Quelques défis actuels de l'informatique

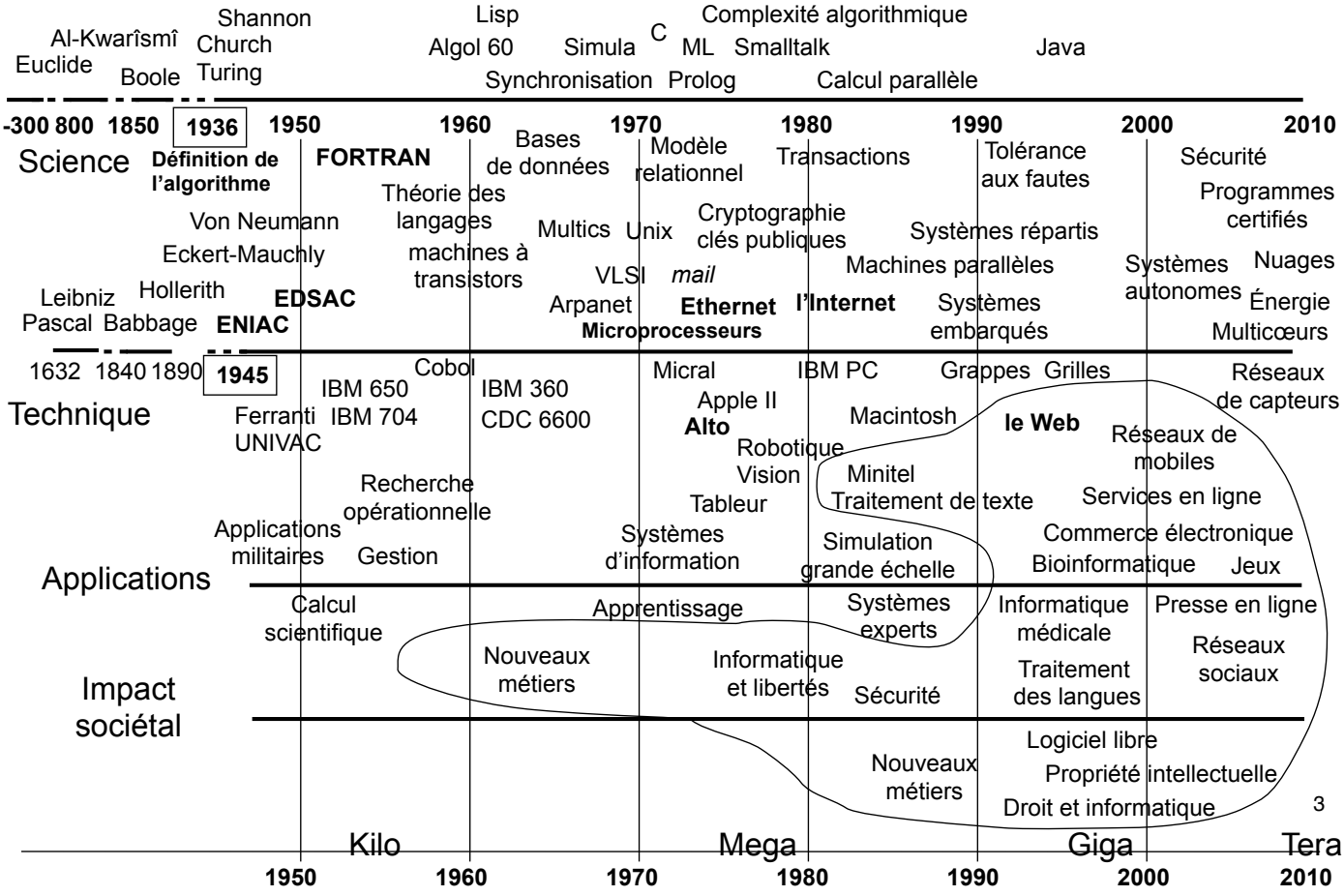
Sacha Krakowiak
Université de Grenoble
proton.inrialpes.fr/~krakowia/

C'est quoi, l'informatique ?

- Quatre facettes
  - Une science
    - Science de l'artificiel …
      - … mais pas seulement
  - Une technique et une industrie
    - Matériel, logiciel, services
  - Des applications
    - Dont le champ est croissant
  - Un impact sociétal
- Quatre concepts
- Une méthode
  - La modélisation et l'abstraction
  - Un va-et-vient théorie-pratique
  - Des outils issus de l'informatique

- Information
  - Un réducteur d'incertitude
  - Une représentation codée
  - La base de la communication
- Algorithme
  - La notion clé !
    - d'Euclide à Turing
- Machine
  - Réelle ou virtuelle
- Langage
  - Exprime un algorithme
    - pour une machine

Une brève histoire de l'informatique



Quelques grands défis de l'informatique

- Les problèmes choisis
  - Produire des programmes corrects
  - Maîtriser le parallélisme
  - Faire face à l'imprévu
    - Tolérance aux fautes
    - Sécurité
- Pourquoi ce choix ?
  - Ce sont des défis *anciens* …
    - … présents pratiquement depuis les origines
  - Ce sont des défis *toujours actuels* …
    - … mais on progresse
  - Ce sont des défis *fondamentaux* …
    - … conceptuels au moins autant que techniques

C'est un choix arbitraire !

Autres défis :

- nouveaux modes de calcul (après le silicium)
- maîtrise de l'énergie
- systèmes hybrides
- sémantique des données (atelier 013)
- …

Produire des programmes corrects

Le logiciel : comment çà se fabrique ?

✤ 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

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

A-t-on réussi ? **Jamais du premier coup !**

Vérification et validation

Test

Preuve

Qu'est-ce qu'un programme correct ? (1)

- Un programme qui fait «ce qu'on veut qu'il fasse» …
- Le problème de l'expression des besoins
  - Le programme est-il conforme aux spécifications ?
  - Les spécifications traduisent-elles le cahier des charges ?
- Les propriétés à respecter
  - Propriétés fonctionnelles : le «quoi»
  - Propriétés non fonctionnelles, ou *qualité de service* : le «comment»
- La qualité de service : une expression difficile …
  - Performances
  - Maintenabilité, fiabilité, robustesse
  - Facilité d'utilisation, ergonomie
  - …

Qu'est-ce qu'un programme correct ? (2)

- Un obstacle fondamental …
- Tout programme non trivial contient des itérations (boucles)
- Le programme va-t-il s'arrêter ?
- Réponse (Turing, 1936)
  - C'est un problème indécidable !
  - Il n'y a pas d'algorithme *général* pour déterminer si un programme se termine
- Néanmoins …
  - On peut faire une preuve de terminaison pour des programmes spécifiques

(informellement)

```
T : si termine(T)
      alors boucle
      sinon retour
```

```
PGCD(p,q) {p, q > 0, entiers} :
x:=p; y:=q
tant que x≠y {
   si x > y
      alors x:=x-y
      sinon y:=y-x }
PGCD:=x
```

Les *bugs* (bogues)

- Il n'y a pas de *bugs* informatiques, il n'y a que des erreurs humaines

  "Les programmeurs parlent de *bugs* pour préserver leur santé mentale. Reconnaître un pareil nombre d'erreurs serait psychologiquement insupportable."

  Martin Hopkins, 1968

- Quelques *bugs* (tristement) célèbres
  - Les accidents d'irradiation de Therac 25 (1985-87)
  - Le ver de Morris (exploitation malveillante d'un *bug* d'Unix, 1988)
  - La panne du réseau téléphonique ATT (1990)
  - La perte d'Ariane 5 lors de son premier vol (1996)
  - La perte de la sonde Mars Climate Orbiter (1999)
  - La panne d'électricité au Nord-Est des États-Unis (2003)

En quête de la validité …

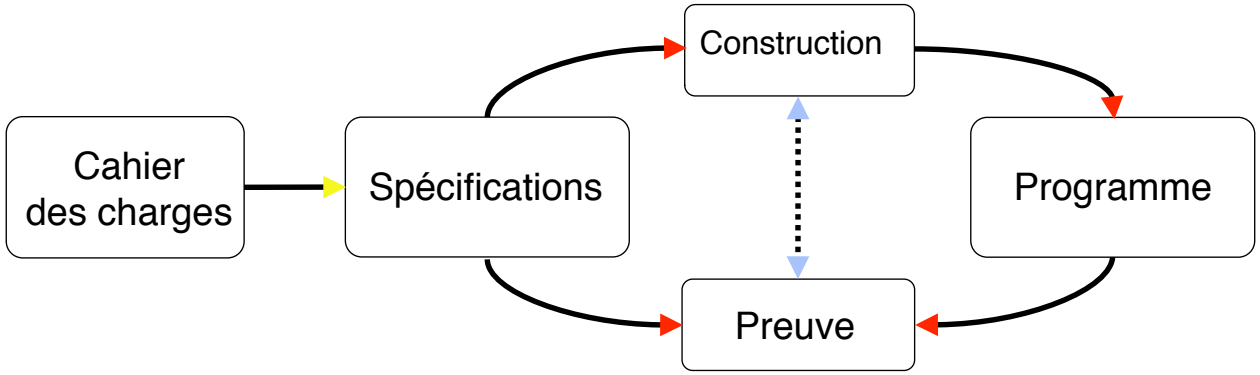
Pour un composant élémentaire

- Écrire le programme, et se convaincre qu'il est correct
  - Par le test
  - Par la vérification
  - Par la preuve
- Construire en même temps le programme et la preuve
- Engendrer le programme 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

Spécification, construction, preuve



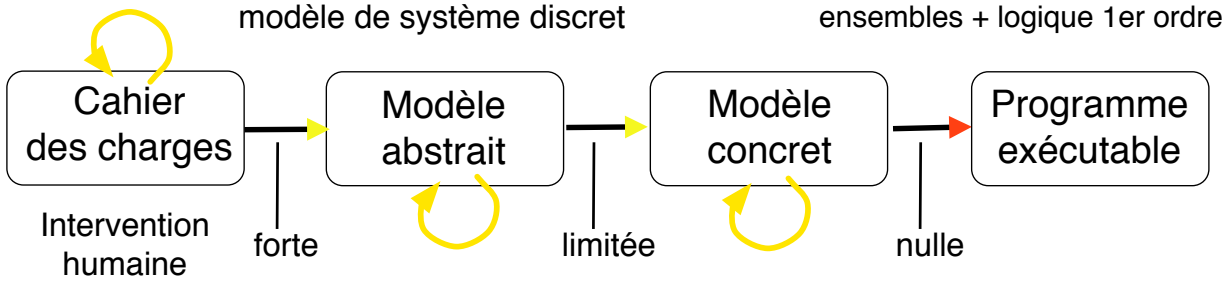
- Ingrédients de la spécification
  - Une base formelle (logique)
  - Un langage d'expression
  - Des outils d'aide à la construction
  - Des outils d'aide à la preuve

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

B, une méthode de construction de programmes

Machine à transition d'états, modèle de système discret

Bases du modèle : théorie des ensembles + logique 1er ordre



- Formalisation de la notion de machine abstraite
  - Propriétés : invariants, sûreté, vivacité
  - Obligation de preuve à chaque étape
    - Assure la validité du raffinement
    - Majoritairement engendrée et vérifiée par des outils
  - Le concepteur ne développe que des modèles

Une application : la ligne de métro 14 à Paris

Jean-Raymond Abrial, *The B-Book: Assigning Programs to Meanings*, Cambridge University Press, 1996

La méthode B, en pratique

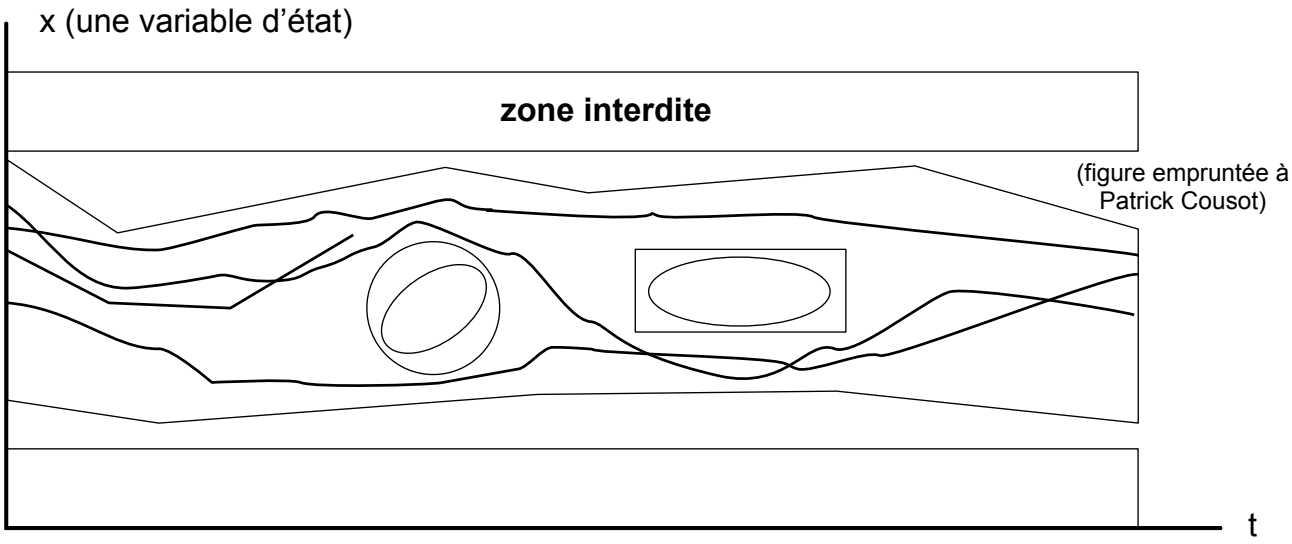
- Un exemple très simple (et sommairement traité …)
  - réservation de places
  - état initial : N places libres, 0 places réservées
- Opérations
  - réserver 1 place [précondition : il reste au moins 1 place libre]
  - consulter nombre de places libres
  - libérer une place [précondition : la place doit avoir été réservée]
- Modèle abstrait : ensemble de places
  - invariant : card{places libres} + card{places réservées} = N
- Modèle concret
  - représentation des ensembles (exemple : tableau)
  - préciser choix d'une place libre (exemple : plus petit indice)
  - démontrer que l'invariant est respecté après chaque opération

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 les équations traduisant les propriétés
- Difficultés communes
  - Explosion des états
  - Vérification de systèmes composés

Interprétation abstraite



Déterminer une "enveloppe" (peut partiellement être automatisé)
Vérifier que l'enveloppe respecte les limites
L'importance d'une «bonne» abstraction (choix de l'enveloppe)
- faux positifs
- faux négatifs

Coq, un assistant de preuve (1)

- Base : un système logique, le Calcul des Constructions Inductives (Coquand, Huet, 1985 ; Paulin-Mohring, 1991)
  - Logique d'ordre supérieur + langage fonctionnel typé
  - Démarche "constructive" (construction = preuve !)
- Ce que permet Coq
  - Définir et évaluer des fonctions et des prédicats
  - Écrire des théorèmes et des spécifications
  - Développer (semi-interactivement) des preuves de théorèmes
  - Certifier mécaniquement ces preuves
  - Produire des programmes dans des langages fonctionnels (OCaml, Scheme)

http://coq.inria.fr

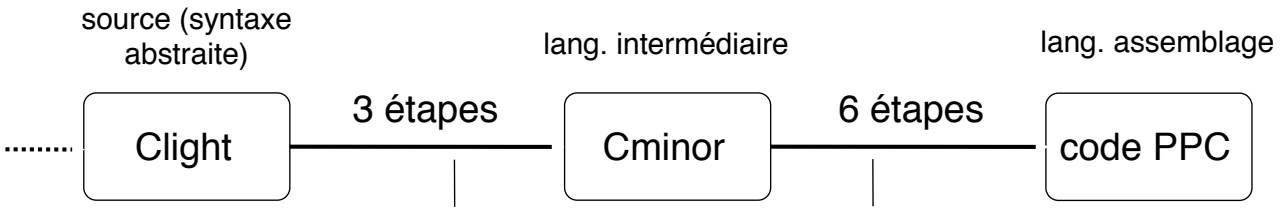
Coq, un assistant de preuve (2)

- Les bases scientifiques
  - Une avancée majeure à la frontière entre logique et informatique
  - Correspondance entre construction et preuve, soit entre calcul et démonstration
  - Origine lointaine : l'école intuitionniste (~1925)
- Quelques succès de Coq
  - Une preuve rigoureuse du théorème des 4 couleurs (2005)
  - Le premier compilateur «réaliste» certifié (2009)
  - Le premier système de bases de données relationnel certifié (2010)
- Les limites actuelles
  - Usage encore réservé à des spécialistes

Voir atelier 056 ! (lundi matin)

Vérification d'un compilateur

- Un compilateur pour un large sous-ensemble de C

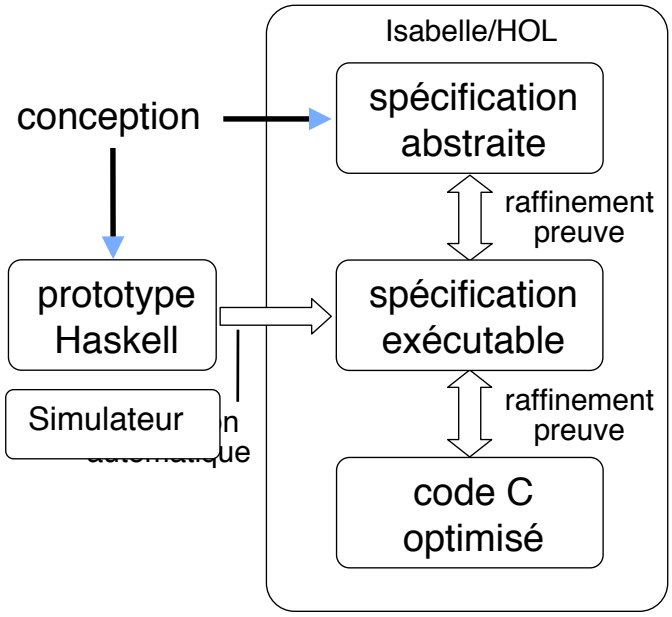


- Preuve et construction (avec Coq) pour chaque étape
  - Preuve que le comportement du programme objet est identique à celui du programme source (si pas d'erreur)
  - Génération d'un programme OCaml traduisant les spécifications
  - L'efficacité du code produit est très acceptable (-12% / gcc-02)

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

Vérification d'un noyau de système d'exploitation

- Une version du micro-noyau L4
- Difficultés
  - Asynchronisme
  - Gestion de la mémoire (pointeurs)
  - Accès direct aux fonctions du matériel (MMU, …)
- Quelques remèdes
  - Monoprocesseur
  - Gestion des interruptions par scrutation (*polling*)
  - Modèle de la machine cible
  - Restrictions sur usage de C



G. Malecha, G. Morrisett, A. Shinnar, R. Wisnesky, "Toward a Verified Relational Database System", *Proc. 37th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages* (*POPL 2010*), Madrid, Jan. 20-22 2010

Remarques finales sur la programmation

- 1968 : la «crise du logiciel» et l'invention du génie logiciel
- Quelques mythes des années 1970
  - La solution «par les masses» : si le projet est en retard, ajouter de la force de travail
  - La solution «par les outils» : mais la qualité des équipes est le facteur prépondérant
  - La «programmation automatique» : le métier de programmeur va disparaître
- Le métier a beaucoup évolué …
  - Élévation du niveau de formation
  - Importance de l'abstraction et des modèles
  - Encore beaucoup de travail pour rendre accessibles les méthodes avancées

Maîtriser le parallélisme

Comprendre et domestiquer le parallélisme

- Parallélisme = concurrence + asynchronisme
  - Concurrence : plusieurs activités se déroulent en même temps
    - Problème : ordonnancement (allouer les ressources aux activités)
  - Asynchronisme : les activités concurrentes ont des horloges non corrélées
    - Problème : synchronisation (assurer que les événements se déroulent dans un “bon” ordre)
  - Interaction entre activités concurrentes
    - Deux modèles : mémoire commune, échange de messages
- Pourquoi le parallélisme ?
  - Naturel : les événements du monde se déroulent en parallèle
  - Voulu : amélioration du rapport coût/efficacité
  - Imposé : les processeurs séquentiels atteignent leurs limites

Un problème de synchronisation

Deux opérations en parallèle sur un compte bancaire

processus p1

```
1. courant = lire_compte (1867A)
2. nouveau = courant + 1000
3. ecrire_compte (1867A, nouveau)
```

processus p2

```
1. courant = lire_compte (1867A)
2. nouveau = courant + 3000
3. ecrire_compte (1867A, nouveau)
```

Deux exemples possibles d'exécution

exécution 1 : p2.1 ; p2.2 ; p2.3 ; p1.1 ; p1.2 ; p1.3
exécution 2 : p1.1 ; p1.2 ; p2.1 ; p2.2 ; p2.3 ; p1.3

quels sont les résultats ? que peut-on en conclure ?

Un problème de synchronisation

Opérations atomiques

A1 **processus p1**

```
1. courant = lire_compte (1867A)
2. nouveau = courant + 1000
3. ecrire_compte (1867A, nouveau)
```

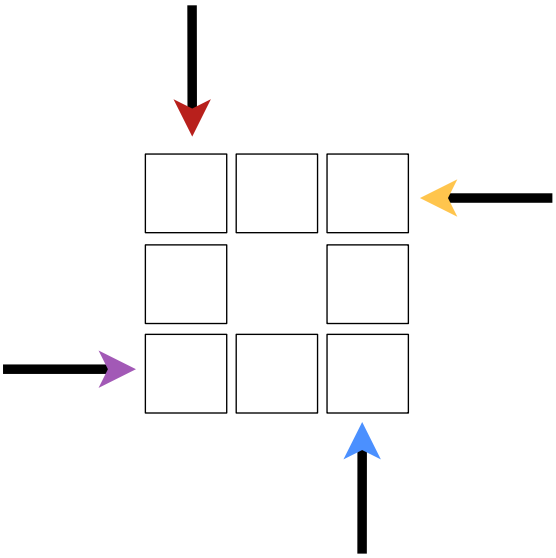
A2 **processus p2**

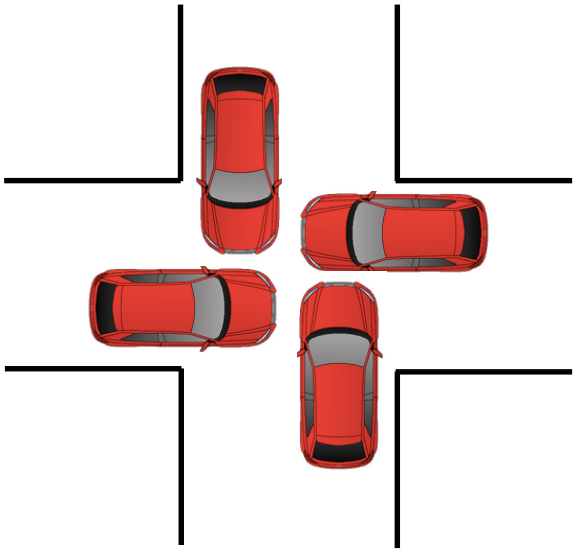
```
1. courant = lire_compte (1867A)
2. nouveau = courant + 3000
3. ecrire_compte (1867A, nouveau)
```

Restrictions sur l'ordre d'exécution

exécution 0 : p1.1 ; p1.2 ; p1.3 ; p2.1 ; p2.2 ; p2.3 **A1 ; A2**
~~**exécution 1 : p2.1 ; p2.2 ; p2.3 ; p1.1 ; p1.2 ; p1.3**~~ impossible
exécution 2 : p2.1 ; p2.2 ; p2.3 ; p1.1 ; p1.2 ; p1.3 **A2 ; A1**

Un problème d'ordonnancement





4 processus
Chacun a besoin de 3 blocs de mémoire

Interblocage !

Éviter l'interblocage

- Avec l'analogie du carrefour … deux solutions
- Mettre des feux
  - Traduction : chaque processus réserve globalement ses ressources
  - Mais on réduit le parallélisme
- Faire un rond-point giratoire
  - Traduction : tous les processus réservent leurs ressources dans le même ordre
  - C'est une solution souvent utilisée, mais pas toujours applicable
- Une troisième solution …
  - Ne rien faire …
  - … et accepter le coût du déblocage, si on estime que les risques sont faibles

Asynchronisme : premières avancées

- Les origines
  - Comportement non déterministe des systèmes d'exploitation (attribué aux défaillances du matériel)
- Premiers remèdes
  - Schémas d'exécution asynchrone
    - Processus / *threads* (abstraction du processeur physique)
    - Événement - réaction (abstraction de l'interruption)
  - Atomicité, isolation, exclusion mutuelle
    - si A atomique, A est exécutée entièrement ou pas du tout
    - si A et B atomiques, A || B équivaut à (A ; B) ou (B ; A)
  - Synchronisation : contraintes sur l'ordre des événements
  - Mécanismes élémentaires de synchronisation : verrous, sémaphores, moniteurs

P. Brinch Hansen (editor), *The Origins of Concurrent Programming*, Springer, 2002

Asynchronisme : une solution radicale

- Éliminer l'asynchronisme !
  - Contexte : systèmes réactifs
    - Réagissent aux signaux (ou aux flots de données) émis par l'environnement
  - Hypothèse : on a le temps d'élaborer la réaction avant le prochain signal (ou arrivée de données)
    - Hypothèse réaliste dans de nombreuses situations
  - Modèle d'exécution (périodique)
    - Réaction instantanée, exécution atomique
- Les langages synchrones (ou réactifs)
  - Estérel (impératif)
  - Lustre (flot de données)
  - Signal (flot de données)

A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic, R. de Simone, "The synchronous lan-guages 12 years later", *Proceedings of the IEEE*, 11:1, Jan 2003, pp. 64 - 83

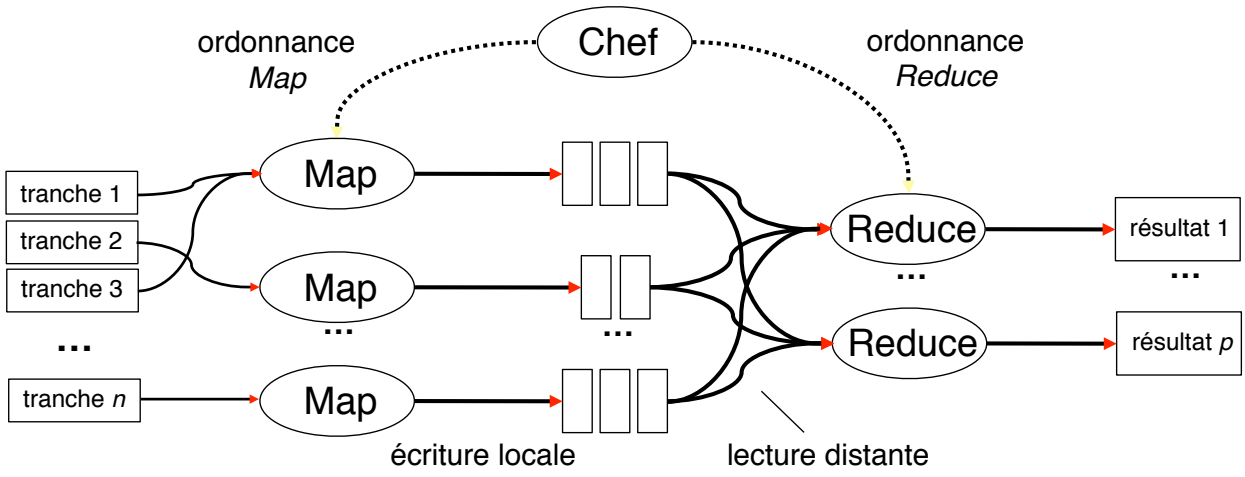
Parallélisme pratique à grande échelle

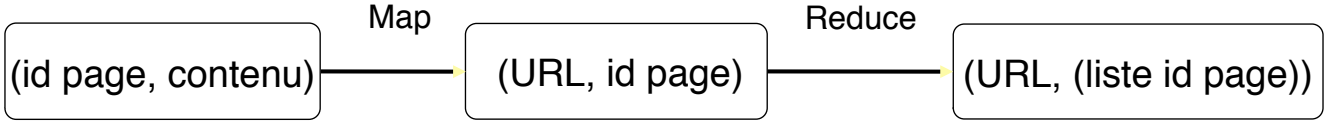
- Une classification des grandes applications parallèles
  - Un nombre relativement faible de schémas types (13 actuellement selon groupe de Berkeley)
  - Remplacent les bancs d'essai standard (SPLASH, etc.)

Bibliothèques	**Patrons et canevas**
Matrices denses	MapReduce
Matrices creuses	Traversée graphes
FFT et assimilés	Programmation dynamique
Combinatoire	Branch & Bound
Machines états finis	N-corps
	Grilles

- Schémas de composition
  - Programme séquentiel appelant des bibliothèques
  - Canevas intégrant des traitements spécifiques
  - Le gain décroît très vite avec la fraction parallélisable …

Un canevas pour le calcul parallèle : MapReduce





M. Stonebraker et al., "MapReduce and Parallel DBMSs: Friends or Foes?", *Comm. ACM,* 53:1, January 2010, pp. 64-71

Parallélisme à grain fin

- Contexte : les processeurs multicœurs
- Motivation
  - Loi de Moore : augmentation exponentielle de la densité de transistors dans les circuits intégrés
  - Densité forte => difficulté d'évacuation de la chaleur
  - Donc impossibilité d'augmenter la vitesse …
  - … d'où la solution multicœurs (processeurs multiples à l'intérieur d'une puce)
- Problème
  - Programmer efficacement à grain fin, de préférence sans les verrous
- Une voie d'approche : la mémoire transactionnelle
  - quelques détails plus loin

Faire face à l'imprévu

Faire face à l'imprévu

- Défaillances
  - Matériel
  - Logiciel
  - Communication
  - Fautes humaines
- Variations de la charge
  - Pics de la demande
  - Avalanche d'événements
- Mobilité
  - Connexion intermittente
- Attaques
  - Accès indu
  - Perturbation ou destruction

Objectifs : maintenir

la permanence et la qualité des services

l'intégrité et la cohérence des données

Une grande diversité de situations,
donc l'analyse des risques est essentielle

Outils :

- Atomicité
- Redondance
- Transformation
- Adaptation

X

X

Tolérance aux fautes

- Fautes, erreurs, défaillances
- Fiabilité et disponibilité
- Classes de défaillances
  - de la panne franche à la panne byzantine
- Les principes invariants
  - On ne peut pas espérer éliminer les fautes. Il faut vivre avec
  - La tolérance aux fautes repose sur la redondance
  - Pas de tolérance aux fautes sans hypothèses explicites
- Plusieurs approches
  - Maintenir la cohérence des données : les transactions
  - Maintenir la disponibilité d'un service
    - Concertation forte (à petite échelle)
    - Concertation faible ou nulle (à grande échelle)

Transactions (1)

- Problème : maintenir la cohérence des données
  - En permettant les accès concurrents
  - Malgré l'occurrence de pannes
- Solution : rendre atomique une action complexe
  - Atomicité (tout ou rien)
  - Cohérence (définition propre à l'application)
  - Isolation (l'autre face de l'atomicité)
  - Durabilité (permanence des modifications validées)
- Séparation des responsabilités
  - L'utilisateur garantit C, le système garantit A, I, D
  - L'exécution concurrente d'un ensemble de transactions ACID préserve la cohérence globale

Transactions (2)

- Comment çà marche …
  - Accès concurrents : théorie de la sérialisabilité
  - Atomicité (tout ou rien) : journalisation en mémoire stable
  - Durabilité : mémoire stable
- Extensions
  - Transactions réparties : validation atomique
  - Transactions longues : relâcher ACID
    - Transactions emboîtées, "sagas" (succès encore mitigé …)
  - Mémoire transactionnelle
    - Limitations des verrous
    - Réalisation mixte matériel/logiciel
    - Nombreux problèmes ouverts …

James Larus, Ravi Rajwar, *Transactional Memory*, Morgan & Claypool, 2006

Jim Gray, Andreas Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, 1993

Abstractions pour un service disponible

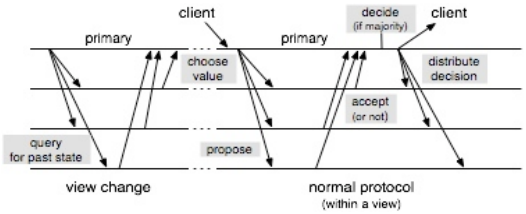
- La machine à états en *N* exemplaires
- Deux modes de réalisation
  - Redondance active
    - Requêtes dans le même ordre sur tous les serveurs
    - Diffusion atomique (totalement ordonnée)
  - Serveur primaire, serveurs de secours
    - Vue uniforme pour tous les serveurs : “vues synchrones”

Machine en N copies → Maintien de la cohérence → Décision concertée → abstraction → Consensus

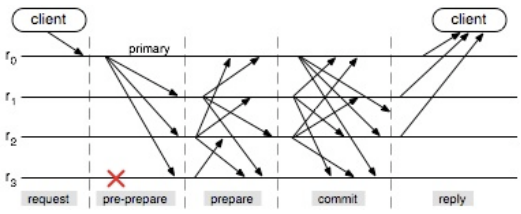
- Le consensus : décider en environnement incertain
  - Diffusion atomique et vues synchrones se réduisent au consensus
  - Accord, intégrité, validité, terminaison

Brève histoire du consensus

- Pannes franches en asynchrone
  - Pas d’algorithme déterministe !
  - Mais des palliatifs ingénieux
    - Détecteurs imparfaits
    - Paxos



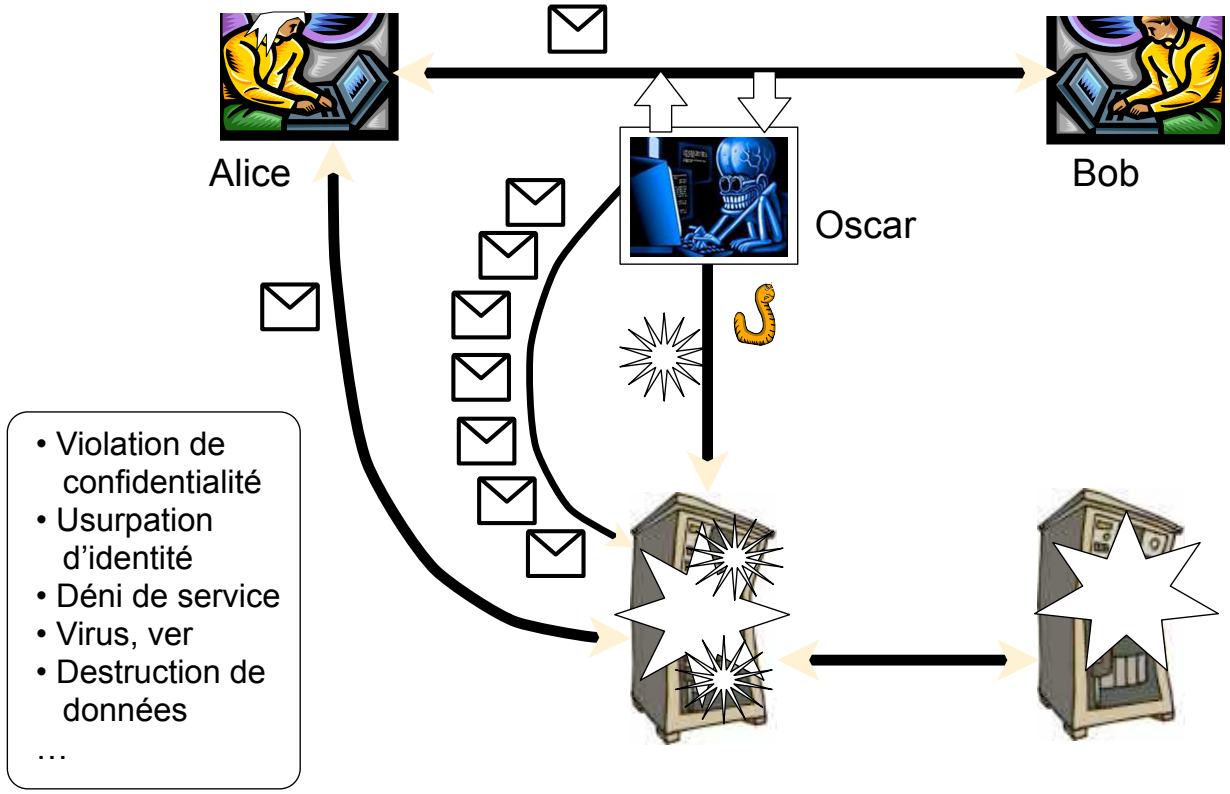
- Pannes byzantines
  - Redondance forte
    - 3*f*+1 pour tolérer *f* fautes
  - Complexité
    - Jusqu’en 1999 : exponentielle
    - Depuis : ~ quadratique, sujet “chaud”



Tolérance aux fautes à grande échelle

- Limites des algorithmes à base de consensus
  - Applicables à des grappes de serveurs
  - Ne passent pas à l'échelle de l'Internet
- Redondance massive
  - Algorithmes de diffusion épidémiques (P2P)
    - Résistance naturelle aux défaillances, garanties statistiques
  - Auto-stabilisation, auto-réparation
- Adaptation et reconfiguration
  - Exemple ancien : le routage adaptatif dans l'Internet
  - Exemples courants : réseaux de capteurs, mobiles
- Calcul à hautes performances
  - Temps de sauvegarde-reprise > temps moyen entre pannes !
  - Tolérance aux fautes = forte ponction sur la puissance de calcul

Problèmes de sécurité informatique



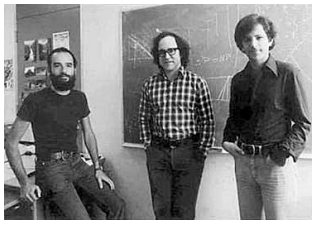
Quelques instruments de la sécurité

- Dissimuler une information
  - La rendre indéchiffrable même si elle est captée
    (cryptographie)
  - La cacher dans une autre information d'apparence anodine
    (stéganographie)
- Authentifier une personne ou une organisation
  - Fournir une preuve certifiée de son identité
- Authentifier un document
  - Signature électronique
- Protéger l'accès à une ressource
- Détecter une intrusion
  - Au besoin, la provoquer pour piéger l'adversaire …
- Détecter et détruire un virus

La sécurité reste un problème encore mal maîtrisé …

Une avancée importante pour la sécurité

- La cryptographie à clé publique
  - Diffie, Hellman, Merkle (1976) ; Rivest, Shamir, Adleman (1978)
  - La cryptographie «classique» repose sur le *partage* d'une clé secrète
  - Mais un secret partagé risque de ne pas rester secret



- Deux clés pour chacun …
  - Une clé publique, connue de tous
    - Elle sert à chiffrer les messages envoyés
  - Une clé privée, *connue de son détenteur seul*
    - Elle sert à déchiffrer les messages reçus
  - Il est (bien sûr) *très difficile* de trouver la clé privée à partir de la clé publique …

RSA : un protocole très utilisé dans le commerce électronique, la banque, etc.
(le petit cadenas sur les pages web)

Cryptographie à clé publique : fonctionnement

- L'algorithme de RSA

  Choisir p, q nombres premiers grands ($> 10^{100}$)

  soit $n = p \times q$ et soit $z = (p - 1)(q - 1)$

  Choisir d tel que d et z soient premiers entre eux

  Trouver e tel que $e \times d = 1 \bmod z$

  choisir le plus petit de $\{z + 1, 2z + 1, 3z + 1, ...\}$ divisible par d

  Découper le message clair en blocs de k bits, avec $2^k < n$

  La clé publique est (e, n). Chiffrement : $\{M\} = M^e \bmod n$

  La clé privée est (d, n). Déchiffrement : $M = \{M\}^d \bmod n$

  Çà marche parce que

  si a et n premiers entre eux, alors $a^z \bmod n = 1$

  donc : $M^{e.d} \bmod n = M^{e.d - 1} \times M \bmod n = M \bmod n$

  Le décryptage nécessite de trouver p et q, donc de factoriser n (difficile !)

Cryptographie à clé publique : propriétés

Rappel

La clé publique est (e, n). Chiffrement : $\{M\} = M^e \bmod n$

La clé privée est (d, n). Déchiffrement : $M = \{M\}^d \bmod n$

- Propriété 1 : réversibilité

  d et e jouent un rôle symétrique. L'algorithme est réversible

  Un message chiffré avec la clé privée est déchiffré avec la clé publique. Application : authentification (signature électronique)

- Propriété 2 : homomorphisme (pour le produit)

  Si a et b sont des nombres, alors {a} x {b} = {a x b}

  Application : faire exécuter a x b sans dévoiler a et b !

  … mais cela ne marche que pour la multiplication

Un sujet chaud : la cryptographie homomorphe

- Une motivation : l'informatique dans les nuages (*cloud computing*)
  - Virtualisation de l'environnement d'exécution des applications
  - Un problème : la sécurité des données (quelles garanties ?)
- Une piste : la cryptographie homomorphe
  - Objectif : généraliser la propriété de RSA pour la multiplication à *toutes* les opérations (à commencer par l'addition)
    - $\{F(a_1, a_2, \ldots a_k)\} = F(\{a_1\}, \{a_2\}, \ldots \{a_k\})$, idéalement pour tout F
  - Problème *difficile* : pas de progrès pendant 30 ans
  - Une percée (Craig Gentry, 2009)
    - Mais encore partiel et peu efficace
  - De nombreux travaux en cours

Vote électronique

- Pourquoi le vote électronique ?
  - Machines à voter : gain de temps, économie de papier (?)
  - Par l'Internet : évite les déplacements pour les personnes peu mobiles
- Le cahier des charges
  - Tout vote remplissant les conditions légales doit être validé
  - Le résultat du vote doit refléter le choix de chaque électeur
  - Le vote est anonyme (il est impossible de faire le lien entre un électeur et un vote)
  - Le vote est vérifiable (par chaque électeur, par une autorité)
- Une histoire mouvementée …
  - Les *bugs* des machines à voter …
- Peut-on se fier au vote électronique ?

Aspects de la sécurité informatique

- Aspects scientifiques et techniques
  - Des outils techniques existent …
  - … mais des progrès à faire sur les aspects fondamentaux
    - Une base logique pour la sécurité ?
- Aspects légaux et réglementaires
  - Infrastructure à clés publiques
  - Signature électronique, certificats
  - Tiers de confiance
- Aspects sociétaux
  - Des progrès à faire sur la diffusion de la culture de sécurité …
    - Appréciation des risques, comportements dangereux
    - Utilisation des outils
  - … et sur l'équilibre entre sécurité et respect de la vie privée

Observations finales (1)

- Des avancées significatives appuyées par la théorie …
  - Vers des systèmes certifiés
  - Vers une prévision fine des performances
  - Vers une gestion contrôlée des ressources
  - Vers une administration sûre des grands systèmes
- … mais il reste une place pour l'art de l'architecte
- Défis anciens et nouveaux

Butler W. Lampson, "Hints for Computer Systems Design", *IEEE Software*, 1:1, 11-28, 1984

Théorie du parallélisme	Sécurité
Complexité algorithmique	Du discret au continu
Sémantique des données	Maîtrise des très grands systèmes
Spécification et preuve	Construction et composition
…	…

Observations finales (2)

- La démarche scientifique en informatique
  - Approches théoriques
    - Modèles formels et semi-formels
  - Approches expérimentales
    - Conception, construction, évaluation
  - Interaction et complémentarité
    - “Théorique” ne veut pas dire “coupé du réel”
    - “Pratique” ne veut pas dire “non rigoureux”
  - Incidence sur l’enseignement
- La force (et la difficulté) des modèles formels
  - L’apport des mathématiques à l’informatique
  - L’apport de l’informatique aux mathématiques (et à la logique)

Pour aller plus loin

- Un site de diffusion de la culture informatique
  - Interstices (INRIA, CNRS, universités) : interstices.info/
- Un cours en ligne : Gérard Berry, Collège de France
  - www.college-de-france.fr/default/EN/all/inn_tec2007/
- Sur les ponts entre mathématiques et informatique
  - Gilles Dowek. Les métamorphoses du calcul, Le Pommier, 2007
- Sur l’histoire de l’informatique
  - 6 conférences (2011), liens sur proton.inrialpes.fr/~krakowia/Presentations/Histoire/
  - Site d’Aconit : aconit.org/
- Une biographie
  - Andrew Hodges. Alan Turing, ou l’énigme de l’intelligence, Payot, 1988

Merci de votre attention