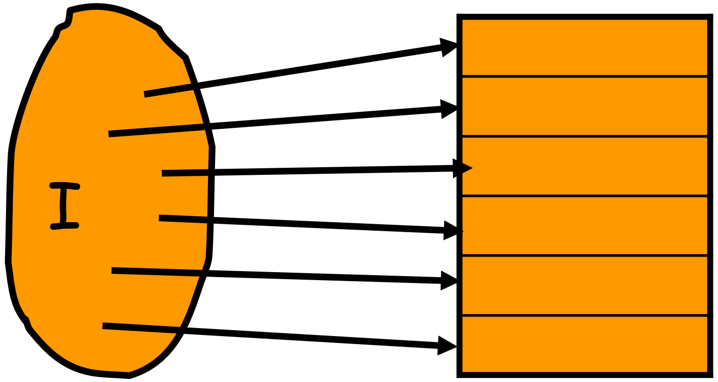
מערכים, מטריצות דלילות, ורשימות מקושרות

חומר קריאה לשיעור זה

Chapter 11.2- Linked lists (204 - 213)

©Geiger & Itai, 2001

מערך כמבנה נתונים



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

אחזר (get) $A[i]$

שמור (store) $A[i] = e$

מערך מוגדר ע"י הפעולות הבאות:

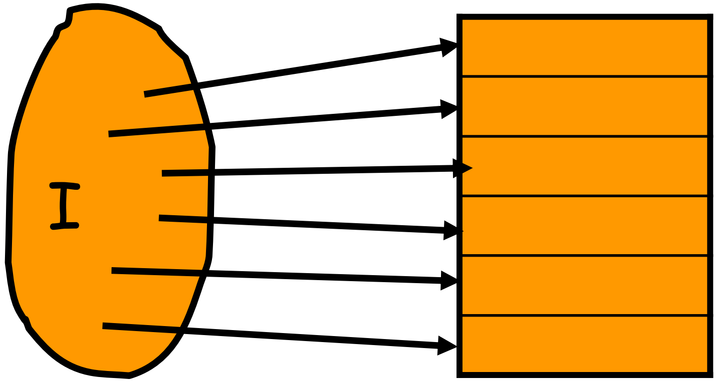
create(type,I) - מחזיר מערך A של איברים מטיפוס type כאשר האינדקסים הם הקבוצה הסופית I.

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.

©cs,Technion

מערך כמבנה נתונים



בשפת c האינדקסים הם הקבוצה I={0,...,n-1}

אחזור (get) A[i]

שמור (store) A[i] = e

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 ∈ i בתוך A.

store(A,i,e) - מאחסן במערך A, תחת אינדקס I ∈ i, את ערך הביטוי e.

כללים הנשמרים ע"י הפעולות:

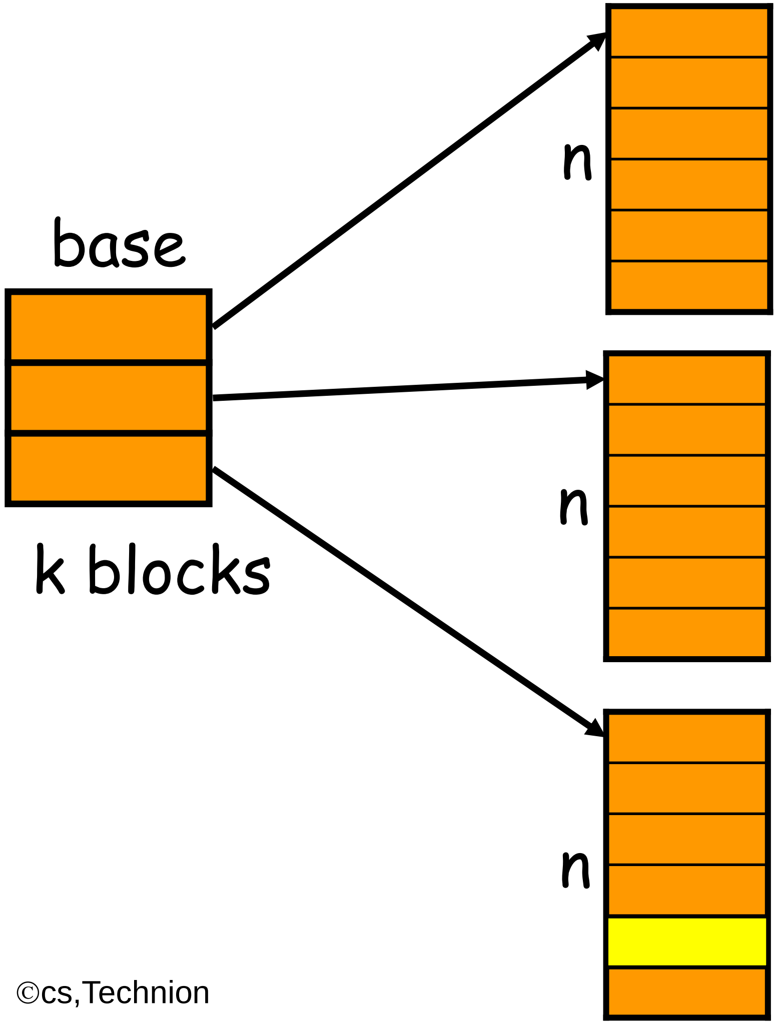
- כל הערכים במערך מאותו טיפוס, והוא נקבע בהכרזה (type).
- הערך המאוחזר לפי אינדקס i , הוא הערך האחרון שנשמר לפי אינדקס i.
- אחזור לפי אינדקס i מחזיר ערך בלתי מוגדר אם מעולם לא אוחסן ערך לפי אינדקס זה.

©cs,Technion

מימושים למערך

• אזור זיכרון רצוף. כל הפעולות מתבצעות בזמן $O(1)$.

• מספר אזורים בזיכרון שלכל אחד גודל קבוע n.



כדי לממש מערך בגודל m יש לבחור $k=\lceil m/n \rceil$.
האיבר $A[i]$ נמצא בכתובת $base[\lfloor i/n \rfloor]+ i\%n$.

לדוגמא כאשר $n=6, i=16$ אז הכתובת של $A[16]$ מחושבת ע"י

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

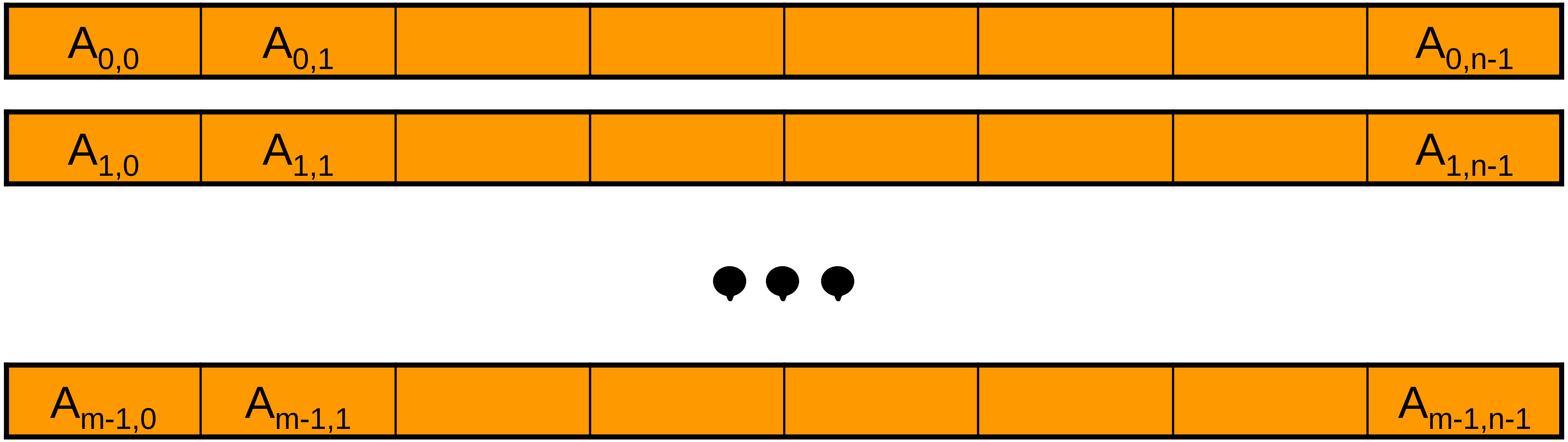
זמן חישוב הכתובת $O(1)$.

©cs,Technion

מערך דו-ממדי

int a[m][n]

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



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

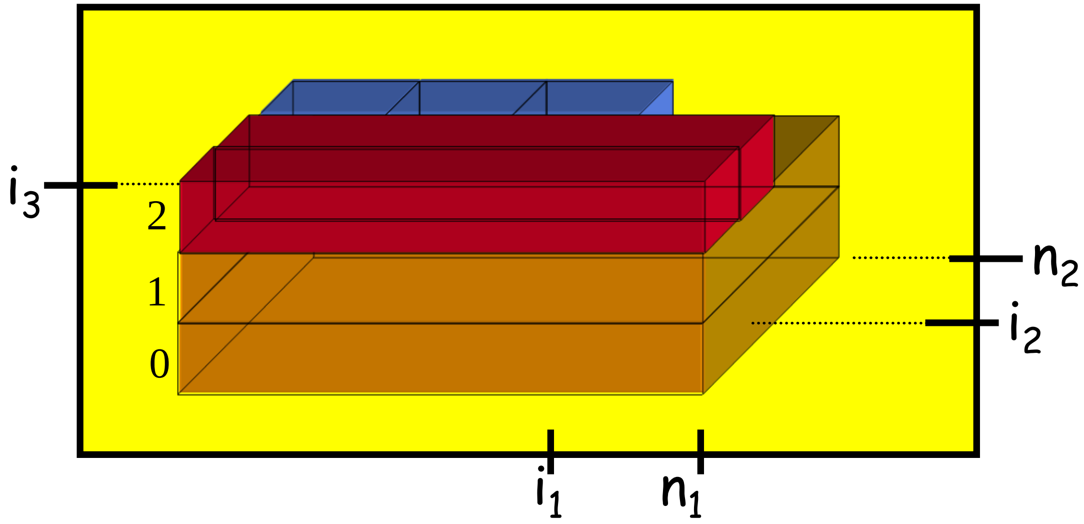
הכתובת של שורה $A[i]$ היא $base + i \cdot n$

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

מערך רב-ממדי

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

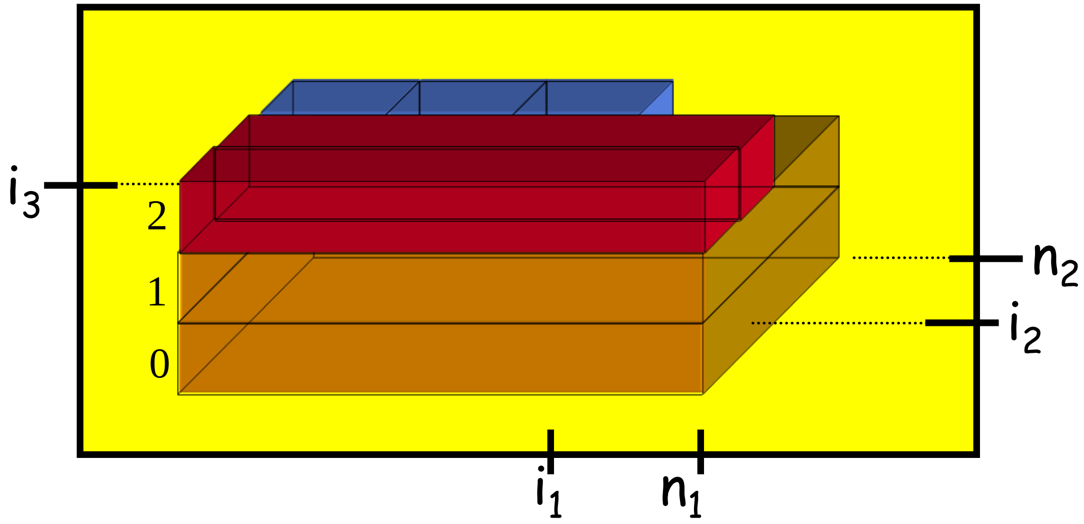
במערך תלת-ממדי $A[n_3][n_2][n_1]$ int הכתובת של איבר $A[i_3][i_2][i_1]$ מחושבת ע"י הנוסחה: $base + (i_3 n_2 + i_2) n_1 + i_1 = base + i_3 \cdot n_2 \cdot n_1 + i_2 \cdot n_1 + i_1$



מערך רב-ממדי

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: $base + i_3 \cdot n_2 \cdot n_1 + i_2 \cdot n_1 + i_1 = base + (i_3 n_2 + i_2) n_1 + i_1$



במערך רב-ממדי $int\ A[n_d]...[n_3][n_2][n_1]$ הכתובת של $A[i_d]...[i_3][i_2][i_1]$ מחושבת ע"י הנוסחה: $base + i_d \cdot n_{d-1} \cdots n_1 + i_{d-1} n_{d-2} \cdots n_1 + ... + i_1$

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

מערך רב-ממדי (המשך)

הנוסחה לחישוב הכתובת במערך רב ממדי דומה *לכלל הורנר* 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;
```

סריקת מערך רב-ממדי

זמן סריקת כל איברי מערך $A[n_d]...[n_3][n_2][n_1]$ תלוי בסדר הסריקה.

סריקה לפי סדר האחסון מהירה יותר בגלל לוקליות של גישה לזיכרון. במימוש שהצגנו, סריקה כאשר האינדקסים הנמוכים נסרקים ראשונים מהירה יותר מסריקה בה האינדקסים הגבוהים נסרקים ראשונים. השיפור הוא בקבוע בלבד ואינו מתבטא בסימון O.

איתחול מערך בזמן O(1)

הבעיה: בייצוג הרגיל אתחול מערך דורש זמן $O(n)$. נראה ייצוג שיאפשר לאתחל מערך בזמן $O(1)$. בתמורה נשתמש ביותר זיכרון.

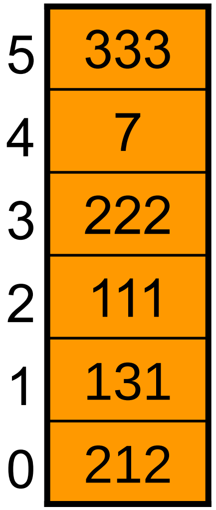
איתחול מערך בזמן O(1)

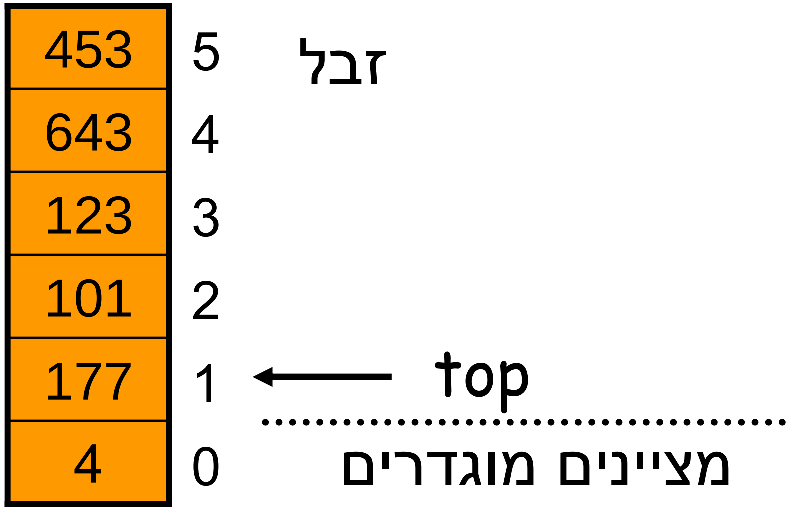
<u>הבעיה</u>: בייצוג הרגיל אתחול מערך דורש זמן $O(n)$. נראה ייצוג שיאפשר לאתחל מערך בזמן $O(1)$. בתמורה נשתמש ביותר זיכרון.

5	333
4	7
3	222
2	111
1	131
0	212

איתחול מערך בזמן O(1)

הבעיה: בייצוג הרגיל אתחול מערך דורש זמן $O(n)$. נראה ייצוג שיאפשר לאתחל מערך בזמן $O(1)$. בתמורה נשתמש ביותר זיכרון.

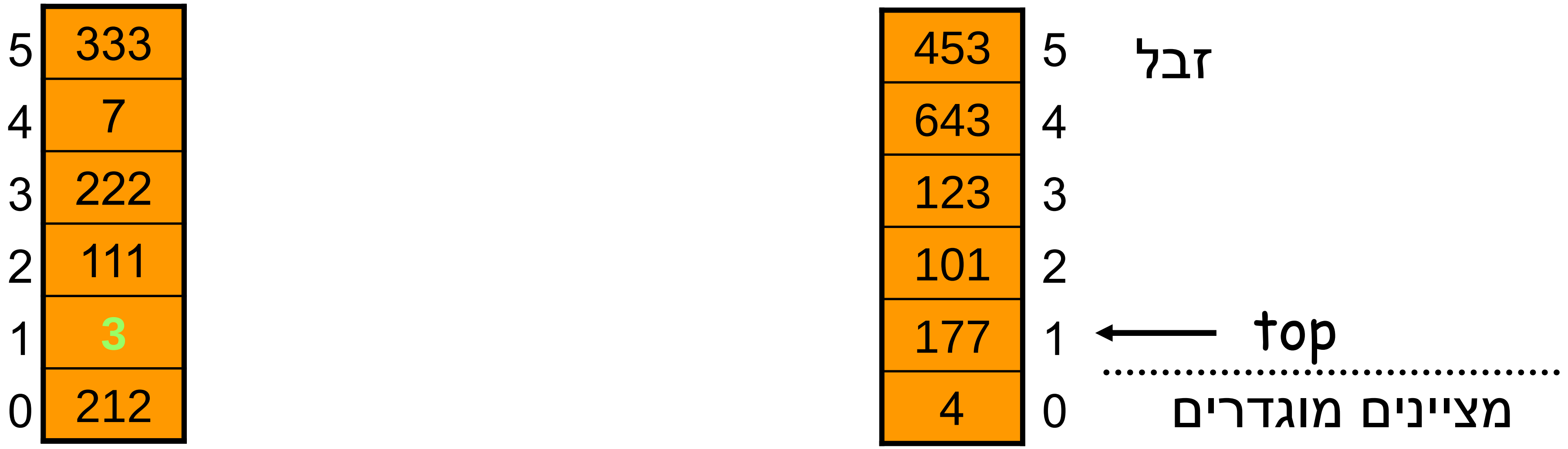




©cs,Technion

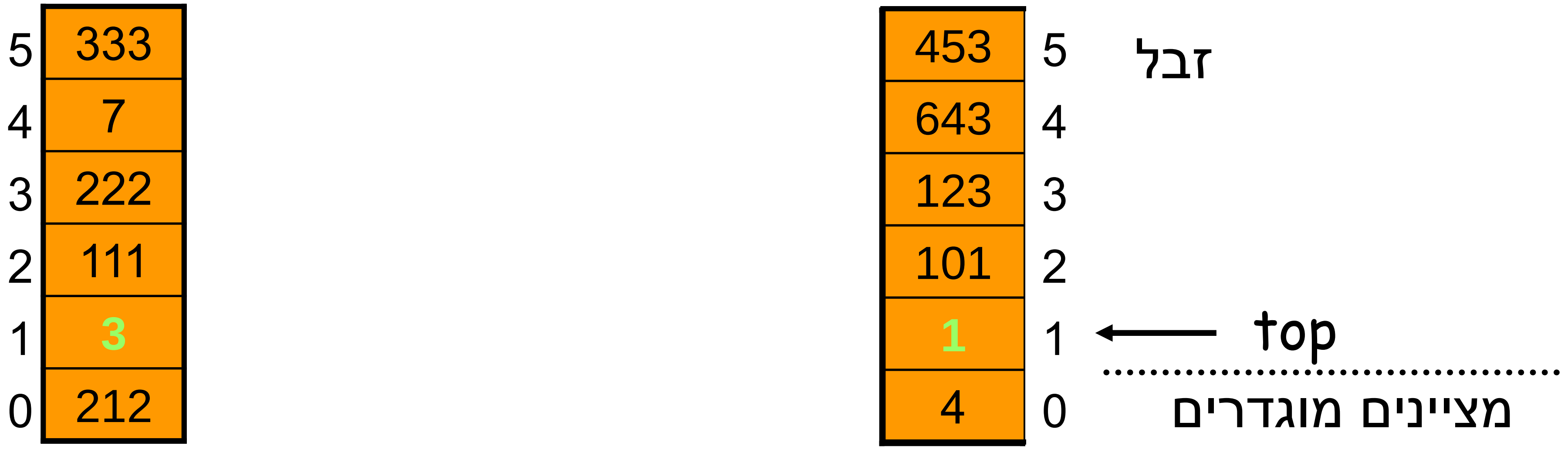
איתחול מערך בזמן O(1)

הבעיה: בייצוג הרגיל אתחול מערך דורש זמן $O(n)$. נראה ייצוג שיאפשר לאתחל מערך בזמן $O(1)$. בתמורה נשתמש ביותר זיכרון.



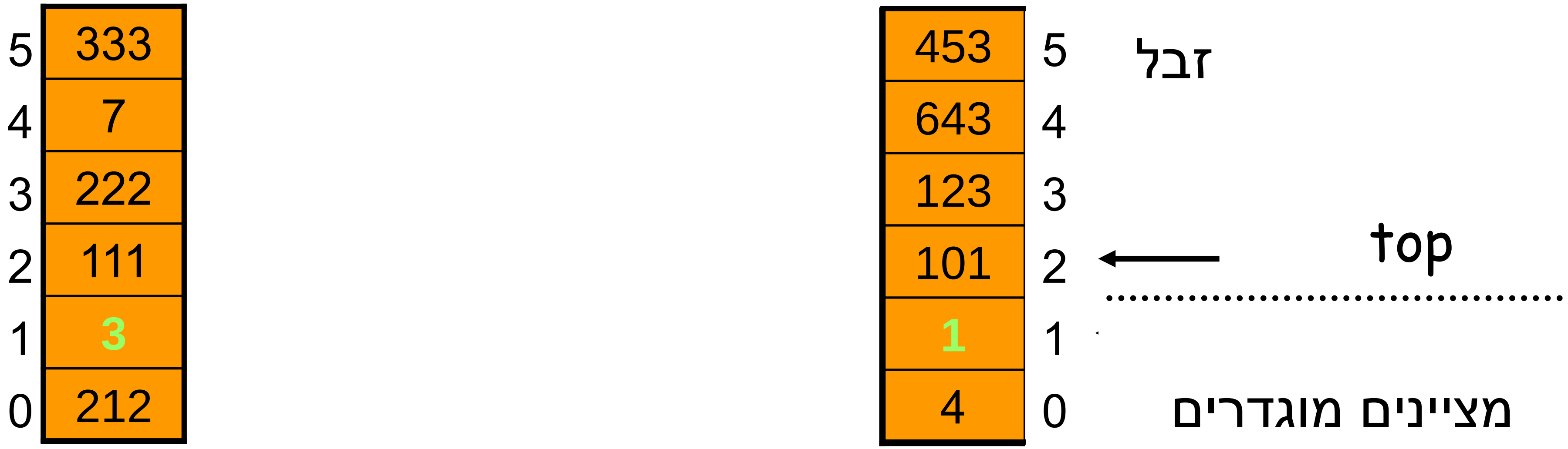
איתחול מערך בזמן O(1)

הבעיה: בייצוג הרגיל אתחול מערך דורש זמן $O(n)$. נראה ייצוג שיאפשר לאתחל מערך בזמן $O(1)$. בתמורה נשתמש ביותר זיכרון.



איתחול מערך בזמן O(1)

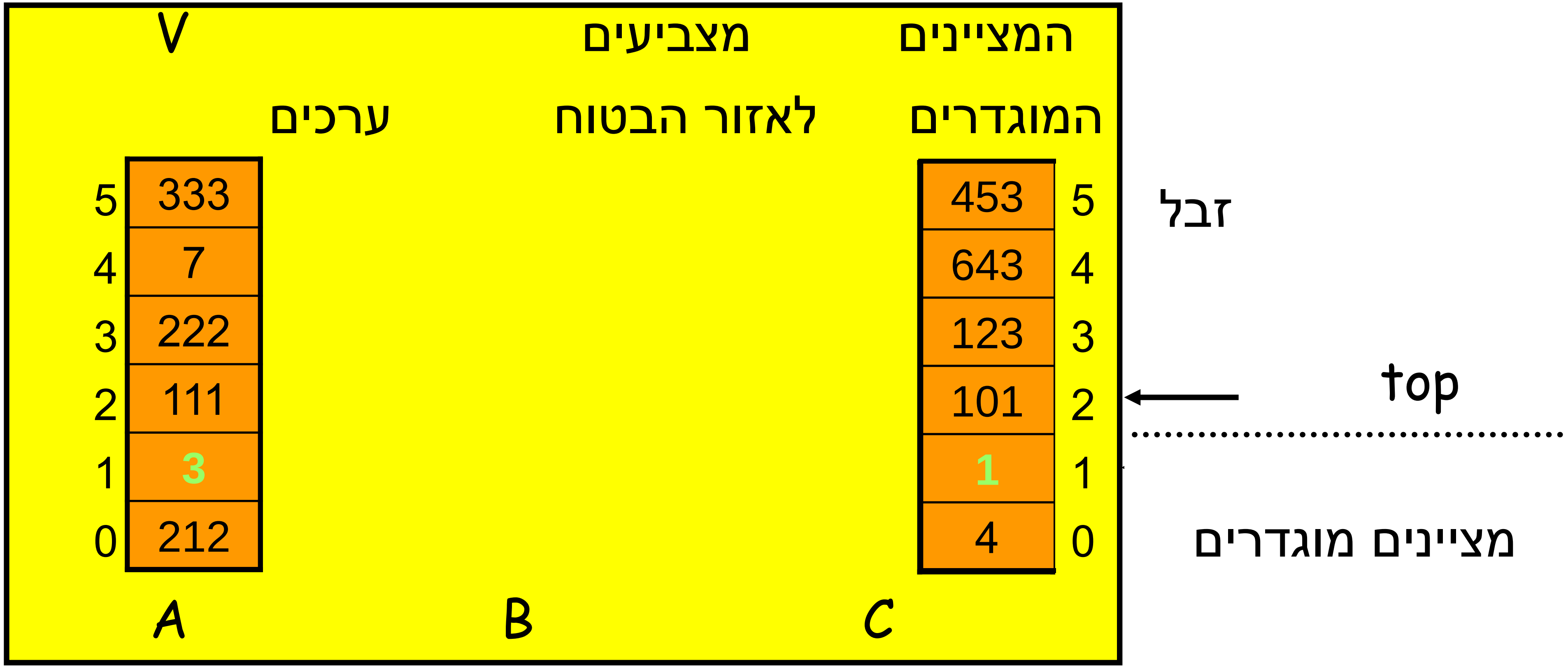
<u>הבעיה</u>: בייצוג הרגיל אתחול מערך דורש זמן $O(n)$. נראה ייצוג שיאפשר לאתחל מערך בזמן $O(1)$. בתמורה נשתמש ביותר זיכרון.



©cs,Technion

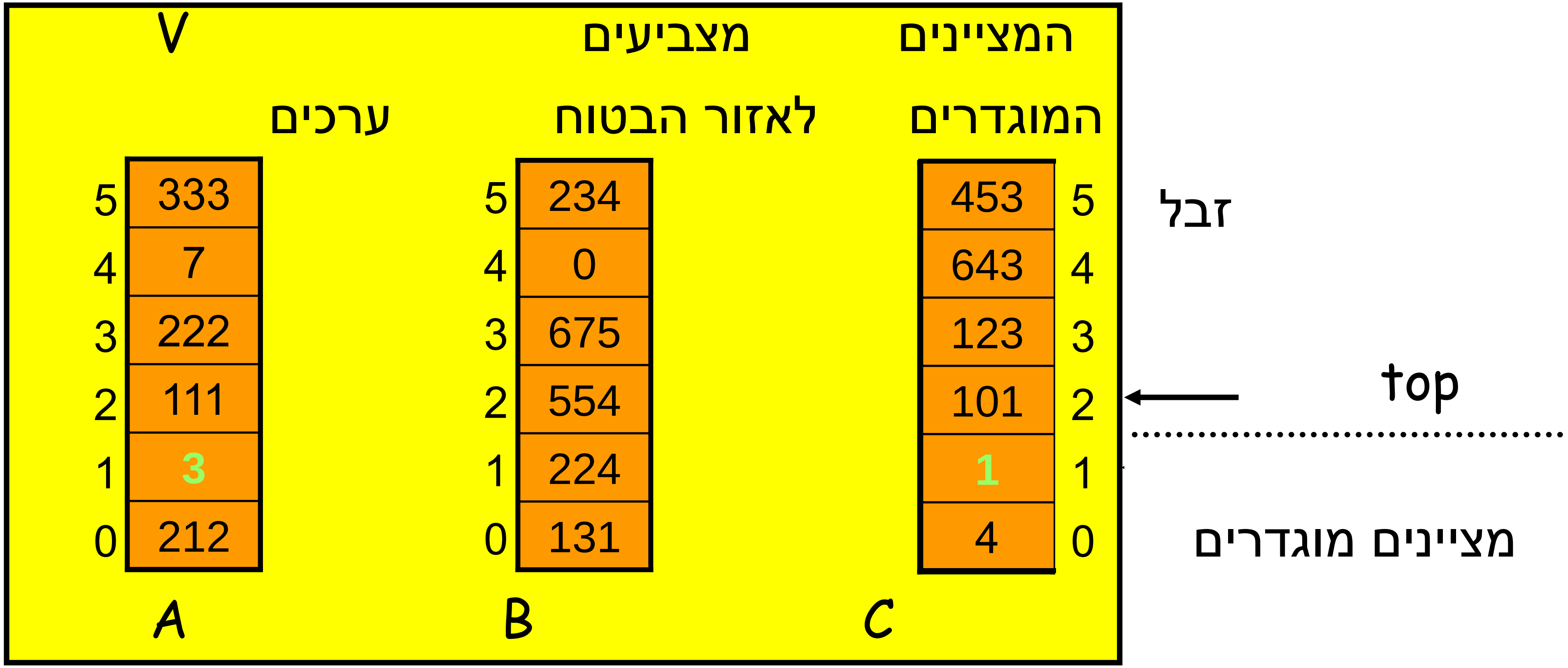
איתחול מערך בזמן O(1)

<u>הבעיה</u>: בייצוג הרגיל אתחול מערך דורש זמן $O(n)$. נראה ייצוג שיאפשר לאתחל מערך בזמן $O(1)$. בתמורה נשתמש ביותר זיכרון.



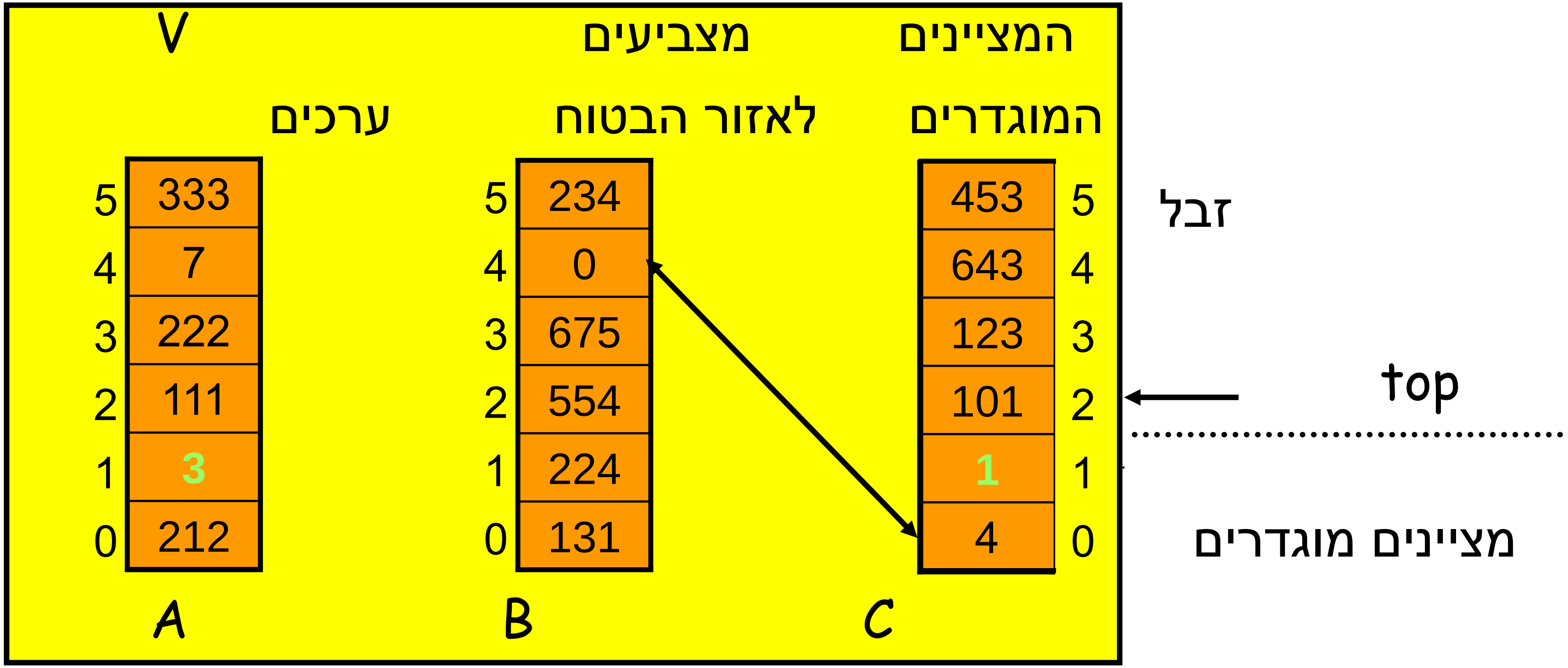
איתחול מערך בזמן O(1)

הבעיה: בייצוג הרגיל אתחול מערך דורש זמן $O(n)$. נראה ייצוג שיאפשר לאתחל מערך בזמן $O(1)$. בתמורה נשתמש ביותר זיכרון.



איתחול מערך בזמן O(1)

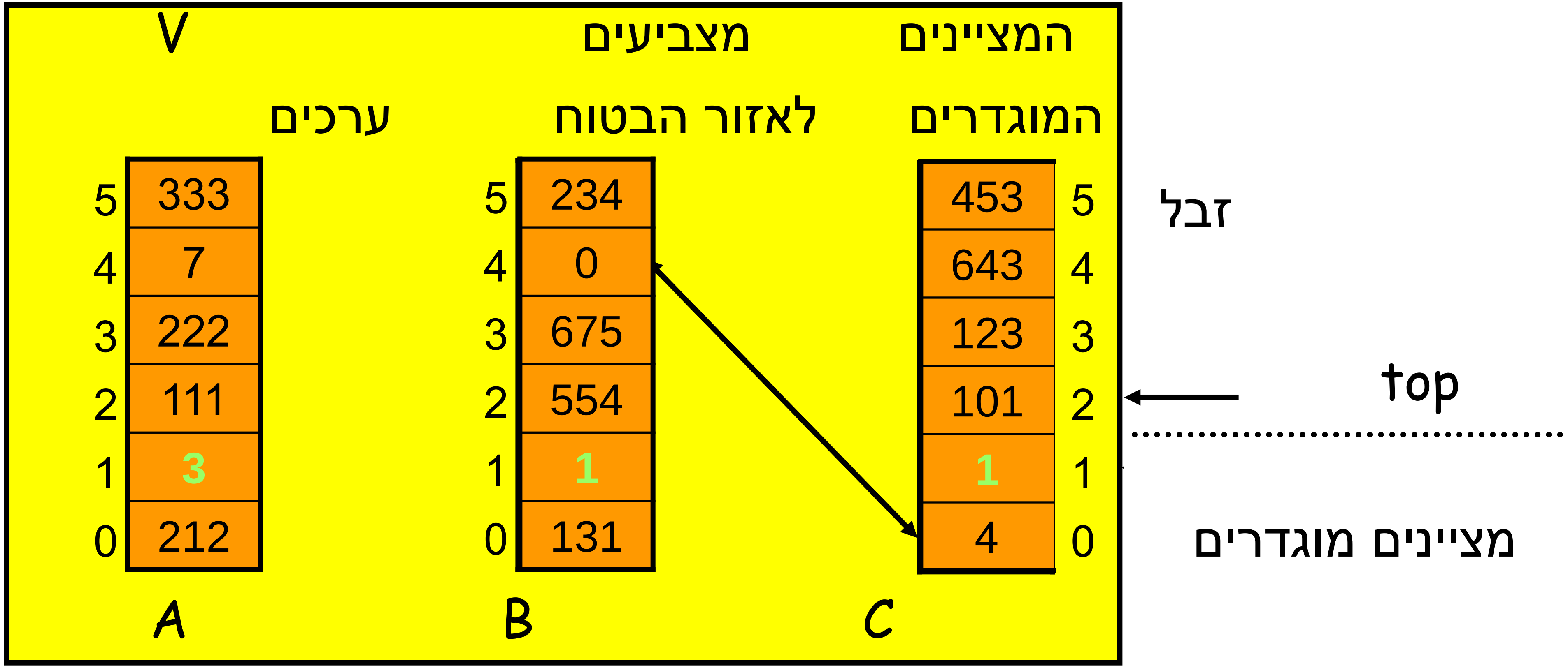
הבעיה: בייצוג הרגיל אתחול מערך דורש זמן $O(n)$. נראה ייצוג שיאפשר לאתחל מערך בזמן $O(1)$. בתמורה נשתמש ביותר זיכרון.



©cs,Technion

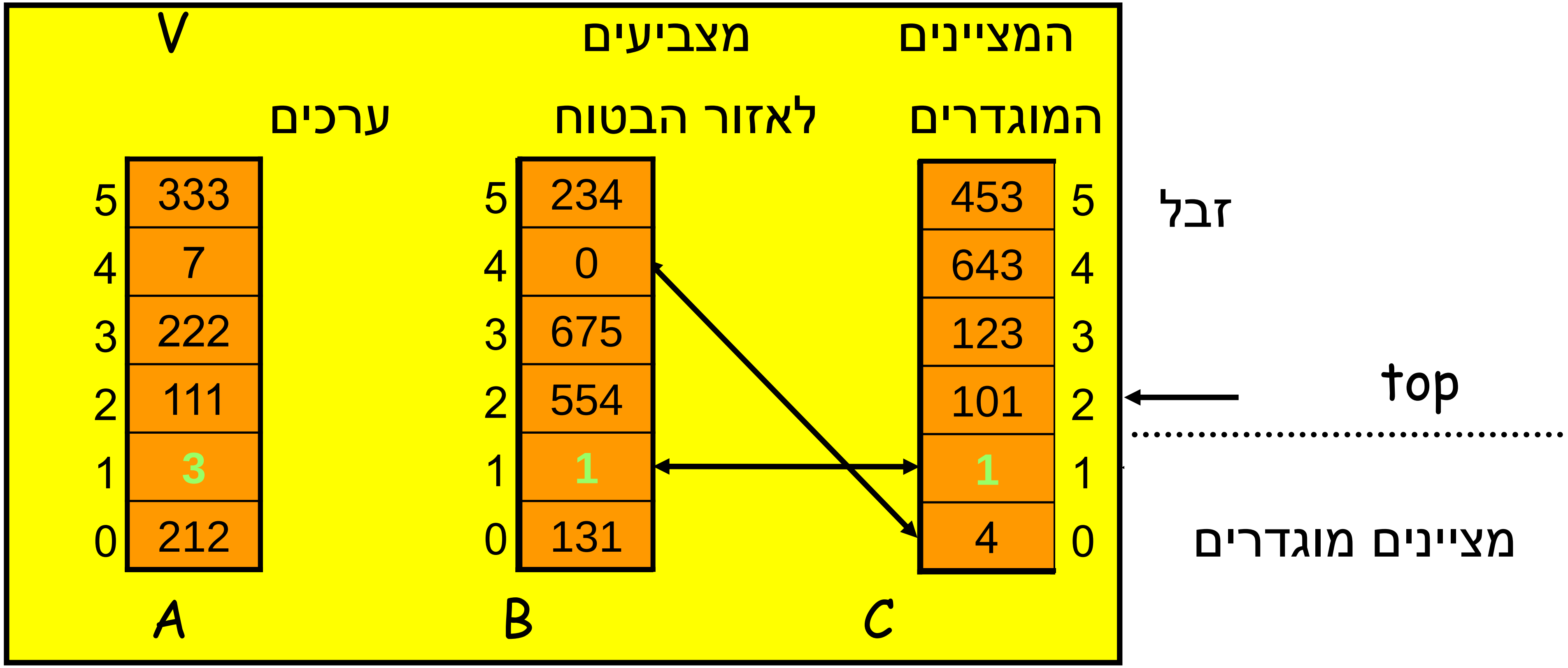
איתחול מערך בזמן O(1)

הבעיה: בייצוג הרגיל אתחול מערך דורש זמן $O(n)$. נראה ייצוג שיאפשר לאתחל מערך בזמן $O(1)$. בתמורה נשתמש ביותר זיכרון.



איתחול מערך בזמן O(1)

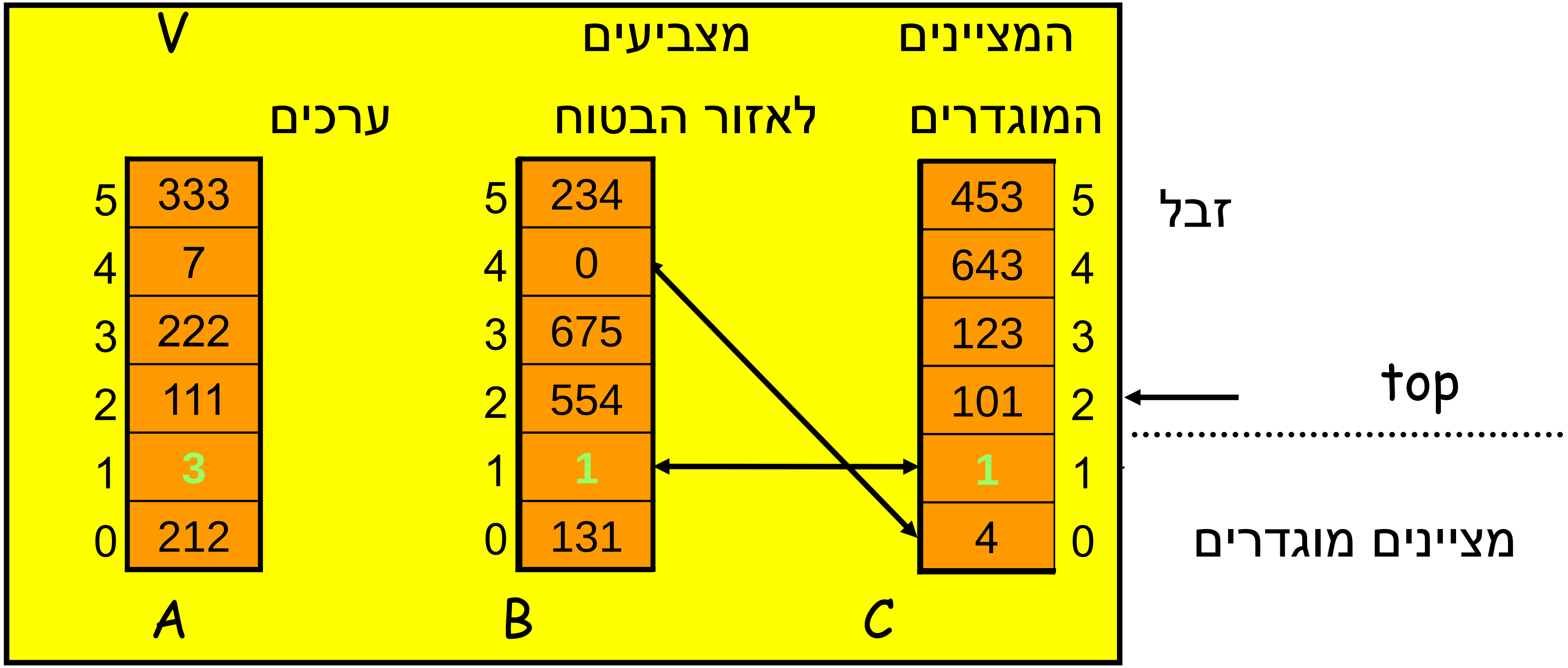
הבעיה: בייצוג הרגיל אתחול מערך דורש זמן $O(n)$. נראה ייצוג שיאפשר לאתחל מערך בזמן $O(1)$. בתמורה נשתמש ביותר זיכרון.



©cs,Technion

איתחול מערך בזמן O(1)

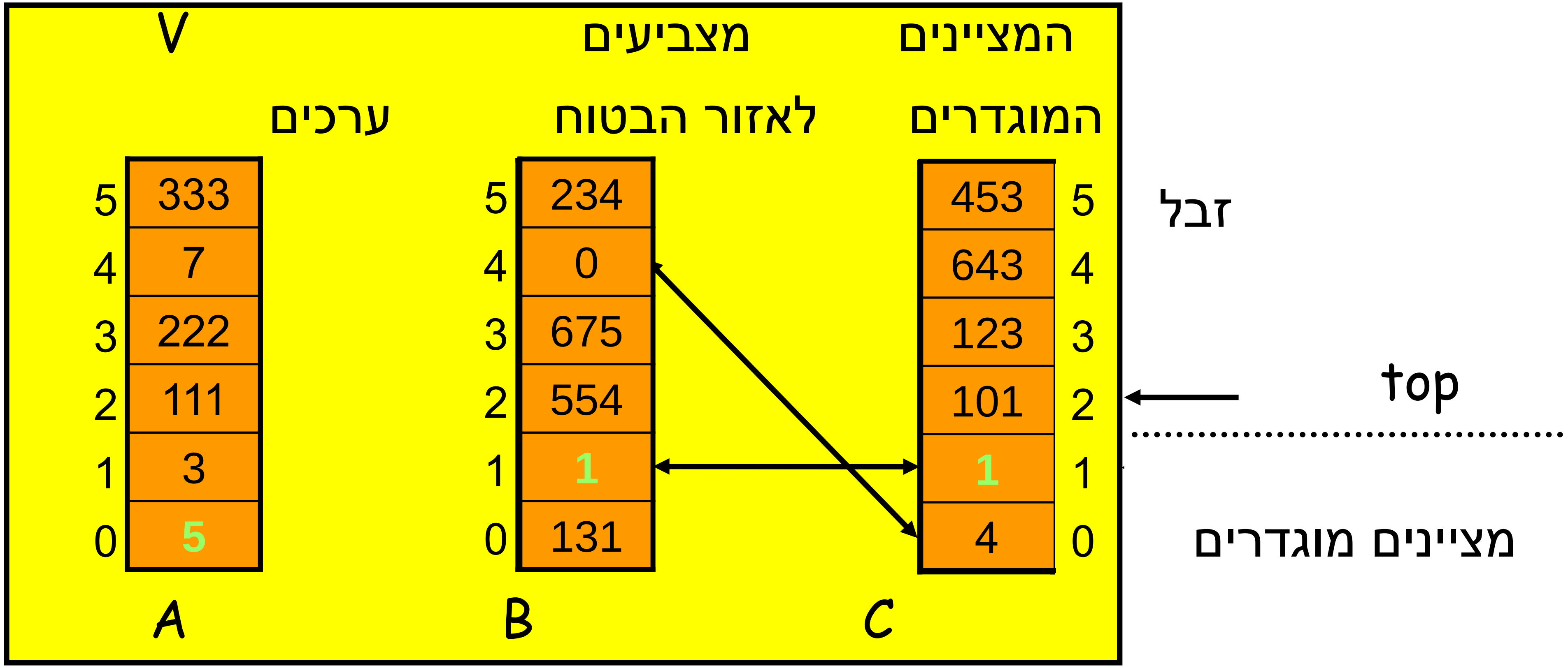
<u>הבעיה</u>: בייצוג הרגיל אתחול מערך דורש זמן $O(n)$. נראה ייצוג שיאפשר לאתחל מערך בזמן $O(1)$. בתמורה נשתמש ביותר זיכרון.



בדוגמא זו: V[0]=5, V[1]=3, V[4]=7

איתחול מערך בזמן O(1)

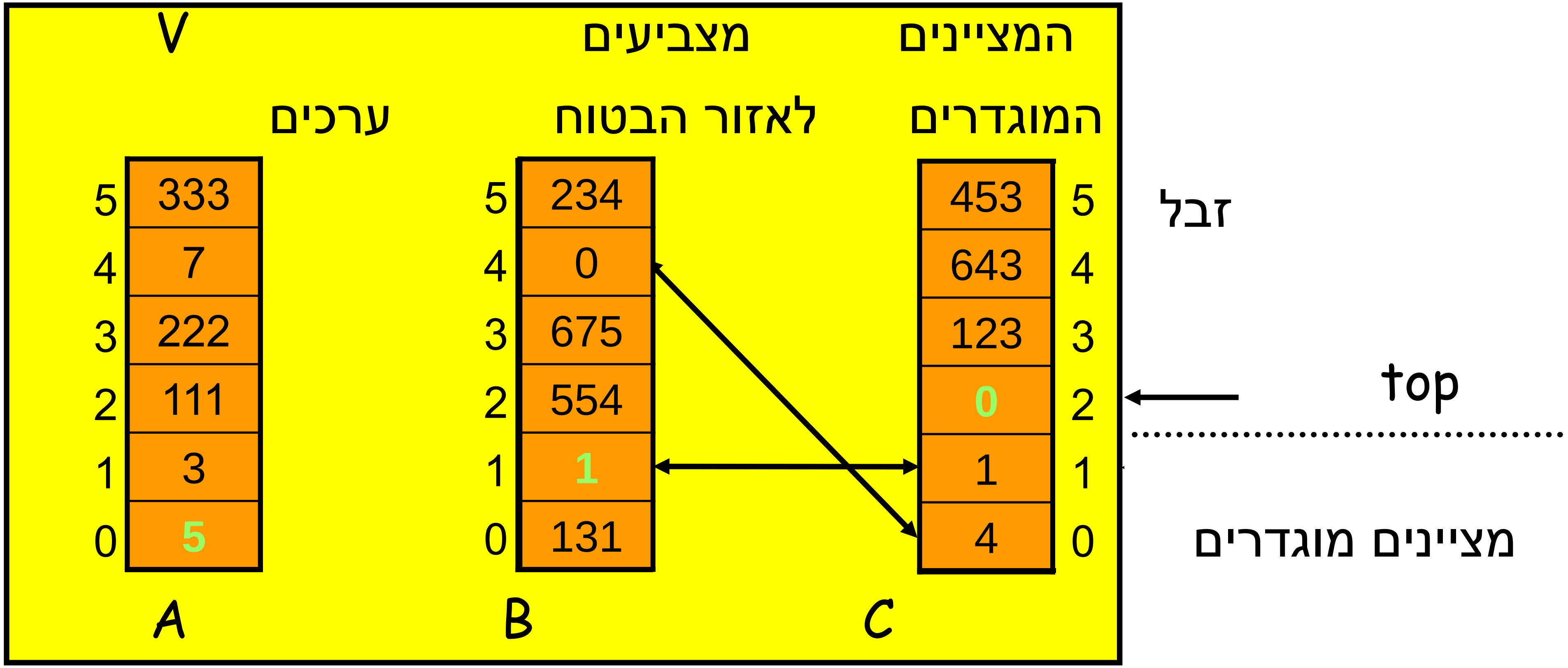
הבעיה: בייצוג הרגיל אתחול מערך דורש זמן $O(n)$. נראה ייצוג שיאפשר לאתחל מערך בזמן $O(1)$. בתמורה נשתמש ביותר זיכרון.



בדוגמא זו: V[0]=5, V[1]=3, V[4]=7

איתחול מערך בזמן O(1)

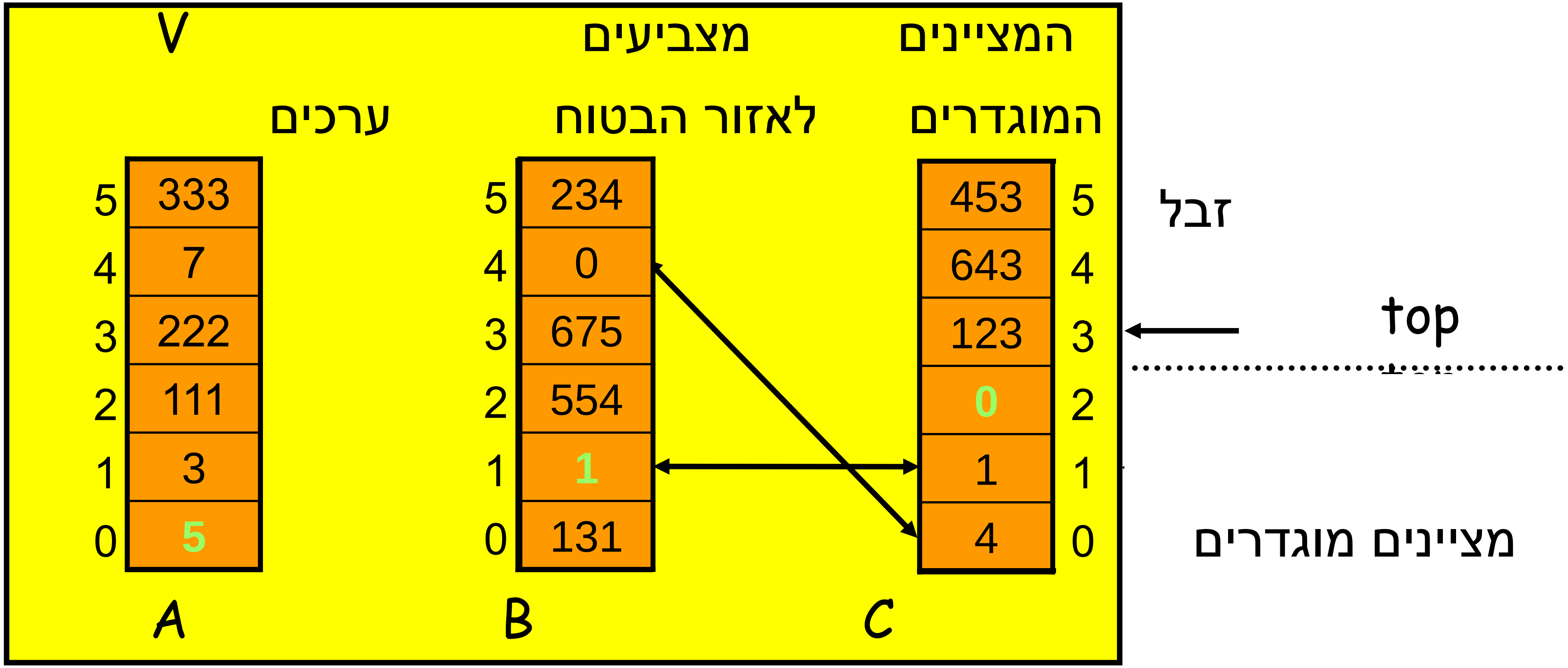
<u>הבעיה</u>: בייצוג הרגיל אתחול מערך דורש זמן $O(n)$. נראה ייצוג שיאפשר לאתחל מערך בזמן $O(1)$. בתמורה נשתמש ביותר זיכרון.



בדוגמא זו: V[0]=5, V[1]=3, V[4]=7

איתחול מערך בזמן O(1)

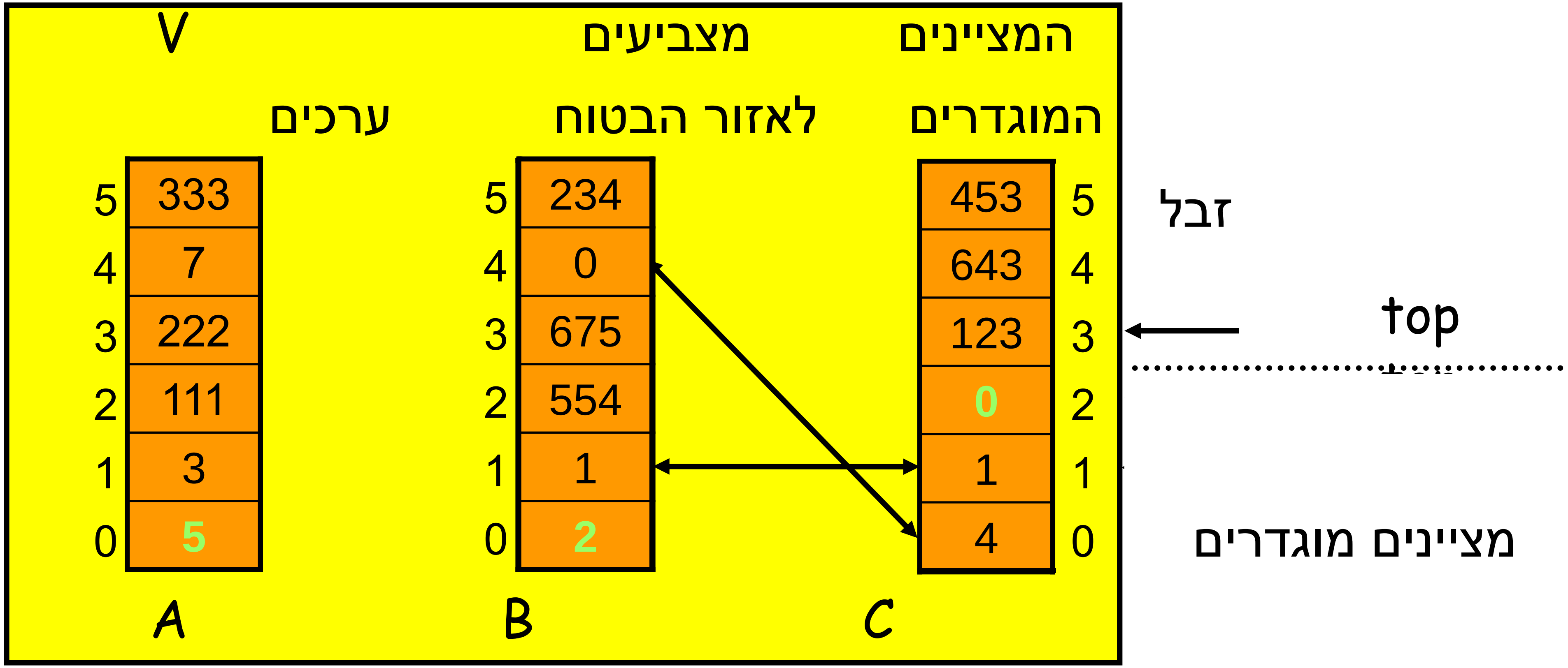
<u>הבעיה</u>: בייצוג הרגיל אתחול מערך דורש זמן $O(n)$. נראה ייצוג שיאפשר לאתחל מערך בזמן $O(1)$. בתמורה נשתמש ביותר זיכרון.



בדוגמא זו: V[0]=5, V[1]=3, V[4]=7

איתחול מערך בזמן O(1)

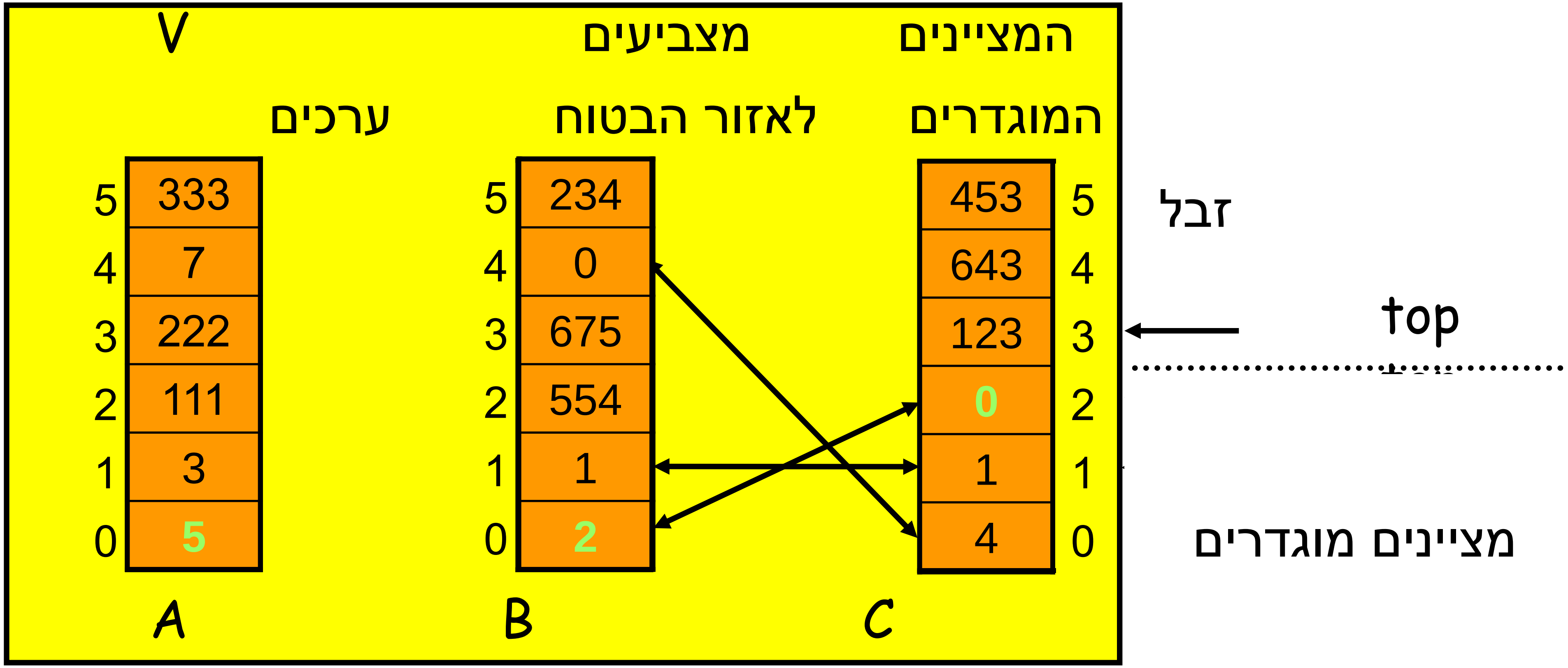
הבעיה: בייצוג הרגיל אתחול מערך דורש זמן $O(n)$. נראה ייצוג שיאפשר לאתחל מערך בזמן $O(1)$. בתמורה נשתמש ביותר זיכרון.



בדוגמא זו: V[0]=5, V[1]=3, V[4]=7

איתחול מערך בזמן O(1)

הבעיה: בייצוג הרגיל אתחול מערך דורש זמן $O(n)$. נראה ייצוג שיאפשר לאתחל מערך בזמן $O(1)$. בתמורה נשתמש ביותר זיכרון.



בדוגמא זו: V[0]=5, V[1]=3, V[4]=7

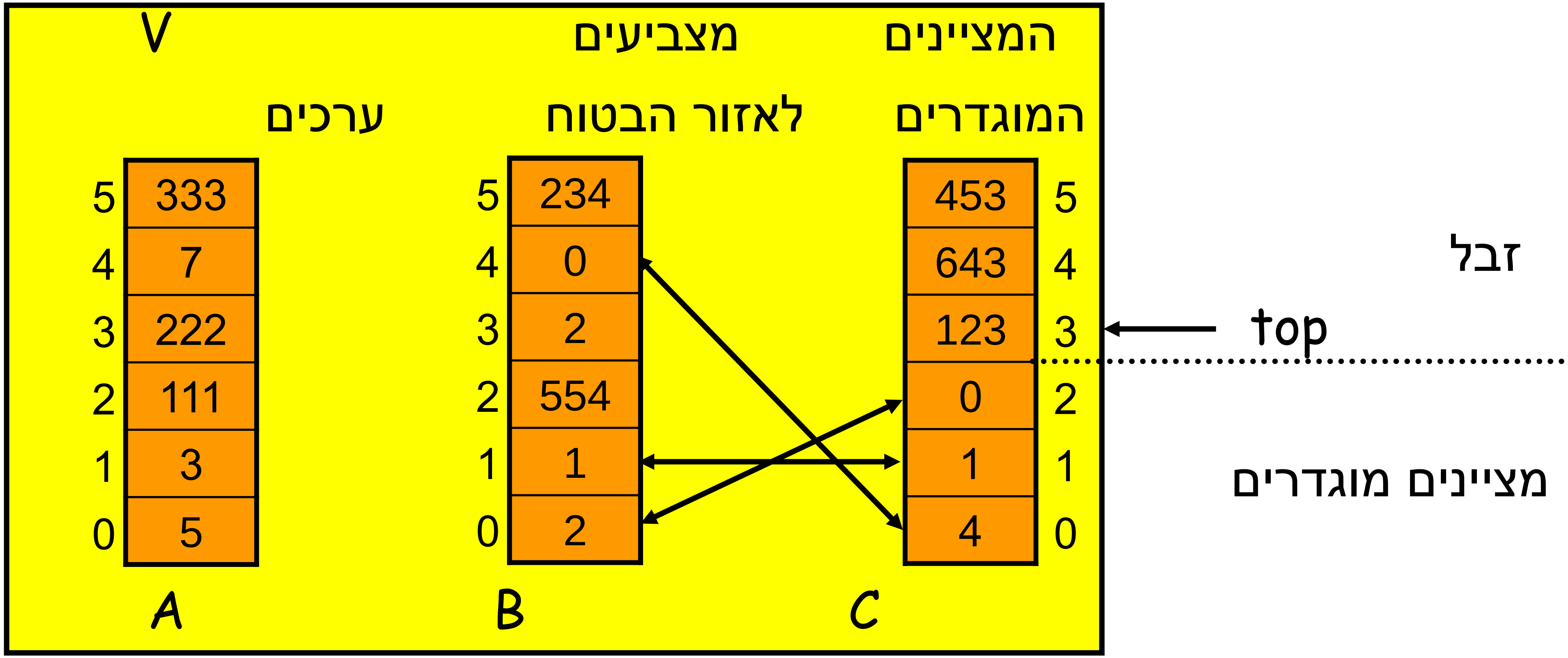
©cs,Technion

הקוד לאיתור זבל

```
int  is_garbage(int i) {
return !(( B[i] < top && (B[i] >= 0) && C[B[i]] = = i));  }
```

C[B[4]] = = 4   דוגמא לאיבר שהוכנס למבנה:

C[B[3]]= = 0 וכן C[B[5]] > top דוגמאות לזבל:



הקוד לפעולות המערך

אתחל init(V,const):

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

אחזר get(V,i):

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

שמור store(V,i,e):

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

הערה: ברגע שהמערך מתמלא (top > n) ניתן להשתמש במערך A בלבד
כיוון שאין, ויותר לא יהיה, זבל ב-A.

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:

<u>1</u>   <u>2 3</u>   <u>4 5 6</u>   <u>7 8 9 10</u>

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

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

חצי מערך תחתון (if $i \geq j$)

חצי מערך עליון (Otherwise)

ייצוג מטריצות דלילות

מטריצה דלילה היא מערך דו-ממדי עבורו "רוב מוחלט" של

האיברים שווה לאפס (או לקבוע c כלשהו).מהו רוב מוחלט ?

	0	1	2	3	4	5	6	7
0			3			1		
1		5						
2				4				
3								
4		2			7	8		

בדוגמא זו 7 מתוך 28
איברים שונים מהקבוע.

ייצוג מטריצות דלילות

מטריצה דלילה היא מערך דו-ממדי עבורו "רוב מוחלט" של

האיברים שווה לאפס (או לקבוע c כלשהו).מהו רוב מוחלט ?

	0	1	2	3	4	5	6	7
0			3			1		
1		5						
2				4				
3								
4		2			7	8		

בדוגמא זו 7 מתוך 28 איברים שונים מהקבוע.

<u>הגדרה אסימפטוטית</u>: אם סדרת מטריצות ... $M_1 \ldots M_n$ מקיימת שמספר האיברים השונים מקבוע c הוא מסדר גודל של $o(m(n))$ (o קטן) כאשר $m(n)$ הוא מספר המקומות המקסימלי במטריצה M_n אז זוהי סדרה של מטריצות דלילות. <u>דוגמא</u>: מטריצות אלכסוניות. במטריצות ריבועיות דלילות מגודל $n \cdot n$ מספר האיברים השונים מקבוע c הוא $o(n^2)$.

ייצוג מטריצות דלילות

מטריצה דלילה היא מערך דו-ממדי עבורו "רוב מוחלט" של

האיברים שווה לאפס (או לקבוע c כלשהו).מהו רוב מוחלט ?

	0	1	2	3	4	5	6	7
0			3			1		
1		5						
2				4				
3								
4		2			7	8		

בדוגמא זו 7 מתוך 28 איברים שונים מהקבוע.

הגדרה אסימפטוטית: אם סדרת מטריצות ... $M_1...M_n$ מקיימת שמספר האיברים השונים מקבוע c הוא מסדר גודל של $o(m(n))$ (o קטן) כאשר $m(n)$ הוא מספר המקומות המקסימלי במטריצה M_n אז זוהי סדרה של מטריצות דלילות. דוגמא: מטריצות אלכסוניות. במטריצות ריבועיות דלילות מגודל $n{\cdot}n$ מספר האיברים השונים מקבוע c הוא $o(n^2)$.

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)

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: חוסך בזיכרון עבור מטריצות דלילות.

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 נ"ל. זמן המיזוג הוא אורכן הכולל של הרשימות כלומר $O(m+k)$. כלומר עבור מטריצות ריבועיות דלילות זמן החיבור הוא $o(n^2)$.

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: חוסך בזיכרון עבור מטריצות דלילות.

מאיץ פעולות חיבור וכפל של מטריצות דלילות.

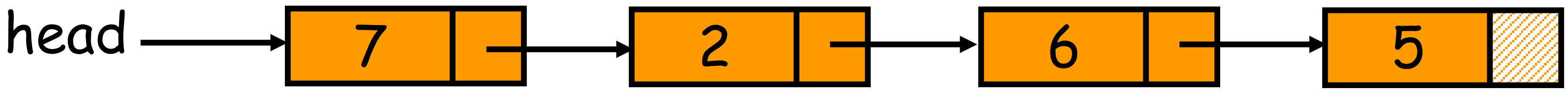
חיבור מטריצות בגודל $n \times n$ לוקח בייצוג רגיל זמן $O(n^2)$.

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 נ"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

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

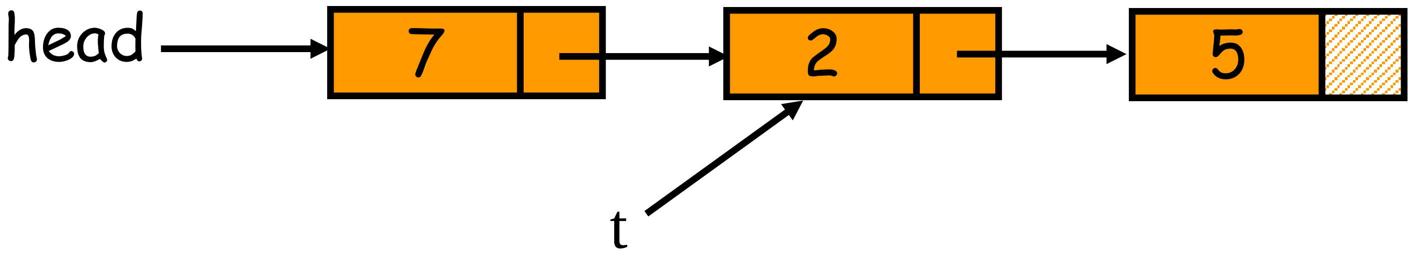
```
void  init (NODE  *head){
        (* head) = NULL;
}
```

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

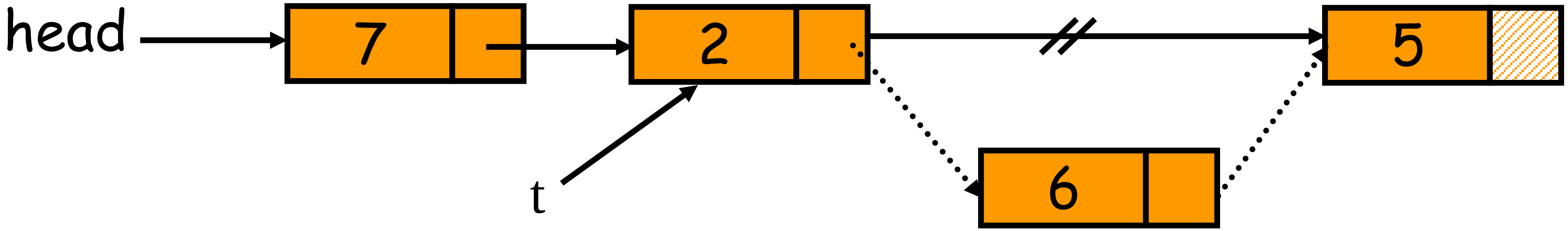
```
int  insert ( NODE  *t,  DATA_TYPE x){
          NODE *p;
     if (t == NULL) return 0;
     p = (NODE *) malloc (sizeof (NODE));
     p -> info = x;
     p -> next = t -> next ;
     t -> next  = p ;
     return 1;
}
```

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).



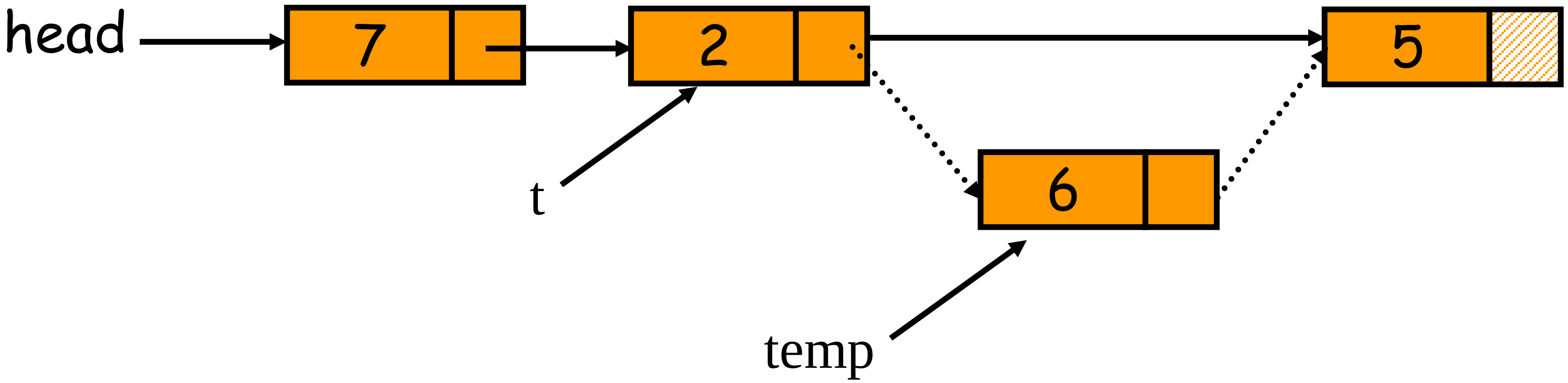
הוצאת איבר מרשימה מקושרת

פעולת delete(t):

הפרמטר t מצביע לצומת שלפני הצומת שמוציאים.

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

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

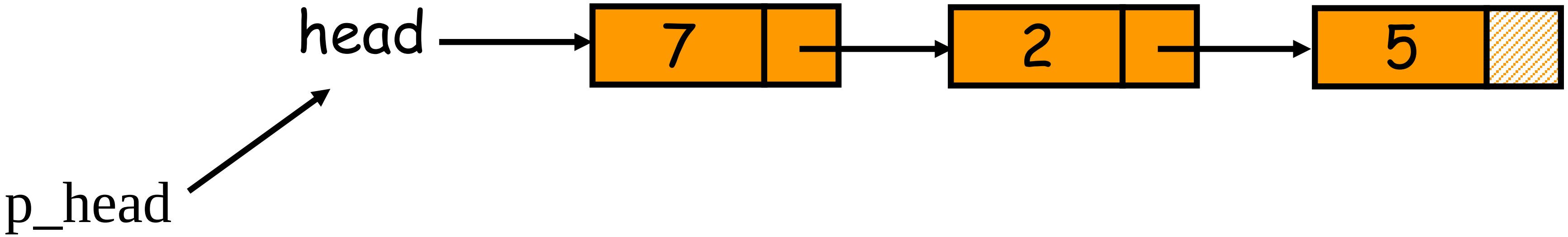


©cs,Technion

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):



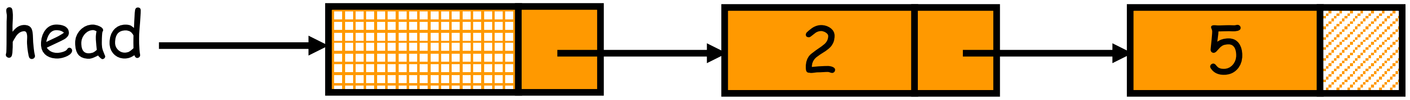
תכנות זה מסורבל. מהו הפתרון לכך ?

רשימות עם כותרת

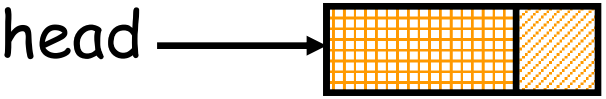
מוסיפים איבר "ריק" בתחילת הרשימה. תוספת זו חוסכת:

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

בייצוג זה רשימה בת שתי איברים ממומשת כך:



ורשימה ריקה ממומשת כך:

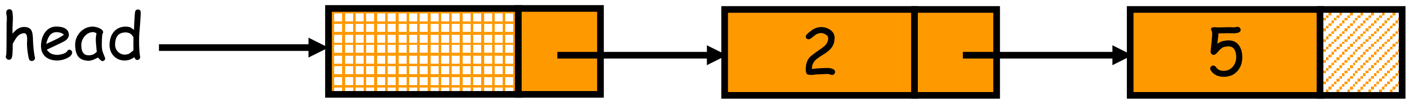


רשימות עם כותרת

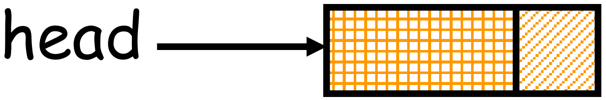
מוסיפים איבר "ריק" בתחילת הרשימה. תוספת זו חוסכת:

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

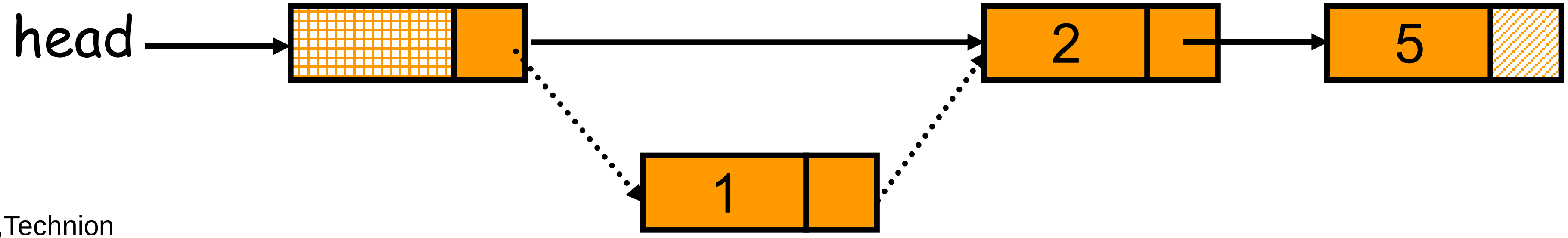
בייצוג זה רשימה בת שתי איברים ממומשת כך:



ורשימה ריקה ממומשת כך:

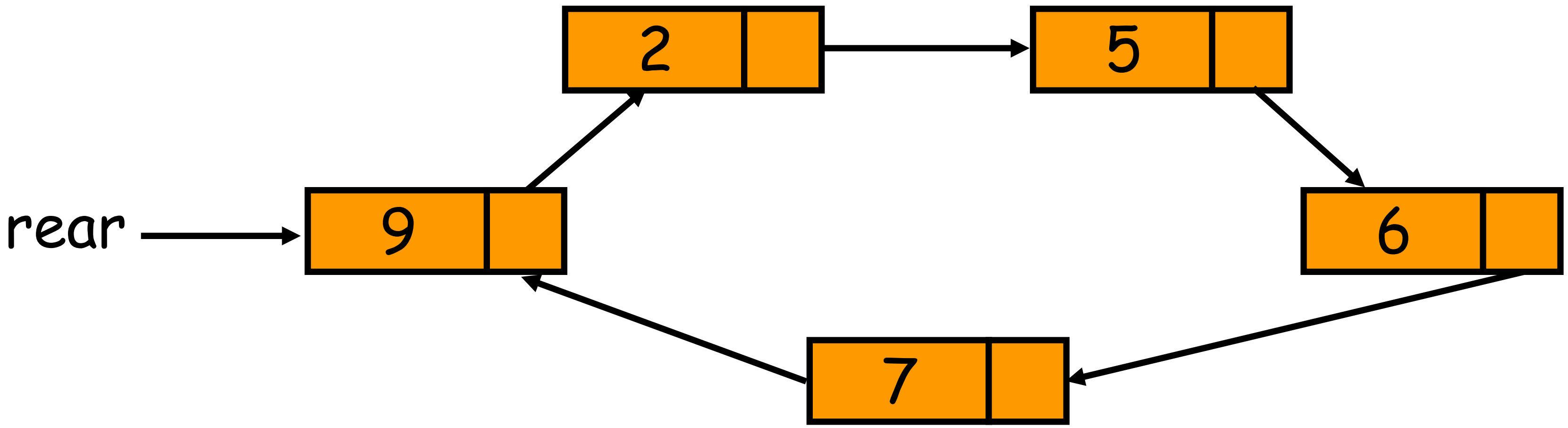


הכנסה לפני איבר ראשון זהה להכנסה לפני איבר כלשהו:



©cs,Technion

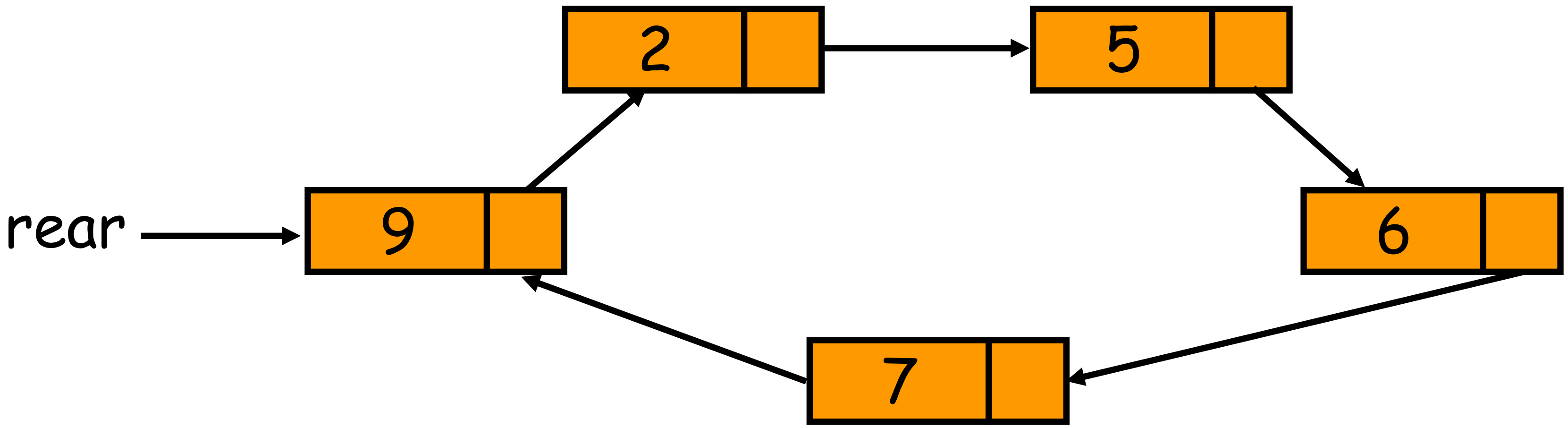
רשימות מעגליות



יתרונות:

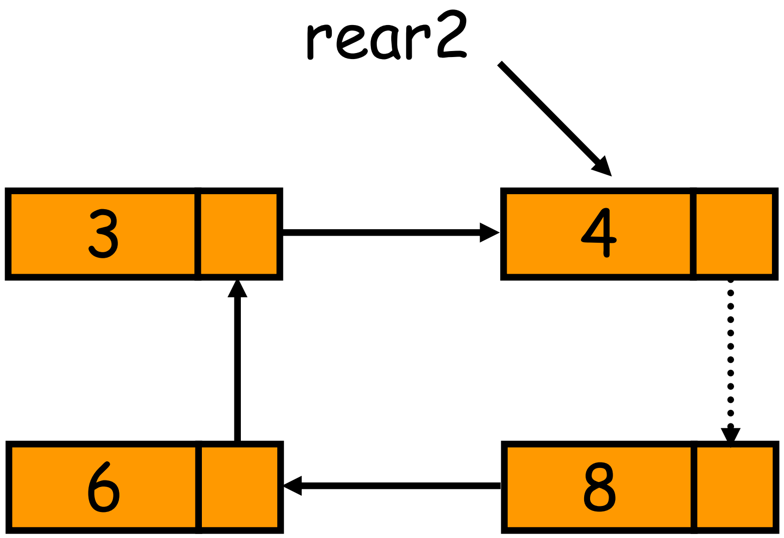
- אפשר להגיע מכל איבר לכל איבר
- ניתן לשרשר רשימות מעגליות בזמן קבוע.

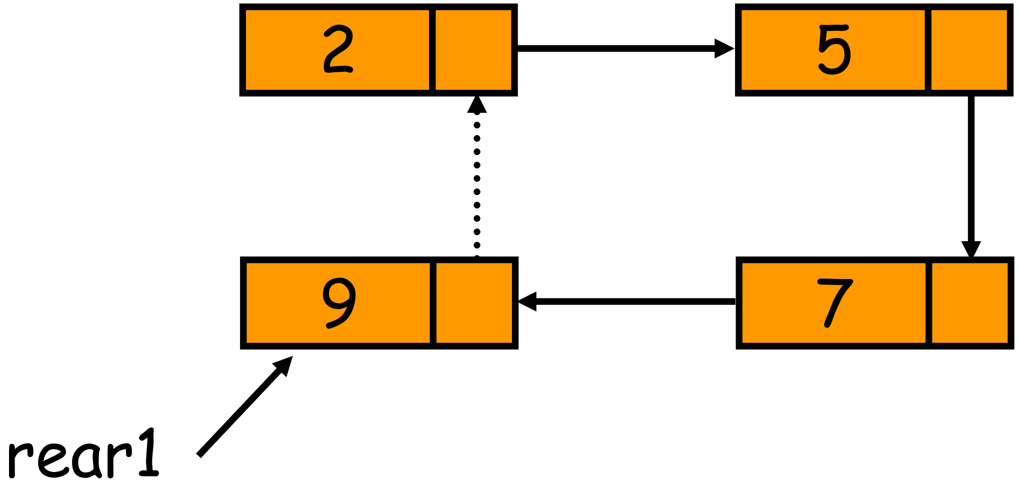
רשימות מעגליות



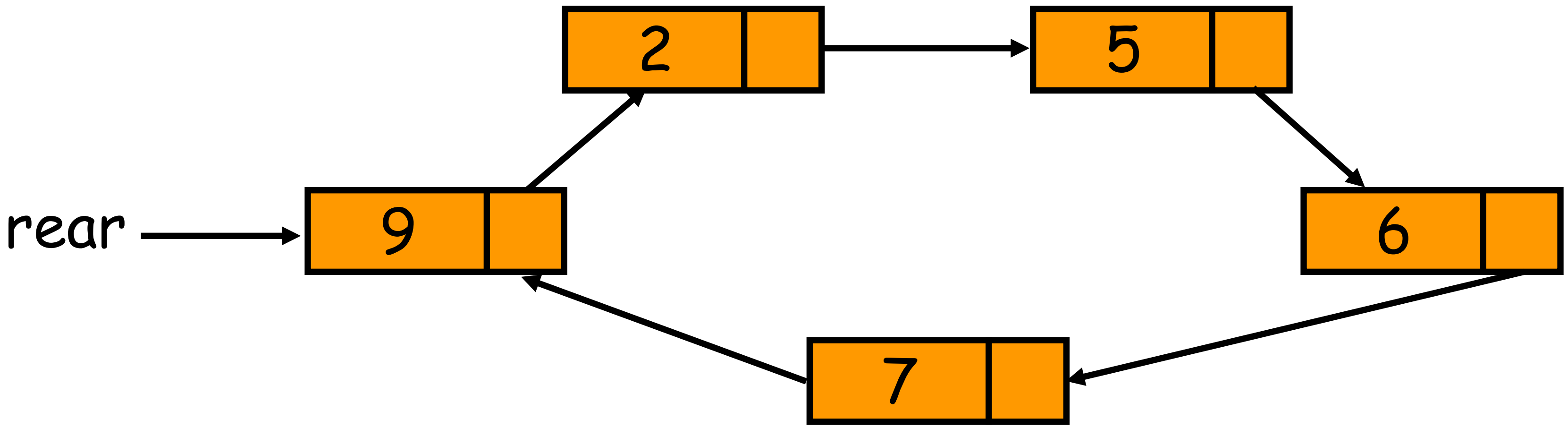
יתרונות:

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



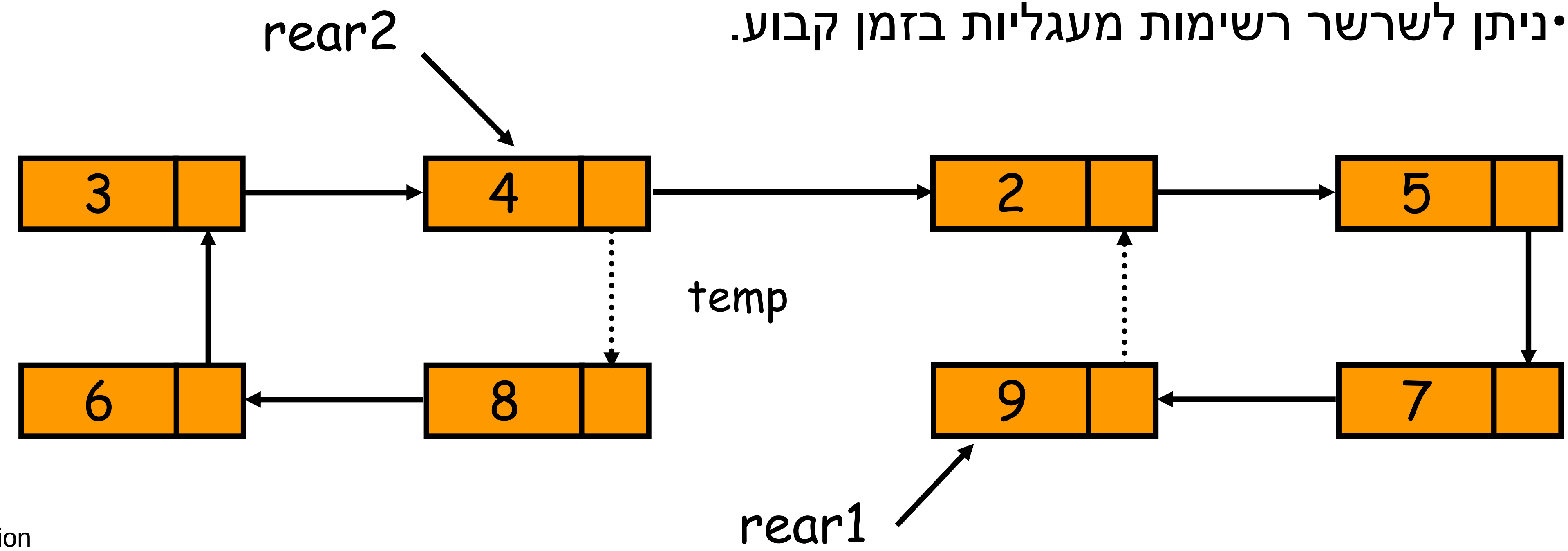


רשימות מעגליות

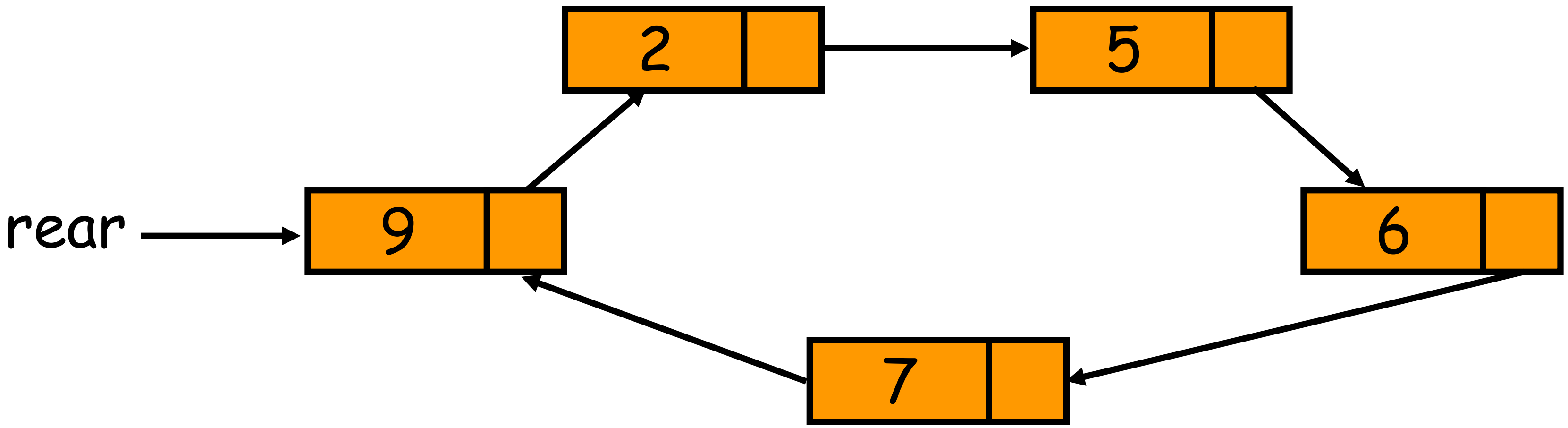


<u>יתרונות:</u>

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

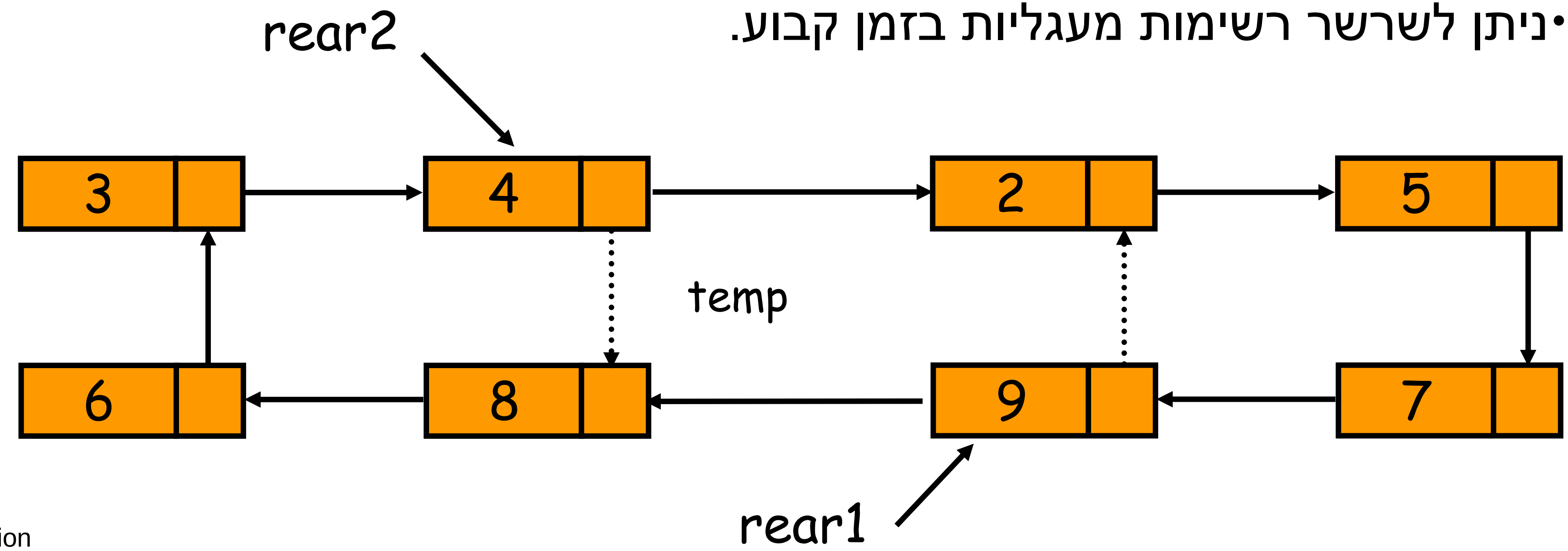


רשימות מעגליות



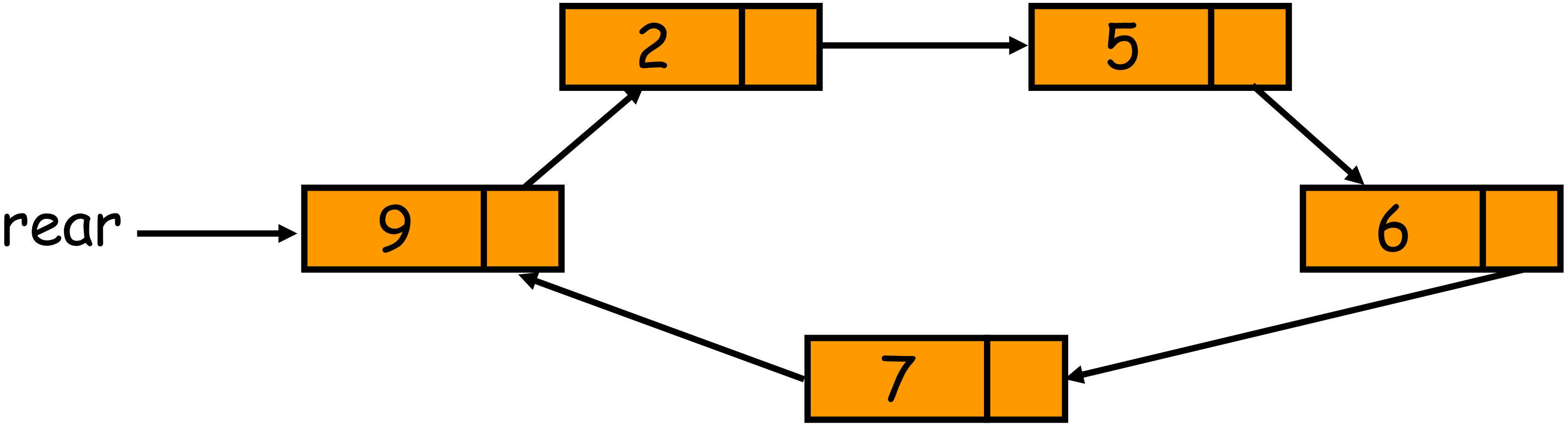
יתרונות:

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



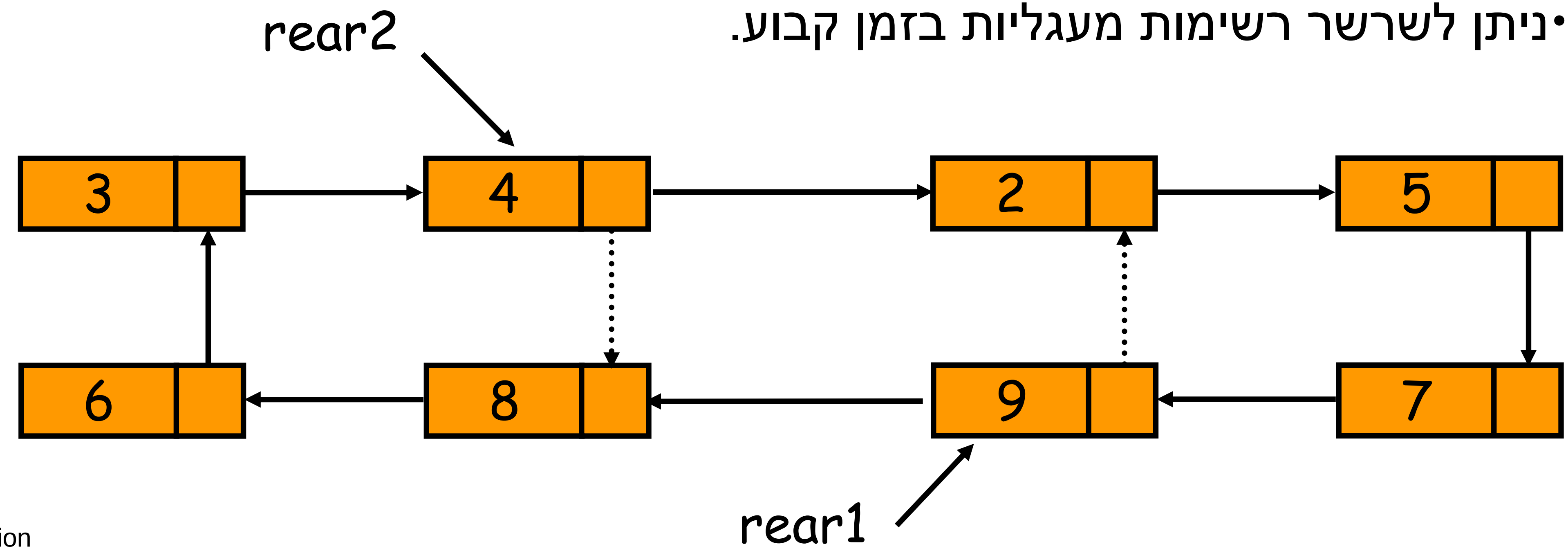
©cs,Technion

רשימות מעגליות

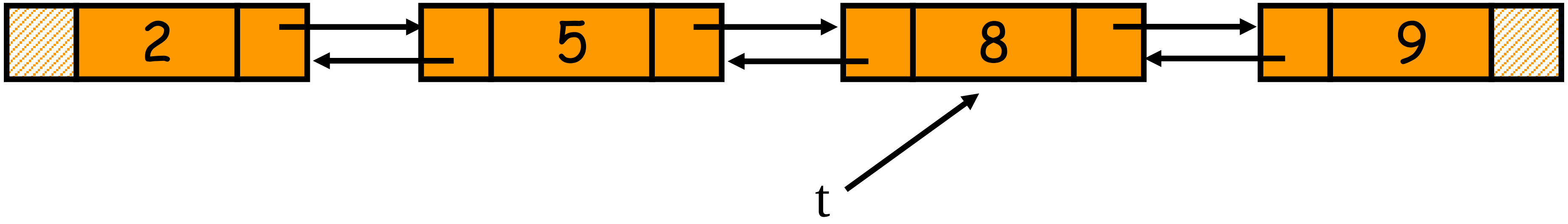


יתרונות:

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



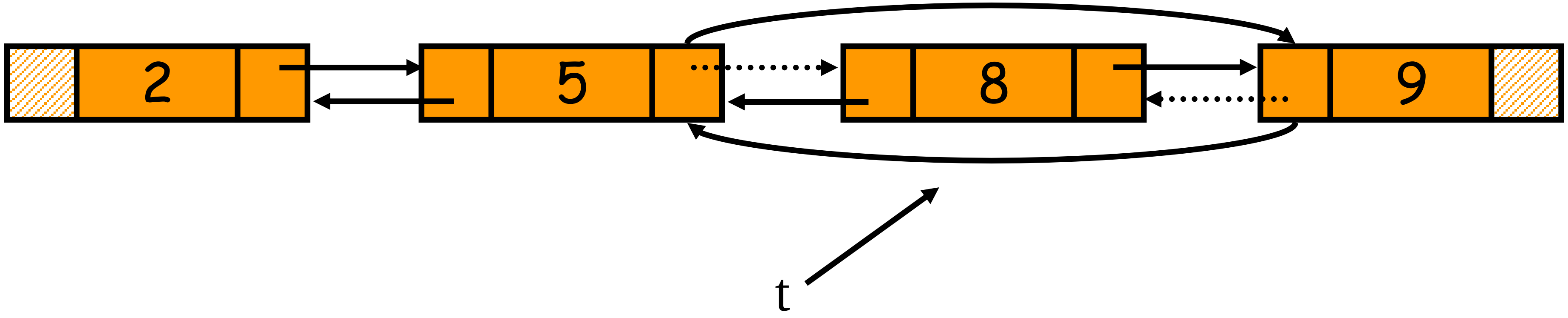
רשימה מקושרת דו-כיוונית



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

$t \rightarrow next \rightarrow prev = t \rightarrow prev ;$

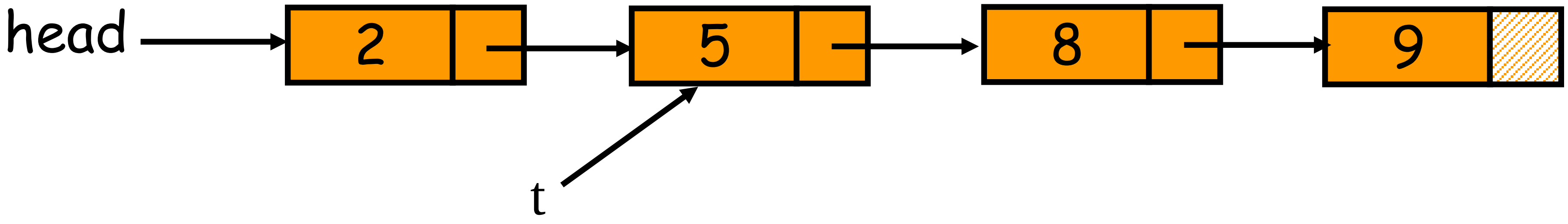
$t \rightarrow prev \rightarrow next = t \rightarrow next ;$



©cs,Technion

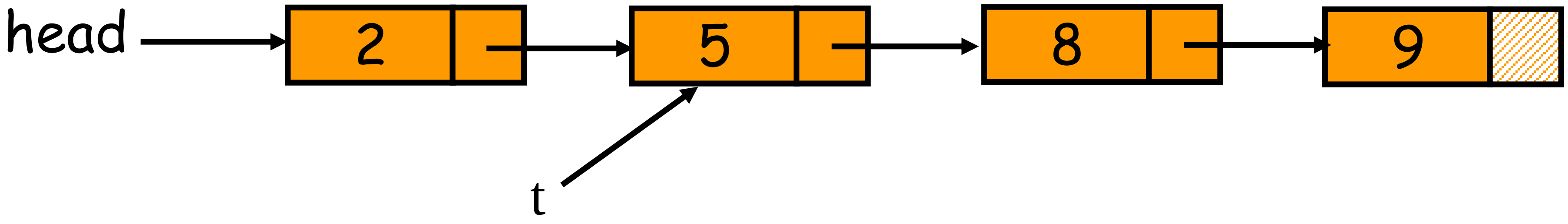
הוצאה "טריקית"

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



הוצאה "טריקית"

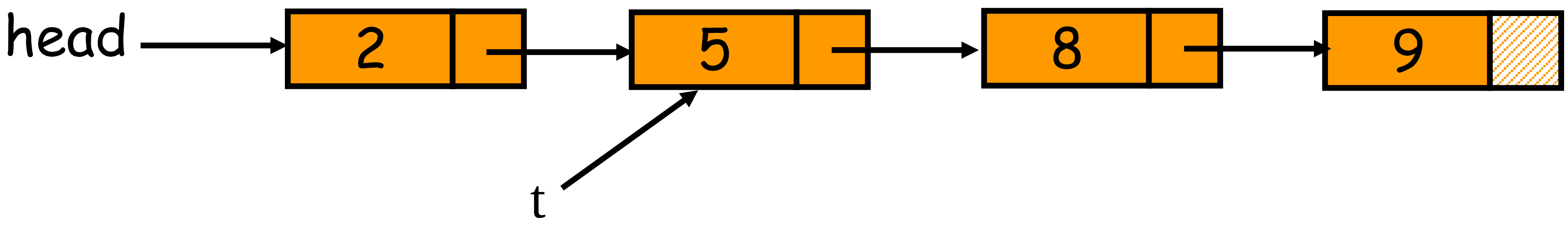
האם ניתן להוציא איבר אליו מצביע t גם מרשימה חד-כוונית ?



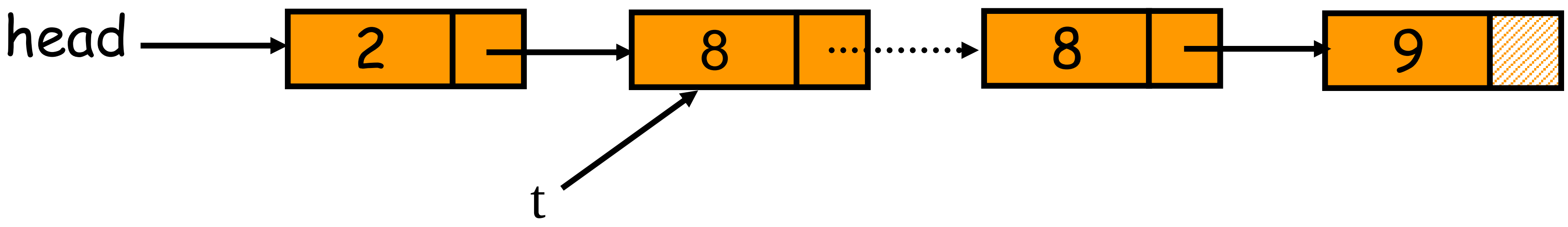
פשוט נעתיק את האינפורמציה בתא העוקב ל- t לתא ש- t מצביע אליו ונוציא את התא העוקב.

הוצאה "טריקית"

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



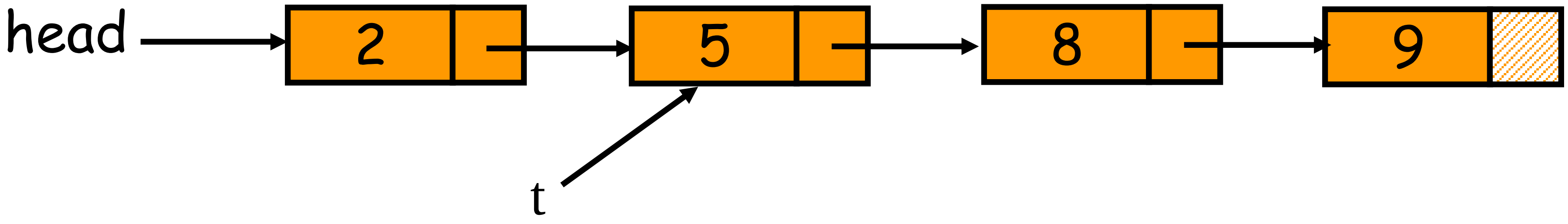
פשוט נעתיק את האינפורמציה בתא העוקב ל- t לתא ש- t מצביע אליו ונוציא את התא העוקב.



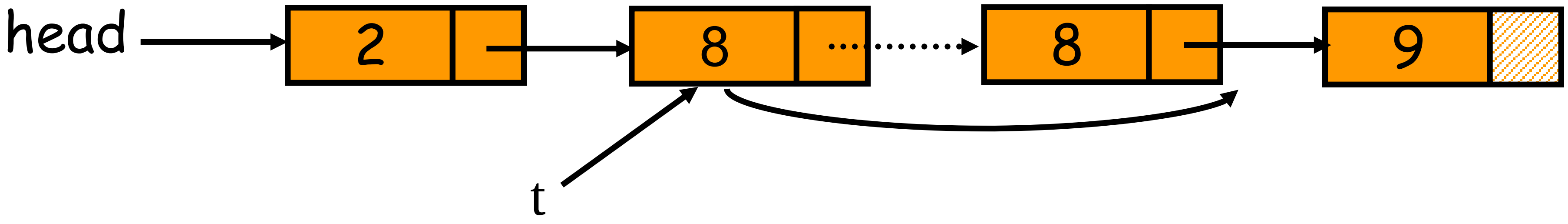
©cs,Technion

הוצאה "טריקית"

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

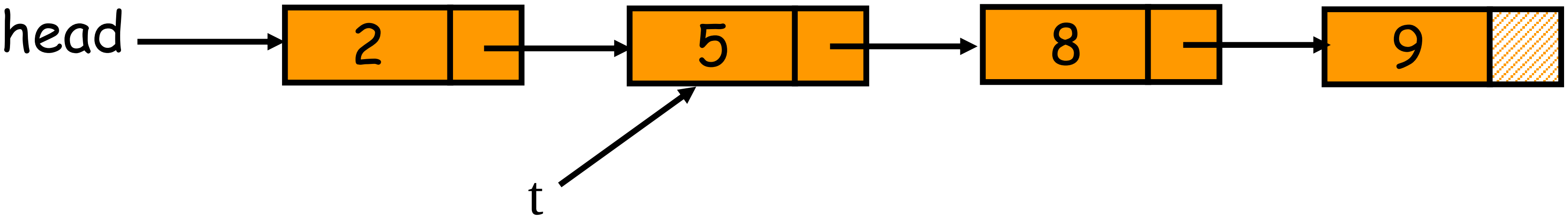


פשוט נעתיק את האינפורמציה בתא העוקב ל- t לתא ש- t מצביע אליו ונוציא את התא העוקב.

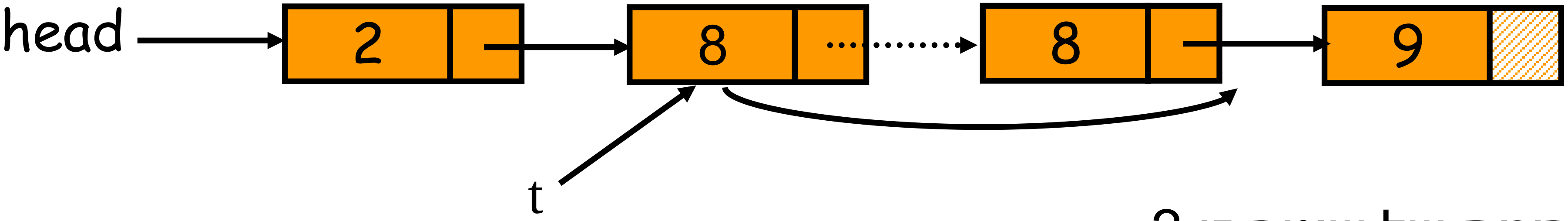


הוצאה "טריקית"

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.

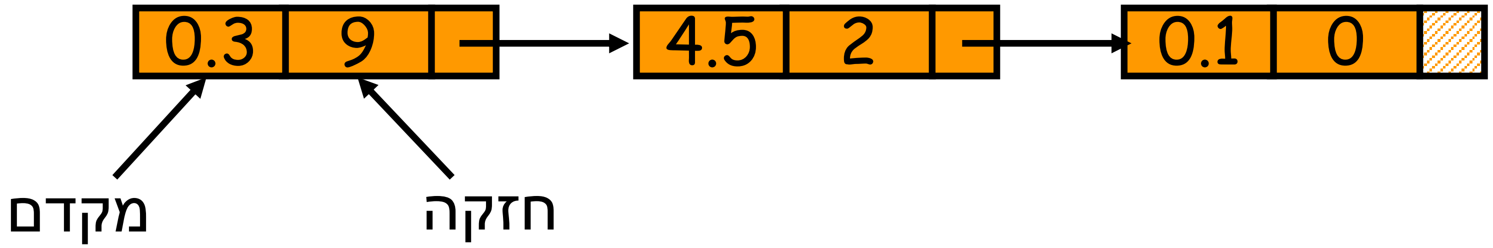


מהן החסרונות של שיטה זו ?

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

ייצוג פולינומים

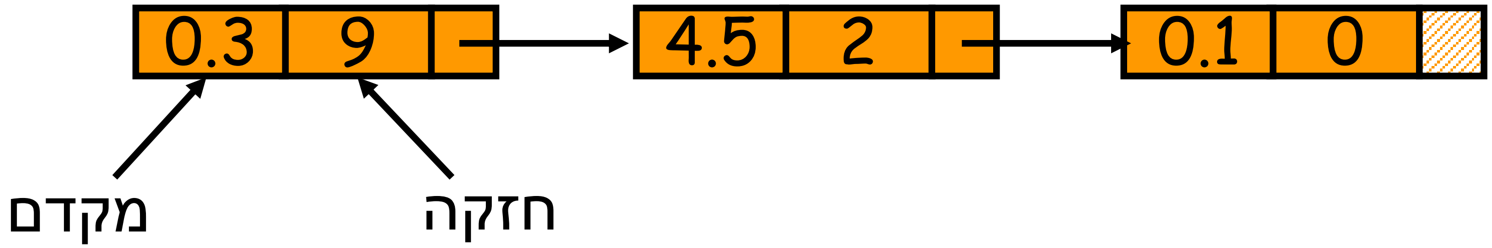
<u>נעלם אחד</u>. דוגמא: $0.3x^9 + 4.5x^2 + 0.1$



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

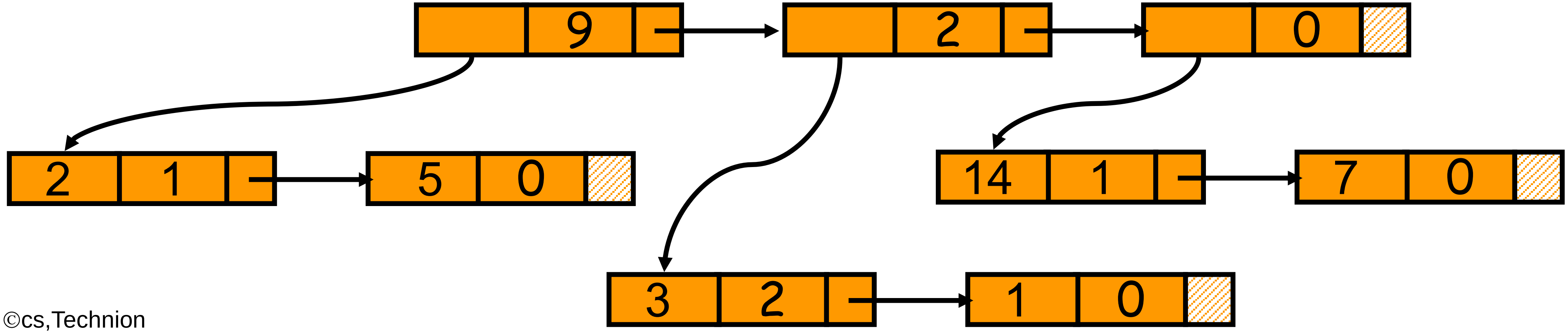
ייצוג פולינומים

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



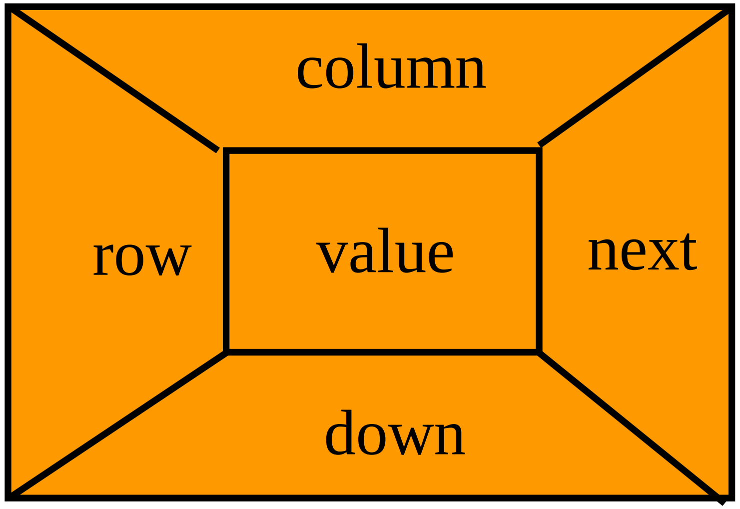
מימוש זה יעיל כאשר ההפרש בין החזקה הגבוהה והנמוכה גדול וכן הרבה מקדמים של חזקות הביניים מתאפסים.

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

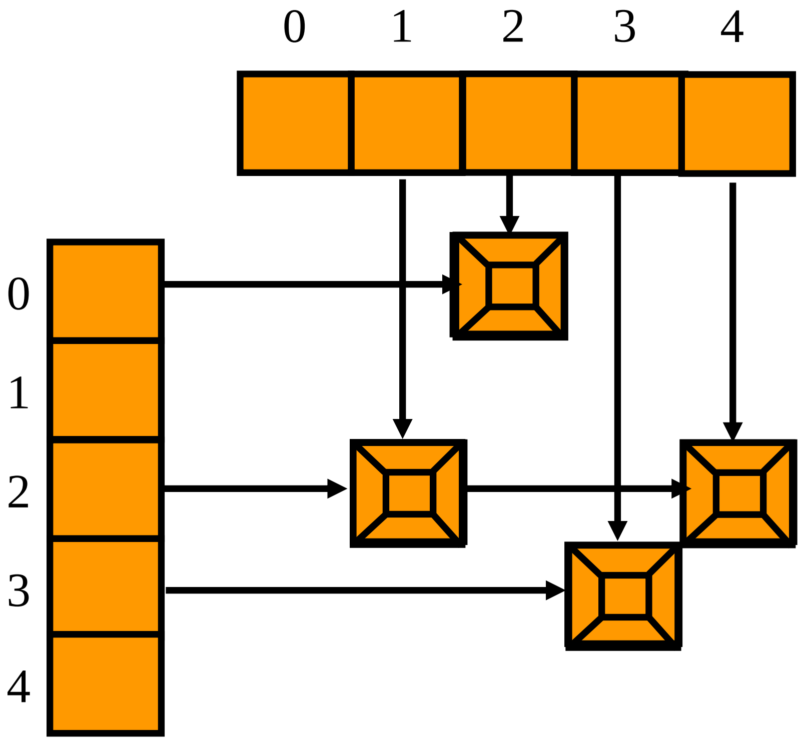


ייצוג מטריצות דלילות

מבנה צומת:




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



כפל מטריצות דלילות

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

```
#define N_row 20
#define N_col 20
NODE *row_header[N_row],
          *col_header[N_col];
/* Compute the value of c[i][j]  */
float mult_row_col(int i, int j){
float 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;
}
```

Arrays+lists