Plan

I. Gestion des procédures par pile

II. Flot de contrôle

1. Flot de contrôle avec pile logique
2. Flot de contrôle avec pile physique

III. Activation de procédure

1. Arbres d'activation
2. Exemple: QuickSort
3. Arbres d'activation de l'exemple QuickSort

IV. Allocation de l'espace mémoire

1. Allocation statique
2. Allocation par pile
3. Allocation en tas

Introduction

- Cette séance se concentre sur le lien entre le programme source et les actions qui se produiront à l'exécution du programme, hormis le code et sa génération.
- L'allocation et la désallocation des données sont gérées par le module d'infrastructure d'exécution.
- À chaque fois qu'une procédure (ou fonction) est exécutée, on parle d'une activation de la procédure.
- Dans le cas de procédures récursives, plusieurs activations de la même procédure peuvent coexister.

Rappel

- Une procédure est une déclaration qui associe un identificateur à un énoncé.

L'identificateur est le nom de la procédure. L'énoncé est le corps de la procédure.

- Suivant le vocabulaire de certains langages, une procédure qui retourne une valeur est une fonction.

- On dispose d'une notation (comme "f(a, b, c)") servant à indiquer que la procédure est appelée.

- Certains identificateurs inclus dans la définition de procédure ont un rôle spécial et sont appelés paramètres formels.

- Des arguments, ou paramètres actuels, peuvent être passés à la procédure lors de l'appel. Ils "remplacent" les paramètres formels de la procédure au cours de cet appel.

Plan

I. Gestion des procédures par pile

II. Flot de contrôle

1. Flot de contrôle avec pile logique
2. Flot de contrôle avec pile physique

III. Activation de procédure

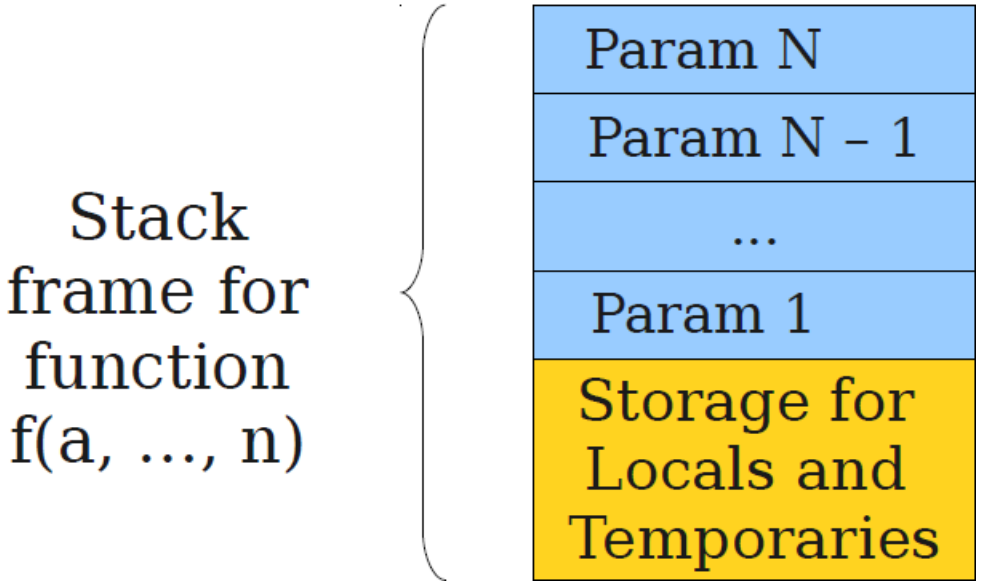
1. Arbres d’activation
2. Exemple: QuickSort
3. Arbres d’activation de l’exemple QuickSort

IV. Allocation de l’espace mémoire

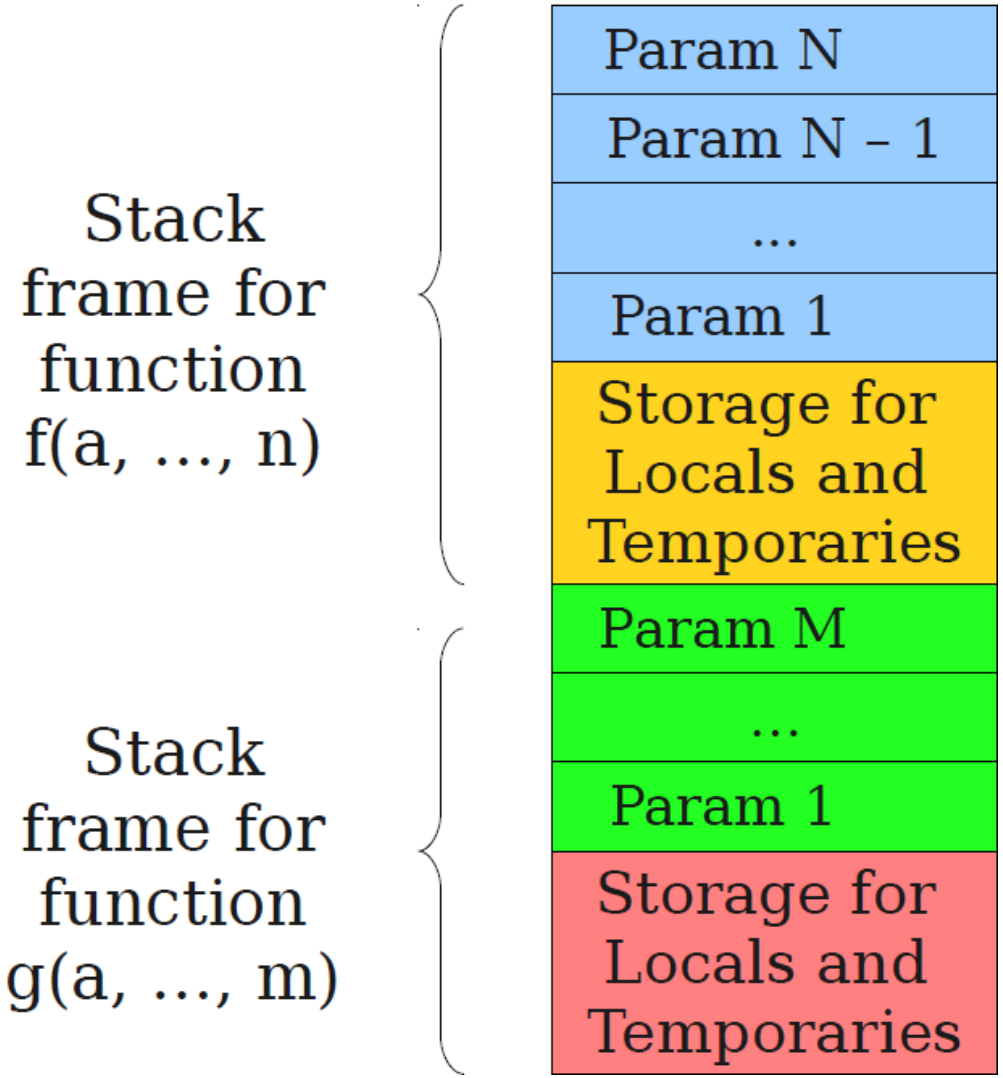
1. Allocation statique
2. Allocation par pile
3. Allocation en tas

I- Gestion des procédures par pile

La procédure "f" dans la pile:



La deuxième procédure "g" dans la pile:



I- Gestion des procédures par pile

Donner le code 3@ du programme "**SimpleFonction**" suivant:

```
void SimpleFn(int z) {
    int x, y;
    x = x * y * z;
}

void main() {
   SimpleFn(137);
}
```

I- Gestion des procédures par pile

Le code 3@ du programme "**SimpleFonction**" :

```
void SimpleFn(int z) {
    int x, y;
    x = x * y * z;
}

void main() {
   SimpleFn(137);
}
```

```
_SimpleFn:
    BeginFunc 16;
    _t0 = x * y;
    _t1 = _t0 * z;
    x = _t1;
    EndFunc;

main:
    BeginFunc 4;
    _t0 = 137;
    Param _t0;
    Call _SimpleFn 1;
    EndFunc;
```

I- Gestion des procédures par pile

Le code 3@, **plus bas niveau** que le précédent, du programme "**SimpleFonction**" :

```
void SimpleFn(int z) {
    int x, y;
    x = x * y * z;
}

void main() {
   SimpleFn(137);
}
```

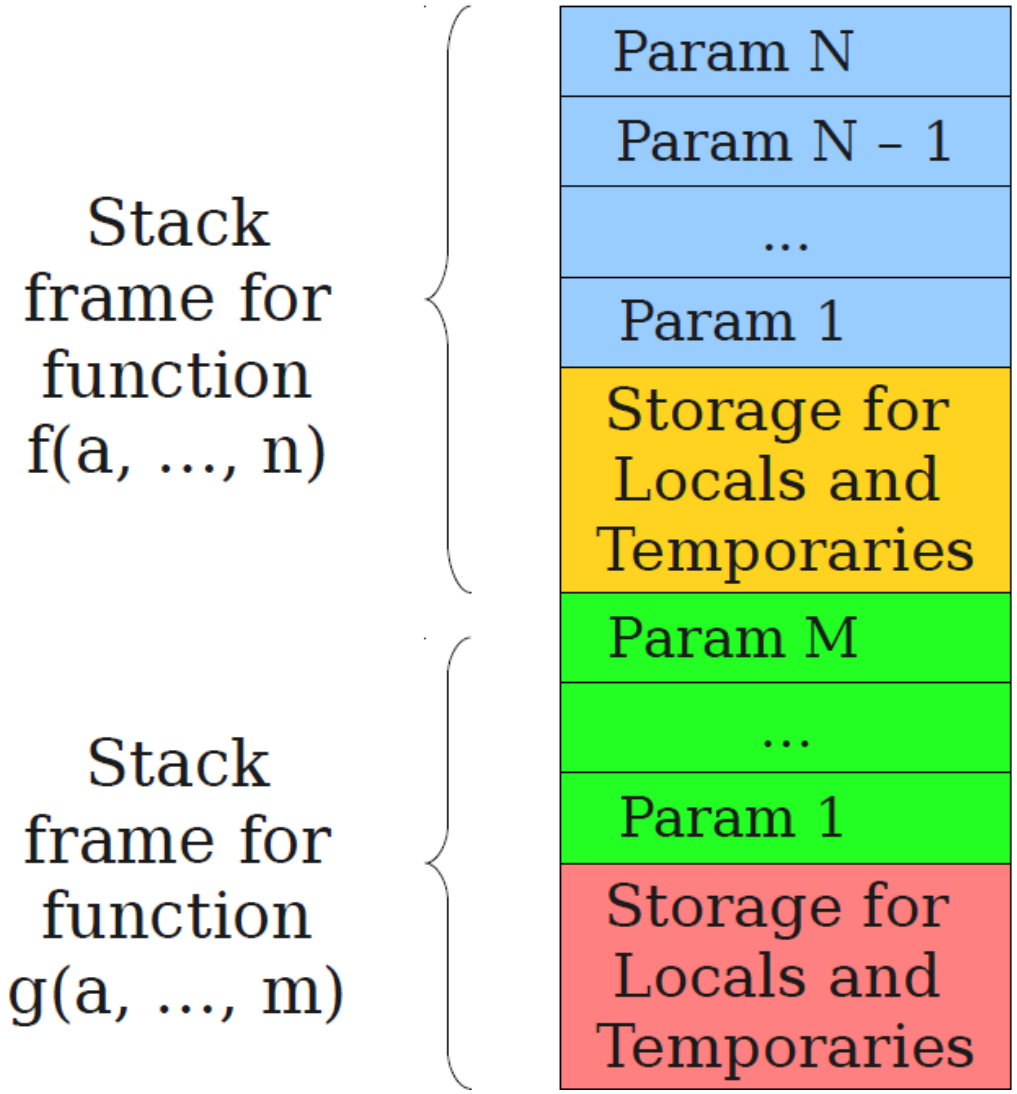
```
_SimpleFn:
    BeginFunc 16;
    _t0 = x * y;
    _t1 = _t0 * z;
    x = _t1;
    EndFunc;

main:
    BeginFunc 4;
    _t0 = 137;
    PushParam _t0;
    LCall _SimpleFn;
    PopParams 4;
    EndFunc;
```

I- Gestion des procédures par pile

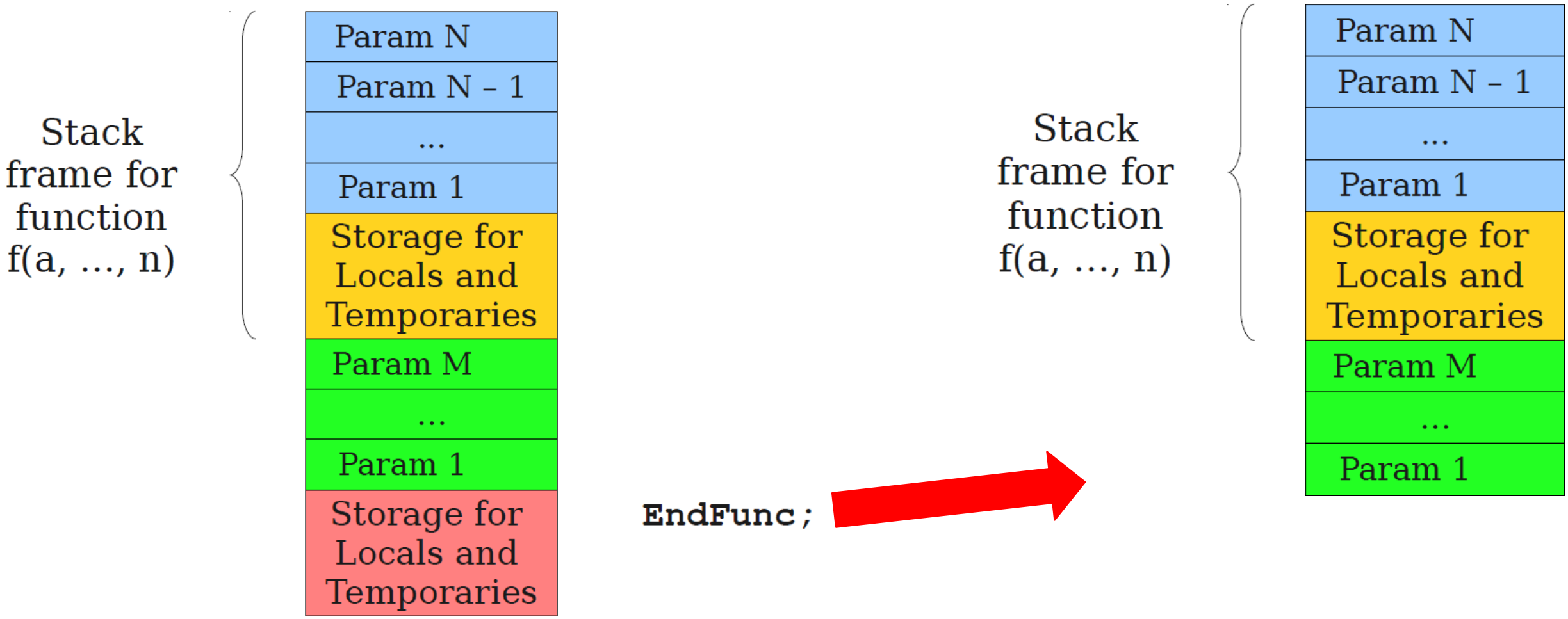
- L'appelant empile un paramètre avec l'instruction **PushParam var**.
- **BeginFunc N** réserve de la place pour les variables locales et temporaires.
- **EndFunc** libère les octets réservés par **BeginFunc N**.
- L'espace des paramètres est libéré par l'appelant en utilisant **PopParams M**. Attention, le **M** est mesuré en octets et non pas le nombre de paramètre.

I- Gestion des procédures par pile

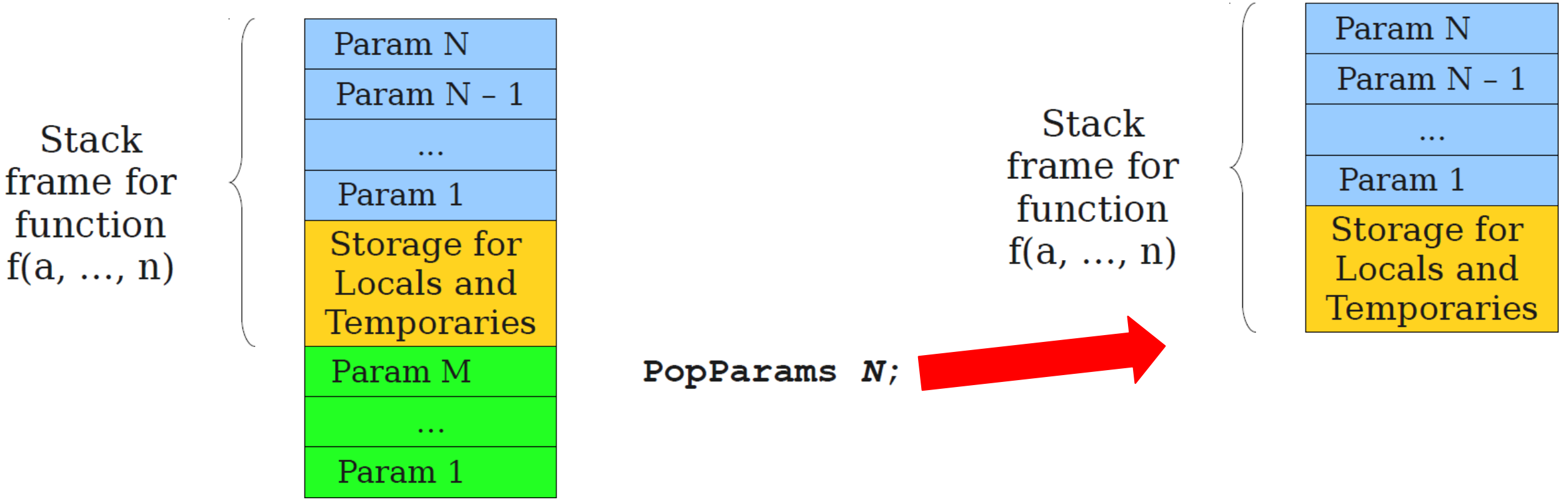



```
PushParam var;
PushParam var;
PushParam var;
BeginFunc N;
```

I- Gestion des procédures par pile



I- Gestion des procédures par pile



Plan

I. Gestion des procédures par pile

II. Flot de contrôle

1. Flot de contrôle avec pile logique
2. Flot de contrôle avec pile physique

III. Activation de procédure

1. Arbres d'activation
2. Exemple: QuickSort
3. Arbres d'activation de l'exemple QuickSort

IV. Allocation de l'espace mémoire

1. Allocation statique
2. Allocation par pile
3. Allocation en tas

II- Flot de contrôle

- Le flot de contrôle est une suite de pas dans le code d'un programme, chaque pas indiquant l'énoncé à exécuter ou l'expression à évaluer.

- On fait les hypothèses suivantes à propos du flot de contrôle:
  - Le flot de contrôle se déplace séquentiellement.
  - L'exécution d'une procédure commence au début du corps de la procédure et rend éventuellement le contrôle au point qui suit immédiatement l'endroit d'où s'est fait l'appel.

Plan

I. Gestion des procédures par pile

II. Flot de contrôle

1. Flot de contrôle avec pile logique
2. Flot de contrôle avec pile physique

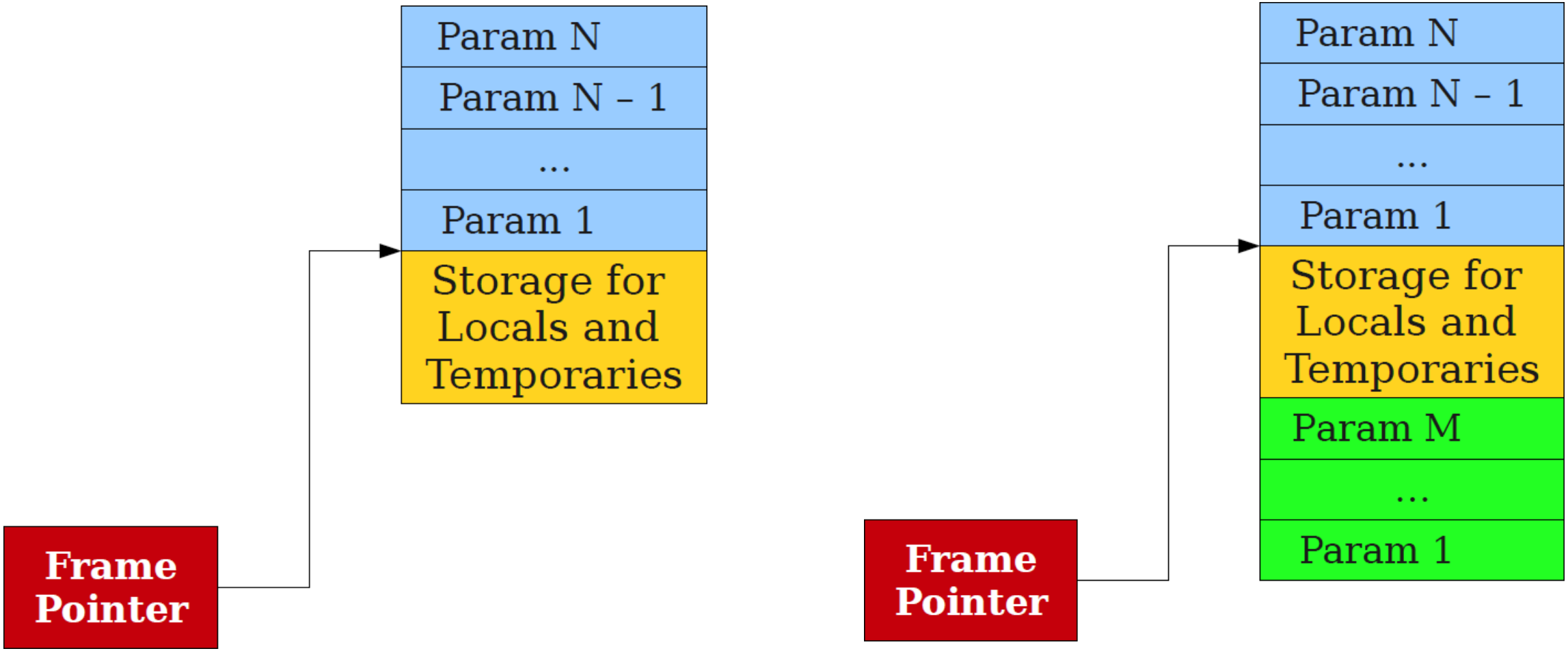
III. Activation de procédure

1. Arbres d'activation
2. Exemple: QuickSort
3. Arbres d'activation de l'exemple QuickSort

IV. Allocation de l'espace mémoire

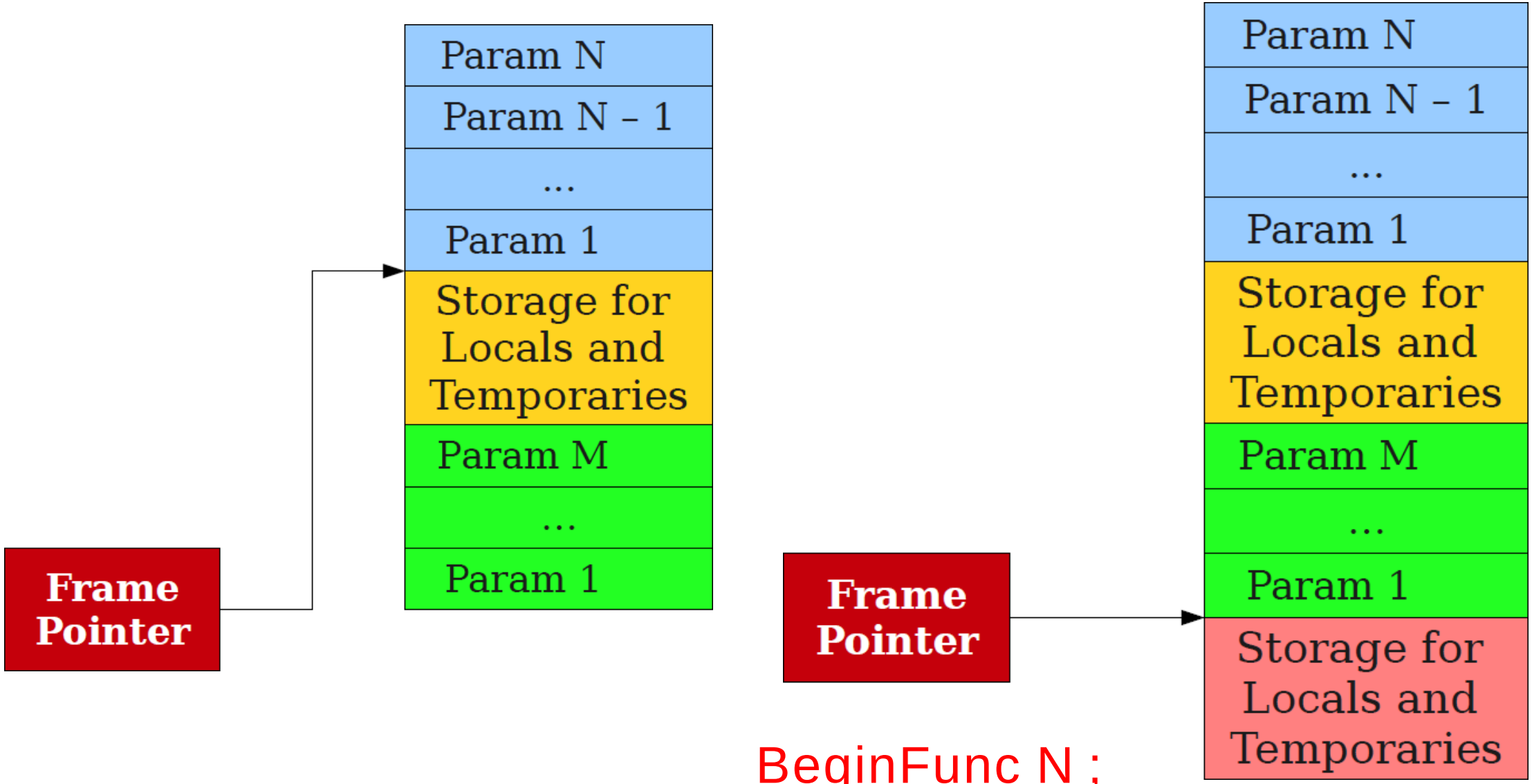
1. Allocation statique
2. Allocation par pile
3. Allocation en tas

II-1. Flot de contrôle avec pile logique

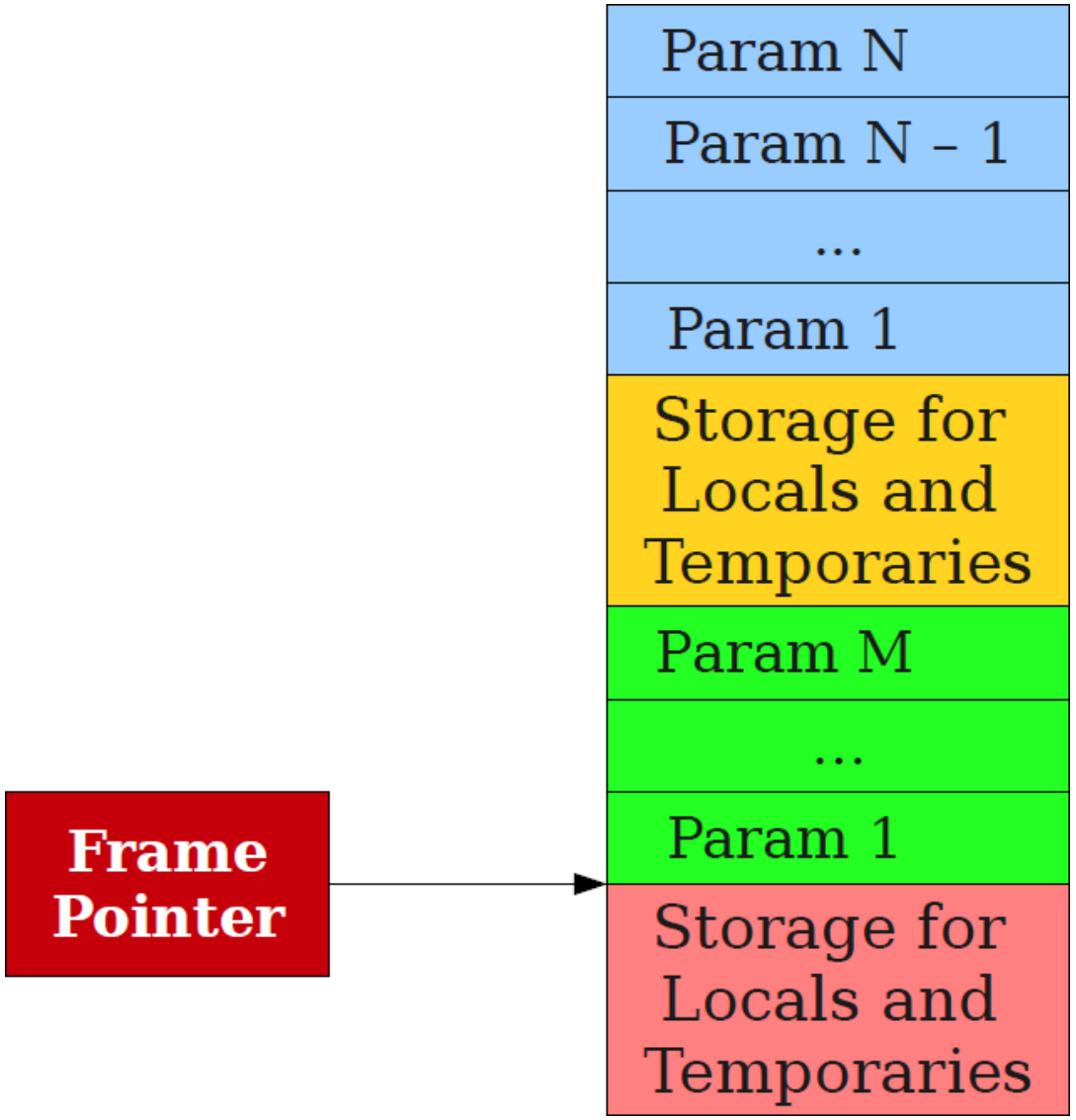


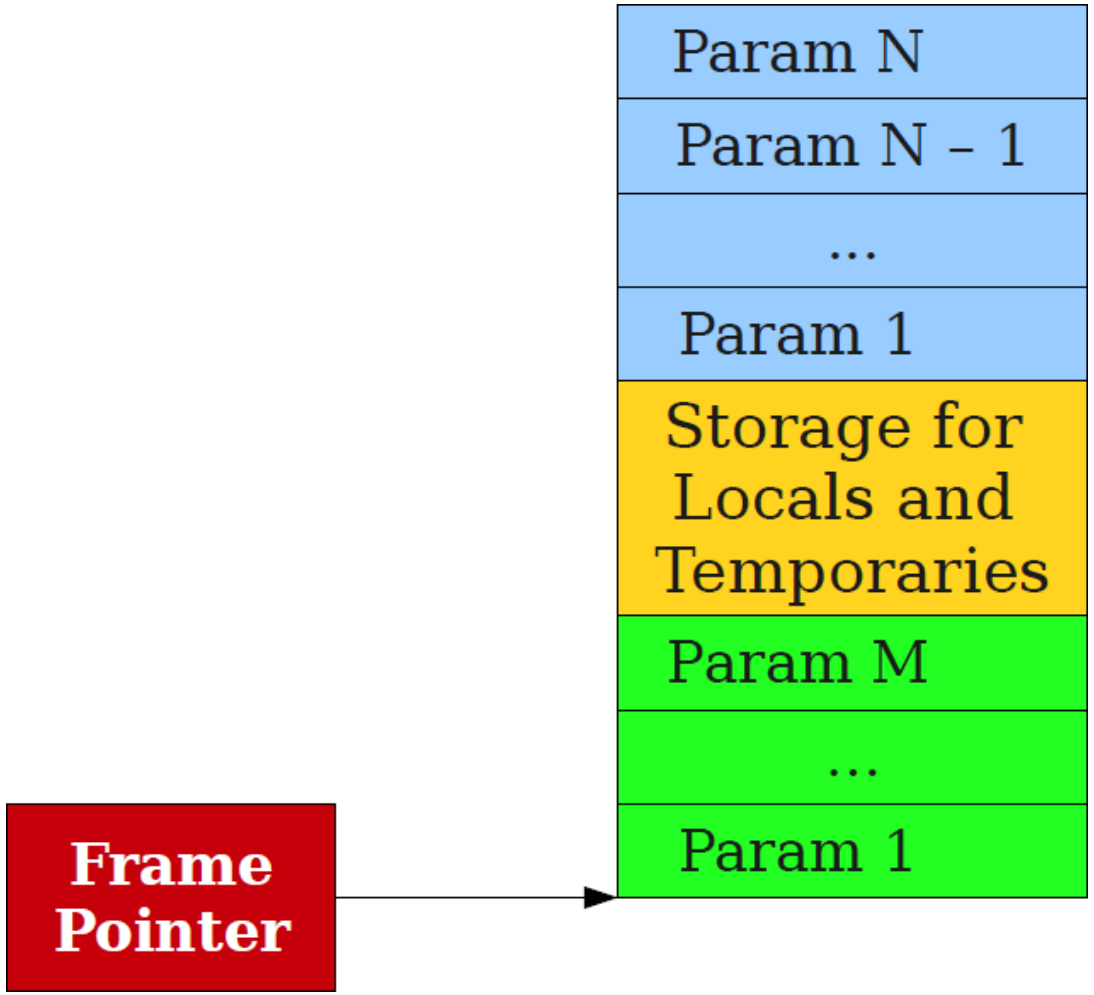
PushParam var ;

II-1. Flot de contrôle avec pile logique



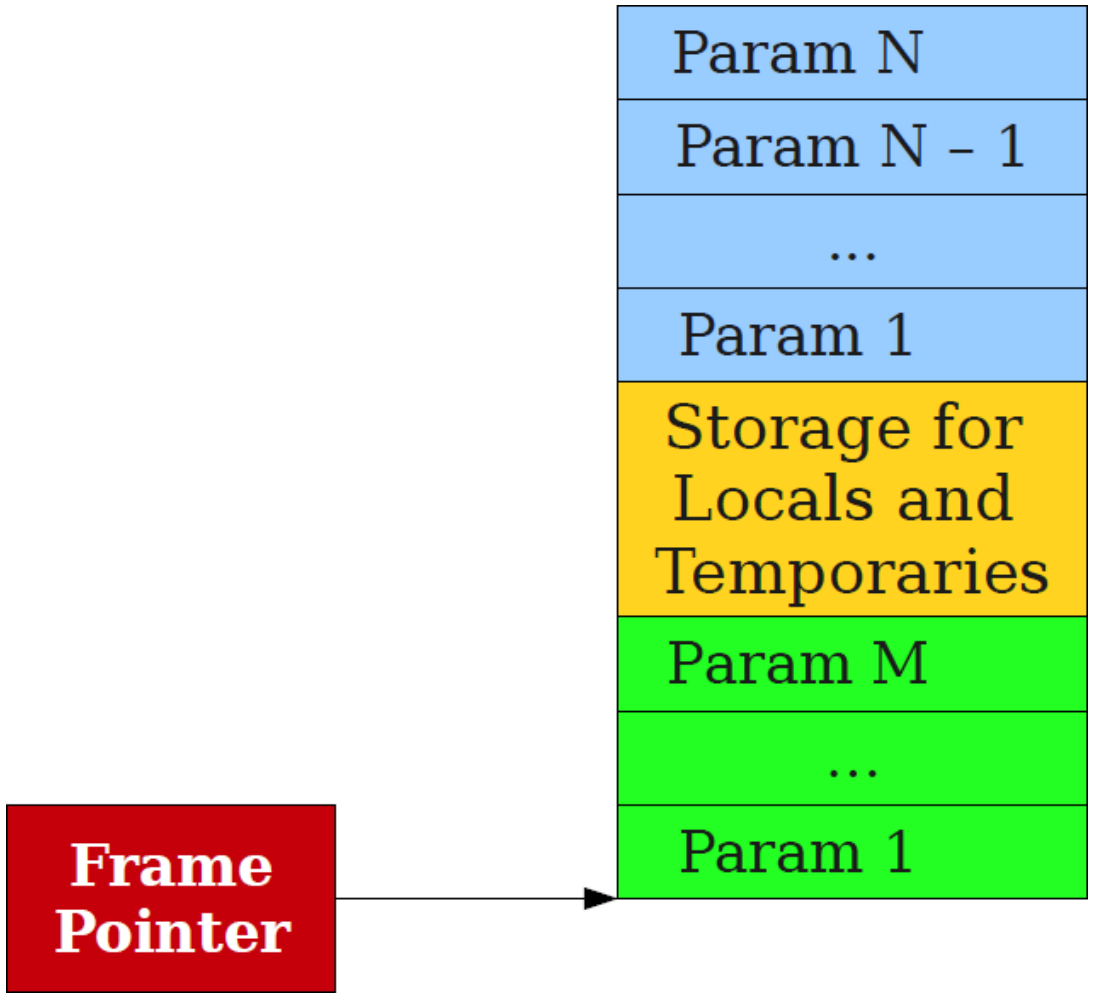
II-1. Flot de contrôle avec pile logique

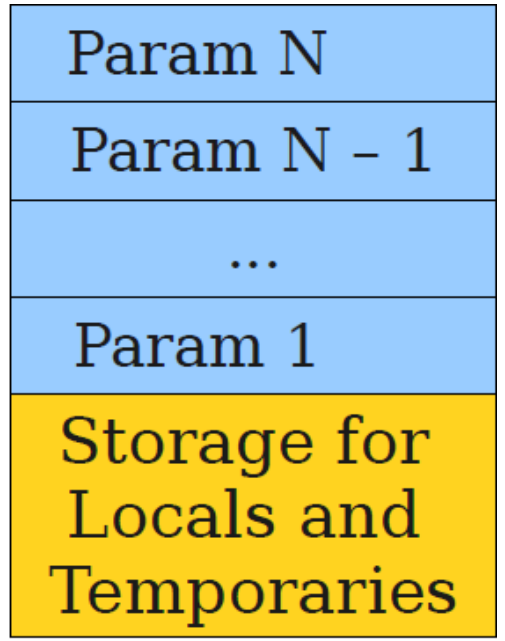




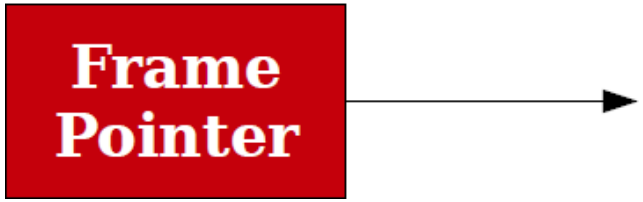
EndFunc ;

II-1. Flot de contrôle avec pile logique



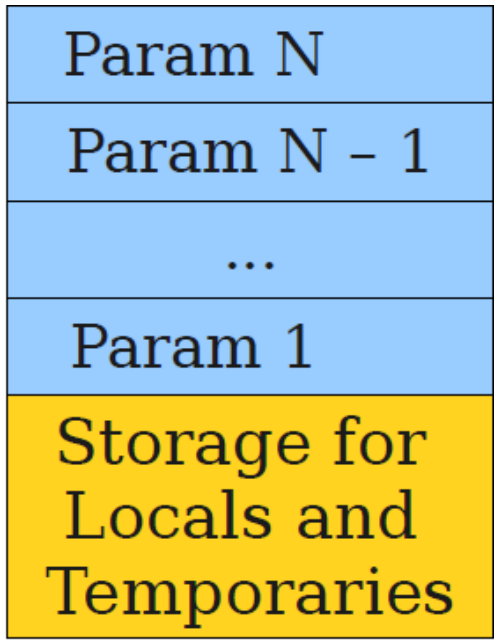


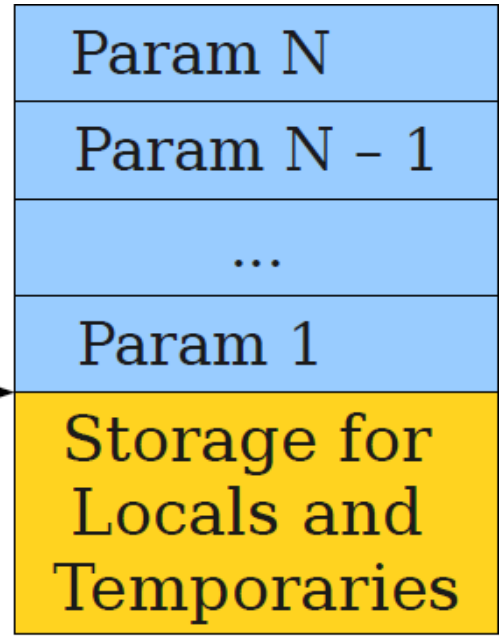
Sur quoi pointer ?



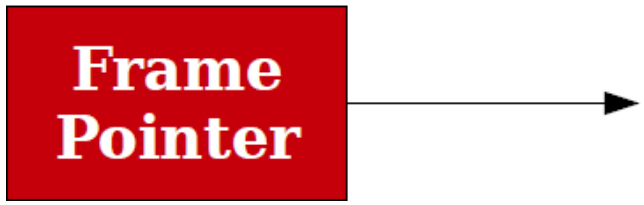
PopParams N ;

II-1. Flot de contrôle avec pile logique





Sur quoi pointer ?



Plan

I. Gestion des procédures par pile

II. Flot de contrôle

1. Flot de contrôle avec pile logique
2. Flot de contrôle avec pile physique

III. Activation de procédure

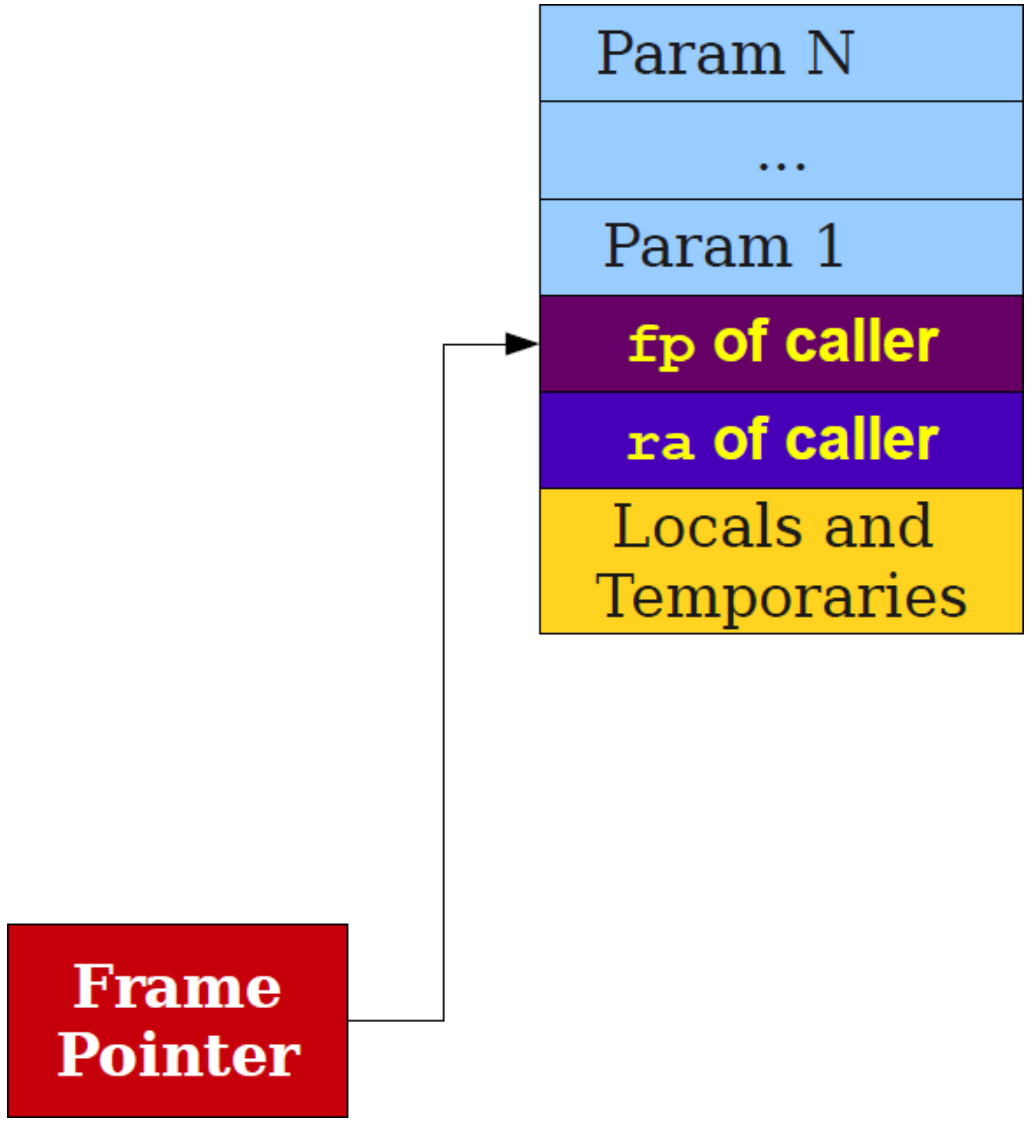
1. Arbres d'activation
2. Exemple: QuickSort
3. Arbres d'activation de l'exemple QuickSort

IV. Allocation de l'espace mémoire

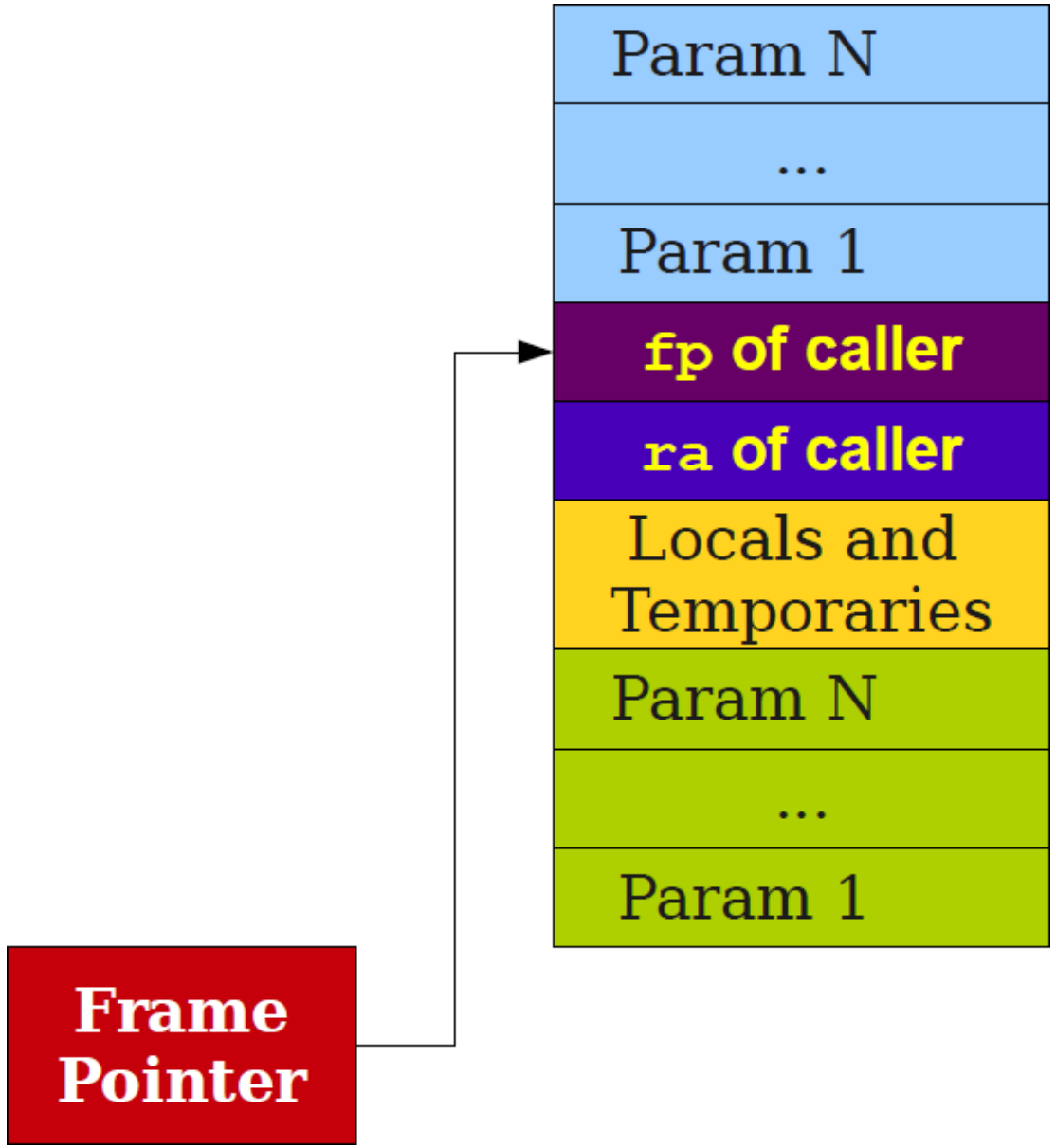
1. Allocation statique
2. Allocation par pile
3. Allocation en tas

II-2. Flot de contrôle avec pile physique

Appelant et Adresse de retour

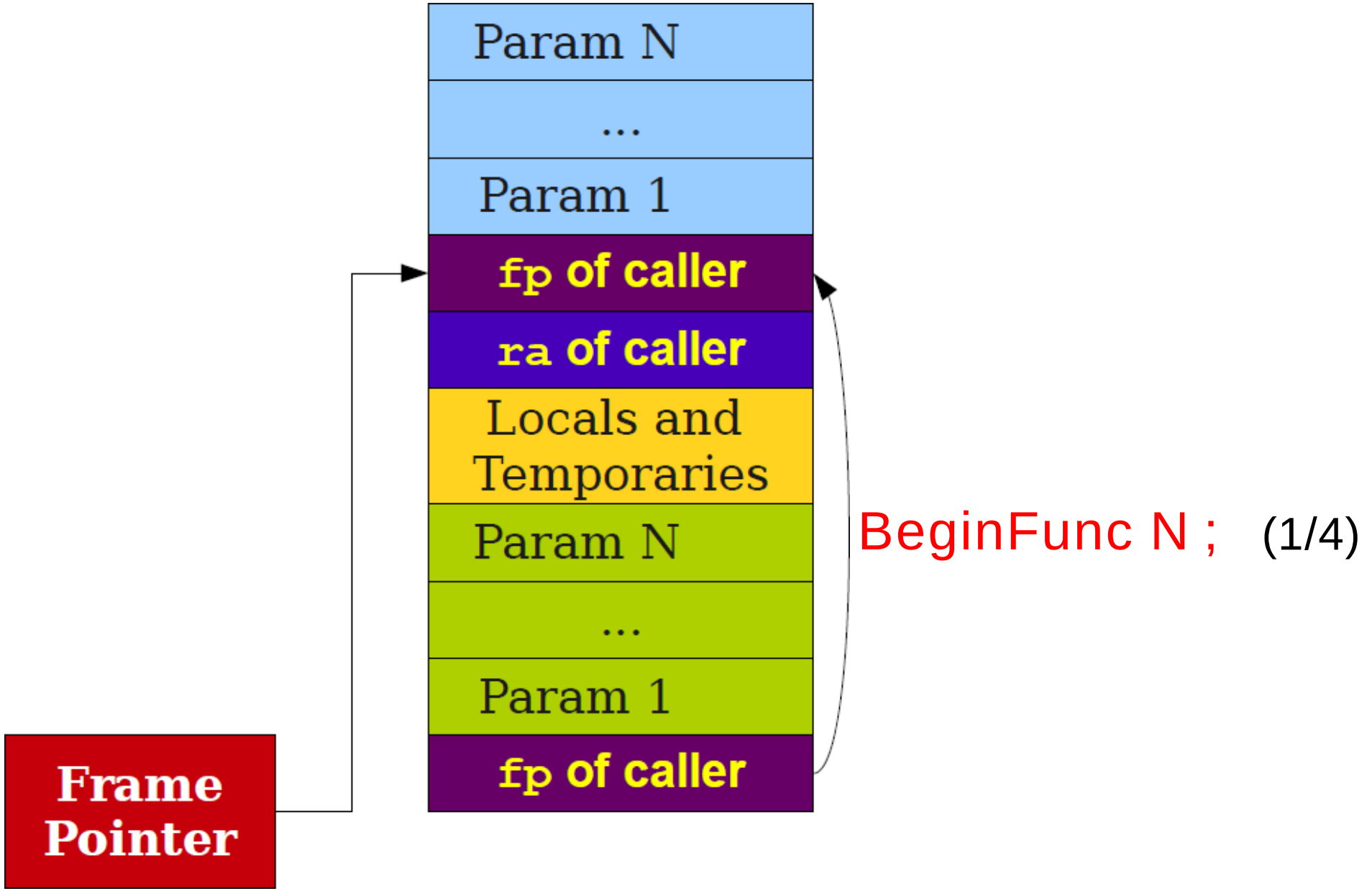


II-2. Flot de contrôle avec pile physique

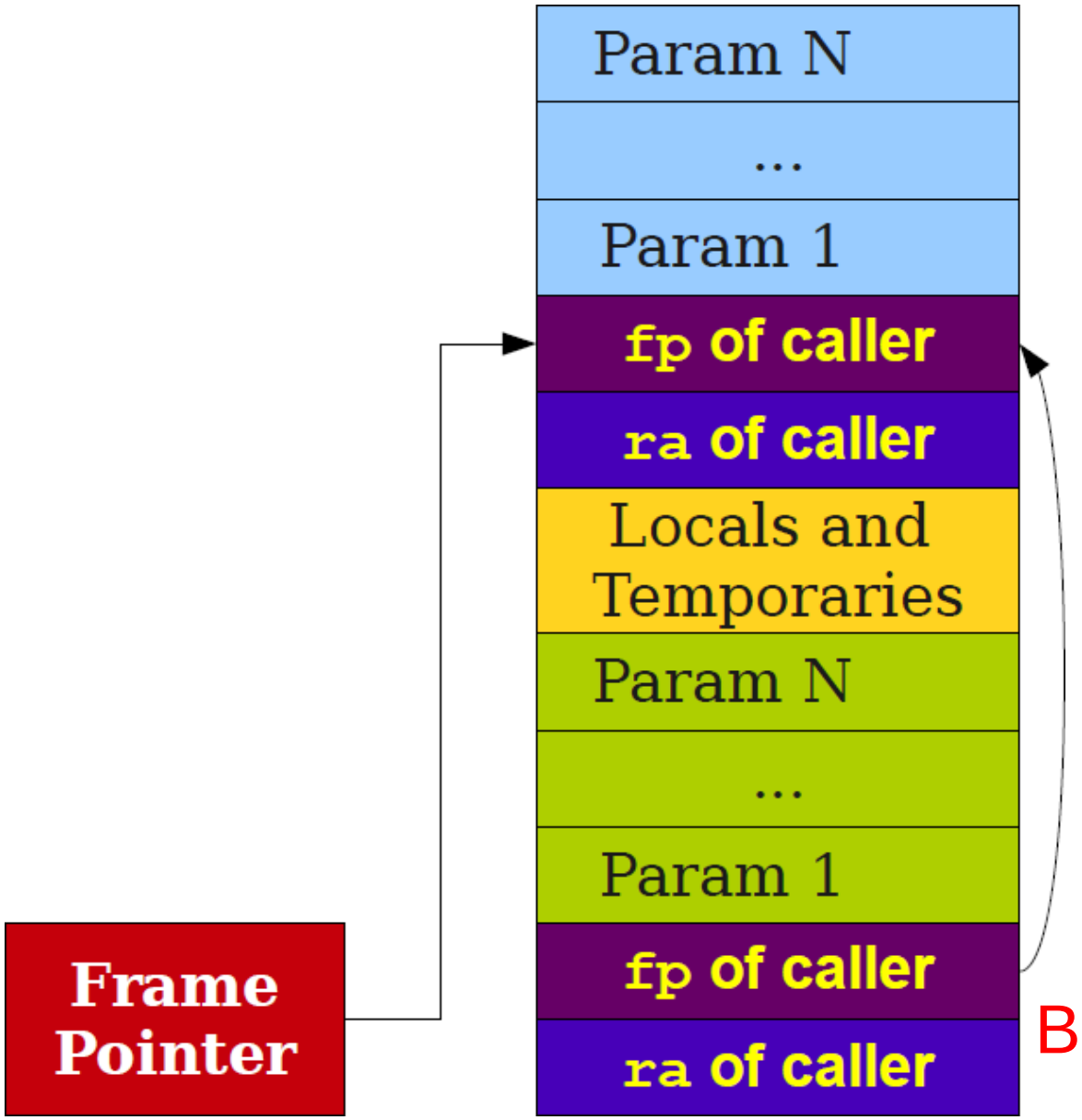


PushParam var ;

II-2. Flot de contrôle avec pile physique

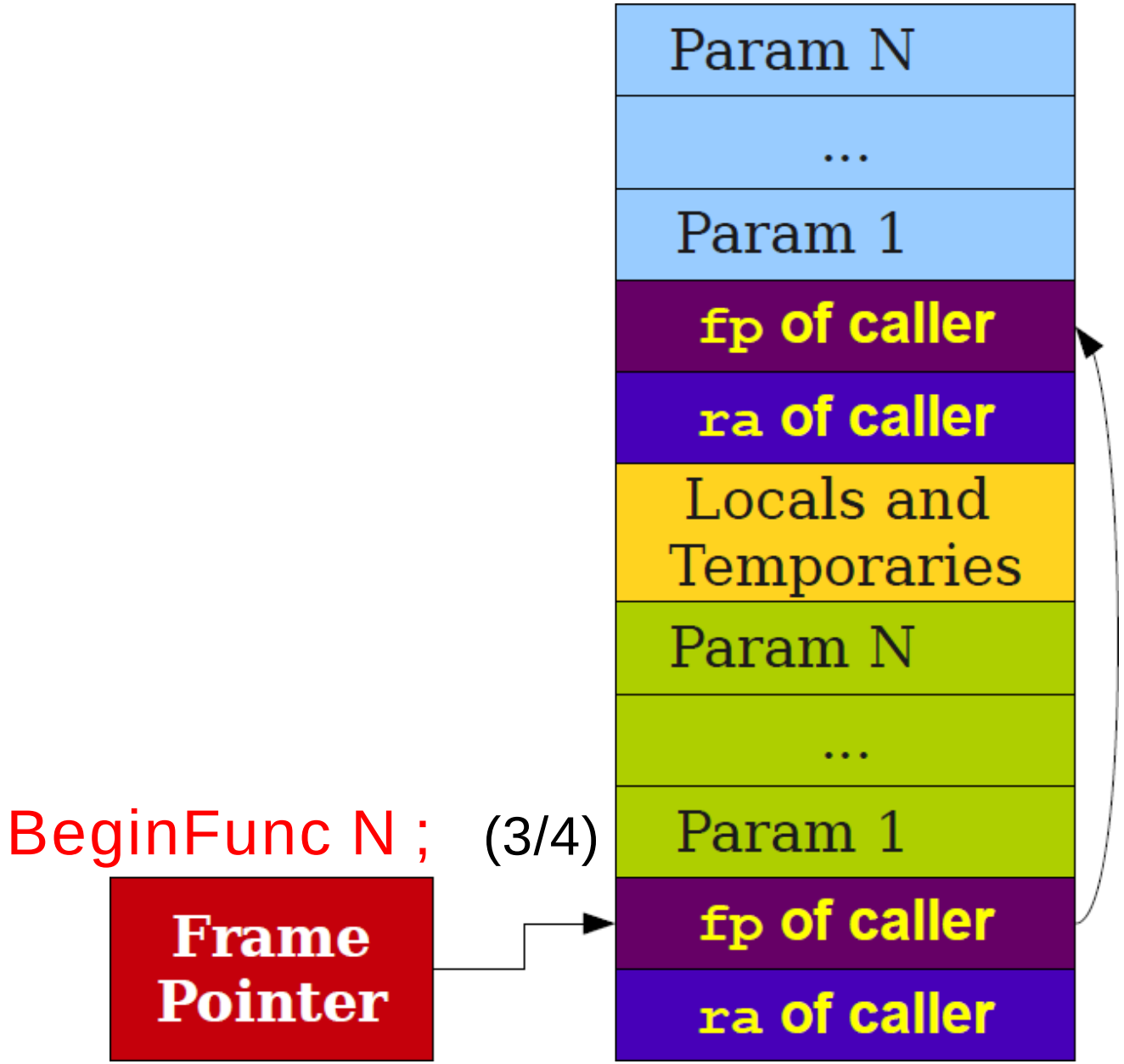


II-2. Flot de contrôle avec pile physique

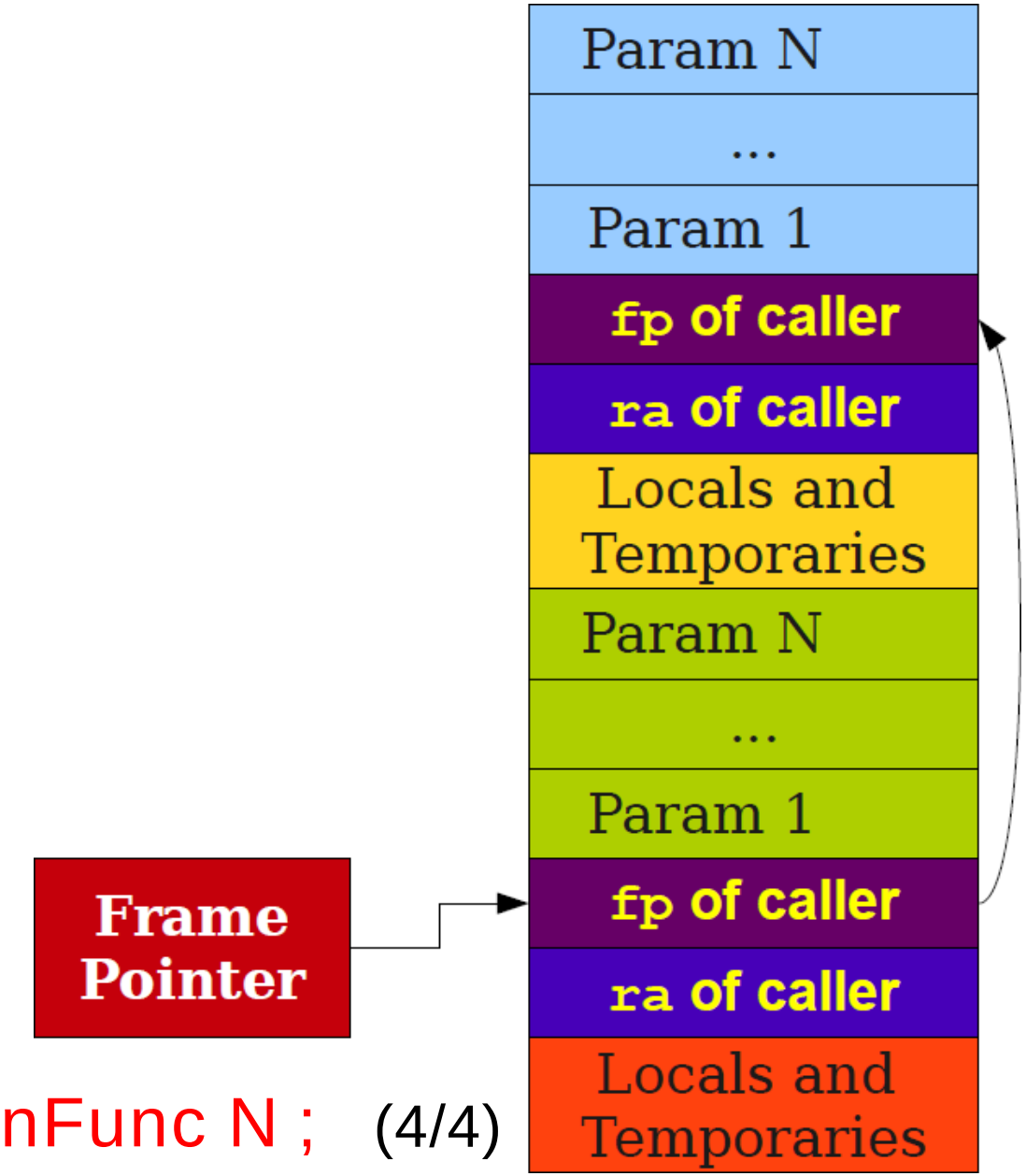


BeginFunc N ; (2/4)

II-2. Flot de contrôle avec pile physique



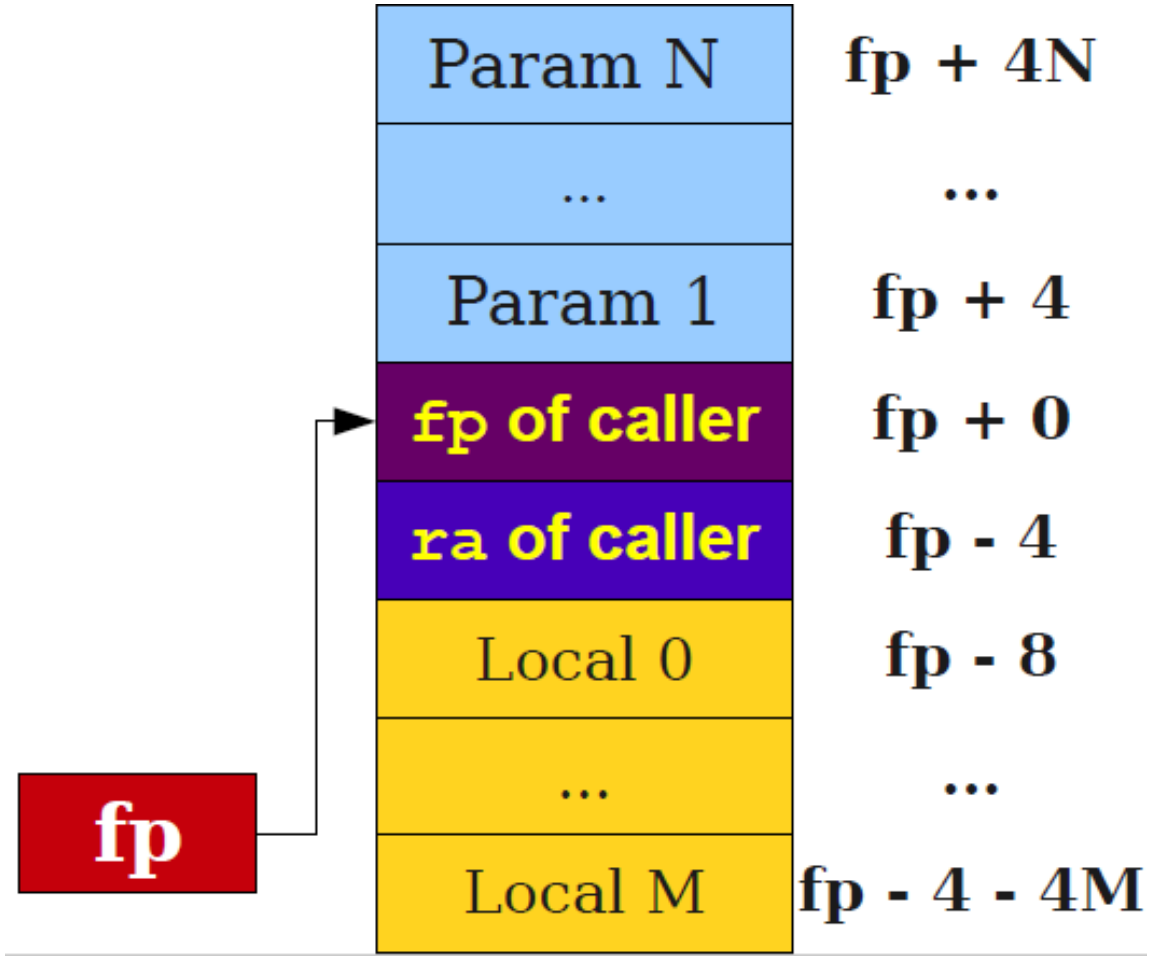
II-2. Flot de contrôle avec pile physique



BeginFunc N ; (4/4)

II-2. Flot de contrôle avec pile physique

- Paramètres et variables **fp-relatives**:
- Les paramètres commencent à **fp+4** et continuent vers le **haut**.
- Les variables locales et temporaires commencent à **fp-8** et continuent vers le **bas**.



II-2. Flot de contrôle avec pile physique

Exercice :

Donner les adresses relatives de toutes les variables du programme "**SimpleFonction**" sachant que le **fp** du main est 100:

```
_SimpleFn:
    BeginFunc 16;
    _t0 = x * y;
    _t1 = _t0 * z;
    x = _t1;
    EndFunc;

main:
    BeginFunc 4;
    _t0 = 137;
    PushParam _t0;
    LCall _SimpleFn;
    PopParams 4;
    EndFunc;
```

Plan

I. Gestion des procédures par pile

II. Flot de contrôle

1. Flot de contrôle avec pile logique
2. Flot de contrôle avec pile physique

III. Activation de procédure

1. Arbres d'activation
2. Exemple: QuickSort
3. Arbres d'activation de l'exemple QuickSort

IV. Allocation de l'espace mémoire

1. Allocation statique
2. Allocation par pile
3. Allocation en tas

III- Activation de procédure

- La durée de vie de l'activation d'une procédure "**p**" est la séquence des pas entre le premier et le dernier pas de l'exécution du corps de la procédure, inclusivement.

- **Propriété**: Si ***a*** et ***b*** sont des activations de procédures, alors leurs durées de vie sont soit disjointes, soit imbriquées.

- Une procédure est récursive si une nouvelle activation de la procédure peut débuter avant la terminaison d'une activation ayant débuté auparavant. La récursivité n'a pas à être directe.

Plan

I. Gestion des procédures par pile

II. Flot de contrôle

1. Flot de contrôle avec pile logique
2. Flot de contrôle avec pile physique

III. Activation de procédure

1. Arbres d'activation
2. Exemple: QuickSort
3. Arbres d'activation de l'exemple QuickSort

IV. Allocation de l'espace mémoire

1. Allocation statique
2. Allocation par pile
3. Allocation en tas

III-1. Arbres d'activation

On peut présenter graphiquement le flot de contrôle d'une exécution qui implique des procédures à l'aide d'un arbre d'activation où:

- Chaque nœud représente une activation d'une procédure;
- La racine représente l'activation du programme principal;
- Le nœud d'une activation ***a*** est le parent d'une activation ***b*** si et seulement si le contrôle passe [directement] de ***a*** à ***b***;
- Le nœud associé à ***a*** se situe à la gauche du nœud associé à ***b*** si et seulement si la durée de vie de ***a*** a lieu avant la durée de vie de ***b***.

Plan

I. Gestion des procédures par pile

II. Flot de contrôle

1. Flot de contrôle avec pile logique
2. Flot de contrôle avec pile physique

III. Activation de procédure

1. Arbres d'activation
2. Exemple: QuickSort
3. Arbres d'activation de l'exemple QuickSort

IV. Allocation de l'espace mémoire

1. Allocation statique
2. Allocation par pile
3. Allocation en tas

III-2. Exemple: QuickSort

Le tri QuickSort, tri rapide en français, est un algorithme de tri qui est basé sur le principe de "diviser pour régner":
Décomposer le problème de tri d'un tableau en sous-problèmes de tri de sous tableaux.

Principe du QuickSort :

1. On sélectionne un élément pivot de la liste de départ ;
2. On partitionne le tableau en deux sous-tableaux:
   - Un premier tableau avec les éléments qui sont inférieurs ou égaux au pivot;
   - Un deuxième tableau avec les éléments qui sont supérieurs au pivot

   Le pivot se retrouve ainsi placé dans sa position définitive entre ces deux sous-tableaux;
3. Répéter récursivement les étapes 1 et 2 aux sous-tableaux jusqu'à ce qu'ils soient réduits à un élément.

III-2. Exemple: QuickSort

Algorithm du QuickSort :

```
algorithm quicksort(m, n) is
    if m < n then
        p := partition(m, n)
        quicksort(m, p)
        quicksort(p+1, n)

algorithm partition(m, n) is
    pivot := A[m]
    i := m - 1
    j  :=  n  +  1
    loop forever
        do i := i + 1 while array[i] < pivot
        do j := j - 1 while array[j] > pivot
        if i >= j then return j
        swap array[i] with array[j].
```

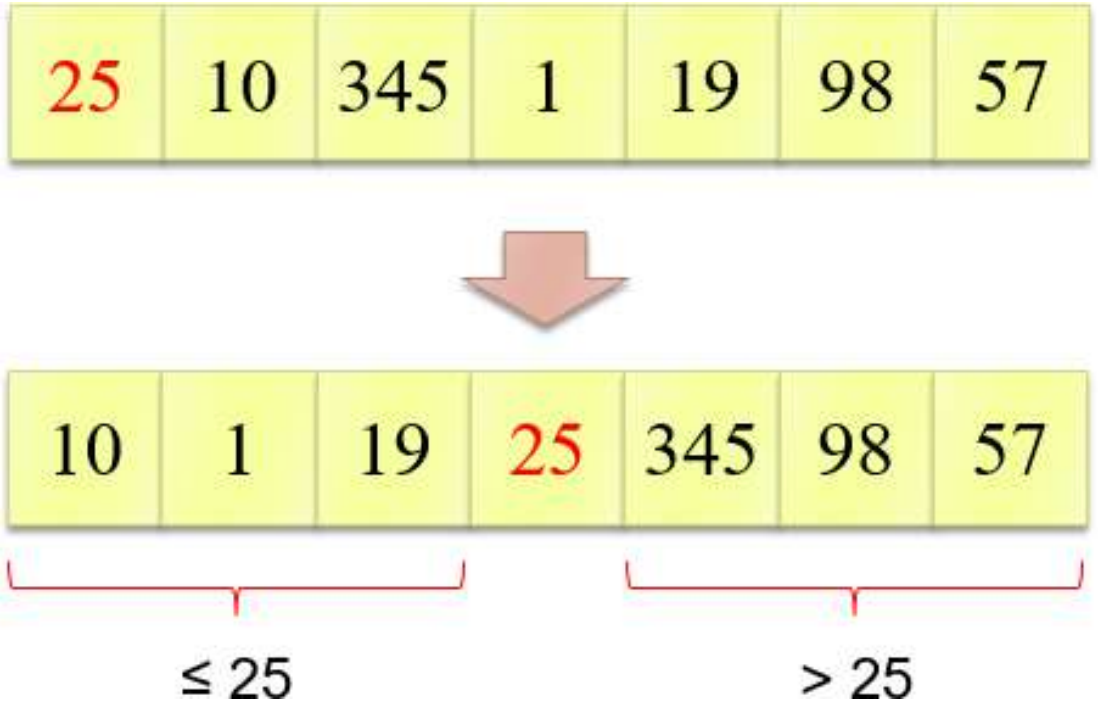
III-2. Exemple: QuickSort

Exemple :

III-2. Exemple: QuickSort

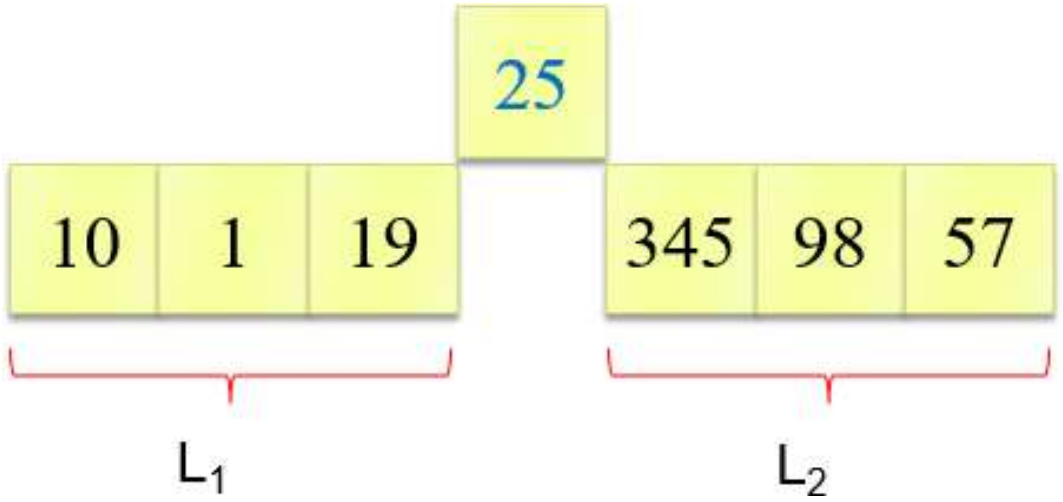
1. Choisir un pivot (premier élément):

2. Partitionner le tableau :



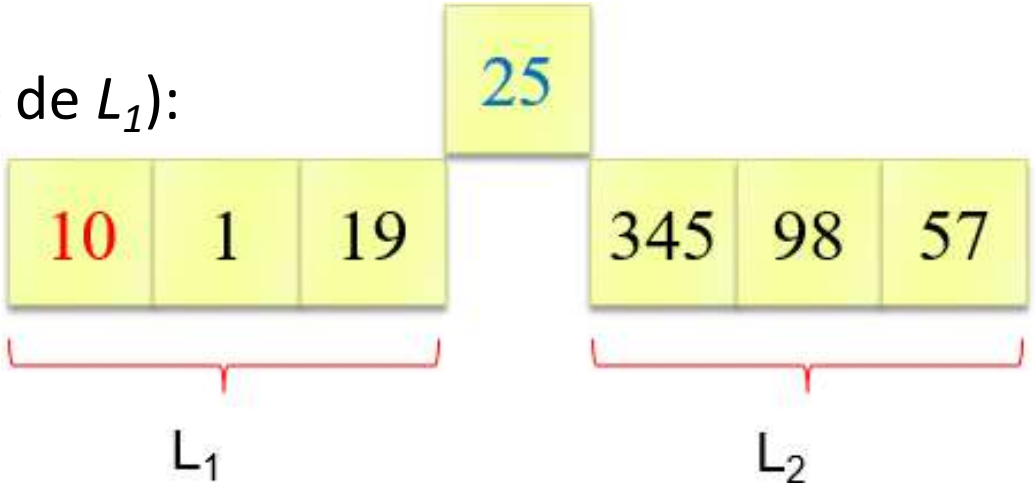
III-2. Exemple: QuickSort

3. Refaire les deux premières étapes sur les deux tableaux L_1 et L_2 :

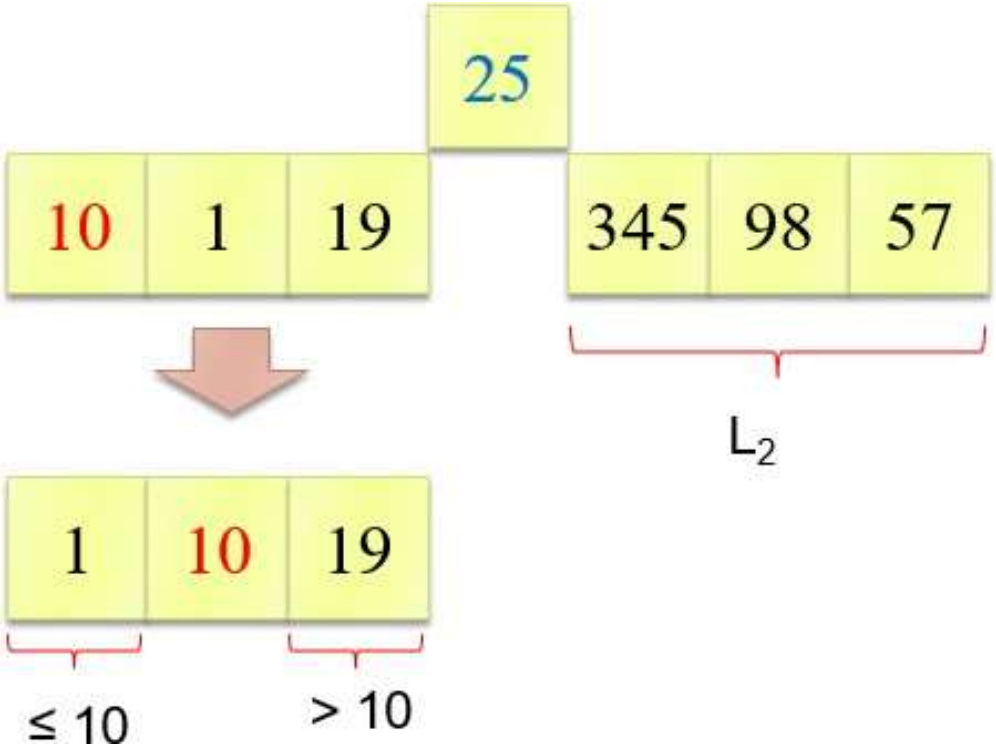


III-2. Exemple: QuickSort

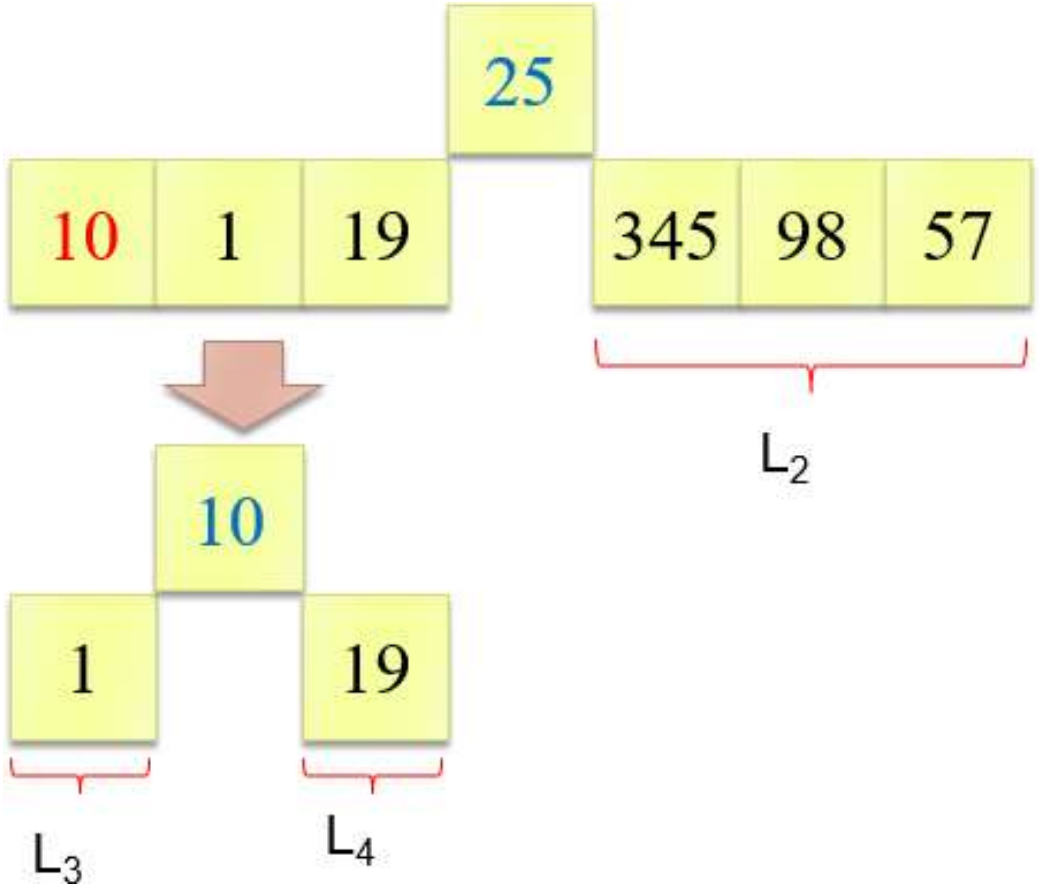
1. Choisir un pivot (premier élément de L_1):



2. Partitionner le tableau L_1 en L_3 *et* L_4



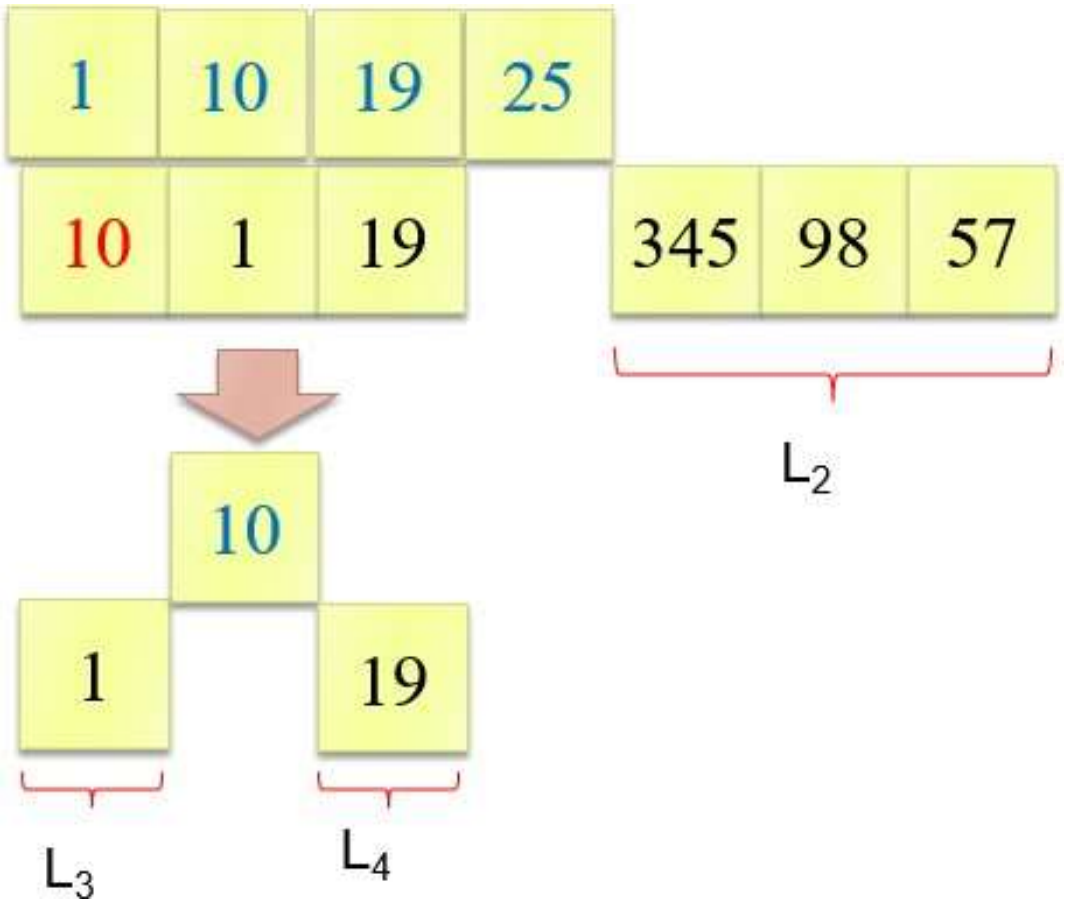
III-2. Exemple: QuickSort



3. Comme L_3 et L_4 ont un élément chacun, le tri de L_1 est terminé

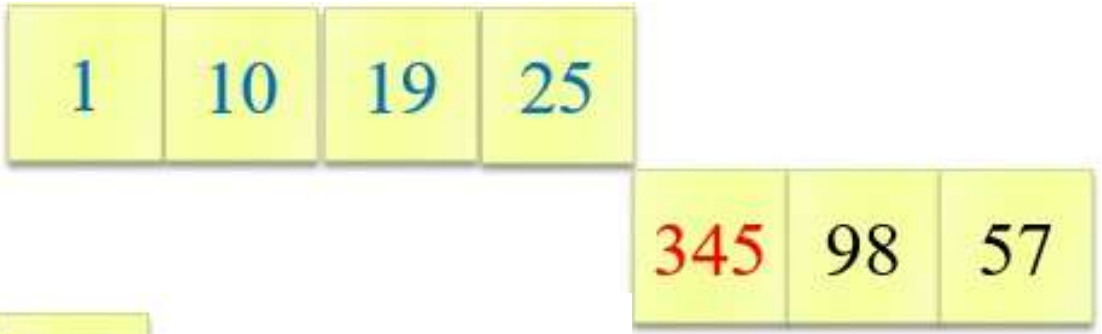
III-2. Exemple: QuickSort

Continuer récursivement avec L_2 :

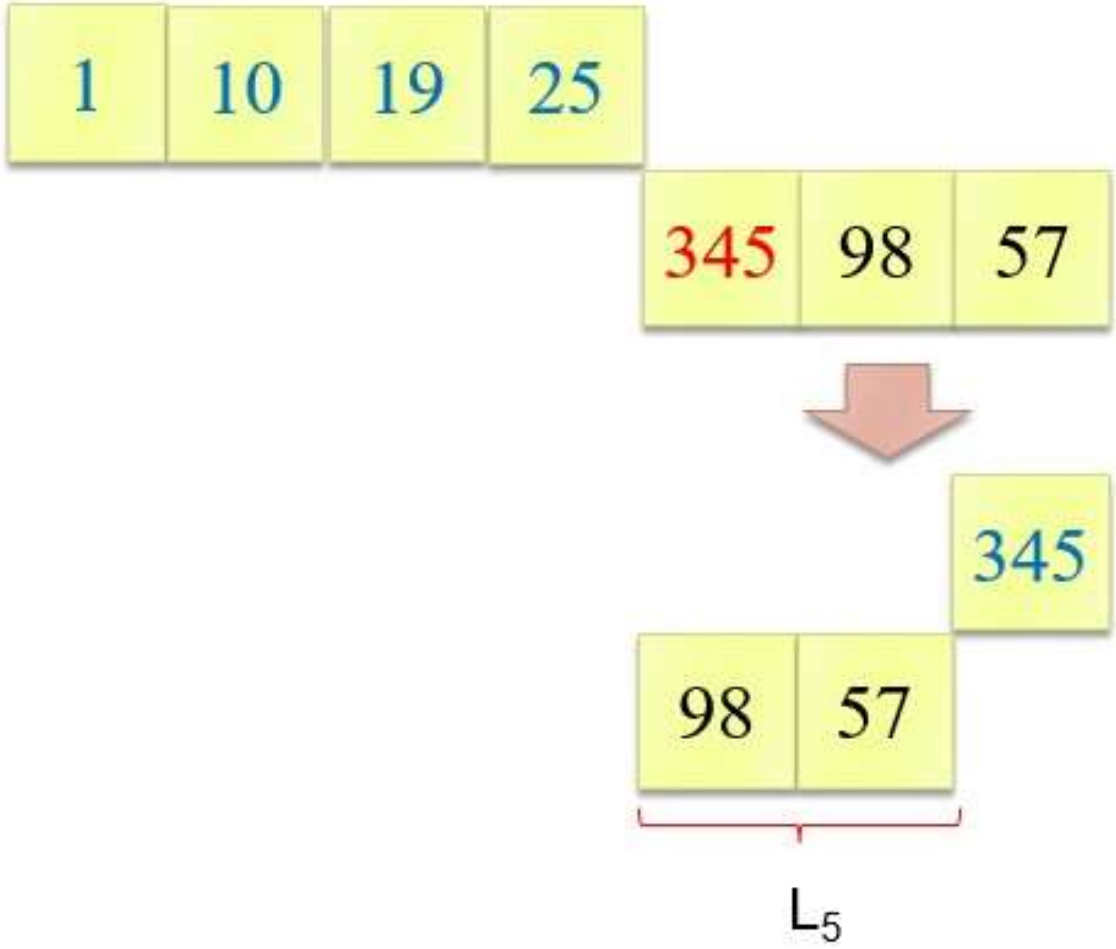


III-2. Exemple: QuickSort

1. Choisir un pivot (premier élément de L_2):



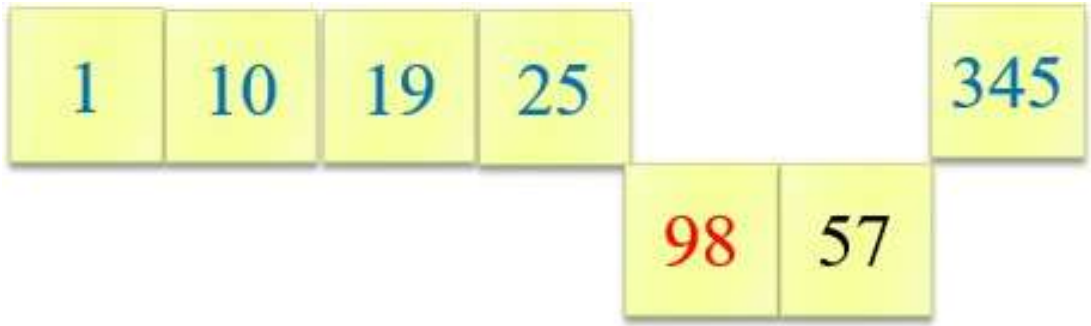
2. Partitionner le tableau L_2 :



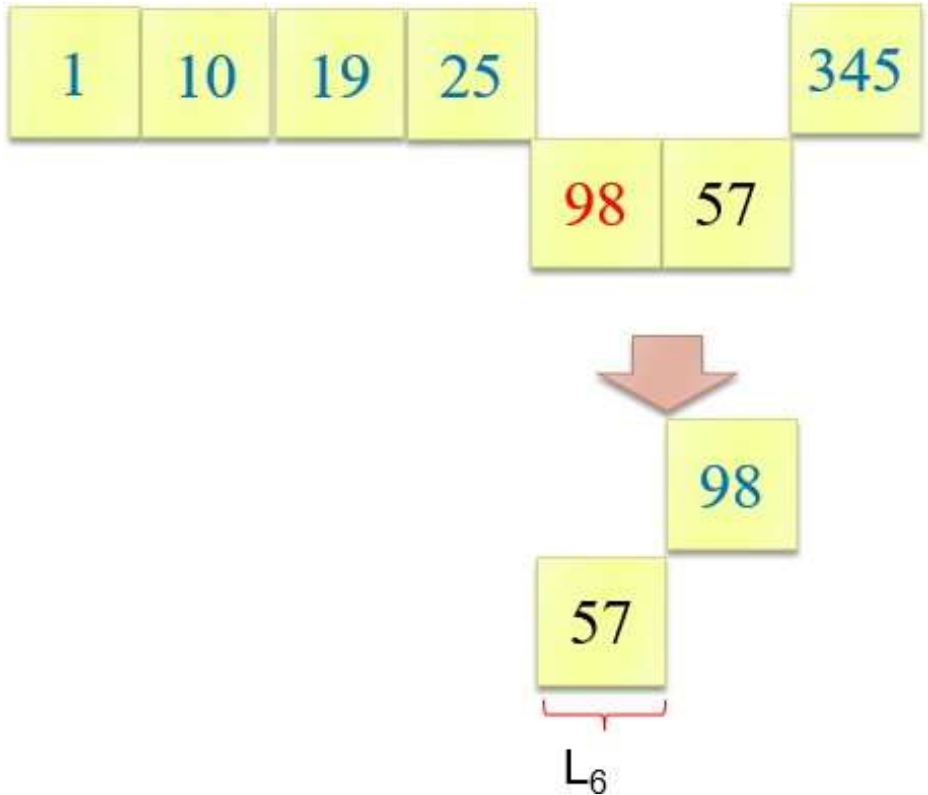
III-2. Exemple: QuickSort

Continuer récursivement avec L_5 :

1. Choisir un pivot (premier élément de L_5):



2. Partitionner le tableau L_5 :



III-2. Exemple: QuickSort

Tableau final trié:

Complexité:
cas moyen : $O(n \log n)$
pire des cas : $O(n^2)$

Plan

I. Gestion des procédures par pile

II. Flot de contrôle

1. Flot de contrôle avec pile logique
2. Flot de contrôle avec pile physique

III. Activation de procédure

1. Arbres d'activation
2. Exemple: QuickSort
3. Arbres d'activation de l'exemple QuickSort

IV. Allocation de l'espace mémoire

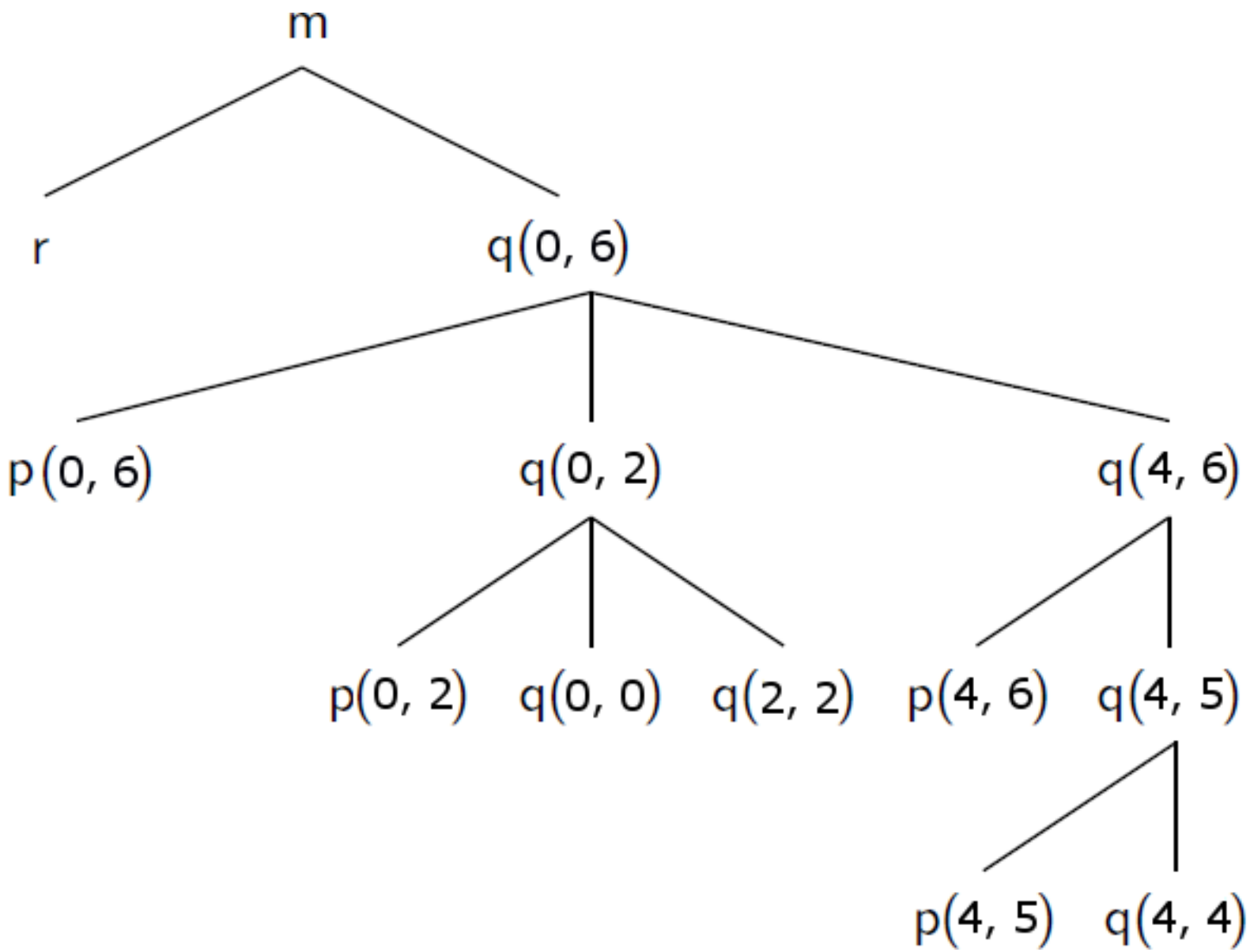
1. Allocation statique
2. Allocation par pile
3. Allocation en tas

III-3. Arbres d'activation de l'exemple QuickSort

On peut présenter graphiquement le flot de contrôle d'une exécution qui implique des procédures à l'aide d'un arbre d'activation où:

- Chaque nœud représente une activation d'une procédure;
- La racine représente l'activation du programme principal;
- Le nœud d'une activation ***a*** est le parent d'une activation ***b*** si et seulement si le contrôle passe [directement] de ***a*** à ***b***;
- Le nœud associé à a se situe à la gauche du nœud associé à b si et seulement si la durée de vie de ***a*** a lieu avant la durée de vie de ***b***.

III-3. Arbres d'activation de l'exemple QuickSort



Plan

I. Gestion des procédures par pile

II. Flot de contrôle

1. Flot de contrôle avec pile logique
2. Flot de contrôle avec pile physique

III. Activation de procédure

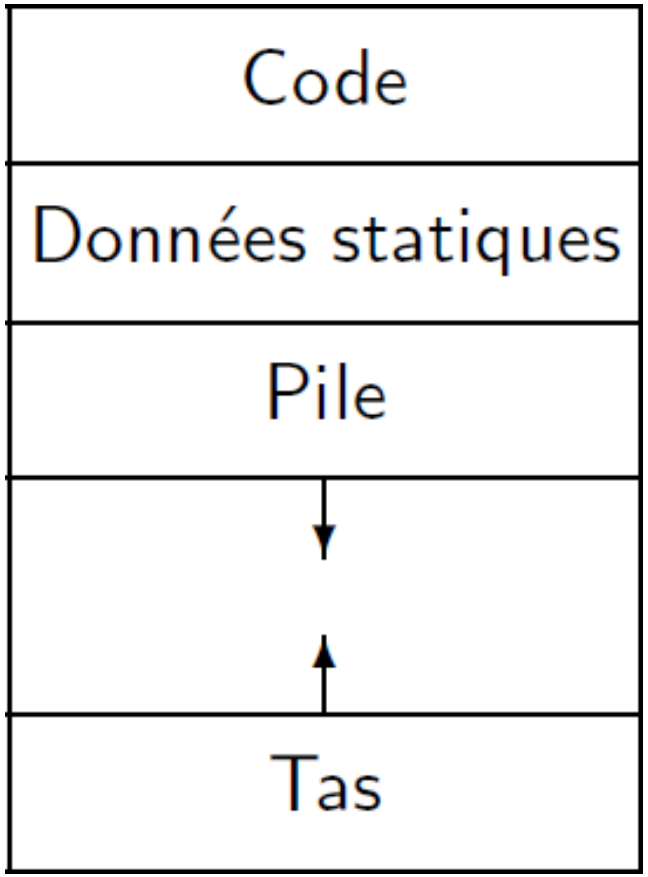
1. Arbres d'activation
2. Exemple: QuickSort
3. Arbres d'activation de l'exemple QuickSort

IV. Allocation de l'espace mémoire

1. Allocation statique
2. Allocation par pile
3. Allocation en tas

IV- Allocation de l'espace mémoire

L'organisation de l'espace de rangement dans les langages comme Fortran, Pascal, C et Java, etc. :



IV- Allocation de l’espace mémoire

Les stratégies suivantes sont utilisées dans les trois zones de données d’un programme:

- L’allocation **statique** détermine la disposition finale des données;
- L’allocation **par pile** gère l’espace à la manière d’une pile;
- L’allocation **en tas** permet l’allocation et la libération des objets sans contraintes sur l’ordre où ces opérations sont effectuées.

Plan

I. Gestion des procédures par pile

II. Flot de contrôle

1. Flot de contrôle avec pile logique
2. Flot de contrôle avec pile physique

III. Activation de procédure

1. Arbres d'activation
2. Exemple: QuickSort
3. Arbres d'activation de l'exemple QuickSort

IV. Allocation de l'espace mémoire

1. Allocation statique
2. Allocation par pile
3. Allocation en tas

IV-1. Allocation statique

- L’allocation statique permet de lier chaque nom à un emplacement fixe et ce, dès la compilation.
- L’allocation statique est couramment utilisée pour les variables globales mais permet aussi à des noms locaux de devenir persistants; c’est-à-dire que la valeur de ces variables est conservée d’une activation d’une procédure à l’autre.

IV-1. Allocation statique

Avantages :

- C’est l’allocation la plus simple.
- C’est l’allocation la plus efficace: plus rien à faire à l’exécution.

Inconvénients :

- L’allocation dynamique est impossible.
- La taille et les contraintes d’alignement des objets doivent être connues à la compilation.
- Des procédures récursives générales ne peuvent pas être implantées.
- Les blocs d’activations et les données des procédures occupent de l’espace, même en l’absence d’activation.

Plan

I. Gestion des procédures par pile

II. Flot de contrôle

1. Flot de contrôle avec pile logique
2. Flot de contrôle avec pile physique

III. Activation de procédure

1. Arbres d'activation
2. Exemple: QuickSort
3. Arbres d'activation de l'exemple QuickSort

IV. Allocation de l'espace mémoire

1. Allocation statique
2. Allocation par pile
3. Allocation en tas

IV-2. Allocation par pile

- L'allocation par pile profite du fait qu'une pile de contrôle est déjà utilisée pour effectuer l'exécution du programme.
- À chaque activation de procédure, on réserve (empile) un bloc de mémoire pour accommoder la structure d'activation et les variables locales de la procédure.
- À la fin de l'activation, le bloc est libéré (dépilé) : l'espace de rangement qui servait à contenir les valeurs des variables locales est libéré.

IV-2. Allocation par pile

Avantages :

- L'allocation dynamique est possible.
- Permet la récursivité générale.
- Seuls les blocs d'activations et les données des procédures activées occupent de l'espace.
- Allocation et libération rapides.
- Accès aux données presqu'aussi efficace qu'avec l'allocation statique.

Inconvénients :

- Le compilateur doit calculer statiquement la position relative des variables et des champs de la structure d'activation.
- Allocation dynamique, mais seulement en régime de pile.

Plan

I. Gestion des procédures par pile

II. Flot de contrôle

1. Flot de contrôle avec pile logique
2. Flot de contrôle avec pile physique

III. Activation de procédure

1. Arbres d'activation
2. Exemple: QuickSort
3. Arbres d'activation de l'exemple QuickSort

IV. Allocation de l'espace mémoire

1. Allocation statique
2. Allocation par pile
3. Allocation en tas

IV-3. Allocation en tas

L'allocation en tas devient nécessaire dans les situations suivantes:

- La valeur de certaines variables locales doit continuer à exister après l'activation d'une procédure.

- L'activation de l'appelé survit plus longtemps que l'activation de l'appelant.
  Exemple: optimisation de la récursivité terminale.