Congrès Specif 2010

C’est quoi, au fond, l’informatique ?

Sacha Krakowiak
Université de Grenoble

Congrès Specif 2010

C’est quoi, au fond, l’informatique ?

Objet(s), démarche, évolution

Sacha Krakowiak
Université de Grenoble

L’informatique, une science de l’artificiel ?

L'informatique, une science de l'artificiel ?

L'informatique est, très largement, une "science de l'artificiel" (†)

(†) Herbert A. Simon, *The Sciences of the Artificial*, 2nd ed., MIT Press, 1981

L'informatique, une science de l'artificiel ?

L'informatique est, très largement, une "science de l'artificiel" (†)

Nous construisons nous-mêmes les objets de notre étude

Nous construisons même les outils pour cette construction (et les méta(*)-outils ...)

Le processus de conception et de construction est l'un des objets de notre discipline

Il y a une interaction féconde entre théorie et pratique

(†) Herbert A. Simon, *The Sciences of the Artificial*, 2nd ed., MIT Press, 1981

Au cœur de l'informatique, l'algorithme (1)

Au cœur de l'informatique, l'algorithme (1)

Algorithme : procédé utilisant un répertoire fini d'opérations *effectives* pour résoudre une classe spécifiée de problèmes en un nombre *fini* d'étapes.

Notion ancienne (Euclide, Al Khwarizmi, …), utilisée dans d'autres disciplines, y compris non scientifiques.

Donald E. Knuth, *The Art of Computer Programming*, vols. 1, 2, 3, Addison-Wesley, 1997, 1998. (Parties du vol. 4 disponibles en ligne)

Au cœur de l'informatique, l'algorithme (1)

Algorithme : procédé utilisant un répertoire fini d'opérations *effectives* pour résoudre une classe spécifiée de problèmes en un nombre *fini* d'étapes.

Notion ancienne (Euclide, Al Khwarizmi, …), utilisée dans d'autres disciplines, y compris non scientifiques.

Un algorithme doit être correct (se terminer, faire ce qu'on lui demande) et efficace (le faire au moindre coût).

Donald E. Knuth, *The Art of Computer Programming*, vols. 1, 2, 3, Addison-Wesley, 1997, 1998. (Parties du vol. 4 disponibles en ligne)

Au cœur de l'informatique, l'algorithme (1)

Algorithme : procédé utilisant un répertoire fini d'opérations *effectives* pour résoudre une classe spécifiée de problèmes en un nombre *fini* d'étapes.

Notion ancienne (Euclide, Al Khwarizmi, …), utilisée dans d'autres disciplines, y compris non scientifiques.

Un algorithme doit être correct (se terminer, faire ce qu'on lui demande) et efficace (le faire au moindre coût).

Robert Sedgewick, Philippe Flajolet, *Introduction à l'analyse des algorithmes*, International Thomson Publishing France (1996)

Donald E. Knuth, *The Art of Computer Programming*, vols. 1, 2, 3, Addison-Wesley, 1997, 1998. (Parties du vol. 4 disponibles en ligne)

Au cœur de l'informatique, l'algorithme (2)

Au cœur de l'informatique, l'algorithme (2)

Une formalisation de l'algorithme : la machine de Turing (1936)

Andrew Hodges, *Alan Turing: The Enigma*, Simon & Schuster Vintage edition, 1992

Au cœur de l'informatique, l'algorithme (2)

Une formalisation de l'algorithme : la machine de Turing (1936)

Thèse de Church-Turing : ***tout*** algorithme est représentable par une machine de Turing

Andrew Hodges, *Alan Turing: The Enigma*, Simon & Schuster Vintage edition, 1992

Au cœur de l'informatique, l'algorithme (2)

Une formalisation de l'algorithme : la machine de Turing (1936)

Thèse de Church-Turing : ***tout*** algorithme est représentable par une machine de Turing

- Trois résultats fondamentaux :

  L'équivalence des différents modèles de calcul (λ-calcul, machine de Turing, …)

Andrew Hodges, *Alan Turing: The Enigma*, Simon & Schuster Vintage edition, 1992

Au cœur de l'informatique, l'algorithme (2)

Une formalisation de l'algorithme : la machine de Turing (1936)

Thèse de Church-Turing : ***tout*** algorithme est représentable par une machine de Turing

- Trois résultats fondamentaux :
  - L'équivalence des différents modèles de calcul (λ-calcul, machine de Turing, …)
  - La machine de Turing universelle (qui peut émuler toute machine)

Andrew Hodges, *Alan Turing: The Enigma*, Simon & Schuster Vintage edition, 1992

Au cœur de l'informatique, l'algorithme (2)

Une formalisation de l'algorithme : la machine de Turing (1936)

Thèse de Church-Turing : ***tout*** algorithme est représentable par une machine de Turing

- Trois résultats fondamentaux :
  - L'équivalence des différents modèles de calcul (λ-calcul, machine de Turing, …)
  - La machine de Turing universelle (qui peut émuler toute machine)
  - L'indécidabilité du problème de l'arrêt

Andrew Hodges, *Alan Turing: The Enigma*, Simon & Schuster Vintage edition, 1992

L’algorithme à la conquête du monde

L'algorithme à la conquête du monde

Un paysage maintenant familier :

- Les avatars de la machine de Turing dans le monde physique.
- La notion de système, ensemble complexe matériel et logiciel construit pour fournir un service dans un environnement donné.
- L'informatisation des communications, via l'Internet et la numérisation du téléphone, de la télévision, des sons et des images.
- La "numérisation du monde" (*) et l'extension des domaines d'application (biologie, médecine, applications à impact sociétal).

(*) Gérard Berry, *Pourquoi et comment le monde devient numérique*, cours au Collège de France, 2007, http://www.college-de-france.fr/default/EN/all/inn_tec2007/

Notre cahier des charges

Un algorithme doit être correct (faire ce qu'on lui demande), et efficace (le faire au moindre coût).

Notre cahier des charges

Un algorithme doit être correct (faire ce qu'on lui demande), et efficace (le faire au moindre coût).

Mais, dans le monde réel, cela ne suffit pas.
Nous devons construire des systèmes qui soient

- Corrects
- Efficaces
- Sûrs
- Conviviaux

Notre cahier des charges

Un algorithme doit être correct (faire ce qu'on lui demande), et efficace (le faire au moindre coût).

Mais, dans le monde réel, cela ne suffit pas.
Nous devons construire des systèmes qui soient

- Corrects
- Efficaces
- Sûrs
- Conviviaux

Conformes à leurs spécifications. Mais écrire de "bonnes" spécifications est un problème non trivial.

Notre cahier des charges

Un algorithme doit être correct (faire ce qu'on lui demande), et efficace (le faire au moindre coût).

Mais, dans le monde réel, cela ne suffit pas.
Nous devons construire des systèmes qui soient

- Corrects

  Conformes à leurs spécifications. Mais écrire de "bonnes" spécifications est un problème non trivial.

- Efficaces

  Économes en temps, espace, énergie, …

- Sûrs

- Conviviaux

Notre cahier des charges

Un algorithme doit être correct (faire ce qu'on lui demande), et efficace (le faire au moindre coût).

Mais, dans le monde réel, cela ne suffit pas.
Nous devons construire des systèmes qui soient

- Corrects

  Conformes à leurs spécifications. Mais écrire de "bonnes" spécifications est un problème non trivial.

- Efficaces

  Économes en temps, espace, énergie, …

- Sûrs

  Résistants aux événements imprévus ou indésirables : défaillances, attaques, surcharge, …

- Conviviaux

Notre cahier des charges

Un algorithme doit être correct (faire ce qu'on lui demande), et efficace (le faire au moindre coût).

Mais, dans le monde réel, cela ne suffit pas.
Nous devons construire des systèmes qui soient

- Corrects — Conformes à leurs spécifications. Mais écrire de "bonnes" spécifications est un problème non trivial.
- Efficaces — Économes en temps, espace, énergie, …
- Sûrs — Résistants aux événements imprévus ou indésirables : défaillances, attaques, surcharge, …
- Conviviaux — Qui prennent en compte les besoins des humains

Quelques forces qui façonnent l'informatique

- Quatre objectifs
  - Construire des systèmes corrects, efficaces, sûrs, conviviaux

Quelques forces qui façonnent l'informatique

- Quatre objectifs
  - Construire des systèmes corrects, efficaces, sûrs, conviviaux
- Deux moteurs et une contrainte
  - Les besoins
  - La technologie
  - L'environnement

Quelques forces qui façonnent l'informatique

- Quatre objectifs
  - Construire des systèmes corrects, efficaces, sûrs, conviviaux
- Deux moteurs et une contrainte
  - Les besoins
  - La technologie
  - L'environnement
- Trois défis
  - La complexité
  - Le parallélisme
  - L'imprévu

Maîtriser la complexité

Maîtriser la complexité

- Un outil (intellectuel) fondamental : l'abstraction
  - Abstraire : transformer une entité en gardant l'essentiel, et en oubliant provisoirement l'accessoire
    - Clé du succès : savoir définir « l'essentiel »
  - Quelques aspects de l'abstraction
    - Virtualisation
    - Modularité
    - Modélisation

Maîtriser la complexité

- Un outil (intellectuel) fondamental : l'abstraction
  - Abstraire : transformer une entité en gardant l'essentiel, et en oubliant provisoirement l'accessoire
    - Clé du succès : savoir définir « l'essentiel »
  - Quelques aspects de l'abstraction
    - Virtualisation
    - Modularité
    - Modélisation

Jerome H. Saltzer, M. Frans Kaashoek, *Principles of Computer System Design: An Introduction*, Morgan Kaufmann, 2009

Maîtriser la complexité

- Un outil (intellectuel) fondamental : l'abstraction
  - Abstraire : transformer une entité en gardant l'essentiel, et en oubliant provisoirement l'accessoire
    - Clé du succès : savoir définir « l'essentiel »
  - Quelques aspects de l'abstraction
    - Virtualisation
    - Modularité
    - Modélisation

Jerome H. Saltzer, M. Frans Kaashoek, *Principles of Computer System Design: An Introduction*, Morgan Kaufmann, 2009

- Applications de la démarche d'abstraction
  - Pour le calcul et la communication
  - Pour la conception et la construction
  - Pour la vérification et la preuve

Abstractions du calcul (séquentiel)

Abstractions du calcul (séquentiel)

- Des modèles de calcul aux langages de programmation
  - Machine (universelle) de Turing
    - Les machines (physiques) de von Neumann
    - Les langages impératifs
  - λ-calcul
    - Les langages fonctionnels

Abstractions du calcul (séquentiel)

- Des modèles de calcul aux langages de programmation
  - Machine (universelle) de Turing
    - Les machines (physiques) de von Neumann
    - Les langages impératifs
  - λ-calcul
    - Les langages fonctionnels
- Le programme comme donnée
  - Réalisation effective du calcul
    - Démarche descendante : le compilateur (traducteur)
    - Démarche ascendante : la machine virtuelle (interprète)
    - Démarche mixte
  - Génération, transformation
  - Vérification, preuve

Abstractions du calcul (séquentiel)

- Des modèles de calcul aux langages de programmation
  - Machine (universelle) de Turing
    - Les machines (physiques) de von Neumann
    - Les langages impératifs
  - λ-calcul
    - Les langages fonctionnels
- Le programme comme donnée
  - Réalisation effective du calcul
    - Démarche descendante : le compilateur (traducteur)
    - Démarche ascendante : la machine virtuelle (interprète)
    - Démarche mixte
  - Génération, transformation
  - Vérification, preuve

Harold Abelson, Gerald J. Sussman, Julie Sussman, *Structure and Interpretation of Computer Programs*, MIT Press, 1984, 1992

En quête du “langage idéal” …

En quête du “langage idéal” …

- Les grands paradigmes
  - Langages impératifs : FORTRAN, COBOL, Algol 60, Pascal, Ada, C, …
  - Langages fonctionnels : LISP, Scheme, ML, OCaml, Haskell, …
  - Langages logiques : Planner, Prolog, …
  - Langages (impératifs) à objets : Simula, Smalltalk, Eiffel, Java, C#, …
  - Langages déclaratifs : SQL, XML, …
  - Langages à flots de données (FD) : Lucid, Lustre, …
  - Langages d’acteurs : Erlang, …
  - Langages de scripts : *shell*, Tcl-Tk, JavaScript, Python, …

Des milliers de langages recensés …

En quête du “langage idéal” …

- Les grands paradigmes
  - Langages impératifs : FORTRAN, COBOL, Algol 60, Pascal, Ada, C, …
  - Langages fonctionnels : LISP, Scheme, ML, OCaml, Haskell, …
  - Langages logiques : Planner, Prolog, …
  - Langages (impératifs) à objets : Simula, Smalltalk, Eiffel, Java, C#, …
  - Langages déclaratifs : SQL, XML, …
  - Langages à flots de données (FD) : Lucid, Lustre, …
  - Langages d’acteurs : Erlang, …
  - Langages de scripts : *shell*, Tcl-Tk, JavaScript, Python, …
- Une tentative : la synthèse
  - Fonctionnel + objets + logique + FD + concurrence : Oz
  - Fonctionnel + logique : ALF
  - Fonctionnel + objets + acteurs : Scala

Des milliers de langages recensés …

En quête du “langage idéal” …

- Les grands paradigmes
  - Langages impératifs : FORTRAN, COBOL, Algol 60, Pascal, Ada, C, …
  - Langages fonctionnels : LISP, Scheme, ML, OCaml, Haskell, …
  - Langages logiques : Planner, Prolog, …
  - Langages (impératifs) à objets : Simula, Smalltalk, Eiffel, Java, C#, …
  - Langages déclaratifs : SQL, XML, …
  - Langages à flots de données (FD) : Lucid, Lustre, …
  - Langages d’acteurs : Erlang, …
  - Langages de scripts : *shell*, Tcl-Tk, JavaScript, Python, …
- Une tentative : la synthèse
  - Fonctionnel + objets + logique + FD + concurrence : Oz
  - Fonctionnel + logique : ALF
  - Fonctionnel + objets + acteurs : Scala
- Une autre tentative : la spécialisation
  - Langages dédiés : *Domain Specific Languages*

Des milliers de langages recensés …

Un concept unificateur : les types

Un concept unificateur : les types

- Le programme comme base de raisonnement logique
  - Une voie vers la production de programmes corrects

Un concept unificateur : les types

- Le programme comme base de raisonnement logique
  - Une voie vers la production de programmes corrects
- Types pour les langages de programmation
  - Type = assertion associée à un élément du langage
  - Règles de typage : utiliser un élément conformément à son type
  - Langage statiquement typé : un programme “bien typé” est exempt de certaines erreurs à l’exécution
  - Inférence de types (ML, …) : détermination automatique de types

Benjamin C. Pierce, *Types and Programming Languages*, MIT Press, 2002

Un concept unificateur : les types

- Le programme comme base de raisonnement logique
  - Une voie vers la production de programmes corrects
- Types pour les langages de programmation
  - Type = assertion associée à un élément du langage
  - Règles de typage : utiliser un élément conformément à son type
  - Langage statiquement typé : un programme “bien typé” est exempt de certaines erreurs à l’exécution
  - Inférence de types (ML, …) : détermination automatique de types
- Types en logique
  - Correspondance calcul - preuve

Gilles Dowek, *Théories des types*, cours de l’École polytechnique, en ligne

Benjamin C. Pierce, *Types and Programming Languages*, MIT Press, 2002

Un autre concept unificateur : la modularité

Un autre concept unificateur : la modularité

- Un objectif d'apparence simple …
  - Composer un système à partir de pièces élémentaires
  - Éléments réutilisables et remplaçables ("échange standard")
  - Interface visible, réalisation cachée

Un autre concept unificateur : la modularité

- Un objectif d'apparence simple …
  - Composer un système à partir de pièces élémentaires
  - Éléments réutilisables et remplaçables ("échange standard")
  - Interface visible, réalisation cachée
- L'abstraction procédurale
  - Séparation interface-réalisation
  - Signature, formalisation de l'interface
    - Paramètres formels et paramètres effectifs
    - Types et modes de passage (valeur, référence, …)
    - Procédures en paramètres

Un autre concept unificateur : la modularité

- Un objectif d'apparence simple …
  - Composer un système à partir de pièces élémentaires
  - Éléments réutilisables et remplaçables ("échange standard")
  - Interface visible, réalisation cachée
- L'abstraction procédurale
  - Séparation interface-réalisation
  - Signature, formalisation de l'interface
    - Paramètres formels et paramètres effectifs
    - Types et modes de passage (valeur, référence, …)
    - Procédures en paramètres

Peter Naur (editor), *Report on the Algorithmic Language Algol 60*, Regnecentralen, Copenhagen, May 1960

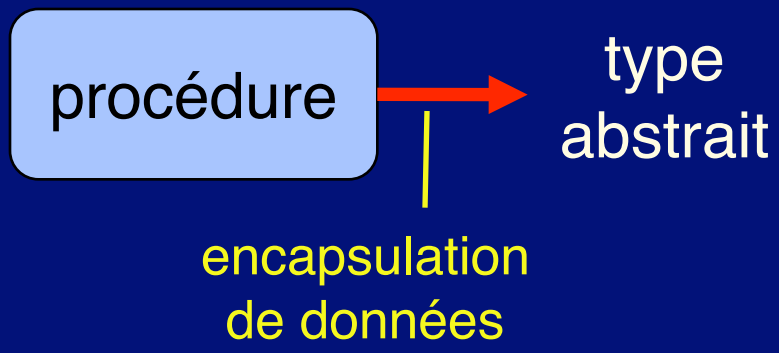
Un autre concept unificateur : la modularité

- Un objectif d'apparence simple …
  - Composer un système à partir de pièces élémentaires
  - Éléments réutilisables et remplaçables ("échange standard")
  - Interface visible, réalisation cachée
- L'abstraction procédurale
  - Séparation interface-réalisation
  - Signature, formalisation de l'interface
    - Paramètres formels et paramètres effectifs
    - Types et modes de passage (valeur, référence, …)
    - Procédures en paramètres
  - Limites et problèmes
    - Pas d'encapsulation des données
    - Variables globales
    - Effets de bord

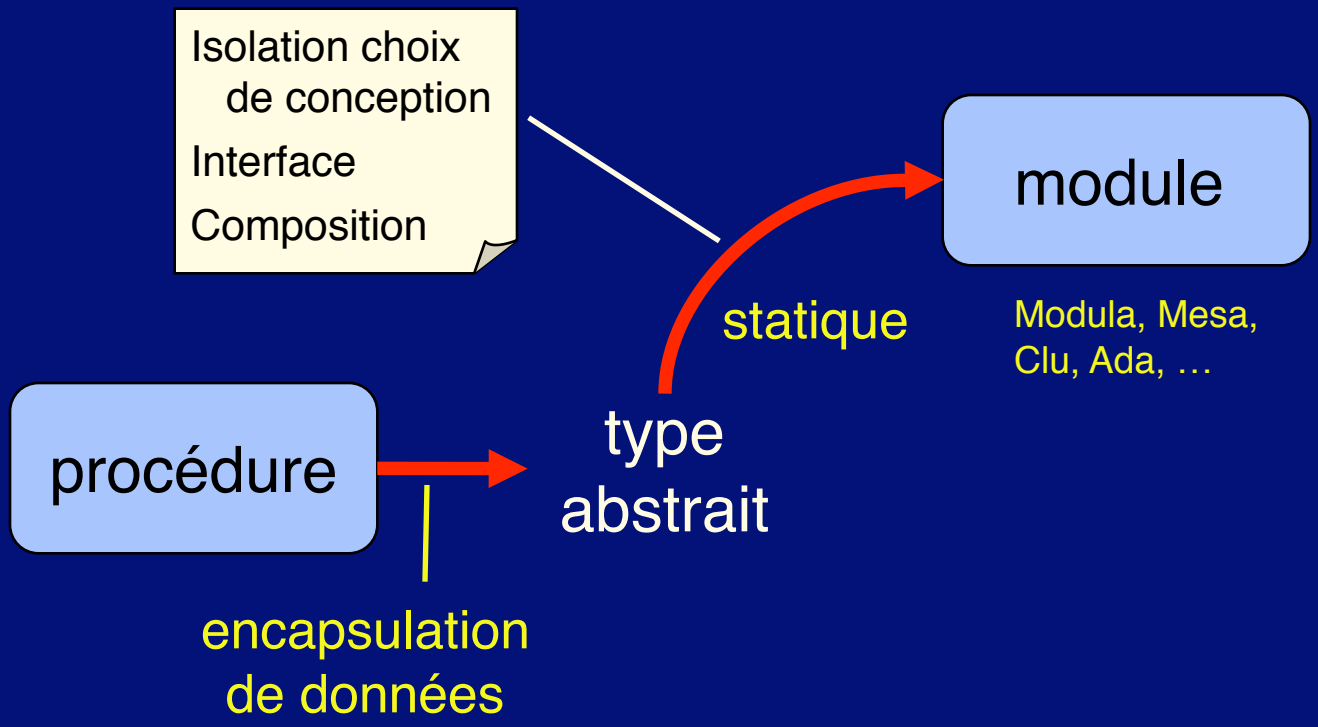
Peter Naur (editor), *Report on the Algorithmic Language Algol 60*, Regnecentralen, Copenhagen, May 1960

Des procédures aux composants

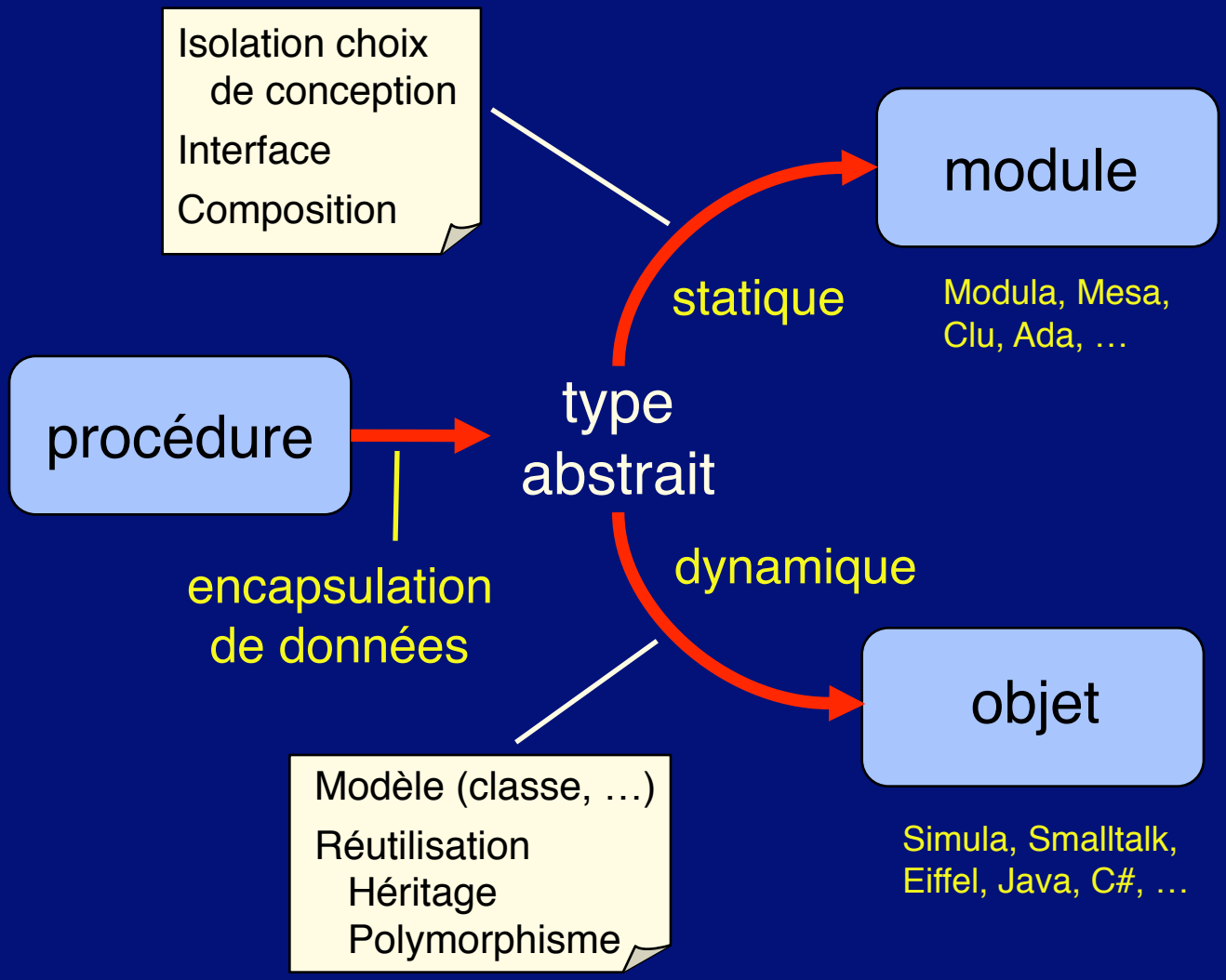
Des procédures aux composants



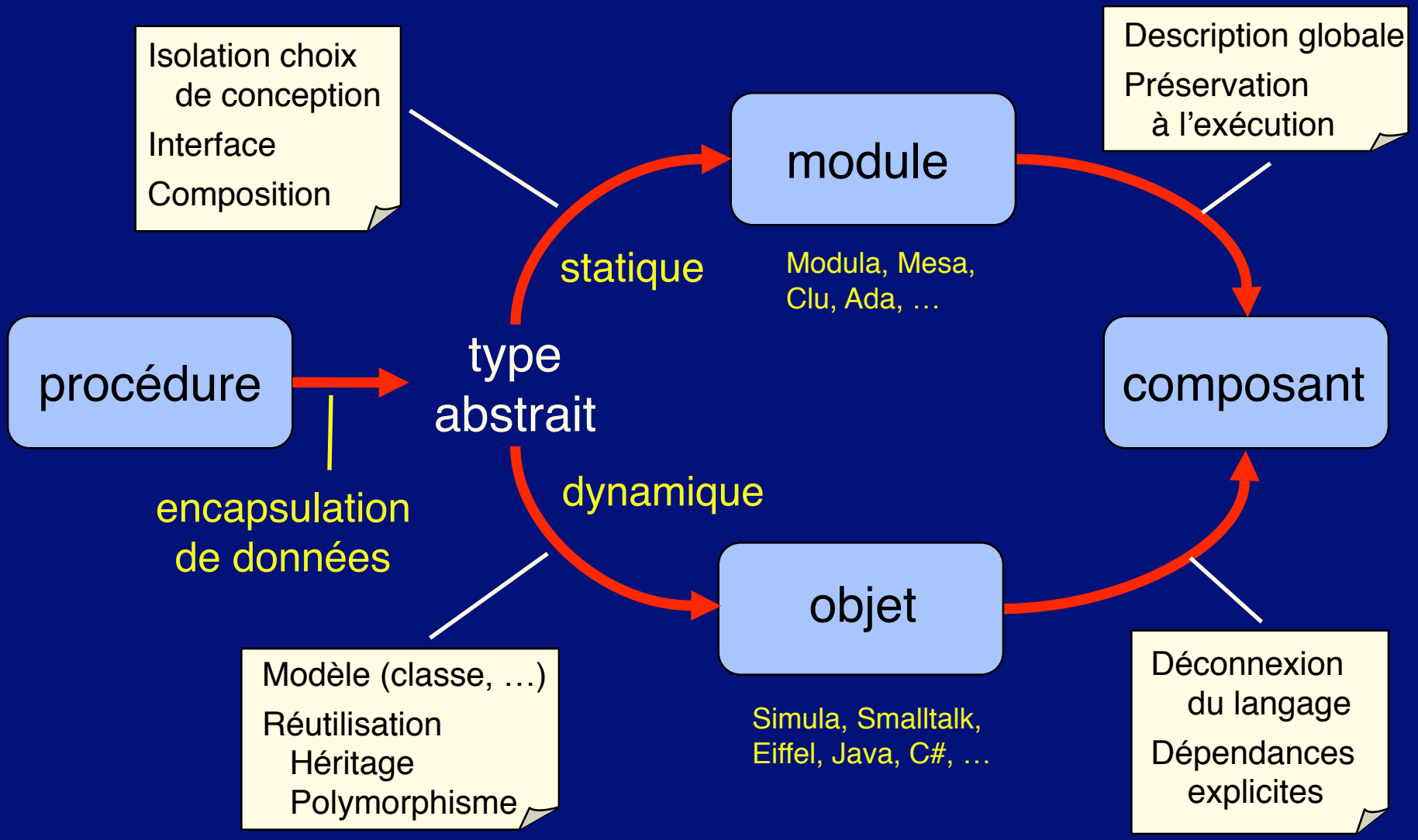
Des procédures aux composants



Des procédures aux composants



Des procédures aux composants



Architecture logicielle

Architecture logicielle

- Description globale d'un système
  - Composants, connecteurs, configurations
  - Règles de composition (cf. plus loin)
  - Langage de description d'architecture (ADL)
    - Base pour construction, vérification et preuve, administration

Architecture logicielle

- Description globale d'un système
  - Composants, connecteurs, configurations
  - Règles de composition (cf. plus loin)
  - Langage de description d'architecture (ADL)
    - Base pour construction, vérification et preuve, administration
- L'architecture logicielle, base pour l'administration
  - Configuration
  - Déploiement
  - Reconfiguration

Architecture logicielle

- Description globale d'un système
  - Composants, connecteurs, configurations
  - Règles de composition (cf. plus loin)
  - Langage de description d'architecture (ADL)
    - Base pour construction, vérification et preuve, administration
- L'architecture logicielle, base pour l'administration
  - Configuration
  - Déploiement
  - Reconfiguration

Les erreurs de configuration sont la première cause des défaillances de services sur l'Internet

Architecture logicielle

- Description globale d'un système
  - Composants, connecteurs, configurations
  - Règles de composition (cf. plus loin)
  - Langage de description d'architecture (ADL)
    - Base pour construction, vérification et preuve, administration
- L'architecture logicielle, base pour l'administration
  - Configuration
  - Déploiement
  - Reconfiguration

Les erreurs de configuration sont la première cause des défaillances de services sur l'Internet

- Vers une formalisation de l'administration de systèmes
  - Installation "saine" de logiciel libre
  - Réaction autonome aux pannes et surcharges

Architecture logicielle

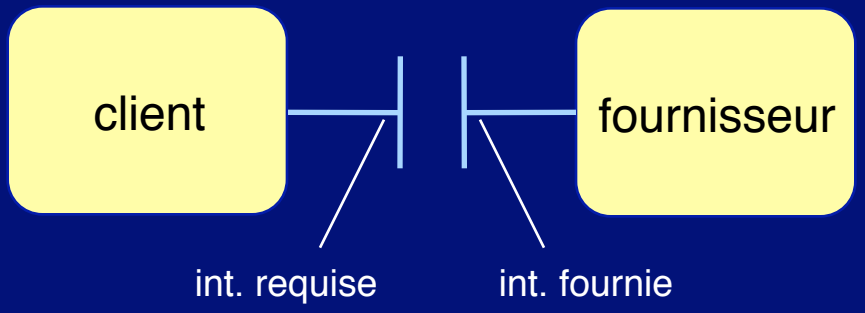
- Description globale d'un système
  - Composants, connecteurs, configurations
  - Règles de composition (cf. plus loin)
  - Langage de description d'architecture (ADL)
    - Base pour construction, vérification et preuve, administration
- L'architecture logicielle, base pour l'administration
  - Configuration
  - Déploiement
  - Reconfiguration

Mary Shaw, David Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, 1996

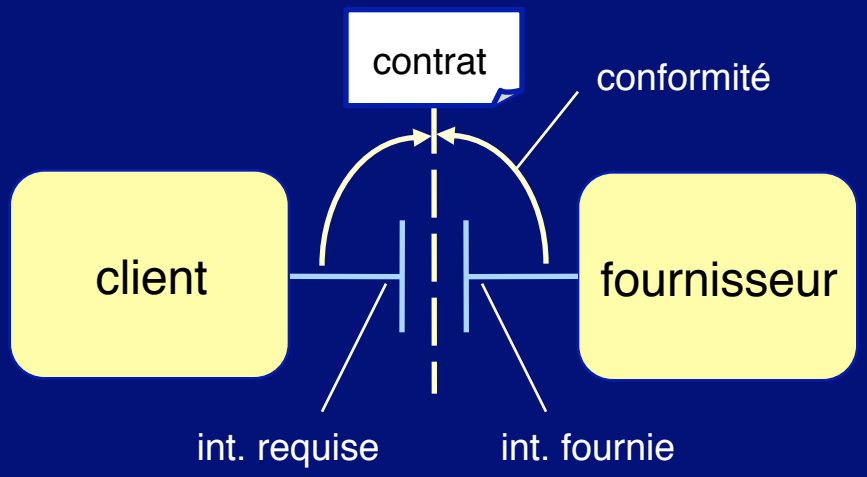
- Vers une formalisation de l'administration de systèmes
  - Installation "saine" de logiciel libre
  - Réaction autonome aux pannes et surcharges

Interfaces

Interfaces



Interfaces

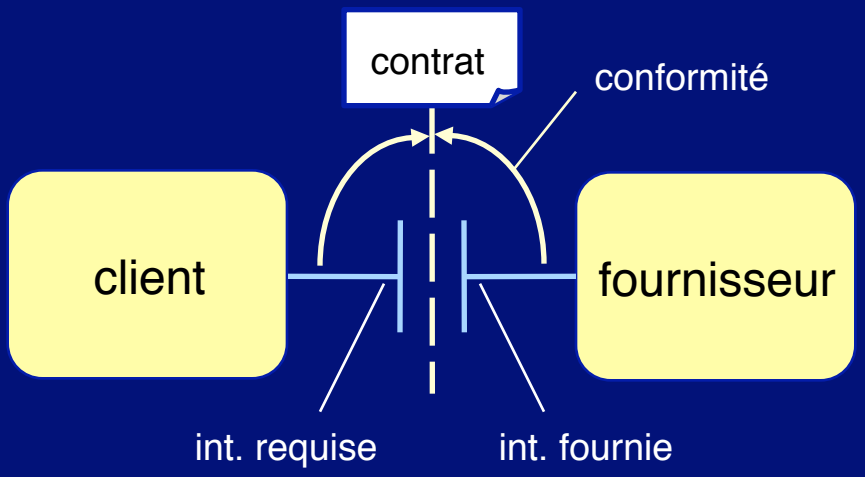


Contrat = définition de la conformité (compatibilité entre interfaces)

Au delà de l'interface, chaque composant est une boîte noire pour l'autre

Échange standard si contrat respecté

Interfaces



Contrat = définition de la conformité (compatibilité entre interfaces)

Au delà de l'interface, chaque composant est une boîte noire pour l'autre

Échange standard si contrat respecté

- Contrat d'interface
  - Syntaxe : langage de définition d'interface (IDL)
    - Conformité de types
  - Sémantique : plusieurs niveaux
    - Spécification des "méthodes"
    - Synchronisation
    - Propriétés non fonctionnelles (encore peu formalisé)

La virtualisation, transformateur d'interfaces

Virtualiser une entité = transformer son interface, en fournir autant d'exemplaires que nécessaire, cacher sa réalisation

La virtualisation, transformateur d'interfaces

Virtualiser une entité = transformer son interface, en fournir autant d'exemplaires que nécessaire, cacher sa réalisation

- On peut virtualiser …
  - … un système logiciel
    - hiérarchie de "machines abstraites"
  - … une ressource isolée
    - processeur ~ fil d'exécution (*thread),* mémoire ~ mémoire virtuelle, disque ~ fichier, écran ~ fenêtre, …
  - … une machine entière
  - … un réseau

La virtualisation, transformateur d'interfaces

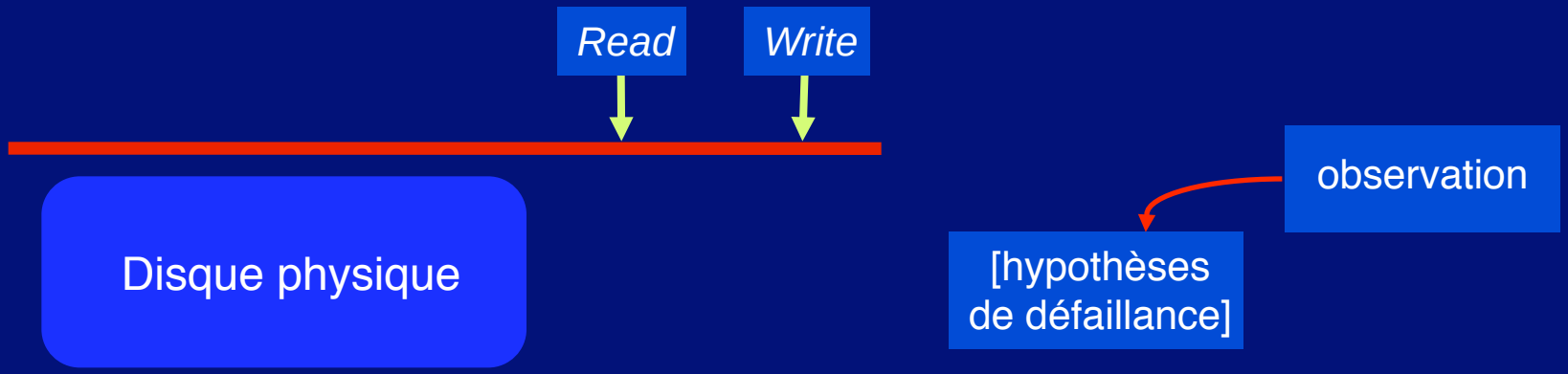
Virtualiser une entité = transformer son interface, en fournir autant d'exemplaires que nécessaire, cacher sa réalisation

- On peut virtualiser …
  - … un système logiciel
    - hiérarchie de "machines abstraites"
  - … une ressource isolée
    - processeur ~ fil d'exécution (*thread),* mémoire ~ mémoire virtuelle, disque ~ fichier, écran ~ fenêtre, …
  - … une machine entière
  - … un réseau

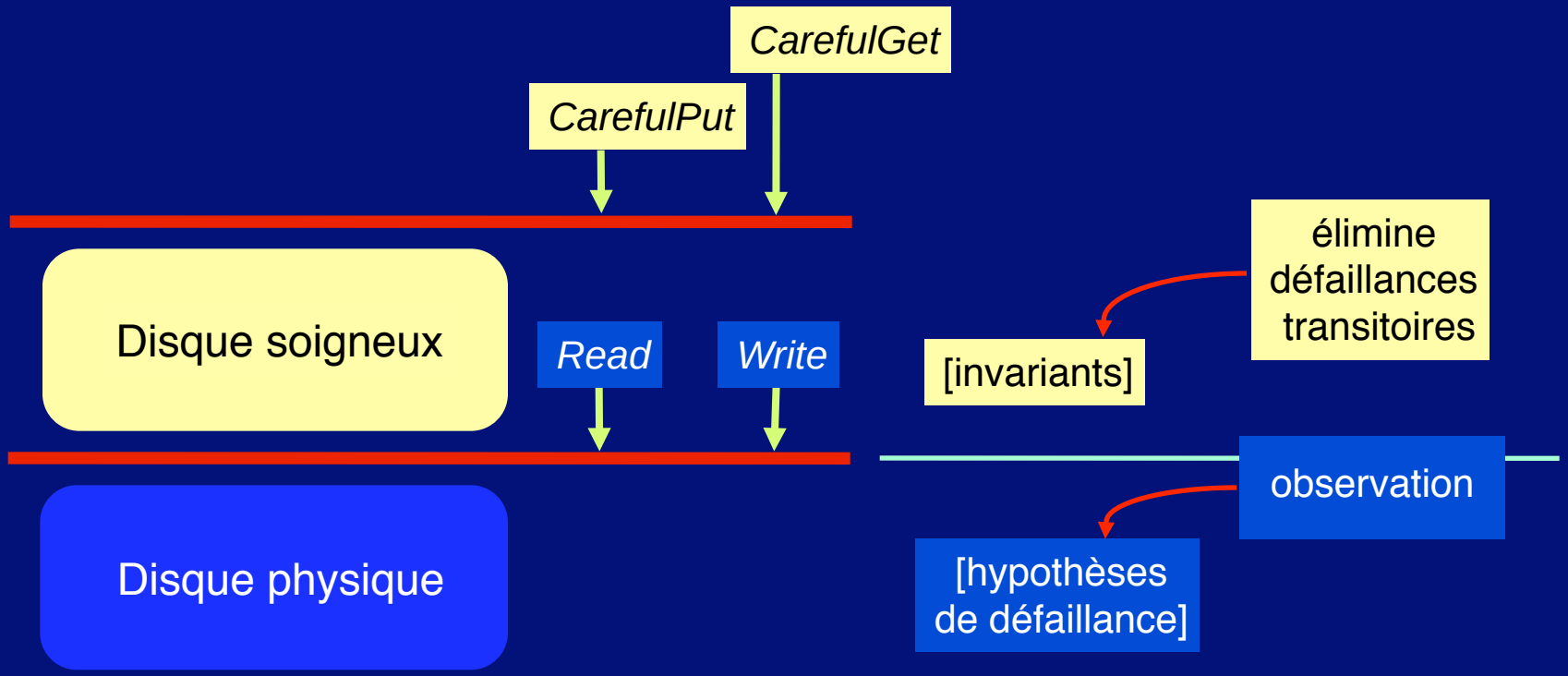
James E. Smith, Ravi Nair, *Virtual Machines*, Morgan Kaufmann, 2005

- Les machines virtuelles
  - machine virtuelle différente de la machine physique : JVM, CLI
  - machine virtuelle identique (ou presque) à la machine physique
    - CP-67, VM 370, … , Xen (Citrix), VMware
  - environnement d'exécution (*cloud computing*)

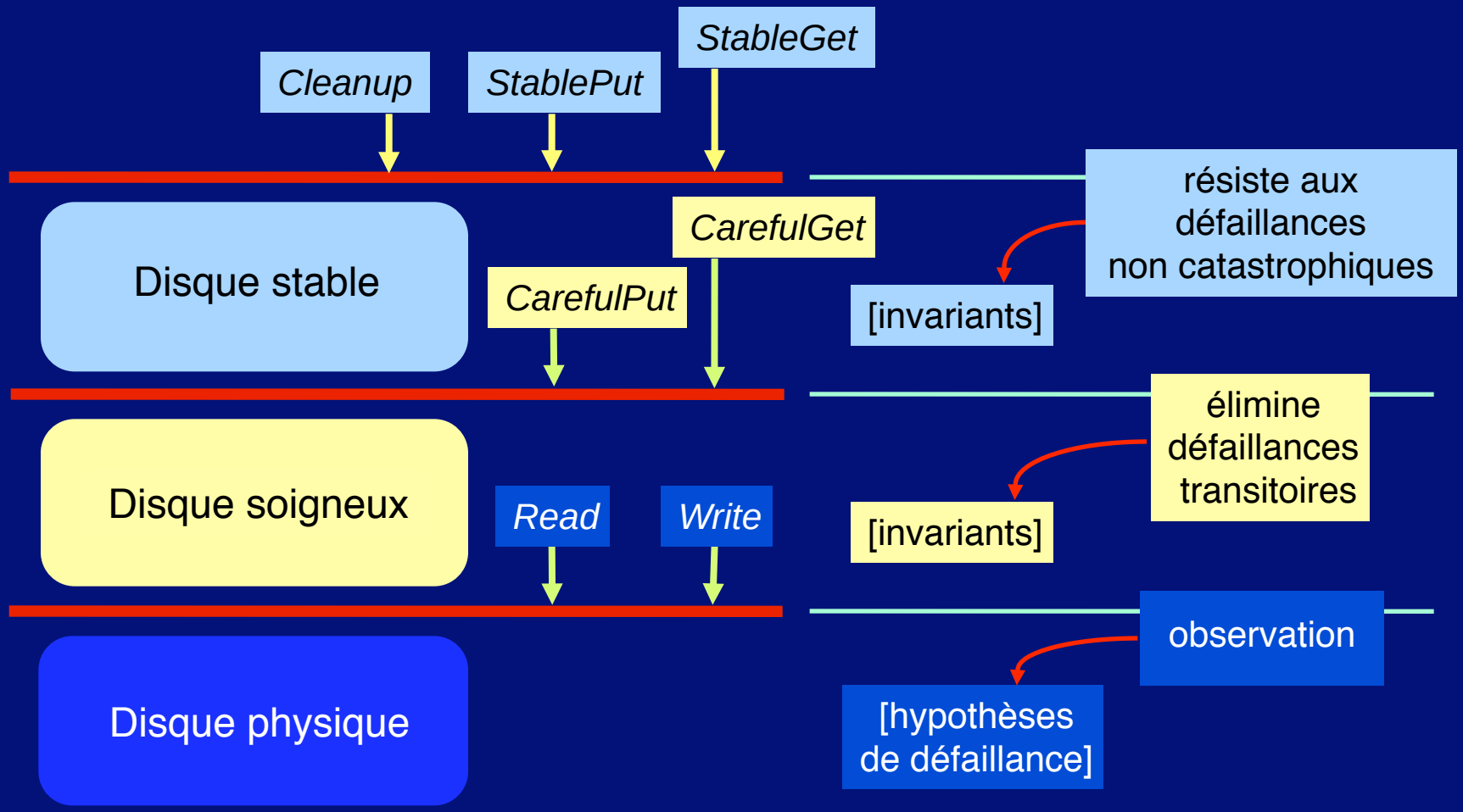
Exemple de virtualisation : la mémoire stable



Exemple de virtualisation : la mémoire stable

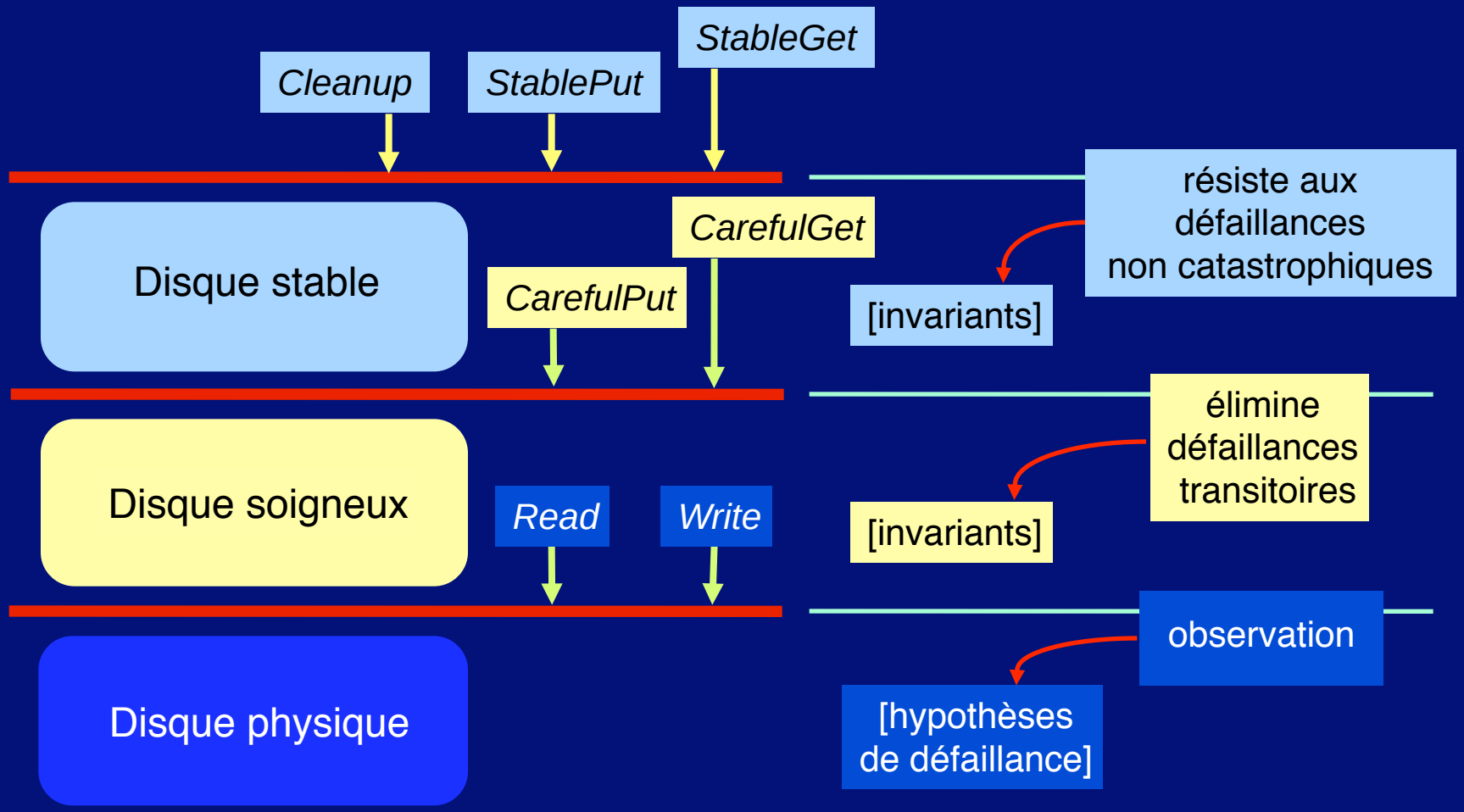


Exemple de virtualisation : la mémoire stable



B. W. Lampson and H. E. Sturgis. Crash Recovery in a Distributed Data Storage System, unpublished technical report, Xerox PARC, June 1979, 25 pp.

Exemple de virtualisation : la mémoire stable



B. W. Lampson, “Atomic Transactions”, in *Distributed Systems—Architecture and Implementation*, Lampson, Paul, and Siegert (editors), LNCS 105, Springer, 1981, pp. 246-265 and 357-370.

En quête de la validité …

Pour un composant élémentaire

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

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

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

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

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)

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

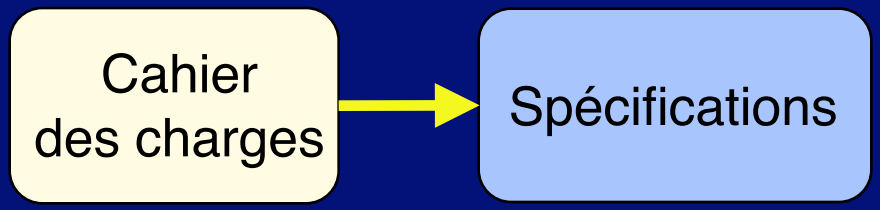
Vérification par analyse statique

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

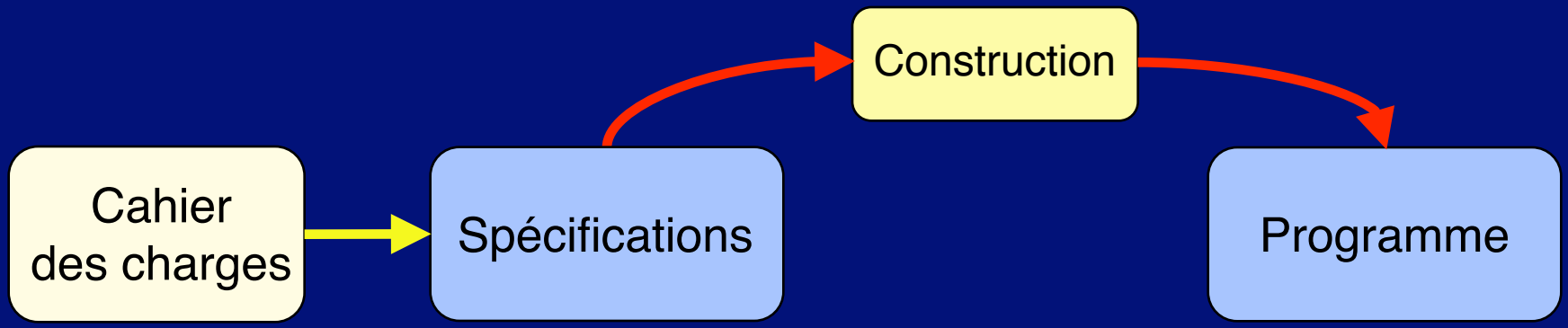
- *Model checking* (Clarke, Emerson, Sifakis)
  - Modéliser le système par un graphe de transition d'états
  - Vérifier que le modèle satisfait une spécification (en logique temporelle)
- Interprétation abstraite (Cousot)
  - Modéliser l'évolution du système par ses traces d'exécution
  - Définir des sémantiques à divers niveaux d'approximation
  - Résoudre l'équation (de point fixe) traduisant une propriété
- Difficultés communes
  - Explosion des états
  - Vérification de systèmes composés

Spécification, construction, preuve

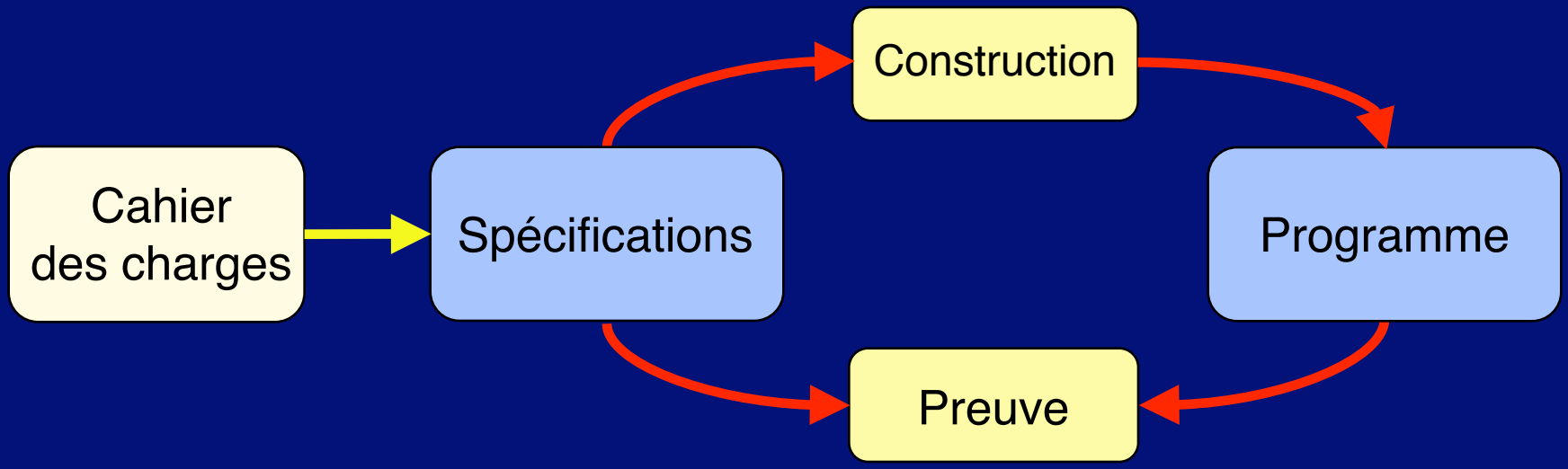
Spécification, construction, preuve



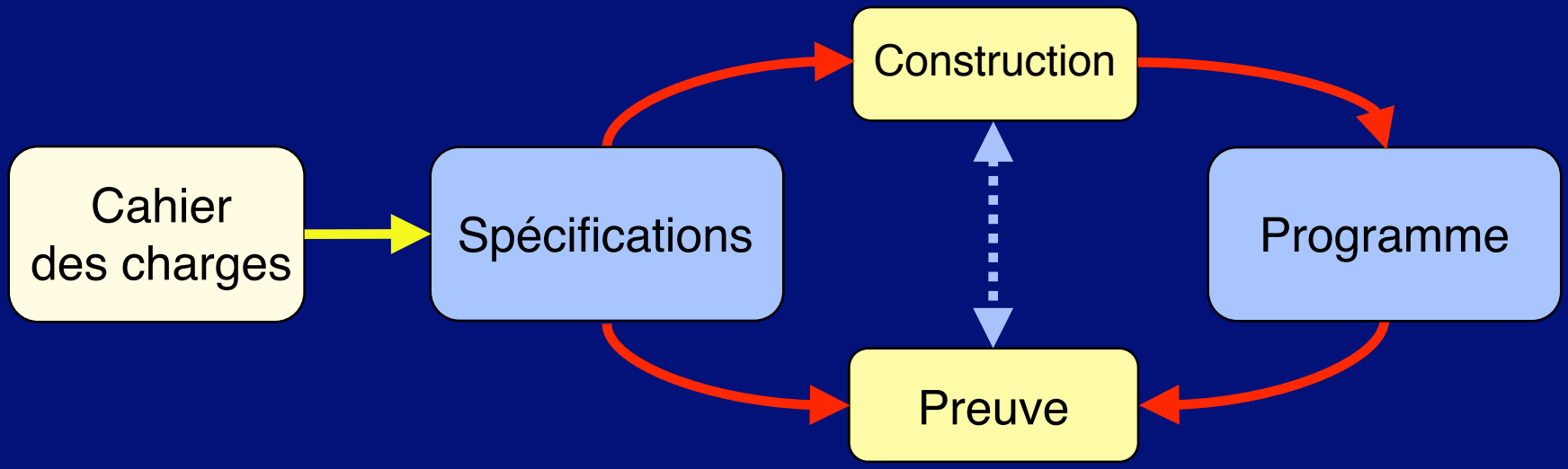
Spécification, construction, preuve



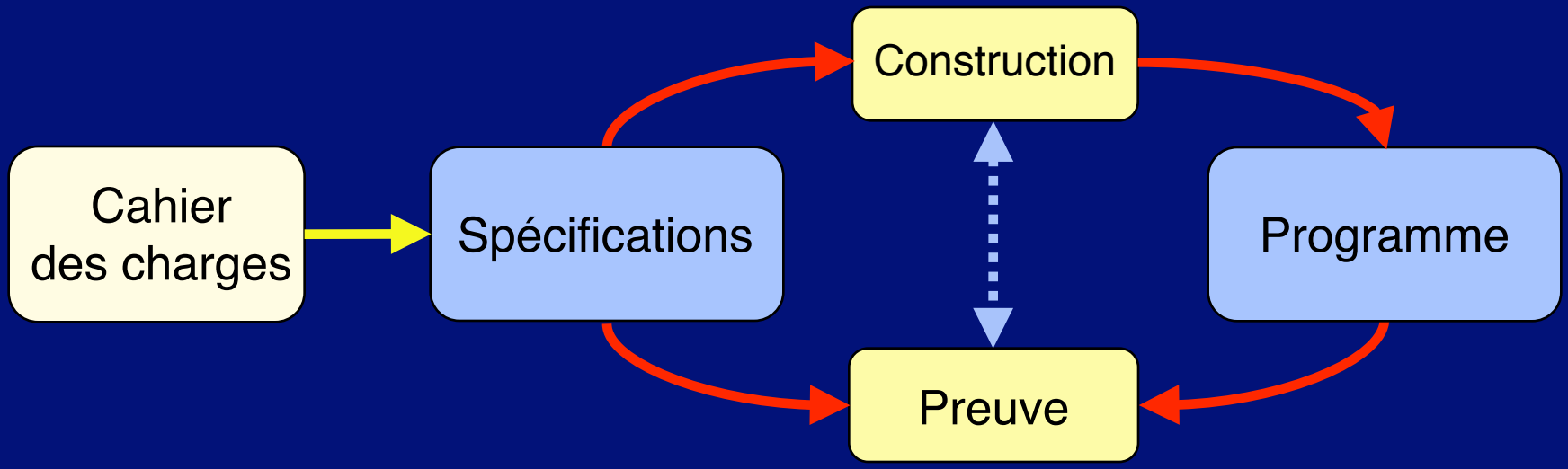
Spécification, construction, preuve



Spécification, construction, preuve

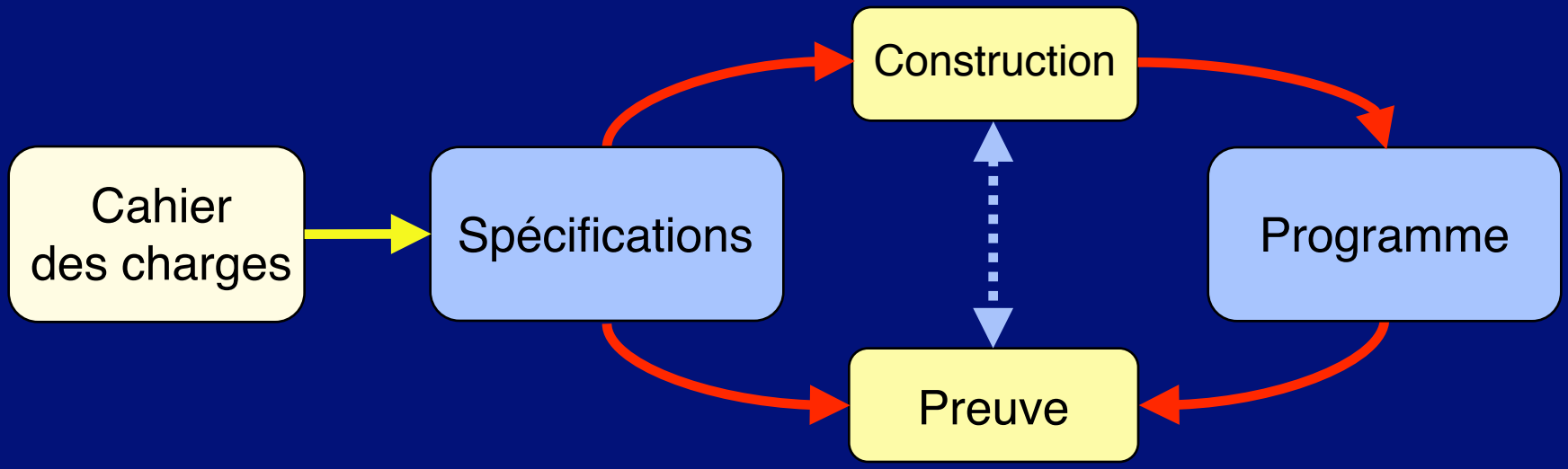


Spécification, construction, preuve



- 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

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

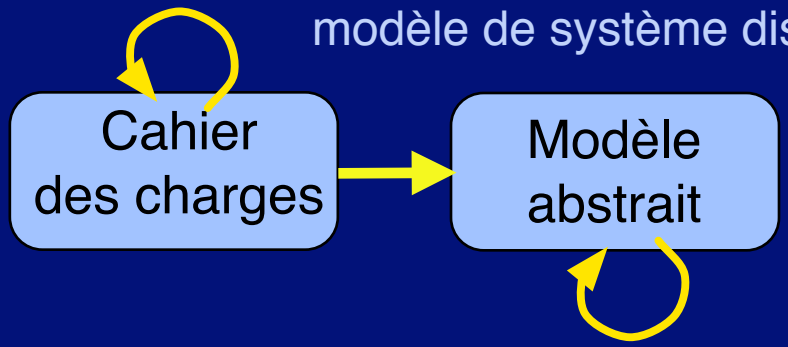
Pendant longtemps :
pas de base formelle
langage naturel
pas ou peu d'outils

B, une méthode de construction de programmes

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

B, une méthode de construction de programmes

Machine à transition d'états, modèle de système discret



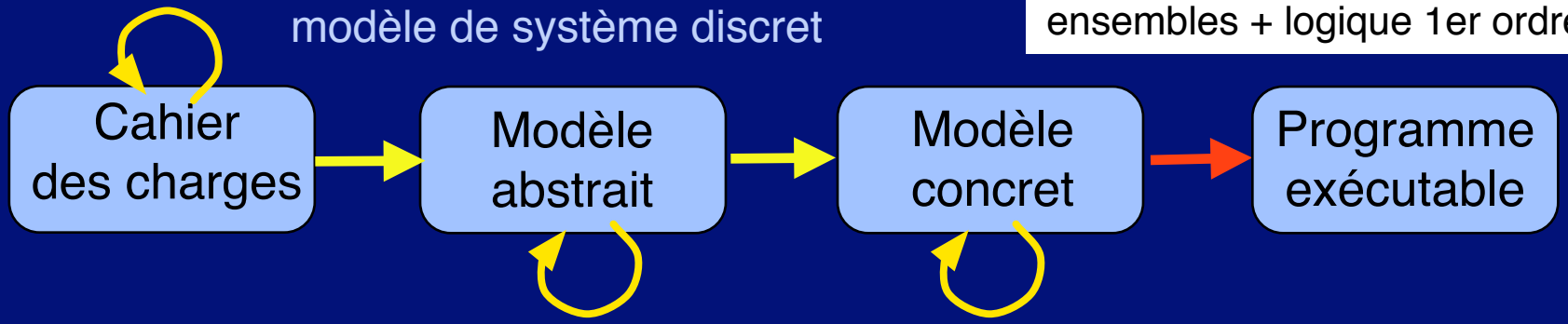
Bases du modèle : théorie des ensembles + logique 1er ordre

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

B, une méthode de construction de programmes

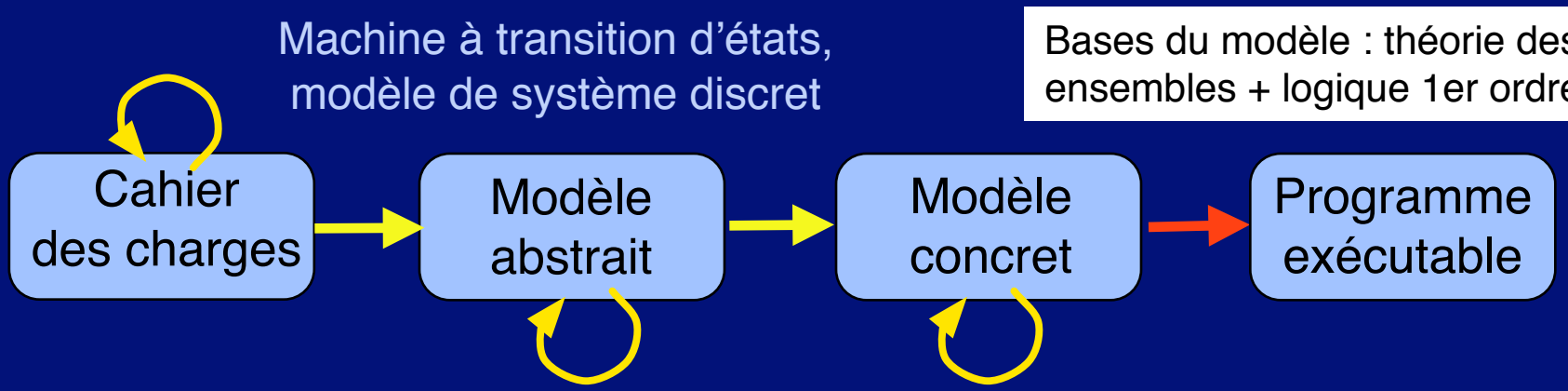
Machine à transition d'états, modèle de système discret

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



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

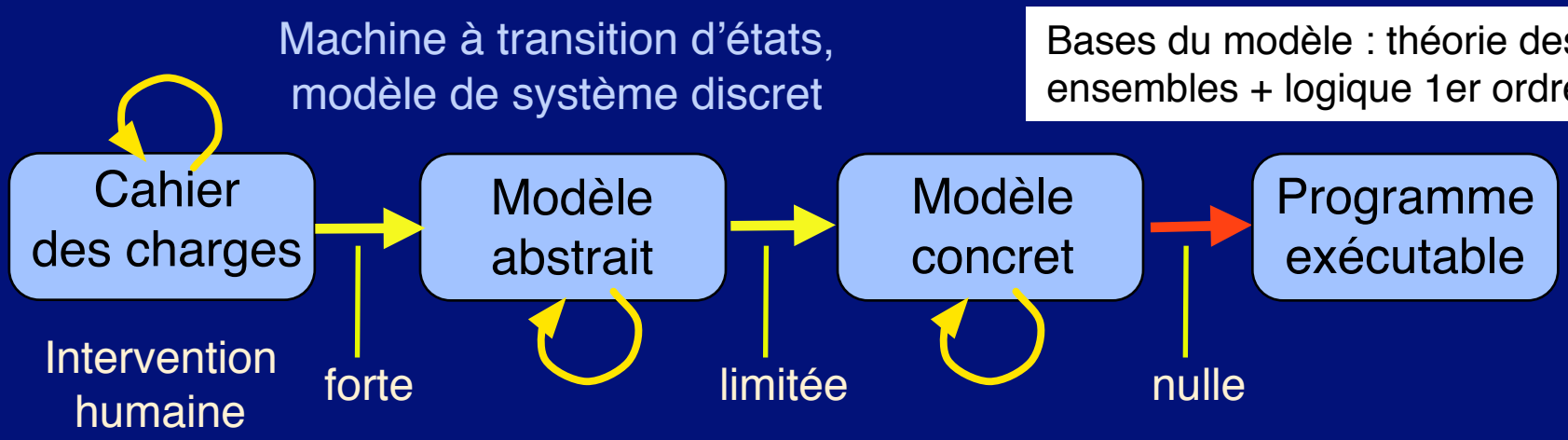
B, une méthode de construction de programmes



- 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

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

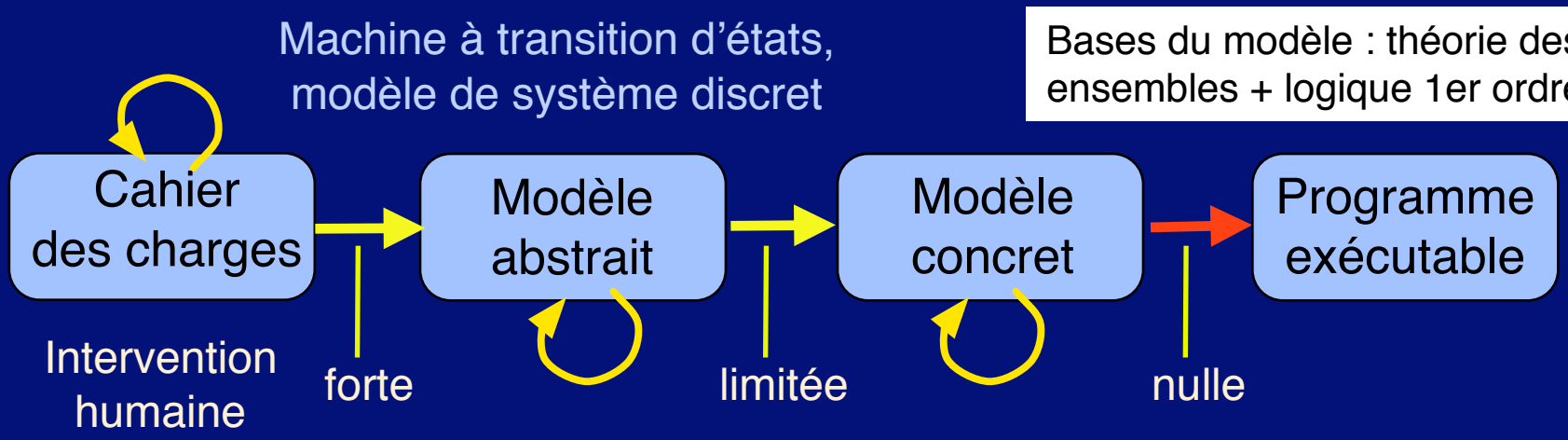
B, une méthode de construction de programmes



- 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

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

B, une méthode de construction de programmes



- 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

Coq, un assistant de preuve

Coq, un assistant de preuve

- 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 !)

Coq, un assistant de preuve

- 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

Vérification d'un compilateur

- Un compilateur pour un large sous-ensemble de C

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

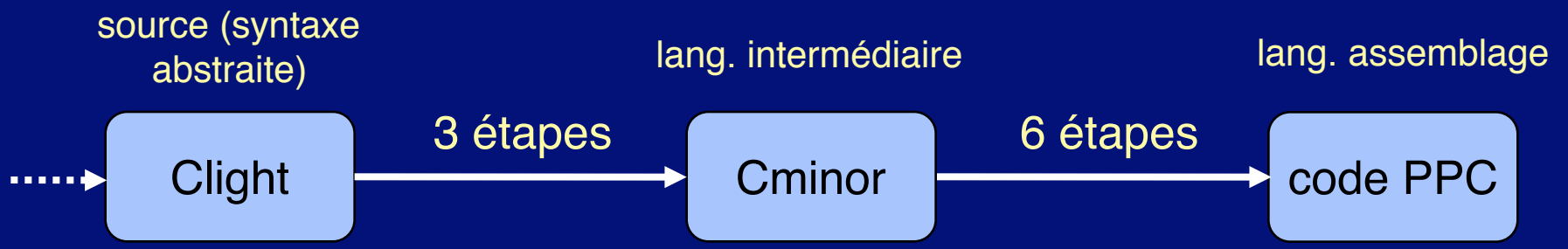
Vérification d'un compilateur

- Un compilateur pour un large sous-ensemble de C

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

Vérification d'un compilateur

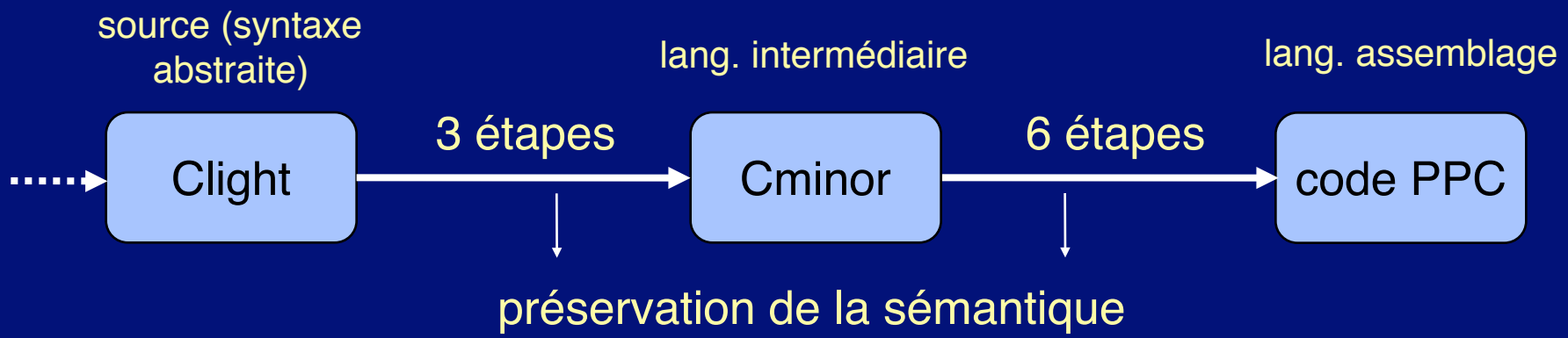
- Un compilateur pour un large sous-ensemble de C



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

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

Vérification d'un noyau de système d'exploitation

- Une version du micro-noyau L4

Gerwin Klein et al., "seL4: Formal Verification of an OS kernel", *Proc. 22nd ACM Symposium on Operating Systems Principles* (*SOSP 2009*), Big Sky, 11-14 Oct. 2009

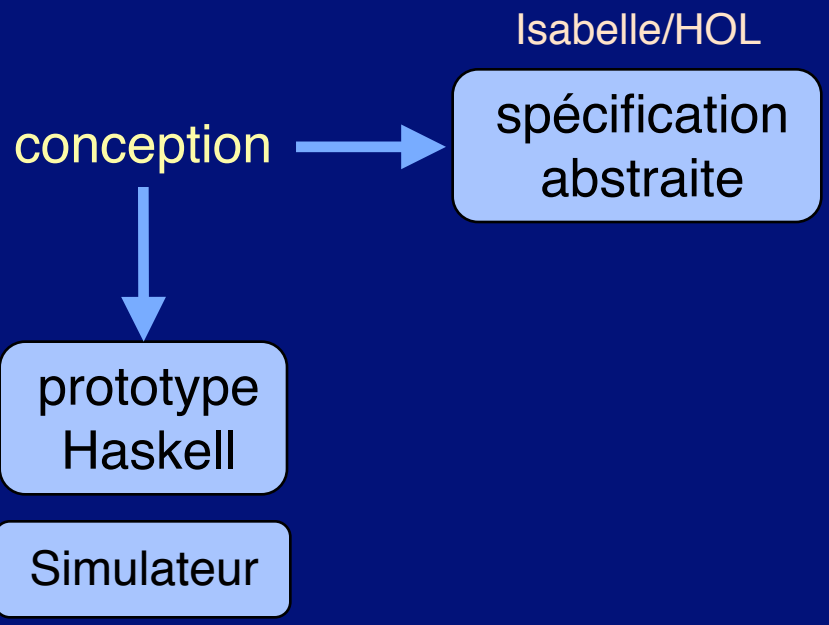
Vérification d'un noyau de système d'exploitation

- Une version du micro-noyau L4
- Difficultés
  - Asynchronisme
  - Gestion de la mémoire (pointeurs)
  - Accès direct aux fonctions du matériel (MMU, …)

Gerwin Klein et al., "seL4: Formal Verification of an OS kernel", *Proc. 22nd ACM Symposium on Operating Systems Principles* (*SOSP 2009*), Big Sky, 11-14 Oct. 2009

Vérification d'un noyau de système d'exploitation

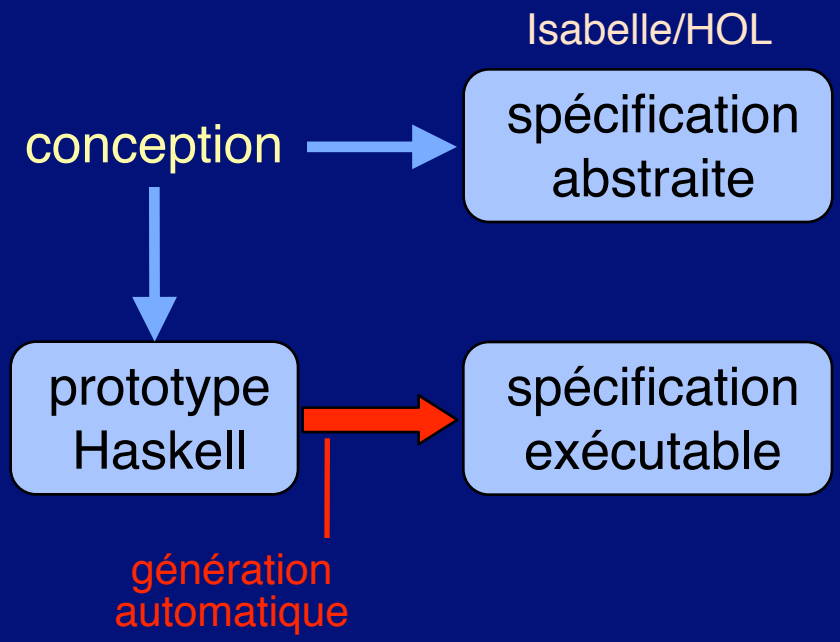
- Une version du micro-noyau L4
- Difficultés
  - Asynchronisme
  - Gestion de la mémoire (pointeurs)
  - Accès direct aux fonctions du matériel (MMU, …)



Gerwin Klein et al., "seL4: Formal Verification of an OS kernel", *Proc. 22nd ACM Symposium on Operating Systems Principles* (*SOSP 2009*), Big Sky, 11-14 Oct. 2009

Vérification d'un noyau de système d'exploitation

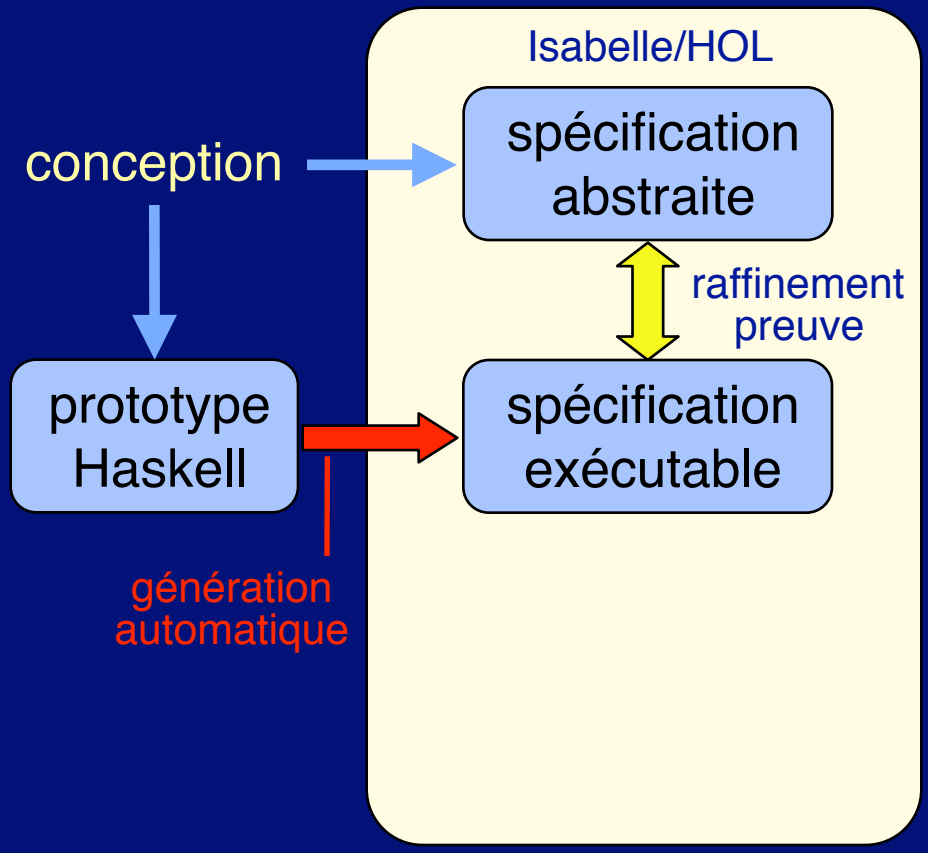
- Une version du micro-noyau L4
- Difficultés
  - Asynchronisme
  - Gestion de la mémoire (pointeurs)
  - Accès direct aux fonctions du matériel (MMU, …)



Gerwin Klein et al., "seL4: Formal Verification of an OS kernel", *Proc. 22nd ACM Symposium on Operating Systems Principles* (*SOSP 2009*), Big Sky, 11-14 Oct. 2009

Vérification d'un noyau de système d'exploitation

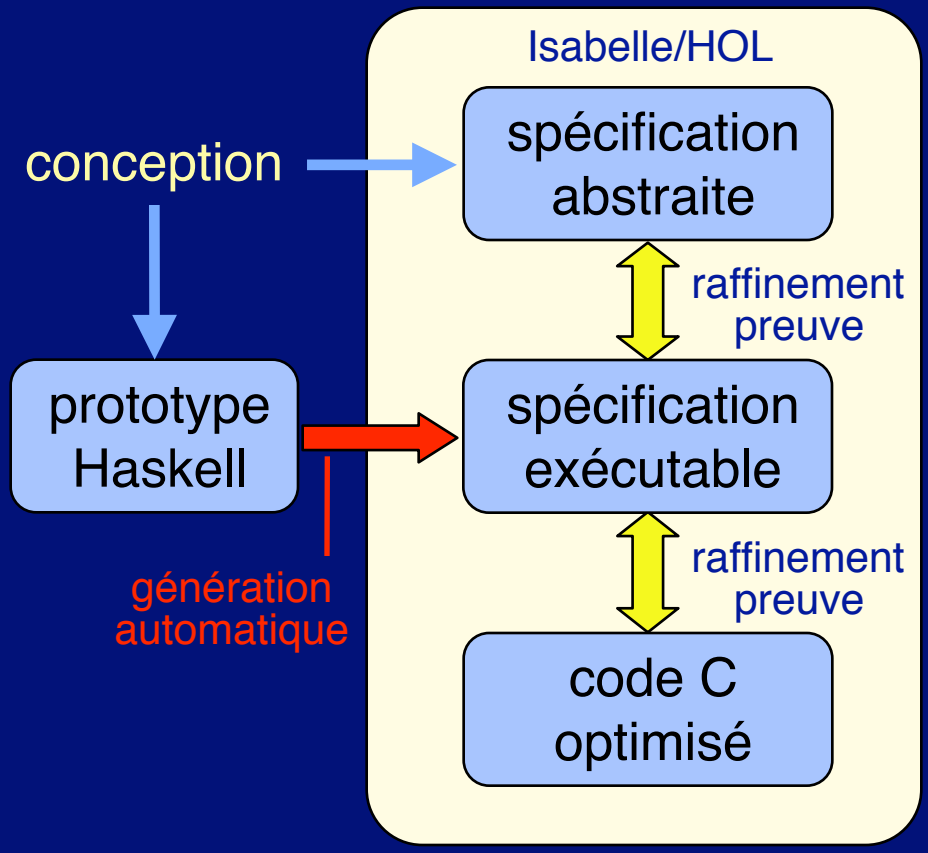
- Une version du micro-noyau L4
- Difficultés
  - Asynchronisme
  - Gestion de la mémoire (pointeurs)
  - Accès direct aux fonctions du matériel (MMU, …)



Gerwin Klein et al., "seL4: Formal Verification of an OS kernel", *Proc. 22nd ACM Symposium on Operating Systems Principles* (*SOSP 2009*), Big Sky, 11-14 Oct. 2009

Vérification d'un noyau de système d'exploitation

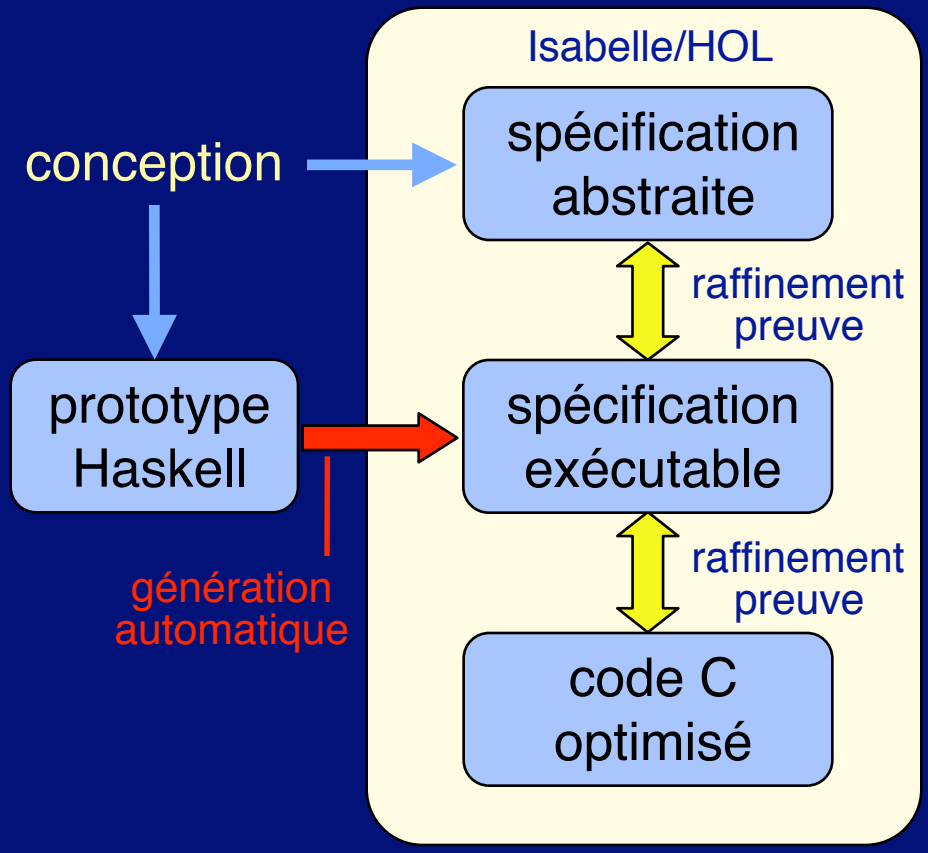
- Une version du micro-noyau L4
- Difficultés
  - Asynchronisme
  - Gestion de la mémoire (pointeurs)
  - Accès direct aux fonctions du matériel (MMU, …)



Gerwin Klein et al., "seL4: Formal Verification of an OS kernel", *Proc. 22nd ACM Symposium on Operating Systems Principles* (*SOSP 2009*), Big Sky, 11-14 Oct. 2009

Vérification d'un noyau de système d'exploitation

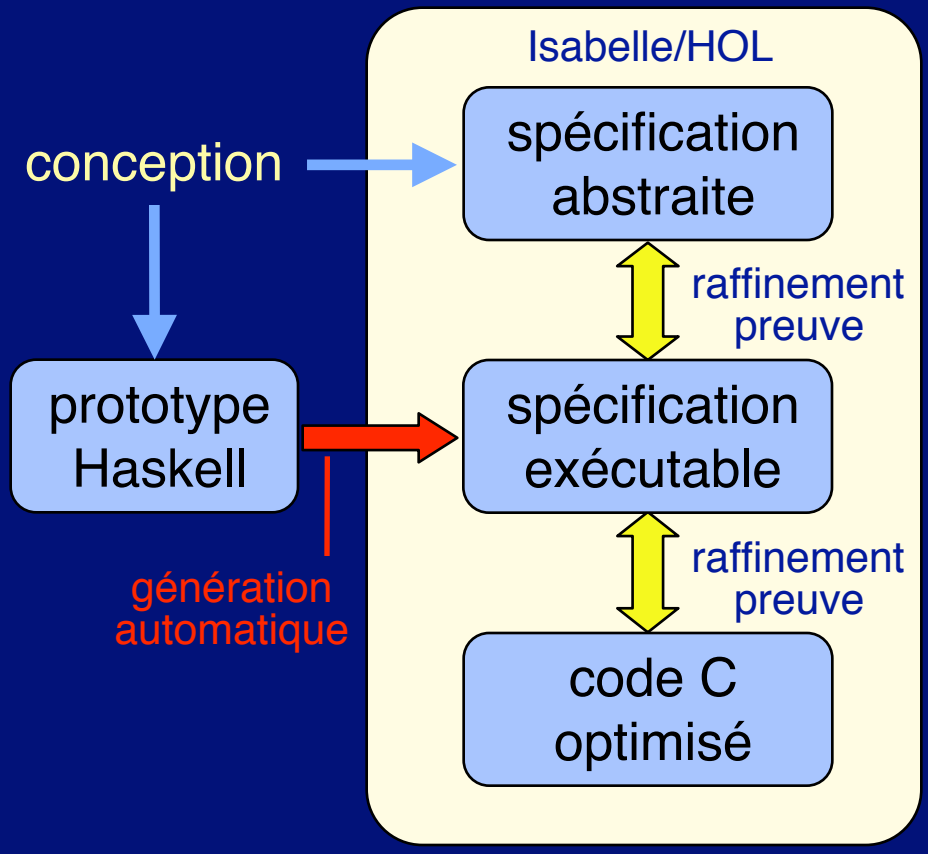
- Une version du micro-noyau L4
- Difficultés
  - Asynchronisme
  - Gestion de la mémoire (pointeurs)
  - Accès direct aux fonctions du matériel (MMU, …)
- Quelques remèdes
  - Monoprocesseur
  - Gestion des interruptions par scrutation (*polling*)
  - Modèle de la machine cible
  - Restrictions sur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



Gerwin Klein et al., "seL4: Formal Verification of an OS kernel", *Proc. 22nd ACM Symposium on Operating Systems Principles* (*SOSP 2009*), Big Sky, 11-14 Oct. 2009

Vérification d'un noyau de système d'exploitation

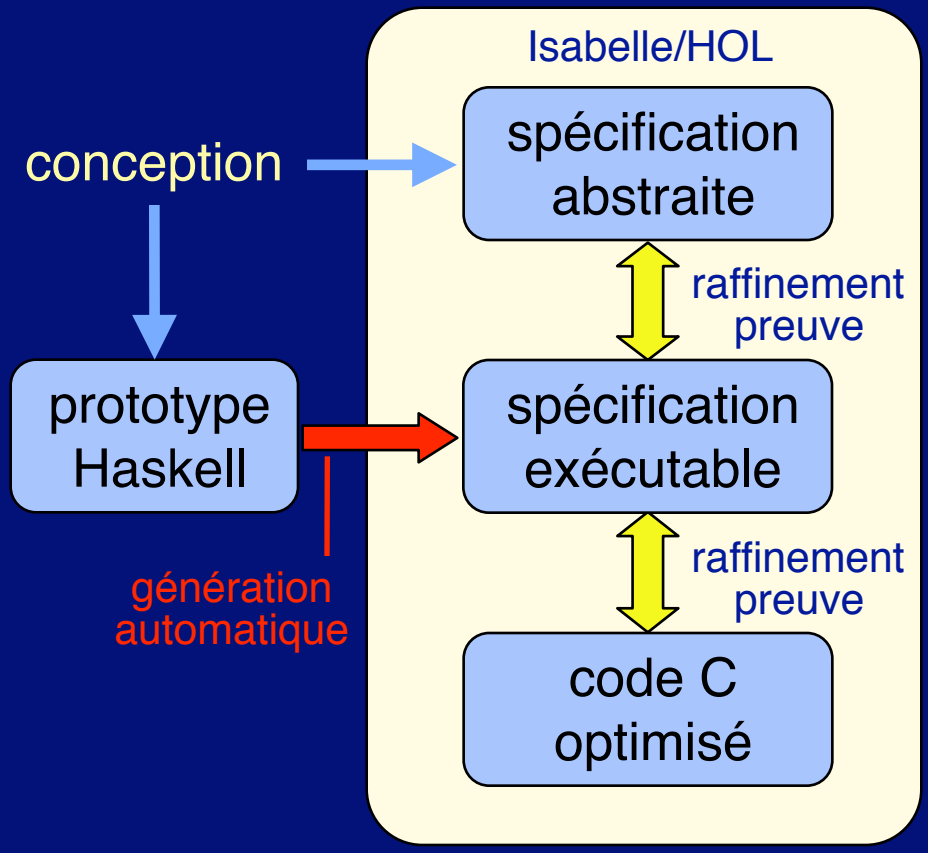
- Une version du micro-noyau L4
- Difficultés
  - Asynchronisme
  - Gestion de la mémoire (pointeurs)
  - Accès direct aux fonctions du matériel (MMU, …)
- Quelques remèdes
  - Monoprocesseur
  - Gestion des interruptions par scrutation (*polling*)
  - Modèle de la machine cible
  - Restrictions sur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

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



Gerwin Klein et al., "seL4: Formal Verification of an OS kernel", *Proc. 22nd ACM Symposium on Operating Systems Principles* (*SOSP 2009*), Big Sky, 11-14 Oct. 2009

Comprendre et domestiquer le parallélisme

Comprendre et domestiquer le parallélisme

- Parallélisme = concurrence + asynchronisme

Comprendre et domestiquer le parallélisme

- Parallélisme = concurrence + asynchronisme
  - Concurrence : plusieurs activités se déroulent en même temps
    - Problème : ordonnancement (allouer les ressources aux activités)

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)

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

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

Le parallélisme, un univers fragmenté …

Le parallélisme, un univers fragmenté …

- Modèles du parallélisme
  - Logique temporelle
  - Calculs de processus
  - Systèmes et langages synchrones

Le parallélisme, un univers fragmenté …

- Modèles du parallélisme
  - Logique temporelle
  - Calculs de processus
  - Systèmes et langages synchrones
- Méthodes et outils pour maîtriser l'asynchronisme
  - Mécanismes de synchronisation
  - Langages pour le parallélisme
  - Algorithmique parallèle à grain fin
  - Programmation des multiprocesseurs

Le parallélisme, un univers fragmenté …

- Modèles du parallélisme
  - Logique temporelle
  - Calculs de processus
  - Systèmes et langages synchrones
- Méthodes et outils pour maîtriser l'asynchronisme
  - Mécanismes de synchronisation
  - Langages pour le parallélisme
  - Algorithmique parallèle à grain fin
  - Programmation des multiprocesseurs
- Calcul à hautes performances
  - Parallélisation des applications
  - Outils pour le calcul parallèle à grande échelle

Le parallélisme, un univers fragmenté …

- Modèles du parallélisme
  - Logique temporelle
  - Calculs de processus
  - Systèmes et langages synchrones
- Méthodes et outils pour maîtriser l'asynchronisme
  - Mécanismes de synchronisation
  - Langages pour le parallélisme
  - Algorithmique parallèle à grain fin
  - Programmation des multiprocesseurs
- Calcul à hautes performances
  - Parallélisation des applications
  - Outils pour le calcul parallèle à grande échelle
- Applications réparties à bas coût
  - Mobiles, réseaux de capteurs, informatique ubiquitaire, …

Asynchronisme : premières avancées

Asynchronisme : premières avancées

- Les origines
  - Comportement non déterministe des systèmes d'exploitation (attribué aux défaillances du matériel)

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 ll B équivaut à (A ; B) ou (B ; A)
  - Synchronisation : contraintes sur l'ordre des événements
  - Mécanismes élémentaires de synchronisation : verrous, sémaphores, moniteurs

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 ll 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

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

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)

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 languages 12 years later", *Proceedings of the IEEE*, 11:1, Jan 2003, pp. 64 - 83

Modèles du parallélisme

Modèles du parallélisme

- Calculs de processus
  - Équivalent du λ-calcul en séquentiel
  - Communication par canaux et messages
    - Transmission de canaux et processus (reconfiguration, mobilité)
  - Un concept de base : la bisimulation (équivalence)
  - Exemples : CSP, CCS, Lotos, π-calcul, Ambients, …

Modèles du parallélisme

- Calculs de processus
  - Équivalent du λ-calcul en séquentiel
  - Communication par canaux et messages
    - Transmission de canaux et processus (reconfiguration, mobilité)
  - Un concept de base : la bisimulation (équivalence)
  - Exemples : CSP, CCS, Lotos, π-calcul, Ambients, …
- Impact pratique
  - Barrières
    - Expressivité limitée, apprentissage difficile
    - Nombreux langages voisins, choix difficile
  - Avancées
    - Outils de vérification utilisés dans l'industrie …
    - ... mais encore peu d'impact sur la programmation parallèle

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

Matrices denses
Matrices creuses
FFT et assimilés
Combinatoire
Machines états finis

Patrons et canevas

MapReduce
Traversée graphes
Programmation dynamique
Branch & Bound
N-corps
Grilles

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
Matrices denses
Matrices creuses
FFT et assimilés
Combinatoire
Machines états finis

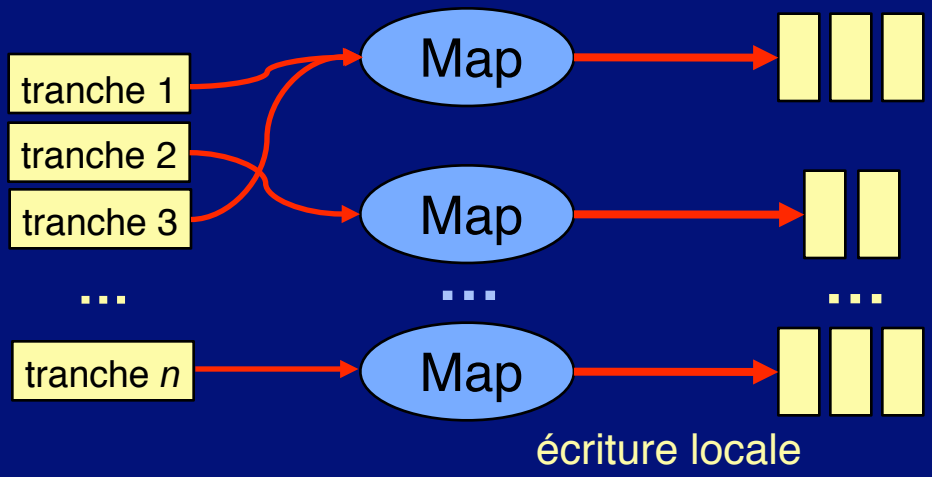
Patrons et canevas
MapReduce
Traversée graphes
Programmation dynamique
Branch & Bound
N-corps
Grilles

- 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

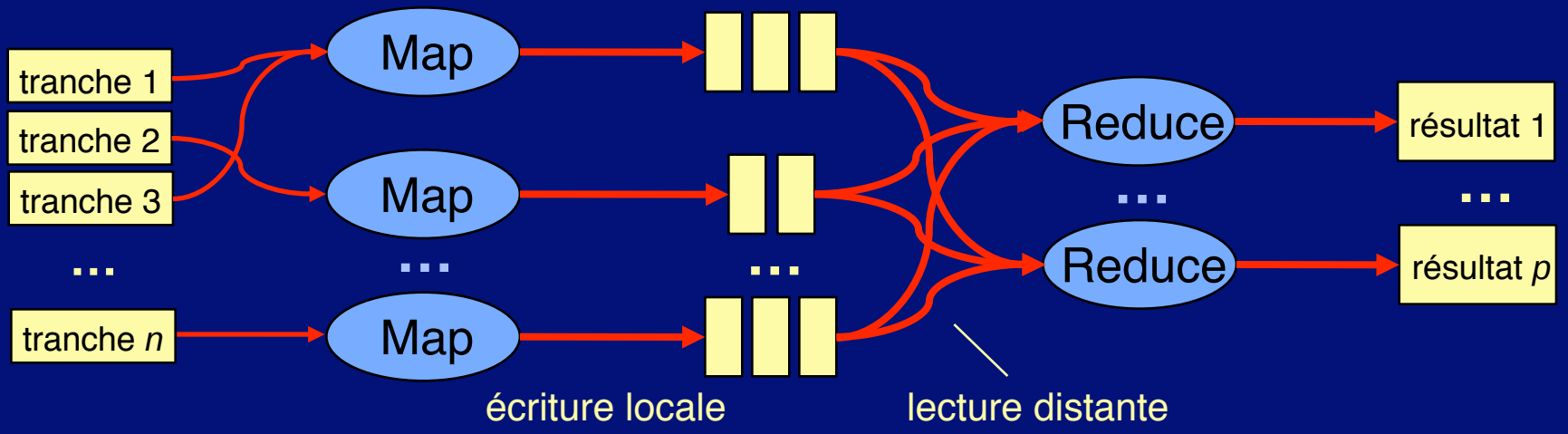
Jeffrey Dean, Sanjay Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters”, *Proc. ACM Conf. on Operating Systems Design & Implementation* (*OSDI*), 2004

Un canevas pour le calcul parallèle : MapReduce



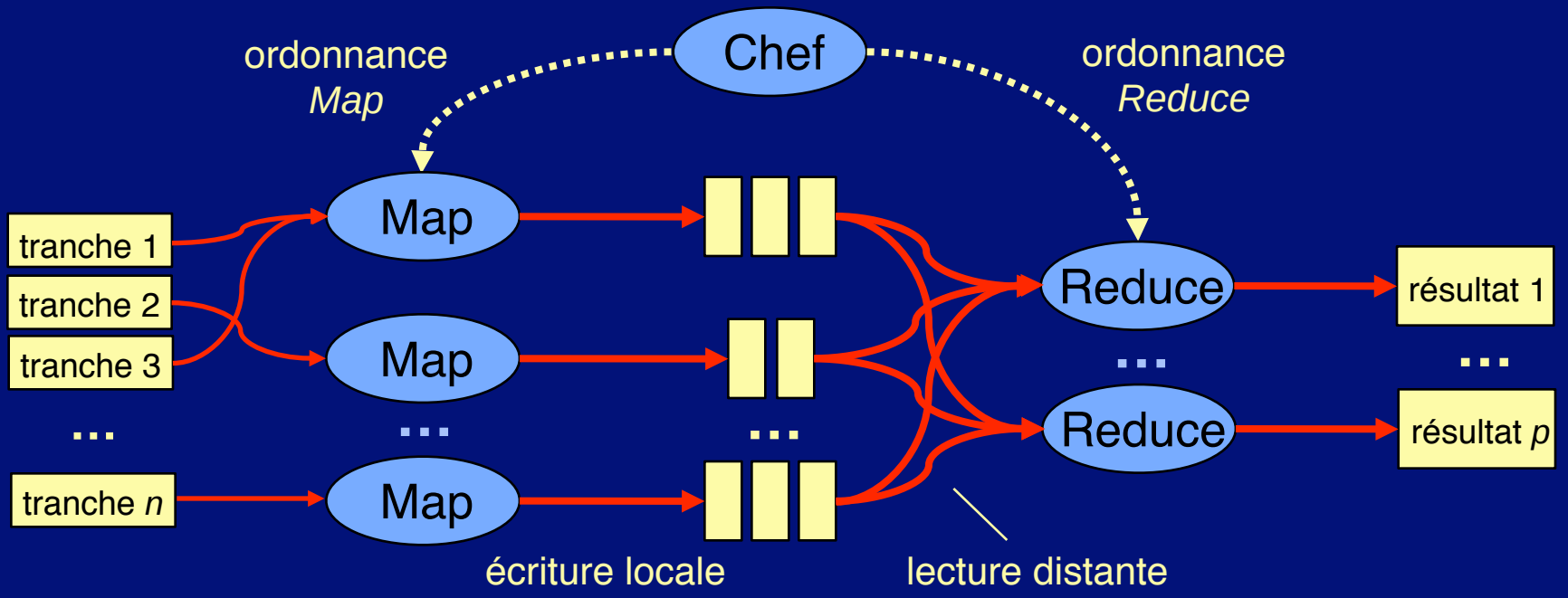
Jeffrey Dean, Sanjay Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters”, *Proc. ACM Conf. on Operating Systems Design & Implementation* (*OSDI*), 2004

Un canevas pour le calcul parallèle : MapReduce



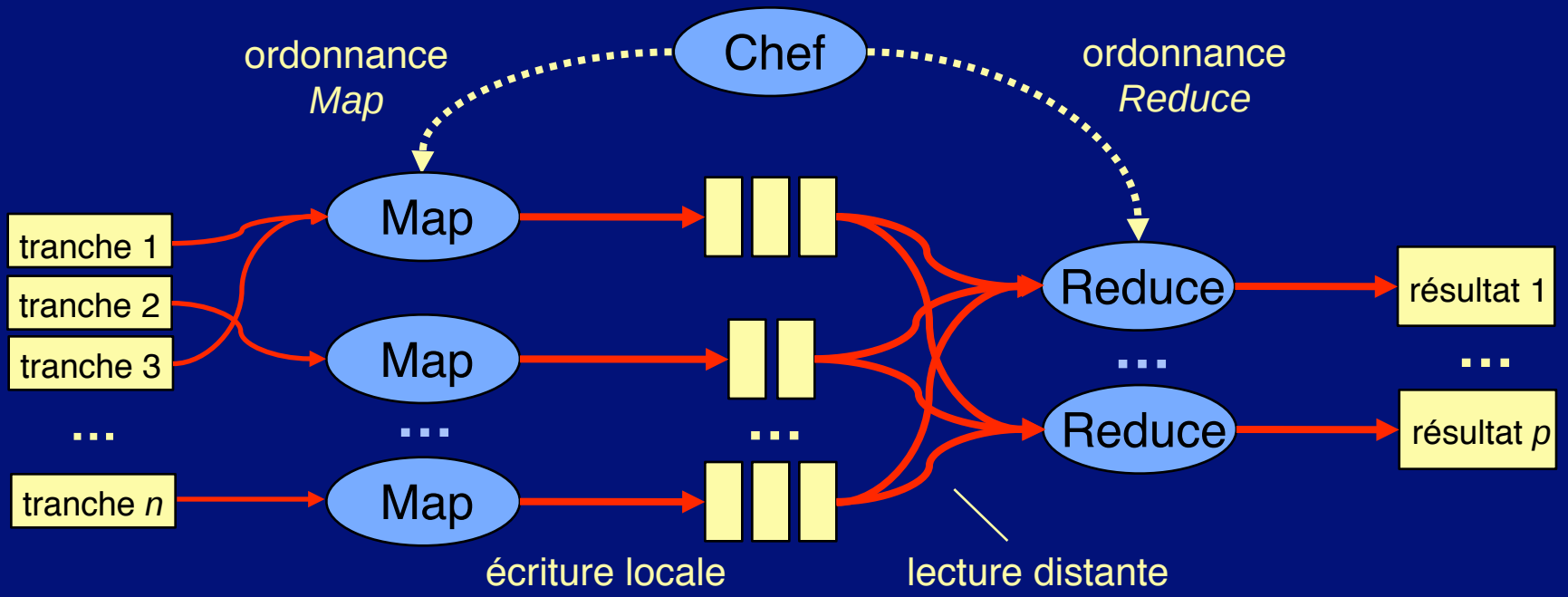
Jeffrey Dean, Sanjay Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters”, *Proc. ACM Conf. on Operating Systems Design & Implementation* (*OSDI*), 2004

Un canevas pour le calcul parallèle : MapReduce



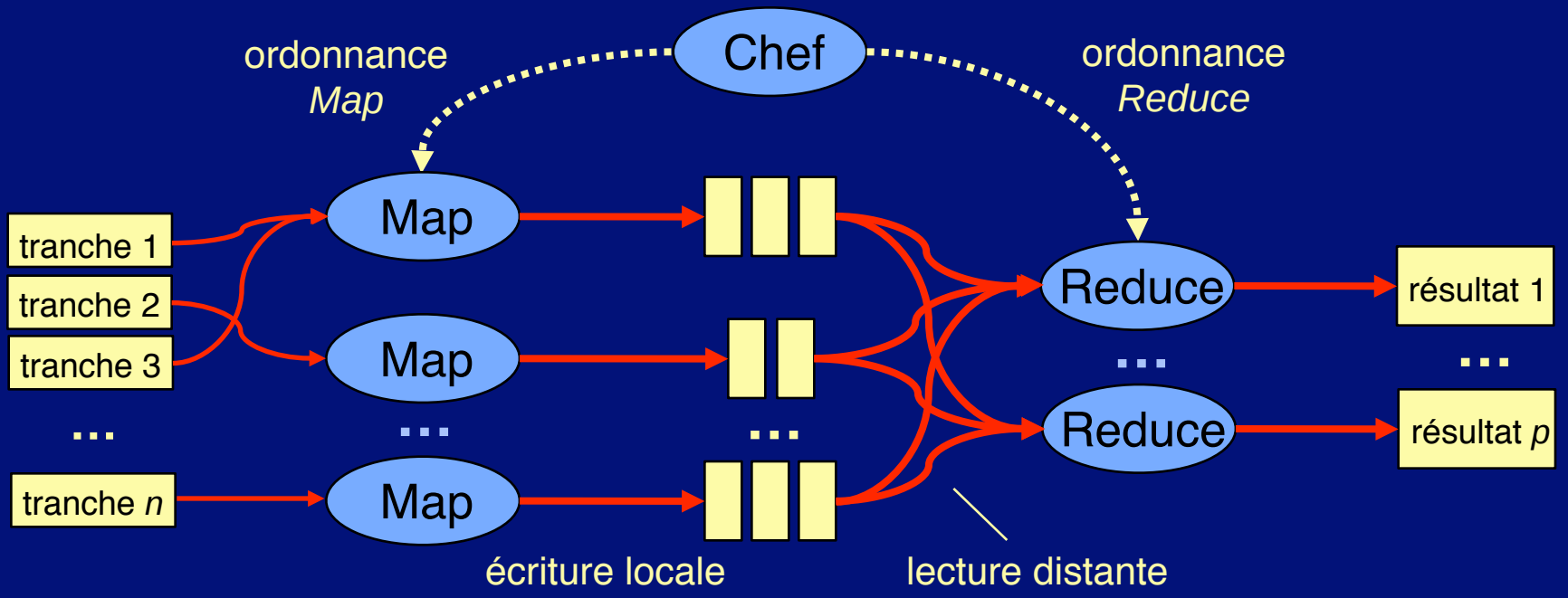
Jeffrey Dean, Sanjay Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters”, *Proc. ACM Conf. on Operating Systems Design & Implementation* (*OSDI*), 2004

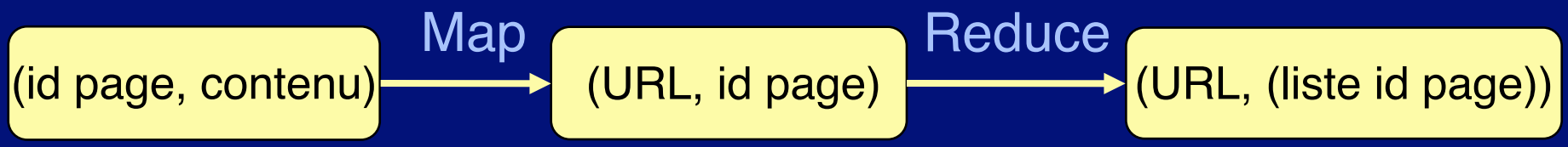
Un canevas pour le calcul parallèle : MapReduce



Jeffrey Dean, Sanjay Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters”, *Proc. ACM Conf. on Operating Systems Design & Implementation* (*OSDI*), 2004

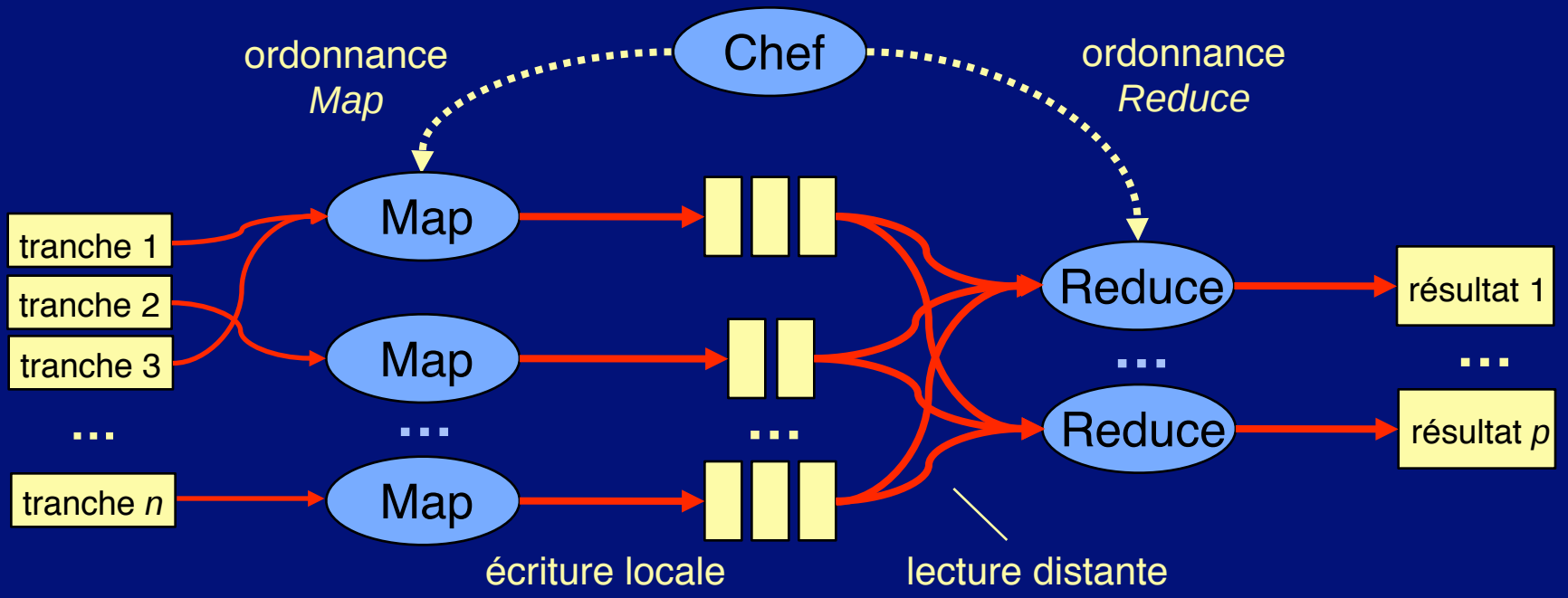
Un canevas pour le calcul parallèle : MapReduce

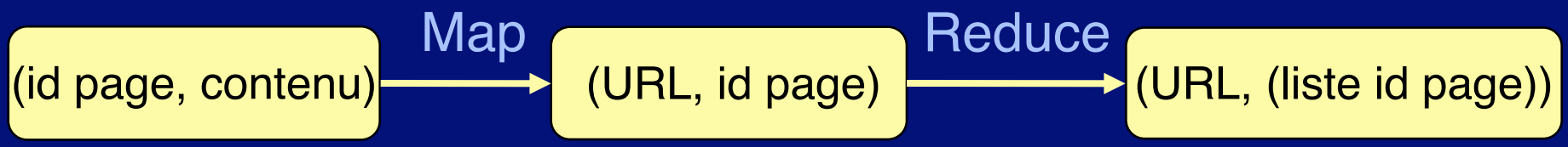




Jeffrey Dean, Sanjay Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters”, *Proc. ACM Conf. on Operating Systems Design & Implementation* (*OSDI*), 2004

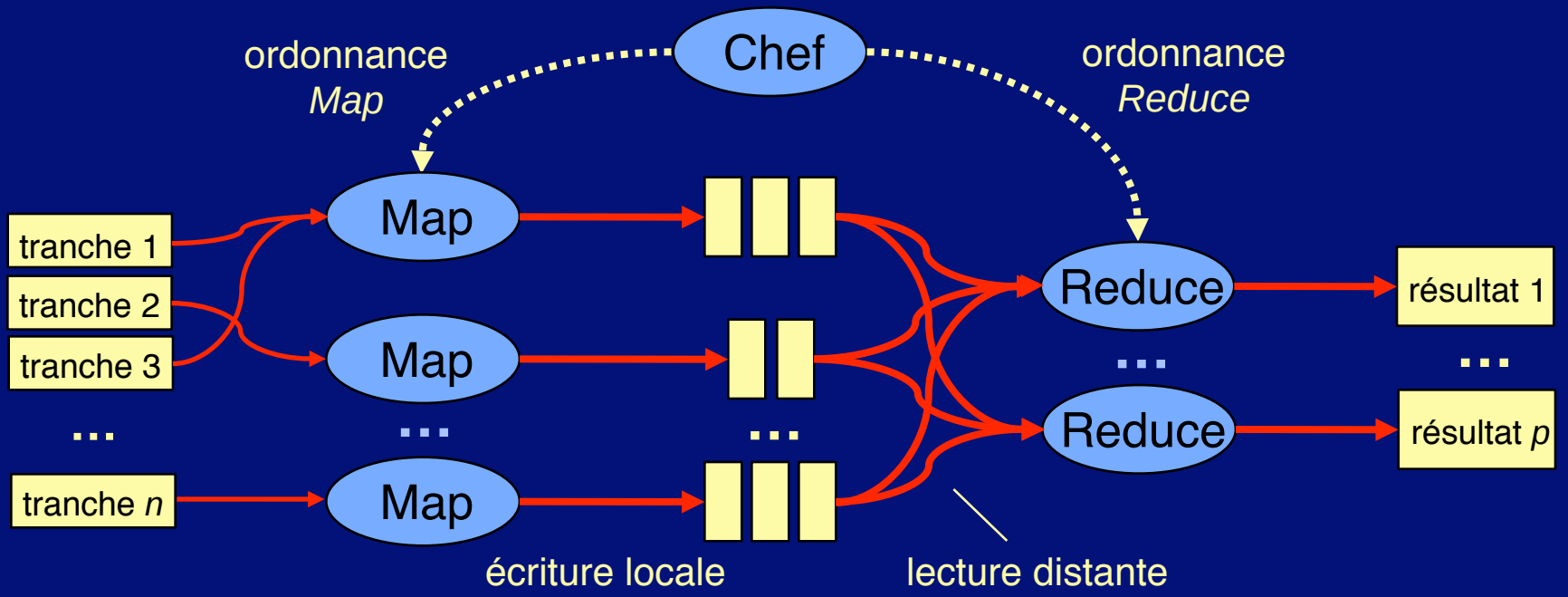
Un canevas pour le calcul parallèle : MapReduce

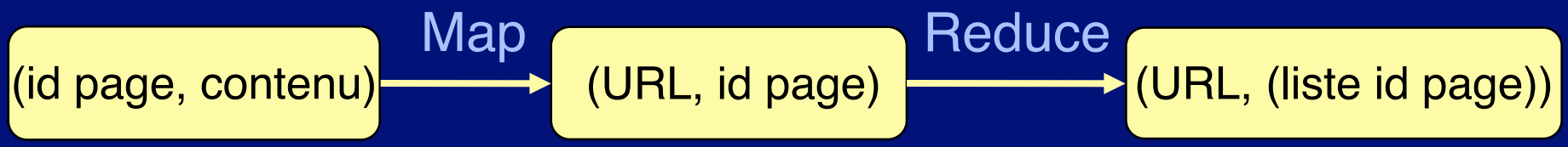




M. Stonebraker et al., “MapReduce and Parallel DBMSs: Friends or Foes?”, *Comm. ACM,* 53:1, January 2010, pp. 64-71

Un canevas pour le calcul parallèle : MapReduce





Jeffrey Dean, Sanjay Ghemawat, “MapReduce: A Flexible Data Processing Tool”, *Comm. ACM,* 53:1, January 2010, pp. 72-77

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

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

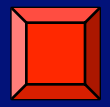
Faire face à l'imprévu

- 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



Outils :

Atomicité
Redondance
Adaptation

Transactions (1)

Transactions (1)

- Problème : maintenir la cohérence des données
  - En permettant les accès concurrents
  - Malgré l'occurrence de pannes

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)

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)

Transactions (2)

- Outils
  - Accès concurrents : théorie de la sérialisabilité
  - Atomicité (tout ou rien) : journalisation en mémoire stable
  - Durabilité : mémoire stable

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

Transactions (2)

- Outils
  - 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”
  - Mémoire transactionnelle
    - Limitations des verrous
    - Réalisation mixte matériel/logiciel
    - Nombreux problèmes ouverts …

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

Transactions (2)

- Outils
  - 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”
  - 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

Tolérance aux fautes

Tolérance aux fautes

- Fautes, erreurs, défaillances
- Fiabilité et disponibilité
- Classes de défaillances
  - de la panne franche à la panne byzantine

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

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
- Deux approches
  - Concertation forte (à petite échelle)
  - Concertation faible ou nulle (à grande échelle)

Abstractions pour un service disponible

Abstractions pour un service disponible

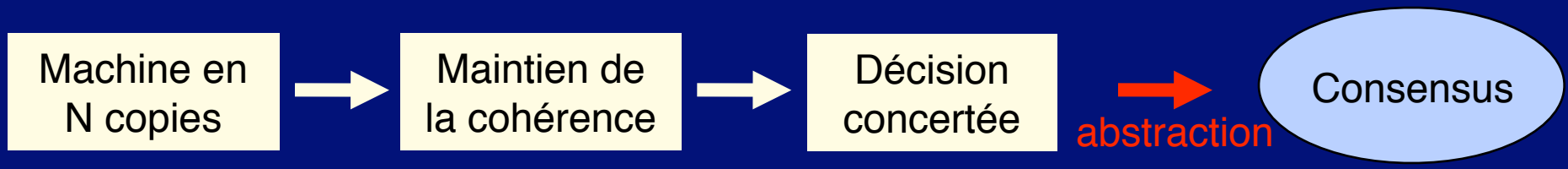
- La machine à états en *N* exemplaires
- Deux modes de réalisation
  - Redondance active
    - Requêtes dans le même ordre sur tous les serveurs
    - Diffusion atomique (totalement ordonnée)
  - Serveur primaire, serveurs de secours
    - Vue uniforme pour tous les serveurs : “vues synchrones”

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”

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”



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

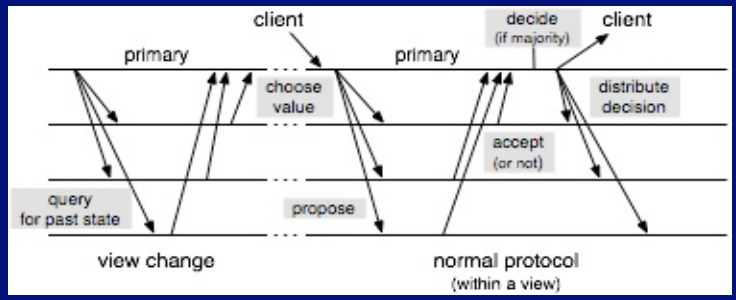
Brève histoire du consensus

Brève histoire du consensus

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

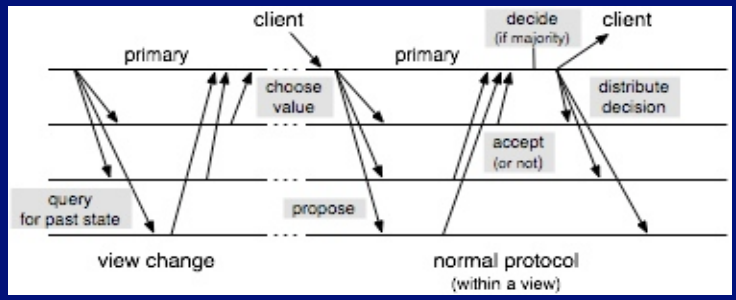
Brève histoire du consensus

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



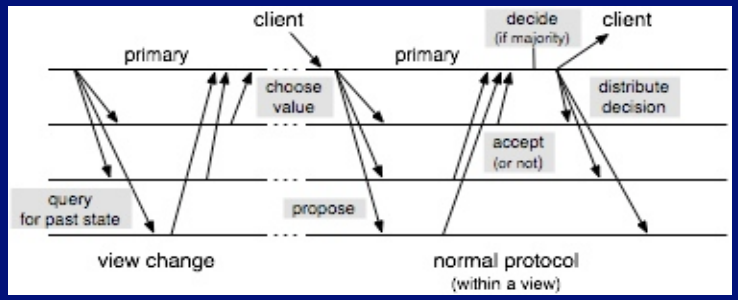
Brève histoire du consensus

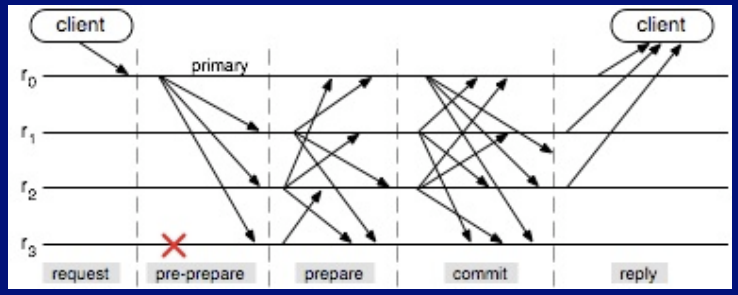
- Pannes franches en asynchrone
  - Pas d'algorithme déterministe !
  - Mais des palliatifs ingénieux
    - Détecteurs imparfaits
    - Paxos
- Pannes byzantines
  - Redondance forte
    - $3f+1$ pour tolérer f fautes
  - Complexité
    - Jusqu'en 1999 : exponentielle
    - Depuis : ~ quadratique, sujet "chaud"



Brève histoire du consensus

- Pannes franches en asynchrone
  - Pas d'algorithme déterministe !
  - Mais des palliatifs ingénieux
    - Détecteurs imparfaits
    - Paxos
- Pannes byzantines
  - Redondance forte
    - $3f+1$ pour tolérer f fautes
  - Complexité
    - Jusqu'en 1999 : exponentielle
    - Depuis : ~ quadratique, sujet "chaud"





Tolérance aux fautes à grande échelle

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

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

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

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 > MTTF !
  - Tolérance aux fautes = forte ponction sur la puissance de calcul

Observations finales

Observations finales

- Succès croissant dans l'application de 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

Observations finales

- Succès croissant dans l'application de 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

Observations finales

- Succès croissant dans l'application de 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

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

Observations finales

- Succès croissant dans l'application de 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
  - Théorie du parallélisme
  - Complexité algorithmique
  - Sémantique des données
  - Spécification et preuve
  - …
  - Sécurité
  - Du discret au continu
  - Maîtrise des très grands systèmes
  - Construction et composition
  - …

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

Pistes pour l’enseignement

Pistes pour l'enseignement

- Ne pas dissocier théorie et pratique
  - "Pratique" ne veut pas dire "non rigoureux"
  - "Théorique" ne veut pas dire "coupé du réel"
  - Apprendre à spécifier …
    - … y compris en environnement incertain

Pistes pour l'enseignement

- Ne pas dissocier théorie et pratique
  - "Pratique" ne veut pas dire "non rigoureux"
  - "Théorique" ne veut pas dire "coupé du réel"
  - Apprendre à spécifier …
    - … y compris en environnement incertain
- Ne pas cacher la complexité …
  - … mais montrer comment on la maîtrise (sur des cas concrets)
  - Avantage : motivation

Pistes pour l'enseignement

- Ne pas dissocier théorie et pratique
  - "Pratique" ne veut pas dire "non rigoureux"
  - "Théorique" ne veut pas dire "coupé du réel"
  - Apprendre à spécifier …
    - … y compris en environnement incertain
- Ne pas cacher la complexité …
  - … mais montrer comment on la maîtrise (sur des cas concrets)
  - Avantage : motivation
- Ne pas oublier l'histoire
  - Pour comprendre l'évolution des idées
  - Pour s'inspirer des beaux exemples

Pistes pour l'enseignement

- Ne pas dissocier théorie et pratique
  - "Pratique" ne veut pas dire "non rigoureux"
  - "Théorique" ne veut pas dire "coupé du réel"
  - Apprendre à spécifier …
    - … y compris en environnement incertain
- Ne pas cacher la complexité …
  - … mais montrer comment on la maîtrise (sur des cas concrets)
  - Avantage : motivation
- Ne pas oublier l'histoire
  - Pour comprendre l'évolution des idées
  - Pour s'inspirer des beaux exemples
- Introduire plus tôt l'informatique comme une science

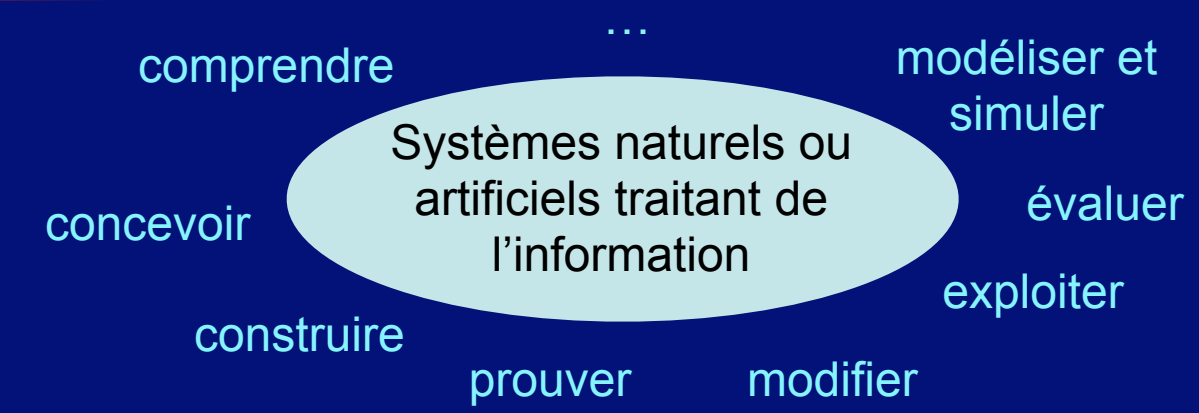
C’est quoi, au fond, l’informatique ?

C’est quoi, au fond, l’informatique ?

Systèmes naturels ou artificiels traitant de l’information

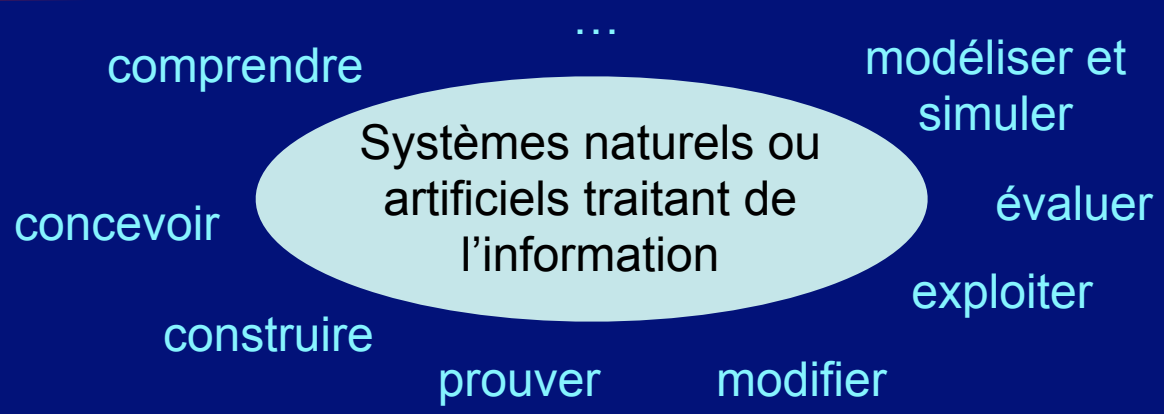
C’est quoi, au fond, l’informatique ?

Faire quoi ?



C’est quoi, au fond, l’informatique ?

Faire quoi ?



Avec quels moyens ?

architecture des systèmes matériels et logiciels

…

représentation, interaction

sûreté de fonctionnement

communication

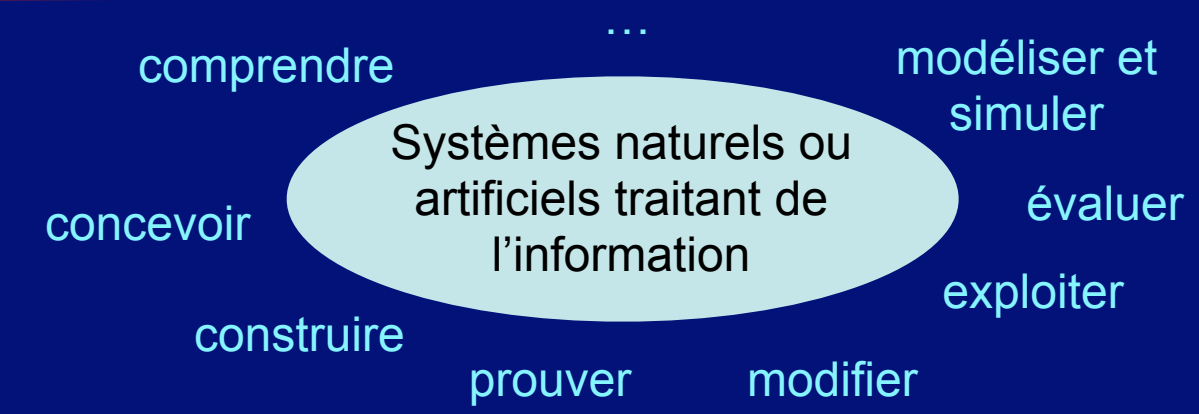
données, connaissance, raisonnement

modèles de calcul, construction, preuve

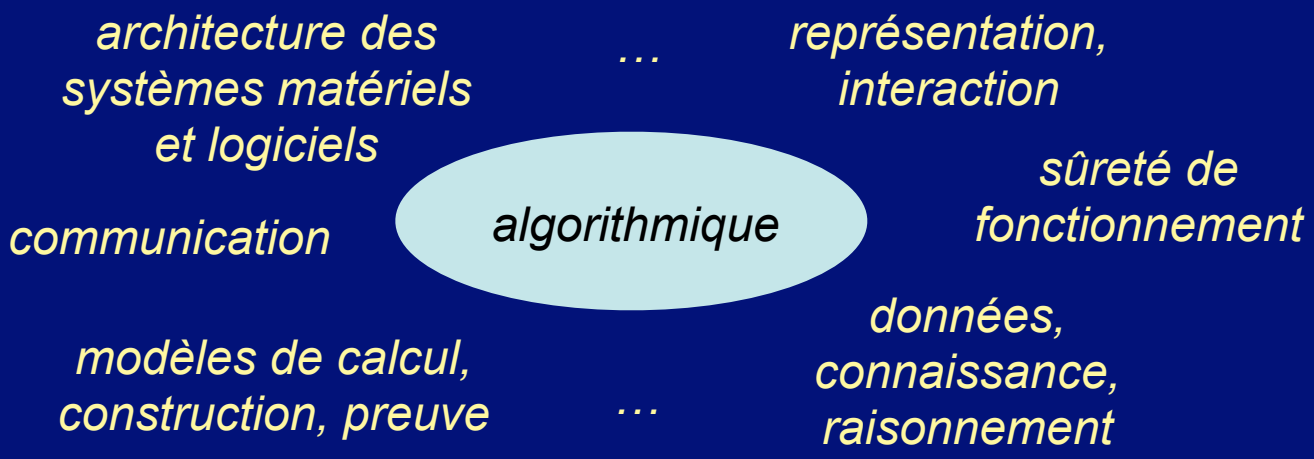
…

C’est quoi, au fond, l’informatique ?

Faire quoi ?

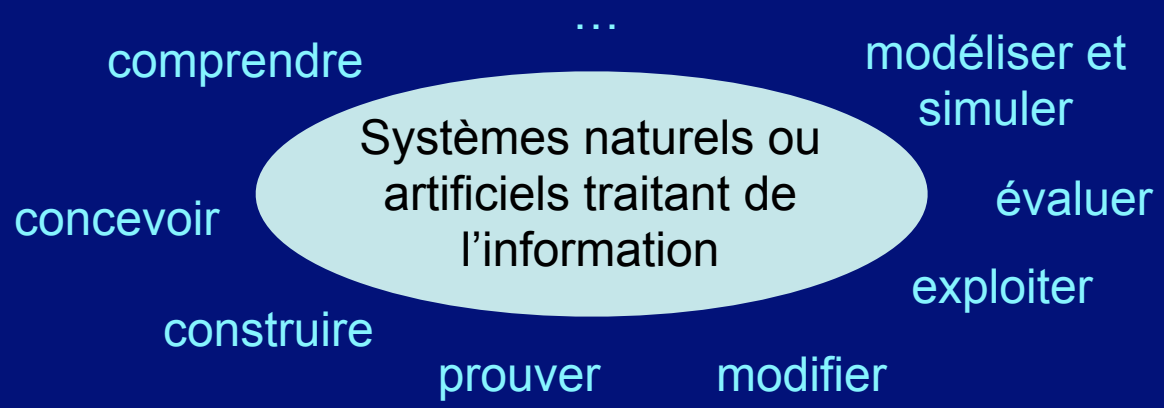


Avec quels moyens ?

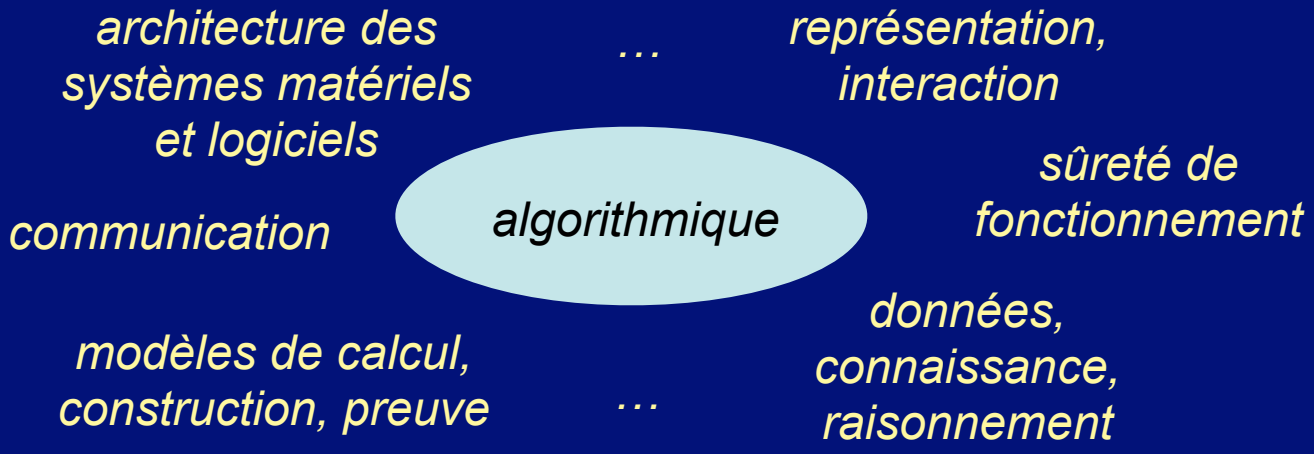


C’est quoi, au fond, l’informatique ?

Faire quoi ?



Avec quels moyens ?



Sur quelles bases ?

Merci de votre attention