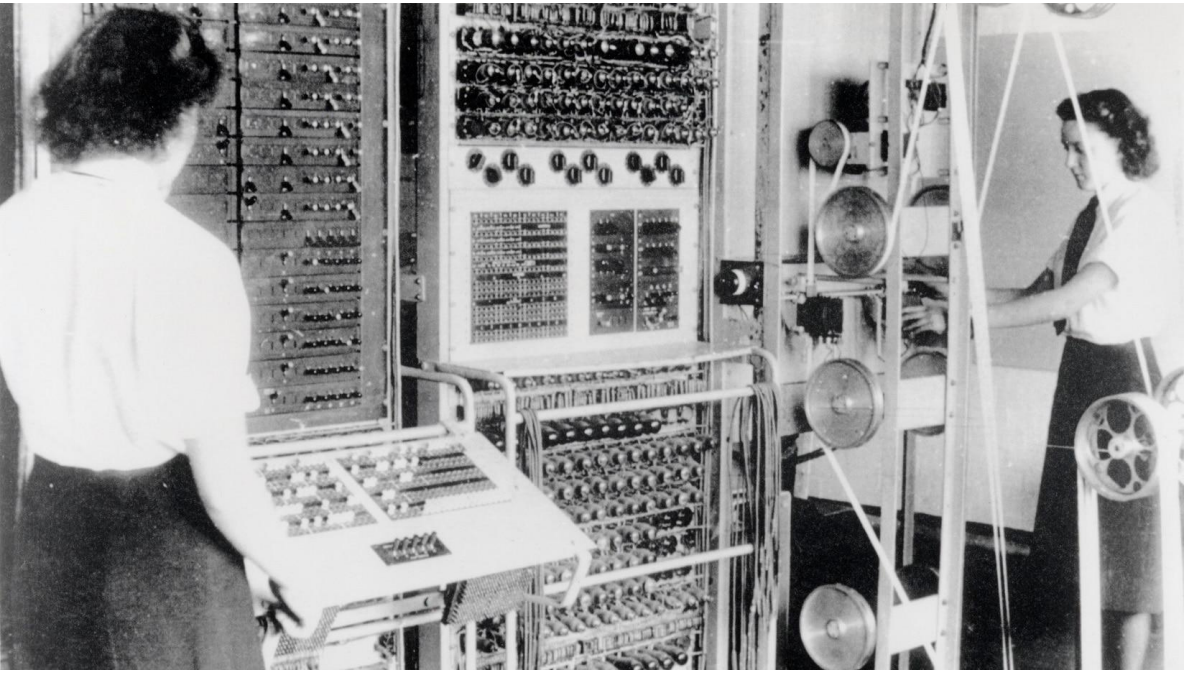
Operating System Architecture and Microkernels

Nges Brian

Outline

- Motivation
- Introduction to Operating Systems
- What is an Operating System?
- Functionalities of the Operating System
- Operating Systems Architectures
- Introduction to Microkernel
- An Example of a Microkernel - SeL4
- Conclusion

Motivation for Operating Systems



- Computers in the early 1940s and 1950s ran one program at a time.
- A programmer will write one program on punch cards.
- A dedicated operator fits the program into the computer.
- It then runs it and spits out some output.

Motivation

- The process usually takes long hours, days, and even weeks to run a program.
- Over time, computers became much faster,
  - Humans running around with programs took more time than running the programs themselves.
- Thus, there was a need for computers to operate themselves; hence, Operating systems were born.

Introduction to operating Systems

- Operating systems (OS) are programs with special privileges on the hardware that allow them to run and manage other programs.
- They are typically the first to start when a computer is turned on.
- The OS runs all subsequent programs.
- They started in the 1950s as computers became more powerful and widespread.
- The very first OSs augmented the manual task of loading programs by hand
  - Instead of loading a single program, programs were loaded in batches, called batch processing.

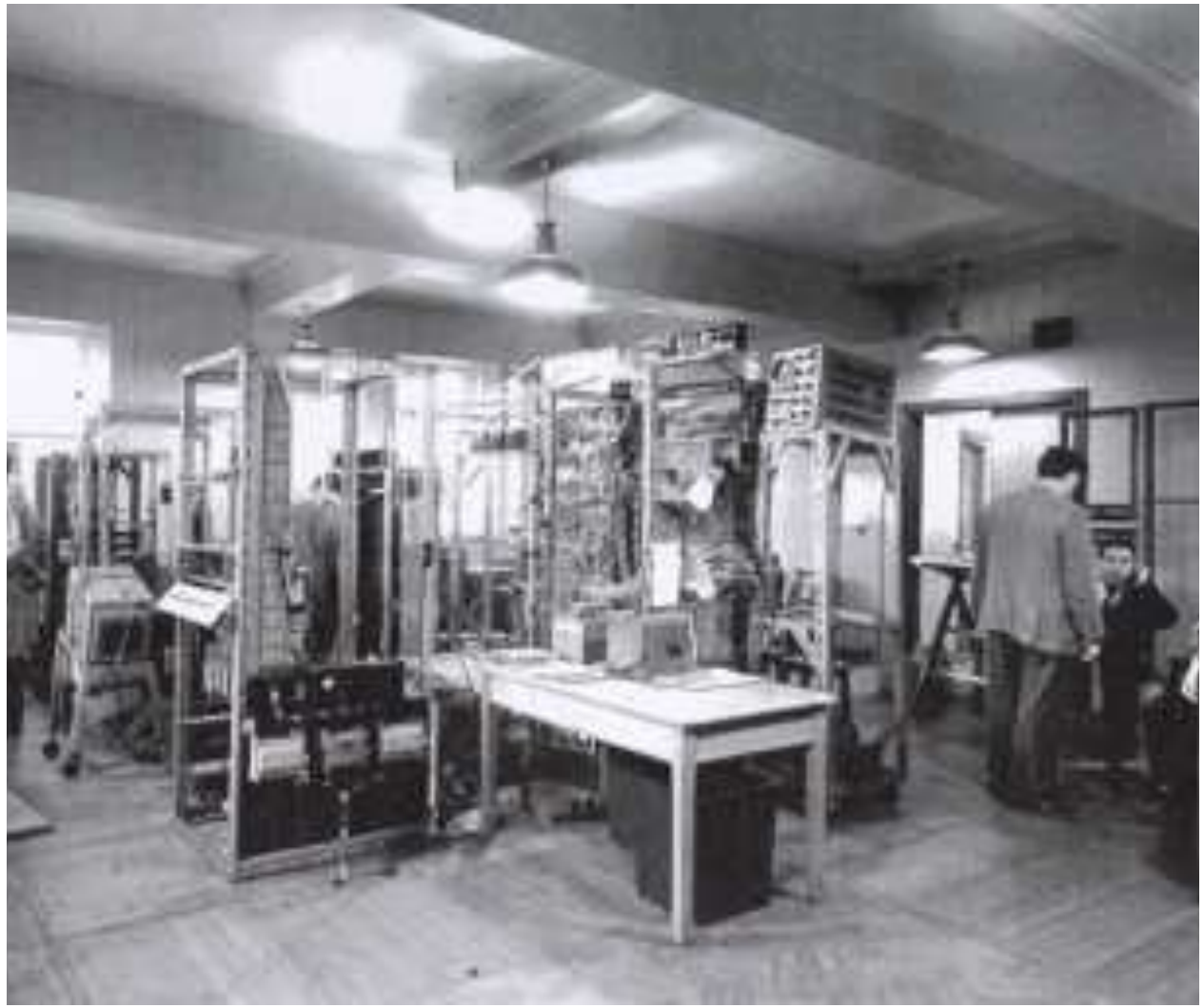
Introduction to Operating Systems

- Computers got faster and cheaper, thus becoming available in many universities and government offices.
- Soon, people started sharing software, leading to a new problem.
  - Managing computer configurations and peripherals.
- Computer configurations were not always identical.
  - Programmers now have to worry about different interfaces requiring them to know intimate hardware details about every device.
- Operating systems stepped in as intermediaries between software programs and hardware peripherals.
- They provided a software abstraction through APIs called device drivers, making the programmer's life much easier

Introduction to Operating Systems

- Using standardized mechanisms, OS allowed programmers to talk to input and output (I/O) hardware.
- By the end of the 1950s, processors had gotten so fast that programs were often blocked at the I/O for too long.
- The creation of the Atlas supercomputer by the University of Manchester between 1956 and 1962.
- They then developed the Atlas Supervisor.
  - The OS loaded programs automatically, like in the batch systems.
  - Run several programs simultaneously on its single CPU using scheduling algorithms.

Introduction to Operating Systems



- Scheduling was good, so they updated their computer with four paper tape readers, four paper tape punches, and eight magnetic tape drives.
- Introducing multitasking in computer systems sharing the same CPU.
- Introduced other issues like memory management since each program requires its own memory and can't lose its data when the CPU switches to another program

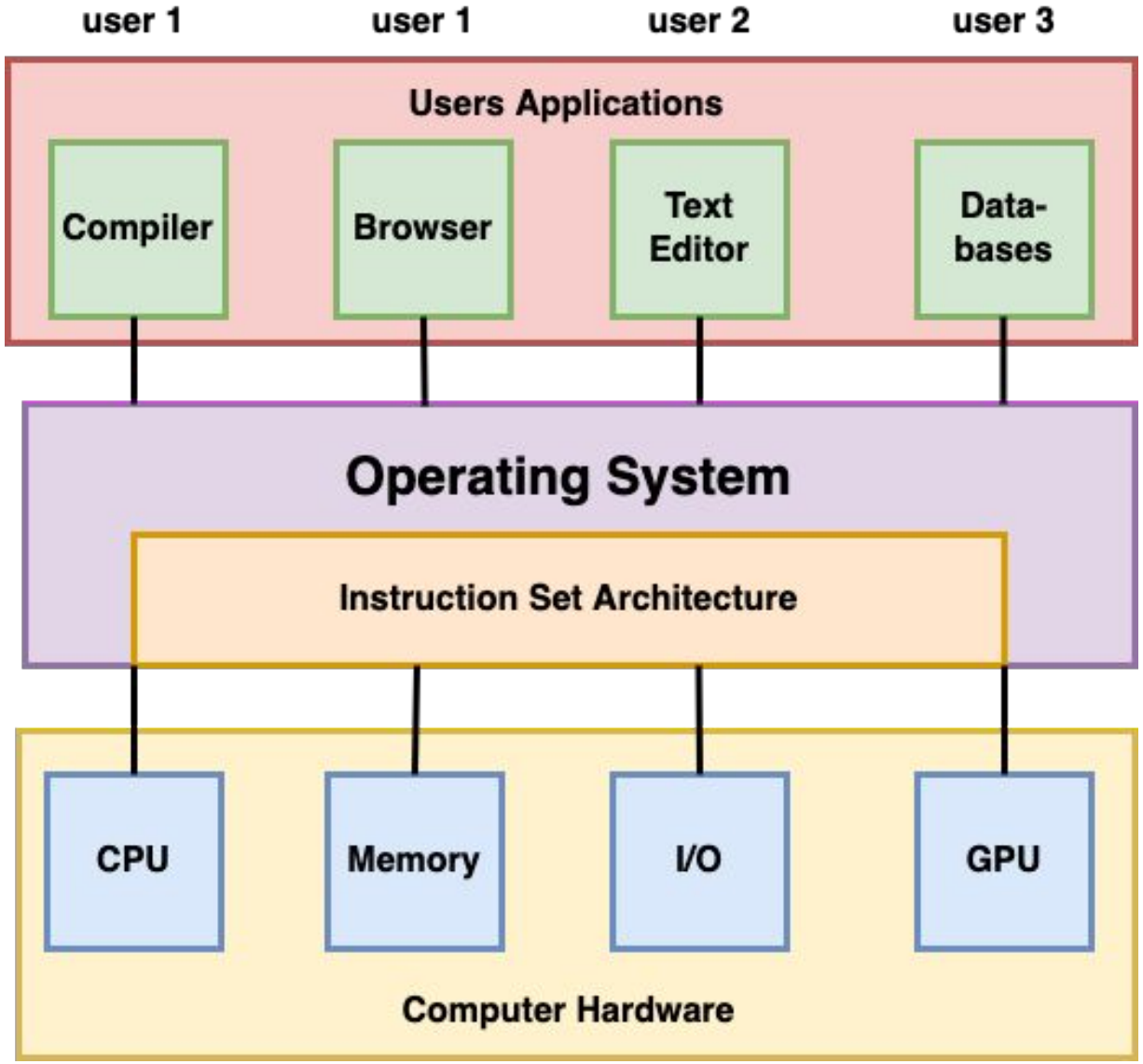
Introduction to Operating Systems

- By 1970, universities could buy computers and allow users to use them simultaneously through multiple terminals.
- New issues like time-sharing of resources, multiple users, etc.
  - Time-sharing Operating Systems were introduced like Multics

What is an Operating System

- It is the software that manages a computer's hardware resources and software applications.
- It is the layer between the applications and hardware and provides services to programs and users of the computer.
- It offers services like memory management, processes management, interface with peripherals, and ensure security.
- Popular examples include Linux, Windows, MacOS, Android, etc

Abstract view of the Computer System



Think of the OS as an orchestra conductor providing various computer components.

Outline

- Motivation
- Introduction to Operating Systems
- What is an Operating System?
- Functionalities of the Operating System
- Operating Systems Architectures
- Introduction to Microkernel
- An Example of a Microkernel - SeL4
- Conclusion

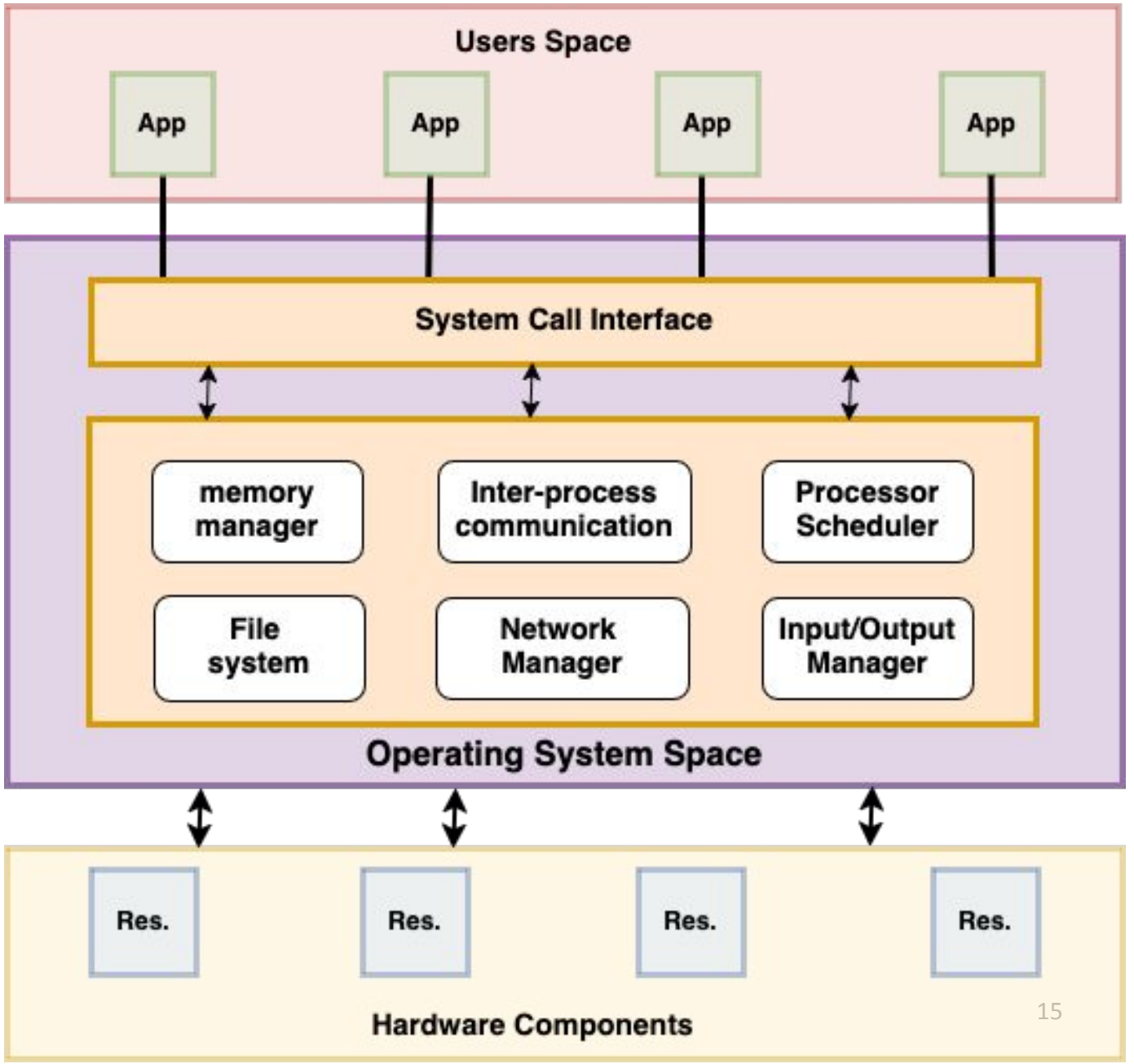
Functionalities of the Operating System

- Process Management:
  - Creates, schedules, and terminates processes
  - Manages process synchronization and communication.
  - Handles process states
- Memory Management
  - Allocate and deallocate memory to processes.
  - Implement virtual memory, paging, and segmentation
  - Provide security by preventing memory leaks, fragmentations
- File System Management
  - Organizes and manages files and directories
  - Controls access permissions and security.
  - It supports file operations such as reading, writing, and executing.

Functionalities of the operating system

- Device Management
  - Controls the hardware devices through drivers.
  - Manage input/output operations.
  - Ensures device coordination and error handling
- CPU Scheduling
  - Allocate CPU time to processes
  - Uses scheduling algorithms such as FCFS, Round Robin, etc
  - Handles multitasking and process prioritization.
- Security and Access Control
  - Manages authentication
  - Implement encryption and access control mechanisms
  - Prevent unauthorized access and malware attacks.
- User Interface (CLI, GUI, etc)
- Networking (communication between devices)

Operating System Components



Outline

- Motivation
- Introduction to Operating Systems
- What is an Operating System?
- Functionalities of the Operating System
- Operating Systems Architectures
- Introduction to Microkernel
- An Example of a Microkernel - SeL4
- Conclusion

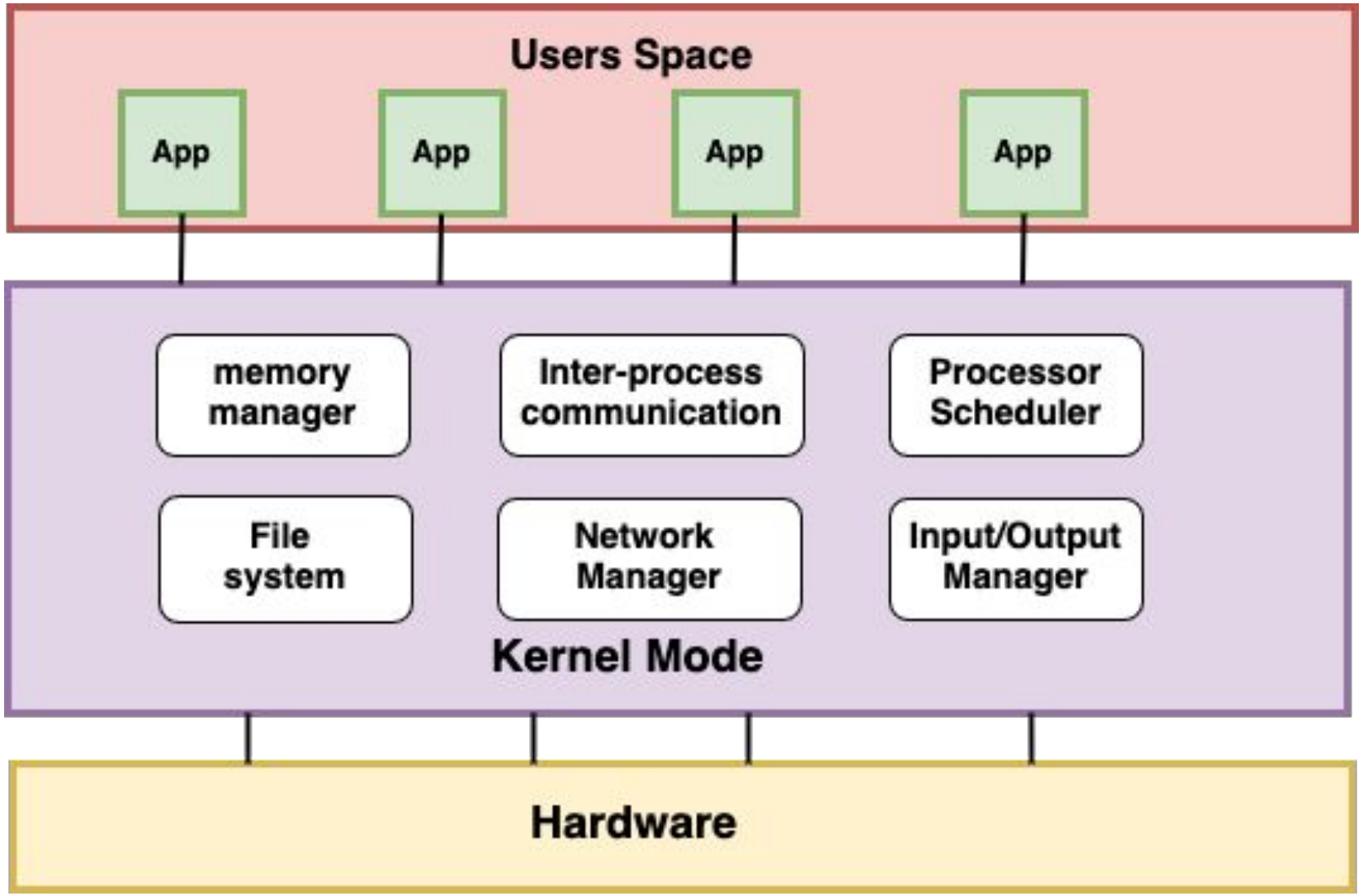
Operating Systems Architectures

- They define how operating system components interact to manage resources and provide services.
- They include:
  - Monolithic Architecture
  - Layered Architecture
  - Microkernel Architecture
  - Hybrid Kernel Architecture
  - Exokernel Architecture
  - Client-Server Architecture

Operating Systems Architectures

- Monolithic Architecture:
  - The Operating system runs as a single large kernel in kernel mode.
  - All of the OS services are integrated into one code base.
- Advantages:
  - Very fast execution since there is direct communication between all components.
  - Efficient system calls and process handing.
- Disadvantages:
  - Difficult to maintain and debug.
  - Large, complex, and less modular system.
  - Poor security
- Examples include Unix, Linux, and MS-DOS.

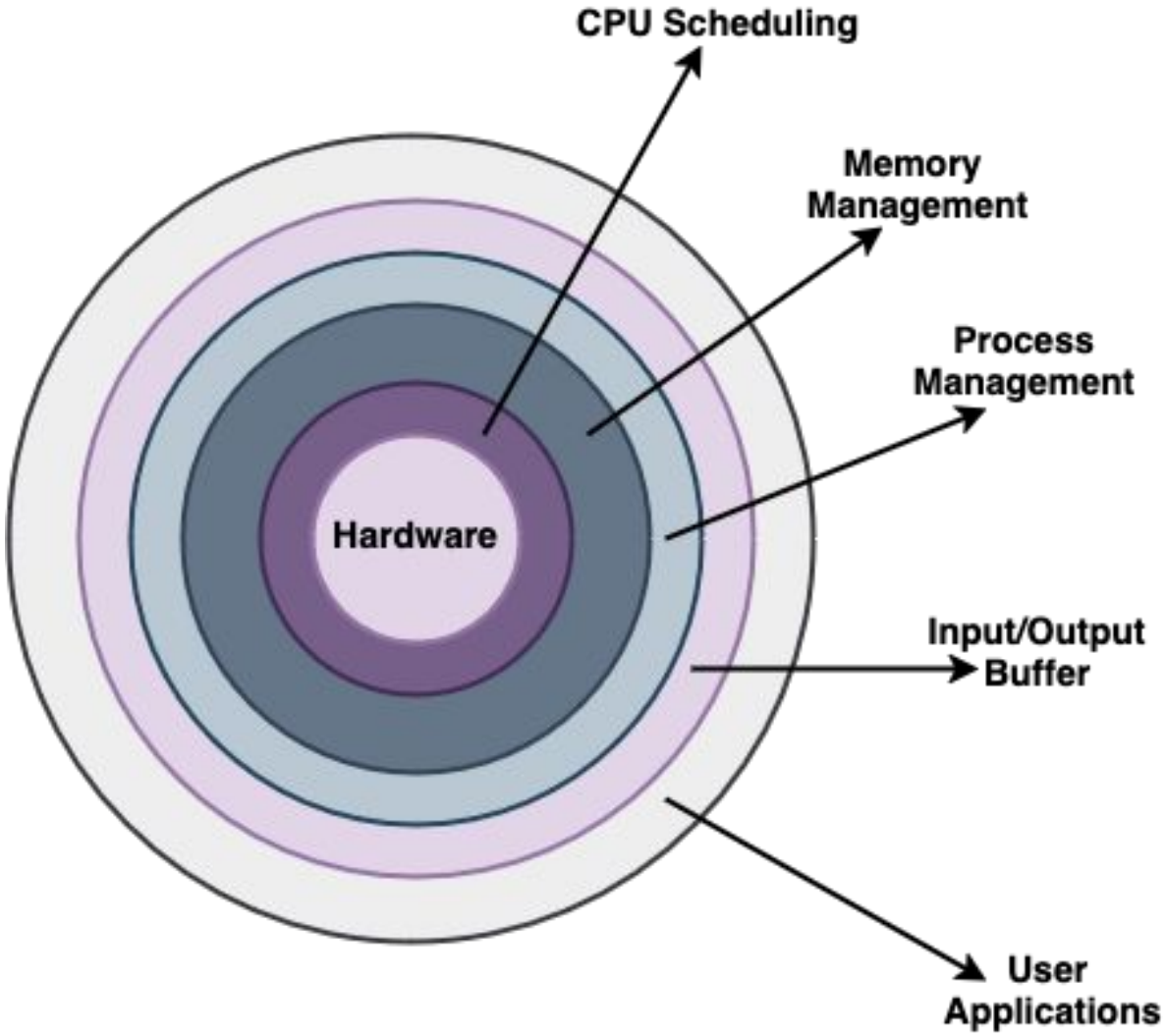
Monolithic OS Architecture



Operating Systems Architectures

- Layered Architectures:
  - The OS is divided into multiple layers, each handling specific functionalities.
  - Higher layers rely on lower layers but cannot interact directly.
- Advantages:
  - It has a modular design and is thus easy to debug and maintain.
  - Improved security and stability.
- Disadvantages:
  - Performance overhead.
  - It is difficult to define the proper layer.
- Examples include THE OS (Dijkstra), Multics

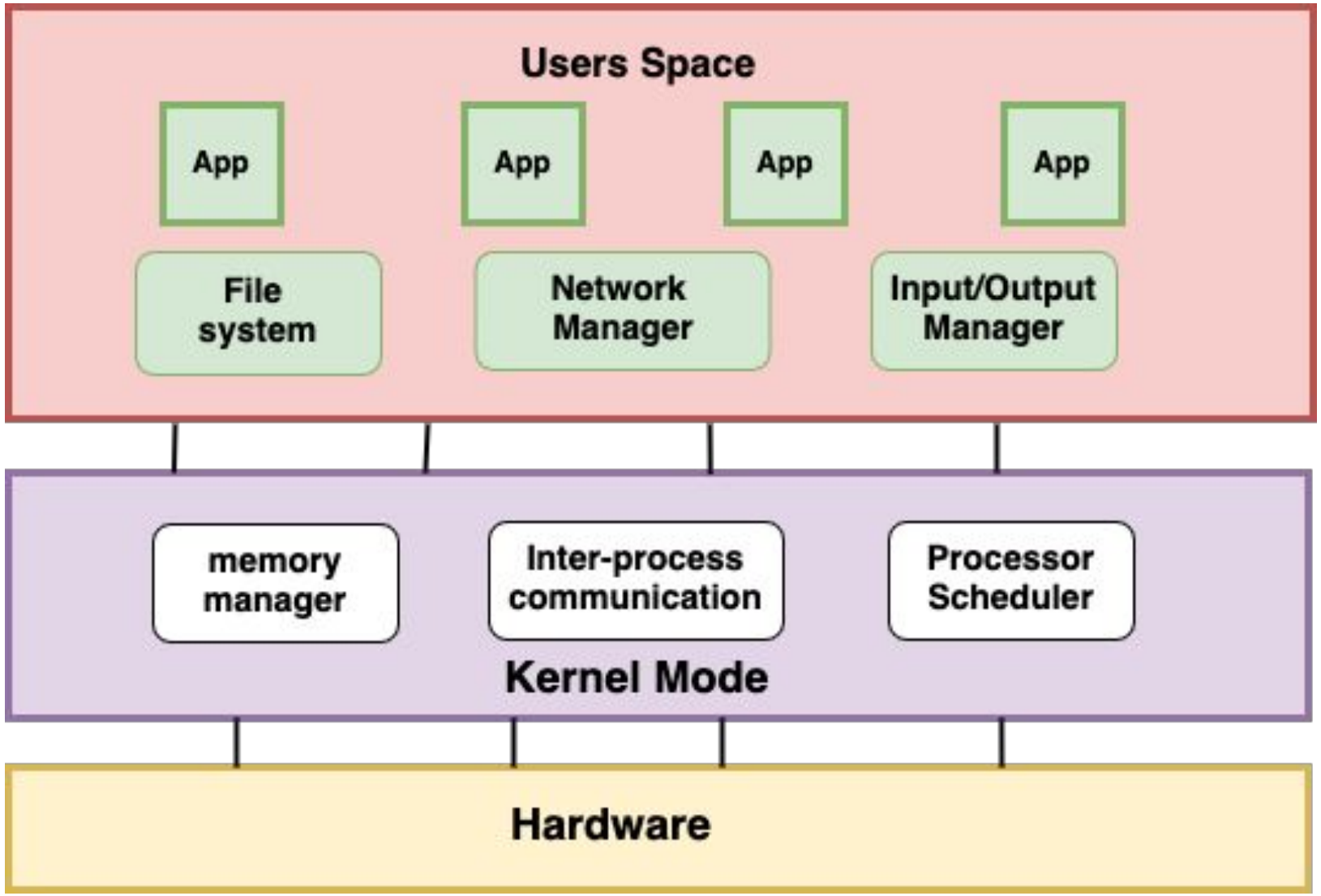
Layered OS Architecture



Operating Systems Architectures

- Microkernel Architectures
  - Only essential services run in the kernel space, such as process management, memory management, and inter-process communication (IPC).
- Advantages
  - Increased security and stability due to its modular structure.
  - Easier to extend and maintain.
  - Better fault isolation
- Disadvantages
  - Slower performance due to more context switching
  - Very complex IPC mechanism
  - More complex development
- Examples include QNX and L4 microkernels, Sel4

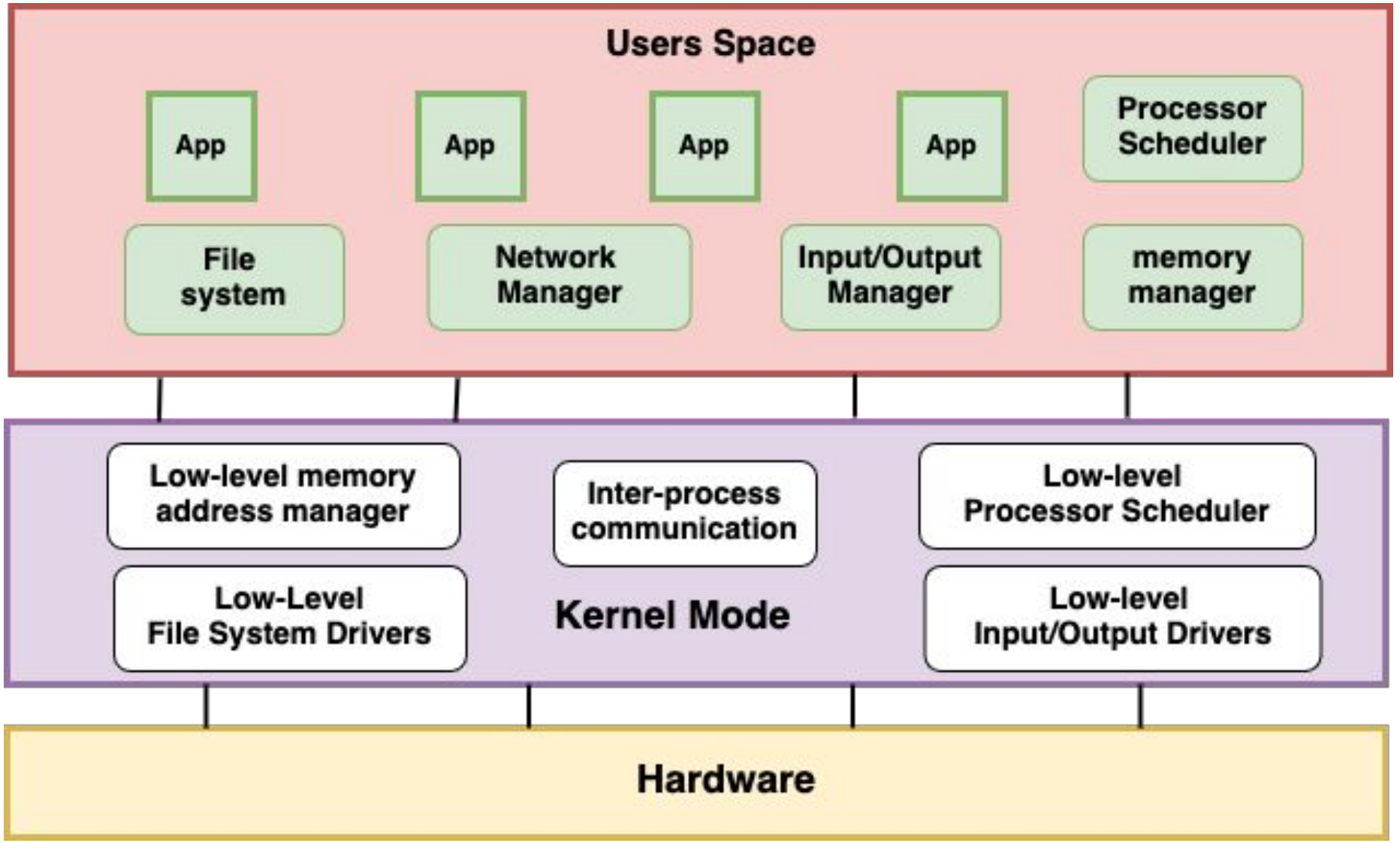
Microkernel OS Architecture



Operating Systems Architectures

- Hybrid Kernel Architectures
  - Combine the features of monolithic and microkernel architectures.
  - Most services run in kernel mode, but some run in user space.
- Advantages:
  - Balanced performance and modularity.
  - Better security than monolithic OS.
- Disadvantages
  - Still Complex to manage.
  - Not as fast as monolithic kernels.
- Examples include MacOS and Windows

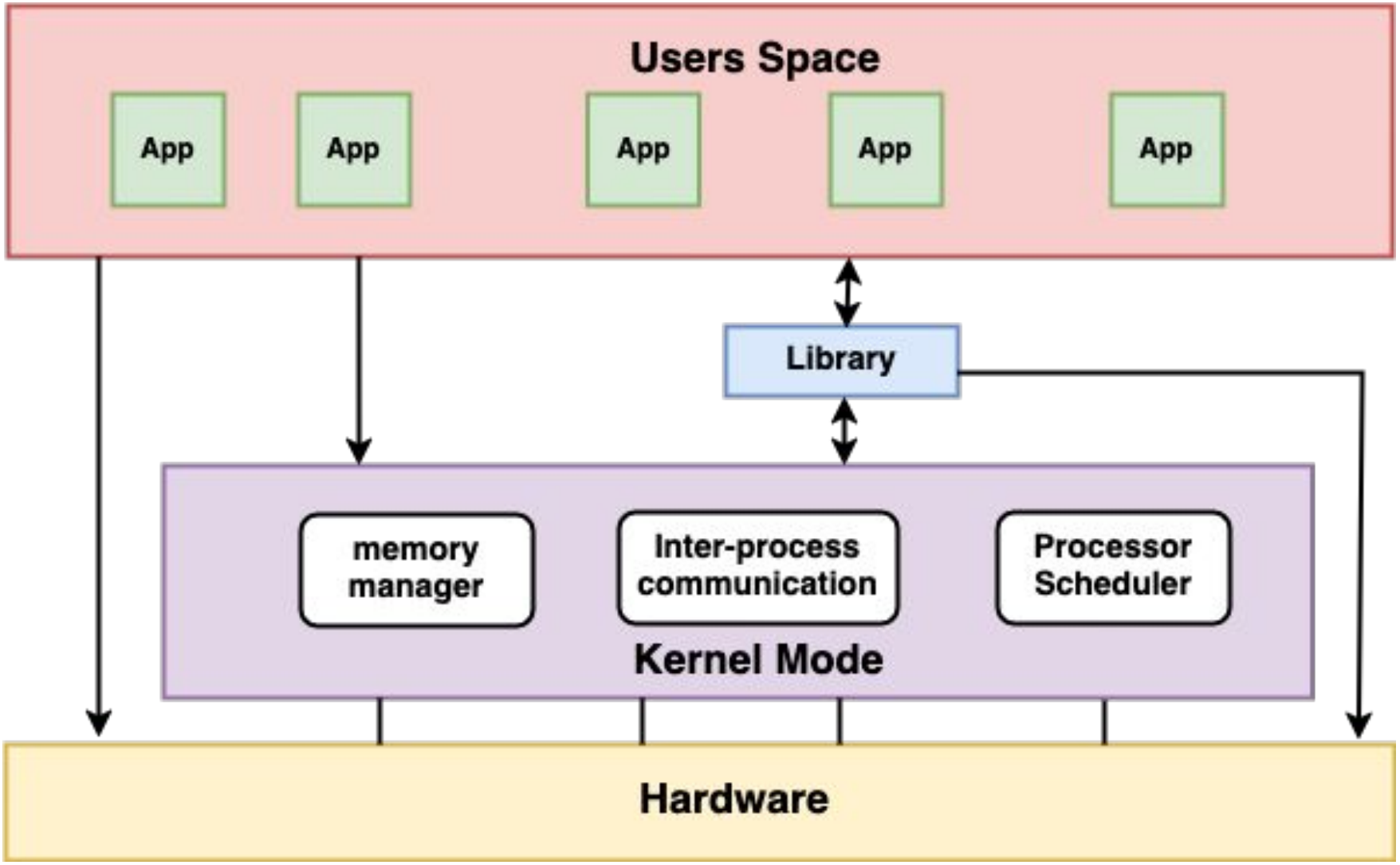
Hybrid OS Architecture



Operating Systems Architecture

- Exokernel Architecture
  - Minimalist OS kernel that provides bare-metal access to hardware.
  - Every application is allowed to manage its own resources directly.
- Advantages
  - Highly flexible and efficient
  - Reduces OS Overhead
- Disadvantages
  - Complex application development since every app handles low-level resource management.
  - Very high security risks.
- ExOs, Nemesis

Exokernel OS Architecture



Operating Systems Architectures

- Client-Server Architecture
  - The OS is divided into server processes running in user mode, while the kernel provides minimal services.
  - The system follows a request-response model.
- Advantages
  - Fault isolation, thus any crashes in user space, does not affect the kernel
  - Improved modularity and easier maintenance
- Disadvantages
  - High IPC overhead
  - Slower Performance compared to Monolithic OS
- Examples include Windows (partially), MacOS (partially)

Operating Systems Architectures

Architecture	Performance	Security	Maintainability	Examples
Monolithic	High	Low	Low	Linux, Unix
Layered	Medium	High	High	Multics
Microkernel	Medium	High	High	SeL4, QNX
Hybrid	High	Medium	Medium	Windows, MacOS
Exokernel	Very High	Low	Low	ExOs, Nemesis
Client-server	Medium	High	High	Windows, MacOS

Outline

- Motivation
- Introduction to Operating Systems
- What is an Operating System?
- Functionalities of the Operating System
- Operating Systems Architectures
- Introduction to Microkernel
- An Example of a Microkernel - SeL4
- Conclusion

Introduction to Microkernel

- It is an OS design where only the most essential services run inside the kernel.
- This separation improves security, modularity, and fault tolerance.
- The Key word here is **ROBUSTNESS**
- Key features:
  - Minimal kernel
    - Run only essential services such as process management, memory management, and IPC.
  - Inter-process communication (IPC)
    - All processes communicate through a message passing instead of direct function calls
  - Modular and Flexible
    - Additional features, such as new drivers, can be added without modifying the kernel.
  - Better fault Isolation
    - A crash in one service does not crash the entire OS

Microkernel IPC mechanisms

- IPC is the most common method in microkernels
- It is crucial for passing messages between processes, user-space services, and the kernel.
- The kernel only acts as a message router, avoiding direct function calls between processes.
- Mechanism:
  - Sender process – creates a message and sends via send() system call.
  - Kernel – handles the message and routes it to the receiver.
  - Receiver Process – reads the message using the receive() system call.
  - Feedback – The receiver sends results to the microkernel, which forwards to the sender process

Types of Message

Type	Message
Synchronous (blocking)	Sender waits until the receiver gets the message
Asynchronous (non-blocking)	The sender continues execution without waiting
Buffered (Queued)	Messages are stored in a queue until processed.
Direct or indirect	Messages can be sent directly or through kernel-managed message queues

System Call Handing in Microkernels

- System calls are handled as IPCs.
- System call Requests from the user process
  - A user invokes a system call, eg, read(), or write()
  - The request is sent to the microkernel through a software interrupt.
- Microkernel handles the Request
  - It identifies the appropriate user-space service, eg, file system service for read()
- IPC Message Passing
  - The microkernel forwards the request as an IPC message to the responsible service.
- Response sent back
  - After processing, the service sends the results back via IPC to the microkernel
  - The microkernel forwards the response to the user.

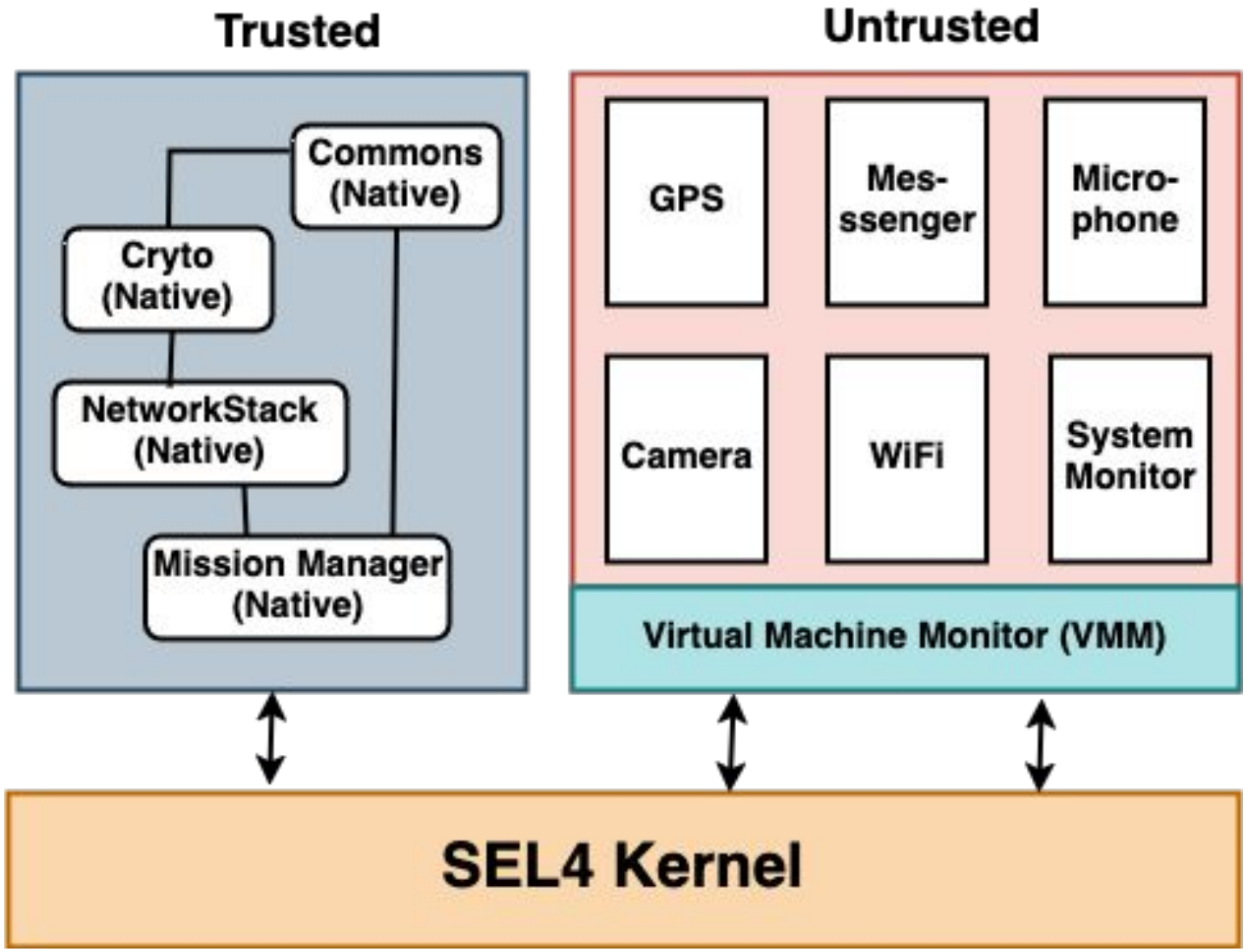
An Example of a Microkernel - SeL4

- SeL4 is a high-assurance, formally verified microkernel designed for security, safety, and performance.
- It is a part of the L4 family of microkernels.
- It has been a fully open-source project since 2014.
- It is used in autonomous vehicles (Toyota), to prevent cyber threats at DoD, aerospace, drones, medical devices, etc.
- Key features of SeL4:
  - The first microkernel to be formally verified
  - Enforces strict access control through capability-based isolation mechanisms.
  - High-optimized fast IPC with very low overhead.
  - It is a very small and minimalistic design with a codebase of ~10k lines of C code.

How SeL4 works

- The Kernel offers four main functions:
  - Process and thread management
  - IPC management
  - Memory management
  - Capability management.
- IPC management:
  - Uses fast synchronous IPC with message-passing between processes.
  - It supports shared memory for high-speed communication between trusted processes.
- Capability management
  - It uses capabilities (special access tokens) to control the permissions of resources.
  - A process can only access memory or a service with the correct capability.

Use-case of SeL4



Conclusion

Questions??