Les langages de programmation

Gérard Berry

Gerard.Berry@college-de-france.fr

Xavier Leroy

Xavier.Leroy@inria.fr

Chaire d'innovation technologique Liliane Bettencourt

Collège de France

8 février 2008

Le logiciel

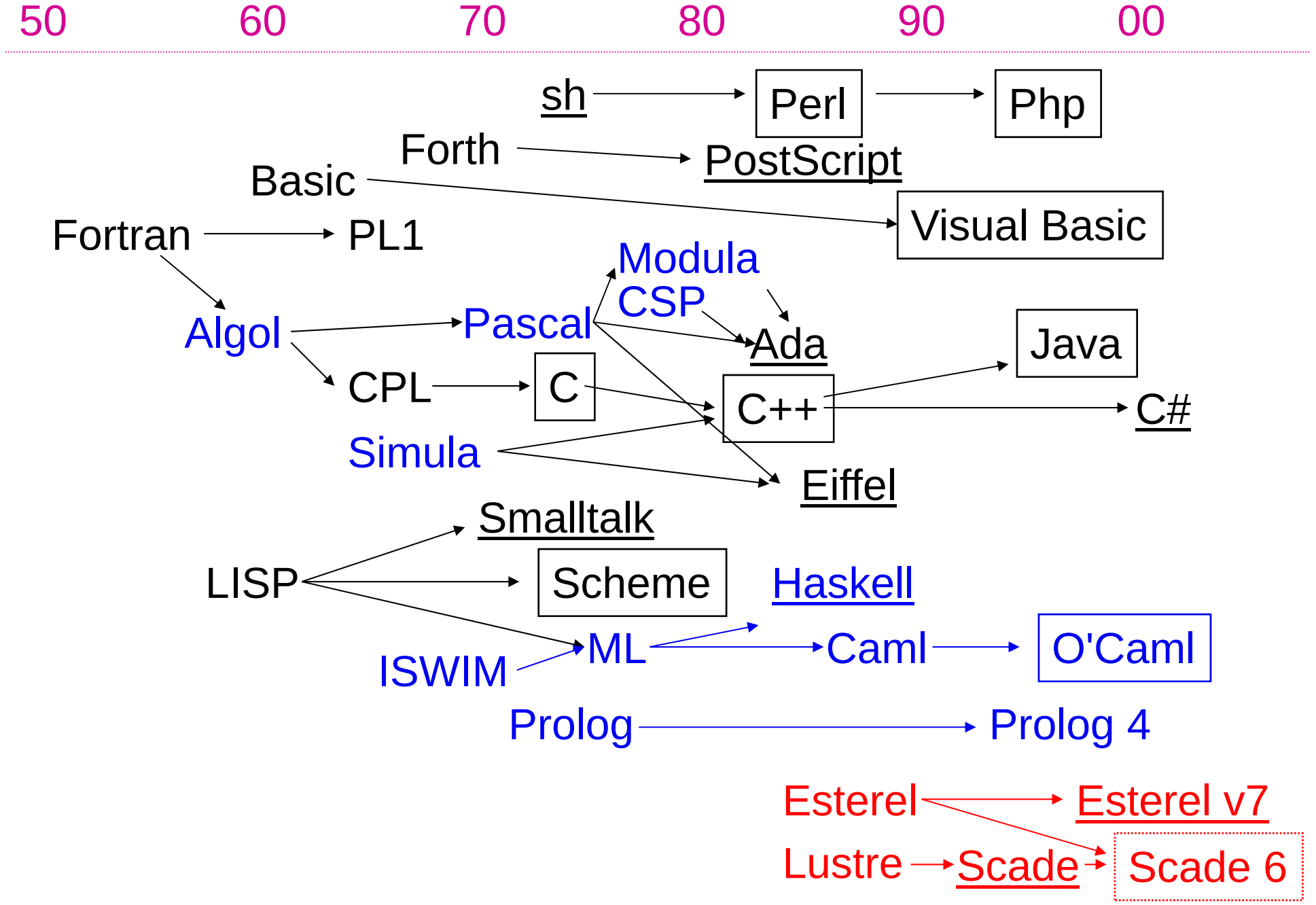
Un circuit ne fait que des choses très simples
Il en fait des milliards par seconde, et sans erreur
Le logiciel : spécifier quoi faire, dans tous les détails

- Très long texte dans un langage très technique
- Circuits peu variés, logiciels très variés
- Circuits très rigides, logiciels très souples
- Mais de plus en plus gros (millions de lignes)
- Et pas de loi de Moore....

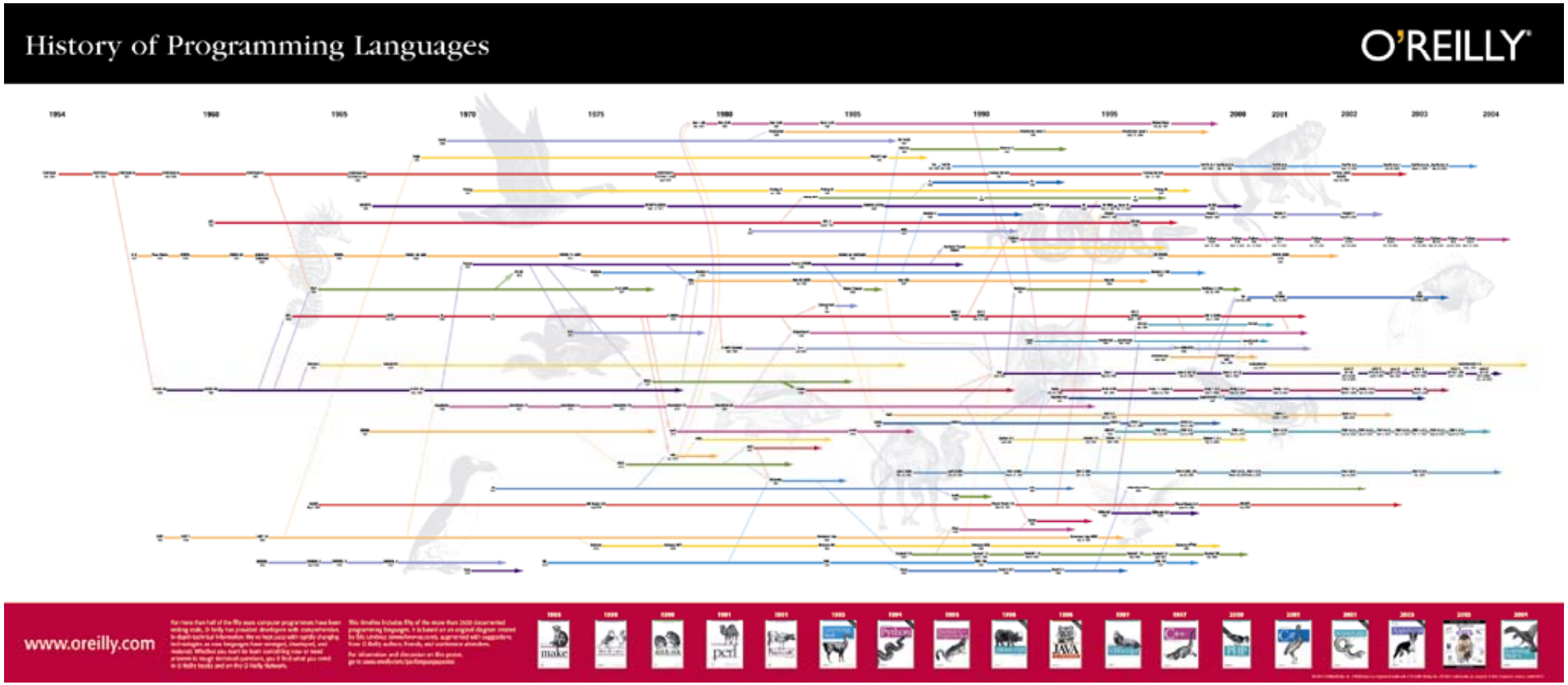
Utiliser la machine pour programmer plus sûr !

Les fonctions du langage

- L'outil d'écriture des programmes
  - par des hommes ou des programmes
  - pour la machine ou les autres hommes (cf an 2000)
  - à différents niveaux d'abstraction
- Le support d'interaction avec la pensée
  - les langages induisent des styles distincts
  - impératif, fonctionnel, logique, temporel, etc...
- Le support de guerres de religions
  - rares sont ceux qui aiment deux styles...
  - et ceux qui font abstraction du NIH (Not Invented Here)


50
60
70
80
90
00
sh
Perl
Php
Forth
PostScript
Basic
Visual Basic
Fortran
PL1
Modula
CSP
Algol
Pascal
Ada
Java
CPL
C
C++
C#
Simula
Eiffel
Smalltalk
LISP
Scheme
Haskell
ML
Caml
O'Caml
ISWIM
Prolog
Prolog 4
Esterel
Esterel v7
Lustre
Scade
Scade 6

The Programming Language Poster



http://www.oreilly.com/news/graphics/prog_lang_poster.pdf

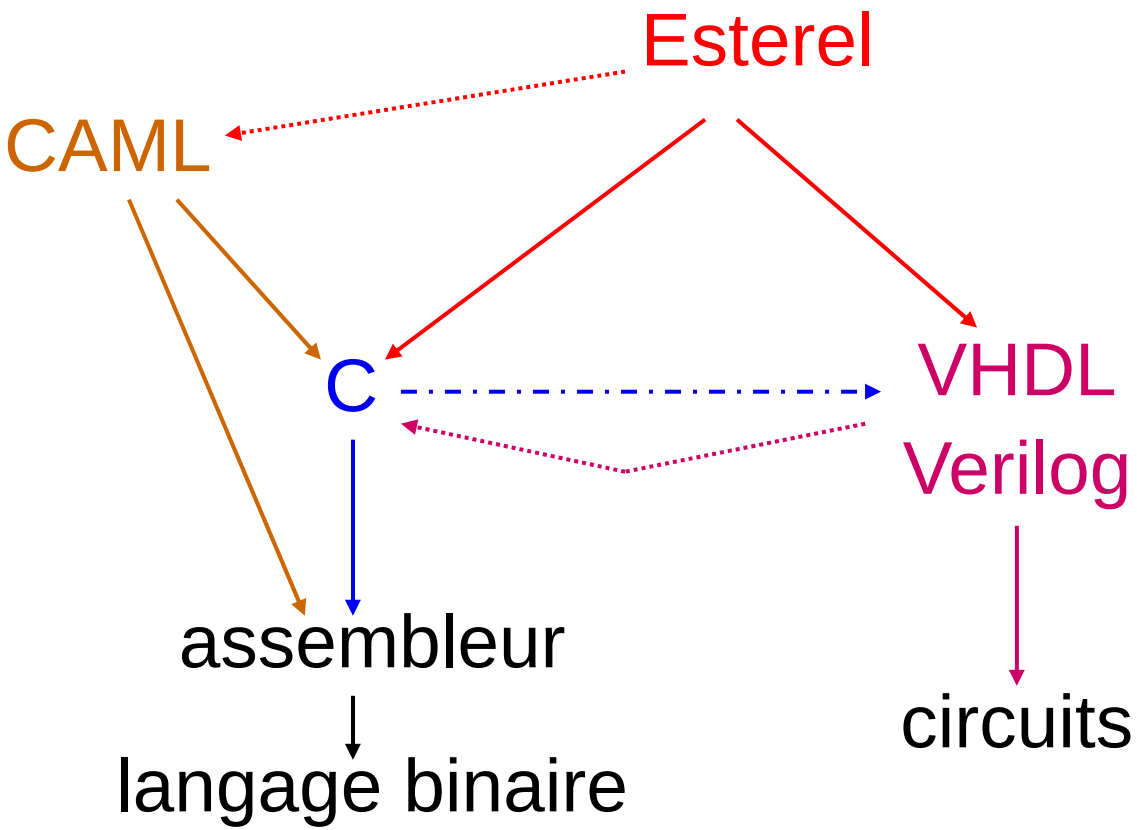
Pourquoi tant de langages ?

- Beaucoup d'aspects à traiter ensemble
  - données, actions = programming in the small
  - architecture = programming in the large
  - comment réduire et détecter les bugs
- Beaucoup d'innovations successives
  - fonctionnel, polymorphisme, modules, objets, parallélisme,...
- Enormément de compromis possibles
  - plusieurs grandes classes et beaucoup de dialectes
- Mais, heureusement, des théories solides
  - automates, grammaires, lambda-calcul, logiques

Les composants d'un langage

- Des principes de calcul et d'architecture
  - calcul : impératif, fonctionnel, logique, temporel, etc.
  - architecture : modularité, héritage, polymorphisme, etc
- Une syntaxe et un style syntaxique
  - détermine les programmes bien construits
- Un systèmes de types
  - évite d'additionner des choux et des carottes
- Une sémantique plus ou moins formelle
  - définit le sens des programmes
- Un outillage
  - compilateurs, débogueurs, manuels, exemples, librairies
  - interfaces avec d'autres langages ou systèmes
- Une communauté d'utilisateurs

Les compilateurs



Une exigence fondamentale : la portabilité

Style piquant : C, C++, Java,...

```
int fact (int n) {
   int r = 1, i;
   for (i = 2; i <= n; i++) {
      r = r*i;
   }
   return r;
}
```

Style rond : Lisp, Scheme

```
(de fact (n)
   (if (= n 1)
      1
       (* n (fact (- n 1)))))
```

```
(de map (f l)
   (if (null l)
       nil
       (cons (f (car l)) (map f (cdr l)))))
```

Style matheux : ML, Haskell, ...

```
let rec fact n =
  if n=1 then 1 else n* fact (n-1)
in fact 4;;
- : int = 24
```

```
let rec map f list =
  match list with
     [] -> []
  |  head :: tail -> (f head) :: (map f tail);;
val map : ('a -> 'b) -> 'a list -> 'b list = <fun>
```

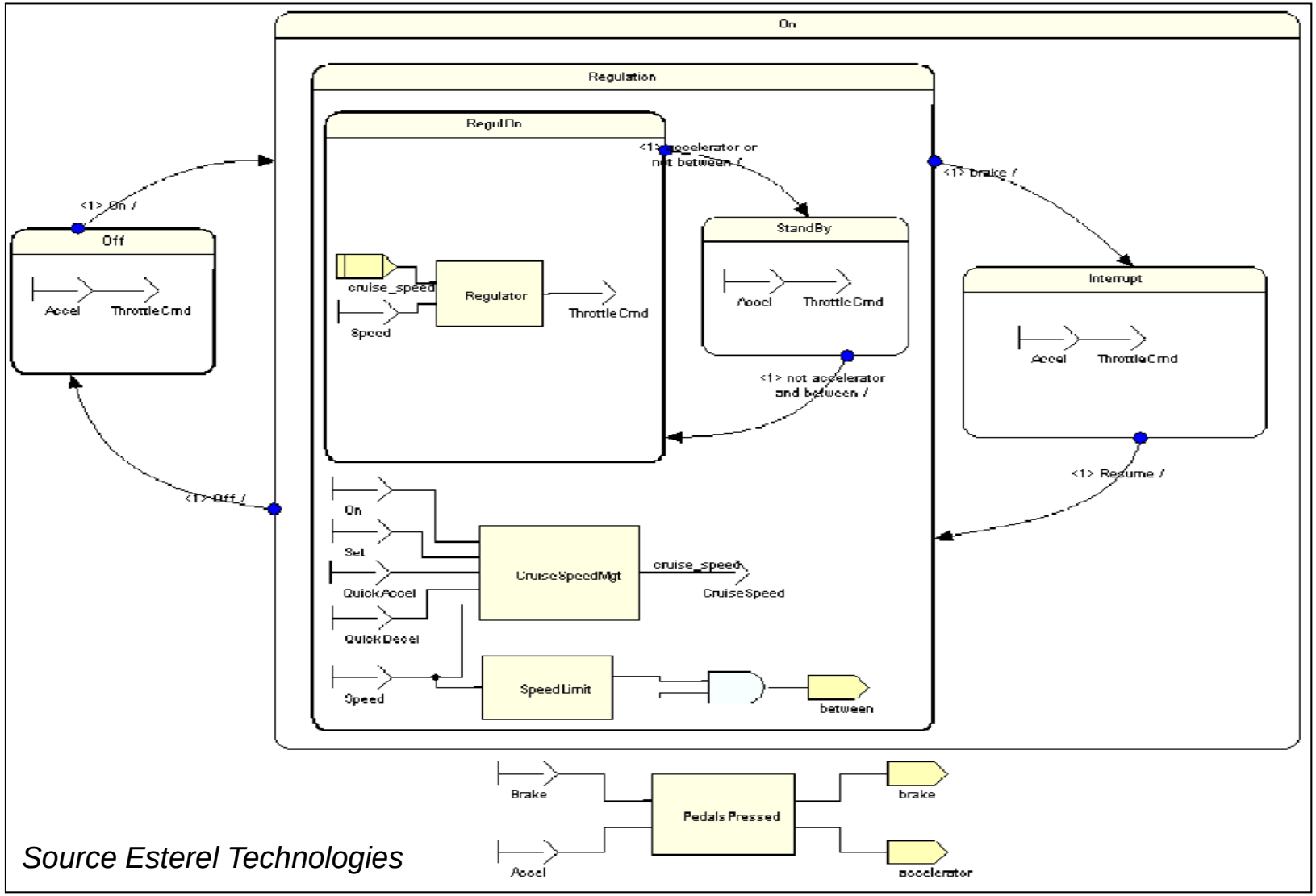
Style logique : Prolog

```
Pere (Gerard, Antoine).
Pere (Robert, Gerard).
Pere (x,y), Pere (y,z) :- GrandPere (x,z).

?- GrandPere (GP, Antoine).

> GP = Robert
```

Style verbal : Ada, Esterel,...

```
abort
   run SLOWLY
when 50 Meter;
abort
   every Step do
      emit Jump || emit Breathe
   end every
when 15 Second
```

Style graphique: Statecharts, Scade



Source Esterel Technologies

Mots et phrases

Identificateurs : x, fact, toto, foo, WikiPedia, ...
nombres : 1, 3.14, 6.023 E 23
mot-clefs : int, for, while, fun, let, rec, emit, ..

```
int i, j ;
int fact (int n) { ... }
let rec fact n = ...
```

déclarations

```
x := 1 ;
for (i=0 ; i<N ; i++) { ... }
return 3*x + fact (i) ;
```

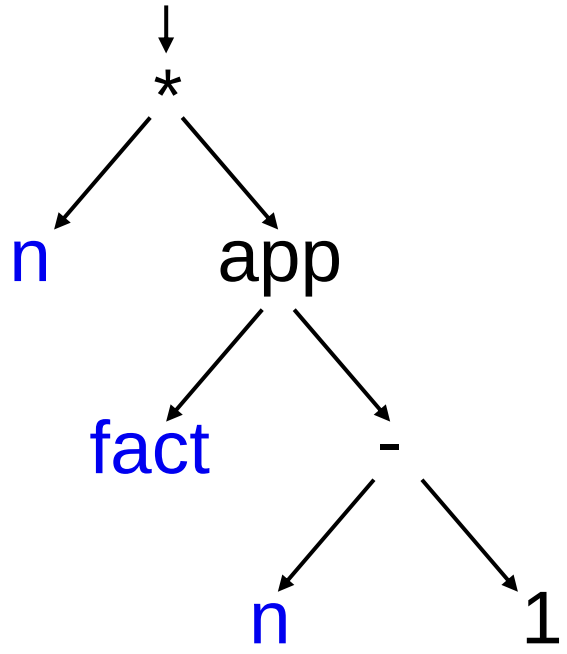
instructions

expressions

La vraie syntaxe : un arbre

C, CAML : n * fact (n-1)

Lisp, Scheme : (* n (fact (- n 1))



syntaxe concrète

syntaxe abstraite

Théorie des automates et grammaires formelles
Générateurs d'analyseurs de syntaxe

Grammaire formelle (BNF)

```
Expression ::= nombre
             | ident
             | Expression op Expression
             | '(' Expression ')'
             | ident '(' Liste_Expression ')';
Liste_Expression : = Expression
                   | Liste_Expression ',' Expression;
```

Et désambiguation par priorités :

a + b / c $\rightarrow$ a + (b / c) et pas (a + b) / c

La grammaire elle-même est écrite dans un langage de programmation!

Le modèle de calcul (1)

- Impératif ou fonctionnel ?
  - impératif = efficace
  - fonctionnel = élégant et sûr
- Général ou spécifique ?
  - général = puissant mais sauvage
  - spécifique = contrôlé mais limité
- Typé ou non typé ?
  - typé = bien plus sûr

Modèle impératif

```
int fact (int n) {
   int r = 1, i;
   for (i = 2; i <= n; i++) {
      r = r*i;
   }
   return r;
}
```

fact(4) : n ← 4; r ← 1;
i ← 2; r ← 1*2 = 2;
i ← 3; r ← 2*3 = 6;
i ← 4; r ← 6*4 = 24;
i ← 5;
return 24;

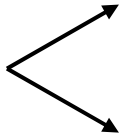
Modèle fonctionnel

```
let rec fact n =
  if n=1 then 1 else n* fact (n-1)
in fact 4 ;;
```

fact 4 = 4 * fact(3)
= 4 * (3 * fact(2))
= 4 * (3 * (2 * fact(1)))
= 4 * (3 * (2 * 1))
= 24

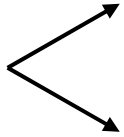
Proche de la définition mathématique

Le typage

1 + "bonjour" 

Lisp : erreur à l'éxécution

C, CAML : erreur à la compilation

3.14 + 1 

C : 4.14, car 1 "promu" en 1.0

CAML : erreur de type

+ : int, int → int

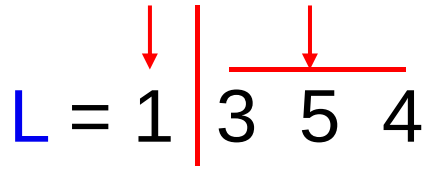
+ : float, float → float

C, CAML

C, surcharge de +

map : une fonctionnelle

head :: tail



map fact L = fact (1) fact (3) fact (5) fact (4)

= 1 6 120 24

```
let rec map f list =
  match list with
     [] -> []
  |  head :: tail -> (f head) :: (map f tail)
```

map : (int -> int) -> int list -> int list

Extension polymorphe (Milner)

```
let rec map f list =
  match list with
     [] -> []
  |  head :: tail -> (f head) :: (map f tail);;
```

val map : ('a -> 'b) -> 'a list -> 'b list = <fun>

```
let rec map f list =
  match list with
     [] -> []
  |  head :: tail -> (map f tail) :: (f head);;
```

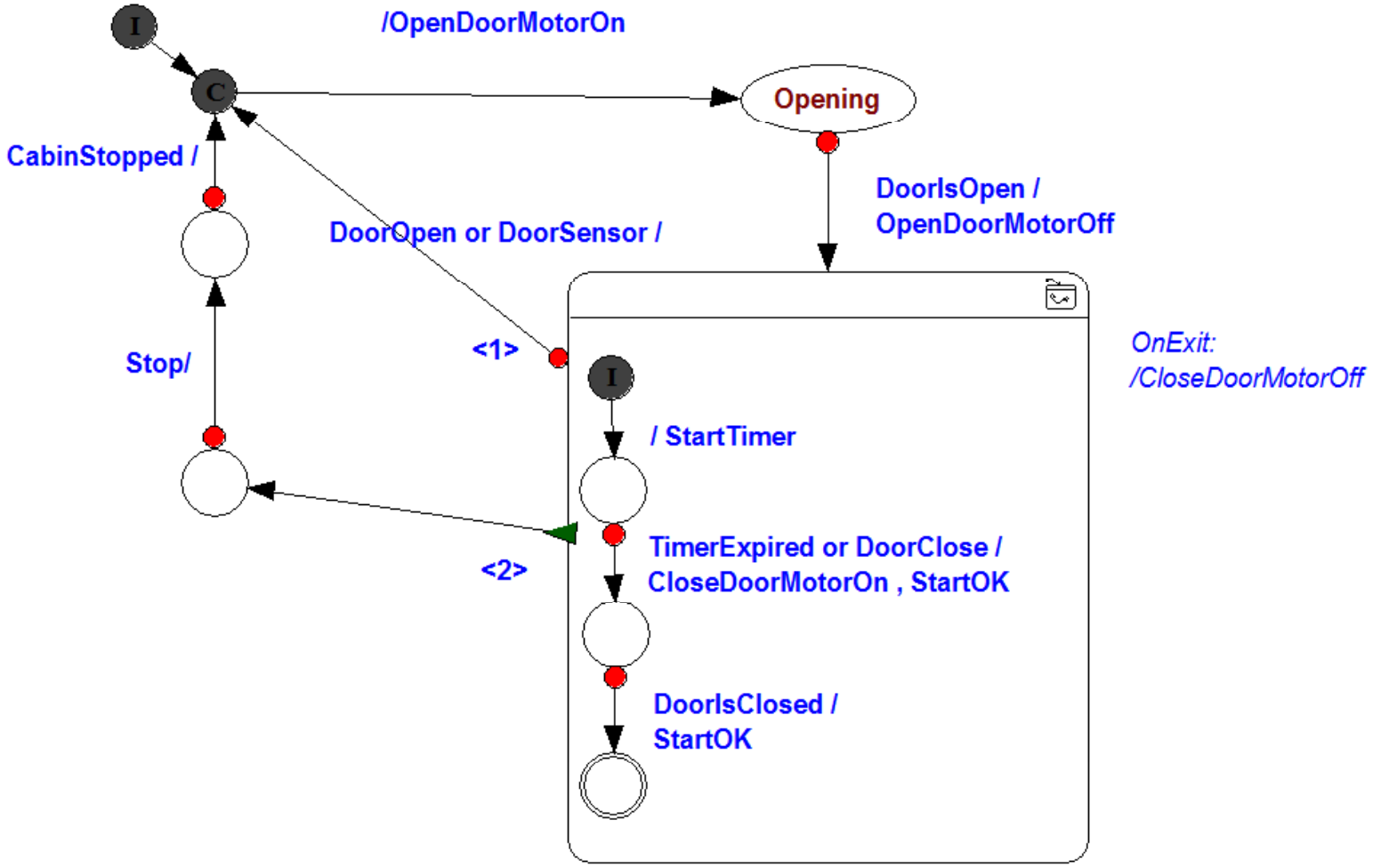
type error !

Héritage: Smalltalk, C++, Eiffel, Java, O'CAML, Esterel, ...

```
class Point_2D {
   int x;
   int y;
};
class Point_3D : Point_2D {
   // héritage de x et y
   int z;
};
```

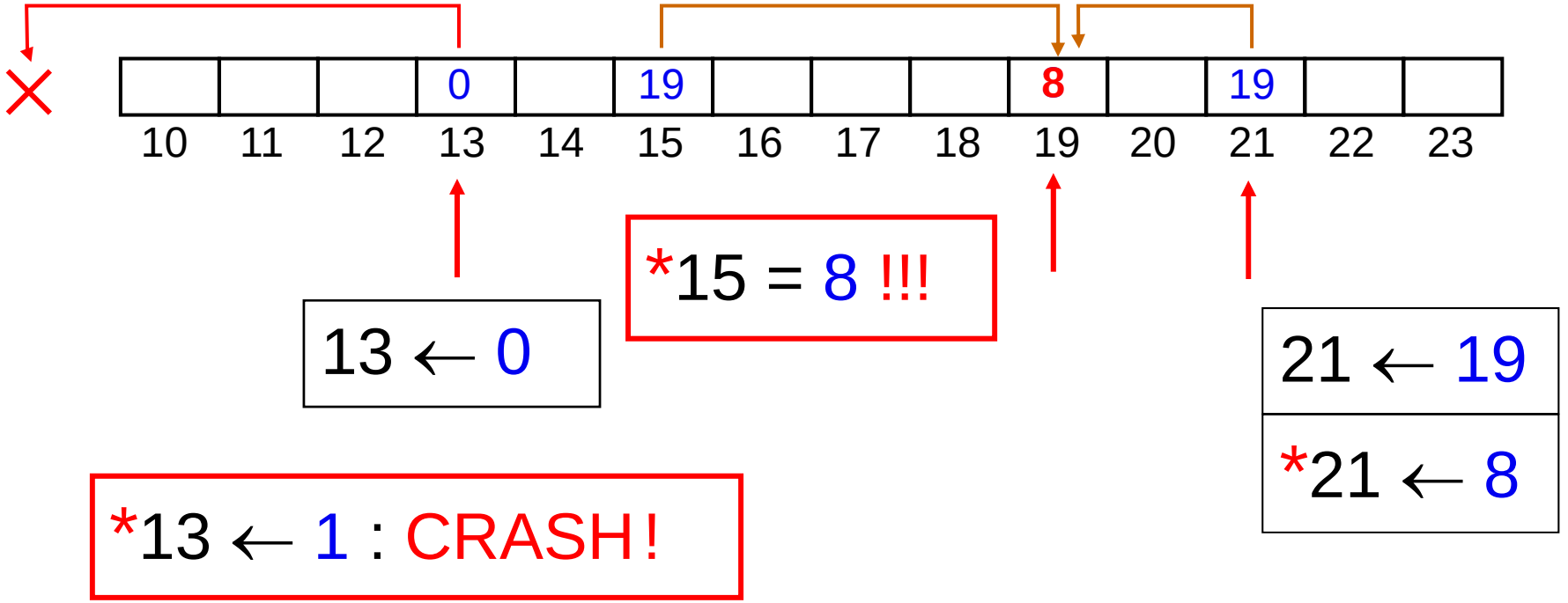
Pas facile à mélanger avec le polymorphisme

Un objet domestiqué : l'automate



Sur les automates, on sait tout calculer

Un objet sauvage: le pointeur



Gestion mémoire: Construction et destruction des objets

- Manuelle (C, C++)

```
int* P1 = malloc(64);            // allocation
Point* P2 = new Point(2,3);      // création
delete P1, P2;                   // nid à bugs!
```

- Automatique (Lisp, Smalltalk, CAML, Java)
  - objets créés et détruits automatiquement
  - Garbage Collector (GC)

Automatique = moins efficace mais tellement plus sûr!

A quoi sert d'être efficace si on ne maîtrise pas?

Architecture

- Nécessité: utilisation de composants standards
  - déterminés par leurs fonctions
  - éprouvés, validés, certifiés, prouvés,...
- Qu'est ce qu'un composant?
  1. une interface exposée à l'utilisateur
  2. un contrat d'utilisation informel ou formel
  3. une implémentation cachée à l'utilisateur
  4. un confinement et un traitement des erreurs

API = Application Programmer Interface

Exemple : la file fifo (first in - first out)

```
class Fifo {
 public:
   Fifo (int size) ; // constructor
   void Put (unsigned m)
      throw (FullError) ;
   unsigned Get ()
      throw (EmptyError) ;
   bool IsEmpty () ;
   bool IsFull () ;
... };
```

C++

```
interface FifoIntf :
   input  Put : unsigned ;
   input  Get ;
   output DataOut : unsigned;
   output IsEmpty ;
   output IsFull ;
   output EmptyError ;
   output FullError ;
end interface
```

Esterel

Conclusion

- Langage = objet complexe
- Pas d'espéranto en vue
- Equilibre délicat général / spécialisé
- Droit d'entrée devenu très élevé

Il faut militer pour des langages mathématiquement rigoureux