מערכים, ורשימות מקושרות שמושים

חומר קריאה לשיעור זה

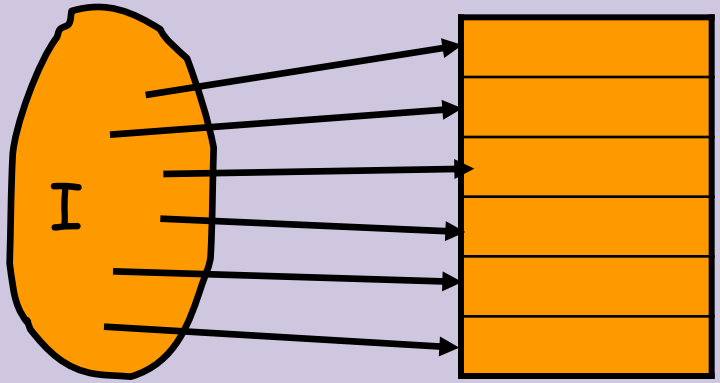
Chapter 11.2- Linked lists (204 - 213)

©Geiger & Itai, 2001

מערכים

©cs,Technion

מערך כטיפוס נתונים אבסטרקטי



מערך מוגדר ע"י הפעולות הבאות:

create(type,I) – מחזיר מערך A של איברים מטיפוס type כאשר האינדקסים הם הקבוצה הסופית I.

get(A,i) – מחזיר את הערך של האיבר עם אינדקס $i \in I$ בתוך A.

store(A,i,e) – מאחסן במערך A, תחת אינדקס $i \in I$, את ערך הביטוי e.

בשפת c האינדקסים הם הקבוצה $I=\{0,\dots,n-1\}$

אחזר (get) A[i]

שמור (store) A[i] = e

כללים הנשמרים ע"י הפעולות:

- כל הערכים במערך מאותו טיפוס, והוא נקבע בהכרזה (type).
- הערך המאוחזר לפי אינדקס i , הוא הערך האחרון שנשמר לפי אינדקס i.
- אחזור לפי אינדקס i מחזיר ערך בלתי מוגדר אם מעולם לא אוחסן ערך לפי אינדקס זה.

©cs,Technion

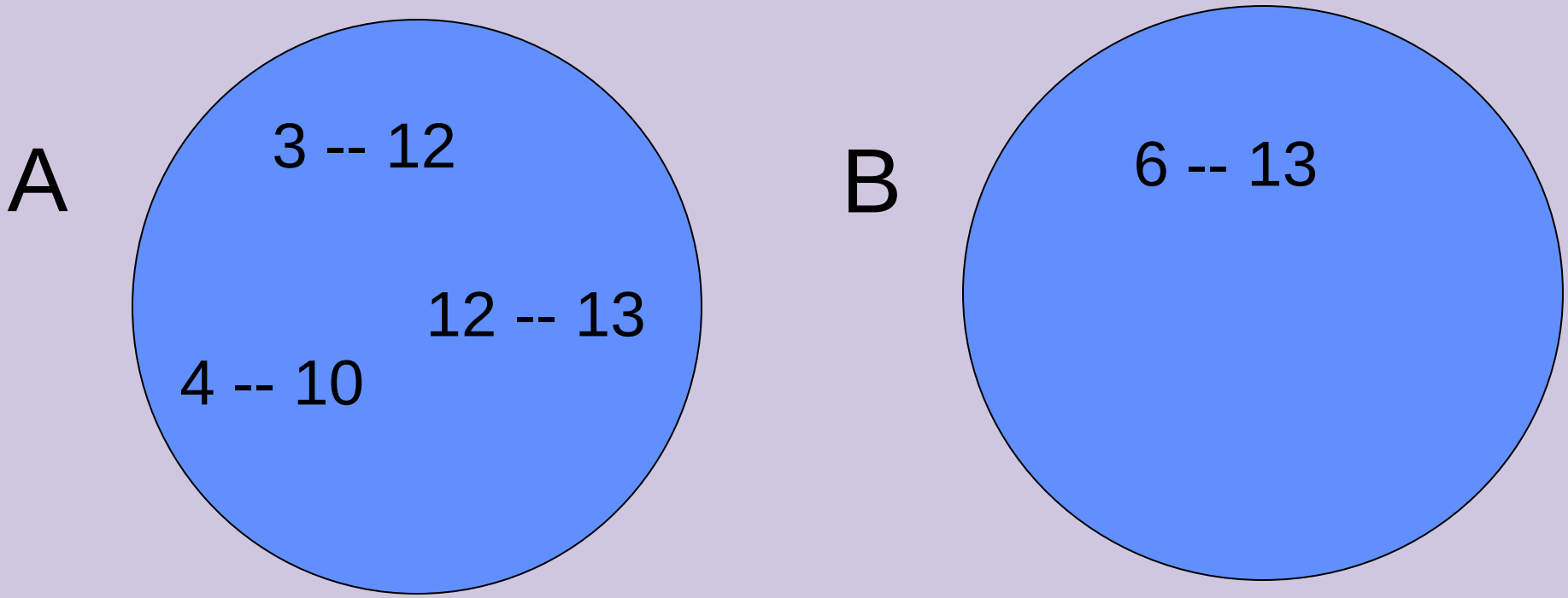
Creat(type,I)	מחזיר מערך מטיפוס type עם אינדקסים מקבוצה I
Get(A,i)	מחזיר את הערך של האיבר עם אינדקס i בתוך A
Store(A,i,e)	מאחסן במערך A תחת אינדקס i את ערך הביטוי e

©cs,Technion

A=Create(integer,{0,…,20})

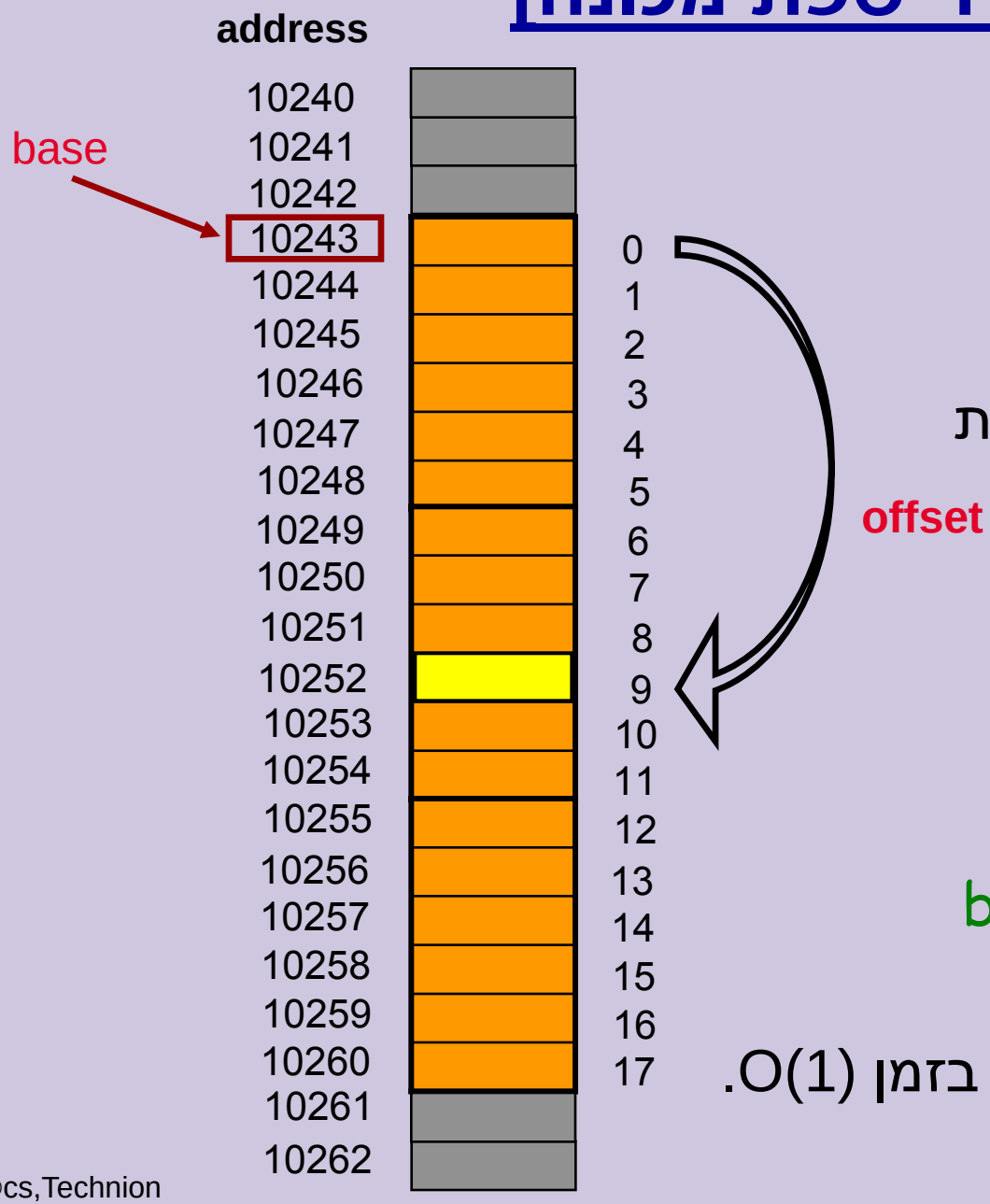
Store(A,3,12) Store(A,4,10) B=Create(integer,{0,3,6,9})

Store(B,6,Get(A,3)+1) Store(A, Get(A,3),Get(B,6))



©cs,Technion

מימוש למערך (1) [קומפילר-שפת מכונה]



אזור זיכרון רצוף

הפעולות: מתבססות על חישובי כתובות

base+offset

דוגמא: n=9

הכתובת של A[9] היא:

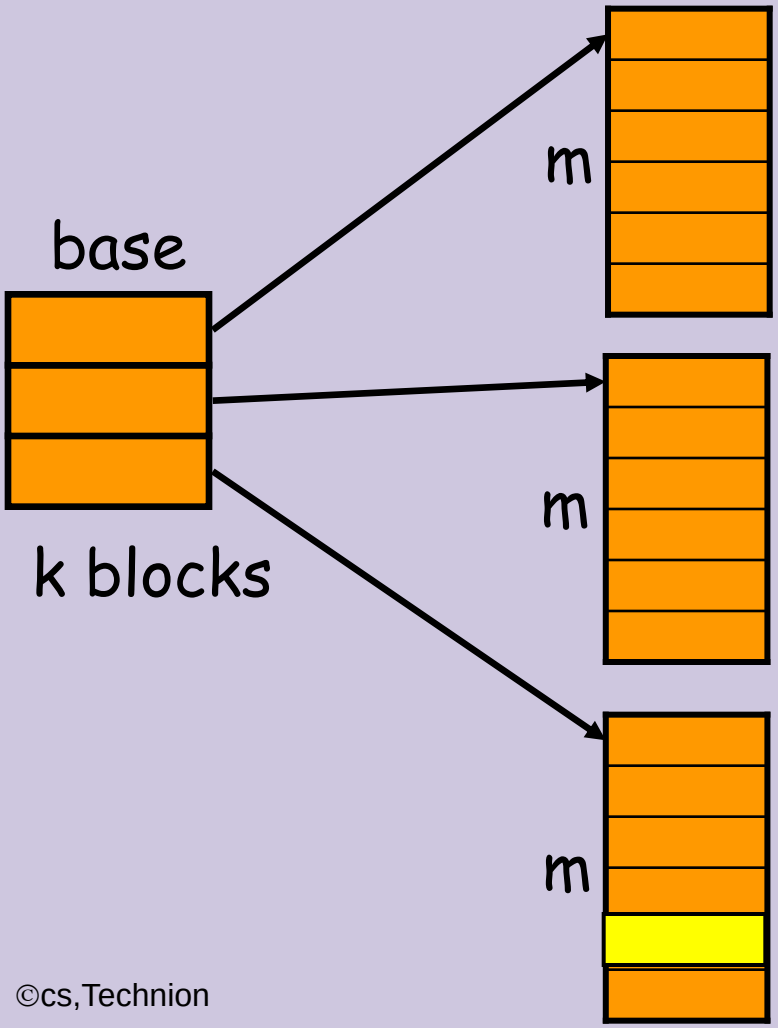
base+ offset = 10243 + 9 = 10252

כל הפעולות (חוץ מ- Create) מתבצעות בזמן O(1).

©cs,Technion

מימושים למערך (2)

מימוש באמצעות מספר אזורים בזיכרון: לכל אזור גודל קבוע m.



כדי לממש מערך בגודל n יש לבחור $k=\lceil n/m \rceil$.

נַקְצֶה מערך עזר (base) בגודל k

האיבר $A[i]$ נמצא בכתובת $base[\lfloor i/m \rfloor] + i\%m$.

דוגמא: $m=6$

הכתובת של $A[16]$ היא:

$base[\lfloor 16/6 \rfloor] + 16\%6 = base[2] + 4$

זמן חישוב הכתובת: $O(1)$.

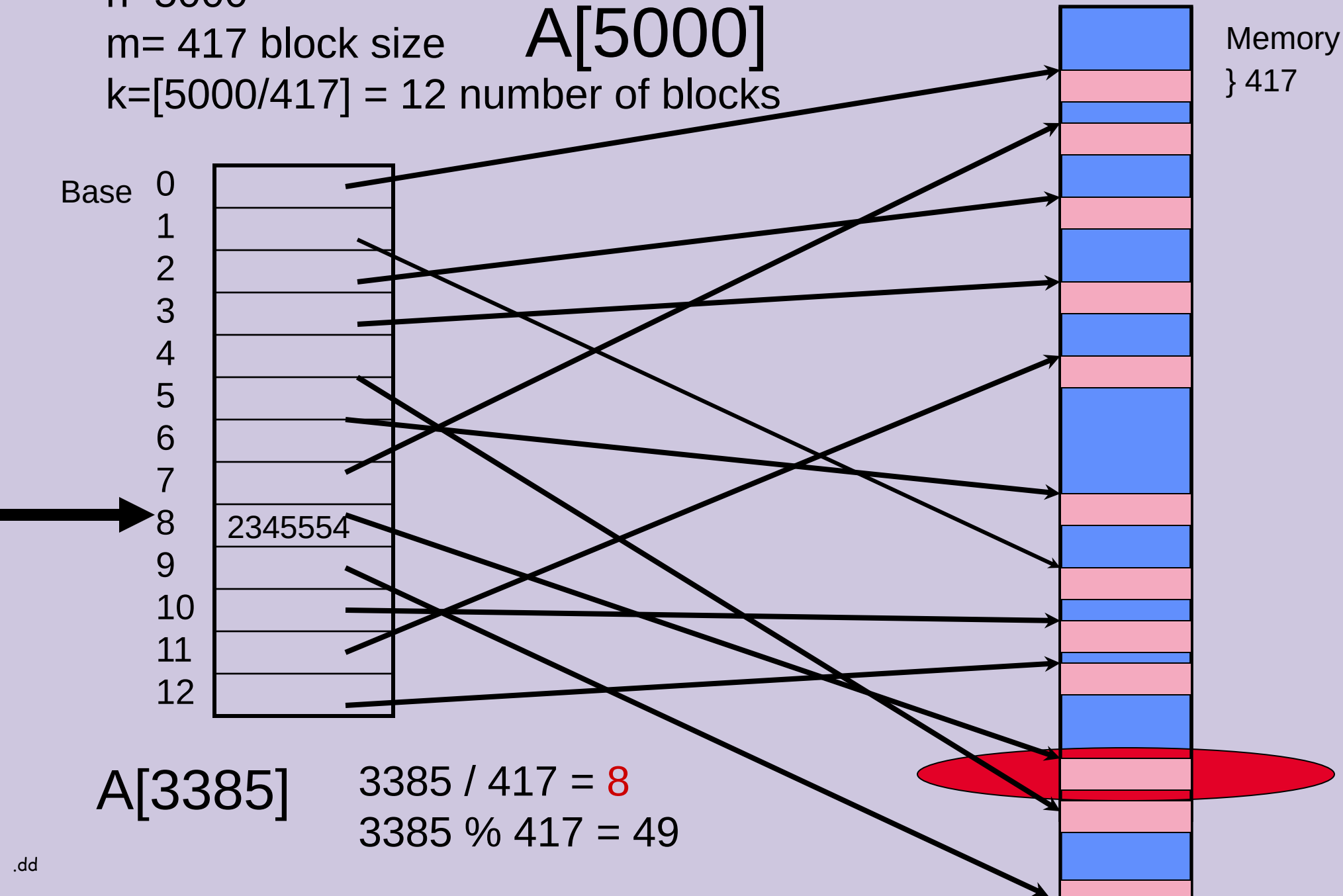
©cs,Technion

n=5000

m= 417 block size

k=[5000/417] = 12 number of blocks

A[5000]

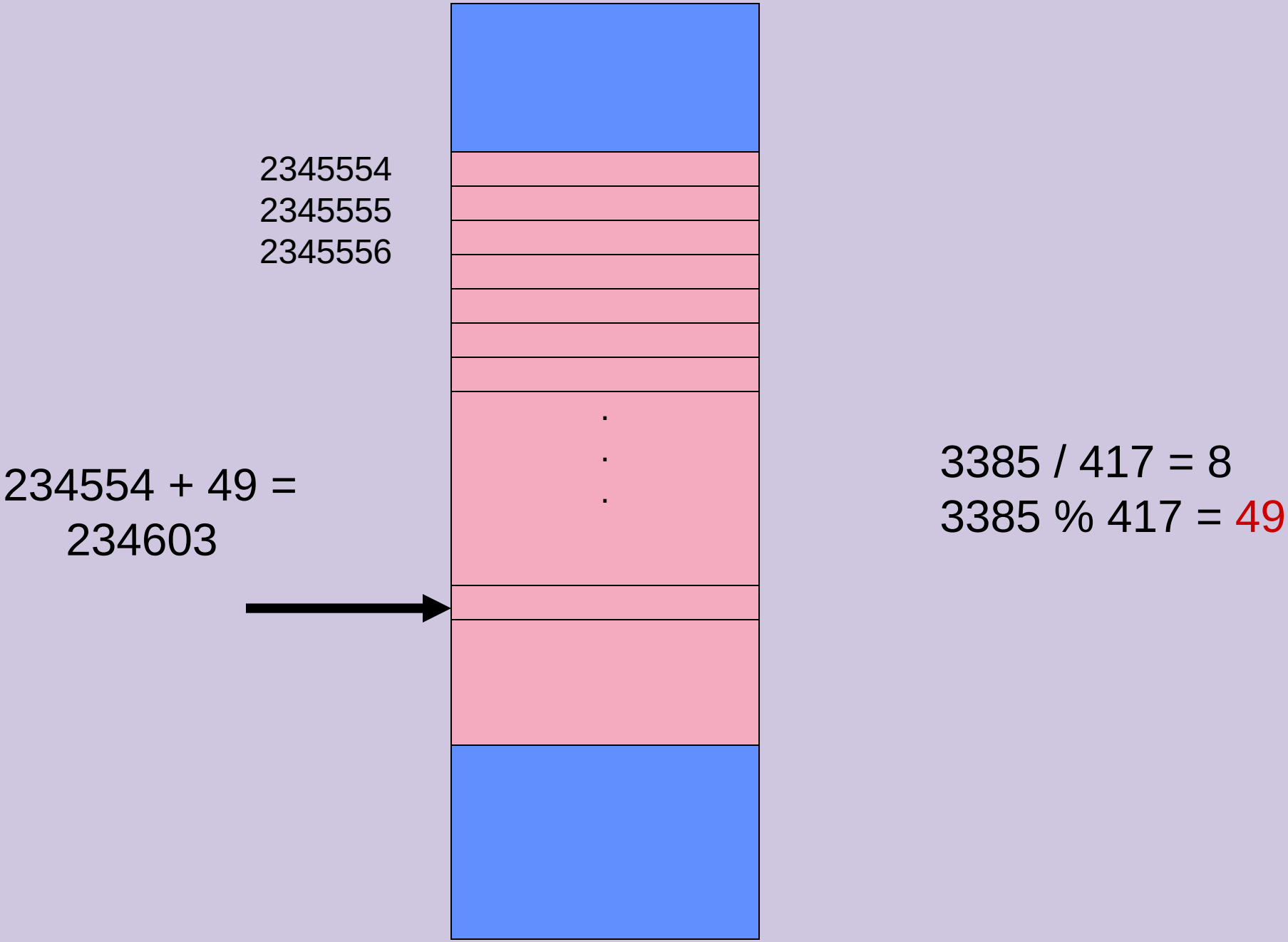


A[3385]

3385 / 417 = 8

3385 % 417 = 49

.dd


2345554
2345555
2345556
234554 + 49 =
234603
3385 / 417 = 8
3385 % 417 = 49

©cs,Technion

מערך דו-ממדי

int a[m][n]

כל שורה $A[i]$ היא מערך חד-ממדי באורך n.

$A_{0,0}$	$A_{0,1}$						$A_{0,n-1}$
$A_{1,0}$	$A_{1,1}$						$A_{1,n-1}$
...							
$A_{m-1,0}$	$A_{m-1,1}$						$A_{m-1,n-1}$

נניח ש- base היא הכתובת של $A[0][0]$ והמימוש הוא באזור זיכרון רצוף.

הכתובת של שורה $A[i]$ היא $base + i \cdot n$

הכתובת של איבר $A[i][j]$ היא $base + i \cdot n + j$

©cs,Technion

מערך רב-ממדי (1)

במערך דו-ממדי: $int\ A[n_2][n_1]$ הכתובת של איבר $A[i_2][i_1]$ מחושבת, כפי שראינו, ע"י

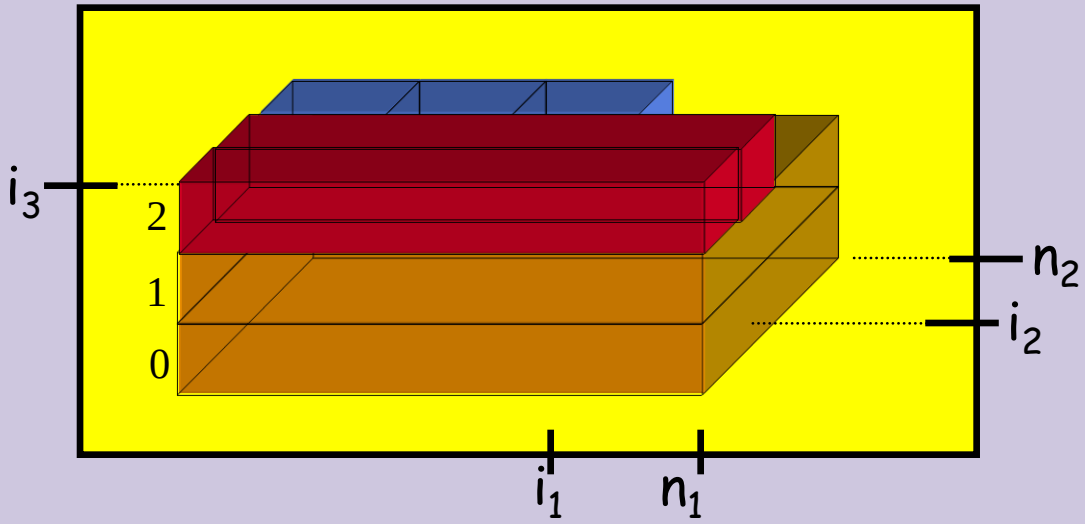
הנוסחה: $base + i_2 \cdot n_1 + i_1$

במערך תלת-ממדי: $int\ A[n_3][n_2][n_1]$ הכתובת של איבר $A[i_3][i_2][i_1]$ מחושבת ע"י

הנוסחה: $base + i_3 \cdot n_2 \cdot n_1 + i_2 \cdot n_1 + i_1$

$$i_3 \cdot \underbrace{n_2 \cdot n_1}_{\text{גודל מישור}} + i_2 \cdot \underset{\uparrow}{n_1} + i_1$$

אורך שורה



מערך רב-ממדי (2)

במערך d-ממדי: int $A[n_d]\ldots[n_3][n_2][n_1]$

הכתובת של $A[i_d]\ldots[i_3][i_2][i_1]$ מחושבת ע"י הנוסחה:

$$\text{base} + i_d \cdot n_{d-1} \cdots n_1 + i_{d-1}\, n_{d-2} \cdots n_1 + \ldots + i_1$$

$$= \text{base} + \sum_{k=1}^{d} i_k \cdot \prod_{\ell=1}^{k-1} n_\ell$$

כמה פעולות במקרה הגרוע? $O(d^2)$

אבל אפשר לבטא את הכתובת באופן אלטרנטיבי:

$$= \text{base} + ((\ldots((i_d n_{d-1}+i_{d-1})n_{d-2} + i_{d-2})n_{d-3} + \ldots + i_3)\, n_2+i_2)n_1 + i_1$$

כמה פעולות במקרה הגרוע? $O(d)$

מערך רב-ממדי (3)

הנוסחה לחישוב הכתובת במערך רב ממדי דומה לכלל Horner לחישוב ערך פולינום:

$$p(x_0) = a_k x^k + a_{k-1}x^{k-1} + \ldots + a_1 x + a_0 = (\ldots(a_k x + a_{k-1})x + \ldots + a_1)x + a_0$$

התוכנית הבאה מחשבת את ערך הפולינום בנקודה x:

```
p=a[k];
for (j=k-1; j>=0; j--)
    p = p·x + a[j]
```

באופן אנלוגי, התוכנית הבאה מחשבת את הכתובת של $A[i_d]\ldots[i_3][i_2][i_1]$ הנתונה ע"י:

$$base + ((\ldots((i_d n_{d-1}+i_{d-1})n_{d-2} + i_{d-2})n_{d-3} + \ldots + i_3)\, n_2+i_2)n_1 + i_1$$

```
addr=i[d];
for (j=d-1; j>=1; j--)
        addr = addr*n[j] + i[j];
addr= base + addr;
```

איתחול מערך

שאלה: כמה עולה לאתחל מערך באורך n?

תשובה: איתחול נאיבי – $O(n)$

נתאר מבנה נתונים לאיתחול מערך בזמן קבוע!

הרעיון: לא באמת נאתחל את המערך. נשתמש בזיכרון נוסף לציין האם כבר נכתב ערך בתא במערך

©cs,Technion

Get(A,i)

A[i] if i is defined

if i is not defined

Store(A,i,x)

A[i] := x

Init(A)

A[i] := # for all i

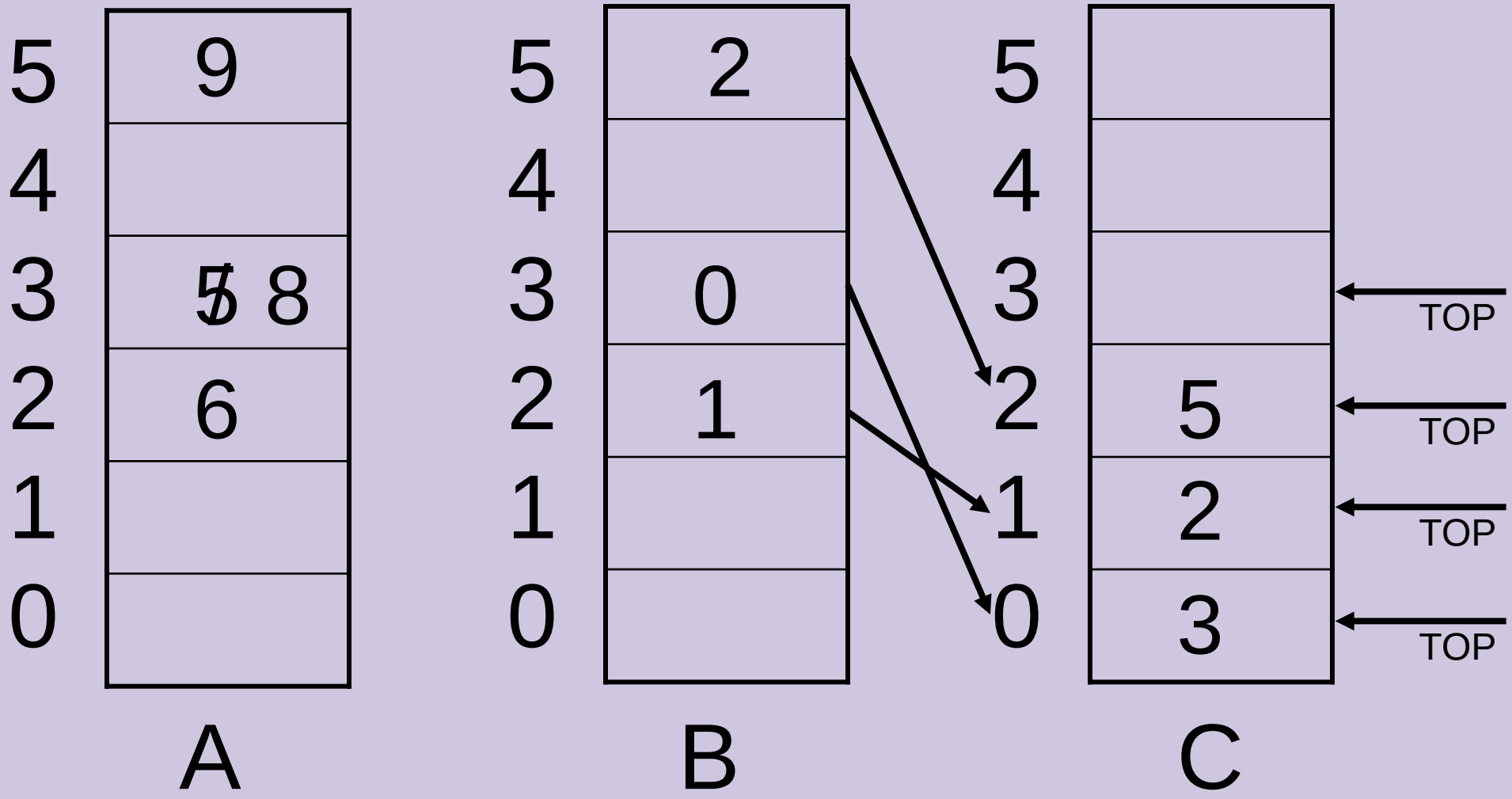
©cs,Technion

Store(A,3,5) A[3] := 5

Store(A,2,6) A[2] := 6

Store(A,5,9) A[5] := 9

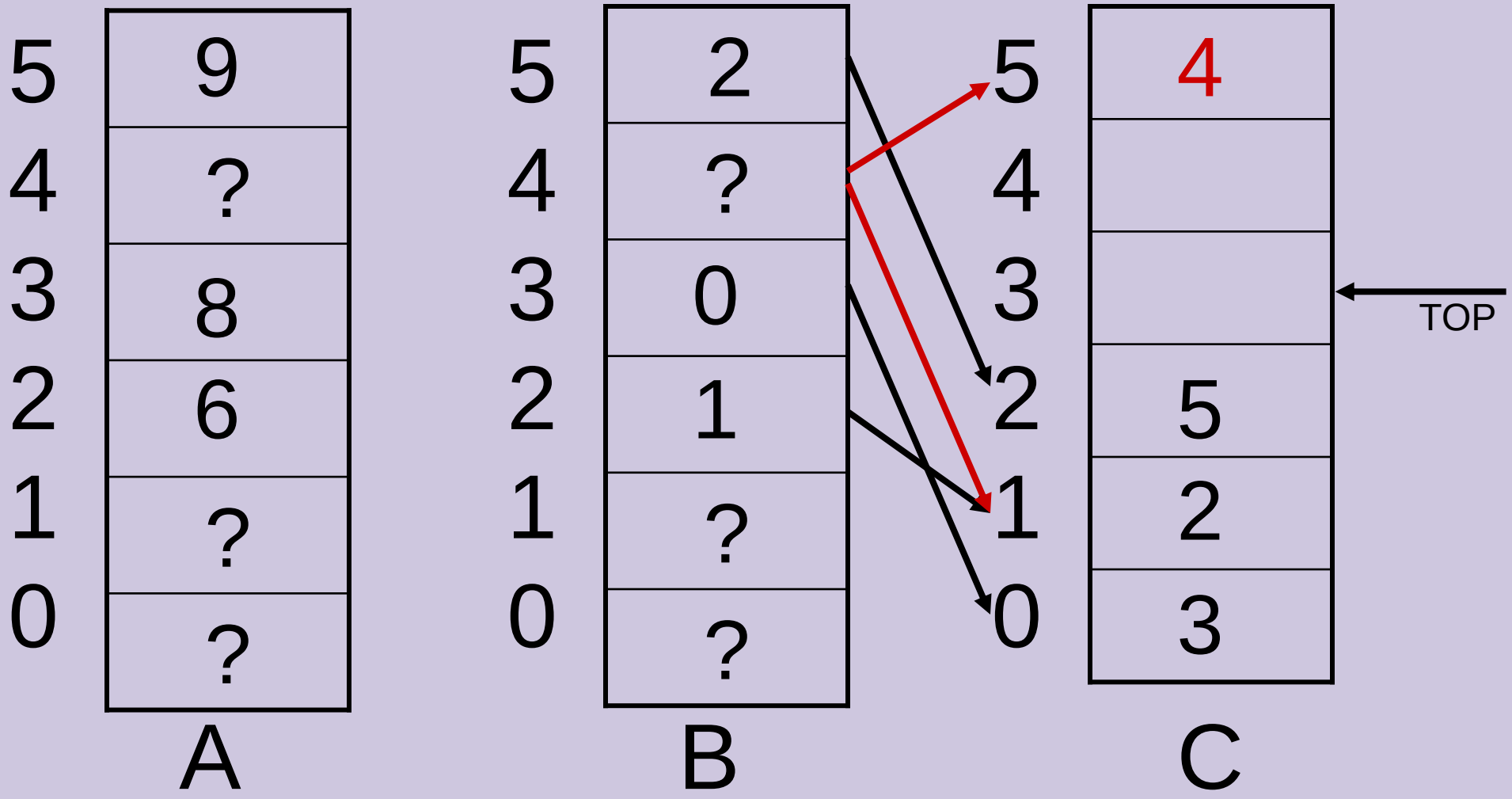
Store(A,3,8) A[3] := 8



3 is in the stack C → B[3] < TOP C[B[3]] = 3

©cs,Technion

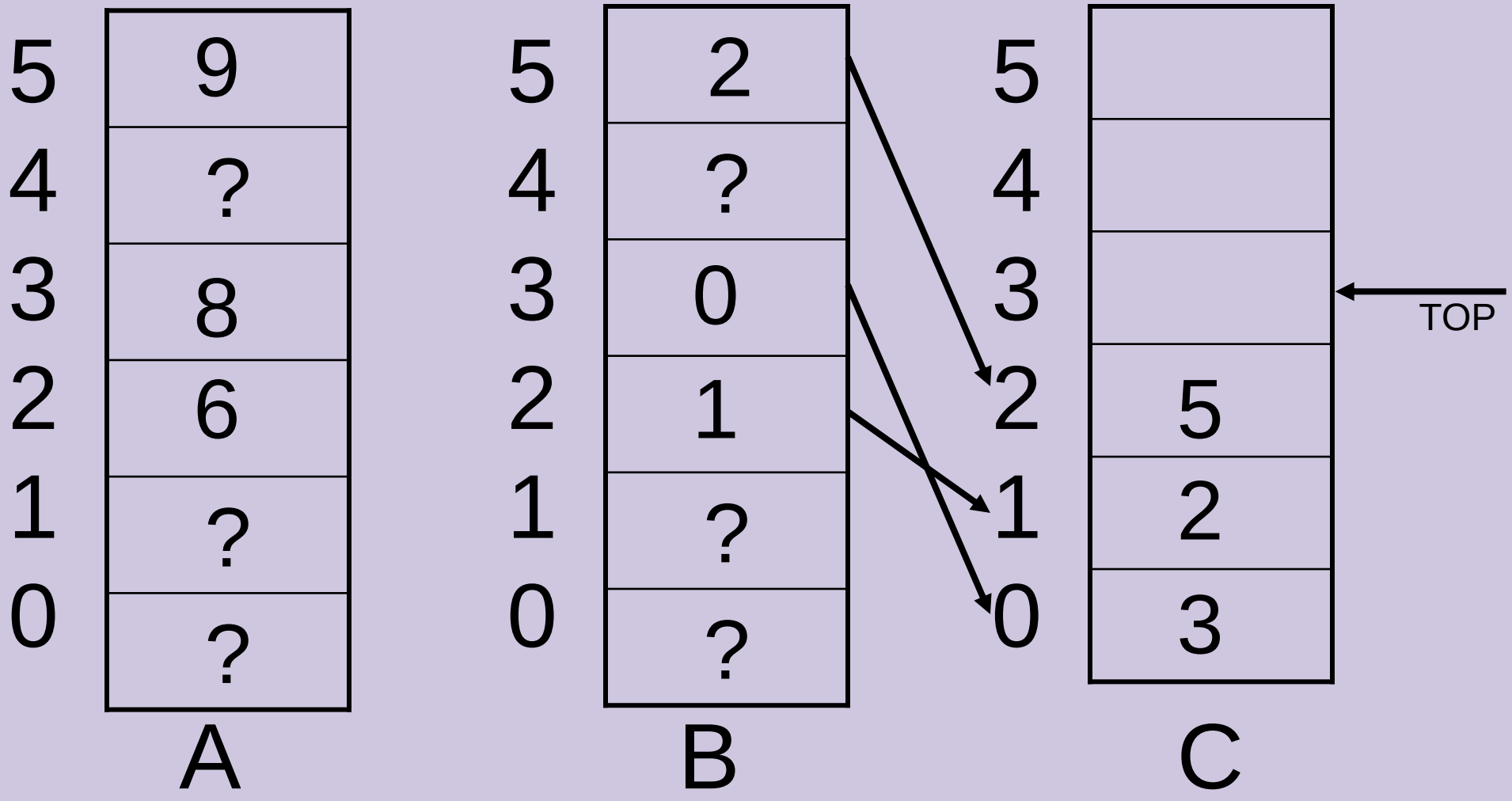
Get(A,4)
?A[4]



4 is in the stack C → B[4] < TOP C[B[4]] = 4

©cs,Technion

Init(A)



©cs,Technion

הקוד לפעולות המערך

```
is_garbage(i)
B[i] >= top | C[B[i]] != i;
```

```
top = 0;
constant = const;
```

אתחל init(V,const):

```
if  (is_garbage(i))
        return  constant;
else
        return A[i];
```

אחזר get(V,i):

```
if  (is_garbage(i)) {
        C[top] = i ;
        B[i] = top ;
        top = top +1;}
A[i]= e;
```

שמור store(V,i,e):

ייצוג מטריצות סימטריות

מטריצה סימטרית היא מערך A דו-ממדי המקיים $A[i,j]=A[j,i]$.

	0	1	2	3	n
0	1				
1	2	3			
2	4	5	6		
3	7	8	9	10	
n	...		...		...

הייצוג: אוסף וקטורים המאוחסנים במערך רציף יחיד:

1 2 3 4 5 6 7 8 9 10

מהי הכתובת של איבר $A[i,j]$? נשתמש בנוסחה: $1+2+\ldots+i = i(i+1)/2$

חצי תחתון

חצי עליון

$$Addr(i,j)=\begin{cases} i(i+1)/2 + j & \text{if } i \geq j \\ Addr(j,i) & \text{Otherwise} \end{cases}$$

ייצוג מטריצות דלילות

מטריצה דלילה היא מערך דו-ממדי עבורו "רוב מוחלט" של האיברים שווה לאפס (או לקבוע c כלשהו).

נשתמש ב- n ו- m

מטריצה n × n

m אברים שונים מ- c

בדוגמא זו n=5 m=7

7 מתוך 25 איברים שונים מהקבוע.

	0	1	2	3	4
0	1		3		
1		5			
2				4	
3					
4	8	2			7

הגדרה אסימפטוטית:

מהו רוב מוחלט? $m=o(n^2)$

דוגמא: מטריצות אלכסוניות.

$n=o(n^2)$

ייצוג: "רשימת" שלשות (i,j,A_{ij}) מסודרות בסדר לקסיקוגרפי.

דוגמא: (0,2,3), (0,5,1), (1,1,5), (2,3,4), (4,1,2), (4,4,7), (4,5,8)

n=10

n x n	1	2	3	4	5	6	7	8	9	10
1										
2										
3		12			34					45
4										
5							0			
6		23			45					
7								23		
8										
9		35			78					56
10										

$m << n^2$

מספר האברים m

	1	2	3	4	5	6	7	8	9	10
1										
2										
3		12			34					45
4										
5										
6		23			45					
7								23		
8										
9		35			78					56
10										

3	2	12
3	5	34
3	10	45
6	2	23
6	5	45
7	8	23
9	2	35
9	5	78
9	10	56

$3m = O(m) \quad << O(n^2)$

סדר לקסיגורפי

3	2	12
3	5	34
3	10	45
6	2	23
6	5	45
7	8	23
9	2	35
9	5	78
9	10	56

שורה **עמודה**

גישה לאבר

	1	2	3	4	5	6	7	8	9	10
1										
2										
3		12			34					45
4										
5										
6		23			45					
7								23		
8										
9		35			78					56
10										

A[6,2]

גישה לאבר O(1)

3	2	12
3	5	34
3	10	45
6	2	23
6	5	45
7	8	23
9	2	35
9	5	78
9	10	56

גישה לאבר
בסדר לקסיגורפי
A[6,2]

גישה לאבר
O(log(m))

	1	2	3	4	5	6	7	8	9	10
1										
2										
3		12			34					45
4										
5										
6		23			45					
7								23		
8										
9		35			78					56
10										

חיבור מטריצות

$$C_{i,j}=A_{i,j}+B_{i,j}$$

חיבור מטריצות ב- $O(n^2)$

חיבור מטריצה ב- O(m)

3	2	12
3	5	34
3	10	45
6	2	23
6	5	45
7	8	23
9	2	35
9	5	78
9	10	56

3	2	42
3	6	4
3	10	47
6	2	53
7	5	15
7	8	3
9	2	11
9	5	22
10	10	46

3	2	54
3	5	34
3	6	4
3	10	92

סיבוכיות

$2m=O(m)$

תוכנית לחיבור מטריצות C=A+B

```
typedef struct NODE {
        int row, col;
        float  val;
}
NODE  A[mA+1], B[mB+1],  C[mA+mB+1];
```

```
int lex(NODE a, NODE b);
/* lexicographically compares a and b;
Returns:   -1  if (a.row,a.col) < (b.row,b.col)
            0  if (a.row,a.col) = (b.row,b.col)
            1  if (a.row,a.col) > (b.row,b.col)  */
```

```
void ADD(NODE *A, NODE *B, NODE *C, int *mC){
        int i=0, j=0, *mc=0;
        A[mA]=B[mB]={+∞,+∞, 0};
        while (i < mA || j < mB){
                switch lex(A[i],B[j]){
                case –1:   C[*mC++] = A[i++]; break;
                case  1:   C[*mC++] = B[j++]; break;
                case  0:   C[*mC].row = A[i].row;
                           C[*mC].col  = A[i].col;
                           C[*mC].val  = A[i++].val + B[j++].val;
                           if (C[*mC].val ≠ 0) *mC++;
}}}
```

©cs,Technion

	מערך דו מימדי	מערך שלשות
סבוכיות מקום	$O(n^2)$	$O(m)$
גישה לאבר	$O(1)$	$O(\log m)$
סריקת שורה	$O(n)$	$O(\min(n,m))$
סריקת עמודה	$O(n)$	$O(m)$
חיבור	$O(n^2)$	$O(m)$
מכפלה	$O(n^3)$	$O(mn)$

©cs,Technion

חסרונות ויתרונות של הייצוג

הייצוג: רשימת שלשות (i,j,A_{ij}) מסודרות בסדר לקסיקוגרפי.

חסרון עקרי: אין גישה אקראית לפי מציין בזמן $O(1)$.

יתרונות: 1. חוסך בזיכרון עבור מטריצות דלילות.

2. מאיץ פעולות חיבור וכפל של מטריצות דלילות.

חיבור מטריצות בגודל $n \times n$ לוקח בייצוג רגיל זמן $O(n^2)$.

נניח כעת שיש m איברים שונים מהקבוע c במטריצה אחת ו- k בשניה. חיבור שתי המטריצות נעשה ע"י מיזוג שתי הרשימות המייצגות את המטריצות הנ"ל. זמן המיזוג הוא אורכן הכולל של הרשימות כלומר $O(m+k)$. כלומר עבור מטריצות ריבועיות דלילות זמן החיבור הוא $o(n^2)$.

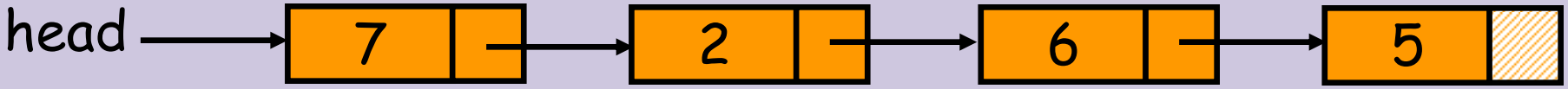
תרגיל ממבחן: הראו שכפל מטריצות ריבועיות דלילות בייצוג זה לוקח זמן $o(n^3)$ במקום $O(n^3)$. לשם כך נדרש מיון לפי עמודות של מטריצה B.

רשימות מקושרות

©cs,Technion

שמוש ברשימות מקושרות

נזכר כעת כיצד מתבצע חיפוש, הכנסה, והוצאה (find, insert, delete) ברשימות מקושרות ונגדיר וריאציות עליהן.



יתרונות בהשוואה למערך:

1. מאפשר הקצאת זיכרון דינמית.
2. אין צורך להקצות מקום זיכרון רצוף.
3. ניתן להוציא איבר מתוך רשימה מקושרת בלי להשאיר "חור" כמו במערך ולכן ניתן לסרוק את כל האיברים בזמן ליניארי.

חסרונות בהשוואה למערך:

1. אין גישה לפי אינדקס בזמן $O(1)$.

©cs,Technion

חיפוש ברשימה מקושרת

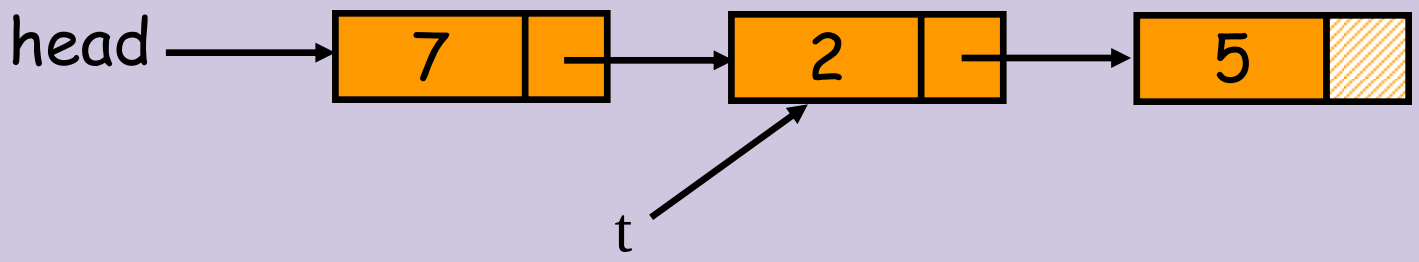
```
void  init (NODE  *head){
        (* head) = NULL;
}
```

פעולת איתחול
:init(head)

```
NODE * find (DATA_TYPE x, NODE  *head){
        NODE *t;
    for (t = head; t != NULL && t -> info != x;
                        t = t -> next );
    return t;
}
```

פעולת חיפוש
:find(x,head)

חיפוש איבר בזמן $O(n)$ במקרה הגרוע.



הכנסת איבר לרשימה מקושרת

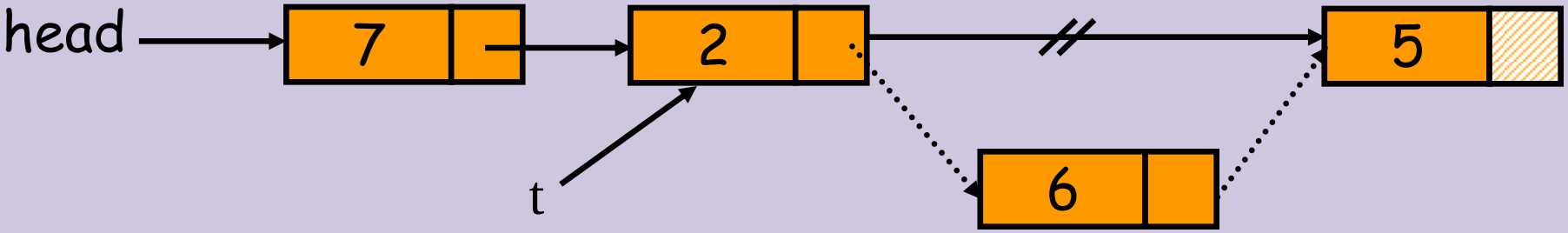
```
int  insert ( NODE  *t,  DATA_TYPE x){
        NODE *p;
    if (t == NULL) return 0;
    p = (NODE *) malloc (sizeof (NODE));
    p -> info = x;
    p -> next = t -> next ;
    t -> next  = p ;
    return 1;
}
```

פעולת insert(t,x):

הפרמטר t מצביע לצומת שאחריו מוסיפים צומת חדש.

הכנסת איבר בזמן $O(1)$ כאשר ידוע מקום ההכנסה.

דוגמא: insert(t,6).



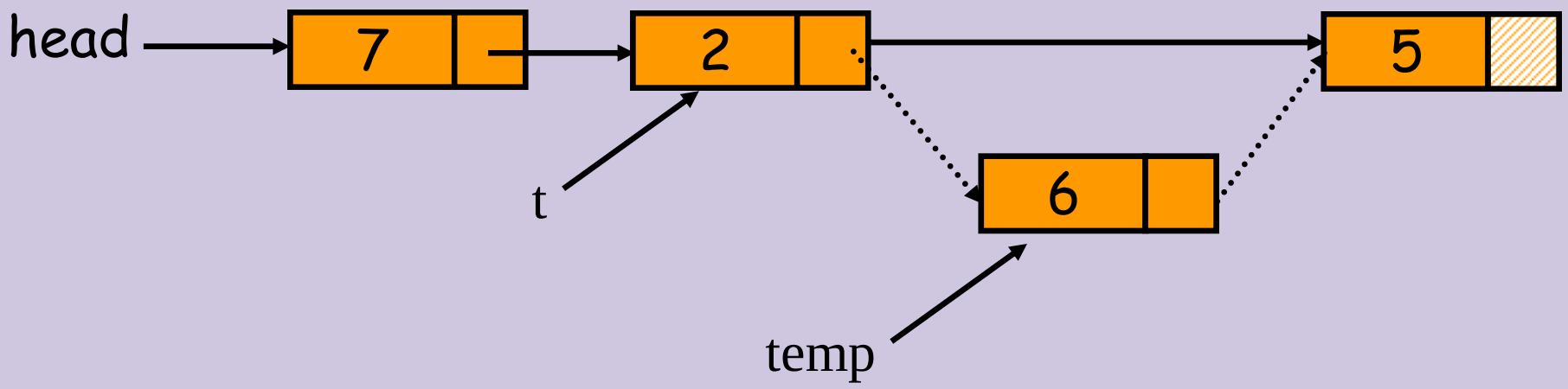
הוצאת איבר מרשימה מקושרת

```
delete ( NODE  *t){
        NODE  * temp ;
        temp = t → next;  /* מצביע לצומת שמורידים */
        t → next = temp → next ;
        free (temp) ;
}
```

פעולת delete(t):

הפרמטר t מצביע לצומת שלפני הצומת שמוציאים.

הוצאת איבר בזמן $O(1)$ כאשר ידוע מקום ההוצאה.

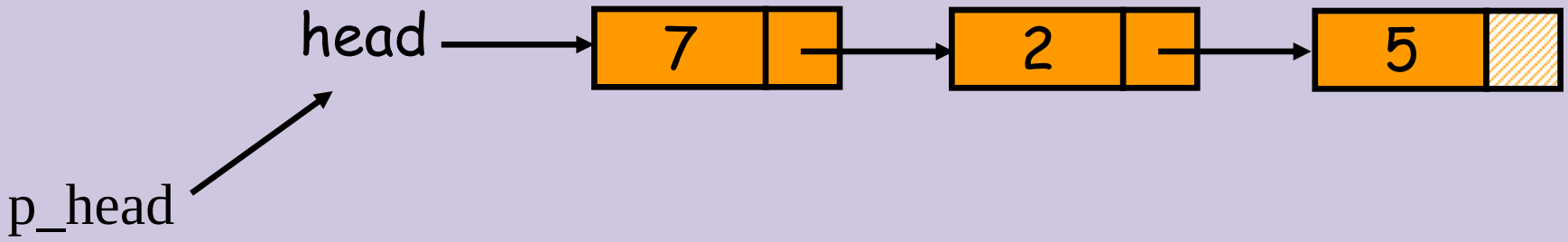


©cs,Technion

הוצאת איבר מראש רשימה מקושרת

```
int delete_first ( NODE  **p_head){
        NODE  * temp ;
        if (*p_head = = NULL)  return 0 ;
        temp = *p_head;
        *p_head = temp → next ;
        free (temp) ;
        return 1;          /* success code */
}
```

פעולת delete_first(r):



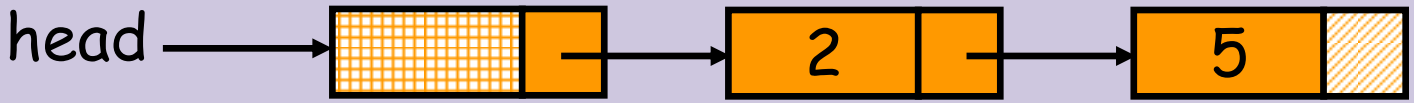
תכנות זה מְסֻרְבָּל. מהו הפתרון לכך ?

רשימות עם כותרת

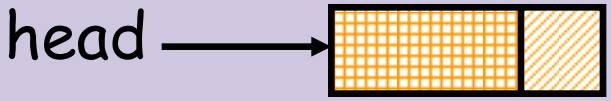
מוסיפים איבר "ריק" בתחילת הרשימה. תוספת זו חוסכת:

- טיפול מיוחד ברשימות ריקות
- טיפול מיוחד בזמן הכנסה לפני הצומת הראשון.

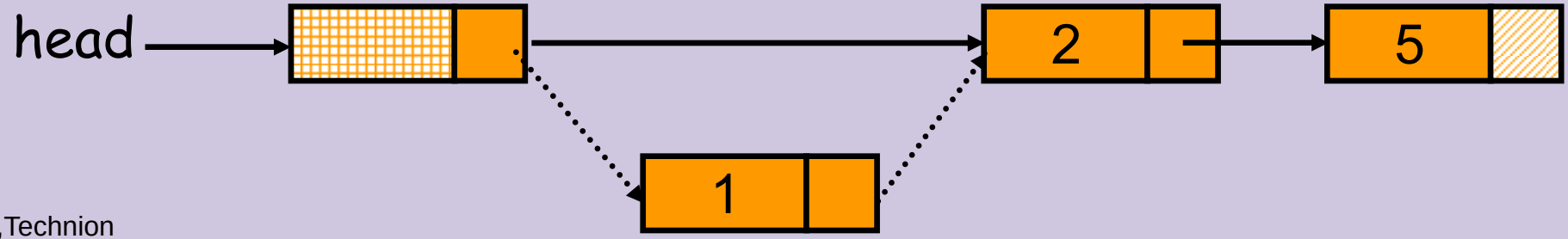
בייצוג זה רשימה בת שתי איברים ממומשת כך:



ורשימה ריקה ממומשת כך:

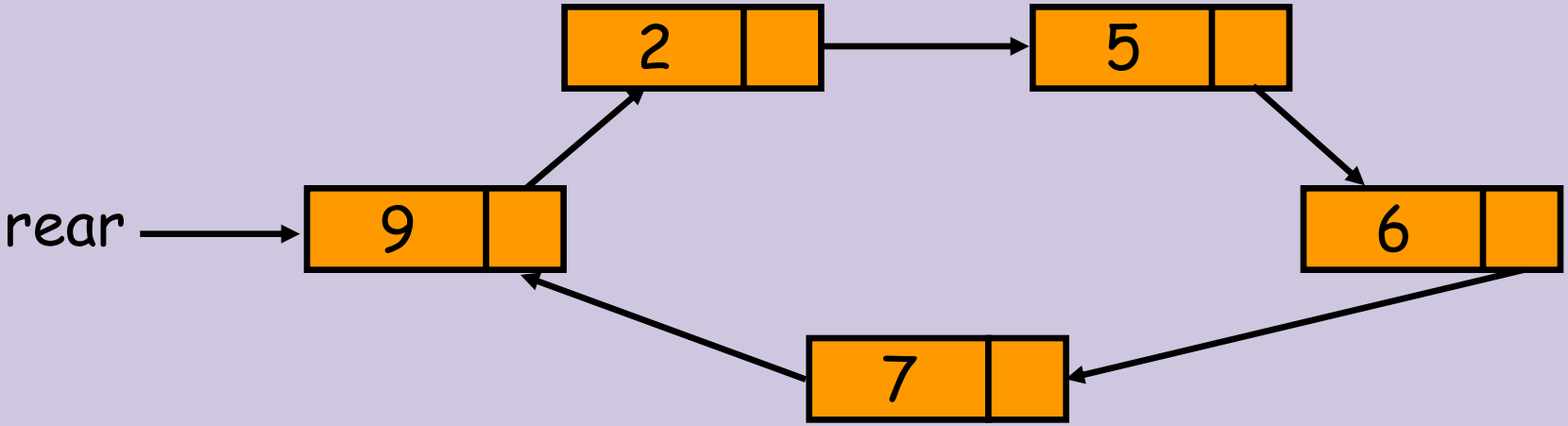


הכנסה לפני איבר ראשון זהה להכנסה לפני איבר כלשהו:



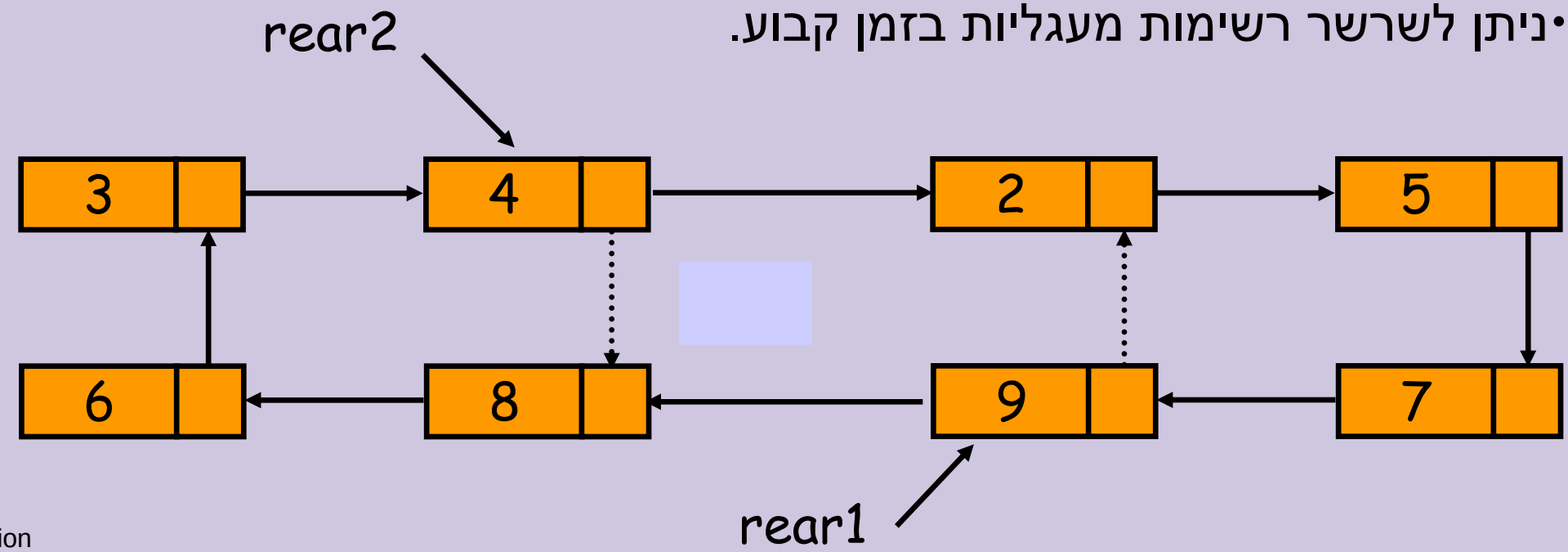
©cs,Technion

רשימות מעגליות

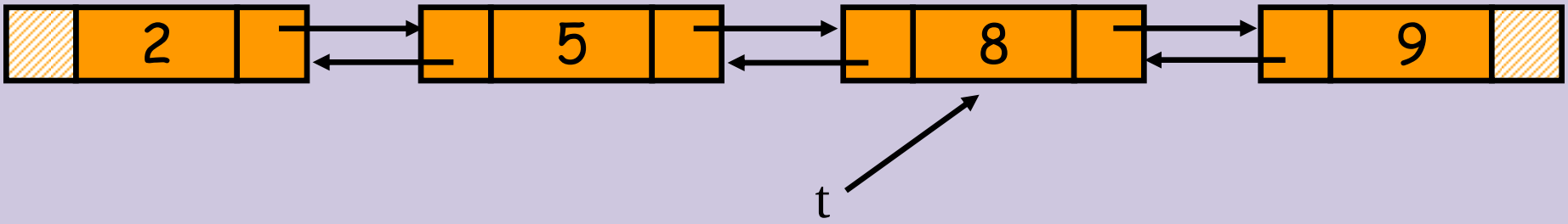


יתרונות:

- אפשר להגיע מכל איבר לכל איבר
- ניתן לשרשר רשימות מעגליות בזמן קבוע.

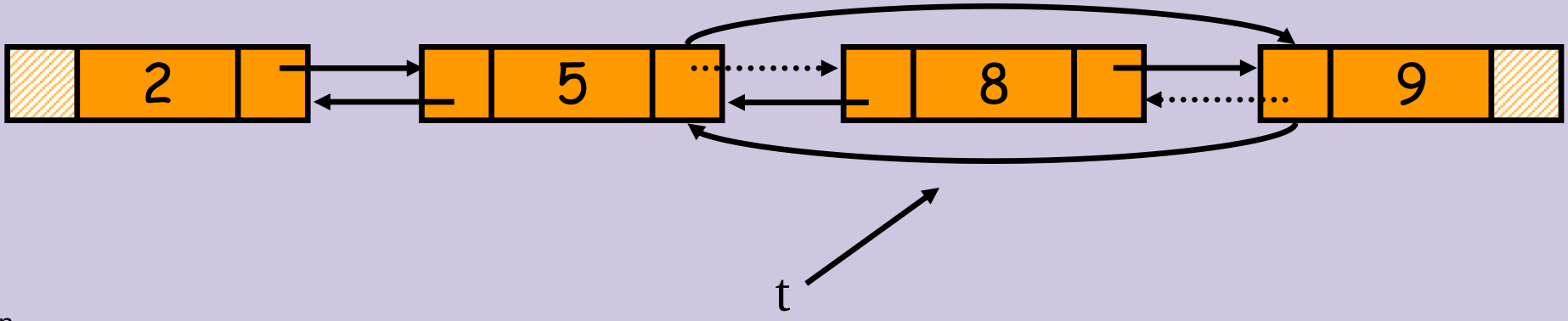


רשימה מקושרת דו-כיוונית



יתרון: מאפשר להוציא איבר בהינתן מצביע t אליו (ולא רק את האיבר שאחריו).

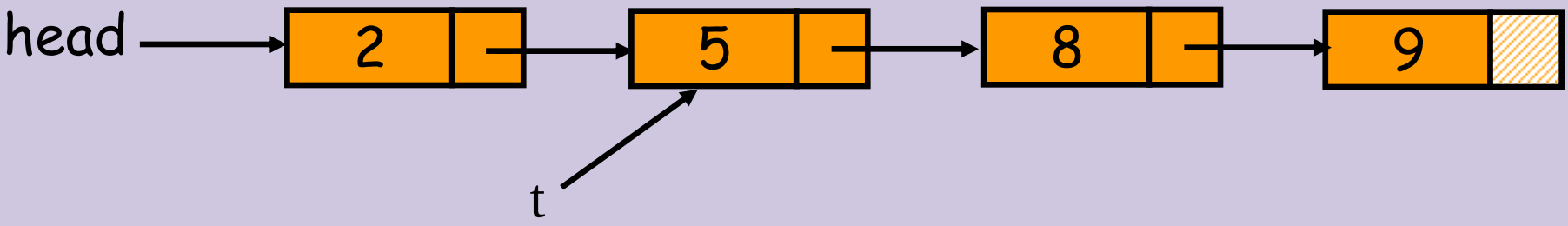
```
t → next → prev = t → prev ;
t → prev → next = t → next ;
```



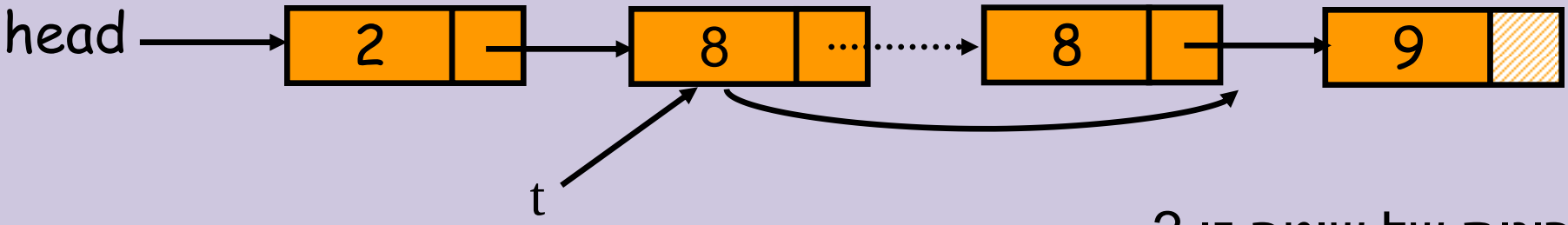
©cs,Technion

הוצאה "טריקית"

האם ניתן להוציא איבר אליו מצביע t גם מרשימה חד-כוונית ?



פשוט נעתיק את האינפורמציה בתא העוקב ל- t לתא ש- t מצביע אליו ונוציא את התא העוקב.



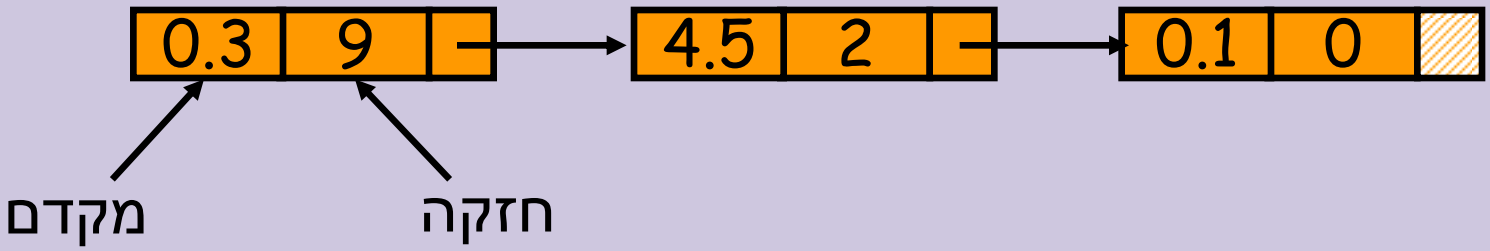
מהן החסרונות של שיטה זו ?

- צומת יכול להכיל הרבה אינפורמציה ולכן העתקה עלולה להיות פעולה ארוכה.
- יתכן וישנם מצביעים נוספים לתא שהוצא ואין אפשרות לעדכן מצביעים אלה.

©cs,Technion

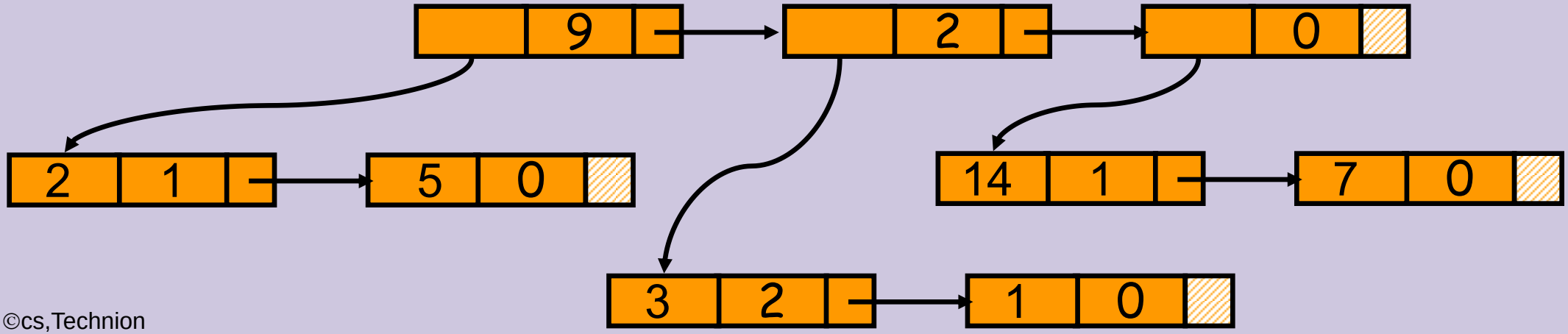
ייצוג פולינומים

נעלם אחד. דוגמא: $0.3x^9 + 4.5x^2 + 0.1$



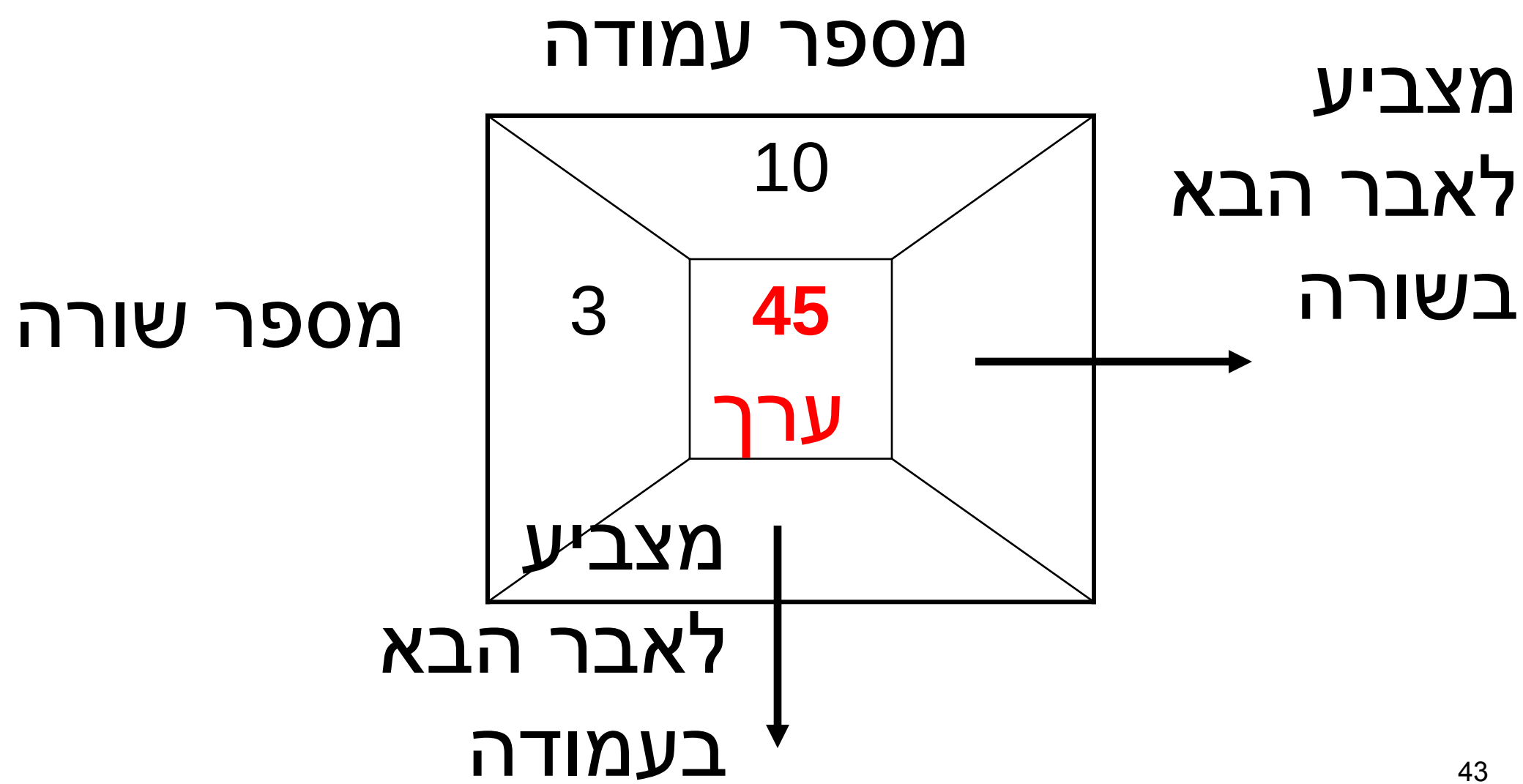
מימוש זה יעיל כאשר ההפרש בין החזקה הגבוהה והנמוכה גדול וכן הרבה מקדמים של חזקות הביניים מתאפסים.

שני נעלמים. דוגמא: $(2y+5)x^9 + (3y^2+1)x^2 + (14y+7)$



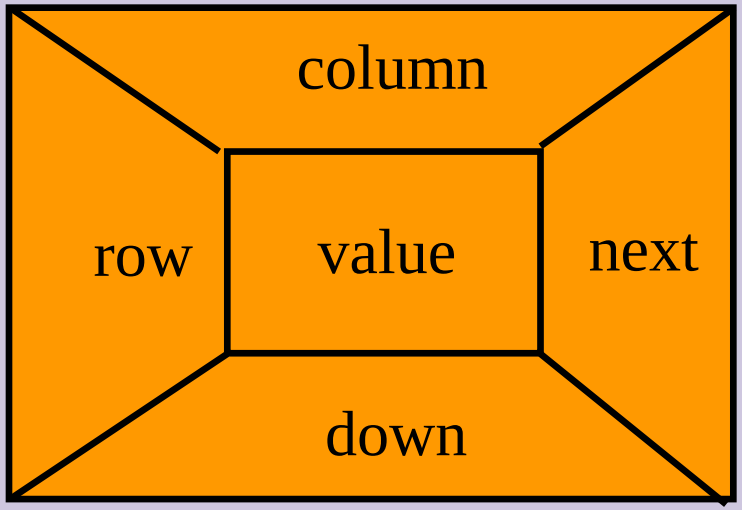
©cs,Technion

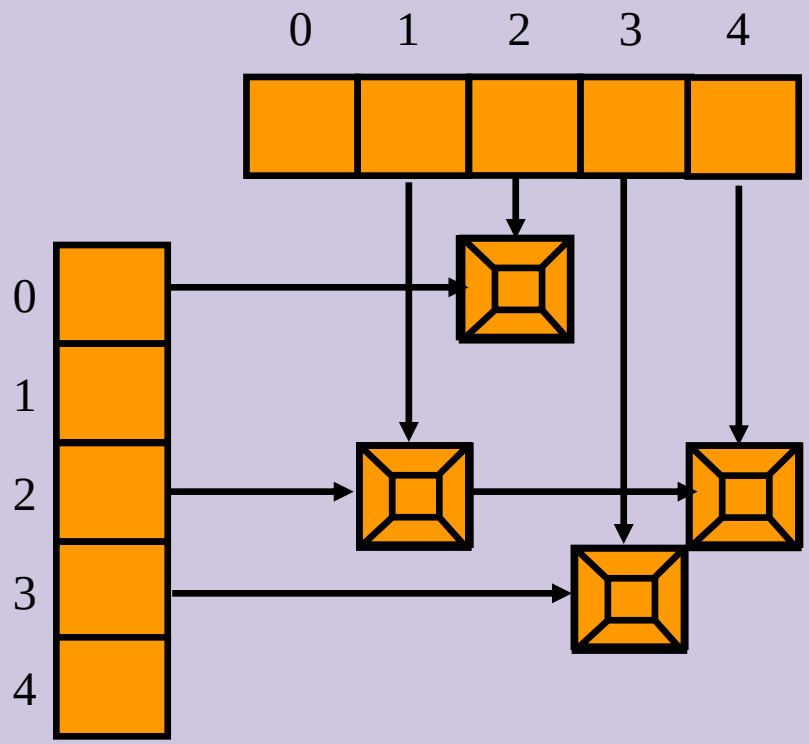
מטריצה דלילה מימוש 2



ייצוג מטריצות דלילות

מבנה צומת:

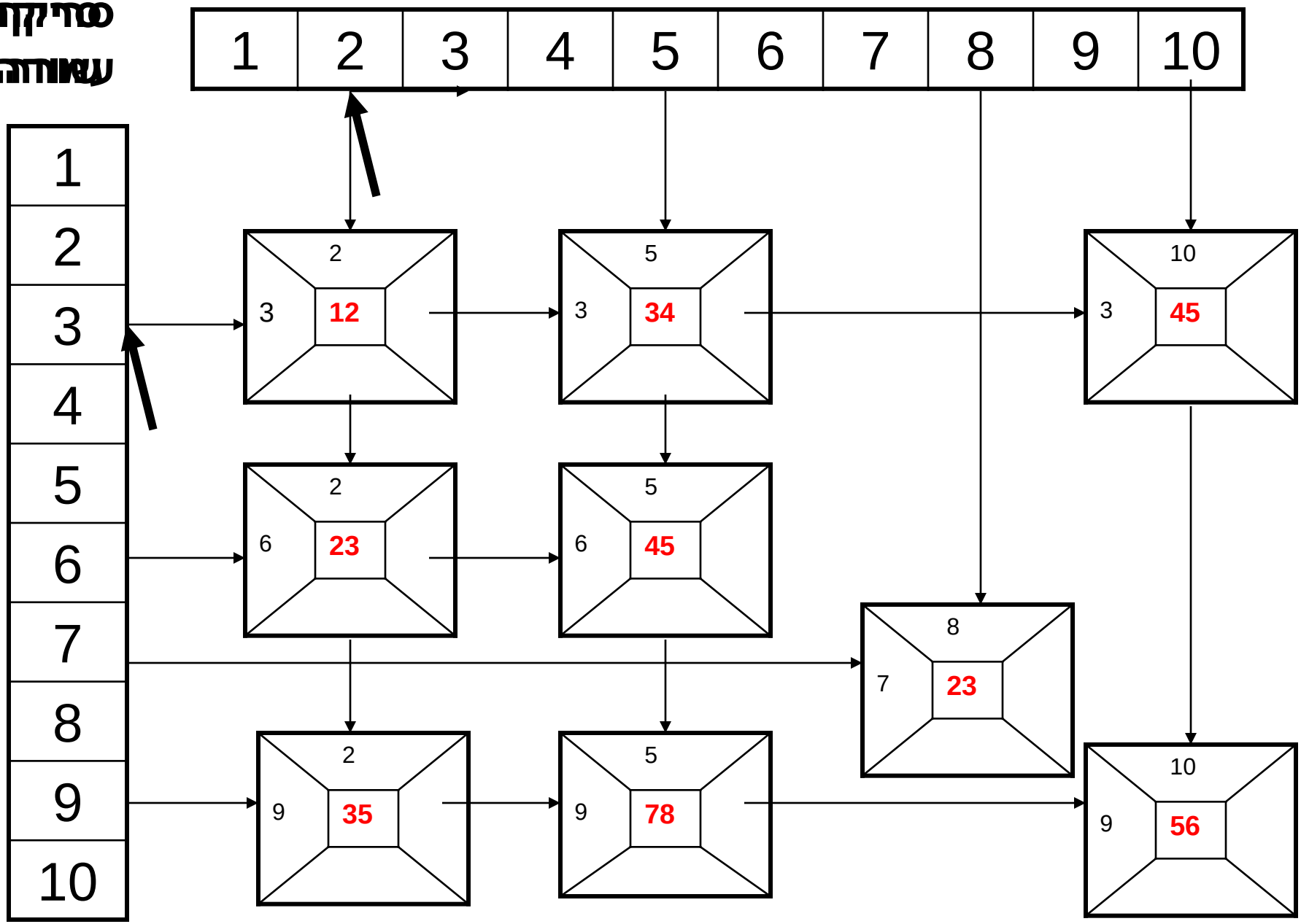





```
typedef struct node {
        float value ;
        struct node*next, *down ;
        int row, column ;
} NODE ;
```

©cs,Technion

סדר יקדון
עשרה 23



כפל מטריצות דלילות

```
#define N_row 20
#define N_col 20
NODE *row_header[N_row],
        *col_header[N_col];
/* Compute the value of c[i][j]  */
double mult_row_col(int i, int j){
double c =  0 ;
NODE *r = row_header[i],
        *k = col_header[j];

          while (r != NULL && k != NULL){
          if (r → column < k → row)   r = r → next ;
              else (r → column > k → row)   k = k → down ;
                      else  /* r->column ==k->row */
                      c + =  r → value  *  k → value ;
                       r = r → next ; k = k → down ;  }
          return c;
}
```

כפל מטריצות C = AB. התוכנית מחשבת את האיבר C_{ij}.

	מערך דו מימדי	מערך שלשות	רשימה מקושרת
סבוכיות מקום	$O(n^2)$	$O(m)$	$O(n+m)$
גישה לאבר	$O(1)$	$O(\log m)$	$O(\min(n,m))$
סריקת שורה	$O(n)$	$O(\min(n,m))$	$O(\min(n,m))$
סריקת עמודה	$O(n)$	$O(m)$	$O(\min(n,m))$
חיבור	$O(n^2)$	$O(m)$	$O(n+m)$
מכפלה	$O(n^3)$	$O(mn)$	$O(mn+n^2)$

©cs,Technion