Structures chaînées

Sébastien Jean

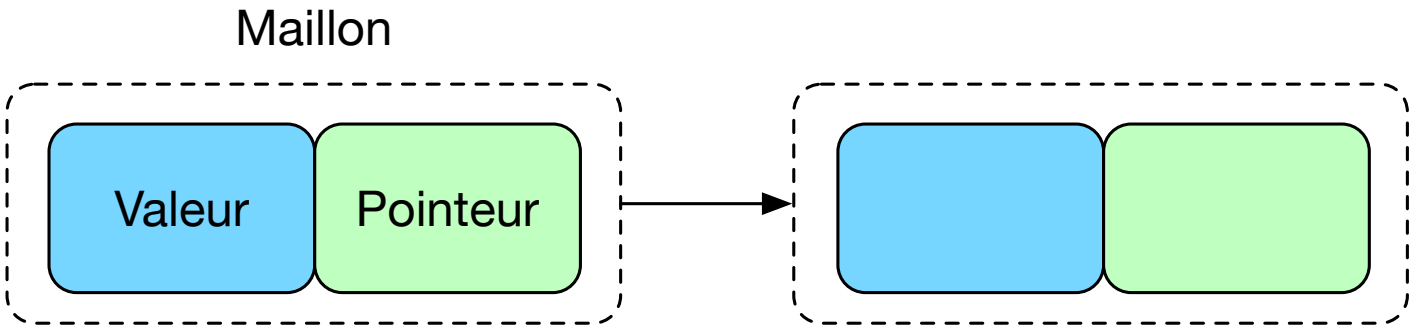
IUT de Valence
Département Informatique

v1.0, 15 décembre 2025

IUT Valence
Université Grenoble Alpes

Structure chaînée

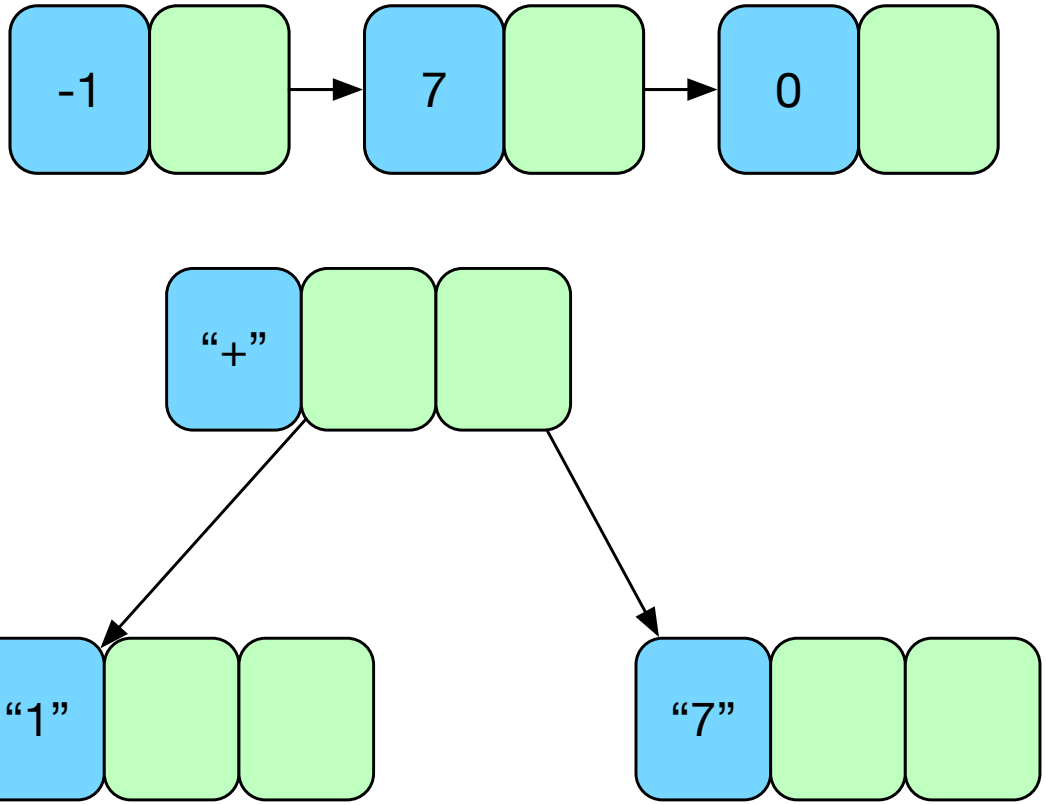
- Une **structure chaînée** est un moyen d'organiser des données dynamiquement, en établissant des liaisons exprimant des relations
  - Prédécesseur/ successeur , ...
- L'élément de base d'une structure chaînée est un **maillon**, composé :
  - d'une valeur (valeur simple, enregistrement, collection)
  - d'un (ou plusieurs) pointeur(s), matérialisant la ou les liaisons avec d'autres maillons



IUT Valence
Université Grenoble Alpes

Structure chaînée

- Les structures chaînées permettent de mettre en œuvre des **vecteurs**, des **piles**, des **files**, des **arbres**, ...

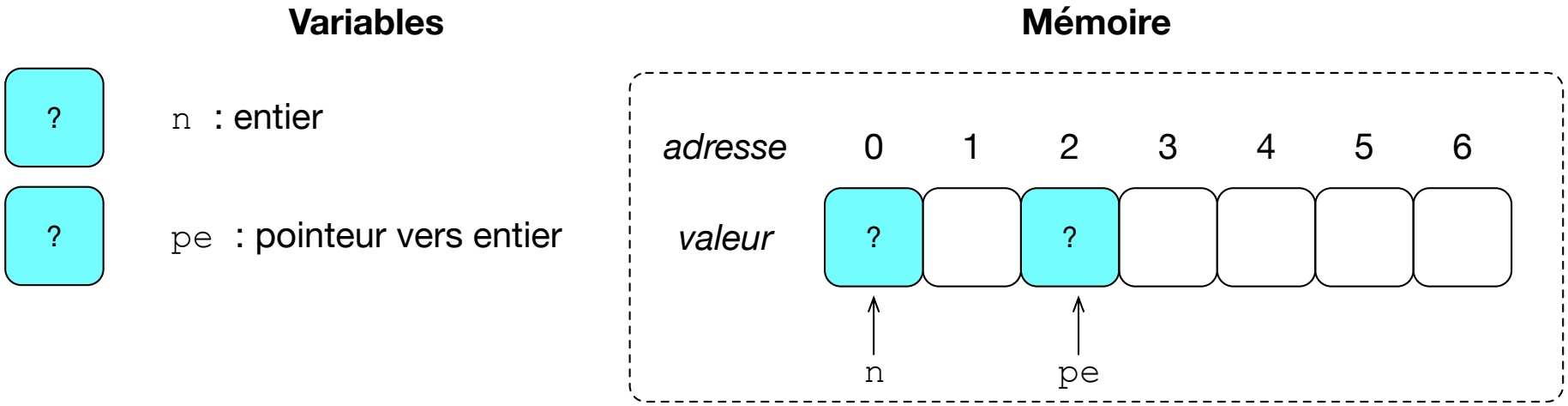


IUT Valence
Université Grenoble Alpes

Rappels : variable de type pointeur

- Une **variable de type *pointeur vers un type T*** est une **variable dont la valeur est l'adresse d'un emplacement mémoire** où est stockée une valeur de type T

```
VARIABLE pe : pointeur vers entier
```



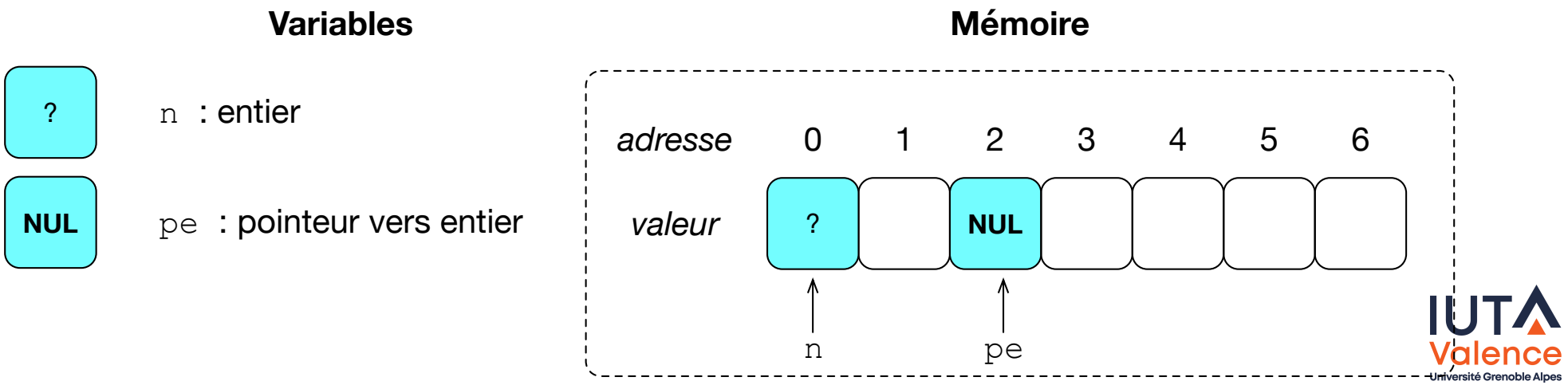
IUT Valence Université Grenoble Alpes

Rappels : initialisation d'une variable de type pointeur

- Une variable de type *pointeur vers un type T* ne doit pas être lue si elle n'est pas initialisée
- On peut **initialiser une variable de type pointeur** avec la valeur `NUL` pour indiquer qu'elle ne désigne pas encore d'emplacement
  - Elle peut alors être lue, on saura qu'elle ne désigne pas encore d'emplacement

---

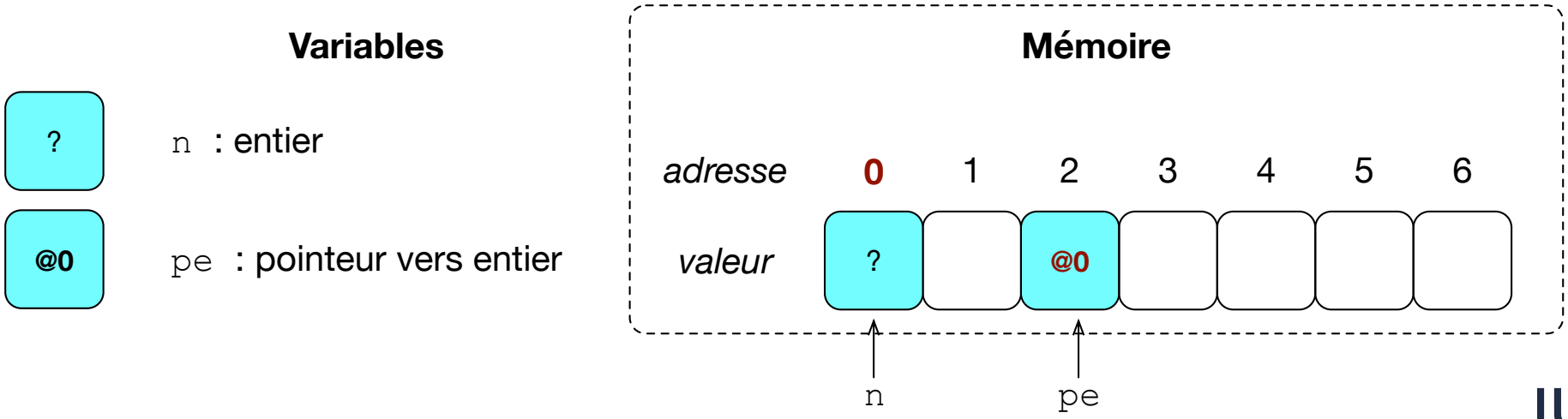
`pe` ← `NUL`



Rappels : initialisation d'une variable de type pointeur

- On ne peut pas « fabriquer d'adresse », **on ne peut affecter comme valeur à une variable de type pointeur que l'adresse d'une variable (ou d'un paramètre) du type attendu**
- l'opérateur & désigne l'**adresse d'une variable**

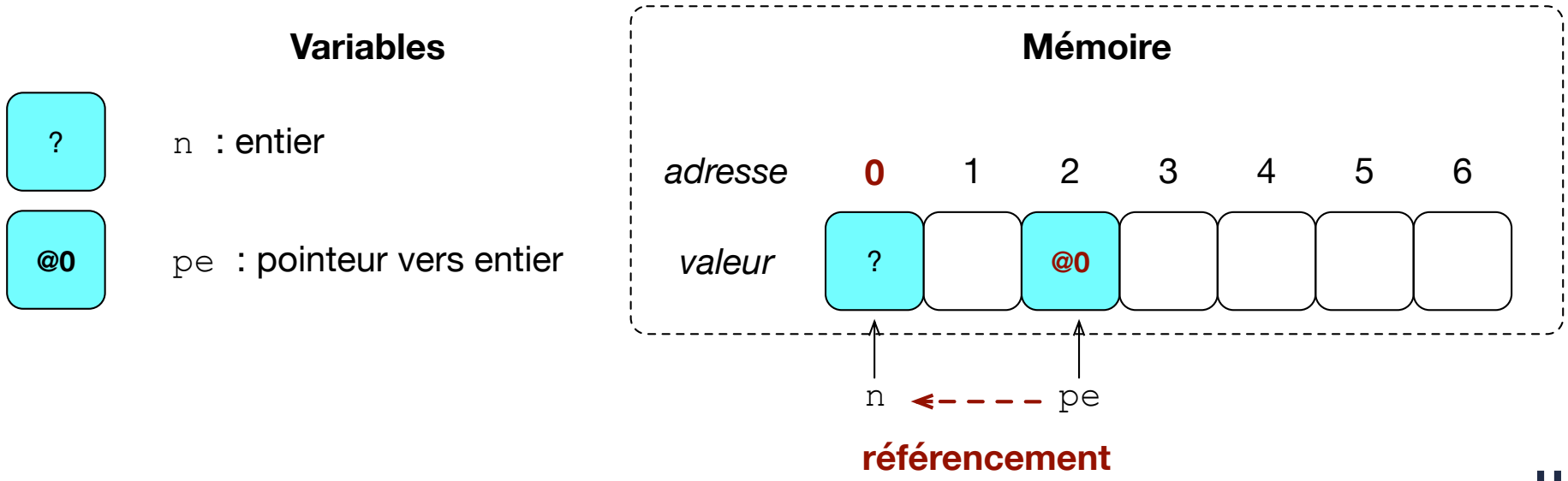
```
pe ← &n
```



Rappels : initialisation d'une variable de type pointeur

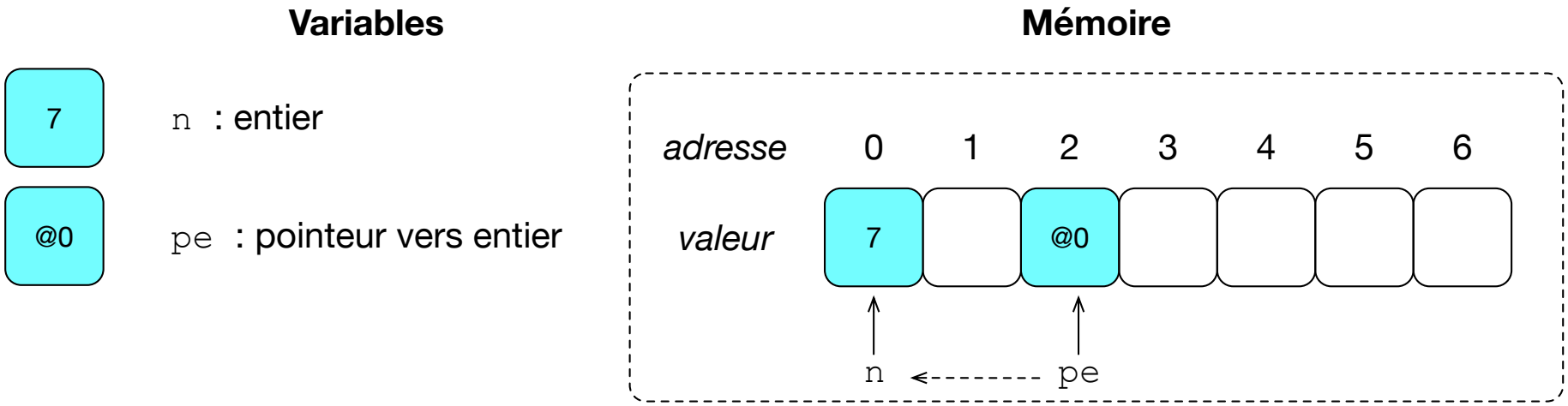
- La valeur d'une variable de type pointeur vers le type `T` est l'**adresse de l'emplacement mémoire d'une valeur de type `T`**
  - On dit que le pointeur **référence** la variable

```
pe ← &n
```



Rappels : déréférencement d'une variable de type pointeur

- L'opérateur ↑ exprime le **déréférencement**, c'est à dire la **désignation de l'emplacement mémoire dont l'adresse est contenue dans le pointeur**



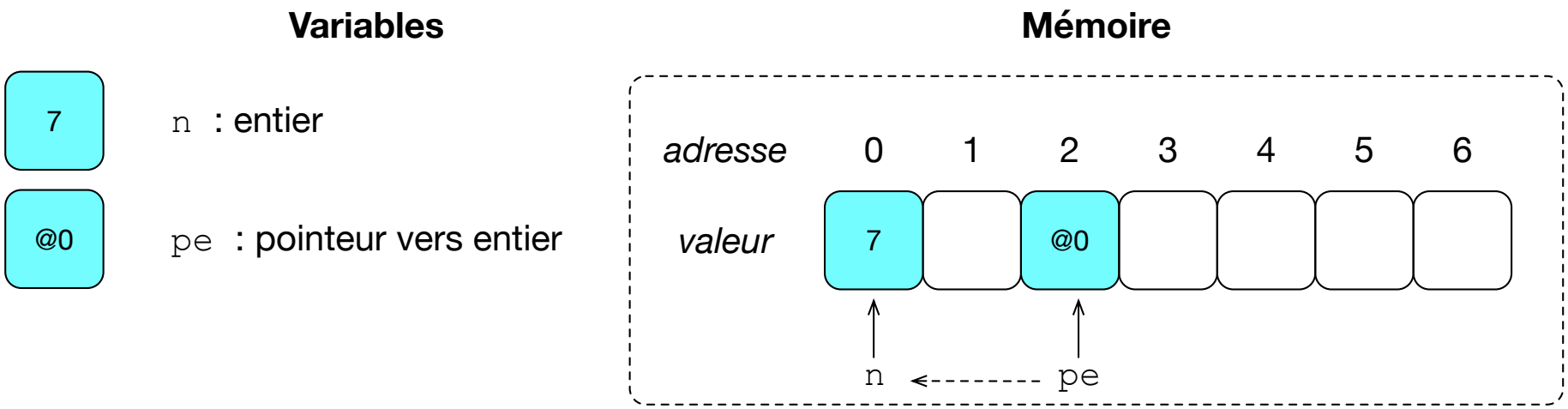
- Un pointeur dont la valeur est `NUL` ou dont la valeur désigne un emplacement non initialisé ne doit pas être déréférencé

IUT Valence Université Grenoble Alpes

Rappels : déréférencement d'une variable de type pointeur

- L'opérateur ↑ peut être utilisé dans une expression
  - Dans ce cas l'**expression vaut la valeur contenue dans l'emplacement dont l'adresse est contenue dans le pointeur**

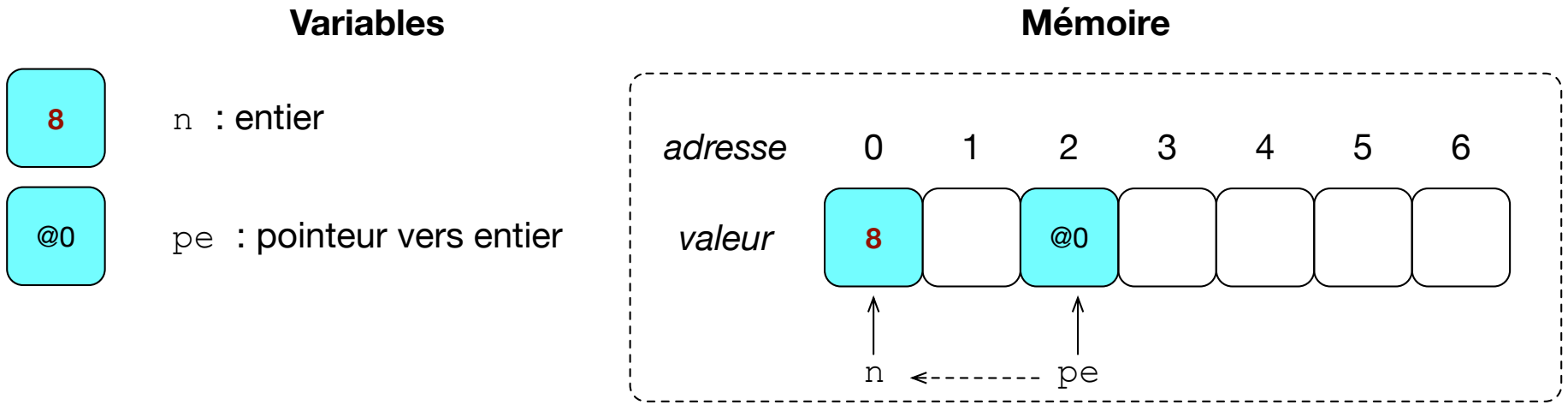
```
afficher_entier(pe↑)                    // affiche 7
```



Rappels : déréférencement d'une variable de type pointeur

- L'opérateur ↑ peut être utilisé dans une affectation
  - Dans ce cas l'**emplacement dont l'adresse est contenue dans le pointeur est réaffecté avec la nouvelle valeur**

`pe↑ ← 8`



`pe↑ ← pe↑ +1` `// ?`

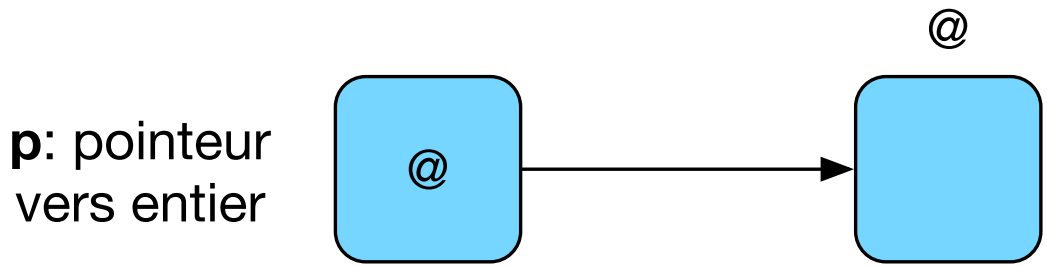
IUT Valence Université Grenoble Alpes

Rappels : allocation dynamique de mémoire

- **Allocation dynamique**
  - Réservation de zone mémoire *à la volée*, sans variable

```
VARIABLE p : pointeur vers entier

p ← ALLOUER entier
```

- `ALLOUER` réserve une nouvelle zone mémoire de taille adaptée au stockage d'une valeur de type donné et **retourne l'adresse de début de la zone mémoire**



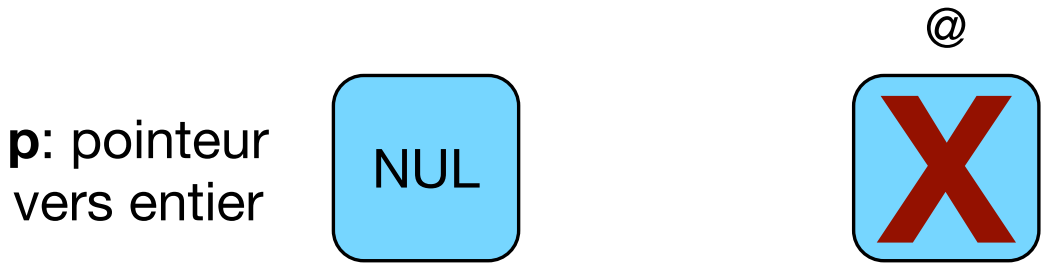
IUT Valence
Université Grenoble Alpes

Rappels : allocation dynamique de mémoire

- **Libération de mémoire**
  - En théorie : pas de contrainte sur la quantité de mémoire disponible
  - En pratique : ressources limitées, judicieux d'économiser.

```
VARIABLE p : pointeur vers entier
p ← ALLOUER entier
...
LIBERER p
```

- `LIBERER` libère la zone mémoire située à partir de l'adresse contenue dans le pointeur et de la taille correspondant au type de valeur référencée par le pointeur



IUT Valence
Université Grenoble Alpes

Rappels : enregistrements

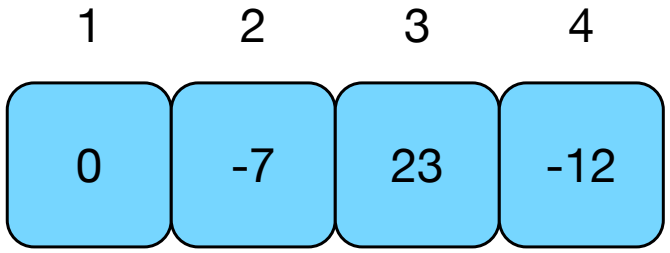
- Un **enregistrement** est **composé de plusieurs valeurs** appelées champs ou membres
  - Nombre de champs **fixe**, champs **nommés** et de **type quelconque**
  - Les opérations se limitent à la **lecture et l'affectation des champs**
  - Accès aux champs via la **notation pointée** (`variable.champs`)

```
ENREGISTREMENT Point

  CHAMPS x  : réel
  CHAMPS y  : réel

FIN ENREGISTREMENT
```

```
VARIABLE p : Point

  p.x ← 0.0
  p.y ← p.x + 1
```

IUT Valence Université Grenoble Alpes

Rappels : TDA : *Vecteur*



- Type de données linéaire et **dynamique**, collection de valeurs (de type `T`) stockées dans des cases contigües identifiées par un indice
  - Indices allant de 1 au **nombre d'éléments**
- Possibilité de **lire** et **écrire** l'élément à un indice donné
- Possibilité d'**insérer** et **retirer** un élément à un indice donné
- Possibilité d'**obtenir la taille** (nombre d'éléments)

IUT Valence Université Grenoble Alpes

Rappels : TDA : *Vecteur*

- **Nom** : `Vecteur` **(de T)**
- **Dépendances** : `Entier`, `T` (type des éléments)
- **Opérations** :
  - **Constructeurs** :
    - vecteur_vide : $\rightarrow$ `Vecteur`
  - **Transformateurs** :
    - ecrire : `Vecteur` x `Entier` x `T` $\rightarrow$ `Vecteur`
    - inserer : `Vecteur` x `Entier` x `T` $\rightarrow$ `Vecteur`
    - retirer : `Vecteur` x `Entier` $\rightarrow$ `Vecteur`
  - **Observateurs** :
    - `lire` : `Vecteur` x `Entier` $\rightarrow$ `T`
    - `taille` : `Vecteur` $\rightarrow$ `Entier`
- *A suivre . . .*

IUT Valence
Université Grenoble Alpes

Rappels : TDA : *Vecteur*

- **Pré-conditions**
  - `ecrire(v, n, e)` $\rightarrow 1 \leq$ `n` $\leq$ taille(v)
  - `inserer(v, n, e)` $\rightarrow 1 \leq$ `n` $\leq$ taille(v) $+ 1$
  - `retirer(v, n)` $\rightarrow 1 \leq$ `n` $\leq$ taille(v)
  - `lire(v, n)` $\rightarrow 1 \leq$ `n` $\leq$ taille(v)

IUT Valence Université Grenoble Alpes

Rappels : TDA : *Vecteur*

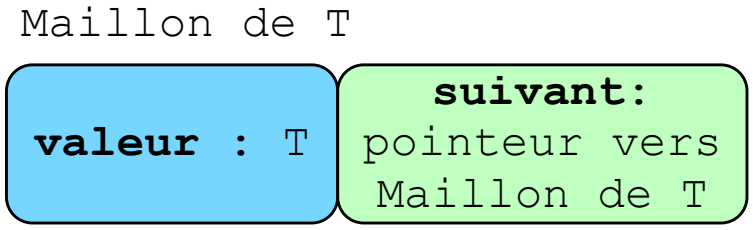
- **Axiomes**
  - `taille(vecteur_vide()) = 0`
  - `taille(ecrire(v, n, e)) = taille(v)`
  - `taille(inserer(v, n, e)) = taille(v) + 1`
  - `taille(retirer(v, n)) = taille(v) - 1`
  - `lire(ecrire(vecteur_vide(), 0, e), 0) = e`
  - `lire(ecrire(v, 0, e), n) = lire(v, n) si n` $\geq$ `1`
  - `retirer(inserer(v, n, e), n) = v`
  - `lire(inserer(v, n, e), n) = e`
  - `lire(inserer(v, n, e), p) = lire(v, p) si p` $<$ `n`
  - `lire(inserer(v, n, e), p) = lire(v, p-1) si p` $\geq$ `n`
  - `lire(retirer(v, n), p) = lire(v, p) si p` $<$ `n`
  - `lire(retirer(v, n), p) = lire(v, p+1) si p` $\geq$ `n`

IUT Valence Université Grenoble Alpes

Exercice

- On définit un enregistrement `Maillon de T` permettant de représenter un maillon de structure simplement chaînée composé :
  - d'une valeur (de type `T`)
  - d'un pointeur vers `Maillon de T`

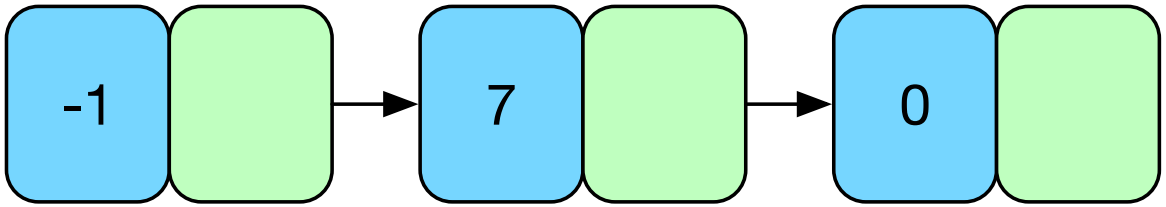
```
ENREGISTREMENT Maillon de T

  CHAMPS valeur : T
  CHAMPS suivant : pointeur vers Maillon de T

FIN ENREGISTREMENT
```



Exercice

- On définit un enregistrement `Vecteur de T` permettant de manipuler un `Vecteur` (dynamique) en s'appuyant sur une **structure chainée**

```
ENREGISTREMENT Vecteur de T

   CHAMPS taille : entier
   CHAMPS debut : pointeur vers Maillon de T

FIN ENREGISTREMENT
```



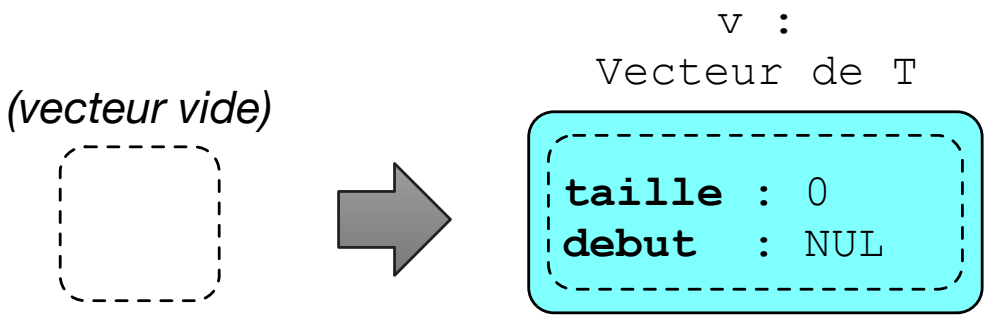
IUT Valence
Université Grenoble Alpes

Exercice

- On considère que :
  - La déclaration d'une variable `Maillon de T` ou `Vecteur de T` **n'initialise pas les enregistrements**
  - Les transformateurs de `Vecteur de T` sont des mutateurs
  - Les enregistrements sont passés par copie
- Schématiser et écrire les opérations :
  - `vecteur_vide`, `taille`,
  - `lire`, `ecrire`,
  - `retirer` et `inserer`

IUT Valence Université Grenoble Alpes

vecteur_vide

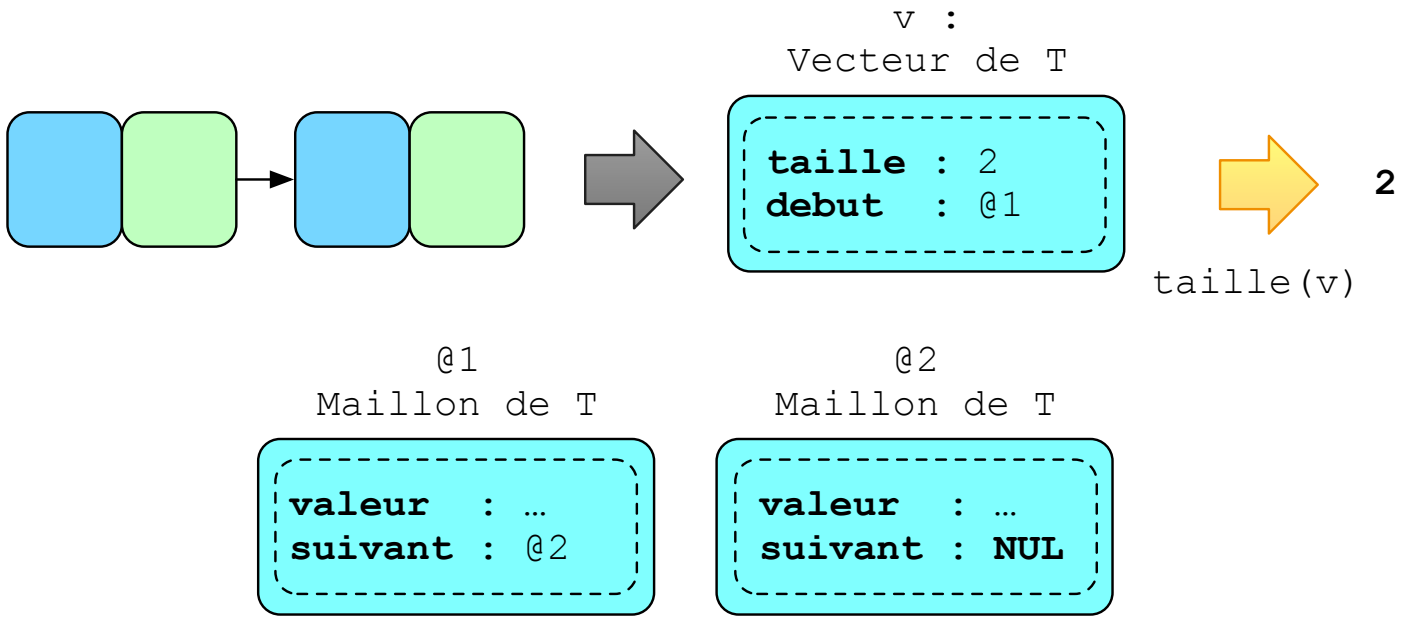



```
FONCTION vecteur_vide
         (p : pointeur vers Vecteur de T) : (aucun)

   (p↑).taille ← 0
   (p↑).debut ← NUL

FIN FONCTION
```

taille

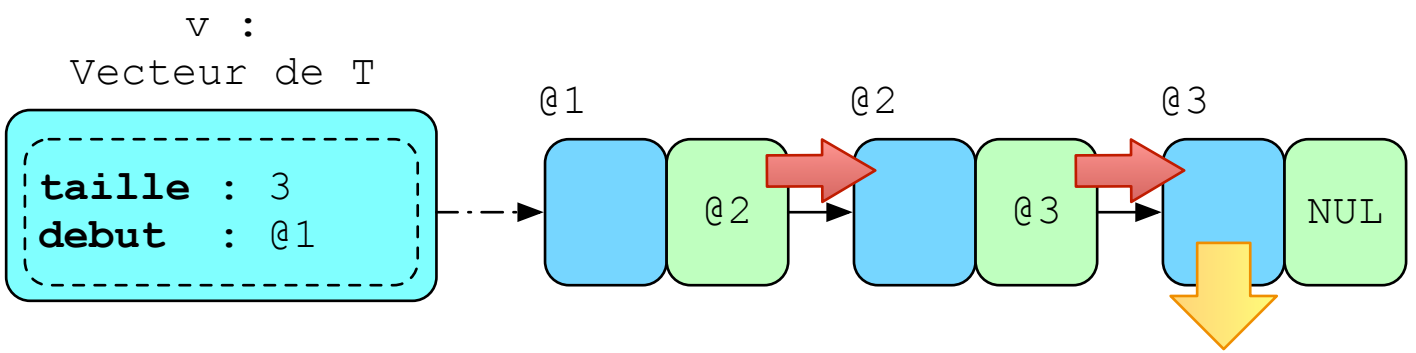
```
FONCTION taille (v : Vecteur de T) : entier

   RETOURNER v.taille

FIN FONCTION
```

- Cas d'un vecteur avec 2 éléments



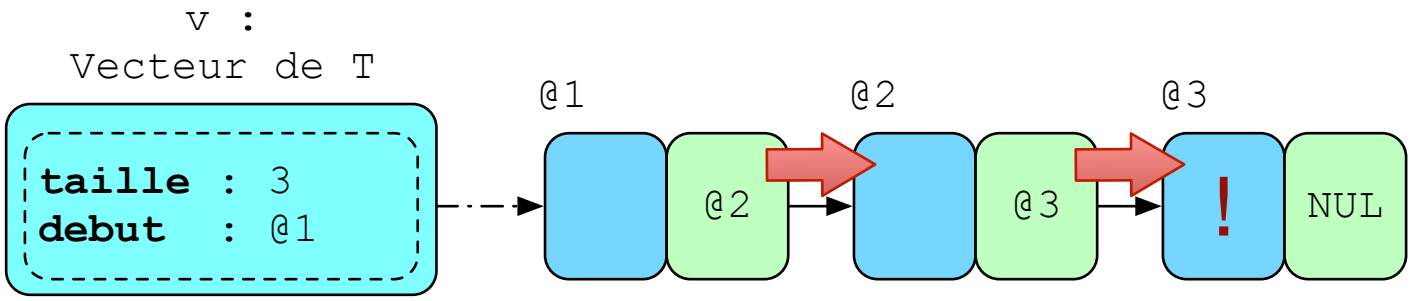
lire

```
FONCTION lire (v : Vecteur de T, index : entier) : T

   VARIABLE i : entier
   VARIABLE p : pointeur vers Maillon de T

   p ← v.debut
   POUR i DE 1 à index - 1 PAR PAS DE 1
      p ← (p↑).suivant
   FIN POUR
   RETOURNER (p↑).valeur

FIN FONCTION
```



ecrire

```
FONCTION ecrire (v : Vecteur de T, index : entier,
                 t : T) : (aucun)

  VARIABLE i : entier
  VARIABLE p : pointeur vers Maillon de T

  p ← v.debut
  POUR i DE 1 à index - 1 PAR PAS DE 1
    p ← (p↑).suivant
  FIN POUR
  (p↑).valeur ← t

FIN FONCTION
```



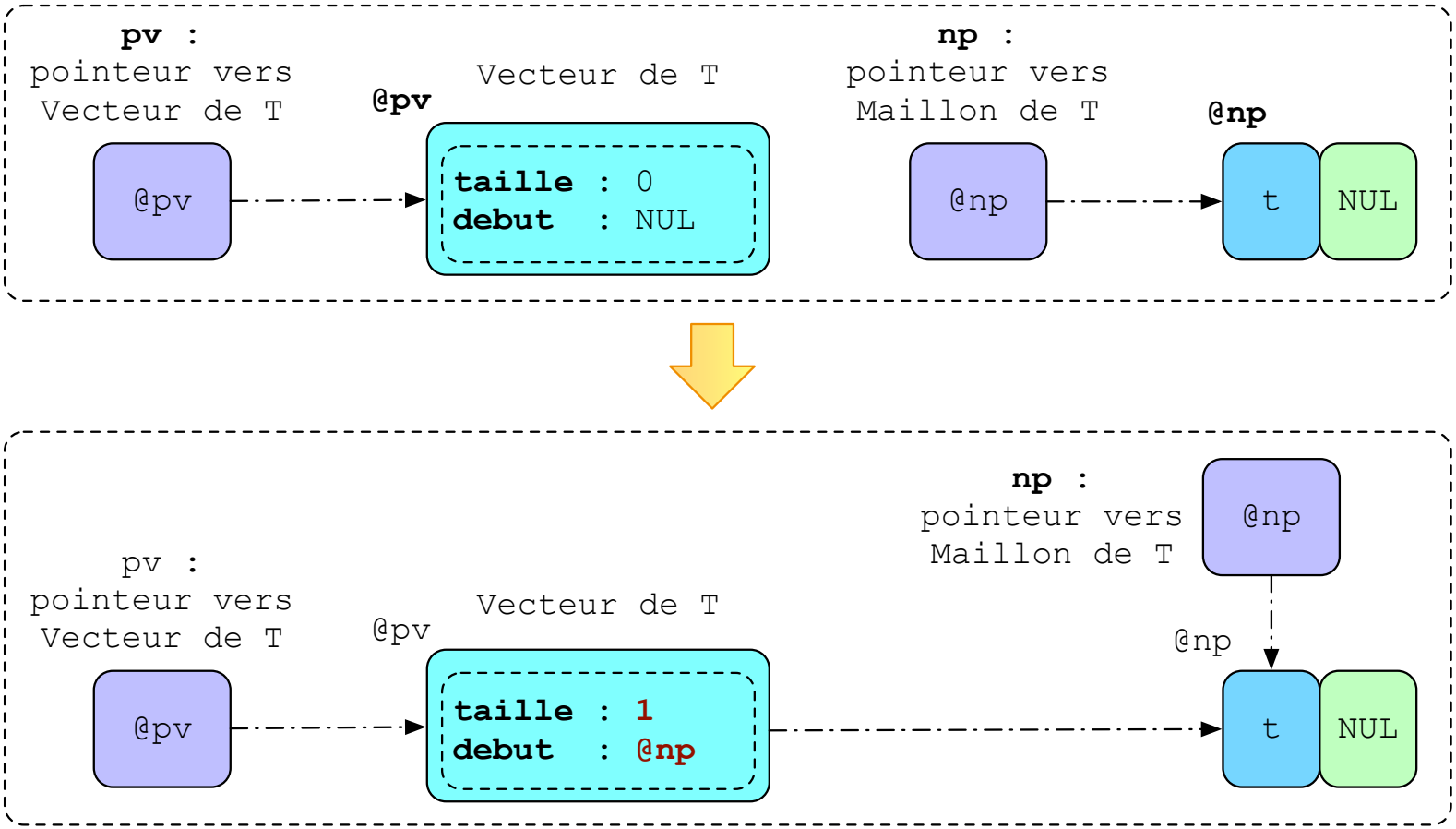
inserer

```
FONCTION inserer (pv : pointeur vers Vecteur de T,
                  index : entier, t : T) : (aucun)
  VARIABLE i : entier
  VARIABLE p : pointeur vers Maillon de T
  VARIABLE np : pointeur vers Maillon de T

  np ← ALLOUER Maillon de T
  (np↑).valeur ← t
  (np↑).suivant ← NUL
  SI (pv↑).taille = 0 ALORS // Cas 1 : vecteur vide
  SINON
    SI index = 1 ALORS
      // Cas 2 : insertion en tête, vecteur non vide
    SINON
      // Cas 3 :  insertion au milieu, vecteur non vide
    FIN SI
  FIN SI
  (pv↑).taille ← (pv↑).taille + 1
FIN FONCTION
```

IUT Valence Université Grenoble Alpes

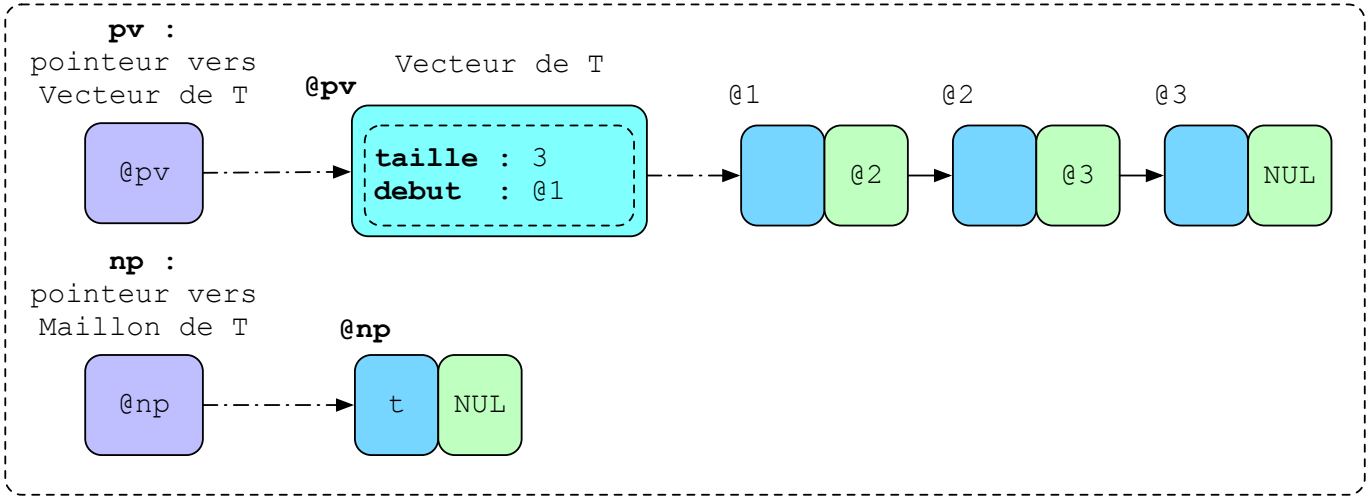
inserer : vecteur vide

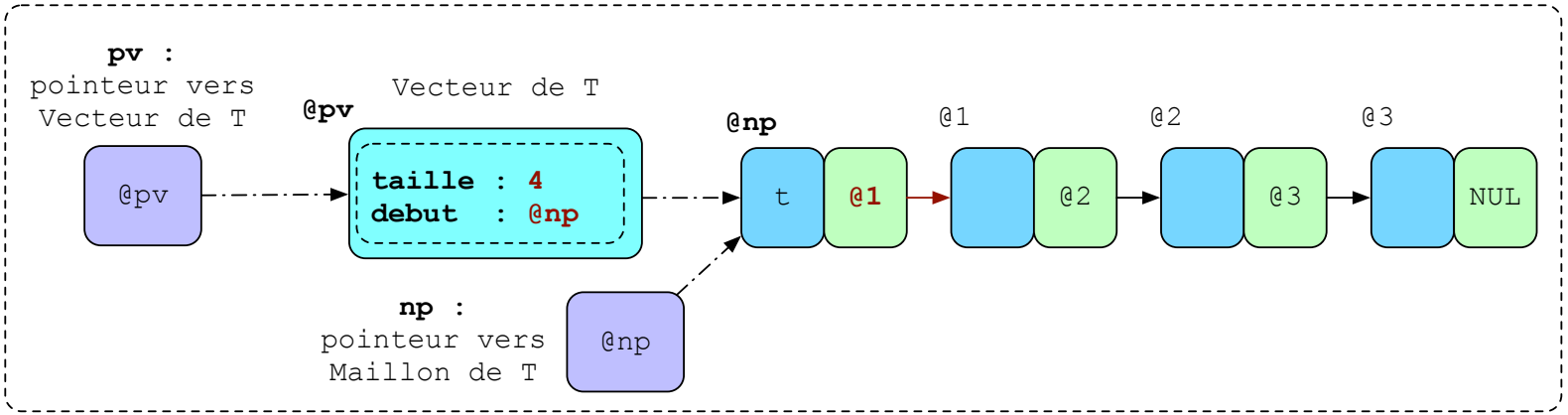
(pv↑).debut ← np



inserer : insertion en tête, vecteur non vide

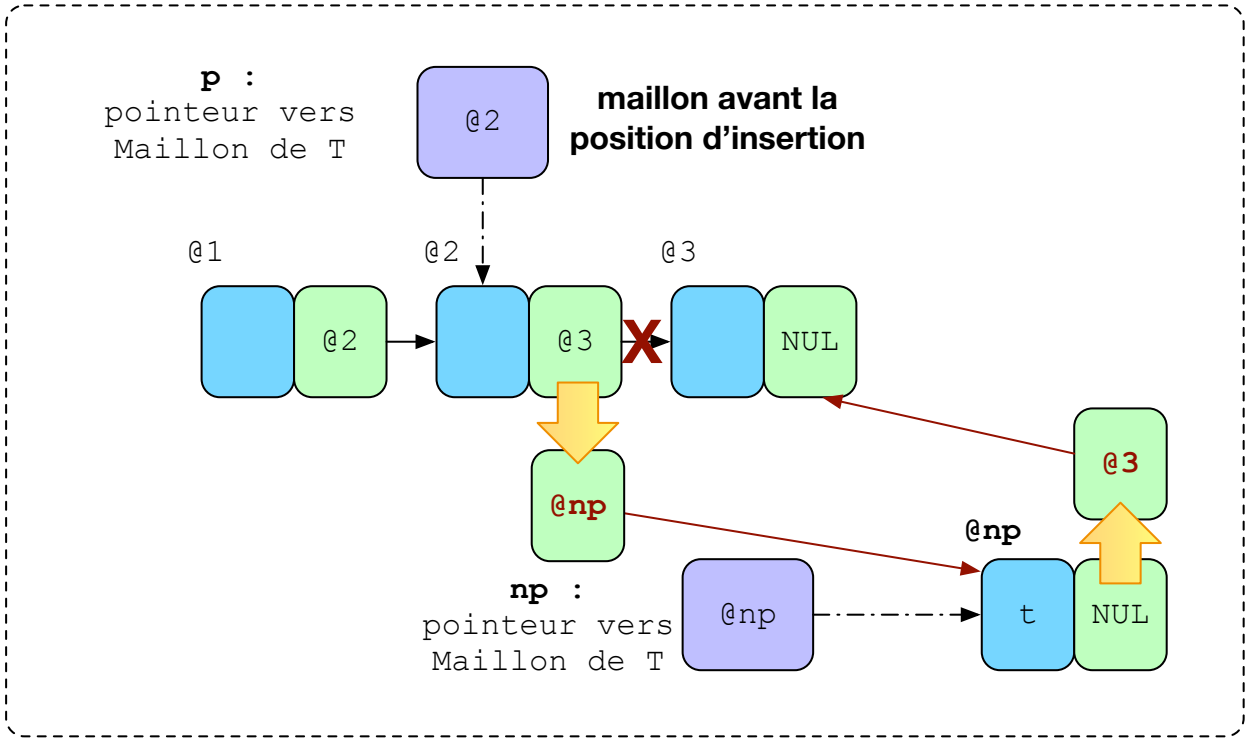
(np↑).suivant ← (pv↑).debut
(pv↑).debut ← np





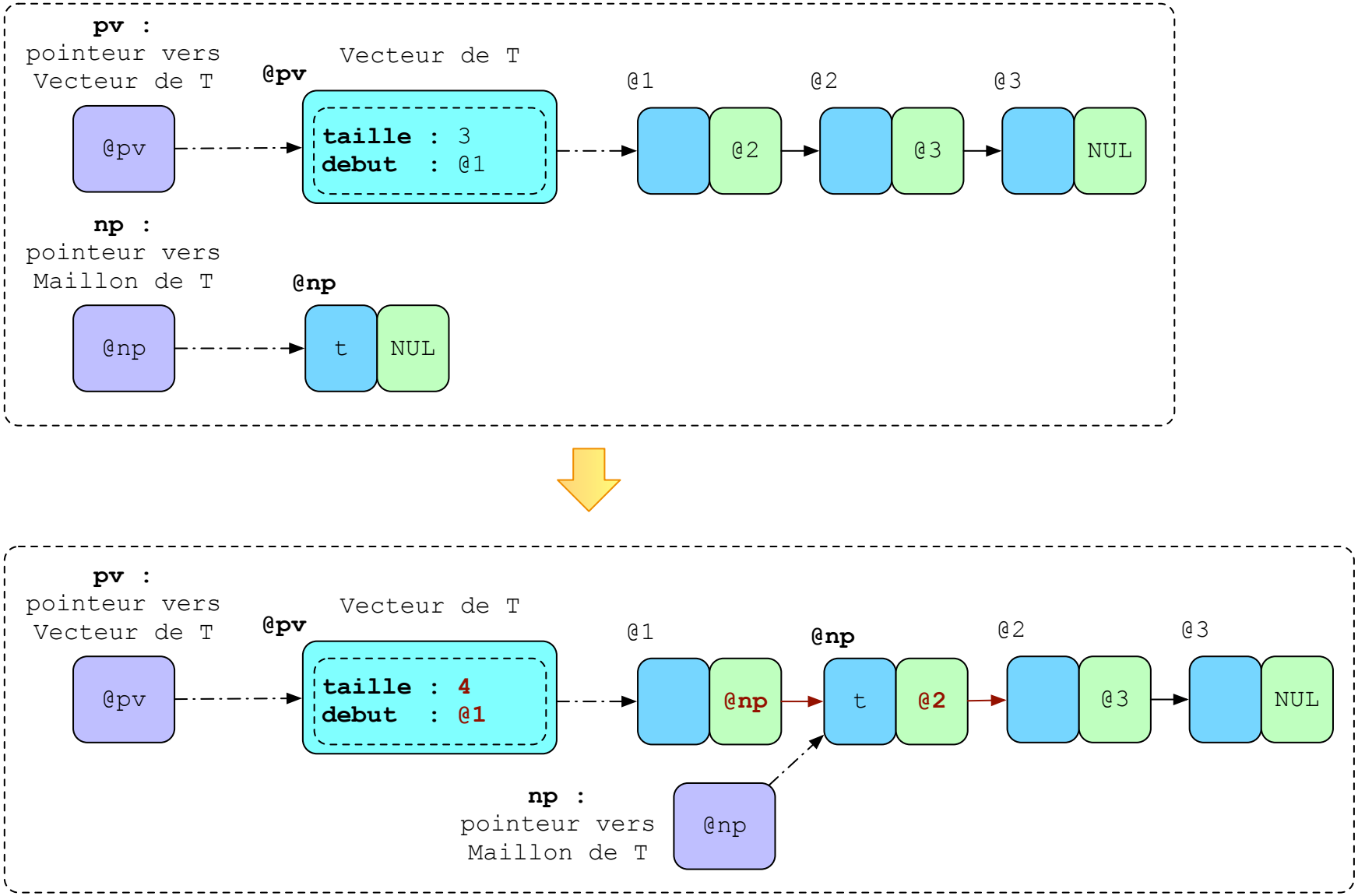
inserer : insertion au milieu, vecteur non vide

```
p ← (pv↑).debut
POUR i DE 1 à index - 2 PAR PAS DE 1
   p ← (p↑).suivant
FIN POUR
(np↑).suivant ← (p↑).suivant
(p↑).suivant ← np
```



inserer : cas du vecteur non vide

- Exemple avec insertion en position 2



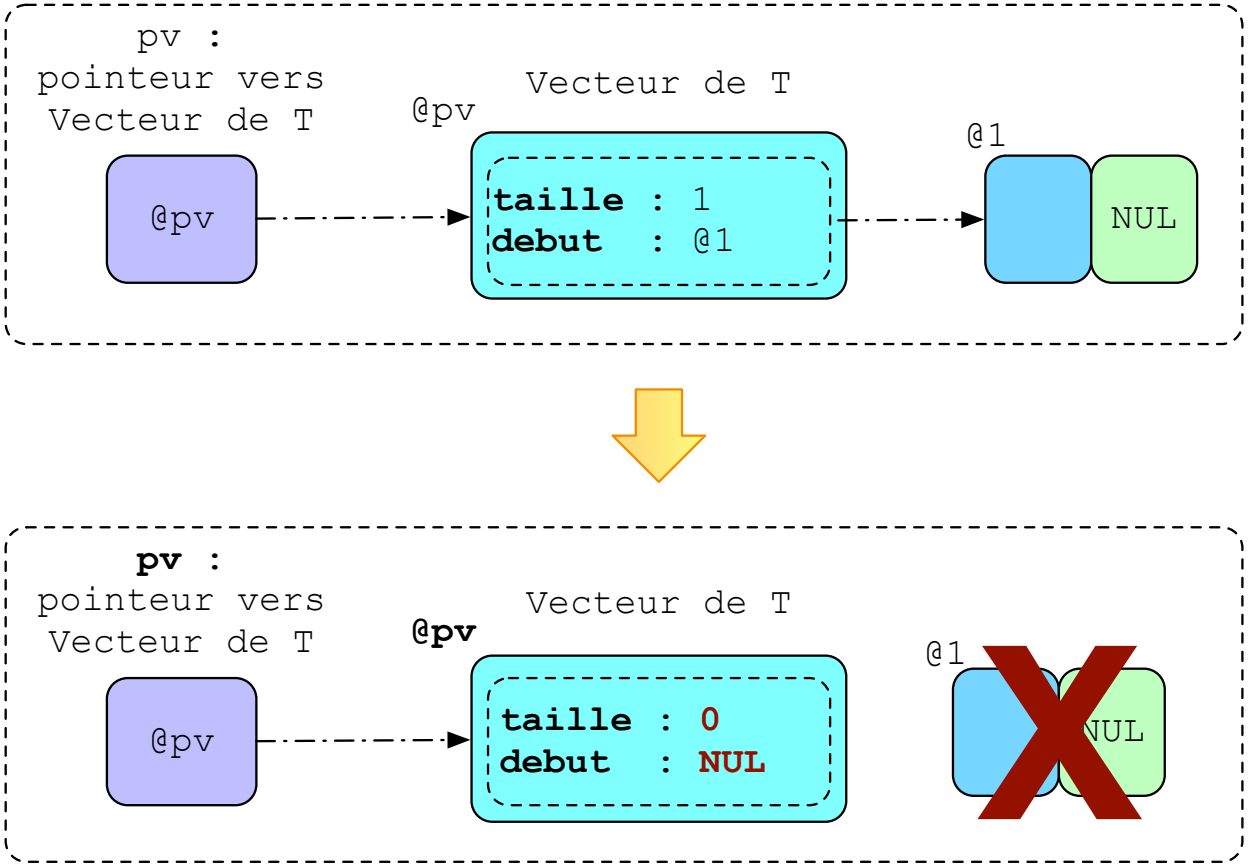
retirer

```
FONCTION retirer (pv : pointeur vers Vecteur de T,
                  index : entier, t : T) : (aucun)

  VARIABLE i : entier
  VARIABLE p : pointeur vers Maillon de T
  VARIABLE xp : pointeur vers Maillon de T

  SI (pv↑).taille = 1 ALORS
     // Cas du dernier (seul) élément
  SINON
    SI index = 1 ALORS
      // Cas du premier élément
    SINON
      // Autre cas...
    FIN SI
  FIN SI

  (pv↑).taille ← (pv↑).taille - 1
```

IUT Valence Université Grenoble Alpes

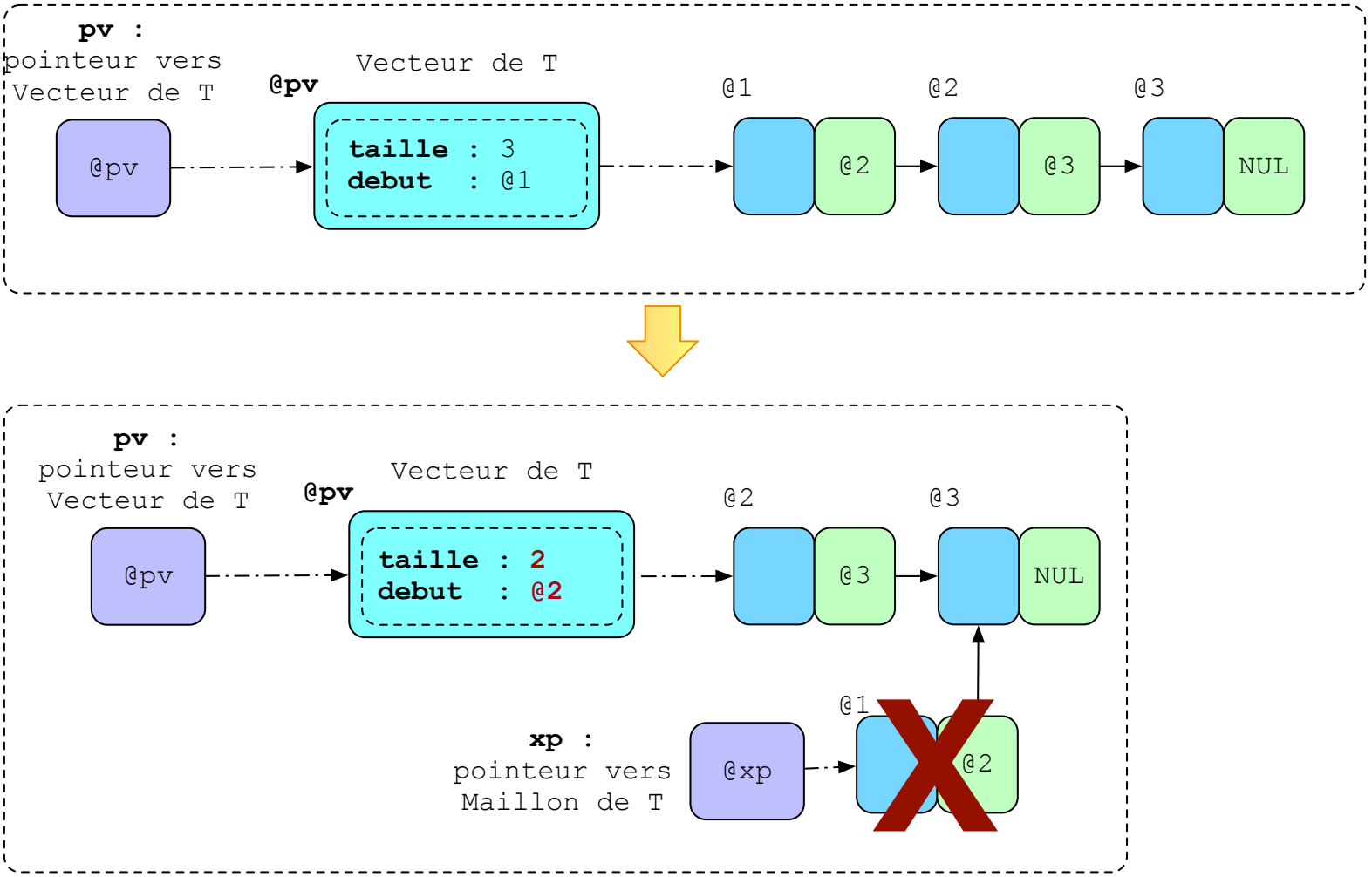
retirer : cas du seul élément

```
LIBERER (pv↑).debut
(pv↑).debut = NUL
```



retirer : cas du premier élément

```
xp ← (pv↑).debut
(pv↑).debut ← (xp↑).suivant
LIBERER (xp)
```



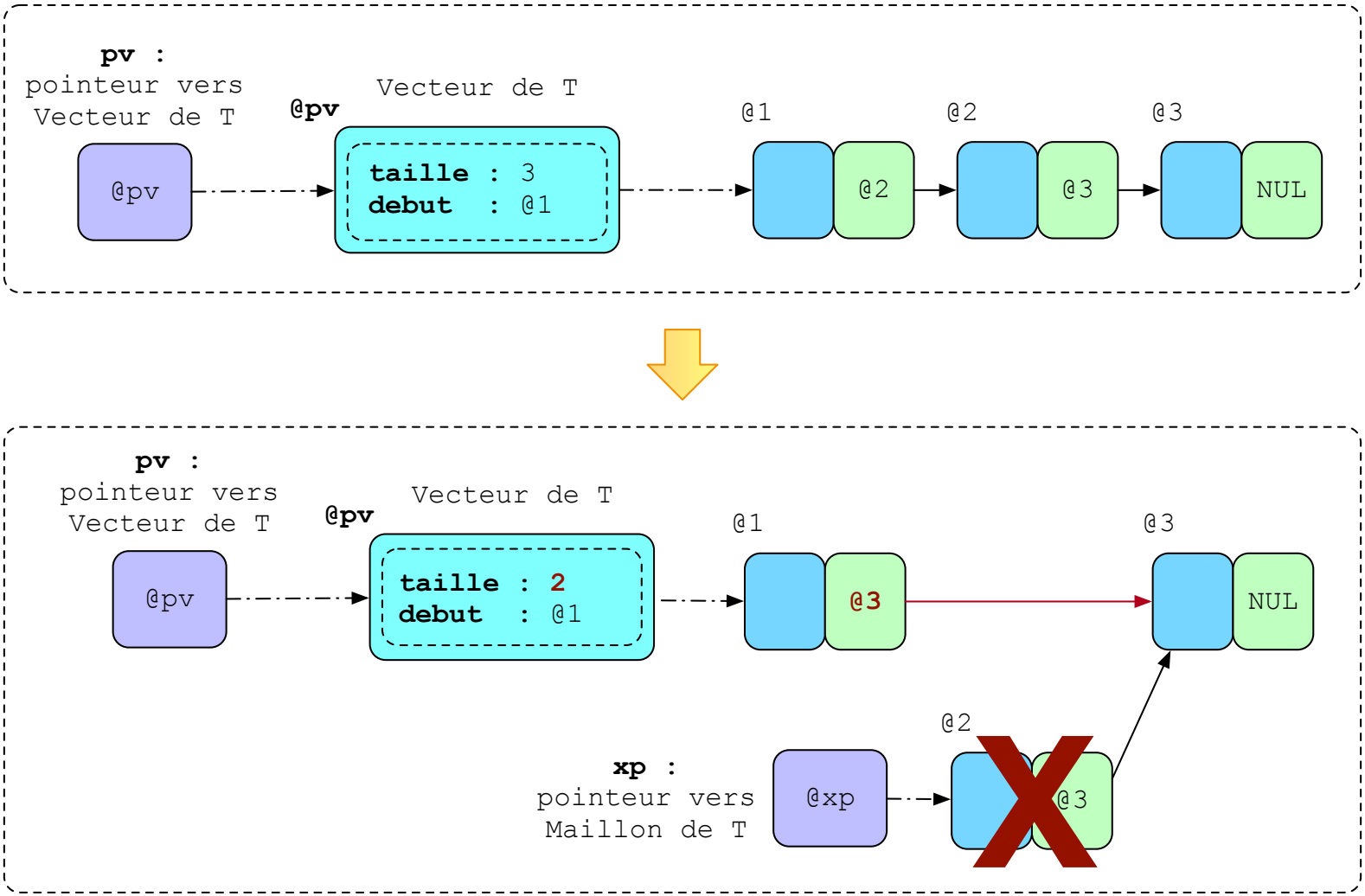
retirer : autre cas

```
p ← (pv↑).debut
POUR i DE 1 à index-2 PAR PAS DE 1
   p ← (p↑).suivant
FIN POUR
xp ← (p↑).suivant
(p↑).suivant ← (xp↑).suivant
LIBERER (xp)
```

IUT Valence
Université Grenoble Alpes

retirer : autre cas

- Exemple avec retrait en position 2



Fin !

IUT Valence
Université Grenoble Alpes