# Un point sur les micro-noyaux

Jérôme Ermont

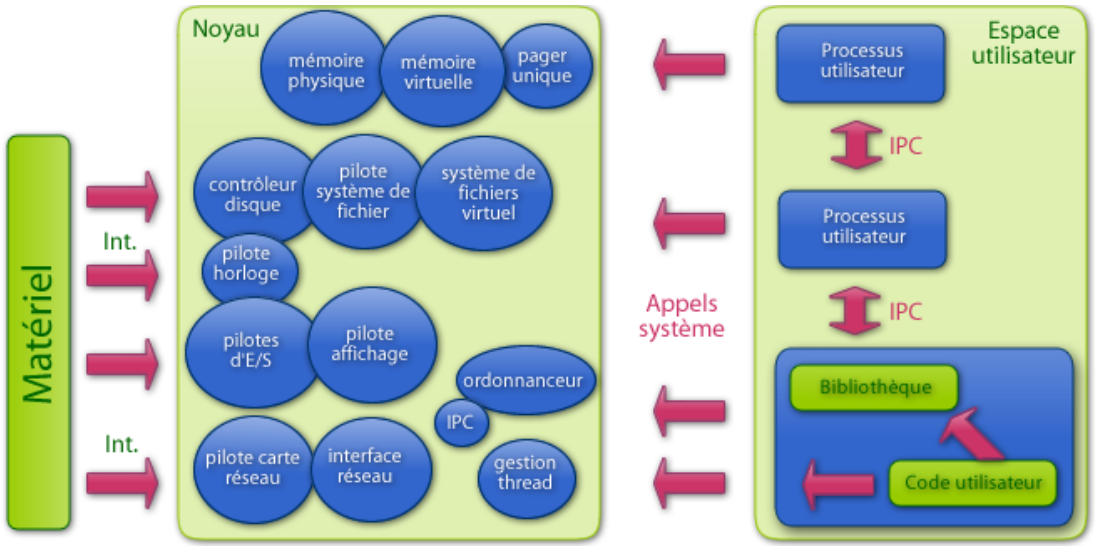
IRIT - Toulouse INP/ENSEEIHT

# Plan de la présentation


1. Introduction
2. Les IPC
3. Le micronoyau Mach
4. La famille L4
5. Au final

# Les différents types de système

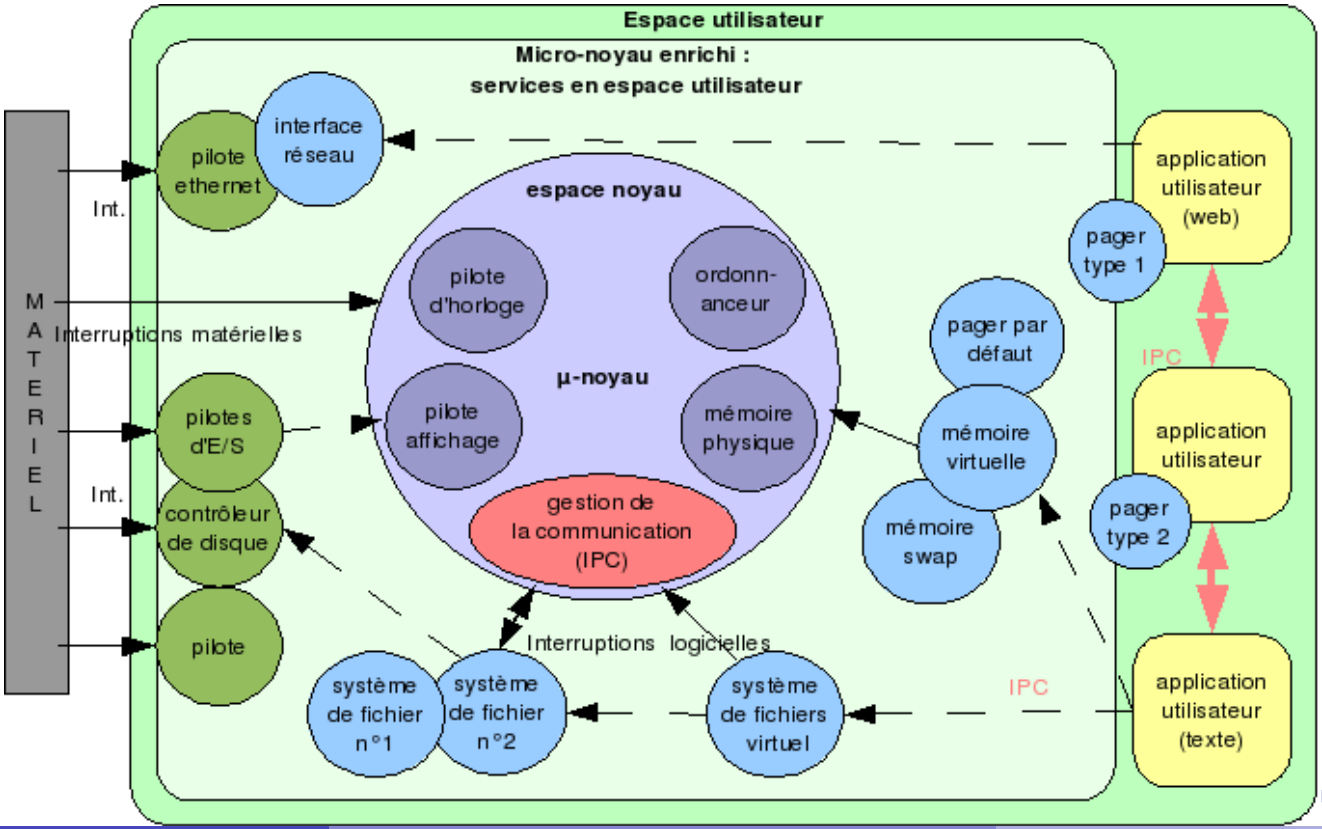
- Systèmes monolithiques :
  - Organisation la plus répandue
  - « Grand désordre » $\leftrightarrow$ ensembles de procédures et de fonctions sans véritable structuration
  - Noyau de taille importante, difficile à maintenir, à débogger
  - Performant




- Systèmes à couches :
  - Plusieurs couches hiérarchiques, les unes reposant sur les autres
  - Système THE de Dijkstra, MULTICS

# Les différents types de système

- Systèmes clients/serveurs (micro-noyaux) :
  - Noyau minimal
  - 2 types de processus : les clients et les serveurs
  - Parallèlisme des traitements
  - Transition facile vers les systèmes répartis
  - Sûr et sécurisé
  - Exemple : Noyau Mach (*BSD, MacOS), Minix, L4



# Qui est le meilleur ?

## LINUX is obsolete - Andrew Tanenbaum, 29/01/1992

I was in the U.S. for a couple of weeks, so I haven't commented much on LINUX (not that I would have said much had I been around), but for what it is worth, I have a couple of comments now. As most of you know, for me MINIX is a hobby, something that I do in the evening when I get bored writing books and there are no major wars, revolutions, or senate hearings being televised live on CNN. My real job is a professor and researcher in the area of operating systems.
As a result of my occupation, I think I know a bit about where operating are going in the next decade or so. Two aspects stand out :
1. MICROKERNEL VS MONOLITHIC SYSTEM Most older operating systems are monolithic, that is, the whole operating system is a single a.out file that runs in 'kernel mode.' This binary contains the process management, memory management, file system and the rest. Examples of such systems are UNIX, MS-DOS, VMS, MVS, OS/360, MULTICS, and many more.
The alternative is a microkernel-based system, in which most of the OS runs as separate processes, mostly outside the kernel. They communicate by message passing. The kernel's job is to handle the message passing, interrupt handling, low-level process management, and possibly the I/O. Examples of this design are the RC4000, Amoeba, Chorus, Mach, and the not-yet-released Windows/NT.
While I could go into a long story here about the relative merits of the two designs, suffice it to say that among the people who actually design operating systems, the debate is essentially over. **Microkernels have won.** The only real argument for monolithic systems was performance, and there is now enough evidence showing that microkernel systems can be just as fast as monolithic systems (e.g., Rick Rashid has published papers comparing Mach 3.0 to monolithic systems) that it is now all over but the shoutin'.
MINIX is microkernel-based system. The file system and memory management are separate processes, running outside the kernel. The I/O drivers are also separate processes (in the kernel, but only because the brain-dead nature of the Intel CPUs makes that difficult to do otherwise). LINUX is a monolithic style system. This is a giant step back into the 1970s. That is like taking an existing, working C program and rewriting it in BASIC. **To me, writing a monolithic system in 1991 is a truly poor idea.**

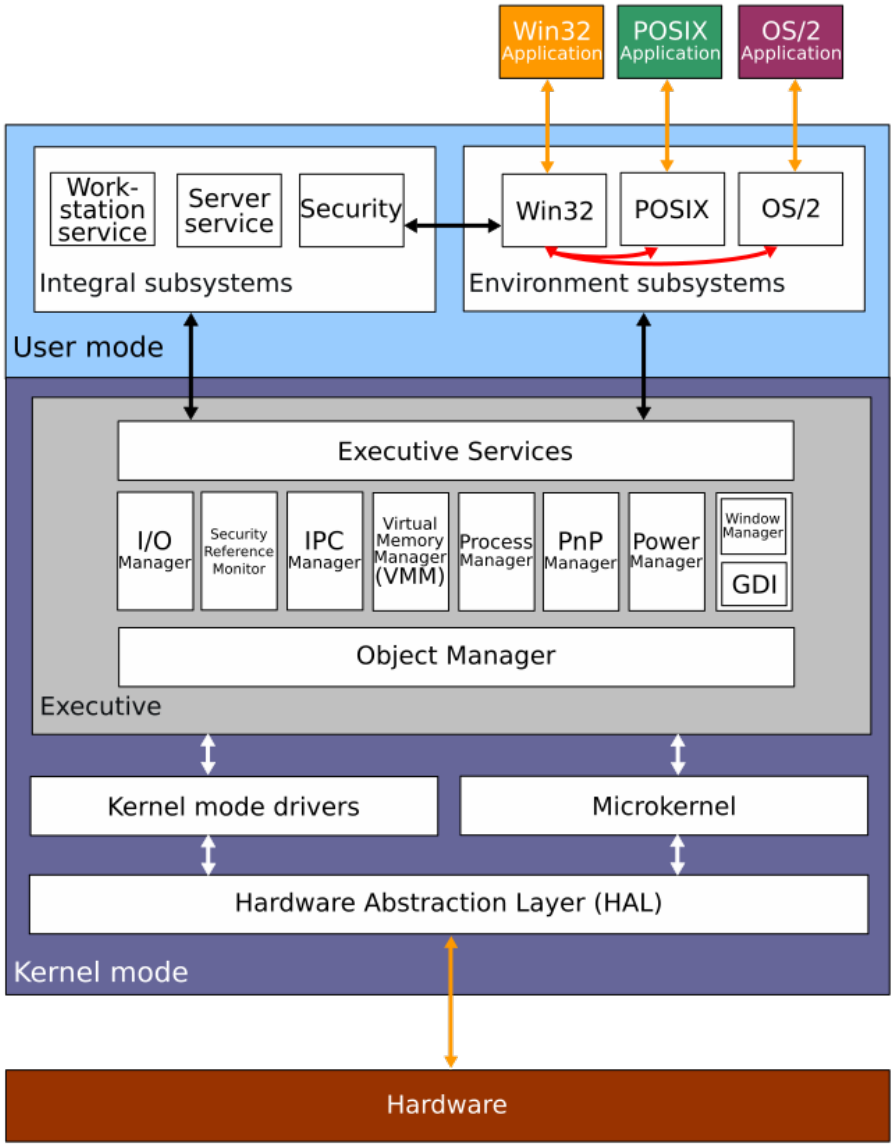
# Avantages des micro-noyaux

- Adaptation au besoin
  - Les serveurs peuvent être configurés pour être adaptés au matériel cible (petits systèmes embarqués, PC, systèmes multi-processeurs, ...)
  - Ajout/Suppression des serveurs au besoin
- Meilleure gestion des accès au système
  - Interface bien définie entre les composants (IPC)
  - Pas d'accès à ces composants en dehors de ces interfaces
  - Evolution/mise à jour facilitée
- Meilleures protections contre les erreurs
  - Erreur d'un composant ne crashe pas tout le système
  - Plus sécurisé via une protection inter-composant

# Avantages des micro-noyaux

- Sous-système de protection / isolation
- Taille du code :
  - Micro-noyaux :
    - ★ Noyau Fiasco : 34000 lignes
  - Linux sur x86
    - ★ noyau : 2,5 millions de lignes
    - ★ +pilotes : 5 millions de lignes

# Et Windows/NT et successeurs ?




Source de l'image : Wikipedia

# Les systèmes hybrides monolithiques/micro-noyaux

- Avantages des deux mondes
- Inconvénients des noyaux monolithiques
- Exemple : Windows NT
- Systèmes modulaires :
  - Les pilotes de périphériques sont lancés aux fur et à mesure de leur besoin
  - Exemple : Linux

## Microkernels have won ?

- Nombre d'utilisateurs de Windows (+Linux)
- Nombre de développeurs de Linux

# Plan de la présentation


1. Introduction
2. Les IPC
3. Le micronoyau Mach
4. La famille L4
5. Au final

# Inter-Process Communication

- IPC est le mécanisme fondamental des systèmes à base de micro-noyau :
  - Pour les échanges de données
  - La synchronisation
  - La gestion du temps
  - Gestion des interruptions
  - Accès aux ressources (mémoire, ports d'E/S, ...)
  - Exceptions
- Jochen Liedtke : "IPC performance is the master."

# Fonctionnement

- Echange de messages
- Fonctions de type : `msgsnd, msgrcvd, ...`
- Enregistrement/stockage des messages émis/reçus

# Différents types de IPC

- IPC asynchrones (Mach)
  - "Fire and Forget"
  - Enregistrement des messages dans le noyau
  - Problèmes :
    - ★ Données en double
    - ★ Attaque en DoS possible sur les accès mémoire

- IPC synchrones (L4)
  - Attente tant que le "partenaire" n'est pas disponible
  - Copie directe entre l'émetteur et le récepteur
  - Remote Procedure Call (RPC)

# Plan de la présentation


1. Introduction
2. Les IPC
3. Le micronoyau Mach
4. La famille L4
5. Au final

# Le micro-noyau Mach

- 1ère génération de micro-noyau
- Développé à l'Université de Carnegie-Mellon (USA) entre 1985 et 1994

- Base pour plusieurs systèmes réels :
  - BSD4.3 on Mach
  - MkLinux (OSF)
  - IBM Workplace OS
  - XNU (macos)

# Principes de base

- Simple, extensible noyau communicant :
  - "Everything is a pipe"
  - Les ports IPC sont des canaux de communication sûrs
- Support multiprocesseur
- Système multi-serveurs
- Compatible POSIX
- Inconvénients :
  - Performances
  - Pilotes de périphériques dans le noyau

# Les éléments constituants

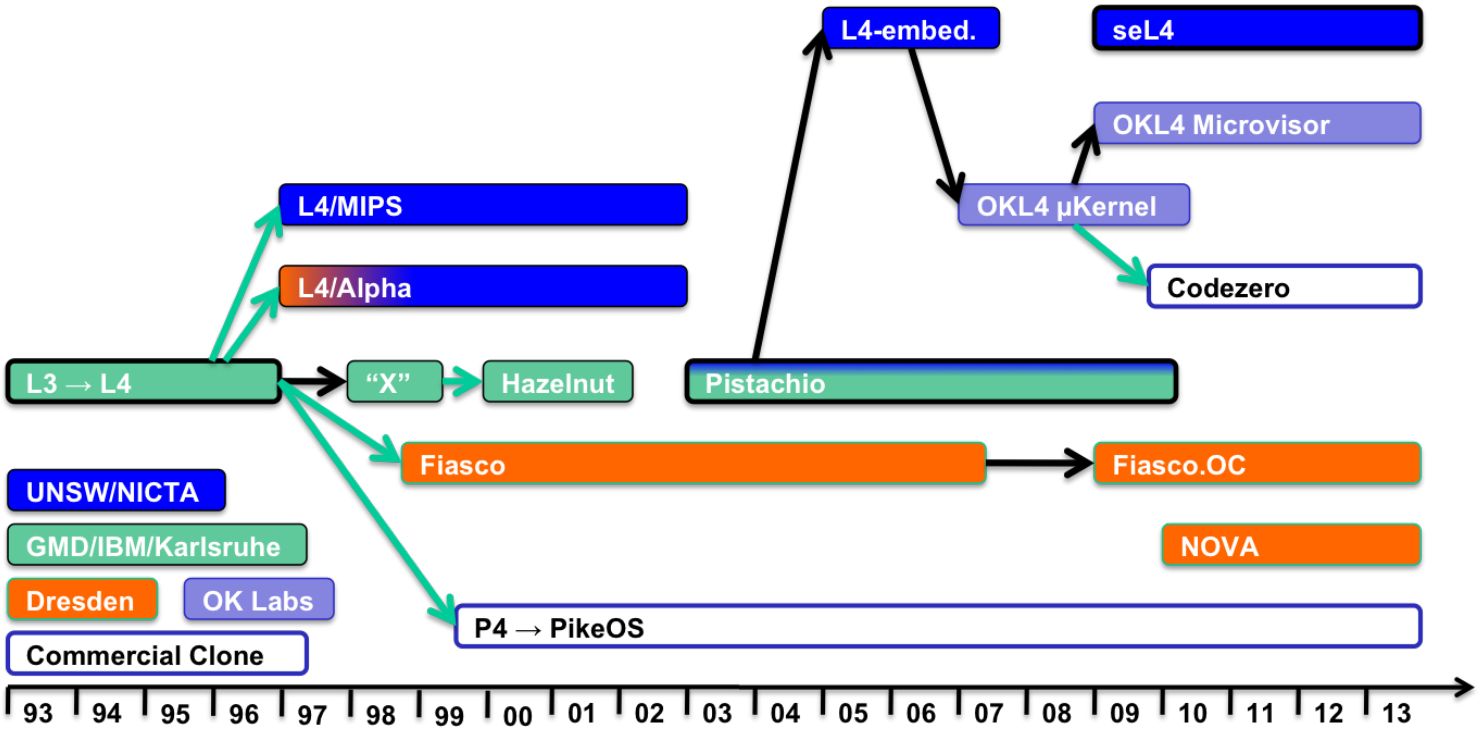
- Tâche
  - Elément de base de l'allocation de ressources
  - Espace d'adressage virtuel, communication
- Thread
  - Elément de base d'exécution
- Port
  - Canal de communication des IPC
- Message
  - Elément d'échange avec le noyau
  - Contient les informations concernant les ports, pointeurs, ...
- Accès mémoire

# Plan de la présentation


1. Introduction
2. Les IPC
3. Le micronoyau Mach
4. La famille L4
5. Au final

# La famille L4

- 1ère version créée par Joshen Liedtke (1993)
- Objectif : améliorer les performances des micro-noyaux




Source : "L4 Microkernels : The Lessons from 20 Years of Research and Deployment", GERNOT HEISER and KEVIN ELPHINSTONE, NICTA and UNSW, Sydney, Australia

# Les concepts de L4

- Jochen Liedtke :
  " A microkernel does no real work"
  - Le noyau ne réalise que les opérations fondamentales
- Qu'est-ce qui est fondamental ?
  - Abstractions
    - ★ Threads
    - ★ Espaces d'adressage (Tâches)
  - Mécanismes
    - ★ Communication
    - ★ Allocation de ressources
    - ★ Ordonnancement

# Principes de L4

- Dans P4, tout est un objet
  - Tâche — Espaces d'adressage
  - Thread — Exécution, ordonnancement
  - IPC Gate — Communication, gestion des ressources
  - IRQ — Communication
  - Factory — Création d'autres objets, respect des quotas d'accès aux ressources
- Un appel système : **invoke_object()**
  - Paramètres dans le descripteur de processus en espace utilisateur (UTCB)
  - Les paramètres dépendent de l'objet

# Gestion des IPC

- Un objet pour la communication : IPC Gate
- Inter-process communication (IPC)
  - Entre les threads
  - Synchrone
- Communication via IPC gate
  - Le thread émetteur poste son message dans son UTCB
  - Il fait appel à IPC gate, son exécution est stoppée en attente de la réception
  - Le noyau copie le message dans l'UTCB du thread récepteur
  - L'émetteur et le récepteur continue lors exécution

# L'objet Threads

- Thread
  - Unité d'exécution
  - Objet du noyau
- Propriétés gérées par le noyau
  - Pointeur d'instruction
  - Pile
  - Registres
  - UTCB

# Les interruptions

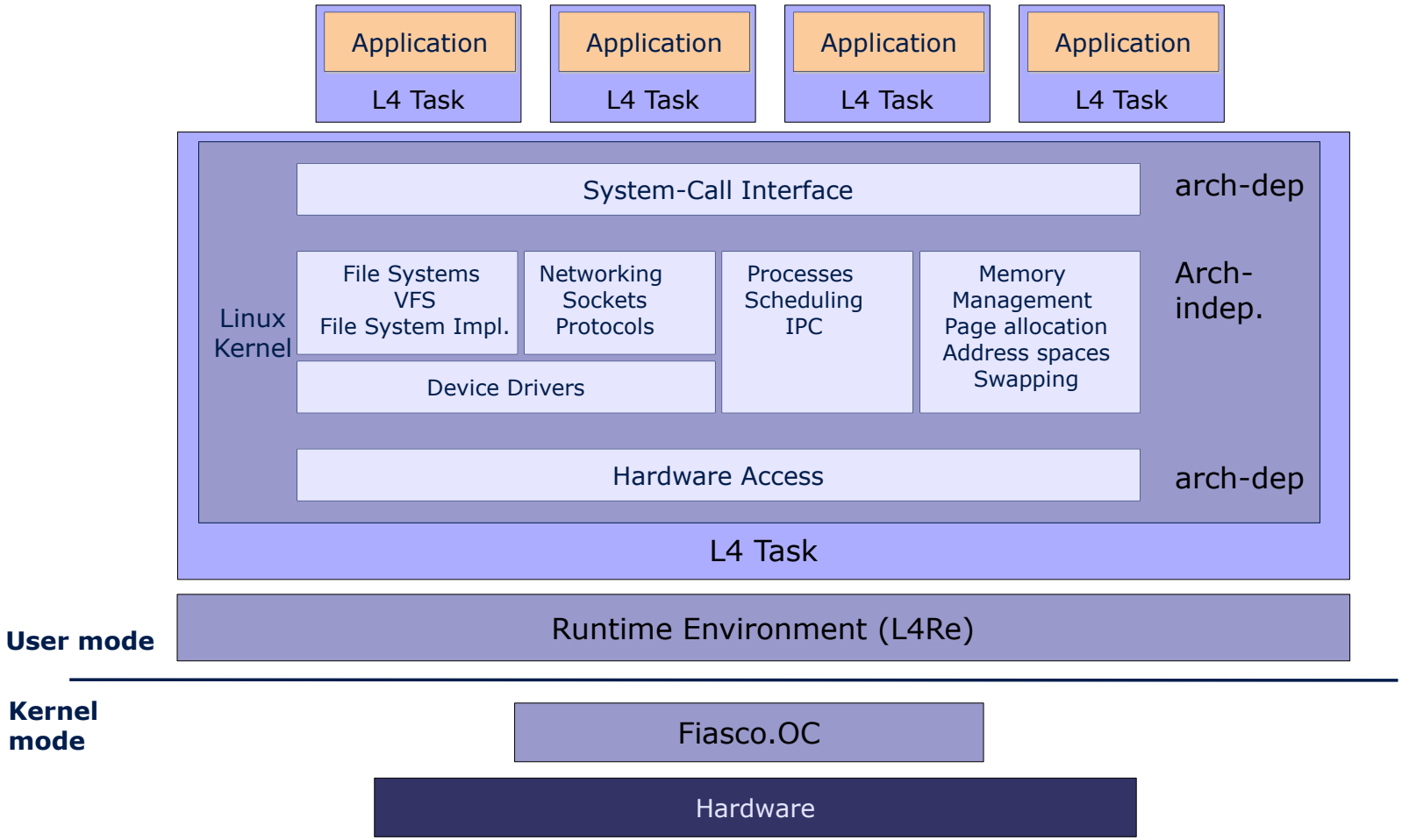
- L'objet du noyau IRQ
- Utilisé pour les interruptions matérielles et logicielles
- Permet la gestion asynchrone :
  - invoke_object(irq_cap, WAIT)
  - invoke_object(irq_cap, TRIGGER)

# Utilisation de L4

## L4Linux

- Dans Linux, 2 parties séparées :
  - Une partie dépendante de l'architecture
  - Une partie indépendante
- Pour L4Linux
  - La partie dépendante de l'architecture est modifiée pour L4
  - La partie indépendante n'est pas changée
  - L4 n'a pas été modifié pour Linux
- Le noyau Linux est un service utilisateur de L4
  - Il s'exécute comme un thread L4
  - Il crée des threads L4 pour ces processus
  - Gère l'espace d'adressage des threads utilisateur

# Architecture L4Linux



# Utilisation de L4

## L4Android

- Android s'exécute dans une machine virtuelle
- La machine virtuelle est un thread pour L4
- Permet une meilleure sécurisation
- Plusieurs applications sont possibles
  - Mobile professionnel et personnel à la fois
  - Utilisation d'une HAL et Android implante une seule architecture

# Utilisation de L4

**OKL4**

- Développé par OK Labs
- Utilisé dans les puces Qualcomm
- Intégré à la majorité des téléphones mobiles
- Sécurisation

**seL4**

- 3ème génération de micro-noyau
- Système vérifié
- Applications dans la téléphonie, les systèmes embarqués

# Plan de la présentation


1. Introduction
2. Les IPC
3. Le micronoyau Mach
4. La famille L4
5. Au final

# Noyaux monolithiques / micro-noyaux

**Qui a gagné ?**