Etude des Micro-noyaux

Plan

- Présentation générale des micro-noyaux
- Mach
- Chorus
- Amoeba, V system
- Comparaison des différentes approches

Les architectures de Noyau

- Monolithique
- Extensible
- **Micro-noyau**
- **Exo-noyau**

Introduction Micro-Noyau

Objectifs :

Portabilité

Informatique répartie et coopérative

Environnement facilitant l'intégration de nouvelles fonctions

Principe :

Terme micro-noyau introduit par Ira Galdstein de l'OSF (Open Software Foundation)

Issue de travaux sur les systemes répartis des années 80

Division du système d'exploitation en deux parties :

1°) Le micro-noyau

2°) Un ensemble de modules serveur

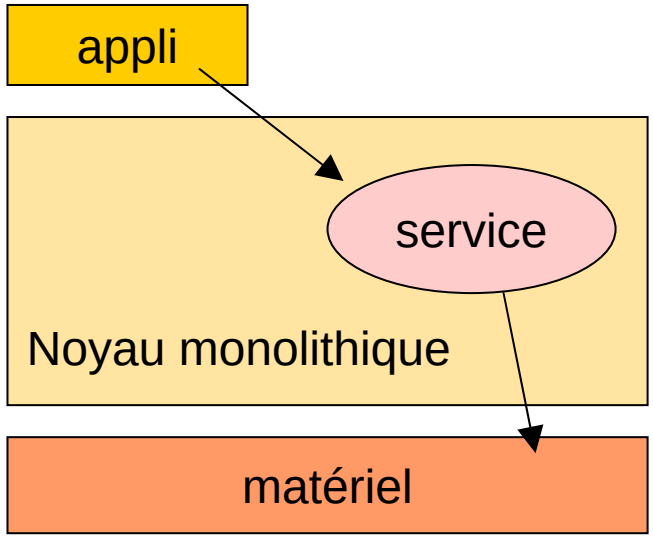
Historique :

Carnegie Mellon (1980) isole les fonctions élémentaires du système

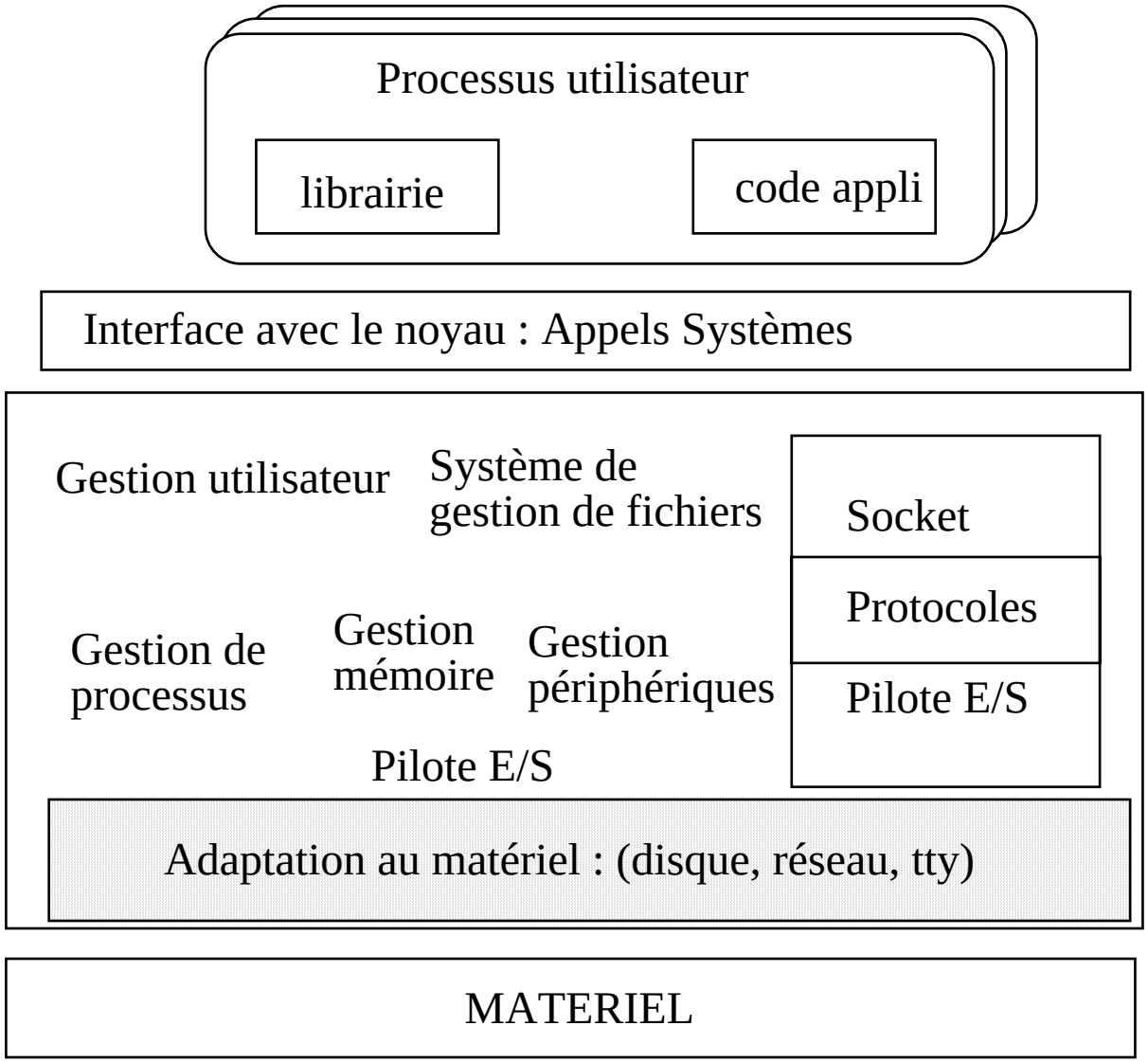
A partir de 1979 l'INRIA implémente des fonctions principales sous forme de modules indépendants.

Noyau monolithiques

- 1eres générations d'Unix, Linux, AIX…
- Performant et relativement sécurisé.
- Peu extensible, maintenance délicate.

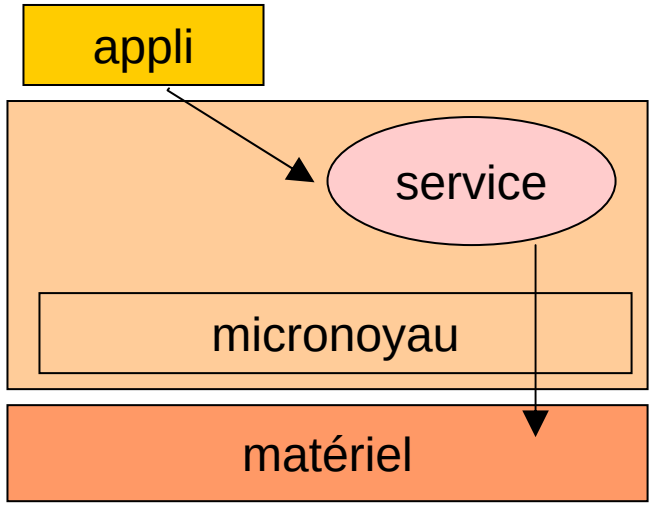


Système Monolithique



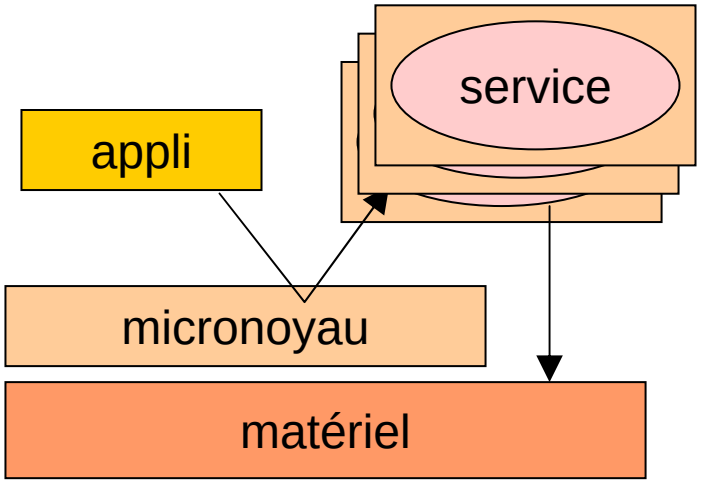
Noyau extensible

- Linux, AIX, Solaris…
- Chargement dynamique de code dans le noyau (module)
- Manque de sécurité



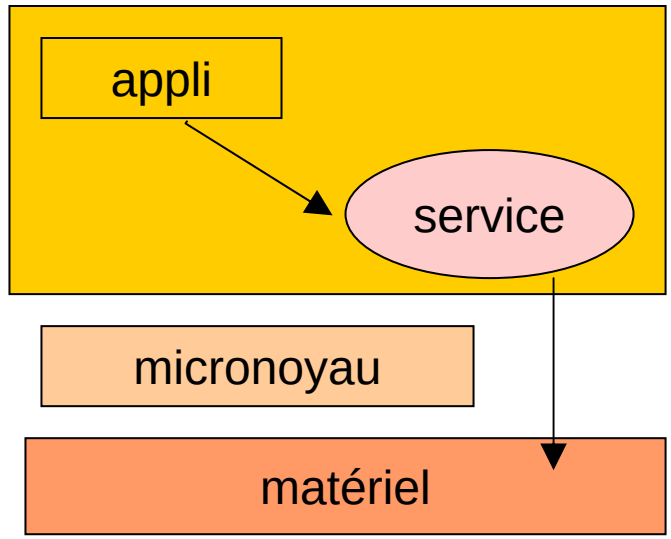
Micro-noyau

- Mach(BSD, MacOSX), GNU-HURD
- Très sécurisé
- Peu performant (Génération actuelles optimisées)



Exo-noyau

- Exokernel
- Performance Extensible
- Perte de contrôle des ressources par le noyau
- Difficile à mettre en oeuvre



Comparaison Monolithique/ Micro-noyau

Monolithique

Principes :
Un seul programme constitue le système

Avantages :
bonnes performances
(partage de données dans le noyau)

Inconvénients :
programme gisgantesque
difficile à maintenir et à faire évoluer

Exemples :
Locus, Sprite, Unix

Micro-noyau

Principes :
Un noyau minimal fournit des fonctionnalités de bas niveau
Les fonctionnalités du système réalisées par un ou plusieurs serveurs au-dessus du micro-noyau

Avantages :
Facilité de mise au point
Evolution facile
Modèle client-serveur adapté aux systèmes répartis

Inconvénients :
Performances

Exemples :
Amoeba, Mach, Chorus, V Kernel

Historique

Dans les années 80 : Recherche

Grapevine (Xerox)

Accent (CMU)

Amoeba (Amsterdam 84)

Chorus (Inria) 79

Mach (CMU) 86

V-System (Stanford) 83

Sprite (Berkeley)

Fin 80 debut 90 : Systèmes commerciaux

Chorus Systèmes 86

Mach-OSF-1

Amoeba (ACE)

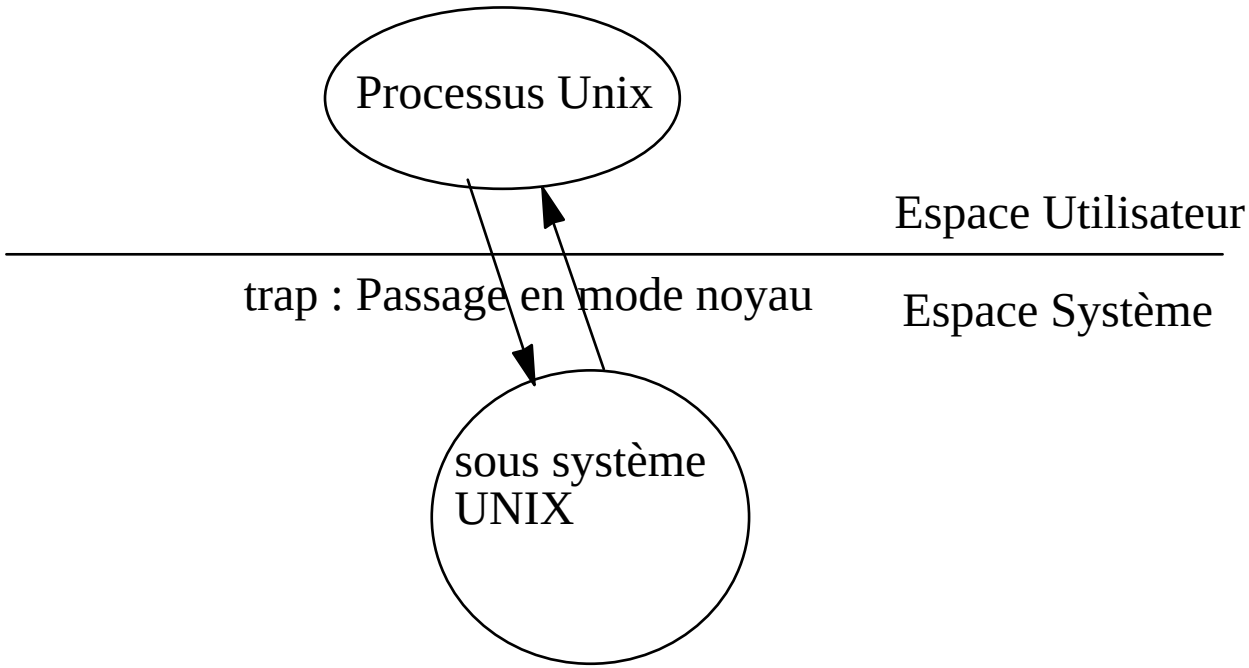
Windows NT (Microsoft)

Organisation

. Micro-noyau (base logicielle)

Serveurs
- de type système (dans l'espace du système)
- de type utilisateur

Sous systèmes : machines virtuelles (assuser la compatibilité binaire)



Conception de systèmes au-dessus de micro-noyaux

2 approches d'implémentation :

Un serveur unique

Avantage : Implémentation rapide à partir de l'existant (portage),
Inconvenient : Pas modulaire => non extensible

Multi-serveurs

Avantages : Modularité, extensibilité, mise au point
Inconvénients : Reconception totale,
Difficulté pour connaitre l'état global,
Communication entre les serveurs,
Performances

Le micro-noyau Mach

Objectifs :

Base pour la conception de systèmes d'exploitation

Support d'espace d'adressage

Accès transparent aux ressources distantes

Exploitation maximal du parallèlisme

Portabilité

Fonctionnalités de Mach

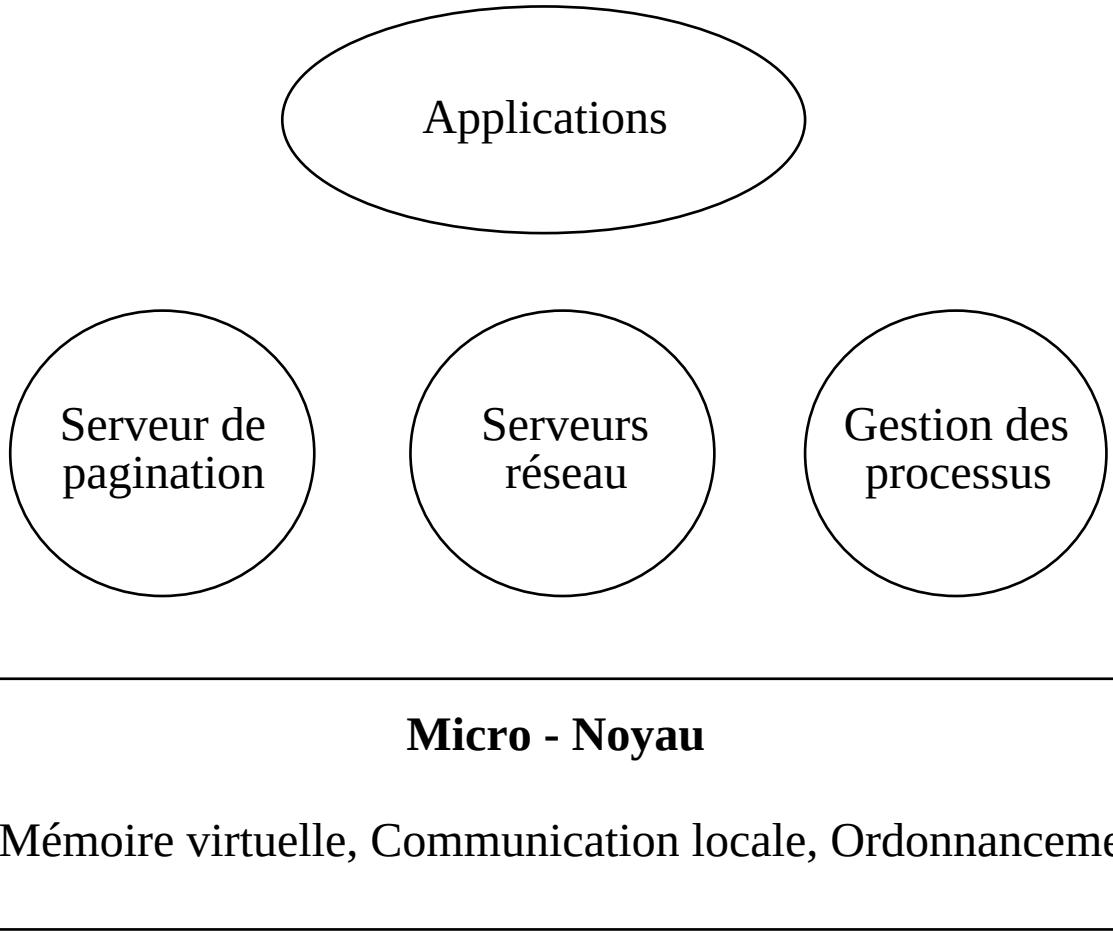
Mach fournit :

- Gestion de tâches et d'activités (thread)
- Communication entre tâches
- Gestion de la mémoire physique et virtuelle
- Gestion des périphériques physiques

Mach ne fournit pas :

- Gestion d'un système de fichier hiérarchique
- Gestion de processus
- Gestion de terminaux
- Chargement et écriture de pages
- Gestion du réseau

Architecture du Système Mach



Abstraction de Mach

Abstractions de base :

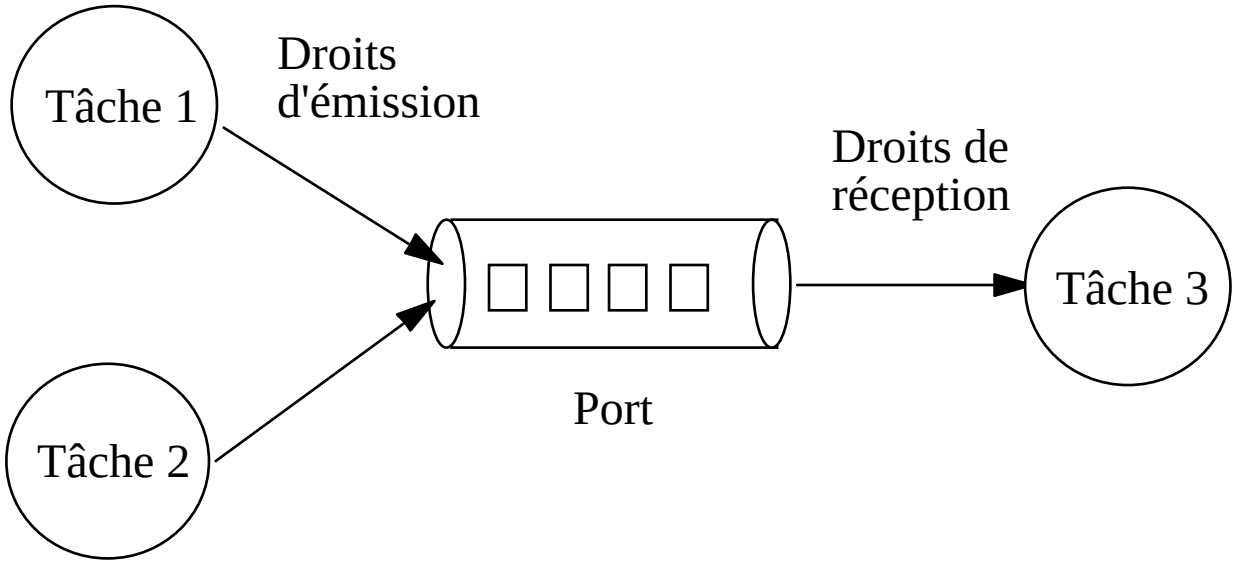
- Tâches (Tasks)
  Environnement d'exécution des activités (espace d'adressage)
- Activités (Thread)
  Unité d'exécution pour l'ordonnancement
- Ports
  Canal de communication (file de messages)
- Messages
  données typées

Abstractions secondaires :

- Ensemble de ports (Port Sets)

Communication : Les ports

- Un canal de communication
- Communication unidirectionnelle
- Un récepteur
  Un ou plusieurs émetteurs
- Droits d'accès transmissibles : en émission, en réception



Création : *Port_allocate*

Primitives de communication

Communication Asynchrone :

msg_send(msg, option, timeout)
msg_receive(msg, option, timeout)

Communication synchrone :

msg_rpc(message, option, ...)

Les options d'envoi (nombreuses) :

Specifier l'action si la file est pleine:
(attendre indéfiniment, délai, pas d'attente, transmettre le message à Mach)

Droits sur les ports

Accès à un port via un droit :

droit de réception
droit d'émission
droit d'émission unique (send_one) utilisé pour les RPC

Gérés par le noyau :

Table interne à Mach

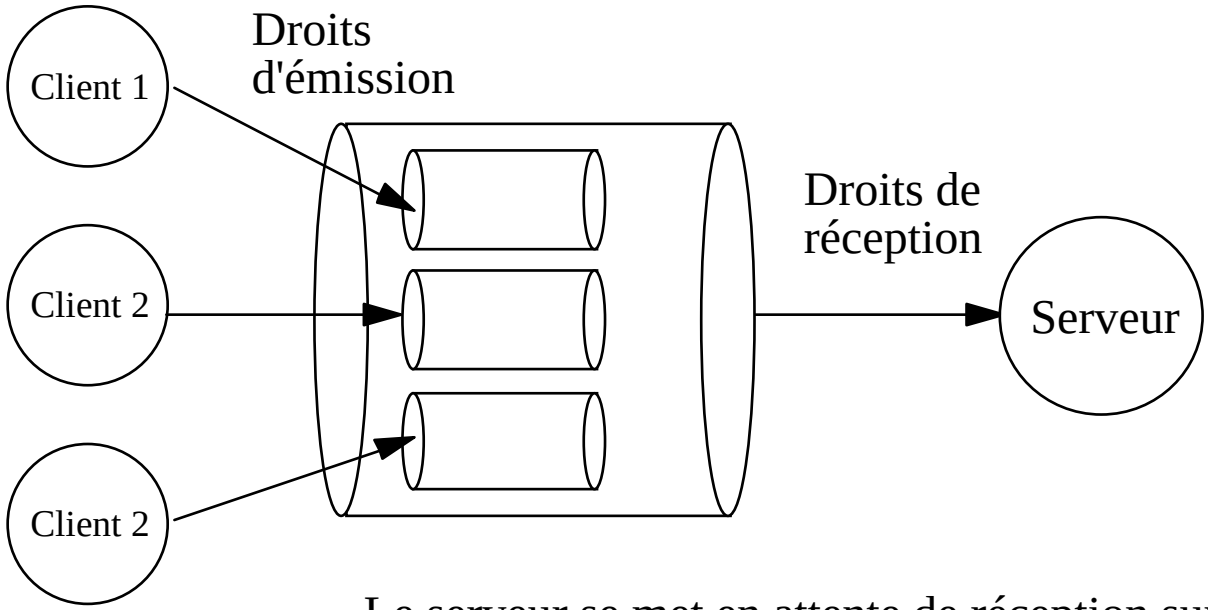
Les droits sur les ports sont transmis par msg_send

Transmission sur les droits d'émission :
l'émetteur et le récepteur ont les droits

Transmission sur les droits de réception :
l'émetteur perd les droits et le récepteur les récupère

Ensembles de ports

Regroupement de ports : Permet de définir plusieurs points d'entrée à un serveur



Le serveur se met en attente de réception sur un des port
(analogie avec le Select des sockets Bsd)

Opérations :

port_set_allocate, port_set_add, port_set_remove

Un outils : Le Mach Interface Generator (MIG)

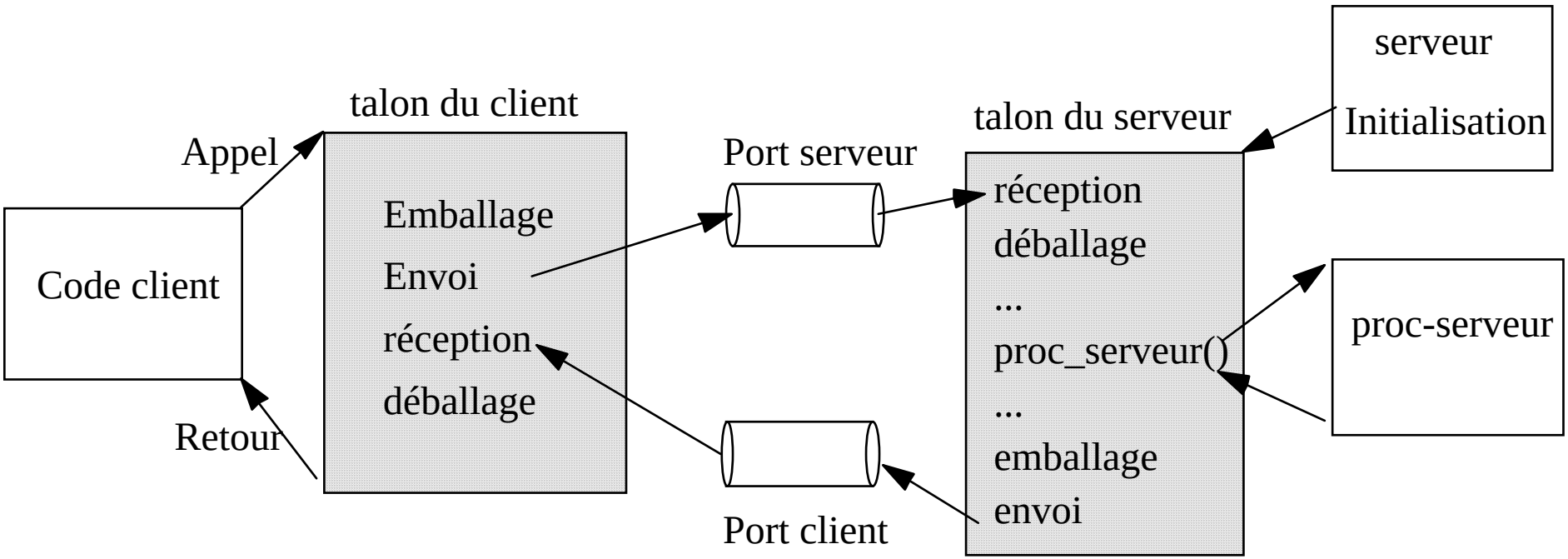
Objectif : Simplifier l'écriture d'applications réparties

MIG utilisé pour générer automatiquement les fonctions de communications de type RPC (de manière similaire au rpcgen de Sun)

A partir d'un **fichier de spécification** 3 programmes sont générés :

- "User Interface Module": code de la partie cliente (le "stub" client)
- "User Header Module" : définitions des types et prototype
- "Server Interface Module" : code de la partie serveur ("stub" serveur)

Utilisation de MIG



Partie générée par le MIG

Partie écrite par le programmeur

MIG - Exemple d'utilisation

Exemple pour la fonction :

Ma_procedure(int a, int *b)

Fichier de spécification (mfonc.defs) :

```
routine Ma_fonction (
    server_port : port_t;
    arg : int;        /* parametre d'entree */
out   reponse : int; /* parametre de sortie */
)
```

mig -v mfonc.defs

Exemple - MIG

```
/* Exemple : gestion compte */
subsystem compte 32768;

userprefix compte_;
serverprefix do_;

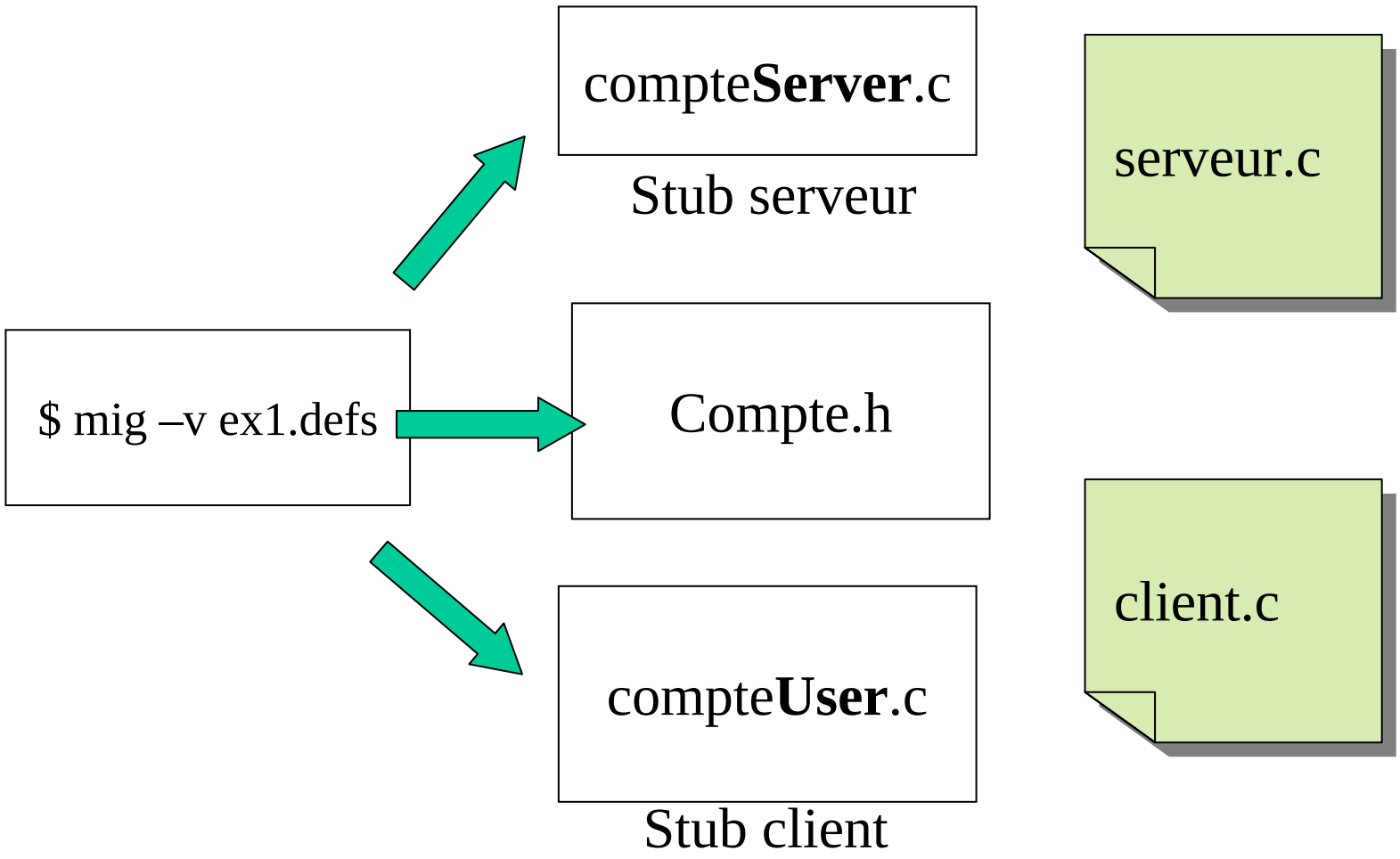
#include <mach/std_types.defs>

routine ajout
(
        serveur : mach_port_t;
        in val : int;
        out nouv : int
);

routine retrait
(
        serveur : mach_port_t;
        in val : int;
        out nouv : int
);
```

Ex1.def

Exemple


compte**Server**.c
Stub serveur
serveur.c
$ mig –v ex1.defs
Compte.h
client.c
compte**User**.c
Stub client

MIG : serveur.c

- ```
  #include <stdio.h>
  #include <mach/mach.h>
  #include <mach/mach_error.h>
  #include <mach/mig_errors.h>
  #include <mach/message.h>
  #include <mach/notify.h>
  #include "compte.h"

  #define MAX_MSG_SIZE 512

  extern boolean_t        compte_server();
  int val_compte = 0; /* Le compte */

  /* implementation des 2 fonctions */

  kern_return_t do_ajout(mach_port_t s, int val, int *nouv) {
    val_compte += val;
    *nouv = val_compte;

    return KERN_SUCCESS;
  }

  kern_return_t do_retrait(mach_port_t s, int val, int *nouv) {
    val_compte -= val;
    *nouv = val_compte;
    return KERN_SUCCESS;
  }
  ```
  ```

Mig: serveur.c (2)

```
int main()
 {
   port_t ServerPort;
   kern_return_t retcode;

          /* Allouer un  port au serveur  */
   retcode = mach_port_allocate(mach_task_self(), MACH_PORT_RIGHT_RECEIVE,
 &ServerPort);

   if (retcode != KERN_SUCCESS){
     printf("mach_port_allocate %s\n",
              mach_error_string(retcode));
     exit(1);
   }

   /* Enregister le port dans le serveur de nom */
   retcode  = netname_check_in(bootstrap_port,"Essai", mach_task_self(),ServerPort);

   if (retcode != KERN_SUCCESS){
          printf("netname_check_in : %s\n",
                                        mach_error_string(retcode));
          exit(1);
   }

   /* Attente de message */
   retcode = mach_msg_server(compte_server, MAX_MSG_SIZE, ServerPort,0);

   printf ("(* !!!!! Server exited !!!!! : %s\n", mach_error_string(retcode));
   return 0;
 }
```

Mig : client.c

```
#include <stdio.h>
 #include <mach/mach.h>
 #include <mach/message.h>
 #include <mach/mach_error.h>
 #include "compte.h"

 int main() {
   mach_port_t serveur;
   kern_return_t ret;
   int c;

   /* Rechercher le port du serveur */
   task_get_bootstrap_port(mach_task_self(), &name_server_port);
   ret = netname_look_up(name_server_port, "", “Essai", &serveur);

   if (ret != KERN_SUCCESS) {
     printf("Pb look_up : %s\n", mach_error_string(ret));
     exit(1);
   }

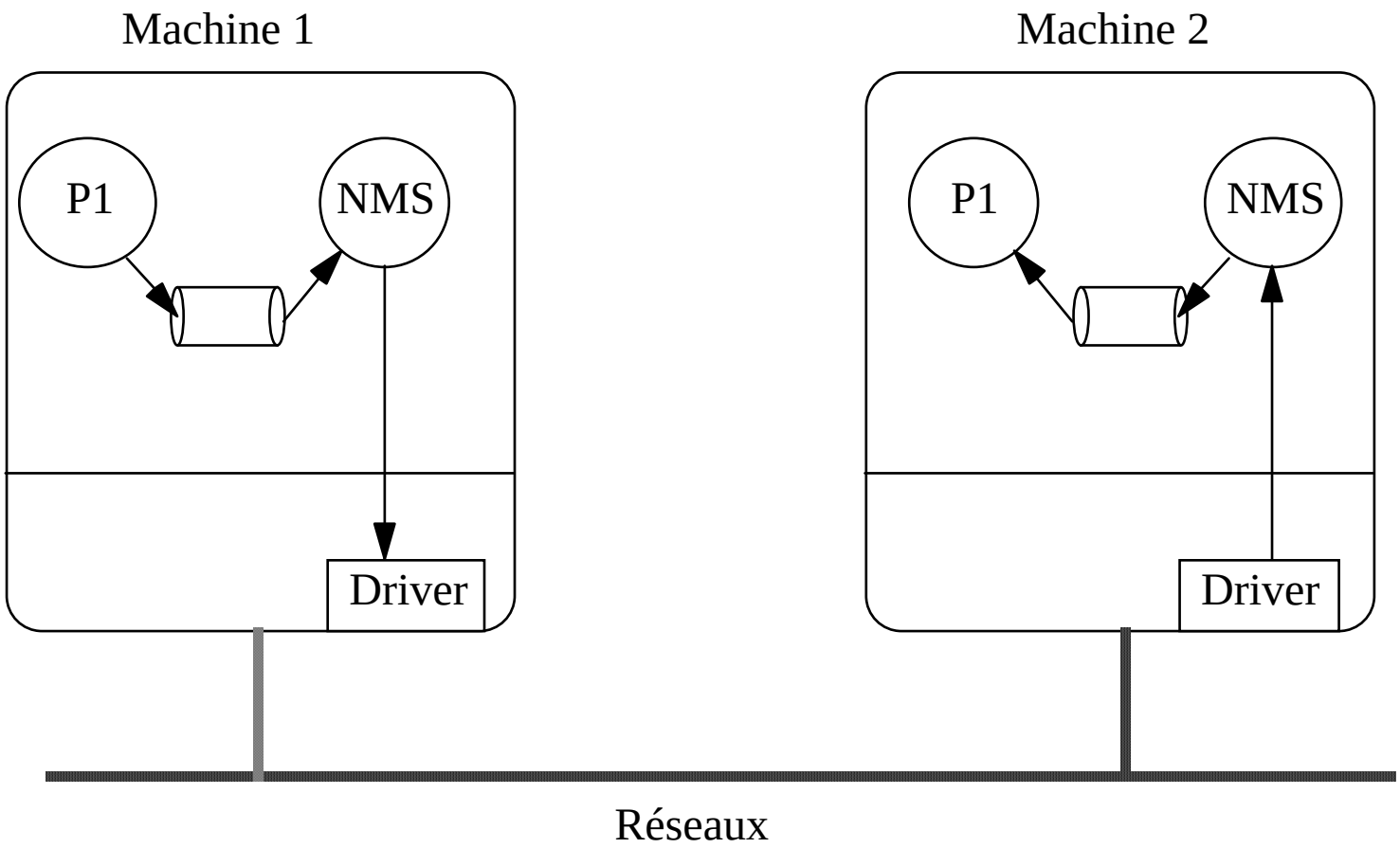
   /* Appel des stub clients */
   compte_ajout(serveur, 10, &c);
   printf("Valeur du compte = %d\n", c);

   compte_retrait(serveur, 5, &c);
   printf("Valeur du compte = %d\n", c);
   return 0;

 }
```

Communications distantes

Première approche : Un serveur réseau sur chaque machine



Les serveurs réseau

Netmsgserver

- Développé par Carnegie Mellon en 1986
- Composé d'une tâche multithreadée en mode utilisateur
- Chaque serveur à une vue cohérente de toutes les tâches s'exécutant sur le réseau (cohérence assurer à base de diffusion)
- Offre un service de communcation transparent à l'utilisateur

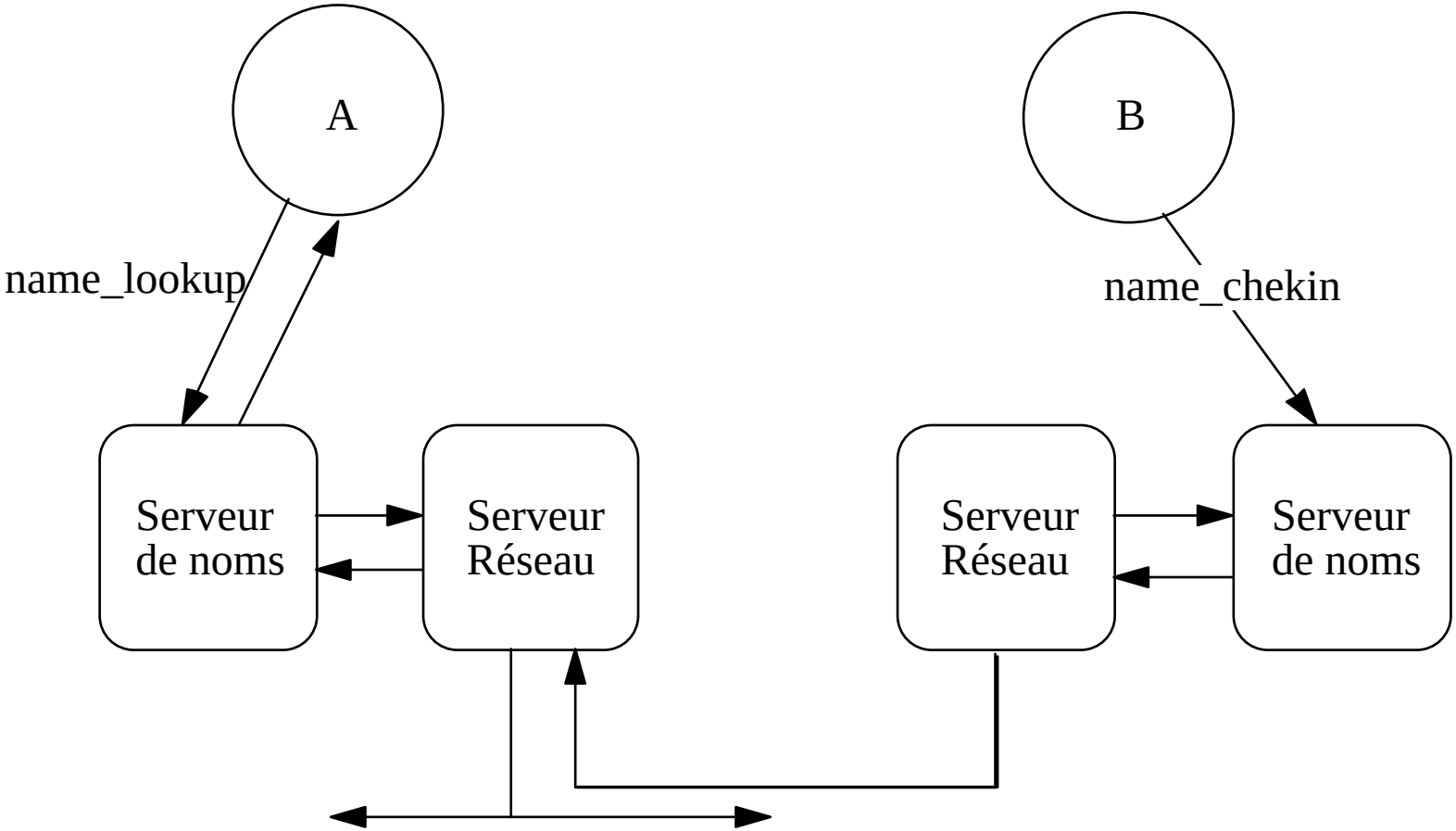
Mais problème de performances

Le serveur réseau générique (Masix)

- Développer au Masi en 1995
- Garantit la transparence des communications
- Optimisation dans le protocole de résolution (gestion des caches)
- Limite les changements de contexte (deporter une partie du traitement directement dans l'espace des applications)

Gestion des accès transparents

Exemple de Masix



Communication Distantes (2)

2ème approche : Gestion des communications intégrée dans le noyau

- **NORMA (OSF)** : Définition de port "NORMA" globaux uniques dans le système
- Chaque site maintient dans le noyau une table globale des ports NORMA

  Avantage : les performances

  Inconvénient : modification du micro-noyau => problème de portabilité

Les systèmes existants au-dessus de Mach

Différentes catégories :

Système UNIX : Lites, OSF1, BSD SS, MachOS X (NextStep), Mklinux, Hurd

Multi-environnements : MASIX, Windows

3 approches de conception :

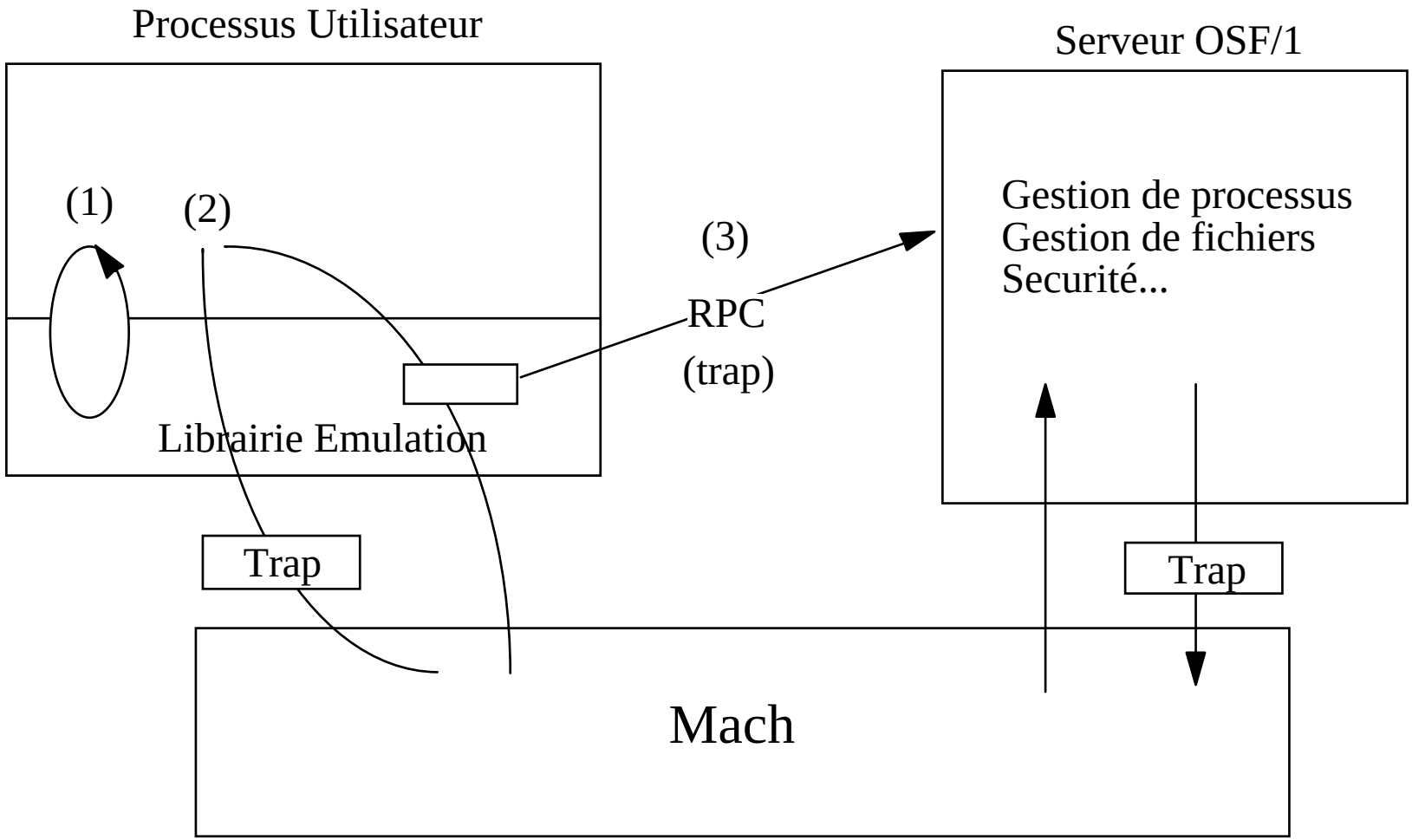
Monolithiques : OSF/1 IK

Serveur unique : OSF/1 MK, BSD SS, Sprite, MKlinux

Multi-serveurs : **Hurd**, Unix Multi server (CMU), Guide, Masix

Approche Mono-serveur : OSF/ 1 MK

Réalisation des appels systèmes



Approche Multi-serveusr : MASIX

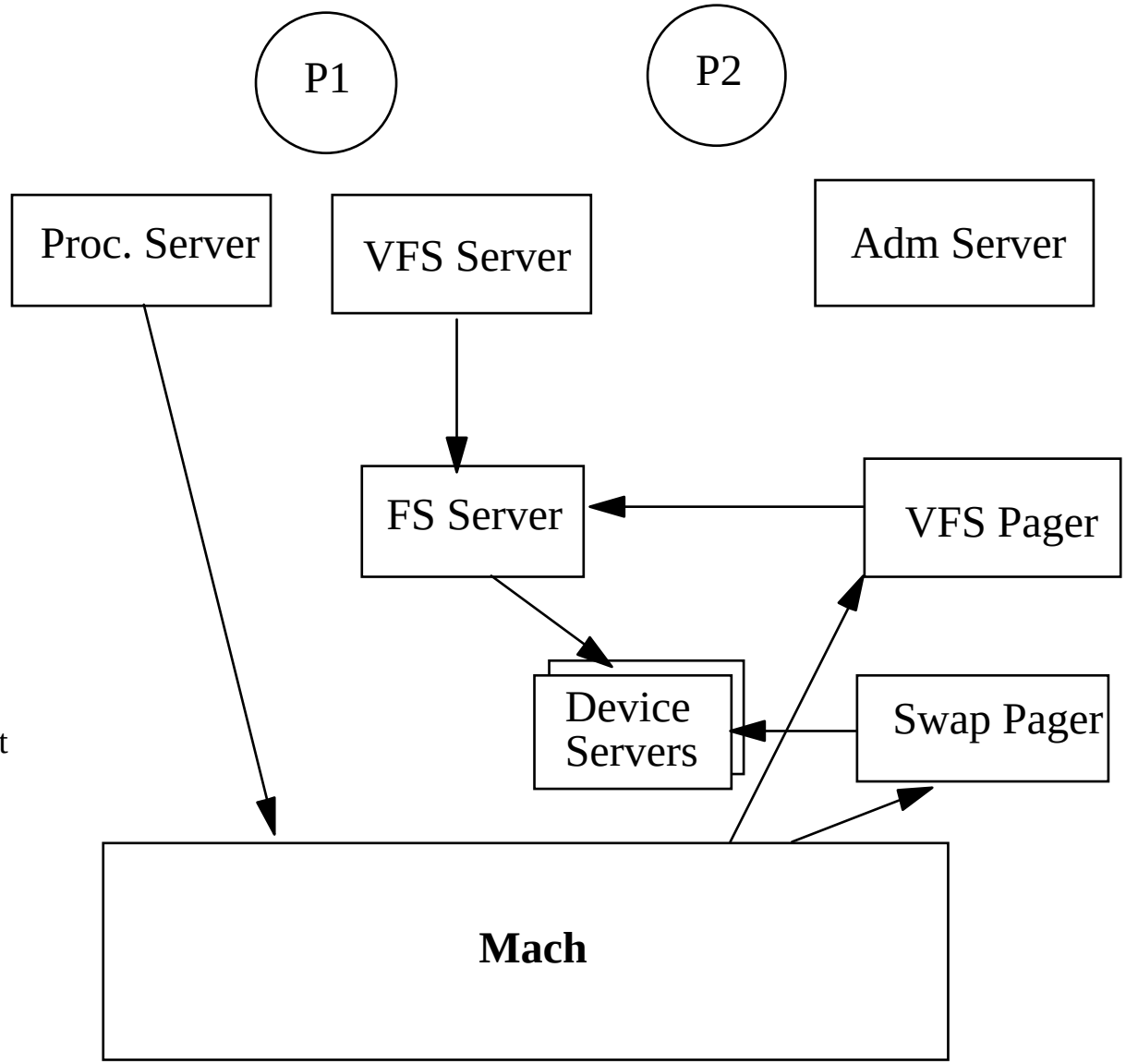
Adm Server :
Initilisation + Terminaison

Proc. Server:
Processus + signaux

VFS server:
Fichiers

VFS Pager :
Fichiers "mappés"

Swap Pager :
Chargement + Déchargement



Introduction Chorus

Propriétés :

Temps réel

Répartition transparente des traitements et des données

Modularité

Structure :

Petit noyau temps réel (100Ko) intégrant :

- la gestion de la mémoire
- les communications entre tâches

Un ensemble de sous systèmes indépendants

Le système Chorus : Abstraction

- **Acteurs** :
  Unité d'encapsulation de ressources
  (espace d'adressage, activités, communications)

  Acteurs utilisateurs :
  espace d'adressage privé

  Acteurs système :
  réaliser certains appels système

  Acteurs superviseurs :
  espace d'adressage du noyau

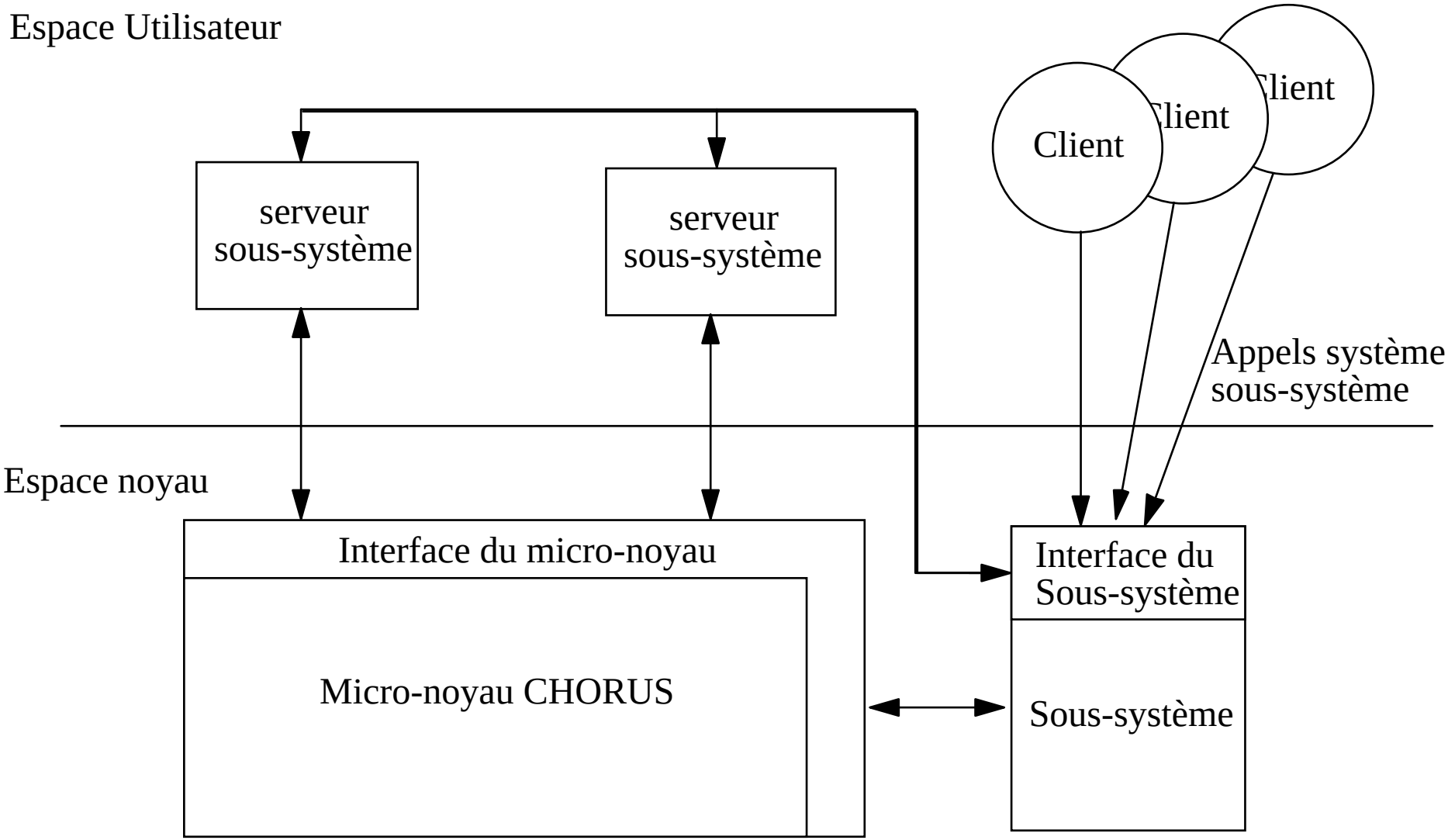
- **Activités** :
  Unités d'exécution appartenant à un acteur
  entités indépendantes pour l'ordonnancement

  Activité de supervision

  Activités utilisateurs

- **Désignation :**
  Identificateurs Uniques (UI) <site de création, type, n° incarnation+compteur >
  (64 bits)

Architecture Générale



Nommage des objets

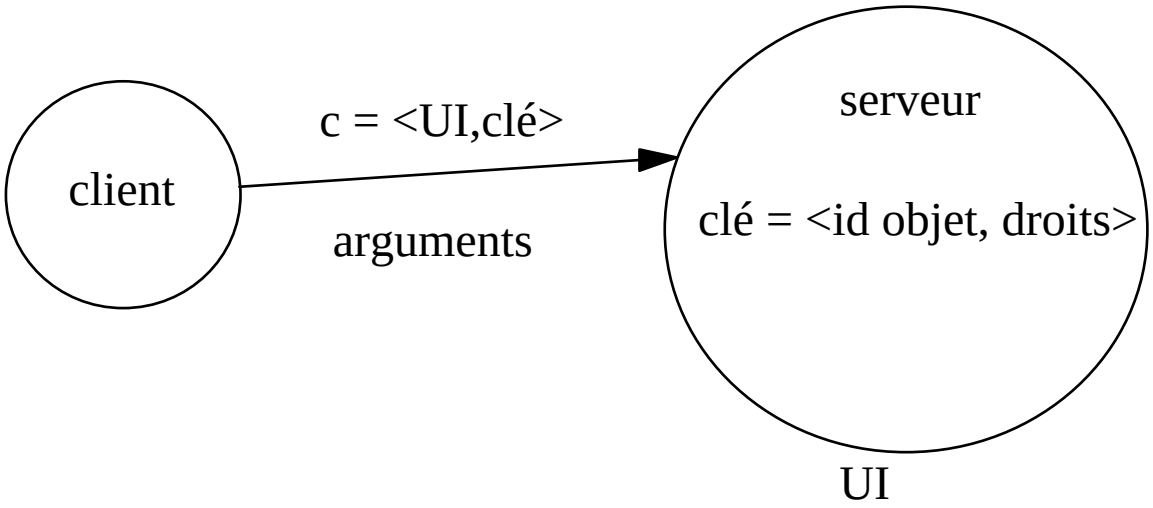
3 niveaux :

1°) Local Identifier (LI) : A l'intérieur d'un acteur

2°) Unique Identifer (UI) :Global et idenpendant du site (64 bits)
Site de création (13 bits), type de l'objet (3 bits), incanration + compteur (48 bits)

3°) Capacité (groupes, acteurs, segments) :Global + Protection
UI + Clé (64 bits)

=> Gestion plus fine de l'objet



IPC - Chorus

Communication locale à un acteur

partage de données dans l'espace d'adressage
contrôle de concurrence : sémaphores

Communication entre activités d'acteurs différents

Communication par message (locale ou distante)
Communication synchrone / asynchrone
Transparence vis-à-vis de la localisation

Communication - Les portes (2)

- Identification : UI
- Localisation :

  Une cache sur chaque site
  Diffusion réquête de localisation`
- Droits d'émission vers une porte :

  Connaissance de l'UI
  Protection assurée par sous-système
- Droits de réception :

  Appartenance à l'acteur possédant la porte

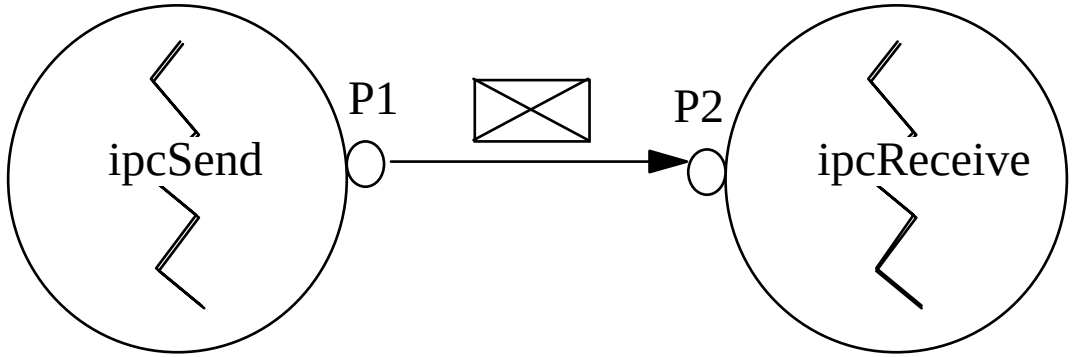
Création :
  portCreate : UI générée par le noyau
  portDeclare : UI construite
Suppression :
  portDelete
Migration :
  portMigrate (option, K_WITHMSGS, K_KILLMSGS)

Modes de communication

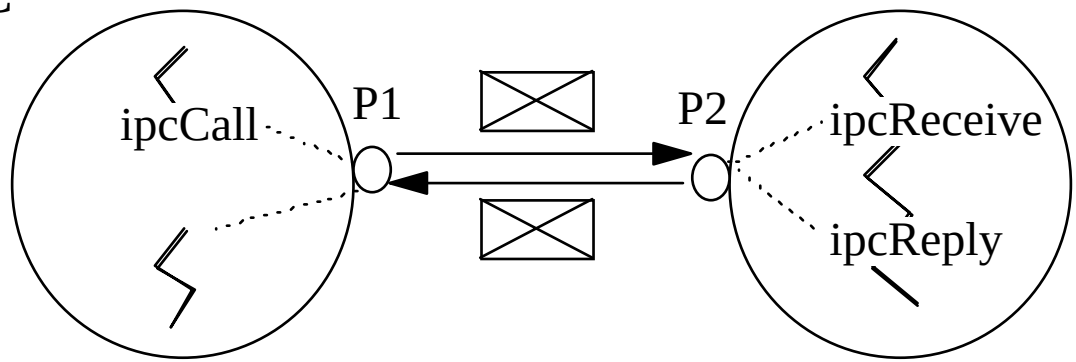
Messages

Forme libre

Asynchrone



Synchrone / RPC



sémantique au plus une fois (at most once)

Groupe de Portes

Objectif : Permettre de garantir une certaine stabilité du système en cas de défaillance d'un serveur

- Identification par un **UI de groupe**

- Capacité associée :
  <UI du groupe, clé>
  La clé sert à modifier la composition du groupe

Groupe statique :

Nom de groupe connu (utilisé pour générer une capacité)
(analogue aux ports des sockets Bsd ou aux clés des IPC système V)

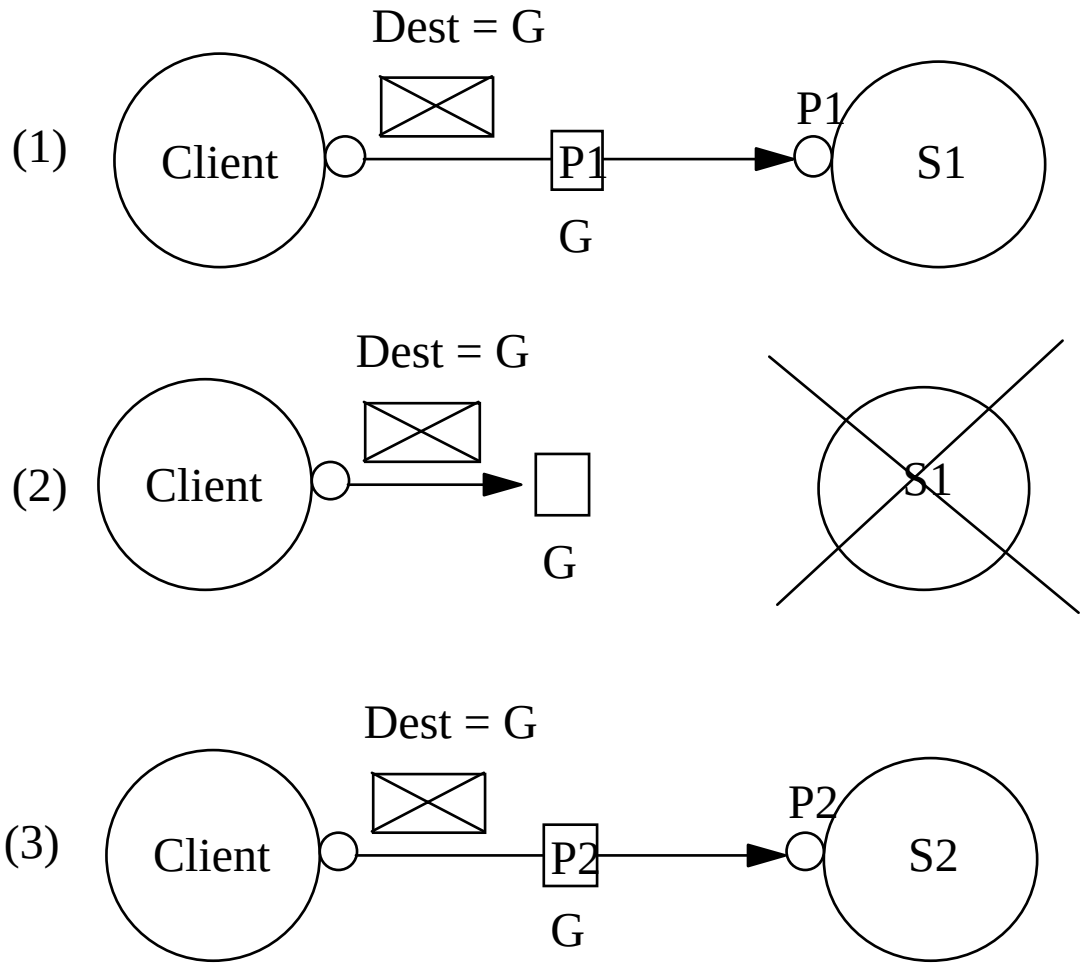
Groupe dynamique :

Une nouvelle capacité générée par le système

grpAllocate(K_DYNAMIC | K_STATUSER,...)

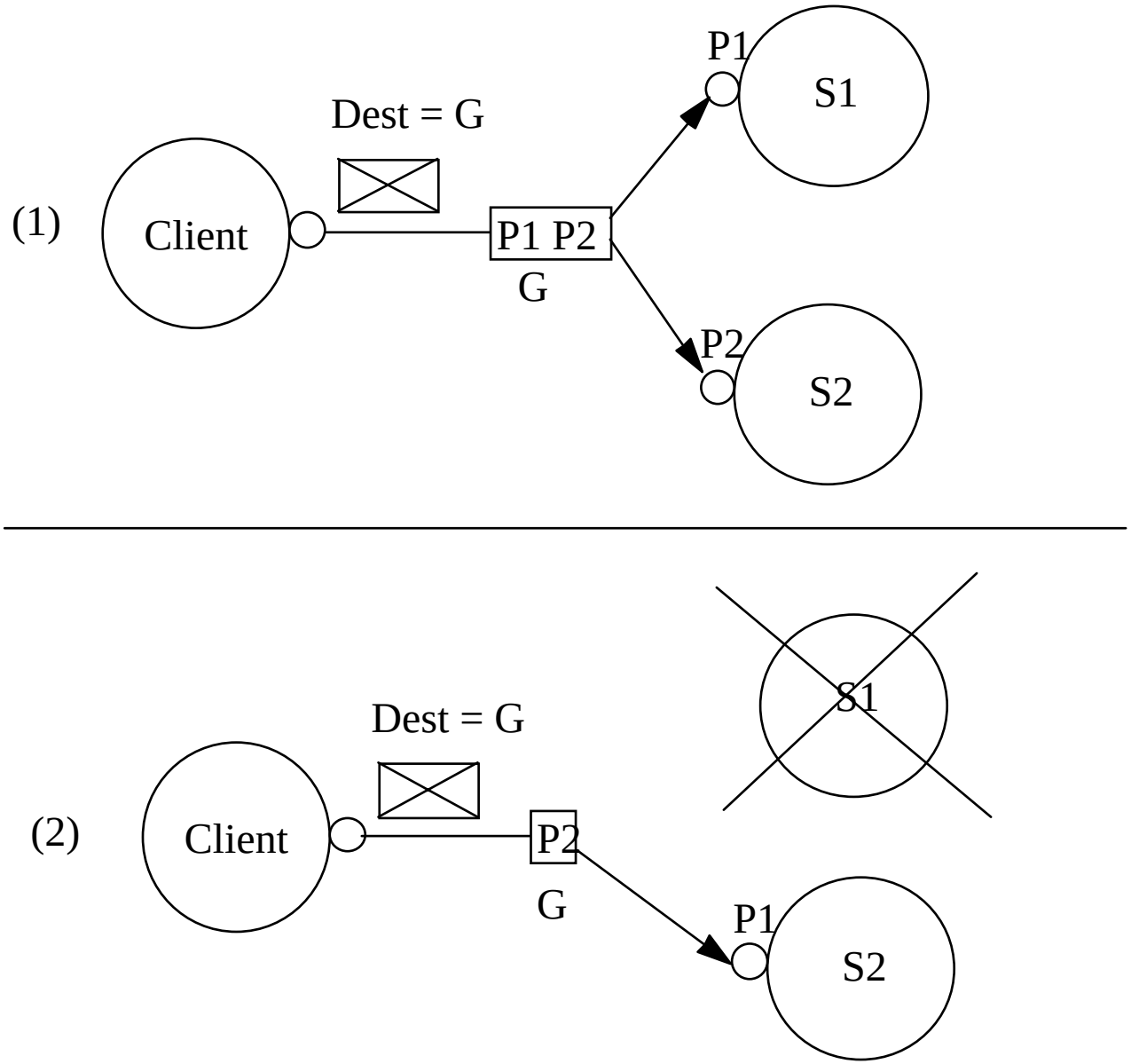
Utilisation des groupes

1) Reconfiguration



Utilisation des groupes

2) Réplication



Opérations et Modes d'adressage

Opérations sur les groupes :

Insertion : grpPortInsert

Extraction : grpPortRemove

Modes d'adressage:

- Diffusion (Broadcast) : vers toutes les portes du groupe
- Fonctionnel simple : vers une des portes du groupe
- Fonctionnel indicé : vers une porte du groupe sur le même site
- Fonctionnel exclusif : vers une porte du groupe sur un site différent

Exemple

```
KnUniqueId    myportUI;
int           myPort;
char *        message = "Hello world"
KnMsgDesc     msg;
KnIpcDest     dest;
int           result;

main()
{
      /* création de mon port */
      myPort = portCreate(K_MYACTOR, &myPortUi);

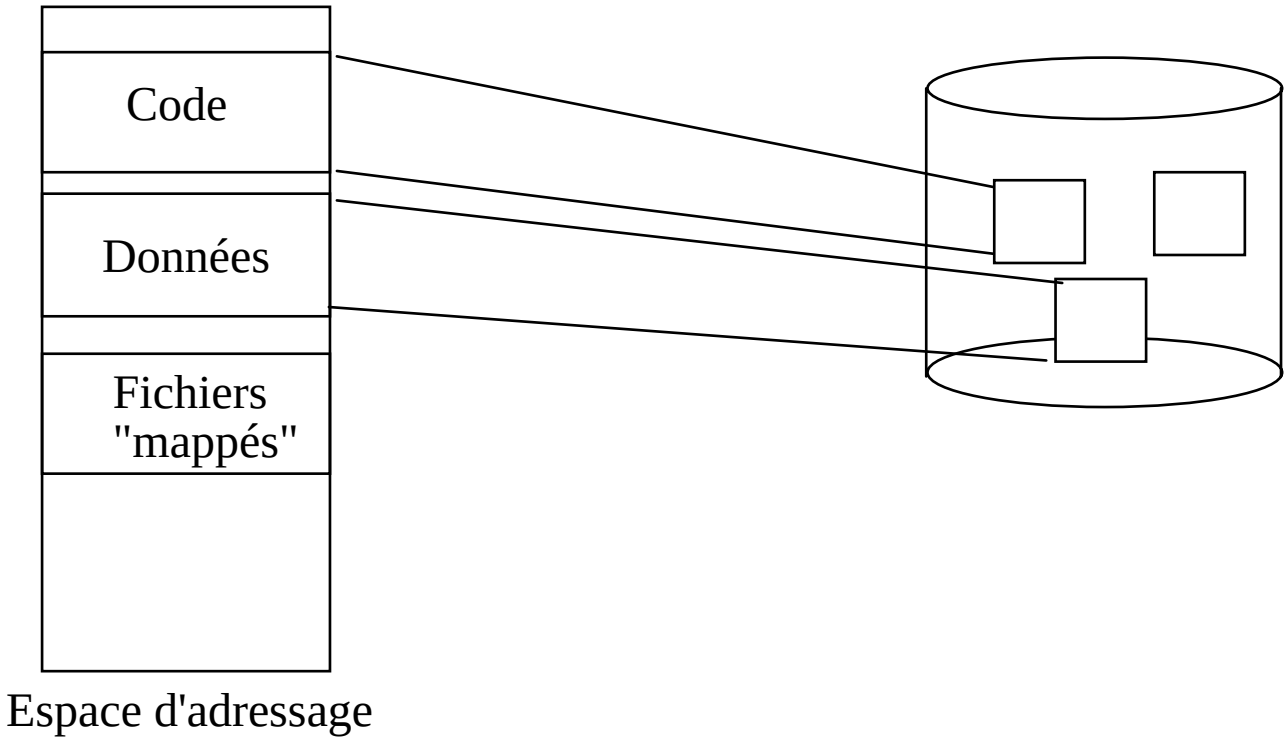
      /* Emission d'un message à partir de myPort vers dest*/
      msg.flags = 0;
      msg.annexAddr = 0;
      msg.bodySize = strlen(message);
      msg.bodyAddr = (VmAddr) message;
      ipcSend(&msg, myport, dest);

      /* Réception d'un message sur myPort/*
      msg.flags = 0;
      msg.annexAddr = 0;
      msg.bodySize = 80;
      msg.bodyAddr = (VmAddr) message;
      result = ipcReceive(&msg, &myPort, K_NODELAY);
      if (result < 0)
          printf("Erreur réception");
      else
          printf("réception de %d octets", result);
}
```

Gestion de la mémoire

Espace d'adressage d'un acteur est découpé en **région**

Segment : abstraction associée à un objet qui pourra être projeté dans une région d'un acteur



Gestion de la mémoire (2)

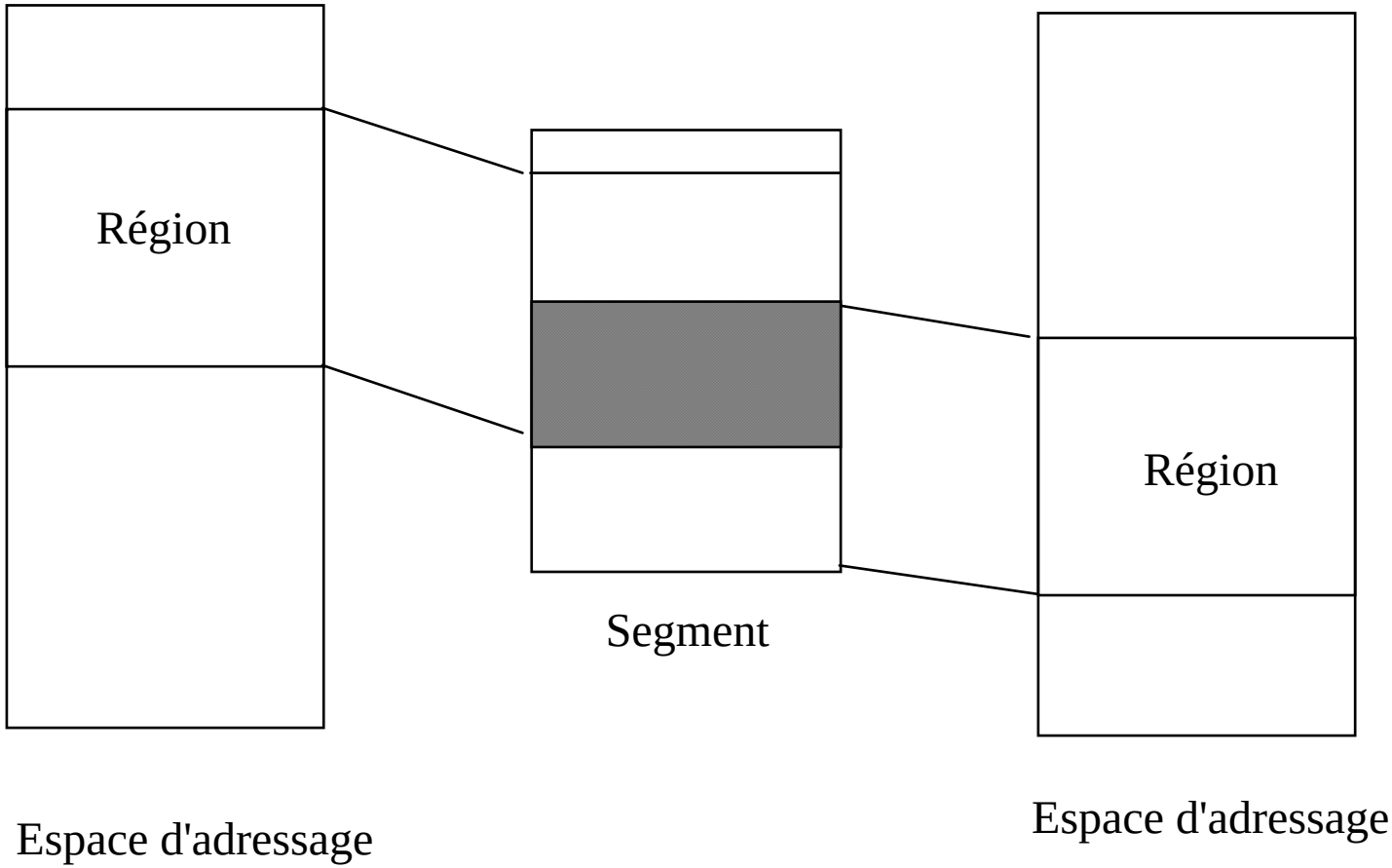
Segment :

- Désigné par une capacité => indépendant de la localisation
- Utilisé pour implémentation de fichiers mappés, mémoire paginée, mémoire partagéé
- Géré par un serveur (mapper)
- Chaque serveur possède sa propre gestion de la cohérence, protection et représentation

Région :

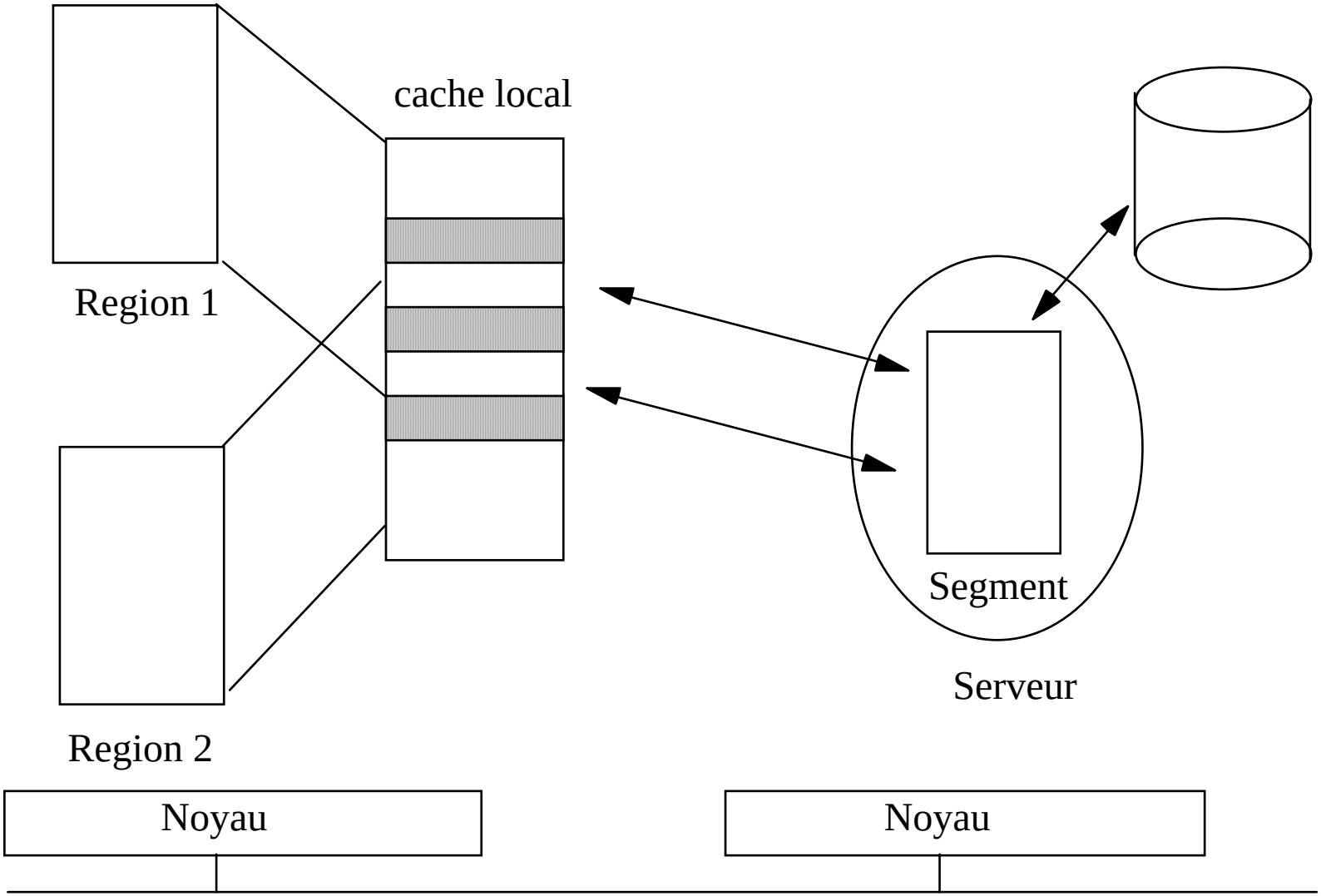
Associé à un portion de segment
Attributs :
- Position
- Protection
- Héritage (copie ou partage)
- Pagination à la demande autorisé

Partage

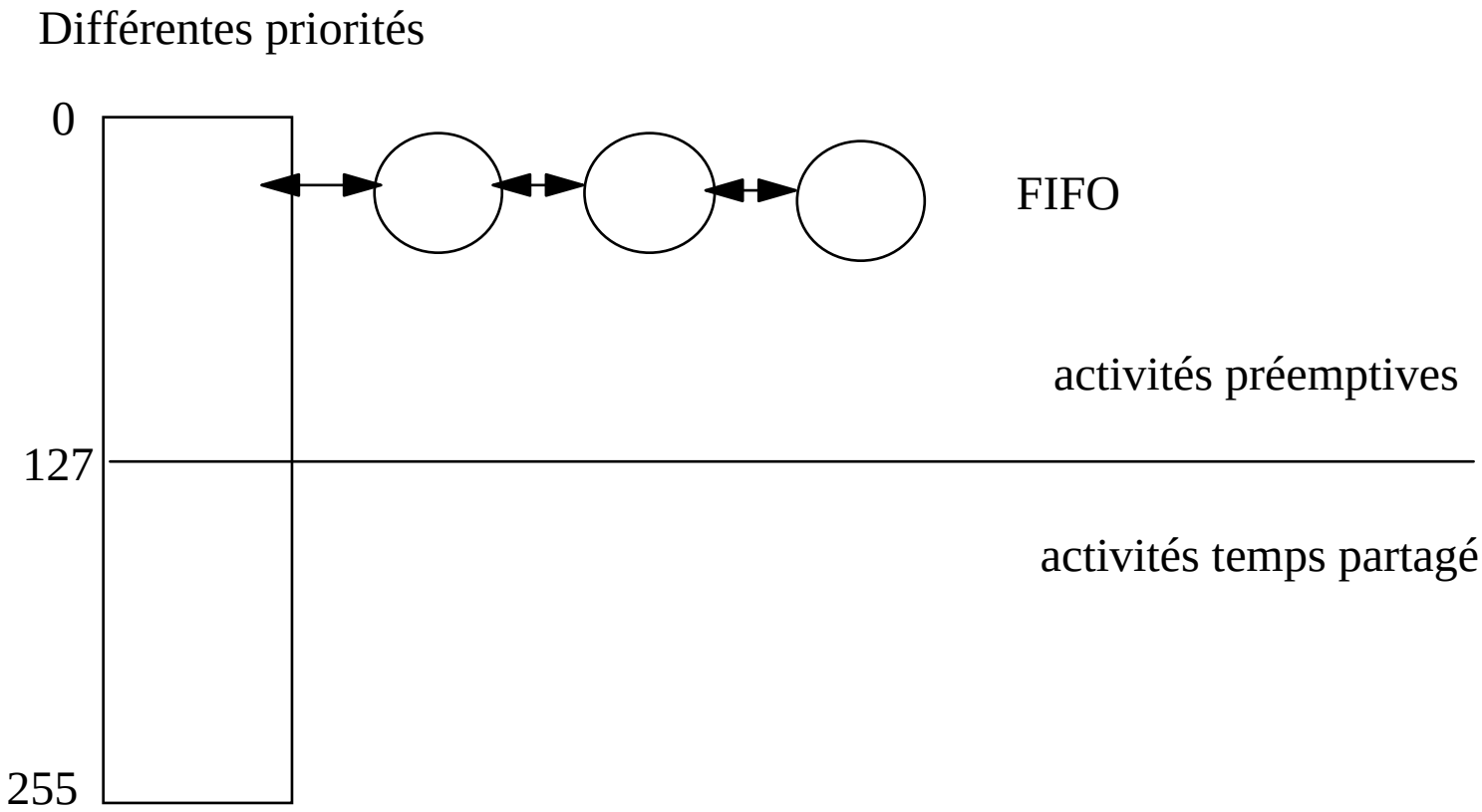


Cohérence gérée uniquement pour un partage entre acteurs locaux

Représentation des segments en mémoire physique



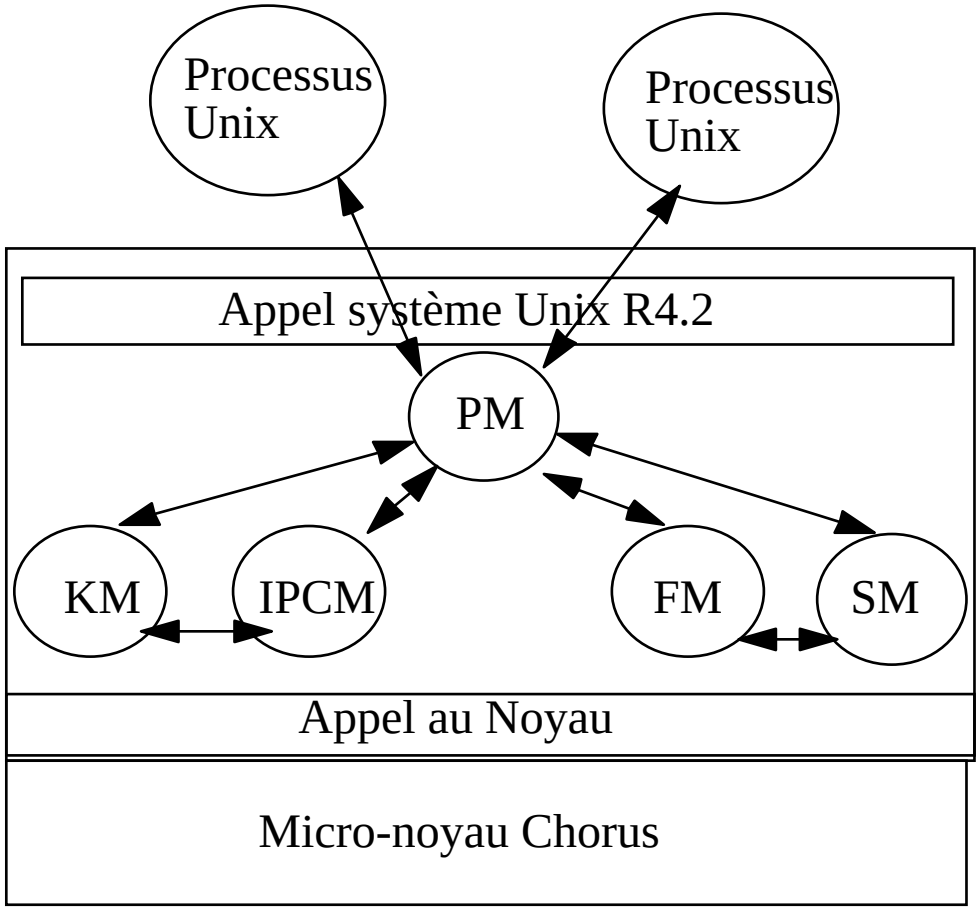
Ordonnancement Chorus



Modularité : Implanter plusieurs politiques d'ordonnancement
1 classe par défauts + 1 classe pour le sous-système Unix

Sous-système : Exemple Chorus MIX

SUSI : Single Unix System Image



PM = Process Manager
FM = File Manager
SM = Streams Manager
IPCM = IPC Manager (System V)
KM = Key Manager

Amoeba

Micro-noyau multi-serveurs utilise le modèle de "pool de processeurs"

Orienté objet

Objets référencés par des capacités

Intègre la notion de threads

Pas de gestion de mémoire virtuelle !

Communication :

Utilise des ports (similaire à Mach)

Possibilité de diffuser des messages à des groupes logiques de ports

V - Kernel

Notion de processus legers

Défintion de groupe de processus légers

Ordonnancement à deux niveaux :
Micro-noyau, Externe

Communication :

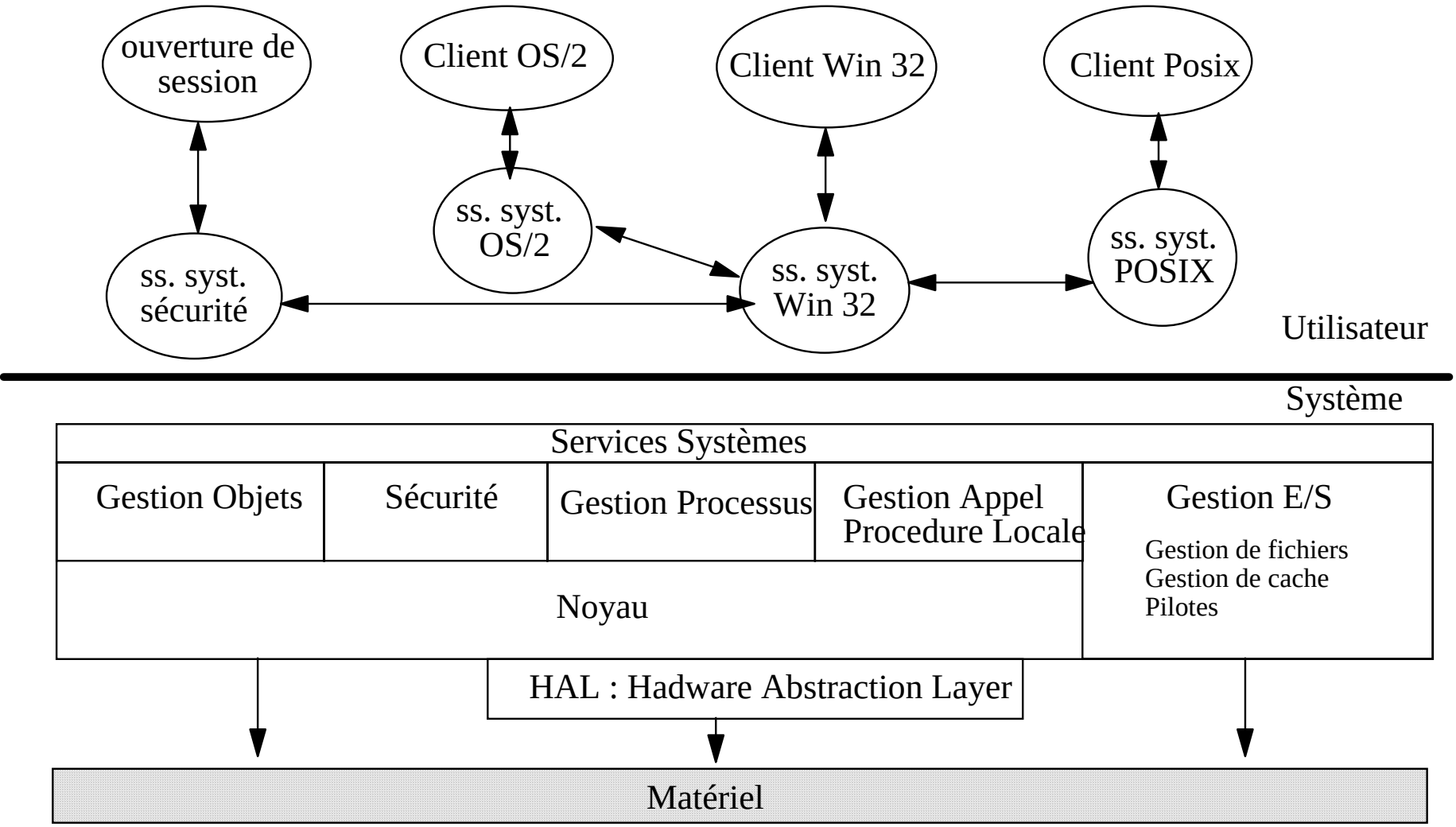
Pas de port (on nomme le processus destinataire)

Communication par RPC uniquement

Diffusion vers un ensemble de processus

Messages de 32 bits !

Architecture de Windows NT



Synthèse sur les micro-noyaux

Points communs :

Tâches et activités

Communications

Spécificités :

Amoeba : pas de mémoire virtuelle

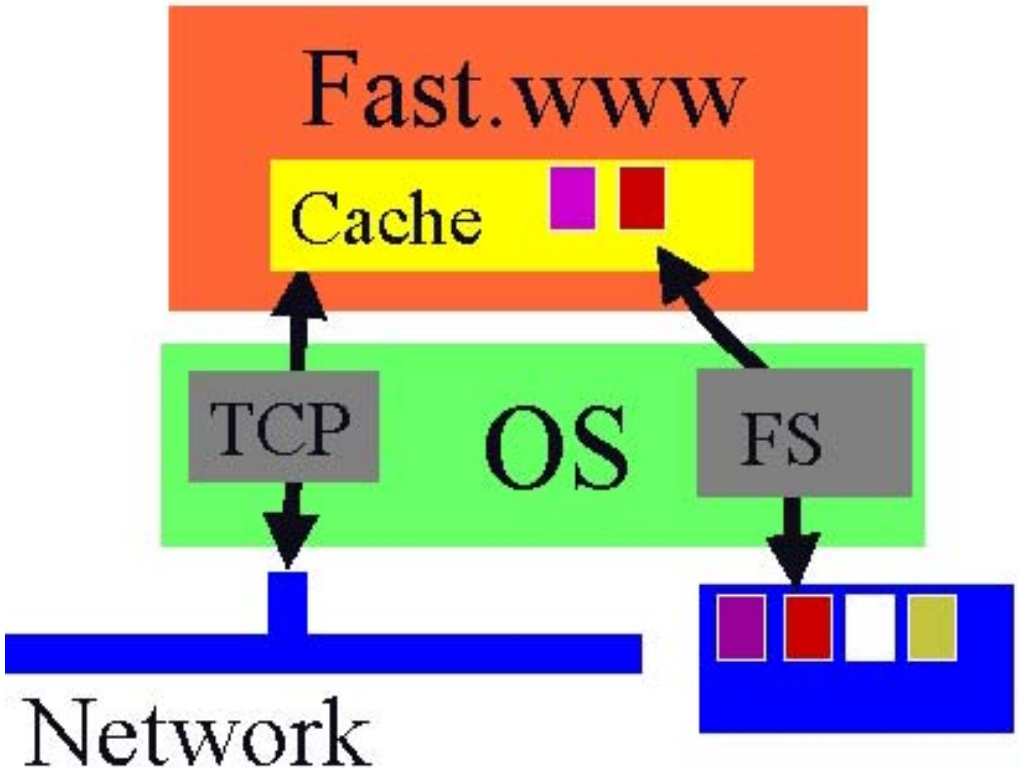
Chorus : Migration de portes, Message handler

Mach : Ecoute sur un ensemble de ports

V kernel : pas de ports, messages de 32 bits

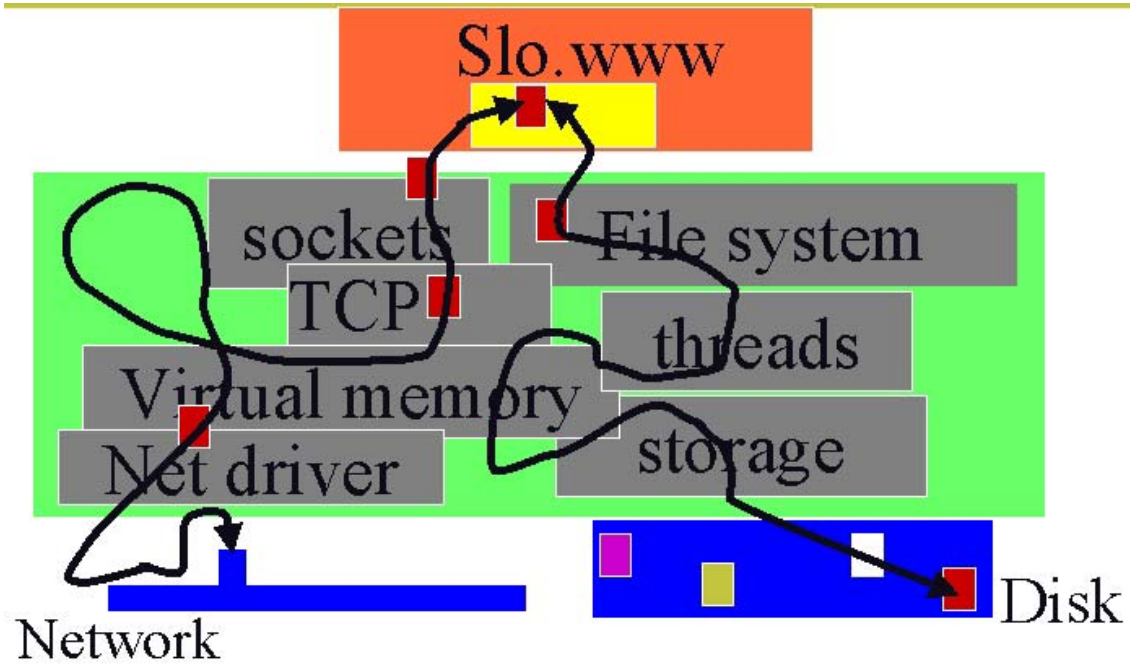
Exokernel (MIT)

- Lourdeur des systèmes traditionnel :
  Exemple un serveur web



Exokernel : Motivations

Système traditionnel

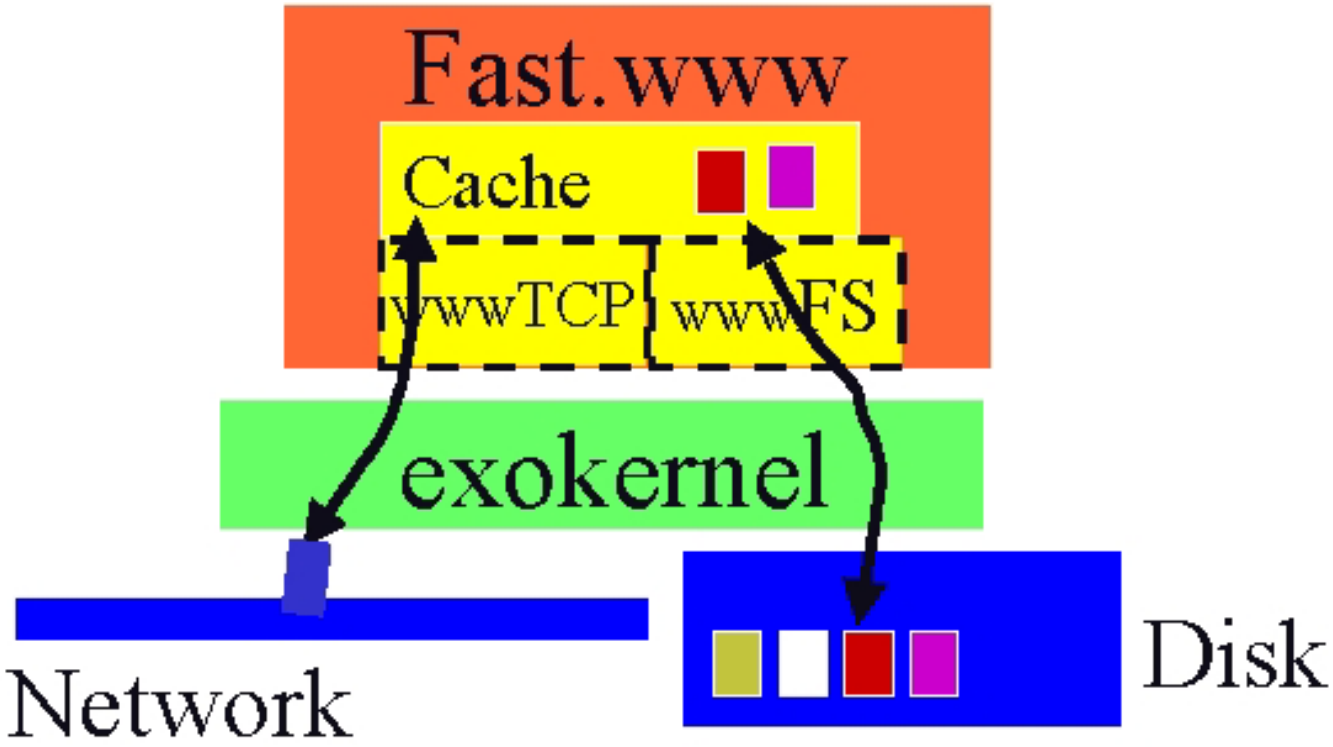


La solution Exokernel

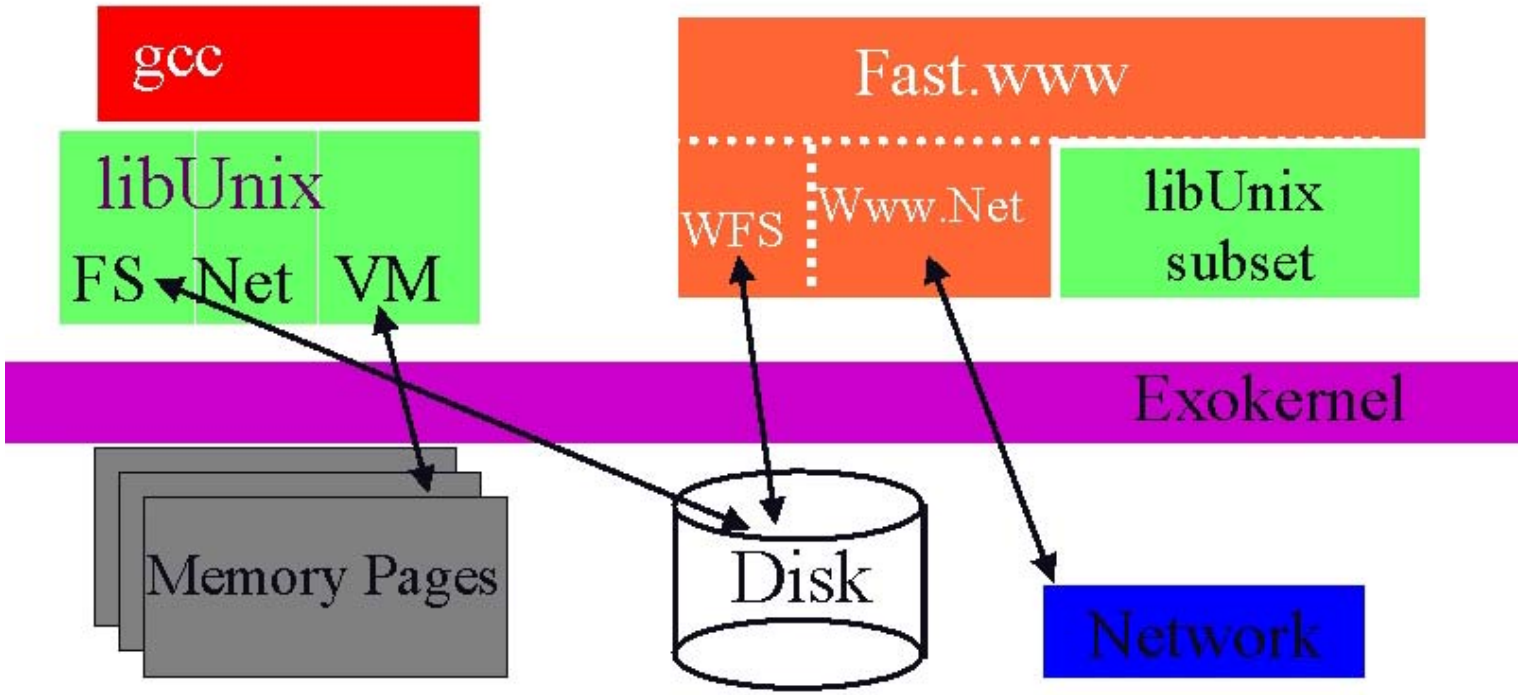
- Noyau externalisé dans l’espace utilisateur
- Noyau minimum
- Toutes les fonctions dans des bibliothèques : les libOS

=> moins de recopies plus de partages

Architecture Exokernel

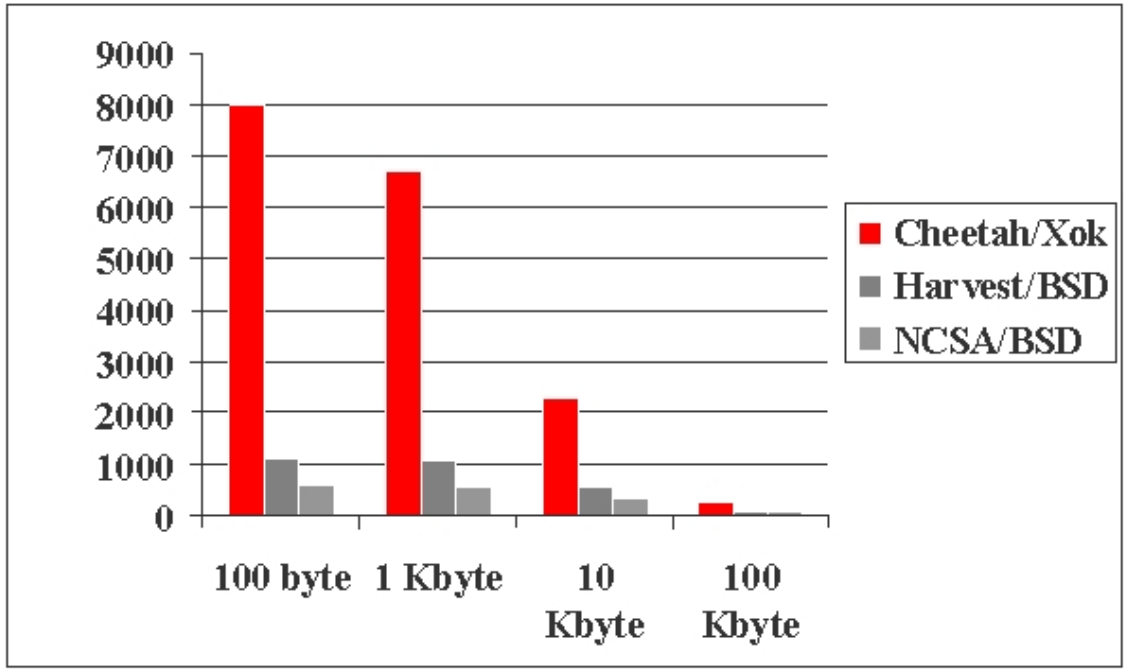


Architecture (2)



Performances Exokernel

The Cheetah Web Server



Exokernel - Conclusion

- Approche alternative performante
- Complexité de mise en œuvre (développement des libOS)
- Sécurité

L4 Microkernel

- 1995 - German National Research Center for IT

- Nouveau micro-noyau :
  - Objectif améliorer les performances des micro-noyaux existants (Mach)
  - => Améliorer les échanges entre serveurs (IPC)

L4 Abstractions

- Espace d’adressages
  - Map, Grant, Unmap
- Threads
- IPC
  - Messages courts (registres)
  - Copie limitée de grands messages (partage de l’espace de l’émetteur)
  - Ordonnancement paresseux (Limiter les interactions avec le noyau)

L4 Performance

	8 Byte IPC	512 Byte IPC
L4	5 µs	18 µs
MACH	115 µs	172 µs