



UNIVERSITÉ
CÔTE D'AZUR

Production et Exécution des Programmes

Présentation: **Stéphane Lavirotte**

Auteurs: ... et al*



(*) Cours réalisé grâce aux documents de :
Sacha Krakowiak, Stéphane Lavirotte, Jean-Paul Rigault

Mail: Stephane.Lavirotte@univ-cotedazur.fr

Web: <http://stephane.lavirotte.com/>

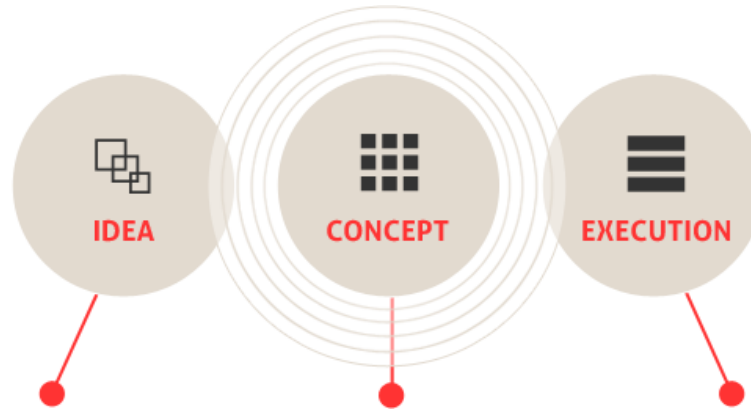
Université Côte d'Azur



Positionnement de ce Cours

- ✓ **Cours de liaison avec les cours du 1^{er} semestre**
 - **Programmation C et Système**
 - Introduction au langage C (types, fonctions, préprocesseur, pointeurs, programmation modulaire...)
 - **Architecture et Réseaux (Principe d'exécution des programmes)**
 - Représentation des nombres, opérateur de base, logique et circuit combinatoire, unité arithmétique et logique, assembleur
 - **Programmation Orientée Objet**
 - Classes, objets, héritage, design patterns, machine virtuelle

- ✓ **Pour aller vers un point essentiel**
 - La couche entre vos programmes et la machine
 - « Système d'Exploitation ». Pourquoi ? Comment ?



Concepts de base de l'exécution des programmes

Au niveau du processeur



Modèle de von Neumann

- ✓ **A la base, machine de Turing (1936)**
 - **Modèle *abstrait* du fonctionnement des appareils mécaniques de calcul**
 - Un ruban infini
 - Une tête de lecture/écriture
 - Un registre d'état
 - Une table d'actions
- ✓ **Puis, Modèle de von Neumann (étendu) (1945)**
 - **Mémoire pour programmes et données**
 - Programme non inscriptible (non automodifiable)
 - **Unité centrale avec registres de travail**
 - **Mot d'état du processeur**
 - Modes utilisateur/système, instructions privilégiées
 - **Exécution séquentielle**
 - Compteur ordinal
 - Ruptures de séquence (branchement)

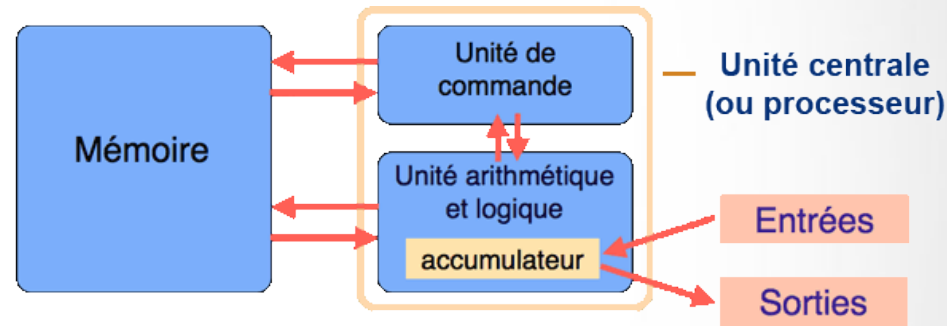


Une Architecture Novatrice

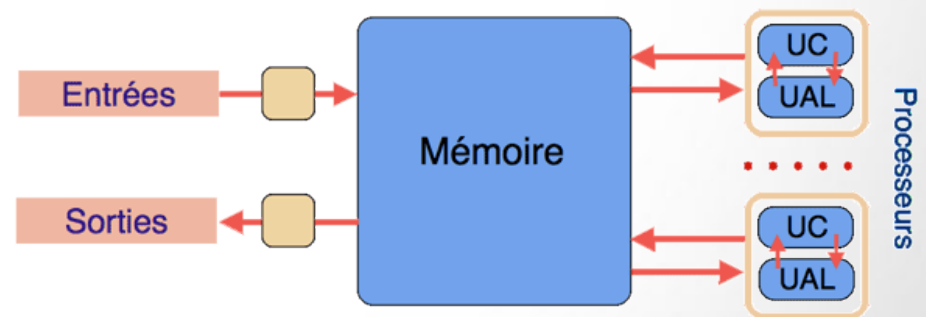
✓ “First Draft of a Report on EDVAC”

– Document de 101 pages décrivant:

- un schéma d'architecture de calculateur: 3 éléments
 - unité arithmétique
 - unité de commande
 - mémoire contenant programme et données
- des principes de réalisation pour ces éléments
 - les opérations arithmétiques



Modèle originel



Modèle actuel

Modèle de von Neumann

Concepts de base (1/2)

- ✓ **Atomicité des instructions élémentaires (?)**
- ✓ **Procédure et activité**
 - Procédure : élément structurant d'un programme
 - Activité : exécution d'une procédure
- ✓ **Programme et processus**
 - Programme : ensemble de procédures, le résultat statique d'une compilation/édition de liens (binaire exécutable)
 - Ensemble de procédures
 - Processus : l'activité dynamique résultant de l'exécution d'un programme
 - Enchaînement (séquentiel ou concurrent) d'activités

Modèle de von Neumann

Concepts de base (2/2)

✓ Contexte

- Information caractérisant l'état courant d'un processus
 - Sauvegardé lors de la suspension du processus et rechargé lors de sa reprise
- Contexte matériel
 - Mot d'état, compteur ordinal, registres de l'unité centrale
- Contexte logiciel (ou système)
 - Segments de texte (instructions), de données, de pile...
 - Attributs : identification utilisateur, droits d'accès, priorité...
 - Ressources utilisées : fichiers ouverts...

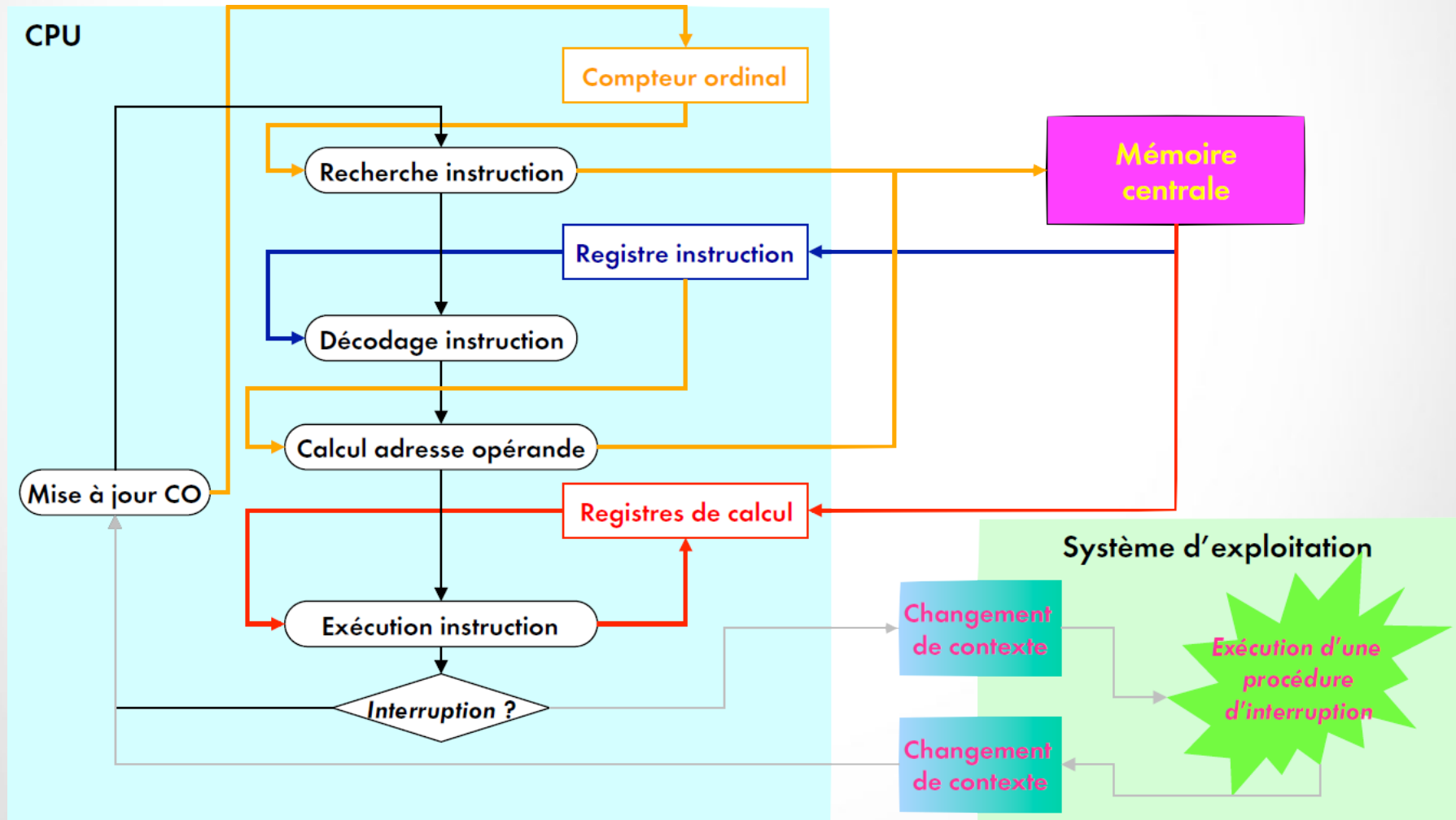
Événements, Interruptions et Déroutements (1/2)

- ✓ **Événements : interruption et déroutement**
 - **Interruption** : événement asynchrone (inopiné)
 - Fin d'entrée-sortie
 - Condition « anormale » d'un processus extérieur (temps réel)
 - **Déroutement (ou exception)** : événement provoqué par l'exécution même du programme, correspondant à une situation « exceptionnelle »
 - Exception arithmétique, violation de protection mémoire...
 - Passage du mode utilisateur au mode système (TRAP)
 - **Traitement uniforme par la plupart des architectures**

Événements, Interruptions et Déroutements (2/2)

- ✓ Événements : interruption et déroutement (suite)
 - Hiérarchie des interruptions (priorité)
 - Masquage/démasquage
 - Armement/désarmement
 - Procédure (de traitement) d'interruption
 - Déclenchement (ordonnancement) par le matériel
 - Notion de réentrance
 - Réexécution d'une procédure actuellement suspendue (par suite d'une IT)
 - Conditions nécessaires : invariance du code, séparation des données
 - La récursivité implique la réentrance

Cycle élémentaire d'Exécution d'une Instruction





Production et Exécution de Programmes

Langage de Programmation (de Haut Niveau)

- ✓ **Modèle de calcul plus proche des applications**
 - Types de données et opérations
 - Mécanismes d'abstraction

- ✓ **Vérification et renforcement de la sécurité de programmation**
 - Vérifications statiques
 - Vérifications dynamiques
 - Utilisation des données

- ✓ **Nécessité d'un mécanisme de traduction ou d'interprétation en langage-machine**

Langage de Programmation

Modèle de calcul

- ✓ **Types de données et opérations**
 - Structuration des données de base de la machine sous-jacente
 - Nouveaux types de données et opérations
 - définis par le langage : `int`, `long`, `long long`, `complex`...
 - définis par l'utilisateur : `struct`, `union`, `class`

- ✓ **Mécanismes d'abstraction**
 - Structuration du flot de contrôle, des données
 - Sous-programmes, fonctions...
 - Modules, classes...
 - Séparation spécification/implémentation

Langage de Programmation

Sécurité de Programmation

✓ Vérifications statiques

- Avant toute exécution
- Lors du processus de traduction
- Exemples :
 - Analyse de type (en C/C++, Java)
 - Contrôle d'accès (en C++, Java)

✓ Vérifications dynamiques

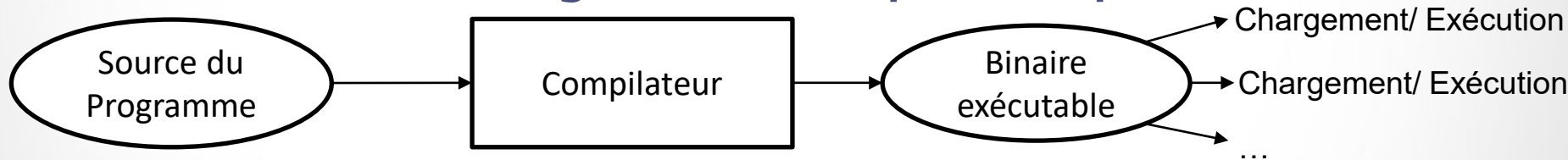
- Lors de l'exécution
- Exemple :
 - Dépassement de borne d'indice
 - Paramètres incorrects



Compilation et Interprétation

✓ Compilation

- Le source du programme est traduit, une fois pour toutes (?) en langage-machine
- Le résultat est un fichier binaire exécutable
- Ce fichier est chargé en mémoire pour chaque exécution



✓ Interprétation

- Le source du programme est traduit « à la volée », instruction
- par instruction, lors de chaque exécution



Compilation et Interprétation

Avantages et Inconvénients

✓ Compilation

- Possibilité de vérifications et d'optimisations statiques et globales
- Renforcement de la sécurité de programmation
- Programme figé
- Cycle de développement lourd

✓ Interprétation

- Le programme peut évoluer (et même se faire évoluer lui-même)
- Grande puissance d'expression
- Mise au point interactive
- Vérifications dynamiques seulement
- Détection d'erreurs à l'exécution et optimisation délicate

Compilation et Interprétation

Combinaison des 2 Approches

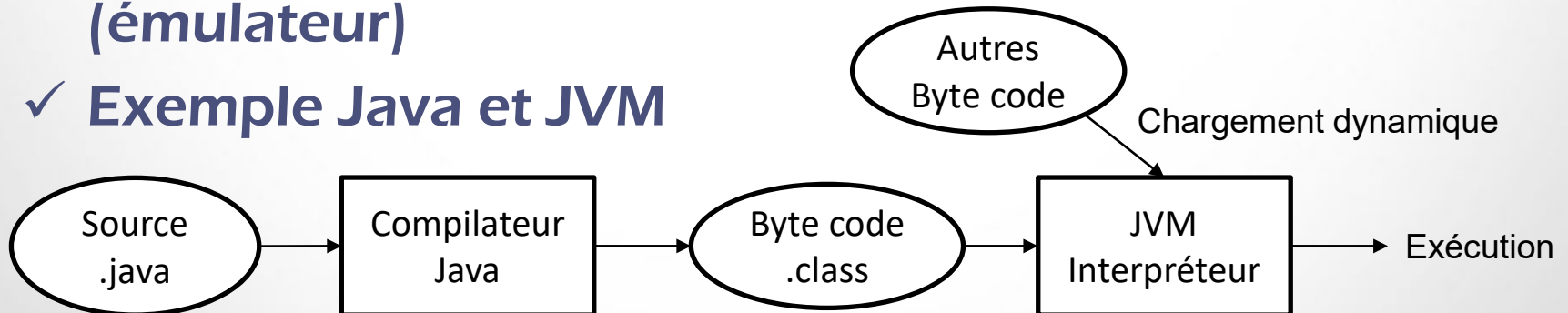
- ✓ **Langages de haut niveau et matériel**
 - Des langages comme C/C++, Ada... sont traduits en langage machine (compilation)
 - Le matériel (processeur) interprète ce langage-machine
 - Les fichiers exécutables sont dépendants du matériel utilisé

- ✓ **Langages de haut niveau et machine virtuelle**
 - Un langage comme Java est compilé en un langage-machine virtuel (byte-code)
 - Ce langage est lui-même interprété par un programme appelé machine virtuelle (voir plus loin)
 - Les fichiers byte-code sont indépendants du matériel

Machine Virtuelle

Principe

- ✓ **Inventer une architecture de processeur adaptée à l'interprétation du langage de haut niveau**
 - Registres
 - Jeux d'instruction
 - Structure de données manipulées ...
- ✓ **Compiler le langage dans le jeu d'instructions de la machine**
- ✓ **Fournir une réalisation logicielle de cette machine (émulateur)**
- ✓ **Exemple Java et JVM**

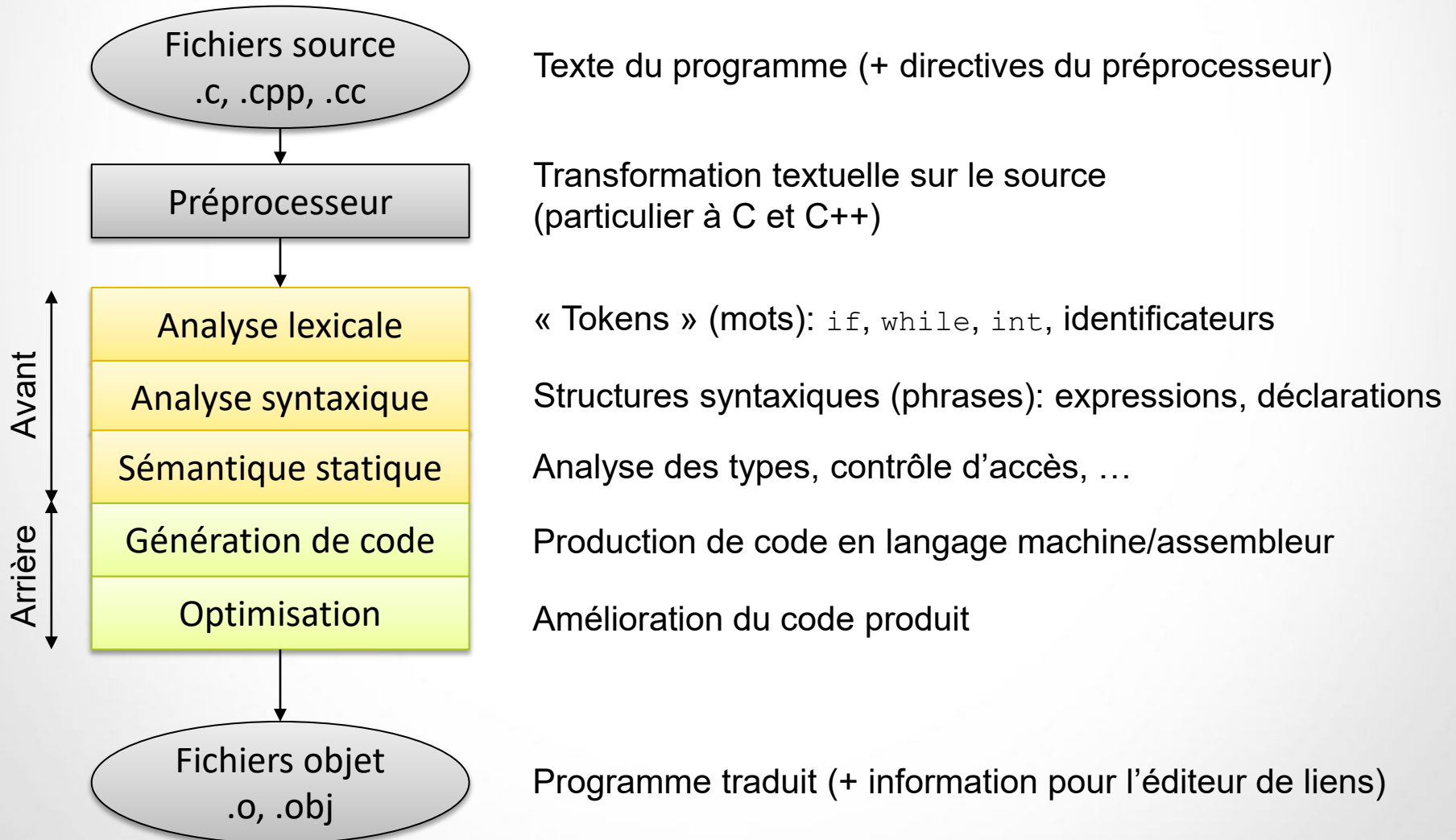


Machine Virtuelle

Avantages Inconvénients

- ✓ **Les programmes sont portables**
 - Seule la machine virtuelle doit être réécrite ...
 - et encore, elle peut elle-même être largement portable
- ✓ **Avantages de l'interprétation et de la compilation réunis**
 - Mécanismes dynamiques et vérifications statiques
- ✓ **Performance**
 - Interprétation logicielle
 - Double traduction
- ✓ **Maintenance, évolution**
 - Un changement du langage implique la mise à jour du compilateur et de la machine virtuelle

Processus de Compilation (cas de C / C++)





Compilation séparée ?

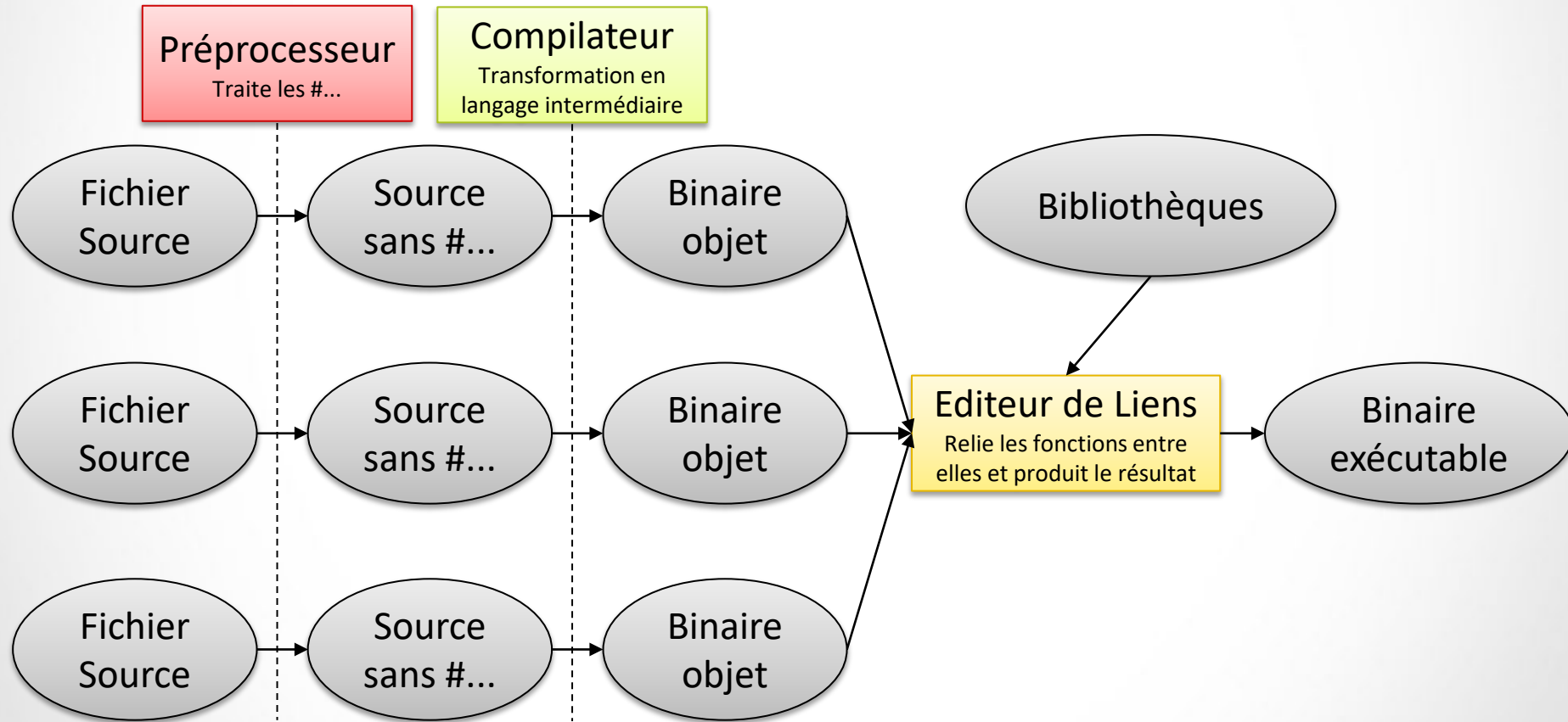
✓ Fichier source unique ?

- Difficile à maintenir
- Inefficace
- Besoin de bibliothèques précompilées

✓ Compilation séparée

- Le source du programme est découpé en unités de compilation
 - En C/C++, unité de compilation = un fichier `.c` (ou `.cc`, `.cpp`) et tous les fichiers `.h` (ou `.hpp...`) qu'il inclut
 - En Java, unité de compilation = un fichier `.java`
- Chaque unité est compilée séparément
- Un éditeur de lien colle les morceaux et produit l'exécutable
 - (sous Unix/Linux, l'éditeur de liens est `ld`)

Etapes de la Compilation de Code Source





Bibliothèques

Edition de liens statique et dynamique



Bibliothèques (« *Library* »)

- ✓ Fichier unique contenant un ensemble de fichiers-objets précompilés
- ✓ Deux types de bibliothèques existent:

- Bibliothèques statiques

- Copiées dans les programmes qui les utilisent

Unix / Linux	Windows	Mac OS
.a	.lib	.a

- Bibliothèques partagées

- Associées aux programmes au moment où ils sont exécutés

Unix / Linux	Windows	Mac OS
.so	.dll	.dylib

- ✓ Commandes communes Unix/Linux

- `ld` : lie des fichiers `.o` entre eux
- `nm`, `readelf` : lisent le contenu d'une bibliothèque ou d'un programme



Exécutable et Bibliothèques

- ✓ **Exécutable utilisant une ou des bibliothèques**
 - Le cas par défaut quand on compile avec `gcc`
`gcc options -o myprog prog1.o prog2.o`
 - Un tel programme utilise au moins la bibliothèque C partagée

- ✓ **Exécutable n'utilisant aucune bibliothèque**
 - Pour qu'un exécutable n'utilise aucune bibliothèque partagée (même la bibliothèque C)
`gcc options -static -o myprog prog1.o prog2.o`
 - Le code « nécessaire » de la bibliothèque est alors ajouté à l'exécutable



Bibliothèques Statiques

- ✓ **Fichier unique contenant un ensemble de fichiers-objets précompilés**
- ✓ **L'éditeur de liens traite les bibliothèques statiques de manière spéciale**
 - Seul le code nécessaire au programme (les définitions d'objets utilisés par le programme) iront dans le binaire exécutable
- ✓ **Commandes Unix/Linux**
 - `ar` : crée une bibliothèque statique par regroupement de `.o`
 - `ranlib` : dote la bibliothèque d'un index (aide pour `ld`)



Edition de Liens Statique

✓ Création de la bibliothèque

```
gcc options -c libobj1.c
```

```
gcc options -c libobj2.c
```

```
...
```

```
ar -r libobj.a libobj1.o libobj2.o ...
```

```
ranlib libobj.a
```

✓ Utilisation de la bibliothèque

```
gcc options -o myprog prog1.o prog2.o libobj.a
```

ou

```
gcc options -o myprog prog1.o prog2.o -L dir -lobj
```



Bibliothèques Partagées

- ✓ **Eviter la duplication du code des bibliothèques courantes (e.g., `libc`, `libstdc++`)**
 - dans le fichier binaire exécutable
 - en mémoire
- ✓ **Permettre de modifier l'implémentation d'une bibliothèque (mais pas son interface !)**
 - sans refaire l'édition de liens des applications
 - sans « rebooter »
- ✓ **Permettre le chargement à la demande de bibliothèques**
- ✓ **Nommage et versions**
 - **Sous Windows**
 - Nom avec extension `.dll`
 - Version : cauchemar ! (amélioration avec `.Net`)
 - **Sous Unix/Linux (principe !)**
 - `libstdc++.so.3.5`

Nom de base. **extension so (shared object)**, **numéro majeur (version interface)**, **numéro mineur (implem interface)**



Edition de Liens Dynamique

✓ Création de la bibliothèque

```
gcc options -fPIC -c libobj1.c
```

```
gcc options -fPIC -c libobj2.c
```

...

```
gcc -shared -Wl,-soname,libobj.so.3 -o libobj.so.3.2  
libobj1.o libobj2.o ...
```

C'est la lettre L minuscule, il n'y a pas d'espace entre les virgules

Nom de la bibliothèque

Fichier contenant le code de la bibliothèque

✓ Utilisation de la bibliothèque

```
gcc options -o myprog prog1.o prog2.o libobj.so  
ou
```

```
gcc options -o myprog prog1.o prog2.o -L dir -lobj  
(on suppose qu'il n'existe pas de libobj.a, sinon utiliser option -Bdynamic)
```

✓ *La bibliothèque devra être accessible à l'exécution (voir plus loin)*

Chargement Dynamique de Bibliothèque

- ✓ Possibilité de chargement/déchargement à la demande d'une bibliothèque sans avoir effectué au préalable une édition de liens avec elle
- ✓ Sous Unix/Linux, utilisation d'une bibliothèque de chargement dynamique, `libdl.so` (en-tête `<dlfcn.h>`)
 - `dlopen()` ouverture d'une bibliothèque partagée
 - `dlclose()` fermeture de la bibliothèque
 - `dlsym()` recherche d'un symbole dans la bibliothèque
 - `dLError()` traitement des erreurs
- ✓ Mécanisme
 - Pour le chargement dynamique de classes dans Java
 - Pour gérer la notion de plugin ou de addons

Commandes de Manipulation des Bibliothèques partagées

- ✓ **Lecture du contenu des bibliothèques partagées**
 - `readelf`

- ✓ **Bibliothèques partagées utilisées par un exécutable ou une autre bibliothèque**
 - `ldd`

- ✓ **Recherche des bibliothèques partagées à l'exécution**
 - **Chargeur dynamique** : `ld.so`
 - **Variable d'environnement** `LD_LIBRARY_PATH`
 - **Commande** `ldconfig` **et fichier de configuration** `ld.so.conf` (réservé au super-utilisateur)

Une Vision Synthétique sur les Bibliothèques

