



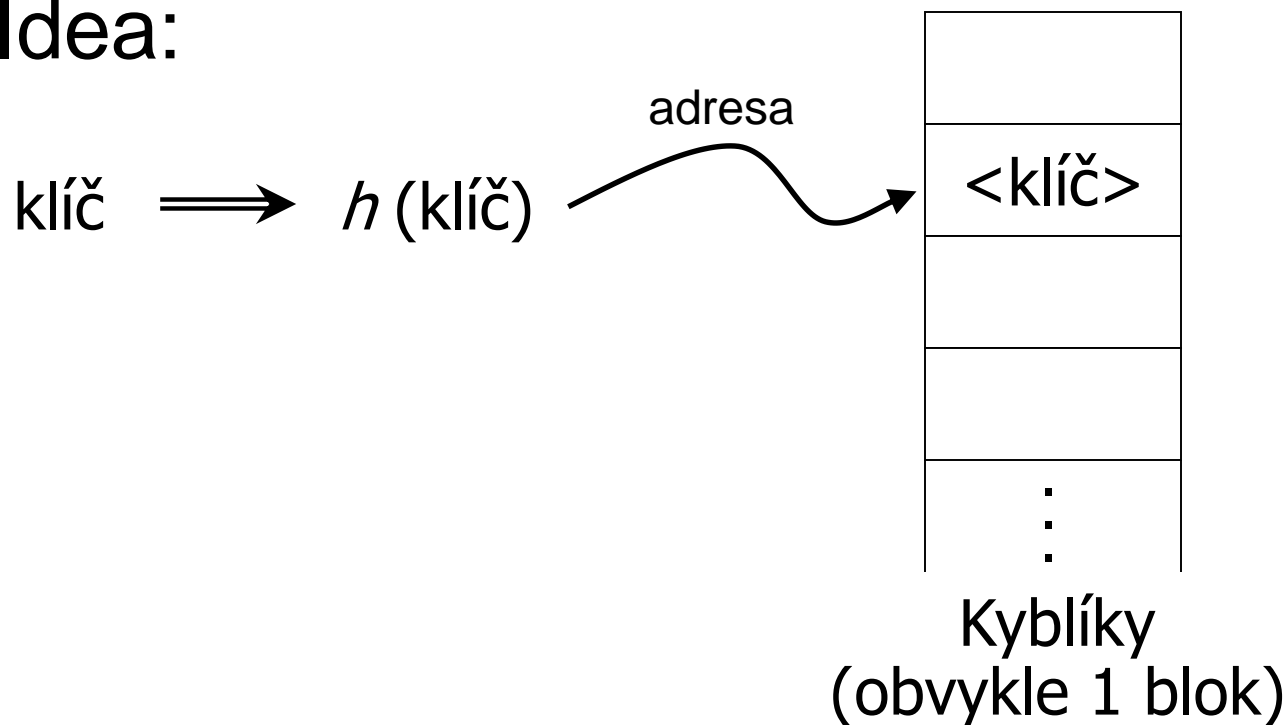
PA152: Efektivní využívání DB  
5. Hašování

Vlastislav Dohnal

# Hašování

- Transformace klíče na adresu
  - Funkce vracející adresu pro vstupní klíč

- Idea:



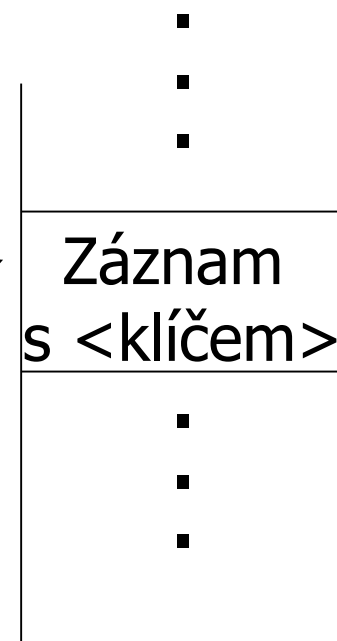
# Možnosti implementace

## ■ Přímá adresace

### □ Adresa záznamu

- Obvykle adresa bloku

klíč  $\implies$   $h$  (klíč)



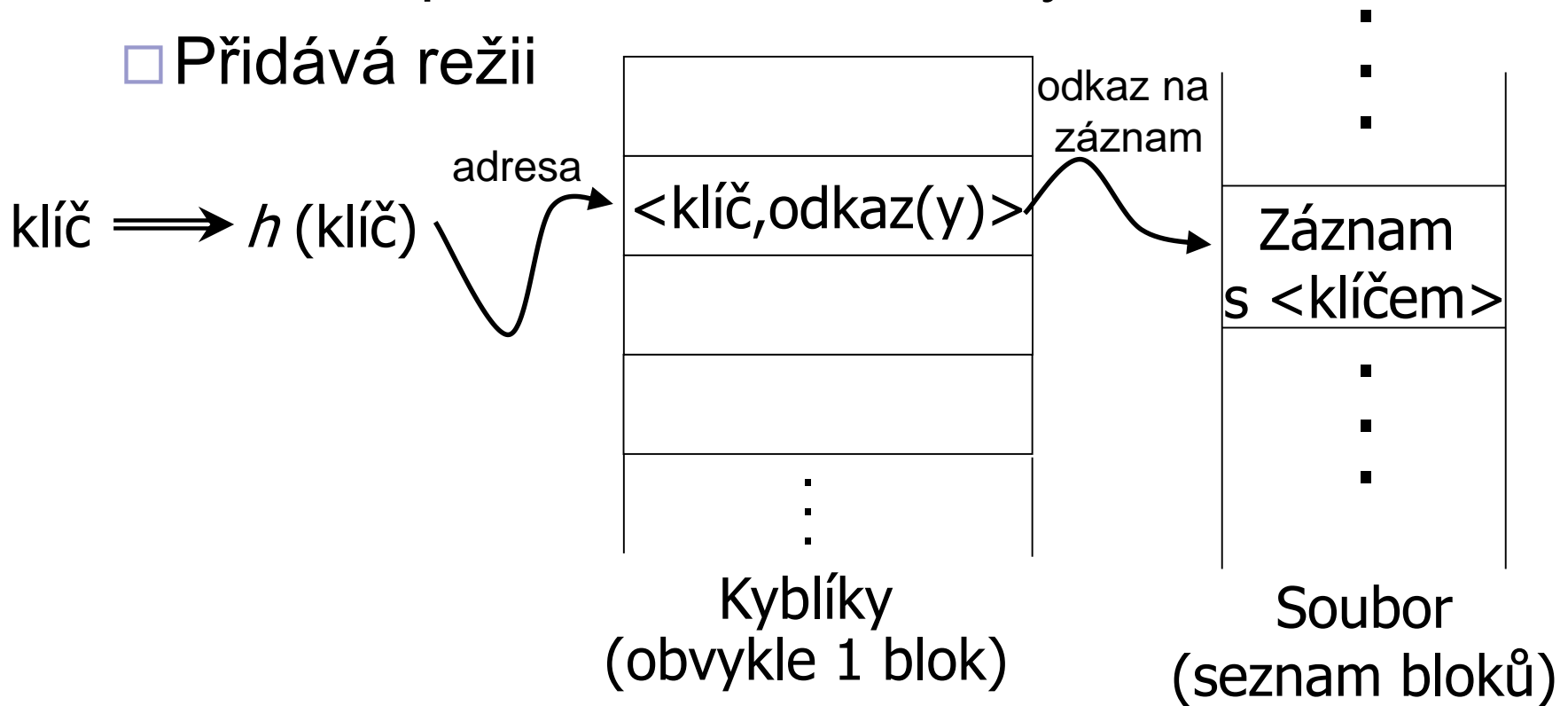
- Vhodné pro hašovaný soubor

Soubor  
(seznam bloků)

# Možnosti implementace

## ■ Nepřímá adresace

- Vhodné pro sekundární indexy
- Přidává režii



Přeskoč [opakování...](#)

# Příklad hašovací funkce

- Klíč je řetězec znaků
  - $n$  bajtů
- Adresní prostor
  - $B$  kyblíků
- Hašovací funkce  $h(x_0x_1\dots x_{n-1})$ 
  - $(x_0+x_1+\dots+x_{n-1}) \bmod B$
  - $(x_0*31^{n-1}+x_1*31^{n-2}+\dots+x_{n-1}) \bmod B$
  - Lze i na proměnlivou délku

# Hašovací funkce

## ■ Požadavky:

### □ Rovnoměrná

- Stejné obsazení všech kyblíků

### □ Náhodná

- Bez korelace vstupu a výstupu

## ■ „Velká věda“ → speciální literatura

### □ Dobrá funkce je alespoň rovnoměrná

# Použití kyblíků

- Kyblík má větší kapacitu než 1
  - Uspořádat klíče?
    - Ano, pokud chceme zrychlit přístup
    - Ano, pokud je málo aktualizací
    - Ne, pokud je hodně aktualizací

# Základní pojmy

- Hašovací funkce

- Kolize

- Na vypočtené adrese je již něco uloženo

- Není problém, pokud lze uložit více klíčů

- Přetečení

- Kapacita kyblíku je naplněna

- Přetoková oblast

- Statické vs. dynamické hašování

- Podle změny velikosti adresového prostoru

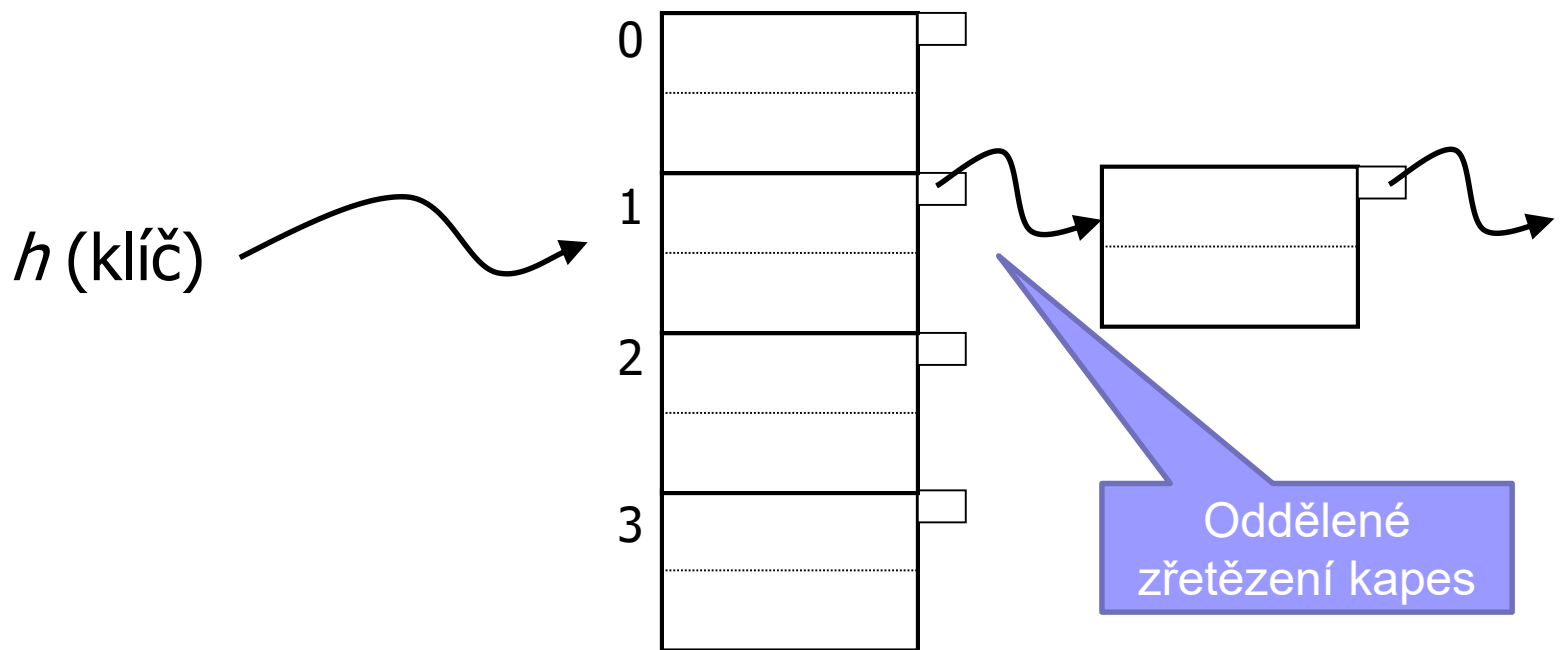


# Řešení kolizí

- Uzavřené adresování (otevřené hašování)
  - Vypočtená adresa je fixní
  - Při přetečení vytvoř nový kyblík (přetoková oblast)
    - Řetězení přetokových oblastí
- Otevřené adresování (uzavřené hašování)
  - Existence kolizní funkce
    - Lineární, kvadratické, dvojité hašování
    - Viz Organizace souborů

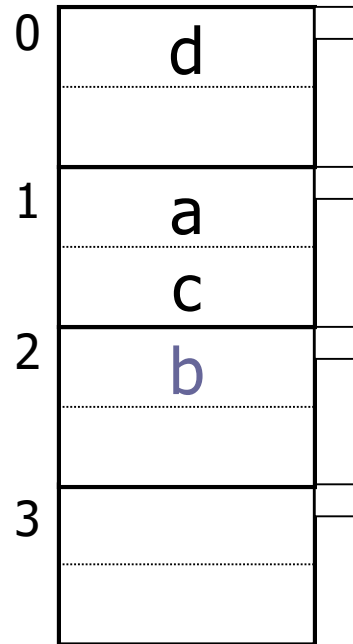
# Příklad

- Statické uzavřené adresování
  - Kolize pomocí přetokových oblastí
  - Kapacita kyblíku = 2 klíče



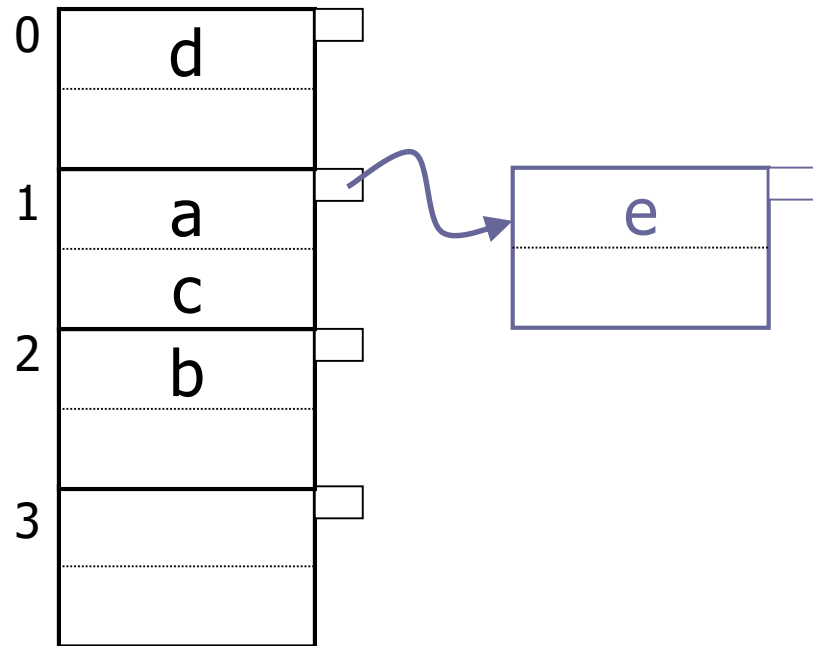
# Příklad vkládání

- $h(„b“) = 2$



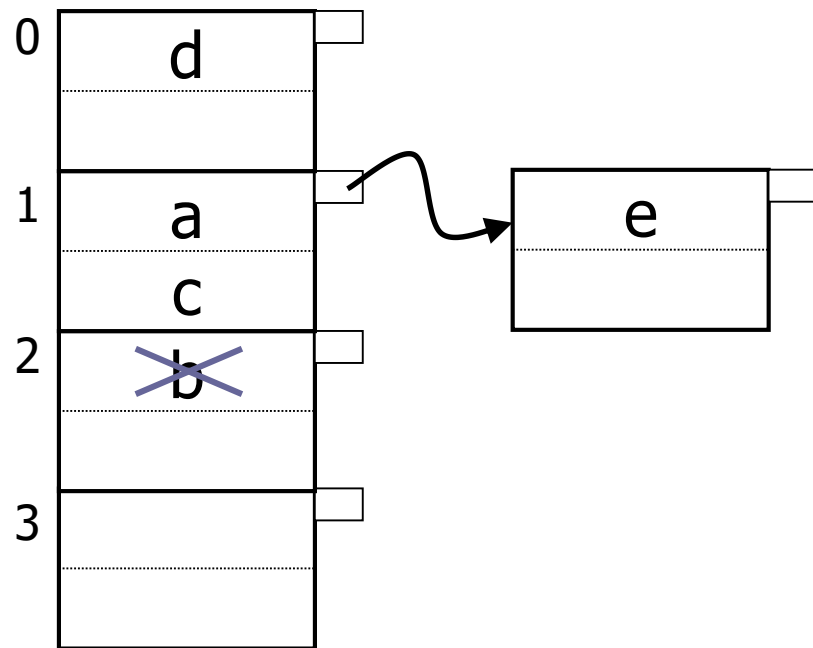
# Příklad vkládání

- $h(„e“) = 1$



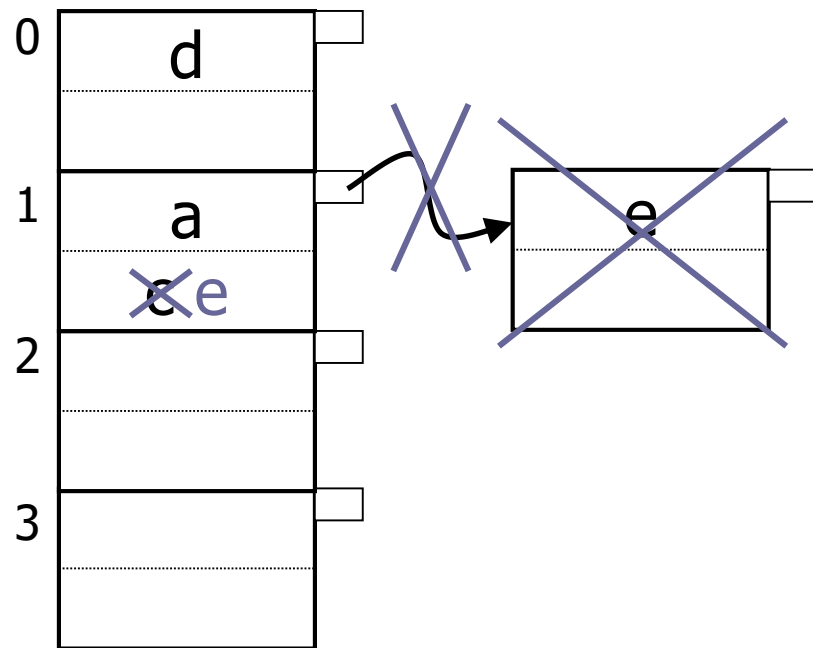
# Příklad mazání

- Uvolňovat přetokové oblasti
- Smaž „b”



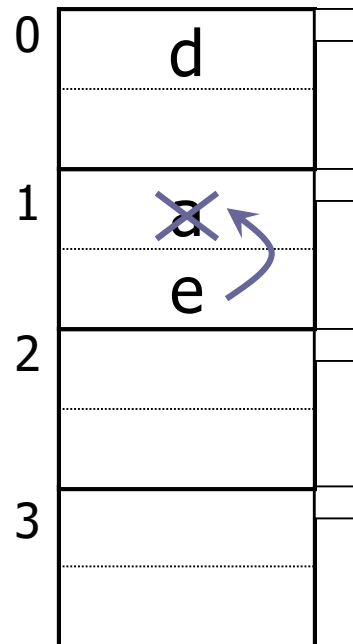
# Příklad mazání

- Uvolňovat přetokové oblasti
- Smaž „c”



# Příklad mazání

- Uvolňovat přetokové oblasti
- Smaž „a”



# Empirická znalost

- Zaplnění udržovat mezi 50% a 80%
  - Zaplněnost = počet klíčů / max. kapacita
  - $< 50\%$  – plýtvání místem
  - $\geq 80\%$  – příliš mnoho kolizí
    - Přetokové oblasti zpomalují vyhledávání i vkládání



# Dynamická data

## ■ Statické hašování

- Přetoky → reorganizace

- Tj. vytvoření nové hašovací funkce

## ■ Dynamické hašování

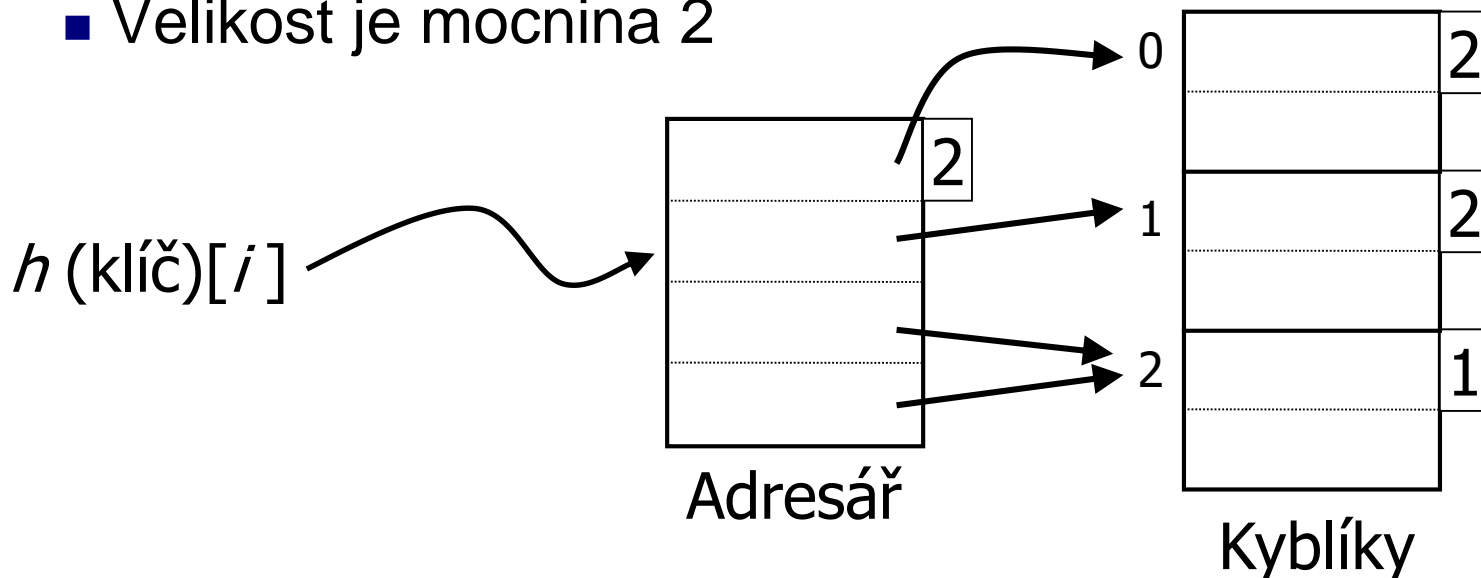
- Rozšiřitelné

- Lineární

# Rozšiřitelné hašování

## ■ Idea:

- Využití pouze několika prvních (horních) bitů adresy
- Přidání další tabulky – adresář
  - Velikost je mocnina 2

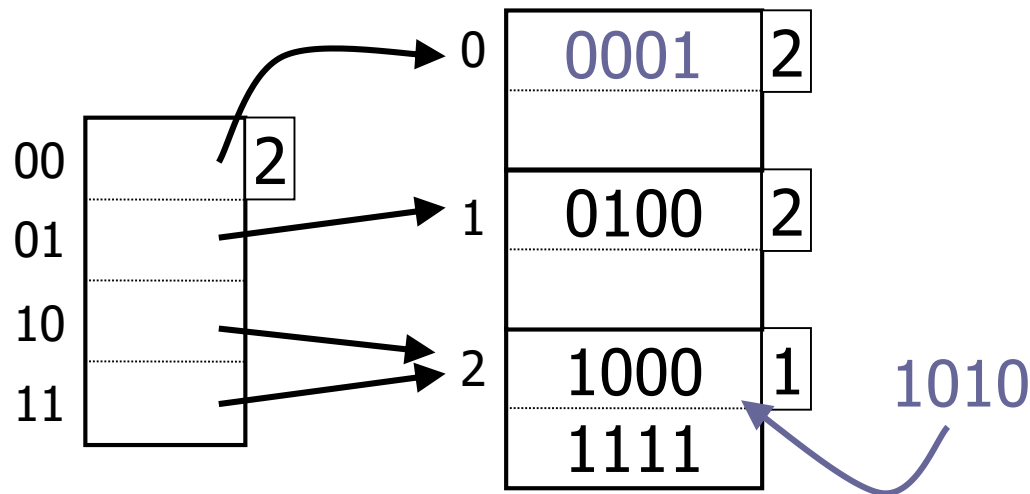


# Rozšiřitelné hašování: vkládání

- Příklad:  $h(x) = x$ , tj. identita
- Najdi kyblík
  - je volné místo  $\rightarrow$  ok
  - není  $\rightarrow$  rozštěpení kyblíku
    - přerozdělení obsahu

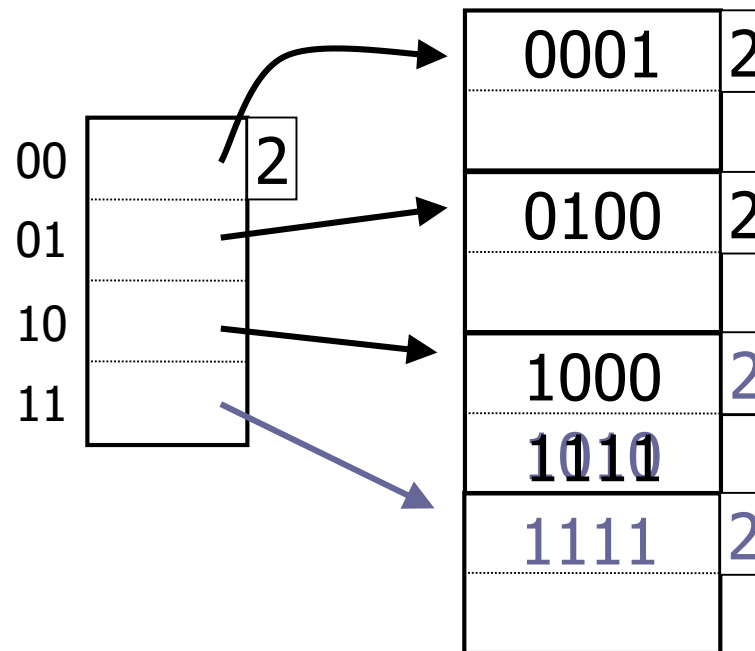
## ■ Vložení

- 0001
- 1010



# Rozšiřitelné hašování: vkládání

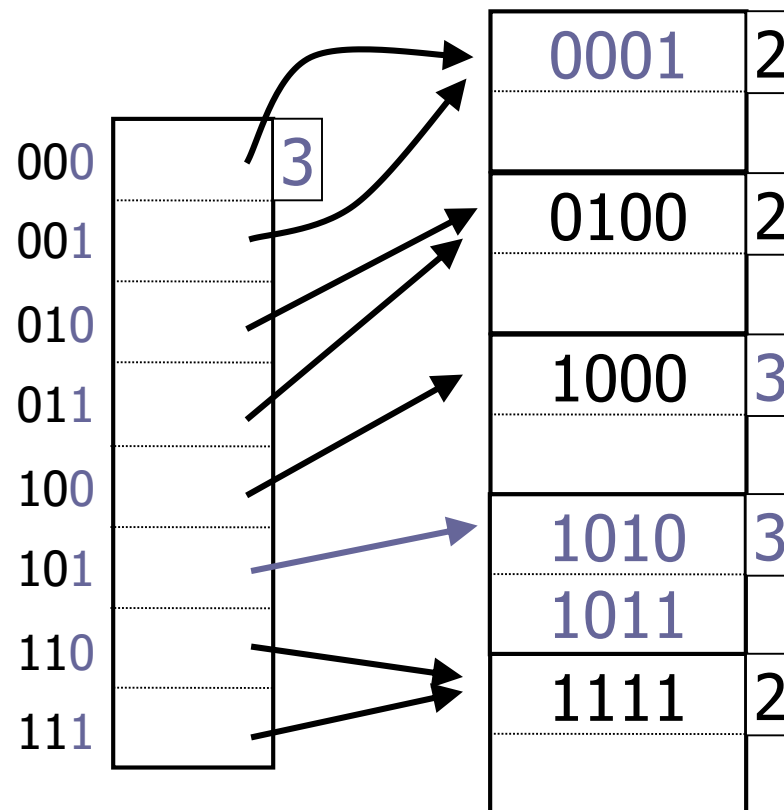
- Při vkládání 1010
  - Rozštěpení kyblíku



# Rozšiřitelné hašování: vkládání

## ■ Vložení 1011

- Adresář je plný → zdvojnásobení (přidání bitu)



# Rozšiřitelné hašování: mazání

## ■ Mazání klíče

- Smaž klíč

- Kyblík je prázdný

- a) Nedělej nic → předpoklad nového vkládání

- b) Sloučení sousedních kyblíků

- Mající stejný prefix o jeden bit kratší

- Může být i zmenšen adresář

# Rozšiřitelné hašování: hodnocení

## ■ Výhody

- Měnící se data
- Méně plýtvá místem (než statické hašování)
- Lokální reorganizace

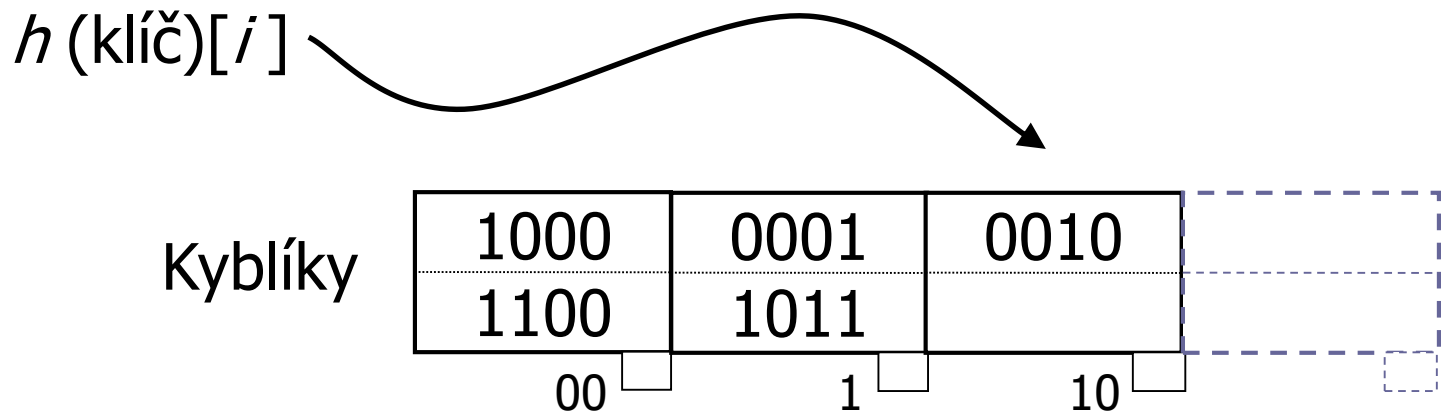
## ■ Nevýhody

- Další úroveň nepřímých odkazů
  - Ok, pokud je adresář v paměti
- Adresář se zdvojnásobuje
  - Nemusí se vejít do paměti
  - Ale kyblíky rostou lineárně!

# Lineární hašování

## ■ Idea:

- Využití pouze několika posledních (dolních) bitů adresy
  - Konkrétně  $i$  bitů
- Žádný adresář
- Soubor roste lineárně





# Lineární hašování: vkládání

## ■ Parametry:

□  $i$  – délka sufixu  $\Rightarrow h(k)[i]$

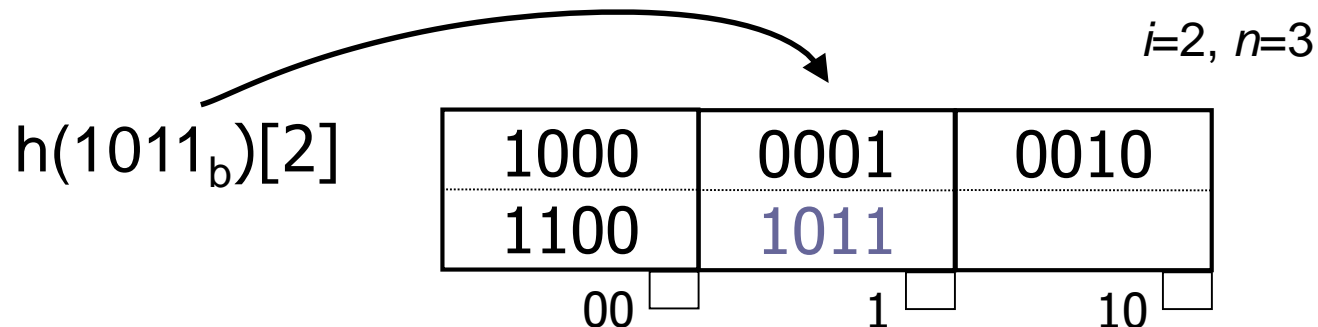
□  $n$  – počet kyblíků

## ■ Vlož $1011_b$

□  $h(1011_b)[2] = 11_b = m$

■ Pokud  $m < n$ , vlož do kyblíku  $m$

■ Jinak vlož do kyblíku  $m - 2^{i-1}$

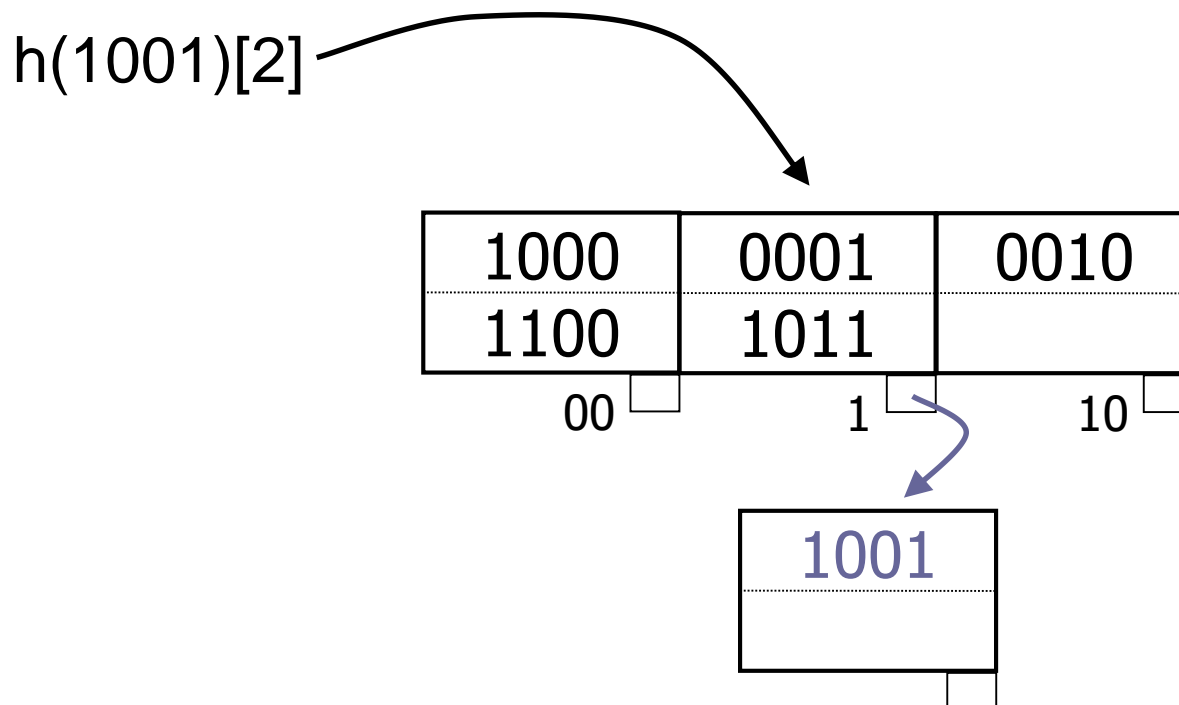


# Lineární hašování: vkládání

## ■ Vlož 1001

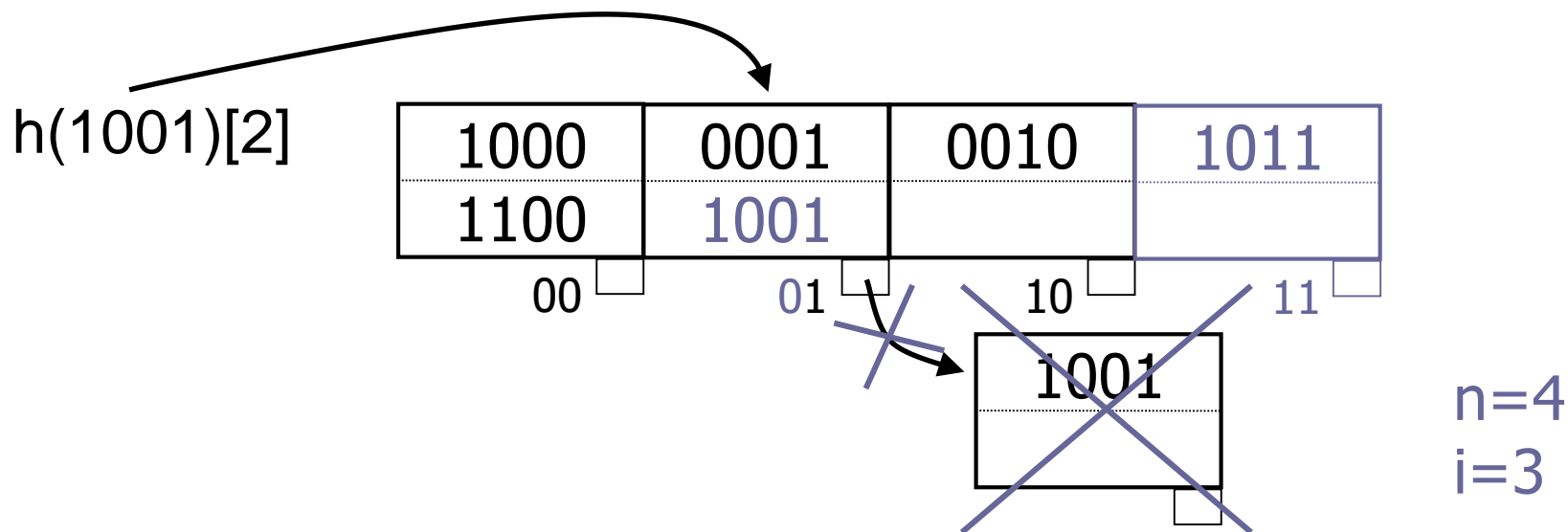
□  $h(1001)[2] = 01$

□ Není místo → přetoková oblast



# Lineární hašování: štěpení kyblíku

- Při vkládání kontroluj naplnění kyblíku
  - >80%, pak rozděl kyblík (jehož adresa je  $0abcd\dots z$ )
    - Resp. přidej nový → adresa je  $1abcd\dots z$
    - Rozděl data v kyblíku  $0abcd\dots z$  mezi  $[01]abcd\dots z$
  - Dělí se vždy adresa  $(n - 2^{i-1})$ , tj.  $3 - 2^1 = 1$



# Lineární hašování

## ■ Vkládání nového klíče

- Může vést k přetokové oblasti
- Může způsobit větší naplnění než 80%
  - Proveďte se štěpení kyblíku
  - Štěpený kyblík může být různý od kyblíku, do kterého se vkládá!
- Po přidání  $2^i$ -tého kyblíku, zvětši  $i$ 
  - Tj. počet kyblíků překročí  $2^i - 1$

# Lineární hašování: hodnocení

## ■ Výhody

- Měnící se data
- Méně plýtvá místem (než statické hašování)
- Lokální reorganizace
- Žádná dodatečná překladová tabulka

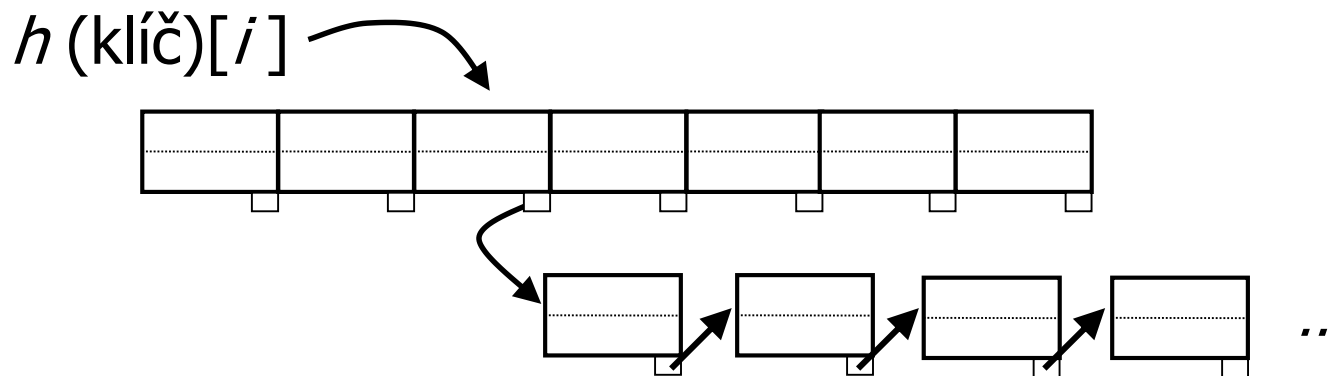
## ■ Nevýhody

- Přetokové oblasti

# Lineární hašování: příklad

## ■ „Chybná“ data

- Posledních  $x$  bitů nerozliší klíče



- Jeden kyblík má všechna data, ostatní nic
  - Faktor naplnění vysoký → stále se štěpí

# Hašování vs. indexování

## ■ Hašování

- Vhodné pro přesné dotazy  
SELECT ... WHERE a=5

## ■ Indexování

- Vhodné pro rozsahové dotazy  
SELECT ... WHERE a>5

# Bitmapový / Rastrový index

- Atribut  $X$  má  $h$  unikátních hodnot
- Kolekce  $h$  bitových polí
  - Pro každou hodnotu je jedno pole
  - Délka pole je počet záznamů ( $n$ )
  - = invertovaný soubor
- Vhodné pro atributy s „malým“ počtem hodnot
  - Typicky  $<10\%$  záznamů



# Bitmapový index: příklad

## ■ Relace $R(F, G)$

$F$	$G$
30	foo
30	bar
40	baz
50	foo
40	bar
30	baz
34	foo

## ■ Bitmapový index pro $G$

<i>Hodnota</i>	<i>Vektor</i>
foo	1001001
bar	0100100
baz	0010010

# Bitmapový index: vlastnosti

## ■ Nevýhody

### □ Paměťová náročnost

- Pokud je klíč primárním klíčem, pak

$$n(n + \log_2 n) \text{ bitů}$$

- Mnoho záznamů se stejnou hodnotou (mnoho jedniček)

- Změň na bitové pole bloků (a nikoli záznamů)

### □ Aktualizace záznamů

- Nová hodnota → nové bitové pole
- Nový záznam → rozšíření všech polí

# Bitmapový index: vlastnosti

## ■ Výhody

- Rychlé operace na bitech (AND, OR)
- Použitelné i pro rozsahové dotazy
- Snadné kombinace více indexů dohromady
  - Typicky jedno-atributové indexy

# Bitmapový index: komprese

- Zmenšení polí
  - Málo 1, hodně 0
  - Obvykle Run-Length Encoding (RLE)
- RLE
  - Rozdělení na části
    - Sekvence „ $i$ “ nul následovaná *jedničkou*
  - Číslo „ $i$ “ uložit binárně
  - Kód čísla „ $i$ “ je  
„Délka binárního kódu  $i$ , vlastní číslo  $i$ “
- Vlastnost
  - Sekvence vždy končí jedničkou
  - Chybějící nuly na konci lze doplnit podle počtu záznamů

# Bitmapový index: RLE

## ■ Příklad komprese – 24 bitů

- Sekvence: 0000 0000 0000 0110 0010 0000
  - 13x nula, 1x jednička
  - 0x nula, 1x jednička
  - 3x nula, 1x jednička
  - 5x nula a nic.... → ignoruji
- Binárně tedy:
  - $13_d \rightarrow 1101_b$
  - $0_d \rightarrow 0_b$
  - $3_d \rightarrow 11_b$
- Kód RLE:
  - Posloupnost „částí“:
    - délka čísla v prefixovém kódu, pak binární číslo
  - Kód: 11101101001011

# Bitmapový index: RLE

## ■ Příklad dekomprese

□ Kód 11101101001011

□ Rozdělení na části a dekomprese...

■ Délka binárního čísla:

□ počet bitů (jednička a pak nula)

■ 11101101001011

□ Dekódování částí

■ 11101101 → 0000 0000 0000 01

■ 00 → 1

■ 1011 → 0 001

□ Výsledná sekvence:

■ 0000 0000 0000 0110 001

■ Chybějící nuly na konci lze doplnit podle počtu záznamů

# Bitmapový index: operace

- Bitové operace

  - AND, OR

- RLE řetězce

  - Dekomprimovat → snadné

  - Bez dekomprese

    - Složitější algoritmus, ale možné

    - AND: suma čísel  $i$  v kódech se musí shodovat

    - OR: analogicky...

    - 11101101001011 OR/AND 1101011101001010

# Bitmapový index: implementace

## ■ Otázky pro efektivní použití:

1. Nalezení bitového pole pro konkrétní hodnotu klíče
2. Mám bitové pole, jak načtu záznamy?
3. Aktualizace záznamů, co s indexem?



# Bitmapový index: řešení

- Ad 1: (Nalezení bitového pole pro konkrétní hodnotu klíče)
  - Pro hodnotu klíče máme bitové pole
    - B<sup>+</sup>-strom pro hodnoty klíče
    - V listu odkaz na bitové pole
  - Uložení bitových polí
    - Záznamy variabilní délky
- Ad 2: (Mám bitové pole, jak načtu záznamy?)
  - Nalezení záznamu  $r$  (pořadí záznamu)
    - Sekundární index pro čísla záznamů
    - Pole odkazů na záznamy (náhrada za bit. pole)
    - Seznam počtu záznamů v blocích

# Bitmapový index: řešení

- Ad 3: (Aktualizace záznamů, co s indexem?)
  - Čísla záznamů jsou fixní
    - Mazání záznamu
      - náhrobek v indexu/souboru a změna 1 na 0 v jednom bitovém poli
    - Vkládání záznamu
      - přidej na konec souboru (nové číslo záznamu)
      - do správného bitového pole připojit 1
        - pole nemusí existovat, vytvoř nové

# Bitmapový index: řešení

- Ad 3: (Aktualizace záznamů, co s indexem?)
  - Čísla záznamů nejsou fixní
    - Reorganizace všech polí (zrušení 1 bitu)
    - Aktualizace pole s odkazy na záznamy
    - Málo používaná verze

# Bitmaps -- data

## Settings:

```
lineitem ( L_ORDERKEY, L_PARTKEY , L_SUPPKY, L_LINENUMBER,  
L_QUANTITY, L_EXTENDEDPRICE ,  
L_DISCOUNT, L_TAX , L_RETURNFLAG, L_LINESTATUS ,  
L_SHIPDATE, L_COMMITDATE,  
L_RECEIPTDATE, L_SHIPINSTRUCT ,  
L_SHIPMODE , L_COMMENT );
```

```
create bitmap index b_lin_2 on lineitem(l_returnflag);
```

```
create bitmap index b_lin_3 on lineitem(l_linestatus);
```

```
create bitmap index b_lin_4 on lineitem(l_linenum);
```

- 100000 rows ; cold buffer
- Dual Pentium II (450MHz, 512KB), 512 MB RAM, 3x18GB drives (10000RPM), Windows 2000.

# Bitmaps -- queries

## Queries:

### □ 1 attribute

```
select count(*) from lineitem where l_returnflag =  
    'N';
```

### □ 2 attributes

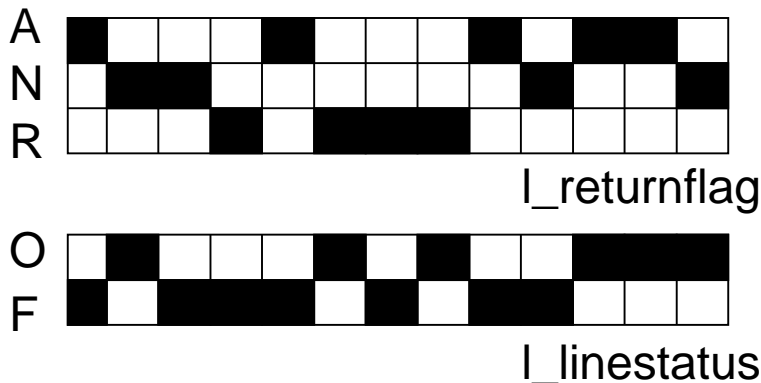
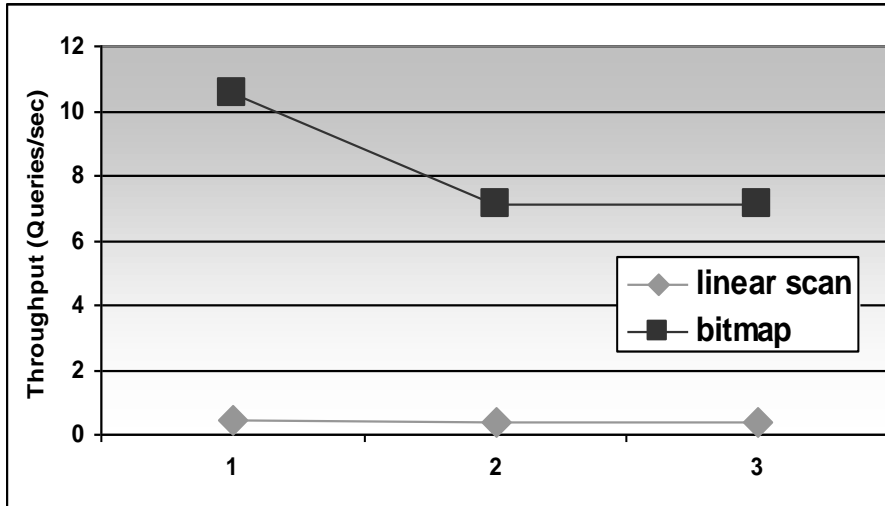
```
select count(*) from lineitem where l_returnflag = 'N'  
    and l_linenumbers > 3;
```

### □ 3 attributes

```
select count(*) from lineitem where l_returnflag =  
    'N' and l_linenumbers > 3 and l_linestatus = 'F';
```

# Bitmaps

- Order of magnitude improvement compared to scan.
- Bitmaps are best suited for multiple conditions on several attributes, each having a low selectivity.



# Víceklíčový / Kompozitní index

- Index nad více atributy

- Důvod:

SELECT jméno, plat FROM zam

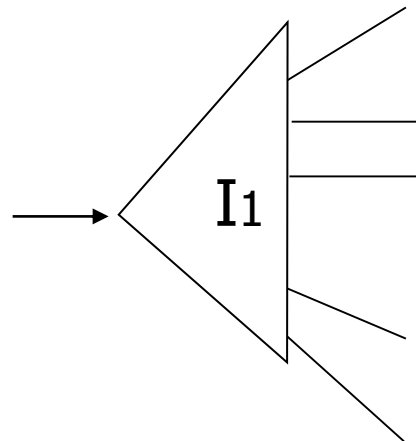
WHERE oddělení='Hračky' AND plat < 10000

- Řešení

- a) Index pro jeden atribut + filtrování
- b) Nezávislé indexy pro atributy + průnik vyhovujících
- c) Index v indexu
- d) Spojení klíčů v jeden

# Index pro jeden atribut

- SELECT jméno, plat FROM zam  
WHERE oddělení='Hračky' AND plat < 10000
- Index pro *oddělení*
  - Nalezené záznamy načítej a filtruj pomocí  
*plat < 10000*

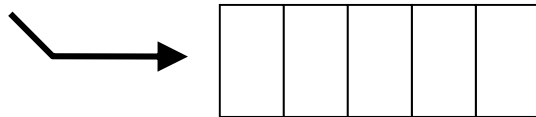




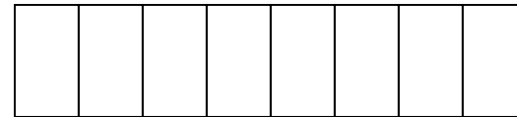
# Nezávislé indexy

- Index pro *oddělení*
- Index pro *plat*
- Každý index vrátí seznam kandidátů
  - Průnik seznamů → výsledek dotazu

oddělení='Hračky'



plat < 10000

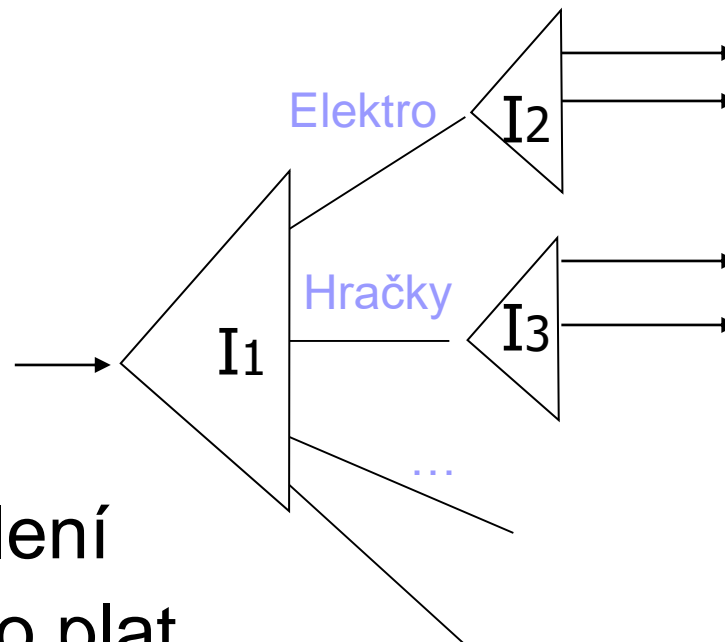


bucket with record pointers

# Index v indexu

- Index pro první atribut

- V listu je odkaz na index pro druhý atribut



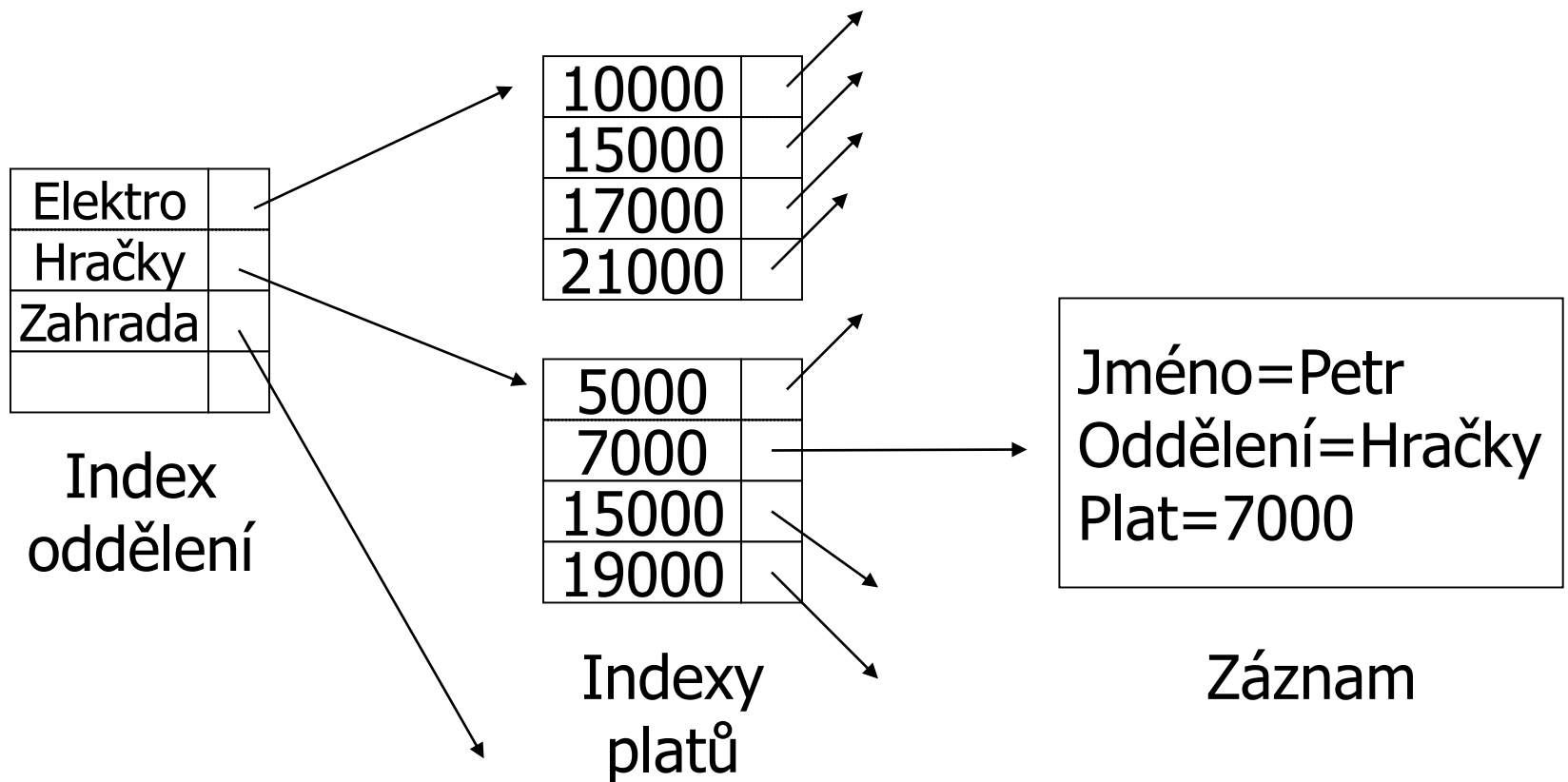
- $I_1$  pro oddělení

- $I_x$ ,  $x=2..k$  pro plat

- $I_2$  obsahuje pouze záznamy se stejným oddělením (Elektro)

# Index v indexu: příklad

- SELECT jméno, plat FROM zam  
WHERE oddělení='Hračky' AND plat < 10000

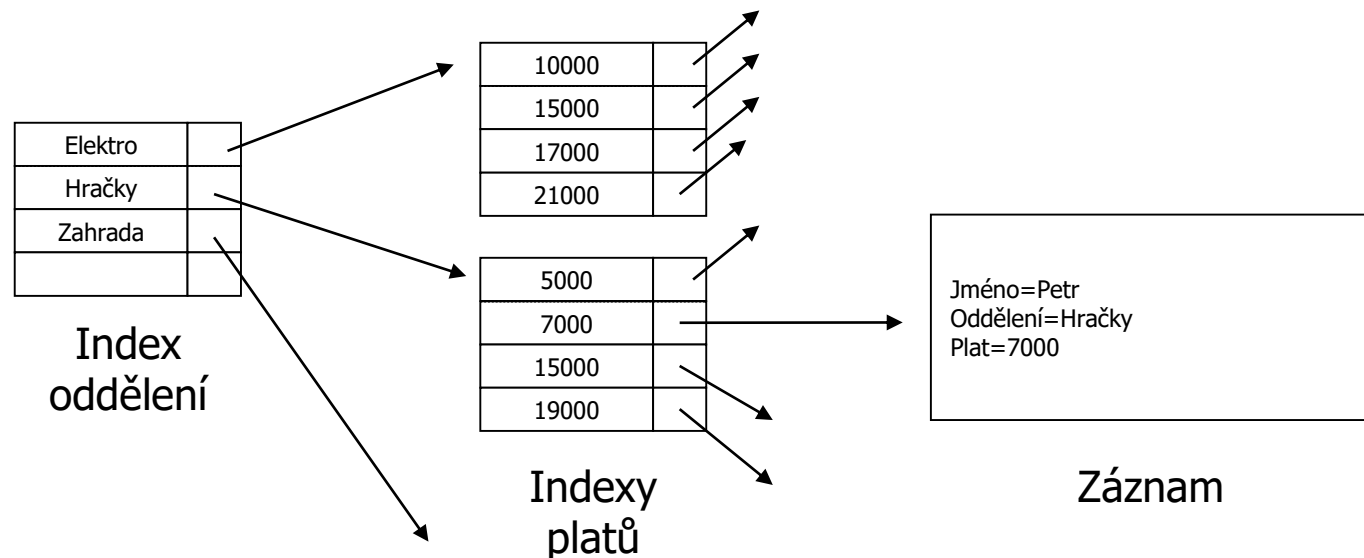


# Index v indexu

## ■ Pro které dotazy je použitelný?

□ SELECT jméno, plat FROM zam WHERE

- a) oddělení = 'Hračky' AND plat ≥ 10000
- b) oddělení = 'Hračky'
- c) plat = 10000



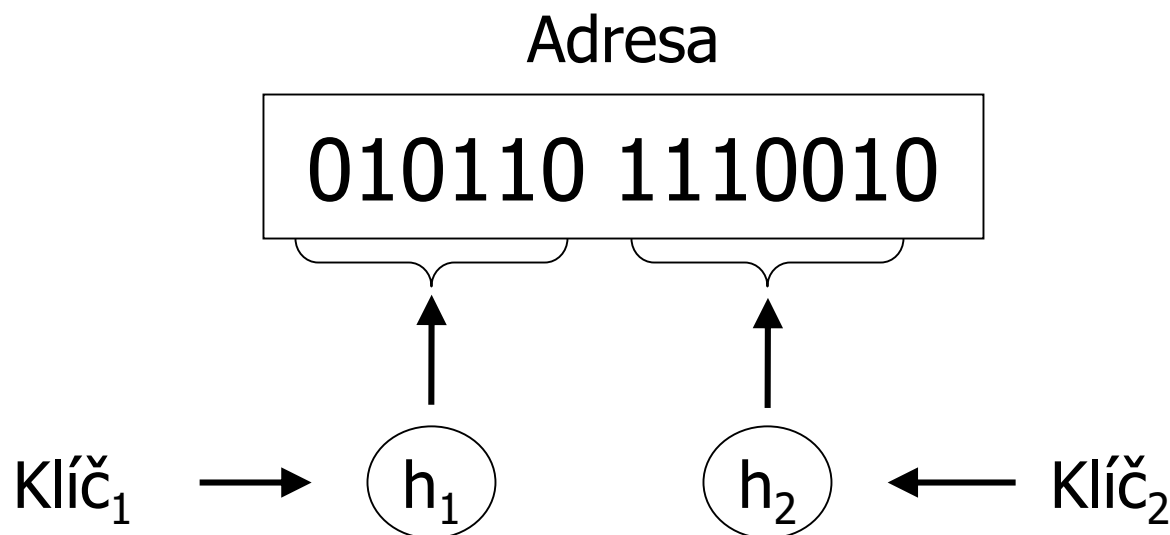
# Spojení klíčů v jeden

- Podobné indexu pro jeden klíč
  - Hodnota klíče je spojená
    - Spojení řetězců, kombinace čísel, ...
- V indexování
  - bez komprimace se příliš nepoužívá
- V hašování
  - dělená hašovací funkce  
(Partitioned hash function)

# Dělená hašovací funkce

## ■ Idea:

- Dva klíče
- Dvě hašovací funkce
- Jedna adresa



# Dělená hašovací funkce: příklad

## ■ Oddělení

$h_1(\text{Elektro}) = 0$

$h_1(\text{Hračky}) = 1$

$h_1(\text{Zahrada}) = 1$

## ■ Plat

$h_2(10000) = 01$

$h_2(20000) = 11$

$h_2(30000) = 10$

$h_2(40000) = 00$

## ■ Záznamy k vložení

□  $\langle \text{Petr}, \text{Elektro}, 10000 \rangle$

□  $\langle \text{Jan}, \text{Hračky}, 10000 \rangle$

□  $\langle \text{Alice}, \text{Zahrada}, 30000 \rangle$

000	
001	$\langle \text{Petr}, \text{Elektro}, 10000 \rangle$
010	
011	
100	
101	$\langle \text{Jan}, \text{Hračky}, 10000 \rangle$
110	$\langle \text{Alice}, \text{Zahrada}, 30000 \rangle$
111	

# Dělená hašovací funkce: příklad

## ■ Oddělení

$h_1(\text{Elektro}) = 0$

$h_1(\text{Hračky}) = 1$

$h_1(\text{Zahrada}) = 1$

## ■ Plat

$h_2(10000) = 01$

$h_2(20000) = 11$

$h_2(30000) = 10$

$h_2(40000) = 00$

000	<Pavel,...> <Lukáš,...>
001	<Petr,...>
010	<Marie,...>
011	
100	<Anna,...>
101	<Jan,...>
110	<Alice,...>
111	<Veronika,...>

## ■ Najdi

- zaměstnance z oddělení hraček a platem 40000.



# Dělená hašovací funkce: příklad

## ■ Oddělení

$h_1(\text{Elektro}) = 0$

$h_1(\text{Hračky}) = 1$

$h_1(\text{Zahrada}) = 1$

## ■ Plat

$h_2(10000) = 01$

$h_2(20000) = 11$

$h_2(30000) = 10$

$h_2(40000) = 00$

000	<Pavel,...> <Lukáš,...>
001	<Petr,...>
010	<Marie,...>
011	
100	<Anna,...>
101	<Jan,...>
110	<Alice,...>
111	<Veronika,...>

## ■ Najdi

zaměstnance s platem 30000

# Dělená hašovací funkce: příklad

## ■ Oddělení

$h_1(\text{Elektro}) = 0$

$h_1(\text{Hračky}) = 1$

$h_1(\text{Zahrada}) = 1$

## ■ Plat

$h_2(10000) = 01$

$h_2(20000) = 11$

$h_2(30000) = 10$

$h_2(40000) = 00$

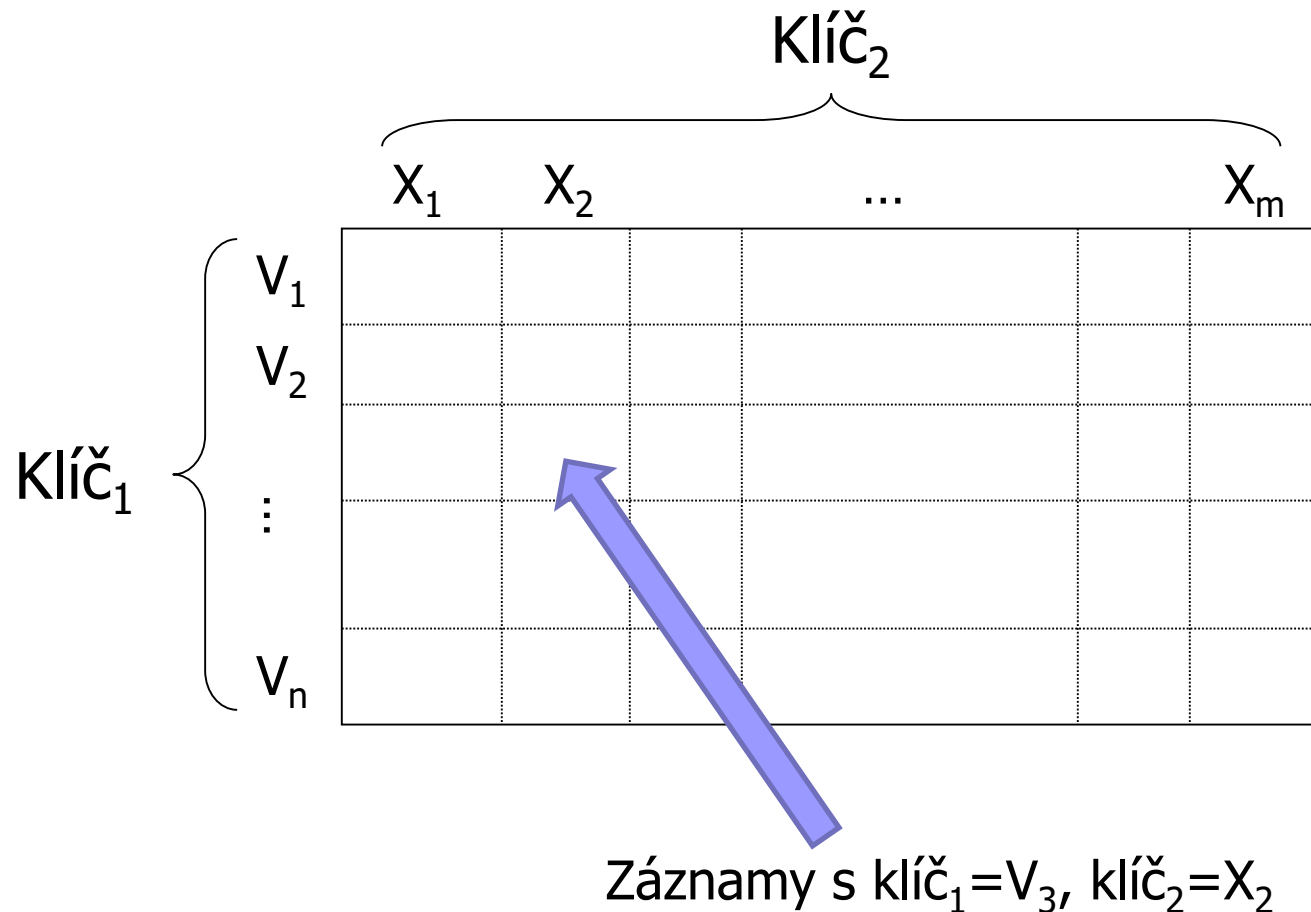
## ■ Najdi

zaměstnance z oddělení hraček

000	<Pavel,...> <Lukáš,...>
001	<Petr,...>
010	<Marie,...>
011	
100	<Anna,...>
101	<Jan,...>
110	<Alice,...>
111	<Veronika,...>

# Jiný víceklíčový index

- Grid (mřížka)
- Idea:



# Grid: vlastnosti

## ■ Rychlé pro přesné dotazy

□ klíč<sub>1</sub> = V<sub>i</sub> ∧ klíč<sub>2</sub> = X<sub>j</sub>

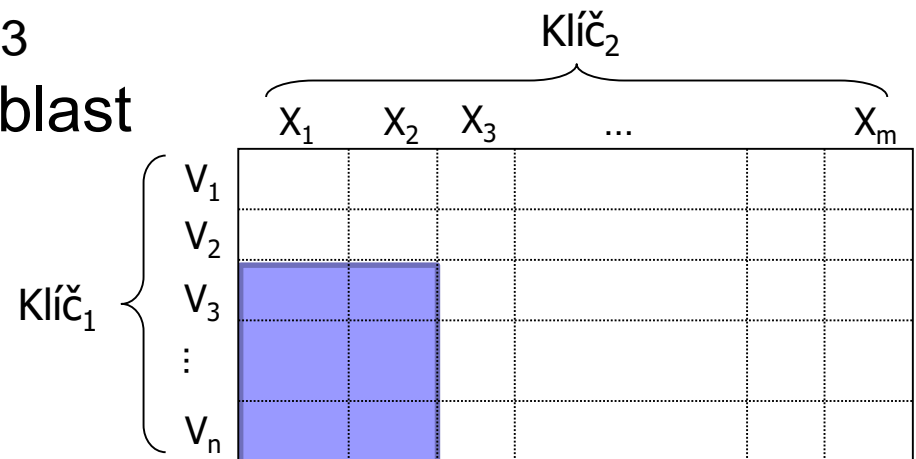
□ klíč<sub>1</sub> = V<sub>i</sub>

□ klíč<sub>2</sub> = X<sub>j</sub>

## ■ Rozsahové dotazy

□ klíč<sub>1</sub> ≥ V<sub>3</sub> ∧ klíč<sub>2</sub> < X<sub>3</sub>

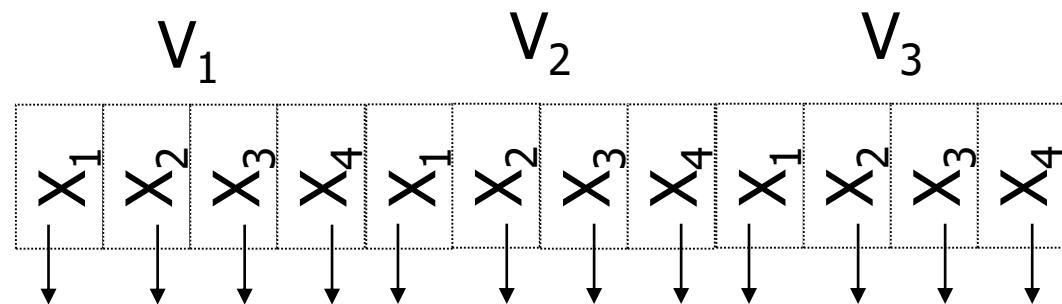
- Vznikne čtvercová oblast



# Grid: implementace

- Jak ukládat mřížku na disku?

- Jako pole



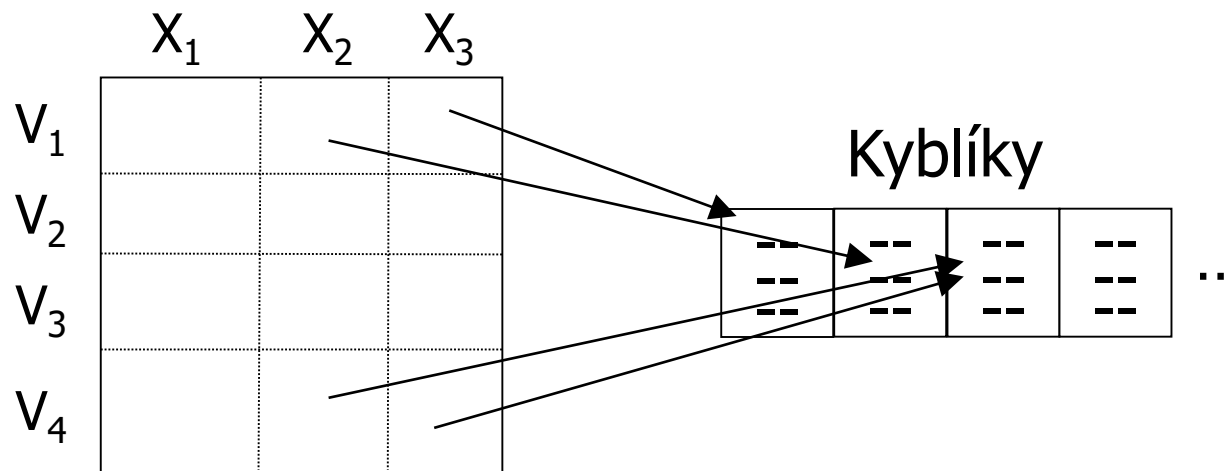
- Problém: rozměr mřížky vs. velikost buňky

- Nevýhoda

- Potřeba pevného rozměru mřížky pro výpočet indexu políčka  $\langle V_x, X_y \rangle$  v poli.
  - Omezená velikost buňky

# Grid: implementace

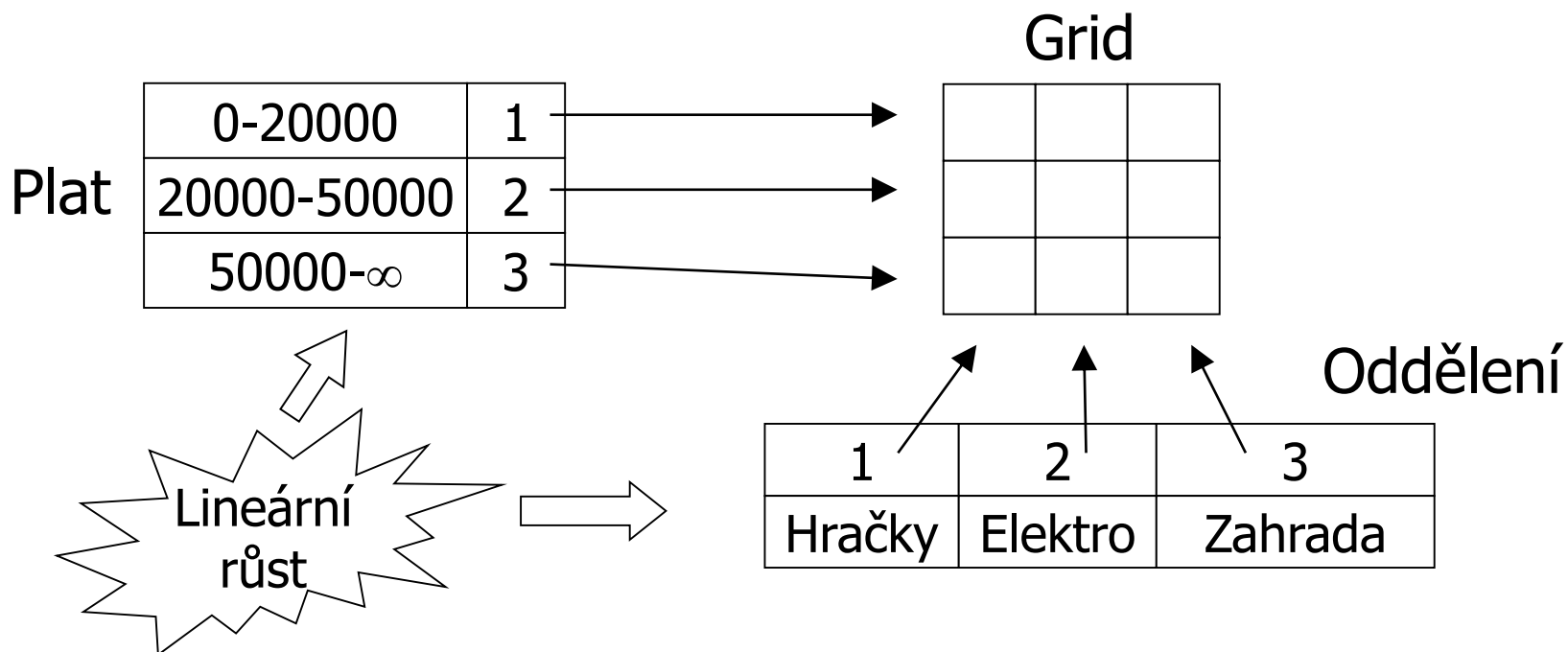
- Použití kyblíků, tj. nepřímé adresování
  - Buňka mřížky odkazuje na kyblík



- Nyní je mřížka pevné velikosti
  - Ovšem přibyla režie s odkazy

# Grid: definice mřížky

- Analýzou dat a požadavků na hledání
  - Zjistíme rozměry mřížky
  - Hodnota osy mřížky může být i interval
    - Např. číselné domény



# Grid index: hodnocení

## ■ Výhody

- Vhodné pro víceklíčové indexy

## ■ Nevýhody

- Mřížka je pevná, zabírá místo
  - Řešením může být hierarchický grid
- Volba rozsahů mřížky → rovnoměrné rozdělení dat



# Další pojmy

- Clustered index
  - = index-sekvenční soubor / B-file
  - Záznamy jsou v listech indexu a nelze je načítat jinak než prostřednictvím indexu
- Non-clustered index
  - = sekundární index / B-strom
- Pokrývající index (Covering index)
  - Význam: dotaz lze vyřešit použitím indexu a bez přístupu k celým záznamům
  - MS SQL Server: *Index se zahrnutými sloupci*
    - Není více atributovým indexem, ale obsahuje v listech hodnoty i pro další atributy
- Indexovaný pohled
  - materializovaný pohled s clustered indexem

# Návrh indexů

- Dobrá selektivita
- Co „nejkratší“ atributy
  - zvýšení arity stromu
- Pro atributy pro spojení více relací
- Raději více indexů než jeden víceatributový
- Málo indexů pro často aktualizované tabulky
- Žádné indexy pro malinké tabulky