Algorithmique et programmation

7 octobre 2016

David Pichardie

Langage support

- Langage de programmation multi-paradigme (objet, fonctionnel, concurrent)
- Points forts
  - une syntaxe épurée
  - des librairies de calculs scientifiques puissantes (numpy, sage)
  - très populaire : vos problèmes de syntaxe trouveront toujours une réponse sur le web…

Factoriel

C

```c
#include<stdio.h>
#include<stdlib.h>

long fact(int n)
{
   if(n == 0)
      return 1;
   else
      return n * fact(n-1);
}

int main(int argc, char *argv[])
{
  int n = atoi(argv[1]);
  for (int i=0; i < n; i++) {
    printf("(%d, %ld)\n", i, fact(i));
  }
  return 0;
}
```

Factoriel

OCaml

```
let rec fact n =
  if n = 0 then 1
  else n * fact (n-1)

let _ =
  let n = int_of_string Sys.argv.(1) in
  for i=0 to n-1 do
    Printf.printf "(%d, %d)\n" i (fact i)
  done
```

Factoriel

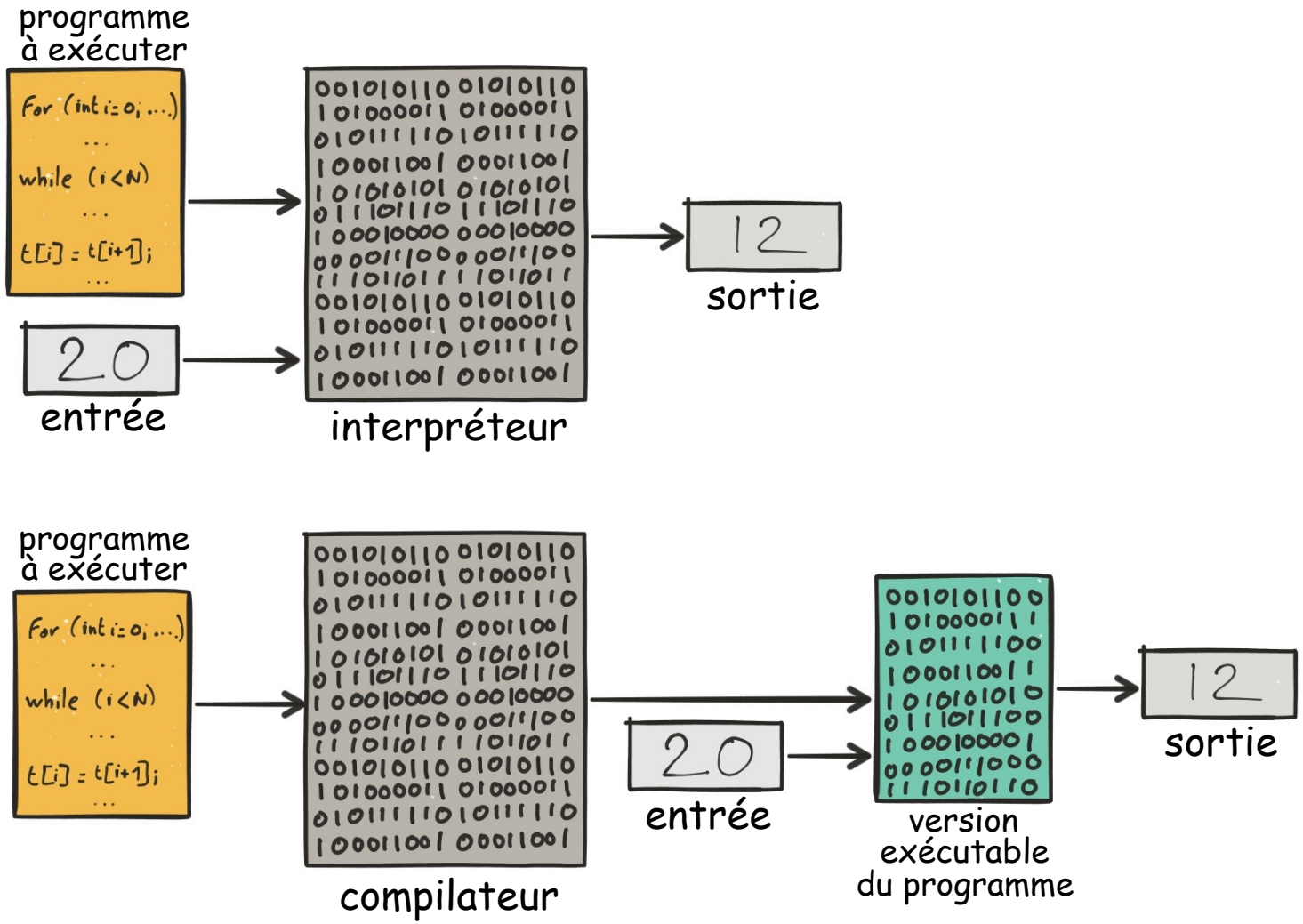
Python

```python
import sys

def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n-1)

n = int(sys.argv[1])
for i in range(n):
    print (i,fact(i))
```

Modes d’exécution

- Aucun des trois programmes précédents n’est écrit dans un format directement exécutable par un ordinateur
- Solution 1 : un *interpréteur* exécute le programme
- Solution 2 : un *compilateur* transforme le programme en un programme exécutable

Interpréteur/compilateur



Interpréteur/compilateur

Exemples

Interpréteur

```
$ python fact.py 5
(0, 1)
(1, 1)
(2, 2)
(3, 6)
(4, 24)
```

```
$ ocaml fact.ml 5
(0, 1)
(1, 1)
(2, 2)
(3, 6)
(4, 24)
```

Compilateur

```
$ gcc -o fact fact.c
$ ./fact 5
(0, 1)
(1, 1)
(2, 2)
(3, 6)
(4, 24)
```

```
$ ocamlc -o fact fact.ml
$ ./fact 5
(0, 1)
(1, 1)
(2, 2)
(3, 6)
(4, 24)
```

Python C OCaml

Modes d’exécution comparaison

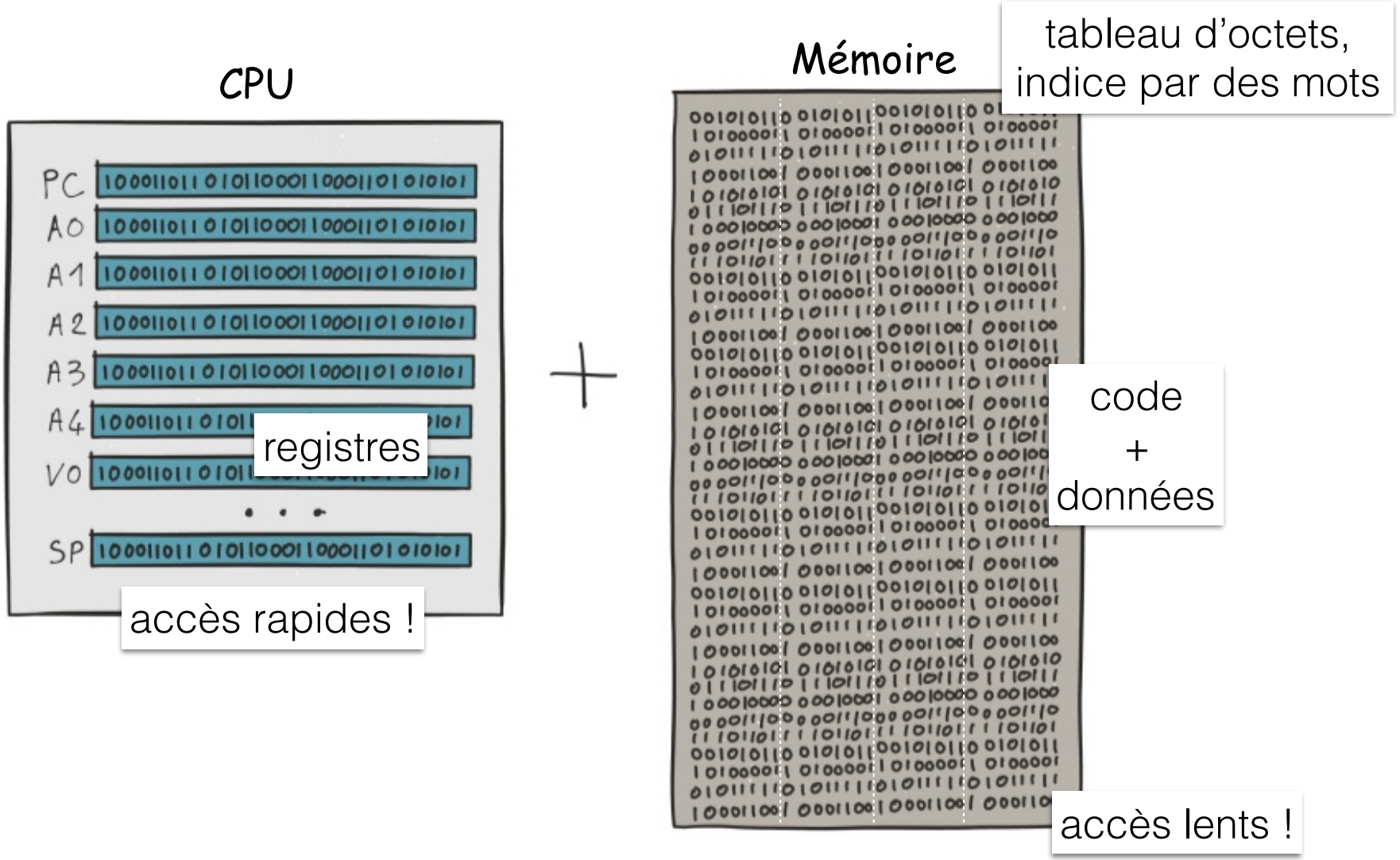
- Le mode compilé augmente généralement l’efficacité
- Le mode interprété améliore la portabilité
- Un mode hybride : la compilation à la volée

Programme machine ?

Vocabulaire

- Le bit : 0 ou 1
- L’octet : 8 bits
- Le mot : 32 ou 64 bits (selon l’architecture)

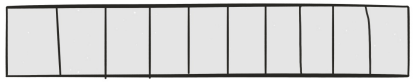
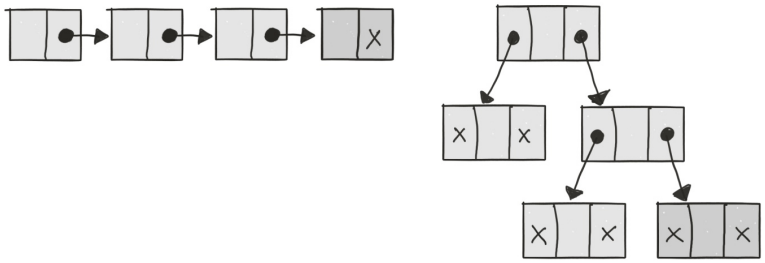
Une machine simplifiée



Exécution d'un programme machine

- Le registre PC contient l'adresse de la prochaine instruction à exécuter
- Les 4 (ou 8) octets à cette adresse sont lues et décodées comme une instruction
- L'instruction est exécutée (ce qui peut modifier l'état courant des registres et de la mémoire)
- Le registre PC est mis à jour et on recommence…

Les données

- Elles peuvent représenter des nombres dans des formats divers…
  - entiers de capacités limités,
  - un sous-ensemble des nombres réels : les *flottants*
- Mais aussi des données structurées
  - tableaux 
  - structures chaînées 

Récapitulatif

- Les programmes sont exécutés selon deux modes d'exécution : compilation ou interprétation
- Le langage machine s'appuie sur la notion de mots de bits et suit un algorithme d'exécution extrêmement simple. Le compilateur est en charge de traduire nos programmes vers cette représentation.
- Les données sont représentées par des séquences de bits.

Exercice : tris

- une procédure de tri appliquée à un tableau de nombres `t`, doit modifier le tableau de façon à conserver ses éléments (et leurs nombres d'occurrence) et les classer par ordre croissants.

$$\texttt{t}[0] \leq \texttt{t}[1] \leq \cdots \texttt{t}[\texttt{len}(t) - 1]$$

Exercice : Tri insertion

- Écrire une fonction insert_sort(t), qui trie en place un tableau en utilisant l'algorithme du tri insertion.

- Cet algorithme procède ainsi: pour k = 0,…, len(t), on insère le k-ième élément du tableau à sa place dans les k premières cases (les k-1 premières étant déjà triées), en parcourant t[0…k-1] de la droite vers la gauche.

Exercice : Tri insertion

```
def insert_sort(t):
    for i in range(len(t)):
        # on suppose que t[0..i-1] est déjà trié
        # on décale la valeur t[i] vers la gauche pour trier t[0..i]
        for j in range(i,0,-1):
            # range(i,0,-1) = [i,i-1,...,1]
            if t[j] < t[j-1]:
                t[j-1], t[j] = t[j], t[j-1]
            else:
                break

#### Tests ####
t = [56, 89, 77, 9, 18, 73, 75, 59, 12, 53]
print "t =", t
insert_sort(t)
print "insert_sort(t) =", t
```

Exercice : Tri rapide

- Écrire une fonction quick_sort(t), qui trie en place un tableau en utilisant l'algorithme du tri rapide. L'algorithme procède ainsi :
- Si le tableau a plus d'un élément, choisir un pivot (un élément quelconque du tableau).
- Mettre tous les éléments plus petits que le pivot à gauche de celui-ci, et ceux plus grands à droite de celui-ci (indication : utiliser deux indices i, j, l'un croissant, l'autre décroissant).
- Faire un appel récursif sur les deux sous-tableaux ainsi créés.