Lecture # 3.1
Operating System Structures

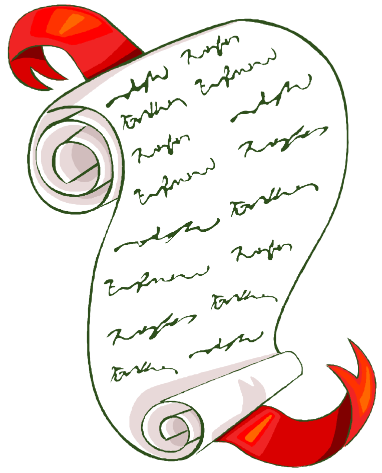
Course: Advanced Operating System

Instructor: Arif Butt

Punjab University College of Information Technology (PUCIT)
University of the Punjab

Today's Agenda

- Introduction to OS Structures
- Monolithic Structure
- DOS like Structure
- Microkernel Structure
- SPIN Approach
- Exokernel Approach
- L3 Microkernel Approach



Overview of OS Structure

Operating System Structure

- Operating System is a large and complex program and therefore must be engineered carefully, if it is to function properly and to be modified easily
- Operating System Structure means the way an operating system is organized with respect to the applications that it serves and the underlying hardware that it manages
- There are different design alternatives that appear in literature:
  - Monolithic Structure
  - DOS like Structure
  - Microkernel Approach
  - SPIN Approach (An OS kernel developed by University of Washington)
  - Exokernel Approach (An OS kernel developed by MIT)
  - L3 Microkernel Approach
  - THE (Technische Hogeschool Eindhoven, an OS developed by Dijkstra)

Goals of Operating System Structure

- **Protection:** Within and across users and the OS itself
- **Performance:** Time taken to perform the services requested by applications
- **Flexibility/Extensibility:** A service is extensible or adaptable to the requirements of applications
- **Scalability:** As you add more h/w resources, the performance goes up
- **Agility:** How quickly the OS changes/adapts itself to match the requirements of apps or resources availability
- **Responsiveness:** How quickly the OS reacts to external events, e.g., mouse clicks in a video game

Some of these goals seems conflicting with each other, e.g., to achieve performance we may have to sacrifice protection and flexibility

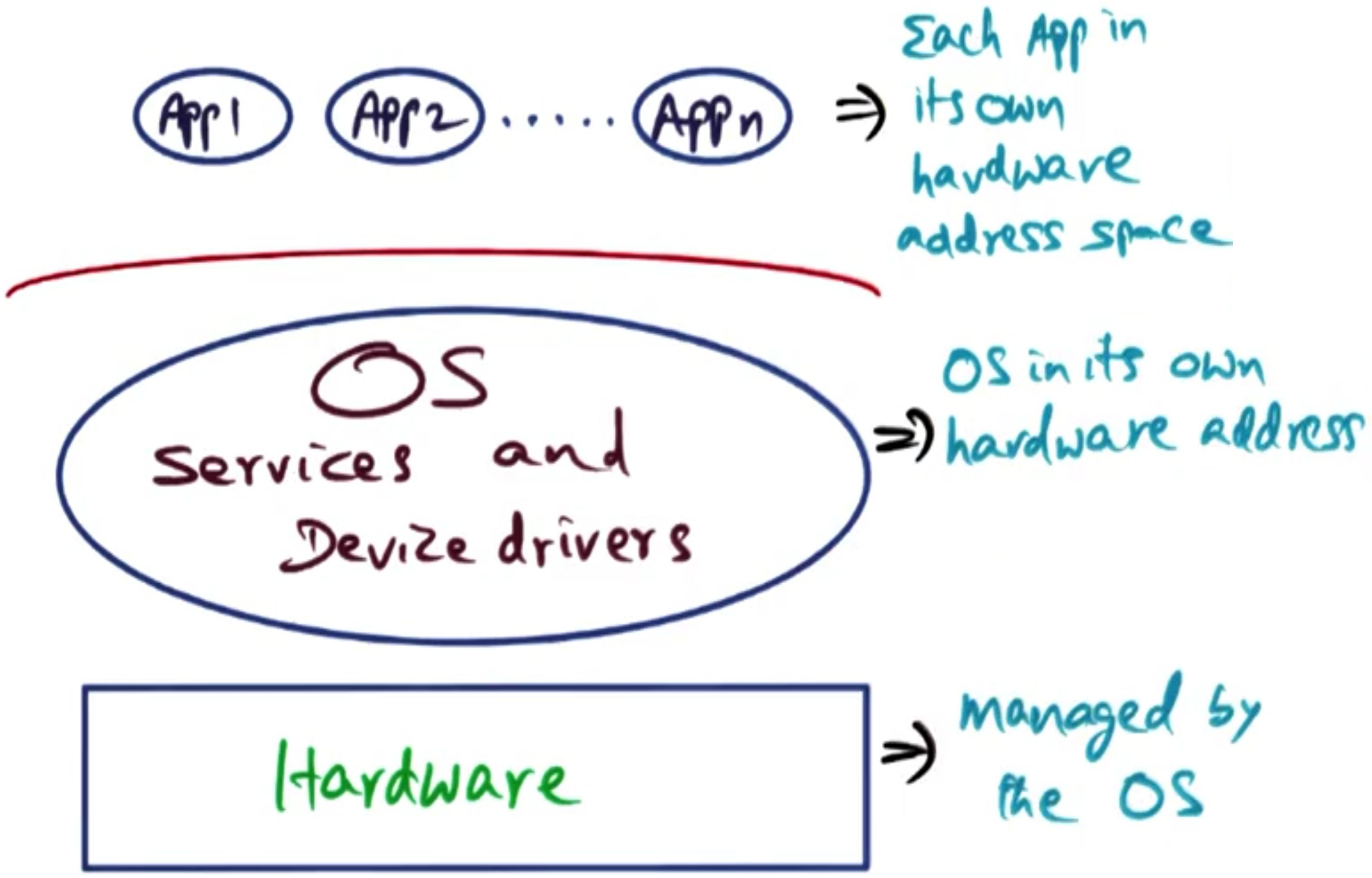
100$ Question: Are all these goals simultaneously achievable in a given OS?

Monolithic Structure

Monolithic Structure

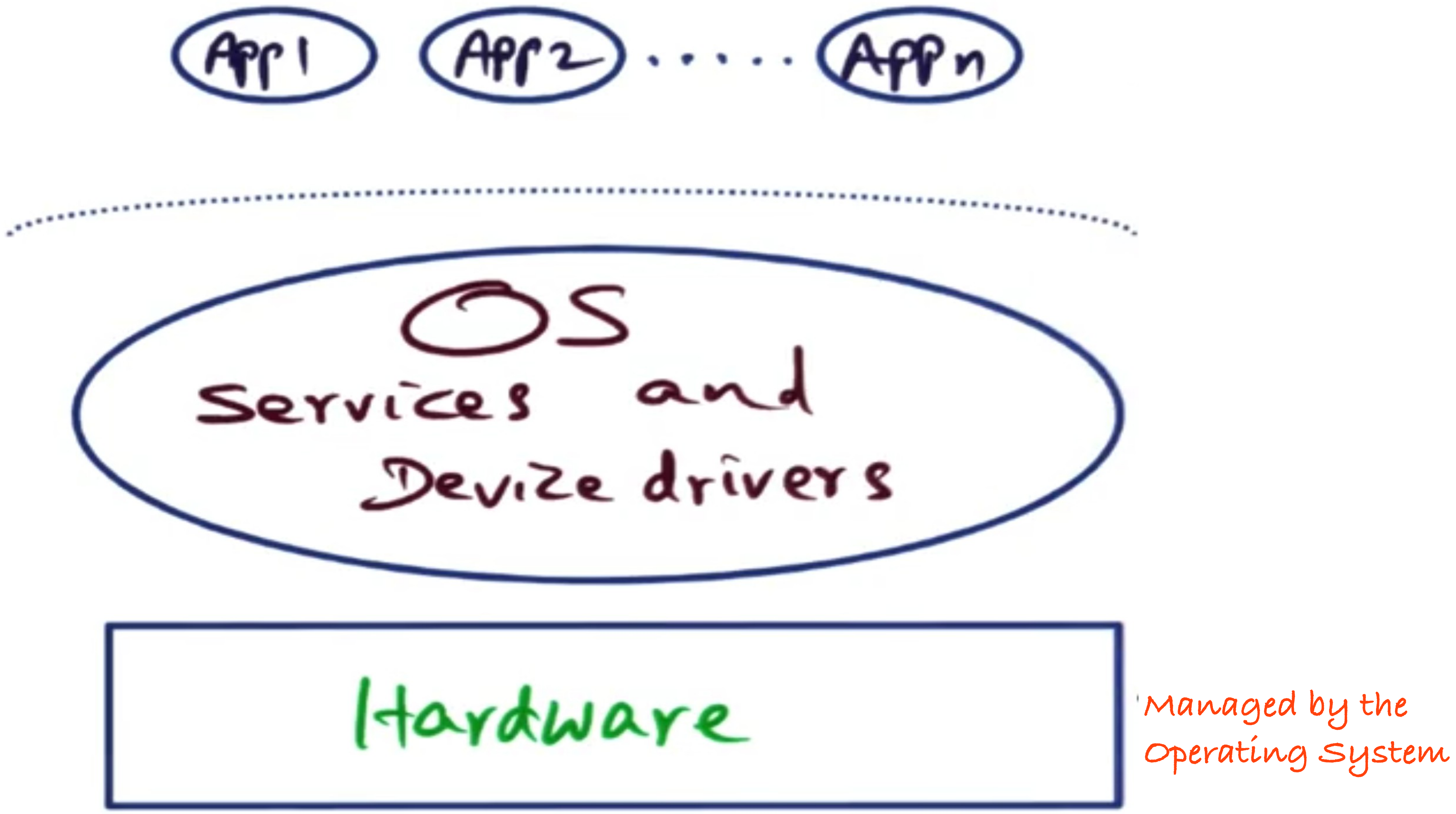
- The hardware at the bottom is managed by the Operating System, which include the CPU, memory and peripheral devices
- The applications are at the top and each application runs in its own hardware address space. So every application is protected from one another because the h/w ensures that the address space occupied for one application is different from the other
- The kernel provides the file system, CPU scheduling, memory management, and other operating system services through system calls
- An application can request the services provided by the OS by placing the parameters in a well-defined place and then executing a trap machine instruction to switch from user mode to kernel mode

Monolithic Structure


App1
App2
.....
Appn
⇒ Each App in its own hardware address space
OS
Services and Devize drivers
⇒ OS in its own hardware address
Hardware
→ managed by the OS

DOS Like Structure

DOS Like Structure



DOS like Structure (cont...)

What is gained with DOS like Structure

- **Performance**: Since the applications and the OS live in the same address space, so access to system services is just like making a procedure call. (Note the dotted line between application layer and OS)

What is lost with DOS like Structure

- **Protection**: There is no protection between the applications and the operating system. So the integrity of an OS can be compromised by a runaway application either maliciously or unintentionally, corrupting the OS data structures

Why DOS choose this Structure?

- At those early days it was for a single user and single application, so protection was not the primary concern
- Performance and simplicity were the key concerns

Monolithic vs DOS Structures

- In DOS like structure, there is **no protection**, which is unacceptable for a general purpose OS these days
- Monolithic structure gives the protection, but at the same time **reduces performance** by consolidating all the OS services in one big monolithic structure. So the application has to move from its own address space to OS address space to get services. To provide a service to an application different components of operating system may also have to talk to one another. For Example, suppose a process make a call to file system to open a file. The file system in turn may have to call the storage module in order to find where the file is residing. Later the file system may have to call the memory management module and request it to load the contents of file at an appropriate location in memory
- Another downside of monolithic structure is that it **shuts down the ability of customizing** the OS services to meet different needs of different applications. (It take the approach of one size fits all)

Monolithic vs DOS Structures (cont...)

100$ Question: Why do we need to customize OS services for different applications, why not one size fits all. To understand this, consider the example of a video game application and an application that computes all the prime numbers. These are two different classes of applications with different needs. For the video game the key determinant of a good OS is responsiveness, ie., how quickly the OS is responding to the clicks of the joy stick. For the prime number computing app the key determinant of a good OS is the amount of cpu time that is available for number crunching

Opportunities for Customization in an OS?

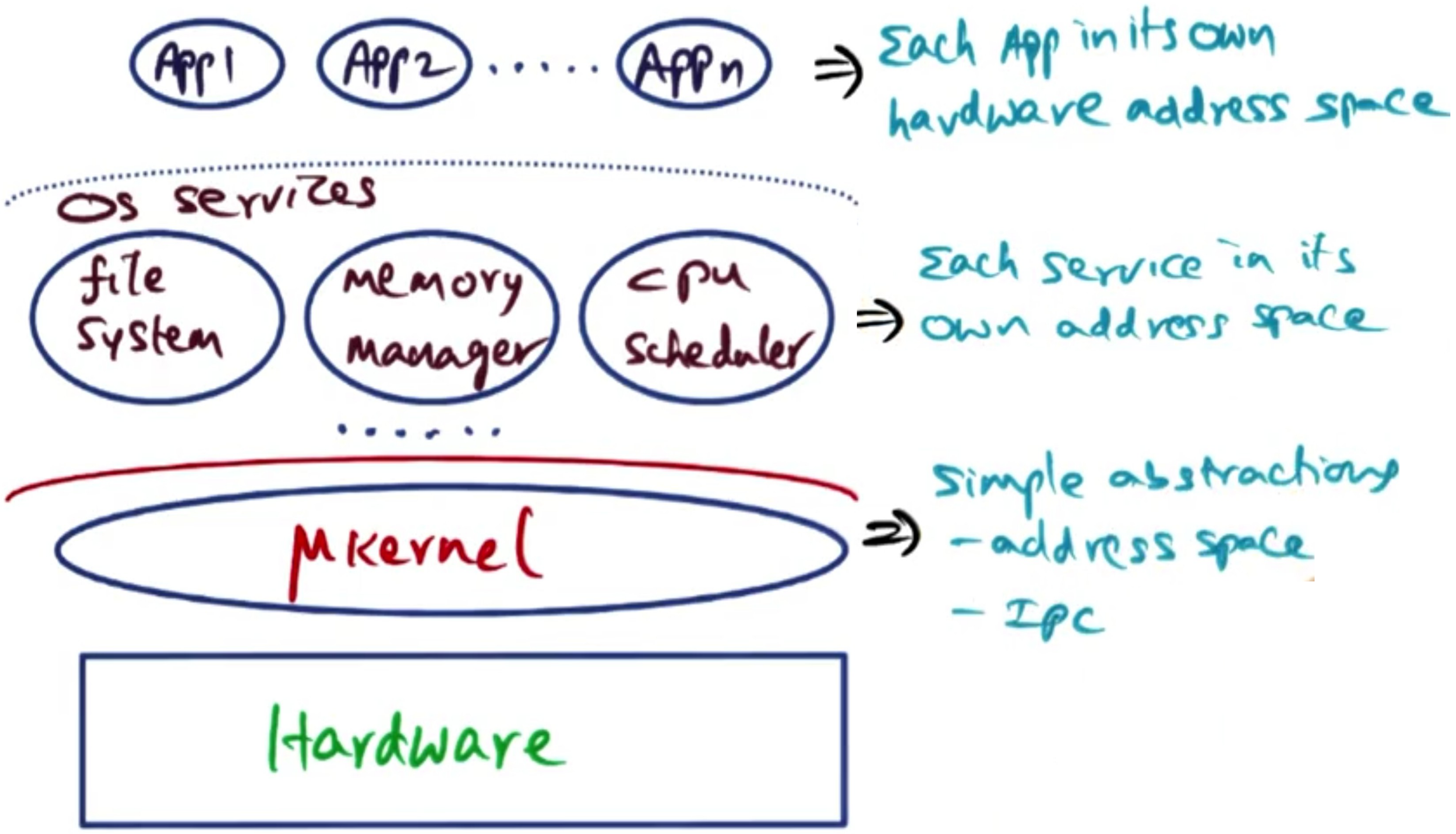
- Page Replacement algorithm in memory management module
- How OS schedules processes on the CPU?
- How OS reacts to external events and interrupts?

Microkernel Based OS Structure

Microkernel Based OS Structure

- The initial versions of UNIX used the monolithic approach and later a bit of layered approach. But as time passed the UNIX kernel became large and difficult to manage
- The word “kernel” is used to denote the part of the OS that is mandatory and common to all other softwares
- In mid 1980s, researchers at CMU developed an OS called “Mach” that modularized the kernel using the microkernel approach. This method structures the OS by removing all nonessential components form the kernel and implementing them as system and user-level programs
- The OS services such as MMU, CPU scheduler, file system, are implemented as servers above the microkernel. So the system services executes with same privileges as the apps and runs in their own addr space
- So there exist very strong protection among the applications, between the applications and the system services, among the system services, and between applications, system services and the microkernel
- The main task of the microkernel is to provide a communication facility between apps and OS services and among the OS services via IPC

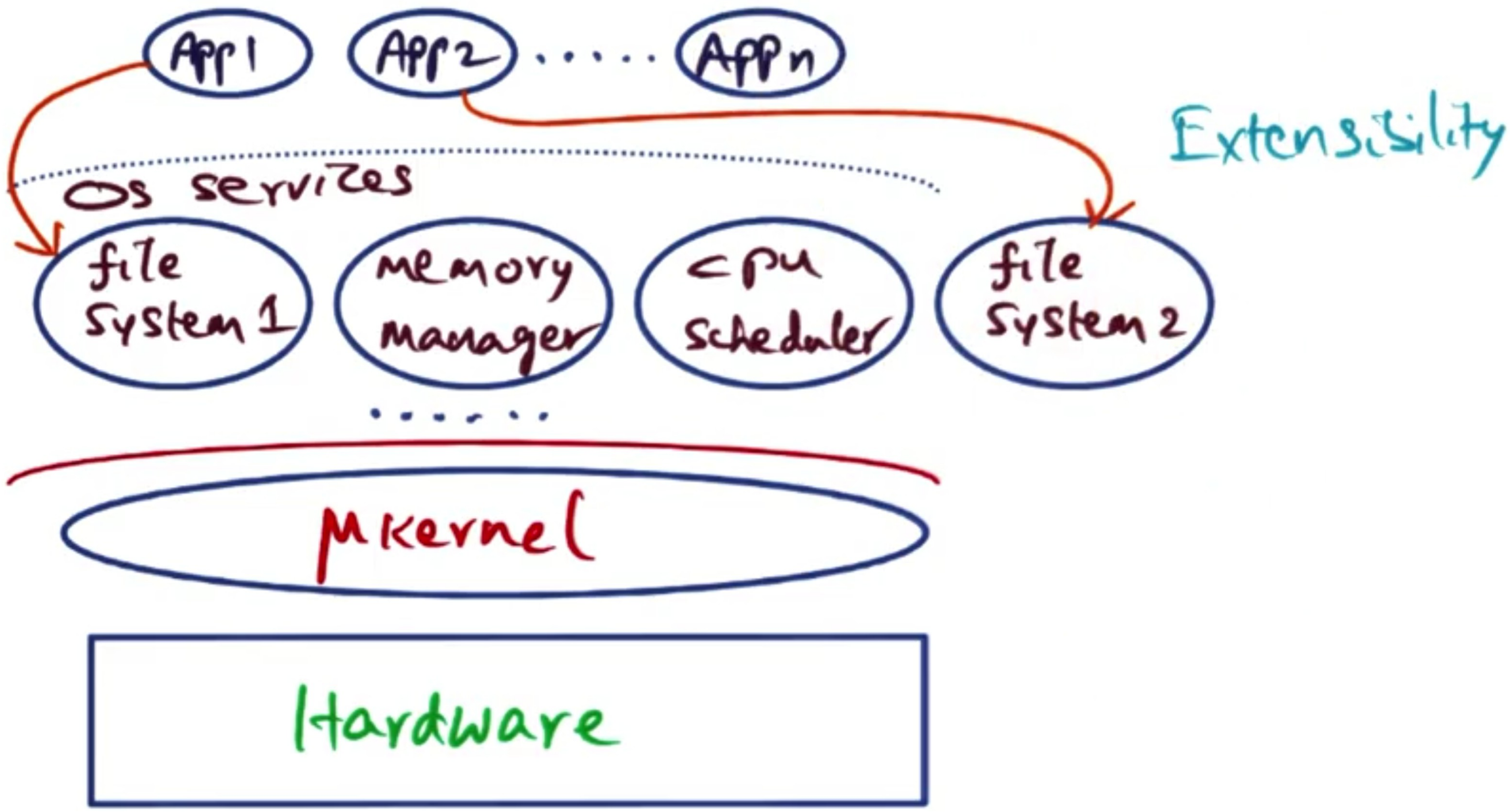
MicroKernel Based OS Structure



Advantages of Microkernel Structure

- One of the advantage of microkernel approach is separation of policy from mechanism. Policies are likely to change across places over time so they are implemented as OS services. Mechanisms are defined in the OS kernel. For example, the timer construct is a mechanism for ensuring CPU protection, but deciding how long the timer is to be set for a particular user is a policy decision. Similarly, we can have two different file systems to serve different applications
- Extensibility: It is easy to extend a service or write a new service in microkernel based approach. A new service can be added to user space and consequently do not require modification of the kernel. If a service fails, the rest of the operating system remains untouched

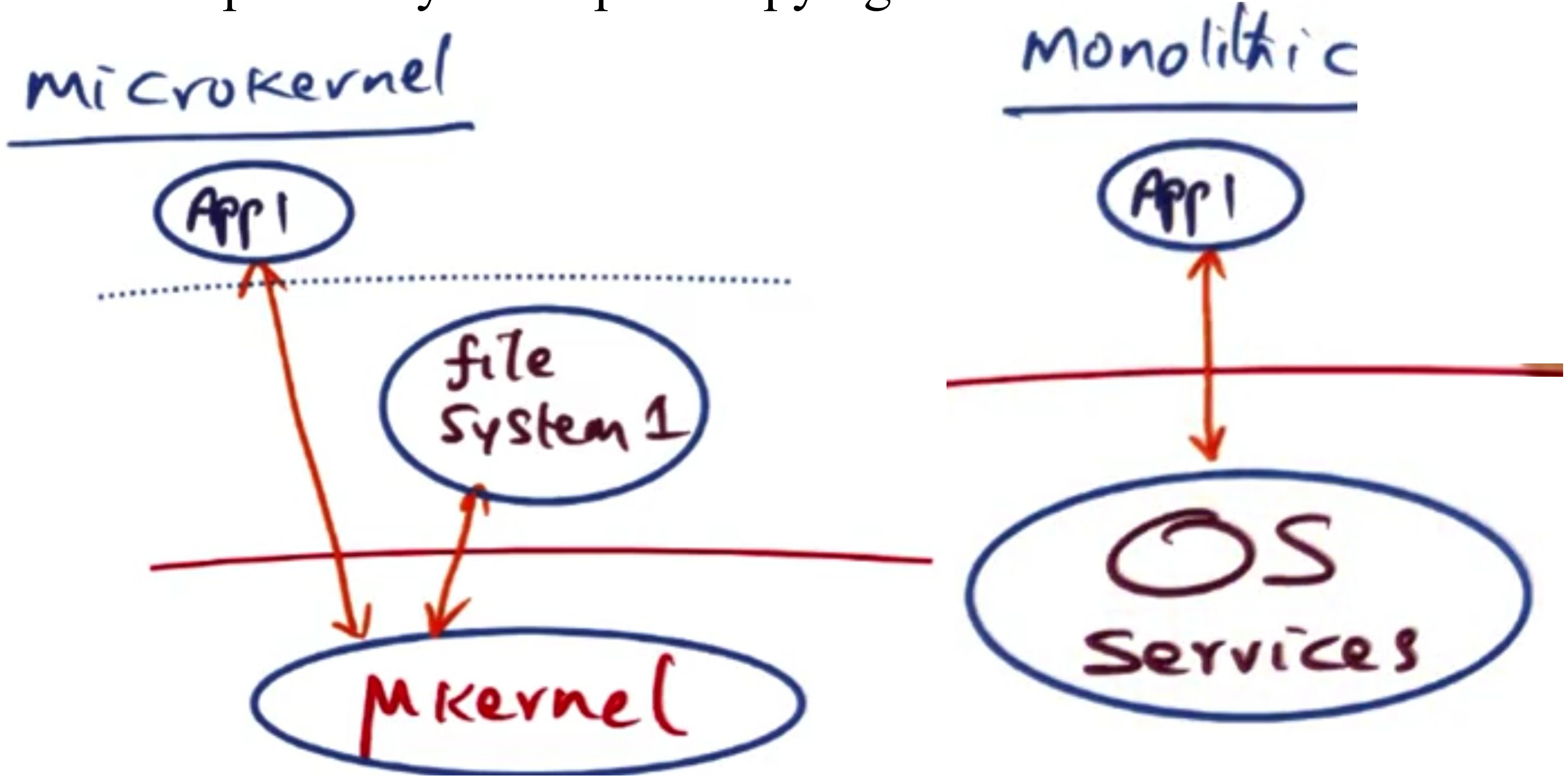
Advantages of MicroKernel based Design



Downside to Microkernel Based Design

Potential for performance loss due to border crossings

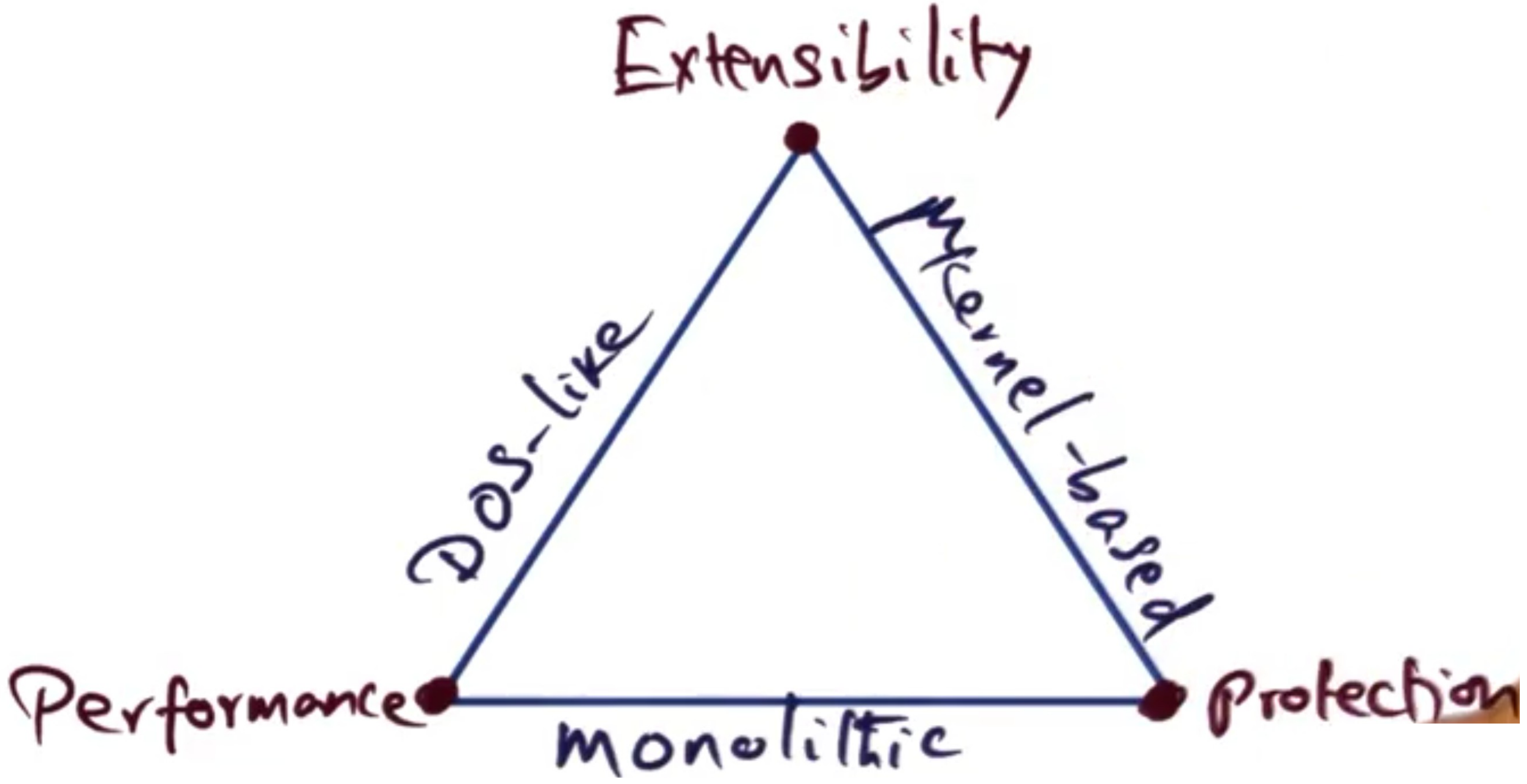
- Change in locality
- User space :: System Space copying



Comparison

Feature	monolithic OS	DOS-like OS	μkernel OS
Extensibility	✗	✓	✓
Protection	✓	✗	✓
Performance	✓	✓	✗

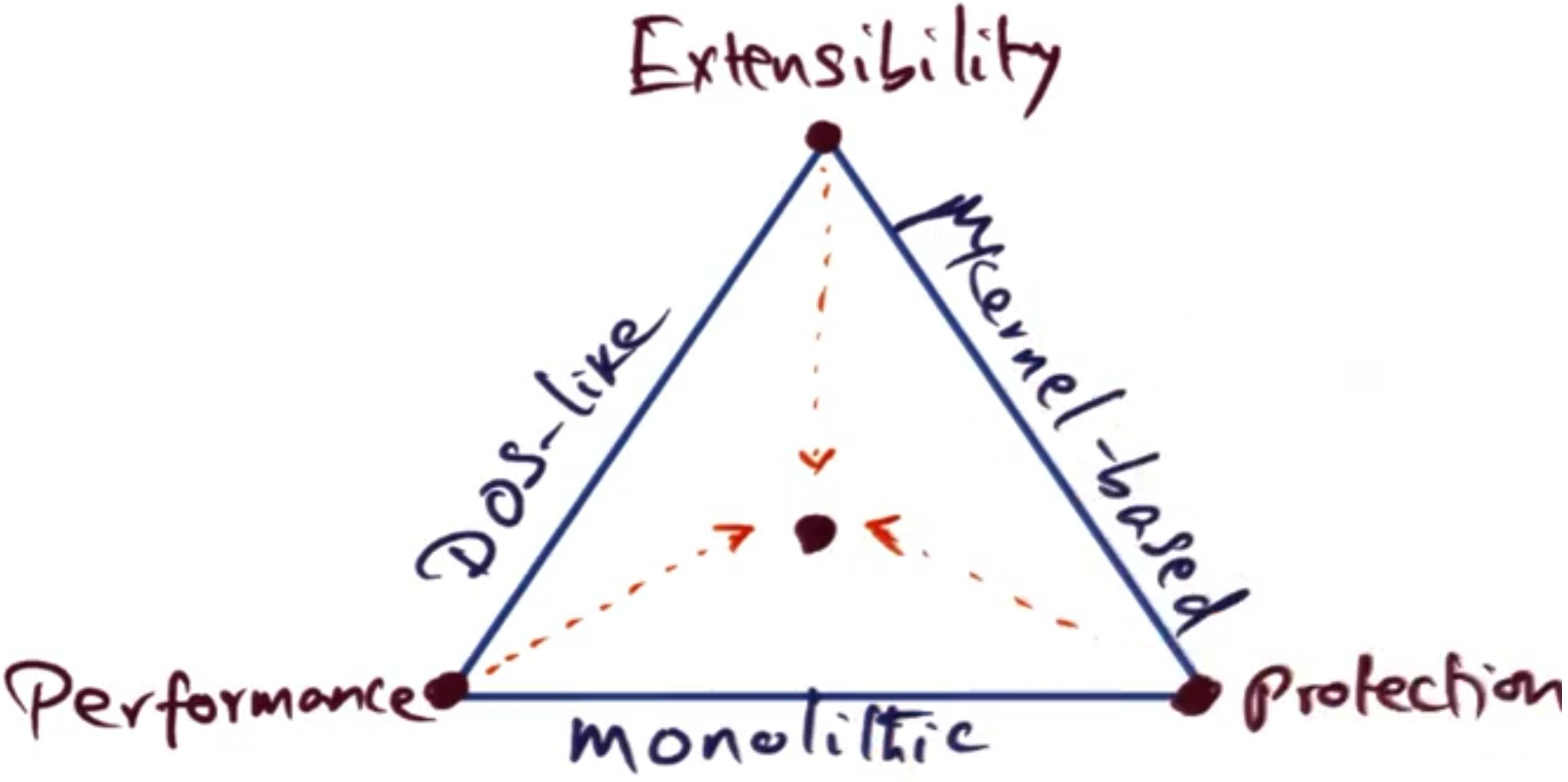
Comparison



Extensibility
DOS-like
μkernel-based
Performance
Monolithic
Protection

What do we want?

Can we have all three characteristics in an OS?



SPIN Approach of OS Structure

What we are Trying to Achieve?

- Thin like microkernel (only mechanisms not policies)
- Access to system resources without border crossing (like DOS)
- Extensibility for resource management (like microkernel) without sacrificing protection and performance (like monolithic)
- So in a nut shell, what we want from an operating system is performance, protection and extensibility

The SPIN operating system is a research project developed at University of Washington implemented in the computer programming language Modula-3. It is an open source project designed with three goals in mind: performance, protection and flexibility/extensibility

SPIN Structure

- The key idea in SPIN is to colocate a minimum kernel with its extensions in the same hardware address space and avoid the border crossing between the components of the kernel and the extensions of the kernel that are containing the specific services that the application needs. Since we are colocating the kernel and its extensions in the same hardware address space, this make the extensions as cheap as procedure calls (among each other), thus achieving **performance**
- The question is if we are colocating the kernel and its extensions in the same h/w address space, isn't it compromising **protection** (as was the case in DOS)
- To handle this SPIN took the approach of a strongly typed programming language (Modula3) to build the operating system, so that the compiler can enforce the modularity that we need to enforce protection. So SPIN do not depends on hardware address spaces to provide protection between different services and the kernel

SPIN Structure (cont...)

- Kernel provides well defined interfaces (declaring function prototypes in header files and having the actual implementation of the procedures in other files) and applications can dynamically bind to different implementation of functions at run time
- This dynamic call binding provides **flexibility/extensibility**, i.e., applications can dynamically bind to different implementations of the same interface function
- So in a nut shell, what we have accomplished with the spin approach, we are banking on the characteristics of a strongly typed programming language that enforces strong typing and therefore allows the operating system designer to implement logical protection domains

SPIN Mechanisms for Protection Domains

Write your code/service as Modula3 program with well defined entry points. Then use the three mechanisms available in SPIN to create protection domains and use them:

1. **Create:** The create mechanism is used to create a logical protection domain. It initiates an object file with the contents and export the names that are contained as entry point methods inside the object to be visible outside. For example, if I am creating a memory management service, I can write the entry point function in my memory management service and export the names using this create mechanism

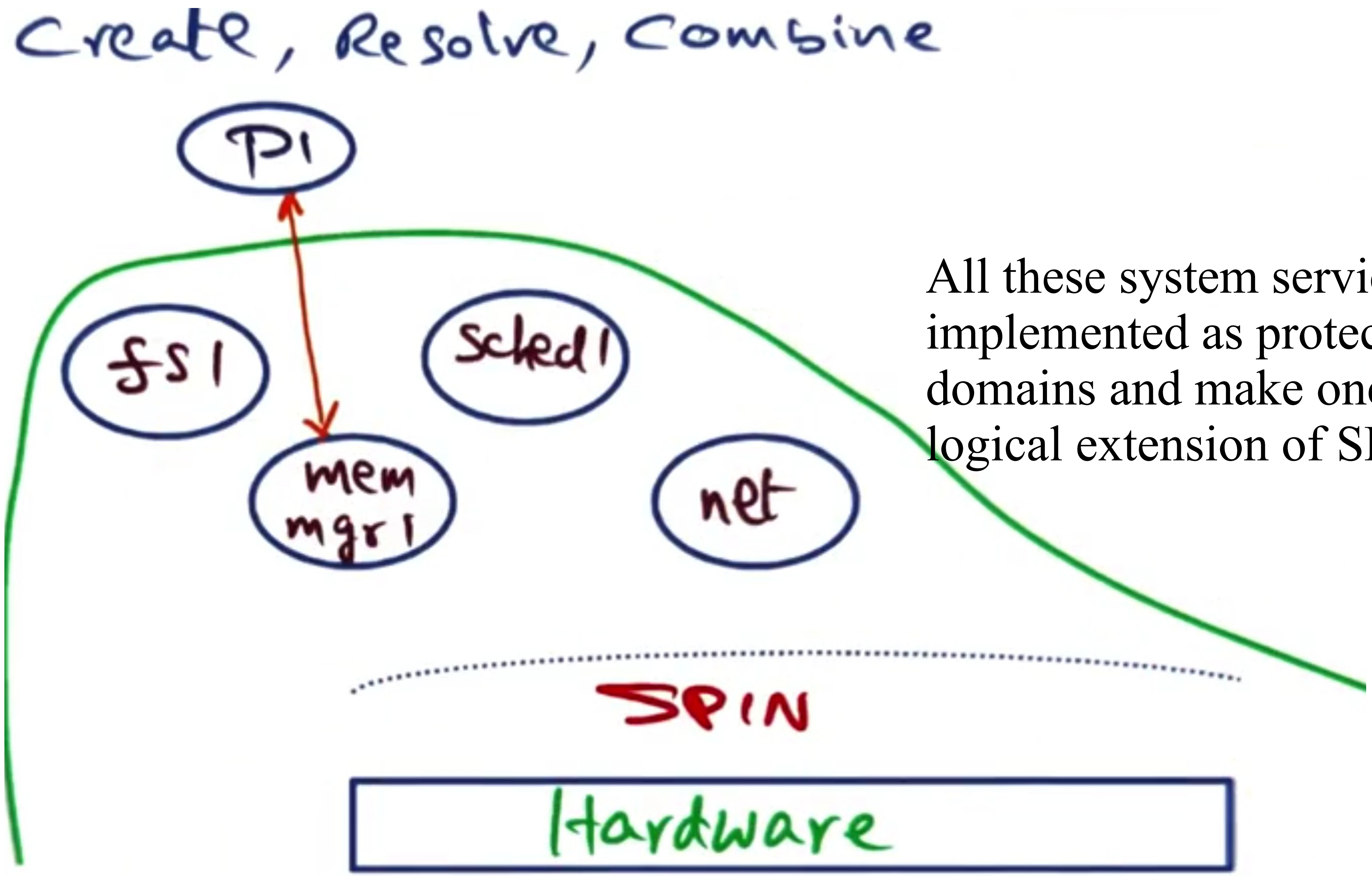
SPIN Mechanisms for Protection Domains (...)

2. **Resolve:** If one protection domain wants to use the names that are there in another protection domain, it can do so by using the resolve primitive. It is similar to linking two separately compiled files together and building a global symbol table from the symbol tables of the two. It resolves the names that are used in the source logical protection domain and the target logical protection domain. As a result of this resolve step, the source logical domain and the target logical domain are dynamically linked and bound together. So accessing methods of target logical domain happens as efficiently as making a simple procedure call

SPIN Mechanisms for Protection Domains (...)

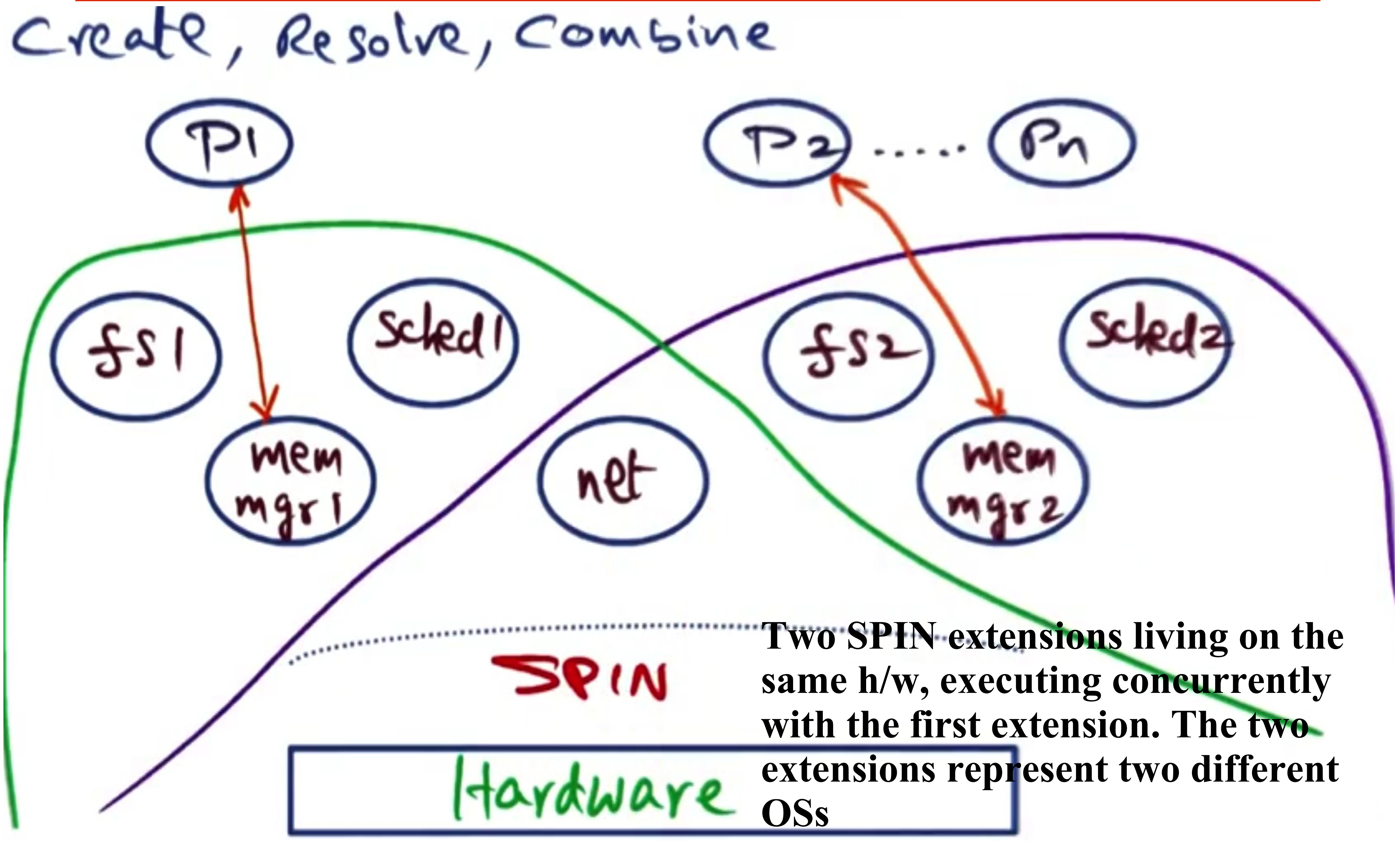
3. **Combine:** The combine primitive allows aggregation of two or more logical protection domains. The aggregate logical protection domain is the union of all the entry points that are available in the component logical protection domains. So this combine primitive in SPIN is mainly used to combat the proliferation of many small domains and have one large aggregate domain

Customized OS with SPIN



All these system services are implemented as protection domains and make one logical extension of SPIN

Customized OS with SPIN (cont...)



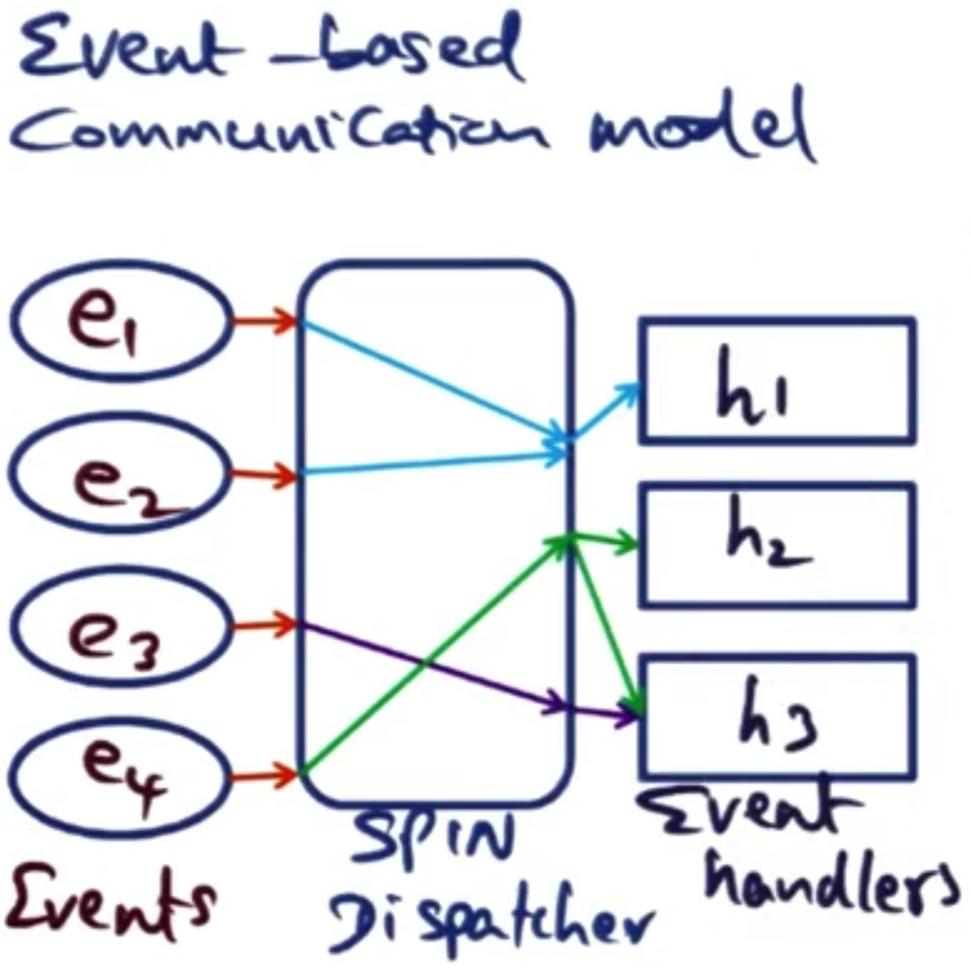
Two SPIN extensions living on the same h/w, executing concurrently with the first extension. The two extensions represent two different OSs

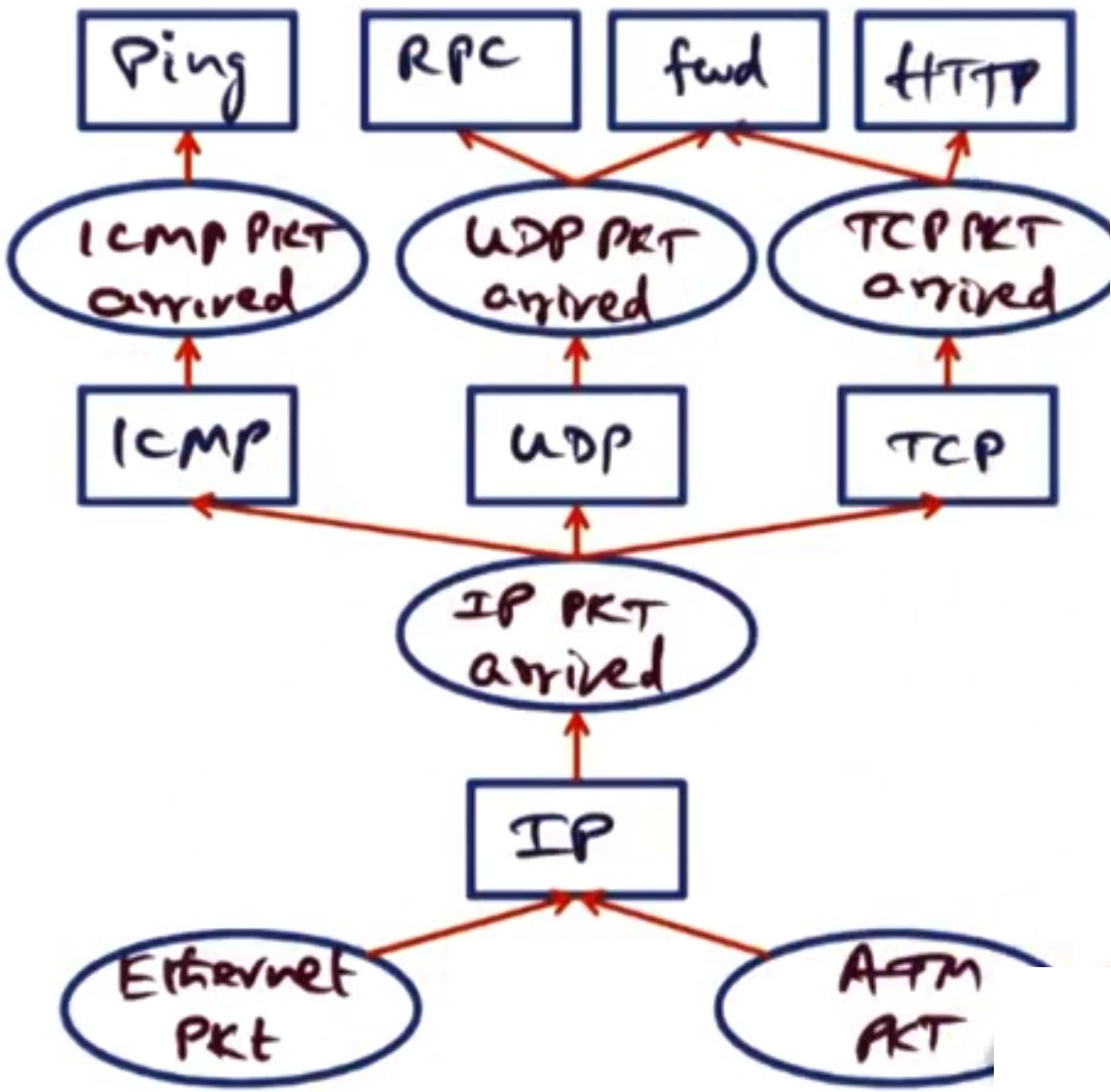
Question

Which of the following structures will result in least border crossings:

- Monolithic Structure
- Microkernel Structure
- SPIN
- Either SPIN or Monolithic

SPIN Mechanisms for Events





Writing Core Services in SPIN

- Now we know the tool box provided by SPIN for building an operating system
- One can build each of the services like memory management, CPU scheduling, threads, file system, network protocol stack and so on from scratch as extensions to SPIN
- An extensible operating system should dictate how these services should be implemented. SPIN provides interface procedures for implementing these services in the operating system

Default Core Services in SPIN

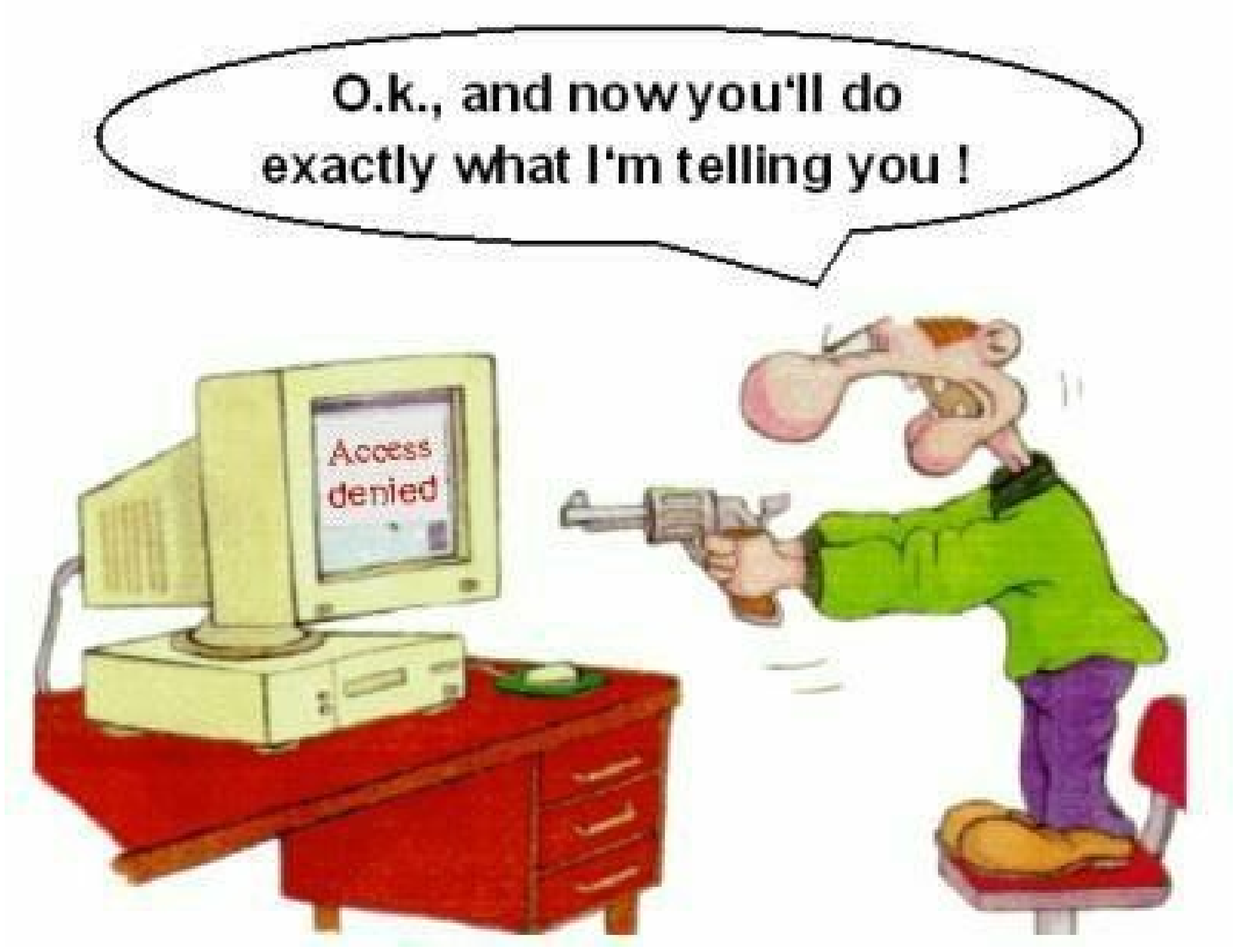
Memory Management:

- Physical Address
  - Allocate, deallocate, reclaim
- Virtual Address
  - Allocate, deallocate
- Translation
  - Create/destroy address space, add/remove mapping
- Event Handlers
  - Page fault, access fault, bad address

CPU Scheduling:

- SPIN abstraction: strand
  - Semantics defined by extension
- Event Handlers
  - Block, unblock, checkpoint, resume
- SPIN Global Scheduler
  - Interacts with application threads package

Things to do!



If you have problems visit me in counseling hours