COS3043
System Fundamentals

Lecture 2 (Part 1)

List of Discussion

Part 1

- OS Structure Overview
- The SPIN Approach

Part 2

- The Exokernel Approach
- The L3 Microkernel Approach

OS Structure Overview

What is Operating System Structure?

- In a nutshell, OS structure is the way the operating system software is organized with respect to the applications it serves and the underlying hardware it manages.

Question?

- Why OS structure is important?
  - Protection?
  - Performance?
  - Flexibility?
  - Scalability?
  - Agility?
  - Responsiveness?

Characteristics of Good OS Structure

Goal 1

Protection

Within and across users + OS itself.

Goal 2

Performance

Time taken to perform a service.

Goal 3

Extensibility

Flexibility => NOT one size fits all.

Goal 4

Scalability

Performance ↑, when resources ↑.

Goal 5

Agility

Adapting to application changes such as need of resources.

Goal 6

Responsiveness

Reacting to external events.

Question?

- Do you think an OS including the commercial OS such as Windows, Mac and Linux can meet all the goals at the same time?
  - Protection?
  - Performance?
  - Flexibility?
  - Scalability?
  - Agility?
  - Responsiveness?

Foundation of OS Structure

Type
Monolithic Structure
DOS-Like Structure
Microkernel-Based Structure

Monolithic Structure

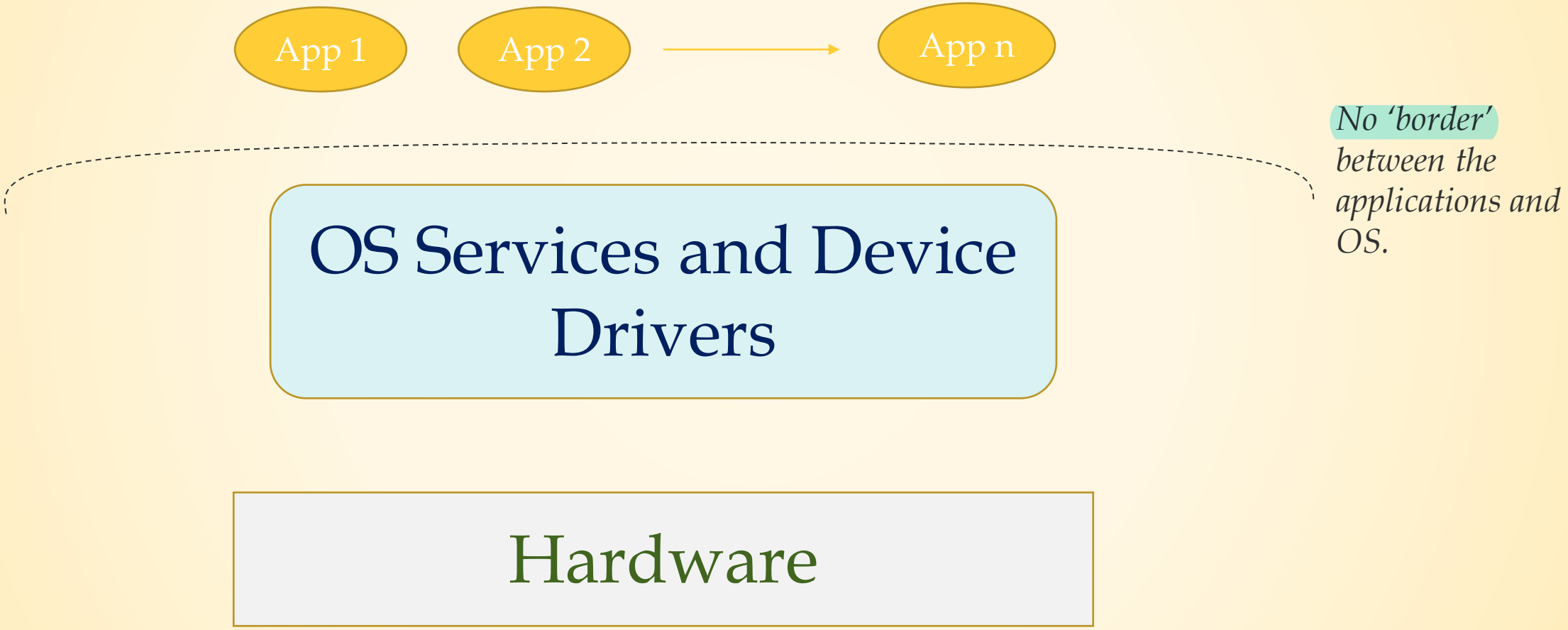
Each app is on its own hardware address space – it's protect from one another.

OS Services and Device Drivers

Hardware

- *OS services => files system, network access, scheduling to CPU, virtual memory management etc.*
- *The code and data structure of the OS are contained in its own hardware address space – it's protected from the applications.*

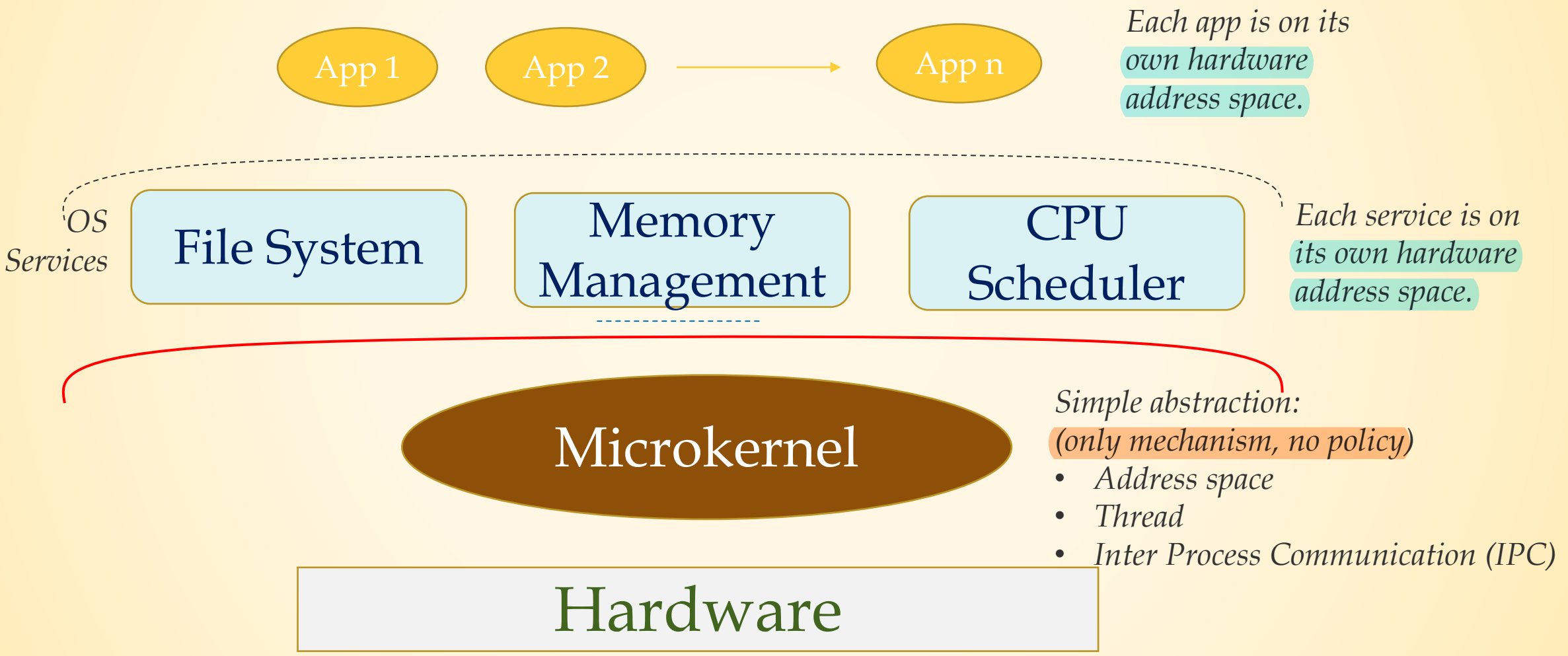
DOS-Like Structure



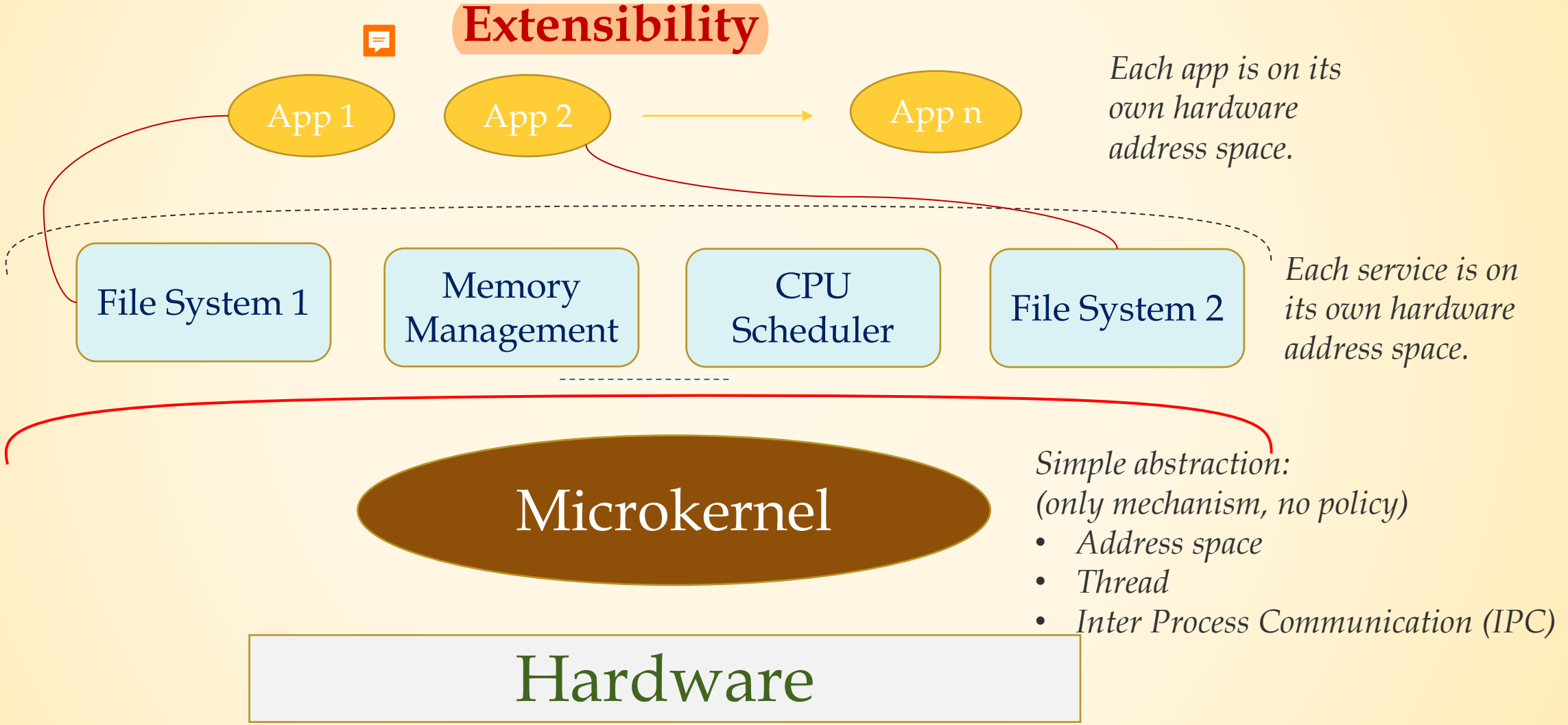
Comparison

- DOS-Like
  - Loss of protection, unacceptable for a general purpose OS.
- Monolithic
  - Loss of performance for protection?
  - Reduce performance loss by consolidation.
  - Consolidation – consolidate the components of OS in a single structure so that interaction among components can be expedited.
  - But because of the consolidation, there is no customization for different applications. But why customization?

Microkernel-Based Structure

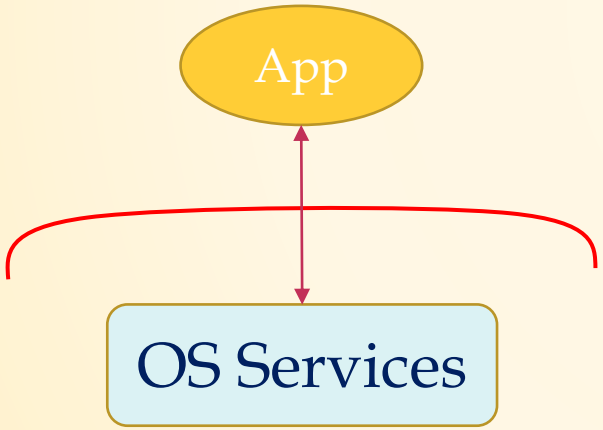


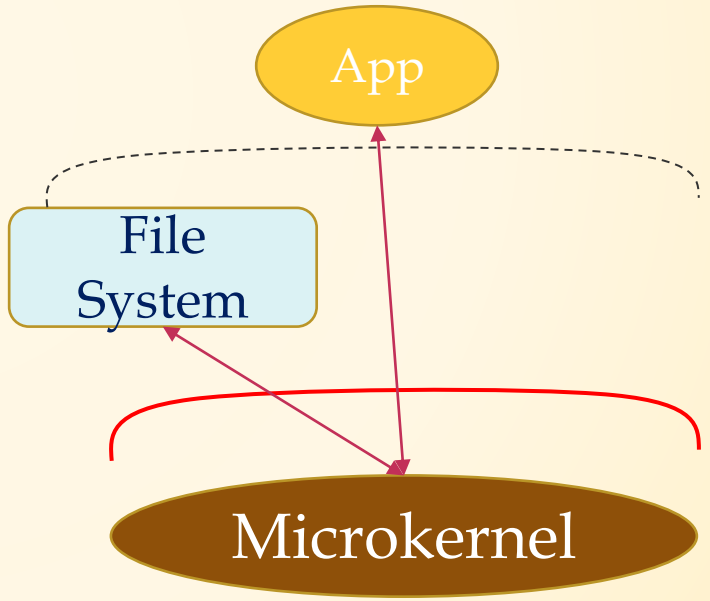
Advantage of Microkernel-Based Structure



Shortcoming of Microkernel-Based Structure

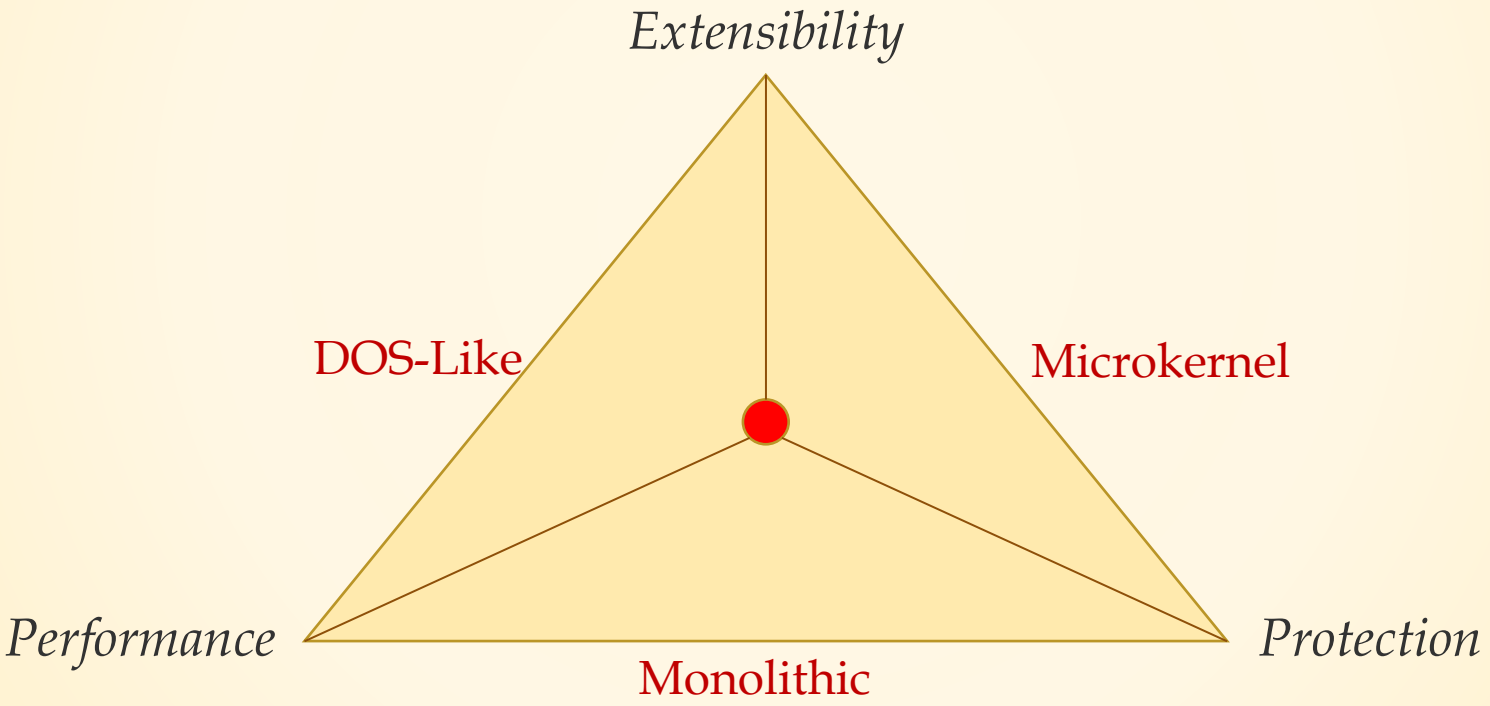
- Potential of performance loss.
  - Border crossing
    - Change in locality
    - User space <-> system space copying.





Summary

- So what do we want at the end? Let's check again what do we have from the three structures…



The SPIN Approach

Introduction

- Two important premises prior to create this approach:
  - Microkernel-based design compromising on performance due to frequent border crossing.
  - Monolithic design does not lend itself to extensibility.
- So, what do we really aim for a 'perfect' OS structure?
  - Thin (like microkernel), only mechanisms, no policies.
  - Access to resources without border-crossing (like DOS).
  - Flexibility for resources management (like microkernel), without scarifying protection and performance (like monolithic).

Examples of Approach Towards Extensibility

- [Hydra OS - 1981]
  - Kernel mechanisms for resource allocation.
  - Capability based security to access resource.
  - Resource managers as coarse-grained objects to reduce border-crossing.
  - However, because of the implementation of Capability based in managing resource, Hydra did not achieve the goal of extensibility.
- [Mach 1990s]
  - Focused on portability + extensibility
  - Huge loss in performance because of 'portability' – painted a bad image for microkernel.

SPIN Approach Towards Extensibility

- Co-location of kernel + extensions (cf kernel for OS services)
  - Avoid border crossing.
  - But if there are same hardware address space for kernel + extensions, then does it mean it has to give in the goal of protection?
- Compiler enforced modularity
  - Use strongly typed language concept for the kernel.
- Logical protection domains
  - Not relying on hardware address space.
- Dynamic call binding
  - Flexibility.

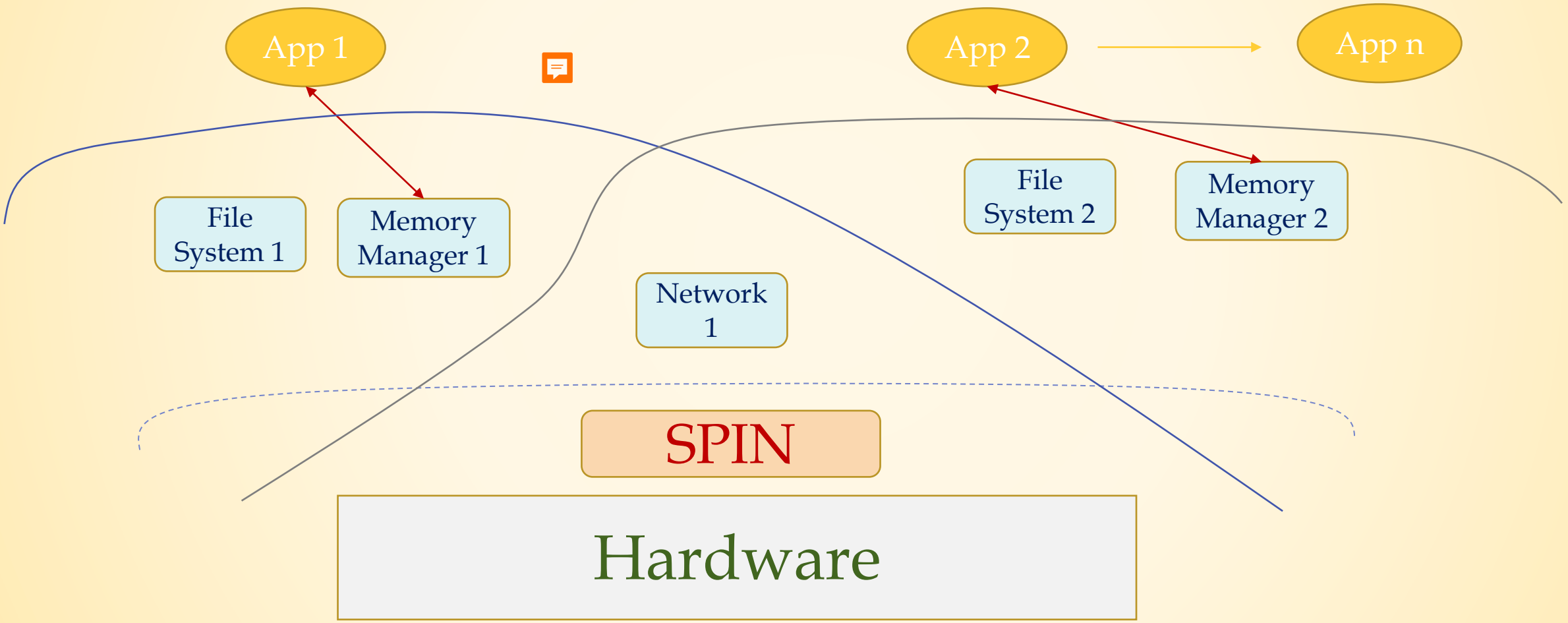
Logical Protection Domains

- Modula-3 => safety + encapsulation mechanisms
  - Type safety, auto storage management.
  - Objects, threads, exception, generic interfaces.
- Fine-grained object => protection via Capabilities
  - Hardware resources (ex: page frame).
  - Interfaces (ex: page allocation module)
  - Collection of interfaces (ex: the entire virtual memory sub system)
- Capabilities here differs from the one used for Hydra OS
  - As language supported pointers.

SPIN Mechanisms for Protection Domains

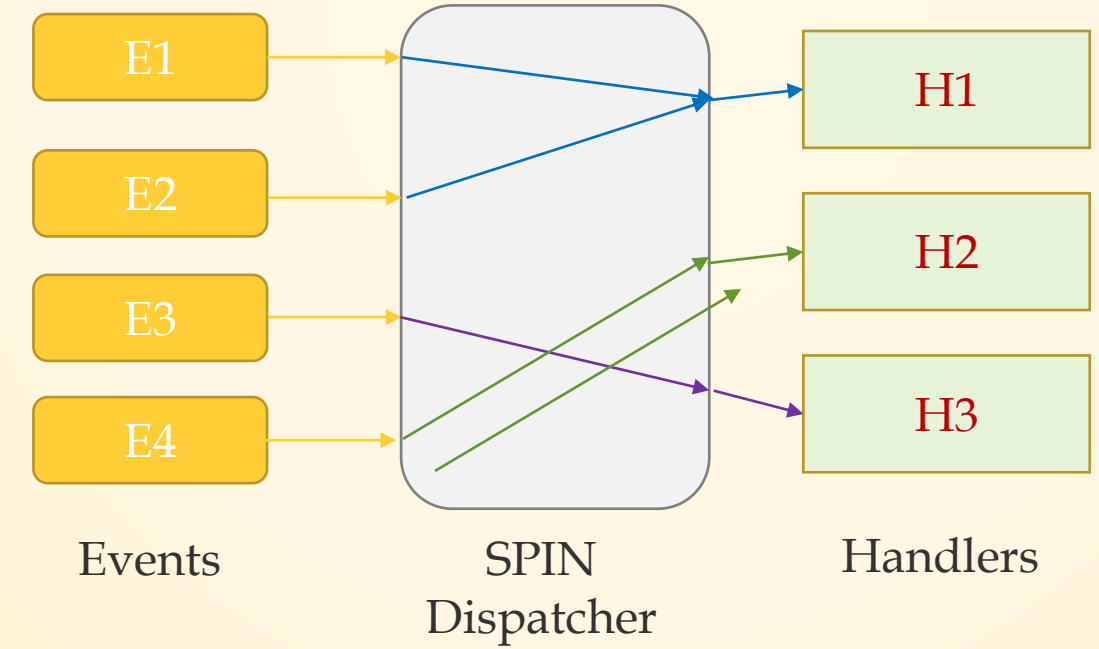
- Create
  - Initiate with object file contents + export names.
- Resolve
  - Names between source + target domains
  - Once resolved, resource sharing at memory speed.
- Combine
  - To create an aggregate domain.

SPIN Approach



SPIN Mechanisms for Events

- Every OS has to answer to external events (interruption, exception)
- SPIN uses Event-Based Communication model to deal with events.



Default Core Services in SPIN

- Thus far, we know that we can build OS services from scratch as an extension in SPIN.
- However, some of the core services such as memory management and CPU scheduling should not be dictated by the extension.
- Thus, SPIN provides interface procedures to implement those core services.