Petite histoire des langages de programmation

Judicaël Courant

19 novembre 2019

Lignes directrices

Introduction

Objectif

Comprendre l'évolution des langages de programmation jusqu'à Python.

Lignes directrices

Les ancêtres des ordinateurs

- Quelques exemples :

Ludique	Automates, orgues de barbarie, etc.
Industrie	Métier Jacquard.
Scientifique	Calcul de tables (trigo, log, balistiques).
Gestion	Calcul de statistiques.

- Sont conçus pour exécuter *un seul* (type de) programme(s).

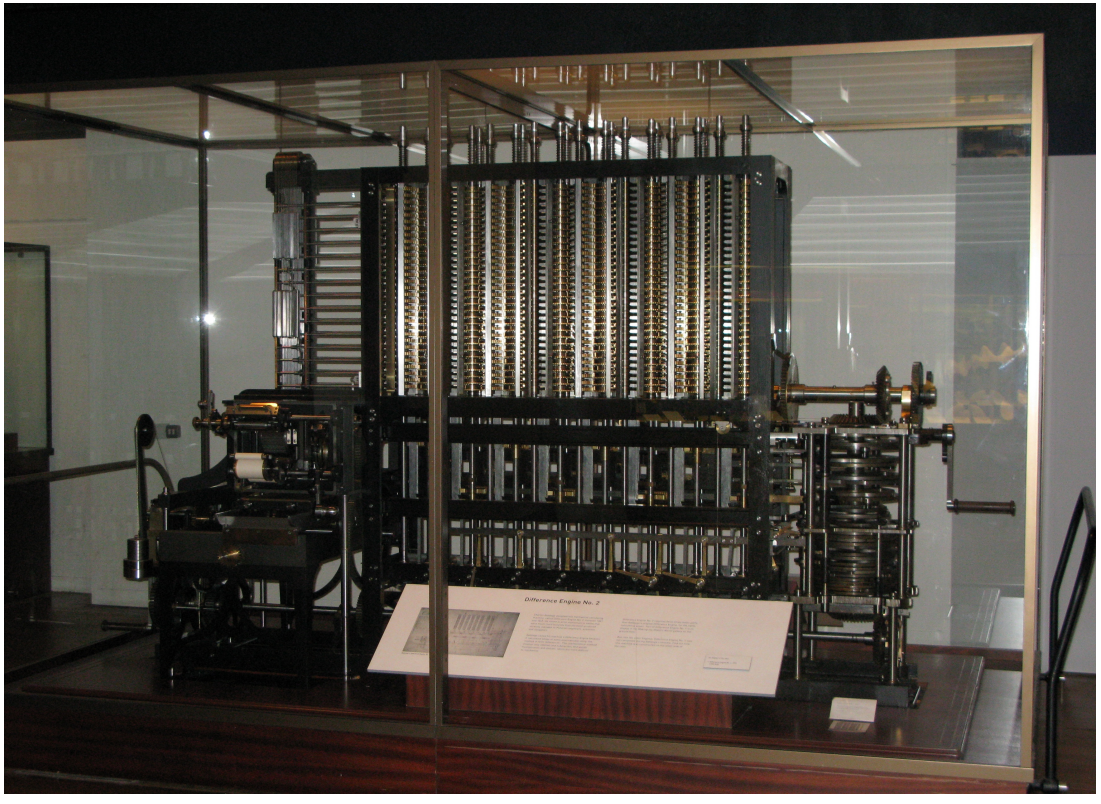
≈ 1780 : La joueuse de Tympanon

- Joue réellement du tympanon.
- Le programme est dans un cylindre en laiton commandant des cames.

1801 : Le métier Jacquard

- Programmé par cartes perforées.
- On peut changer le programme !
- Mais seulement pour tisser.

1819 – 1842 : La machine différentielle de Charles Babbage



Calcule Des polynômes (méthode des différences finies).

But Production de tables sans erreurs (log, sin, . . .).

Coût 17 k£ en finances publiques ($\approx 1,7$ M£ de 2019)

Scandale Babagge a délaissé le projet pour un autre bien plus ambitieux. . .

$\approx$ 1837 La machine analytique

Machine mythique

- Jamais construite (Babbage ruiné, projet démesuré).
- 4 opérations (racine carrée en option).
- Mémoire d'environ $16ko$ (énorme).
- Programmable par cartes perforées.
- Puissance fournie par une machine à vapeur.

Documents principaux

- Transcription en 1842 d'une conférence de Babagge en Français.
- Traduite en anglais par Ada Lovelace...
- ... qui ajoute le triple de notes et commentaires !
- Elle est le probablement le premier programmeur de l'histoire.

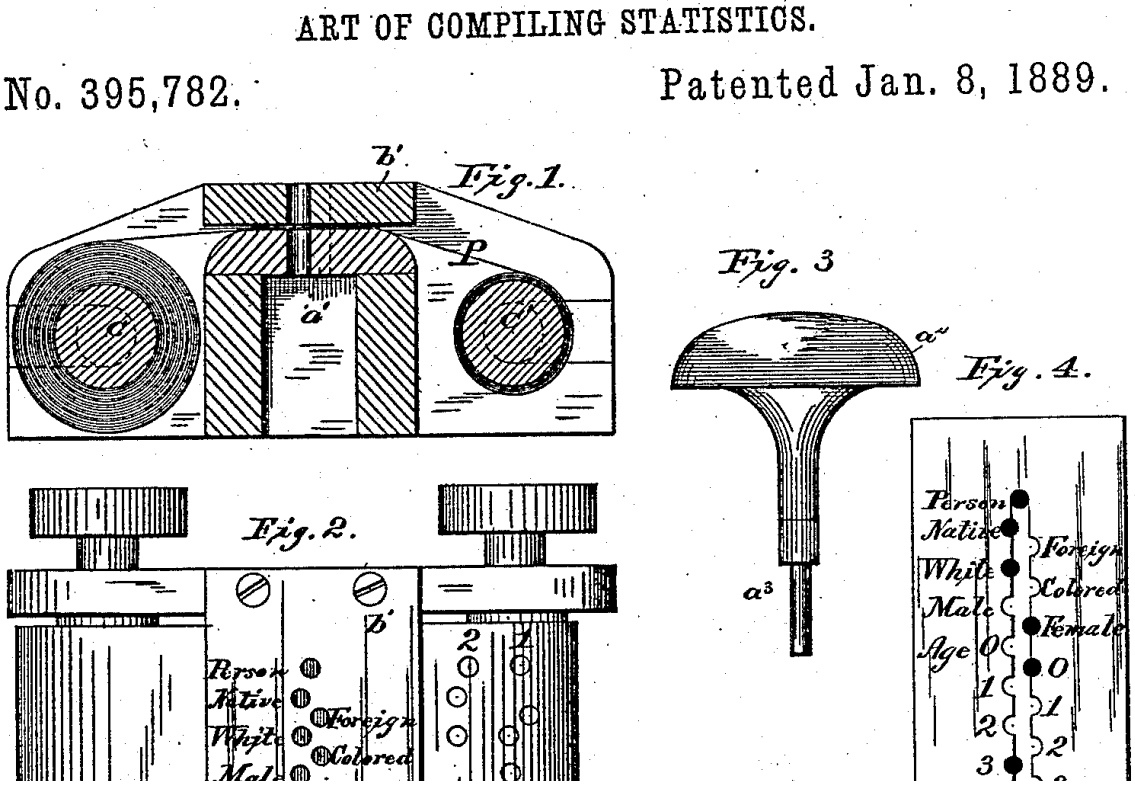
Ada Lovelace (1815-1852) : vous avez dit numérique ?

« De nombreuses personnes qui connaissent mal les études mathématiques pensent que parce que le travail de la machine est de donner des résultats en notation numérique, la nature du processus doit forcément être [...] numérique [...]. C'est une erreur... La machine peut produire trois types de résultats : [...] symboliques [...] ; numériques [...] ; et algébriques en notation littérale. »

Notes d'Ada Lovelace, 1842.

(peut-être qu'un jour, on enseignera l'informatique plutôt que le numérique)

≈ 1880 Débuts de la mécanographie


ART OF COMPILING STATISTICS.

No. 395,782. Patented Jan. 8, 1889.

Fig. 1.

Fig. 2.

Fig. 3

Fig. 4.

Idée (Hollerith)

Utiliser des cartes perforées pour stocker et traiter des *données*.

1890 : Tabulatrice Hollerith

Figure – Tabulatrice Hollerith

- Utilisée pour le recensement de 1890 aux USA ;
- Le recensement de 1880 avait duré 8 ans !

Lignes directrices

1936 : À propos du *Entscheidungsproblem* de Hilbert

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions

Article fondateur de l'informatique théorique :

- Modélise la notion d'algorithme
- À l'aide d'une machine (théorique)
- Modèle des (futurs) ordinateurs

Une machine de Turing en papier

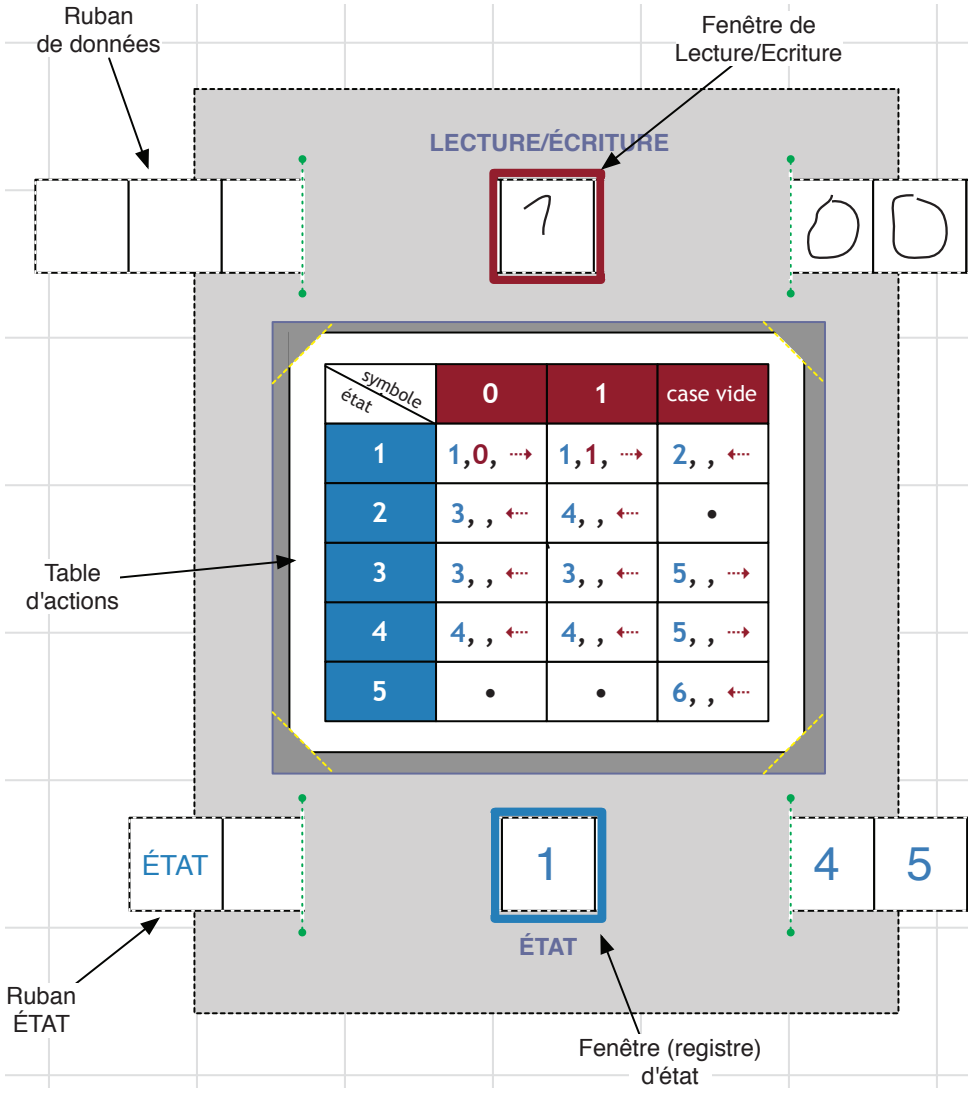


Figure – Computer Paper (Espace-Turing.fr)

La machine de Turing

Exécute : un unique programme (comme toute machine à l'époque) ;

Programme : défini par la table de transitions T (et l'ensemble des états) ;

Entrée : suite finie de symboles s codée sur le ruban ;

Résultat : suite finie de symboles s' sur le ruban quand la machine s'arrête, $\perp$ si elle ne s'arrête pas.

Ruban : infini à gauche et/ou à droite.

1936 : Existence d'une machine universelle

Définition

U *machine universelle* $\iff_{def}$ pour toute machine M et toute donnée d'entrée, ces deux calculs donnent le même résultat :

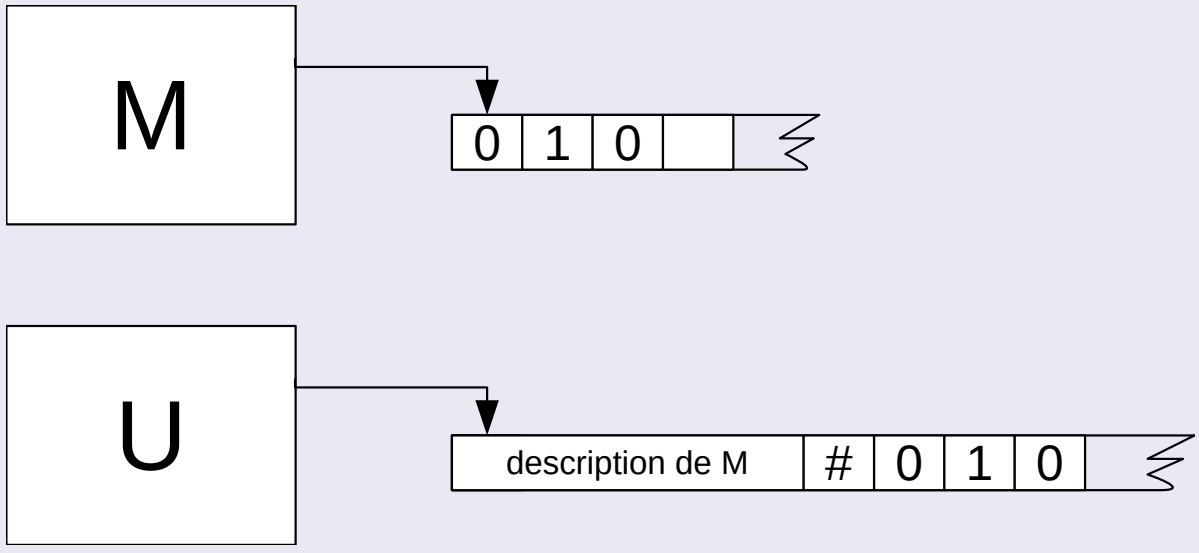


Figure – Machine de Turing universelle

Théorème (Turing 1936)

Il existe une (des) machine(s) universelle(s).

Les machines universelles

- Ouvrent la voie à des machines complètement programmables après leur fabrication.
- Mais personne n'en comprend la portée à l'époque !
- Il faudra attendre $\approx$ 1945 pour la mise en pratique.
- En attendant, on fait des machines :
  - électromécaniques ;
  - plus ou moins générales ;
  - qu'on peut recâbler au besoin.

≈ 1945 : Premiers ordinateurs

Quel a été le premier ?

- Une dizaine de prétendants au titre (Z3 1941, ENIAC 1945, . . .)
- L'EDVAC en fait partie :
  - calculateur en base 2 entièrement électronique ;
  - conçu et réalisé de 1944 à 1949 ;
  - budget 100 k$ ($\approx 1,4$ M$ 2019)

First Draft of a Report on the EDVAC

- John Von Neumann, 1945.
- Décrit l'architecture de l'EDVAC.
- Rapport classifié.
- Se diffuse très vite mondialement.
- Von Neumann n'est pas le (seul) concepteur de l'EDVAC.
- Mais on parle de l'*architecture de Von Neumann*.

1945 : L'architecture de Von Neumann

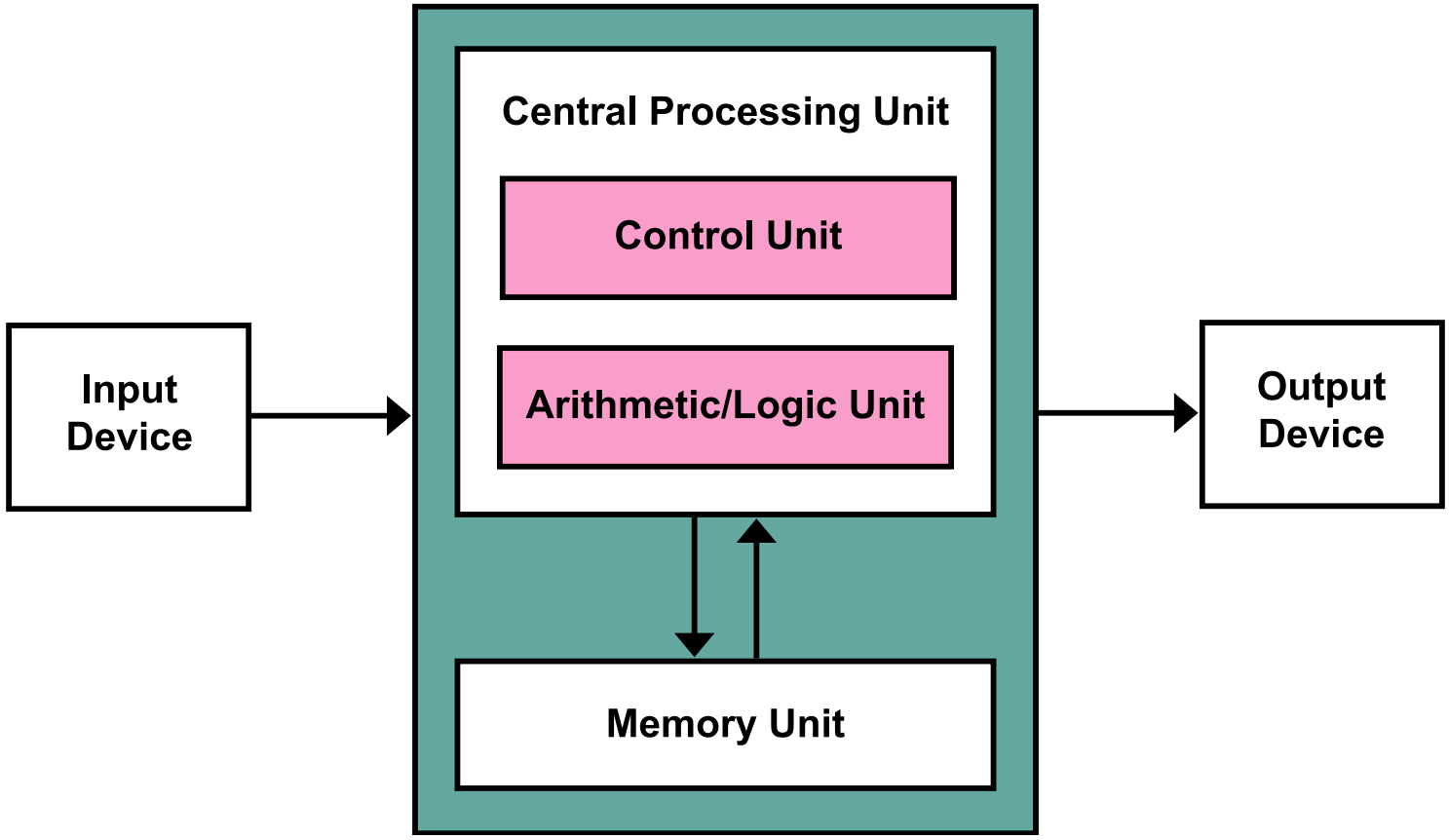
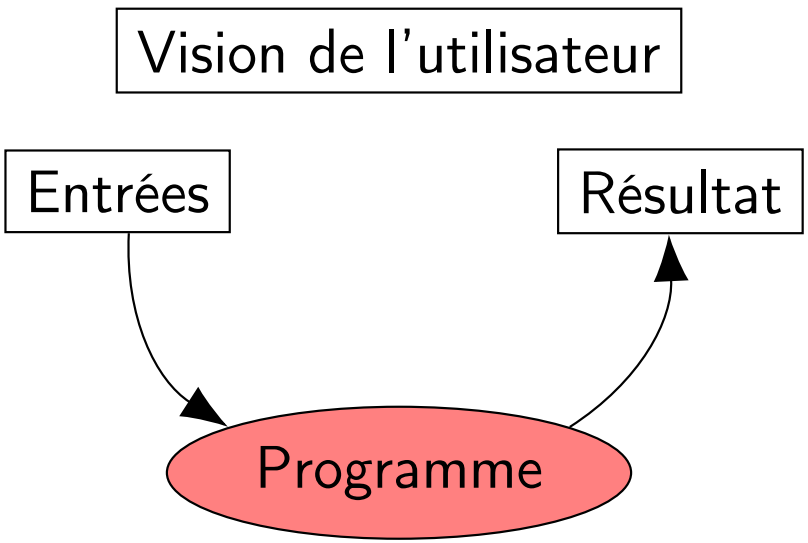
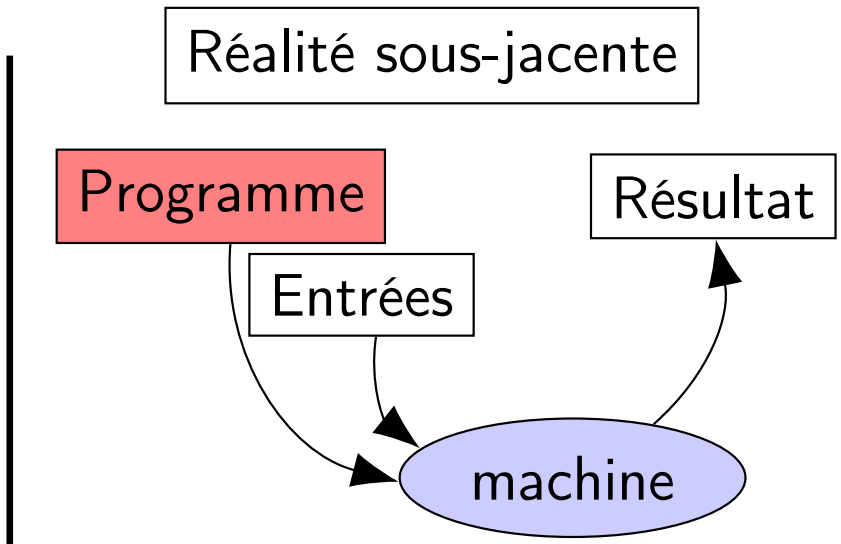


Figure – L'architecture de Von Neumann

Avec les premiers ordinateurs





Lignes directrices

Et si on faisait du langage machine ?

Un exemple anachronique :

- J'ai écrit la fonction $(x, y) \mapsto x * (y + 7) + 48$
- En langage machine (pour mon PC).
- Elle tient en 12 octets seulement !
- La voici

Et si on faisait du langage machine ?

Un exemple anachronique :

- J'ai écrit la fonction $(x, y) \mapsto x * (y + 7) + 48$
- En langage machine (pour mon PC).
- Elle tient en 12 octets seulement !
- La voici (en binaire) :

```
100010011111000010000011110000000000011100001111
101011111100011110000011110000000011000011000011
```

Et si on faisait du langage machine ?

Un exemple anachronique :

- J'ai écrit la fonction $(x, y) \mapsto x * (y + 7) + 48$
- En langage machine (pour mon PC).
- Elle tient en 12 octets seulement !
- La voici (en binaire) :

```
100010011111000010000011110000000000011100001111
101011111100011110000011110000000011000011000011
```

- De façon bien plus lisible, en hexadécimal :

```
89 f0 83 c0 07 0f af c7 83 c0 30 c3
```

Et si on faisait du langage machine ?

Un exemple anachronique :

- J'ai écrit la fonction $(x, y) \mapsto x * (y + 7) + 48$
- En langage machine (pour mon PC).
- Elle tient en 12 octets seulement !
- La voici (en binaire) :

```
100010011111000010000011110000000000011100001111
101011111100011110000011110000000011000011000011
```

- De façon bien plus lisible, en hexadécimal :

```
89 f0 83 c0 07 0f af c7 83 c0 30 c3
```

- C'est difficile à écrire sans entraînement.
- Ça reste peu lisible.

Les mnémoniques

Dans le manuel du processeur, chaque instruction a un petit nom, appelé *mnémonique*. On préfère écrire sous cette forme, car c'est bien plus lisible :

```
movl   %esi, %eax
addl   $0x7, %eax
imull  %edi, %eax
addl   $0x30, %eax
ret
```

La phase d'*assemblage*

- On écrit le programme en mnémoniques
- On le traduit en langage machine : c'est la phase d'*assemblage*.

```
1145:       89 f0                movl   %esi,%eax
1147:       83 c0 07             addl   $0x7,%eax
114a:       0f af c7             imull  %edi,%eax
114d:       83 c0 30             addl   $0x30,%eax
1150:       c3                   ret
```

Pour l'avoir pratiqué à la main (il y a 0x20 ans), je peux témoigner que :

- c'est pénible à faire ;
- c'est source d'erreurs ;
- qui sont difficiles à débusquer !

Les assembleurs

Très vite, des programmes ont été écrits pour faire cette traduction :

les *programmes d'assemblage* ou *assembleurs*

- Investissement lourd (rentable si beaucoup de programmes à traduire).
- Recette :
  - Écrire son assembleur en mnémoniques ;
  - le traduire à la main en binaire ;
  - finie la traduction manuelle !
- Par métonymie, on appelle *langage assembleur* le langage des mnémoniques.

Vécu :

- un assembleur change votre vie !

Au delà des assembleurs

- Écrire en langage assembleur est long, pénible et difficile ;
- D'où l'idée de se simplifier la vie :
  - Définissons une notation de haut niveau pour les programmes ;
  - Écrivons un programme pour le faire comprendre de la machine.

Quelques considérations économiques

Domaines d'applications

- Initialement scientifique (militaire) : tables balistiques, aéronautique.
- Gestion :
  - marché postérieur mais beaucoup plus important ;
  - fabrication en série : baisse du prix du matériel ;
  - pénurie de programmeurs qualifiés.

Objectif de réduction des coûts

Coût des programmes énorme :

- Besoin de programmeurs de haut niveau.
- Faible productivité des programmeurs.
- Débogage très difficile.

Concevoir des langages de haut niveau doit permettre de réduire les coûts.

Considérations sociologiques

Le logiciel est initialement considéré subalterne

- Il s'agit de «câbler» l'ordinateur.
- C'est un travail laissé aux femmes (très qualifiées).

De fait

ENIAC	6 programmeurs, 100% de femmes
1972-1985	Proportion élevée d'étudiantes en informatique.

Une considération informatique

Inventer de nouveaux langages permet de mieux penser

Hypothèse de Sapir-Whorf

Les représentations mentales dépendent des catégories linguistiques.

Application aux langages de programmation

> *A language that doesn't affect the way you think about programming, is not worth knowing.*
>
> Alan Perlis, ACM SIGPLAN Notices 17 (9), 1982.

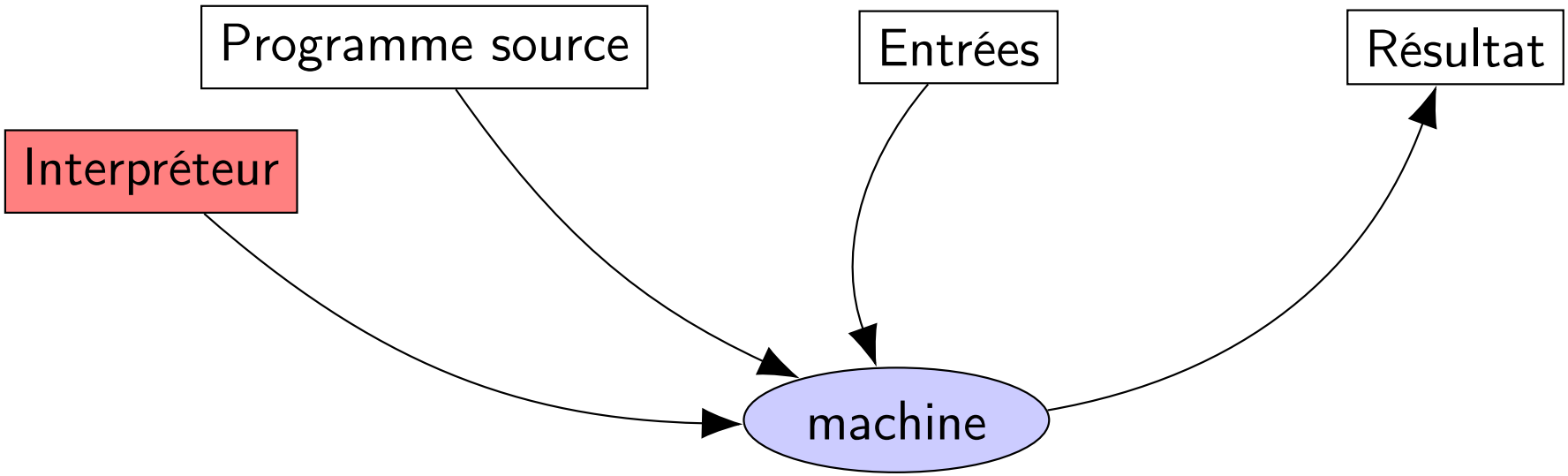
Lignes directrices

Interpréteurs et compilateurs

Deux choix possibles pour implanter un langage de haut niveau.

- Interpréteur
- Compilateur

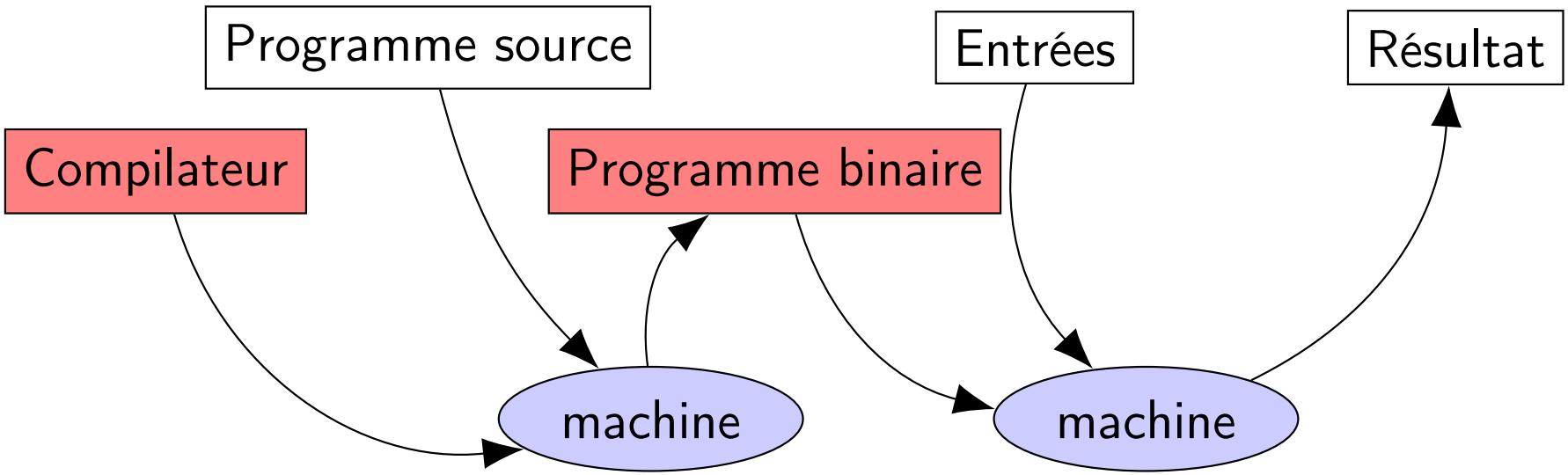
Interpréteur



Idée

- Écriture en binaire ou assembleur d'un *interpréteur*.
- L'interpréteur exécute un programme écrit dans un langage de haut niveau (programme *source*).
- Exécution et décodage du programme sont simultanés.
- Premier interpréteur : 1960 (John McCarthy, Steve Russel)

Compilateur



Idée

- Deux phases :
  - Traduction par le compilateur langage source $\rightarrow$ langage binaire.
  - Exécution par la machine du binaire obtenu.
- Premier compilateur : 1952 (Grace Hopper)

Interpréteur ou compilateur ?

Interpréteur

- Relativement facile à écrire.
- Permet une interaction avec l'utilisateur.
- Temps de traduction proportionnel au temps d'exécution total.

Compilateur

- Plus compliqué à écrire qu'un interpréteur.
- Moins gourmand en ressources :
  - Temps de traduction proportionnel à la taille du programme.
  - Espace mémoire nécessaire plus faible (traduction / exécution séparées).

Lignes directrices

Trois paradigmes majeurs

Impératif : `x = x + 1`

- On modifie l'état d'une machine
- Proche de la machine donc a priori efficace
- Peu mathématique, peu naturel

Fonctionnel : $f \mapsto f(0)$

- Proche des mathématiques : spécification mathématique facile
- Originellement difficile à exécuter efficacement
- Parfois trop l'apanage des théoriciens (ou perçu comme tel)

Objet : `window.close()`

- Le paradigme des interfaces graphiques
- Très en vogue dans les années 90
- Séduisant mais souffre d'adaptations parfois discutables

Lignes directrices

1954-1956 : FORTRAN

«Since FORTRAN would virtually eliminate coding and debugging, it should be possible to solve problems for half the cost»

J. Backus, *Preliminary report [...] FORmula TRANslating System*, 1954

- Langage pour les physiciens.
- Fonctionnalités de FORTRAN :
  - expressions arithmétiques ;
  - boucle définie (équivalent de `for`) ;
  - saut conditionnel ;
  - types entier et flottant uniquement.
- Mais :
  - variables globales uniquement ;
  - pas de fonctions (autres que prédéfinies).
- Deux ans d'efforts
- 25 000 lignes pour le premier compilateur

Exemple de programme FORTRAN

Programme FORTRAN II (1958) :

```
C AREA OF A TRIANGLE
      READ INPUT TAPE 5, 501, IA, IB, IC
  501 FORMAT (3I5)
      IF (IA) 777, 777, 701
  701 IF (IB) 777, 777, 702
  702 IF (IC) 777, 777, 703
  703 IF (IA+IB-IC) 777, 777, 704
  704 IF (IA+IC-IB) 777, 777, 705
  705 IF (IB+IC-IA) 777, 777, 799
  777 STOP 1
C USING HERON'S FORMULA
  799 S = FLOATF (IA + IB + IC) / 2.0
      AREA = SQRTF( S * (S - FLOATF(IA)) * (S - FLOATF(IB)) *
     +      (S - FLOATF(IC)))
      WRITE OUTPUT TAPE 6, 601, IA, IB, IC, AREA
  601 FORMAT (4H A= ,I5,5H  B= ,I5,5H  C= ,I5,8H  AREA= ,F10.2,
     +        13H SQUARE UNITS)
      STOP
      END
```

FORTRAN

- FORTRAN I : à peine un langage de haut niveau.
- Mais révolutionnaire à l'époque : énorme succès !
- Longue évolution (des sous-routines de Fortran II à Fortran 202x).
- Mais qui traînent le poids de la compatibilité.
- Évolution des mentalités des physiciens :

1990	Fortran, what else ?
Aujourd'hui	Oh non, du Fortran ! Je vais le piloter en Python.

1972 : Fortran vu par Dijkstra

The sooner we can forget that FORTRAN has ever existed, the better, for as a vehicule of thought it is no longer adequate : it wastes our brainpower, is too risky and therefore too expensive to use. FORTRAN's tragic fate has been its wide acceptance, mentally chaining thousands and thousands of programmers to our past mistakes. I pray daily that more of my fellow-programmers may find the means of freeing themselves from the curse of compatibility.

Edsger W. Dijkstra, The Humble Programmer, 1972.

≈ 1960 Algol 60 (**alg**orithmic **o**riented **l**anguage)

Introduction de la programmation structurée :

- Procédures et fonctions globales *et locales*
- Typage statique
- Récursivité
- Blocs (`Begin` / `End`)
- Boucle `for` (fait aussi office de boucle `while`)

```
procedure Transpose (a) Order: (n); value n;
array a; integer n;
begin real w; integer i, k;
for i:=1 step 1 until n do
    for k:=1+i step 1 until n do
    begin w:=a[i, k];
          a[i, k]:=a[k, i];
          a[k, i]:=w
    end
end Transpose
```

- Auteurs : Backus, Naur, McCarthy, Perlis, ...

Algol 60 : un grand succès

- Langage défini proprement, notamment à l'aide d'une *grammaire* BNF (*Backus-Naur Form*).
- Langage apprécié par ceux qui l'ont utilisé.
- Forte influence par la suite : Simula, Pascal, ...

2.4. IDENTIFIERS.

2.4.1. Syntax.

<identifier> ::= <letter>|<identifier><letter>|<identifier><digit>

2.4.2. Examples. q

Soup

V17a

a34kTMNs

MARILYN

Figure – La grammaire des identificateurs

≈ 1960 COBOL

Objectif et réalisation

- La gestion : *COmmon Business-Oriented Language*.
- Conçu par un comité.
- Influence de Grace Hopper, inventeuse de FLOW-MATIC.

Succès commercial

- Fait pour manipuler des fichiers de données.
- Standardisé, fait pour être portable.
- Imposé par le DoD pour ses fournisseurs.
- Langage le plus utilisé dans le monde en 1970.
- En 1997 :
  - 80% des applications de gestion
  - 200×10^9 lignes de code
  - $+5 \times 10^9$ chaque année

COBOL : un échec scientifique ?

Des apports

- Notion d'enregistrement.
- Gestion des lectures/écritures dans fichiers structurés en enregistrement.

Défauts du langage :

- Non structuré (GOTO).
- Verbeux (300 mots clés).
- Notion de sous-routine. . . à manipuler avec prudence !

Absence de lien avec la recherche

- Industriels et scientifiques se sont ignorés.
- Explique les défauts du langage et sa difficile évolution.
- Notion de grammaire formelle réinventée pour spécifier COBOL.

≈ 1967 : compilation vers des machines virtuelles

Deux descendants d'ALGOL : BCPL et Pascal

BCPL

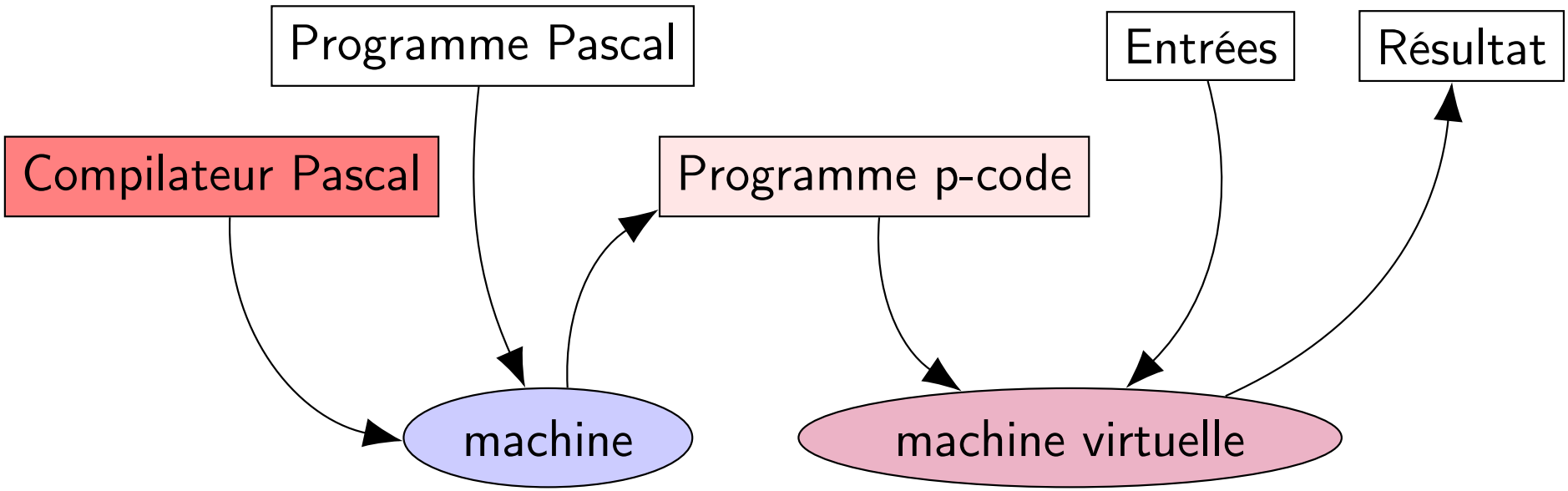
- Compilation en deux phases :
  - Compilateur de BCPL vers du code pour une machine virtuelle (O-code)
  - Compilateur du O-code vers le code de la machine réelle
- Avantage : moins de 20% du compilateur à récrire pour une nouvelle machine

Pascal

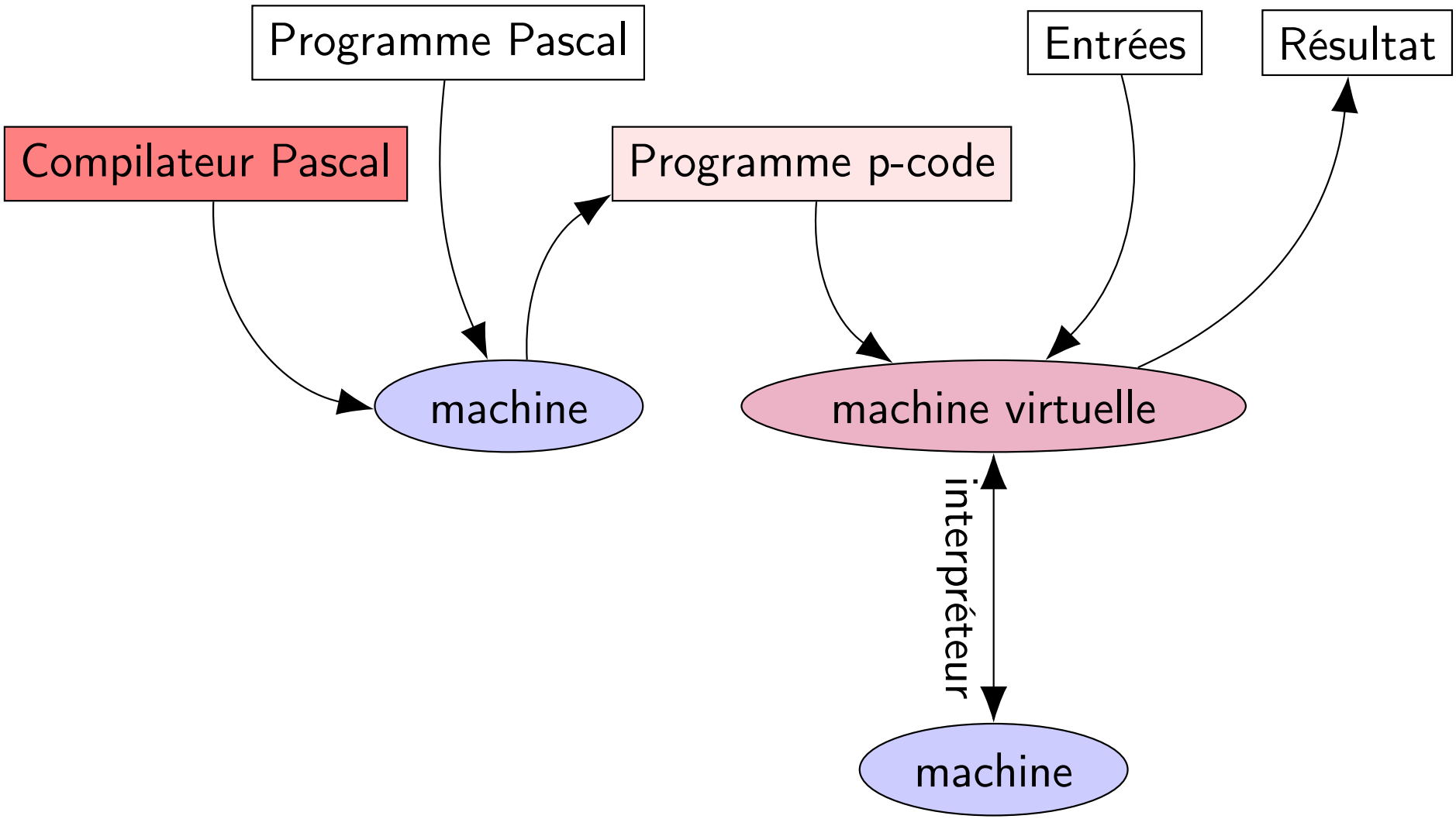
Pousse l'idée plus loin :

- Compilation vers une machine virtuelle (p-code)
- Exécution du p-code par un *interpréteur* de p-code.

Compilation et exécution de Pascal



Compilation et exécution de Pascal



≈ 1970 : le langage C (Ken Thompson et Denis Ritchie)

- Issu de BCPL (d'abord appelé B).
- Développé aux Bell Labs.
- Manipulations de bas niveau : allocation mémoire, pointeur.

```
/* Renvoie le tableau { 0**2, 1**2, ..., (n-1)**2 } */
int* carres(int n) {
  int* p;
  int i;
  p = (int *) malloc(n * sizeof(int));
  for (i = 0; i < n; i++) { p[i] = i * i; }
  return p;
}
```

- Descendants : C++, Objective C (NextStep, Mac OS X).

C : un succès planétaire.

Unix : un nouvel OS pour développeurs...

- 1969 : Développé aux Bell Labs.
- 1972 : C développé pour écrire des programmes sous Unix.
- 1973 : Unix récrit en C pour une nouvelle machine.
- 1973 : Donné car ne peut être vendu (loi antitrust).
- 1975 : Jugé intéressant pour ARPAnet (aujourd'hui Internet).
- 1980 : Disponible sur 16 types de machines différentes.

... qui rend C hégémonique

- Sur les stations de travail.
- Sur les systèmes et programmes qui y sont développés.
- Sur tous les Unix (Linux, Mac OS, Android) et MS-Windows

C : une catastrophe planétaire.

Gestion mémoire manuelle

- Origine de 80% des bugs en C.
- Cause de la majorité des failles de sécurité des systèmes d'exploitation.

Une évolution difficile

- Monoculture et poids de la compatibilité.
- C++ a les mêmes problèmes.
- Peu d'alternatives aujourd'hui.
- Une piste : Rust ?

Lignes directrices

≈ 1958 : Définition de LISP (John McCarthy)

≈ 1930 calculabilité du point de vue logique (avant Turing)

- fonctions récursives (Herbrand, Gödel)
- λ-calcul (Church)

LISP

- Défini mathématiquement :
  - Notion d'*expressions* représentant des *valeurs* ;
  - Fonction d'évaluation *eval* réduisant une expression en valeur : $\mathrm{eval}[(+\ 3\ x); ((x \cdot 2))] = 5$
- Les expressions peuvent représenter des nombres mais aussi des listes, des arbres, des expressions symboliques.
- D'où le nom : LISt Processing language.

≈ 1958 : Implantation de LISP

- L'équipe de McCarthy se lance dans la réalisation d'un compilateur.
- Projet très ambitieux (manipulation d'expressions symboliques).

Une surprise

- Steve Russel implante (en assembleur) la fonction (mathématique) d'évaluation.
- Ça marche ! ! !
- On obtient ainsi un interpréteur :
  - on demande une expression à l'utilisateur (read) ;
  - on calcule sa valeur (eval) ;
  - on affiche le résultat (print) ;
  - et on recommence.
- LISP est né comme un système *interactif*

LISP : un succès peu connu

- Le langage des débuts de l'intelligence artificielle.
- Nombreux succès :
  - Logiciels de calculs formels (Macsyma 1968, etc.).
  - Démonstrateurs automatiques (Boyer-Moore, etc.).
  - Compilateurs.
  - AutoCAD.
  - Emacs.
  - Viaweb (Yahoo ! stores).
  - Jak and Daxter.
- De nombreuses implantations aujourd'hui.
- Dialectes modernes : Clojure, Scheme (Guile), Racket.

LISP : quelques citations

Lisp has jokingly been called "the most intelligent way to misuse a computer". I think that description is a great compliment because it transmits the full flavor of liberation : it has assisted a number of our most gifted fellow humans in thinking previously impossible thoughts.

Edsger W. Dijkstra, The Humble Programmer, 1972

Greenspun's Tenth Rule : Any sufficiently complicated C or Fortran program contains an ad hoc informally-specified bug-ridden slow implementation of half of Common Lisp.

LISP : Quelques descendants notables

1966 : ISWIM (If you See What I Mean), P. Landin

- Langage théorique resté à des implantations expérimentales
- Indentation significative
- Réflexions profondes sur les langages de programmation :
  - Doivent procéder d'une conception et non d'un empilement de fonctionnalités
  - Distinction syntaxe concrète/syntaxe abstraite/sémantique.

1973 : ML (Robin Milner)

- Typé statiquement mais *implicitement*
- Représentants notables :
  - Caml (G. Cousineau, 1985)
  - OCaml (X. Leroy, 1996)

≈ 1990 : Haskell, un langage paresseux

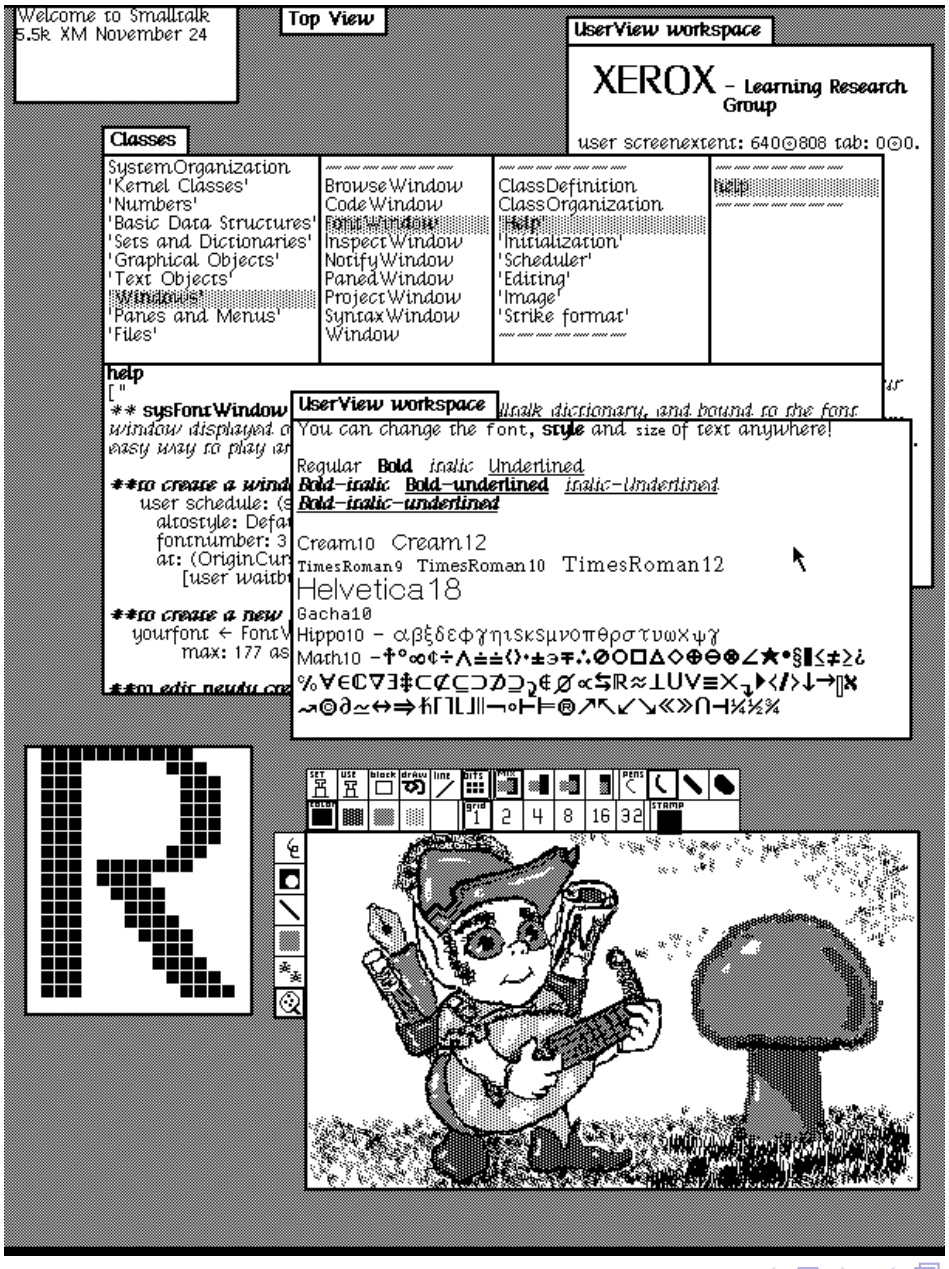
- Inspiré par LISP, ISWIM et Miranda (1985)
- Standardisé.
- Paresseux. Le programme ci-dessous termine :

```
f (n) = n + f(n-1) -- fonction ne terminant pas
g (x, y) = x
g(1+2, f(3+5)) -- f(3+5) n'est pas évalué ici
```

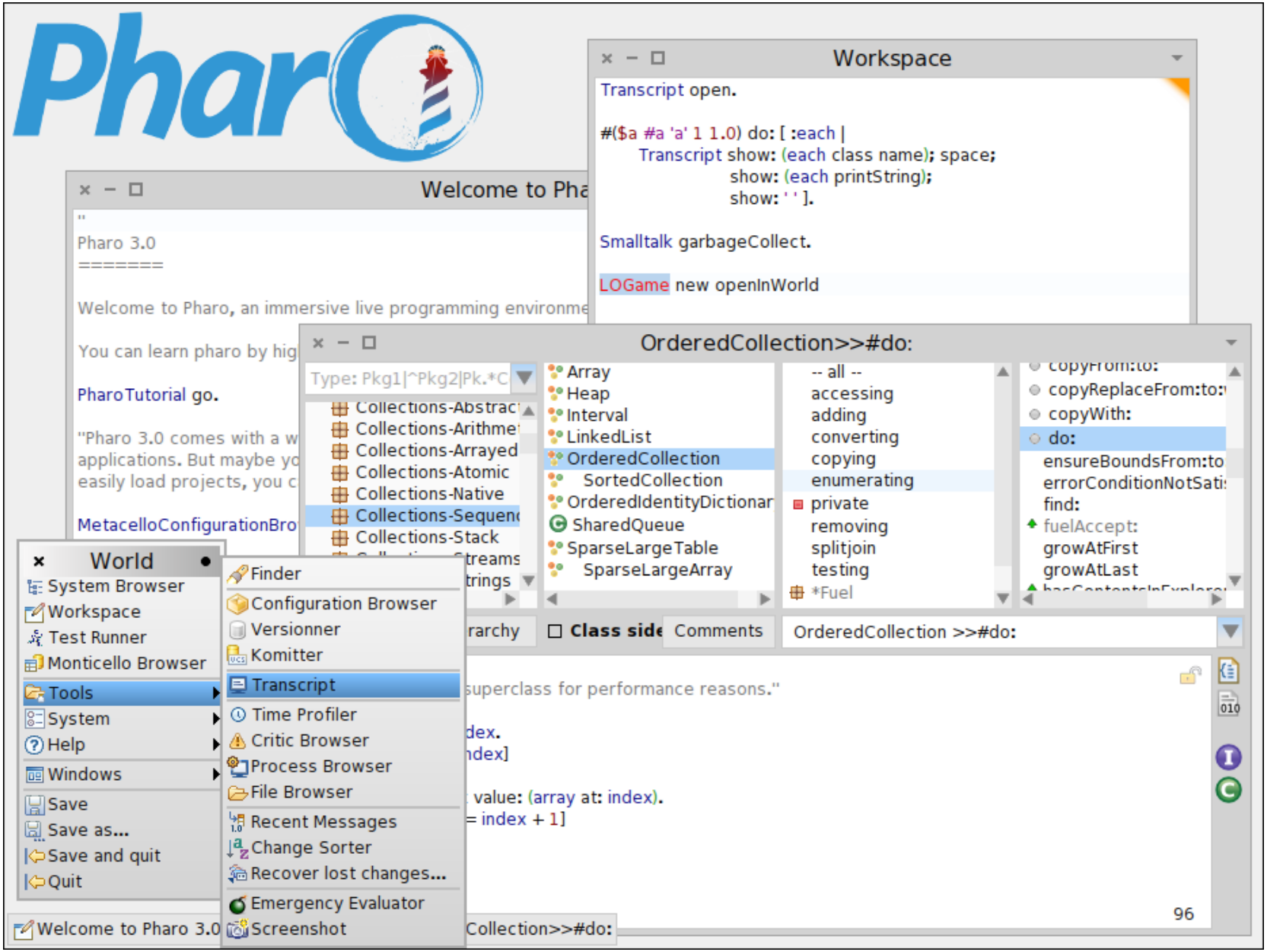
- La paresse permet de manipuler des structures de données infinies.

Lignes directrices

≈ 1971 : Smalltalk (Alan Kay)



Smalltalk : le roi de l'interface graphique



≈ 1971 : Smalltalk

- Langage dynamiquement typé issu de Simula 67.
- Idée :
  - *objets* auxquels on envoie des *messages* ;
  - tout est objet.
- Syntaxe minimaliste (6 mots-clés en Smalltalk-80).

```
exampleWithNumber: x
    "Toute la syntaxe de Smalltalk est là."
    | y |
    true & false not & (nil isNil) ifFalse: [self halt].
    y := self size + super size.
    #($a #a "a" 1 1.0)
        do: [ :each |
            Transcript show: (each class name);
                       show: ' '].
    ^x < y
```

Figure – Toute la syntaxe smalltalk en une carte postale.

Smalltalk : apprécié et imité (mais peu connu)

Des environnement de programmation inégalés

- Exploration et modification interactive des objets d'un programme.
- Y compris leur code.
- Pendant l'exécution !

Un nouveau paradigme : la *Programmation Orientée Objets*

- Descendants : CLOS, C++, Objective-C, Java, Python, Ruby.
- Versions modernes : Squeak, Pharo (*Seaside*).

Lignes directrices

1991 : Python (G. van Rossum)

Origine

- Langage à visée pédagogique

Paradigmes

- Impératif
- Fonctionnel
- Objet

Succès de Python

Points forts

- Syntaxe claire.
- Interactivité.
- Simple pour le débutant.
- Système de modules simple et efficace.
- «*Batteries included*» : nombreuses bibliothèques.

Succès au-delà de l'enseignement

- Calcul numérique (numpy / scipy) et bigdata.
- Applications web.

Points faibles

- Faibles performances (de l'implantation de référence).
- Très peu de vérifications avant l'exécution.

Implantation de référence (CPython)

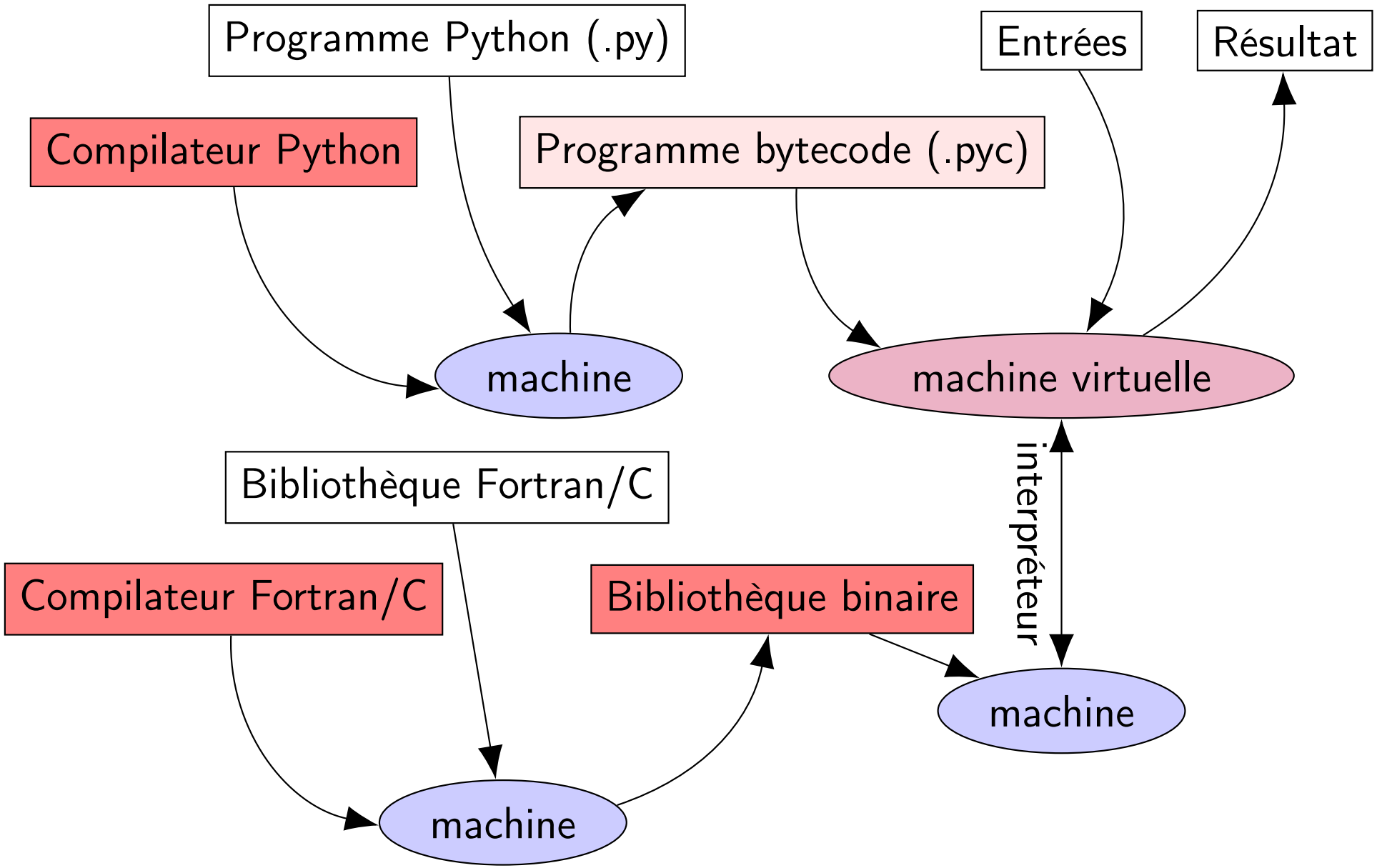
Conceptuellement, deux programmes

- Compilateur vers le binaire d'une machine virtuelle (bytecode).
- Interpréteur de bytecode.

Compilateur et interpréteur de bytecode

- Écrits en C.
- Compilés (par un compilateur C) pour obtenir un exécutable.
- Portables sur toute machine de type Unix (plus Windows).

Implantation de référence (CPython)



Lignes directrices

Copyleft

Cette oeuvre est libre, vous pouvez la copier, la diffuser et la modifier selon les termes de la Licence Art Libre `http://artlibre.org`, version 1.3 ou ultérieure.