

# Structuri de date – Curs 4

Prof. univ. dr. Cristian CIUREA  
Departamentul de Informatică și Cibernetică Economică  
Academia de Studii Economice din București  
[cristian.ciurea@ie.ase.ro](mailto:cristian.ciurea@ie.ase.ro)

# Agenda

- ▶ Metode de regăsire
- ▶ Funcții de dispersie
- ▶ Tabele de dispersie

# Tabele de dispersie

- ▶ Structurile de date trebuie să permită regăsirea rapidă a obiectelor utilizate, pentru prelucrare eficientă.
- ▶ Metode de regăsire:
  - *căutare secvențială* (în listă): durează proporțional cu numărul de elemente;
  - *căutare binară* (într-un vector): durează logaritmic, dar structura trebuie menținută sortată; pe ansamblu rezultă un efort similar dacă sunt multe inserări/ștergeri;
  - *căutare în arbore binar*: durează tot logaritmic, dar arborele trebuie menținut relativ echilibrat.

# Tabele de dispersie

Operația de căutare a datelor:

- ▶ fiecare valoare din colecția de date are asociată o poziție unică în colecție;
- ▶ pe baza valorii căutate, se determină poziția acesteia în cadrul colecției de date.

# Tabele de dispersie

- ▶ În cadrul vectorilor, asocierea dintre valoarea unei chei numerice întreagă și poziția acesteia în vector se realizează prin:
  - definirea unui vector cu număr de elemente egal cu valoarea maximă posibilă a cheii de căutare;
  - stabilirea unei valori neutilizate în cadrul problemei de rezolvat pentru a indica dacă elementul cu cheia căutată există.

# Tabele de dispersie

Asocierea valoare cheie–poziție în VECTOR:

- ▶ numărul de elemente egal cu valoarea maximă a cheii de căutare;
- ▶ elementele există sau sunt șterse logic (valoare element din afara valorilor de cheie);
- ▶ căutare date în acces direct => MINIMIZARE timp de regăsire.

# Tabele de dispersie

Dezavantaje:

- ▶ dimensiunea memoriei ocupate:

$$\text{MEMORIE} = \text{maxim}(\text{valoare\_cheie\_căutare}) * \text{dimensiune}(\text{element})$$

- ▶ valoare maximă foarte mare  $\Rightarrow$  spațiu de memorie considerabil;
- ▶ nu se ține cont de numărul real de elemente utilizate; cazul cel mai nefavorabil: număr foarte mic de elemente și valoare mare a cheii maxime.
- ▶ tipul cheii de căutare: tip numeric – trebuie să fie index în accesarea elementelor din vector.

# Tabele de dispersie

## ► Exemplu:

```
typedef struct  
{  
    char nume[20];  
    int varsta;  
    char facultate[20];  
    int nrMatricol;  
} Student;
```

- dacă valoarea maximă a *nrMatricol* este 55630, iar numărul real de studenți este 1450, rezultă un spațiu ocupat:
- $MEMORIE = \max(nrMatricol) * dimensiune(Student)$   
 $= 55630 * 48 = 2,54 \text{ MB}$



# Tabele de dispersie

- ▶ Eliminarea dezavantajelor: tabele de dispersie (*hash tables*) – colecții de date în care, pe baza unei funcții *hash*, cheia de căutare este pusă în corespondență cu poziția elementului în cadrul colecției).
- ▶ Tabela de dispersie:
  - structură de stocare și căutare;
  - cheia de căutare asociată cu poziția elementului în colecția de date prin funcția *hash*.

# Tabele de dispersie

- ▶ Funcția de dispersie (*hash*) presupune găsirea unei funcții  $H()$  cu o valoare numerică unică pentru fiecare obiect considerat, într-un domeniu restrâns (utilizabil ca indice)  $\Rightarrow$  memorăm fiecare obiect  $x$  într-un tablou la indicele  $H(x)$ .
- ▶ Tehnica se numeste dispersie (*hashing*), ceea ce înseamnă că obiectele sunt dispersate într-un tablou (*hash table*).

# Tabele de dispersie

Avantaje ale utilizării tabelei de dispersie:

- ▶ utilizare mai eficientă a resursei memorie: nu se stochează elemente care nu sunt utilizate;
- ▶ funcție *hash* cu un nivel de complexitate scăzut:
  - $hash(X) = X \text{ modulo } 1500$
  - X aparține [54130, 55630]
- ▶ implementarea de chei alfanumerice: se poate utiliza tip alfanumeric pentru cheia de căutare.

# Tabele de dispersie

- ▶ Funcția *hash* translatează valoarea alfanumerică într-o valoare întreagă pozitivă;
- ▶ Funcția *hash* pentru un string (în limbajul Java):
  - $\text{hash}(S) = s[0] * 31^{n-1} + s[1] * 31^{n-2} + \dots + s[n-2] * 31 + s[n-1]$
  - $s[i]$  – codul ASCII ;
  - $n$  – dimensiunea șirului de caractere
- ▶ Ex:  $\text{hash}(\text{"salut"}) = 115 * 31^4 + 97 * 31^3 + 108 * 31^2 + 117 * 31 + 116 = 10920217$

# Tabele de dispersie

- ▶ Valoarea obținută este introdusă din nou într-o funcție *hash* pentru a identifica poziția corespondentă din colecția de date.
- ▶ Proprietăți ale funcțiilor de dispersie (*hash*):
  - să fie rapid calculabile (pentru eficiență);
  - să aibă o distribuție de valori cât mai uniformă, pentru a minimiza probabilitatea de coliziune (valori egale pentru obiecte diferite).

# Tabele de dispersie

- ▶ În limbajele pur obiectuale (C# sau Java) se folosesc colecții, realizându-se distincția între vectori și colecții.
- ▶ Vectorii au dezavantajul că au dimensiune fixă, iar colecțiile sunt redimensionabile dinamic, ele alocându-se element cu element.
- ▶ Elementele unui vector sunt de un anumit tip, în timp ce elementele unei colecții sunt de tip generic “object”; rezultă că o colecție poate conține elemente de orice tip.

# Tabele de dispersie

## ► Exemplu Java:

```
@Override
```

```
public int hashCode() {  
    return 31*firstName.hashCode() + lastName.hashCode();  
}
```

```
Hashtable<Integer,ArrayList<String>> ht = new  
    Hashtable<Integer,ArrayList<String>>();  
ht.put(2215,new ArrayList<String>());  
ht.get(2215).add("Java");  
ht.get(2215).add("C#");  
ht.put(2320,new ArrayList<String>());  
ht.get(2320).add("POO");  
ht.get(2320).add("SDD");  
System.out.println("La sala 2215:" + ht.get(2215));  
System.out.println("La sala 2320:" + ht.get(2320));
```

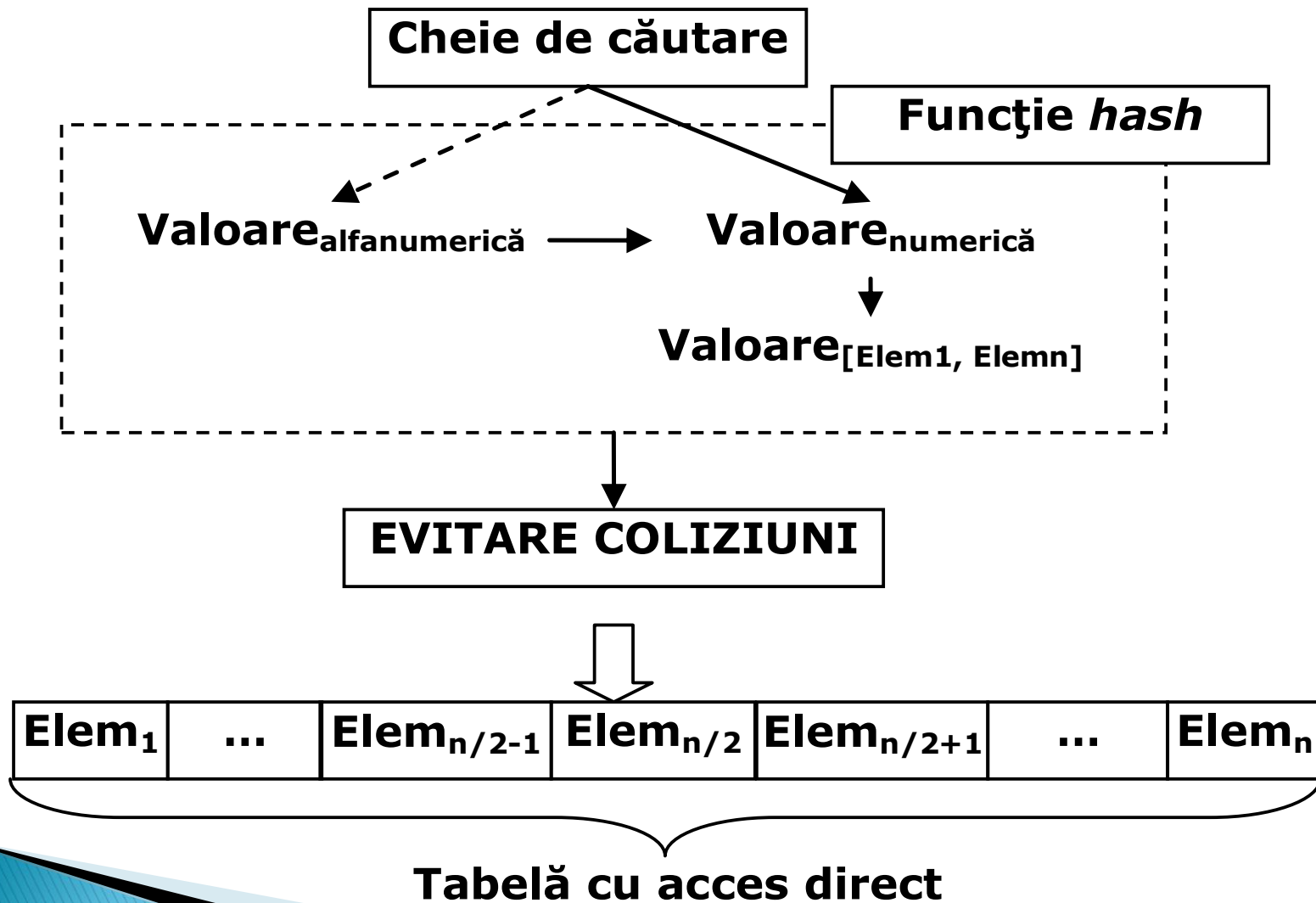
# Tabele de dispersie

Dezavantajul tabelelor de dispersie:

- ▶ prelucrarea suplimentară dată de funcția *hash* care poate avea în unele situații un nivel de complexitate ridicat;
- ▶ apariția în cadrul tabelii a coliziunilor: două valori  $X_h$  și  $Y_h$  conduc la  $\text{hash}(X_h) = \text{hash}(Y_h)$ ; evitarea coliziunilor se face prin operații suplimentare, precum: chaining, re-hashing, linear probing, quadratic probing și overflow area.



# Tabele de dispersie



# Tabele de dispersie

În funcție de tipul valorii cu rol de cheie:

- ▶ *chei numerice*: tipuri fundamentale definite de limbajul de programare utilizat;
- ▶ *chei alfanumerice*: șiruri de caractere;
- ▶ *chei compuse*: mai multe attribute.

# Tabele de dispersie

Funcția *hash*:

- ▶ prelucrează cheia asociată fiecărei înregistrări;
- ▶ determină poziția în cadrul tablei de dispersie a elementului;
- ▶ nu există o funcție *hash* generală;
- ▶ alegerea funcției *hash* se face în funcție de caracteristicile mulțimii de valori chei.

# Tabele de dispersie

Modele matematice ale funcției *hash*:

- ▶ împărțire în modul: complexitate scăzută, ușurință de implementare; cheia de căutare este transformată într-o valoare numerică și apoi transpusă în mulțimea  $[0; n-1]$ ,  $n$  – dimensiunea tabeli de dispersie:

**pozitie\_tabela = val\_cheie % val\_baza**

pozitie\_tabela: valoarea *hash* obținută

val\_cheie: valoare cheie numerică

val\_baza: dimensiunea tabeli de dispersie; numere prime apropiate de numărul total de înregistrări; caz general:  $(4*i+3)$  cu  $i = 0, 1, 2, 3, \dots$

# Tabele de dispersie

Modele matematice ale funcției *hash* (continuare):

- ▶ înmulțirea cu un număr real aleatoriu din  $[0;1)$  și prelucrarea ulterioară a părții zecimale cuprinsă în  $[0;1)$ ; înmulțirea rezultatului cu dimensiunea tablei de dispersie  $n$  duce la obținerea poziției elementului în  $[0; n-1]$ ;

$$\text{val\_hash} = ((\text{val\_cheie} * \text{random}_{[0;1)}) - [(\text{val\_cheie} * \text{random}_{[0;1)})]) * n$$

val\_hash: valoare *hash*

val\_cheie: valoare cheie de căutare

random<sub>[0;1)</sub>: număr aleatoriu din  $[0;1)$

$n$ : dimensiune tabelă

# Tabele de dispersie

Modele matematice ale funcției *hash* (continuare):

- ▶ prelucrarea codurilor ASCII ale caracterelor alfanumerice: pe baza primului caracter din cheie se definește relația:

$$\text{val\_hashs}_1 = \text{string\_cheie}[0] \% 255$$

val\_hashs<sub>1</sub>: valoarea *hash*

string\_cheie: valoare cheie de căutare

- ▶ val\_hashs<sub>1</sub>: model cu un nivel de complexitate scăzut pentru gestiunea unei colectivități mici de elemente.
- ▶ modelul este ineficient deoarece generează multe coliziuni pentru șiruri diferite care încep cu același caracter.

# Tabele de dispersie

- ▶ Rafinarea modelului: preluarea mai multor caractere din șirul pentru care se determină valoarea hash (primul și ultimul caracter):
- ▶  $val\_hashs_2 = (string\_cheie[0] + string\_cheie[lungime_{string\_cheie}] ) \% n$
- ▶  $val\_hashs_2$  : valoare *hash*
- ▶  $string\_cheie$ : cheie de căutare
- ▶  $lungime_{string\_cheie}$  : dimensiune șir de caractere
- ▶  $n$ : dimensiune tabelă de dispersie

# Tabele de dispersie

- ▶ Pentru a nu reduce dimensiunea tabelii la maxim 255 elemente, se utilizează un număr prim,  $n$ , suficient de mare.
- ▶ Alte funcții *hash* de prelucrare a cheilor alfanumerice analizează toate caracterele din șir:

- ▶  $\text{val\_hashs}_3 = \sum_{i=1}^{\text{lungime\_cheie}} \text{ASCII}(\text{string\_cheie}[i]) \% n$



# Tabele de dispersie

Evitarea coliziunilor:

- ▶ metode de regăsire a elementelor descrise de chei cu valori diferite, dar care conduc la valori *hash* identice;
- ▶ chaining, re-hashing, linear probing, quadratic probing, overflow area.

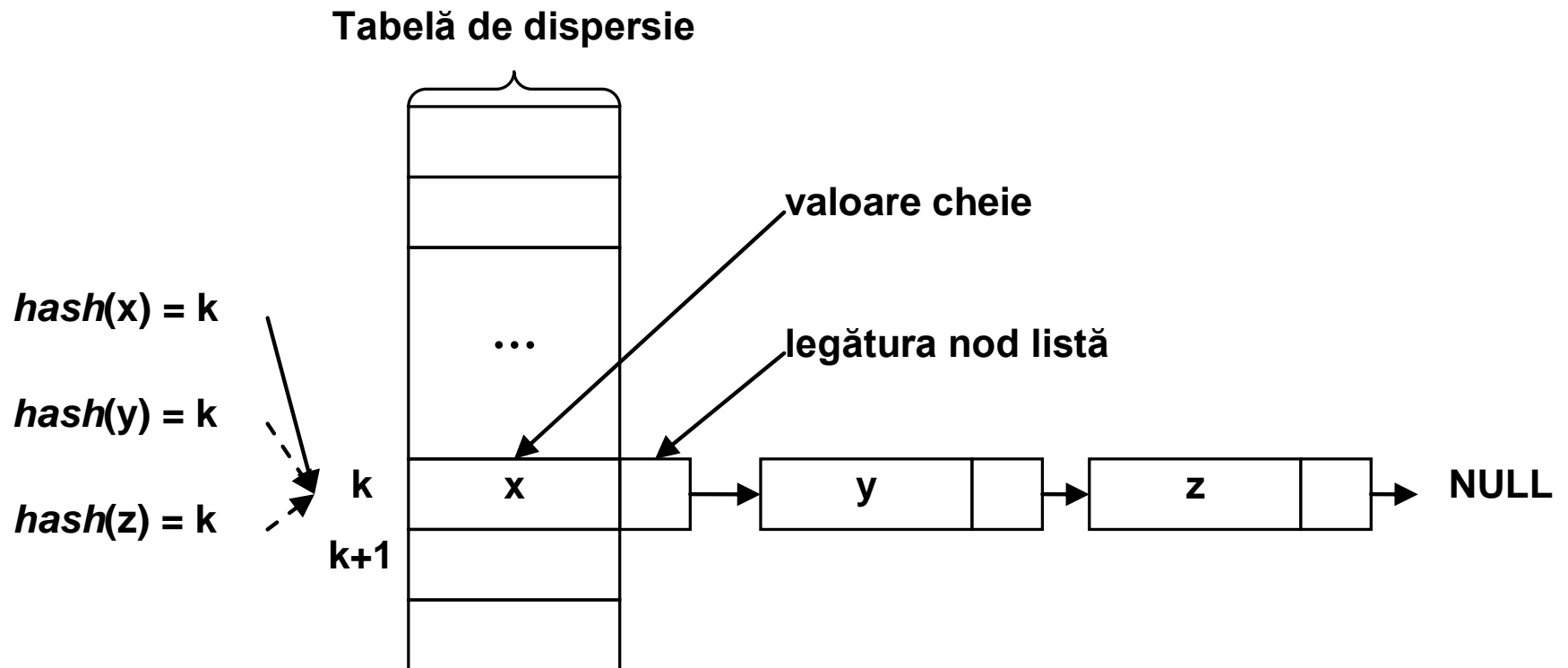
# Tabele de dispersie

## Chaining:

- ▶ implementează lucrul cu liste;
- ▶ fiecare poziție din tabela de dispersie conține adresa unei liste de elemente cu valori *hash* egale;
- ▶ regăsirea unui element: determinarea poziției în cadrul tabelii prin calculul valorii *hash* și parcurgerea secvențială a listei atașate poziției.

# Tabele de dispersie

## ► Chaining:



# Tabele de dispersie

## ► Chaining:

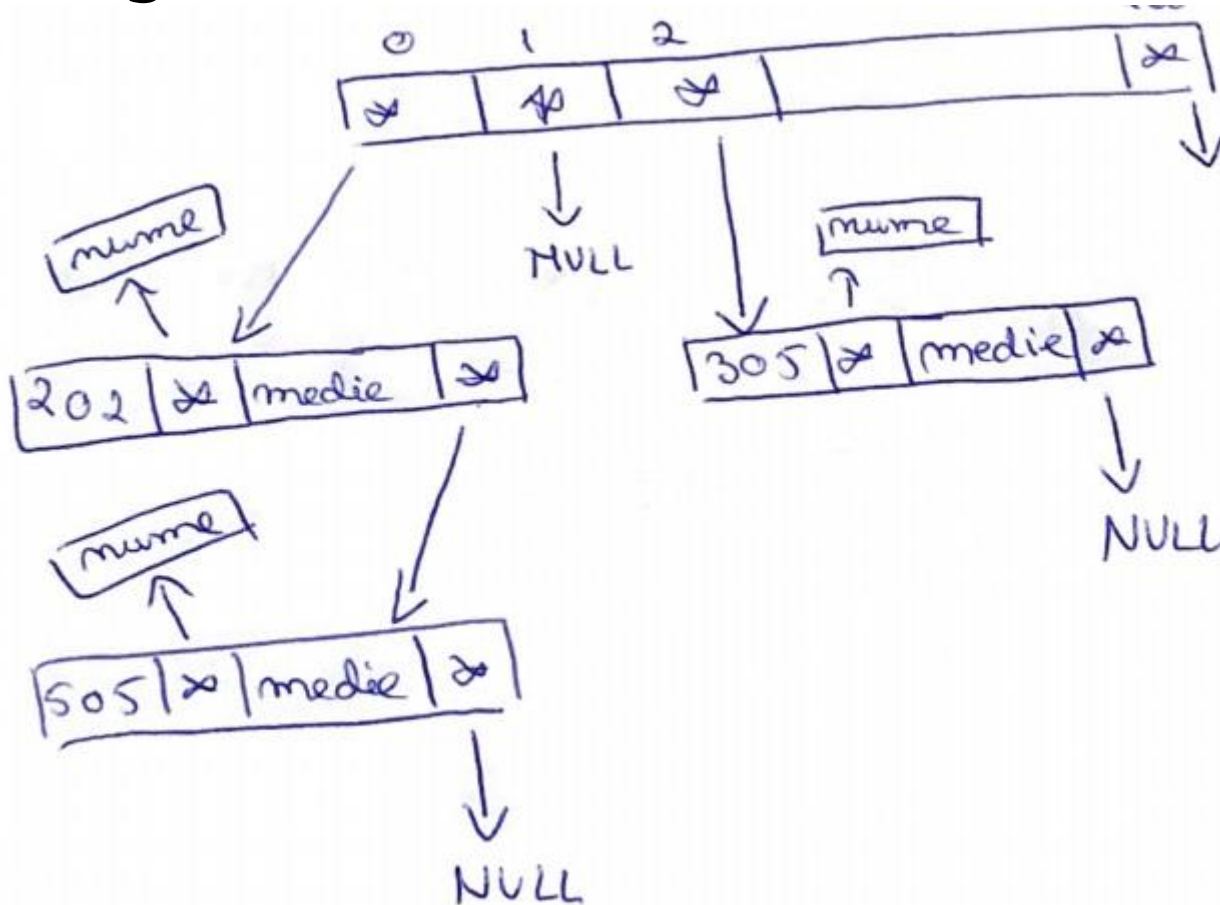
```
typedef struct
{
    int cod;
    char *nume;
    float medie;
} student;
```

```
typedef struct
{
    student inf;
    struct nodLS *next;
} nodLS;
```

```
typedef struct
{
    struct nodLS **vect;
    int size;
} hashT;
```

# Tabele de dispersie

## ► Chaining:



# Tabele de dispersie

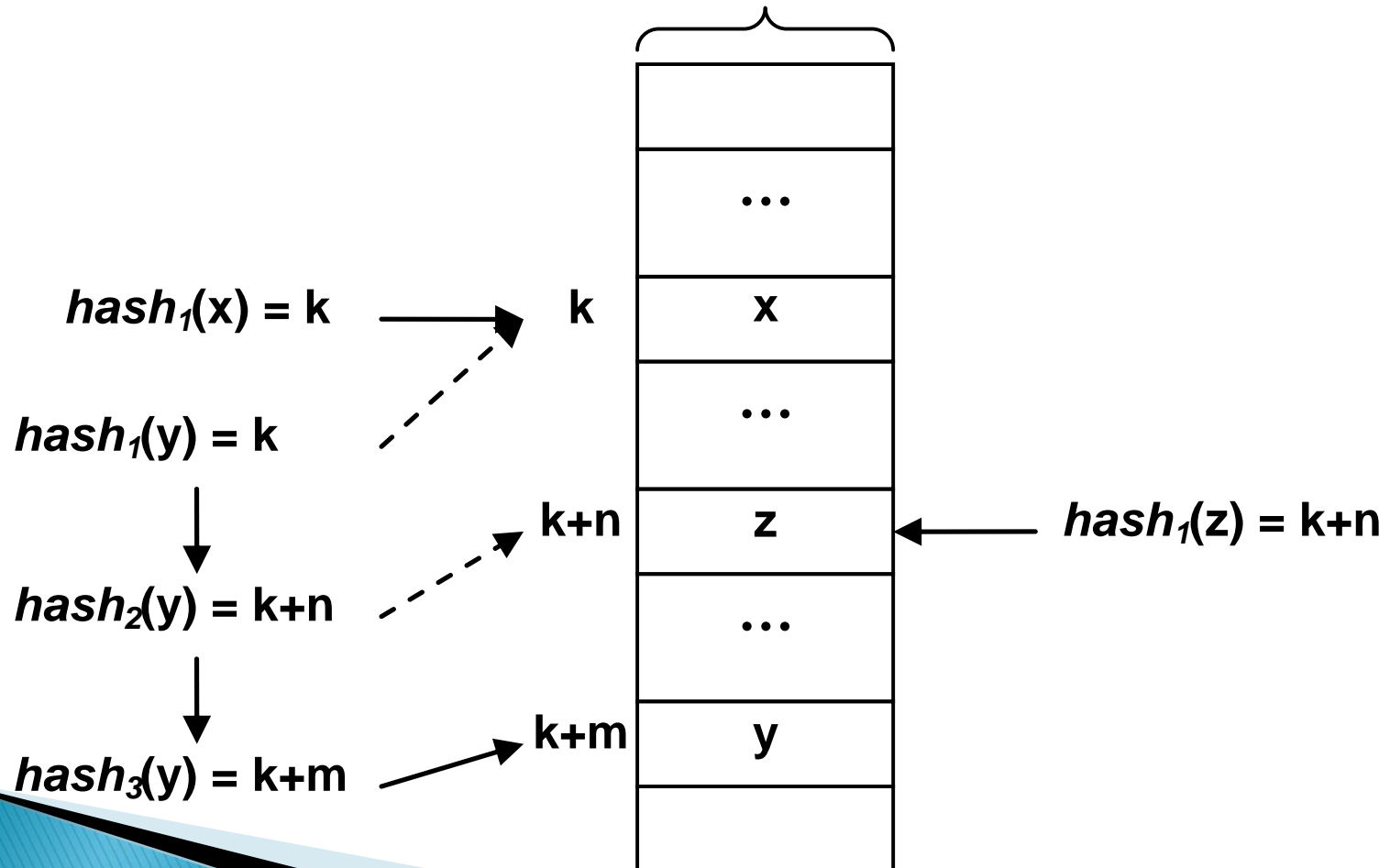
## Re-hashing:

- ▶ aplicarea în cascadă a aceleiași funcții *hash* sau a altui model dintr-o mulțime de funcții, până când valoarea obținută reprezintă o poziție liberă din tabela de dispersie;
- ▶ la fiecare pas al procesului de căutare: valoarea cheii de căutare este introdusă într-o listă de funcții *hash* până când se identifică elementul cu valoarea căutată sau nu mai există alte posibilități de a recalcula valoarea *hash*.

# Tabele de dispersie

## ► Re-hashing:

Tabelă de dispersie



# Tabele de dispersie

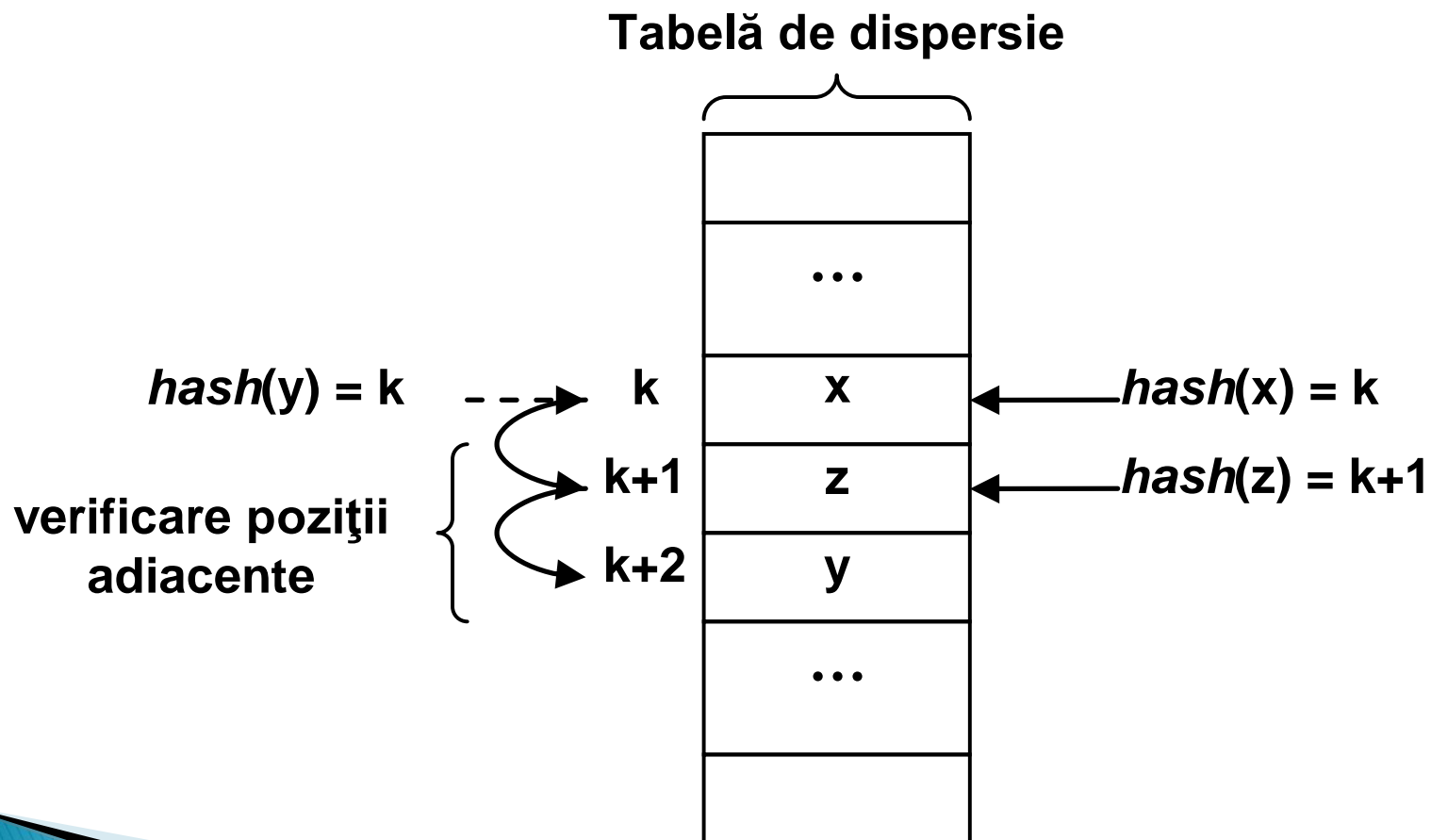
## Linear probing:

- ▶ căutarea secvențială a primei poziții libere unde este inserat elementul nou (la stânga sau la dreapta coliziunii);
- ▶ la căutare: verificarea elementelor adiacente poziției indicate de valoarea *hash*;
- ▶ gruparea coliziunilor de același tip în aceeași zonă (cluster); rezultă creșterea probabilității de apariție a coliziunilor pentru valorile *hash* adiacente.



# Tabele de dispersie

## ► Linear probing:



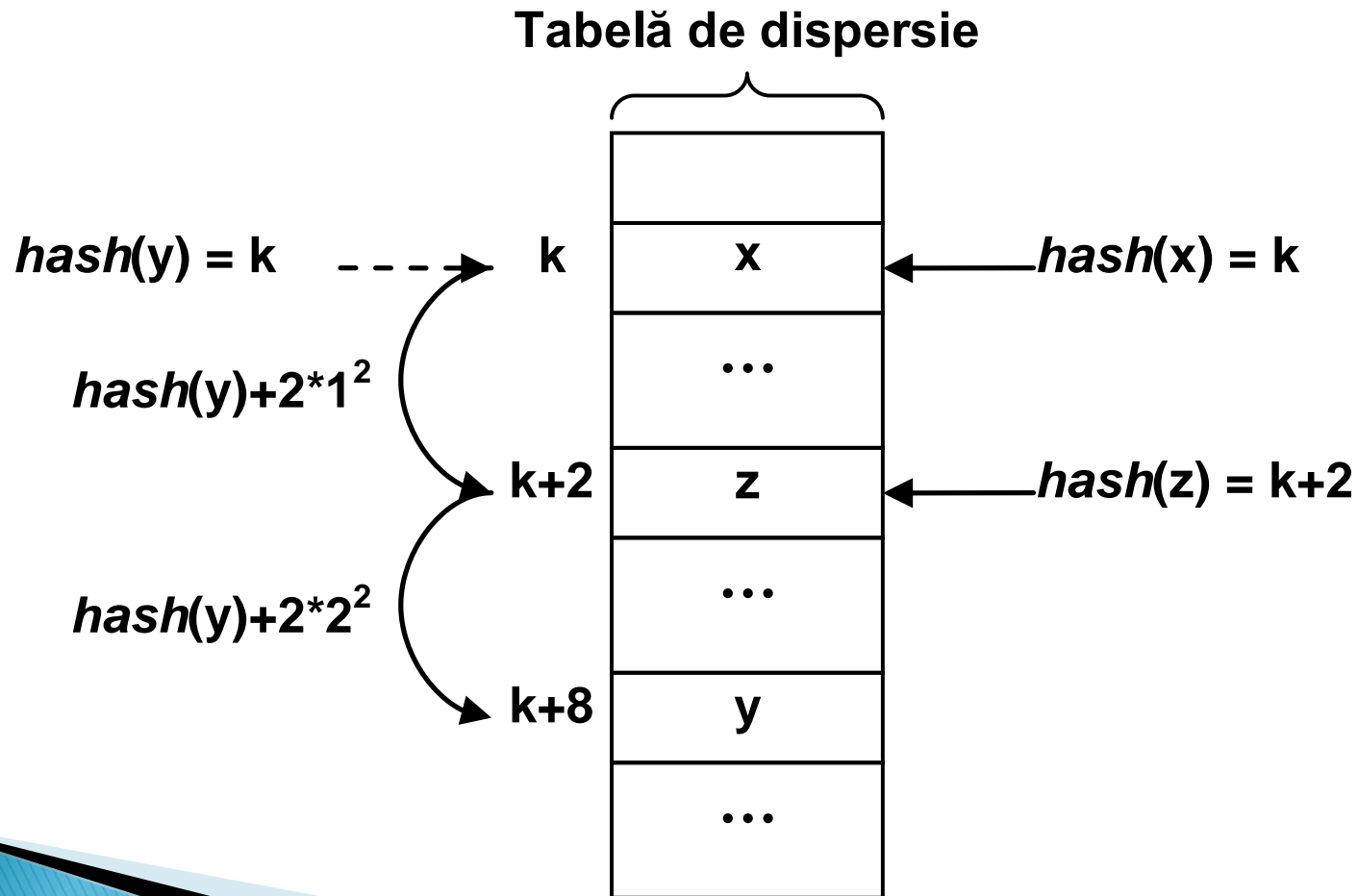
# Tabele de dispersie

## Quadratic probing:

- ▶ evită crearea grupurilor de coliziuni prin utilizarea unui pas de regăsire a următoarei poziții libere diferită de 1; salturi în tabela de dispersie din două în două poziții sau din patru în patru;
- ▶ determinarea următoarei poziții de inserat:
  - **$\text{poziție} = \text{hash}(X) + c \cdot i^2$**
  - poziție: noua poziție din tabelă pentru inserare sau căutare element
  - X: cheia asociată elementului
  - $\text{hash}(X)$ : poziția indicată de valoarea hash a elementului
  - c: valoare constantă {1, 2, 4}
  - i: număr operație re-hash sau număr de poziții verificate

# Tabele de dispersie

- ▶ Quadratic probing:



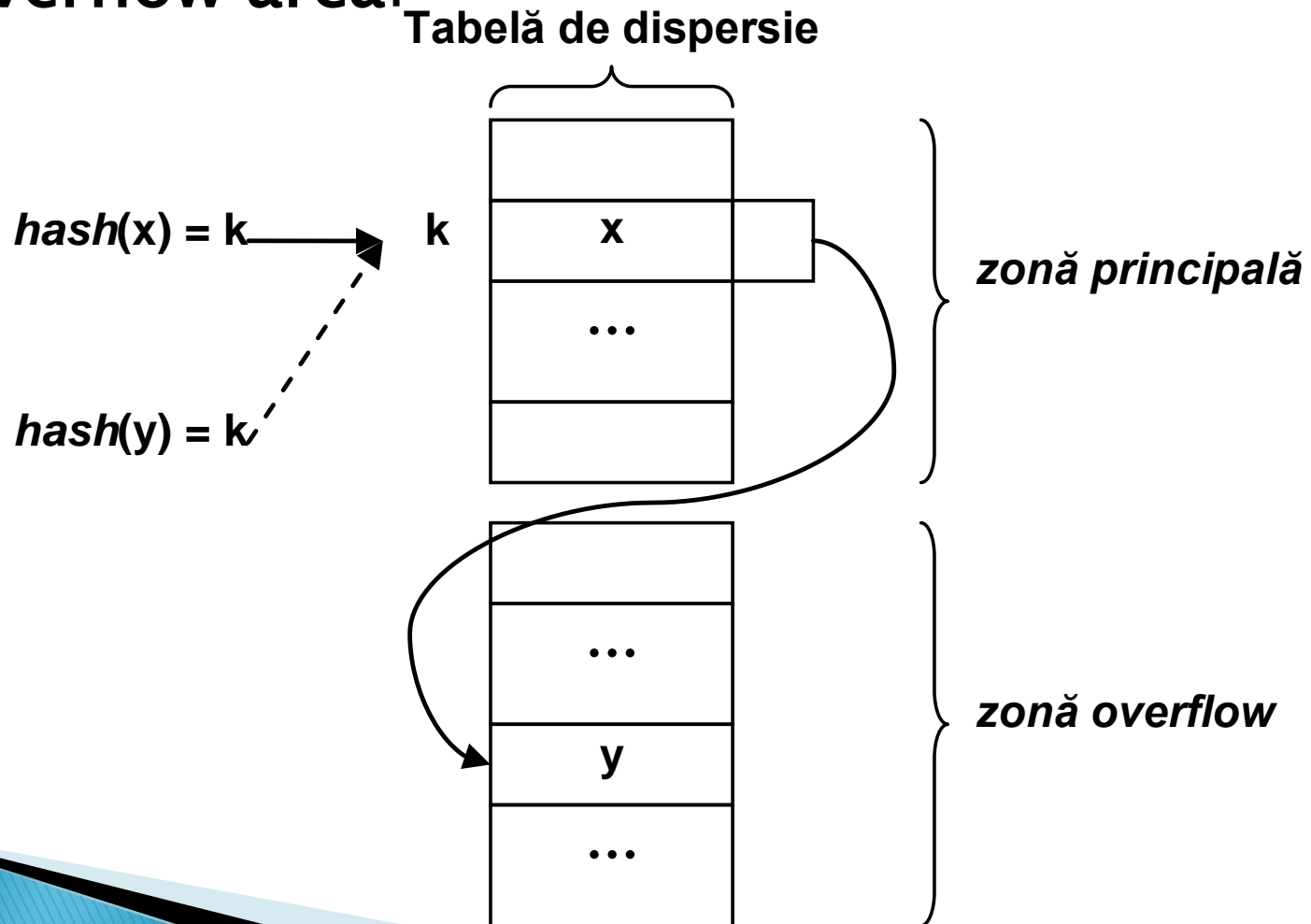
# Tabele de dispersie

## Overflow area:

- ▶ împarte tabela de dispersie în:
  - zona primară: reținerea elementelor inițiale;
  - zona secundară alocată elementelor ce generează coliziuni;
- ▶ se utilizează un element al zonei secundare pentru a reține noua valoare sau pentru a continua căutarea;
- ▶ accesul la zona secundară: prin pointer din zona primară;
- ▶ regăsire mai rapidă a informațiilor decât metoda chaining.

# Tabele de dispersie

## ► Overflow area:



# Tabele de dispersie

- ▶ Probabilitatea de apariție a coliziunilor la inserare sau la căutare crește proporțional cu gradul de utilizare a tabelii.
- ▶ Funcțiile *hash* cu un grad redus de complexitate nu conduc la rezultate unice pentru valori de intrare distincte.

# Tabele de dispersie

- ▶ Cu cât tabela are un număr din ce în ce mai mic de poziții disponibile, cu atât crește riscul de a avea elemente cu chei de căutare diferite, dar care se regăsesc pe poziții identice.
- ▶ Eficiența operației de căutare la un nivel acceptabil: grad de ocupare a tablei de dispersie  $< 50\%$  (ineficiență a spațiului, viteză de căutare mare).

# Bibliografie

- ▶ Marius Popa, Cristian Ciurea, Mihai Doinea, Alin Zamfiroiu – *Structuri de date: teorie și practică*, Editura ASE, București, 2023, 280 pg.
  - Cap. 5. Tabela de dispersie