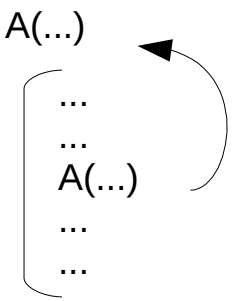
III – La récursivité

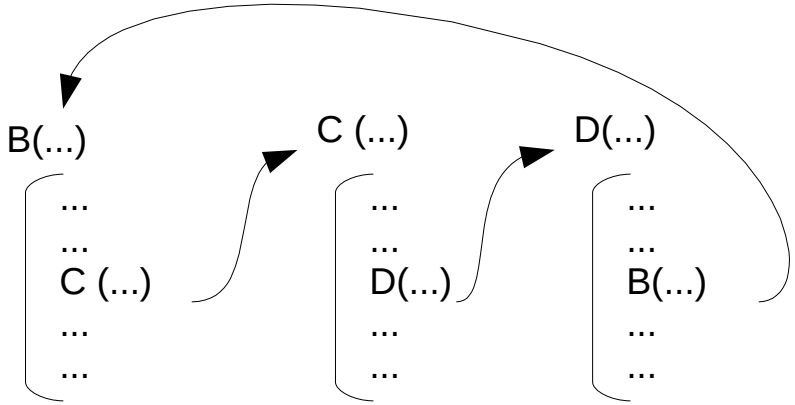
Définition,
Conception de solutions
et
Mise en oeuvre

Définition

- Une procédure ou une fonction est dite récursive si elle fait appel à elle même, directement ou indirectement



Récursivité directe



Récursivité indirecte

- Appel récursif --> Boucle
  - Pour éviter les boucles infinies, les appels récursifs doivent être conditionnels (à l'intérieur d'un SI ou un SINON ou un TQ ...etc)

Exemple de la fonction factorielle

n! = n * (n-1) * (n-2) * .... * 1 pour n > 0 et 0! = 1

n! = n * (n-1)! pour n > 0 et n! = 1 pour n = 0

```
Fact( n:entier ) : entier
   SI  n=0
       Fact := 1                      /* cas particulier */
   SINON
       Fact := n * Fact( n-1 )        /* cas général */
   FSI
```

Déroulement pour n=4 :

1- 4! = 4 * 3!
2- 3! = 3 * 2!
3- 2! = 2 * 1!
4- 1! = 1 * 0!
5- 0! = 1 (cas particulier)

l'exécution i attend la terminaison de l'exécution i+1 pour continuer son traitement

Exemple de la fonction Fibonacci

(attention! solution très inefficace mais simple)

$U_0 = 1,\ U_1 = 1,\ U_n = U_{n-1} + U_{n-2}$ pour $n > 1$

```
Fib( n:entier ) : entier
    SI n < 2
        Fib := 1
    SINON
        Fib := Fib( n-1 )  +  Fib( n-2 )
    FSI
```

Déroulement :

Fib(4) = Fib(3) + Fib(2) = 5

= Fib(2) + Fib(1)

= Fib(1) + Fib(0) 1

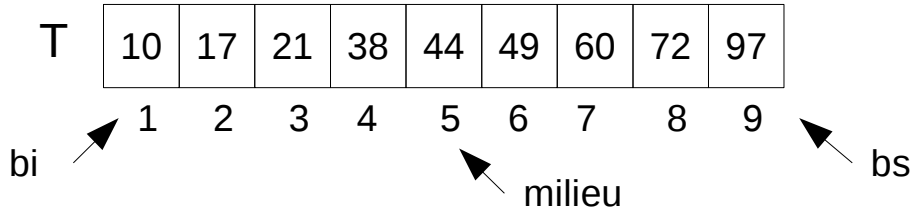
1 1

= Fib(1) + Fib(0)

1 1

Exemple de la recherche dichotomique

- recherche binaire dans une table ordonnée -



un élément x se trouve soit dans la 1ere moitié soit dans la 2e moitié
pour le savoir en compare par rapport au milieu

```
Rech( x:Tqlq; bi, bs : entier ) : entier  /* l'indice de x dans T. ou bien -1  */
   /* bi : borne inf.   bs : borne sup.  */
   SI (bi > bs)  Rech := -1                          /* cas particulier */
   SINON
        milieu := (bi+bs) div 2;
        SI (x = T [ milieu ] )   Rech := milieu  /* cas particulier */
        SINON
             SI  (x < T [milieu] )
                  Rech( x , bi , milieu-1 )        /* 1ere moitié */
             SINON
                  Rech( x , milieu+1 , bs )        /* 2e moitié */
             FSI
        FSI
   FSI
```

Remarques

- Dans un module récursif (procédure ou fonction) les paramètres doivent être clairement spécifiés
- Dans le corps du module il doit y avoir:
  - un ou plusieurs cas particuliers
    - ce sont les cas simples (wayouts) qui ne nécessitent pas d'appels récursifs
  - un ou plusieurs cas généraux
    - ce sont les cas complexes qui sont résolus par des appels récursifs
- L'appel récursif d'un cas général doit toujours mener vers un des cas particuliers

Conception de solutions récursives

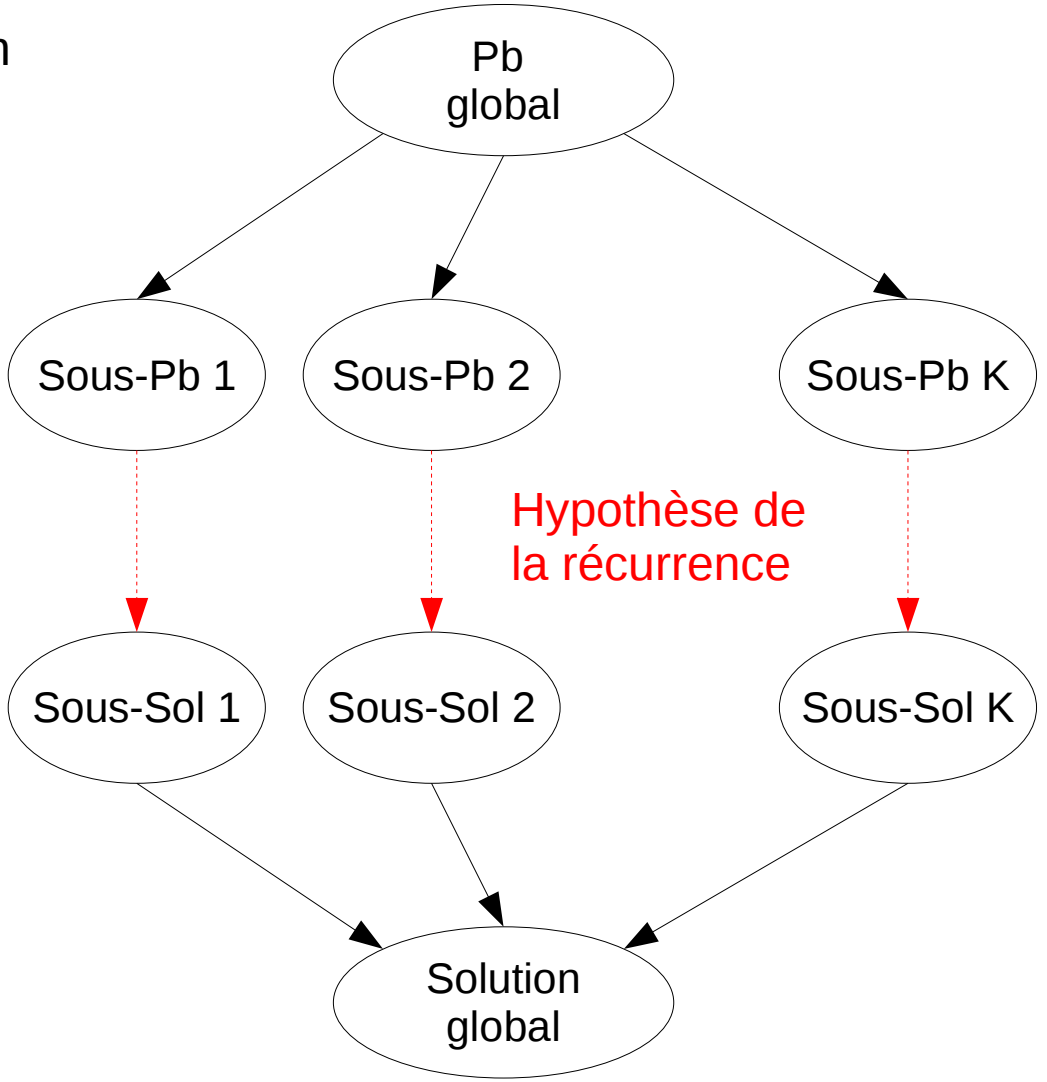
- La récursivité est un outils puissant
  - permet de concevoir des solutions (simples) sans se préoccuper des détails algorithmiques internes
- Approche par « décompositions »
  - spécifier la solution du pb en fonction de la (ou des) solution(s) d'un (ou de plusieurs) sous-pb plus simple(s).
  - comment trouver la solution d'un sous-pb ?
    - ce n'est pas important car on prendra comme hypothèse que chaque appel récursif résoudra un sous-pb
    - si on arrive à trouver une solution globale en utilisant ces hypothèses, alors ces dernières (hypothèses) sont forcément correctes, de même que la solution globale. Cela ressemble à la « démonstration par récurrence »

Schéma général d'une décomposition récursive

Décomposition du pb en K sous-Pb

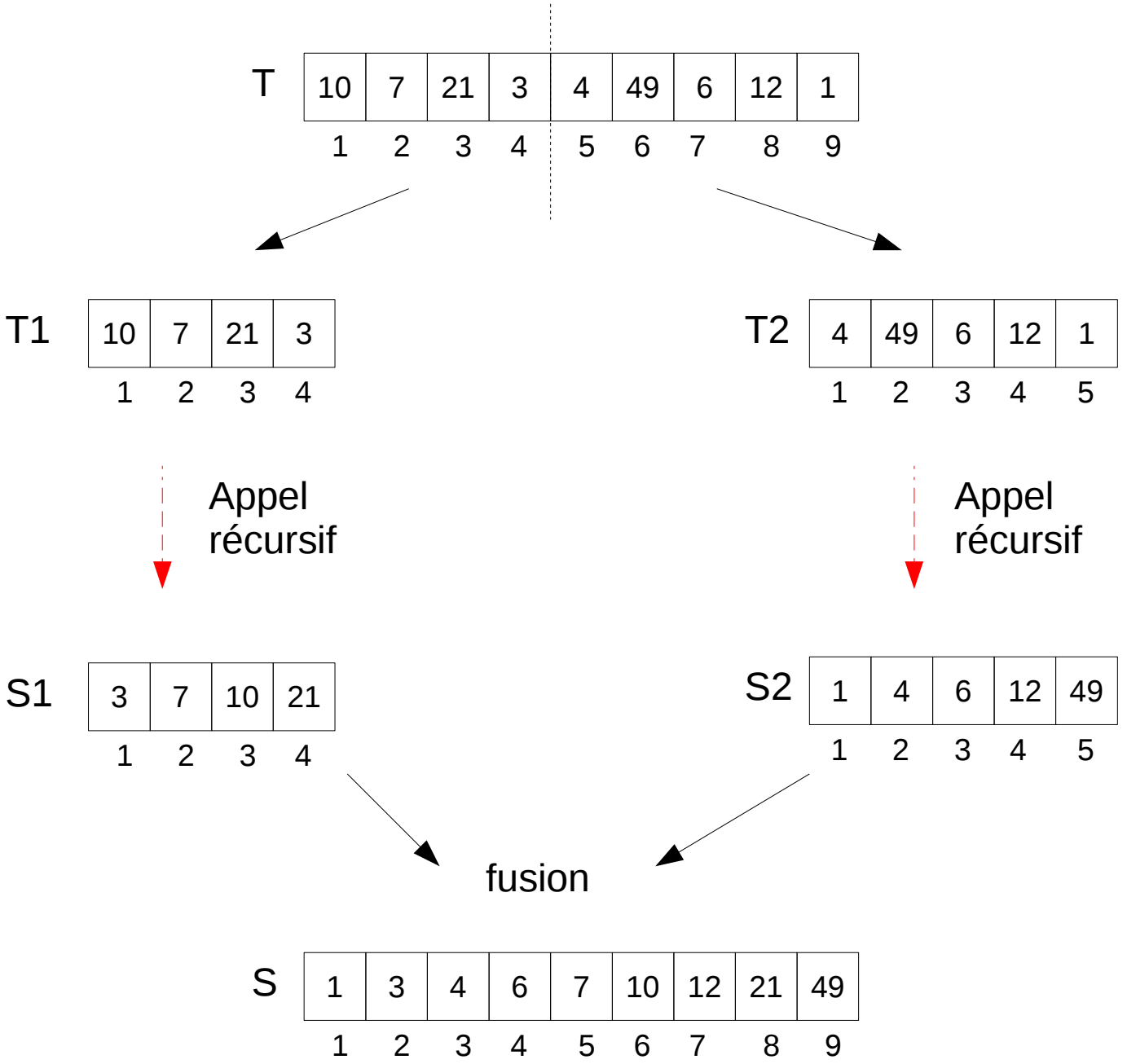
les appels récursifs permettent de passer des sous-pb aux sous-sol

Combinaison des solutions obtenues



En réalité, chaque sous-pb sera résolu par la même décomposition récursive... jusqu'à l'obtention de sous-pb triviaux.

Exemple du tri d'un tableau



Algorithme du tri par fusion

Tri_Fusion(T:Tab; n:entier; var S:Tab)

```
SI (n = 1 )   S[1] := T[1]

SINON
   /* récuperer les 2 moitiés T1 et T2 */
   POUR i=1,n
      SI ( i <= n/2 )  T1[ i ] := T [ i ]   SINON  T2[ i-n/2 ] := T[ i ]
   FP;
   n1 := n/2;   n2 := n - n1;

   /* Trier T1 et T2 en utilisant des appels récursifs */
   Tri_Fusion( T1, n1, S1 );
   Tri_Fusion( T2, n2 , S2);

   /* Fusionner les deux sous-solutions S1 et S2 */
   Fusion( S1, n1, S2, n2, S );

FSI
```

Exemple des tours de Hanoi

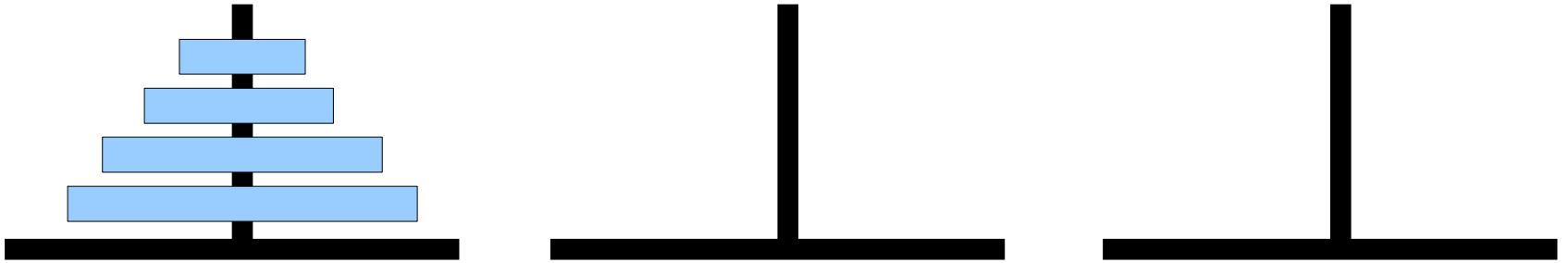
Déplacer n disques (empilés les uns sur les autres) d'un piquet (A) vers un autre piquet (B) en utilisant un troisième (C) comme intermédiaire (ce sont des piles, on ne peut accéder qu'au disque du sommet)

un disque ne peut jamais être placé au dessus d'un autre plus petit

piquet A

piquet B

piquet C

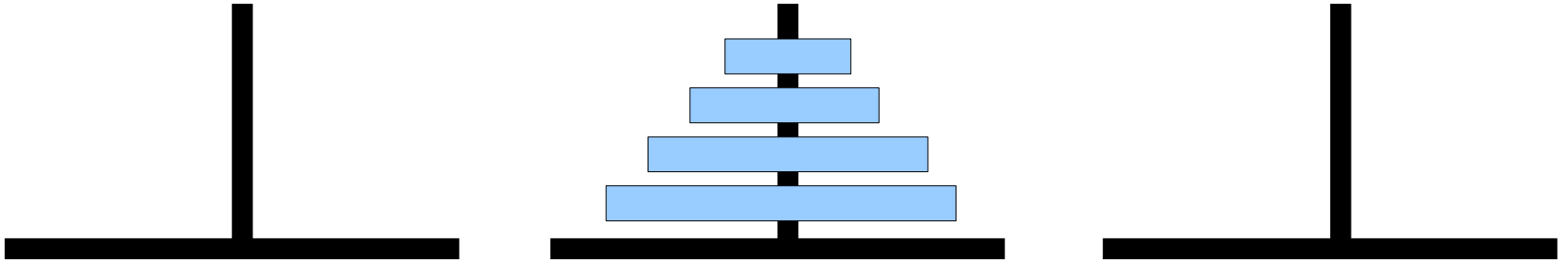


état initial

piquet A

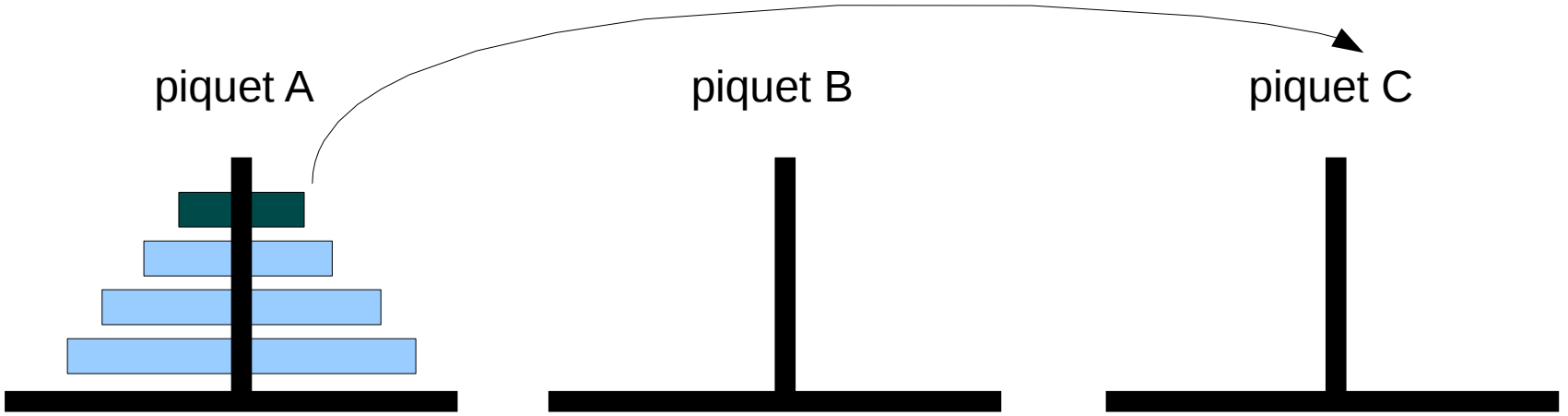
piquet B

piquet C

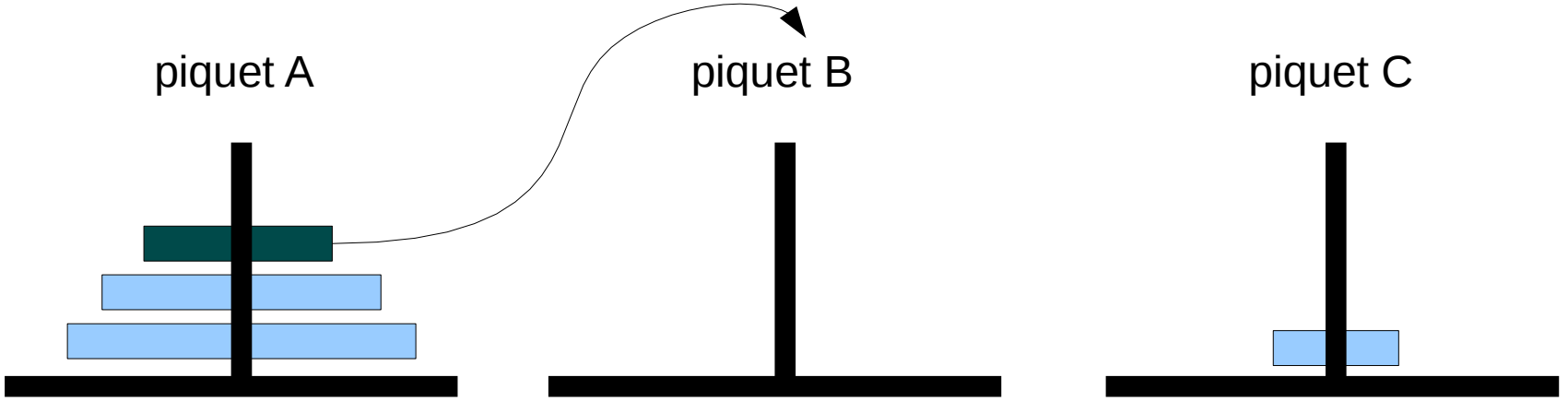


état final

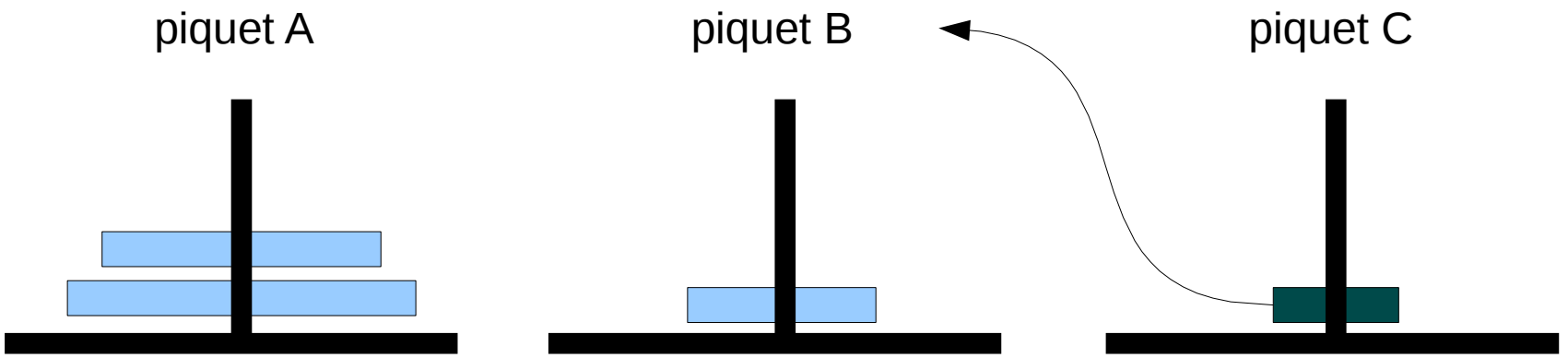
Déroulement pour n = 4



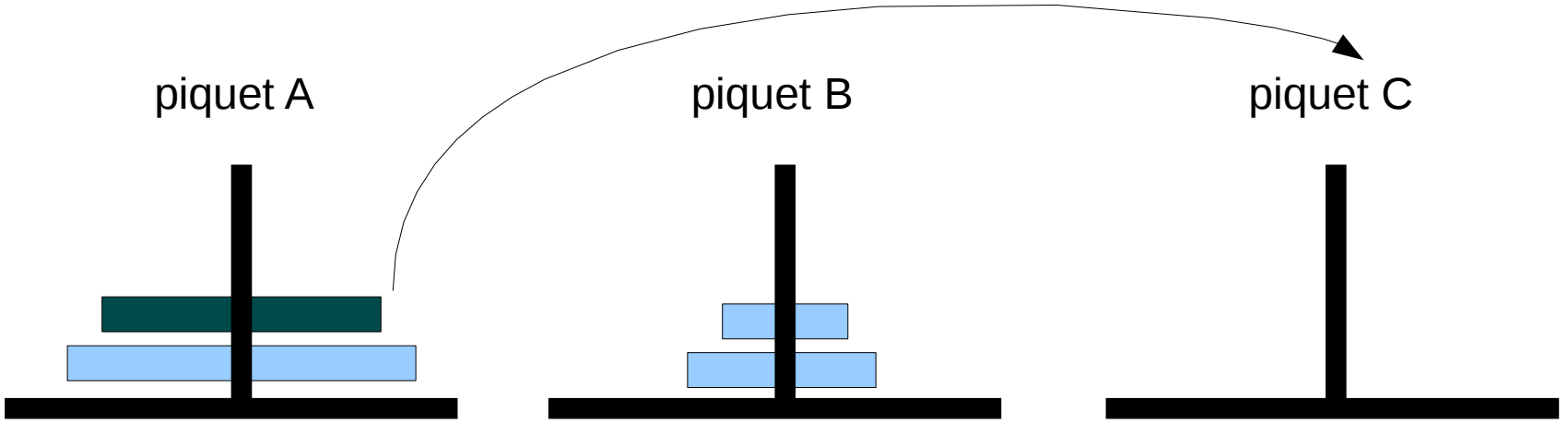
Déroulement pour n = 4



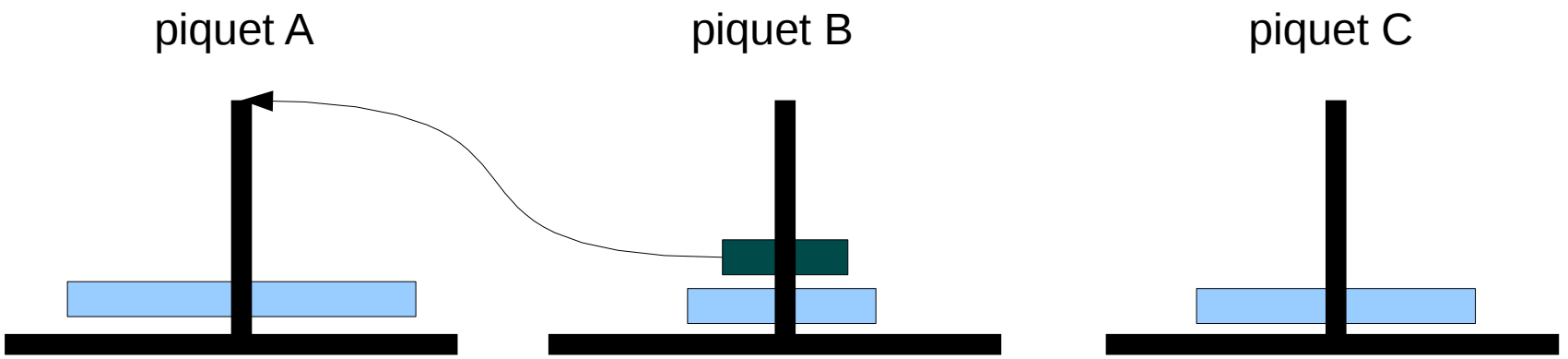
Déroulement pour n = 4



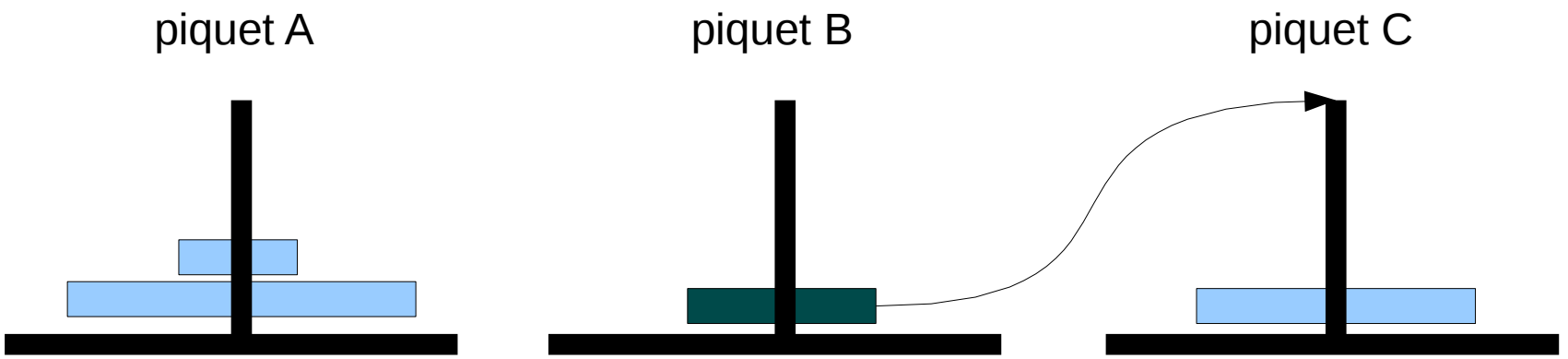
Déroulement pour n = 4



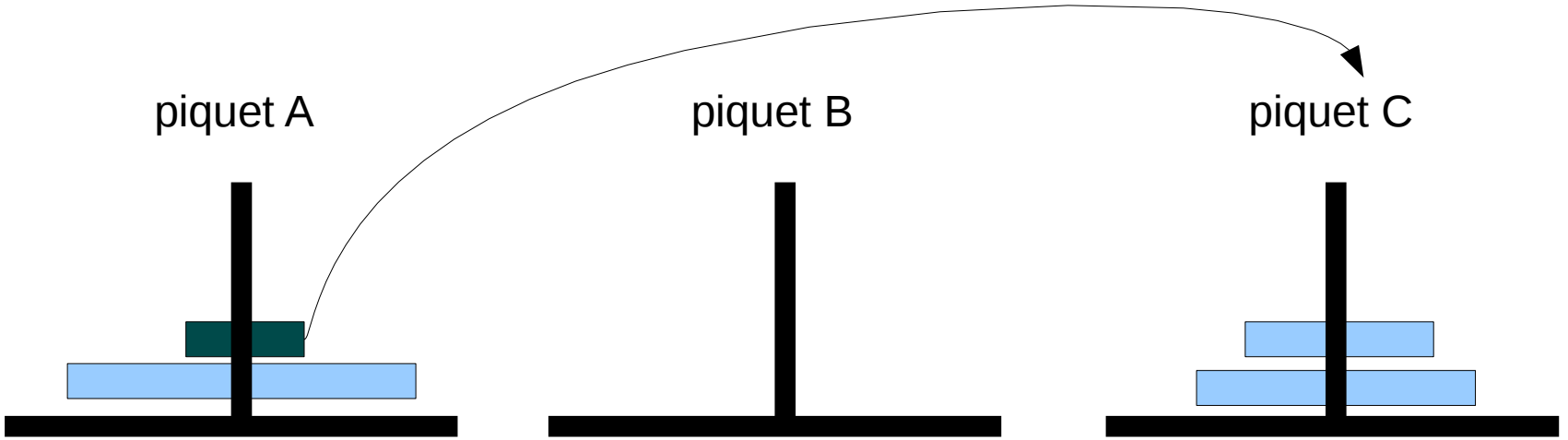
Déroulement pour n = 4



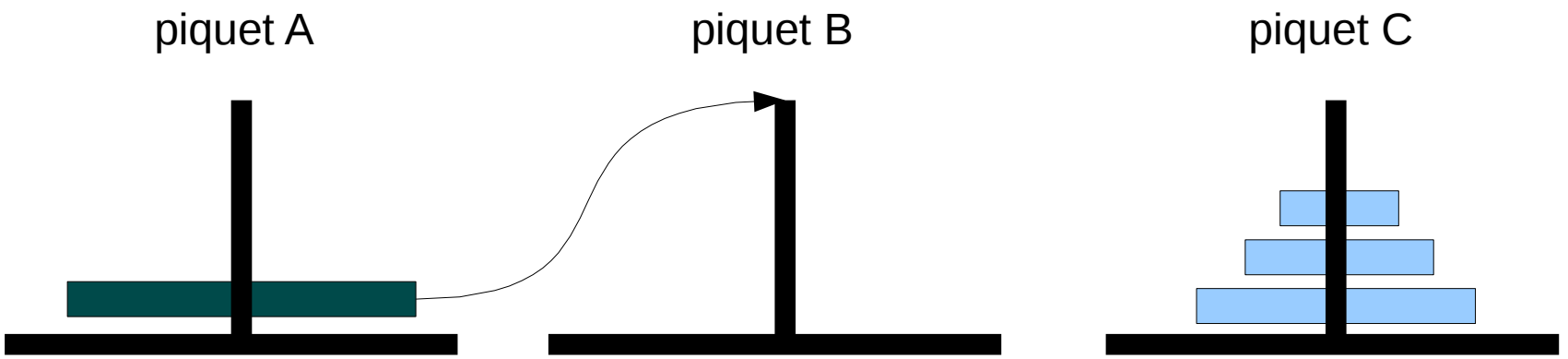
Déroulement pour n = 4



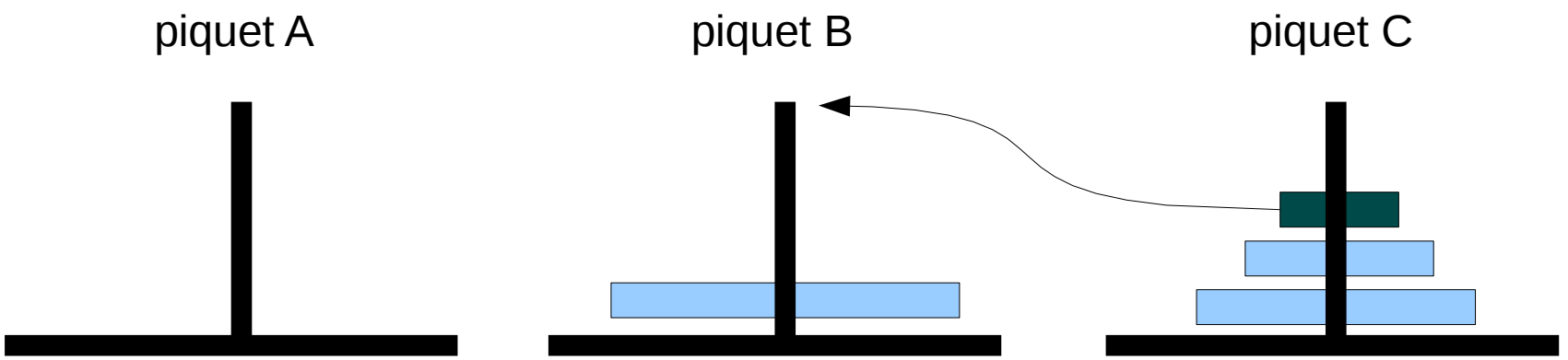
Déroulement pour n = 4



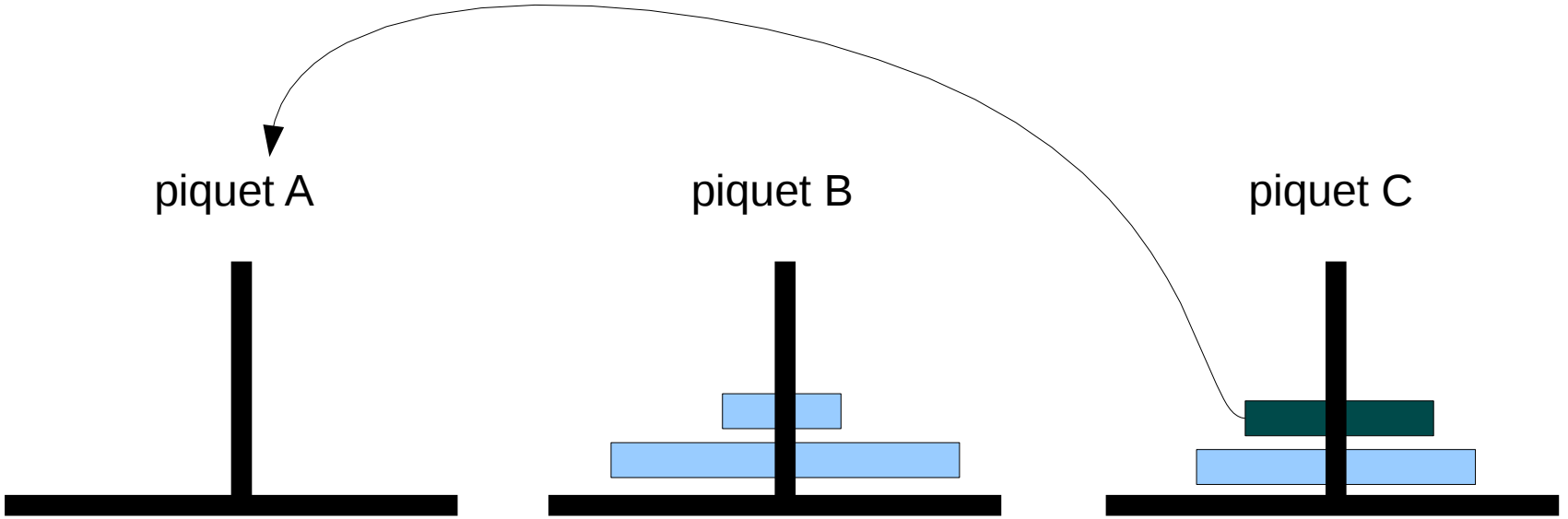
Déroulement pour n = 4



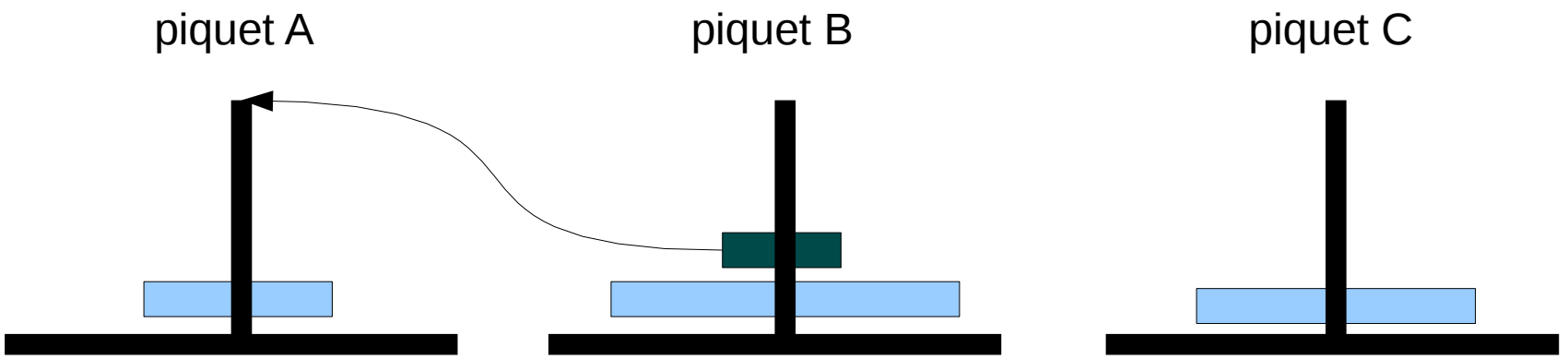
Déroulement pour n = 4



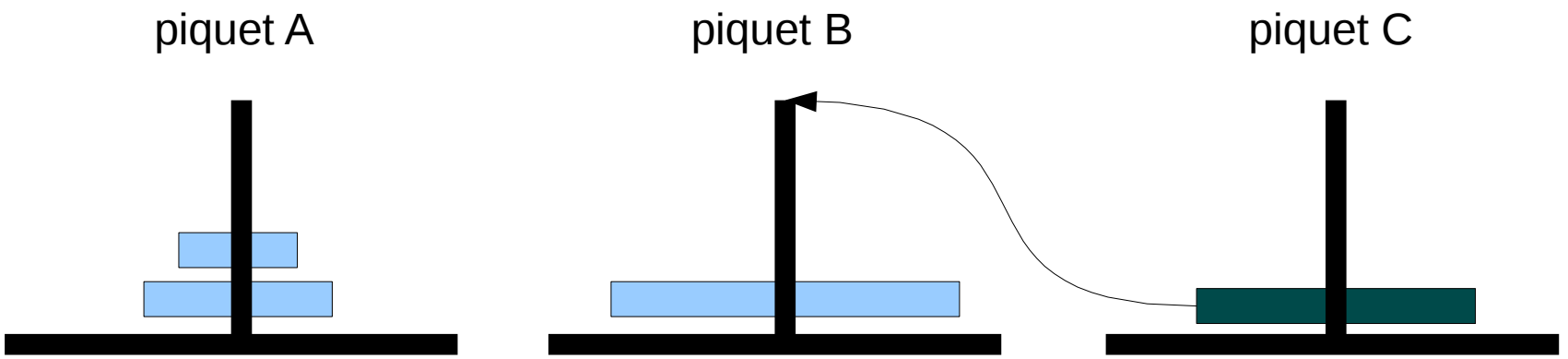
Déroulement pour n = 4



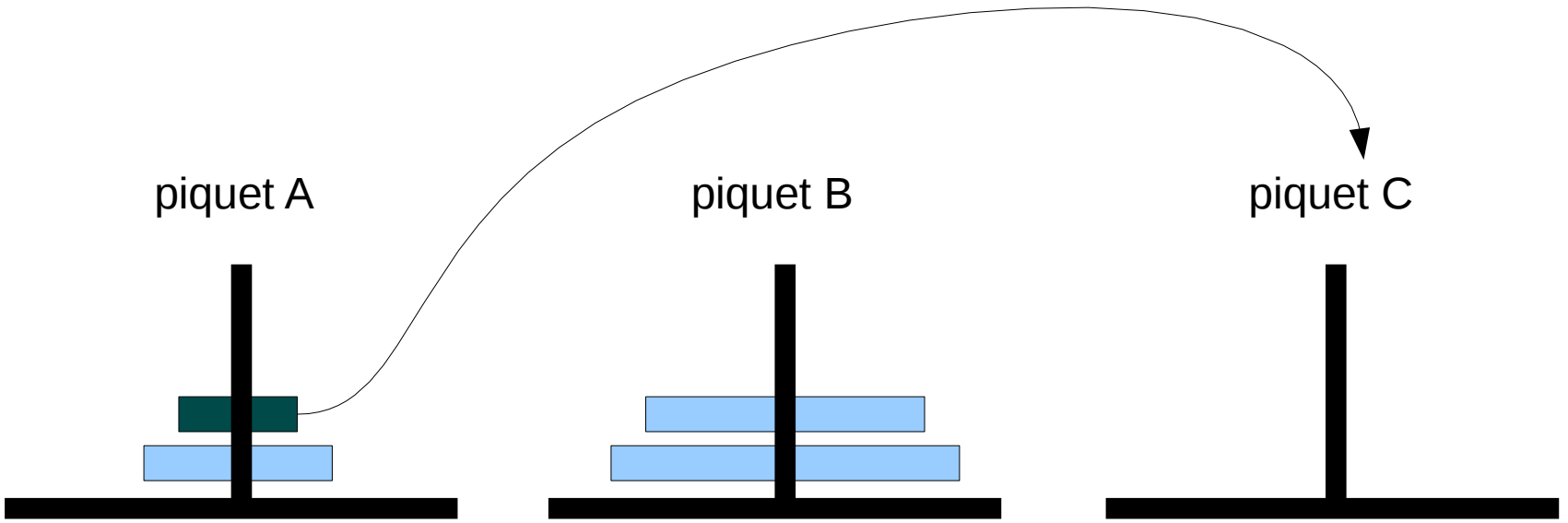
Déroulement pour n = 4



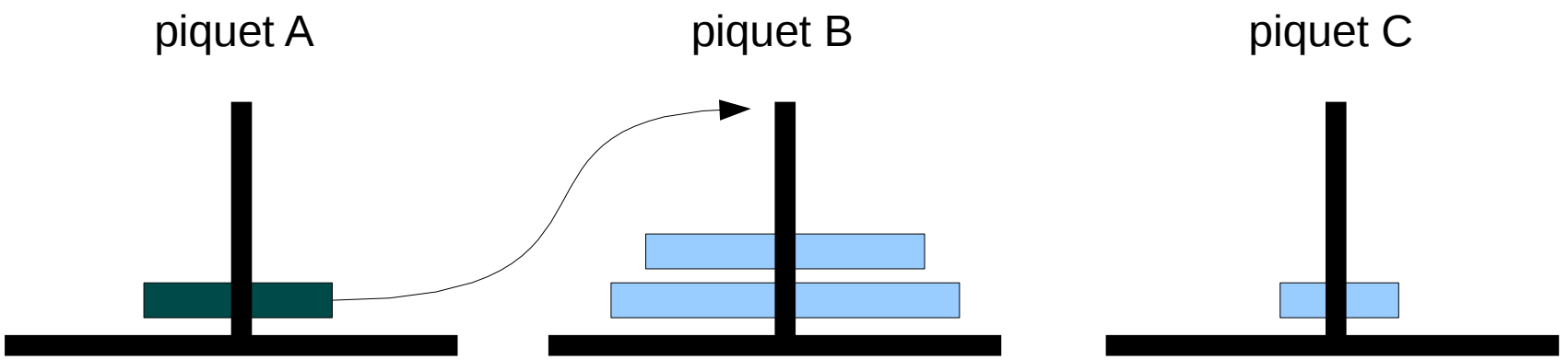
Déroulement pour n = 4



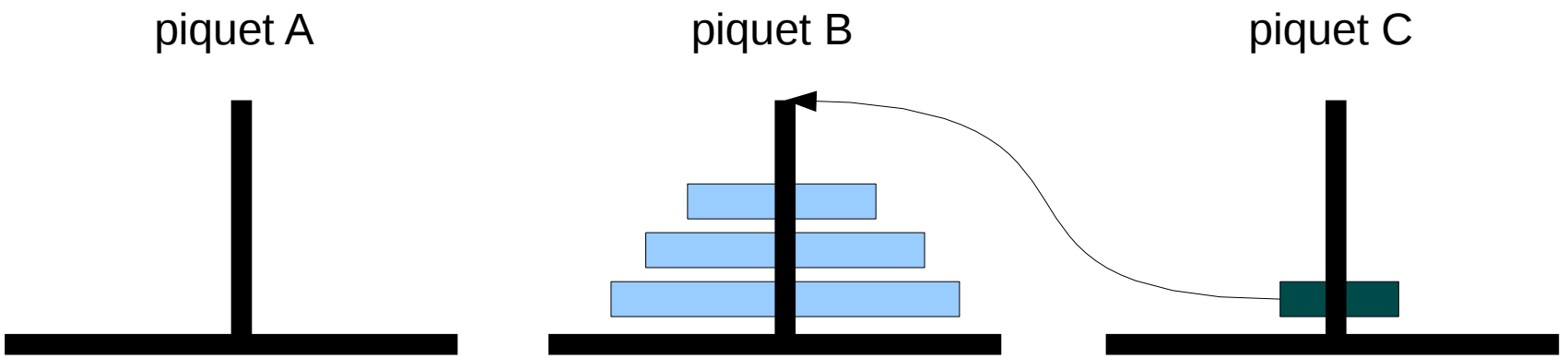
Déroulement pour n = 4



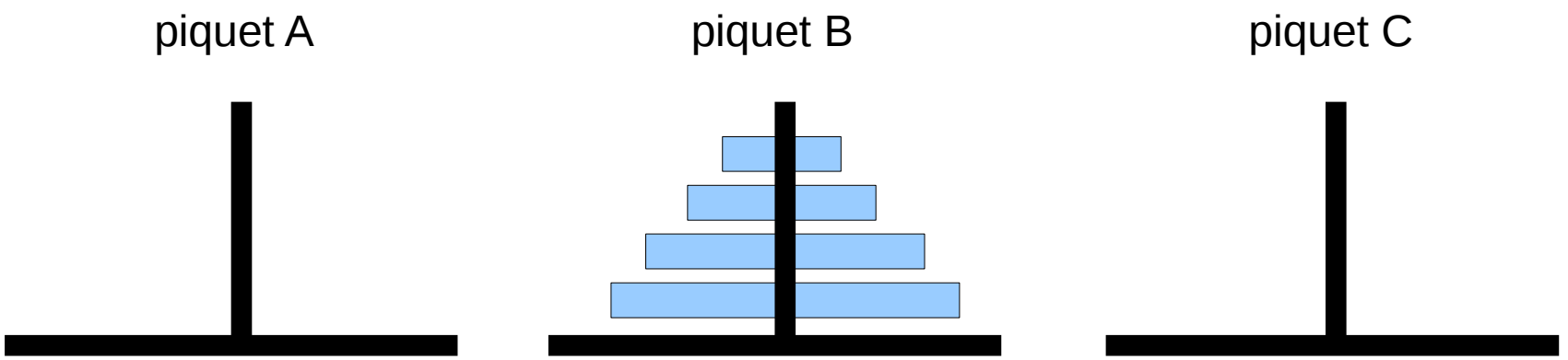
Déroulement pour n = 4



Déroulement pour n = 4



Déroulement pour n = 4

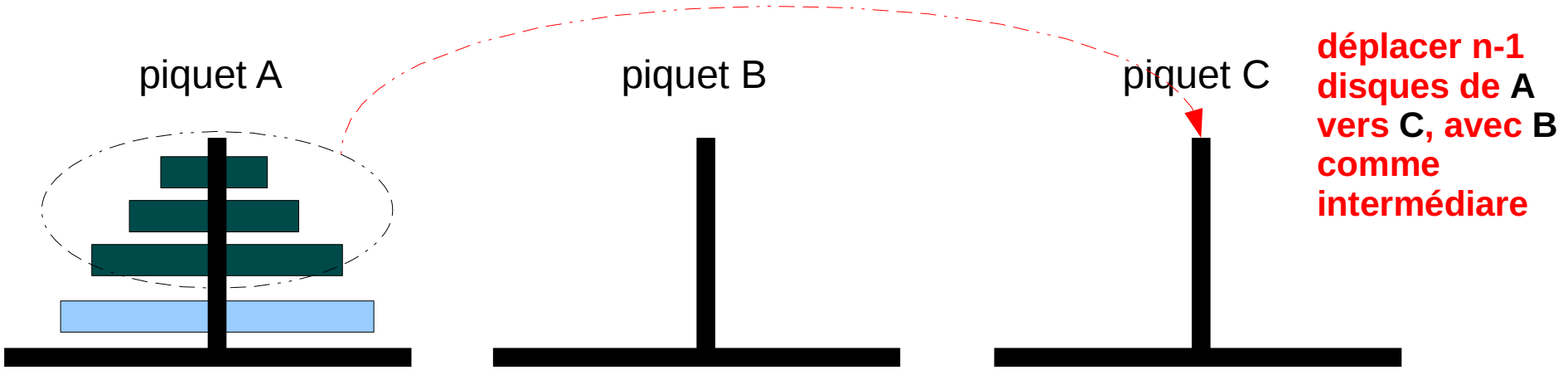


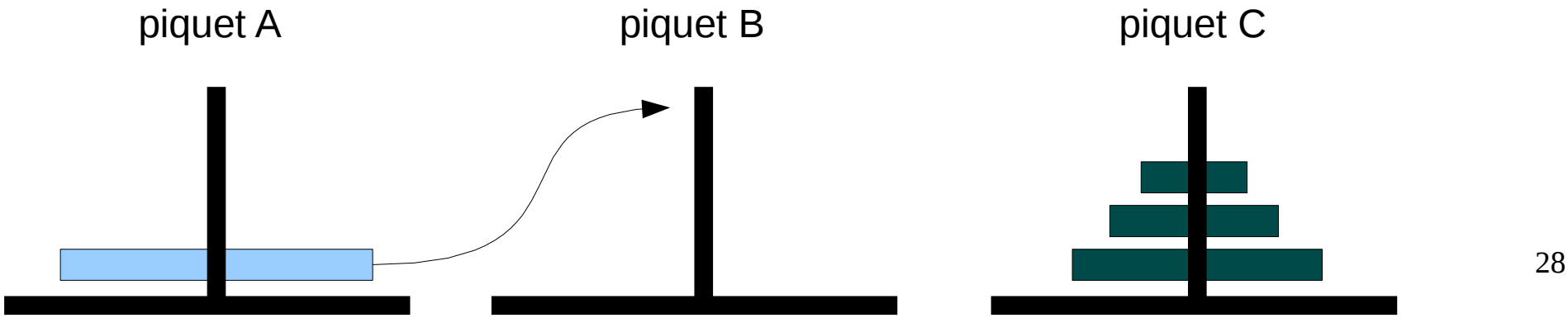
Conception de la solution récursive

Il faut pouvoir exprimer :

« le déplacement de n disques » en fonction d'un ou de plusieurs « déplacement de m disques » (avec m < n).

Par exemple (m=n-1) si on avait une solution pour déplacer n-1 disques, comment l'utiliser pour déplacer n disques ?



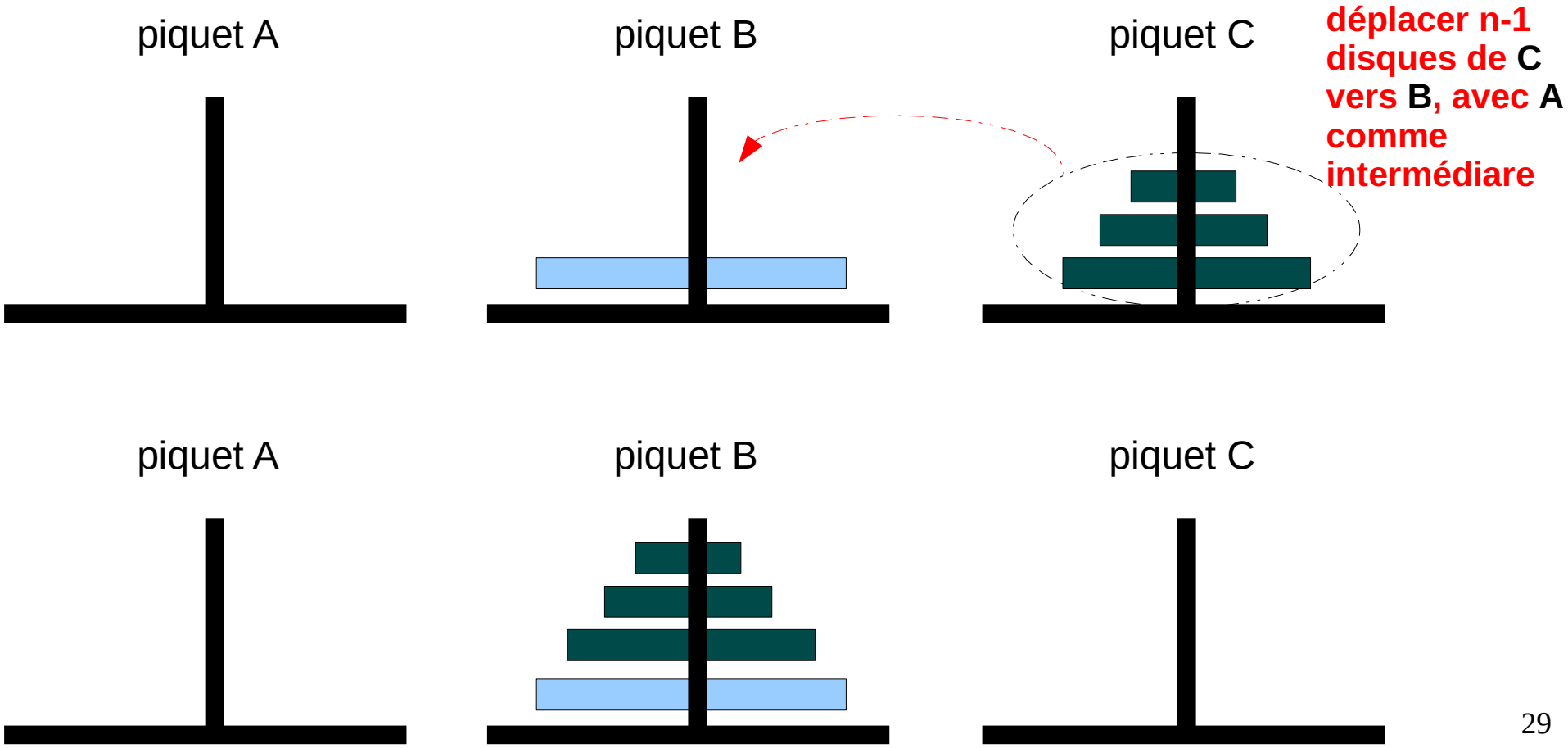


Conception de la solution récursive

Il faut pouvoir exprimer :

« le déplacement de n disques » en fonction d'un ou de plusieurs « déplacement de m disques » (avec m < n).

Par exemple (m=n-1) si on avait une solution pour déplacer n-1 disques, comment l'utiliser pour déplacer n disques ?



Algorithme informel

Pour Transférer **n** disques de **X** vers **Y** avec **Z** comme intermédiaire, il faut :

Cas général (n > 1)

Transférer les **n-1** premiers disques de **X** vers **Z**, en utilisant **Y** comme intermédiaire

déplacer l'unique disque qui reste dans **X** vers **Y**

Transférer les **n-1** premiers disques de **Z** vers **Y**, en utilisant **X** comme intermédiaire

Cas particulier (n = 1)

déplacer le disque de **X** vers **Y**.

Algorithme Final

Hanoi(n:entier; X,Y,Z : caractères)

SI (n = 1)
écrire(« déplacer un disque de », X, « vers », Y);

SINON

Hanoi (n-1, X, Z, Y);

écrire(« déplacer un disque de », X, « vers », Y);

Hanoi(n-1, Z, Y, X);

FSI

une autre version

Hanoi(n:entier; X,Y,Z : caractères)

SI (n > 0)

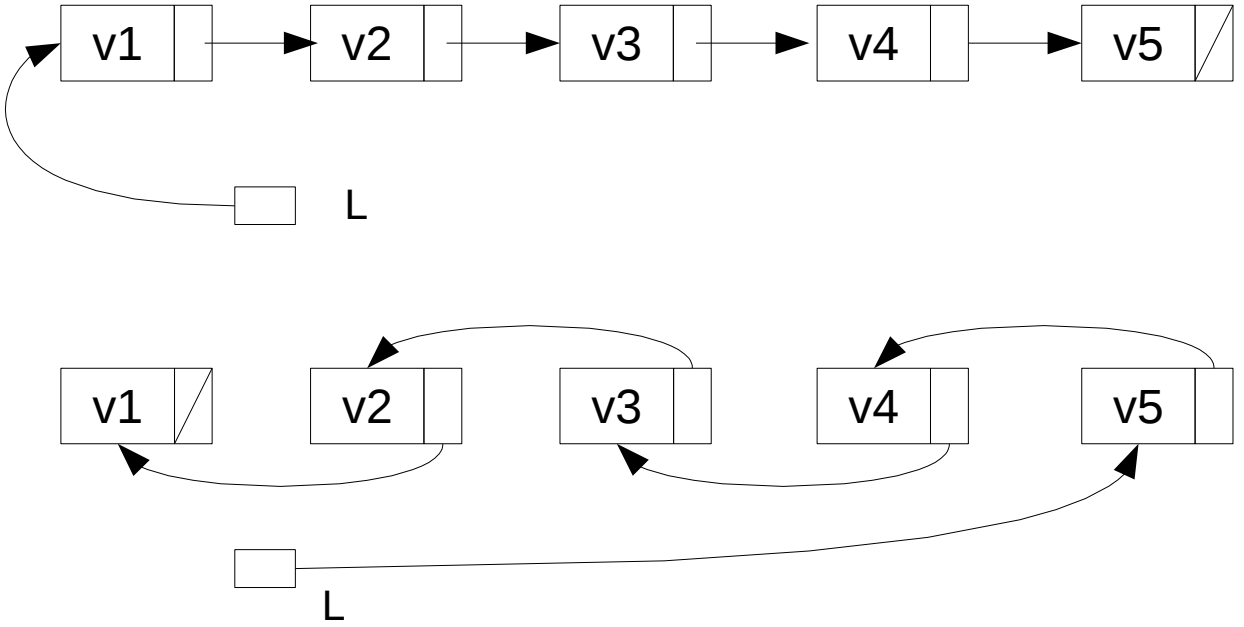
Hanoi (n-1, X, Z, Y);

écrire(« déplacer un disque de », X, « vers », Y);

Hanoi(n-1, Z, Y, X);

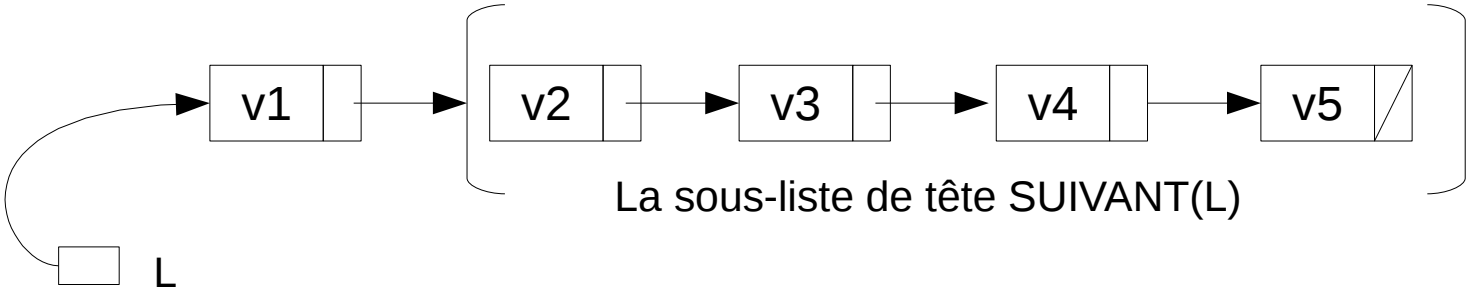
FSI

Exemple de l'inversion d'une liste



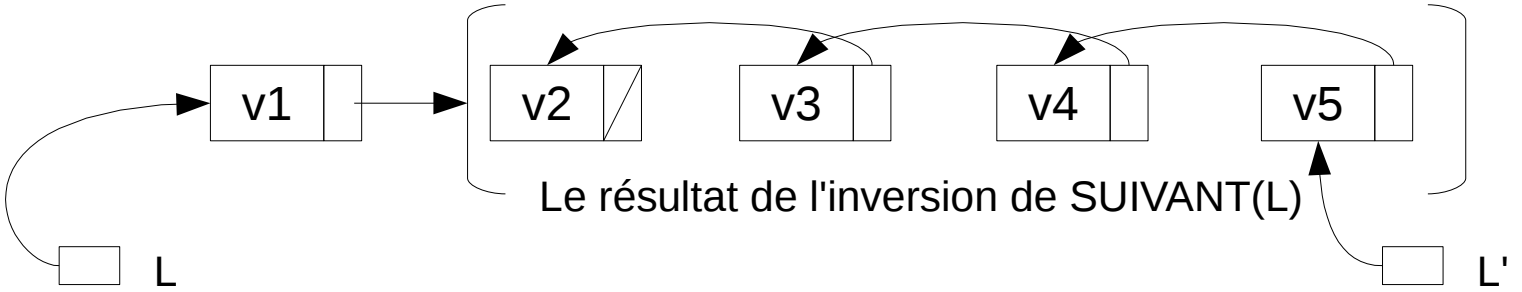
Comment exprimer le problème de l'inversion d'une liste en termes d'inversion d'une ou de plusieurs sous-listes ?

une solution avec un seul appel récursif



Décomposant le problème en un seul sous-problème:
pour inverser la liste L, il faut d'abord inverser la sous-liste de tête Suivant(L) (par un appel récursif)

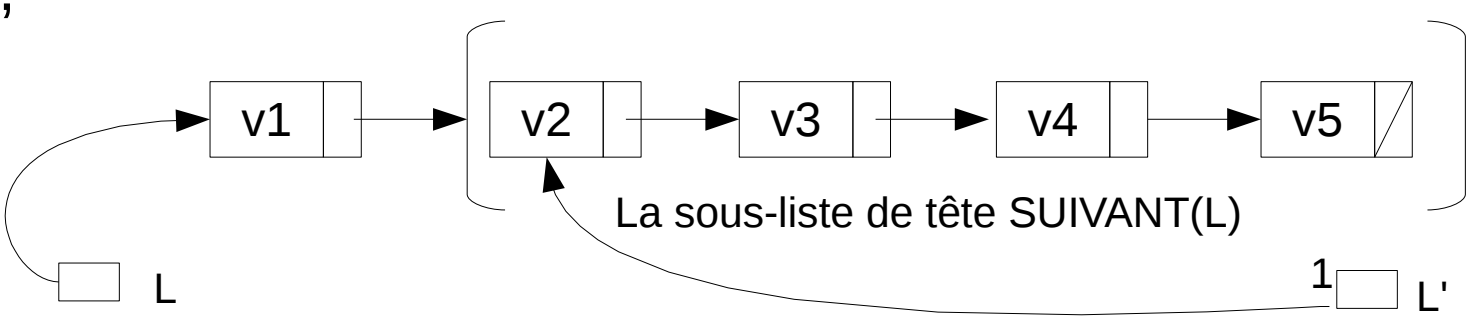
Soit L' le résultat de l'inversion de la sous-liste Suivant(L), qu'est-ce qu'il faut faire encore pour compléter l'inversion de L ?



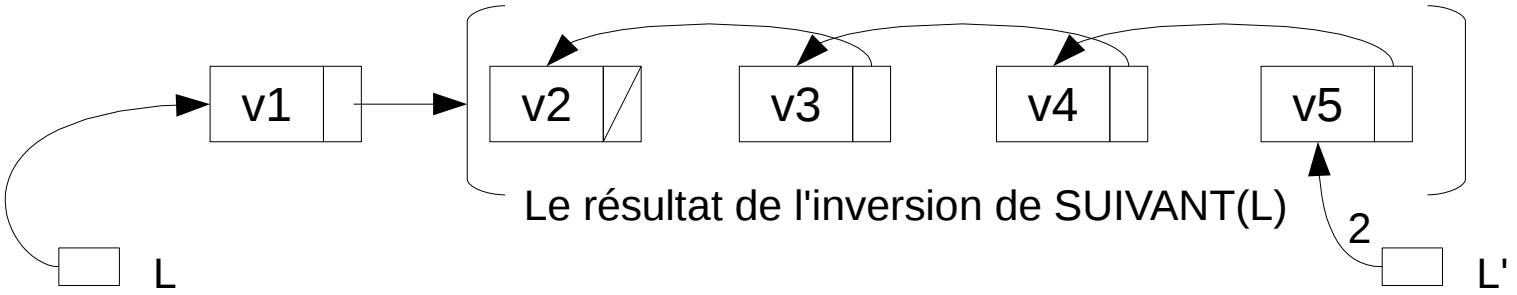
Inv(var L)

...

1 L' := Suivant(L);



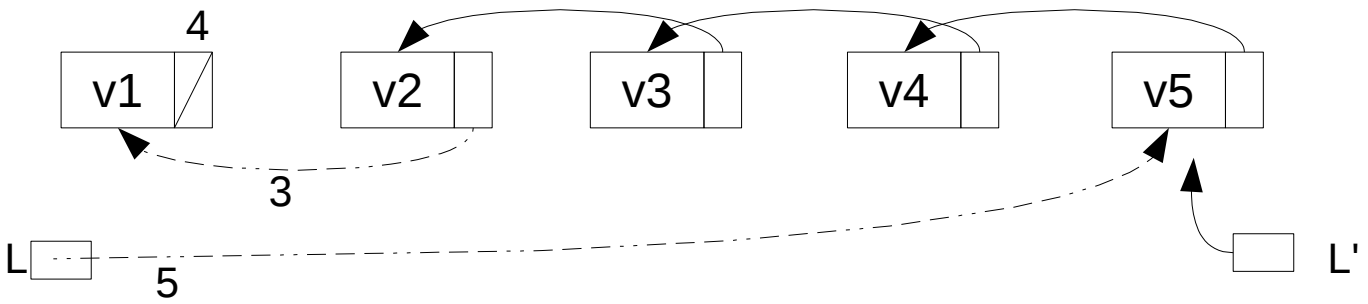
2 **Inv(L')**;



3 Aff-adr(Suivant(L) , L);

4 Aff-adr(L , NIL);

5 L := L'



Algorithme final (avec cas particuliers)

```
Inv( var L : ptr )

    SI ( L <> NIL )

        SI ( Suivant(L) <> NIL )

            L' := Suivant(L);
            Inv( L' );

            Aff-adr( Suivant(L) , L );
            Aff-adr( L , NIL );
            L := L'

        FSI

    FSI
```

Mise en oeuvre de la récursivité

- Comment dérouler (en détail) un algorithme récursif
  - Gestion des Zones de Données
- Avoir une méthode d'élimination des appels récursifs
  - Transformer (automatiquement) un algorithme récursif en un algorithme non récursif

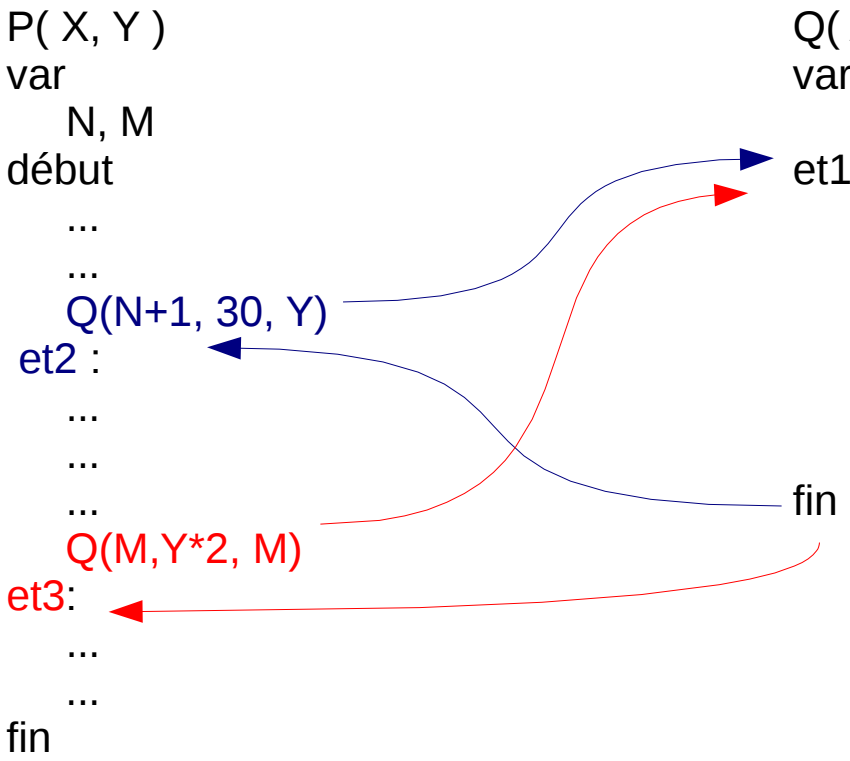
Notions sur les appels de fonctions et de procédures

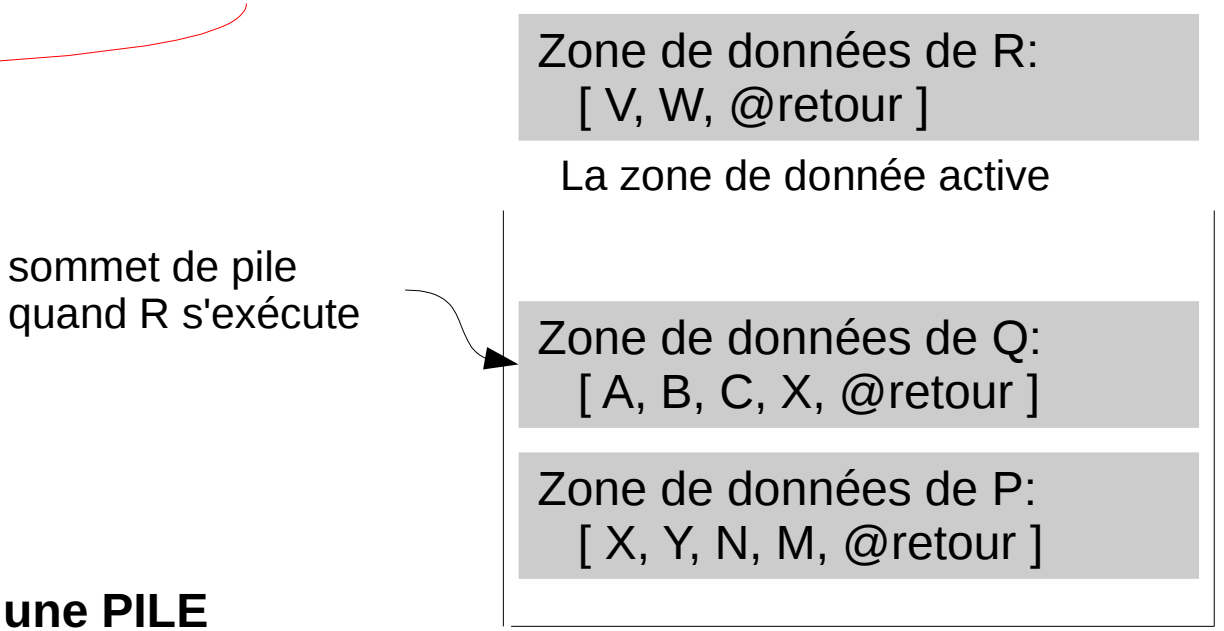
Lors de l'exécution d'un programme, le code (instructions) est chargé en MC.

Chaque Instruction est donc associée à une adresse qu'on appellera par la suite : étiquette

Lorsqu'une procédure ou fonction est entrain de s'exécuter, une zone en mémoire lui est allouée renfermant ses variables locales ==> c'est la Zone de Données

Lorsqu'une procédure ou fonction se termine, sa zone de données est détruite (libérée) et l'exécution continue dans le module appelant, juste après l'instruction qui a fait l'appel (point de retour)





L'imbrication des appels
==> Gestion des ZDD par une PILE

- A chaque appel il y a Empilement
- A chaque fin il y a Dépilement

Elimination des appels récursifs
(cas d'une fonction ou procédure sans paramètres de sortie)

- Définir la ZDD (var locales + param d'appel + adr de retour)
- Associer une étiquette après chaque appel
  - il y a toujour un appel externe
- **Transformation d'un appel**
  - Empiler la ZDD courante
  - Préparer la nouvelle ZDD
    - initialiser les paramètres d'appel et l'adr de retour
  - Aller au début de la fonction/procédure
- **A la fin de la fonction/procédure**
  - Récupérer l'adr de retour dans un temporaire tmp
  - Dépiler une ZDD
  - Aller à tmp

Exemple de transformation

```
Fact( n:entier ) : entier
        var
            X,Y : entier
   et1:début
            SI  (n=0)
                Fact := 1
            SINON
                X := n-1;
   et2:         Y := Fact(X);
                Fact : = n * Y;
            FSI
        fin
```

Ex. de Prog. appelant:

```
        ...
   et0:écrire( Fact(4) );
        ...
```

Déclarations pour la transformation:

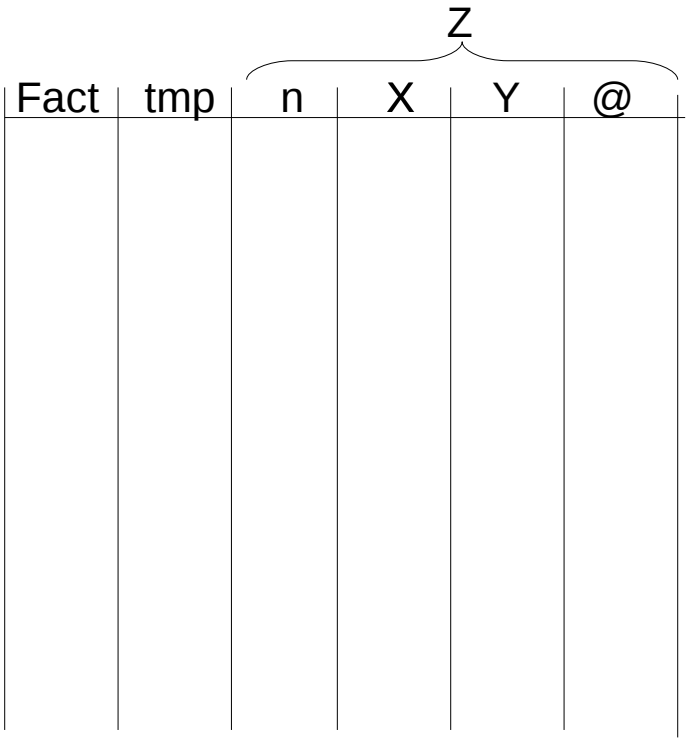
```
type
    Tzdd = struct
        n,X,Y : entier;
        @ : étiquette
    fin;

var
    Z : Tzdd; /* ZDD active */
    P : Pile de Tzdd;
    Fact : entier  /* résultat */
    tmp : étiquette;
```

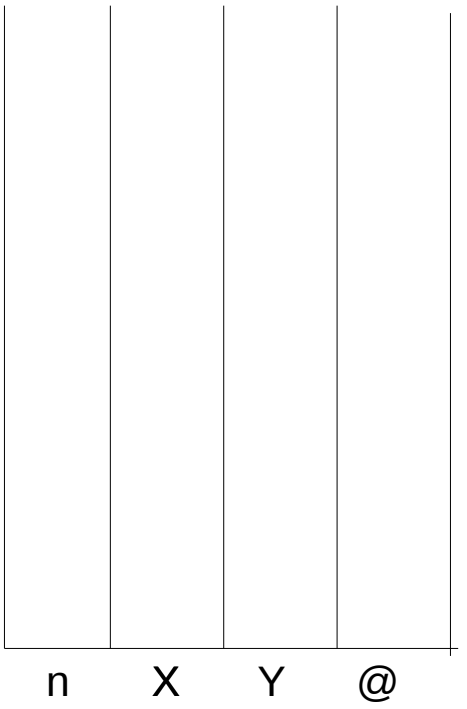
```
Prog. appelant:
        ...
   et0:écrire( Fact(4) );
        ...
```

```
Fact( n:entier ) : entier
        var
            X,Y : entier
   et1:début
            SI  (n=0)
                Fact := 1
            SINON
                X := n-1;
   et2:         Y := Fact(X);
                Fact : = n * Y;
            FSI
     fin
```

```
        CreerPile(P);
        Empiler(P,Z);
        Z.n := 4;   Z.@ := et0;
        Aller à et1
   et0: écrire( Fact );
        stop.

/* Traduction de la fonction */
   et1:SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
   et2:     Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
        FSI
        tmp := Z.@;
        Depiler(P,Z);
        Aller à tmp
```

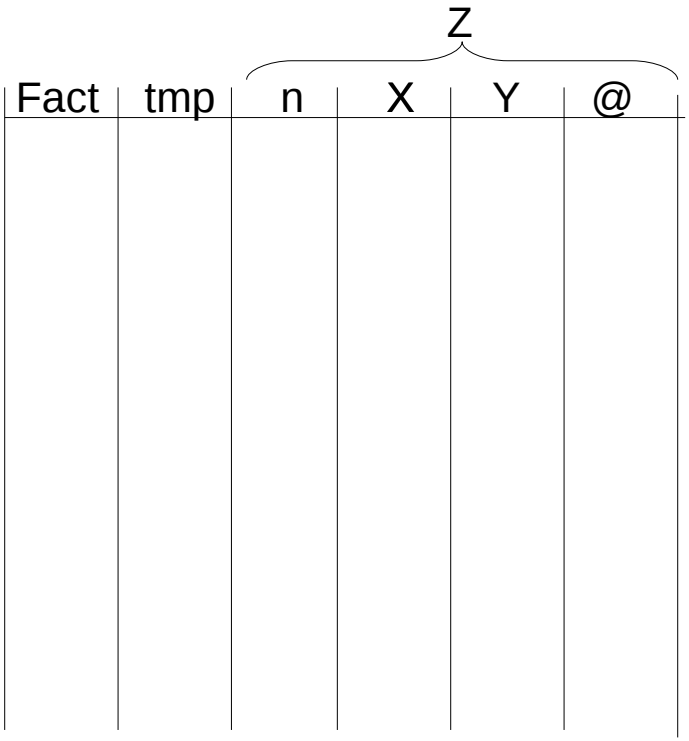
```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
       SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

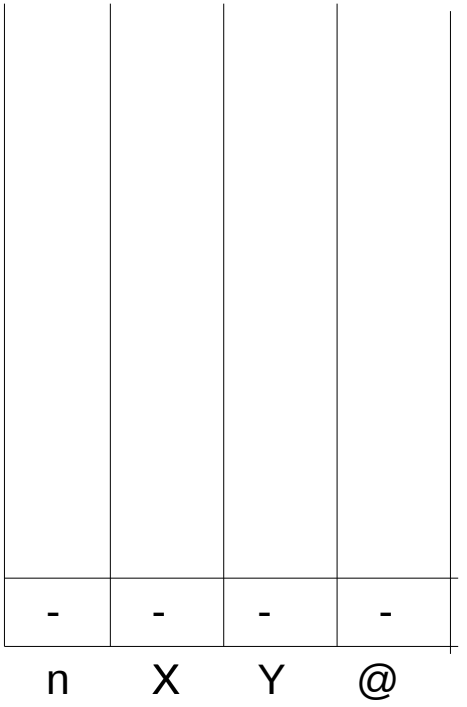




la pile P

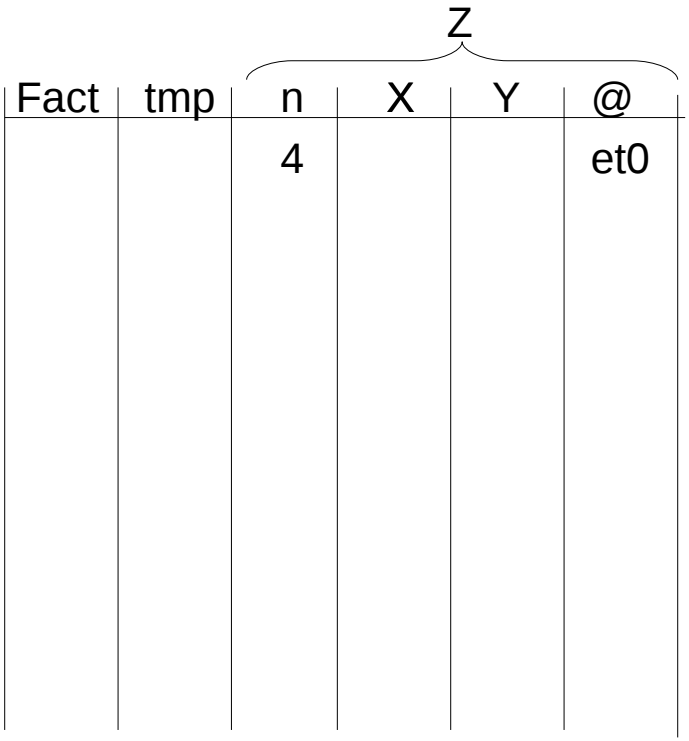
```
        CreerPile(P);
        Empiler(P,Z);
        Z.n := 4;   Z.@ := et0;
        Aller à et1
   et0: écrire( Fact );
        stop.

/* Traduction de la fonction */
   et1:SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
   et2:     Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
        FSI
        tmp := Z.@;
        Depiler(P,Z);
        Aller à tmp
```

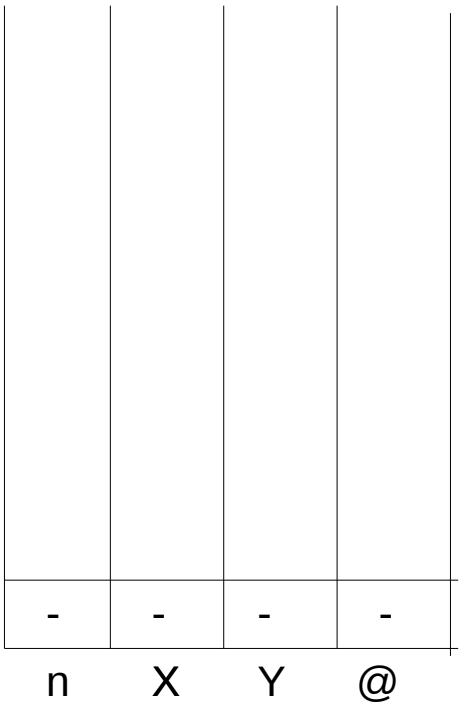




la pile P

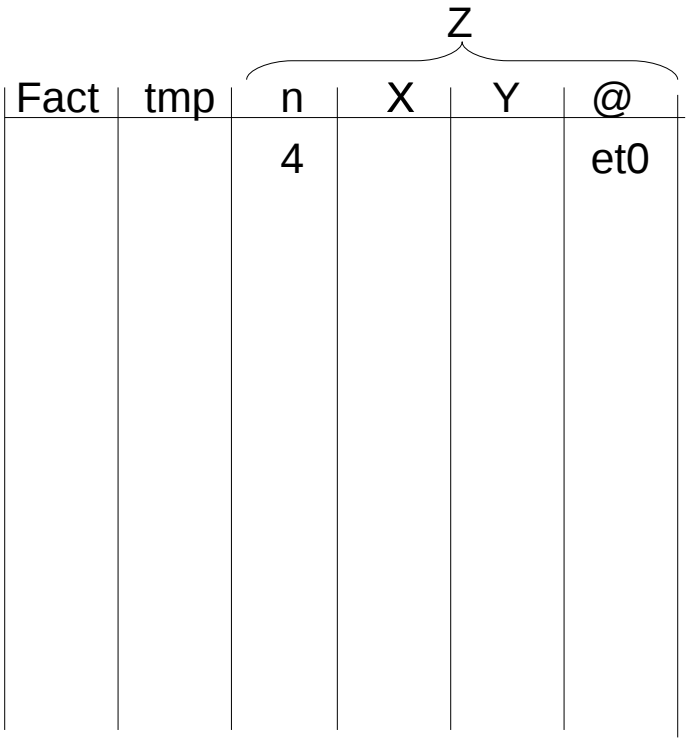
```
        CreerPile(P);
        Empiler(P,Z);
        Z.n := 4;   Z.@ := et0;
        Aller à et1
  et0: écrire( Fact );
        stop.

/* Traduction de la fonction */
    et1:SI  (Z.n=0)  Fact := 1
         SINON
              Z.X := Z.n - 1;
              Empiler(P,Z);
              Z.n := Z.X;
              Z.@ := et2;
              Aller à et1;
    et2:      Z.Y := Fact ;
              Fact : = Z.n * Z.Y;
        FSI
        tmp := Z.@;
        Depiler(P,Z);
        Aller à tmp
```

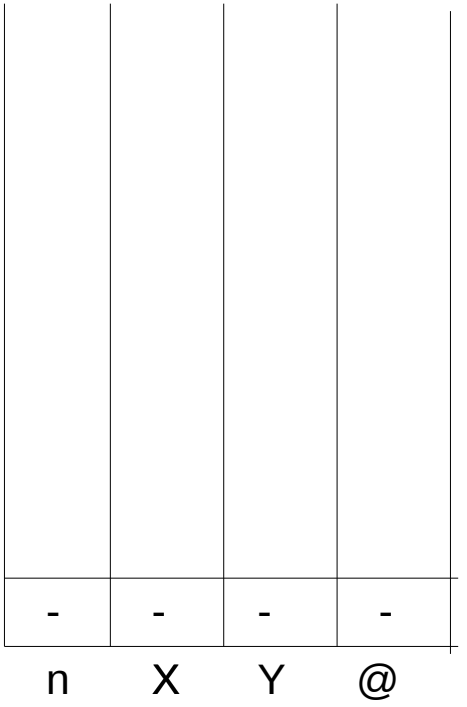




la pile P

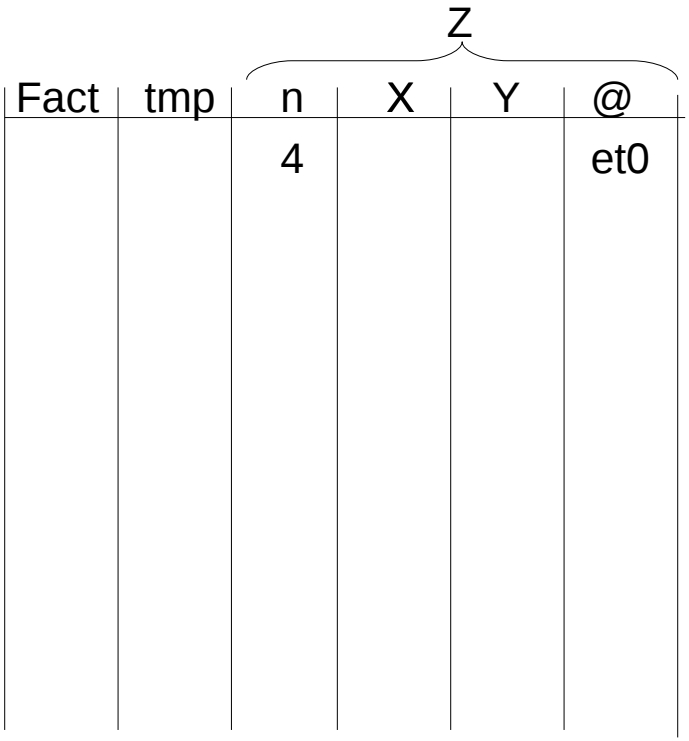
```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
       SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

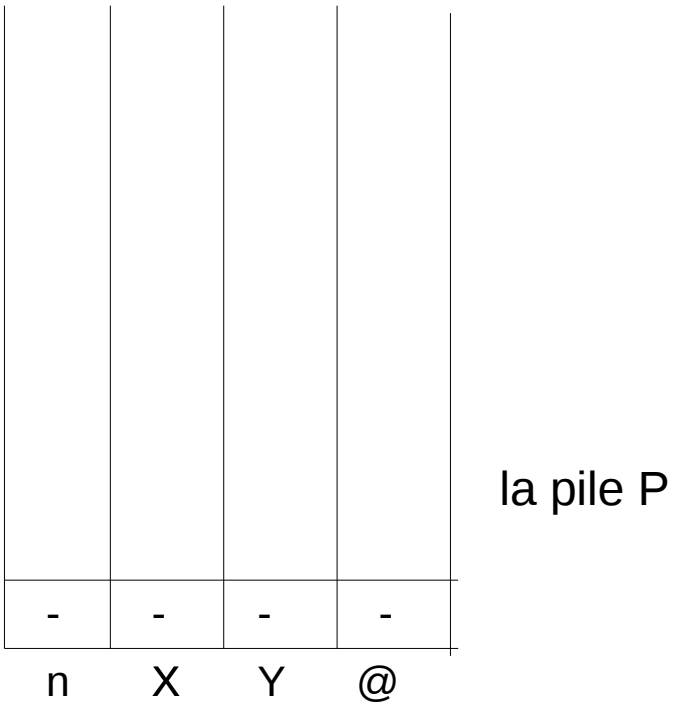




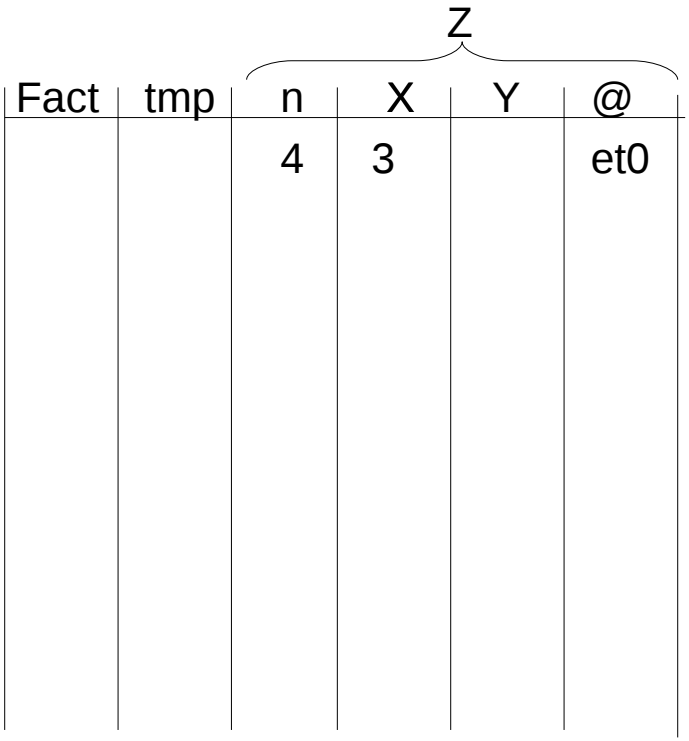
la pile P

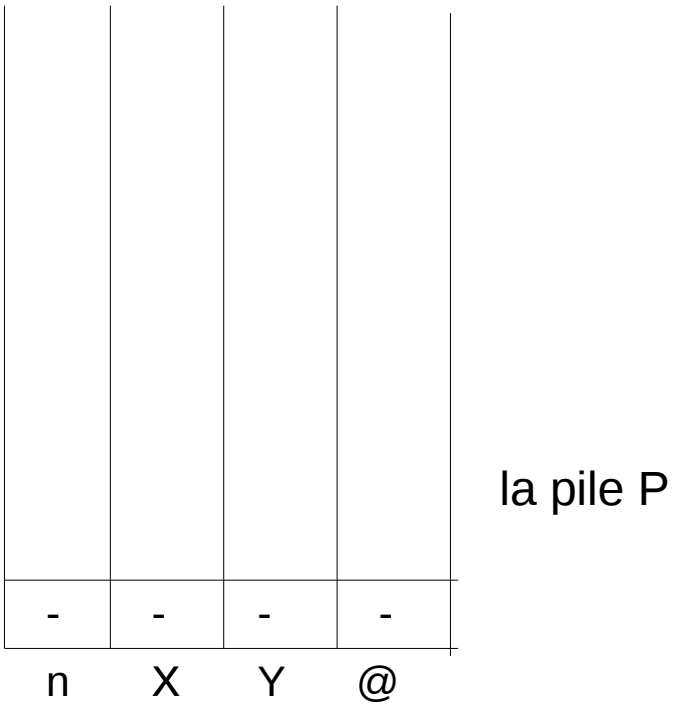
```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:SI  (Z.n=0)  Fact := 1
      SINON
          Z.X := Z.n - 1;
          Empiler(P,Z);
          Z.n := Z.X;
          Z.@ := et2;
          Aller à et1;
et2:      Z.Y := Fact ;
          Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```





```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
      SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```





```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
       SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0

4	3	-	et0	la pile P
-	-	-	-	
n	X	Y	@	

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
       SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

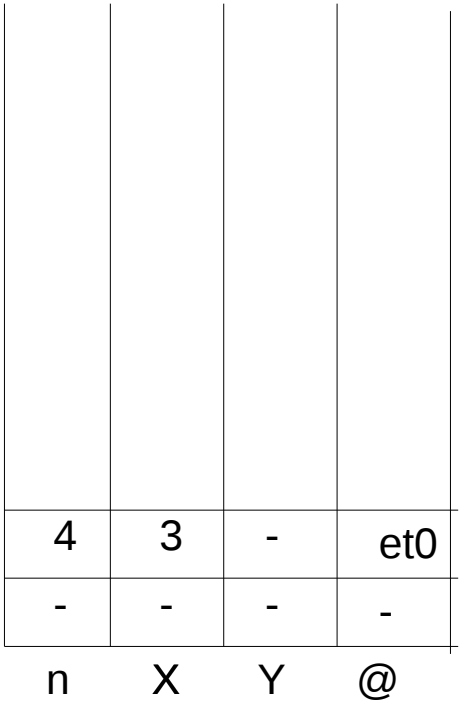
		Z			
Fact	tmp	n	X	Y	@
		4 3	3		et0

4	3	-	et0	
-	-	-	-	
n	X	Y	@	

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
       SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3			et2



la pile P

```
        CreerPile(P);
        Empiler(P,Z);
        Z.n := 4;   Z.@ := et0;
        Aller à et1
   et0: écrire( Fact );
        stop.

/* Traduction de la fonction */
   et1:SI  (Z.n=0)  Fact := 1
         SINON
             Z.X := Z.n - 1;
             Empiler(P,Z);
             Z.n := Z.X;
             Z.@ := et2;
             Aller à et1;
   et2:      Z.Y := Fact ;
             Fact : = Z.n * Z.Y;
        FSI
        tmp := Z.@;
        Depiler(P,Z);
        Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3			et2

n	X	Y	@
4	3	-	et0
-	-	-	-

la pile P

```
        CreerPile(P);
        Empiler(P,Z);
        Z.n := 4;   Z.@ := et0;
        Aller à et1
    et0: écrire( Fact );
        stop.

/* Traduction de la fonction */
    et1:SI  (Z.n=0)  Fact := 1
         SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
    et2:    Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
        FSI
        tmp := Z.@;
        Depiler(P,Z);
        Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3			et2

la pile P

n	X	Y	@
4	3	-	et0
-	-	-	-

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
       SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2

4	3	-	et0
-	-	-	-
n	X	Y	@

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2

n	X	Y	@
3	2	-	et2
4	3	-	et0
-	-	-	-

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2			

n	X	Y	@
3	2	-	et2
4	3	-	et0
-	-	-	-

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
       SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2			et2

n	X	Y	@
3	2	-	et2
4	3	-	et0
-	-	-	-

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
       SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2			et2

n	X	Y	@
3	2	-	et2
4	3	-	et0
-	-	-	-

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:SI  (Z.n=0)  Fact := 1
       SINON
           Z.X := Z.n - 1;
           Empiler(P,Z);
           Z.n := Z.X;
           Z.@ := et2;
           Aller à et1;
et2:       Z.Y := Fact ;
           Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2			et2

n	X	Y	@
3	2	-	et2
4	3	-	et0
-	-	-	-

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
      SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2

n	X	Y	@
3	2	-	et2
4	3	-	et0
-	-	-	-

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2

n	X	Y	@
2	1	-	et2
3	2	-	et2
4	3	-	et0
-	-	-	-

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1			

n	X	Y	@
2	1	-	et2
3	2	-	et2
4	3	-	et0
-	-	-	-

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1			et2

n	X	Y	@
2	1	-	et2
3	2	-	et2
4	3	-	et0
-	-	-	-

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
       SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1			et2

n	X	Y	@
2	1	-	et2
3	2	-	et2
4	3	-	et0
-	-	-	-

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:SI  (Z.n=0)  Fact := 1
       SINON
           Z.X := Z.n - 1;
           Empiler(P,Z);
           Z.n := Z.X;
           Z.@ := et2;
           Aller à et1;
et2:       Z.Y := Fact ;
           Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1			et2

n	X	Y	@
2	1	-	et2
3	2	-	et2
4	3	-	et0
-	-	-	-

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2

n	X	Y	@
2	1	-	et2
3	2	-	et2
4	3	-	et0
-	-	-	-

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2

n	X	Y	@
1	0	-	et2
2	1	-	et2
3	2	-	et2
4	3	-	et0
-	-	-	-

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
		0			

1	0	-	et2
2	1	-	et2
3	2	-	et2
4	3	-	et0
-	-	-	-
n	X	Y	@

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
		0			et2

1	0	-	et2
2	1	-	et2
3	2	-	et2
4	3	-	et0
-	-	-	-
n	X	Y	@

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
       SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
		0			et2

1	0	-	et2
2	1	-	et2
3	2	-	et2
4	3	-	et0
-	-	-	-
n	X	Y	@

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:SI  (Z.n=0)  Fact := 1
       SINON
           Z.X := Z.n - 1;
           Empiler(P,Z);
           Z.n := Z.X;
           Z.@ := et2;
           Aller à et1;
et2:       Z.Y := Fact ;
           Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1		0			et2

n	X	Y	@
1	0	-	et2
2	1	-	et2
3	2	-	et2
4	3	-	et0
-	-	-	-

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1		0			et2

n	X	Y	@
1	0	-	et2
2	1	-	et2
3	2	-	et2
4	3	-	et0
-	-	-	-

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2

1	0	-	et2
2	1	-	et2
3	2	-	et2
4	3	-	et0
-	-	-	-
n	X	Y	@

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0	-	et2

n	X	Y	@
2	1	-	et2
3	2	-	et2
4	3	-	et0
-	-	-	-

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0	-	et2

2	1	-	et2
3	2	-	et2
4	3	-	et0
-	-	-	-
n	X	Y	@

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0	- 1	et2

2	1	-	et2
3	2	-	et2
4	3	-	et0
-	-	-	-
n	X	Y	@

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0	-	et2
1				1	

2	1	-	et2
3	2	-	et2
4	3	-	et0
-	-	-	-
n	X	Y	@

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0	-	et2
1				1	

2	1	-	et2
3	2	-	et2
4	3	-	et0
-	-	-	-
n	X	Y	@

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0	-	et2
1	et2			1	

n	X	Y	@
2	1	-	et2
3	2	-	et2
4	3	-	et0
-	-	-	-

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
       SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0	-	et2
1	et2			1	
		2	1	-	et2

n	X	Y	@
3	2	-	et2
4	3	-	et0
-	-	-	-

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
       SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0	-	et2
1	et2			1	
		2	1	-	et2

n	X	Y	@
3	2	-	et2
4	3	-	et0
-	-	-	-

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
       SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0	-	et2
1	et2			1	
		2	1	-	et2
				1	

n	X	Y	@
3	2	-	et2
4	3	-	et0
-	-	-	-

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0	-	et2
1	et2			1	
			1	-	et2
2				1	

n	X	Y	@
3	2	-	et2
4	3	-	et0
-	-	-	-

la pile P

```
        CreerPile(P);
        Empiler(P,Z);
        Z.n := 4;   Z.@ := et0;
        Aller à et1
    et0: écrire( Fact );
        stop.

/* Traduction de la fonction */
    et1:SI  (Z.n=0)  Fact := 1
         SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
    et2:    Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
        FSI
        tmp := Z.@;
        Depiler(P,Z);
        Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0	-	et2
1	et2			1	
			1	-	et2
2				1	

3	2	-	et2
4	3	-	et0
-	-	-	-
n	X	Y	@

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

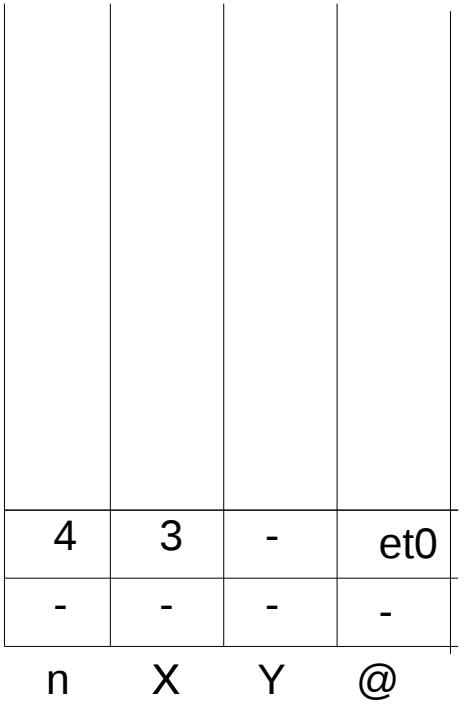
		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0	-	et2
1	et2			1	
			1	-	et2
2	et2			1	

n	X	Y	@
3	2	-	et2
4	3	-	et0
-	-	-	-

la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

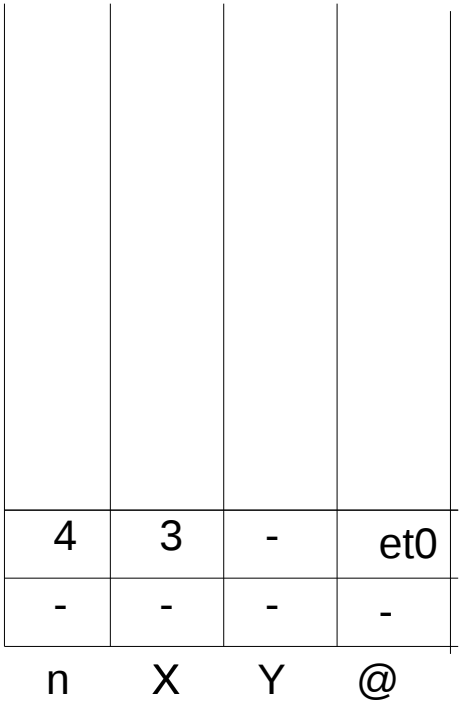
		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0	-	et2
1	et2			1	
			1	-	et2
2	et2			1	
		3	2	-	et2



la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

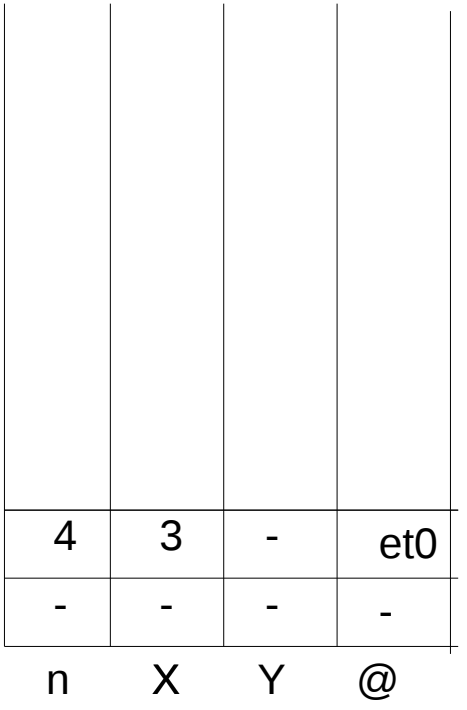
		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0	-	et2
1	et2			1	
			1	-	et2
2	et2			1	
		3	2	-	et2



la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
       SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

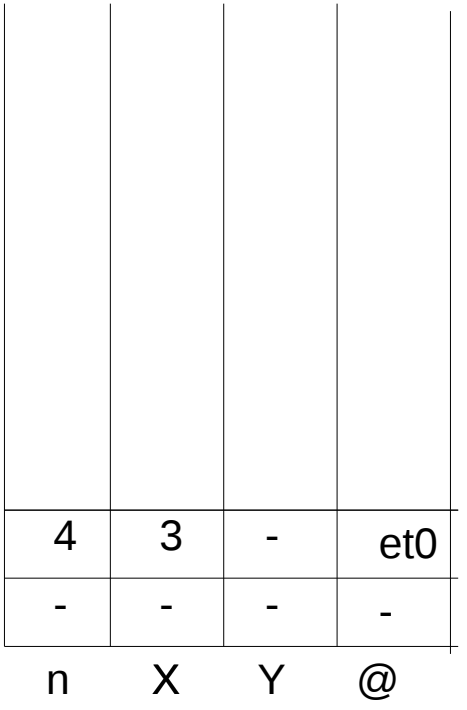
		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0		et2
1	et2			-1	
			1		et2
2	et2			-1	
		3	2		et2
				-2	



la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
       SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

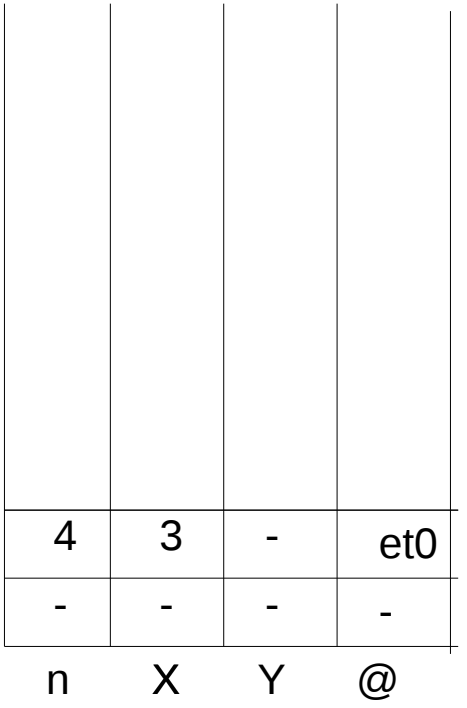
		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0	-	et2
1	et2			1	
			1	-	et2
2	et2			1	
		3	2	-	et2
6				2	



la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

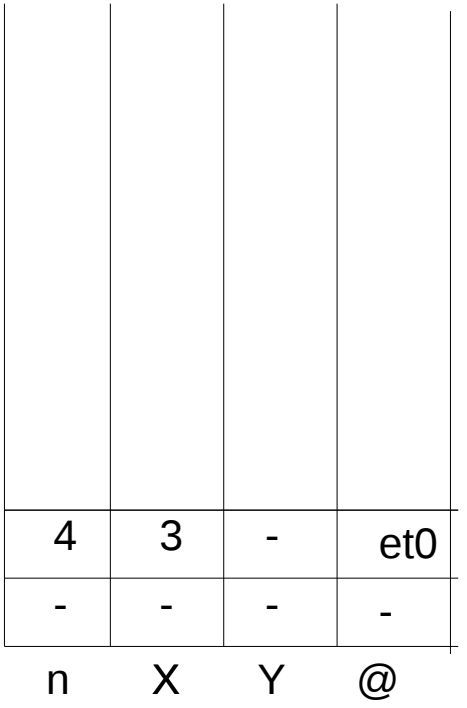
		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0		et2
1	et2			-1	
			1		et2
2	et2			-1	
		3	2		et2
6				-2	



la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

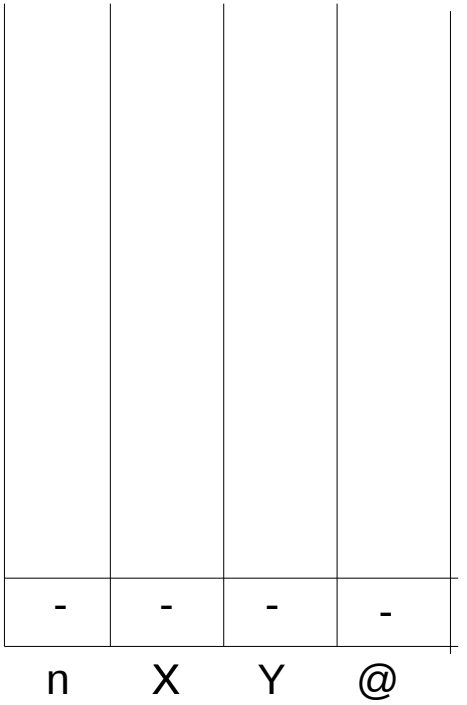
		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0	-	et2
1	et2			1	
			1	-	et2
2	et2			1	
		3	2	-	et2
6	et2			2	



la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

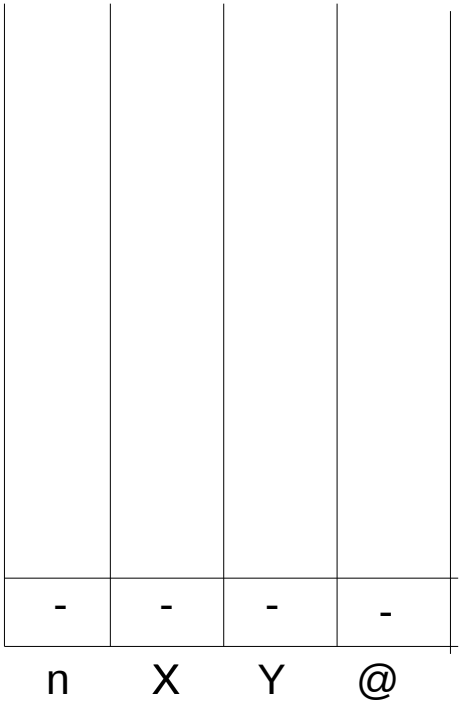
		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0	-	et2
1	et2			1	
			1	-	et2
2	et2			1	
		3	2	-	et2
6	et2			2	
		4	3	-	et0



la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
       SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

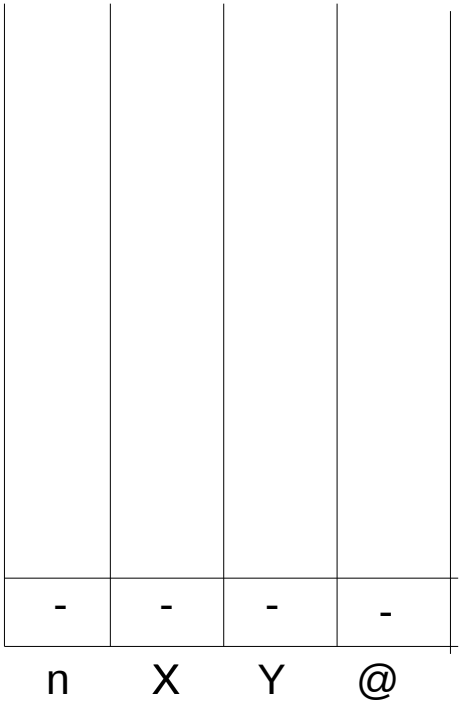
		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0	-	et2
1	et2			1	
			1	-	et2
2	et2			1	
		3	2	-	et2
6	et2			2	
		4	3	-	et0



la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
       SINON
           Z.X := Z.n - 1;
           Empiler(P,Z);
           Z.n := Z.X;
           Z.@ := et2;
           Aller à et1;
et2:       Z.Y := Fact ;
           Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

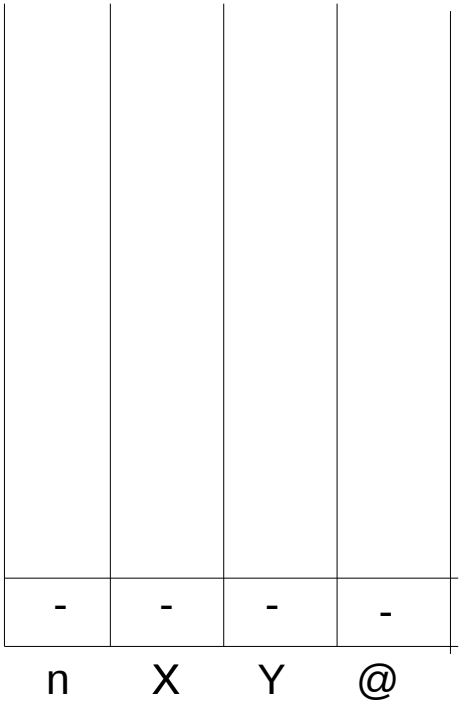
		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0		et2
1	et2			-1	
			1		et2
2	et2			-1	
		3	2		et2
6	et2			-2	
		4	3		et0
				-6	



la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

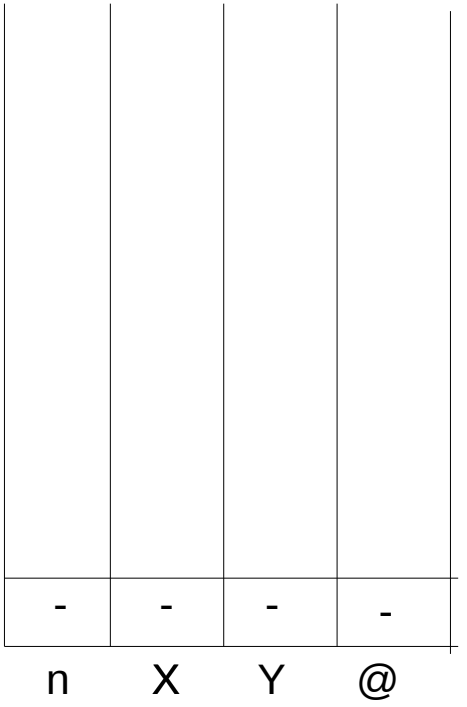
		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0		et2
1	et2			-1	
			1		et2
2	et2			-1	
		3	2		et2
6	et2			-2	
		4	3		et0
24				-6	



la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

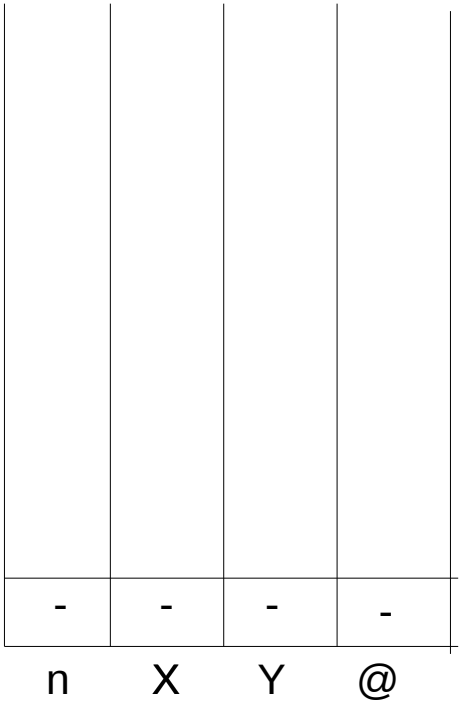
		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0		et2
1	et2			-1	
			1		et2
2	et2			-1	
		3	2		et2
6	et2			-2	
		4	3		et0
24				-6	



la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

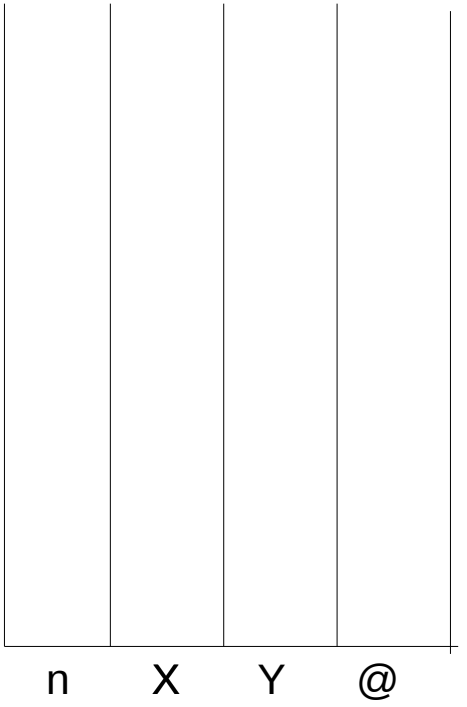
		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0		et2
1	et2			-1	
			1		et2
2	et2			-1	
		3	2		et2
6	et2			-2	
		4	3		et0
24	et0			-6	



la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

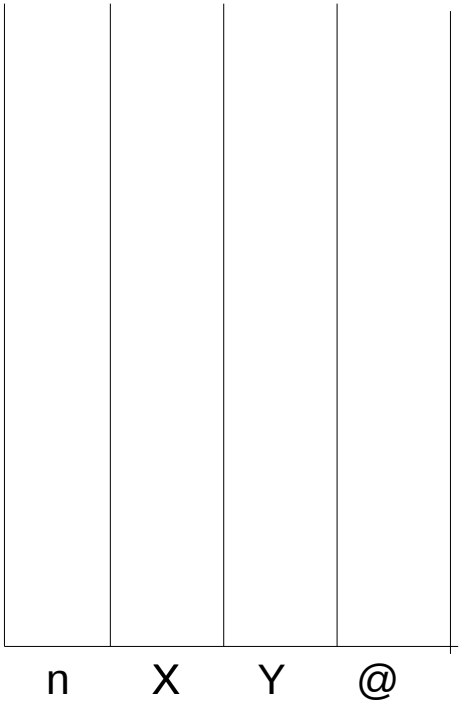
		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0	-	et2
1	et2			1	
			1	-	et2
2	et2			1	
		3	2	-	et2
6	et2			2	
		4	3	-	et0
24	et0			6	
		-	-	-	-



la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0:  écrire( Fact );
      stop.

/* Traduction de la fonction */
et1:  SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

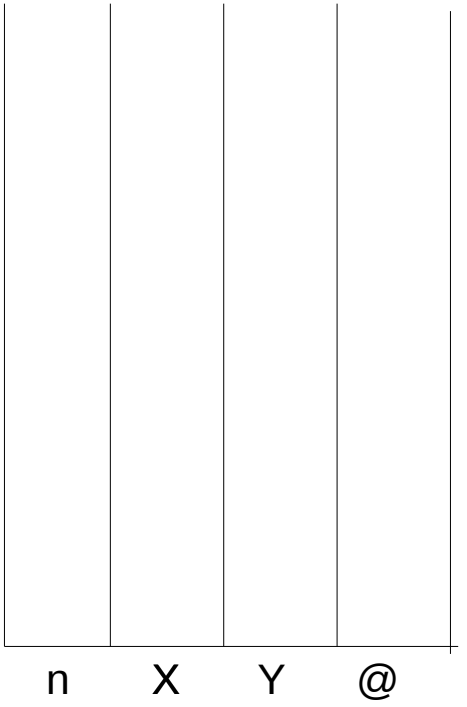
		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0	-	et2
1	et2			1	
			1	-	et2
2	et2			1	
		3	2	-	et2
6	et2			2	
		4	3	-	et0
24	et0			6	
		-	-	-	-



la pile P

```
      CreerPile(P);
      Empiler(P,Z);
      Z.n := 4;   Z.@ := et0;
      Aller à et1
et0: écrire( Fact );
      stop.

/* Traduction de la fonction */
   et1:SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
   et2:     Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
      FSI
      tmp := Z.@;
      Depiler(P,Z);
      Aller à tmp
```

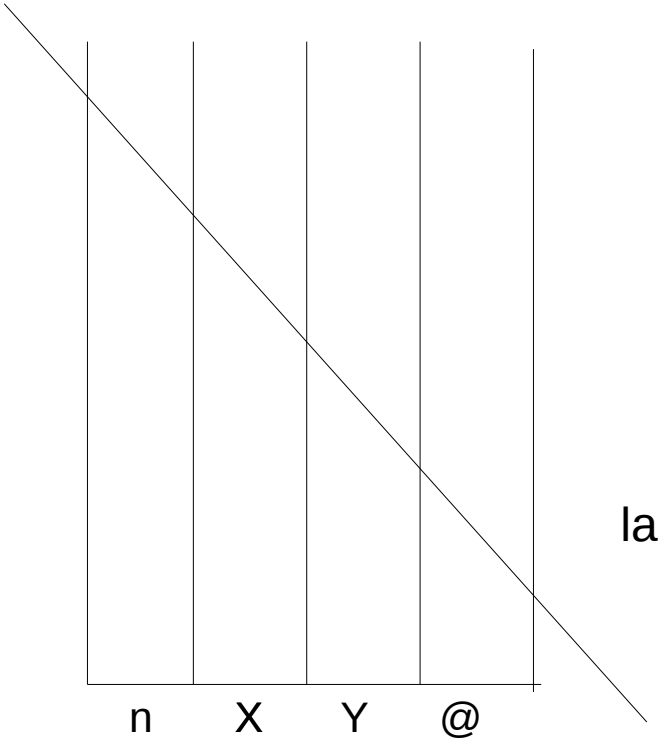
		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0	-	et2
1	et2			1	
			1	-	et2
2	et2			1	
		3	2	-	et2
6	et2			2	
		4	3	-	et0
24	et0			6	
		-	-	-	-



la pile P

```
        CreerPile(P);
        Empiler(P,Z);
        Z.n := 4;   Z.@ := et0;
        Aller à et1
et0:    écrire( Fact );
        stop.

/* Traduction de la fonction */
et1:    SI  (Z.n=0)  Fact := 1
        SINON
            Z.X := Z.n - 1;
            Empiler(P,Z);
            Z.n := Z.X;
            Z.@ := et2;
            Aller à et1;
et2:        Z.Y := Fact ;
            Fact : = Z.n * Z.Y;
        FSI
        tmp := Z.@;
        Depiler(P,Z);
        Aller à tmp
```

		Z			
Fact	tmp	n	X	Y	@
		4	3		et0
		3	2		et2
		2	1		et2
		1	0		et2
1	et2	0			et2
		1	0	-	et2
1	et2			1	
			1	-	et2
2	et2			1	
		3	2	-	et2
6	et2			2	
		4	3	-	et0
24	et0			6	
		-	-	-	-



la pile P

Amélioration de la transformation

- Réduction de la taille de la zone de données
  - Une variable (locale) doit être dans la ZDD uniquement si elle est initialisée avant un appel et réferencée après un appel.
  - Dans tous les autres cas, on peut l'enlever de la zone de données et la considérer comme variable globale
- S'il n'y a qu'un seul appel récursif dans le corps de la procédure ou de la fonction, on peut alors éliminer ce champ de la ZDD
  - car à chaque fois qu'on dépile on revient toujours à la même étiquette. Sauf pour le dernier dépilement où le retour devra se faire vers le programme appelant.
- Eliminer les « Aller à » en transformant les boucles en « Tantque » (difficile à faire)

Application à l'exemple précédent

X n'est pas référencé après l'appel

Y n'a pas de valeur avant l'appel

Donc on peut enlever X et Y de la ZDD et les considérer comme var globales

L'adr. de retour peut être enlevée, puisqu'il n'y a qu'un seul point de retour dans la fonction

Si on empile pas la première fois (dans le prog. appelant), on pourra savoir quand est-ce qu'il faudra revenir au prog. appelant (quand la pile devient vide)

```
Fact( n:entier ) : entier
  var
      X,Y : entier
  début
      SI  (n=0)
          Fact := 1
      SINON
          X := n-1;
          Y := Fact(X);
          Fact : = n * Y;
      FSI
 fin
```

```
    CreerPile(P);
    Z.n := 4;
    Aller à et1
et0: écrire( Fact );
    stop.
et1:SI  (Z.n=0)  Fact := 1
     SINON
         X := Z.n - 1;
         Empiler(P,Z);
         Z.n :=X;
         Aller à et1;
et2:     Y := Fact ;
         Fact : = Z.n * Y;
    FSI
    SI Non PileVide(P)
        Depiler(P,Z);
        Aller à et2
    SINON
        Aller à et0;
    FSI
```

```
    CreerPile(P);
    Z.n := 4;
et1:SI  (Z.n=0)  Fact := 1
     SINON
         X := Z.n - 1;
         Empiler(P,Z);
         Z.n :=X;
         Aller à et1;
et2:     Y := Fact ;
         Fact : = Z.n * Y;
    FSI
    SI Non PileVide(P)
        Depiler(P,Z);
        Aller à et2
    FSI
    écrire( Fact );
    stop.
```

boucle 1

boucle 2

```
CreerPile(P);
Z.n := 4;
et1:SI  (Z.n=0)  Fact := 1
     SINON
         X := Z.n - 1;
         Empiler(P,Z);
         Z.n :=X;
         Aller à et1;
et2:     Y := Fact ;
         Fact : = Z.n * Y;
    FSI
    SI Non PileVide(P)
        Depiler(P,Z);
        Aller à et2
    FSI
    écrire( Fact );
    stop.
```

boucle 1

```
TQ ( Z.n<>0 )
    X := Z.n - 1;
    Empiler(P,Z);
    Z.n :=X;
FTQ;
Fact := 1;
```

boucle 2

```
TQ ( Non PileVide(P) )
    Depiler(P,Z);
    Y := Fact ;
    Fact : = Z.n * Y;
FTQ
```

```
CreerPile(P);
Z.n := 4;
écrire( Fact );
stop.
```

Les Variables X et Y ne servent pratiquement à rien, on peut les enlever, de plus on peut remplacer Z par n

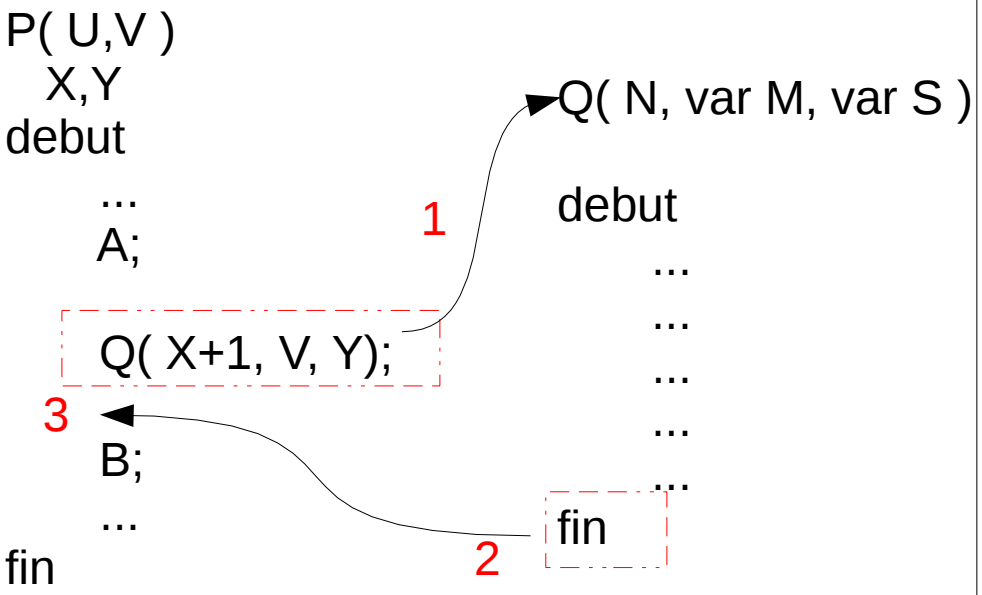
```
CreerPile(P);
Z.n := 4;

TQ ( Z.n<>0 )
    X := Z.n - 1;
    Empiler(P,Z);
    Z.n :=X;
FTQ;
Fact := 1;

TQ ( Non PileVide(P) )
    Depiler(P,Z);
    Y := Fact ;
    Fact : = Z.n * Y;
FTQ

écrire( Fact );
stop.
```

```
CreerPile(P);
n := 4;

TQ ( n<>0 )
    Empiler(P,n);
    n := n-1;
FTQ;

Fact := 1;

TQ ( Non PileVide(P) )
    Depiler(P,n);
    Fact : = n * Fact;
FTQ

écrire( Fact );
stop.
```

Cas des paramètres de sortie
(passage par référence)

On peut simuler leur comportement (sans utiliser l'indirection)

- en transmettant les valeurs lors de l'appel
- en récupérant leur dernières valeurs (juste avant le dépilement) dans des temporaires pour les réaffecter aux paramètres effectifs adéquats lors du retour.



1: Lors de l'appel
Empiler Zp
Zq.N := Zp.X + 1
Zq.M := Zp.V
Zq.S := Zp.Y
Aller au début de Q

2: Lors de la fin de Q
tmp0 := Zq.@
tmp1 := Zq.M
tmp2 := Zq.S
Depiler Zp
Aller à tmp0

3: Lors du retour
Zp.V := tmp1
Zp.Y := tmp2
B;
...