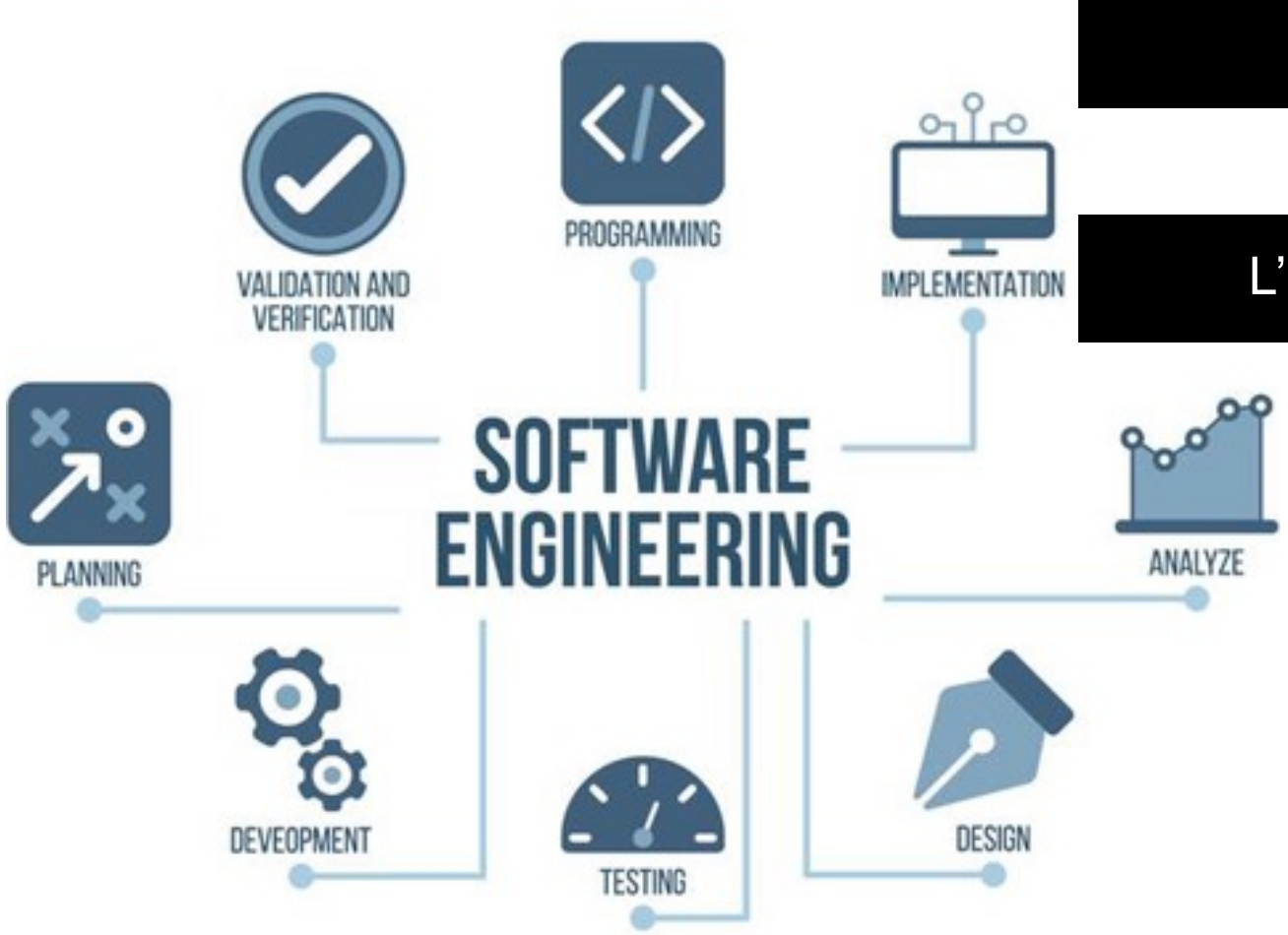
La complexité des logiciels

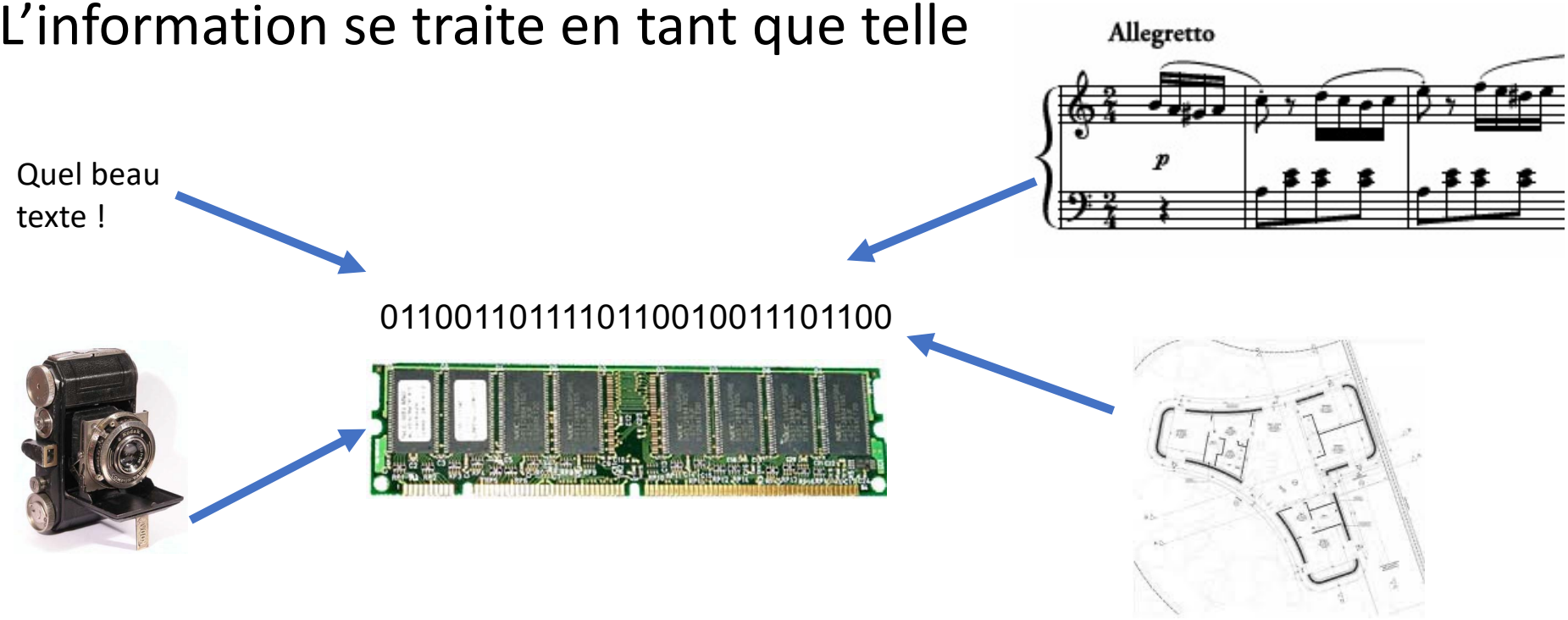


L’importance du génie logiciel

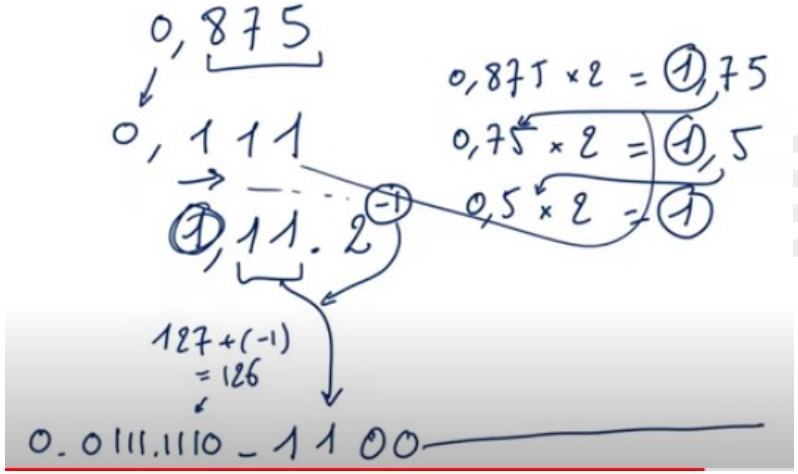
Claude Jard

La numérisation du monde…

- Maintenant : indépendance et convergence
- L'information se traite en tant que telle



La numérisation du monde…
Codage binaire

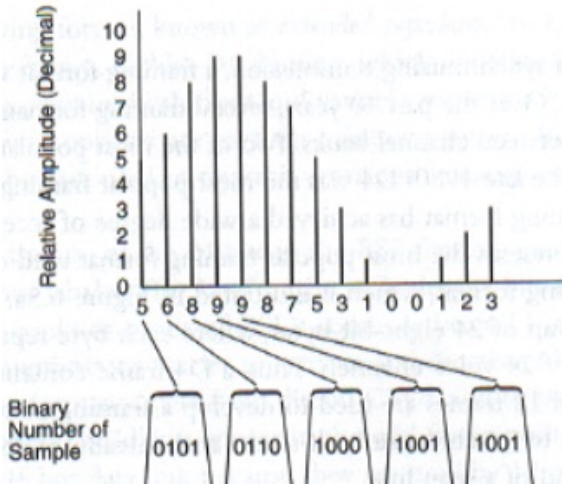
- 2023 -> 11111100110 (1+2+4+32+64+128+256+512+1024)
- 

Perte de précision possible

- Caractères : OK -> 01001111 01001011 (ASCII)
- Mots : point dans un espace à de multiples dimensions (permet de capturer une notion de proximité)

La numérisation du monde…
Codage binaire

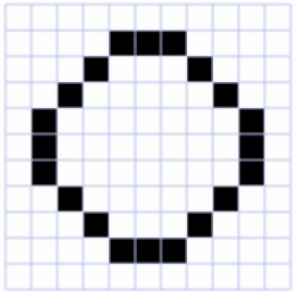
- Son



Bruit d'échantillonnage & de quantification

Connaissance du décodage

- Image



0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0	0
0	0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	0	0
0	0	0	1	0	0	0	1	0	0	0
0	0	0	0	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

Codage couleur ou RVB de chaque pixel

Les ordinateurs sont partout…

- Infestation massive
- Communiquant, reliés à Internet
- De plus en plus critiques et autonomes (robots)

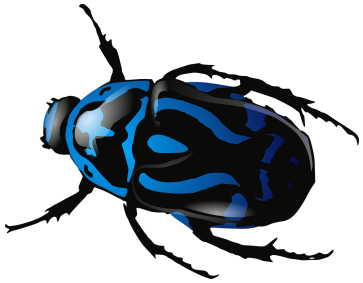
Le logiciel

- Un circuit électronique ne fait que des choses très simples
- Mais il en fait des milliards par seconde et sans erreur

- Le logiciel doit décrire quoi faire, dans tous les détails
- C’est un 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 : des millions de lignes, co-écrits par de grosses équipes

A petits bugs, grandes conséquences



- Pas une défaillance de la machine, mais défaillance du concepteur (mauvais ordre)
- Coût économique considérable
- Failles et cyber-sécurité
- Une tolérance du public étonnante, mais un enjeu économique majeur pour beaucoup d’applications à venir

Des exemples innombrables

- Plantages des ordinateurs, distributeurs bancaires, systèmes de réservation, sites Web, …
- Micro-bugs dans les navigateurs Web –> macro-attaques
- Blocages de téléphones, serveurs, …
- Prises de tête des conducteurs et des garagistes
- Crash du téléphone interurbain américain

  (une ligne mal placée sur un million)
- Explosion d'Ariane 501 (débordement arithmétique dans un code inutile), pertes de satellites
- Bug subtil dans la division flottante du Pentium

  (coût : 470 millions de dollars pour Intel)

A la chasse aux bugs
(les détecter avant mise en service)

- Utiliser des techniques rigoureuses du Génie Logiciel :
  - Documentation, revues de code, tests intensifs
  - Certification externe (e.g. avionique)
- Rendre visuel ce qui est invisible :
  - Environnements de débogage
  - Prototypes virtuels animés
- Utiliser des méthodes formelles :
  - Conception vérifiée (preuves, « model-checking »)
  - Compilateurs certifiés

En bref, pourquoi vérifier du logiciel ?

- Ecrire des programmes semble facile, mais écrire un programme correct est très difficile (spécialement lorsqu'il y a des interactions et de la communication)
- Le génie logiciel doit être une science, c'est-à-dire fondée sur des modèles

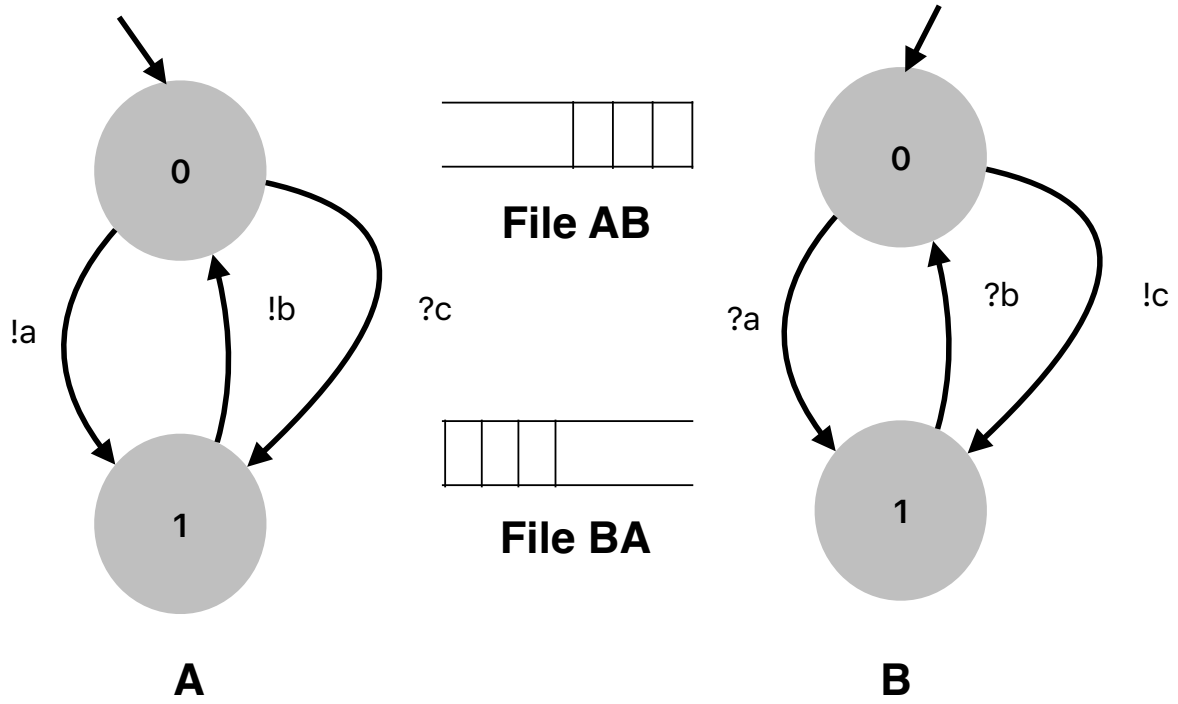
Des pionniers

- Mathematical approach towards program correctness
  [Turing, 1949]
- Syntax-based technique for sequential programs
  [Hoare, 1969]
- Syntax-based technique for concurrent programs
  [Pnueli, 1977]
- Automated verification of concurrent programs
  [Emerson, Clarke & Sifakis, 1981]

Un tout petit exemple de conception d'un système

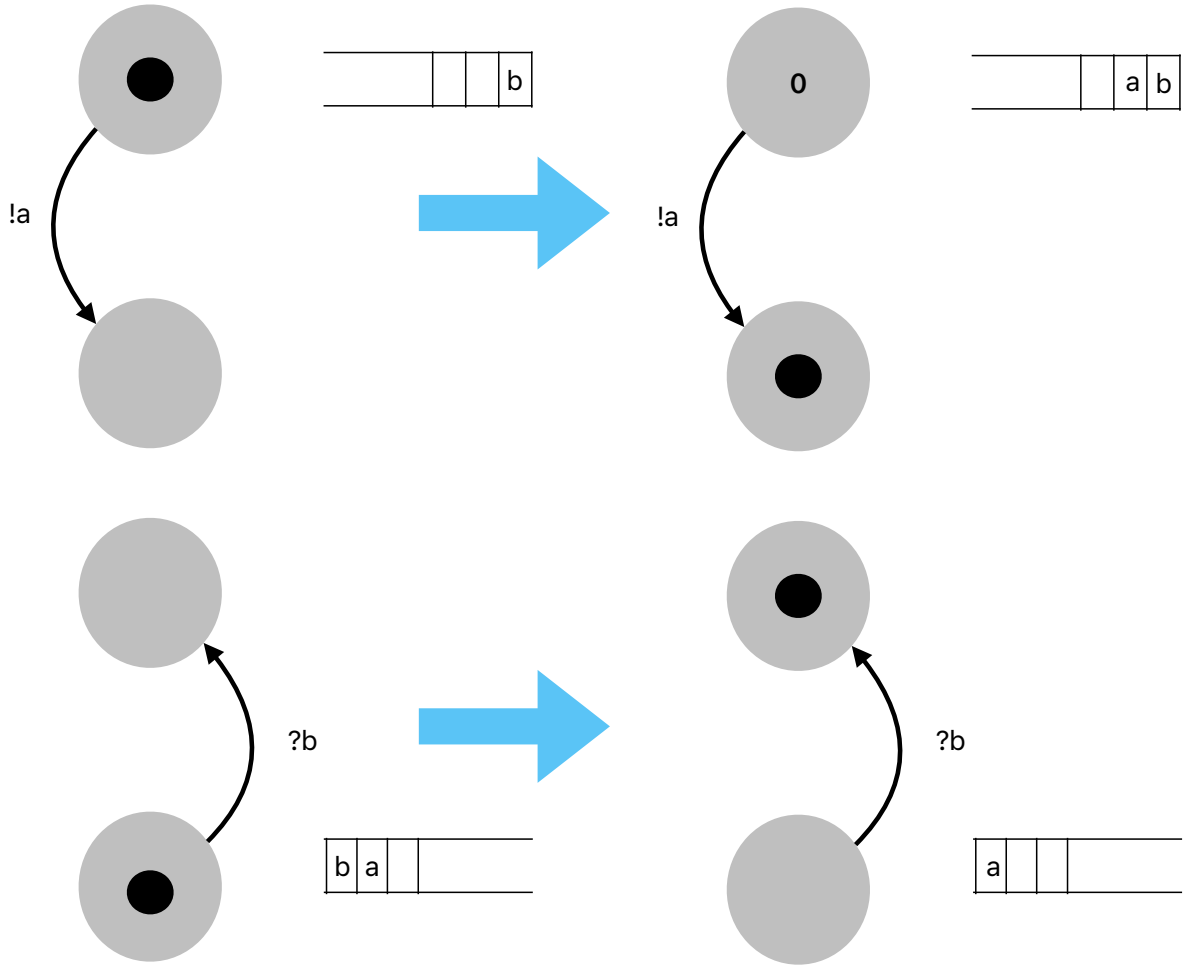
- La modélisation de l'interaction entre un utilisateur et un réveil accessible via Internet
- On va utiliser le formalisme élémentaire de deux automates communicants
- Automate :
  - Notion d'état (statique)
  - Notion de transition/action (changement dynamique d'état)

Deux automates communicants

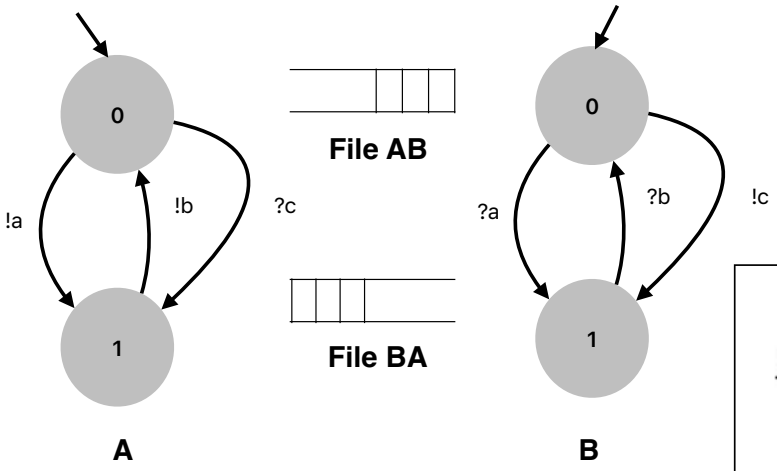


- **Automate A : utilisateur**
- **Automate B : réveil distant**
- **File AB : connexion A vers B**
- **File BA : connexion B vers A**
- **!a : envoi armement réveil**
- **!b : envoi désarmement réveil**
- **?c : réception alarme**
- **?a : réception armement**
- **?b : réception désarmement**
- **!c : envoi alarme**

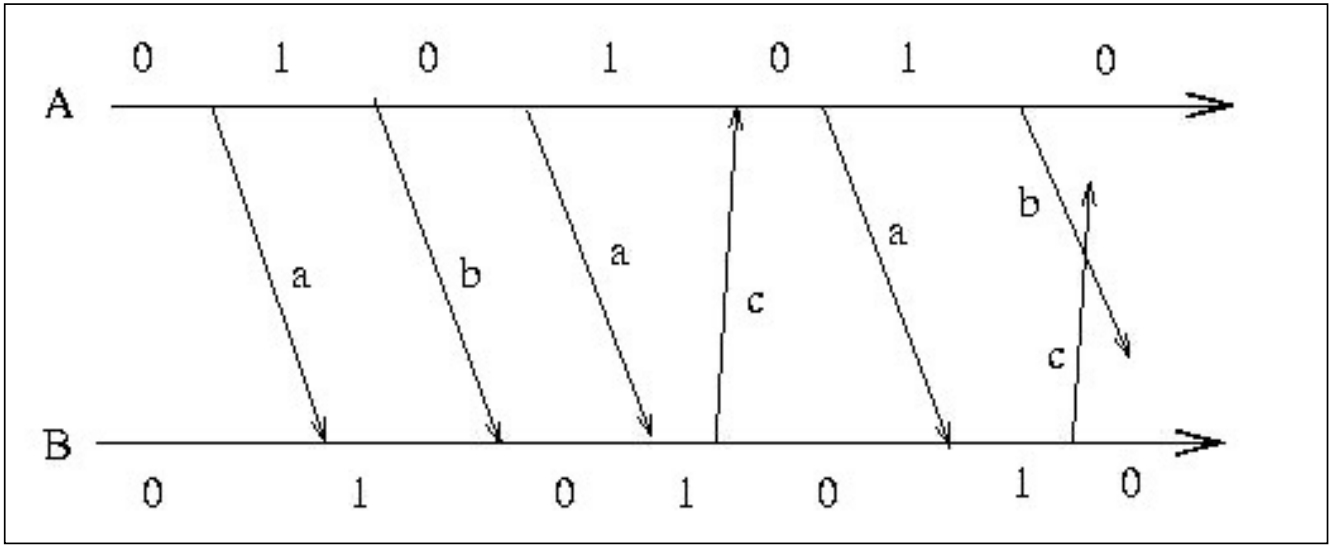
Fonctionnement du modèle (sémantique)



Exemple de comportement



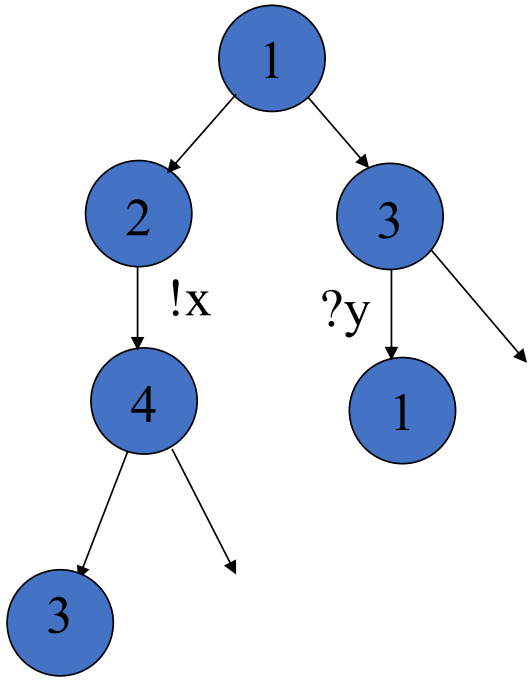
!a.?a.!b.?b.!a.?a.!c.?c.!a.?a.!b.!c



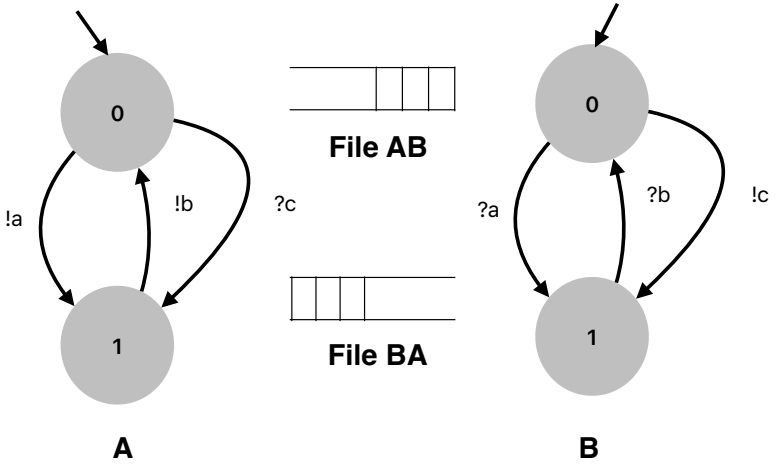
Simulation “exhaustive”

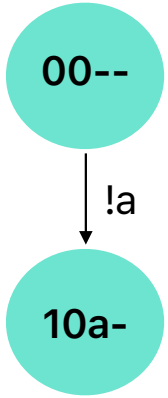
- Exploration des différents états possibles du système
- Etat = (Etat A, Etat B, Etat F_AB, Etat F_BA)

- Construit un graphe (potentiellement infini !)

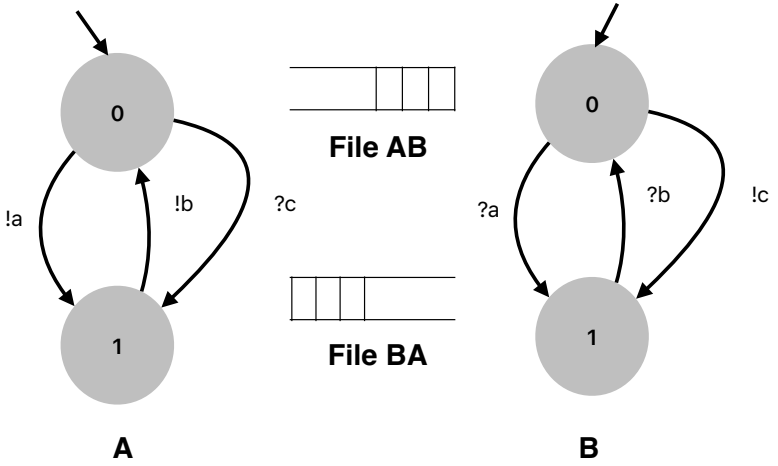


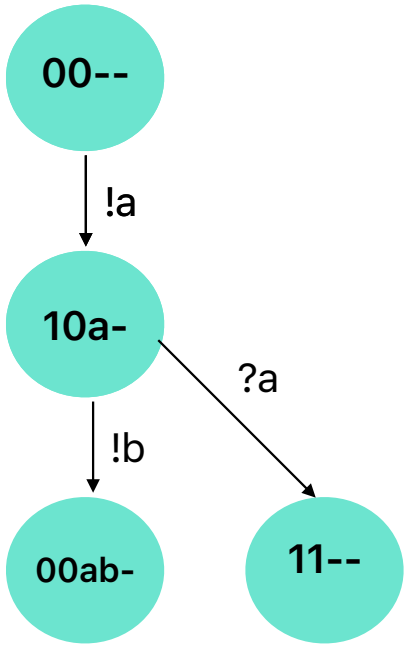
Simulation “exhaustive”



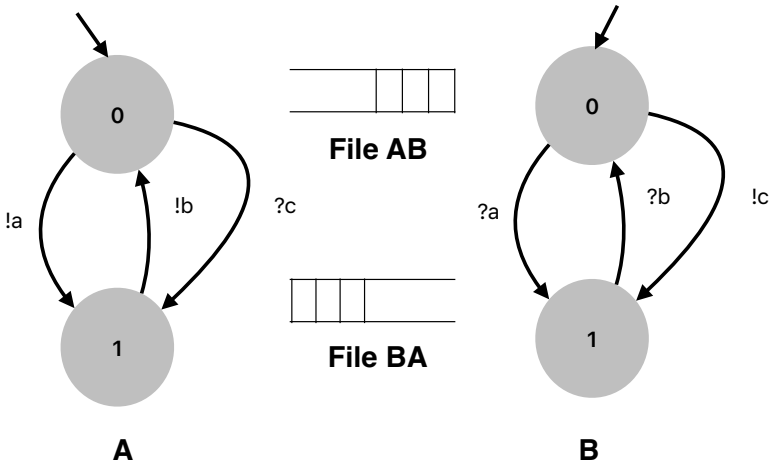


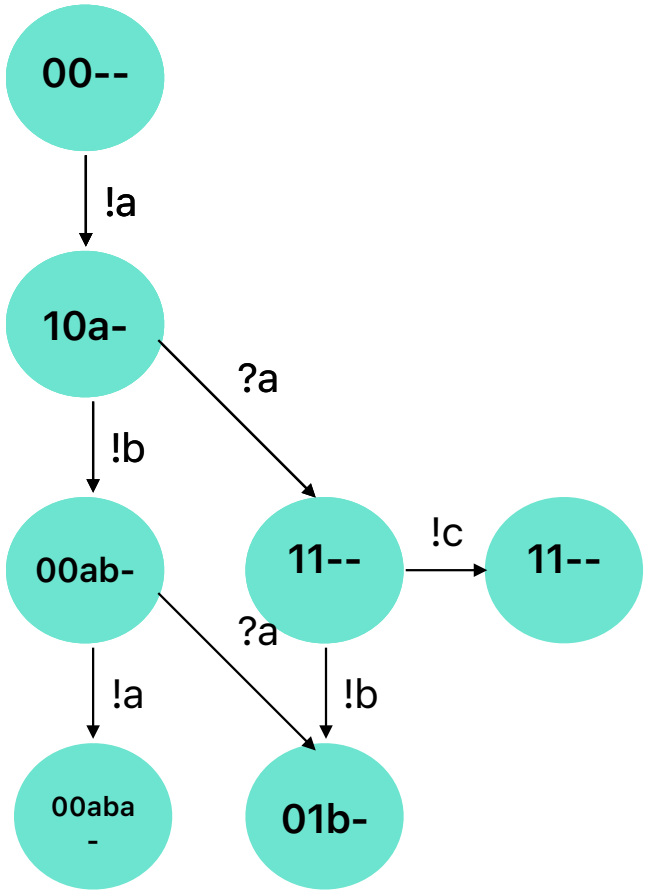
Simulation “exhaustive”



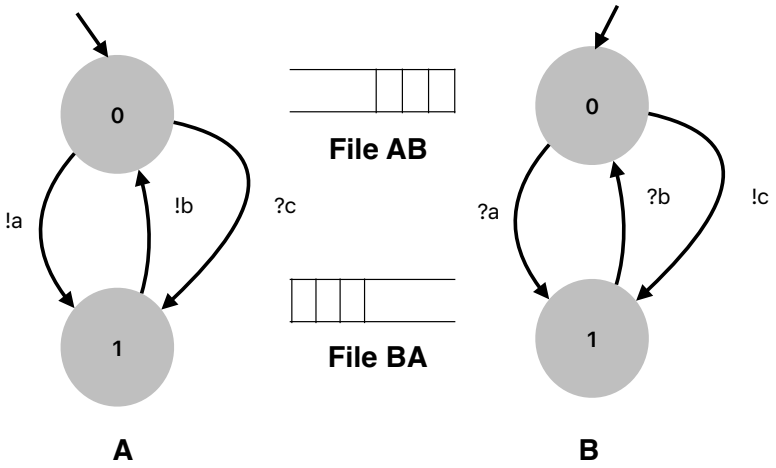


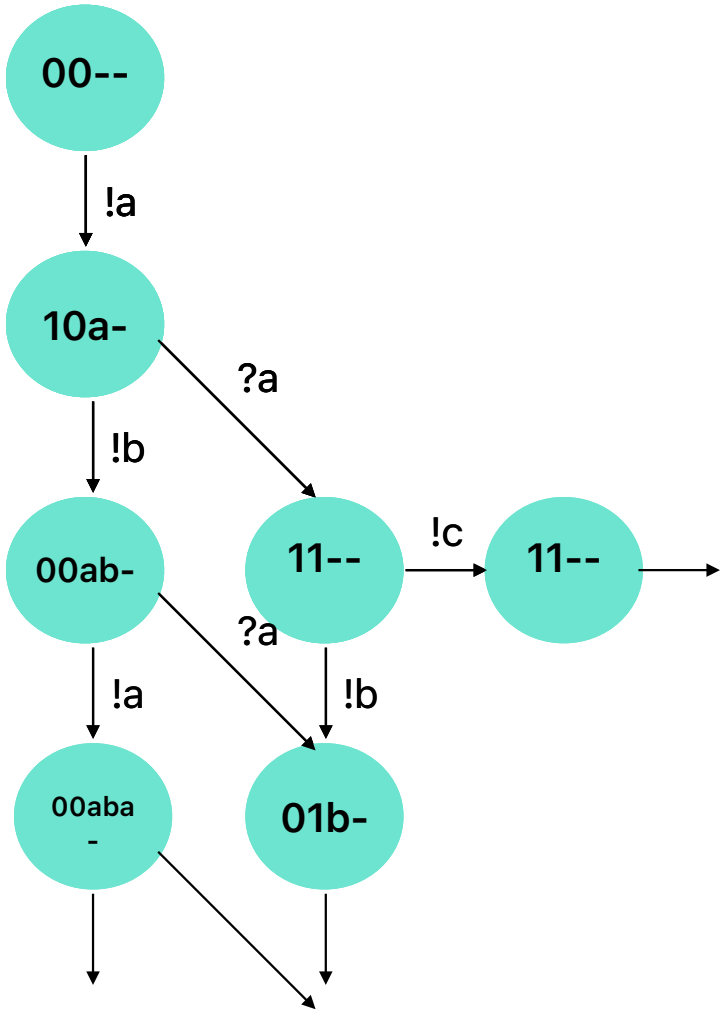
Simulation “exhaustive”





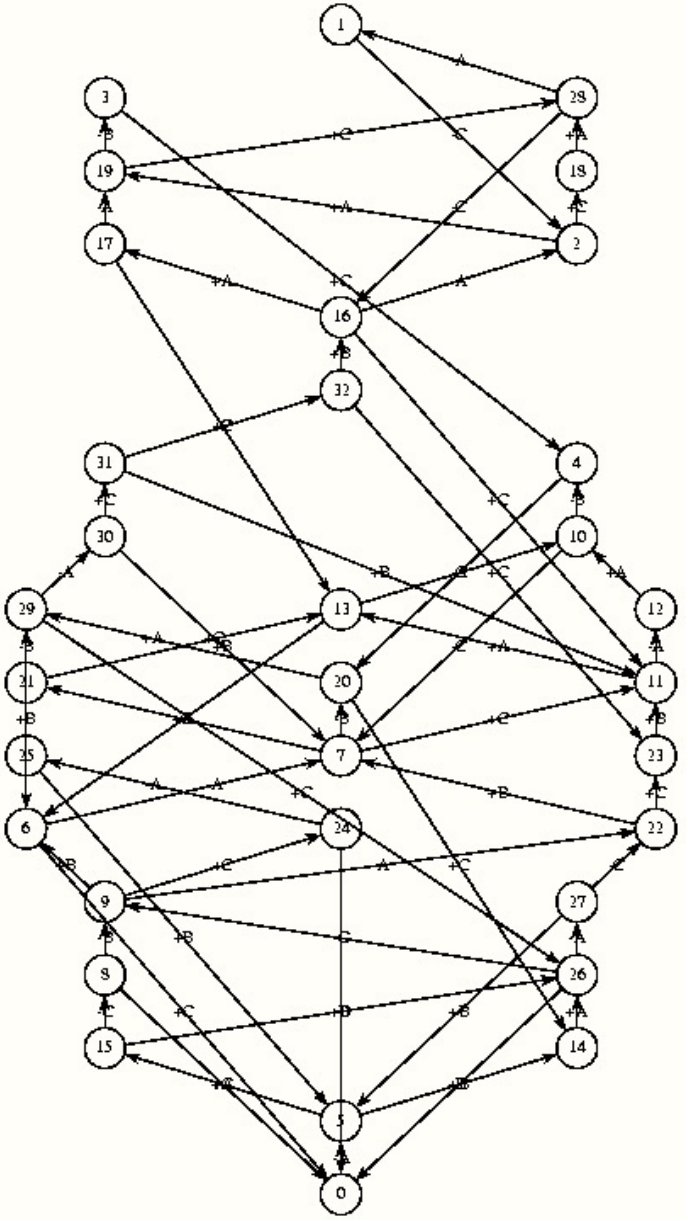
Simulation “exhaustive”





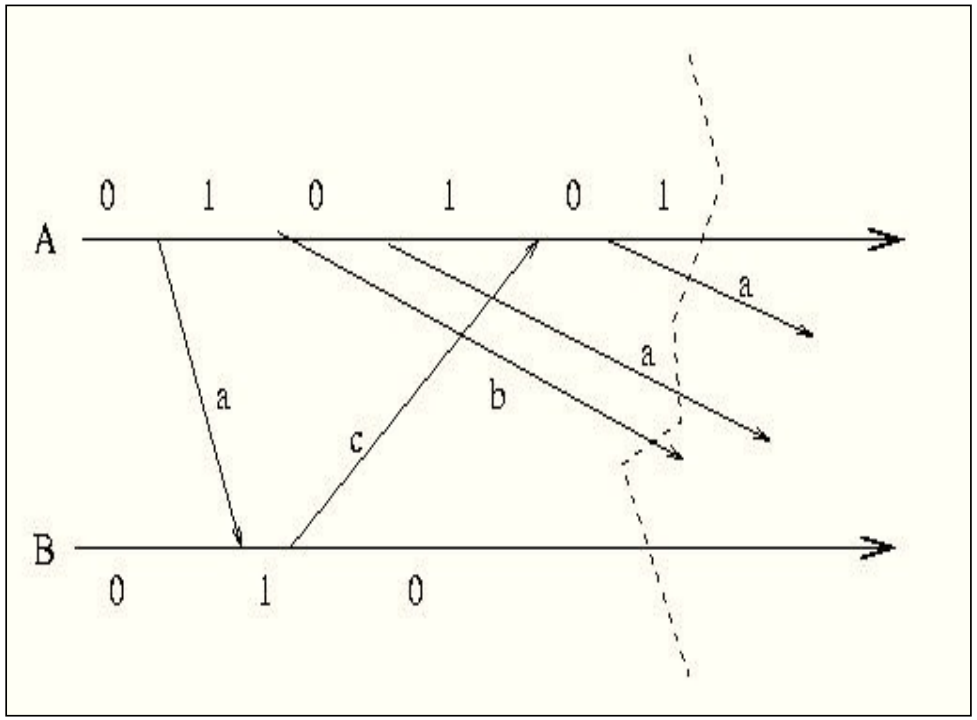
Le graphe des états

- Files limitées à 3 messages dans notre exemple
- Hypercubes = signature du parallélisme
- En vrai : graphes de plusieurs millions d'états et de transitions



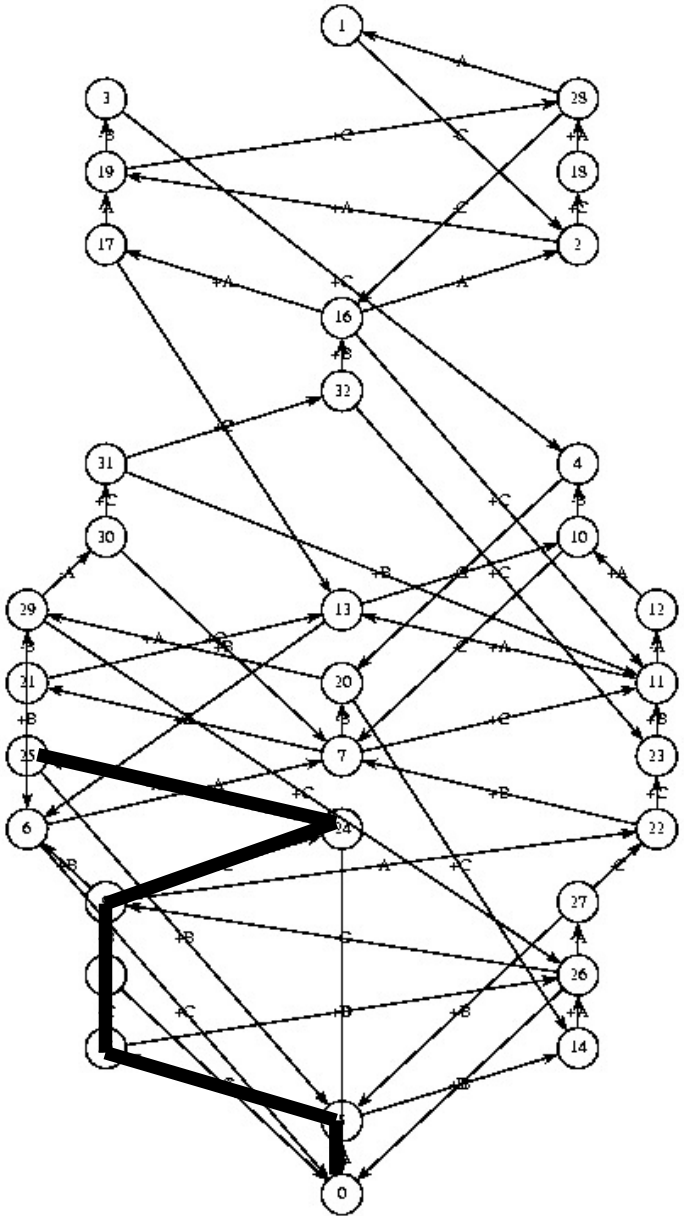
Preuve de propriétés

- Motif “aa” dans la file AB ?
- Collision de désarmements, détectable à la profondeur 7 !



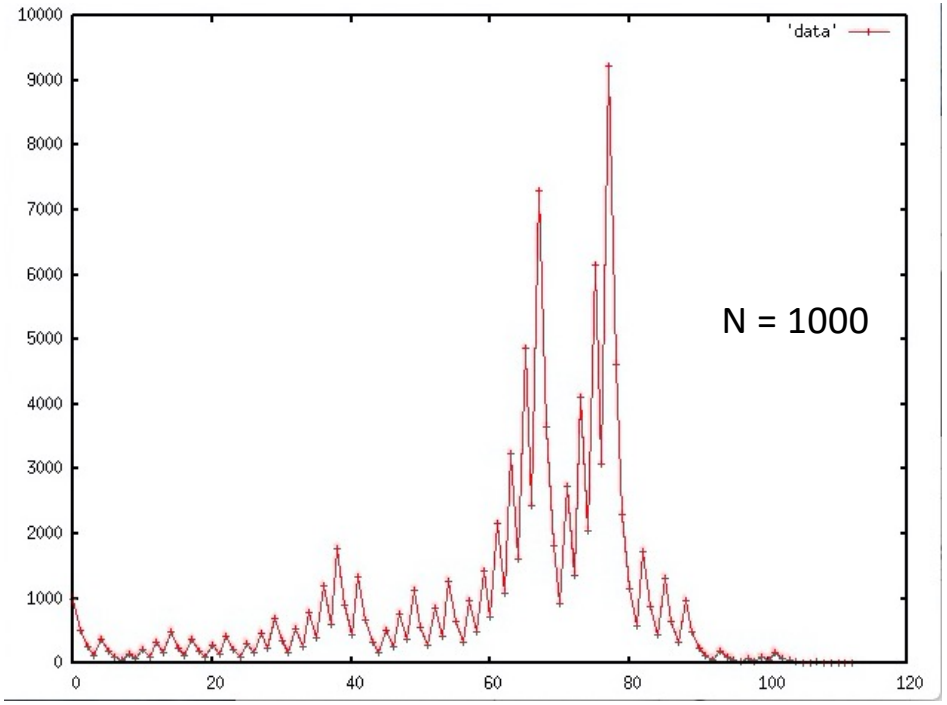
Preuve “manuelle” ou informatisée ?

- Découvert en une fraction de seconde par un algorithme de parcours de graphe



Utiliser des programmes pour calculer sur les programmes... sans se mordre la queue !

```
Lire N
Tant que N>1 faire
        si N pair alors N := N/2
        sinon N := 3xN+1
```



Cet algorithme termine-t-il quelque soit la valeur d'entrée N ?

Propriétés : sûreté & vivacité

- Les propriétés de sûreté spécifient que les mauvaises choses n'arrivent jamais
- Ne rien faire remplit bien les conditions d'une propriété de sûreté

- Il faut complémenter la sûreté par la vivacité qui demande que les bonnes choses vont arriver
- Pour violer la vivacité, il faut un temps infini ; Violer la sûreté en un temps fini est possible

Sur notre exemple

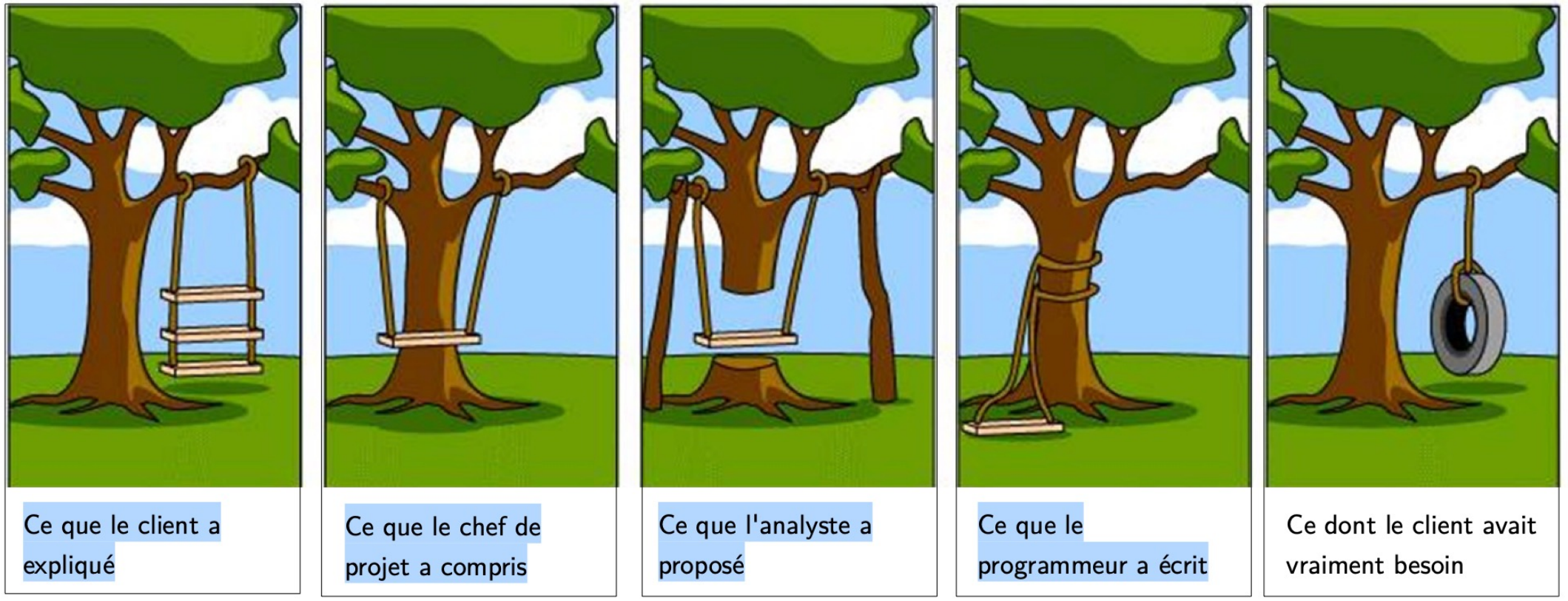
- Sûreté : A ne reçoit pas d'alarme dans l'état 0 (réveillé)
- Vivacité : Après avoir armé le réveil et avant de le désarmer, une alarme doit être reçue

Le génie logiciel dans son ensemble

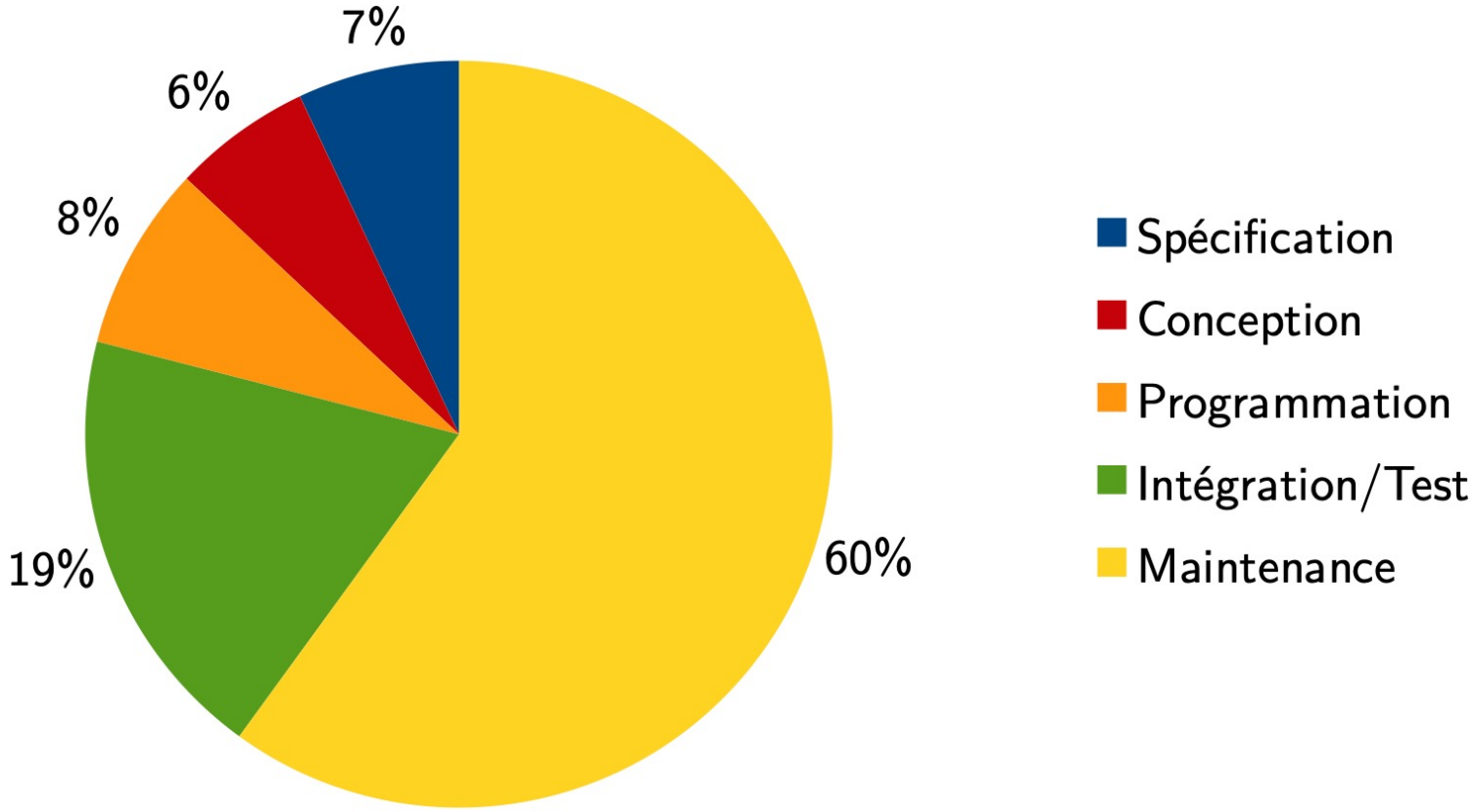
Objectif : Avoir des procédures systématiques, rigoureuses et mesurables pour des logiciels afin que :

- la spécification corresponde aux besoins réels du client
- le logiciel respecte sa spécification
- les délais et les coûts alloués à la réalisation soient respectés

Un constat décevant



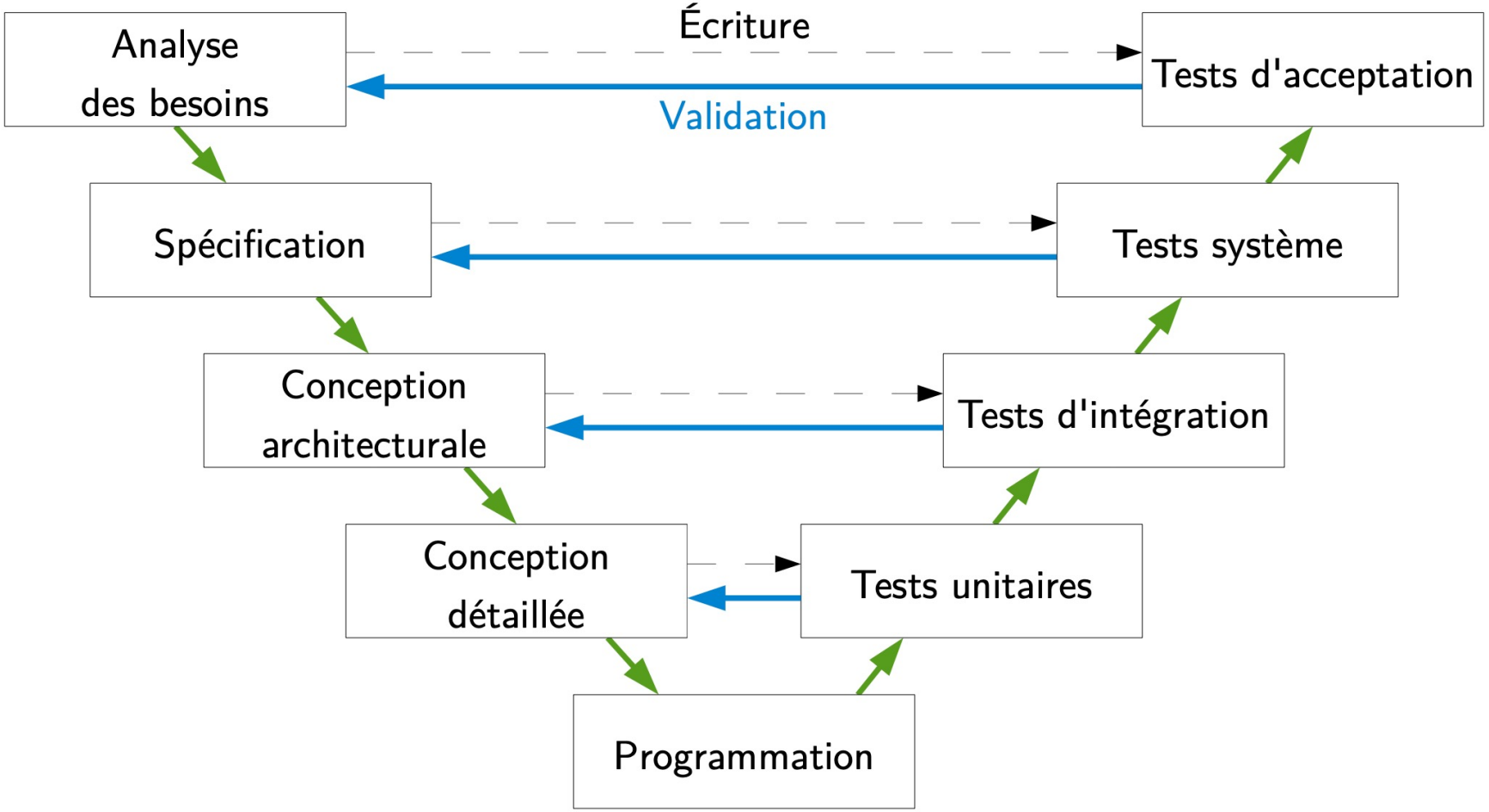
Un effort souvent mal réparti



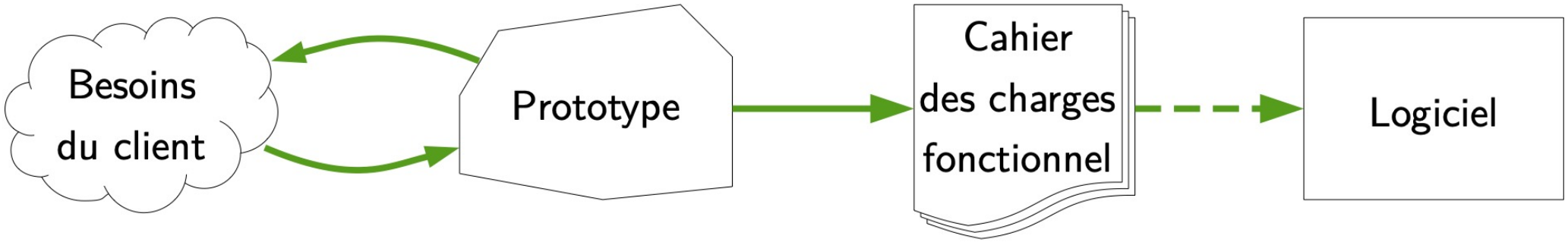
Activité du génie logiciel

- Analyse des besoins
- Spécification
- Conception
- Programmation
- Validation et vérification
- Livraison
- Maintenance

Le processus en « V »



Le prototypage



Méthodes agiles et « extreme programming »

- Implication constante du client
- Programmation en binôme (revue de code permanente)
- Développement dirigé par les tests
- Cycles de développement rapides pour réagir aux changements

En conclusion…

- Le logiciel est un objet complexe (facile à écrire, mais difficile à rendre correct)
- Le coût du bug est faramineux dans certains domaines
- La situation se dégrade avec les systèmes critiques autonomes
- Variété de méthodes pour un véritable génie logiciel qui doit être encouragé