

Langages et exécution d'un programme

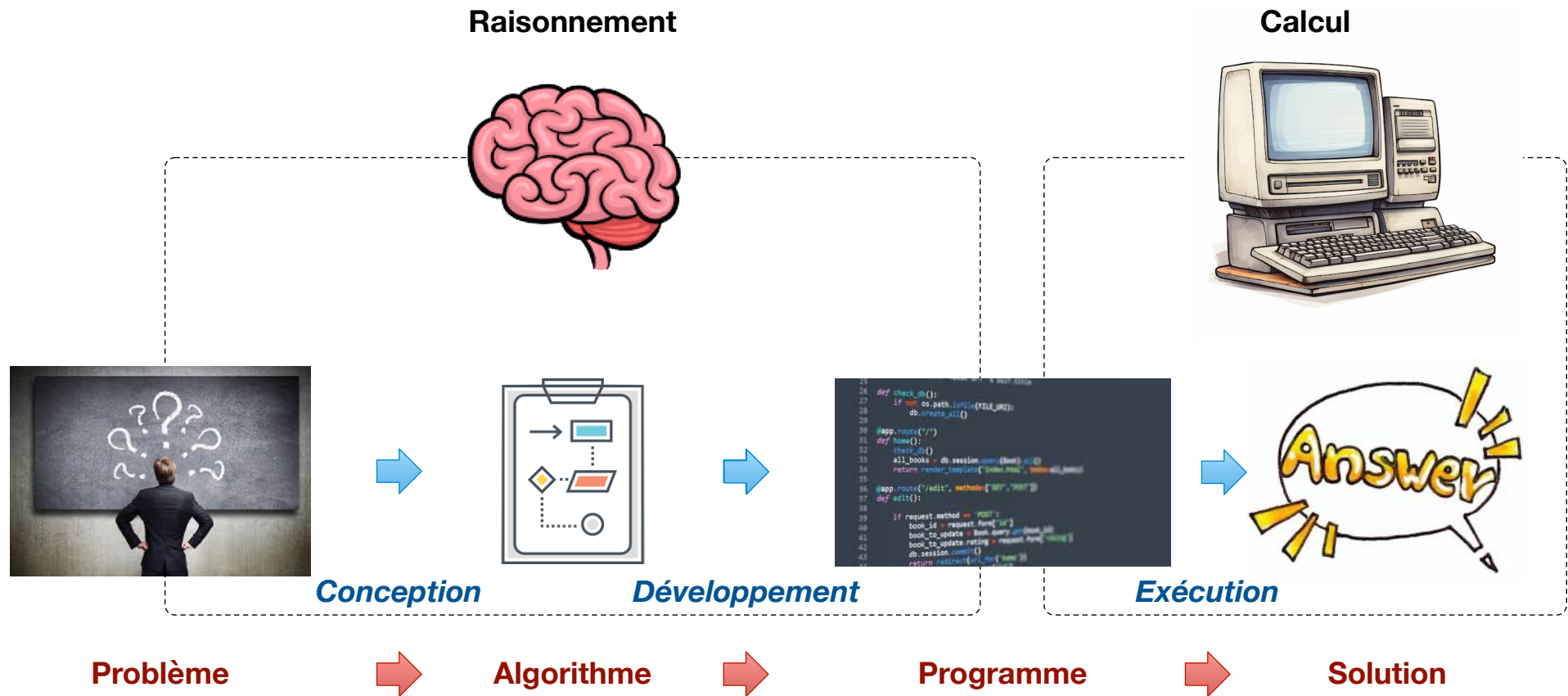
Sébastien Jean

IUT de Valence
Département Informatique

v1.0, 22 septembre 2025

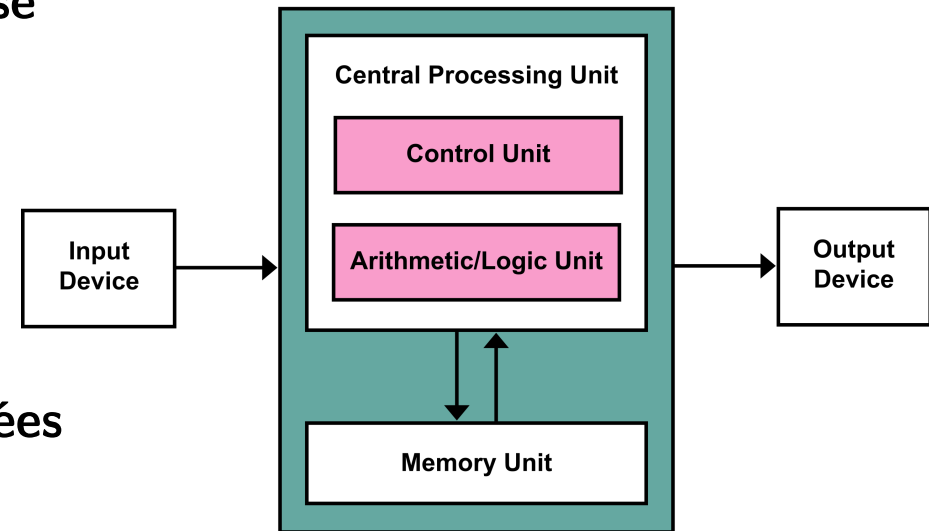
Rappels

- Du problème à la solution



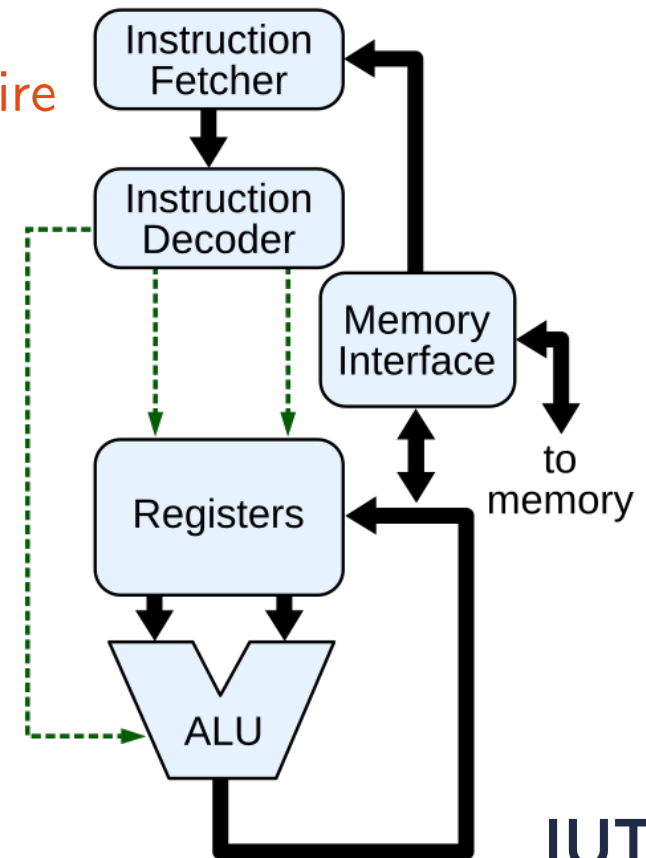
Architecture Von Neumann

- (John Von Neumann, 1945)
- **Modèle architectural d'ordinateur** à 4 composants :
 - **Unité arithmétique et logique (UAL)**
 - Effectue les opérations de base
 - **Unité de contrôle (UC)**
 - Séquence les opérations
 - **Mémoire**
 - Contient programme et données
 - **Dispositif d'entrée-sortie**
 - Permet de communiquer avec le monde extérieur
- $UAL + UC = \text{CPU}$ (*Central Processing Unit*)



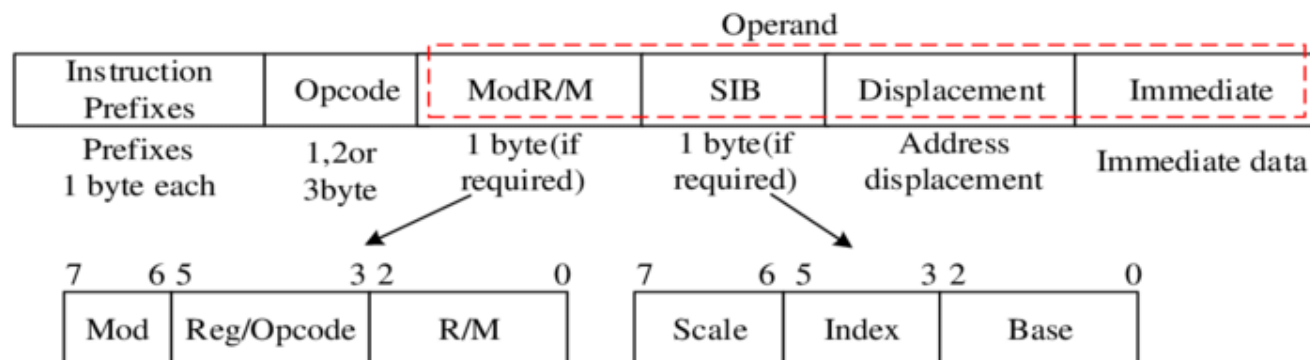
Fonctionnement de base d'un processeur : cycle FDX

- **Lecture en mémoire** de la **prochaine instruction** (*Fetch*)
- **Décodage** de l'instruction (*Decode*)
- **Exécution** de l'instruction (*eXecute*)
 - Les **registres** sont des **emplacements mémoire internes au processeur**
 - Accès en lecture/écriture immédiat
 - N.B. : l'ALU n'a pas d'accès direct à la mémoire mais seulement aux registres



Langage machine

- Ensemble d'instructions compréhensibles directement par le processeur
 - Lecture/écriture en mémoire, opérations arithmétiques/logiques
 - Sauts, entrées/sorties, ...
- Exemple :
 - Jeu d'instructions x86 : 81 instructions (originellement)



- Les premiers ordinateurs *Von Neumann* étaient programmés en langage machine

Niveaux de langage

- Langages de **bas niveau** (langage **machine**, langage **assembleur**)
 - Proches du matériel, **reflètent (quasi-immédiatement) les capacités de la machine**
 - Les programmes ne nécessitent pas ou peu de traitement avant d'être exécutés
- Langages de **haut niveau** (C, Python, Java, ...)
 - Proches du développeur, **reflètent une (ou plusieurs) façon(s) de penser (paradigme)**
 - Les programmes nécessitent un traitement plus ou moins important avant d'être exécutés

Langage d'assemblage (ex. du processeur 8080)

- **Langage de bas niveau** qui représente le langage machine sous une **forme lisible par un humain**
 - Les séquences binaires/hexadécimales du langage machine sont représentées par des symboles (**mnémoniques**) faciles à retenir et des valeurs faciles à exprimer (décimal, hexadécimal)

MVI A, 05h	;	Charger 5 dans le registre A
ADI 07h	;	Ajouter 7 au contenu de A
STA 1000h	;	Stocker le résultat en mémoire à l'adresse 1000h
HLT	;	Arrêter le programme

- Un **assembleur convertit** le langage d'assemblage en langage machine

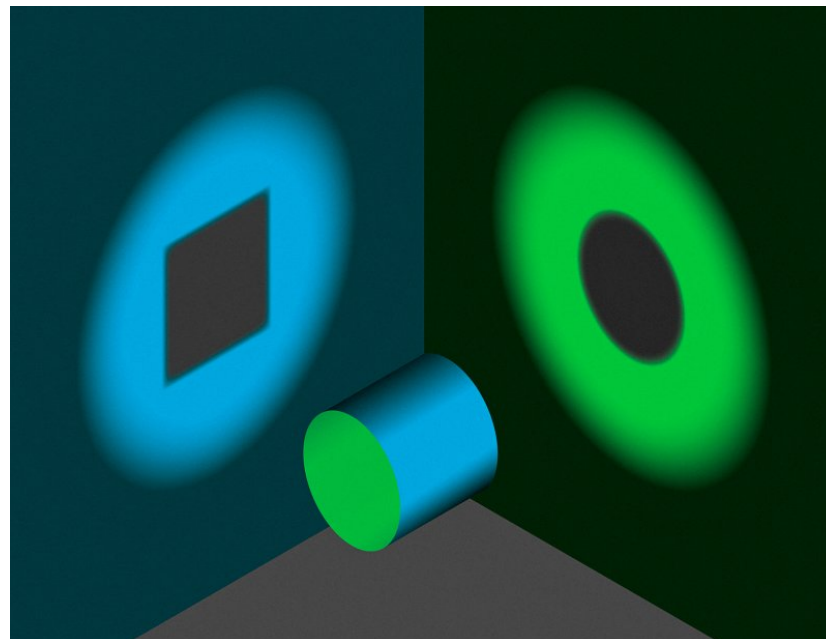
3E 05	;	Charger 5 dans l'accumulateur (registre A)
C6 07	;	Ajouter 7 à l'accumulateur
32 00 10	;	Sauvegarder le résultat en mémoire à l'adresse 1000h
76	;	Arrêter le programme

Paradigmes de programmation

Définition (Wikipedia)

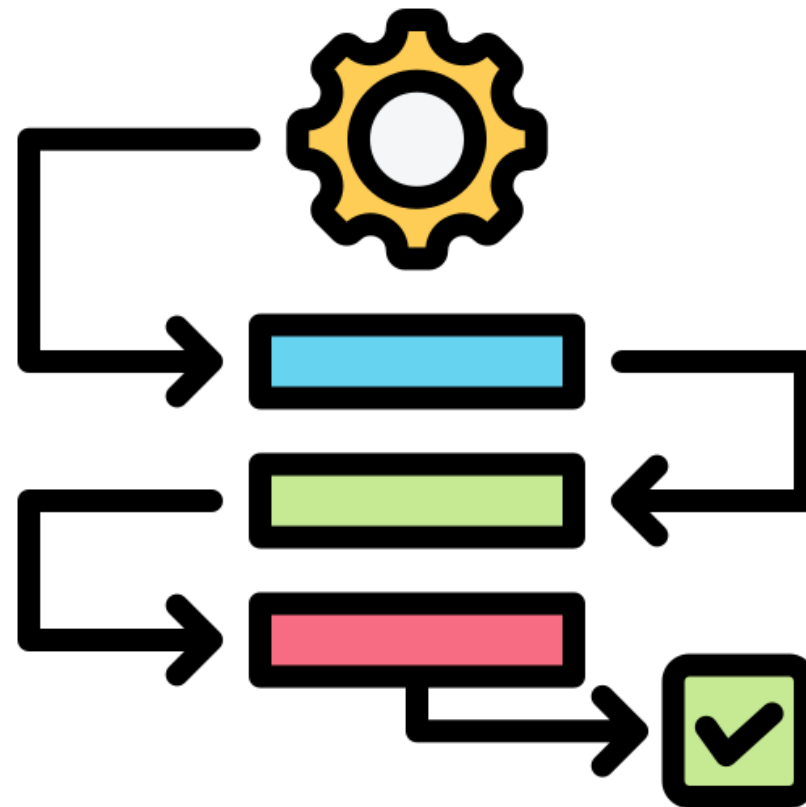
Un **paradigme de programmation** est une façon d'approcher la programmation informatique et de formuler les solutions aux problèmes et leur **formalisation** dans un langage de programmation approprié.

Il détermine la **vue qu'a le développeur de l'exécution** de son programme.



Principaux paradigmes de programmation

- Programmation **impérative** (structurée, procédurale)
 - Centré sur **comment on résout le problème**
 - Décrit les opérations en **séquences d'instructions** exécutées pour modifier l'**état du programme**



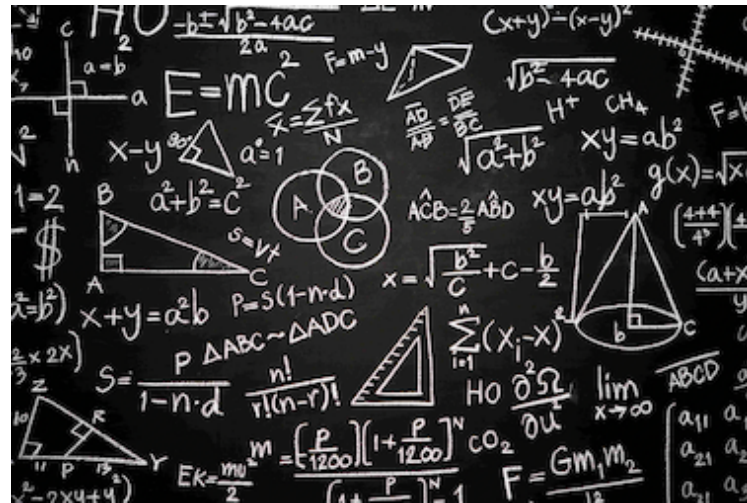
Focus : programmation procédurale

- Basé sur le concept d'**appel procédural**
 - Une procédure (routine, sous-routine ou fonction) contient une **séquence d'instructions** et peut être **appelée à n'importe quelle étape de l'exécution** du programme (**réutilisabilité**)
- Exemple en BASIC :

```
10 N = 10
20 F = 1
30 I = 1
40 IF I > N THEN GOTO 80
50 F = F * I
60 I = I + 1
70 GOTO 40
80 END
```

Principaux paradigmes de programmation

- Programmation **déclarative** (fonctionnelle, logique, par contraintes)
 - Centré sur **quel est le problème**
 - **Aucun état interne**, pas d'effet de bord
 - Correspondance avec une **logique mathématique**



Focus : programmation fonctionnelle

- Considère le calcul en tant qu'évaluation de fonctions mathématiques
 - Imbrication de fonctions *boîtes noires* (plusieurs entrées, une seule sortie)
 - Pas de modification des données, pas d'affectation
- Exemple en LISP

```
(defun factorial (n)
  (if (<= n 1)
      1
      (* n (factorial (- n 1)))))
```

Focus : programmation logique

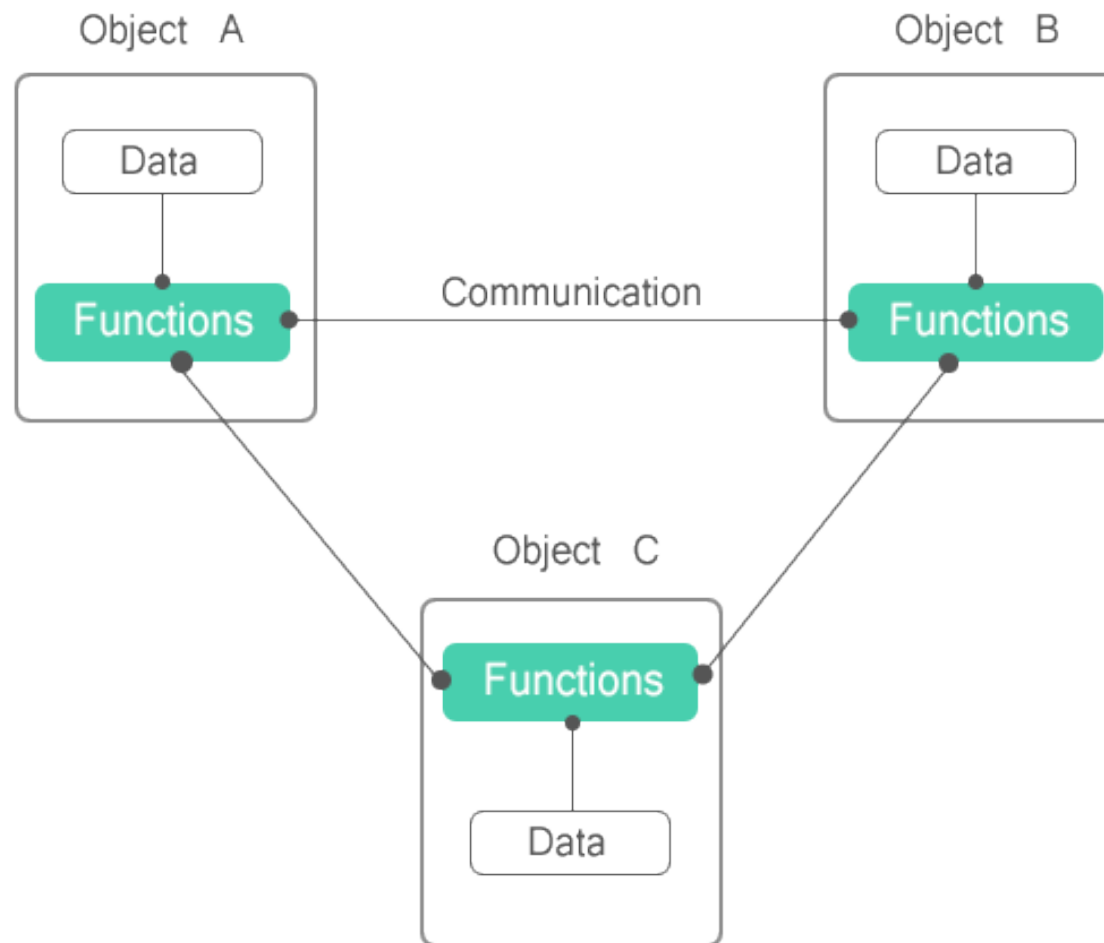
- Définit les applications à l'aide :
 - d'une **base de faits** (vrai/faux)
 - d'une base de **règles** concernant les faits
 - d'un **moteur d'inférence** exploitant faits et règles pour répondre à une question
- Exemple en PROLOG :

```
parent(Darth Vader, Princess Leia).  
parent(Darth Vader, Luke Skywalker).  
parent(Princess Leia, Kylo Ren).  
parent(Han Solo, Kylo Ren).
```

```
grand_parent(X, Y) :-  
    parent(X, Z),  
    parent(Z, Y).
```

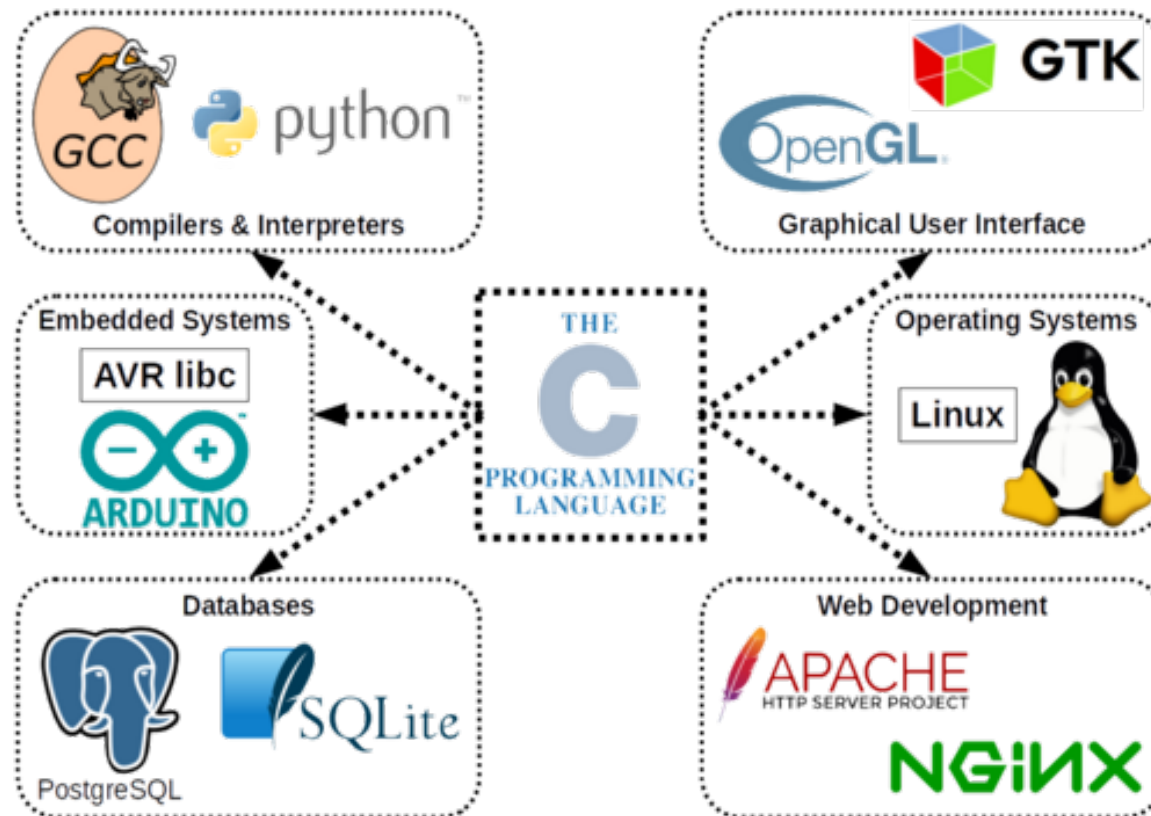
Principaux paradigmes de programmation

- Programmation **Objet** (à classes, à prototypes)
 - Définition et interaction de **briques logicielles** (objets) représentant chacune un **concept** et possédant un **état interne** et un **comportement**



Anatomie de quelques langages usuels

- **C**
 - Dennis Ritchie (1972)
 - **Impératif** (structuré / procédural)



Anatomie de quelques langages usuels

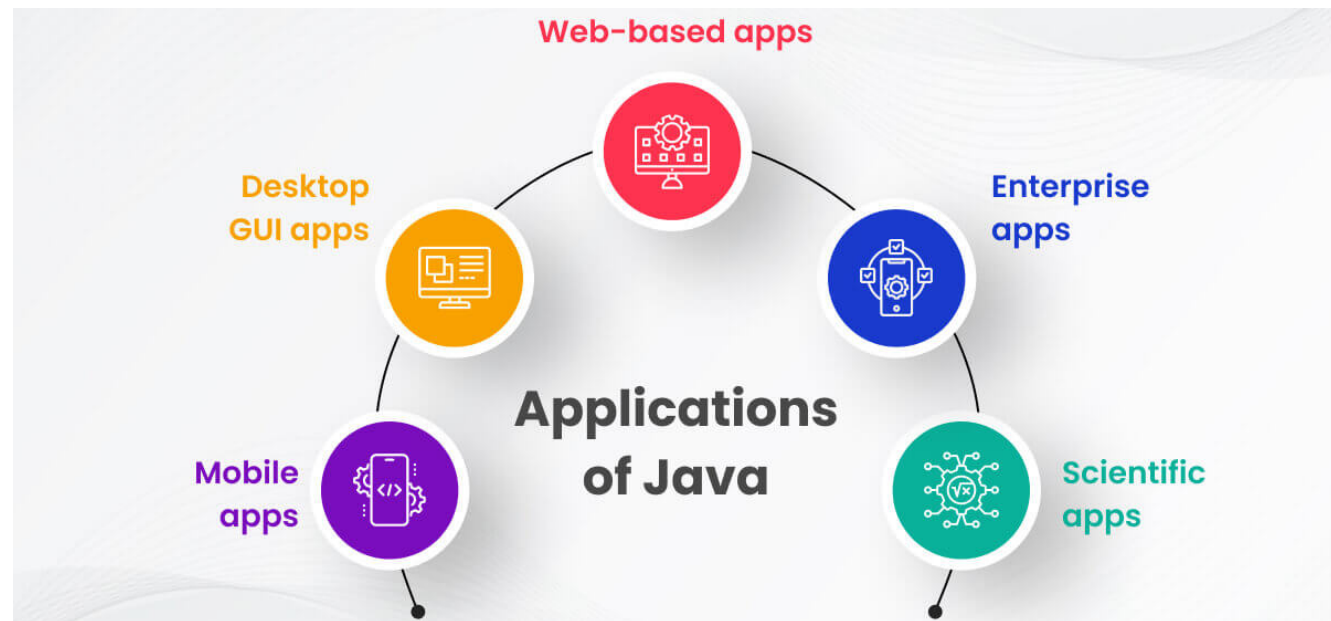
- **C++**
 - Bjarne Stroustrup (1985)
 - **Multi-paradigmes**
 - **Objet** (à classes)
 - **Impératif**
 - **Fonctionnel**



Anatomie de quelques langages usuels

- **Java**

- James Gosling (1995)
- **Multi-paradigmes**
 - **Objet** (à classes)
 - **Impératif**
 - **Fonctionnel**



Anatomie de quelques langages usuels

- **Python**

- Guido Van Rossum (1991)

- **Multi-paradigmes**

- **Objet** (à prototypes)

- **Impératif**

- **Fonctionnel**



7 Things Python Can Do

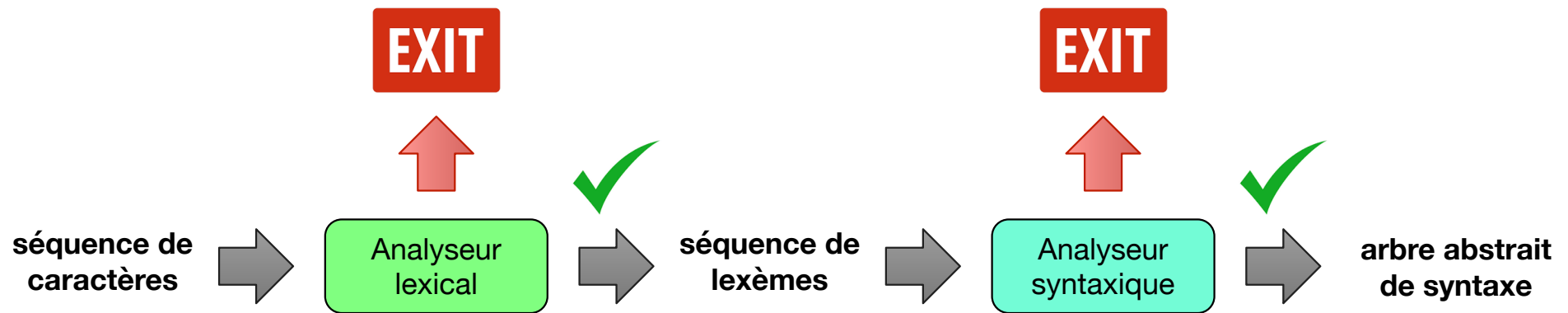
1	Data science
2	Machine learning
3	Web development
4	Financial analysis
5	Desktop applications
6	Business applications
7	Scripting and utility software

Du code source à l'exécution

- 3 manières de **passer du code source à son exécution**
 - **Compilation**
 - **Interprétation**
 - Utilisation d'une **machine virtuelle**

Étapes préliminaires de traitement du code source

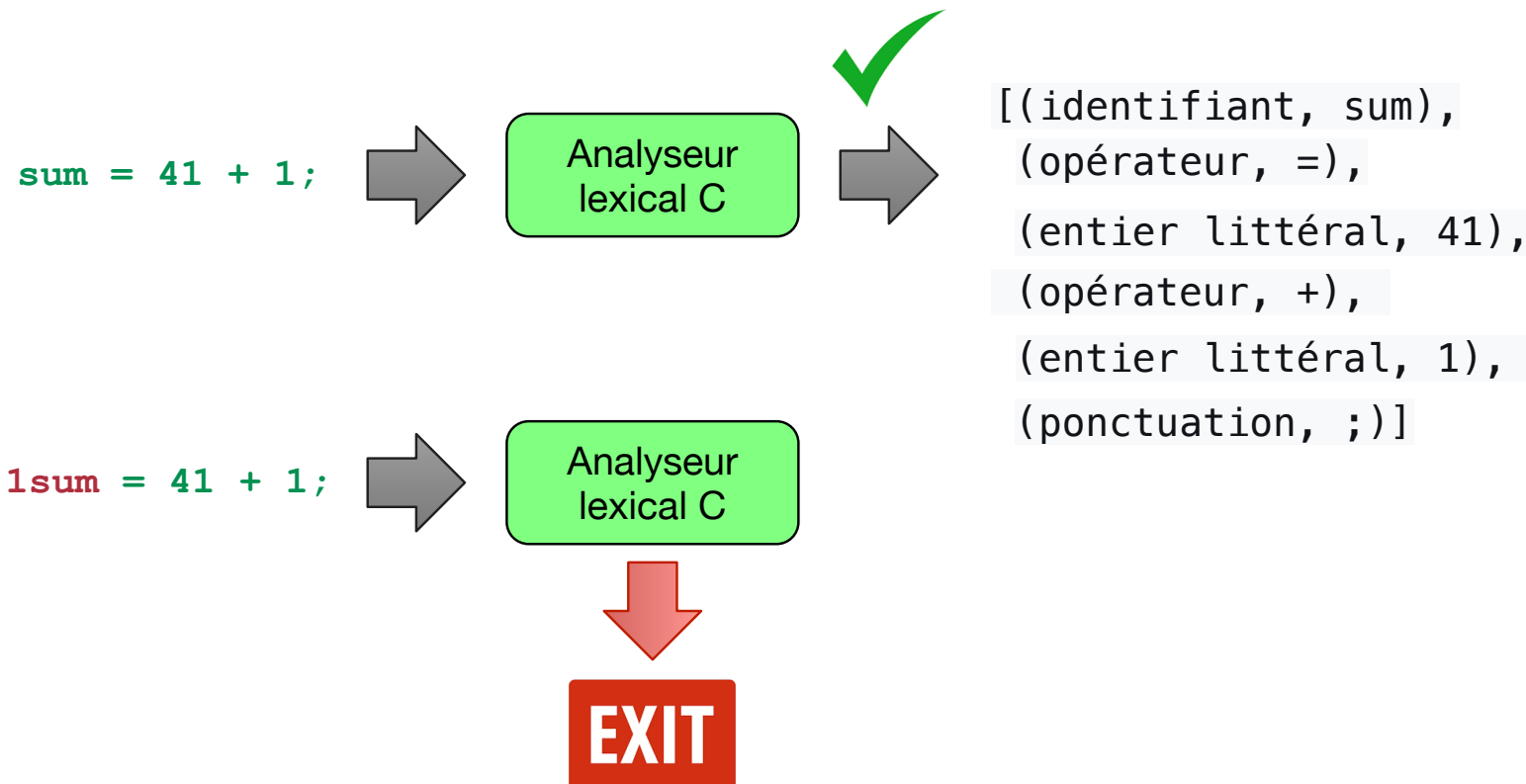
- Les processus de compilation, d'interprétation et d'exécution ont **quelques étapes/outils en commun**



- Durant ces étapes, des **erreurs** peuvent être **détectées et signalées**

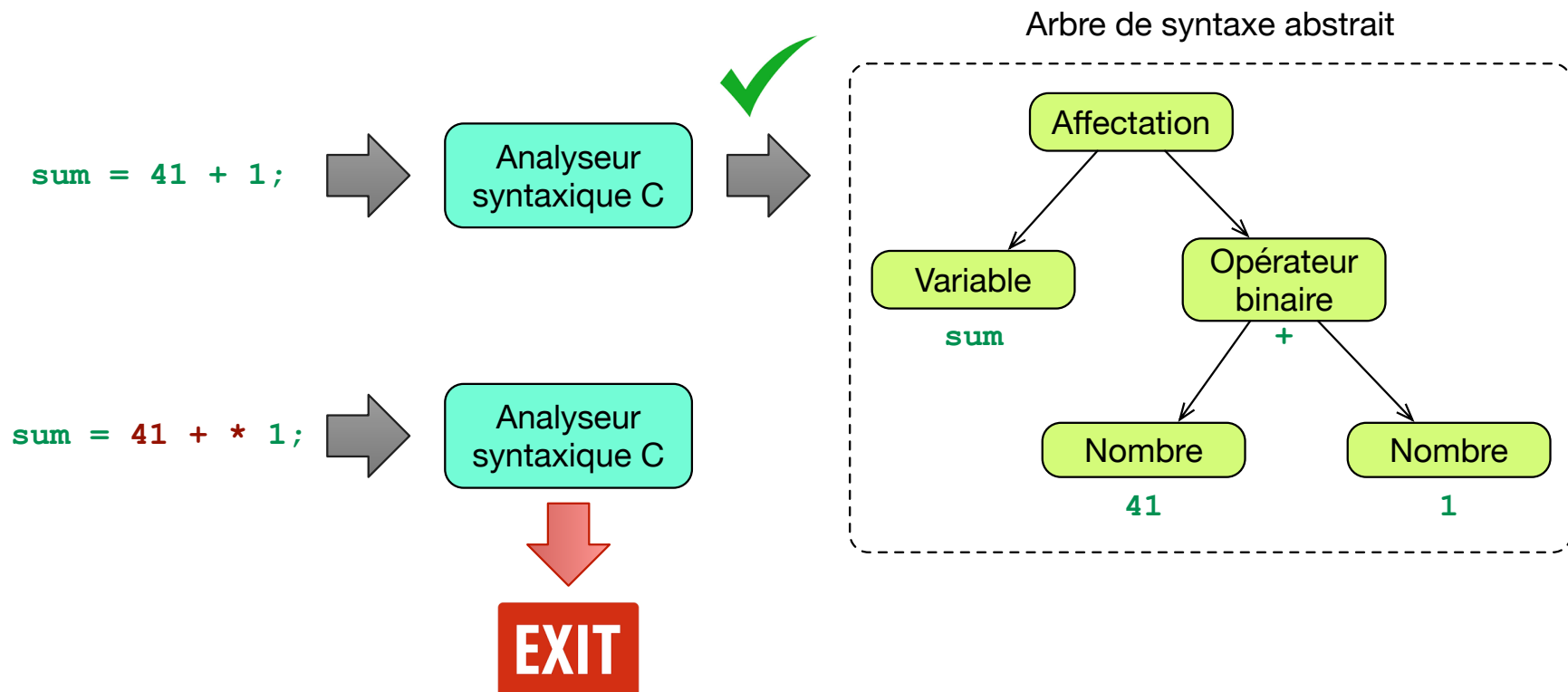
Analyse lexicale

- Chaque langage est défini par ses **unités lexicales (lexèmes)**
 - Identifiants, mots-clés réservés, ponctuation, opérateurs, littéraux
- L'**analyseur lexical découpe** la suite de caractères formant le code source en **lexèmes**

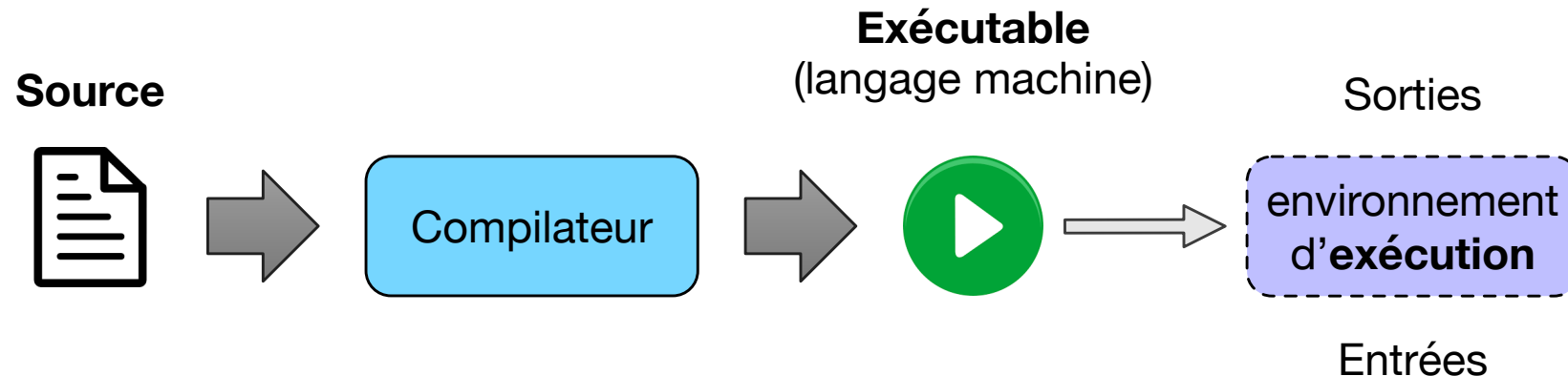


Analyse syntaxique

- Chaque langage est défini par sa **grammaire**
 - Règles de syntaxe, de constructions d'instructions valides
- L'**analyseur syntaxique** vérifie le respect de la grammaire et transforme la suite de lexèmes en un **arbre de syntaxe abstrait**

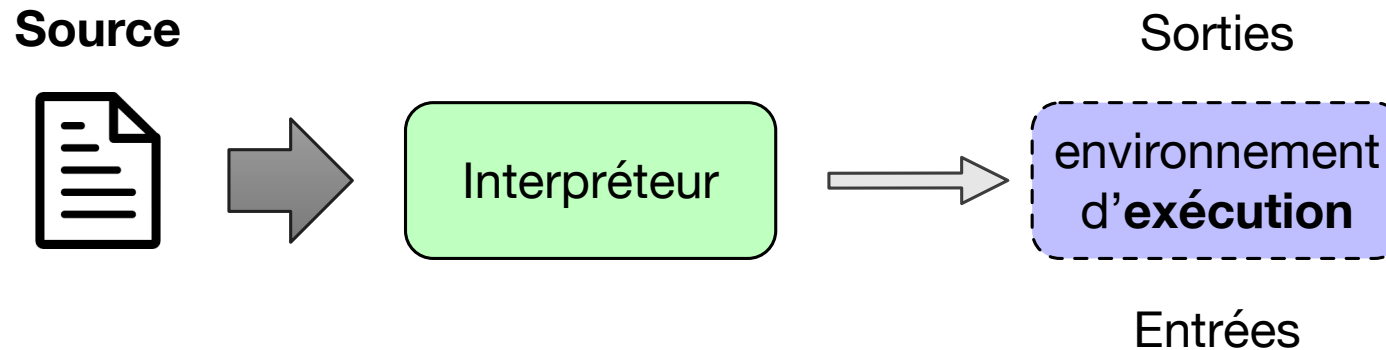


Compilation



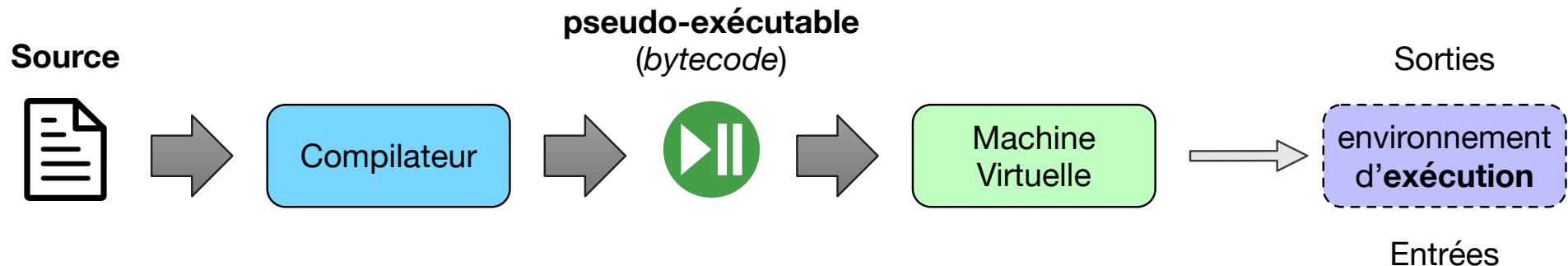
- La **chaîne de compilation** (*toolchain*) transforme un **code source** en un **fichier exécutable** (instructions en langage machine)
- L'exécutable est **généré** pour une **architecture (processeur)** et un **système d'exploitation** (ex : Intel x86 64 bits / Linux)
 - Il n'est pas portable
 - Pour être exécuté sur un autre couple architecture/système le fichier source doit être recompilé
- On parle de **compilation croisée** quand le compilateur s'exécute sur une architecture/système **différente de la cible finale**

Interprétation



- Un **interpréteur** est un **programme exécutable** qui **traduit** les instructions (en langage de haut niveau) d'un fichier source en instructions d'un langage machine (indirectement)
- Il est écrit dans un langage de haut niveau, il peut être adapté (porté) facilement pour pouvoir être compilé sur des architectures/systèmes différents
- Le fichier source est portable dans la mesure où il suffit d'un interpréteur sur la cible pour l'exécuter

Utilisation d'une machine virtuelle



- Une **machine virtuelle** est un **programme exécutable** qui **donne l'illusion d'être une machine réelle**
- Elle exécute des instructions dans son langage machine (**bytecode**)
- Le *bytecode*, proche d'un langage machine réel, est facile à traduire
- Le compilateur transforme le code source en un fichier *bytecode*
- Le fichier source n'est pas portable mais le fichier *bytecode* est portable

Exécution pour les langages usuels



- Compilation, interprétation ou utilisation d'une VM ?



Fin !

