Indexy založené na hashovaní

Úvod

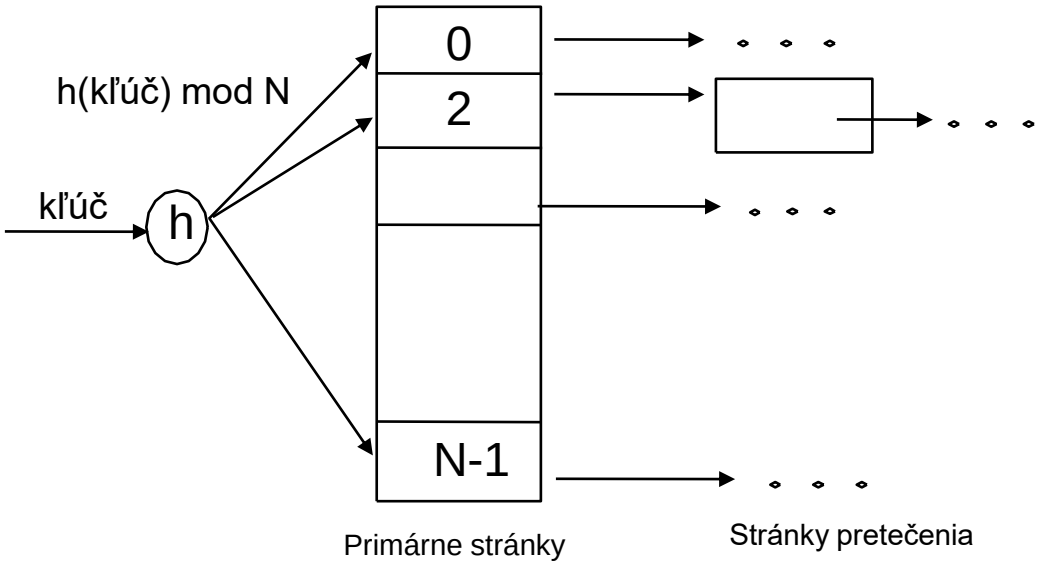
- Indexy založené na hashovaní sú **vhodné len pre dopyty s rovnosťou**. **Nepodporujú rozsahové** (range) **dopyty**.

- Dobrá hashovacia funkcia musí byť:
  - všade použiteľná
  - rýchla
  - rovnomerne distribuovať hodnoty

- Existujú statické a dynamické hashovacie techniky; výber podobný ako pri ISAM a B+ stromoch.

Statické hashovanie

- Počet primárnych stránok je fixný, uložené sekvenčne, nikdy nemazané; v prípade potreby sa použijú stránky pretečenia.
- **h**(*k*) mod M = oblasť, do ktorej patria údaje s kľúčom *k.* (M = počet oblastí)



Statické hashovanie (pokr.)

- Oblasti obsahujú k*, <k,rid> alebo <k, list-rid>.

- Hashovacia funkcia pracuje s kľúčom záznamu *r*. Jej úlohou je určiť oblasť z rozsahu 0 … M-1.
  - **h**(*kľúč*) = (a * *kľúč* + b) je typická jednoduchá hashovacia funkcia.
  - *a* a *b* sú konštanty; existuje viacero vylepšení funkcie **h**.

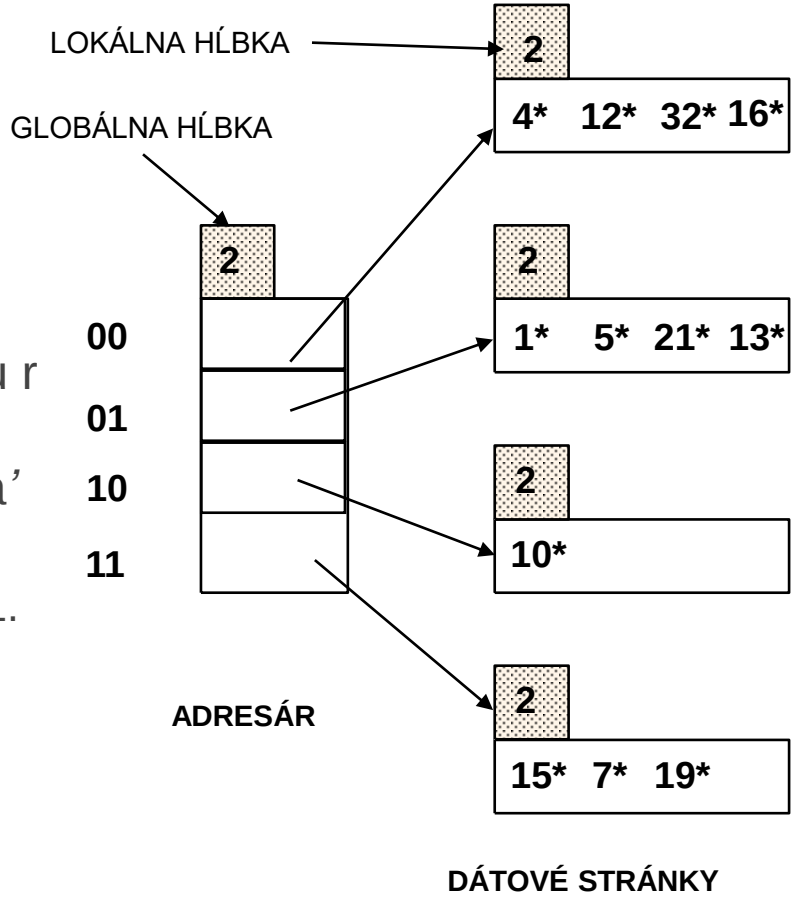
- Môžu však vzniknúť **príliš dlhé oblasti s mnohými stránkami pretečenia** a tým sa zníži výkon.
  - **Rozšíriteľné** a **Lineárne hashovanie**: dynamické techniky, ktoré riešia tento problém.

Rozšíriteľné (Extendible) hashovanie

- Situácia: Bunka (primárna stránka) je už plne obsadená. Prebudujeme oblasti tak, že ich zdvojnásobíme
  - Čítanie a zapisovanie všetkých stránok je drahá operácia!
  - **Myšlienka**: Použiť **adresár smerníkov na oblasti**, zdvojnásobime počet oblastí zdvojnásobením adresára a fyzicky rozdelíme len tú oblasť, ktorá pretiekla.
  - Adresár je menší ako samotný dátový súbor, preto je jeho zdvojnásobenie lacnejšia operácia. Rozdelíme len jednu oblasť - **Nevznikajú žiadne stránky pretečenia!**
  - Trik spočíva v prispôsobení hashovacej funkcie

Príklad

- Adresár je pole o veľkosti 4.
- Pre nájdenie oblasti záznamu r vezmeme toľko posledných bitov **h**(*r*) ako 'globálna hĺbka'
  - Ak **h**(*r*) = 5 = binárne 101, čítame smerník v políčku 01.

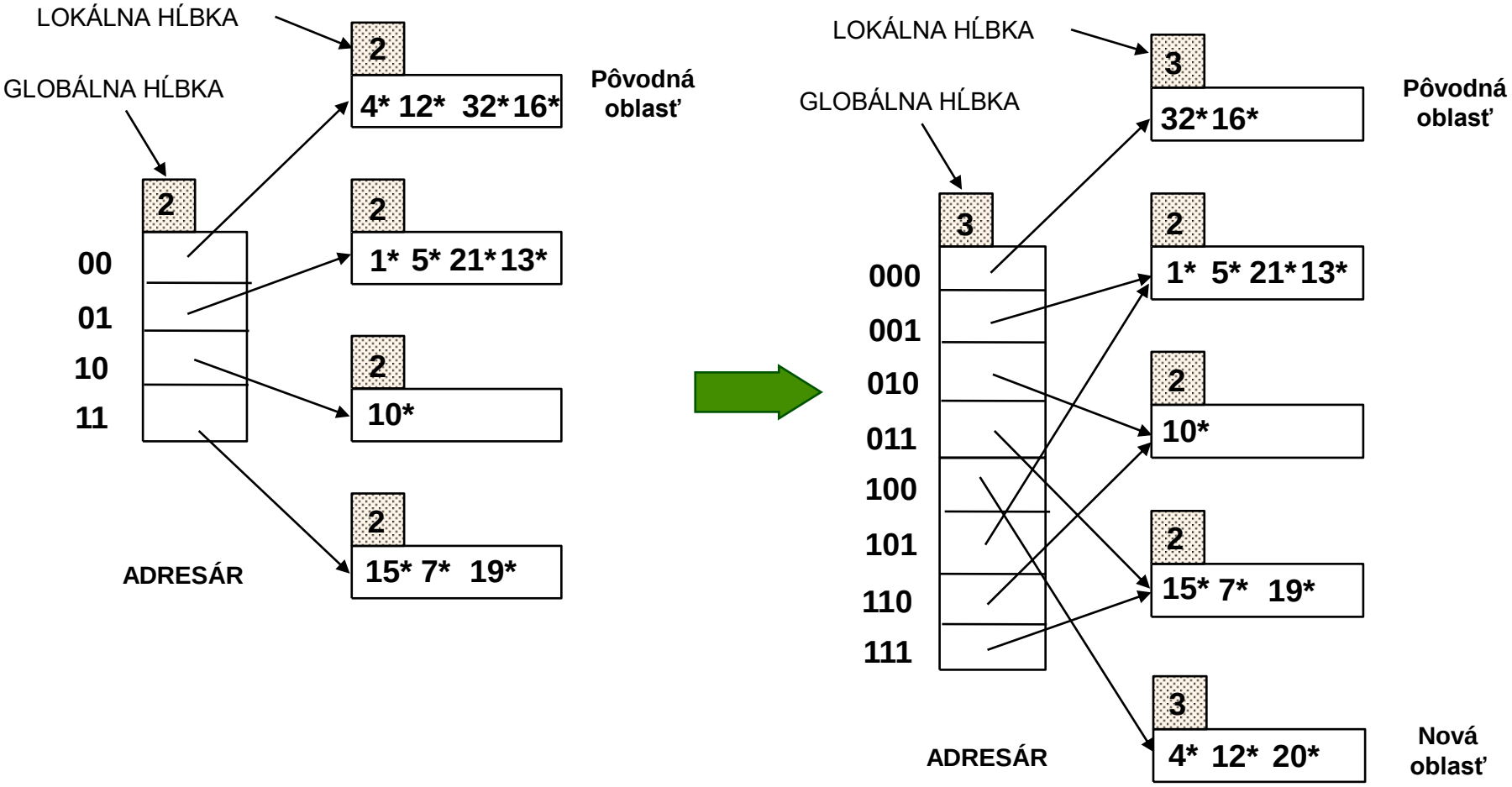


Vkladanie:

1.) Ak je bunka plná, **rozdelíme ju** (pridáme novú stránku).

2.) Ak je lokálna hĺbka delenej oblasti rovná globálnej, zväčšíme adresár.

Vloženie h(r)=20 (binárne 10100)



Niekoľko poznámok

- 20 = binárne 10100. Posledné **2** bity (00) nám hovoria, že 20* patrí do pôvodnej alebo novej oblasti. Posledné **3** bity hovoria, do ktorej z nich.
  - **Globálna hĺbka adresára**: Maximálny počet bitov potrebný na určenie, ktorej bunke patrí daná položka.
  - **Lokálna hĺbka oblasti**: Počet bitov potrebných na zistenie, či položka patrí do danej oblasti.

- Kedy spôsobuje rozdelenie bunky zdvojenie adresára?
  - Pred vložením platí: lokálna hĺbka bunky = globálna hĺbka. Po vložení je lokálna hĺbka väčšia ako globálna.
  - Adresár je zdvojený jeho skopírovaním a pripojením na koniec a nastavením ukazovateľa na rozdelenú stránku. (Využitie najmenej významných bitov (least significant bits) umožňuje efektívne zdvojnásobenie adresára!)

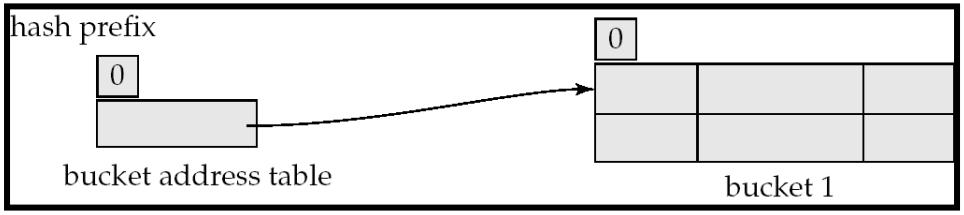
Ďalšie poznámky

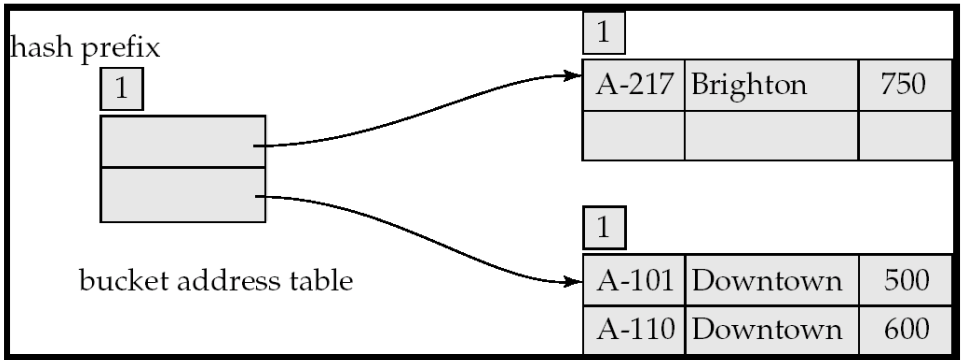
- Ak sa adresár zmestí do pamäte, dopyt s rovnosťou vieme vykonať na jeden prístup; v opačnom prípade na 2.
  - 100 B na záznam, 1 000 000 záznamov (riadkov), ak máme 4KB stránky so zaplnením cca 80% máme cca 33 záznamov na oblasť, potrebujeme cca 33 000 stránok
  - 32 000 * 8B = 256 kB - šance, že sa adresár vôjde do pamäte sú veľké.
  - Ak sú údaje "šikmé", adresár môže narásť do obrovských rozmerov.
  - Viacnásobné záznamy s rovnakou hodnotou hashu môžu spôsobiť pretečenie!

- **Mazanie**: Ak odstránenie záznamu vyprázdni oblasť, môžeme ju zlúčiť s oblasťou s rovnakým suffixom o 1 menším ako lokálna hĺbka. Ak sú všetky lokálne hĺbky menšie ako globálna, môžeme skresať adresár na polovicu.

Príklad rozšíriteľného hashovania

branch_name	h(*branch_name*)
Brighton	0010 1101 1111 1011 0010 1100 0011 0000
Downtown	1010 0011 1010 0000 1100 0110 1001 1111
Mianus	1100 0111 1110 1101 1011 1111 0011 1010
Perryridge	1111 0001 0010 0100 1001 0011 0110 1101
Redwood	0011 0101 1010 0110 1100 1001 1110 1011
Round Hill	1101 1000 0011 1111 1001 1100 0000 0001



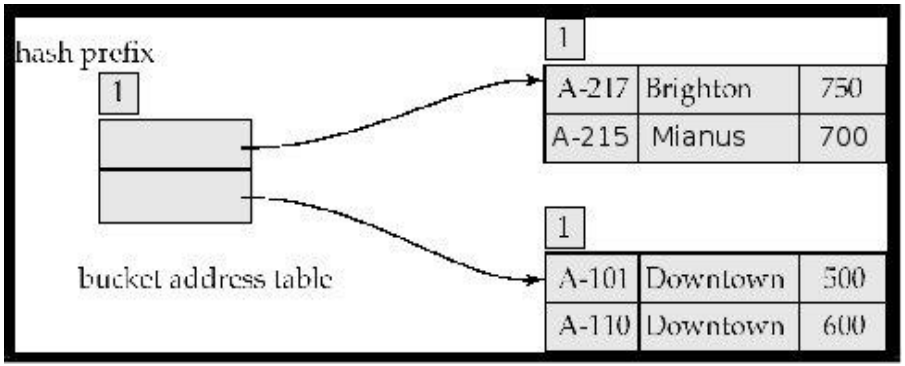


Vkladáme postupne:

- Brighton
- Downtown
- Downtown
- Mianus
- Perryridge
- Perryridge
- Perryridge
- Round Hill

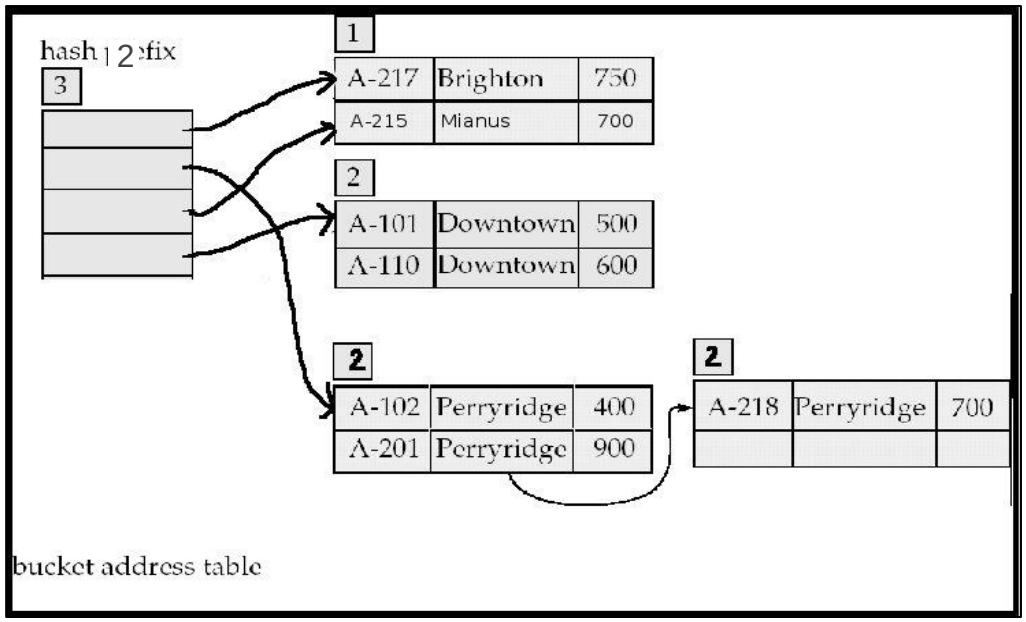
Príklad rozšíriteľného hashovania

branch_name	h(*branch_name*)
Brighton	0010 1101 1111 1011 0010 1100 0011 0000
Downtown	1010 0011 1010 0000 1100 0110 1001 1111
Mianus	1100 0111 1110 1101 1011 1111 0011 1010
Perryridge	1111 0001 0010 0100 1001 0011 0110 1101
Redwood	0011 0101 1010 0110 1100 1001 1110 1011
Round Hill	1101 1000 0011 1111 1001 1100 0000 0001



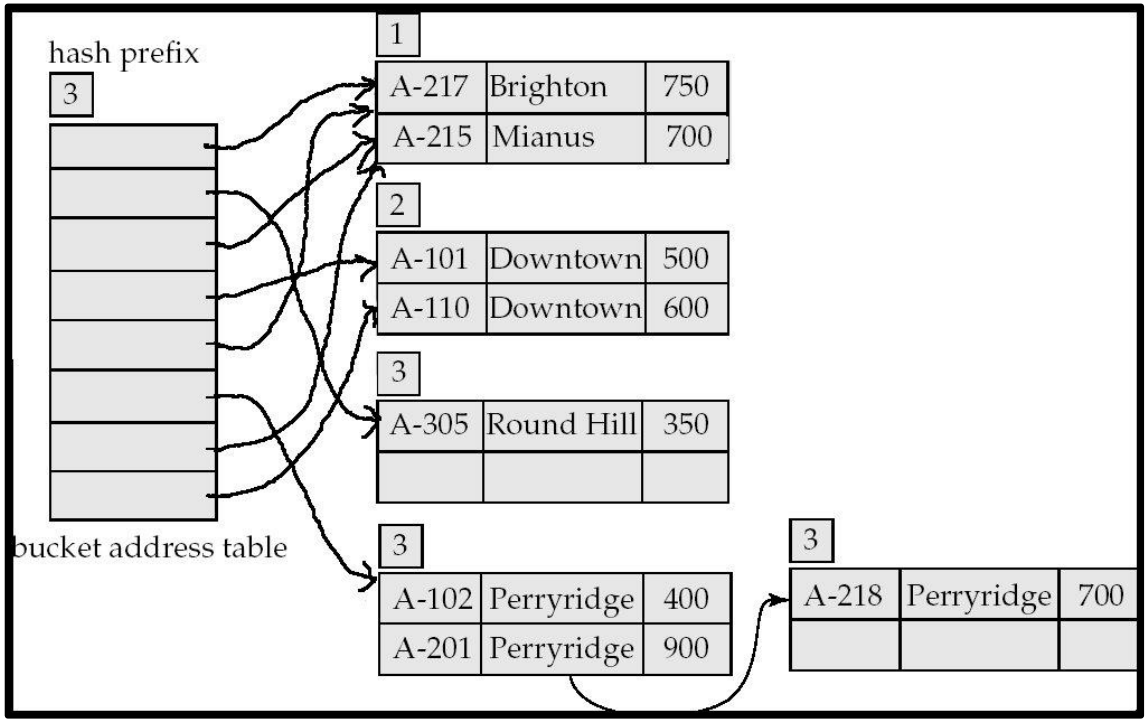
Príklad rozšíriteľného hashovania

branch_name	h(*branch_name*)
Brighton	0010 1101 1111 1011 0010 1100 0011 0000
Downtown	1010 0011 1010 0000 1100 0110 1001 1111
Mianus	1100 0111 1110 1101 1011 1111 0011 1010
Perryridge	1111 0001 0010 0100 1001 0011 0110 1101
Redwood	0011 0101 1010 0110 1100 1001 1110 1011
Round Hill	1101 1000 0011 1111 1001 1100 0000 0001



Príklad rozšíriteľného hashovania

branch_name	h(*branch_name*)
Brighton	0010 1101 1111 1011 0010 1100 0011 0000
Downtown	1010 0011 1010 0000 1100 0110 1001 1111
Mianus	1100 0111 1110 1101 1011 1111 0011 1010
Perryridge	1111 0001 0010 0100 1001 0011 0110 1101
Redwood	0011 0101 1010 0110 1100 1001 1110 1011
Round Hill	1101 1000 0011 1111 1001 1100 0000 0001

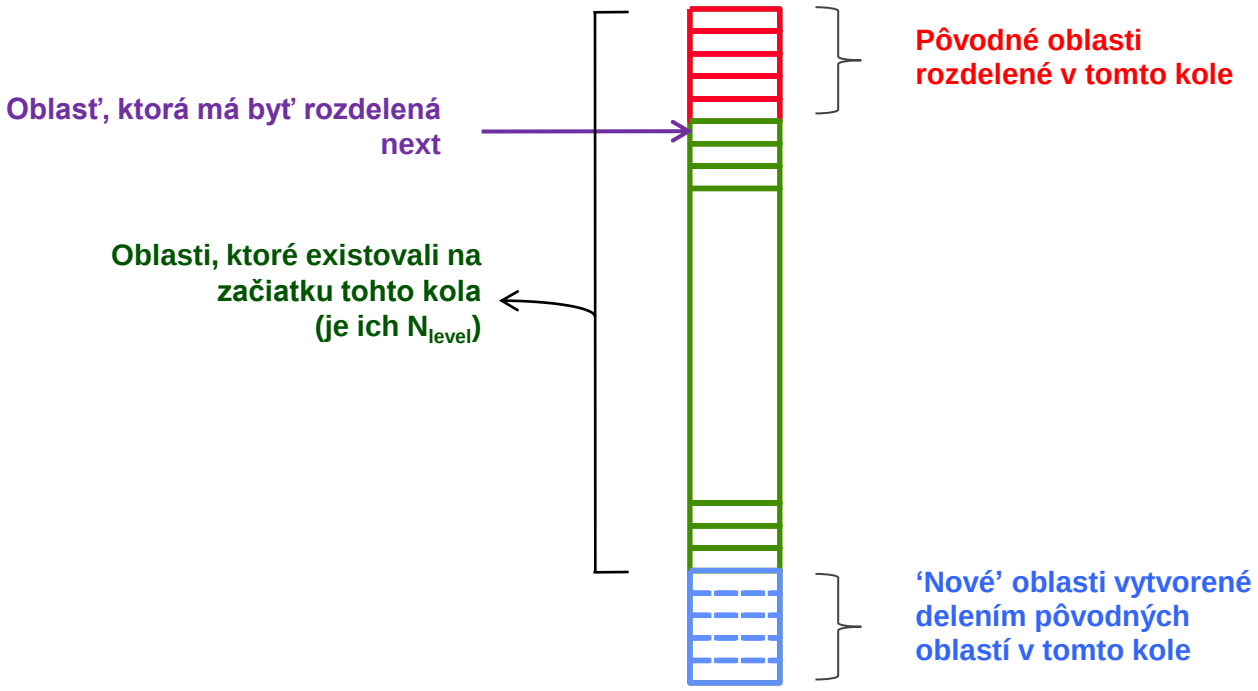


Lineárne hashovanie

- Dynamická hashovacia schéma, alternatíva ku rozšíriteľnému hashovaniu.
- LH na riešenie problémov so stránkami pretečenia nepoužíva adresár, ale pracuje s duplicitou.
- Princíp:
  - Delenia sa realizujú v **kolách**.
  - Aktuálne kolo je označené **level**.
  - Kolo končí, keď všetky oblasti, prítomné na začiatku kola (počet oblastí = N_{level}), sú rozdelené ($N_{level+1} = 2*N_{level}$).
  - Uprostred kola: Oblasti od 0 po *next-1* už boli rozdelené; oblasti od *next* po N_{level} sa ešte budú deliť.

Prehľad súboru LH

Stav uprosted kola:



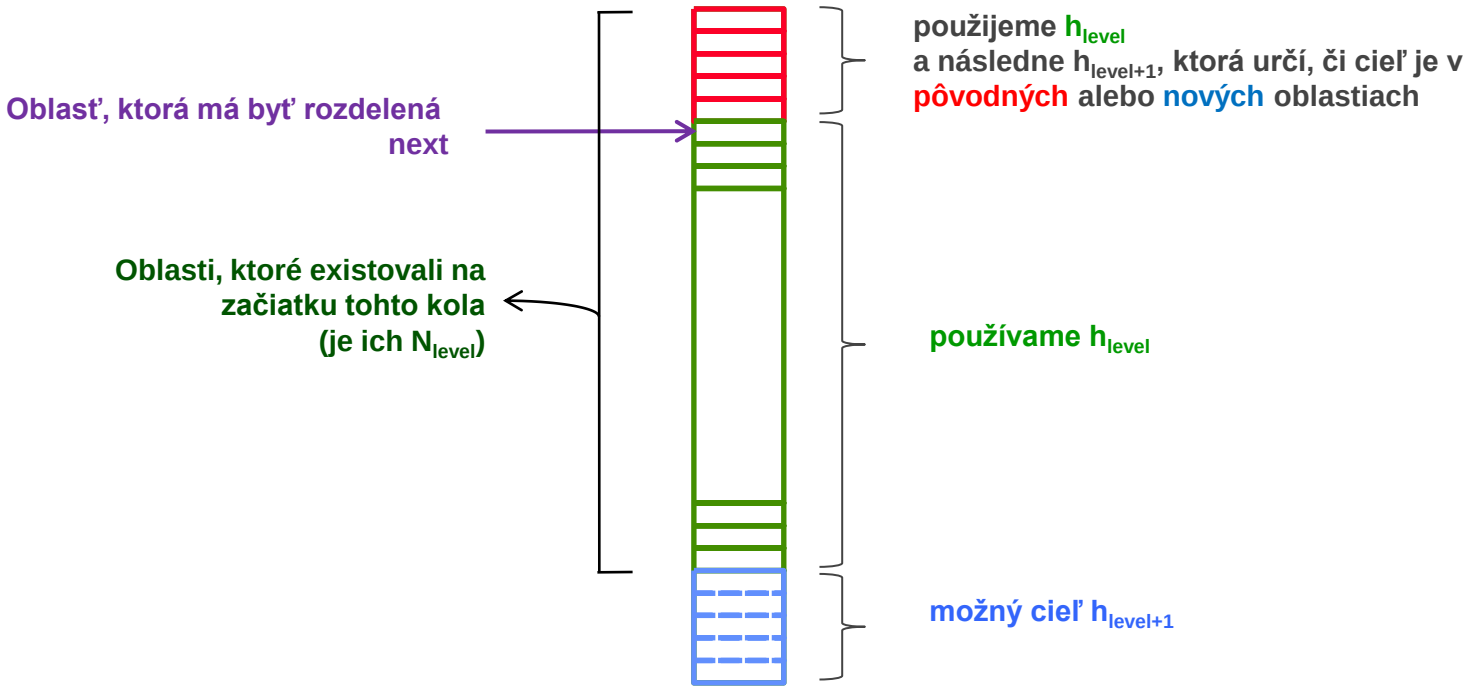
Lineárne hashovanie (pokr.)

- **Myšlienka**: Použitie postupnosti hash f-cií $\mathbf{h}_0$, $\mathbf{h}_1$, $\mathbf{h}_2$, ...
  - $\mathbf{h}_{level}(k) = \mathbf{h}(k) \bmod (2^{level} N)$; N = počiatočný počet oblastí
  - **h** je nejaká hashovacia funkcia
  - Ak $N = 2^d$, pre nejaké d, $\mathbf{h}_{level}$ vyrátame aplikovaním **h** a pozretím sa na posledných d_{level} bitov, kde $d_{level} = d + level$.
  - $\mathbf{h}_{level+1}$ zdvojnásobuje rozsah $\mathbf{h}_{level}$ (podobne ako pri zdvojení adresára)

- **Hľadanie:** Pre nájdenie oblasti podľa kľúča *k,* musíme vypočítať $\mathbf{h}_{level}(k)$*:*
  - Ak $\mathbf{h}_{level}(k)$ je v rozsahu od *next* po $N_{level} - 1$, ideme do oblasti $\mathbf{h}_{level}(k)$.
  - Hľadaná oblasť je buď $\mathbf{h}_{level}(k)$ alebo $\mathbf{h}_{level}(k) + N_{level}$, na zistenie musíme aplikovať $\mathbf{h}_{level+1}(k)$.

Prehľad súboru LH

Stav uprosted kola:



Lineárne hashovanie (pokr.)

- **Vkladanie**: Nájdenie bunky aplikovaním $\mathbf{h}_{Level}$ / $\mathbf{h}_{Level+1}$:
  - Ak bunka do ktorej ideme vkladať je plná:
    - Pridaj stránku pretečenia a ulož do nej záznam.
    - Rozdeľ bunku označenú *next* a inkrementuj *next*.

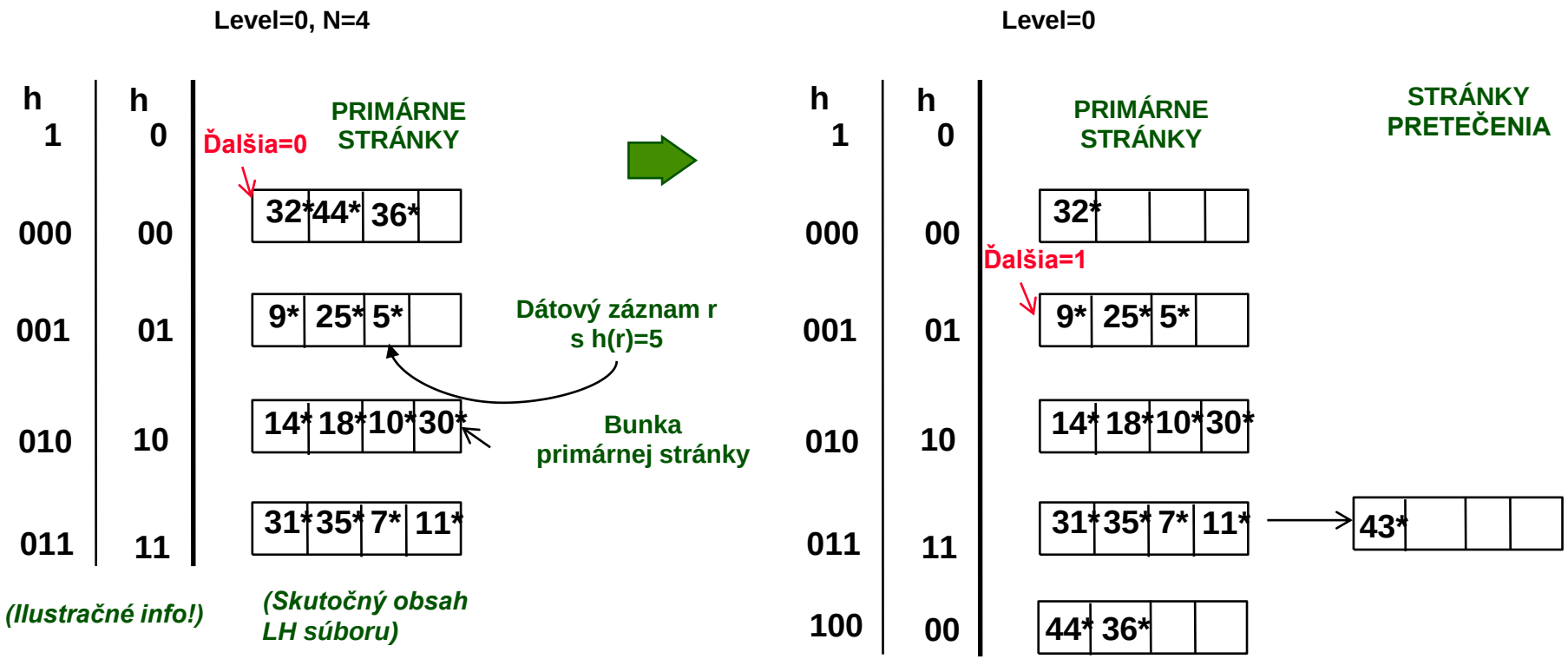
- Kritérium na spustenie delenia je voliteľné.

- Keďže bunky sú delené po kolách, nie sú vytvorené zbytočne dlhé reťaze buniek pretečenia.

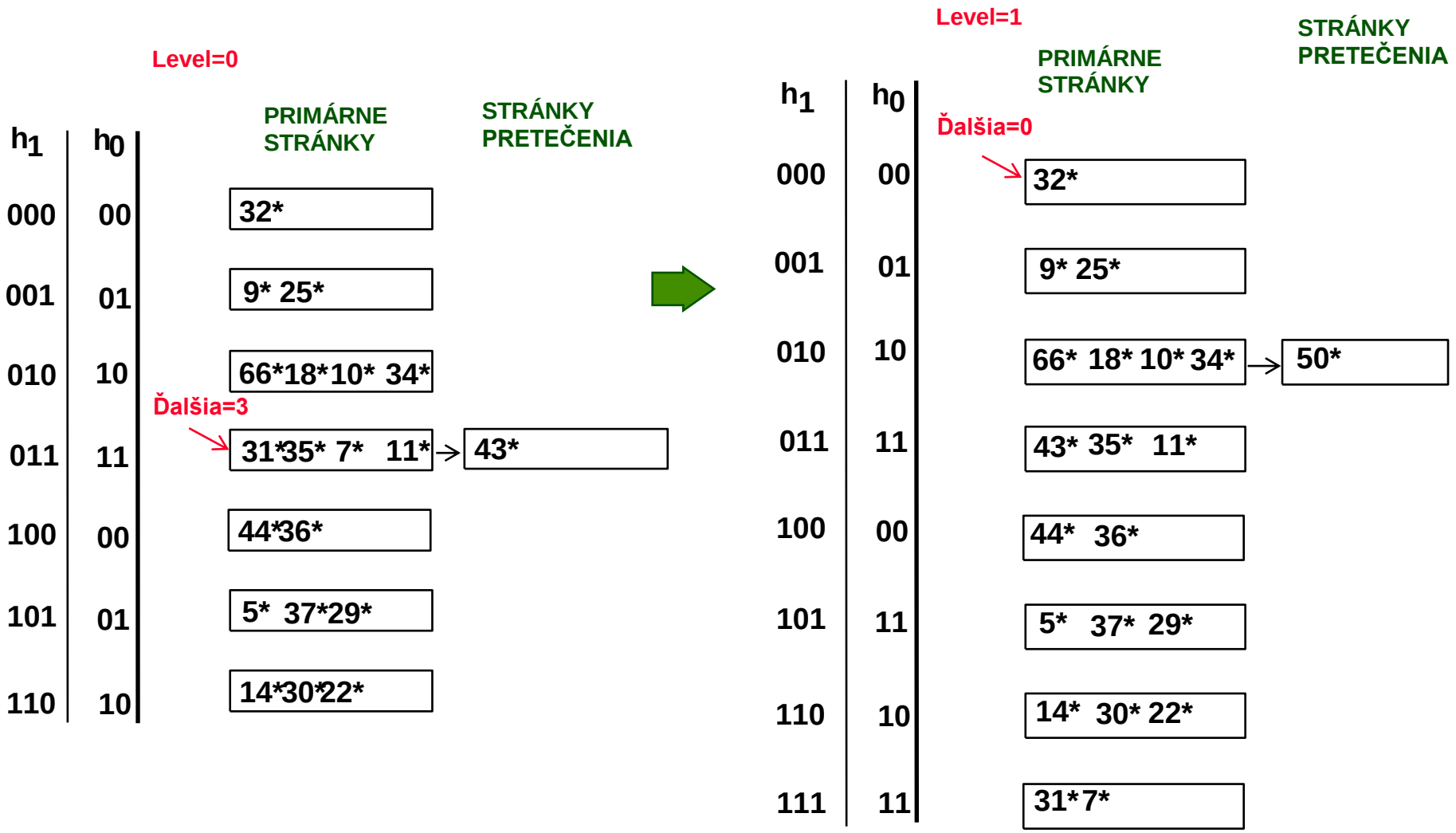
- Zdvojenie adresára v PH je podobné; menenie hashovacích funkcí je *implicitne* dané tým ako rastie počet bitov, ktoré sú skúmané.

Príklad na LH

Pri delení, $\mathbf{h}_{Level+1}$ je použitá na preusporiadanie záznamov.



Príklad - koniec kola



LH ako variant RH

- Obidve schémy sú si dosť podobné:
  - Začiatok s RH indexom, kde adresár má *N* prvkov.
  - Použitie stránok pretečenia, bunky delené po kolách.
  - Prvé delenie je na bunke 0. (Predstavte si, že sa v tomto momente zdvojnásobí adresár.) Ale prvky <1,*N*+1>, <2,*N*+2>, ... ostávajú rovnaké. Takže je potrebné vytvoriť len prvok adresára N, ktorý je rôzny od 0.
    - Keď delíme bunku 1, vytvoríme prvok adresára *N*+1, atď.

- To znamená, že adresár rastie postupne. Taktiež, primárne bunky sú vytvárané v poradí. Ak by boli tiež tak alokované, nepotrebovali by sme adresár.

Záver

- Indexy založené na hashovaní: použiteľné pre dopyty s rovnosťou, nepodporujú rozsahové dopyty.
- Statické hashovacie metódy vytvárajú dlhé reťazce stránok pretečenia.
- Rozšíriteľné hashovanie zabraňuje vytváraniu stránok pretečenia delením oblastí
  - Smerníky na oblasti obsahuje adresár, ktorý postupne rastie
  - Pri šikmých údajoch rastie príliš rýchlo, ak sa nezmestí do pamäte, narastie počet I/O

Záver (pokr.)

- Lineárne hashovanie nepoužíva adresár, ale delí oblasti po kolách.
  - Tým pádom nevznikajú dlhé reťazce stránok pretečenia.
  - Ľahké narábanie s duplikátmi
  - Využitie miesta môže byť horšie ako pri rozšíriteľnom hashovaní, keďže delenia nie sú koncentrované na 'husté' dátové oblasti.
    - Kritérium na delenie je možné vylepšiť

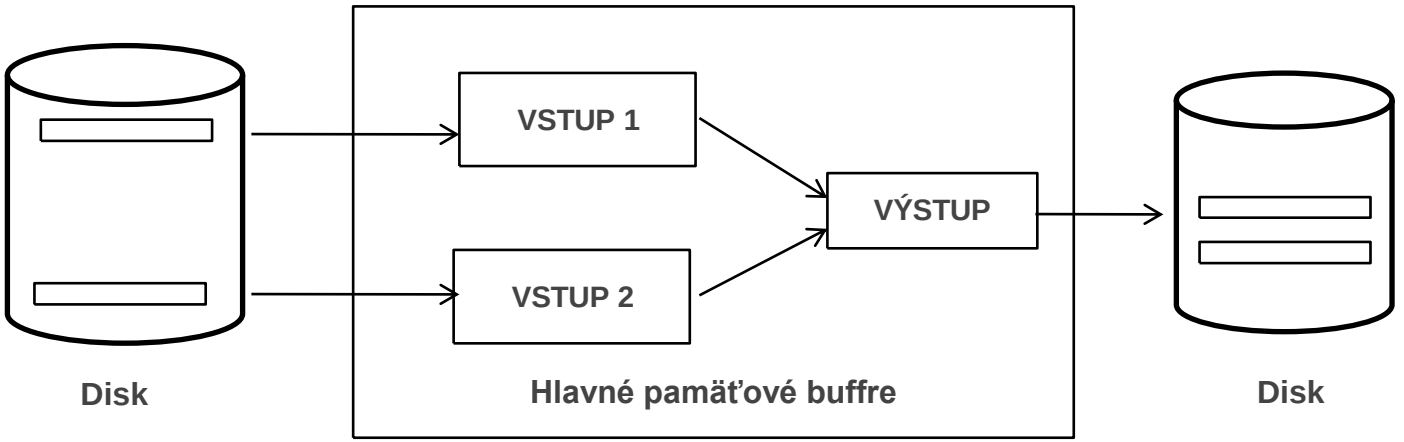
Externé triedenie

Prečo triediť?

- Klasický problém v informatike!
- Údaje zvyčajne potrebujeme mať utriedené
  - napr. vypísať študentov utriedených podľa priemeru
- Triedenie je prvý krok pri hromadnom načítaní indexu B+ stromu.
- Triedenie je užitočné pri odstraňovaní duplicít v kolekciách (Prečo?)
- **Sort-merge** spájací algoritmus zahŕňa triedenie.
- Problém: utriediť 1Gb údajov v 1Mb RAM.
  - Prečo nie vo virtuálnej pamäti?

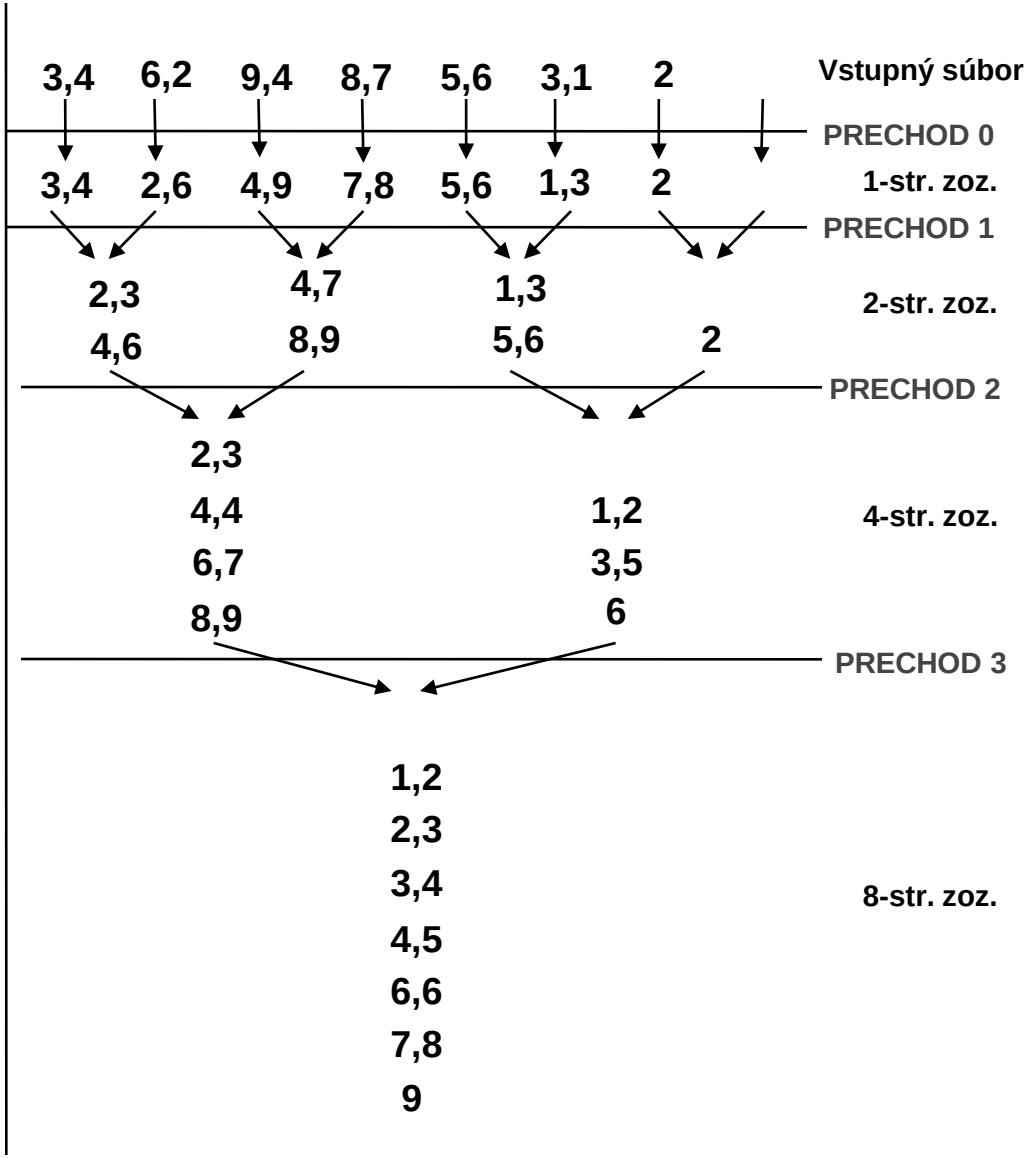
2-smerné triedenie – vyžaduje 3 buffre

- Prechod 1: Načítaj stránku, utrieď ju, zapíš ju.
- Len jeden buffer je použitý
- Prechody 2, 3, …, atď.:
- Tri buffre sú použité



2-smerný externý – Merge sort

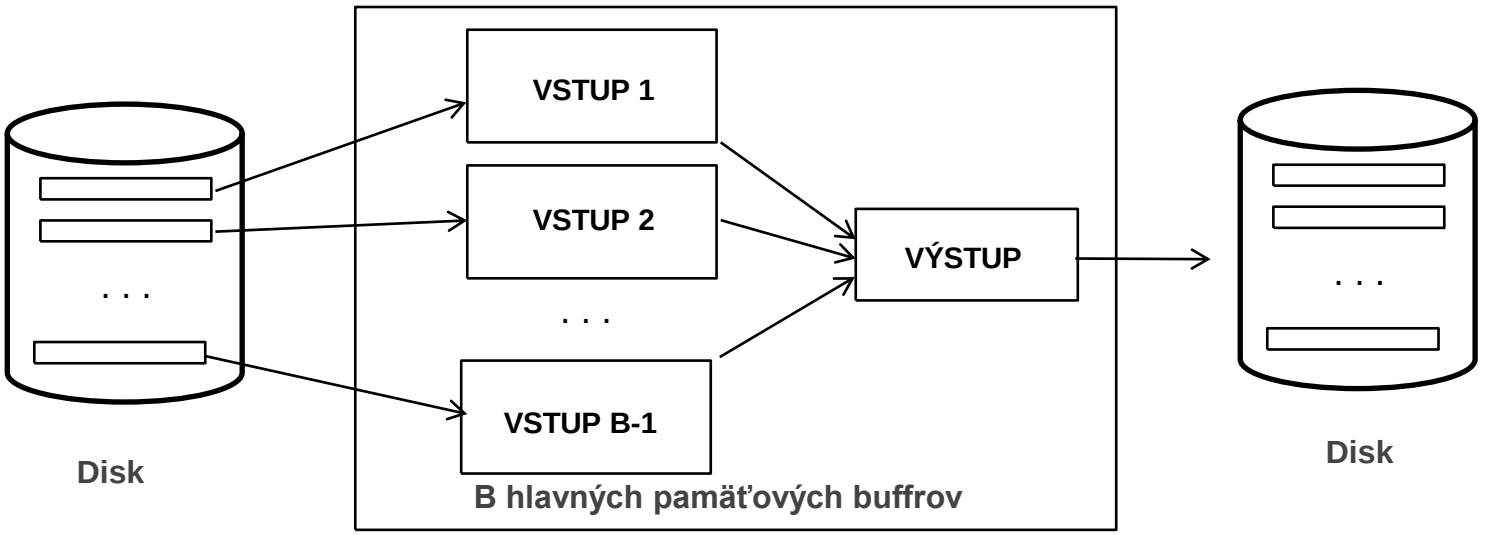
- V každom prechode čítame a zapisujeme každú stránku v súbore.
- N stránok v súbore => počet prechodov
  - $= \lceil \log_2 N \rceil + 1$
- Teda celková suma je:
  - $2N\left(\lceil \log_2 N \rceil + 1\right)$
- *<u>Myšlienka:</u>* ***Rozdeľuj a panuj:*** *utriediť podsúbory a spojiť*



Všeobecný externý Merge sort

Viac ako 3 buffrovacie stránky. Ako ich využiť?

- Utriediť súbor s N stránkami pomocou B buffrov:
- Prechod 1: použi *B* buffrovacích stránok. Čo produkuje (N/B) utriedených zoznamov s B stránkami
- Prechod 2, …, atď.: spája *B-1 zoznamov*.



Cena externého Merge sort

- Počet prechodov: $1 + \lceil \log_{B-1} \lceil \frac{N}{B} \rceil \rceil$
- Cena = 2N * (počet prechodov)
- Napr. s 5 buffrovacími stránkami, na utriedenie 108 stránok v súbore:
  - Prechod 1: $\lceil \frac{108}{5} \rceil$ = 22 utriedených zoznamov o veľkosti 5 stránok (v poslednom zozname len 3 stránky)
  - Prechod 2: $\lceil \frac{22}{4} \rceil$ =6 utr. zoznamov o veľkosti 20 stránok (v poslednom len 8 stránok)
  - Prechod 3: 2 utr. zoznamy, 80 stránok a 28 stránok v posl.
  - Prechod 4: Utriedený súbor 108 stránok

Počet prechodov externého Merge Sort

N	B = 3	B = 5	B = 9	B = 17	B = 129	B = 257
100	7	4	3	2	1	1
1,000	10	5	4	3	2	2
10,000	13	7	5	4	2	2
100,000	17	9	6	5	3	3
1,000,000	20	10	7	5	3	3
10,000,000	23	12	8	6	4	3
100,000,000	26	14	9	7	4	4
1,000,000,000	30	15	10	8	5	4

I/O pri externom Merge Sort

- Zrýchlenie vieme zabezpečiť čítaním viacerých stránok súčasne
- Teda, čítame **bloky** stránok sekvenčne!
- Naznačuje to, aby aj každý buffer (input/output) bol *blokom stránok.*
  - Ale to zredukuje vetvenie počas zlučovania prechodov!
- V skutočnosti, väčšina súborov bude utriedená v *2-3* **prechodoch**.

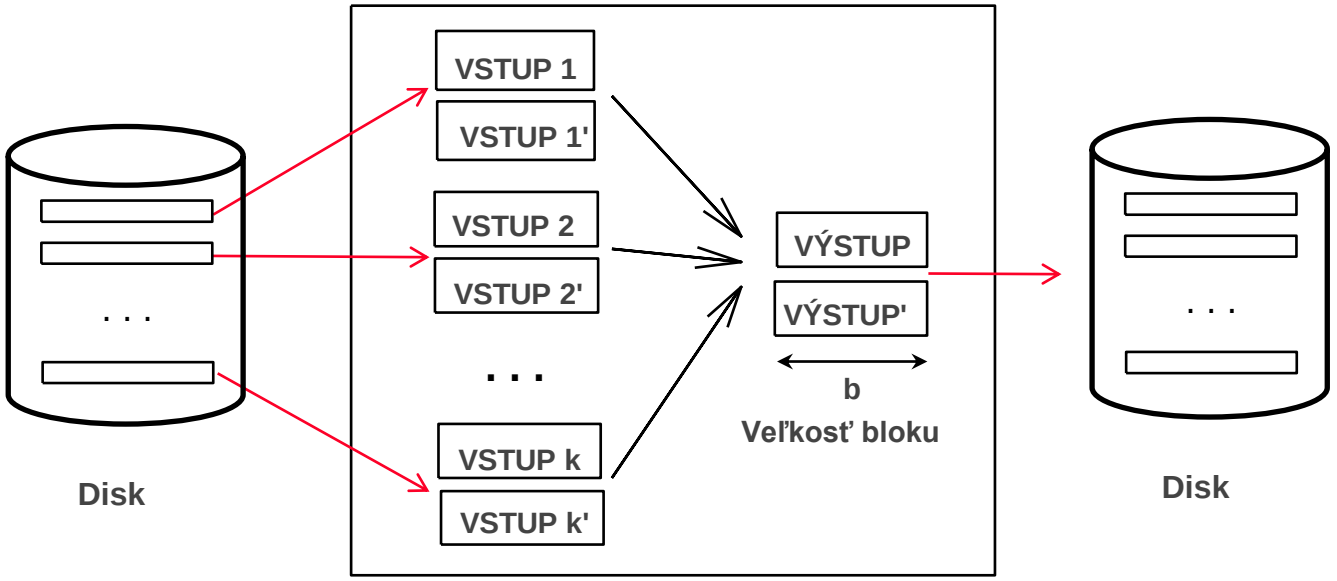
Počet prechodov po optimalizácií

N	B=1,000	B=5,000	B=10,000
100	1	1	1
1,000	1	1	1
10,000	2	2	1
100,000	3	2	2
1,000,000	3	2	2
10,000,000	4	3	3
100,000,000	5	3	3
1,000,000,000	5	4	3

Veľkosť bloku = 32

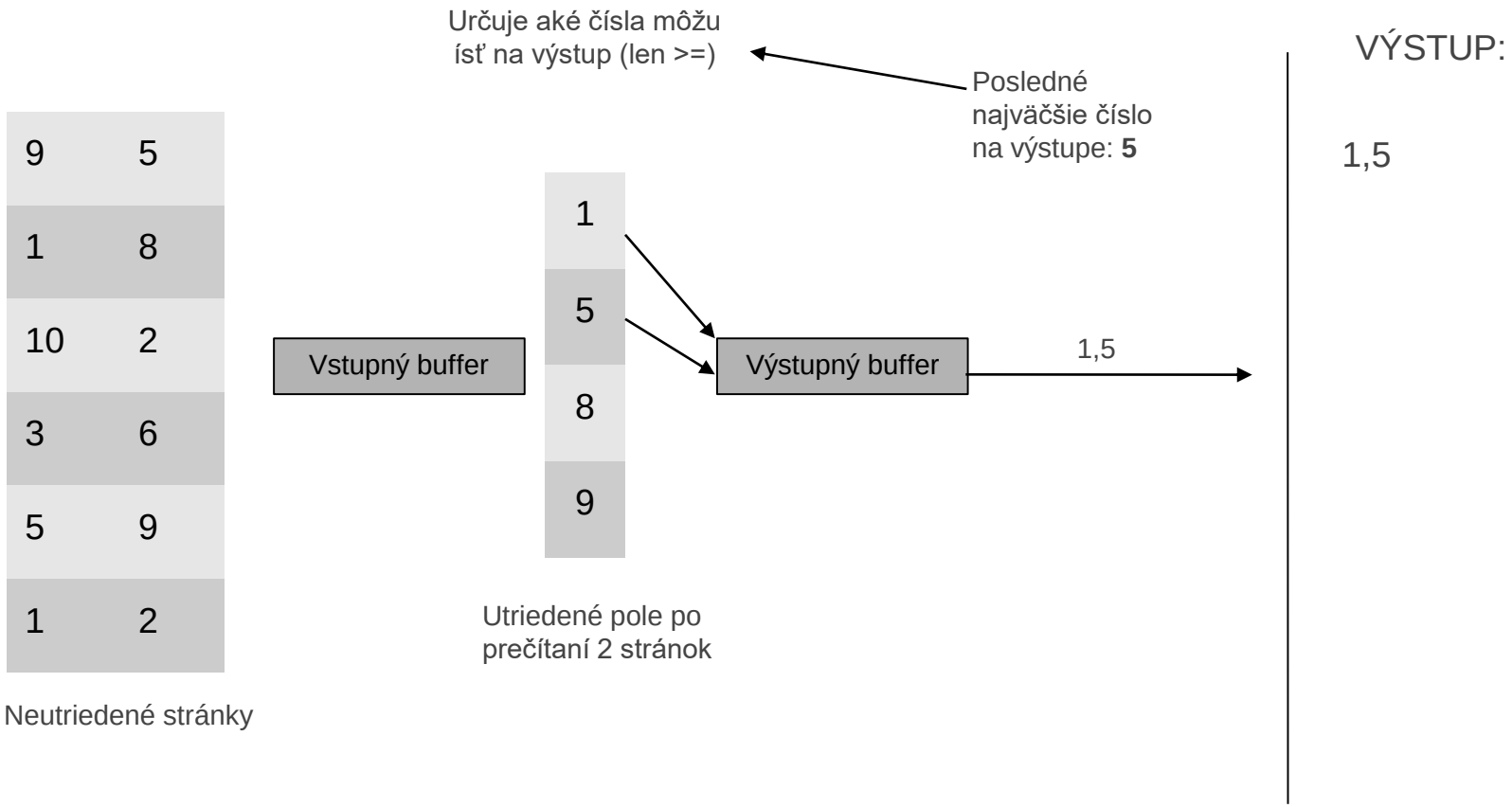
Dvojité buffrovanie

- Aby sme minimalizovali čas čakania na I/O operácie, môžeme *predpripraviť dáta* do, čakajúcich blokov.
- Pravdepodobne, viac prechodov; no v skutočnosti aj ja tak väčšina súborov utriedi v 2-3 prechodoch.

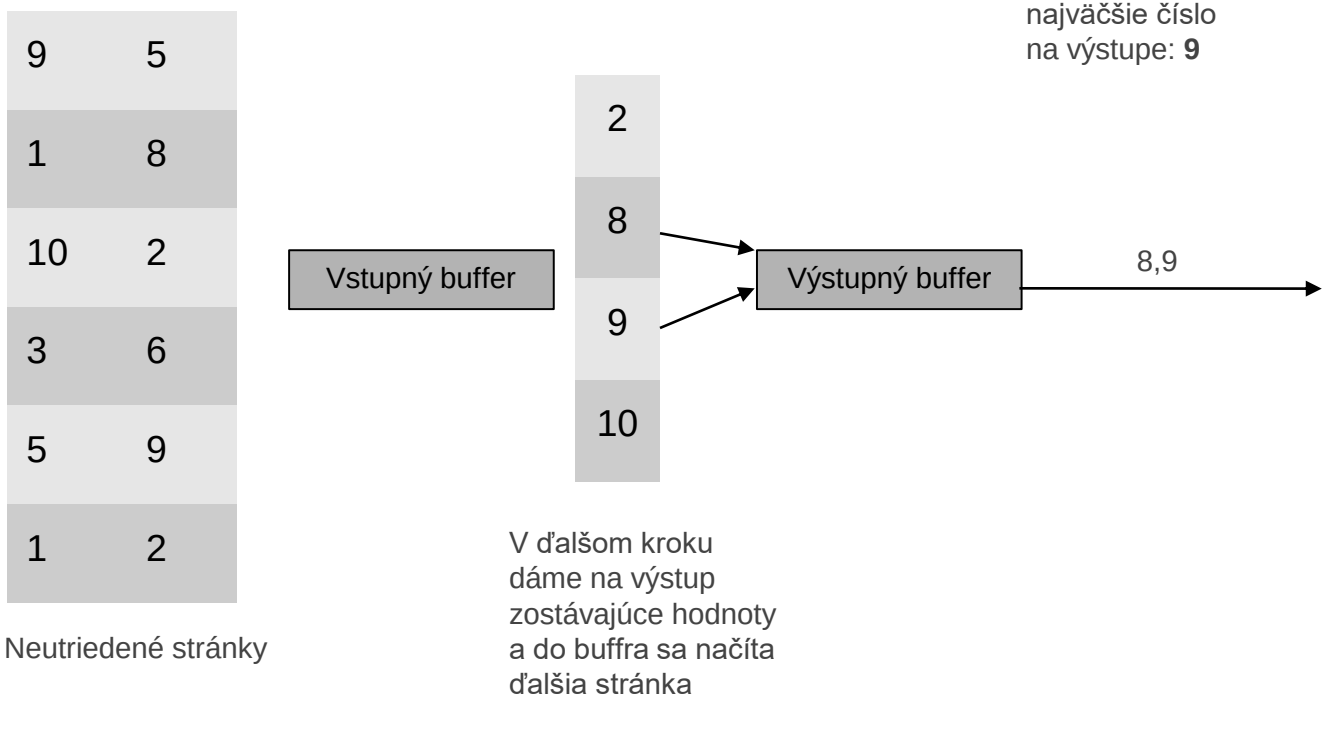


B hlavných pamäťových buffrov, k-smerné spájanie

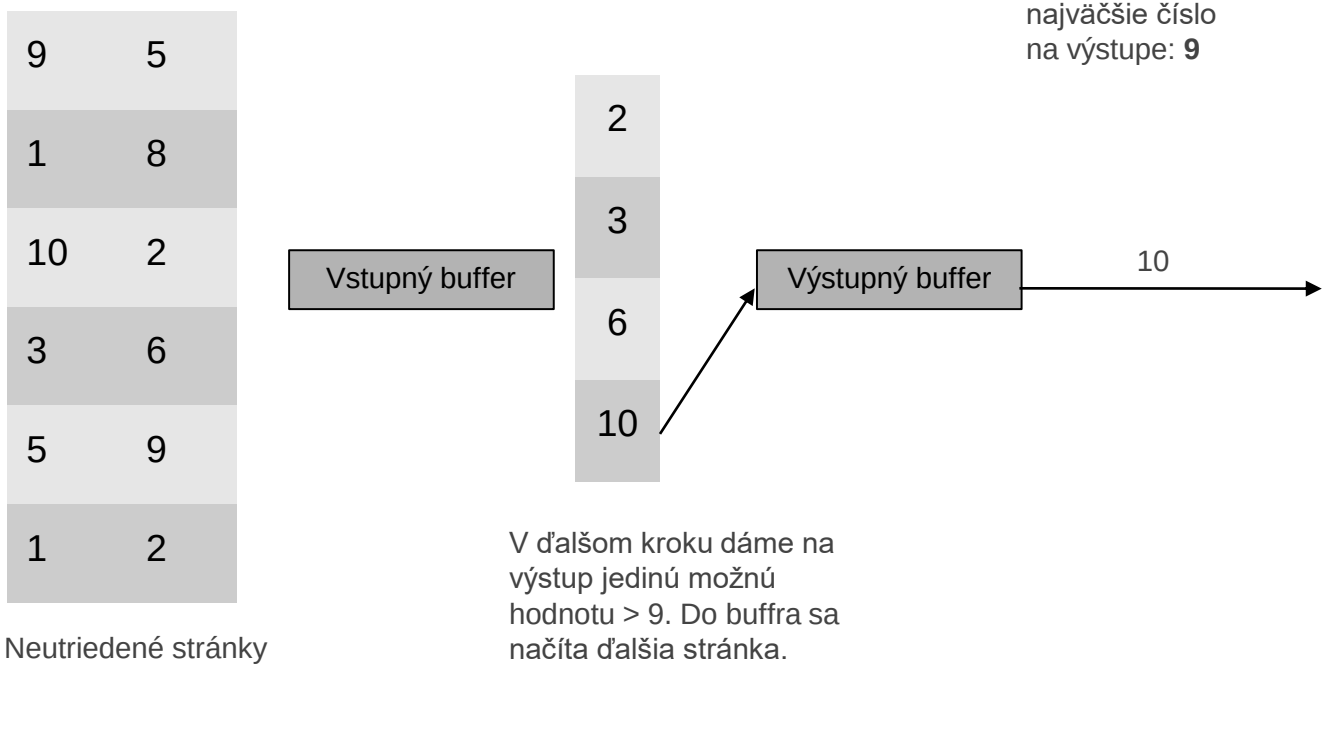
Replacement sort



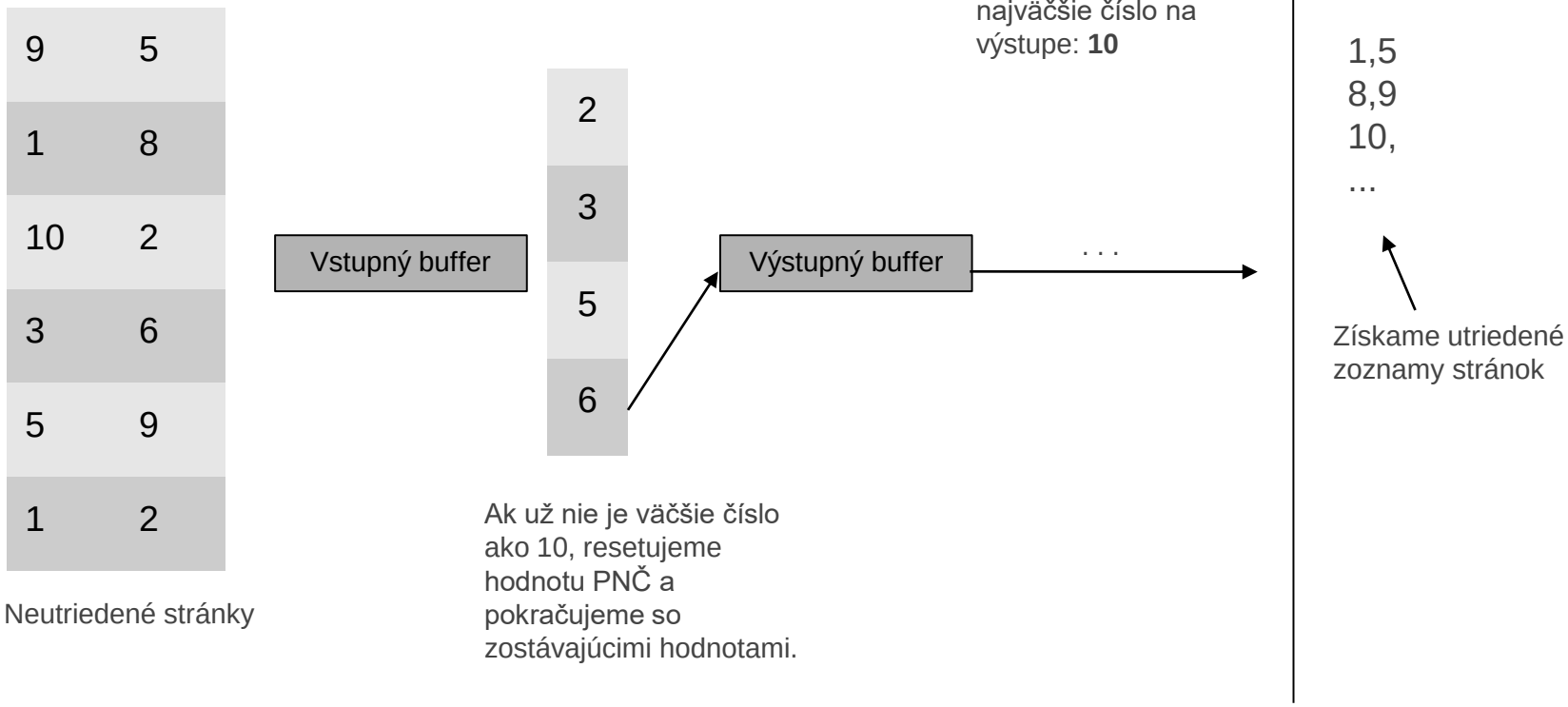
Replacement sort



Replacement sort



Replacement sort

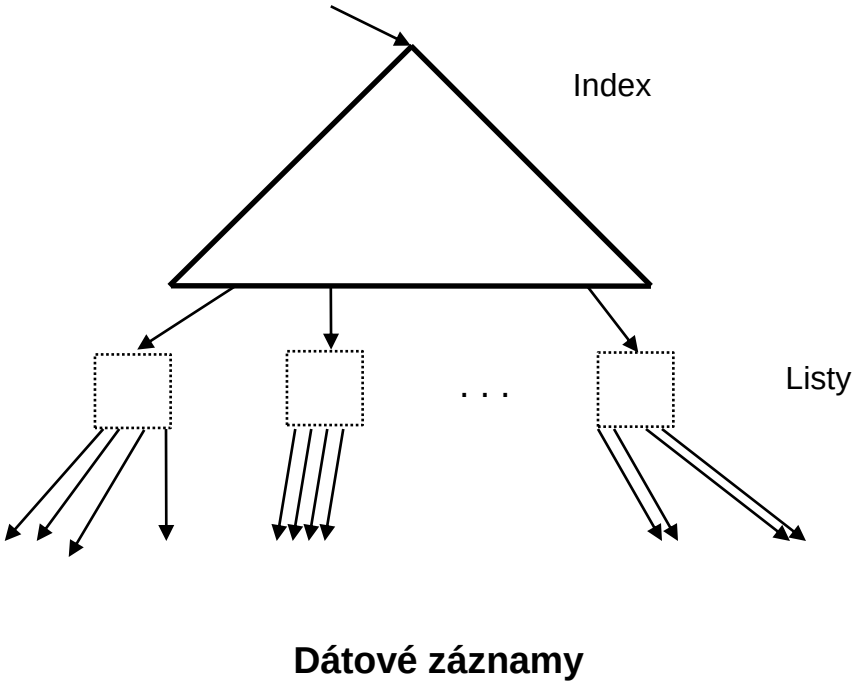


Použitie B+ stromov na triedenie

- Scenár: Tabuľka, ktorú ideme triediť má ako index B+ strom na triedených stĺpcoch.
- Myšlienka: Záznamy vieme získať v poradí prechodom po listových stránkach.
- Čo treba brať do úvahy
  - Ak B+ strom je klastrovaný **Dobrý nápad!**
  - Ak **nie je** klastrovaný Asi veľmi **zlý nápad!**

Klastrovaný B+ strom použitý na triedenie

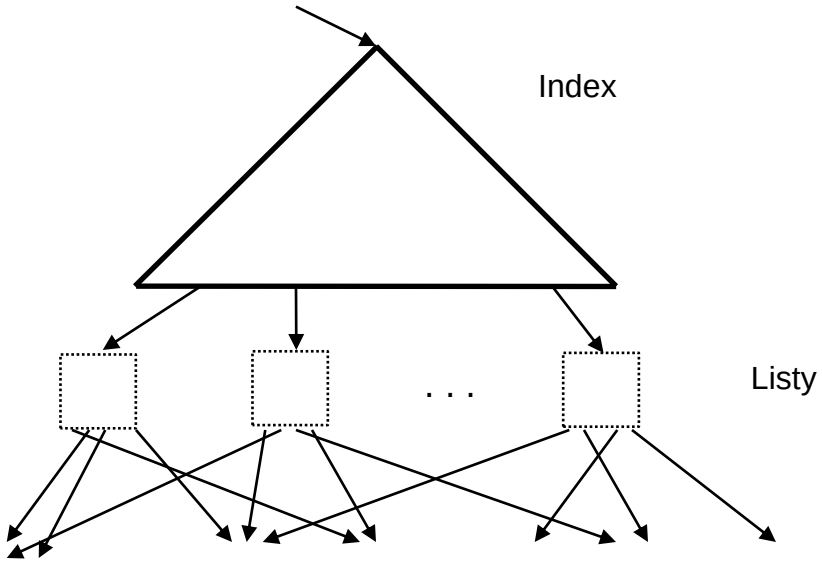
- Cena: od koreňa prejsť po najľavejší list a prechádzať po ďalších (Prvá alternatíva)
- A ak je použitá druhá alternatíva? Musíme prirátať cenu získania záznamov: každá stránka je získaná len raz.



Vždy lepšie ako externé triedenie

Neklastrovaný B+ strom použitý na triedenie

- Alternatíva (2) pre údaje; každá množina údajov obsahuje *rid* dátového záznamu. Vo všeobecnosti je to, 1 I/O na záznam!



Dátové záznamy

Externé triedenie vs. neklastrovaný index

N	Sorting	p= 1	p= 10	p= 100
100	200	100	1,000	10,000
1,000	2,000	1,000	10,000	100,000
10,000	40,000	10,000	100,000	1,000,000
100,000	600,000	100,000	1,000,000	10,000,000
1,000,000	8,000,000	1,000,000	10,000,000	100,000,000
10,000,000	80,000,000	10,000,000	100,000,000	1,000,000,000

***p*: počet záznamov na stránku**
B=1,000 a veľkosť bloku=32 pre triedenie
p=100 je najrealistickejšia hodnota.

Záver

- Externé triedenie je dôležité; DBMS môže časť buffrovacieho poolu venovať na triedenie!

- Externý merge sort minimalizuje diskové I/O operácie:
  - Prechod 1: Produkuje utriedené **zoznamy** veľkosti **B** (počet buffrovacích stránok). Neskoršie prechody: **spájajú** zoznamy.
  - Počet zoznamov spájaných v čase závisí na **B**, a **veľkosti bloku**.
  - Väčšie veľkosti blokov znamenajú menej I/O operácií na stránku.
  - Väčšie veľkosti blokov znamenajú menej prechodov, ktoré treba na spájanie zoznamov.
  - V skutočnosti, počet prechodov je zriedkavo väčší ako 2 or 3.

Záver (pokr.)

- Výber vnútorného triediaceho mechanizmu dosť zaváži:
  - Quicksort: Rýchly!
  - Heap/tournament sort: pomalý (2x), väčšie zoznamy
- Najlepšie triediace algoritmy sú šialene rýchle:
  - Aj takmer po 40 rokoch výskumu ich stále zlepšujeme
- Klastrovaný B+ strom je dobrý na získanie utriedených zoznamov; neklastrovaný je zvyčajne veľmi zlý.