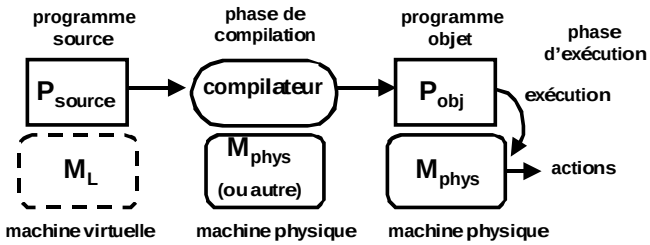
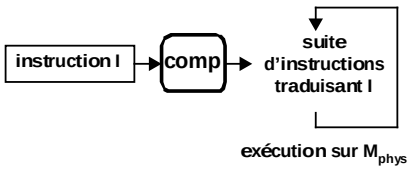
## Schémas d'exécution d'un programme

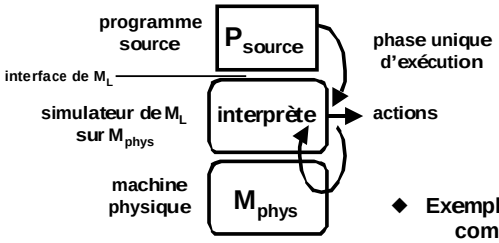
- Un programme $P_{source}$ dans un langage (impératif) L décrit une suite d'actions à exécuter par une machine (virtuelle) $M_L$
  - Le programme est une suite d'instructions dont chacune spécifie une action (passage d'un état initial à un état final)
  - La machine $M_L$ est capable d'exécuter (interpréter) le programme $P_{source}$, c'est-à-dire de traduire en actions de $M_L$ la suite d'instructions de $P_{source}$.
- Comment faire si on ne dispose pas de la machine $M_L$, mais d'une machine différente, $M_{phys}$ ?
  - Deux solutions de base
    - a) Traduire le programme $P_{source}$ dans un programme $P_{obj}$ "équivalent" pour la machine $M_{phys}$, et faire exécuter $P_{obj}$ sur $M_{phys}$ ("équivalent" = qui a le même effet). C'est un schéma de compilation (la traduction de $P_{source}$ en $P_{obj}$ est faite par un compilateur)
    - b) Construire (par programme) sur $M_{phys}$ un simulateur de la machine $M_L$, et faire exécuter le programme original $P_{source}$ sur ce simulateur. C'est un schéma d'interprétation (le simulateur de $M_L$ sur $M_{phys}$ est un interprète)
  - Une solution mixte est possible
    - compiler $P_{source}$ pour une machine intermédiaire $M_{int}$ ; construire un interprète de $M_{int}$ sur $M_{phys}$

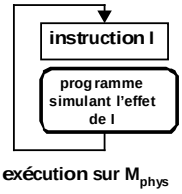


## Compilation et interprétation : principe




- Exemples
  compilation : C, C++, Fortran, Ada
  interprétation : lang. machine, *shell* Unix, Tcl-TK, PostScript
  l'un ou l'autre : Lisp, Scheme
  schéma mixte : Java (cf plus loin), SmallTalk

## Compilation et interprétation : comparaison

**Compilation**

- **Efficacité**
  - le code engendré s'exécute directement sur la machine physique
  - ce code peut être optimisé
- **Mise au point**
  - pas toujours facile de relier une erreur d'exécution au texte source
- **Cycle de modification - réexécution**
  - toute modification du texte source impose de refaire le cycle complet (compilation, édition de liens, exécution)

**Interprétation**

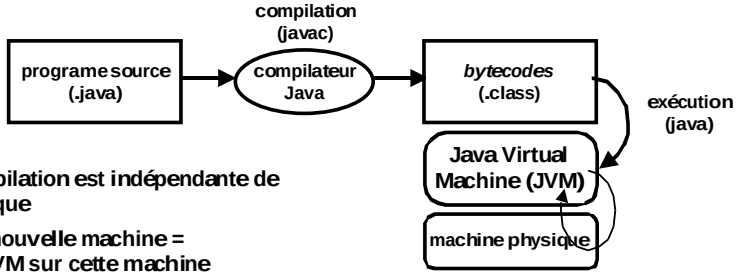
- **Efficacité**
  - l'interprétation directe est souvent longue (appel de sous-programmes)
  - pas de gain sur les boucles
  - facteur de x10 à x100 ...
- **Mise au point**
  - lien direct entre instruction et exécution
  - possibilités étendues d'observation et trace intégrées
- **Cycle de modification - réexécution**
  - cycle très court (modifier et réexécuter)

L'augmentation de puissance des processeurs amène un regain des techniques d'interprétation



## Schéma mixte d'exécution

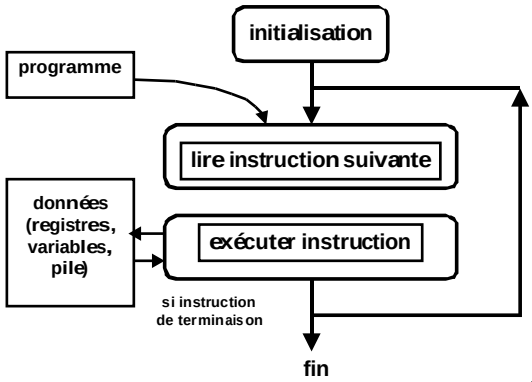
- **Objectifs**
  - Essayer de combiner les avantages des schémas de compilation et d'interprétation
  - Améliorer la portabilité des programmes entre machines, via un langage intermédiaire standard
- **Principe**
  - Définir un langage intermédiaire et une machine virtuelle capable d'interpréter ce langage
  - Écrire un compilateur du langage initial (source) vers le langage intermédiaire
  - Écrire un interprète du langage intermédiaire (c'est-à-dire un simulateur de la machine virtuelle)
- **Exemple : Java**




  - La phase de compilation est indépendante de la machine physique
  - Portage sur une nouvelle machine = réécriture de la JVM sur cette machine

## Principe de fonctionnement d'un interprète (1)

- **Définition de la machine virtuelle**
  - Éléments du "pseudo-processeur" (analogie avec processeur physique)
    - pseudo-registres (zones de mémoire réservées)
    - pseudo-instructions (réalisées par une bibliothèque de programmes)
  - Structures d'exécution
    - allocation des variables
    - pile d'exécution
- **Cycle d'interprétation**
  - Analogie avec cycle d'un processeur
  - Pseudo-compteur ordinal






## Principe de fonctionnement d'un interprète (2)

- **Raffinement de la "boîte" *exécuter instruction***
  - Format de l'instruction : instruction ::= <code opération>, <suite de paramètres>
  - Opérations

```
Décoder instruction (isoler code op., paramètres)
selon code op faire
        code1 : exécuter programme de code1 (paramètres)
        ...
        coden : exécuter programme de coden (paramètres)
```

  - Exécution du programme correspondant à un code op.

```
Dépend de l'instruction, mais quelques points communs
        allouer éventuellement place en mémoire pour données
        charger paramètres dans registres ou pile
        exécuter l'opération sur les paramètres
        mettre éventuellement à jour les résultats en mémoire
        déterminer l'instruction suivante à exécuter
```

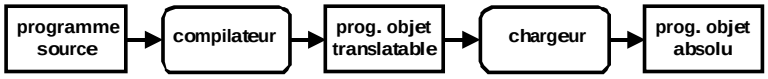
  - Autres opérations possibles

```
Observation, mise au point (exécution pas à pas), trace
Transformation préalable du programme dans une forme adaptée à
l'exécution (postfixé, etc.)
```

## Cycle de vie d'un programme compilé

- **Compilation vers programme objet absolu (adresses fixées en mémoire)**


  - contrainte : le programme ne peut pas être déplacé en mémoire (par ex. pour le combiner avec d'autres)
- **Compilation vers programme objet translatable (adresses définies à une translation près)**




  - nécessité de passer par un chargeur pour rendre le programme exécutable (le programme translatable n'est pas exécutable tel quel)
- **Exemples**
  - gcc -c prog.c ; produit un programme objet translatable dans le fichier prog.o
  - gcc -o prog prog.c ; produit un programme objet absolu dans le fichier prog (appelle le compilateur et le chargeur)



## Code exécutable et chargeurs

Nous illustrons le fonctionnement du chargeur avec un format simple d'instructions machine

<code-op> <registre><adresse>

Programme objet absolu (exécutable) : toutes les adresses sont absolues
Programme objet translatable (ou relogeable) : les adresses sont relatives au début du programme

```
programme
        ...
        BRN,R1 TOTO
        ...
TOTO:   LI, R2   5
        ...

BRN = branchement si R<0
LI = chargement immédiat
```

absolu (à partir de l'adresse 24000)

| | | | |
|---|---|---|---|
| | | | 24000 |
| 48 | 1 | 25010 | 24200 |
| | | | |
| 87 | 2 | 5 | 25010 |
| | | | |
| | | | |
| | | | |

translatable

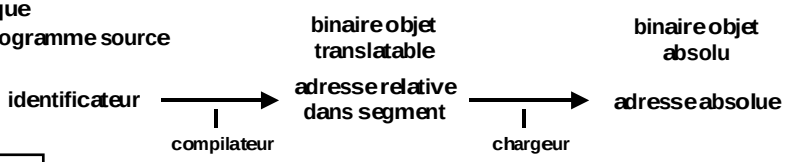
| | | | |
|---|---|---|---|
| | | | 0 |
| 48 | 1 | 1010 | 200 |
| | | | |
| 87 | 2 | 5 | 1010 |
| | | | |
| | | | |
| | | | |

## Fonctionnement d'un chargeur

- **Format du binaire translatable**
  - toutes les informations dans la partie adresse ne sont pas translatables
  - il faut donc un indicateur associé à toute adresse, signalant si elle est translatable ou non ; cet indicateur est placé par le compilateur lors de la génération du code
- **Fonctionnement du chargeur**
  - l'adresse absolue de chargement *A* est fournie en paramètre
  - algorithme du chargeur:

> copier le segment *St* contenant le binaire translatable dans un segment *Sa*
> pour toute adresse *a* contenue dans le segment *Sa* faire
> si *a* translatatable alors *a* = *a* +*A*
> le segment *Sa* contient maintenant le binaire absolu

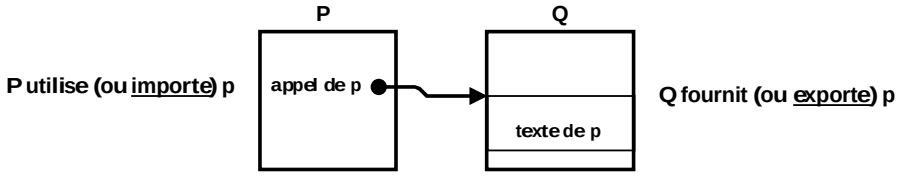
- **Illustration de la notion de liaison**
  - liaison = établissement de la relation entre une entité (abstraite) et sa réalisation physique



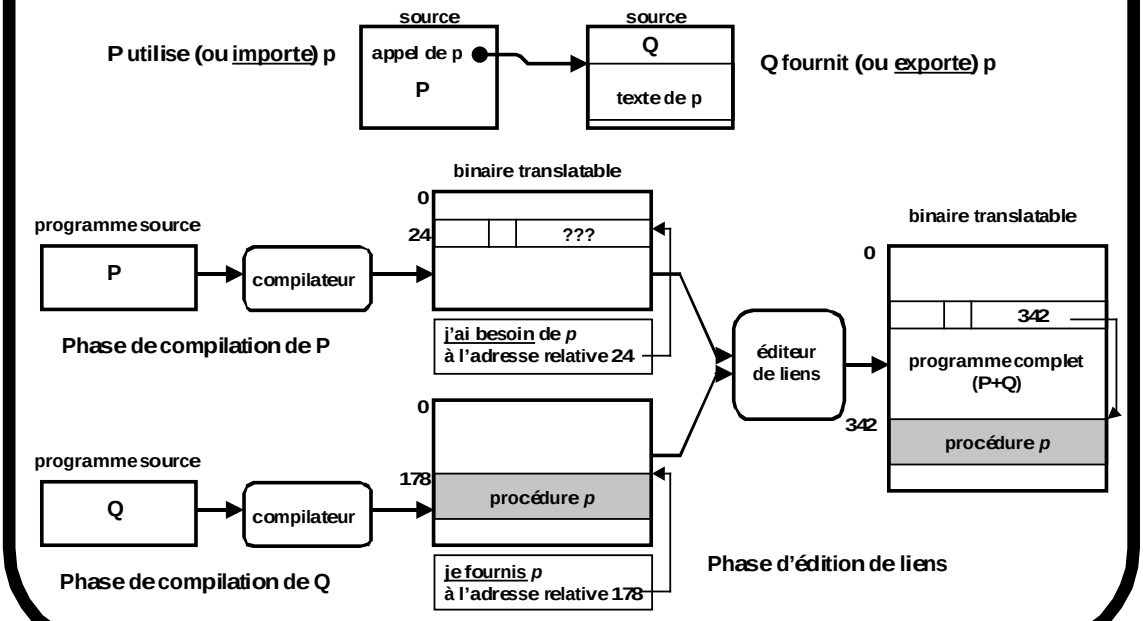



## Composition de programmes

- **Pourquoi parler de composition de programmes ?**
  - La plupart des applications sont réalisées par assemblage de parties
    - construction modulaire, facilite la conception et l'évolution des programmes
  - La plupart des applications utilisent des fonctions founies par des "bibliothèques"
  - Conséquence : un programme ne s'exécute pratiquement jamais "seul"
- **Le problème de la composition : encore un exemple de liaison**
  - Soit un programme P faisant appel à une procédure p incluse dans un programme Q construit séparément
  - Question : lors de la compilation de P, quelle adresse va-t-on associer à p ?
  - Réponse : il est impossible de le savoir tant qu'on ne connaît pas le programme Q
  - La liaison dans P entre la procédure p et son adresse ne peut être faite que dans une phase postérieure à la compilation, utilisant simultanément P et Q : c'est l'édition de liens



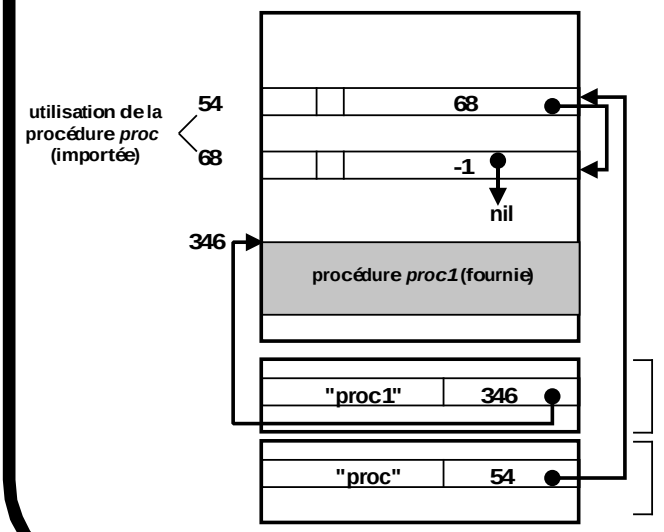
## Fonctionnement d'un éditeur de liens (1)






## Fonctionnement d'un éditeur de liens (2)

■ **Format du binaire translatable (très simplifié)**




- En général, un programme importe certains noms et en exporte d'autres
- Dans le binaire translatable, le compilateur crée une table des références externes pour les noms importés et une table des définitions externes pour les noms exportés.
- Ces tables sont utilisées par l'éditeur de liens

## Fonctionnement d'un éditeur de liens (3)

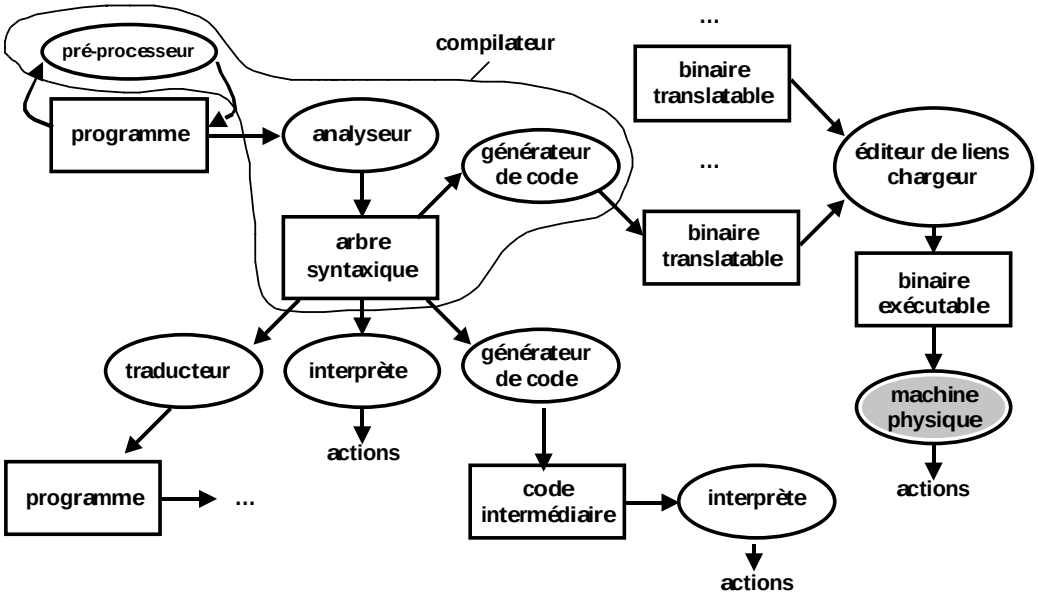
- **Principe de l'éditeur de lien à 2 passes (simplifié)**
  - Phase initiale : l'éditeur de liens réunit bout à bout l'ensemble des segments binaires translatables pour faire un segment objet unique. Toute information a alors une adresse (relative) dans ce segment objet (le binaire translatable global)
  - Passe 1 : collecte des références. À partir des tables des définitions externes, l'éditeur de liens construit une table globale des symboles. Une entrée de cette table contient le nom d'une procédure fournie et l'adresse (relative) correspondante dans le segment objet.
  - Passe 2 : résolution des références externes. À partir des tables des références externes, l'éditeur de liens retrouve toutes les instructions faisant référence à une procédure externe, et met dans la partie adresse de chacune de ces instructions l'adresse relative trouvée dans l'entrée de la table globale des symboles correspondant à la procédure externe utilisée.
  - À la fin de la passe 2, toutes les références externes doivent avoir été résolues. Si ce n'est pas le cas, l'éditeur de liens signale les références non résolues, et le programme global ne peut pas être exécuté.
  - Le programme global doit encore passer par le chargeur pour obtenir un binaire exécutable. Souvent, la fonction du chargeur est incluse dans l'éditeur de liens, qui produit alors directement le binaire exécutable.



## Exemples

- **La commande *gcc* permet d'appeler à la fois le compilateur, l'éditeur de liens et le chargeur**
  - gcc prog.c ; produit (par défaut) le binaire exécutable dans le fichier a.out
  - gcc -o prog prog.c ; produit le binaire exécutable dans le fichier prog
  - gcc -c prog.c ; produit (par défaut) le binaire translatable dans le fichier prog.o
  - Supposons qu'un programme soit composé de deux parties prog1.c et prog2.c. Alors :
  - gcc prog1.c prog2.c produit le binaire exécutable du programme complet dans a.out
  - gcc - c prog1.c prog2.c produit les deux binaires translatables prog1.o et prog2.o
  - gcc -o prog prog.o prog1.o produit le binaire exécutable du programme complet dans prog, à partir des binaires translatables prog1.o et prog2.o
- **Utilisation de bibliothèques**
  - On peut rechercher les références externes dans des collections de programmes appelées bibliothèques, ou archives, fournies par le système sous forme de binaires translatables pour les utilitaires courants
  - On spécifie alors -l<nom>, et l'éditeur de liens recherche la bibliothèque lib<nom>.a
  - On peut spécifier des règles de recherche (où trouver les bibliothèques, etc.)

## Schéma général d'exécution de programmes






## Résumé de la séance 5

- **Exécution de programmes**
  - Programmes compilés, programmes interprétés
  - Avantages respectifs des deux schémas, schémas mixtes
- **Principe de fonctionnement d'un interprète**
- **Cycle de vie d'un programme compilé**
  - Programmes translatables, principe d'un chargeur
  - Composition de programmes, principes d'un éditeur de liens
- **Première idée de la notion de liaison**
  - mise en correspondance d'un nom et d'une adresse