



PA152: Efektivní využívání DB  
3. Ukládání dat

Vlastislav Dohnal

# Osnova

- *Ukládání dat*
- Záznamy
- Organizace bloků
- Příklady

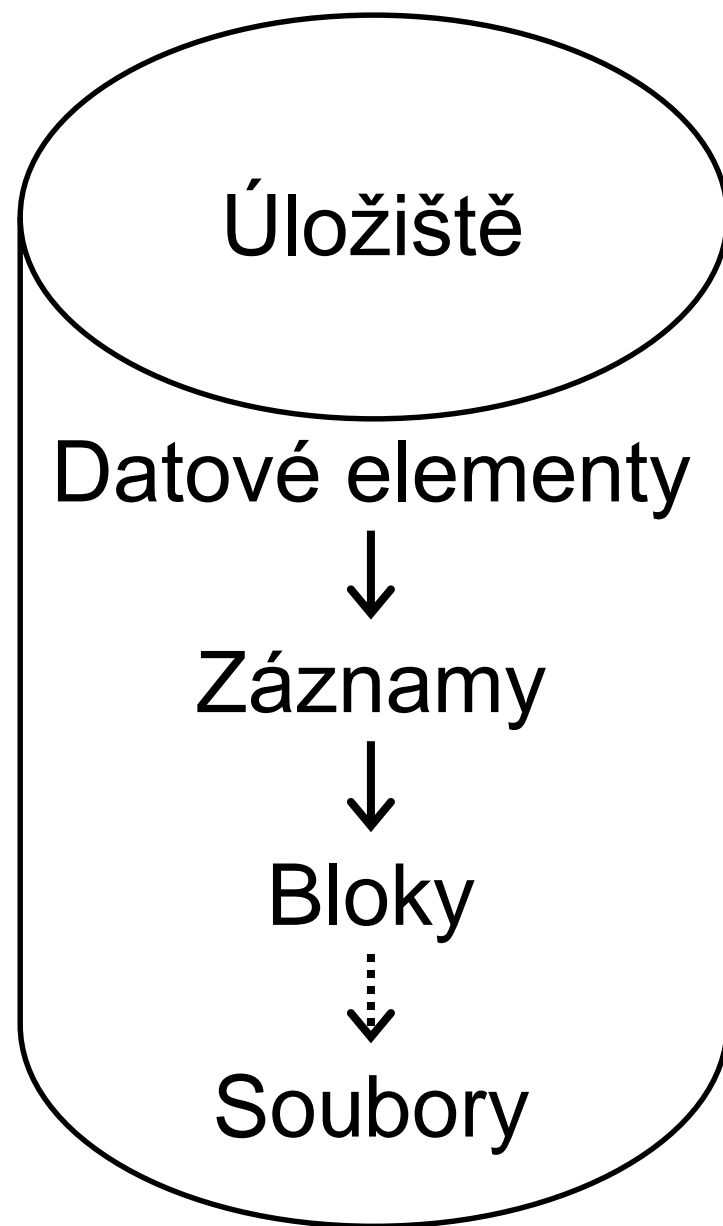
# Uložení dat

## ■ Co chceme ukládat?

- jméno
- plat
- datum
- obrázek

## ■ Jak ukládat?

- bajty → posloupnost bajtů



# Typy datových elementů

## ■ Celá čísla

- Podle rozsahu: 2, 4, 8 bajtů
- Např. 35 v 16 bitech

00000000

00100011

- Obvykle přímý kód nebo inverzní kód

## ■ Reálná čísla

- Plovoucí čárka
  - $n$  bitů rozděleno na mantisu a exponent (dle IEEE 754)
- Pevná čárka ( $number(p,s)$ )
  - Kódování každých 9 cifer (základ 10) do 4 bajtů
  - Uložení jako řetězec

# Typy datových elementů

## ■ Znaky

- Nejčastěji v ASCII kódování – 1 bajt

- Více bajtové znaky

- UCS-2 (UTF-16) – kódování UTF-8 do 16 bitů

- Znaky s kódy 0 až 65535

- UTF-8 – kódování variabilní délky

- Znak může zabírat 1 až 4 bajty

- Původně až 6, ale omezeno na stejný rozsah kódů jako UTF-16.

- Kódování:

11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

celkový počet bytů kódu

# Typy datových elementů

- Pravdivostní hodnota (boolean)
  - Obvykle jako celé číslo
  - True 

|           |
|-----------|
| 1111 1111 |
|-----------|
  - False 

|           |
|-----------|
| 0000 0000 |
|-----------|
  - Méně než 1 bajt nemá velký význam
- Bitové pole
  - Délka + bity
    - Tj. zaokrouhleno na celé bajty

# Typy datových elementů

## ■ Datum

- počet dní od „počátku“ (např. 1.1.1970)
- komprese vzorcem  $DD + MM \times 32 + YYYY \times 16 \times 32$
- řetězec YYYYMMDD (8 znaků)
  - YYYYDDD (7 znaků)
  - Proč ne YYMMDD?

## ■ Čas

- počet sekund od půlnoci
  - počet milisekund nebo mikrosekund
  - or as a packed 3-byte integer  
 $DD \times 24 \times 3600 + HH \times 3600 + MM \times 60 + SS$
- fractions of second
  - As string HHMMSSFF or as above with fractional part separately (up to 3 bytes for 6 digits)
- časové zóny – čas uložen v UTC
  - Ukládání – konverze z daného pásma do UTC

# Typy datových elementů

## ■ Datetime

- Combining date and time

- Year\*13+month; hour, min, sec (5-6bit each)

- 5 bytes + fractional part

## ■ Timestamp

- Seconds since epoch (midnight Jan 1, 1970 UTC)

## ■ Výčtový typ

- Očíslování hodnot → integer

- red → 1, green → 2, blue → 3, yellow → 4, ...



# Typy datových elementů

## ■ Řetězce

### □ Pevná délka

#### ■ Omezení velikosti

- Kratší řetězce doplněny mezerami
- Delší oříznuty

### □ Proměnlivá délka

#### ■ Uložená délka, pak řetězec

#### ■ Ukončení nulou

- nutnost číst celý
- nelze uložit nulu v textu

### □ Problém znakové sady (kódování)

# Uložení datových elementů

- Každý element „má“ svůj typ
  - interpretace bitů
  - velikost
  - speciální hodnota „neznámá“ (NULL)
- Většinou pevná délka
  - každý typ má svoji bitovou reprezentaci
- Proměnlivá délka
  - velikost na začátku hodnoty

# Osnova

- Ukládání dat
- *Záznamy*
- Organizace bloků
- Příklady

# Záznam (record)

- Seznam souvisejících datových elementů
  - Resp. jejich hodnot
  - Fields, Attributes
- Např.
  - Zaměstnanec
    - jméno – Novák
    - plat – 1234
    - datum\_přijetí – 1.1.2000

# Schéma záznamu

- Popisuje strukturu záznamu
- Uložené informace
  - Počet atributů
  - Pořadí atributů
  - Typ / název každého atributu

# Typy záznamů podle schématu

## ■ Pevné schéma

- Schéma společné pro všechny záznamy

- Je uloženo mimo záznamy (tzv. data dictionary)

## ■ Proměnlivé schéma

- Každý záznam obsahuje svoje schéma

- Vhodné pro:

- „řídké“ záznamy (hodně NULL)

- opakování stejných atributů

- vyvíjející se formát

- změny schématu během života db

# Typy záznamů podle délky

## ■ Pevná délka

- Každý záznam má stejnou délku (počet bajtů)

## ■ Proměnlivá délka

- Ušetření paměti
- Složitější implementace
- Možnost pro uložení velkých dat (obrázky, ...)

# Příklad – pevné schéma i délka

## ■ Zaměstnanec

- 1) id – 2 byte integer
- 2) jméno – 10 znaků
- 3) oddělení – 2 byte code

} schéma

|    |             |    |
|----|-------------|----|
| 55 | n o v á k   | 02 |
| 83 | d l o u h ý | 01 |

} záznamy

## ■ Zarovnání na „vhodnou“ délku

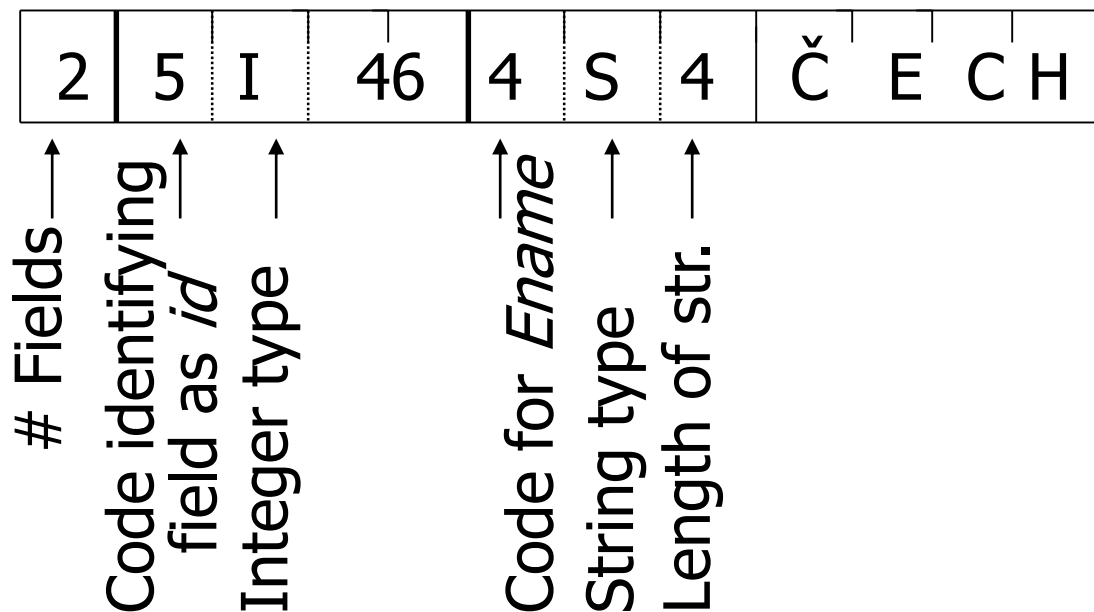
- Rychlejší přístup k paměti zarovnané na 4 (8) bajtů

|    |           |    |     |
|----|-----------|----|-----|
| 55 | n o v á k | 02 | - - |
|----|-----------|----|-----|



# Příklad – proměnlivé schéma i délka

## ■ Zaměstnanec:



Kódy identifikující názvy atributů mohou být přímo textové řetězce, tzv. tagy.

□ Nazýváno „Tagged fields“

# Příklad – opakující se atribut

- Zaměstnanec má děti

|   |                  |             |             |
|---|------------------|-------------|-------------|
| 3 | Jméno: Jan Novák | Dítě: Tomáš | Dítě: Pavel |
|---|------------------|-------------|-------------|

- Vhodné v případě polí (arrays) atp.

- Opakování atributu neznamená proměnlivou délku ani schéma

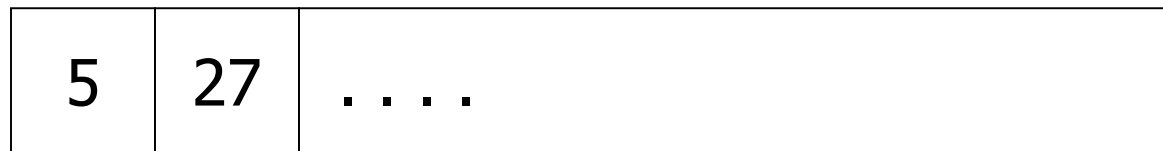
- Lze určit maximální počet opakování

- Nevyužitě místo vyplnit NULL

|       |          |       |    |
|-------|----------|-------|----|
| Novák | Potápění | Šachy | -- |
|-------|----------|-------|----|

# „Mezivarianta“

- Mezi pevným a variabilním schématem
- Přidání „schéma záznamu“ do hlavičky záznamu



↑  
record type  
tells me what  
to expect  
(i.e. points to schema)

record length

# Hlavička záznamu

- = informace o záznamu (fixní délka)  
(nesouvisející s hodnotami atributů)
  - Schéma záznamu (odkaz)
  - Délka záznamu
  - Čas vytvoření / změny / čtení záznamu
  - OID (Object Identifier) – „ID“ záznamu
  - Pole pro NULL hodnoty
    - Jeden bit pro každý atribut
  - ...

# Další problémy

## ■ Komprese

- Zvýšení rychlosti (méně čtu)
- Uvnitř záznamu (hodnoty zvlášť)
- Více záznamů
  - Efektivnější (lze vytvořit slovník)
  - Složitější

## ■ Kódování (šifrování)

- Co potom s indexy?
- Jak řešit rozsahové dotazy?
- ...

# Uložení objektů

- Současné komerční DB podporují objekty
  - Rozšíření relačních DBMS
  - OODBMS
- Objekt má atributy
  - Jednoduché typy → uložit jako záznam
  - Kolekce → vytvořit novou relaci a tam uložit
    - Prvky pak referencovat pomocí jejich OID

# Uložení relace

- Řádkové

- Zatím uvažovaná varianta

- Sloupcové

- Hodnoty stejného atributu pohromadě

- Příklad řádkového uložení:

- Order(id, cust, prod, store, price, date, qty)

|     |       |       |        |        |       |      |
|-----|-------|-------|--------|--------|-------|------|
| id1 | cust1 | prod1 | store1 | price1 | date1 | qty1 |
| id2 | cust2 | prod2 | store2 | price2 | date2 | qty2 |
| id3 | cust3 | prod3 | store3 | price3 | date3 | qty3 |

# Sloupcové uložení

## ■ Relace

- Order(id, cust, prod, store, price, date, qty)

|     |       |
|-----|-------|
| id1 | cust1 |
| id2 | cust2 |
| id3 | cust3 |
| id4 | cust4 |
| ... | ...   |
|     |       |

|     |       |
|-----|-------|
| id1 | prod1 |
| id2 | prod2 |
| id3 | prod3 |
| id4 | prod4 |
| ... | ...   |
|     |       |

...

Id mohou ale nemusí být uloženy,  
lze využít pořadí hodnot



# Porovnání

## ■ Výhody sloupcového uložení

- Kompaktnější uložení (nezarovnávání na bajty, komprese, ...)
- Efektivní čtení (např. pro data mining)
  - Zpracovávání mála atributů, ale všech hodnot

## ■ Výhody řádkového uložení

- Aktualizace / vkládání je rychlejší
- Efektivnější při přístupu k celým záznamům

Mike Stonebraker, Elizabeth O'Neil, Pat O'Neil, Xuedong Chen, et al.:  
*C-Store: A Column-oriented DBMS*, VLDB Conference, 2005.  
[http://www.cs.umb.edu/~poneil/vldb05\\_cstore.pdf](http://www.cs.umb.edu/~poneil/vldb05_cstore.pdf)

# Osnova

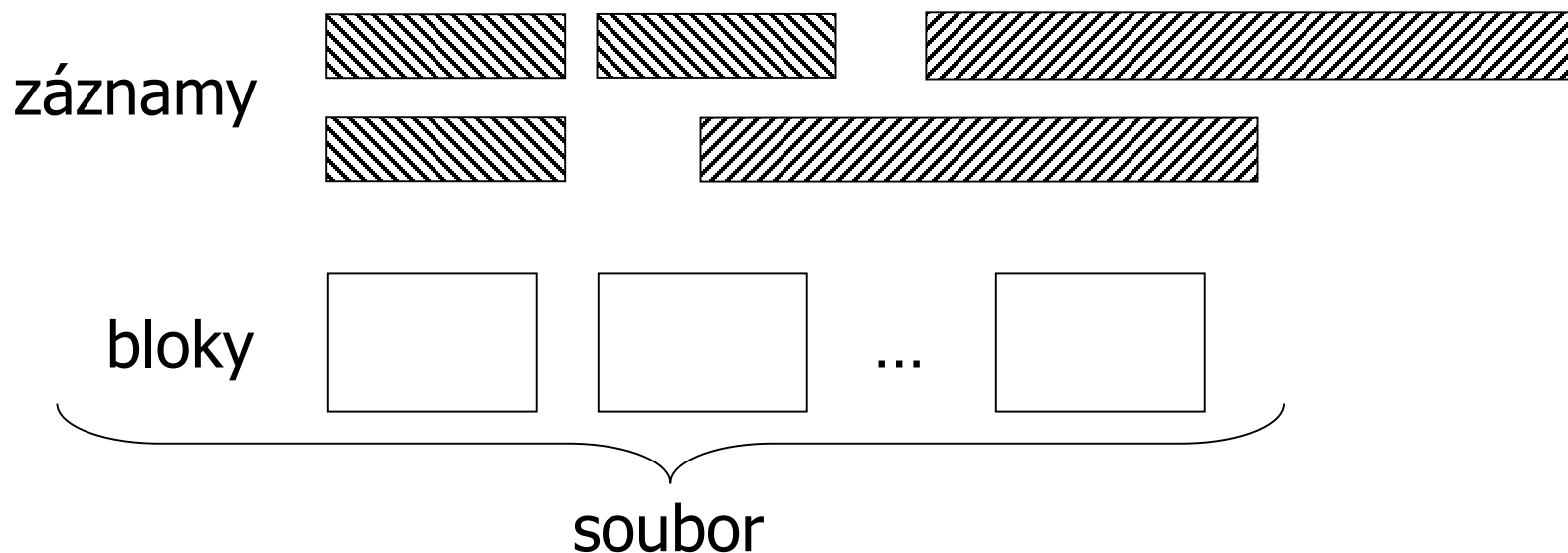
- Ukládání dat
- Záznamy
- *Organizace bloků*
- Příklady

# Uložení záznamů do bloků

## ■ Záznamy

- Pevné délky
- Proměnné délky

## ■ Bloky pevné velikosti



# Uložení záznamů do bloků

## ■ Problémy k řešení:

### □ Oddělování záznamů

- Separating records

### □ Rozdělování / nerozdělování záznamů

- Spanned vs. unspanned

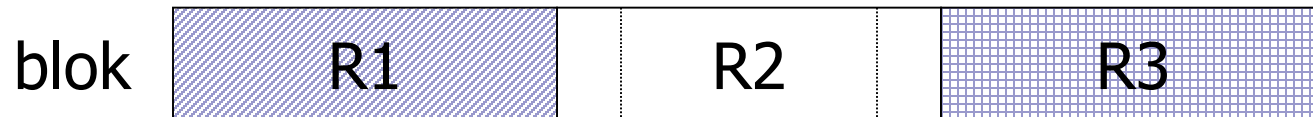
### □ Uspořádání záznamů

- Sequencing

### □ Odkazy na záznamy

- Indirection

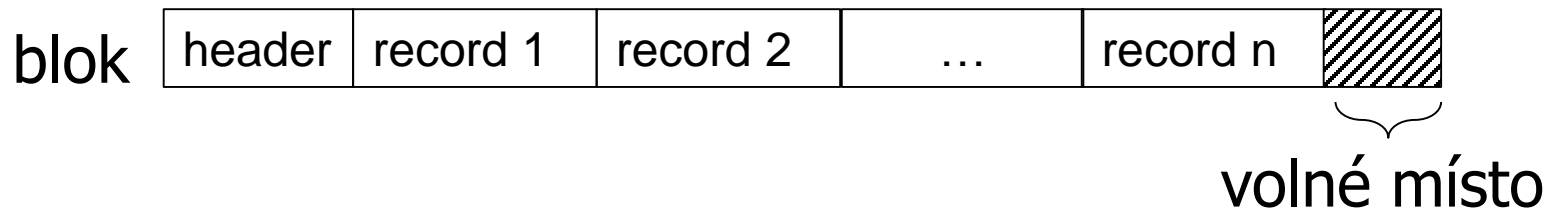
# Oddělování záznamů



- Záznamy pevné délky
  - Žádný oddělovač
  - Pamatovat počet a ukazatel na první záznam
- Záznamy proměnné délky
  - Oddělovač
  - Ukládání délek záznamů (nebo počátků)
    - V rámci záznamu
    - V hlavičce bloku

# Oddělování záznamů (pokr.)

- Záznamy proměnné délky:
- Blok má hlavičku, pak záznamy

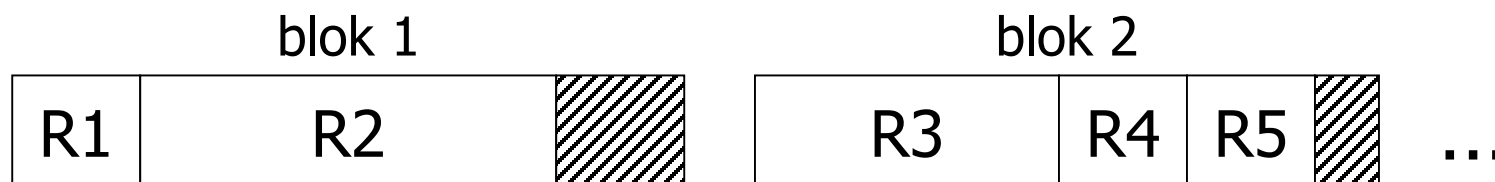


- Hlavička
  - Odkaz na další bloky (přetokové, indexu, ...)
  - Typ bloku (přetokový, index, ...)
  - Příslušnost relaci
  - (Adresář offsetů na záznamy)
  - Časová razítka (vytvoření, modifikace, čtení)

# Rozdělování záznamů

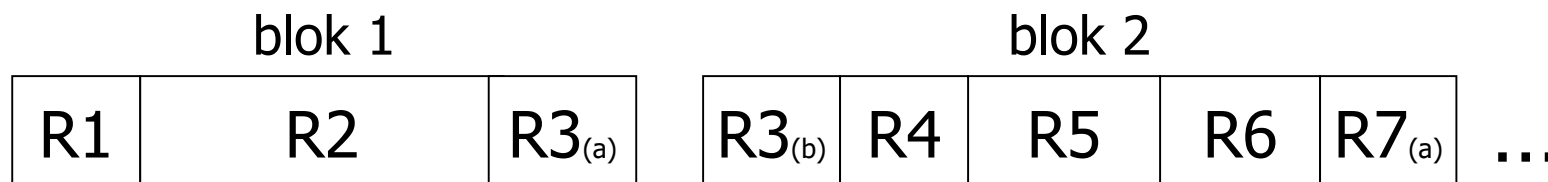
## ■ Nerozdělování

- každý záznam součástí jednoho bloku
- jednodušší, ale může plýtvat místem



## ■ Rozdělování

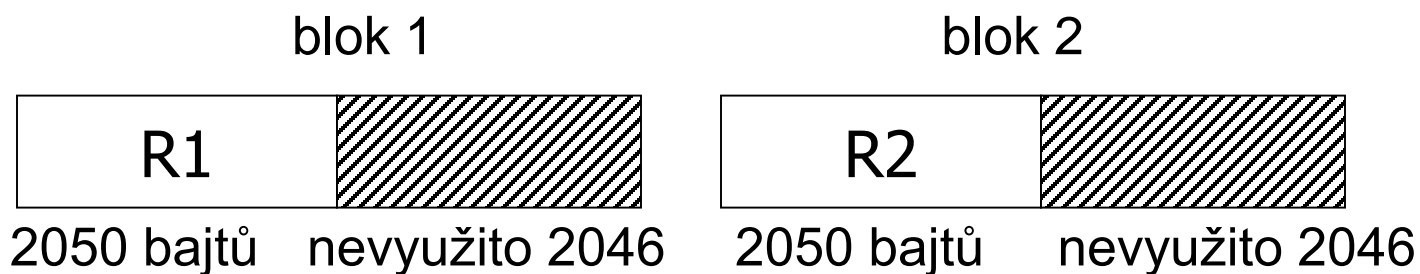
- Záznam „přetéká“ mezi bloky
- Nutné, pokud je záznam větší než blok!



# Rozdělování záznamů: příklad

## ■ Nerozdělování záznamů

- $10^6$  záznamů, každý 2 050 bajtů (pevná délka)
- Velikost bloku 4 096 bajtů

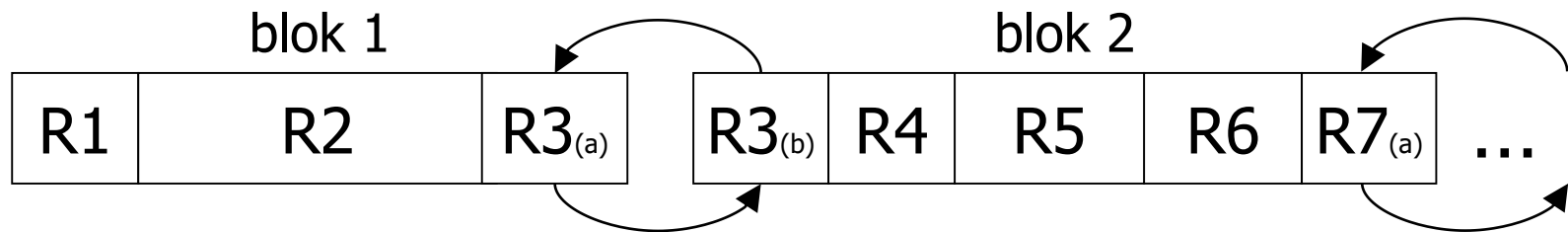


- Celkem alokováno:  $10^6 * 4096B$
- Celkem využito:  $10^6 * 2050B$
- Využití paměti: 50,05%



# Rozdělování záznamů

- Záznam „přetéká“ mezi bloky
  - Musíme udržovat pořadí bloků
    - Lze používat ukazatele



- Záznam je rozdělen na „fragmenty“
  - Bitový příznak, zda je fragmentován
  - Ukazatel na další / předchozí fragment

# Rozdělování záznamů

## ■ Velký atribut

### □ The Oversized-Attribute Storage Technique

#### ■ TOAST or *"the best thing since sliced bread"*\*\*

\*\* [cit. dokumentace PostgreSQL]

#### ■ Rozdělení dlouhých hodnot atributu

- Vzniká tzv. TOAST table (chunk\_id, chunk\_seq, value)
- Hodnota rozdělena na kousky („chunks“)
  - Chunks tvoří fyzické záznamy v TOAST table
  - Chunk je identifikován: chunk\_id, chunk\_seq
- Original space is used to store length of the value, toast table id and chunk id.

### □ Komprese

### □ Rozdělení do více fyzických záznamů (interně)

# Rozdělování záznamů

- Velká data (LOB – Large OBjects)
  - Bez ohledu na typ: binární i textová
  - Uloženo zvlášť
    - posloupnost bloků (jiného souboru)
  - DB neindexuje, neumí uvnitř vyhledávat

# Uspořádání záznamů

- Záznamy jsou v souboru (a bloku) uspořádány
  - Podle hodnoty nějakého klíče
  - Tj. sekvenční soubor
- Důvod:
  - Efektivní čtení záznamů v daném pořadí
  - Např. pro merge-join, order by, ...

# Uspořádání záznamů

- Uložené za sebou



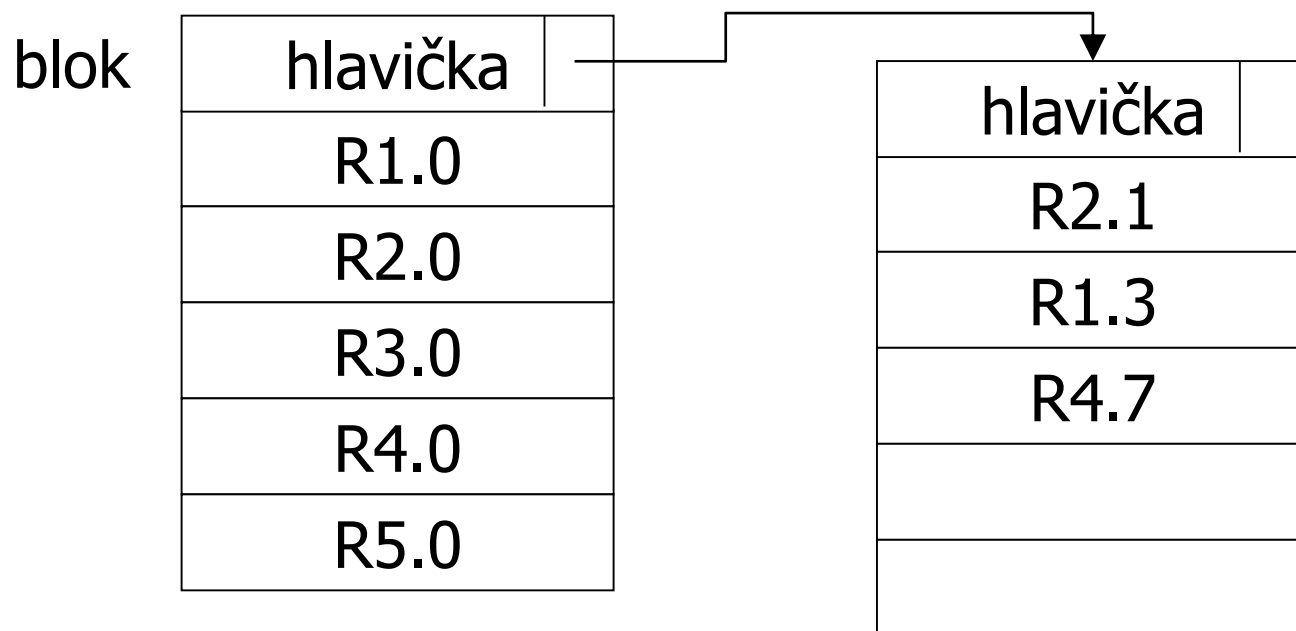
- Zřetězený seznam



# Uspořádání záznamů

## ■ Přetoková oblast

- Záznamy jsou uložené za sebou
  - nutné reorganizace při aktualizaci
- Odkaz na přetokovou oblast / blok



# Prokládání (shlukování) relací

- Záznamy různých relací v jednom bloku
  - Některá data jsou vyžadována současně
  - Uložit je společně → zrychlené čtení
  - Složitější implementace

# Prokládání (shlukování) relací: příklad

- Relace: zákazník (zid, jméno, adresa)  
vklady (zid, vkl\_id, výše\_vkladu)

- Vhodné pro dotaz Q1:

- SELECT jméno, adresa, výše\_vkladu  
FROM vklady, zákazník  
WHERE vklady.zid = zákazník.zid AND zákazník.zid = 2354

block

(2354, 'Petr Malý', 'Brno')

(2354, 999001, 100)

(2354, 999010, 1500)

⋮

⋮



# Prokládání (shlukování) relací: příklad

- Dotaz Q2:
  - `SELECT * FROM zákazník`
- Shlukování nevhodné pro Q2
  - Záleží na četnosti jednotlivých dotazů

# Prokládání (shlukování) relací

## ■ Řešení:

- Nemíchat v rámci bloku
- Ukládat bloky blízko sebe (stejný válec disku,...)

bloky

(2354, 'Petr Malý', 'Brno')

(2356, 'Petr Novák', 'Brno')

⋮

(2354, 999001, 100)

(2354, 999010, 1500)

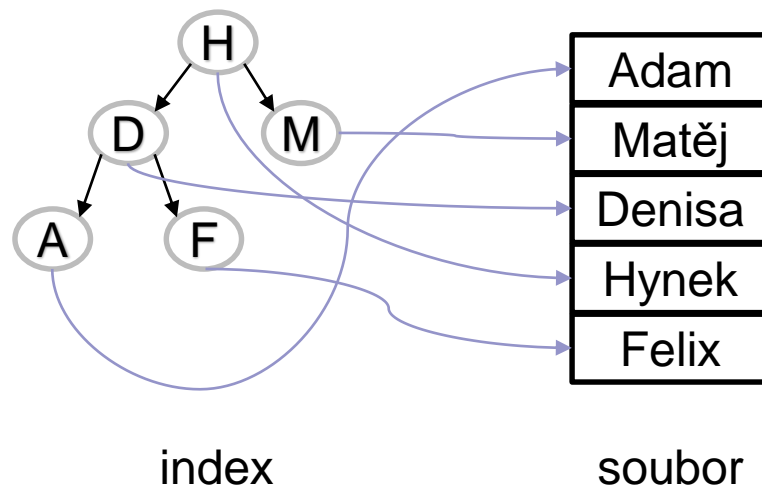
(2356, 924013, 5500)

⋮

# Odkazy na záznamy

## ■ Použití

- Rozdělování záznamů
- Odkazování bloků / záznamů (viz indexy)
- Zřetězení bloků (viz indexy)
- OODBMS: objekty ukazují na jiné objekty



# Odkazy na záznamy

## ■ Adresa záznamu

### □ V paměti (memory address)

- přímá adresace
- 4bajtový / 8bajtový ukazatel ve virtuální paměti procesu

### □ V úložišti (db address)

- sekvence bajtů popisující umístění
- přímá vs. nepřímá adresace

# Odkazy na záznamy

## ■ Přímá adresace

### □ Fyzická adresa záznamu

#### ■ Adresa v úložišti

- ID disku, stopa, povrch, blok, offset v bloku

### □ Nepraktické

- Např. realokace bloku nebo záznamu

# Odkazy na záznamy

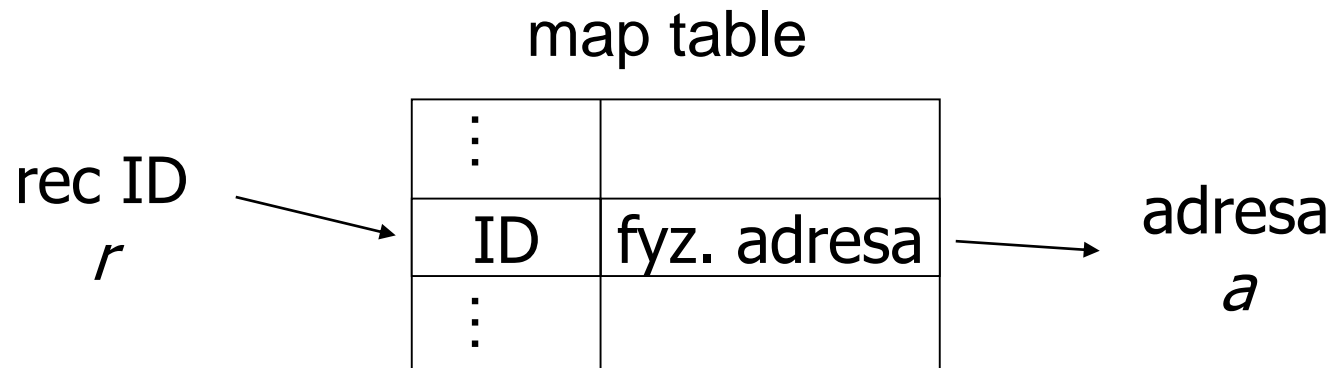
## ■ Nepřímá adresace

□ Záznam (i blok) je identifikován svým ID

□ ID = logická adresa

■ libovolná posloupnost bitů

□ Převodní tabulka (map table): ID → fyz. adresa



# Odkazy na záznamy

## ■ Nepřímá adresace

### □ Nevýhoda

#### ■ Zvýšené náklady

- Průchod map table
- Uložení map table

### □ Výhoda

#### ■ Velká flexibilita

- Mazání záznamů, vkládání
- Optimalizace uložení bloků

# Odkazy na záznamy

## ■ Vhodná varianta = **kombinace**

□ Fyz. adresa záznamu =

fyz. adresa bloku + offset

■ Offset je pořadí záznamu v bloku

□ V hlavičce je obvykle seznam odkazů na záznamy.

□ Výhody

■ Lze přesouvat záznamy v bloku

□ Beze změny fyz. adresy

■ Lze přesunout záznam do jiného bloku

□ V původním místě udělám odkaz na nový blok + offset

■ Lze zrušit map table

□ Nevýhoda

■ Nízká flexibilita přesouvání bloků (defragmentace)



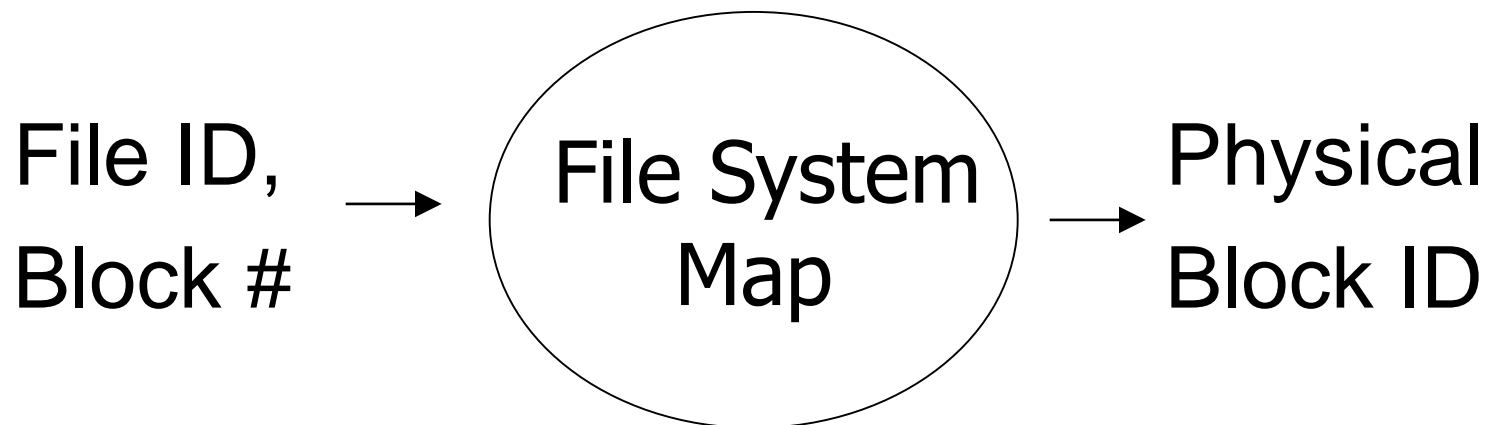
# Odkazy na záznamy

## ■ Používaná varianta

□ Adresa záznamu =

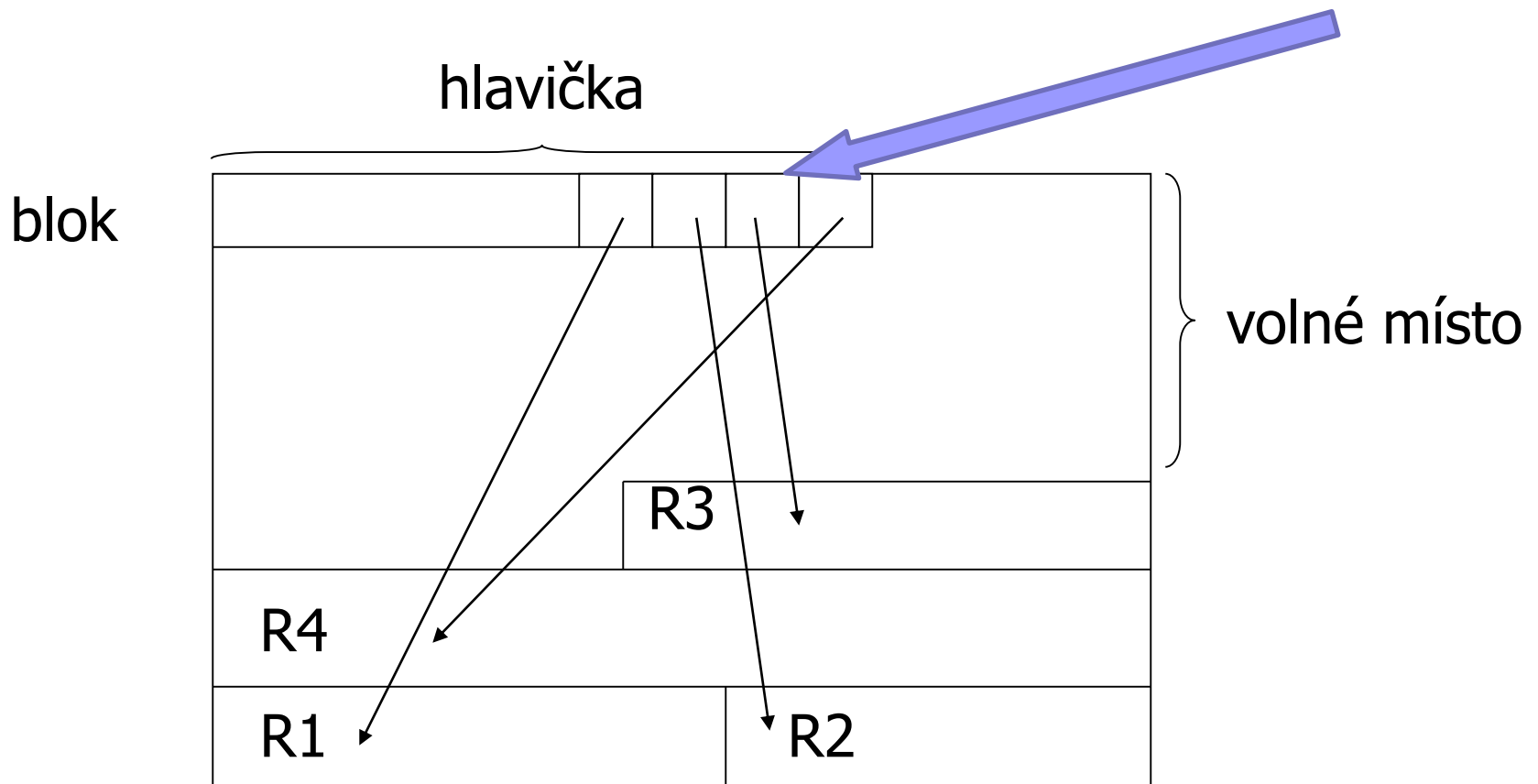
    ID souboru + číslo bloku + offset v bloku

□ Uložení bloku určuje systém souborů (file system)



# Odkazy na záznamy

- Nepřímý odkaz v bloku (Slotted Page Structure)
  - Fyz. adresa záznamu = fyz. adr. bloku + **offset**



# Hlavička bloku

- Přítomná v každém bloku
  - File ID (or RELATION ID or DB ID)
  - Typ bloku
    - např. záznamy typu 4, přetoková oblast, TOAST záznamy, ...
  - ID bloku (tohoto)
  - Adresář záznamů (odkazy na data záznamů)
  - Ukazatel na volné místo (začátek, konec)
  - Ukazatel na další blok (např. pro indexy)
  - Čas modifikace (popř. verze)

# Modifikace záznamů v bloku

## ■ Vkládání

- Obvykle snadné

## ■ Mazání

- Správa volného místa

## ■ Změna

- Stejná velikost

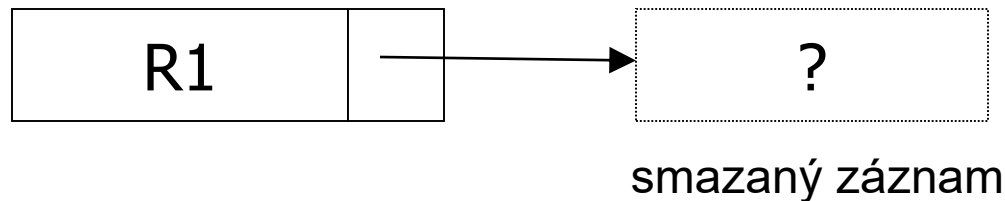
- Ok

- Jiná velikost

- Problém stejný jako při vkládání / mazání

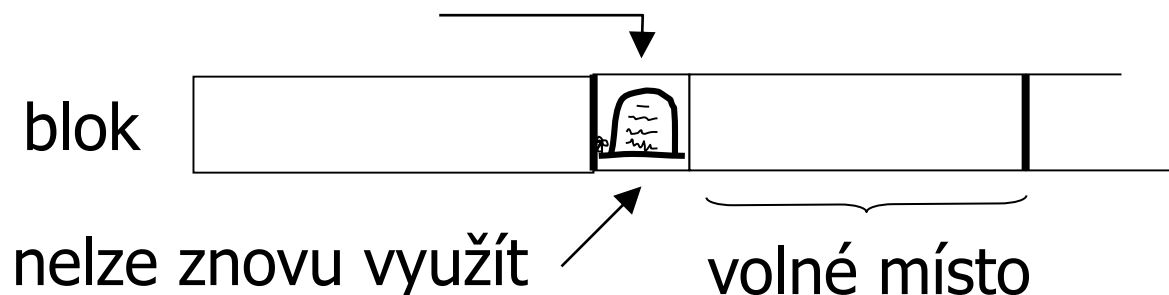
# Mazání záznamů

- Problém s odkazy na smazané záznamy
  - Musí být neplatné
  - Nesmí odkazovat na jiná nová data
  - Tzv. *dangling pointers*



# Mazání záznamů

## ■ Přímá adresa záznamu (fyzická adresa)



### □ Označ jako smazané

#### ■ Vytvořením značky (tombstone)

#### ■ Stačí 1 bit

□ Implementace: obvykle několik bajtů (zarovnání)

### □ Oznámit volné místo


#### ■ Zřetězení volných míst

# Mazání záznamů

## ■ Nepřímá adresace

- *Map table*
- Smazaný záznam uvolní místo v bloku
- Náhrobek je v map table
  - nebo „mapování“ smažu

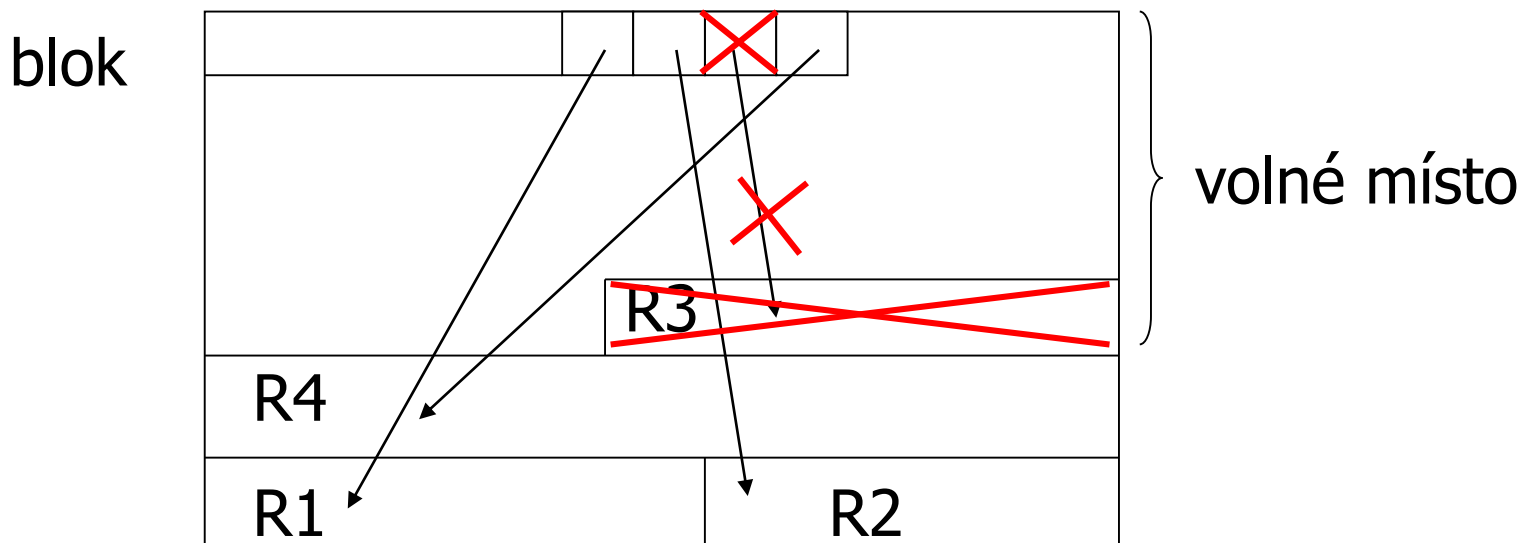
Map Table

| ID   | LOC   |
|------|---|
| ⋮    |   |
| 7788 |  |
| ⋮    |   |

← Tuto položku nelze již znovu využít.

# Mazání záznamů

- Adresa záznamu je adr. bloku + offset
  - Ihned uvolni místo
  - Defragmentuj ostatní záznamy
    - volné místo je souvislé
  - V adresáři záznamů nastav ukazatel na ***null***





# Mazání záznamů

- Uložení ID záznamu přímo v záznamu
  - Při čtení záznamu kontrolujeme ID na shodu

# Vkládání záznamů

- Neuspořádaný (nesekvenční) soubor
  - Vkládáme na konec
    - Poslední blok, popř. nový
  - Vkládáme do volného místa v existujícím bloku
    - Může být problematické v případě proměnlivé délky záznamu

# Vkládání záznamů

- Uspořádaný (sekvenční) soubor
  - Nemožné, pokud není nepřímá adresace ani se nepoužívají offsety
  - Najdi místo v „blízkém“ bloku → reorganizuj
    - Přesunutí posledního záznamu do následujícího bloku
      - Nutné přidat příznak do původního bloku
  - Ulož do přetokového bloku
    - Odkaz na přetokový blok je součástí hlavičky bloku

# Aktualizace záznamů

## ■ Zvětšení délky záznamu

- Nemusí se vytvářet náhrobky
- Posunout následující záznamy
- Vytvořit přetokový blok
- ...

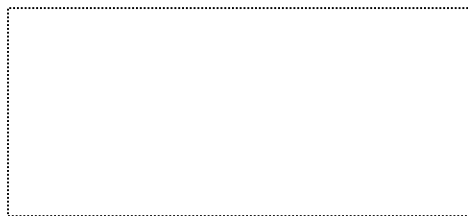
## ■ Zmenšení délky záznamu

- dtto
- Zrušení uvolněných přetokových bloků

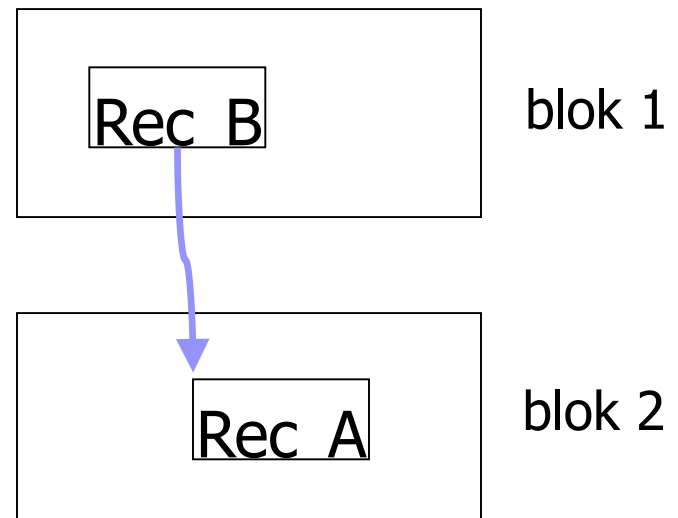
# Vyrovnávací paměti a odkazy

- Problém odkazování na záznamy v paměti
  - Prohazování odkazů (pointer swizzling)
    - Změna DB odkazu na memory odkaz a zpět

Paměť (buffers/cache)

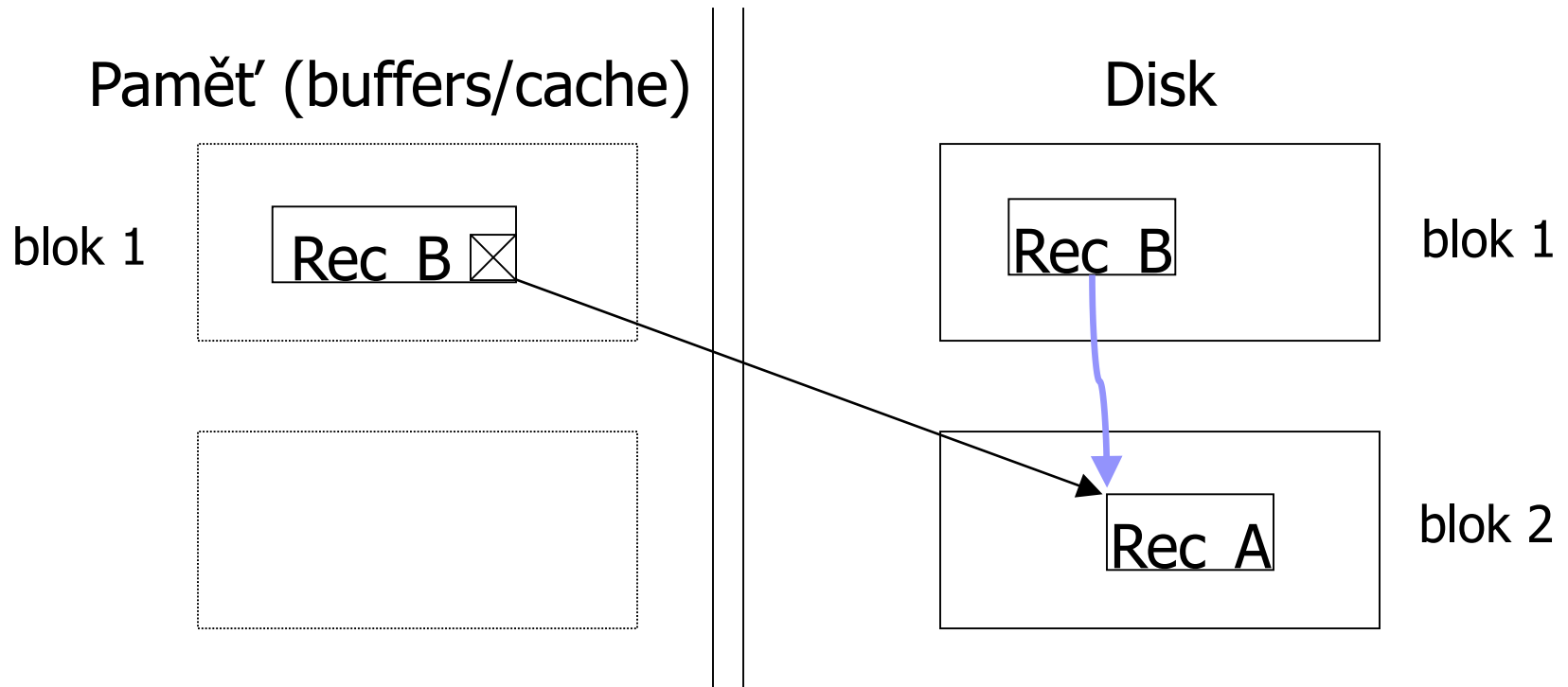


Disk



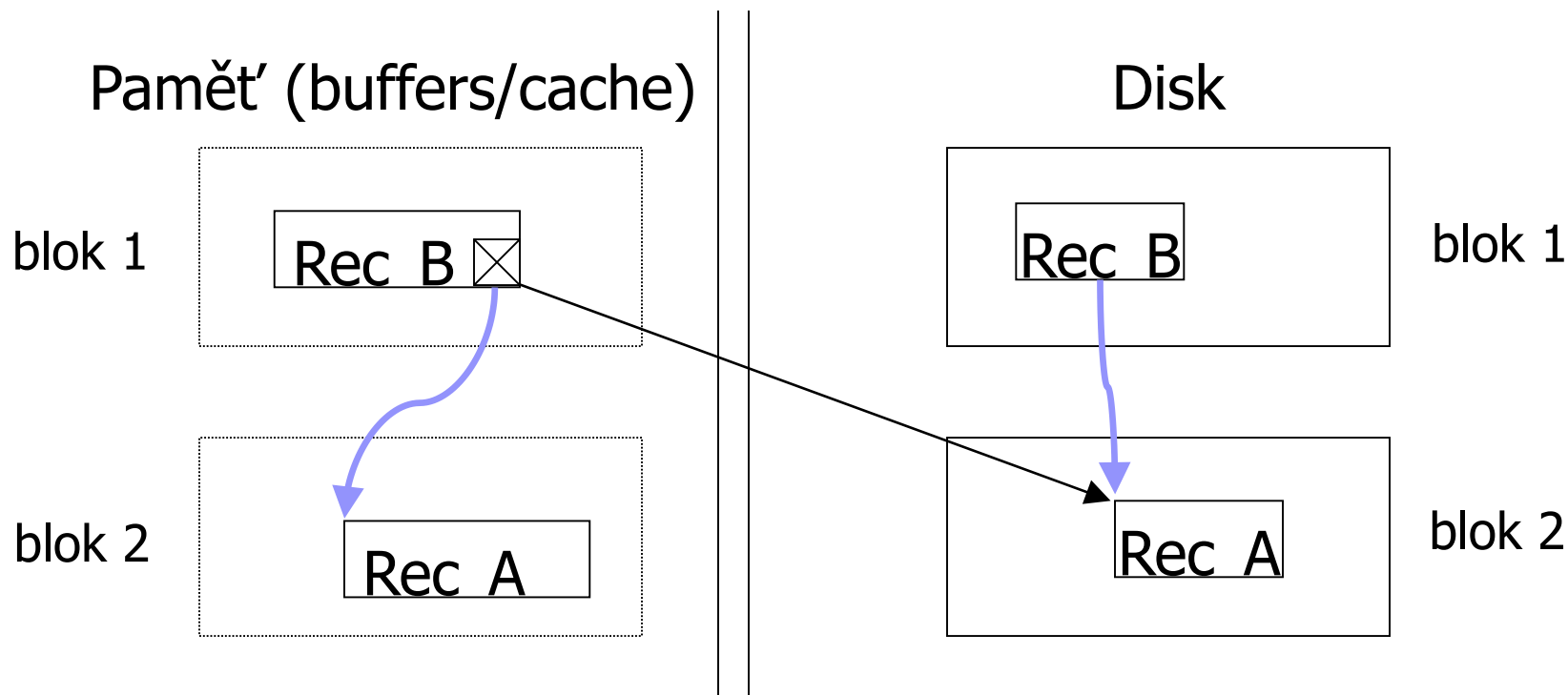
# Vyrovnávací paměti a odkazy

- Po načtení bloku 1 je vše OK



# Prohazování odkazů

- Po načtení bloku 2 je provedena změna odkazu



# Prohazování odkazů

## ■ Kdy:

- Automaticky – ihned po načtení
- Na žádost – při prvním použití
- Nikdy – vždy se používá překladová tabulka

## ■ Implementace:

- DB adresa je měněna na paměťovou adresu
  - Buduje se *Translation table*
    - dvojice (disková adresa, paměť. adresa) pro každý záznam
- Ukazatel má příznak, zda byl prohozen či ne



# Správa vyrovnávací paměti

## ■ Specifikum databází

- Někdy je nutné ponechat bloky v cache „déle“
  - Indexy, vyhodnocování spojení relací, ...

## ■ Různé strategie

- LRU, FIFO, pinned blocks, toss-immediate, ...

# Správa vyrovnávací paměti

## ■ LRU

- Při každém přístupu aktualizovat čas přístupu
- → může být časově náročné

## ■ FIFO

- Uloží se čas načtení a ten se dále nemění
- → nevhodné pro stále používané bloky
  - Např. kořen B<sup>+</sup> stromu

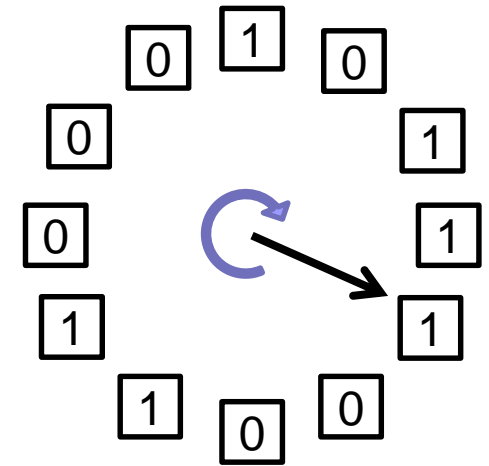
## ■ Pinned blocks

- Bloky trvale v paměti

# Správa vyrovnávací paměti

## ■ Hodiny („Clock“)

- Efektivní aproximace LRU
- Ručka ukazuje na poslední načtený blok.
- Pro načtení nového bloku se ručka otáčí, dokud nenalezne volné místo (blok s nulou).
- Pak načte požadovaný blok a nastaví 1
- Ručka při otáčení snižuje číslo (až na nulu).



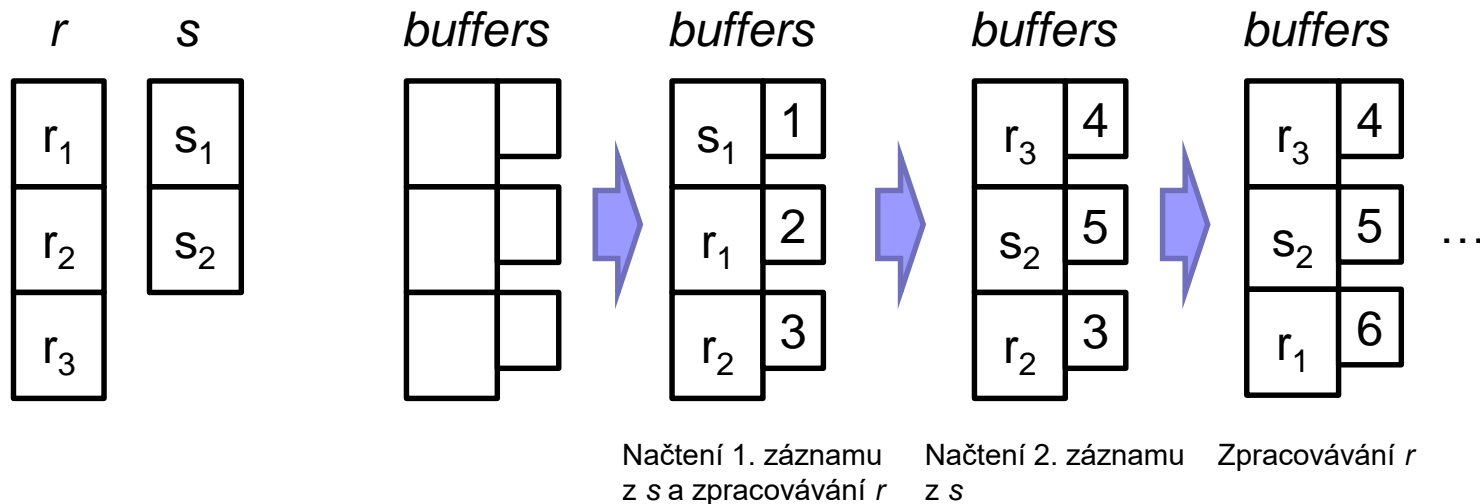
## ■ Lze implementovat i *pinned blocks*. Jak?

# Správa vyrovnávací paměti

## ■ LRU a výpočet spojení dvou relací:

□ Vnořené cykly:

For each  $b_s$  in  $s$  do  
 For each  $b_r$  in  $r$  do  
 Join tuples in  $b_r$  and  $b_s$



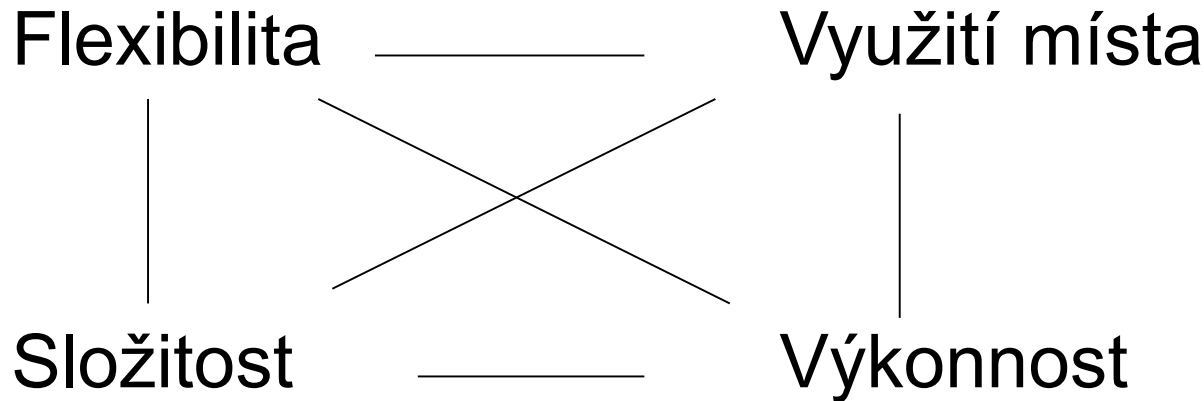
□ LRU nevhodný: přepisují se bloky relace  $s$

■ Nutné použít *pinned blocks* pro relaci  $s$

# Osnova

- Ukládání dat
- Záznamy
- Organizace bloků
- *Příklady*

# Vlastní implementace



- **Otázka náročnosti operací:**
  - Načtení záznamu s daným klíčem
    - Načtení následujícího záznamu
  - Vložení / smazání / aktualizace záznamů
  - Čtení celého souboru
  - Reorganizace souboru

# Specializované systémy

## ■ BigTable

- Distribuované úložiště n-tic od Google
- Škálovatelnost do petabajtů (1PB=1000TB)

F. Chang, J. Dean, S. Ghemawat, et al.:

*Bigtable: A Distributed Storage System for Structured Data*,  
Seventh Symposium on Operating System Design and  
Implementation (OSDI), 2006.

<http://labs.google.com/papers/bigtable-osdi06.pdf>

## ■ HBase

- Distribuované úložiště n-tic
- Open-source Apache projekt Hadoop

<http://hadoop.apache.org/>

# Vlastnosti BigTable a HBase

- Nejsou relační databáze
  - Tzv. NoSQL databáze
- Storage as a “key→value” map
  - row\_id, column\_id, time → value
- Proměnné schéma relace
- Verzování podle času
- Uspořádané podle row\_id