Nelson Mandela *(1918-2013)*

Education is the most powerful weapon which you can use to change the world.

Algorithmique & Programmation

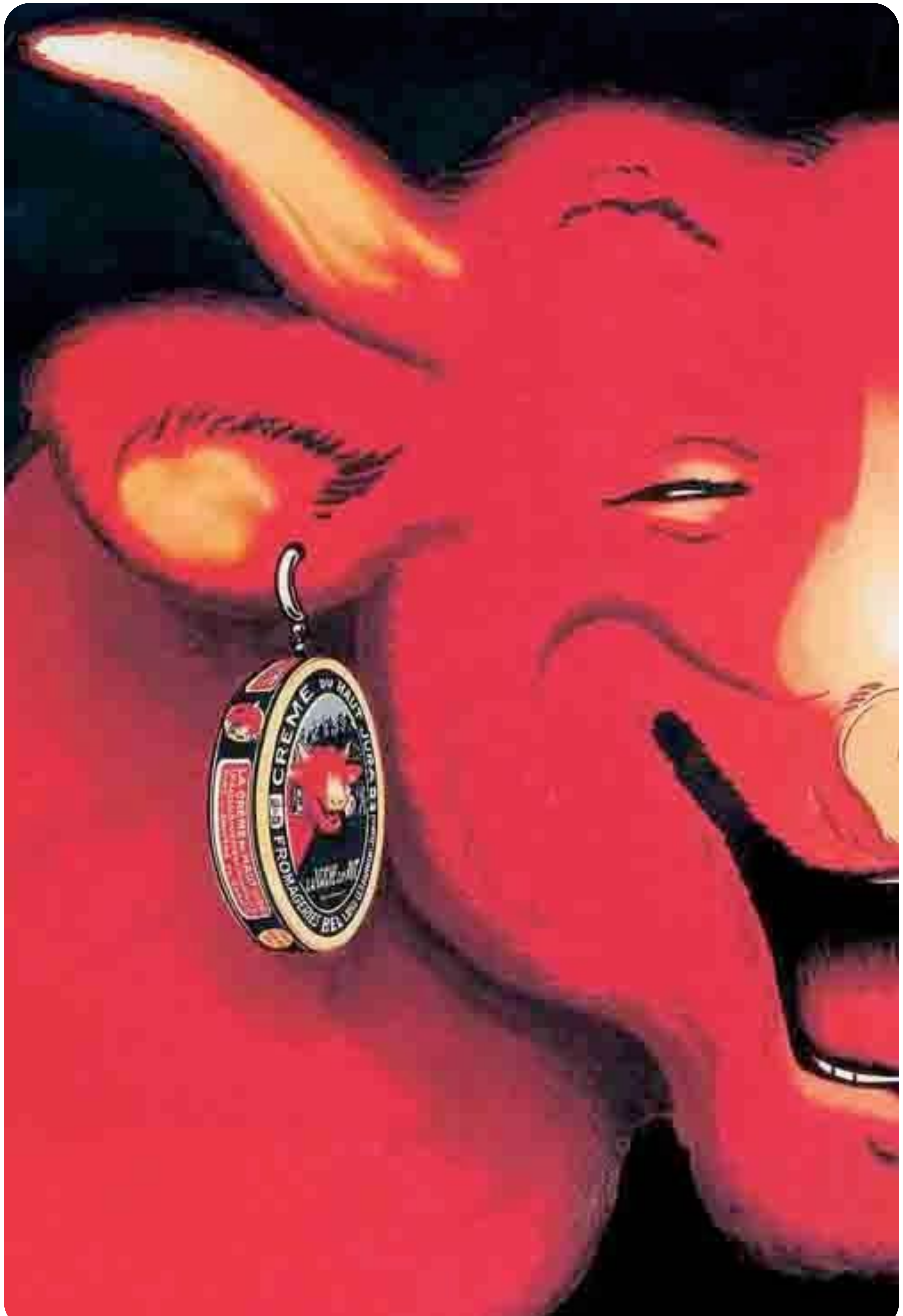
La notion de récursivité

yann.secq@univ-lille.fr

BONEVA Iovka, CAPELLE Cindy, CASTILLON Antoine, DELECROIX Fabien, LEPRETRE Éric, PLACE Jean-Marie, RICHARD Grégoire, SECQ Yann, TEDJINI Takwa

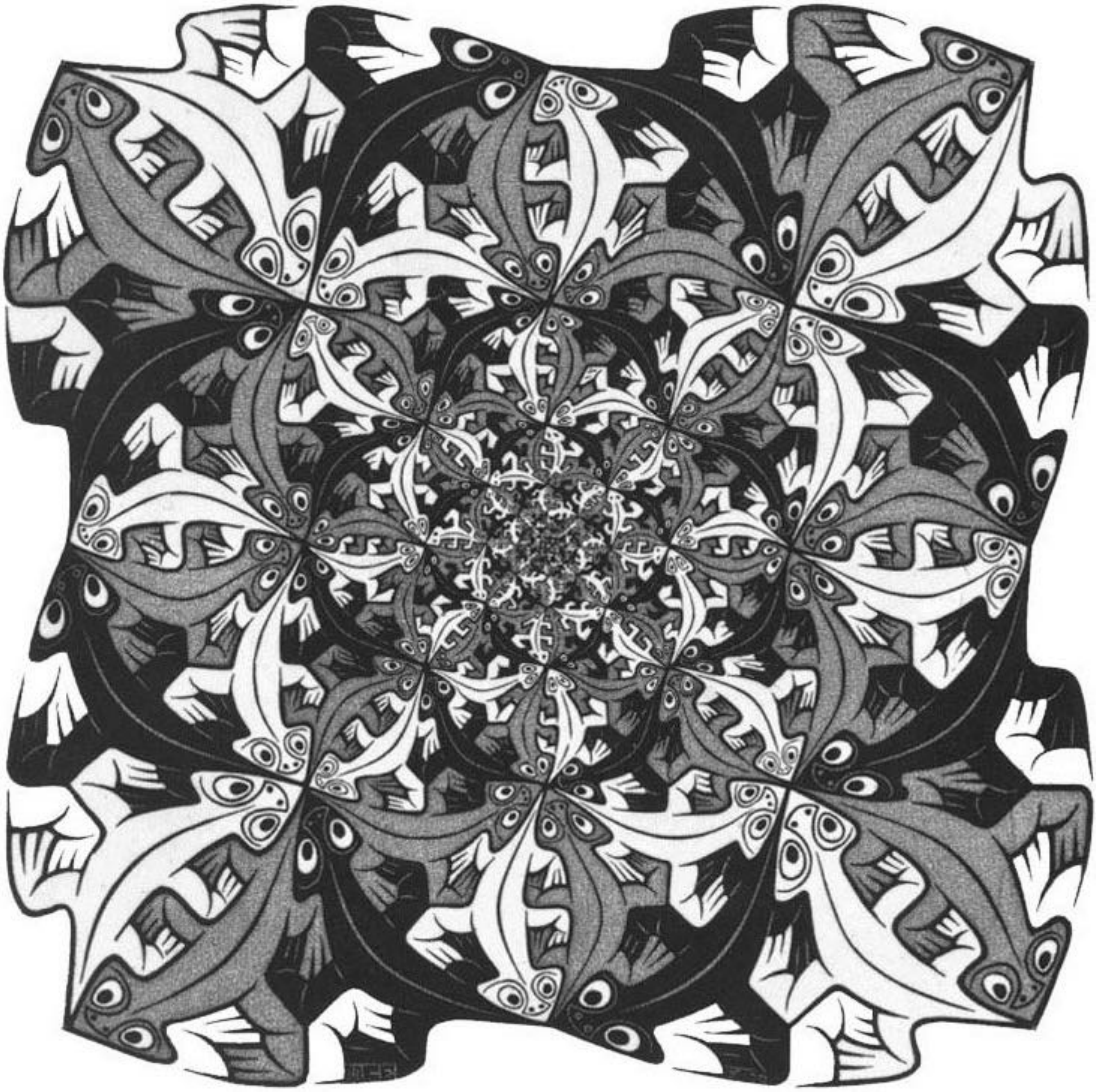
Saison 2022-2023 / R1.01+R1.02 : Initiation au développement

CREME DU HAUT JURA
FROMAGERIES BEL
Benjamin Rabier

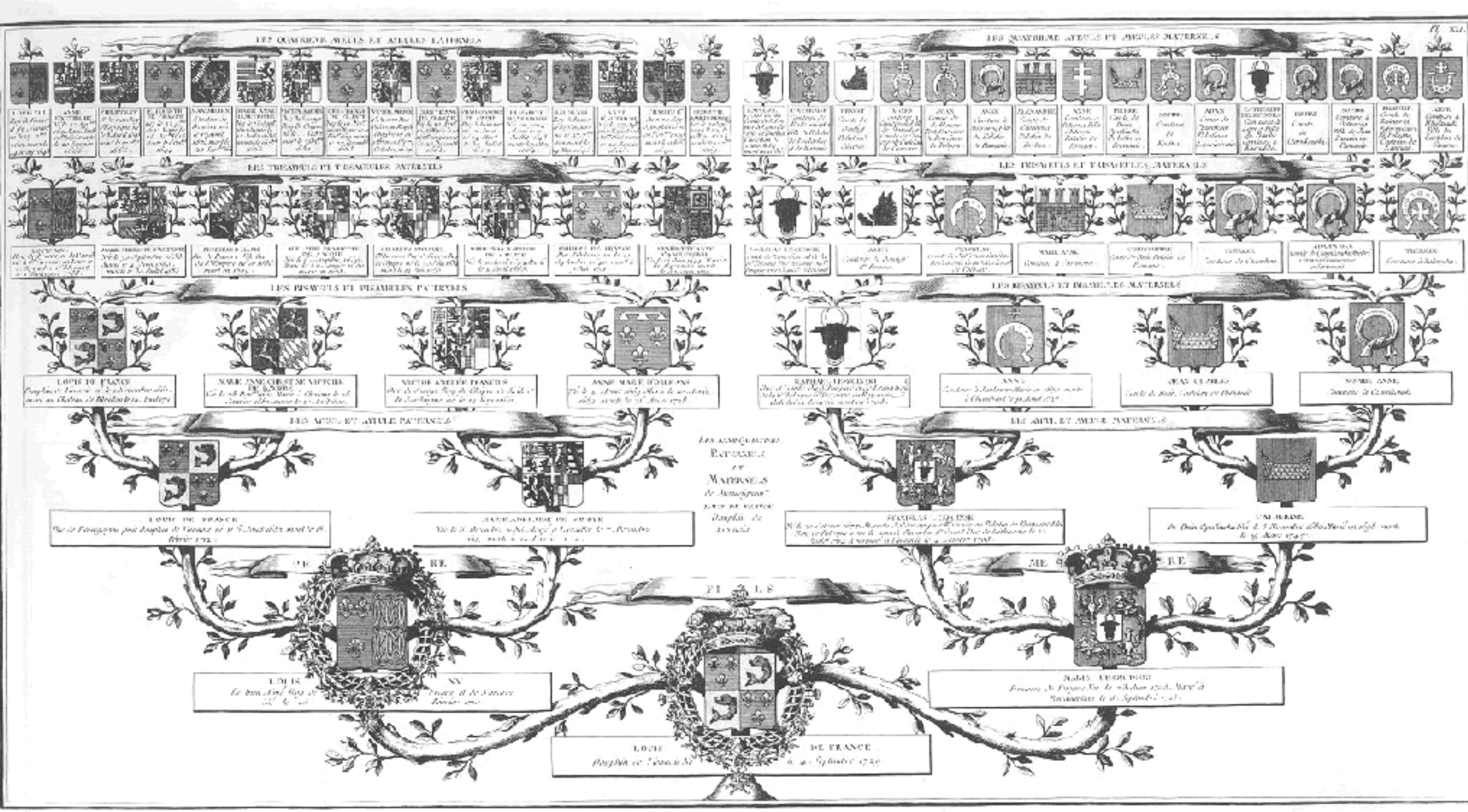


CREME DU HAUT JURA
FROMAGERIES BEL

CREME DU HAUT JURA
FROMAGERIES BEL



Qui est ancêtre de ?



...ustration : L'Encyclopédie, par Diderot et d'Alembert - www.blason-armoiries.org

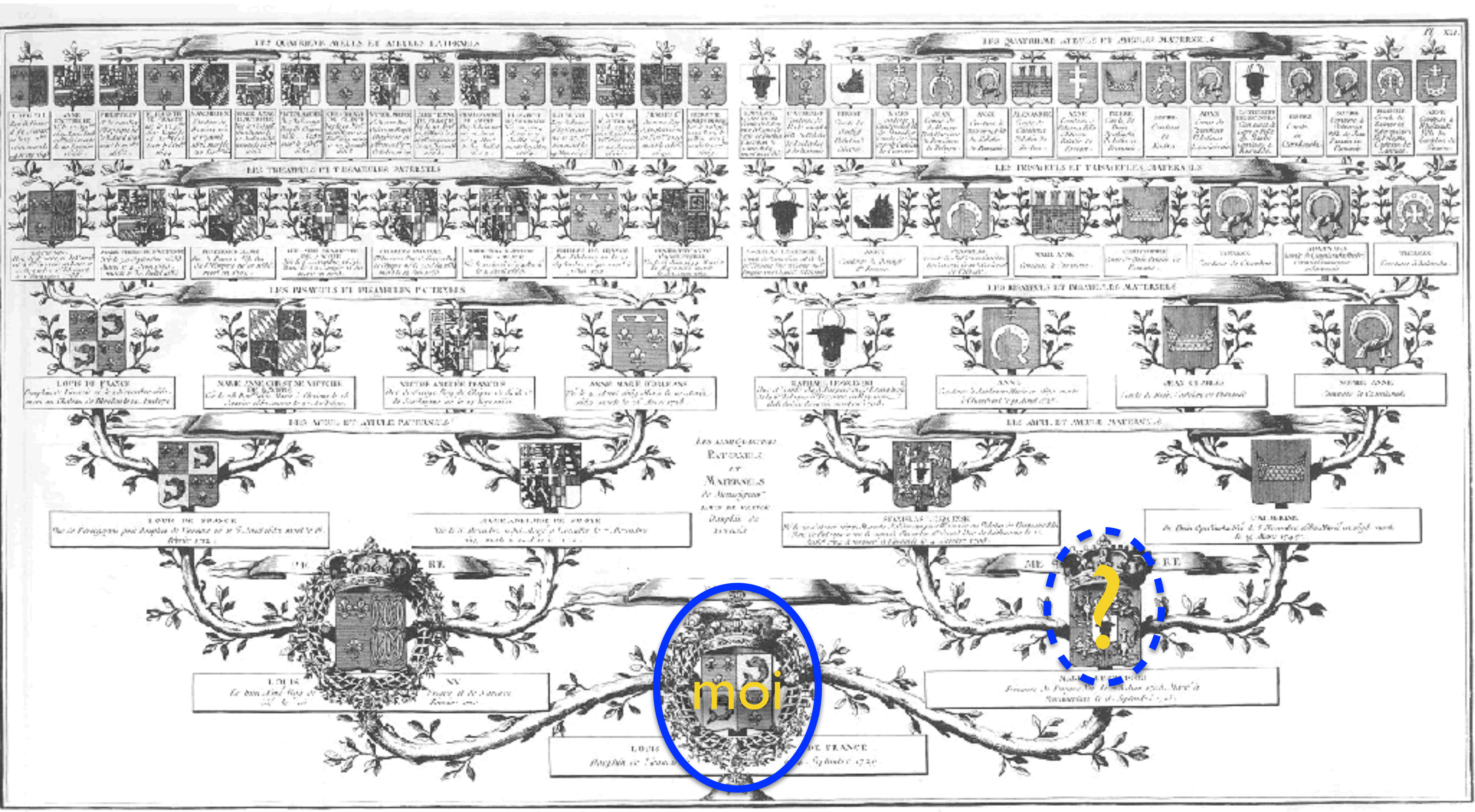
Ancêtre de X ?

- Une définition simple

X est mon **ancêtre** si:
- X est mon père ou ma mère
- X est un **ancêtre** de mon père ou ma mère

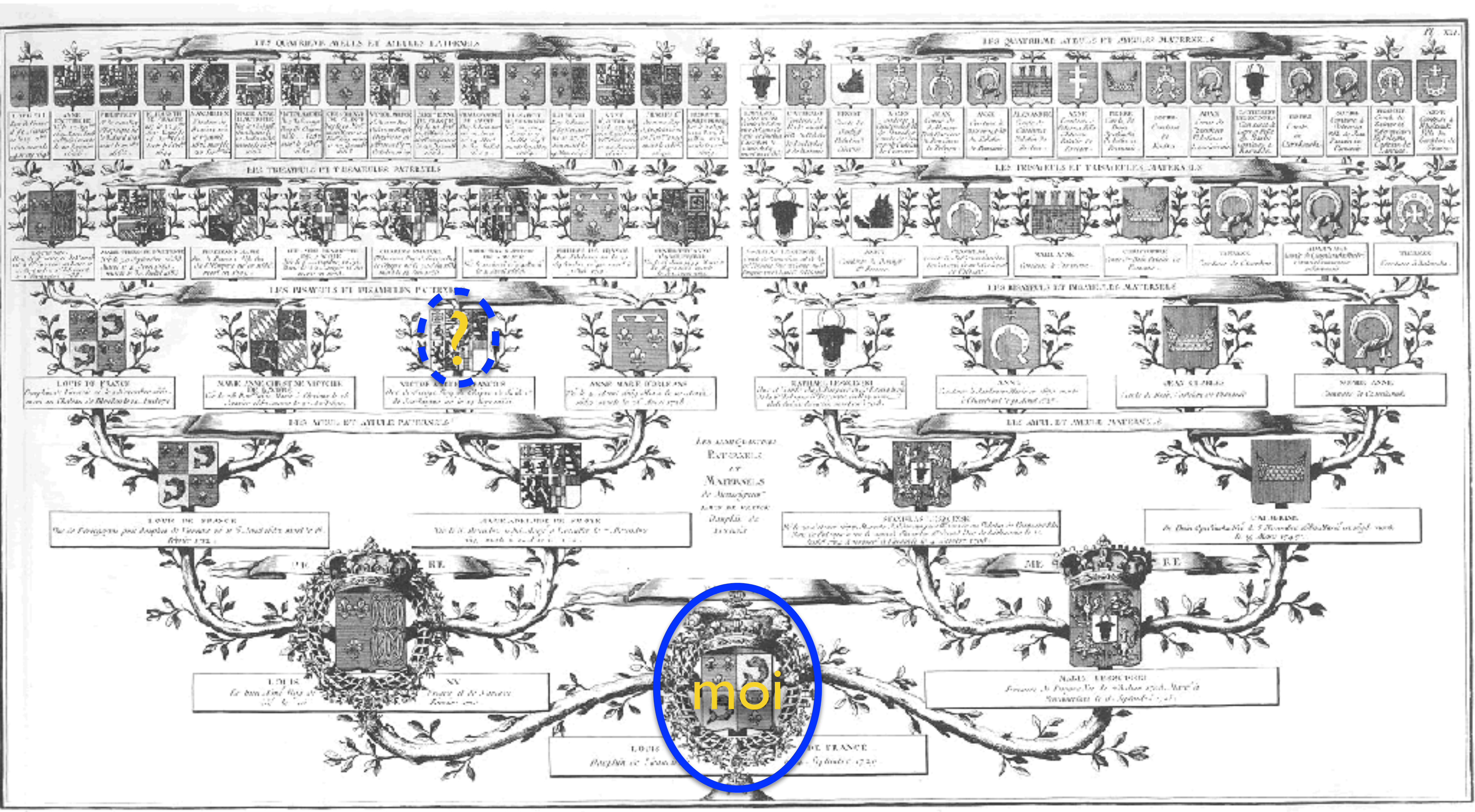
- Définition qui fait appel à elle même !
- C’est une définition récursive :)

Qui est ancêtre de ?



ustration : L'Encyclopédie, par Diderot et d'Alembert – www.blason-armoiries.org

Qui est ancêtre de ?



ustration : *L'Encyclopédie*, par Diderot et d'Alembert – www.blason-armoiries.org

Qui est ancêtre de ?

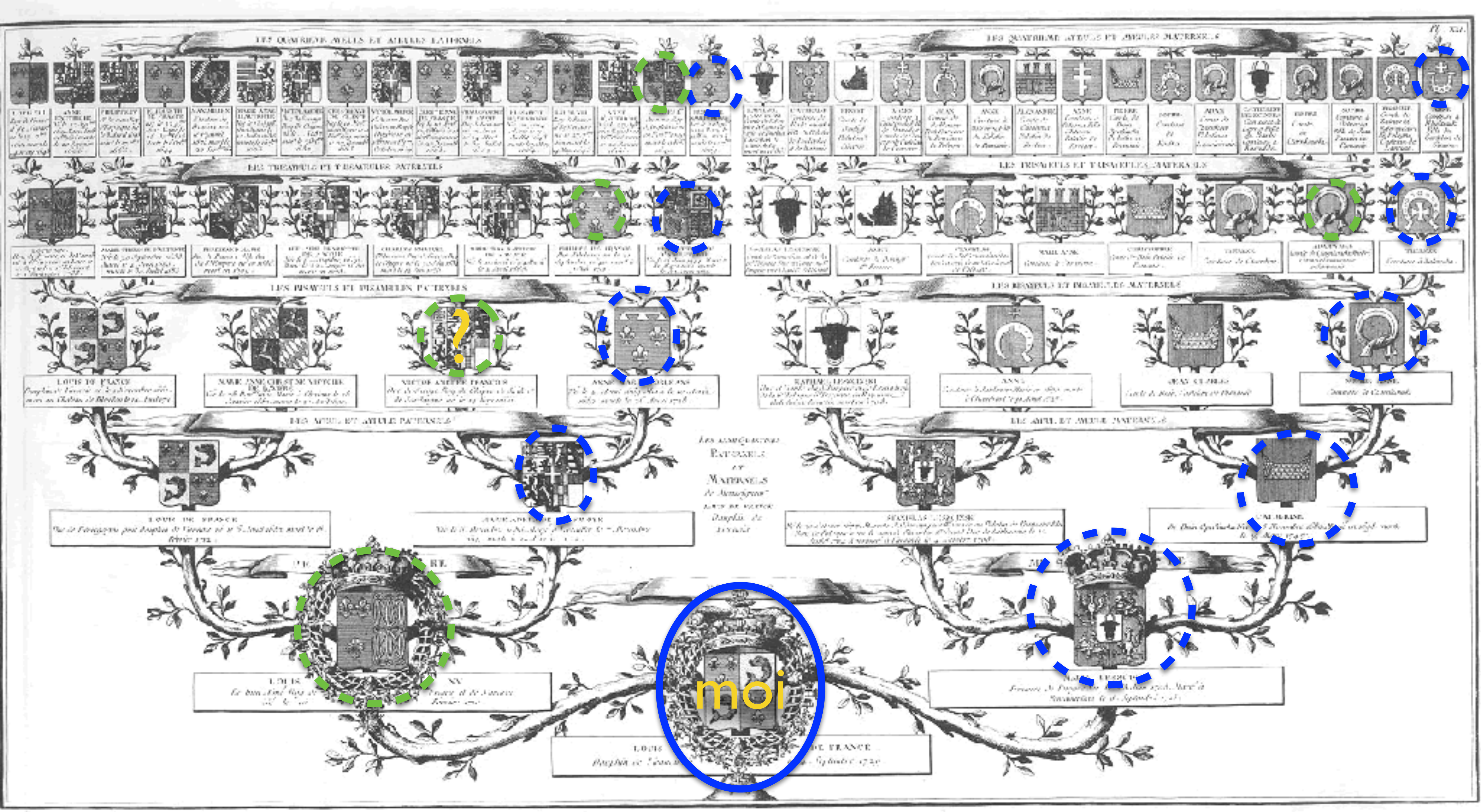


Illustration : L'Encyclopédie, par Diderot et d'Alembert – www.blason-armoiries.org

Le `Hello World` de la récursivité: factorielle !

- En itératif (déjà fait) ou en récursif:
  - `n! = n * (n-1) * ... * 2 * 1 et 0!=1`
  - `n! = n * (n-1)! et 0! = 1`
- Equivalent à l'équation de récurrence:

$$\texttt{fact(n)} \begin{cases} \texttt{1 si n=0} \\ \texttt{n*fact(n-1) sinon} \end{cases}$$

Factorielle en récursif

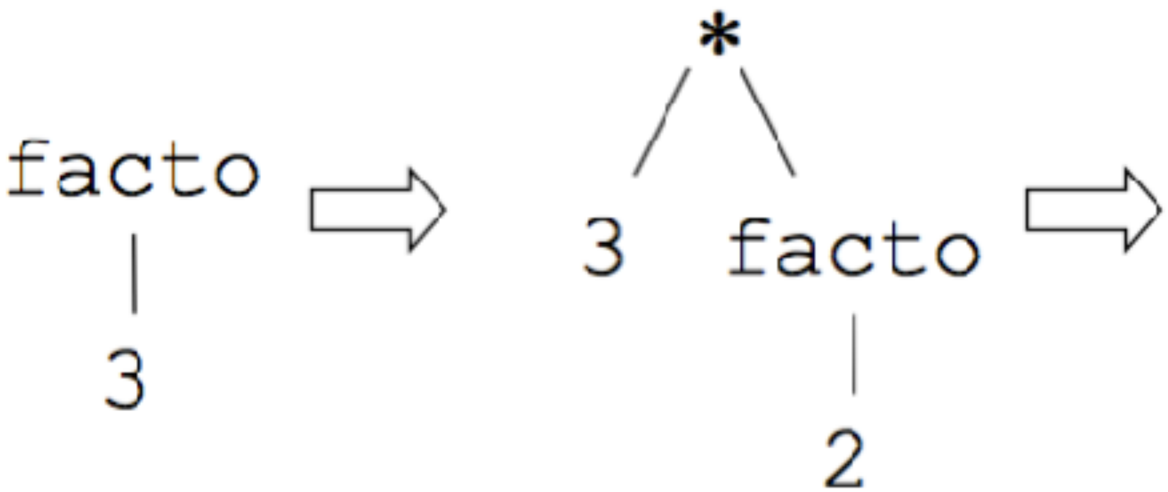
```
class Factorielle extends Program {

  int fact(int n) {
    if (n == 0) {
      return 1;
    }
    return n * fact(n-1);
  }

  void algorithm() {
    int a = readInt();
    println(a +"! = "+ fact(a));
  }
}
```

Petite trace des appels

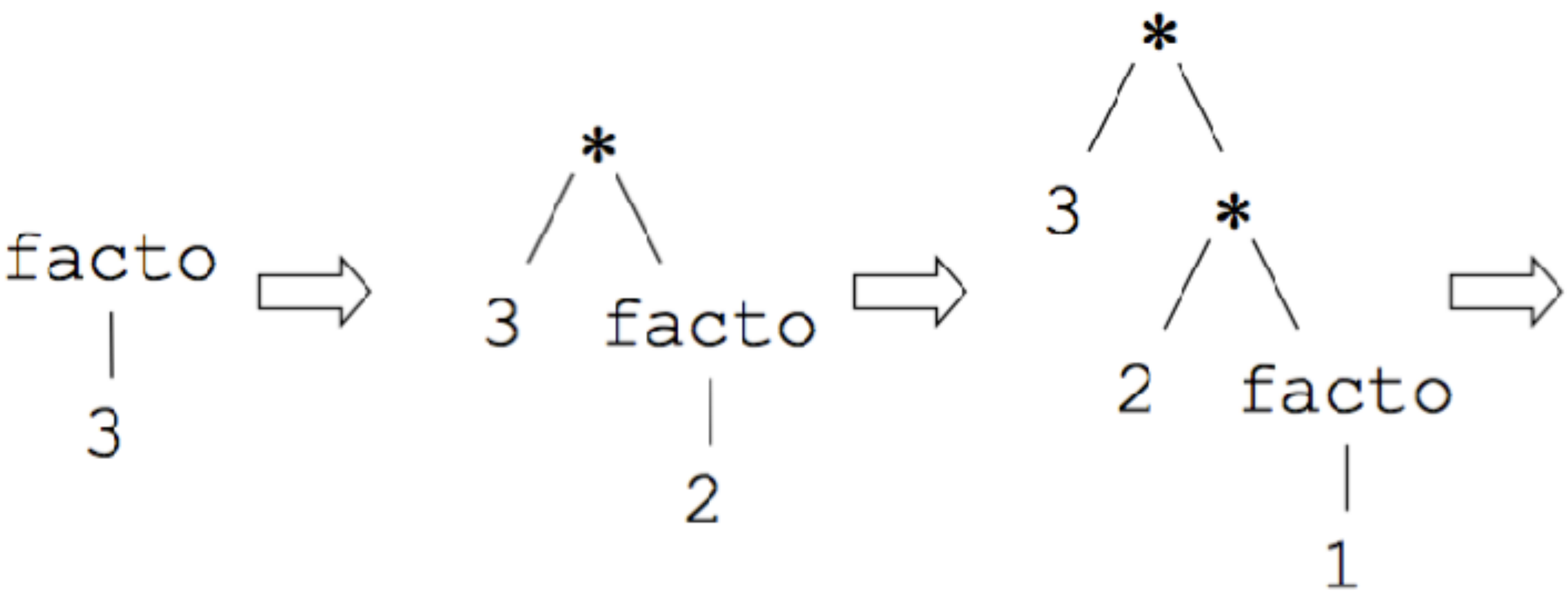
Phase d'empilement

```
facto  ⇨
  |
  3
```

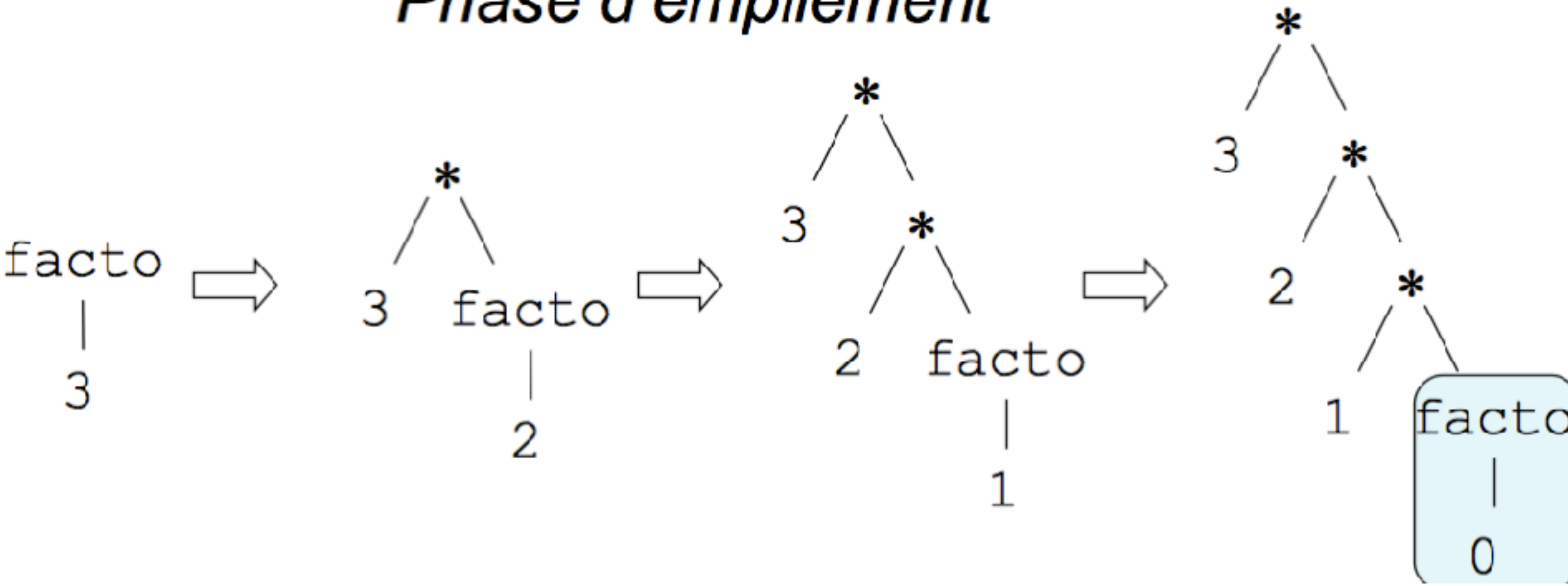
Phase d'empilement



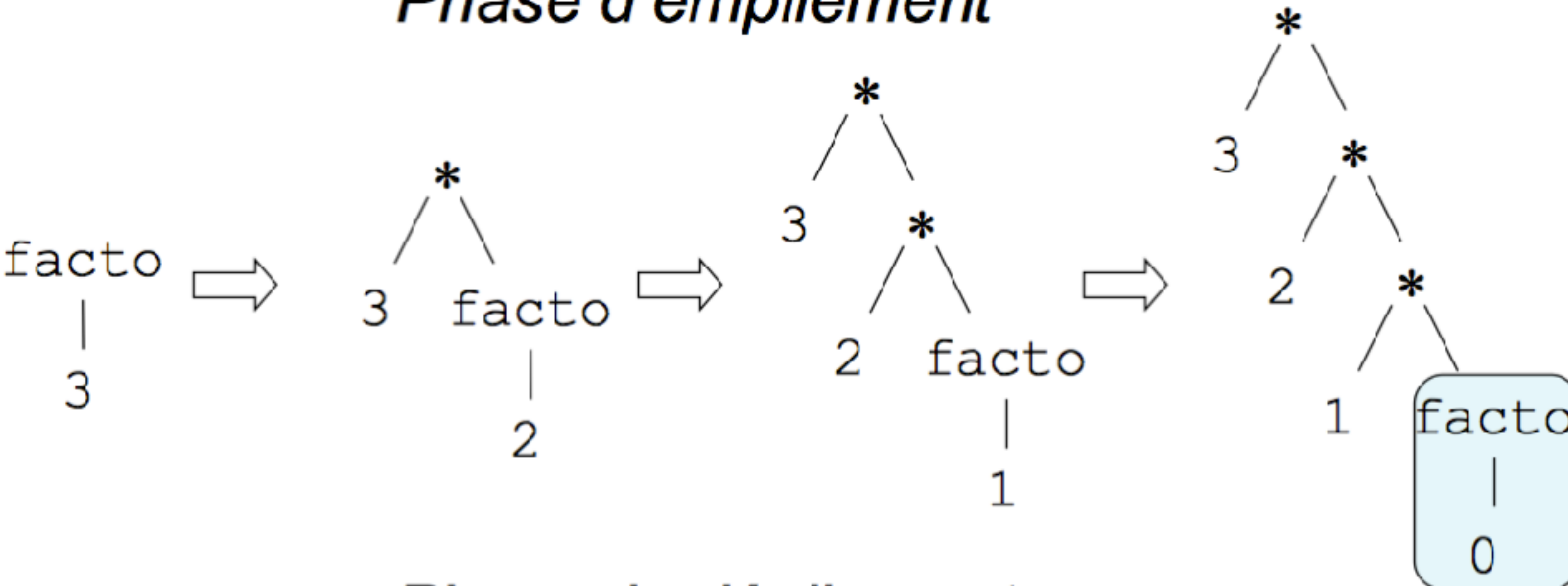
Phase d'empilement



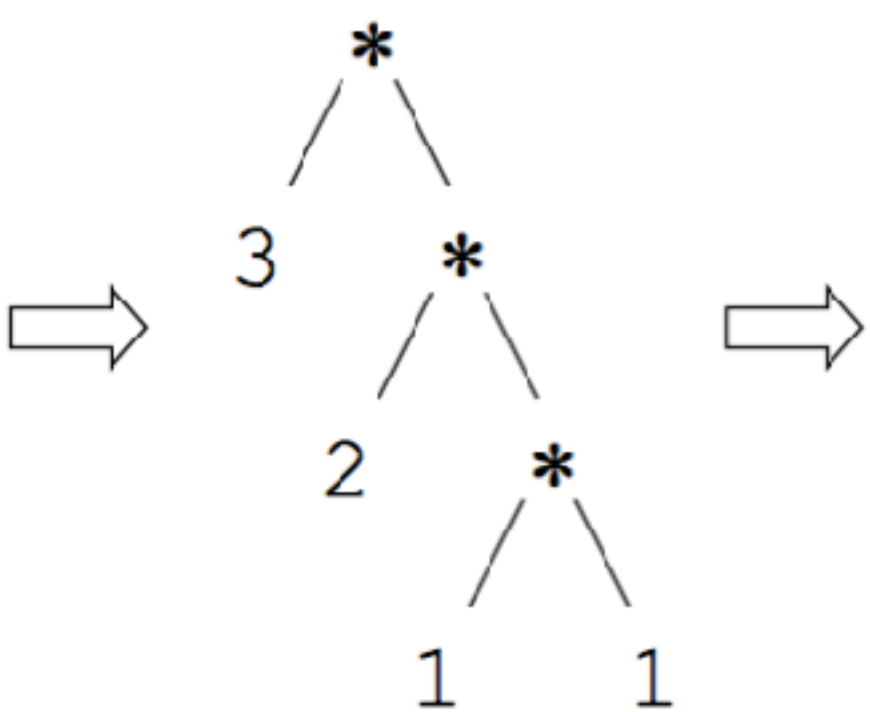
Phase d'empilement



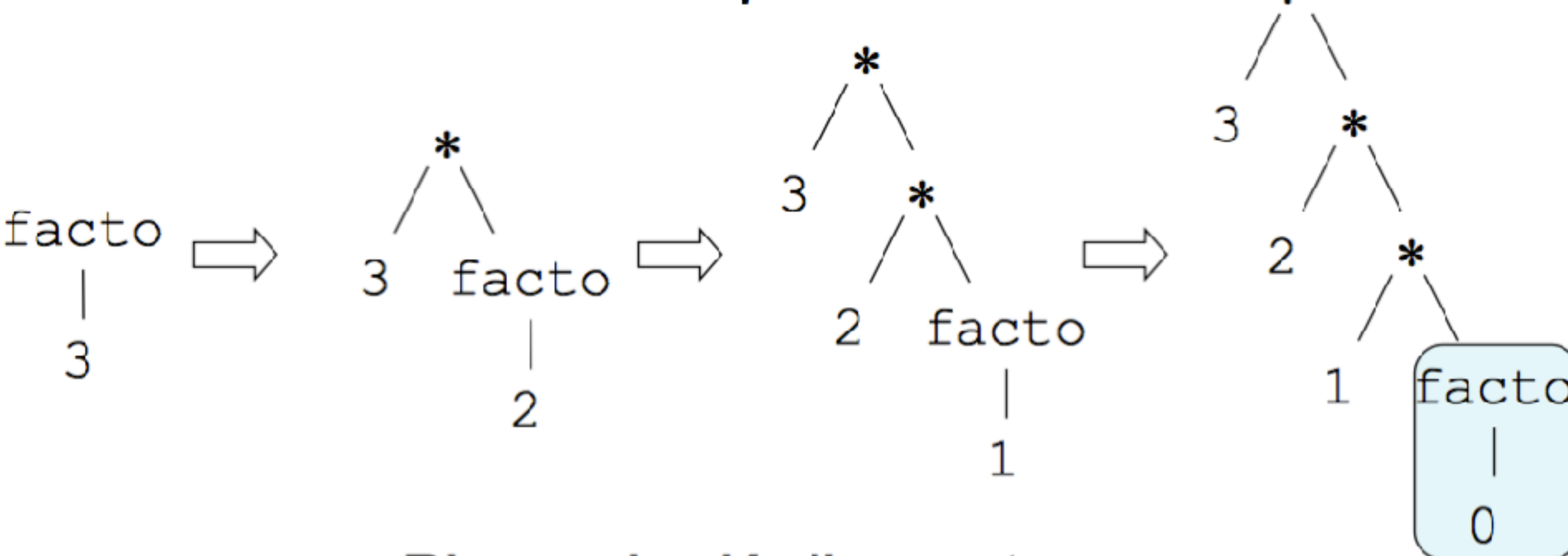
Phase d'empilement



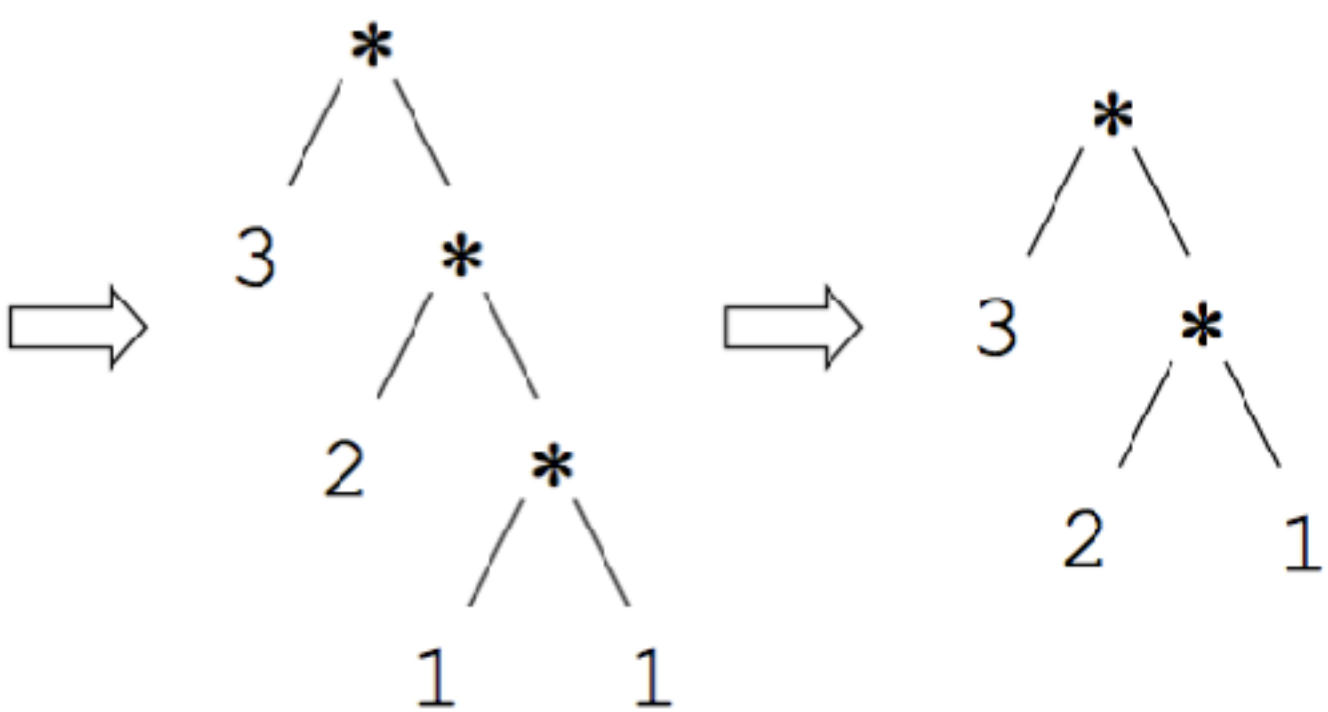
Phase de dépilement



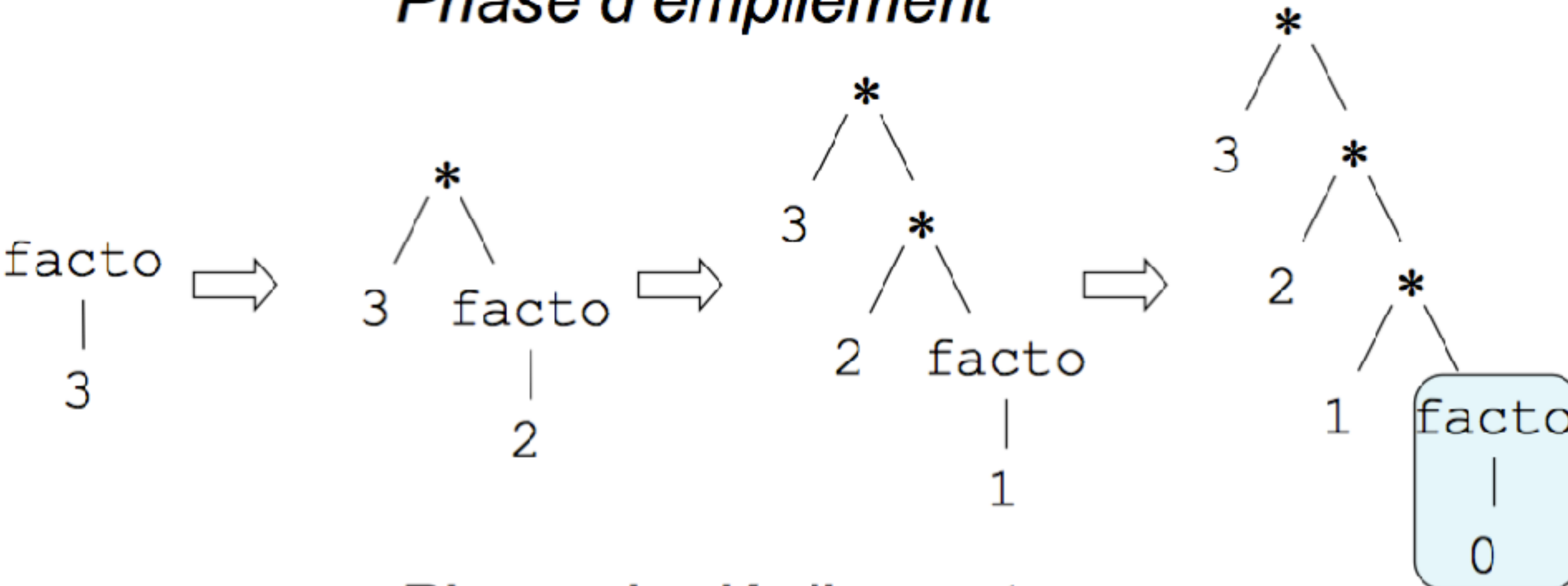
Phase d'empilement



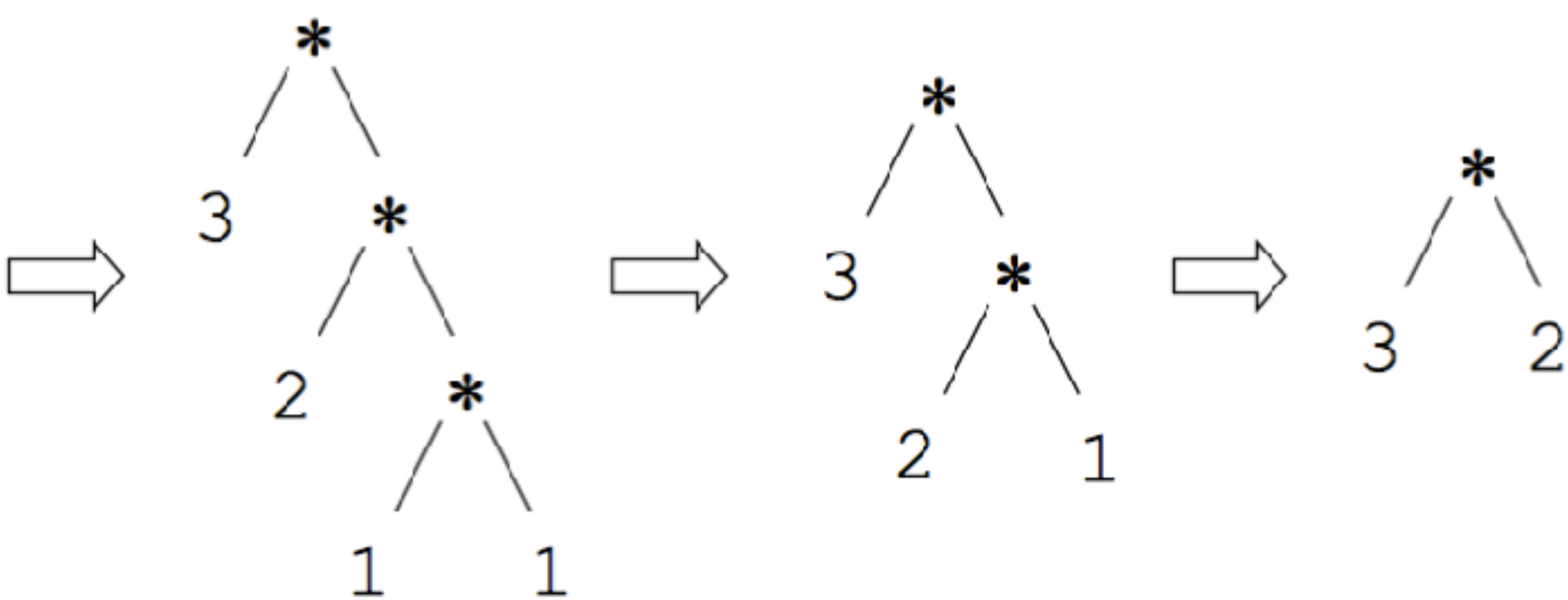
Phase de dépilement



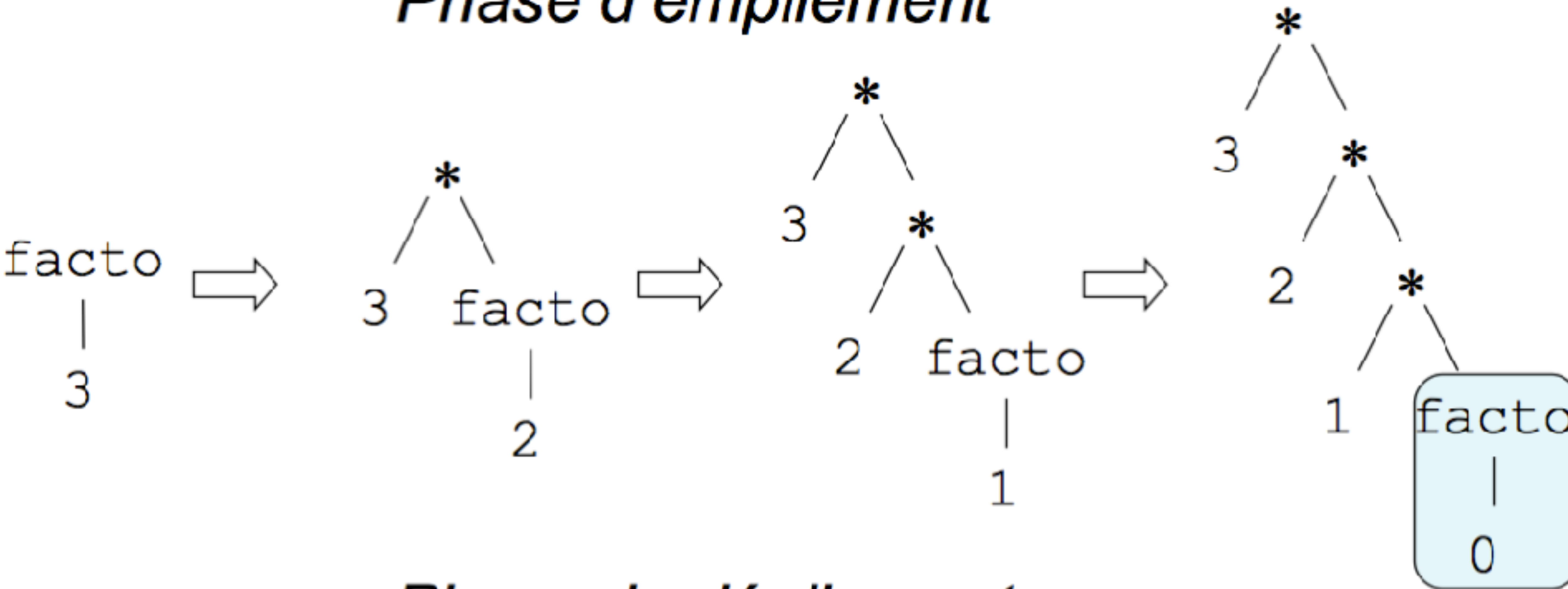
Phase d'empilement



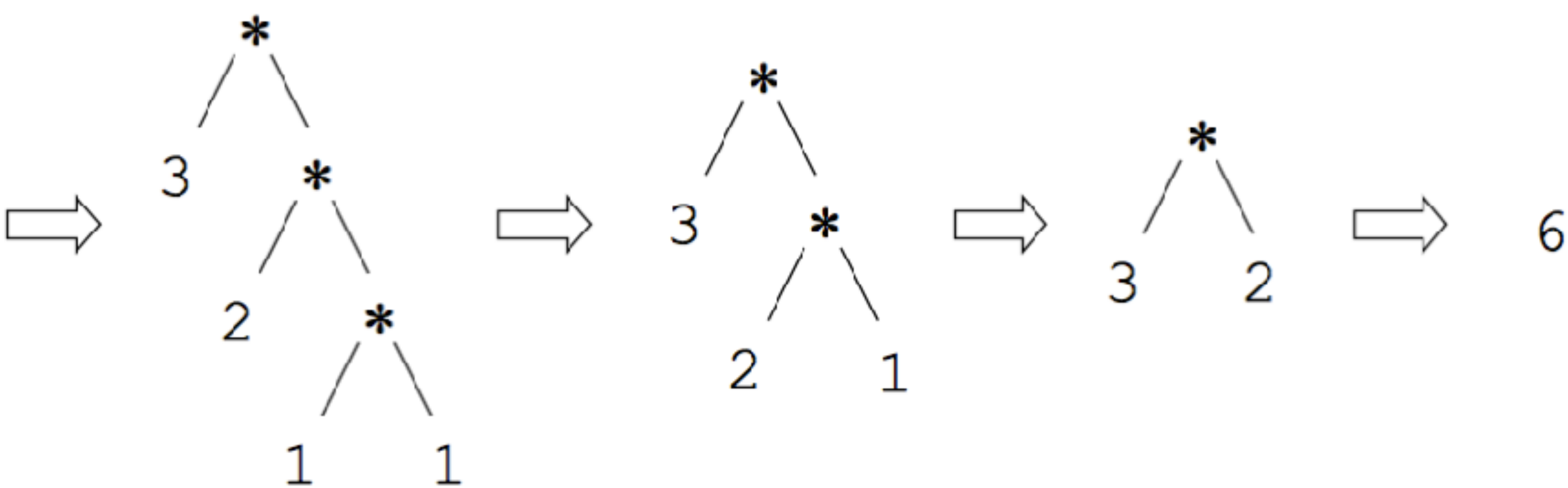
Phase de dépilement



Phase d’empilement



Phase de dépilement



Récursivité: définitions

- Un algorithme est dit récursif si l'expression qui le définit fait appel à elle même
- Un appel récursif correspond à un appel à une fonction `f` provoqué par l'évaluation d'un autre appel à `f`
- On parlera de fonction récursive pour une fonction définie par un algorithme récursif

Règles fondamentales

- Deux règles à respecter impérativement:
  - un algorithme récursif est défini par une expression conditionnelle dont l'**un des cas mène à une évaluation sans appel récursif** (que l'on appellera **la/les conditions d'arrêt**)
  - il faut s'assurer que pour toute valeur du ou des paramètres, il n'y aura qu'**un nombre fini d'appels récursifs**

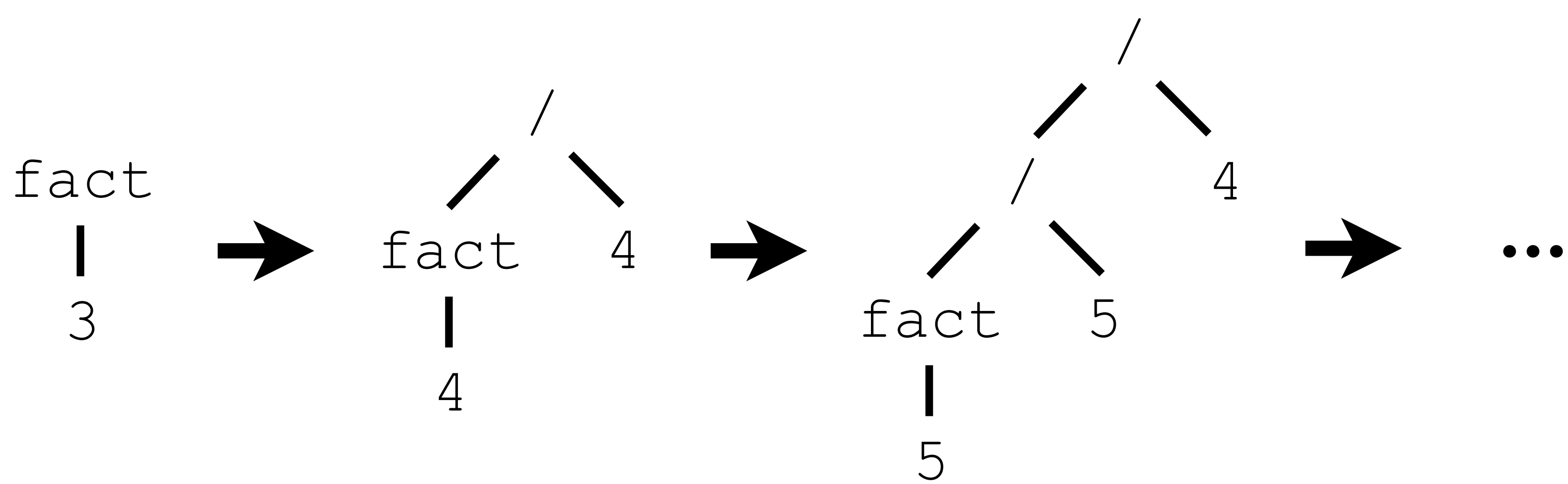
Mauvaises récursivité

- Deux exemples classiques
  - une récursivité infinie
  - une récursivité trop longue
- Récursivité infinie : appel récursif comme première instruction de la fonction ...
- Autre exemple: un arbre qui croît infiniment

Factorielle ... infinie !

- **Factorielle:** `n! = (n+1)!/(n+1)` **et** `0!=1`

```
int fact(int n) {
   if (n == 0) {
      return 1;
   }
   return fact(n+1)/(n+1);
}
```



Quelques exemples

- Un exemple classique sur les chaînes
- Des fonctions récursives siamoises ?
- Difficulté de maintenir un état dans une cascade d'appels récursifs ...

Présence d'un caractère

- Avec les chaînes, souvent une condition d'arrêt portant sur la chaîne vide et ensuite le traitement du caractère courant
- Un caractère `c` est-il présent dans phrase ?

Présence d'un caractère

- Un caractère c est-il présent dans phrase ?
  - Si la chaîne est vide, non !
  - Si *le premier caractère* == *c*, alors oui, sinon on recommence sur la chaîne privée de son premier caractère

```
class CaracterePresentDansChaine extends Program {

  boolean estPresent(char c, String m) {
    if (length(m)==0) {
      return false;
    } else if (charAt(m,0)==c) {
      return true;
    }
    return estPresent(c, substring(m,1,length(m)));
  }

  void algorithm() {
    String phrase = readString();
    char car = readChar();
    if (estPresent(car,phrase)) {
      println(car +" est présent dans "+ phrase);
    } else {
      println(car +" absent de "+ phrase);
    }
  }
}
```

Arbre d’évaluation

- Pas de croissance de l’arbre d’évaluation !

```
estPresent          estPresent          estPresent
   /   \       ➔       /   \       ➔       /   \       ➔     VRAI
'z'  "kazu"         'z'   "azu"         'z'   "zu"
```

- **Récursivité terminale** = mémoire constante et calcul terminé lors du dernier appel récursif

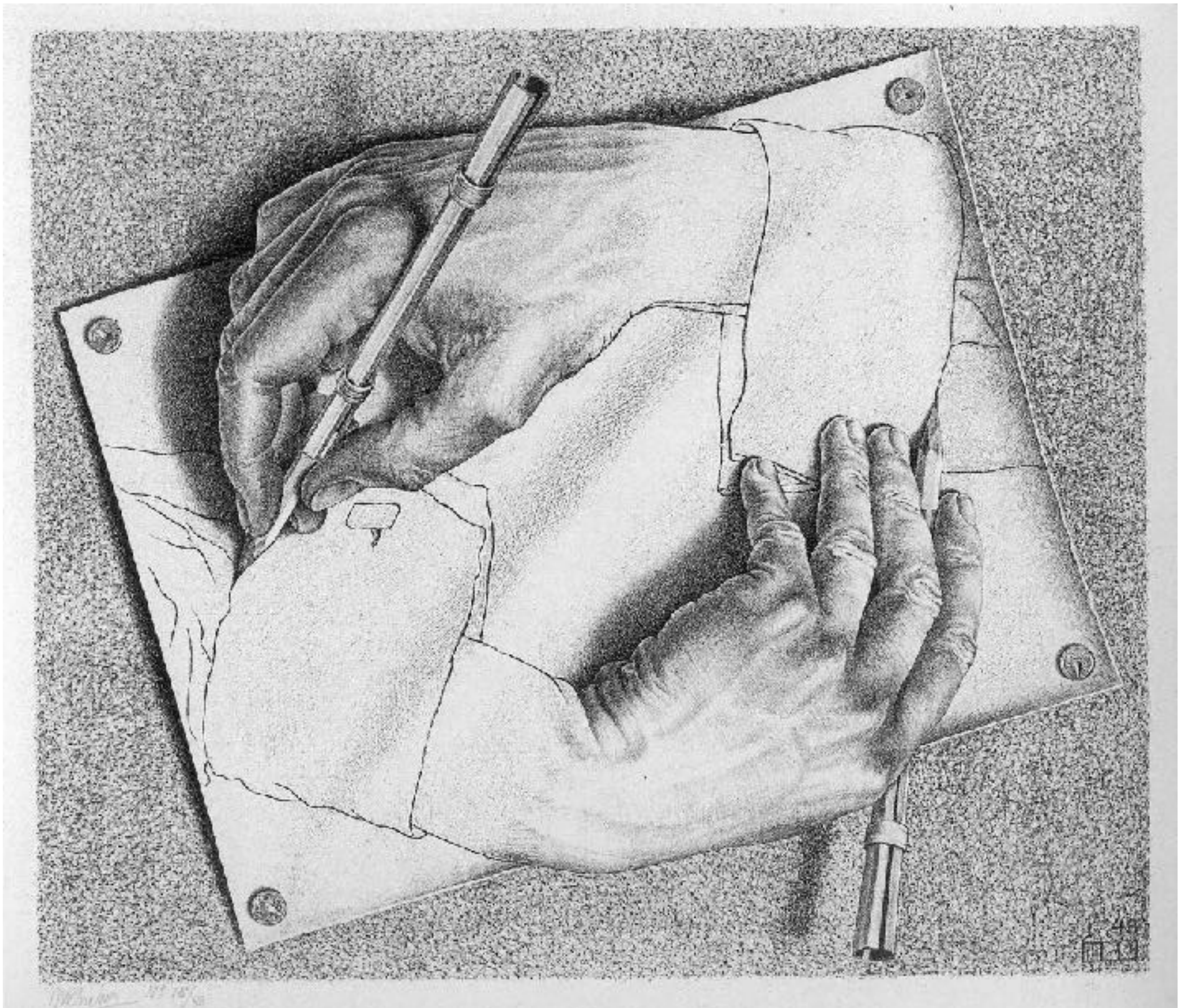
Fonctions récursives siamoises ?

- Une forme de récursivité implicite
- Parité d'un nombre version récursive ($n \geq 0$)

$$\texttt{estPair(n)} = \begin{cases} \texttt{VRAI} & \texttt{si n = 0} \\ \texttt{estImpair(n-1)} & \texttt{sinon} \end{cases}$$

$$\texttt{estImpair(n)} = \begin{cases} \texttt{FAUX} & \texttt{si n = 0} \\ \texttt{VRAI} & \texttt{si n = 1} \\ \texttt{estPair(n-1)} & \texttt{sinon} \end{cases}$$

estPair / estImpair



Maintenir un état avec des fonctions récursives ?

- Variables locales = portée restreinte à la fonction (ie. recréées à chaque appel !)
- Interdit d'utiliser des variables globales !
- Comment maintenir un état d'appel récursif en appel récursif ?

Solution: paramètres!

- Nécessité d’ajouter un paramètre pour transmettre l’état d’appel en appel
- Besoin d’une fonction auxiliaire pour respecter la spécification initiale
- Fonction principale réduite à l’appel de la fonction auxiliaire

Comparaison d’algorithmes récursifs

- Sans fonction auxiliaire et avec
- Différentes conceptions impliquent des différences de performances
- Objectif : **déterminer la plus “grande” lettre d’une chaîne**

Fonctions pratiques

- Tête et reste d'une chaîne (très pratique avec des fonctions récursives)

```
String tete(String c) {
  return substring(c,0,1);
}

String reste(String c) {
  return substring(c,1,length(c));
}
```

Premier algorithme

- Le plus grand élément d'une chaîne à un seul élément est cet élément.
- Le plus grand élément est le maximum entre son premier élément et le plus grand élément du reste de la chaîne.

$$\texttt{plusGrand1(c)} = \begin{cases} \texttt{tete(c) si longueur(c)=1} \\ \texttt{max(tete(c),plusGrand1(reste(c)))} \end{cases}$$

Deuxième algorithme

- Comparer deux éléments consécutifs, du premier au dernier, en mémorisant au fur et à mesure, le plus grand rencontré

```
plusGrand2(Chaine ch) => pG2(tete(ch), reste(ch))
```

$$\texttt{pG2(x, c)} = \begin{cases} \texttt{x si c est la chaîne vide} \\ \texttt{pG2(x , reste(c)) si tete(c) < x} \\ \texttt{pG2(tete(c), reste(c)) si x < tete(c)} \end{cases}$$

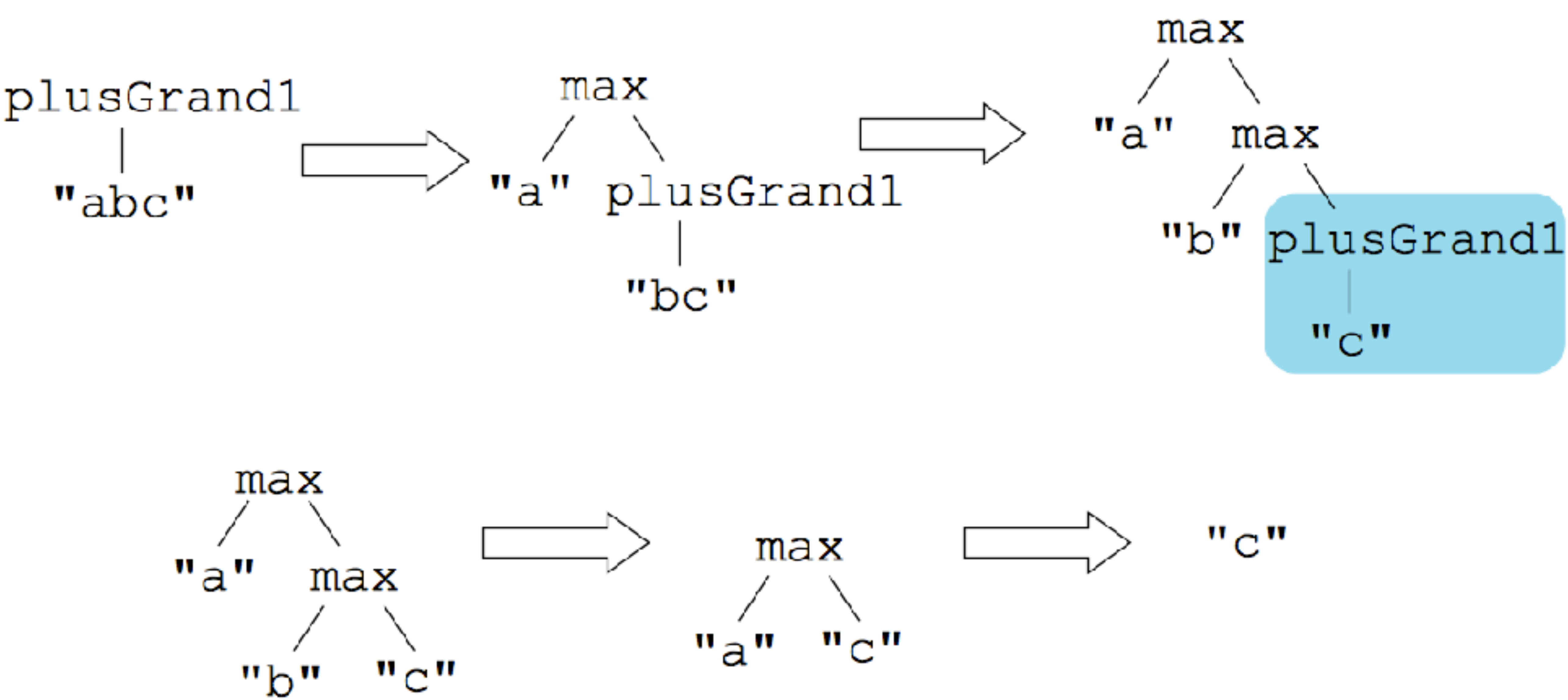
Troisième algorithme

- Même approche que le précédent mais en mémorisant le maximum actuel en tête de la chaîne

$$\texttt{plusGrand3(c)} = \begin{cases} \texttt{x} & \texttt{si longueur(c)=1} \\ \texttt{max(x,y)} & \texttt{si longueur(c)=2} \\ \texttt{plusGrand3(x+l')} & \texttt{si c = xyl' et x > y} \\ \texttt{plusGrand3(y+l')} & \texttt{si c = xyl' et y > x} \end{cases}$$

Premier algorithme

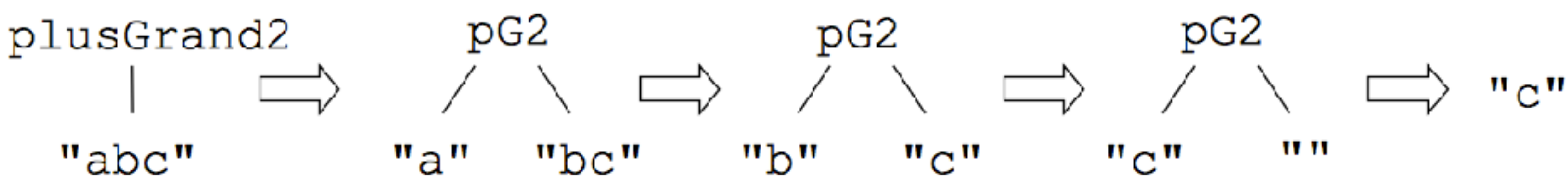
```
                  ⎧ tete(c) si longueur(c)=1
plusGrand1(c)  =  ⎨
                  ⎩ max(tete, plusGrand1(reste(c)))
```



Deuxième algorithme

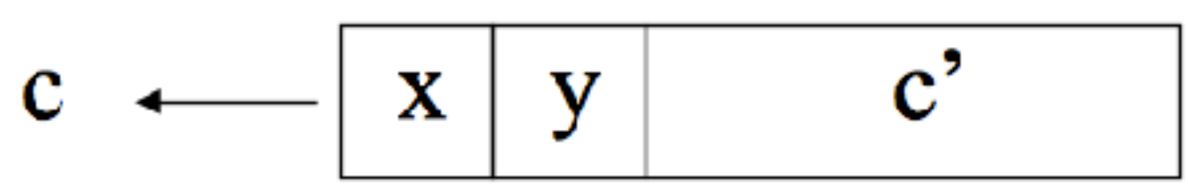
```
plusGrand2(Chaine ch) => pG2(tete(ch), reste(ch))
```

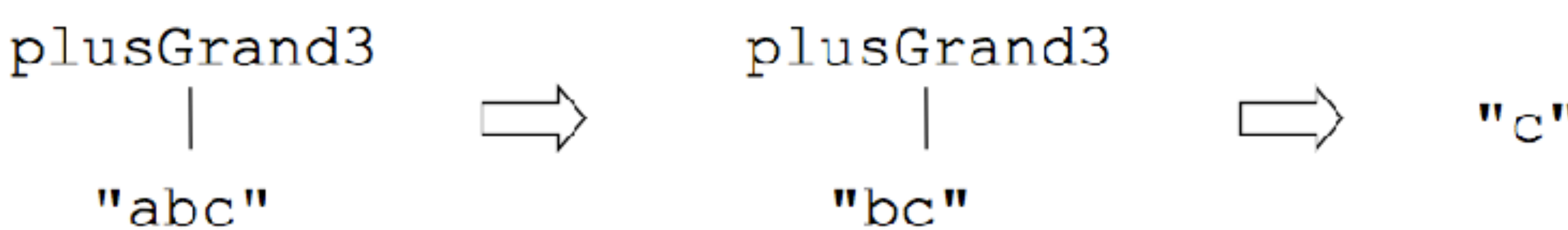
$$
\texttt{pG2(x, c)} = \begin{cases} \texttt{x si c est la chaîne vide} \\ \texttt{pG2(x , reste(c)) si tete(c) < x} \\ \texttt{pG2(tete(c), reste(c)) si x < tete(c)} \end{cases}
$$



Troisième algorithme

$$\texttt{plusGrand3(c)} = \begin{cases} \texttt{x} & \texttt{si longueur(c)=1} \\ \texttt{max(x,y)} & \texttt{si longueur(c)=2} \\ \texttt{plusGrand3(x+c')} & \texttt{si c = xyc' et x > y} \\ \texttt{plusGrand3(y+c')} & \texttt{si c = xyc' et y > x} \end{cases}$$





Conclusion ?

- Algo 1: moins performant car récursivité non terminale
- Algo 2 et 3: récursivité terminale
- Algo 3 le plus performant
- Importance des équations de récurrence utilisées pour résoudre le problème !

Concevoir un algo récursif

- Déterminer l'information sur laquelle porte la récurrence
- Identifier le ou les cas ou conditions d'arrêt (généralement des cas « dégénérés »)
- Définir les équations de récurrence pour le cas général
- Vérifier que l'algorithme s'arrête !
- Déterminer si la récursivité est terminale
- Si ce n'est pas le cas, modifier l'algorithme pour avoir une récursivité terminale

Récursivité directe/indirecte

- Deux types d'algorithmes récursifs
  - « *directs* » : proche des équations de récurrences (ex: calcul du nième terme d'une suite numérique)
  - « *indirects* » : nécessitant l'introduction de paramètres supplémentaires (ex: déterminer si il y a plus de 'e' que de 'a' dans une chaîne)
- Distinction repose sur la gestion d'un état lors des appels récursifs
  - Pour les « *directs* », pas d'état, les paramètres sont suffisants
  - Pour les « *indirects* », un état, qui doit être rajouté en paramètre, ce qui entraîne l'**introduction d'une fonction auxiliaire**

Quelques exemples

- Le retour du palindrome
- Le monde merveilleux des fractales
- Plus de 'a' que de 'e' ?
- La vengance du palindrome ...

Palindrome, le retour

- Si la chaîne ne contient qu'un caractère ou aucun alors c'est un palindrome,
- Sinon on vérifie que la première et la dernière sont identiques et on rappelle palindrome avec la chaîne privée de son premier et dernier caractère

Palindrome en récursif

```
boolean palindrome(String ch) {
  if (length(ch) < 2) {
    return true;
  } else if (charAt(ch, 0) != charAt(ch, length(ch)-1)) {
    return false;
  }
  return palindrome(substring(ch, 1, length(ch)-1));
  }
}
```

```
palindrome        palindrome
    |       ⇨         |        ⇨   FAUX
 "roder"            "ode"

palindrome        palindrome        palindrome
    |       ⇨         |       ⇨         |        ⇨   VRAI
 "radar"            "ada"              "d"
```

Synthèse

- **Identifier l'information sur laquelle peut porter la récurrence**
- **Identifier les cas de base/conditions d'arrêt**
- **Tendre vers les cas d'arrêts lors des appels récursifs**
- Se poser la question de la récursivité terminale ou non
- Etat à gérer ? Introduction d'une fonction auxiliaire

M.C. Escher