



LABORATORIS

PROGRAMACIÓ CIENTÍFICA

Sessió 10: Utilitats
avançades

GENERACIÓ DE

NOMBRES ALEATORIS

Generació de nombres aleatoris

Nombres pseudo-aleatoris

- Molt sovint haurem de generar nombres aleatoris: llençar una moneda, llençar un dau, reordenar aleatòriament un conjunt de dades, generar nombres aleatoris d'una distribució, simular processos estocàstics...
- Com es generen els nombres aleatoris?
 - **Mètode 1:**
 - Mesurem algun fenòmen físic que és aleatori (o impossible de modelar) i els fem servir d'entrada a una funció que generarà nombres aleatoris.
 - **Mètode 2:**
 - Fer servir algorismes computacionals que són capaços de produir seqüències molt llargues de nombres aparentment aleatoris, però que en el fons estan determinats per un valor inicial concret (la llavor o seed).
 - Per tant, tota la seqüència de nombres aleatoris es pot reproduir si coneixem la llavor.
 - Això genera nombres pseudo-aleatoris.
 - La llavor acostuma a ser l'hora del sistema.

Generació de nombres aleatoris

Llibreria `<stdlib.h>`

- Per generar nombres pseudo-aleatoris en C, farem servir la funció **`srand()`**, disponible a la llibreria **`stdlib.h`**

```
int rand(void);
```

Retorna un valor pseudo-aleatori enter entre `0` i `RAND_MAX` (0 i `RAND_MAX` inclosos).

`RAND_MAX`

Valor enter constant que depèn de la implementació. Es garanteix que aquest valor serà almenys `32767`.

```
void srand (unsigned seed);
```

`srand()` configura la llavor (seed) del generador de nombres aleatoris que utilitza `rand()`.

Només s'ha d'executar un cop, al principi del programa.

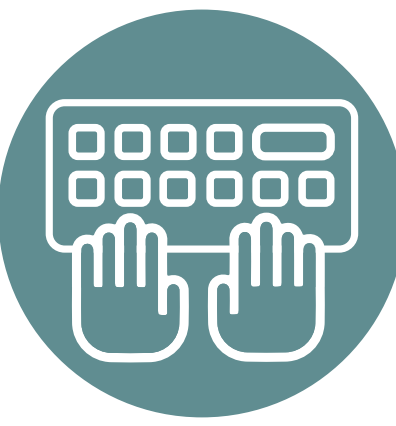
Si `rand()` és cridat abans de cridar `srand()`, `rand()` es comporta com si tingués llavor 1: `srand(1)`.

Si la llavor és constant (e.g. `srand(123)`) la seqüència de valors aleatoris sempre serà la mateixa en execucions diferents

El més comú és fer servir l'hora del sistema com a llavor. `srand(time(NULL));`

Per fer-ho, necessitem cridar la funció **`time()`**, que està disponible a la llibreria **`<time.h>`**

Generació de nombres aleatoris



Exemple


- Com es genera un nombre aleatori.
- Escriviu aquest programa i comproveu que dona valors aleatoris.
- Què passa si l'executeu molts cops seguits sense que passi massa temps entre ells?

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 int main(void)
6 {
7     srand(time(NULL)); // use current time as seed for random generator
8     int aleatori = rand();
9     printf("Valor aleatori value en rang [0,%d]: %d\n", RAND_MAX, aleatori);
10 }
```

Generació de nombres aleatoris

Exemple

- Com es genera un valor aleatori entre $[0, 1]$?



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 int main(void)
6 {
7     srand(time(NULL)); // use current time as seed for random generator
8
9     int aleatori;
10    float aleatori_0_1;
11
12    aleatori = rand();
13    aleatori_0_1 = (float) aleatori / RAND_MAX;
14
15    printf("El nombre aleatori en rang [0,1] es %.4f\n", aleatori_0_1);
16 }
```

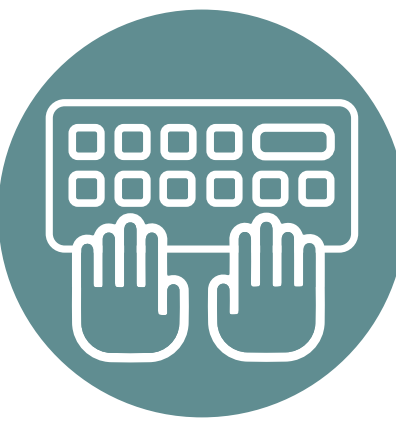
Generació de nombres aleatoris

Exemple

- Com es genera un valor aleatori entre [MIN, MAX] ?

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 #define MIN 1
6 #define MAX 6
7
8 int main(void)
9 {
10     srand(time(NULL)); // use current time as seed for random generator
11
12     int aleatori;
13     float aleatori_0_1;
14     int aleatori_min_max;
15
16     aleatori = rand();
17     aleatori_0_1 = (float) aleatori / RAND_MAX;
18     aleatori_min_max = aleatori_0_1 * (MAX - MIN + 1) + MIN;
19
20     printf("El nombre aleatori en rang [%d,%d] es %d\n", MIN, MAX, aleatori_min_max);
21 }
```


Generació de nombres aleatoris



Exemple

- Com es genera un valor aleatori **enter** entre [MIN, MAX] ?

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 /* Exemple per llençar un dau amb el modul */
6 #define MOD 6
7
8 int main(void)
9 {
10     srand(time(NULL)); // use current time as seed for random generator
11
12     int aleatori;
13     aleatori = rand() % MOD + 1;
14     printf("He tirat un dau i ha sortit %d\n", aleatori);
15
16 }
```


TAULES I CADENES 2D

NIVELL PRO

Taules de 2D en temps d'execució

Com llegir de fitxer taules 2D

- Si hem de llegir una matriu de fitxer, necessitem reservar la memòria en temps d'execució.
- Ja sabem com fer-ho per taules unidimensionals:

```
/* Pel cas d'enters: */  
int * v;  
// Esbrinar nombre d'elements  
v = (int *) malloc (nombre_elems * sizeof(int))
```

Taules de 2D en temps d'execució

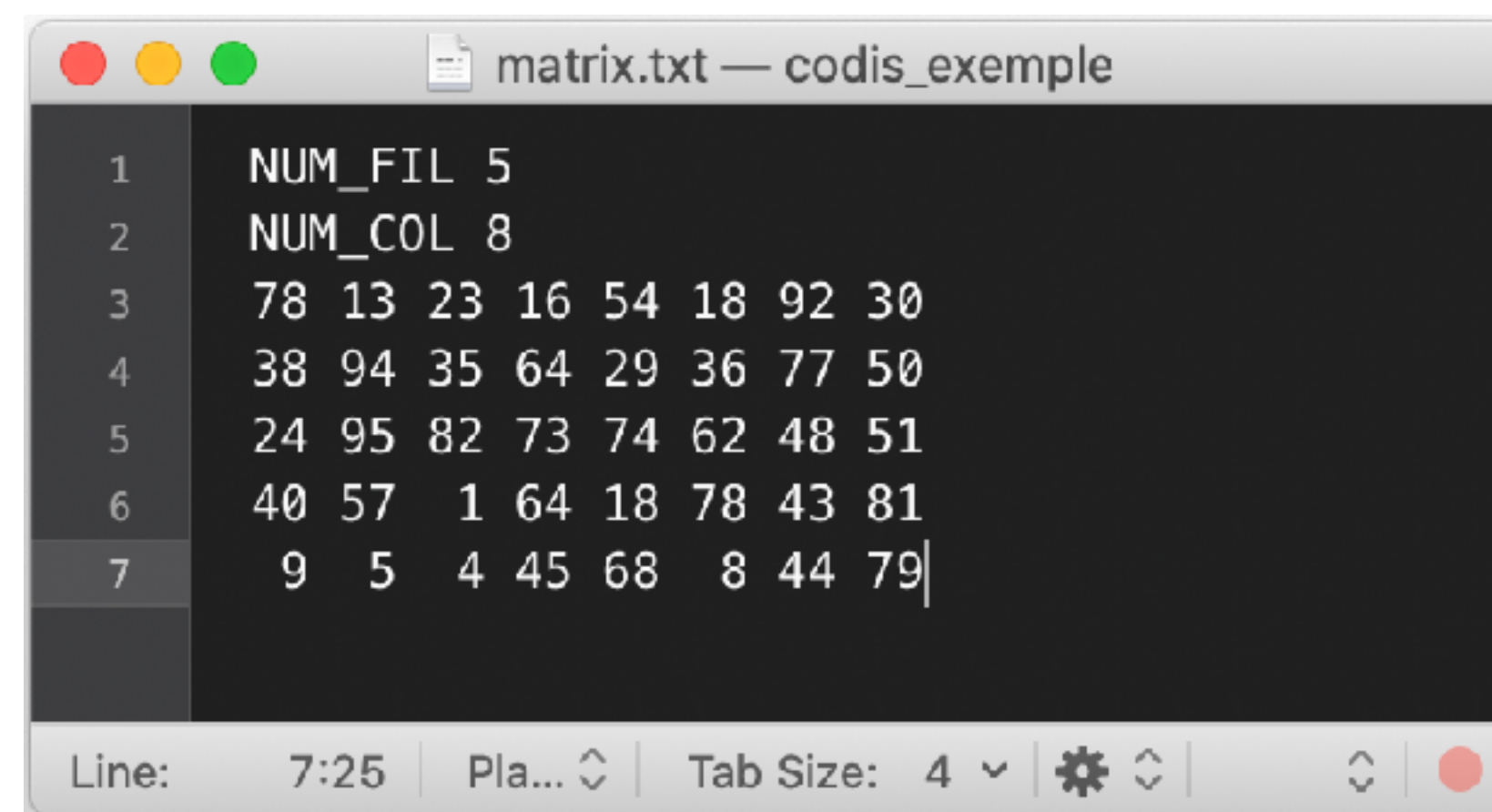
Com llegir de fitxer taules 2D

- Si hem de llegir una matriu de fitxer, necessitem reservar la memòria en temps d'execució.
- Ja sabem com fer-ho per taules unidimensionals:

```
/* Pel cas d'enters: */  
int * v;  
// Esbrinar nombre d'elements  
v = (int *) malloc (nombre_elems * sizeof(int))
```

- Com ho fem si volem llegir un arxiu com aquest?

matrix.txt



```
1 NUM_FIL 5  
2 NUM_COL 8  
3 78 13 23 16 54 18 92 30  
4 38 94 35 64 29 36 77 50  
5 24 95 82 73 74 62 48 51  
6 40 57 1 64 18 78 43 81  
7 9 5 4 45 68 8 44 79
```


Taules de 2D en temps d'execució

Opció 1: Reservar un únic bloc de memòria i calcular la posició a cada accés

- Recordem què és una matriu en C:



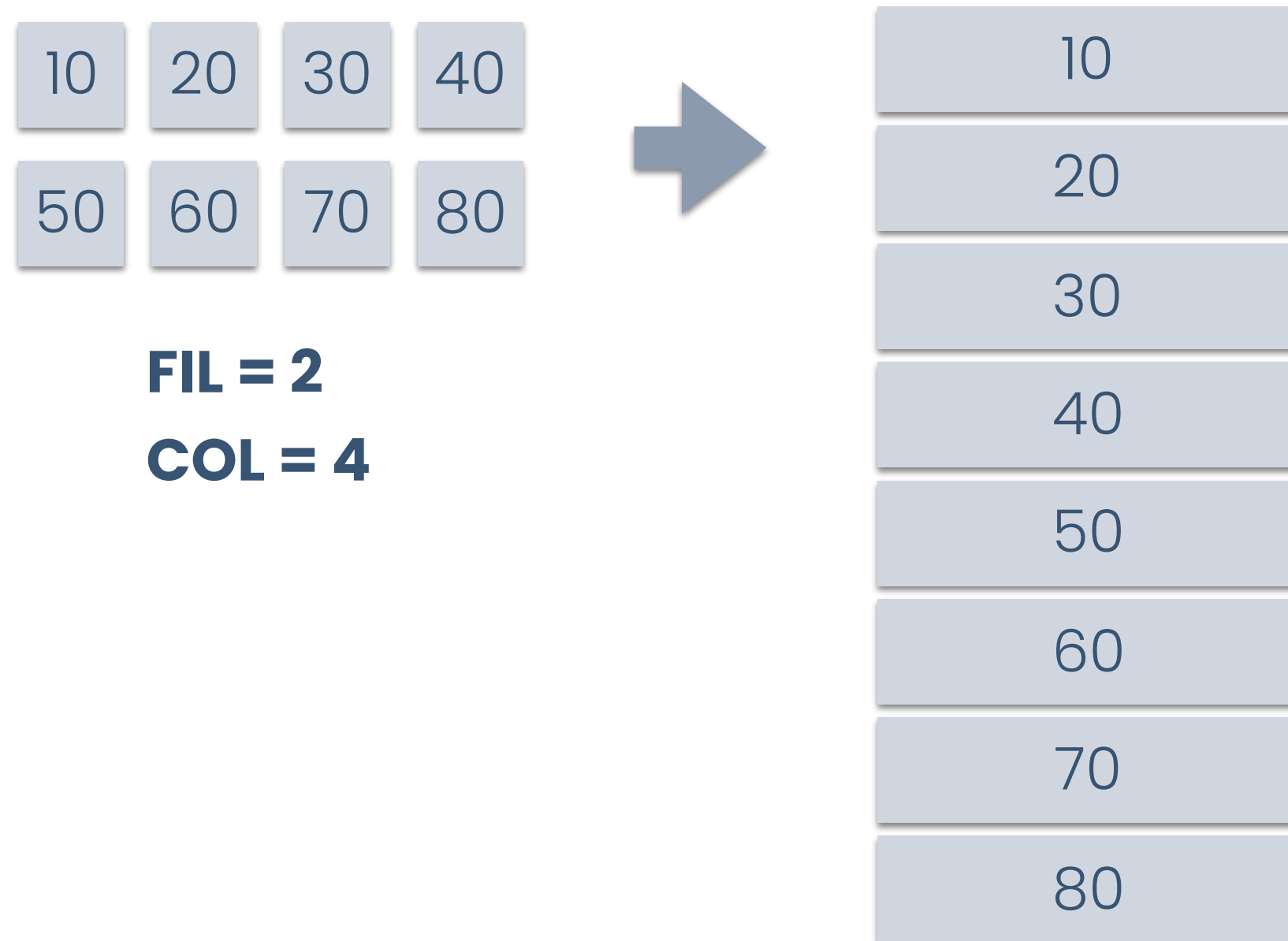
FIL = 2

COL = 4

Taules de 2D en temps d'execució

Opció 1: Reservar un únic bloc de memòria i calcular la posició a cada accés

- Recordem què és una matriu en C:



Taules de 2D en temps d'execució

Opció 1: Reservar un únic bloc de memòria i calcular la posició a cada accés

- Recordem què és una matriu en C:

10	20	30	40
50	60	70	80

FIL = 2

COL = 4



10
20
30
40
50
60
70
80

← **v[0][0]** (i=0, j=0)

← **v[0][1]** (i=0, j=1)

← **v[0][2]** (i=0, j=2)

← **v[0][3]** (i=0, j=3)

← **v[1][0]** (i=1, j=0)

← **v[1][1]** (i=1, j=1)

← **v[1][2]** (i=1, j=2)

← **v[FIL-1][COL-1]**

Taules de 2D en temps d'execució

Opció 1: Reservar un únic bloc de memòria i calcular la posició a cada accés

- Recordem què és una matriu en C:

10	20	30	40
50	60	70	80

FIL = 2
COL = 4



10
20
30
40
50
60
70
80

← **v[0][0]** (i=0, j=0)
← **v[0][1]** (i=0, j=1)
← **v[0][2]** (i=0, j=2)
← **v[0][3]** (i=0, j=3)
← **v[1][0]** (i=1, j=0)
← **v[1][1]** (i=1, j=1)
← **v[1][2]** (i=1, j=2)
← **v[FIL-1][COL-1]**

Si ho tracto com a vector...

v[0] (Posició 0)
v[1] (Posició 1)
v[2] (Posició 2)
v[3] (Posició 3)
v[4] (Posició 4)
v[5] (Posició 5)
v[6] (Posició 6)
v[7] (Posició 7)

Taules de 2D en temps d'execució

Opció 1: Reservar un únic bloc de memòria i calcular la posició a cada accés

- Recordem què és una matriu en C:

10	20	30	40
50	60	70	80

FIL = 2

COL = 4



10
20
30
40
50
60
70
80

← **v[0][0]** (i=0, j=0)

← **v[0][1]** (i=0, j=1)

← **v[0][2]** (i=0, j=2)

← **v[0][3]** (i=0, j=3)

← **v[1][0]** (i=1, j=0)

← **v[1][1]** (i=1, j=1)

← **v[1][2]** (i=1, j=2)

← **v[FIL-1][COL-1]**

Si ho tracto com a vector...

v[0] (Posició 0)

v[1] (Posició 1)

v[2] (Posició 2)

v[3] (Posició 3)

v[4] (Posició 4)

v[5] (Posició 5)

v[6] (Posició 6)

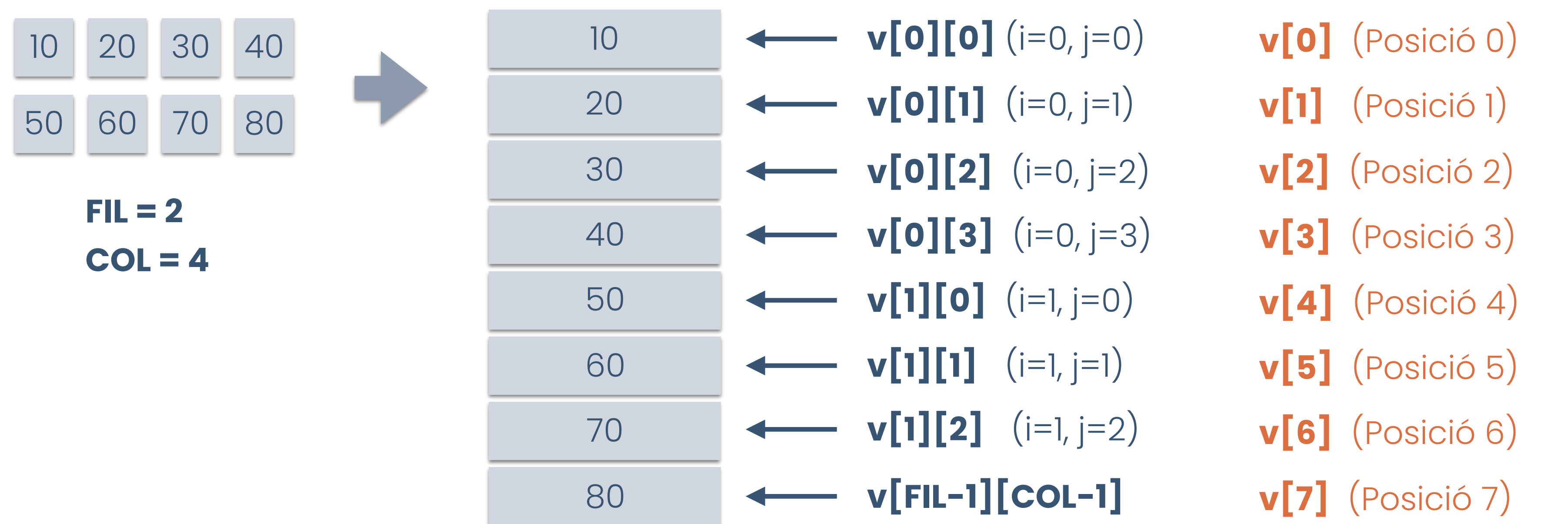
v[7] (Posició 7)

- Com hi podríem accedir si consideréssim que la nostra matriu és un vector?

Taules de 2D en temps d'execució

Opció 1: Reservar un únic bloc de memòria i calcular la posició a cada accés

- Recordem què és una matriu en C:



- Com hi podríem accedir si consideréssim que la nostra matriu és un vector?

v[i][j] → **v[COL*i + j]**

Taules de 2D en temps d'execució

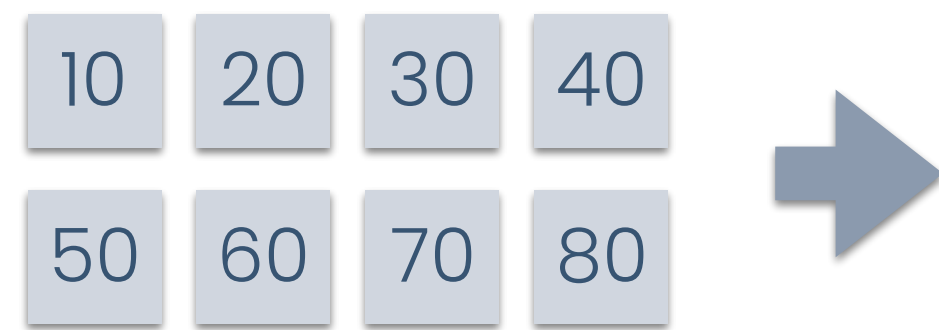
Opció 1: Reservar un únic bloc de memòria i calcular la posició a cada accés

```
1 ...
2  int * v;
3
4  // Obrir fitxer i comprovacions
5  // Llegir dues primeres línies i guardar a num_fil i num_col
6  fscanf(f_in, "%s%d", s, &num_fil);
7  fscanf(f_in, "%s%d", s, &num_col);
8
9
10 /* Alocatar un únic bloc de memòria*/
11 v = (int *) malloc (num_fil * num_col * sizeof(int));
12
13 /* Accés a la taula */
14 for(int i=0; i<num_fil; i++){
15     for(int j=0; j < num_col; j++){
16         fscanf(f_in, "%d", &num);
17         v[num_col*i+j] = num;
18     }
19 }
20 ...
```

Taules de 2D en temps d'execució

Opció 2: Declarar un vector (estàtic) de tipus punter a enter

- Aquesta aproximació requereix que sapiguem el nombre de files **NUM_FIL** en temps de compilació (a l'opció 3 ja veurem com podem fer-ho sense aquesta condició)
- La manera de procedir és declarar un vector en temps de compilació que guardi punters a enter, i al·locar memòria per cadascun d'aquests punters:



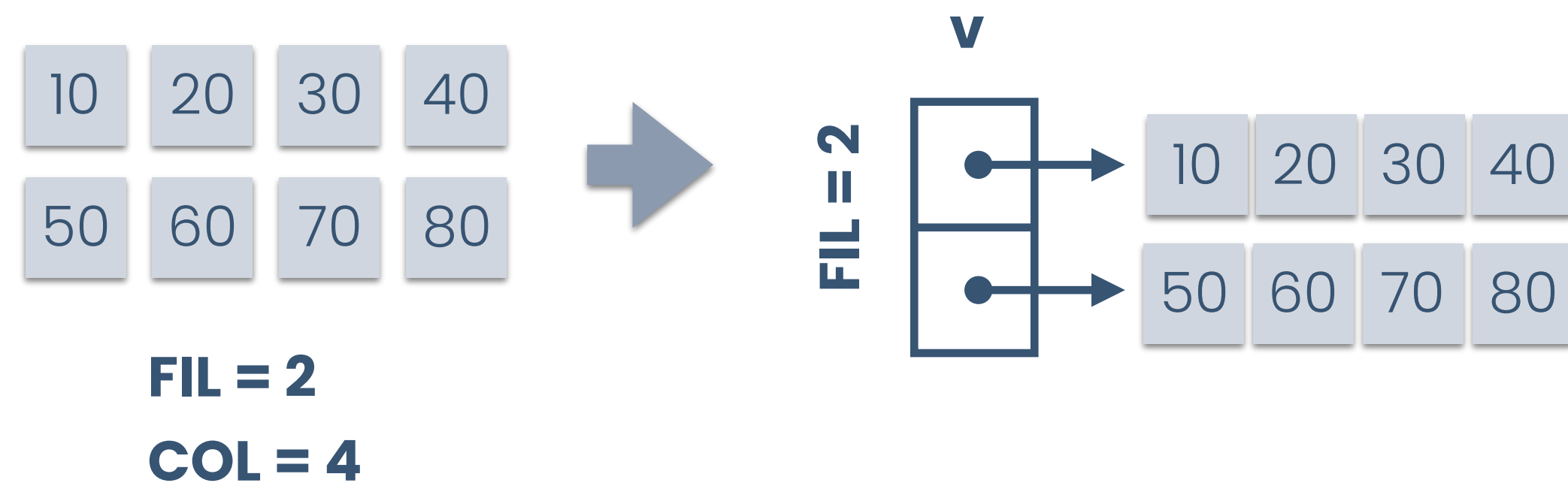
FIL = 2

COL = 4

Taules de 2D en temps d'execució

Opció 2: Declarar un vector (estàtic) de tipus punter a enter

- Aquesta aproximació requereix que sapiguem el nombre de files **NUM_FIL** en temps de compilació (a l'opció 3 ja veurem com podem fer-ho sense aquesta condició)
- La manera de procedir és declarar un vector en temps de compilació que guardi punters a enter, i al·locar memòria per cadascun d'aquests punters:



Taules de 2D en temps d'execució

Opció 2: Declarar un vector (estàtic) de tipus punter a enter

```
1 #define NUM_FIL 5
2 ...
3 /* Array de mida NUM_FIL de tipus punter a enter */
4 int * v[NUM_FIL];
5
6 // Obrir fitxer i comprovacions (...)
7 // Llegir dues primeres línies i guardar a num_fil i num_col
8 fscanf(f_in, "%s%d", s, &num_fil); // Aquesta no la fem servir
9 fscanf(f_in, "%s%d", s, &num_col);
10
11 /* Alocatar un bloc de memòria per cada vector*/
12 for(int i = 0; i<NUM_FIL; i++){
13     v[i] = (int *) malloc (num_col * sizeof(int));
14     /* Comprovació "hi ha memòria" */
15 }
16
17 /* Accés normal: */
18 for(int i=0; i < NUM_FIL; i++){
19     for(int j=0; j < num_col; j++){
20         fscanf(f_in, "%d", &num);
21         v[i][j] = num;
22     }
23 }
24 ...
```

Taules de 2D en temps d'execució

Opció 3: Declarar un vector (dinàmic) de tipus punter a enter

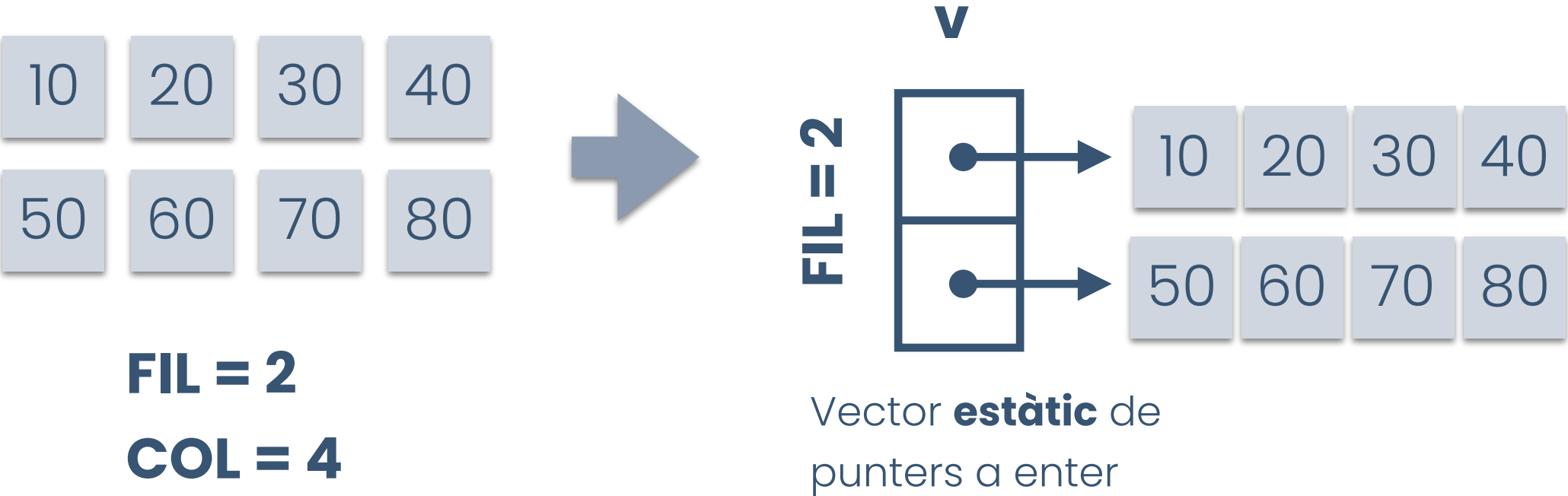
- Com que generalment no sabrem el nombre de files en temps de compilació, podem fer una aproximació semblant a l'anterior però on el **vector de punters no és estàtic**.

Taules de 2D en temps d'execució

Opció 3: Declarar un vector (dinàmic) de tipus punter a enter

- Com que generalment no sabrem el nombre de files en temps de compilació, podem fer una aproximació semblant a l'anterior però on el **vector de punters no és estàtic**.

A l'opció 2...

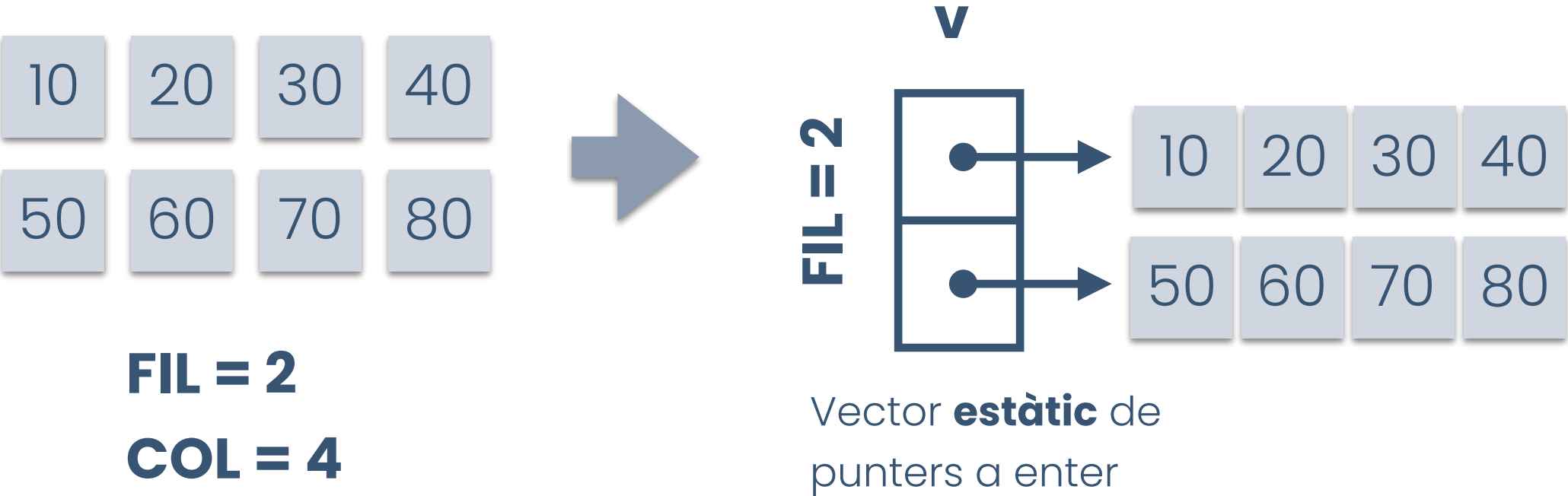


Taules de 2D en temps d'execució

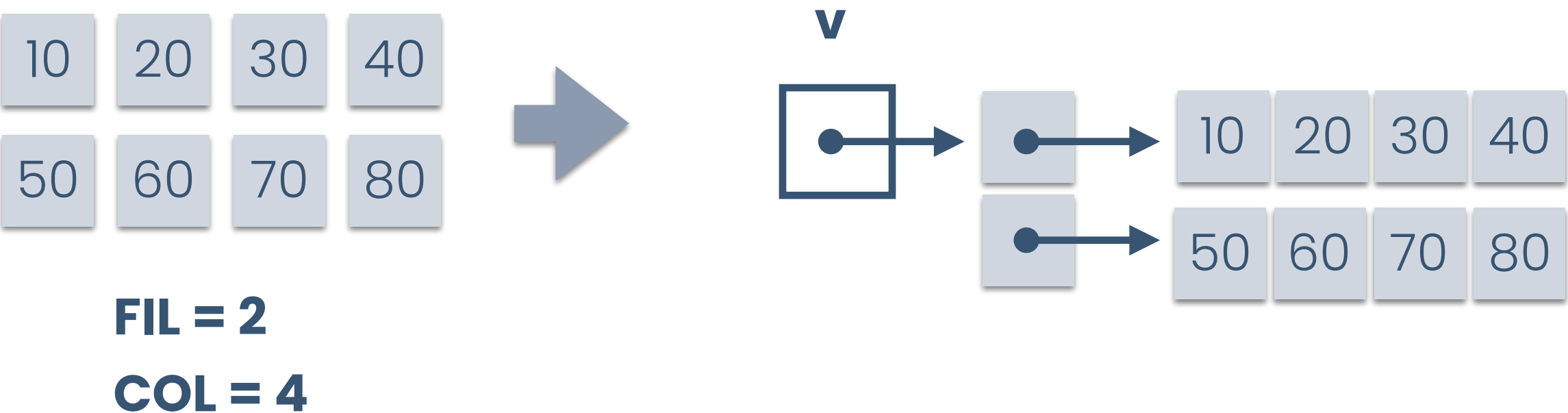
Opció 3: Declarar un vector (dinàmic) de tipus punter a enter

- Com que generalment no sabrem el nombre de files en temps de compilació, podem fer una aproximació semblant a l'anterior però on el **vector de punters no és estàtic**.

A l'opció 2...



A l'opció 3...

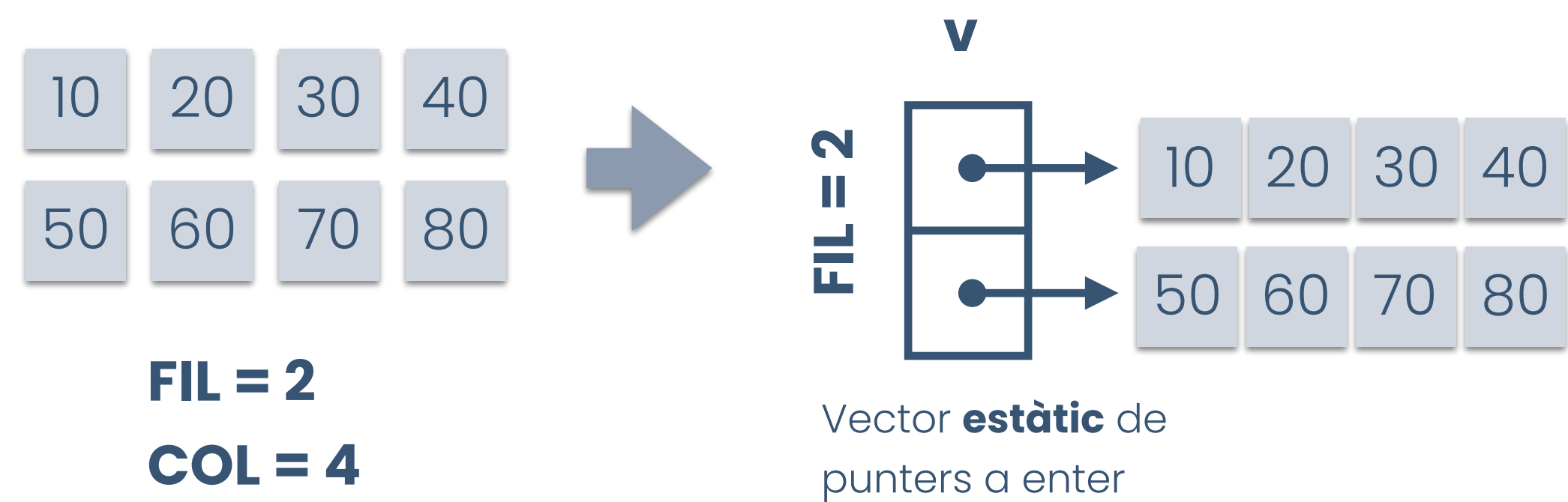


Taules de 2D en temps d'execució

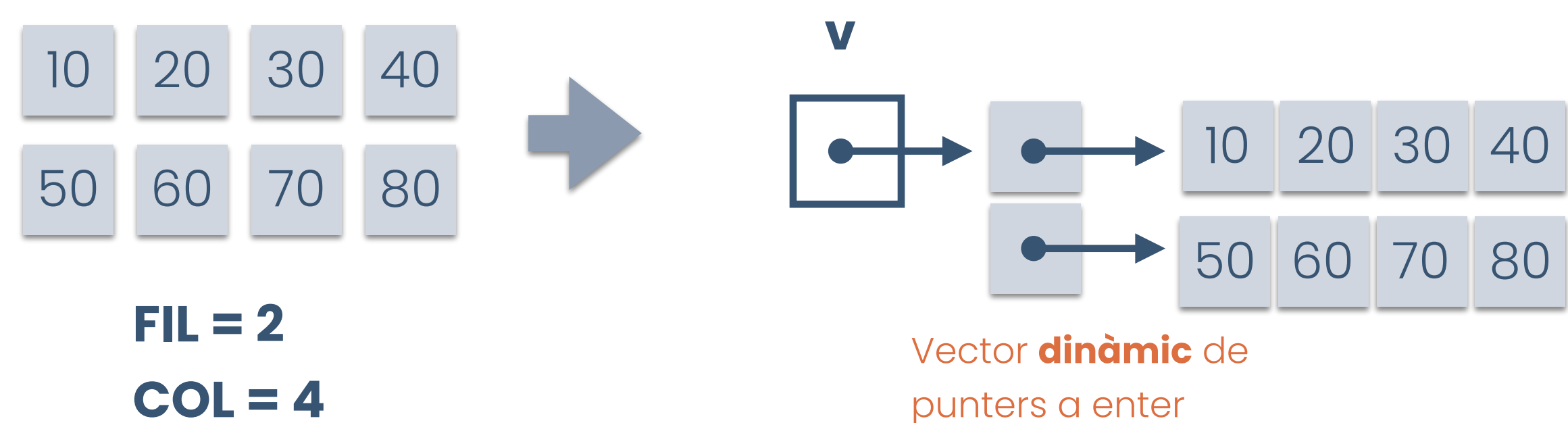
Opció 3: Declarar un vector (dinàmic) de tipus punter a enter

- Com que generalment no sabrem el nombre de files en temps de compilació, podem fer una aproximació semblant a l'anterior però on el **vector de punters no és estàtic**.

A l'opció 2...



A l'opció 3...

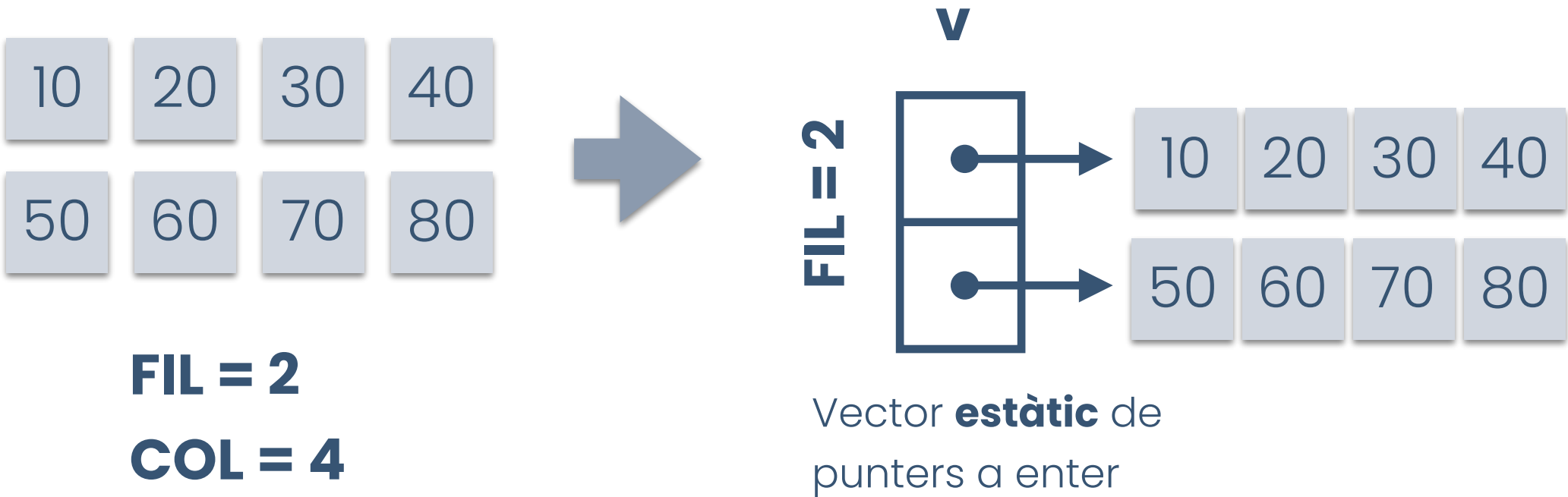


Taules de 2D en temps d'execució

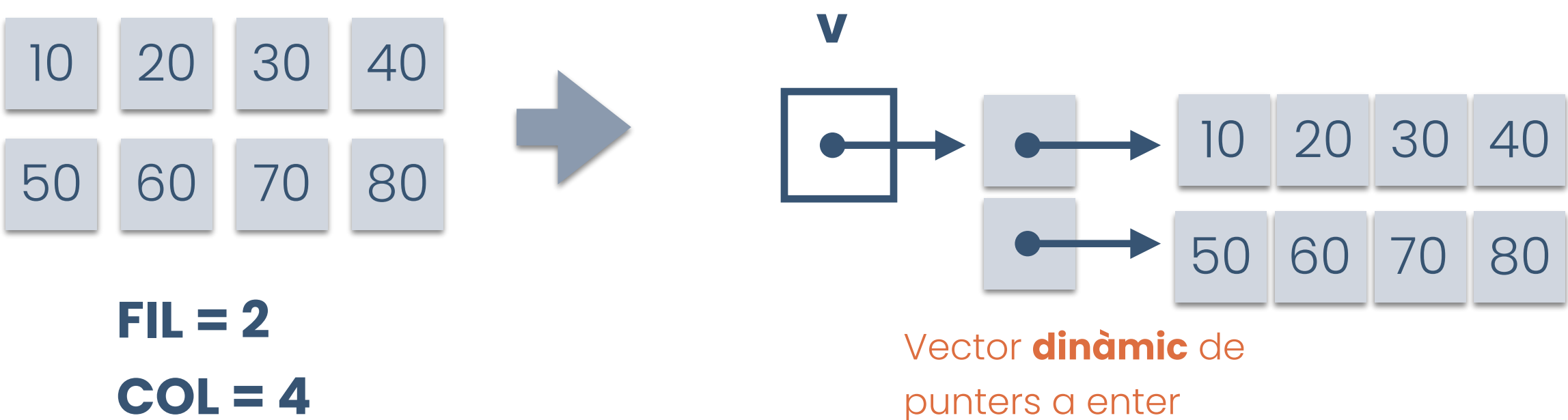
Opció 3: Declarar un vector (dinàmic) de tipus punter a enter

- Com que generalment no sabrem el nombre de files en temps de compilació, podem fer una aproximació semblant a l'anterior però on el **vector de punters no és estàtic**.

A l'opció 2...



A l'opció 3...



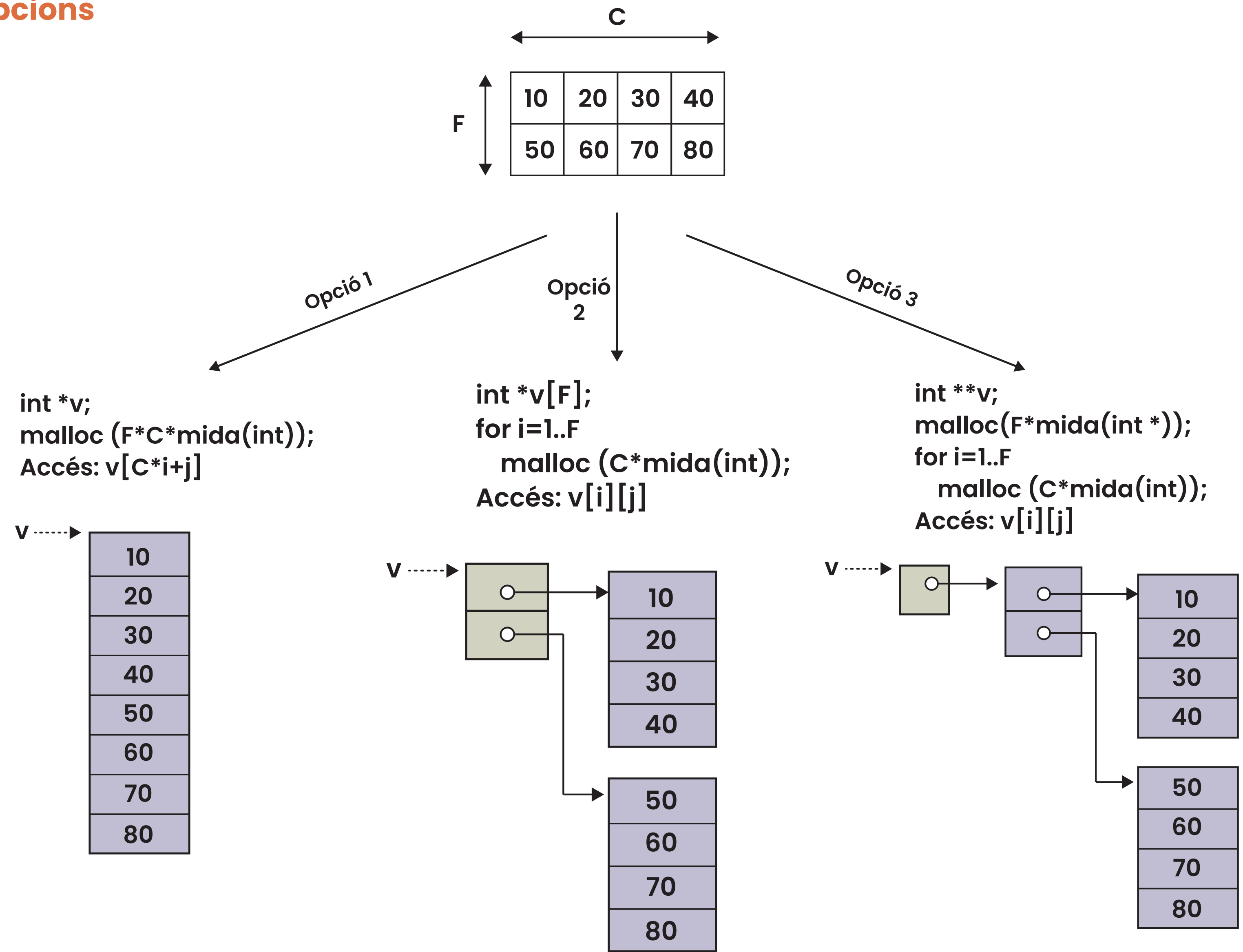
Taules de 2D en temps d'execució

**Opció 3: Declarar un vector
(dinàmic) de tipus punter a enter**

```
1 ...
2  /* Punter a punter a enter */
3  int ** v;
4  // Obrir fitxer i comprovacions (...)
5  // Llegir dues primeres línies i guardar a num_fil i num_col
6  fscanf(f_in, "%s%d", s, &num_fil);
7  fscanf(f_in, "%s%d", s, &num_col);
8
9  /* Alocatar memòria per la taula de punters a enter*/
10 v = (int**) malloc (num_fil * sizeof(int*));
11 // Comprovació memòria (...)
12 /* Alocatar memòria per les taules d'enters*/
13 for(int i = 0; i < num_fil; i++){
14     v[i] = (int*) malloc (num_col * sizeof(int));
15     // Comprovació memòria (...)
16 }
17
18 /* Accés normal: */
19 for(int i=0; i < num_fil; i++){
20     for(int j=0; j < num_col; j++){
21         fscanf(f_in, "%d", &num);
22         v[i][j] = num;
23     }
24 }
25 ...
```

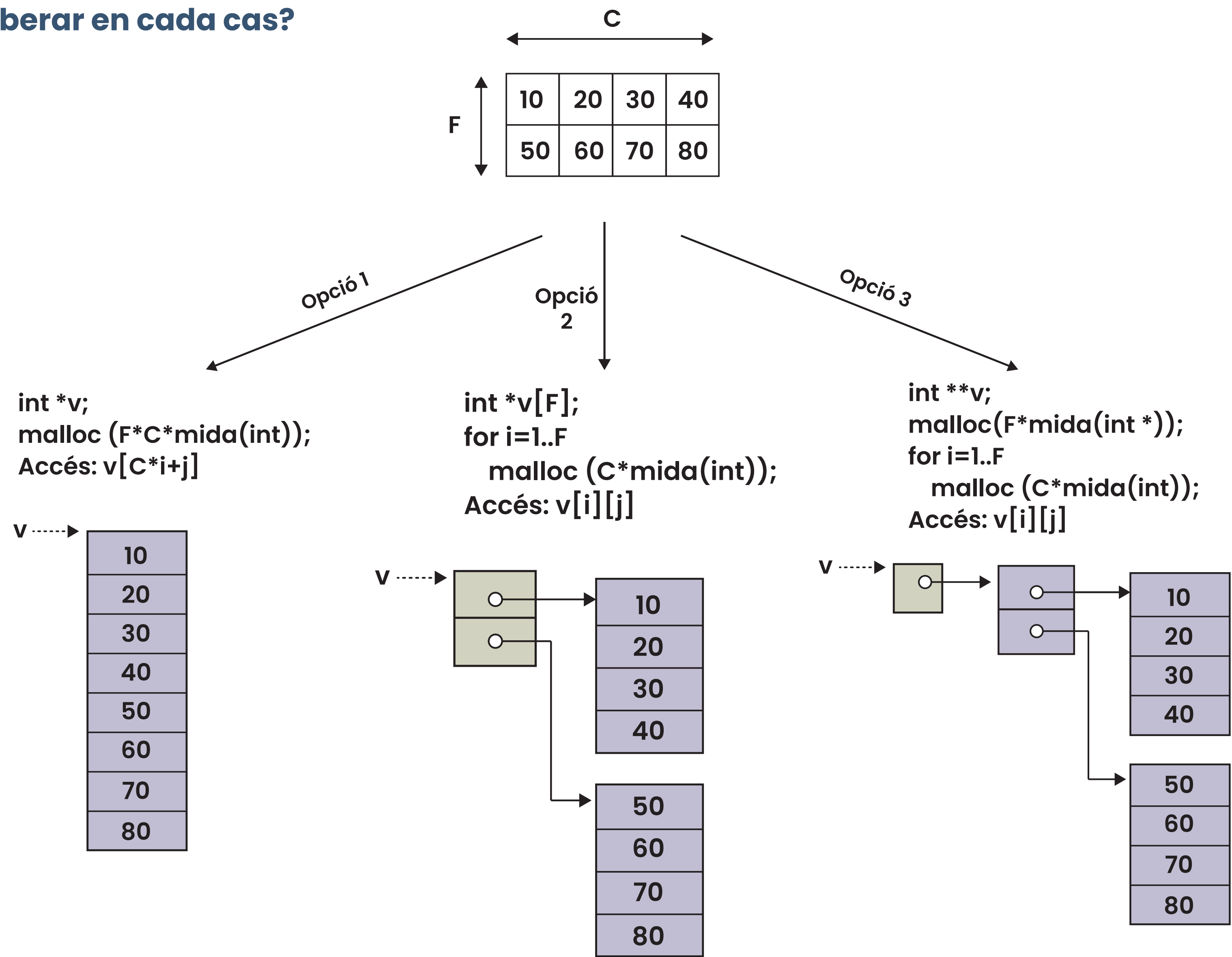
Taules de 2D en temps d'execució

Resum de les opcions



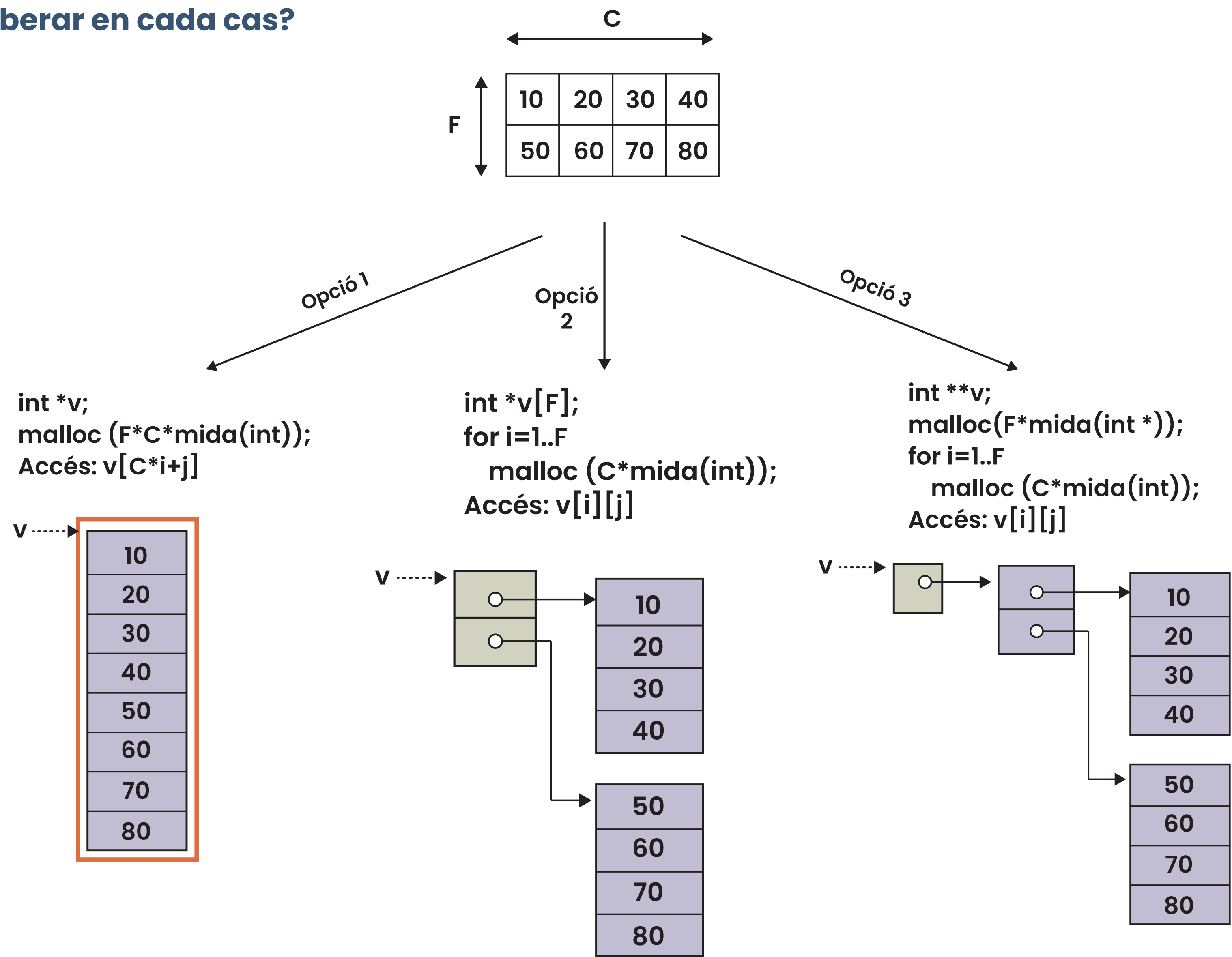
No ens hem oblidat d'alguna cosa?

Quina memòria hem d'alliberar en cada cas?



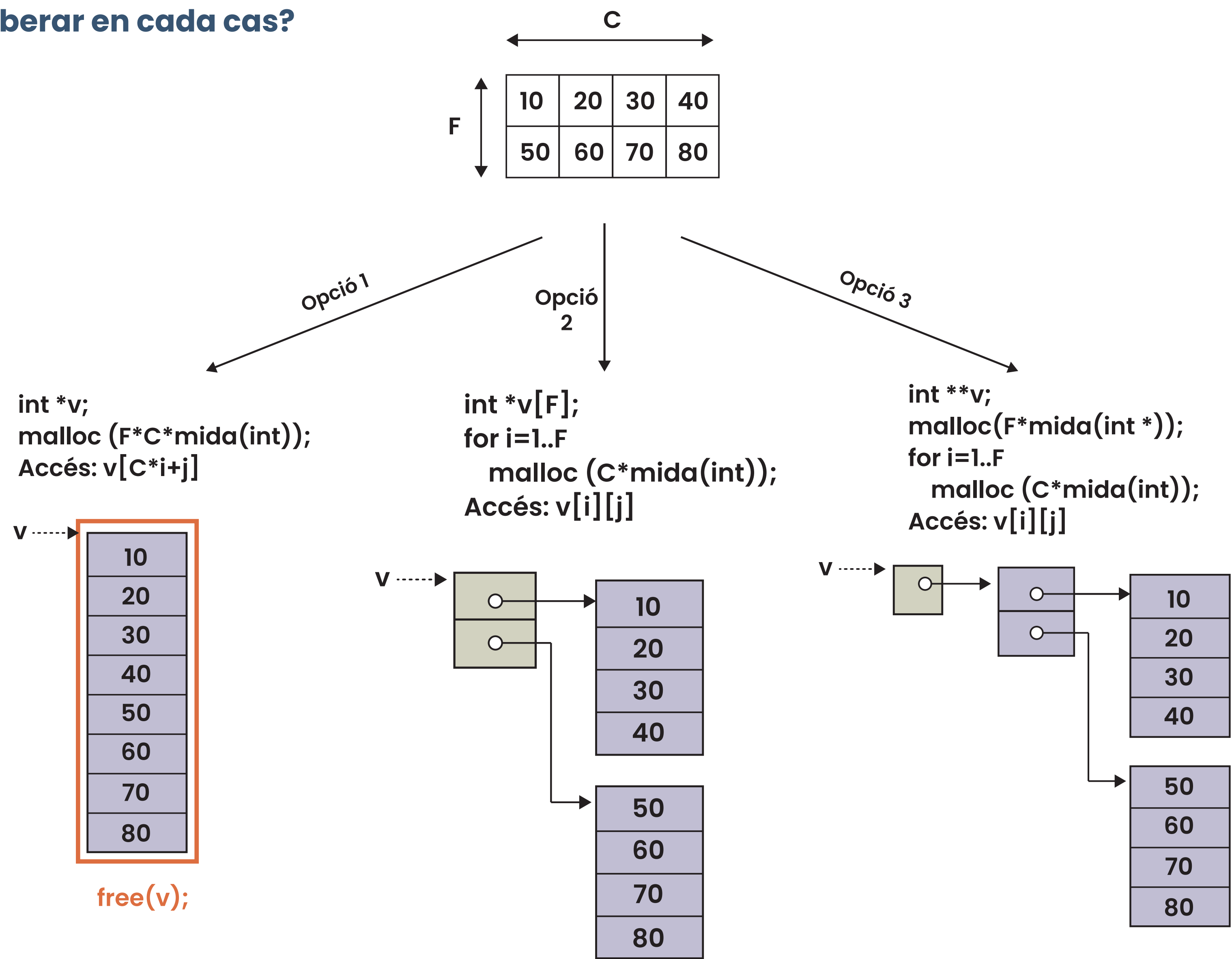
No ens hem oblidat d'alguna cosa?

Quina memòria hem d'alliberar en cada cas?



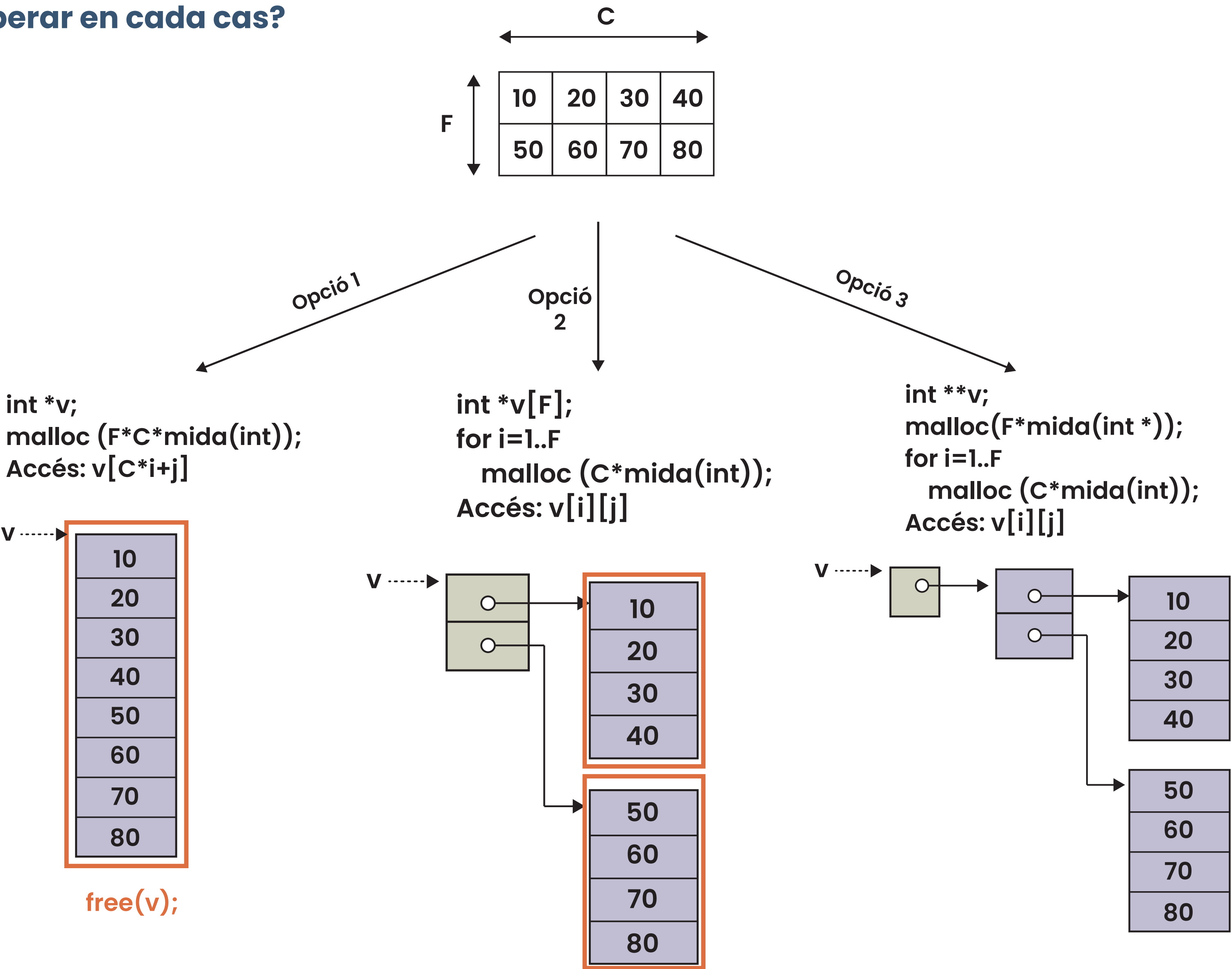
No ens hem oblidat d'alguna cosa?

Quina memòria hem d'alliberar en cada cas?



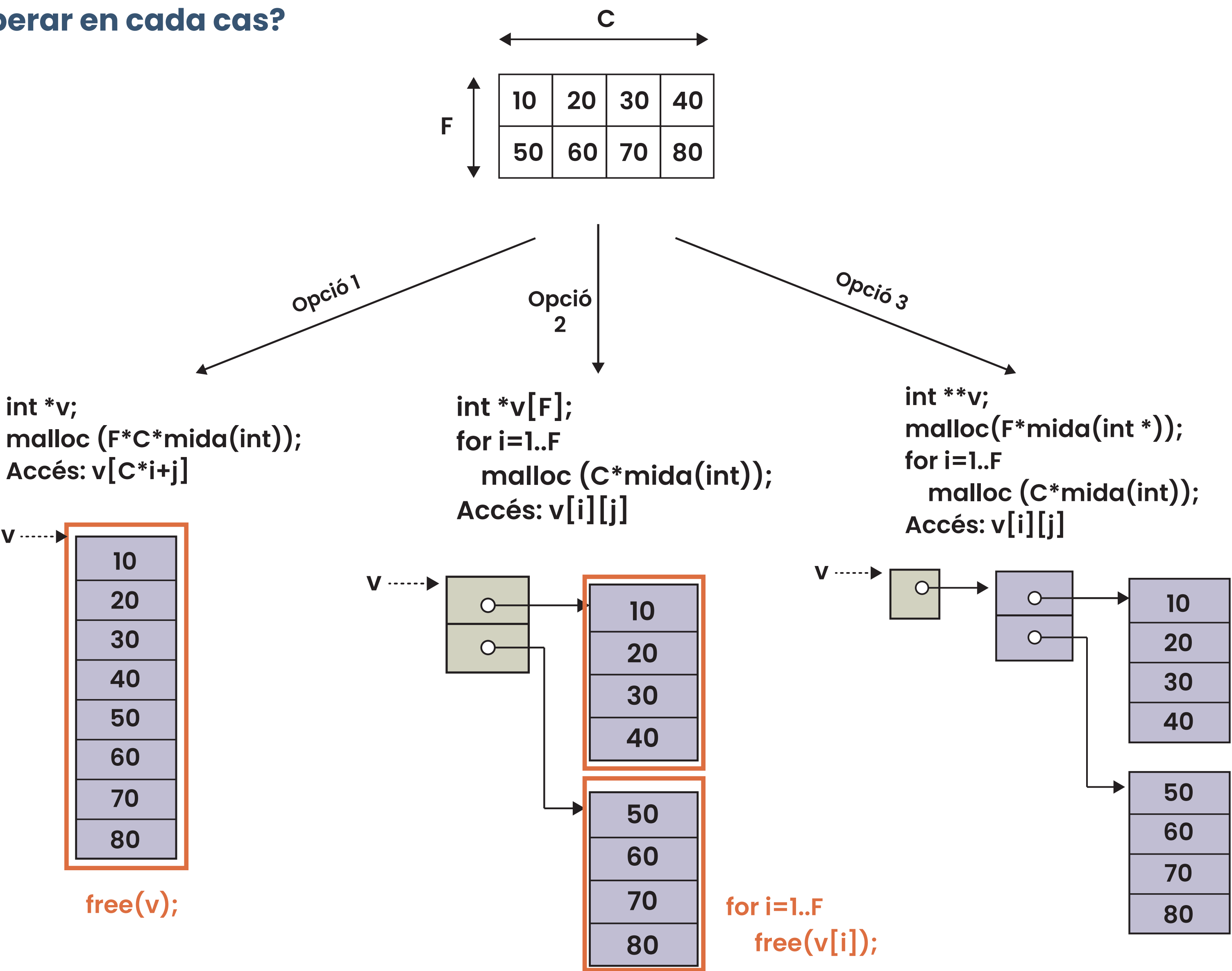
No ens hem oblidat d'alguna cosa?

Quina memòria hem d'alliberar en cada cas?



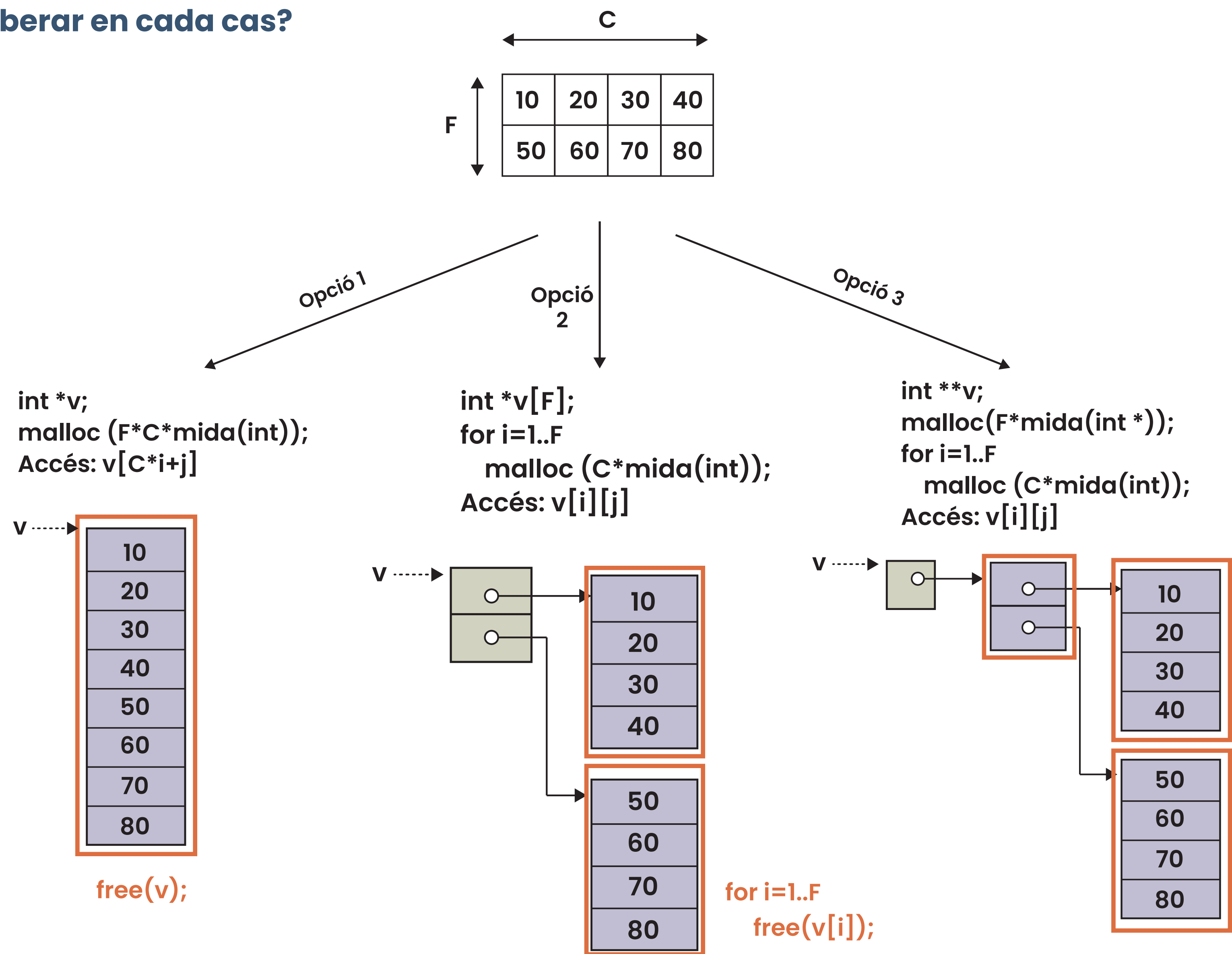
No ens hem oblidat d'alguna cosa?

Quina memòria hem d'alliberar en cada cas?



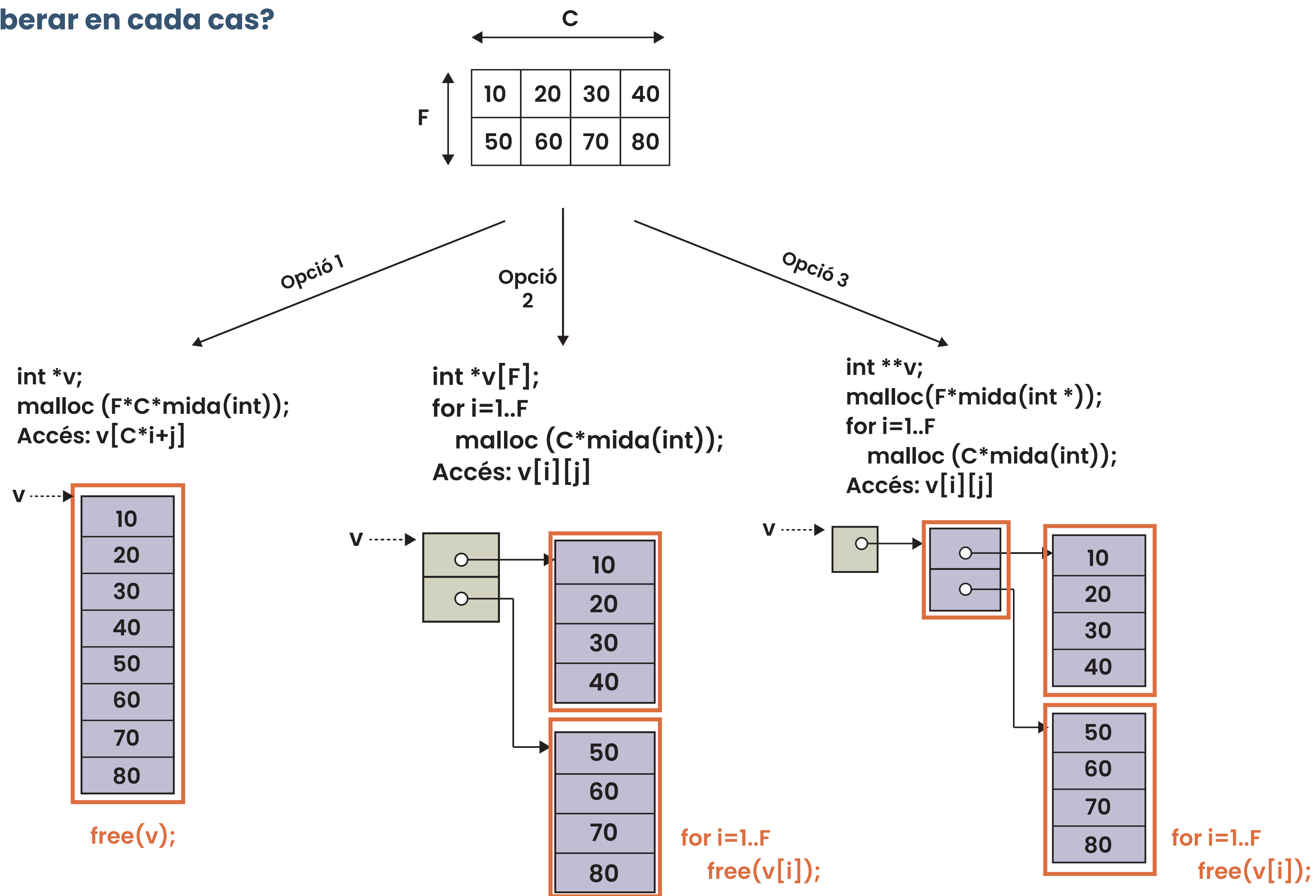
No ens hem oblidat d'alguna cosa?

Quina memòria hem d'alliberar en cada cas?



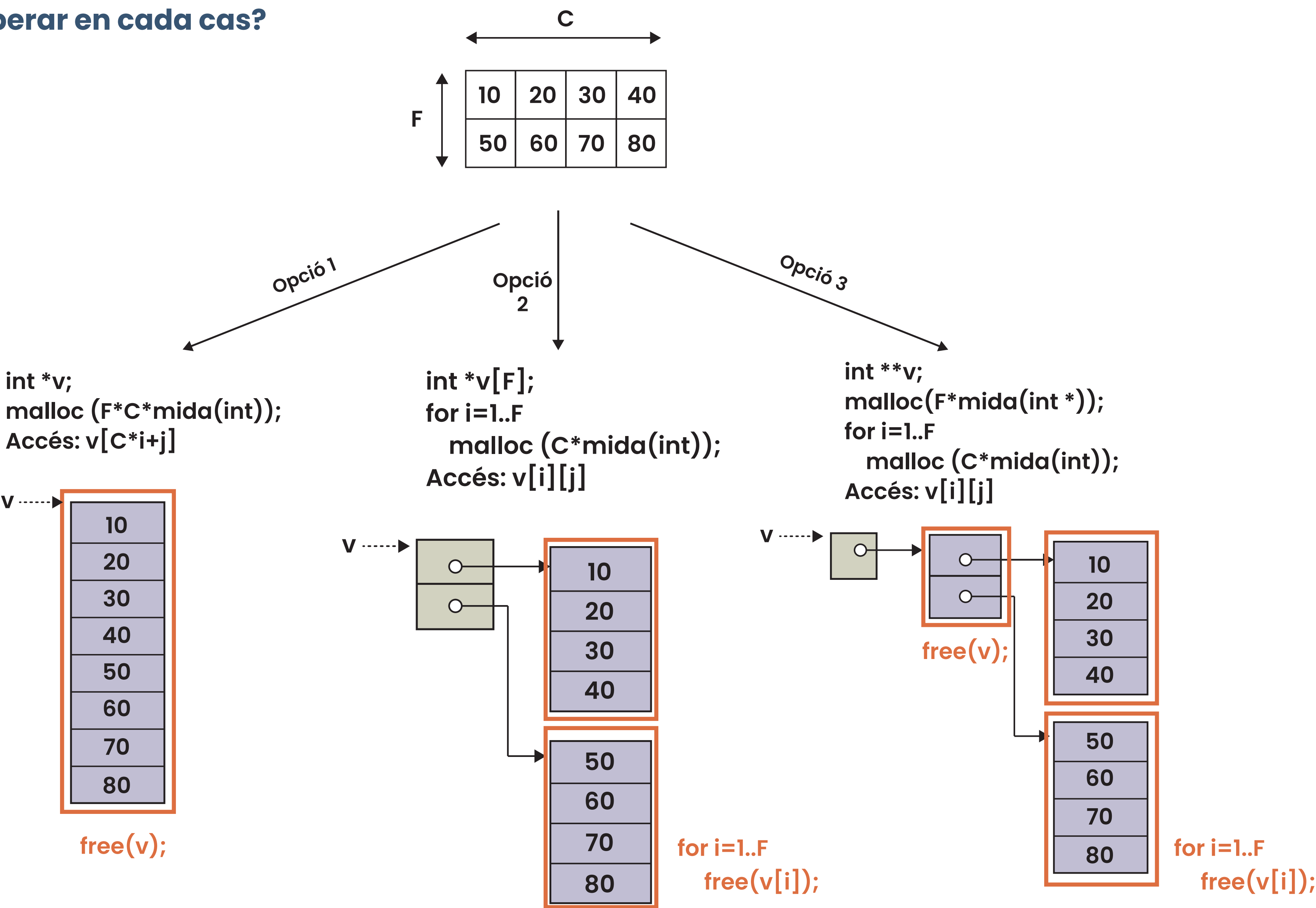
No ens hem oblidat d'alguna cosa?

Quina memòria hem d'alliberar en cada cas?



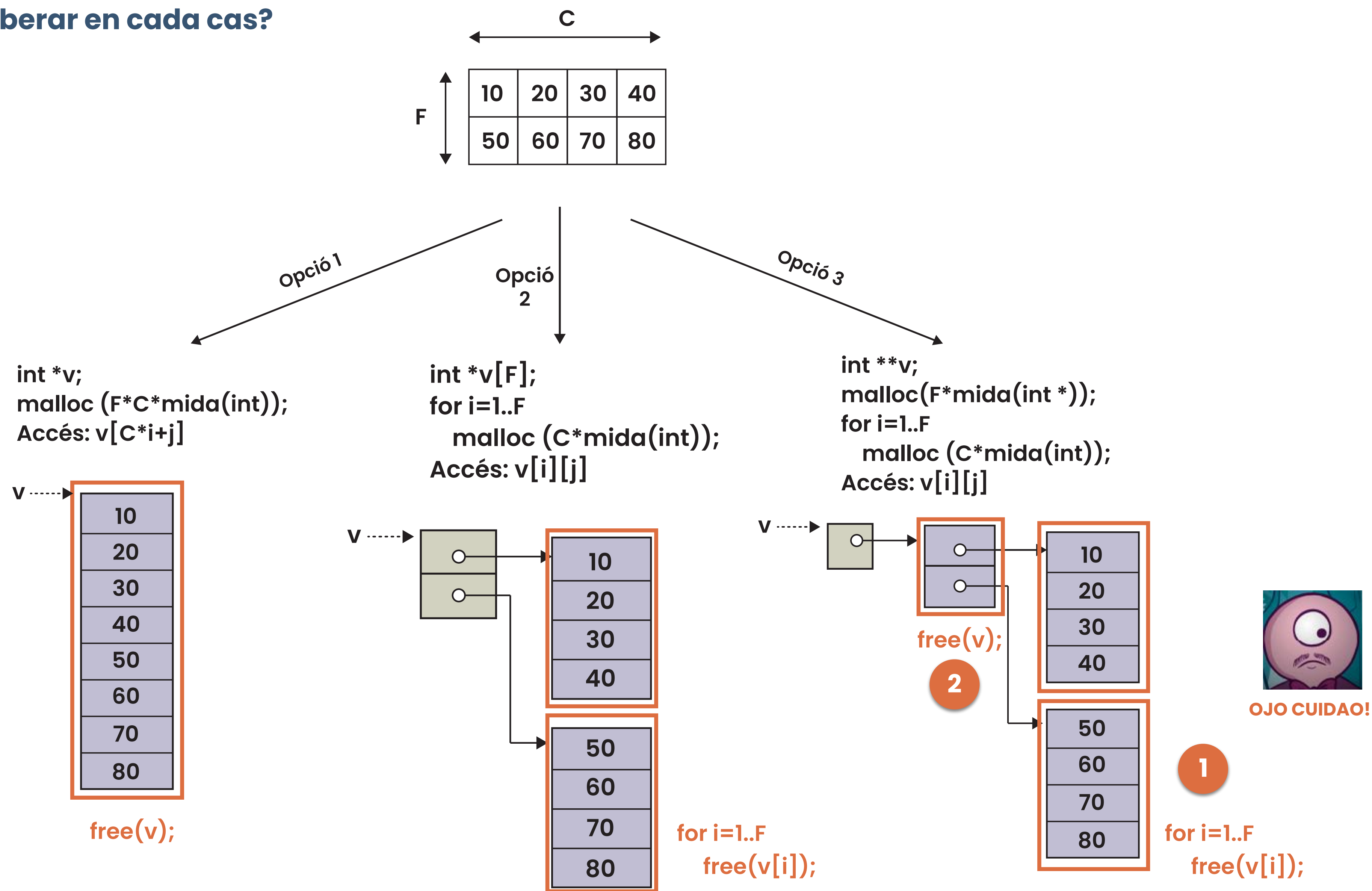
No ens hem oblidat d'alguna cosa?

Quina memòria hem d'alliberar en cada cas?



No ens hem oblidat d'alguna cosa?

Quina memòria hem d'alliberar en cada cas?



Taules de Strings

Com guardar múltiples strings

- Un string és una taula de caràcters.
- Si volem guardar múltiples strings, haurem de fer servir una **taula de strings** = **taula de taula de caràcters**
- Les opcions que hem vist abans ens serveixen també pels strings

Taules de Strings

Com guardar múltiples strings

- Un string és una taula de caràcters.
- Si volem guardar múltiples strings, haurem de fer servir una **taula de strings = taula de taula de caràcters**
- Les opcions que hem vist abans ens serveixen també pels strings

Quina opció triar?

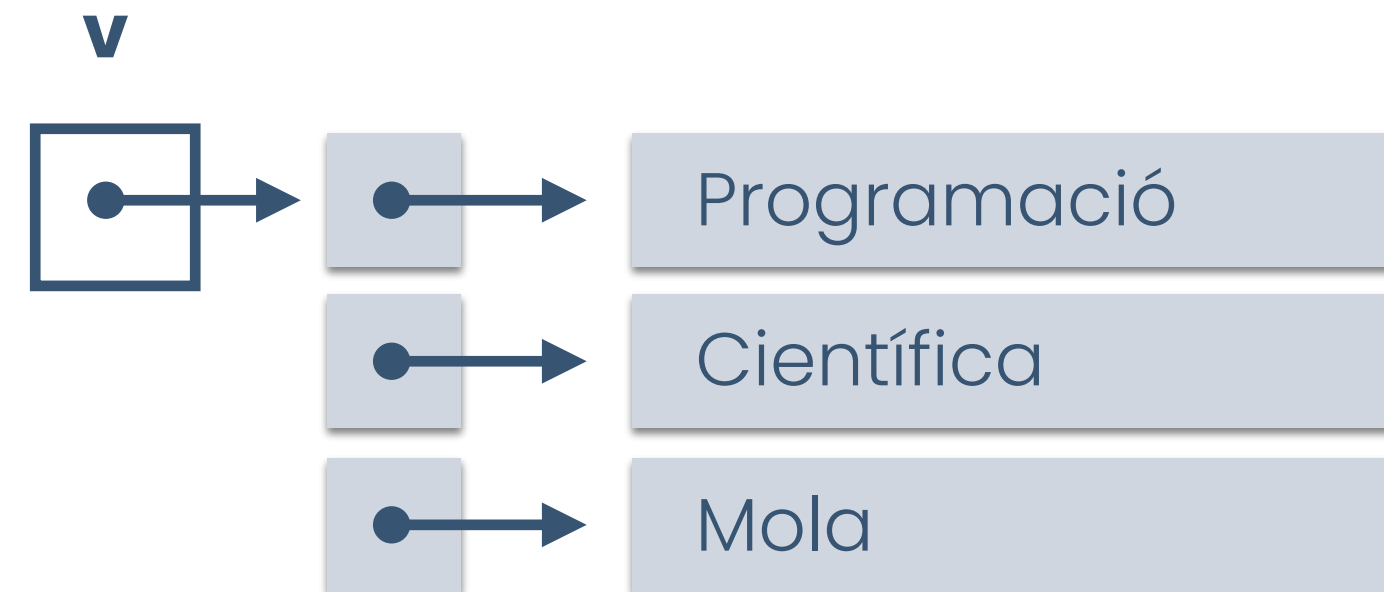
- Si sabem [en temps de compilació] el nombre de paraules/frases/strings que hem de llegir, podem fer l'Opció 2.
- Si no sabem aquest nombre, haurem de fer l'Opció 3.
- De fet, l'opció 3 és la que ens serveix per qualsevol situació.

Taules de Strings

Exemple: Guardar strings que l'usuari introdueix per teclat [amb Opció 3]

- Si no sabem la longitud dels strings, suposar una llargada màxima. [max_llarg] (*equivalent a num_col*)
- Preguntar a l'usuari quants strings vol introduir: [num_strings] (*equivalent a num_fil*)
- Reservar espai de memòria per la taula de punters a string
- Reservar espai de memòria per cada string
- Guardar-los

max_llarg = 50
num_strings = 3

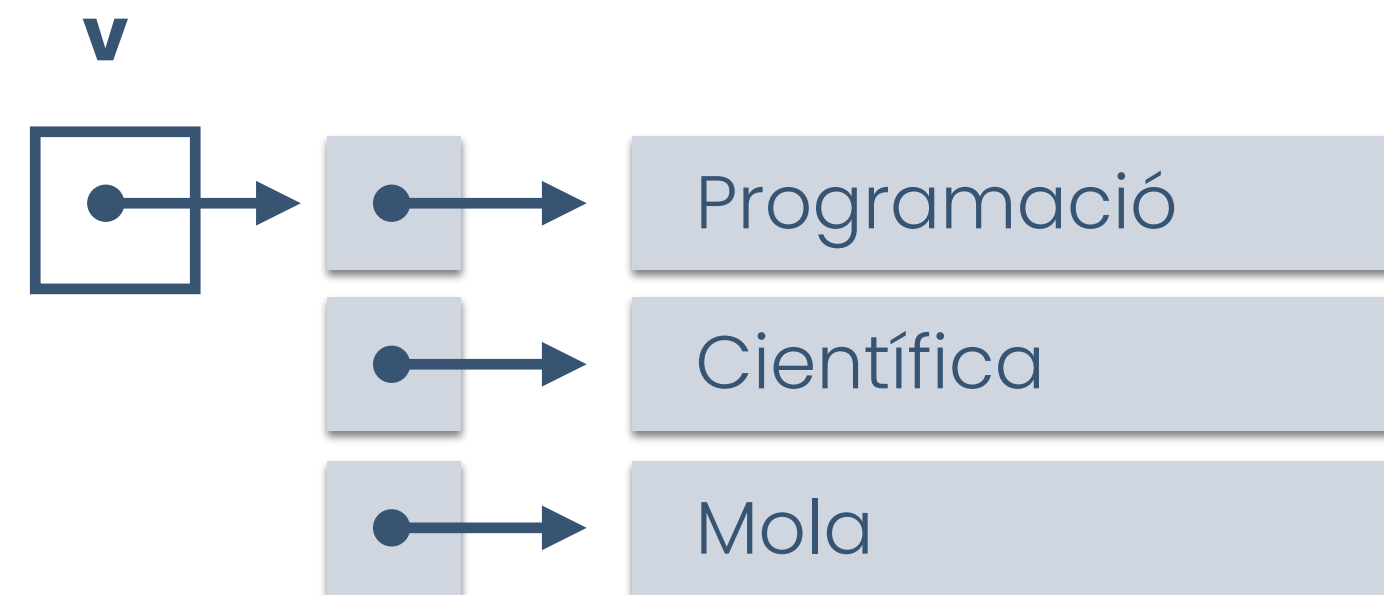


Taules de Strings

Exemple: Guardar strings que l'usuari introdueix per teclat [amb Opció 3]

- Si no sabem la longitud dels strings, suposar una llargada màxima. [max_llarg] (*equivalent a num_col*)
- Preguntar a l'usuari quants strings vol introduir: [num_strings] (*equivalent a num_fil*)
- Reservar espai de memòria per la taula de punters a string
- Reservar espai de memòria per cada string
- Guardar-los

max_llarg = 50
num_strings = 3

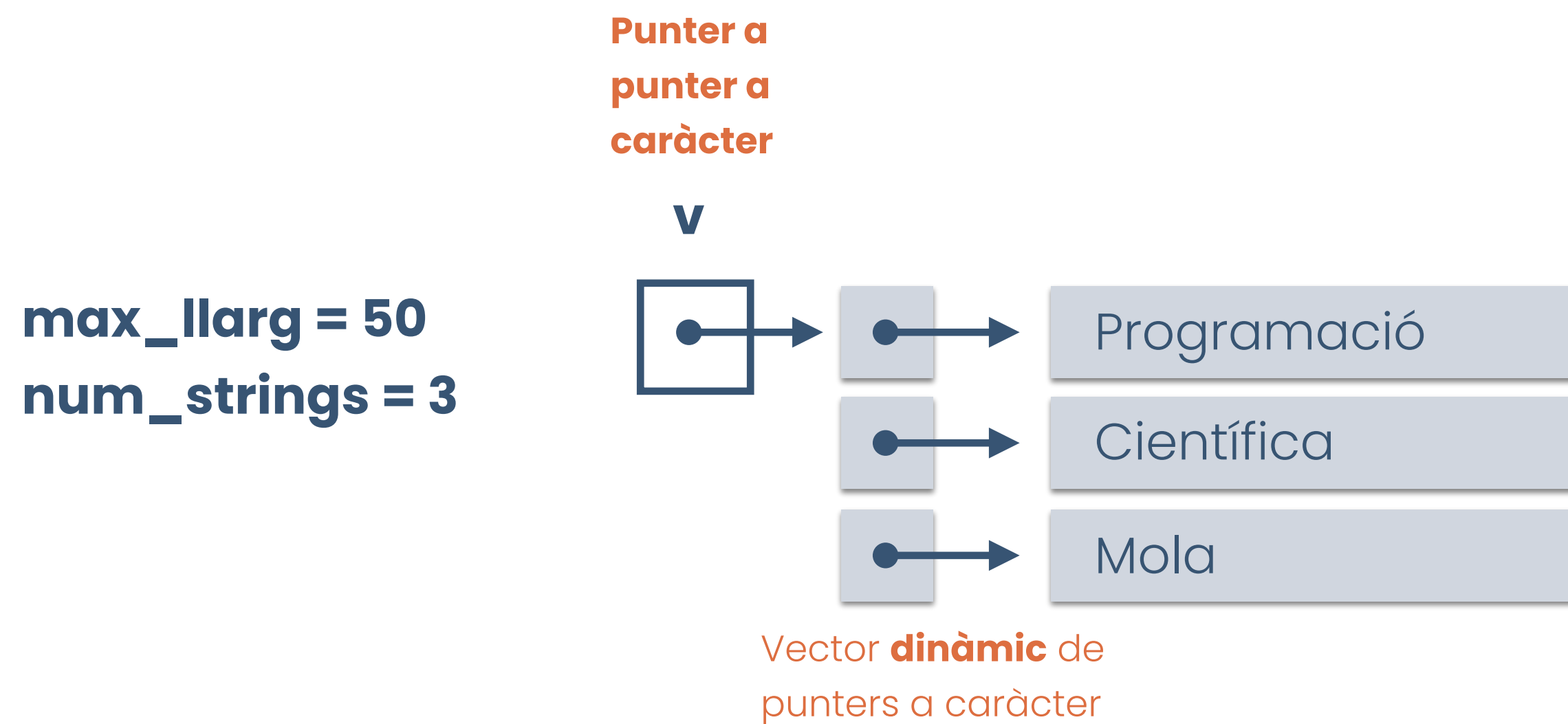


Vector **dinàmic** de
punters a caràcter

Taules de Strings

Exemple: Guardar strings que l'usuari introdueix per teclat [amb Opció 3]

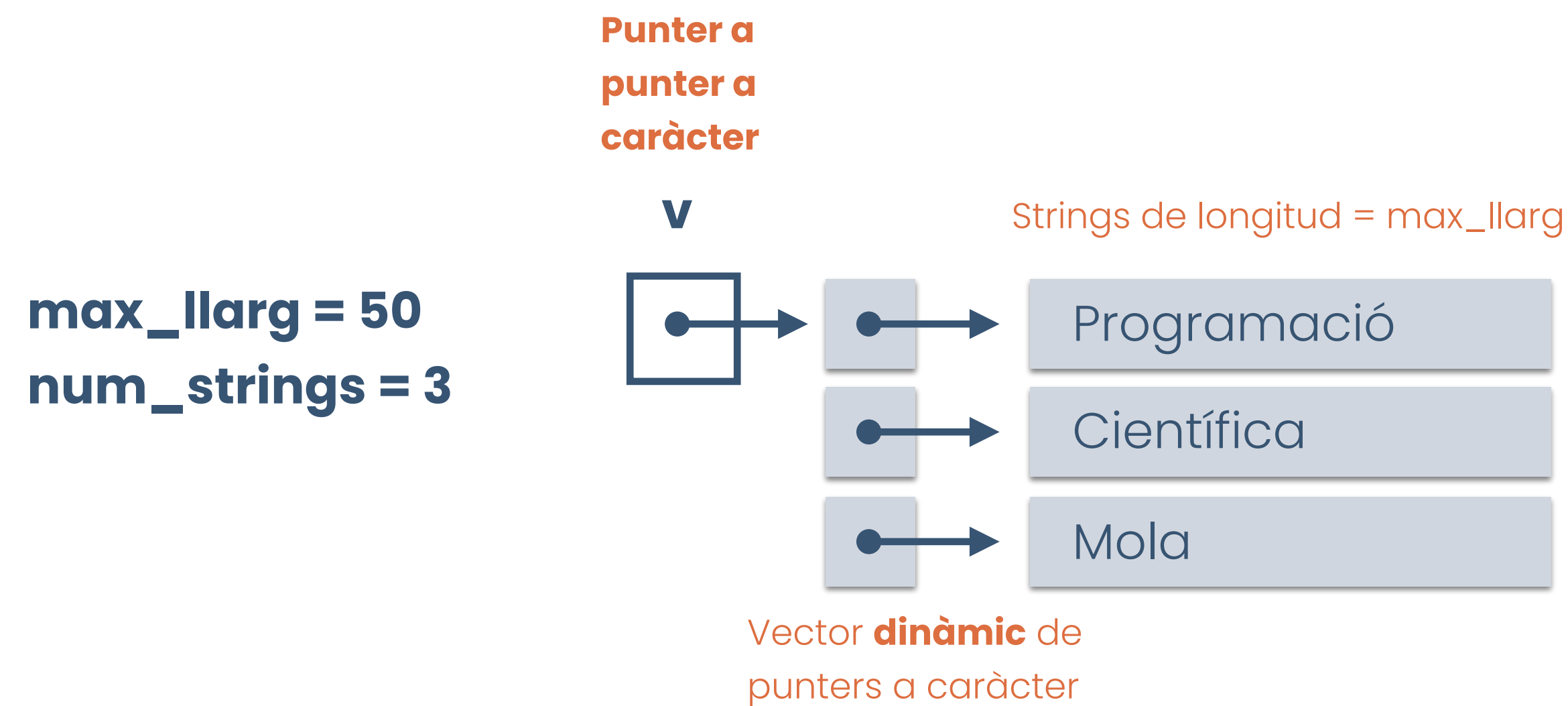
- Si no sabem la longitud dels strings, suposar una llargada màxima. [max_llarg] (*equivalent a num_col*)
- Preguntar a l'usuari quants strings vol introduir: [num_strings] (*equivalent a num_fil*)
- Reservar espai de memòria per la taula de punters a string
- Reservar espai de memòria per cada string
- Guardar-los



Taules de Strings

Exemple: Guardar strings que l'usuari introdueix per teclat [amb Opció 3]

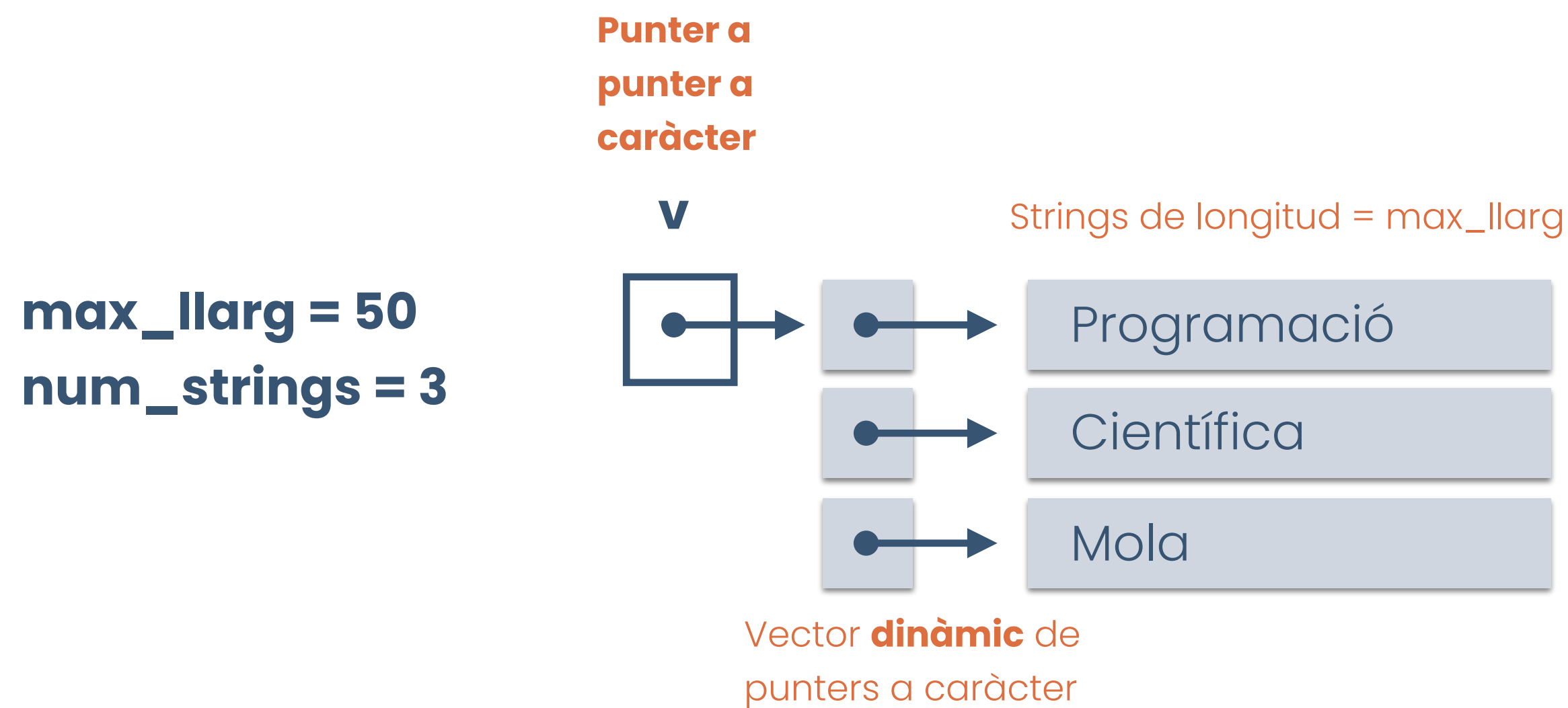
- Si no sabem la longitud dels strings, suposar una llargada màxima. [max_llarg] (*equivalent a num_col*)
- Preguntar a l'usuari quants strings vol introduir: [num_strings] (*equivalent a num_fil*)
- Reservar espai de memòria per la taula de punters a string
- Reservar espai de memòria per cada string
- Guardar-los



Taules de Strings

Exemple: Guardar strings que l'usuari introdueix per teclat [amb Opció 3]

- Si no sabem la longitud dels strings, suposar una llargada màxima. [max_llarg] (*equivalent a num_col*)
- Preguntar a l'usuari quants strings vol introduir: [num_strings] (*equivalent a num_fil*)
- Reservar espai de memòria per la taula de punters a string
- Reservar espai de memòria per cada string
- Guardar-los



- **Escriviu un programa que demani a l'usuari quantes paraules vol guardar, i les guardi en una taula amb aquesta estructura.**

Taules de Strings

Exemple: Guardar strings que l'usuari introdueix per teclat [amb Opció 3]. Solució.

- Declarar variables i demanar num_paraules a l'usuari.

```
#include <stdio.h>
#include <stdlib.h>

int main() {

    char ** taula_paraules; /*punter a punter a caracter*/

    /* Llargada màxima, pot ser constant, o pot dir-ho l'usuari */
    const int max_llarg = 50;
    int num_paraules;

    /* Preguntar a l'usuari quantes paraules vol */
    printf("Quantes paraules vols introduir?\n");
    scanf("%d", &num_paraules);
```

Taules de Strings

Exemple: Guardar strings que l'usuari introdueix per teclat [amb Opció 3]. Solució.

- Alocatar memòria i guardar paraules a la taula.
- Imprimir la taula:

```
/* Alocatar memòria per vector de punter a char de mida num_paraules*/
taula_paraules = (char **) malloc (num_paraules * sizeof(char*));
// Comprovacio memòria (...)
/* Alocatar memòria per cada string */
for(int i = 0; i<num_paraules; i++){
    taula_paraules[i] = (char *) malloc (max_llarg * sizeof(char));
    // Comprovació memoria (...)
    /* Llegir string d'usuari i guardar-lo a taula */
    scanf("%s",taula_paraules[i]);
}
/* Imprimir taula */
for(int i = 0; i < num_paraules; i++){
    printf("Paraula %d: %s\n", i, taula_paraules[i]);
}
```

Taules de Strings

Exemple: Guardar strings que l'usuari introdueix per teclat [amb Opció 3]. Solució.

- Alliberar memòria i acabar programa

```
/* Alliberar memòria */  
for(int i = 0; i < num_paraules; i++){  
    free(taula_paraules[i]);  
}  
free(taula_paraules);  
  
return 0;  
}
```

CRONOMETRAR

Cronometrar l'execució d'un programa

Com es fa?

- Si volem saber quant ha trigat un programa sencer o bé només una part del codi, podem cronometrar-lo.
- El temps del sistema es mesura amb el rellotge del sistema, que compta el nombre de ticks que han passat des d'una data inicial arbitrària.

Cronometrar l'execució d'un programa

Com es fa?

- Si volem saber quant ha trigat un programa sencer o bé només una part del codi, podem cronometrar-lo.
- El temps del sistema es mesura amb el rellotge del sistema, que compta el nombre de ticks que han passat des d'una data inicial arbitrària.
- Necessitarem incloure la llibreria **<time.h>**
- Necessitarem declarar variables de tipus **clock_t** (clock ticks)
- Farem servir la funció **clock()**

Cronometrar l'execució d'un programa

Com es fa?

- Si volem saber quant ha trigat un programa sencer o bé només una part del codi, podem cronometrar-lo.
- El temps del sistema es mesura amb el rellotge del sistema, que compta el nombre de ticks que han passat des d'una data inicial arbitrària.
- Necessitarem incloure la llibreria **<time.h>**
- Necessitarem declarar variables de tipus **clock_t** (clock ticks)
- Farem servir la funció **clock()**
 - **clock()**: Retorna el temps de processador que ha fet servir el programa des del començament de l'execució o retorna -1 si no està disponible

```
clock_t clock(void)
```

Cronometrar l'execució d'un programa

Com es fa?

- Si volem saber quant ha trigat un programa sencer o bé només una part del codi, podem cronometrar-lo.
 - El temps del sistema es mesura amb el rellotge del sistema, que compta el nombre de ticks que han passat des d'una data inicial arbitrària.
 - Necessitarem incloure la llibreria **<time.h>**
 - Necessitarem declarar variables de tipus **clock_t** (clock ticks)
 - Farem servir la funció **clock()**
 - **clock()**: Retorna el temps de processador que ha fet servir el programa des del començament de l'execució o retorna -1 si no està disponible
- ```
clock_t clock(void)
```
- Per convertir el nombre de ticks a segons necessitem dividir per la constant **CLOCKS\_PER\_SEC** (que depèn de la màquina)

# Cronometrar l'execució d'un programa

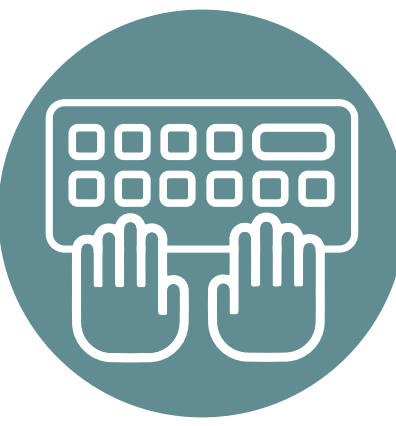
## Com es fa?

- Per saber quant de temps ha trigat el tros de codi necessitem saber el nombre de ticks abans i després:

```
1 #include <stdio.h>
2 #include <time.h>
3
4 int main(){
5
6 clock_t inici, final;
7 double temps_consumit;
8
9 inici = clock();
10 /* Aquí les coses que vulguem cronometrar*/
11 final = clock();
12 temps_consumit = ((double) (final-inici)) / CLOCKS_PER_SEC;
13
14 return 0;
15 }
```

*La llibreria <time.h> ens permet cronometrar un programa*

# Cronometrar l'execució d'un programa



## Exemple complet

- Baixeu del moodle el fitxer **clock\_exemple.c** i executeu-lo

```
1 #include <stdio.h>
2 #include <time.h>
3 #define N 1000000
4
5 int main(){
6
7 clock_t inici, final;
8 double segons_total;
9
10 inici = clock();
11 for(int i=0; i<N; i++){
12 printf("Soc el numero %d\n",i);
13 }
14 final = clock();
15
16 segons_total = (double)(final-inici)/CLOCKS_PER_SEC;
17 printf("Han passat %.4f segons\n",segons_total);
18
19 return 0;
20 }
```

# Pausar un programa

## Per què?

- A vegades podrem voler pausar un programa a l'espera d'alguna cosa.
- Tenim diversos tipus de pausa:



# Pausar un programa

## Per què?

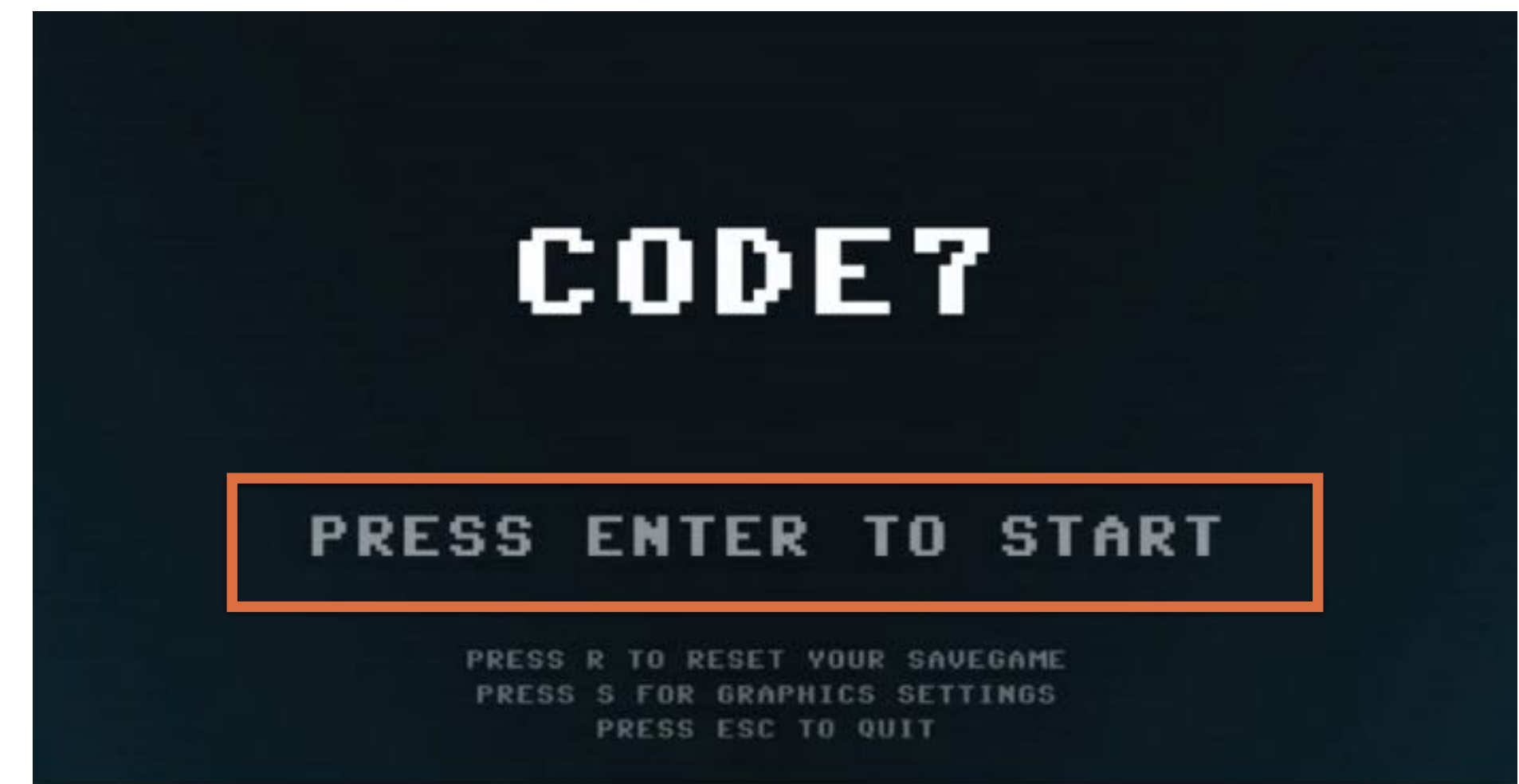
- A vegades podrem voler pausar un programa a l'espera d'alguna cosa.
- Tenim diversos tipus de pausa:

- **Cas 1. Pausa esperant a que l'usuari introdueixi una tecla**

Generalment ho fem perquè l'usuari doni el tret d'inici d'un procés

La longitud de la pausa depèn de quan l'usuari premi la tecla. Mentrestant, el nostre procés està en espera.

El cas típic és el <PRESS ENTER TO START>



# Pausar un programa

## Per què?

- A vegades podrem voler pausar un programa a l'espera d'alguna cosa.
- Tenim diversos tipus de pausa:

- **Cas 1. Pausa esperant a que l'usuari introdueixi una tecla**

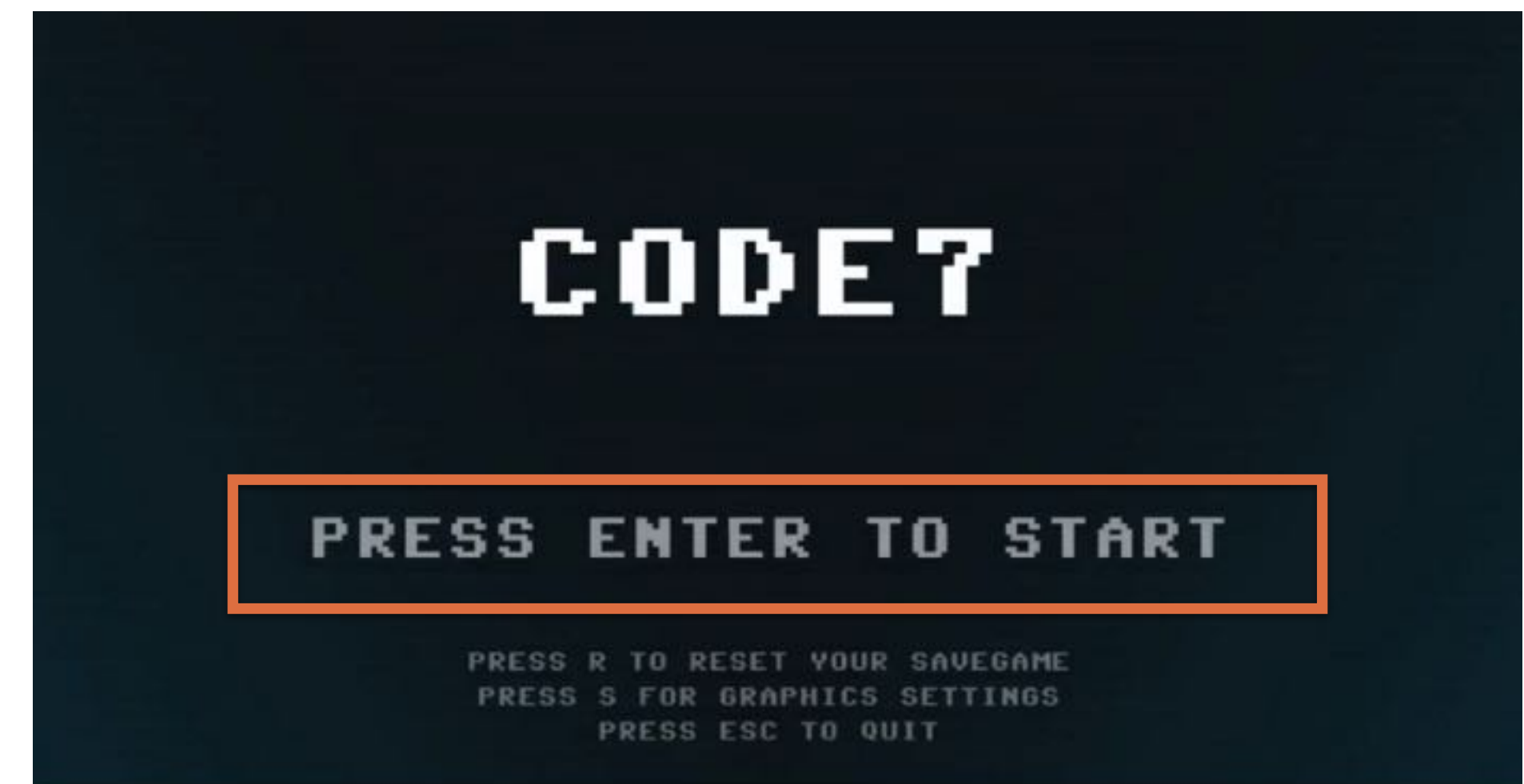
Generalment ho fem perquè l'usuari doni el tret d'inici d'un procés

La longitud de la pausa depèn de quan l'usuari premi la tecla. Mentrestant, el nostre procés està en espera.

El cas típic és el <PRESS ENTER TO START>

- **Cas 2. Pausa d'un temps determinat perquè necessitem parar el nostre programa un temps**

Per què necessitem parar-lo? Possiblement perquè estem mostrant una sèrie de coses per pantalla que són massa ràpides per l'usuari i necessitem afegir un temps "humà" de retard entre elles.



# Pausar un programa

## Cas 1. Pausa esperant a que l'usuari introdueixi una tecla

- Per fer-ho, senzillament aprofitarem que les funcions que demanen alguna cosa a l'usuari s'esperen a què l'usuari introdueixi alguna cosa.
- Millor opció: `getchar ( )`
  - La funció `getchar` llegeix de teclat un únic caràcter.
  - La seva capçalera és: `int getchar(void)` (Sí, retorna `int` i no `char...`)

# Pausar un programa

## Cas 1. Pausa esperant a que l'usuari introdueixi una tecla

- Per fer-ho, senzillament aprofitarem que les funcions que demanen alguna cosa a l'usuari s'esperen a què l'usuari introdueixi alguna cosa.
- Millor opció: `getchar( )`
  - La funció `getchar` llegeix de teclat un únic caràcter.
  - La seva capçalera és: `int getchar(void)` (Sí, retorna `int` i no `char`...)

- Com funciona `getchar()`:

```
1 #include <stdio.h>
2 int main()
3 {
4 char c;
5 printf("Introdueix un caràcter\n");
6 c = getchar();
7 printf("Has introduit: %c\n", c);
8
9 return 0;
10 }
```

```
Introdueix un caràcter
a
Has introduit: a
```

# Pausar un programa

## Cas 1. Pausa esperant a que l'usuari introdueixi una tecla

- Fer servir `getchar()` per pausar el programa

```
1 #include <stdio.h>
2
3 int main()
4 {
5 printf("Press <Enter> to continue... \n");
6 getchar();
7 printf("Aquí comença el programa... \n");
8
9 return 0;
10 }
```

Press <Enter> to continue...  
Aquí comença el programa...

# Pausar un programa

## Cas 2. Pausa d'un temps determinat perquè necessitem parar el nostre programa un temps


- Podem construir un procediment "delay" a partir del que hem après a fer servir amb els rellotges?



# Pausar un programa

## Cas 2. Pausa d'un temps determinat perquè necessitem parar el nostre programa un temps

- Podem construir un procediment "delay" a partir del que hem après a fer servir amb els rellotges?



```
1 void retard (int milliseconds)
2 {
3 long pausa;
4 clock_t inici, ara;
5
6 pausa = milliseconds*(CLOCKS_PER_SEC/1000);
7
8
9 inici = ara = clock();
10
11
12 while((ara-inici) < pausa){
13 ara = clock();
14 }
15 }
```

*Acció "delay" que no acaba fins al cap del nombre de milisegons introduït*

# Pausar un programa

## Cas 2. Pausa d'un temps determinat perquè necessitem parar el nostre programa un temps

- Podem construir un procediment "delay" a partir del que hem après a fer servir amb els rellotges?

```
1 void retard (int milliseconds)
2 {
3 long pausa;
4 clock_t inici, ara;
5
6 pausa = milliseconds*(CLOCKS_PER_SEC/1000);
7 Convertim els milisegons a clocks de CPU
8
9 inici = ara = clock();
10
11
12 while((ara-inici) < pausa){
13 ara = clock();
14 }
15 }
```

*Acció "delay" que no acaba fins al cap del nombre de milisegons introduït*

# Pausar un programa

## Cas 2. Pausa d'un temps determinat perquè necessitem parar el nostre programa un temps

- Podem construir un procediment "delay" a partir del que hem après a fer servir amb els rellotges?

```
1 void retard (int milliseconds)
2 {
3 long pausa;
4 clock_t inici, ara;
5
6 pausa = milliseconds*(CLOCKS_PER_SEC/1000);
7 Convertim els milisegons a clocks de CPU
8
9 inici = ara = clock();
10 Inicialitzem els rellotges al temps actual
11
12 while((ara-inici) < pausa){
13 ara = clock();
14 }
15 }
```

*Acció "delay" que no acaba fins al cap del nombre de milisegons introduït*

# Pausar un programa

## Cas 2. Pausa d'un temps determinat perquè necessitem parar el nostre programa un temps

- Podem construir un procediment "delay" a partir del que hem après a fer servir amb els rellotges?

```
1 void retard (int milliseconds)
2 {
3 long pausa;
4 clock_t inici, ara;
5
6 pausa = milliseconds*(CLOCKS_PER_SEC/1000);
7 Convertim els milisegons a clocks de CPU
8
9 inici = ara = clock();
10 Inicialitzem els rellotges al temps actual
11
12 while((ara-inici) < pausa){
13 ara = clock(); Aquest bucle es farà mentre el temps
14 } transcorregut sigui menor a la pausa
15 } que volem
```

*Acció "delay" que no acaba fins al cap del nombre de milisegons introduït*

# Pausar un programa

## Cas 2. Pausa d'un temps determinat perquè necessitem parar el nostre programa un temps

- Podem construir un procediment "delay" a partir del que hem après a fer servir amb els rellotges?

```
1 void retard (int milliseconds)
2 {
3 long pausa;
4 clock_t inici, ara;
5
6 pausa = milliseconds*(CLOCKS_PER_SEC/1000);
7 Convertim els milisegons a clocks de CPU
8
9 inici = ara = clock();
10 Inicialitzem els rellotges al temps actual
11
12 while((ara-inici) < pausa){
13 ara = clock(); Aquest bucle es farà mentre el temps
14 } transcorregut sigui menor a la pausa
15 } que volem
```

Acció "delay" que no acaba fins al cap del nombre de milisegons introduït

```
1 int main()
2 {
3 for (int i = 0; i < 10; i++) {
4 // delay of one second
5 delay(1000);
6 printf("%d seconds have passed\n", i + 1);
7 }
8 return 0;
9 }
```

Programa principal que fa servir la funció delay

# Pausar un programa

## Cas 2. Pausa d'un temps determinat perquè necessitem parar el nostre programa un temps

- Podem construir un procediment "delay" a partir del que hem après a fer servir amb els rellotges?

```
1 void retard (int milliseconds)
2 {
3 long pausa;
4 clock_t inici, ara;
5
6 pausa = milliseconds*(CLOCKS_PER_SEC/1000);
7 Convertim els milisegons a clocks de CPU
8
9 inici = ara = clock();
10 Inicialitzem els rellotges al temps actual
11
12 while((ara-inici) < pausa){
13 ara = clock(); Aquest bucle es farà mentre el temps
14 } transcorregut sigui menor a la pausa
15 } que volem
```

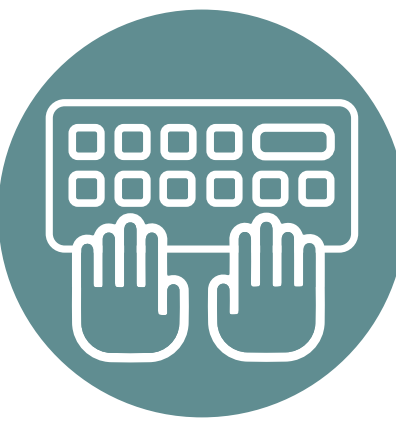
*Acció "delay" que no acaba fins al cap del nombre de milisegons introduït*

```
1 int main()
2 {
3 for (int i = 0; i < 10; i++) {
4 // delay of one second
5 delay(1000); Introdueixo el retard en milisegons
6 printf("%d seconds have passed\n", i + 1);
7 }
8 return 0;
9 }
```

*Programa principal que fa servir la funció delay*



# Pausar un programa



## Cas 2. Pausa d'un temps determinat perquè necessitem parar el nostre programa un temps

- Podem construir un procediment "delay" a partir del que hem après a fer servir amb els rellotges?

```
1 void retard (int milliseconds)
2 {
3 long pausa;
4 clock_t inici, ara;
5
6 pausa = milliseconds*(CLOCKS_PER_SEC/1000);
7 Convertim els milisegons a clocks de CPU
8
9 inici = ara = clock();
10 Inicialitzem els rellotges al temps actual
11
12 while((ara-inici) < pausa){
13 ara = clock(); Aquest bucle es farà mentre el temps
14 } transcorregut sigui menor a la pausa
15 } que volem
```

Acció "delay" que no acaba fins al cap del nombre de milisegons introduït

pausar\_programa.c

```
1 int main()
2 {
3 for (int i = 0; i < 10; i++) {
4 // delay of one second
5 delay(1000); Introdueixo el retard en milisegons
6 printf("%d seconds have passed\n", i + 1);
7 }
8 return 0;
9 }
```

Programa principal que fa servir la funció delay