CS 5460/6460
Operating Systems

Fall 2009

Instructor: **Matthew Flatt**

Lecturer: **Kevin Tew**

TAs: Bigyan Mukherjee, Amrish Kapoor

Reminders

- HW1 Due: Tuesday, September 8th, 2009 9:40am
  - Submit a single C file using handin on a CADE machine: handin cs5460 hw1 <file>
- Join the Mailing List!
- Make sure you can log into the CADE machines

OS Structures & Services

OS Components:

- Processes
- Synchronization
- Memory & Secondary Storage Management
- File Systems
- I/O Systems
- Distributed Systems

Three example OS organizations

- Monolithic kernel
- Microkernel
- Exokernel

From the Architecture to the OS to the User

Hardware	Example OS Services	User Abstraction
Processor	Process management, Scheduling, Traps, Protection, Billing, Synchronization	Process
Memory	Management, Protection, Virtual memory	Address space
I/O devices	Concurrency with CPU, Interrupt handling	Terminal, Mouse, Printer, (System Calls)
File system	Management, Persistence	Files
Distributed systems	Network security Distributed file system	RPC system calls, Transparent file sharing

Processes

- The OS manages a variety of activities:
  - User programs
  - Batch jobs and command scripts
  - System programs: printers, spoolers, name servers, file servers, network listeners, etc.
- Each of these activities is encapsulated in a process.
- A process includes the execution context (PC, registers, VM, resources, etc.) and all the other information the activity needs to run.

Processes

A process is not a program. A process is one instance of a program in execution. Many processes can be running the same program. Processes are independent entities.

- The OS creates, deletes, suspends, and resumes processes.
- The OS schedules and manages processes.
- The OS manages inter-process communication and synchronization.
- The OS allocates resources to processes.

All activity on a computer is either in the OS or in a process.

Synchronization Example:

Main memory

- Cooperating processes on a single account:
    - ATM transaction
    - balance computation
    - monthly interest computation and addition

Synchronization Example:

Main memory

- Cooperating processes on a single account:
  - ATM transaction
  - balance computation
  - monthly interest computation and addition

Question:

All of the processes are trying to access the same account simultaneously. What can happen?

Memory & Secondary Storage Management

Main memory

- is the direct access storage for the CPU.
- Processes must be stored in main memory to execute.
- The OS must:
  - allocate memory space for processes,
  - recover all process memory upon termination,
  - maintain the mappings from virtual to physical memory (page tables),
  - keep track of memory pages that are swapped to disk,
  - decide when to refuse a request for more memory from a process (allocation policies).

File System

Secondary storage devices (disks) are too crude to use directly for long term storage.

- The file system provides logical objects and operations on these objects (files).
- A file is the long-term storage entity: a named collection of persistent information that can be read or written. ("Persistent" means available across reboots.)
- File systems support directories which contain the names of files and other directories along with additional information about the files and directories (e.g., when they were created and last modified).

File System

The File System provides **file management**, a standard interface to

- create and delete files and directories
- manipulate (read, write, extend, rename, copy, protect) files and directories
- other services: filesystem-based security (permissions), backups, quotas, etc.

The File System must efficiently map files and directories onto disk blocks (the unit of secondary storage).

Secondary Storage (disk)

Secondary Storage is

- the persistent memory, i.e., it survives system reboots and crashes (we hope).
- Low-level OS routines are typically responsible for low-level disk functions, such as scheduling of disk operations, head movement, and error handling.
- These routines may also be responsible for managing the disk space (for example, keeping track of the free space).
- The line between managing the disk space and the file system is very fuzzy, these routines are sometimes in the file system.

Example: A program executable is stored in a file on disk. To execute a program, the OS must load the program from disk into memory.

I/O Systems

The I/O system supports communication with external devices: terminal, keyboard, printer, mouse, ...

The I/O system

- Supports buffering and spooling of I/O

  These help compensate for the speed differential between processors and I/O devices.

- Provides a general device driver interface, hiding the differences among devices

  In Unix many devices look like files.

- Provides device driver implementations specific to individual devices.

  Typically the majority of OS code is in device drivers.

Distributed Services

A distributed system is a collection of processors that do not share memory or a clock.

- To use non-local resources in a distributed system, processes must communicate over a network,
- The OS must provide additional mechanisms for dealing with failures and deadlock that are not encountered in a centralized system.

The OS can support a distributed file system on a distributed system.

- Users, servers, and storage devices are all dispersed among the various sites.
- The OS must carry out its file services across the network and manage multiple, independent storage devices.

Putting Services in Context

The point of the OS is to provide services. This does not mean that all services go in the OS because there are other ways to provide services:

- Subroutine library
    - Examples: printf, graphics library, numerical library
    - Advantages: very efficient and flexible
    - Disadvantages: sharing between processes and machines is harder, library can’t be protected from user, somewhat language dependent
- OS kernel
    - Examples: TCP/IP, interprocess communication, file access
    - Advantages: efficient, secure, language independent. can provide coordination between clients
    - Disadvantages: adds complexity and overhead
- Process
    - Examples: print server, web server, logging facility
    - Advantages: can be cleanly terminated independently of clients, can provide coordination between clients
    - Disadvantages: good performance is more difficult

Networks and programming languages also provide services.

Where to put each service?

Ideally, each service is implemented in the best possible way where "best" is some combination of:

- Most efficient
- Easiest to implement
- Most flexible
- Most reliable
- Most secure
- Most portable
- How it used to be done

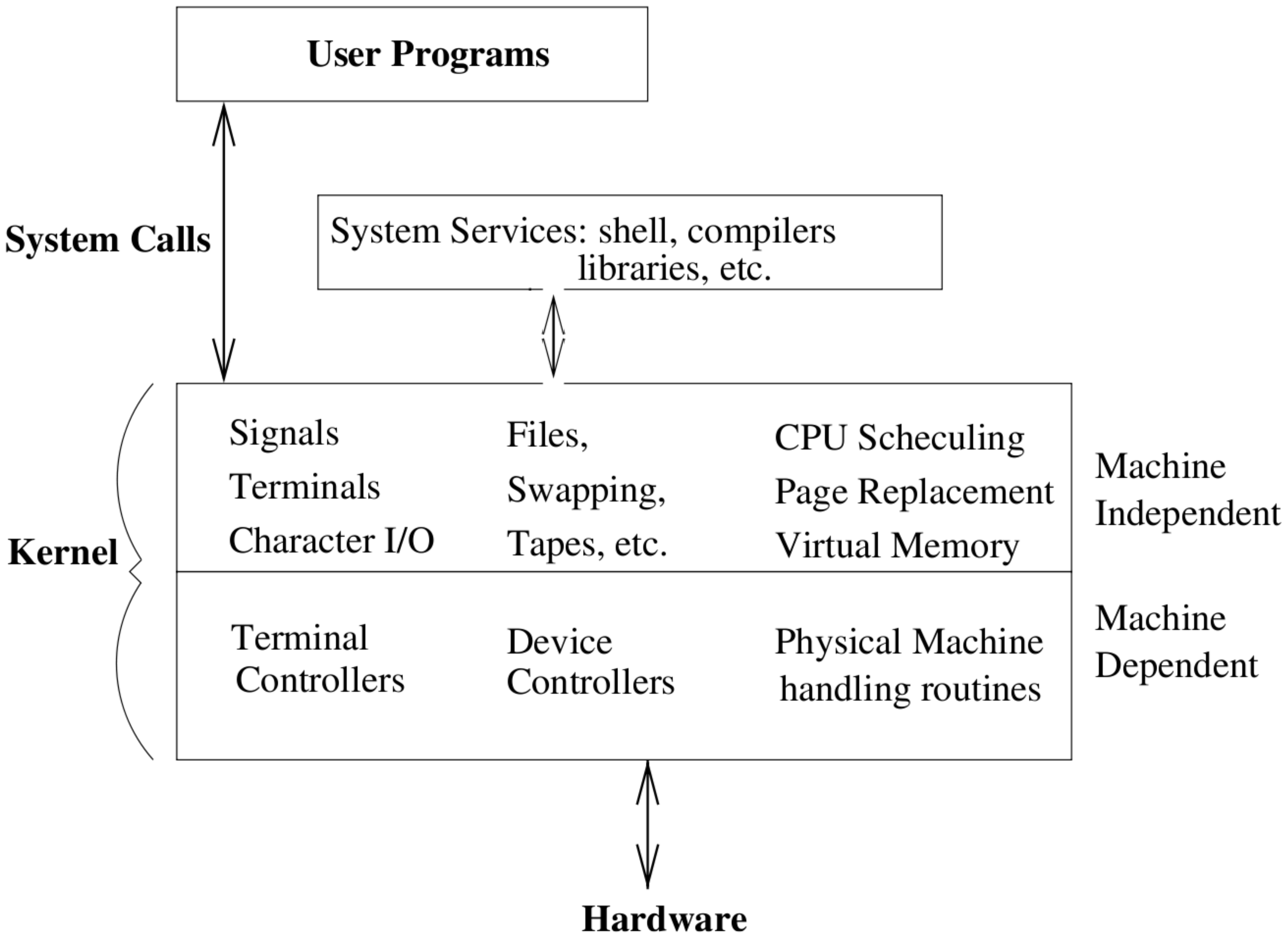
There are always tradeoffs! For example:

- Putting the windowing system in the kernel is more efficient (Windows) but putting it in a process is more flexible and more reliable (Unix and X)
- Putting network support in a library is more efficient than in the kernel (for very fast network interfaces) but this makes it difficult for multiple processes to share the interface

OS Organization Influences Service Placement

- Monolithic kernel: more services implemented in the OS kernel
- Microkernel: more services implemented in processes
- Exokernel: more services implemented in libraries

Monolithic Kernel

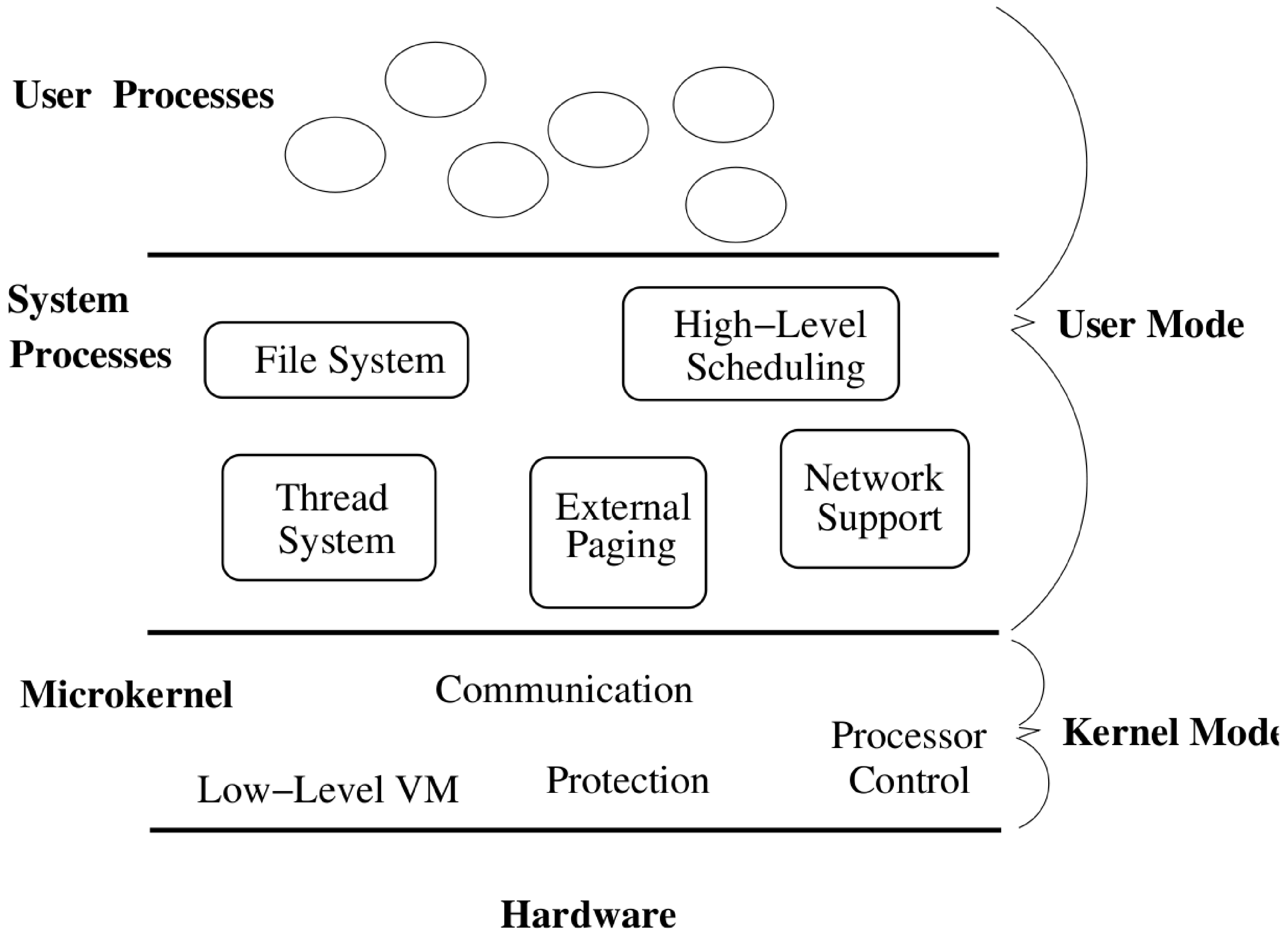


Monolithic Kernel

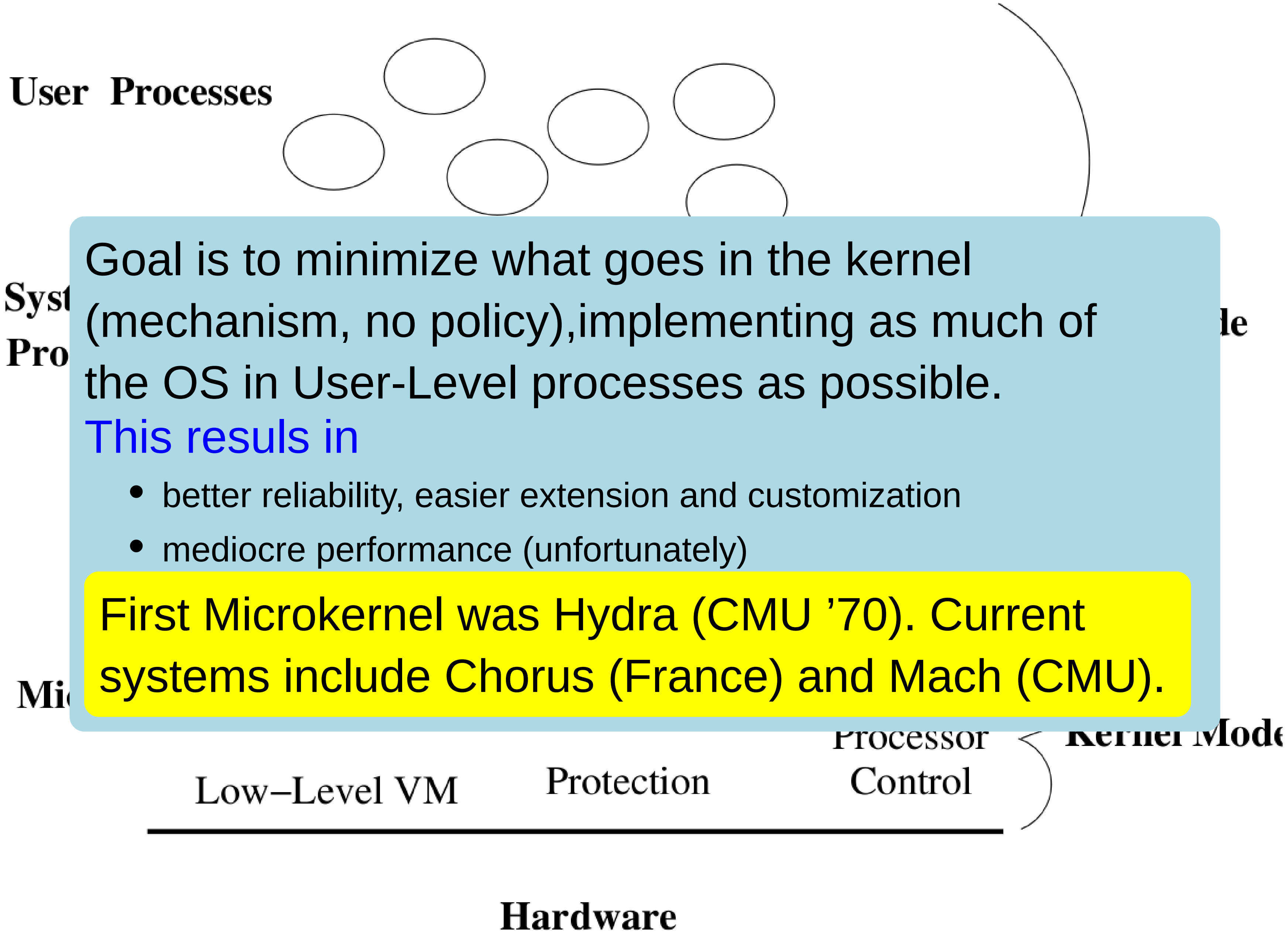
The kernel is the protected part of the OS that runs in kernel mode, protecting the critical OS data structures and device registers from user programs.

There is a lot of debate about what functionality should go into the kernel – this is just one possible organization, used by original Unix OS (Windows 2000 and XP are not that different).

Microkernel



Microkernel



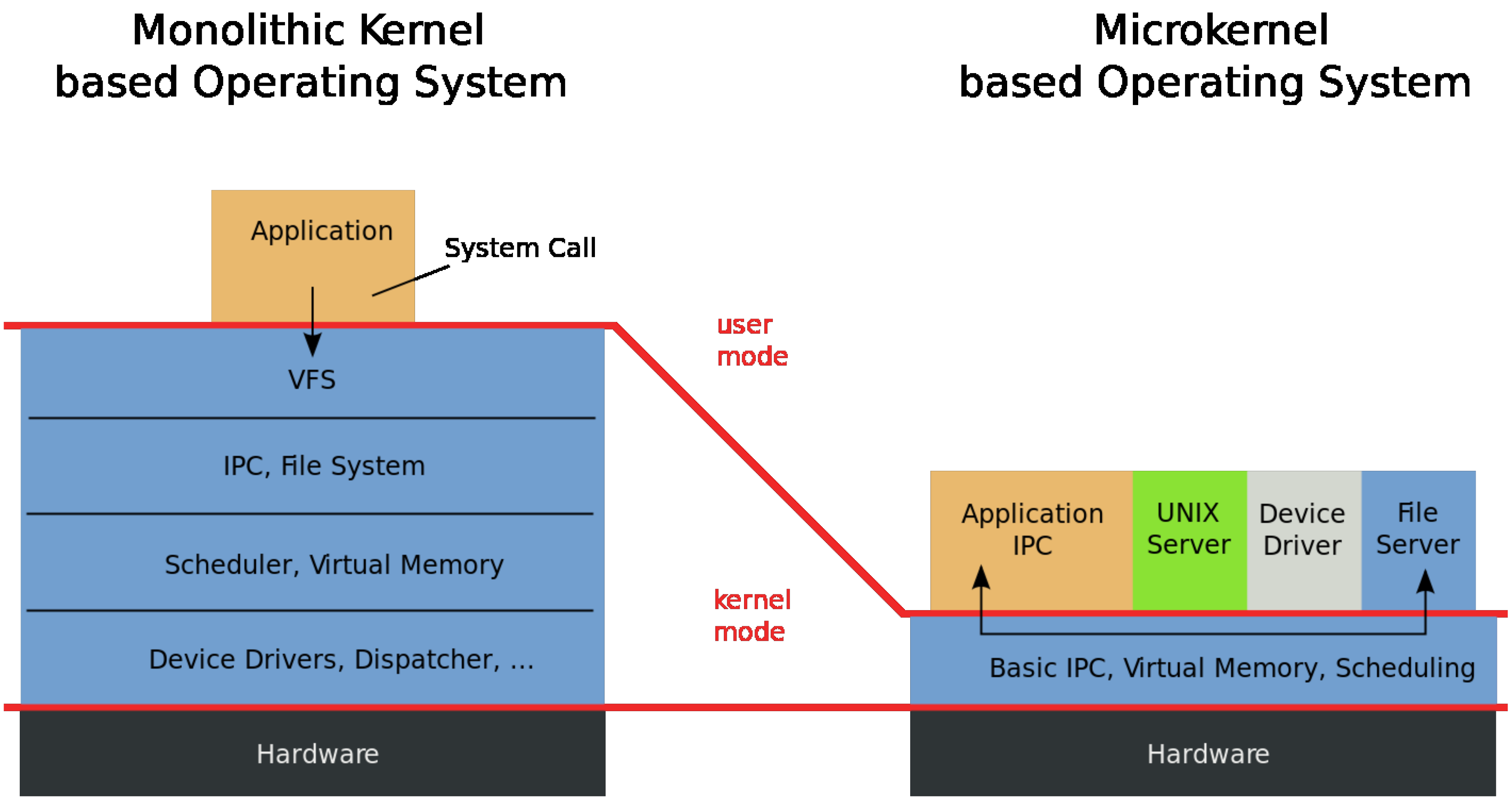
Goal is to minimize what goes in the kernel (mechanism, no policy),implementing as much of the OS in User-Level processes as possible.

This resuls in

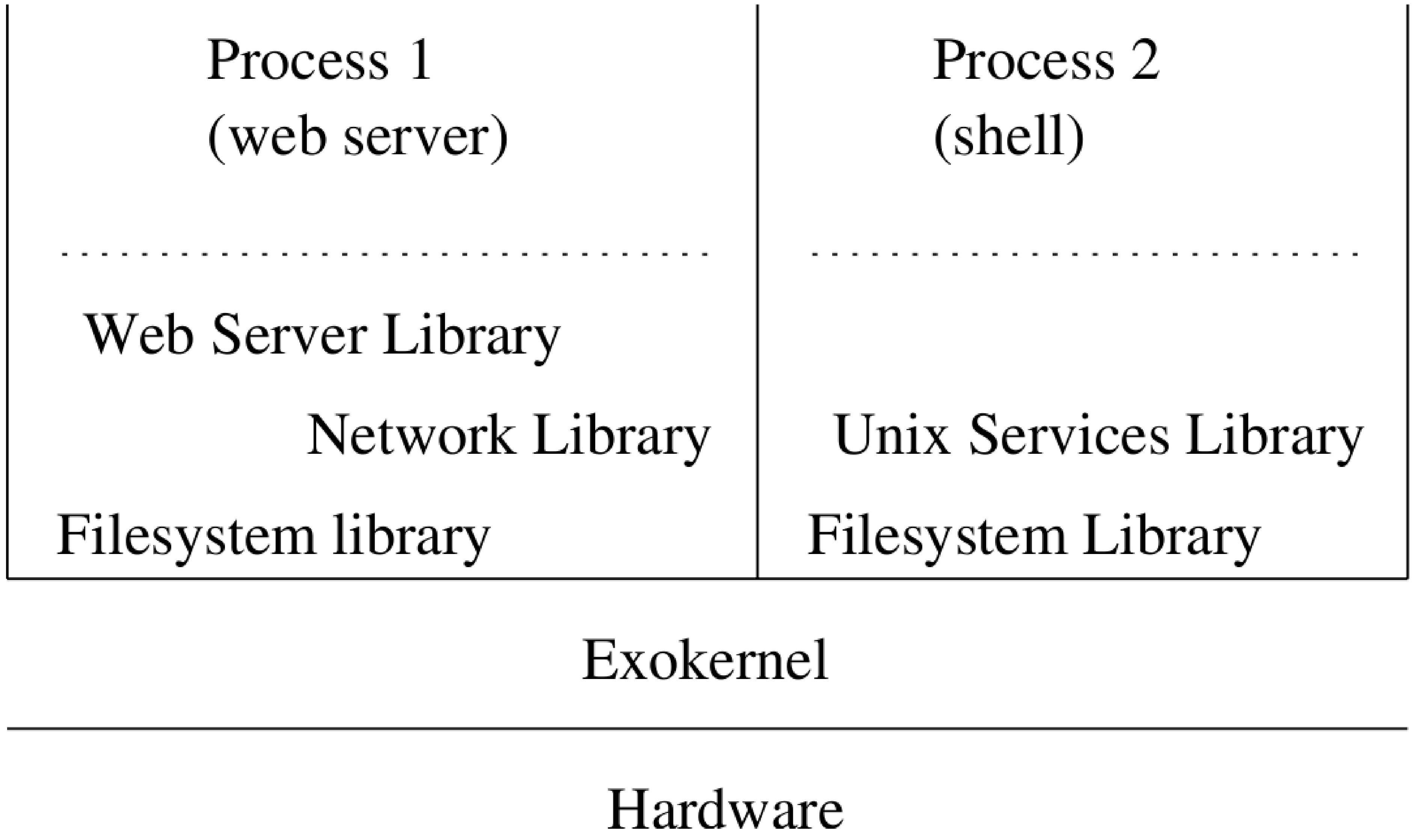
- better reliability, easier extension and customization
- mediocre performance (unfortunately)

First Microkernel was Hydra (CMU ’70). Current systems include Chorus (France) and Mach (CMU).

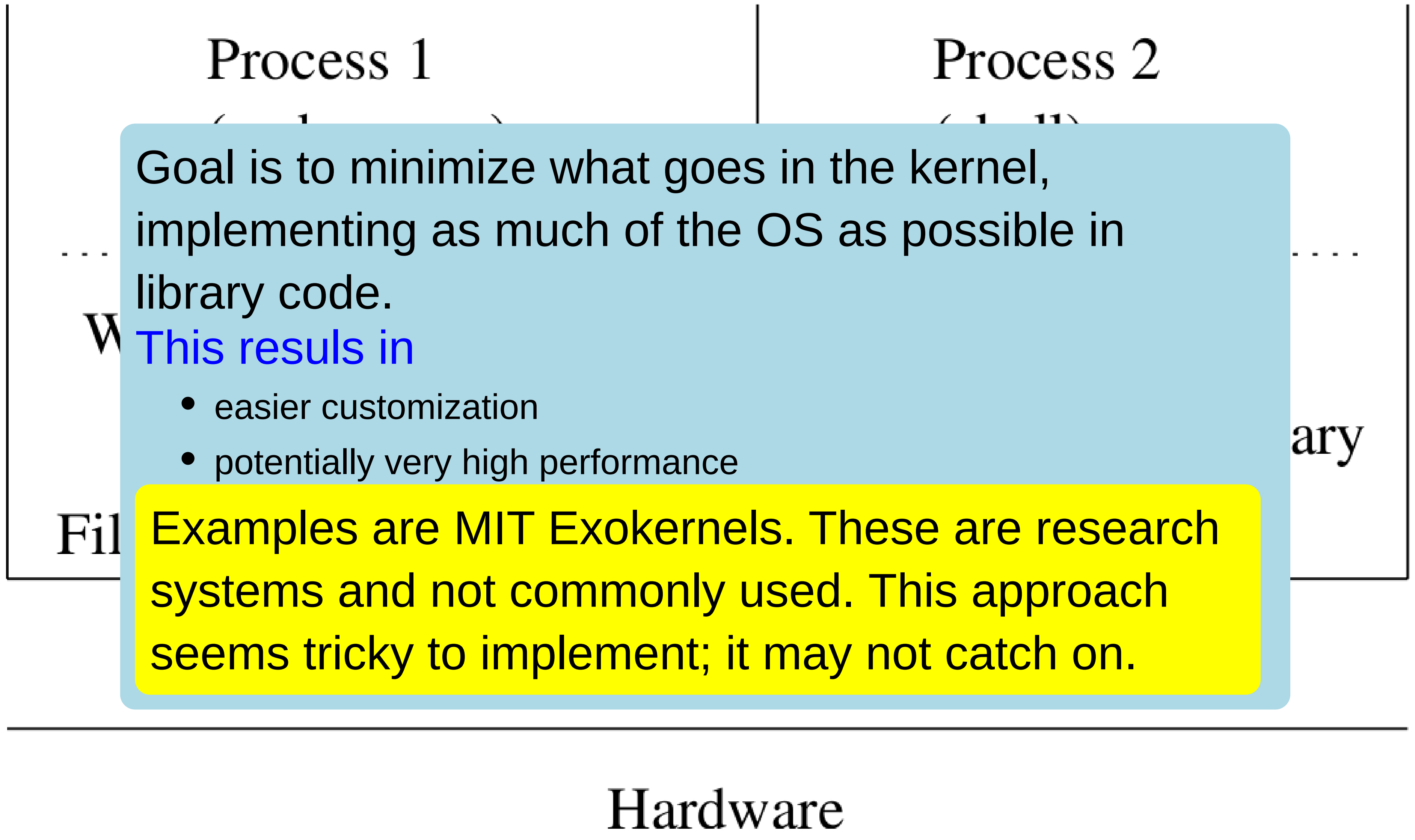
Monolithic vs Microkernel


Monolithic Kernel based Operating System
Microkernel based Operating System
Application
System Call
VFS
IPC, File System
Scheduler, Virtual Memory
Device Drivers, Dispatcher, ...
Hardware
user mode
kernel mode
Application IPC
UNIX Server
Device Driver
File Server
Basic IPC, Virtual Memory, Scheduling
Hardware

Exookernel


Process 1
(web server)
Process 2
(shell)
Web Server Library
Network Library
Unix Services Library
Filesystem library
Filesystem Library
Exokernel
Hardware

Exookernel



Goal is to minimize what goes in the kernel, implementing as much of the OS as possible in library code.

This resuls in

- easier customization
- potentially very high performance

Examples are MIT Exokernels. These are research systems and not commonly used. This approach seems tricky to implement; it may not catch on.

Summary

Big Design Issue: How do we make the OS efficient, reliable, and extensible?

Difficult because there is considerable leeway in how services are implemented!

General OS Philosophy: OS design involves a constant tradeoff between simplicity and performance. As a general rule, strive for simplicity except when you have a strong reason to believe that you need to make a particular component complicated to achieve acceptable performance (strong reason = simulation or evaluation study)