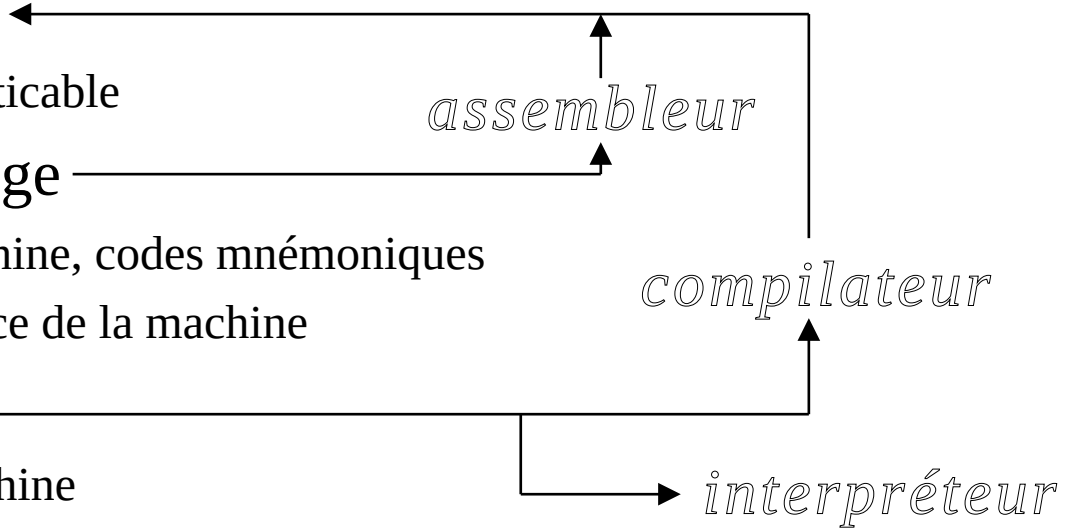
La traduction des langages de programmation

Principes (1)

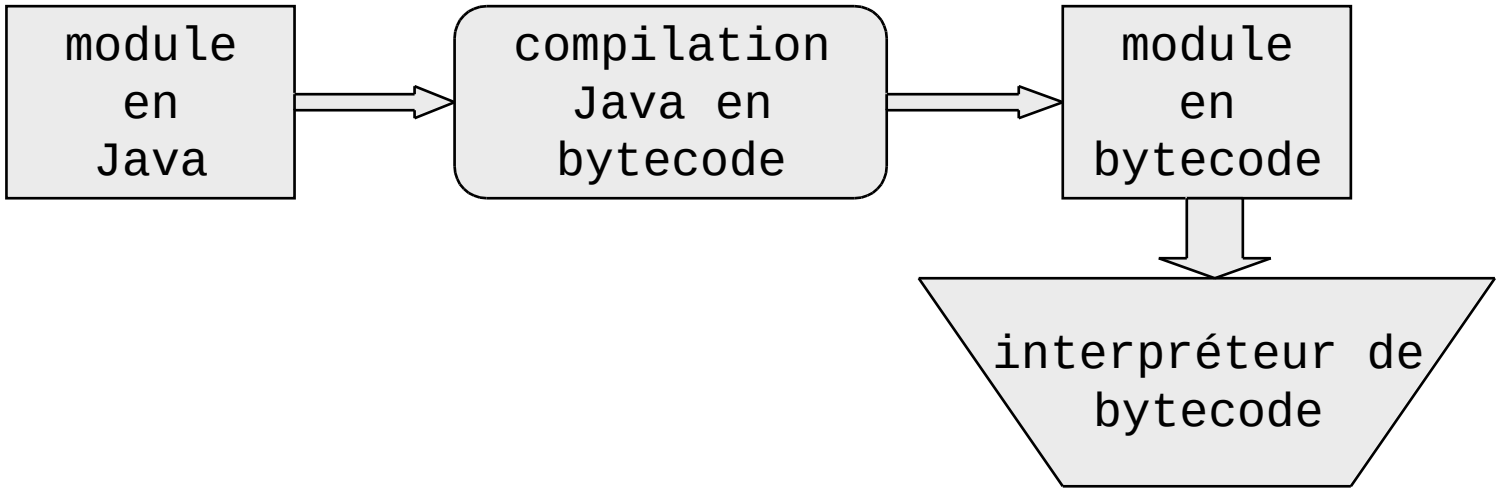
- Langage machine
  - ♦ constitué de bits, impraticable
- Langage d ’assemblage
  - ♦ proche du langage machine, codes mnémoniques
  - ♦ nécessite la connaissance de la machine
- Langage évolués
  - ♦ indépendants de la machine
  - ♦ expression plus proche des besoins des programmeurs
  - ♦ spécialisation en fonction des problèmes
    - – FORTRAN: applications scientifiques
    - – COBOL: applications de gestion



traducteur *module source* ⟹ *module objet*

Principes (2)

- Interpréteur = programme qui simule une machine qui comprend le langage
- Combiner compilation et interprétation
- Exemple:



Analyse lexicale (1)

- module source
  - suite de caractères,
  - représentation d ’une suite de symboles
- Retrouver les symboles à partir des caractères
  - exemple: `begin x := 23 ; end`
  - retrouver le regroupement:

begin	x	:=	23	;	end

- Coder les symboles dans une représentation interne
- Implique une structure lexicale non ambiguë

Analyse lexicale (2)

identificateur: `lettre { lettre | chiffre | _ }`

constante numérique: `chiffre { chiffre | lettre}`

mots clés: comme identificateur, mais mots réservés

symboles spéciaux:

caractères spéciaux habituels ou combinaison

exemples: `+` `-` `*` `/` `=` `>=` `:=` etc…

espaces, tabulations, fin de ligne

sans signification, si ce n'est séparateur de symboles

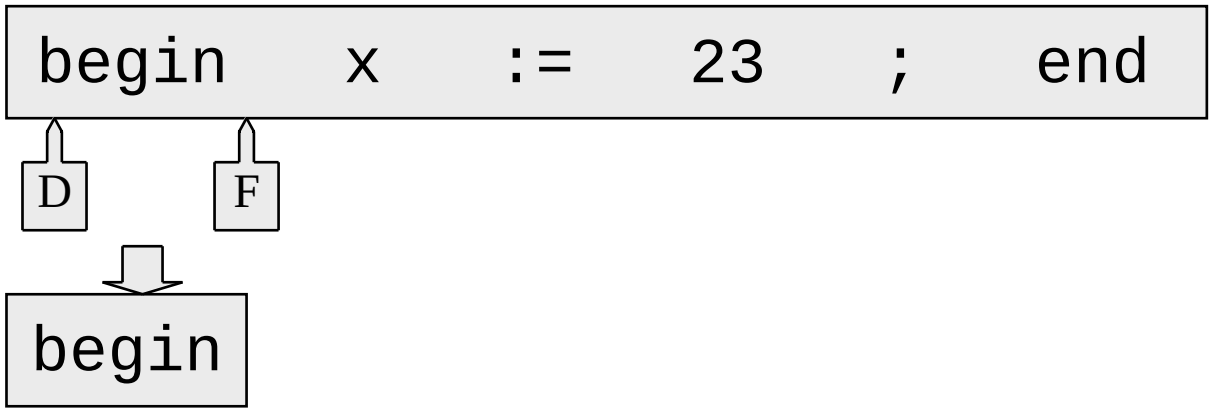
exemples entre "`begin`" et "`x`"

Analyse lexicale (3)

- Principe de l'analyse
  - sachant où commence un symbole,
  - rechercher où il se termine
  - chercher où commence le suivant
- Générateurs d'analyseur lexical (`lex`)
  - on fournit la description de la structure lexicale d'un langage L
  - le résultat est un programme qui fait l'analyse lexicale de L

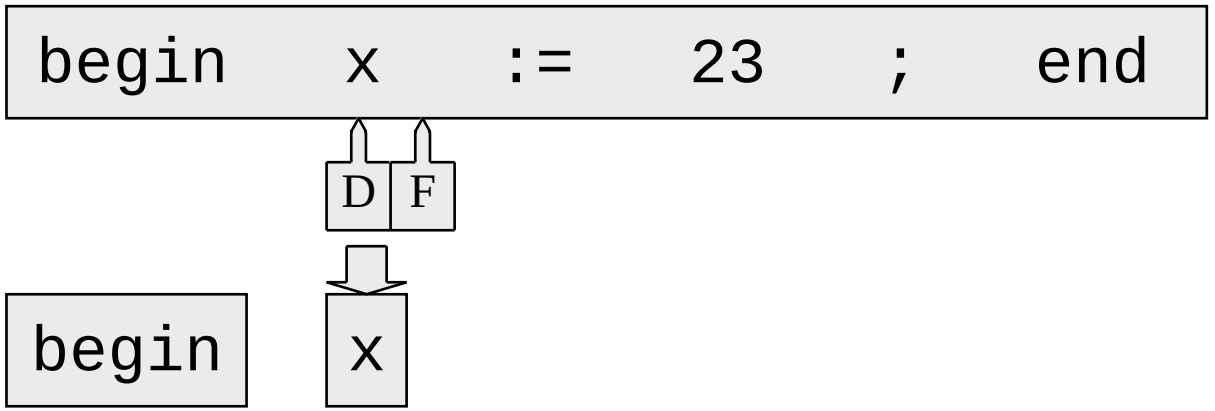
Analyse lexicale (4)

- début sur le "`b`"
- fin sur l'espace qui suit "`begin`"
- il s'agit d'un mot clé



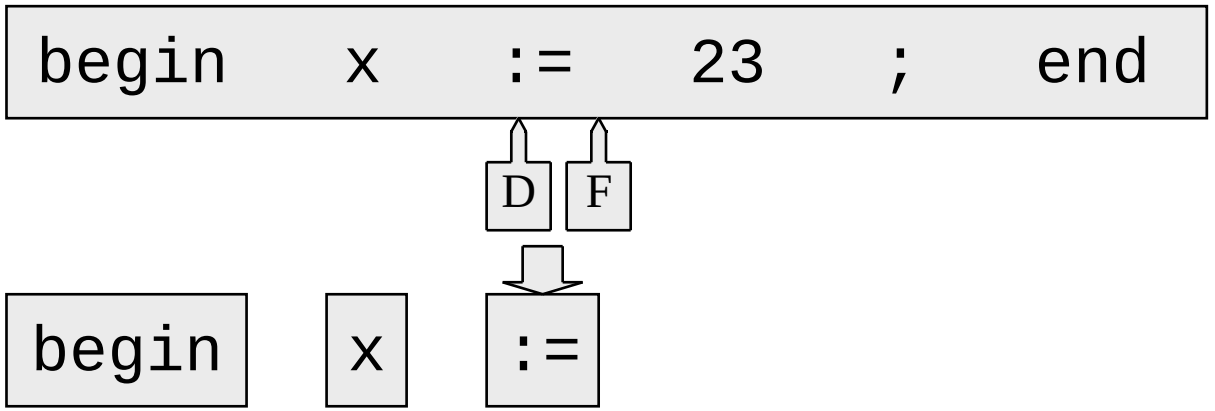
Analyse lexicale (5)

- recherche du début suivant sur le "x"
- fin sur l'espace qui suit "x"
- identificateur



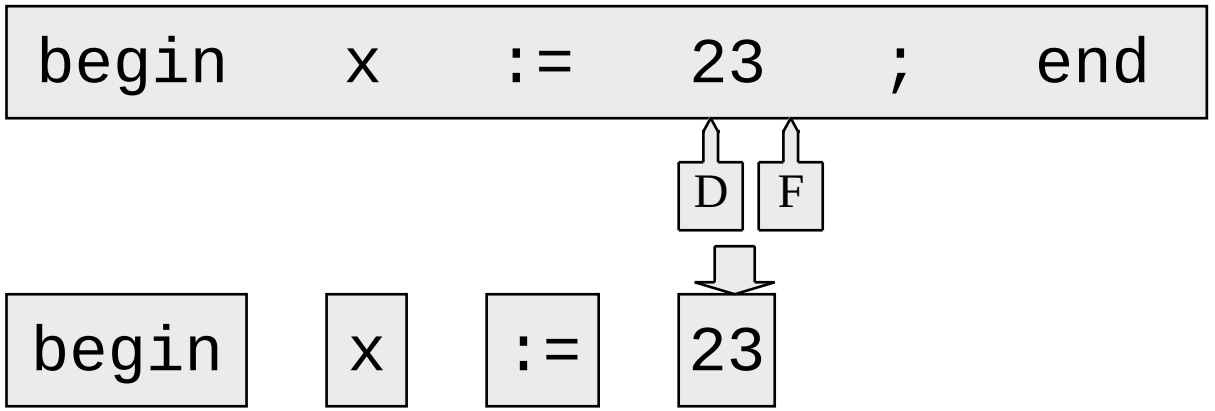
Analyse lexicale (6)

- recherche du début suivant sur le "`:`"
- fin sur l'espace qui suit "`:=`"
- symbole spécial composé



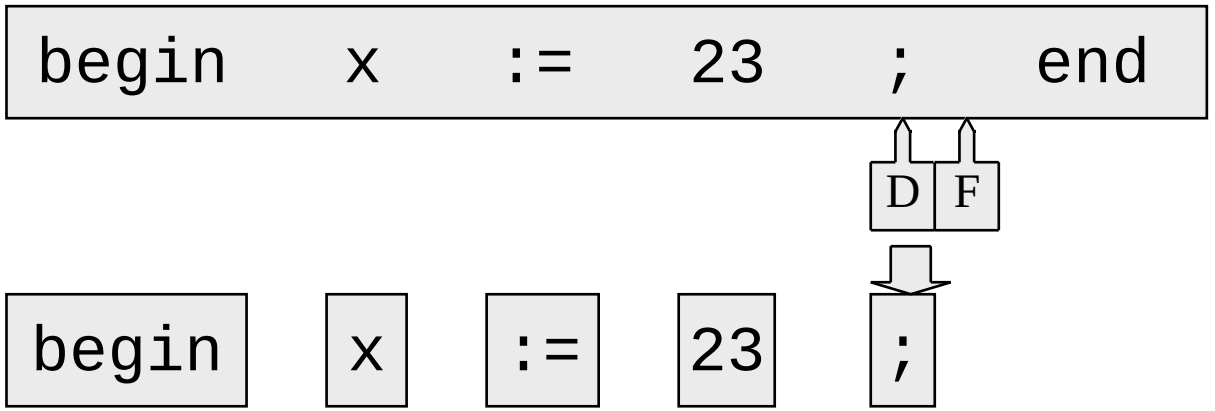
Analyse lexicale (7)

- recherche du début suivant sur le "2"
- fin sur l'espace qui suit "23"
- nombre, qui pourrait être codé par sa valeur



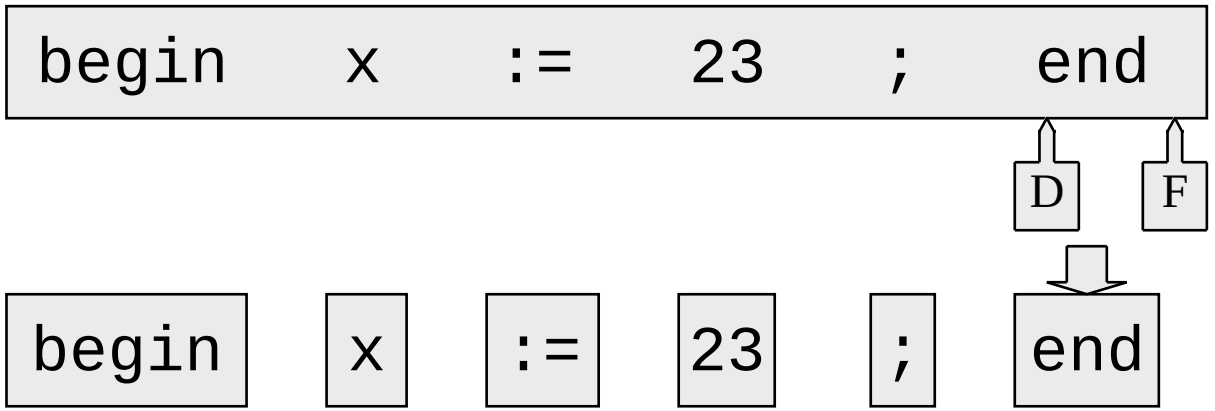
Analyse lexicale (8)

- recherche du début suivant sur le "`;`"
- fin sur l'espace qui suit "`;`"
- symbole spécial



Analyse lexicale (9)

- recherche du début suivant sur le "`e`"
- fin sur l'espace qui suit "`end`" ou la fin de ligne
- il s'agit d'un mot clé



Analyse syntaxique (1)

- Vérifier la concordance de la suite de symboles avec la structure du langage
- *règles de production*
  - règles de construction d'une suite de symboles du langage
- *analyse syntaxique*
  - retrouver les règles utilisées par le programmeur
- Syntaxe
  - non ambiguë
  - indépendante du contexte
  - doit faciliter cette reconstruction

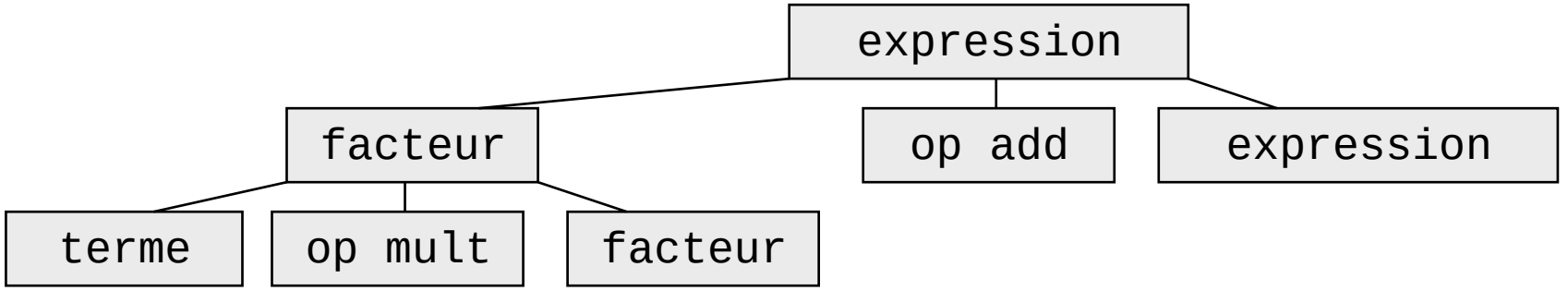
Analyse syntaxique (2)

- Forme normale de Backus-Naur (nombre restreint de signes)
  - "<" et " >" encadrent une dénomination syntaxique
  - "::=" peut s'énoncer comme *se réécrit en*
  - "|" énonce une alternative
  - "{", "}" la séquence entre les deux peut être répétée 0 fois ou plus
  - "[", "]" la séquence entre les deux est une option (0 ou 1 fois)
- Exemple

```
<expression> ::= <facteur> | <facteur> <op add> <expression>
<facteur>    ::= <terme> | <terme> <op mult> <facteur>
<terme>      ::= <identificateur> | <nombre> | ( <expression> )
<op add>     ::= + | -
<op mult>    ::= * | /
```

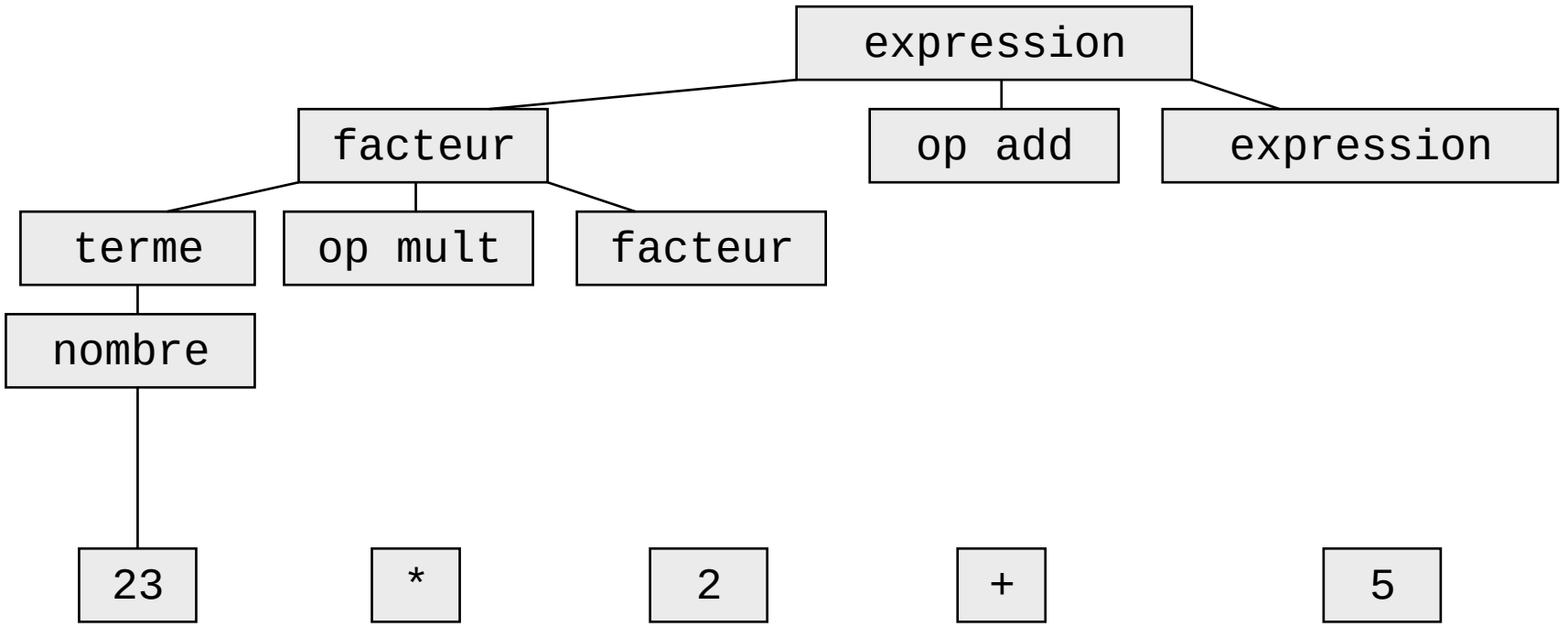
Analyse syntaxique (3)

- développement de `expression`
- développement de `facteur`



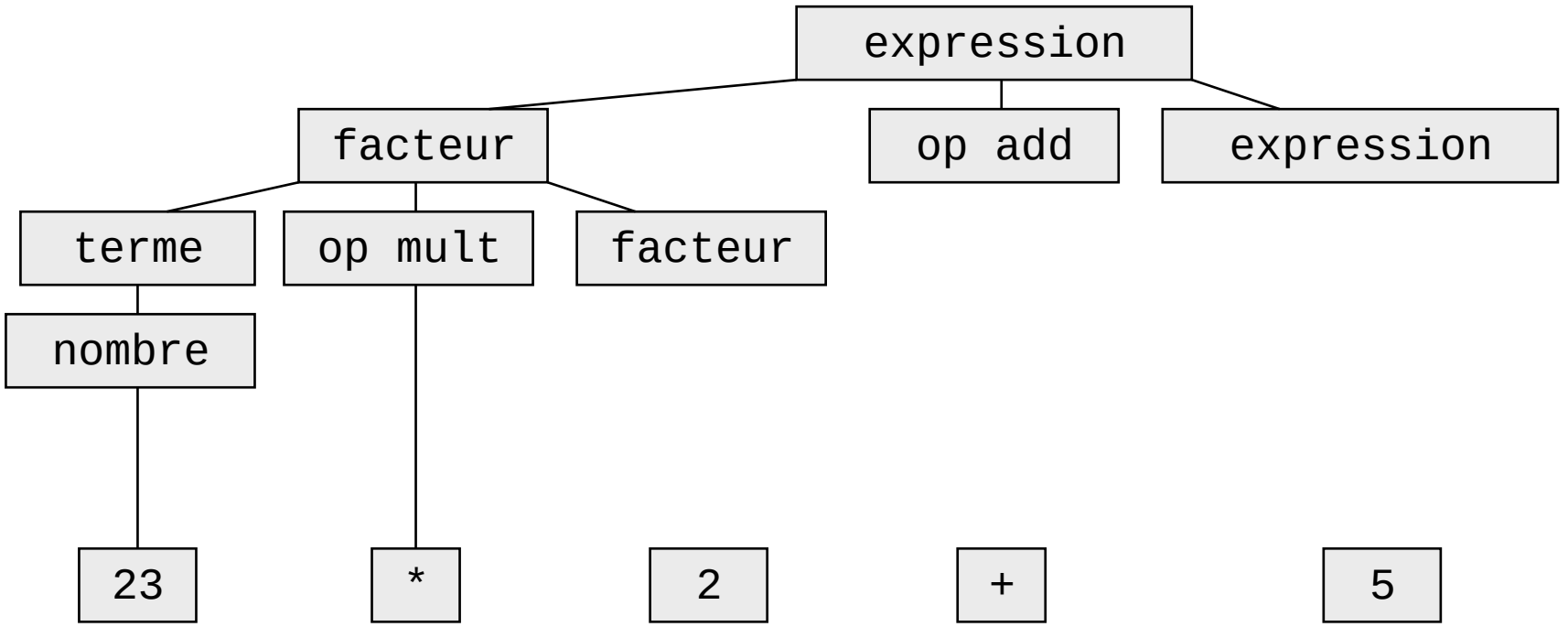
Analyse syntaxique (4)

- `terme` se développe en `nombre` qui correspond au symbole `23`



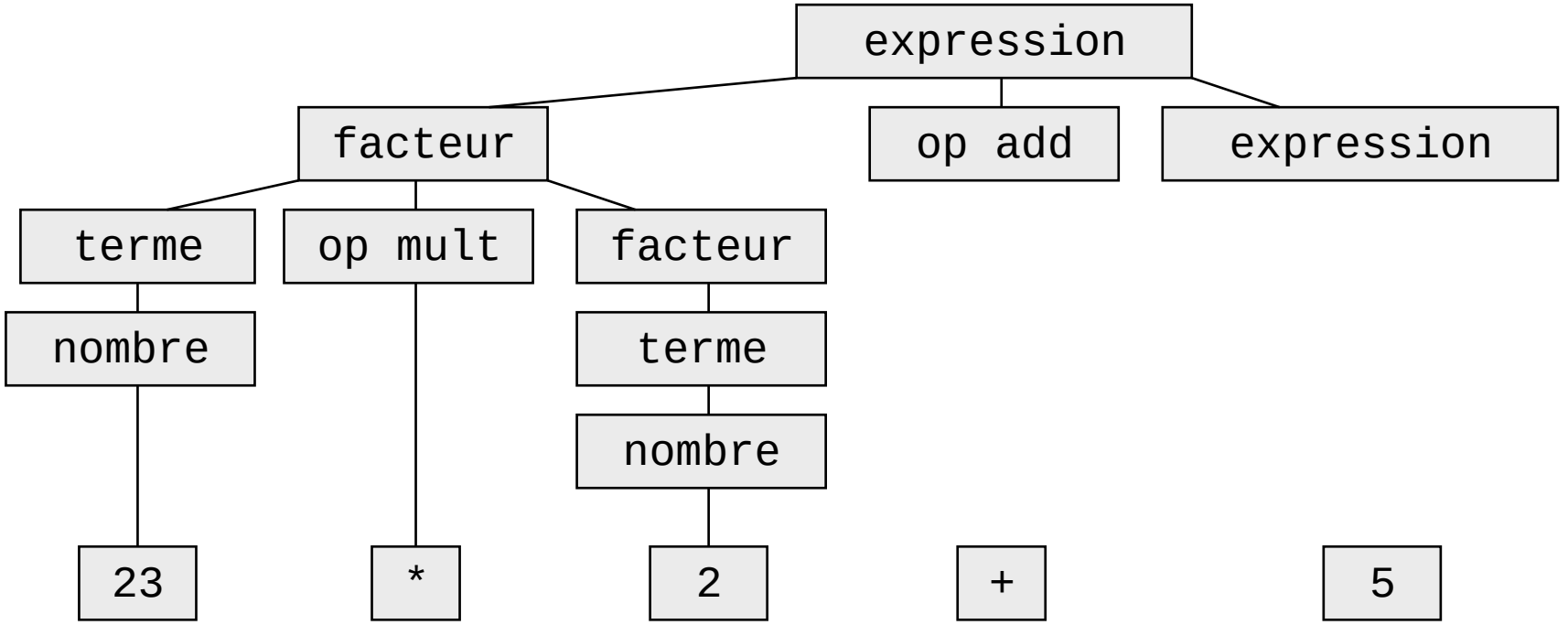
Analyse syntaxique (5)

- `op mult` correspond au symbole `*`



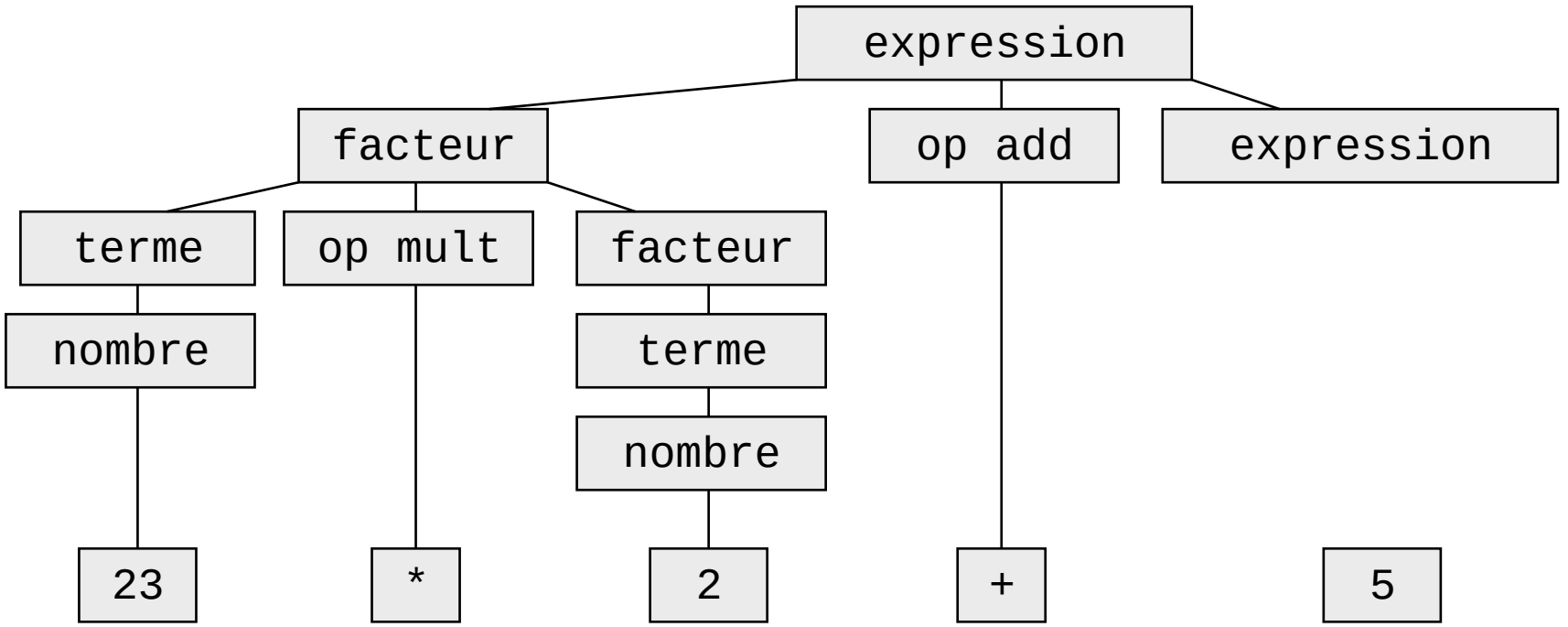
Analyse syntaxique (6)

- `facteur` se développe en `terme`, puis `nombre` qui correspond au symbole `2`



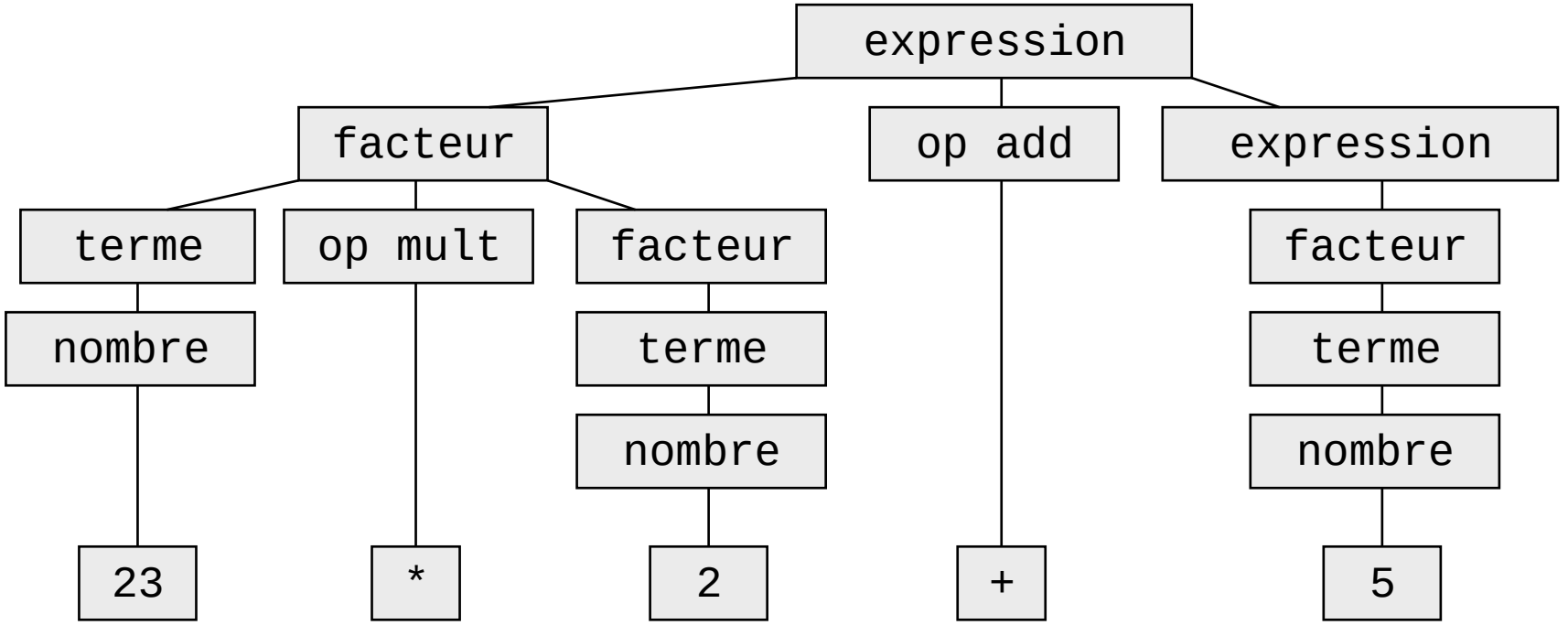
Analyse syntaxique (7)

- `op add` correspond au symbole `+`



Analyse syntaxique (8)

- `expression` de droite se développe en `facteur`, puis `terme`, puis `nombre` qui correspond au symbole `5`
- Remarque: la priorité des opérateurs est dans la syntaxe

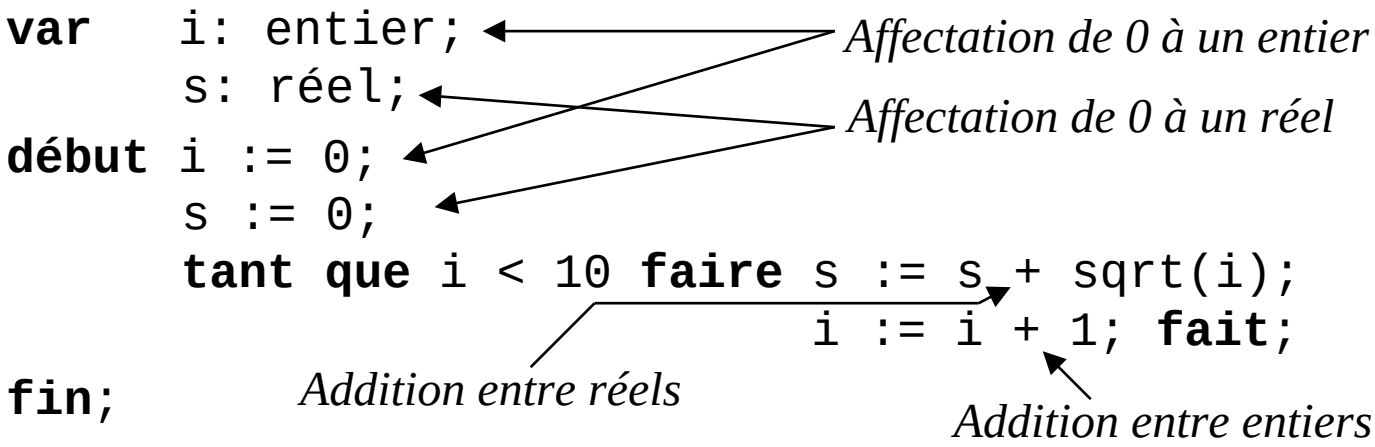


Analyse sémantique (1)

- trouver les objets manipulés par le programme
  - ♦ désignés par des identificateurs
- trouver les propriétés d'un objet
  - ♦ Comment?
    - – déduites implicitement de l'utilisation
    - – à partir de déclarations explicites
  - ♦ son *type* => sémantique des opérations, combinaison d'opérations machine
  - ♦ sa *durée de vie* => quand doit-il être créé ou détruit
  - ♦ sa *taille* => nombre d'emplacements mémoire
  - ♦ son *adresse* => désignation dans les instructions machine

Analyse sémantique (2)

- trouver les actions du programme sur ces objets

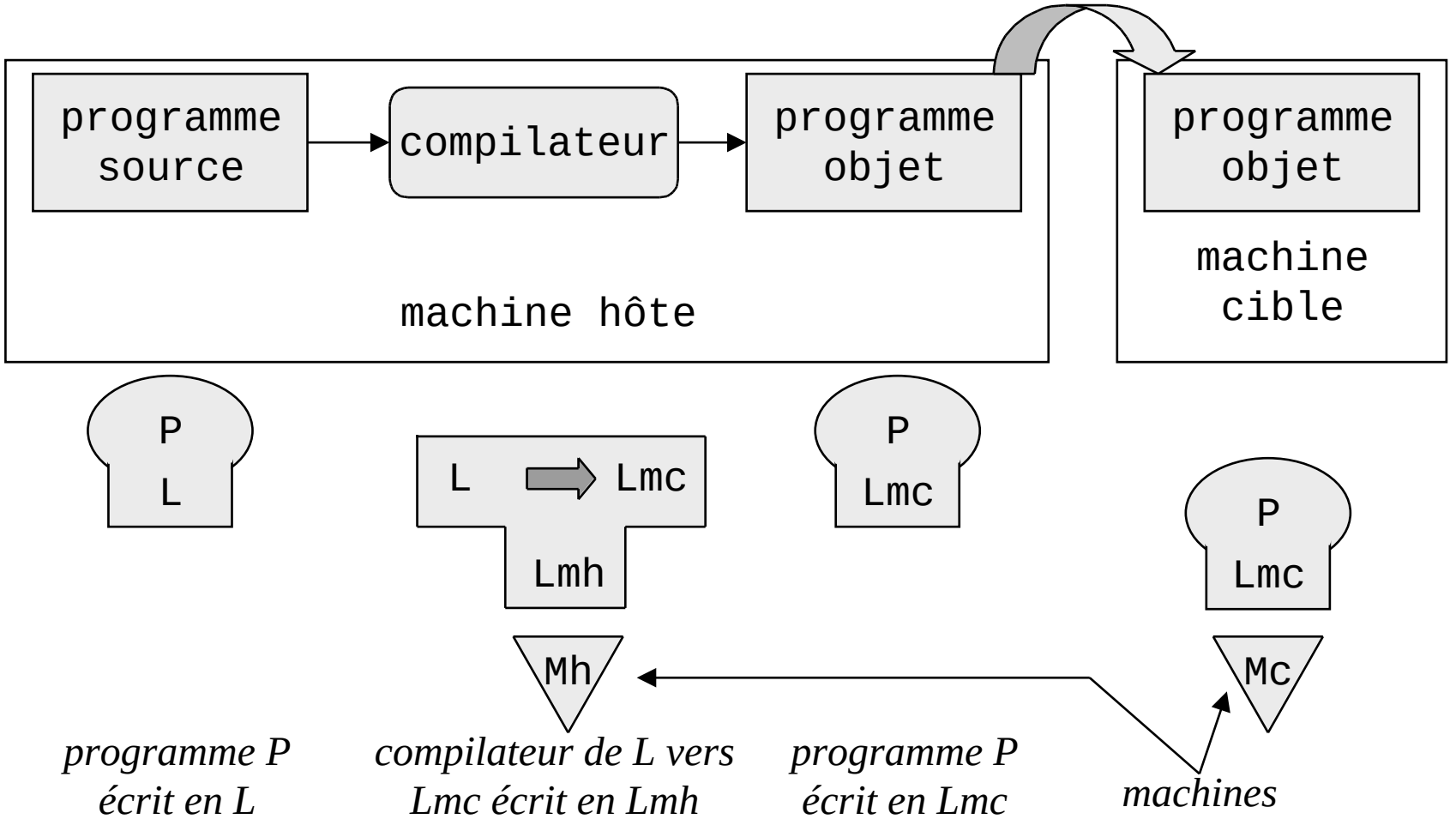


- Contrôle:
  - déclarations présentes, utiles et compatibles
  - signification des expressions, etc...
- Problème: la sémantique est-elle celle désirée => unicité

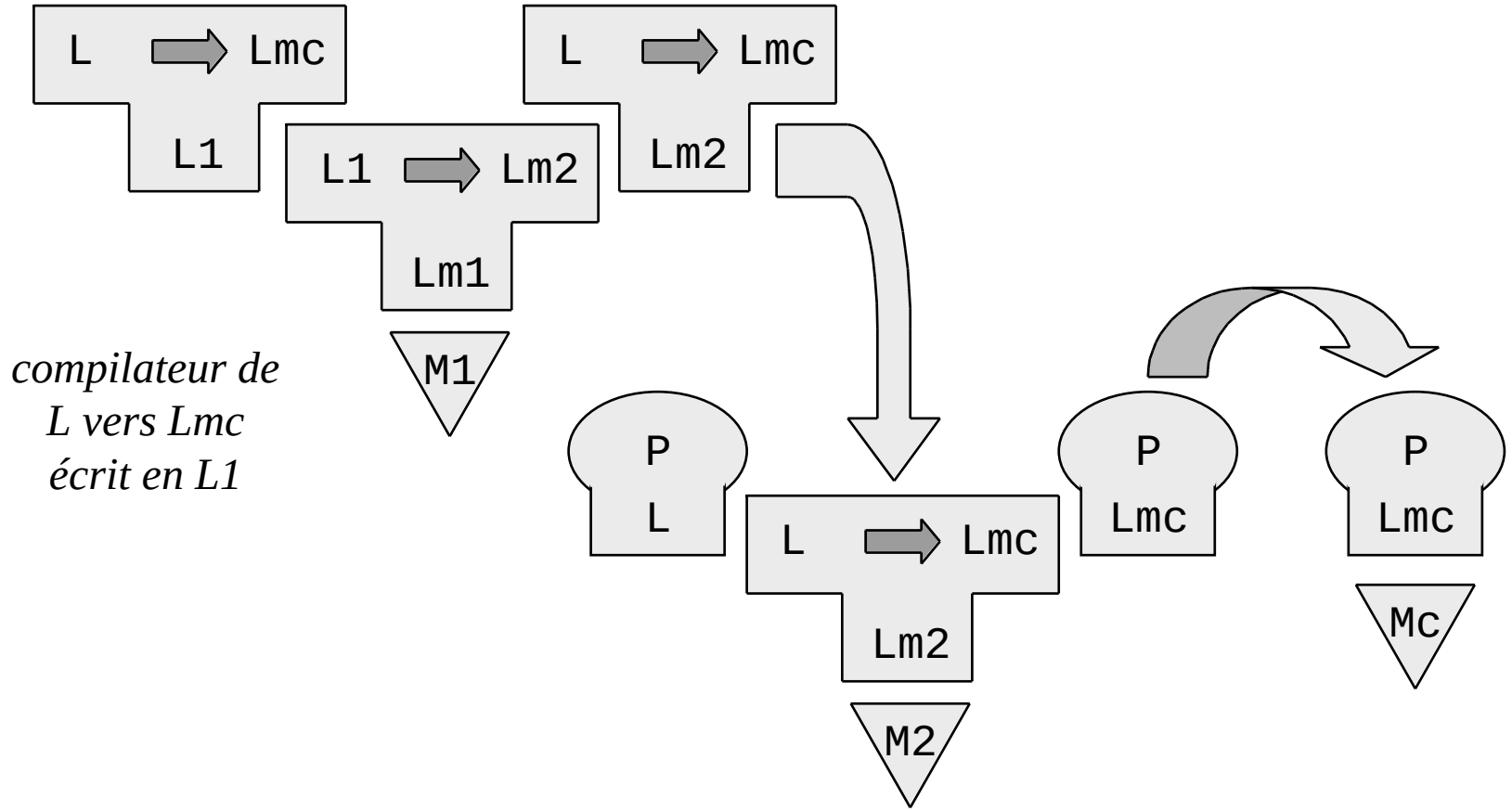
Génération de code et optimisation

- Génération
  - construction du programme machine équivalent
- Optimisation
  - améliorer l'efficacité du résultat produit (*et non le programme lui-même*)
  - compenser la perte d'efficacité due à la programmation de haut niveau
  - mesure de la qualité d'un compilateur
- Exemples
  - expressions constantes évaluées à la compilation
  - élimination du code inaccessible => compilation conditionnelle
  - sortir des boucles les instructions ayant même résultat
  - remplacer les calculs liés aux variables de boucles par d'autres + essentiels => progression d'adresses au lieu d'indice de tableau

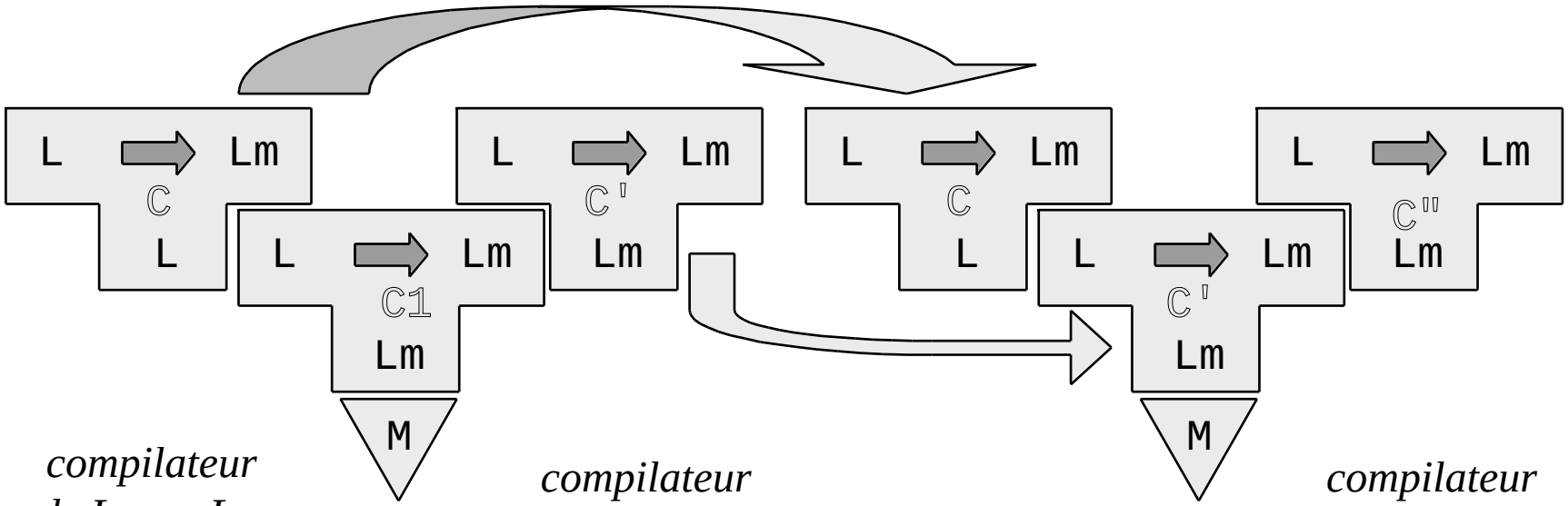
Traduction croisée (1)



Traduction croisée (2)



Autocompilation

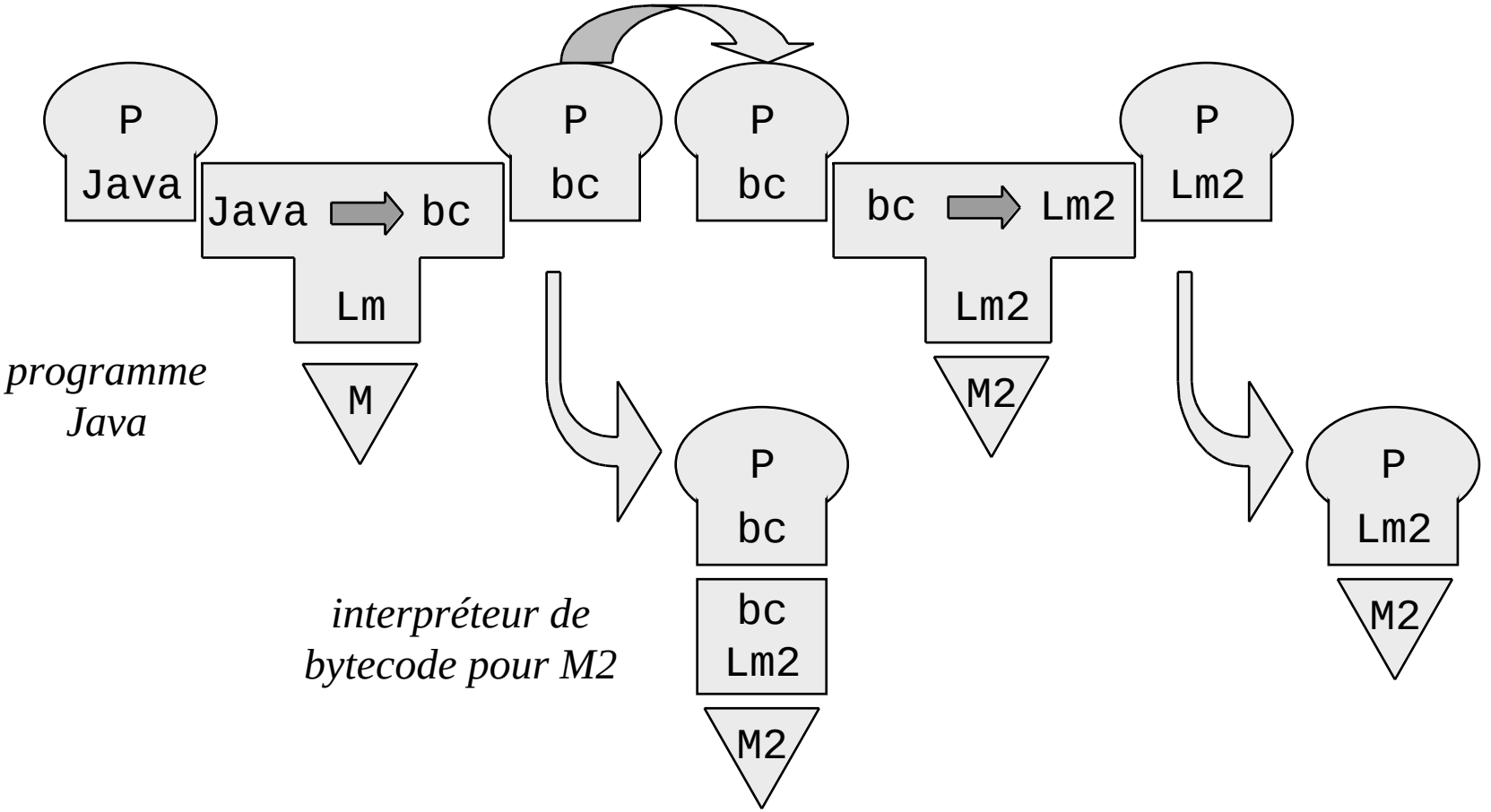


compilateur de L vers Lm écrit en L

compilateur de L vers Lm écrit en Lm qualité de C1

compilateur de L vers Lm écrit en Lm qualité de C

Système Java



Conclusion

- Résume des phases de traduction
  - analyse lexicale
  - analyse syntaxique
  - analyse sémantique
  - optimisation
  - génération de code
- Tout programme qui interprète des données venant d'un utilisateur devrait comporter les trois premières phases