# IFT1015 Programmation 1
# Introduction à la programmation

Marc Feeley
(avec ajouts de Aaron Courville, Pascal Vincent et Stefan Monnier)

# Quiz!

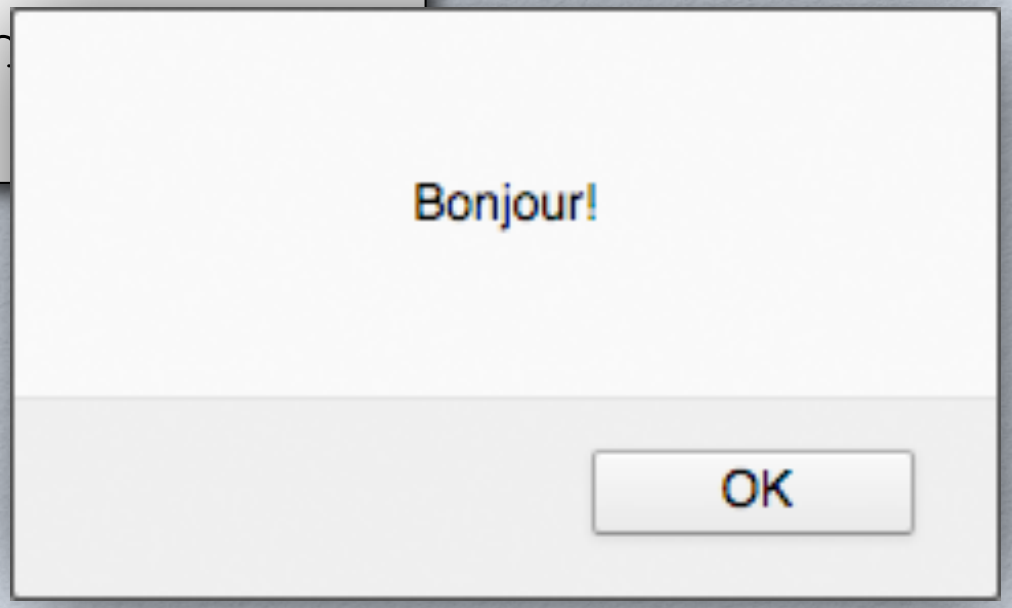
# Programme #1

```
alert("Bonjour!");
```

- Quel *langage de programmation*?
- Que fait le programme?

# Programme #1

```
alert("Bon
```




- Quel *langage de programmation*? Rép: JavaScript
- Que fait le programme? Rép: affiche un message

# Programme #2

```
public class Bonjour {
    public static void main(String[] args) {
        System.out.println("Bonjour!");
    }
}
```

- Quel *langage de programmation*?
- Que fait le programme?

# Programme #2

```
public class Bonjour {
    public static void main(Stri
        System.out.println("Bonj
    }
}
```

```
$ javac Bonjour.java
$ java Bonjour
Bonjour!
```

- Quel *langage de programmation*? Rép: Java
- Que fait le programme? Rép: affiche un message

# Programme #3

```
(display (* 4 (atan 1)))
```

- Quel *langage de programmation*?
- Que fait le programme?

# Programme #3

```
(display (* 4 (atan 1)))
```

```
$ gsi -e "(display (* 4 (atan 1)))"
3.141592653589793
```

- Quel *langage de programmation*? Rép: Scheme
- Que fait le programme? Rép: affiche $\pi$ (approx)

# Programme #4

```
%!PS
/Times-Roman findfont 200 scalefont setfont
200 80 moveto
60 rotate
(IFT) show
1 0 0 setrgbcolor (1015) show
showpage
```

- Quel *langage de programmation*?
- Que fait le programme?

# Programme #4

```
%!PS
/Times-Roman findfont 200 scalefont s
200 80 moveto
60 rotate
(IFT) show
1 0 0 setrgbcolor (1015) show
showpage
```


- Quel *langage de programmation*? Rép: Postscript
- Que fait le programme? Rép: imprime un message

# Programme #5

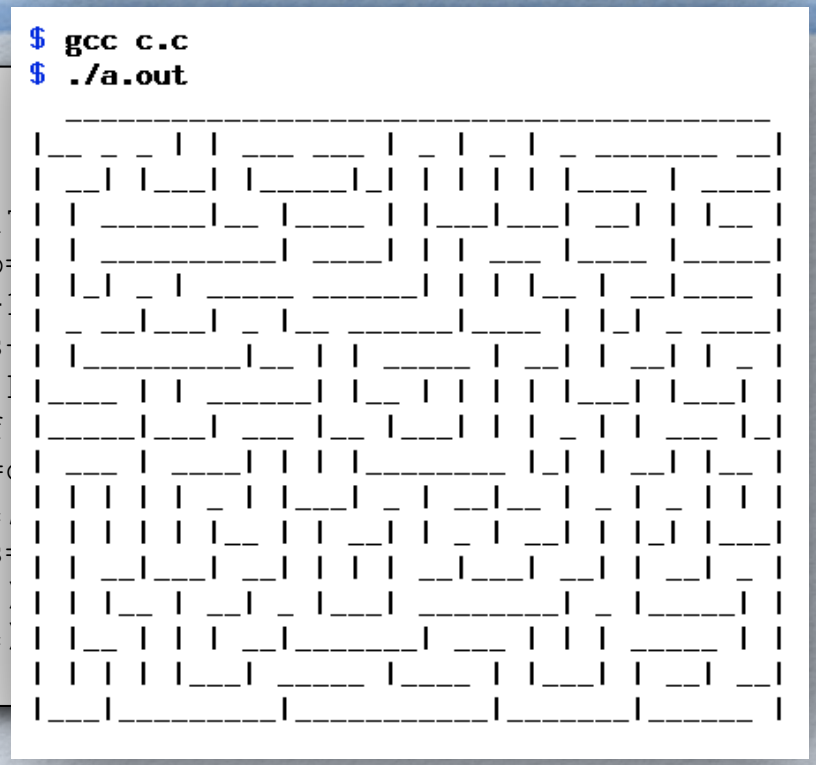
```
#define P(X)j=write(1,X,1)
#define C 21
int M[5000]={2},*u=M,N[5000],R=17,a[4],l[]=
{0,-1,C-1,-1},m[]={1,-C,-1,C},*b=N,*d=N,c,e
,f,g,i,j,k,s;main(){for(M[i=C*R-1]=24;f|d>=
b;){c=M[g=i];i=e;for(s=f=0;s<4;s++)if((k=m[
s]+g)>=0&&k<C*R&&l[s]!=k%C&&(!M[k]||!j&&c>=
16!=M[k]>=16))a[f++]=s;rand();if(f){f=M[e=m
[s=a[rand()%f]]+g];j=j<f?f:j;f+=c&-16*!j;M[
g]=c|1<<s;M[*d++=e]=f|1<<(s+2)%4;}else e=d>
b++?b[-1]:e;}P(" ");P(" ");for(s=C;--s;P("_"
))P("_");for(;P("\n"),R--;P("|"))for(e=C;e--
;P("_"+(*u++/8)%2))P("|_"+(*u/4)%2);}
```

- Quel *langage de programmation*?
- Que fait le programme?

# Programme #5

```
#define P(X)j=write(1,X,1)
#define C 21
int M[5000]={2},*u=M,N[5000],R=1
{0,-1,C-1,-1},m[]={1,-C,-1,C},*b=
,f,g,i,j,k,s;main(){for(M[i=C*R-
b;){c=M[g=i];i=e;for(s=f=0;s<4;s+
s]+g)>=0&&k<C*R&&l[s]!=k%C&&(!M[
16!=M[k]>=16))a[f++]=s;rand();if
[s=a[rand()%f]]+g];j=j<f?f:j;f+=
g]=c|1<<s;M[*d++=e]=f|1<<(s+2)%4
b++?b[-1]:e;}P(" ");P(" ");for(s=
))P("_");for(;P("\n"),R--;P("|")
;P("_"+(*u++/8)%2))P("|_"+(*u/4
```

```
$ gcc c.c
$ ./a.out
```



- Quel *langage de programmation*? Rép: C
- Que fait le programme? Rép: affiche un labyrinthe

# Programme #6


- Quel *langage de programmation*?
- Que fait le programme?

# Programme #6

```
11101000          call    x
00001001
00000000
00000000
00000000
01000010          .byte   'B'
01101111          .byte   'o'
01101110          .byte   'n'
01101010          .byte   'j'
01101111          .byte   'o'
01110101          .byte   'u'
01110010          .byte   'r'
00100001          .byte   '!'
00001010          .byte   '\n'
01011001      x:  pop     %ecx
00110001          xor     %ebx, %ebx
11011011
01000011          inc     %ebx
10001101          lea     8(%ebx), %edx
01010011
00001000
10001101          lea     3(%ebx), %eax
01000011
00000011
11001101          int     $128
10000000
00110001          xor     %ebx, %ebx
11011011
10001101          lea     1(%ebx), %eax
01000011
00000001
11001101          int     $128
10000000
```

- Quel *langage de programmation*?
- Que fait le programme?

# Programme #6

code machine

```
11101000
00001001
00000000
00000000
00000000
01000010
01101111
01101110
01101010
01101111
01110101
01110010
00100001
00001010
01011001
00110001
11011011
01000011
10001101
01010011
00001000
10001101
01000011
00000011
11001101
10000000
00110001
11011011
10001101
01000011
00000001
11001101
10000000
```

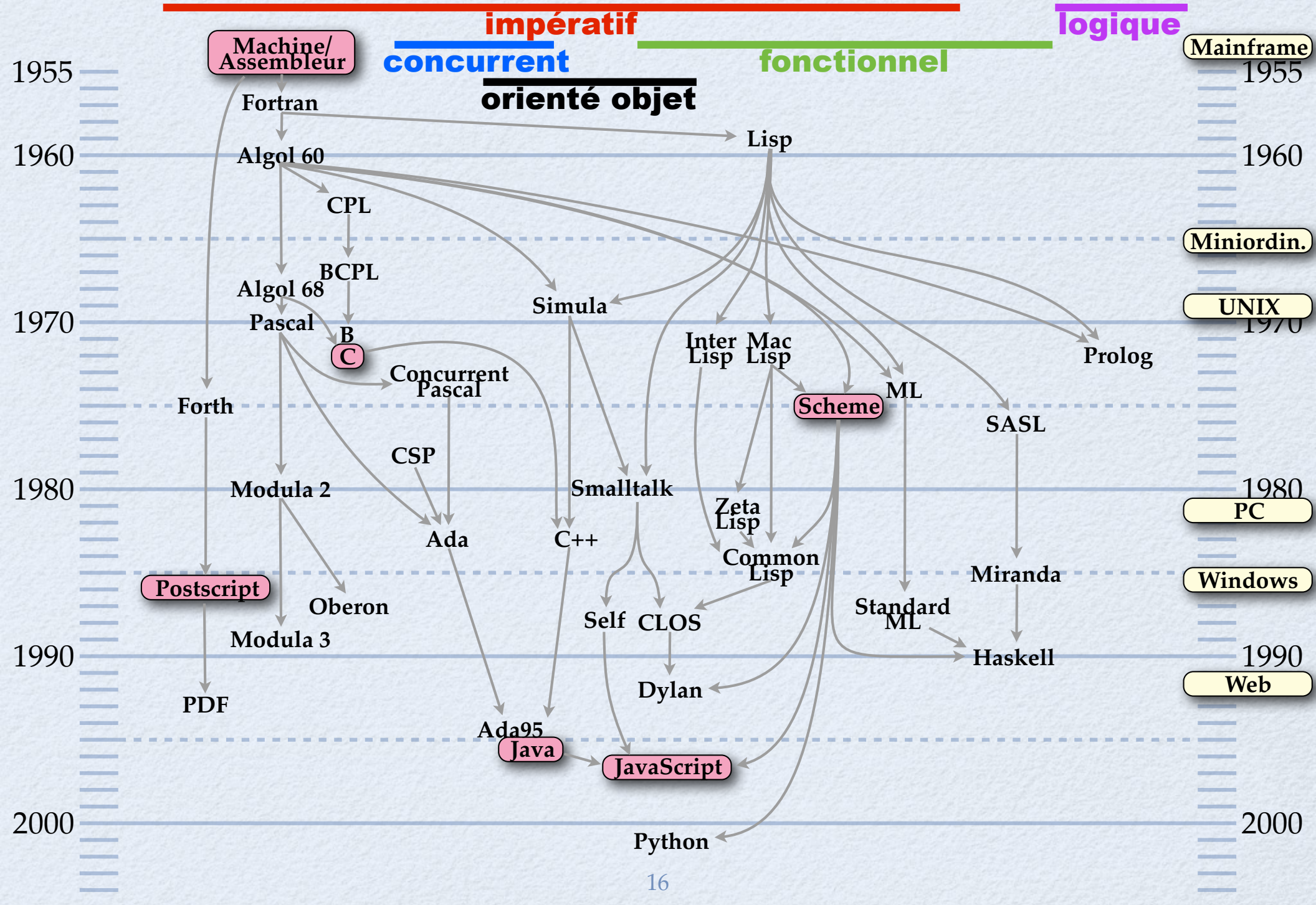
```
    call    x

            .byte   'B'
            .byte   'o'
            .byte   'n'
            .byte   'j'
            .byte   'o'
            .byte   'u'
            .byte   'r'
            .byte   '!'
            .byte   '\n'
x:  pop     %ecx
    xor     %ebx, %ebx

    inc     %ebx
    lea     8(%ebx), %edx

    lea     3(%ebx), %eax

    int     $128

    xor     %ebx, %ebx

    lea     1(%ebx), %eax

    int     $128
```

code assembleur

```
$ as --32 -o bonjour.o bonjour.as
$ ld -melf_i386 -s -o bonjour bonjour.o
$ ./bonjour
Bonjour!
```

- Quel *langage de programmation*? Rép: x86-32
- Que fait le programme? Rép: affiche un message

# Plus de 2000 langages de programmation!



# Définitions de base

# Programme

- **Programme** : une *description* des opérations de traitement d'information à effectuer pour réaliser une tâche spécifique
- **Application** : un ou plusieurs programmes qui fournissent un *service* à son utilisateur (jeu vidéo, traitement de texte, fûreteur Web, …)
- **Logiciel** : terme générique pour tout programme ou application

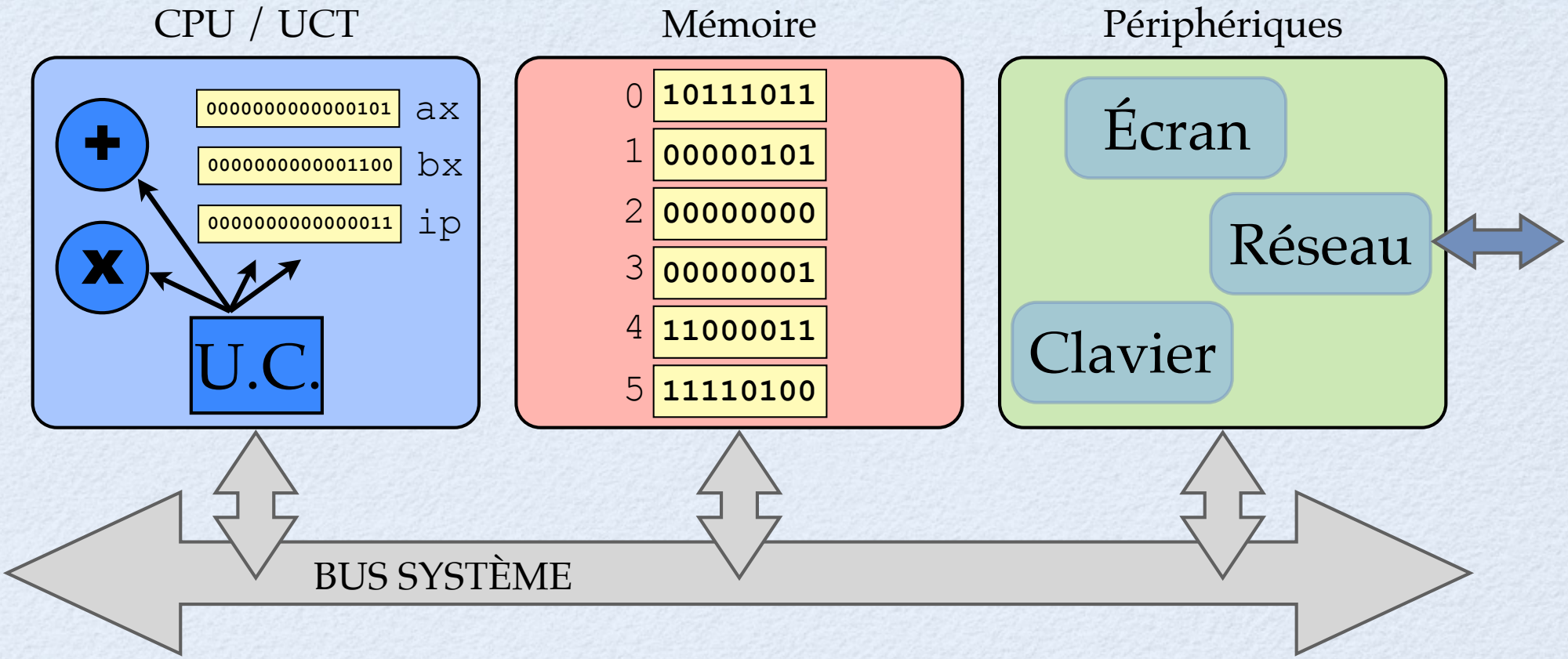
# Langage de programmation

- **Langage de programmation** : ensemble de règles définissant la forme que doit prendre un programme valide (sa **syntaxe**) et le sens qui y est attaché (sa **sémantique**)
  - C, Java, JavaScript, Pascal, Python, x86, …
- La **définition d'un langage** peut être plus ou moins formelle (*tutoriel, manuel de référence, norme internationale, grammaire/sémantique*)

# Processeur

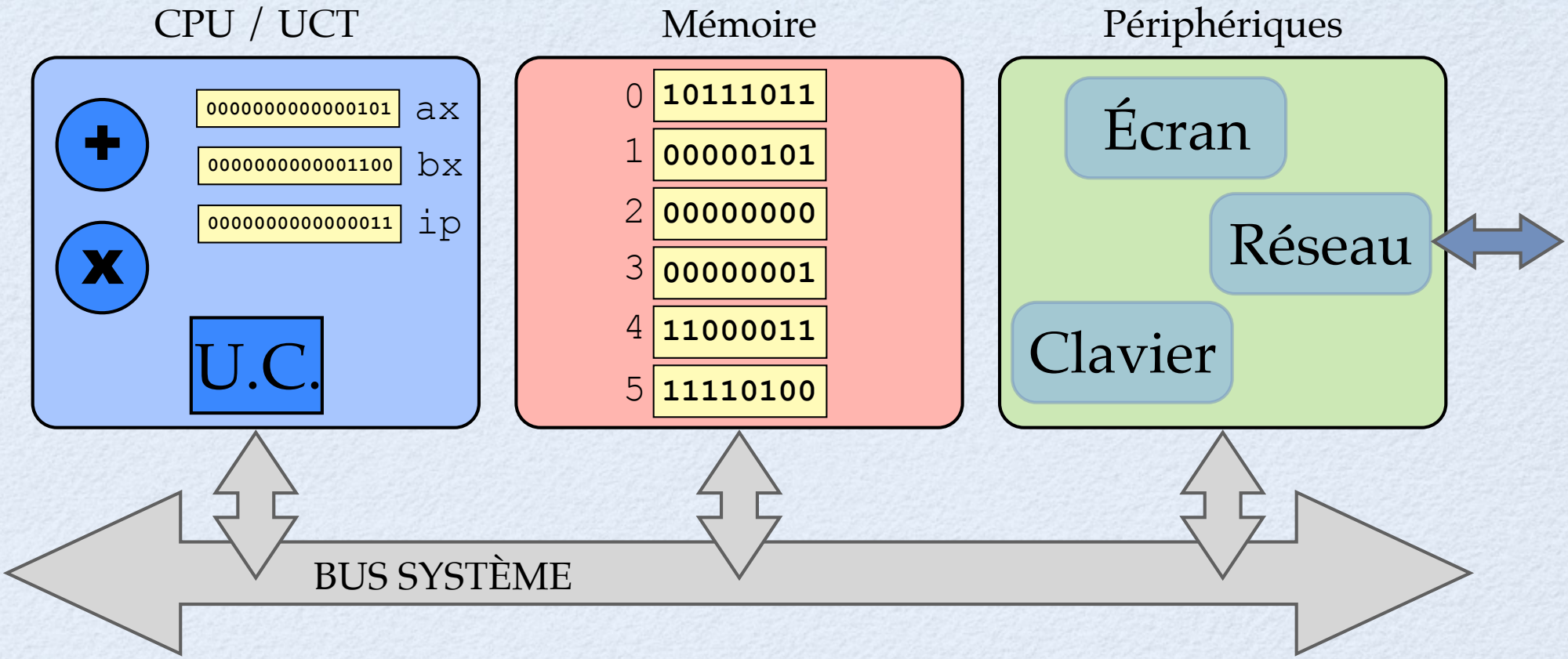
- **Code** : représentation d'un programme dans un *langage de programmation spécifique*
- **Processeur** : un dispositif de traitement d'information qui fait l'**exécution** du code
- **Ordinateur** : un processeur réalisé en **matériel** pouvant exécuter du code en **langage machine** (x86, ARM, MIPS, ...)

# Ordinateur




- Modèle typique : **architecture de Von Neumann** (code et données dans la même mémoire)
- L'**unité de contrôle** (U.C.) décide quoi faire quand

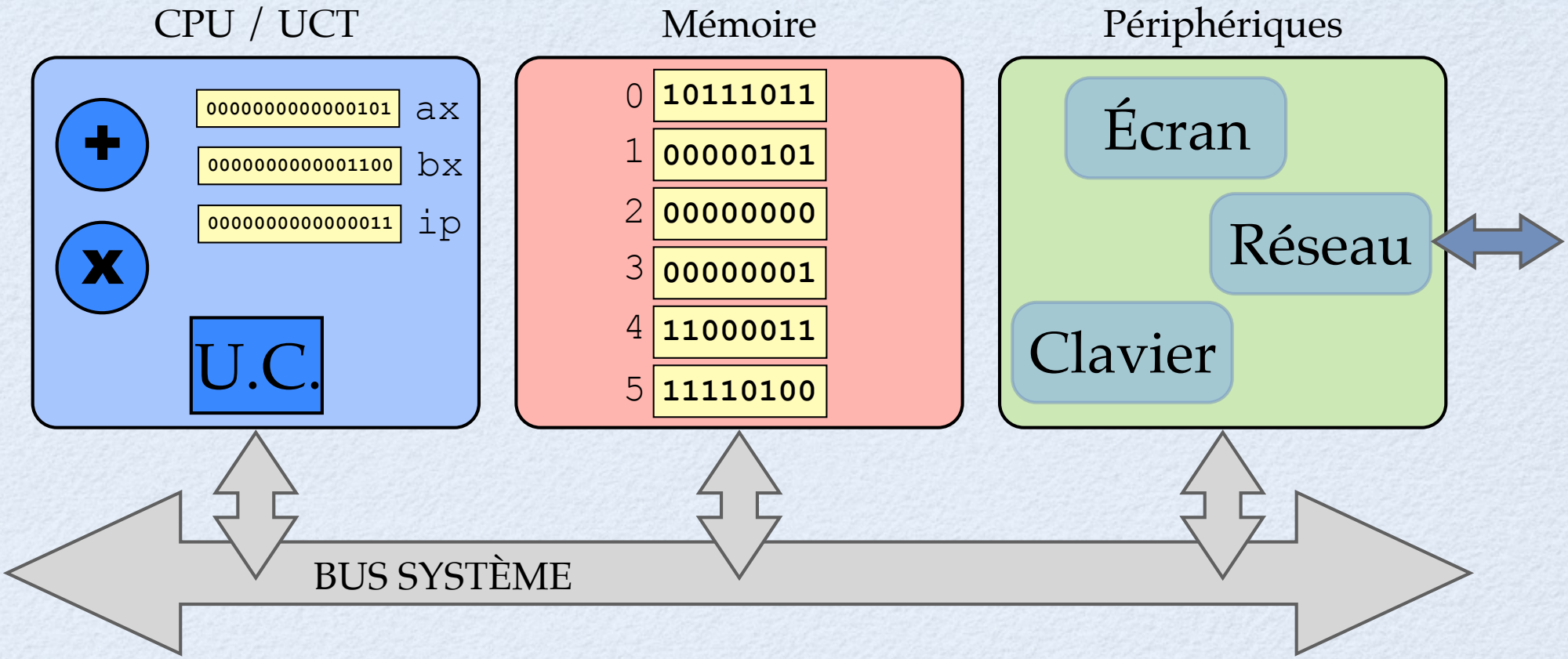
# Ordinateur




- Le code et les données sont encodées en **binaire** :

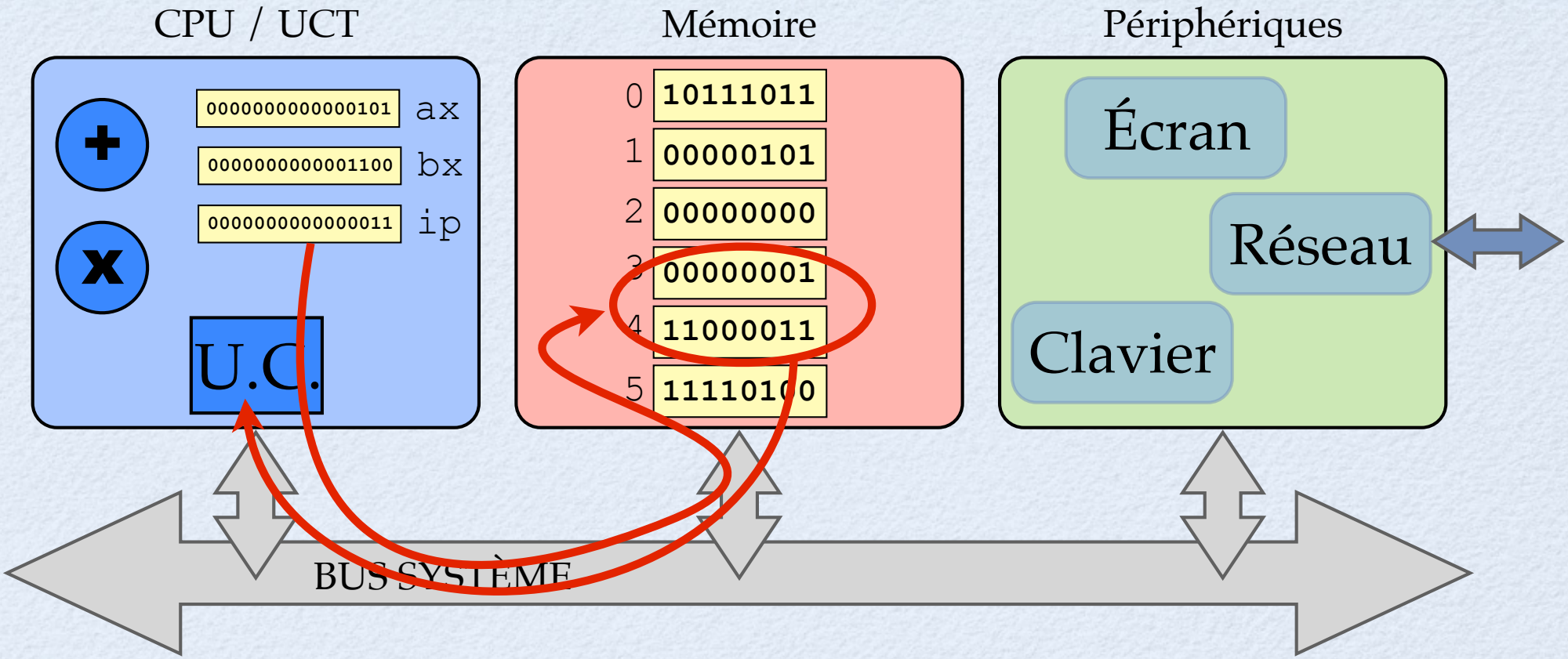
$$00000101_2 = 5_{10}$$

# Fonctionnement de l'ordinateur



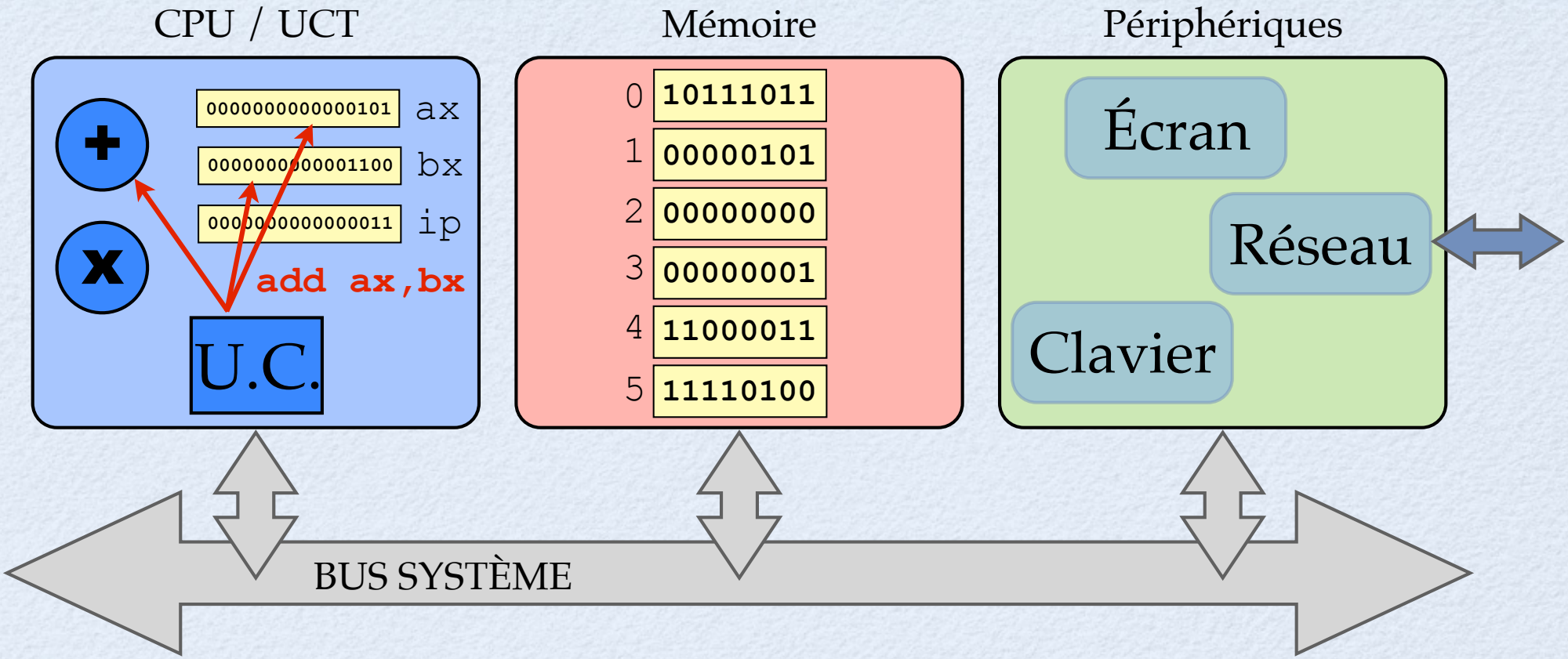

- Fonctionnement = répéter les étapes :
  1. *lecture* de l'instruction machine en mémoire
  2. *décodage* de l'instruction
  3. *exécution* de l'instruction

# Fonctionnement de l'ordinateur



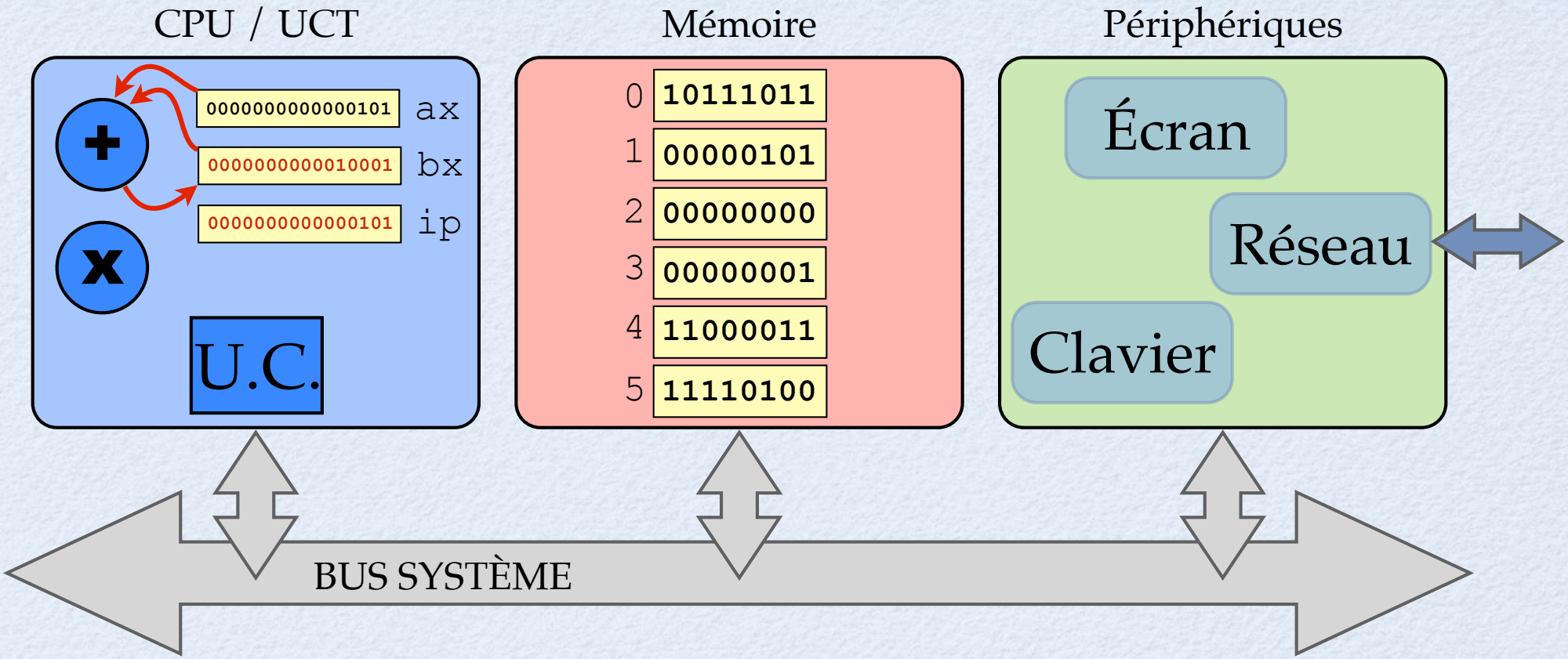

- Fonctionnement = répéter les étapes :
  1. *lecture* de l'instruction machine en mémoire
  2. *décodage* de l'instruction
  3. *exécution* de l'instruction

# Fonctionnement de l'ordinateur



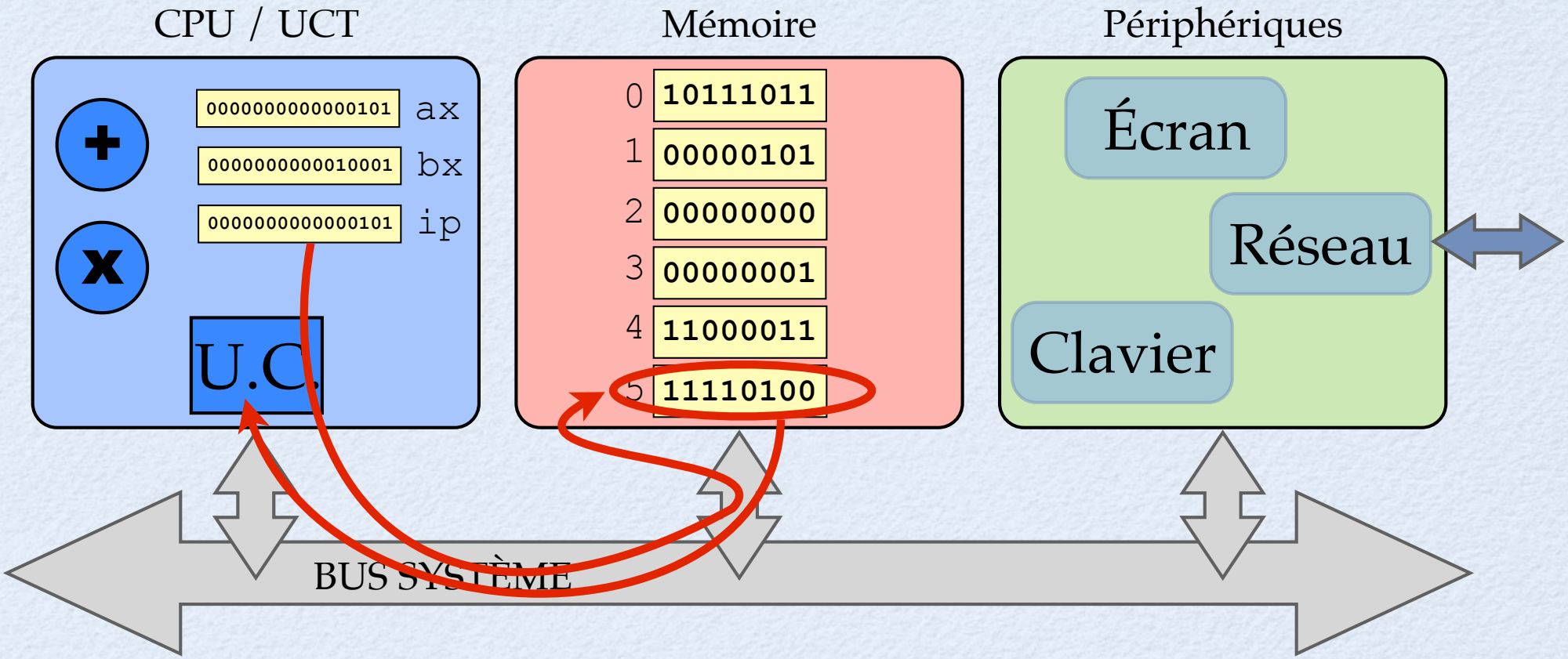

- Fonctionnement = répéter les étapes :
  1. *lecture* de l'instruction machine en mémoire
  2. *décodage* de l'instruction
  3. *exécution* de l'instruction

# Fonctionnement de l'ordinateur




- Fonctionnement = répéter les étapes :
  1. *lecture* de l'instruction machine en mémoire
  2. *décodage* de l'instruction
  3. *exécution* de l'instruction

# Fonctionnement de l'ordinateur




- Fonctionnement = répéter les étapes :
  1. *lecture* de l'instruction machine en mémoire
  2. *décodage* de l'instruction
  3. *exécution* de l'instruction

# Langages machine/assembleur

- Le **langage assembleur** est une représentation textuelle du langage machine
- Ces langages sont basés sur des instructions qui font des **opérations élémentaires :**
  - Transfer d'une unité d'information (un nombre) d'un endroit à un autre
  - Calcul élémentaire (+, −, ×, …) sur des nombres

# Langages machine/assembleur

- Exemple : ajouter 5

| code machine | code assembleur |
|---|---|
| 10111011<br>00000101<br>00000000 | `mov $5,%ax` |
| 00000001<br>11000011 | `add %ax,%bx` |

# Langages machine/assembleur

- Avantages :
  - **Contrôle total** sur l'ordinateur
  - Possibilité d'**exécution rapide**
- Désavantages :
  - **Inintelligible** pour les humains
  - Demande **beaucoup de code** pour faire peu
  - Programmeur doit se **soucier des particularités de la machine**, qui ne seront pas les mêmes d'une machine à l'autre

# Langages de bas/haut niveau

- Ces langages sont dits **de bas niveau** (d'*abstraction*) car le programmeur passe son temps à penser aux particularités de la machine plutôt que du traitement à réaliser
- Les langages **de haut niveau** facilitent le travail du programmeur en l'isolant des détails reliés à la machine et en offrant des opérations plus proches de celles requises par l'application
- Par exemple, en C : `x+5`

# Langages de haut niveau

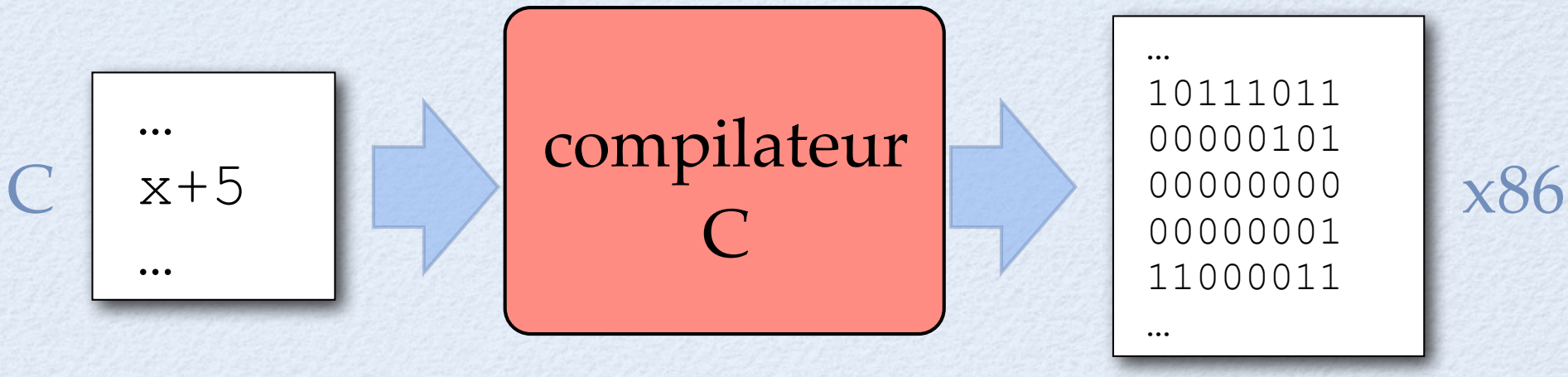
- Avantages :
  - Accélère le **codage** (l'écriture du code) et sa compréhension par d'autres programmeurs
  - Réduit le nombre d'**erreurs** de programmation
  - Donne des programmes plus **portables** (qui peuvent exécuter sur des ordinateurs variés)

# Outils de base

# Compilateurs

- **Compilateur** : programme qui traduit un programme source, en **langage source**, en un programme équivalent en **langage cible**
- Normalement le langage source est un langage de haut niveau et le langage cible est le **langage machine de l'ordinateur**
- Le compilateur s'occupe des particularités de la machine et, dans certains cas, il **optimise** le code pour une exécution rapide

# Compilateurs

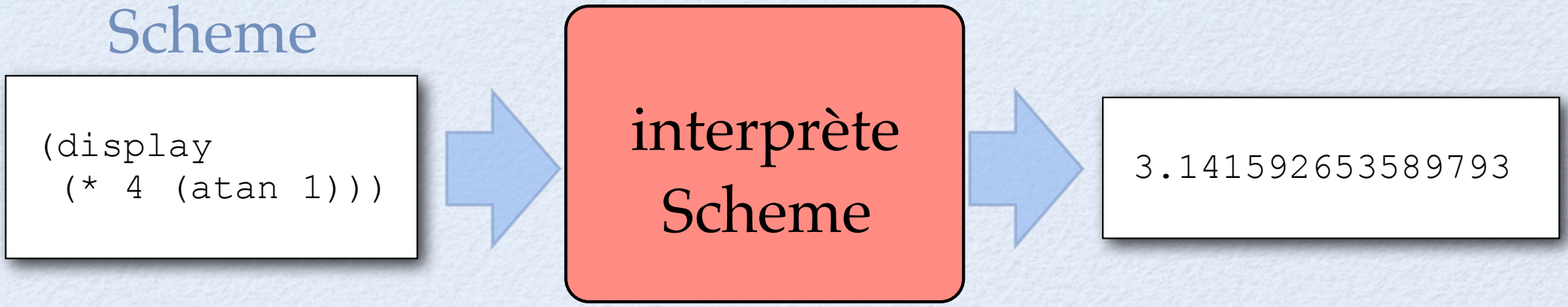



- 2 phases : le compilateur produit un programme qui peut être exécuté, plus tard, sur la *machine cible*
- On peut cibler d'autres machines (*portabilité*)
- Certains bogues de codage sont détectés par le compilateur, d'autres à l'exécution du progr.

# Interprètes

- **Interprète** : programme qui est un processeur de son *langage source*
- Normalement le langage source est un langage de *haut niveau*
- L'interprète exécute le programme *immédiatement*

# Interprètes




- Généralement :
  - Les interprètes favorisent le développement **interactif** et offrent un **cycle de débogage** plus rapide (rétroaction immédiate)
  - L'exécution du programme est **plus lente** qu'avec un compilateur

# Développement de logiciels

# Développement de logiciels

- Les activités principales dans la *vie d'un logiciel*
  - **Spécification**
  - **Conception (*Design*)**
  - **Codage (*Coding*)**
  - **Tests (*Testing*)**
  - **Mise en service**
  - **Maintenance**

# Spécification

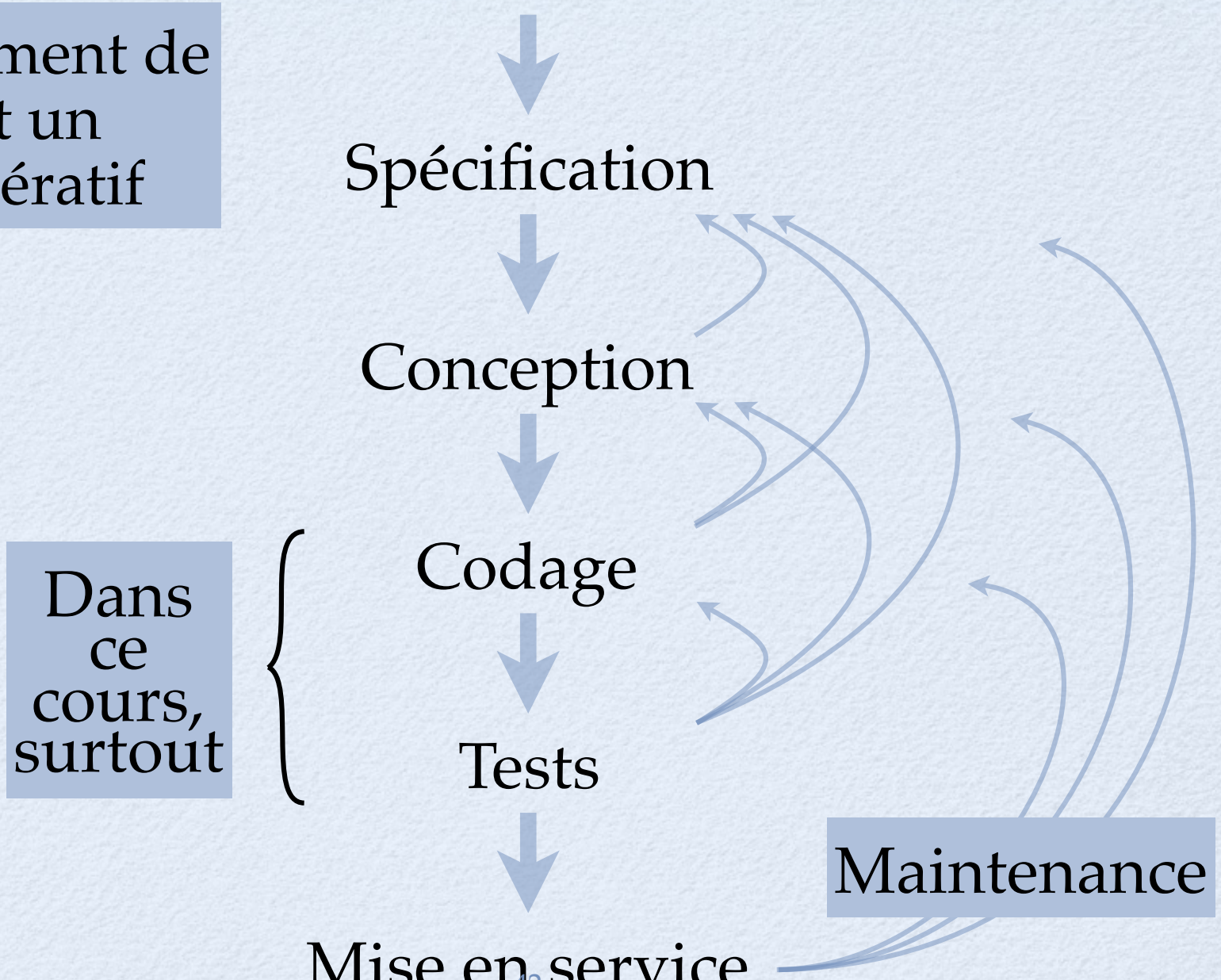
- **Spécification** : ensemble des caractéristiques voulues d'un logiciel (*cahier des charges*)
  - Description des comportements attendus
- La spécification est un **contrat** entre un *fournisseur de service* (le développeur du logiciel) et un *client* (l'utilisateur du logiciel)
- **Bogue** ("bug") : défaut dans un logiciel qui fait qu'il ne respecte pas la spécification (l'élimination des bogues = **débogage**)

# Conception, codage, etc.

- **Conception** : établissement de l'*architecture* du logiciel (structure que prendra le logiciel et sa décomposition en sous-systèmes), choix d'*algorithmes*, langages et technologies, …
- **Codage** : *écriture* des parties en un/des langage(s) de programmation spécifique(s)
- **Tests** : vérification du bon fonctionnement
- **Mise en service** : distribution/exploitation
- **Maintenance** : correction/extension

# Cycle de vie d'un logiciel

Le développement de logiciel est un processus itératif




Dans ce cours, surtout

# Analogie culinaire

- **Spécification** : tarte aux pommes pour 4 personnes
- **Conception** : 10”, croute, pommes, sucre, croute
- **Codage** : “1 – chauffer four à 250 °C, 2 – couper en morceaux 8 pommes, 3 – faire 2 croutes, …”
- **Tests** : essayer la recette puis…

  A – croute brulée –> codage/200 °C

  B – trop liquide –> conception/une seule croute

  C – on n’aime pas le goût –> spécif/prunes
- **Maintenance** : on devient diabétique, …

# Quelques critères de qualité

- Important pour l'*utilisateur* :
  - **Correct** : le programme n'a pas de bogue
  - **Performant** : il est rapide et réactif
  - **Convivial** : il est facile à utiliser et flexible
- Important pour le *développeur* :
  - **Maintenable** : il est facile à corriger et étendre
- Nous en verrons plusieurs autres…

# Quelques critères de qualité

- Par ordre d'importance pour ce cours (de façon générale) :
  - **Correct** : il est conforme à la spécification
  - **Maintenable** : il est écrit dans un style compréhensible par d'autres programmeurs
  - **Convivial et Performant** : ces critères sont secondaires dans la mesure du raisonnable, à moins que ça soit précisé dans la spécification (par ex. niveau de performance minimal)

# Quelques critères de qualité

- Dans le cours nous utiliserons des **revues de code** (“**code reviews**”) pour discuter des critères de qualité
  - **Revue de code** : examen systématique du code source d’un logiciel dans le but d’en améliorer la qualité et éliminer les bogues
- La revue de code par tous les membres d’une équipe de développement fait souvent partie intégrante des pratiques de l’équipe