ARTIST Summer School in Europe 2010

Autrans (near Grenoble), France
September 5-10, 2010

The L4 Microkernel

Invited Speaker: Prof. Hermann Härtig
Technische Universität Dresden

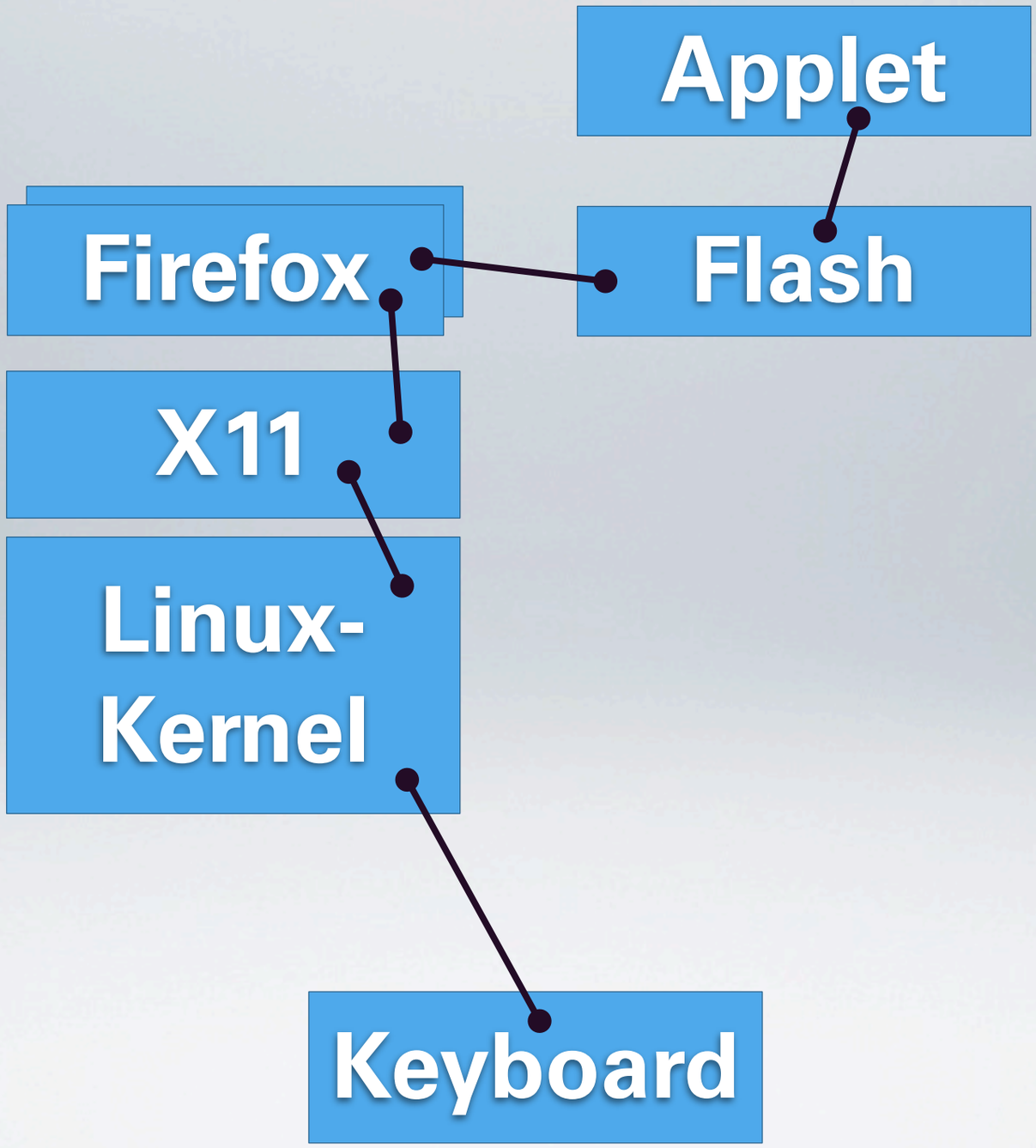
http://www.artist-embedded.org/

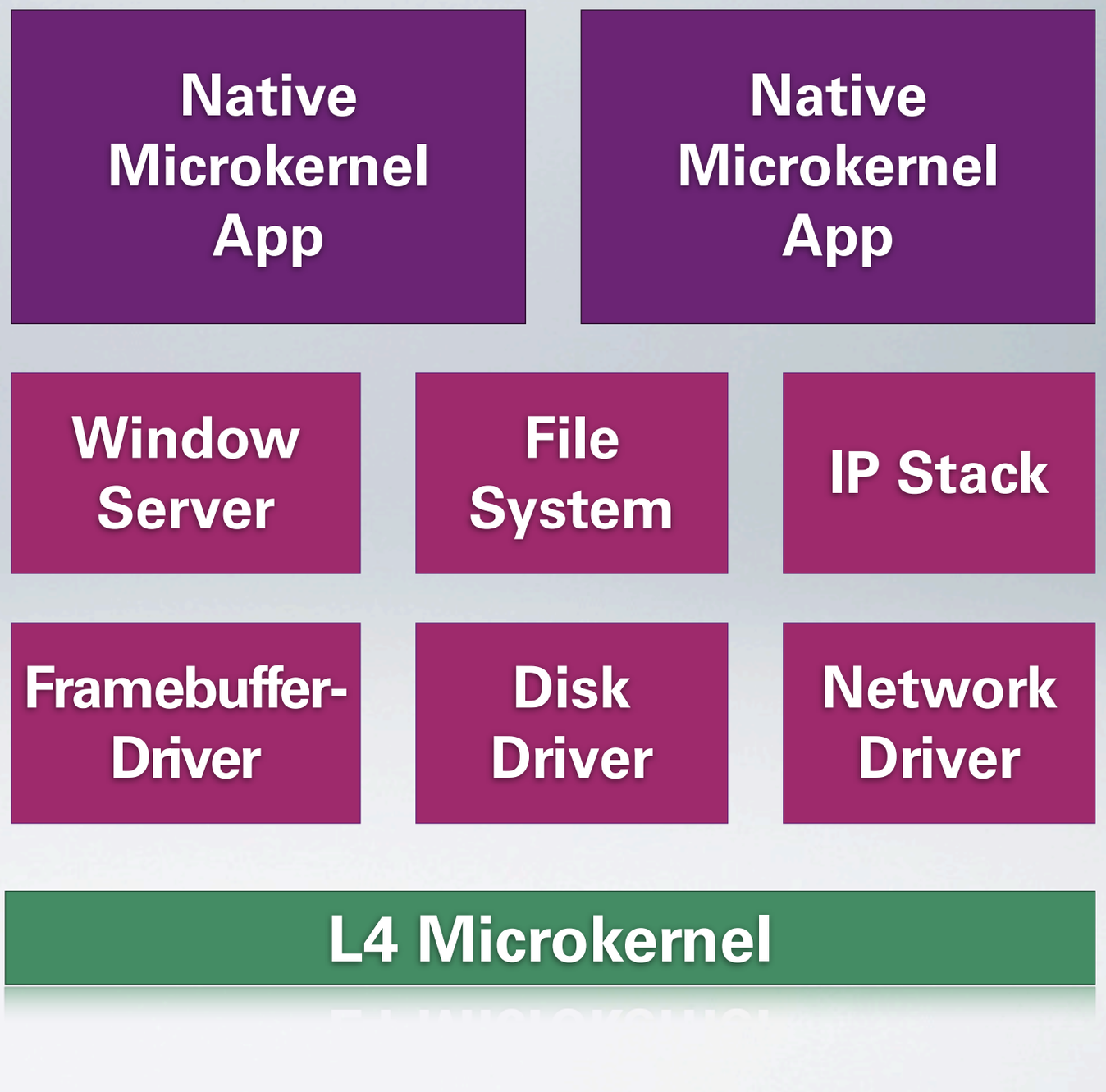
artist

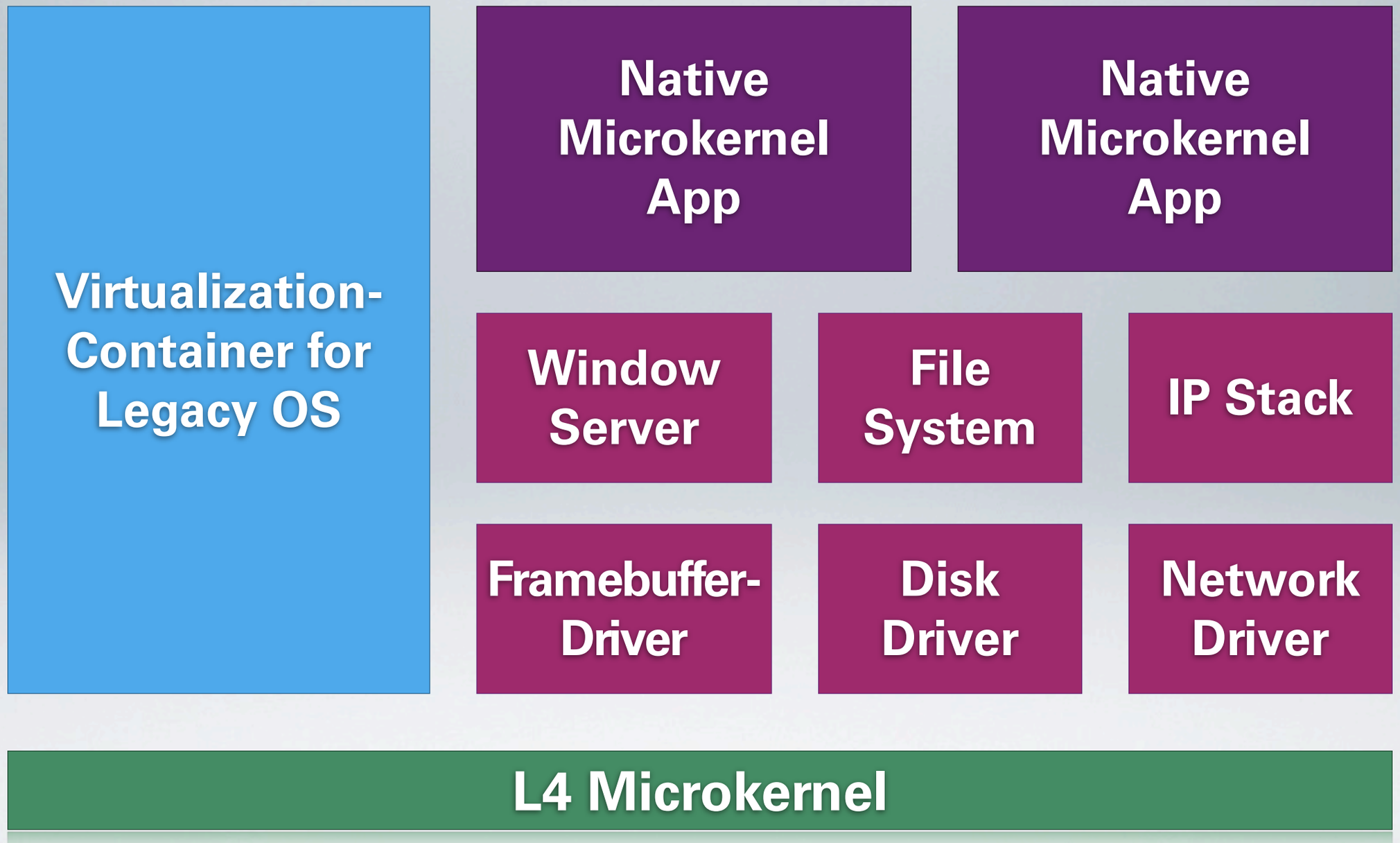
L4

SEVENTH FRAMEWORK PROGRAMME

COTS - SW




Native Microkernel App
Native Microkernel App
Window Server
File System
IP Stack
Framebuffer-Driver
Disk Driver
Network Driver
L4 Microkernel


Virtualization-Container for Legacy OS
Native Microkernel App
Native Microkernel App
Window Server
File System
IP Stack
Framebuffer-Driver
Disk Driver
Network Driver
L4 Microkernel

OUTLINE

- Using a small kernel – Hermann Härtig
  - Motivation, Cost & Benefit
  - Case studies
  - A short history of L4
  - L4 Kernel interface
- Capability system design – Michael Roitzsch
- Mobile use cases – Adam Lackorzynski

TECHNISCHE UNIVERSITÄT DRESDEN

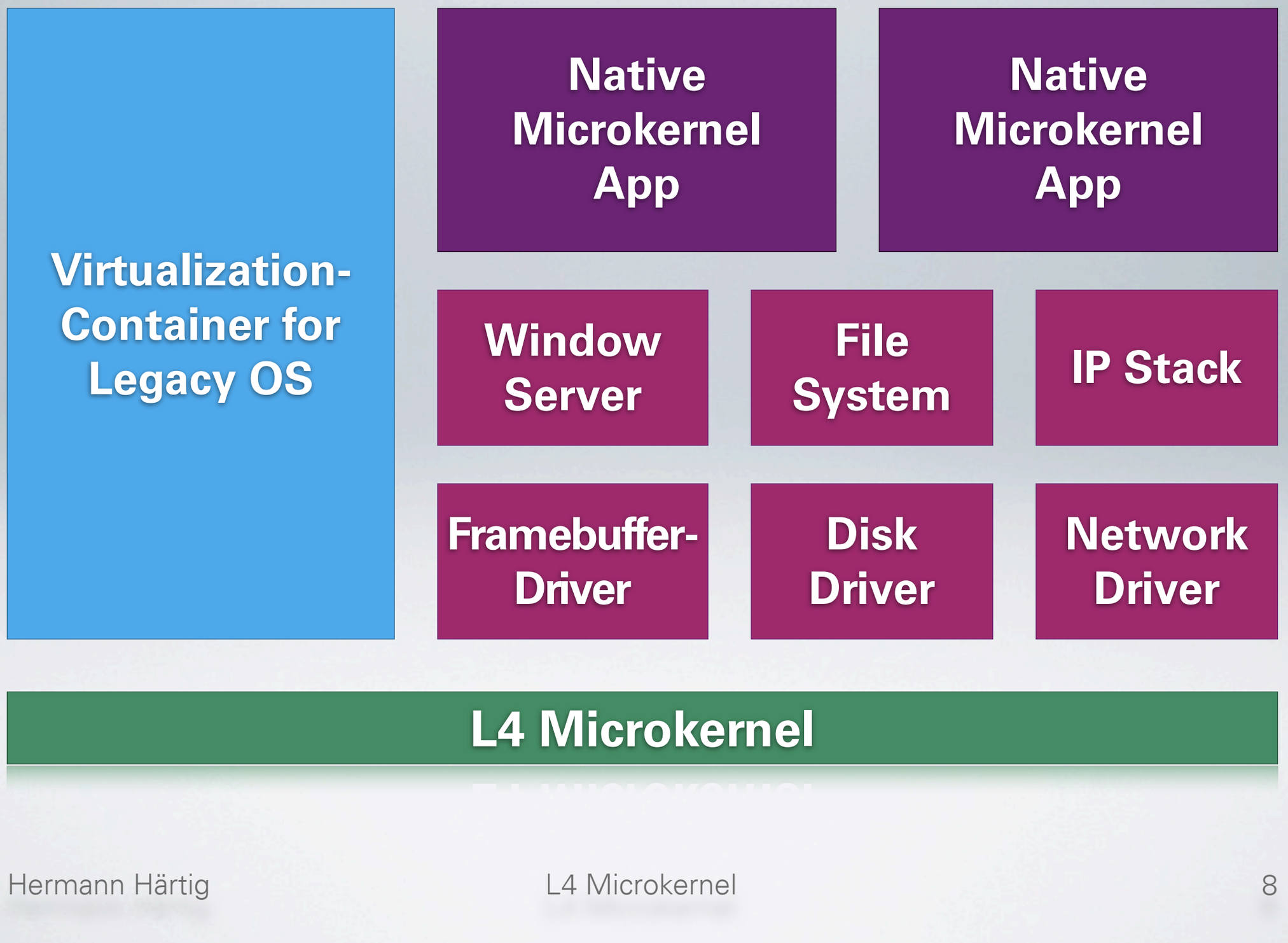
Department of Computer Science Institute of System Architecture, Operating Systems Group

USING A SMALL KERNEL

HERMANN HÄRTIG

TECHNISCHE UNIVERSITÄT DRESDEN

WHY MICRO



COST & BENEFIT

- Performance
- (Failure) Isolation
- Openness
- Small (Minimal) Trusted Computing Base

BENEFITS

- Flexibility?
  SW engineering ./. microkernels
- Difficulty to build?
  can be harder to build

ISOLATION

- Separate address spaces for components
- Message passing interfaces
- Communication controlled by capabilities

- Immediate: I/O Drivers tamed
- Base for fault containment

- fault recovery?

Virtual machines

- provide separate machines
- requires
  - emulation of physical HW interface
  - an operating system in each VM
- large-grained components

more later

ALTERNATIVE

Languages with component-support

- Use one language or language family
- Fine-grained components (modules, objects, …)
- Compiler and runtime enforce isolation
- Closed systems
- Examples: Burroughs 7700, B extended Algol, Espol, Concurrent Pascal, Modula, Oberon, various Java systems

Microkernels

- Minimal kernel and hardware enforce separation
- Only kernel runs in CPU privileged mode
- Components are user-level processes
- No restrictions on component software
- Reuse of legacy software

MINIMAL TCB

Trusted Computing Base

„A small amount of software and hardware that security depends on and that we distinguish from a much larger amount that can misbehave without affecting security.“

— Lampson et al.

MINIMAL TCB

Trusted Computing Base

„A small amount of software and hardware that * _____ depends on and that we distinguish from a much larger amount that can misbehave without affecting * _____ .“

In this lecture:

_____: security, real-time, fault tolerance, ...

TCB is application specific

MINIMAL TCB

- General Approach:
  - Divide system into uncritical and (minimal) critical parts
    - Include minimal set of components into TCB
      - offload uncritical parts, e.g. into legacy SW
  - Split critical part into isolated components
- Benefit:
  - small application-specific TCB

TECHNISCHE UNIVERSITÄT DRESDEN

CASE STUDIES

App

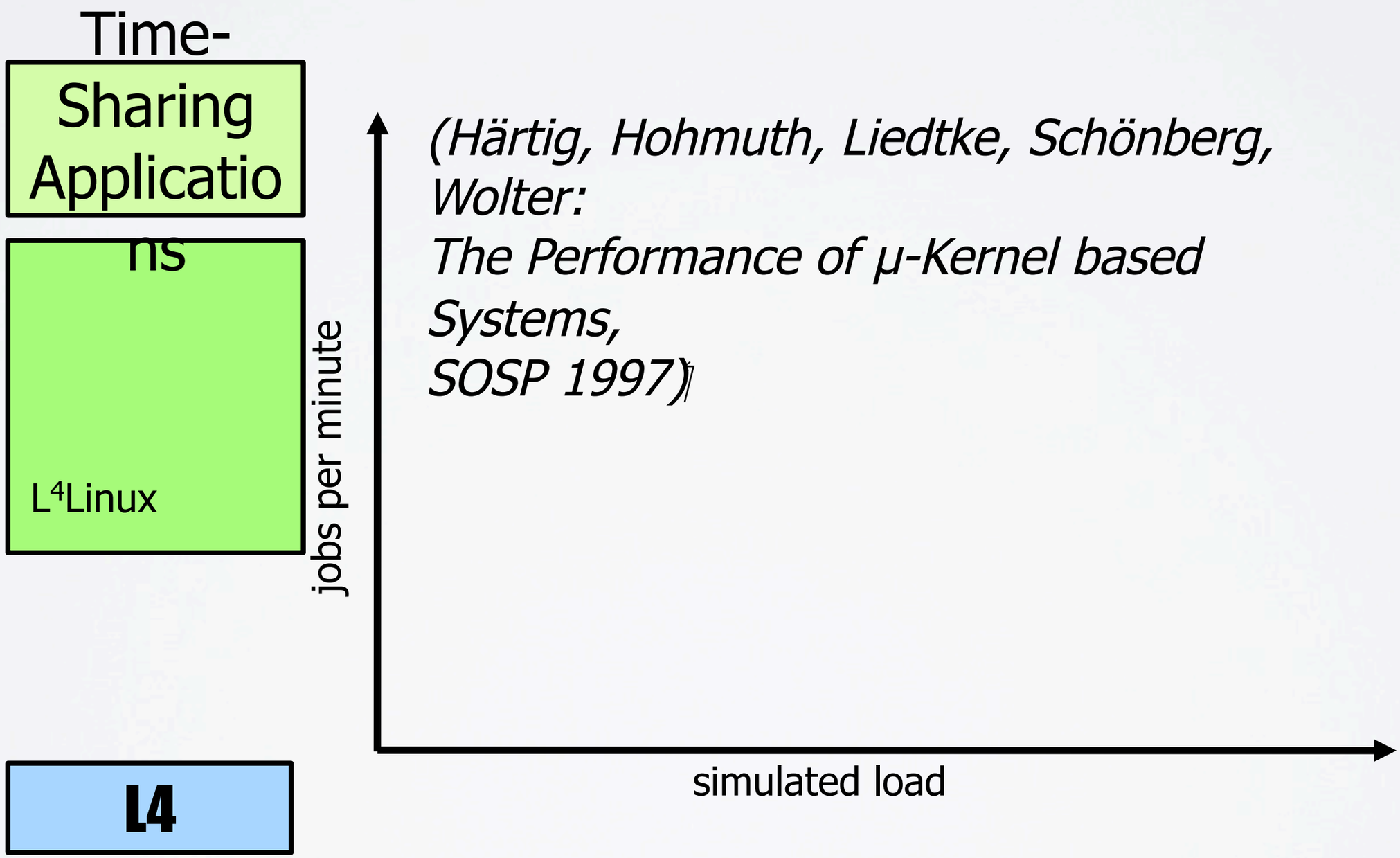
App

X11

L4Linux Server

L4 Microkernel Fiasco.OC

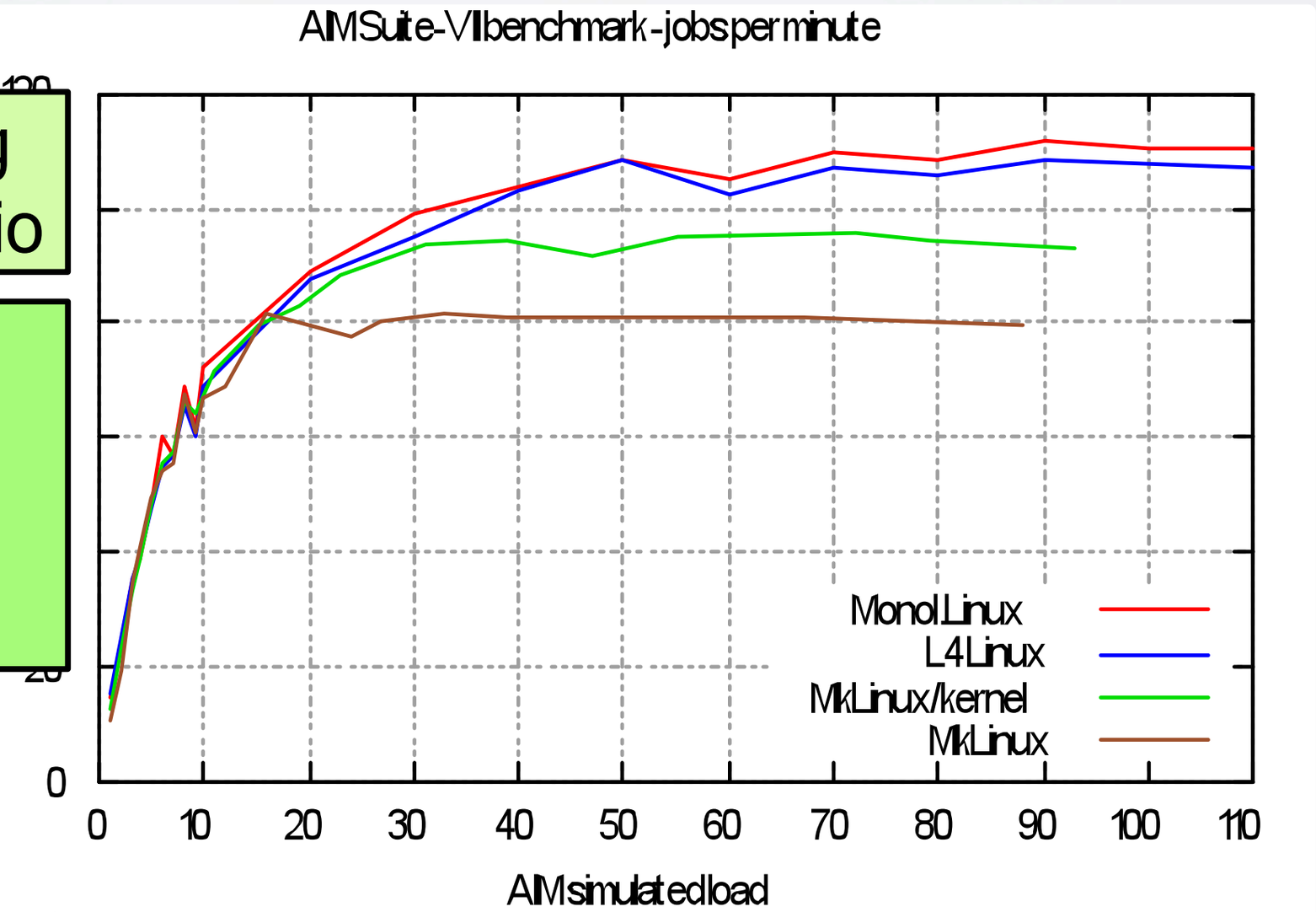
COST ≈1997



(Härtig, Hohmuth, Liedtke, Schönberg, Wolter:
The Performance of µ-Kernel based Systems,
SOSP 1997)

Time-Sharing Applications

L⁴Linux



L4

App App

X11

L4Linux Server

App App

X11

L4Linux Server

L4 Microkernel Fiasco.OC

App App App App

X11 X11

L⁴Linux Server L⁴Linux Server

Resource Management and Virtualization Support Layer

moe ned io rtc mag

L4 Microkernel Fiasco.OC

App App App App

X11

L⁴Linux Server

L⁴Symbian Server

moe ned io rtc mag

L4 Microkernel Fiasco.OC

App App

X11

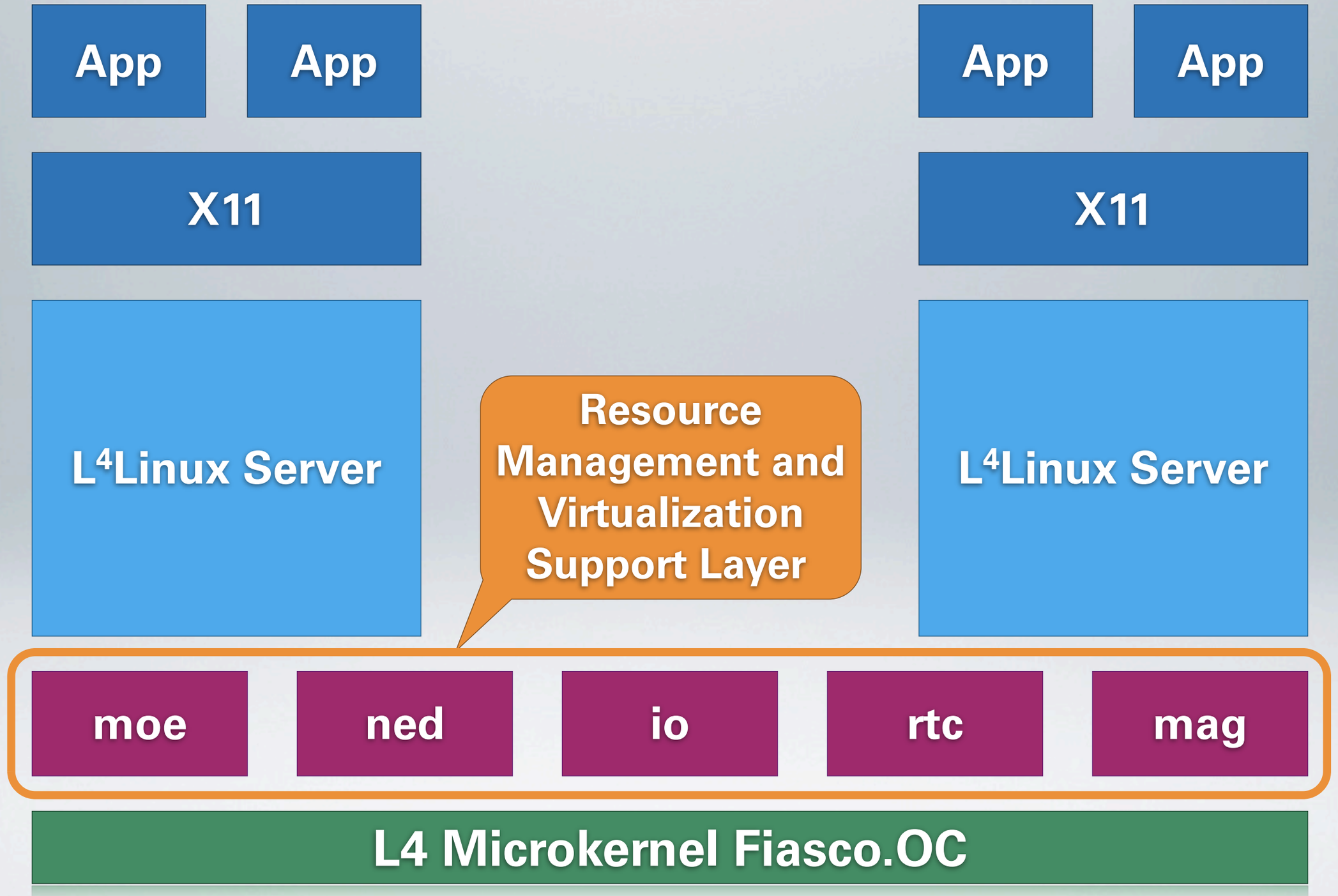
L4Linux Server

App App

X11

L4Linux Server

L4 Microkernel Fiasco.OC


App
App
App
App
X11
X11
L4Linux Server
L4Linux Server
Resource Management and Virtualization Support Layer
moe
ned
io
rtc
mag
L4 Microkernel Fiasco.OC

App App App App

X11

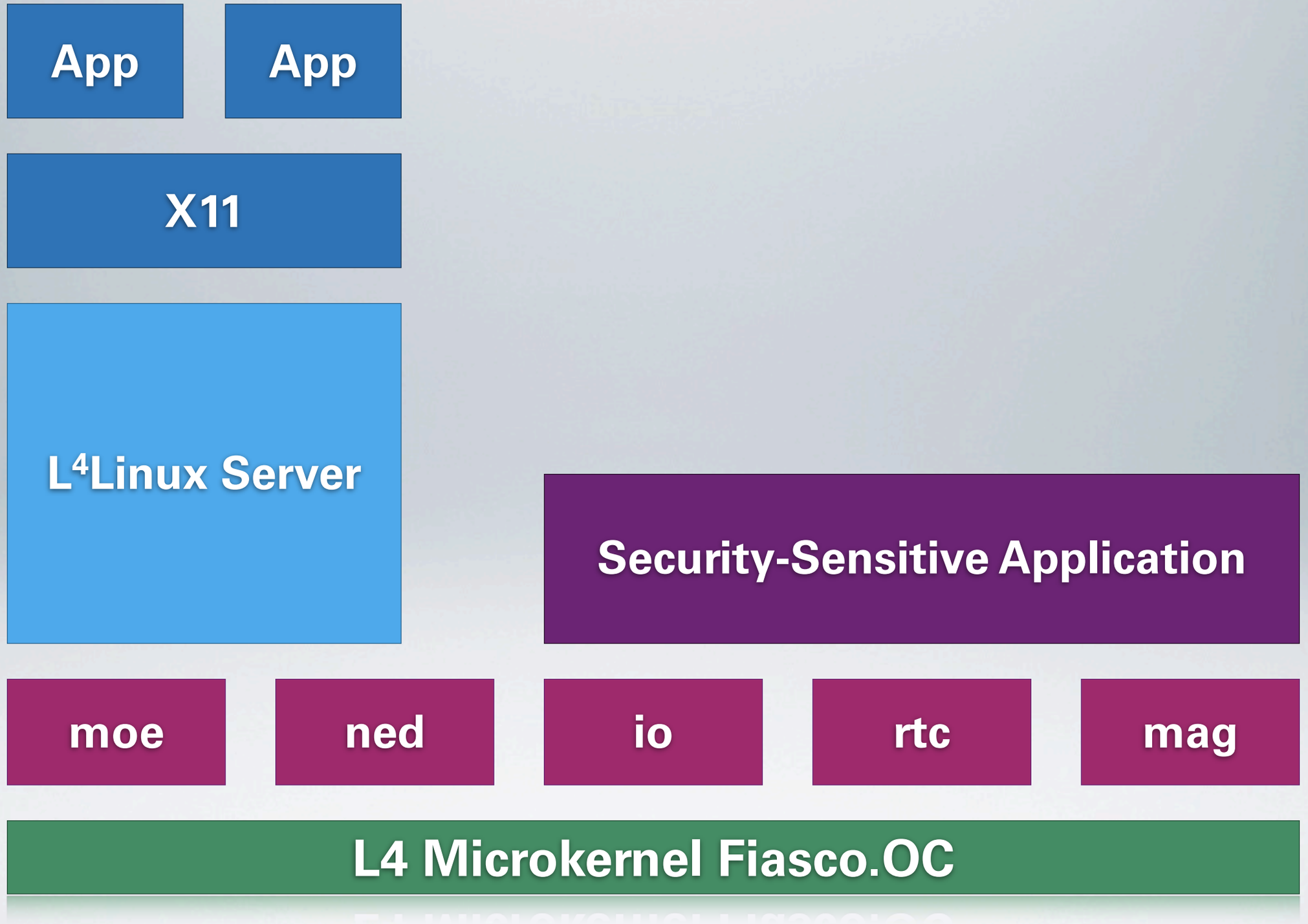
L⁴Linux Server

L⁴Symbian Server

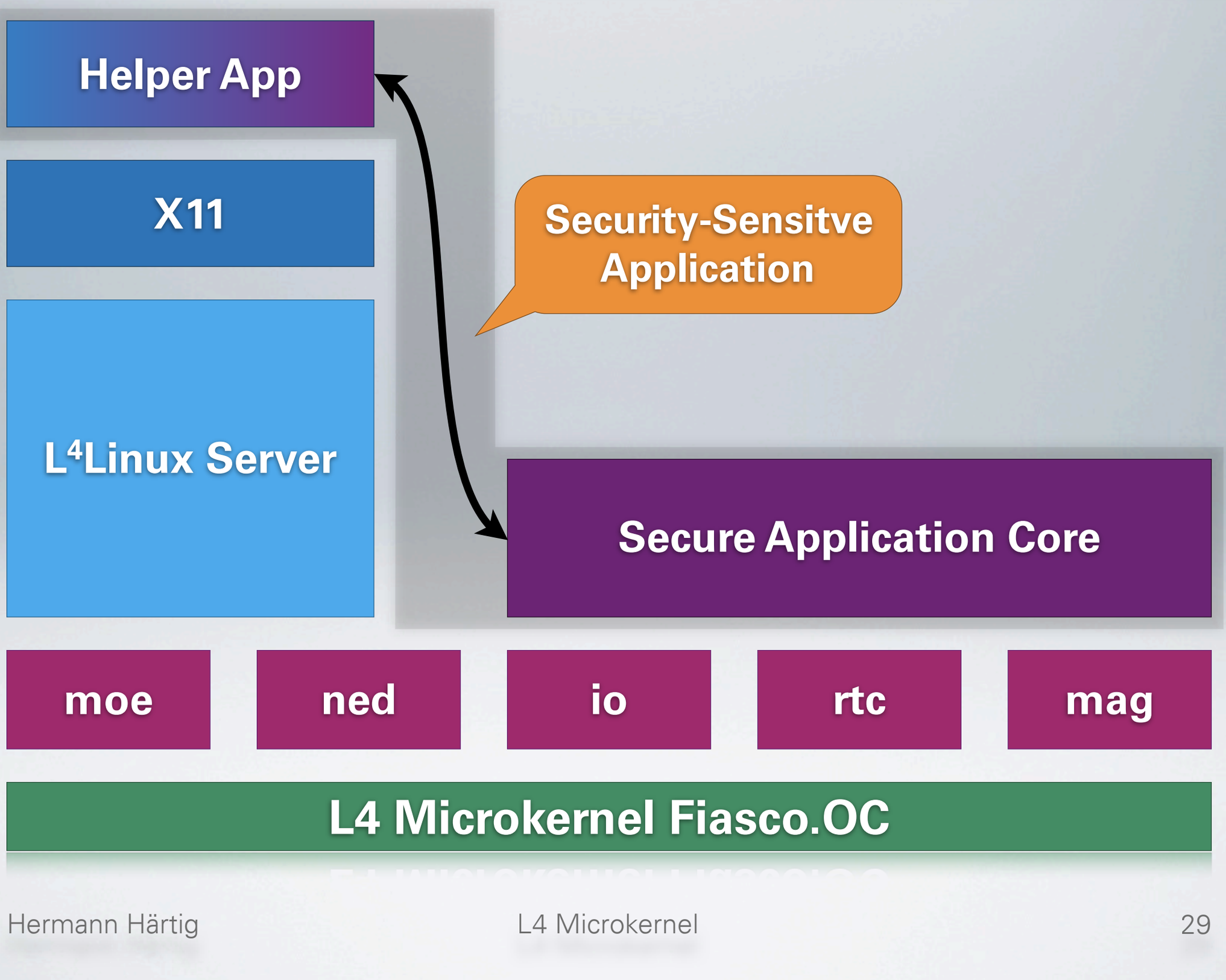
moe ned io rtc mag

L4 Microkernel Fiasco.OC

NATIVE APPS


App
App
X11
L4Linux Server
Security-Sensitive Application
moe
ned
io
rtc
mag
L4 Microkernel Fiasco.OC

HYBRID APPS

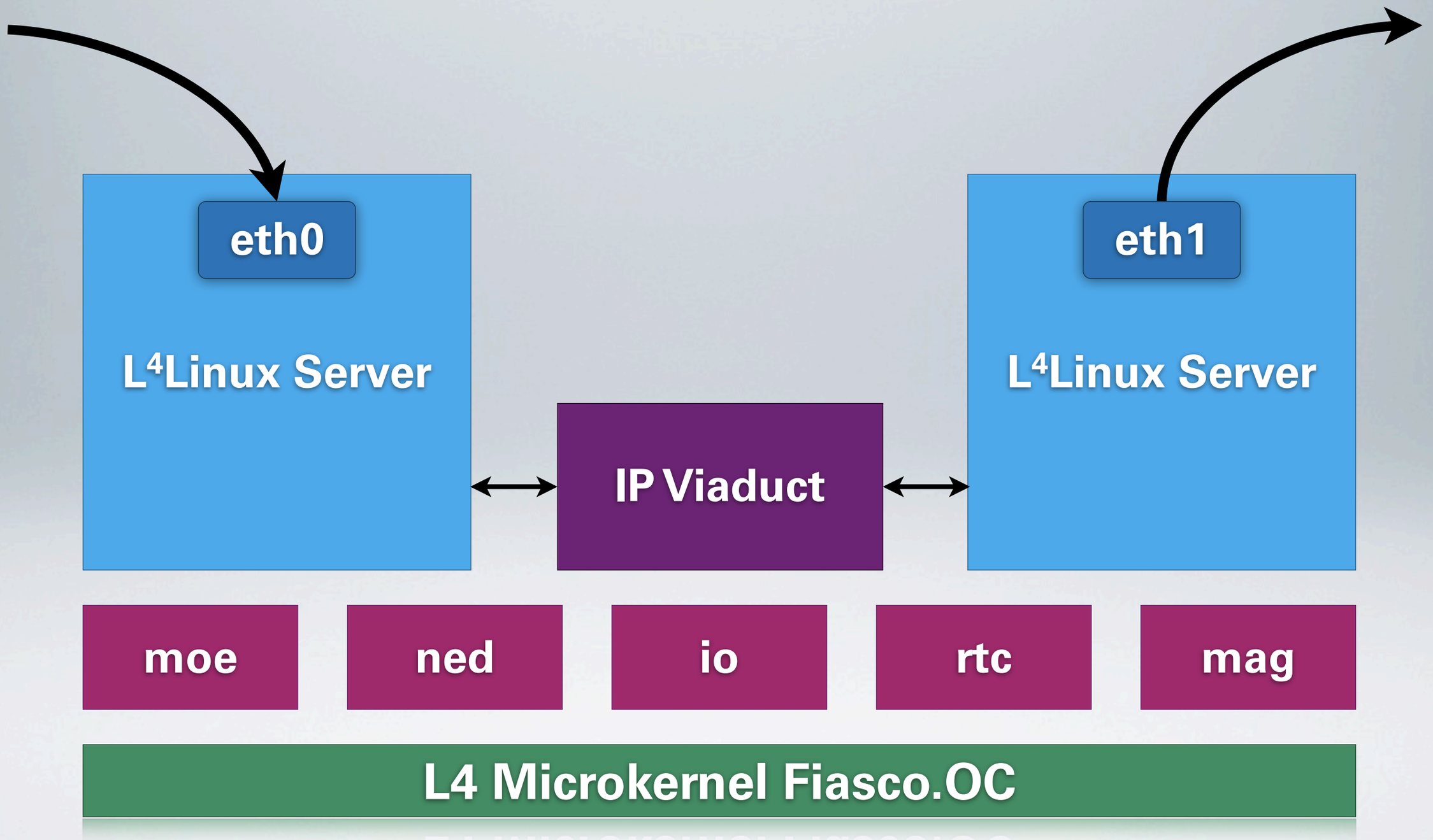

Helper App
X11
L4Linux Server
Security-Sensitve Application
Secure Application Core
moe
ned
io
rtc
mag
L4 Microkernel Fiasco.OC

CASE STUDY

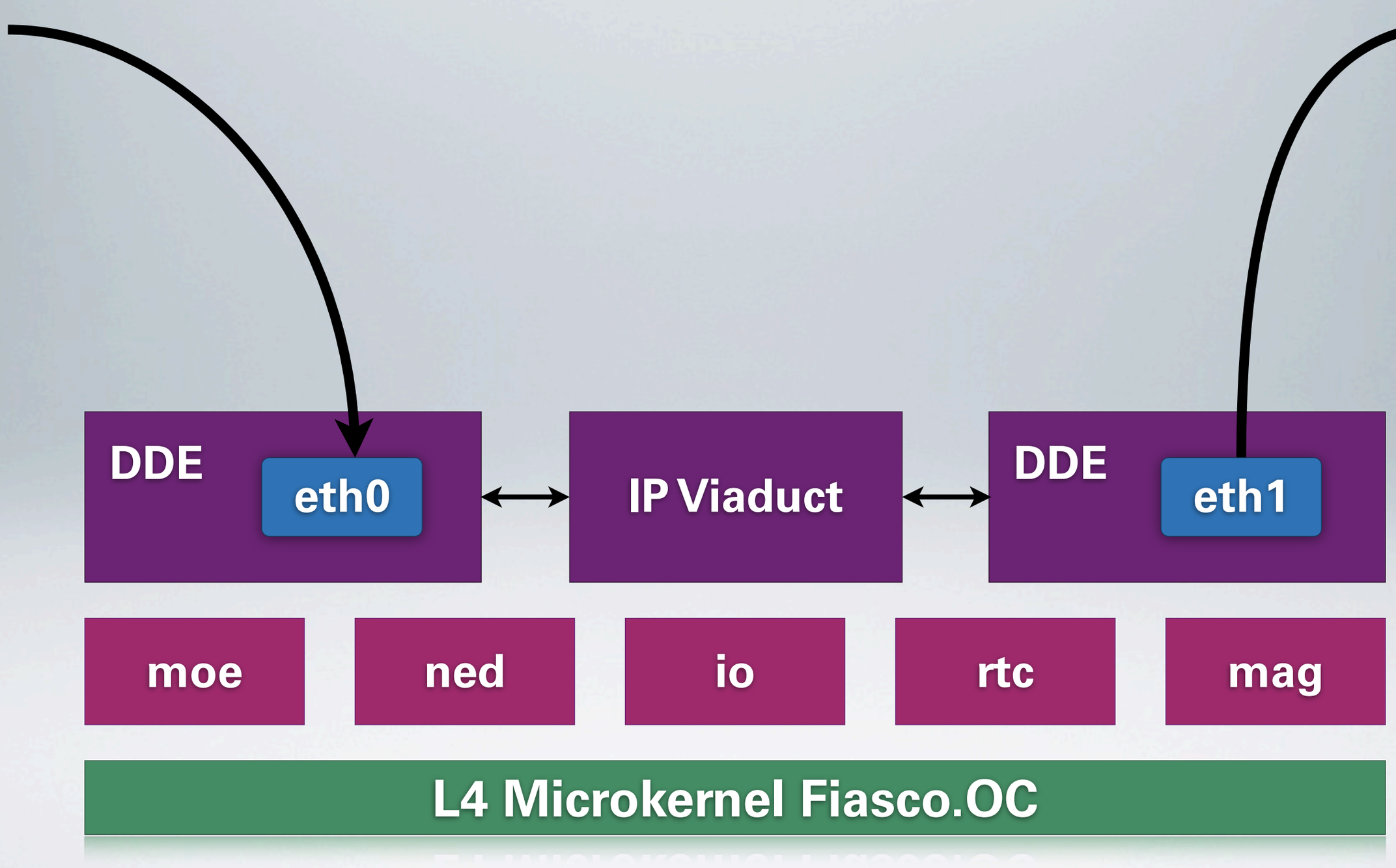
Micro-SINA VPN Box

- security goals:
  - connect sets of trusted machines
  - ensure confidentiality and integrity
- non goal:
  - availability

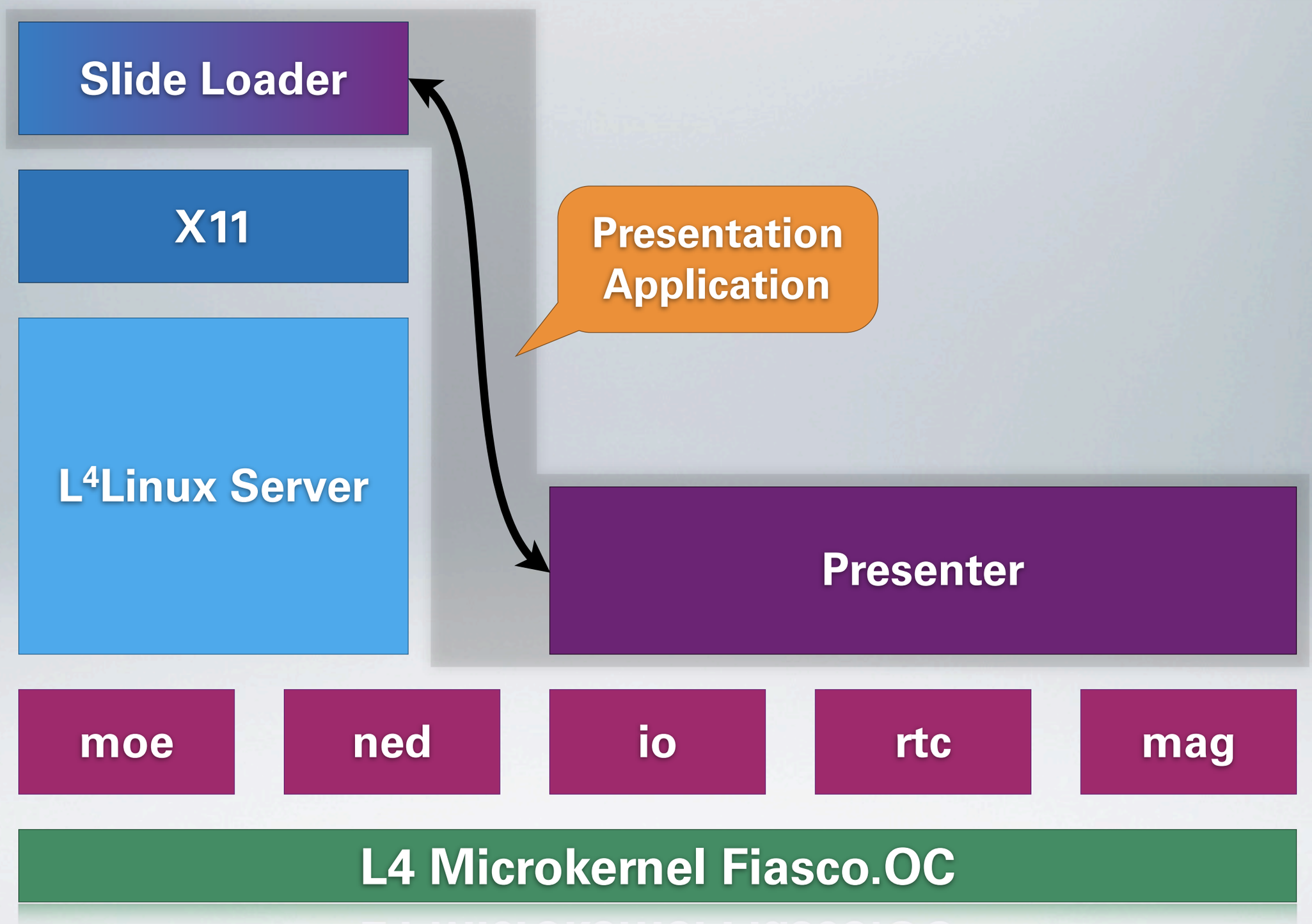
USE CASES


eth0
eth1
L4Linux Server
IP Viaduct
L4Linux Server
moe
ned
io
rtc
mag
L4 Microkernel Fiasco.OC

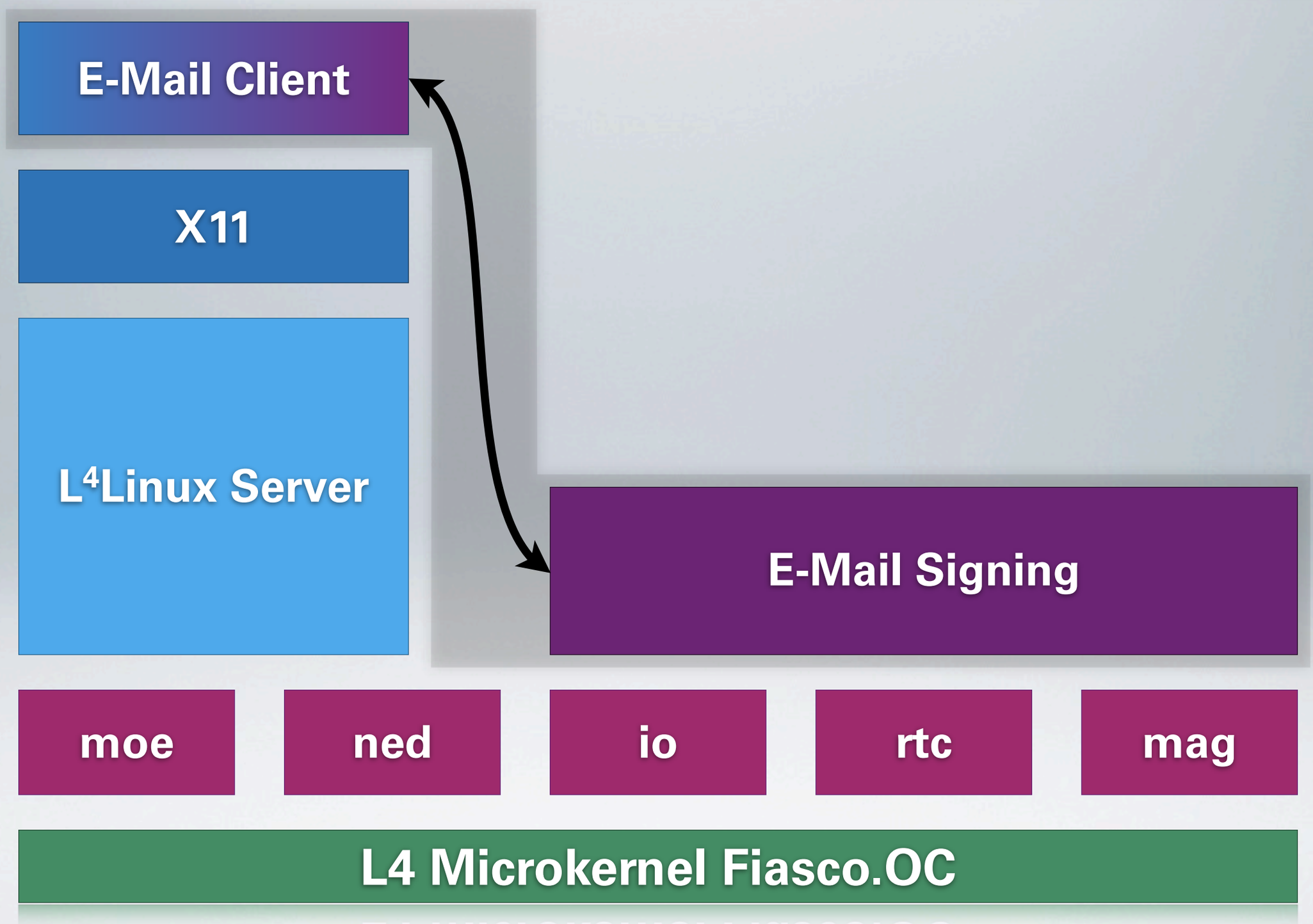
USE CASES


DDE
eth0
IP Viaduct
DDE
eth1
moe
ned
io
rtc
mag
L4 Microkernel Fiasco.OC

HYBRID APPS

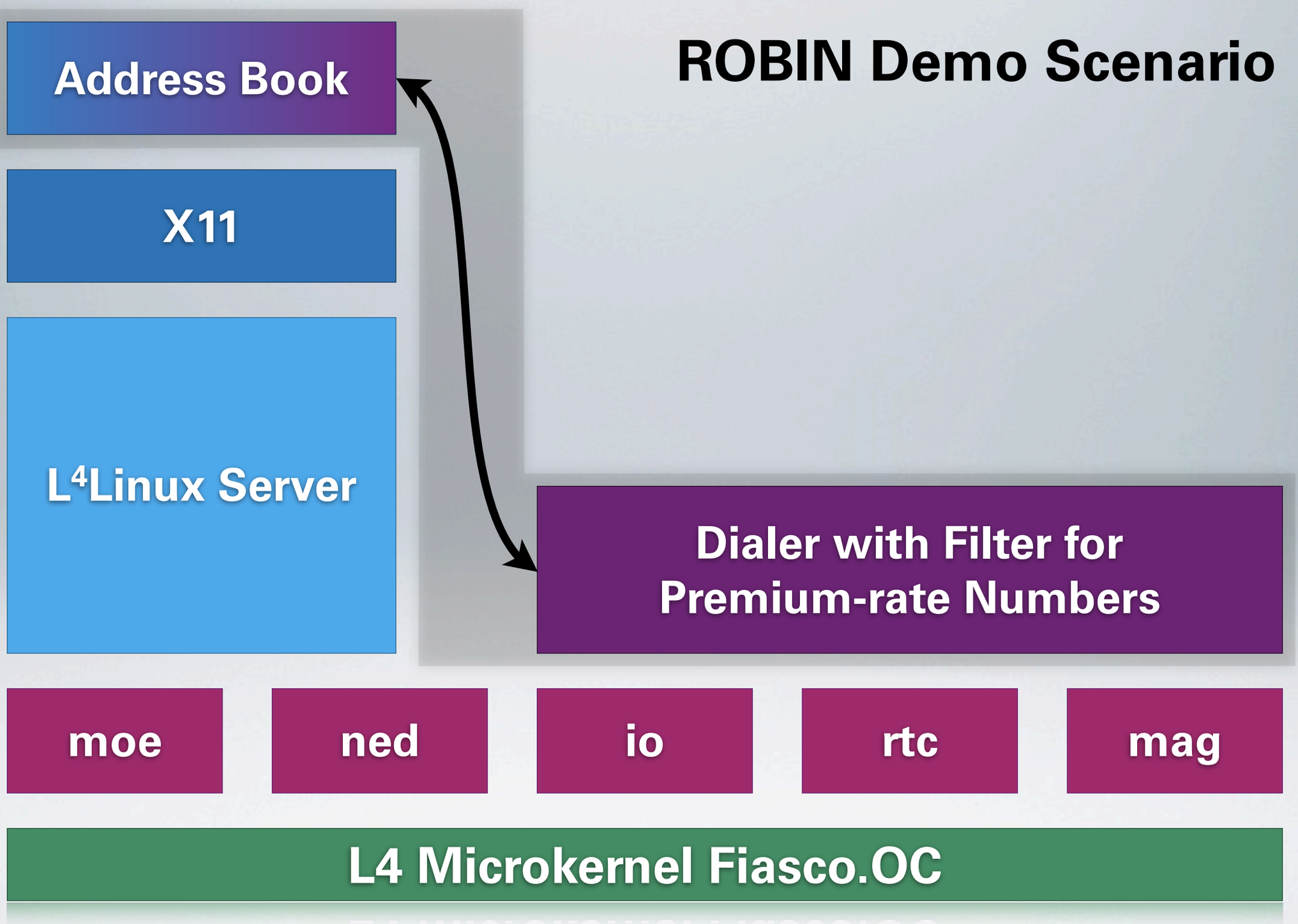

Slide Loader
X11
L4Linux Server
Presentation Application
Presenter
moe
ned
io
rtc
mag
L4 Microkernel Fiasco.OC

HYBRID APPS


E-Mail Client
X11
L4Linux Server
E-Mail Signing
moe
ned
io
rtc
mag
L4 Microkernel Fiasco.OC

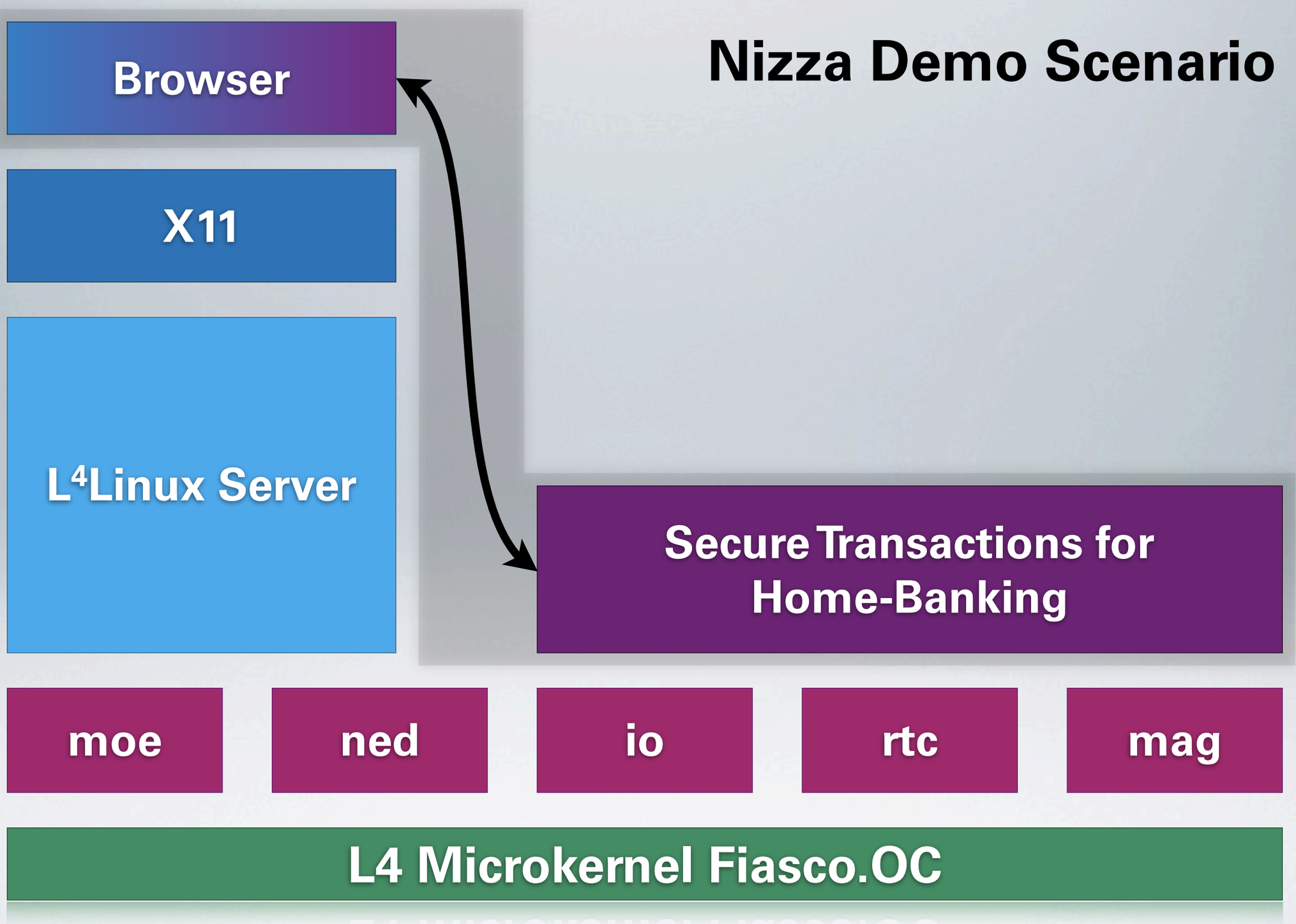
HYBRID APPS

ROBIN Demo Scenario



HYBRID APPS

Nizza Demo Scenario



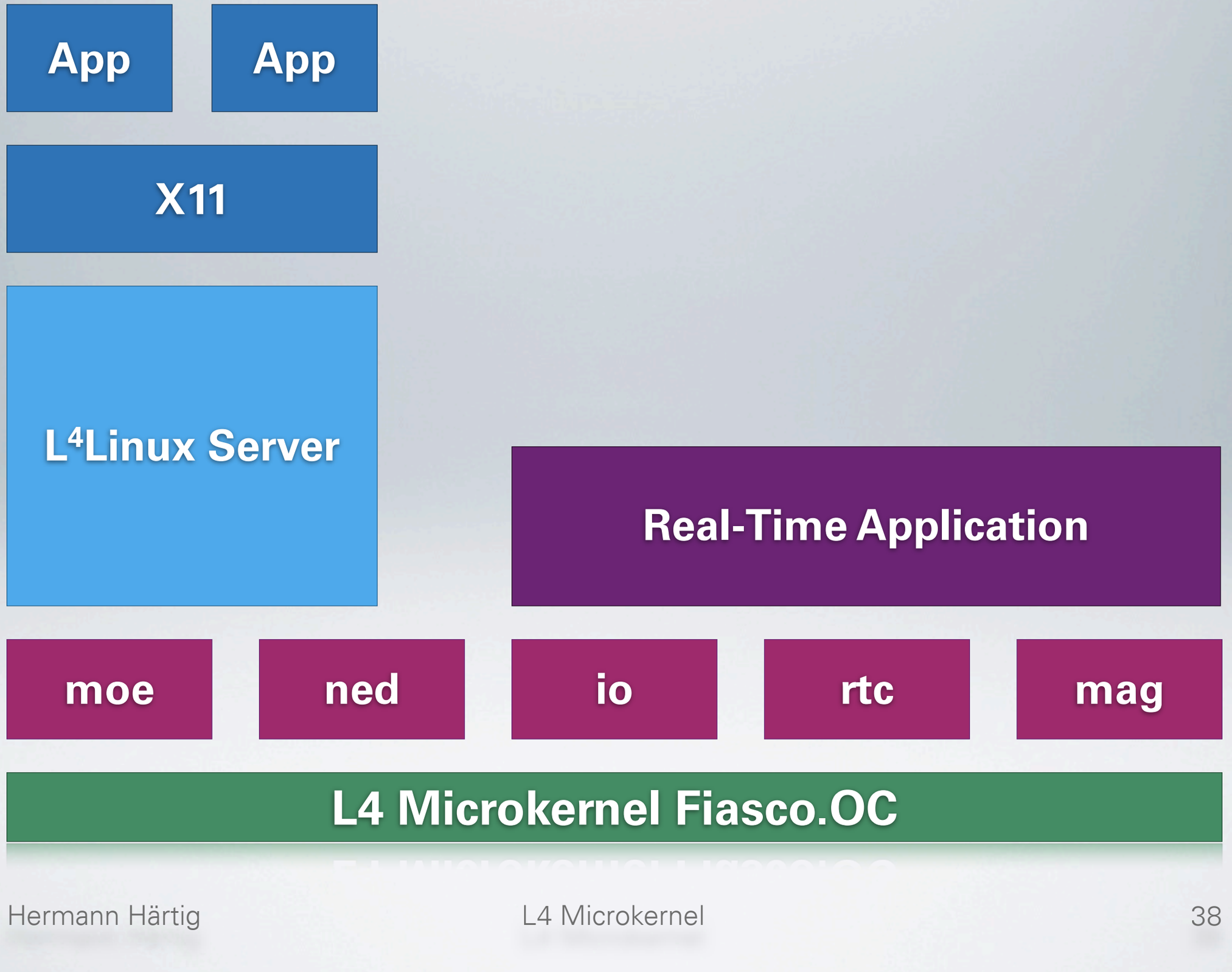
TECHNISCHE UNIVERSITÄT DRESDEN

NUMBERS

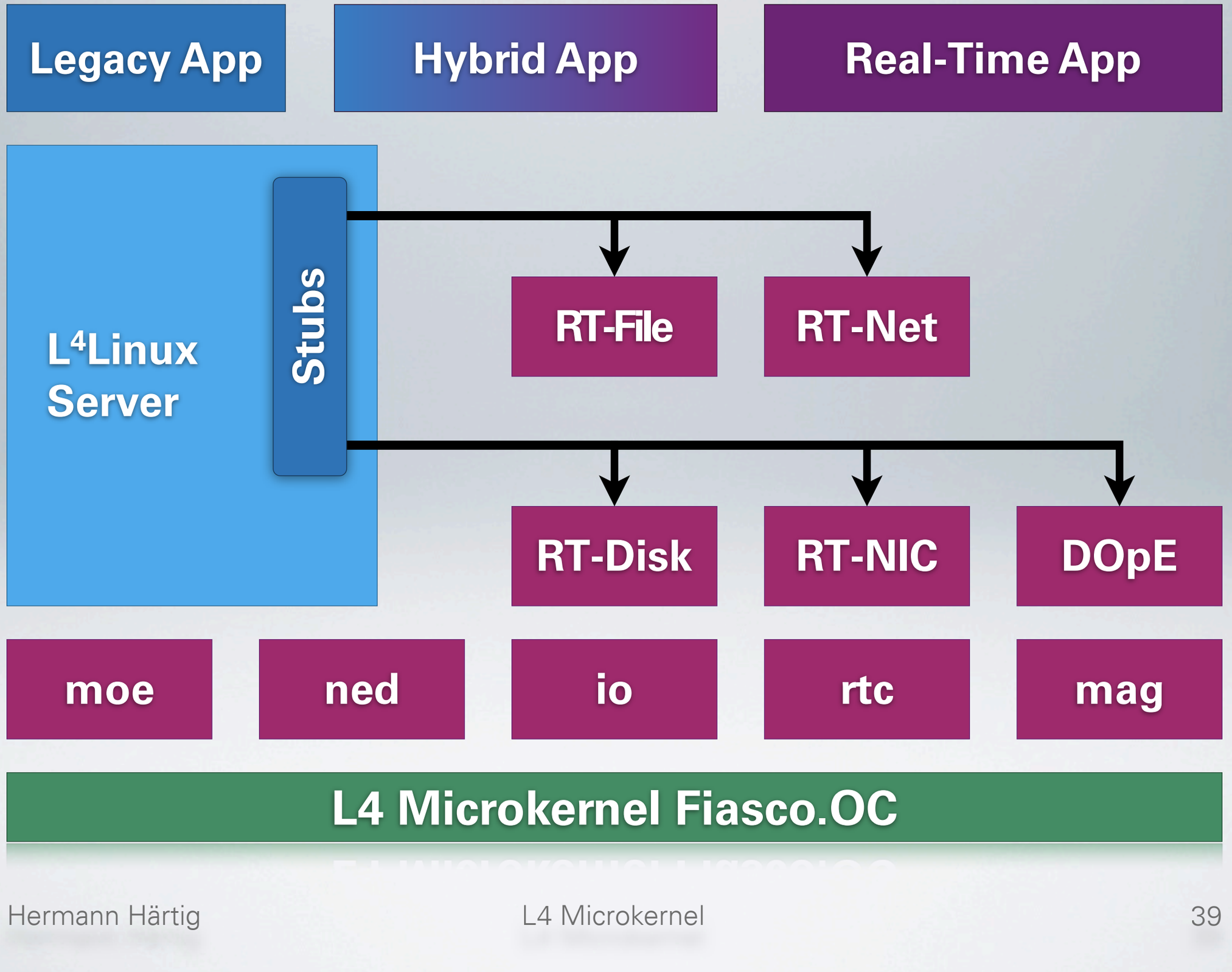
Scenario	Original		AppCore		Reduction Factor
	kLOC	kMCC	kLOC	kMCC	
e-commerce Browser	978	151	10	15	100×
VPN Gateway FreeS/WAN	155	25	74	10	2.1×
Email signer Thunderbird	250	45	54	11	4.6×
TCB Linux + X11	1485	238	100	14	14×

Reducing TCB Complexity for Security-Sensitive Applications: Three Case Studies
Lenin Singaravelu, Calton Pu, Hermann Härtig, Christian Helmuth, EuroSys 2006

REAL-TIME


App
App
X11
L4Linux Server
Real-Time Application
moe
ned
io
rtc
mag
L4 Microkernel Fiasco.OC

REAL-TIME


Legacy App
Hybrid App
Real-Time App
L4Linux Server
Stubs
RT-File
RT-Net
RT-Disk
RT-NIC
DOpE
moe
ned
io
rtc
mag
L4 Microkernel Fiasco.OC

TECHNISCHE UNIVERSITÄT DRESDEN

ORTHOGONAL

Microkernel	Virtual Machine
Isolation	Rehosting

Light-Weight Microkernels

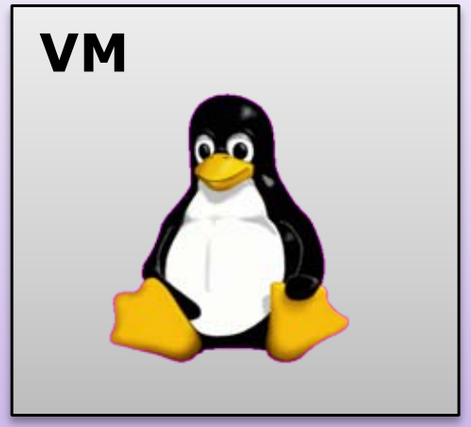
- Componentization of operating system
- Split applications
  - Critical part on microkernel
  - Uncritical on commodity OS
- Small Trusted Computing Bases

Microkernel	Virtual Machine
Isolation	Rehosting

Virtual Machines

- Provide virtual hardware interface
- Reuse of COTS operating systems and applications with no modification
- At the price of complexity

guest mode



Monolithic Hypervisor

x86 Virtualization

> 100,000 lines of code

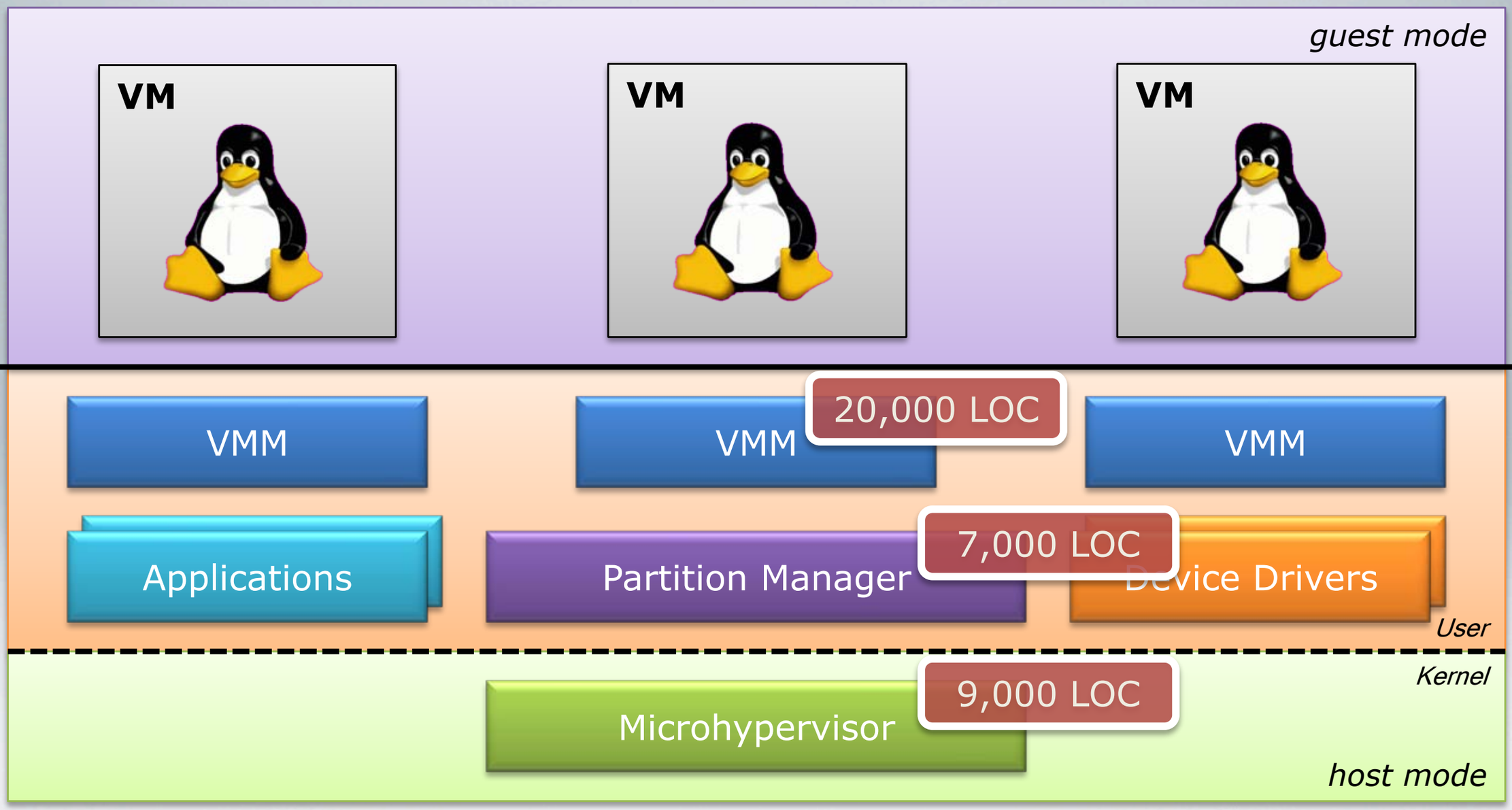
Storage

Management

Network

Device Drivers

host mode


guest mode
VM
VM
VM
VMM
VMM
20,000 LOC
VMM
Applications
Partition Manager
7,000 LOC
Device Drivers
User
Kernel
Microhypervisor
9,000 LOC
host mode

SHORT HISTORY

- Eumel, L3, BirliX
- 1995 first version of L4
- 1997 L4Linux, the first major application
- 1998 Fiasco, the first HLL implementation
- PikeOS: first commercial derivative
- real-time systems based on Fiasco
- Pistachio (Uni Karlsruhe)

SHORT HISTORY

- 2000 First commercial usage (Fiasco on a DRM product)
- 2005 Qualcomm adopts L4 kernel from NICTA (Pistachio derivative)
- 2009 Full formal verification of implementation in Haskell (NICTA)
- 2009 Fiasco.OC, L4Re
- 2009 NOVA

TECHNISCHE UNIVERSITÄT DRESDEN

L4 KERNEL ABSTRACTIONS

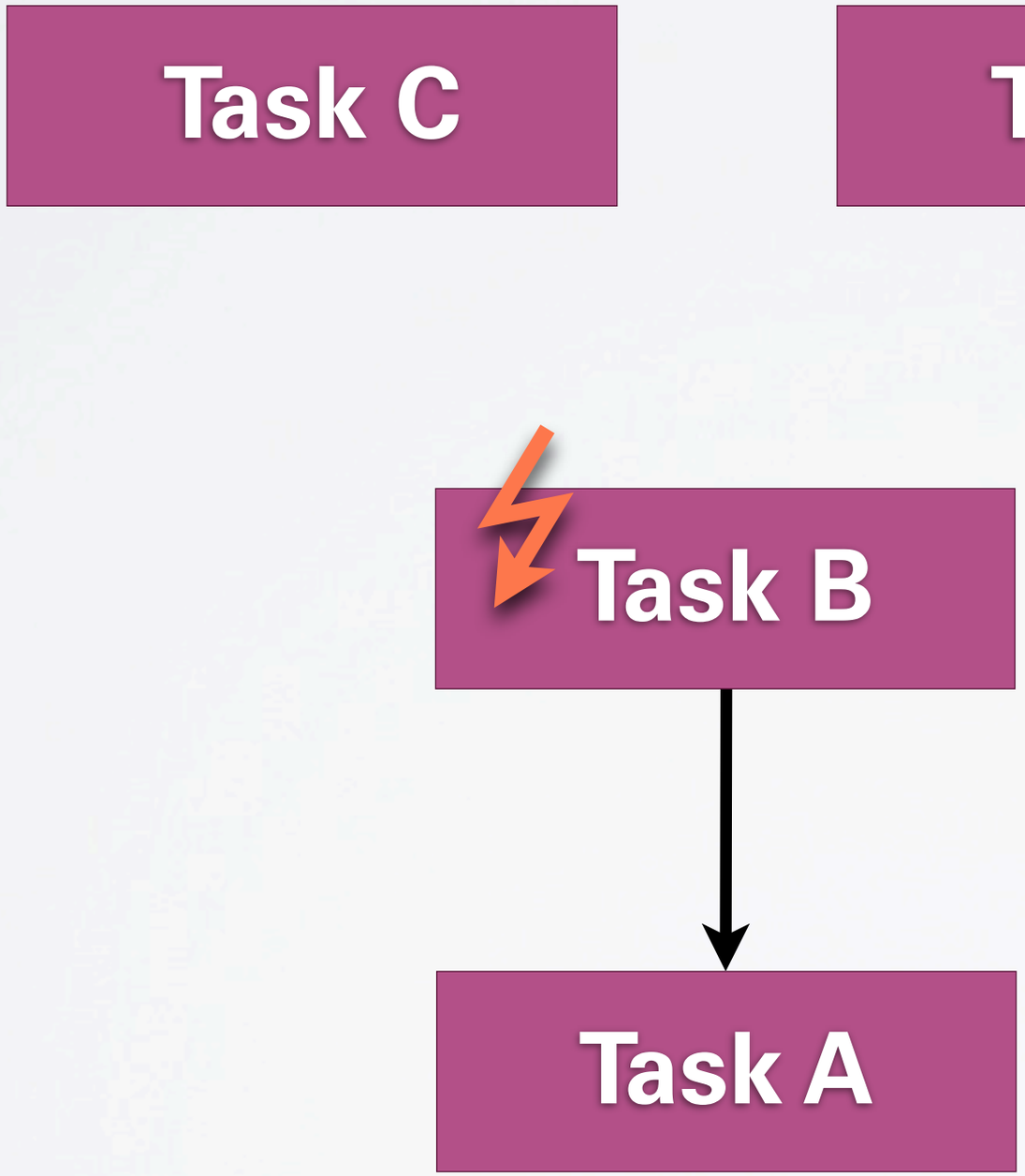
ABSTRACTIONS

- Task (Address space: memory & capabilities)
- Thread
- Communication (IPC)

MAIN MEMORY

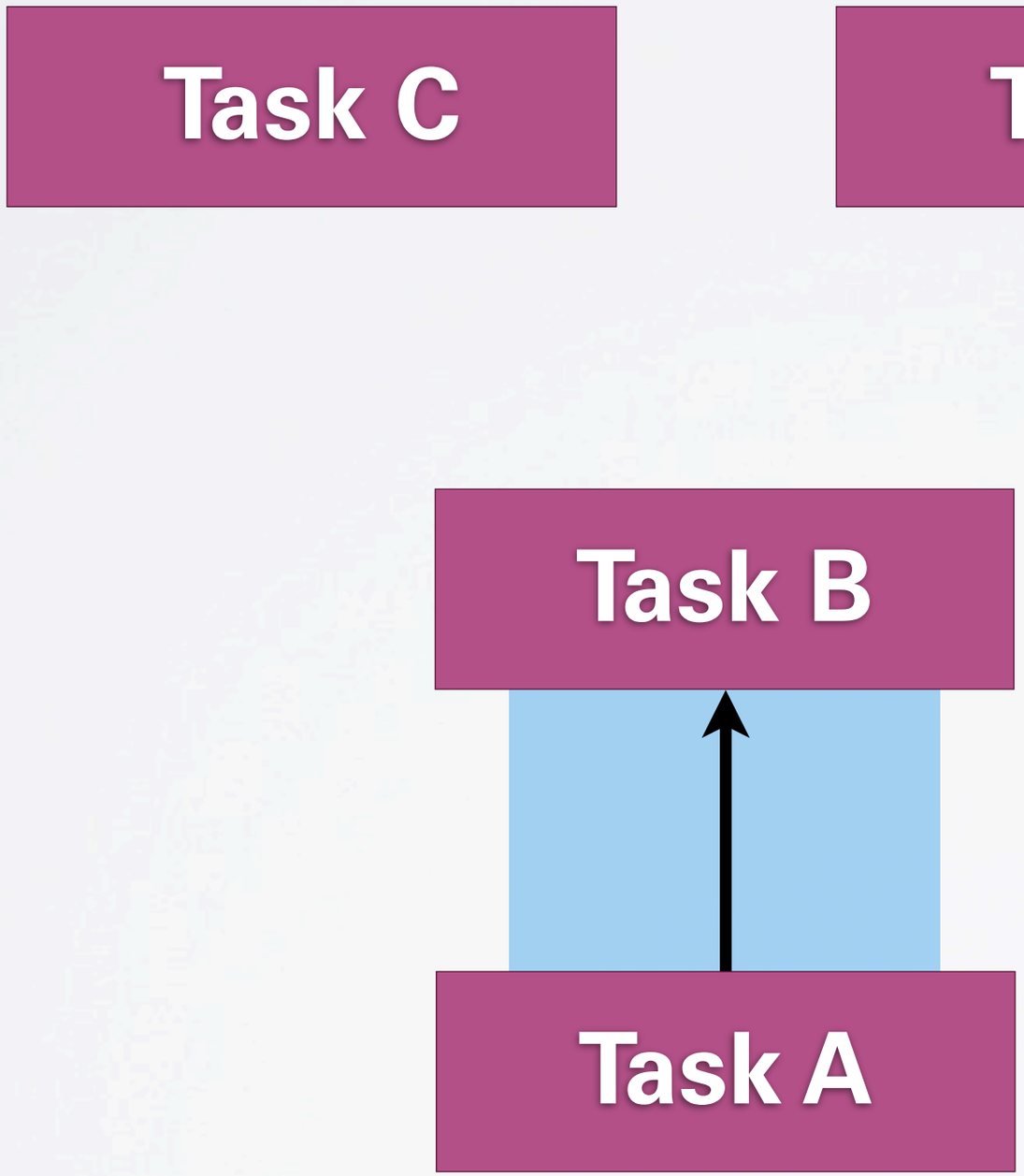
- Management of Physical Memory at user level?
- only kernel can access page tables?

MAIN MEMORY



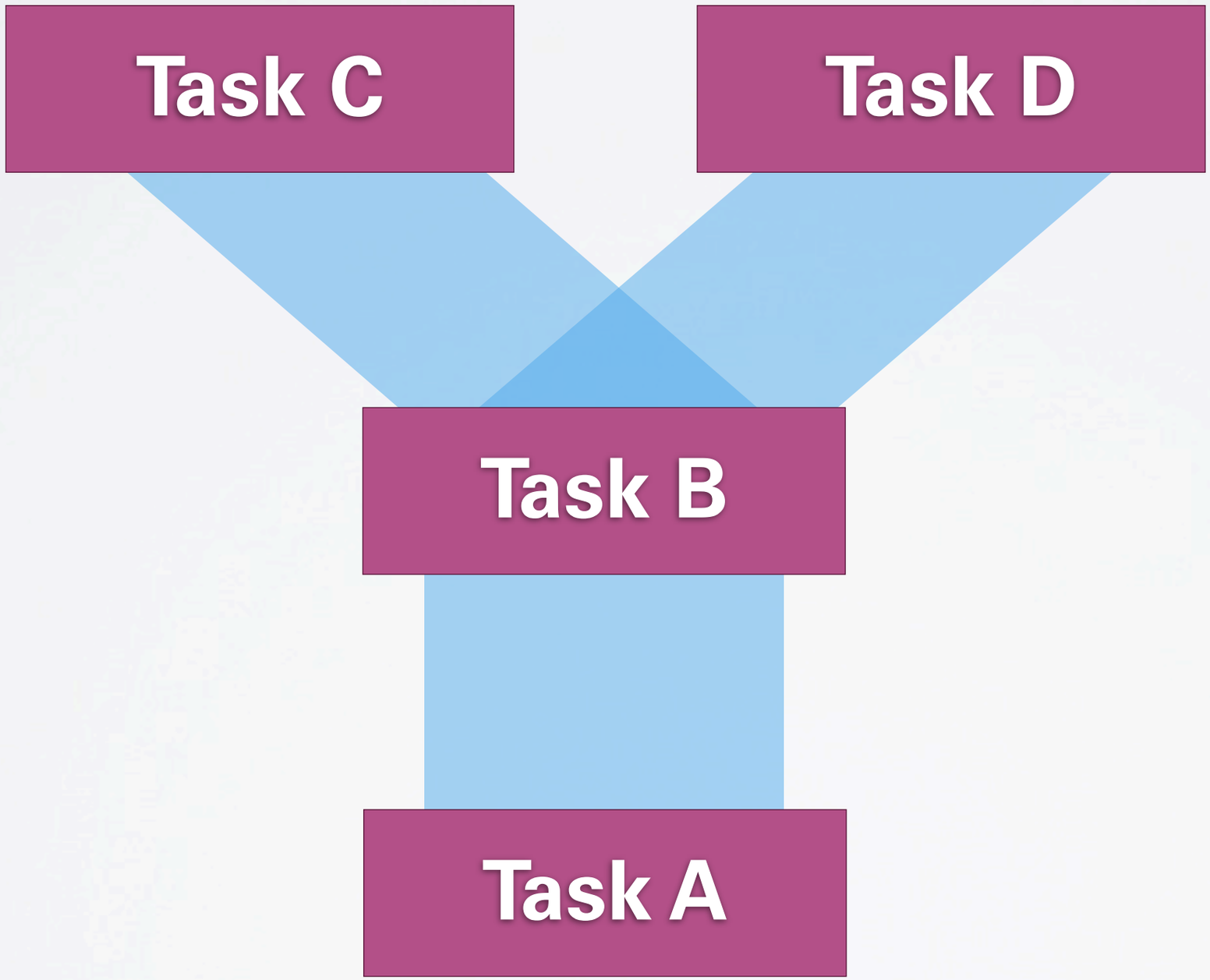
Page Fault transformed into message to handler

MAIN MEMORY

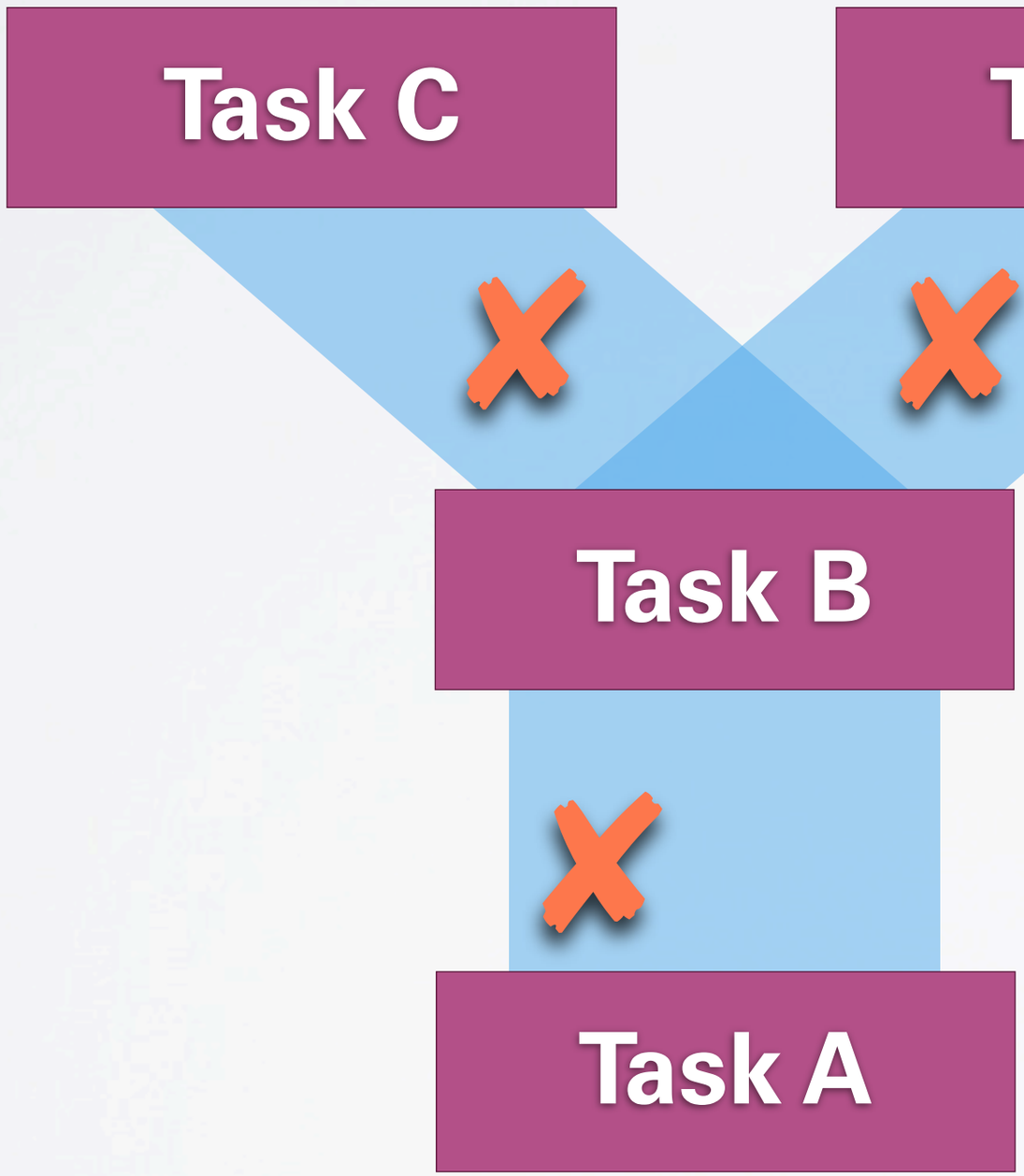


Handler returns message with PTE as payload, kernel adds to address space

MAIN MEMORY



MAIN MEMORY



revoke:
kernel maintains data structure for revoking

USES OF IPC

- data
- exceptions
- interrupts
- memory references (page fault handling)
- capabilities

NEXT ...

- Using a small kernel – Hermann Härtig
- **Capability system design – Michael Roitzsch**
- Mobile use cases – Adam Lackorzynski

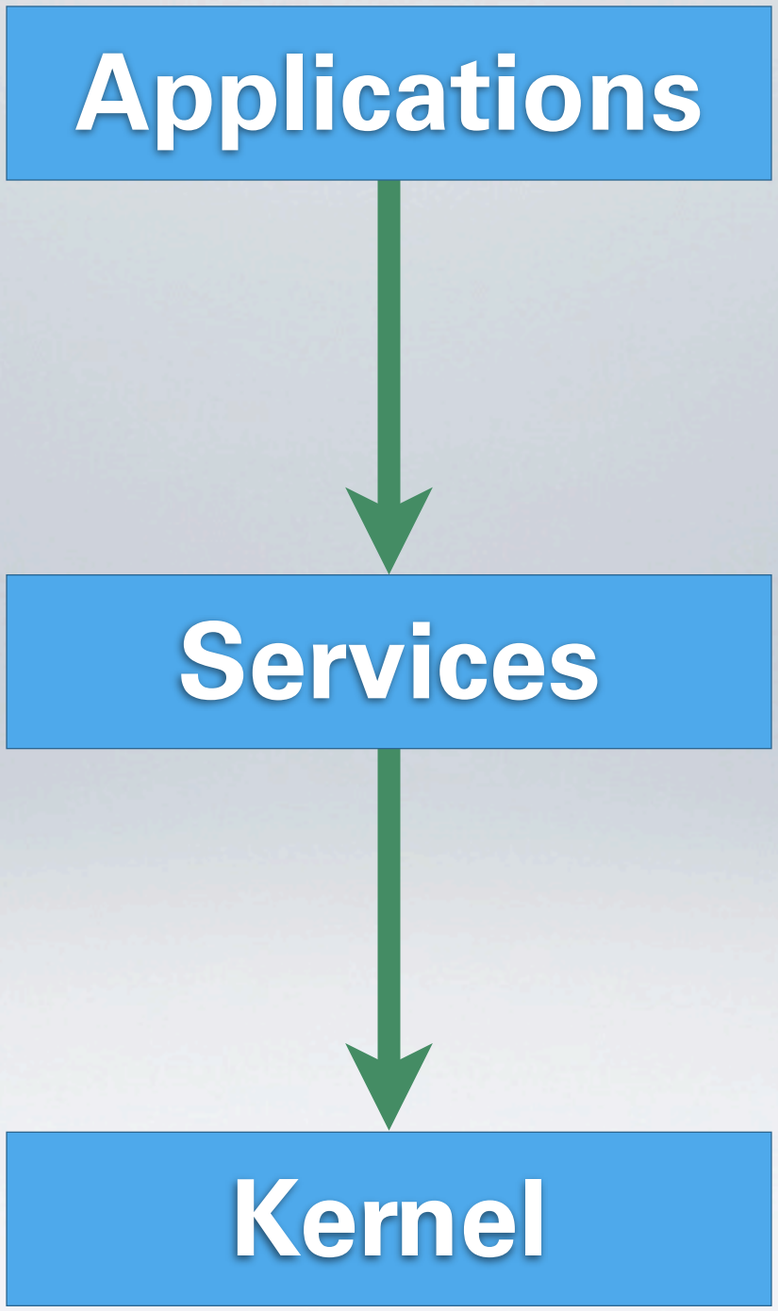
TECHNISCHE UNIVERSITÄT DRESDEN

Department of Computer Science Institute of System Architecture, Operating Systems Group

DESIGN OF A CAPABILITY SYSTEM

MICHAEL ROITZSCH

SYSTEM DESIGN

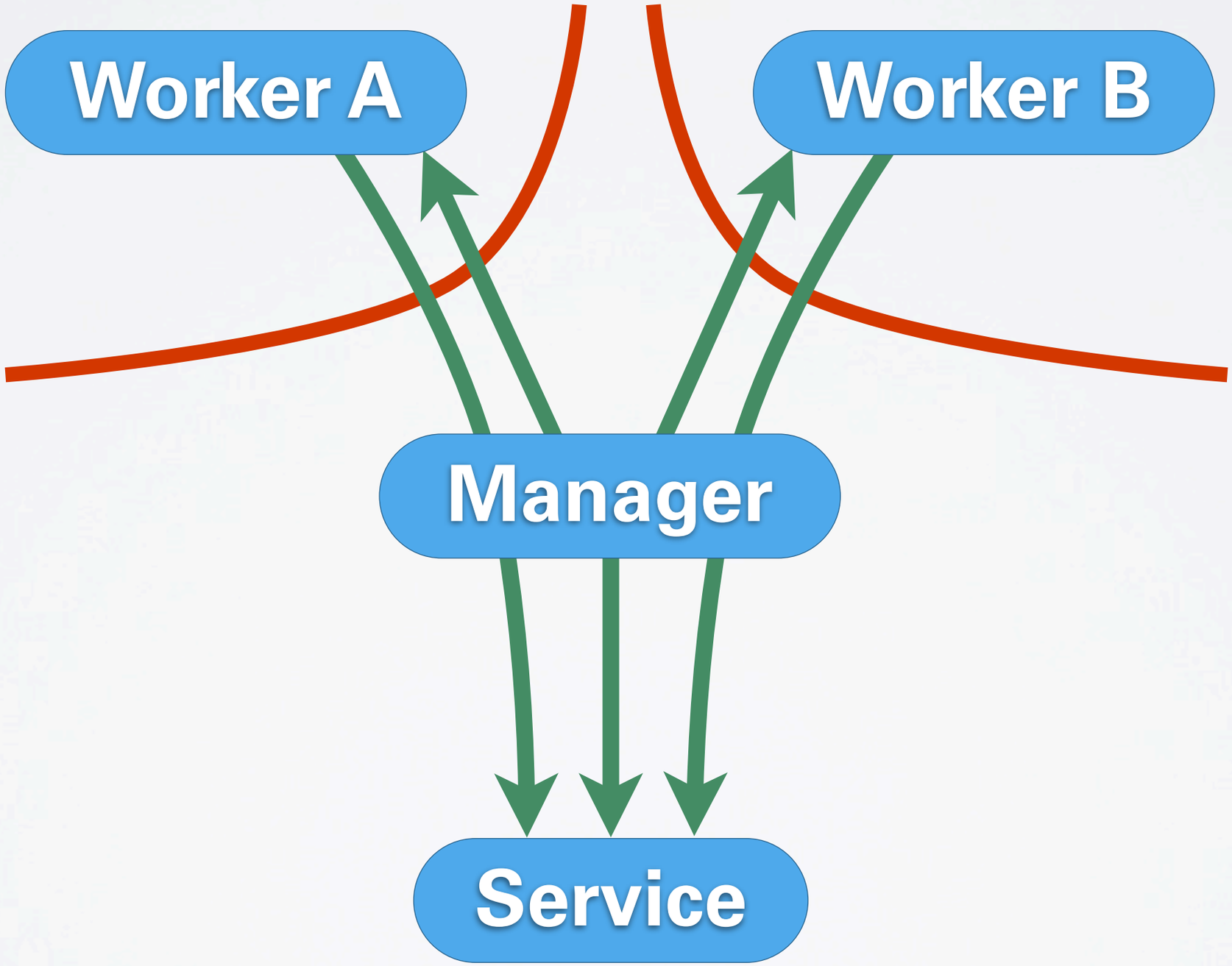


TECHNISCHE UNIVERSITÄT DRESDEN

DESIGN GOALS

- application-centric interfaces
- object-based design
- easy setup and destruction of subsystems
- object invocation by message passing
- uniform security model
- all services virtualizable
- flexible and efficient support for multicore

EXAMPLE



TECHNISCHE UNIVERSITÄT DRESDEN

GOOGLE CHROME

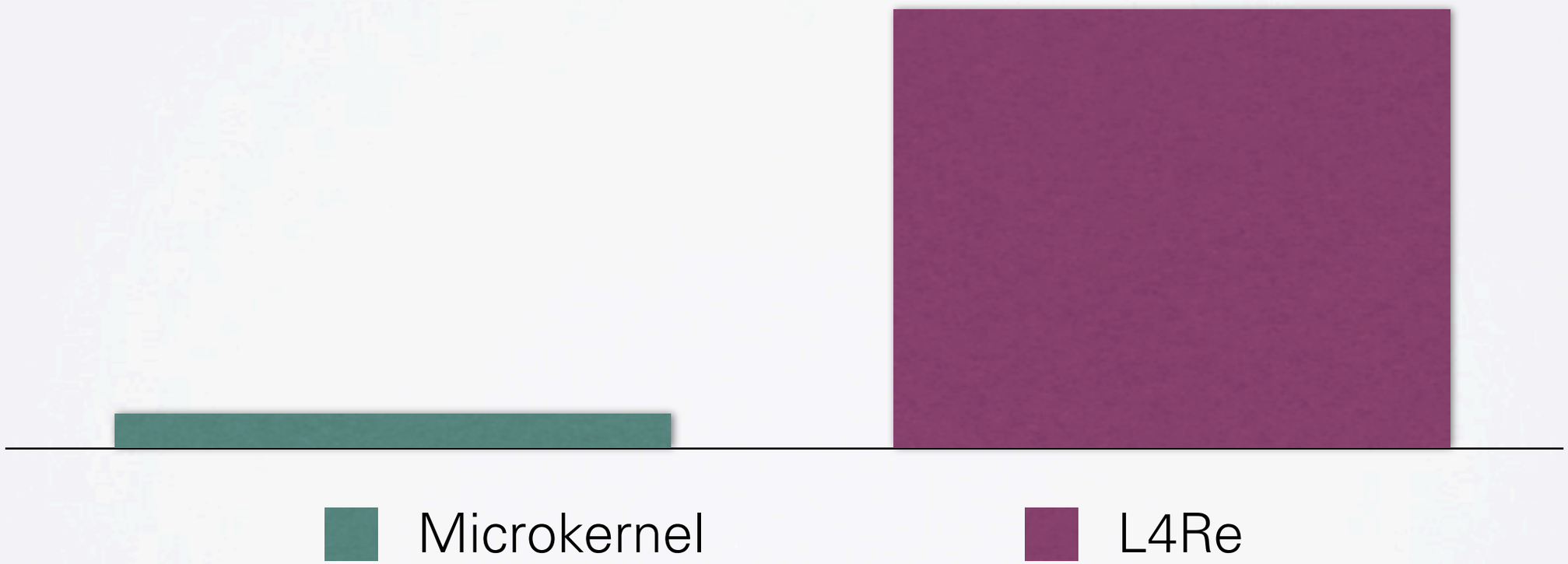
- separate processes
  - chrome parent
  - sandboxes for tabs
- implementation on Linux: glorious mix of chroot(), clone() and setuid()
- there must be a better way…

TWO WORLDS

POSIX	POLA
operations allowed by default	nothing allowed by default
some limited restrictions apply	every right must be granted
ambient authority	explicit authority

L4Re — the L4 Runtime Environment

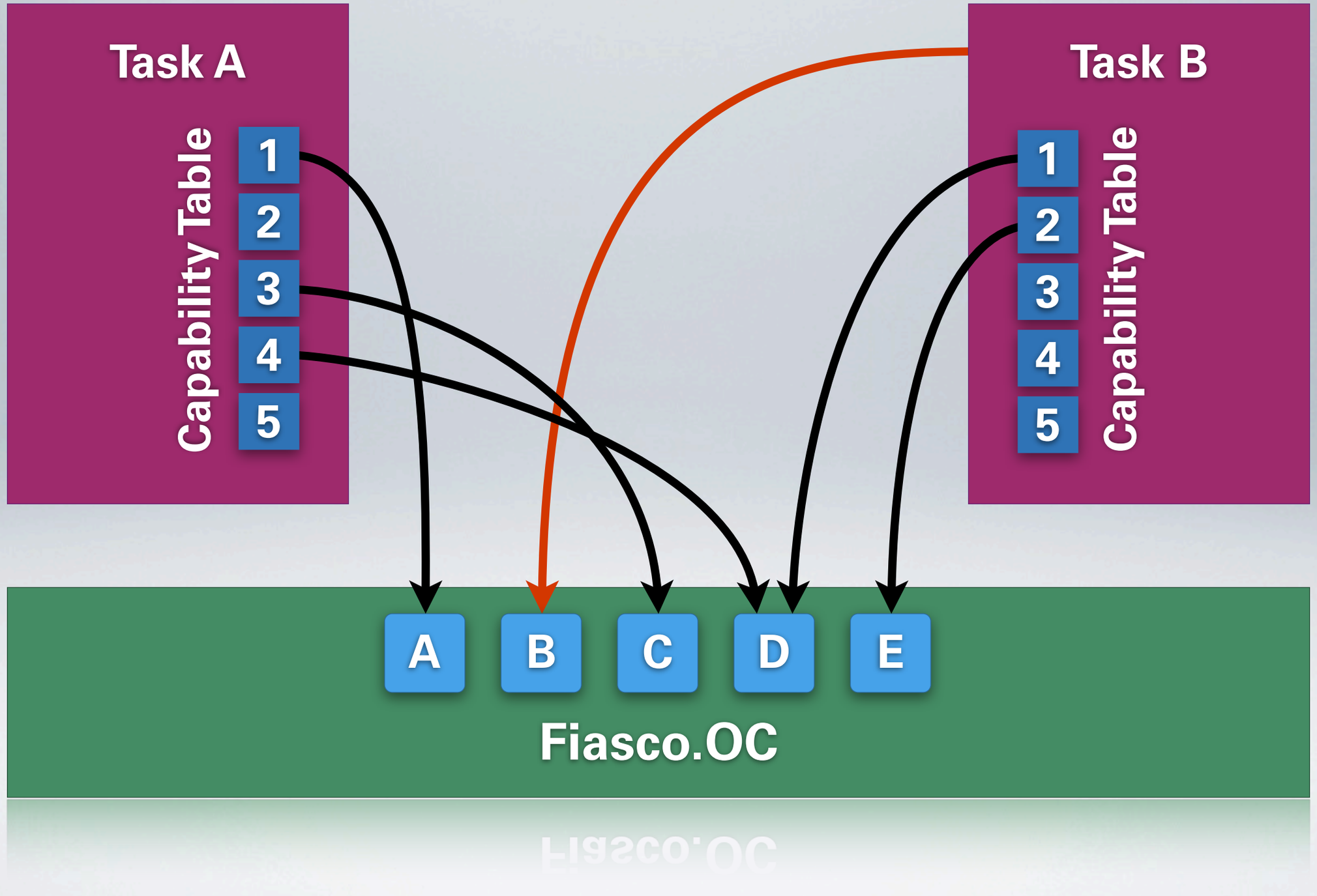
set of libraries and system services on top of the Fiasco.OC microkernel



CAPABILITIES

- Fiasco.OC and L4Re form an **object-capability** system
- actors in the system are **objects**
  - objects have local state and behavior
- **capabilities** are references to objects
  - object interaction requires a capability
  - capabilities cannot be forged

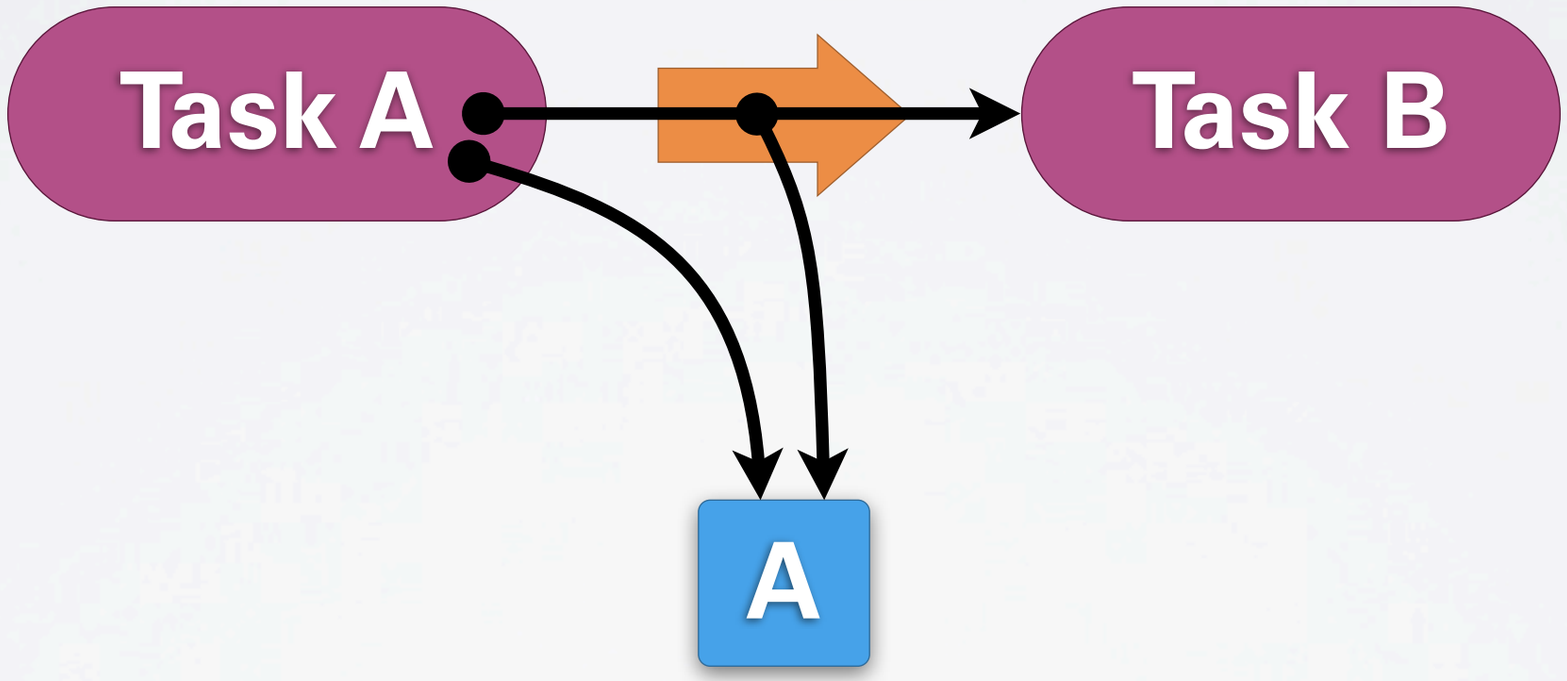
CAPABILITIES



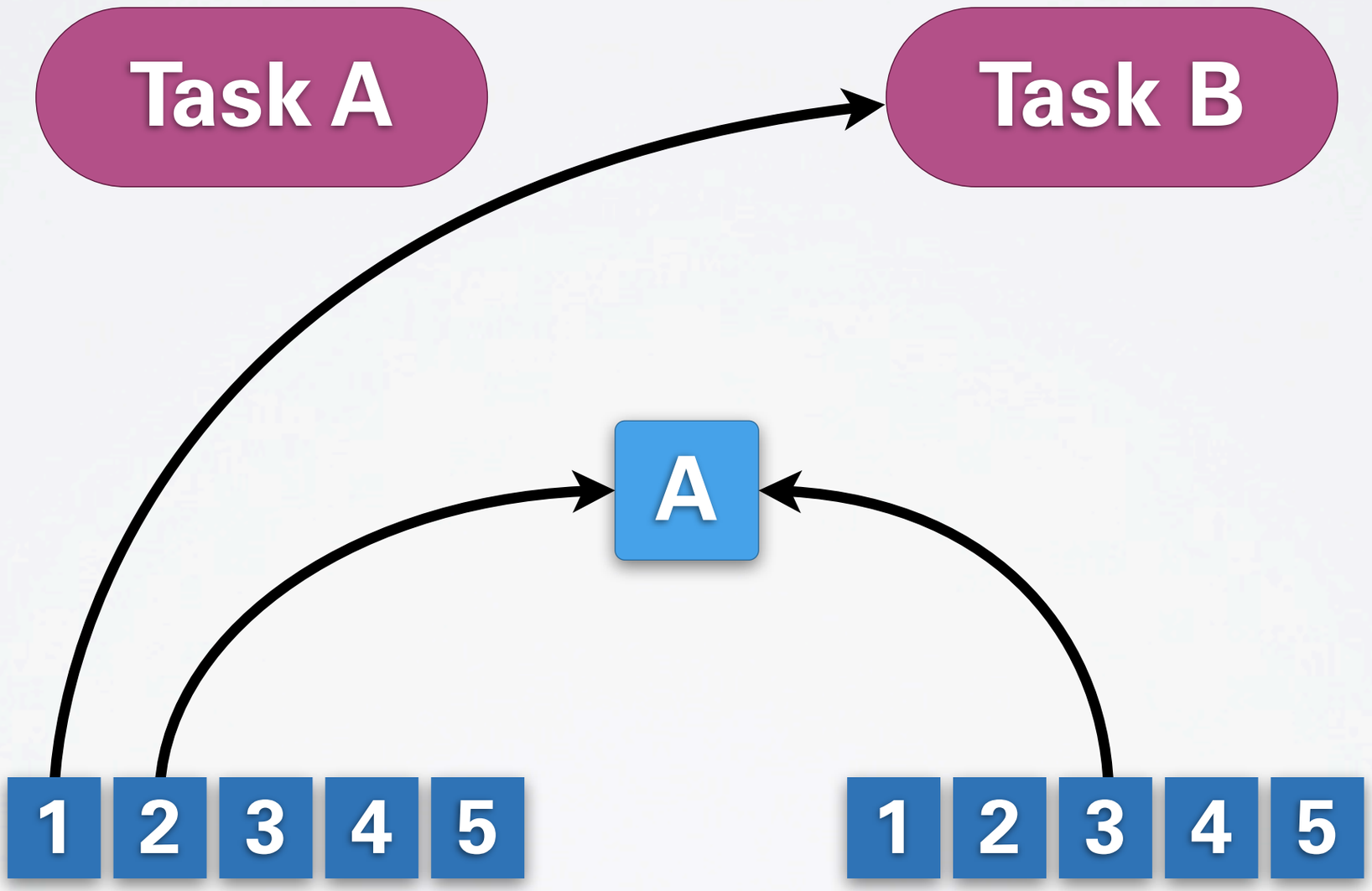
HOW TO USE?

- invocation of any object requires a capability to that object
  - no global names
- no sophisticated rights representation beyond capability ownership
  - just four rights bits on objects
- C++ language integration
- capabilities passed as message payload

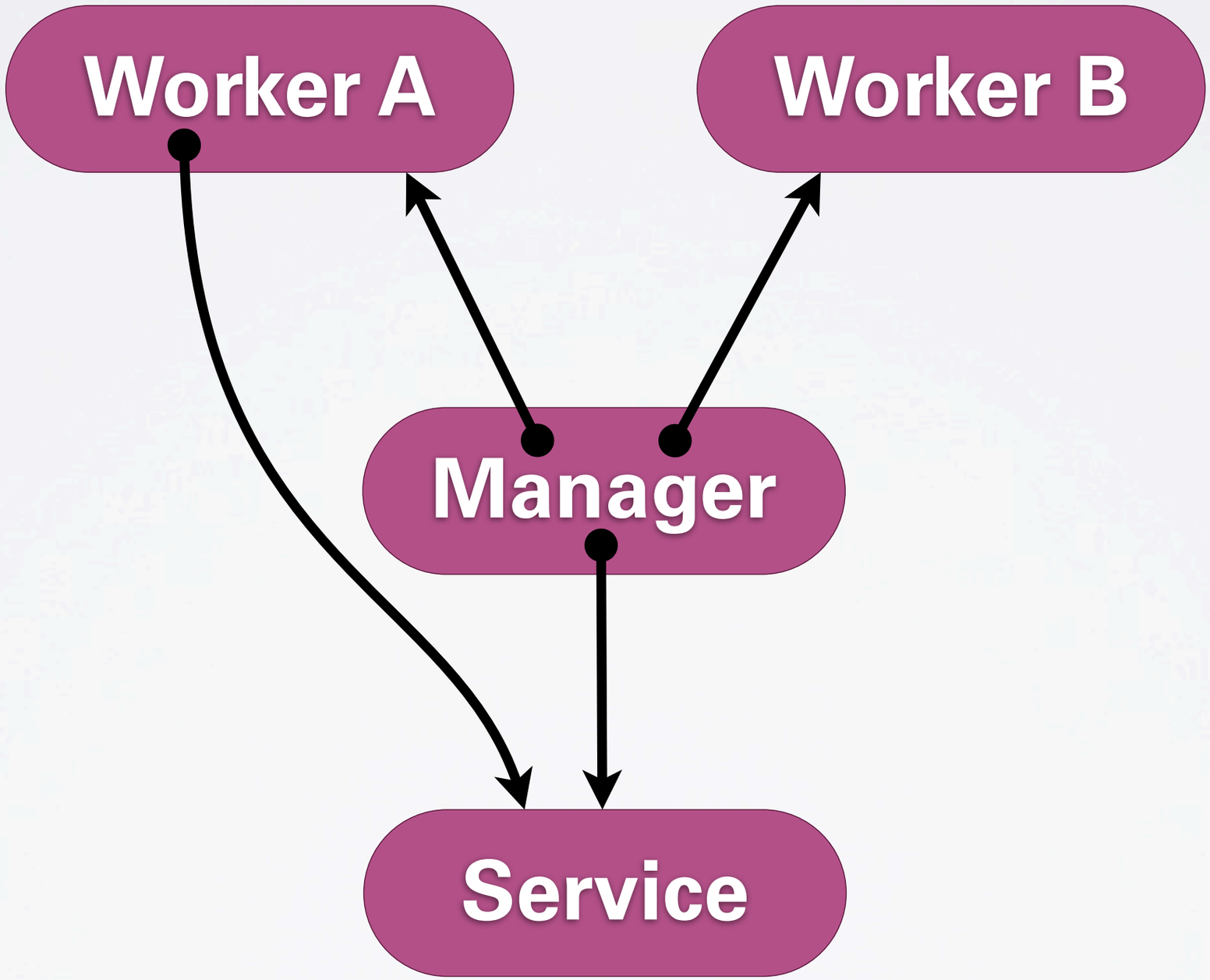
CAP TRANSFER



CAP TRANSFER



EXAMPLE

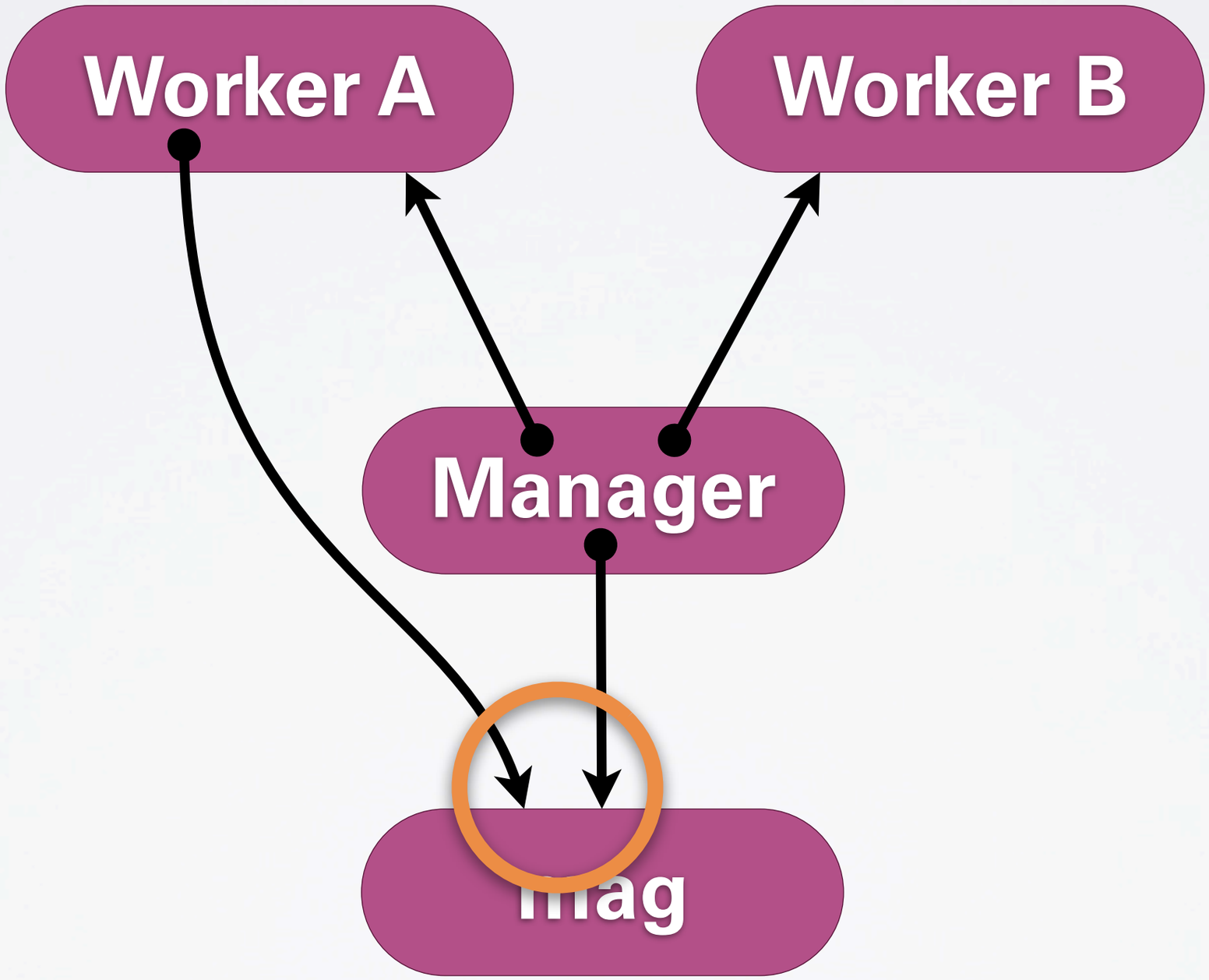


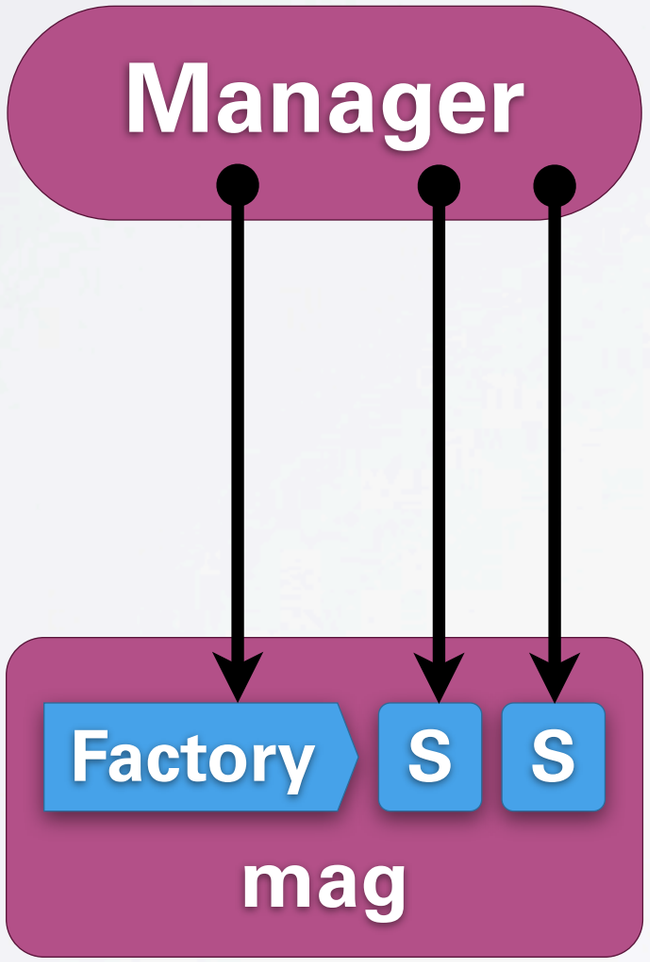
ANSWERING

How do you send an answer to a client?

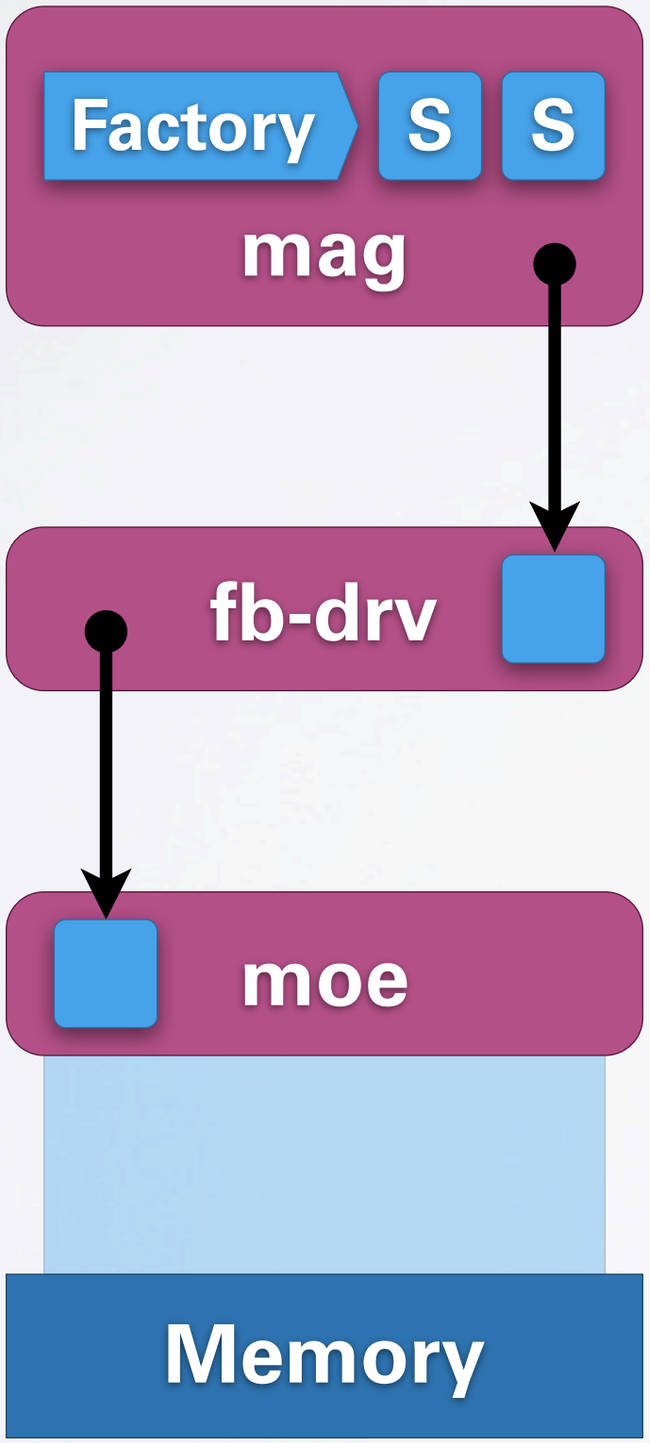
- Always include a backward capability in every request?
- Establish backward capability once and cache?
- call-return-semantics as the standard case
- **implicit reply capability**
- use-once, cannot be passed on

EXAMPLE





- **factory** for new framebuffer **sessions**
- session object
  - backing store memory
  - view: visible rectangle on the backing store
  - metadata, refresh method
- How does it appear on the screen?

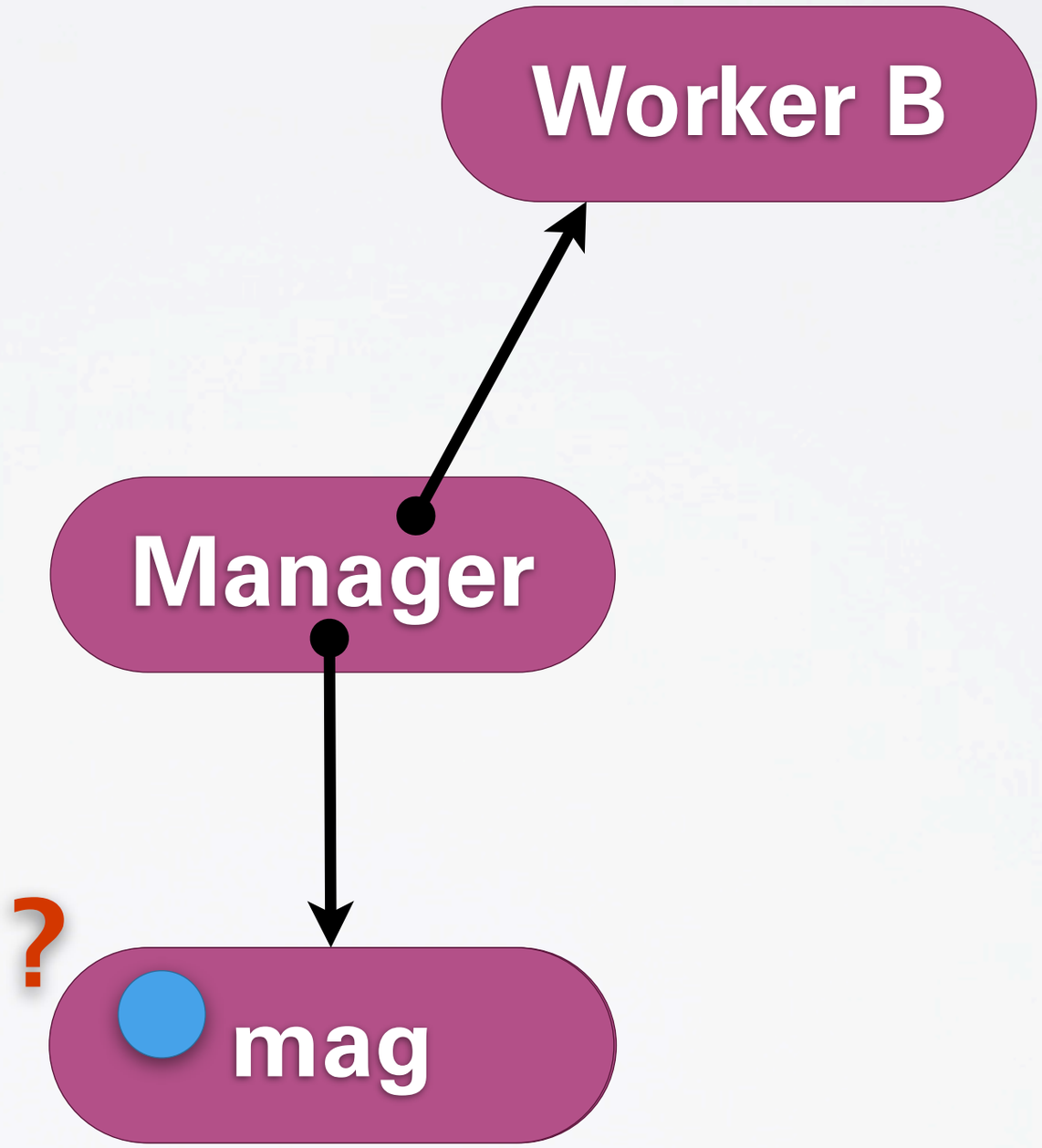


- hardware framebuffer is memory with side effect
- all memory is initially mapped to the roottask
- **framebuffer driver**
  - find framebuffer memory
  - wrap in FB-interface
- same interface as mag's

INTERFACES

- L4Re uses one interface per resource
- low-level system resources are managed by the kernel
  - CPU, memory, IRQ
  - minimal policy
- user-level servers can reimplement and augment interfaces
- **virtualizable interfaces**

EXAMPLE



Subsystem Life

- subsystems are opaque
- parents can restrict the resources
- parents cannot restrict their sub-structure

Subsystem Death

- How to deallocate resources in servers?
- notify all servers used by the subsystem?
- **garbage collection**

TECHNISCHE UNIVERSITÄT DRESDEN

CONCLUSION

- ✔ coherent per-resource interfaces
- ✔ all services provided as objects
- ✔ garbage collection for server resources
- ✔ invocation is the only system call
- ✔ object-capability system
- ✔ all interfaces can be interposed
- ✔ **see next talk**

TECHNISCHE UNIVERSITÄT DRESDEN

Department of Computer Science Institute of System Architecture, Operating Systems Group

MOBILE USE CASES

ADAM LACKORZYŃSKI

OUTLINE

- ICT-eMuCo project
- Multi-Cores and Load Balancing
- Virtualization

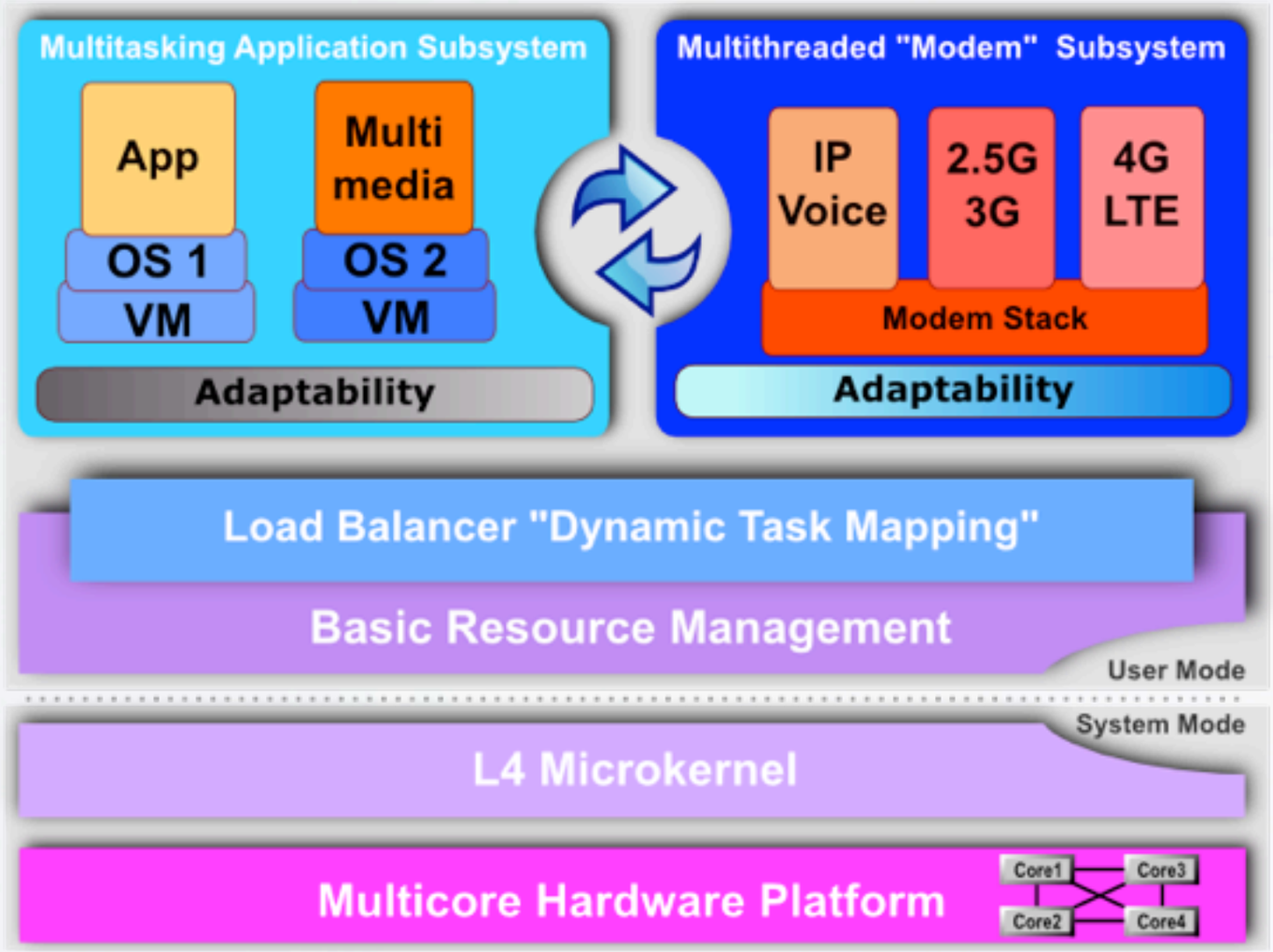
ICT-EMUCO

- **E**mbedded **Mu**lticore **Co**mputing
- FP7 Project, STREP
- Feb 2008 – Jan 2010

- Partners:
  - ARM, Infineon, Ruhr-Uni Bochum, IBM, Uni of Timisoara, Uni York, TU-Dresden

TECHNISCHE UNIVERSITÄT DRESDEN

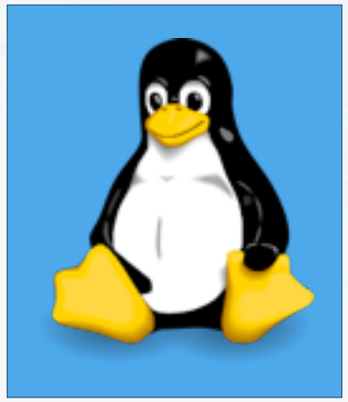
ARCHITECTURE



- Microkernel based system
- ARM11MPCore
  - 4 cores
- Modem Stack
- App-OS

THE EMUCO OS

- Isolation
- Secure communication
- Timing properties
- Multi-core capable
- Power Management
- Embedded systems
- Flexible & usable



OUTLINE

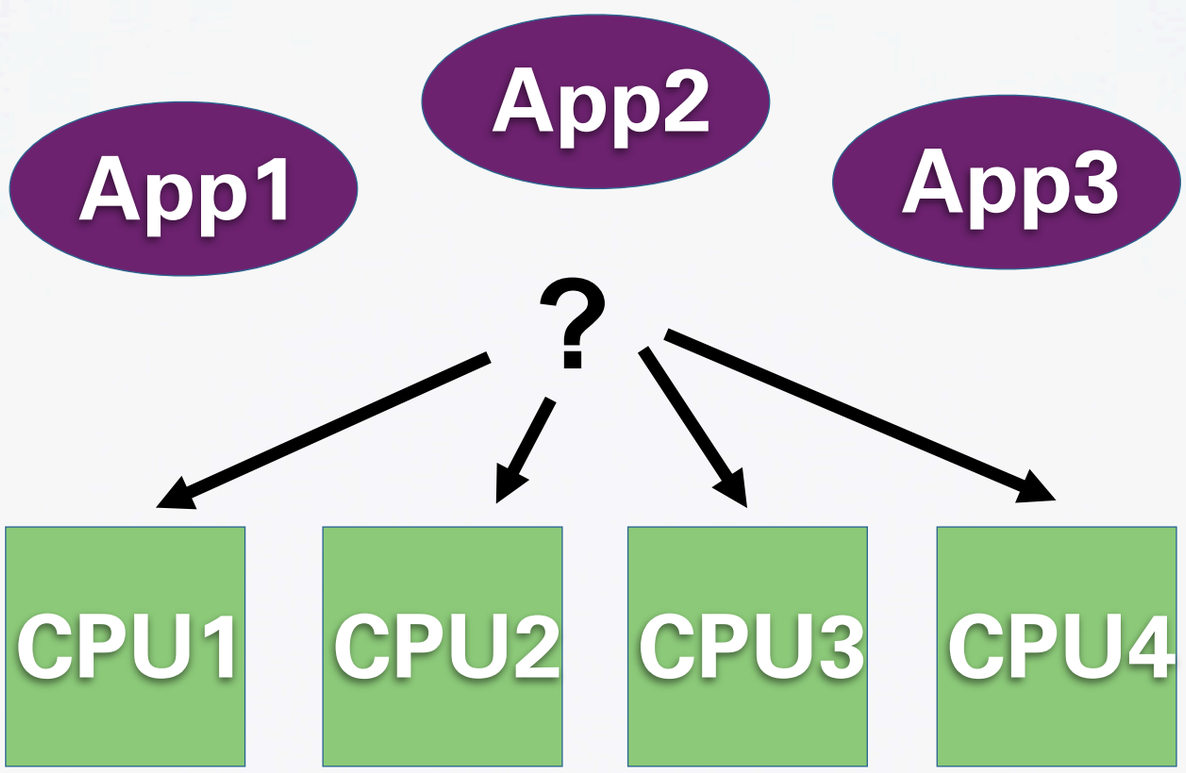
- ICT-eMuCo project
- Multi-Cores and Load Balancing
- Virtualization

MULTI-CORES

- Shared memory multi-processor systems
- Model:
  - Cross CPU tasks
  - Cross CPU notifications
  - Cross CPU IPC
  - Shared memory
  - Local scheduling
    - Fixed-prio, round-robin scheduler on each core

How to Distribute Work?

- Kernel provides migration mechanism
- No automatic migration, decision is policy

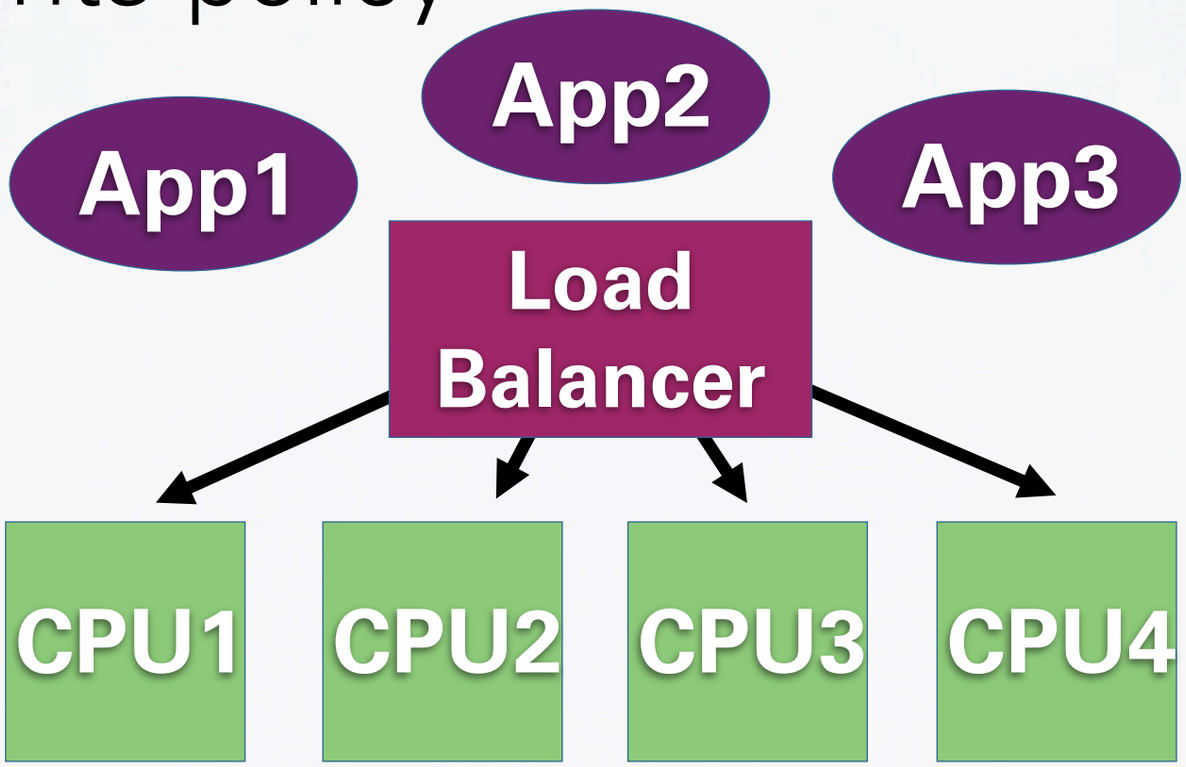


L4::SCHEDULER

- L4::Scheduler interface
  - Run/Migrate a thread with parameters
    - Priority, CPU set, budget, ...
    - `scheduler.run_thread(thread, sched_param);`
- Kernel implementation
  - Singleton, has all resources (all CPUs, all CPU time)
  - Chooses one CPU from CPU set, fixed

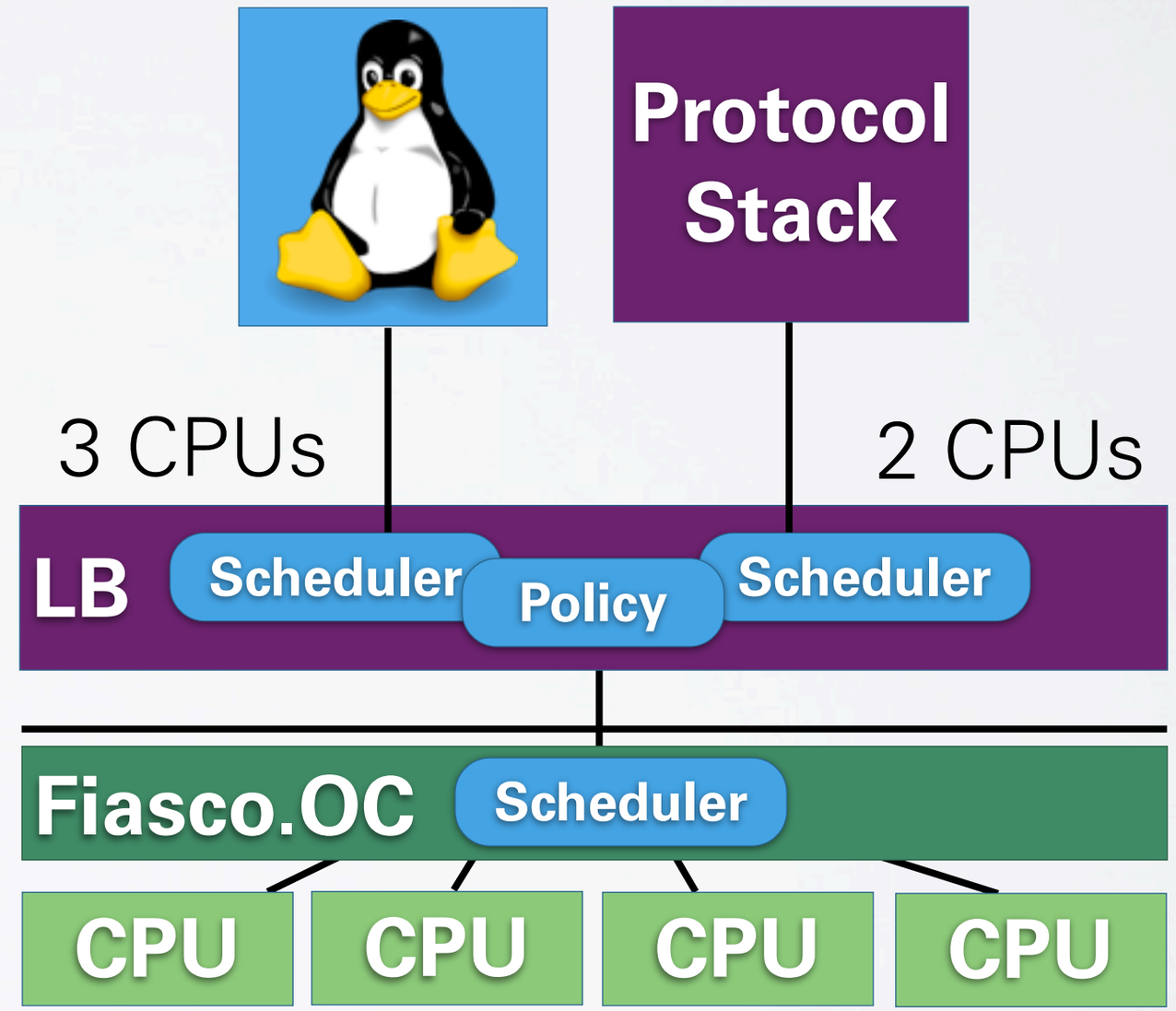
LOAD BALANCING

- Load balancer component
- Implements L4::Scheduler interface
- Hides platform details from application
- Implements policy



LOAD BALANCER

- Implements balancing strategy
  - Application specific scheduler instances
- Enforces scheduling policies
- Combines client policies



USE CASES

- Multi-threaded program
  - Distribute and balance
- Sophisticated real-time application
  - No migration by load balancer
  - Threads always on the same CPU (cache locality)
  - Threads always on different CPUs (no interference)
- Virtual machine

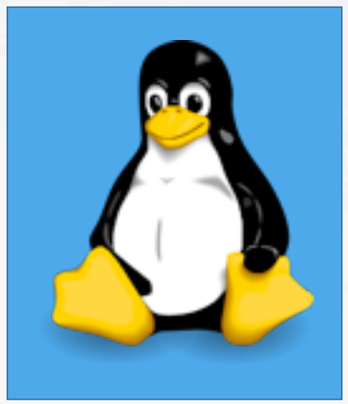
TECHNISCHE UNIVERSITÄT DRESDEN

OUTLINE

- ICT-eMuCo project
- Multi-Cores and Load Balancing
- Virtualization

VIRTUALIZATION

- Application side
  - Standard OS → Standard applications
- Integration in the system
  - Resource and device usage
- Isolation of the virtual machine
  - No disturbance of other programs



TECHNISCHE UNIVERSITÄT DRESDEN

EMUCO PLATFORM

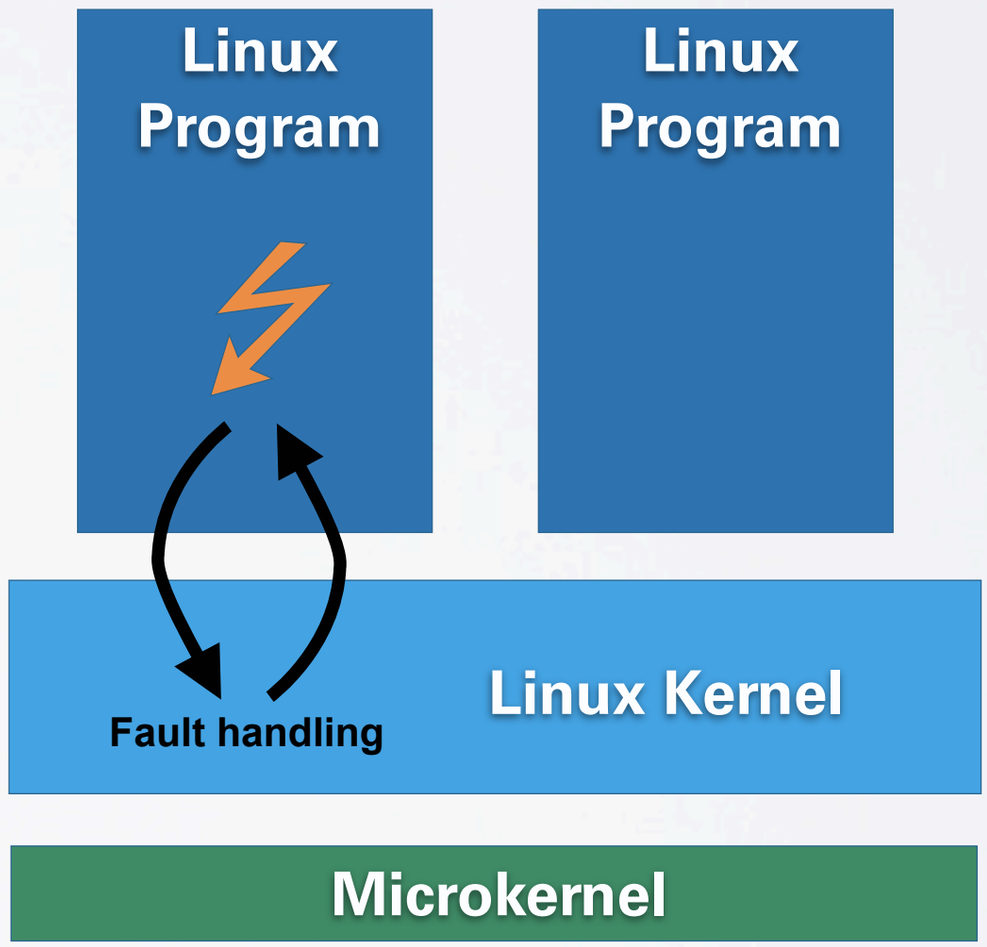
- ARM architecture
- Available CPU features: MMU
- Paravirtualization – L^{4}Linux
- TECOM-FP7: TrustZone for Virtualization

L4LINUX

- Adapted Linux kernel
  - runs on Fiasco.OC & L4Re
  - „Normal“ program, runs Linux kernel code in user mode, including device drivers
- Binary compatible for applications
- Address space for kernel and each program
  - Programs isolated from each other
  - Kernel isolated from programs
- CPU, memory, devices

L4Linux CPU Virtualization

- Native execution
- Exceptions reflected by microkernel
  - System calls
  - Page faults
  - Other exceptions



Multi-Processor Virtualization

- Linux has multiple virtual CPUs (vCPU)
  - Shared memory between cores
- Migration of application threads is done by the Linux kernel
- Inter (v)CPU communication done with microkernel primitives

L⁴Linux Memory Virtualization

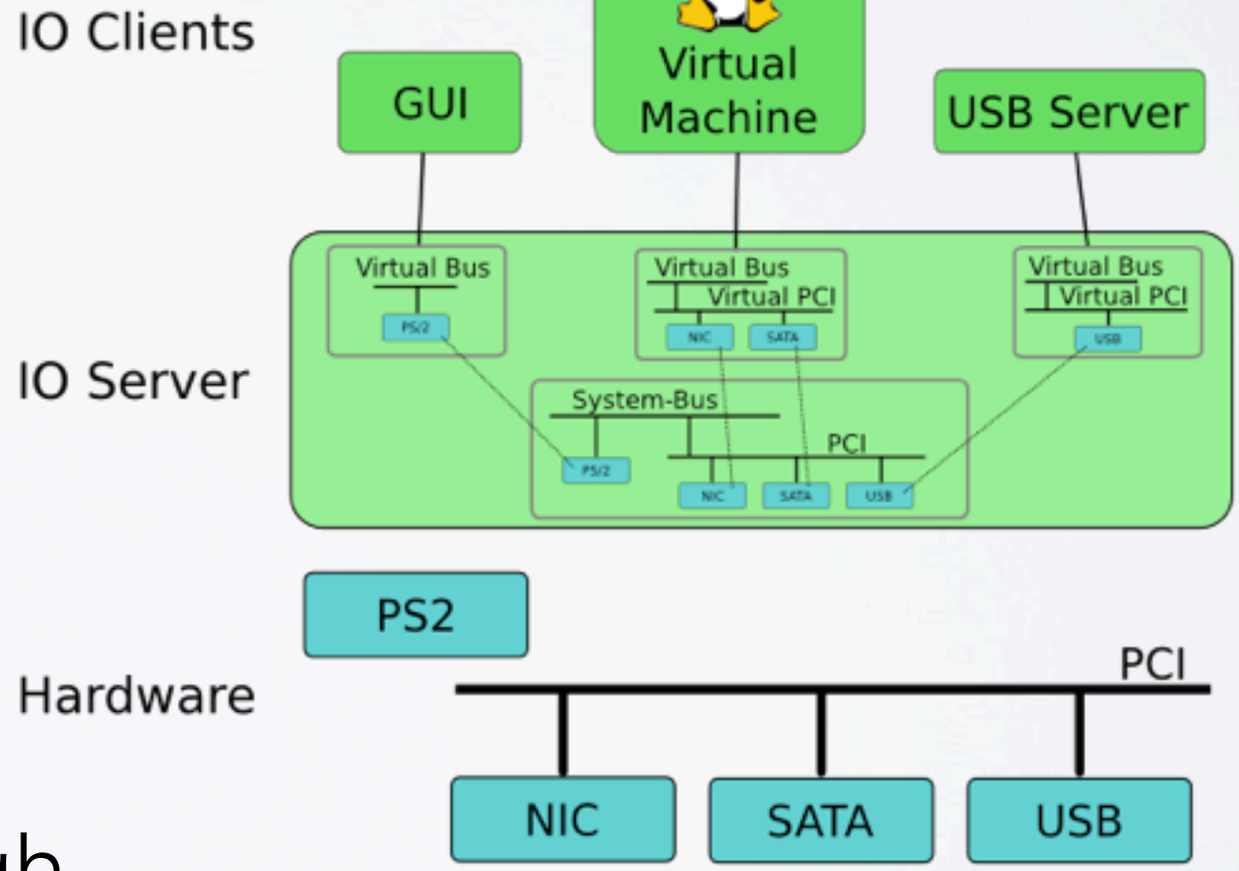
- L4Re supplies Linux memory (virtual)
- MMU managed by microkernel
- Hooks in Linux page-table code use Fiasco memory-mapping functionality

DEVICES

- Virtual PCI bus and/or platform devices
  - Pass-through devices according to configuration
- Stub drivers for L4Re services:
  - Framebuffer driver for windowing system
  - Input driver for keyboard/mouse events
  - Serial driver for basic input/output
  - Network drivers for virtual switch

PLATFORM

- Central IO service
  - Device discovery
  - Device enumeration
- Per client device access
  - Virtual buses
  - Virtual interrupt controller
  - Device pass-through



Faithful Virtualization

- Unmodified guest OS
- AMD SVM, Intel VT
- 1500 LoC in Fiasco for SVM and VT support
- Off-the-shelf VMM (e.g. QEmu on L4Linux)

WHAT ELSE?

- DDE, virtual network switch
- Debugging: Valgrind
- Checkpoint & Restart
- ARM Virtualization
- Run-time environment:
  libc, libstdc++, virtual-FS, pthread, communication framework, dynamic linking, scriptable startup with lua, ...

GO DOWNLOAD

http://L4Re.tudos.org