Une brève histoire de l'informatique

3. Histoire du logiciel

après 1960

Sacha Krakowiak

Université Grenoble Alpes & Aconit

Une brève histoire de l'informatique

- La naissance de l'informatique
- Les ordinateurs
- Le logiciel
- Les systèmes d'exploitation et les réseaux
- L'intelligence artificielle

Au sommaire

- Le génie logiciel

  Approche raisonnée de la production de logiciel

- Les langages de programmation

  Après les fondateurs (Fortran, Algol, Cobol, Lisp), de nouveaux paradigmes

- Les bases de données

  Conservation, organisation, manipulation de l'information

Les systèmes d'exploitation seront traités avec les réseaux

Le génie logiciel

✤ Les motivations

Produire des programmes corrects est une entreprise difficile…

… mais il a fallu longtemps pour s'en rendre compte

Il est déjà difficile de définir un «programme correct»

Un visionnaire :

«Je pris conscience, et cette révélation me frappa avec toute sa force, que j'allais passer une bonne partie du restant de ma vie à trouver des erreurs dans mes propres programmes».

Maurice Wilkes, 1949

✤ L'objectif

Définir et appliquer des méthodes, des outils et des pratiques, *fondés sur des principes scientifiques*, et propres à assurer la production de logiciel répondant à des besoins spécifiés et respectant certains critères de qualité, eux-mêmes spécifiés, ainsi que des contraintes économiques.

✤ A-t-on réussi ?

Les années 1956-69

- Les premiers langages de programmation

  FORTRAN, COBOL, Algol 60...

  Utilisés pour les applications…

  … mais pas pour le logiciel de base

  Améliorent les choses, mais n'éliminent pas les erreurs

- Les premiers logiciels de grande taille

  5.10^5 à 5.10^6 lignes de code ; des dizaines, voire centaines de programmeurs

  SABRE, SAGE

  première notion de «cycle de vie»

  OS/360

  «*a multi-million dollar mistake*» (Fred Brooks)

- Les débuts de l'industrie du logiciel

  Les sociétés de service, le «dépaquetage» (*unbundling*, IBM 1969)

La «crise du logiciel»
(fin des années 1960)

✤ Un constat

On ne maîtrise plus le développement des grands projets logiciels

- délais non tenus
- budget dépassé
- qualité médiocre, voire échec total

✤ Une prise de conscience

Produire des programmes corrects est difficile

On en est encore à un stade artisanal

✤ Des conclusions

Il faut développer des méthodes et des outils

Il faut mieux former les gens

Il faut passer à un stade industriel

L'OS/360 (1963-67)

- Un système gros et complexe, en langage assembleur (> 10^6 lignes de code)
- Un domaine nouveau (multiprogrammation)
- Une conception défaillante
- Des retards croisssants, mal traités

Au total :

- Un an de retard
- Budget $500M (X 4)
- Système de mauvaise qualité, difficile à maintenir (1 000 bugs/version)

Les conférences fondatrices

- ## Une réaction à la crise du logiciel

  Promouvoir un «génie logiciel» (*software engineering*)

  Fonder la pratique sur des principes scientifiques

  Définir les traits de cette nouvelle discipline

- ## Une préoccupation partagée

  Initiative : division des affaires scientifiques de l'OTAN

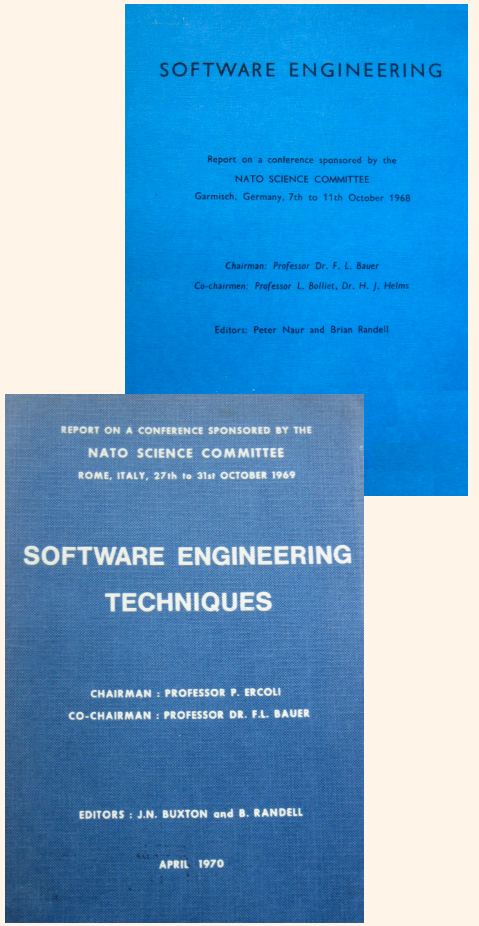
  Participation : 50 personnes

  scientifiques, constructeurs, sociétés de service

- ## Deux conférences

  Octobre 1968, Garmisch

  Octobre 1969, Rome



Les sept premières années
1968-1975

- Premières idées sur le cycle de vie
- Des avancées de la théorie… — Floyd, Hoare
  - … mais sans impact sur la pratique
- Des avancées architecturales… — Dijkstra, Wirth, Parnas
  - … pas toujours bien intégrées
- Les tout premiers outils de conception… — Z, VDM
  - … une influence encore limitée
- Les premières «méthodes» — Warnier
- Les premiers ateliers de génie logiciel — Maestro
- L'importance des facteurs humains — Weinberg
- Un premier bilan — Brooks

Le cycle de vie du logiciel : vue initiale

✤ Les étapes de la création

Que veut-on faire ?

- Cahier des charges
- Spécification

Comment le faire ?

- Le principe : l'algorithme
  - une méthode correcte et efficace
- La mise en œuvre : le programme
  - une réalisation correcte et efficace de l'algorithme

Des méthodes et outils de travail

- Décomposition
- Programmation et mise au point ⟷ A-t-on réussi ? **Jamais du premier coup !**
- Intégration de l'ensemble
- **… et on recommence**

Modélisation

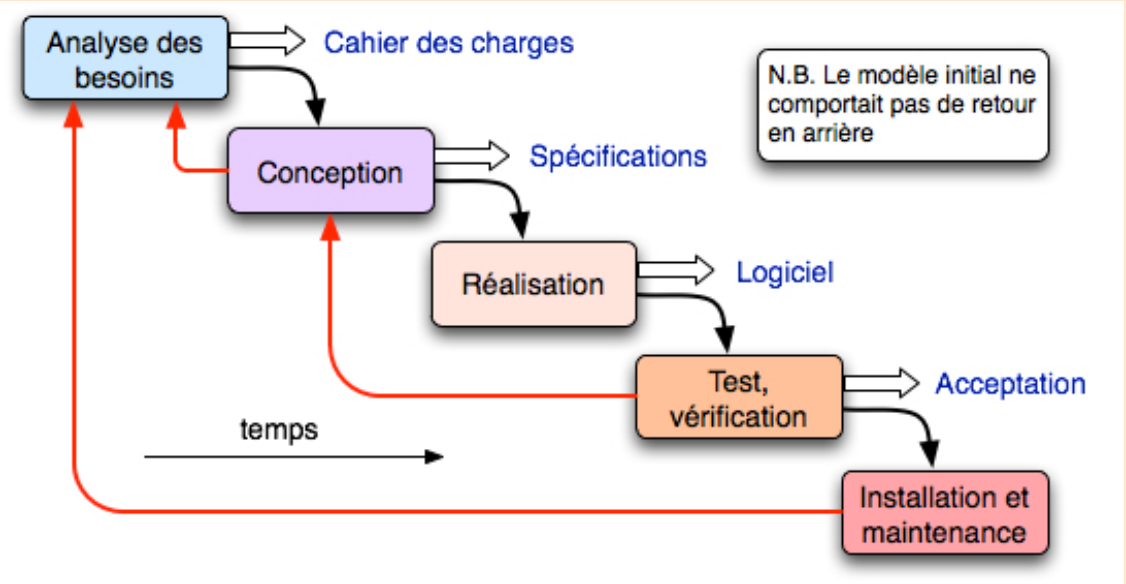
Représentation des objets du monde réel par des objets informatiques

La programmation n'est qu'une petite partie de la construction du logiciel

Test
Vérification et validation
Preuve

Le cycle de vie logiciel, en 1968

- Le modèle de la cascade



- Une autre vue…

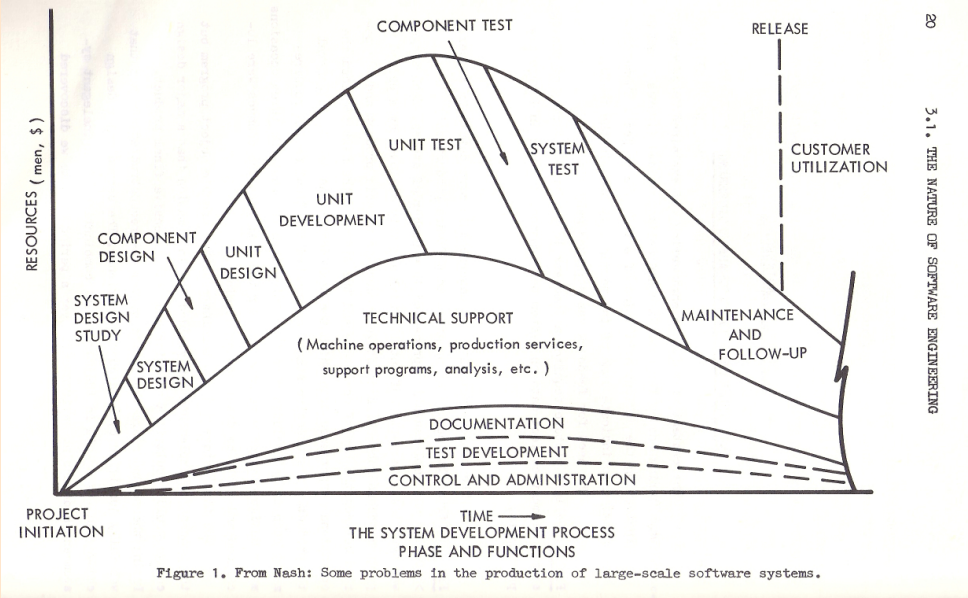


Figure 1. From Nash: Some problems in the production of large-scale software systems.

faible poids de la conception
beaucoup de ressources pour le test
peu d'accent sur la documentation

Évolution du cycle de vie

- Objectif : une plus grande réactivité…

  Cycle en V (vers 1970)

  peu de différence avec cascade

  Cycle incrémental (dès 1960)

  une version disponible tôt

  progrès par itération

  Cycle en spirale (vers 1986)

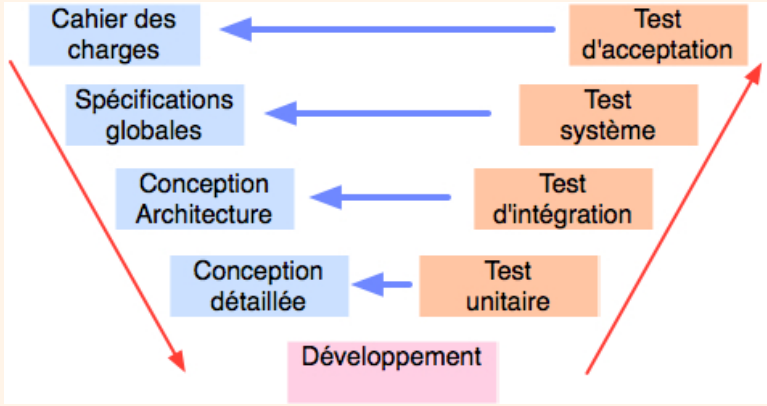
  améliore le cycle incrémental

  Cycle prototype

  le projet peut se redéfinir en fonction de l'expérience

  Cycle «agile» (vers 1995)

  cycle court, participation active du client



- … mais peu de support théorique

  Effets de mode, redécouverte, peu d'évaluation

Fred Brooks : un premier bilan (1975)

- *The Mythical Man-Month*

  Une revue critique de la situation du génie logiciel

  Quelques points saillants

- Si le projet est en retard, ajouter de la force de travail

  Faux ! Cela ne fait qu'aggraver le problème

- L'effet «second système»

  Une confiance injustifiée, facteur de risque

- L'intégrité conceptuelle, qualité principale d'un projet

  L'expérience de l'OS/360

  Le rôle central de l'architecte

  L'importance de la documentation

- Prévoyez de mettre une version à la poubelle : vous le ferez de toutes façons

Autres mythes des années 70

- ## Le mythe de la solution «par les outils»

  On résoudra les problèmes en créant des outils
  plus raffinés

  Très discutable ! la qualité des équipes est le facteur dominant

- ## Le mythe de la «programmation automatique»

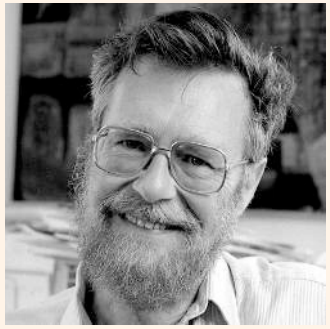
  Les programmes du futur seront créés automatiquement

  Le métier de programmeur va disparaître

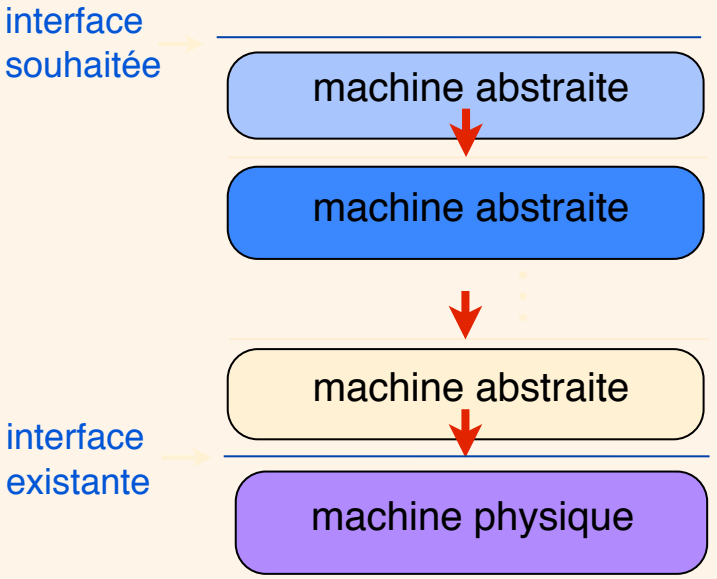
  Faux ! Le métier va se transformer, mais dans le sens
  d'une plus grande qualification

Avancées architecturales

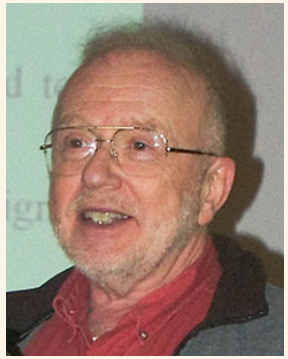
- **Objectif : organiser l'architecture globale d'un logiciel**
  - pour réduire sa complexité
- **Conception descendante**
  - hiérarchie de «machines abstraites»
  - réalisation en logiciel
  - raffinement progressif



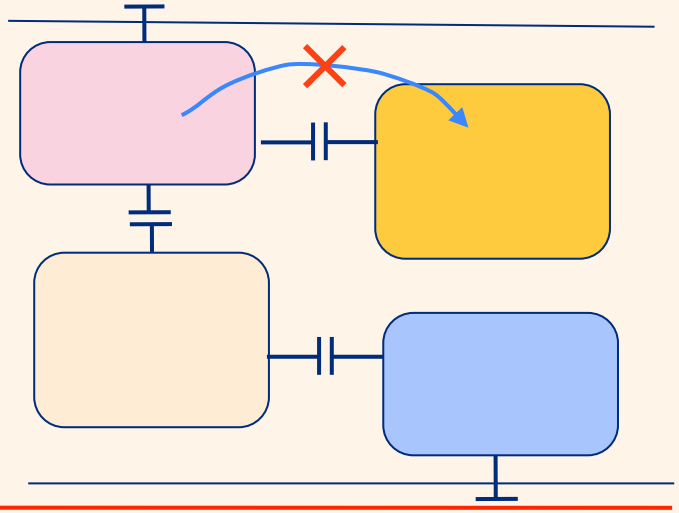
Edsger W. Dijkstra
(1930 - 2002)
crédit : U. of Texas at Austin



- **Décomposition en modules**
  - interface visible, réalisation cachée
  - tout doit passer par l'interface



David L. Parnas
(1941 -)
Wikimedia Commons, CC-BY-SA 3.0 H. Baumeister



En quête de la validité …

Pour un composant élémentaire

- Écrire l'algorithme, et se convaincre qu'il est correct
  - Par le test
  - Par la vérification
  - Par la preuve
- Construire en même temps l'algorithme et la preuve
- Engendrer l'algorithme à partir des spécifications
  - Par un procédé dont on a prouvé la validité

Pour un système complexe

- Composer les preuves
  - Déterminer les propriétés du système à partir de celles de ses composants et des règles de composition

Vérification par analyse statique

Idée : déterminer des propriétés dynamiques d'un système, sans exécuter son programme

- *Model checking* (Clarke, Emerson, Sifakis)

  Modéliser le système par un graphe de transition d'états

  Vérifier que le modèle satisfait une spécification (en logique temporelle)

- Interprétation abstraite (Cousot)

  Modéliser l'évolution du système par ses traces d'exécution

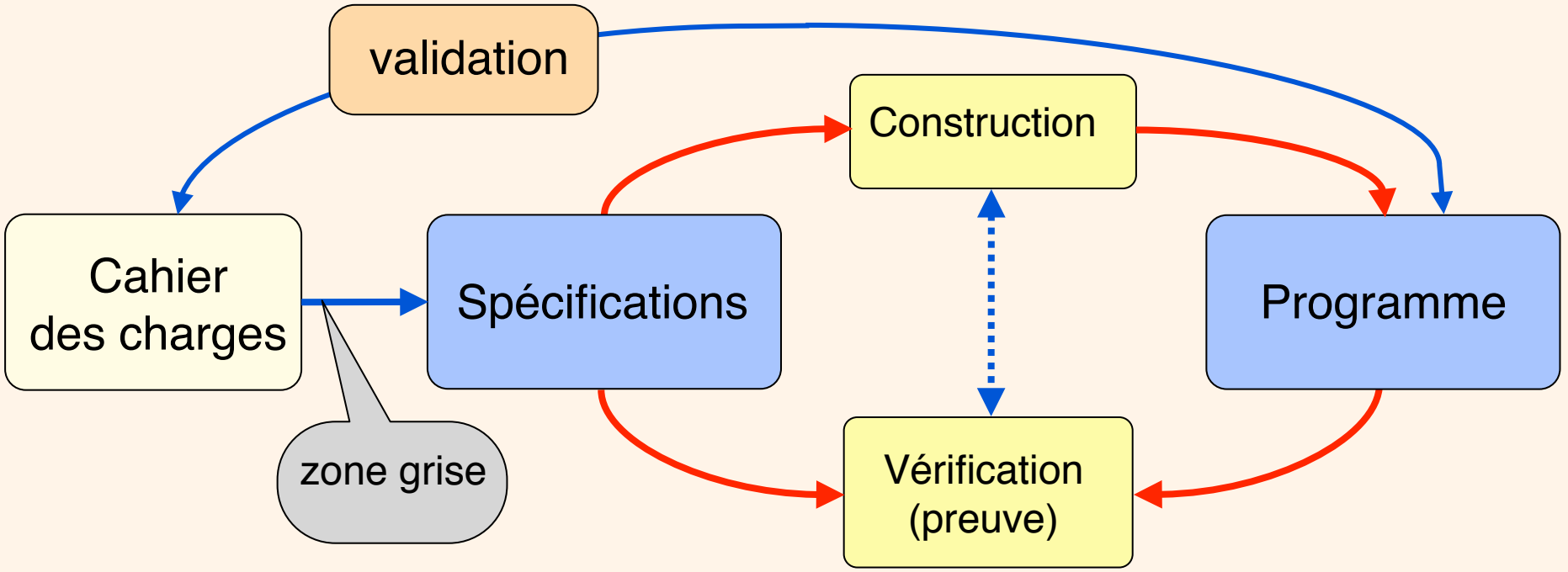
  Définir des sémantiques à divers niveaux d'approximation

  Résoudre l'équation (de point fixe) traduisant une propriété

- Difficultés communes

  Explosion des états

Spécification, construction, preuve



- Ingrédients de la spécification
  - Une base formelle (logique)
  - Un langage d'expression
  - Des outils d'aide à la construction
  - Des outils d'aide à la preuve

Pendant longtemps :
- pas de base formelle
- langage naturel
- pas ou peu d'outils

Une piste prometteuse :
les assistants de preuves

Perspectives du génie logiciel

- > 20 ans après 1995…
  - *No Silver Bullet !*
  - Des défis persistants
    - sécurité
    - tolérance aux fautes
    - adaptation

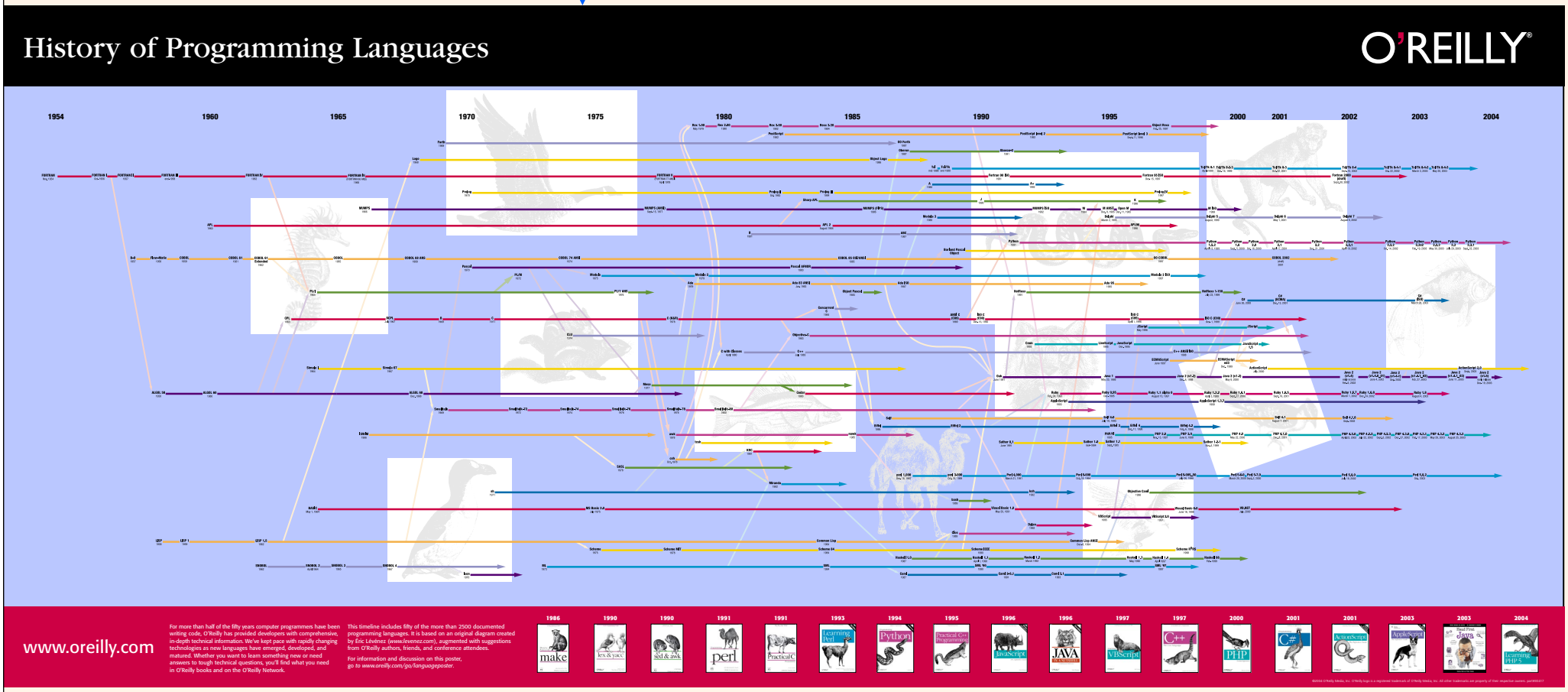
Nouveaux modes de construction de logiciel

«La cathédrale et le bazar»
L'*open source*

- La disjonction persiste entre théorie et pratique
  - Des avancées dans les pratiques, avec peu de bases théoriques
  - Des avancées dans la théorie, mais peu d'impact pratique à court ou moyen terme
  - Quelques espoirs tout de même
- Vers une refondation ?

SEMAT
Software Engineering Method And Theory

Les langages de programmation



© O'Reilly

Nouveaux paradigmes

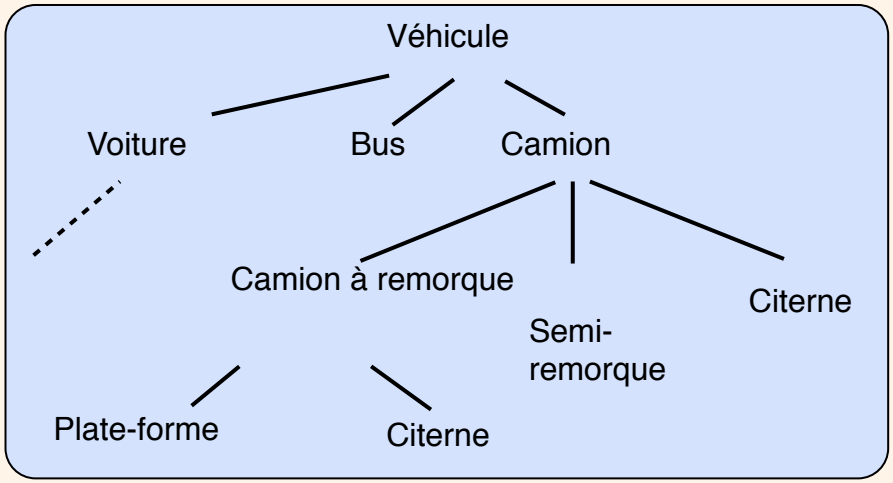
✤ Langages à objets : Modéliser le monde réel

Simula (Dahl - Nygaard, 1967)

- regrouper données-fonctions d'accès
- objets, classes, héritage

Smalltalk (Kay et al., 1978-80)

- métaclasses, polymorphisme
- machine virtuelle à objets
- environnement de programmation



✤ Langages logiques : Modéliser le raisonnement

Représentation des connaissances
(McCarthy, Minsky, Hewitt, années 1960)

- déclaratif vs procédural

Prolog (Colmerauer - Roussel - Kowalski, 1972)

- faits et règles, moteur d'inférence

Programmation par contraintes

Pierre est fils de Paul ; Julie est fille de Paul
Marie est fille de Pierre ; Cécile est fille de Pierre
Émile est fils de Julie
A enfant de B = A fils de B **ou** A fille de B
A cousin germain de B = A enfant de X
et B enfant de Y **et** X enfant de Z
et Y enfant de Z **et** X différent de Y
Quels sont les cousins germains d'Émile ?
--> Marie, Cécile

Retour vers la machine

Motivation

programmer le logiciel de base (compilateurs, systèmes d'exploitation),
dépendant des caractéristiques de la machine

Première approche : l'assembleur déguisé

Exemple : PL360 (Wirth, 1968) et langages analogues (LP10070, etc.)

Visibilité des registres, constructions de haut niveau (IF, WHILE, etc.)

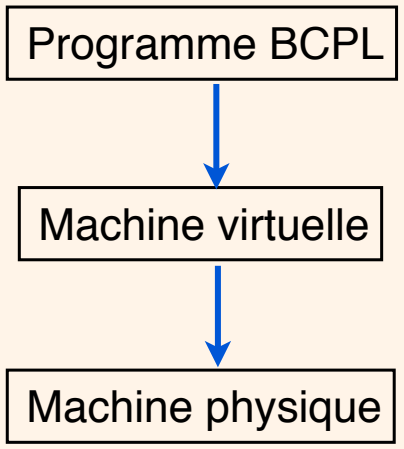
Procédures

Deuxième approche : la machine virtuelle

BCPL (Martin Richards, Cambridge, 1967)

Structure de données unique : le mot

Le prédécesseur de C



Les langages de script

Motivation et évolution

Objectif : langages simples, faciles à apprendre, peu contraignants, d'exécution rapide ; programmes interactifs, facilement modifiables, intégrables dans une application

Initialement : langage de commande, organisation des tâches (ex : *shell* d'Unix)

Plus tard : manipulation de texte (ex : awk, Perl)

Actuellement : langages dynamiques (ex : tcl, Python)

Caractéristiques

En général, langages **interprétés** (mais parfois compilation possible)

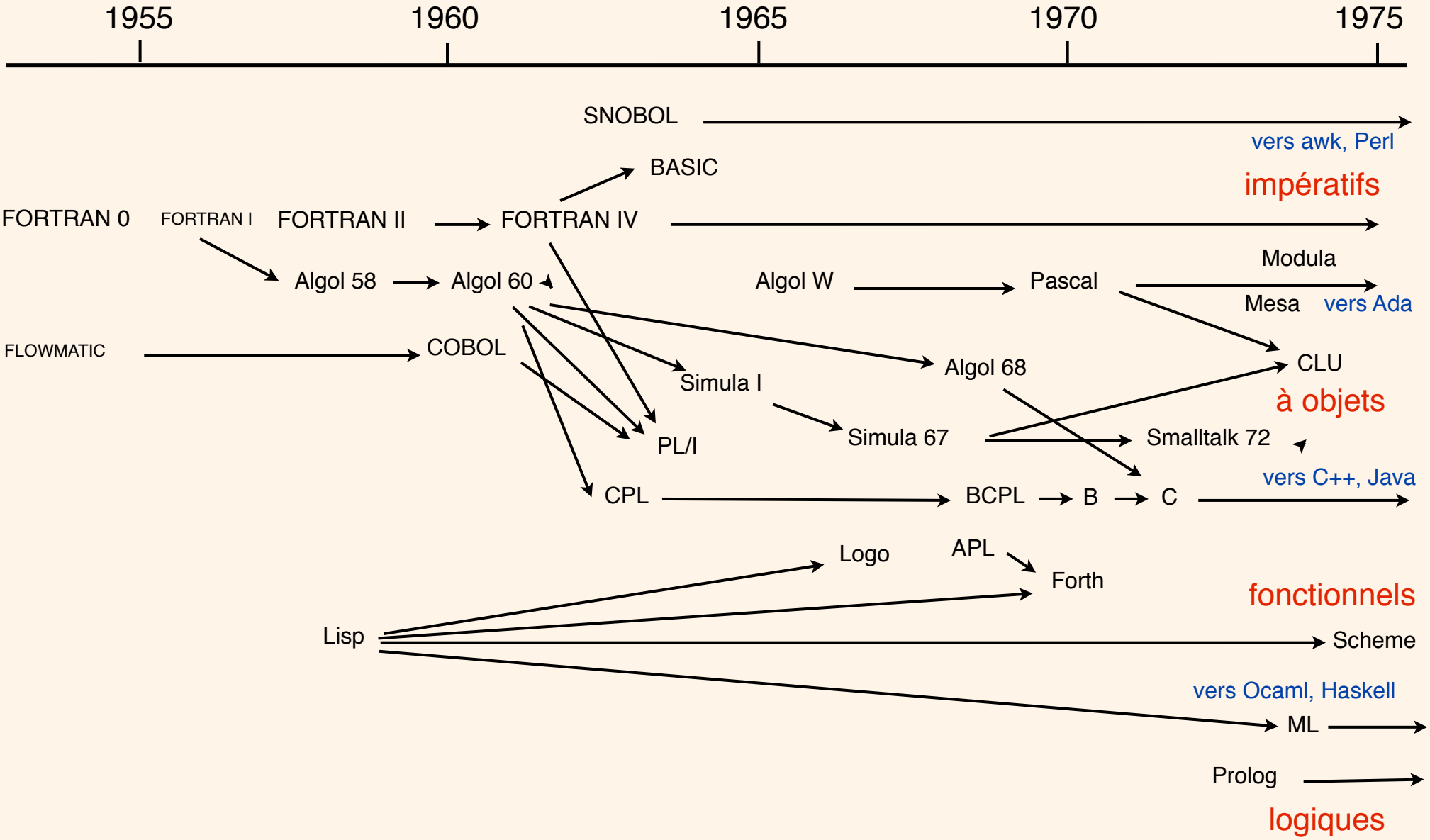
Peuvent être spécifiques à un domaine, ou généraux

Spécifiques : Perl, bsh, tcsh, postscript, Javascript, XSLT

Généraux : tcl, Python

Langages de programmation

Les 20 premières années



Donnée, information, connaissance

- **Donnée**

  Description élémentaire d'une réalité

  Peut prendre des formes diverses selon son type

  nombre, chaîne de caractères, image, etc.

- **Information**

  Donnée ou ensemble de données interprétées (combinées, mises en contexte)

- **Connaissance**

  Information dotée d'un sens (dans un univers logique), permettant de modéliser la réalité, de guider une action

Les frontières entre ces notions sont souvent floues…

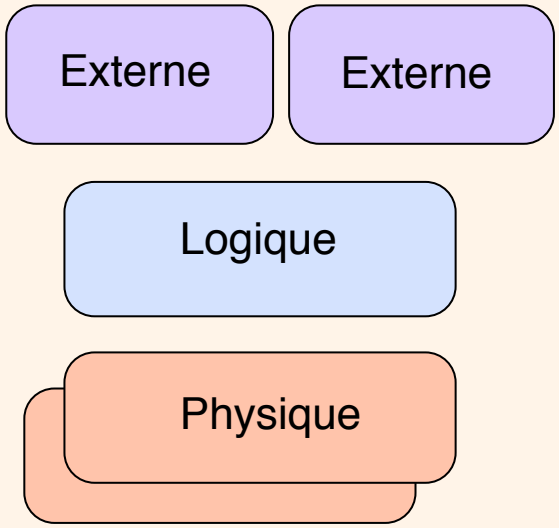
De quoi parle-t'on ?

✤ Qu'est-ce qu'une base de données ?

Un ensemble de données organisé en vue de sa conservation, de sa mise à jour et de sa consultation

Des opérations sur une base de données permettent d'en extraire des informations, voire des connaissances

✤ Les trois aspects des bases de données



Externe : définit différentes «vues» sur les données, notamment selon divers niveaux d'abstraction

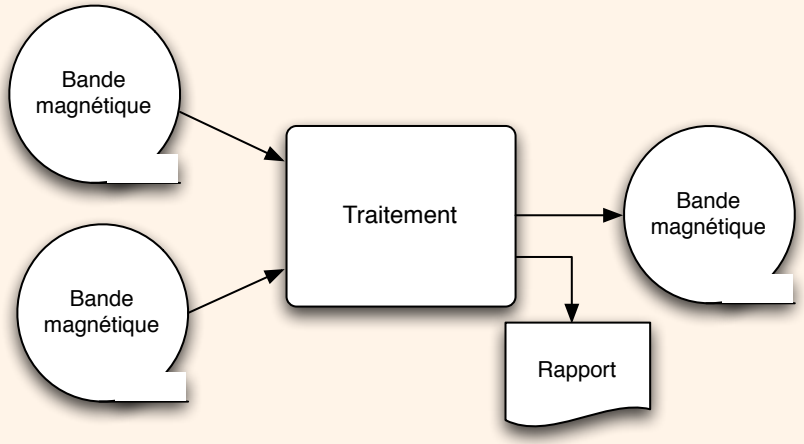
Logique : définit l'organisation des données et les opérations que l'on peut réaliser sur ces données

Physique : définit le mode de représentation et de conservation des données et la mise en œuvre des opérations

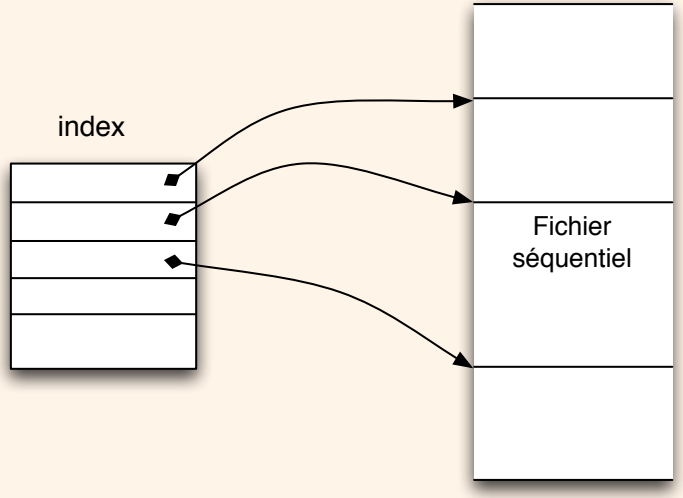
Un aspect supplémentaire : Dynamique : définit les modalités d'exécution d'actions complexes pour respecter des critères de qualité (sécurité, tolérance aux fautes, etc.)

La préhistoire

- À l'origine….
  - Bandes magnétiques (1950)
    - donc fichiers séquentiels
  - Exemples de traitement
    - Tri, fusion
    - Mise à jour
    - Recherche et extraction
  - Avancées
    - Génération automatique des programmes (RPG)



- Plus tard…
  - Disques magnétiques (1960-62)
    - Accès direct
  - Avancées
    - Accélération de l'accès
    - Hachage ou indexation



Exemple de traitement de fichiers

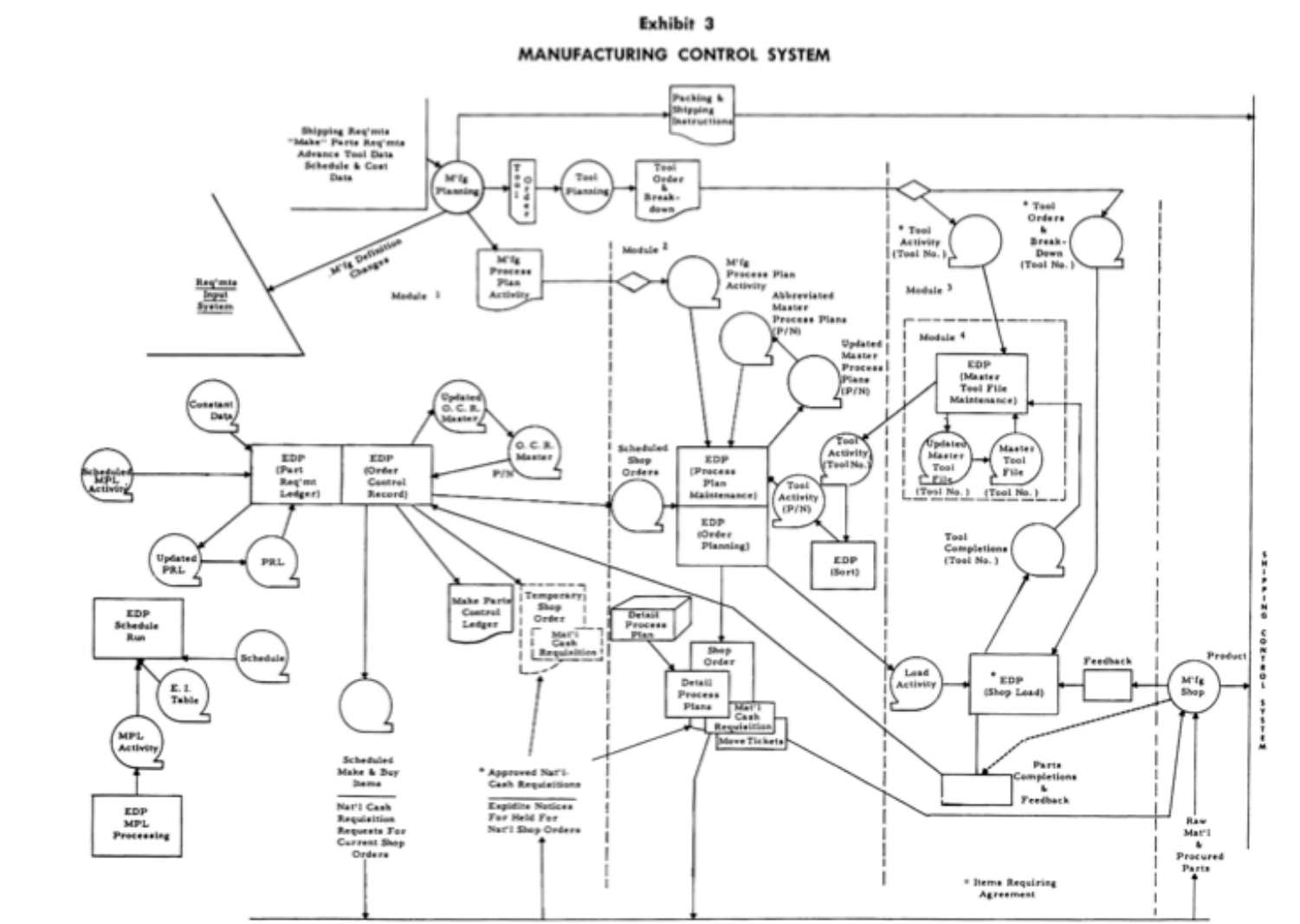


Figure 1. Part of a file-based system for manufacturing control, circa 1962. The system would have involved many runs, evidenced by the large number of programs and intermediate files involved.[7]

Extrait de : A. D. Meacham and V. B. Thompson, eds. *Total Systems*, American Data Processing, 1962, p. 153

Les premiers modèles

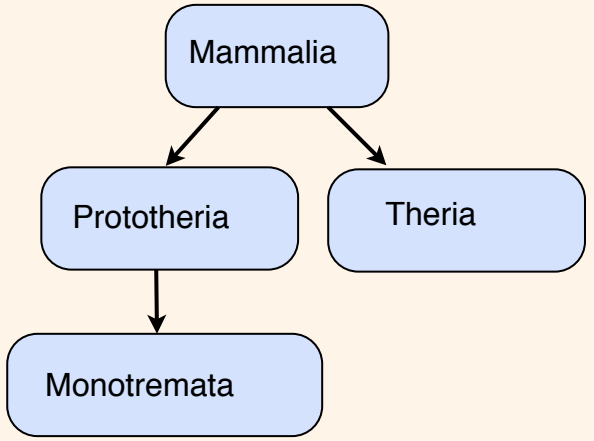
Le modèle hiérarchique

Une organisation simple
mais peu adaptable

Supprimer un nœud supprime
sa descendance

Le premier SGBD d'IBM (IMS, 1966)

Toujours utilisé (notamment avec XML)



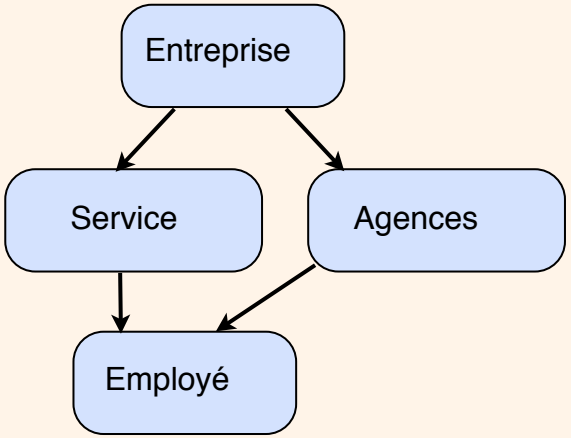
Le modèle réseau

Lève des limitations du modèle hiérarchique

Organisation plus souple…

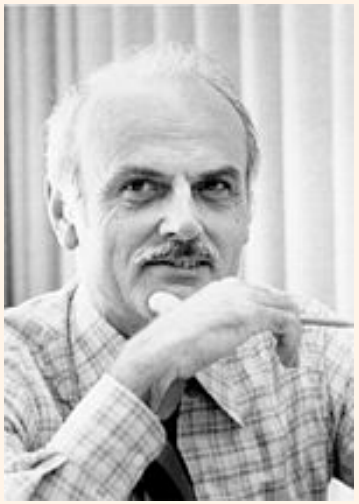
… mais gestion complexe des pointeurs

Le premier SGBD de l'histoire (IDS, 1962)



Le modèle relationnel (1)

- Un nouveau modèle de données
  - Edgar Codd (IBM, 1970)
  - Fondé sur des tables (relations) et des clés
- Des avantages
  - Une base mathématique : l'algèbre relationnelle
  - L'indépendance des données
- Un défi : l'efficacité



Edgar Codd
(1923-2003)

Source : Wikipedia, fair use

Nom	Prénom	Lieu	Service	clé	…
Martin	Jeanne	5	Études	11	
Duval	Jacques	3	Après vente	12	
Bernard	Pierre	4	Commercial	13	
Lefèvre	Paule	1	Après vente	15	
…	…				

Lieu	Ville	Adresse	…
1	Bordeaux	…	
3	Paris	Centre	
4	Paris	Défense	
5	Grenoble	…	

Le modèle relationnel (2)

- ## Le calcul relationnel

  Base : logique du premier ordre

  Une requête :

$$\exists \text{ nom, prénom, ville (Employés (nom, prénom, «Commercial», lieu)} \wedge \text{Localisation (lieu, ville))} ?$$

- ## Une traduction en SQL (cf plus loin)

```
SELECT (Nom, Prénom, Ville)
      FROM (Employés, Localisation)
WHERE (Employés.lieu)=(Localisation.lieu)
     AND Service=«Commercial»
```

Les premiers SGBDR commerciaux

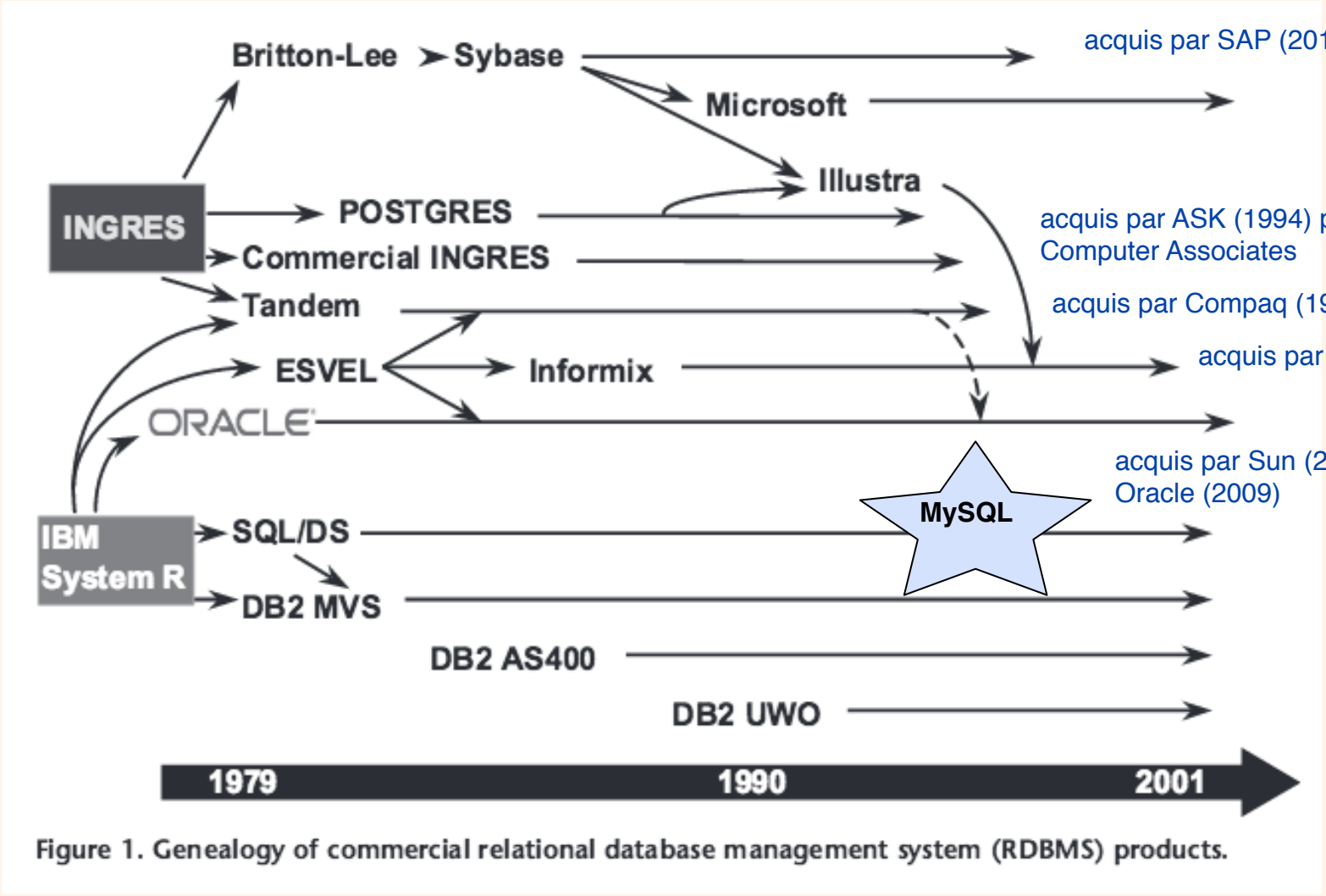


Figure 1. Genealogy of commercial relational database management system (RDBMS) products.

Extrait de : Andrew Mendelsohn. The Oracle Story: 1984-2001, *IEEE Annals of the History of Computing*, vol. 35, No 2, April-June 2013, pp. 10-23. By permission of IEEE for nonprofit educational use.

Le Web, l'ultime base de données ?

- Indexer le Web ?
  - Des dizaines de milliards de pages...
  - Deux problèmes
    - la taille de l'index
    - la charge des serveurs
- Deux solutions
  - L'index réparti entre les machines
    - par une technique de hachage
  - Le parallélisme massif («embarassant»)
    - pour accélérer les recherches
    - pour résister à l'accroissement de la charge
      - exemple : MapReduce
- La prochaine étape (en cours)
  - Des données aux connaissances : le Web sémantique

- Compléments non présentés

Les avancées de la théorie (1)

- Objectif : définir une sémantique des programmes

  Pour répondre à la question «mon programme fait-il bien ce que je lui demande ?»

  Plus ambitieux : «peut-on prouver que mon programme fait bien ce que je lui demande ?»

- Robert Floyd (1967)

  Assigning Meaning to Programs

  Le programme comme transformateur de l'état

  Des assertions comme propriétés de l'état

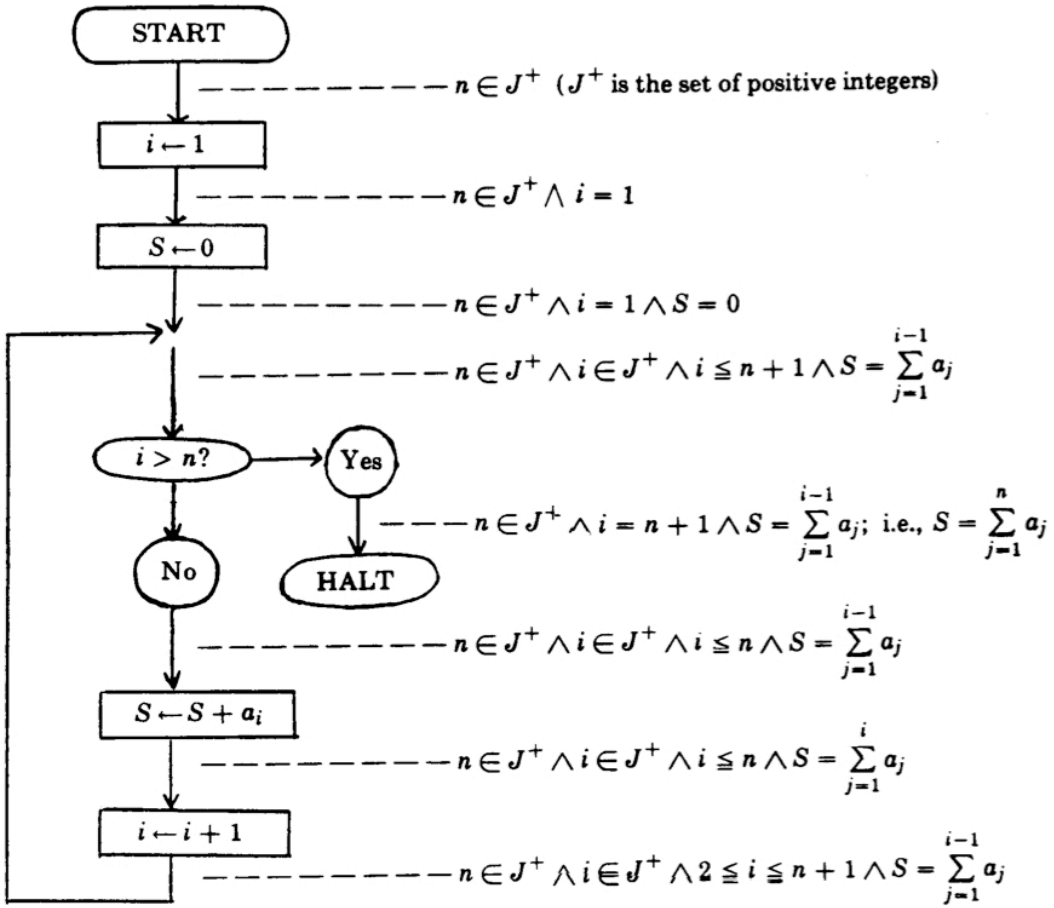


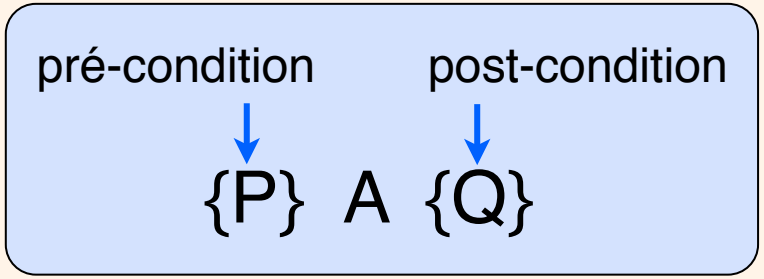
FIGURE 1. Flowchart of program to compute $S = \sum_{j=1}^{n} a_j$ $(n \geqq 0)$

Les avancées de la théorie (2)

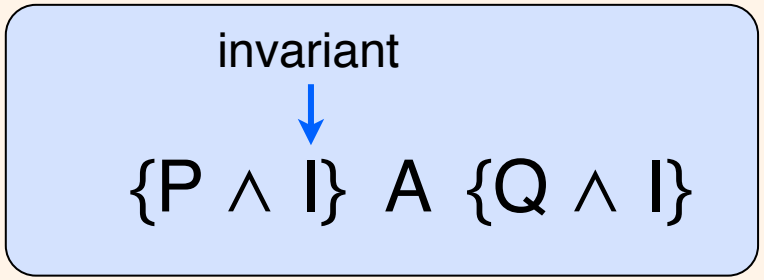
- La logique de Hoare (1969)

  Étend le travail de Floyd

  Construction de base →



  Construction étendue →



  Des règles d'inférences
  pour les compositions usuelles :
  - séquence
  - instruction conditionnelle
  - itérations

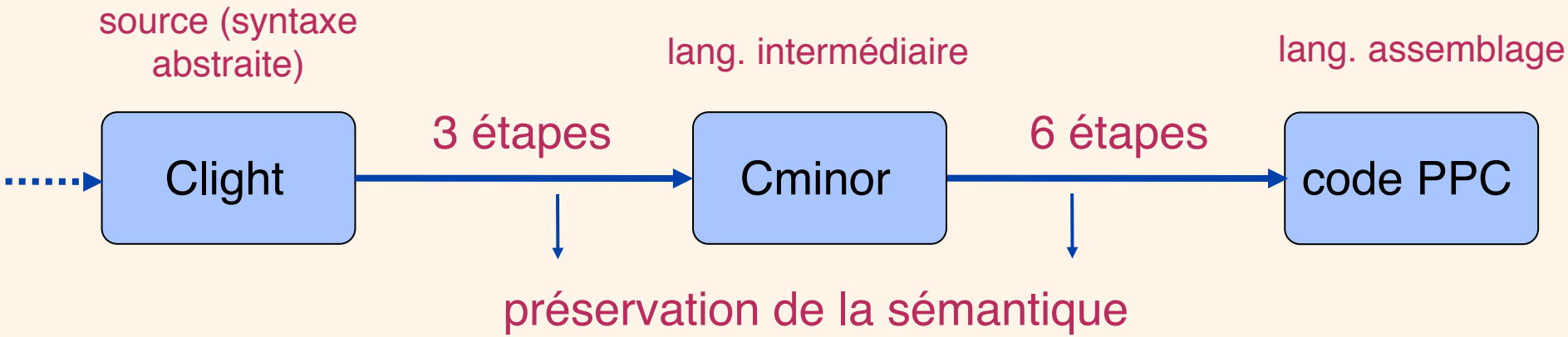
  Terminaison de A
  - preuve séparée
  - «variant» décroissant avec plancher

C. A. R. «Tony» Hoare
(1934 -)

by Rama, Wikimedia CommonsCC-BY-SA 2.0

Vérification d'un compilateur

- Un compilateur pour un large sous-ensemble de C



- Preuve et construction (avec Coq) pour chaque étape
  - Preuve que le comportement du programme objet est identique à celui du programme source (si pas d'erreur)
  - Génération d'un programme OCaml traduisant les spécifications
  - L'efficacité du code produit est très acceptable (-12% / gcc-02)

Xavier Leroy, "Formal Verification of a Realistic Compiler", *Comm. ACM,* 52:7, July 2009