PA152: Efektivní využívání DB

5. Hašování

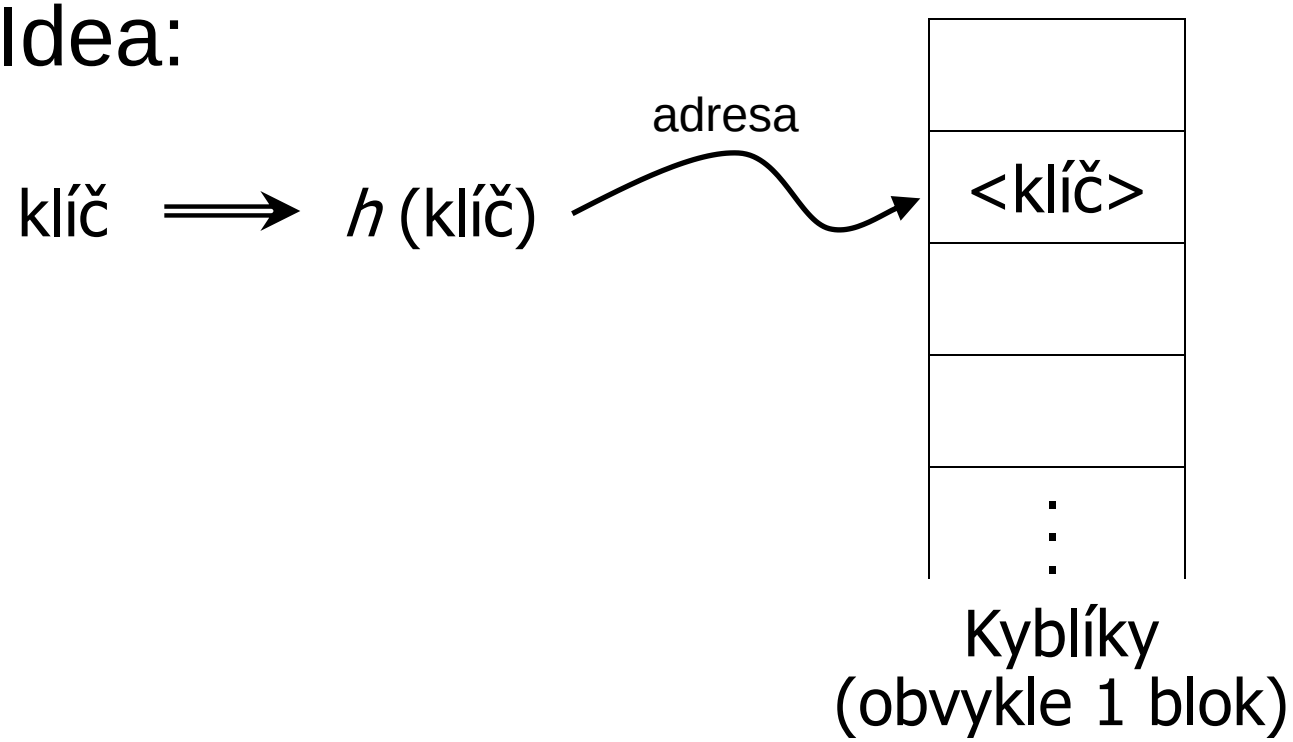
Vlastislav Dohnal

Poděkování

- Zdrojem materiálů tohoto předmětu jsou:
  - Přednášky CS245, CS345, CS345
    - Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom
    - Stanford University, California

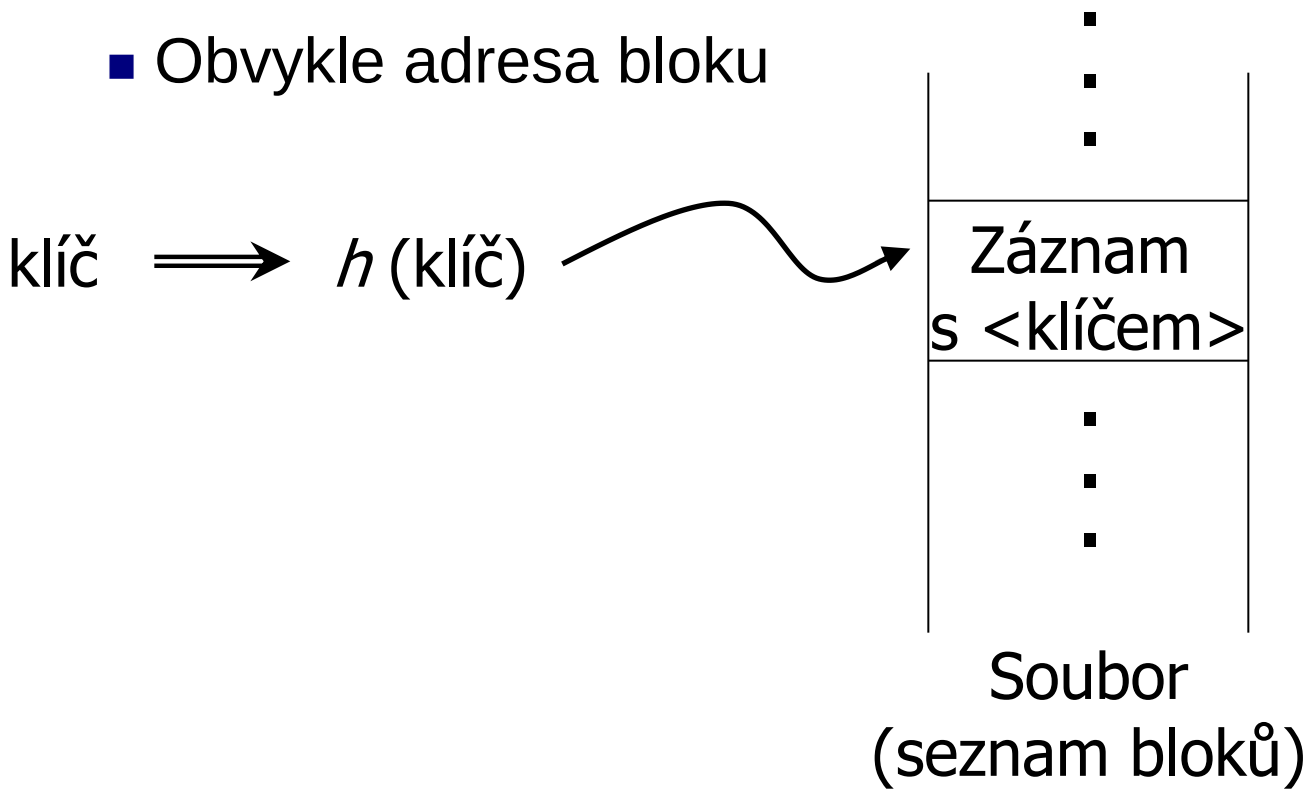
Hašování

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



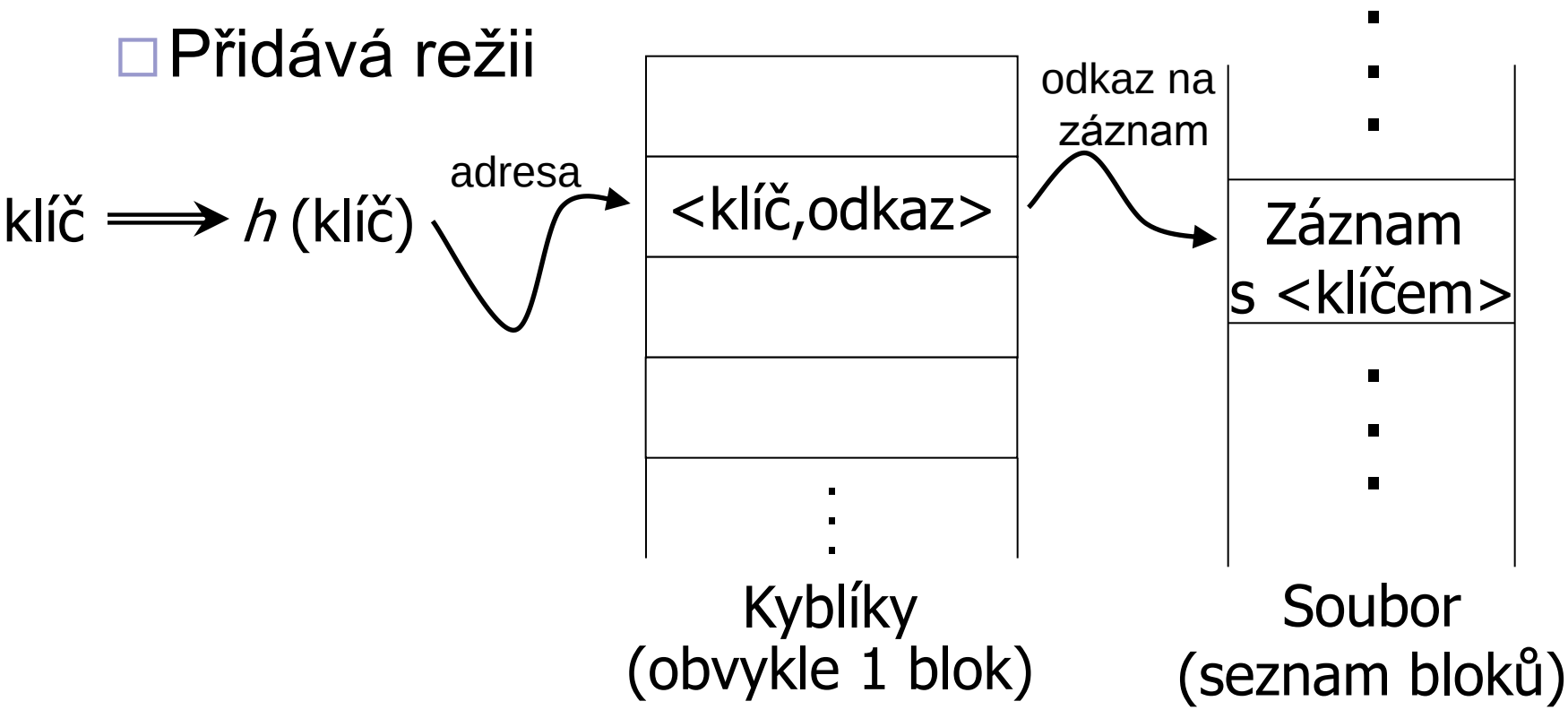
Možnosti implementace

- Přímá adresace
  - Adresa záznamu
    - Obvykle adresa bloku



Možnosti implementace

- Nepřímá adresace
  - Vhodné pro sekundární indexy
  - Přidává režii



Příklad hašovací funkce

- Klíč je řetězec znaků
  - n bajtů
- Adresní prostor
  - B kyblíků
- Hašovací funkce $h(x_0x_1\ldots x_{n-1})$
  - $(x_0+x_1+\ldots+x_{n-1}) \bmod B$
  - $(x_0*31^{n-1}+x_1*31^{n-2}+\ldots+x_{n-1}) \bmod B$
  - 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
  - Uspořádávat klíče?
    - Ano, pokud chceme zrychlit přístup
    - Ano, pokud je málo aktualizací
    - Ne, pokud je hodně aktualizací

Základní pojmy

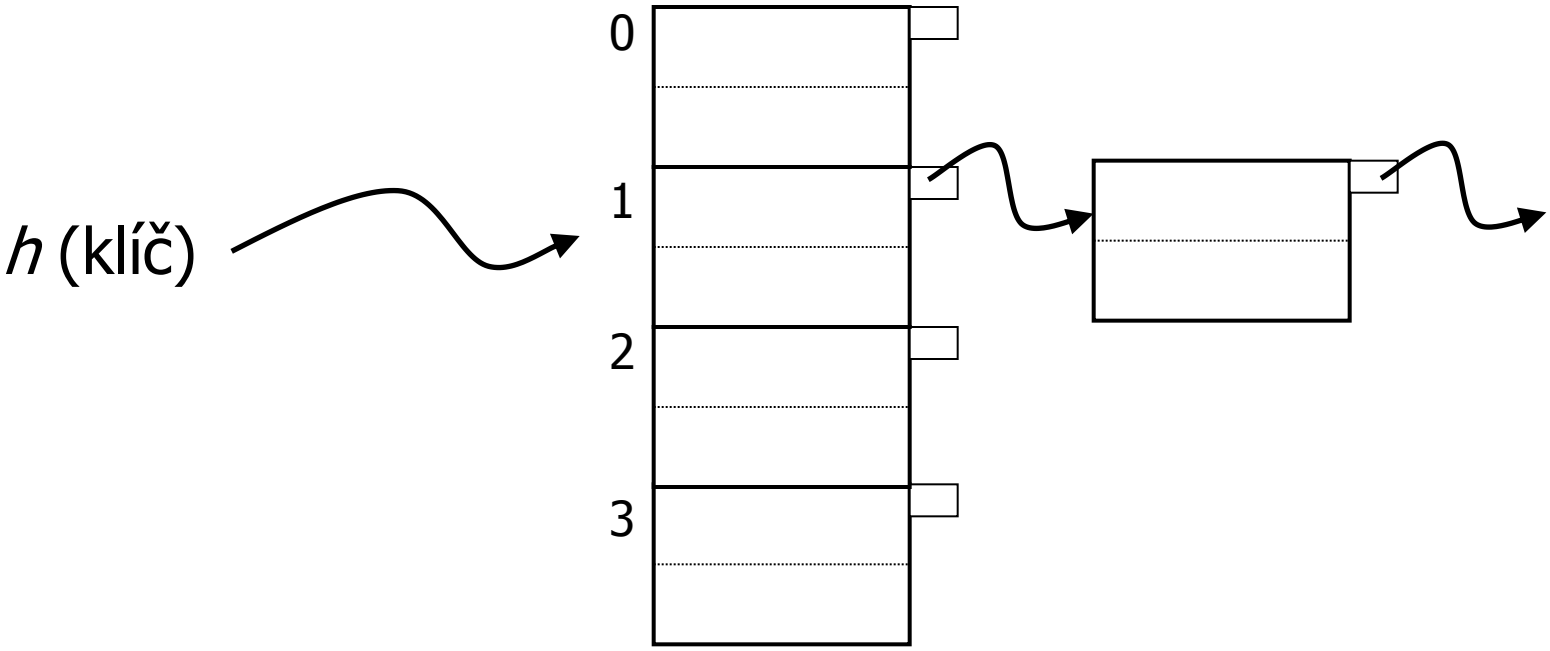
- Hašovací funkce
- Kolize
  - Na vypočtené adrese je již něco uloženo
  - Není problém, pokud lze uložit více klíčů
- Přetečení
  - Kapacita kyblíku je naplněna
  - Přetoková oblast
- Statické vs. dynamické hašování
  - Podle změny velikosti adresového prostoru

Řešení kolizí

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

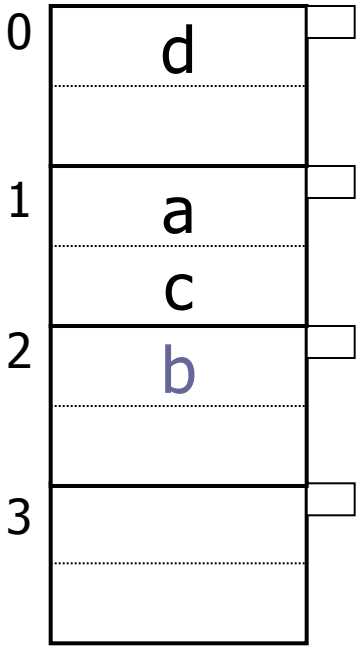
Příklad

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



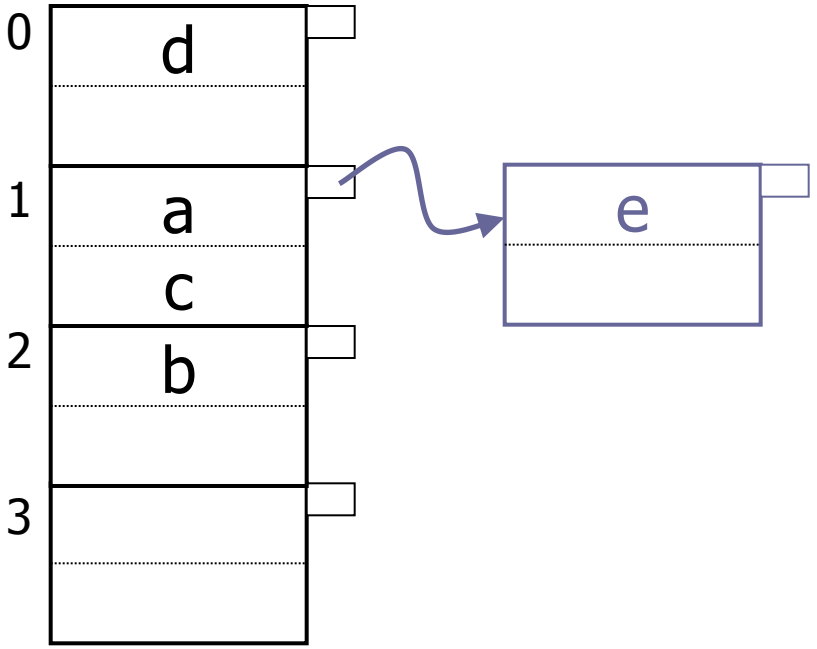
Příklad vkládání

- h(b) = 2



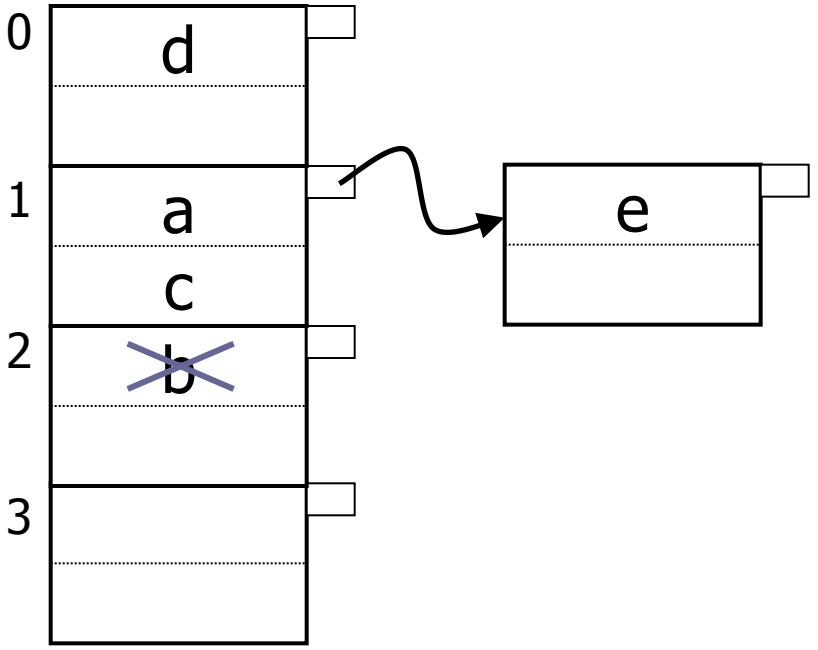
Příklad vkládání

- *h*(e) = 1



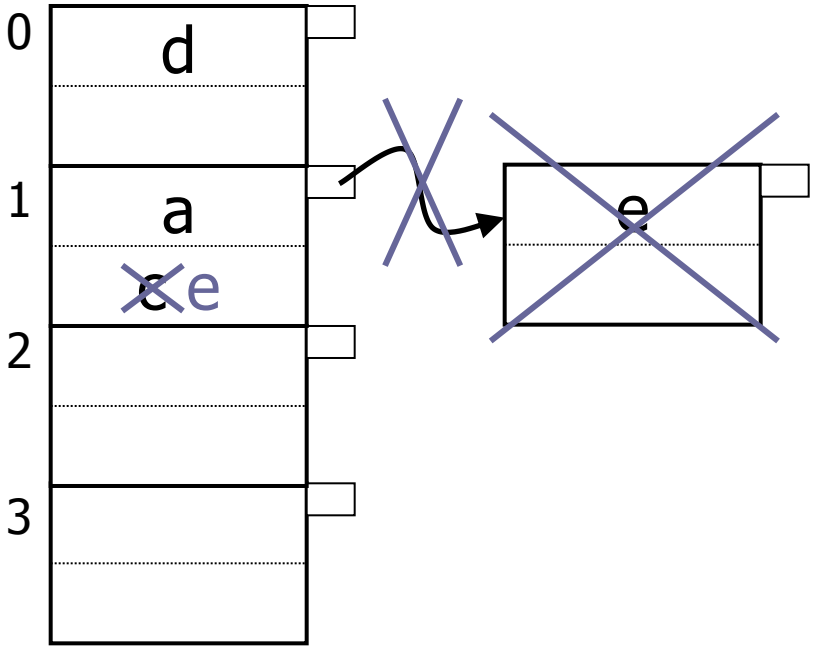
Příklad mazání

- Uvolňovat přetokové oblasti
- Smaž
  - b



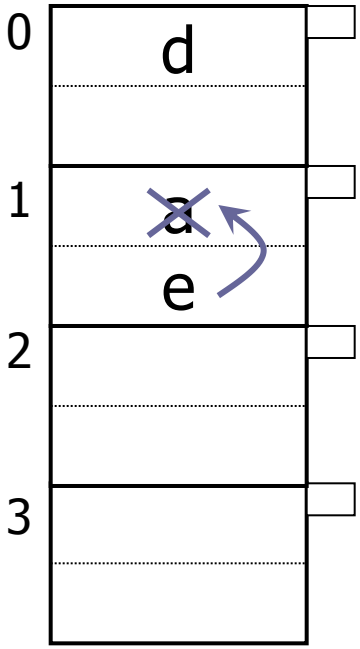
Příklad mazání

- Uvolňovat přetokové oblasti
- Smaž
  - c



Příklad mazání

- Uvolňovat přetokové oblasti
- Smaž
  - a



Empirická znalost

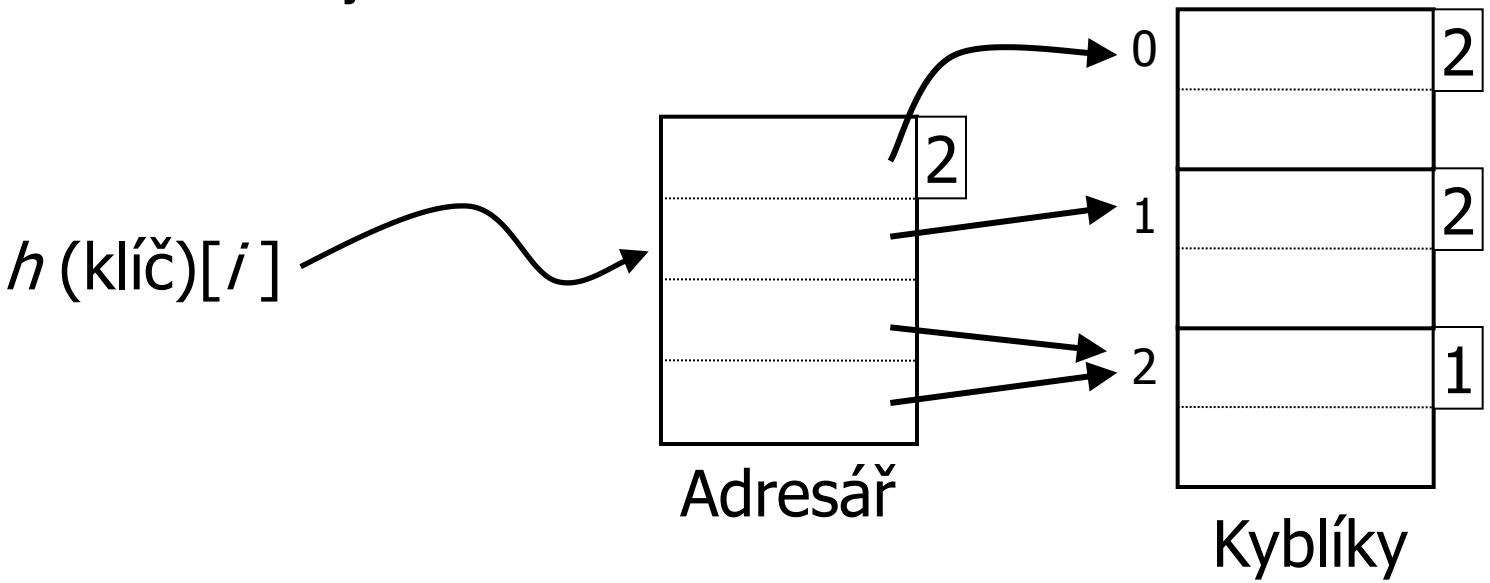
- Zaplnění udržovat mezi 50% a 80%
  - Zaplněnost = počet klíčů / max. kapacita
  - < 50% – plýtvání místem
  - ≥ 80% – příliš mnoho kolizí
    - Přetokové oblasti zpomalují vyhledávání i vkládání

Dynamická data

- Statické hašování
  - Přetoky $\rightarrow$ reorganizace
    - Tj. vytvoření nové hašovací funkce
- Dynamické hašování
  - Rozšiřitelné
  - Lineární

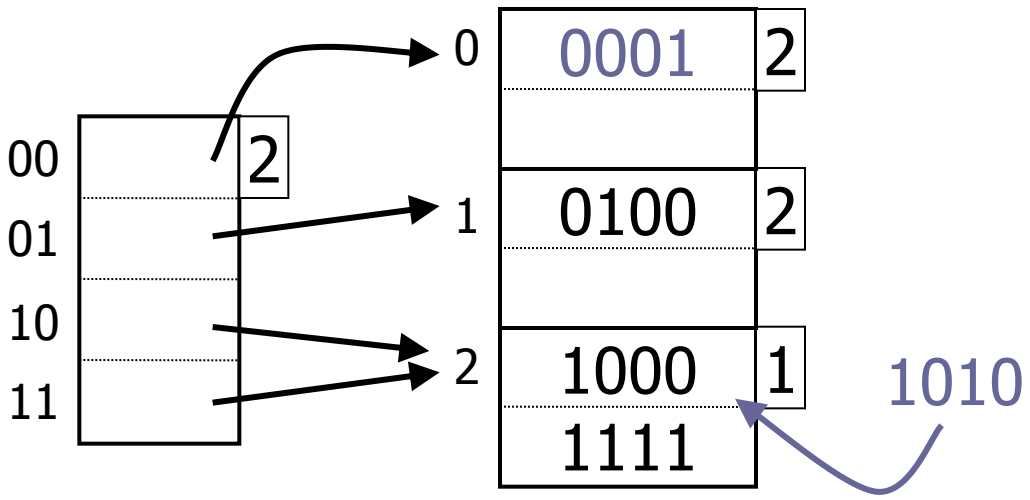
Rozšiřitelné hašování

- Idea:
  - Využití pouze několika prvních bitů adresy
  - Přidání další tabulky – adresář
    - Velikost je mocnina 2



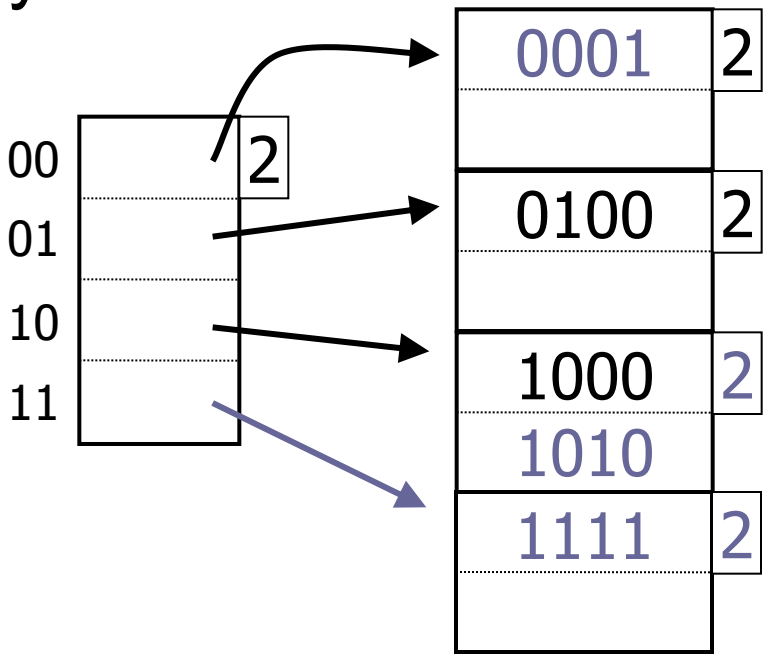
Rozšiřitelné hašování: vkládání

- Najdi kyblík
  - je volné místo → ok
  - není → rozštěpení kyblíku
    - přerozdělení obsahu
- Vložení
  - 0001
  - 1010



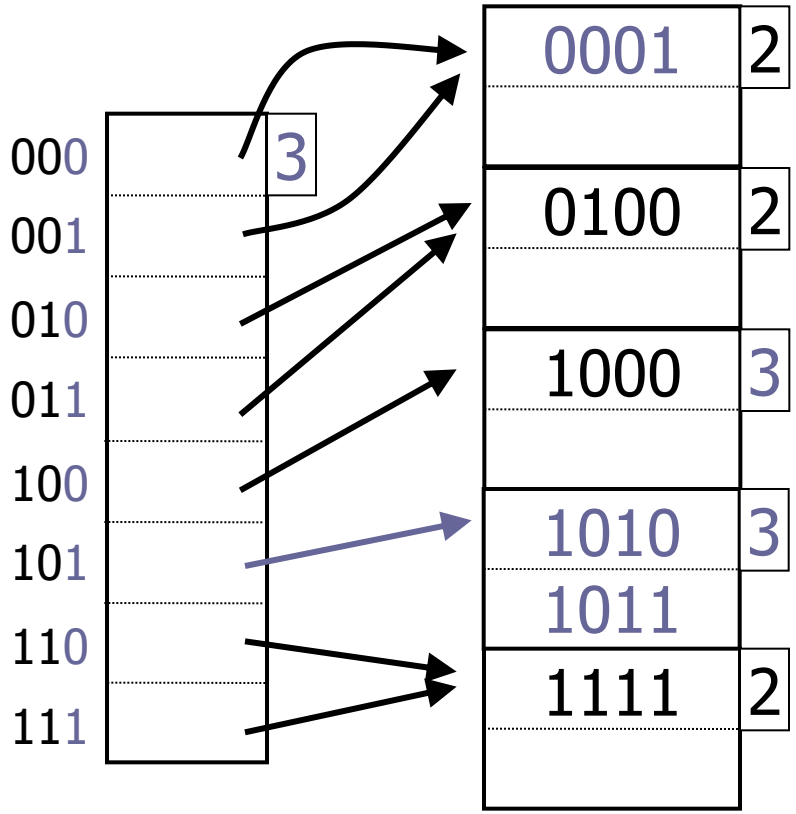
Rozšiřitelné hašování: vkládání

- Vložení
  - 1010
  - Rozštěpení kyblíku



Rozšiřitelné hašování: vkládání

- Vložení 1011
  - Adresář je plný → zdvojnásobení (přidání bitu)



Rozšiřitelné hašování: mazání

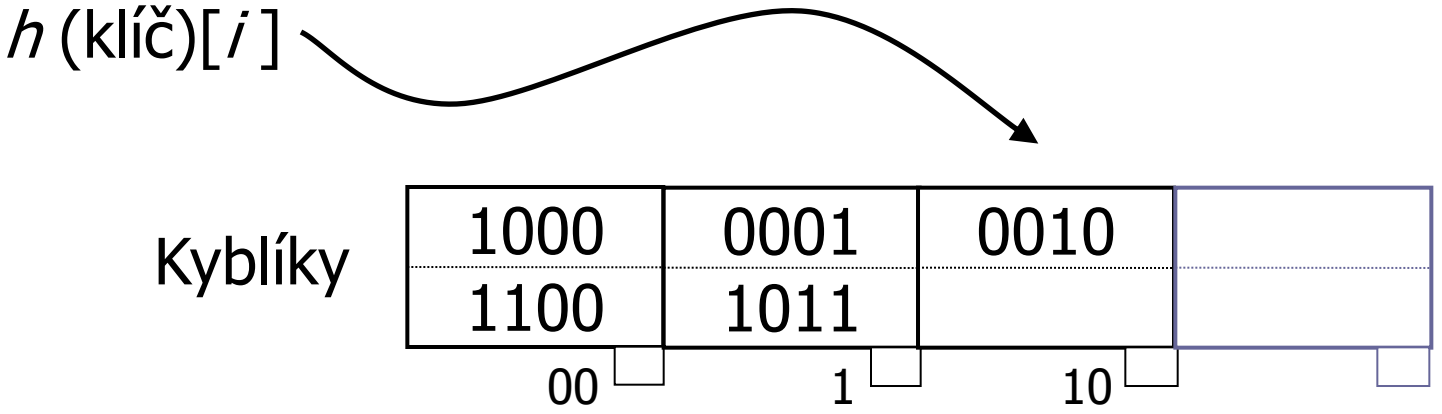
- Mazání klíče
  - Smaž klíč
  - Kyblík je prázdný
    - Nedělej nic → předpoklad nového vkládání
    - Sloučení sousedních kyblíků
      - Mající stejný prefix o jeden bit kratší
      - Může být i zmenšen adresář

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!

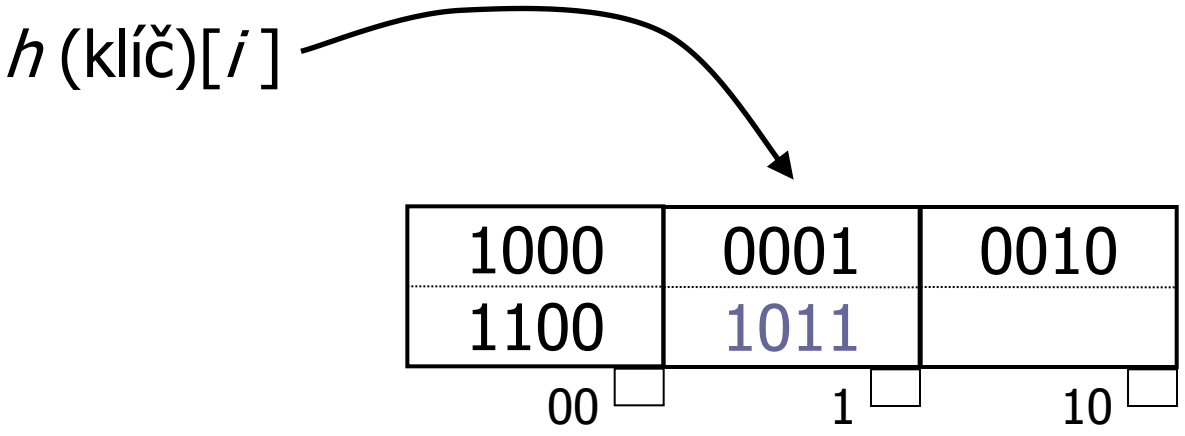
Lineární hašování

- Idea:
  - Využití pouze několika *posledních* bitů adresy
    - Konkrétně *i* bitů
  - Žádný adresář
  - Soubor roste lineárně



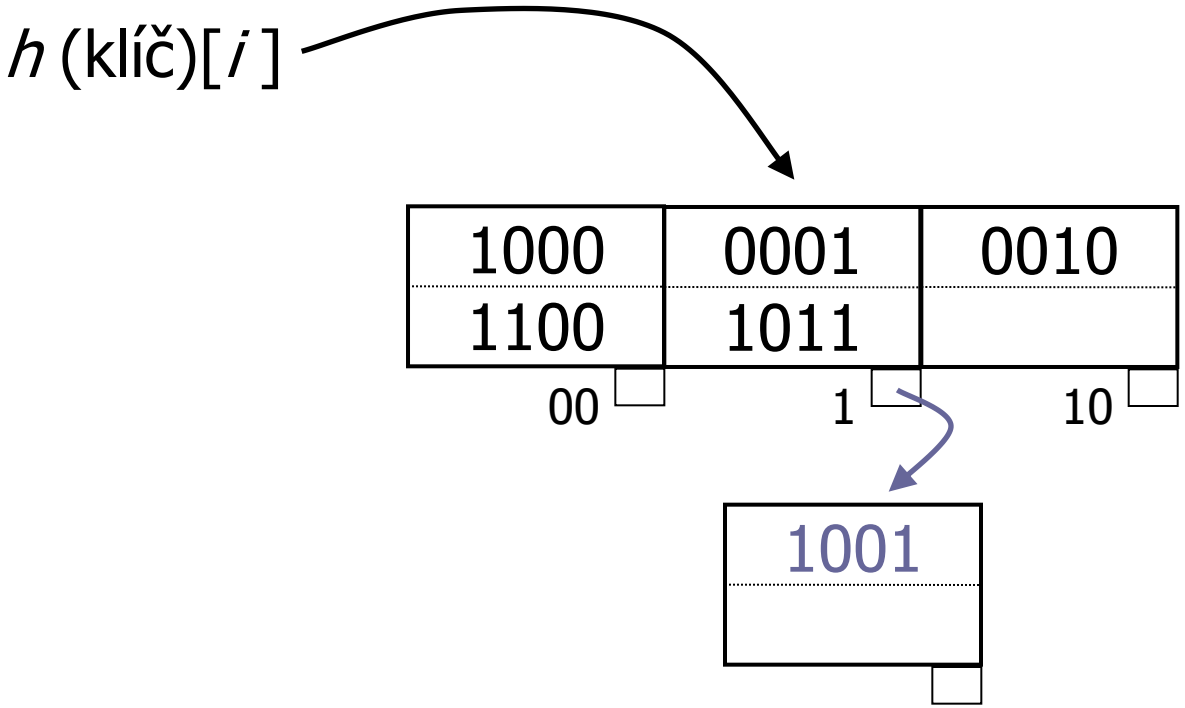
Lineární hašování: vkládání

- Parametry: i=2, n=3
- Vlož 1011
  - h(1011)[2] = 11 = m
    - Pokud $m < n$, vlož do kyblíku m
    - Jinak vlož do kyblíku $m - 2^{i-1}$



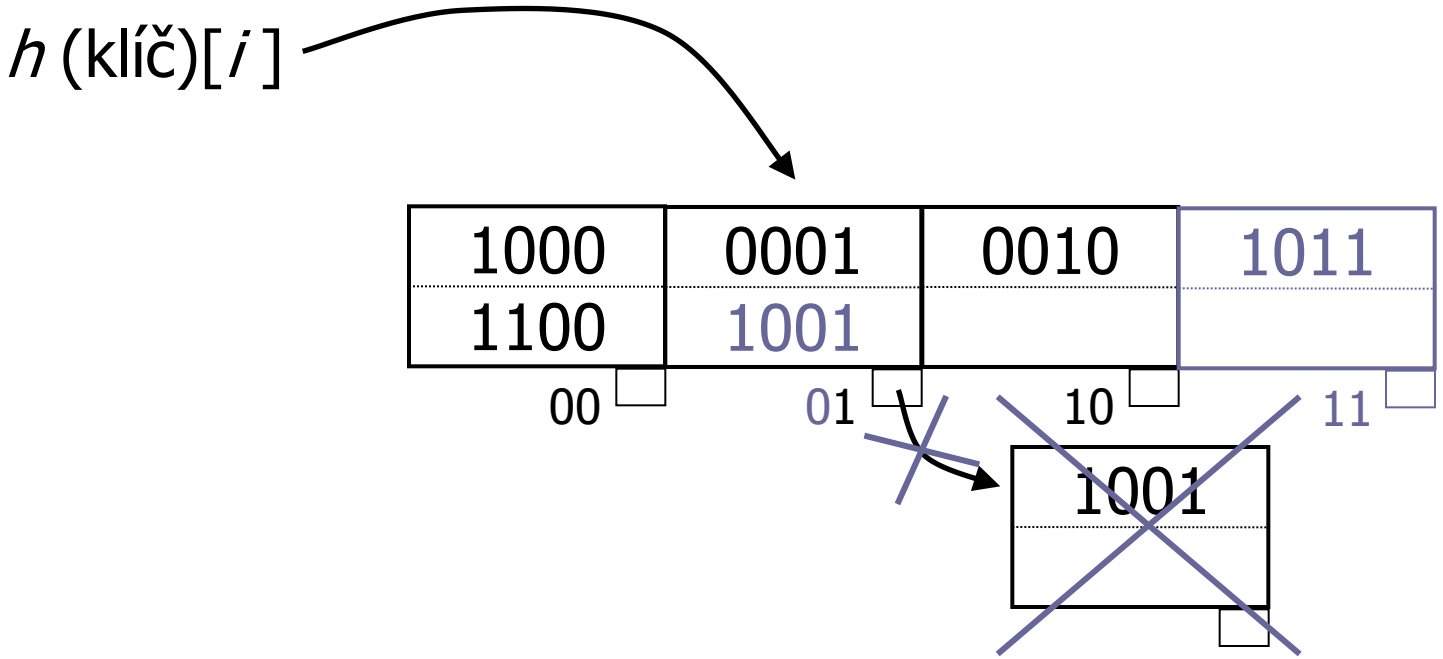
Lineární hašování: vkládání

- Vlož 1001
  - h(1001)[2] = 01
  - Není místo → přetoková oblast



Lineární hašování: štěpení kyblíku

- Při vkládání kontroluj naplnění
  - >80%, pak rozděl kyblík (jehož adresa je *0abcd…z*)
    - Resp. přidej nový → adresa je *1abcd...z*
    - Rozděl data v kyblíku *0abcd…z* mezi *[01]abcd…z*



Lineární hašování

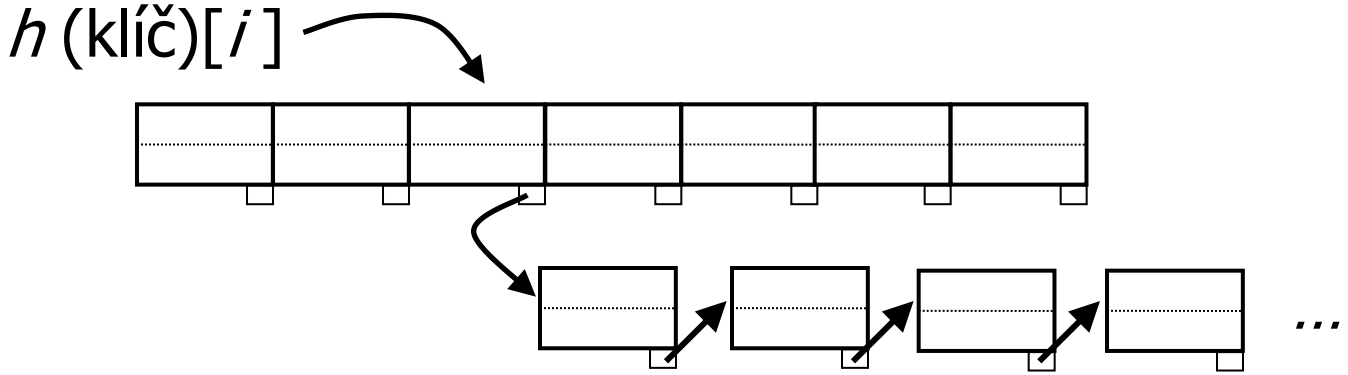
- Vkládání nového klíče
  - Může vést k přetokové oblasti
  - Může způsobit větší naplnění než 80%
    - Provede se štěpení kyblíku
    - Štěpený kyblík může být různý od kyblíku, do kterého se vkládá!
  - Po přidání 2^i*-tého* kyblíku, zvětši *i*
    - Tj. počet kyblíků překročí 2^i

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
  - Prvních 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

- Kolekce bitových polí
  - Pro každou hodnotu je jedno pole
  - Délka pole je počet záznamů
  - = invertovaný soubor
- Vhodné pro atributy s „malým“ počtem hodnot

Bitmapový index: příklad

- Relace R(F,G)

F	*G*
30	foo
30	bar
40	baz
50	foo
40	bar
30	baz

- Bitmapový index pro *G*

Hodnota	*Vektor*
foo	100100
bar	010010
baz	001001

Bitmapový index: vlastnosti

- Nevýhody
  - Paměťová náročnost
    - Pokud je klíč primárním klíčem, pak zabírá $n(n \log_2 n)$ bitů
  - Aktualizace záznamů
    - Vkládání, mazání
    - Nová hodnota $\rightarrow$ nové bitové pole
- Výhody
  - Rychlé operace na bitech (AND, OR)
  - Lze použít pro rozsahové dotazy
  - Snadné kombinace více indexů dohromady

Bitmapový index: komprese

- Zmenšení polí
  - Málo 1, hodně 0
  - Obvykle Run-Length Encoding (RLE)
    - Sekvence *i nul* následovaná *jedničkou*
    - Kódování je uložení čísla *i* binárně
- Příklad
  - Kód: 1110110100101011
  - Sekvence: 0000 0000 0000 0110 0001
    - Najdi první nulu → počet bitů pro uložení *i*
    - Načti *i*
- Vlastnost
  - Sekvence vždy končí jedničkou
  - Chybějící nuly lze doplnit podle počtu záznamů

Bitmapový index: operace

- Bitové operace
  - AND, OR
- RLE řetězce
  - Dekomprimovat → snadné
  - Bez dekomprese
    - Složitější algoritmus, ale možné
    - AND: čísla *i* v kódech se musí shodovat
    - OR: analogicky…

Bitmapový index: použití

- 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^+-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 *x* (pořadí záznamu)
    - Sekvenční soubor $\rightarrow$ snadné
    - Sekundární index pro čísla záznamů

Bitmapový index: řešení

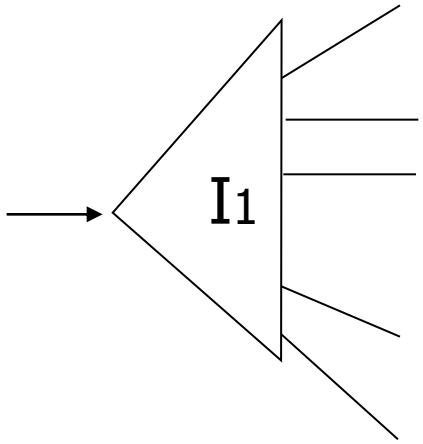
- Ad 3: (Aktualizace záznamů, co s indexem?)
  - Čísla záznamů jsou fixní
    - Mazání záznamu
      → náhrobek v souboru a změna 1 na 0 v jednom bitovém poli
      → smazání 1 bitu ve *všech* polích
    - Vkládání záznamu
      → přidej na konec souboru (nové číslo záznamu)
      → do správného bitového pole připojit 1
      → pole nemusí existovat, vytvoř nové
  - Čísla záznamů nejsou fixní
    - Reorganizace všech polí
    - Málo používaná verze

Víceklíčový index

- Index pro více atributů
- Důvod:
  SELECT jméno, plat FROM zam
  WHERE oddělení='Hračky' AND plat < 10000
- Řešení
  a) Index pro jeden atribut + filtrování
  b) Oddělené indexy pro atributy + průnik vyhovujících
  c) Index v indexu
  d) Spojení klíčů v jeden

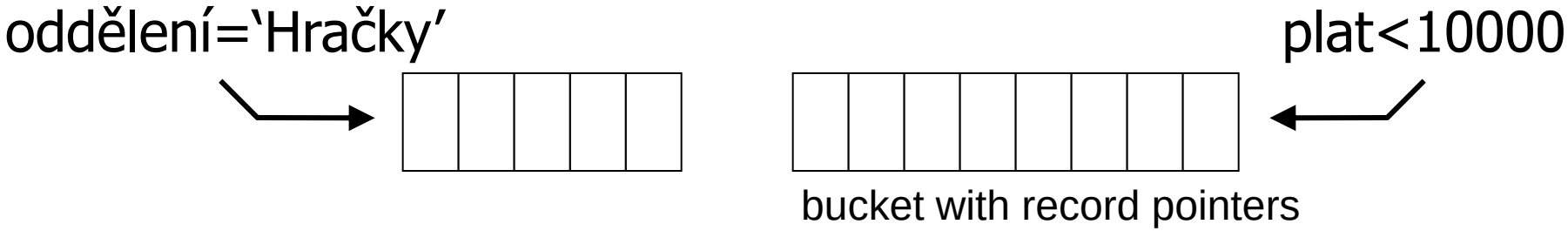
Index pro jeden atribut

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



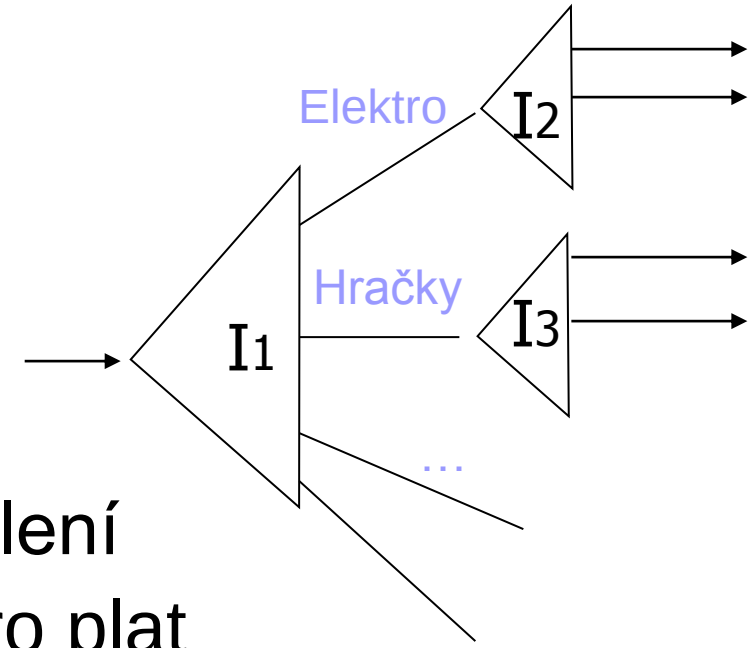
Nezávislé indexy

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



Index v indexu

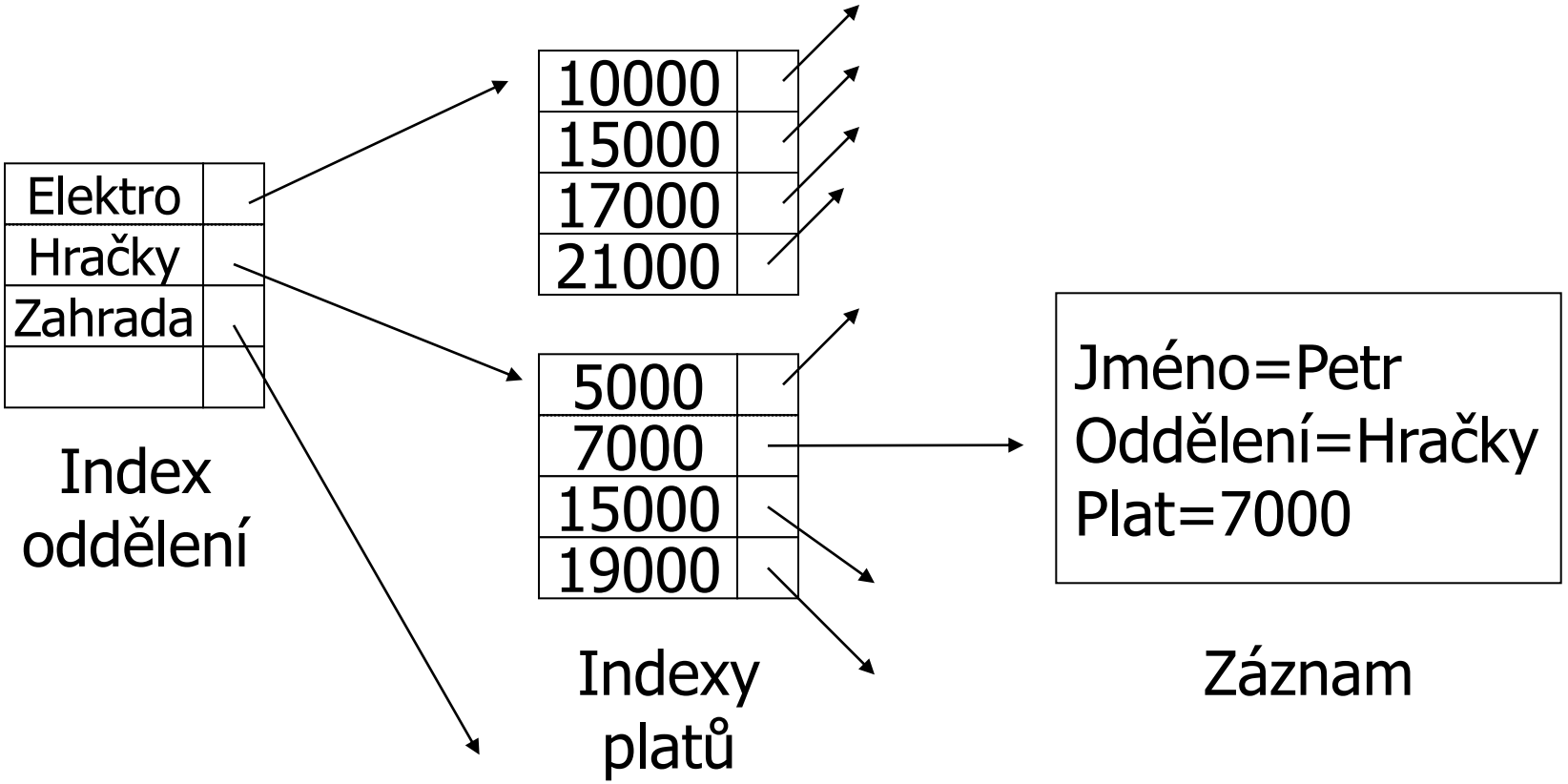
- Index pro první atribut
  - V listu je odkaz na index pro druhý atribut



  - I_1 pro oddělení
  - $I_{x,}$ x=2..k pro plat
    - I_2 obsahuje pouze záznamy se stejným oddělením (Elektro)

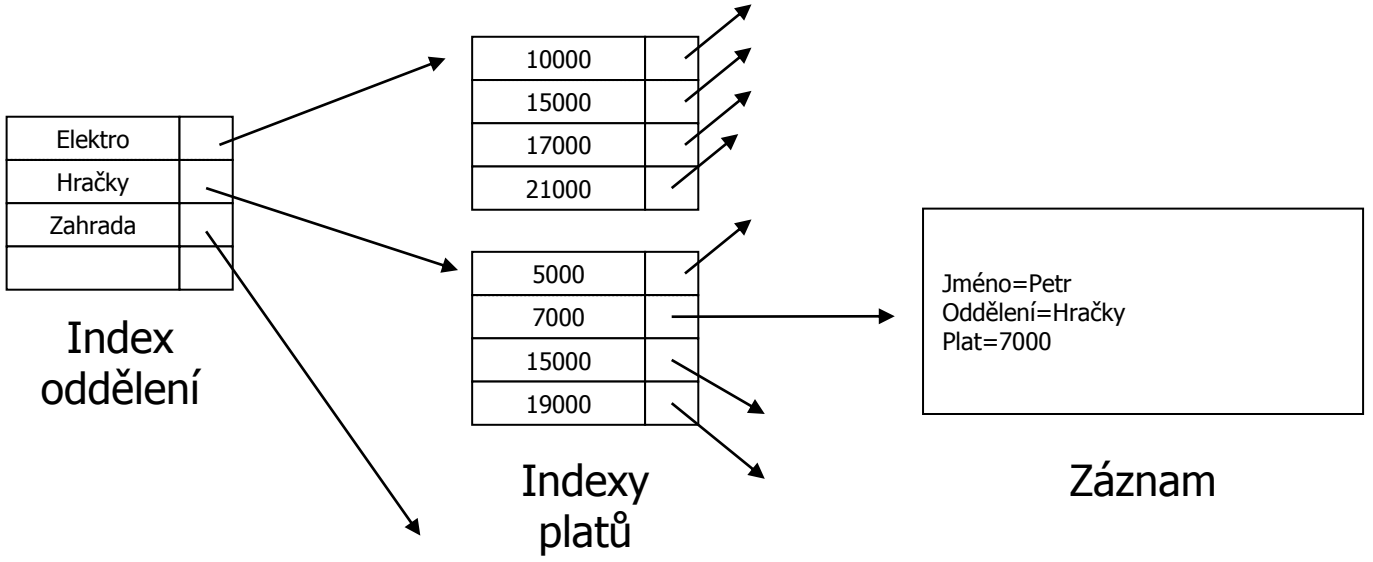
Index v indexu: příklad

- 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

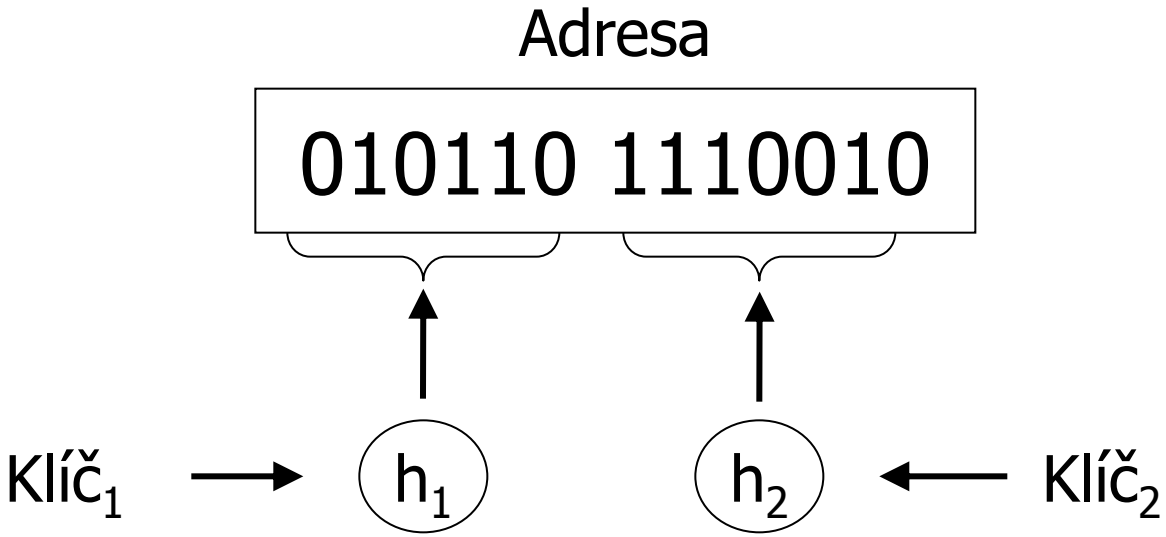


Spojení klíčů v jeden

- Podobné indexu pro jeden klíč
  - Hodnota klíče je spojená
    - Spojení řetězců, vynásobení čísel, …
- V indexování
  - 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(Elektro) =0
  - h_1(Hračky) =1
  - h_1(Zahrada) =1
- Plat
  - h_2(10000) =01
  - h_2(20000) =11
  - h_2(30000) =10
  - h_2(40000) =00
- Záznamy ke vložení
  - <Petr,Elektro,10000>
  - <Jan,Hračky,10000>
  - <Alice,Zahrada,30000>

000	
001	<Petr,…>
010	
011	
100	
101	<Jan,…>
110	<Alice,…>
111	

Dělená hašovací funkce: příklad

- Oddělení
  - h_1(Elektro) =0
  - h_1(Hračky) =1
  - h_1(Zahrada) =1
- Plat
  - h_2(10000) =01
  - h_2(20000) =11
  - h_2(30000) =10
  - h_2(40000) =00
- Najdi
  - zaměstnance z oddělení hraček a platem 40000.

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

Dělená hašovací funkce: příklad

- Oddělení
  - h_1(Elektro) =0
  - h_1(Hračky) =1
  - h_1(Zahrada) =1
- Plat
  - h_2(10000) =01
  - h_2(20000) =11
  - h_2(30000) =10
  - h_2(40000) =00
- Najdi
  - zaměstnance s platem 30000

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

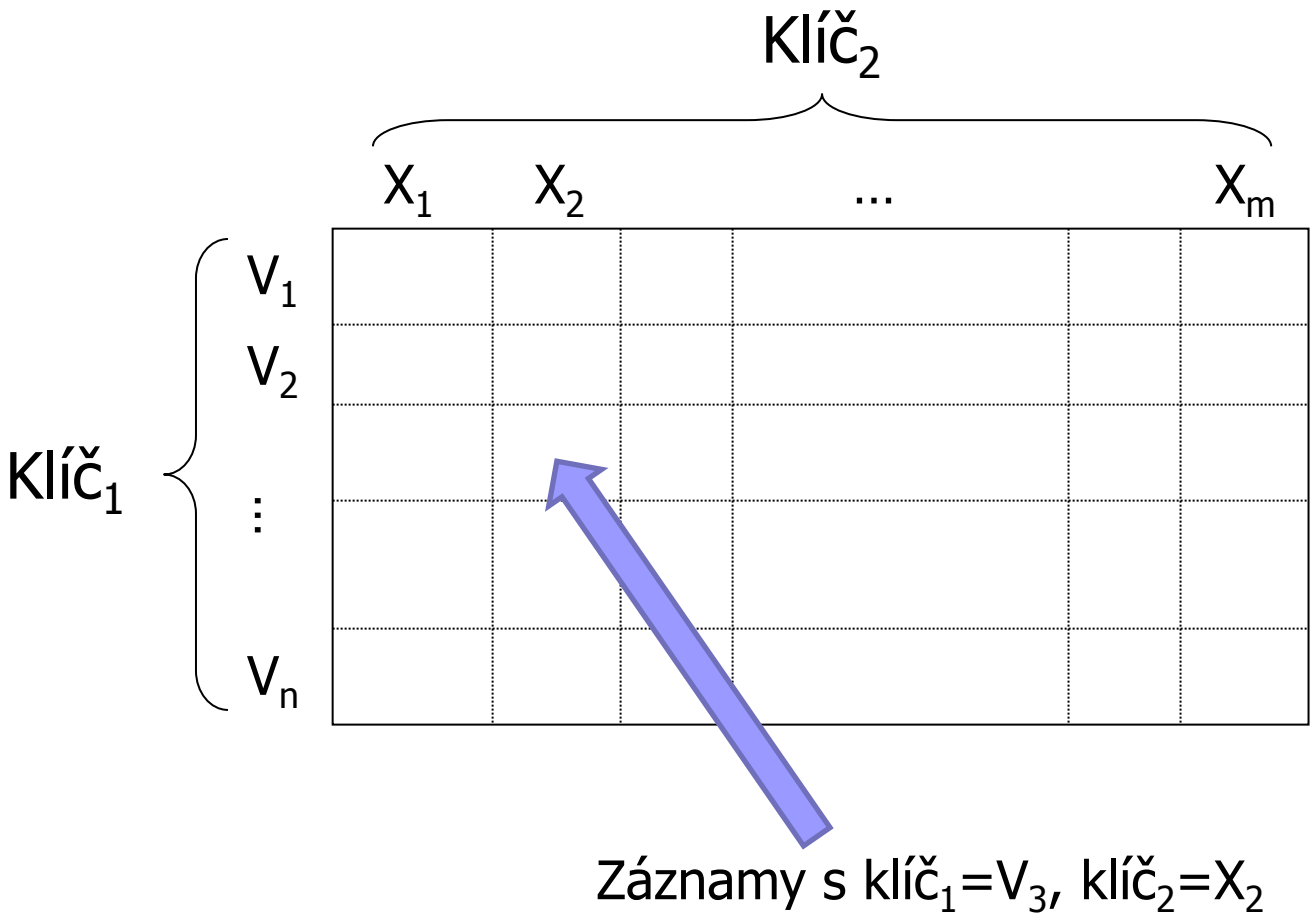
Dělená hašovací funkce: příklad

- Oddělení
  - h_1(Elektro) =0
  - h_1(Hračky) =1
  - h_1(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,…>

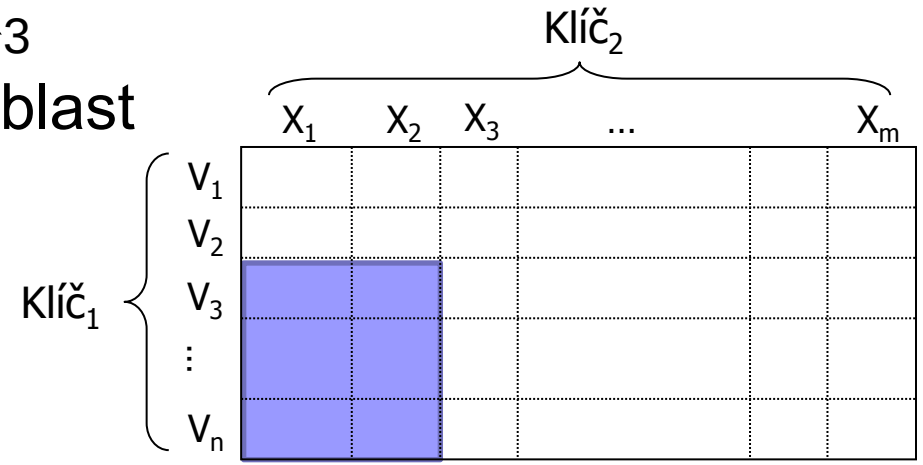
Jiný víceklíčový index

- Grid (mřížka)
- Idea:



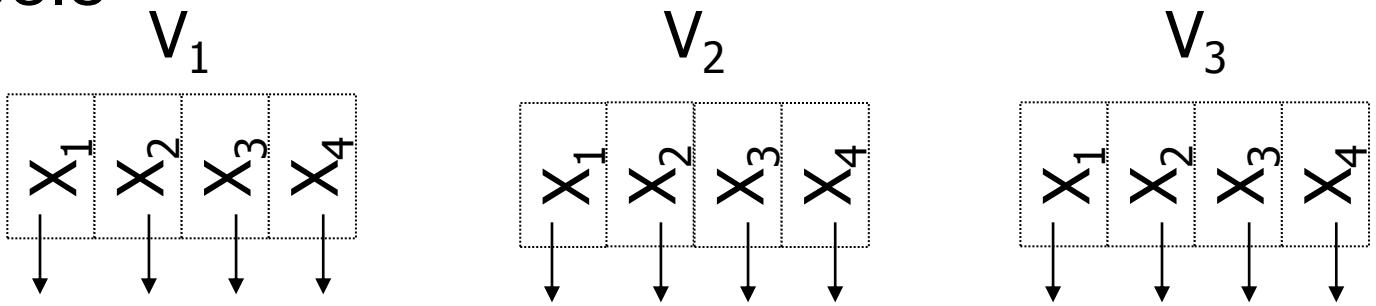
Grid: vlastnosti

- Rychlé pro přesné dotazy
  - $klíč_1 = V_i \wedge klíč_2 = X_j$
  - $klíč_1 = V_i$
  - $klíč_2 = X_j$
- Rozsahové dotazy
  - $klíč_1 \geq V_3 \wedge klíč_2 < X_3$
    - Vznikne čtvercová oblast



Grid: implementace

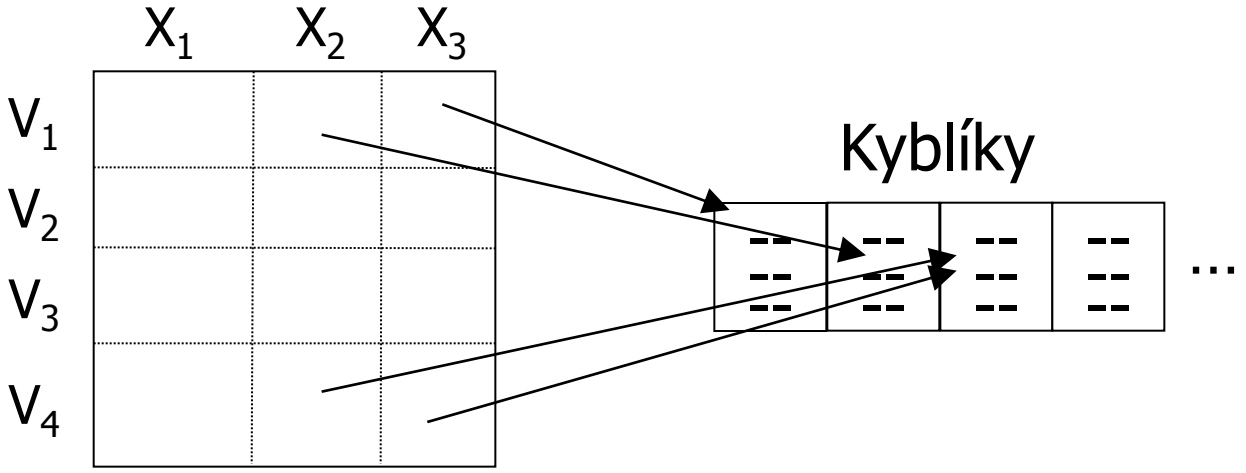
- Jak ukládat mřížku na disku?
  - Jako pole



  - Problém: rozměr mřížky vs. velikost buňky
- Nevýhoda
  - Potřeba pevného rozměru mřížky pro výpočet indexu políčka <V_x,X_y> v poli.
  - Omezená velikost buňky

Grid: implementace

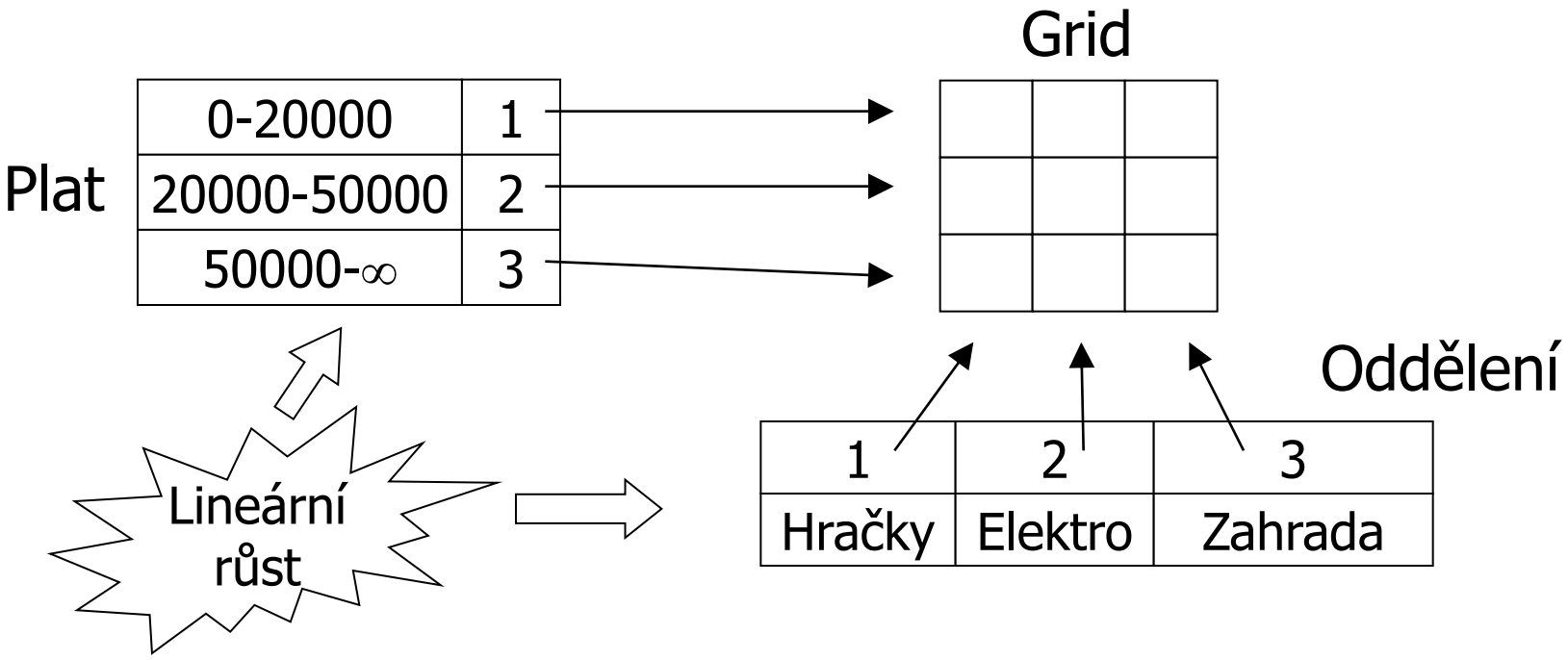
- Použití kyblíků, tj. nepřímé adresování
  - Buňka mřížky odkazuje na kyblík



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

Grid: definice mřížky

- Analýzou dat a požadavků na hledání
  - Zjistíme rozměry mřížky
  - Políčko mřížky může být i interval hodnot
    - 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