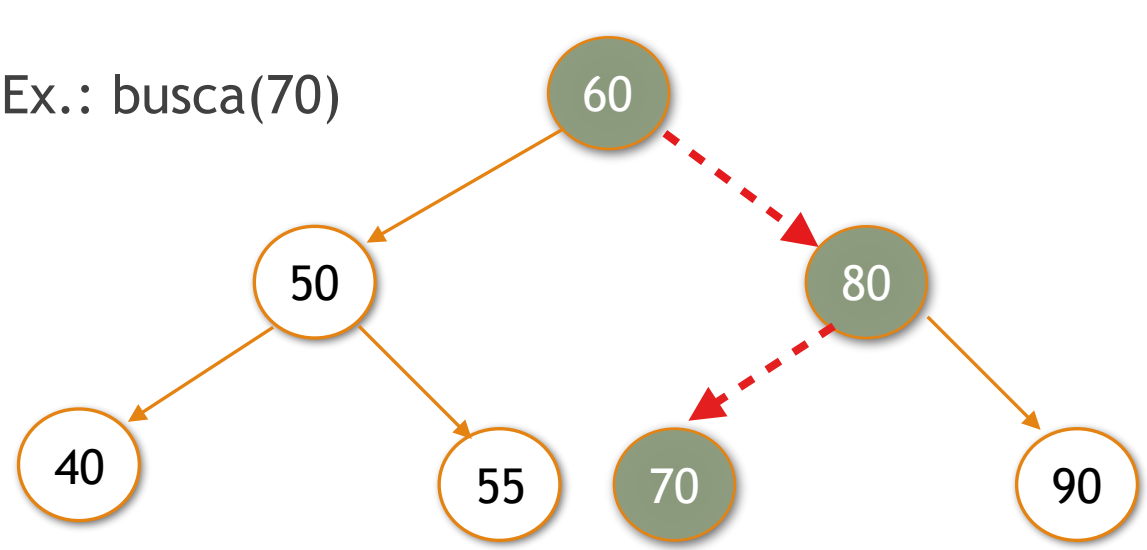


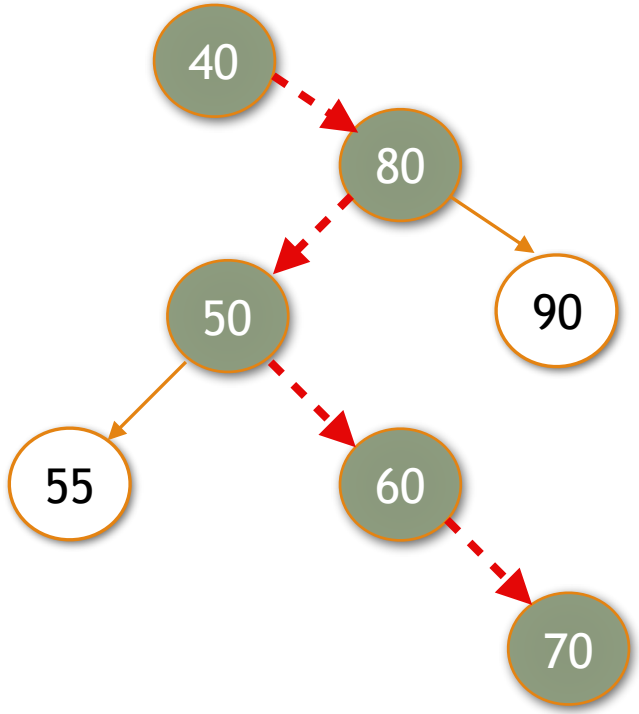


Árvore de busca binária

mas como farei busca eficiente (eg. log(n))?

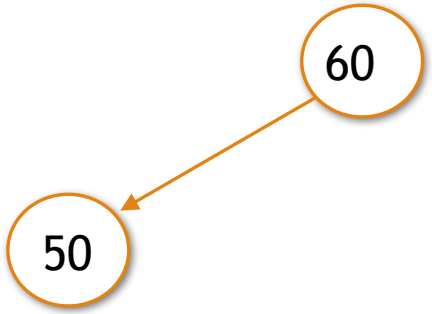
Ex.: busca(70)





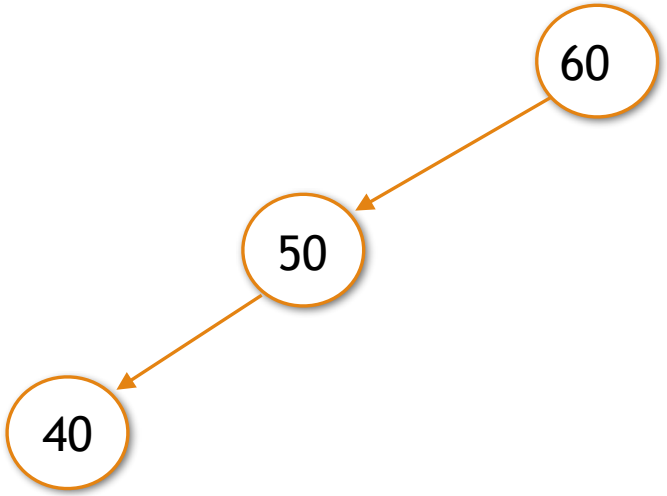
Árvore de busca binária balanceada

altura da árvore é ligada a eficiência nas operações (eg. log(n)), mas como tornar a árvore eficiente?



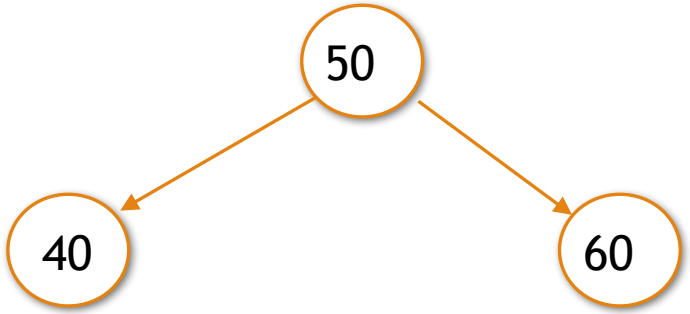
Árvore de busca binária balanceada

altura da árvore é ligada a eficiência nas operações (eg. log(n)), mas como tornar a árvore eficiente?



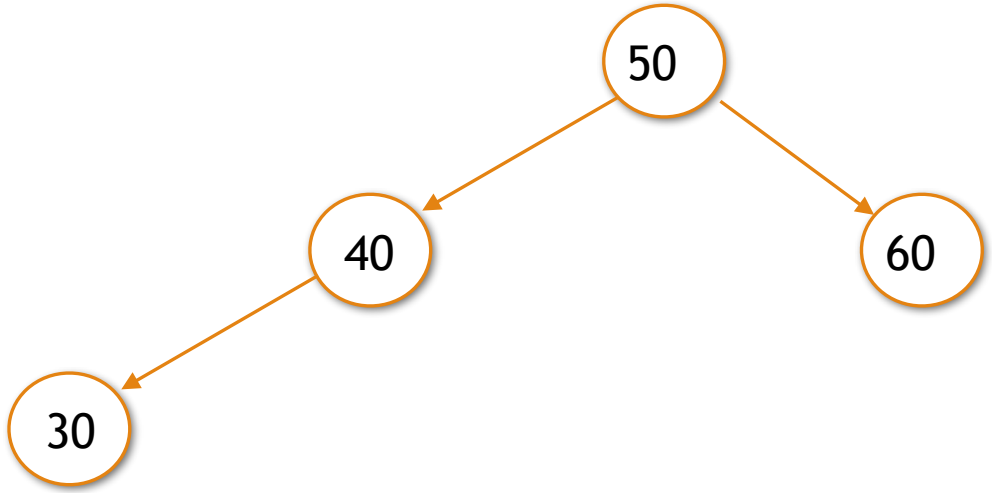
Árvore de busca binária balanceada

altura da árvore é ligada a eficiência nas operações (eg. log(n)), mas como tornar a árvore eficiente?



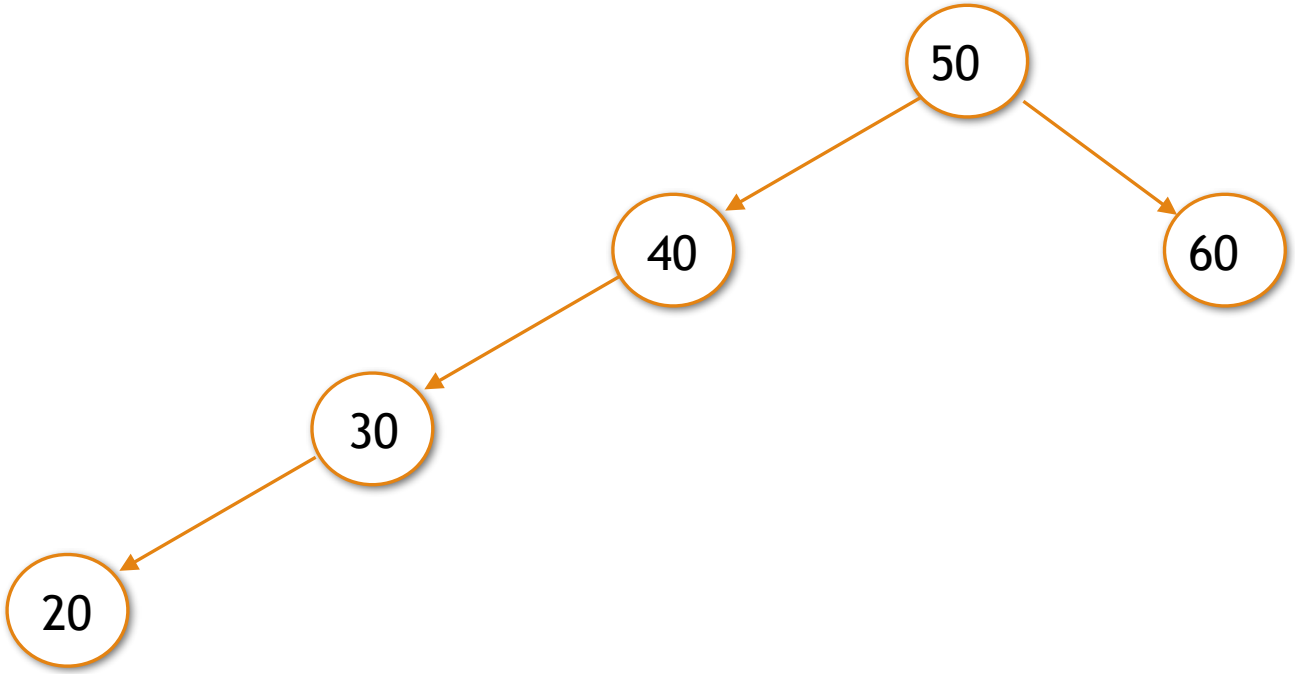
Árvore de busca binária balanceada

altura da árvore é ligada a eficiência nas operações (eg. log(n)), mas como tornar a árvore eficiente?



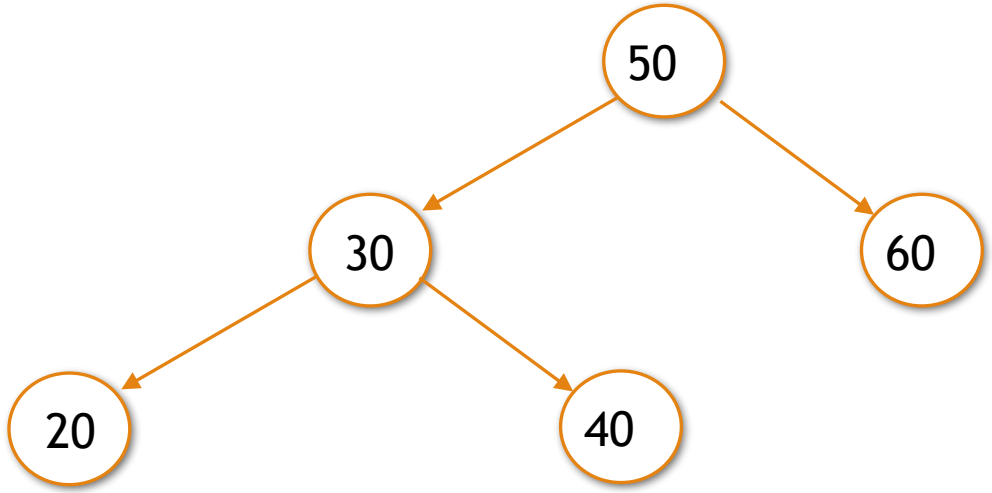
Árvore de busca binária balanceada

altura da árvore é ligada a eficiência nas operações (eg. log(n)), mas como tornar a árvore eficiente?



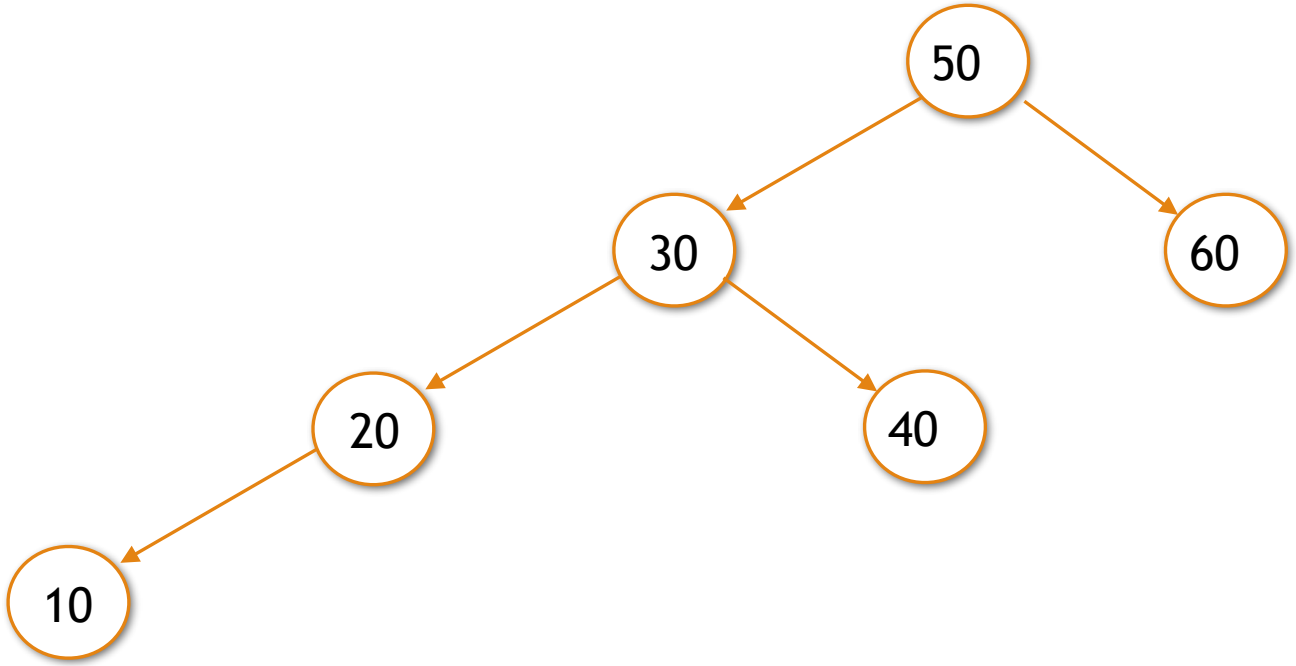
Árvore de busca binária balanceada

altura da árvore é ligada a eficiência nas operações (eg. log(n)), mas como tornar a árvore eficiente?



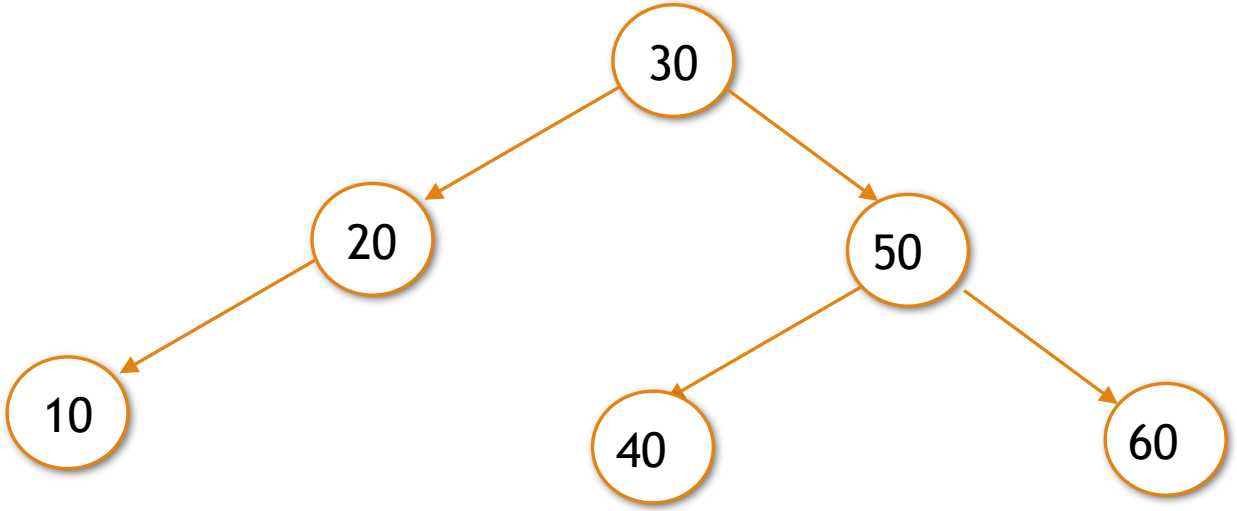
Árvore de busca binária balanceada

altura da árvore é ligada a eficiência nas operações (eg. log(n)), mas como tornar a árvore eficiente?



Árvore de busca binária balanceada

altura da árvore é ligada a eficiência nas operações (eg. log(n)), mas como tornar a árvore eficiente?



Árvore de busca binária balanceada

Onde usamos árvores de busca binária balanceadas?

R: Assumindo que essa árvore cabe em memória

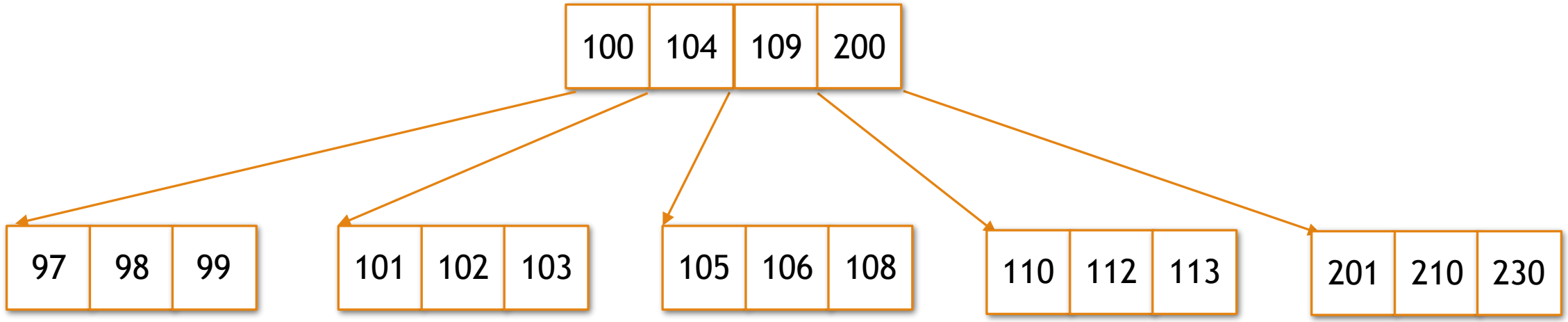
- agendamento de processos de sistemas operacionais
- cache de sistemas
- roteadores de internet
- bibliotecas de programação (eg., Java TreeMap)
- compiladores na avaliação de expressões matemáticas

…

Será possível ter uma árvore que não cabe inteiramente em memória?

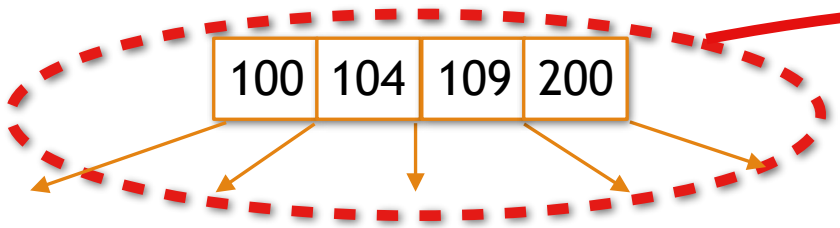
Árvore n-ária: Árvore-B

- Nodos armazenam “n” chaves



Árvore-B

- Nodos armazenam “n” chaves

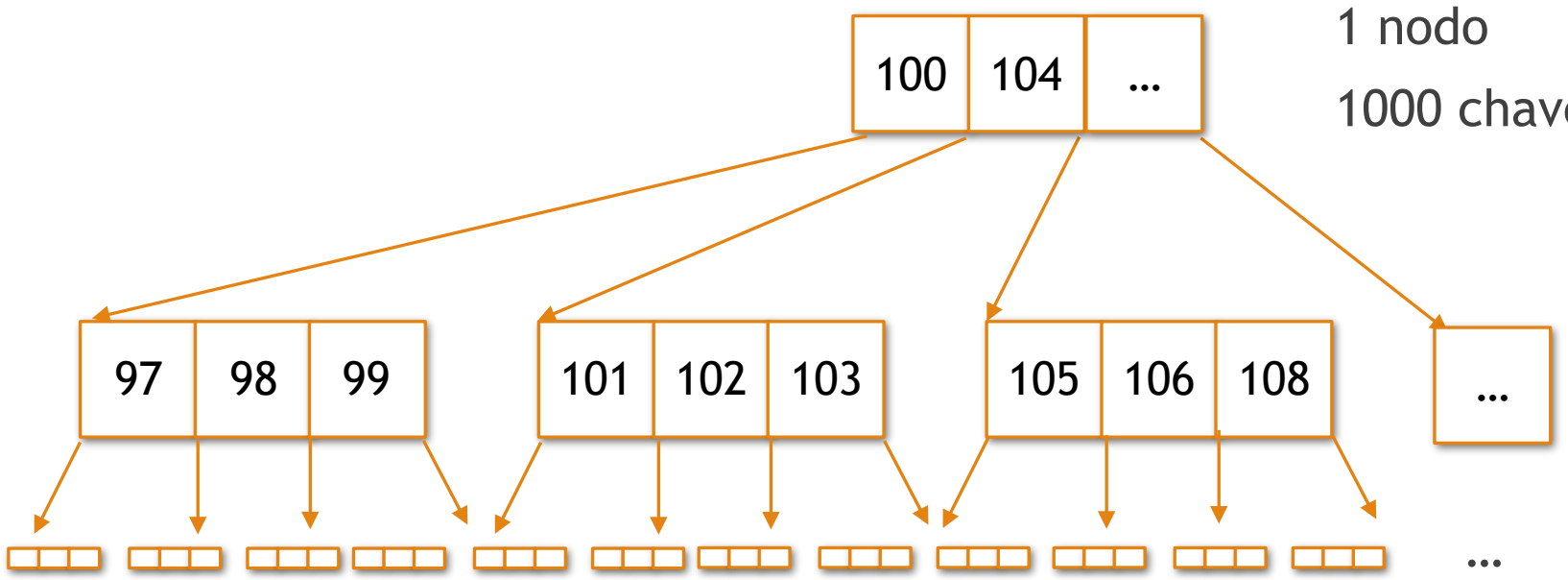



```
struct nodo {
    int n; // # de chaves
    int chaves[M-1];
    struct node *p[M];

};
```

Árvore-B

- Nodos armazenam “n” chaves



1 nodo

1000 chaves

1001 nodos

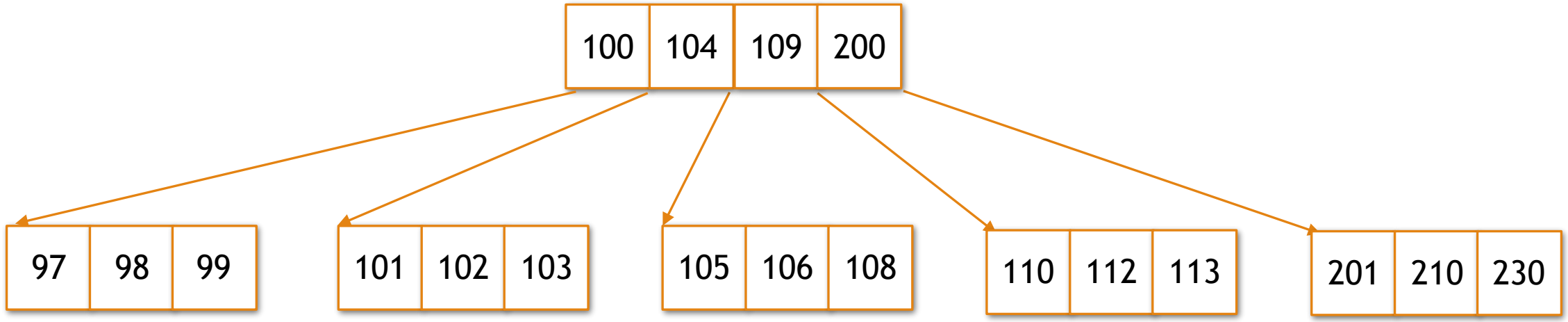
1.001.000 chaves

1.002.001 nodos

1.002.100.000 chaves

Árvore-B

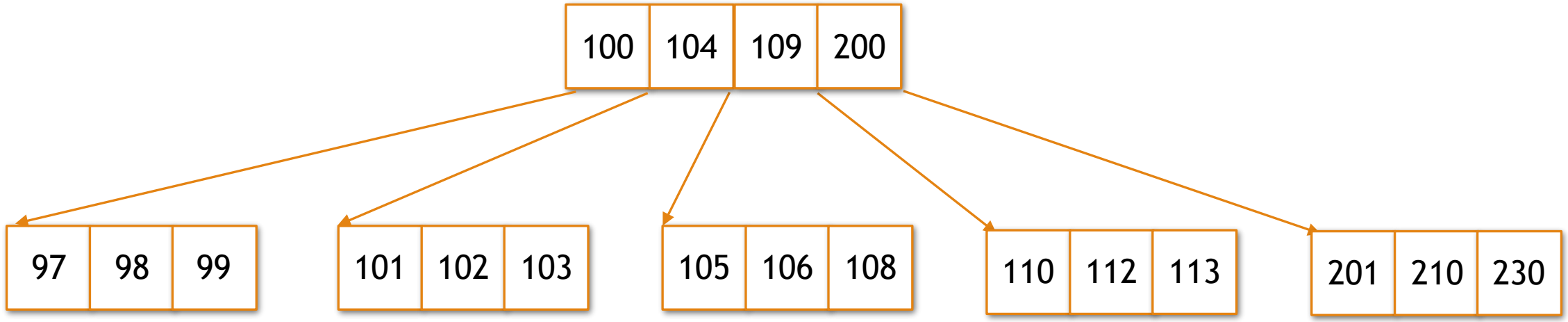
- Nodos armazenam “n” chaves
- Espaço de chaves indicam caminho pra sub-arvores
- Todas os nodos folha na mesma altura



Árvore-B

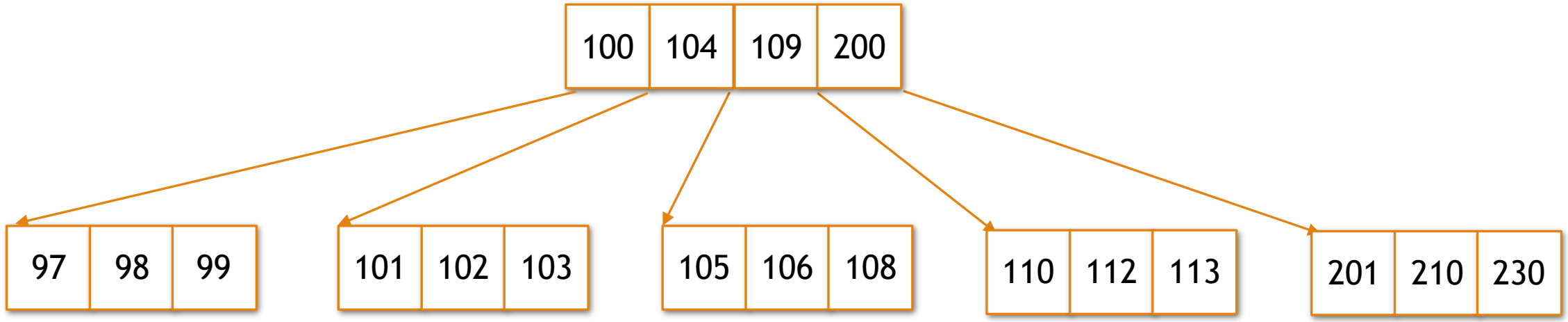
dado uma chave x:

- percorre nodo
- desce quando esq.chave < x < dir.chave



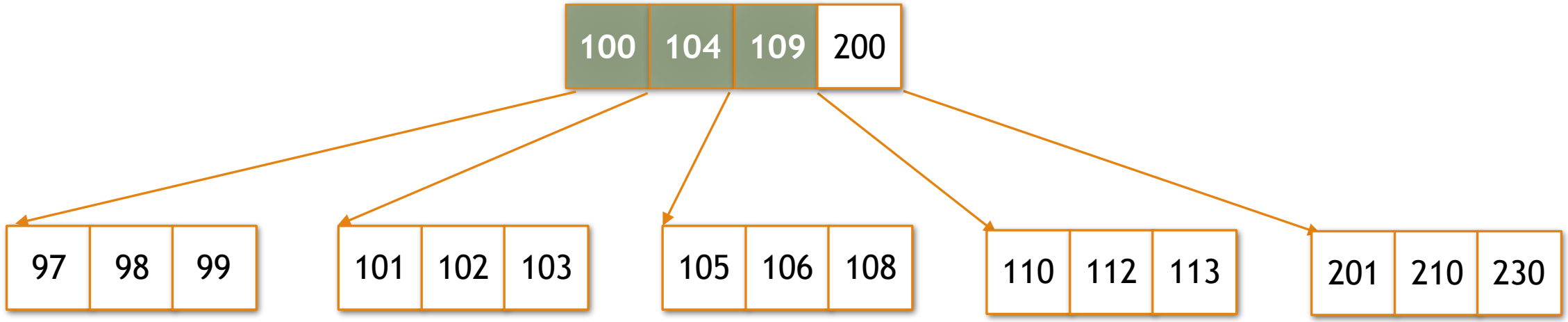
Árvore-B

Exemplo: busca(112)



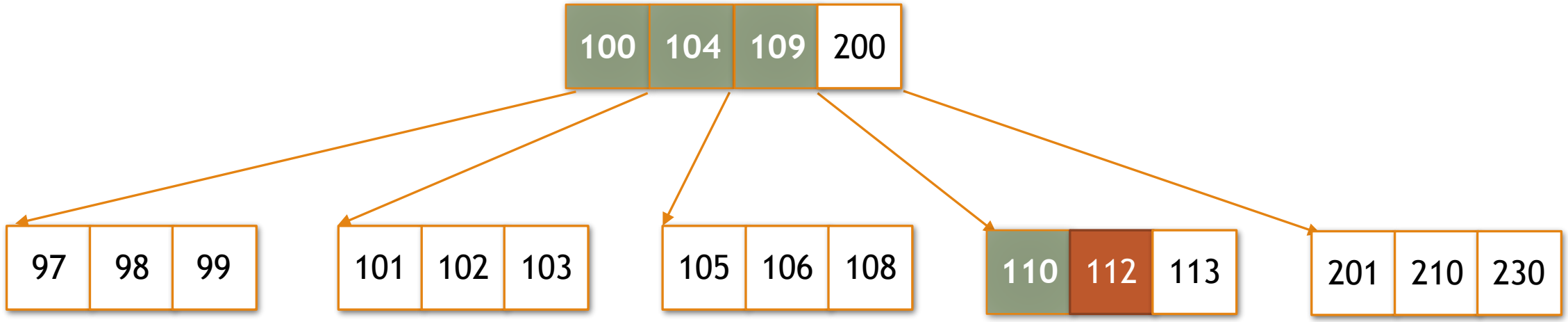
Árvore-B

Exemplo: busca(112)



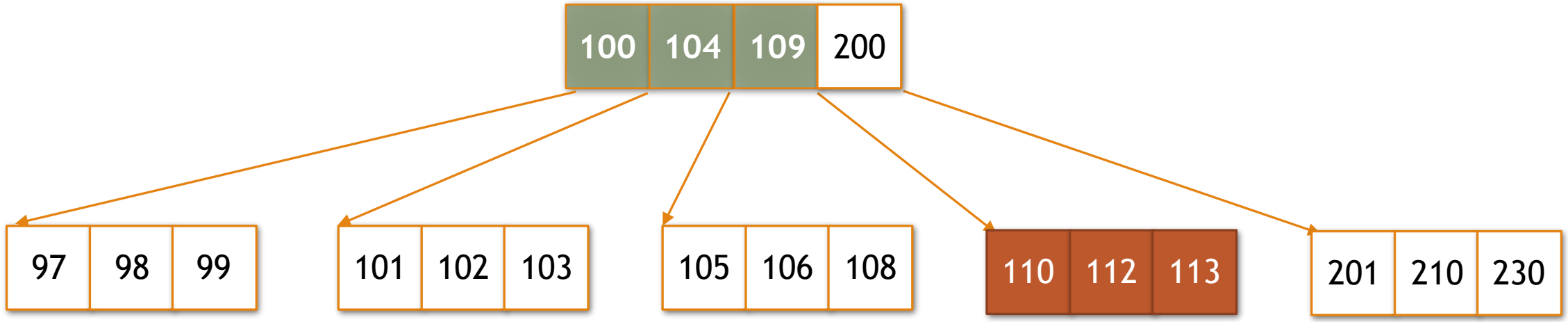
Árvore-B

Exemplo: busca(112)



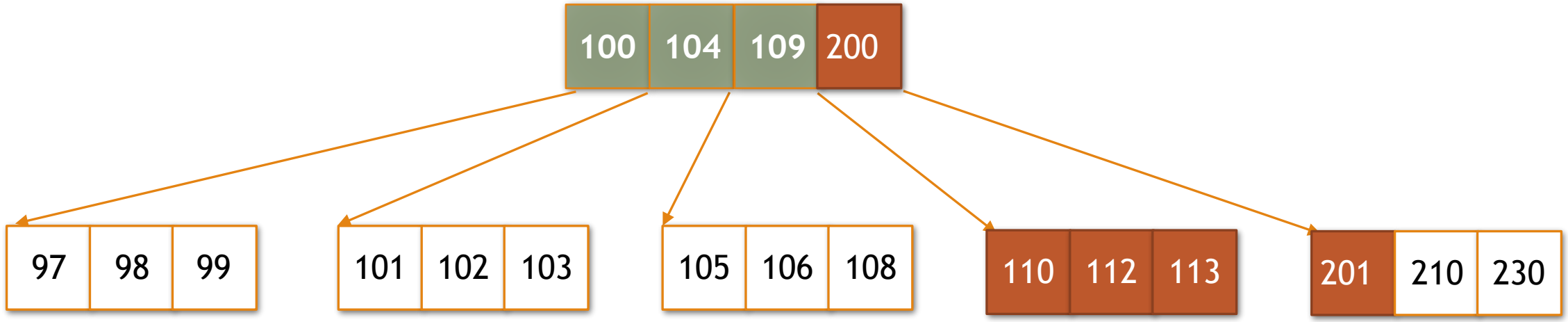
Árvore-B

Exemplo: busca(entre 110 e 113)



Árvore-B

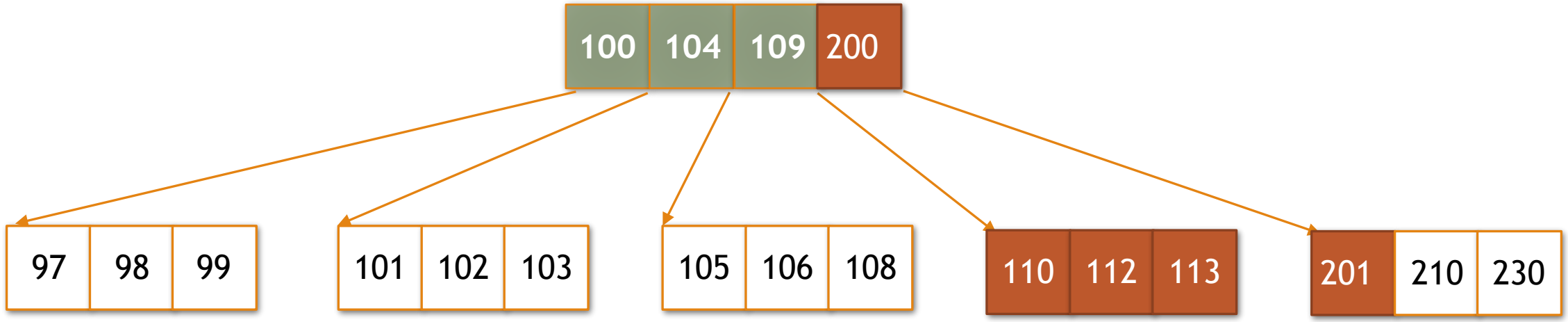
Exemplo: busca(entre 110 e 201)



Árvore-B

Exemplo: busca(entre 110 e 201)

-> Ineficiente



Árvore B+

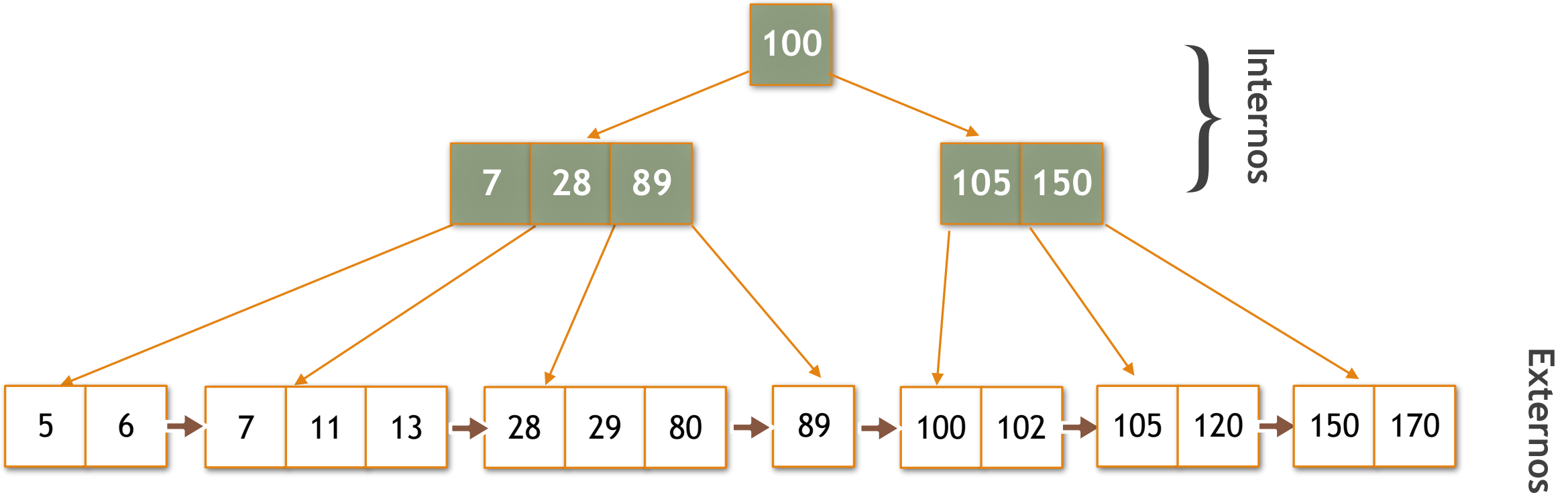
Árvore n-ária derivada da Árvore-B

Utilizada para minimizar operações de leitura de disco rígido

- sistemas de bancos de dados: indexação e armazenamento
- sistemas de arquivos: Linux, Unix e Windows

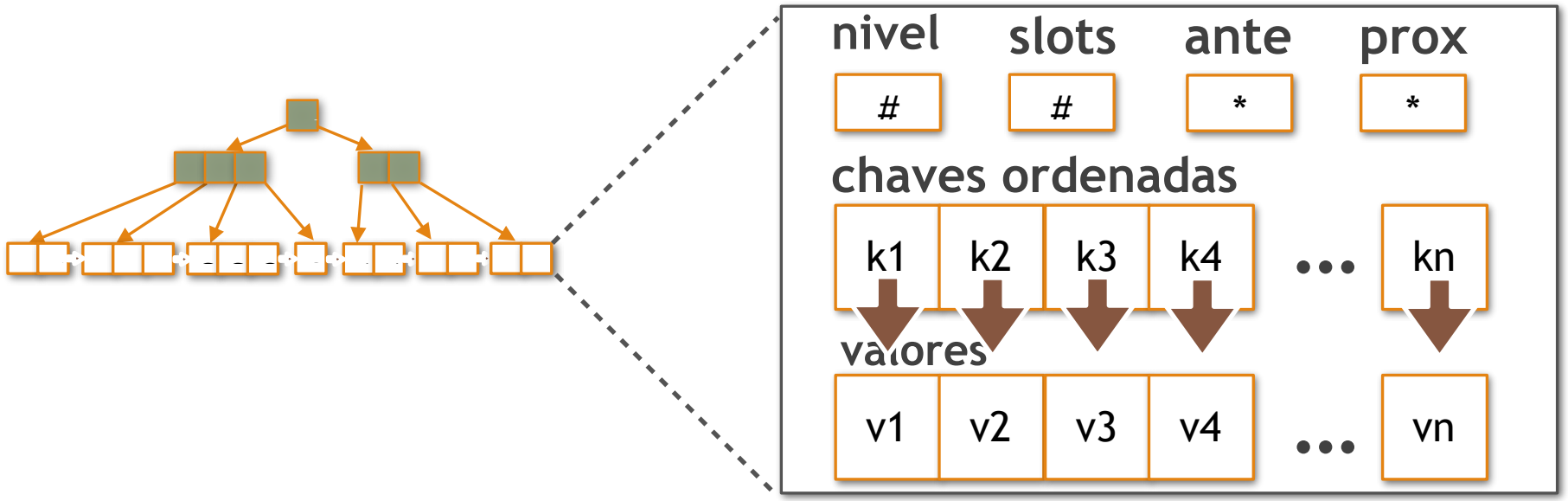
Árvore B+

- nodos internos guardam chaves
- nodos externos guardam chaves, valores e ponteiros entre eles



Árvore B+: nodos externos (folha)

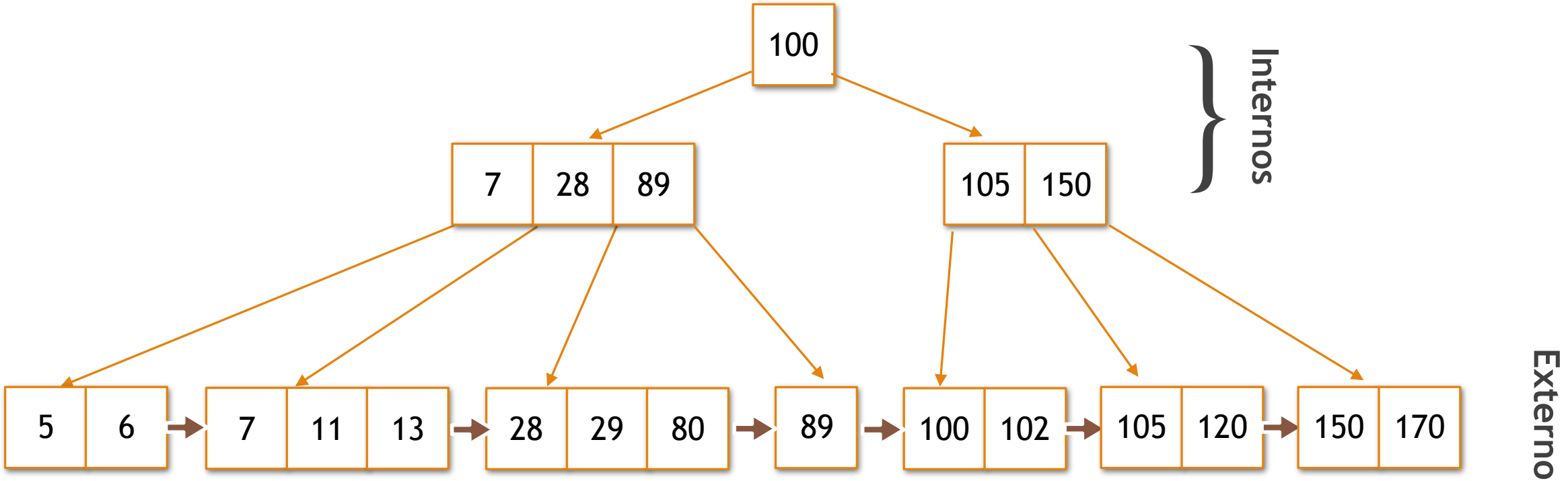
- nodos internos guardam chaves
- nodos externos guardam chaves, valores e ponteiros entre eles



Fonte: Andy Pavlo - Query Processing

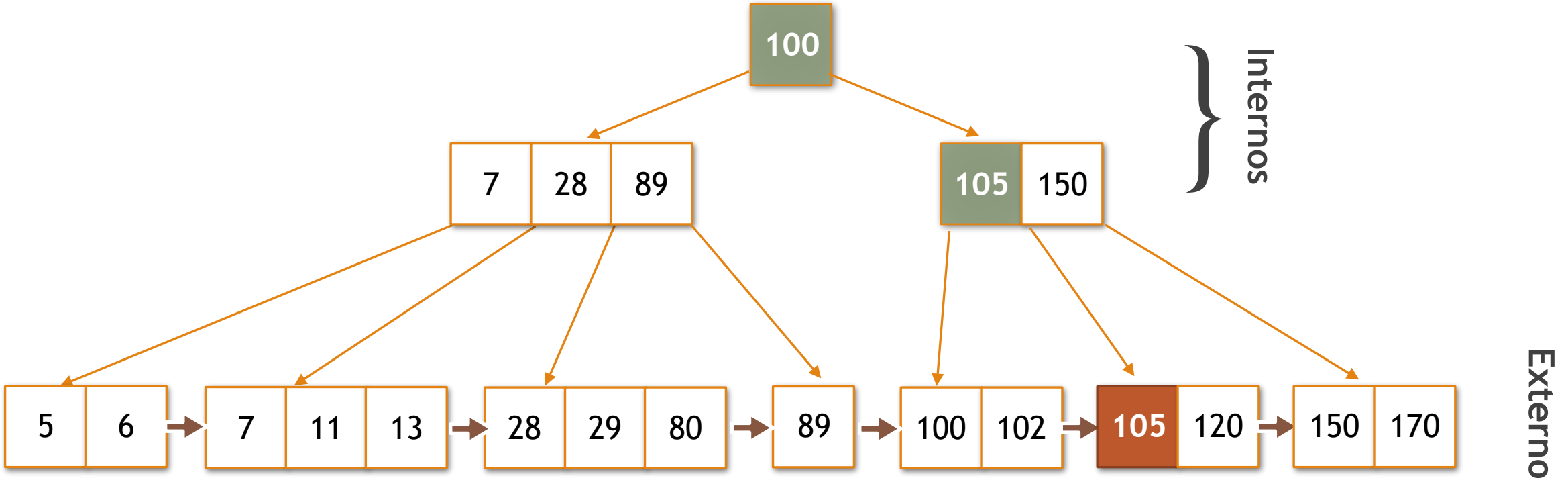
Árvore B+

busca(105)



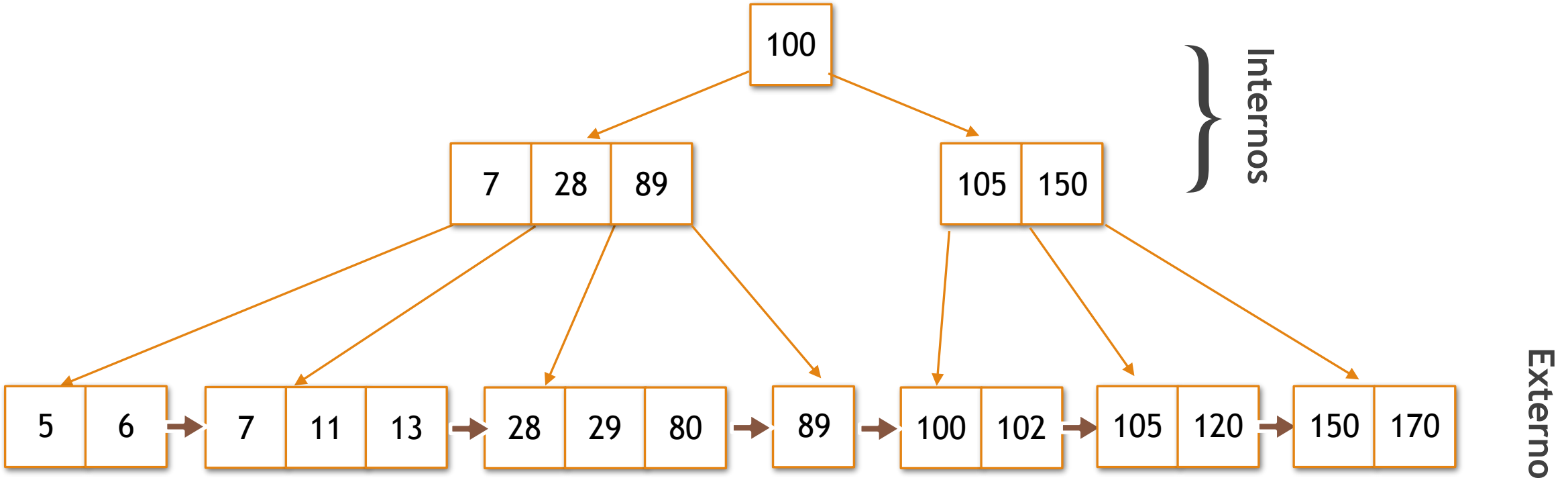
Árvore B+

busca(105)



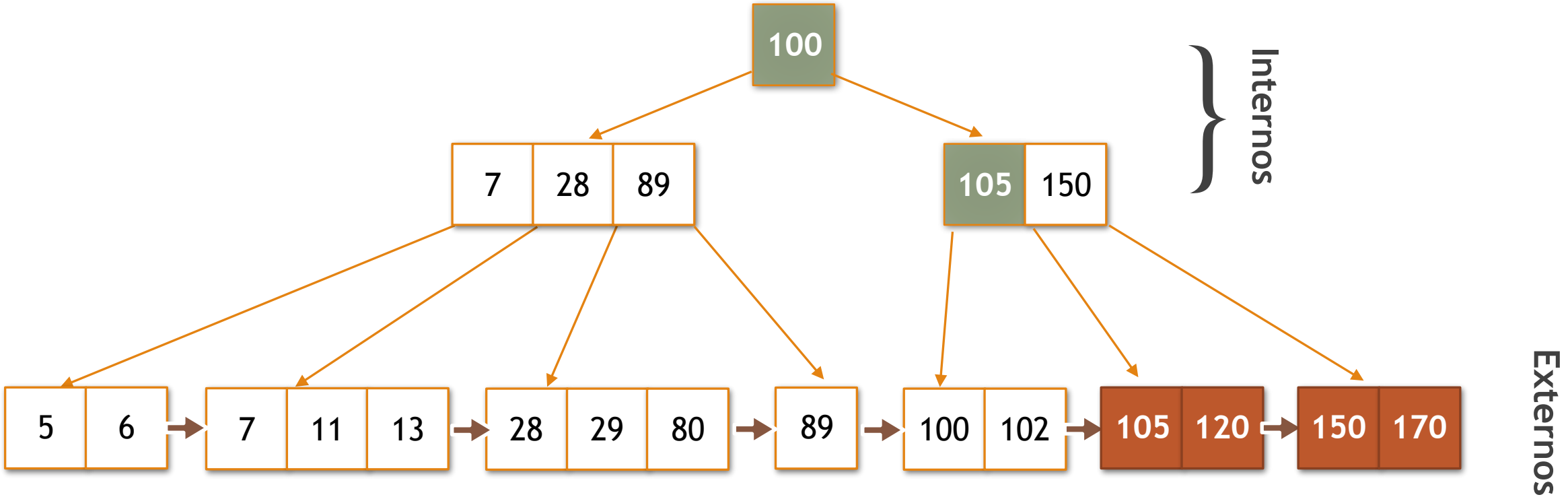
Árvore B+

busca(entre 105 e 170)



Árvore B+

busca(entre 105 e 170)



Árvore B+ na prática

Tipico fator de preenchimento: 67%

Capacidade típica:

-> Altura 3 = 312.900.721 entradas

-> Altura 2 = 2.406.104 entradas

Páginas por nível:

-> Nível 0 = 1 página = 8KB

-> Nível 1 = 134 páginas = 1MB

-> Nível 2 = 17.956 páginas = 140MB

Fonte: Andy Pavlo - Query Processing

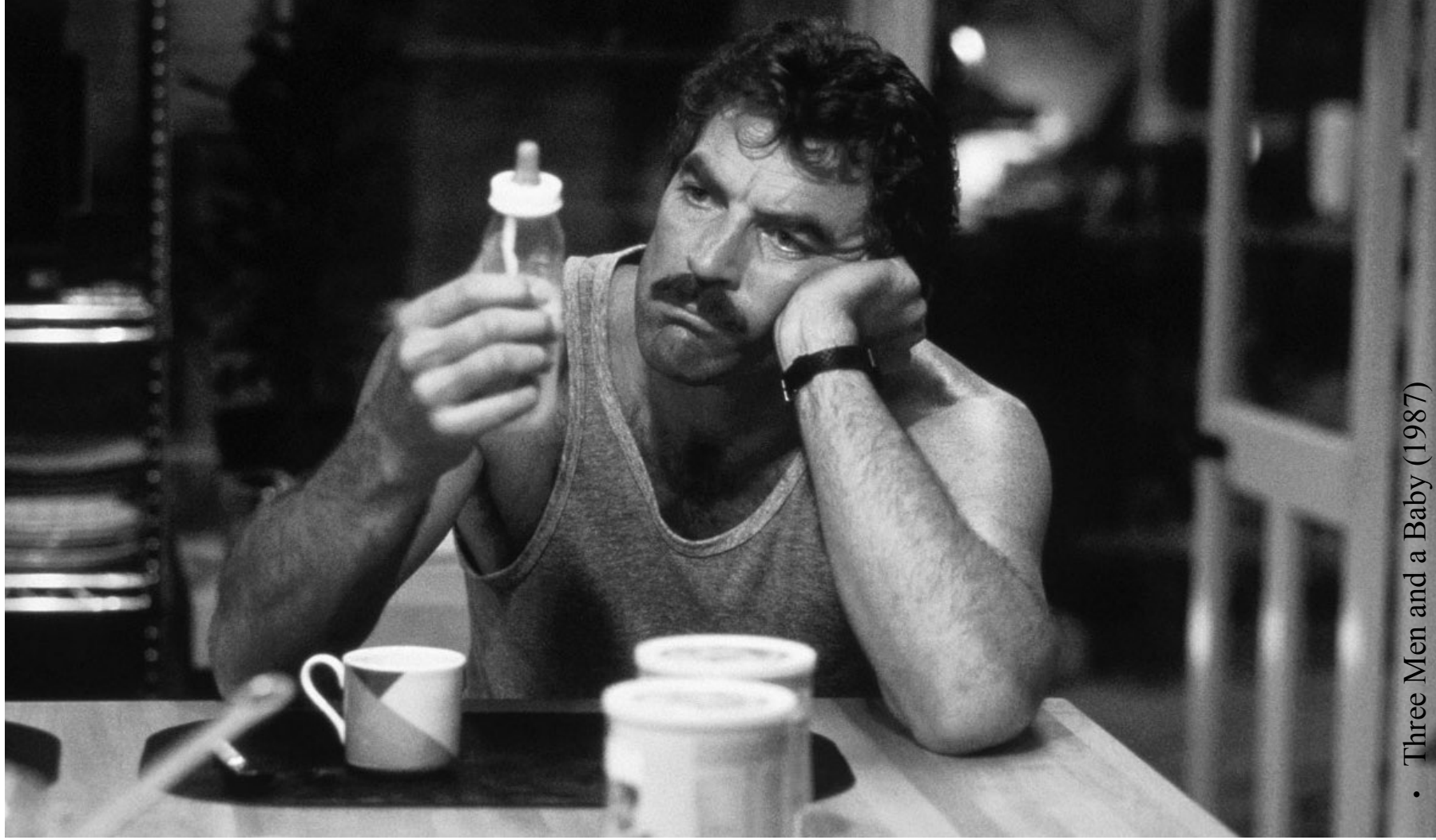
Conclusões

- Árvores B+ são sempre uma boa opção para armazenamento e busca em grandes bancos de dados

- SkipLists possuem propriedades interessantes para grande vazão de escrita de dados (e.g., data streaming), mas não são “cache-amigáveis” (e.g., dados em paginas separadas)

- Arvores binárias balanceadas: excelentes para o problema de “agendamento” de processos

Modulo: BigData



• Three Men and a Baby (1987)

Aula #3 - Estruturas de Dados

EDUARDO CUNHA DE ALMEIDA