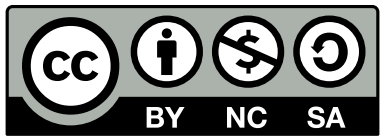
Chapitre 1: Introduction

INF7641 Compilation

Jean Privat

Université du Québec à Montréal

INF7641 Compilation
v251

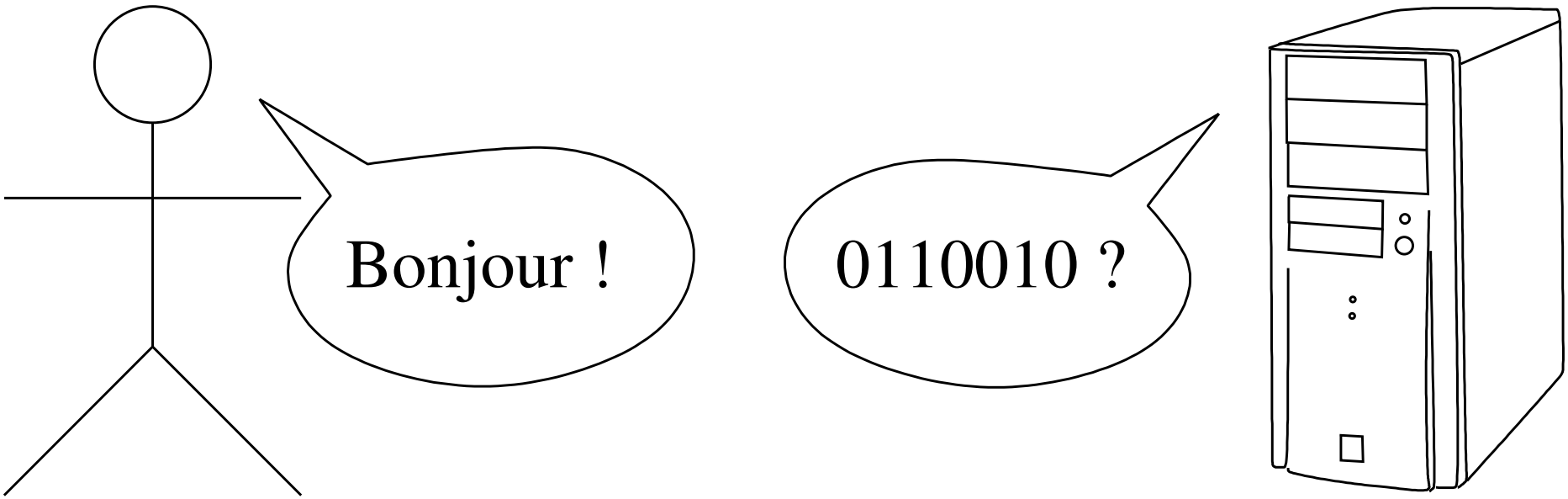


Plan

1. Problème de communication
2. Compilation
3. Exemple: Compilation du langage C
4. Interpréteurs et machines virtuelles
5. Structure de compilateurs et interpréteurs

Problème de communication

Programmation = Problème de communication



Langages de programmation

Permet l'écriture de programmes par un humain

- Distinct du code machine, inadapté pour les humains

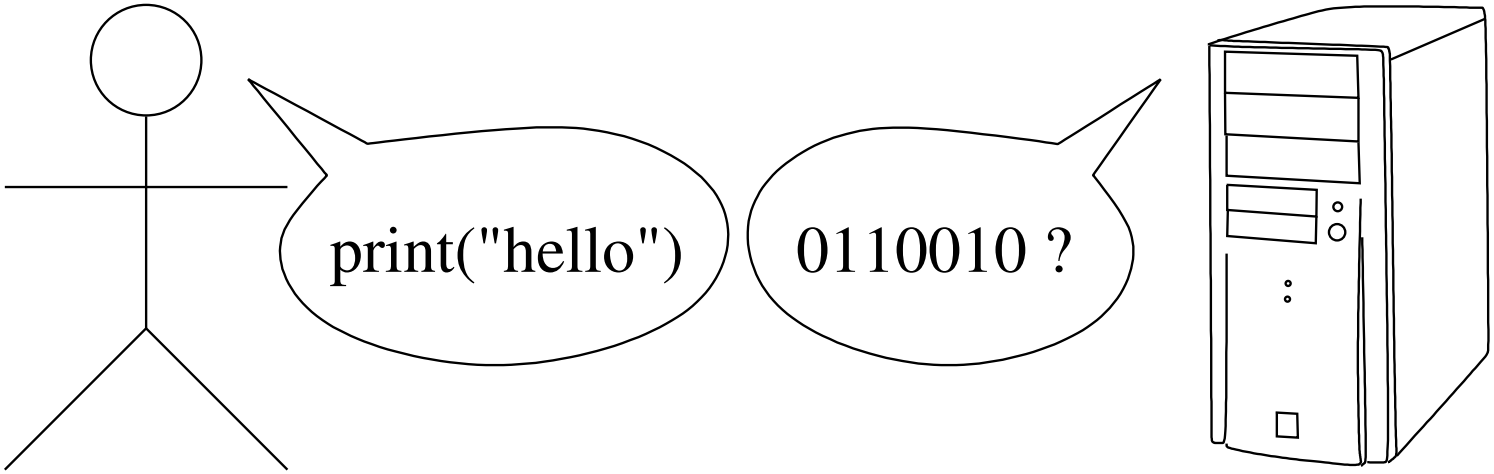
Quelques langages de programmation

- Assembleur, C, Python, Haskell, Prolog, Smalltalk, etc.

Diversité et évolution

- Nombreux langages existants avec leurs caractéristiques, forces et faiblesses
- Évolution continue de ces langages

Exécution d'un programme



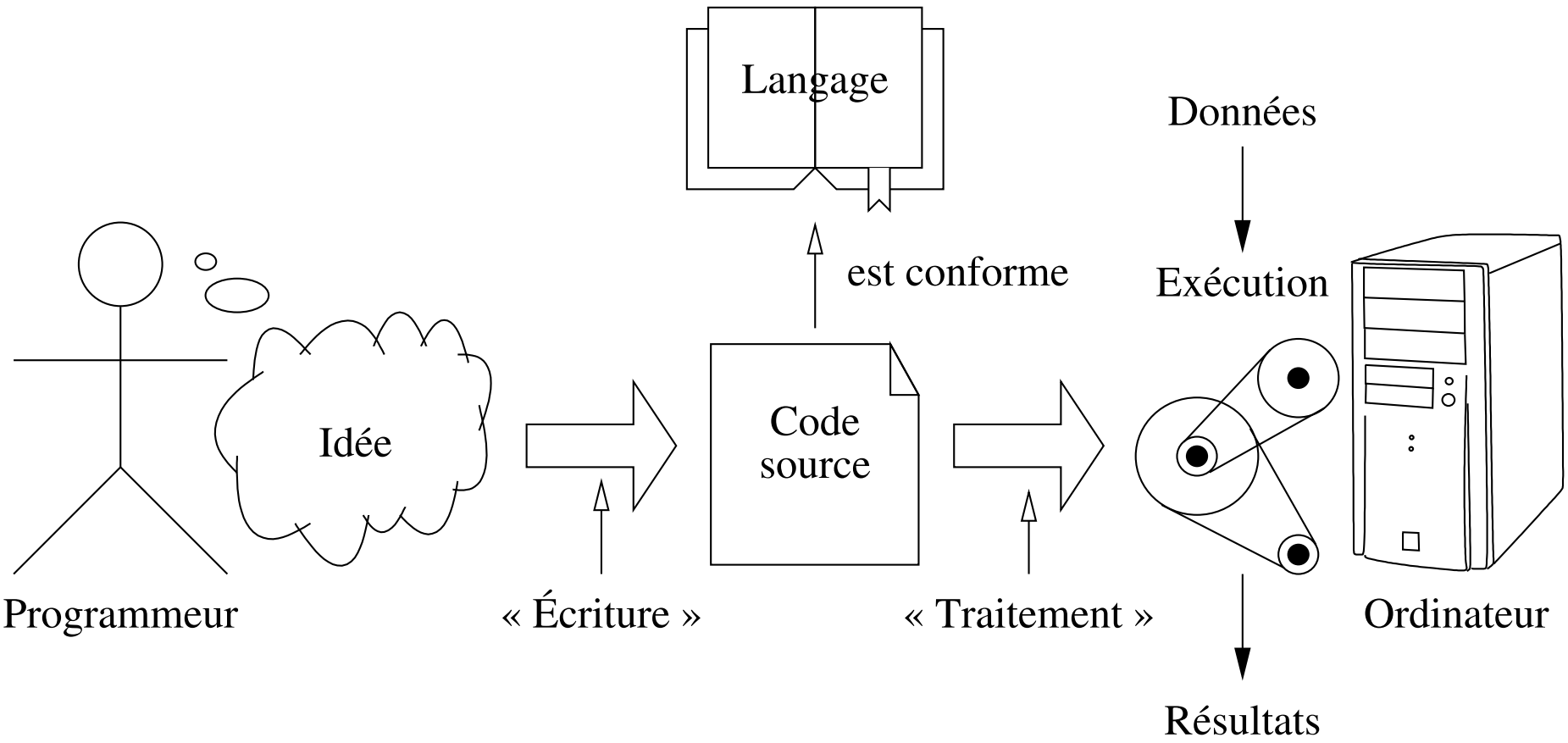
Un microprocesseur ne peut pas en soit exécuter directement le code source d'un programme

- Il ne comprend que son propre langage machine

Le code source doit être traité pour pouvoir être exécuté par une machine

- Transformé (compilation)
- Évalué (interprétation)

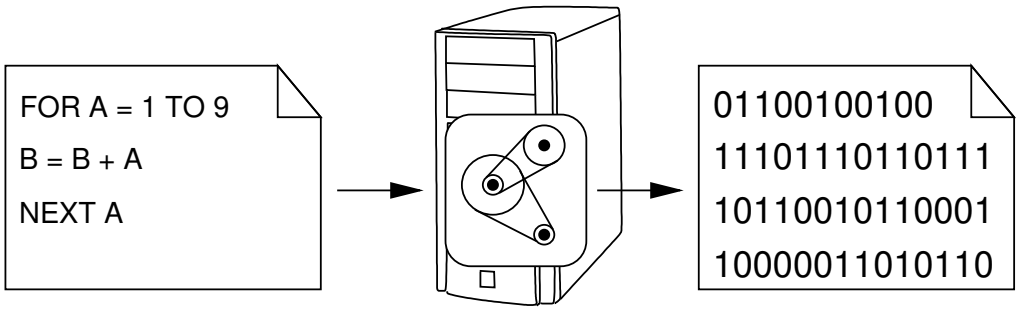
Exécution d'un programme

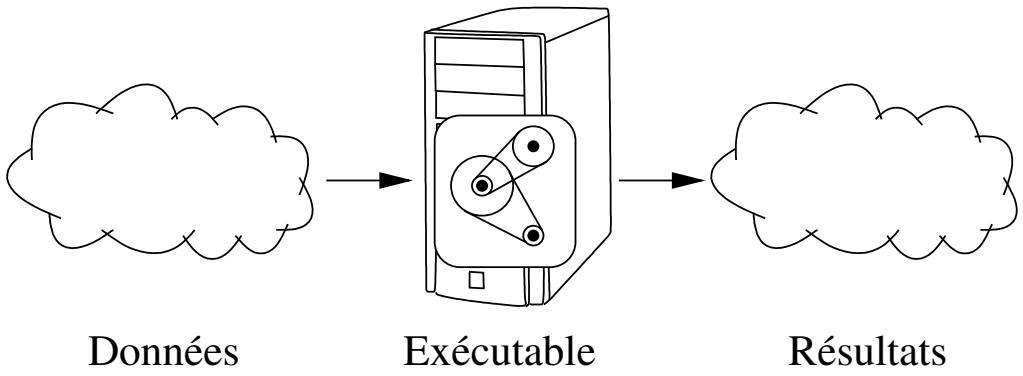


Compilation

Compilateur

- Programme qui transforme du code source en du code exécutable par une machine
- La compilation a lieu avant l'exécution





Un compilateur et un programme comme les autres

Avec « tout » ce qui vient avec :

- Interface utilisateur : CLI, messages d'erreurs, options, etc.
- Analyse, conception et maintenance
- Structure de données et algorithmes
- Performances et ressources utilisées
- Bugs ?

Problème de l'amorçage (*bootstrap*)

- Question de l'œuf et de la poule
- Problème fréquent en informatique

Questions pratiques

- Compilateurs autogènes : écrit dans son propre langage
- Compilateurs croisés : cible une autre architecture que l'architecture courante

Types de compilateurs

Compilateurs traditionnels

D'un langage de haut niveau (utilisé par un humain)
à un langage de bas niveau (compris par une machine)

- `gcc` de C vers un langage machine (x86, riscv, etc.)
- `javac` de Java vers du bytecode Java (JVM)

Autre ?

- Préprocesseur, comme cpp (`gcc -E`)
  Prétraite du code source augmenté de directives
- Générateurs de code, comme `sablecc`
  Génère du code source à partir d'un DSL dédié
- Compilateurs source-à-source (*transpiler*, *transcompilateur*)
  Convertit un programme d'un langage à un autre langage de « niveau » équivalent

La catégorisation est parfois floue...

Types de compilateurs

Décompilateurs

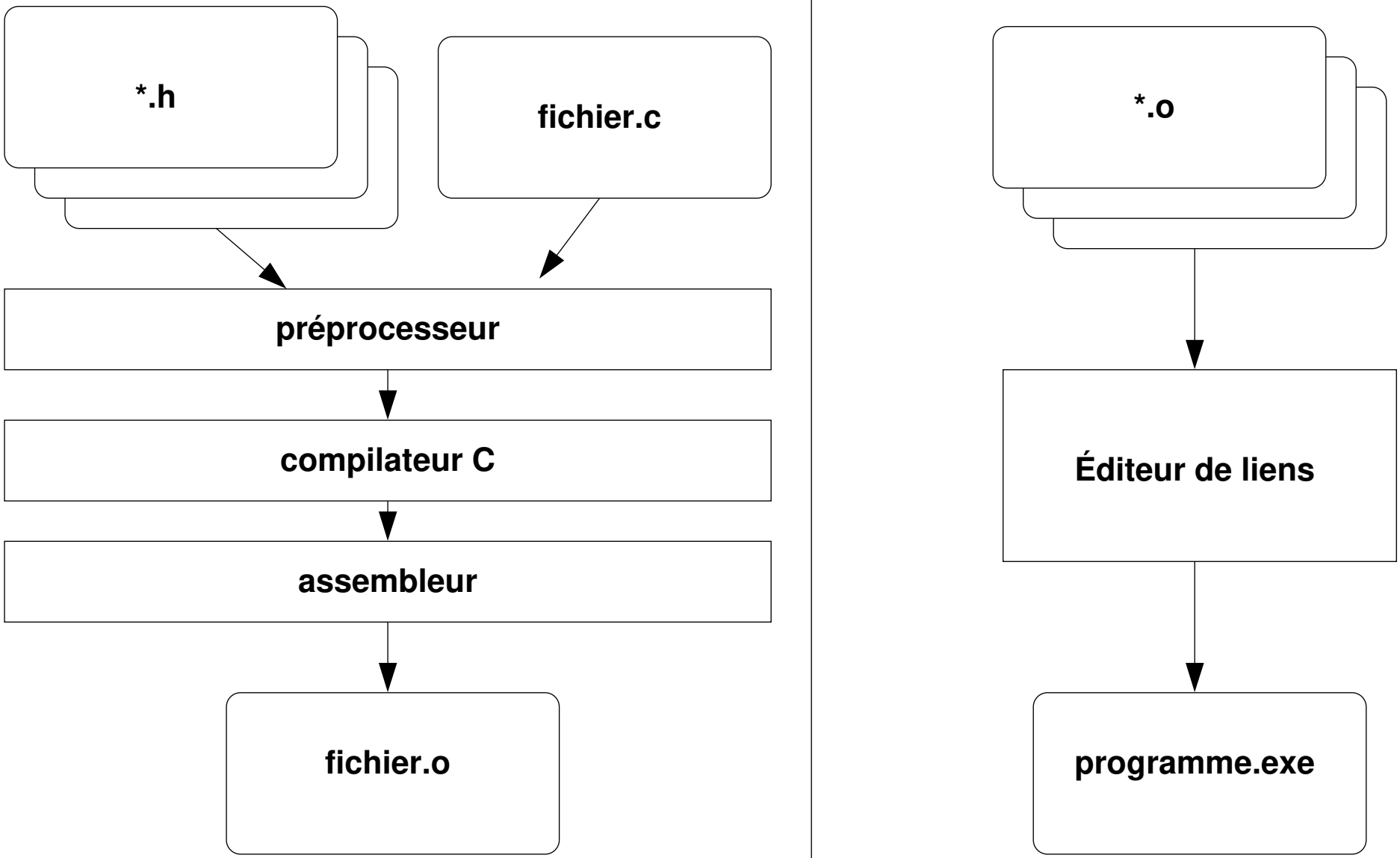
- Travail inverse
- Outil de rétro-ingénierie
- Voir INF600C
- Difficile
- Hors du cadre du cours

Domaine de la « compilation » au sens large ?

- Traitements de programmes
  qu'ils soient données et/ou résultats
- Analyse, transformation et exécution

Exemple: Compilation du langage C

Schéma général d'un compilateur C



GCC (GNU Compiler Collection)

- Depuis 1987 par Richard Stallman, et bien d'autres depuis
- C, C++, fortran, go, etc.
- Nombreuses architectures cibles (≈ 50)
- Utilise `bintools` pour les détails bas niveau
  - Utile comme *frontend* pour `as` et `ld` (surtout `ld`)
  - `gcc hello.s -static -nostdlib -o hello`
- Écrit en C++ (depuis 2015), licence GPL3

GNU binutils

- Outils binaires bas niveau de GNU
- Écrit en C, licence GPL3

GNU assembler (as)

- Famille d'assembleurs, génère des fichiers objets Unix ELF (*Executable and Linkable Format*)
- Documentation, 448 pages
- `as hello.s -o hello.o`

GNU linker (ld)

- Combine des fichier objets (`.o`) en un exécutable (ou autre)
- `ld hello.o -o hello`

Options de compilation GCC

- Plus de 3000 options: langage source, langage cible, diagnostiques, optimisations, assemblage, édition de liens, génération de code, machine cible, etc.
- Liste des options de GCC

Options pertinentes

- Langage source `-X` (ou extension du fichier)
  - et versions du standard `-std`
- Optimisations : `-O`
  - `-O0`, `-O1`, `-O2`, `-O3`, et `-f...`
- « Profondeur » de compilation `-E`, `-S`, `-c`
- Architecture cible `-march` (ISA)
  - et micro-architecture cible: `-mtune`

Clang/LLVM

- Sponsorisé par Apple
- Écrit en C++, licence Apache2

Clang

- Depuis 2007 par Chris Lattner, et bien d'autres depuis
- C, C++, etc.
- Sous-projet de LLVM
- Cherche à être compatible avec gcc

LLVM

- Depuis 2003
- N'est plus un acronyme de *Low Level Virtual Machine*
- Middle-end et back-end de compilateurs
- Langage intermédiaire (LLVM IR) stable

Compiler explorer

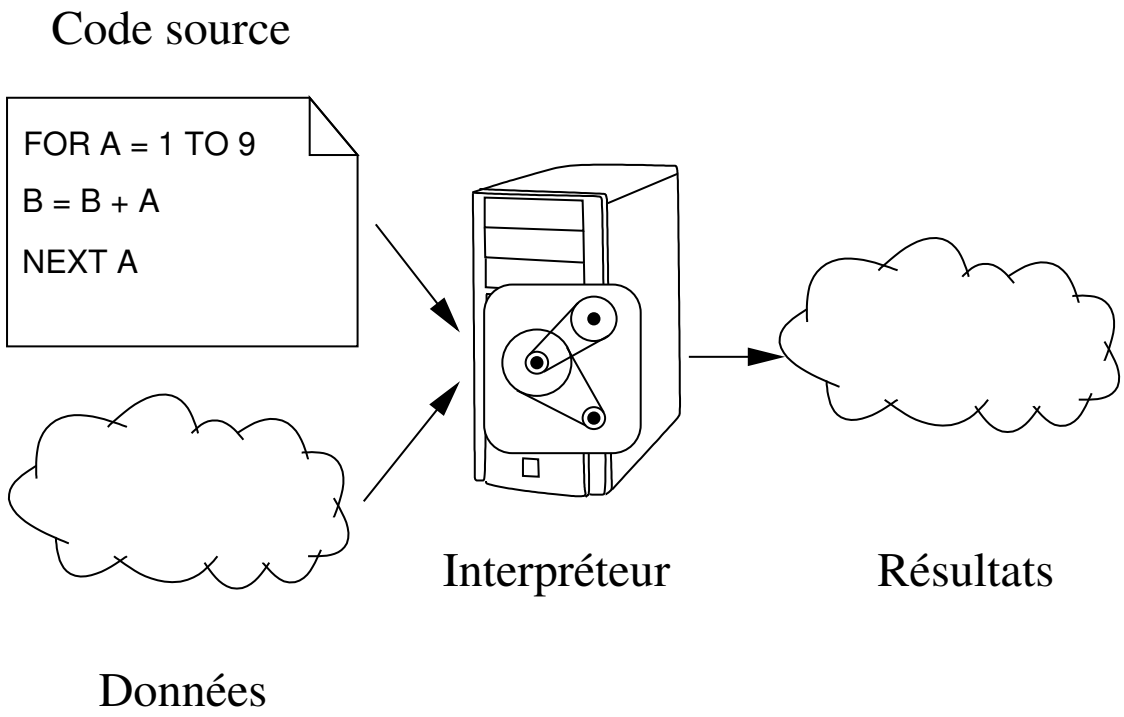
Permet l'expérimentation interactive

- Plusieurs compilateurs (et versions)
- Plusieurs architectures
- Plusieurs options
- $\rightarrow$ https://godbolt.org/

Interpréteurs et machines virtuelles

Interpréteur

- Programme qui prends un programme en entrée et l'exécute directement
- Souvent interactifs et réflexifs (parce que c'est facile)



Note: l'interpréteur est un programme comme les autres

Interpréteurs

Interpréteurs traditionnels

- `ruby` pour Ruby
- `bash` pour le shell GNU+POSIX

Machines virtuelles

- `java` pour le bytecode Java
- Émulateurs comme `qemu` ou `snes9x` pour les langages machines
- Question: est-ce que Wine est un émulateur ?

Note linguistique

- Simulation = pour l'analyse et l'étude (on modélise)
- Émulation = pour l'utilisation et la substitution (on réplique)

Interpréteurs au sens large

- Description de comportement

Compilateurs vs. interpréteurs

- Vitesse d'exécution ?
- Souplesse ?
- Facilité de développement ?

Machine virtuelle + JIT

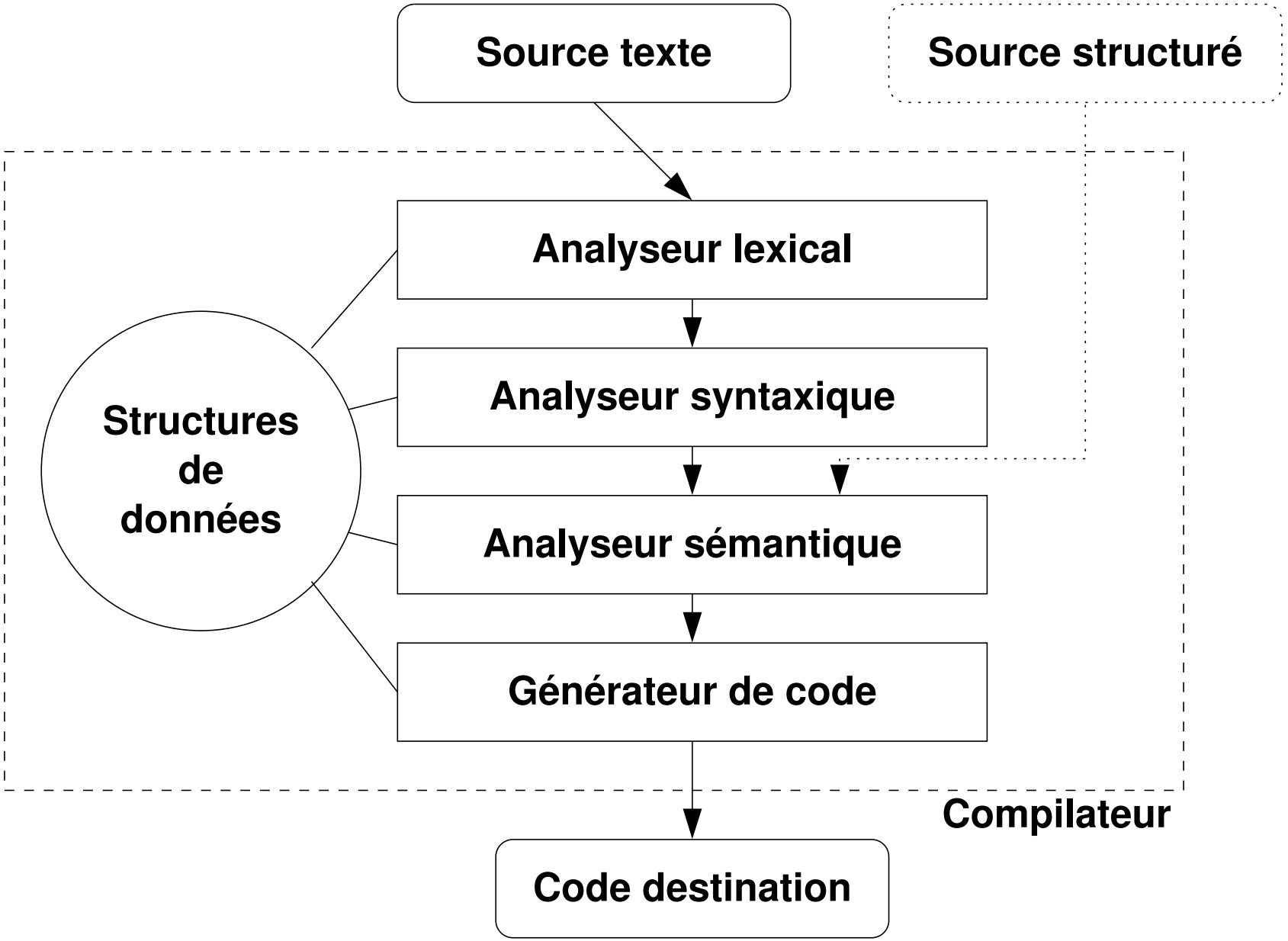
Compilateur juste-à-temps, (*just-in-time*, JIT)

- Aussi appelé *dynamic translation*
- Interpréteur spécialisé qui **génère du code machine à la volée**
- Exemples: machines virtuelles Java, C#, JavaScript ; QEMU...
- Très sophistiqué, détails en INF7741

Note: la machine virtuelle est un programme comme les autres

Structure de compilateurs et interpréteurs

Structure d'un compilateur



Analyse lexicale

Rôle

- Extraire les mots (lexèmes, jetons) du tas de caractères qu'est le code source

Exemple: « `for i = 1 to 10 do print i` »

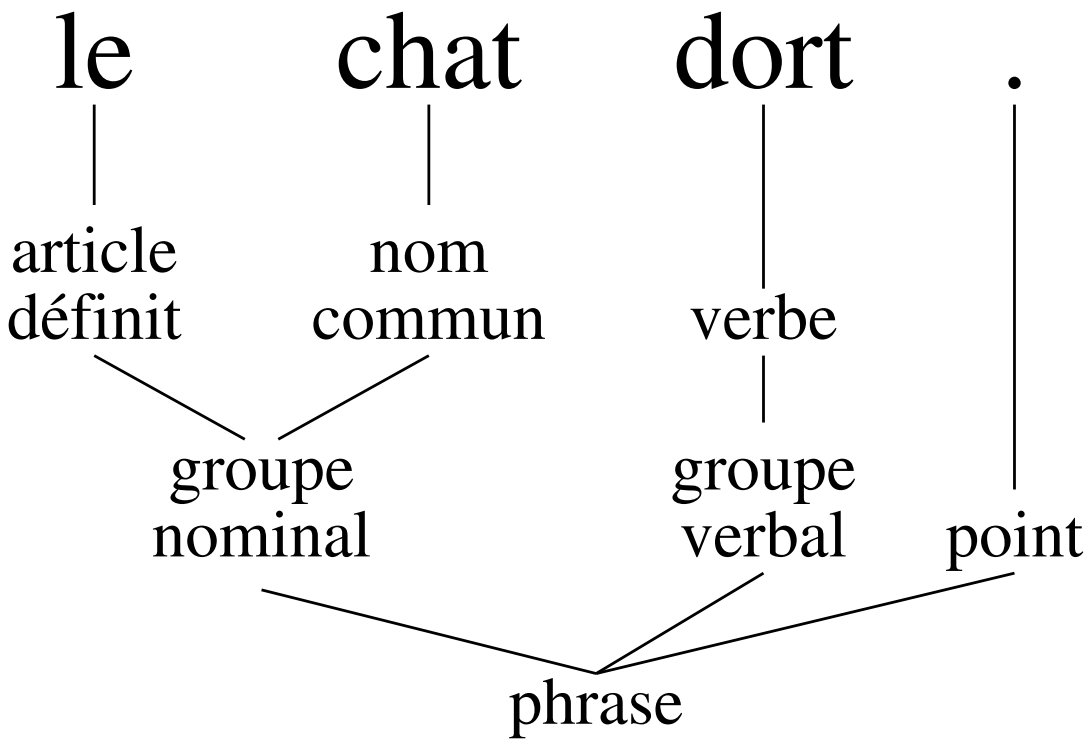
- Mot clé « `for` »
- Identifiant « `i` »
- Symbole égal « = »
- Entier « `1` »
- Mot clé « `to` »
- etc.

Analyse syntaxique

Rôle

- Reconstruire la structure syntaxique à partir des lexèmes

Exemple: « le chat dort. »



Analyse sémantique

Objectifs multiples

- Extraire le « sens » du code
- Vérifier sa cohérence
- Calculer des informations additionnelles

Exemple

- « Le crayon mange une pomme.»
- Syntaxe: OK
- Sens: pas OK

Génération de code

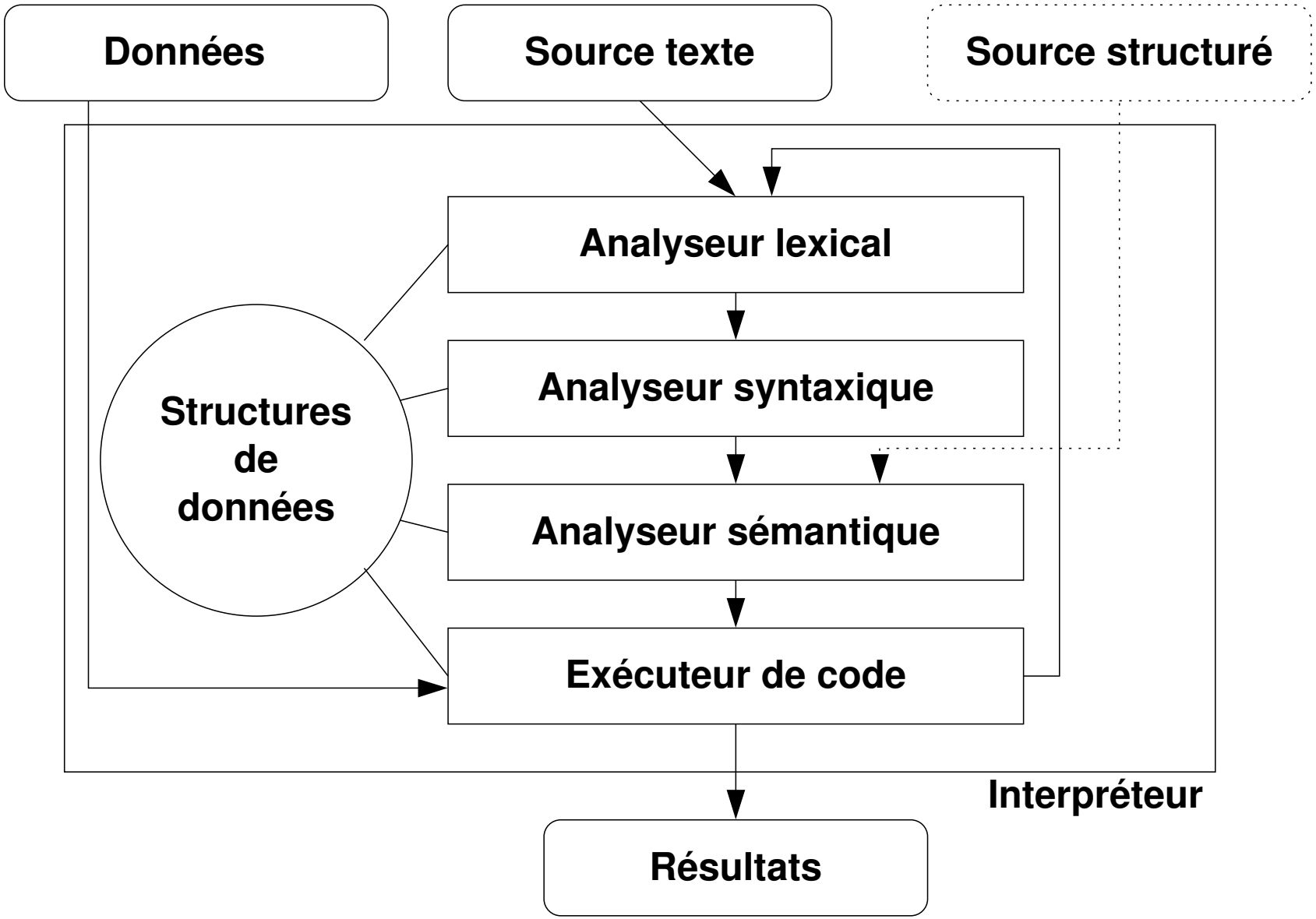
Générer du code en langage cible

- Utilisation des informations calculées aux étapes précédentes

En plusieurs étapes (*langages intermédiaires*)

- Si le langage source est très distant du langage cible
- Si le compilateur est optimisant

Structure d'un interpréteur



Exécution du code

Exécuter du code au fur et à mesure

- Utilisation des informations calculées aux étapes précédentes
- Maintient de structures de données additionnelles pour représenter l'état d'exécution du programme

Interpréteur mixte

- Interprète une représentation intermédiaire
- Invoque un compilateur juste-à-temps