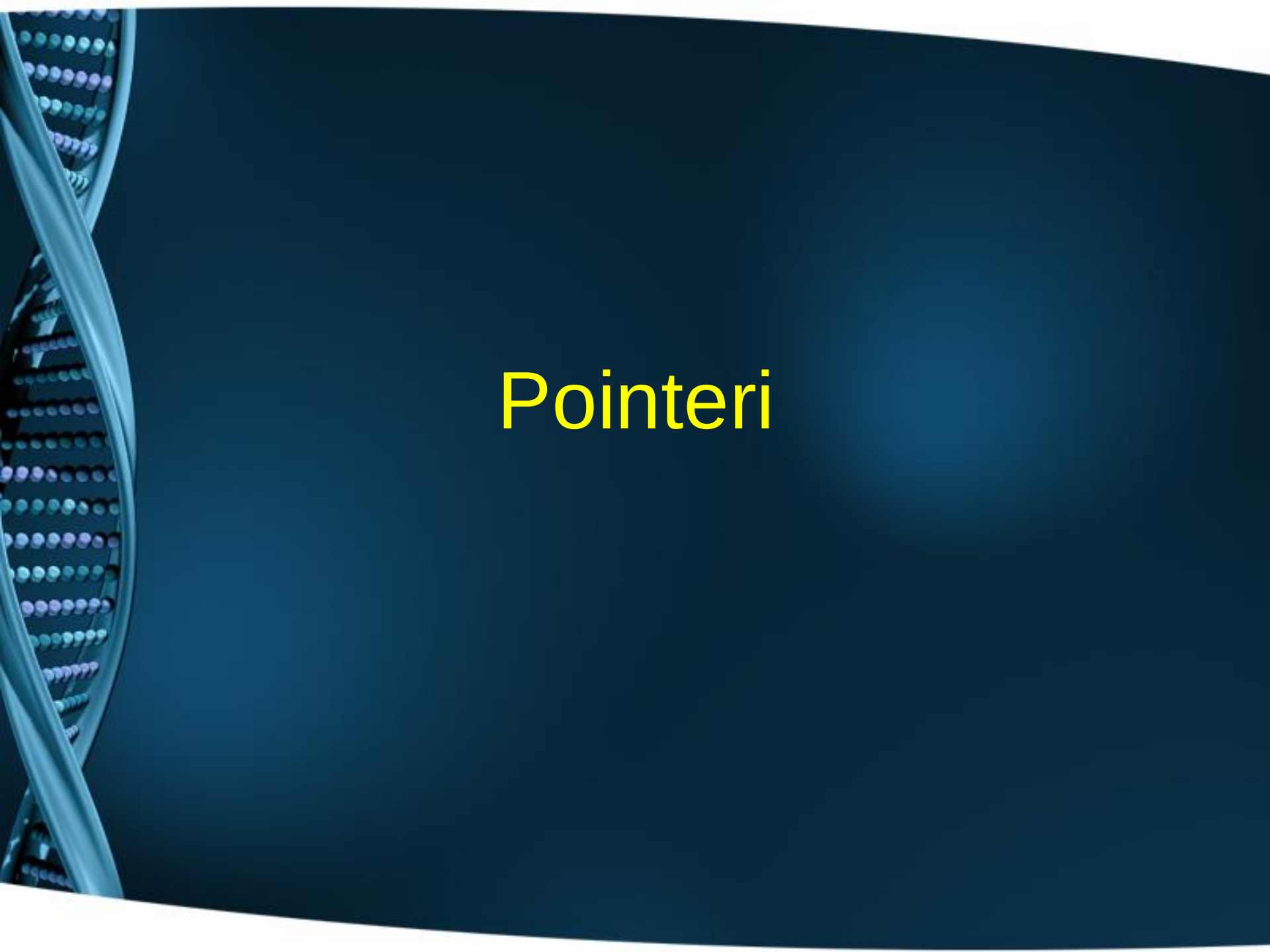
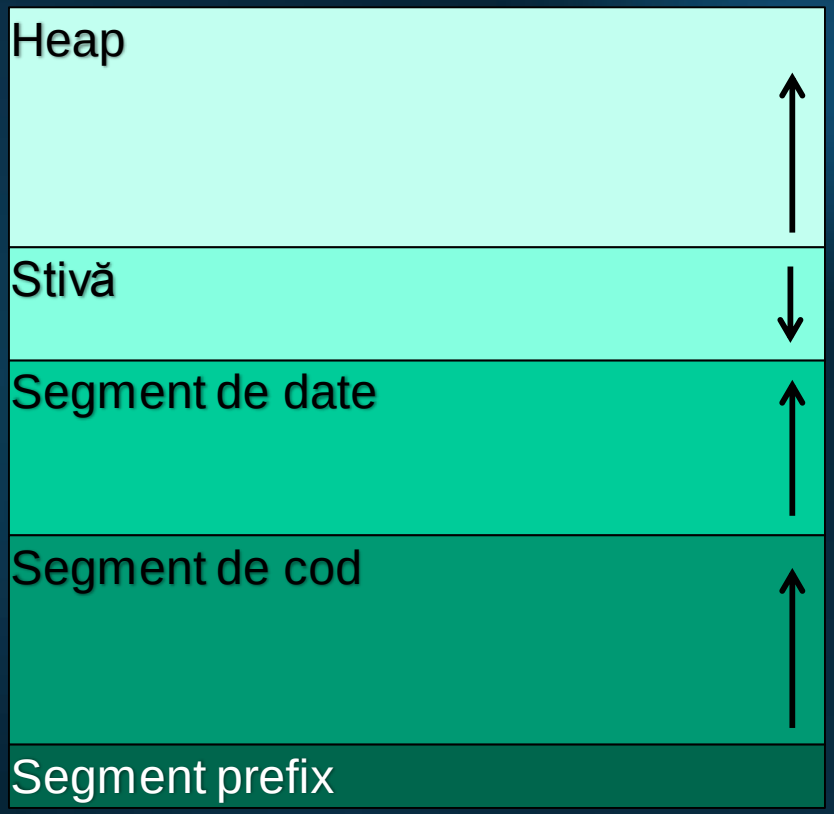

Pointeri

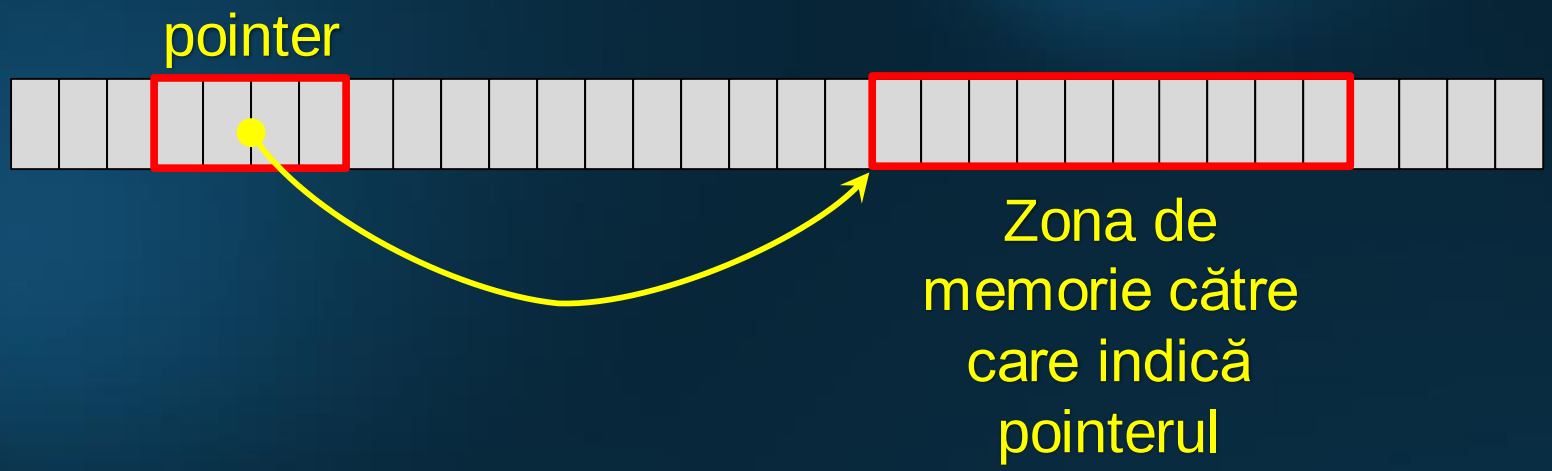
Structura memoriei la execuția unui program

- S.O. alocă o zonă de memorie programului
- Segmente de memorie



Tipul de dată pointer

- Pointer ⇔ adresă de memorie
  - accesarea unei zone de memorie fără ”nume”
  - constantă sau variabilă
  - cu tip sau fără tip



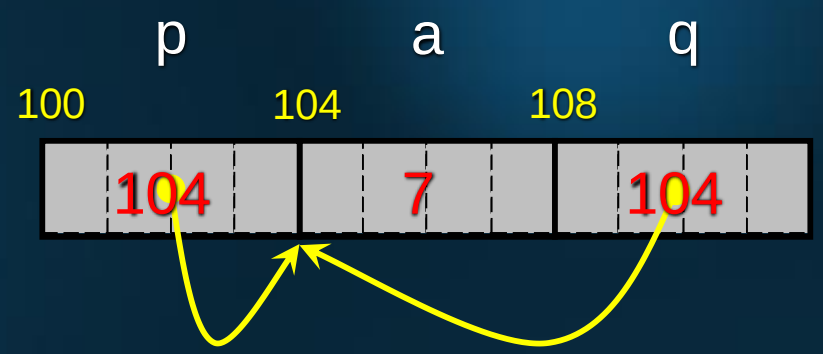
- Pointerul - dată statică sau dinamică?

Operatori specifici pointerilor

Nume	Simbol	Rol	Utilizare
Operator de **referenţiere**	*	Definirea de tipuri de dată pointer	tip* int* float* int** void*
Operator de **referenţiere**	&	"Extrage" adresa unei variabile	&a
Operator de **dereferenţiere**	*	Accesează zona de memorie indicată de un pointer	*p

Declarare și inițializare

```
int* p, a, *q;    ⇔    int *p; int a; int *q;
p = &a;
q = p;
*q = 7;
```



pointer:	p	q
valoarea lui p:	p	
adresa lui p:	&p	
zona către care indică p:	*p	
valoarea către care indică p:	*p	

Atribuire

- Operatorul de atribuire **=**
- Numai între pointeri de acelaşi tip

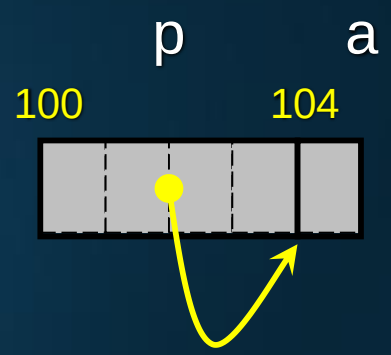
```
int *p, *q;
float *r, *s;

p = q;
r = s;
p = r;
s = q;
```

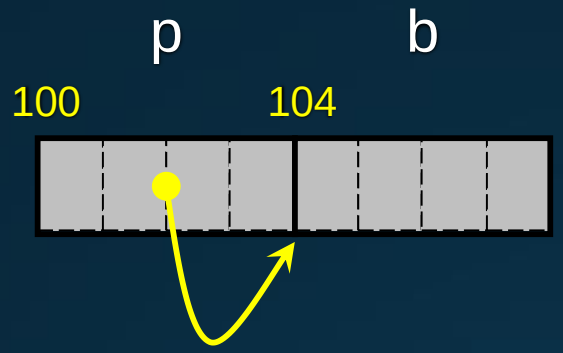
Operaţii aritmetice cu pointeri

- **Unitate**: dimensiunea zonei de memorie către care indică pointerul (depinde de tip)
  - noţiunea e definită doar pentru pointerii cu tip

```
char *p, a;
p = &a;
```



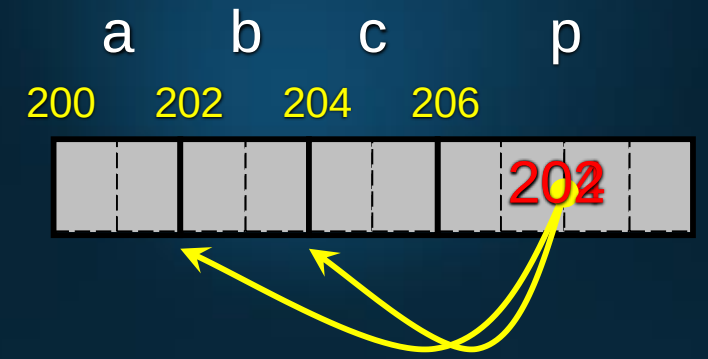

```
float *q, b;
q = &b;
```



Incrementare / decrementare

- Modificarea valorii operandului cu o **unitate**

```
short int a, b, c, *p;
p = &b;
p++;
p--;
```



Incrementare / decrementare

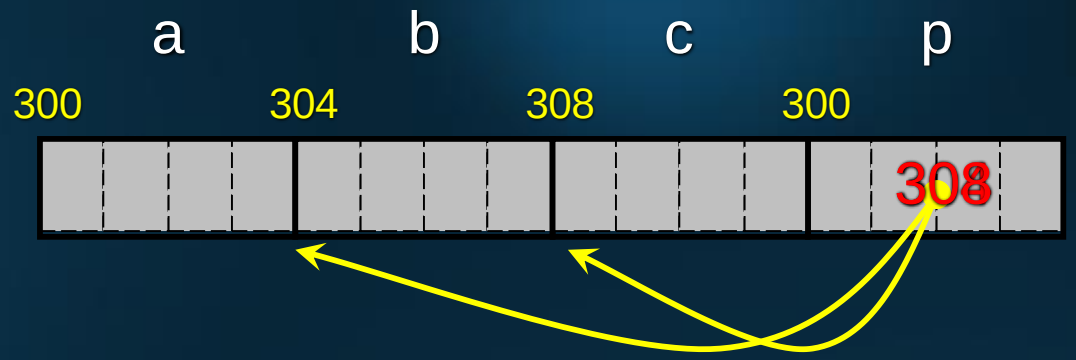
- Modificarea valorii operandului cu o **unitate**

float a, b, c, *p;

p = &b;

p++;

p--;



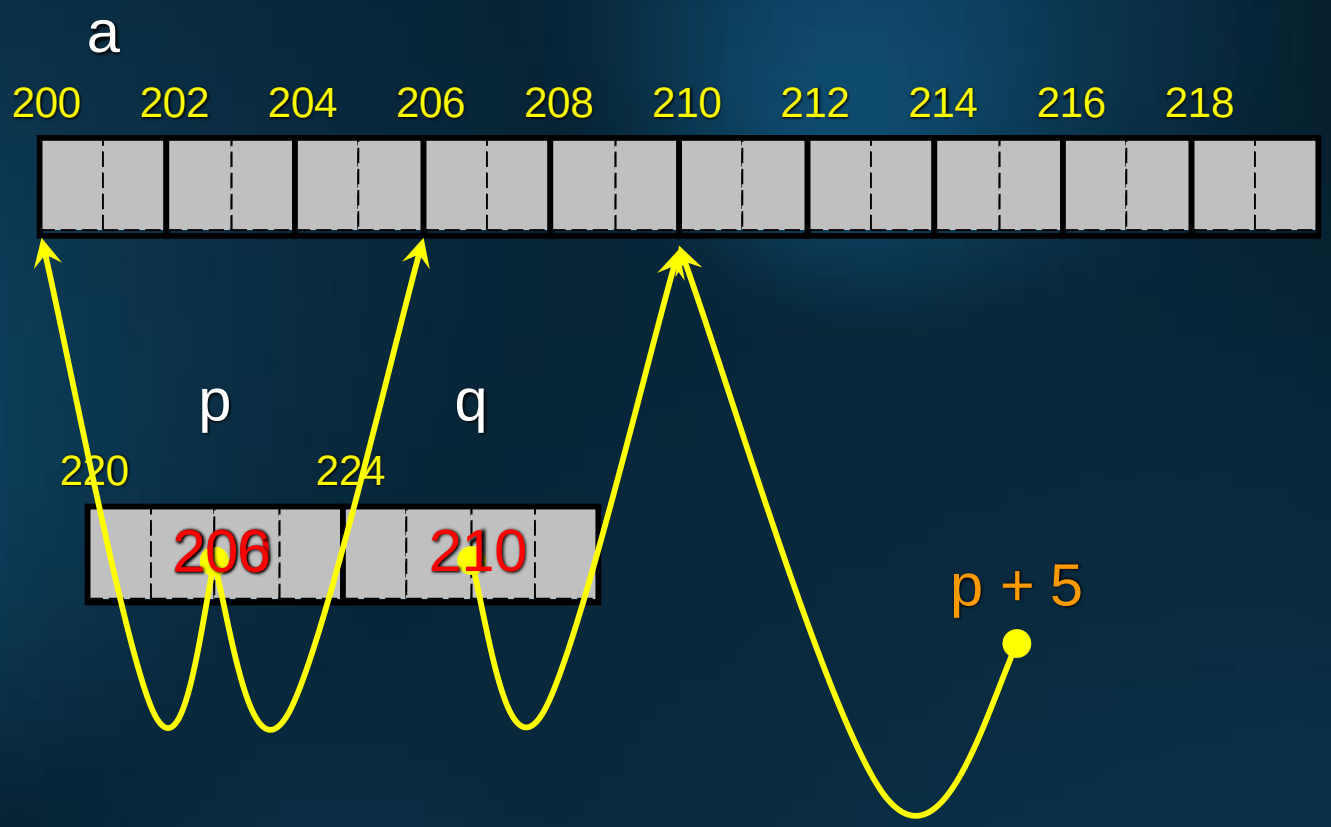
Adunarea / scăderea unui număr întreg

```
short int a[10], *p, *q;
p = &a[0];
```

p + 5

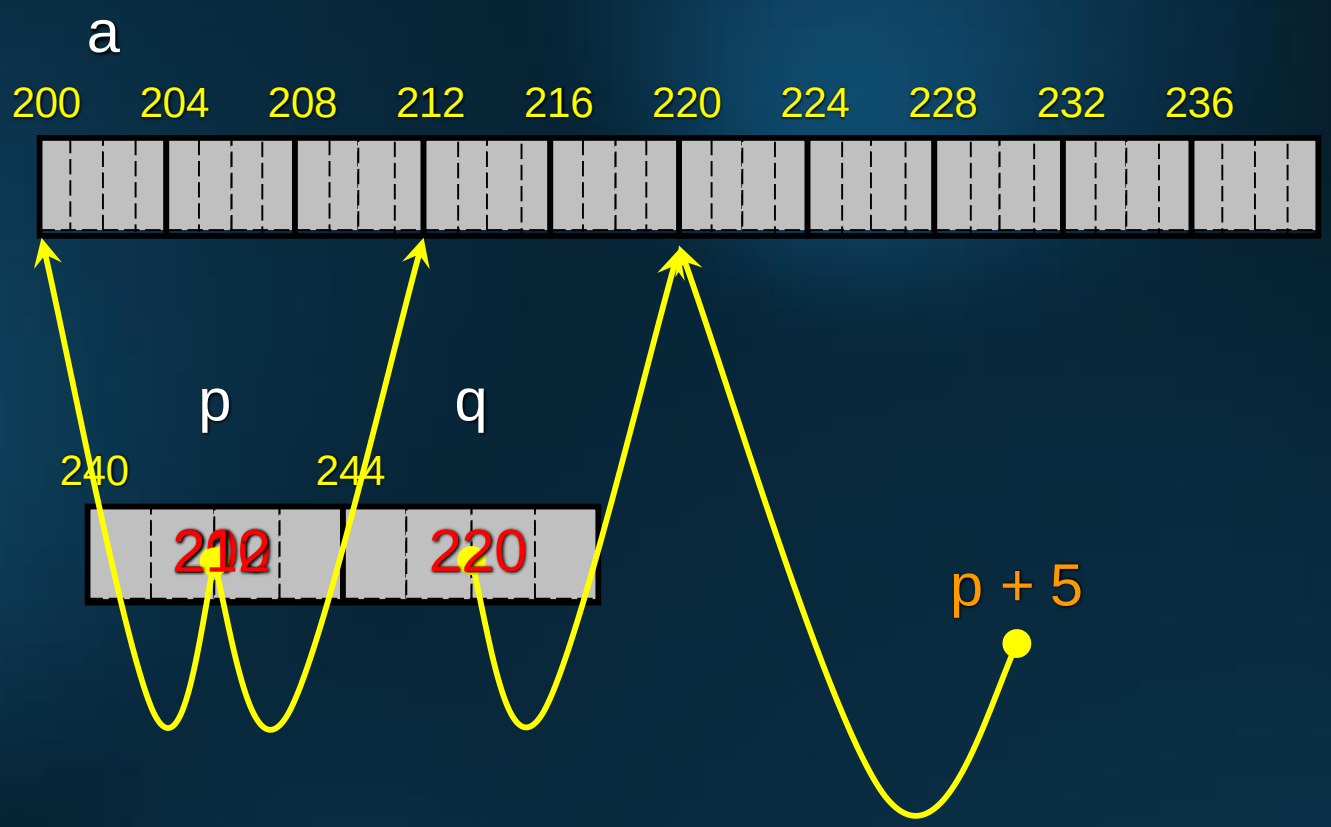
q = p + 5;

p = q - 2;



Adunarea / scăderea unui număr întreg

```
float a[10], *p, *q;
p = &a[0];
p + 5
q = p + 5;
p = q - 2;
```



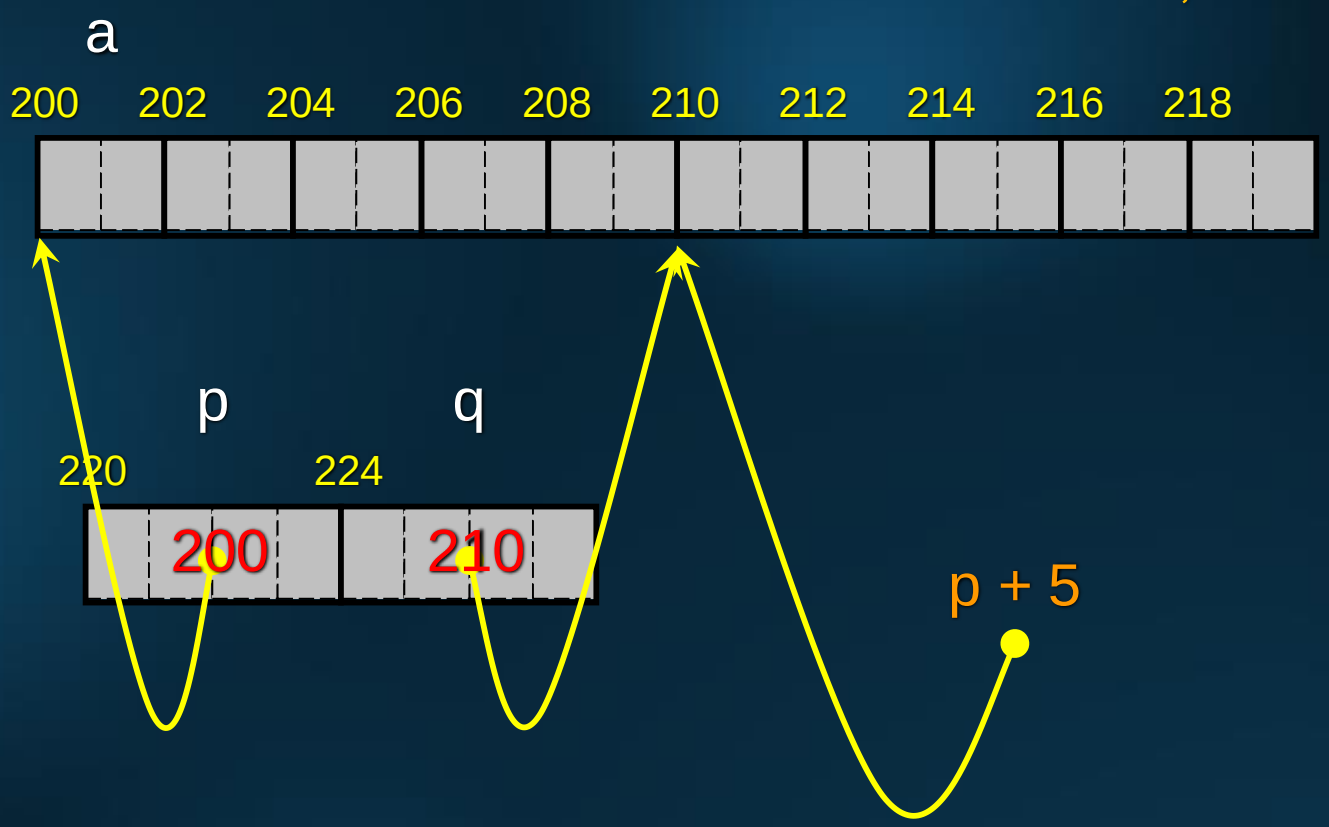
Diferența dintre doi pointeri

```
short int a[10], *p, *q;
p = &a[0];
```

diferența dintre *q* și *p*

q – p = 5

unități



Operatori relaţionali

- Toţi operatorii relaţionali: <, >, <=, >=, ==, !=
- Valoarea nulă pentru pointeri (pointerul nul)

  NULL
- **Pointer nul ⇔ neiniţializat**
  - **Atenţie: nu se poate dereferenţia un pointer nul!**

```
int *p;
…
if( p == NULL ) //nul, nu se poate lucra cu el
else //nenul, poate fi accesată zona indicată de el

if( !p )
```

Alocarea dinamică a memoriei

- Se include

  malloc.h (Visual C)

- Funcţii importante

  - malloc

  ```
  tip *p;
  p = (tip*) malloc(dimensiune);
  ```

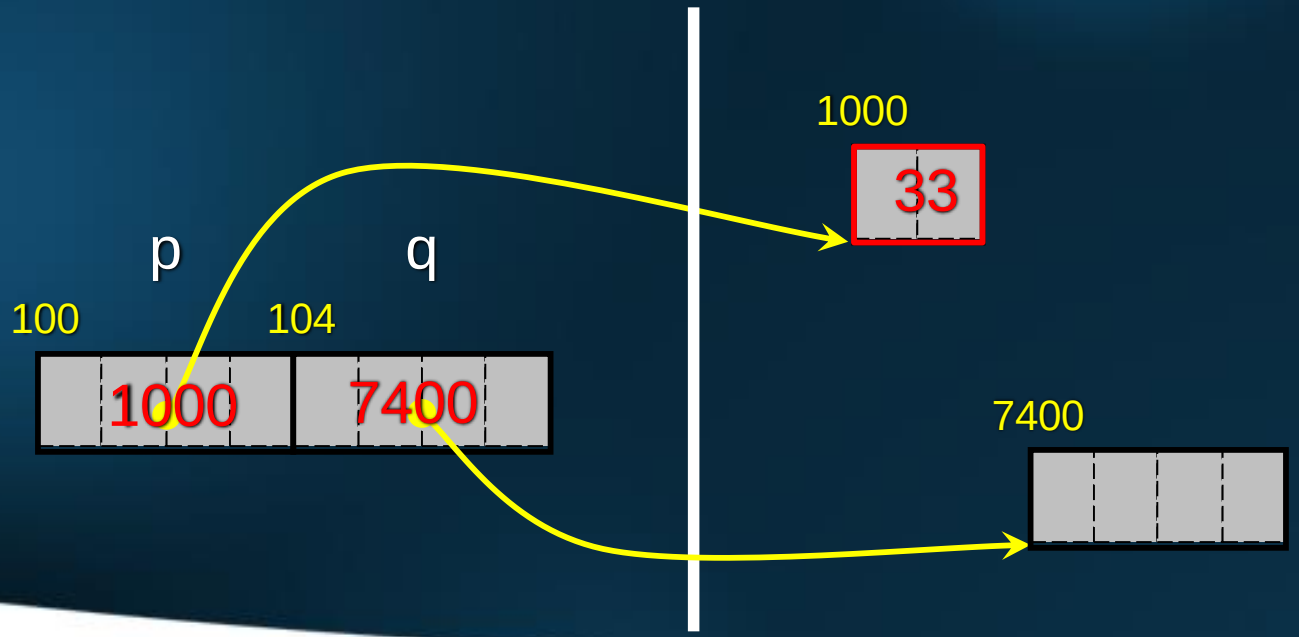
  - free

  ```
  free(p);
  ```

Alocarea dinamică a memoriei

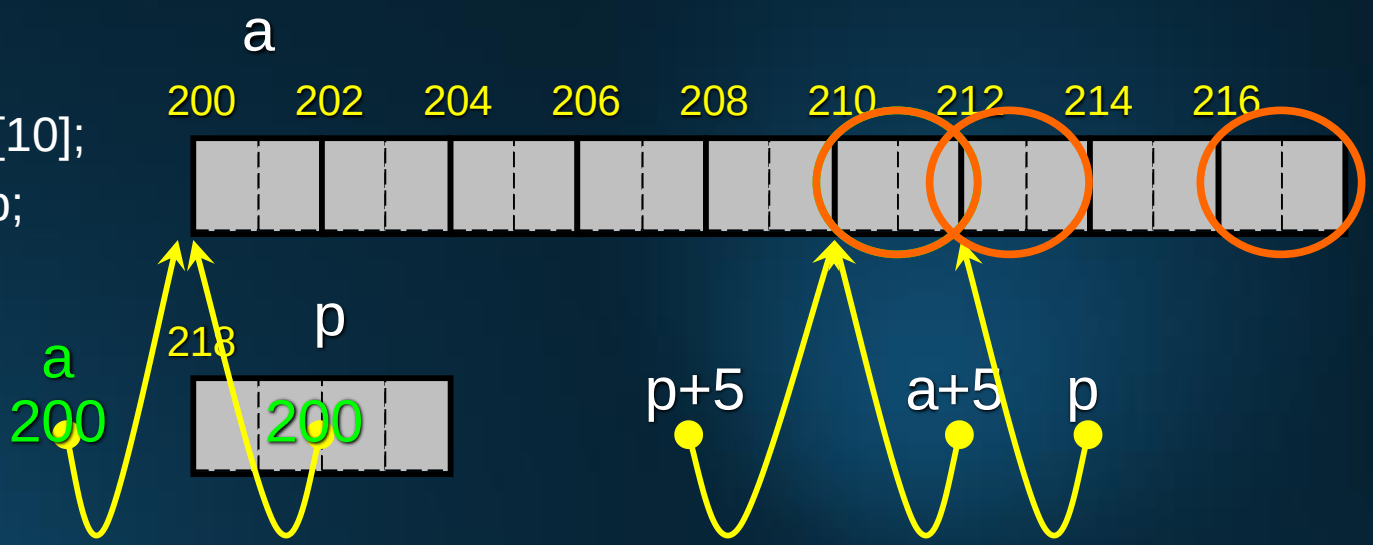
```
short int *p;
float *q;
p = ( short int* ) malloc (sizeof(short int));
*p = 33;
q = ( float* ) malloc (sizeof(float));
free(p);
```

SD / SS **Heap**



Legătura dintre vectori şi pointeri

short int a[10];
short int *p;



p = a;

~~a = p;~~

`*(a+5)` ⇔ `a[5]`

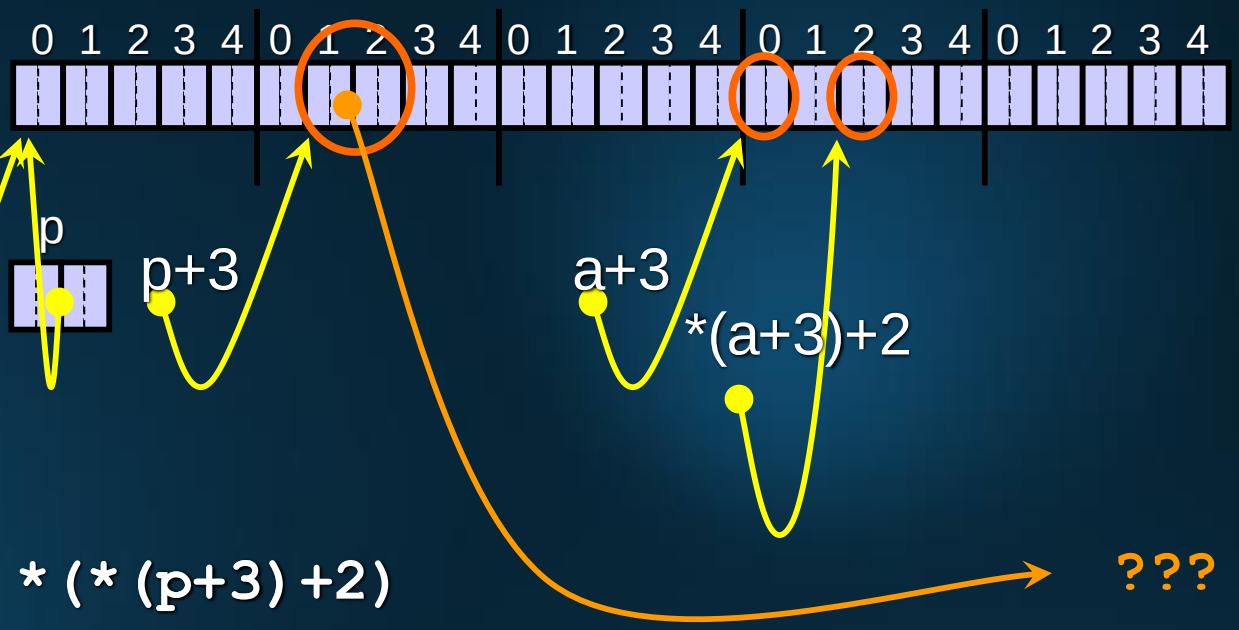
`*(p+5)` ⇔ `p[5]`

p = a + 6;

`p[0]` `p[2]`

Legătura dintre matrice şi pointeri

short int a[5][5];
short int **p;



p = a;

`p[3][2]` `*(*(p+3)+2)` ???

`a[3][2]` `*(*(a+3)+2)` `*(*(a+3))` ⇔ `a[3][0]`

Vectori alocaţi dinamic

Vector static

- Declarare

  tip p[dim]; int n,i;
- Alocare

  -
- Utilizare

  p[i]
- Eliberare (dezalocare)

  -

Vector dinamic

- Declarare

  tip* p; int n,i;
- Alocare

  p = (tip*) malloc(n * sizeof(tip));
- Utilizare

  p[i]
- Eliberare (dezalocare)

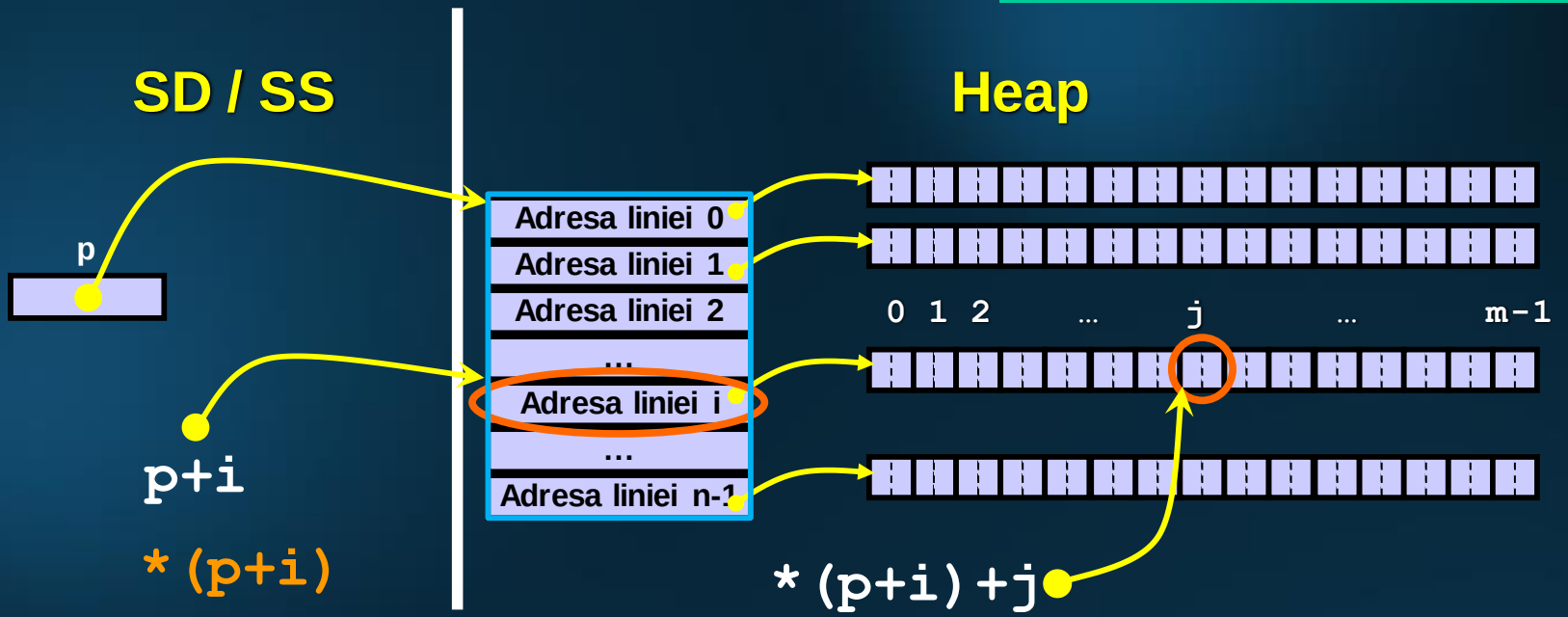
  free(p);

Matrice alocate dinamic

```
tip **p;      //*(p+i)  -> adresa liniei cu indicele i (i=0..n-1)

p = (tip**) malloc(n*sizeof(tip
for(i=0;i<n;i++)
  p[i] = (tip*) malloc(m*sizeof
```

```
for(i=0;i<n;i++)
   free(p[i]);
free(p);
```



Cum se accesează [i][j]?

((p+i)+j) ⇔ p[i][j]


Spor la învăţat!