Deuxième Conférence Française sur les Systèmes d'Exploitation (CFSE)
Septième Symposium sur les Architectures Nouvelles de Machines (SYMPA)
Treizièmes Rencontres Francophones du Parallélisme (RENPAR)

# Architecture des systèmes

## passé et avenir

**Sacha Krakowiak**
Université Joseph Fourier
Laboratoire Sirac (INPG - INRIA - UJF)

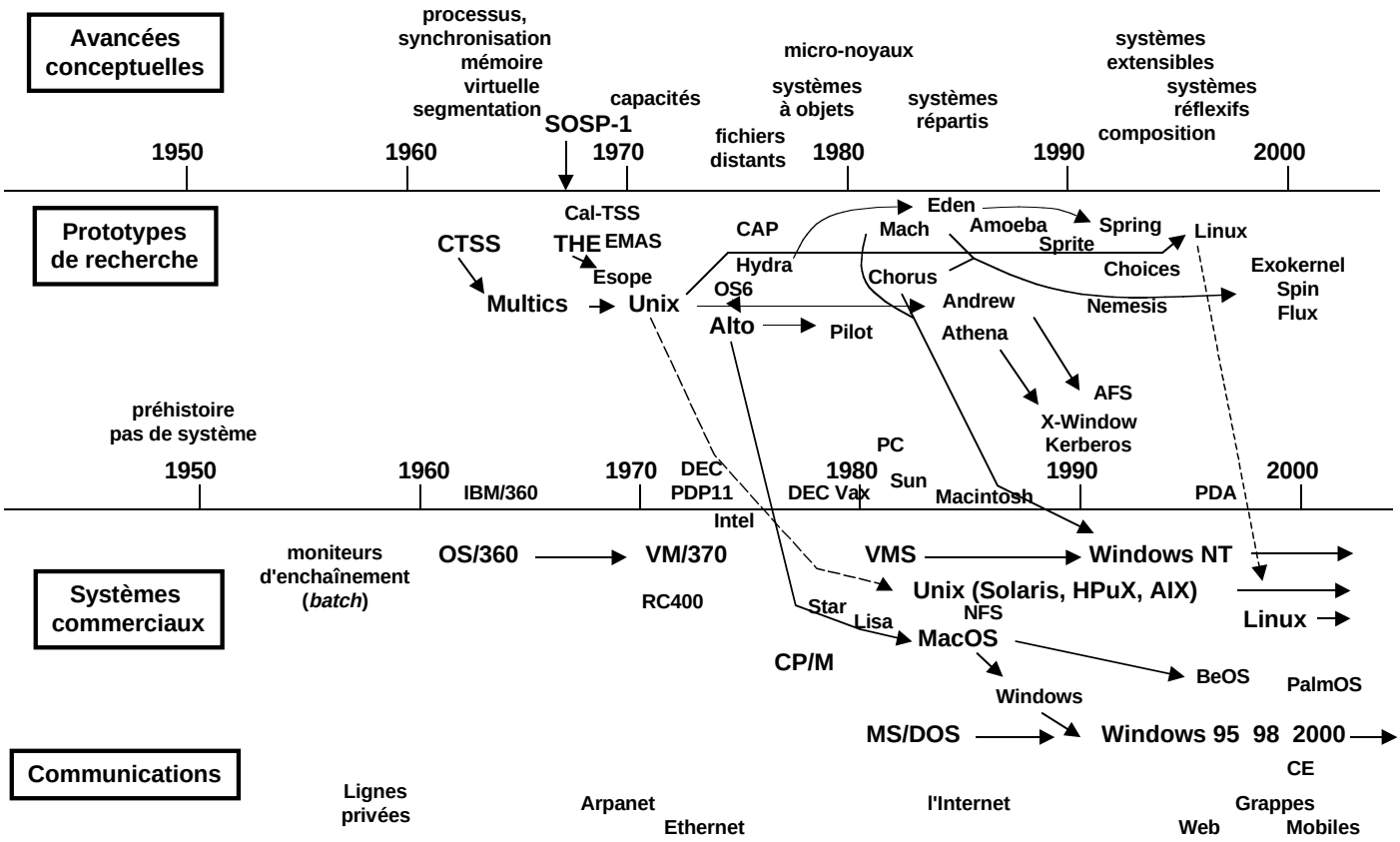
http://sirac.imag.fr/~krakowia

## 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
    - Une vision globale
    - Une structure
    - Des principes d'organisation
- **"Les architectures" : des modes de structuration**
  - monolithiques, machine virtuelle, micro-noyaux, noyaux extensibles, modules, objets, composants, etc.

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

# Un peu d'histoire




# À quoi servent les chercheurs en systèmes ?

## ■ Le chercheur, "marchand d'abstraction" ?

***We operating systems researchers are abstraction merchants***

**Andrew Black**, SOSP, 1987 (?)

***Exterminate all operating systems abstractions***

**Dawson Engler & Frans Kaashoek**, HotOS V, 1995

## ■ Plutôt, créateur de 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**

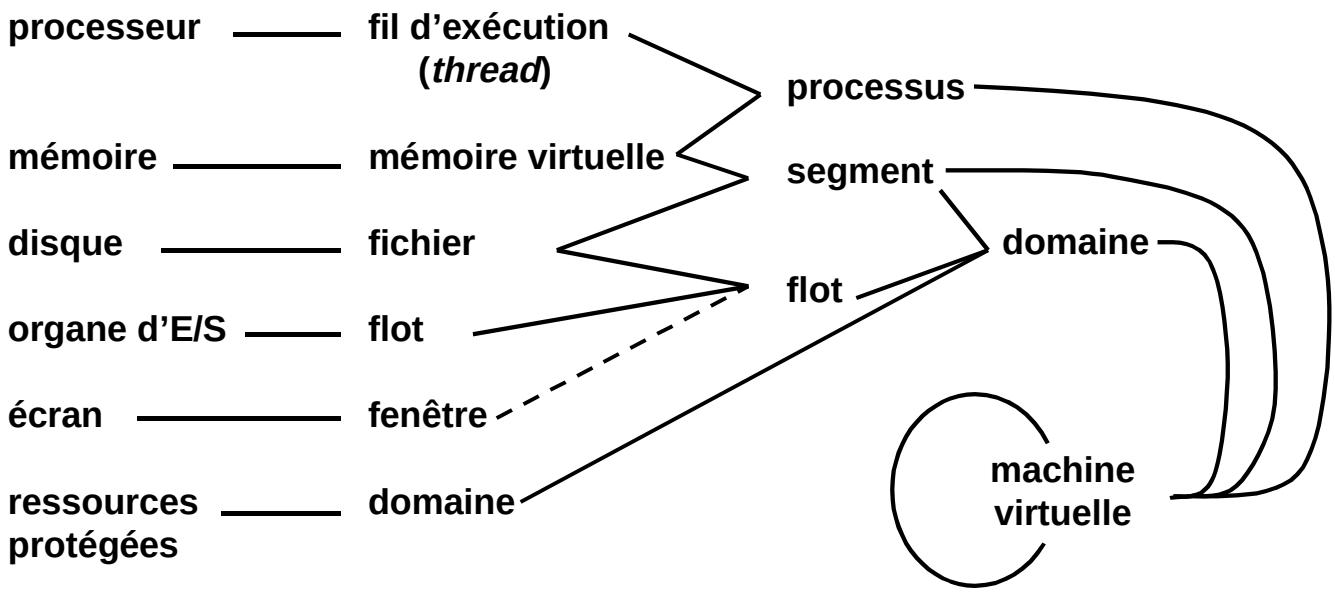
# Quelques paradigmes de l'architecture des systèmes

- **Virtualisation**
  - avatars de la notion d'abstraction
  - interfaces et boîtes noires
- **Composition et décomposition**
  - interposition et interception
  - flots et filtres
- **Liaison**
  - le moment et la manière
  - substitution, indirection, génération
- **Réification**
  - trop d'abstraction nuit
  - réflexion et méta-objets




# Virtualisation

**Virtualiser une ressource = en fournir une interface abstraite, cacher la réalisation, organiser l'allocation des éléments physiques**

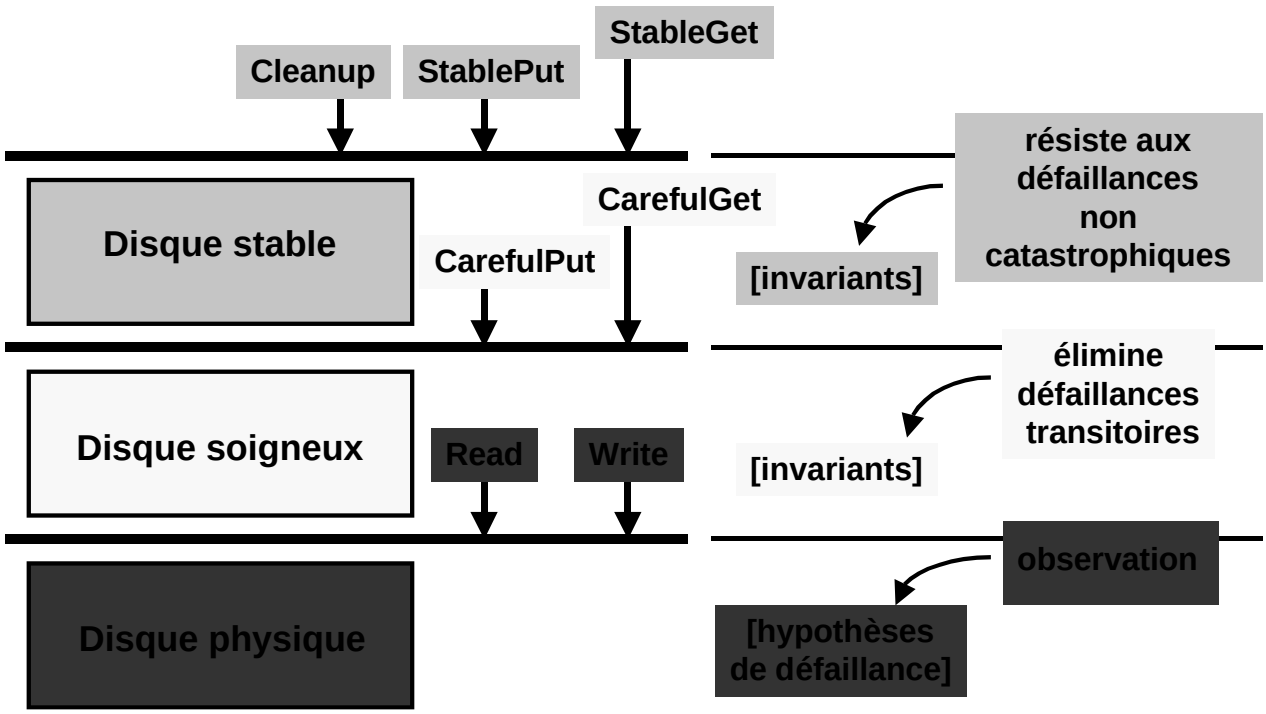


# Applications de la virtualisation

- **Hiérarchie de ressources (virtuelles)**
  - Une ressource (ou classe de ressources) est réalisée par une "machine" qui en fournit une vue abstraite
  - Chaque machine utilise des machines de niveau inférieur (plus proches du niveau physique)
  - Exemples : THE, disque virtuel, …
- **Machines virtuelles**
  - Virtualisation de l'ensemble d'une machine
    - La machine virtuelle est différente de la machine physique (simulateur) : DISCO, JVM, …
    - La machine virtuelle est identique à la machine physique (hyperviseur) : CP-67, VM-370, …
  - Machines virtuelles récursives
    - Hiérarchie de machines virtuelles : VM, Fluke

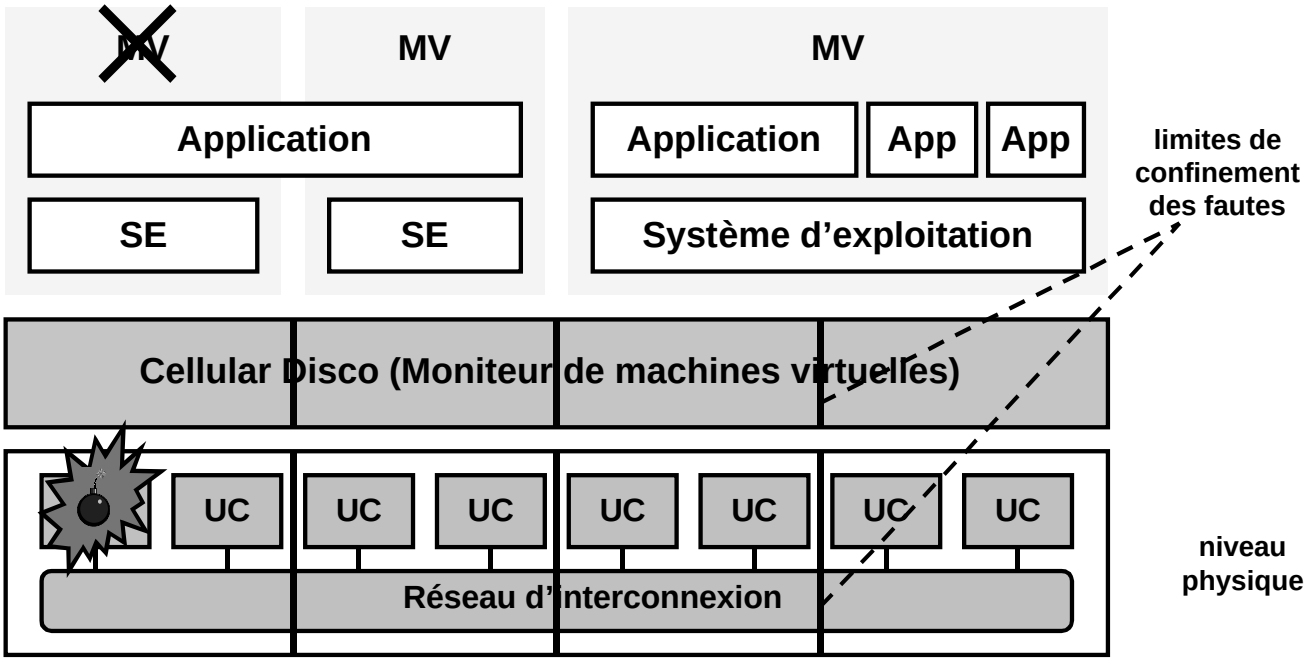



# Virtualisation, exemple 1 : la mémoire stable




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

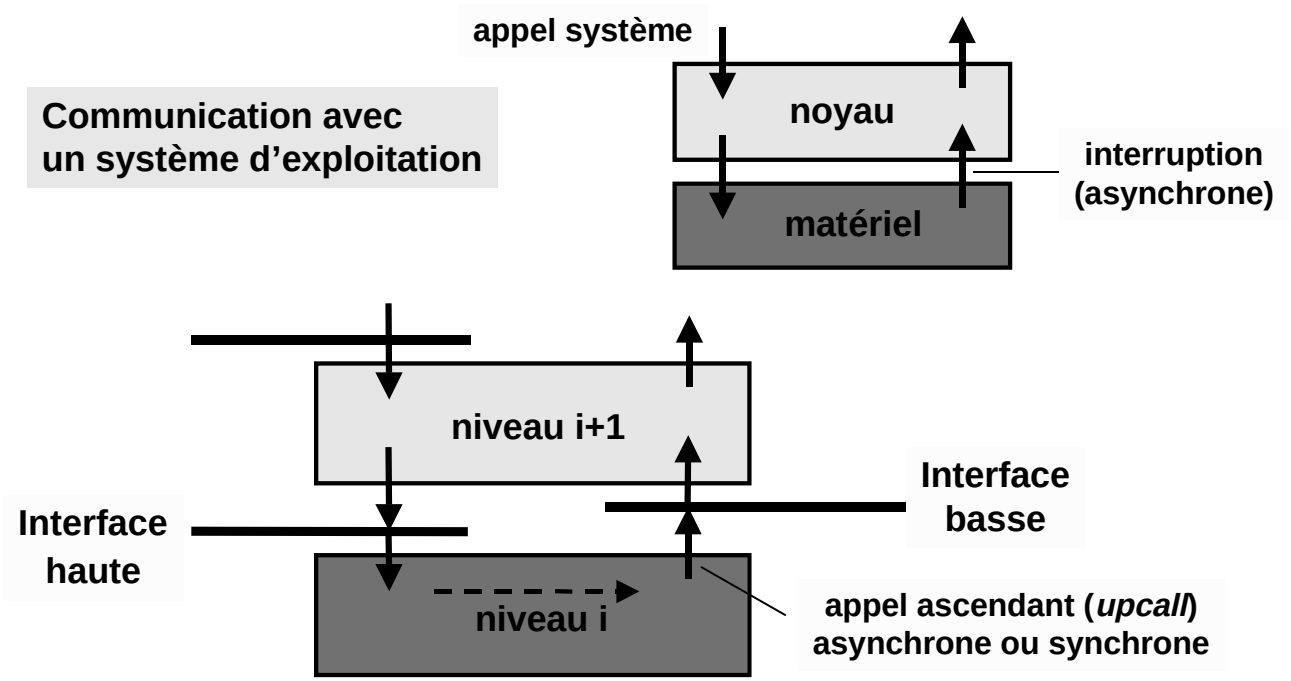
# Virtualisation, exemple 2 : grappes virtuelles




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



# Propagation des appels dans un système hiérarchique




D. D. Clark. The Structuring of Systems Using Upcalls, *Proc. 10th ACM Symposium on Operating Systems Principles*, Dec. 1985

## Conclusion sur la virtualisation

- Un outil de conception
  - définition des interfaces
- Un point d'appui pour le raisonnement
  - invariants et preuves
- Un guide pour la réalisation
  - implémentation des interfaces
- Un environnement de recherche
  - outil élaboré de simulation
  - exploration de choix de conception
- Une aide à l'évolution
- Deux problèmes potentiels
  - les performances
  - l'excès d'abstraction (perte de contrôle des ressources)



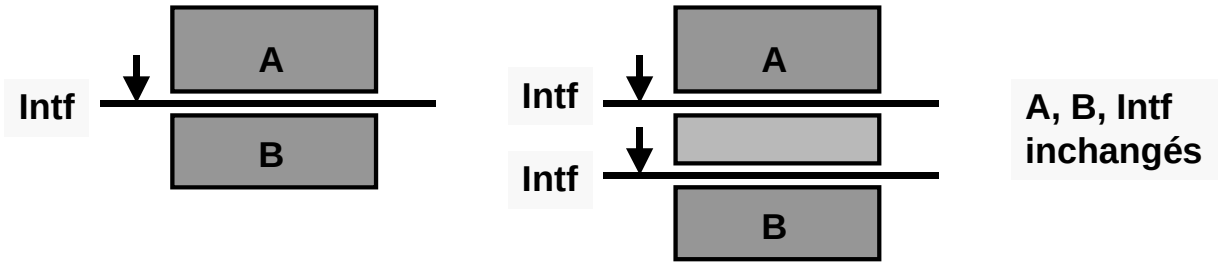
## Composition et décomposition

- La composition modulaire, un objectif ancien …
  - Construction à partir d'éléments (modules, composants, objets, …)
    - accès par interfaces spécifiées (encapsulation)
    - indépendance interface-réalisation
  - Mise en évidence de la structure d'ensemble
  - Facilité d'évolution
  - Composants standard, familles de produits
  - Aide au déploiement et à l'administration
  - Description formelle, preuves
- Quelques paradigmes de la composition
  - Composition hiérarchique (déjà vu, cf. machines virtuelles)
  - Interception et interposition
  - Composition dirigée par le flot de données

# Interception et interposition (1)

## ■ Exploitation de la notion d'interface

- ◆ En interposant une couche supplémentaire



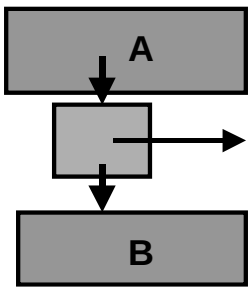

- ◆ La couche de logiciel ajoutée intercepte et réinterprète les appels
- ◆ Grâce à l'indépendance entre interface et réalisation, l'interception est "transparente" pour A et pour B
- ◆ S'applique aussi à la communication asynchrone (messages ou événements)



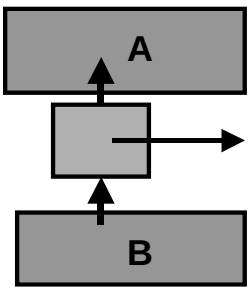
# Interception et interposition (2)

## ■ Quelques cas typiques d'interposition

détournement (conditionnel) d'une opération

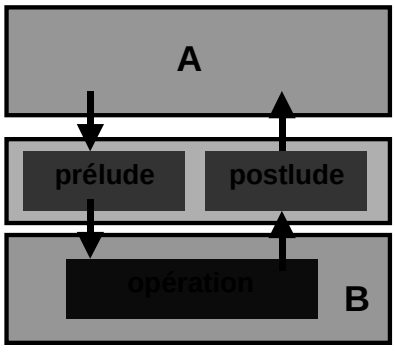



idem pour les appels ascendants




remplace <opération> par :

prélude ; <opération> ; postlude

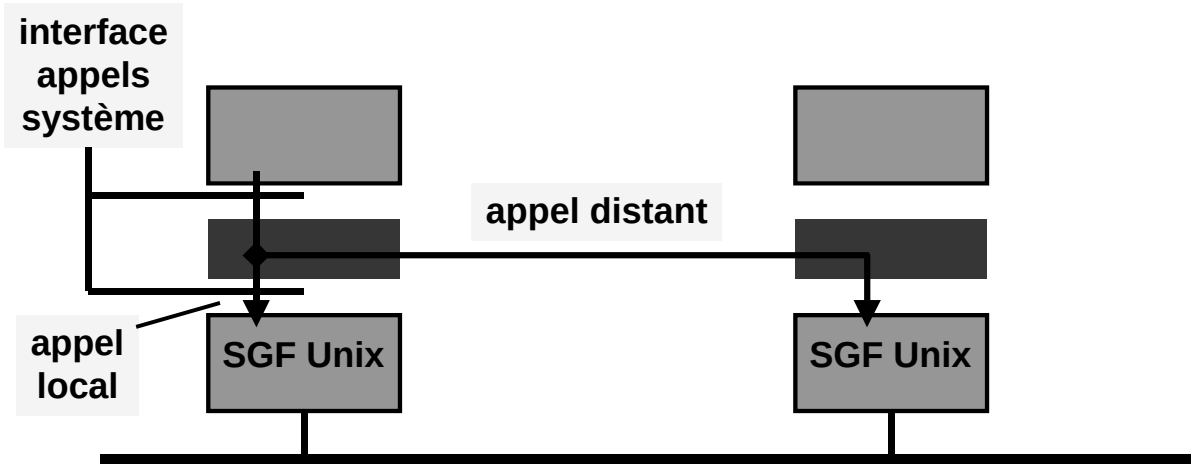



Autres usages :
- traces
- statistiques

Généralisation :
- enveloppes
- conteneurs

# Exemple d'interposition : *Newcastle Connection*

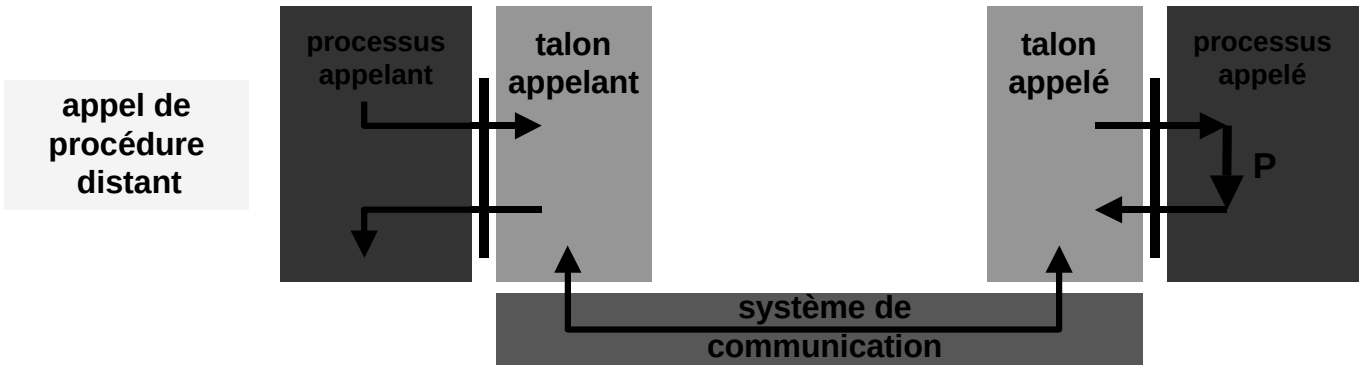
- **Le premier Unix réparti**
  - **unifie les SGF sous une "super-racine" commune**




**D. R. Brownbridge, L. F. Marshall, B. Randell. The Newcastle Connection - or "Unixes of the World, Unite !", *Software - Practice and Experience*, 12(12), December 1982, pp. 1147-1162**



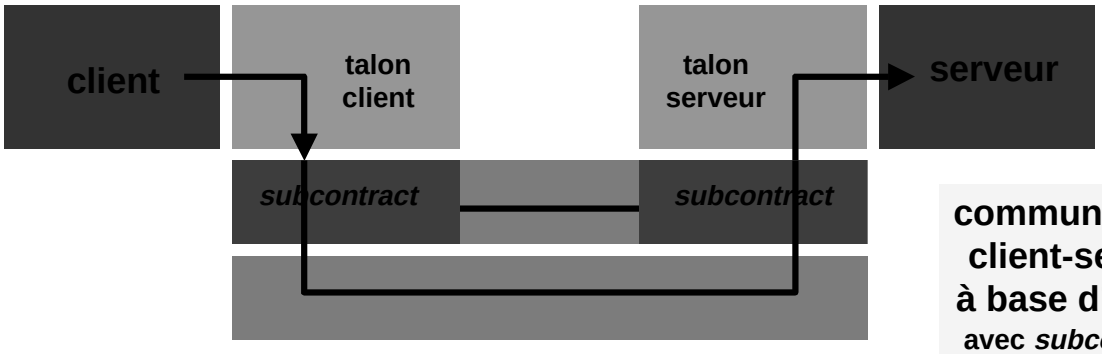
# Exemple d'interposition : RPC




- **Talons (*stubs*) : modules d'interposition, qui réalisent**
  - **l'emballage / déballage des paramètres**
  - **la communication**
  - **le traitement des erreurs**
- **Chacun assure un rôle de "réprésentant" distant**
- **L'interface est (presque) préservée**

**A. Birrell, B. Nelson. Implementing Remote Procedure Calls, *ACM Trans. on Computer Systems*, 2(1), pp. 39-59, Feb. 1984**

# Exemple d'interposition : *subcontracts* (Spring)




**communication client-serveur à base d'objets**
avec *subcontracts*

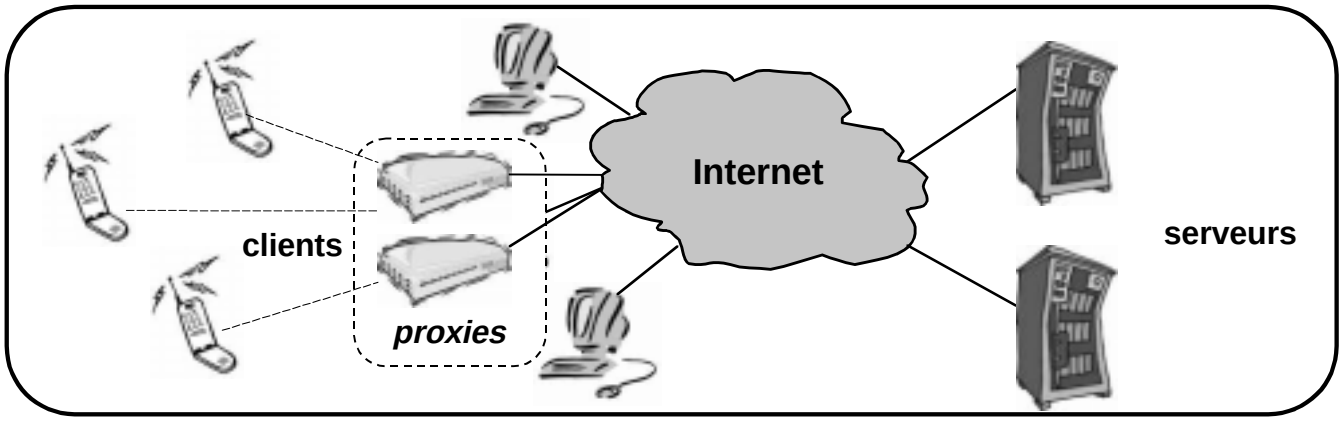
- Les *subcontracts* permettent de spécialiser le comportement du système
  - accès à un groupe d'objets
  - gestion de caches
  - mode déconnecté

G. Hamilton, M. Powell, J. Mitchell. Subcontract: A Flexible Base for Distributed Programming, *Proc. 14th ACM Symposium on Operating Systems Principles*, Dec. 1993



# Interposition pour l'adaptation

**Objectif : porter une application répartie complexe sur des clients "légers" (PDAs)**



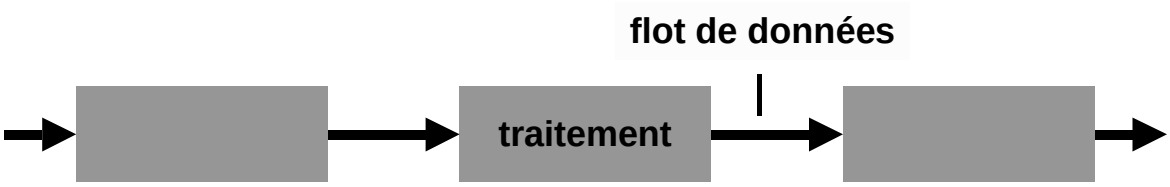

**Méthode : reporter les fonctions coûteuses dans un ensemble de mandataires (*proxies*)**

- Fonctions des *proxies* :
  filtrage et compression (avec perte) des images, filtrage de texte (HTML)
  agrégation de réponses aux requêtes, gestion de caches
- Programmation indépendante du serveur, adaptée aux caractéristiques des clients

A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer. Adapting to Network and Client Variation Using Infrastructural Proxies: Lessons and Perspectives, *IEEE Personal Communications,* August 1998

# Composition dirigée par le flot de données

- **Schéma de base : flots et filtres**

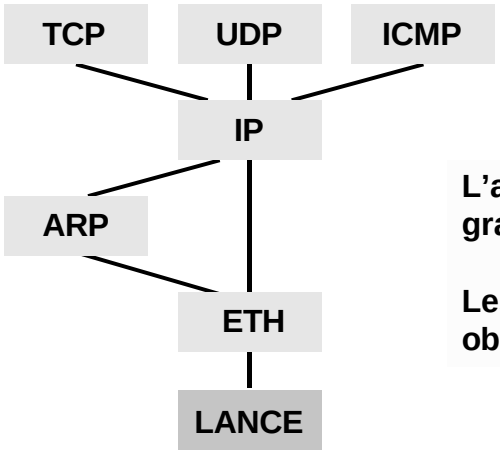



- **Variantes**
  - 1 processus par flot (ou par message)
    - les messages dans *x-kernel*
  - 1 processus par module de traitement
    - les *pipes* d'Unix, les *streams* d'OS6 et Pilot



# Application 1 : construction de protocoles

- La composition est dirigée par les flux de messages
  - File, arbre, ou graphe de filtres
- Exemple : *x*-kernel, un canevas de composition pour les protocoles

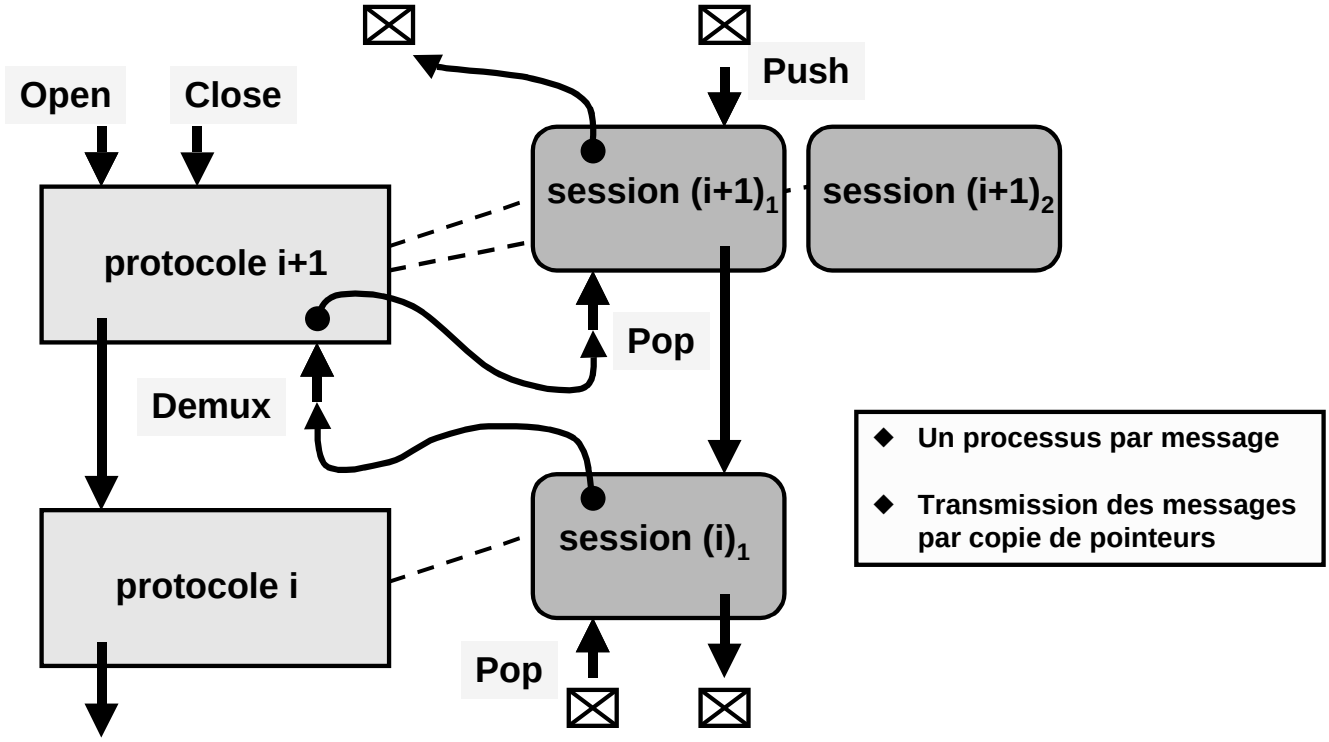



L'architecture du système suit le graphe de protocoles

Les interfaces entre modules obéissent à des règles uniformes

N. Hutchinson and L. Peterson. The *x*-kernel: An Architecture for Implementing Network Protocols, *IEEE Transactions on Software Engineering*, 17(1), pp. 64-76, Jan. 1991

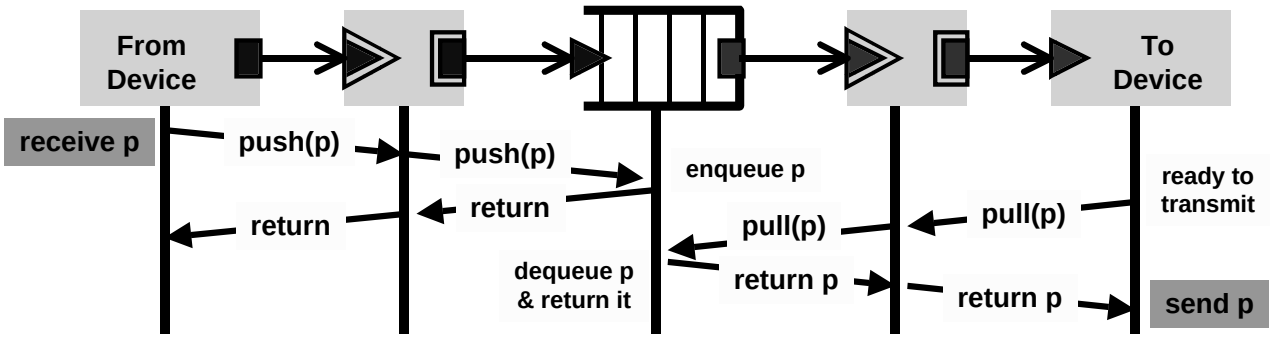
# Communication dans *x*-kernel





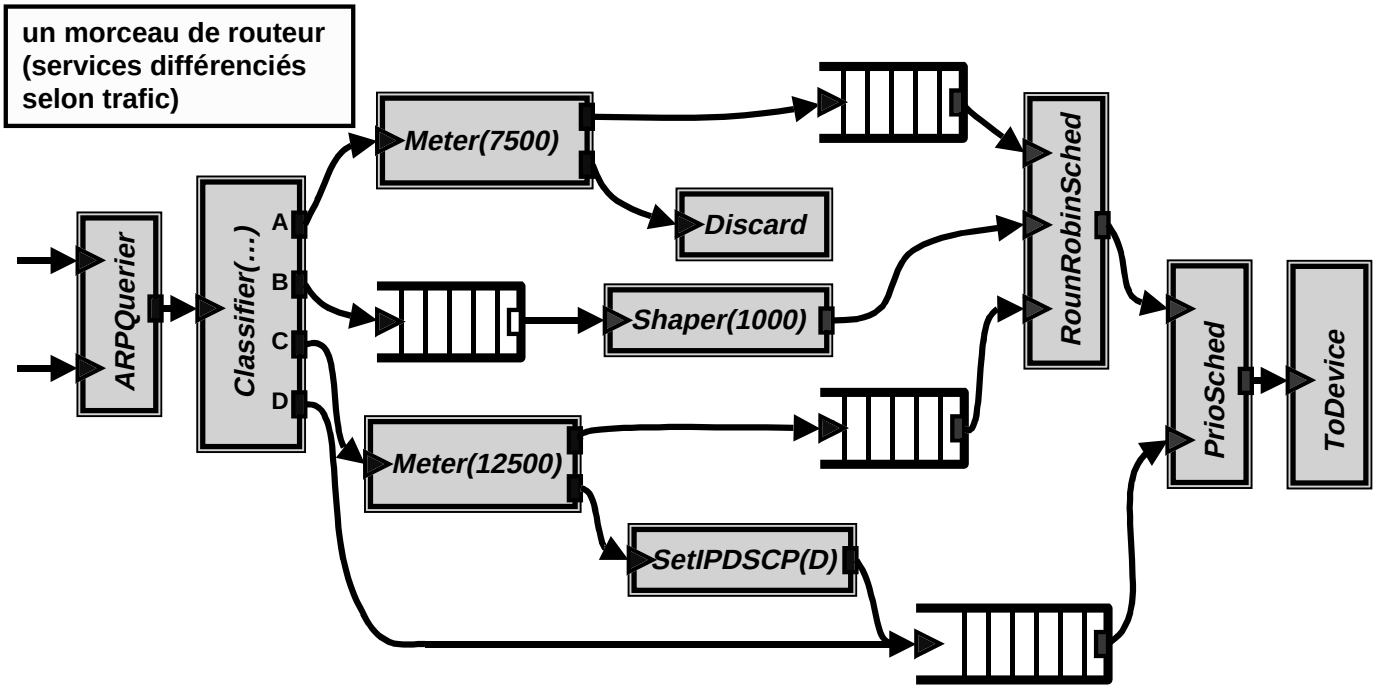

# Application 2 : assemblage de composants

- **Exemple : Click, canevas pour la composition de routeurs**
  - **généralise le principe de *x*-kernel**
  - **graphe quelconque (y compris avec circuits)**
  - **règles pour la composition correcte**
  - **mini-langage de description globale Ł preuves de propriétés**




E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click Modular Router, *ACM Trans. on Computer Systems*, 18(3), Aug. 2000

# Assemblage de composants Click




autres travaux récents : A. Reid, M. Flatt, L. Stoller, J. Lepreau, and E. Eide. Knit: Component Composition for Systems Software. *Proc. of the 4th Operating Systems Design and Implementation Conference (OSDI)*, Oct. 2000.



# Conclusion sur la composition

- Une opération apparemment simple …
- … mais dont la maîtrise nous échappe encore
  - pas (ou trop ?) de support(s) formel(s)
  - pas (ou trop ?) de mécanisme(s) unificateur(s)
- Quelques voies prometteuses
  - méthodes et outils spécifiques à un domaine d'application
  - mise en œuvre spécifique à un langage ou à un support de communication
  - description déclarative comme base
    - d'outils de génération (configuration, déploiement)
    - d'outils d'administration (surveillance, reconfiguration)
    - de preuves de validité

# Liaison

- **La fonction essentielle d'un système**
  - **… et la moins bien comprise**
- **Désignation vs liaison**
  - **désignation : associer un nom et une chose**
  - **liaison : via le nom, donner accès à la chose**
    - **les noms sont eux-mêmes des choses (chaîne de liaison)**
- **Un facteur clé : le moment de liaison**
  - **précoce : optimisation, évaluation partielle Ł efficacité**
  - **tardive (dynamique) : évaluation paresseuse Ł souplesse, adaptation**



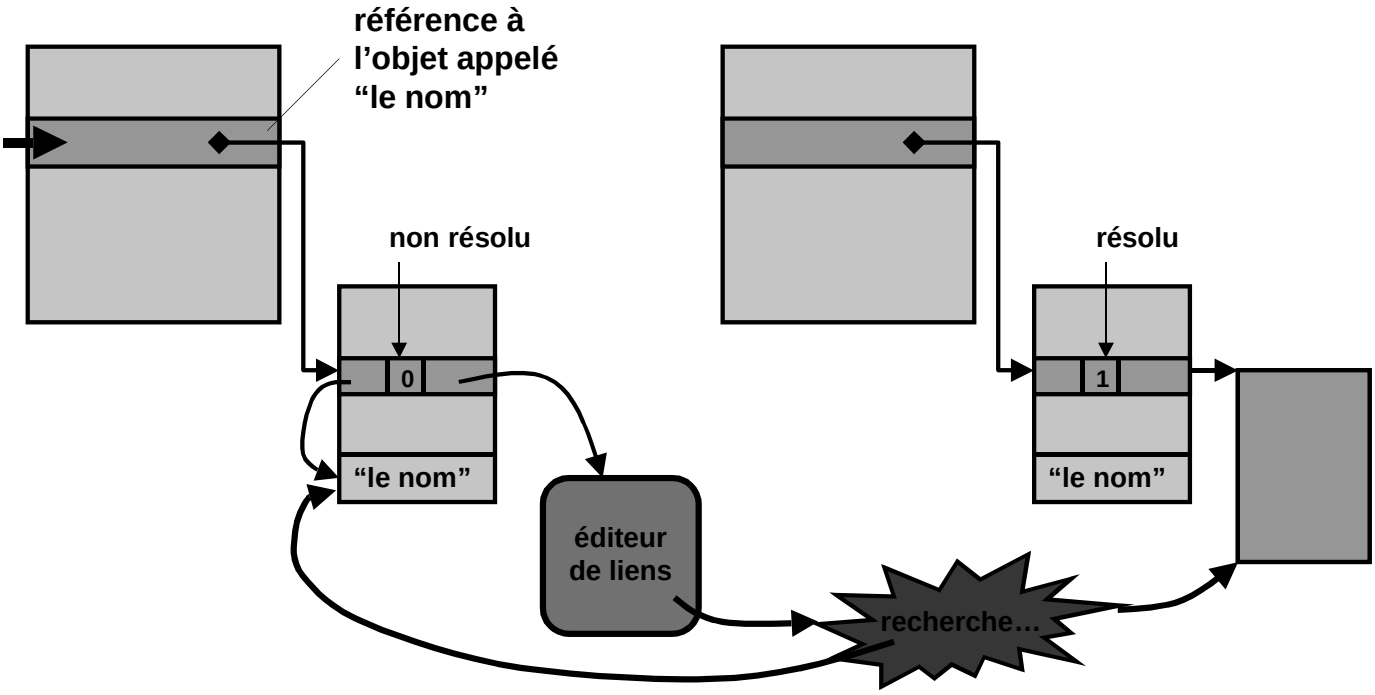
# Liaison d'un objet local

- **Principe**
  - **Accès à un objet = interprétation d'une adresse (physique) par un programme**
  - **Opération de liaison**
    - **obtenir l'adresse physique**
    - **(éventuellement) modifier, voire créer, le programme**
- **Réalisation**
  - **substituer l'adresse physique à la désignation initiale**
    - **éditeurs de liens**
  - **utiliser l'indirection : un objet intermédiaire contient l'adresse physique**
    - **descripteurs de fichiers, segments de liaison, etc.**
    - **indirection multiple**
  - **génération ou chargement du code d'accès**

**+ tardif**
**+ dynamique**

# La puissance de l'indirection (1)

## ■ Exemple classique de liaison dynamique : Multics





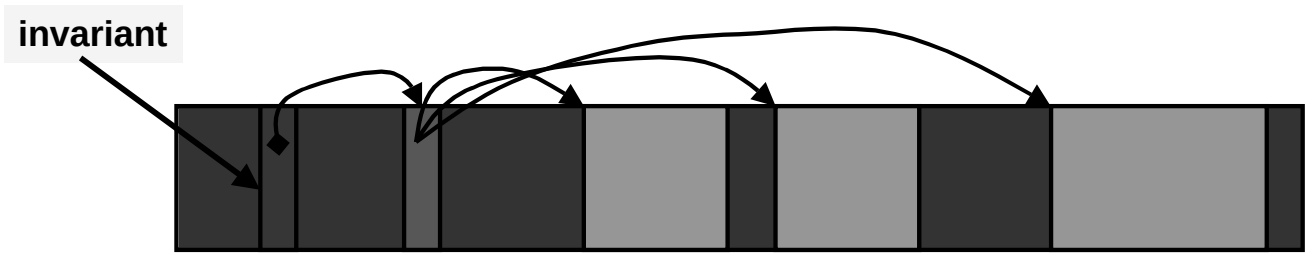

# La puissance de l'indirection (2)

***Any problem in computing can be solved by another level of indirection.***

**David Wheeler**

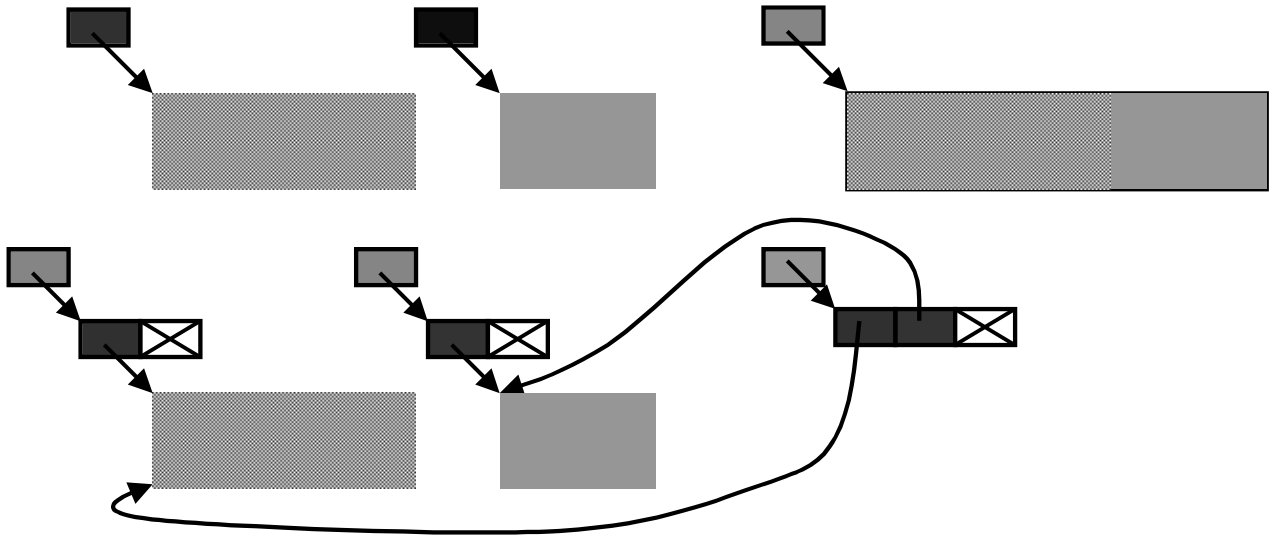
## ■ Exemple : allocation dynamique de mémoire

- **poignée (*handle*) : pointeur sur un pointeur**
- **désigne une zone de mémoire d'adresse variable par une adresse invariante**



# La puissance de l'indirection (3)

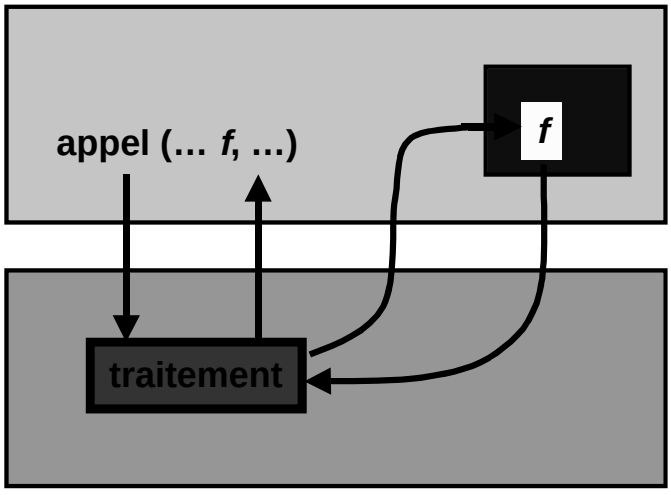
- **Propagation de messages sans copie physique**
  - composer et transmettre des messages par mouvement de pointeurs
  - exemple : concaténer deux messages





# Liaison dynamique : fonctions en paramètre

- **Appel en retour (*callback*)**




***f*** **réalise un traitement spécifique pour l'objet (composant, …) appelant, par exemple la gestion de la persistance, etc.**

**Intérêt : interfaces génériques ; séparation des fonctions ; évolution de *f* indépendamment du traitement principal**

**Exemples : inclusion de code utilisateur dans un appel système**
**réalisation des serveurs dans les systèmes à micro-noyau**

## Liaison par spécialisation de programme (1)

- **Idée : évaluation partielle**
  - L'évaluation de f(x, y) peut être accélérée si on connaît à l'avance la valeur x0 de x (on produit le code de $f_{x0}(y) = f(x0,y)$)
  - En fait, la situation est en général moins simple
    - x est une structure de données complexe
    - l'optimisation est non triviale, et le gain n'est pas assuré
    - le fait que x est constant n'est pas certain, mais probable
- **La spécialisation (liaison) peut être précoce ou tardive**
  - Précoce : le code spécialisé est créé à l'avance (à la compilation)
  - Tardive : le code spécialisé est créé au dernier moment, à la volée (au moment de l'exécution)



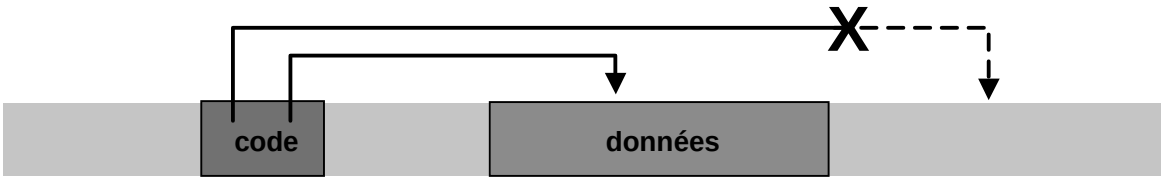
## Liaison par spécialisation de programme (2)

- **Exemple : le noyau Synthesis**
- **Trois méthodes d'optimisation**
  - Factorisation des constantes
  - Élimination des coûts superflus
    - pas de synchronisation sur données si utilisateur unique
    - commutation de contexte réduite au strict nécessaire
    - synchronisation optimiste si conflit peu probable
      - ▲ retour en arrière si conflit détecté
  - Structures de données exécutables
    - insertion d'instructions dans les structures de données pour optimiser parcours répétés

C. Pu, H. Massalin, J. Ioannidis. The Synthesis Kernel, *Computing Systems*, 1(1), 11-32, 1988

# Insertion de code pour l'interposition

- Protection mutuelle de plusieurs programmes partageant un espace d'adressage
  - une zone de confinement pour chaque programme
  - appels interzones possibles, via guichets
- Tout accès à la mémoire est vérifié
  - statiquement (avant exécution), si c'est possible
  - sinon, dynamiquement, par insertion de code
    - vérification de l'adresse avant chaque instruction d'accès
  - surcoût acceptable (entre 5 et 30%)




R. Wahbe, S. Lucco, T. Anderson, S. Graham. Efficient Software-Based Fault Isolation, *Proc. 14th Symposium on Operating Systems Principles*, Dec. 1993

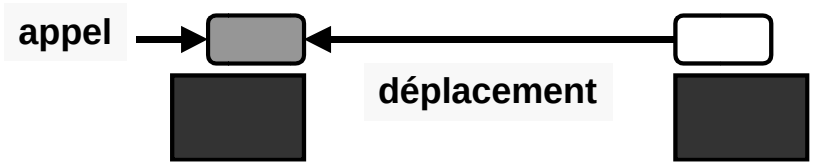


# Liaison en système réparti

- Problème : accéder à un objet distant
- Principe : se ramener à un accès local
  - en créant un exemplaire local de l'objet
    - mobilité ou duplication
  - en créant (ou chargeant) le code d'accès sur le site de l'objet
    - "function shipping"
    - code mobile, agents mobiles
    - réseaux actifs
  - en créant un "représentant" local de l'objet distant
    - généralise le principe du RPC
  - toutes ces opérations peuvent être itérées

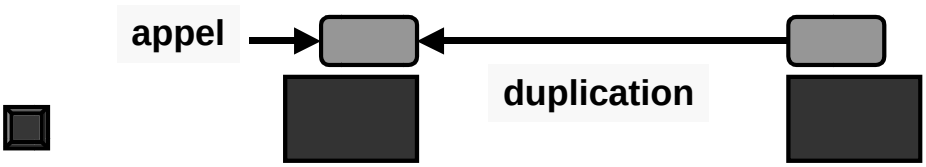
# Liaison dynamique d'objets: mobilité et duplication

- **Mobilité**
  - Chargement dynamique d'un objet appelé sur le site de l'appel
    - *call by move* (Emerald)




effet ping-pong (ou appel distant)

- **Duplication**
  - Création dynamique d'une copie de l'objet appelé sur le site de l'appel

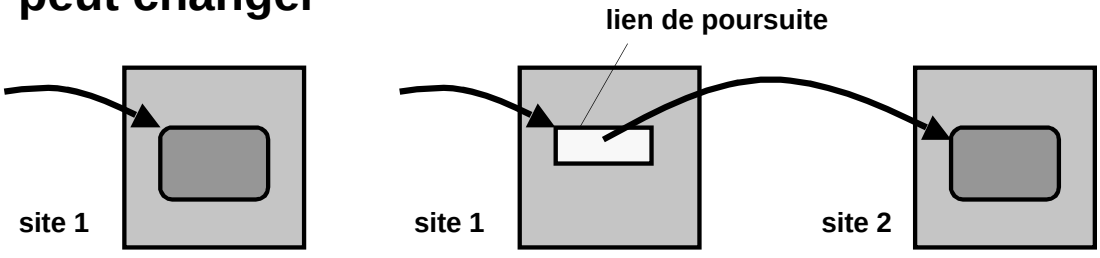



maintien de la cohérence




# Indirection en réparti : liens de poursuite

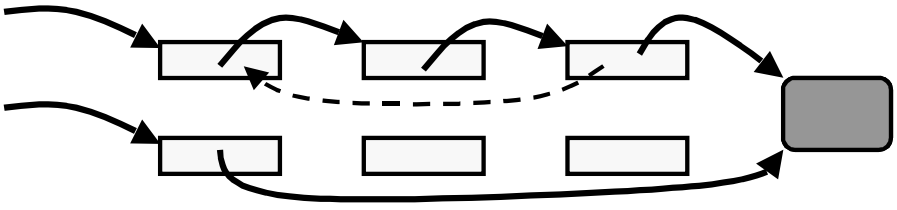
- **Problème : accéder à un objet dont la localisation peut changer**




**Problèmes induits**

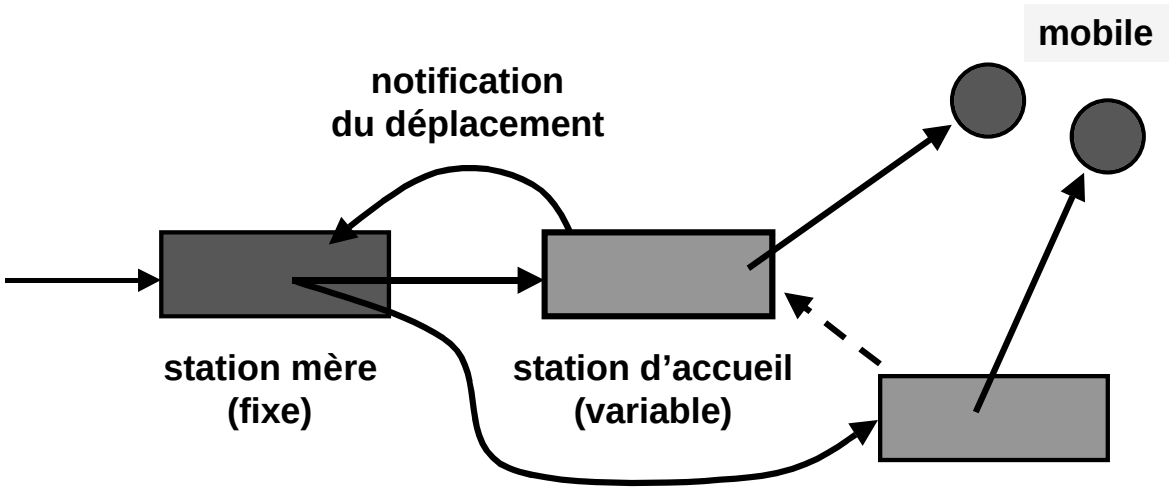
raccourcissement des chaînes
ramassage des miettes

durée de vie des liens
atomicité



# Indirection en réparti : Mobile IP

- **Problème : accéder à une station mobile via une adresse IP fixe**




**Problèmes induits**

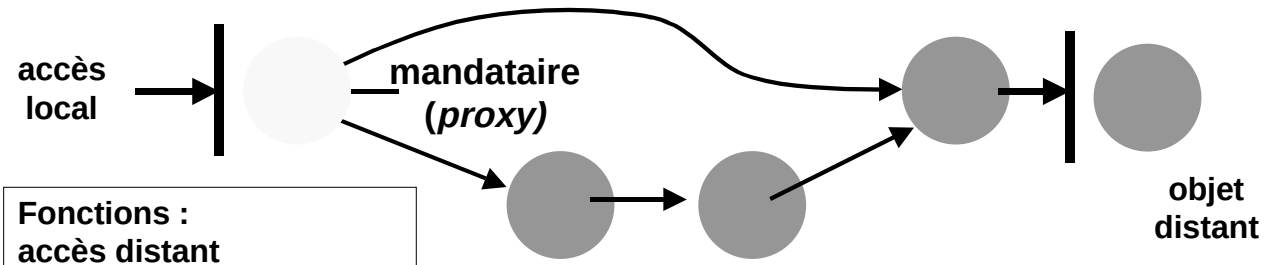
**gestion des régimes transitoires**

**raccourcissement des liens entre deux mobiles**



# Liaison avec un objet distant

- **Principe**
  - **Généraliser l'idée du RPC**
  - **Créer un représentant local de l'objet distant avec la même interface**

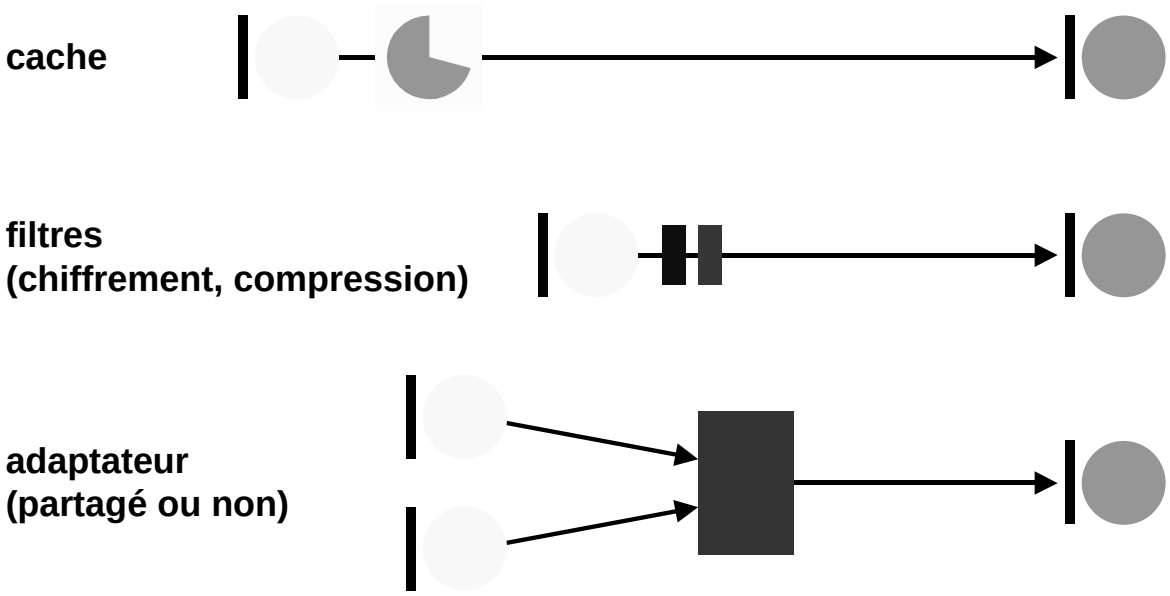



**Fonctions :**
**accès distant**
**emballage de paramètres**
**extensions diverses**

M. Shapiro. Structure and Encapsulation in Distributed Systems: the Proxy Principle, *Proc. International Conference on Distributed Computing Systems (ICDCS)*, 1986

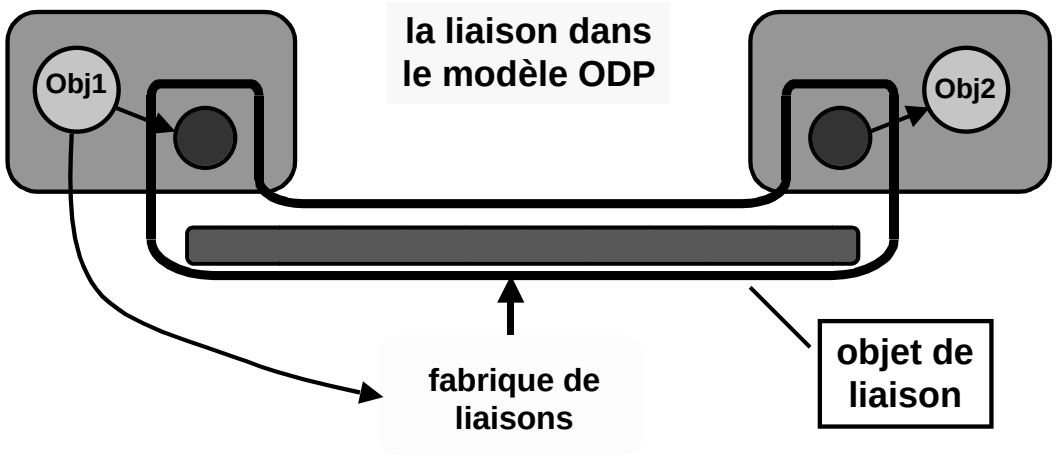
# Avatars du mandataire

- La notion de mandataire se prête à de multiples extensions






# L'interposition comme mécanisme de liaison




- L'objet de liaison peut prendre des formes très diverses
  - talon + squelette + protocole(s) de communication + réseau
  - duplication ou migration d'objet
  - génération ou chargement de code
- canevas unificateur : la fabrique de liaisons (*binding factory*)

# Conclusion sur la liaison

- **Une grande variété de mécanismes…**
- **…mais quelques paradigmes de base**
  - **interposition vs substitution de nom**
  - **interposition : variantes multiples, complexité croissante**
    - **indirection simple ou multiple**
    - **“représentant” ou “mandataire”**
  - **copie ou déplacement**
- **Vers un degré croissant de dynamisme**
  - **modification dynamique de la liaison**
  - **modification dynamique du mécanisme générateur de liaison**
- **Vers un début d’unification**
  - **fabrique de liaisons**
  - **fabriques (de fabriques)***



# Réification

**Réifier une entité, un concept = lui donner une forme, une représentation concrète – Démarche inverse de la virtualisation**

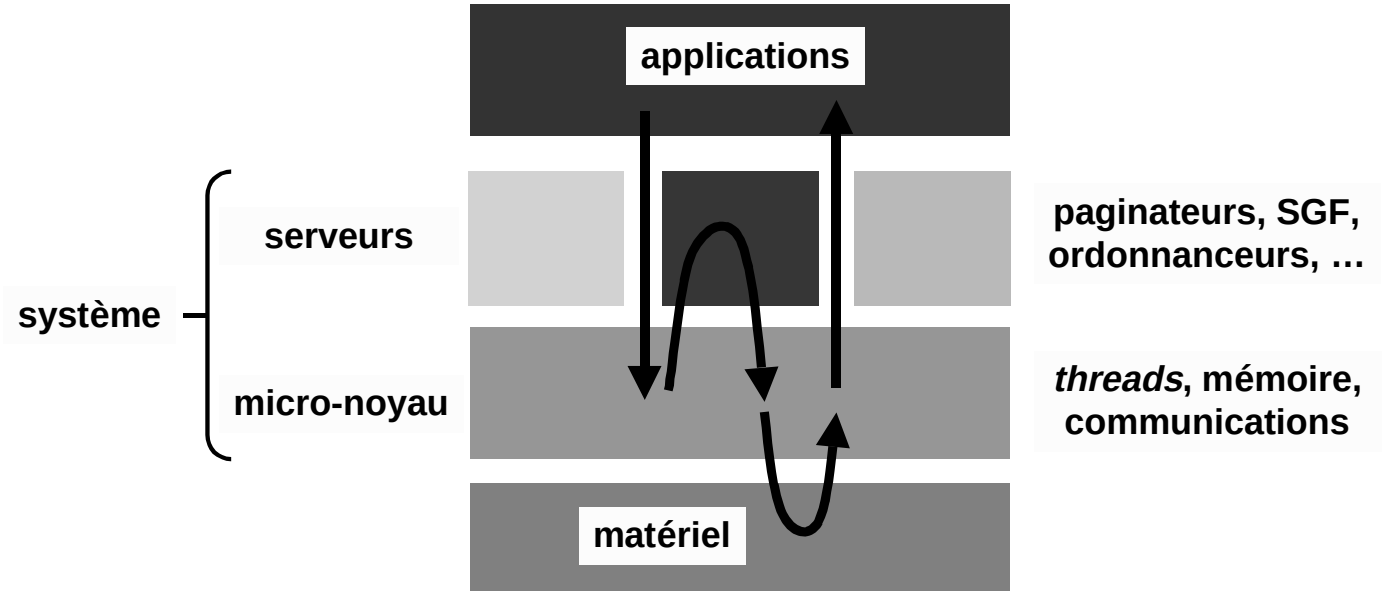
- **Motivation**
  - **la virtualisation dégage l’utilisateur du souci de la gestion des ressources virtualisées …**
    - **… mais le prive par là-même de la maîtrise de leur usage**
    - **… et dissimule leur interface**
  - **l’excès de virtualisation pénalise les performances et l’adaptabilité**
- **Méthodes**
  - **extension dynamique**
  - **méta-objets et systèmes réflexifs**
- **Domaines d’application**
  - **micro-, nano-, exo-, ... noyaux**
  - ***middleware*, systèmes répartis**

# Micro-noyaux : motivations

- **Éviter les défauts des noyaux monolithiques**
  - peu compréhensibles, peu modifiables
  - politique imposée pour la gestion des ressources
- **Rechercher les avantages d'une structure modulaire et d'abstractions simples et maniables**
  - construction de systèmes "sur mesure"
    - adaptabilité à des situations diverses
    - coexistence possible (et isolation mutuelle) de plusieurs systèmes
  - meilleure utilisation des ressources
    - contrôle fin "personnalisé" des politiques de gestion
  - réduction du noyau (mode superviseur) au strict minimum



# Micro-noyaux : organisation



## Micro-noyaux : réalités

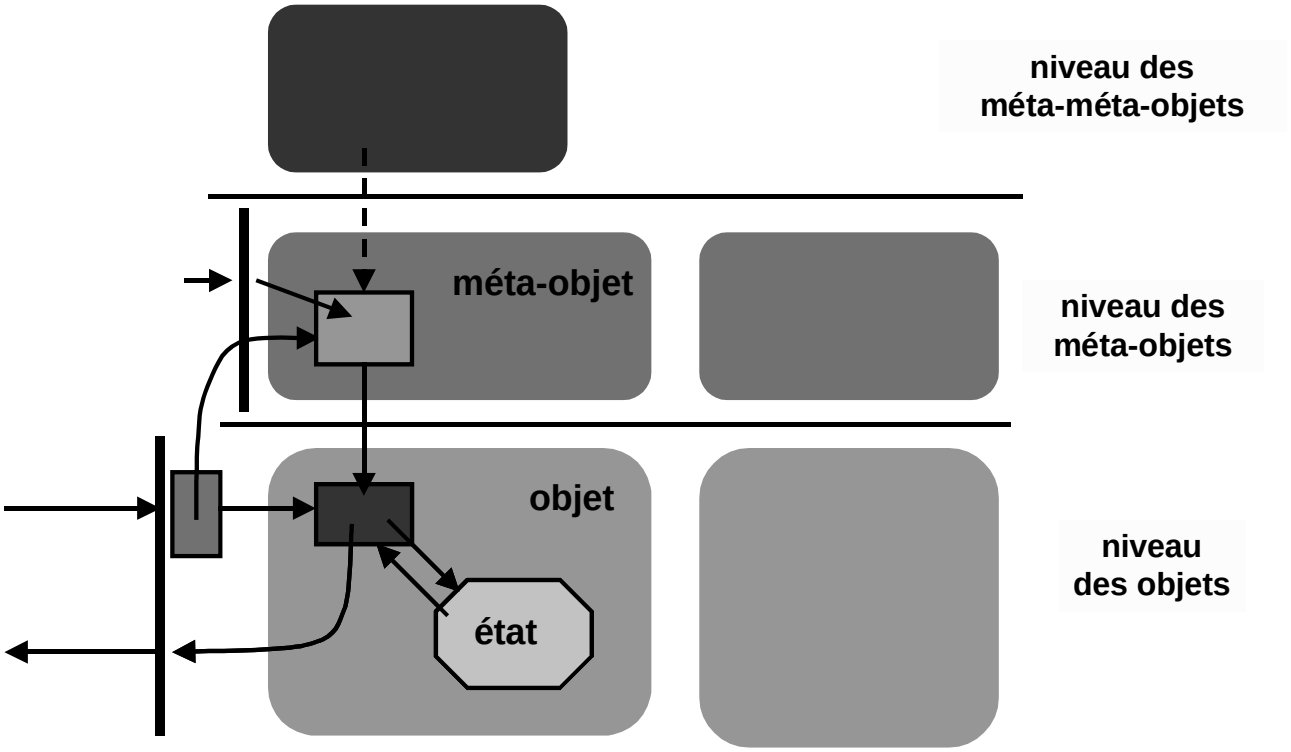
- **La première génération de micro-noyaux n'a que partiellement répondu aux attentes …**
  - **les micro-noyaux ont (en partie) reproduit les inconvénients qu'ils visaient à éviter**
    - **interfaces trop complexes**
    - **noyaux trop volumineux**
    - **noyaux monolithiques, peu adaptables**
    - **trop de choix prédéfinis et imposés**
  - **conséquences**
    - **performances décevantes**
    - **degré insuffisant d'adaptabilité des systèmes**



## De meilleurs micro-noyaux : quelques voies d'approche (pouvant être combinées)

- **Améliorer les performances**
  - **analyse fine des causes de pertes, adaptation étroite de l'implémentation au matériel : L3, L4**
- **"Éliminer les abstractions"**
  - **réduction du noyau au minimum, interface (et non seulement implémentation) dépendante du matériel : Exokernel, Nemesis**
    - **le système est essentiellement dans l'espace de l'application**
    - **le noyau réifie (et alloue) les divers composants du matériel**
- **Rendre le système extensible**
  - **par chargement dynamique de code : Spin, Vino, Flux**
  - **par génération dynamique de code : Synthetix**
- **Rendre le système réflexif**
  - **méta-objets réifiant diverses fonctions du système : Aperios, Panda**

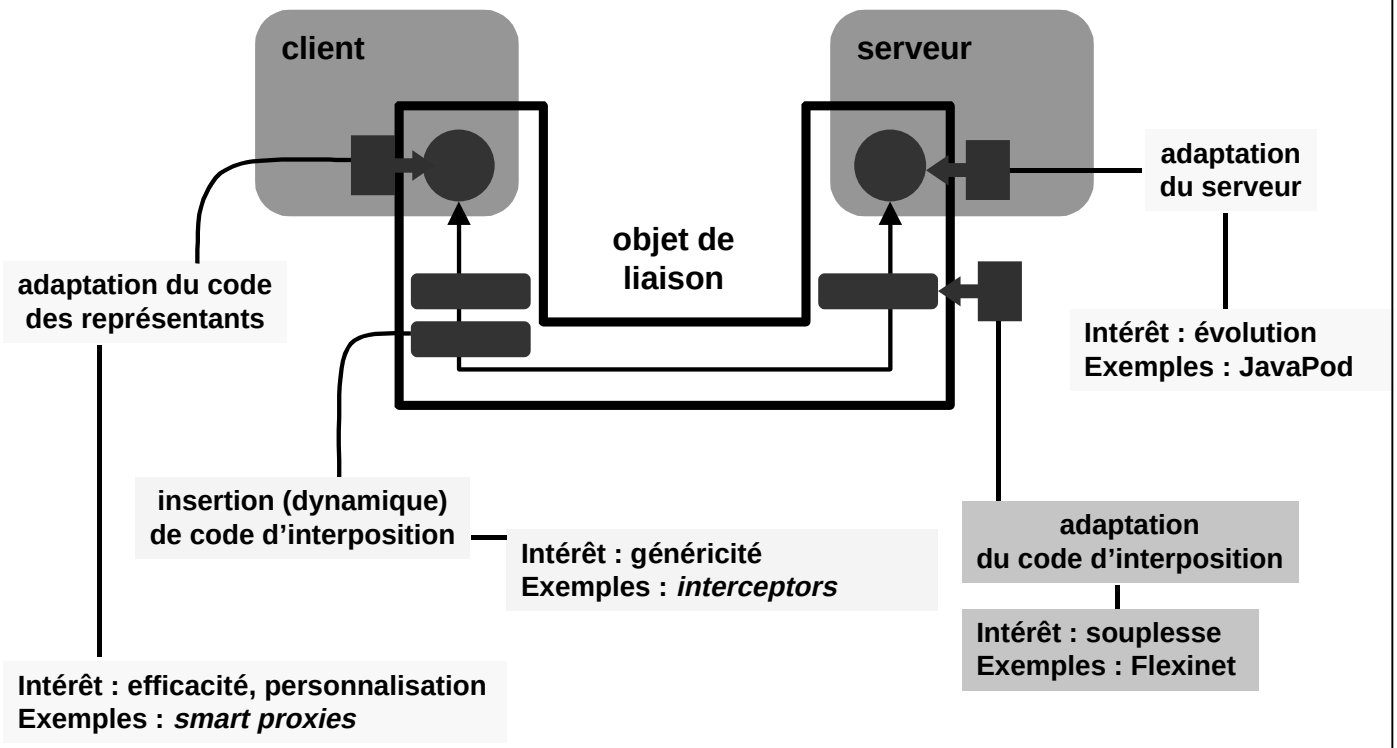
# Réflexion et méta-objets







# Réification de la liaison



## Conclusion sur la réification

- **Un domaine de recherche actif**
  - recherche du compromis entre :
    - abstraction et contrôle fin des ressources
    - adaptation statique ou dynamique des applications
- **De nombreux problèmes**
  - sécurité
  - performances
  - composition des aspects (indépendance ou interaction)
  - maîtrise du développement, expressivité, uniformité du modèle,
- **Quelques résultats**
  - expérience sur micro-, nano-, exo-noyaux
  - premiers canevas logiciels pour *middleware* adaptable
    - Flexinet, TAO, Jonathan, …



## En guise de conclusion (1)

- **Nous vivons une époque passionnante**
- **Où seront les OS futurs ?**
  - de plus en plus dans des appareils divers, non dans des ordinateurs
  - de plus en plus communicants et réactifs, plutôt qu'isolés
- **Conséquences**
  - "déconstruction"
  - ouverture, adaptation
  - retour à la simplicité
  - systèmes prouvables ?
  - génération automatique ?
  - fin de l'OS ? l'OS dans les applications ?

# En guise de conclusion (2)

- **Sur les paradigmes de l'architecture**
  - permanence des concepts, raffinement (lent…) dans leur application
    - Pipes Unix (74), Streams Unix (85), *x*-kernel (91), Click et Knit (99)
    - CP (67), RPC (84), Subcontracts (93), usines (réflexives) à liaisons (actuel)
  - nouveaux paradigmes à inventer : mobilité, sécurité, …
- **Quelques défis pour l'avenir**
  - Conceptuels
    - modélisation (formalisation ?) de l'architecture
    - modélisation de la sécurité
  - Pratiques
    - génération d'OS pour environnements spécialisés
    - description déclarative des environnements et contraintes
    - administration à grande échelle



# Formation

- **Ne pas oublier l'histoire**
  - pour identifier les principes et les paradigmes
  - pour analyser les facteurs de succès et d'échec
  - pour s'inspirer des "beaux ouvrages"

P. Brinch Hansen. *Classic Operating Systems*, Springer, 2000

- **Apprendre en faisant**
  - des projets aussi ambitieux que possible, mais juste assez pour être faisables
  - expliciter les choix de conception et de réalisation
  - produire de la documentation, pas seulement du code
  - l'idéal : un système partiellement réalisé, à compléter
  - un bon support : le logiciel libre
- **Apprendre à tirer les leçons de l'expérience**

# Un conseil final …

***Be aware of the fact that experience does by no means automatically lead to wisdom and understanding; in other words, make a conscious effort to learn as much as possible from your previous experiences.***

**E. W. Dijkstra**

***in*** **"The Structure of the THE Multiprogramming System", *Comm. ACM*, May 1968**

## ■ … et un paradigme de son application

**B. W. Lampson. Hints for Computer Systems Design, *IEEE Software*, 1(1), pp. 11-28, Jan. 1984 (initialement dans *ACM SOSP*, 1983)**