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 18th 2022

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

- Schedule
  - Lecture: every Tuesday, 16:40, APB/E001
  - Exercises: (roughly) bi-weekly, Tuesday, 14:50
  - (mostly) hybrid via BBB
  - Switch slots of lecture and exercise?
- Slides: https://tudos.org/ → Studies → Lectures → MOS
- Subscribe to our mailing list: https://os.inf.tu-dresden.de/mailman/listinfo/mos2022
- TUD-Matrix?

Exercises

- Practical exercises in the computer lab
- Paper reading exercises in APB/E008
  - Read a paper **beforehand**
  - Sum it up and prepare 3 questions
  - Actively participate in the discussion
- Exercises may also take place online
- Announced on website and mailing list

More Hands-On: Complex lab

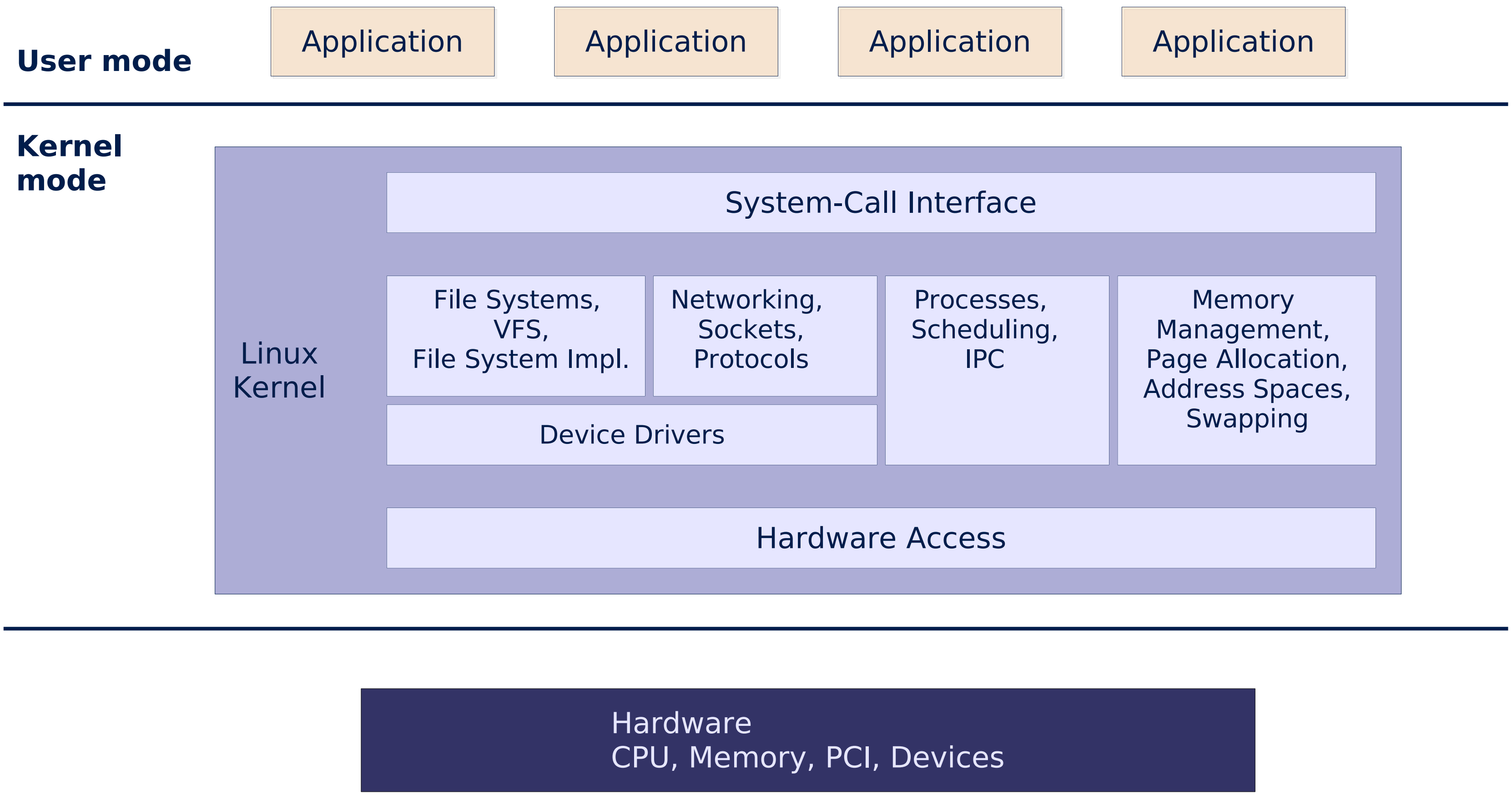
- Complex lab “Microkernel-Based Operating Systems” in parallel to this lecture
- Build several components of an MOS
- This term online
- Coordination via dedicated mailing list, check the complex lab website

Monoliths vs. Microkernels

Purpose of an Operating System

- Manage the available resources
  - Hardware (CPU, memory, ...)
  - 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 / HW-specific operations
  - x86: ring 0 vs. ring 3
  - Device drivers
- Provide separation and collaboration
  - Isolate users / processes from each other
  - Allow cooperation if needed (e.g., sending messages between processes)

Monolithic Kernels: Linux



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?

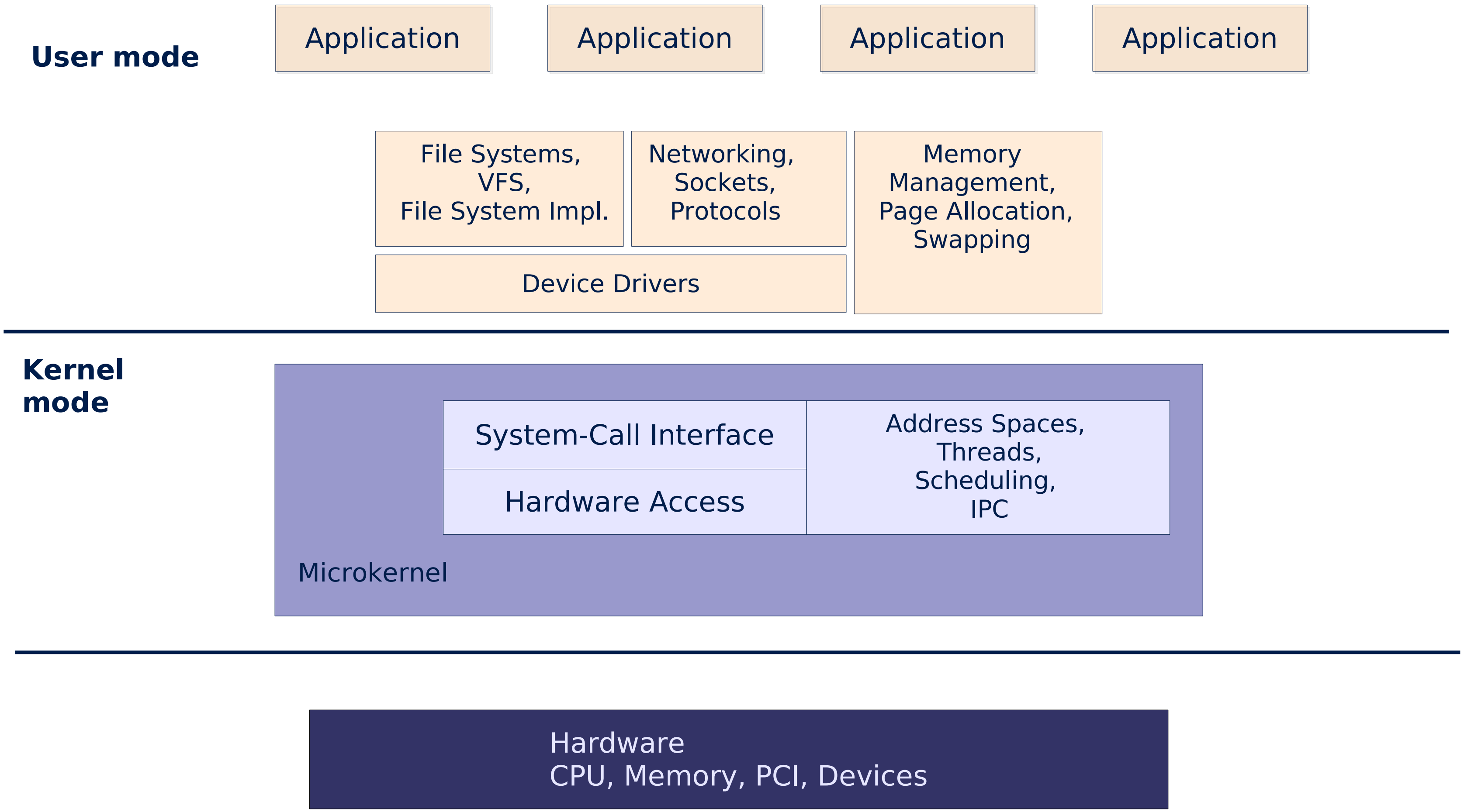
The Microkernel Vision

- Minimal OS kernel
  - less error prone
  - 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



What Microkernels Can Give Us...

- OS personalities
- Customizability
  - 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

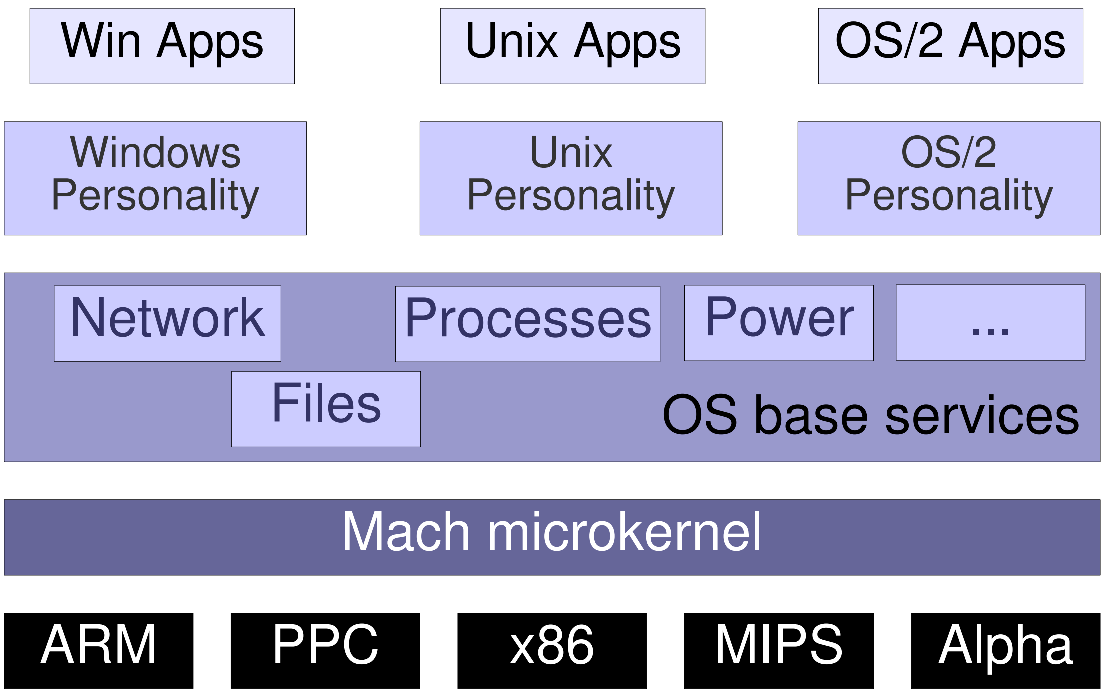
- Developed at CMU, 1985 – 1994
  - Rick Rashid (former head of MS Research)
  - Avie Tevanian (former Apple CTO)
  - Brian Bershad (professor @ U. of Washington)
  - ...
- Foundation for several real systems
  - Single Server Unix (BSD4.3 on Mach)
  - MkLinux (OSF)
  - IBM Workplace OS
  - NeXT OS →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

Mach: Features

- 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 1 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
- Flexibility … but monolithic kernels became flexible, too (Linux kernel modules)
- Better design … but monolithic kernels also improved (restricted symbol access, layered architectures)
- Maintainability … still very complex
- Performance matters a lot

TECHNISCHE UNIVERSITÄT DRESDEN

Microkernels: Proven Advantages

- Subsystem protection / isolation
- Code size (generated using David A. Wheeler's 'SLOCCount')
  - Microkernel-baed OS
    - Fiasco kernel: ~ 34,000 LoC
    - “HelloWorld” (+boot loader +root task): ~ 10,000 LoC
  - Linux kernel (3.0.4., x86 architecture):
    - Kernel: ~ 2.5 million LoC
    - +drivers: ~ 5.4 million LoC
- Customizability
  - Tailored memory management / scheduling / … algorithms
  - Adaptable to embedded / real-time / secure / … systems

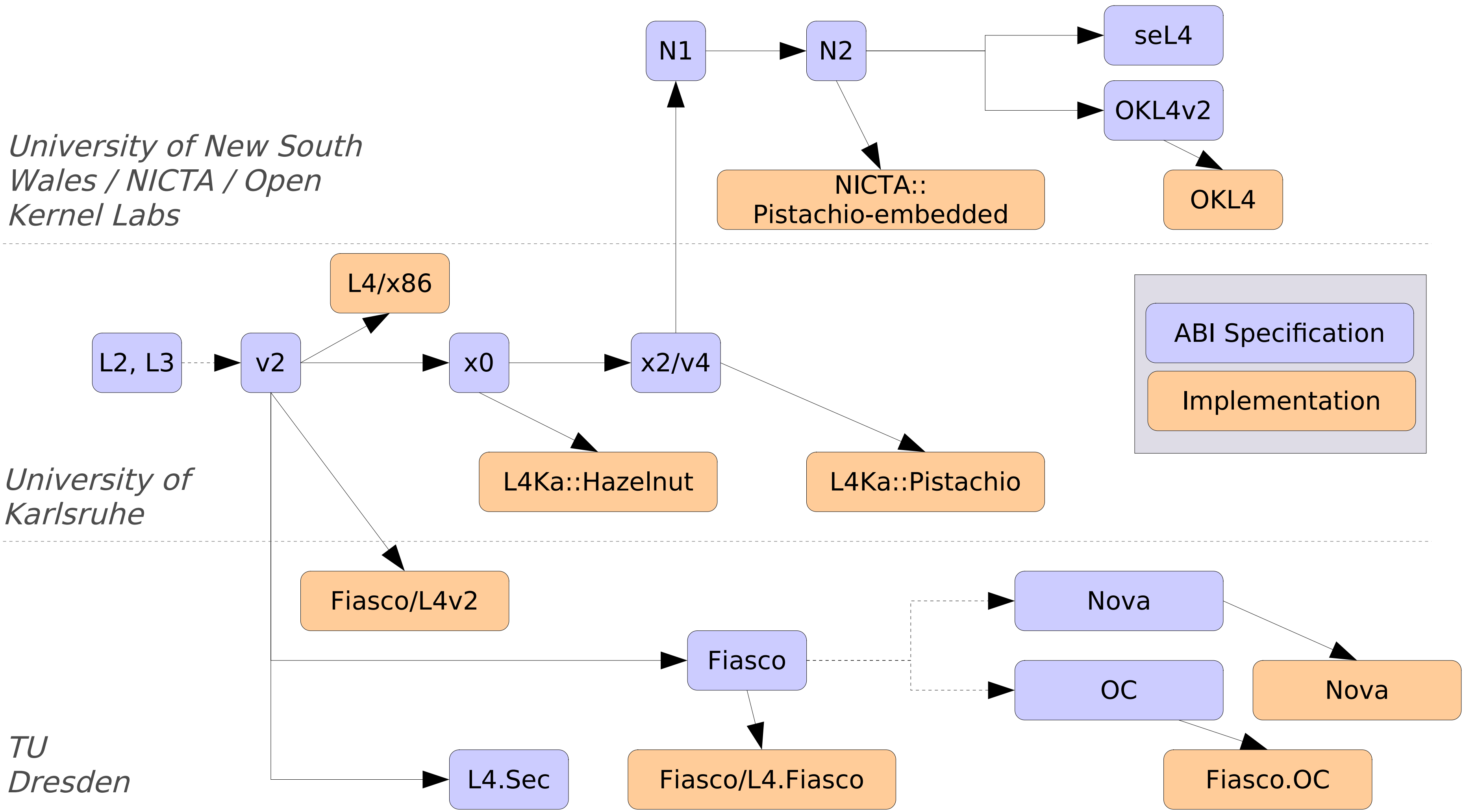
Challenges

- We need fast and efficient kernels
  - Covered in the “Microkernel construction” lecture in the summer term
- We need fast and efficient OS services
  - Memory and resource management
  - Synchronization
  - 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.”
  - Kernel only provides inevitable mechanisms.
  - 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**

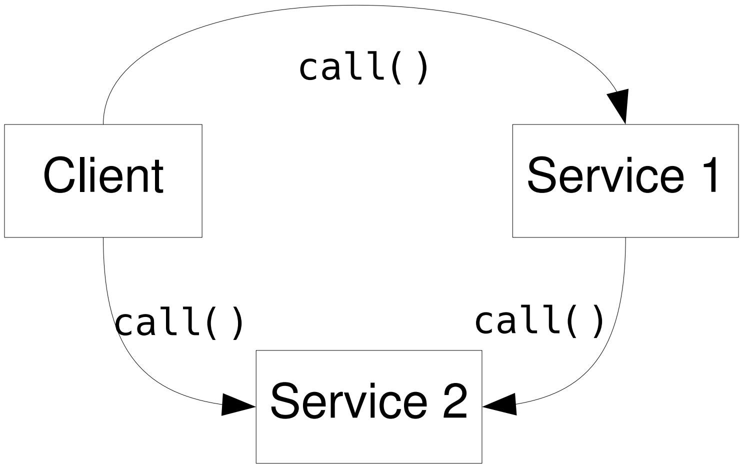
- “Everything is an object”
  - Task — Address spaces
  - Thread — Activities, scheduling
  - IPC Gate — Communication, resource mapping
  - IRQ — Communication
  - Factory — Create other objects, enforce resource quotas
- One system call: **`invoke_object()`**
  - Parameters passed in UTCB
  - Types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

L4/Fiasco.OC: Kernel Objects

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

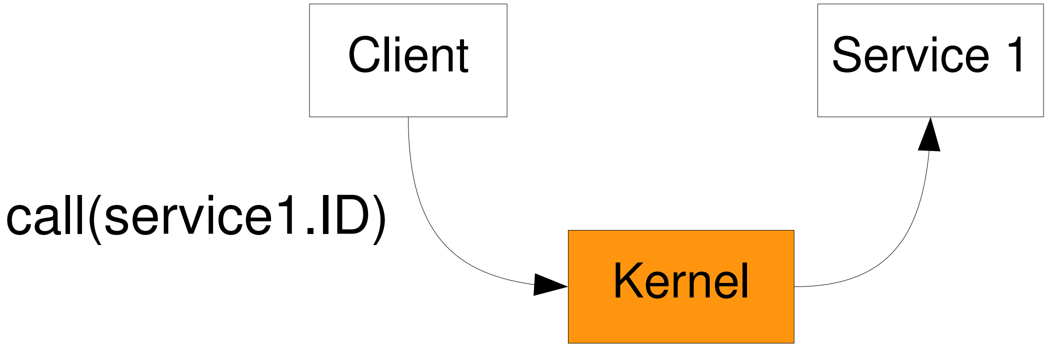
- Everything above kernel built using user-level objects that provide a service
  - Networking stack
  - File system
  - ...



- 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
  - ...



- 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

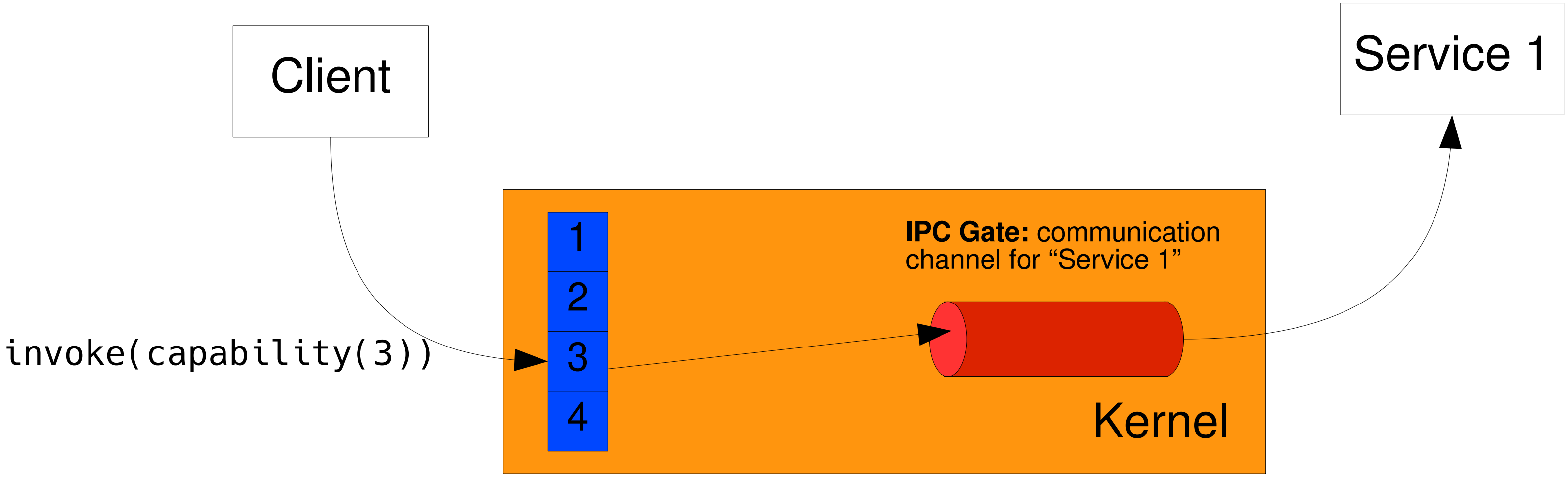
L4/Fiasco.OC: Capabilities

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

- Solution in Fiasco.OC
  - Task-local *capability space* as an indirection
  - *Object capability* <u>required</u> 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 indexes into this table



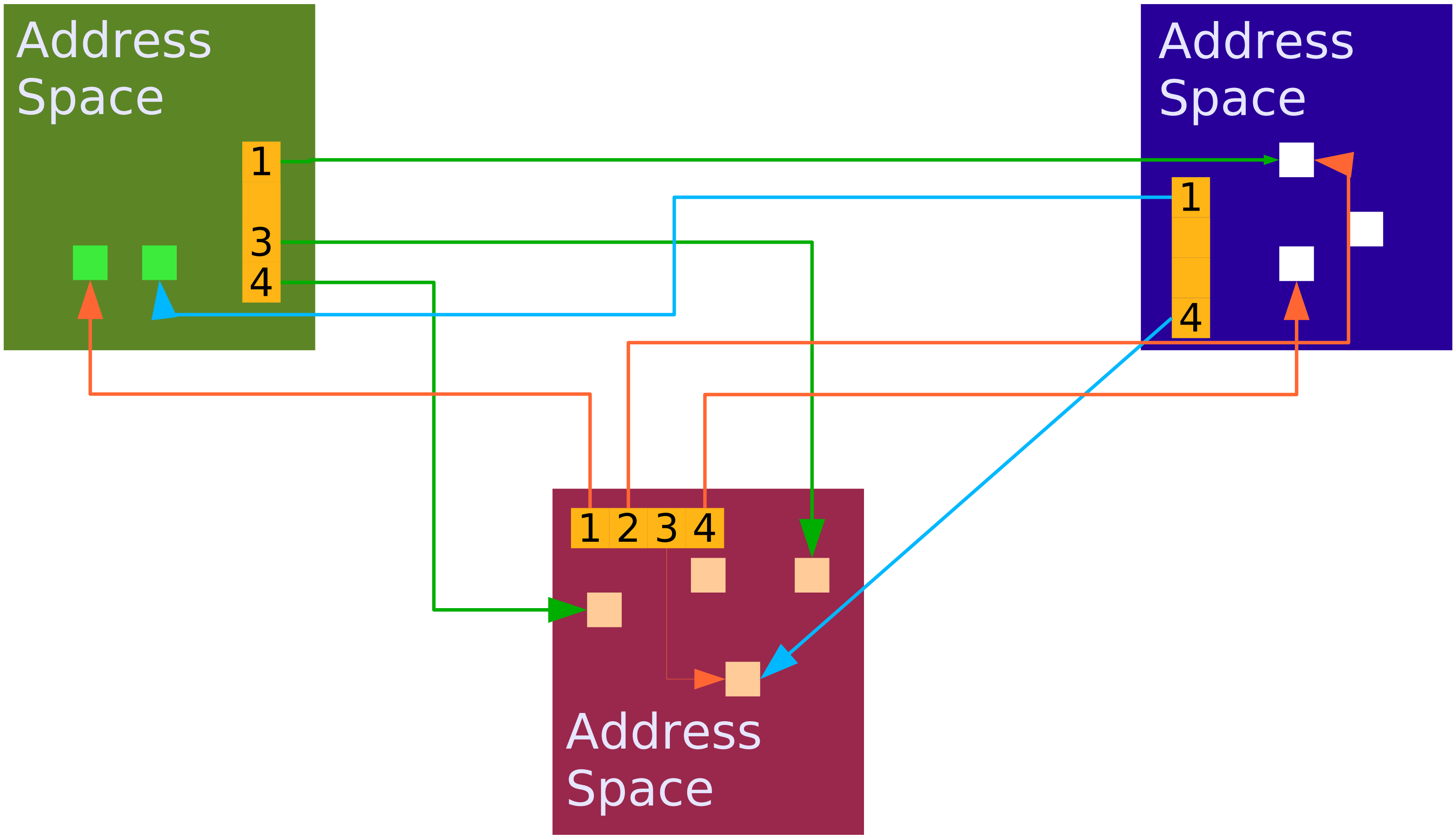
L4/Fiasco.OC: Communication

- 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

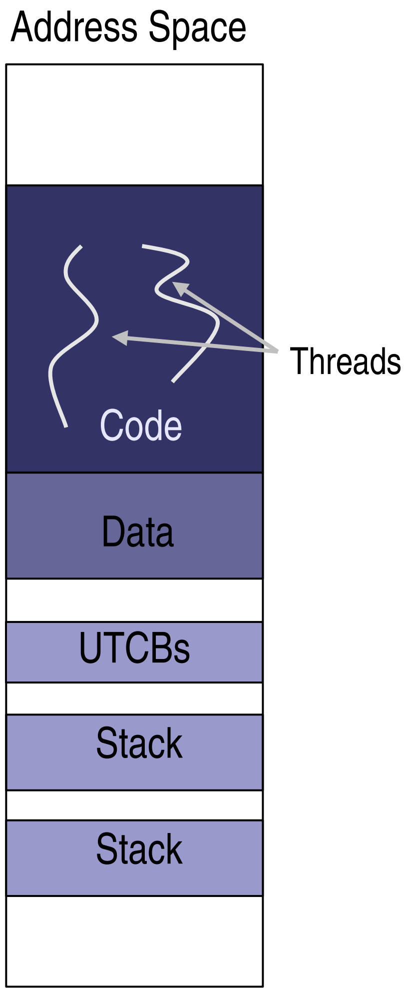
Capabilities == Local Names



More L4 concepts

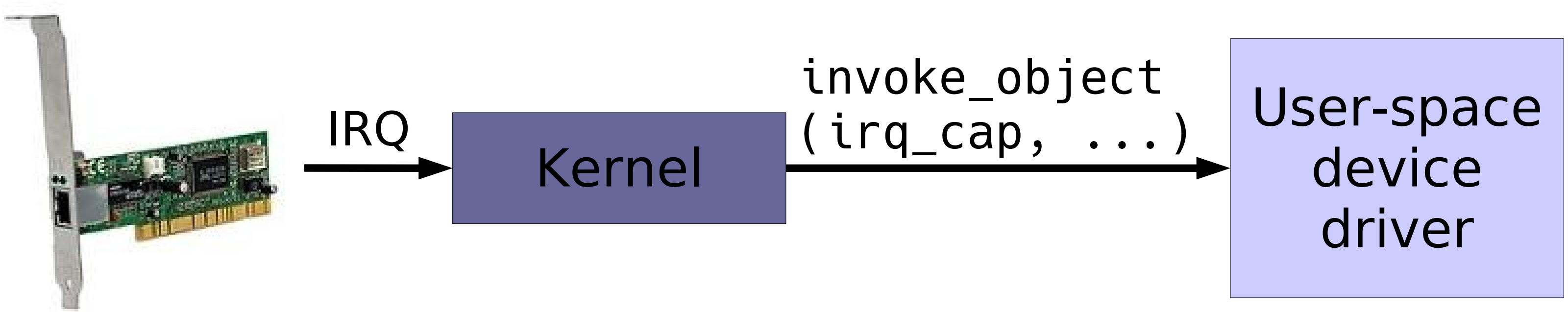
L4/Fiasco.OC: Threads

- 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
  - …

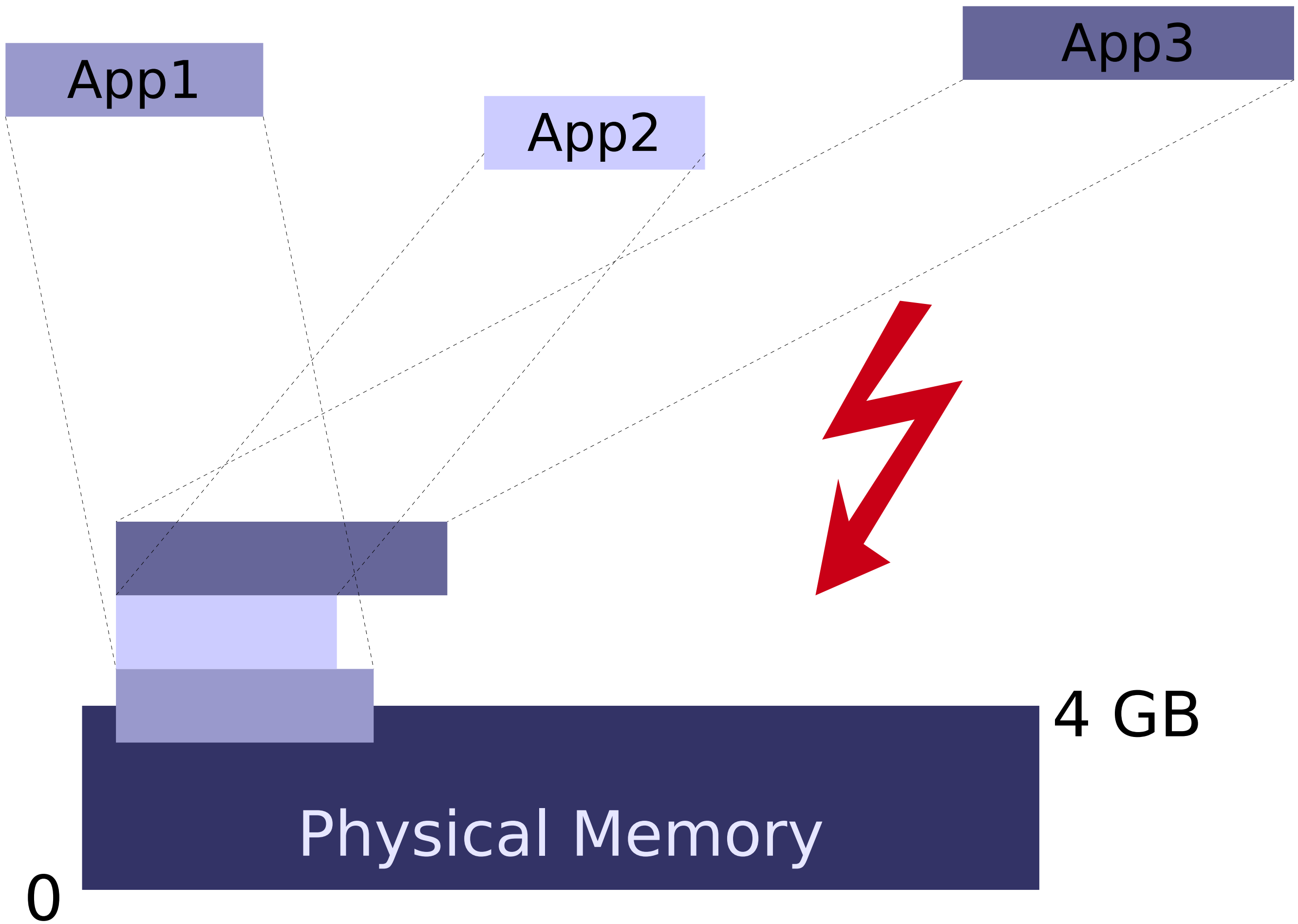


L4/Fiasco.OC: Interrupts

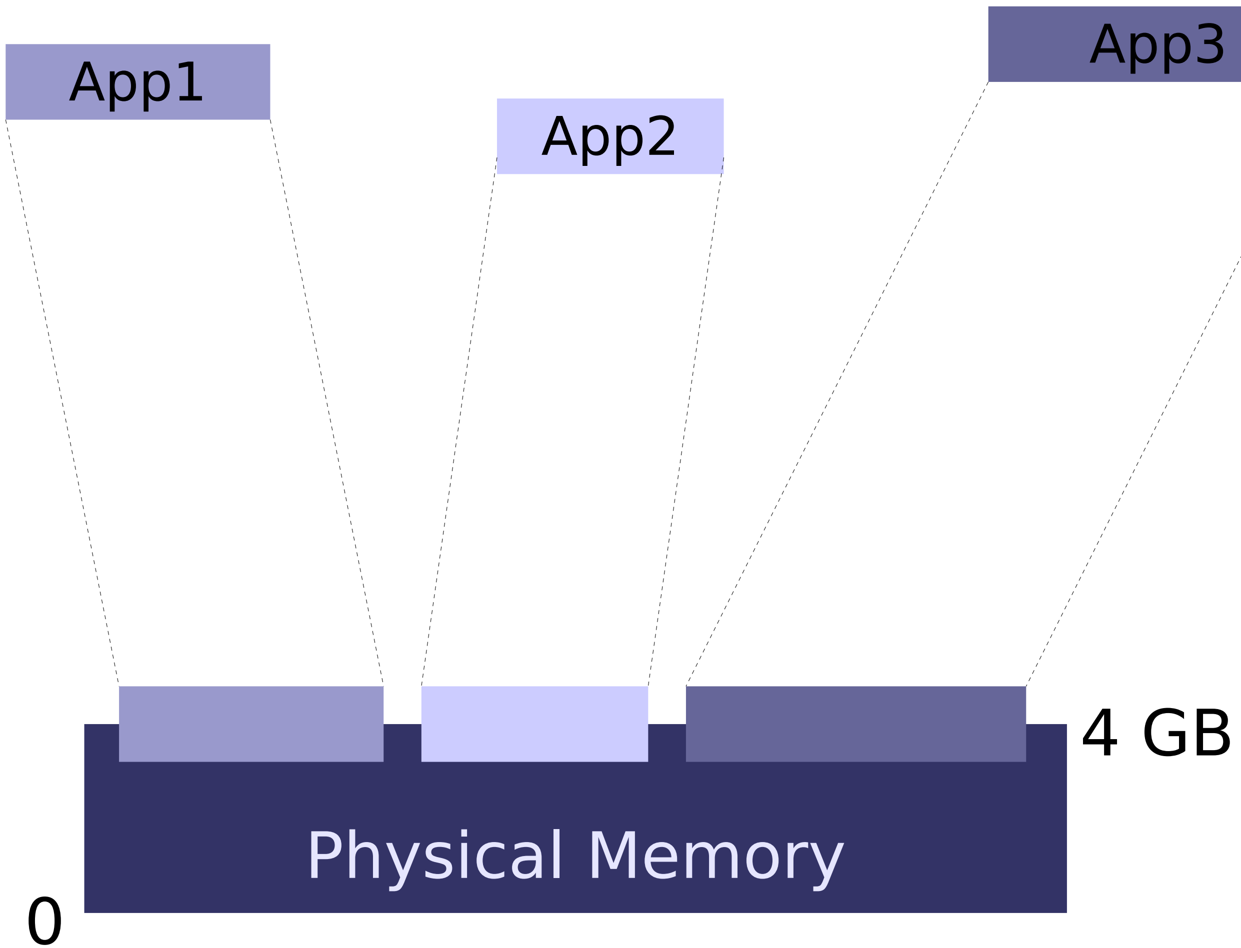
- Kernel object: IRQ
- Used for hardware and software interrupts
- Provides asynchronous signaling
  - `invoke_object(irq_cap, WAIT)`
  - `invoke_object(irq_cap, TRIGGER)`



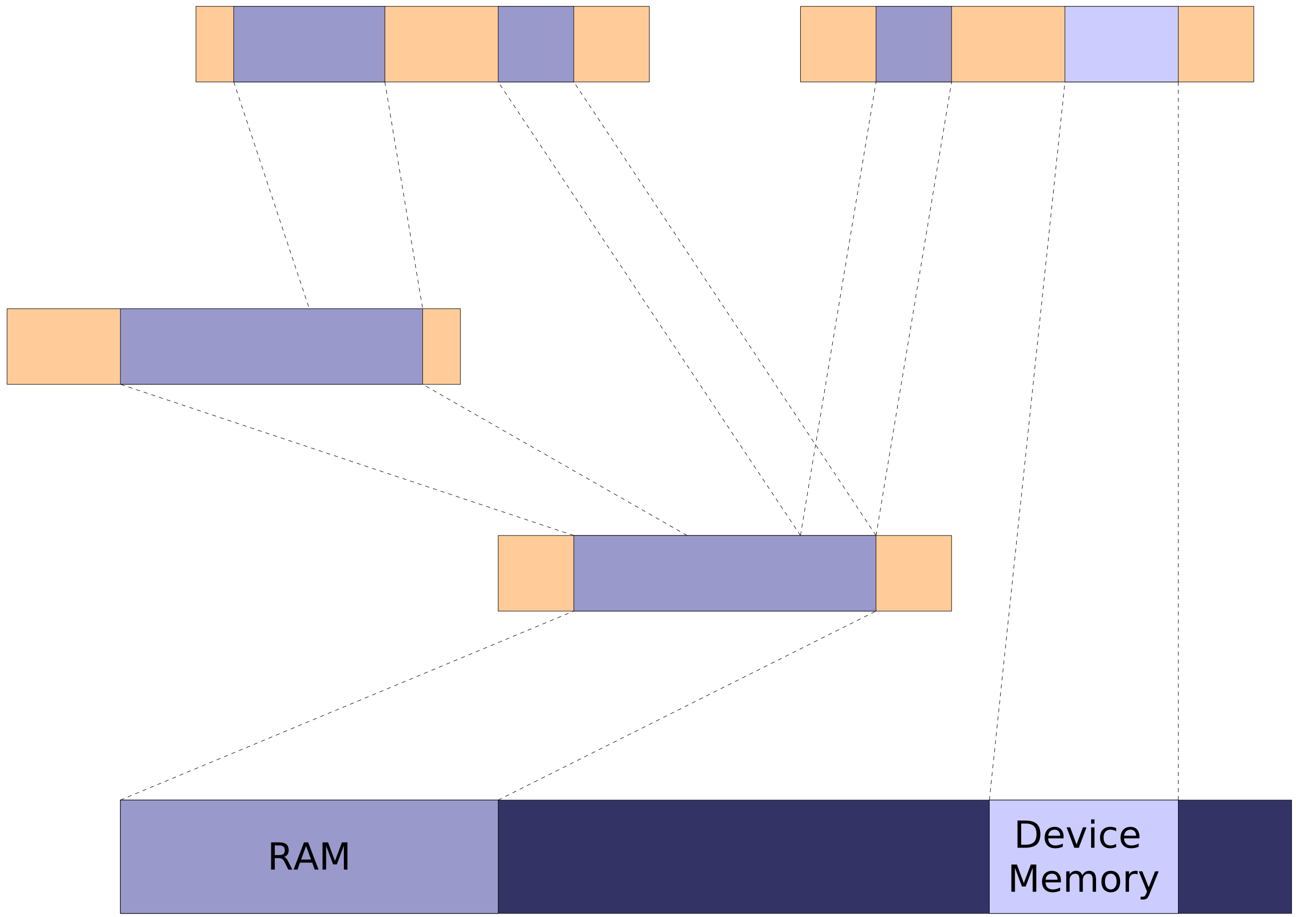
Problem: Memory partitioning



Solution: Virtual Memory



L4: Recursive Address Spaces



Physical Address Space

L4: Resource Mappings

- If a thread has access to a capability, it can map this capability to another thread
- Mapping / not mapping of capabilities used for implementing access control
- Abstraction for mapping: *flexpage*
- Flexpage describes mapping
  - 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

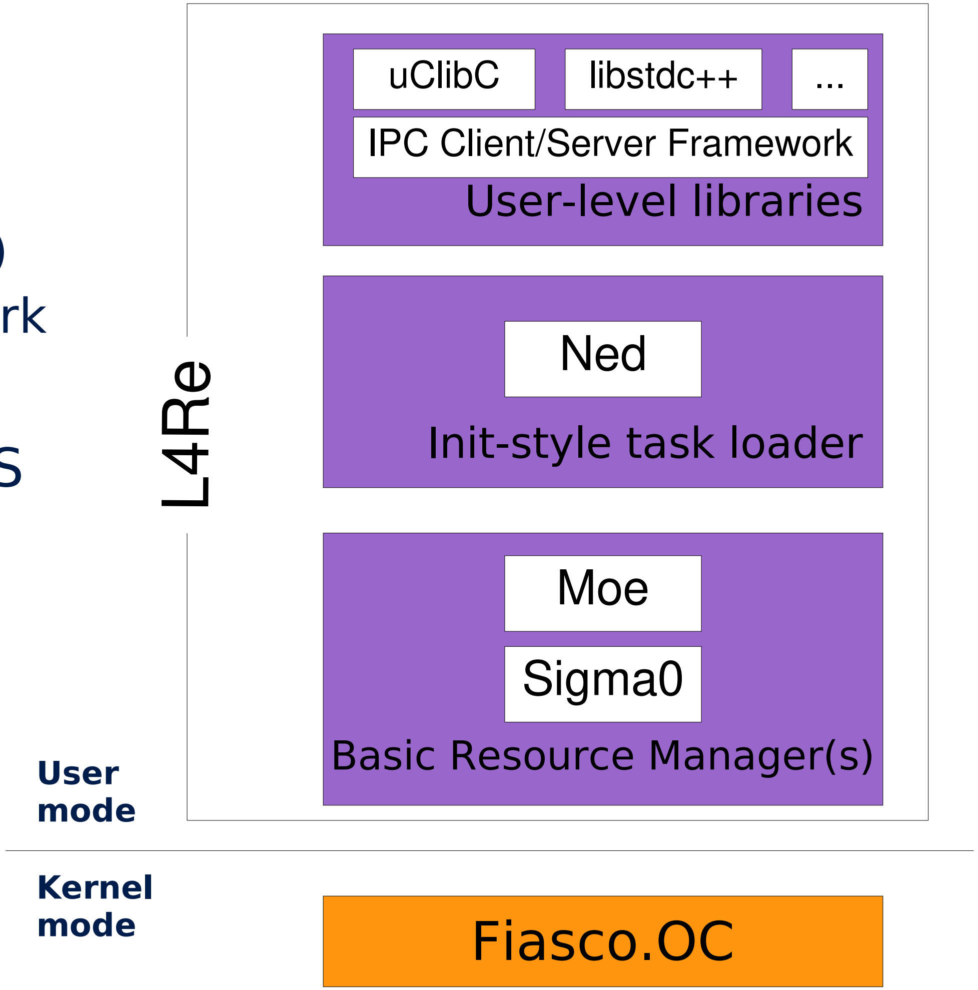
- Each task gets initial set of capabilities for some of these objects at startup

What can we build with this?

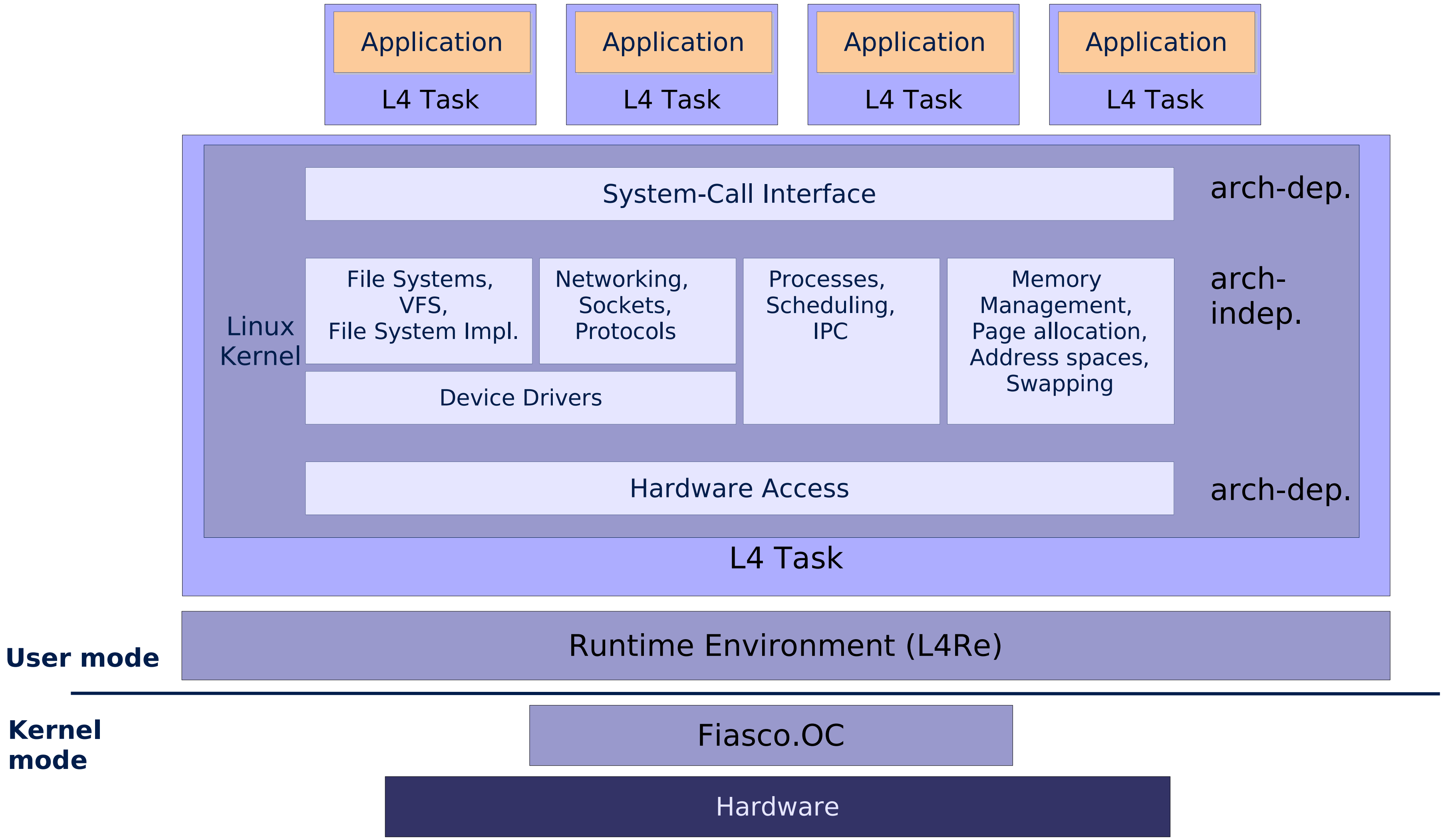
Kernel vs. Operating System

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

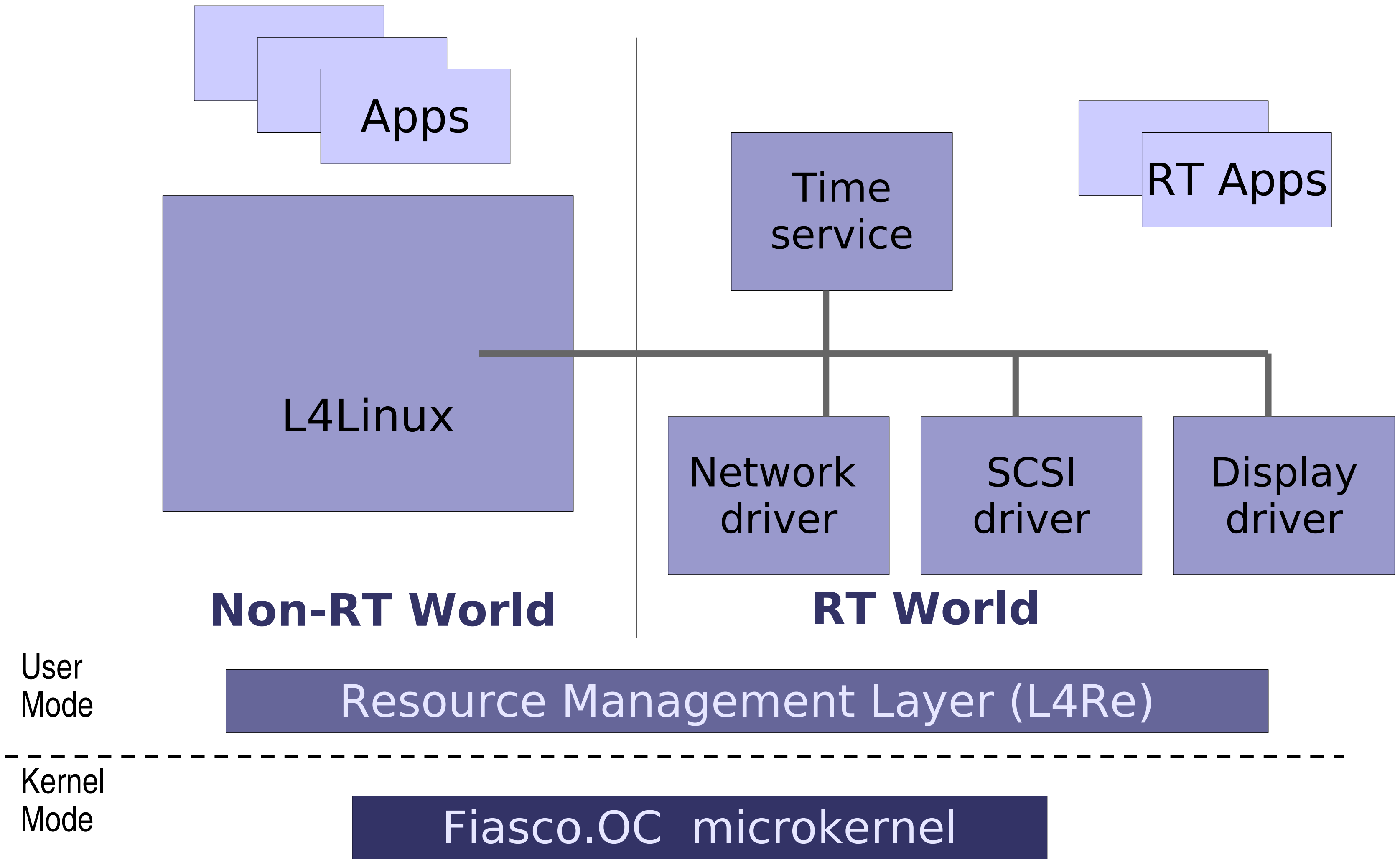
L4Re = L4 Runtime Environment



Linux on L4

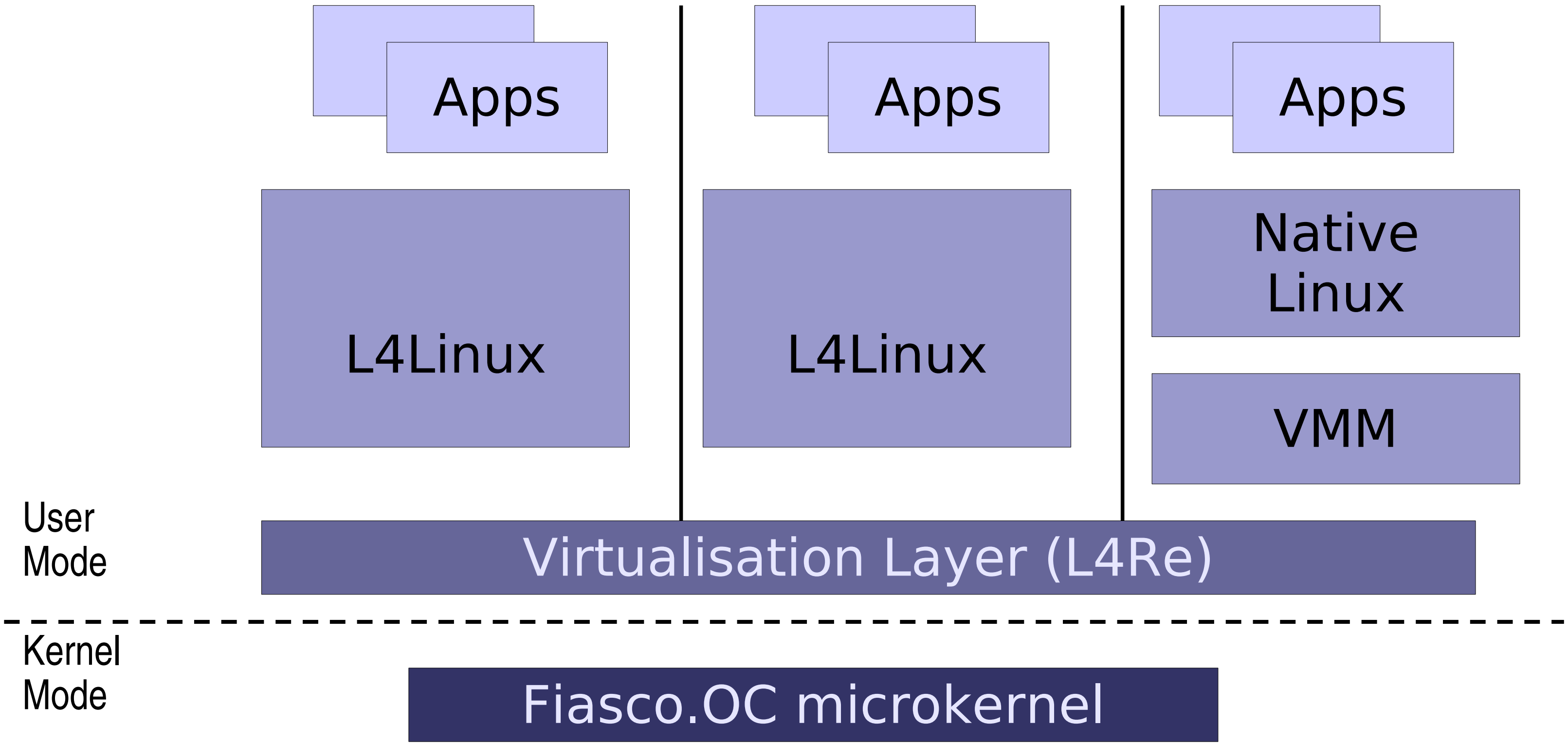

Application
L4 Task
Application
L4 Task
Application
L4 Task
Application
L4 Task
System-Call Interface
arch-dep.
Linux Kernel
File Systems, VFS, File System Impl.
Networking, Sockets, Protocols
Processes, Scheduling, IPC
Memory Management, Page allocation, Address spaces, Swapping
arch-indep.
Device Drivers
Hardware Access
arch-dep.
L4 Task
Runtime Environment (L4Re)
User mode
Kernel mode
Fiasco.OC
Hardware

The Dresden Real-Time Operating System



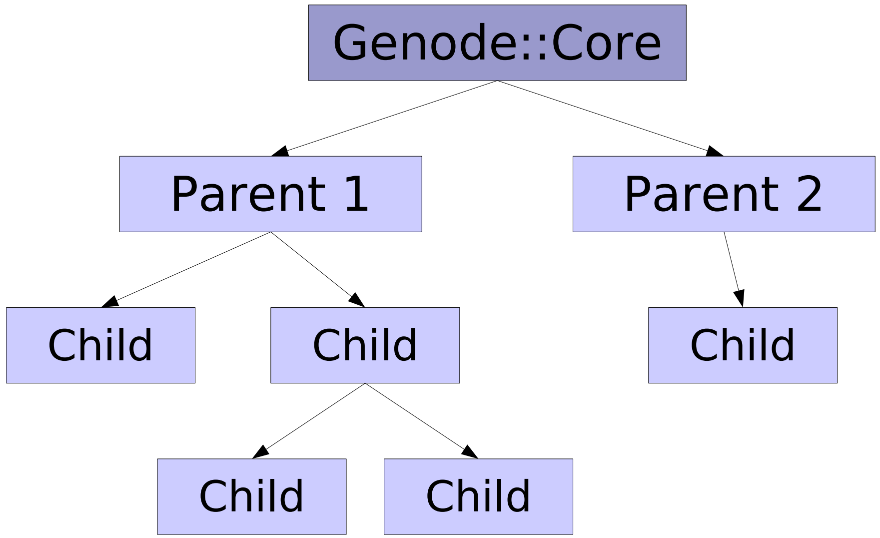
Virtual Machines

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



Genode

- 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
- **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?

TECHNISCHE UNIVERSITÄT DRESDEN

Outlook

- Next lecture:
  - “Threads & Synchronization”
  - Next week (Oct 25, 14:50)

- First exercise:
  - Per Brinch Hansen: *“The nucleus of a multiprogramming system”*
  - Link will be on the website
  - Next week (Oct 25, 16:40)
  - **Read the paper!**