

Programmation fonctionnelle et Parallélisme

Cours 1

Jean-Jacques Lévy

`jean-jacques.levy@inria.fr`

`http://jeanjacqueslevy.net/prog-fp`

Plan

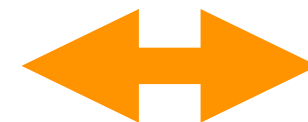
- machines informatiques
- systèmes informatiques
- langages de programmation
- étude comparative sur plusieurs exemples
- langages du cours: Ocaml et Haskell

dès maintenant: **télécharger Ocaml en** `http://www.ocaml.org`

Machines informatiques

- les premiers vrais ordinateurs en 1960: IBM 704, 7040, 360/370, ...
- le langage des machines est exprimé en **binaire**: 0 ou 1 [pour des raisons électriques]

```
mac$ od -x a.out |head -4
00000000  0162  0006  8f30  2e89  6000  0000  0060  0000
00000020  0038  0007  010b  020a  4000  0000  2000  0000
00000040  0000  0000  0140  0040  0000  0040  0000  1000
00000060  2000  1000  fffe  f3ff  0000  0000  1013  0000
mac$ od -x a.out |tail -5
01164000  0000  0028  0216  0000  3030  0040  1046  0000
01164020  0000  0023  021c  0000  1580  1000  f0c5  ffff
01164040  0000  0023  0220  0000  12e0  1000  f341  ffff
01164060  0000  0023  0228  0000  12e4  1000  f341  ffff
```



```
mac$ cat t.c
#include <stdio.h>

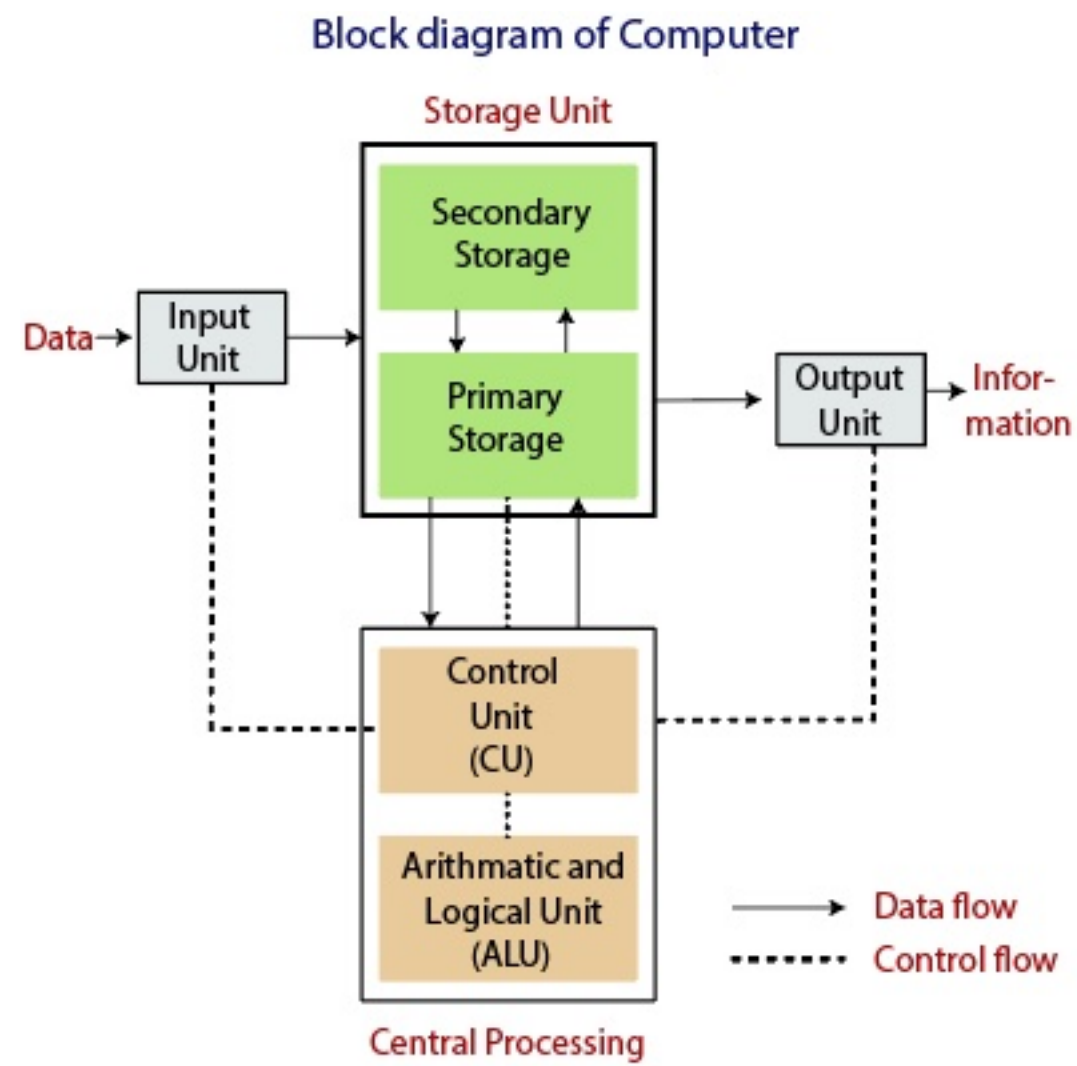
char s[100] = "voici une chaîne de caractères" ;

main()
{
    printf ("%s\n", s);
}
```

- ici 40256 octets = 322048 bits
- dans ce binaire, il y a des commandes [opérations à effectuer] et des données
- ce binaire est chargé dans la mémoire de l'ordinateur
- et l'ordinateur exécute les commandes du binaire
- chaque opération prend quelques micro-secondes

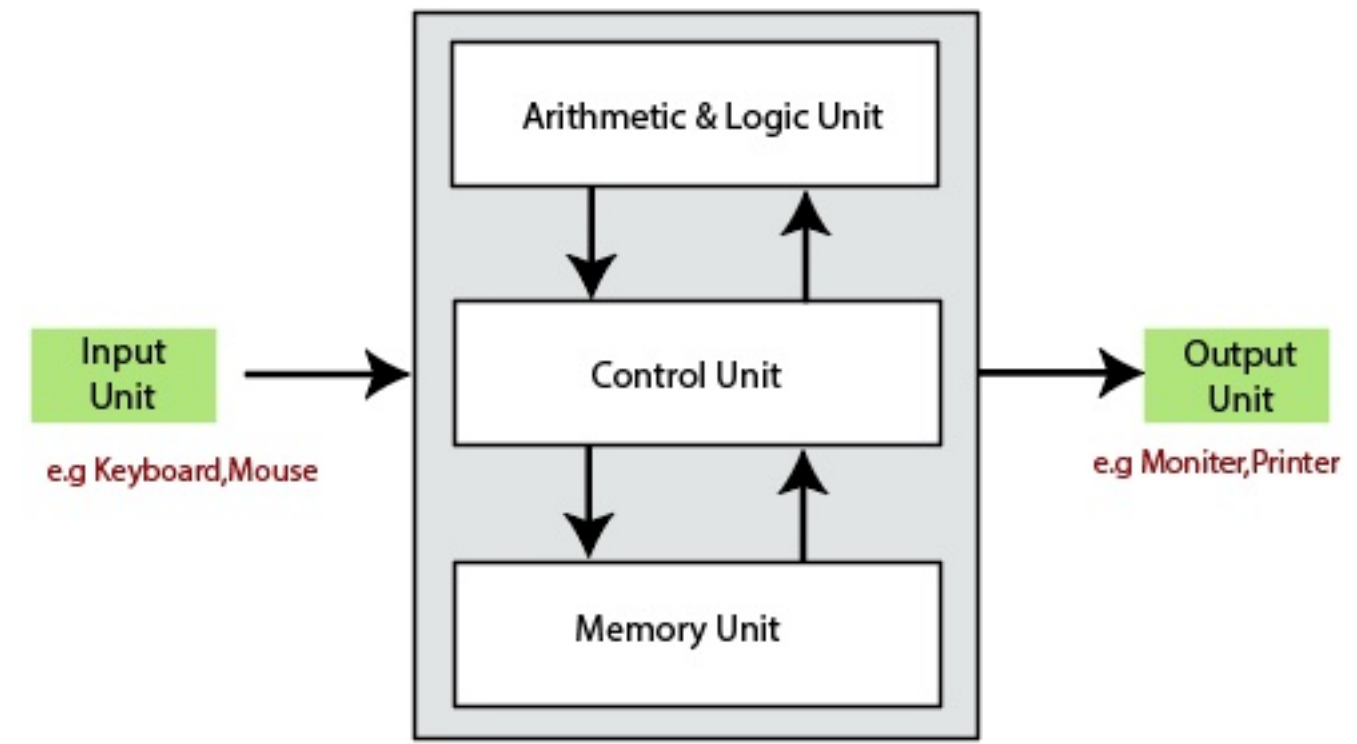
*bits = binary digit
octets = 8 bits = bytes*

Machines informatiques

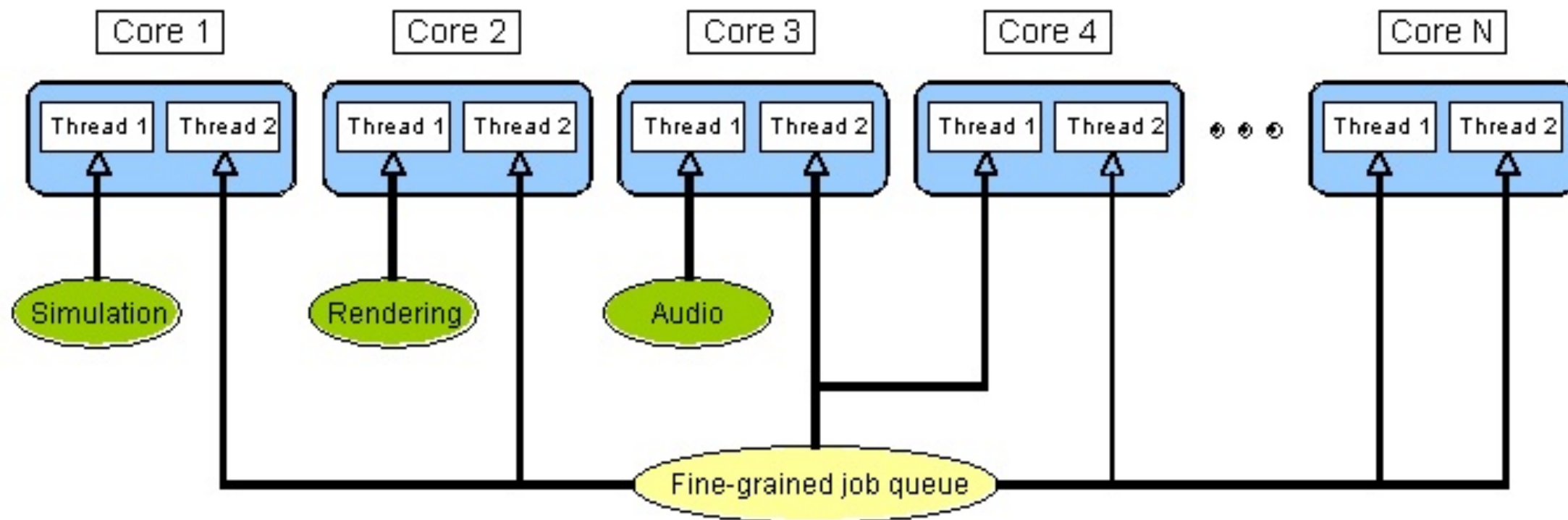


processeur

Central Processing Unit (CPU)



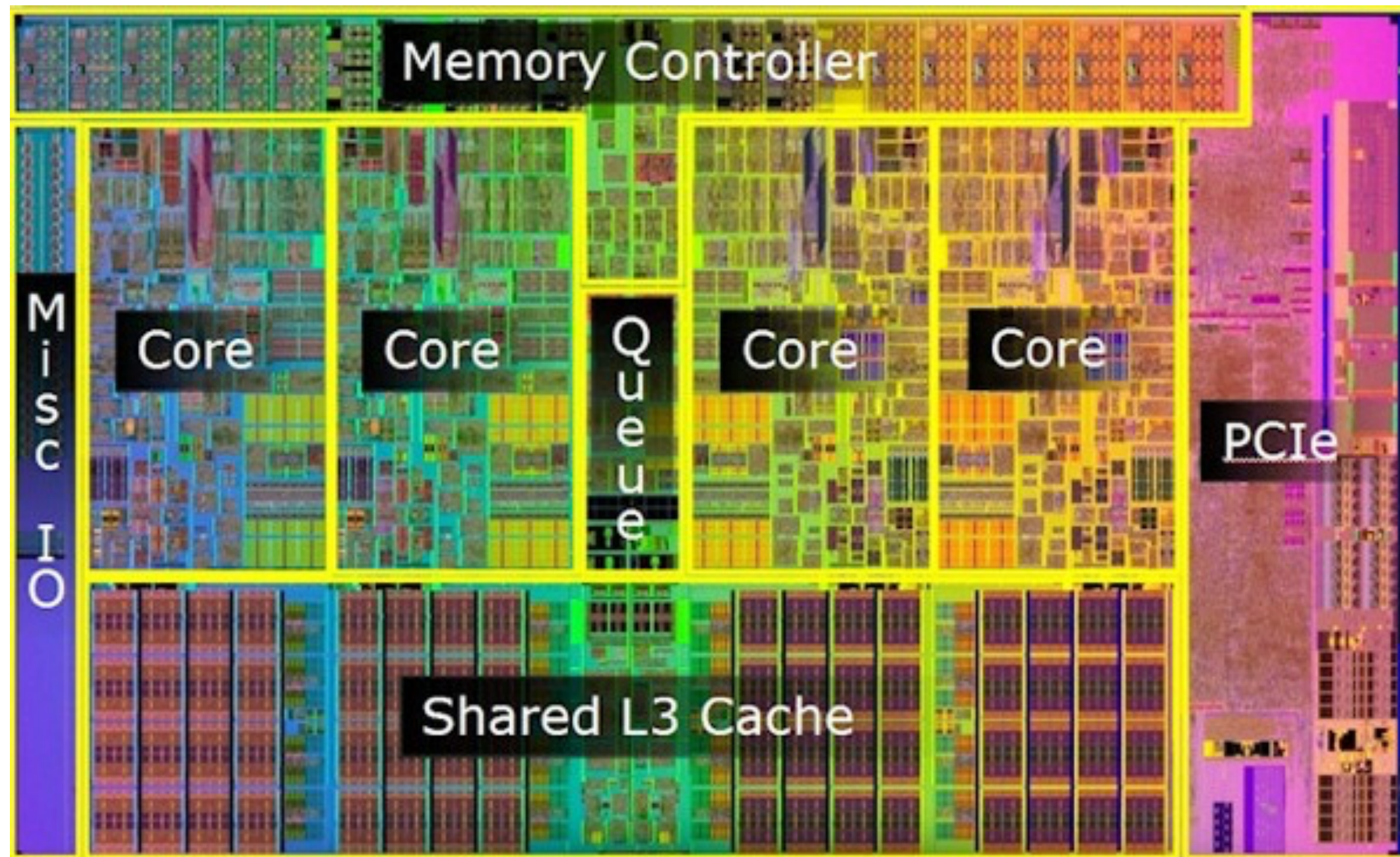
unité centrale



multicore

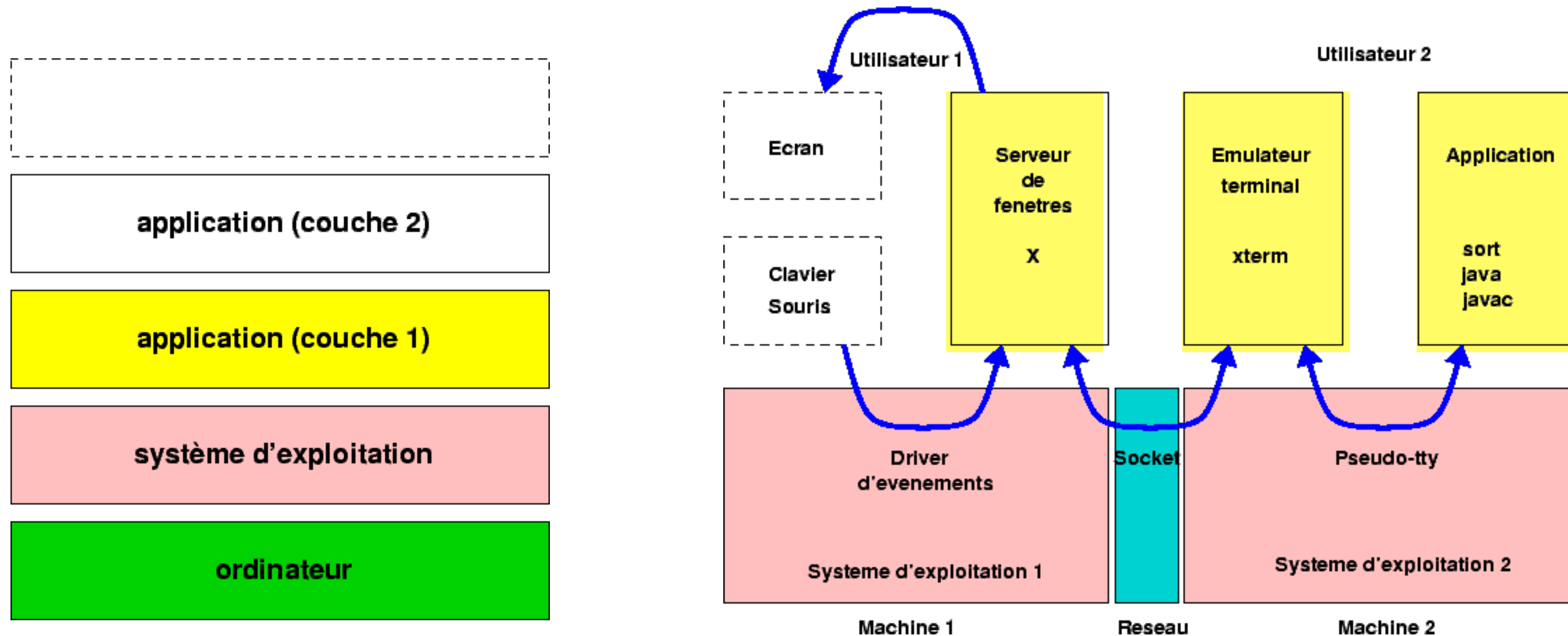
Machines informatiques

- Intel Core i5, 3.3GHz, 774m transistors, surface 296 mm²
- gravure 14 (en 2014) nm à 45nm



Systemes informatiques

- Windows, MacOS, Unix, Linux, ios, android, GFS, ...
- faire marcher l'ordinateur avec ses périphériques (écran, disques, clavier, souris, utilisateurs, réseau, ...)
- les OS sont d'une complexité extrême (~100 millions de lignes de code)



*OS = operating system
= système d'exploitation*




















- au 15ème siècle, on construisait des cathédrales
- au 20ème siècle, on a construit des systèmes informatiques

Langages de programmation et abstraction



- on écrit des programmes dans des langages plus ou moins éloignés du langage machine

Langages de programmation et buts professionnels

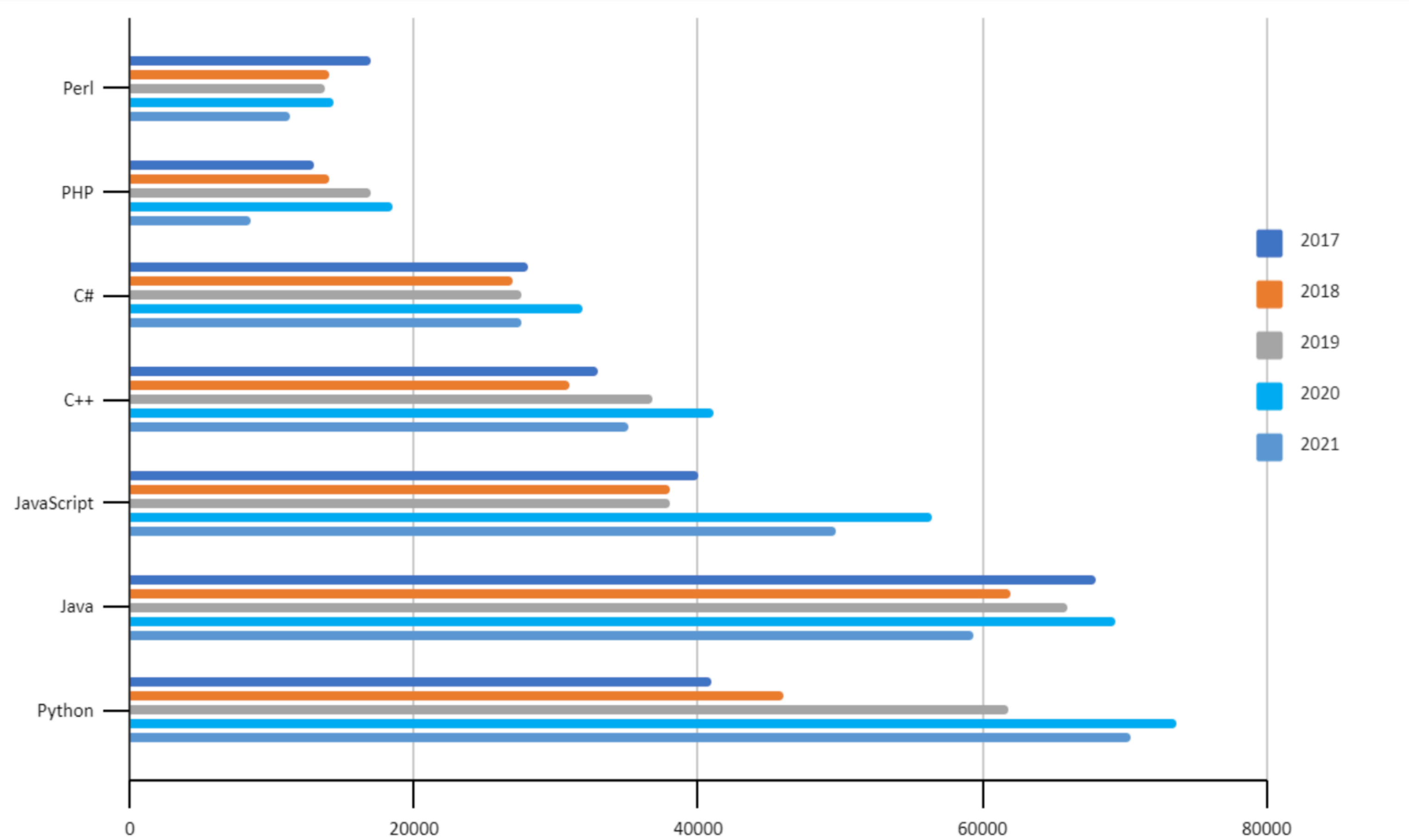
Front-end web development	Back-end web development	Mobile development
 JavaScript	 JavaScript	 Swift
 Elm	 Scala	 Java
 TypeScript	 Python	 Objective C
	 Go	 JavaScript
	 Ruby	
Game development	Desktop applications	Systems programming
 Unity	 Scala	 Go
 TypeScript	 Go	 Rust
	 Python	

Langages de programmation et buts professionnels

- **Front-end web development:** JavaScript
- **Back-end web development:** JavaScript, Java, Python, PHP, Ruby
- **Mobile development:** Swift, Java, C#
- **Game development:** C++, C#
- **Desktop applications:** Java, C++, Python
- **Systems programming:** C, Rust

Langages de programmation et histoire

The Top Programming Languages of Previous Years Compared to 2021



TIOBE Index for February 2021

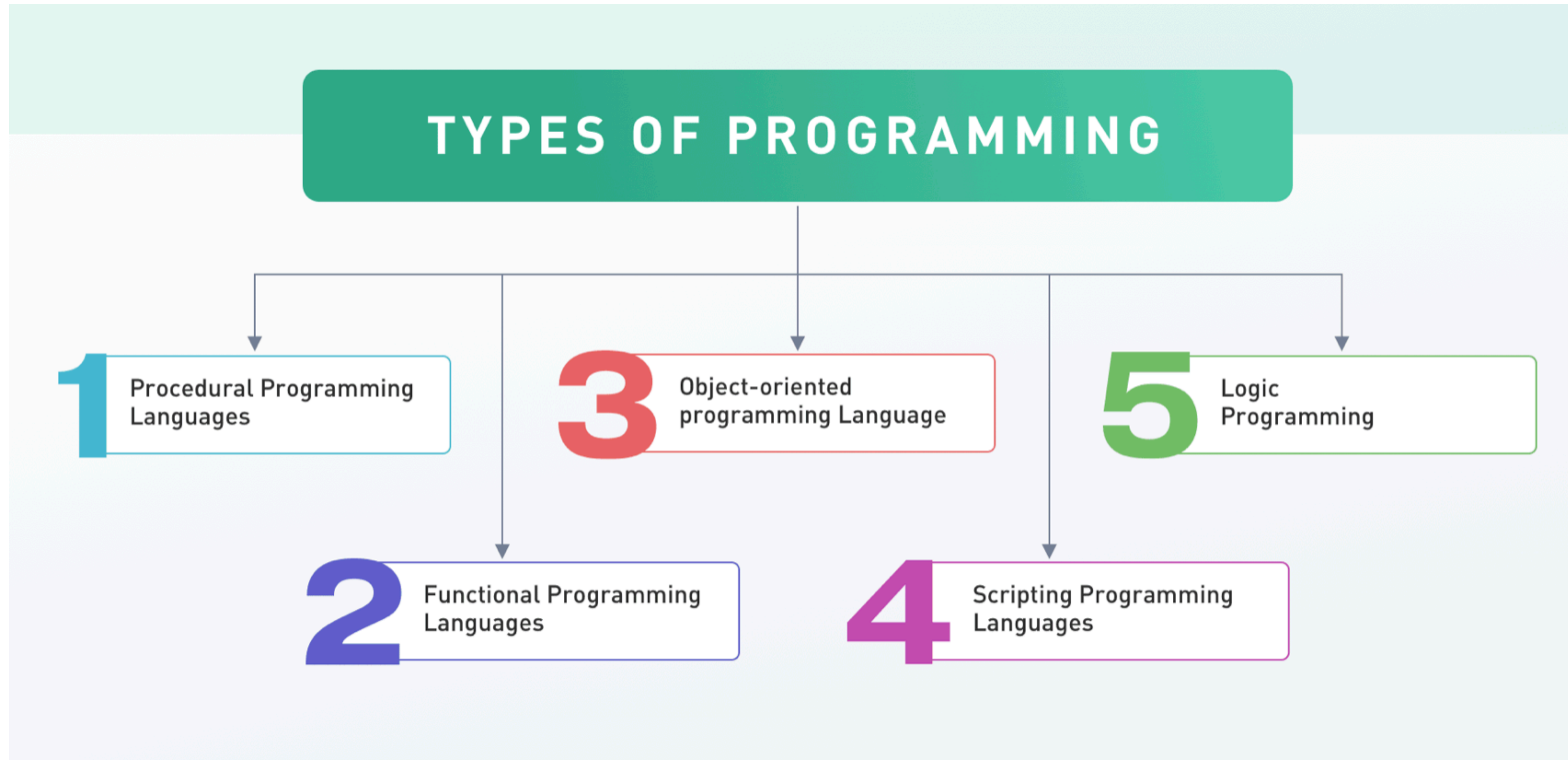
Feb 2021	Feb 2020	Change	Programming Language	Ratings	Change
1	2	▲	C	16.34%	-0.43%
2	1	▼	Java	11.29%	-6.07%
3	3		Python	10.86%	+1.52%
4	4		C++	6.88%	+0.71%
5	5		C#	4.44%	-1.48%
6	6		Visual Basic	4.33%	-1.53%
7	7		JavaScript	2.27%	+0.21%
8	8		PHP	1.75%	-0.27%
9	9		SQL	1.72%	+0.20%
10	12	▲	Assembly language	1.65%	+0.54%
11	13	▲	R	1.56%	+0.55%
12	26	▲▲	Groovy	1.50%	+1.08%
13	11	▼	Go	1.28%	+0.15%
14	15	▲	Ruby	1.23%	+0.39%
15	10	▼▼	Swift	1.13%	-0.33%
16	16		MATLAB	1.06%	+0.27%
17	18	▲	Delphi/Object Pascal	1.02%	+0.27%
18	22	▲▲	Classic Visual Basic	1.01%	+0.40%
19	19		Perl	0.93%	+0.23%
20	20		Objective-C	0.89%	+0.20%

21	Scratch	0.82%
22	SAS	0.66%
23	Fortran	0.63%
24	D	0.59%
25	COBOL	0.58%
26	Transact-SQL	0.55%
27	Prolog	0.54%
28	PL/SQL	0.52%
29	Julia	0.52%
30	Rust	0.49%
31	Ada	0.47%
32	Dart	0.42%
33	(Visual) FoxPro	0.42%
34	ABAP	0.38%
35	Lisp	0.34%
36	Scala	0.34%
37	Lua	0.34%
38	Logo	0.33%
39	Kotlin	0.32%
40	TypeScript	0.29%
41	VHDL	0.26%
42	Bash	0.25%
43	LabVIEW	0.24%
44	Haskell	0.24%
45	VBScript	0.24%
46	Ladder Logic	0.23%
47	Apex	0.23%
48	Elixir	0.22%
49	Alice	0.22%
50	PowerShell	0.21%

TIOBE Index for February 2021

Programming Language	2021	2016	2011	2006	2001	1996	1991	1986
C	1	2	2	2	1	1	1	1
Java	2	1	1	1	3	28	-	-
Python	3	5	6	7	24	15	-	-
C++	4	3	3	3	2	2	2	8
C#	5	4	5	6	9	-	-	-
JavaScript	6	7	10	9	6	30	-	-
PHP	7	6	4	4	19	-	-	-
R	8	14	39	-	-	-	-	-
SQL	9	-	-	-	-	-	-	-
Go	10	57	16	-	-	-	-	-
Perl	14	9	7	5	4	3	-	-
Lisp	29	24	13	13	17	5	3	2
Ada	33	23	21	15	15	6	9	3

Types de langages de programmation



Les langages du cours

- Python [Guido van Rossum, 1995]
- C [Dennis Ritchie, 1978]
- Ocaml [Xavier Leroy, 1985]
- Haskell [Simon Peyton-Jones, 1990]
- Java [James Gosling, 1995]
- C++ [Bjarne Stroustrup, 1984]
- on télécharge Python 3 en <http://www.python.org>
- C et C++ sont en natifs sur pratiquement tous les systèmes
- on télécharge Ocaml en <http://www.ocaml.org> et Haskell en <http://www.haskell.org>
- on télécharge Java en <http://www.java.com>

Les langages du cours

- Python (fil conducteur)
 - C
- } **langages impératifs**

- Ocaml
 - Haskell
- } **langages fonctionnels**

- Java
 - C++
- } **langages orienté-objets**

et plus brièvement

- langages de script (php, perl)
- programmation logique (prolog, sql)
- programmation concurrente (rust, modula3, ada)

Exemples de programmes

- Python (fil conducteur)
 - C
- } langages impératifs

- Ocaml
 - Haskell
- } langages fonctionnels

- Java
 - C++
- } langages orienté-objets

- copie de tableaux
 - tri des éléments d'un tableau
- } programmation impérative

- arbre binaire de recherche
 - file d'attente
- } structures de données dynamiques

- interface graphique
 - modularité
- } réutilisabilité

Débuter en Ocaml

- utiliser un système intégré (Visual Studio ou autre)
- ou utiliser une simple fenêtre terminal [le plus simple]
- avec un éditeur de texte (Emacs, VI, TextEdit, ..)

exemple avec Ocaml

- sur la fenêtre terminal, on tape:

```
mac$ ocaml
      OCaml version 4.11.0
```

```
#
```

- et on peut fonctionner en mode calculette:

```
# 23 + 42 ;;
- : int = 65
```

← int (type entier)

```
# 438 * 234 ;;
- : int = 102492
```

```
# (438 * 234) + 35 ;;
- : int = 102527
```

```
# 5.0 ;;
- : float = 5.
```

← float (type réel flottant)

```
# 5 ;;
- : int = 5
```

Débuter en Ocaml

- mode calculette - *toplevel* interactif

```
# let x = 45 ;;  
val x : int = 45
```

← le type du résultat est imprimé

```
# 3 * x + 2 ;;  
- : int = 137
```

- avec des chaînes de caractères

```
# let s = "Bonjour les amis" ;;  
val s : string = "Bonjour les amis »
```

```
# s ^ " et la famille" ;;  
- : string = "Bonjour les amis et la famille"
```

← ^ est l'opérateur de concaténation

```
# s.[0] ;;  
- : char = 'B'
```

```
# s.[1] ;;  
- : char = 'o'
```

```
# s.[2] ;;  
- : char = 'n'
```

Débuter en Ocaml

- définir des fonctions

```
# let double x = x + x ;;  
val double : int -> int = <fun>
```

```
# double 3;;  
- : int = 6
```

```
# let cat s1 s2 = s1 ^ s2 ;;  
val cat : string -> string -> string = <fun>
```

```
# cat "Bonjour " "les amis !" ;;  
- : string = "Bonjour les amis !"
```

```
# let hello = cat "Bonjour ";;  
val hello : string -> string = <fun>
```

```
# hello "Jean-Jacques" ;;  
- : string = "Bonjour Jean-Jacques"
```

```
# hello "les amis!" ;;  
- : string = "Bonjour les amis!"
```

← le résultat est une fonction

← pas aussi simple en Python !

Premiers programmes

Le calcul de la date de Pâques

Premier dimanche après la première lune qui suit l'équinoxe de printemps.

Soit Y l'année, dont on cherche la date de Pâques.

1. **Golden number** $G = (Y \bmod 19) + 1$
2. **Century** $C = \lfloor Y/100 \rfloor + 1$
3. **Corrections** $X = \lfloor 3C/4 \rfloor - 12, Z = \lfloor (8C + 5)/25 \rfloor - 5$
4. **Find Sunday** $D = \lfloor 5Y/4 \rfloor - X - 10$
5. **Epact** $E = (11G + 20 + Z - X) \bmod 30$.
Si $E=25$ et $G>11$, ou si $E=24$, alors $E \leftarrow E+1$
6. **Find full moon** $N = 44 - E$. Si $N < 21$, alors $N \leftarrow N + 30$
7. **Advance to Sunday** $N \leftarrow N + 7 - ((D + N) \bmod 7)$
8. **Getmonth** Si $N > 31$, la date est le $(N - 31)$ AVRIL
Sinon, la date est le N MARS.

```
# let paques y =
  let g = (y mod 19) + 1 in
  let c = y / 100 + 1 in
  let x = 3 * c / 4 - 12 in
  let z = (8 * c + 5) / 25 - 5 in
  let d = 5 * y / 4 - x - 10 in
  let e = (11 * g + 20 + z - x) mod 30 in
  let e' = if e = 25 && g > 11 || e = 24 then
    e + 1
  else e in
  let n = 44 - e' in
  let n' = if n < 21 then
    n + 30
  else n in
  let j = n + 7 - ((d + n) mod 7) in
  if j > 31 then
    Printf.printf "%d %s\n" (j - 31) "avril"
  else
    Printf.printf "%d %s\n" j "mars" ;;
;;

# paques 2021 ;;
4 avril
# paques 2020 ;;
12 avril
# paques 2024 ;;
31 mars
```

modulo

/ division entière

impression
formatée

http://fr.wikipedia.org/wiki/Calcul_de_la_date_de_Pâques

Premiers programmes

- impression

```
# let cat_print0 s1 s2 =  
  print_string (s1 ^ s2) ;  
  print_newline () ;;
```

← ^ est la concaténation

```
# cat_print0 "Hello, " "World!" ;;  
Hello, World!  
- : unit = ()
```

- impression formatée

```
# let cat_print1 s n x =  
  Printf.printf "%s vaut %d ou %.2f\n" s n x ;;  
val cat_print1 : string -> int -> float -> unit = <fun>
```

← %s, %d, %f formats d'impression (comme en C)

```
# cat_print1 "Le résultat" 32 28.5 ;;  
Le résultat vaut 32 ou 28.50  
- : unit = ()
```

Premiers programmes

- constantes littérales: nombres entiers `int`, réels flottants `float`, chaînes `string`, booléens `bool`

```
# let a = 5 ;;  
val a : int = 5
```


```
# let b = 2.0 ;;  
val b : float = 2.
```

```
# let s = "This is a string" ;;  
val s : string = "This is a string"
```


```
# let x = true ;;  
val x : bool = true
```

- les opérateurs sur les entiers et sur les flottants sont différents

```
# let add_entiers x y = x + y ;;  
val add_entiers : int -> int -> int = <fun>
```



```
# let add_flottants x y = x +. y ;;  
val add_flottants : float -> float -> float = <fun>
```



Premiers programmes

- quelques programmes sur des données scalaires

```
# let surface r = 4. *. Float.pi *. r *. r ;;  
val surface : float -> float = <fun>
```

```
# surface 9. ;;  
- : float = 1017.87601976309293
```

- calculer la surface d'un carré ou triangle

- calculer le volume d'un cube ou sphère

Parallélisme et concurrence

- accès concurrents aux variables partagées
- exemple de la réservation de places

Ressources partagées

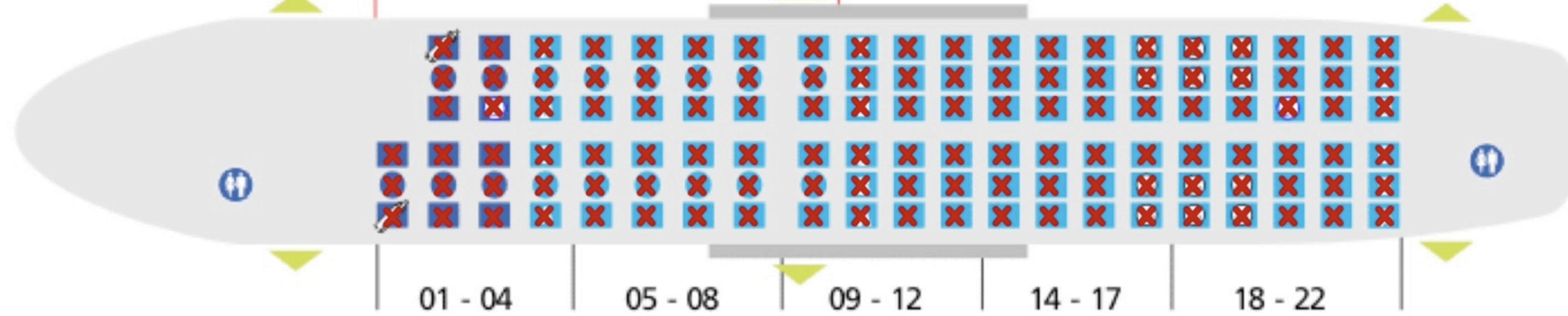
Airbus A 318 "Europe"
118 passagers

L'ESPACE AFFAIRES

Tempo



Cabine l'Espace Affaires modulable en fonction du nombre de passagers



- Sorties & Sorties de secours
- Emplacement des ailes
- Sièges neutralisés

- Toilettes
- Berceaux

- Sièges pour enfants non accompagnés
- Sièges pour passagers devant être portés de/vers leur siège

COPYRIGHT AIR FRANCE - REPRODUCTION INTERDITE



Bob

y-a-t'il une place libre?

"oui"

je la prends



Alice

y-a-t'il une place libre?

"oui"

je la prends

HERCHE IMMUN



INRIA
MICROSOFT RESEARCH

Prochain cours

- un tutorial Ocaml : <http://ocaml.org/docs>
- la programmation fonctionnelle simplifie le parallélisme