Algoritmizace

Hashing II

Jiří Vyskočil, Marko Genyg-Berezovskyj

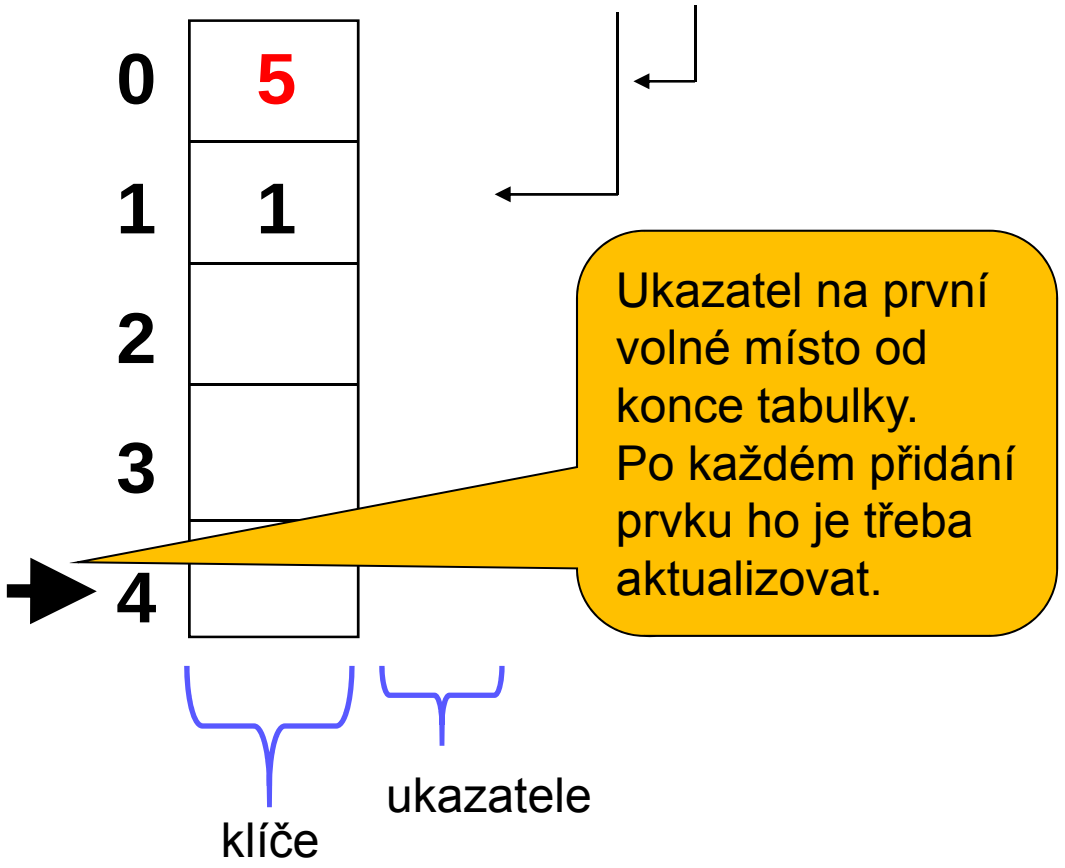
2010

Srůstající hashování (coalesced hashing)

- Znám předem počet prvků (odhad)
- Z důvodů efektivity nechci ukazatele (mezi prvky). Na jednu pozici tabulky připadne právě jeden ukazatel.
- => kolizní prvky (synonyma) se musejí „nějak" ukládat přímo do hashovací tabulky. Ukazatele v tabulce umožňují procházet pouze prvky jedné skupiny. Nedochází zde k tak velkému propojování do clusterů jako v případě otevřeného hashování. Dochází zde pouze k tzv. srůstání.
  - standardní srůstající hashování
    - LISCH (late insert standard coalesced hashing)
    - EISCH (early insert standard coalesced hashing)
  - srůstající hashování s pomocnou pamětí
    - LICH (late insert coalesced hashing)
    - EICH (early insert coalesced hashing)
    - VICH (variable insert coalesced hashing)

Standardní srůstající hashování - LISCH

- $h(k) = k \bmod 5$
- posloupnost: 1, 5, 21, 10, 15

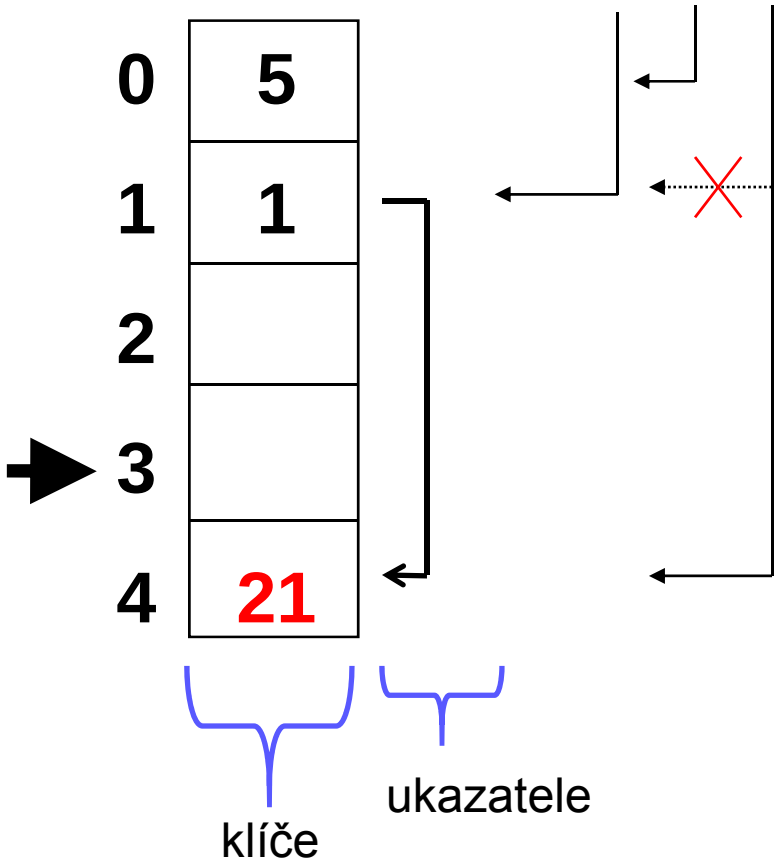


Postup:

1. $i = h(k)$;
2. Prohledej řetězec začínající na místě i a pokud nenajdeš k, přidej ho do tabulky na první volné místo od konce tabulky a připoj ho do řetězce na poslední místo.

Standardní srůstající hashování - LISCH

- $h(k) = k \bmod 5$
- posloupnost: 1, 5, 21, 10, 15

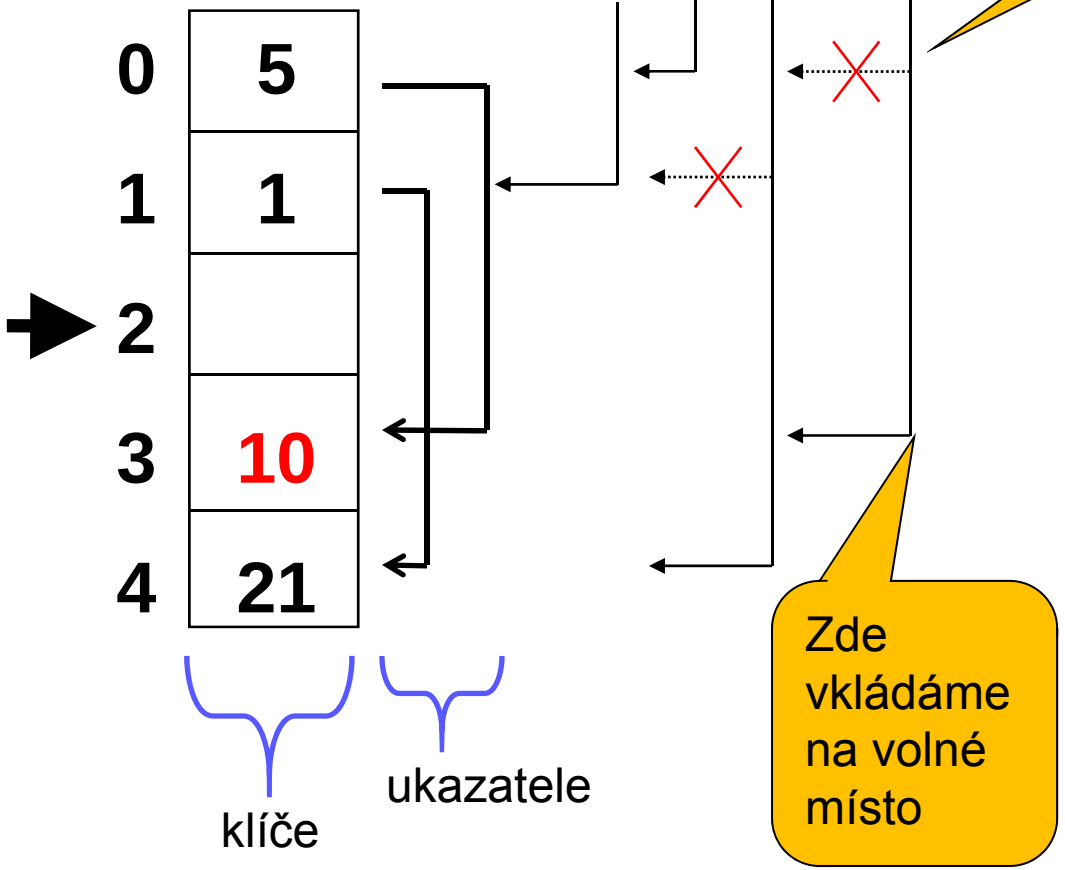


Postup:

1. $i = h(k)$;
2. Prohledej řetězec začínající na místě i a pokud nenajdeš k, přidej ho do tabulky na první volné místo od konce tabulky a připoj ho do řetězce na poslední místo.

Standardní srůstající hashování - LISCH

- $h(k) = k \bmod 5$
- posloupnost: 1, 5, 21, 10, 15



Zde procházíme řetězec prvků.

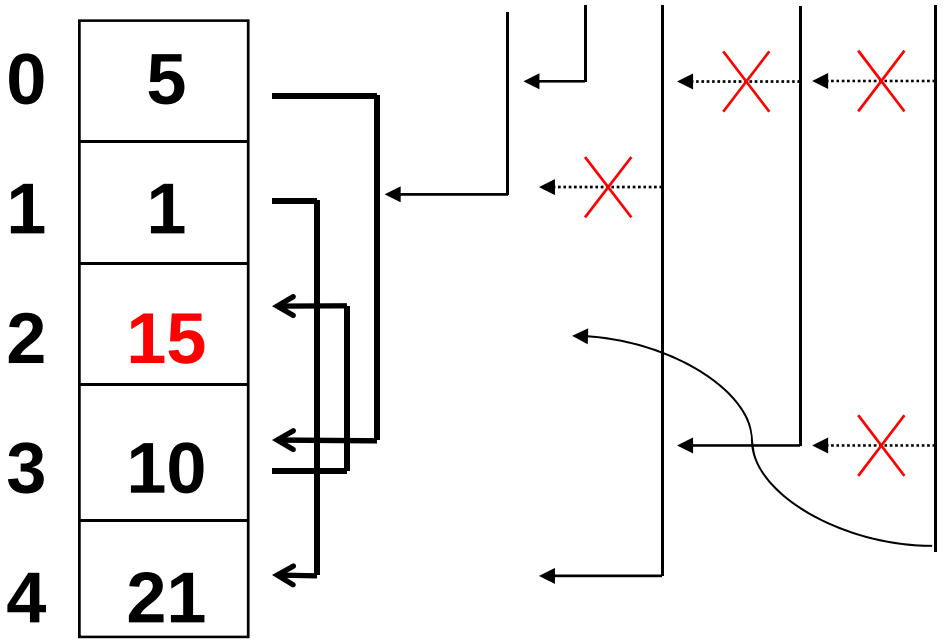
Zde vkládáme na volné místo

Postup:

1. $i = h(k)$;
2. Prohledej řetězec začínající na místě *i* a pokud nenajdeš *k*, přidej ho do tabulky na první volné místo od konce tabulky a připoj ho do řetězce na poslední místo.

Standardní srůstající hashování - LISCH

- $h(k) = k \bmod 5$
- posloupnost: 1, 5, 21, 10, 15



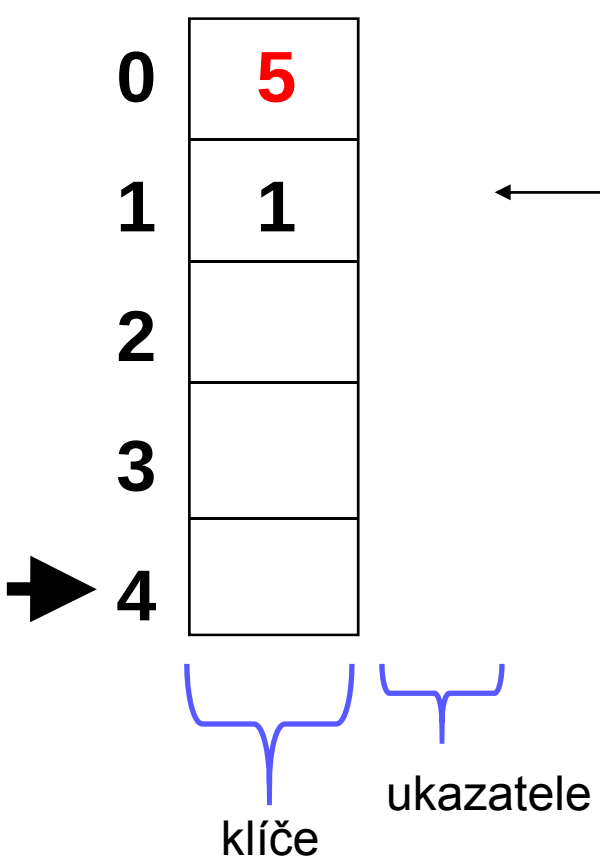
Tabulka je zaplněna.

Postup:

1. $i = h(k)$;
2. Prohledej řetězec začínající na místě i a pokud nenajdeš k, přidej ho do tabulky na první volné místo od konce tabulky a připoj ho do řetězce na poslední místo.

Standardní srůstající hashování - EISCH

- $h(k) = k \bmod 5$
- posloupnost: 1, 5, 21, 10, 15

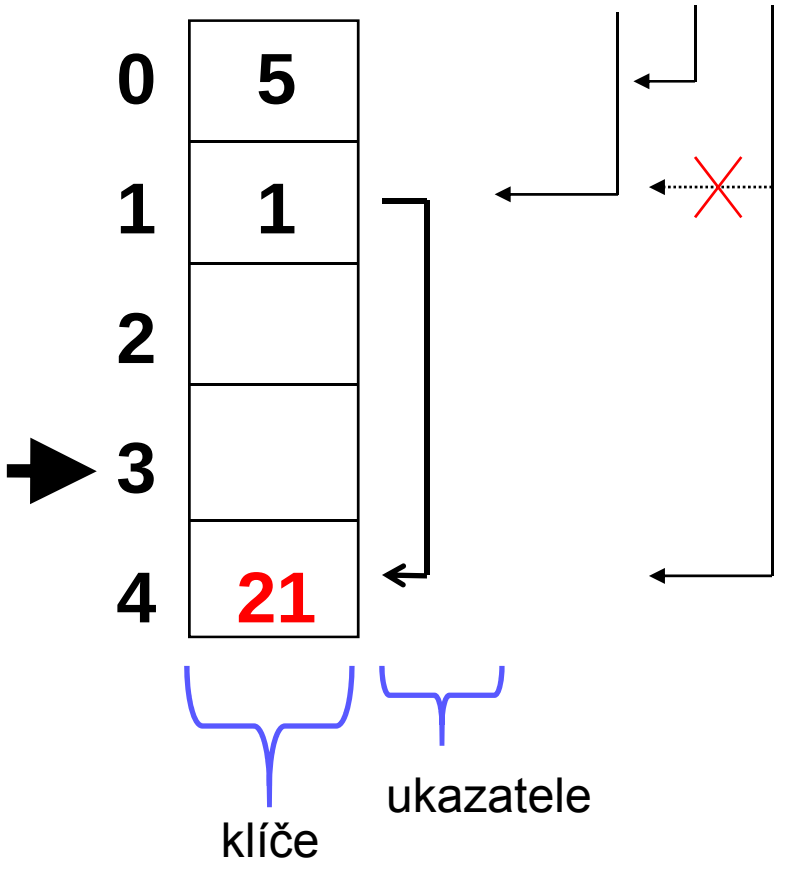


Postup:

1. $i = h(k)$;
2. Prohledej řetězec začínající na místě i a pokud nenajdeš k, přidej ho do tabulky na první volné místo od konce tabulky a připoj ho do řetězce za 1. místo.

Standardní srůstající hashování - EISCH

- $h(k) = k \bmod 5$
- posloupnost: 1, 5, 21, 10, 15

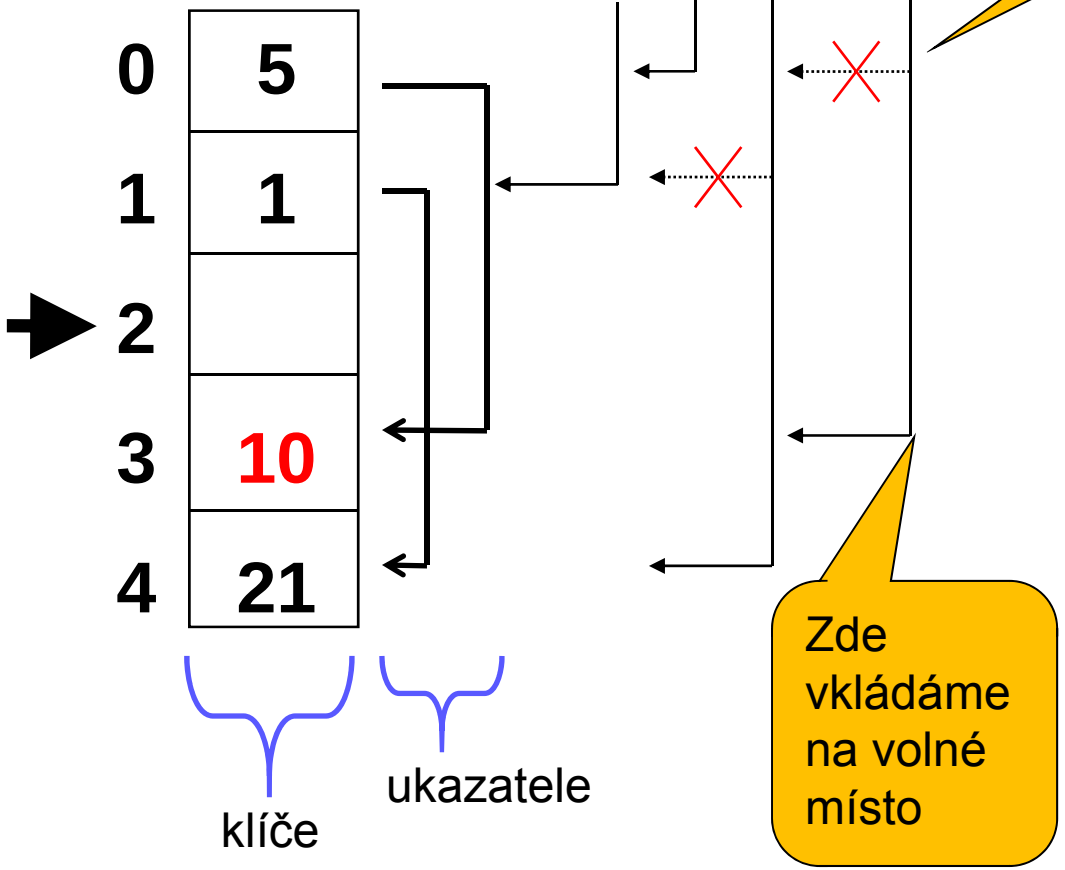


Postup:

1. $i = h(k)$;
2. Prohledej řetězec začínající na místě i a pokud nenajdeš k, přidej ho do tabulky na první volné místo od konce tabulky a připoj ho do řetězce za 1. místo.

Standardní srůstající hashování - EISCH

- $h(k) = k \bmod 5$
- posloupnost: 1, 5, 21, 10, 15



Zde procházíme řetězec prvků.

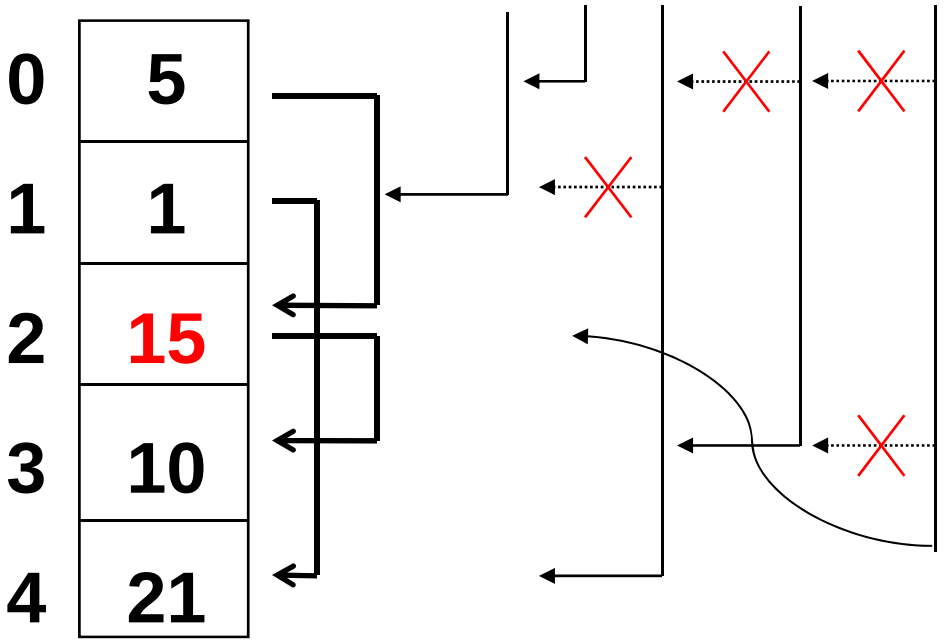
Zde vkládáme na volné místo

Postup:

1. $i = h(k)$;
2. Prohledej řetězec začínající na místě *i* a pokud nenajdeš *k*, přidej ho do tabulky na první volné místo od konce tabulky a připoj ho do řetězce za 1. místo.

Standardní srůstající hashování - EISCH

- $h(k) = k \bmod 5$
- posloupnost: 1, 5, 21, 10, 15

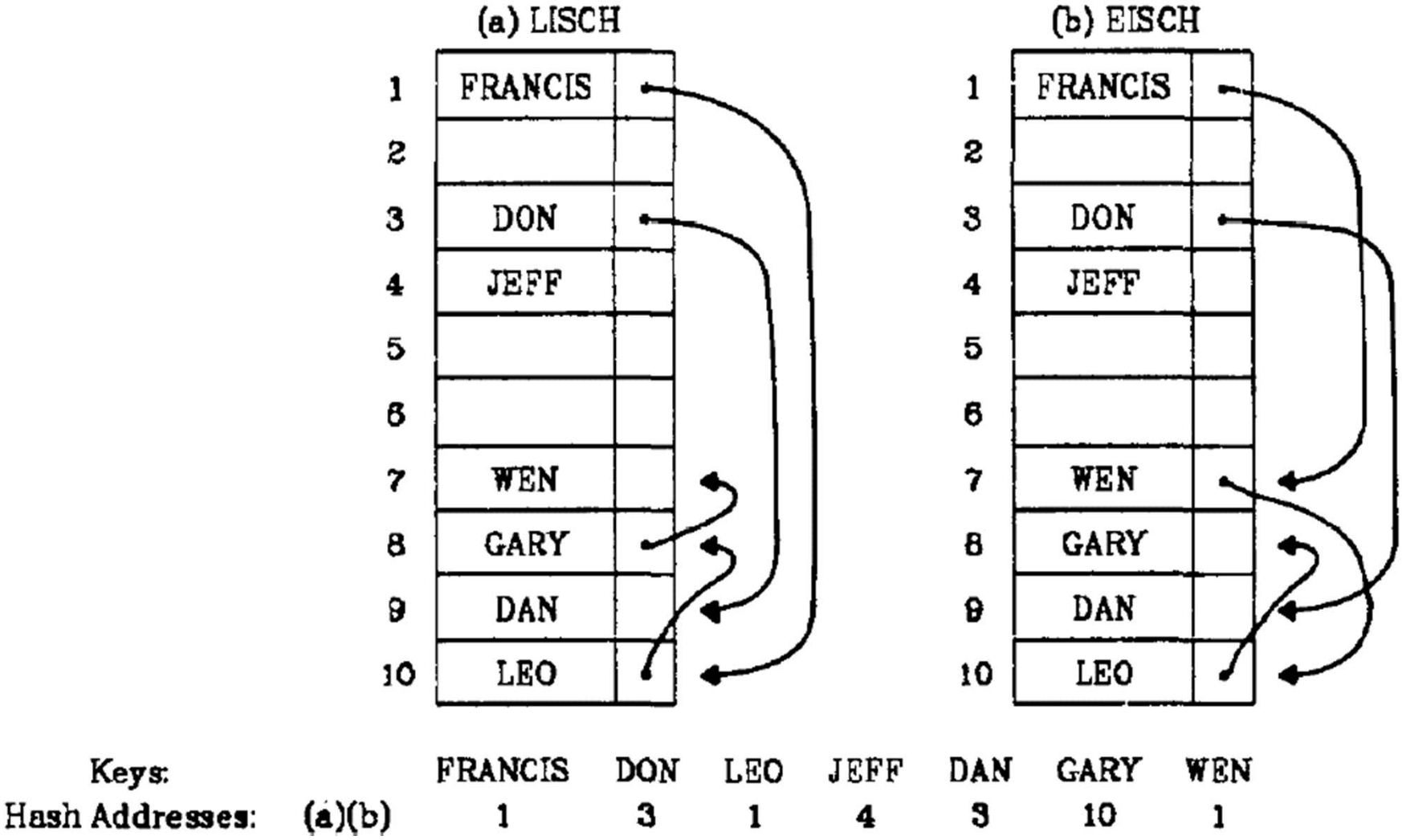


Postup:

1. $i = h(k)$;
2. Prohledej řetězec začínající na místě i a pokud nenajdeš k, přidej ho do tabulky na první volné místo od konce tabulky a připoj ho do řetězce za 1. místo.

➡ Tabulka je zaplněna.

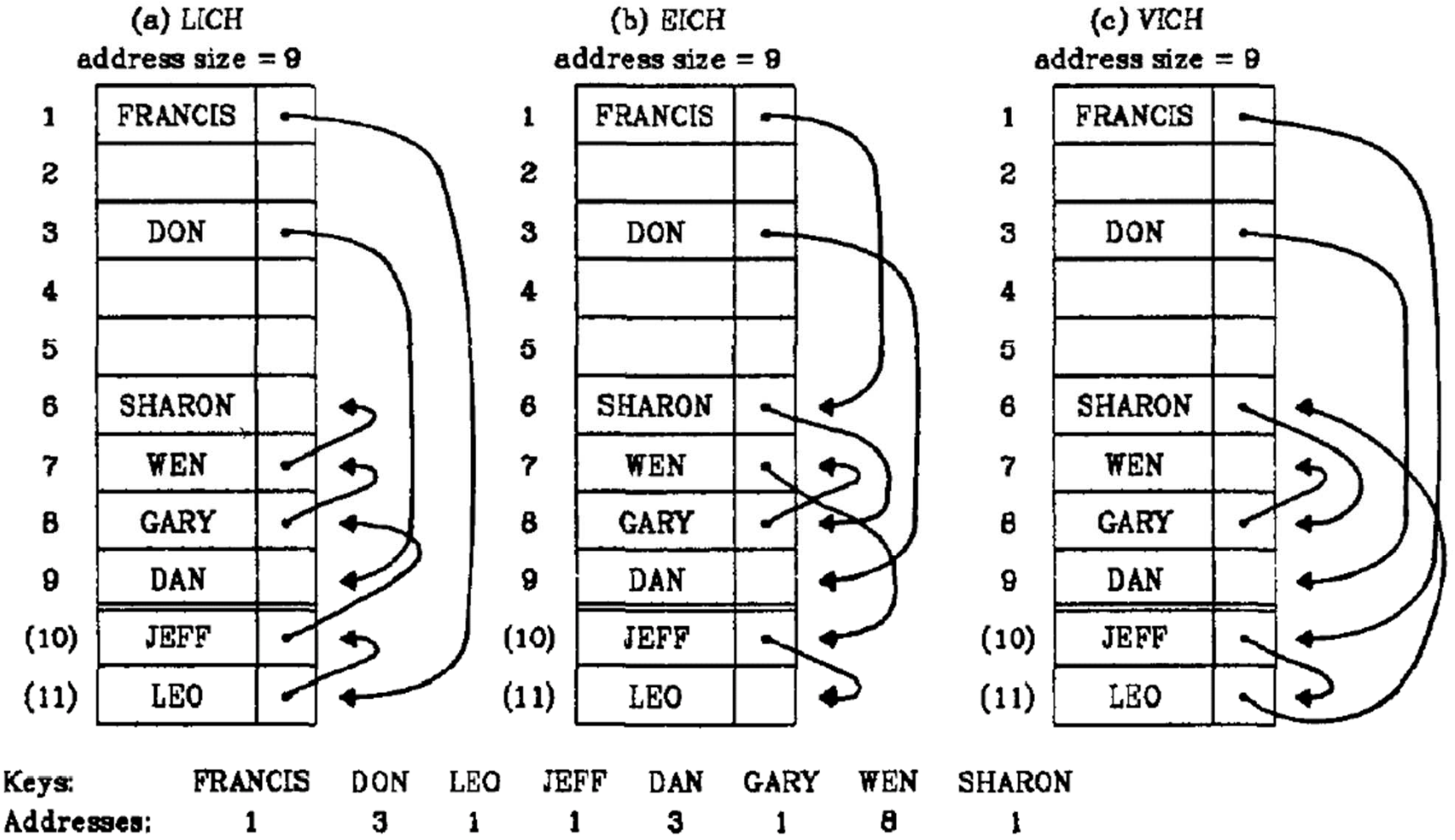
Standardní srůstající hashování – LISCH, EISCH



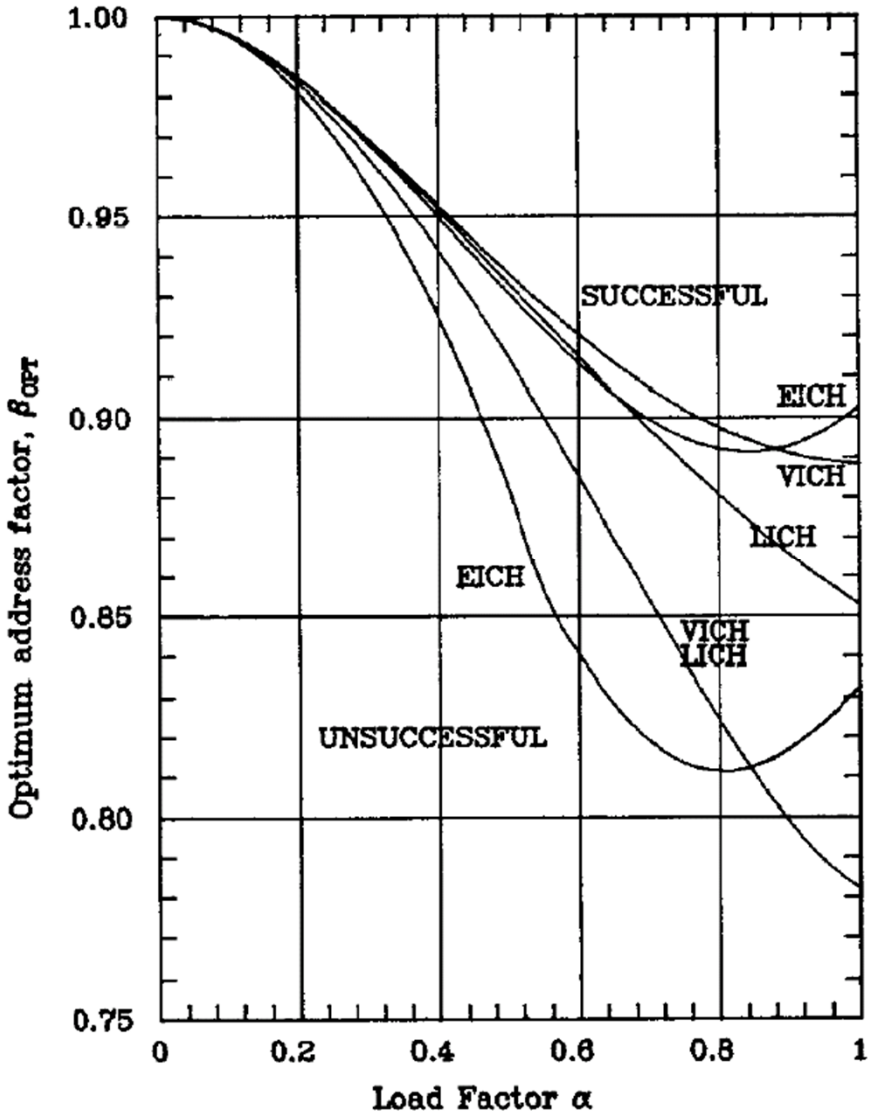
Srůstající hashování s pomocnou pamětí

- Pro snížení srůstání a tedy zvýšení efektivity hashování se tabulka rozšiřuje o pomocnou paměť - tzv. sklep (cellar).
- Sklep je místo na konci tabulky, které není adresovatelné hashovací funkcí (má ale stejnou strukturu jako celá hashovací tabulka).
- Algoritmy LICH a EICH jsou analogické varianty algoritmů LISCH a EISCH s přidáním sklepa.
- Algoritmus VICH (variable insert coalesced hashing) připojuje prvek na konec řetězce, pokud řetězec končí ve sklepě, jinak na místo, kde řetězec opustil sklep.

Srůstající hashování s pomocnou pamětí



Srůstající hashování s pomocnou pamětí



α – faktor naplnění (load factor)
$\alpha = N/M'$

β – faktor adresování (address factor)
$\beta = M/M'$

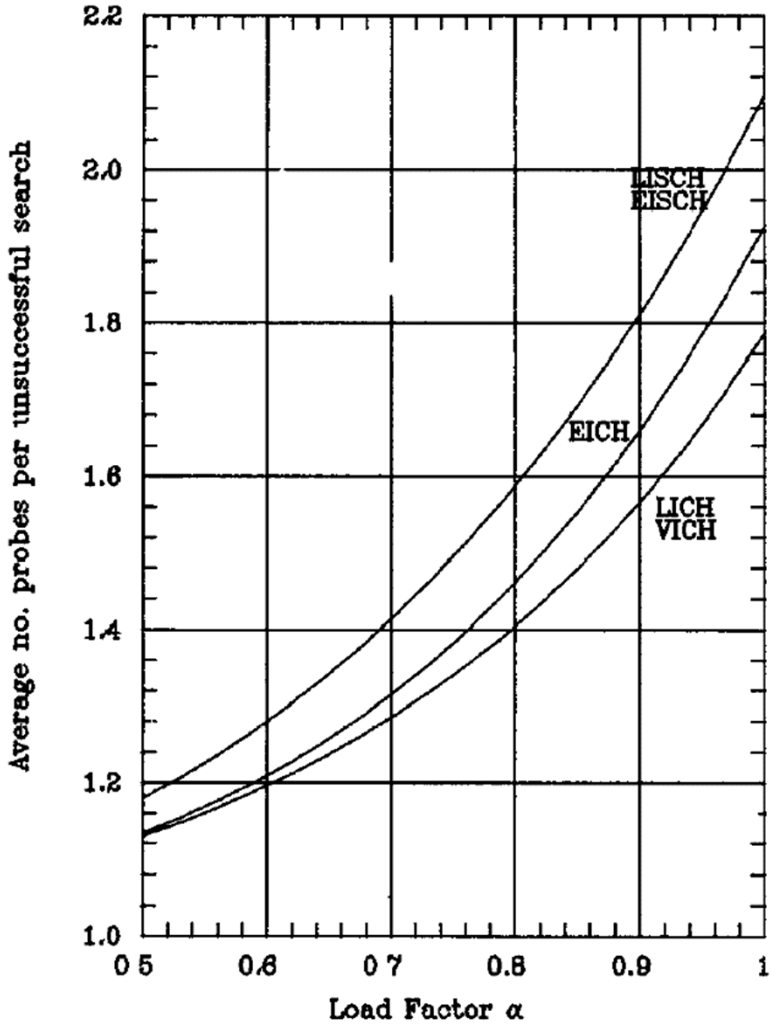
$K = M' - M$ = velikost sklepa

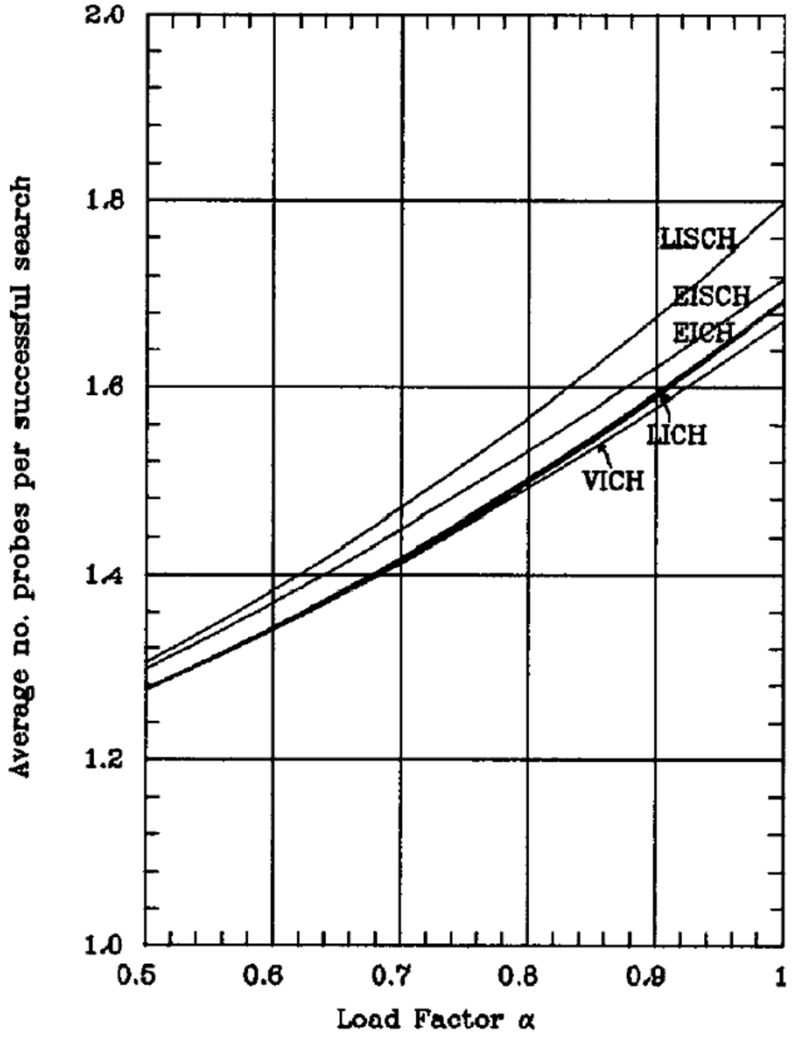
N – počet vložených prvků
M' – počet míst v hashovací tabulce
M – počet míst adresovatelných hashovací funkcí

Křivky představují optimální volbu β v závislosti na daném α při operaci FIND.

Případy:
- SUCCESSFUL - klíč je v tabulce
- UNSUCCESSFUL - klíč není v tabulce

Celkové srovnání srůstajícího hashování





- S použitím sklepa vychází nejlépe VICH. Doporučená velikost β je 0,86.
- Bez sklepa vychází nejlépe EISCH.

Efektivita srůstajícího hashování

α method	0.2	0.4	0.6	0.8	0.9	0.95	0.99
EISCH	1.1065	1.2277	1.3684	1.5290	1.6182	1.6653	1.7033
LISCH	1.1063	1.2316	1.3789	1.5657	1.6737	1.7337	1.7827
BEISCH	1.1055	1.2286	1.3721	1.5336	1.6236	1.6728	1.7107
BLISCH	1.1055	1.2341	1.3836	1.5703	1.6818	1.7423	1.7898
REISCH	1.1063	1.2322	1.3693	1.5257	1.6124	1.6614	1.7014
RLISCH	1.1085	1.2384	1.3876	1.5653	1.6723	1.7296	1.7790
EICH	1.1116	1.2256	1.3408	1.4942	1.5867	1.6347	1.6762
LICH	1.1116	1.2256	1.3406	1.4888	1.5801	1.6281	1.6695

Source: Hsiao, Yeong-Shiou, and Alan L. Tharp, "Analysis of Other New Variants of Coalesced Hashing," Technical Report TR-87-2, Computer Science Department, North Carolina State University, 1987.

- Průměrný počet navštívených klíčů při operaci FIND. Vždy se předpokládá rovnoměrné rozložení klíčů po celém oboru hodnot hashovací funkce. Nemusí nutně odpovídat reálné situaci.

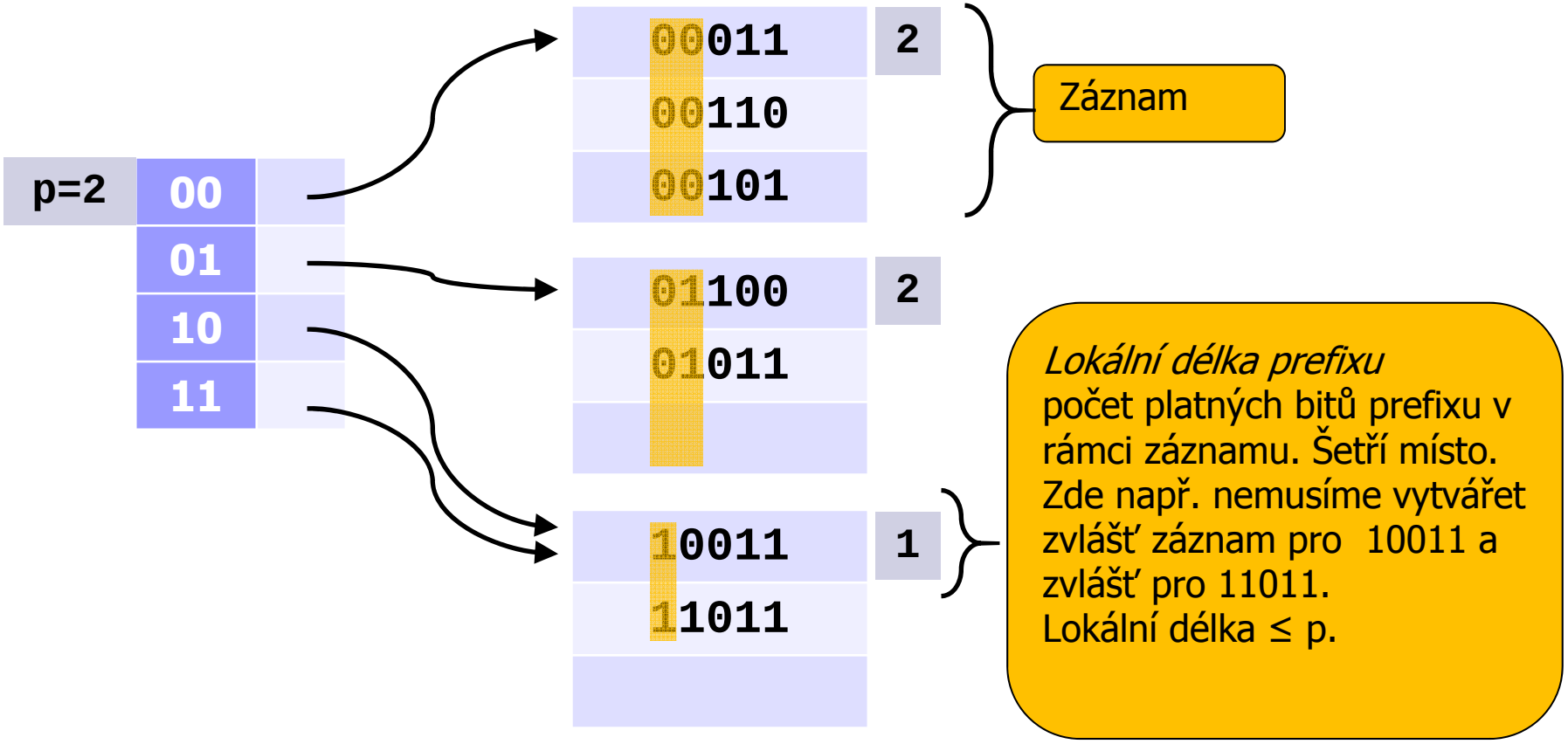
Dynamické hashování

- Inkrementální
  A. Při kritickém naplnění tabulky ji zvětšíme a přehashujeme.

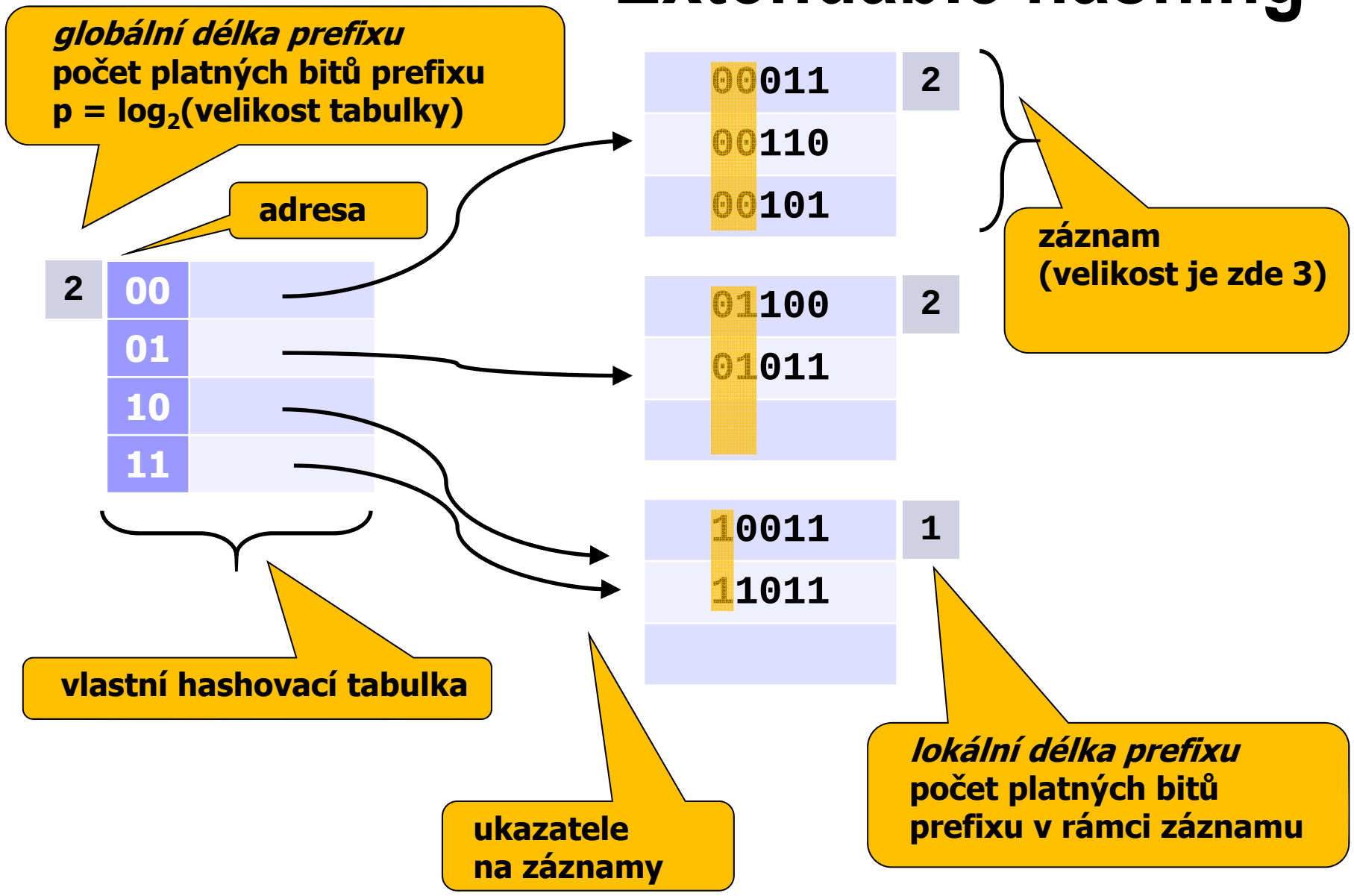
  B. Při kritickém naplnění tabulky naalokujeme novou větší tabulku, do které začneme ukládat všechny nové prvky. Ve staré pouze vyhledáváme a rušíme prvky. Při každé operaci (ať už v nové nebo v staré tabulce) zrušíme jeden prvek ve staré tabulce a vložíme jej do nové tabulky. Po zrušení všech prvků ve staré tabulce tuto tabulku zrušíme. Toto opatření nám zajistí, že bude tabulka zrušena dříve než bude potřeba naalokovat další (tj. třetí) tabulku; a zároveň není potřeba provádět jednorázové přehashovávání jako v předchozím případě (takové přehashování může způsobit významné narušení plynulosti průběhu hashování).
- Rozšiřitelné (Extendable)

Extendable hashing

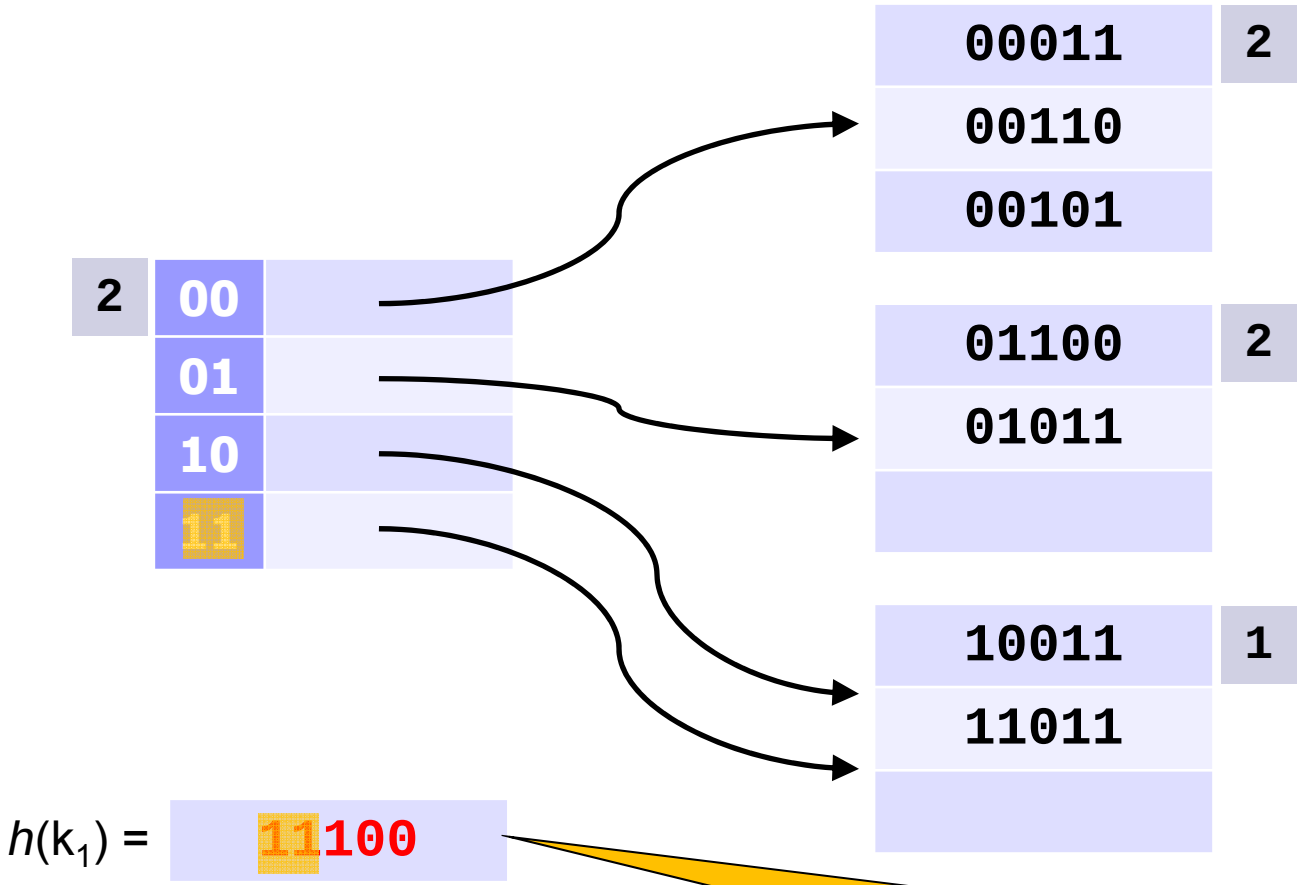
Klíč se interpretuje jako binární číslo. Tabulka má adresář a záznamy. Adresář obsahuje odkazy do záznamů a záznamy obsahují klíče. Záznamy se rozlišují prvními několika bity (prefixy) možných klíčů, klíč je v záznamu se stejným prefixem. Adresář vždy obsahuje všechny možné prefixy, jichž je 2^p, kde p je délka prefixu. Hodnota p se může měnit.



Extendable hashing

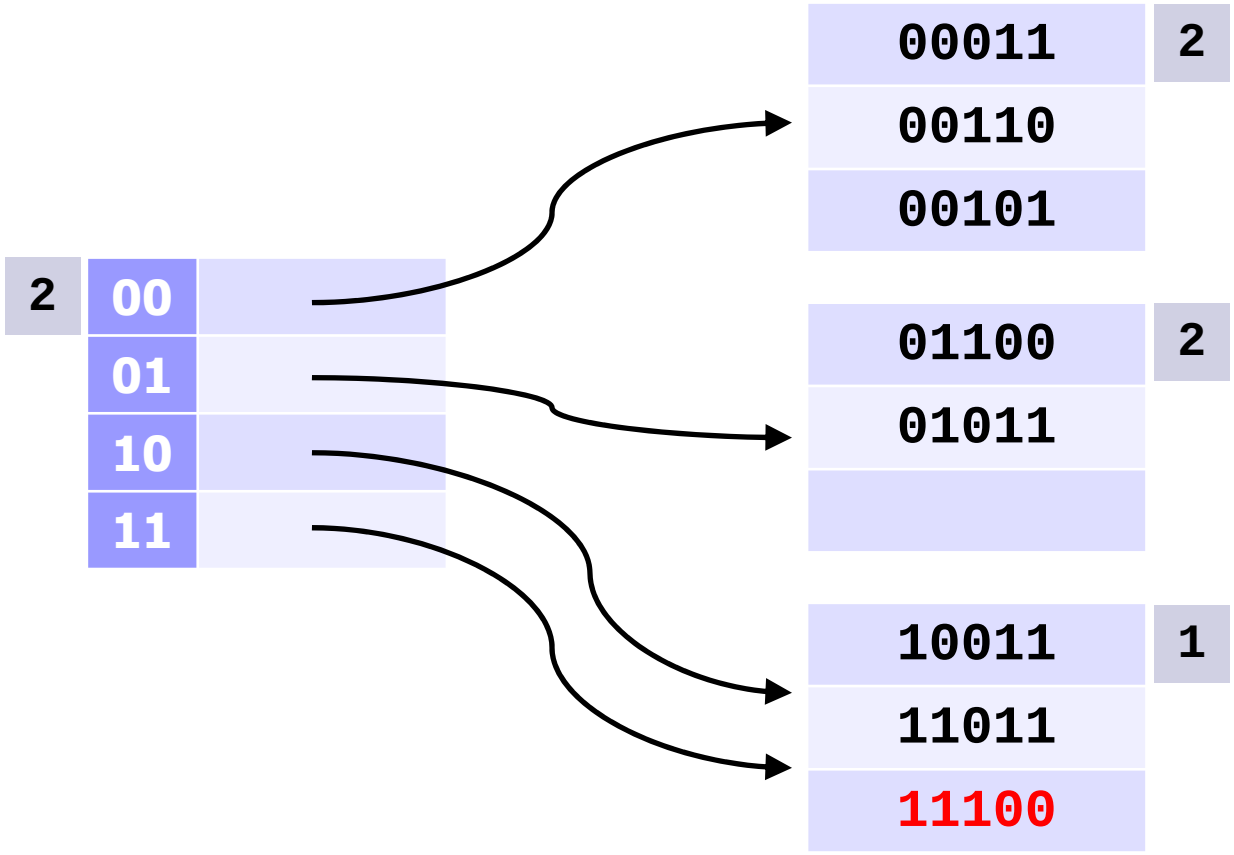


Extendable hashing

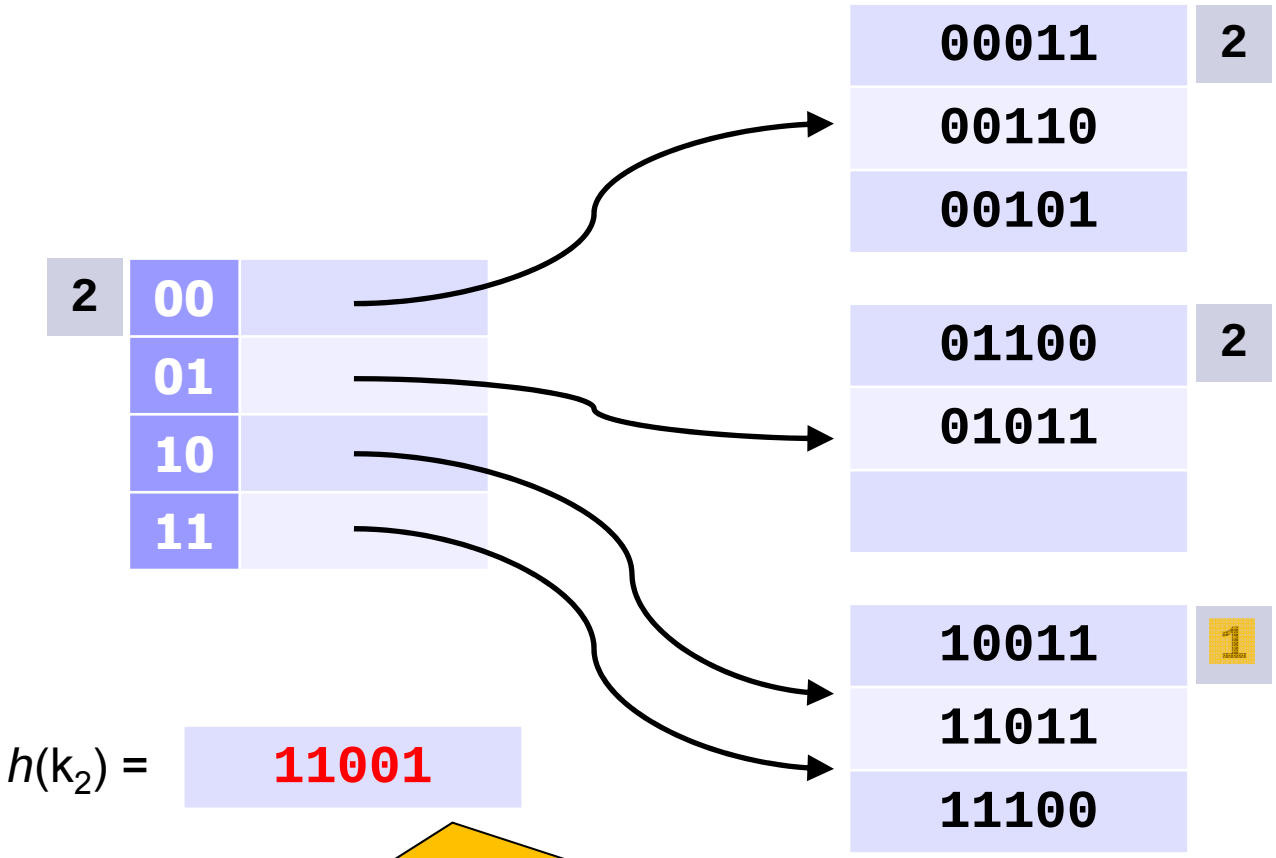


Chceme přidat nový klíč k_1.
Záznam kam patří je určen odkazem na řádku hashovací tabulky s adresou se stejným prefixem jakou má zahashovaná hodnota klíče $h(k_1)$.

Extendable hashing

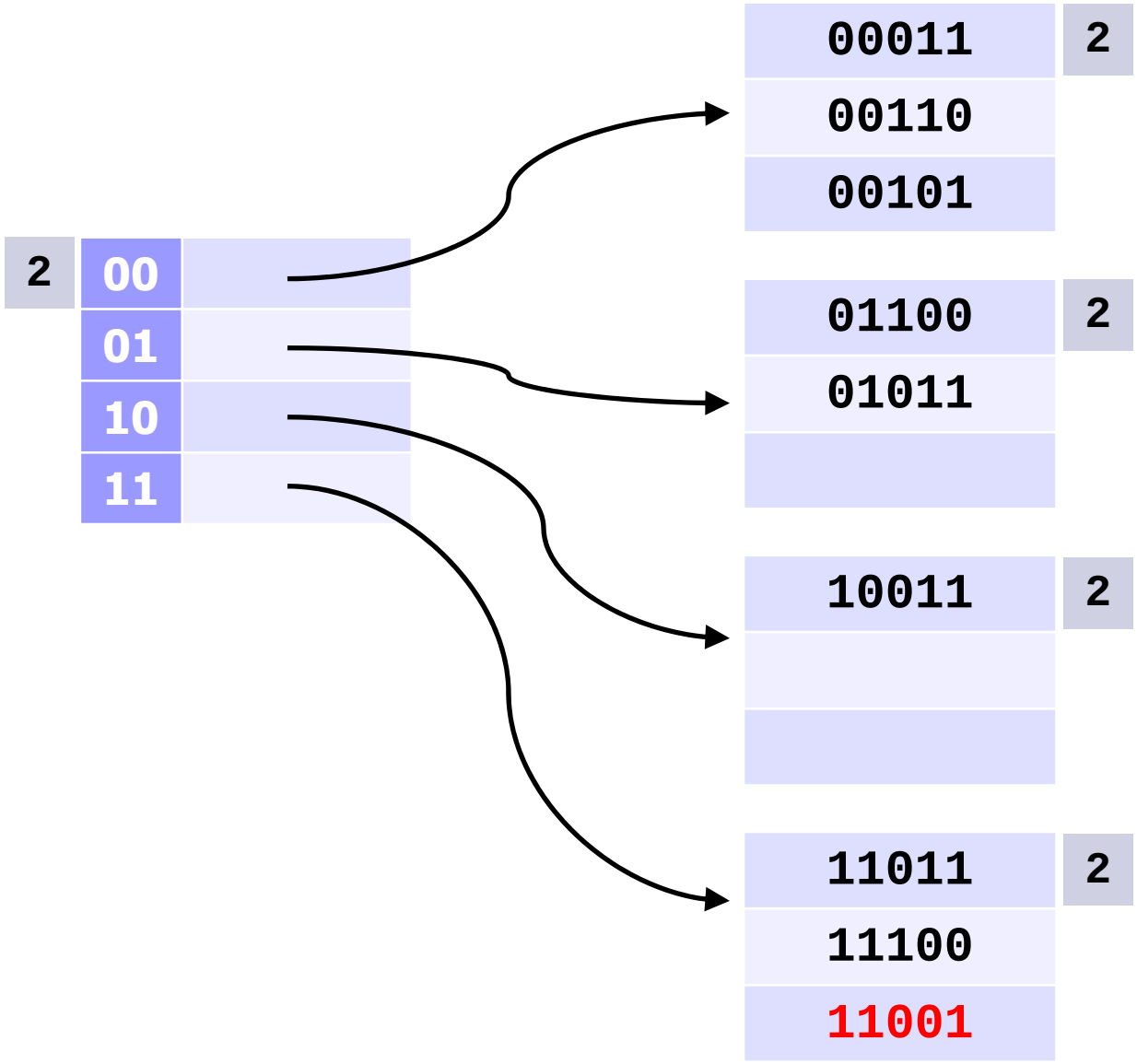


Extendable hashing

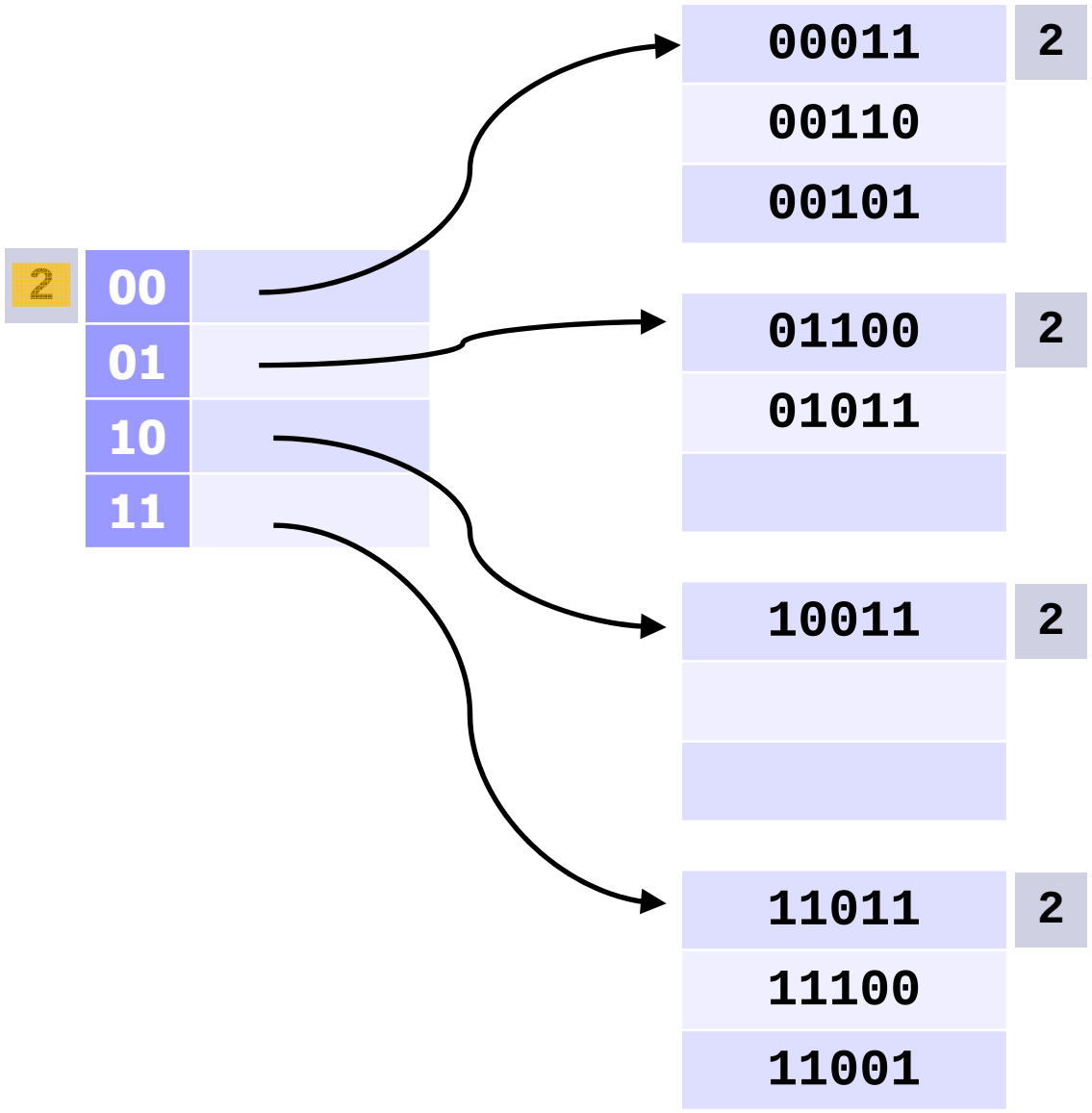


Chceme přidat nový klíč k_2, ale jeho záznam je již plný.
V tomto případě stačí zvětšit hodnotu lokální délky prefixu tohoto záznamu o 1 (lokální délka prefixu musí být vždy ≤ globální délce prefixu), přidat úplně nový záznam se stejnou hodnotou prefixu do tabulky (tj. přepojit na něj první polovinu ukazatelů, které ukazovaly na původní plný záznam) a celý obsah starého plného záznamu přehashovat. Do nově vzniklého místa zahashujeme i nový klíč.

Extendable hashing



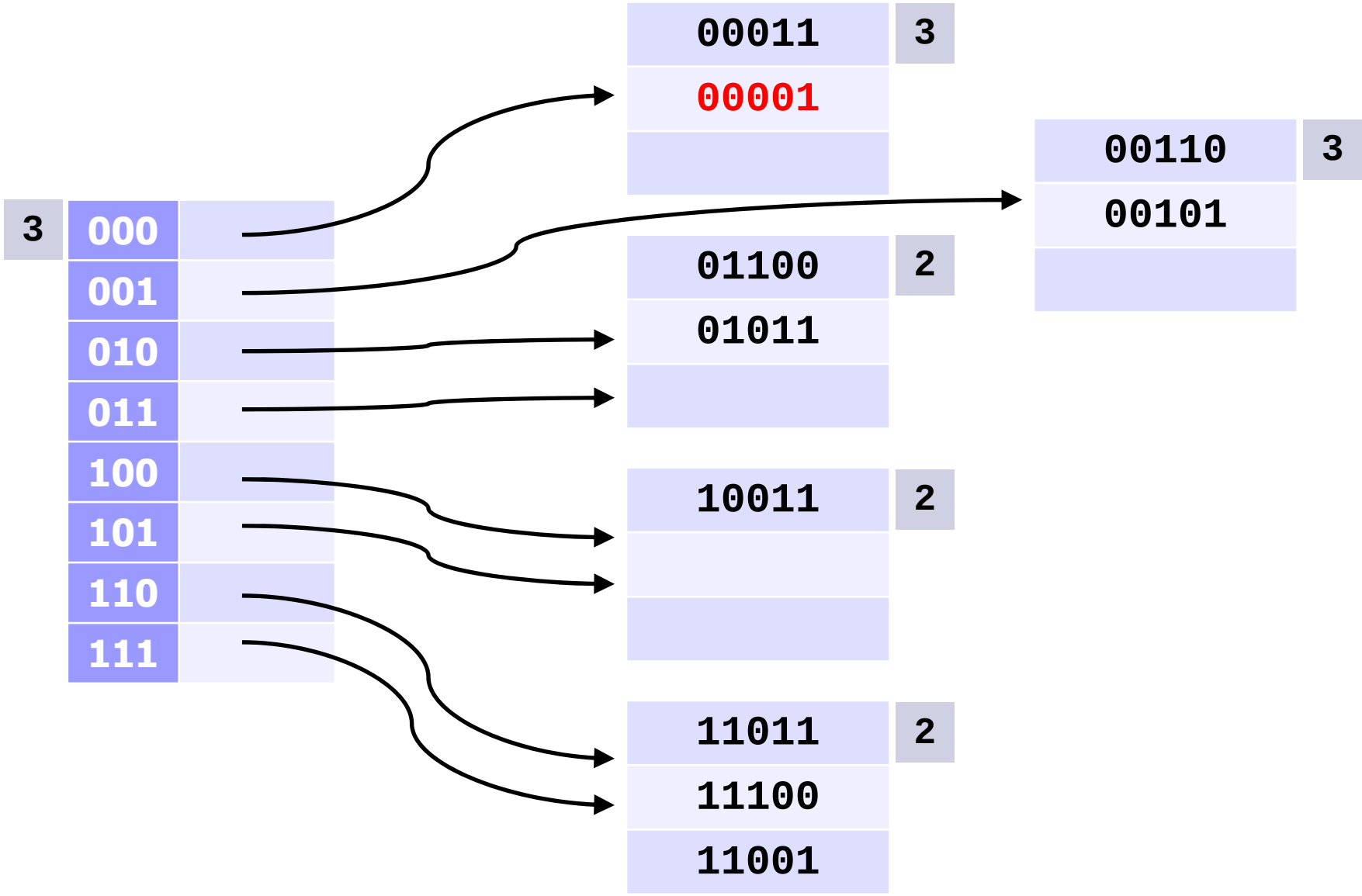
Extendable hashing



$h(k_3)$ = 00001

Chceme přidat nový klíč k_3, ale jeho záznam je již plný a navíc již nemůžeme ani zvětšit hodnotu lokální délky prefixu.
V tomto případě musíme o 1 zvětšit hodnotu globální délky prefixu a tím i zvětšit velikost hashovací tabulky na dvojnásobek. Ukazatele zduplikujeme podle hodnoty přechozího prefixu.
Nyní můžeme aplikovat postup jako v předchozím případě.

Extendable hashing



Univerzální hashování

- Každá hashovací funkce má slabá místa, kdy pro různé klíče dává stejnou adresu. Proto je výhodné přizpůsobit hashovací funkci právě zpracovávaným klíčům.
- **Univerzální hashování**
  - Místo jedné hashovací funkce $h(k)$ máme nějakou konečnou množinu $\boldsymbol{H}$ funkcí mapujících **U** do intervalu {0, 1, …, *m-1*}
  - Množina funkcí $\boldsymbol{H}$ je *univerzální*, pokud pro každou dvojici různých klíčů x,y $\in$ **U** je počet hashovacích funkcí z množiny $\boldsymbol{H}$, pro které $h(x) = h(y)$, nejvýše $|\boldsymbol{H}|/m$.
  - Důsledek: Pravděpodobnost kolize při náhodném výběru funkce $h(k)$ z množiny univerzálních hashovacích funkci $\boldsymbol{H}$ tedy není vyšší než pravděpodobnost kolize při náhodném a nezávislém výběru dvou stejných hodnot z intervalu {0, 1, …, *m-1*} tedy $1/m$.
  - Při prvním spuštění programu jednu náhodně zvolíme. Funkci pak náhodně měníme jen v případě, že počet kolizí převyšuje přípustnou mez. V tomto případě je samozřejmě potřeba přehashovat celou tabulku.