LIG, 2/12/2010

Architecture des systèmes
Avancées et défis

Sacha Krakowiak

Université de Grenoble

Architecture des systèmes

- Architecture

  art de construire les édifices

  art de créer et d'organiser des formes en vue d'une fonction, en présence de contraintes

Architecture des systèmes

- Architecture

  art de construire les édifices

  art de créer et d'organiser des formes en vue d'une fonction, en présence de contraintes

Christopher Alexander, *Notes on the Synthesis of Form*, Harvard University Press, 1964

Architecture des systèmes

- Architecture
  - art de construire les édifices
  - art de créer et d'organiser des formes en vue d'une fonction, en présence de contraintes
- Architecture de systèmes informatiques
  - un (ou des) modèle(s) décrivant la structure et le comportement d'un système en termes d'éléments et de relations entre ces éléments
  - une (ou des) méthode(s) pour construire un système répondant à des spécifications

Christopher Alexander, *Notes on the Synthesis of Form*, Harvard University Press, 1964

Quelques outils de l'architecte

- Des (méta) principes

  Règles génériques pouvant se décliner sous diverses formes concrètes

  Abstraction (ex. : hiérarchie de machines abstraites)

  Séparation des préoccupations

  (ex. : séparation politique/mécanismes, interface/réalisation)

  Économie (ex. : optimiser le cas fréquent, contrôle «de bout en bout»)

  …

Quelques outils de l'architecte

- Des (méta) principes

  Règles génériques pouvant se décliner sous diverses formes concrètes

  Abstraction (ex. : hiérarchie de machines abstraites)

  Séparation des préoccupations

  (ex. : séparation politique/mécanismes, interface/réalisation)

  Économie (ex. : optimiser le cas fréquent, contrôle «de bout en bout»)

  …

- Des paradigmes

  Paradigme : démarche, mode d'organisation, structure applicable à une large classe de situations, et ayant valeur d'exemple, dans les deux sens du terme

  illustration d'une démarche

  modèle à suivre

Quelques paradigmes de l'architecture des systèmes

- Virtualisation
  - concrétiser un objet idéal
  - multiplier un objet réel

Quelques paradigmes de l'architecture des systèmes

- Virtualisation
  - concrétiser un objet idéal
  - multiplier un objet réel
- Composition et décomposition
  - séparer les préoccupations (conception individuelle / assemblage)
  - réutiliser les efforts de conception et de réalisation

Quelques paradigmes de l'architecture des systèmes

- Virtualisation
  - concrétiser un objet idéal
  - multiplier un objet réel
- Composition et décomposition
  - séparer les préoccupations (conception individuelle / assemblage)
  - réutiliser les efforts de conception et de réalisation
- Auto-adaptation et réflexivité
  - réagir au changement (prévu ou imprévu)
  - optimiser des critères de qualité

Quelques paradigmes de l'architecture des systèmes

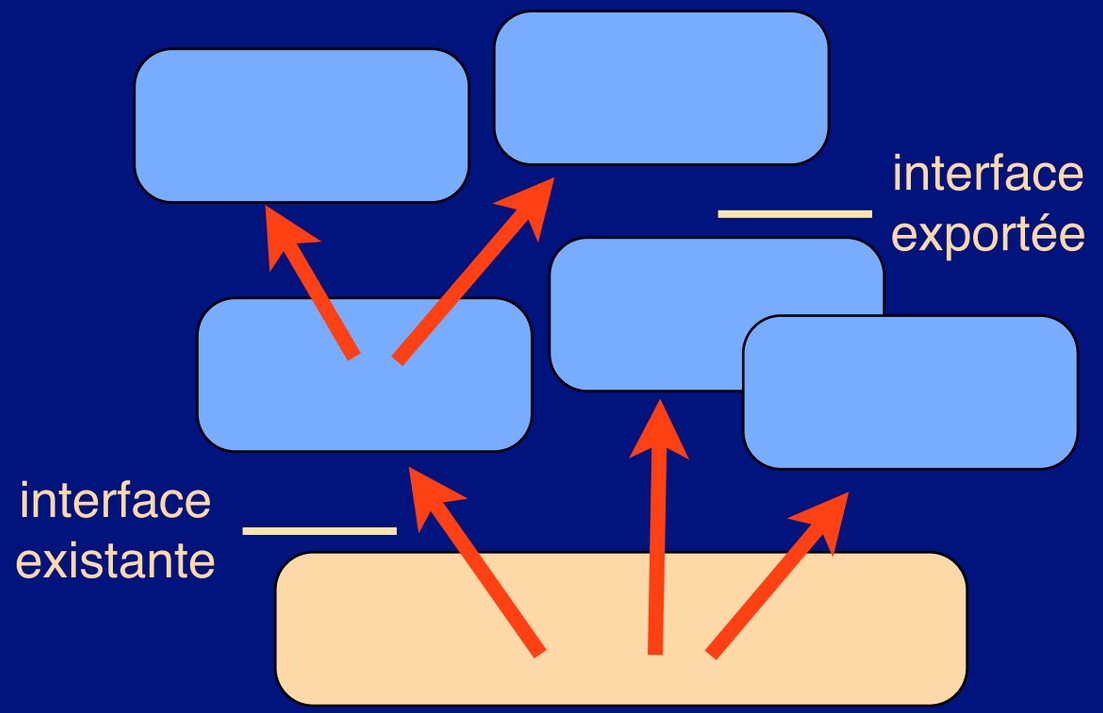
- Virtualisation
  - concrétiser un objet idéal
  - multiplier un objet réel
- Composition et décomposition
  - séparer les préoccupations (conception individuelle / assemblage)
  - réutiliser les efforts de conception et de réalisation
- Auto-adaptation et réflexivité
  - réagir au changement (prévu ou imprévu)
  - optimiser des critères de qualité

une préoccupation «transversale»

Formalisation, preuve

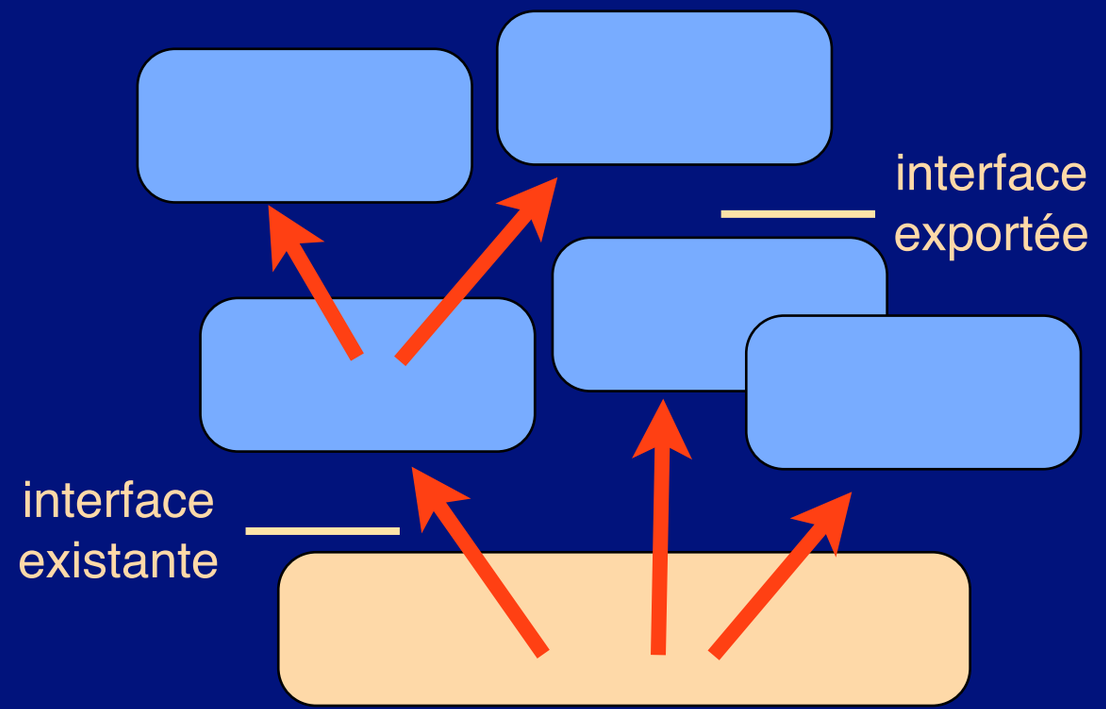
Les deux faces de la virtualisation

- Ascendante (abstraction)
  - Un outil de partage de ressources
  - Création de ressources «hautes»
  - Multiplexage de ressources «basses»

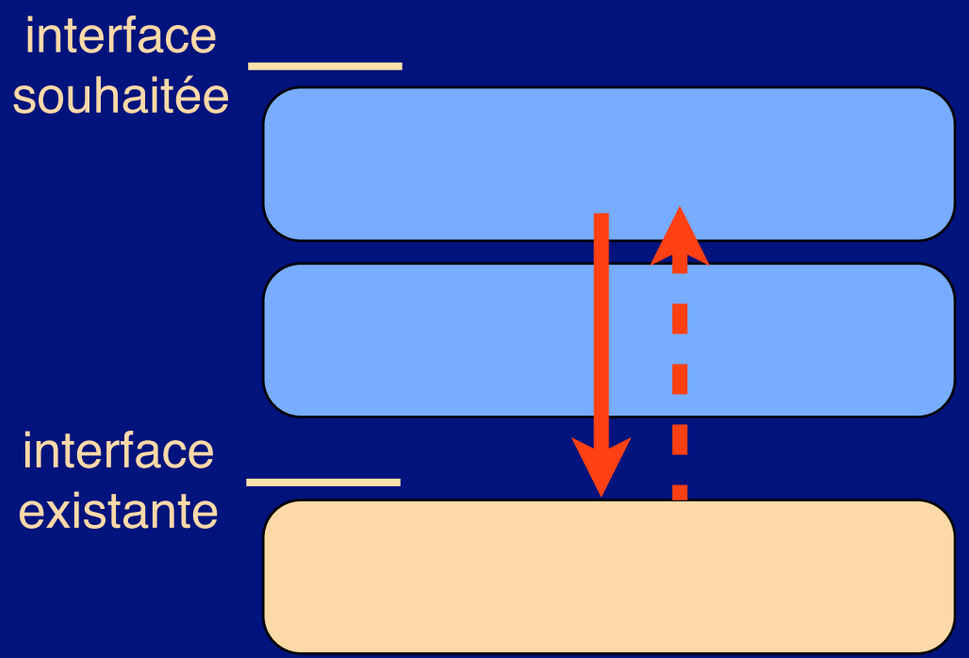


Les deux faces de la virtualisation

- Ascendante (abstraction)
  - Un outil de partage de ressources
  - Création de ressources «hautes»
  - Multiplexage de ressources «basses»



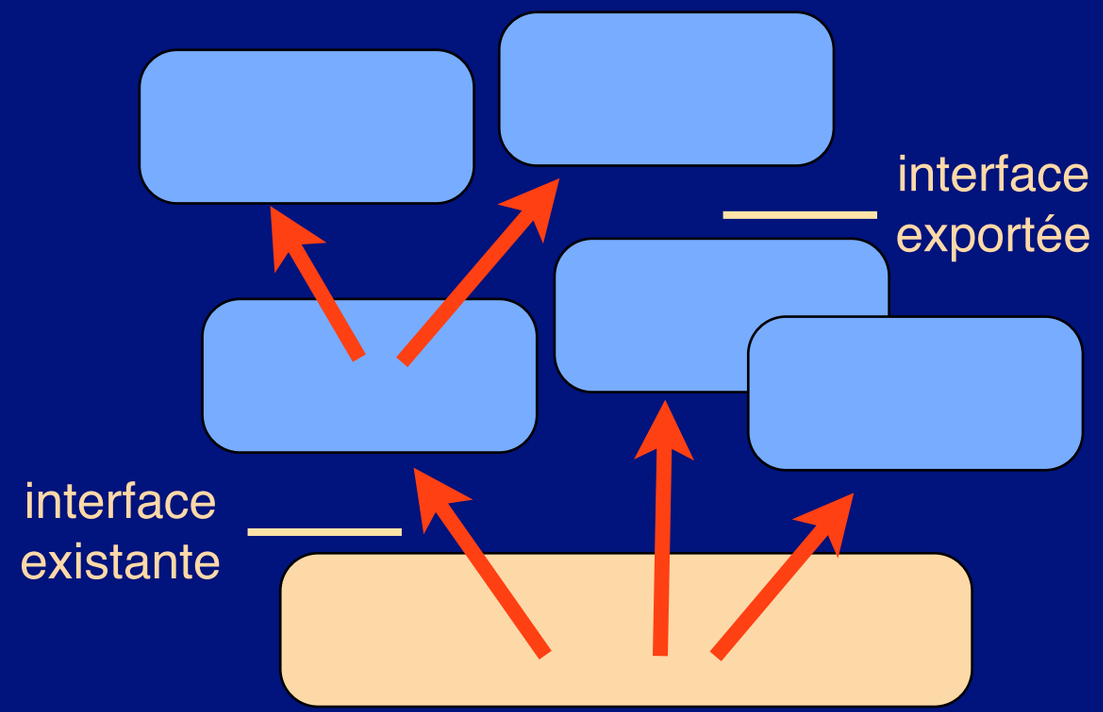
- Descendante (raffinement)
  - Un outil de conception
  - De la spécification à la réalisation
  - Hiérarchie de machines abstraites



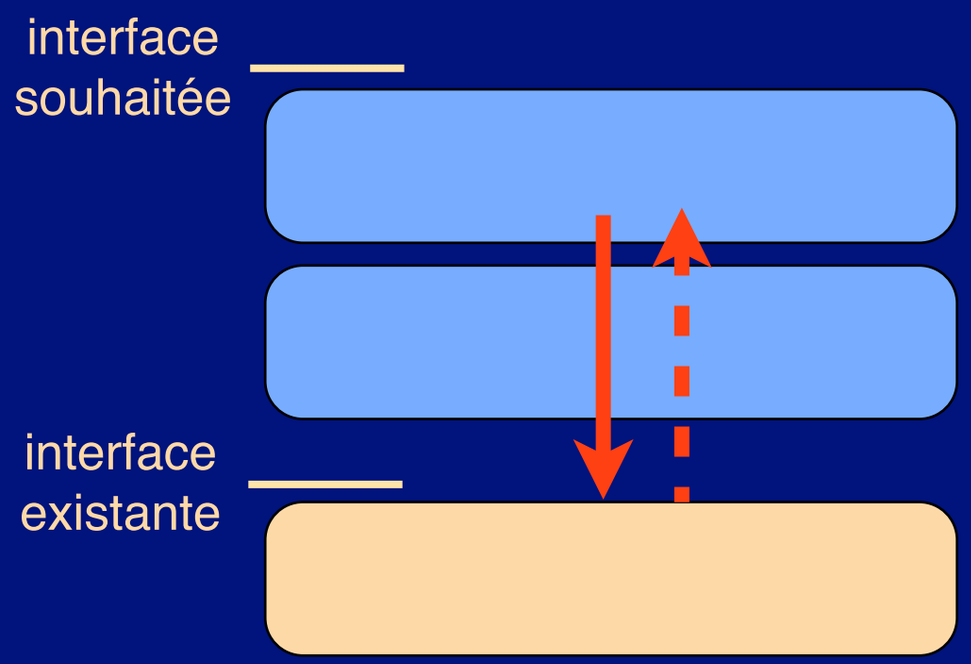
Les deux faces de la virtualisation

- Ascendante (abstraction)
  - Un outil de partage de ressources
  - Création de ressources «hautes»
  - Multiplexage de ressources «basses»

Interface visible, réalisation cachée
Transformation (ou non) d'interfaces
Préservation d'invariants



- Descendante (raffinement)
  - Un outil de conception
  - De la spécification à la réalisation
  - Hiérarchie de machines abstraites



Quoi virtualiser, et pourquoi ?

- Virtualiser des ressources
  - À l'intérieur d'un système d'exploitation
    - processeur => processus, mémoire physique => mémoire virtuelle
    - écran => fenêtre, périphérique => flot d'entrée-sortie, …

Quoi virtualiser, et pourquoi ?

- Virtualiser des ressources
  - À l'intérieur d'un système d'exploitation
    - processeur => processus, mémoire physique => mémoire virtuelle
    - écran => fenêtre, périphérique => flot d'entrée-sortie, …
- Virtualiser une machine
  - Pour multiplexer les ressources
  - Pour accueillir plusieurs systèmes d'exploitation
  - Pour assurer sécurité et tolérance aux fautes

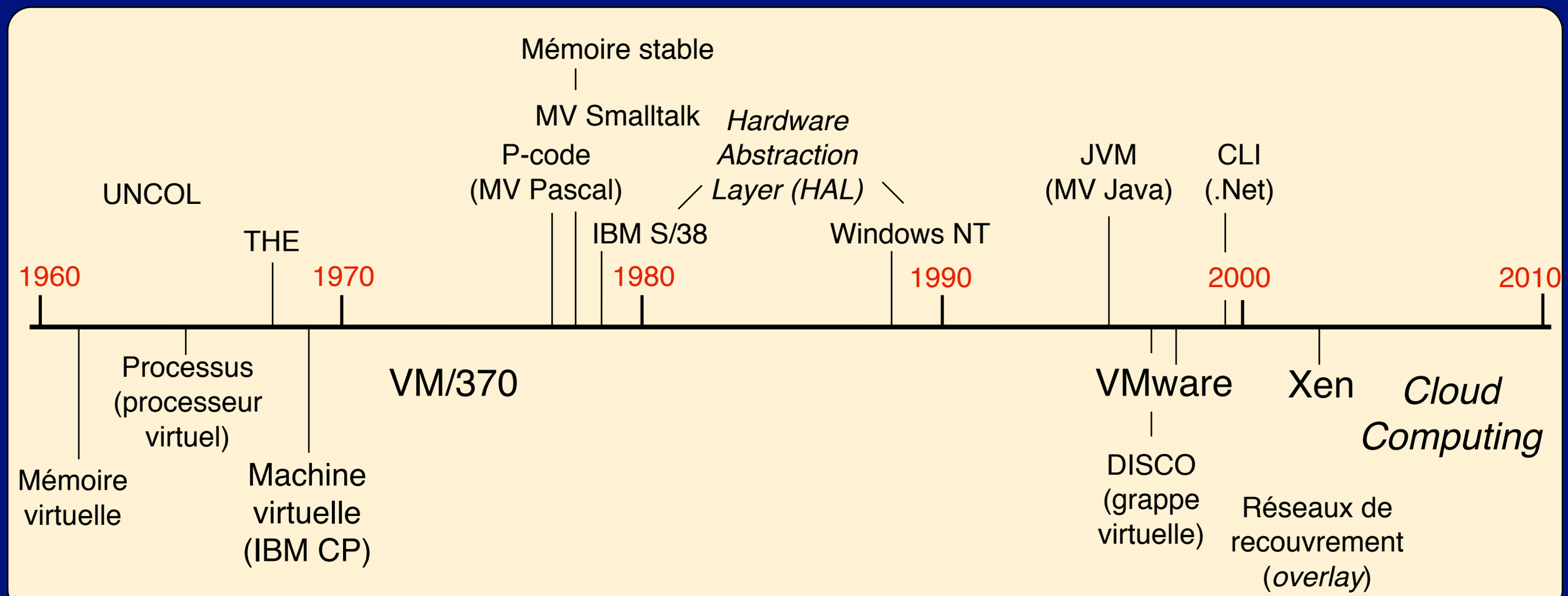
Quoi virtualiser, et pourquoi ?

- Virtualiser des ressources
  - À l'intérieur d'un système d'exploitation
    - processeur => processus, mémoire physique => mémoire virtuelle
    - écran => fenêtre, périphérique => flot d'entrée-sortie, …
- Virtualiser une machine
  - Pour multiplexer les ressources
  - Pour accueillir plusieurs systèmes d'exploitation
  - Pour assurer sécurité et tolérance aux fautes
- Virtualiser un réseau
  - Pour spécialiser une fonction (sécurité, performances, …)

Quoi virtualiser, et pourquoi ?

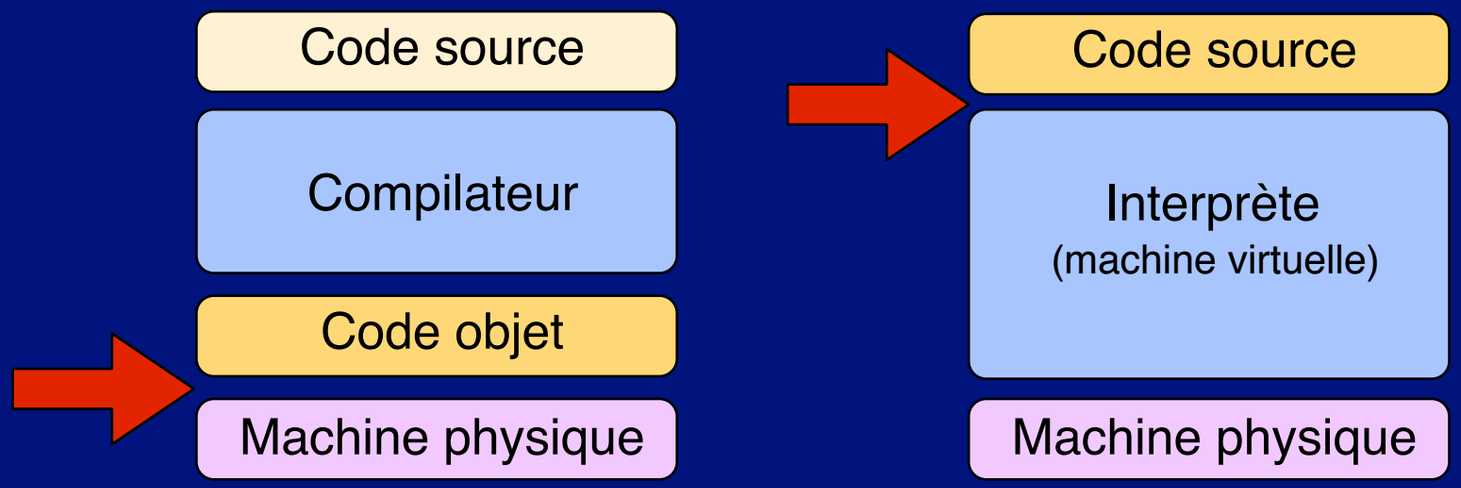
- Virtualiser des ressources
  - À l'intérieur d'un système d'exploitation
    - processeur => processus, mémoire physique => mémoire virtuelle
    - écran => fenêtre, périphérique => flot d'entrée-sortie, …
- Virtualiser une machine
  - Pour multiplexer les ressources
  - Pour accueillir plusieurs systèmes d'exploitation
  - Pour assurer sécurité et tolérance aux fautes
- Virtualiser un réseau
  - Pour spécialiser une fonction (sécurité, performances, …)
- Virtualiser un environnement d'exécution
  - Langage de programmation (machine virtuelle dédiée au langage)
  - Plate-forme (+ intergiciel (+ support d'applications))

Brève histoire de la virtualisation



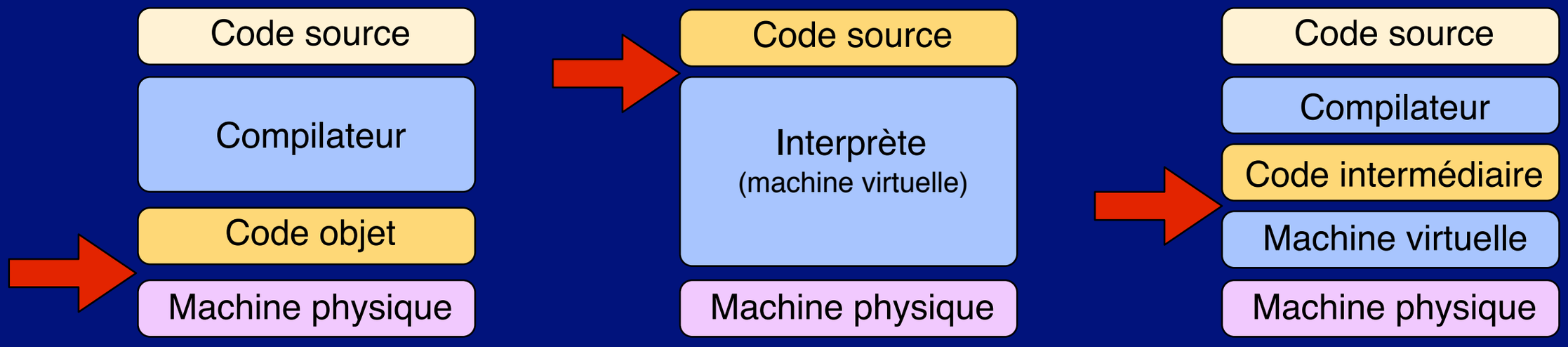
Virtualisation pour les langages

- Solution mixte entre compilation et interprétation



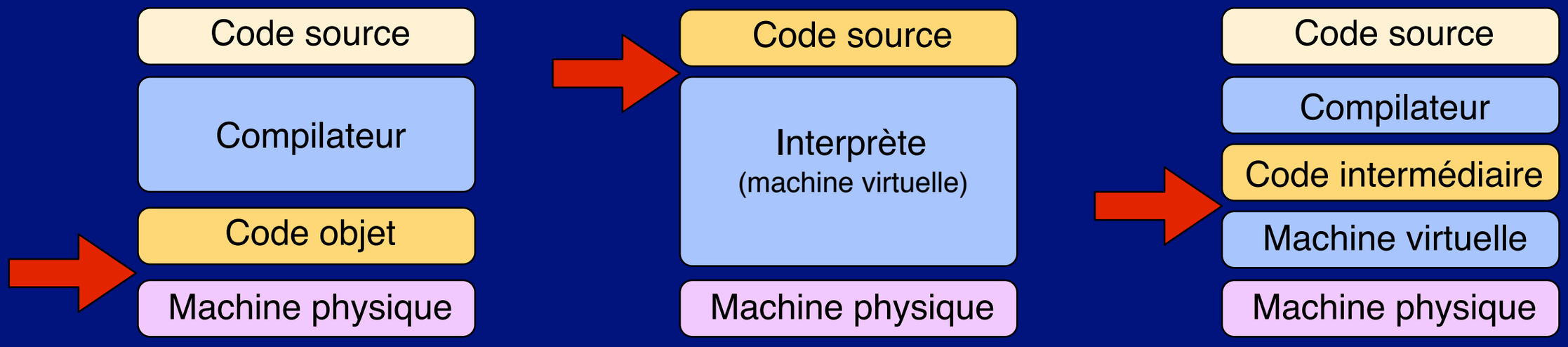
Virtualisation pour les langages

- Solution mixte entre compilation et interprétation



Virtualisation pour les langages

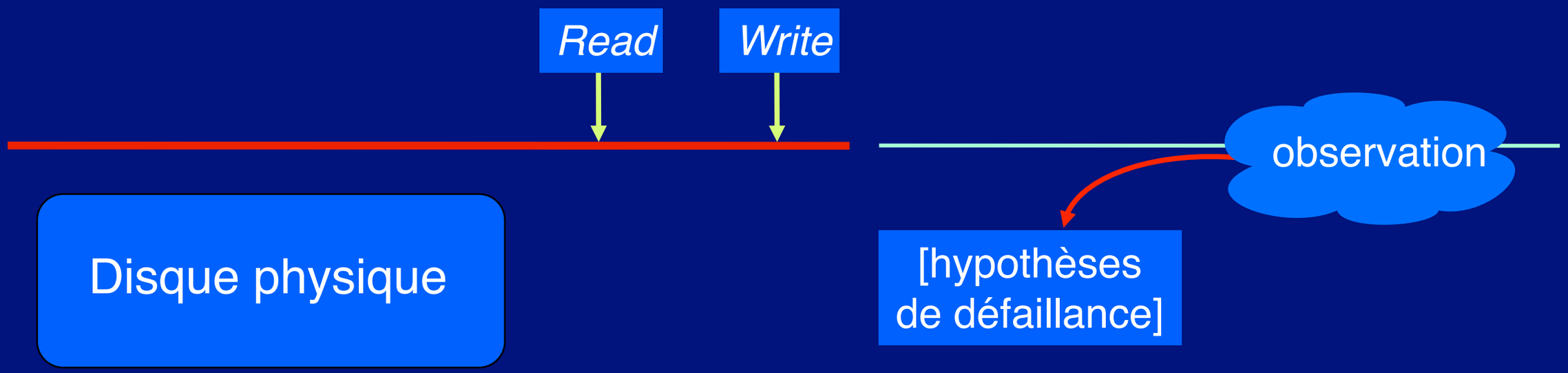
- Solution mixte entre compilation et interprétation



- Avantages
  - Portabilité (un seul compilateur)
  - Abstractions «sur mesure» adaptées à un (une classe de) langage(s)
    - Objets, composants, méta-données, …
  - Fonctions étendues pour la MV
    - Sécurité
    - Mise au point, mesures
    - Réflexivité

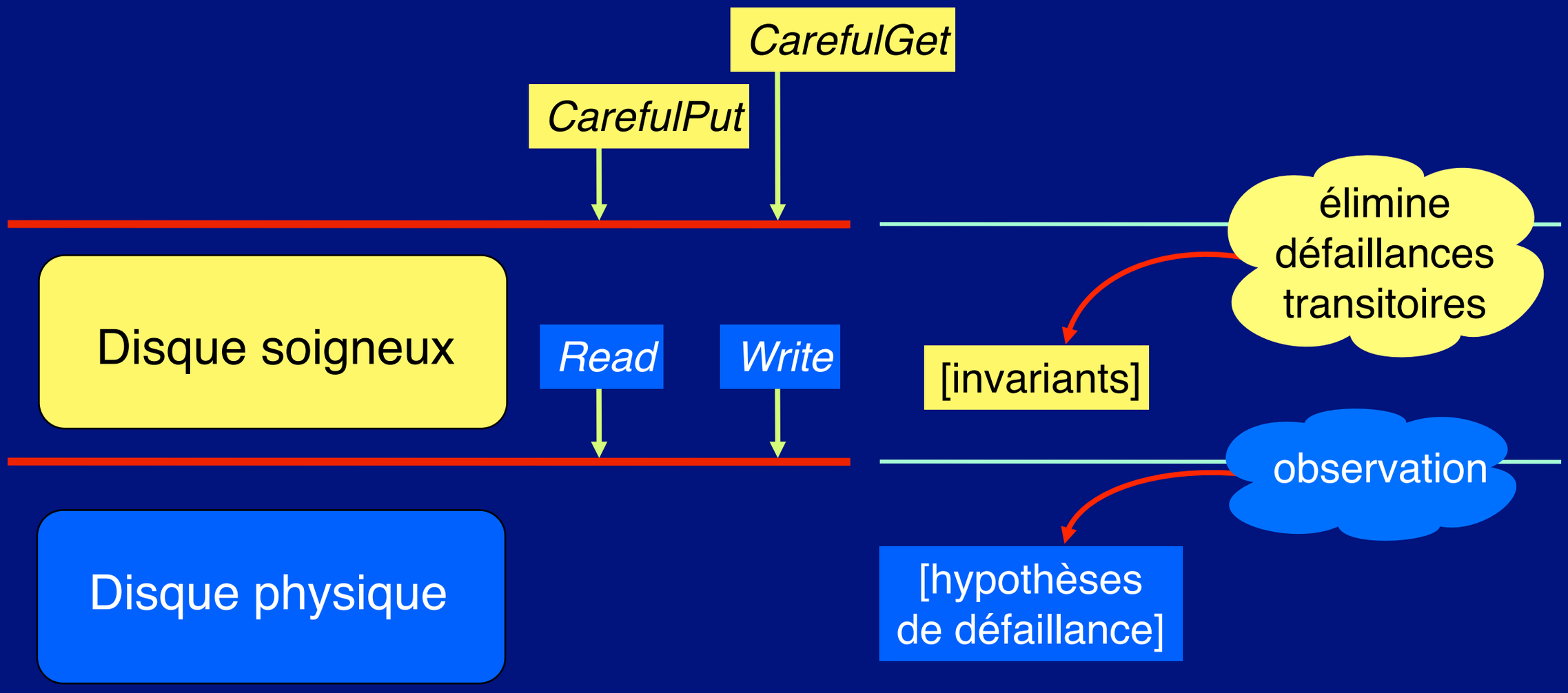
Exemples :
- P-Code (Pascal)
- JVM (Java)
- CLI

Virtualisation d’une ressource : la mémoire stable



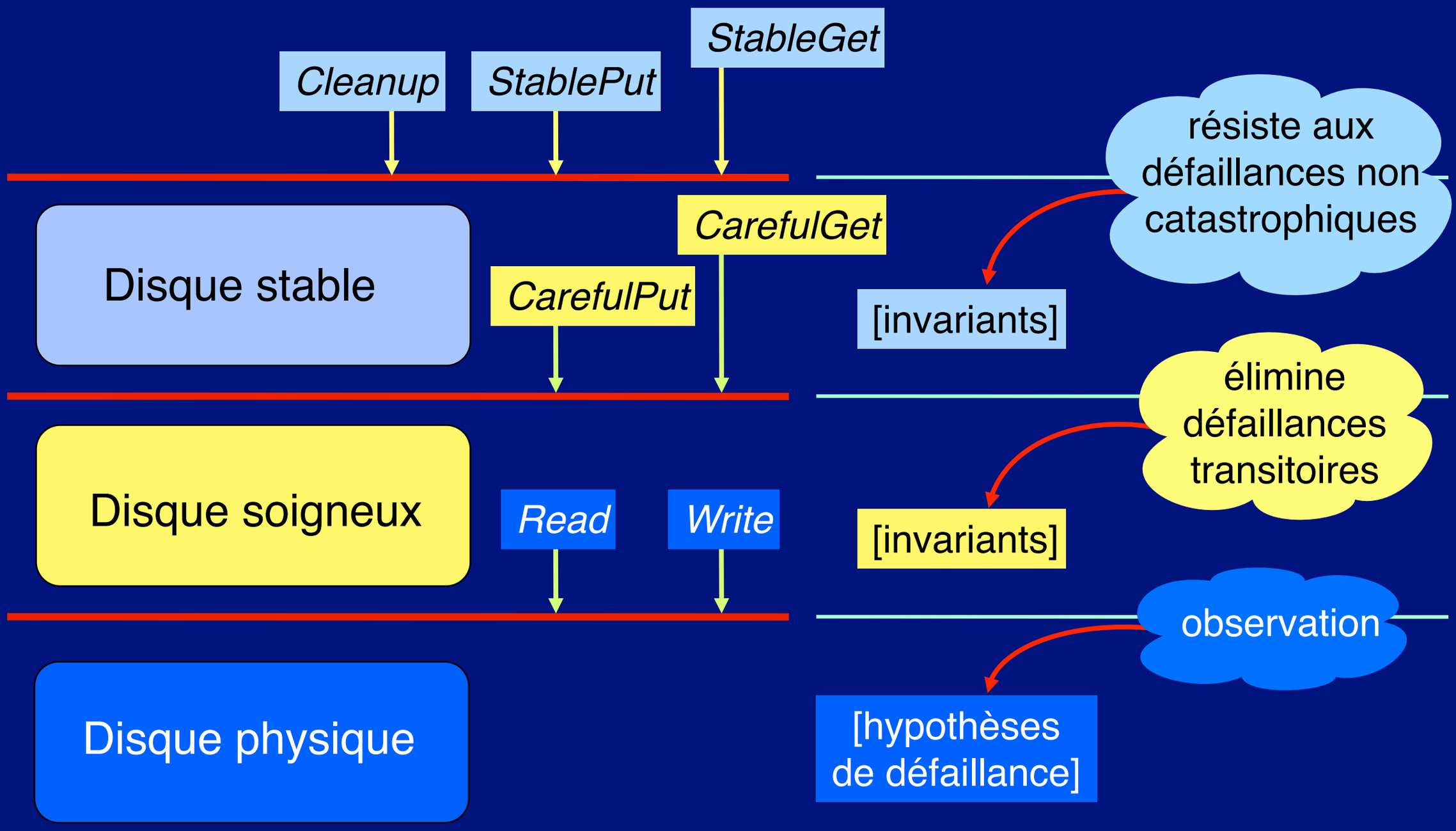
B. W. Lampson and H. E. Sturgis. Crash Recovery in a Distributed Data Storage System, unpublished technical report, Xerox PARC, June 1979, 25 pp.

Virtualisation d'une ressource : la mémoire stable



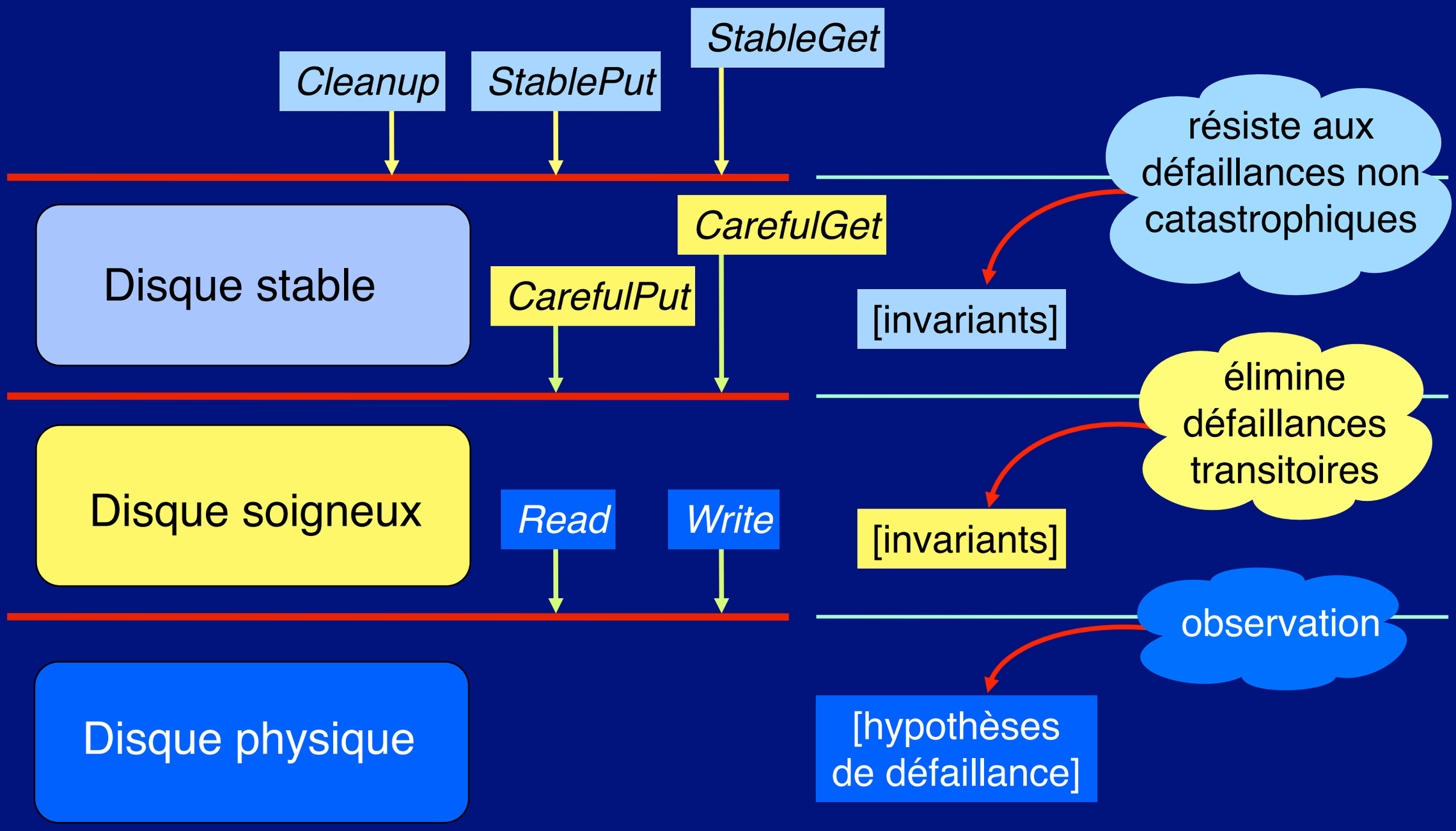
B. W. Lampson and H. E. Sturgis. Crash Recovery in a Distributed Data Storage System, unpublished technical report, Xerox PARC, June 1979, 25 pp.

Virtualisation d'une ressource : la mémoire stable



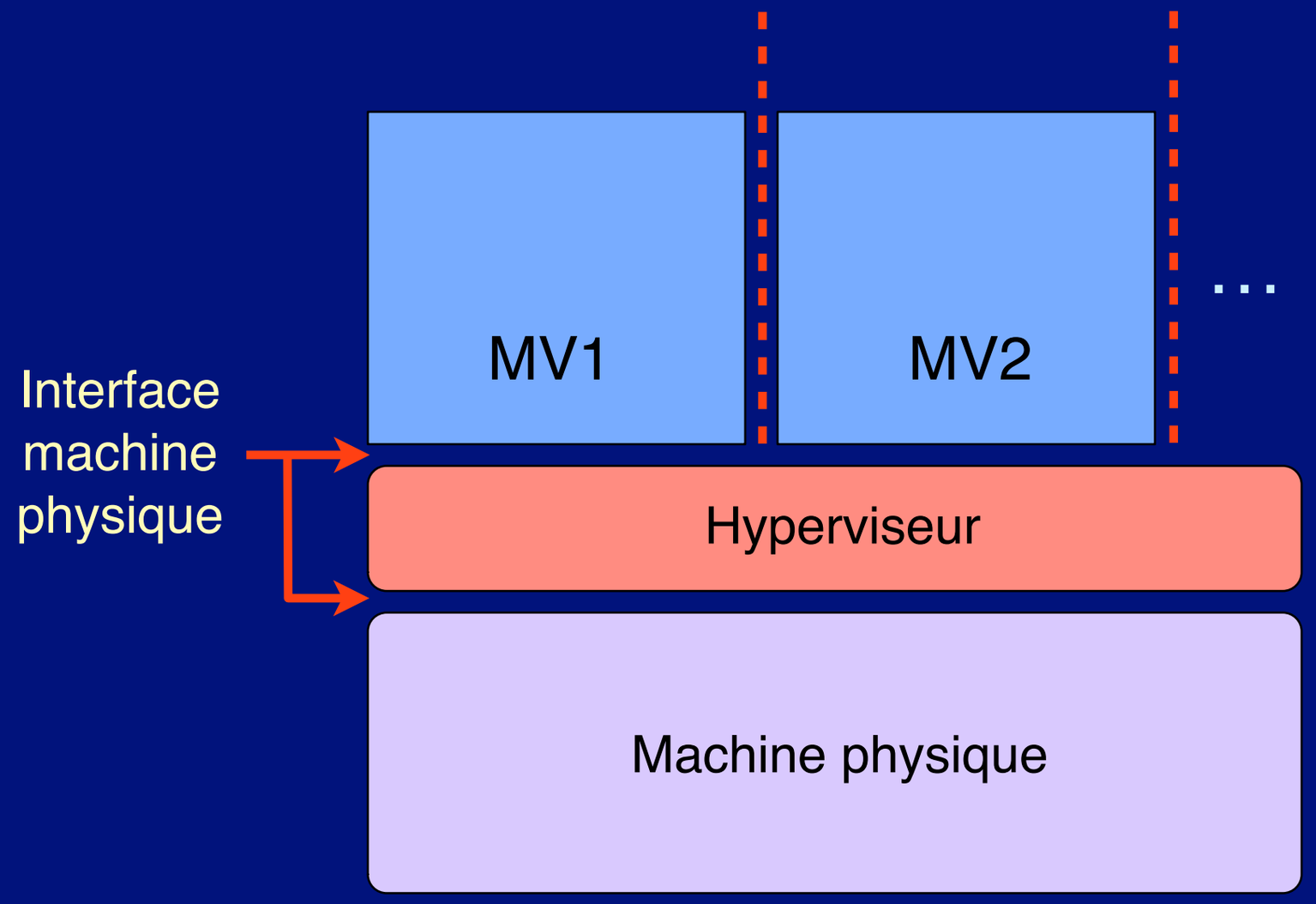
B. W. Lampson and H. E. Sturgis. Crash Recovery in a Distributed Data Storage System, unpublished technical report, Xerox PARC, June 1979, 25 pp.

Virtualisation d'une ressource : la mémoire stable

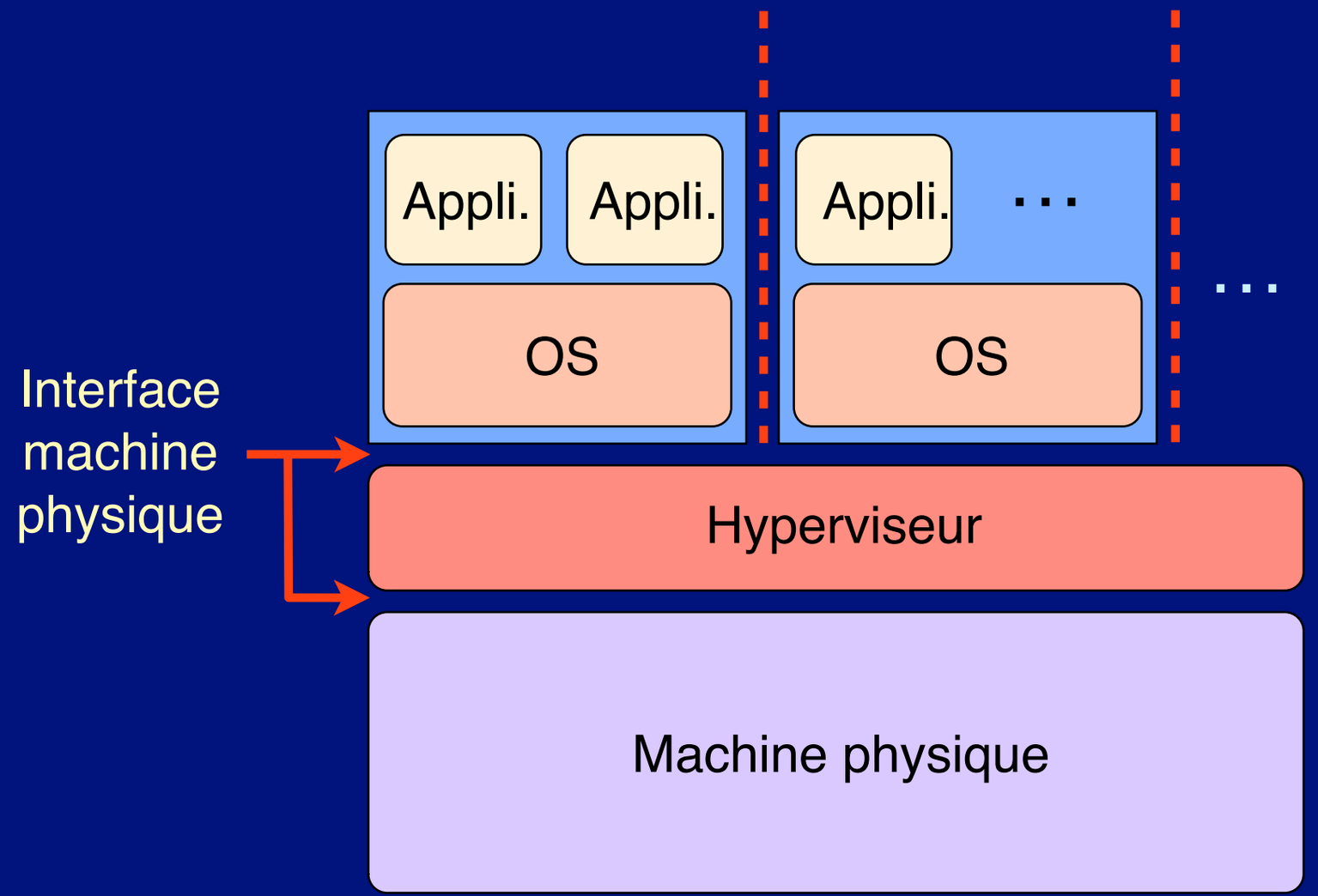


B. W. Lampson, "Atomic Transactions", in *Distributed Systems—Architecture and Implementation*, ed. Lampson, Paul, and Siegert, LNCS 105, Springer, 1981, pp. 246-265.

Machines virtuelles «classiques»

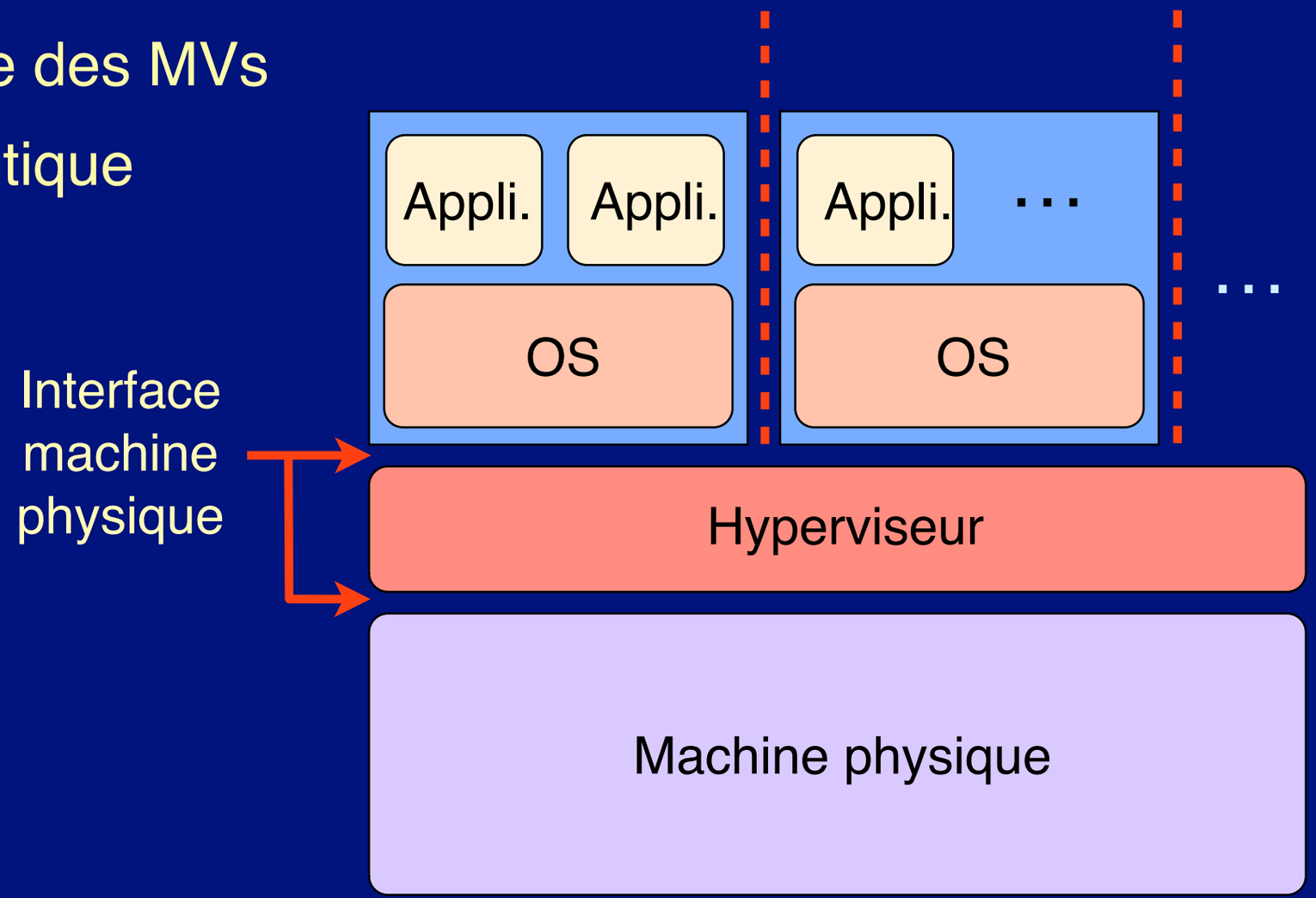


Machines virtuelles «classiques»



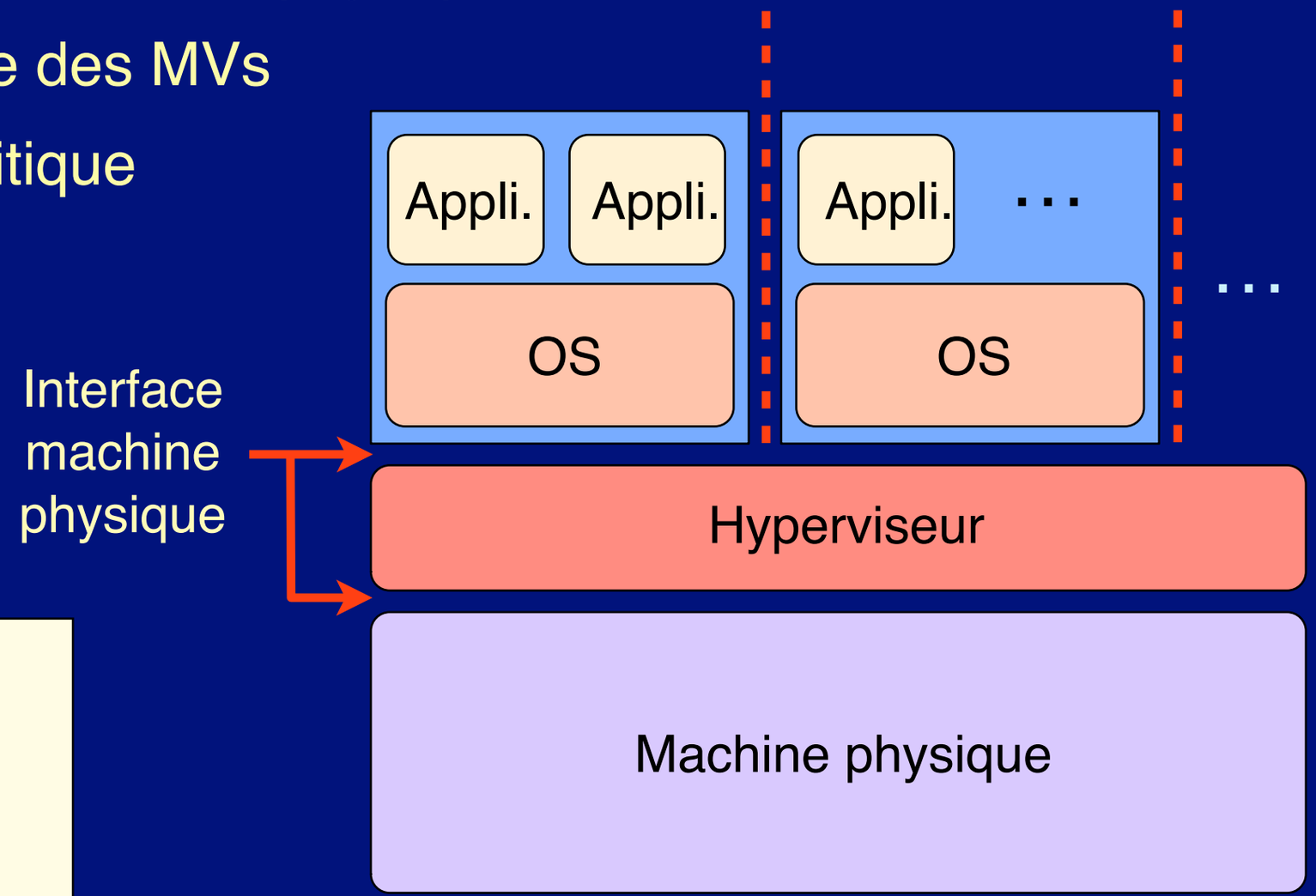
Machines virtuelles «classiques»

- L'hyperviseur, un super-OS
  - Présente une interface uniforme aux machines virtuelles (MVs)
  - Gère (et protège) les ressources physiques
  - Encapsule l'état interne des MVs
  - C'est un composant critique
    - Isolation
    - Défaillances
    - Sécurité



Machines virtuelles «classiques»

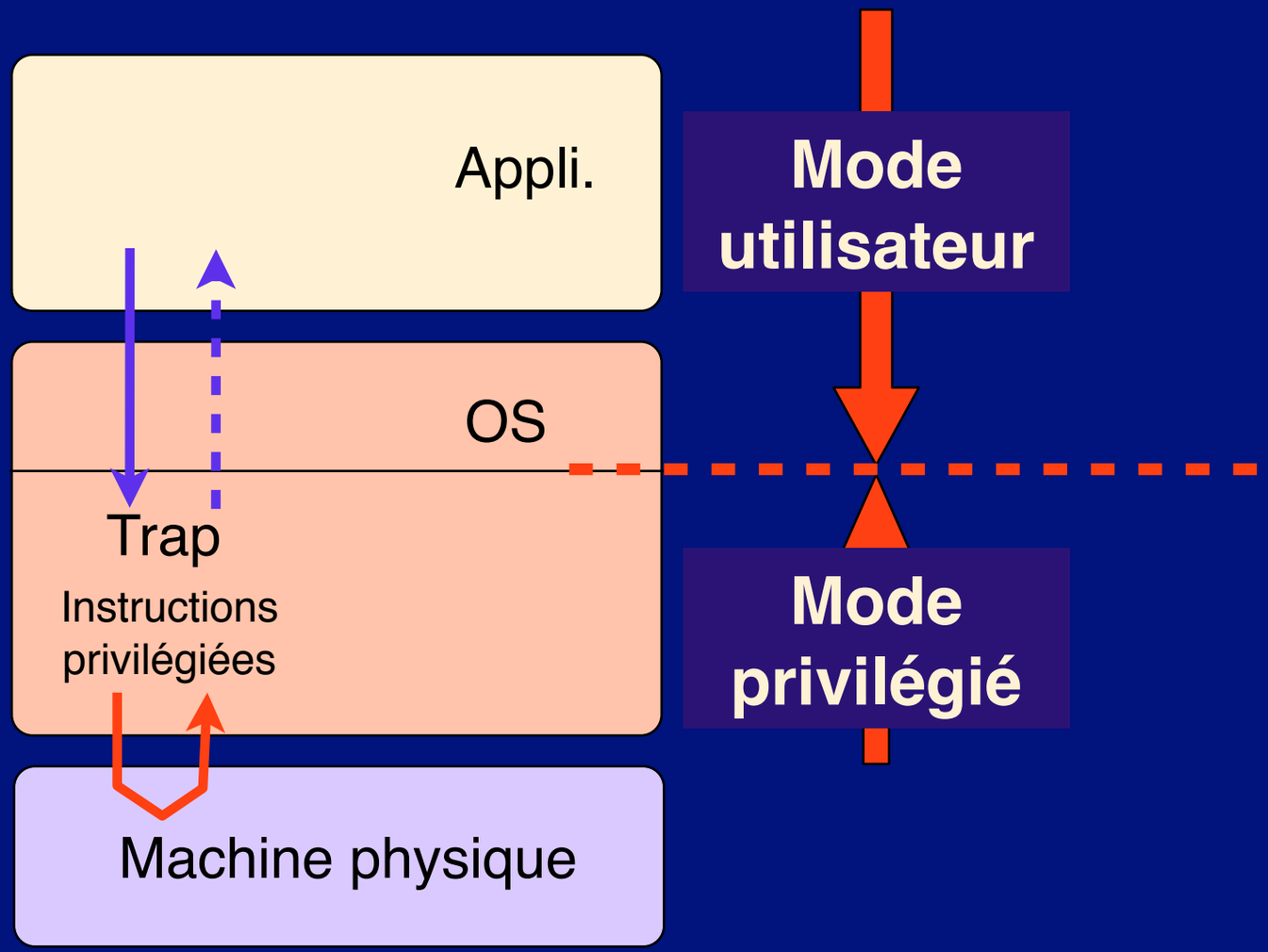
- L'hyperviseur, un super-OS
  - Présente une interface uniforme aux machines virtuelles (MVs)
  - Gère (et protège) les ressources physiques
  - Encapsule l'état interne des MVs
  - C'est un composant critique
    - Isolation
    - Défaillances
    - Sécurité



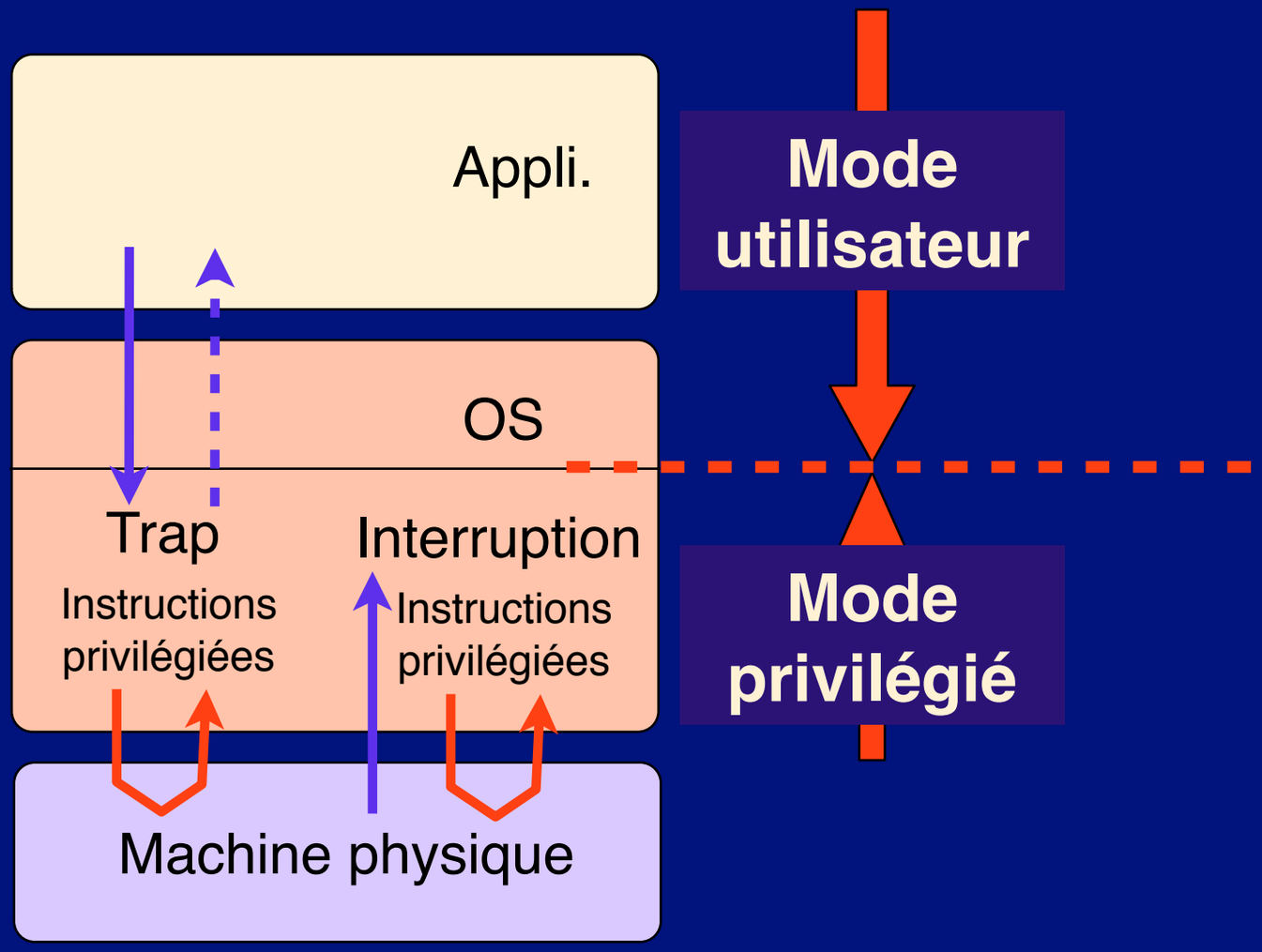
James E. Smith, Ravi Nair, *Virtual Machines*, Morgan Kaufmann, 2005

Virtualization Technologies, *IEEE Computer*, May 2005

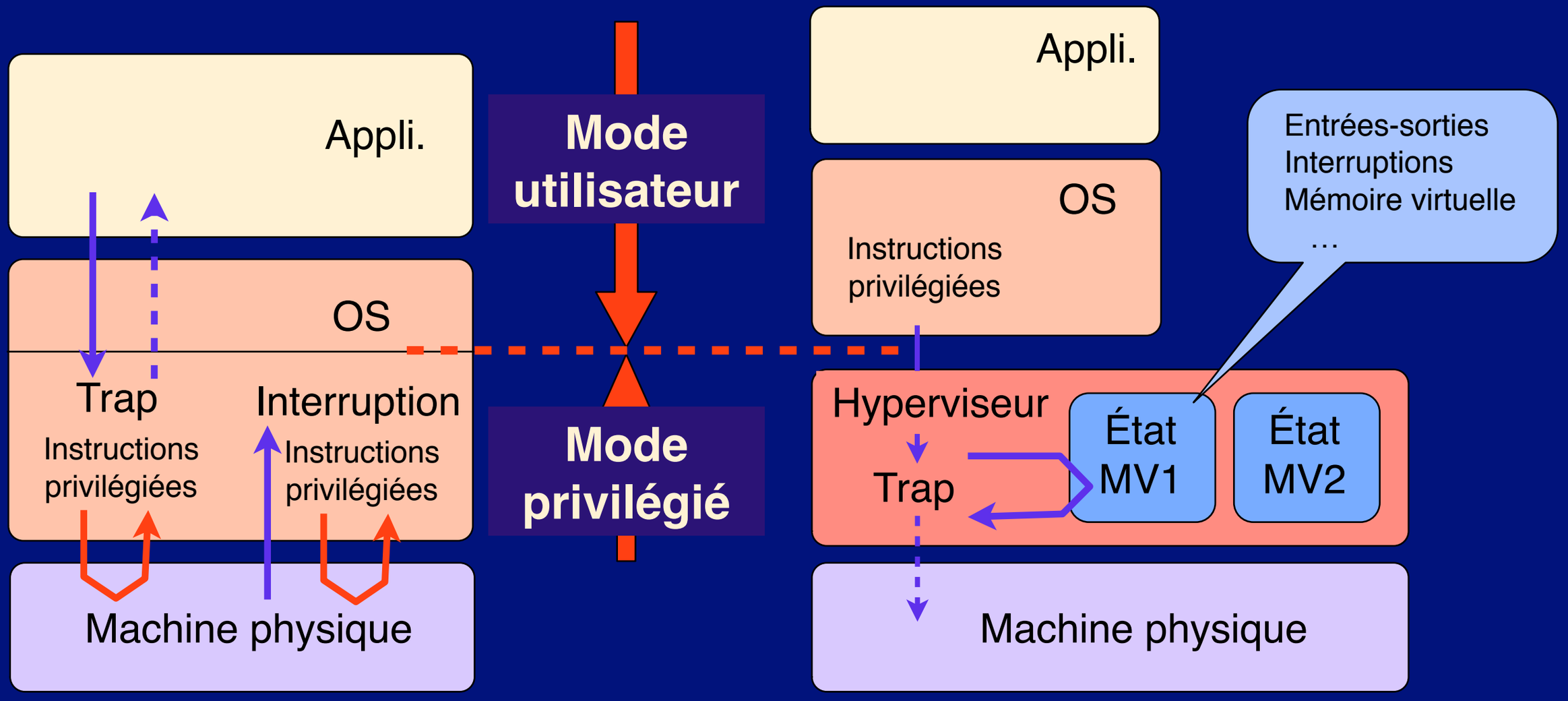
Machines virtuelles : comment çà marche



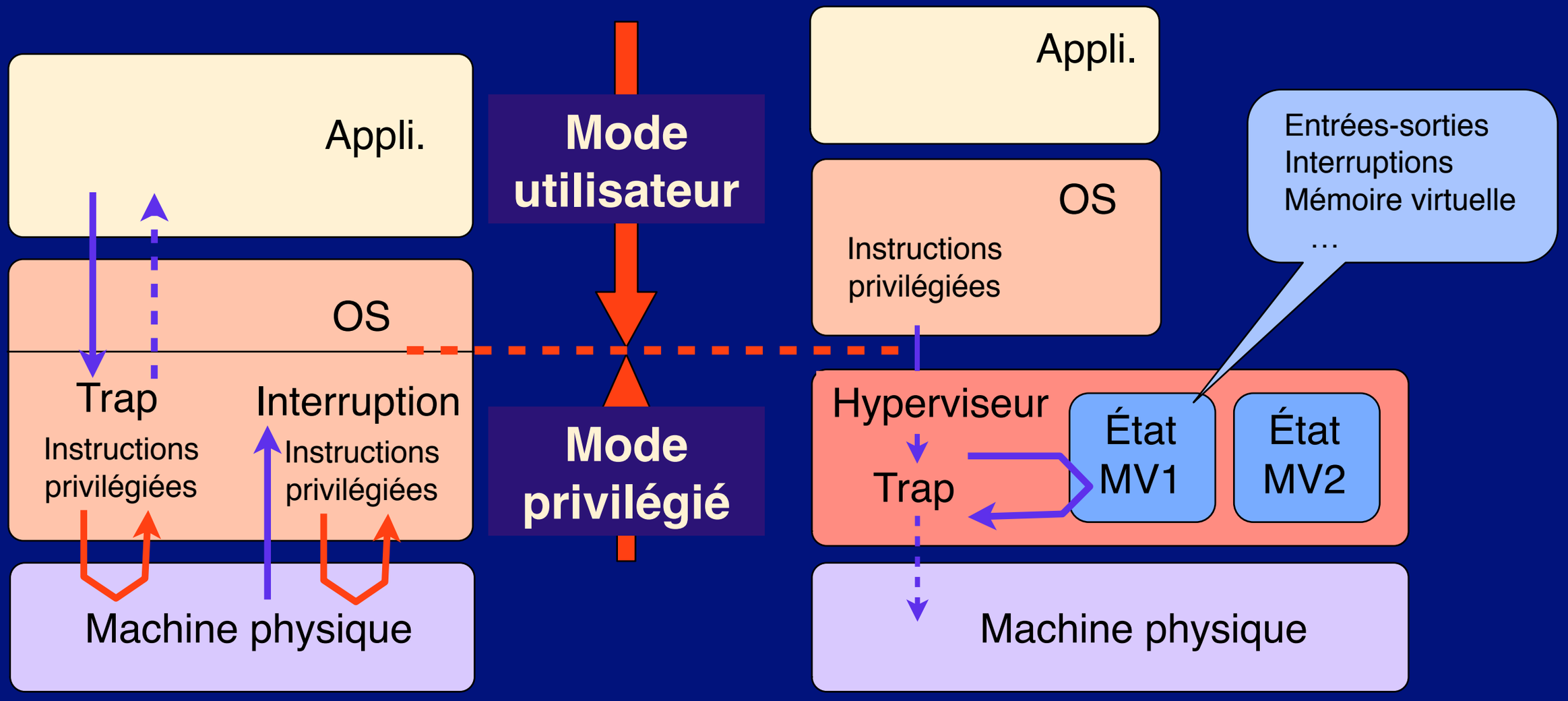
Machines virtuelles : comment çà marche



Machines virtuelles : comment çà marche

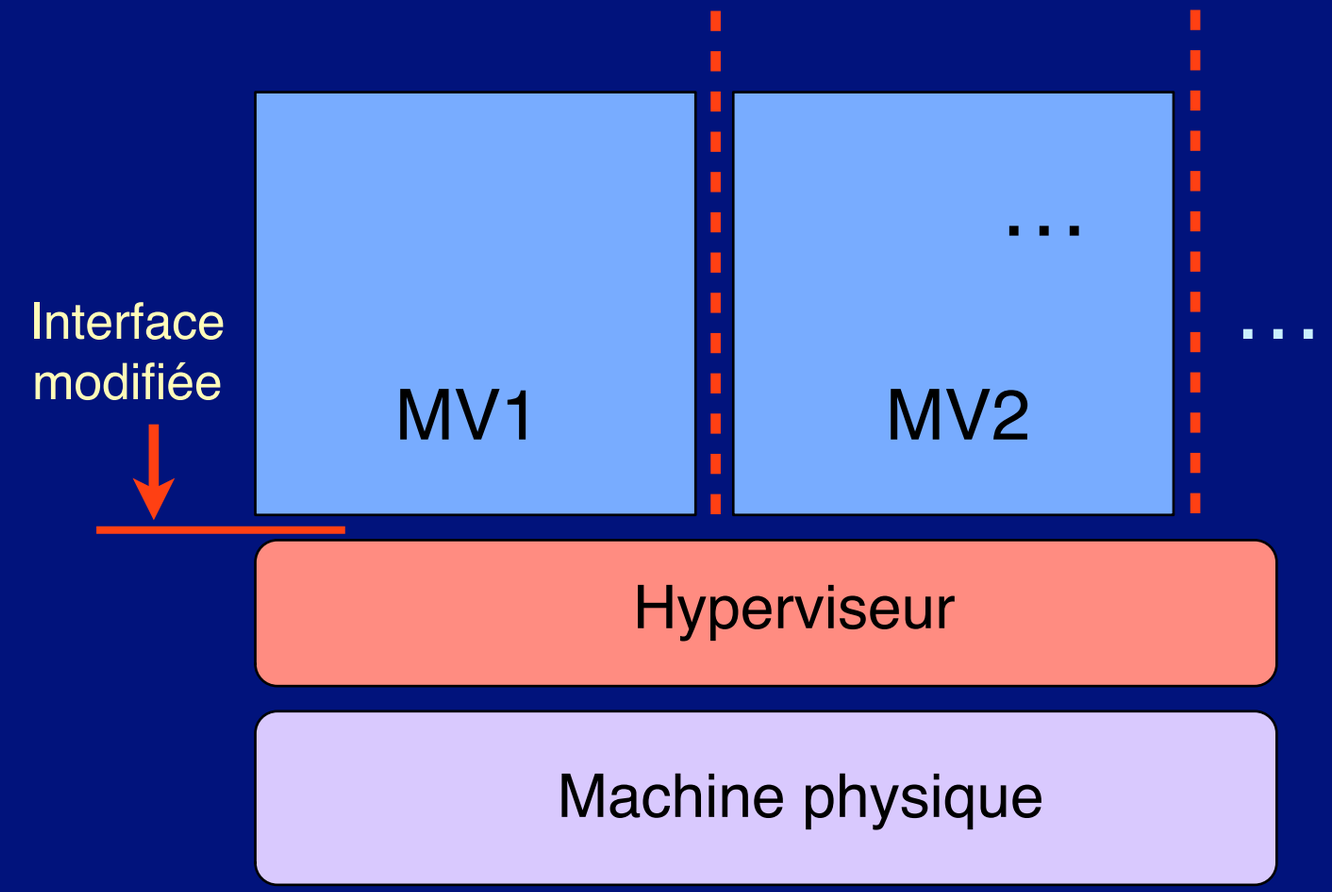


Machines virtuelles : comment çà marche

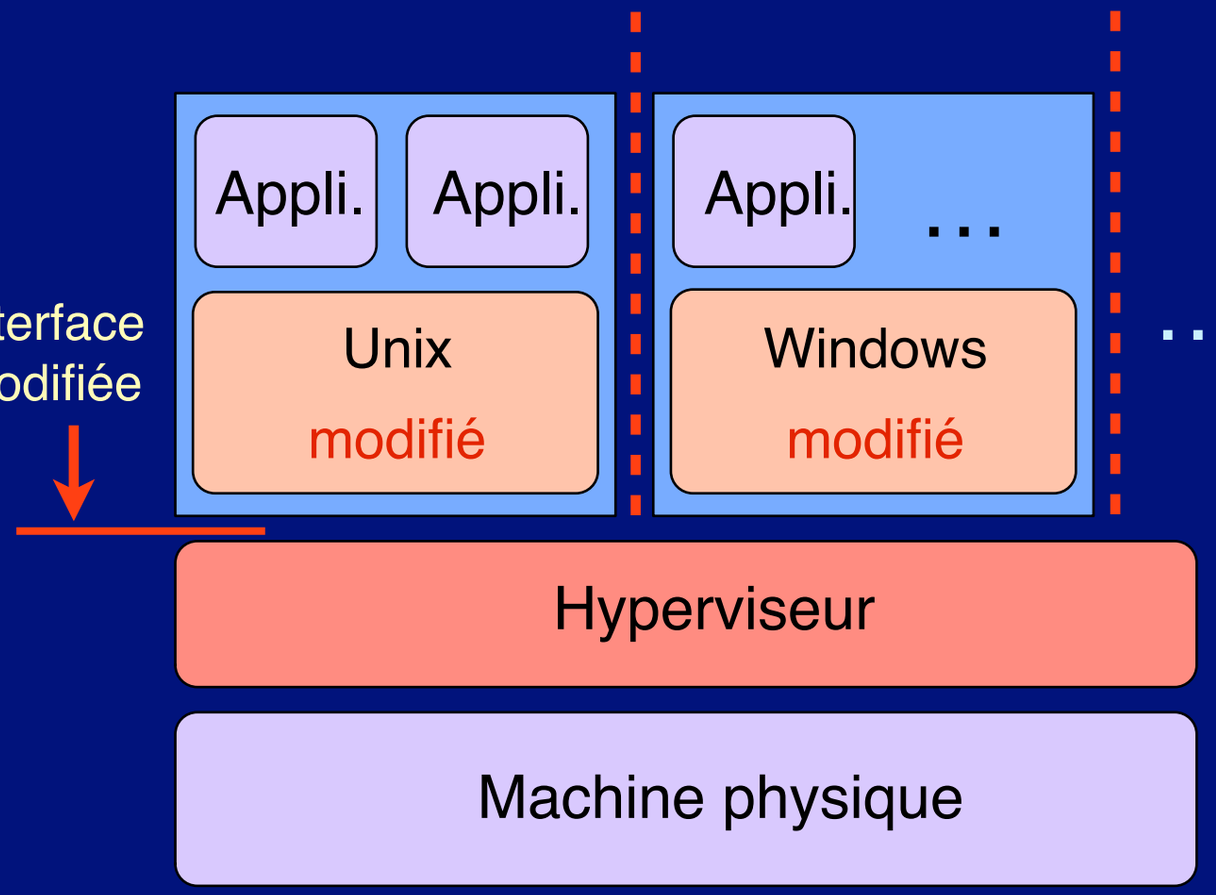


- Un problème …
  - Sur les machines actuelles (IA-32, etc.), l'effet de certaines instructions dépend du mode courant (privilégié ou utilisateur)
  - De telles instructions ne sont pas virtualisables

Machines virtuelles

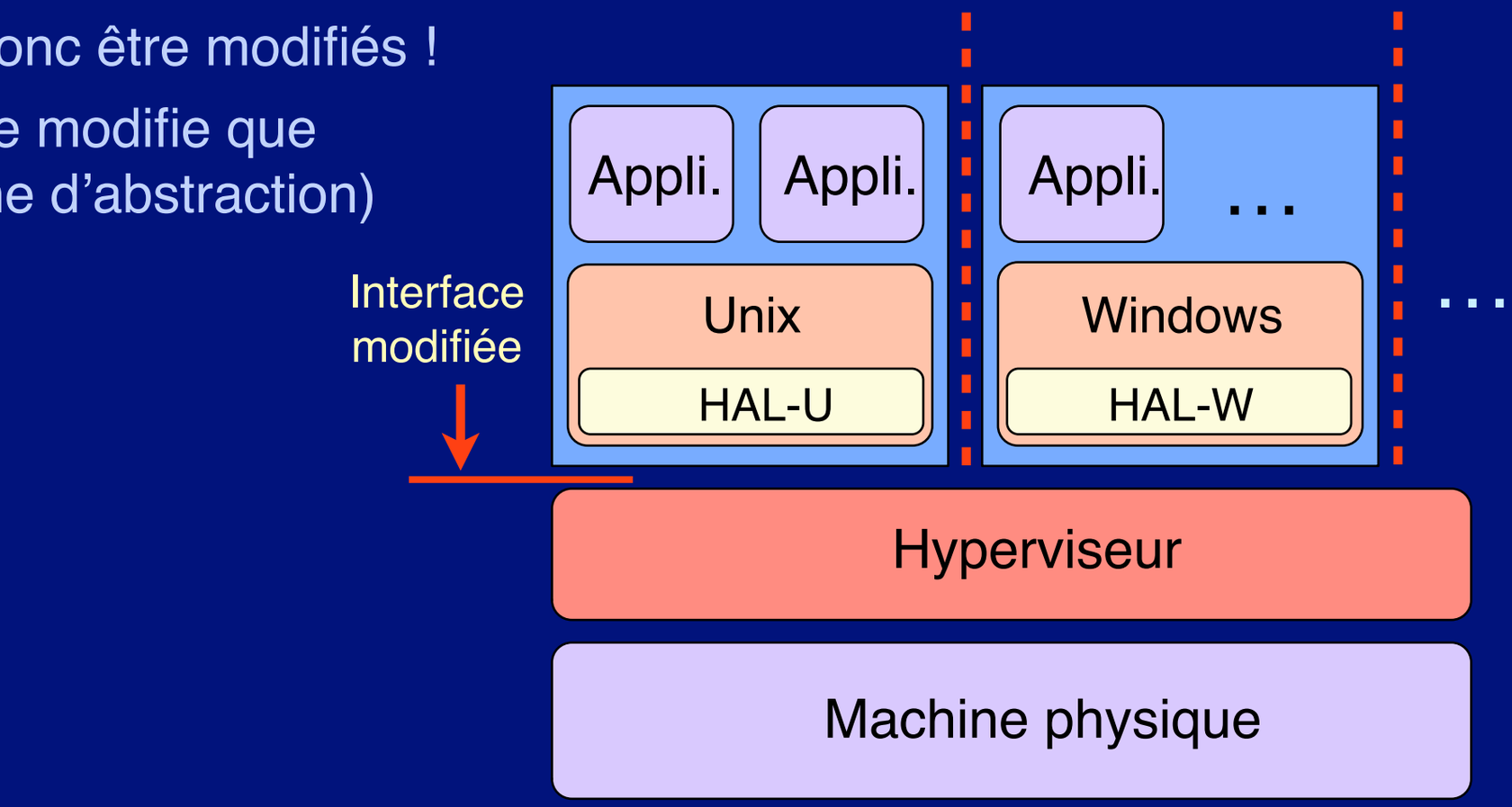


Machines virtuelles



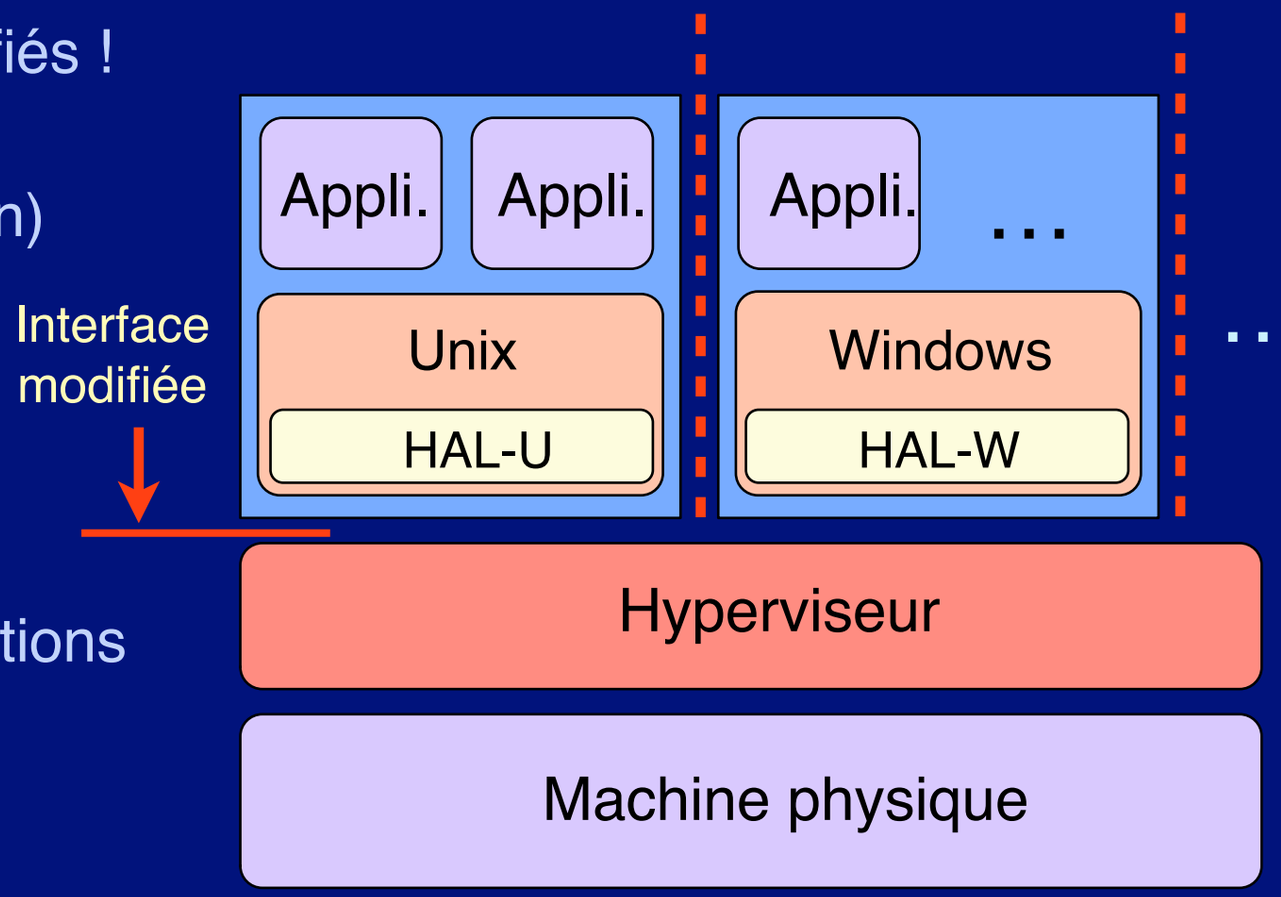
Machines virtuelles

- Comment contourner le problème des instructions multi-modes ?
  - La paravirtualisation : changer l'interface de l'hyperviseur
    - Remplacer les instructions non virtualisables
    - L'interface n'est plus celle de la machine physique
    - Les OS doivent donc être modifiés !
    - En pratique, on ne modifie que la HAL (couche d'abstraction)



Machines virtuelles

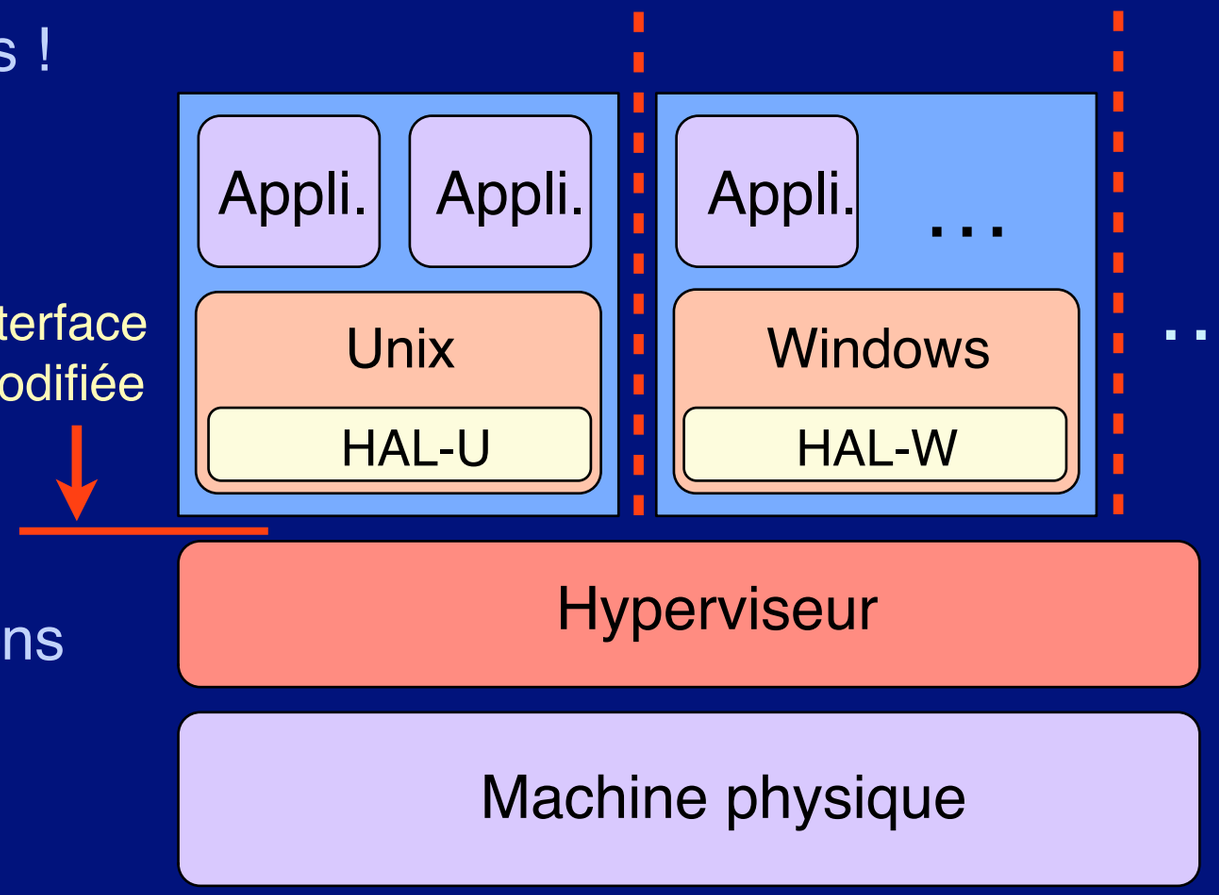
- Comment contourner le problème des instructions multi-modes ?
  - La paravirtualisation : changer l'interface de l'hyperviseur
    - Remplacer les instructions non virtualisables
    - L'interface n'est plus celle de la machine physique
    - Les OS doivent donc être modifiés !
    - En pratique, on ne modifie que la HAL (couche d'abstraction)
  - La traduction dynamique de code binaire :
    - Remplacer à la volée les instructions non virtualisables



Machines virtuelles

- Comment contourner le problème des instructions multi-modes ?
  - La paravirtualisation : changer l'interface de l'hyperviseur
    - Remplacer les instructions non virtualisables
    - L'interface n'est plus celle de la machine physique
    - Les OS doivent donc être modifiés !
    - En pratique, on ne modifie que la HAL (couche d'abstraction)

Interface modifiée



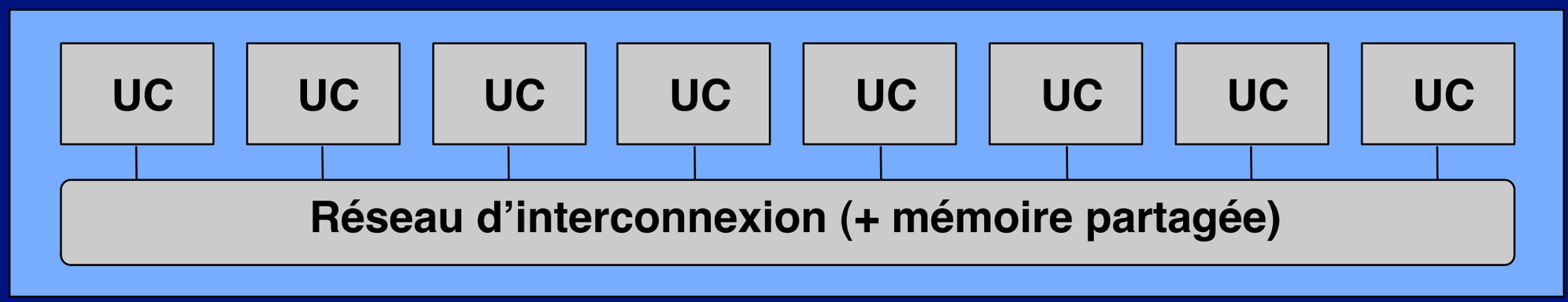
La traduction dynamique de code binaire :

- Remplacer à la volée les instructions non virtualisables

Dans l'avenir :

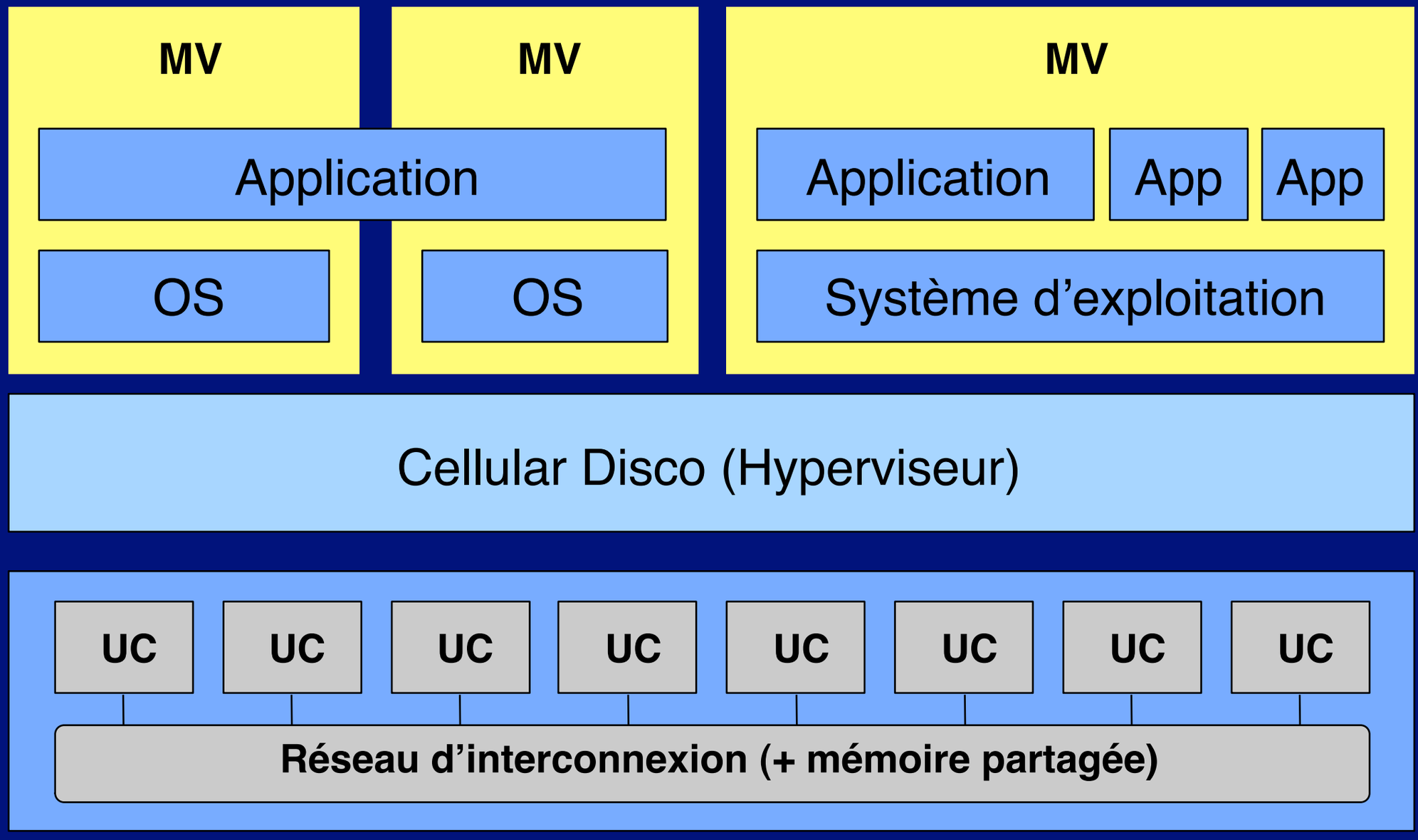
- Les nouveaux processeurs sont conçus pour la virtualisation

Grappes virtuelles



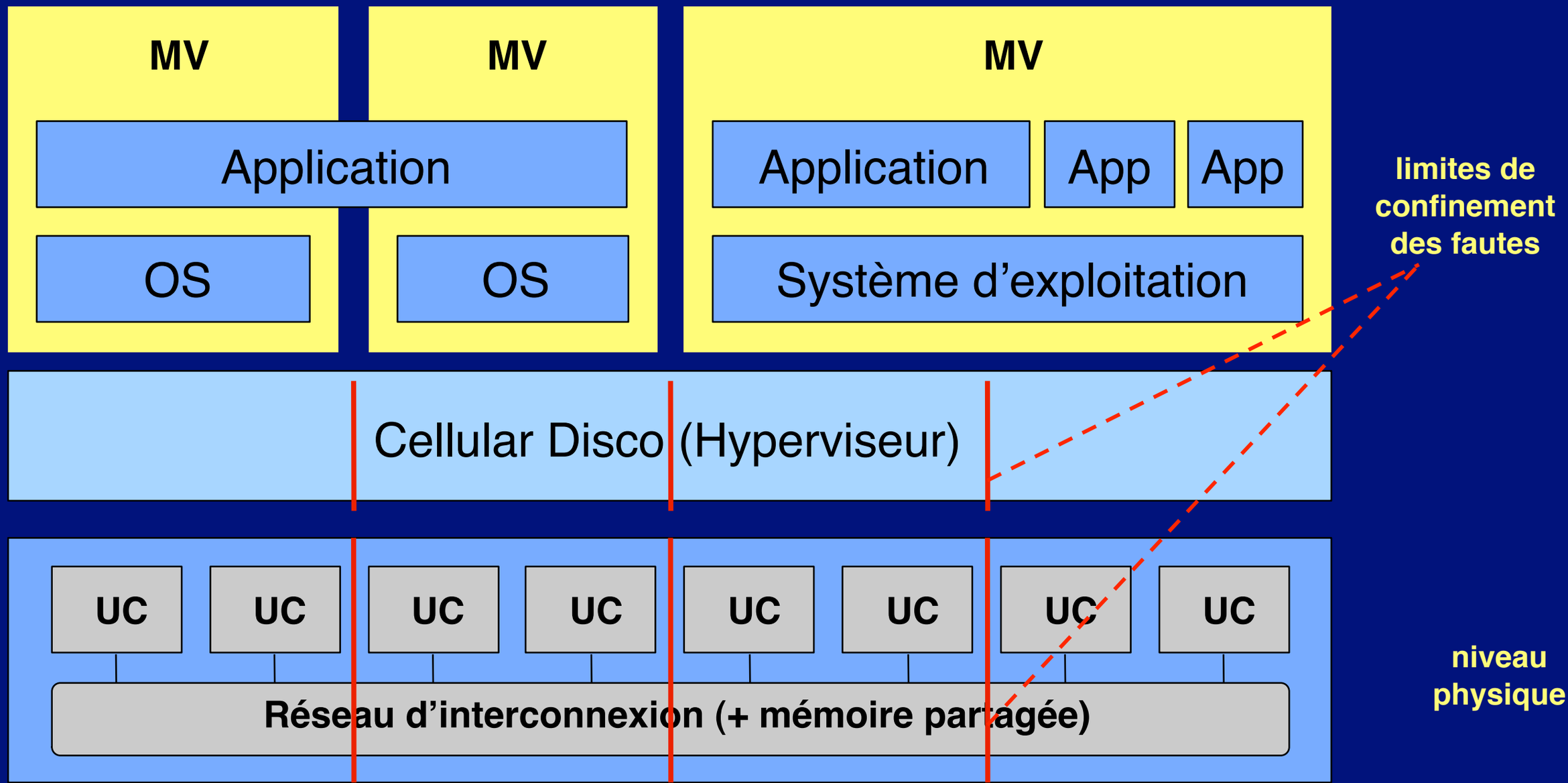
niveau physique

Grappes virtuelles

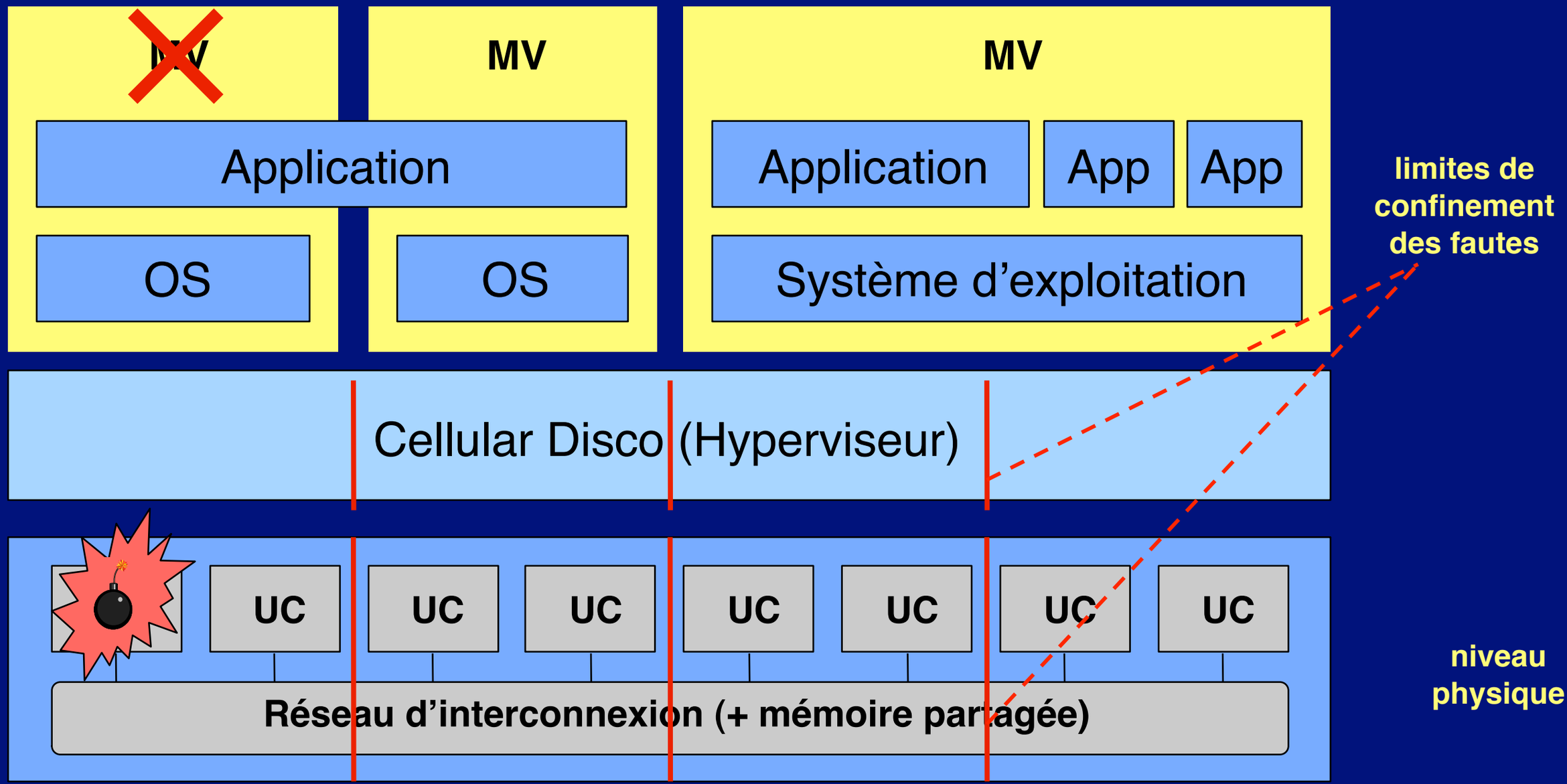


niveau physique

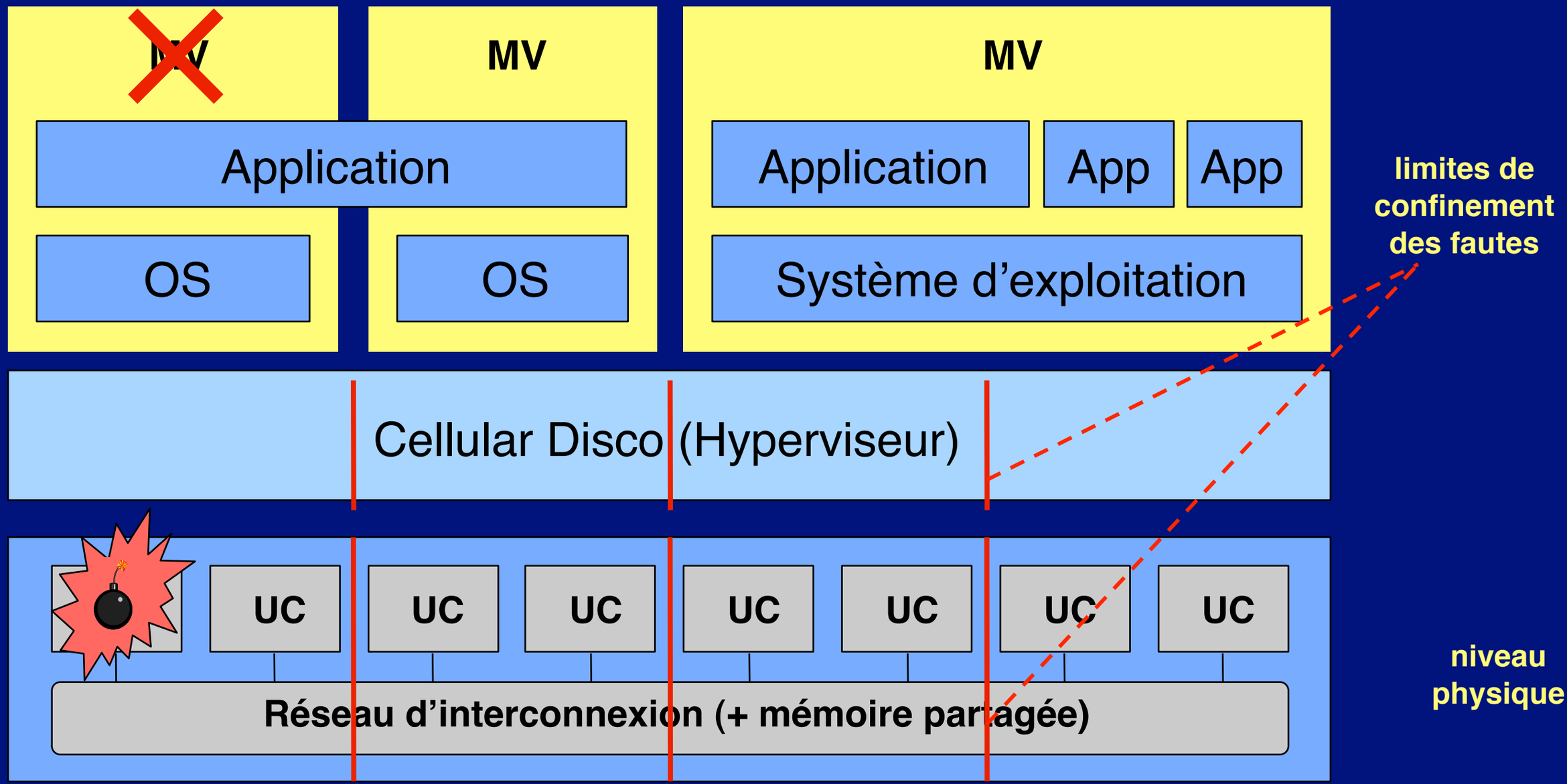
Grappes virtuelles



Grappes virtuelles

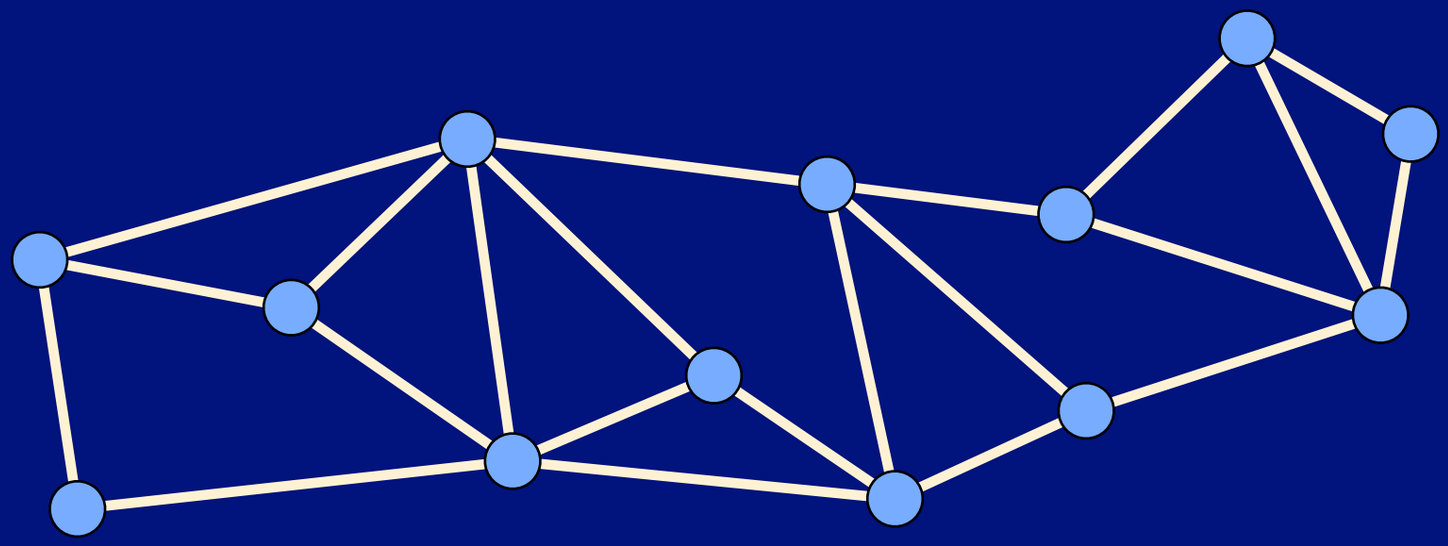


Grappes virtuelles

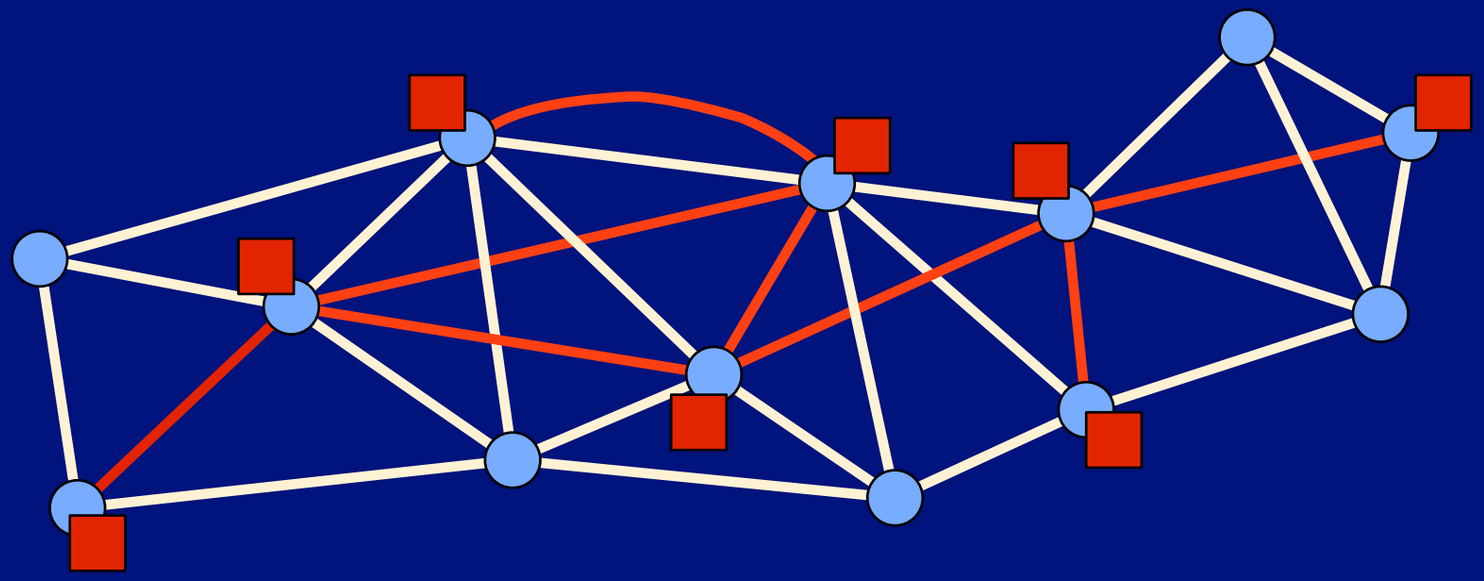


K. Govil, D. Teodosiu, Y. Huang, M. Rosenblum. Cellular Disco: Resource Management Using Virtual Clusters on Shared-Memory Multiprocessors, *ACM Trans. on Computer Systems*, 18(3), Aug. 2000

Réseaux de recouvrement

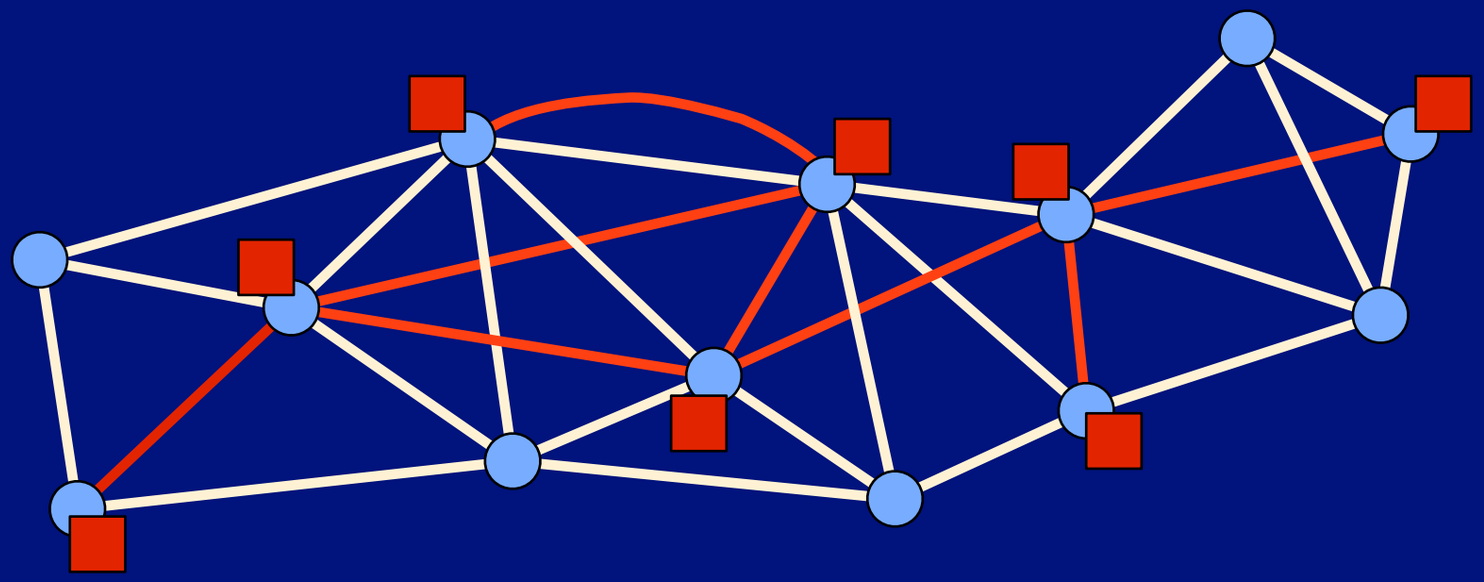


Réseaux de recouvrement



- Objectifs
  - Routage
  - Fiabilité
  - Sécurité, confidentialité
  - Qualité de service
  - Support d’une plate-forme
  - …

Réseaux de recouvrement



- Objectifs
  - Routage
  - Fiabilité
  - Sécurité, confidentialité
  - Qualité de service
  - Support d’une plate-forme
  - …
- Exemples
  - L’Internet lui-même
  - Tables de hachage réparti
  - VPN
  - Service spécialisé (distribution de contenu)
  - …

Cloud computing : l'informatique dans les nuages

- Une vision ancienne…

  «… *computing may someday be organized as a public utility just as the telephone system is a public utility*»

  John McCarthy, 1961

Cloud computing : l'informatique dans les nuages

- Une vision ancienne…

  «… *computing may someday be organized as a public utility just as the telephone system is a public utility*»

  John McCarthy, 1961

- … en voie de réalisation ?

Cloud computing : l'informatique dans les nuages

- Une vision ancienne…

  «… *computing may someday be organized as a public utility just as the telephone system is a public utility*»

  John McCarthy, 1961

- … en voie de réalisation ?
- Virtualisation à grande échelle

  du matériel : *Infrastructure as a Service* (Amazon EC2)

  de l'environnement d'exécution : *Platform as a Service* (Microsoft Azure)

  du support d'applications : *Software as a Service* (Google Docs)

Cloud computing : l'informatique dans les nuages

- Une vision ancienne…

  «… *computing may someday be organized as a public utility just as the telephone system is a public utility*»

  John McCarthy, 1961

- … en voie de réalisation ?
- Virtualisation à grande échelle

  du matériel : *Infrastructure as a Service* (Amazon EC2)

  de l'environnement d'exécution : *Platform as a Service* (Microsoft Azure)

  du support d'applications : *Software as a Service* (Google Docs)

- Un nouveau modèle économique …

  … mais des problèmes potentiels

Cloud computing : l'informatique dans les nuages

- Une vision ancienne…

  «… *computing may someday be organized as a public utility just as the telephone system is a public utility*»

  John McCarthy, 1961

- … en voie de réalisation ?
- Virtualisation à grande échelle

  du matériel : *Infrastructure as a Service* (Amazon EC2)

  de l'environnement d'exécution : *Platform as a Service* (Microsoft Azure)

  du support d'applications : *Software as a Service* (Google Docs)

- Un nouveau modèle économique …

  … mais des problèmes potentiels

- Un domaine de recherche ouvert

  [voir journée *Clouds*, ENS Lyon, 13 décembre 2010]

Questions sur les nuages

- Quels sont les traits originaux ?

Questions sur les nuages

- Quels sont les traits originaux ?
  - «Élasticité» : le client paie ce qu'il consomme, facturation à grain fin
    - Pour le client : économie, pas de risque de sur/sous-dimensionnement, capacité potentiellement illimitée
    - Pour le fournisseur : gain (effet d'échelle, multiplexage statistique, rentabilisation des investissements)
  - Réactivité aux variations de la demande

D. Owens, "Securing Elasticity in the Cloud", *Comm. of the ACM*, vol. 53, no 6, June 2010

Questions sur les nuages

- Quels sont les traits originaux ?
  - «Élasticité» : le client paie ce qu'il consomme, facturation à grain fin
    - Pour le client : économie, pas de risque de sur/sous-dimensionnement, capacité potentiellement illimitée
    - Pour le fournisseur : gain (effet d'échelle, multiplexage statistique, rentabilisation des investissements)
  - Réactivité aux variations de la demande
- Quels sont les risques et les difficultés ?
  - Limites techniques : évolution, passage à grande échelle, latence

D. Owens, "Securing Elasticity in the Cloud", *Comm. of the ACM*, vol. 53, no 6, June 2010

Questions sur les nuages

- Quels sont les traits originaux ?
  - «Élasticité» : le client paie ce qu'il consomme, facturation à grain fin
    - Pour le client : économie, pas de risque de sur/sous-dimensionnement, capacité potentiellement illimitée
    - Pour le fournisseur : gain (effet d'échelle, multiplexage statistique, rentabilisation des investissements)
  - Réactivité aux variations de la demande
- Quels sont les risques et les difficultés ?
  - Limites techniques : évolution, passage à grande échelle, latence
  - Perte de contrôle sur les données (localisation, sécurité, …)

D. Owens, "Securing Elasticity in the Cloud", *Comm. of the ACM*, vol. 53, no 6, June 2010

Questions sur les nuages

- Quels sont les traits originaux ?
  - «Élasticité» : le client paie ce qu'il consomme, facturation à grain fin
    - Pour le client : économie, pas de risque de sur/sous-dimensionnement, capacité potentiellement illimitée
    - Pour le fournisseur : gain (effet d'échelle, multiplexage statistique, rentabilisation des investissements)
  - Réactivité aux variations de la demande

D. Owens, "Securing Elasticity in the Cloud", *Comm. of the ACM*, vol. 53, no 6, June 2010

- Quels sont les risques et les difficultés ?
  - Limites techniques : évolution, passage à grande échelle, latence
  - Perte de contrôle sur les données (localisation, sécurité, …)
  - Pas de baisse significative du prix sans sacrifier
    - les garanties de performances
    - les garanties de disponibilité
    - les garanties de sécurité

D. Durkee, "Why Cloud Computing Will Never Be Free", *Comm. of the ACM*, vol. 53, no 5, May 2010

Virtualisation dans les systèmes embarqués

- Contraintes spécifiques
  - Complexité croissante des applications
  - Maîtrise des performances
  - Réduction de la taille de la base informatique de confiance

Virtualisation dans les systèmes embarqués

- Contraintes spécifiques
  - Complexité croissante des applications
  - Maîtrise des performances
  - Réduction de la taille de la base informatique de confiance
- Un renouveau pour les micro-noyaux
  - Le micro-noyau comme hyperviseur à bas niveau
    - Des systèmes d'exploitation «sur mesure» pour une application
      - permet de réaliser des «appareils virtuels» (appareil + son OS spécifique)
    - Les pilotes peuvent sortir de la base de confiance
    - Les composants critiques peuvent être isolés dans des MV

G. Heiser, "The Role of Virtualization in Embedded Systems", *Proc. First Workshop on Isolation and Integration in Embedded Systems (IIES'08)*, pp 11-16, April 2008

Un micro-noyau vérifié

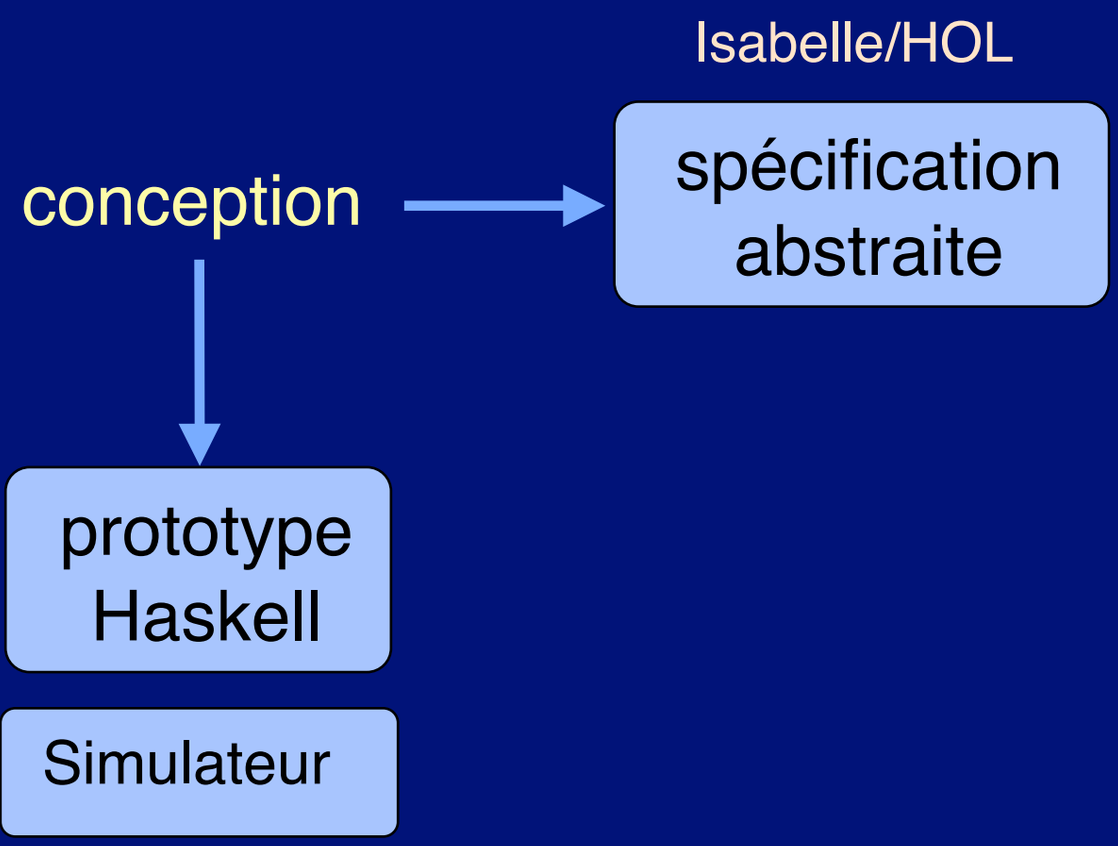
Un micro-noyau vérifié

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

Gerwin Klein et al., “seL4: Formal Verification of an Operating-System Kernel”, *Communications of the ACM*, vol. 53, no 6, pp. 107-115, June 2010

Un micro-noyau vérifié

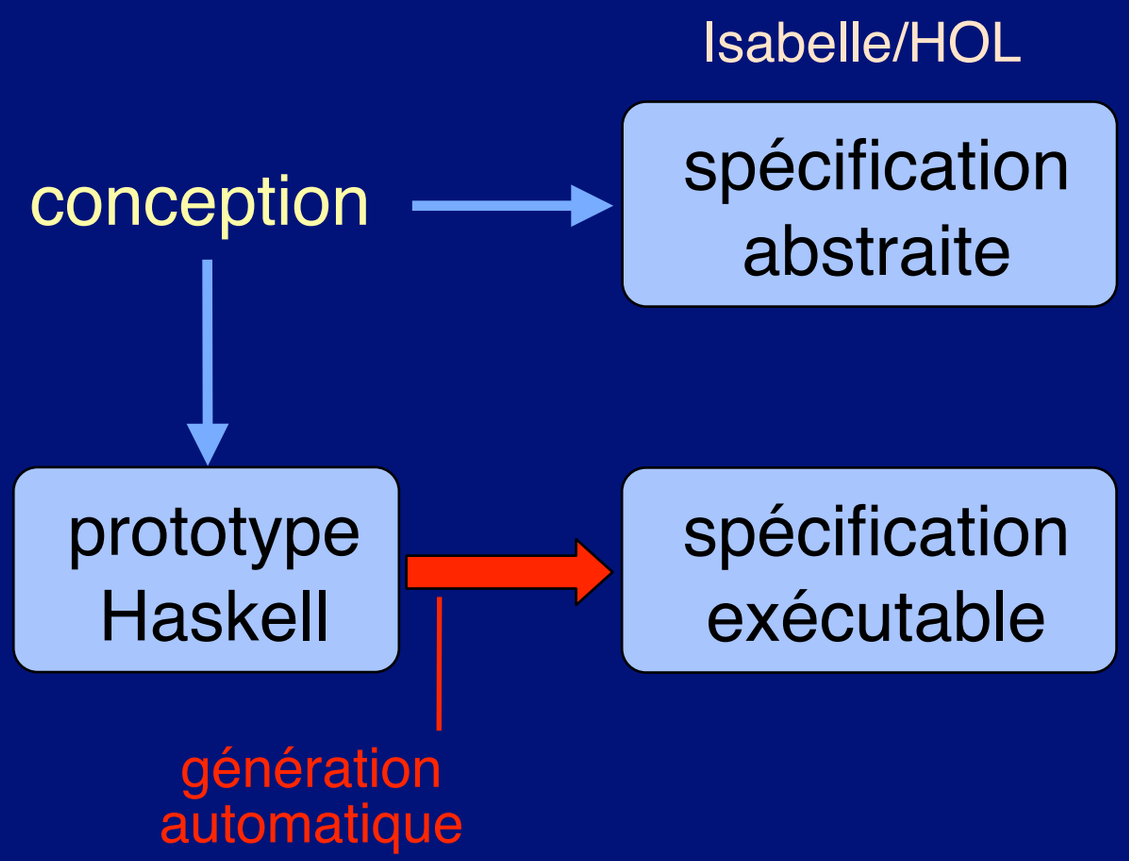
- Une version du micro-noyau L4
  - Machine virtuelle donnant une image simplifiée du matériel
- Difficultés
  - Asynchronisme
  - Gestion de la mémoire (pointeurs)
  - Accès direct aux fonctions du matériel (MMU, …)



Gerwin Klein et al., “seL4: Formal Verification of an Operating-System Kernel”, *Communications of the ACM*, vol. 53, no 6, pp. 107-115, June 2010

Un micro-noyau vérifié

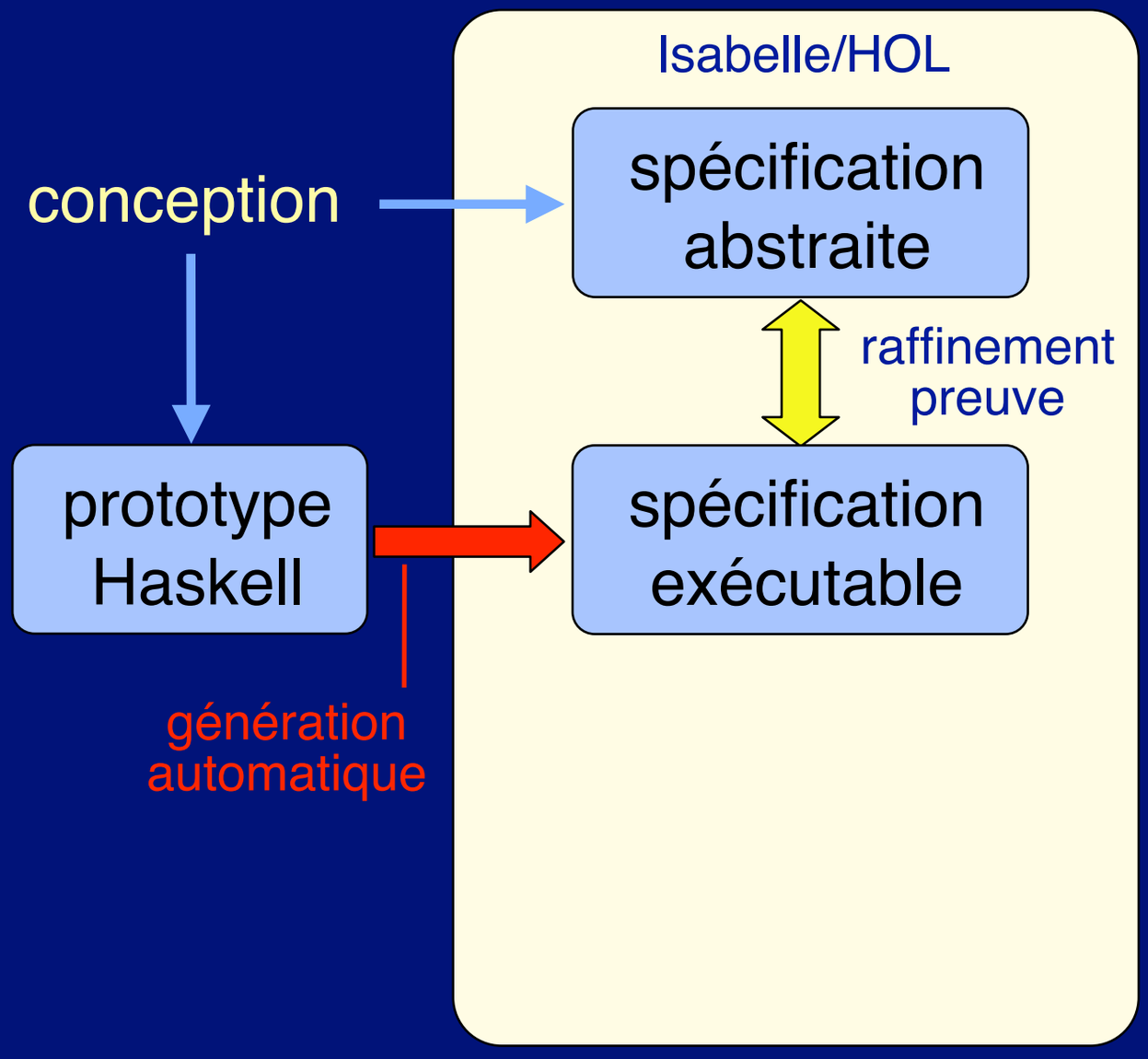
- Une version du micro-noyau L4
  - Machine virtuelle donnant une image simplifiée du matériel
- Difficultés
  - Asynchronisme
  - Gestion de la mémoire (pointeurs)
  - Accès direct aux fonctions du matériel (MMU, …)



Gerwin Klein et al., “seL4: Formal Verification of an Operating-System Kernel”, *Communications of the ACM*, vol. 53, no 6, pp. 107-115, June 2010

Un micro-noyau vérifié

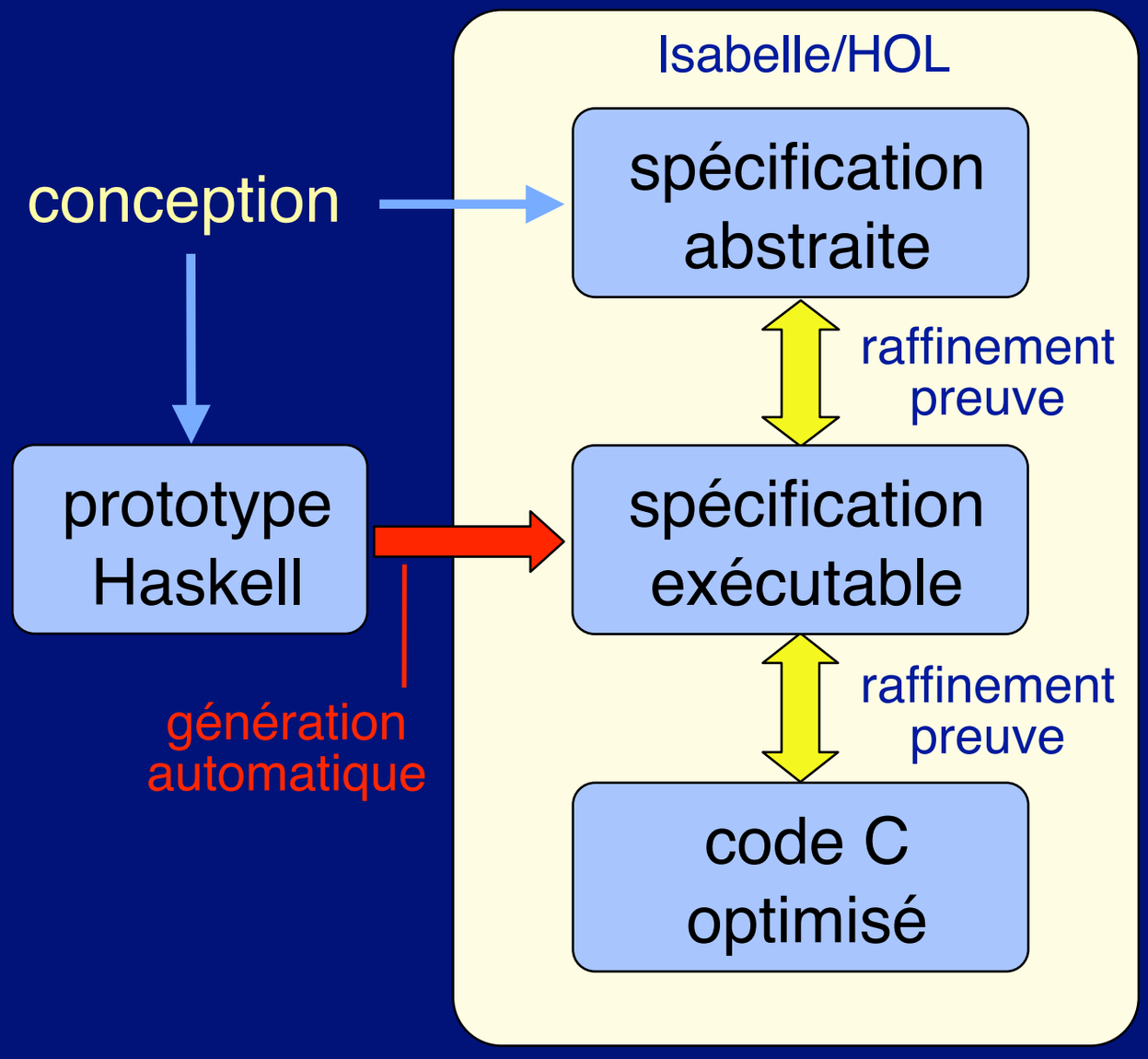
- Une version du micro-noyau L4
  - Machine virtuelle donnant une image simplifiée du matériel
- Difficultés
  - Asynchronisme
  - Gestion de la mémoire (pointeurs)
  - Accès direct aux fonctions du matériel (MMU, …)



Gerwin Klein et al., “seL4: Formal Verification of an Operating-System Kernel”, *Communications of the ACM*, vol. 53, no 6, pp. 107-115, June 2010

Un micro-noyau vérifié

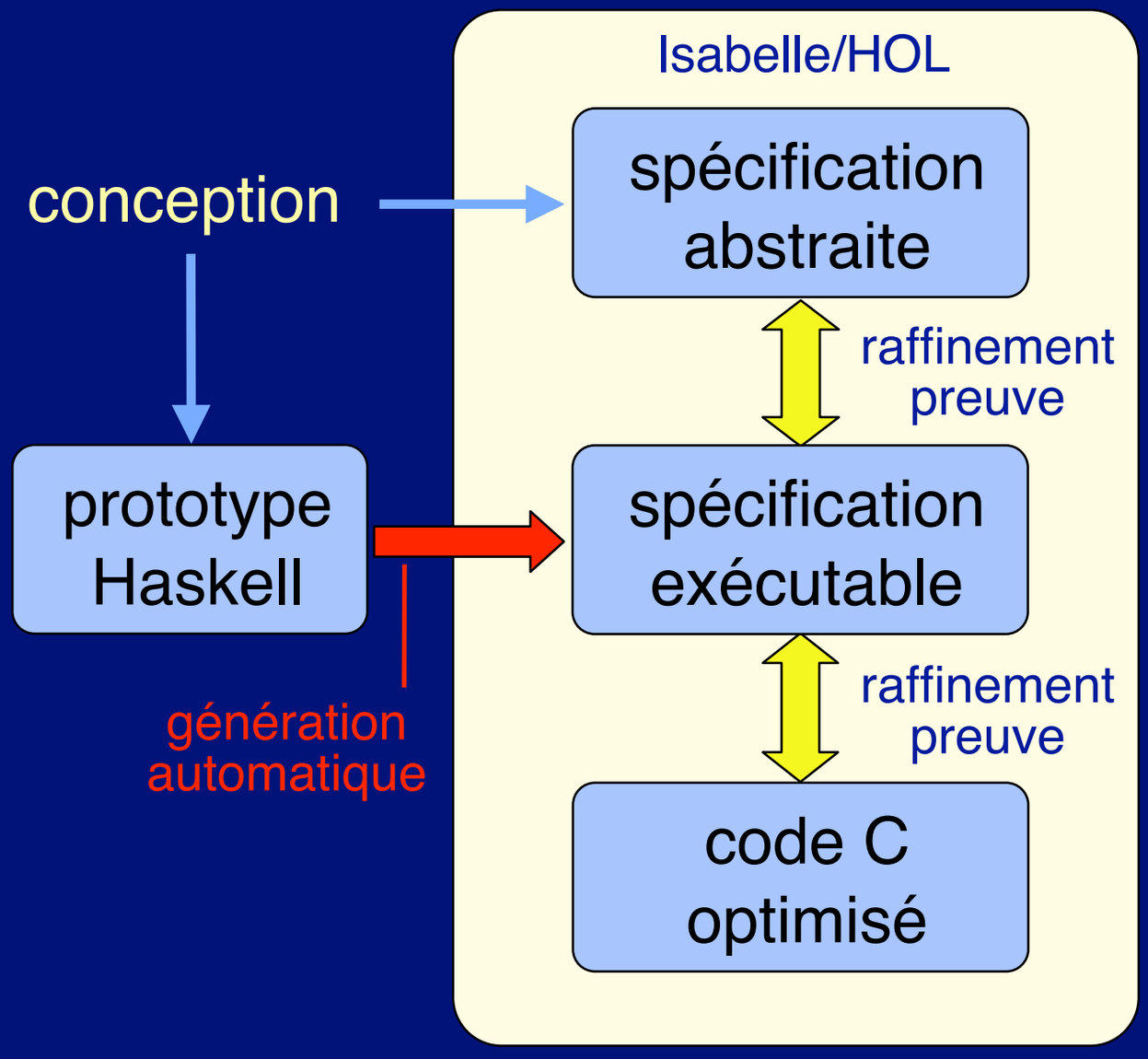
- Une version du micro-noyau L4
  - Machine virtuelle donnant une image simplifiée du matériel
- Difficultés
  - Asynchronisme
  - Gestion de la mémoire (pointeurs)
  - Accès direct aux fonctions du matériel (MMU, …)



Gerwin Klein et al., “seL4: Formal Verification of an Operating-System Kernel”, *Communications of the ACM*, vol. 53, no 6, pp. 107-115, June 2010

Un micro-noyau vérifié

- Une version du micro-noyau L4
  - Machine virtuelle donnant une image simplifiée du matériel
- Difficultés
  - Asynchronisme
  - Gestion de la mémoire (pointeurs)
  - Accès direct aux fonctions du matériel (MMU, …)
- seL4, base d'un "microviseur"
  - OKL4 (Open Kernel Labs)
  - Base installée : un milliard …



Gerwin Klein et al., "seL4: Formal Verification of an Operating-System Kernel", *Communications of the ACM*, vol. 53, no 6, pp. 107-115, June 2010

Avancées et défis pour la virtualisation

- Renaissance et extension du champ de la virtualisation
  - Des nuages aux systèmes embarqués
  - Plate-formes et applications dématérialisés
    - Portables d'un support à un autre, d'un lieu à un autre
  - Outil de gestion globale des ressources
  - Support d'expérimentation

Avancées et défis pour la virtualisation

- Renaissance et extension du champ de la virtualisation
  - Des nuages aux systèmes embarqués
  - Plate-formes et applications dématérialisés
    - Portables d'un support à un autre, d'un lieu à un autre
  - Outil de gestion globale des ressources
  - Support d'expérimentation
- Défis
  - Pour l'utilisateur
    - Contrôle sur le management et les données
    - Garanties de disponibilité et de sécurité
  - Pour le concepteur
    - Modélisation et vérification des hyperviseurs
    - Administration automatique des grandes infrastructures
    - Gestion d'environnements virtuels multiples

Composition (et décomposition)

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

M. D. McIlroy (1968) "Mass Produced Software Components", *in* P. Naur and B. Randell, eds., *Software Engineering*, NATO Science Committee

Composition (et décomposition)

- Un objectif d'apparence simple …
  - Composer un système à partir de pièces élémentaires
  - Éléments réutilisables et remplaçables ("échange standard")
  - Interface visible, réalisation cachée
- … mais une route semée d'embûches
  - Conceptuelles
    - Modèle(s)
    - Expression d'une description globale
    - Garanties
  - Pratiques
    - Configuration et déploiement
    - Gestion de l'évolution
    - Infrastructures

M. D. McIlroy (1968) "Mass Produced Software Components", *in* P. Naur and B. Randell, eds., *Software Engineering*, NATO Science Committee

Propriétés de la composition

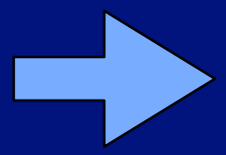
- Composabilité

  Les propriétés de chaque composant sont préservées dans le système composé

Propriétés de la composition

- Composabilité

  Les propriétés de chaque composant sont préservées dans le système composé



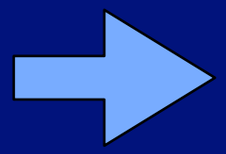
Séparation interface-réalisation

Respect de règles de «bon assemblage»

Propriétés de la composition

- Composabilité

  Les propriétés de chaque composant sont préservées dans le système composé



Séparation interface-réalisation

Respect de règles de «bon assemblage»

- Compositionnalité

  Les propriétés du système composé se déduisent de celles des composants et des règles d'assemblage

Propriétés de la composition

- Composabilité

  Les propriétés de chaque composant sont préservées dans le système composé

  Séparation interface-réalisation

  Respect de règles de «bon assemblage»

Beaucoup plus difficile !

- Compositionnalité

  Les propriétés du système composé se déduisent de celles des composants et des règles d'assemblage

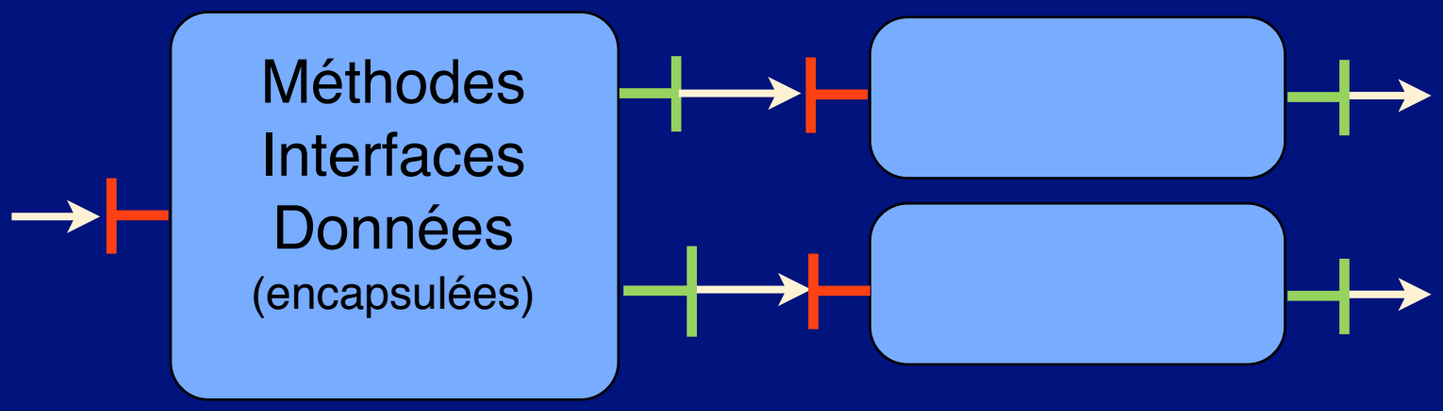
  Modèle formel de composition

  Expression de la sémantique

  Garanties à l'exécution

Deux styles de composition

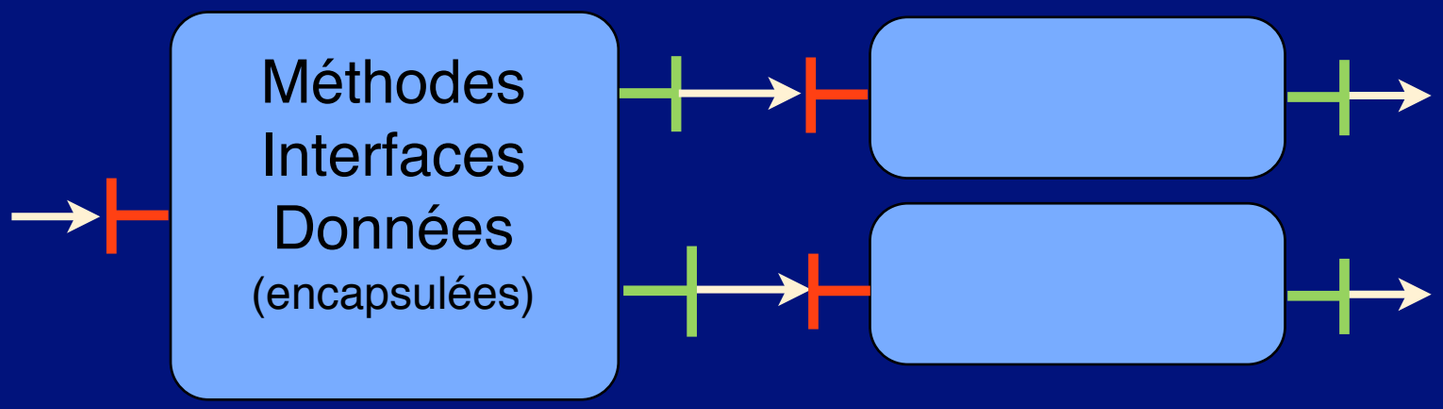
- Dirigé par le flot d'exécution



Dérivé de modules, objets
Divers composants industriels
  Exemples :
    OpenCOM, Fractal, OSGi, …

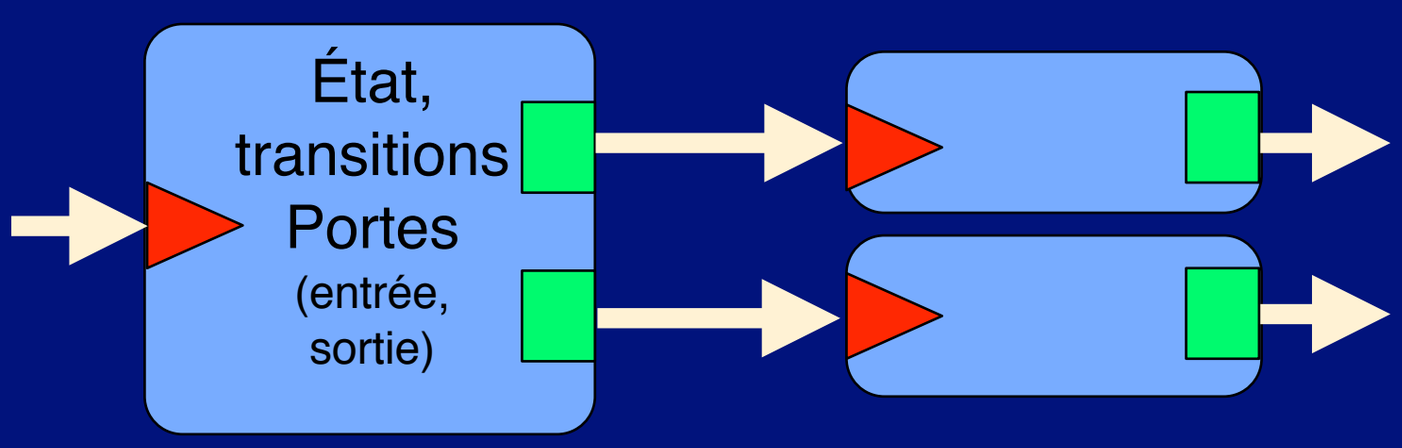
Deux styles de composition

- Dirigé par le flot d'exécution



Dérivé de modules, objets
Divers composants industriels
Exemples :
OpenCOM, Fractal, OSGi, …

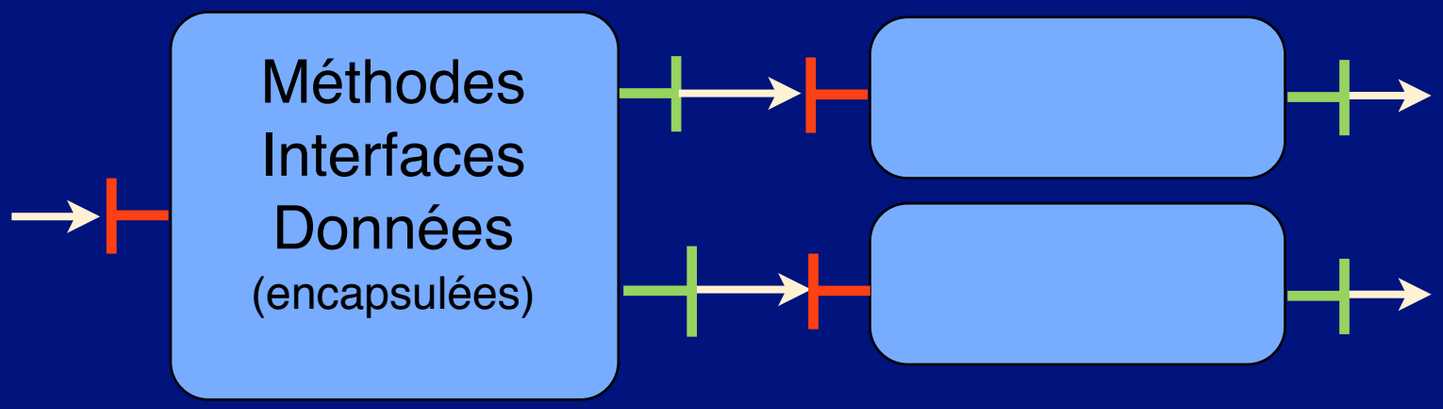
- Dirigé par le flot de données



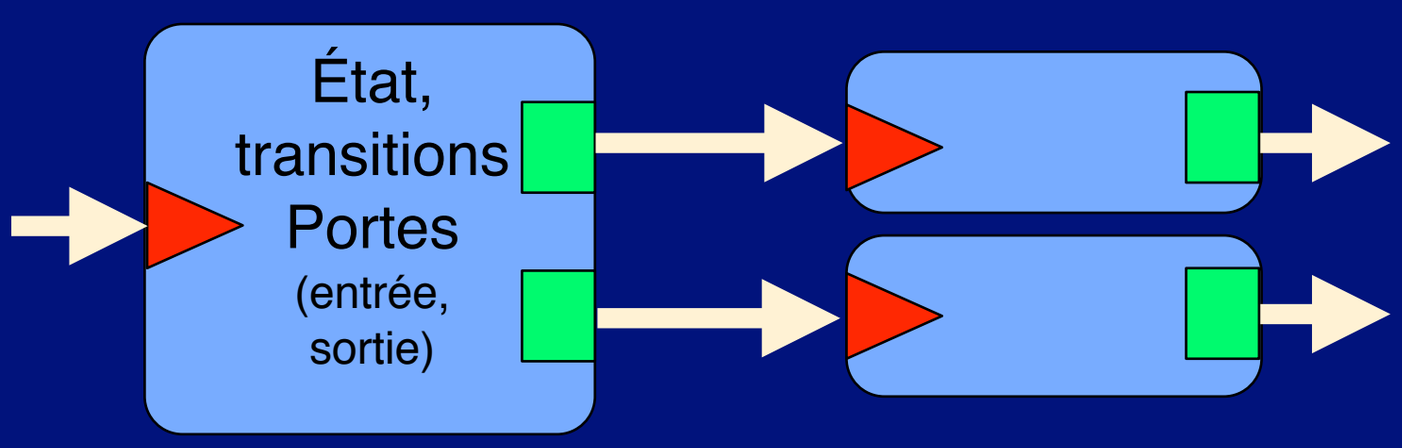
Flots et filtres
Événements
Systèmes réactifs
Exemples :
Ptolemy, Bip, …

Deux styles de composition

- Dirigé par le flot d'exécution



- Dirigé par le flot de données



Les deux peuvent coexister

Dérivé de modules, objets
Divers composants industriels
Exemples :
OpenCOM, Fractal, OSGi, …

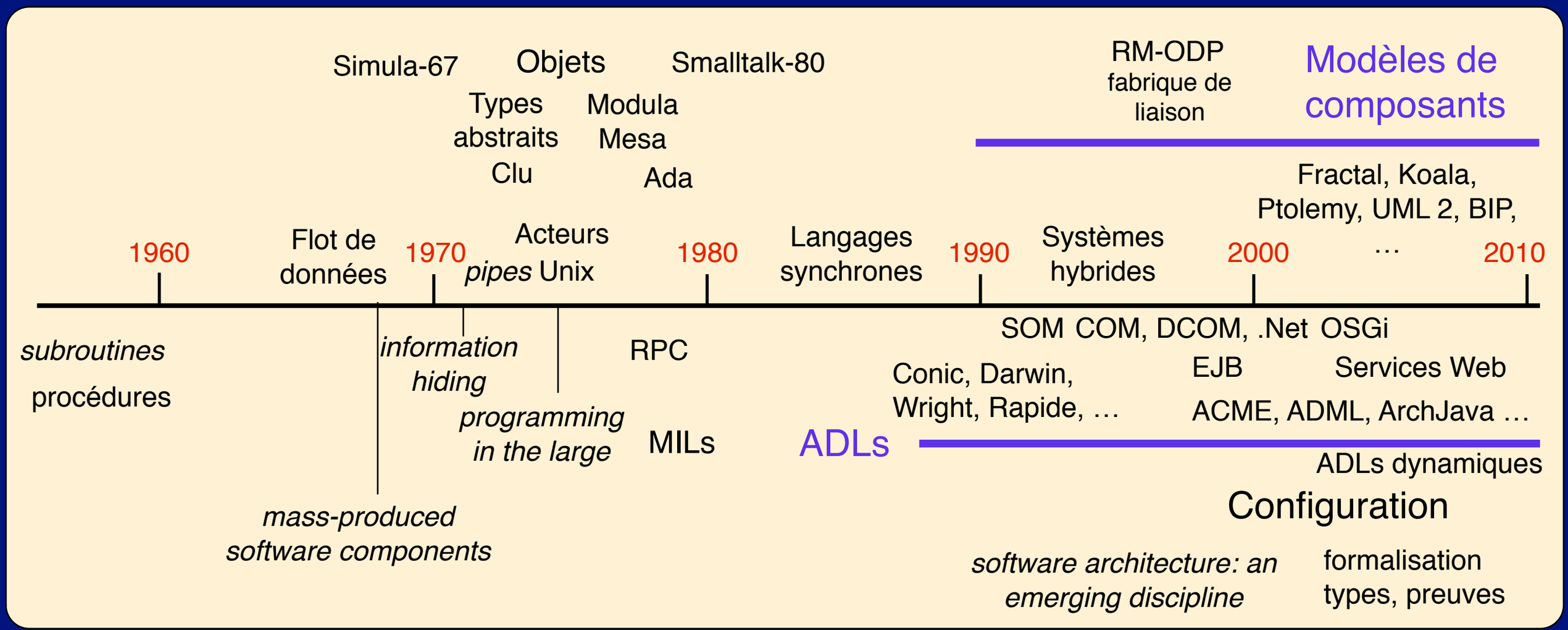
Flots et filtres
Événements
Systèmes réactifs
Exemples :
Ptolemy, Bip, …

Propriétés communes

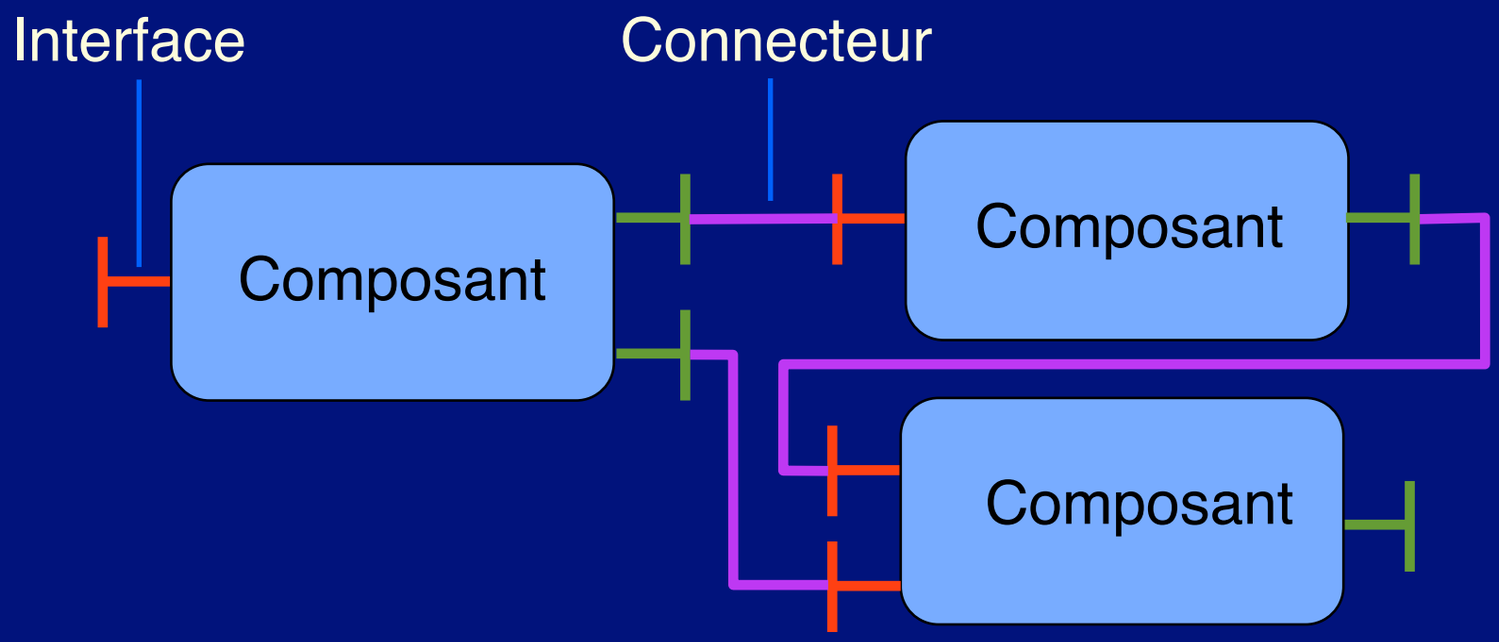
Description globale
Préservation à l'exécution

Indépendance par rapport au langage
Dépendances explicites

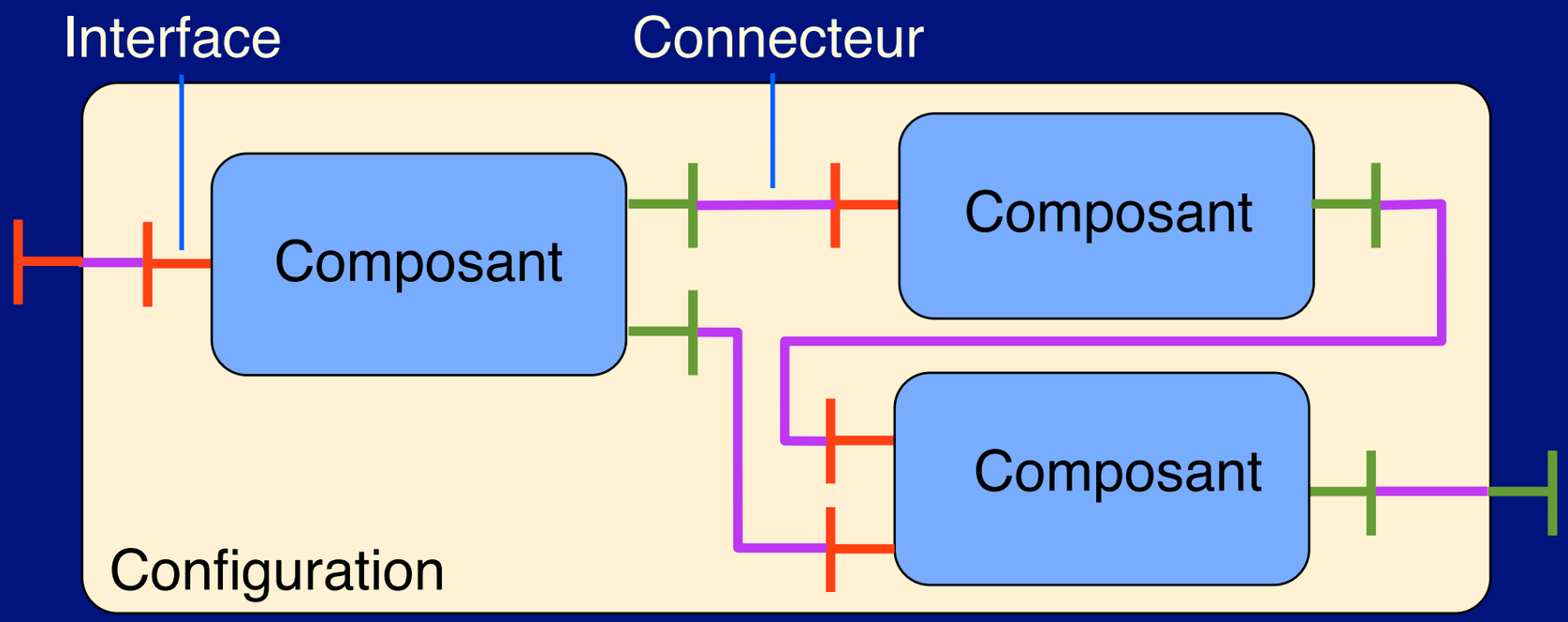
Brève histoire de la (dé)composition



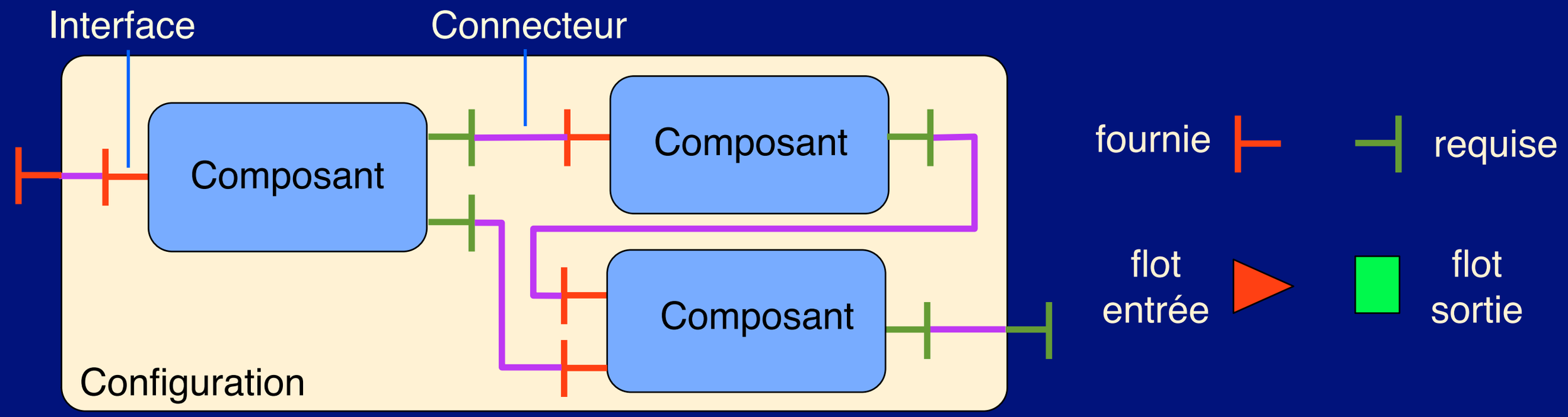
Architecture de systèmes composés (1)



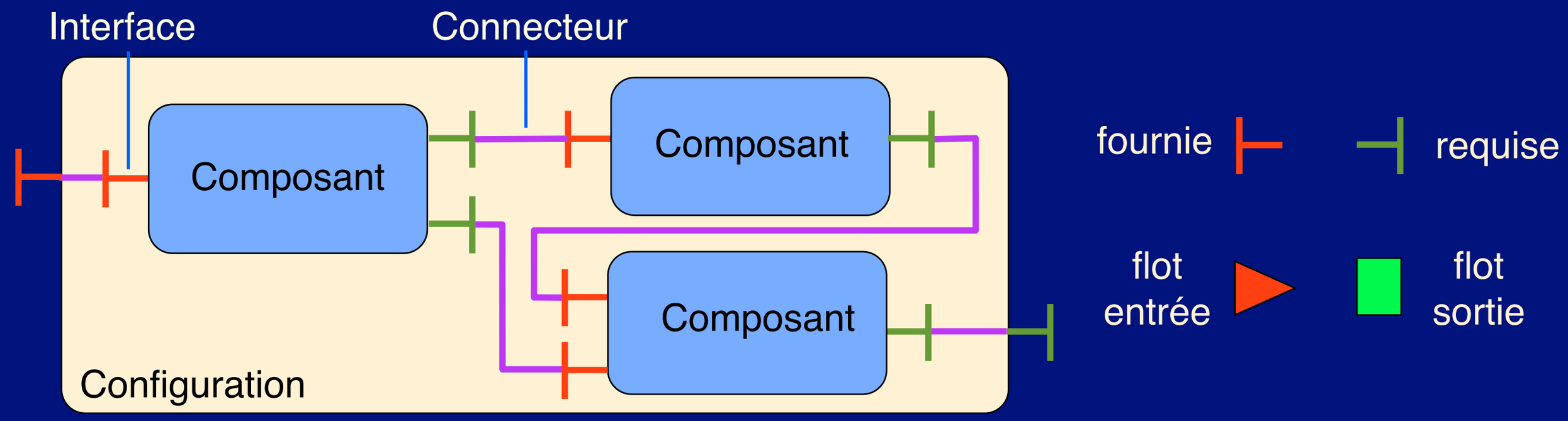
Architecture de systèmes composés (1)



Architecture de systèmes composés (1)

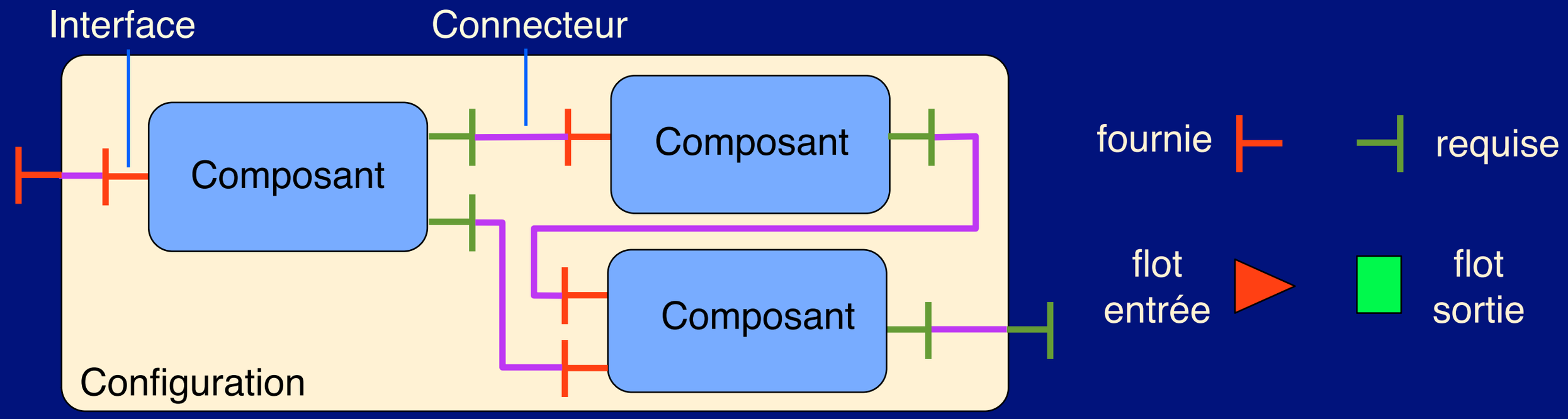


Architecture de systèmes composés (1)



- Flot d'exécution ou flot de données
  - Similitudes
    - Composants matériels ou logiciels
    - Une configuration est un composant
    - Les interfaces sont typées
  - Différences
    - Rôle des interfaces, ou ports
    - Modèle d'interaction (séquentiel, parallèle)
      - Synchrone, rendez-vous, événements, etc.

Architecture de systèmes composés (1)



- Flot d'exécution ou flot de données
  - Similitudes
    - Composants matériels ou logiciels
    - Une configuration est un composant
    - Les interfaces sont typées
  - Différences
    - Rôle des interfaces, ou ports
    - Modèle d'interaction (séquentiel, parallèle)
      - Synchrone, rendez-vous, événements, etc.

Description globale

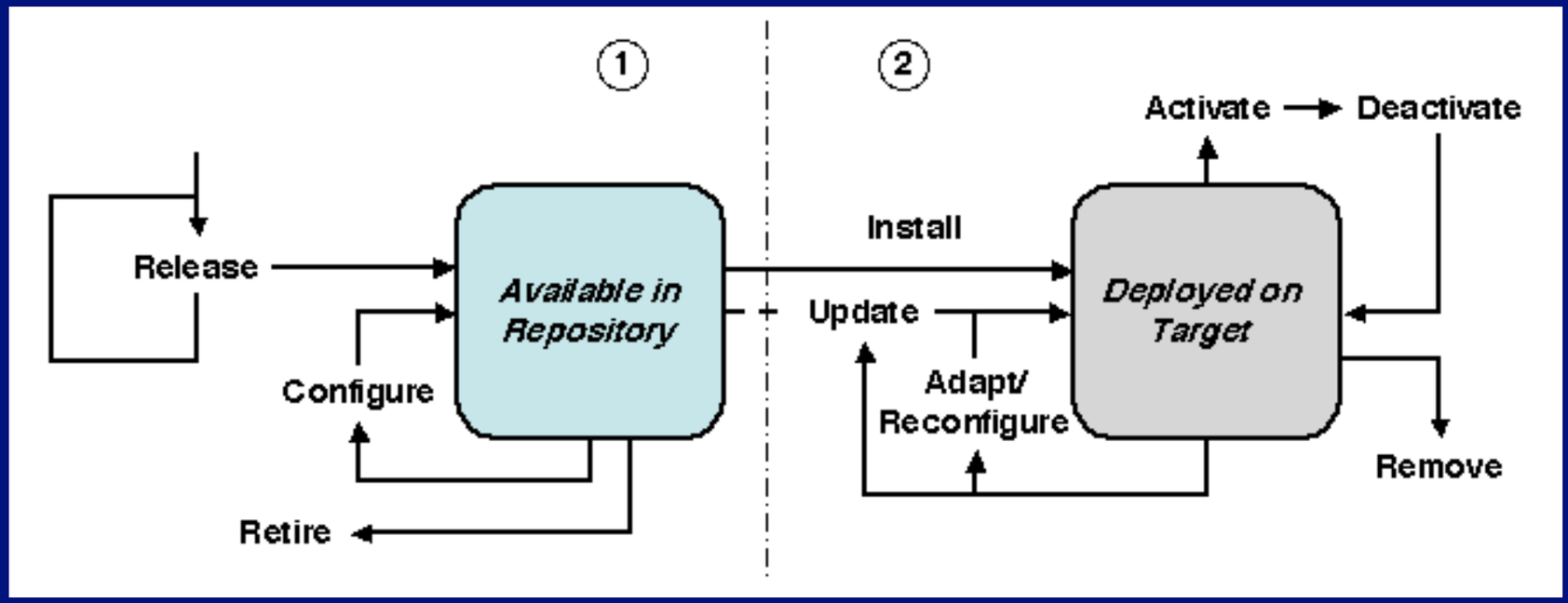
- Implicite (dépendances)
- Explicite (*Architecture Description Language, ADL*)

Rôle des connecteurs

- Assurer la liaison
  - Opération complexe en système réparti
- Gérer l'interaction
  - En particulier dans les modèles parallèles

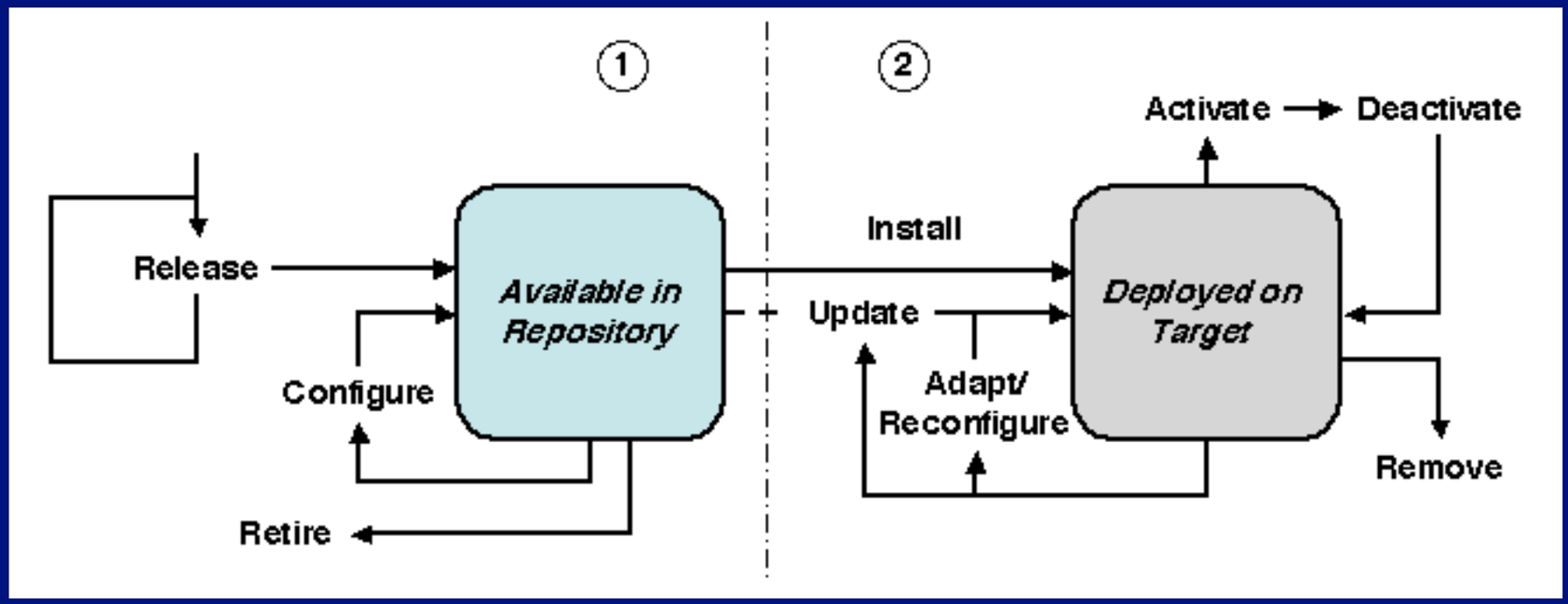
Architecture de systèmes composés (2)

- Cycle de vie
  - Configuration
  - Déploiement
  - Exécution
  - Reconfiguration

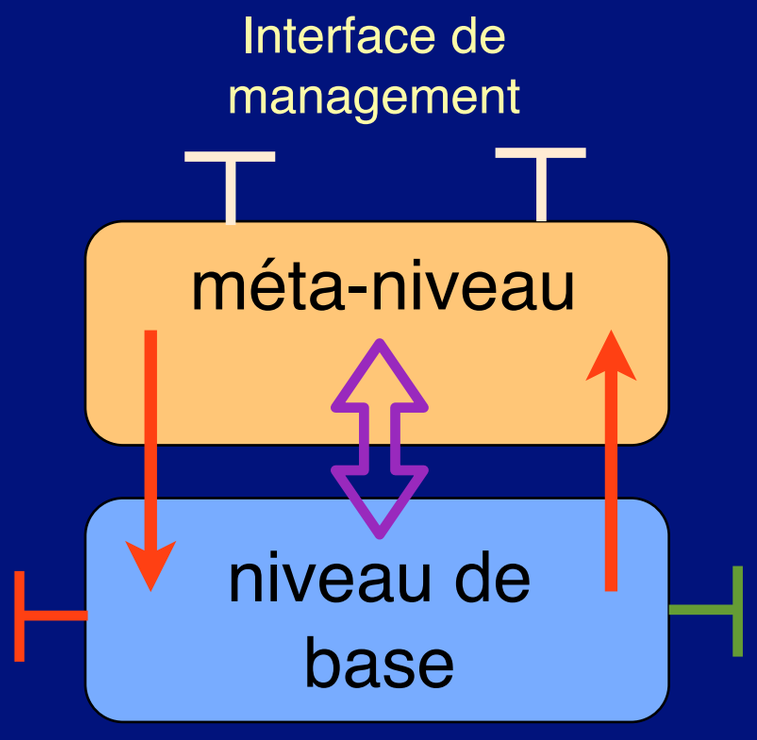


Architecture de systèmes composés (2)

- Cycle de vie
  - Configuration
  - Déploiement
  - Exécution
  - Reconfiguration



- Réflexivité
  - Examiner son propre état («introspection»)
  - Agir sur son propre état («intercession»)
  - Méta-niveau et connexion causale
  - Interface de management
  - Exemples
    - Rainbow, OpenCOM, Fractal, …



Formalisation de la composition : trois exemples

- Déterminer la validité de la *construction* d'un système composé (composabilité)
  - Configuration d'un assemblage de composants

Formalisation de la composition : trois exemples

- Déterminer la validité de la *construction* d'un système composé (composabilité)
  - Configuration d'un assemblage de composants
- Déterminer la validité de *l'exécution* d'un système composé (compositionnalité)
  - Applications du typage

Formalisation de la composition : trois exemples

- Déterminer la validité de la *construction* d'un système composé (composabilité)
  - Configuration d'un assemblage de composants
- Déterminer la validité de *l'exécution* d'un système composé (compositionnalité)
  - Applications du typage
- Déterminer la validité de l'*évolution* d'un système composé
  - Reconfiguration

Un problème de configuration

- Source : EDOS *(Environment for the development & Distribution of Open Source Software)*

  Projet de recherche collaborative du 6ème programme-cadre européen

  Distribution de logiciel libre, constitué de *packages*

  Description implicite, par un ensemble de relations

Référence : http://www.edos-project.org/

Un problème de configuration

- Source : EDOS *(Environment for the development & Distribution of Open Source Software)*

  Projet de recherche collaborative du 6ème programme-cadre européen

  Distribution de logiciel libre, constitué de *packages*

  Description implicite, par un ensemble de relations

- Gestion des relations entre *packages*

  Dépendance : spécifie des *packages* (y compris numéros de version) qui doivent être présents pour faire fonctionner le *package* courant

  Conflit : spécifie des *packages* qui ne peuvent pas coexister avec le *package* courant

  Pré-dépendance : spécifie des *packages* qui doivent déjà être présents pour pouvoir déployer le *package* courant

  Taille typique : 20 000 *packages*, 200 000 relations

Référence : http://www.edos-project.org/

Formalisation de l'installabilité dans EDOS

- chaque *package p* (version *v*) est désigné par une variable booléenne p_v
- chaque contrainte sur une version (e.g., v > 4.0) est traduite dans la disjonction des *packages* qui satisfont cette contrainte, e.g., $p_{v1} \vee p_{v2} \vee \ldots$
- chaque dépendence est interprétée comme une implication, e.g., *aterm* => *libc6* $\wedge$ (*libce6* $\vee$ *xlibs*) $\wedge$ …
- chaque conflit entre des *packages* (soit *a* et *b*) est interprétée comme la formule $\neg (a \wedge b)$

Formalisation de l'installabilité dans EDOS

- chaque *package p* (version *v*) est désigné par une variable booléenne p_v
- chaque contrainte sur une version (e.g., v > 4.0) est traduite dans la disjonction des *packages* qui satisfont cette contrainte, e.g., $p_{v1} \vee p_{v2} \vee \ldots$
- chaque dépendence est interprétée comme une implication, e.g., *aterm* => *libc6* ∧ (*libce6* ∨ *xlibs*) ∧ …
- chaque conflit entre des *packages* (soit *a* et *b*) est interprétée comme la formule ¬ (*a* ∧ *b*)

Alors un *package* p_v est installable ssi il existe une affectation de valeurs qui rende p_v *VRAI* et qui satisfasse la conjonction de toutes les implications logiques introduites par les dépendances et les conflits.

Formalisation de l'installabilité dans EDOS

- chaque *package p* (version *v*) est désigné par une variable booléenne p_v
- chaque contrainte sur une version (e.g., v > 4.0) est traduite dans la disjonction des *packages* qui satisfont cette contrainte, e.g., $p_{v1} \vee p_{v2} \vee \ldots$
- chaque dépendence est interprétée comme une implication, e.g., *aterm* => *libc6* ∧ (*libce6* ∨ *xlibs*) ∧ …
- chaque conflit entre des *packages* (soit *a* et *b*) est interprétée comme la formule ¬ (*a* ∧ *b*)

Alors un *package* p_v est installable ssi il existe une affectation de valeurs qui rende p_v *VRAI* et qui satisfasse la conjonction de toutes les implications logiques introduites par les dépendances et les conflits.

L'installabilité des *packages* est équivalente à la satisfaisabilité booléenne (SAT), problème NP-complet. Dans les situations pratiques, il est néanmoins traitable.

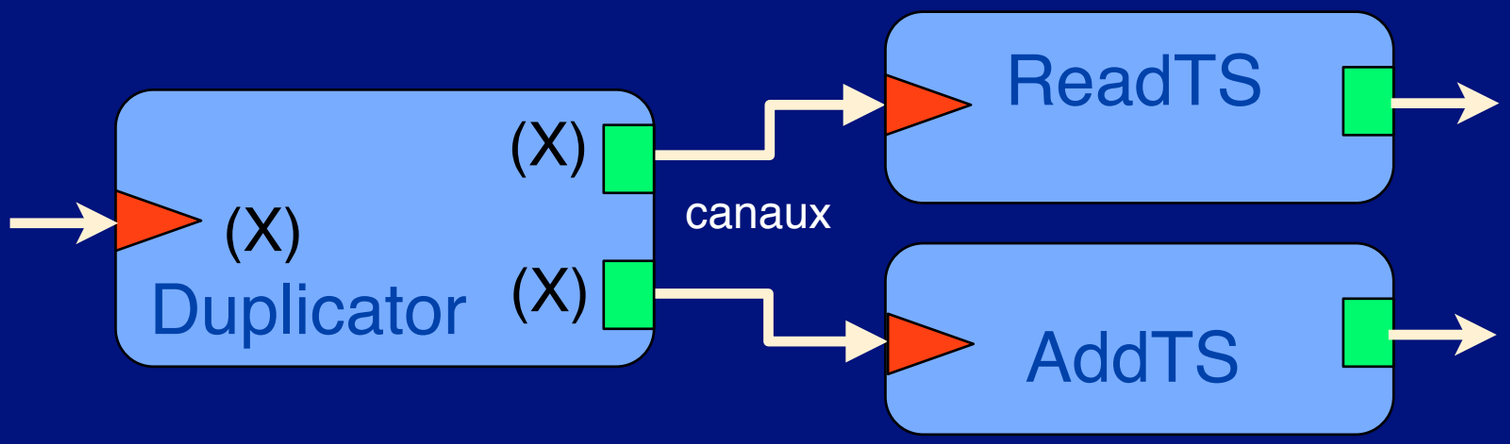
Compositionnalité dans Dream

- Dream : canevas pour la construction d'intergiciels de communication
- Un message est une séquence de champs nommés. Exemple
  - [Nom: "test"] [TS: 10] [IP: 156.875.34.12]
- Un message circule entre des composants qui le traitent
  - Exemple de traitements : ajouter, supprimer, consulter un champ
- Opérations interdites (provoquent une erreur à l'exécution)
  - Ajouter un champ présent, supprimer ou consulter un champ non présent

M. Leclercq, V. Quéma, J.-B. Stefani, "DREAM: A Component Framework for Constructing Resource-Aware, Configurable Middleware," *Distributed Systems Online, IEEE* , 6:9, Sept. 2005

Compositionnalité dans Dream

- Dream : canevas pour la construction d'intergiciels de communication
- Un message est une séquence de champs nommés. Exemple
  - [Nom: "test"] [TS: 10] [IP: 156.875.34.12]
- Un message circule entre des composants qui le traitent
  - Exemple de traitements : ajouter, supprimer, consulter un champ
- Opérations interdites (provoquent une erreur à l'exécution)
  - Ajouter un champ présent, supprimer ou consulter un champ non présent



M. Leclercq, V. Quéma, J.-B. Stefani, "DREAM: A Component Framework for Constructing Resource-Aware, Configurable Middleware," *Distributed Systems Online, IEEE* , 6:9, Sept. 2005

Compositionnalité dans Dream

Dream : canevas pour la construction d'intergiciels de communication

Un message est une séquence de champs nommés. Exemple

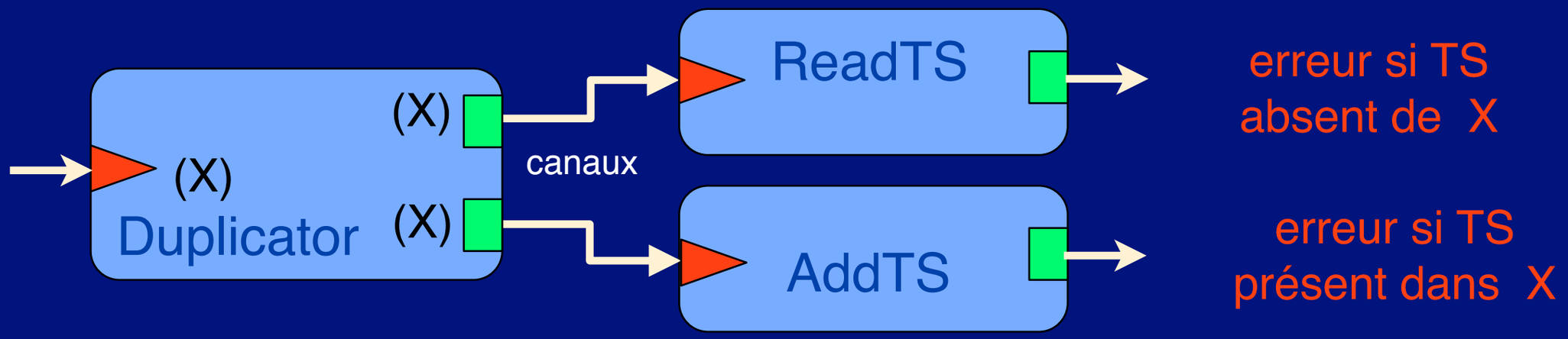
[Nom: "test"] [TS: 10] [IP: 156.875.34.12]

Un message circule entre des composants qui le traitent

Exemple de traitements : ajouter, supprimer, consulter un champ

Opérations interdites (provoquent une erreur à l'exécution)

Ajouter un champ présent, supprimer ou consulter un champ non présent



M. Leclercq, V. Quéma, J.-B. Stefani, "DREAM: A Component Framework for Constructing Resource-Aware, Configurable Middleware," *Distributed Systems Online, IEEE* , 6:9, Sept. 2005

Compositionnalité dans Dream

Dream : canevas pour la construction d'intergiciels de communication

Un message est une séquence de champs nommés. Exemple

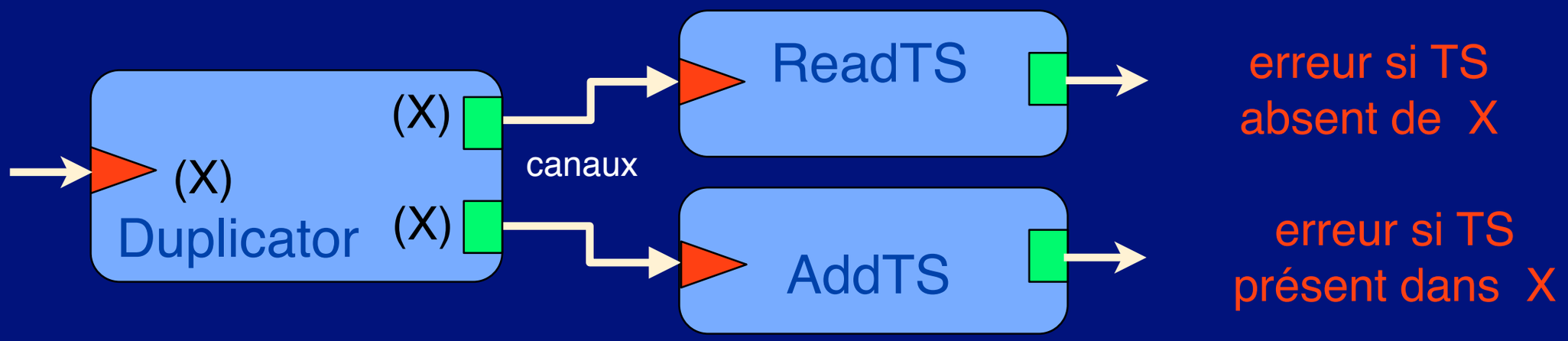
[Nom: "test"] [TS: 10] [IP: 156.875.34.12]

Un message circule entre des composants qui le traitent

Exemple de traitements : ajouter, supprimer, consulter un champ

Opérations interdites (provoquent une erreur à l'exécution)

Ajouter un champ présent, supprimer ou consulter un champ non présent

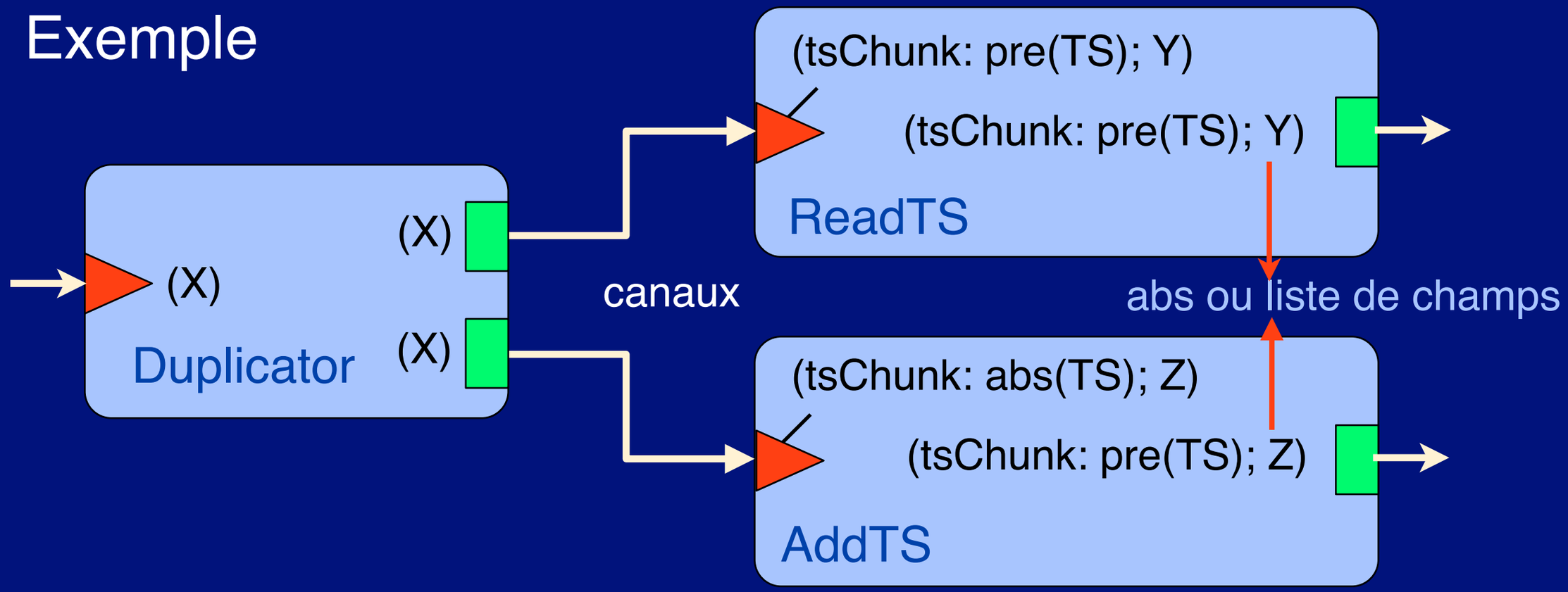


Les types Java ne permettent pas ces vérifications

M. Leclercq, V. Quéma, J.-B. Stefani, "DREAM: A Component Framework for Constructing Resource-Aware, Configurable Middleware," *Distributed Systems Online, IEEE* , 6:9, Sept. 2005

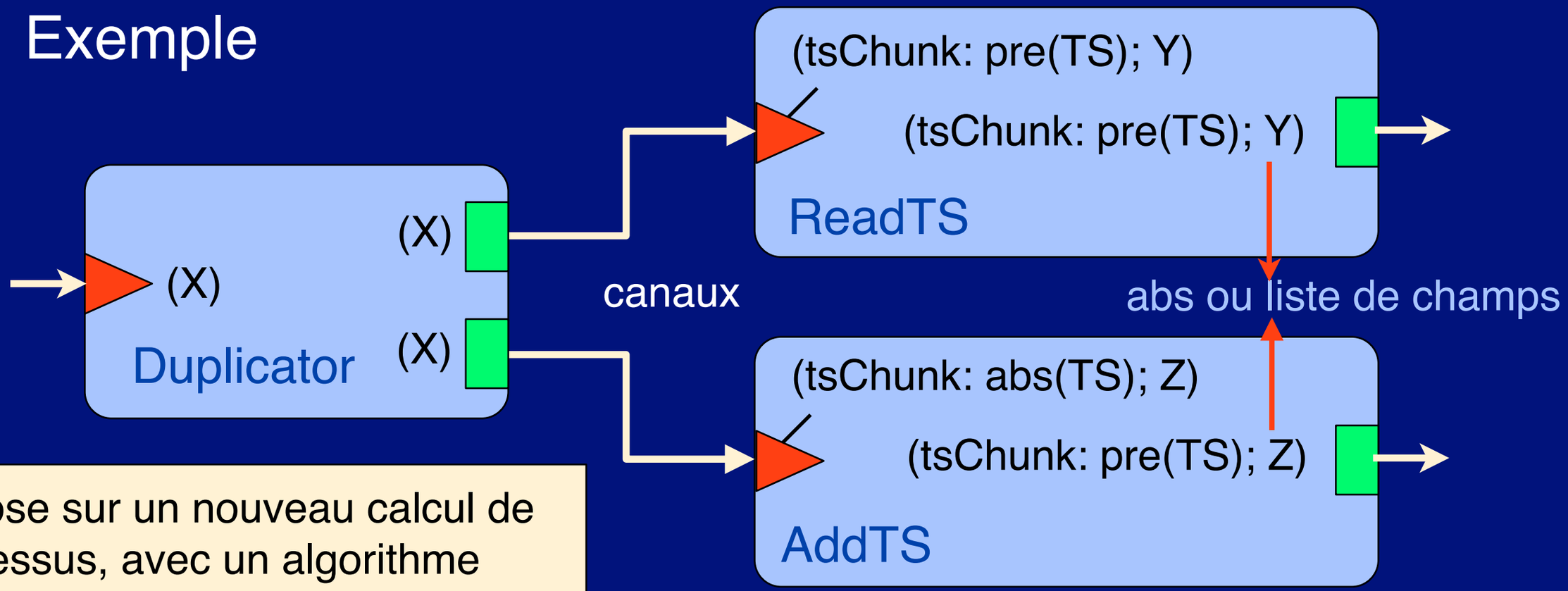
Types Dream

- Exemple



Types Dream

- Exemple

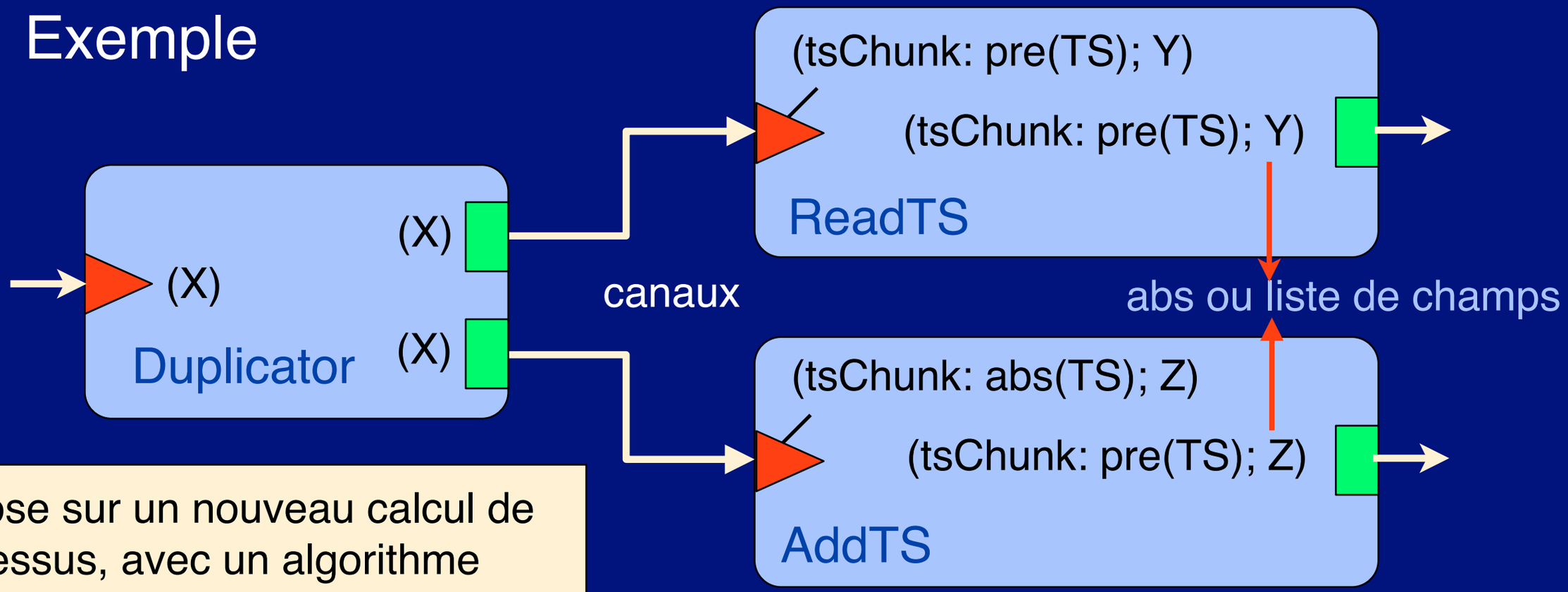


Repose sur un nouveau calcul de processus, avec un algorithme d'inférence de types

Typing Component-Based Communication Systems. M. Lienhardt, C. A. Mezzina, A. Schmitt, and J.-B. Stefani. In *Proc. 11th Formal Methods for Open Object-Based Distributed Systems (FMOODS) & 29th Formal Techniques for Networked and Distributed Systems (FORTE)*, June 2009.

Types Dream

- Exemple



Repose sur un nouveau calcul de processus, avec un algorithme d'inférence de types

Il faut résoudre
(X) = (tsChunk: pre(TS); Y)
(X) = (tsChunk: abs(TS); Z)
ce qui est impossible
Le système n'est pas typable, et aura une erreur à l'exécution

Typing Component-Based Communication Systems. M. Lienhardt, C. A. Mezzina, A. Schmitt, and J.-B. Stefani. In *Proc. 11th Formal Methods for Open Object-Based Distributed Systems (FMOODS) & 29th Formal Techniques for Networked and Distributed Systems (FORTE)*, June 2009.

Reconfiguration (1)

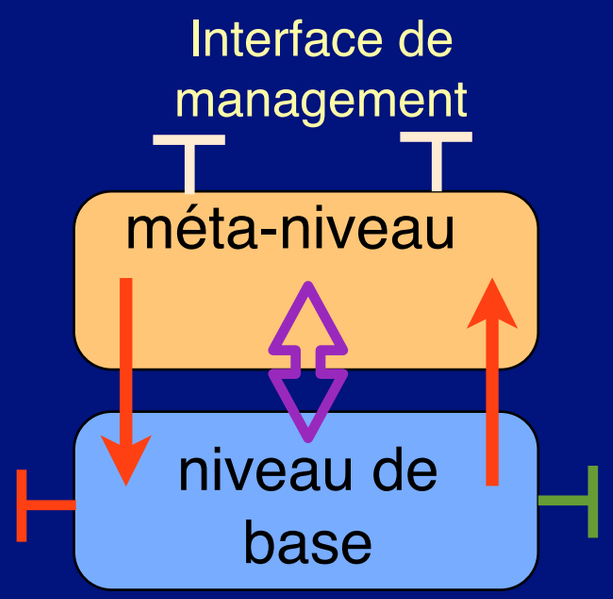
- Qu’est-ce que la reconfiguration (dynamique) ?
  - Changement de la composition et/ou de la structure d’un système en cours d’exécution
    - ajouter/supprimer un composant, déplacer un composant, changer des liaisons, modifier des attributs, …

Reconfiguration (1)

- Qu'est-ce que la reconfiguration (dynamique) ?
  - Changement de la composition et/ou de la structure d'un système en cours d'exécution
    - ajouter/supprimer un composant, déplacer un composant, changer des liaisons, modifier des attributs, …
- Pourquoi reconfigurer un système ?
  - Maintenance, optimisation, insertion de sondes de mesure, réaction aux défaillances, surcharges, attaques, …
  - Opération naturelle pour les mobiles, réseaux de capteurs, etc.

Reconfiguration (1)

- Qu'est-ce que la reconfiguration (dynamique) ?
  - Changement de la composition et/ou de la structure d'un système en cours d'exécution
    - ajouter/supprimer un composant, déplacer un composant, changer des liaisons, modifier des attributs, …
- Pourquoi reconfigurer un système ?
  - Maintenance, optimisation, insertion de sondes de mesure, réaction aux défaillances, surcharges, attaques, …
  - Opération naturelle pour les mobiles, réseaux de capteurs, etc.
- Quelques bonnes pratiques
  - Reconfiguration dirigée par l'architecture
    - Utilisation de la réflexivité
  - Maintien de la cohérence
    - Préservation d'invariants
  - Perturbation minimale



Reconfiguration (2)

- Spécification d'invariants
  - Architecturaux : contraintes sur la structure
    - Pas de circuit, pas de partage de composant, contraintes spécifiques à l'application
  - Comportementaux : contraintes sur l'état
    - Cohérence de l'état, résistance aux pannes

Reconfiguration (2)

- Spécification d'invariants
  - Architecturaux : contraintes sur la structure
    - Pas de circuit, pas de partage de composant, contraintes spécifiques à l'application
  - Comportementaux : contraintes sur l'état
    - Cohérence de l'état, résistance aux pannes
- Utilisation de la réflexivité
  - Réification de l'architecture
  - Inspection et modification de l'état

Reconfiguration (2)

- Spécification d'invariants
  - Architecturaux : contraintes sur la structure
    - Pas de circuit, pas de partage de composant, contraintes spécifiques à l'application
  - Comportementaux : contraintes sur l'état
    - Cohérence de l'état, résistance aux pannes
- Utilisation de la réflexivité
  - Réification de l'architecture
  - Inspection et modification de l'état
- Mécanique de la reconfiguration
  - Détection de "quiescence"
  - Interposition dynamique
  - Remplacement de composant

Mise en œuvre de la reconfiguration : exemple

- Un modèle de composants
  - Fractal, fournit la réflexivité (interfaces de management)

M. Léger, Th. Ledoux, Th. Coupaye. Reliable Dynamic Reconfigurations in a Reflective Component Model, *Proc. CBSE 2010*, LNCS 6092, pp. 74-92, Springer Verlag

Mise en œuvre de la reconfiguration : exemple

- Un modèle de composants
  - Fractal, fournit la réflexivité (interfaces de management)
- Un modèle pour l'expression des règles d'assemblage
  - Relations entre interfaces, attributs, composants
    - ex : un composant possède une interface, une interface est liée à une interface, un composant contient un composant, …
  - Contraintes d'intégrité : prédicats sur les éléments et relations

M. Léger, Th. Ledoux, Th. Coupaye. Reliable Dynamic Reconfigurations in a Reflective Component Model, *Proc. CBSE 2010*, LNCS 6092, pp. 74-92, Springer Verlag

Mise en œuvre de la reconfiguration : exemple

- Un modèle de composants
  - Fractal, fournit la réflexivité (interfaces de management)
- Un modèle pour l'expression des règles d'assemblage
  - Relations entre interfaces, attributs, composants
    - ex : un composant possède une interface, une interface est liée à une interface, un composant contient un composant, …
  - Contraintes d'intégrité : prédicats sur les éléments et relations
- Un outil de vérification des contraintes d'intégrité
  - Cohérence globale : Alloy (langage déclaratif + solveur SAT)
  - Réalisation en Fractal : FPath

M. Léger, Th. Ledoux, Th. Coupaye. Reliable Dynamic Reconfigurations in a Reflective Component Model, *Proc. CBSE 2010*, LNCS 6092, pp. 74-92, Springer Verlag

Mise en œuvre de la reconfiguration : exemple

- Un modèle de composants
  - Fractal, fournit la réflexivité (interfaces de management)
- Un modèle pour l'expression des règles d'assemblage
  - Relations entre interfaces, attributs, composants
    - ex : un composant possède une interface, une interface est liée à une interface, un composant contient un composant, …
  - Contraintes d'intégrité : prédicats sur les éléments et relations
- Un outil de vérification des contraintes d'intégrité
  - Cohérence globale : Alloy (langage déclaratif + solveur SAT)
  - Réalisation en Fractal : FPath
- Une exécution transactionnelle
  - Opérations via FScript (action sur interfaces de management)
  - Garanties ACID

M. Léger, Th. Ledoux, Th. Coupaye. Reliable Dynamic Reconfigurations in a Reflective Component Model, *Proc. CBSE 2010*, LNCS 6092, pp. 74-92, Springer Verlag

Avancées et défis pour la composition

- Avancées
  - Composants, architectures logicielles
  - Patrons et canevas pour la composition
  - Début de formalisation

Avancées et défis pour la composition

- Avancées
  - Composants, architectures logicielles
  - Patrons et canevas pour la composition
  - Début de formalisation
- Défis
  - Bases formelles
    - Multiplicité de modèles et des langages
      - peut-être inévitable …
    - Intégration matériel-logiciel
    - Compositionnalité
      - en particulier : performances, synchronisation, temps physique
    - Systèmes à grande échelle

Systèmes auto-adaptables

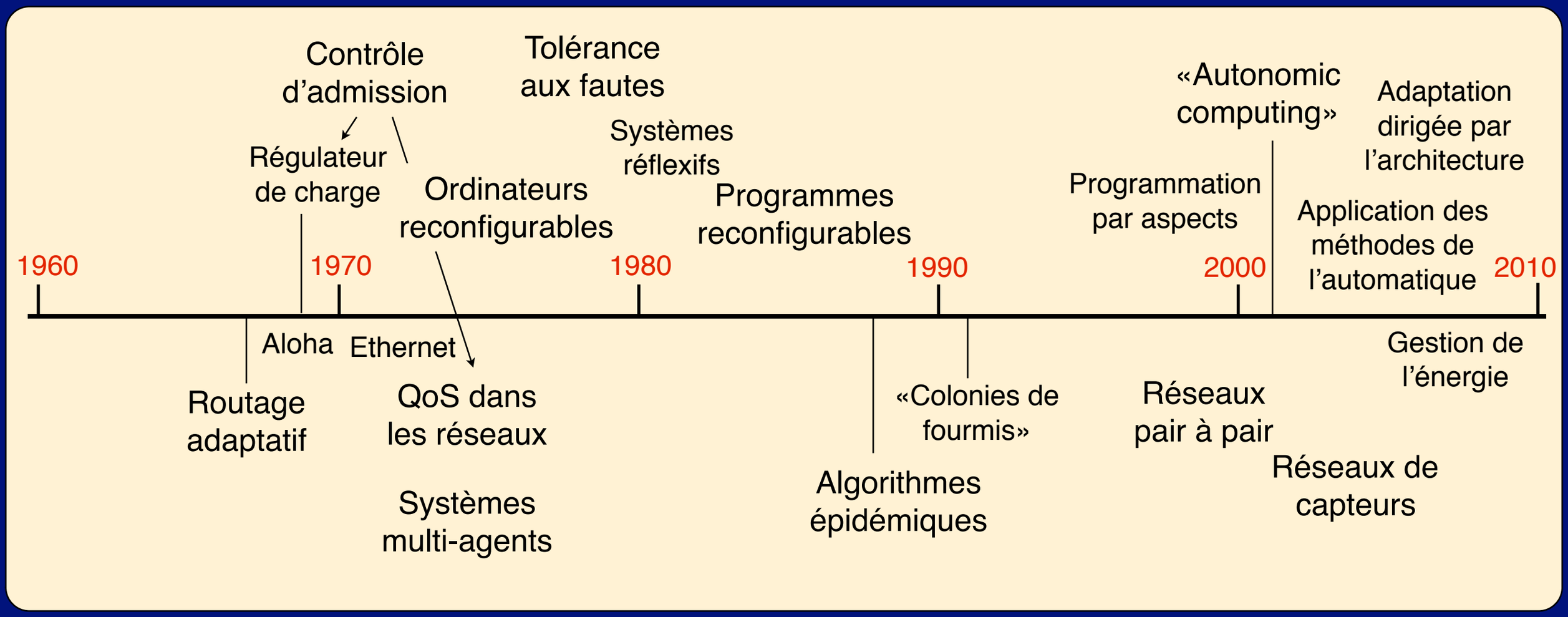
- Pourquoi des systèmes auto-adaptables ?
  - Maintenir l'intégrité et la qualité de service d'un système …
  - … dans un environnement variable et imprévisible
    - Besoins
    - Charge
    - Défaillances
    - Attaques

Systèmes auto-adaptables

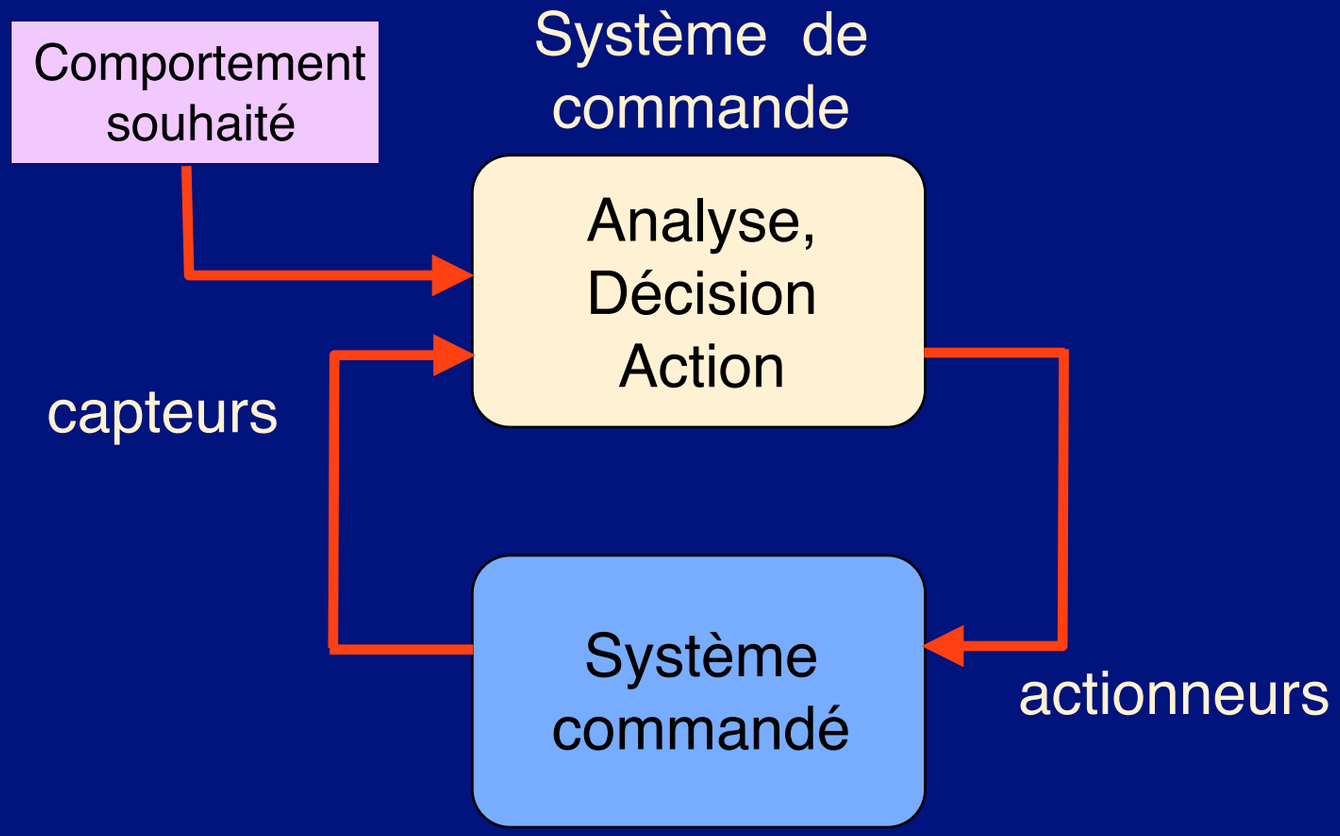
- Pourquoi des systèmes auto-adaptables ?
  - Maintenir l'intégrité et la qualité de service d'un système …
  - … dans un environnement variable et imprévisible
    - Besoins
    - Charge
    - Défaillances
    - Attaques

- Approches de l'auto-adaptation
  - Centralisée
    - Comportement global imposé
    - Modèle : automatique, boucle de commande
  - Décentralisée
    - Comportement global résultant d'interactions locales
    - Modèle : systèmes biologiques

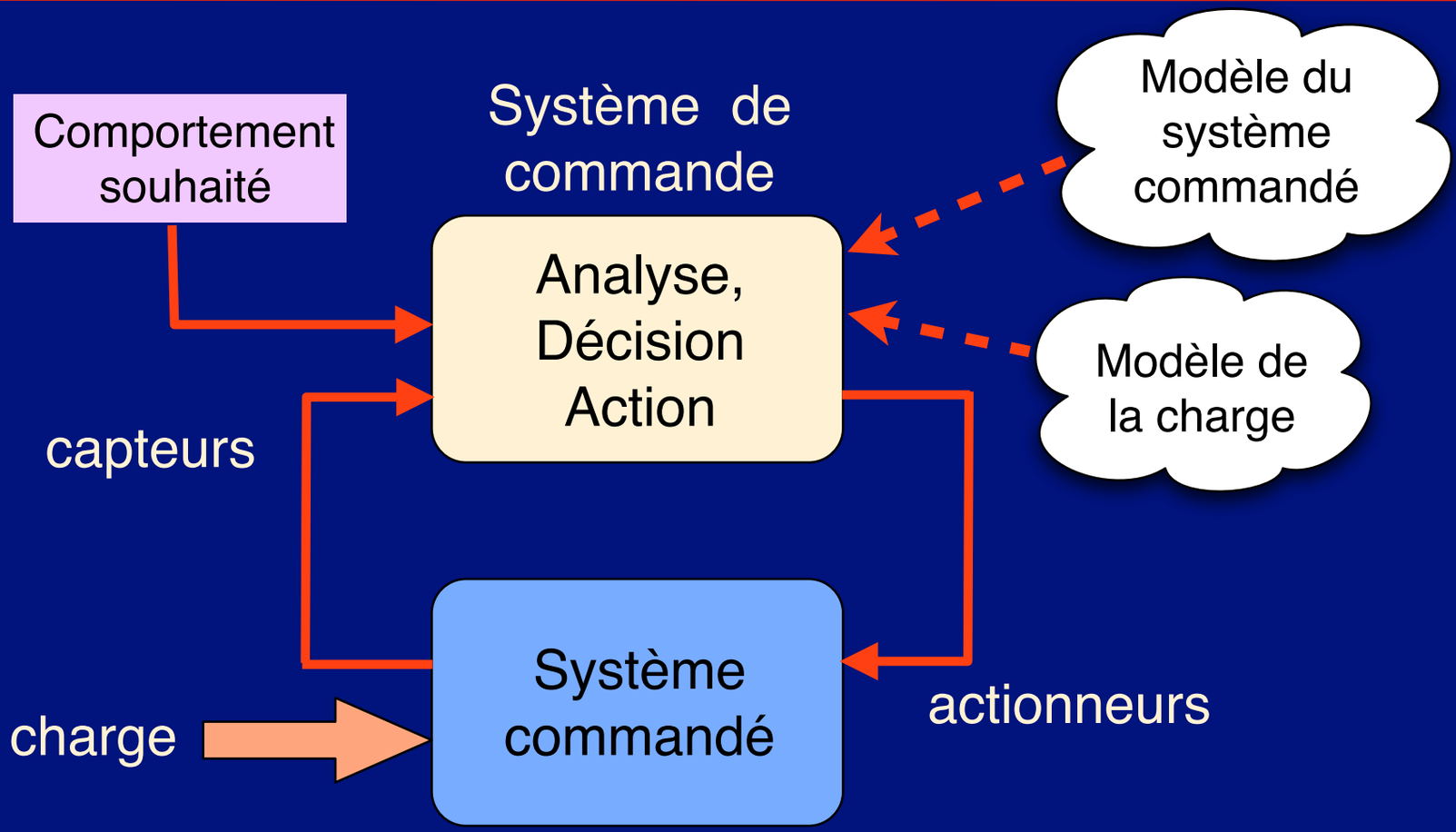
Brève histoire des systèmes auto-adaptables



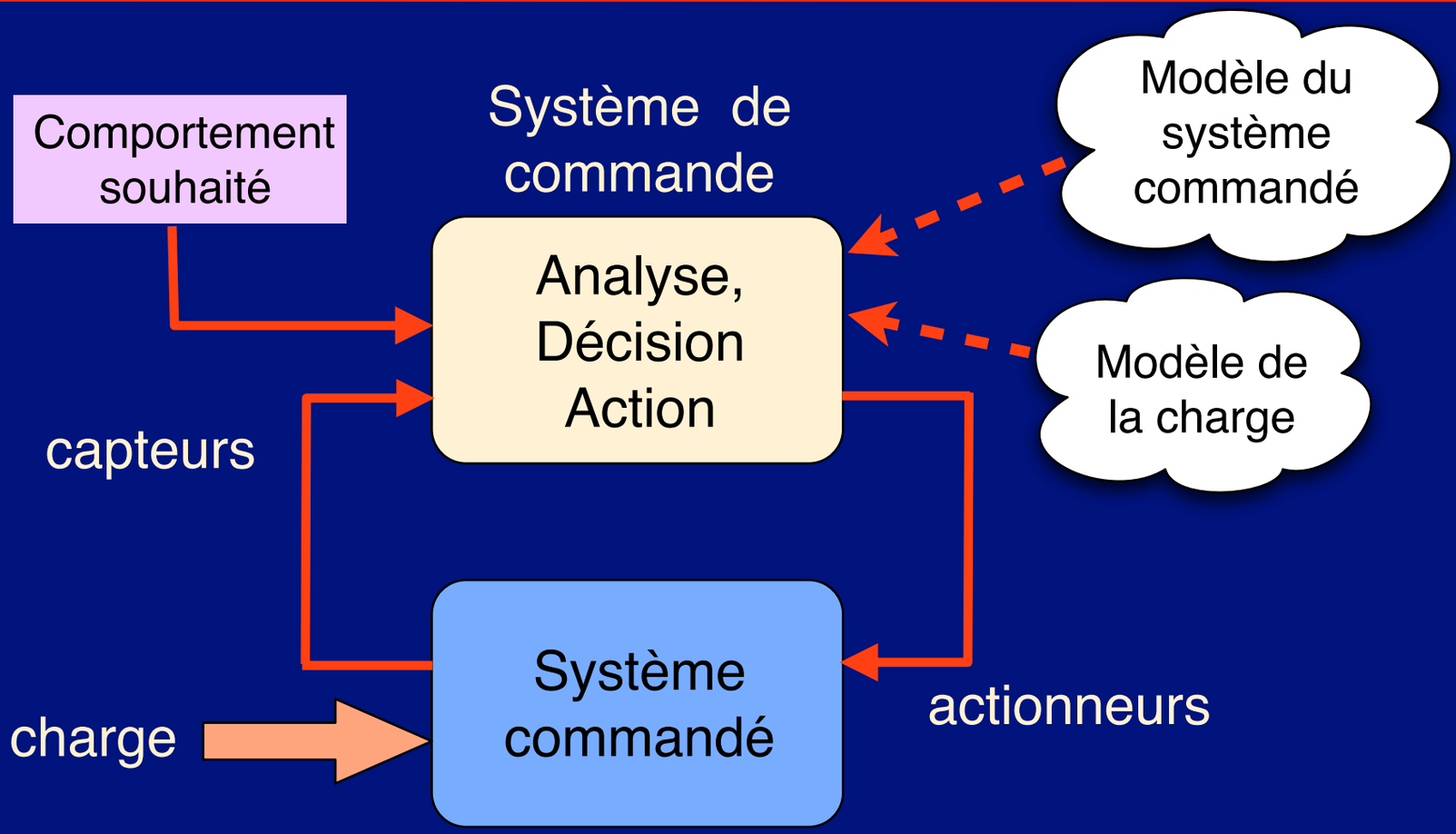
Auto-adaptation par rétroaction



Auto-adaptation par rétroaction



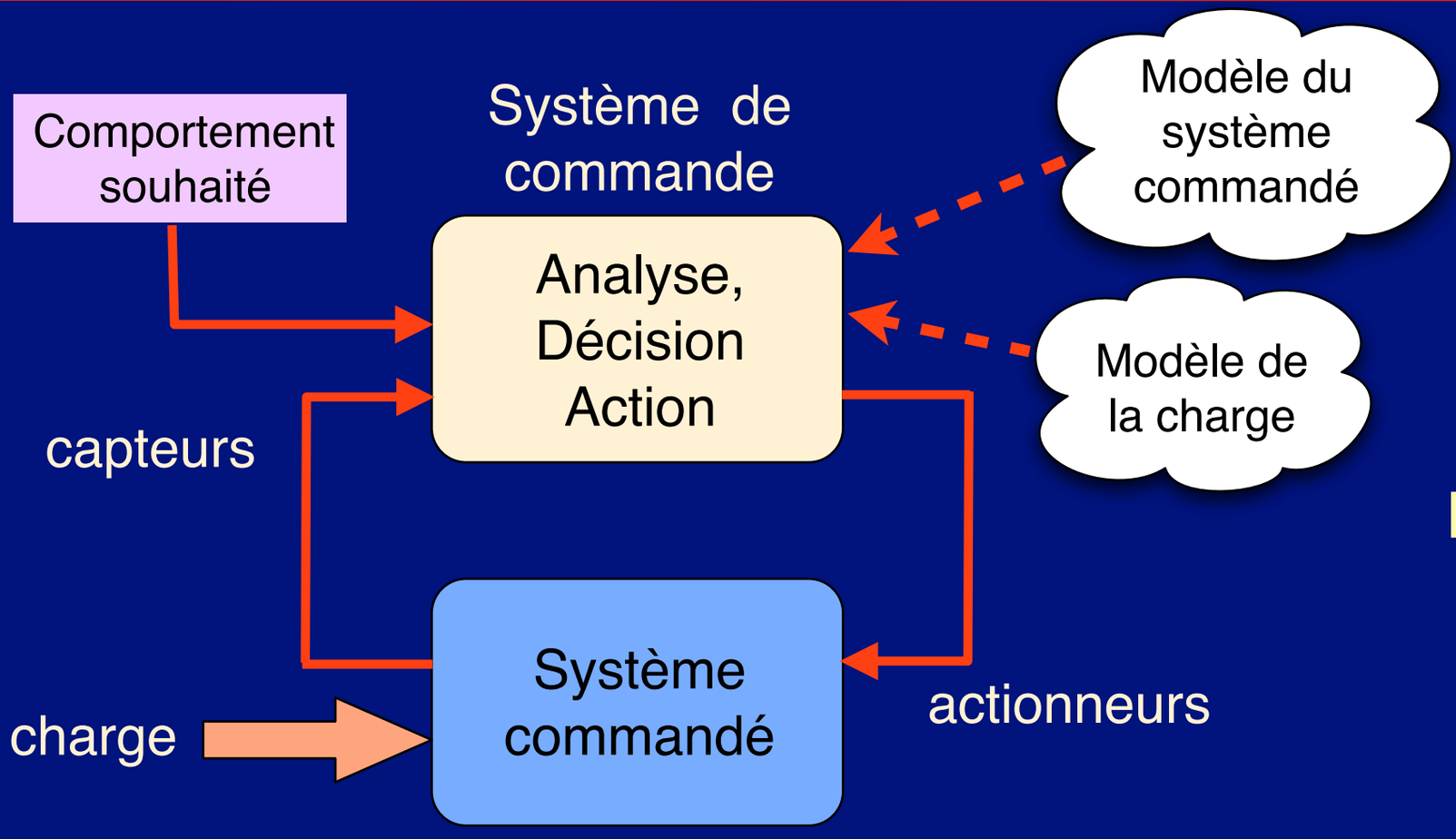
Auto-adaptation par rétroaction



Pour un système informatique, comment définir

- Le comportement souhaité ?
- Les capteurs ?
- Les actionneurs ?
- Les modèles ?
- La stratégie de décision ?

Auto-adaptation par rétroaction



Domaines d'action

- Qualité de service
- Tolérance aux fautes
- Sécurité
- Configuration

Pour un système informatique, comment définir

- Le comportement souhaité ?
- Les capteurs ?
- Les actionneurs ?
- Les modèles ?
- La stratégie de décision ?

Un exemple : auto-adaptation pour la QoS

- Objectifs
  - Contrat (*Service Level Agreement*), décliné en sous-objectifs (SLO)
    - Exemple : 95% des requêtes sont servies en moins de 5 s
  - Réduction du coût

Un exemple : auto-adaptation pour la QoS

- Objectifs
  - Contrat (*Service Level Agreement*), décliné en sous-objectifs (SLO)
    - Exemple : 95% des requêtes sont servies en moins de 5 s
  - Réduction du coût
- Capteurs
  - Problème : les paramètres de QoS sont difficiles à capturer
    - Plus accessibles : l'occupation des ressources

Problème : modélisation de la charge

Un exemple : auto-adaptation pour la QoS

- Objectifs
  - Contrat (*Service Level Agreement*), décliné en sous-objectifs (SLO)
    - Exemple : 95% des requêtes sont servies en moins de 5 s
  - Réduction du coût
- Capteurs
  - Problème : les paramètres de QoS sont difficiles à capturer
    - Plus accessibles : l'occupation des ressources
- Actionneurs
  - Agir sur la demande : le contrôle d'admission
    - Autre idée : applications auto-adaptables
  - Agir sur l'offre : allocation de ressources (physiques, virtuelles)

Problème : modélisation de la charge

Un exemple : auto-adaptation pour la QoS

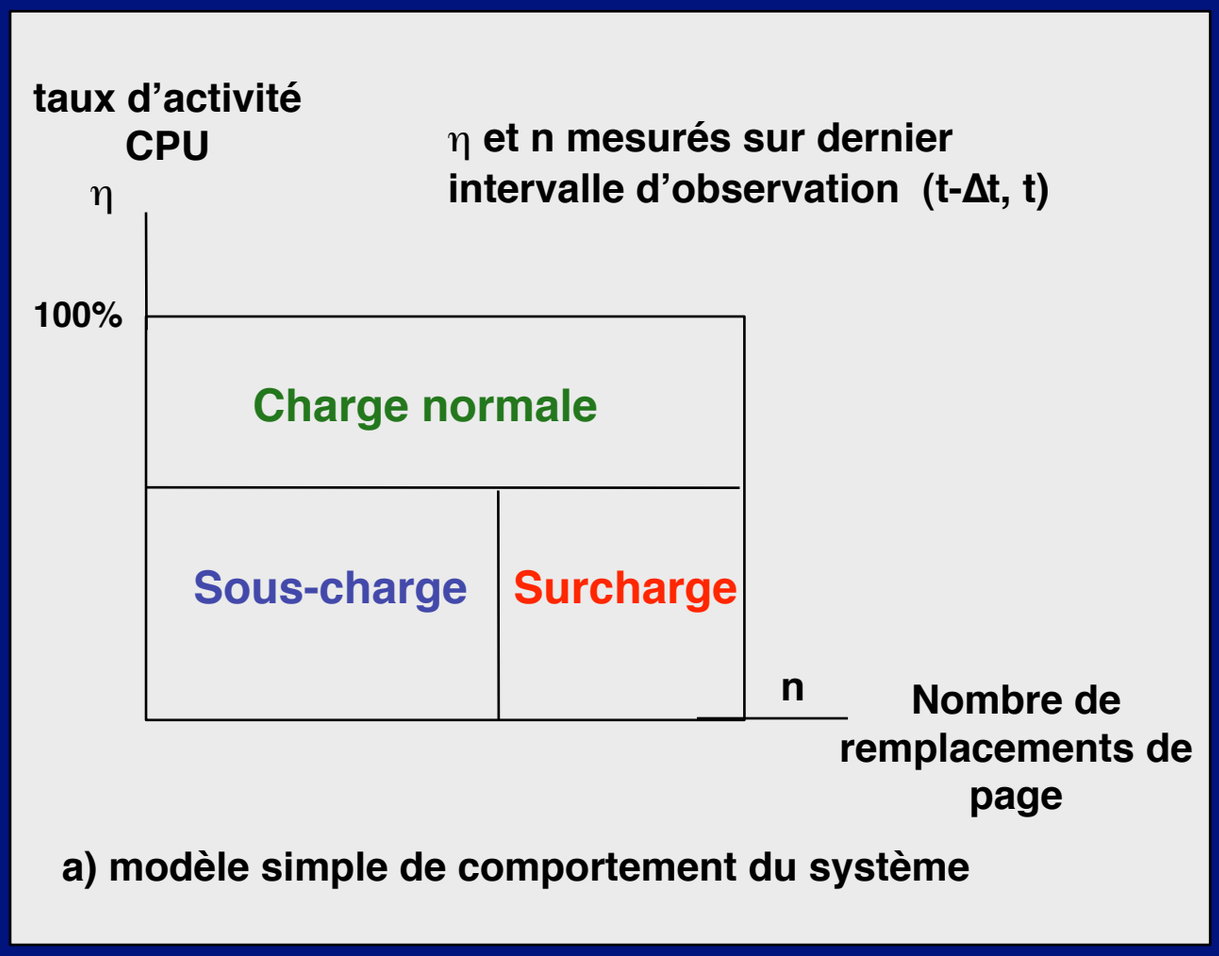
- Objectifs
  - Contrat (*Service Level Agreement*), décliné en sous-objectifs (SLO)
    - Exemple : 95% des requêtes sont servies en moins de 5 s
  - Réduction du coût

Problème : modélisation de la charge

- Capteurs
  - Problème : les paramètres de QoS sont difficiles à capturer
    - Plus accessibles : l'occupation des ressources
- Actionneurs
  - Agir sur la demande : le contrôle d'admission
    - Autre idée : applications auto-adaptables
  - Agir sur l'offre : allocation de ressources (physiques, virtuelles)
- Stratégie
  - Seuils
  - Commande proportionnelle / intégrale

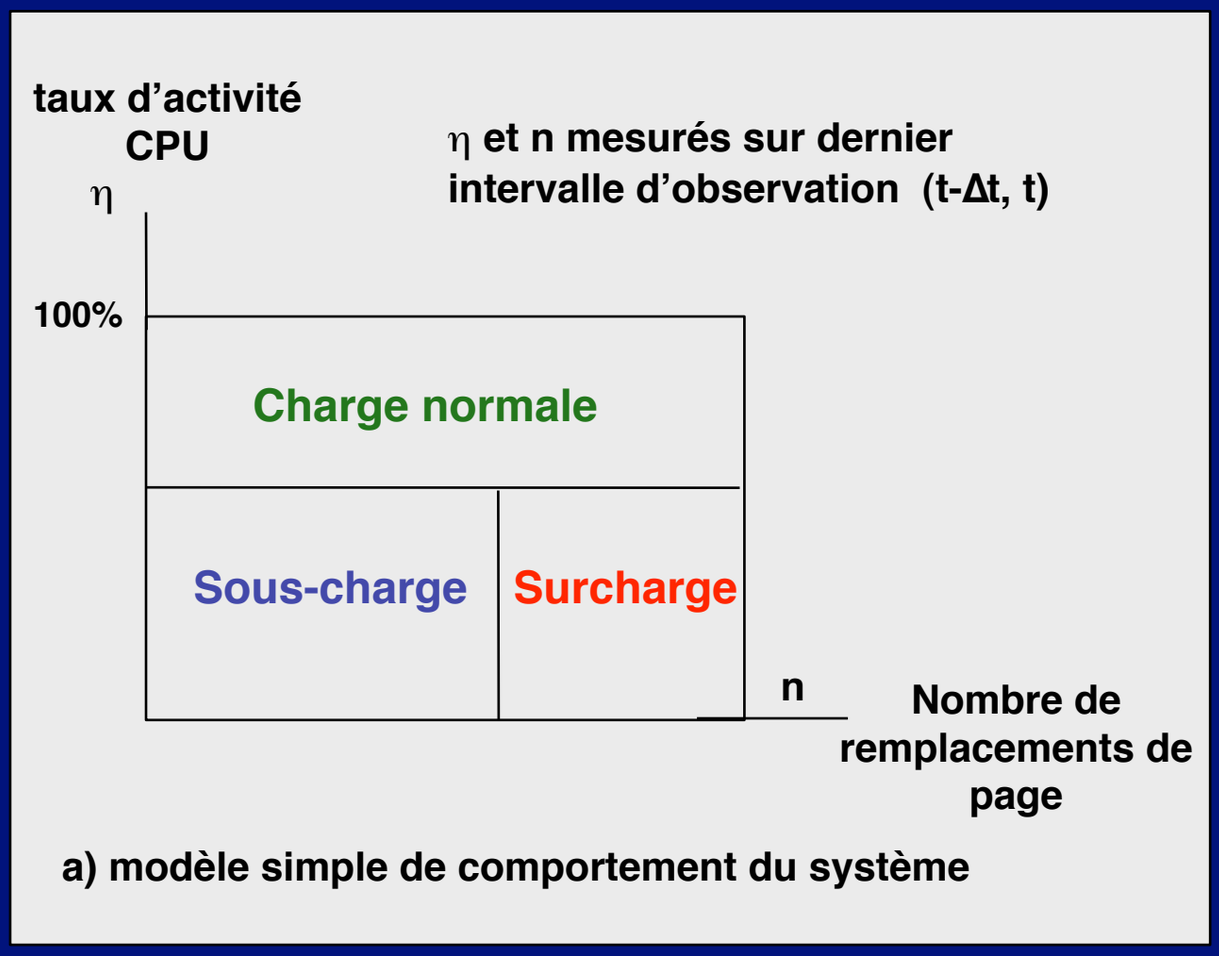
] combinaison

Un exemple ancien avec contrôle d'admission



a) modèle simple de comportement du système

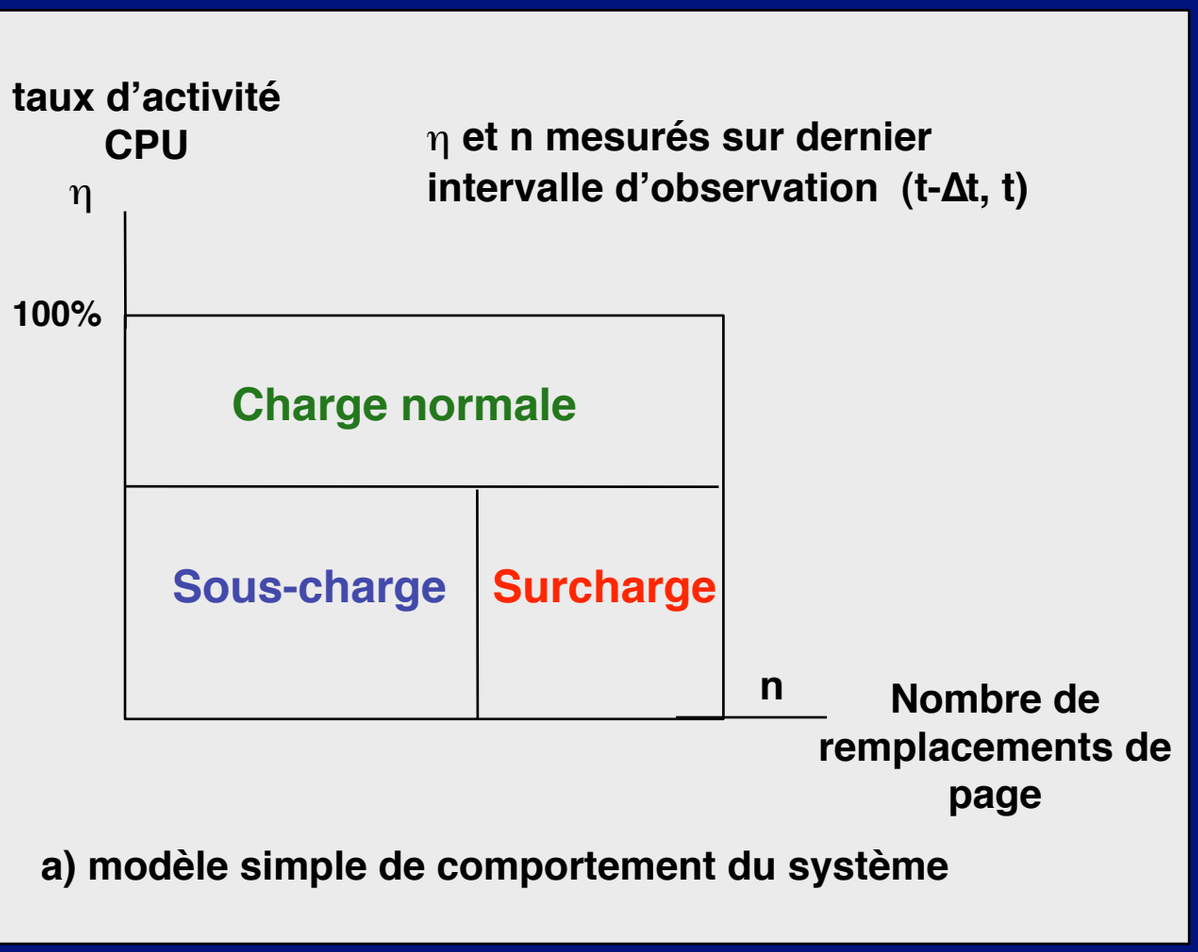
Un exemple ancien avec contrôle d'admission



a) modèle simple de comportement du système

```
every Δt do
    if (overload)
        move one process from ready set to waiting set
    else
        if (underload and (waiting set ≠ ∅))
                admit one waiting process to ready set
```

Un exemple ancien avec contrôle d'admission

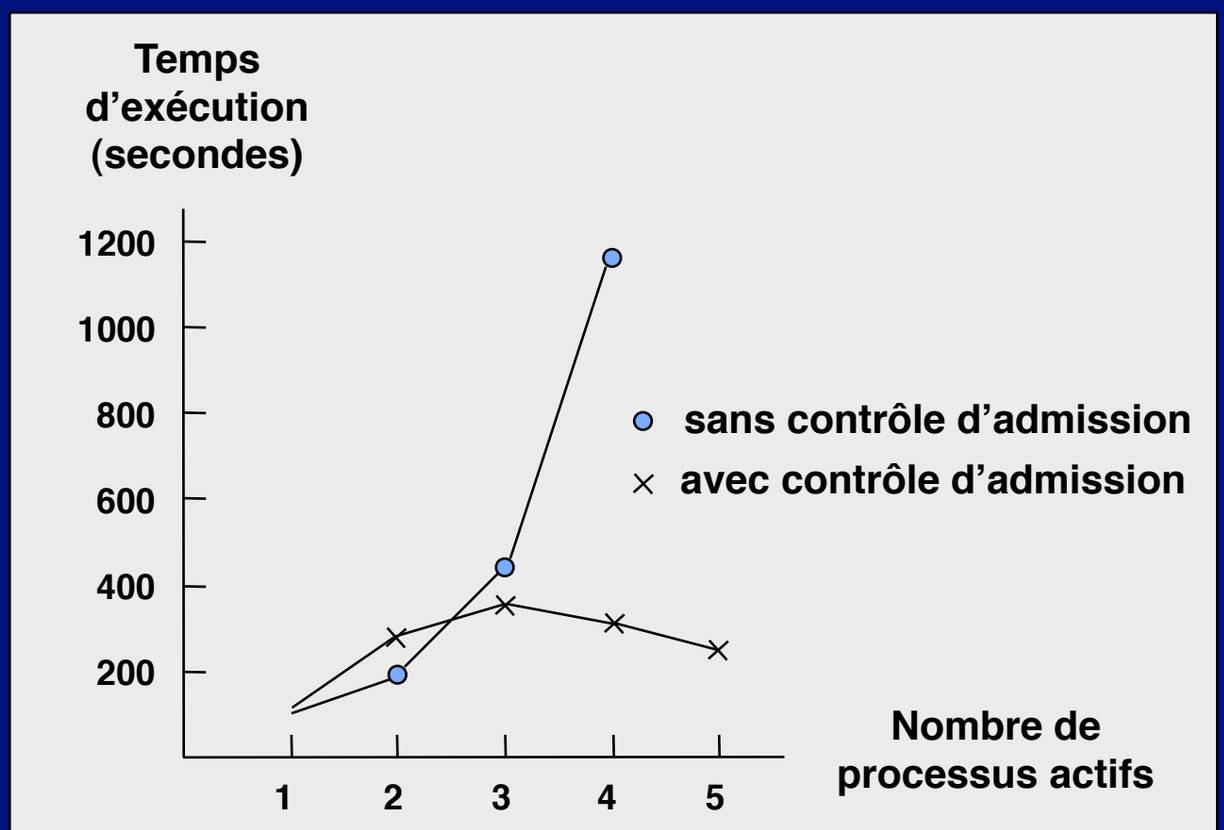


a) modèle simple de comportement du système

```
every Δt do
    if (overload)
        move one process from ready set to waiting set
    else
        if (underload and (waiting set ≠ ∅))
                admit one waiting process to ready set
```

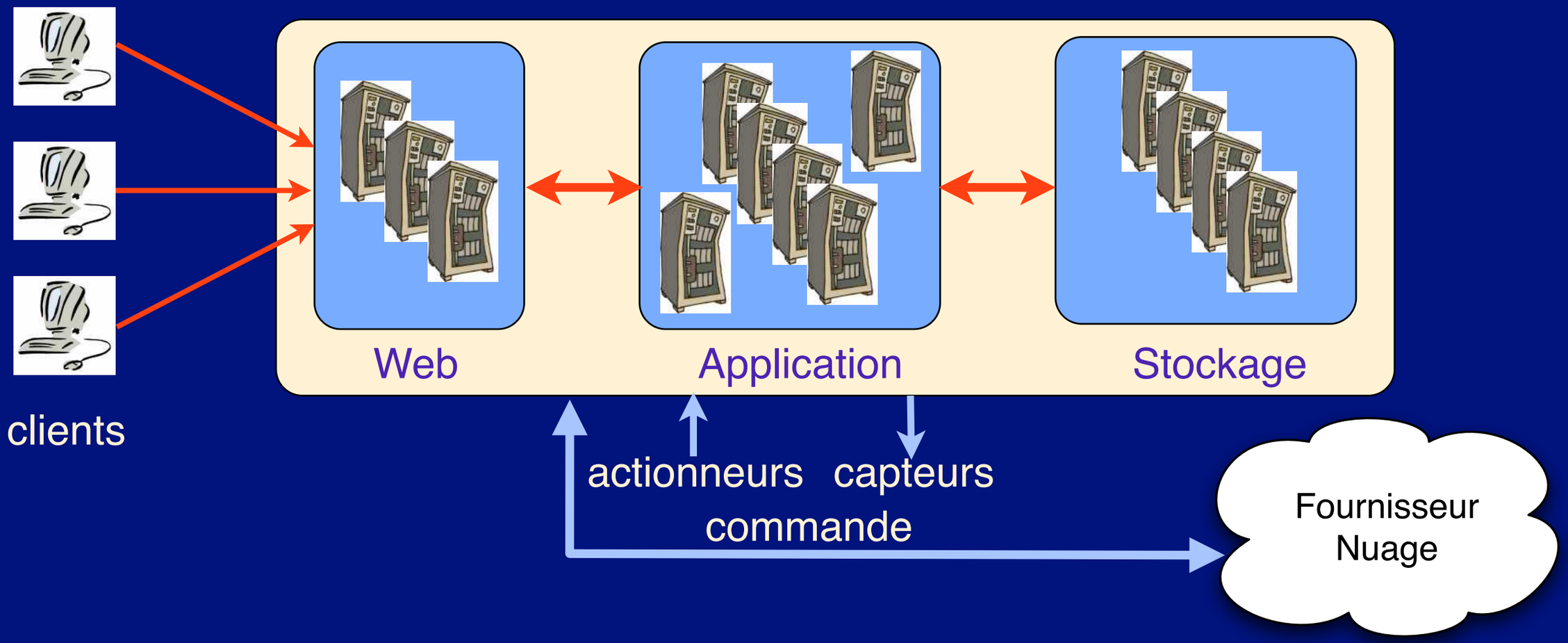
Prévention de l'écroulement : expériences IBM M44/44X (1968)

B. Brawn, F. Gustavson. Program behavior in a paging environment. *Proc. AFIPS FJCC*, pp. 1019-1032 (1968)

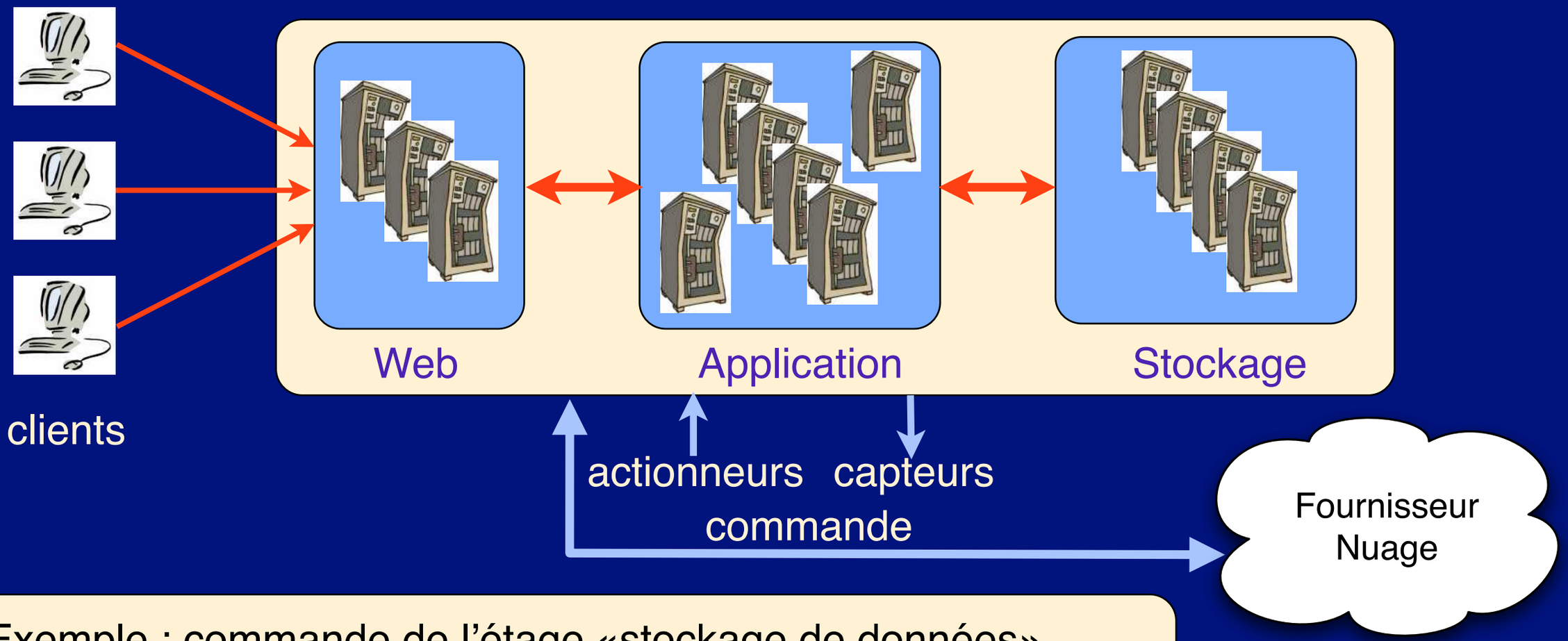


b) effet de la limitation de charge par contrôle d'admission

Auto-adaptation pour la QoS : exemple (1)



Auto-adaptation pour la QoS : exemple (1)



Exemple : commande de l'étage «stockage de données»
- Allocation de serveurs à partir d'un fournisseur de nuages
- Objectif :
  - garantir le temps de réponse en présence de pics de charge
- Expérience sur *Hadoop Distributed File System*

H. C. Lim, S. Babu, J. S. Chase. Automated Control for Elastic Storage, *International Conf. On Autonomic Computing (ICAC)*, June 7-11, 2010

Auto-adaptation pour la QoS : exemple (2)

- Organisation de la commande
  - Pour l'allocation de serveurs
    - actionneur : allouer/libérer des serveurs (interface du fournisseur)
    - capteur : taux d'utilisation du CPU (bien corrélé avec temps de réponse)
    - stratégie : commande intégrale avec seuil (stabilité)

Auto-adaptation pour la QoS : exemple (2)

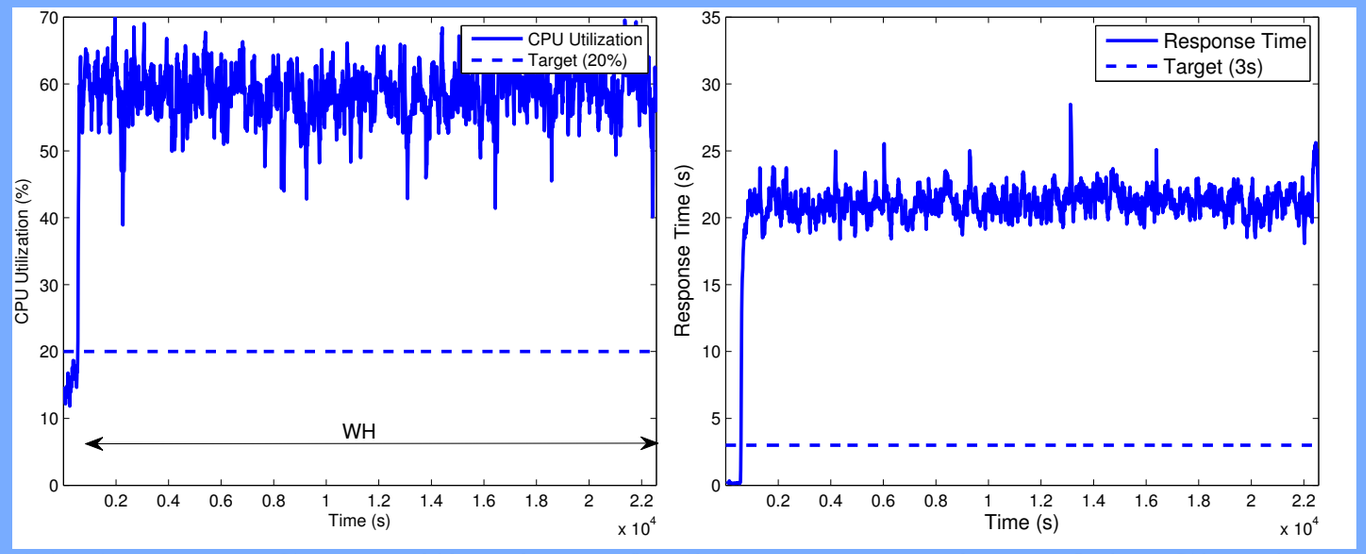
- Organisation de la commande
  - Pour l'allocation de serveurs
    - actionneur : allouer/libérer des serveurs (interface du fournisseur)
    - capteur : taux d'utilisation du CPU (bien corrélé avec temps de réponse)
    - stratégie : commande intégrale avec seuil (stabilité)
  - Pour la reconfiguration de l'étage stockage (répartition de données)
    - actionneur : fraction de bande passante allouée à la reconfiguration (qui interfère avec le traitement)
    - capteur : temps nécessaire en fonction de la taille + impact de la reconfiguration sur le temps de réponse

Auto-adaptation pour la QoS : exemple (2)

- Organisation de la commande
  - Pour l'allocation de serveurs
    - actionneur : allouer/libérer des serveurs (interface du fournisseur)
    - capteur : taux d'utilisation du CPU (bien corrélé avec temps de réponse)
    - stratégie : commande intégrale avec seuil (stabilité)
  - Pour la reconfiguration de l'étage stockage (répartition de données)
    - actionneur : fraction de bande passante allouée à la reconfiguration (qui interfère avec le traitement)
    - capteur : temps nécessaire en fonction de la taille + impact de la reconfiguration sur le temps de réponse
  - Coordination entre les deux commandes ci-dessus
    - but : eviter sur- ou sous-allocations ; éviter les oscillations
    - moyen : machine à états assurant l'alternance entre les deux commandes avec temporisation

Auto-adaptation pour la QoS : exemple (3)

- Résultats
  - Très bonne réactivité à un pic de charge
  - (a posteriori) Bonne corrélation entre temps de réponse et utilisation CPU



(a) Average CPU utilization of the HDFS datanodes with static provisioning

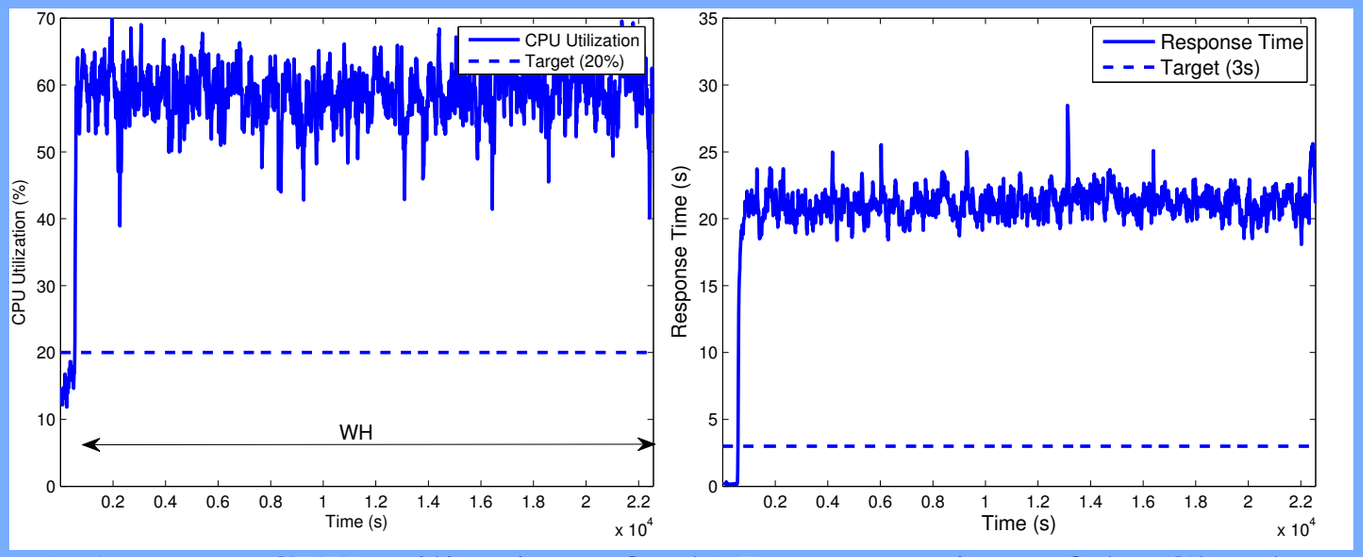
(b) Response time of the Cloud-stone application with static provisioning

Auto-adaptation pour la QoS : exemple (3)

- Résultats
  - Très bonne réactivité à un pic de charge
  - (a posteriori) Bonne corrélation entre temps de réponse et utilisation CPU

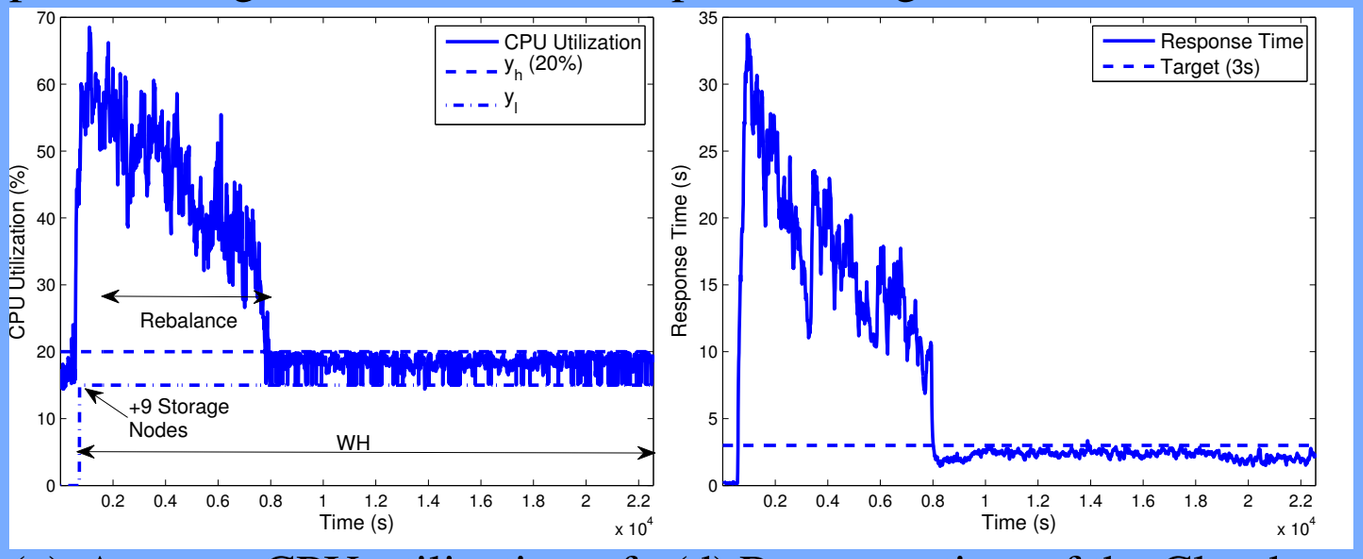
Figure 6 of:

H. C. Lim, S. Babu, J. S. Chase. Automated Control for Elastic Storage, *International Conf. On Autonomic Computing (ICAC)*, June 7-11, 2010



(a) Average CPU utilization of the HDFS datanodes with static provisioning

(b) Response time of the Cloud-stone application with static provisioning



(c) Average CPU utilization of the HDFS datanodes with dynamic provisioning

(d) Response time of the Cloud-stone application with dynamic provisioning

Avancées et défis pour l'auto-adaptation

- Avancées
  - Interaction fructueuse avec l'automatique
    - domaine continu (boucle de commande)
    - domaine discret (synthèse de contrôleurs)
    - résultats pour QoS
  - Composants et architectures réflexifs

Avancées et défis pour l'auto-adaptation

- Avancées
  - Interaction fructueuse avec l'automatique
    - domaine continu (boucle de commande)
    - domaine discret (synthèse de contrôleurs)
    - résultats pour QoS
  - Composants et architectures réflexifs
- Défis
  - Approches multiniveaux (piloté vs auto-organisé)
  - Expression des objectifs
    - objectifs multicritères (performances, énergie, disponibilité, …)
    - prise en compte de situations non prévues
  - Modélisation, vérification, garanties
    - liaison continu-discret, prise en compte du temps
  - Sécurité

Remarques finales

- Sur les paradigmes de l'architecture
  - Permanence des concepts, raffinement (lent…) dans leur mise en œuvre
  - Nouveaux paradigmes
    - mobilité, autonomie, …

Remarques finales

- Sur les paradigmes de l'architecture
  - Permanence des concepts, raffinement (lent…) dans leur mise en œuvre
  - Nouveaux paradigmes
    - mobilité, autonomie, …

puissance de l'abstraction

puissance (et développement) des modèles

Remarques finales

- Sur les paradigmes de l'architecture
  - Permanence des concepts, raffinement (lent…) dans leur mise en œuvre
  - Nouveaux paradigmes
    - mobilité, autonomie, …
- Quelques défis pour l'avenir
  - Conceptuels
    - modéles formels pour l'architecture
    - validité de la construction
    - modélisation de la sécurité
    - systèmes hybrides
  - Pratiques
    - description déclarative des environnements et contraintes
    - génération automatique de systèmes spécialisés
    - administration et qualité de service des très grands systèmes

puissance de l'abstraction

puissance (et développement) des modèles