Cnam - Cedric

CONSTRUCTION DE SYSTEMES

-

MICRO-NOYAUX

Eric Gressier-Soudan

Plan

Introduction

Micro-Noyaux

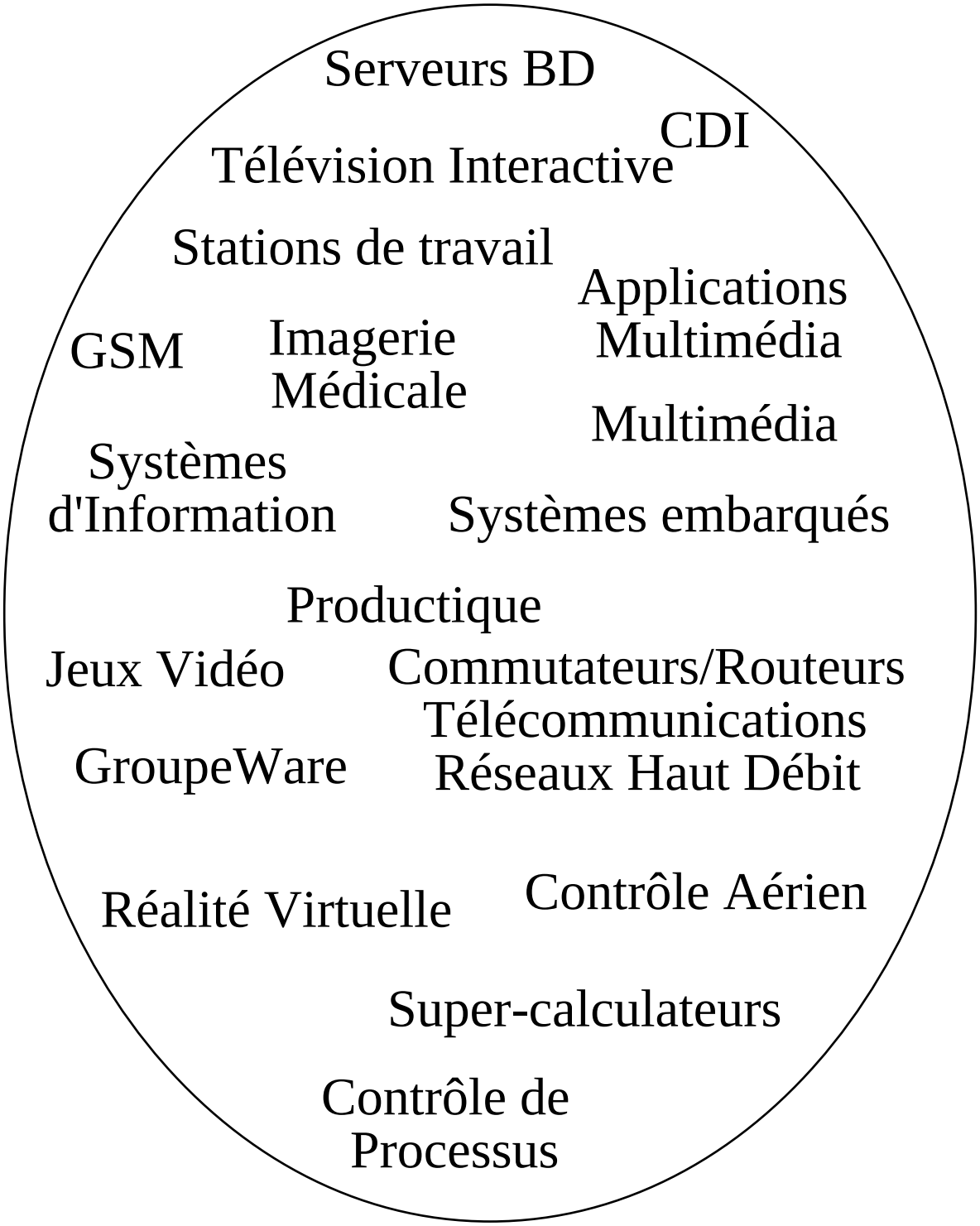
Sous-Systèmes

Systèmes à Objets

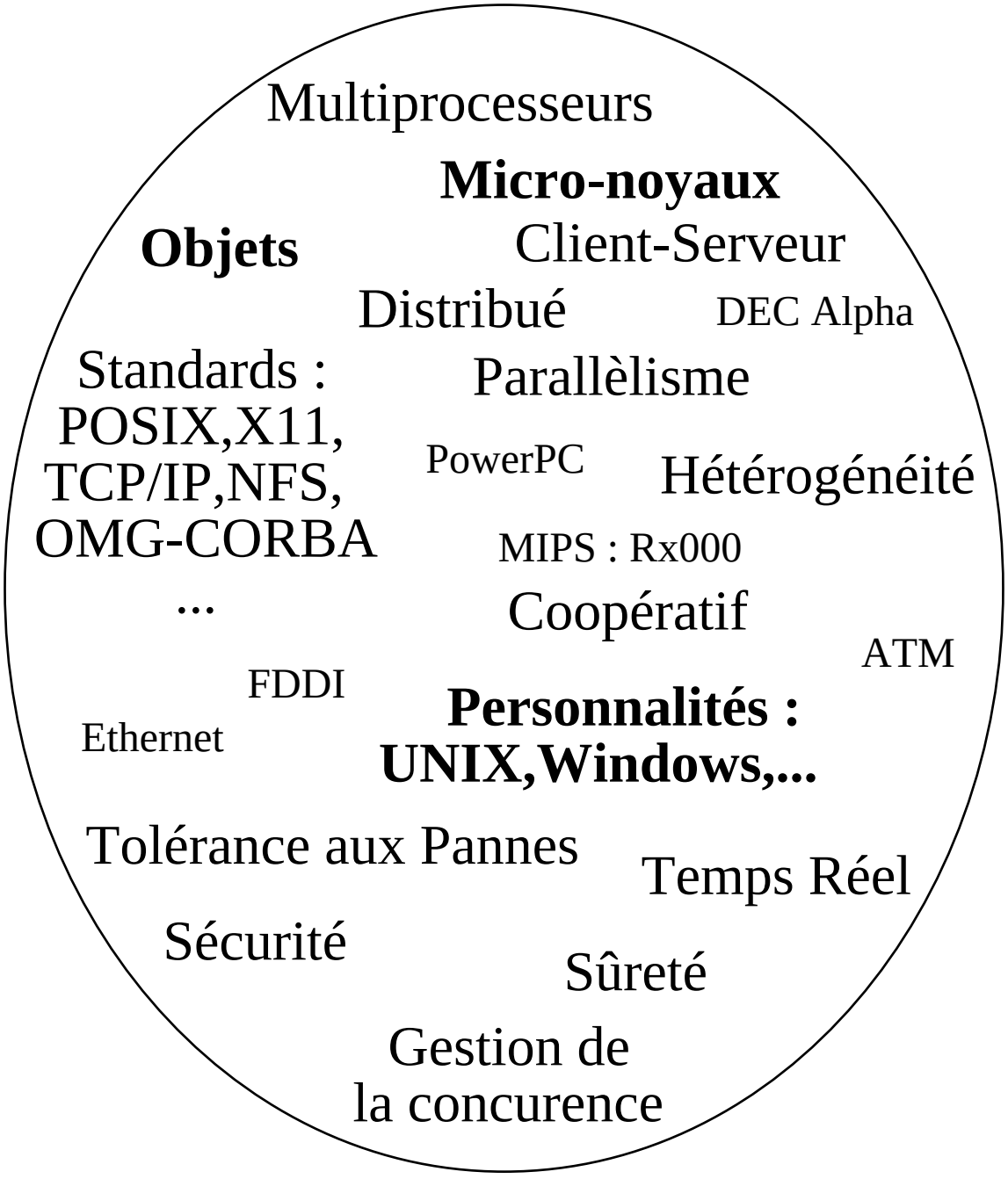
Abstractions de base

INTRODUCTION

APPLICATIONS



Caractéristiques des solutions



Objectifs en conception des systèmes répartis (1)

. **masquer** la répartition (**transparence**)

-> **localisation** des ressources *non perceptible*

-> **migration** des ressources possible sans transformation de l'environnement utilisateur

-> *invisibilité* de la **réplication** des ressources

-> **concurrence** d'accès aux ressources *non perceptible*

-> *invisibilité* du **parallèlisme** sous-jacent offert par l'ensemble de l'environnement d'exécution

Utilisateur = humain ou programme

Objectifs en conception des systèmes répartis (2)

. **Flexibilité** / Modularité

->Noyaux Monolithiques contre **Micro-noyaux**

Objectifs en conception des systèmes répartis (3)

. Fiabilité

N machines plus fiable qu'une seule ?

-> **Disponibilité** :

Leslie Lamport donnant une définition d'un système distribué :
"One on which I cannot get any work done because some machine I have never heard of has crashed"

Avec NFS, par exemple, on souhaiterait éliminer l'interdépendance des serveurs d'un réseau de stations pour éviter les attentes ou les blocages liés au montage de partitions distantes.

* indépendance des composants du système

* redondance logicielle et matérielle

Objectifs en conception des systèmes répartis (4)

. Fiabilité

-> **Tolérance aux pannes / Sûreté de fonctionnement** :

spécification : comportement vérifiable par **un ensemble de prédicats/des assertions** qui tient compte des entrées, des sorties du système et du temps qui lui est imparti pour exécuter son travail

mode normal : le système rend un service qui respecte à tout moment les spécifications sur lesquelles reposent sa conception

mode dégradé : la perte de fonctionnalité n'empêche pas les spécifications d'être respectées mais avec éventuellement une baisse de performance

mode panne : le comportement du système est hors spécifications

Objectifs en conception des systèmes répartis (5)

. Fiabilité

-> Sécurité

Protection contre les accès non autorisés

. propriétés visées :
contrôle d'accès,
authentification,
intégrité,
confidentialité,
non-répudiation

. techniques :
cryptographie à clefs privées ou à clefs publiques,
fonctions univoques ou hachage,
signature
protocoles de sécurité

Objectifs en conception des systèmes répartis (6)

. **Performance** malgré la transparence, la flexibilité, la fiabilité

. **Gérer le grand nombre**

beaucoup d'utilisateurs
de machines
de services
de trafic
=> centralisation impossible

hétérogénéité

intégrabilité et extensibilité

. **Maintenabilité** Pas pour les concepteurs ... mais pour les ingénieurs système !!!

Impossibilité des systèmes répartis

PAS D'ETAT GLOBAL (facilement accessible)

. Pas de référence de temps (HORLOGE) commune entre les différents sites d'un système réparti !!!

. Impossibilité de fabriquer une image du système à tout instant ... pb des états globaux instantanés ("distributed snapshots")

Les prises de décisions se font sur la base d'informations locales à chaque site[1].

Les Univers Distribués sont foncièrement probabilistes

=> . Problèmes d'Algorithmique répartie

. Le Déverminage en réparti est une tâche complexe

---

[1] Le problème d'ouverture de connexion de transport est typiquement un problème d'algorithmique répartie résolu par une démarche probabiliste : au bout de 3 messages(three way handshake) on a un maximum de chances que la connexion soit ouverte entre deux sites ... mais on en a pas la certitude absolue (voir pb des deux armées).

Micro-noyaux

Paradigme

Point de vue de l'ingénierie des systèmes répartis :

Les micro-noyaux forment la base des nouvelles architectures de systèmes.

Démarche adoptée par :

. Novell/USL ... HP
. Microsoft
. Apple
. DEC
. Sun
. ALCATEL
. Siemens
. Cisco
. Unisys
. Cray
...

Génèse des micro-noyaux (1)

70's → 80's

. **Réseaux longues distances**

. **Protocoles de communication**

. **Applications dédiées :**

- Transfert de fichiers

- Terminal Virtuel

- Messagerie Electronique

Réseaux Locaux + Systèmes Monosites

. Systèmes de fichiers répartis (ex NFS-Sun)

. Environnement d'exécution réparti (ex ONC-Sun, NCA-Apollo)

Interconnexion

Services Répartis

→ **90's**

. Multi-calculateurs

. Système d'exploitation réparti et vision unifiée du système (Single System Image)

. Accès aux objets systèmes
- Fichiers
- Processus
- Mémoire
- Périphériques

. Objets Distribués
- Groupes de processus
- Systèmes de fichiers

. Objets Migrants
- Processus
- Fichiers

Distribution du système

Génèse des micro-noyaux (2)

Recherche sur les sytèmes (80's)

Grapevine (XEROX)

Accent (CMU)

Amoeba (Université d'Amsterdam)

Chorus (INRIA)

Mach (CMU)

V-System (Université de Stanford)

Sprite (Université de Berkeley)

Systèmes Commerciaux (90's)

Chorus (Chorus Systèmes)

Mach-OSF/1 (CMU-Open Software Foundation)

Amoeba (ACE)

Windows NT (Microsoft)

Système Monolithique qu'est ce que c'est ? (1)

Système Monolithique :

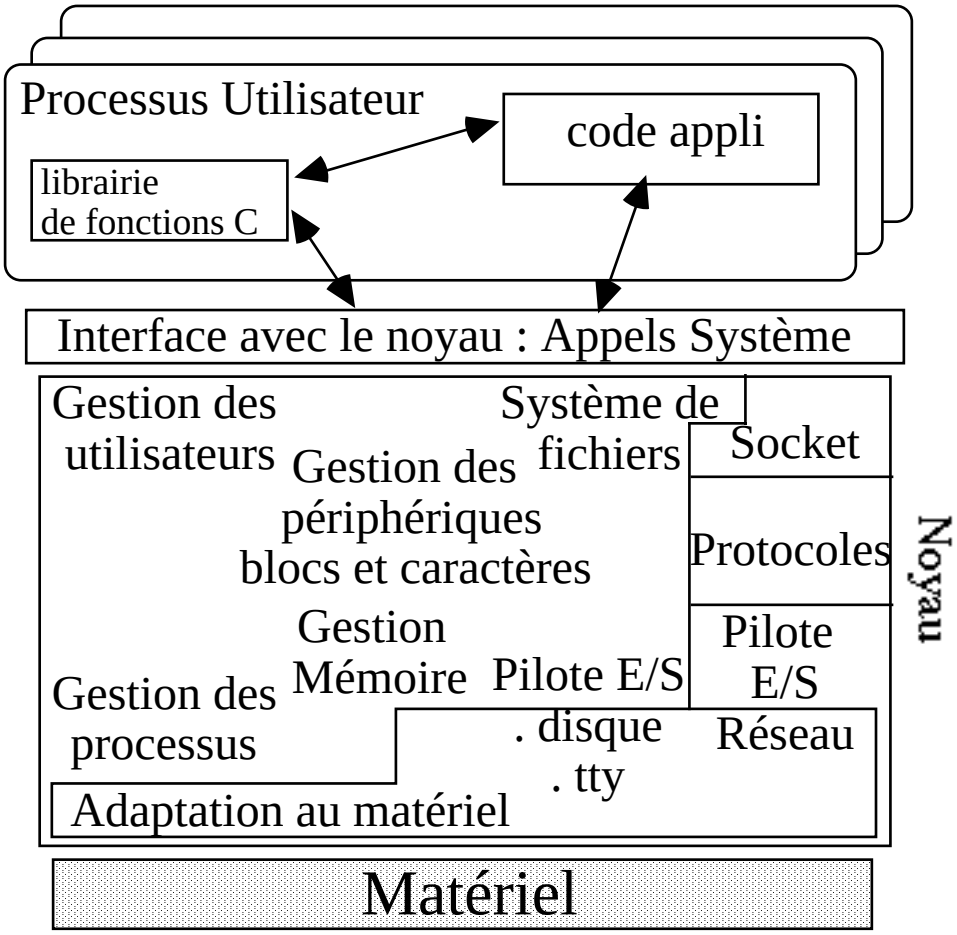
le système est en un seul morceau,

rien ne peut lui être enlevé sans mettre en péril son fonctionnement,

aucune possibilité de voir les ressources (ex fichiers) autrement que part la vue fournie par le système[2]

[2] Pas complètement vrai ... si on reprend Unix System V R4.0 ou 4.4 BSD, le système permet de voir des systèmes de fichiers de différents types : UFS (Berkeley -> rapide), S5 (traditionnel System V-> lent), NFS en particulier, grâce au mécanisme de Virutal File System et de Virtual node (VFS/V-node) empruntés à NFS.

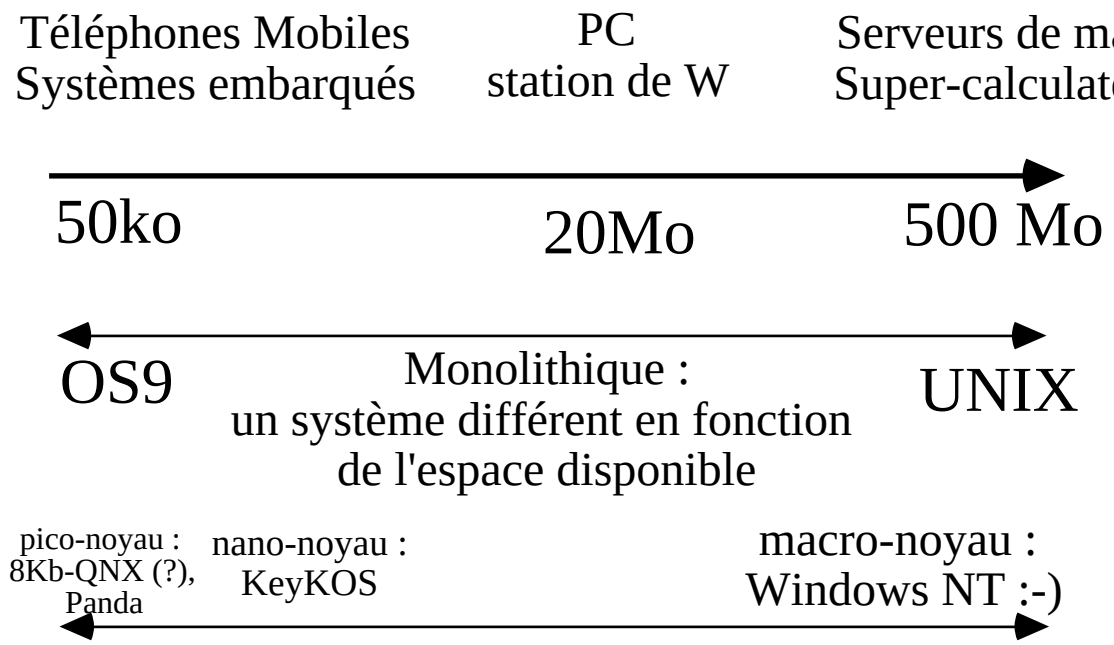
Système Monolithique qu'est ce que c'est ? (2)



Architecture qui ressemble à celle d'un Unix BSD ou d'un Unix System V

Contrainte Matérielle sur la conception du système

Place disponible en mémoire RAM pour héberger le système :



Micro-noyau modulaire :
le même mais avec plus ou moins de fonctions
un exemple : Chorus

Objectifs de conception d'un micro-noyau

. Transparence
- à la localisation
- à la migration
- à la réplication
- au parallèlisme

. Modularité

=>Extensibilité/Portabilité/Configurabilité

- implantation des services sous forme de gestionnaires ou de serveurs spécialisés
- réduction de la complexité
- développement et validation plus faciles

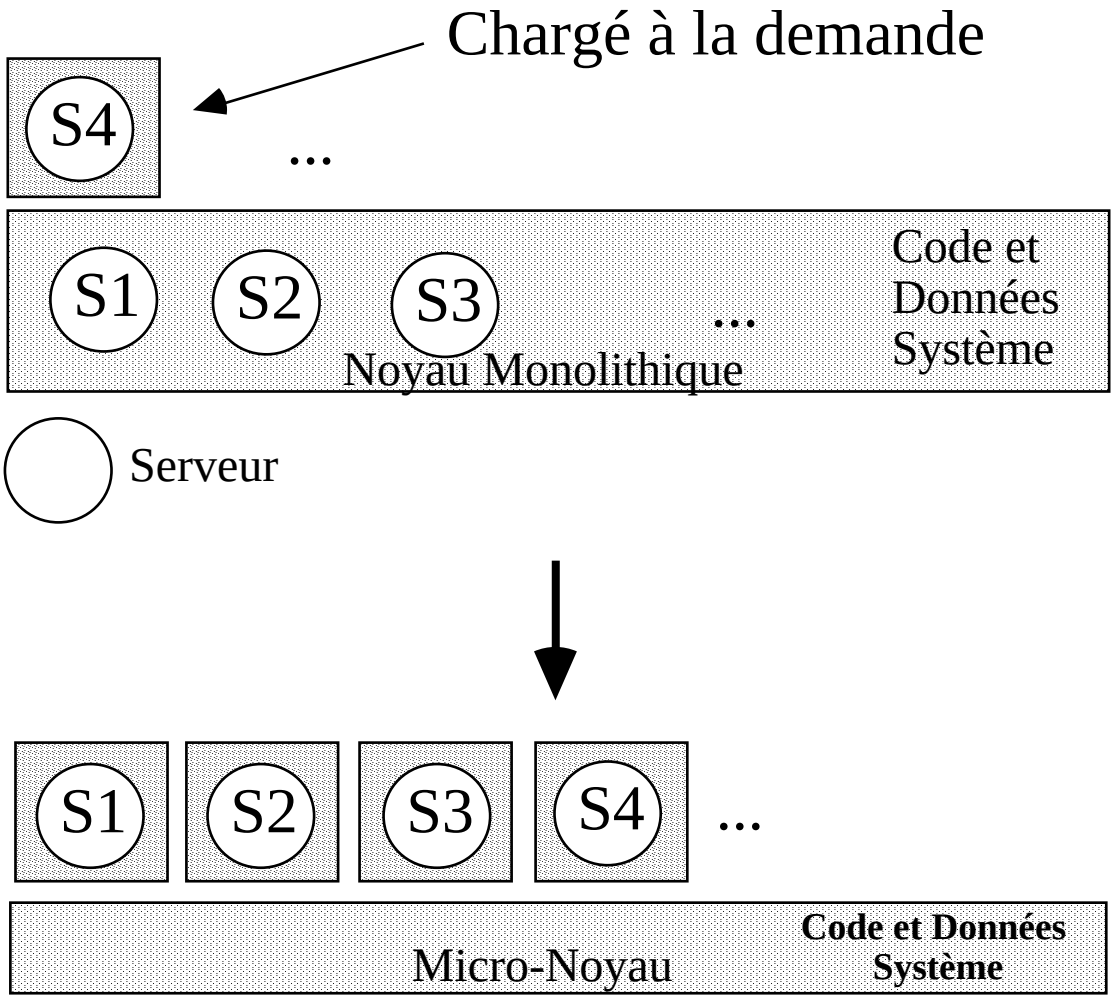
. Tolérance aux fautes/Disponibilité

. Performance

. Sécurité

et un Retour à la simplicité

Gain au niveau de l'architecture



Qu'en est-il de la performance ?

avantage au monolithique[3] ! ? !

[3] mais A. Tanenbaum cite une étude qui compare Sprite (Monolithique) à Amoeba (Micro-noyau) et qui indique qu'il n'y a pas de différence sensible ... il faut relativiser ce type d'affirmation ... il y a un vieux débat qui consiste à dire pour

Organisation du système

. Micro-noyau

=> Fond de panier logiciel

. Serveurs

- de type système : superviseur
- de type utilisateur

. Sous-Systèmes ou machines virtuelles qui définissent des "Personnalités"

l'architecte système dispose d'un "patron"qu'il peut modifier/tailler en fonction de ses besoins

---

gagner de la performance il faut tout compacter le code du système, on y perd toute modularité donc tout les gains liés à l'ingéniérie des logiciels,
par ailleurs, les progrès technologiques, en particulier la baisse du prix des mémoires centrales et la vitesse des processeurs, ont permis de dédier une partie de la bande passante des processeurs à cette modularité sans grande perte de performances, les micro-noyaux entrent dans cette logique

Sous-Système

Personnalités

Sous-système / Sur-système

1. survivre au standard Unix
   pouvoir hériter de l'existant
   -> compatibilité binaire des applications
   éviter l'échec d'Apollo/Domain

2. coexistence d'applications spécifiques avec d'autres conçues pour des systèmes différents
   -> temps réel
   -> Mac
   -> Windows
   -> DOS

3. préparer la transition vers de nouveaux systèmes
   systèmes à objets

Sous-système

Un sous-système :

. une librairie d'appels systèmes
. un serveur ou un ensemble de serveurs

qui permettent la compatibilité binaire pour des applications déjà écrites pour le système émulé.

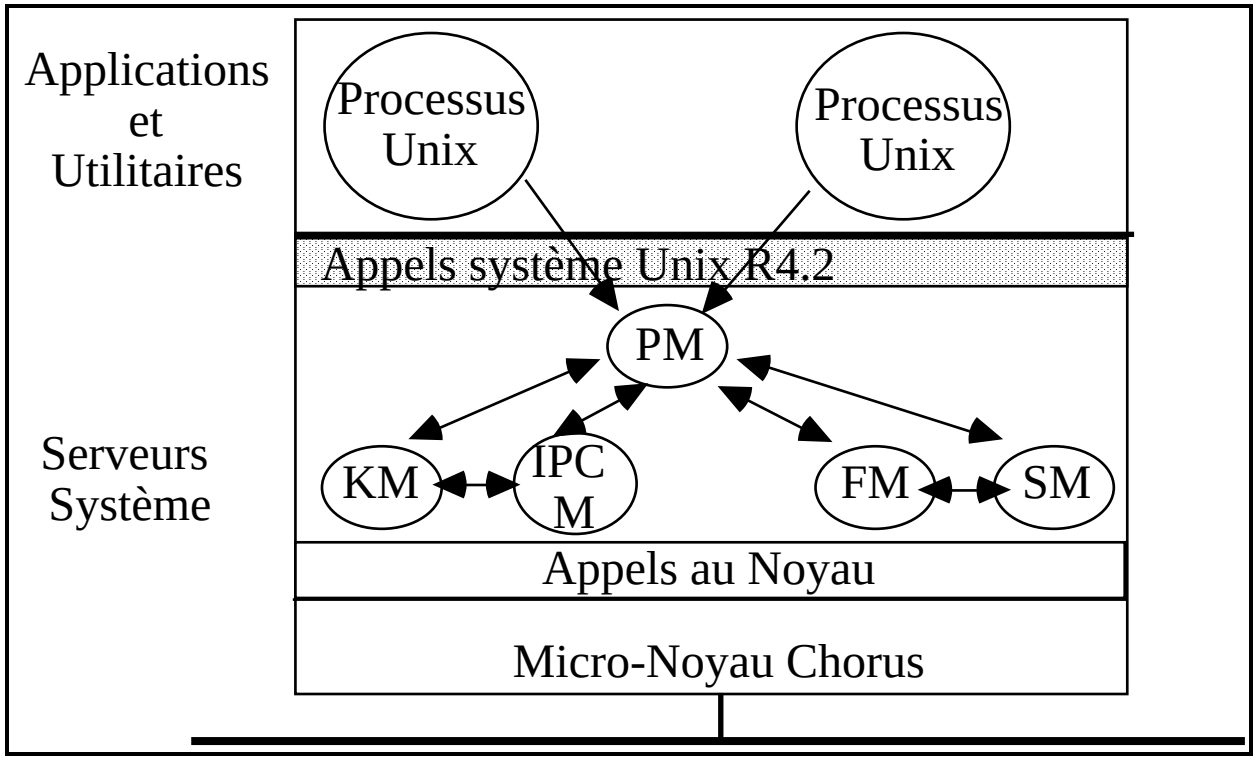
L'utilisateur ne "voit" que le système sur lequel il a l'habitude de travailler.

Image unique du système - "Single System Image"

En général, on lui offre aussi l'accès aux appels système du micro-noyau.

Exemple Chorus et MIX V4 R2

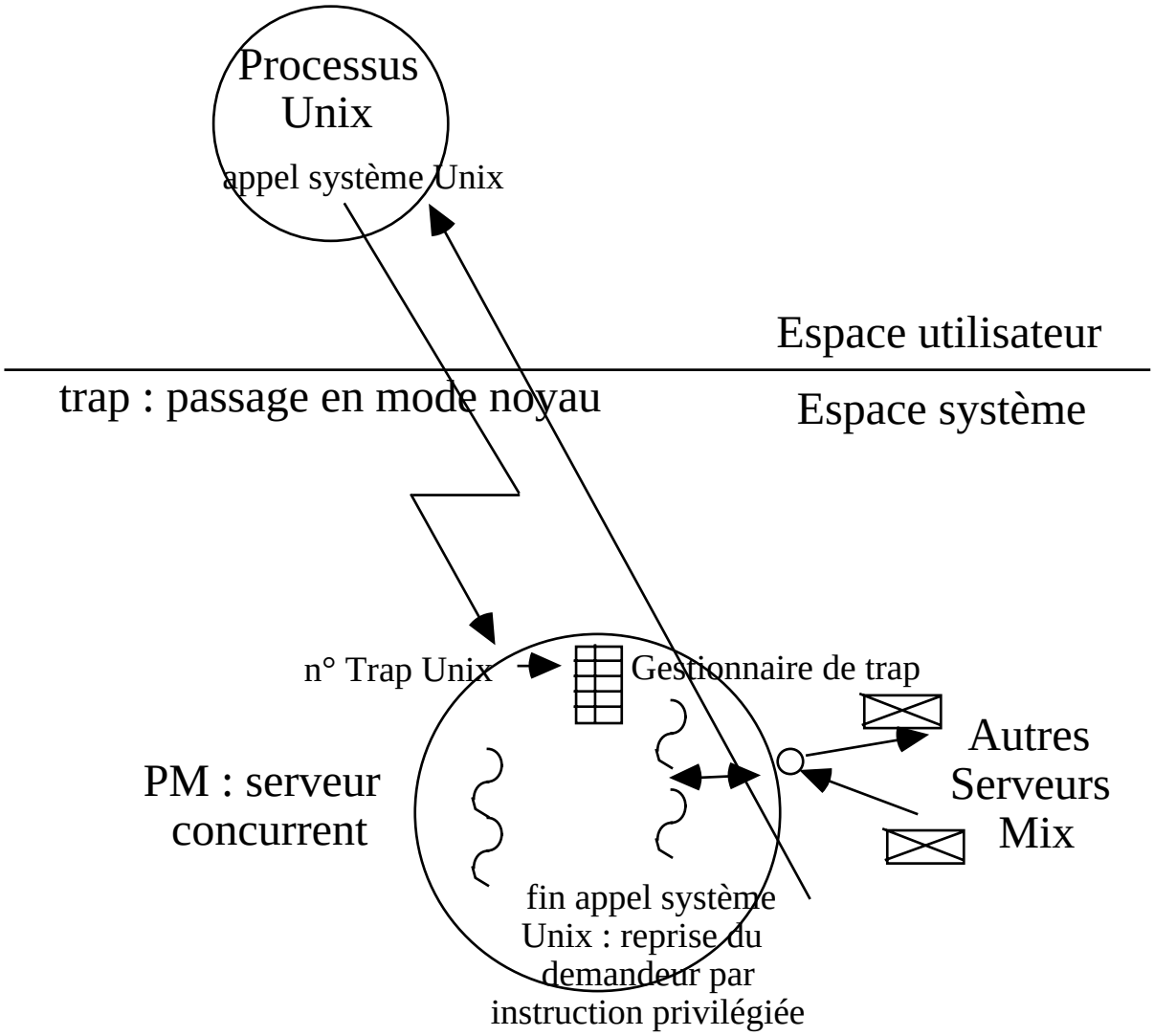
SUSI : Single Unix System Image



PM = Process Manager
FM = File Manager
SM = Streams Manager
IPCM = System V Inter-Process Communication Manager
KM = System V Key Manager
Il existe un sous-système Mix V.3.2, Mix SCO, ClassiX/C-actors, et Mac OS (INT Evry - C. Bac)

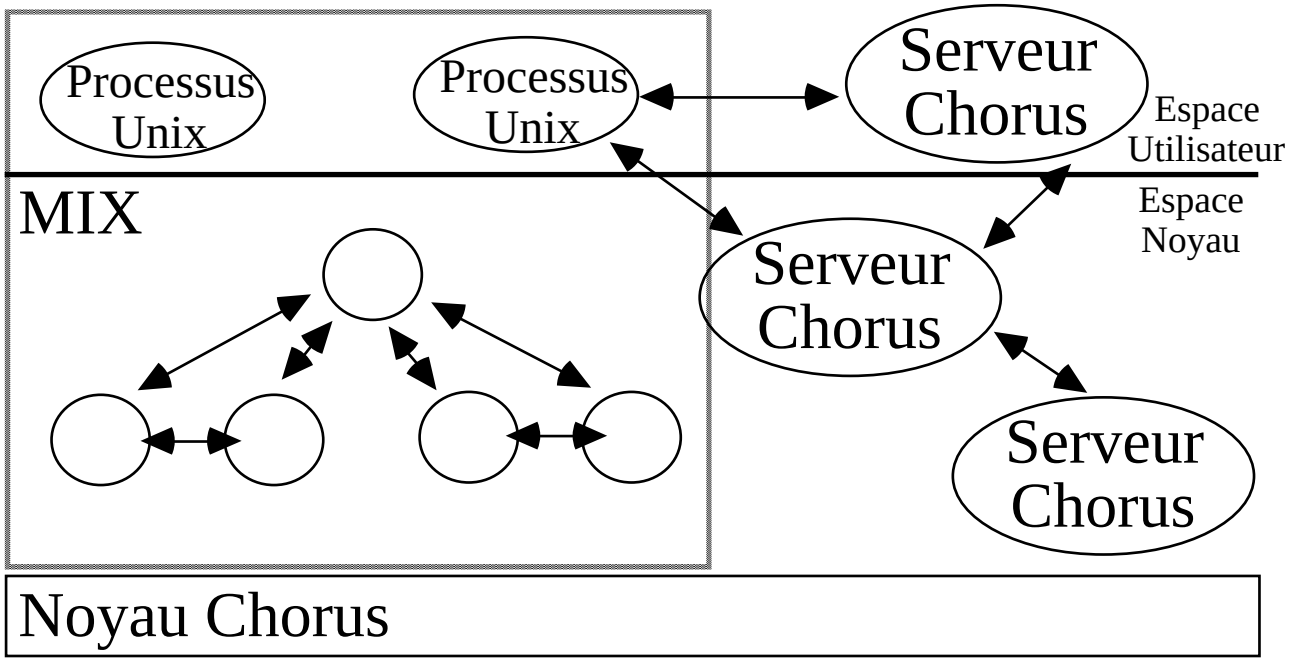
Emulation Unix dans Chorus

Compatibilité Binaire



Avantage de la notion de Sous-système

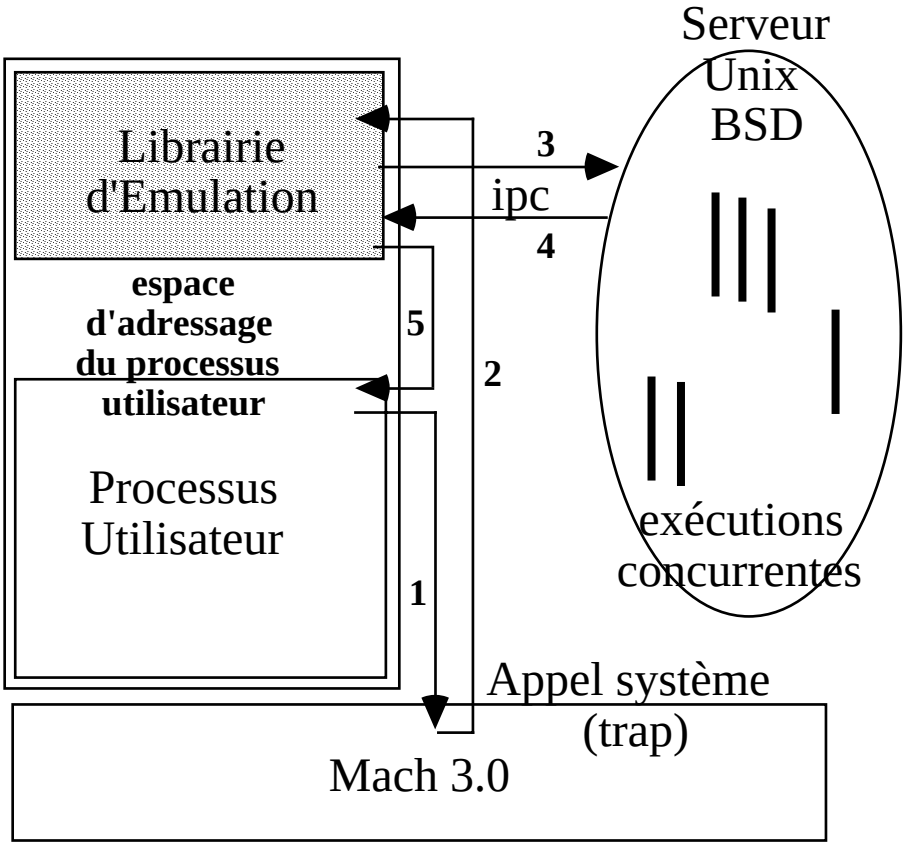
Cohabitation de processus UNIX avec des serveurs Chorus qui coopèrent dans le cadre d'une application temps réel.



Emulation Unix sous Mach

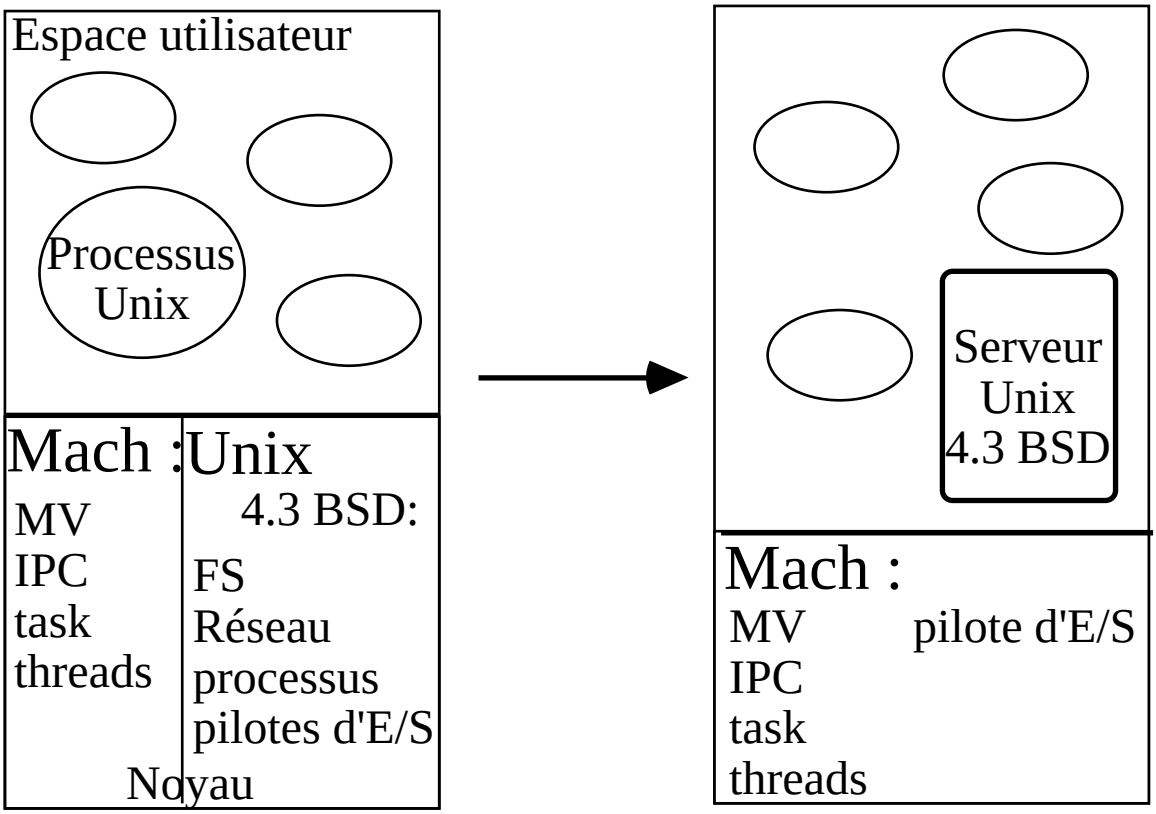
Librairie d'Emulation dans l'espace utilisateur

qui génère des communications (ipc) vers le serveur Unix BSD.



Effet de la modularisation sur les sous-systèmes et le micro-noyau (1)

Génèse de Mach/OSF1 :



Mach 2.5 / OSF1-IK
Integrated Kernel

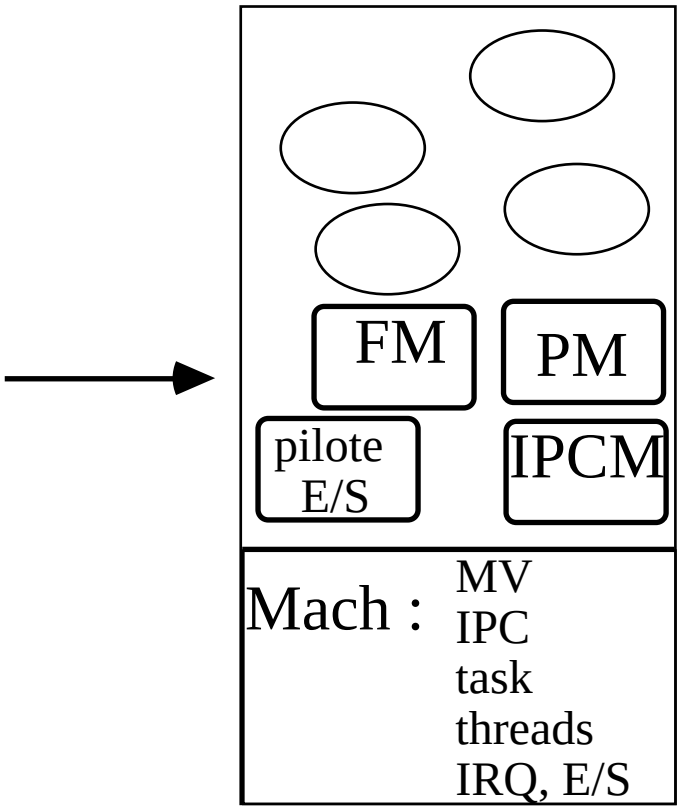
Unix parallèlisé, le support du multiprocesseur

Mach 3.0 / OSF1-MK
Micro-Kernel

Mach possède d'autres sous-systèmes prévus pour cohabiter : MS-DOS, VMS en particulier

Effet de la modularisation sur les sous-systèmes et le micro-noyau (2)

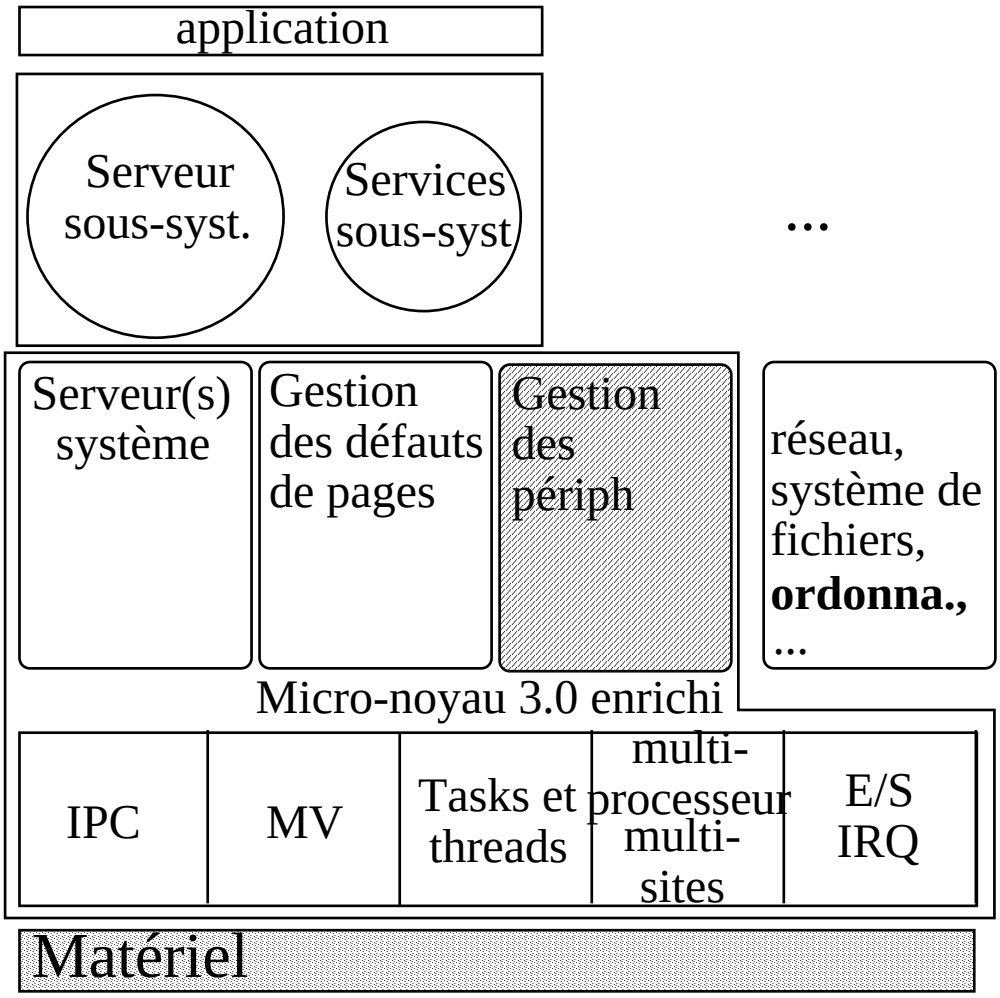
Génèse de Mach/OSF1 (suite) :



Mach / OSF1-AD

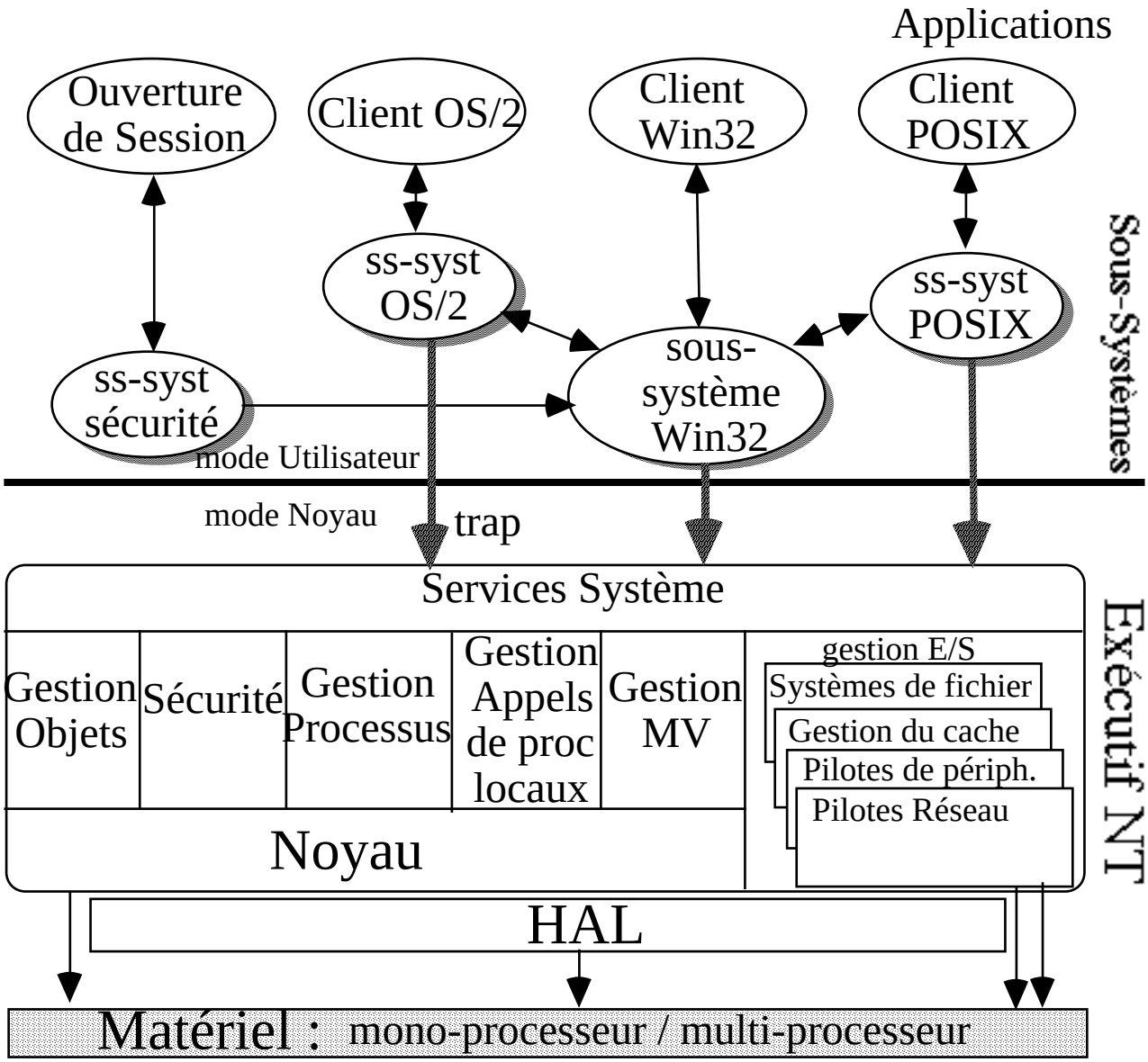
Advanced Development

Exemple : micro-noyau d'IBM pour Workplace OS (abandonné)



prévu pour supporter plusieurs personnalités

Exemple : l'Architecture de Windows NT

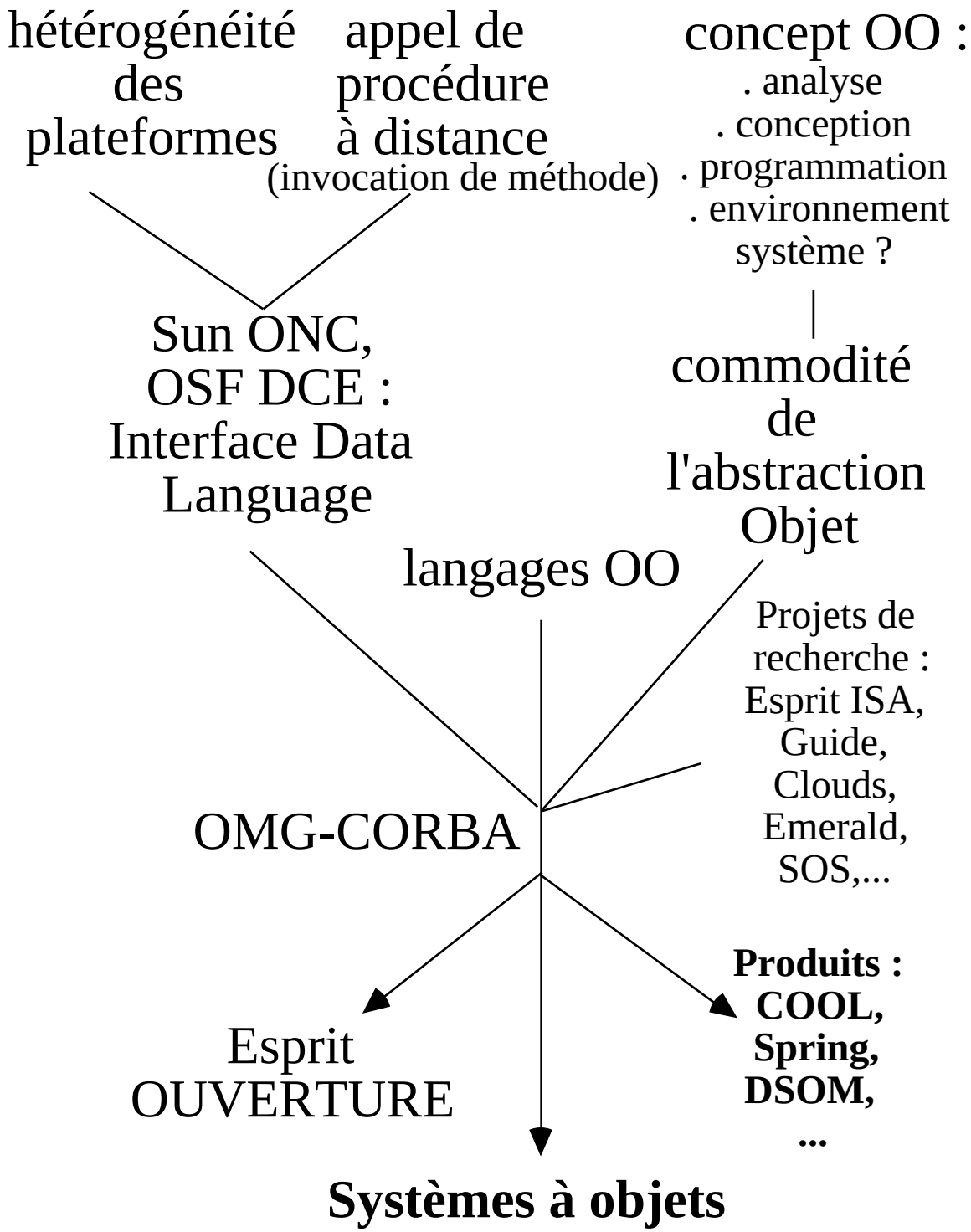


Prévu dès la conception pour fonctionner sur des processeurs différents : Mips, Alpha, Intel.

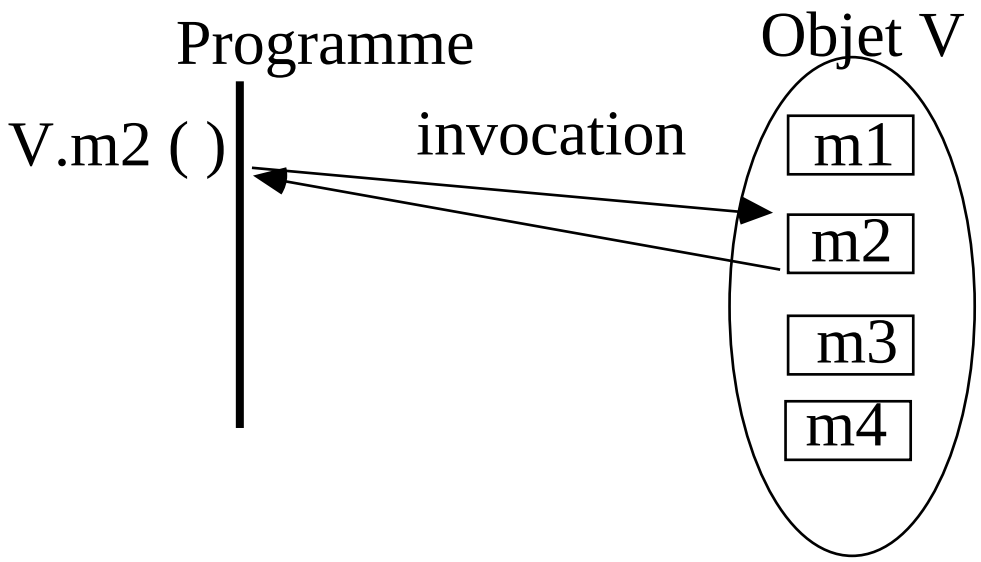
Windows NT bâti sur le modèle micro-noyau

Systèmes à objets

Objets


hétérogénéité des plateformes
appel de procédure à distance
(invocation de méthode)
concept OO :
. analyse
. conception
. programmation
. environnement système ?
Sun ONC, OSF DCE : Interface Data Language
commodité de l'abstraction Objet
langages OO
Projets de recherche : Esprit ISA, Guide, Clouds, Emerald, SOS,...
OMG-CORBA
Esprit OUVERTURE
Produits : COOL, Spring, DSOM, ...
Systèmes à objets

Invocation de méthode

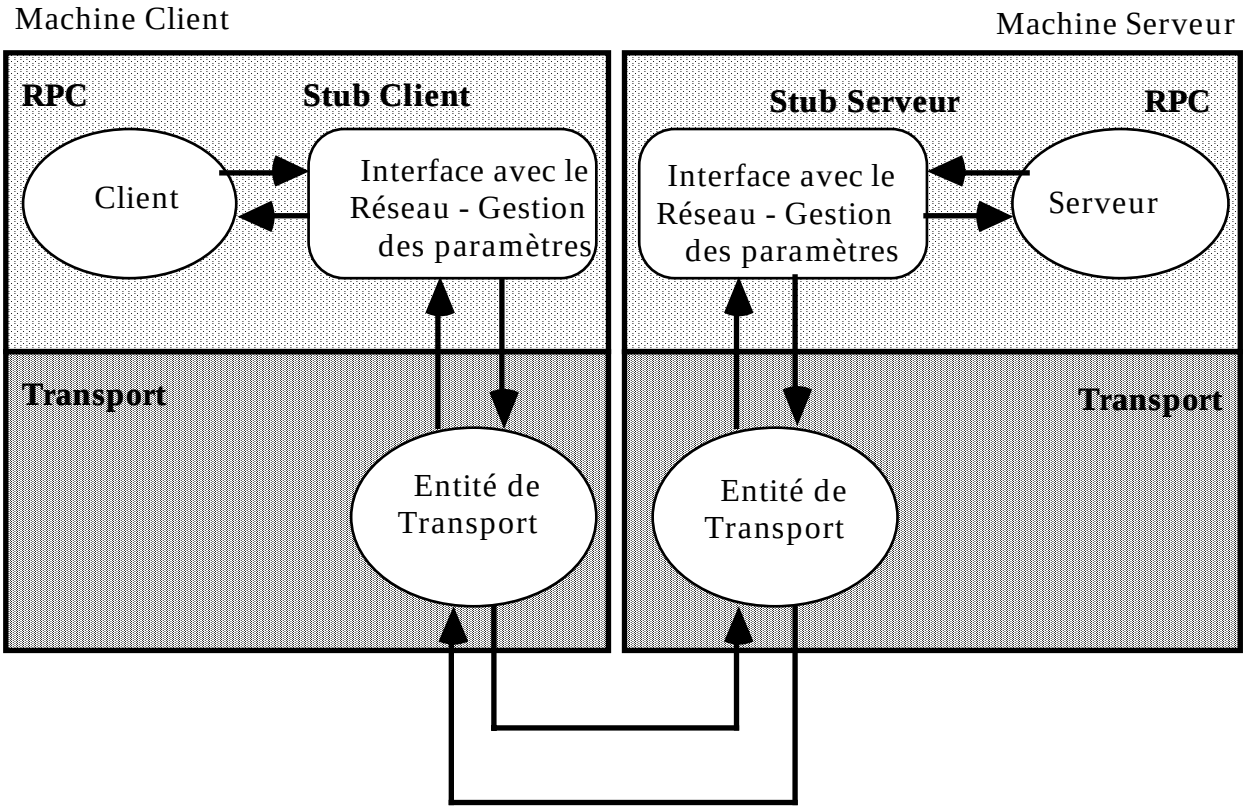


Objet actif : la méthode est exécutée par un programme gestionnaire dans le contexte d'un serveur, modèle d'interaction de type client-serveur

Objet passif : la méthode est exécutée dans le contexte de l'appelant

Appel de Procédure à Distance - RPC

En univers réparti, l'invocation se fait à distance
-> appel de procédure à distance



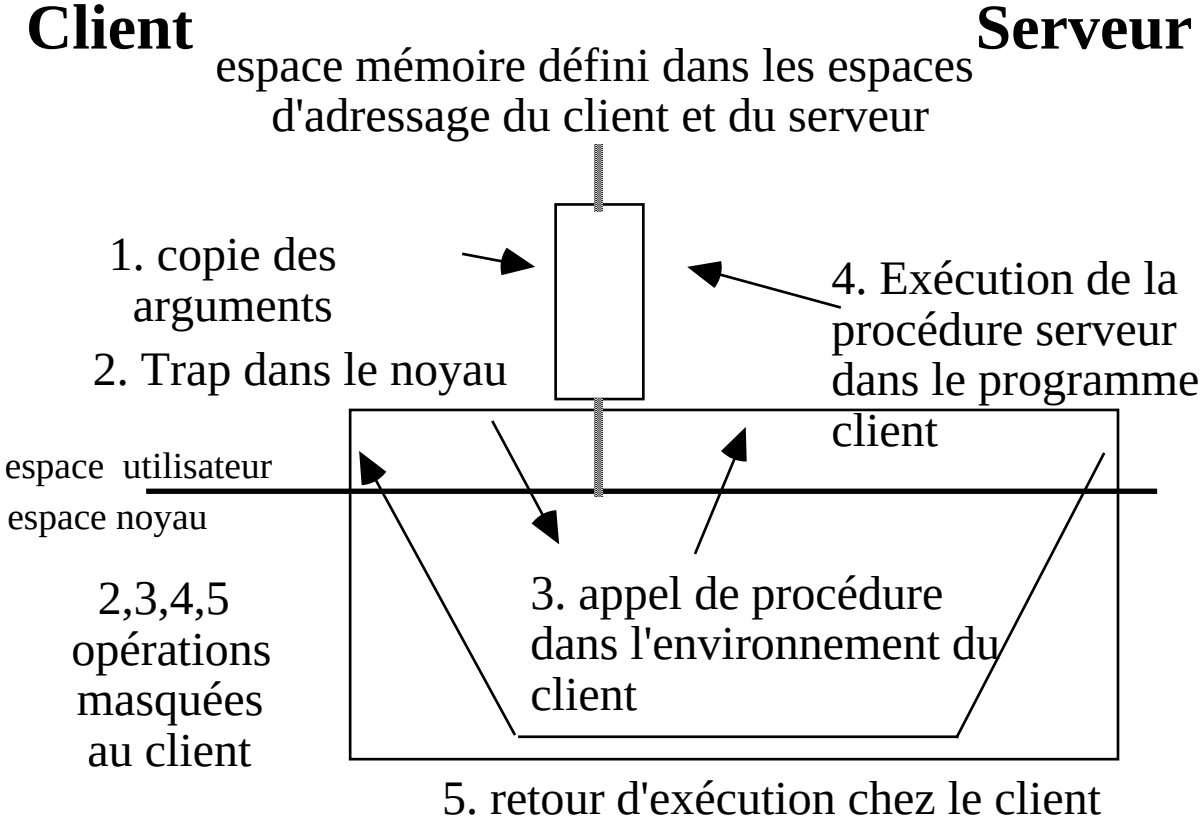
"Full Weight RPC"

appel de procédure à distance original, identique à celui définit par Birrell et Nelson

Appel de Procédure Local ("Light Weight RPC")

Certains appels ne partent pas sur le réseau :

il est possible de faire l'économie des phases d'encapsulation des données, le nombre de copies de buffers et de diminuer le nombre de changements de contexte lors de l'ordonnancement[4].



Enregistrement des procédures du serveur auprès d'une entité système "clerc"

[4] Le temps d'exécution serait diminué par 3 quand on compare les deux méthodes en local d'après Bershal (travail sur RPC Firefly).

Problèmes à résoudre dans les systèmes répartis à objets

Plus généralement il faut résoudre :

. désignation/référence

. contrôle d'accès

. granularité des containers d'objets : grain fin, grain moyen, gros grain

. persistance

. migration

. fragmentation

. partage des objets

. **récupération des objets qui ne sont plus référencés (glanage / ramasse-miettes / GC)**

Tendance pour les systèmes à objets répartis

en univers réparti se pose le problème de l'**interopérabilité**

-> prise en compte de l'**hétérogénéité** des produits/architectures qui permettent aux applications de coopérer

standards : OMG -CORBA

OMG Object Management Group
CORBA Common Object Request Broker Architecture

exemples de produits : DSOM[5] (IBM), COOL[6] (Chorus)

[5] Distributed System Object Model

OMG - CORBA

CORBA :

. IDL : Interface Data Language pour décrire les données impliquées dans les appels de procédure distant qui est compatible avec les langages de programmation, le C++ en particulier

. Un générateur de souche/stub

. Une architecture qui permet l'édition de liens dynamique entre clients et objets dont la méthode est invoquée
qui permet le mariage des concepts orientés objets et de l'hétérogénéité

---

[6] Chorus Object Oriented Layer

Systèmes à Objets et Service de Mémoire Répartie Partagée

L'utilisation d'une mémoire répartie partagée permet de revisiter l'architecture des Systèmes à Objets Répartis :

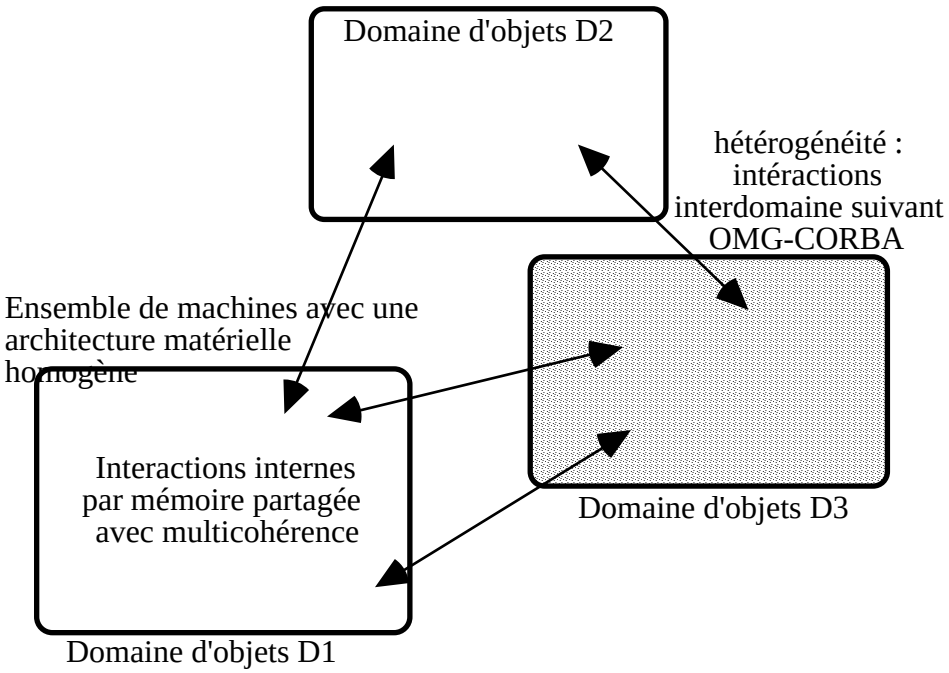
. Projet Larchant : INRIA/SOR

. Projet Arias : IMAG

. Projet Isatis : IRISA

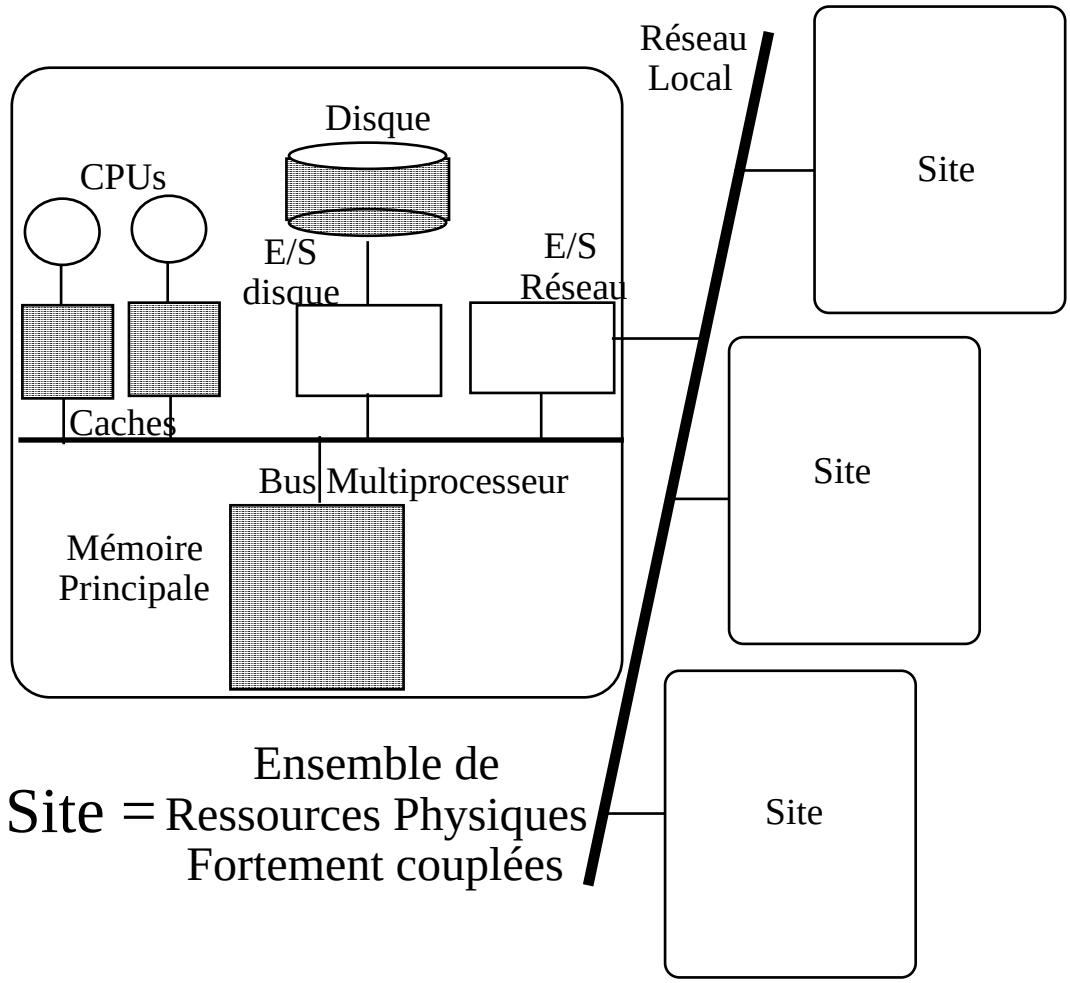
. Projet Saffres : CEDRIC

Plus difficile de gérer l'hétérogénéité :



ABSTRACTIONS DE BASE

Environnement



Ensemble de sites faiblement couplés

Objets Elémentaires

. Noms

. Acteur (Chorus) / Tâche (OSF1) /Processus (Amoeba)

. Activité / Thread / Processus Léger

. Message

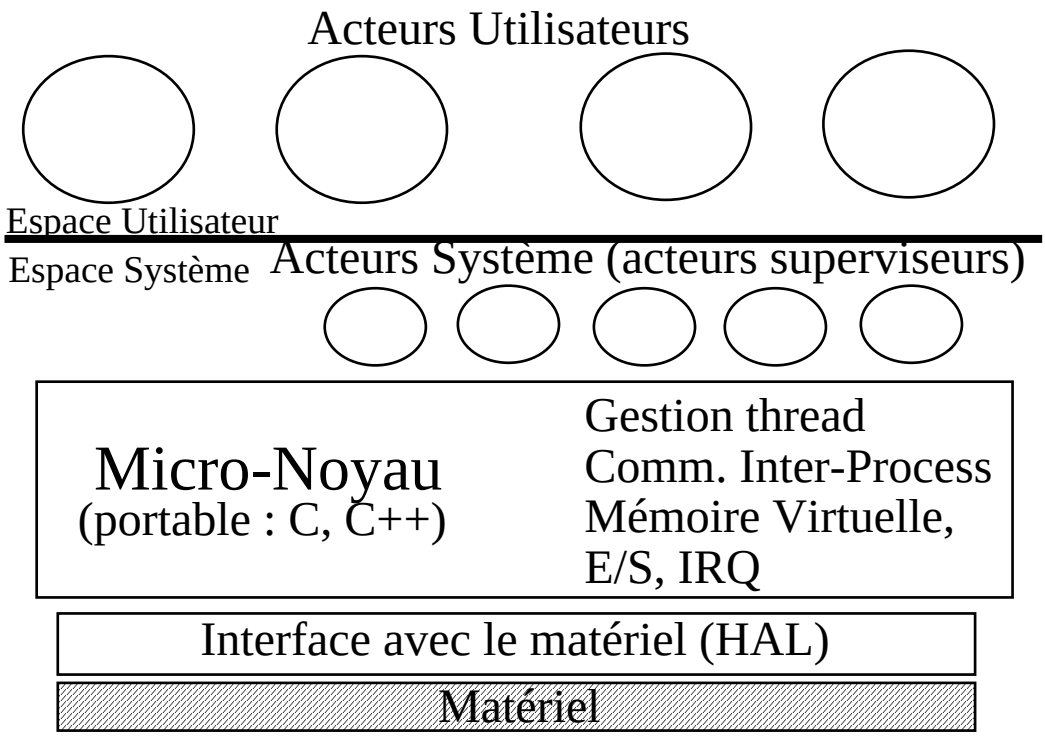
. Porte

. Segment

. Région

Ces abstractions sont combinées et enrichies par les sous-systèmes.

Architecture d'un Système Réparti



. Micro-noyau (minimal) sur toutes les machines

L'ordonnancement (stratégies) doit-il être dans le micro-noyau ? Nouvelle version du micro-noyau CHORUS (v3.6) sort une partie de l'ordonnancement !

. Acteurs Systèmes sur certaines machines en fonction des ressouces présentes et des services à rendre.

Noms (1)

Noms Internes

Noms Uniques et Globaux : **UID**

- Unicité dans le temps : date + nombre aléatoire

- Unicité dans l'espace : contient le nom du site de création de l'objet, parfois son type

permet de référencer les objets élémentaires **indépendamment de leur localisation**, (certains objets peuvent migrer)

Noms (2)

Capacité

Noms intermédiaires :

- connus hors du noyau

- gérés par un serveur

- n'ont de sens que dans le contexte du serveur associé

- liés aux droits d'accès à un objet

Composés :

- nom interne d'un serveur
- clef qui permet d'accèder à la ressource chez le serveur
- protection (droits d'accès)

concept principalement issu d'AMOEBA

Noms (3)

Nom contextuel ou Identificateur Local

Nom qui n'a de sens que par rapport à une entité système.

Equivalent au n° de fichier ouvert pour un processus Unix.

Exemple :

N° de Thread au sein d'un acteur CHORUS

Acteurs/Tâches/Processus (1)

Unité de Structuration =
Acteur/Tâche/Processus

Espace d'allocation de ressources :

. Mémoire : Espace d'adressage (protégé)

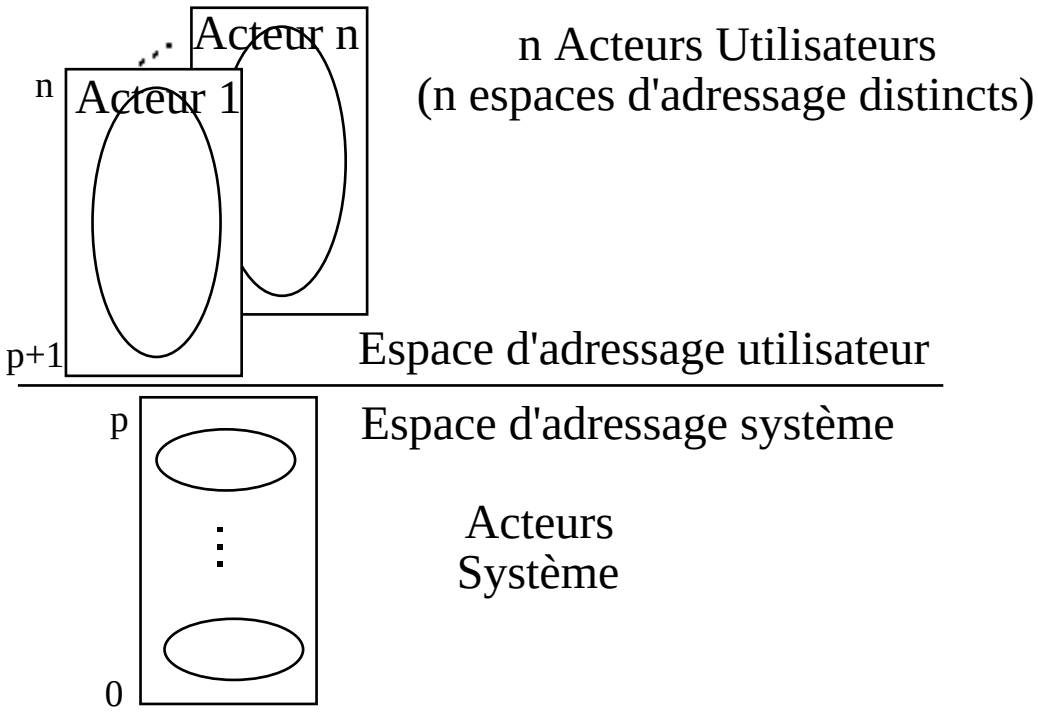
. Ressources système : ports, sémaphores, canaux d'E/S, ...

. Activités

On retrouve le modèle UNIX du processus mais affiné

Pour Chorus, un acteur est une machine virtuelle.

Acteurs/Tâches/Processus (2)



Thread

Unité d'exécution =
Thread/Processus Léger/Activité

. Son exécution a un caractère **Séquentiel**

. Contexte d'exécution :
- état
- compteur ordinal
- registres
- pile d'exécution
- descripteur

. Lié à un acteur

. Partage l'espace d'adressage et les ressources d'un acteur avec d'autres threads

Threads

Ordonnancement

Thread = Unité d'ordonnancement indépendante pour le noyau

Parallèlisme

pseudo-parallèlisme
ou
parallèlisme réel sur multiprocesseur[7]

Que se passe-t-il de différent entre les deux situations suivantes?

. commutation de contexte entre deux threads d'un même acteur

. commutation de contexte entre deux threads d'un acteur différent

---

[7] **Symétrique** : les processeurs ne sont pas distingués, ils ont tous les mêmes capacités d'exécution,
Asymétrique : les processeurs sont distingués, certains ont des capacités différentes, un co-processeur arithmétique par exemple, et certaines threads doivent s'exécuter exclusivement sur ces processeurs

Acteurs Chorus (1)

Création *actorCreate* **:** Comme lors d'un fork Unix, l'acteur créé (fils) peut hériter d'un certain nombre d'attributs de l'acteur créateur (père).

Acteur est créé avec un minimum de ressources:
- une porte
- pas d'activité
- possibilité d'hériter d'une partie de l'espace mémoire de l'acteur père (régions)

Désignation d'un acteur : Un acteur est désigné par une capacité. *actorSelf()* fournit la capacité de l'acteur courant.

Modification: Un acteur qui reçoit l'identification d'un autre peut intervenir sur celui-ci, et peut modifier sa structure :
- création/destruction d'activités,
- modification de l'espace d'adressage.

Un acteur ne migre pas d'un site à un autre.

Acteurs Chorus (2)

Deux types d'acteurs :

- Utilisateurs :

. Utilisateur sans aucun privilège

. Serveur Système accède à certaines primitives du système (équivalent au statut d'un "daemon" Unix)

- Superviseurs : leurs activités s'exécutent en mode système dans l'espace d'adresse du noyau

=> pb graves en cas d'erreur de programmation

Threads Chorus (1)

Thread :

. Liée à un et un seul acteur

. Une thread est créée par une thread éventuellement dans un acteur différent

. Sur un multiprocesseur, les threads d'un même acteur peuvent s'exécuter en même temps de façon concurrente

. Unité d'ordonnancement considérée par le noyau

Threads Chorus (2)

Thread :

. **Classe d'ordonnancement**

. Création : même classe que le créateur

. Possibilité de changer de classe d'ordonnancement par *threadScheduler()*

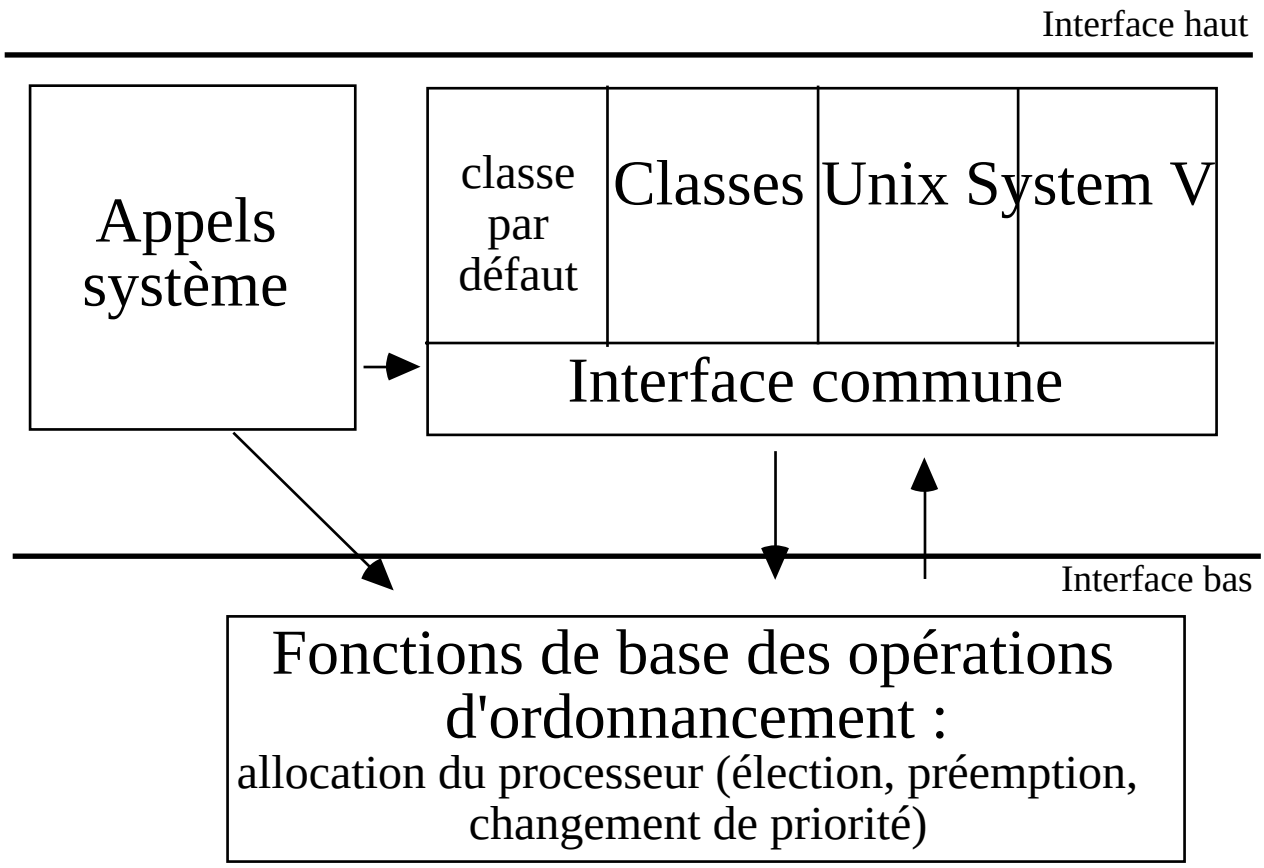
. Priorité : la priorité d'une thread n'a de sens qu'au sein d'une classe d'ordonnancement

. Une partie du contexte du thread est fournie à la création :

- compteur ordinal
- le pointeur de pile (mais pile elle même non allouée)
- le mode d'exécution (système ou utilisateur)

Ordonnancement Chorus (1)

Structure de l'ordonnanceur :

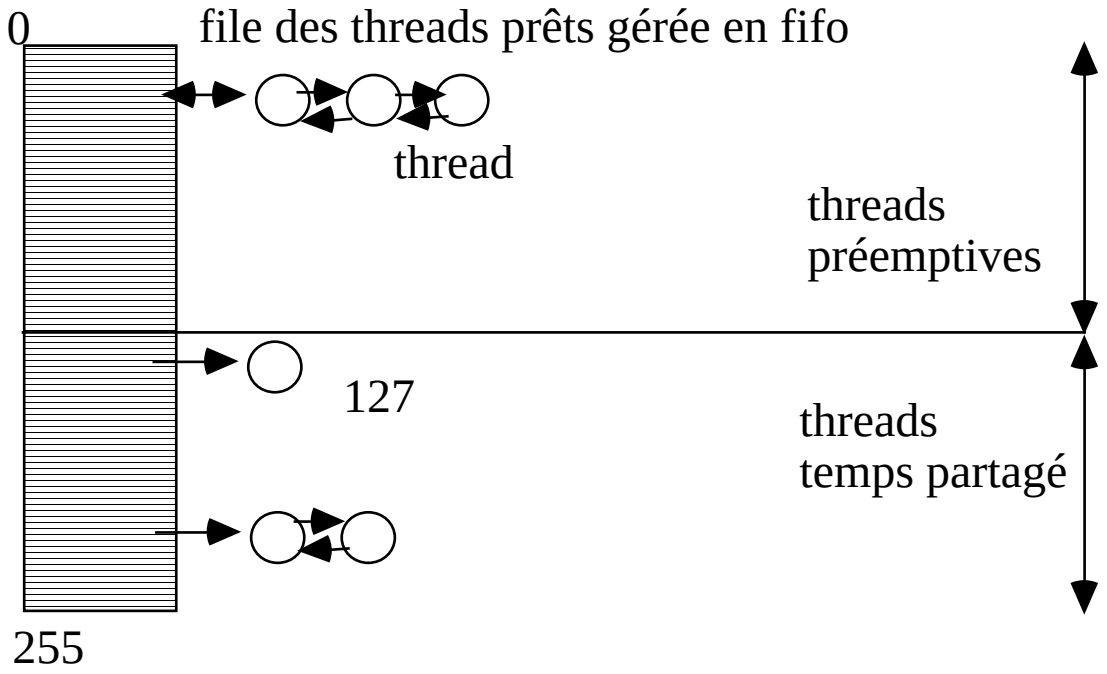


Ordonnancement Chorus (2)

Le noyau ordonnance les threads suivant une priorité à valeur numérique, ***priorité noyau*** :

classe par défaut :

0 plus forte priorité
255 plus faible priorité



Sur un multiprocesseur symétrique à mémoire partagée de N processeurs, le noyau garantit que les N threads élues sont celles qui sont prêtes et de plus forte priorité.

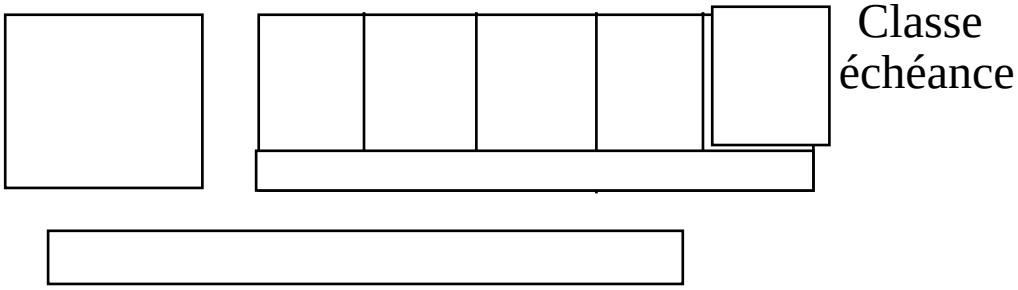
Le noyau est préemptif => **temps réel**

Classes d'ordonnancement

Modularité :

Pouvoir implanter simultanément plusieurs politiques d'ordonnancement

-> une classe par défaut
-> une classe pour le sous-système Unix
-> autre : une classe à échéance par exemple :



travail O. Gaultier et O. Métais

Classe par défaut :

. priorité relative : thread relative à l'acteur
. priorité absolue : thread + acteur

=> priorité noyau

Coopération entre Threads

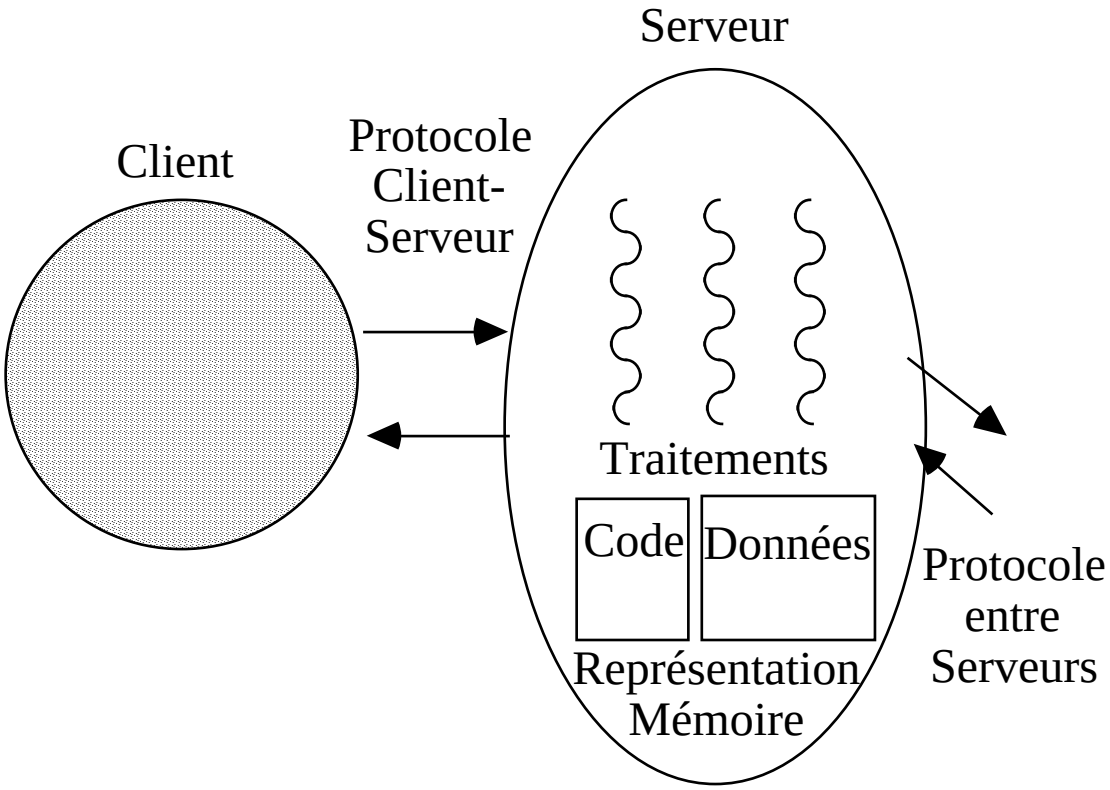
Entre threads d'un même acteur

=> par partage de mémoire

Entre acteurs
=> par messages, en utilisant des portes

Communication Client-Serveur en local

Threads => possibilité de traiter plusieurs requêtes simultanément chez un serveur



Messages

Unité d'échange entre acteurs = Message

Suite d'octets

Non typé dans Chorus, Amoeba, Windows NT

Typé dans Mach en fonction de la nature des données qu'il contient
(plusieurs données dans un message => plusieurs types[8] coexistent dans un message)

[8] Il ne faut pas entendre type au sens du typage des données dans les langages de programmation, d'ASN1 ou de XDR, mais plutôt nature des informations pour la gestion du système.

Portes

Unité d'adressage = Porte

Porte :

- Destination d'un message

- File de messages

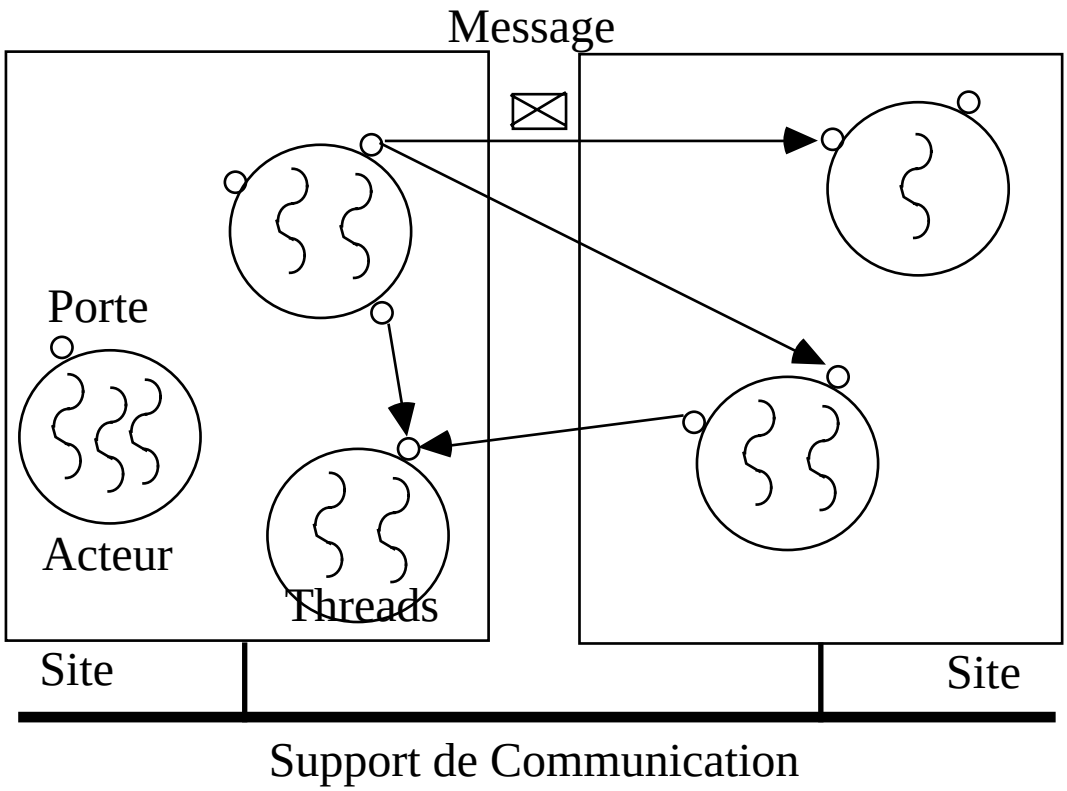
. Droits d'utilisation : [Emission[9]], Réception

. Porte liée à un acteur : seules les activités de cet acteur possèdent le droit de réception
. Droit de réception transmissible (migration de porte vers un autre acteur)

. Droit d'émission transmissible et partageable

[9] Portes unidirectionnelles dans Mach.

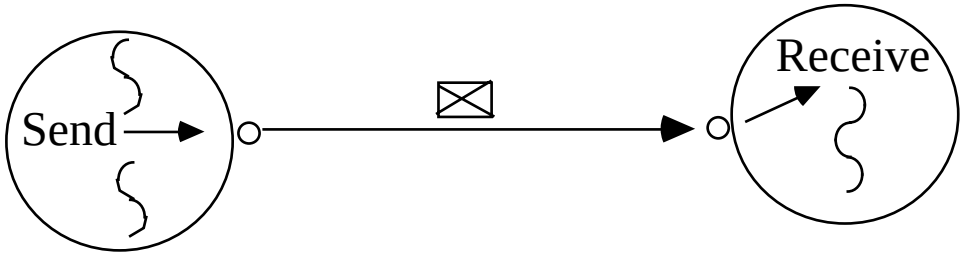
IPC - Communication entre acteurs



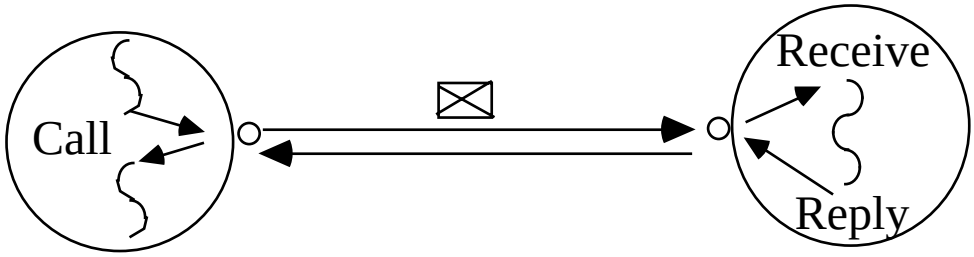
IPC = Inter-Process Communication

IPC

IPC non bloquant/asynchrone :



IPC synchrone :



IPC sans threads dans CHORUS

Programmation par **interruption logicielle** (mécanisme de"message handler") :

On peut associer un traitement à une porte. On connecte un service à une porte.

Dans ce cas, toute réception de message provoque l'exécution du traitement :

- l'ipc est local (LRPC), l'appelant exécute le traitement chez lui

- l'ipc est distant vers un serveur (FRPC), le noyau du site distant dispose d'un "pool de threads", une de ces threads est utilisée quand le serveur est sollicité pour traiter une demande.

Ce mode de traitement est restreint aux acteurs superviseurs et est incompatible avec le mode normal.

Diffusion sur groupe dans Chorus

Groupe de portes

Un groupe de portes est désigné par une capacité

2 modes d'adressage pour l'émission des messages :

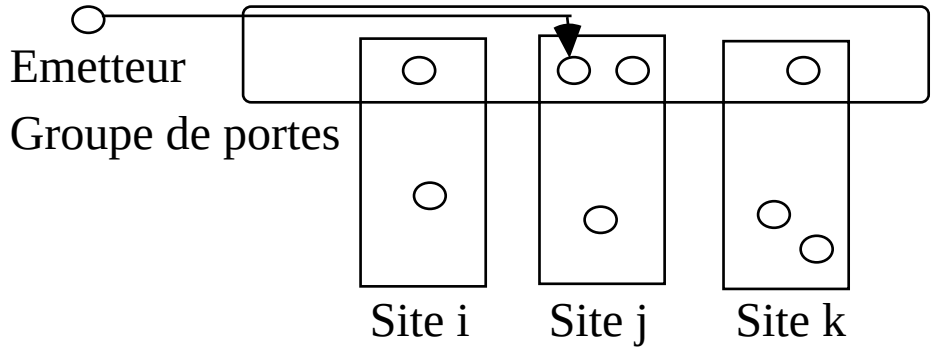
- Diffusion : toutes les portes d'un groupe reçoivent le message

- Fonctionnel : une seule porte reçoit le message

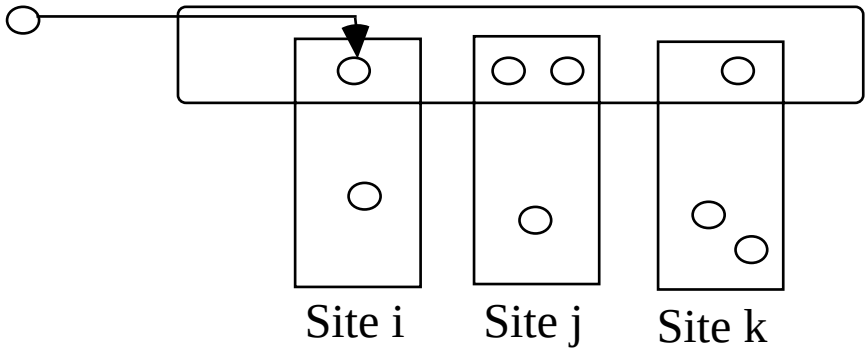
L'adressage fonctionnel peut être avec ou sans sélection du destinataire. Cette sélection est effectuée par l'émetteur sous la forme

Adressage Fonctionnel dans Chorus

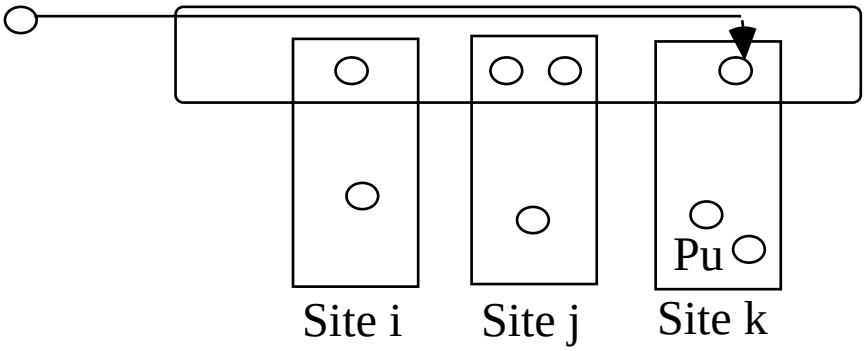
- Adressage Fonctionnel sans sélection : une dans le groupe (idem pour Mach)



- Adressage Fonctionnel avec sélection sur site i



- Adressage Fonctionnel avec sélection sur même site que porte Pu (Pu peut ne pas appartenir au groupe)



Protocoles

Communications locales

=> éviter les recopies (RPC Léger)

Communication à l'intérieur d'un domaine

- une première couche au dessus du réseau avec un protocole de type IP (mode datagramme)

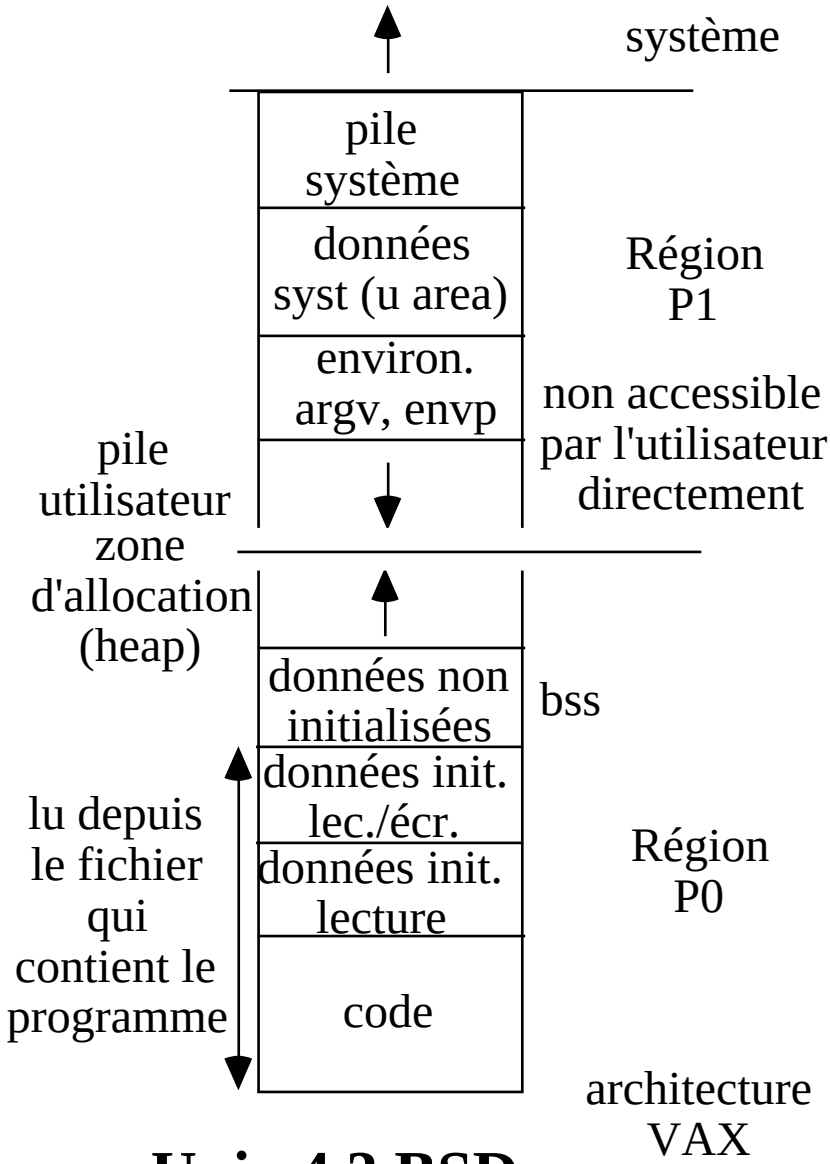
- une seconde couche qui joue le rôle d'une couche transport ou session

Communications extérieures

=> serveurs spécialisés qui sont équipés de piles de communication plus complexes : TCP-IP, OSI, ...

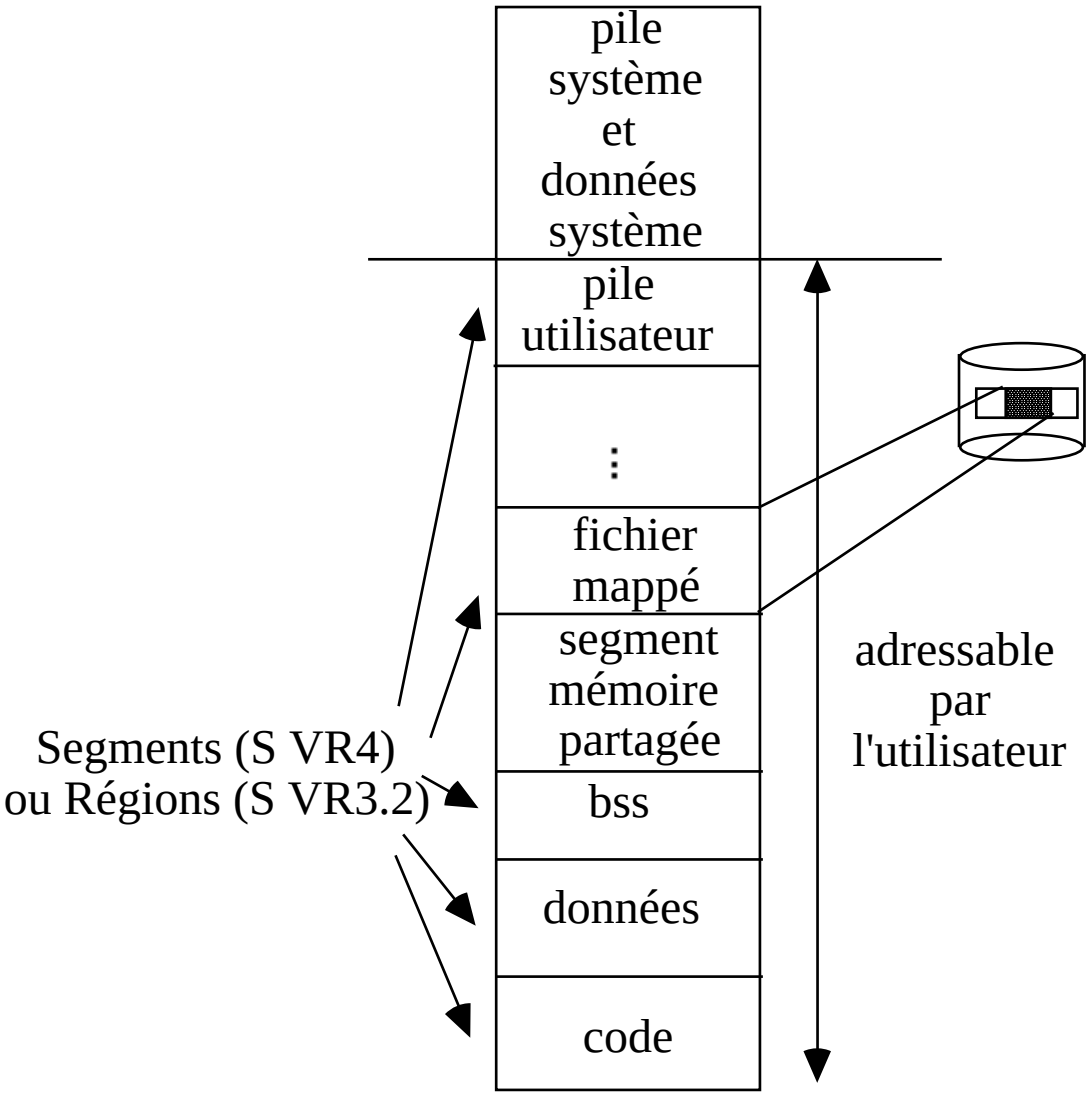
Evolution du Modèle d'Espace d'adressage (1)

4 Go d'espace mémoire : adressage 32 bits (dont 1 gaché puisque non utilisé)



Unix 4.3 BSD

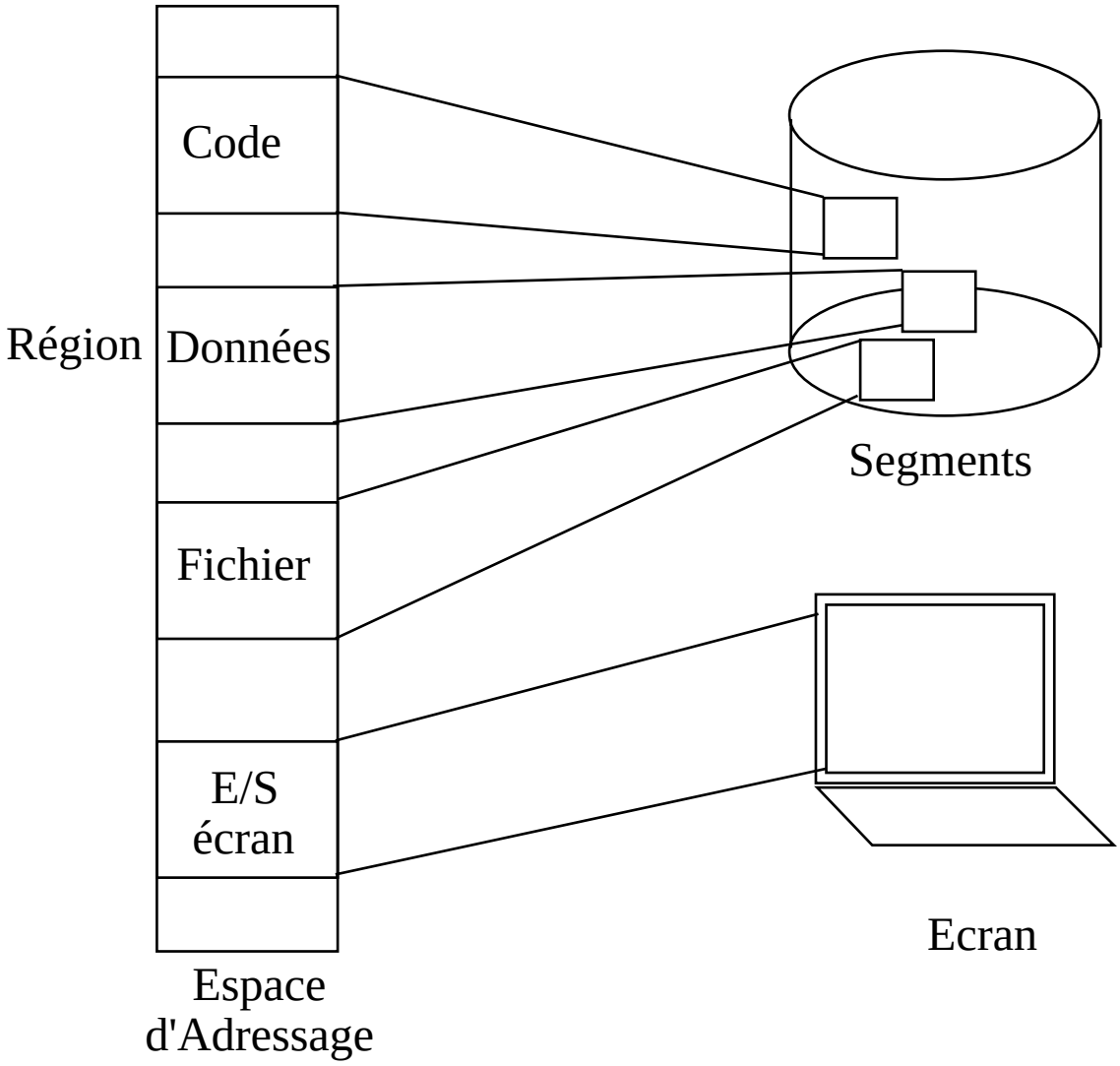
Evolution du Modèle d'Espace d'adressage (2)



Unix System V R4

Objets mémoire

Unité de représentation des Données = **Segment**
Unité d'accès aux données en mémoire = **Région**



Segments et Régions

Segment :

- Données permanentes (fichier) ou temporaires (espace swap)

- Géré par serveur externe au noyau (mappeur)

- Identifié par une capacité

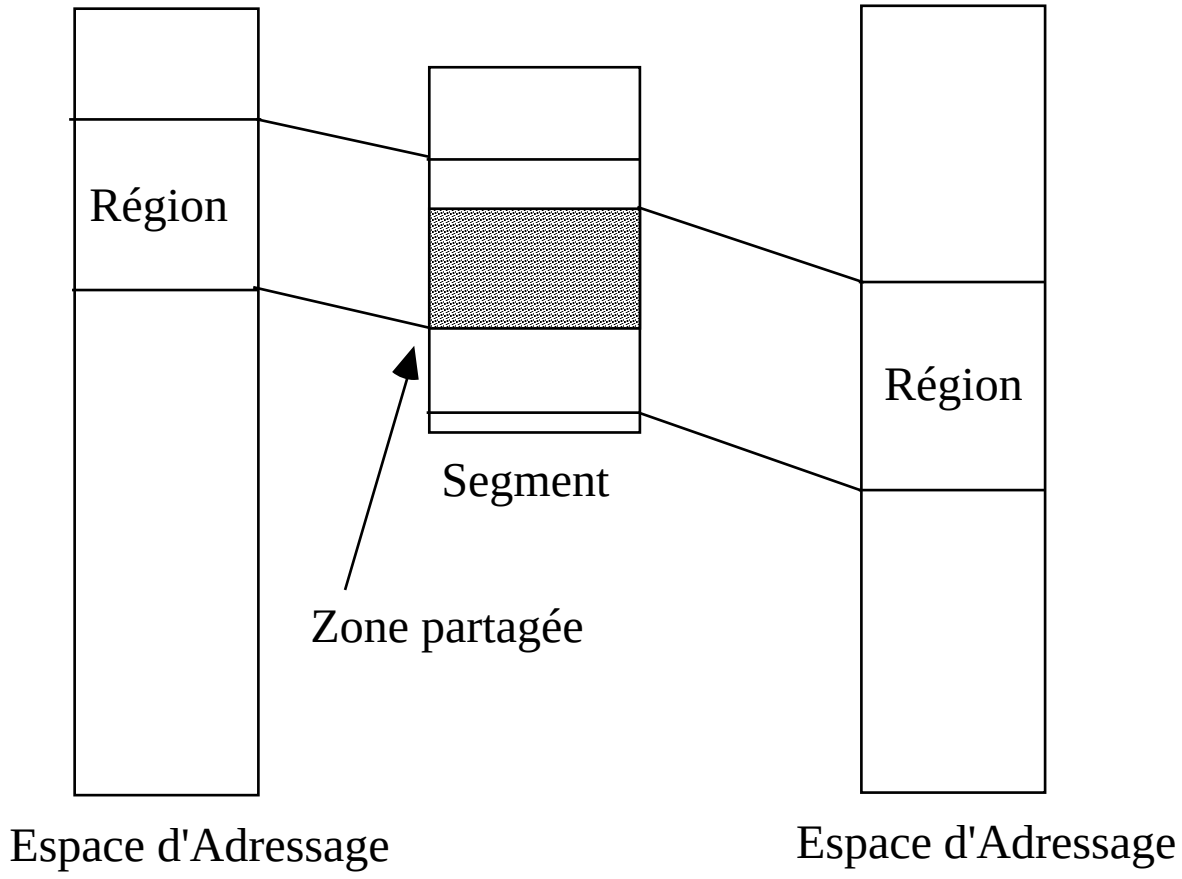
Région :

- L'espace d'adressage d'un acteur est divisé en plusieurs régions

- Partie visible en mémoire d'un segment : mise en correspondance d'un segment dans l'espace d'adressage d'un acteur

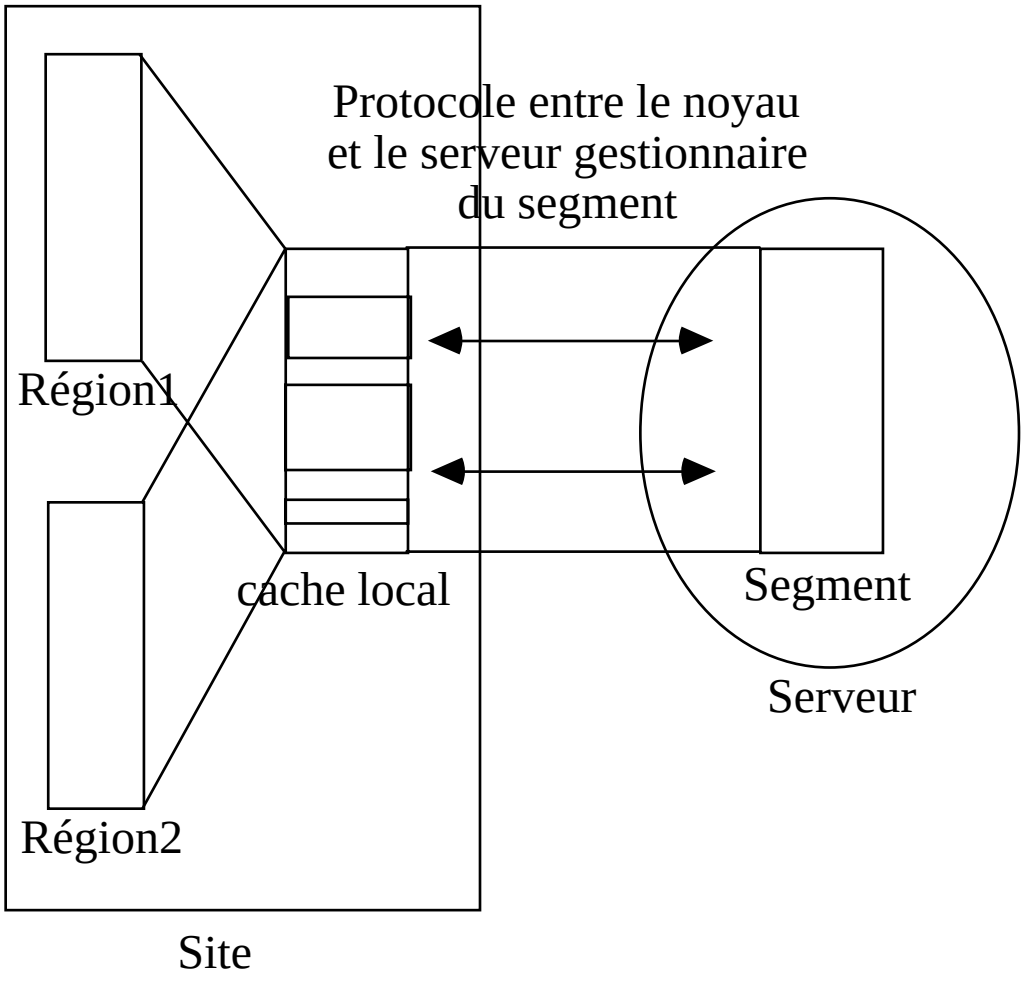
Partage

Partage d'un segment entre acteurs par région



Cache des segments

Le noyau peut gérer un cache des pages virtuelles associées au segment.



-> Pas plus d'un cache par segment et par site
-> Gérer la cohérence du segment entre sites

CHORUS

. Projet INRIA 1981 - 1986 (V1, V2)

. Chorus Systèmes 1987 - (V3)

. Micro-Noyau

. Unix Sous-système : System V R4, 4.3BSD, SCO

. Sous-système orienté objet : COOL

. Architectures faiblement couplées

. Architectures fortement couplées à processeur symétrique

Références Bibliographiques

[1] Construction des systèmes d'exploitation répartis. Editeurs : R. Balter, J-P Banâtre, S. Krakowiak. Inria. Collection Didactique. 1991

[2] Supports de cours : Technologie des micro-noyaux.. Marc Rozier. Ecole d'Eté d'Autran. Septembre 1993.

[3] Distributed Operating Systems. Andrew S. Tannenbaum. Prentice Hall. 1994.

[4] Supporting an Object-oriented distributed system : experience with Unix, Mach and Chorus. Boyer, J. Cayuela, P-Y. Chevalier, A. Freyssinet, D. Hagimont. Rapport Technique Bull-Imag. N 7-90. Décembre 1990.

[5] Inside Windows NT. Helen Custer. Microsoft Press. 1992.

[6] Mach 3 Kernel Principles. Keith Loepere. Open software Foundation and Carnegie Mellon University. NORMA-MK12: July 15, 1992.

[7] Chorus/Mix V.4 : Programmers Guide & Implementation Guide. Chorus Système. 1992.

[8] Chorus Kernel V3 : Programmers Guide & Implementation Guide. Chorus Système. 1993.

[9] Distributed Systems : concepts and Design. George Coulouris, Jean Dollimore, Tim Kindberg. Addison Wesley 1994.

[10] Chorus Jam Session 1994. Proceedings. Septembre 1994.

[11] Advanced Operating Systems. Dossier Spécial. V19 N1. Byte. January 1994.

[12] Projet MaX : Construction d'un système d'exploitation basé sur le micro-noyau Mach. Daniel Azuelos. Mémoire d'ingénieur Cnam. Juin 1994.

[13] Using CHORUS to Develop Real Time Applications. Chorus System. Chorus Real Time Tutorial. Juillet 1994.

[14] Mise en place d'une plate-forme Chorus, Conception et Implantation d'un Ordonnanceur à échéance au sein du noyau Chorus. O. Gaultier, O. Métais. Mémoires d'ingénieur Cnam. Mars 1994.

[15] Distributed Systems : concepts and Design. George Coulouris, Jean Dollimore, Tim Kindberg. Addison Wesley 1994.