TDA : File

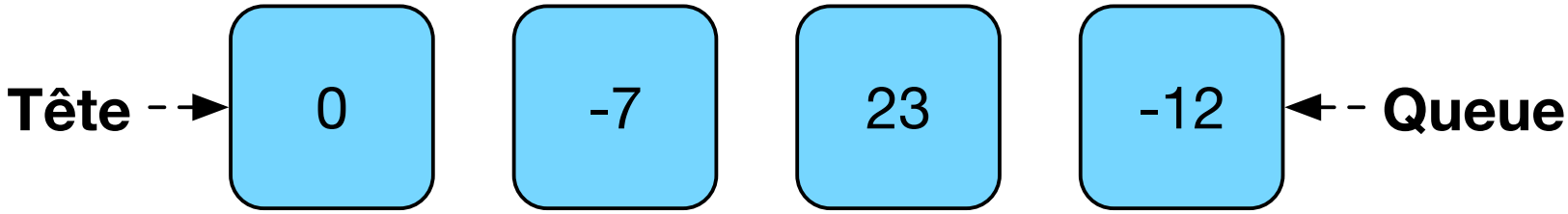
Sébastien Jean

IUT de Valence
Département Informatique

v1.0, 26 novembre 2025

IUT Valence
Université Grenoble Alpes

Type de données *File*



- Type de données linéaire et dynamique similaire à un vecteur mais accès FIFO (*First In First Out*)
- Possibilité d'**obtenir la taille**, de **savoir si la file est vide** , de **voir le prochain élément**, d'**ajouter (en queue) ou retirer (en tête) un élément**

IUT Valence Université Grenoble Alpes

TDA *File*

- **Nom** : `File` **(de T)**
- **Dépendances** : `Booléen`, `Entier`, `T` (type des éléments)
- **Opérations** :
  - **Constructeurs** :
    - `file_vide` : $\rightarrow$ `File`
  - **Transformateurs** :
    - `ajouter` : `File` $\times$ `T` $\rightarrow$ `File`
    - `retirer` : `File` $\rightarrow$ `File`
  - **Observateurs** :
    - `taille` : `File` $\rightarrow$ `Entier`
    - `est_vide` : `File` $\rightarrow$ `Booléen`
    - `voir_prochain` : `File` $\rightarrow$ `T`
- *A suivre . . .*

IUT Valence
Université Grenoble Alpes

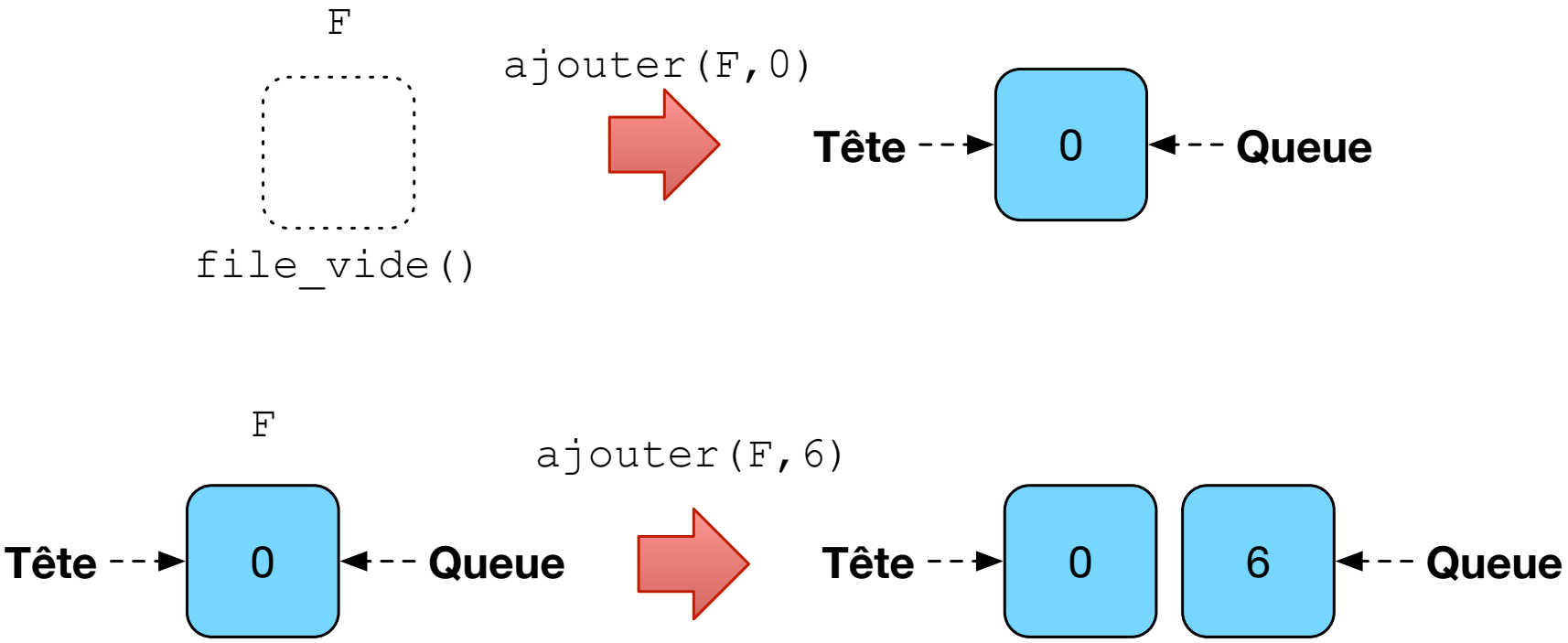
TDA *File* (suite)

- **Pré-conditions**
  - `retirer(f)` $\rightarrow$ `est_vide(f) = FAUX`
  - `voir_prochain(f)` $\rightarrow$ `est_vide(f) = FAUX`

IUT Valence
Université Grenoble Alpes

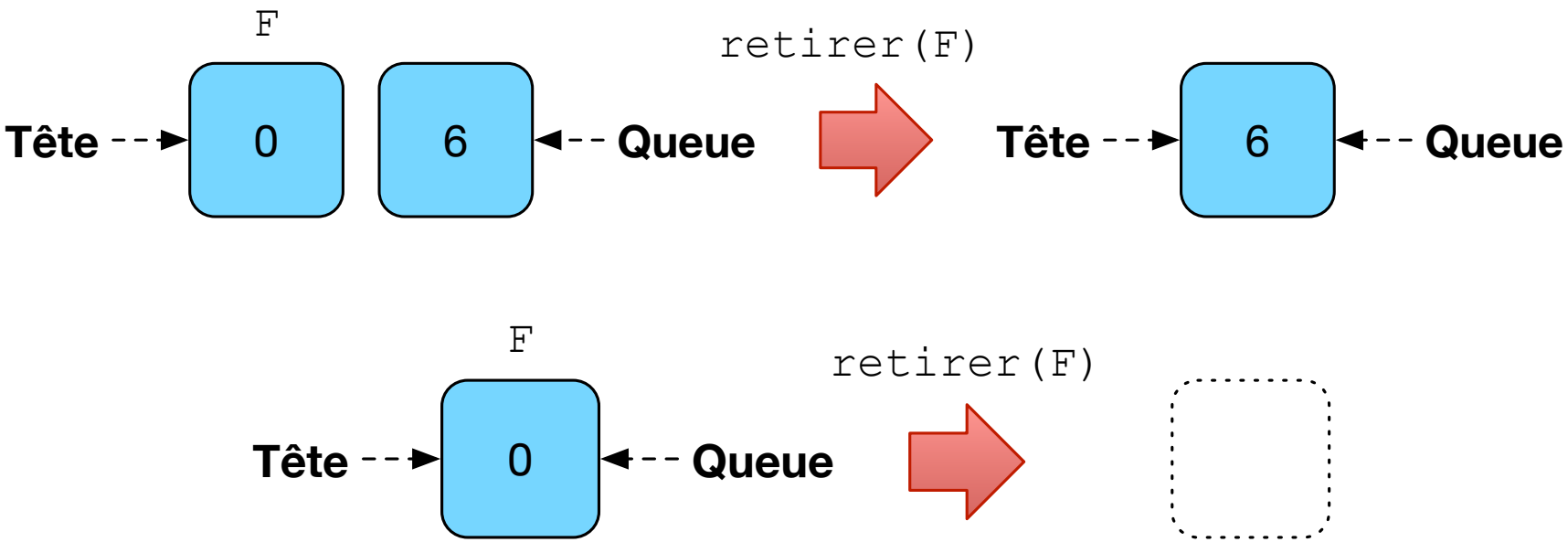
Exemples d ?opérations/situation

- Ajouter un élément



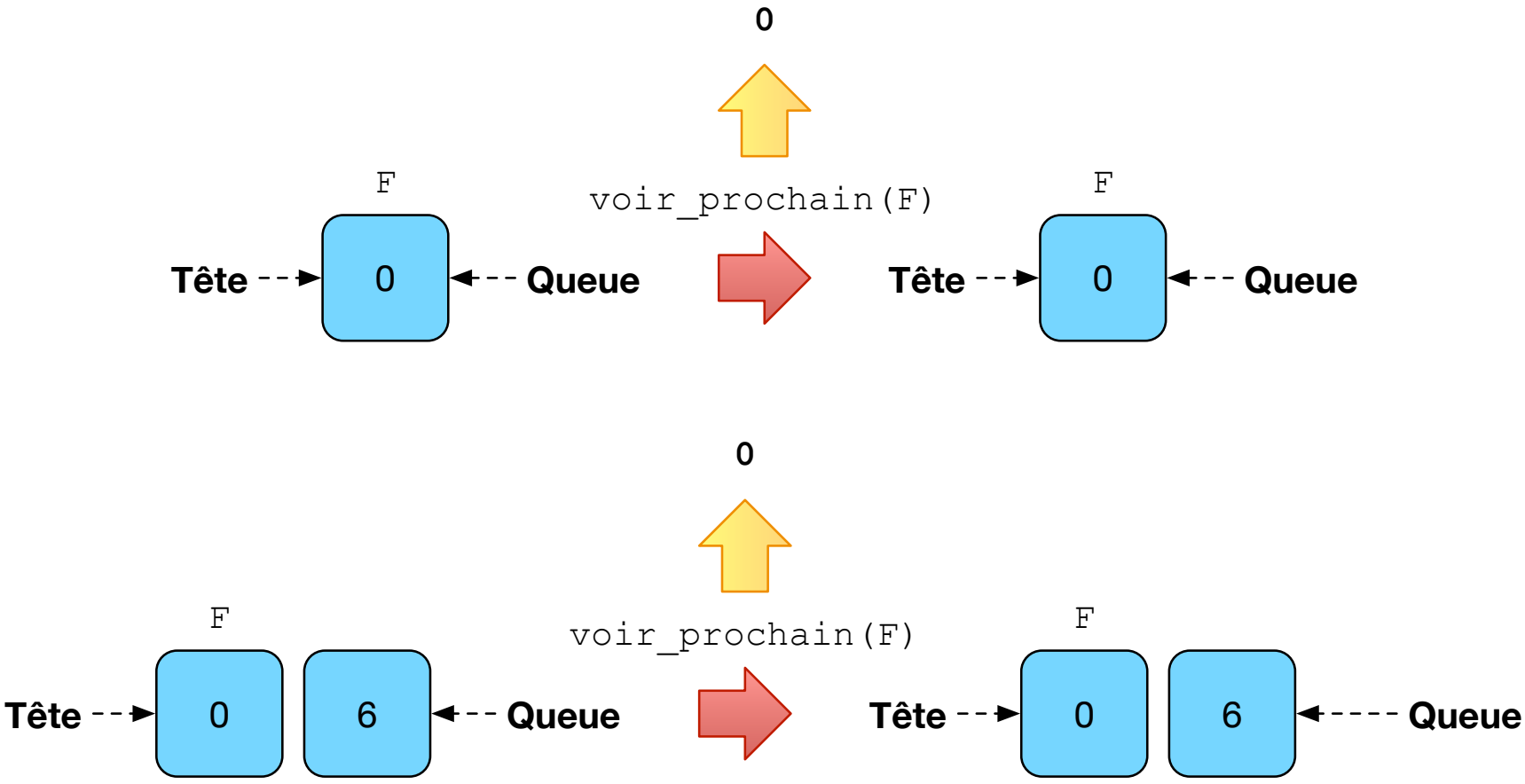
Exemples d ?opérations/situation

- Retirer un élément



Exemples d ?opérations/situation

- Voir le prochain



TDA *File* (suite)

- **Axiomes**
  - `est_vide(file_vide()) = VRAI`
  - `est_vide(ajouter(f, e)) = FAUX`
  - `taille(file_vide()) = 0`
  - `taille(ajouter(f, e)) = taille(f) + 1`
  - `taille(retirer(f)) = taille(f) - 1`
  - `retirer(ajouter(file_vide(), e)) = file_vide()`
  - `voir_prochain(ajouter(f, e)) = e` si `taille(f)` $= 0$
  - `voir_prochain(ajouter(f, e)) = voir_prochain(f)` si `taille(f)` > 0

IUT Valence
Université Grenoble Alpes

File et pseudo code

- Dans notre *pseudo code*, on suppose que
  - Le constructeur est remplacé par la **déclaration de la variable**
  - Les transformateurs sont des **mutateurs**
  - Les paramètres sont **passés par copie**

```
VARIABLE f : File d'entiers
VARIABLE i : entier

ajouter(f, 3)
ajouter(f, 7)

i ← voir_prochain(f)
retirer(f)

AFFICHER(voir_prochain(f))
```

IUT Valence
Université Grenoble Alpes

Exercice

Enoncé du problème

On souhaite inverser les éléments d'une file.

Spécification du problème

- **Donnée d'entrée** : f, file de `T` (la file à inverser)
- **Donnée de sortie** : r, file de `T` (la file inversée)
- **Pré-condition** : (aucune)
- **Post-condition** : `f` et `r` contiennent les mêmes éléments mais dans l'orde inverse

Signature de la fonction

- **inverse_file** (f : file de T) : file de T

IUT Valence Université Grenoble Alpes

Exercice

```
FONCTION inverse_file(f : file de T) : file de T

 VARIABLE compteur  : entier
 VARIABLE p : pile de T
 VARIABLE r : file de T

 POUR compteur de 1 A taille(f) PAR PAS DE 1
  empiler(p, voir_prochain(f))
  retirer(f)
 FIN POUR

  POUR compteur de 1 A taille(f) PAR PAS DE 1
  ajouter(r, voir_sommet(p))
  dépiler(p)
 FIN POUR

 RETOURNER r

FIN FONCTION
```

IUT Valence
Université Grenoble Alpes

Exercice

```
FONCTION inv_file_rec(f : file de T,
                      p : pile de T ) : file de T

 VARIABLE p : pile de T

 SI est_vide(f) ALORS
   POUR compteur de 1 A taille(p) PAR PAS DE 1
    ajouter(f, voir_sommet(p))
    dépiler(p)
  FIN POUR
 FIN SI

 empiler(p, voir_prochain(f)))
 retirer(f)

 RETOURNER inv_file_rec(f, p)

FIN FONCTION
```

IUT Valence
Université Grenoble Alpes

Exercice

```
FONCTION inverse_file(f : file de T) : file de T

 VARIABLE p : pile de T

 RETOURNER inv_file_rec(f, p)

FIN FONCTION
```

Fin !

IUT Valence
Université Grenoble Alpes