



Algoritmizace

Hashing II

Jiří Vyskočil, Marko Genyg-Berezovskyj

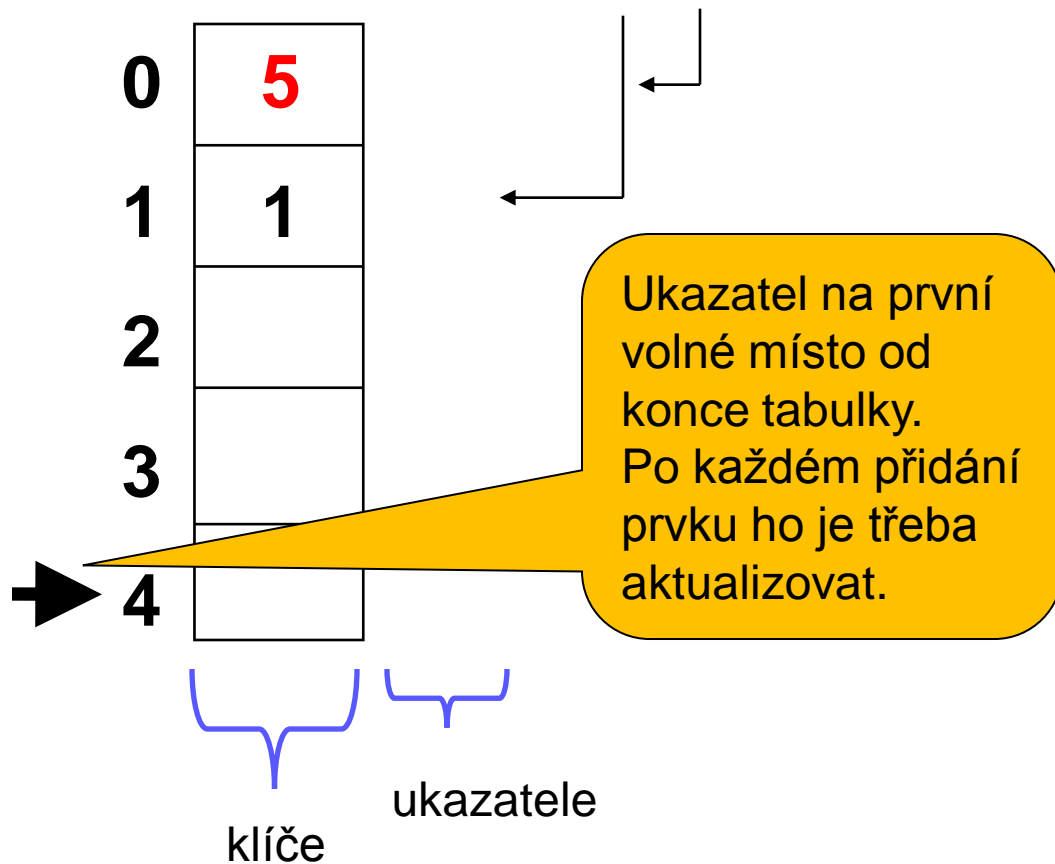
2010

Srůstající hashování (coalesced hashing)

- Znám předem počet prvků (odhad)
- Z důvodů efektivity nechci ukazatele (mezi prvky). Na jednu pozici tabulky případně právě jeden ukazatel.
- => kolizní prvky (synonyma) se musejí „nějak“ ukládat přímo do hashovací tabulky. Ukazatele v tabulce umožňují procházet pouze prvky jedné skupiny. Nedochozí 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

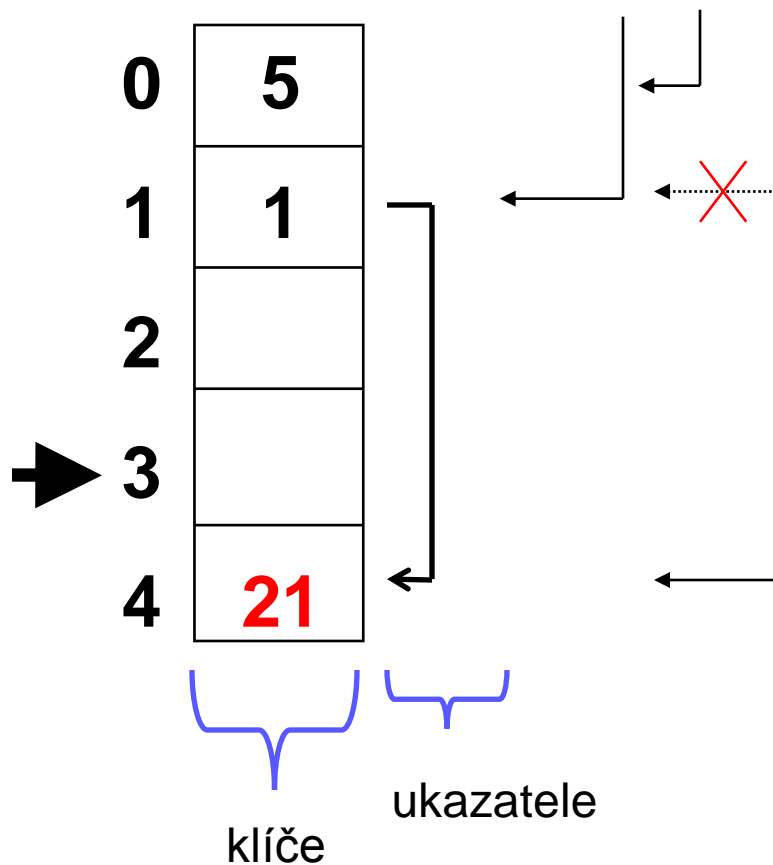


Postup:

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



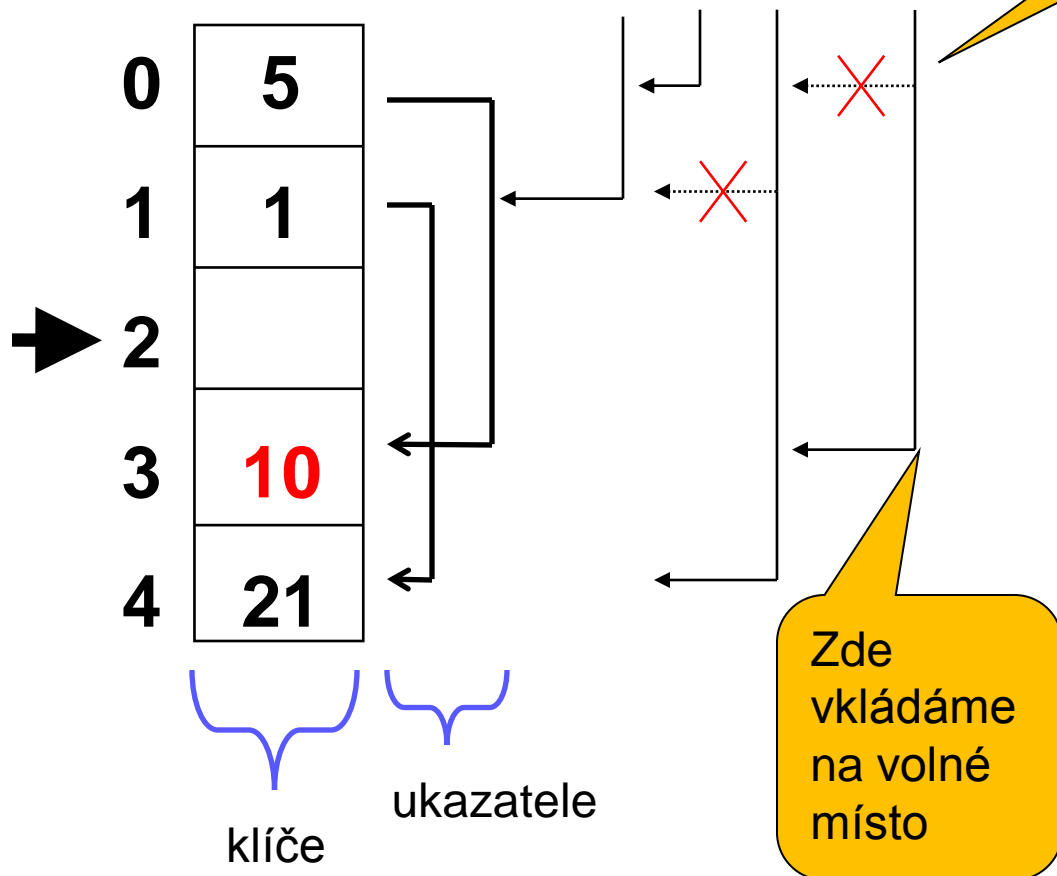
Postup:

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.

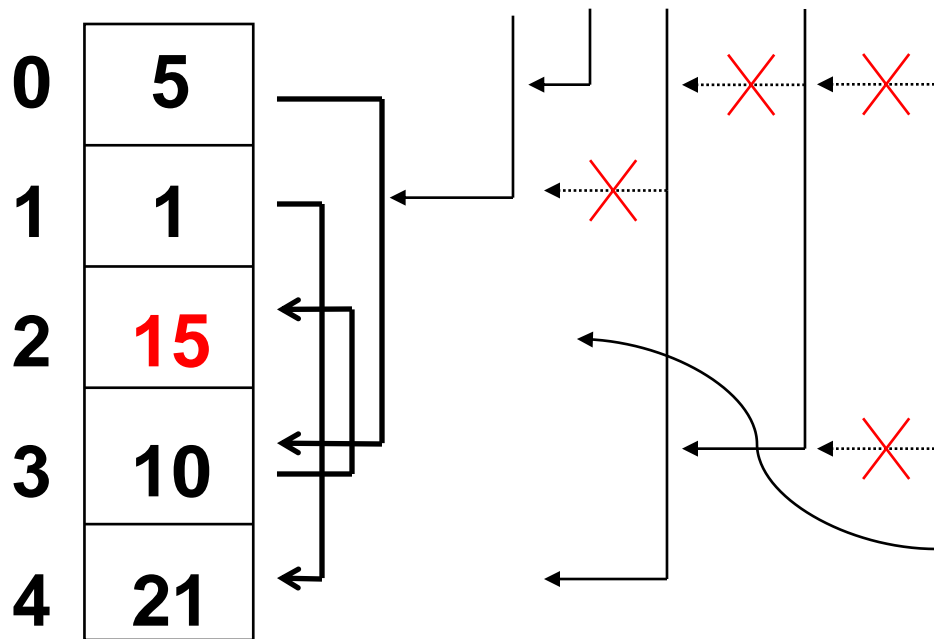
Postup:

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.

Zde vkládáme na volné místo

Standardní srůstající hashování - LISCH

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



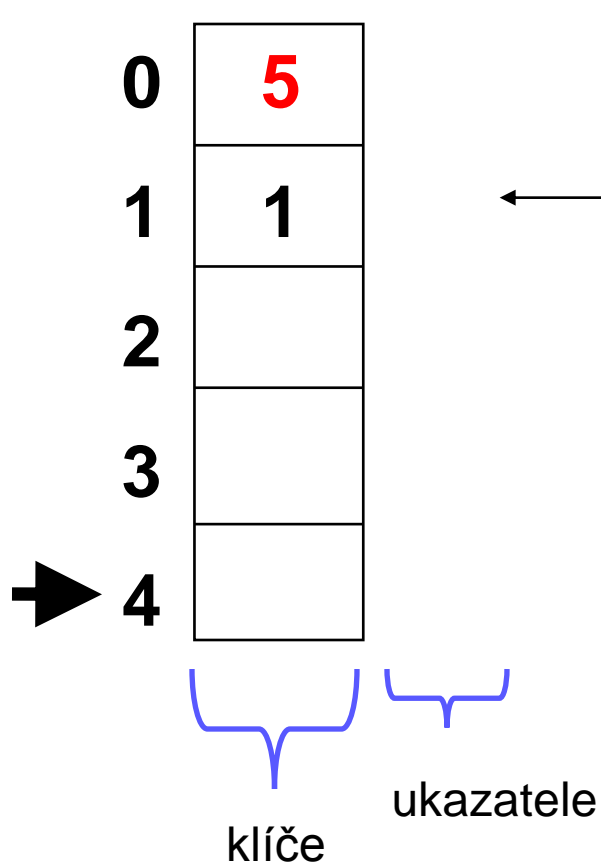
Postup:

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.

➔ Tabulka je zaplněna.

Standardní srůstající hashování - EISCH

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

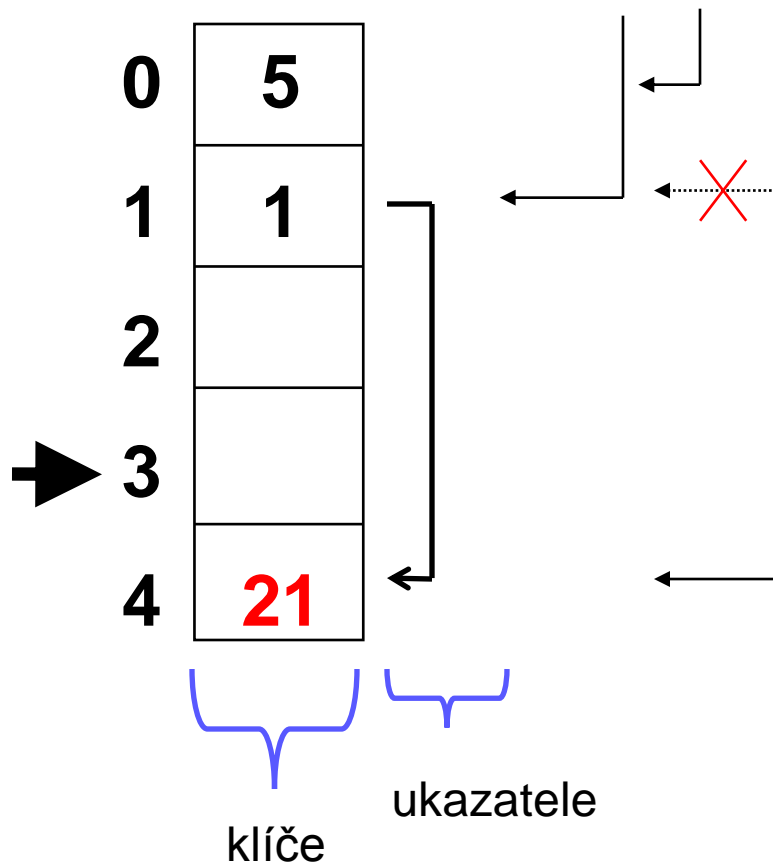


Postup:

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



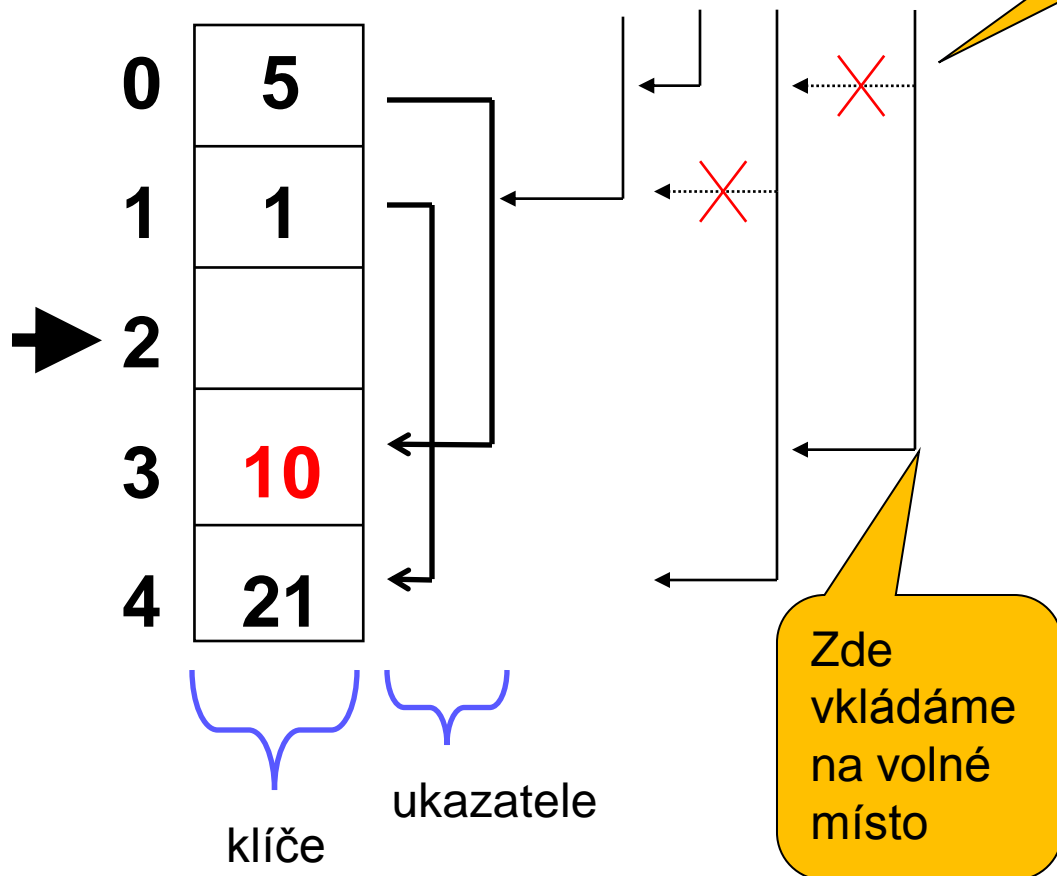
Postup:

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.

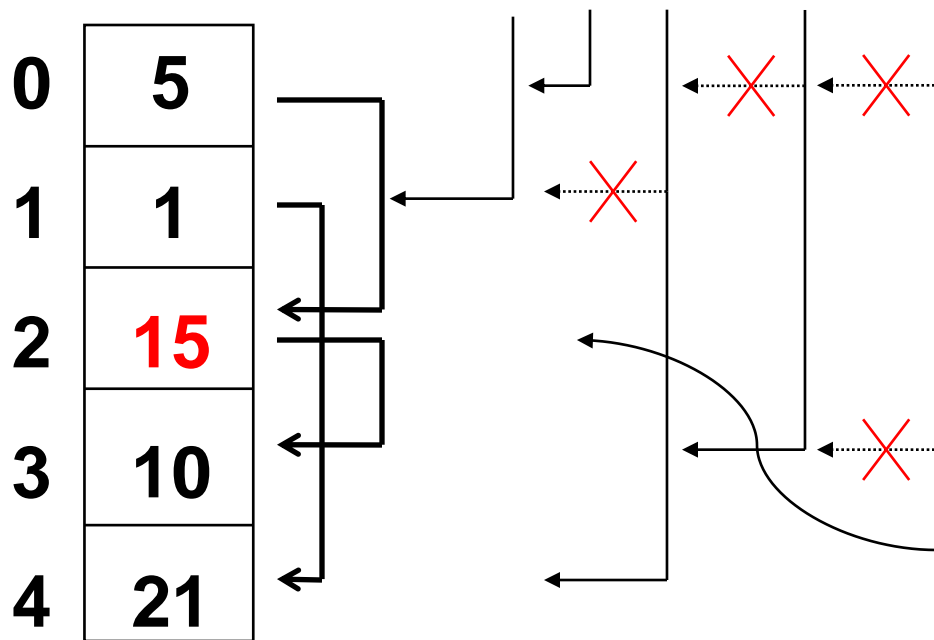
Postup:

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.

Zde vkládáme na volné místo

Standardní srůstající hashování - EISCH

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

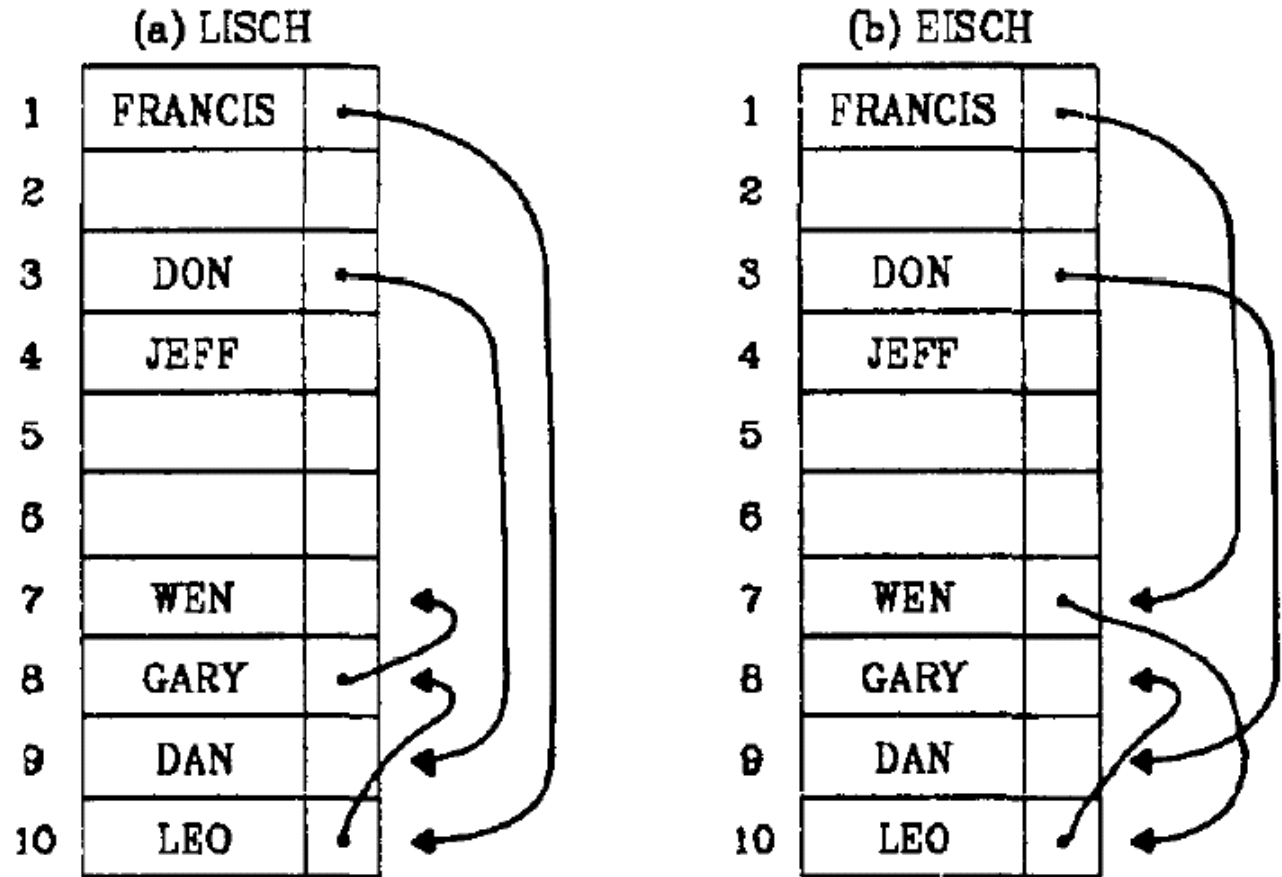


Postup:

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



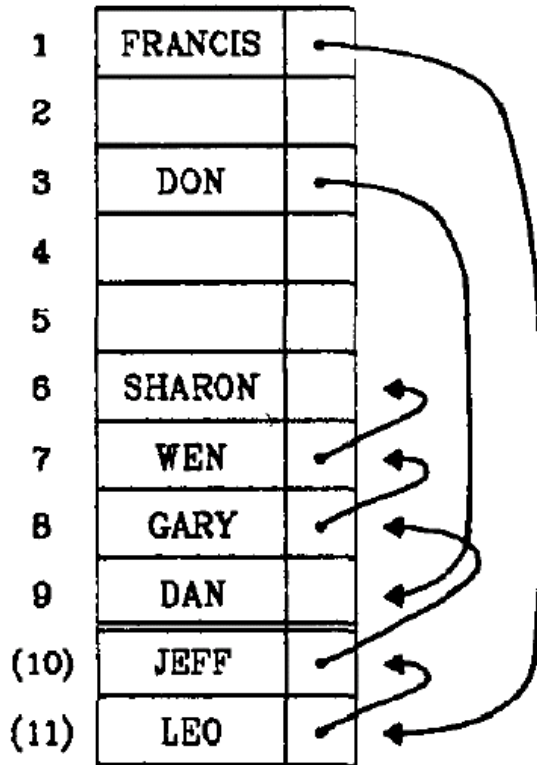
Keys: FRANCIS DON LEO JEFF DAN GARY WEN
 Hash Addresses: (a)(b) 1 3 1 4 9 10 1

Srůstající hashování s pomocnou pamětí

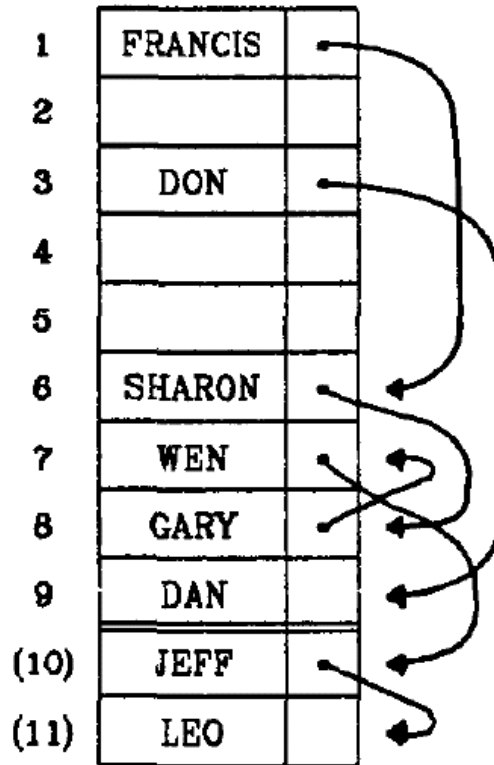
- Pro snížení srůstání a tedy zvýšení efektivity hashování se tabulka rozšiřuje o pomocnou paměť - tzv. sklep (cellar).
- Sklep je místo na konci tabulky, které není adresovatelné hashovací funkcí (má ale stejnou strukturu jako celá hashovací tabulka).
- Algoritmy LICH a EICH jsou analogické varianty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

Srůstající hashování s pomocnou pamětí

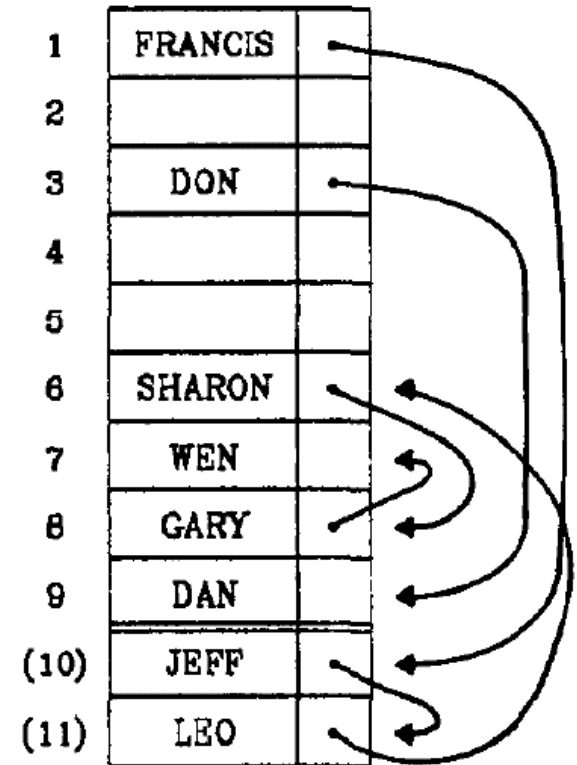
(a) LICH
address size = 9



(b) EICH
address size = 9

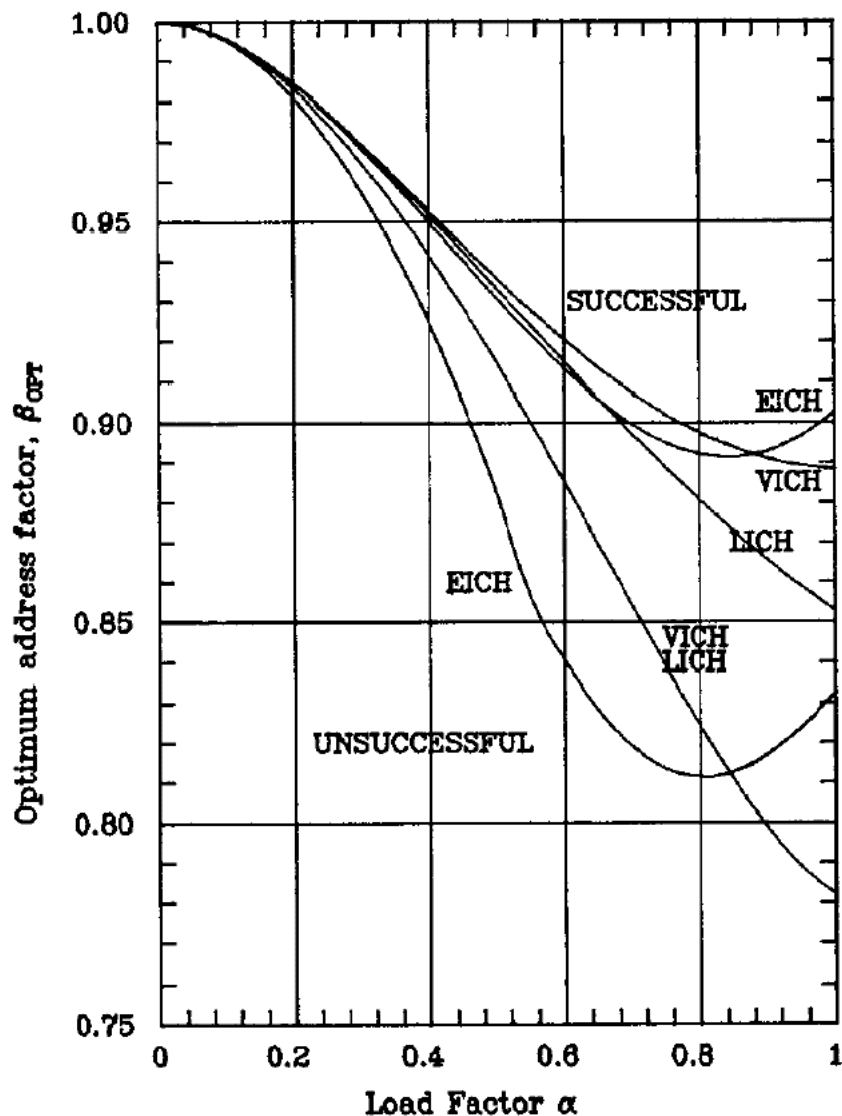


(c) VICH
address size = 9



Keys: FRANCIS DON LEO JEFF DAN GARY WEN SHARON
Hash Addresses: 1 3 1 1 3 1 8 1

Srůstající hashování s pomocnou pamětí



α – faktor naplnění (load factor)

$$\alpha = N/M'$$

β – faktor adresování (address factor)

$$\beta = M/M'$$

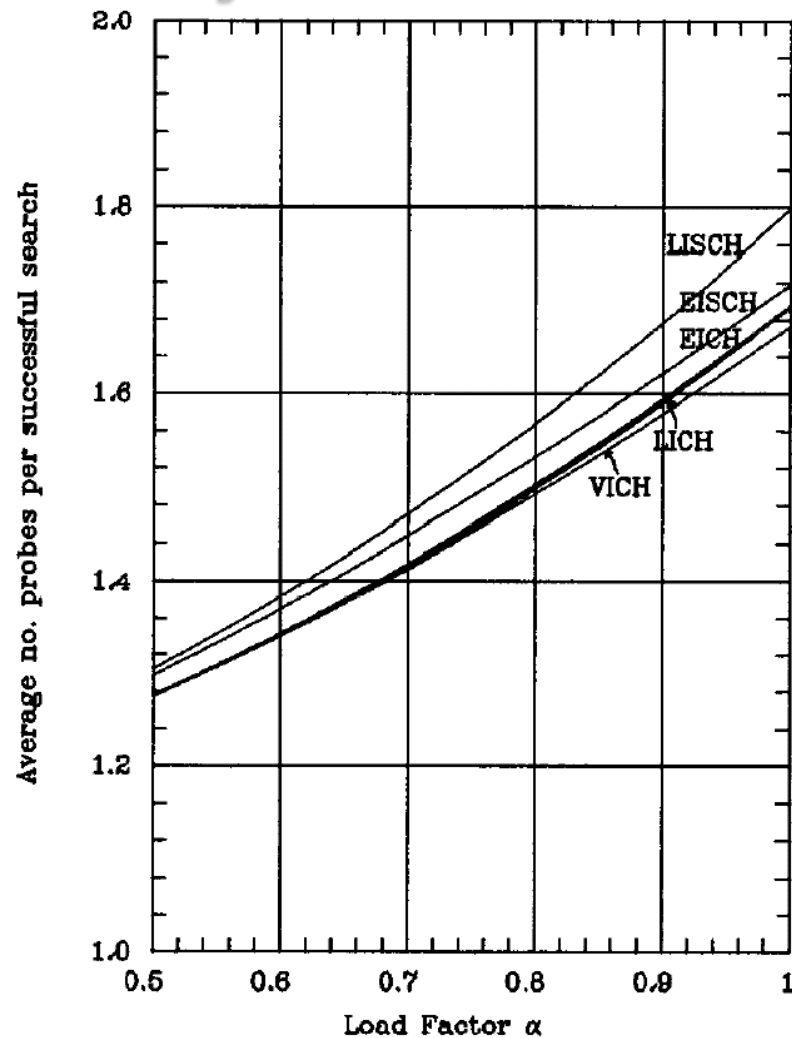
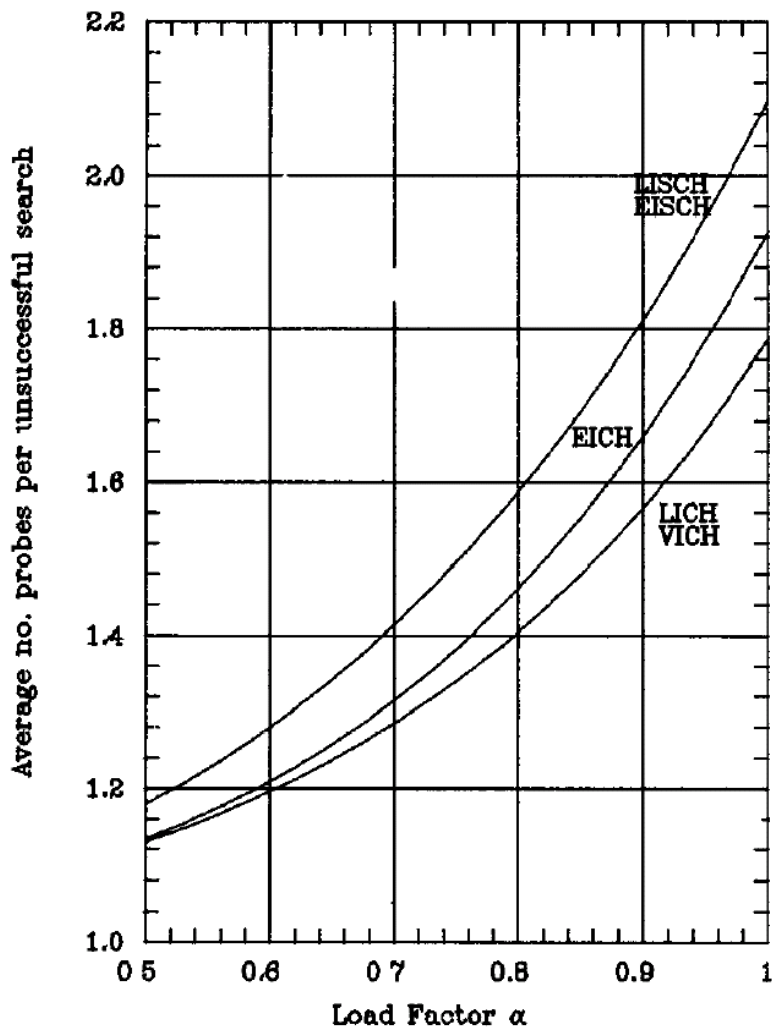
$K = M' - M =$ velikost sklepa

N – počet vložených prvků

M' – počet míst v hashovací tabulce

M – počet míst adresovatelných hashovací funkcí

Celkové srovnání srůstajícího hashování



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

Dynamické hashování

■ Inkrementální

- Při kritickém naplnění tabulky ji zvětšíme a přehashujeme.
- Při kritickém naplnění tabulky naalokujeme novou tabulku, do které začneme ukládat všechny nové prvky. Ve staré pouze vyhledáváme a rušíme prvky. Při každé operaci nad starou hashovací tabulkou zrušíme jeden prvek. Po zrušení všech prvků ze staré tabulky 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 přehashování jako v předchozím případě (takové přehashování může způsobit významné narušení plynulosti průběhu hashování).

■ Rozšiřitelné (Extendable)

Extendable hashing

globální délka prefixu
počet platných bitů prefixu
 $= \log_2(\text{velikost tabulky})$

adresa

2	00	
	01	
	10	
	11	

00	011	2
00	110	
00	101	

01	100	2
01	011	

1	0011	1
1	1011	

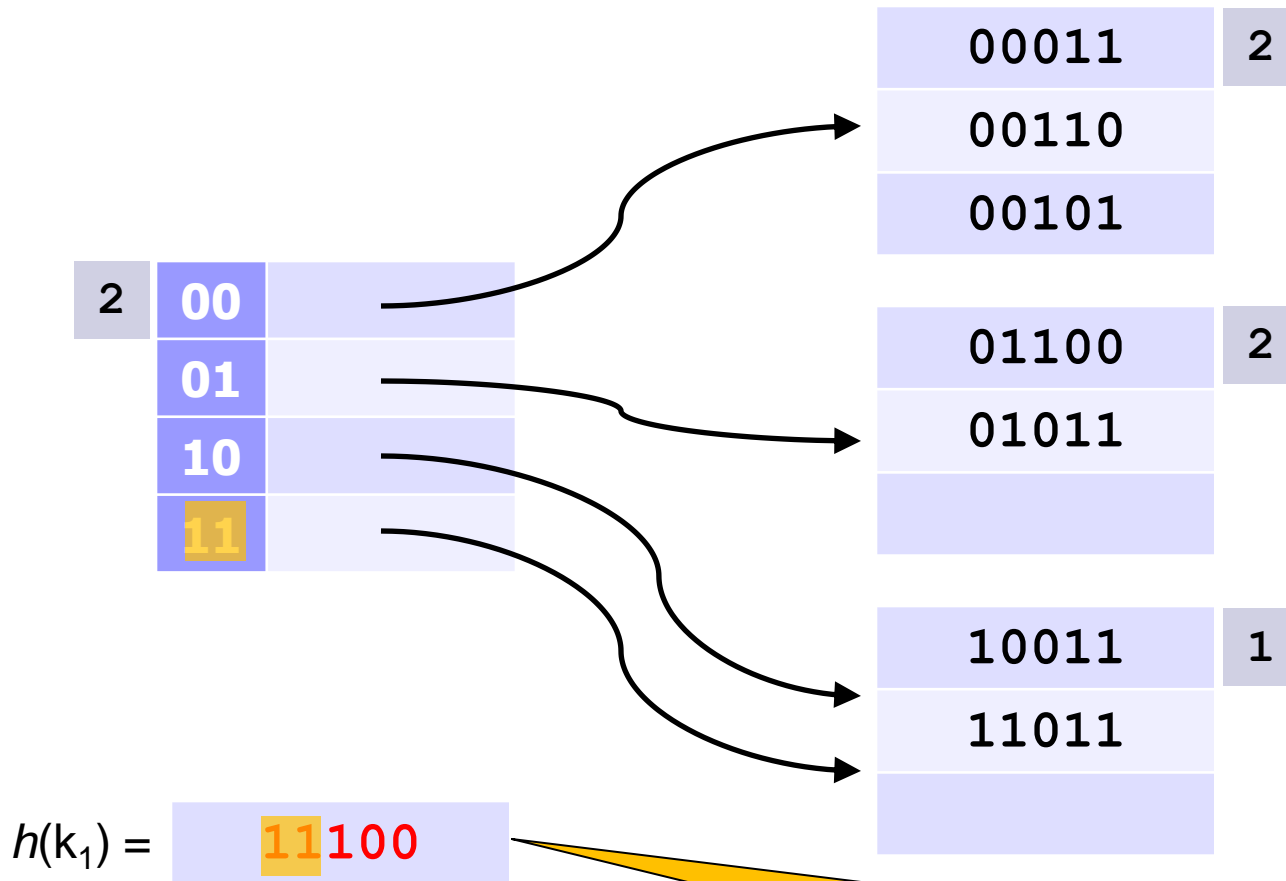
záznam
(velikost je zde 3)

vlastní hashovací tabulka

ukazatele
na záznamy

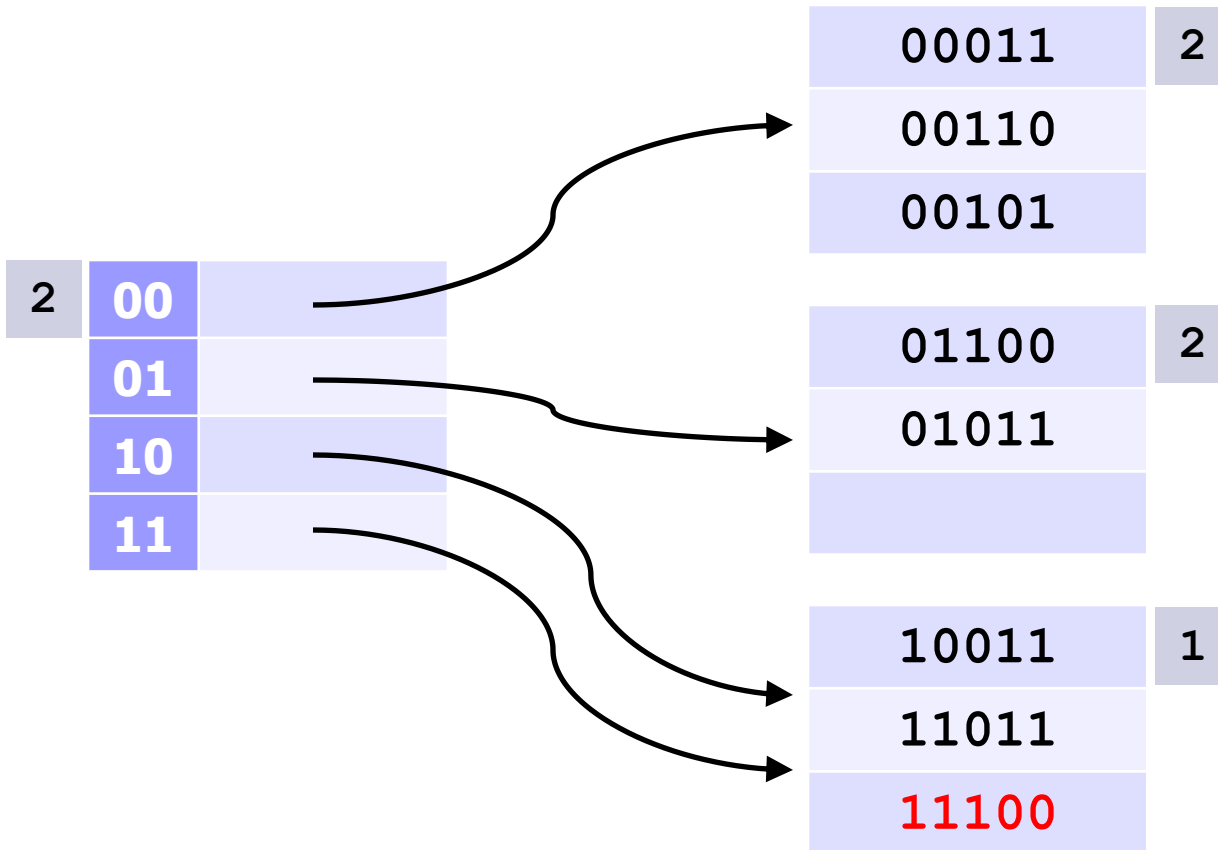
lokální délka prefixu
počet platných bitů prefixu v
rámci záznamu

Extendable hashing

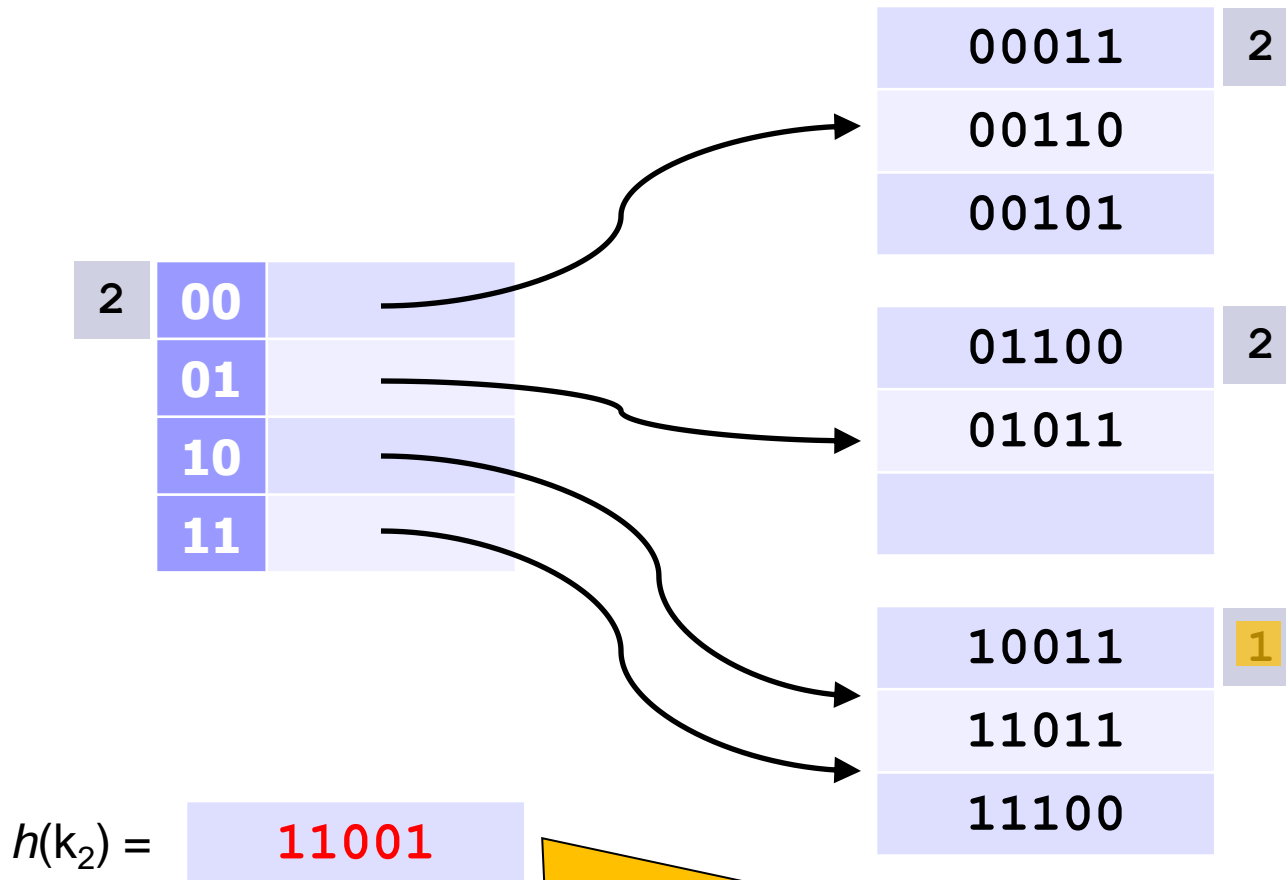


Chceme přidat nový klíč k_1 .
Záznam kam patří je určen odkazem na řádku
hashovací tabulky s adresou se stejným prefixem
jakou má zahashovaná hodnota klíče $h(k_1)$.

Extendable hashing

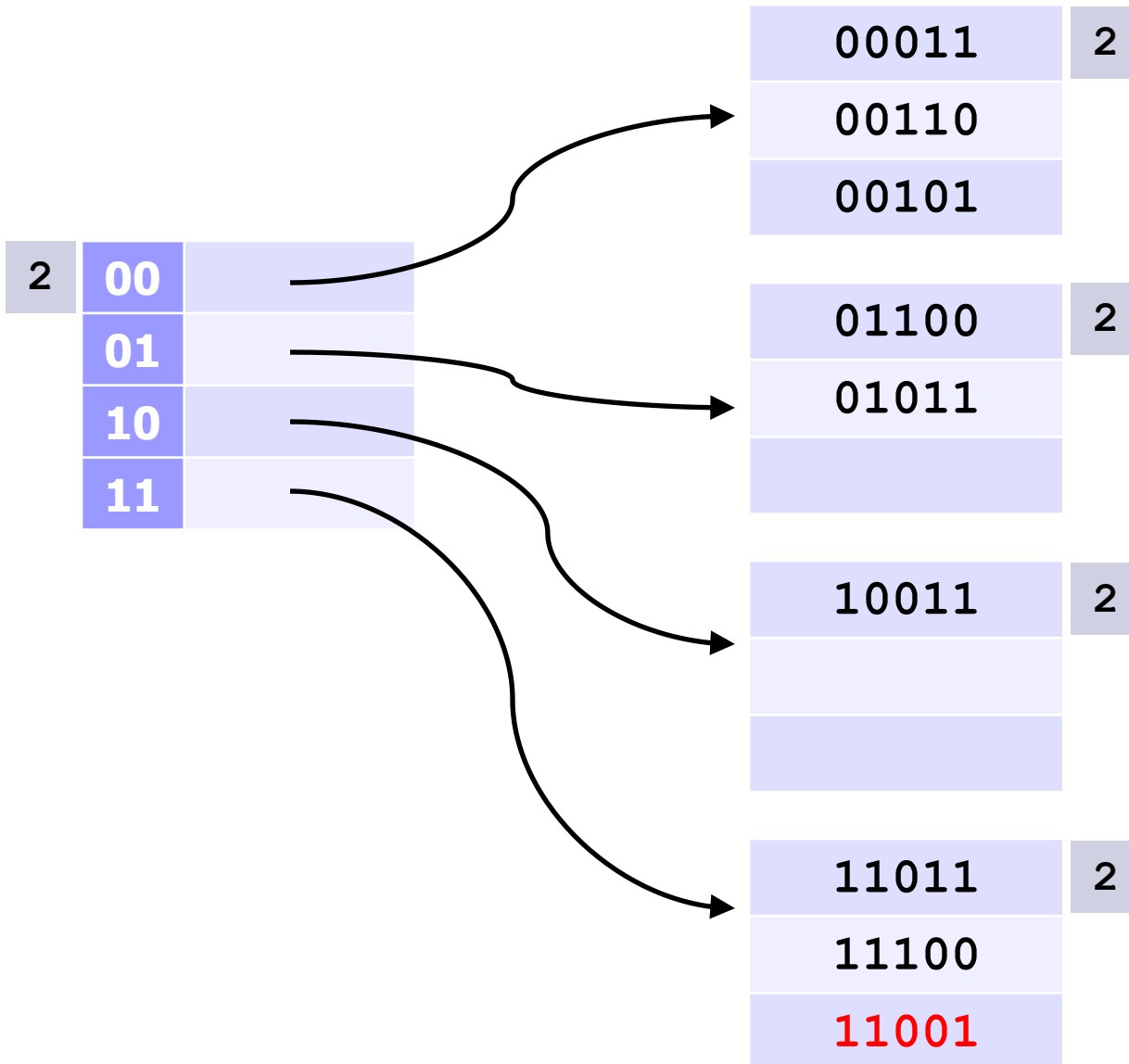


Extendable hashing

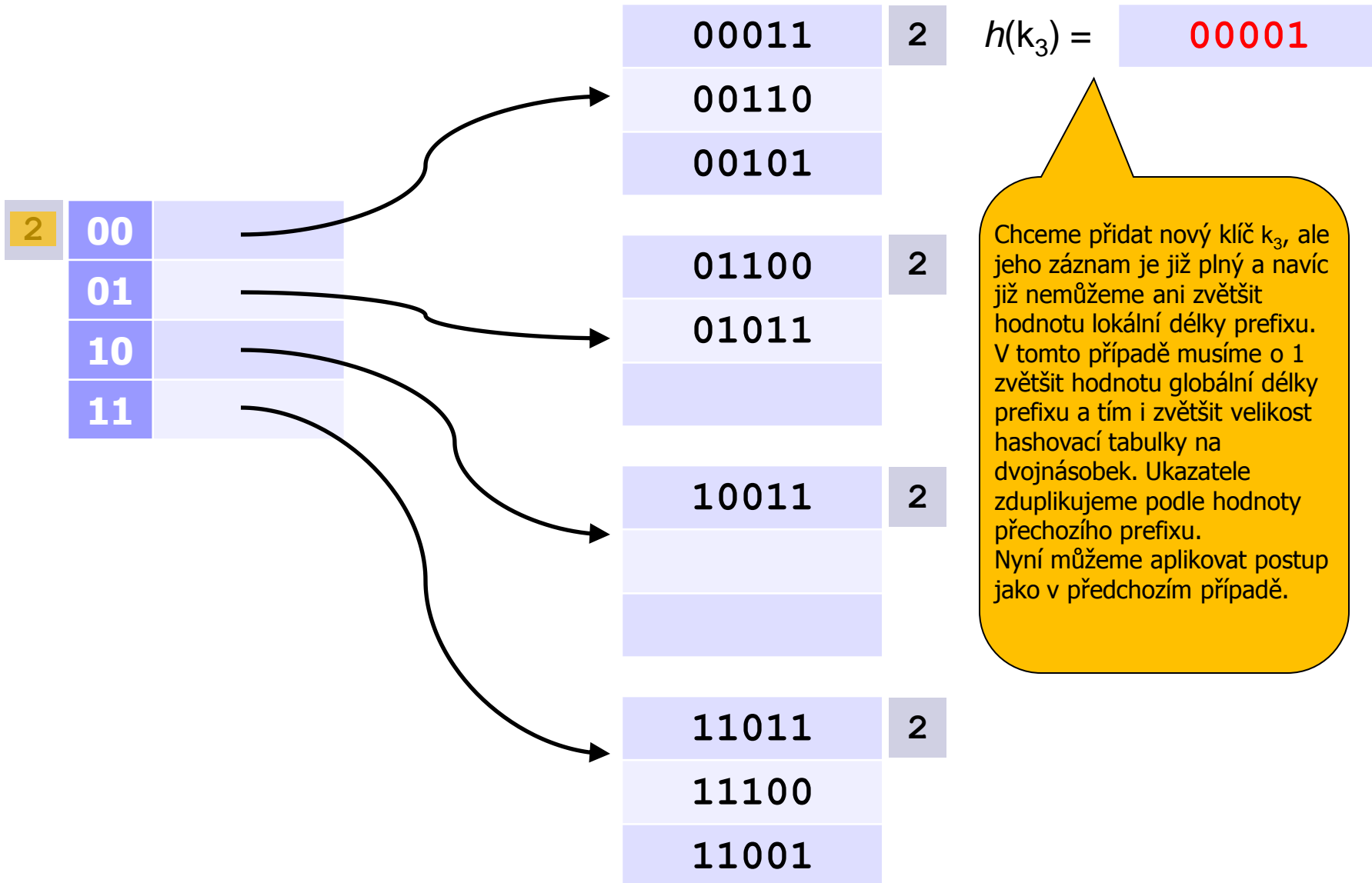


Chceme přidat nový klíč k_2 , ale jeho záznam je již plný.
V tomto případě stačí zvětšit hodnotu lokální délky prefixu tohoto záznamu o 1 (lokální délka prefixu musí být vždy \leq globální délce prefixu), přidat úplně nový záznam se stejnou hodnotou prefixu do tabulky (tj. přepojit na něj první polovinu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

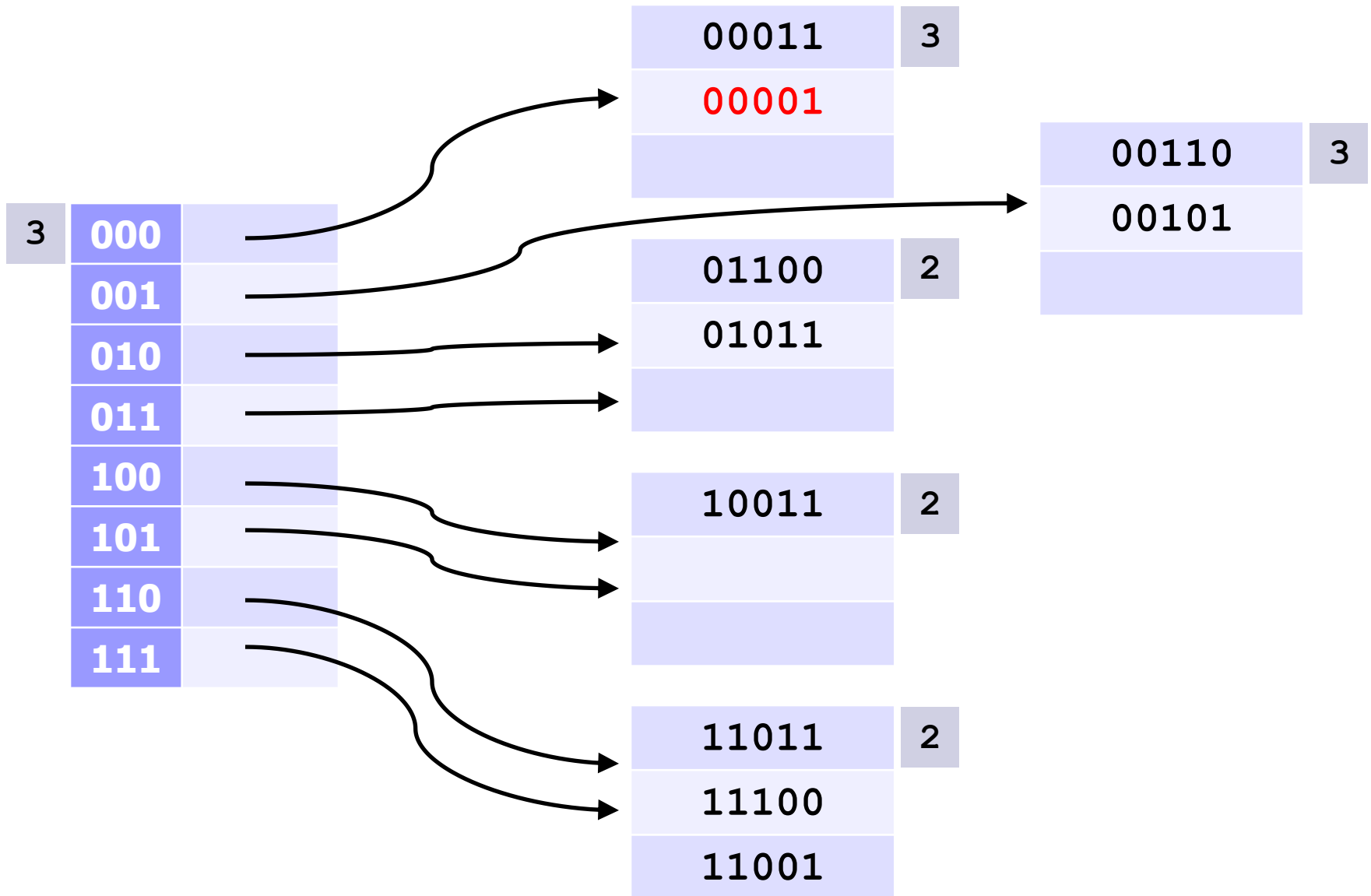
Extendable hashing



Extendable hashing



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 H funkcí mapujících U do intervalu $\{0, 1, \dots, m-1\}$
 - Množina funkcí H je *univerzální*, pokud pro každou dvojici různých klíčů $x, y \in U$ je počet hashovacích funkcí z množiny H , pro které $h(x) = h(y)$, nejvýše $|H|/m$.
 - Důsledek: Pravděpodobnost kolize při náhodném výběru funkce $h(k)$ z množiny univerzálních hashovacích funkcí H tedy není vyšší než pravděpodobnost kolize při náhodném a nezávislém výběru dvou stejných hodnot z intervalu $\{0, 1, \dots, m-1\}$ 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.