TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute for System Architecture, Operating Systems Group

Microkernel-based Operating Systems —Introduction—

Jan Bierbaum
Carsten Weinhold

Dresden, October 14th 2025

Lecture Goals

Lecture Goals

- Provide deeper understanding of OS mechanisms

Lecture Goals

- Provide deeper understanding of OS mechanisms
- Illustrate alternative design concepts

Lecture Goals

- Provide deeper understanding of OS mechanisms
- Illustrate alternative design concepts
- Promote OS research at TU Dresden

Lecture Goals

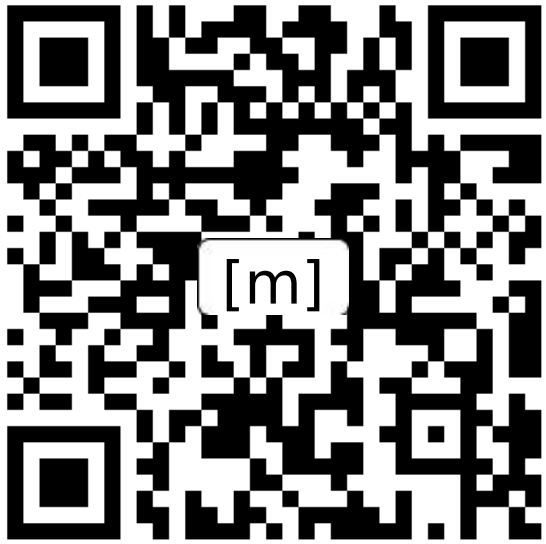
- Provide deeper understanding of OS mechanisms
- Illustrate alternative design concepts
- Promote OS research at TU Dresden
- Make you all enthusiastic about OS development in general and microkernels in particular

Administrativia

- Slides: https://tud.de/inf/os → ☰ → Studies → Lectures → MOS

- Slides: https://tud.de/inf/os → ☰ → Studies → Lectures → MOS
- Mailing list: https://os.inf.tu-dresden.de/mailman/listinfo/mos2025

- Slides: https://tud.de/inf/os → ☰ → Studies → Lectures → MOS
- Mailing list: https://os.inf.tu-dresden.de/mailman/listinfo/mos2025
- Matrix: https://matrix.to/#/#inf-os-mos:tu-dresden.de



Structure

- Lecture:

Structure

- Lecture:
  - 2 SWS
  - Tuesday, 14:50, APB E008
- Exercises:

Structure

- Lecture:
  - 2 SWS
  - Tuesday, 14:50, APB E008
- Exercises:
  - 1 SWS
  - ~~*odd* weeks,~~ Tuesday, 16:40, APB E009 / APB E065
- Lab (except INF-DSE-20-E-MKS):

Structure

- Lecture:
  - 2 SWS
  - Tuesday, 14:50, APB E008
- Exercises:
  - 1 SWS
  - ~~*odd* weeks,~~ Tuesday, 16:40, APB E009 / APB E065
- Lab (except INF-DSE-20-E-MKS):
  - 1 SWS + 2 SWS work on your own
  - ~~*even* weeks,~~ Tuesday, 16:40, APB E009

Structure

- Lecture:
  - 2 SWS
  - Tuesday, 14:50, APB E008
- Exercises:
  - 1 SWS
  - ~~*odd* weeks,~~ Tuesday, 16:40, APB E009 / APB E065
- Lab (except INF-DSE-20-E-MKS):
  - 1 SWS + 2 SWS work on your own
  - ~~*even* weeks,~~ Tuesday, 16:40, APB E009
- (Mostly) hybrid via BBB
- All content relevant for exams

Exercises

- Exercises *may* take place online/hybrid
- Announced on website and mailing list

Exercises

- Exercises *may* take place online/hybrid
- Announced on website and mailing list
- Practical exercises

Exercises

- Exercises *may* take place online/hybrid
- Announced on website and mailing list
- Practical exercises
  - Implement aspect of a MOS in the computer pool
  - First one on *Nov 4th*
- Paper reading exercises

Exercises

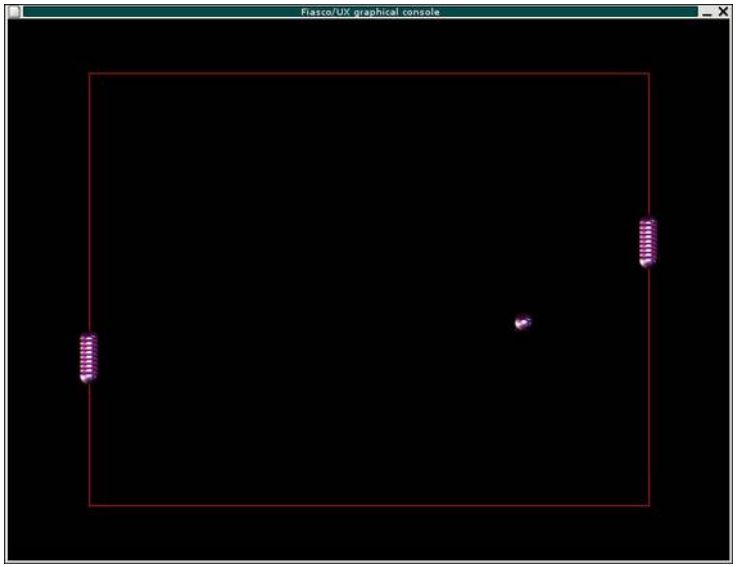
- Exercises *may* take place online/hybrid
- Announced on website and mailing list
- Practical exercises
  - Implement aspect of a MOS in the computer pool
  - First one on *Nov 4th*
- Paper reading exercises
  - Read a paper **beforehand**
  - Prepare questions/observations
  - Actively participate in the discussion
  - First one *next week (Oct 21st)*

Lab

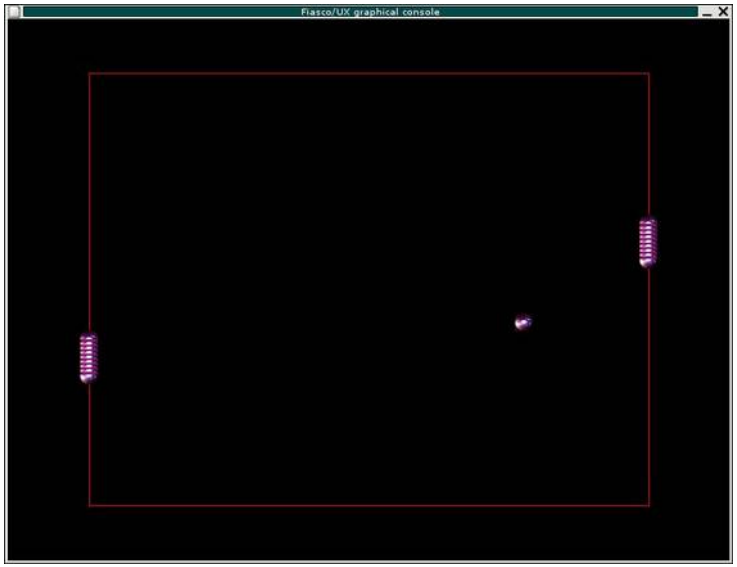
- Formerly known as complex lab “Microkernel-based Operating Systems”

Lab

- Formerly known as complex lab “Microkernel-based Operating Systems”
- Build several components of an MOS
- Mainly programming on your own *(optional)*



- Formerly known as complex lab “Microkernel-based Operating Systems”
- Build several components of an MOS
- Mainly programming on your own *(optional)*
- Coordination via dedicated mailing list → website
- First meeting *today* after the lecture



Monoliths vs. Microkernels

Purpose of an Operating System

- Manage the available resources
  - Hardware (CPU, memory, storage, network, …)
  - Software (file systems, networking stack, …)

Purpose of an Operating System

- Manage the available resources
  - Hardware (CPU, memory, storage, network, …)
  - Software (file systems, networking stack, …)
- Provide easier-to-use interface to access resources
  - Unix: read/write data from/to sockets instead of fiddling with TCP/IP packets on your own

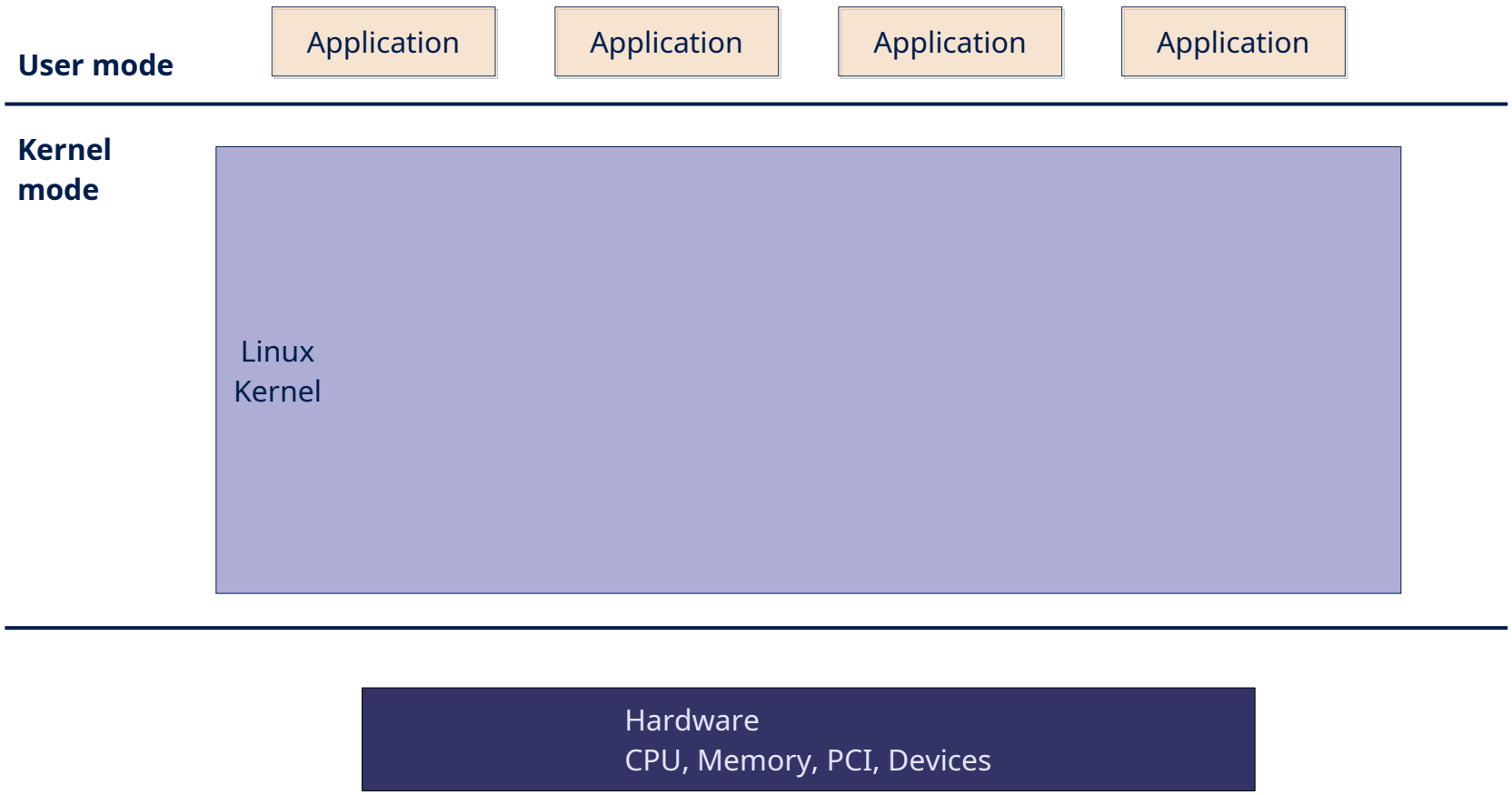
Purpose of an Operating System

- Manage the available resources
  - Hardware (CPU, memory, storage, network, …)
  - Software (file systems, networking stack, …)
- Provide easier-to-use interface to access resources
  - Unix: read/write data from/to sockets instead of fiddling with TCP/IP packets on your own
- Perform privileged or HW-specific operations
  - x86: ring 0 vs. ring 3
  - Device drivers

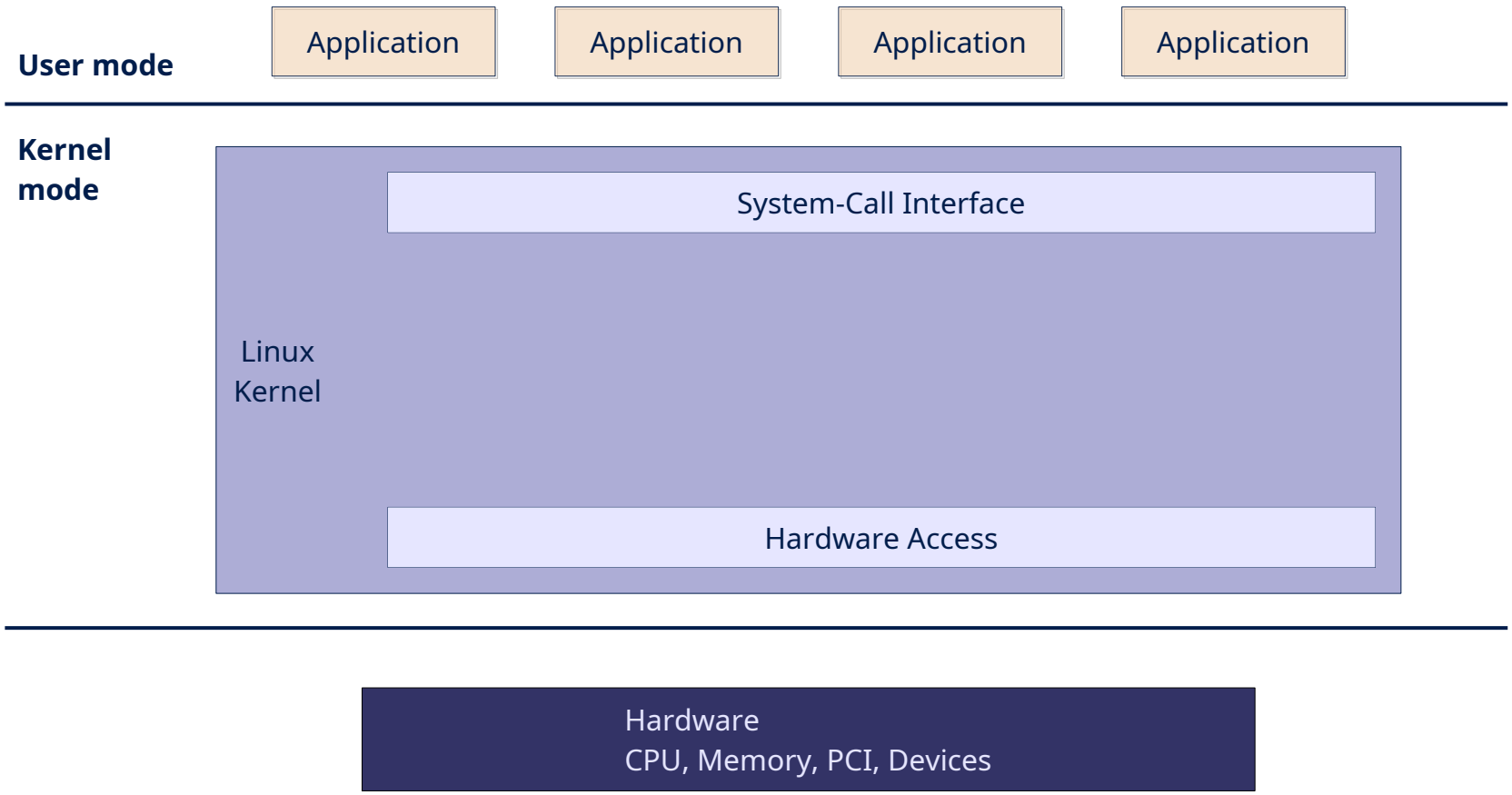
Purpose of an Operating System

- Manage the available resources
  - Hardware (CPU, memory, storage, network, …)
  - Software (file systems, networking stack, …)
- Provide easier-to-use interface to access resources
  - Unix: read/write data from/to sockets instead of fiddling with TCP/IP packets on your own
- Perform privileged or HW-specific operations
  - x86: ring 0 vs. ring 3
  - Device drivers
- Provide separation and means for collaboration
  - Isolate users / processes from each other
  - Allow cooperation if needed (e.g., sending messages between processes)

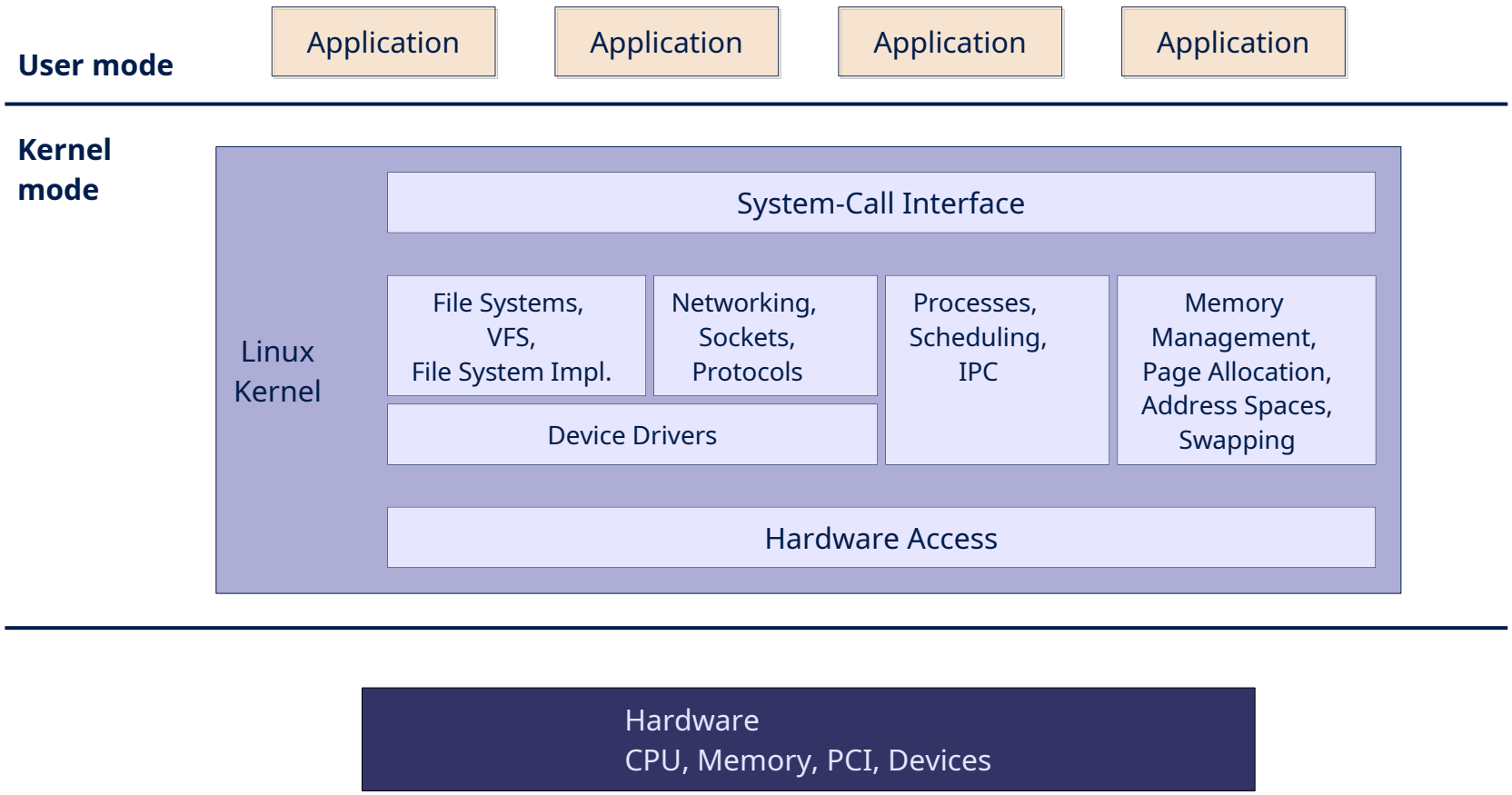
Monolithic Kernels: Linux



Monolithic Kernels: Linux



Monolithic Kernels: Linux



What's the Problem with Monoliths?

- Security issues

What's the Problem with Monoliths?

- Security issues
  - All components in privileged mode
  - Direct access to all kernel-level data
  - Module loading → easy living for rootkits
- Resilience issues

What's the Problem with Monoliths?

- Security issues
  - All components in privileged mode
  - Direct access to all kernel-level data
  - Module loading → easy living for rootkits
- Resilience issues
  - Faulty drivers can crash the whole system
  - 75% of today's OS kernels are drivers
- Software-level issues

What's the Problem with Monoliths?

- Security issues
  - All components in privileged mode
  - Direct access to all kernel-level data
  - Module loading → easy living for rootkits
- Resilience issues
  - Faulty drivers can crash the whole system
  - 75% of today's OS kernels are drivers
- Software-level issues
  - Complexity is hard to manage
  - Custom OS for hardware with scarce resources?

The Microkernel Vision

- Minimal OS kernel
  - Less error prone (less code → fewer errors)
  - Small *Trusted Computing Base*
  - Suitable for verification

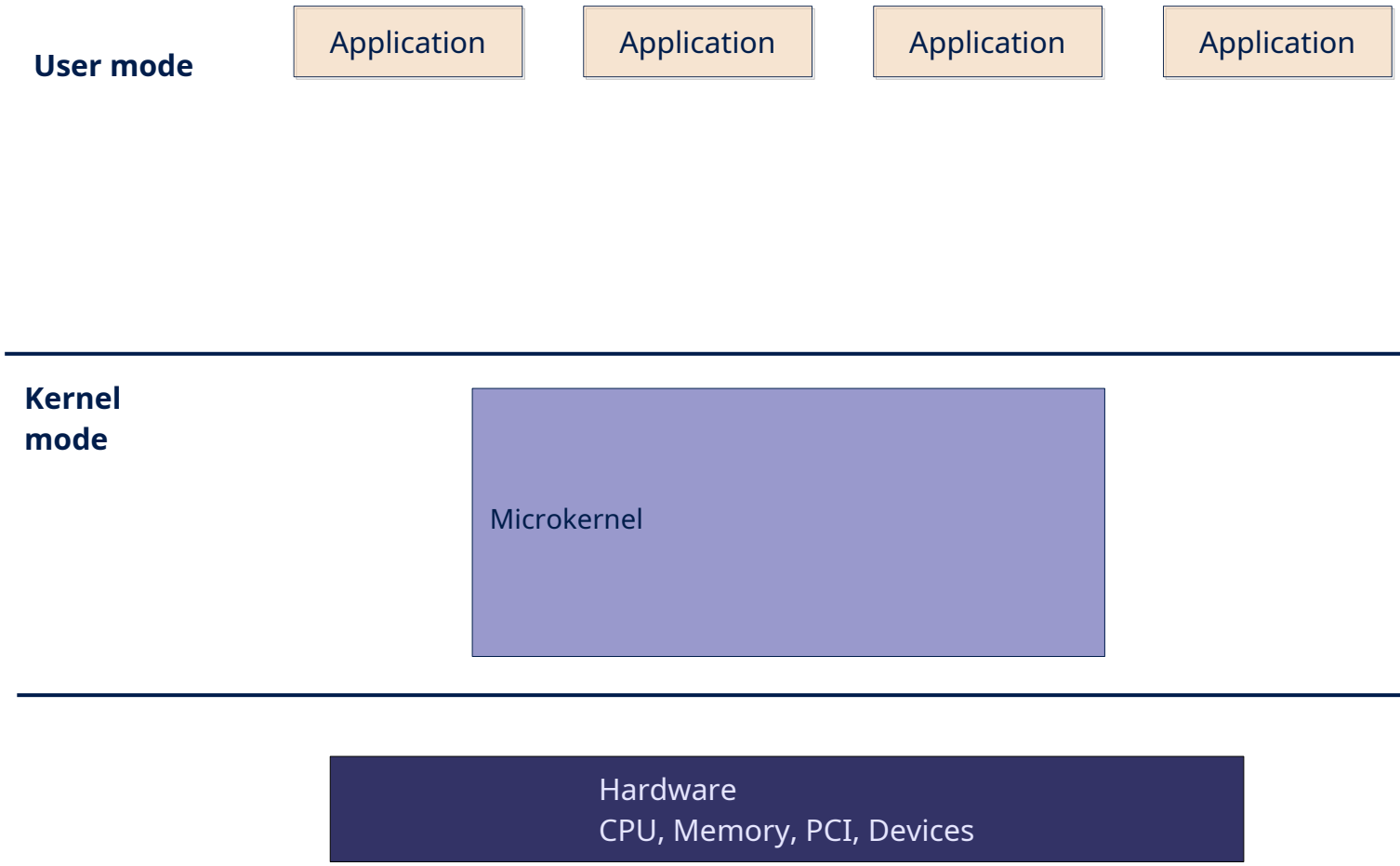
The Microkernel Vision

- Minimal OS kernel
  - Less error prone (less code → fewer errors)
  - Small *Trusted Computing Base*
  - Suitable for verification
- System services in user-level *servers*
  - Flexible and extensible

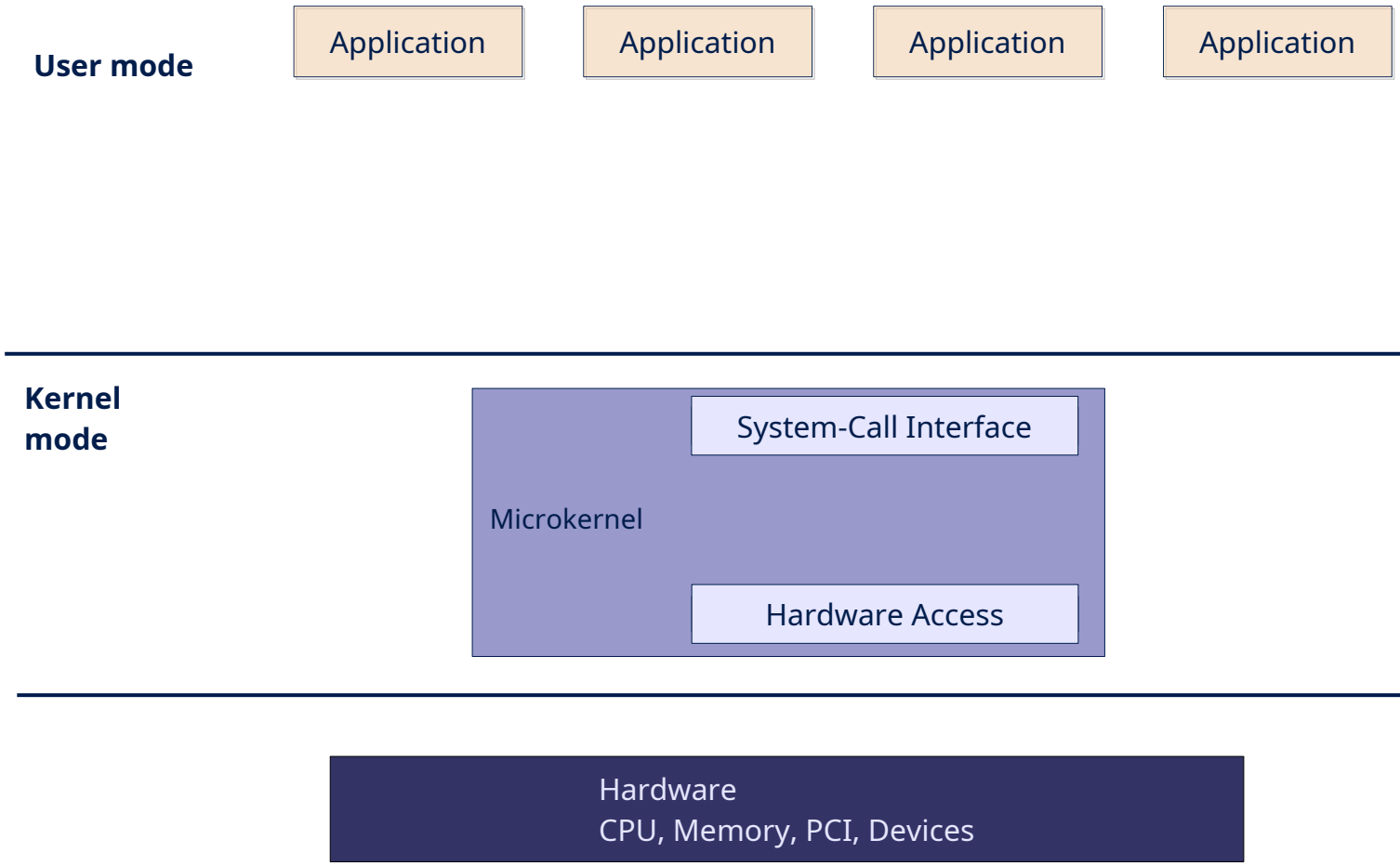
The Microkernel Vision

- Minimal OS kernel
  - Less error prone (less code → fewer errors)
  - Small *Trusted Computing Base*
  - Suitable for verification
- System services in user-level *servers*
  - Flexible and extensible
- Protection between individual components
  - More resilient: crashing component does not (necessarily…) crash the whole system
  - More secure: inter-component protection

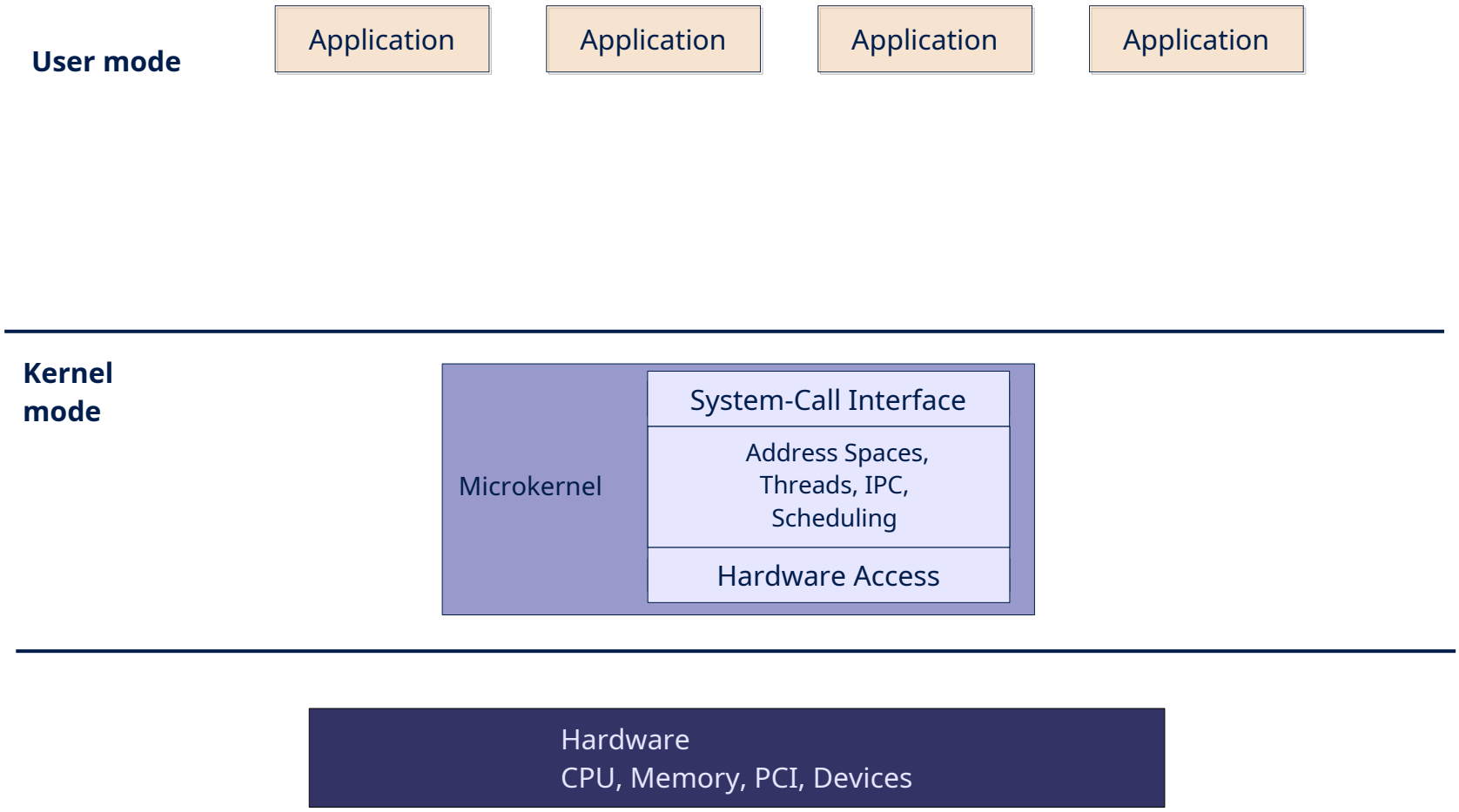
The Microkernel Vision



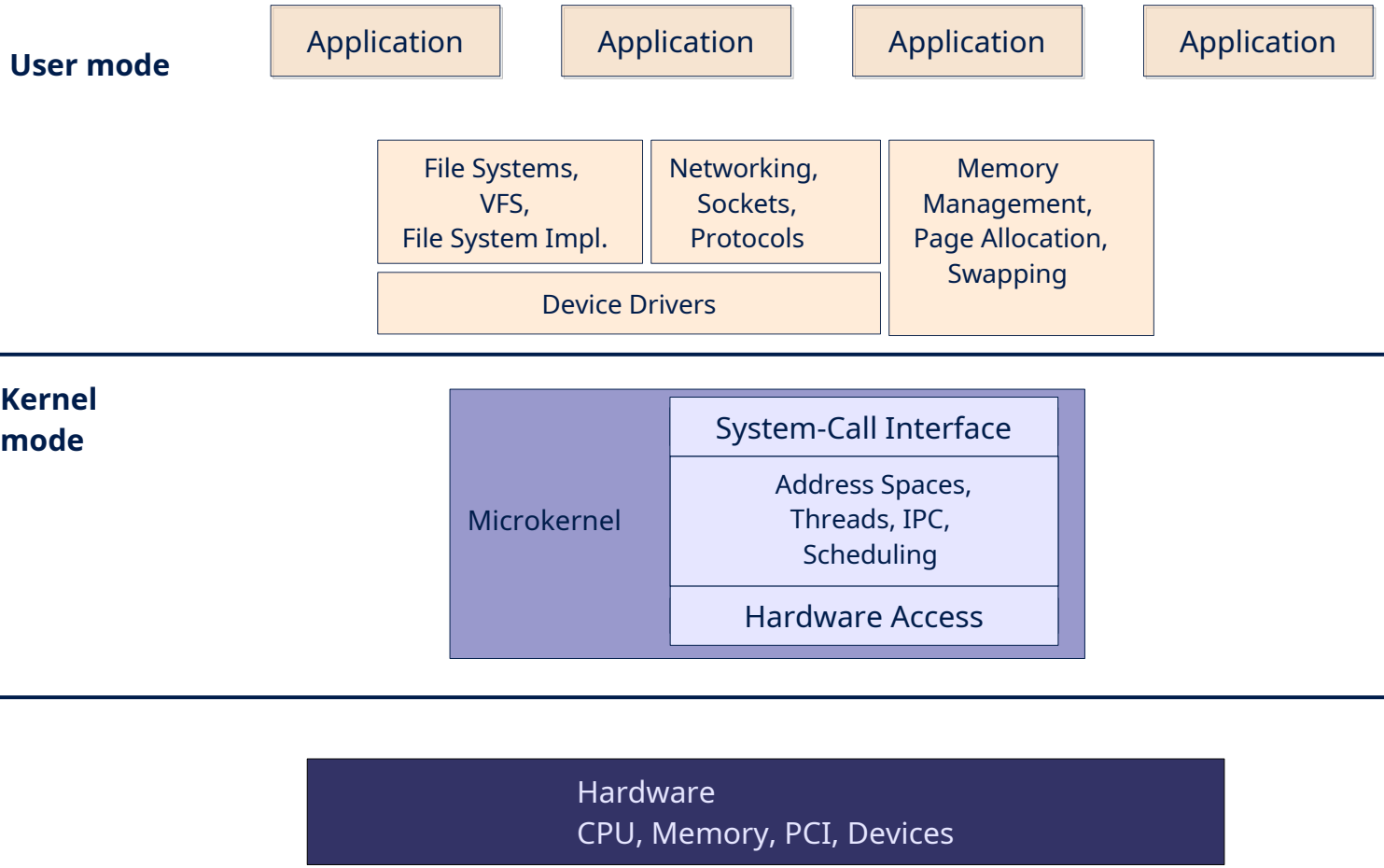
The Microkernel Vision



The Microkernel Vision



The Microkernel Vision



What Microkernels Can Give Us...

- OS personalities

What Microkernels Can Give Us...

- OS personalities
- Customisability
  - Servers may be configured to suit the target system (small embedded systems, desktop PCs, SMP systems, ...)
  - Remove unnecessary servers

What Microkernels Can Give Us...

- OS personalities
- Customisability
  - Servers may be configured to suit the target system (small embedded systems, desktop PCs, SMP systems, ...)
  - Remove unnecessary servers
- Enforce reasonable system design
  - Well-defined interfaces between components
  - Access to components only via these interfaces
  - Improved maintainability

Mach: A 1st-generation Microkernel

- Developed at CMU, 1985 – 1994
  - Rick Rashid (former head of MS Research)
  - Avie Tevanian (former Apple CTO)
  - Brian Bershad (professor @ Univ. of Washington)
  - …

Mach: A 1st-generation Microkernel

- Developed at CMU, 1985 – 1994
  - Rick Rashid (former head of MS Research)
  - Avie Tevanian (former Apple CTO)
  - Brian Bershad (professor @ Univ. of Washington)
  - …
- Foundation for several real systems
  - Single Server Unix (BSD4.3 on Mach)
  - MkLinux (OSF, Apple)
  - IBM Workplace OS
  - NeXT OS

Mach: A 1st-generation Microkernel

- Developed at CMU, 1985 – 1994
  - Rick Rashid (former head of MS Research)
  - Avie Tevanian (former Apple CTO)
  - Brian Bershad (professor @ Univ. of Washington)
  - …
- Foundation for several real systems
  - Single Server Unix (BSD4.3 on Mach)
  - MkLinux (OSF, Apple)
  - IBM Workplace OS
  - NeXT OS → Mac OS X

Mach: Overview

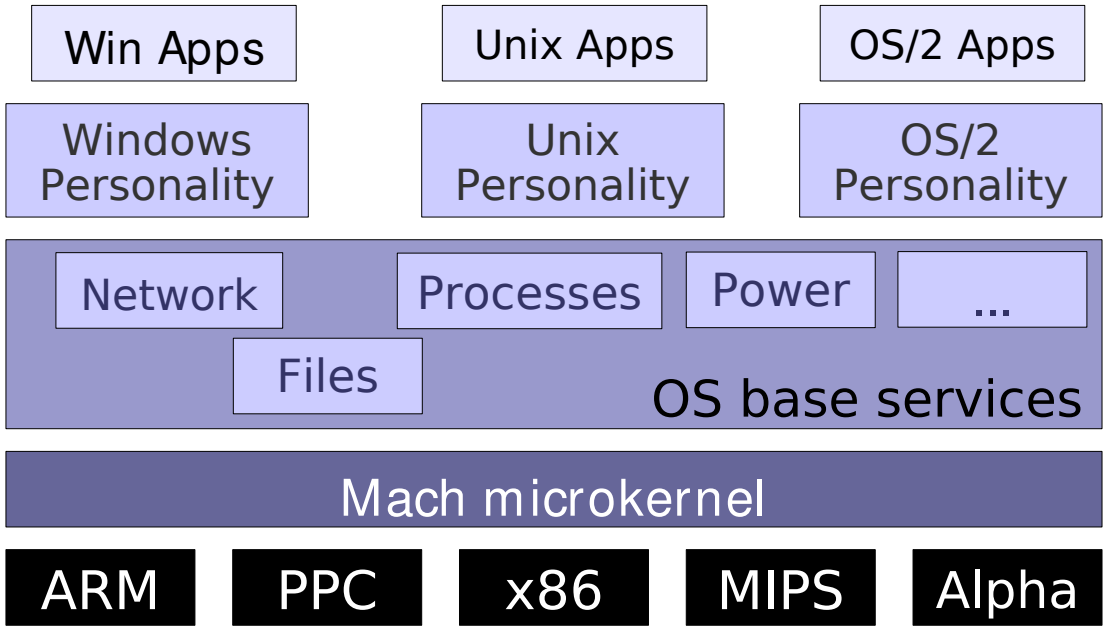
- Simple, extensible *communication kernel*
  - “Everything is a pipe.”
  - *Ports* as secure communication channels
- Multiprocessor support
- Message passing by mapping
- Multi-server OS personality
- POSIX compatibility

Mach: Overview

- Simple, extensible *communication kernel*
  - “Everything is a pipe.”
  - *Ports* as secure communication channels
- Multiprocessor support
- Message passing by mapping
- Multi-server OS personality
- POSIX compatibility
- Shortcomings
  - Performance
  - Drivers in the kernel

Case Study: IBM Workplace OS

- Main goals:
  - Multiple OS personalities
  - Support for multiple HW architectures



IBM Workplace OS: A Failure?

- Never finished … but almost 2 billion US-$ spent
- Causes of failure:

IBM Workplace OS: A Failure?

- Never finished … but almost 2 billion US-$ spent
- Causes of failure:
  - Underestimated difficulties in creating OS personalities

IBM Workplace OS: A Failure?

- Never finished … but almost 2 billion US-$ spent
- Causes of failure:
  - Underestimated difficulties in creating OS personalities
  - Management errors: divisions forced to adopt new system without having a system

IBM Workplace OS: A Failure?

- Never finished … but almost 2 billion US-$ spent
- Causes of failure:
  - Underestimated difficulties in creating OS personalities
  - Management errors: divisions forced to adopt new system without having a system
  - “Second System Effect”: too many fancy features

IBM Workplace OS: A Failure?

- Never finished … but almost 2 billion US-$ spent
- Causes of failure:
  - Underestimated difficulties in creating OS personalities
  - Management errors: divisions forced to adopt new system without having a system
  - “Second System Effect”: too many fancy features
  - Too slow

IBM Workplace OS: A Failure?

- Never finished … but almost 2 billion US-$ spent
- Causes of failure:
  - Underestimated difficulties in creating OS personalities
  - Management errors: divisions forced to adopt new system without having a system
  - “Second System Effect”: too many fancy features
  - Too slow
- Conclusion: Microkernel worked, but system atop the microkernel did not

IBM Workplace OS: Lessons Learned

- OS personalities did not work

IBM Workplace OS: Lessons Learned

- OS personalities did not work
- Flexibility … but monolithic kernels became flexible, too (e.g., Linux kernel modules)

IBM Workplace OS: Lessons Learned

- OS personalities did not work
- Flexibility … but monolithic kernels became flexible, too (e.g., Linux kernel modules)
- Better design … but monolithic kernels also improved (restricted symbol access, layered architectures)

IBM Workplace OS: Lessons Learned

- OS personalities did not work
- Flexibility … but monolithic kernels became flexible, too (e.g., Linux kernel modules)
- Better design … but monolithic kernels also improved (restricted symbol access, layered architectures)
- Maintainability … still very complex

IBM Workplace OS: Lessons Learned

- OS personalities did not work
- Flexibility … but monolithic kernels became flexible, too (e.g., Linux kernel modules)
- Better design … but monolithic kernels also improved (restricted symbol access, layered architectures)
- Maintainability … still very complex
- Performance matters a lot

Microkernels: Proven Advantages

- Subsystem protection / isolation
- Code size (generated using David A. Wheeler's "SLOCCount")

Microkernels: Proven Advantages

- Subsystem protection / isolation
- Code size (generated using David A. Wheeler's "SLOCCount")
  - Microkernel-based OS, x86 architecture
    - Fiasco kernel: ~34k LoC
    - "HelloWorld" (+ boot loader + root task): ~10k LoC

Microkernels: Proven Advantages

- Subsystem protection / isolation
- Code size (generated using David A. Wheeler's "SLOCCount")
  - Microkernel-based OS, x86 architecture
    - Fiasco kernel: ~34k LoC
    - "HelloWorld" (+ boot loader + root task): ~10k LoC
  - Linux 6.17, x86 architecture
    - Kernel (kernel + arch/x86): ~600k LoC
    - Subsystems (net, fs, sound, ...): ~3.7M LoC
    - Drivers: ~20M LoC

Microkernels: Proven Advantages

- Subsystem protection / isolation
- Code size (generated using David A. Wheeler's "SLOCCount")
  - Microkernel-based OS, x86 architecture
    - Fiasco kernel: ~34k LoC
    - "HelloWorld" (+ boot loader + root task): ~10k LoC
  - Linux 6.17, x86 architecture
    - Kernel (kernel + arch/x86): ~600k LoC
    - Subsystems (net, fs, sound, ...): ~3.7M LoC
    - Drivers: ~20M LoC
- Customisability
  - Tailored memory management / scheduling / ... algorithms
  - Adaptable to embedded / real-time / secure / ... systems

Challenges

- We need fast and efficient kernels
  - Covered in the “Microkernel Construction” lecture in the summer term

Challenges

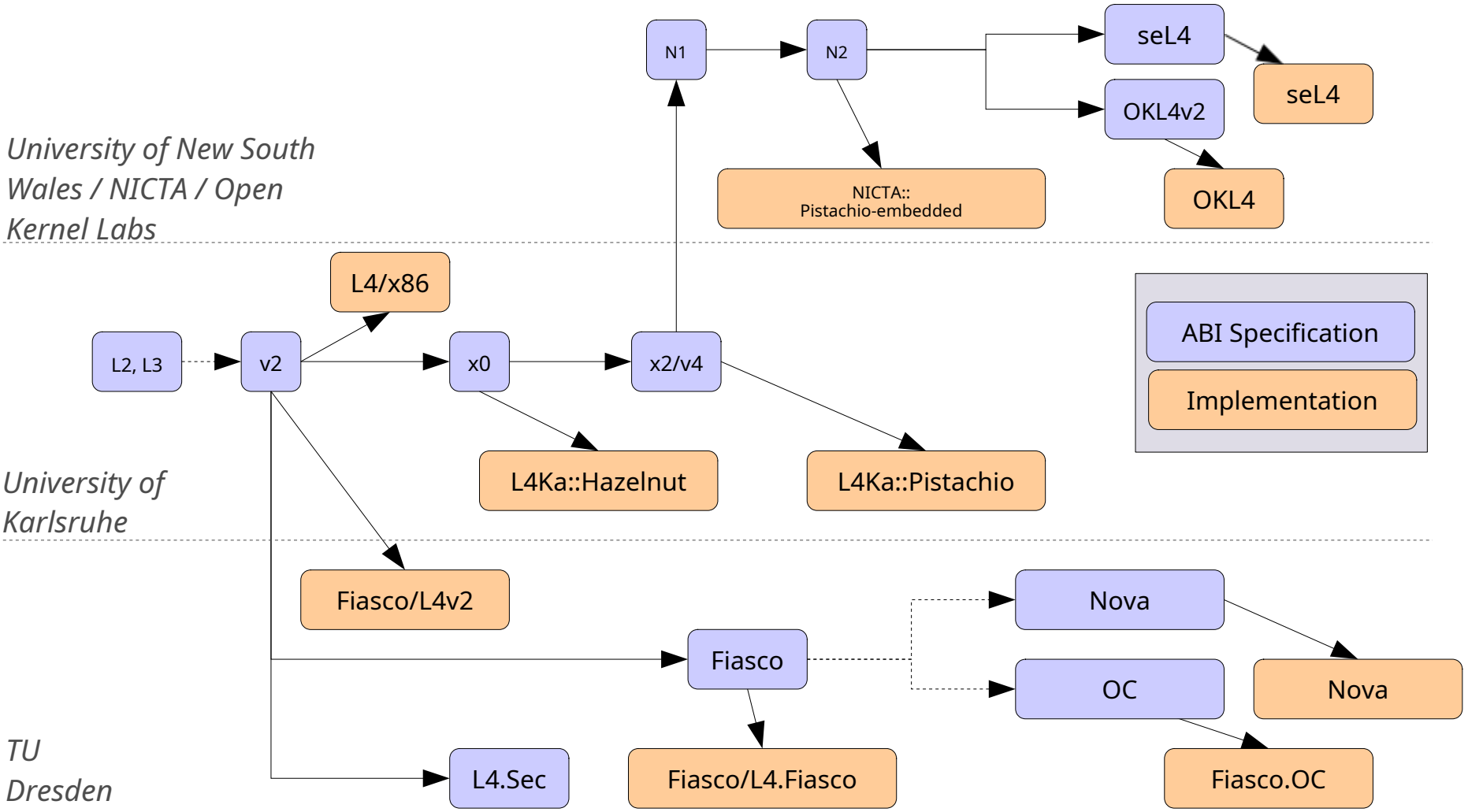
- We need fast and efficient kernels
  - Covered in the “Microkernel Construction” lecture in the summer term
- We need fast and efficient OS services
  - Memory and resource management
  - Synchronisation
  - Device Drivers
  - File systems
  - Communication interfaces

  → Subject of this lecture

Who's Out There?

- Minix @ FU Amsterdam (Andrew Tanenbaum)
- Singularity @ MS Research
- EROS/CoyotOS @ Johns Hopkins University
- The L4 Microkernel Family
  - Originally developed by Jochen Liedtke at IBM and GMD
  - 2nd-generation microkernel
  - Several kernel ABI versions

The L4 Family Tree



L4 Key Concepts

- Jochen Liedtke:
  “A microkernel does no real work.”
  - The kernel only provides inevitable mechanisms.
  - The kernel does not enforce policies.

L4 Key Concepts

- Jochen Liedtke:
  “A microkernel does no real work.”
  - The kernel only provides inevitable mechanisms.
  - The kernel does not enforce policies.
- But what *is* inevitable?

L4 Key Concepts

- Jochen Liedtke:
  “A microkernel does no real work.”
  - The kernel only provides inevitable mechanisms.
  - The kernel does not enforce policies.
- But what *is* inevitable?
  - Abstractions
    - Threads
    - Address spaces (tasks)

L4 Key Concepts

- Jochen Liedtke:
  “A microkernel does no real work.”
  - The kernel only provides inevitable mechanisms.
  - The kernel does not enforce policies.
- But what *is* inevitable?
  - Abstractions
    - Threads
    - Address spaces (tasks)
  - Mechanisms
    - Communication
    - Resource mapping
    - (Scheduling)

Taking a closer look at L4

Case study: **L4/Fiasco.OC**

- “Everything is an object”

- “Everything is an object”
  - Task Address space

- “Everything is an object”
  - Task Address space
  - Thread Activities, scheduling

- “Everything is an object”
  - Task Address space
  - Thread Activities, scheduling
  - IPC Gate Communication, resource mapping

- “Everything is an object”
  - Task Address space
  - Thread Activities, scheduling
  - IPC Gate Communication, resource mapping
  - IRQ Communication

- “Everything is an object”
  - Task Address space
  - Thread Activities, scheduling
  - IPC Gate Communication, resource mapping
  - IRQ Communication
  - Factory Create other objects, enforce quotas

- “Everything is an object”
  - Task Address space
  - Thread Activities, scheduling
  - IPC Gate Communication, resource mapping
  - IRQ Communication
  - Factory Create other objects, enforce quotas
- One system call: **`invoke_object()`**
  - Parameters passed in UTCB
  - Types of parameters depend on type of object

L4/Fiasco.OC: Kernel Objects

- Kernel-provided objects
  - Threads
  - Tasks
  - IRQs
  - …

L4/Fiasco.OC: Kernel Objects

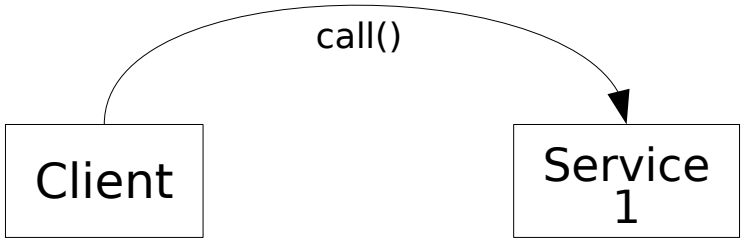
- Kernel-provided objects
  - Threads
  - Tasks
  - IRQs
  - …
- Generic communication object: IPC gate
  - Send message from sender to receiver
  - Allows to implement *new objects* in *user-level* applications

L4/Fiasco.OC: User-Level Objects

- Build everything above kernel using user-level objects that provide a service
  - Networking stack
  - File system
  - …

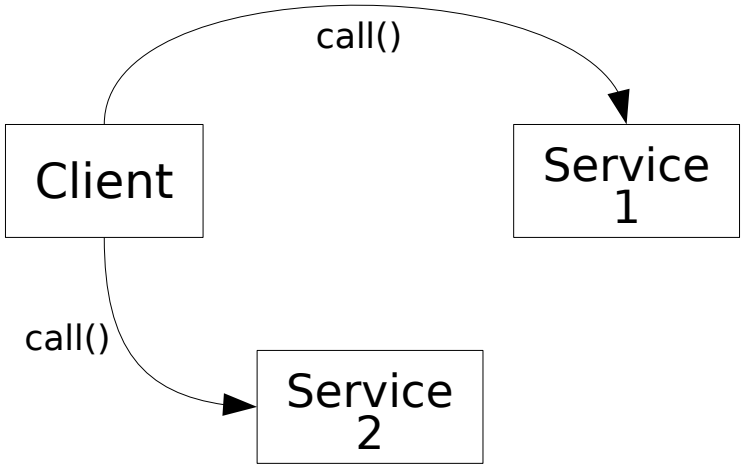
L4/Fiasco.OC: User-Level Objects

- Build everything above kernel using user-level objects that provide a service
  - Networking stack
  - File system
  - …



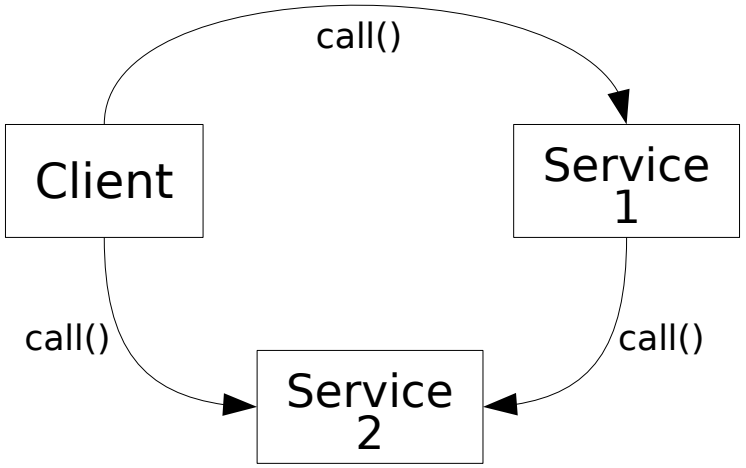
L4/Fiasco.OC: User-Level Objects

- Build everything above kernel using user-level objects that provide a service
  - Networking stack
  - File system
  - …



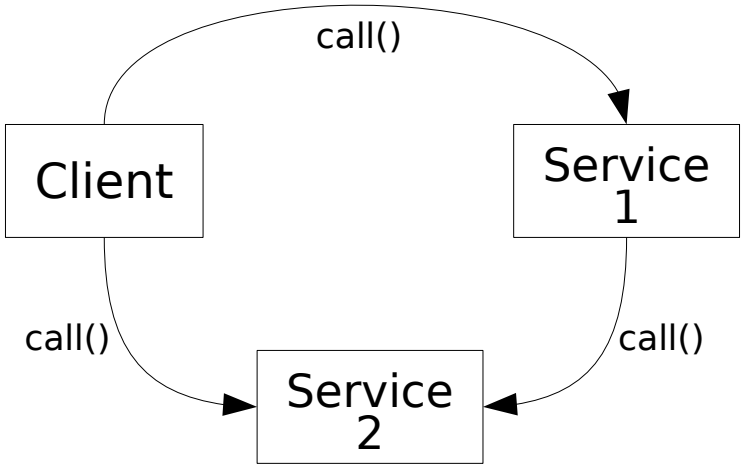
L4/Fiasco.OC: User-Level Objects

- Build everything above kernel using user-level objects that provide a service
  - Networking stack
  - File system
  - …



L4/Fiasco.OC: User-Level Objects

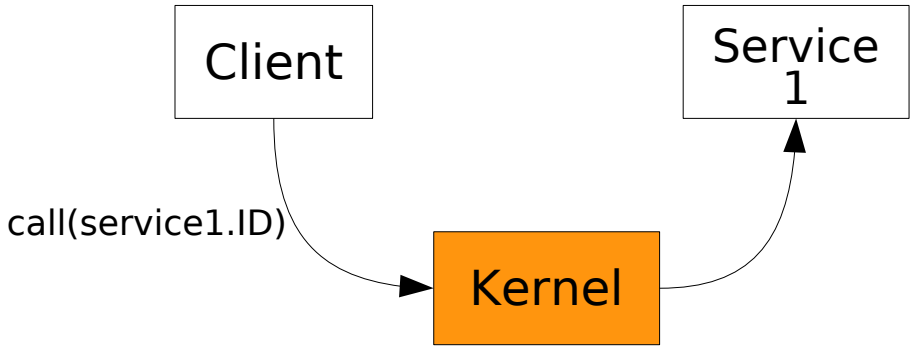
- Build everything above kernel using user-level objects that provide a service
  - Networking stack
  - File system
  - …



- Kernel provides
  - Object creation/management
  - Object interaction: Inter-Process Communication (IPC)

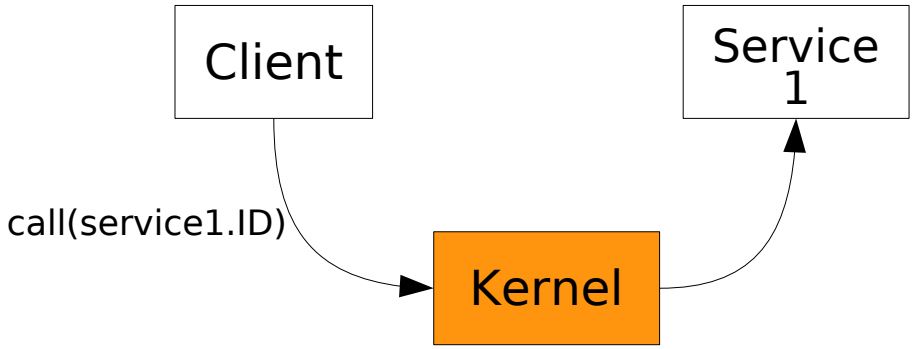
L4/Fiasco.OC: How to Use an Object?

- To call an object, we need an address:
  - Telephone number
  - Postal address
  - IP address
  - ...



L4/Fiasco.OC: How to Use an Object?

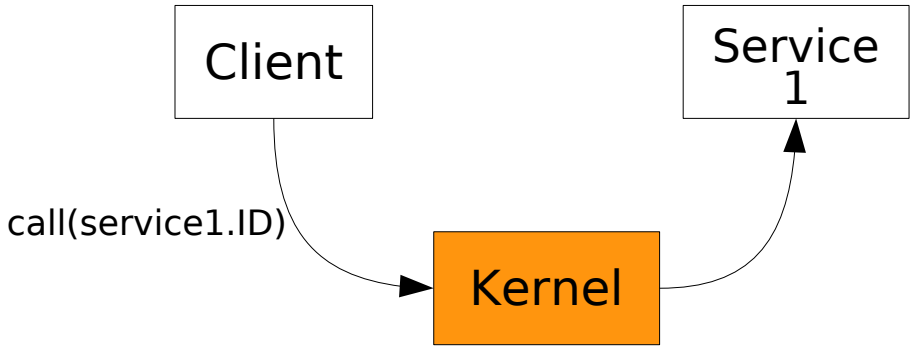
- To call an object, we need an address:
  - Telephone number
  - Postal address
  - IP address
  - …



- Simple idea, isn't it?

L4/Fiasco.OC: How to Use an Object?

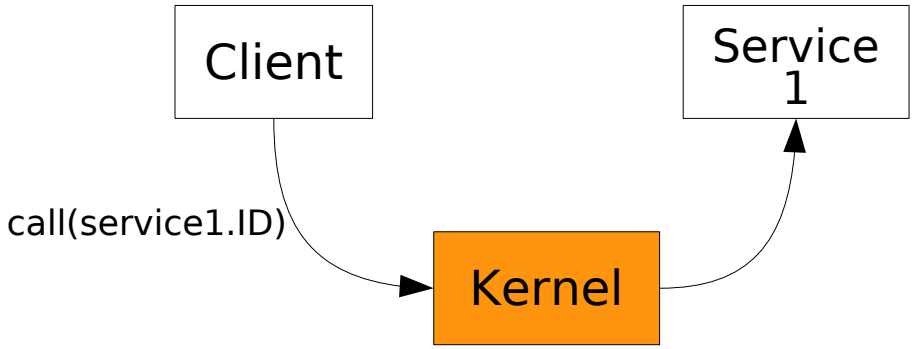
- To call an object, we need an address:
  - Telephone number
  - Postal address
  - IP address
  - …



- Simple idea, isn't it?
- ID is wrong? Kernel returns ENOTEXIST

L4/Fiasco.OC: How to Use an Object?

- To call an object, we need an address:
  - Telephone number
  - Postal address
  - IP address
  - …



- Simple idea, isn't it?
- ID is wrong? Kernel returns ENOTEXIST
- But not so fast! This scheme is insecure:
  - Client could simply "guess" IDs brute-force
  - (Non-)Existence can be used as a covert channel

L4/Fiasco.OC: Capabilities

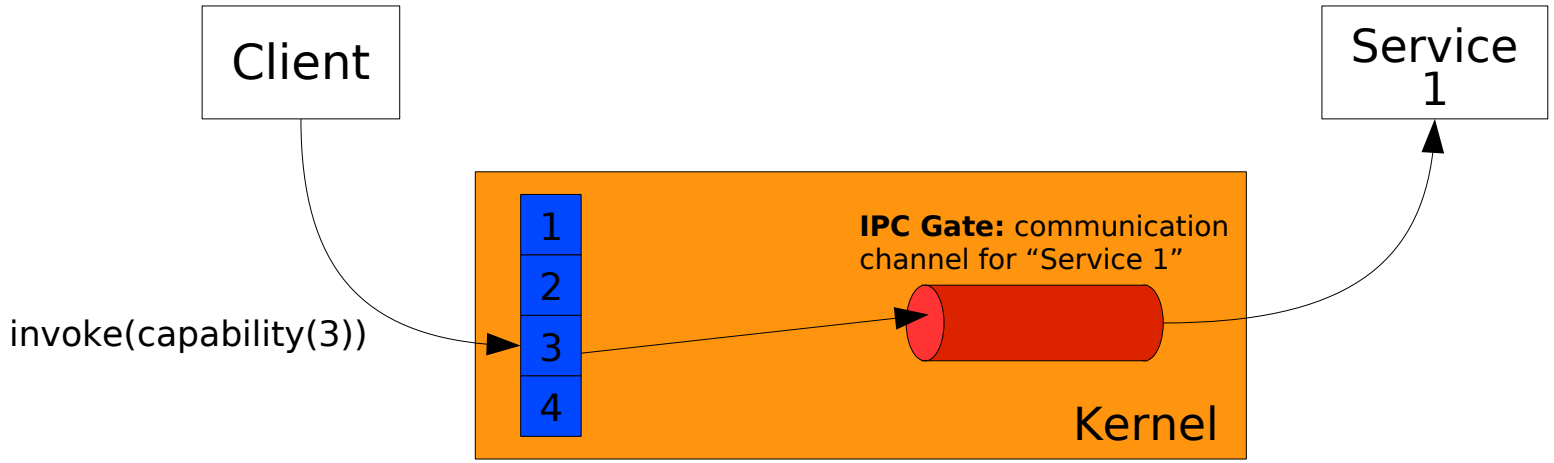
- Global object IDs are
  - insecure (forgery, covert channels)
  - inconvenient (programmer needs to know about partitioning in advance)

L4/Fiasco.OC: Capabilities

- Global object IDs are
  - insecure (forgery, covert channels)
  - inconvenient (programmer needs to know about partitioning in advance)
- Solution in Fiasco.OC
  - Task-local *capability space* as indirection
  - *Object capability* required to invoke object
  - Per-task name space
    - Maps names to object capabilities
    - Configured by task's creator

L4/Fiasco.OC: Object Capabilities

- Capability:
  - Reference to an object
  - Protected by the kernel
    - Kernel knows all capability–object mappings
    - Managed as a per-process capability table
    - User processes only use indices into this table



L4/Fiasco.OC: Communication

- Kernel object for communication: *IPC gate*

L4/Fiasco.OC: Communication

- Kernel object for communication: *IPC gate*
- Inter-process communication (IPC)
  - Between threads
  - Synchronous

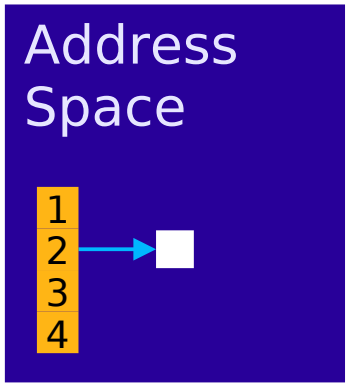
L4/Fiasco.OC: Communication

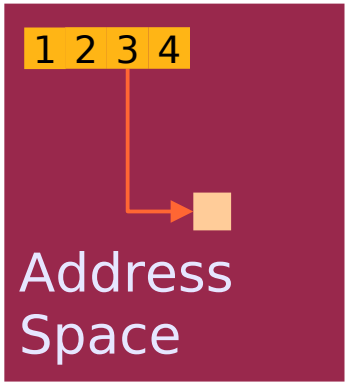
- Kernel object for communication: *IPC gate*
- Inter-process communication (IPC)
  - Between threads
  - Synchronous
- Sequence:
  - Sender writes message into its UTCB
  - Sender invokes IPC gate → blocks sender until receiver ready (i.e., waits for message)

L4/Fiasco.OC: Communication

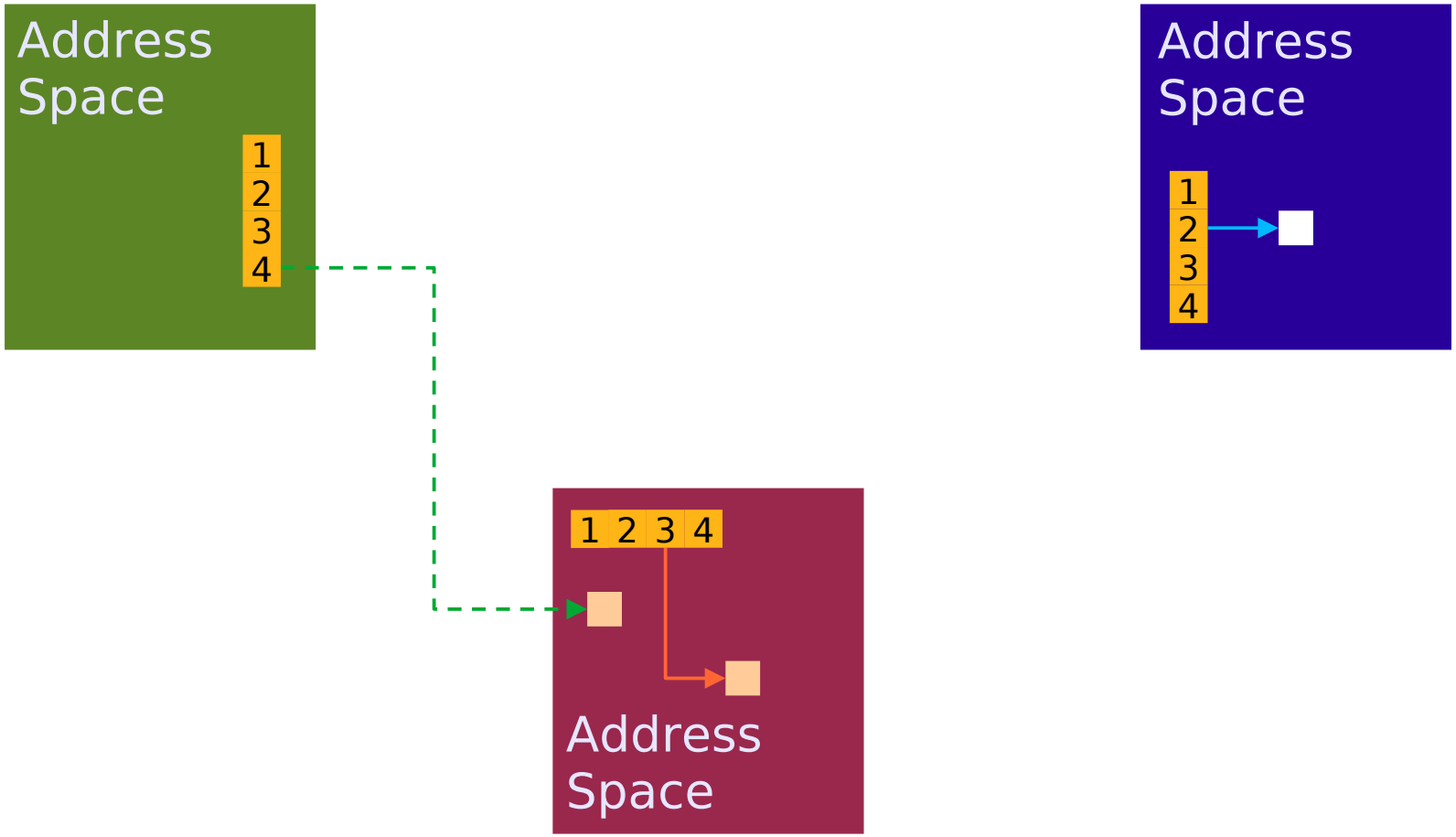
- Kernel object for communication: *IPC gate*
- Inter-process communication (IPC)
  - Between threads
  - Synchronous
- Sequence:
  - Sender writes message into its UTCB
  - Sender invokes IPC gate → blocks sender until receiver ready (i.e., waits for message)
  - Kernel copies message to receiver thread's UTCB
  - Both continue, knowing that message has been transferred/received

Capabilities = Local Names

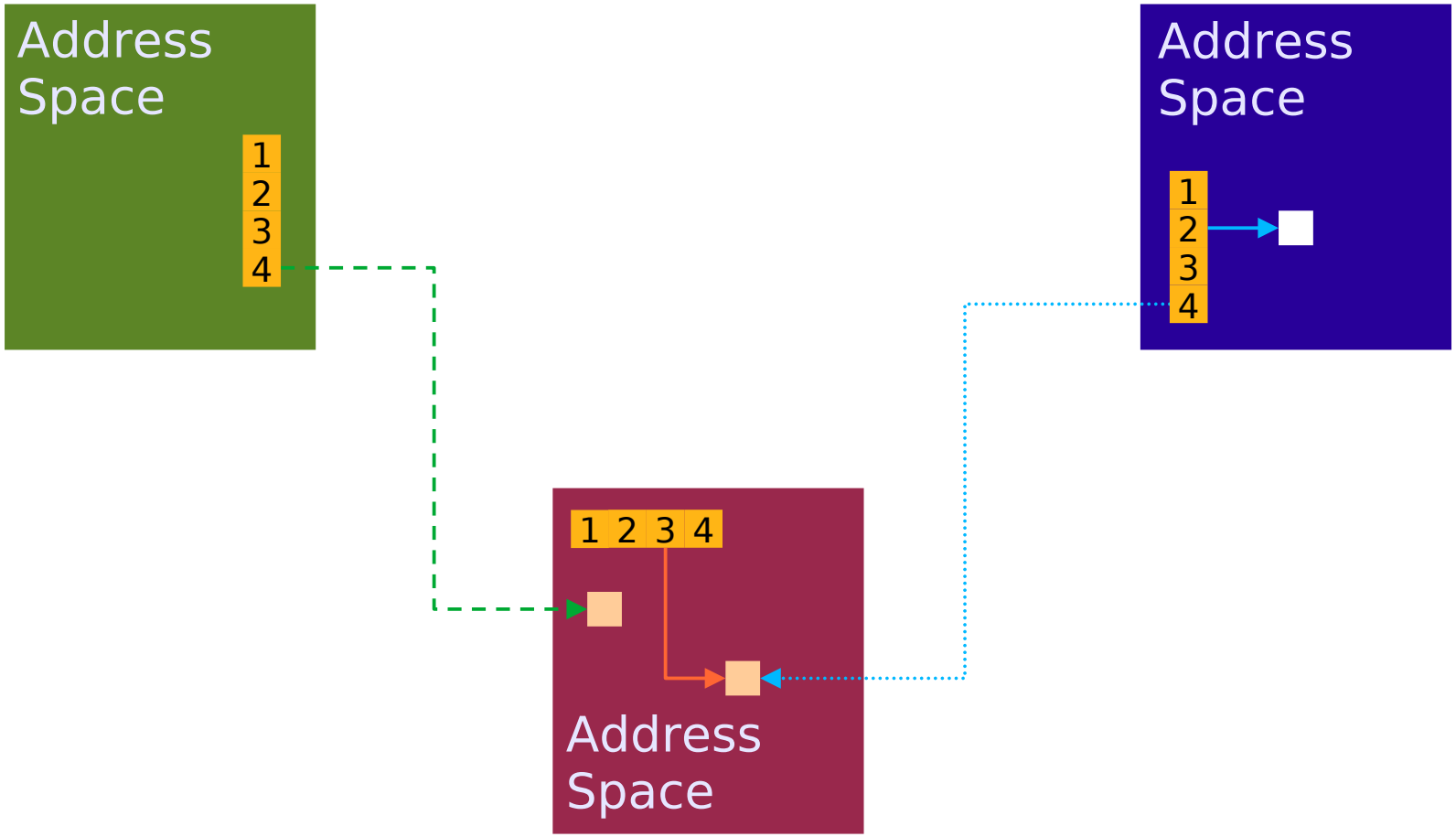




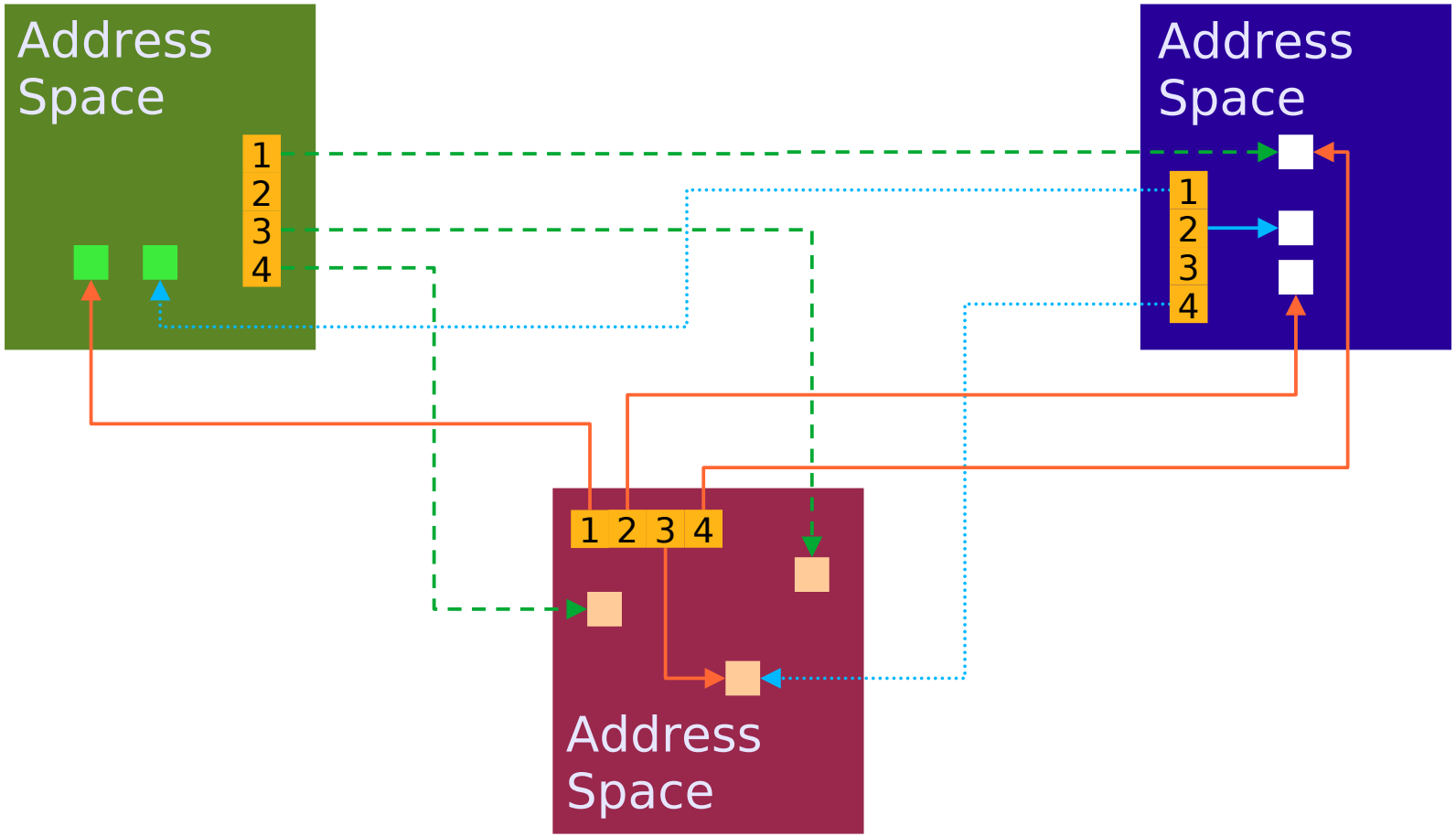
Capabilities = Local Names



Capabilities = Local Names



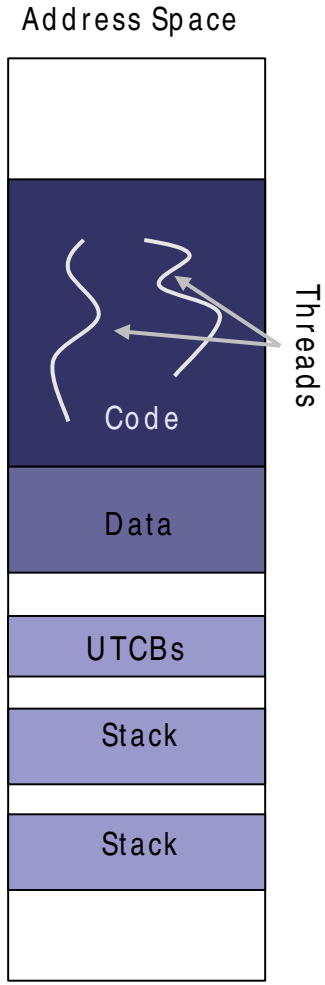
Capabilities = Local Names



More L4 concepts

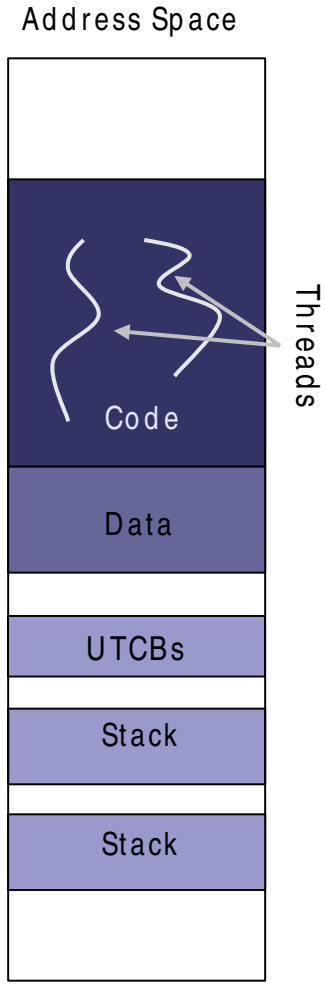
L4/Fiasco.OC: Threads

- Thread
  - Unit of execution
  - Implemented as kernel object



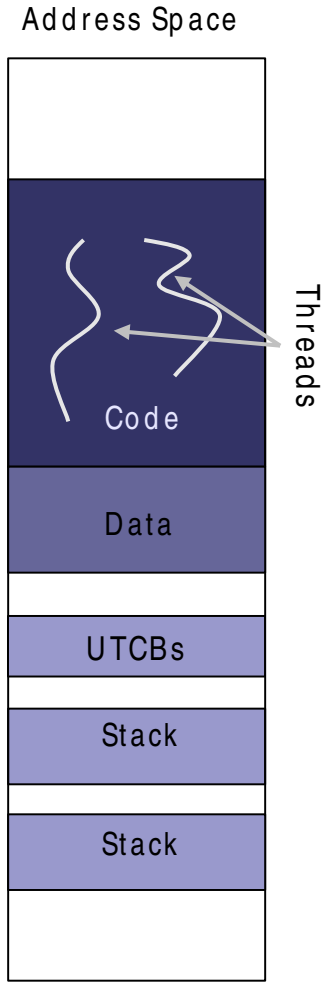
L4/Fiasco.OC: Threads

- Thread
  - Unit of execution
  - Implemented as kernel object
- Properties managed by the kernel
  - Instruction Pointer (EIP)
  - Stack (ESP)
  - Registers
  - User-level thread control block (UTCB)



L4/Fiasco.OC: Threads

- Thread
  - Unit of execution
  - Implemented as kernel object
- Properties managed by the kernel
  - Instruction Pointer (EIP)
  - Stack (ESP)
  - Registers
  - User-level thread control block (UTCB)
- User-level applications need to
  - allocate stack memory
  - provide memory for application binary
  - find entry point
  - …



L4/Fiasco.OC: Interrupts

- Kernel object: IRQ
- Used for hardware and software interrupts
- Provides asynchronous signaling
  - `invoke_object(irq_cap, WAIT)`
  - `invoke_object(irq_cap, TRIGGER)`

L4/Fiasco.OC: Interrupts

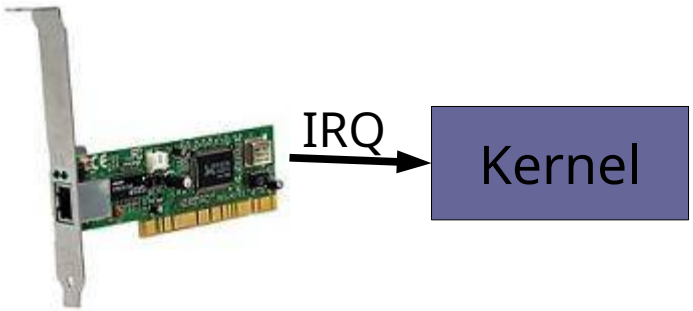
- Kernel object: IRQ
- Used for hardware and software interrupts
- Provides asynchronous signaling
  - `invoke_object(irq_cap, WAIT)`
  - `invoke_object(irq_cap, TRIGGER)`

Kernel

User-space device driver

L4/Fiasco.OC: Interrupts

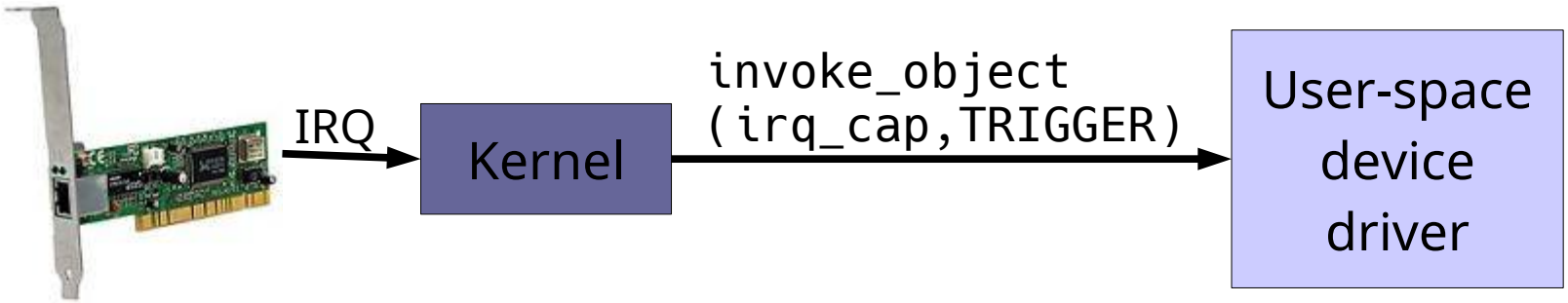
- Kernel object: IRQ
- Used for hardware and software interrupts
- Provides asynchronous signaling
  - `invoke_object(irq_cap, WAIT)`
  - `invoke_object(irq_cap, TRIGGER)`



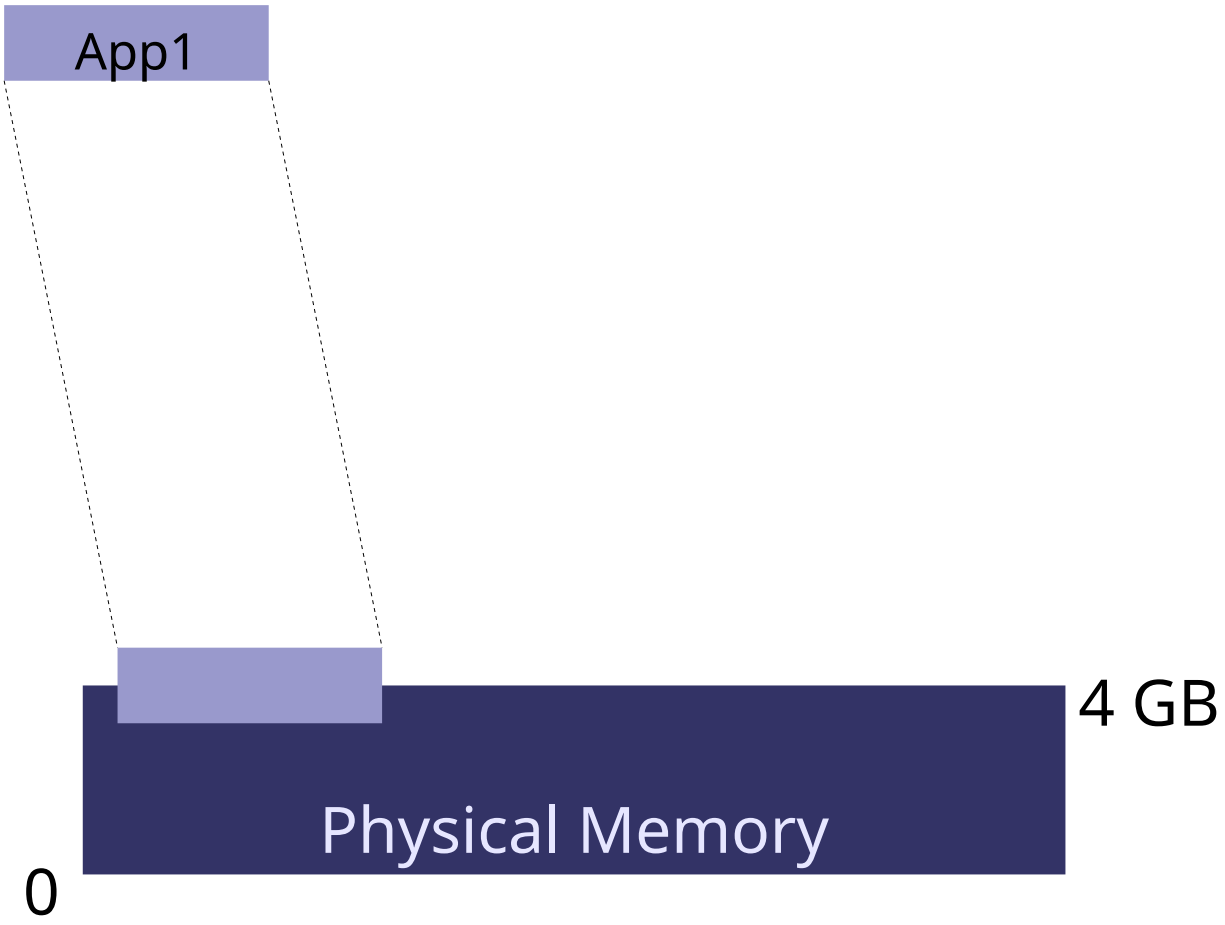
User-space device driver

L4/Fiasco.OC: Interrupts

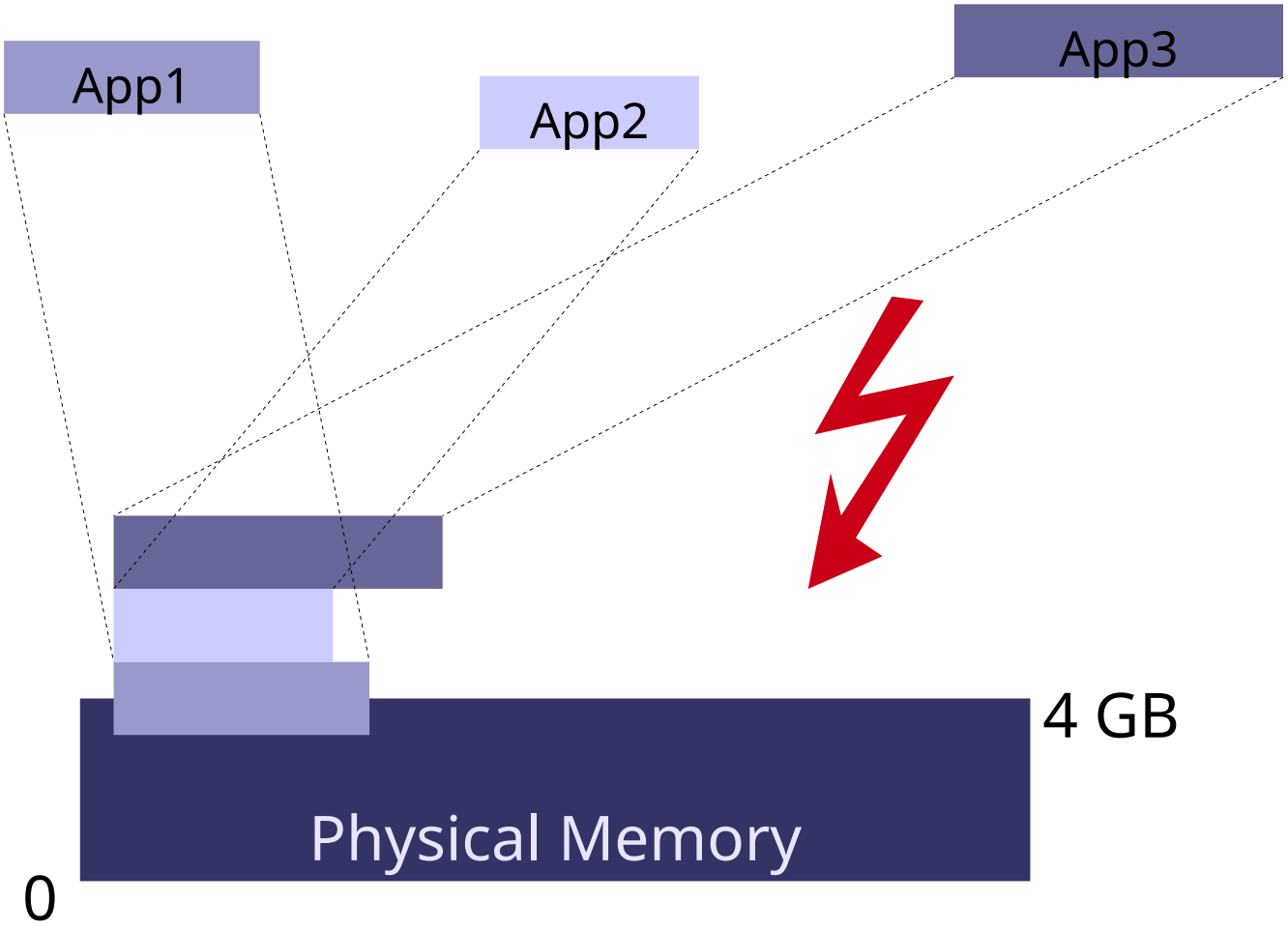
- Kernel object: IRQ
- Used for hardware and software interrupts
- Provides asynchronous signaling
  - `invoke_object(irq_cap, WAIT)`
  - `invoke_object(irq_cap, TRIGGER)`



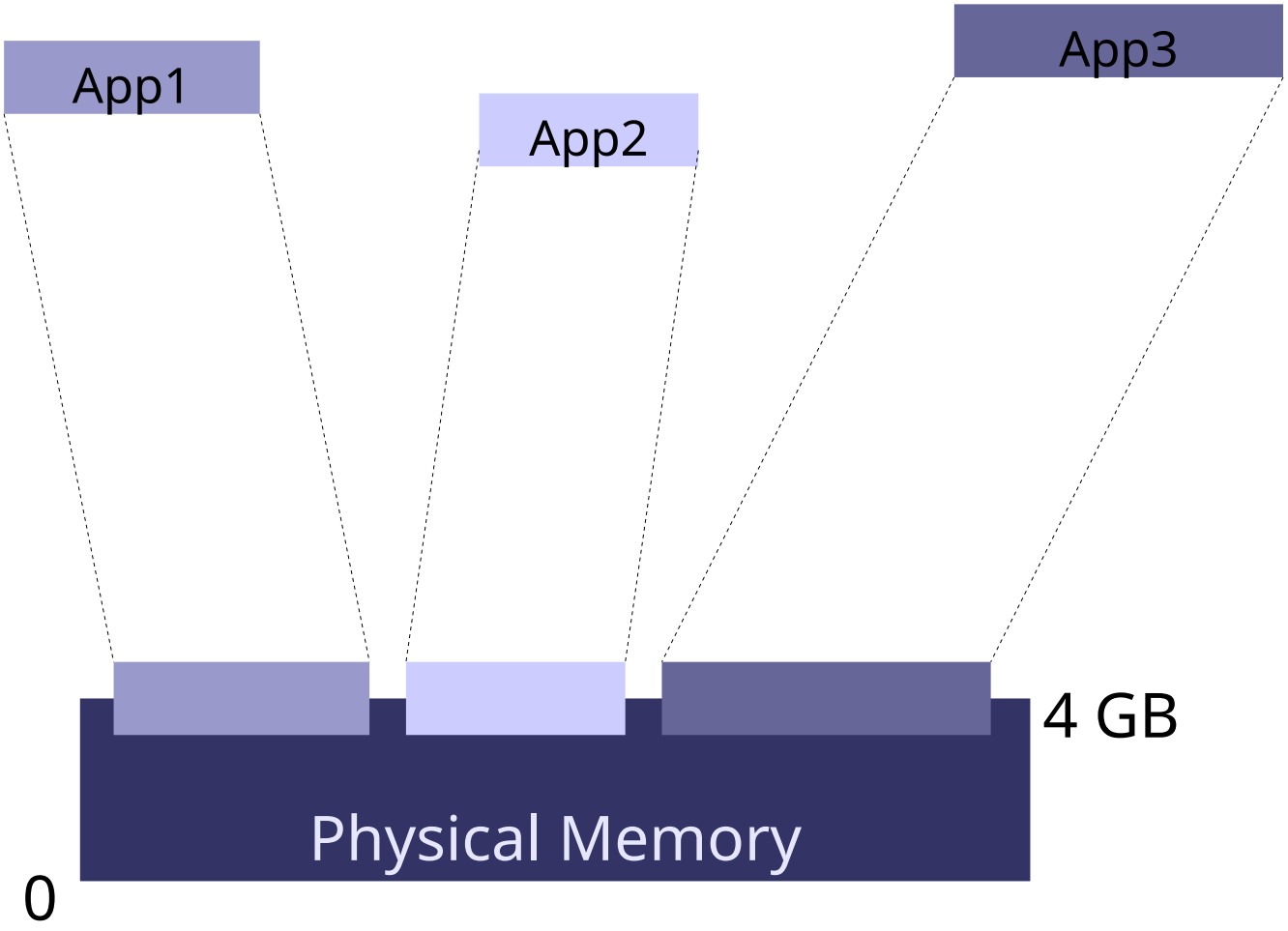
Challenge: Memory partitioning

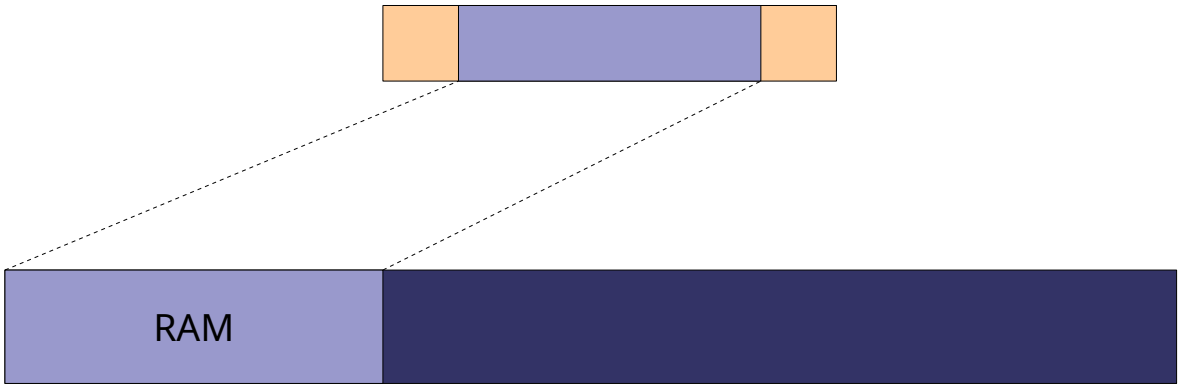


Challenge: Memory partitioning



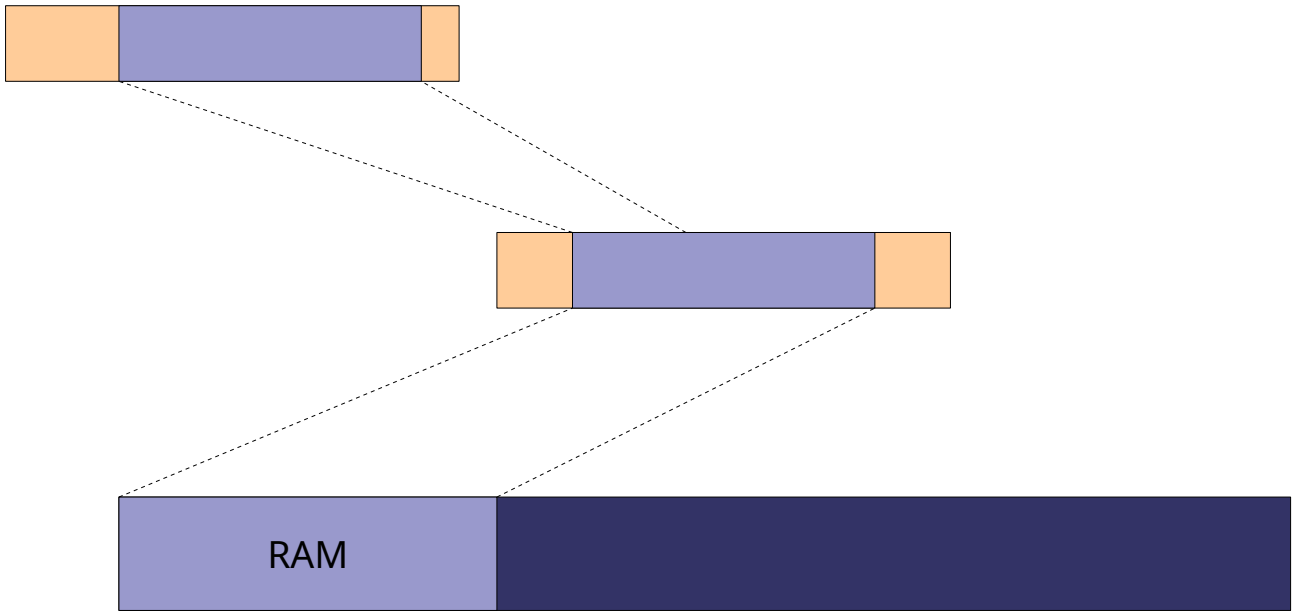
Solution: Virtual Memory





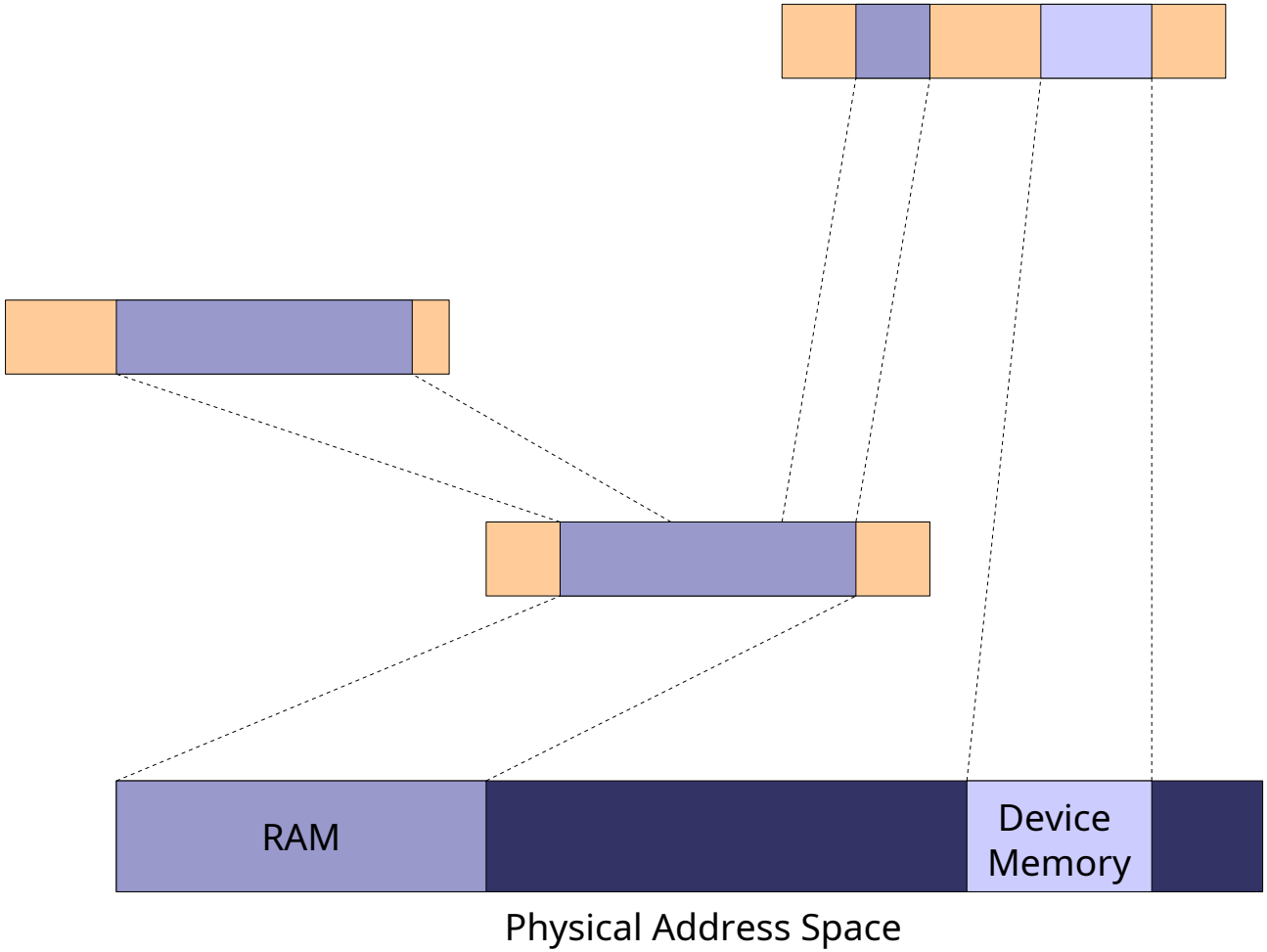
Physical Address Space

L4: Recursive Address Spaces

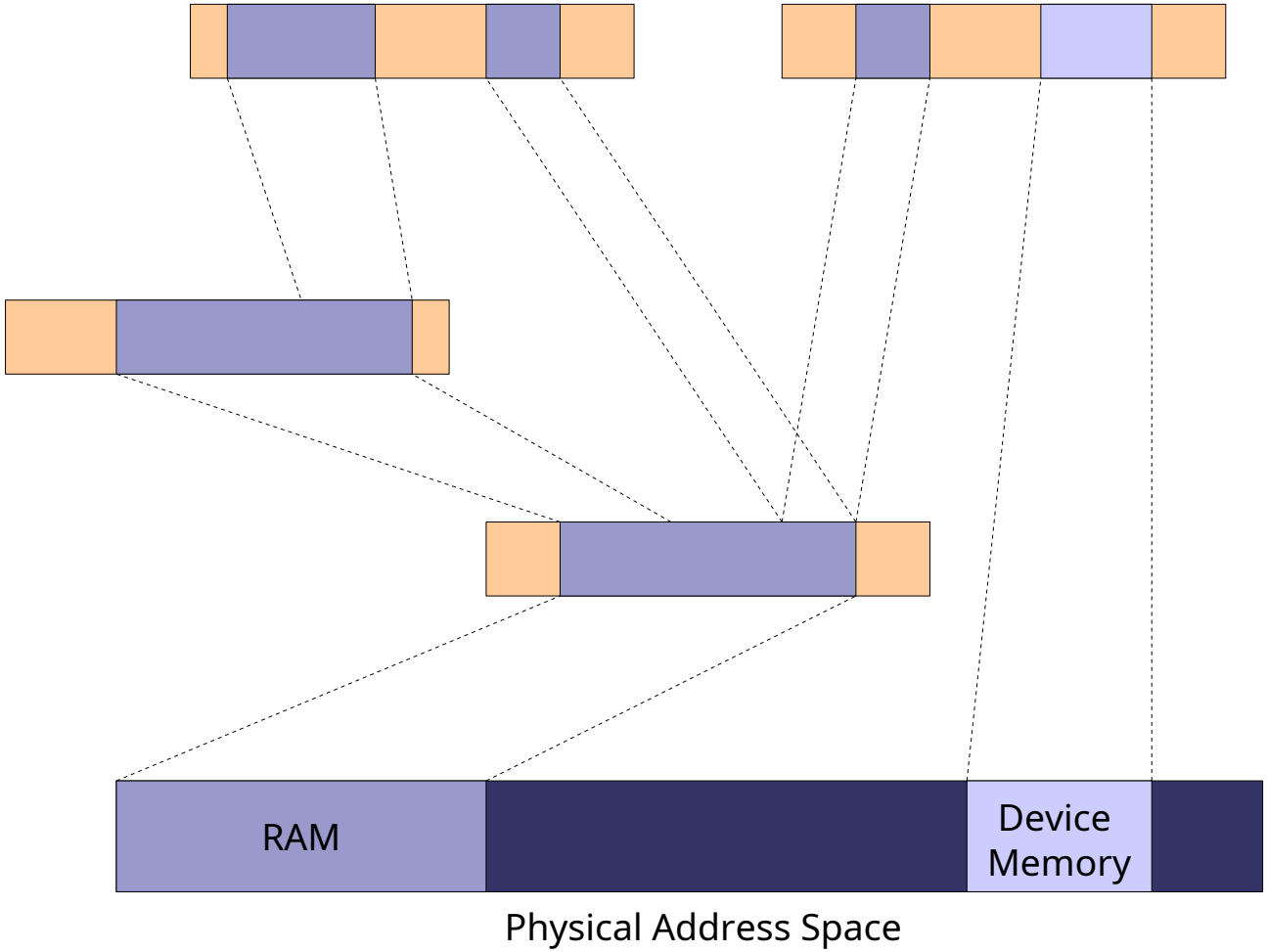


Physical Address Space

L4: Recursive Address Spaces



L4: Recursive Address Spaces



- Thread has access to a capability → can map this capability to another thread

- Thread has access to a capability → can map this capability to another thread
- Mapping / not mapping of capabilities used to implement access control

- Thread has access to a capability → can map this capability to another thread
- Mapping / not mapping of capabilities used to implement access control
- Abstraction for mapping: *flexpage*
  - Location and size of resource
  - Receiver's rights (read-only, mappable)
  - Type (memory, I/O, communication capability)

L4/Fiasco.OC: Object Types (Recap)

- Summary of object types
  - Task
  - Thread
  - IPC Gate
  - IRQ
  - Factory

- Each task gets initial set of capabilities for some of these objects at startup

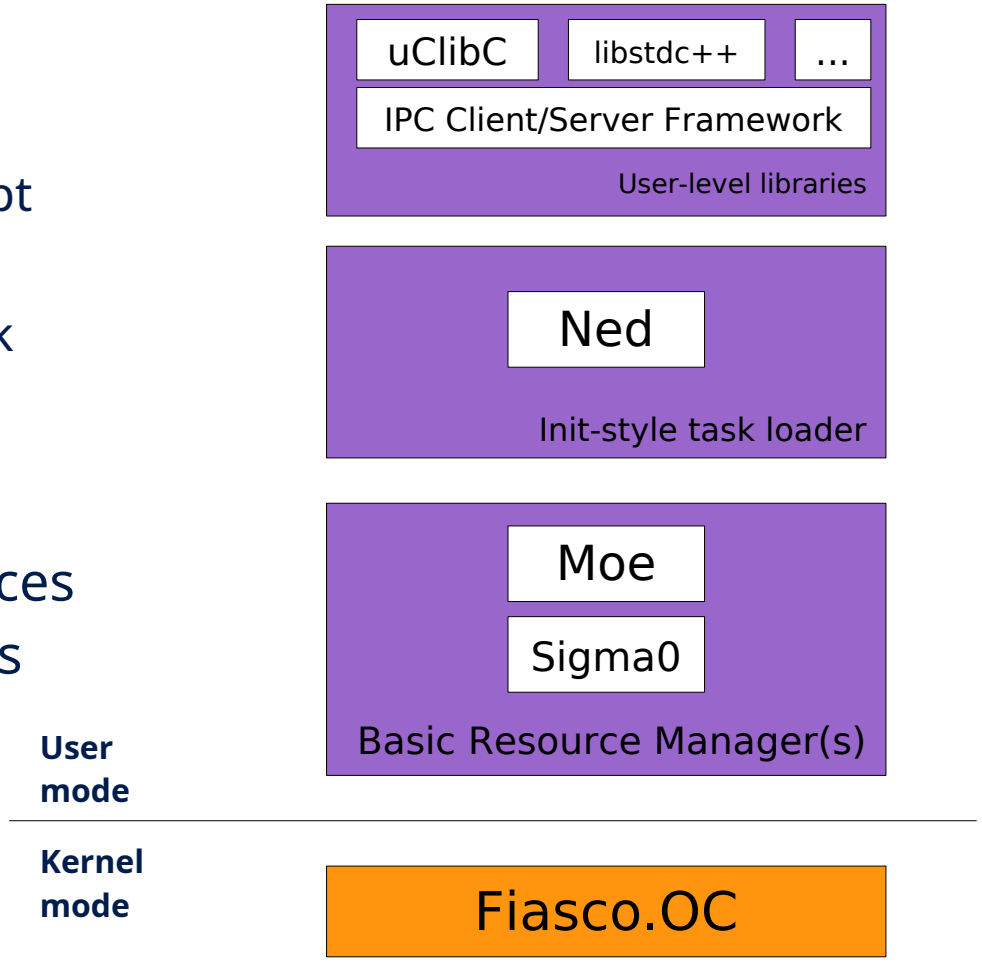
What can we build with this?

Kernel vs. Operating System

- Fiasco.OC is not a full operating system!
  - No device drivers (except UART + timer)
  - No file system / network stack / …

Kernel vs. Operating System

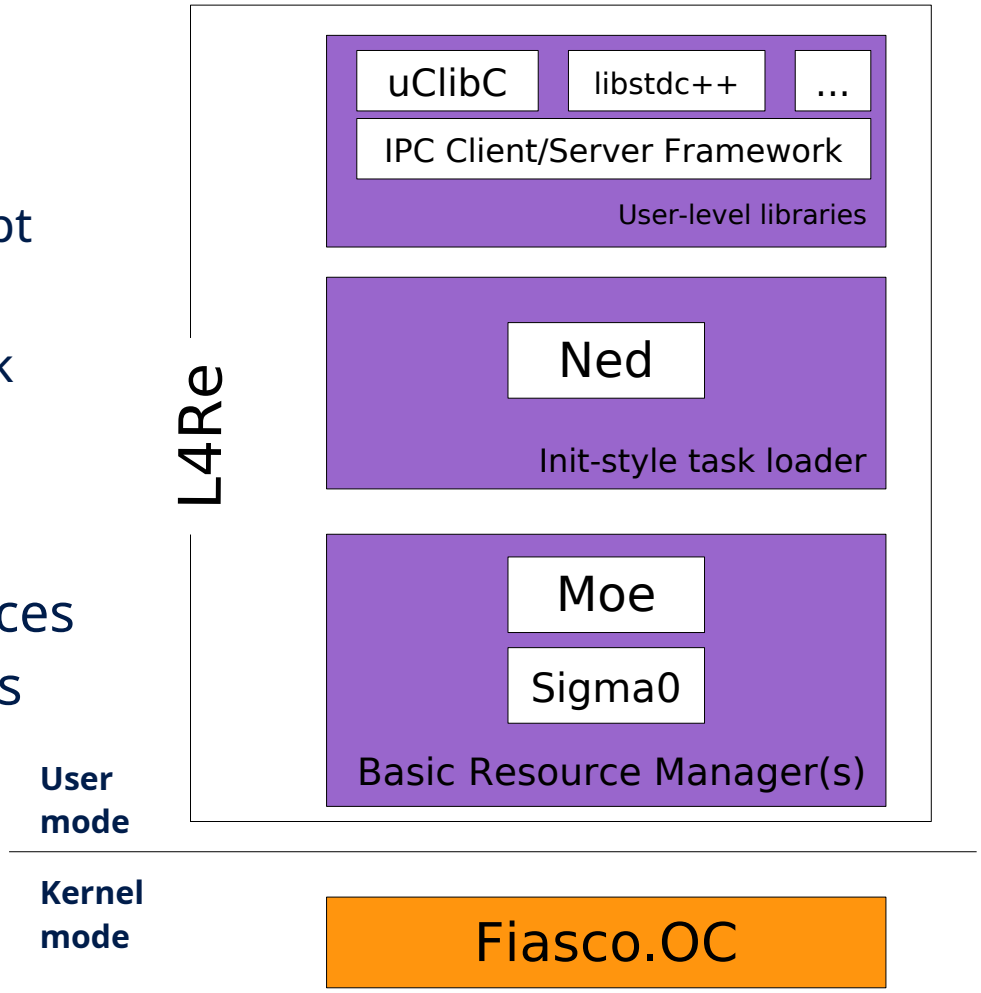
- Fiasco.OC is not a full operating system!
  - No device drivers (except UART + timer)
  - No file system / network stack / ...
- A microkernel-based OS needs to add these services as user-level components



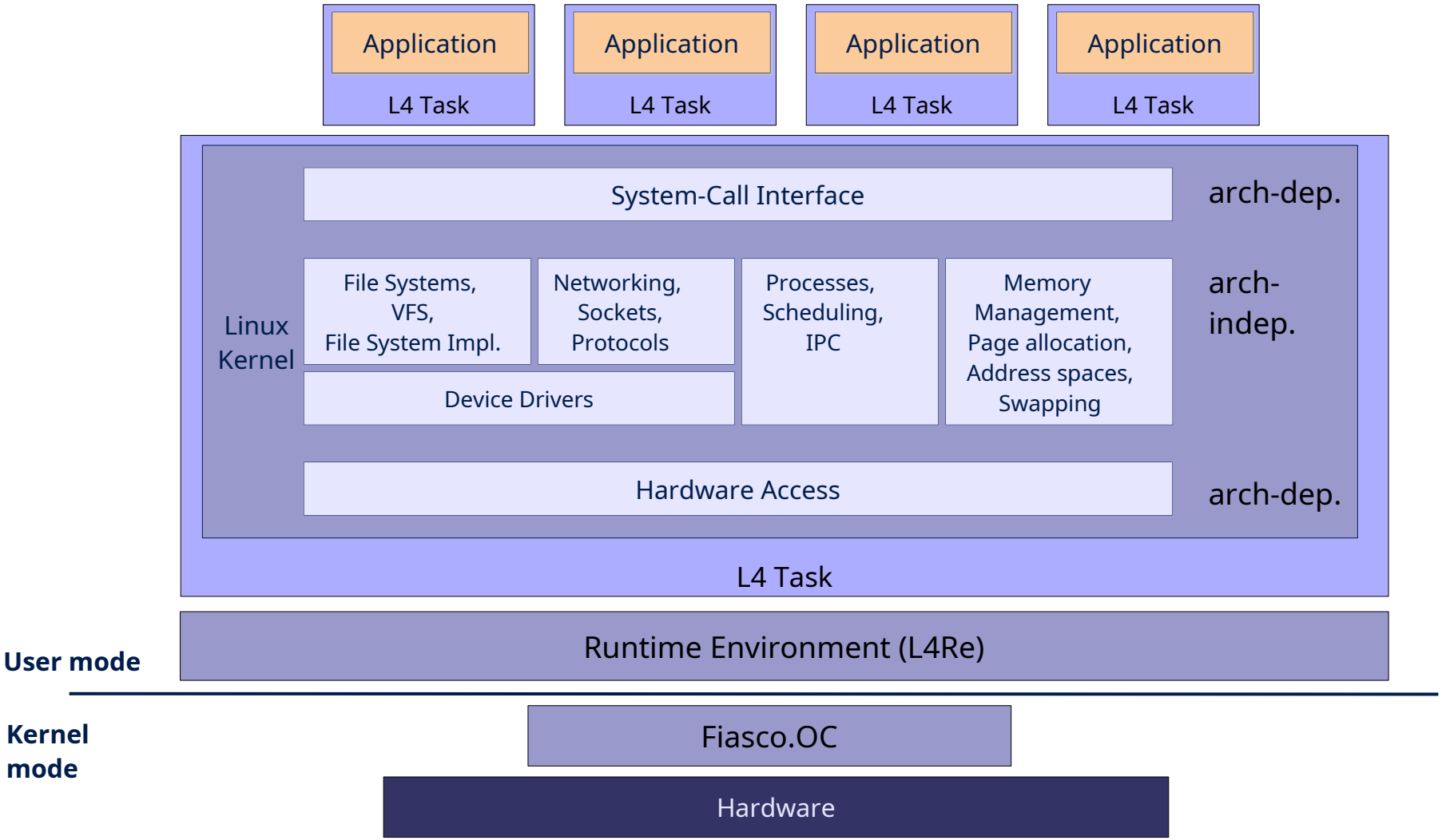
Kernel vs. Operating System

- Fiasco.OC is not a full operating system!
  - No device drivers (except UART + timer)
  - No file system / network stack / ...
- A microkernel-based OS needs to add these services as user-level components

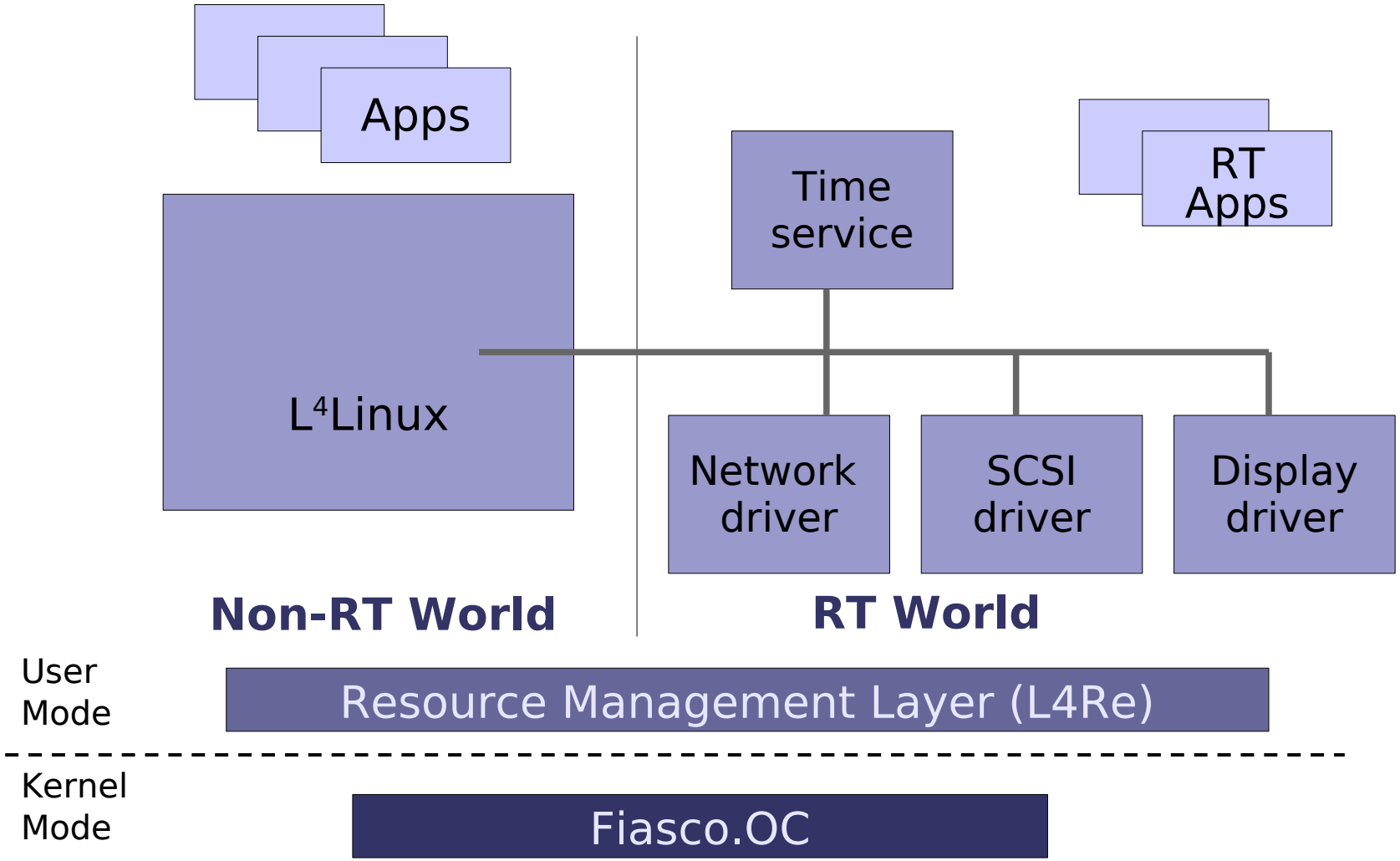
→ L4Re = L4 Runtime Environment



L4Linux: Linux on L4

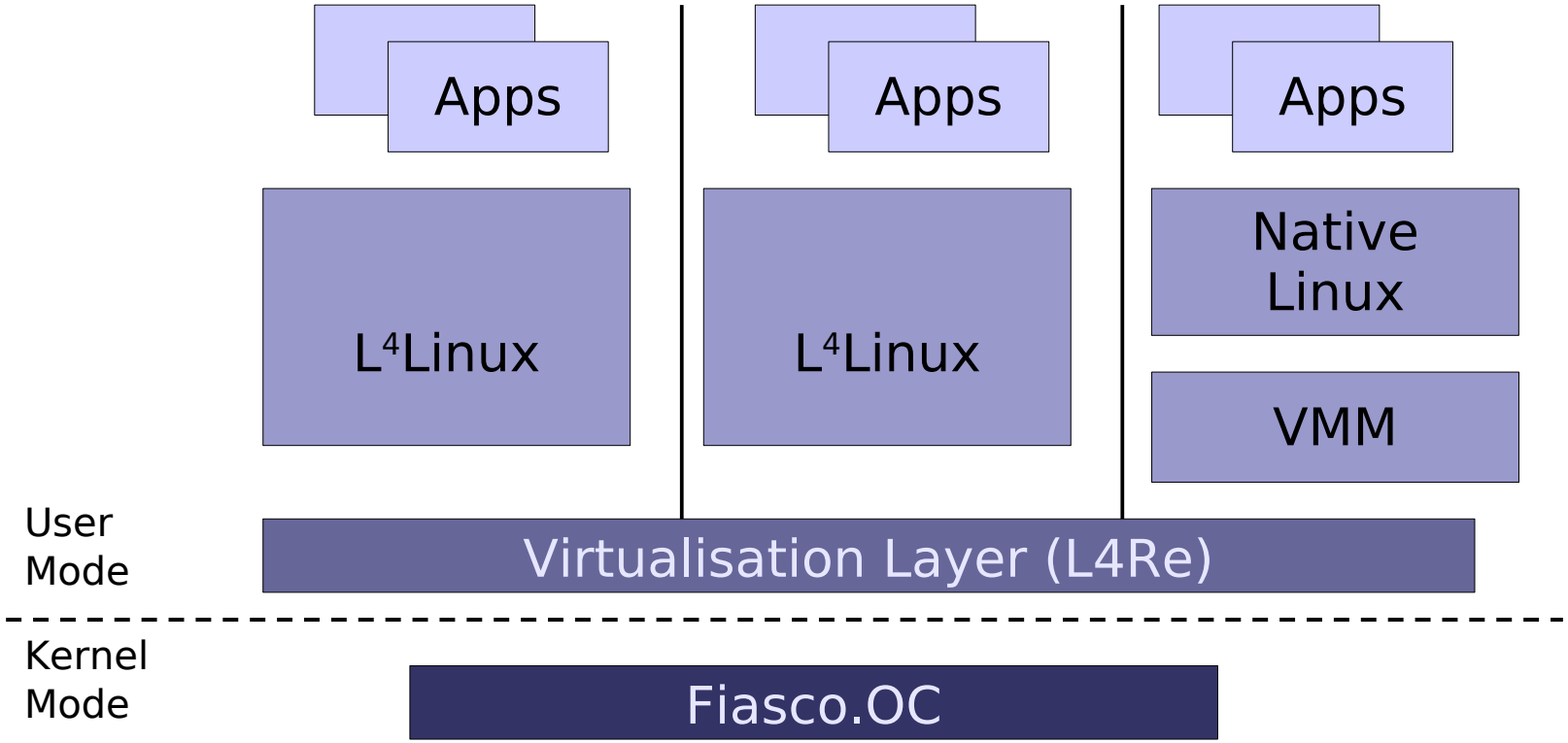


Drops: **D**resden **R**eal-Time **Op**erating **S**ystem



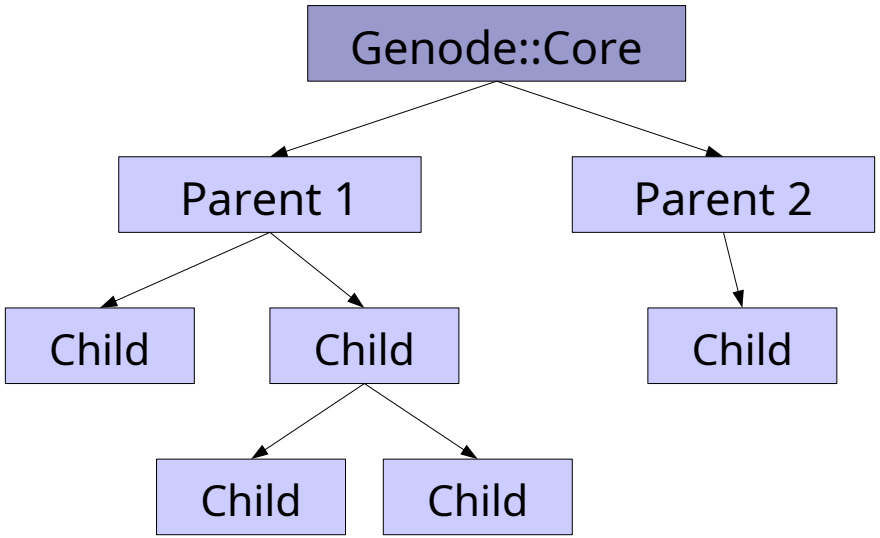
Virtual Machines

- Isolate not only processes, but also complete operating systems (compartments)
- “Server consolidation”



Genode OS Framework

- Genode = C++-based OS framework developed here in Dresden
- Goal: hierarchical system that
  - supports resource partitioning
  - layers security policies on top of each other



What’s to come?

Lecture Outline

- **Basic mechanisms and concepts**
  - Memory management
  - Tasks, Threads, Synchronisation
  - Communication

Lecture Outline

- **Basic mechanisms and concepts**
  - Memory management
  - Tasks, Threads, Synchronisation
  - Communication
- **Building real systems**
  - What are resources and how to manage them?
  - How to build a secure system?
  - How to build a real-time system?
  - How to re-use existing code (Linux, standard system libraries, device drivers, ...)?
  - How to improve robustness and safety?

(Preliminary) Schedule

Date	14:50 @ APB E008	16:40 @ APB E009 / E065
Oct 14	Intro	*Lab: Intro*
Oct 21	IPC	Paper
Oct 28	Threads & Synchronization	—
Nov 4	Memory Management	Practical
Nov 11	Device Drivers	Practical
Nov 18	Real-Time	*Lab*
Nov 25	Resource Management	*Lab*
Dec 2	Virtualisation	Paper
Dec 9	Legacy Reuse	Paper
Dec 16	—	—
Jan 6	Secure Systems	*Lab*
Jan 13	Heterogenous Systems	Practical
Jan 20	—	—
Jan 27	Trusted Computing	Paper
Feb 3	Dependable Systems	—