R2.01 - Développement Orienté Objets

Tout est références

Arnaud Lanoix Brauer
Arnaud.Lanoix@univ-nantes.fr

IUT Nantes
Pôle Sciences et technologie

Nantes Université

Département informatique

Les variables sont des références

- En Kotlin, toutes les variables sont des références (dans la pile mémoire) qui "pointent" vers leur valeur (dans le tas mémoire)
- Une référence en Kotlin correspond à un pointeur en C/C++, avec une gestion simplifiée de l'allocation mémoire :
  - On ne s'occupe pas de réserver de l'espace mémoire
  - On ne gère pas non plus la libération de cet espace : le *Garbage Collector* (=ramasse-miette) s'occupe de libérer l'espace occupé par des objets déréférencés

IUT Nantes

Les variables sont des références

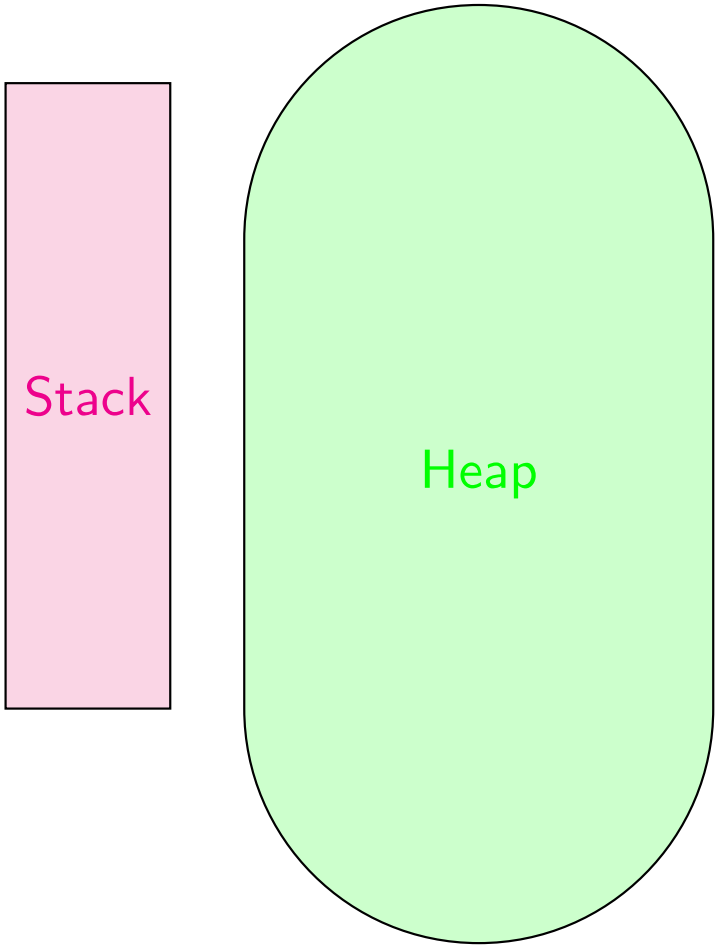
- En Kotlin, toutes les variables sont des références (dans la pile mémoire) qui "pointent" vers leur valeur (dans le tas mémoire)
- Une référence en Kotlin correspond à un pointeur en C/C++, avec une gestion simplifiée de l'allocation mémoire :
  - On ne s'occupe pas de réserver de l'espace mémoire
  - On ne gère pas non plus la libération de cet espace : le *Garbage Collector* (=ramasse-miette) s'occupe de libérer l'espace occupé par des objets déréférencés

L'opérateur d'identité `===`

L'opérateur `===` (3 x `=`) permet de vérifier que deux variables référencent le MÊME objet (en mémoire)

- L'opérateur d'égalité `==` (2 x `=`) regarde l'égalité (des "valeurs")
- `===` implique `==` mais la réciproque n'est pas vraie

Les variables sont des références : exemple de `String`

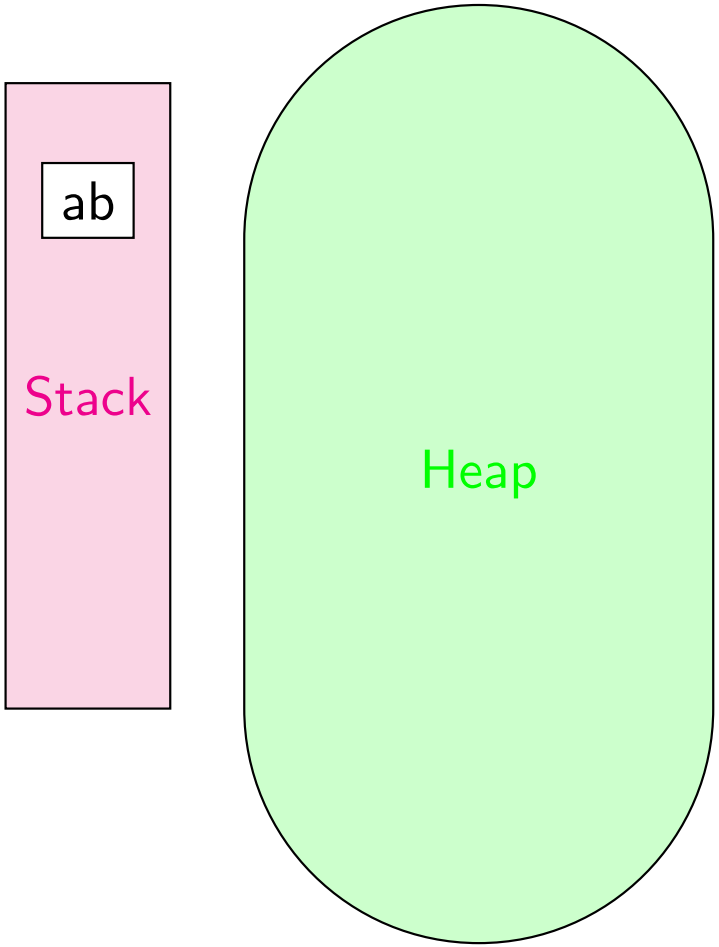



```
var ab : String
var yz : String
ab = "abcd"
yz = "ABCD"
println("val: ${ab == yz}")//false
println("ref: ${ab === yz}")//false
yz = ab
println("val: ${ab == yz}")//true
println("ref: ${ab === yz}")//true
var ij = "ABCD"
yz = ij.lowercase()
println("val: ${ab == yz}") //true
println("ref: ${ab === yz}")//false
```

Schématisation de la mémoire de la JVM

IUT Nantes

Les variables sont des références : exemple de `String`

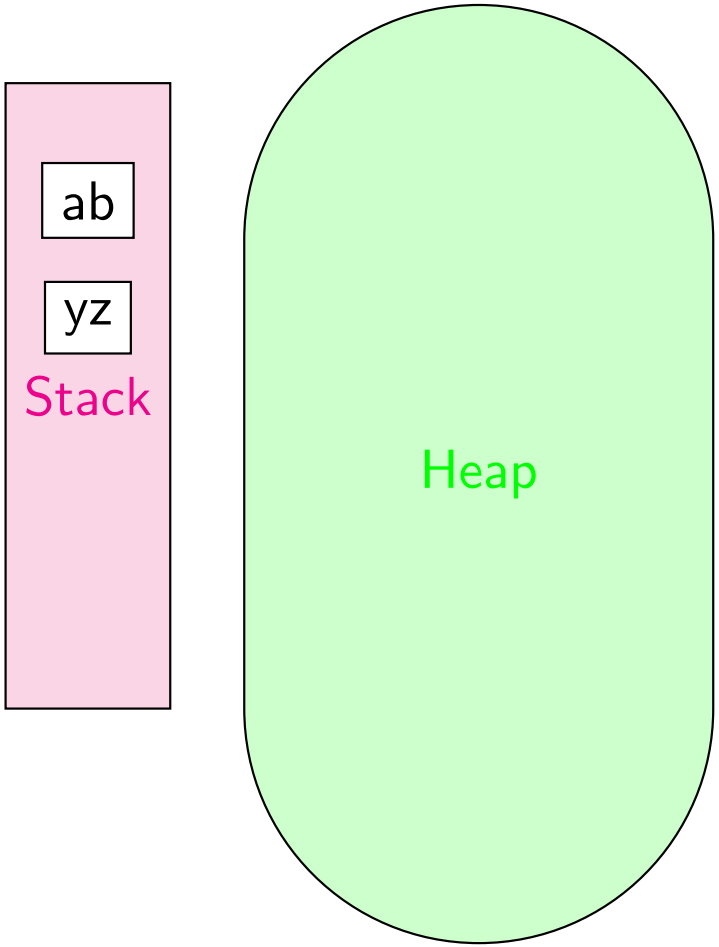



```
var ab : String
var yz : String
ab = "abcd"
yz = "ABCD"
println("val: ${ab == yz}")//false
println("ref: ${ab === yz}")//false
yz = ab
println("val: ${ab == yz}")//true
println("ref: ${ab === yz}")//true
var ij = "ABCD"
yz = ij.lowercase()
println("val: ${ab == yz}") //true
println("ref: ${ab === yz}")//false
```

`ab` créé dans la pile mémoire

IUT Nantes

Les variables sont des références : exemple de `String`

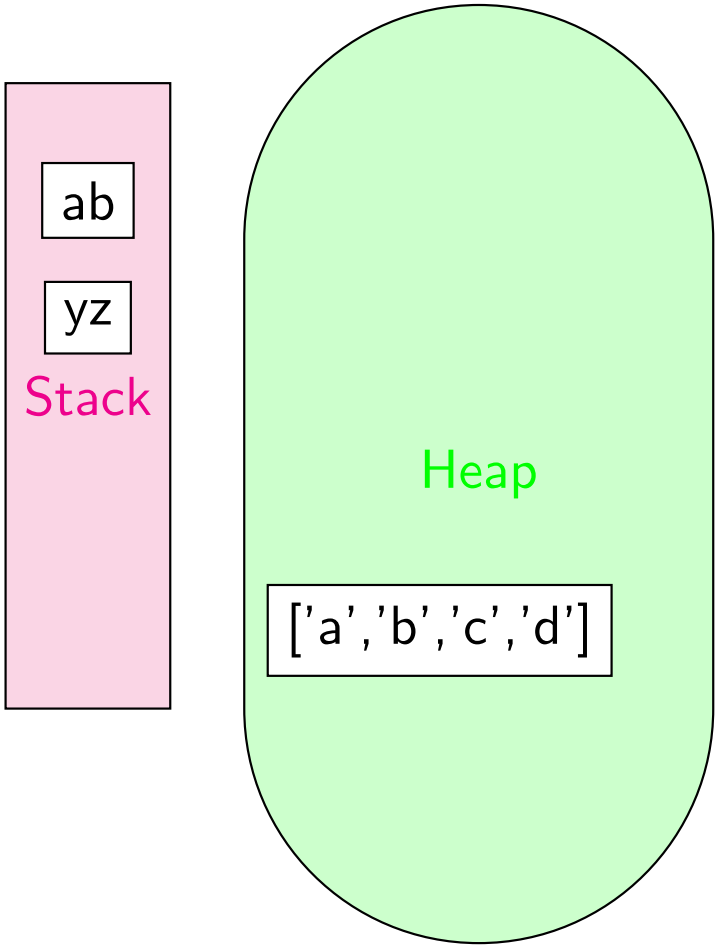



```
var ab : String
var yz : String
ab = "abcd"
yz = "ABCD"
println("val: ${ab == yz}")//false
println("ref: ${ab === yz}")//false
yz = ab
println("val: ${ab == yz}")//true
println("ref: ${ab === yz}")//true
var ij = "ABCD"
yz = ij.lowercase()
println("val: ${ab == yz}") //true
println("ref: ${ab === yz}")//false
```

`yz` créé dans la pile mémoire

IUT Nantes

Les variables sont des références : exemple de `String`

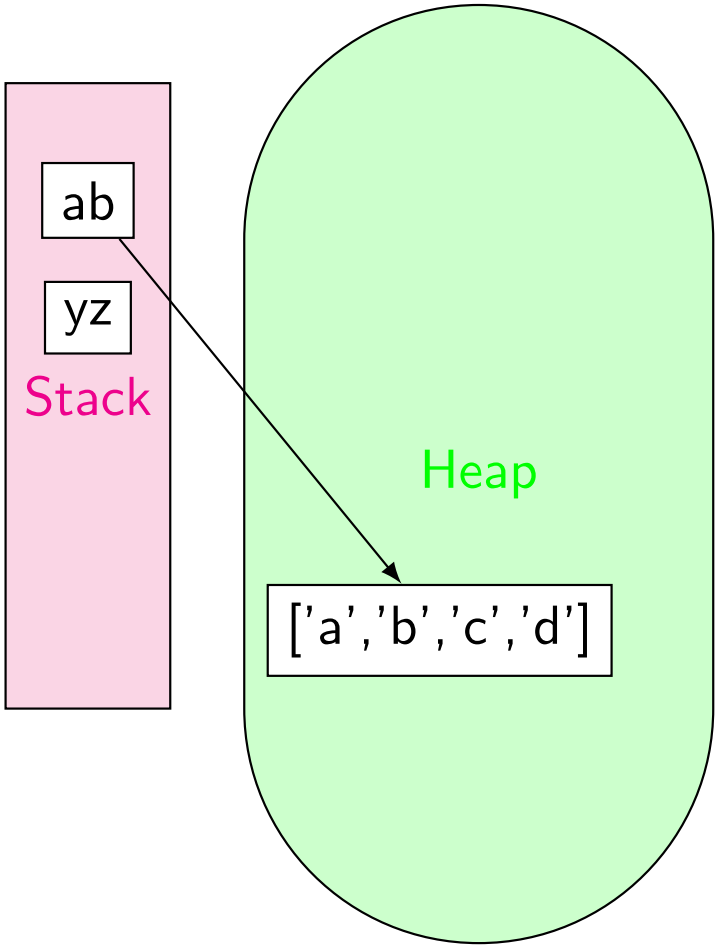



```
var ab : String
var yz : String
ab = "abcd"
yz = "ABCD"
println("val: ${ab == yz}")//false
println("ref: ${ab === yz}")//false
yz = ab
println("val: ${ab == yz}")//true
println("ref: ${ab === yz}")//true
var ij = "ABCD"
yz = ij.lowercase()
println("val: ${ab == yz}") //true
println("ref: ${ab === yz}")//false
```

`"abcd"` créé dans le tas mémoire

IUT Nantes

Les variables sont des références : exemple de `String`

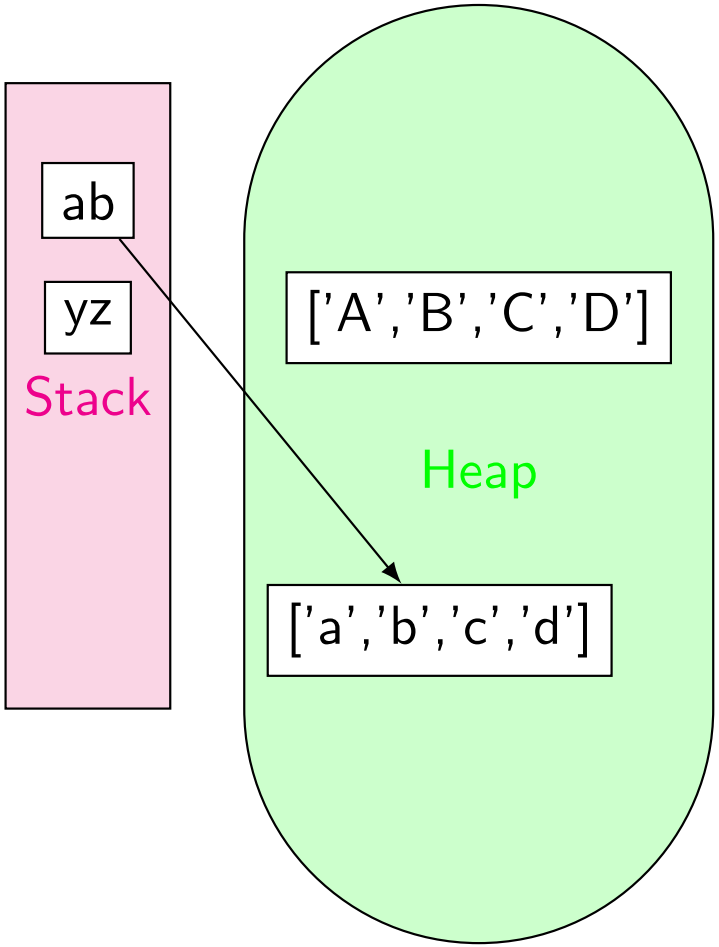



```
var ab : String
var yz : String
ab = "abcd"
yz = "ABCD"
println("val: ${ab == yz}")//false
println("ref: ${ab === yz}")//false
yz = ab
println("val: ${ab == yz}")//true
println("ref: ${ab === yz}")//true
var ij = "ABCD"
yz = ij.lowercase()
println("val: ${ab == yz}") //true
println("ref: ${ab === yz}")//false
```

`ab` "pointe" vers `"abcd"`

IUT Nantes

Les variables sont des références : exemple de `String`

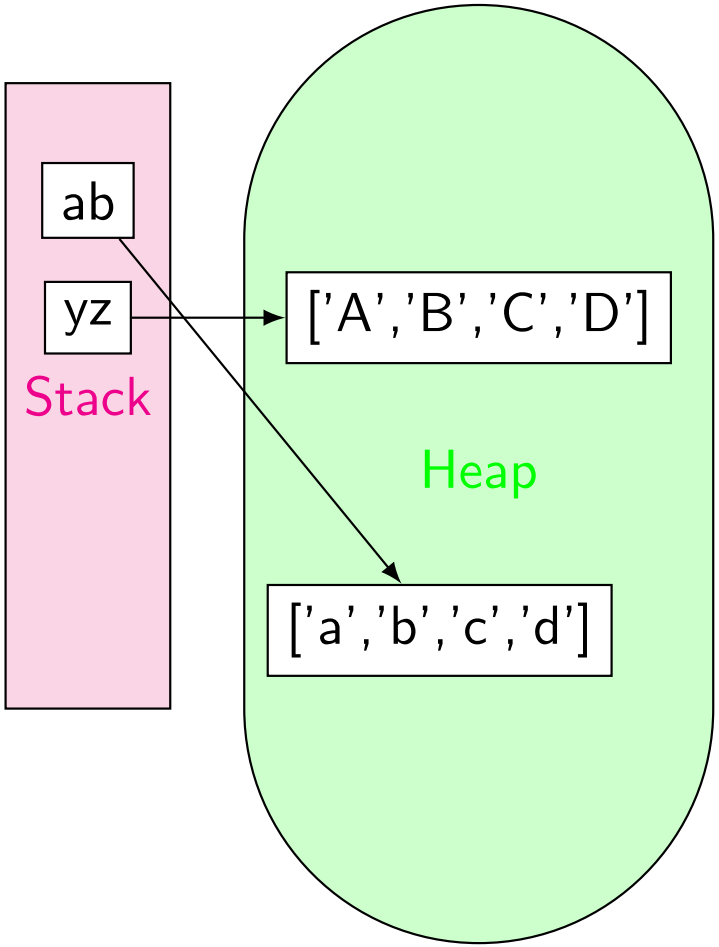



```
var ab : String
var yz : String
ab = "abcd"
yz = "ABCD"
println("val: ${ab == yz}")//false
println("ref: ${ab === yz}")//false
yz = ab
println("val: ${ab == yz}")//true
println("ref: ${ab === yz}")//true
var ij = "ABCD"
yz = ij.lowercase()
println("val: ${ab == yz}") //true
println("ref: ${ab === yz}")//false
```

`"ABCD"` créé dans le tas mémoire

IUT Nantes

Les variables sont des références : exemple de `String`

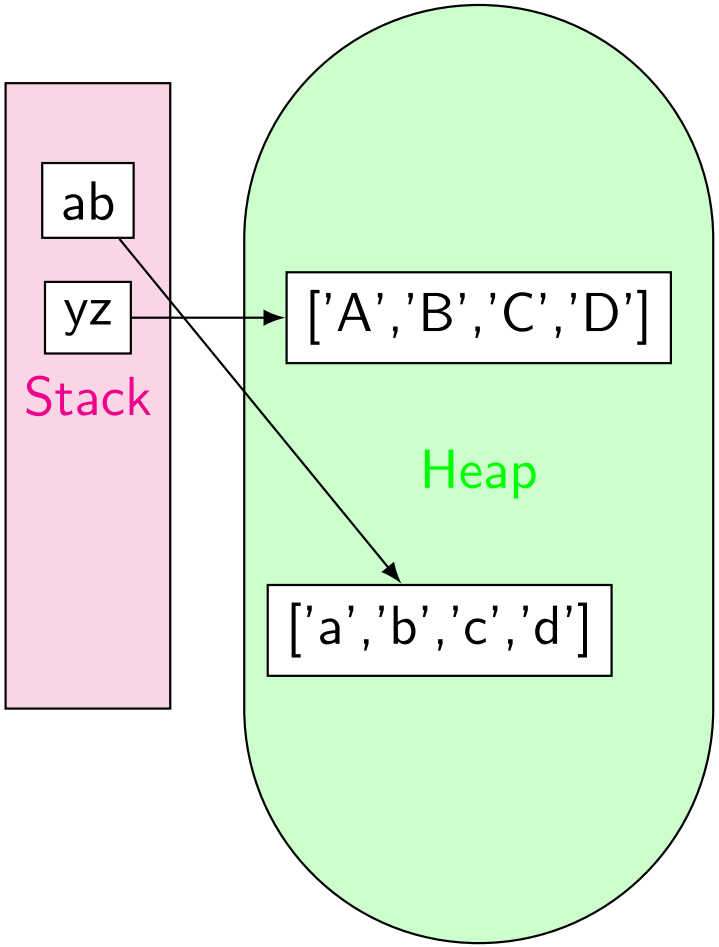



```
var ab : String
var yz : String
ab = "abcd"
yz = "ABCD"
println("val: ${ab == yz}")//false
println("ref: ${ab === yz}")//false
yz = ab
println("val: ${ab == yz}")//true
println("ref: ${ab === yz}")//true
var ij = "ABCD"
yz = ij.lowercase()
println("val: ${ab == yz}") //true
println("ref: ${ab === yz}")//false
```

`yz` "pointe" vers `"ABCD"`

IUT Nantes

Les variables sont des références : exemple de `String`

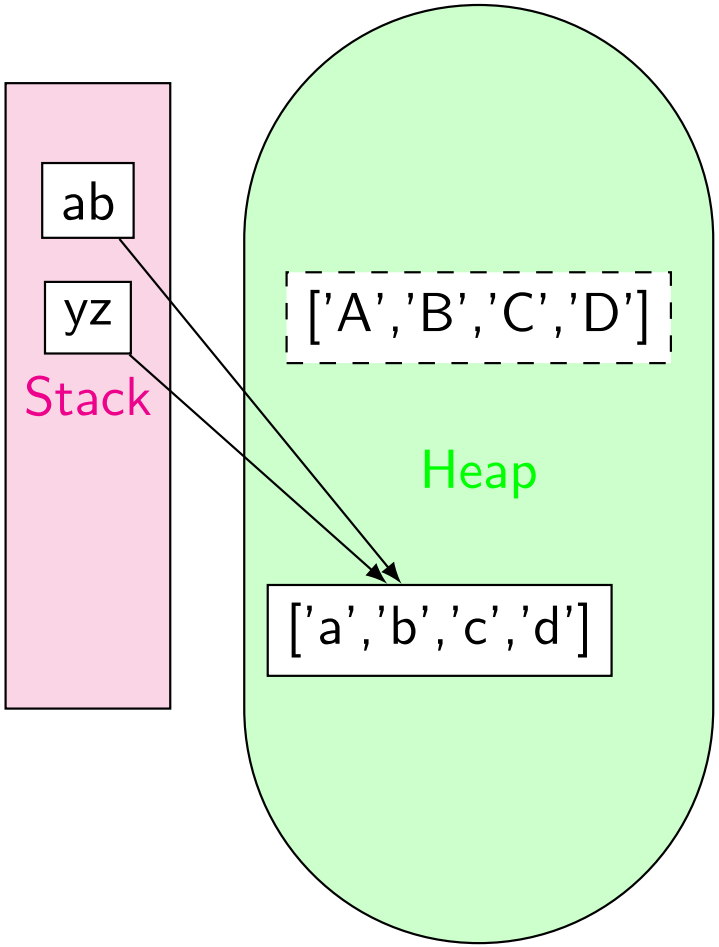



```
var ab : String
var yz : String
ab = "abcd"
yz = "ABCD"
println("val: ${ab == yz}")//false
println("ref: ${ab === yz}")//false
yz = ab
println("val: ${ab == yz}")//true
println("ref: ${ab === yz}")//true
var ij = "ABCD"
yz = ij.lowercase()
println("val: ${ab == yz}") //true
println("ref: ${ab === yz}")//false
```

Les valeurs de `ab` et de `yz` sont $\neq$,
leurs références aussi

IUT Nantes

Les variables sont des références : exemple de `String`

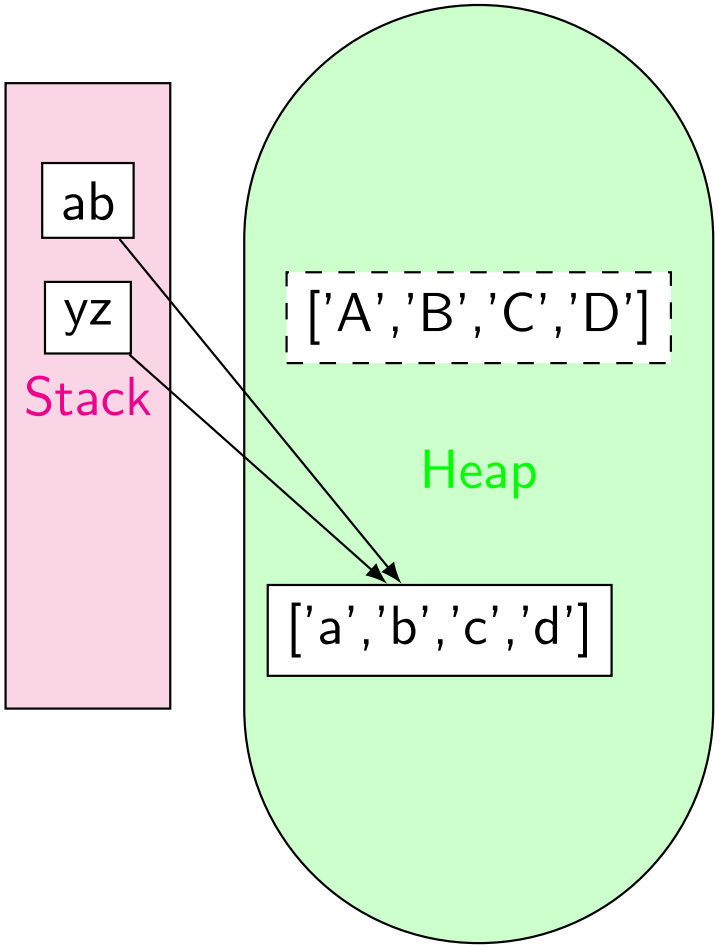



```
var ab : String
var yz : String
ab = "abcd"
yz = "ABCD"
println("val: ${ab == yz}")//false
println("ref: ${ab === yz}")//false
yz = ab
println("val: ${ab == yz}")//true
println("ref: ${ab === yz}")//true
var ij = "ABCD"
yz = ij.lowercase()
println("val: ${ab == yz}") //true
println("ref: ${ab === yz}")//false
```

yz et ab "pointent" vers la même valeur

IUT Nantes

Les variables sont des références : exemple de `String`

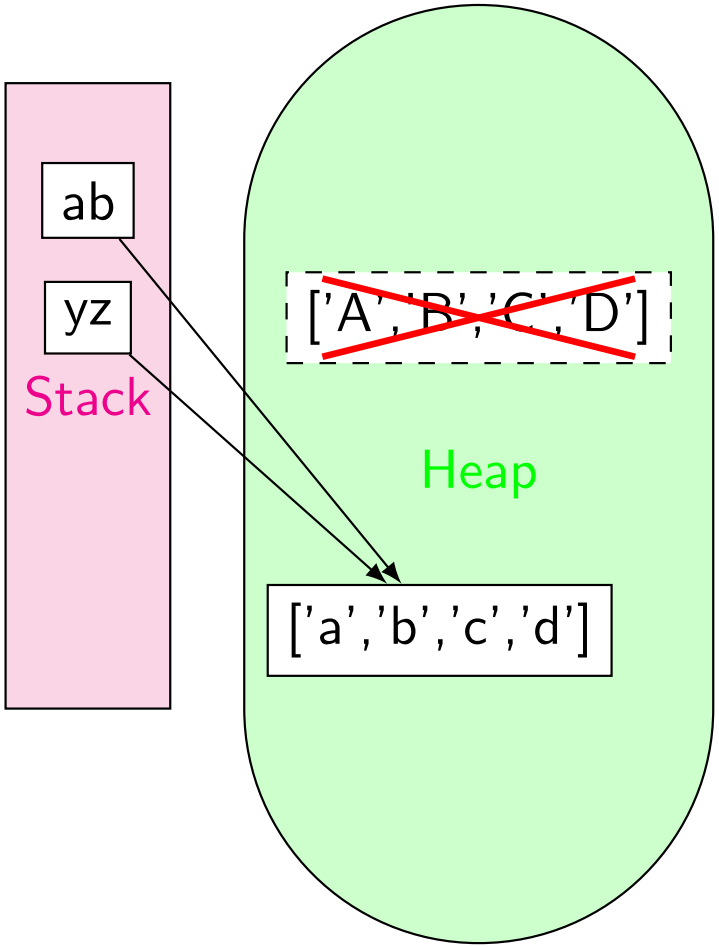



```
var ab : String
var yz : String
ab = "abcd"
yz = "ABCD"
println("val: ${ab == yz}")//false
println("ref: ${ab === yz}")//false
yz = ab
println("val: ${ab == yz}")//true
println("ref: ${ab === yz}")//true
var ij = "ABCD"
yz = ij.lowercase()
println("val: ${ab == yz}") //true
println("ref: ${ab === yz}")//false
```

Les valeurs de `ab` et de `yz` sont $=$
puisque leurs références sont $=$

IUT Nantes

Les variables sont des références : exemple de `String`

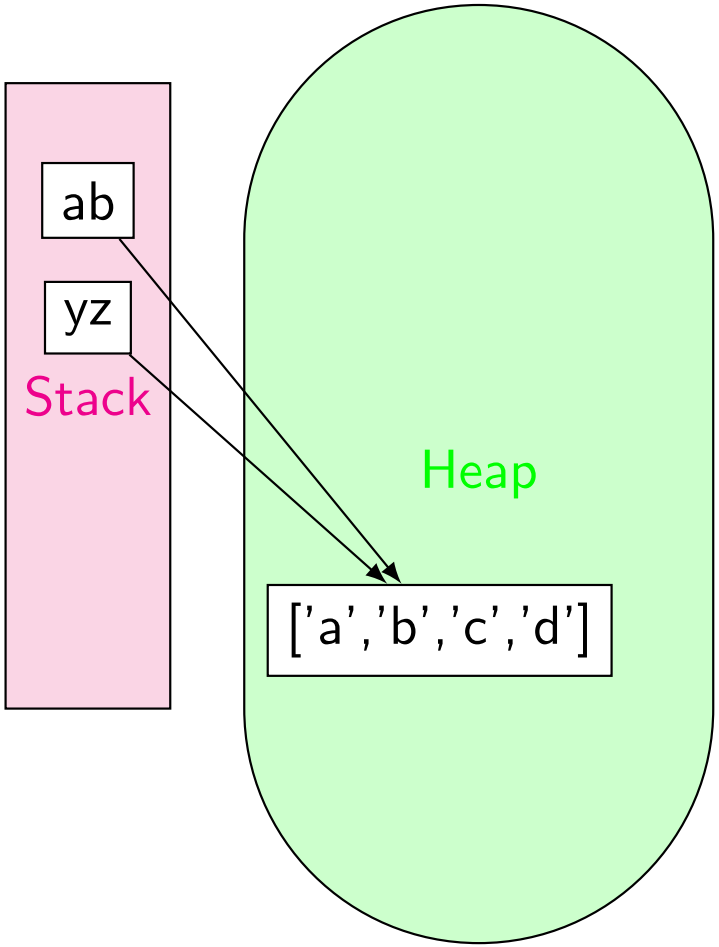



```
var ab : String
var yz : String
ab = "abcd"
yz = "ABCD"
println("val: ${ab == yz}")//false
println("ref: ${ab === yz}")//false
yz = ab
println("val: ${ab == yz}")//true
println("ref: ${ab === yz}")//true
var ij = "ABCD"
yz = ij.lowercase()
println("val: ${ab == yz}") //true
println("ref: ${ab === yz}")//false
```

Le garbage collector efface les objets déréférencés

IUT Nantes

Les variables sont des références : exemple de `String`

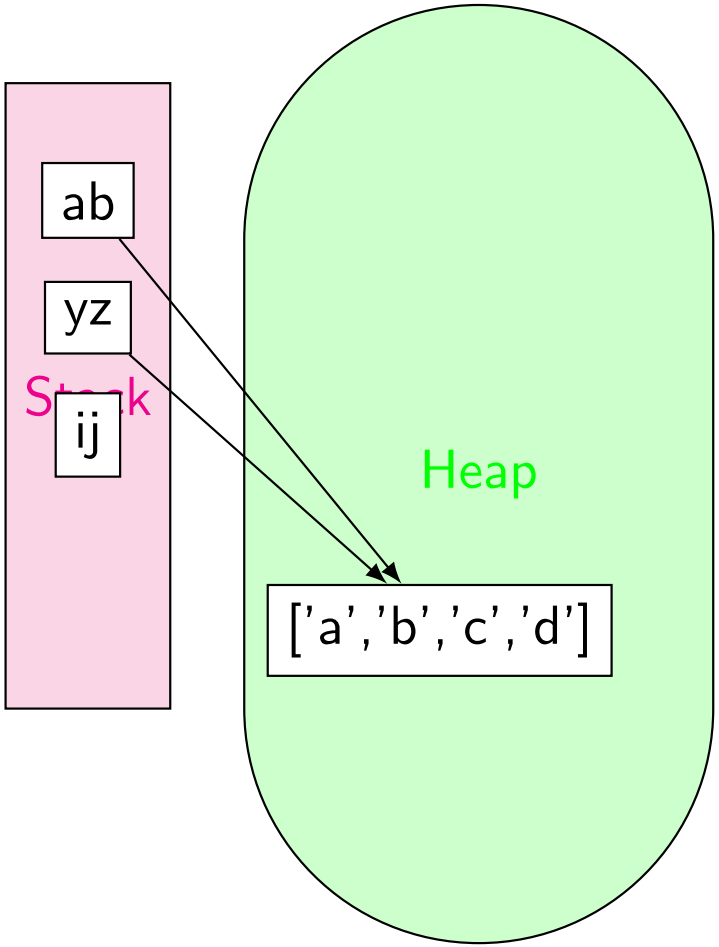



```
var ab : String
var yz : String
ab = "abcd"
yz = "ABCD"
println("val: ${ab == yz}")//false
println("ref: ${ab === yz}")//false
yz = ab
println("val: ${ab == yz}")//true
println("ref: ${ab === yz}")//true
var ij = "ABCD"
yz = ij.lowercase()
println("val: ${ab == yz}") //true
println("ref: ${ab === yz}")//false
```

Le garbage collector efface les objets déréférencés

IUT Nantes

Les variables sont des références : exemple de `String`

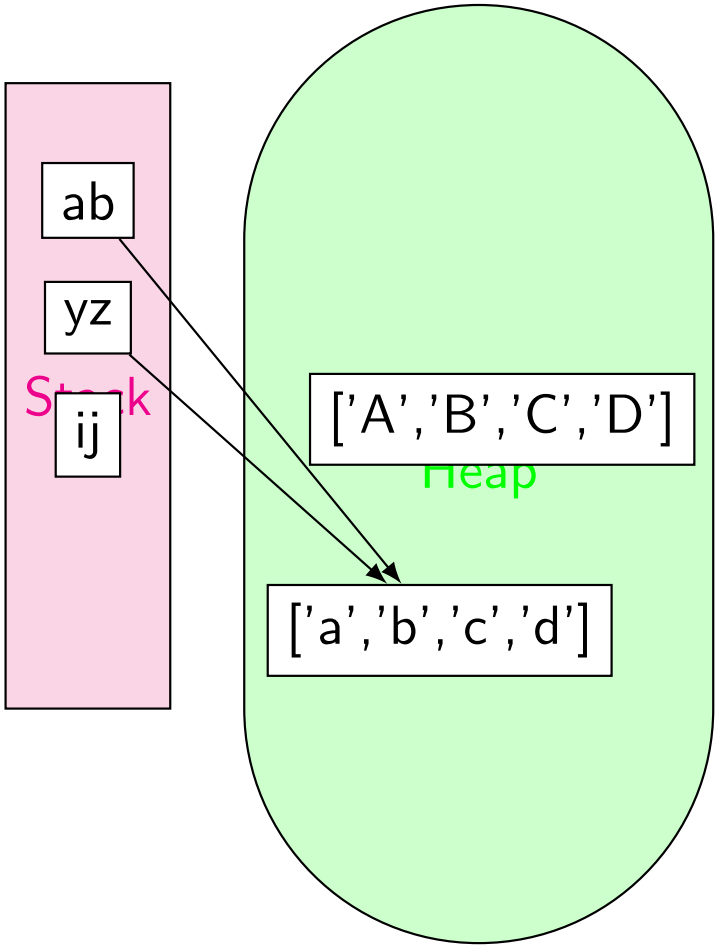



```
var ab : String
var yz : String
ab = "abcd"
yz = "ABCD"
println("val: ${ab == yz}")//false
println("ref: ${ab === yz}")//false
yz = ab
println("val: ${ab == yz}")//true
println("ref: ${ab === yz}")//true
var ij = "ABCD"
yz = ij.lowercase()
println("val: ${ab == yz}") //true
println("ref: ${ab === yz}")//false
```

`ij` créé dans la pile mémoire

IUT Nantes

Les variables sont des références : exemple de `String`

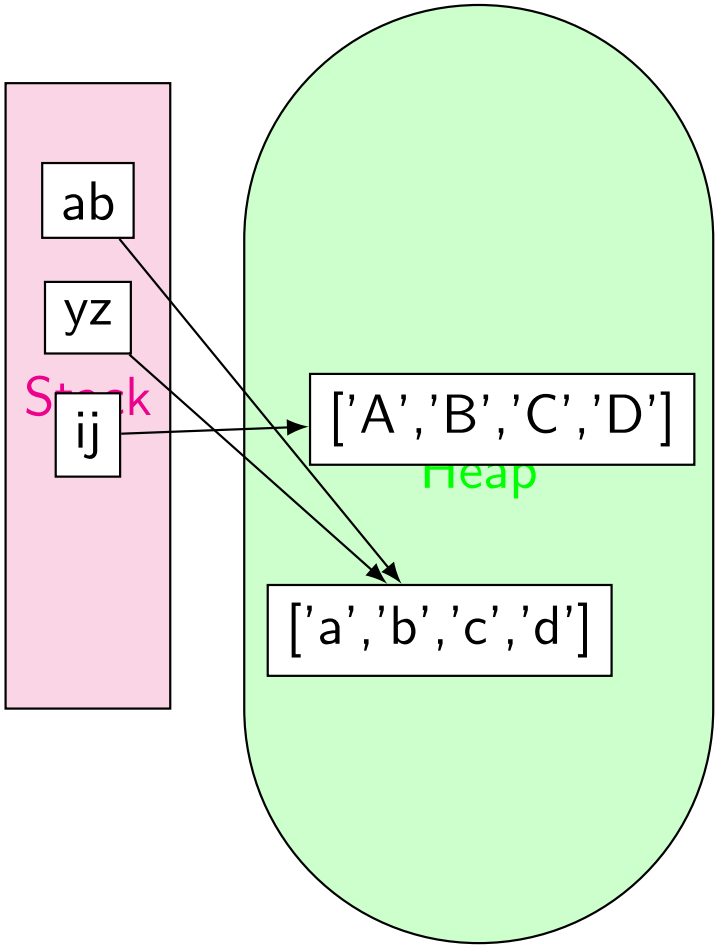



```
var ab : String
var yz : String
ab = "abcd"
yz = "ABCD"
println("val: ${ab == yz}")//false
println("ref: ${ab === yz}")//false
yz = ab
println("val: ${ab == yz}")//true
println("ref: ${ab === yz}")//true
var ij = "ABCD"
yz = ij.lowercase()
println("val: ${ab == yz}") //true
println("ref: ${ab === yz}")//false
```

`"ABCD"` créé dans le tas mémoire

IUT Nantes

Les variables sont des références : exemple de `String`

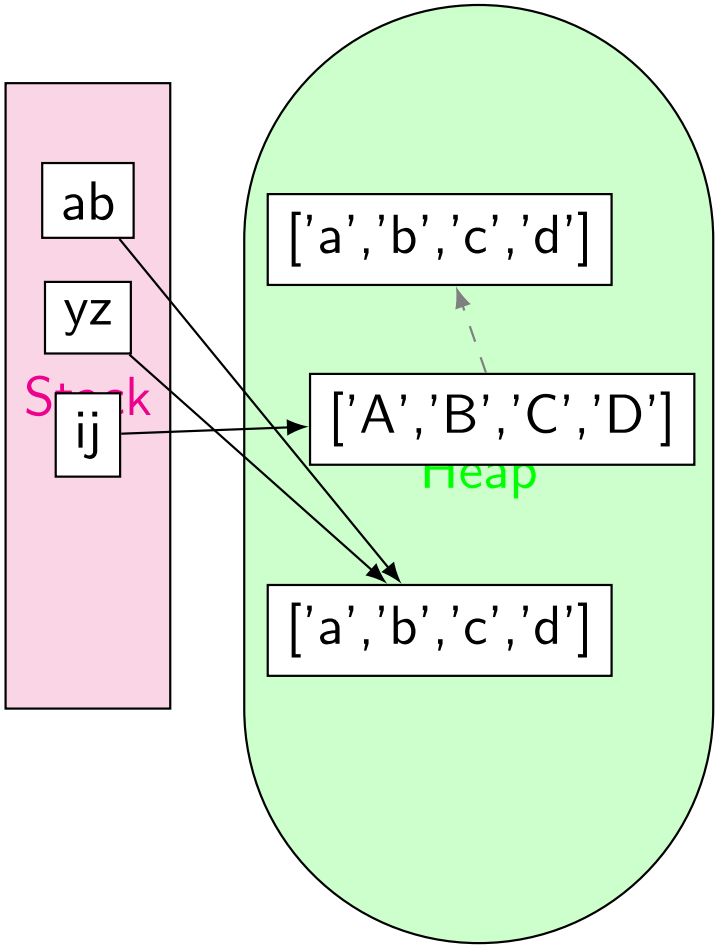



```
var ab : String
var yz : String
ab = "abcd"
yz = "ABCD"
println("val: ${ab == yz}")//false
println("ref: ${ab === yz}")//false
yz = ab
println("val: ${ab == yz}")//true
println("ref: ${ab === yz}")//true
var ij = "ABCD"
yz = ij.lowercase()
println("val: ${ab == yz}") //true
println("ref: ${ab === yz}")//false
```

`ij` "pointe" vers `"ABCD"`

IUT Nantes

Les variables sont des références : exemple de `String`

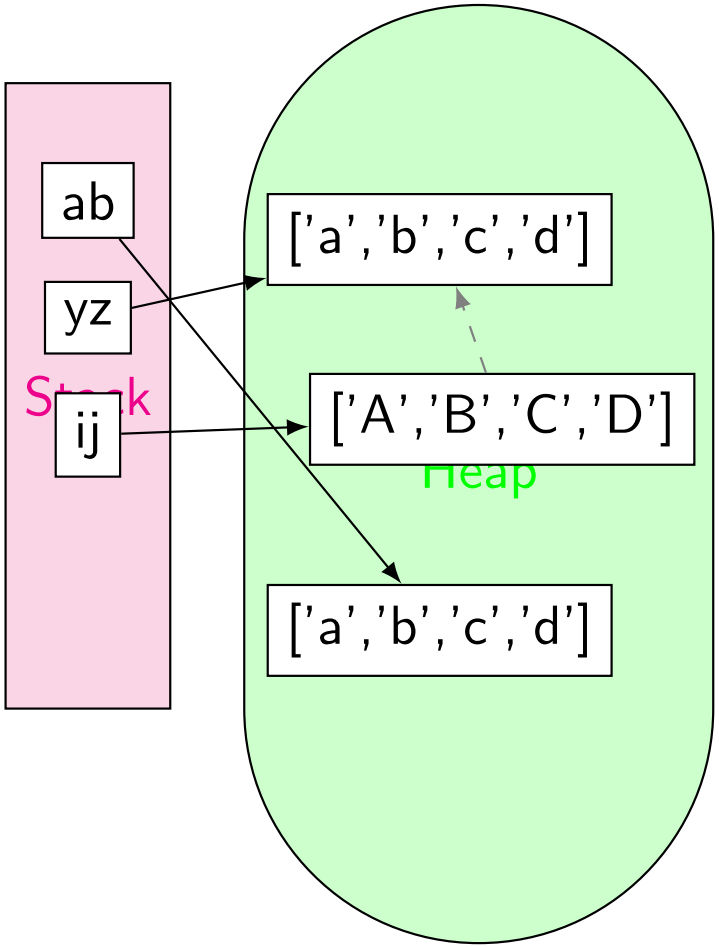



```
var ab : String
var yz : String
ab = "abcd"
yz = "ABCD"
println("val: ${ab == yz}")//false
println("ref: ${ab === yz}")//false
yz = ab
println("val: ${ab == yz}")//true
println("ref: ${ab === yz}")//true
var ij = "ABCD"
yz = ij.lowercase()
println("val: ${ab == yz}") //true
println("ref: ${ab === yz}")//false
```

`ij.lowercase()` créé `"abcd"` dans le tas mémoire

IUT Nantes

Les variables sont des références : exemple de `String`

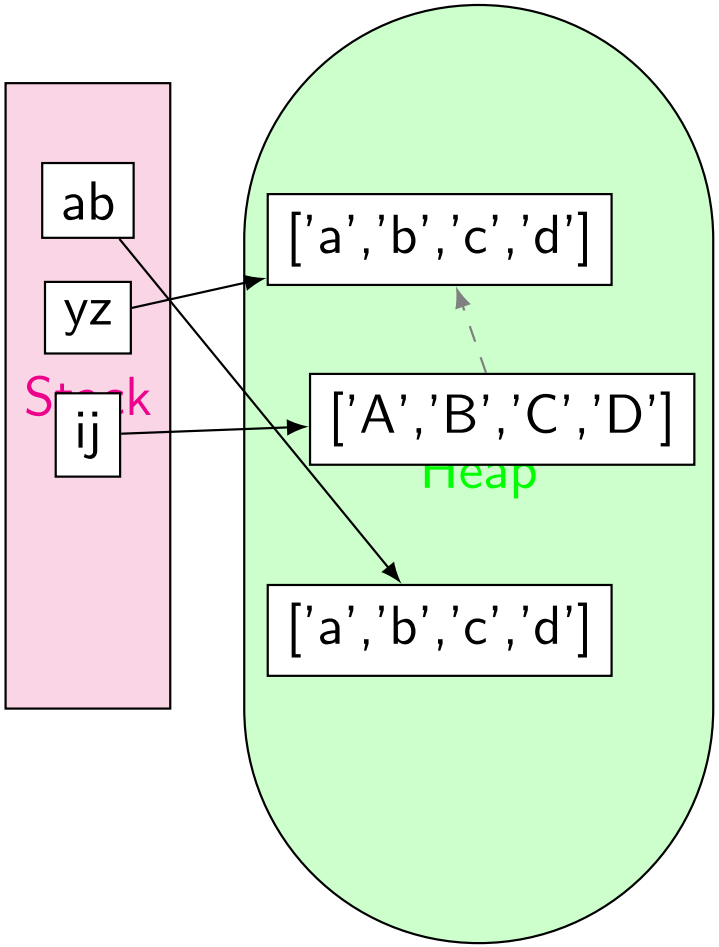



```
var ab : String
var yz : String
ab = "abcd"
yz = "ABCD"
println("val: ${ab == yz}")//false
println("ref: ${ab === yz}")//false
yz = ab
println("val: ${ab == yz}")//true
println("ref: ${ab === yz}")//true
var ij = "ABCD"
yz = ij.lowercase()
println("val: ${ab == yz}") //true
println("ref: ${ab === yz}")//false
```

`yz` "pointe" vers `"abcd"`

IUT Nantes

Les variables sont des références : exemple de `String`




```
var ab : String
var yz : String
ab = "abcd"
yz = "ABCD"
println("val: ${ab == yz}")//false
println("ref: ${ab === yz}")//false
yz = ab
println("val: ${ab == yz}")//true
println("ref: ${ab === yz}")//true
var ij = "ABCD"
yz = ij.lowercase()
println("val: ${ab == yz}") //true
println("ref: ${ab === yz}")//false
```

Les valeurs de `ab` et de `yz` sont $=$,
mais leur références sont $\neq$

IUT Nantes

Variables *nullable*

Si une variable est une référence alors elle peut "pointer" vers rien ? **NON**
Sauf si on a précisé explicitement qu'elle pouvait.

- Ajouter `?` après le type indique que la variable est possiblement `null`
- Les paramètres et/ou le résultat d'une fonction peuvent aussi être possiblement `null`

```
var w : Int
val x : Int?
var y : Double? = 10.0
var z : String? = "totoro"
// w = null
// erreur de compilation
y = null
z = null
```

"*The Billion-Dollar Mistake*" (C.A.R. Hoare)

Forcer à indiquer les variables possiblement `null` permet d'éviter une grand nombre d'erreurs "classiques" en Java : NullPointerEception, qui arrive dès lors qu'on essaie d'accéder à une variable qui ne référence rien

IUT Nantes

Variables *nullable*

Si une variable est une référence alors elle peut "pointer" vers rien ? **NON**
Sauf si on a précisé explicitement qu'elle pouvait.

- Ajouter `?` après le type indique que la variable est possiblement `null`
- Les paramètres et/ou le résultat d'une fonction peuvent aussi être possiblement `null`

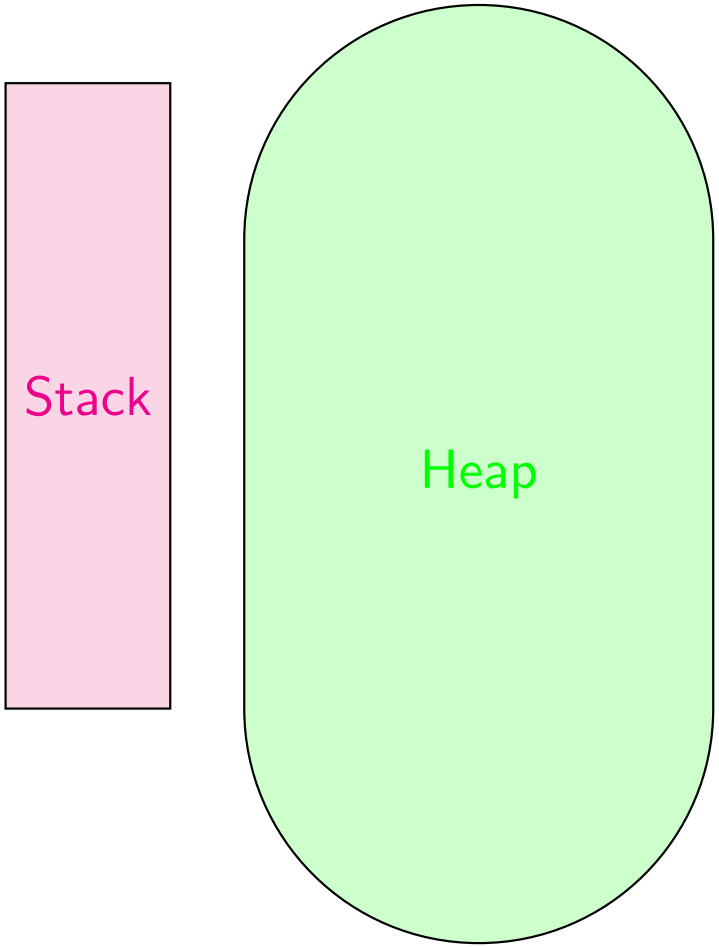
```
var w : Int
val x : Int?
var y : Double? = 10.0
var z : String? = "totoro"
// w = null
// erreur de compilation
y = null
z = null
```

"*The Billion-Dollar Mistake*" (C.A.R. Hoare)

Forcer à indiquer les variables possiblement `null` permet d'éviter une grand nombre d'erreurs "classiques" en Java : NullPointerEception, qui arrive dès lors qu'on essaie d'accéder à une variable qui ne référence rien

IUT Nantes

Les variables sont des références : exemple de `String?`

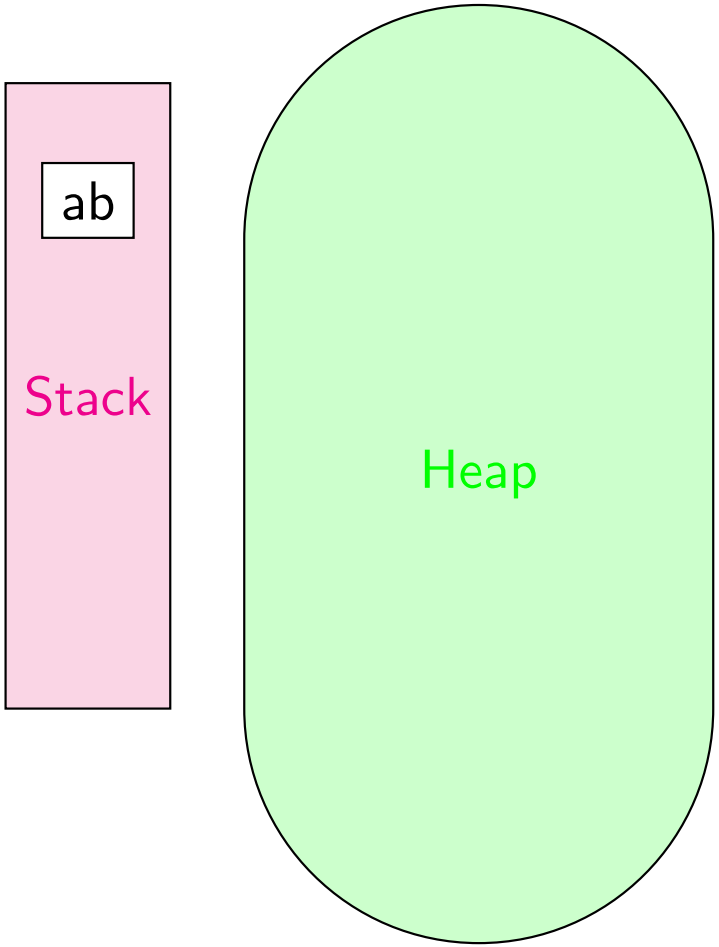



```
var ab : String? = "abc"
var yz : String? = "ABC"
println(ab) //"abc"
println(yz) //"ABC"
yz = ab
println(ab) //"abc"
println(yz) //"abc"
ab = null
println(ab) //null
println(yz) //"abc"
ab.uppercase() //erreur (impossible)
```

Schématisation de la mémoire de la JVM

IUT Nantes

Les variables sont des références : exemple de `String?`

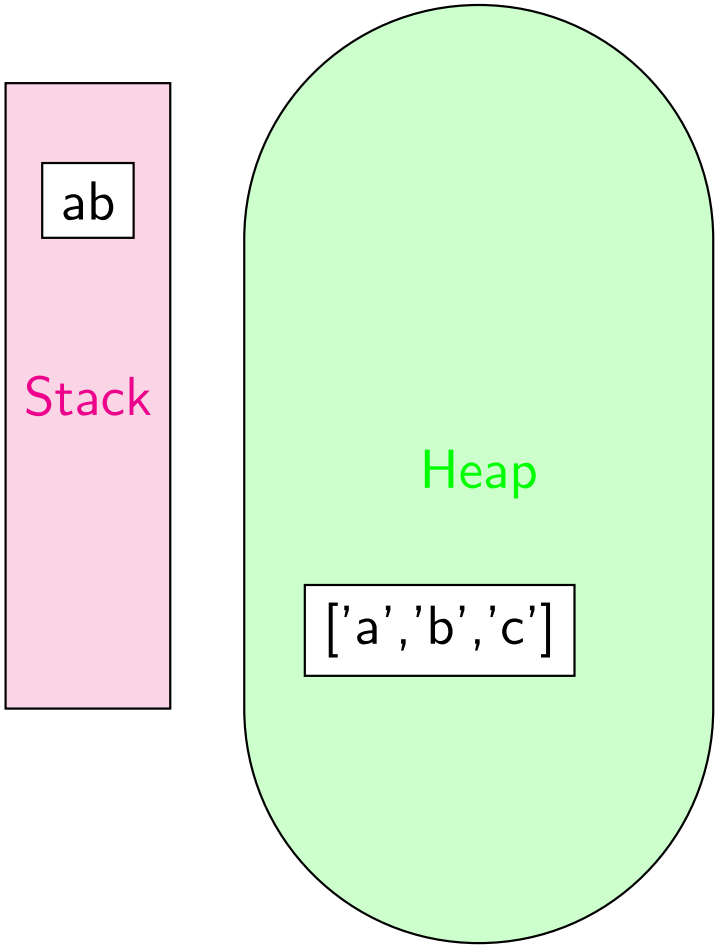



```
var ab : String? = "abc"
var yz : String? = "ABC"
println(ab) //"abc"
println(yz) //"ABC"
yz = ab
println(ab) //"abc"
println(yz) //"abc"
ab = null
println(ab) //null
println(yz) //"abc"
ab.uppercase() //erreur (impossible)
```

`ab` créé dans la pile mémoire

IUT Nantes

Les variables sont des références : exemple de `String?`

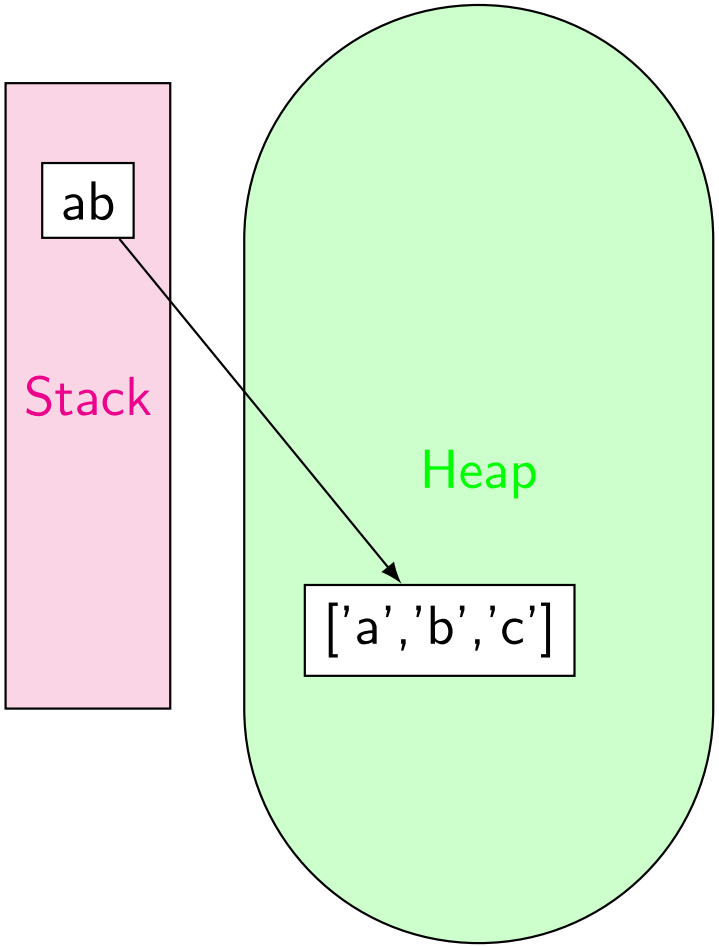



```
var ab : String? = "abc"
var yz : String? = "ABC"
println(ab) //"abc"
println(yz) //"ABC"
yz = ab
println(ab) //"abc"
println(yz) //"abc"
ab = null
println(ab) //null
println(yz) //"abc"
ab.uppercase() //erreur (impossible)
```

`"abc"` créé dans le tas mémoire

IUT Nantes

Les variables sont des références : exemple de `String?`

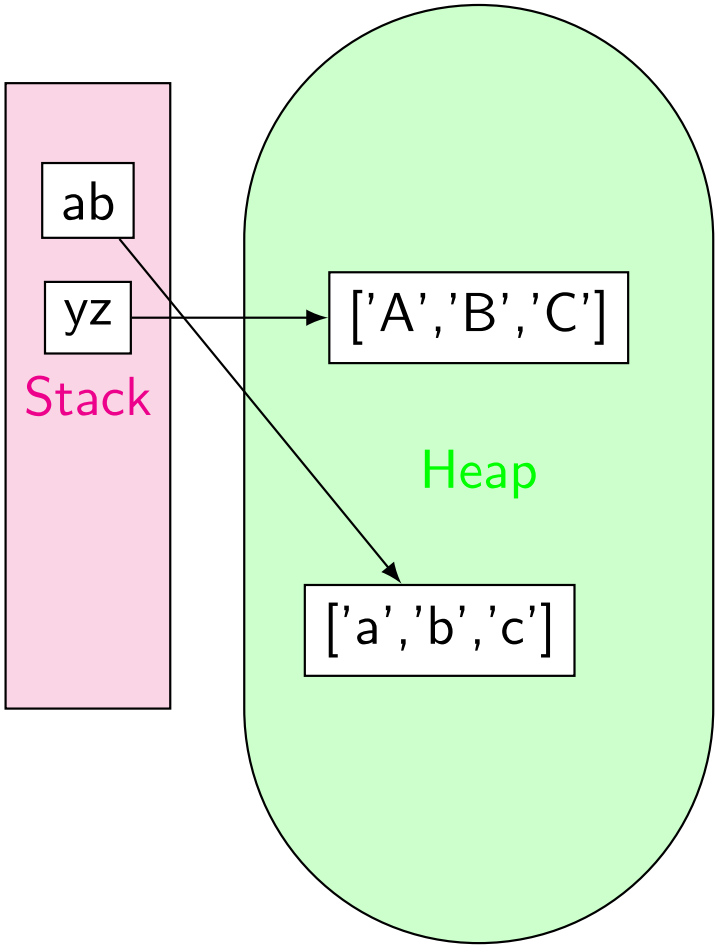



```
var ab : String? = "abc"
var yz : String? = "ABC"
println(ab) //"abc"
println(yz) //"ABC"
yz = ab
println(ab) //"abc"
println(yz) //"abc"
ab = null
println(ab) //null
println(yz) //"abc"
ab.uppercase() //erreur (impossible)
```

`ab` "pointe" vers `"abc"`

IUT Nantes

Les variables sont des références : exemple de `String?`



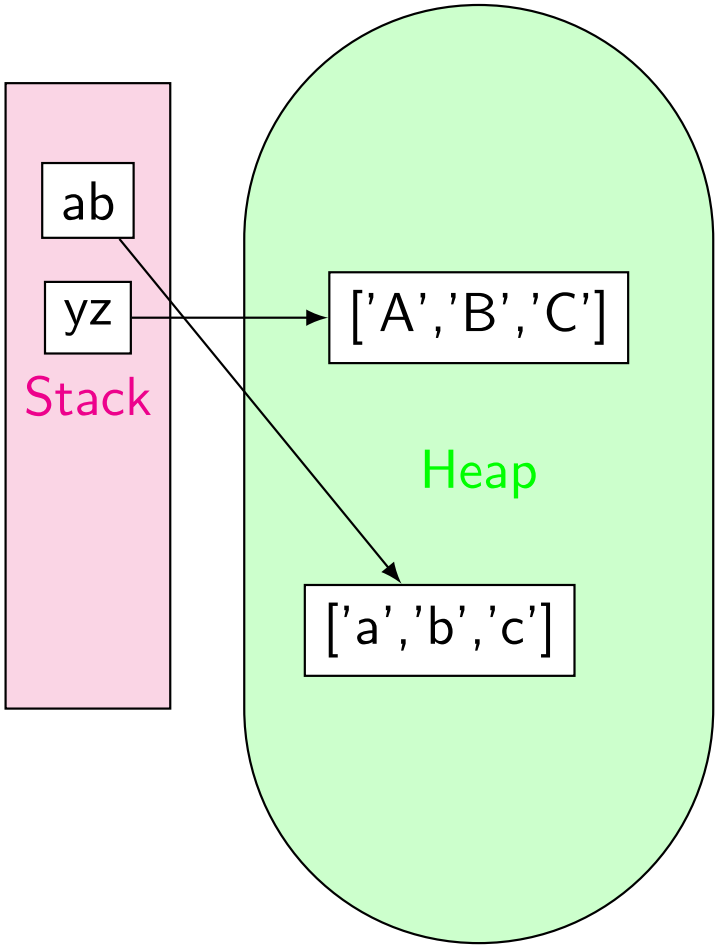

```
var ab : String? = "abc"
var yz : String? = "ABC"
println(ab) //"abc"
println(yz) //"ABC"
yz = ab
println(ab) //"abc"
println(yz) //"abc"
ab = null
println(ab) //null
println(yz) //"abc"
ab.uppercase() //erreur (impossible)
```

`yz` et `"ABC"` créés

`yz` "pointe" vers `"ABC"`

IUT Nantes

Les variables sont des références : exemple de `String?`

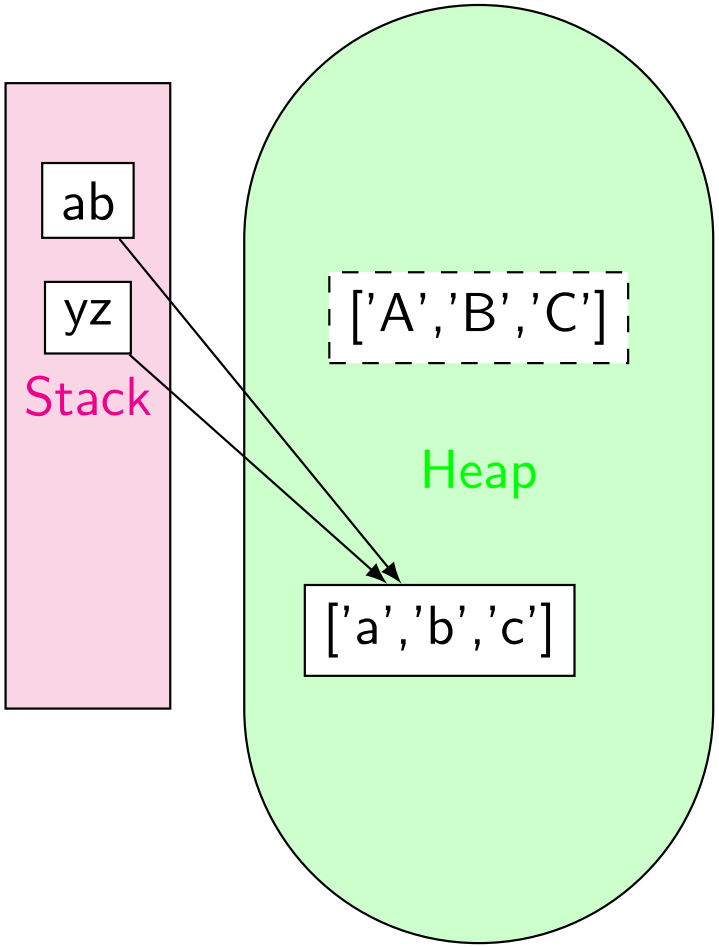



```
var ab : String? = "abc"
var yz : String? = "ABC"
println(ab) //"abc"
println(yz) //"ABC"
yz = ab
println(ab) //"abc"
println(yz) //"abc"
ab = null
println(ab) //null
println(yz) //"abc"
ab.uppercase() //erreur (impossible)
```

Les valeurs de `yz` et `ab` sont $\neq$

IUT Nantes

Les variables sont des références : exemple de `String?`

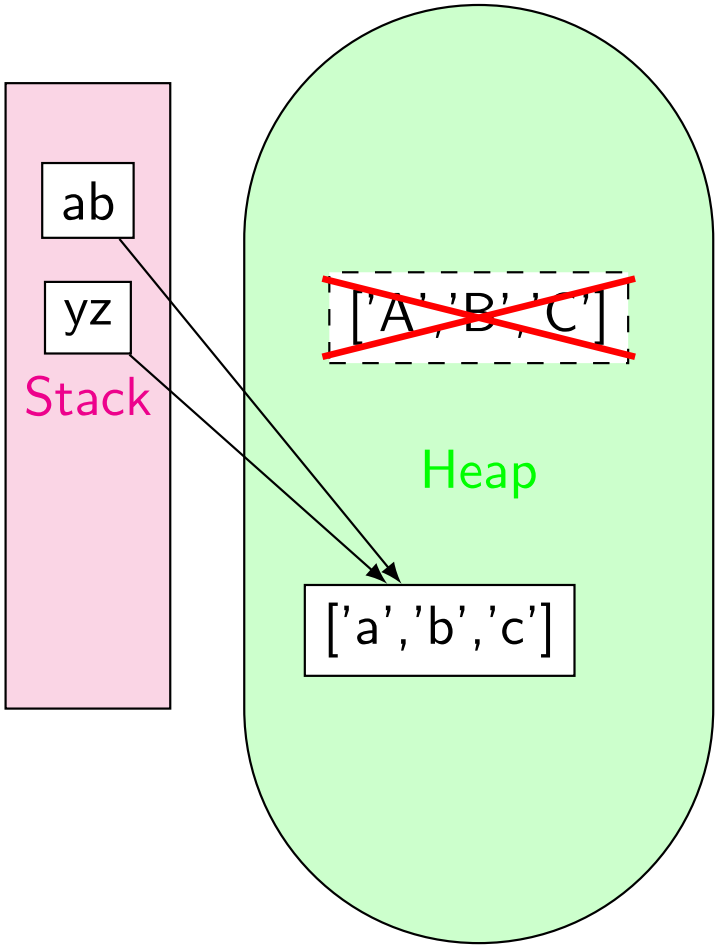



```
var ab : String? = "abc"
var yz : String? = "ABC"
println(ab) //"abc"
println(yz) //"ABC"
yz = ab
println(ab) //"abc"
println(yz) //"abc"
ab = null
println(ab) //null
println(yz) //"abc"
ab.uppercase() //erreur (impossible)
```

yz et ab "pointent" vers la même valeur

IUT Nantes

Les variables sont des références : exemple de `String?`

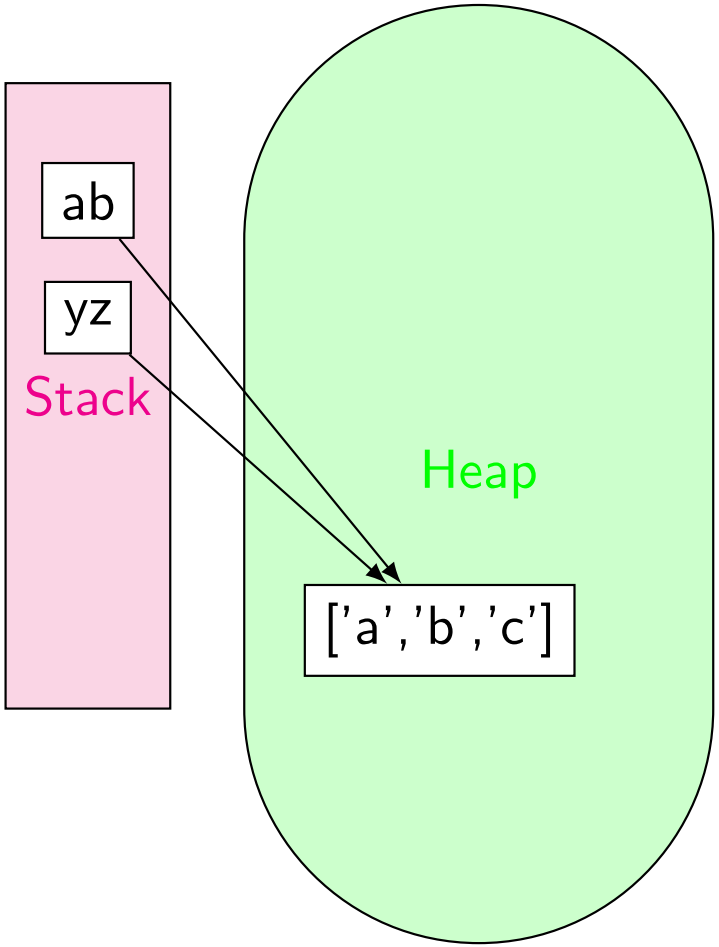



```
var ab : String? = "abc"
var yz : String? = "ABC"
println(ab) //"abc"
println(yz) //"ABC"
yz = ab
println(ab) //"abc"
println(yz) //"abc"
ab = null
println(ab) //null
println(yz) //"abc"
ab.uppercase() //erreur (impossible)
```

Le garbage collector efface les objets déréférencés

IUT Nantes

Les variables sont des références : exemple de `String?`

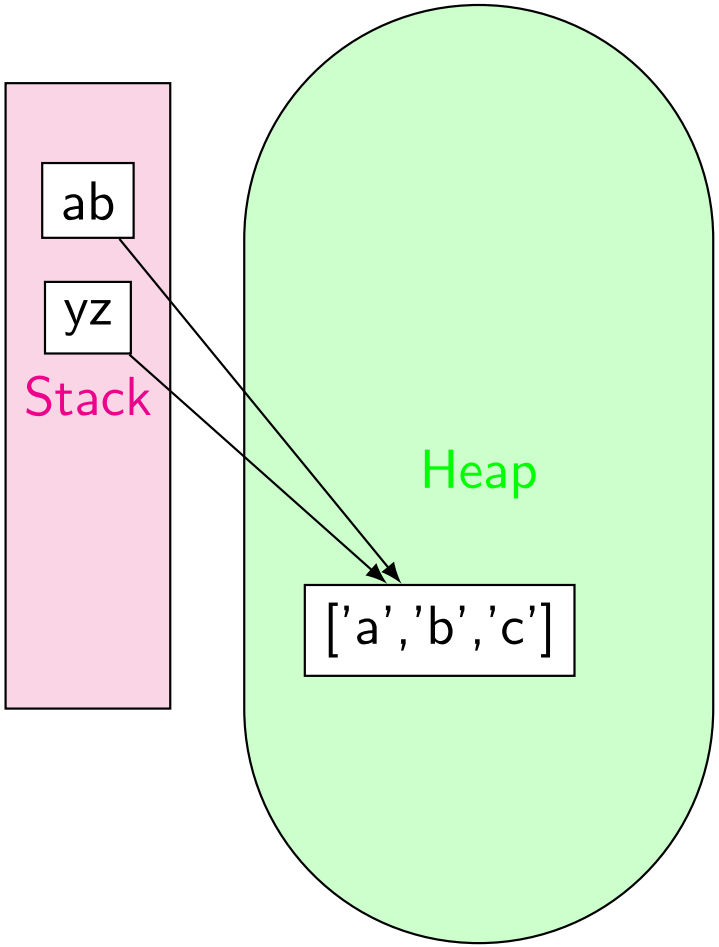



```
var ab : String? = "abc"
var yz : String? = "ABC"
println(ab) //"abc"
println(yz) //"ABC"
yz = ab
println(ab) //"abc"
println(yz) //"abc"
ab = null
println(ab) //null
println(yz) //"abc"
ab.uppercase() //erreur (impossible)
```

Le garbage collector efface les objets déréférencés

IUT Nantes

Les variables sont des références : exemple de `String?`

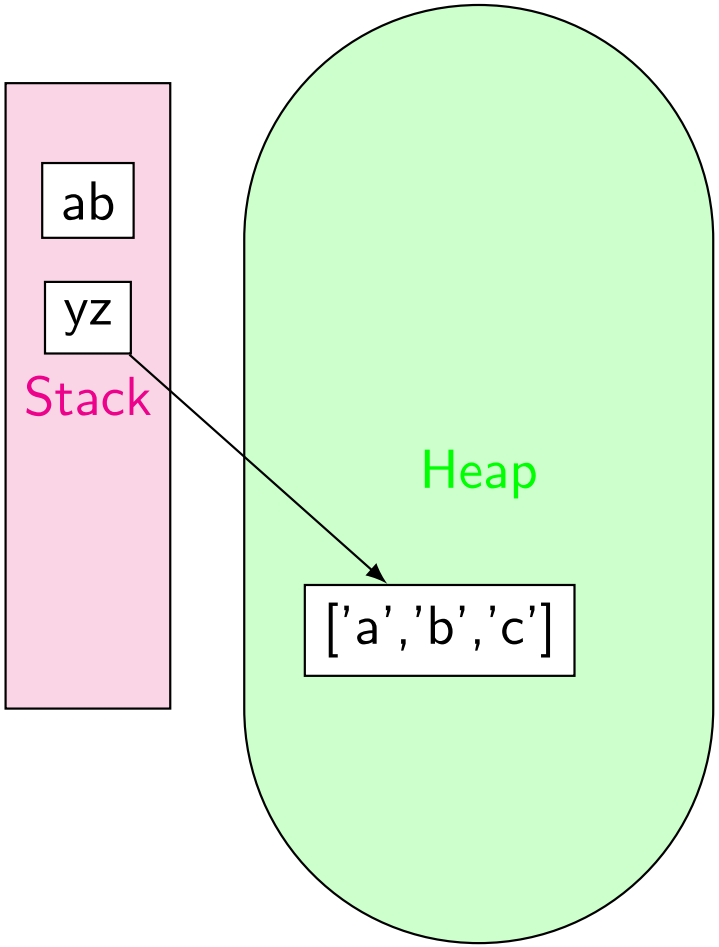



```
var ab : String? = "abc"
var yz : String? = "ABC"
println(ab) //"abc"
println(yz) //"ABC"
yz = ab
println(ab) //"abc"
println(yz) //"abc"
ab = null
println(ab) //null
println(yz) //"abc"
ab.uppercase() //erreur (impossible)
```

Les valeurs de `yz` et `ab` sont =

IUT Nantes

Les variables sont des références : exemple de `String?`

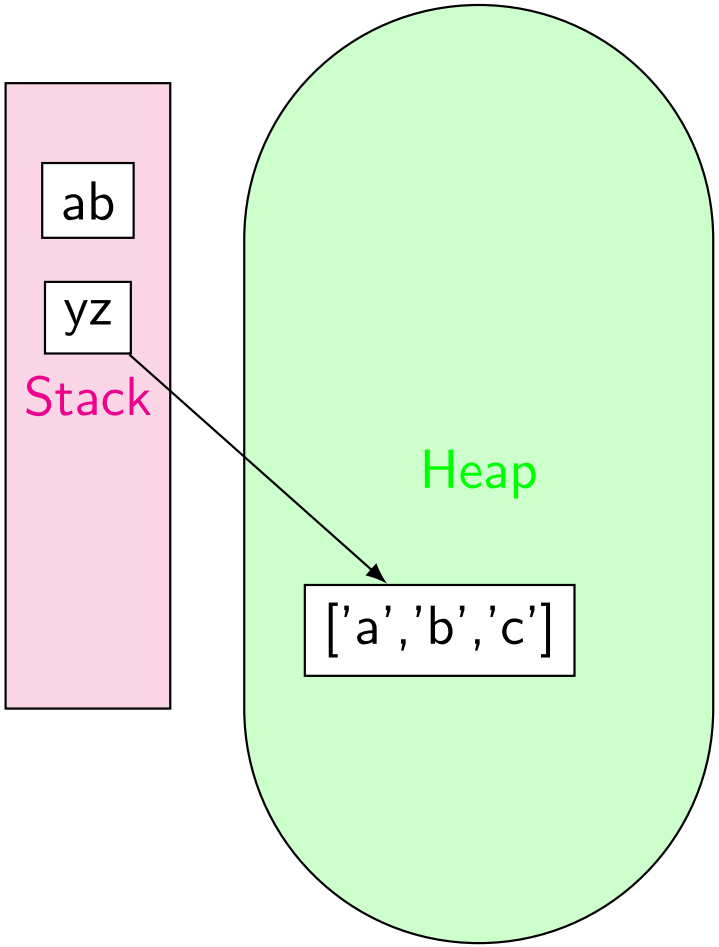



```
var ab : String? = "abc"
var yz : String? = "ABC"
println(ab) //"abc"
println(yz) //"ABC"
yz = ab
println(ab) //"abc"
println(yz) //"abc"
ab = null
println(ab) //null
println(yz) //"abc"
ab.uppercase() //erreur (impossible)
```

ab ne pointe plus vers rien

IUT Nantes

Les variables sont des références : exemple de `String?`

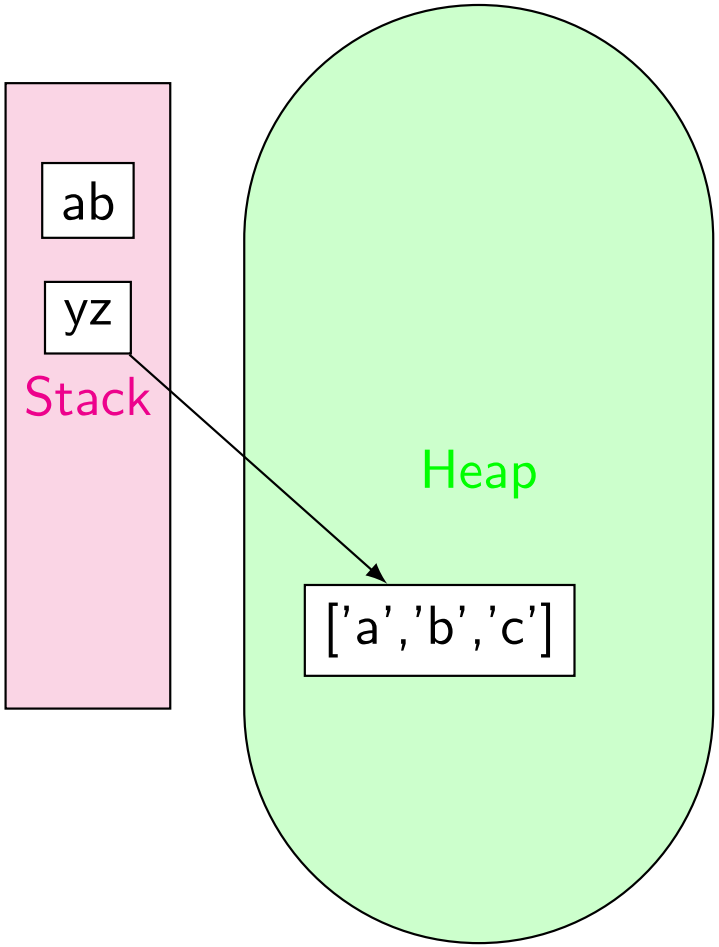



```
var ab : String? = "abc"
var yz : String? = "ABC"
println(ab) //"abc"
println(yz) //"ABC"
yz = ab
println(ab) //"abc"
println(yz) //"abc"
ab = null
println(ab) //null
println(yz) //"abc"
ab.uppercase() //erreur (impossible)
```

yz n'est pas affecté par la mise à null de ab

IUT Nantes

Les variables sont des références : exemple de `String?`




```
var ab : String? = "abc"
var yz : String? = "ABC"
println(ab) //"abc"
println(yz) //"ABC"
yz = ab
println(ab) //"abc"
println(yz) //"abc"
ab = null
println(ab) //null
println(yz) //"abc"
ab.uppercase() //erreur (impossible)
```

Cet appel PROVOQUERAIT une erreur, car `ab` ne pointe vers rien Le compilateur INTERDIT cet appel : il faut "assumer" le risque pris

IUT Nantes

Utiliser des variables *nullable*

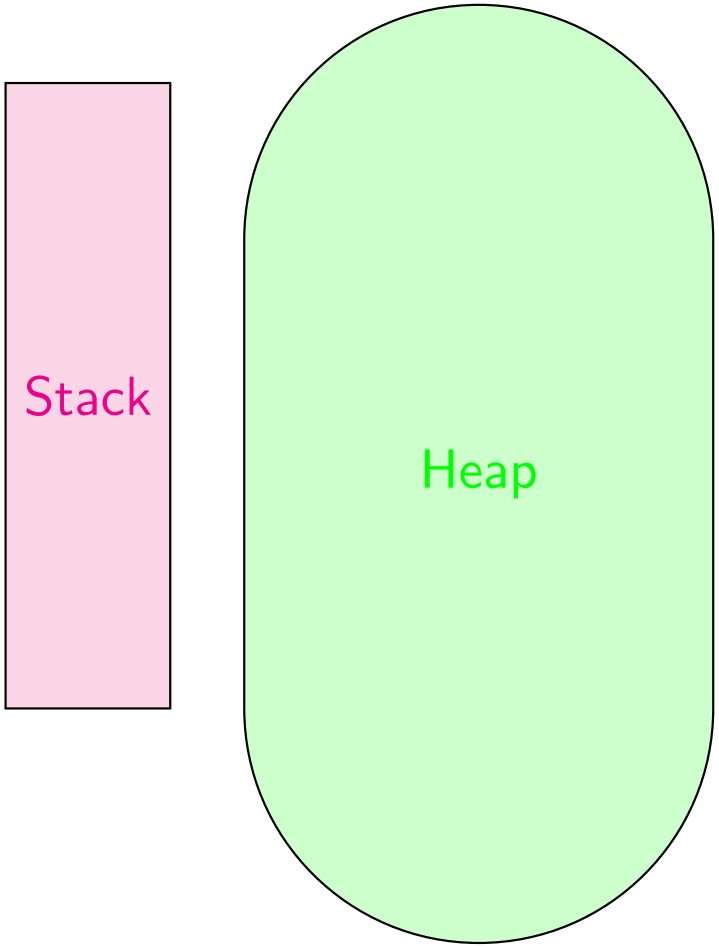
Kotlin vérrouille l'accès aux variables *nullable*

1. Réaliser des appels "sûrs" via `?` :
   `z?.length` retourne `z.length` si `z` $\neq$ `null` sinon retourne `null`
2. Utiliser l'opérateur Elvis `?:`
   `z?.length ?: 0` : si la partie gauche, ici `z?.length`, = `null` alors on retourne la partie droite, ici `0`
3. Forcer l'évaluation via `!!` :
   `z!!` retourne une version non-nulle de `z` si `z` $\neq$ `null`
   mais si `z` = `null`
   NullPointerException

```
var z : String? = "totoro"
...
...
//val l = z.length
// erreur de compilation

var l = z?.length
println(l)

l = z!!.length
println(l)

l = z?.length ?: 0
println(l)
```

IUT Nantes

Les tableaux contiennent (aussi) des références

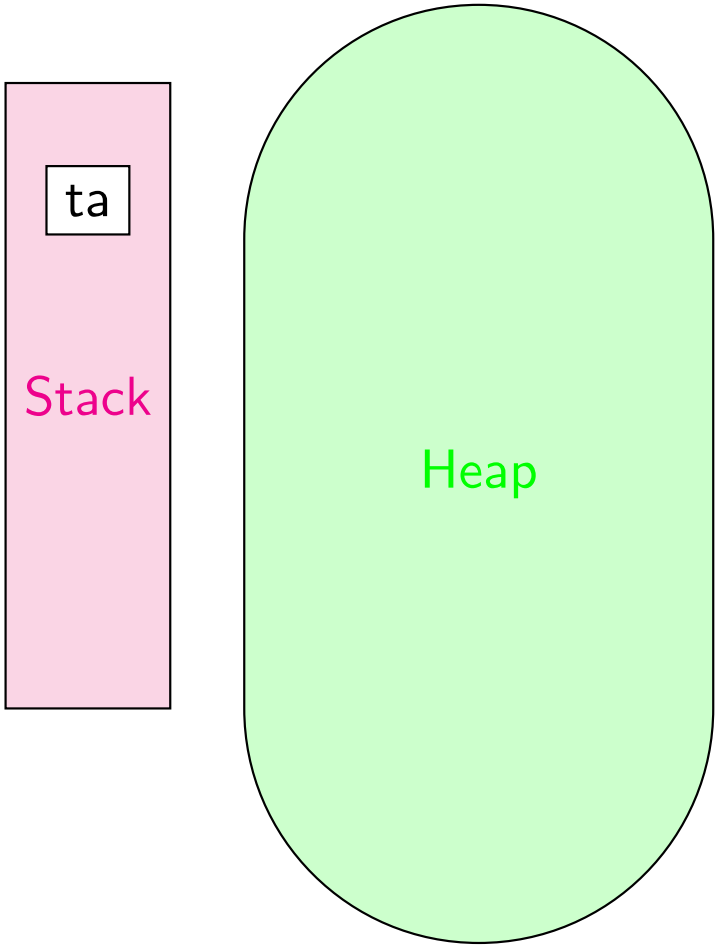



```
var ta : Array<Int?>
var tz : Array<Int?>
ta = arrayOf(3, 5, 7)
tz = ta
ta[2] = 2
tz[1] = 4
tz[0] = null
ta = arrayOfNulls<Int>(4)
ta[3] = tz[2]
ta[2] = tz[1]
```

Schématisation de la mémoire de la JVM

IUT Nantes

Les tableaux contiennent (aussi) des références

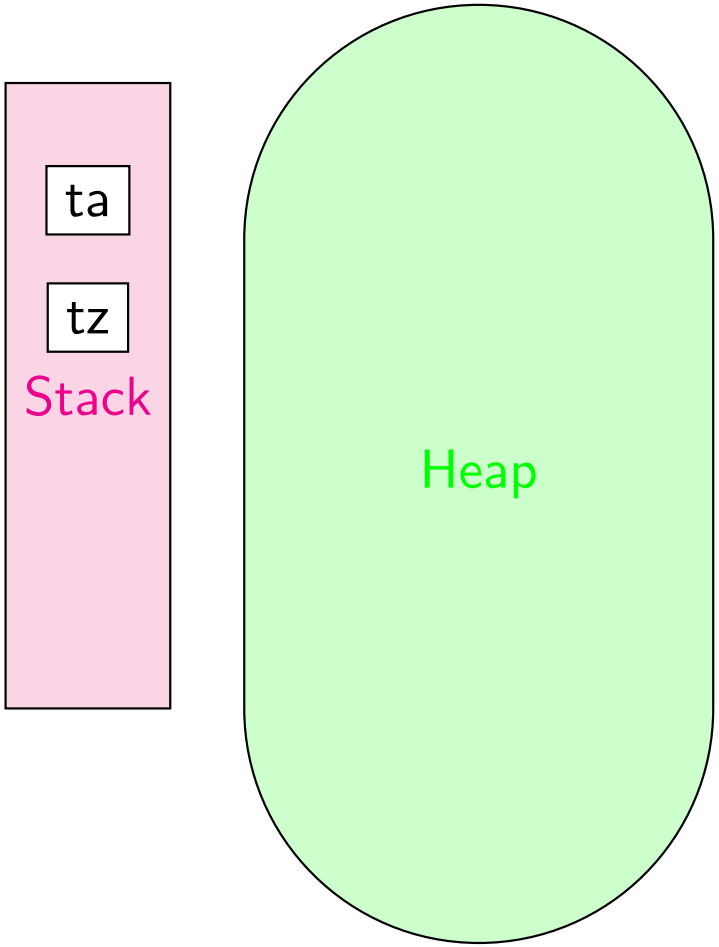



```
var ta : Array<Int?>
var tz : Array<Int?>
ta = arrayOf(3, 5, 7)
tz = ta
ta[2] = 2
tz[1] = 4
tz[0] = null
ta = arrayOfNulls<Int>(4)
ta[3] = tz[2]
ta[2] = tz[1]
```

`ta` créé dans la pile mémoire

IUT Nantes

Les tableaux contiennent (aussi) des références

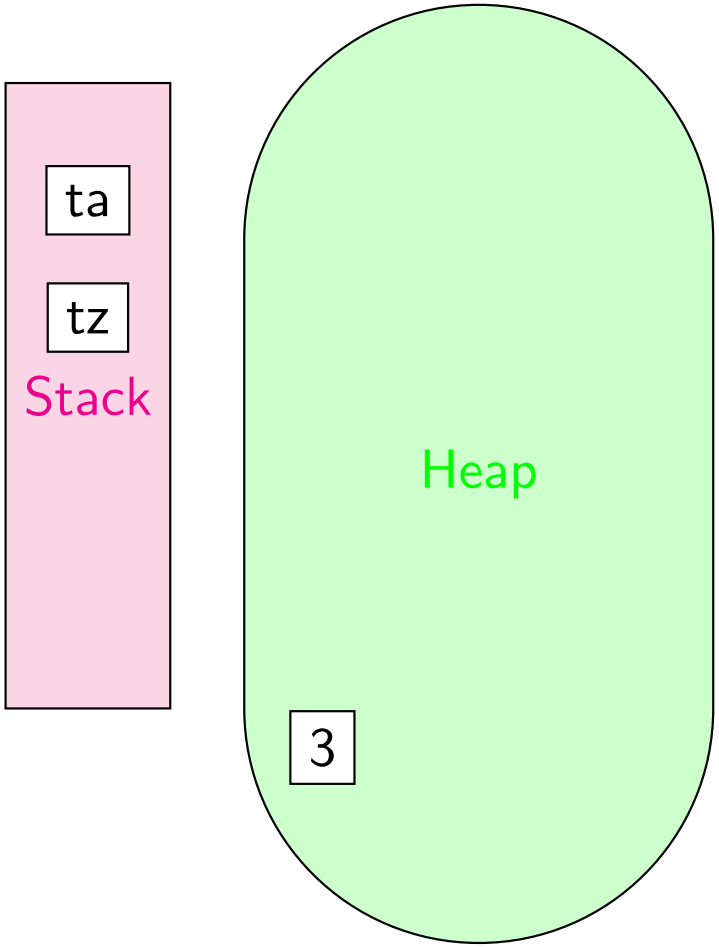



```
var ta : Array<Int?>
var tz : Array<Int?>
ta = arrayOf(3, 5, 7)
tz = ta
ta[2] = 2
tz[1] = 4
tz[0] = null
ta = arrayOfNulls<Int>(4)
ta[3] = tz[2]
ta[2] = tz[1]
```

`tz` créé dans la pile mémoire

IUT Nantes

Les tableaux contiennent (aussi) des références

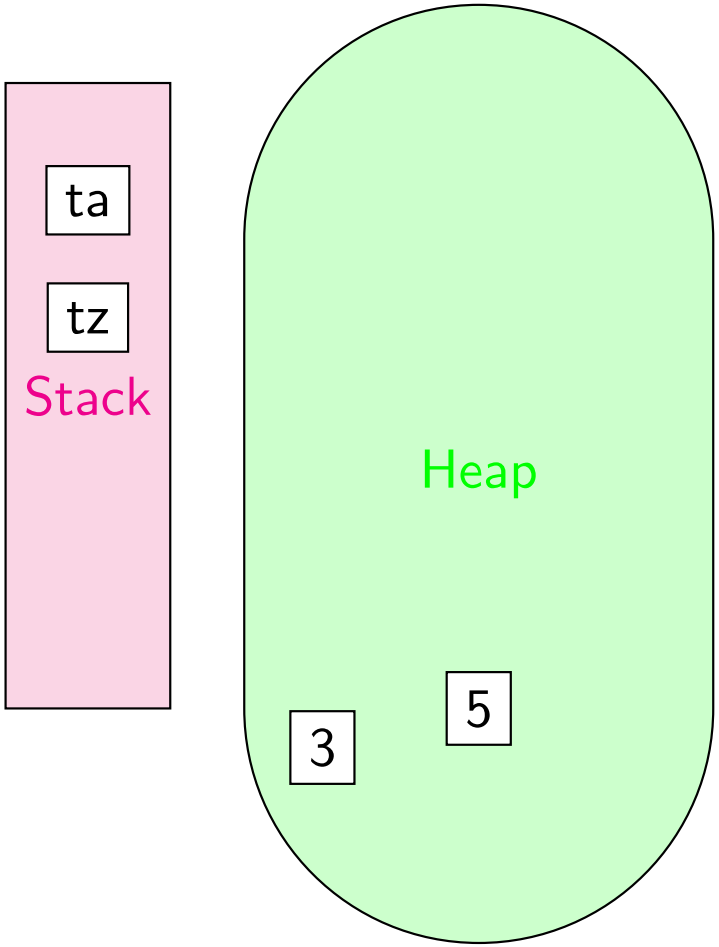



```
var ta : Array<Int?>
var tz : Array<Int?>
ta = arrayOf(3, 5, 7)
tz = ta
ta[2] = 2
tz[1] = 4
tz[0] = null
ta = arrayOfNulls<Int>(4)
ta[3] = tz[2]
ta[2] = tz[1]
```

3 créé dans le tas mémoire

IUT Nantes

Les tableaux contiennent (aussi) des références

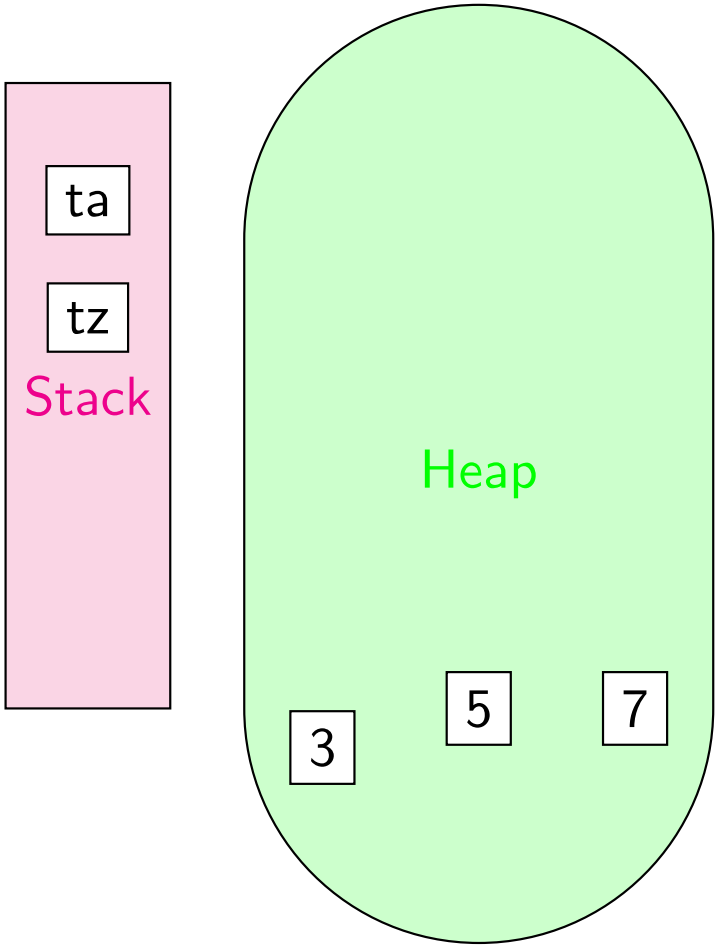



```
var ta : Array<Int?>
var tz : Array<Int?>
ta = arrayOf(3, 5, 7)
tz = ta
ta[2] = 2
tz[1] = 4
tz[0] = null
ta = arrayOfNulls<Int>(4)
ta[3] = tz[2]
ta[2] = tz[1]
```

5 créé dans le tas mémoire

IUT Nantes

Les tableaux contiennent (aussi) des références

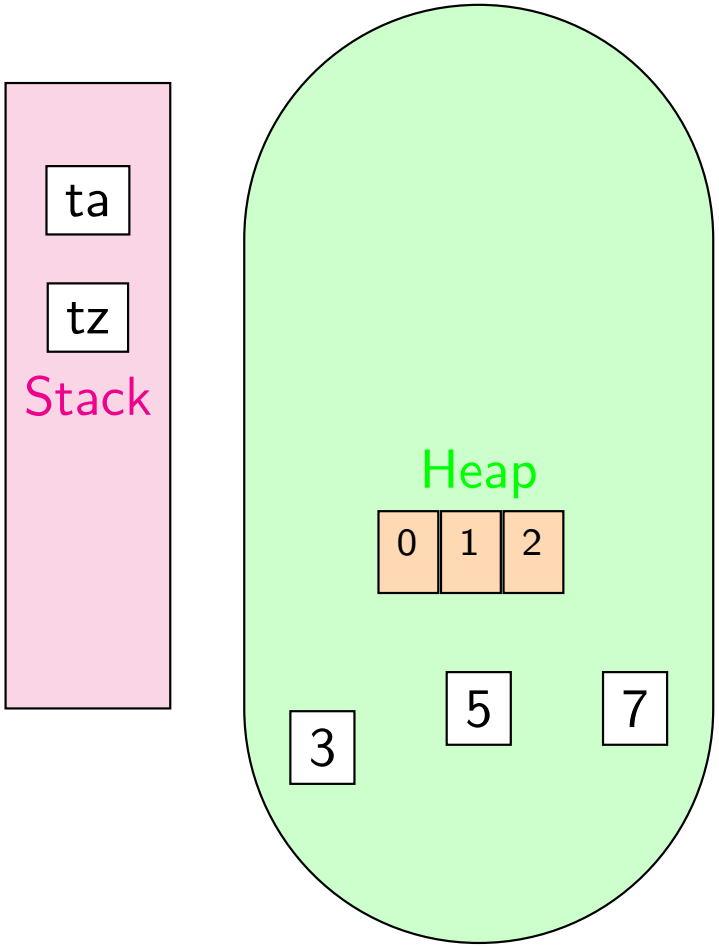



```
var ta : Array<Int?>
var tz : Array<Int?>
ta = arrayOf(3, 5, 7)
tz = ta
ta[2] = 2
tz[1] = 4
tz[0] = null
ta = arrayOfNulls<Int>(4)
ta[3] = tz[2]
ta[2] = tz[1]
```

7 créé dans le tas mémoire

IUT Nantes

Les tableaux contiennent (aussi) des références

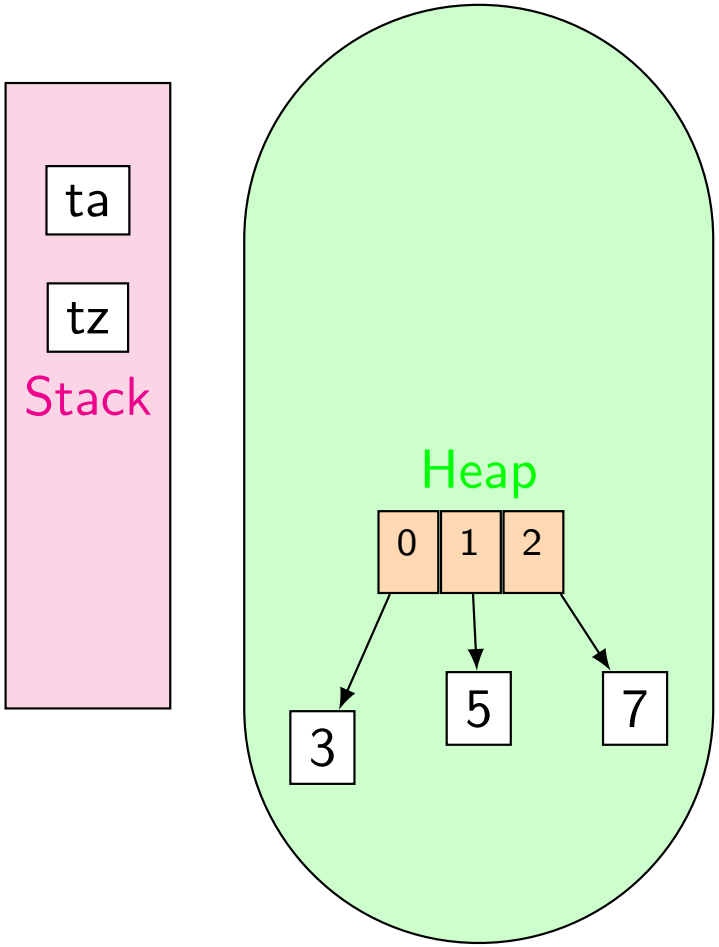



```
var ta : Array<Int?>
var tz : Array<Int?>
ta = arrayOf(3, 5, 7)
tz = ta
ta[2] = 2
tz[1] = 4
tz[0] = null
ta = arrayOfNulls<Int>(4)
ta[3] = tz[2]
ta[2] = tz[1]
```

un tableau `Array<Int?>(3)` créé dans le tas mémoire

IUT Nantes

Les tableaux contiennent (aussi) des références

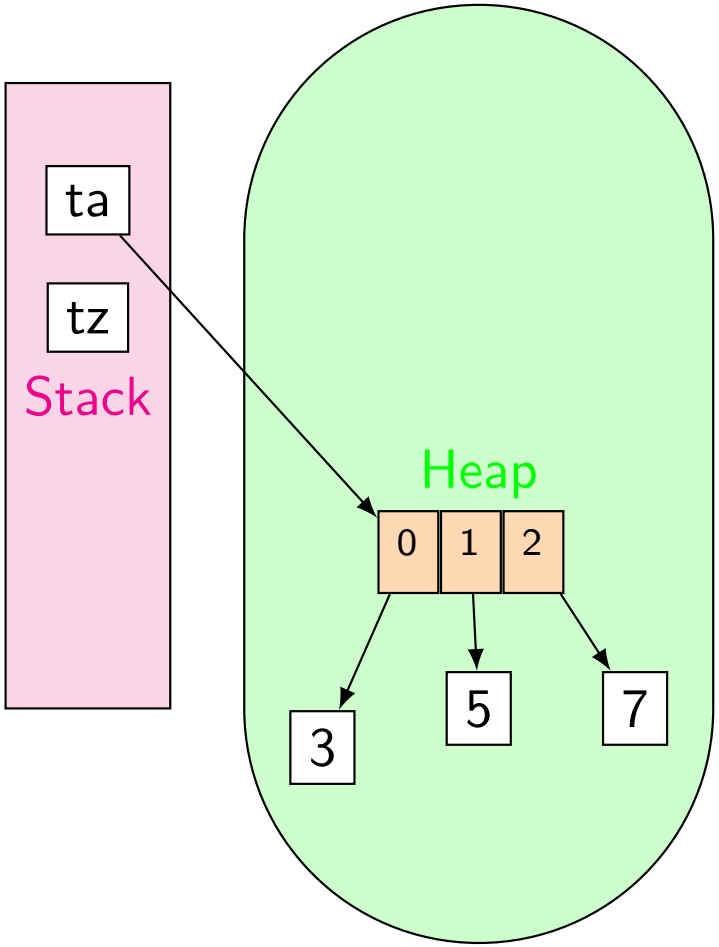



```
var ta : Array<Int?>
var tz : Array<Int?>
ta = arrayOf(3, 5, 7)
tz = ta
ta[2] = 2
tz[1] = 4
tz[0] = null
ta = arrayOfNulls<Int>(4)
ta[3] = tz[2]
ta[2] = tz[1]
```

Le tableau "pointe" vers les valeurs précédemment créées

IUT Nantes

Les tableaux contiennent (aussi) des références

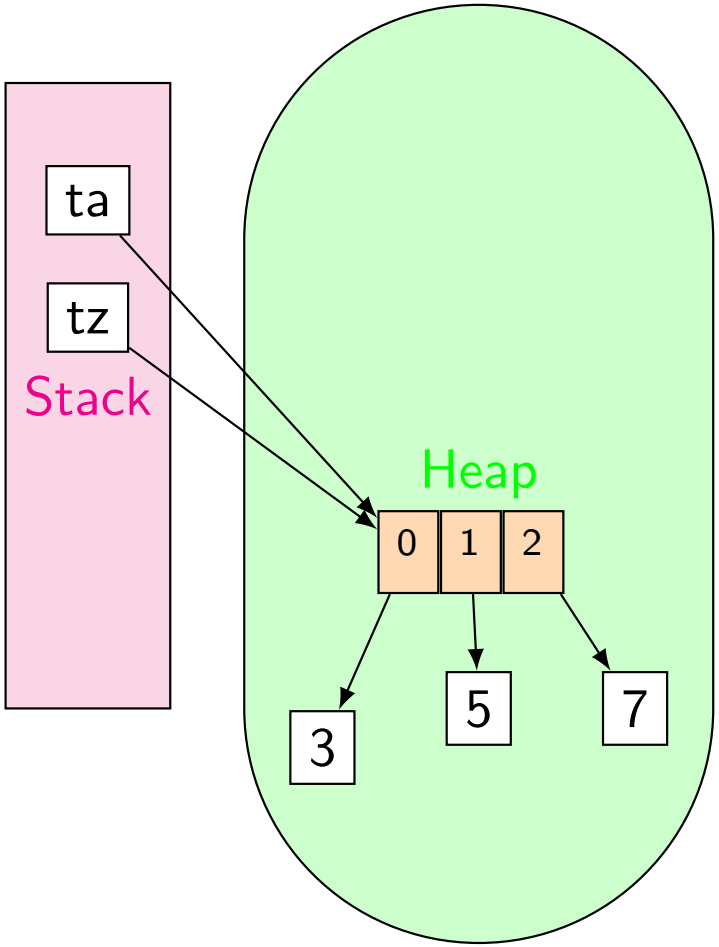



```
var ta : Array<Int?>
var tz : Array<Int?>
ta = arrayOf(3, 5, 7)
tz = ta
ta[2] = 2
tz[1] = 4
tz[0] = null
ta = arrayOfNulls<Int>(4)
ta[3] = tz[2]
ta[2] = tz[1]
```

`ta` "pointe" vers le tableau

IUT Nantes

Les tableaux contiennent (aussi) des références

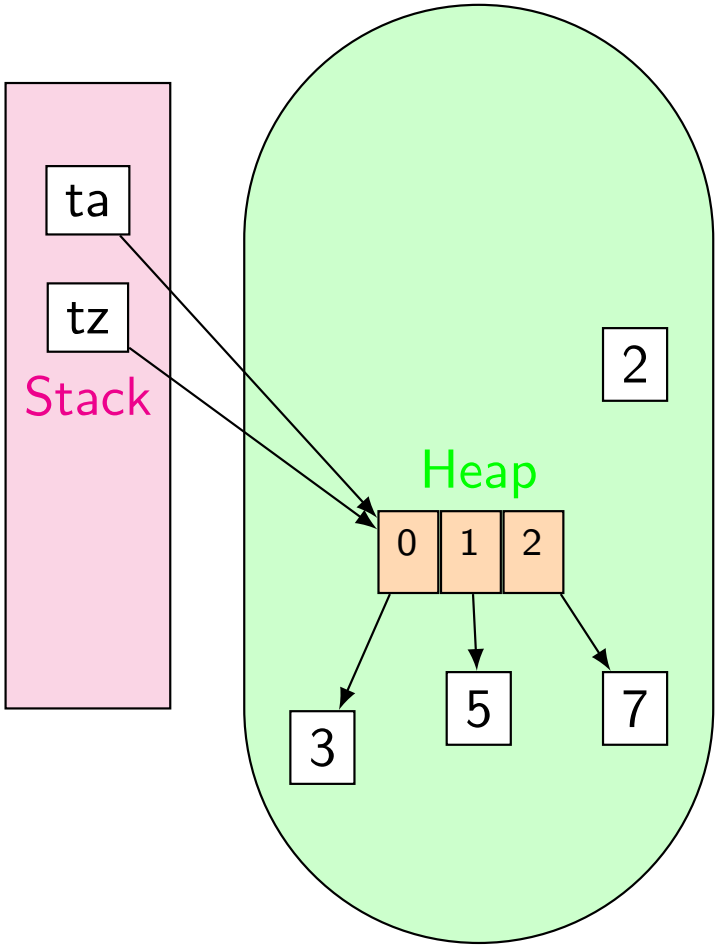



```
var ta : Array<Int?>
var tz : Array<Int?>
ta = arrayOf(3, 5, 7)
tz = ta
ta[2] = 2
tz[1] = 4
tz[0] = null
ta = arrayOfNulls<Int>(4)
ta[3] = tz[2]
ta[2] = tz[1]
```

`tz` "pointe" aussi vers le tableau

IUT Nantes

Les tableaux contiennent (aussi) des références

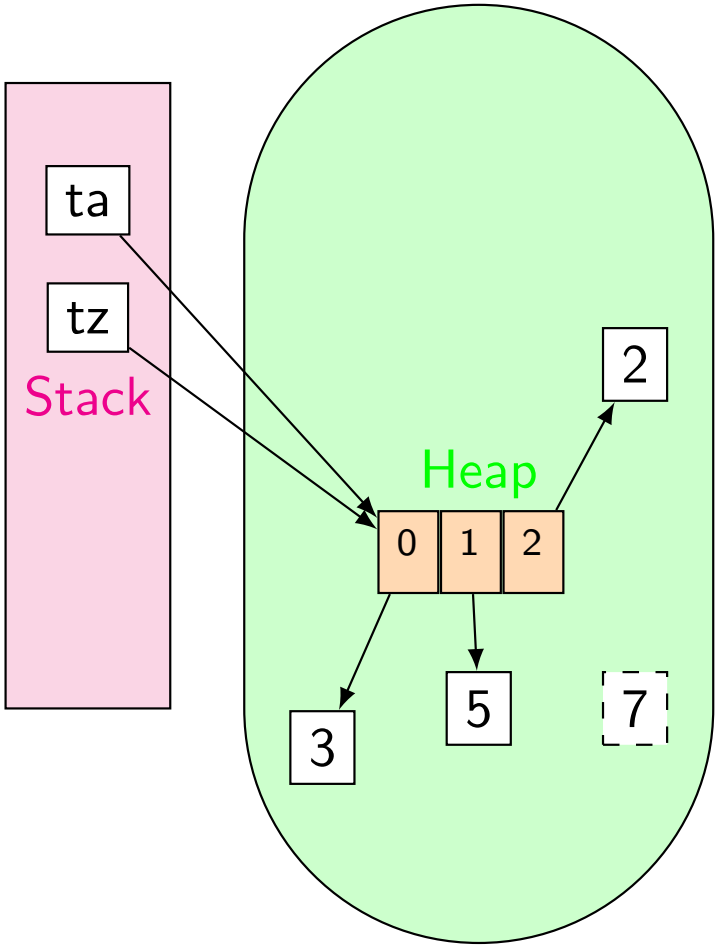



```
var ta : Array<Int?>
var tz : Array<Int?>
ta = arrayOf(3, 5, 7)
tz = ta
ta[2] = 2
tz[1] = 4
tz[0] = null
ta = arrayOfNulls<Int>(4)
ta[3] = tz[2]
ta[2] = tz[1]
```

2 créé dans le tas mémoire

IUT Nantes

Les tableaux contiennent (aussi) des références

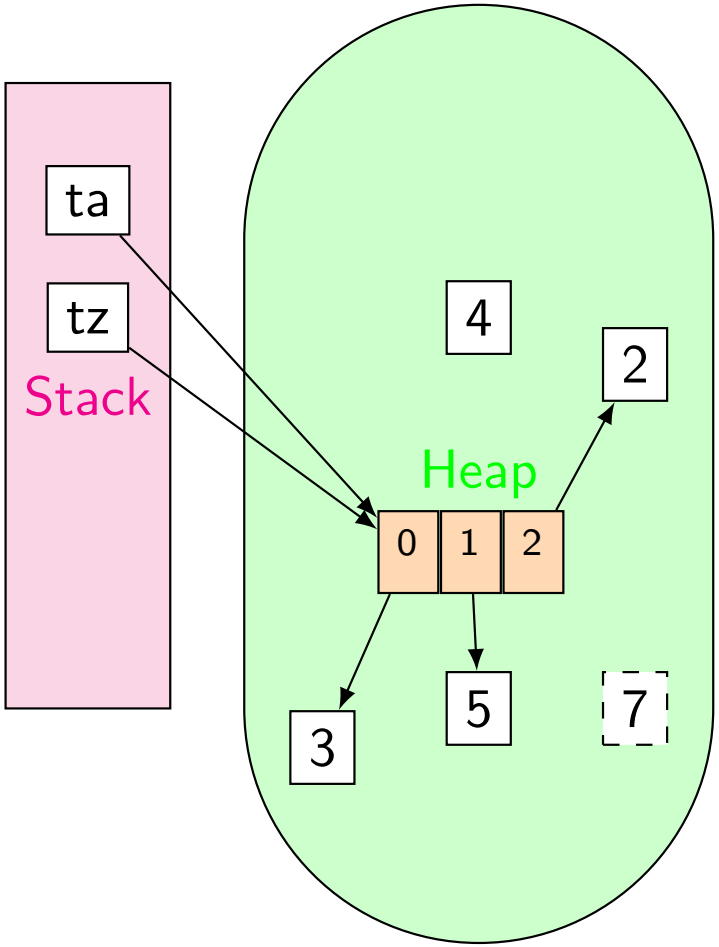



```
var ta : Array<Int?>
var tz : Array<Int?>
ta = arrayOf(3, 5, 7)
tz = ta
ta[2] = 2
tz[1] = 4
tz[0] = null
ta = arrayOfNulls<Int>(4)
ta[3] = tz[2]
ta[2] = tz[1]
```

le tableau "pointe" maintenant vers 2

IUT Nantes

Les tableaux contiennent (aussi) des références

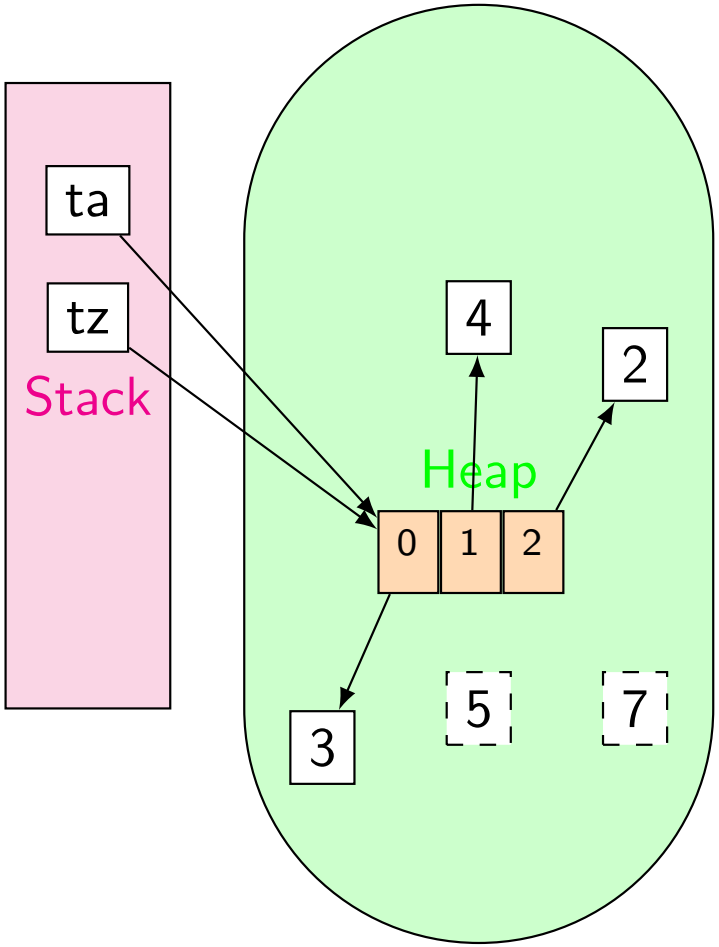



```
var ta : Array<Int?>
var tz : Array<Int?>
ta = arrayOf(3, 5, 7)
tz = ta
ta[2] = 2
tz[1] = 4
tz[0] = null
ta = arrayOfNulls<Int>(4)
ta[3] = tz[2]
ta[2] = tz[1]
```

4 créé dans le tas mémoire

IUT Nantes

Les tableaux contiennent (aussi) des références

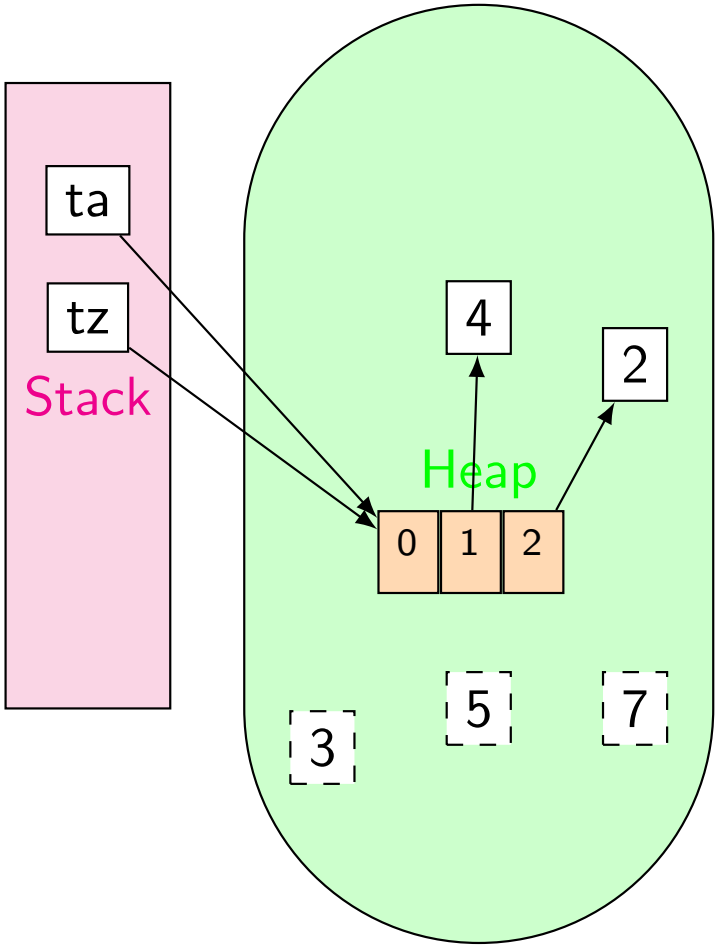



```
var ta : Array<Int?>
var tz : Array<Int?>
ta = arrayOf(3, 5, 7)
tz = ta
ta[2] = 2
tz[1] = 4
tz[0] = null
ta = arrayOfNulls<Int>(4)
ta[3] = tz[2]
ta[2] = tz[1]
```

le tableau "pointe" maintenant vers 4

IUT Nantes

Les tableaux contiennent (aussi) des références

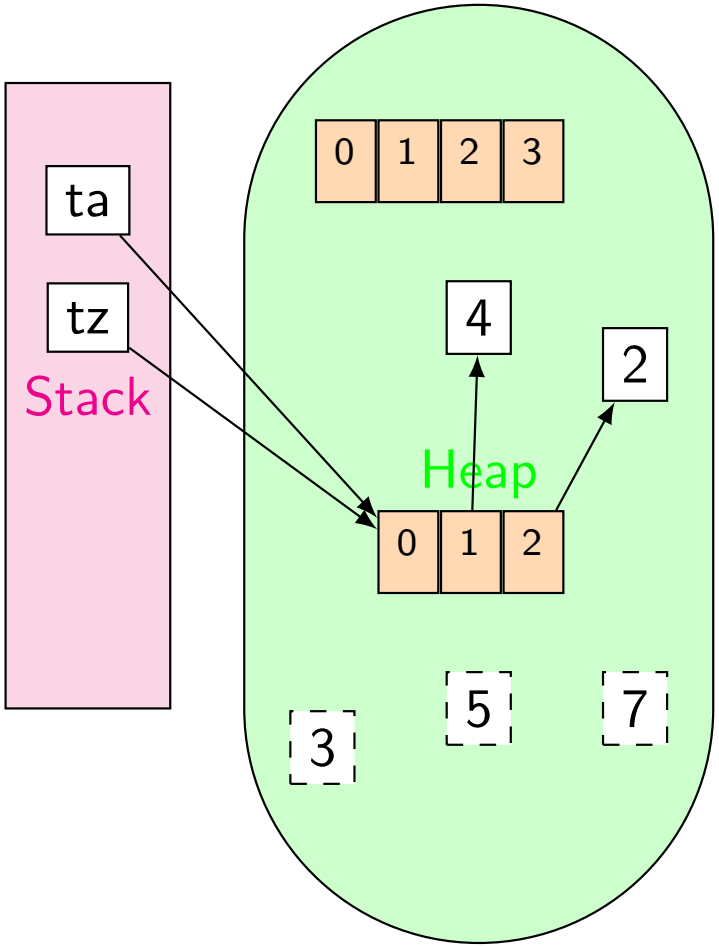



```
var ta : Array<Int?>
var tz : Array<Int?>
ta = arrayOf(3, 5, 7)
tz = ta
ta[2] = 2
tz[1] = 4
tz[0] = null
ta = arrayOfNulls<Int>(4)
ta[3] = tz[2]
ta[2] = tz[1]
```

3 n'est plus référencé

IUT Nantes

Les tableaux contiennent (aussi) des références

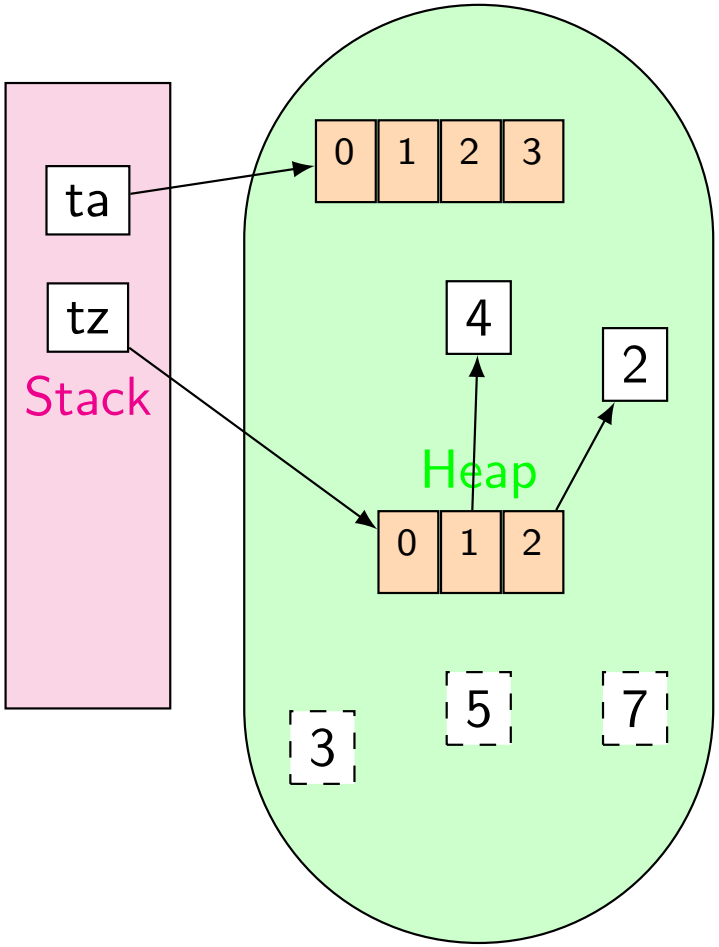



```
var ta : Array<Int?>
var tz : Array<Int?>
ta = arrayOf(3, 5, 7)
tz = ta
ta[2] = 2
tz[1] = 4
tz[0] = null
ta = arrayOfNulls<Int>(4)
ta[3] = tz[2]
ta[2] = tz[1]
```

Un nouveau tableau `Array<Int?>(4)` est créé

IUT Nantes

Les tableaux contiennent (aussi) des références

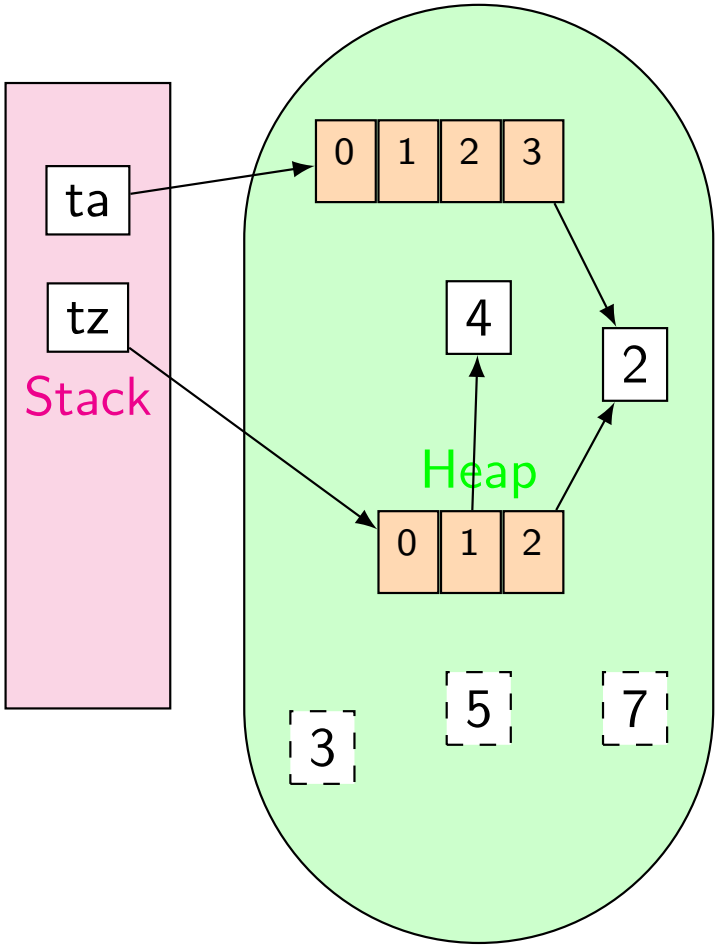



```
var ta : Array<Int?>
var tz : Array<Int?>
ta = arrayOf(3, 5, 7)
tz = ta
ta[2] = 2
tz[1] = 4
tz[0] = null
ta = arrayOfNulls<Int>(4)
ta[3] = tz[2]
ta[2] = tz[1]
```

`ta` "pointe" vers le nouveau tableau

IUT Nantes

Les tableaux contiennent (aussi) des références

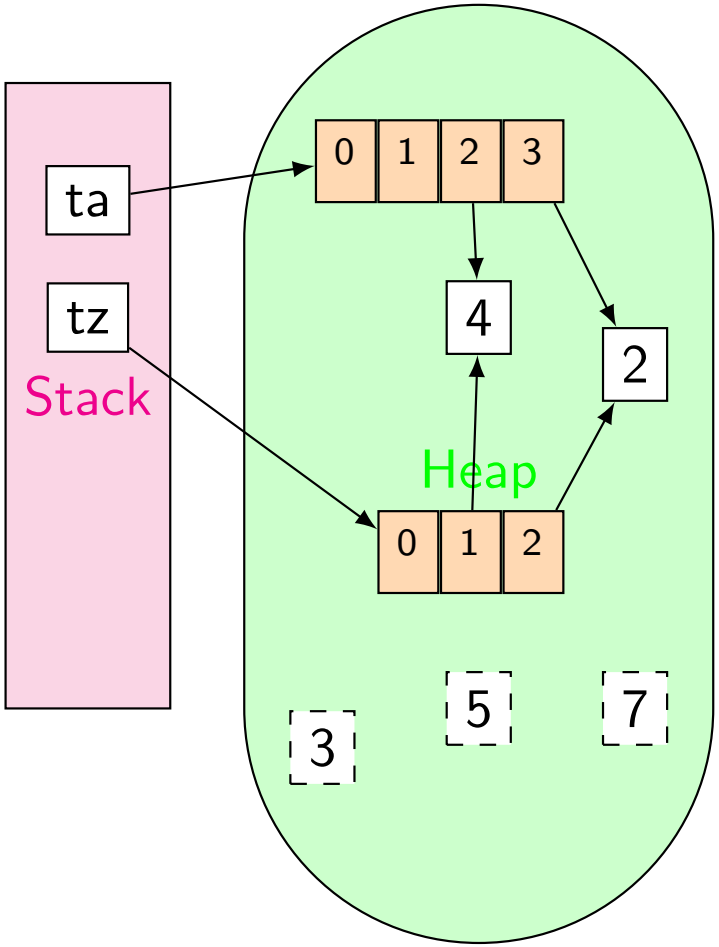



```
var ta : Array<Int?>
var tz : Array<Int?>
ta = arrayOf(3, 5, 7)
tz = ta
ta[2] = 2
tz[1] = 4
tz[0] = null
ta = arrayOfNulls<Int>(4)
ta[3] = tz[2]
ta[2] = tz[1]
```

le nouveau tableau "pointe" maintenant vers 2

IUT Nantes

Les tableaux contiennent (aussi) des références

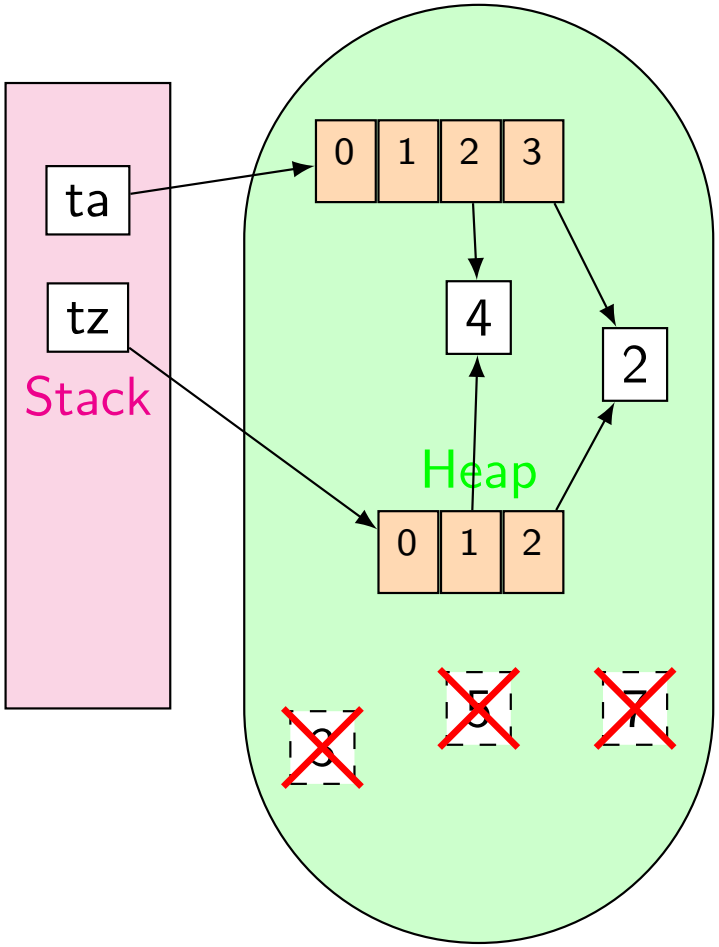



```
var ta : Array<Int?>
var tz : Array<Int?>
ta = arrayOf(3, 5, 7)
tz = ta
ta[2] = 2
tz[1] = 4
tz[0] = null
ta = arrayOfNulls<Int>(4)
ta[3] = tz[2]
ta[2] = tz[1]
```

le tableau "pointe" maintenant vers 4

IUT Nantes

Les tableaux contiennent (aussi) des références

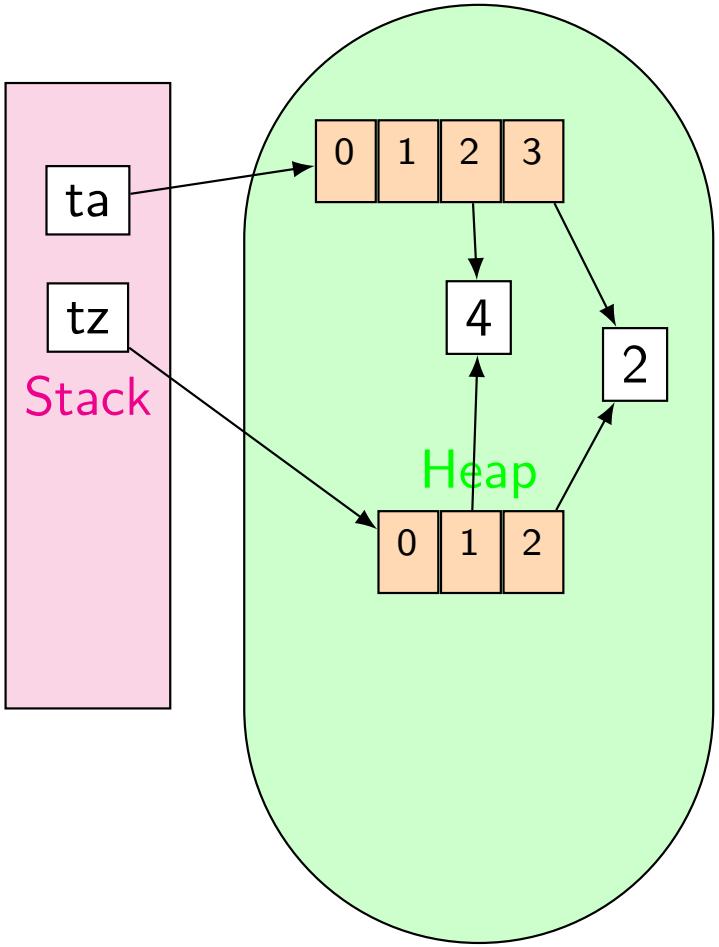



```
var ta : Array<Int?>
var tz : Array<Int?>
ta = arrayOf(3, 5, 7)
tz = ta
ta[2] = 2
tz[1] = 4
tz[0] = null
ta = arrayOfNulls<Int>(4)
ta[3] = tz[2]
ta[2] = tz[1]
```

Le garbage collector efface les objets déréférencés

IUT Nantes

Les tableaux contiennent (aussi) des références




```
var ta : Array<Int?>
var tz : Array<Int?>
ta = arrayOf(3, 5, 7)
tz = ta
ta[2] = 2
tz[1] = 4
tz[0] = null
ta = arrayOfNulls<Int>(4)
ta[3] = tz[2]
ta[2] = tz[1]
```

Le garbage collector efface les objets déréférencés

IUT Nantes