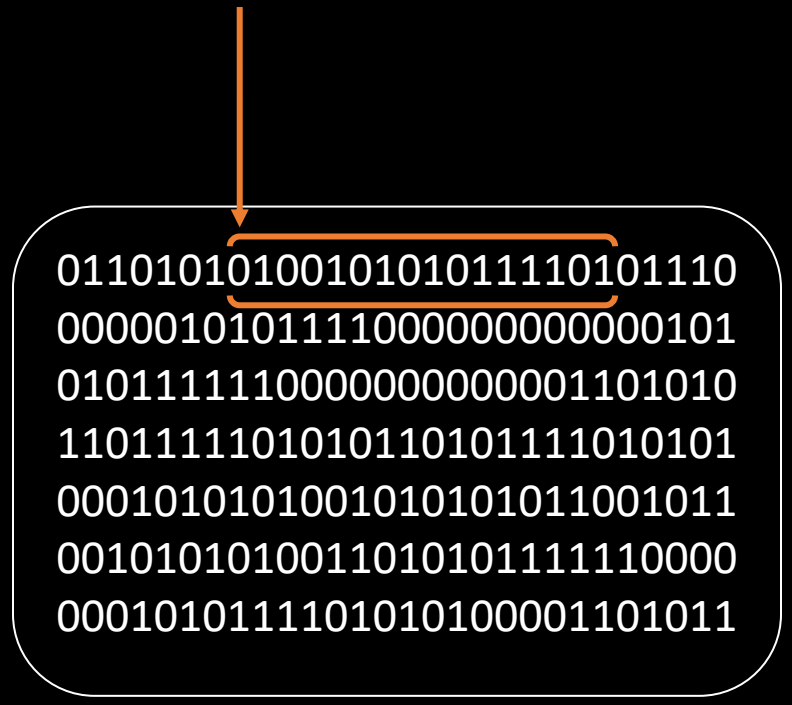
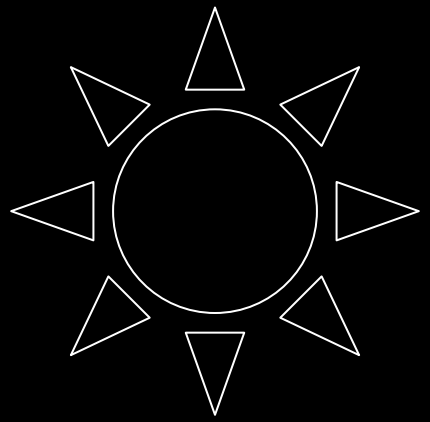
Les p😈inteurs

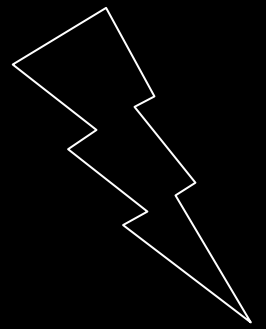
Retrouver sa donnée



Store



Control

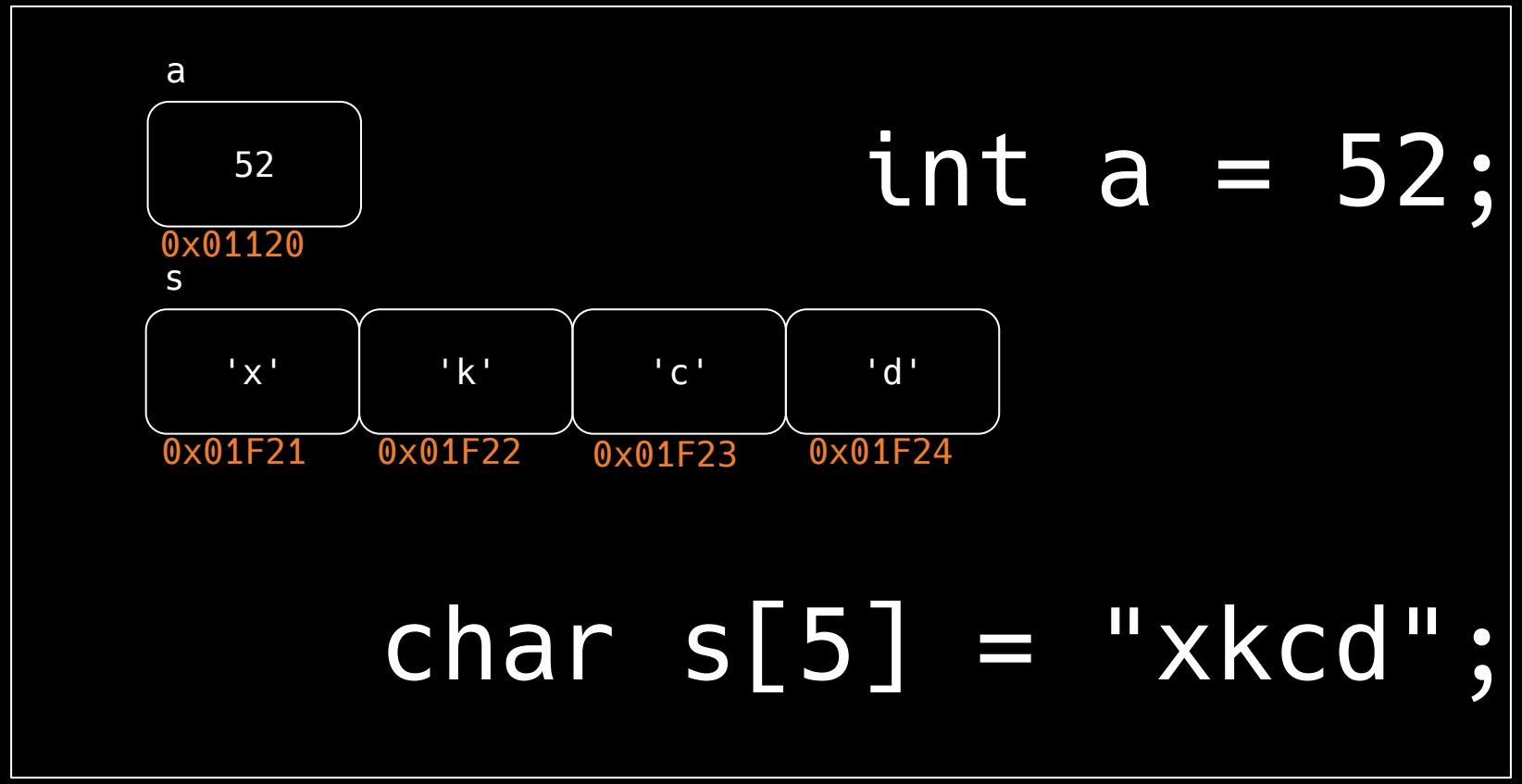


Executive unit

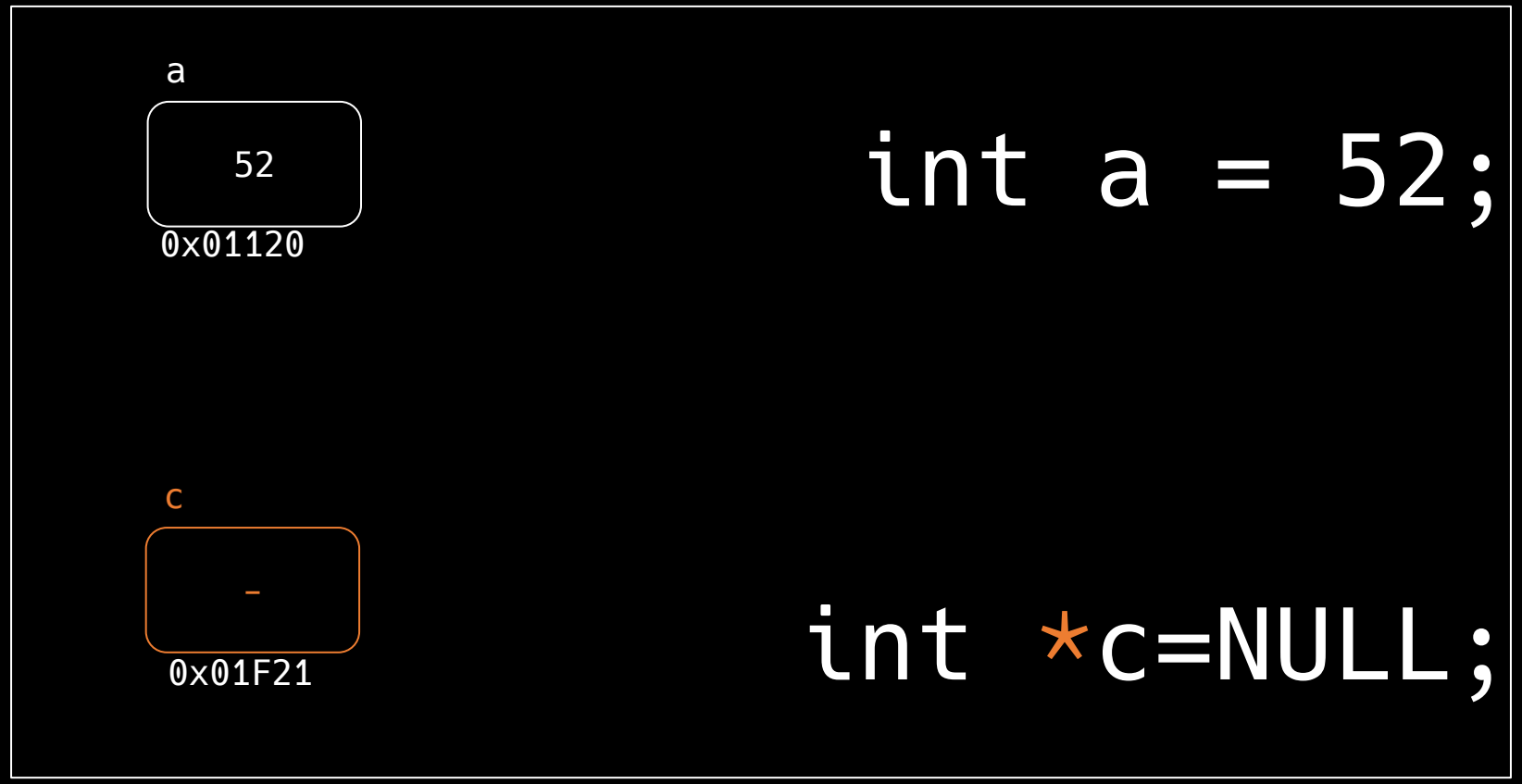
Les p😈inteurs

Les p😇inteurs

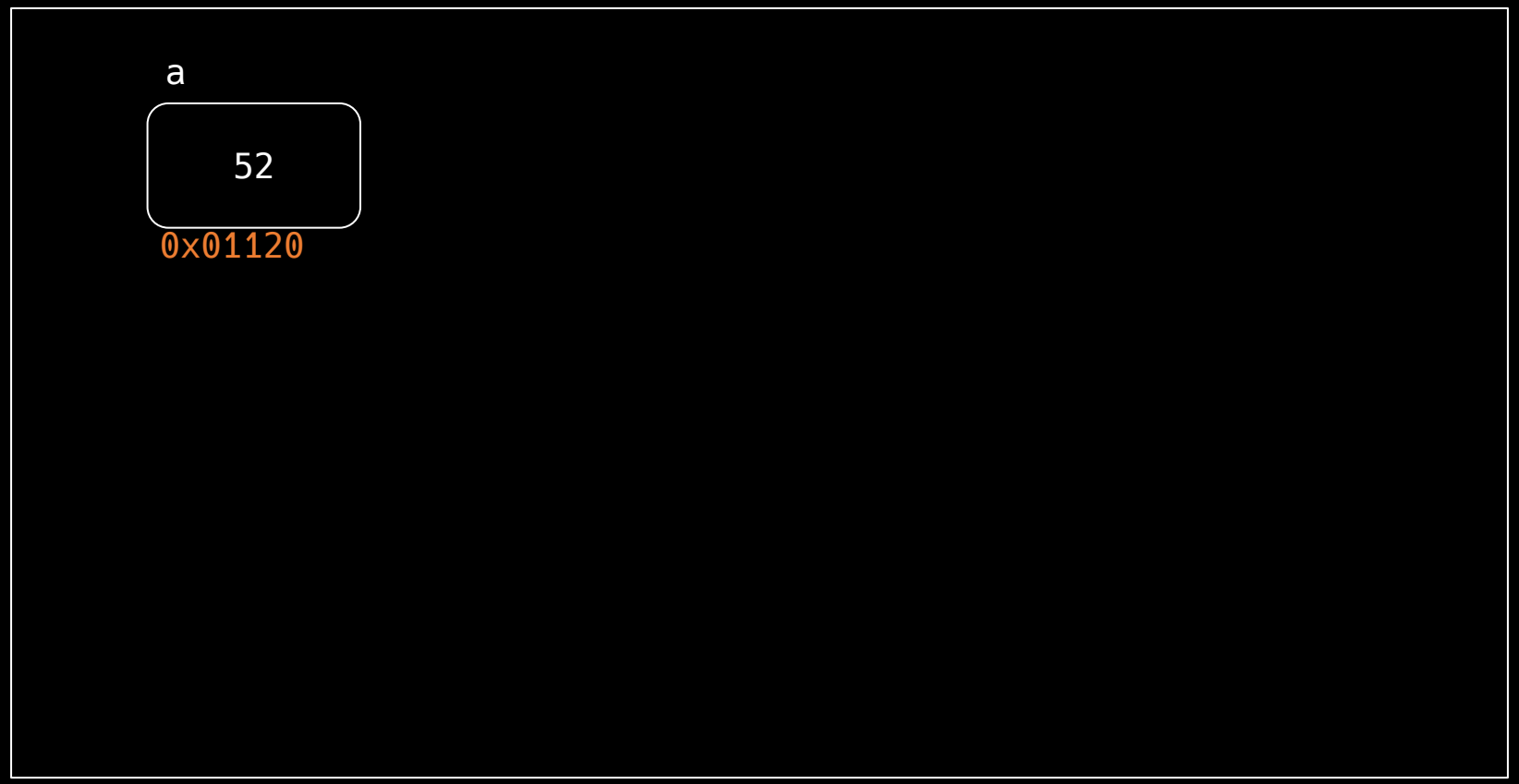
Adresse de la mémoire



Déclaration de pointeur

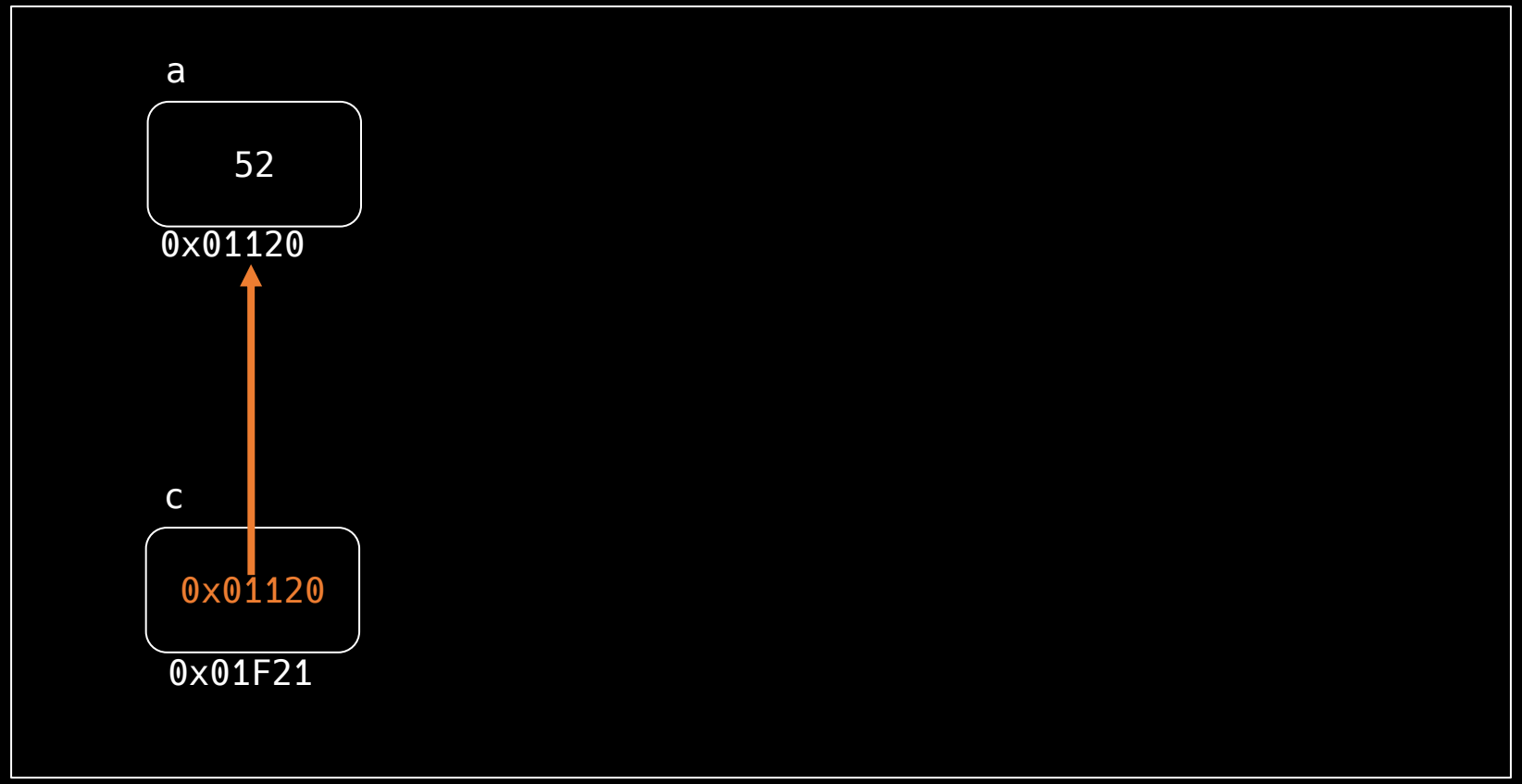


« Donne-moi l'adresse de la variable »



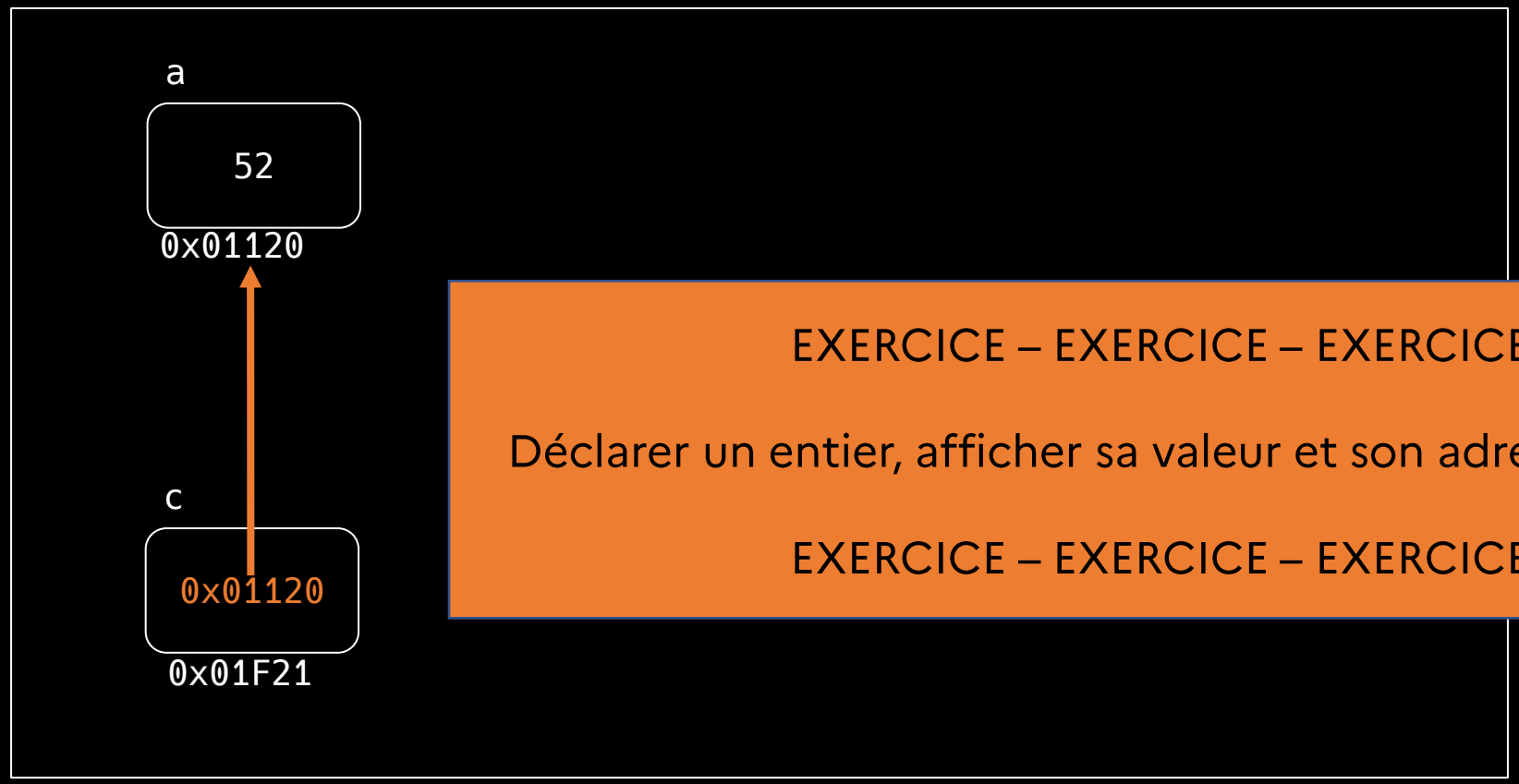
`&a;`

« Donne-moi l'adresse de la variable »




```
c = &a;
```

« Donne-moi l'adresse de la variable »



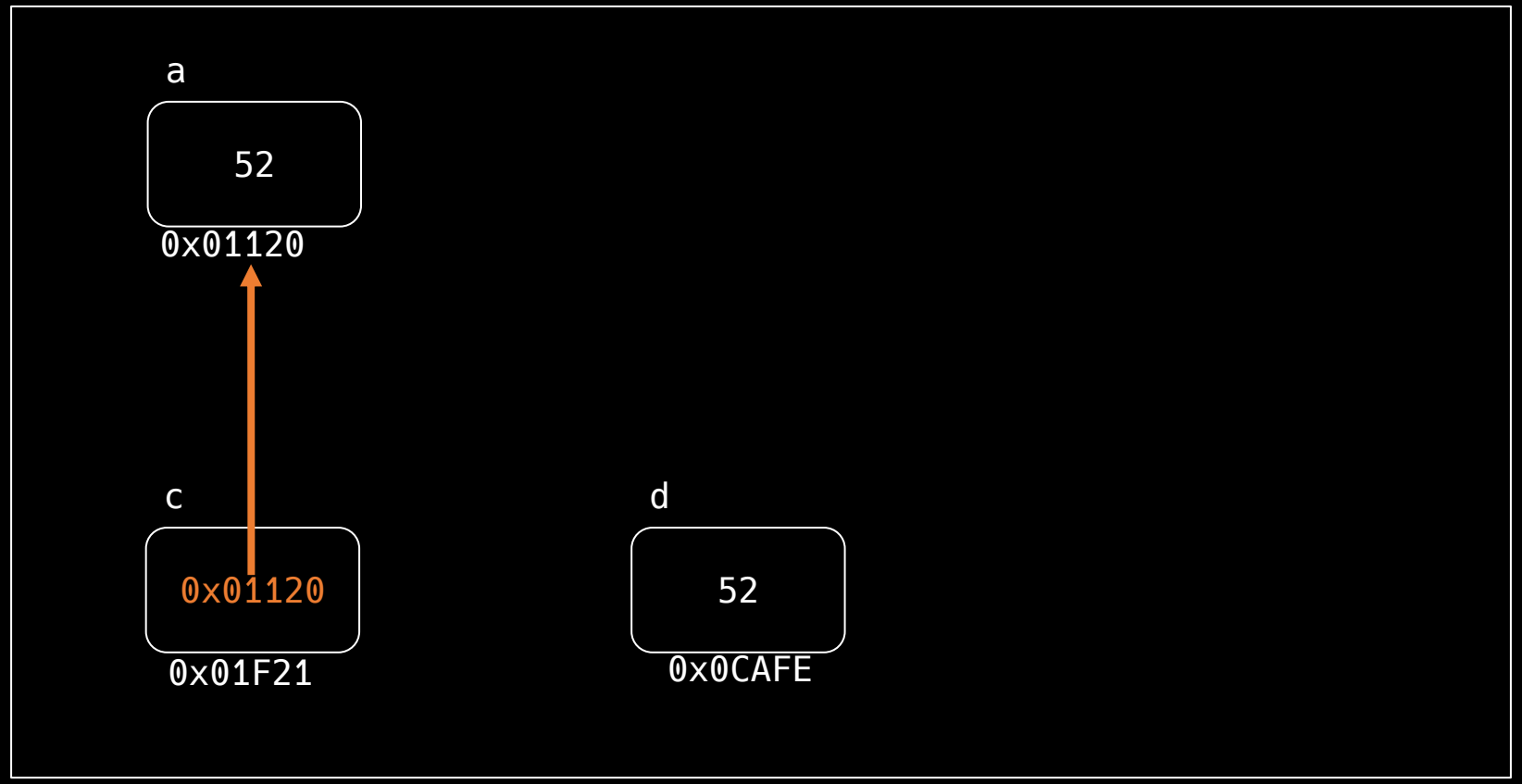
EXERCICE – EXERCICE – EXERCICE

Déclarer un entier, afficher sa valeur et son adresse mémoire.

EXERCICE – EXERCICE – EXERCICE

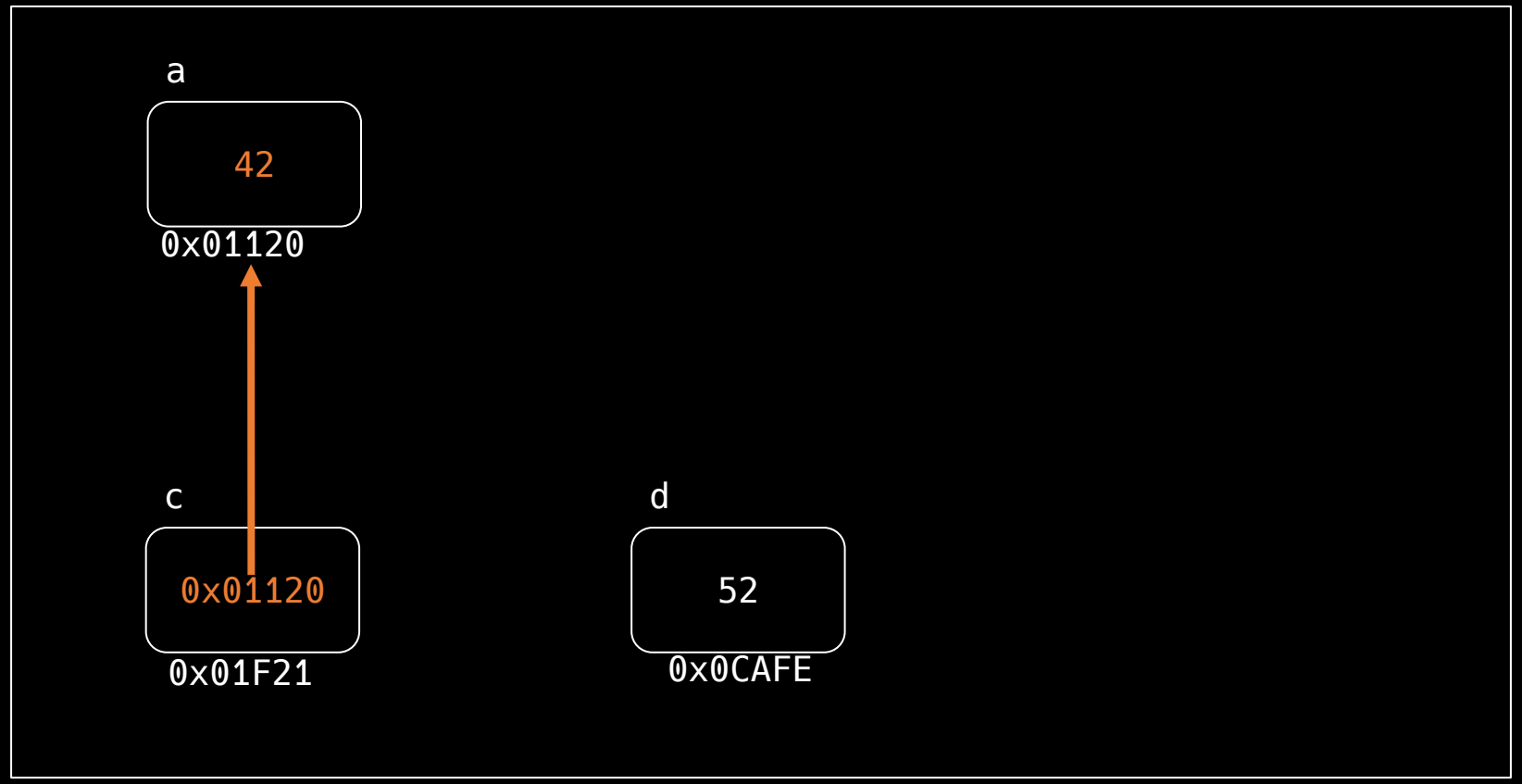
```
c = &a;
```

« Va chercher à l'adresse »



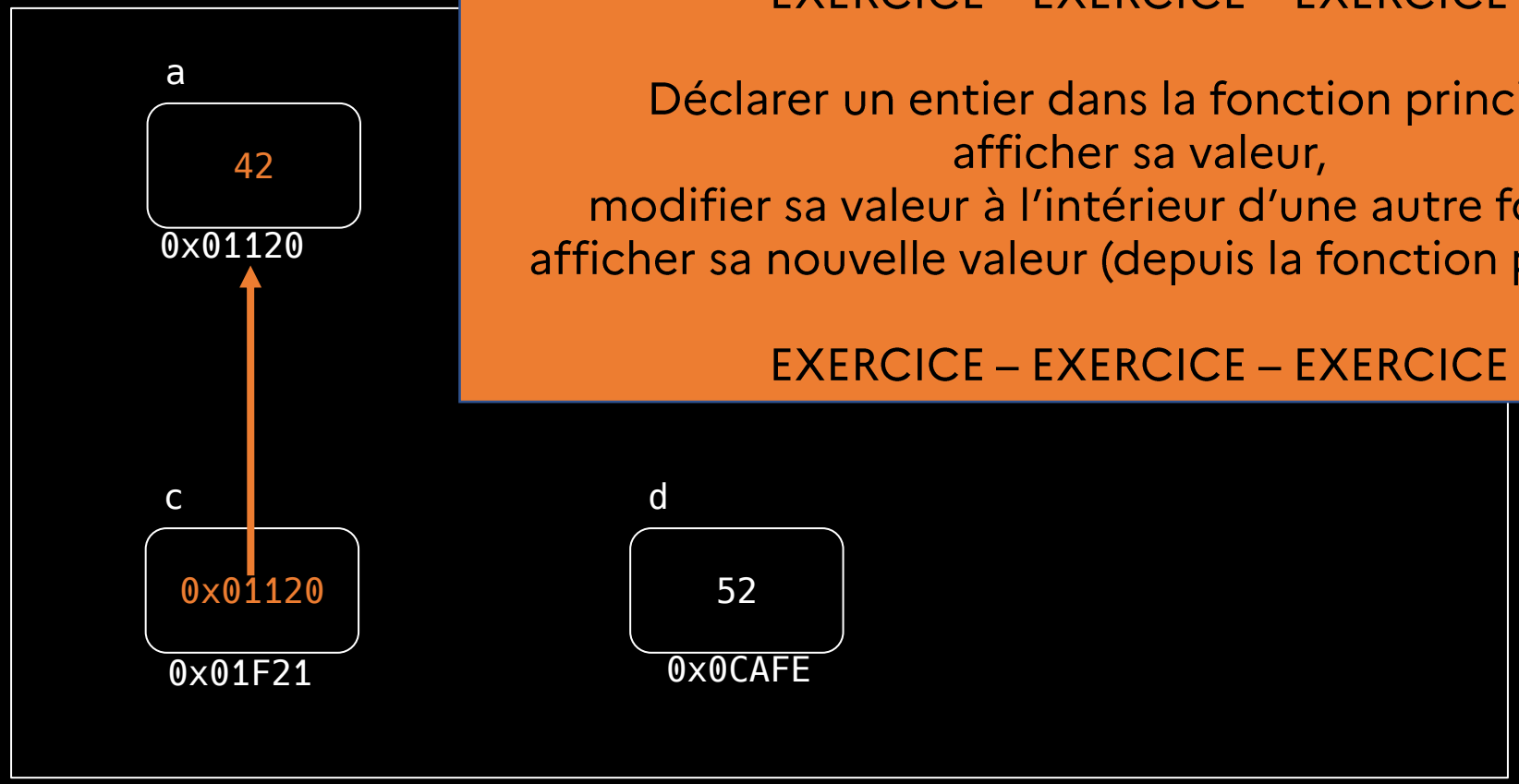

```
int d = *c;
```

« Va chercher à l'adresse »




```
*c = 42;
```

« Va chercher à l'adresse »



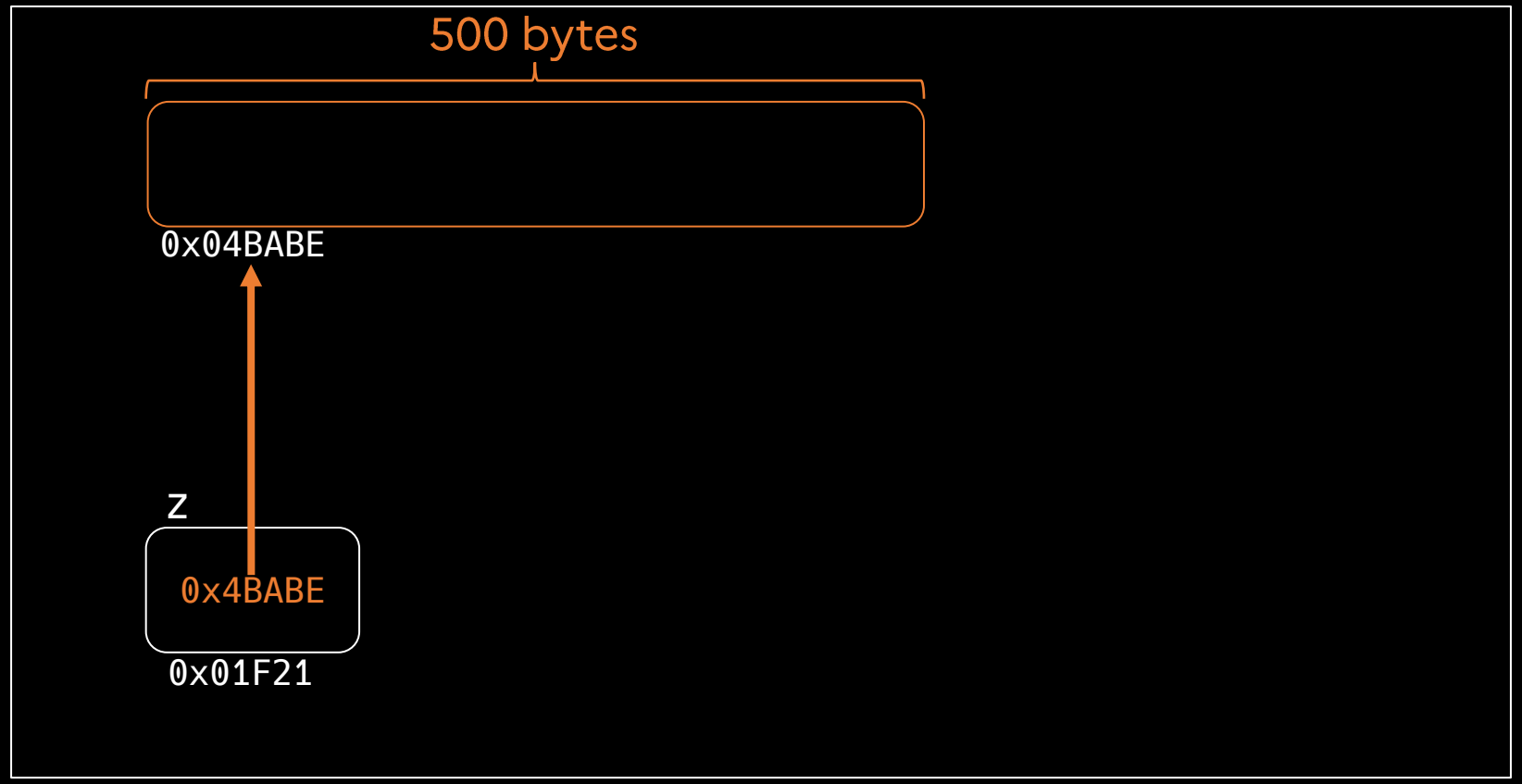
EXERCICE – EXERCICE – EXERCICE

Déclarer un entier dans la fonction principale,
afficher sa valeur,
modifier sa valeur à l'intérieur d'une autre fonction,
afficher sa nouvelle valeur (depuis la fonction principale).

EXERCICE – EXERCICE – EXERCICE

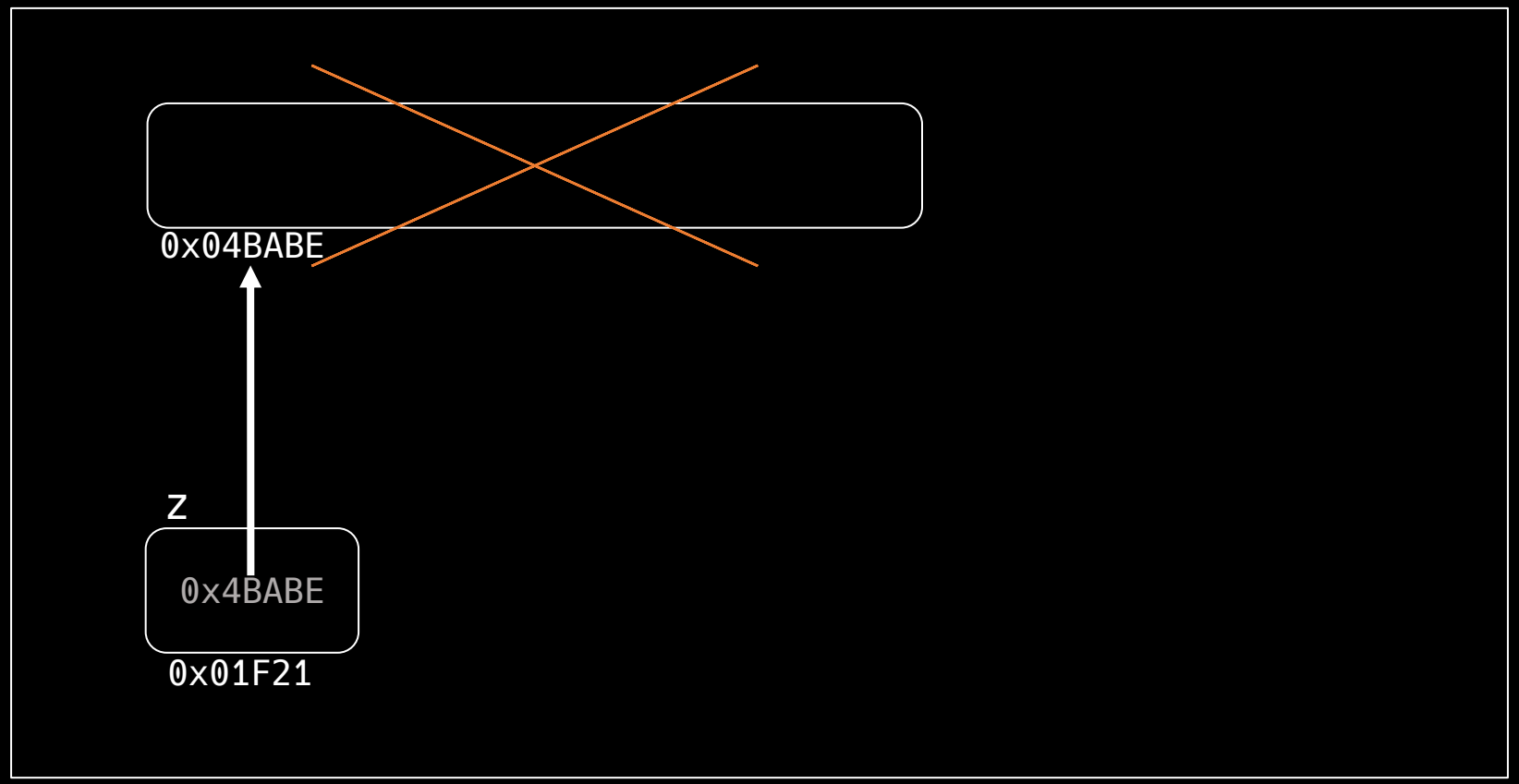
```
*c = 42;
```

« Donne-moi de la mémoire »



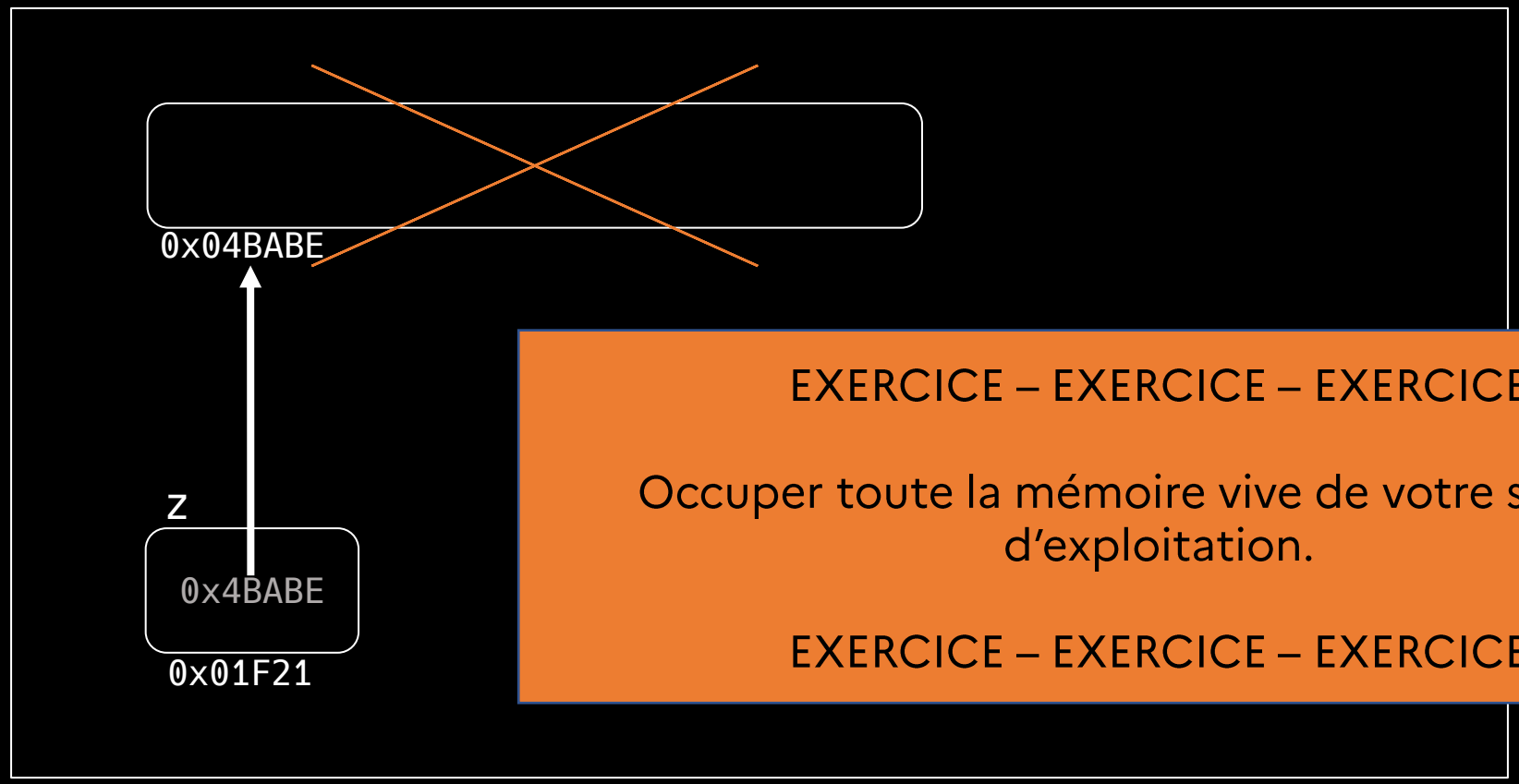

```
void *z = malloc(500);
```

« Je te rends ta mémoire »




```
free(z);
```

« Je te rends ta mémoire »



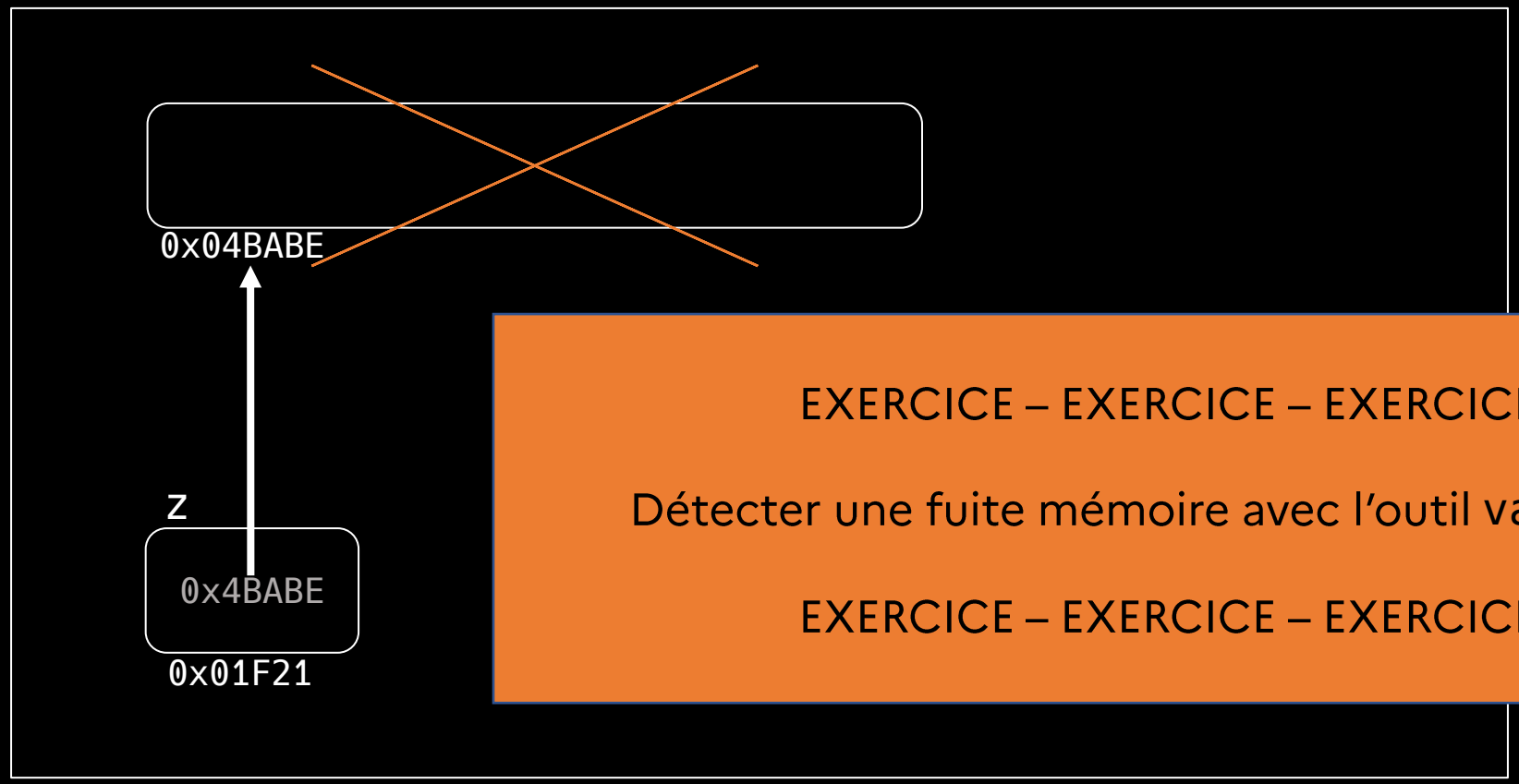
EXERCICE – EXERCICE – EXERCICE

Occuper toute la mémoire vive de votre système d'exploitation.

EXERCICE – EXERCICE – EXERCICE

```
free(z);
```

« Je te rends ta mémoire »



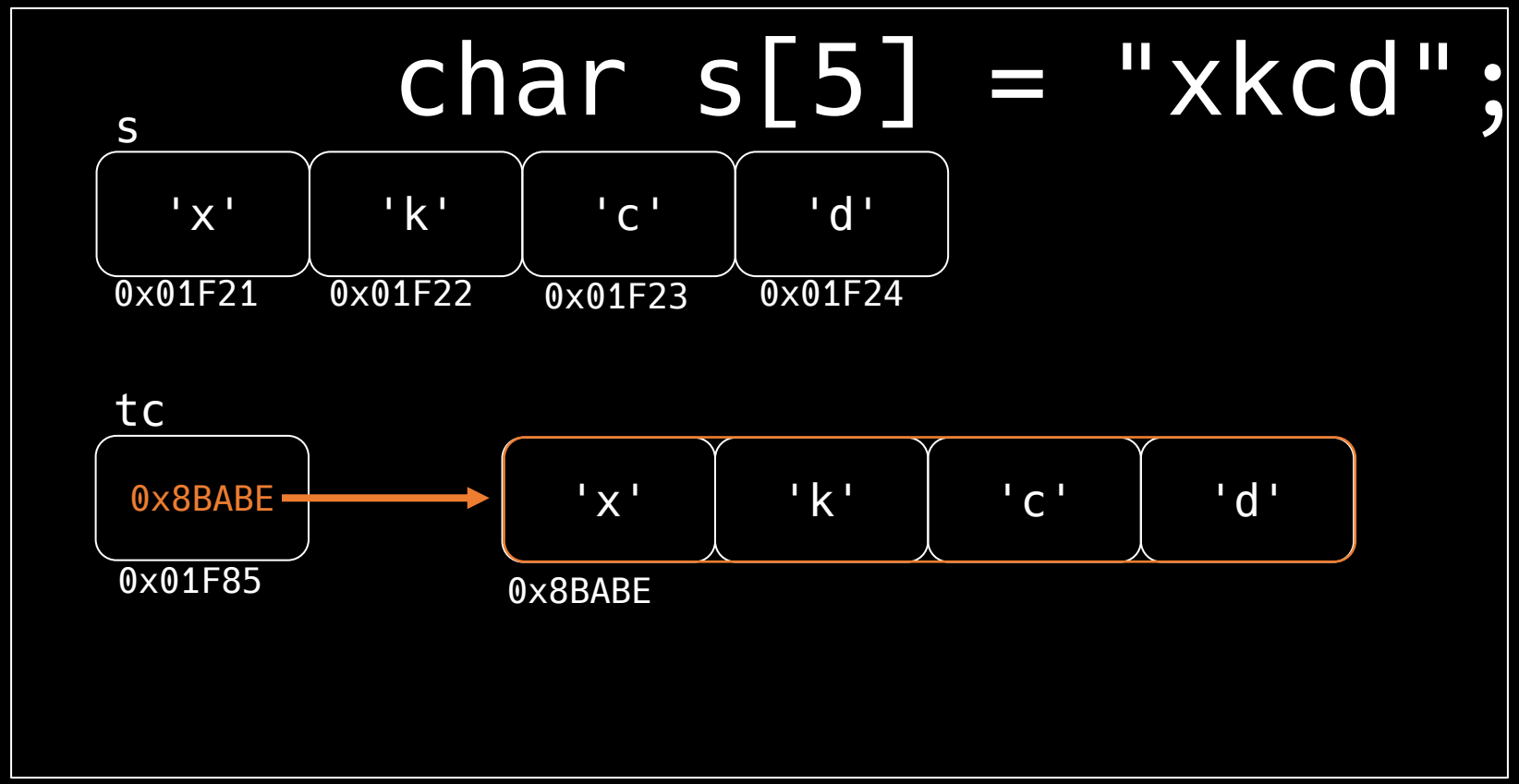
EXERCICE – EXERCICE – EXERCICE

Détecter une fuite mémoire avec l'outil `valgrind`.

EXERCICE – EXERCICE – EXERCICE

```
free(z);
```

Pointeur et liste



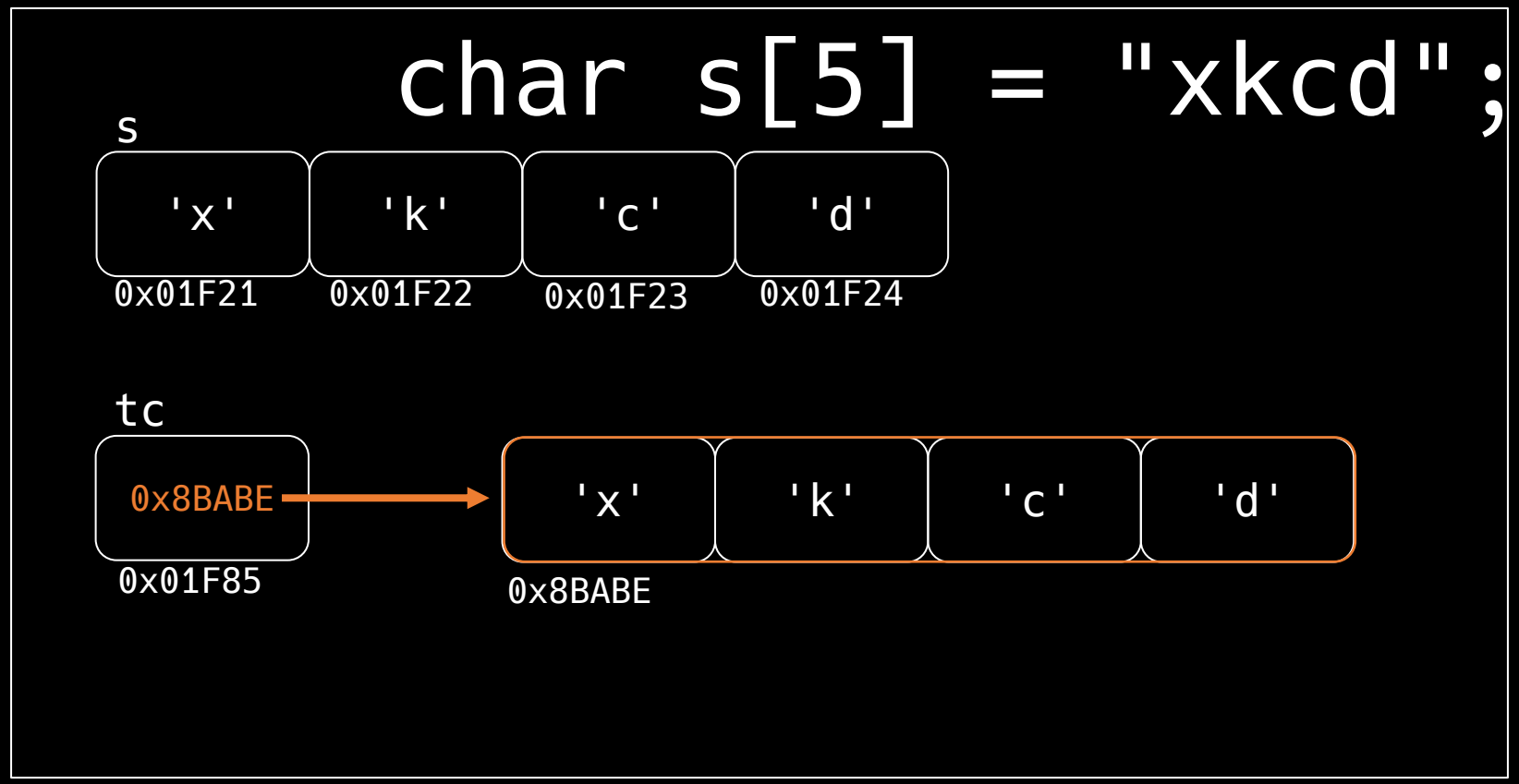

```
char *tc = malloc(4*sizeof(char));
```

Pointeur et liste

EXERCICE – EXERCICE – EXERCICE

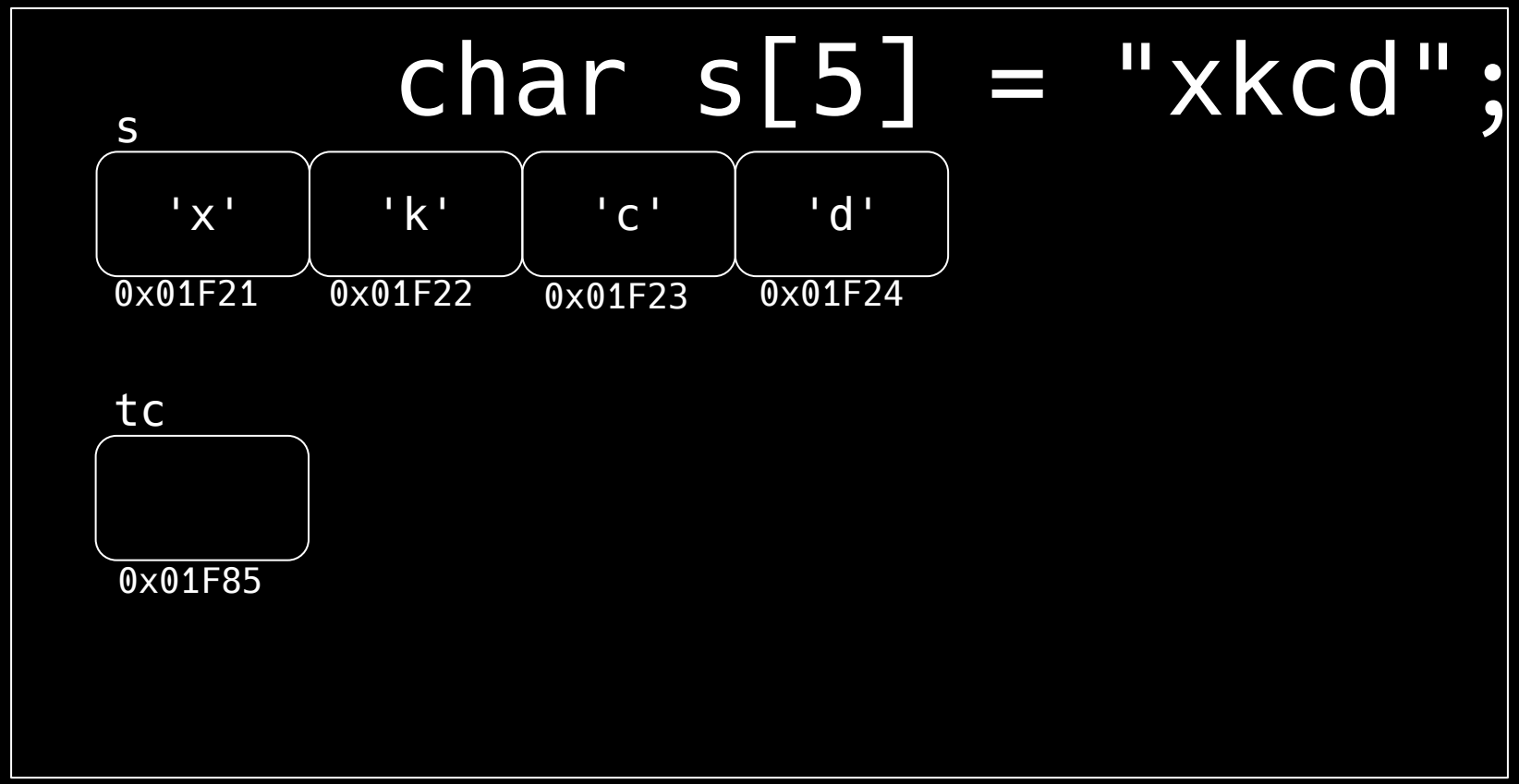
Déclarer une chaîne de caractères de manière dynamique.

EXERCICE – EXERCICE – EXERCICE



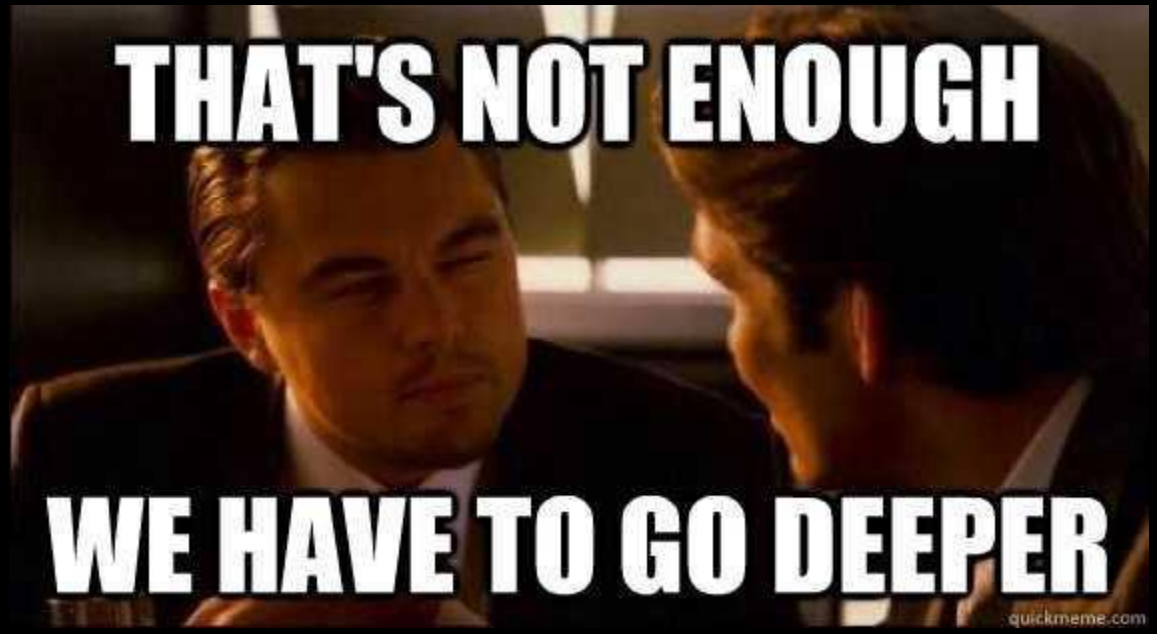

```
char *tc = malloc(4*sizeof(char));
```

Pointeur et liste

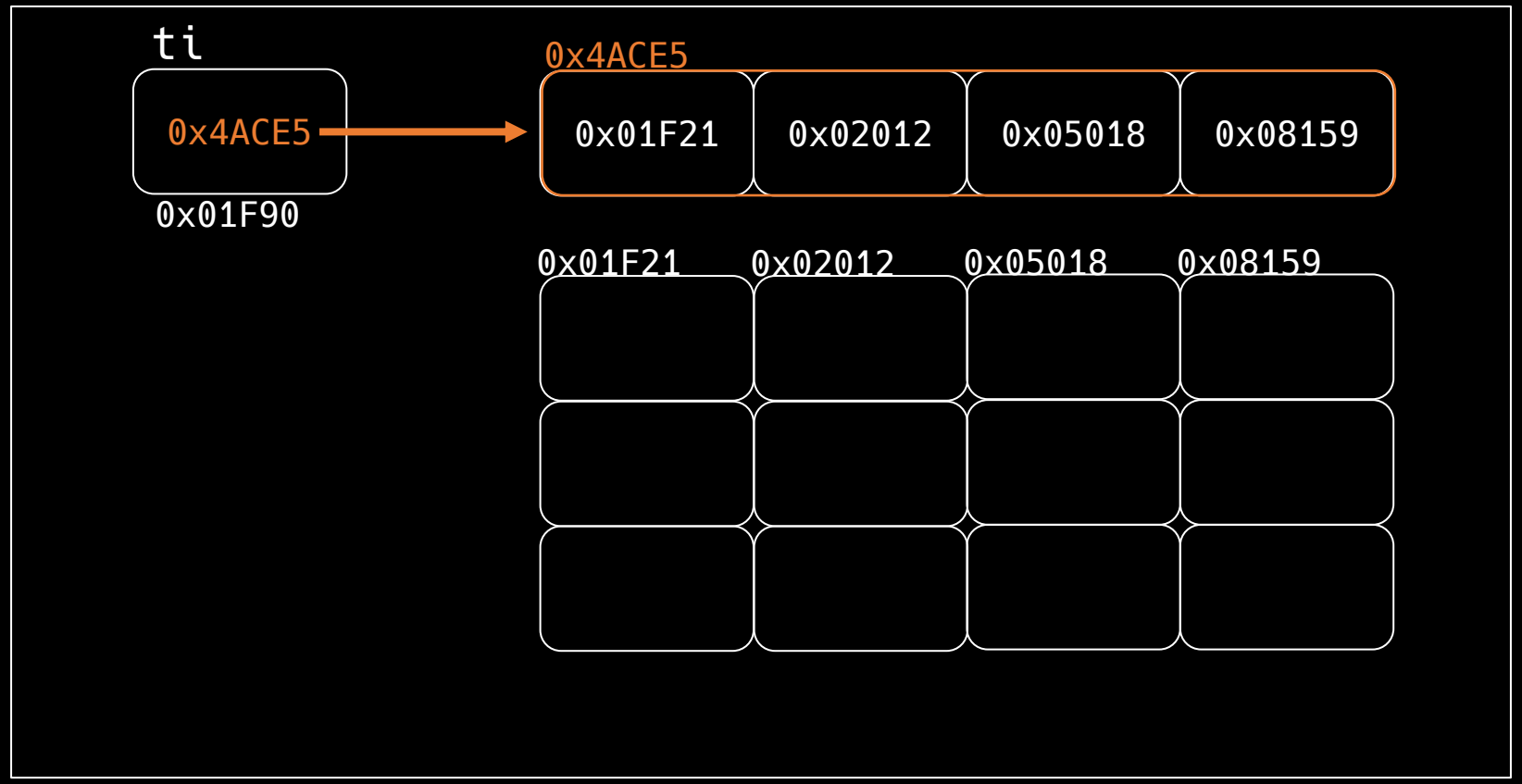



```
int *tc = malloc(4*sizeof(char));
```

Liste multidimensionnelle

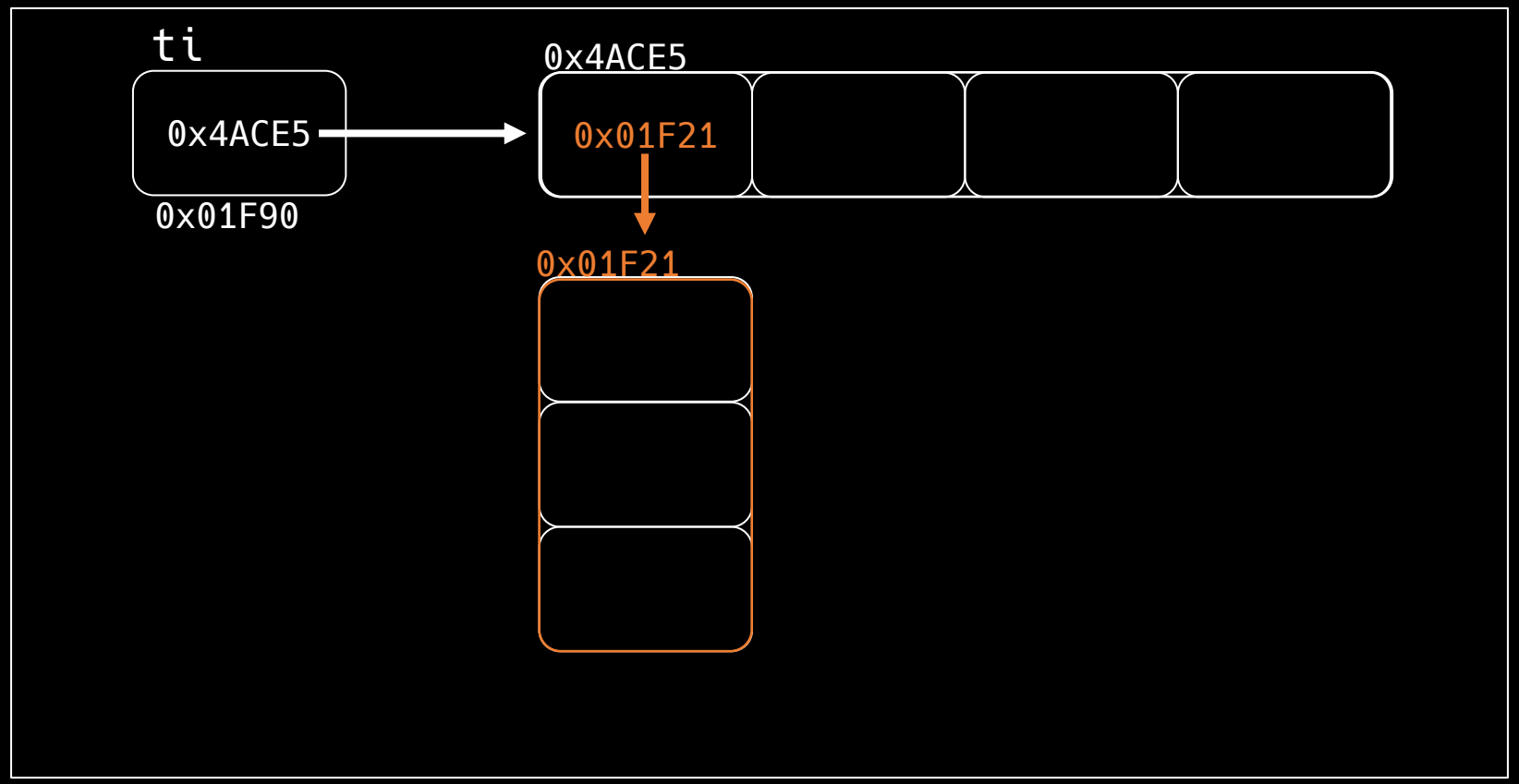


Liste multidimensionnelle



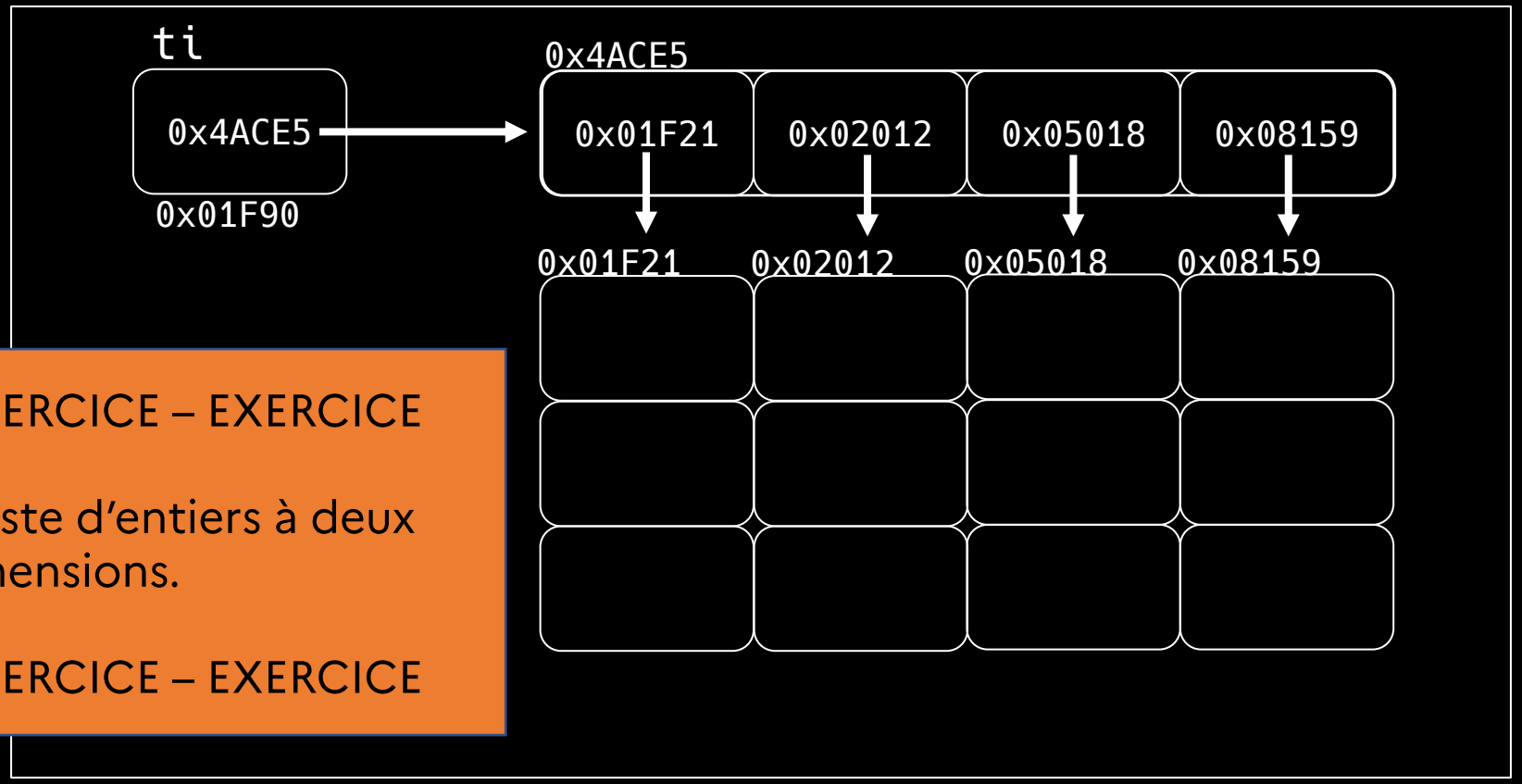

```
int **ti = malloc(4*sizeof(int*));
```

Liste multidimensionnelle




```
ti[0] = malloc(3*sizeof(int));
```

Liste multidimensionnelle



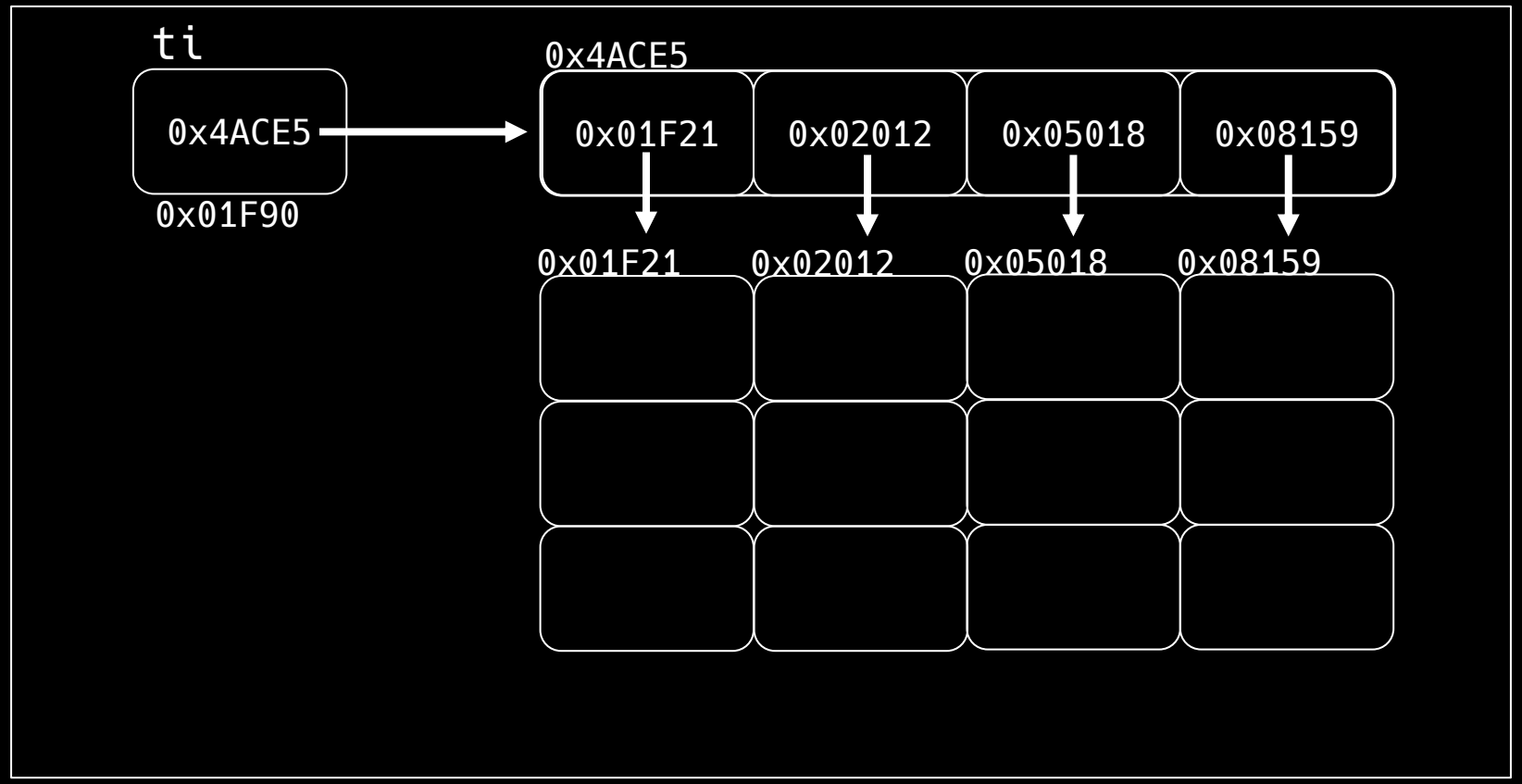
EXERCICE – EXERCICE – EXERCICE

Déclarer une liste d'entiers à deux dimensions.

EXERCICE – EXERCICE – EXERCICE

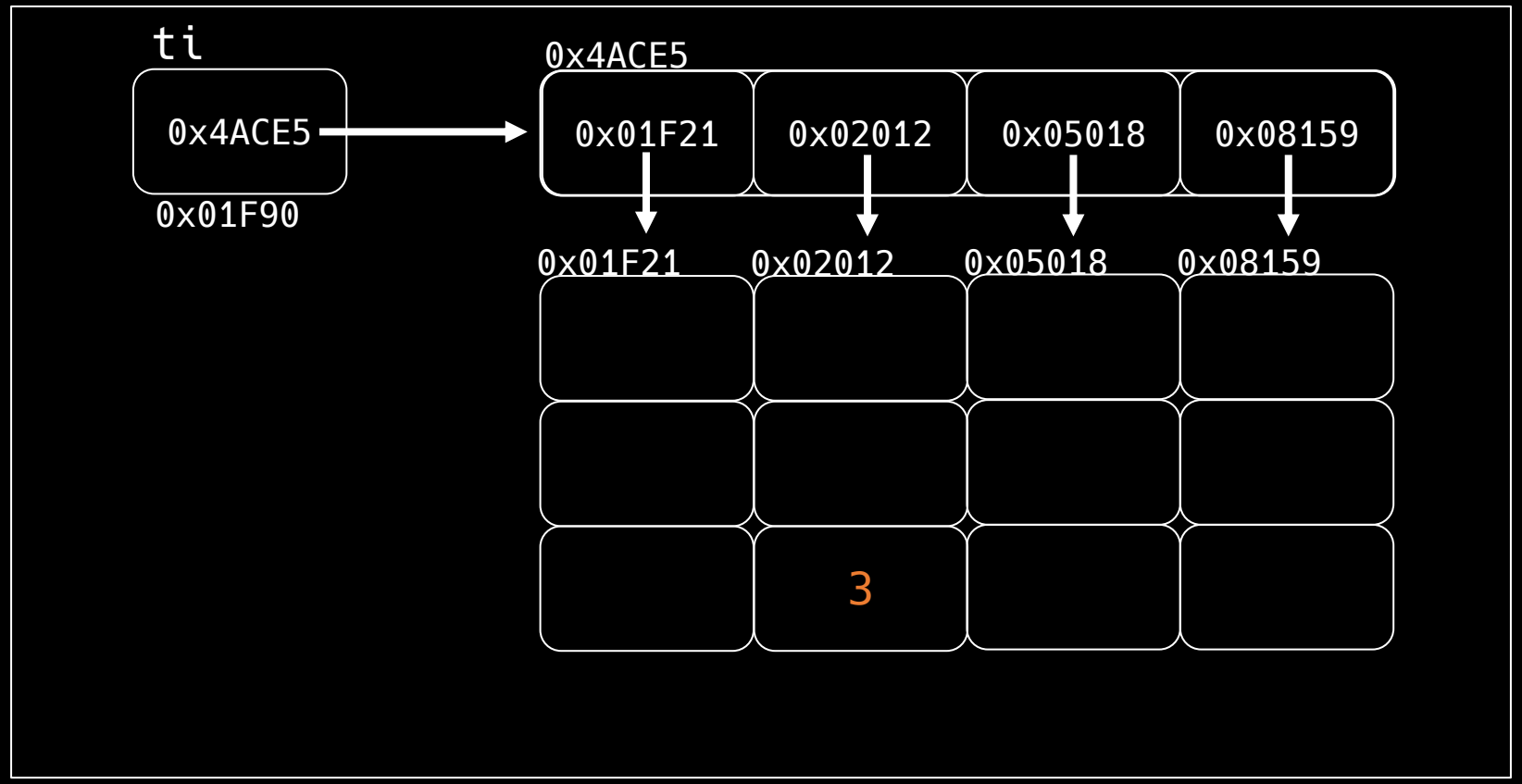
```
ti[i] = malloc(3*sizeof(int));
```

Liste multidimensionnelle



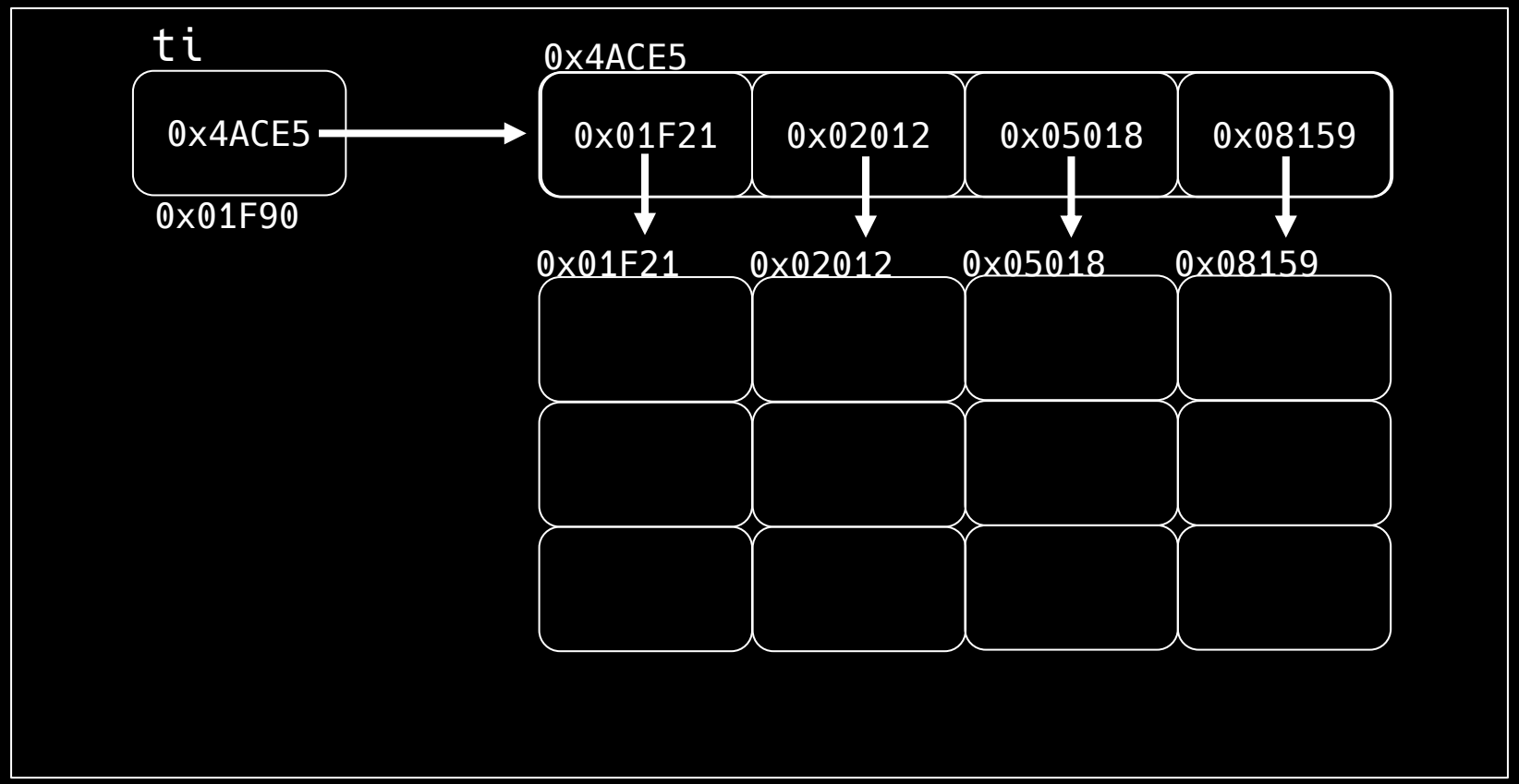

```
ti[1][2] = 3
```

Liste multidimensionnelle



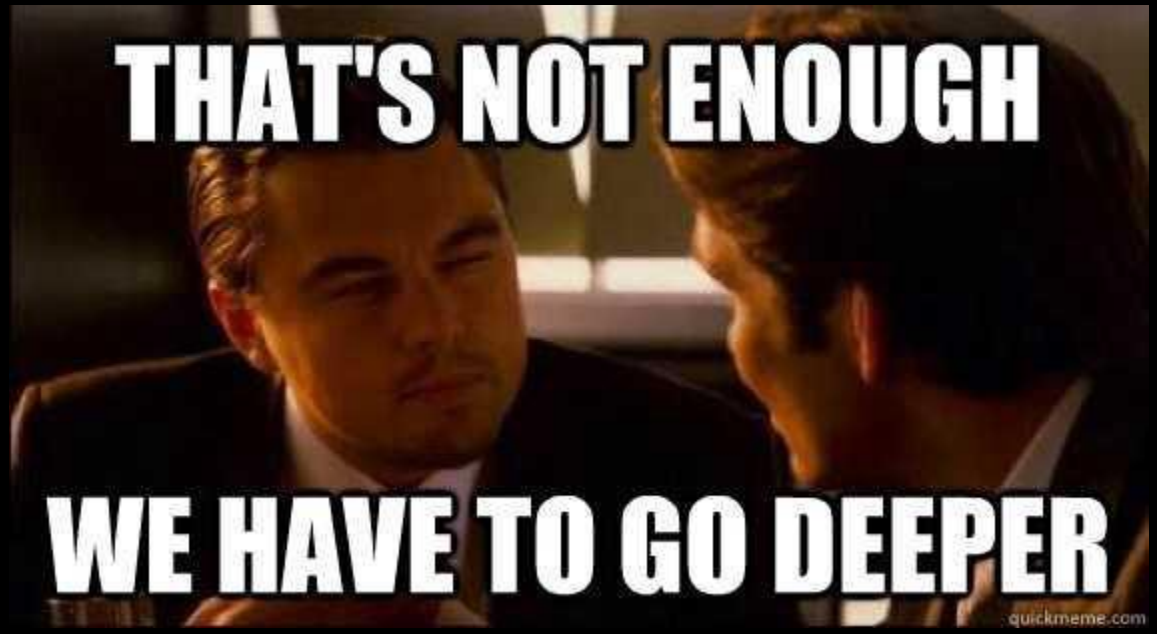

```
ti[1][2] = 3
```

Liste multidimensionnelle



`free(?)`

Liste multidimensionnelle

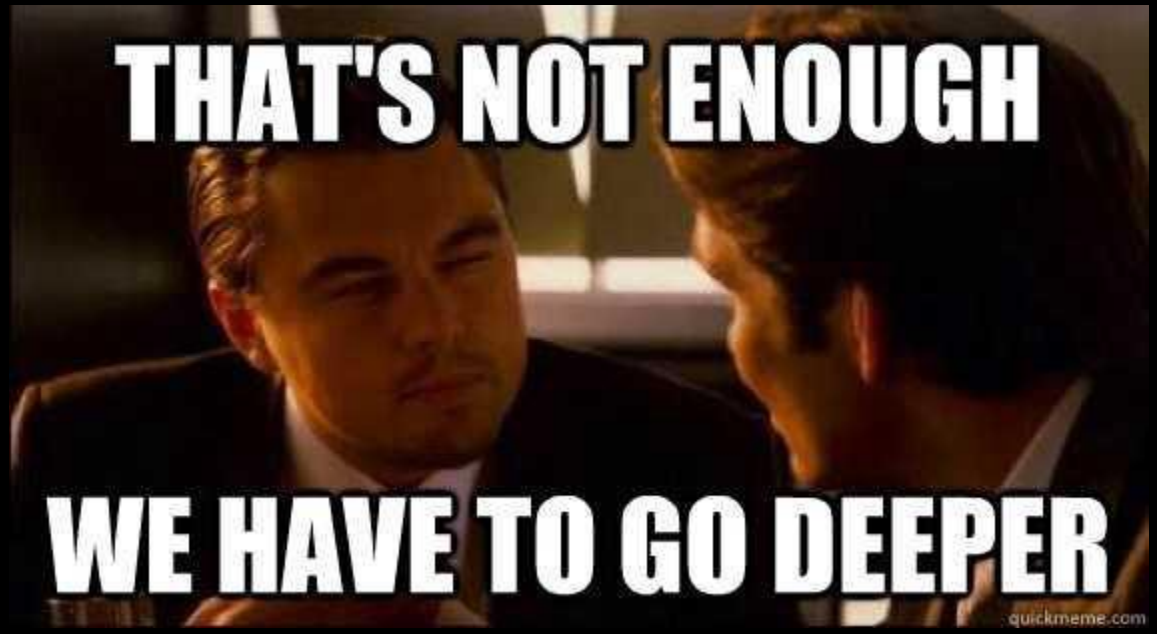


Liste multidimens

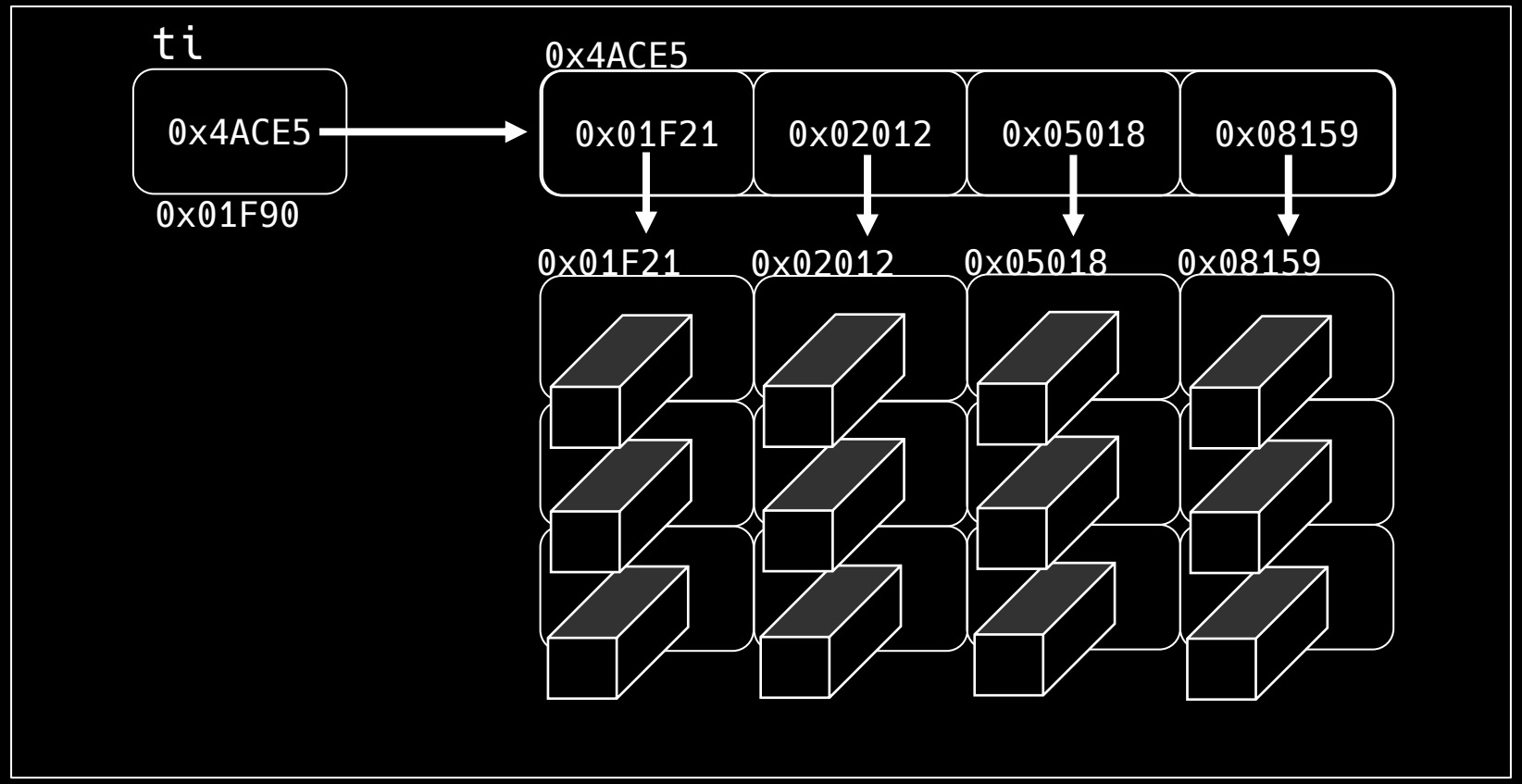
EXERCICE – EXERCICE – EXERCICE

Déclarer une liste d'entiers à trois dimensions.

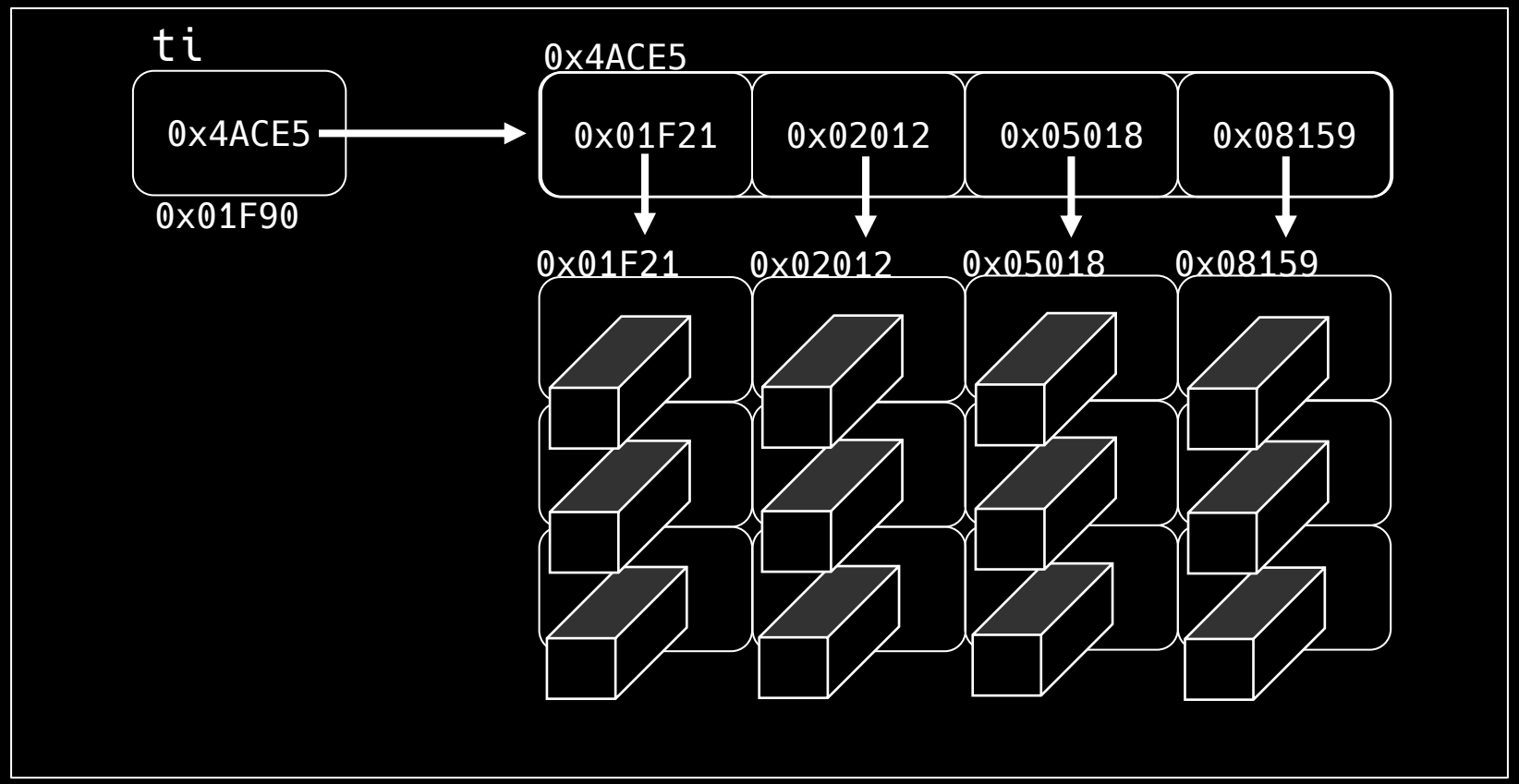
EXERCICE – EXERCICE – EXERCICE



Liste multidimensionnelle

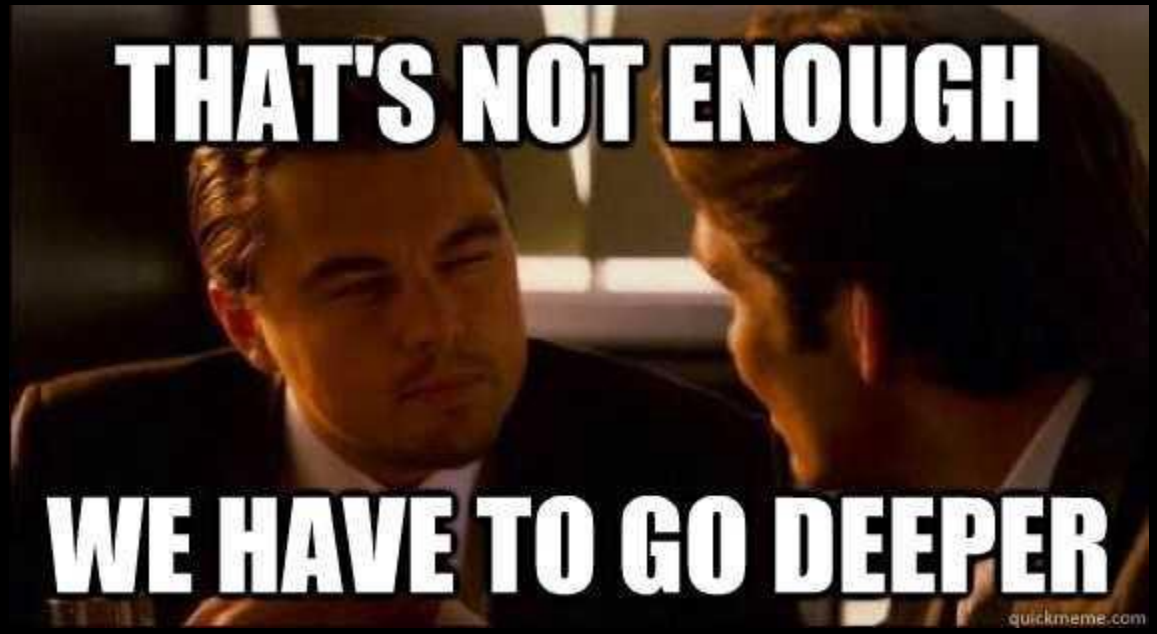


Liste multidimensionnelle

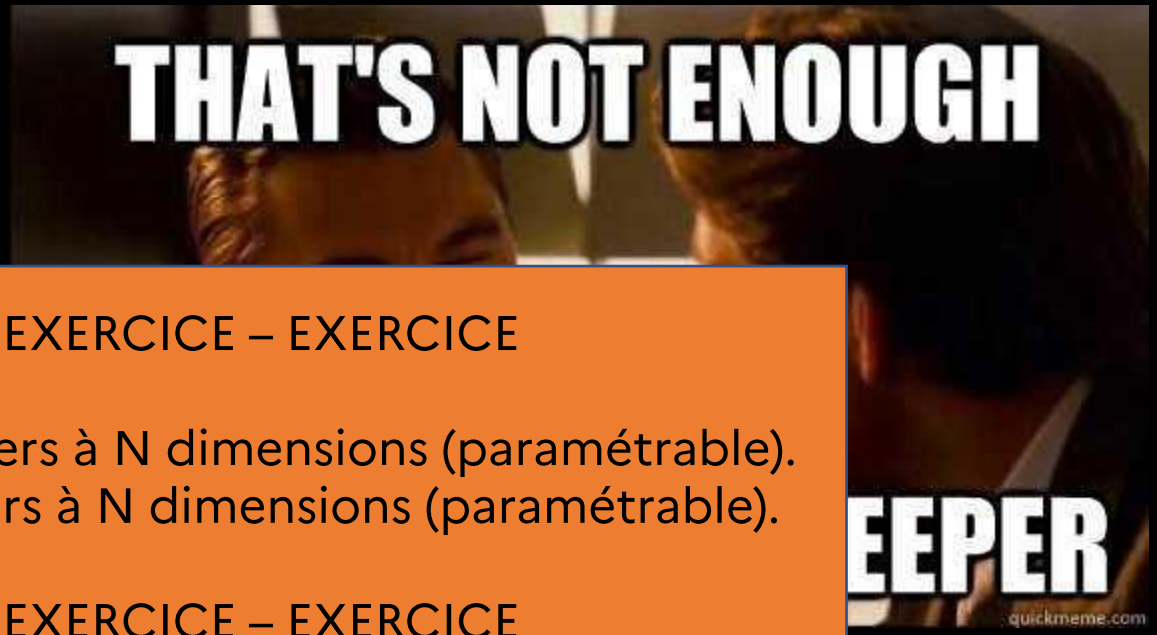


`free(?)`

Liste multidimensionnelle



Liste multidimensionnelle



EXERCICE – EXERCICE – EXERCICE

Déclarer une liste d'entiers à N dimensions (paramétrable).
Libérer une liste d'entiers à N dimensions (paramétrable).

EXERCICE – EXERCICE – EXERCICE

Les structures

L’ancêtre des objets

Création d'une structure

```
struct Car {
    char *name;
    int nbPortes;
}



struct Car myCar;
```

Création d'une structure

```c
struct Car {
    char *name;
    int nbPortes;
}
typedef struct Car MyCarType;
```

```c
typedef int my_int;
```

```c
struct Car myCar;
MyCarType myCar;
```

Création d'une structure

```c
struct Car {
    char *name;
    int nbPortes;
}
typedef struct Car MyCarType;
```

```c
typedef struct Car {
    char *name;
    int nbPortes;
} MyCarType;
```

```c
struct Car myCar;
MyCarType myCar;
```

Utiliser une structure

```
typedef struct car_
{
    char *name;
    int nbPortes;
} car;
```

```
car my_car;
my_car.name = "clio";
my_car.nbPortes = 5;
```

```
car *my_car;
my_car = malloc(sizeof(car));
my_car->name = "clio";
my_car->nbPortes = 5;
```

Utiliser une structure

```
typedef struct car_
{
    char *name;
    int nbPortes;
} car;
```

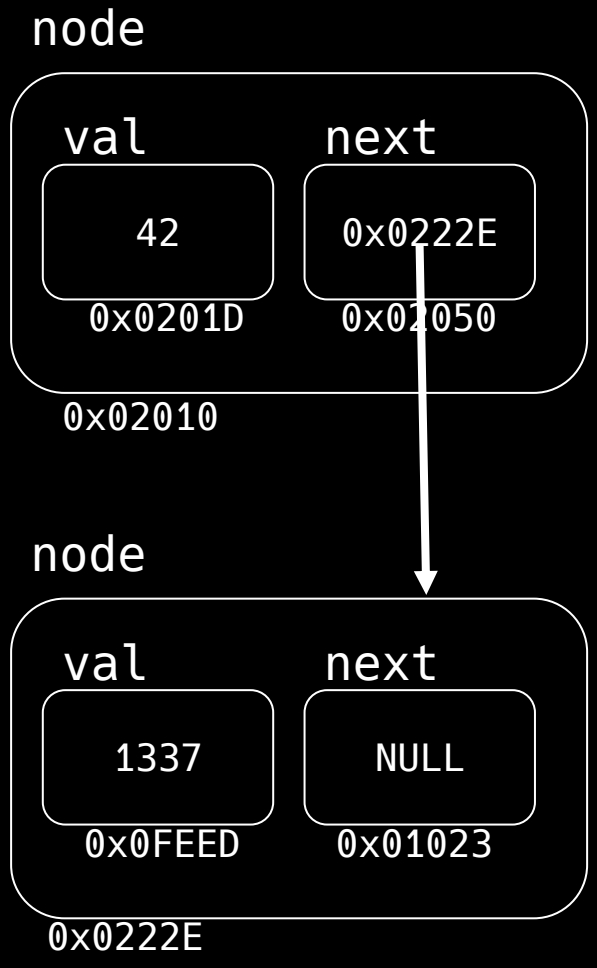
```
car my_car;
my_car.name = "clio";
my_car.nbPortes = 5;

car *my_car;
my_car = car_constructor();
my_car->name = "clio";
my_car->nbPortes = 5;
car_destructor(car);
```

Utilité des structures de données

- (meta-)Structures à tailles variables
- Graphes

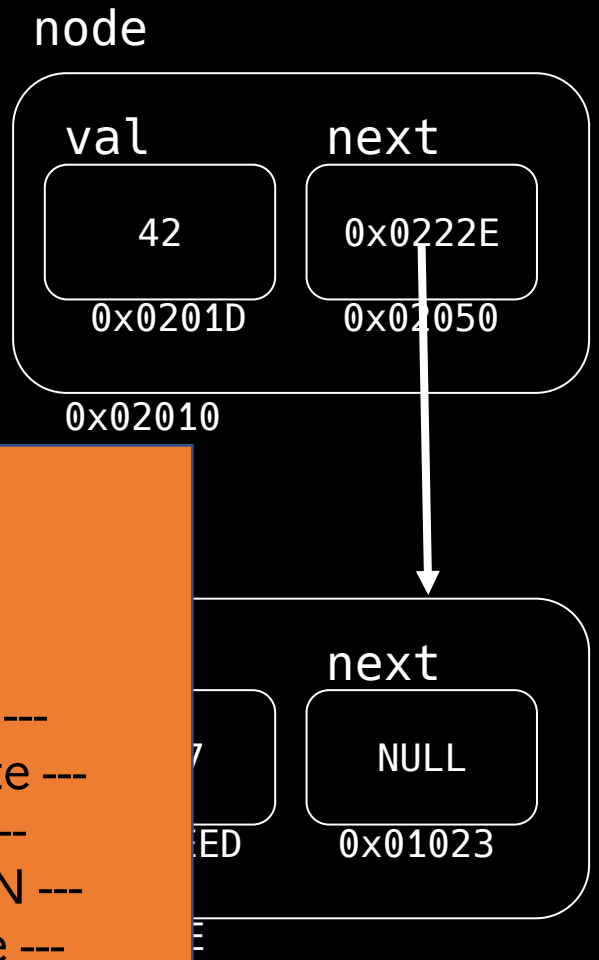
Exemple : liste chaînée

```
typedef struct node {
    int val;
    struct node * next;
} node_t;
```



Exemple : liste chaînée

```
typedef struct node {
    int val;
    struct node * next;
} node_t;
```



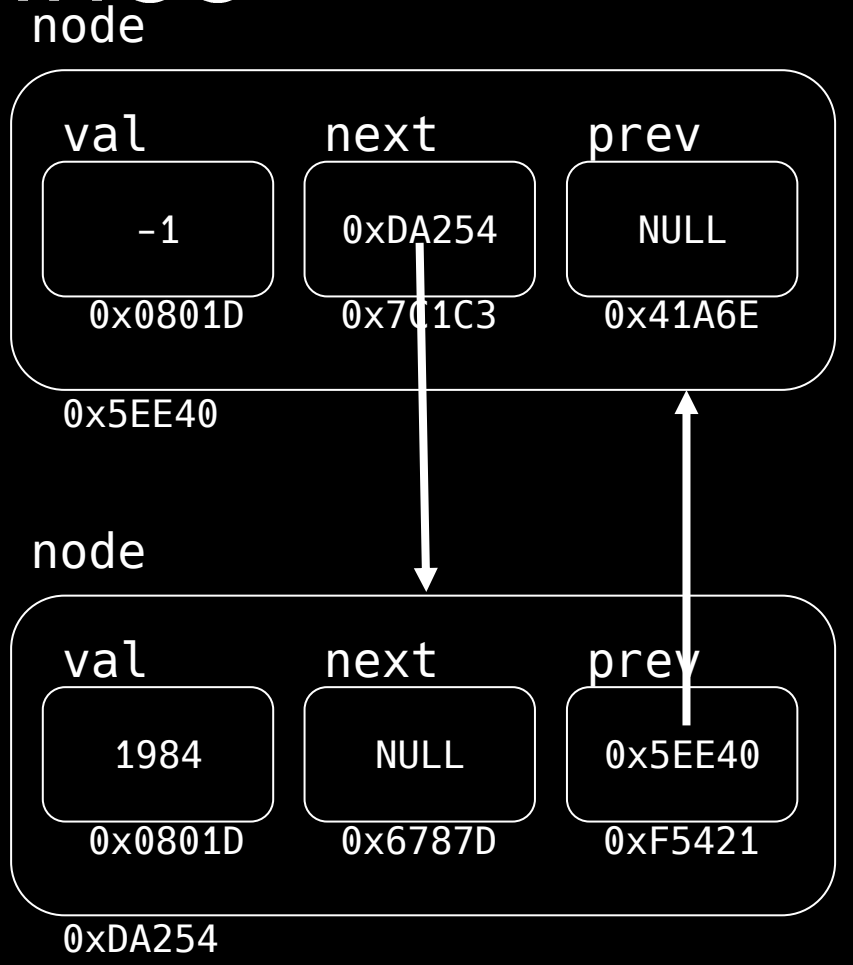
EXERCICE – EXERCICE – EXERCICE

Créer les fonctions de :
--- création d'une liste chaînée ---
--- insertion d'un élément en fin de liste ---
--- insertion d'un élément en début de liste ---
--- lecture d'un élément en position N ---
--- suppression d'un élément en position N ---
--- mesure d'une taille d'une liste chaînée ---

EXERCICE – EXERCICE – EXERCICE

Exemple : liste double chaînée

```
typedef struct node {
    int val;
    struct node * next;
    struct node * prev;
} node_t;
```



Exemple : arbre binaire de recherche

```
typedef struct node {
    int val;
    struct node * a;
    struct node * b;
} node_t;
```