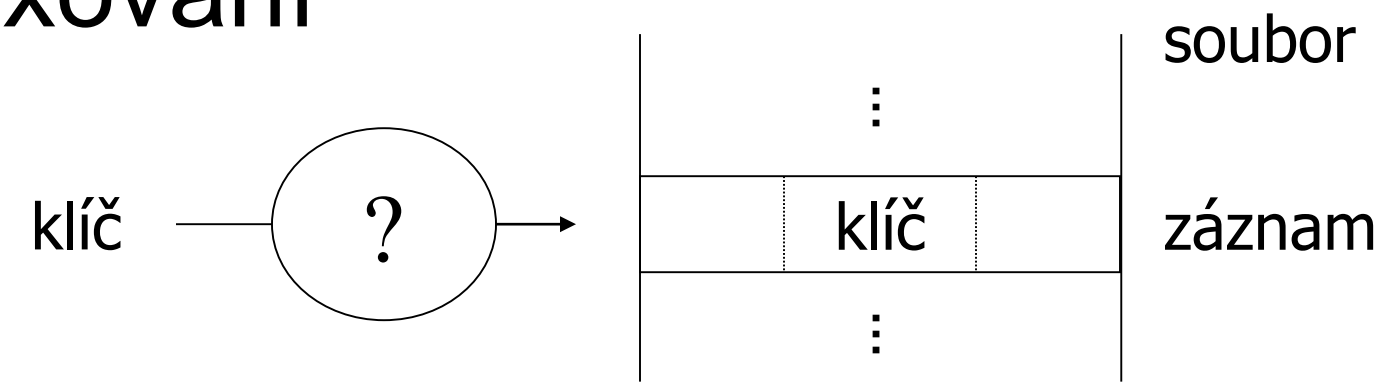
PA152: Efektivní využívání DB

4. Indexování

Vlastislav Dohnal

Indexování



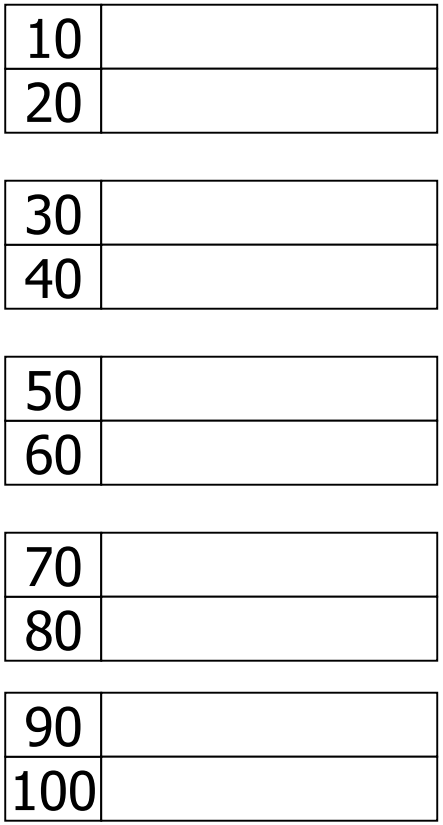
- Důvod: rychlejší přístup k záznamům
  - než sekvenční průchod souborů
- Varianty:
  - Konvenční indexy
  - B-stromy
  - Hašování

Základní pojmy

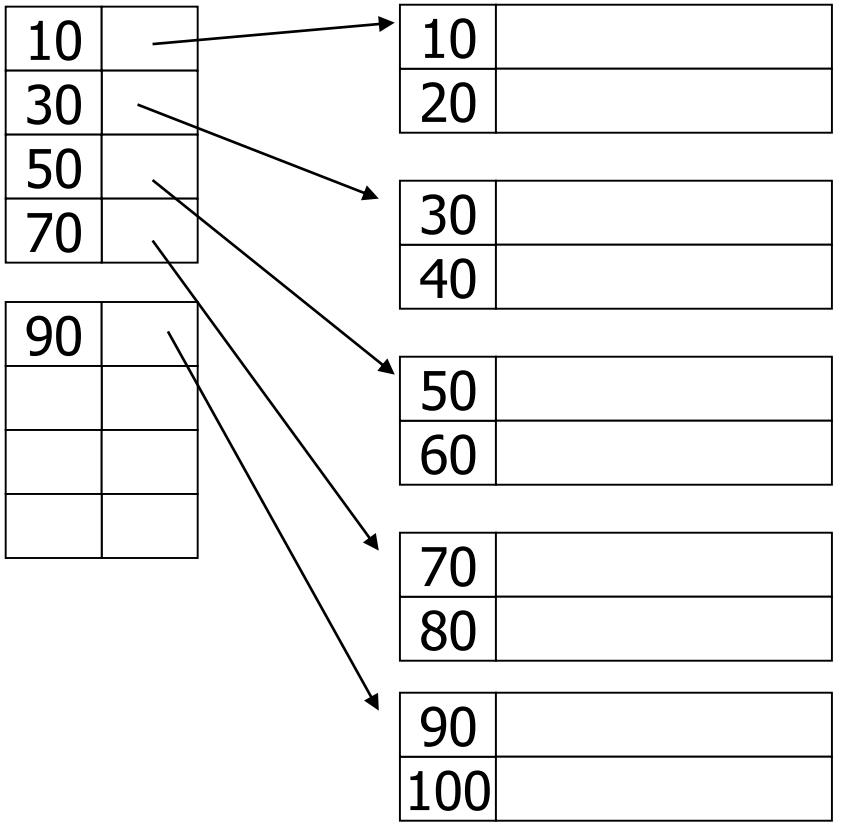
- Sekvenční soubor
  - Index-sekvenční soubor
- Vyhledávací klíč
  - Primární klíč
- Index
  - Primární index
  - Sekundární index
  - Hustý index
  - Řídký index
  - Víceúrovňový index

Soubor

Sekvenční soubor

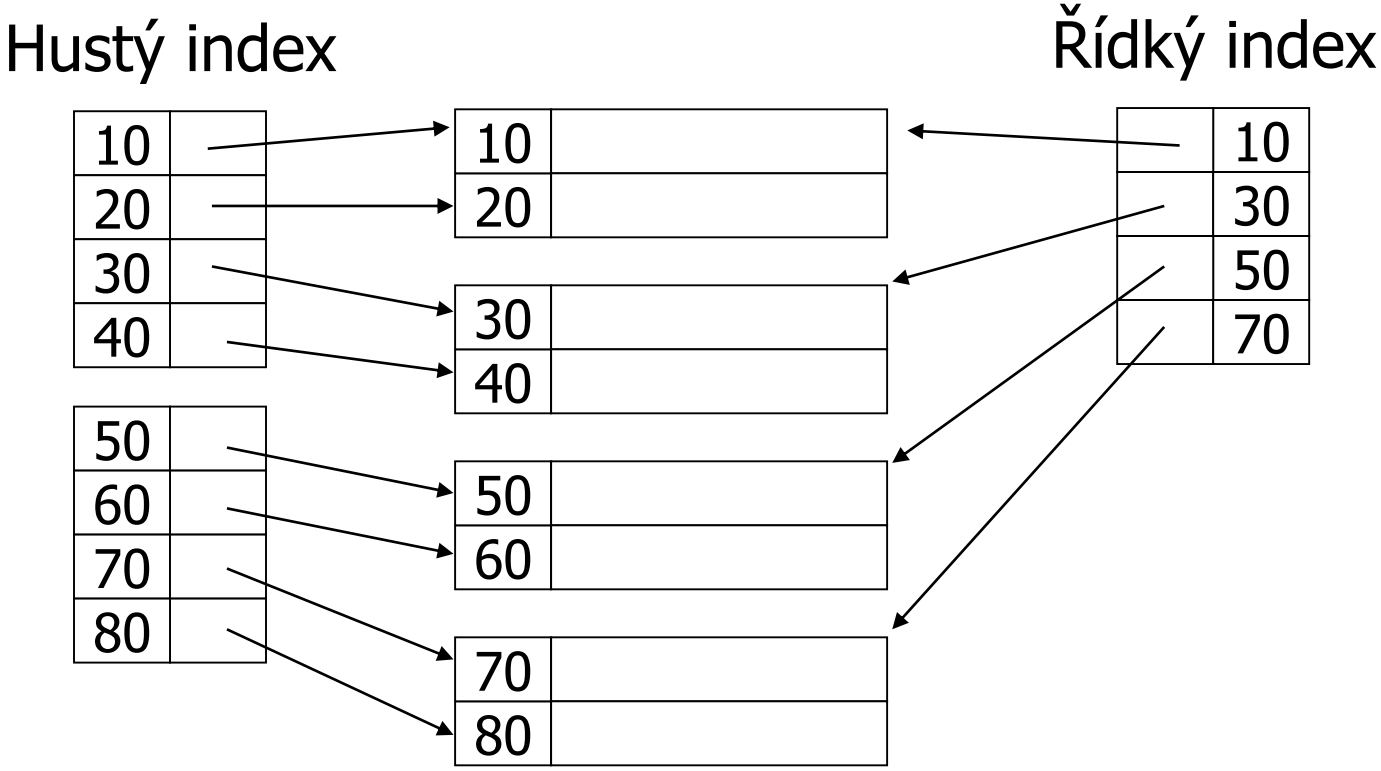


Index-sekvenční soubor



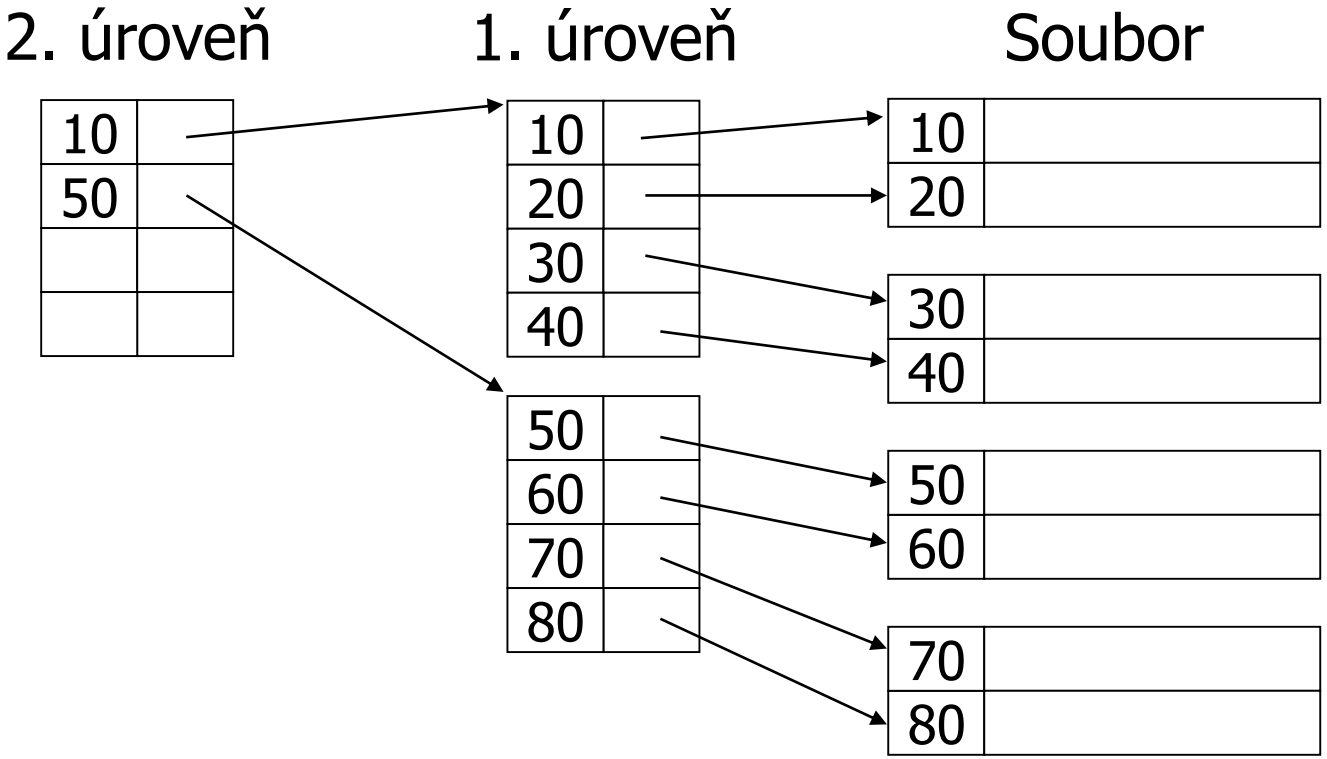
Index

- Položka indexu:
  - <klíč, odkaz na záznam/blok>



Index

- Víceúrovňový index



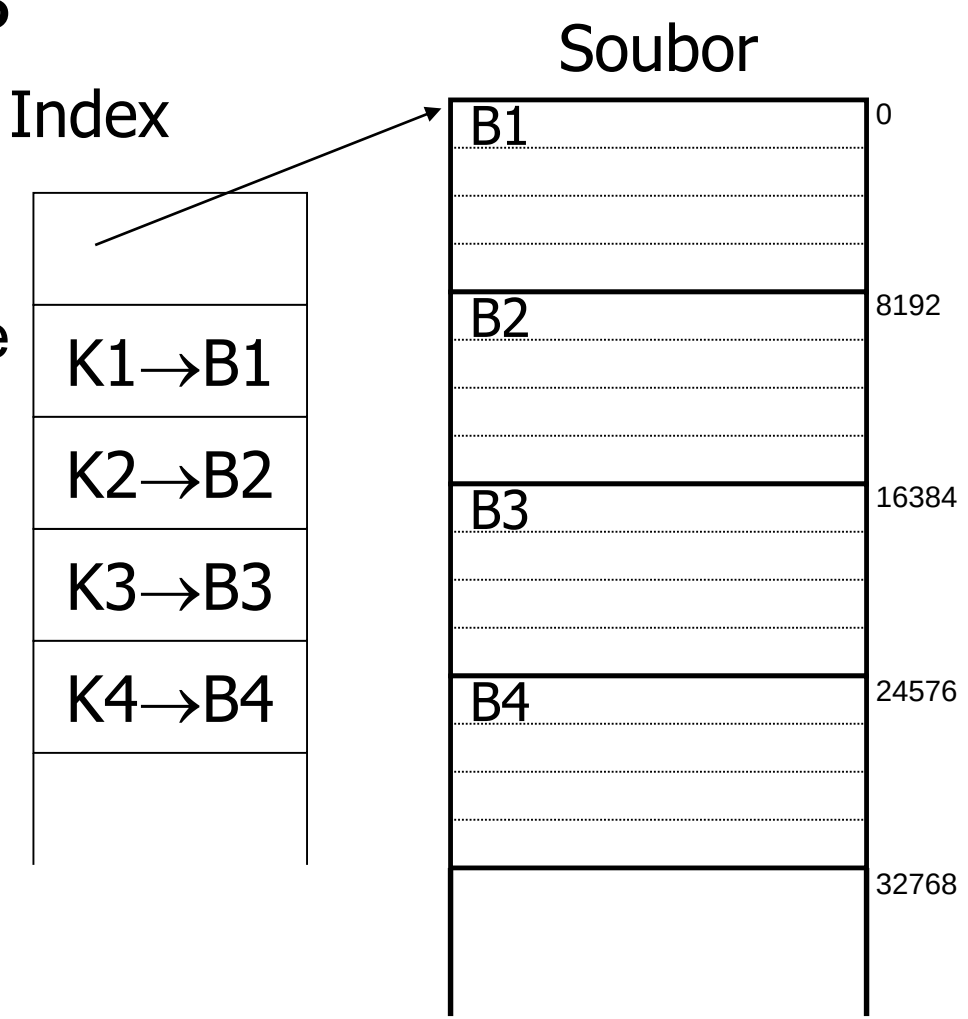
- Lze mít indexy vyšších úrovní husté?

Index a ukazatele

- Ukazatele v indexech
  - Ukazatel na záznam
    - Adresa bloku + číslo (offset) záznamu
  - Ukazatel na blok
    - Adresa bloku
      - identifikace souboru + offset v souboru
      - lépe: identifikace souboru + číslo bloku
  - Soubor je spojitý a uspořádaný
    - Ukazatele na bloky se nemusí ukládat
      - tzv. implicitní odkazy – lze je „spočítat“
      - např. číslo bloku lze určit z pořadí položky v indexu

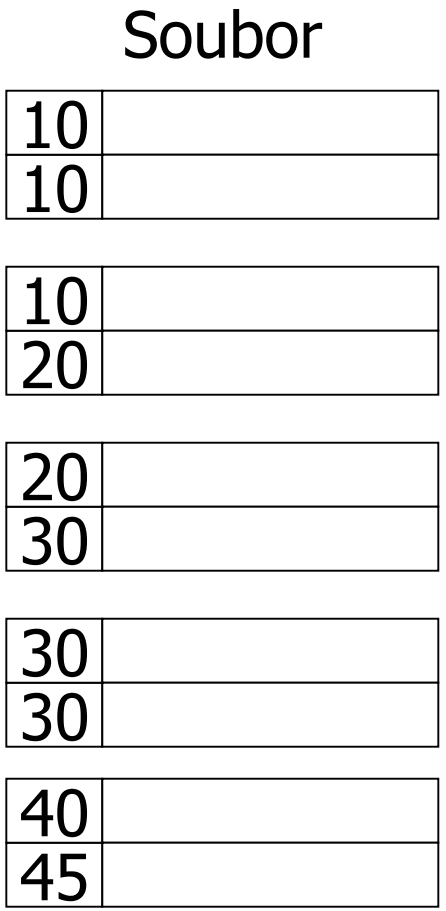
Implicitní ukazatele na blok

- Blok o velikosti 8KiB
- Hledáme „K3“
  - Prohledání indexu
    - K3 je ve třetí položce
      - → třetí blok (B3)
  - Offset v souboru
    - (3-1)·8192=16 384



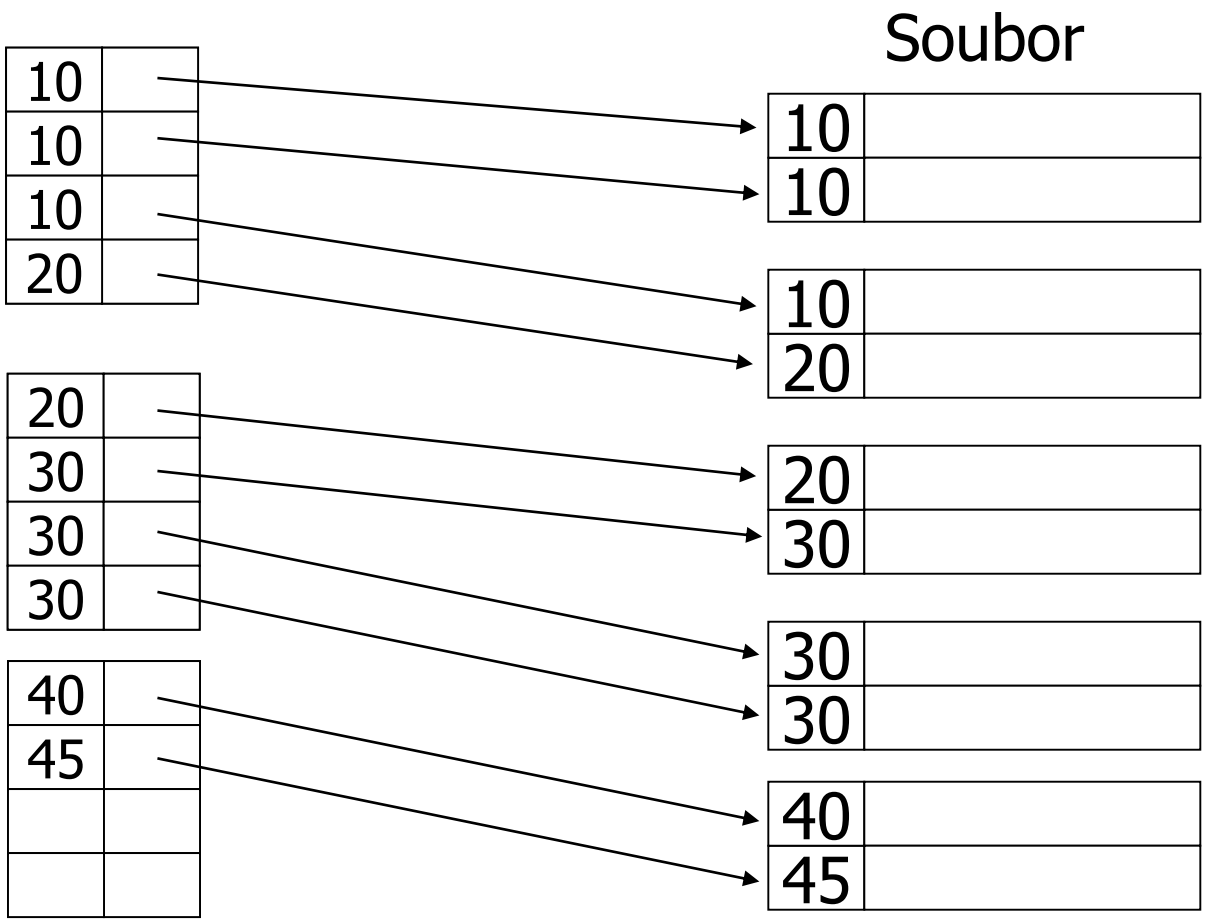
Duplicitní klíče

- Vytvoření indexu
  - hustý index?
  - řídký index?



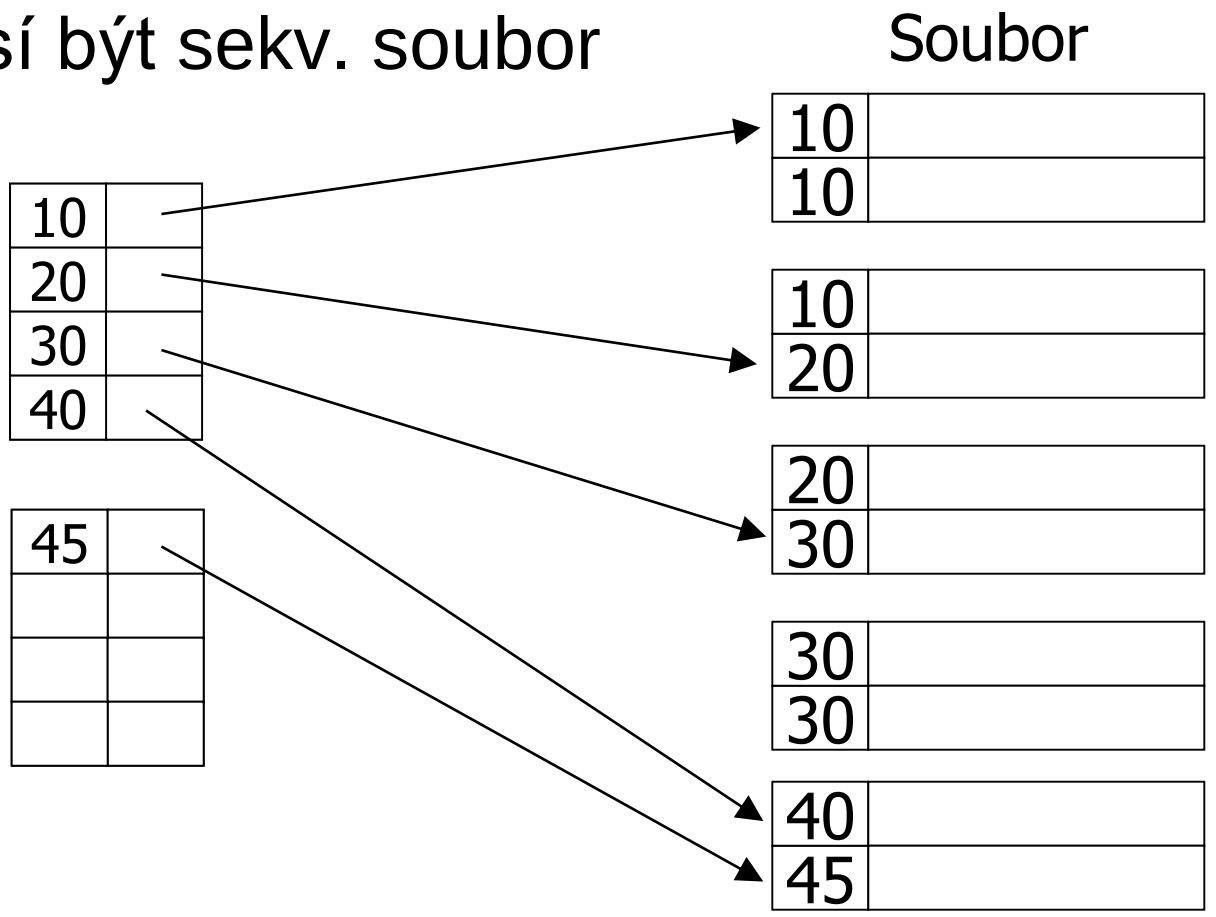
Duplicitní klíče: hustý index

- Duplicity v primárním indexu



Duplicitní klíče: hustý index

- Klíče v primárním indexu jsou unikátní
  - vždy musí být sekv. soubor



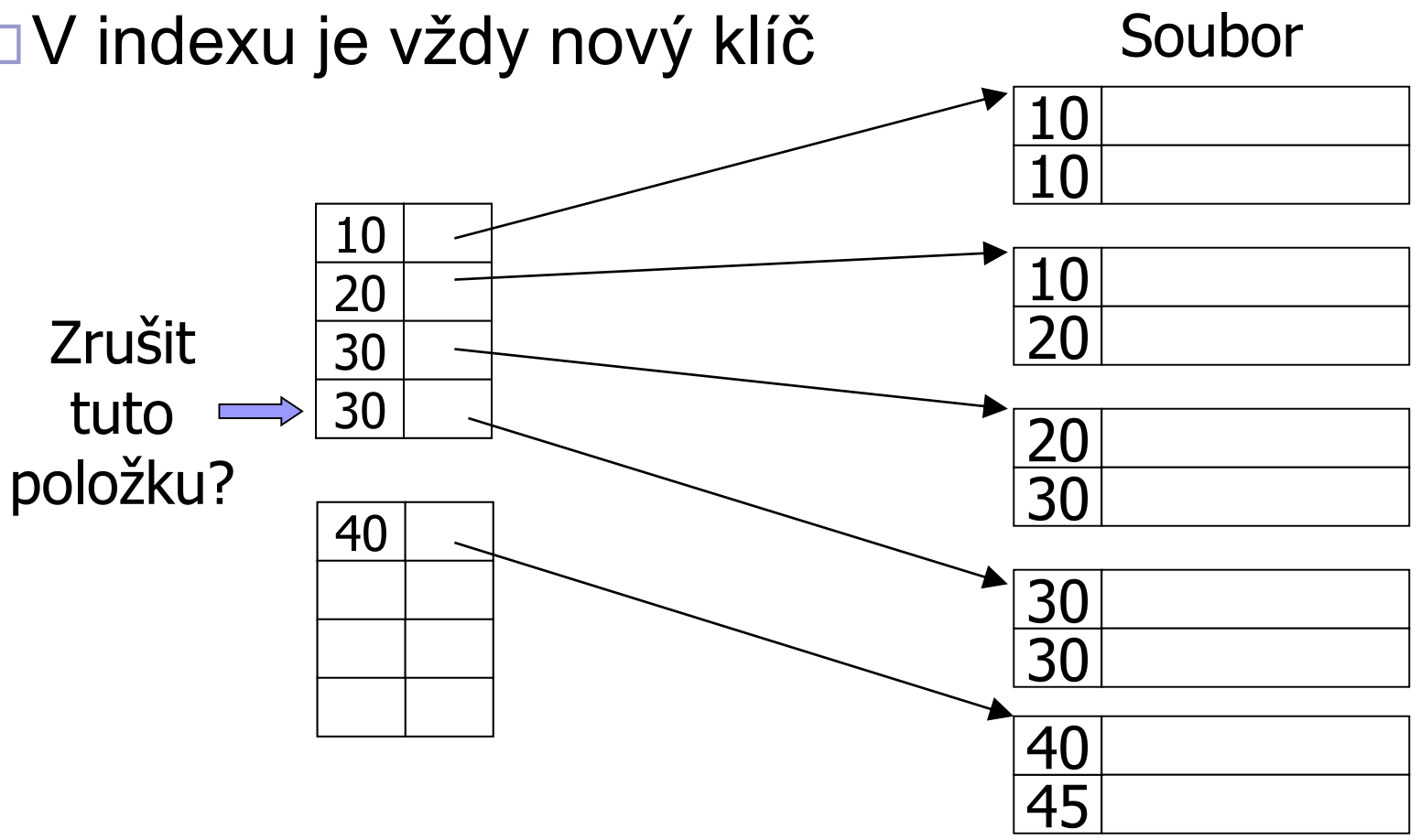
Duplicitní klíče: řídký index

- Odkazy na bloky
  - Duplicitní klíče lze eliminovat

- Vyhledávání záznamů!!!
  - Najdi hodnotu „20“

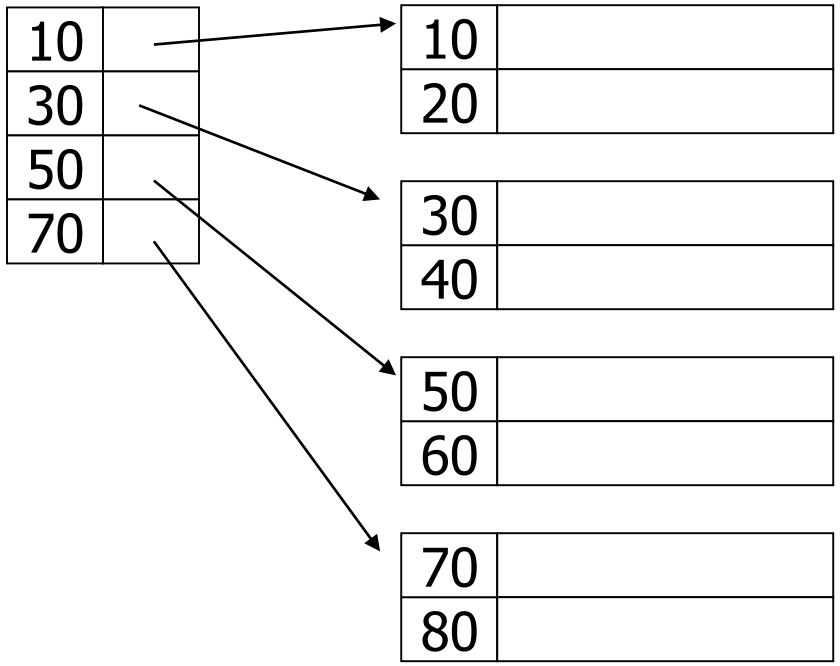
Duplicitní klíče: řídký index

- Odkaz na blok, ale:
  - V indexu je vždy nový klíč



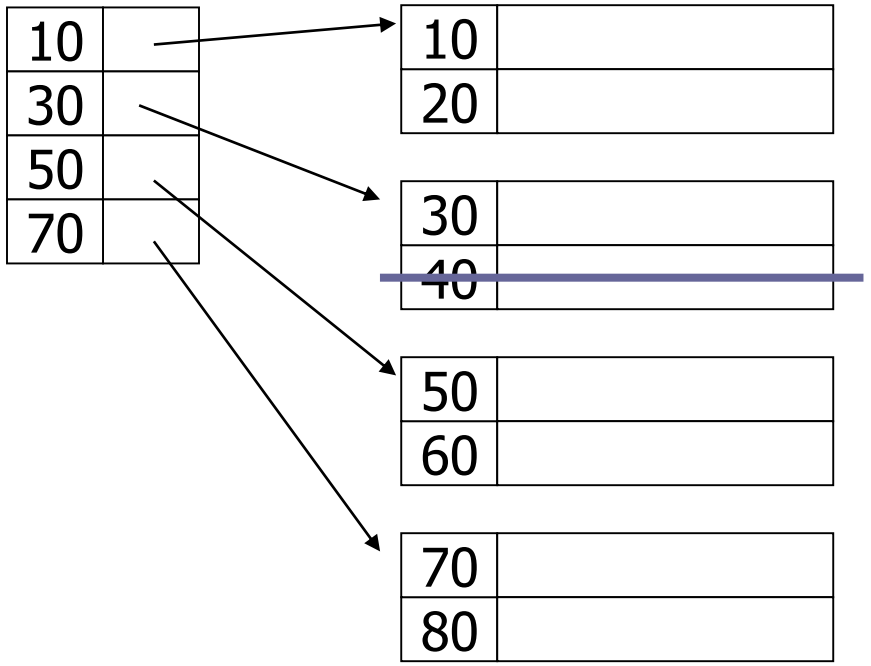
Mazání v indexech

- Řídký index
  - Smazání záznamu s klíčem 40



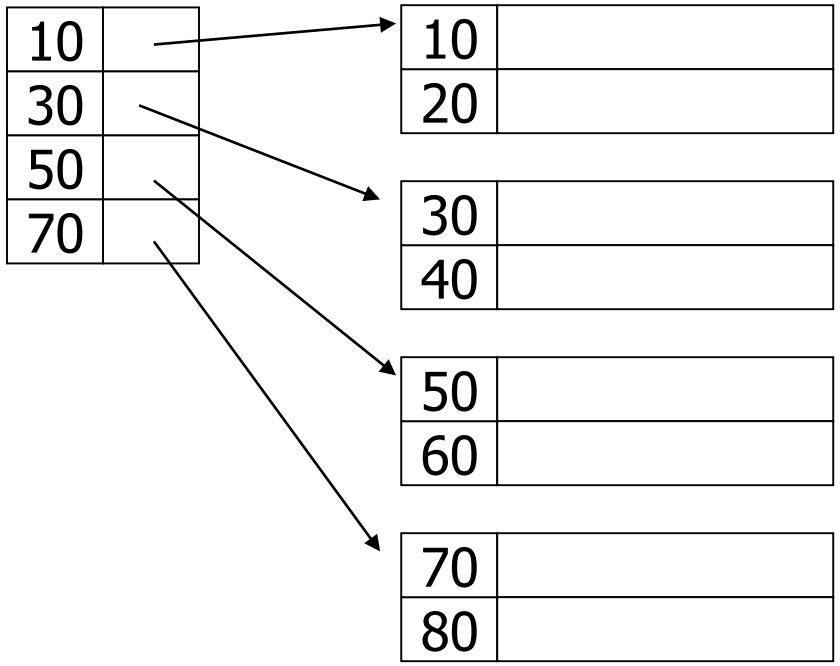
Mazání v indexech: výsledek

- Řídký index
  - Po smazání záznamu s klíčem 40



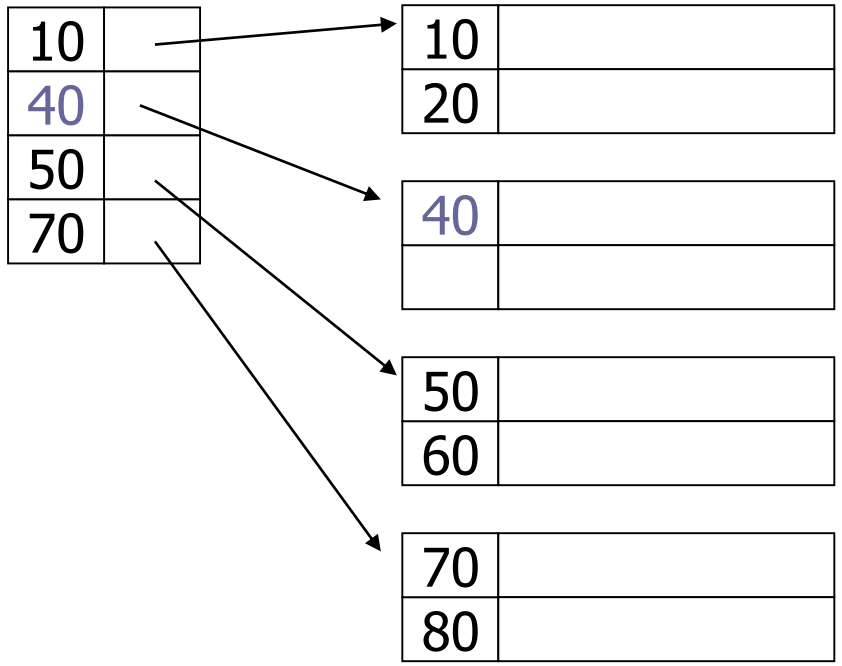
Mazání v indexech

- Řídký index
  - Smazání záznamu s klíčem 30



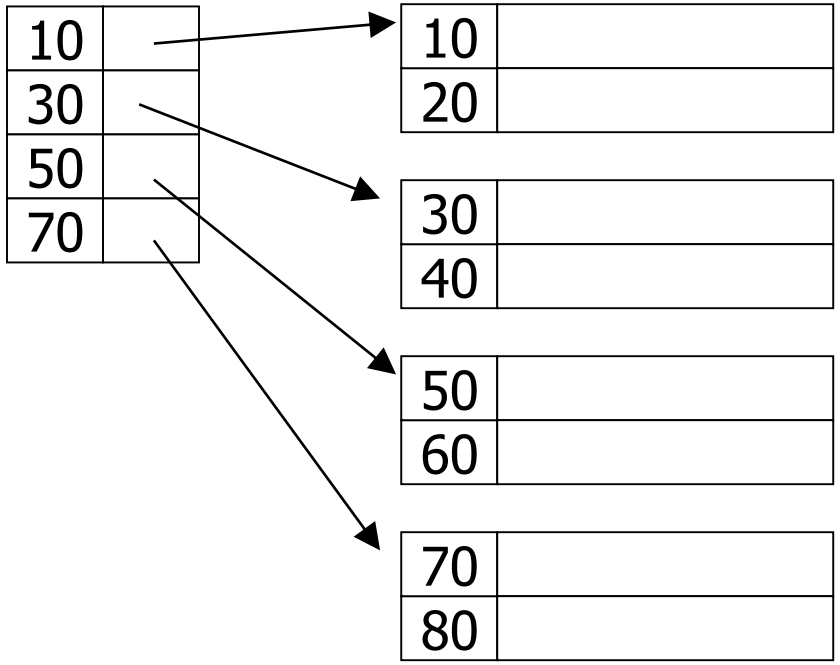
Mazání v indexech: výsledek

- Řídký index
  - Po smazání záznamu s klíčem 30
    - Aktualizace indexu



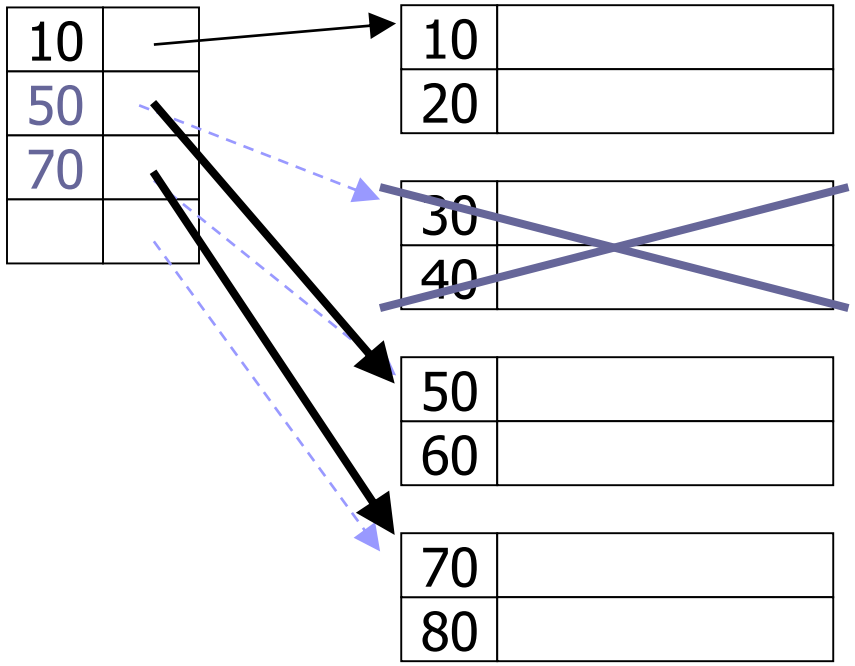
Mazání v indexech

- Řídký index
  - Smazání záznamů s 30 a 40



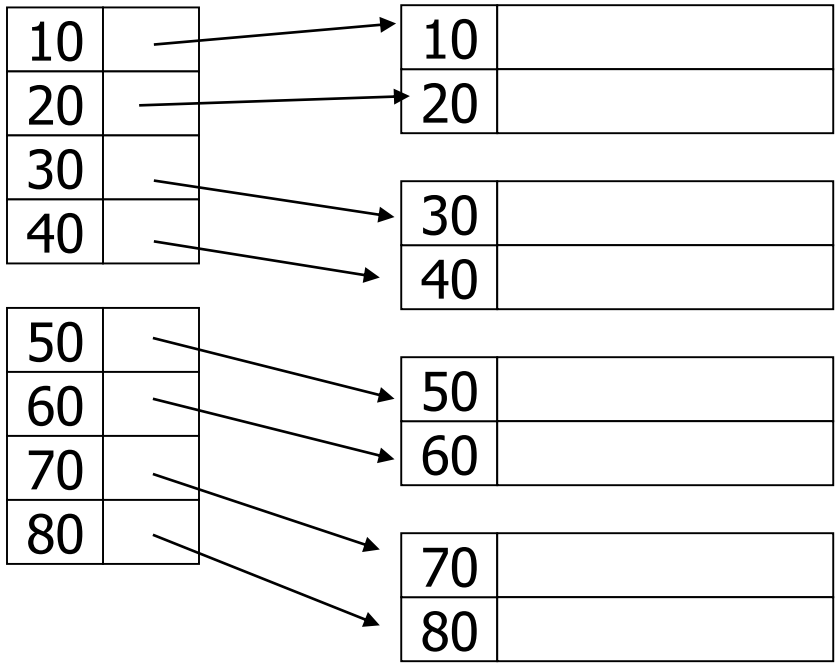
Mazání v indexech: výsledek

- Řídký index
  - Po smazání záznamů s 30 a 40
    - Aktualizace indexu



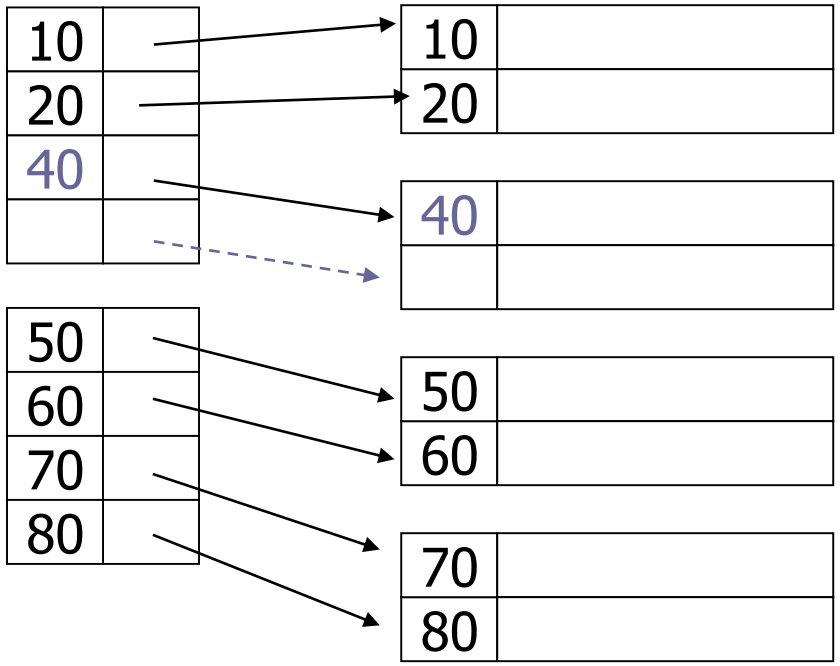
Mazání v indexech

- Hustý index – vždy aktualizace indexu
  - Smazání záznamu s klíčem 30



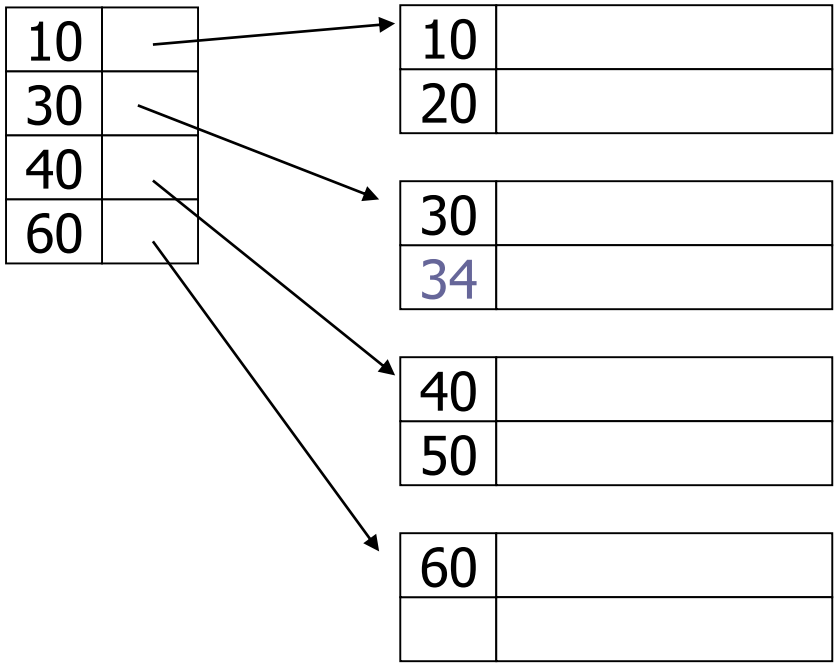
Mazání v indexech: výsledek

- Hustý index
  - Po smazání záznamu s klíčem 30



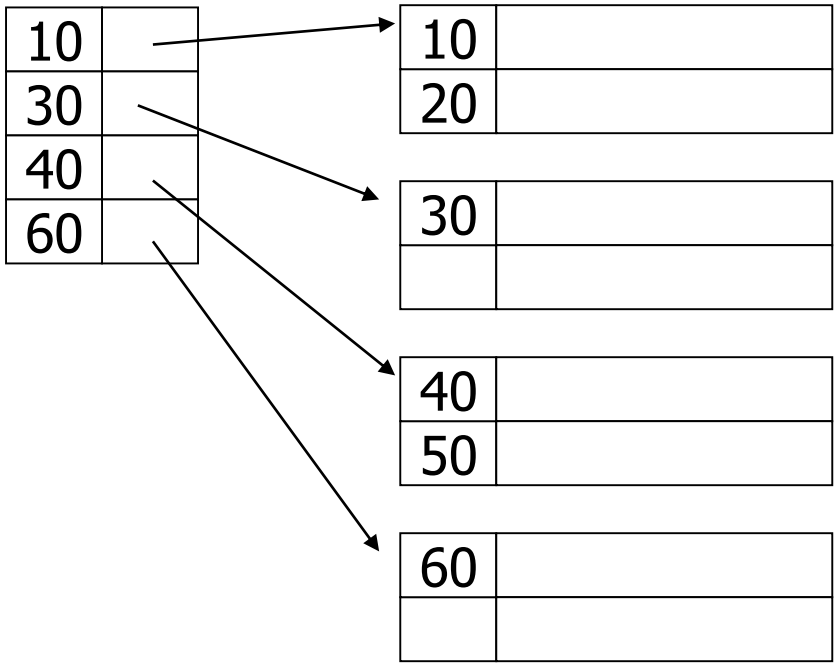
Vkládání v indexech

- Řídký index
  - Vložení záznamu s klíčem 34
    - Volné místo
      → bez reorganizace



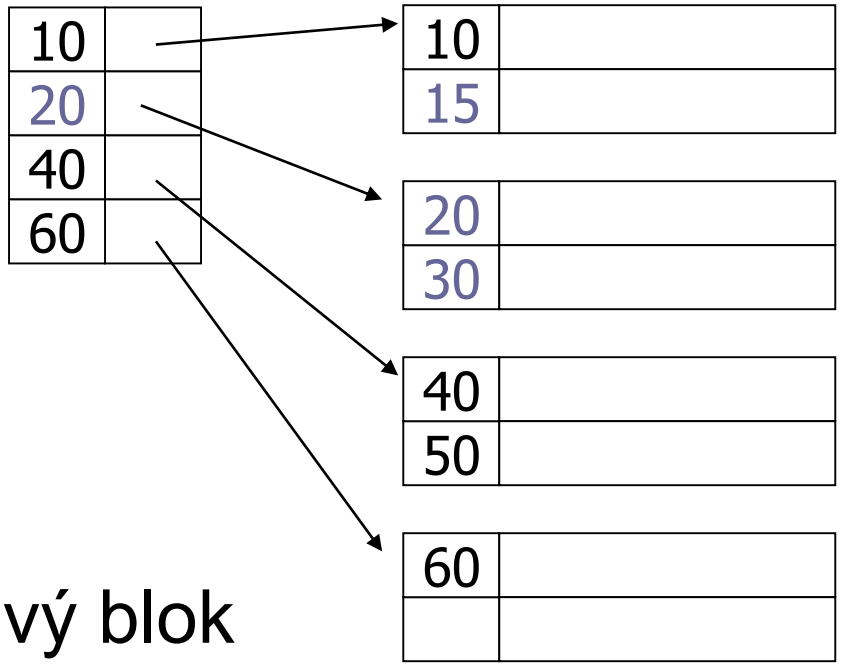
Vkládání v indexech

- Řídký index
  - Vložení záznamu s klíčem 15
    - Volné místo není → reorganizace



Vkládání v indexech

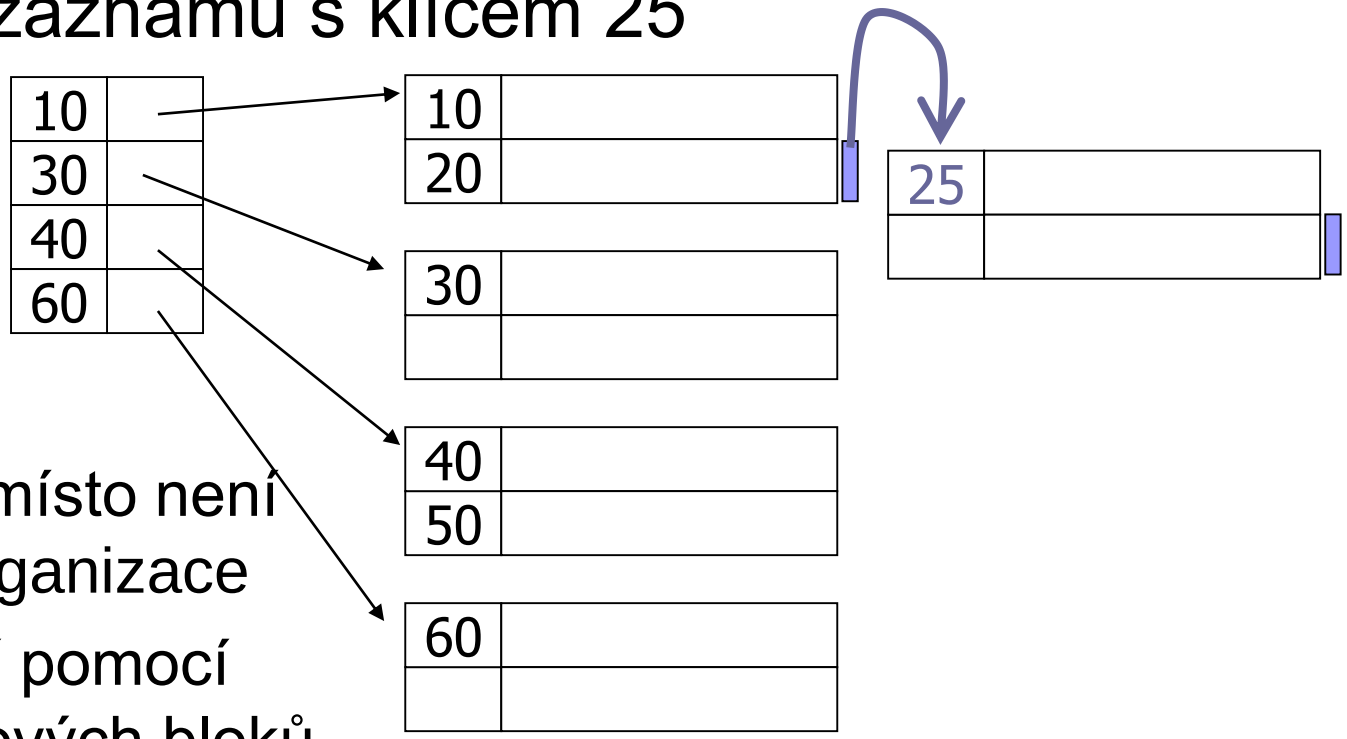
- Řídký index
  - Vložení záznamu s klíčem 15
    - Volné místo není → reorganizace
    - Řešení pomocí přesunu do sousedního bloku



  - Alternativa: vytvoř nový blok
    - U implicitních indexů pozor na případné porušení spojitosti a uspořádanosti bloků souboru

Vkládání v indexech

- Řídký index
  - Vložení záznamu s klíčem 25



- Volné místo není → reorganizace
- Řešení pomocí přetokových bloků
- Reorganizace je provedena později

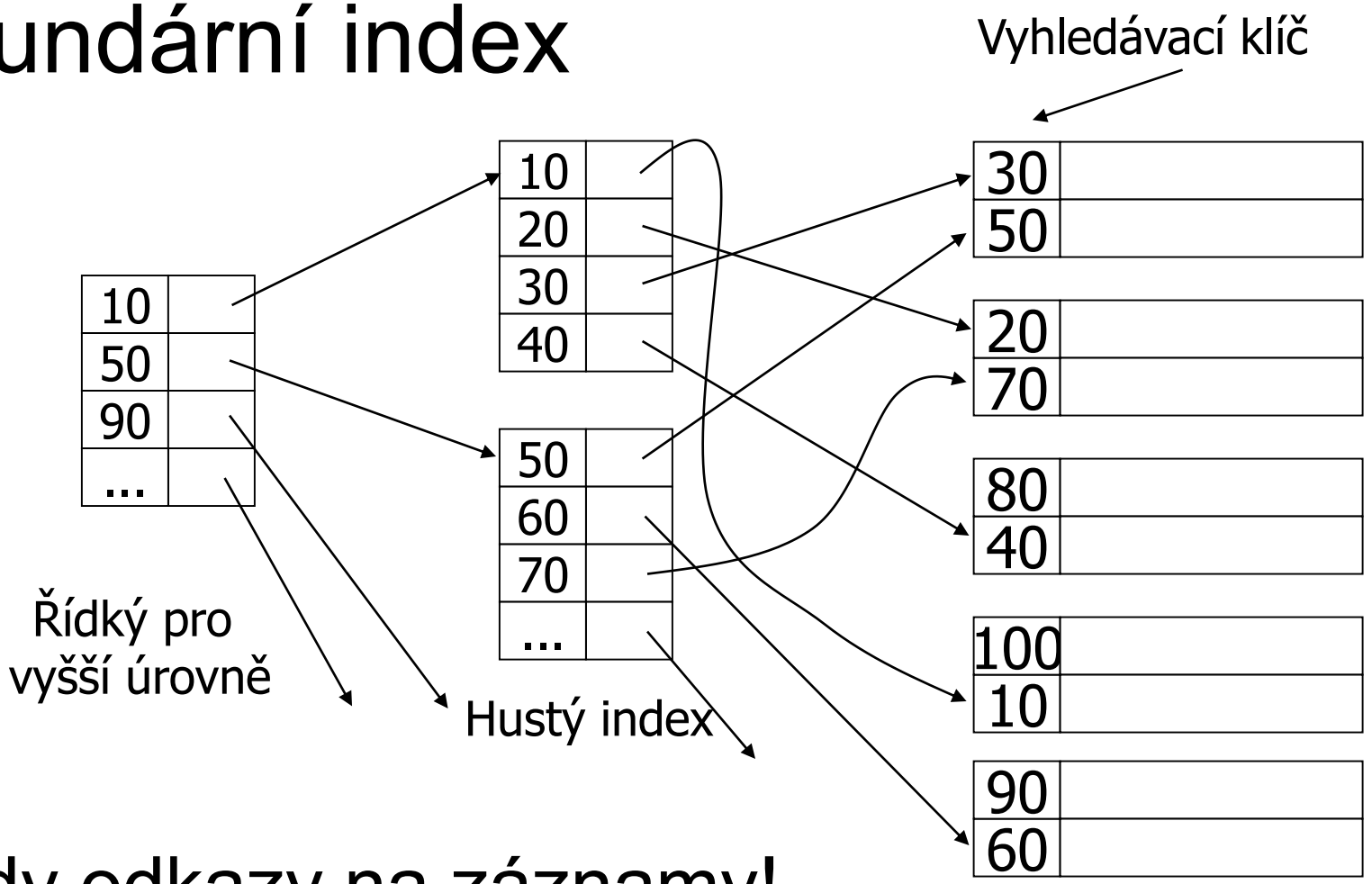
Vkládání v indexech

- Hustý index
  - Vložení záznamu
    - Vždy i do indexu
    - V souboru stejné problémy jako u řídkého indexu

Sekundární index

- Soubor je uspořádaný podle jiného klíče
  - tj. vybudovaný na jiném klíči než je primární soubor
- Volba typu:
  - Hustý nebo řídký?

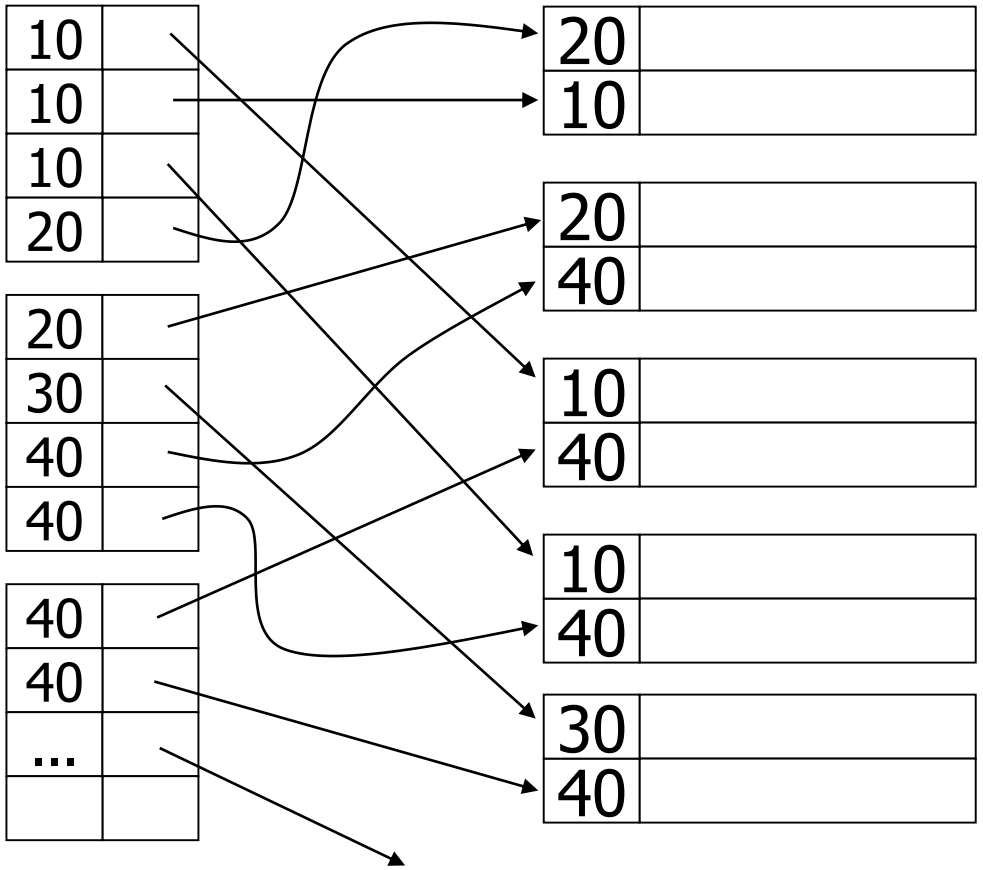
Sekundární index



- Vždy odkazy na záznamy!

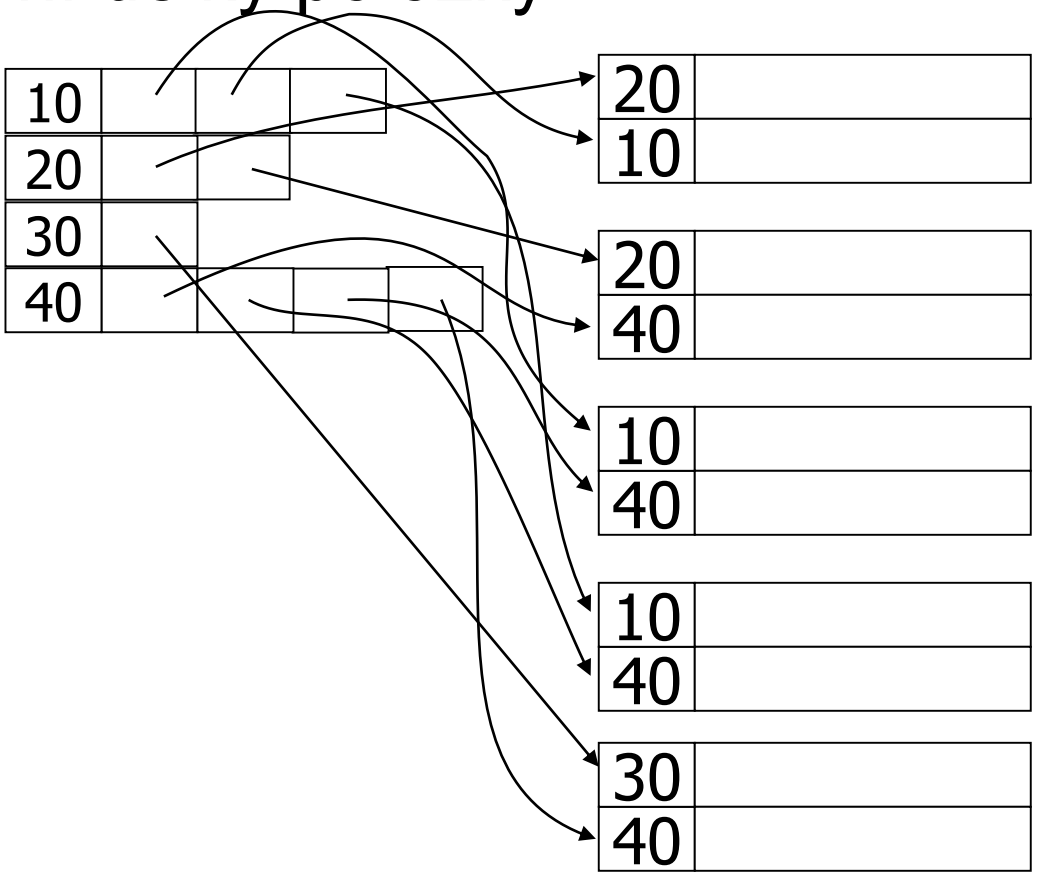
Sekundární index: duplicitní klíče

- Replikace v indexu
  - Zvýšení
    - Prostorových nároků
    - Přístupového času (access time)



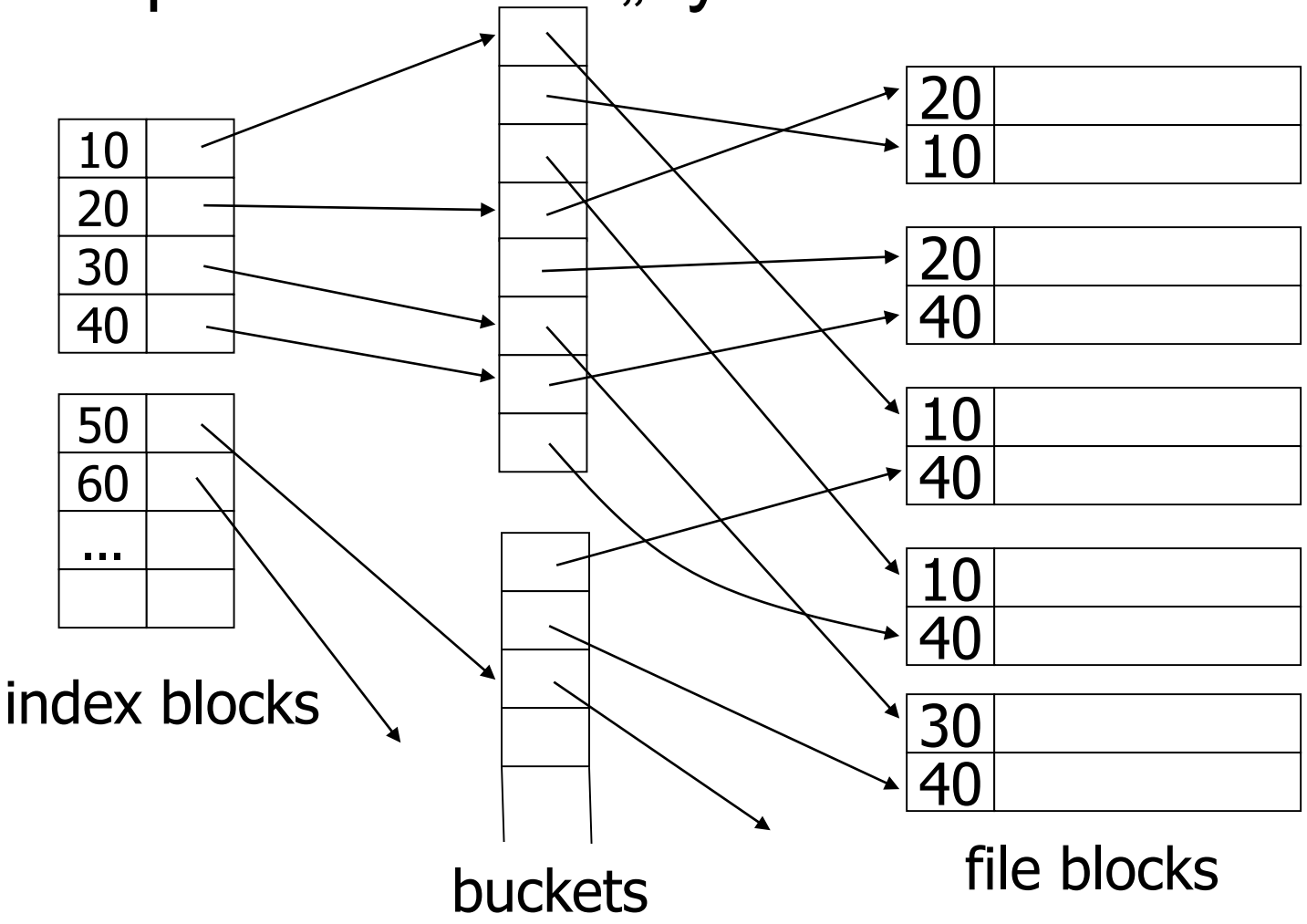
Sekundární index: duplicitní klíče

- Indexová položka má seznam ukazatelů
  - Problém variabilní délky položky



Sekundární index: duplicitní klíče

- Variabilita přesunuta do „kyblíků“



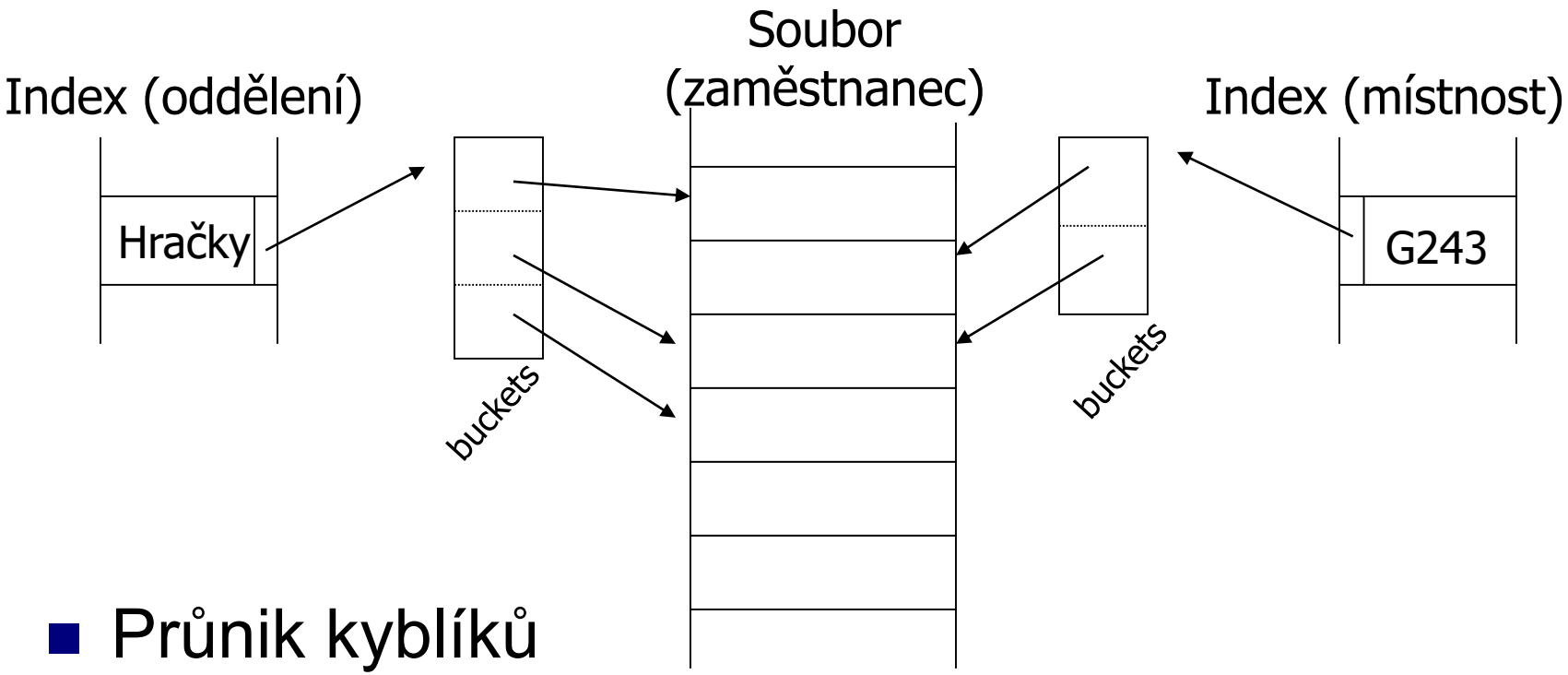
Sekundární index: duplicitní klíče

- Výhoda seznam odkazů na záznam při dotazování
  - Spojení různých podmínek bez čtení záznamů

- Příklad:
  - Relace
    - zaměstnanec(jméno, oddělení, místnost)
  - Indexy:
    - Jméno – primární index
    - Oddělení – sekundární index
    - Místnost – sekundární index

Sekundární index: duplicitní klíče

- Dotaz: zaměstnanci hraček v místnosti G243



- Průnik kyblíků
  - Máme seznam ukazatelů na správné zaměstnance
  - Technika používaná v *text information retrieval*

Příklad: Text Information Retrieval

- „Full-text“ index nad dokumenty
- Textové dokumenty rozděleny na slova

Document 1

Caesar and Brutus are ambitious.

1. Split to words
2. Stemming
3. Ignore ones in stop-list

Word List for Doc1

- *Caesar*
- *Brutus*
- *ambitious*

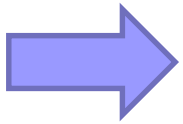
- Vytvoř invertovaný soubor
  - pro všechny dokumenty

Příklad: Text Information Retrieval

- Invertovaný soubor

Relační pohled

Term	docID
ambitious	1
brutus	1
brutus	3
capitol	2
caesar	1
caesar	2



Invertovaný soubor

Term	Posting list of docIDs
ambitious	1
brutus	1, 3
capitol	1
caesar	1, 2

- Najdi dokumenty obsahující *Brutus* i *Caesar*
  - Načti *posting lists* pro Brutus a Caesar
  - Vytvoř jejich průnik

Konvenční indexy: shrnutí

- Základní dělení
  - Řídké vs. Husté
- Vkládání / mazání
  - Duplicitní klíče
    - Zejména u sekundárních indexů
- Výhody
  - Jednoduché
  - Index je sekvenční soubor → vhodné pro „full scan“
- Nevýhody
  - Aktualizace drahá
  - Ztráta „sekvenčnosti“ a vyváženosti
    - kvůli přetokovým oblastem

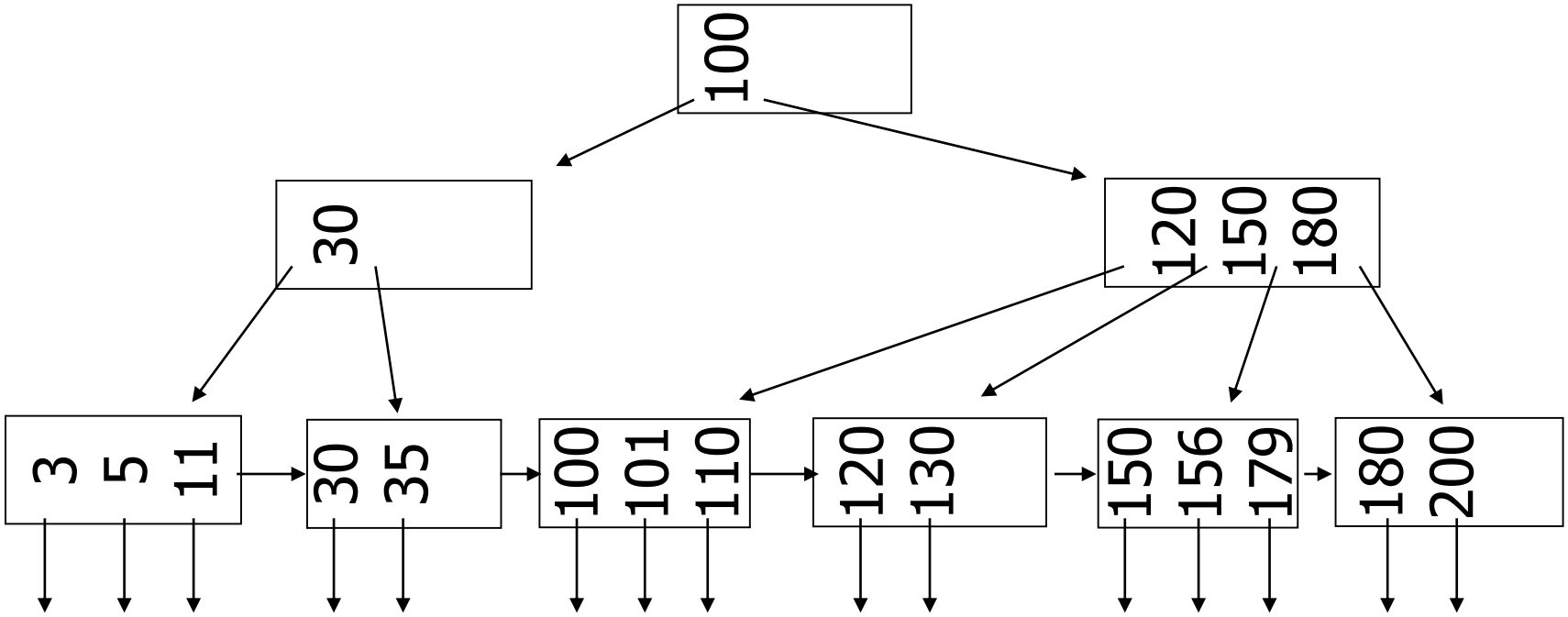
B-stromy

- Jiný typ indexu
  - Sekvenční uspořádání není nutné
  - Garance I/O pro přístup (vyváženost)
- Více variant
  - B-strom, B^+-strom, B^*-strom, …
    - Obvykle se při vyslovení „*B-strom*“ myslí „ *B^+-strom*“!
- Původ
  - Rudolf Bayer and Ed McCreight invented the B-tree while working at Boeing Research Labs in 1971 (Bayer & McCreight 1972)
    - They did not explain what, if anything, the B stands for.
    - Douglas Comer explains:
      - The origin of "B-tree" has never been explained by the authors. As we shall see, "balanced," "broad," or "bushy" might apply. Others suggest that the "B" stands for Boeing. Because of his contributions, however, it seems appropriate to think of B-trees as "Bayer"-trees.

* Source: Wikipedia

B$^+$-strom

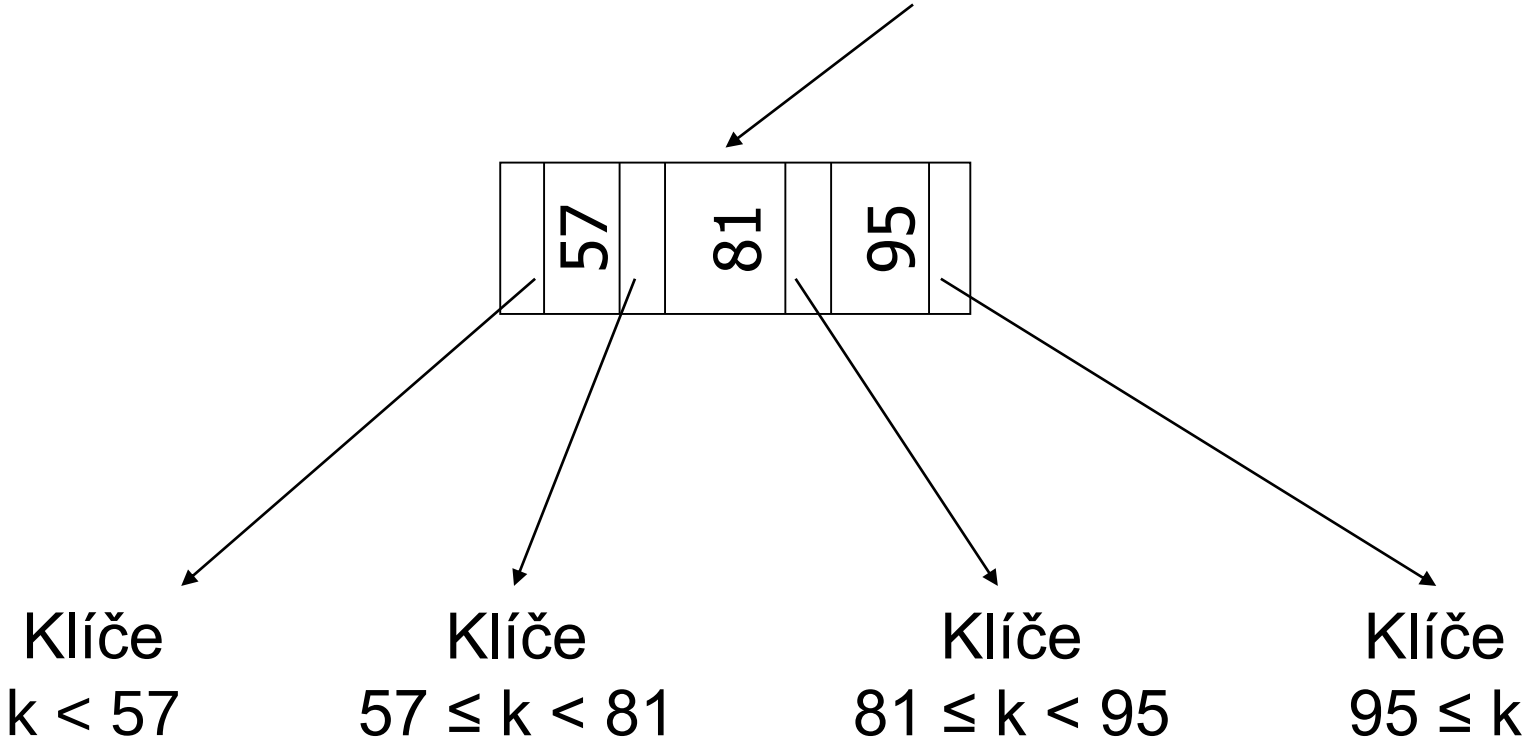
- Příklad *n=4*



… odkazy na záznamy v souboru …

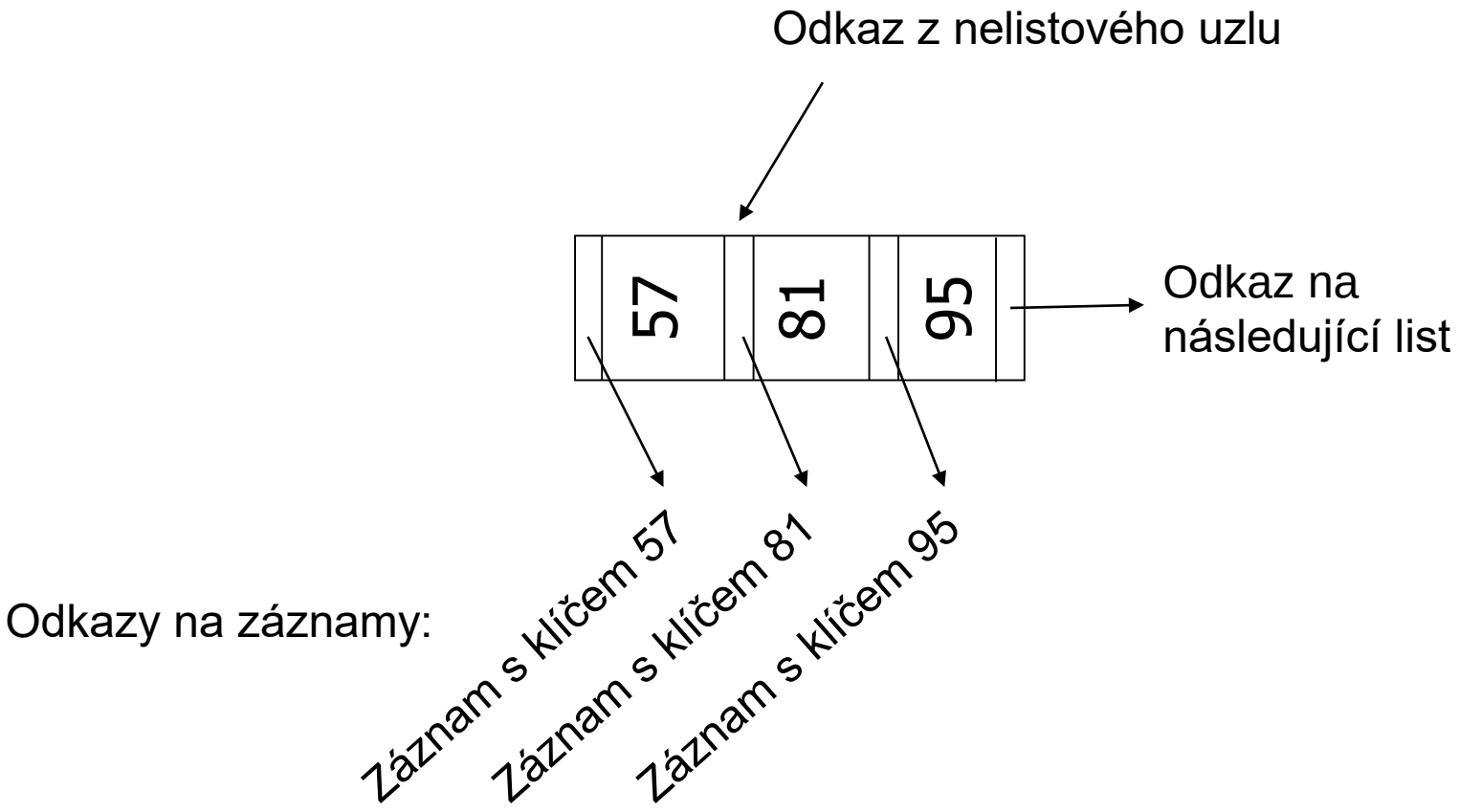
B$^+$-strom

- Vnitřní uzel pro *n=4*



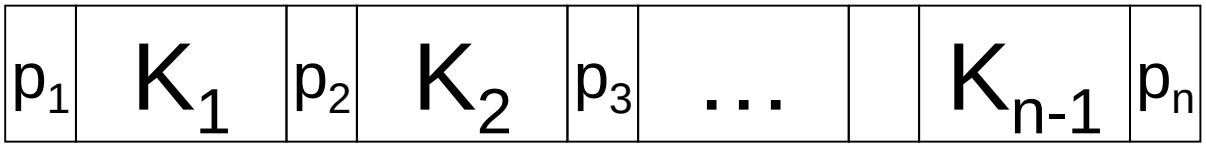
B$^+$-strom

- Listový uzel pro *n=4*



B^+-strom

- Parametr *n* (arita stromu) ovlivňuje:
  - Tvar uzlu:



  - Minimální naplnění
  - Listový uzel
    - Vždy na stejné úrovni
    - p_i odkazuje na záznam s klíčem K_i (data)
    - p_n odkazuje na další list (zřetězení listů)
  - Nelistový uzel
    - p_i odkazuje na uzel, kde jsou klíče K: $K_{i-1} \leq K < K_i$

B^+-strom

- Podmínky naplnění

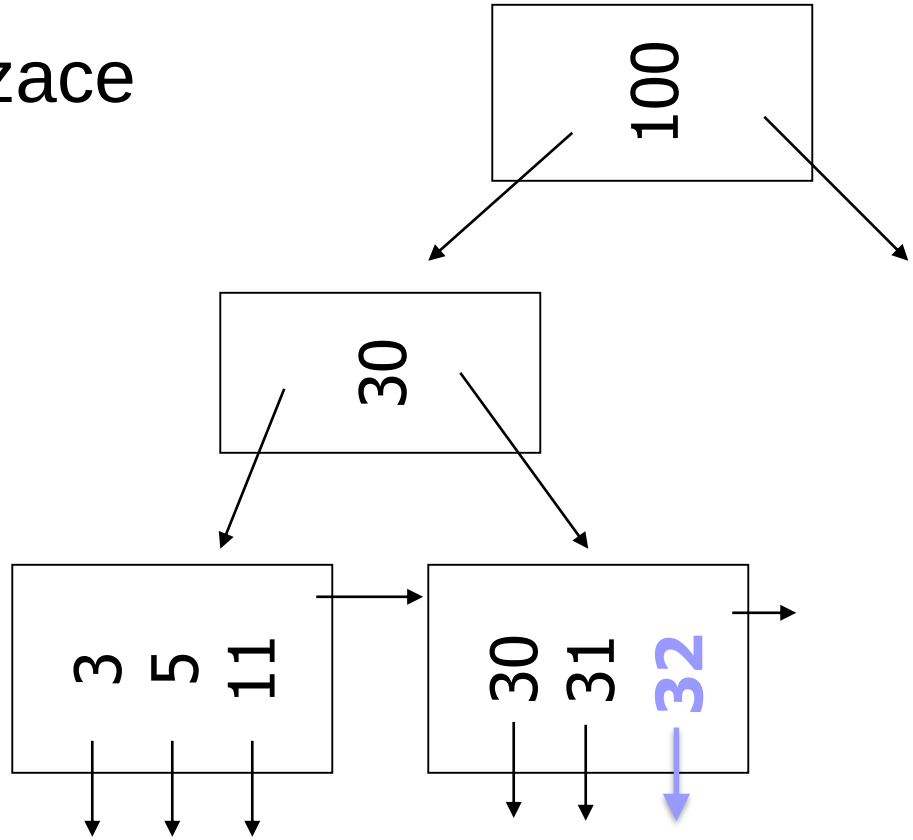
	Max ukazatelů	Min ukazatelů	Max klíčů	Min klíčů
Vnitřní uzel (ne kořen)	n (potomků)	$\lceil n/2 \rceil$ (potomků)	n-1	$\lceil n/2 \rceil$-1
Vnitřní uzel (kořen)	n (potomků)	2 (potomků)	n-1	1
List (ne kořen)	n-1 (záznamů)	$\lceil (n-1)/2 \rceil$ (záznamů)	n-1 (záznamů)	$\lceil (n-1)/2 \rceil$ (záznamů)
List (kořen)	n-1 (záznamů)	0 (záznamů)	n-1 (záznamů)	0 (záznamů)

B^+-strom: vkládání

- Princip: Růst od listu ke kořenu
- Postup: Nalézt listový uzel a vložit klíč
  - Včetně odkazu na záznam
  - Popř. aktualizovat rodiče
- Případy:
  a) Bez reorganizace (snadné)
     - V listu je volné místo
  b) Štěpení listu
  c) Štěpení vnitřního uzlu
  d) Nový kořen

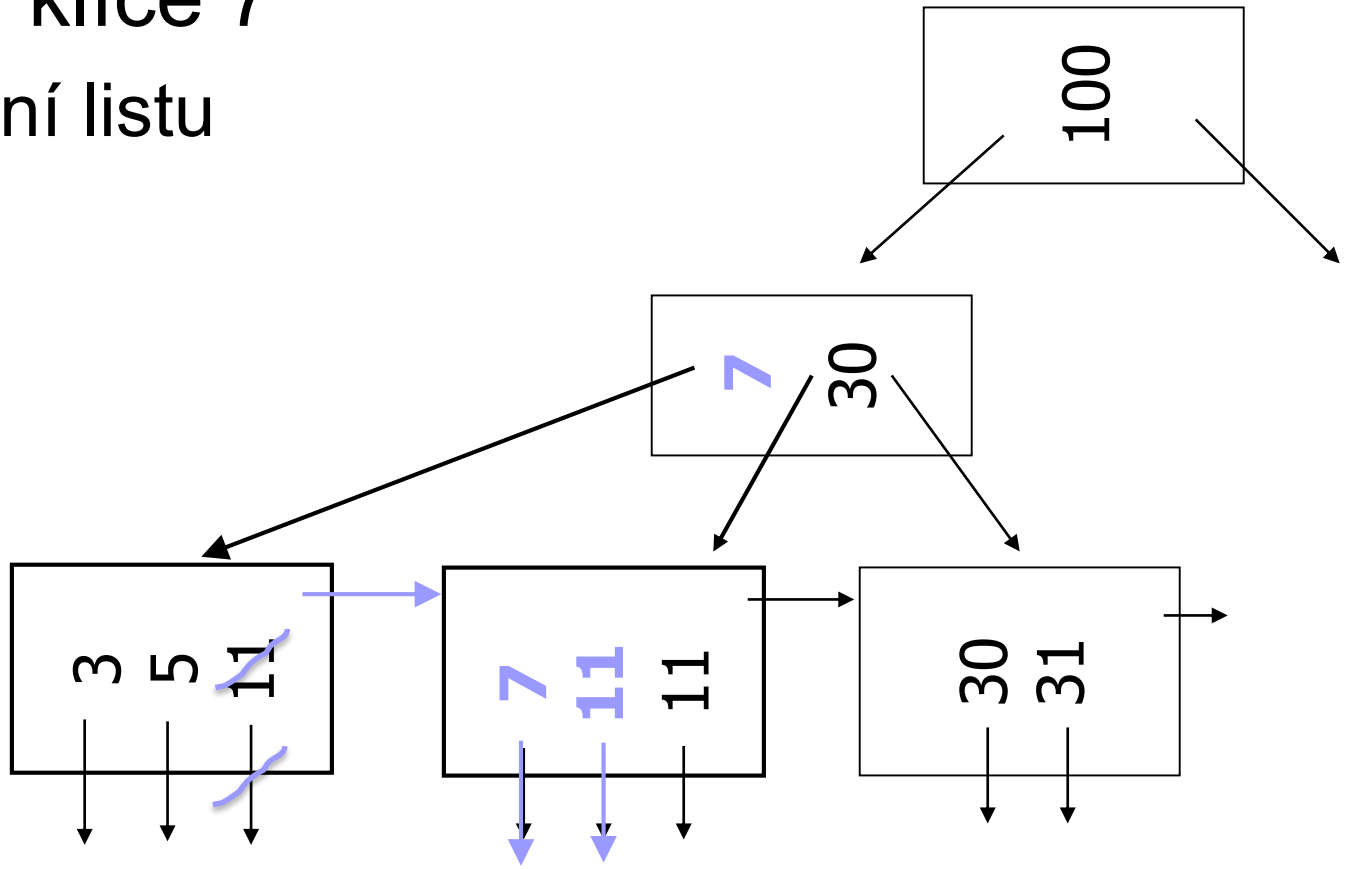
B$^+$-strom: *n=4*

- Vložení klíče 32
  - Bez reorganizace



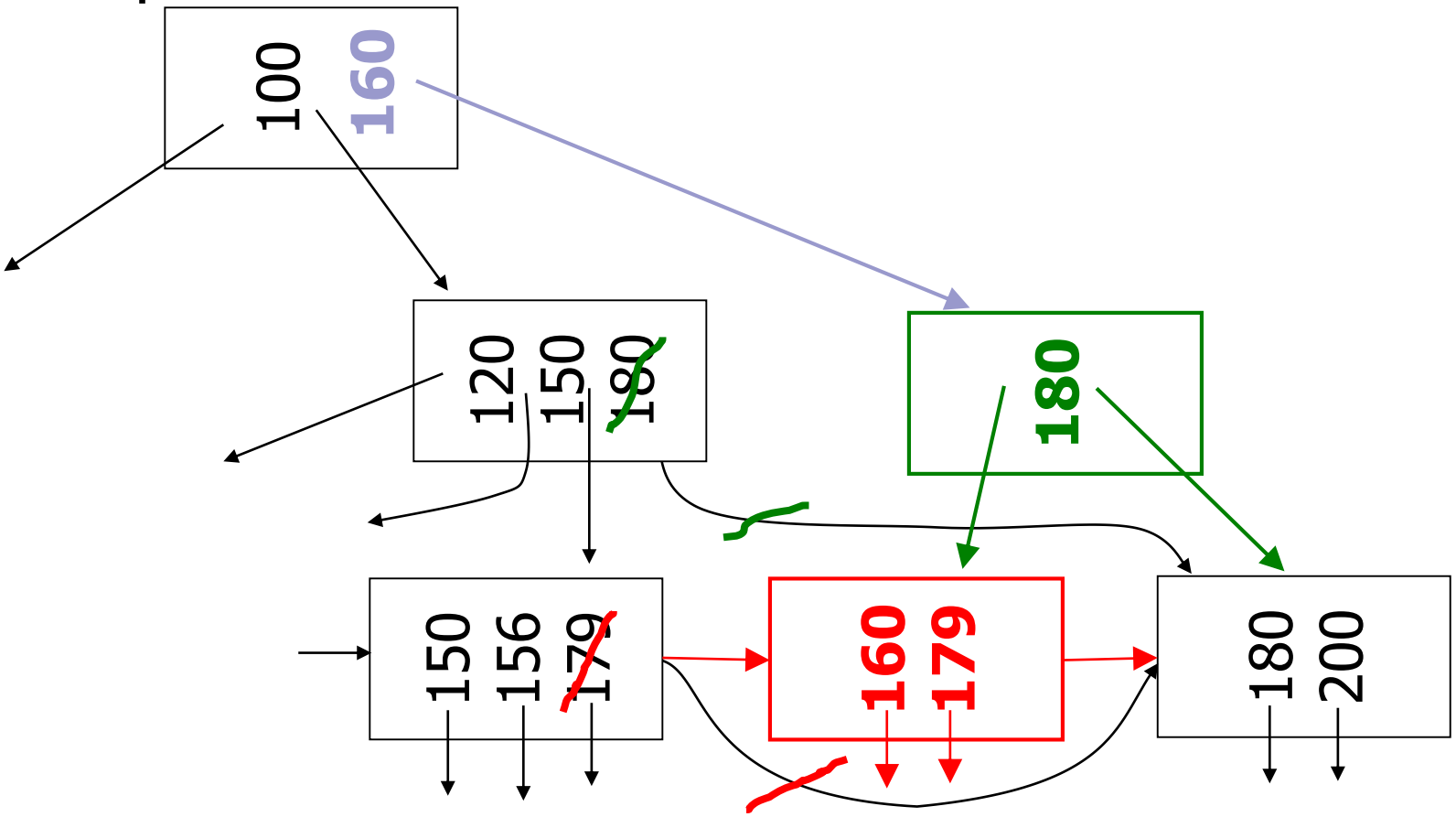
B$^+$-strom: *n=4*

- Vložení klíče 7
  - Štěpení listu



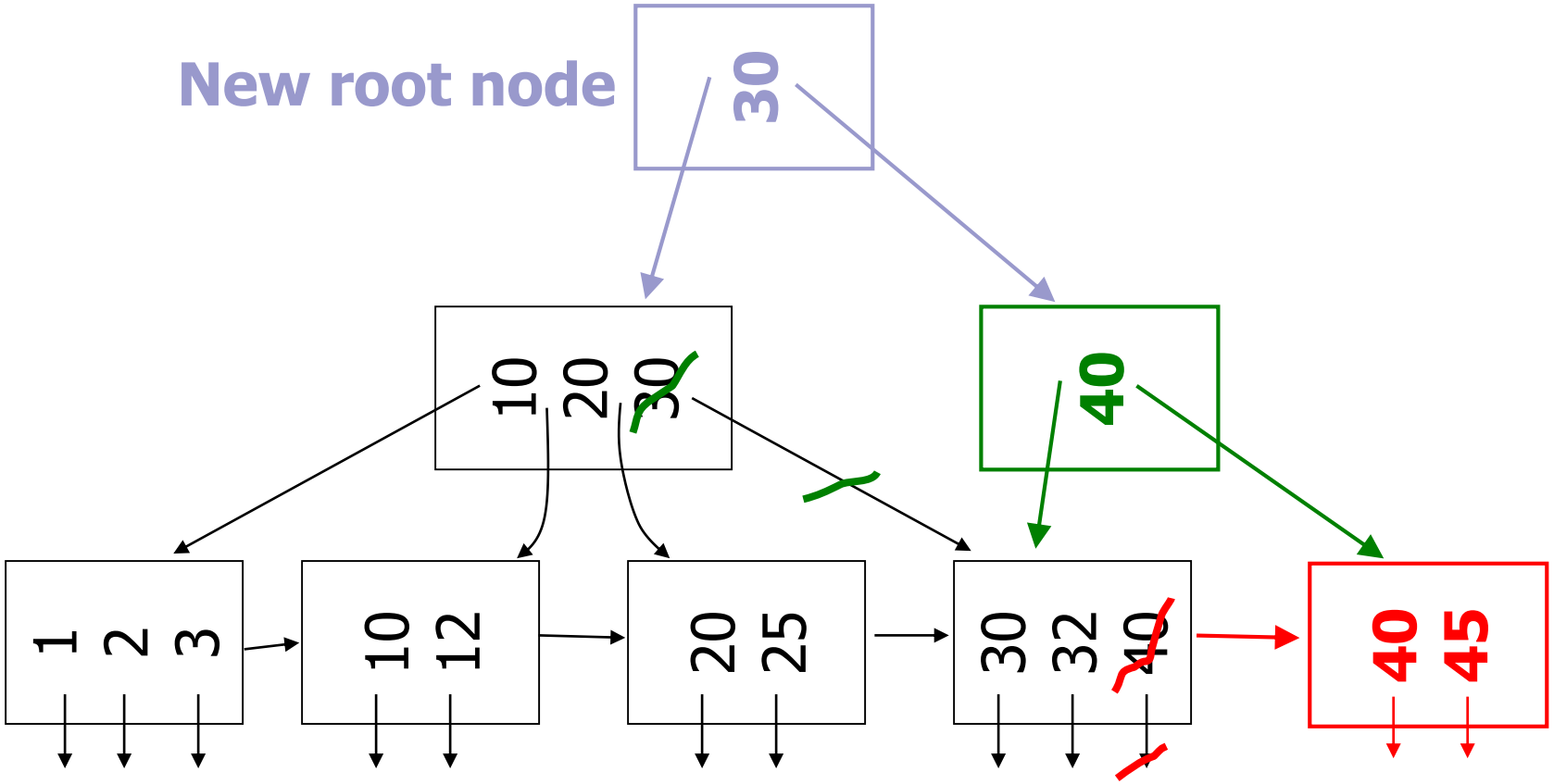
B$^+$-strom: *n=4*

- Vložení klíče 160
  - Štěpení vnitřního uzlu



B$^+$-strom: *n=4*

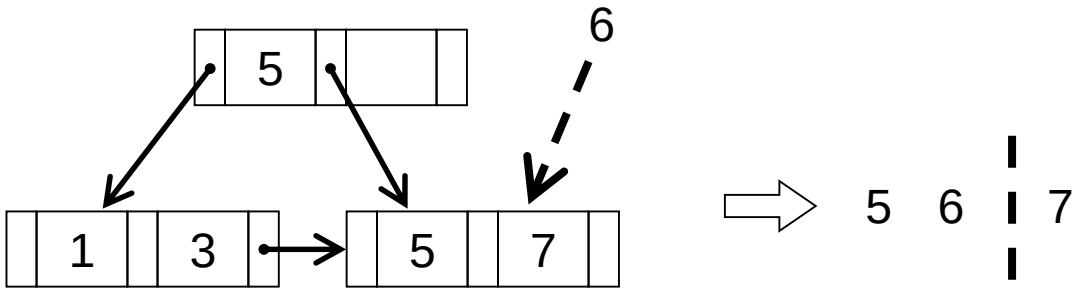
- Vložení klíče 45
  - Nový kořen



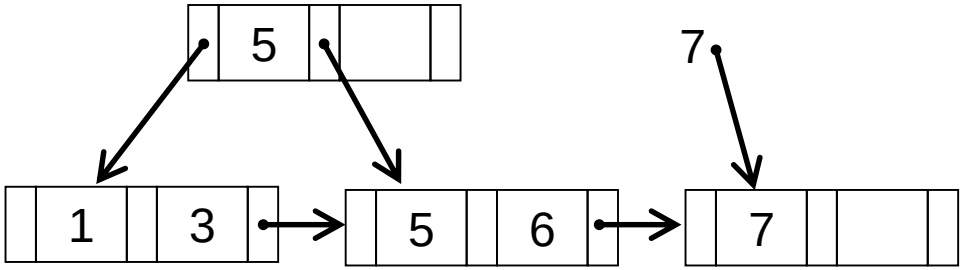
B$^+$-strom: štěpení listu

n=3, vkládáme 6

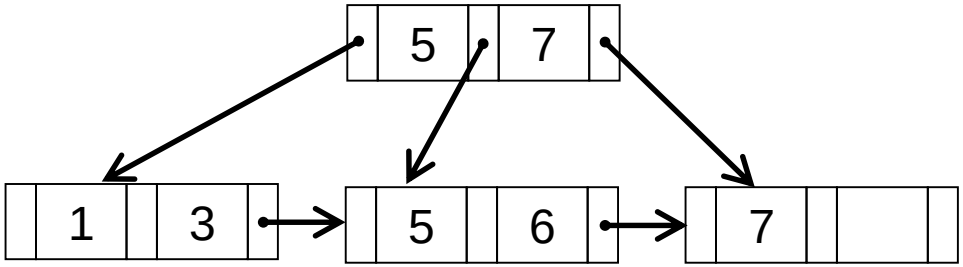
1.



2.

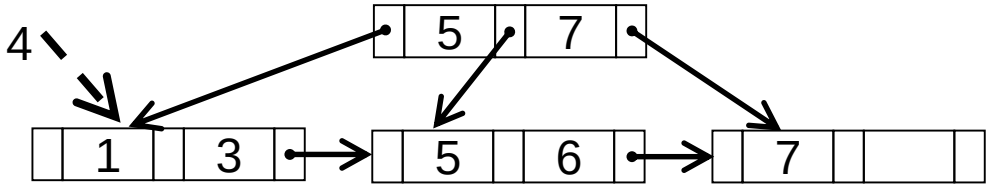


3.



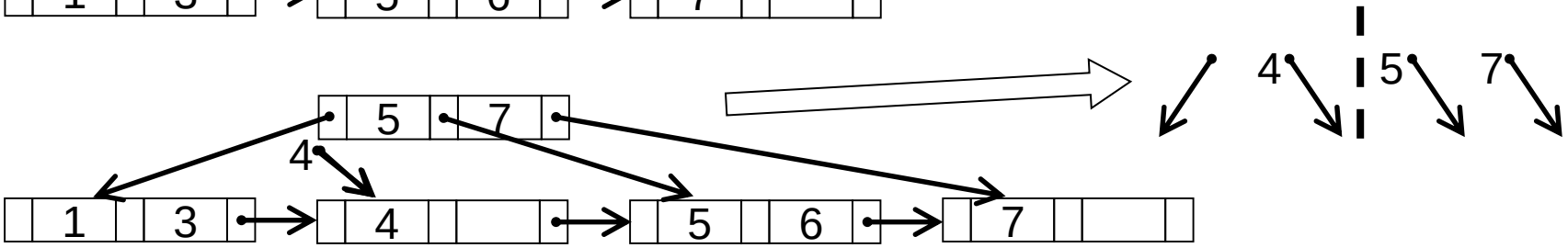
B$^+$-strom: štěpení vnitřního uzlu

1.

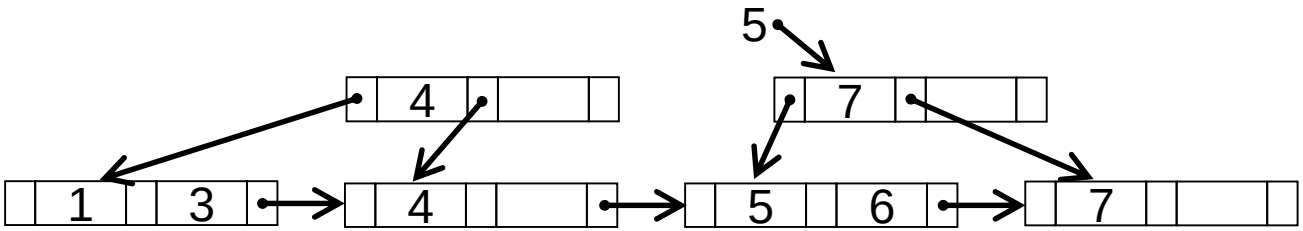


n=3, vkládáme 4

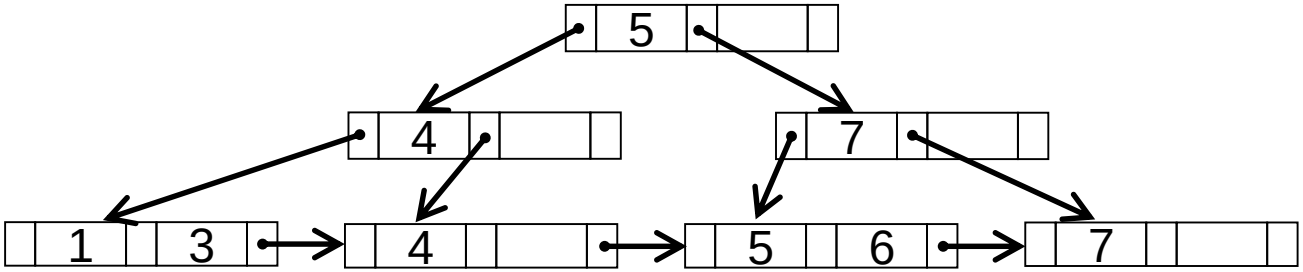
2.



3.



4.

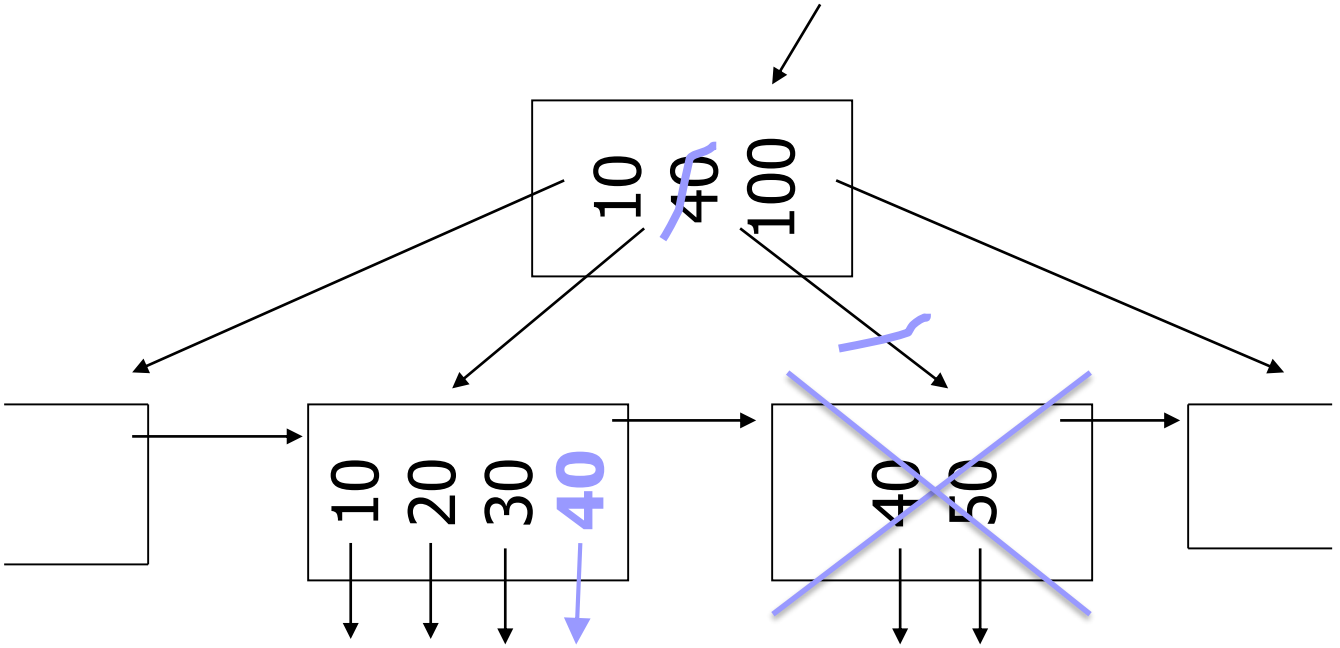


B^+-strom: mazání

- Nalézt listový uzel a smazat klíč
  - Včetně odkazu na záznam
  - Popř. smazat list, …
- Případy:
  a) Bez reorganizace (list není „podplněný“)
  b) Přesun do souseda a smaž uzel
  c) Přesuny mezi sousedy (bez mazání uzlu)
  d) Ad (b) a (c) pro vnitřní uzly

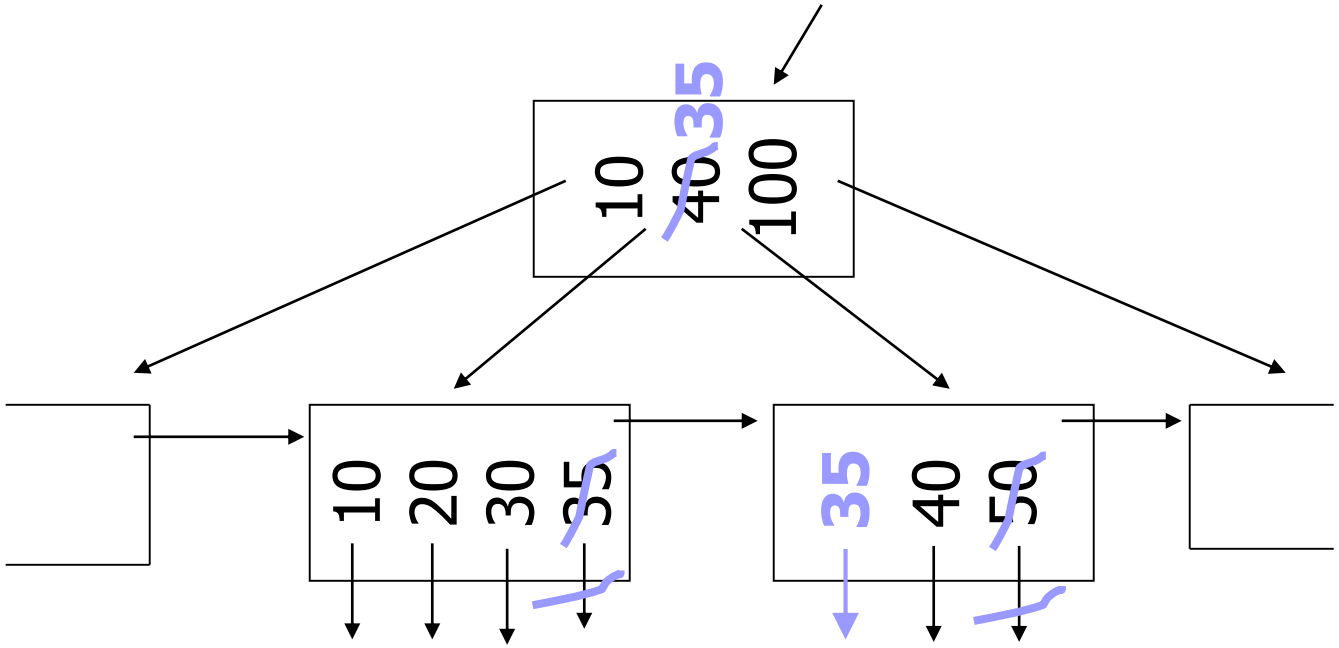
B$^+$-strom: *n*=5

- Mazání klíče 50
  - Přesun do souseda a smaž uzel



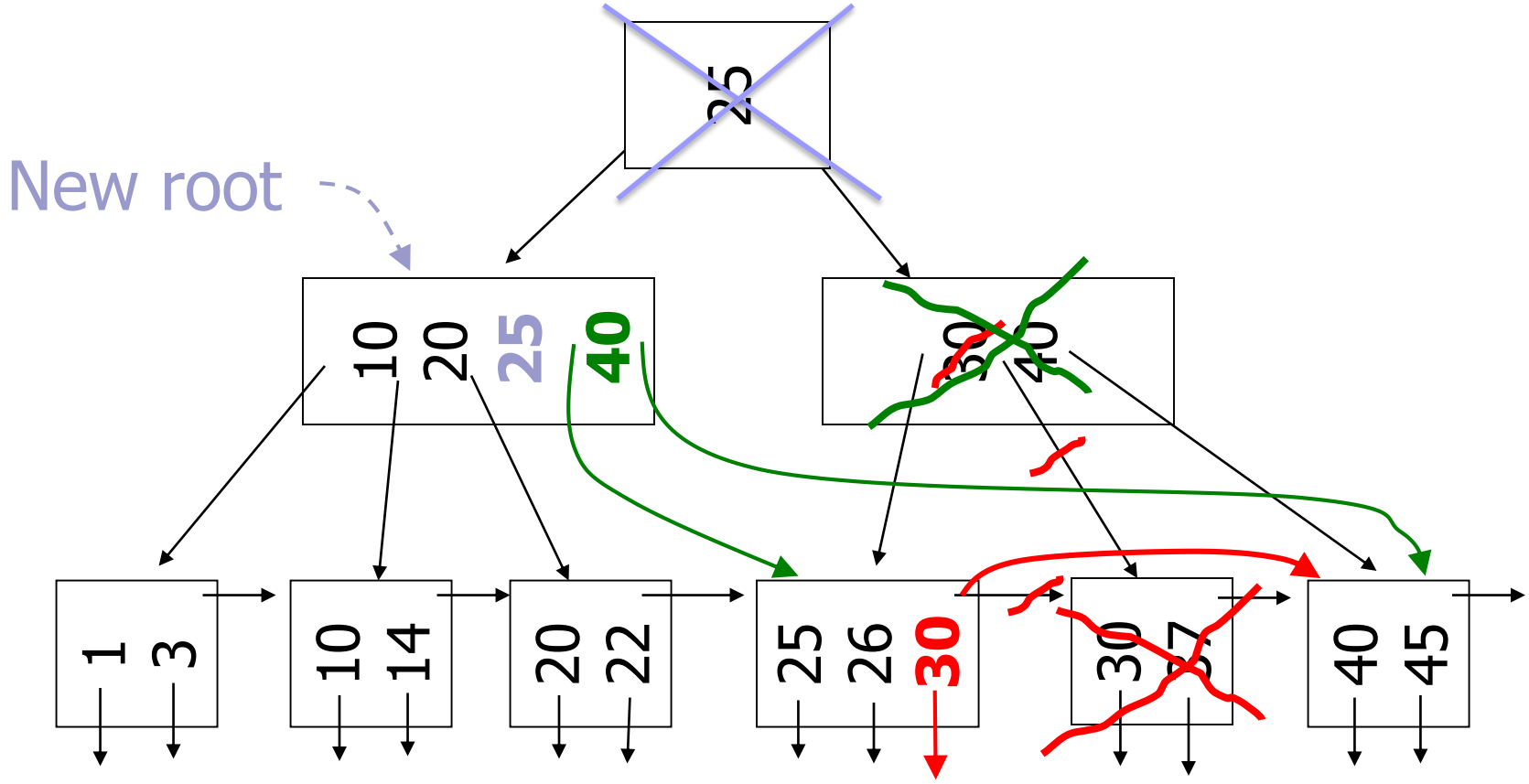
B$^+$-strom: *n*=5

- Mazání klíče 50
  - Přesuny mezi sousedy (bez mazání uzlu)



B$^+$-strom: *n*=5

- Mazání klíče 37
  - Přesuny mezi sousedy pro vnitřní uzly



B^+-strom: mazání

- Praxe:
  - Často nejsou přesuny mezi sousedy implementovány
    - Více vložení než mazaní (náhodně) má zaplnění 65-69% i bez přesunů
  - Příliš složité a nemá velké efekty

B^+-strom vs. index-sekvenční soubor

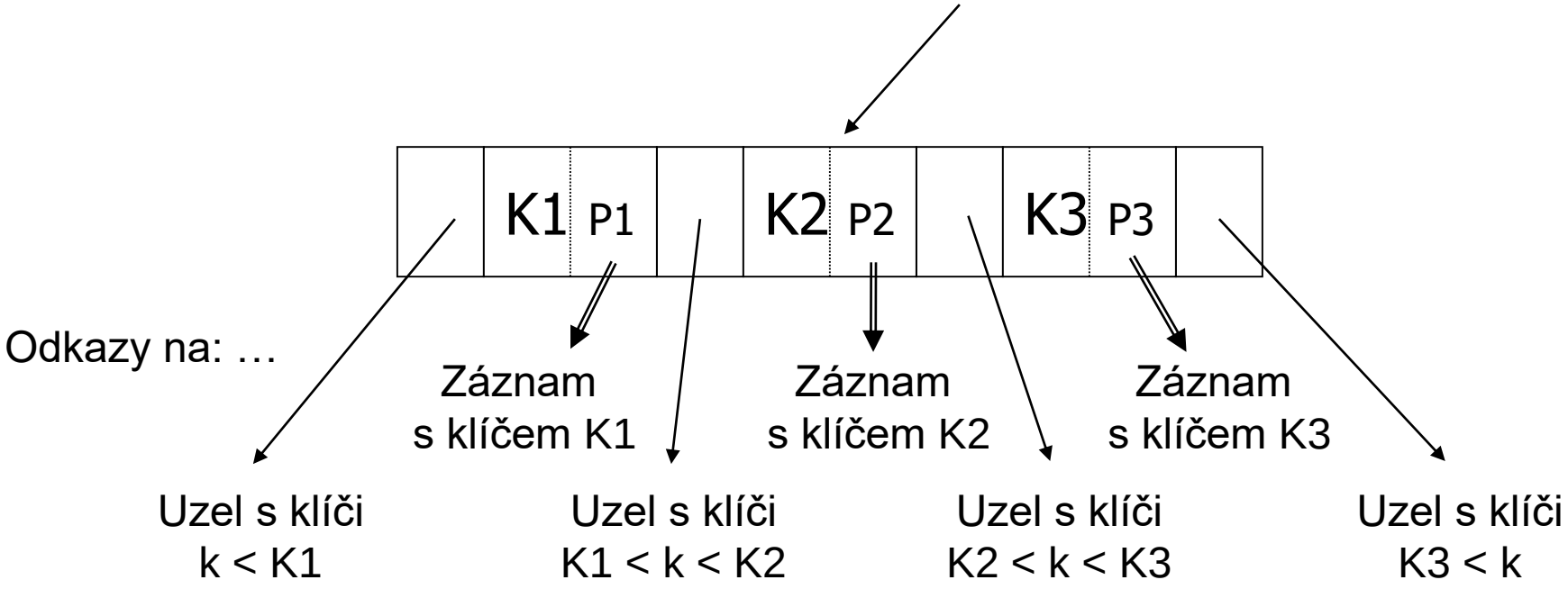
- Kapacita bloku (4 KiB)
  - Klíč = 4 bajty, ukazatel = 4 bajty
  - Řídký index a sekvenční soubor
    - implicitní indexy, tj. pouze klíče $\rightarrow$ 1024 klíčů
  - B^+-strom jako soubor
    - vnitřní uzel: 512 potomků, list již obsahuje 511 odkazů na bloky
- Porovnání – počet datových bloků v indexu:
  - Index o jedné úrovni:
    - Řídký index: každý blok $\rightarrow$ až 1024 bloků souboru
    - B^+-strom: jeden blok $\rightarrow$ až 511 bloků souboru
  - Index o dvou úrovních: (1. úroveň = právě 1 blok)
    - Řídký index : až 1 048 576 bloků souboru
    - B^+-strom: až 261 632 bloků souboru

B^+-strom vs. index-sekvenční soubor

- Závěr (pokračování):
  - ☹ B^+-strom má větší režii (potřebuje více místa)
    - ☺ Je plně dynamický
    - ☹ B^+-strom má složitější zamykání
  - ☹ Statický index vyžaduje reorganizace
    - DB neví kdy reorganizovat!
      - ☺ B^+-strom provádí pouze malé lokální reorganizace
      - ☹ Statický index provádí velké reorganizace
  - Buffer manager
    - ☺ B^+-strom má „fixní“ nároky (log hloubka)
    - ☹ Konvenční index nikoli! (lineární složitost)
      - Kvůli přetokovým oblastem
    - LRU není pro B^+-strom vhodný!
- **B^+-strom je vhodnější organizace.**

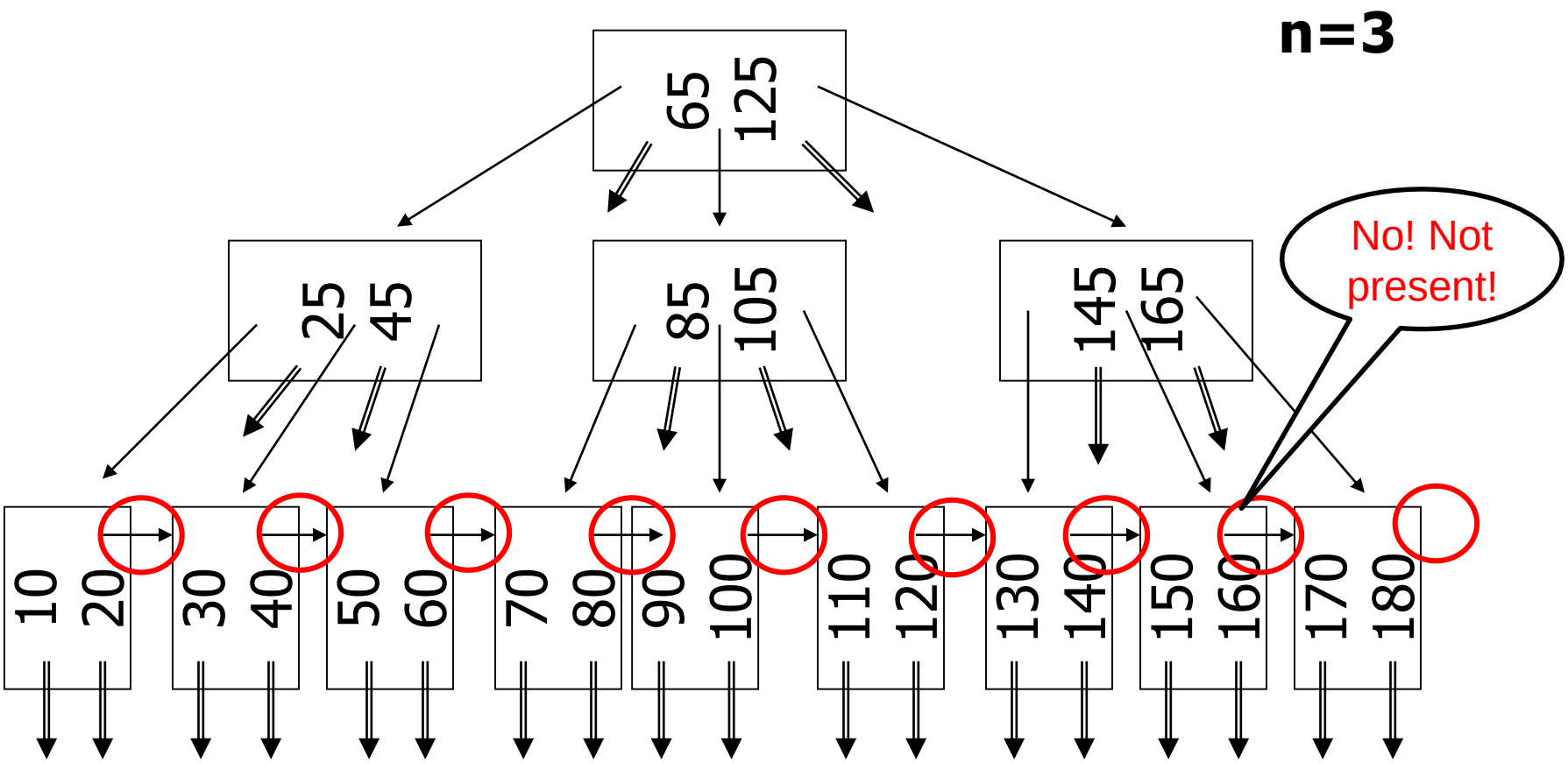
B-strom (pozor bez $^+$)

- Idea: nereplikovat klíče
  - $\rightarrow$ odkazy na záznamy i ve vnitřních uzlech
  - Podmínky pro klíče v podstromech jsou jiné



B-strom: příklad

- Zřetězení listů již nelze využít



B-strom

- Podmínky naplnění

	Max ukazatelů	Min ukazatelů	Max klíčů	Min klíčů
Vnitřní uzel (ne kořen)	n (potomků)	$\lceil n/2 \rceil$ (potomků)	n-1 (klíčů a odkazů)	$\lceil n/2 \rceil$ -1 (klíčů a odkazů)
Vnitřní uzel (kořen)	n (potomků)	2 (potomků)	n-1 (klíčů a odkazů)	1 (klíčů a odkazů)
List (ne kořen)	n-1 (záznamů)	$\lceil (n-1)/2 \rceil$ (záznamů)	n-1 (odkazů na záznamy)	$\lceil (n-1)/2 \rceil$ (odkazů na záznamy)
List (kořen)	n-1 (záznamů)	0 (záznamů)	n-1 (odkazů na záznamy)	0 (odkazů na záznamy)

Porovnání B-stromu a B^+-stromu

- Velikosti
  - Blok = 4KiB
  - Ukazatel = 4 bajty
  - Klíč = 4 bajty
- Uvažujme vždy *plný dvouúrovňový* strom
  - 1 kořen a pak listy
  - Každý uzel v jednom bloku

Porovnání: B-strom

- Kořen:
  - 341 klíčů + 341 ukazatelů na záznam
  - 342 ukazatelů na potomky
    - 341·8 + 342·4 = 4096 bajtů
- List:
  - 512 klíčů + 512 ukazatelů na záznam
    - 512 · 8 = 4096 bajtů
- Počet záznamů:
  - 341 + 342 · 512 = 175 445 záznamů

Porovnání: B^+-strom

- Kořen:
  - 511 klíčů, 512 ukazatelů na potomka
    - $511 \cdot 4 + 512 \cdot 4 = 4092$ bajtů
- List:
  - 511 klíčů + 511 ukazatelů na záznam
    - $511 \cdot 8 + 4 = 4092$ bajtů
- Počet záznamů:
  - $512 \cdot 511 = 261\,632$ záznamů

Porovnání: výsledek

- Počet čtení z disku:
  - B-strom
    - $P_{1\ čtení} = 341 / 175\ 445 = 0{,}2\%$
    - $P_{2\ čtení} = 1 - P_{1\ čtení} = 99{,}8\%$
  - B^+-strom
    - $P_{1\ čtení} = 0 / 261\ 632 = 0\%$
    - $P_{2\ čtení} = 100\%$

Porovnání: závěr

- B-stromy
  - Ve vyhledávání rychlejší
    - Ne vždy platí (viz předchozí příklad)
  - Různé struktury vnitřního a listového uzlu
  - Mazání je obtížnější na implementaci

➔ B^+-stromy jsou preferovány!

B^+-strom

- B^+-strom jako soubor
  - viz Organizace souborů PV062
- Duplicitní klíče
  - Odkazy z listů = odkazy na kyblíky (buckets)
    - Tj. bloky, kde jsou seznamy záznamů se stejnou hodnotou
- Klíče jsou variabilní délky (např. řetězce)
  - Ukládat celé → nízká arita, proměnlivá arita, …
  - Používá se prefix (prefix compression)