barkhausen institut

Microkernel Construction

Introduction

Nils Asmussen

04/16/2026

Organization

- MKC is a 2+2 SWS lecture
- Questions can be asked live or on the mailing list:
  https://tudos.org/mailman/listinfo/mkc2026
- Slides and video recordings are published:
  https://tudos.org → Studies → Lectures → MKC

Organization

Lectures

- In-person in **APB/E008** at 1 PM on Thursday
- Lectures will be live-streamed and recorded: https://bbb.tu-dresden.de/b/nil-idy-kbw-ocw

Exercises

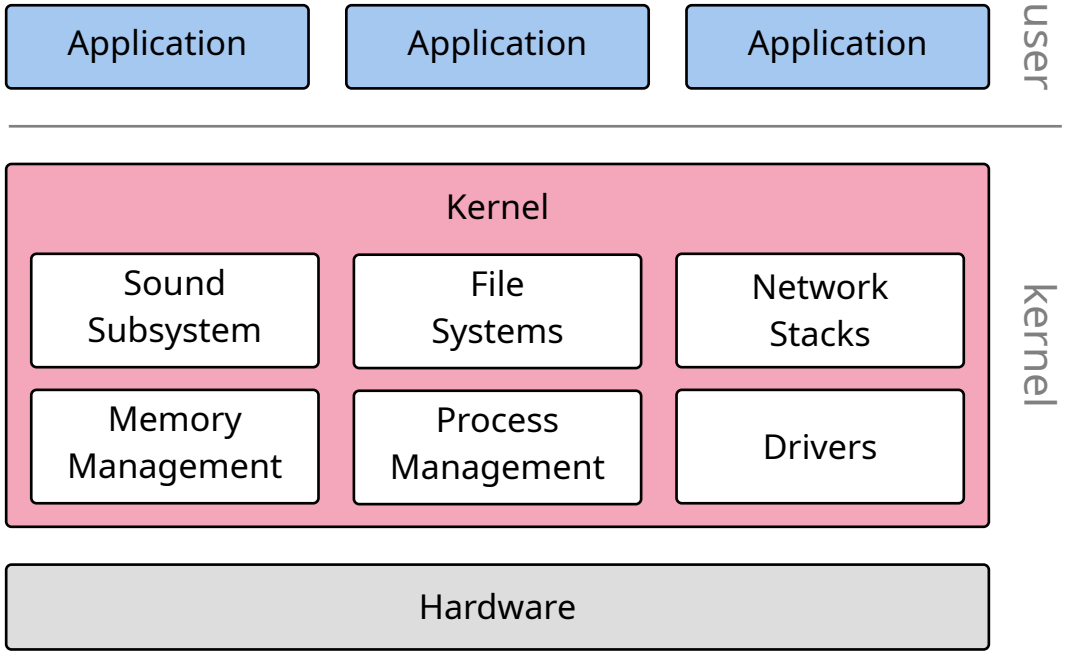
- In-person in **APB/E042** at 2:40 PM on Thursday
- Exercises will **not** be live-streamed / recorded

Goals

1. Provide deeper understanding of OS mechanisms
2. Look at the implementation details of microkernels
3. Make you become enthusiastic microkernel hackers
4. Propaganda for OS research done at TU Dresden and Barkhausen Institut

- Organization
- **Monolithic vs. Microkernel**
  - **Kernel Design Comparison**
  - Examples of Microkernel-based Systems
  - Vision vs. Reality
- Overview About L4/NOVA

Monolithic System Design



Problems of Monolithic Design

1. No protection between system components
   - Faulty driver can crash the whole system
   - Malicious app could exploit bug in faulty driver
   - More than $\frac{2}{3}$ of today's OS code are drivers

Problems of Monolithic Design

1. No protection between system components
   - Faulty driver can crash the whole system
   - Malicious app could exploit bug in faulty driver
   - More than $\frac{2}{3}$ of today's OS code are drivers
2. No need for good system design
   - Direct access to data structures
   - Undocumented and frequently changing interfaces

Problems of Monolithic Design

1. No protection between system components
   - Faulty driver can crash the whole system
   - Malicious app could exploit bug in faulty driver
   - More than $\frac{2}{3}$ of today's OS code are drivers
2. No need for good system design
   - Direct access to data structures
   - Undocumented and frequently changing interfaces
3. Big and inflexible
   - Difficult to replace system components
   - Difficult to understand and maintain

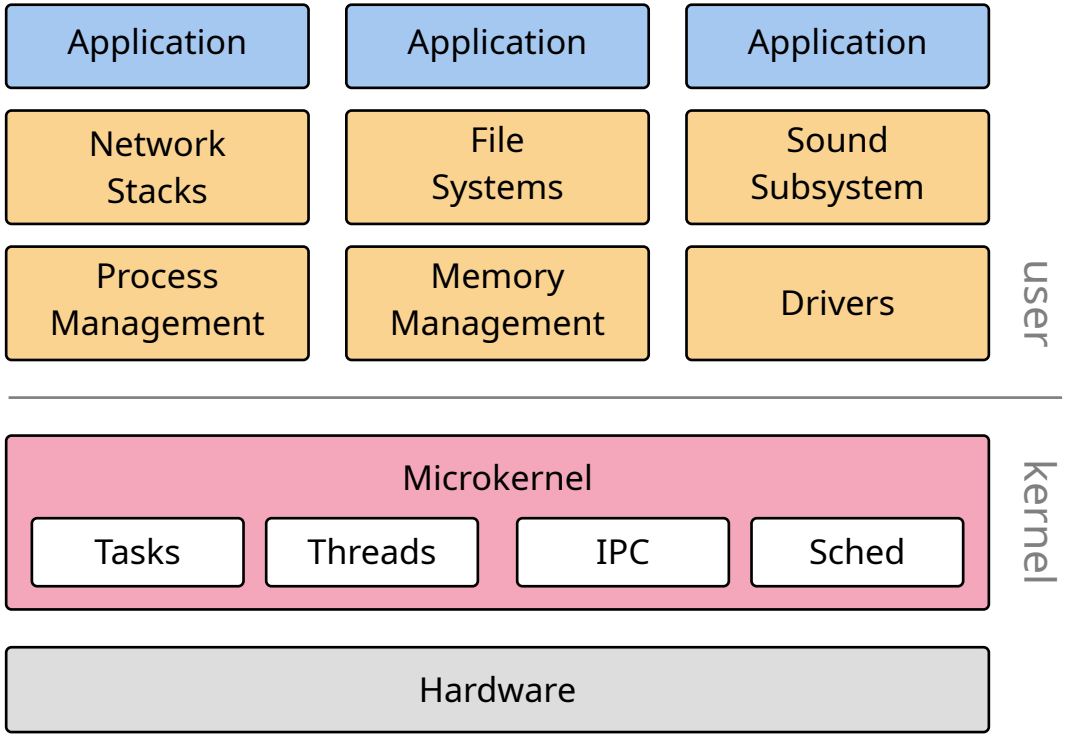
Problems of Monolithic Design

1. No protection between system components
   - Faulty driver can crash the whole system
   - Malicious app could exploit bug in faulty driver
   - More than $\frac{2}{3}$ of today's OS code are drivers
2. No need for good system design
   - Direct access to data structures
   - Undocumented and frequently changing interfaces
3. Big and inflexible
   - Difficult to replace system components
   - Difficult to understand and maintain

Why Something Different?

Increasingly difficult to manage growing OS complexity

Microkernel System Design



Microkernels: Promising Alternative?

- study on critical Linux CVEs between 1999 and 2017

Simon Biggs, Damon Lee, and Gernot Heiser. **The jury is in: Monolithic OS design is flawed.**, APSys 2018

Microkernels: Promising Alternative?

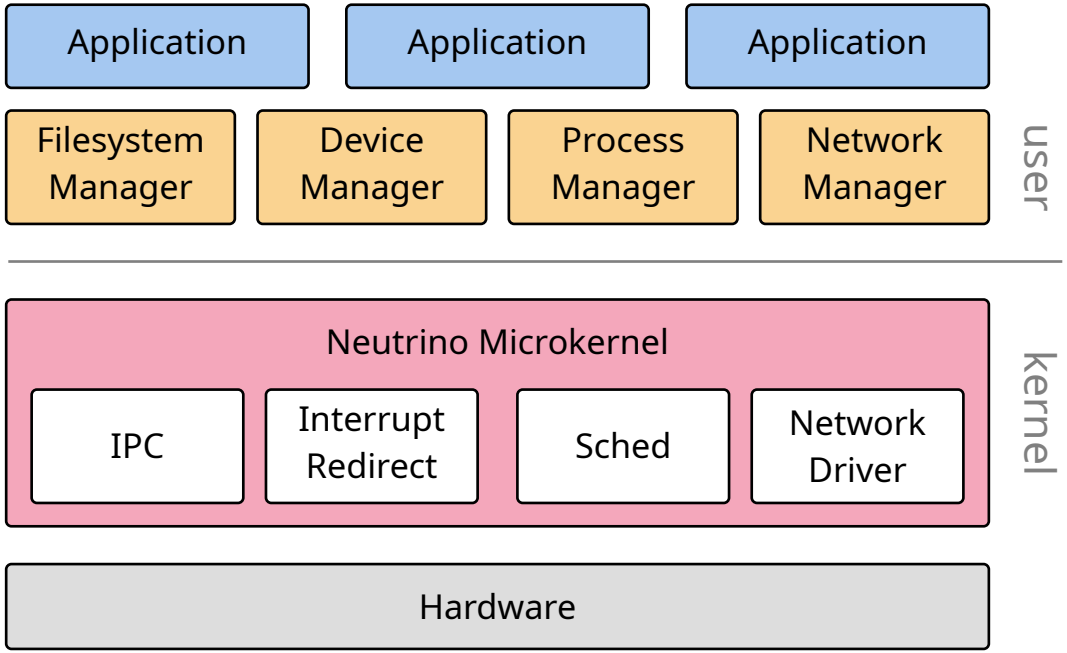
- study on critical Linux CVEs between 1999 and 2017
- 96% of CVEs would not reach critical severity on microkernel-based OSes

Assessment	Count	Fraction	Cumul.
Eliminated	33	29%	29%
Eliminated with verification	12	11%	40%
Strongly mitigated	19	17%	57%
Weakly mitigated	43	38%	96%
Unaffected	5	4%	100%
Total:	**112**	**100%**	**100%**

Simon Biggs, Damon Lee, and Gernot Heiser. **The jury is in: Monolithic OS design is flawed.**, APSys 2018

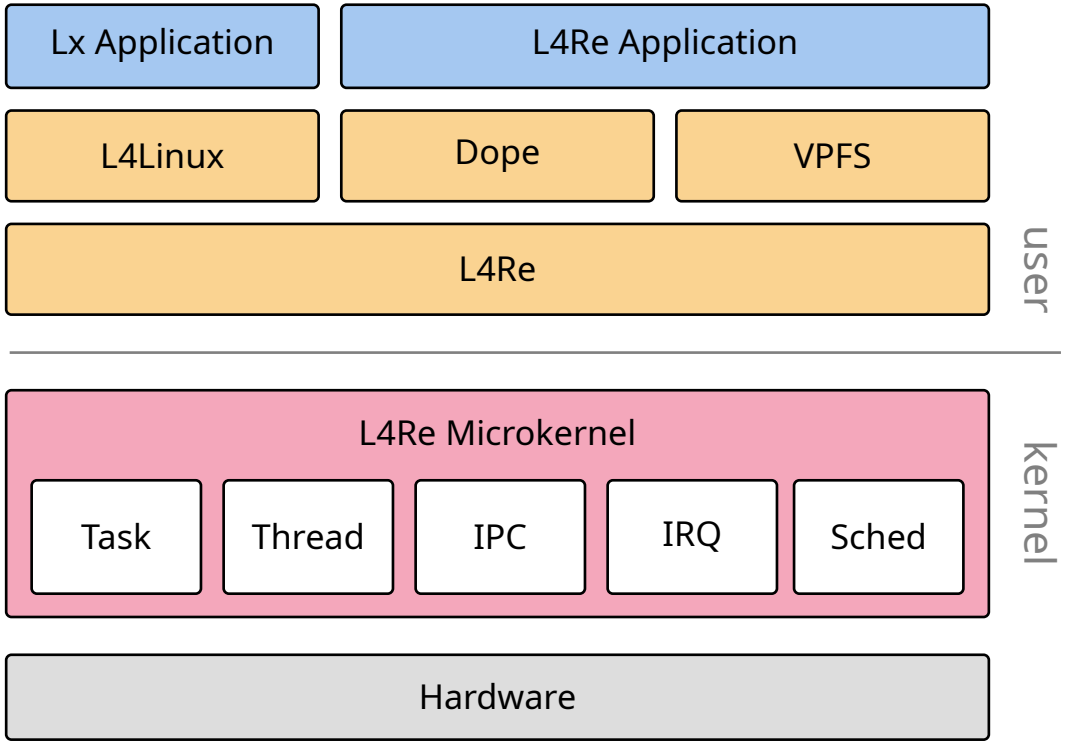
- Organization
- **Monolithic vs. Microkernel**
  - Kernel Design Comparison
  - **Examples of Microkernel-based Systems**
  - Vision vs. Reality
- Overview About L4/NOVA

Example: QNX on Neutrino



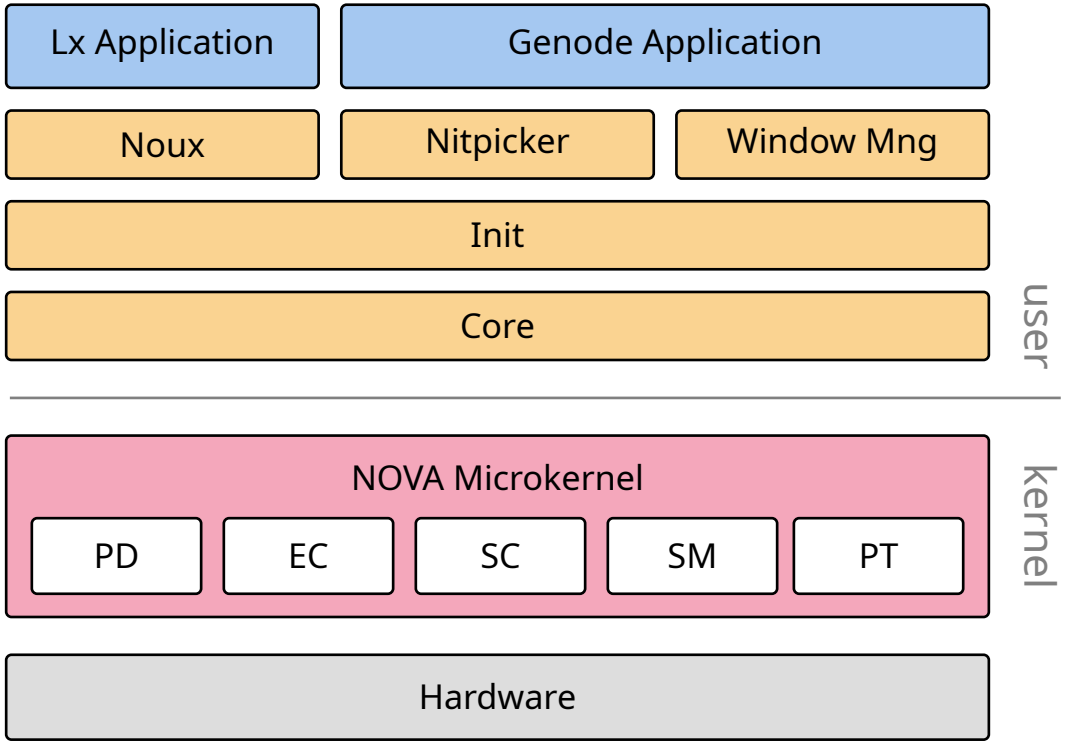
- Commercial
- Embedded systems
- Network transparency

Example: L4Re



- Developed at OS chair
- Now at Kernkonzept
- Part of L4 family

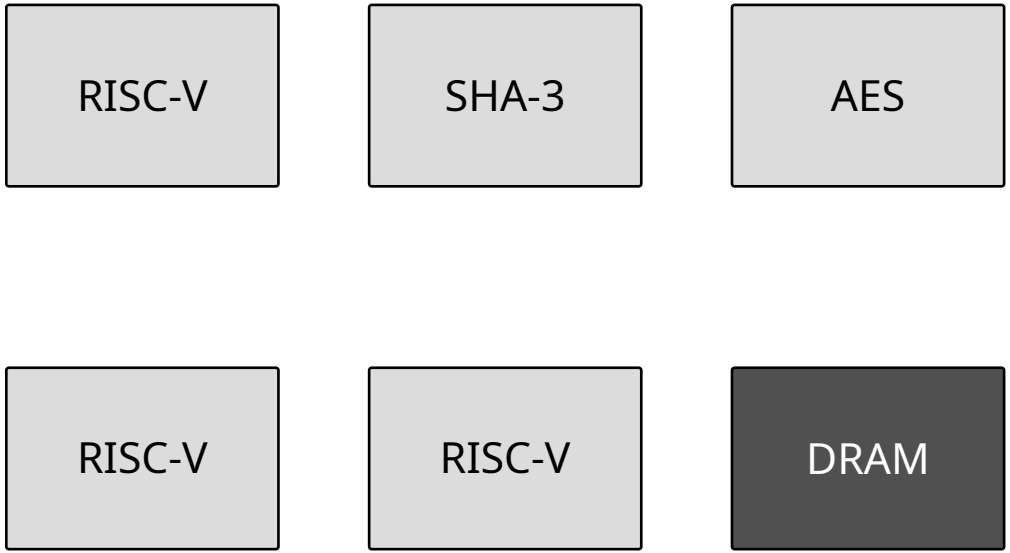
Example: Genode on NOVA



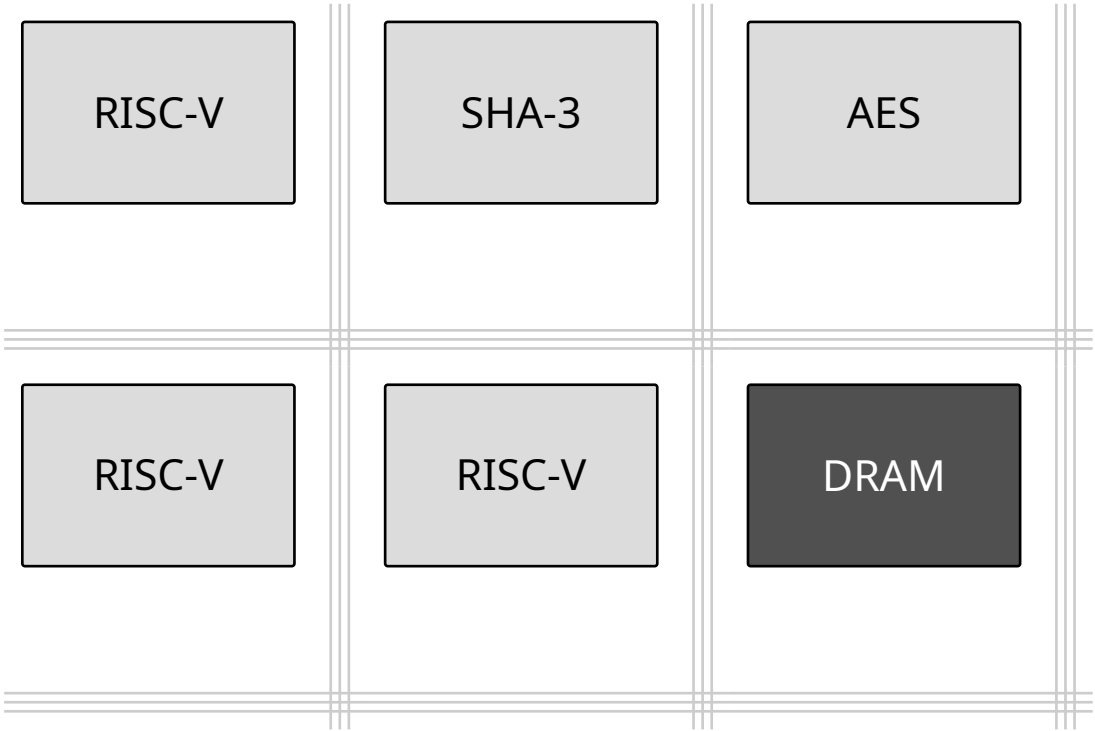
- Genode is a spin-off of the chair
- NOVA was initially built at our chair
- Now developed at BlueRock Security

Example: M^3

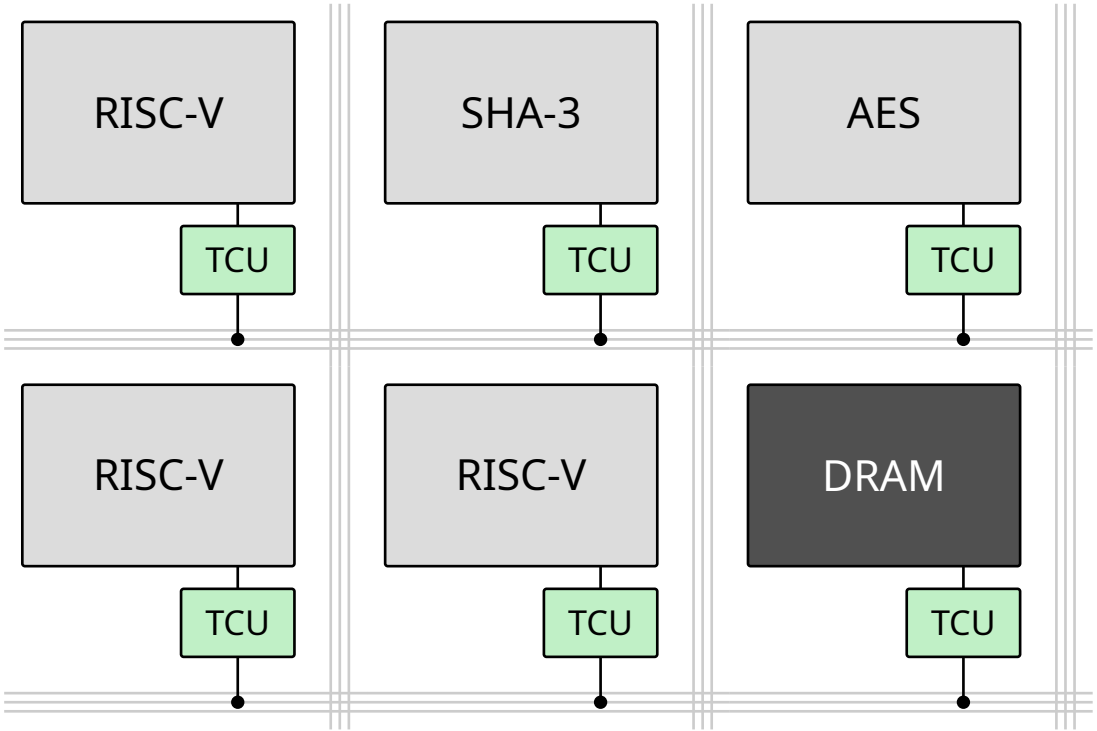
Example: M³



Example: M³

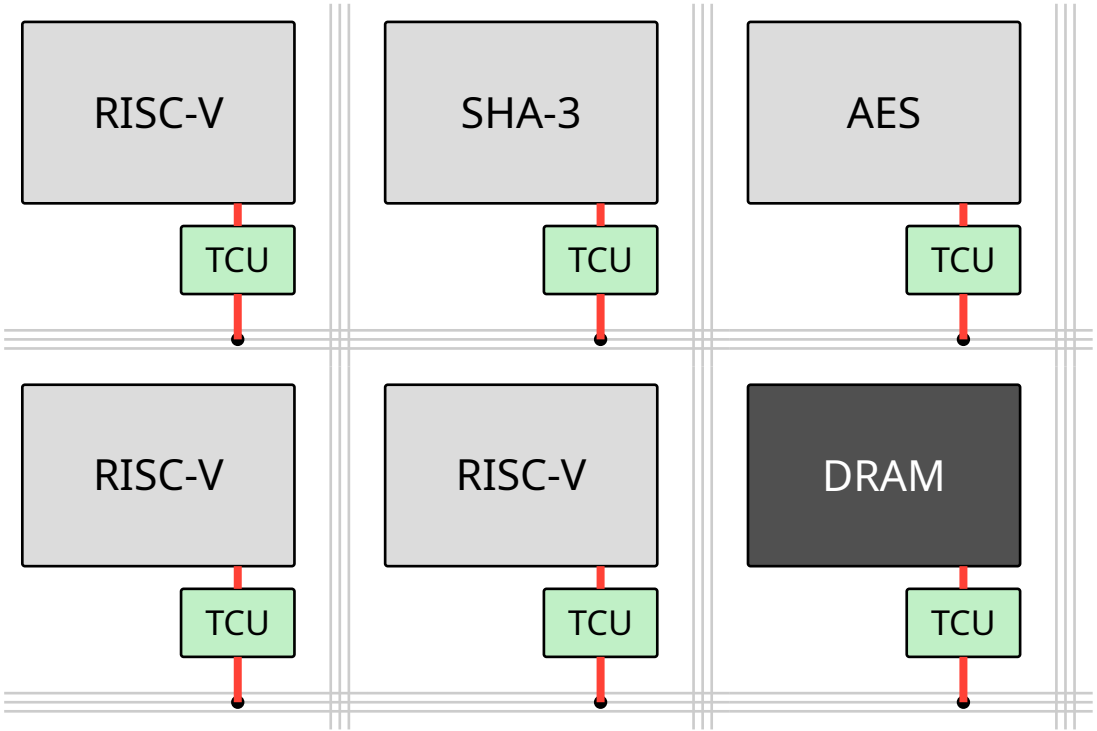


Example: M³



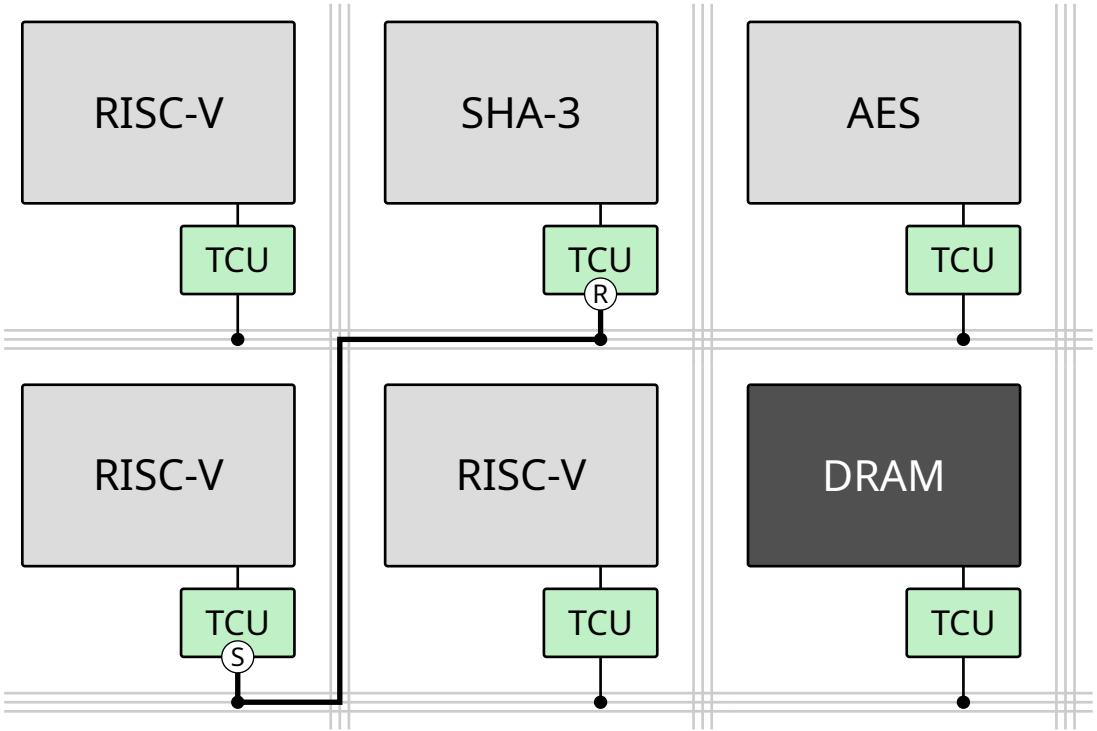
- Apply µkernel concept to hardware
- TCU as new HW component

Example: M³



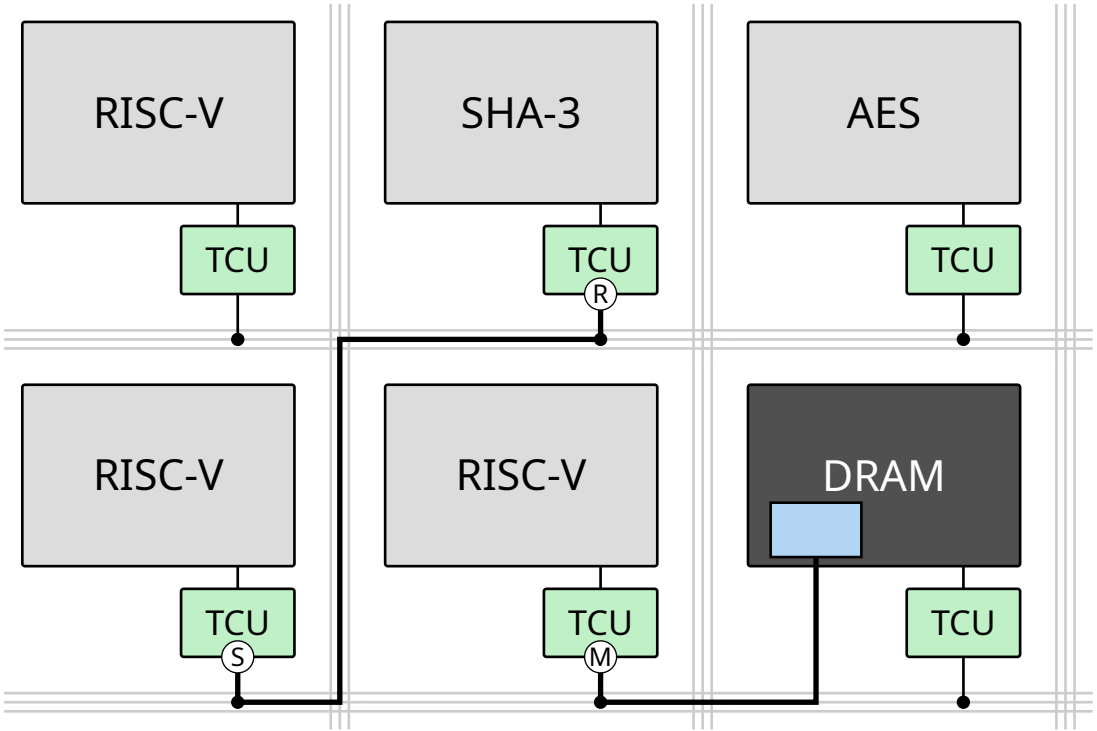
- Apply µkernel concept to hardware
- TCU as new HW component
- Tiles are isolated

Example: M³



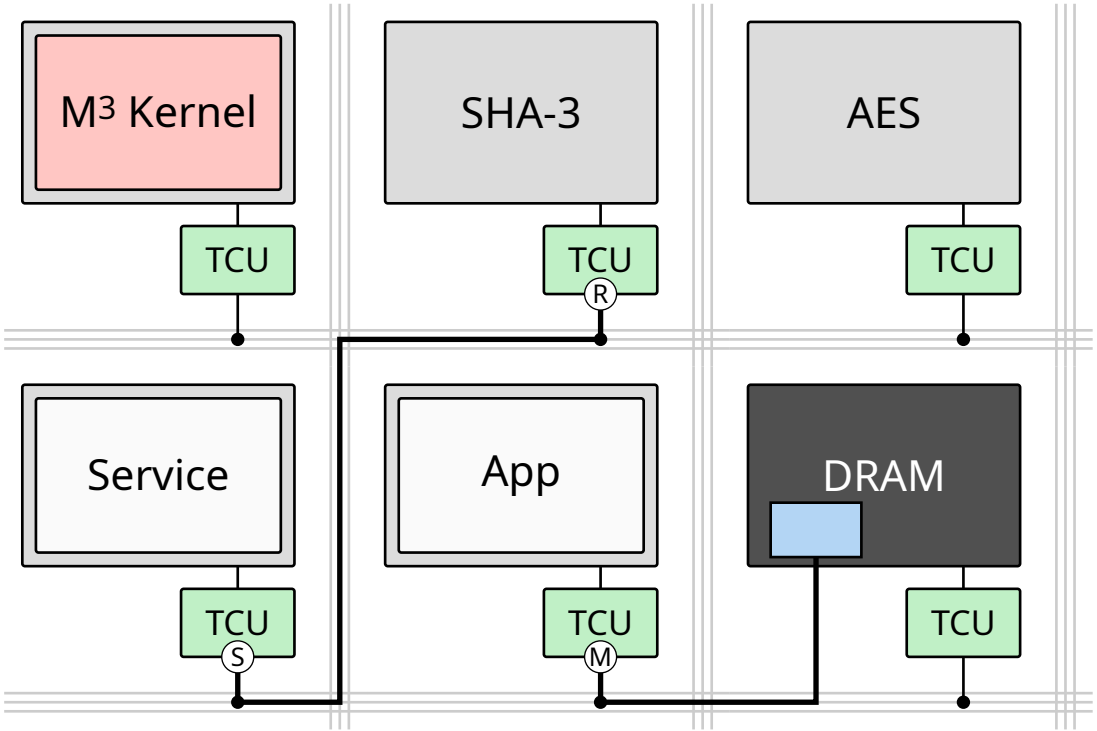
- Apply µkernel concept to hardware
- TCU as new HW component
- Tiles are isolated
- Selective communication

Example: M³



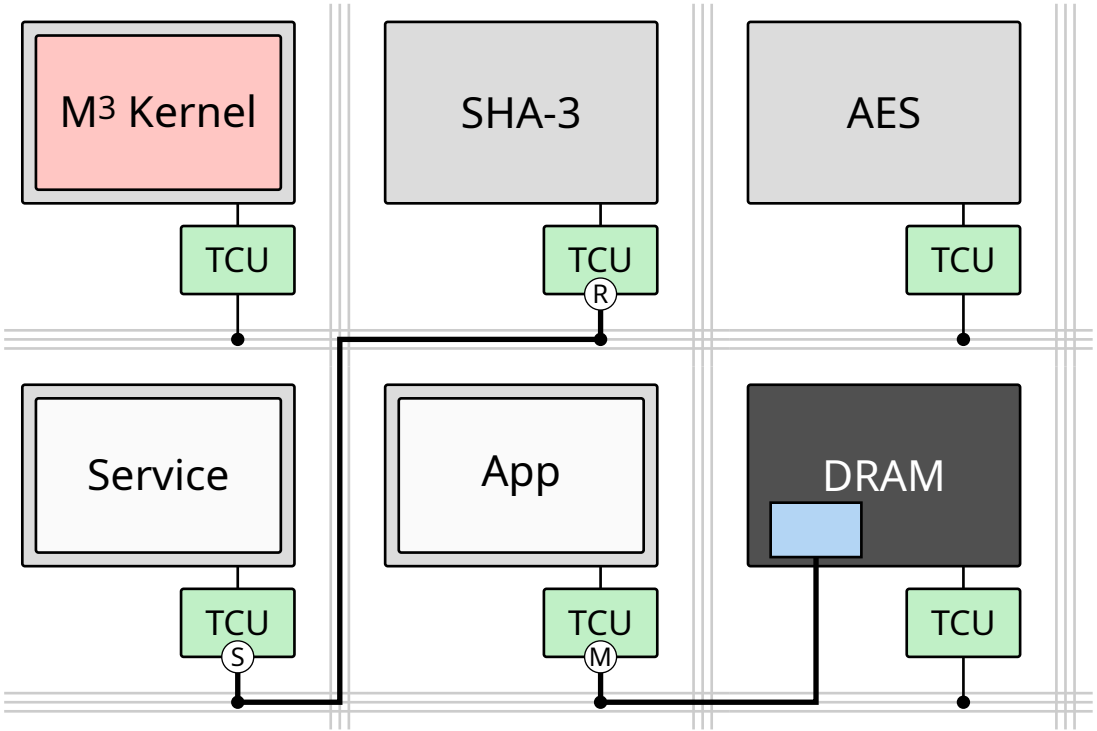
- Apply µkernel concept to hardware
- TCU as new HW component
- Tiles are isolated
- Selective communication

Example: M³



- Apply µkernel concept to hardware
- TCU as new HW component
- Tiles are isolated
- Selective communication
- Dedicated kernel tile

Example: M³



- Apply µkernel concept to hardware
- TCU as new HW component
- Tiles are isolated
- Selective communication
- Dedicated kernel tile
- Started at our chair
- Now continued at Barkhausen Institut

- Organization
- **Monolithic vs. Microkernel**
  - Kernel Design Comparison
  - Examples of Microkernel-based Systems
  - **Vision vs. Reality**
- Overview About L4/NOVA

Monolithic Design: Reality Check

Monolithic Design: Reality Check

✓ Flexibility and Customizability

- Monolithic kernels are typically modular

Monolithic Design: Reality Check

✓ Flexibility and Customizability

- Monolithic kernels are typically modular

✓ Maintainability and complexity

- Monolithic kernels have layered architecture

Monolithic Design: Reality Check

✓ Flexibility and Customizability

- Monolithic kernels are typically modular

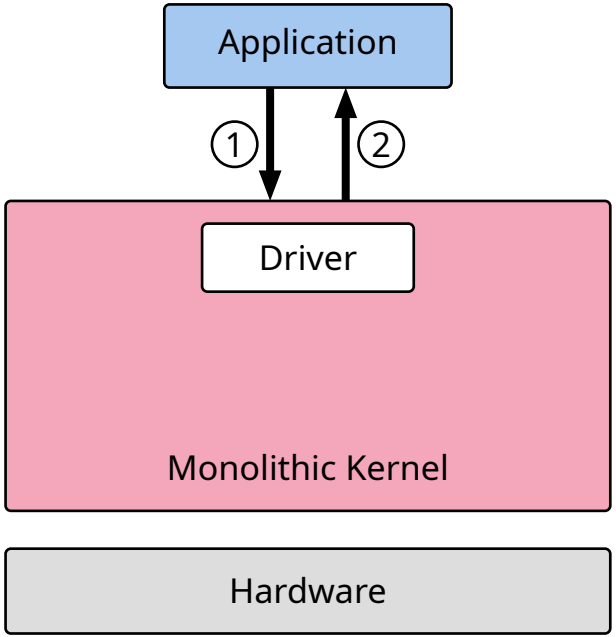
✓ Maintainability and complexity

- Monolithic kernels have layered architecture

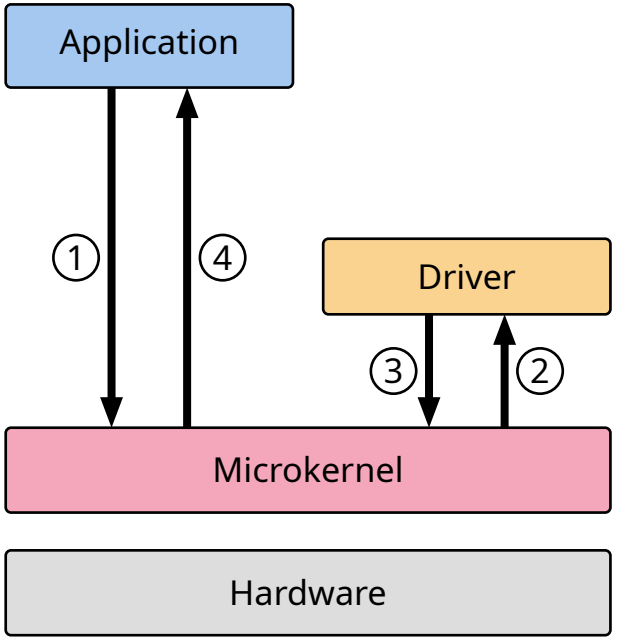
✗ Robustness and Security

- Microkernels are superior due to isolated system components
- Trusted code size
  - NOVA: 9.000 LOC
  - Linux: > 1.000.000 LOC (without drivers, arch, fs)

Performance vs. Robustness

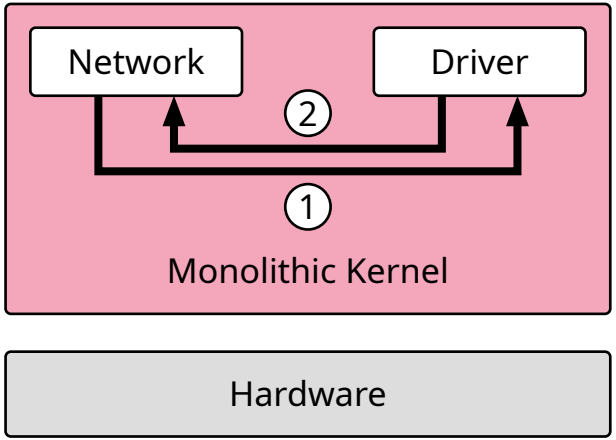


2 kernel entries/exits

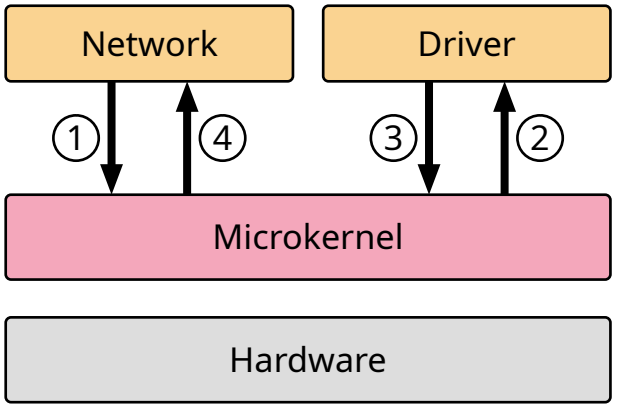


4 kernel entries/exits
2 context switches

Performance vs. Robustness



2 function calls/returns



4 kernel entries/exits
2 context switches

Challenges

Build functionally powerful and fast microkernels

- Provide abstractions and mechanisms
- Fast communication primitive (IPC)
- Fast context switches and kernel entries/exits

➡️ **Subject of this lecture**

Challenges

Build functionally powerful and fast microkernels

- Provide abstractions and mechanisms
- Fast communication primitive (IPC)
- Fast context switches and kernel entries/exits

➡️ **Subject of this lecture**

Build efficient OS services

- Memory management
- Device drivers
- File systems

➡️ **subject of lecture “Microkernel-based operating systems”**

- Organization
- Monolithic vs. Microkernel
- **Overview About L4/NOVA**
  - **Introduction**
  - Abstractions and Mechanisms

L4 Microkernel Family

- Originally developed by Jochen Liedtke
  (GMD / IBM Research)

- Current development:
  - UNSW/OKLABS: OKL4, seL4
  - Kernkonzept/TU Dresden/Barkhausen Institut: L4Re
  - BlueRock Security/Genode Labs/Cyberus Technology: NOVA
  - Barkhausen Institut: M^3

More Microkernels (incomplete)

- Singularity @ Microsoft Research
- K42 @ IBM Research
- Chorus/ChorusOS @ Sun Microsystems
- PikeOS @ SYSGO AG
- EROS/CoyotOS @ John Hopkins University
- Minix @ FU Amsterdam
- Pistachio @ KIT
- Barrelfish @ ETH Zurich
- Harmony OS @ Huawei
- Fuchsia with Zircon microkernel @ Google

L4 Concepts

- Jochen Liedtke: “A microkernel does no real work”
  - Kernel provides only inevitable mechanisms
  - No policies implemented in the kernel

L4 Concepts

- Jochen Liedtke: “A microkernel does no real work”
  - Kernel provides only inevitable mechanisms
  - No policies implemented in the kernel
- Abstractions
  - Tasks with address spaces
  - Threads executing programs/code

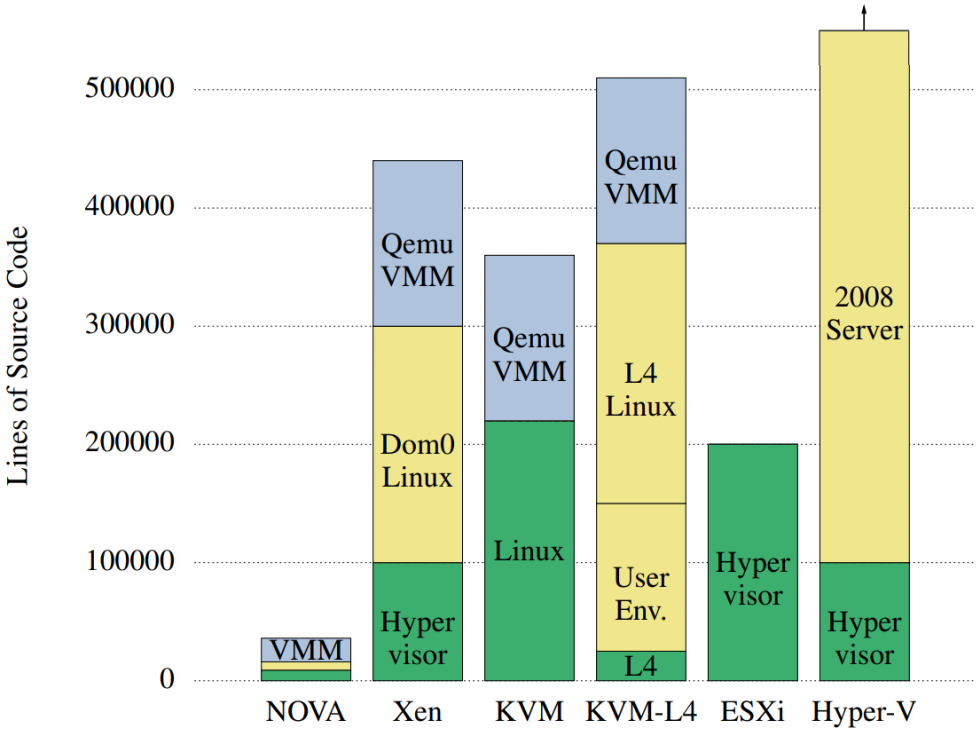
L4 Concepts

- Jochen Liedtke: “A microkernel does no real work”
  - Kernel provides only inevitable mechanisms
  - No policies implemented in the kernel
- Abstractions
  - Tasks with address spaces
  - Threads executing programs/code
- Mechanisms
  - Resource access control
  - Scheduling
  - Communication (IPC)

Why NOVA?

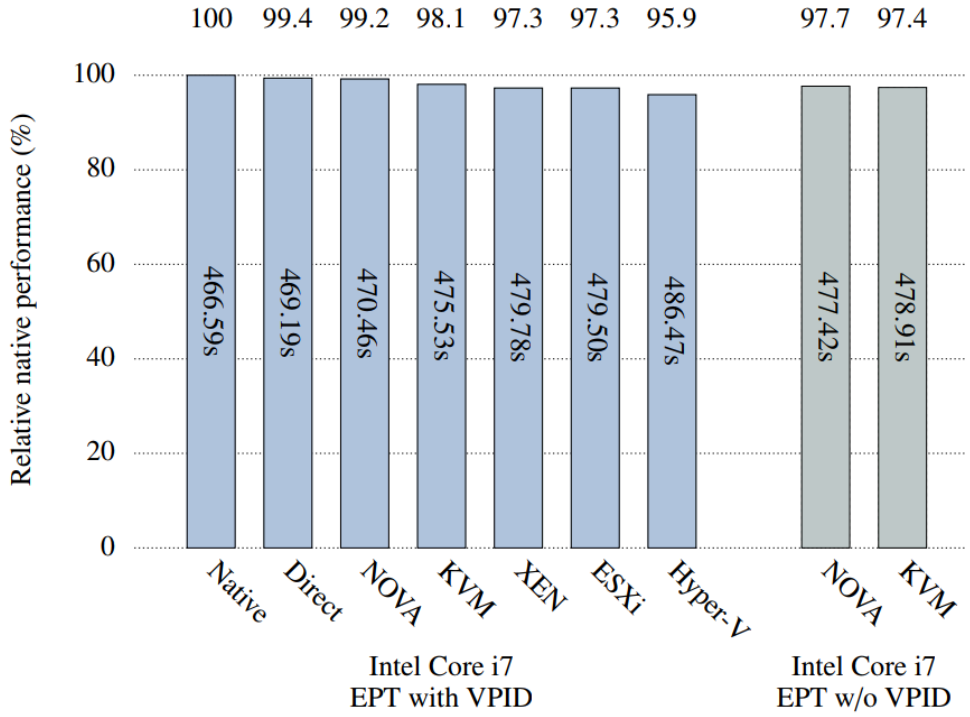
- NOVA is small and simple (≈ 9000 SLOC)
- NOVA is arguably elegant
- NOVA is efficient
- NOVA is open source: https://github.com/udosteinberg/NOVA

Why NOVA: TCB Size



Udo Steinberg and Bernhard Kauer: **NOVA: A microhypervisor-based secure virtualization architecture**, EuroSys 2010

Why NOVA: Performance (Linux kernel compilation)



Udo Steinberg and Bernhard Kauer: **NOVA: A microhypervisor-based secure virtualization architecture**, EuroSys 2010

- Organization
- Monolithic vs. Microkernel
- **Overview About L4/NOVA**
  - Introduction
  - **Abstractions and Mechanisms**

Protection Domain (PD)

- PD is a resource container
  - Object capabilities (e.g., PD, execution context, ...)
  - Memory capabilities (pages)
  - I/O port capabilities (on x86)

Protection Domain (PD)

- PD is a resource container
  - Object capabilities (e.g., PD, execution context, ...)
  - Memory capabilities (pages)
  - I/O port capabilities (on x86)
- Capabilities can be exchanged between PDs
- Typically, PD contains one or more execution contexts
- Not hierarchical (in the kernel)

Protection Domain (PD)

- PD is a resource container
  - Object capabilities (e.g., PD, execution context, …)
  - Memory capabilities (pages)
  - I/O port capabilities (on x86)
- Capabilities can be exchanged between PDs
- Typically, PD contains one or more execution contexts
- Not hierarchical (in the kernel)

NOVA to L4Re

Protection Domain ≃ Task

Execution Context (EC)

- EC is the entity that executes code
  - User code (application)
  - Kernel code (syscalls, interrupts, exceptions)

Execution Context (EC)

- EC is the entity that executes code
  - User code (application)
  - Kernel code (syscalls, interrupts, exceptions)
- Has a user thread control block (UTCB) for IPC
- Belongs to exactly one PD

Execution Context (EC)

- EC is the entity that executes code
  - User code (application)
  - Kernel code (syscalls, interrupts, exceptions)
- Has a user thread control block (UTCB) for IPC
- Belongs to exactly one PD
- Receives time to execute from scheduling contexts
- Pinned on a CPU (not migratable)
- Three variants: Local EC, Global EC and VCPU

Execution Context (EC)

- EC is the entity that executes code
  - User code (application)
  - Kernel code (syscalls, interrupts, exceptions)
- Has a user thread control block (UTCB) for IPC
- Belongs to exactly one PD
- Receives time to execute from scheduling contexts
- Pinned on a CPU (not migratable)
- Three variants: Local EC, Global EC and VCPU

NOVA to L4Re

Execution Context + Scheduling Context ≃ Thread

Scheduling Context (SC)

- SC supplies an EC with time
- Has a budget and a priority

Scheduling Context (SC)

- SC supplies an EC with time
- Has a budget and a priority
- NOVA schedules SCs in round robin fashion
- Scheduling an SC, activates the associated EC

Scheduling Context (SC)

- SC supplies an EC with time
- Has a budget and a priority
- NOVA schedules SCs in round robin fashion
- Scheduling an SC, activates the associated EC

NOVA to L4Re

Execution Context + Scheduling Context ≃ Thread

Portal (PT)

- A portal is an endpoint for synchronous IPC
- Each portal belongs to exactly one (Local) EC

Portal (PT)

- A portal is an endpoint for synchronous IPC
- Each portal belongs to exactly one (Local) EC
- Calling a portal, transfers control to the associated EC
- Data and capability exchange via UTCB
- No cross-core IPC

Portal (PT)

- A portal is an endpoint for synchronous IPC
- Each portal belongs to exactly one (Local) EC
- Calling a portal, transfers control to the associated EC
- Data and capability exchange via UTCB
- No cross-core IPC

NOVA to L4Re

Portal ≃ IPC Gate

Semaphore (SM)

- A semaphore offers asynchronous communication (one bit)
- Supports: up, down and zero

Semaphore (SM)

- A semaphore offers asynchronous communication (one bit)
- Supports: up, down and zero
- Can be used cross-core
- Hardware interrupts are represented as semaphores

Semaphore (SM)

- A semaphore offers asynchronous communication (one bit)
- Supports: up, down and zero
- Can be used cross-core
- Hardware interrupts are represented as semaphores

NOVA to L4Re

Semaphore ≃ IRQ

Capabilities

- Access to kernel objects is provided by capabilities
- Capability is a pair: pointer to kernel object, permissions

Capabilities

- Access to kernel objects is provided by capabilities
- Capability is a pair: pointer to kernel object, permissions
- Every PD has its own capability space (local, isolated)
- Capabilites can be exchanged:
  - Delegate: copy capability from one Cap Space to the other
  - Revoke: remove capability, recursively
- Applications use selectors to denote capabilities

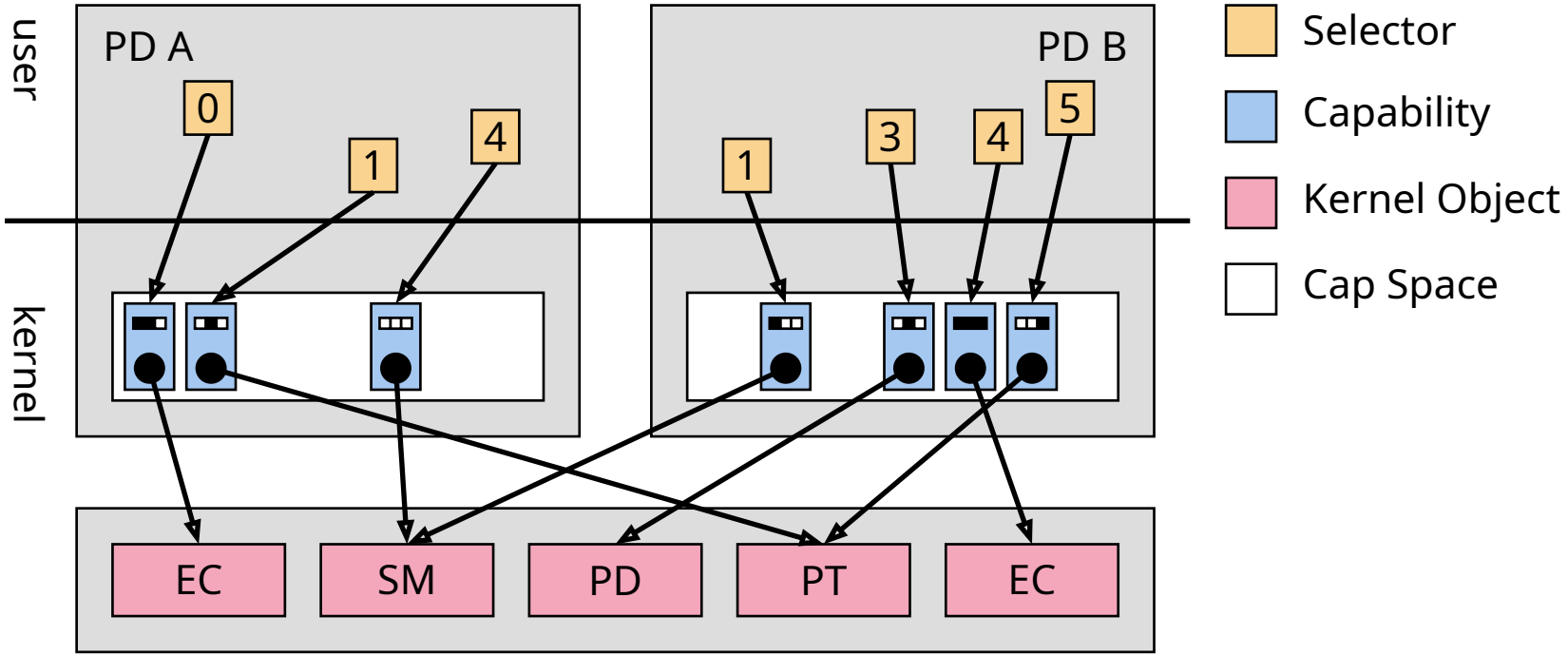
Capabilities

- Access to kernel objects is provided by capabilities
- Capability is a pair: pointer to kernel object, permissions
- Every PD has its own capability space (local, isolated)
- Capabilites can be exchanged:
  - Delegate: copy capability from one Cap Space to the other
  - Revoke: remove capability, recursively
- Applications use selectors to denote capabilities

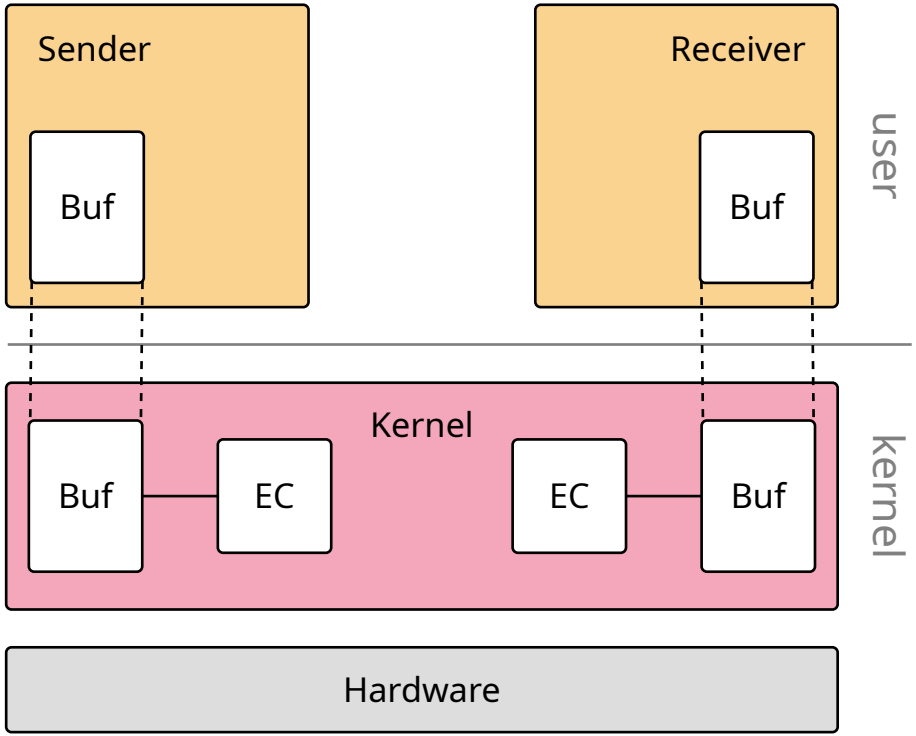
NOVA to L4Re

Delegate = Map

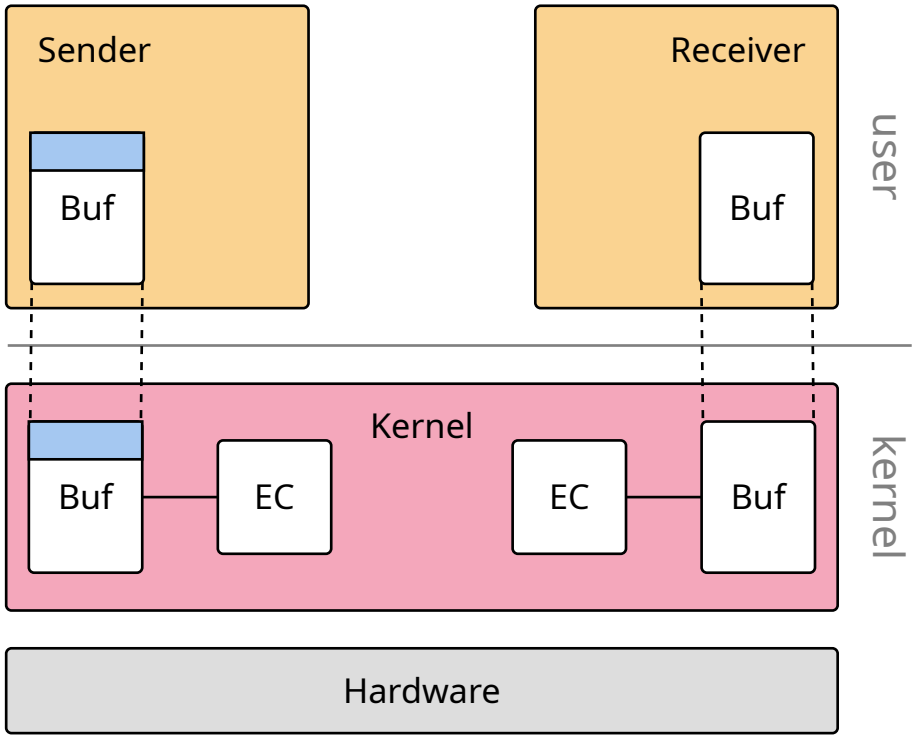
Capabilities Overview



Inter-process Communication

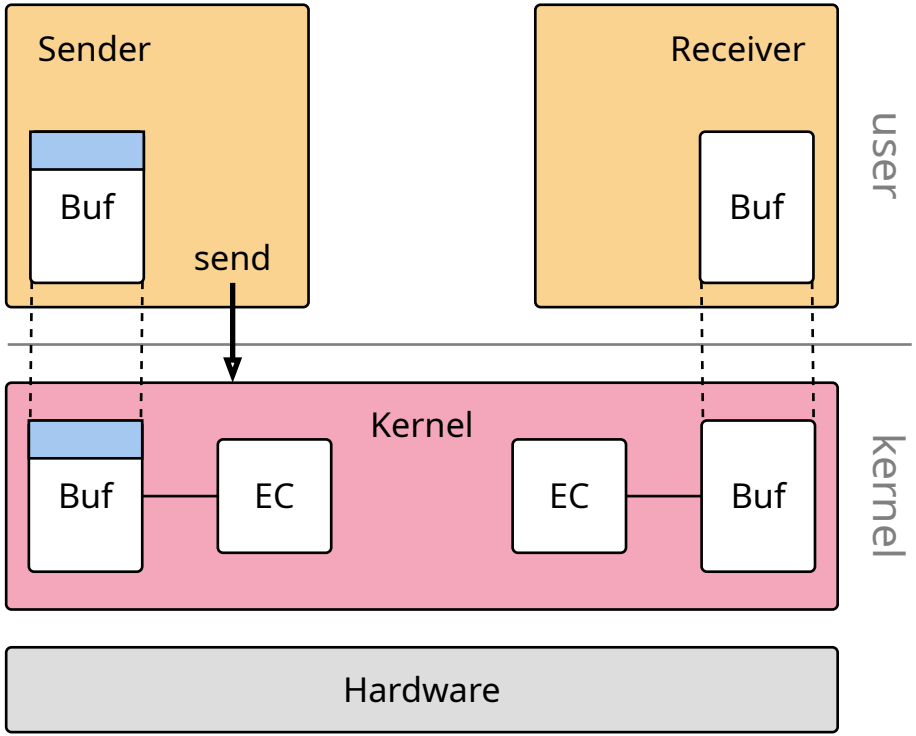


Inter-process Communication



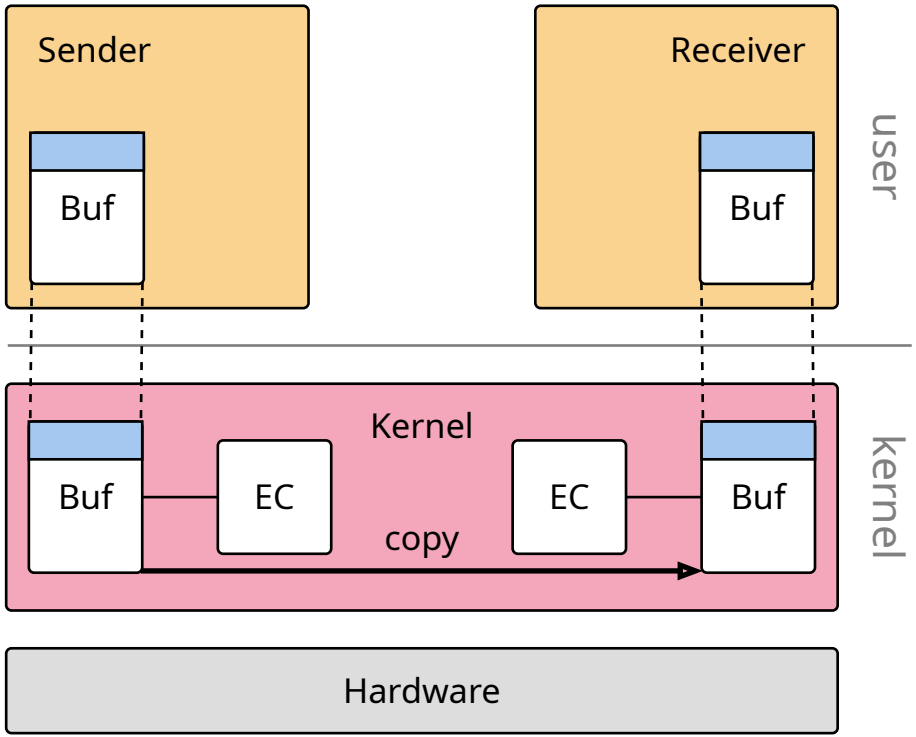
- Write message into buffer

Inter-process Communication



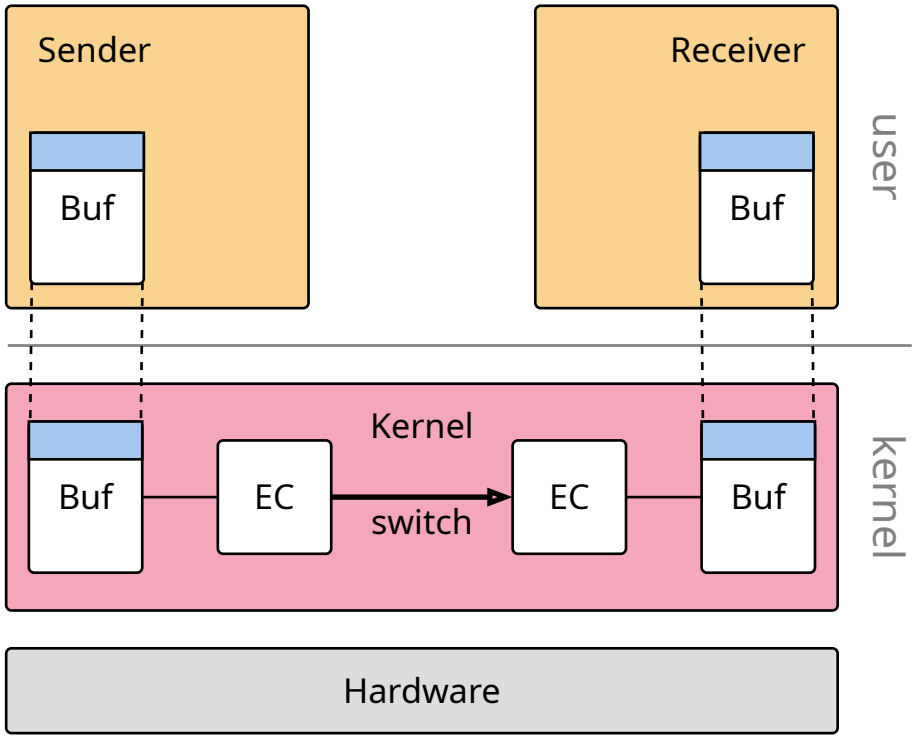
- Write message into buffer
- Perform send syscall

Inter-process Communication



- Write message into buffer
- Perform send syscall
- Kernel copies message

Inter-process Communication



- Write message into buffer
- Perform send syscall
- Kernel copies message
- Kernel switches to receiver

Schedule

Date	Lecture	Exercise
16.04.	Intro	-
23.04.	Threads	-
30.04.	Entry / Exit	Entry / Exit
07.05.	Address Spaces	ELF / Multiboot
14.05.	– *Himmelfahrt* –	
21.05.	IPC	Threads
28.05.	– *Pfingsten* –	
04.06.	Capabilities	IPC
11.06.	Case Study: seL4	Cap Basics
18.06.	Case Study: L4Re	Cap Revoke
25.06.	– *Output* –	
02.07.	Case Study: M^3	M^3 Tutorial
09.07.	Case Study: Escape	