# REPRISE DES CONCEPTS

# DÉMO

# APPEL DE FONCTIONS

- Lorsque l'on appelle une fonction, plusieurs valeurs peuvent lui être envoyées en fonction du nombre de paramètres que la fonction attends. Par exemple:

```
func multiplication(var param1 , var param2):
    var result = param1 * param2
    return result
```

- La fonction multiplication prend deux paramètres en entrée : param1 et param2

# APPEL DE FONCTIONS

- L'appel à la fonction « multiplication » peut se faire de plusieurs manières:

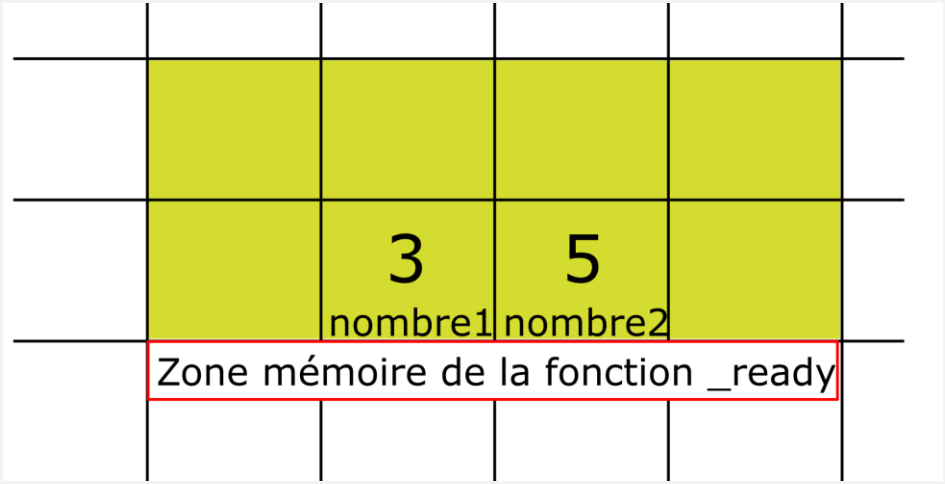
```
# appel de la fonction multiplication
# envoie respectif des valeurs 5 et 3
# récupération du résultat dans result
var result = multiplication(5, 3)
# d'autres exemples
var nombre = 4
result = multiplication(nombre, 3)
result = multiplication(4, nombre)
var nombre1 = 3
var nombre2 = 5
result = multiplication(nombre1, nombre2)
# Question : que se passe t-il pour l'appel suivant??
result = multiplication(multiplication(nombre1, nombre2), nombre2)
```

# APPEL DE FONCTIONS

- Mais alors que ce se passe t-il lorsqu'on appelle la fonction multiplication??? Prenons un des exemples de la slide précédentes.

```
var nombre1 = 3
var nombre2 = 5
result = multiplication(nombre1, nombre2)
```

- Ici deux variables sont déclarées : nombre1 et nombre2. La déclaration de ces deux variables consiste à demander à la mémoire de l'ordinateur de réserver une case mémoire pour chacune de ces variables afin d'y stocker respectivement la valeur 3 pour nombre1 et la valeur 5 pour nombre2.

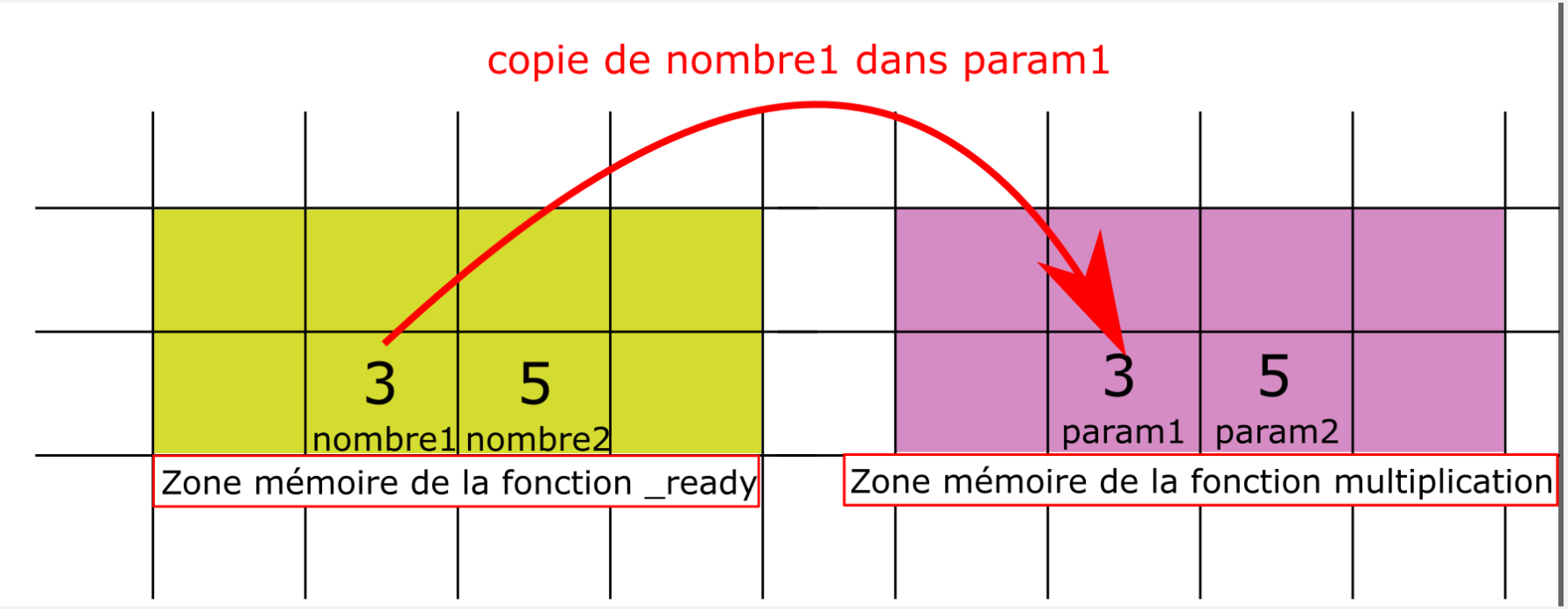


# APPEL DE FONCTIONS

```gdscript
var nombre1 = 3
var nombre2 = 5
result = multiplication(nombre1, nombre2)
```

```gdscript
func multiplication(var param1 , var param2):
    var result = param1 * param2
    return result
```

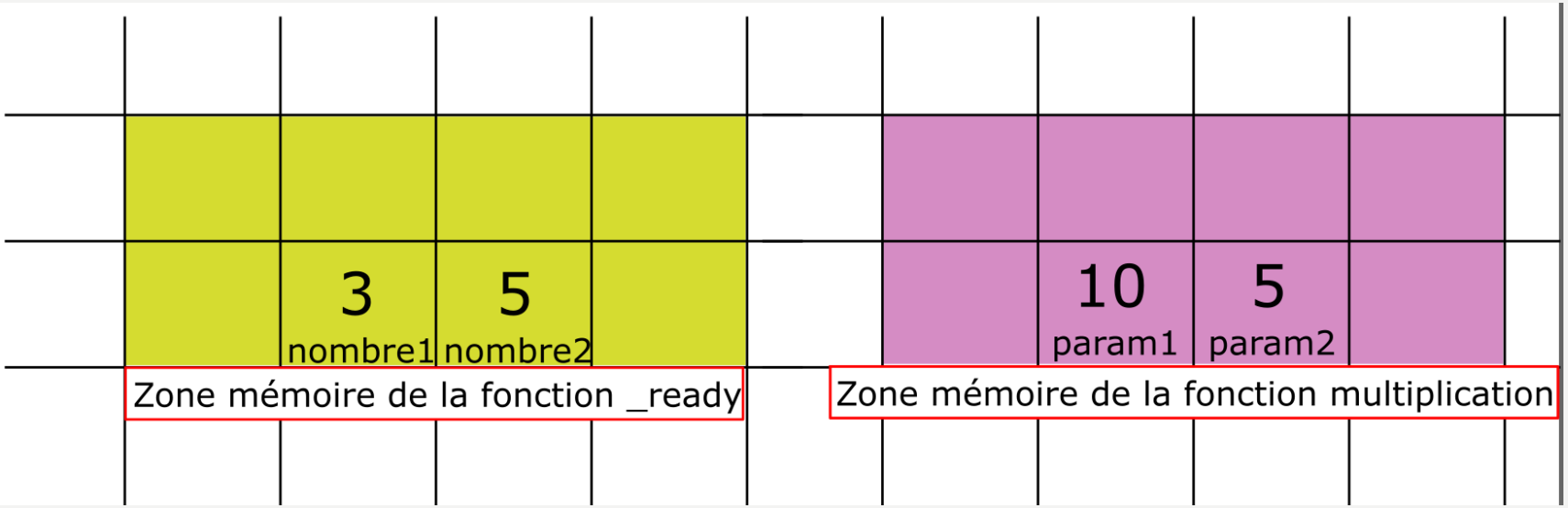
- L'appel à la fonction multiplication consiste alors à **copier** les valeurs contenues dans nombre1 et nombre2 respectivement aux paramètres d'entrée de la fonction multiplication, respectivement param1 et param2. C'est ce qu'on appelle **le passage par valeur** à la fonction multiplication.



# APPEL DE FONCTIONS

```
func multiplication(var param1 , var param2):
    param1 = 10
    var result = param1 * param2
    return result
```

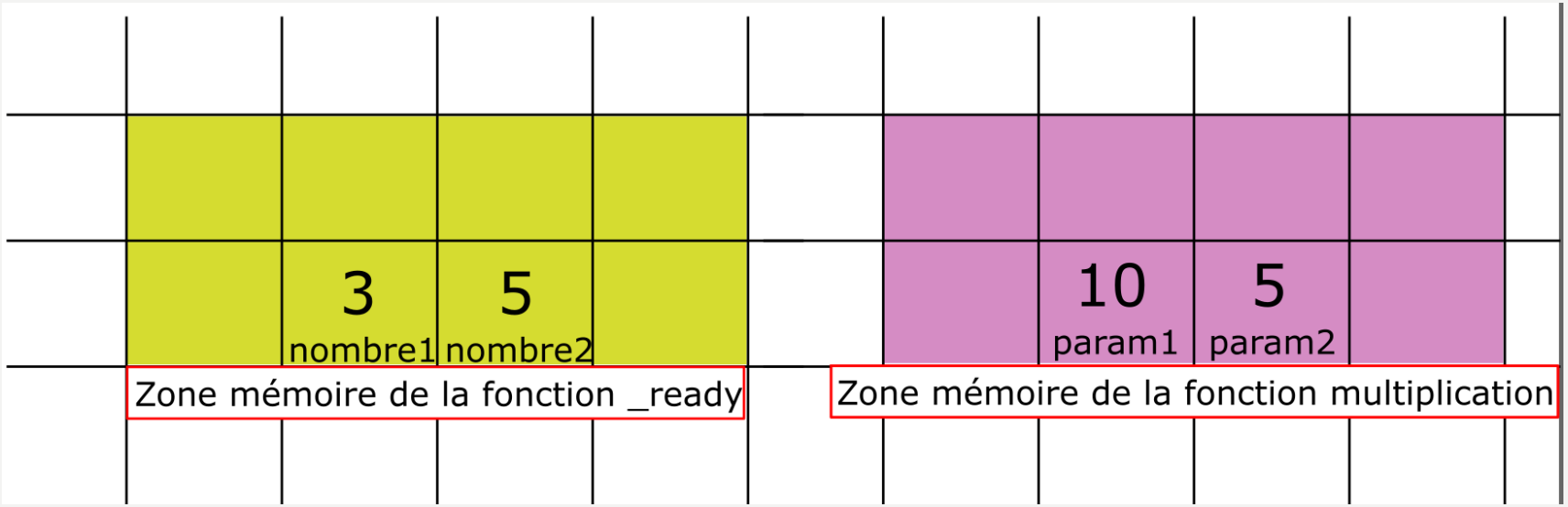
- Ce qui signifie que si param1 est modifiée dans la fonction multiplication, cela n'aura aucun impact pour nombre1, puisque les deux variables sont à deux endroits différents de la mémoire. Exemple ci-dessous, lorsque la valeur 10 est donnée à param1 dans la fonction multiplication.



# APPEL DE FONCTIONS

```
func multiplication(var param1 , var param2):
    param1 = 10
    var result = param1 * param2
    return result
```

- On peut remarquer que le bloc mémoire utilisée par la fonction _ready est différent du bloc mémoire de la fonction multiplication (zones verte et violette). C'est ce qu'on appel le « local scope » de chaque fonction. La fonction_ready ne peut pas accéder aux variables dans la zone mémoire de la fonction multiplication et inversement.



# PASSAGE PAR VALEUR

- Ce système de **passage par valeur** est valable seulement pour un certain nombre de types de variables, que l'on appelle générique. En voici la liste :

- bool
- int
- float
- String
- Vector2
- Rect2
- …

- Pour avoir la liste complète, voici le lien de référence : https://docs.godotengine.org/en/stable/getting_started/scripting/gdscript/gdscript_basics.html#built-in-types
- **Mais ce passage par valeur ne s'applique pas à tous les types!**

# PASSAGE PAR REFERENCE

- Un autre système est appelé le passage par référence. Cela signifie que si une variable qui n'est pas de type « built-in » (bool, int, float, Vector2…), elle n'est pas copiée lorsqu'elle est envoyée à un paramètre d'une fonction.
- Le tableau est un exemple de variable qui n'est pas du type « built-in »
- Essayons de comprendre alors comment le tableau est envoyée à une fonction si il n'est pas copié.

```
func modifier_tableau(var tableau):
    tableau[0] = 5

func _ready():
    var tableau = [0, 3, 5, 2]
    modifier_tableau(tableau)
    print(tableau)
```

- Dans l'exemple au dessus, un tableau est envoyé à la fonction « modifier_tableau » pour pouvoir affecter la valeur 5 au premier élément du tableau

# PASSAGE PAR REFERENCE

```
func modifier_tableau(var tableau):
    tableau[0] = 5

func _ready():
    var tableau = [0, 3, 5, 2]
    modifier_tableau(tableau)
    print(tableau)
```

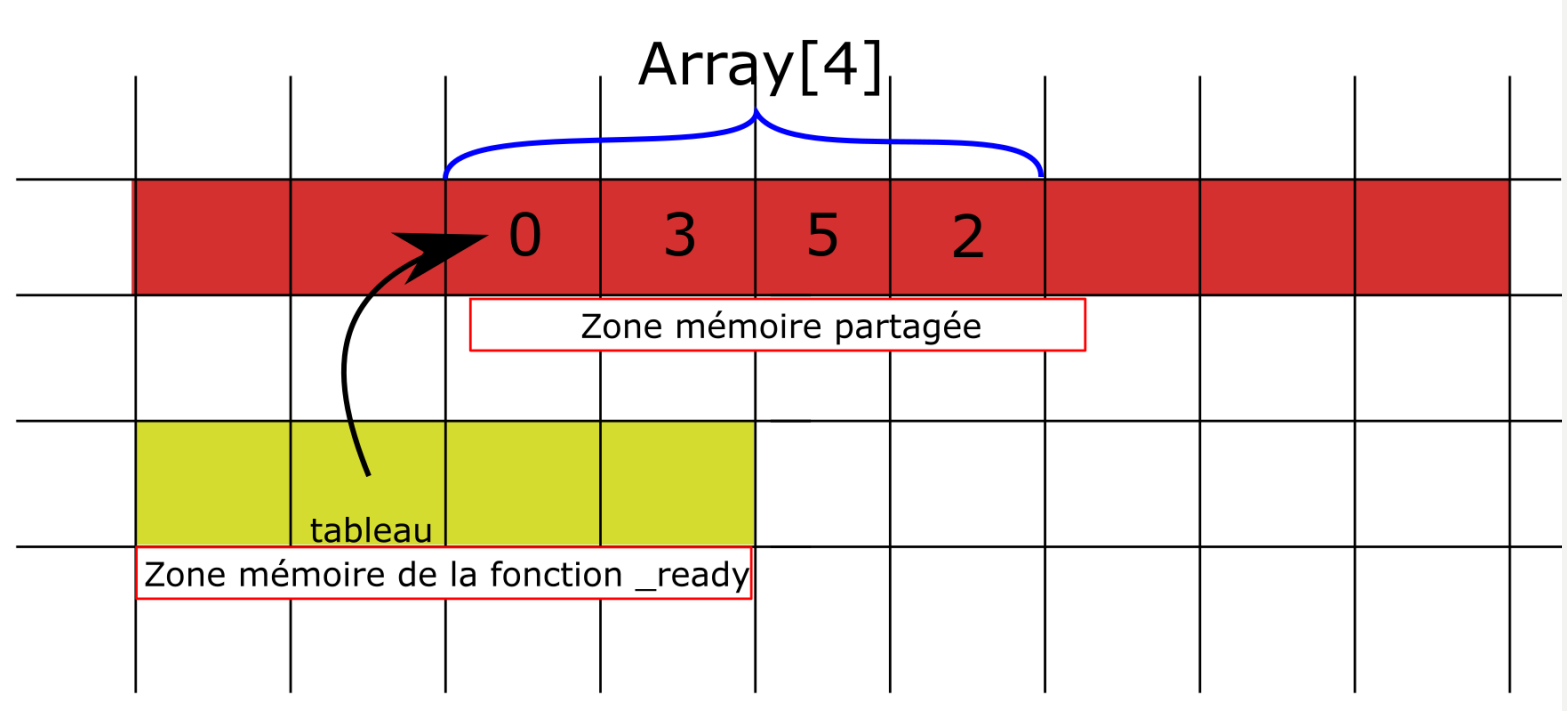
- Voici le résultat de ce morceau de code: `[5, 3, 5, 2]`
- On voit bien que le premier élément a été modifié, donc cela confirme bien qu'en modifiant le tableau dans la fonction « modifier_tableau », on modifie le même tableau que l'on avait créé dans la fonction _ready.
- Mais alors que se passe t-il en mémoire??

# PASSAGE PAR REFERENCE

- J'ai modifié le nom de la variable paramètre de la fonction modifier_tableau pour bien montrer que ce sont deux variables différentes entre tableau et param1.
- Regardons la mémoire maintenant au moment de la création du tableau:

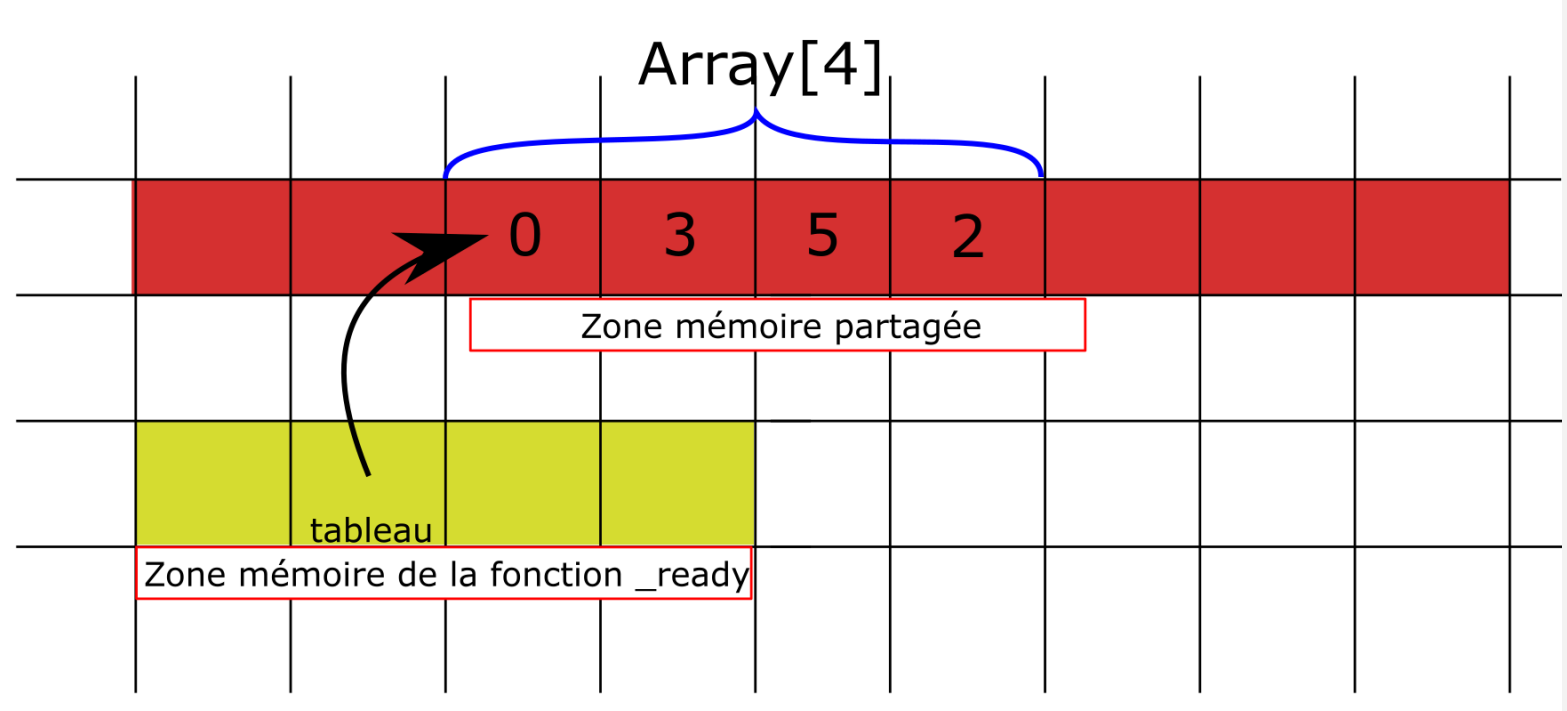
```gdscript
func modifier_tableau(var param1):
    param1[0] = 5

func _ready():
    var tableau = [0, 3, 5, 2]
    modifier_tableau(tableau)
    print(tableau)
```



# PASSAGE PAR REFERENCE

- Le tableau est créé dans une zone mémoire spéciale appelée **la mémoire partagée. Cette zone mémoire est accessible par toutes les fonctions!**
- Ici la variable tableau est créé dans _ready, mais le contenu (c'est-à-dire le array[4]) est stockée dans la mémoire partagée. La variable tableau ne contient alors que le chemin pour accéder à la première case mémoire du Array[4] dans la mémoire partagée.

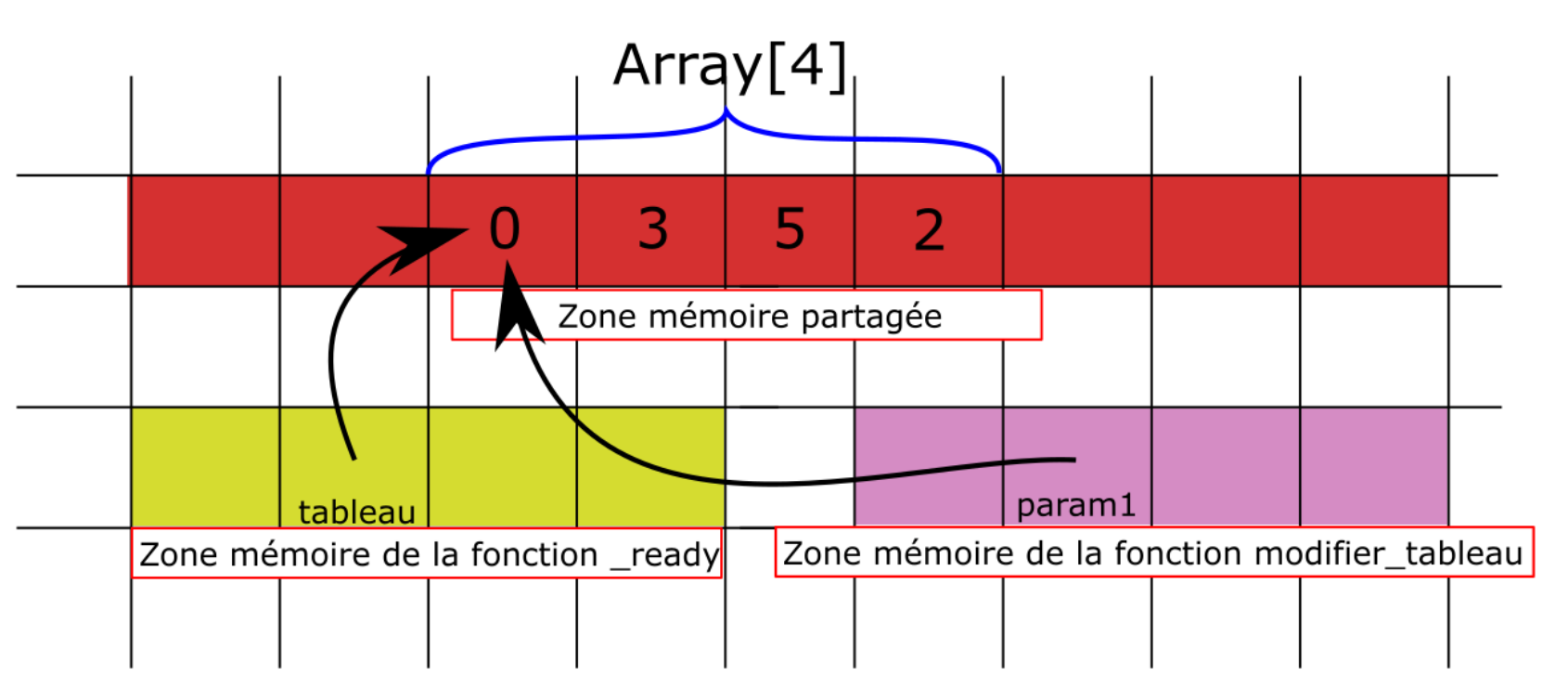


# PASSAGE PAR REFERENCE

- Maintenant regardons ce qu'il se passe lorsque la variable tableau est passée à la fonction modifier_tableau.
- On remarque que seulement le chemin stocké dans tableau est copié dans param1, mais le tableau, lui n'est pas copié.

```
func modifier_tableau(var param1):
    param1[0] = 5

func _ready():
    var tableau = [0, 3, 5, 2]
    modifier_tableau(tableau)
    print(tableau)
```

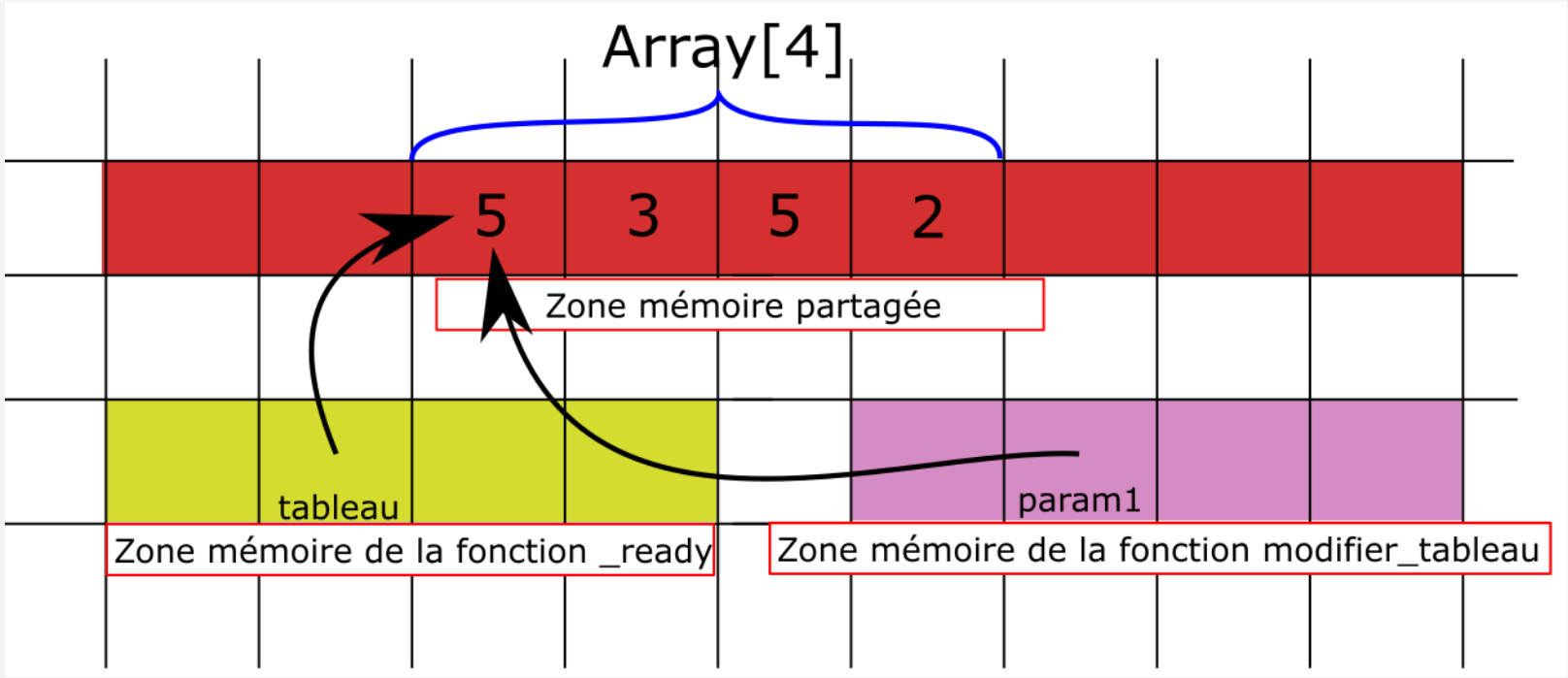


# PASSAGE PAR REFERENCE

- Lorsque la fonction modifier_tableau va modifier la première case du Array[4] dans la mémoire partagée (param[0] = 5), alors la variable tableau ayant le même chemin vers le Array[4] va aussi voir cette première case modifiée!
- C'est ce qu'on appelle alors le passage par référence

```
func modifier_tableau(var param1):
    param1[0] = 5

func _ready():
    var tableau = [0, 3, 5, 2]
    modifier_tableau(tableau)
    print(tableau)
```



# PASSAGE PAR REFERENCE

- Les types qui ne sont pas built-in sont très nombreux. On a vu que les Array en sont un, mais tous les objets le sont aussi comme par exemple Node2D, Line2D, …
- La raison de ne pas copier un type built-in lors son envoie à une fonction built-in est multiple:
  - Si l'objet est très grand en mémoire, la copie peut prendre beaucoup de temps. Imaginons un tableau qui ne fait non pas une taille de 4 éléments, mais de 10000. Il faut alors copier les 10000 éléments à chaque fois qu'on envoie le tableau, et ce serait désastreux niveau temps d'exécution.
  - Lorsqu'une variable non built-in est créée dans une fonction, çà veut dire, que seulement la fonction peut modifier cette variable, ce qui est pas top lorsqu'on veut modifier notre variable dans une autre fonction, comme par exemple la position d'un Node2D.

# QUELQUES EXERCICES

- Création d'une fonction qui additionne deux variables en entrée et qui retourne le résultat. Tester dans la fonction *_ready()* avec des entiers, des vecteurs 2D, des chaines de caractères
- Créer une fonction qui prend en entrée un tableau d'entiers, et qui va calculer la somme de toutes les valeurs contenues dans le tableau.
- Créer une fonction qui indique si le nombre est pair ou impair. Pour savoir si un nombre est pair, le résultat du nombre modulo 2 doit être égal à 0.
  Dans Godot :

```
var result  = 7%2 # result = 1 => impair (7 / 2 = 3*2+1)
var result2 = 4%2 # result = 0 => pair   (4 / 2 = 2*2+0)
```

- Créer une fonction qui prend comme paramètre en entrée : un nombre entier et un nombre de boucles. Cette fonction doit calculer la multiplication du nombre par lui-même **n** fois (**n** le nombre de boucles).
- Créer une fonction qui prend comme paramètre deux nombres : x et y . Et qui retourne un Vecteur2D à partir de ces deux nombres x et y.

# QUELQUES EXERCICES

- Création d'une fonction qui additionne deux variables en entrée et qui retourne le résultat. Tester dans la fonction *_ready()* avec des entiers, des vecteurs 2D, des chaines de caractères
- Mettre Godot dans la scène. Faire en sorte que lorsqu'on appuie sur la touche espace il apparaisse sur une position aléatoire dans l'écran (utilisation de la fonction randi()).
- Avec ce même Godot dans la scène. Faire en sorte que lorsqu'on appuie sur la touche de clavier 't', le godot disparaisse et que cette même touche le fasse réapparaître.
- Ajouter un Godot rouge (changer la couleur du godot dans les propriétés/Modulation). Le faire bouger automatiquement dans la scène de gauche à droite et de droite à gauche au milieu de l'écran.
- Ajouter des zones de collisions aux deux Godot et afficher un message dans le terminal lorsqu'ils se croisent.