IFT339
Structures de données

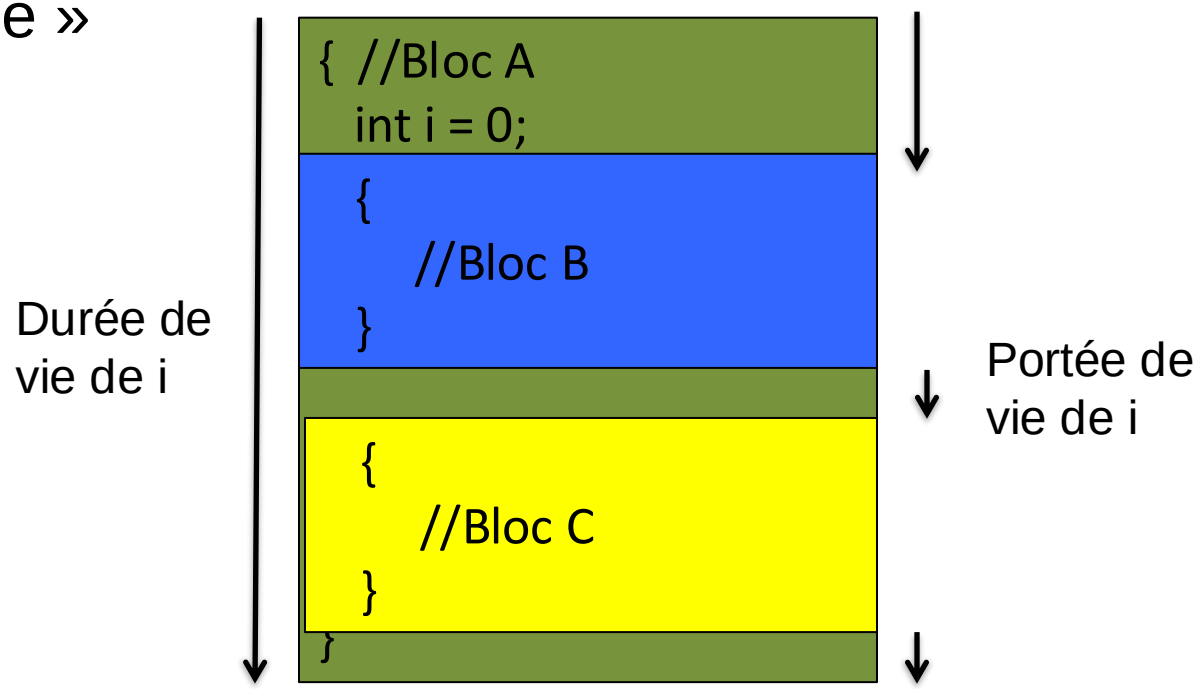
Thème 4 : Allocation automatique versus dynamique

Aïda Ouangraoua
Département d'informatique

UNIVERSITÉ DE
SHERBROOKE

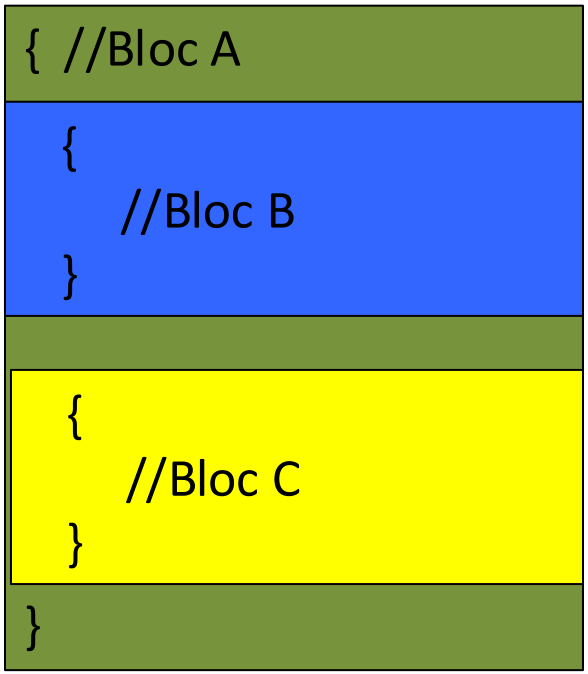
Rappel sur la gestion de la mémoire

- Programmation structurée en bloc
- **Bloc** : partie du code délimitée par deux accolades { }
- **Durée de vie d'un objet** : de sa déclaration à l'ouverture du bloc (allocation de mémoire) à la fermeture du bloc dans lequel il est déclaré (libération)
- **Portée d'un objet** : ensemble des parties du code où l'objet est accessible
- Possibilité de donner un nom à une portée : mot-clé « namespace »

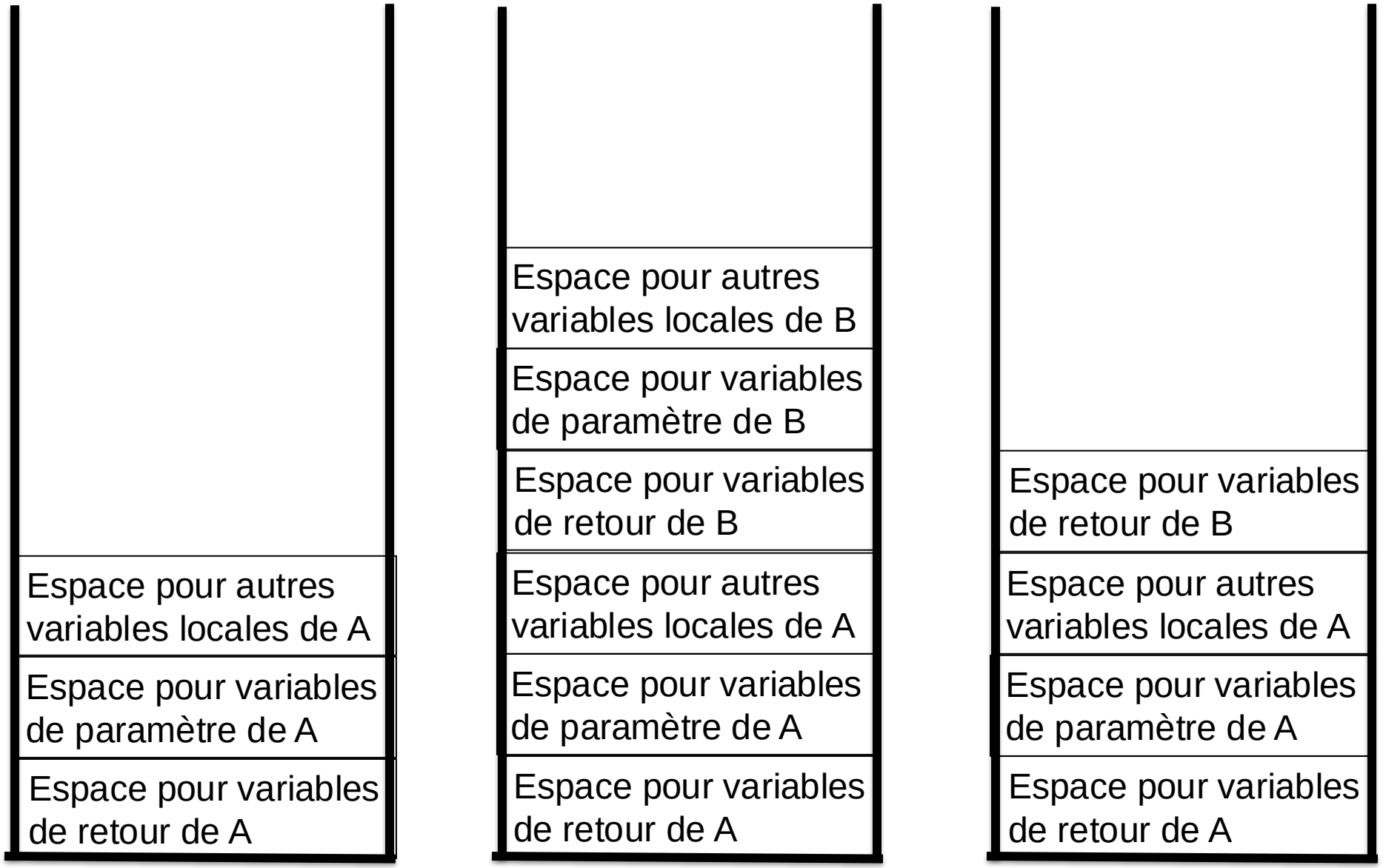


Modèle de la pile d'exécution

- Modèle fictif pour comprendre l'évolution de l'utilisation de la mémoire
- Accolade ouvrante { : allocation de l'espace mémoire requis pour les variables locales déclarées dans le bloc (sauf pour les paramètres passés par référence)
- Accolade fermante } : libération de l'espace mémoire alloué à l'ouverture (sauf pour les variables locales retournées par référence)



Modèle de la pile d'exécution

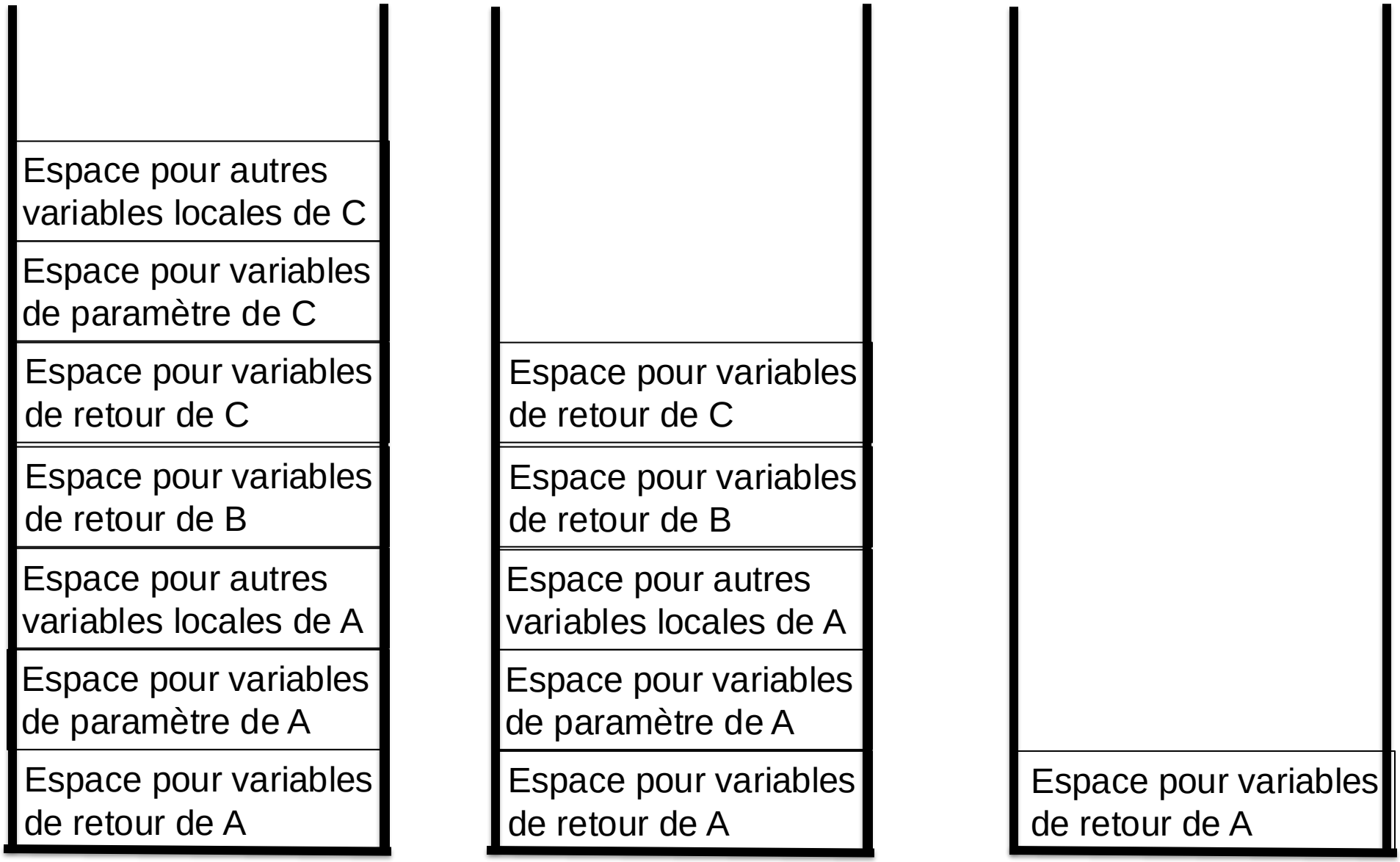


Ouverture de A : Ajout du cadre de pile de A

Ouverture de B : Ajout du cadre de pile de B

Fermeture de B :Suppression du cadre de pile de B (sauf retour)

Modèle de la pile d'exécution



Ouverture de C : Ajout du cadre de pile de C

Fermeture de C :Suppression du cadre de pile de C (sauf retour)

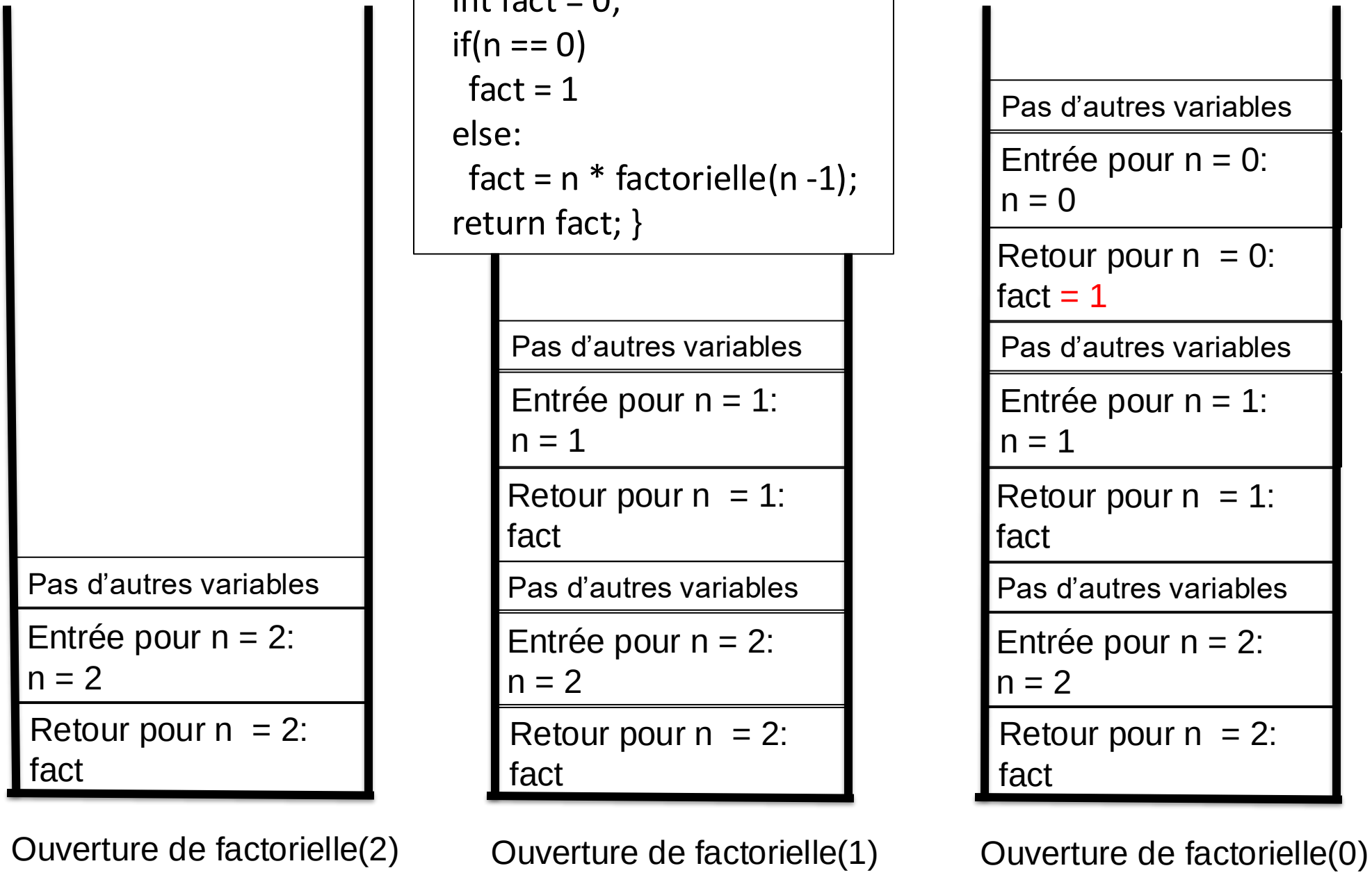
Fermeture de C :Suppression du cadre de pile de C (sauf retour)

- Modèle particulièrement adapté à la gestion de la récursivité

```
Int factorielle(int n) {
   int fact = 0;
   if(n == 0)
     fact = 1
   else:
     fact = n * factorielle(n -1);
   return fact; }
```

❑ Modèle particulièrement adapté à la gestion de la récursivité

```
Int factorielle(int n) {
  int fact = 0;
  if(n == 0)
    fact = 1
  else:
    fact = n * factorielle(n -1);
  return fact; }
```



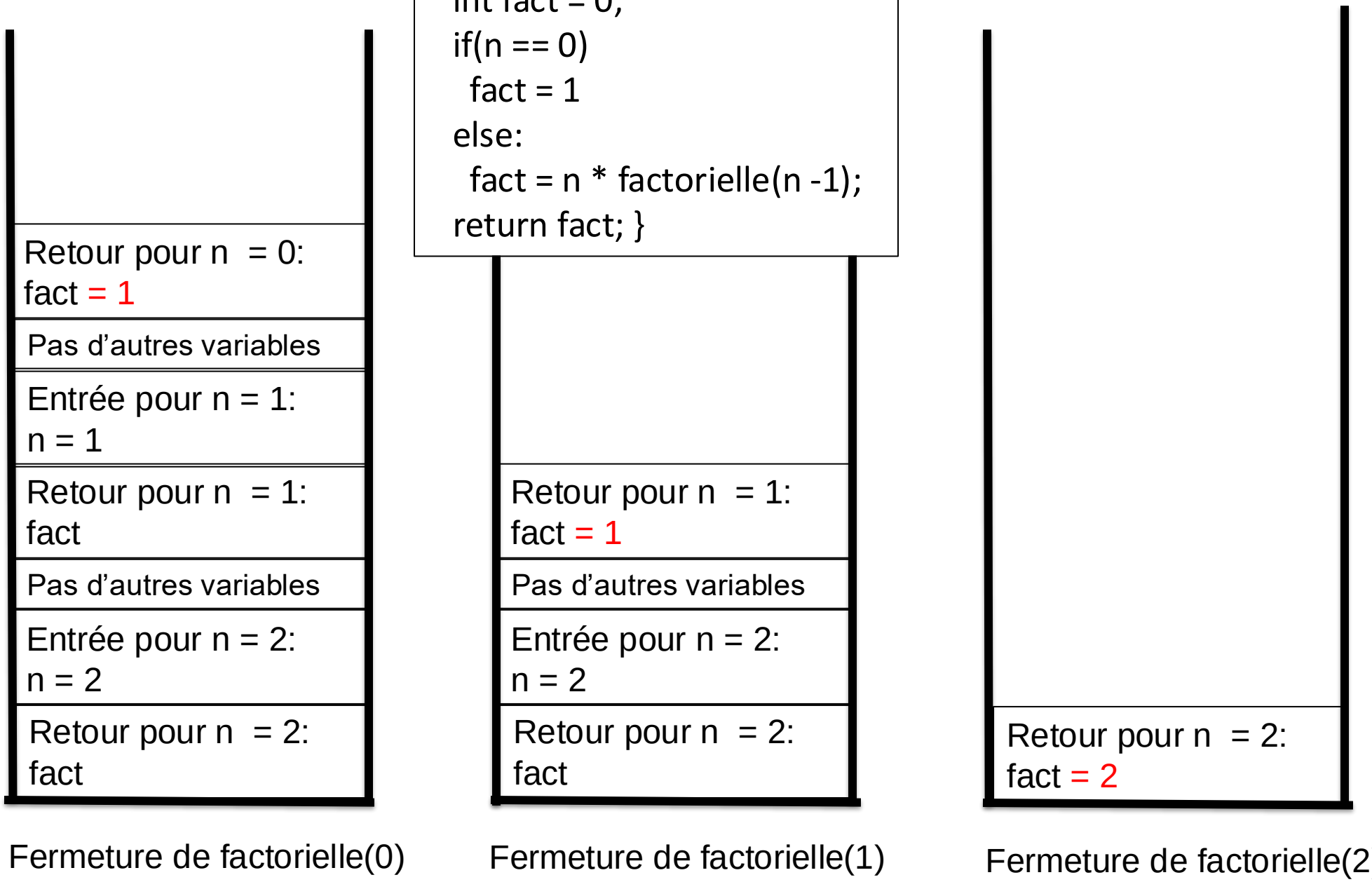
Ouverture de factorielle(2)

Ouverture de factorielle(1)

Ouverture de factorielle(0)

❑ Modèle particulièrement adapté à la gestion de la récursivité

```
Int factorielle(int n) {
  int fact = 0;
  if(n == 0)
    fact = 1
  else:
    fact = n * factorielle(n -1);
  return fact; }
```



Fermeture de factorielle(0)

Fermeture de factorielle(1)

Fermeture de factorielle(2)

Type de passage de paramètre et retour
(voir Thème 3 pour la syntaxe en C++)

- **Paramètre par référence** : Aucun espace n'est alloué pour la variable d'entrée V à l'ouverture du bloc. La variable V reste accessible dans le bloc.

- **Paramètre par valeur** : un nouvel espace pour la variable d'entrée V est alloué à l'ouverture du bloc. La valeur de V est copiée dans cet espace. V n'est pas accessible dans le bloc, mais sa copie est accessible.

Type de passage de paramètre et retour
(voir Thème 3 pour la syntaxe en C++)

- **Paramètre par référence** : Aucun espace n'est alloué pour la variable d'entrée V à l'ouverture du bloc. La variable V reste accessible dans le bloc.

- **Paramètre par valeur** : un nouvel espace pour la variable d'entrée V est alloué à l'ouverture du bloc. La valeur de V est copiée dans cet espace. V n'est pas accessible dans le bloc, mais sa copie est accessible.

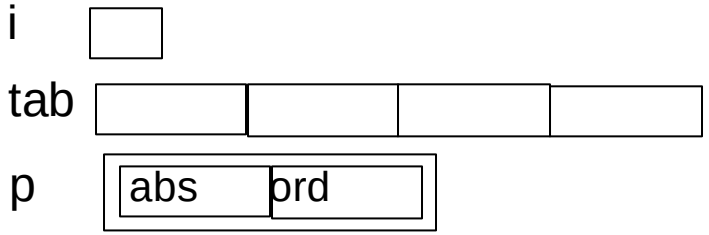
- **Retour par référence** : l'espace pour la variable de retour V n'est pas supprimé à la fermeture du bloc. V reste accessible après la sortie du bloc .

- **Retour par valeur** : À la ferméture du bloc, un nouvel espace pour la variable de retour V est alloué. La valeur de V est copiée dans cet espace, puis V est supprimé. V n'est pas accessible à la sortie du bloc, mais sa copie est accessible.

❑ Allocation/libération automatique :

- Gérées automatiquement et implicitement par le compilateur à l'ouverture et à la fermeture d'un bloc
- Crée et retourne des objets sans les initialiser au début du bloc

```
int i;
float tab[4];
Point p;
```

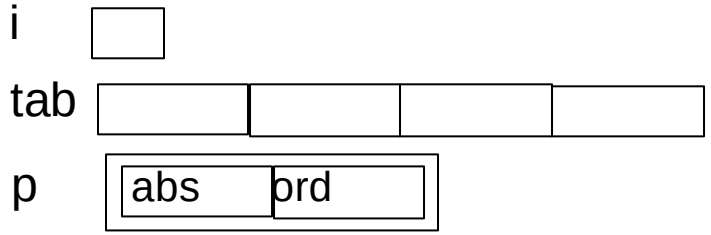


- Supprime les objets sans les nettoyer à la fin (libère uniquement la mémoire allouée automatiquement)

❑ Allocation/libération automatique :

- Gérées automatiquement et implicitement par le compilateur à l'ouverture et à la fermeture d'un bloc
- Crée et retourne des objets sans les initialiser au début du bloc

```
int i;
float tab[4];
Point p;
```

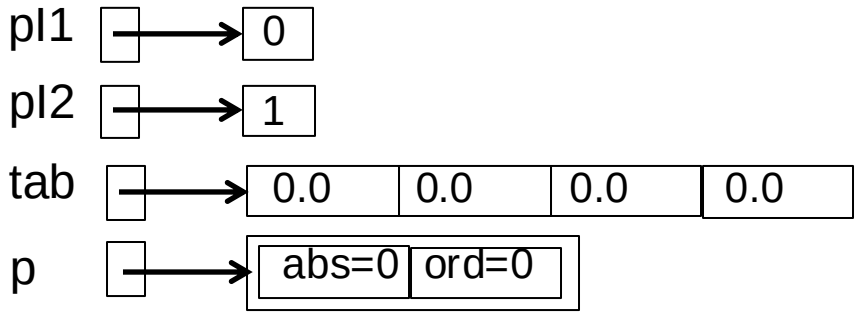


- Supprime les objets sans les nettoyer à la fin (libère uniquement la mémoire allouée automatiquement)

❑ Allocation/libération dynamique :

- Gérées explicitement par l'usager dans un bloc
- Crée des objets et retourne des pointeurs vers les objets
  - Utilisation de l'opérateur « new » : appel à un constructeur

```
int *pI1 = new int;
int *pI2 = new int(1);
float *tab = new float[4];
Point *p = new
Point(0,0);
```

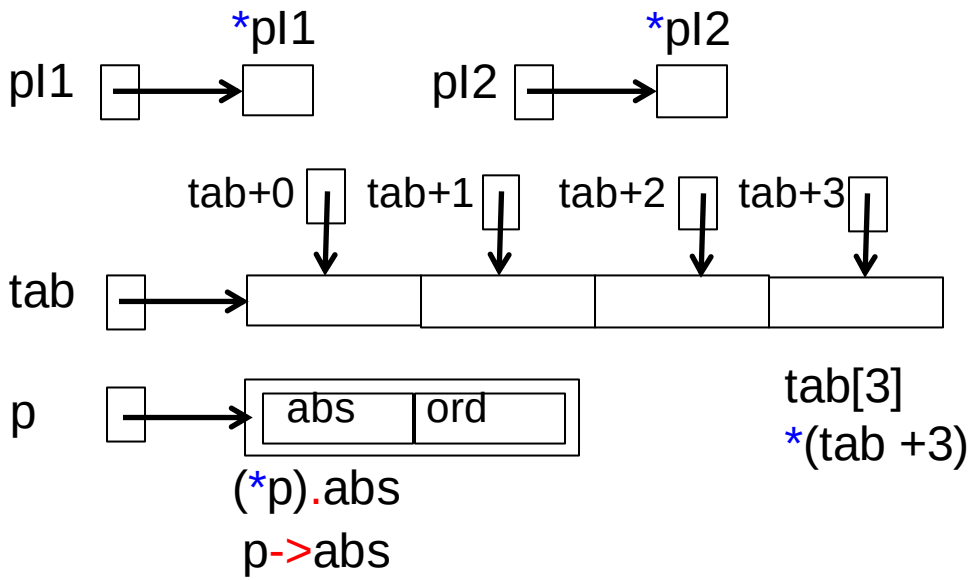


- Les objets créés doivent être supprimés par l'usager

❑ Allocation dynamique

- Accès à un objet : utilisation de l'opérateur de déréférencement (*) appliqué au pointeur

```
int *pI1 = new int;
int *pI2 = new int(1);
float *tab = new float[4];
Point *p = new Point(0,0);
```



- Pour accéder à la valeur des attributs d'un objet : utilisation des opérateurs « . » ou « -> »

- Libération d'un espace alloué dynamiquement : utilisation de l'opérateur « delete » : appel au destructeur

```
delete pI1;
delete pI2;
delete [] tab;
delete p;
```

- **Exemple : Allocation automatique versus dynamique d'une matrice d'entiers 10 x 20**

```
//allocation automatique
int M1[10][20];

//allocation dynamique
int **M2 = new int*[10];
for(int i = 0;i< 10;i++)
   M2[i] = new int[20];

//desallocation dynamique
for(int i = 0;i< 10;i++)
   delete [] M2[i];
 delete [] M2;
```