Microkernel Construction

Introduction

SS2012

Hermann Härtig
Torsten Frenzel

TU Dresden
Operating Systems Group

Class Goals

Provide deeper understanding of OS mechanisms

Introduce L4 principles and concepts

Make you become enthusiastic L4 hackers

Propaganda for OS research at TU Dresden

Hermann Härtig
Torsten Frenzel

TU Dresden
Operating Systems Group

Administration

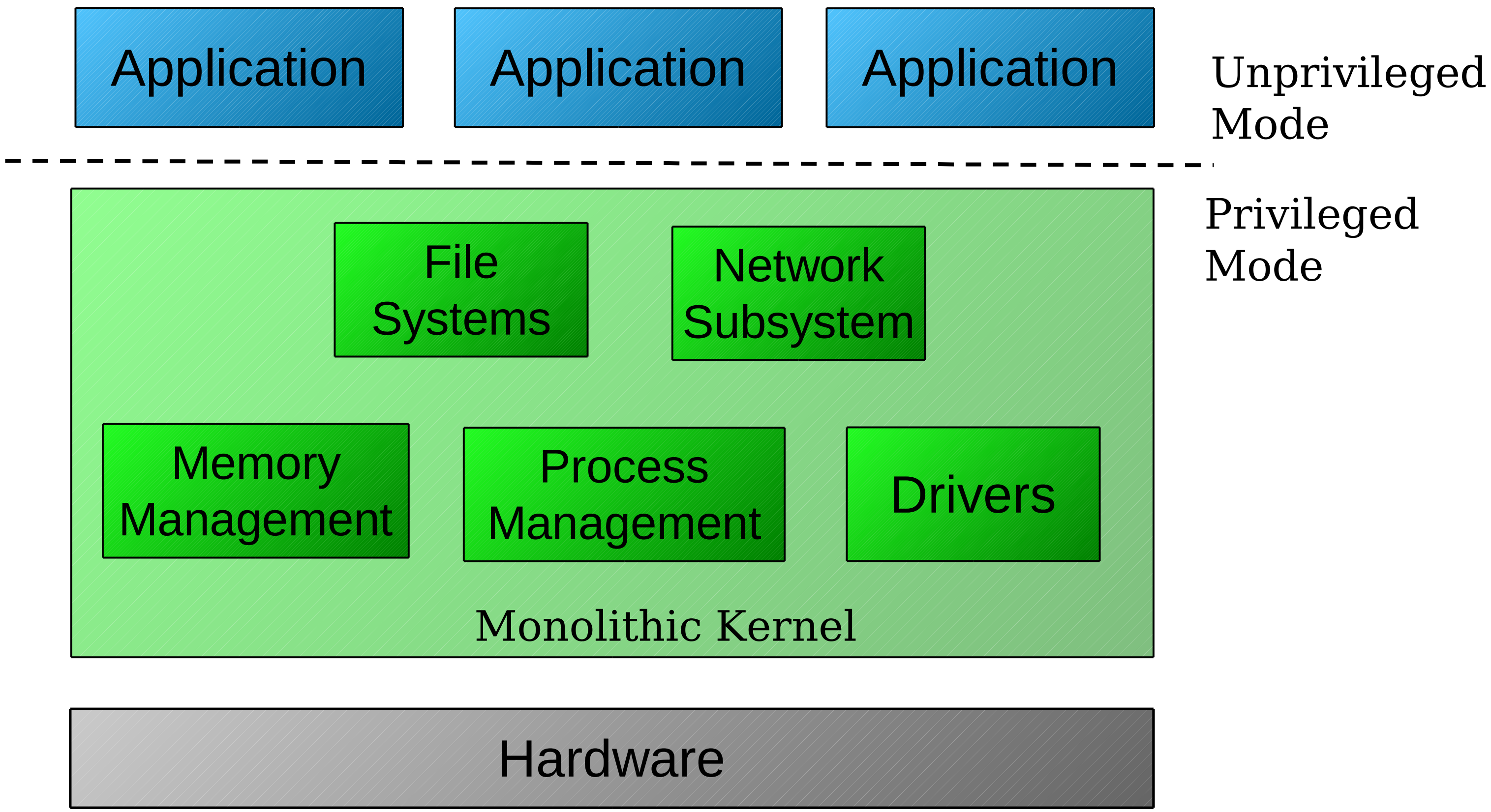
- Thursday, 4th DS, 2 SWS
- Slides: http://www.tudos.org → Teaching → Microkernel Construction
- Subscribe to our mailing list:

  http://www.tudos.org/mailman/listinfo/mkc2012
- In winter term:
  - Construction of Microkernel-based Systems (2 SWS)
  - Various Labs

Hermann Härtig
Torsten Frenzel

TU Dresden Operating Systems Group

„Monolithic“ Kernel System Design



Hermann Härtig
Torsten Frenzel

TU Dresden
Operating Systems Group

Monolithic Kernel OS (Propaganda)

- **System components run in privileged mode**

➔ No protection between system components
  - Faulty driver can crash the whole system
  - More than 2/3 of today's OS code are drivers

➔ No need for good system design
  - Direct access to data structures
  - Undocumented and frequently changing interfaces

➔ Big and inflexible
  - Difficult to replace system components

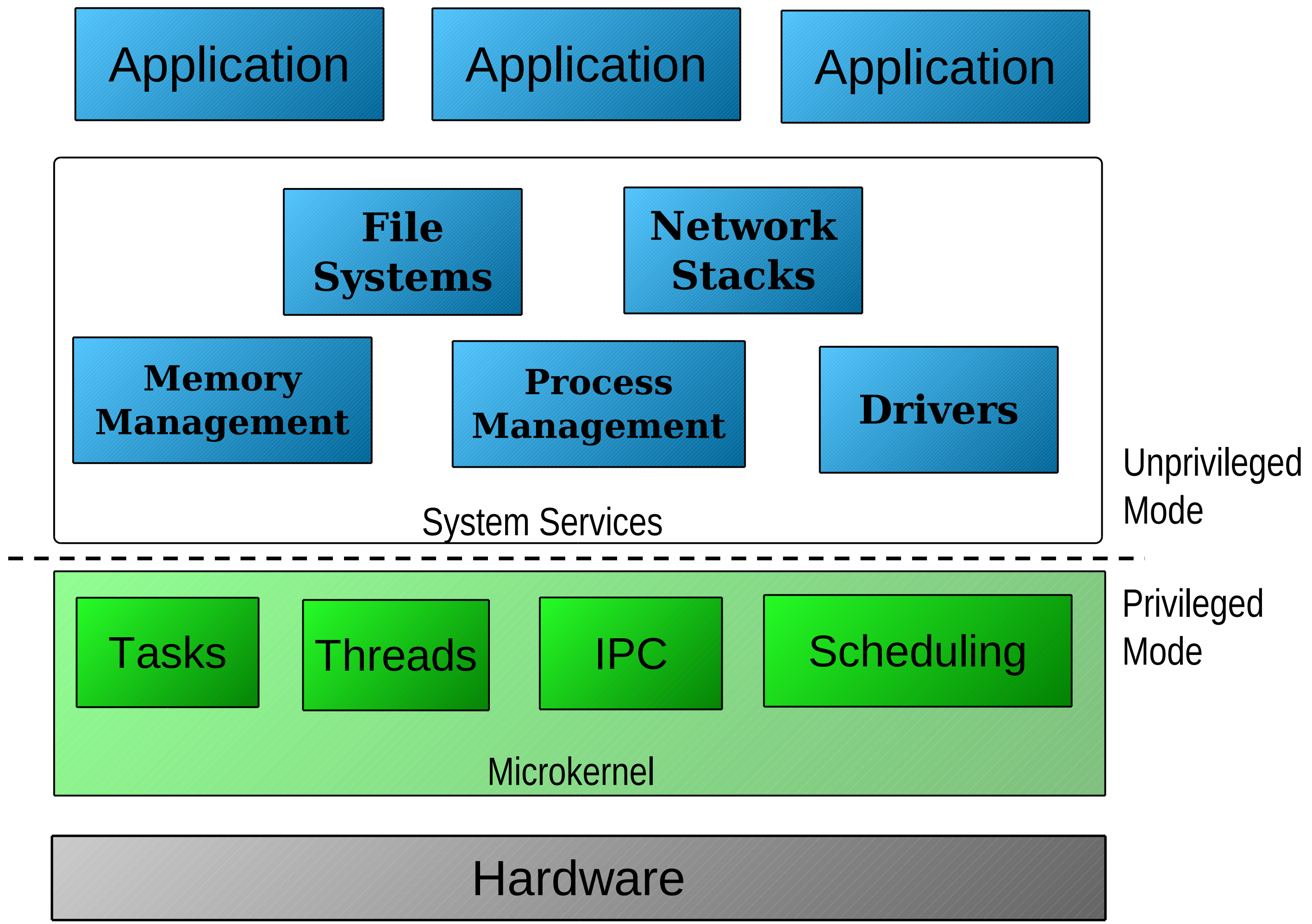
Why something different?

- More and more difficult to manage increasing OS complexity

Hermann
Härtig
Torsten
Frenzel

TU Dresden
Operating
Systems Group

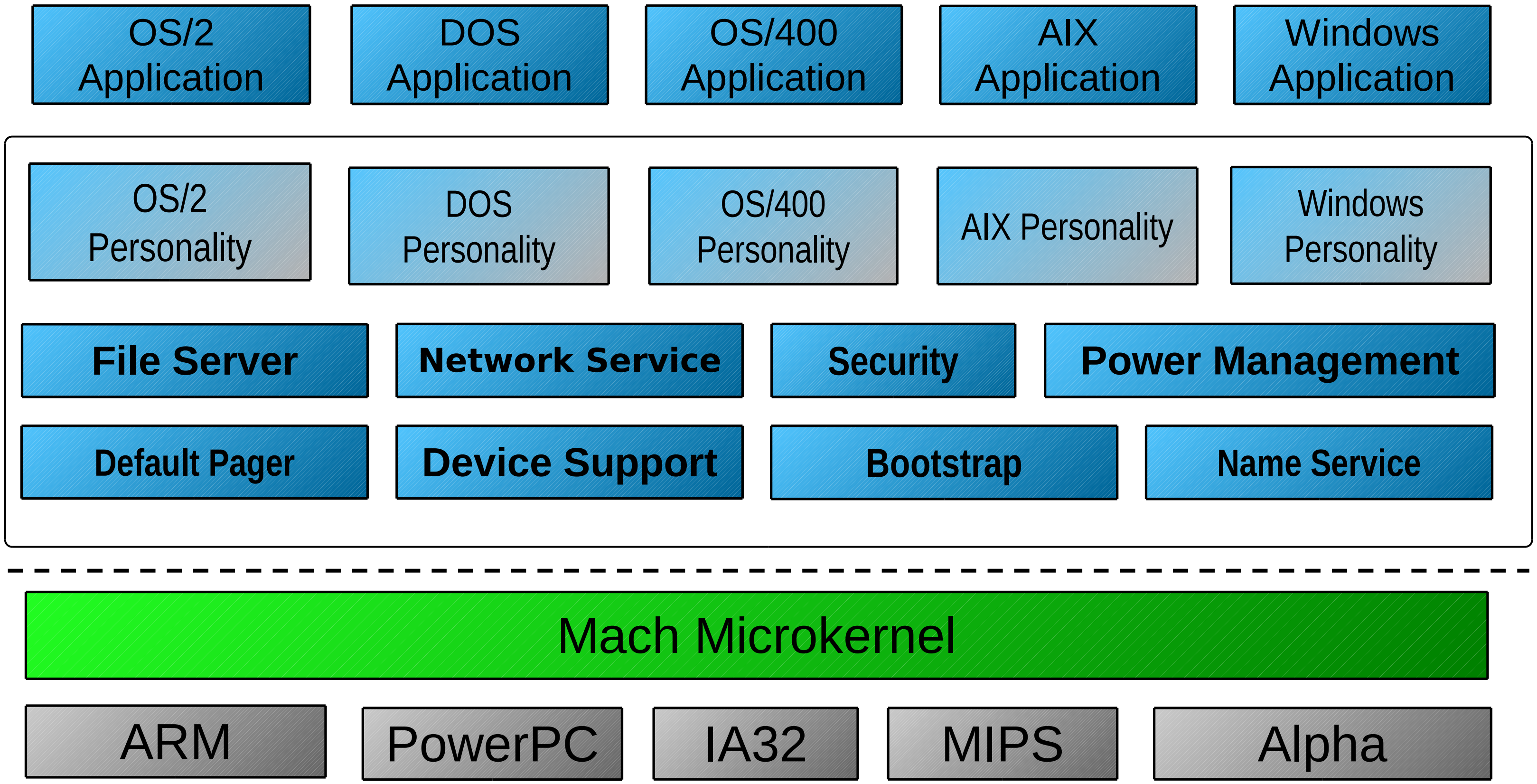
Microkernel System Design



Hermann
Härtig
Torsten
Frenzel

TU Dresden
Operating
Systems Group

Example – IBM Workplace OS / Mach

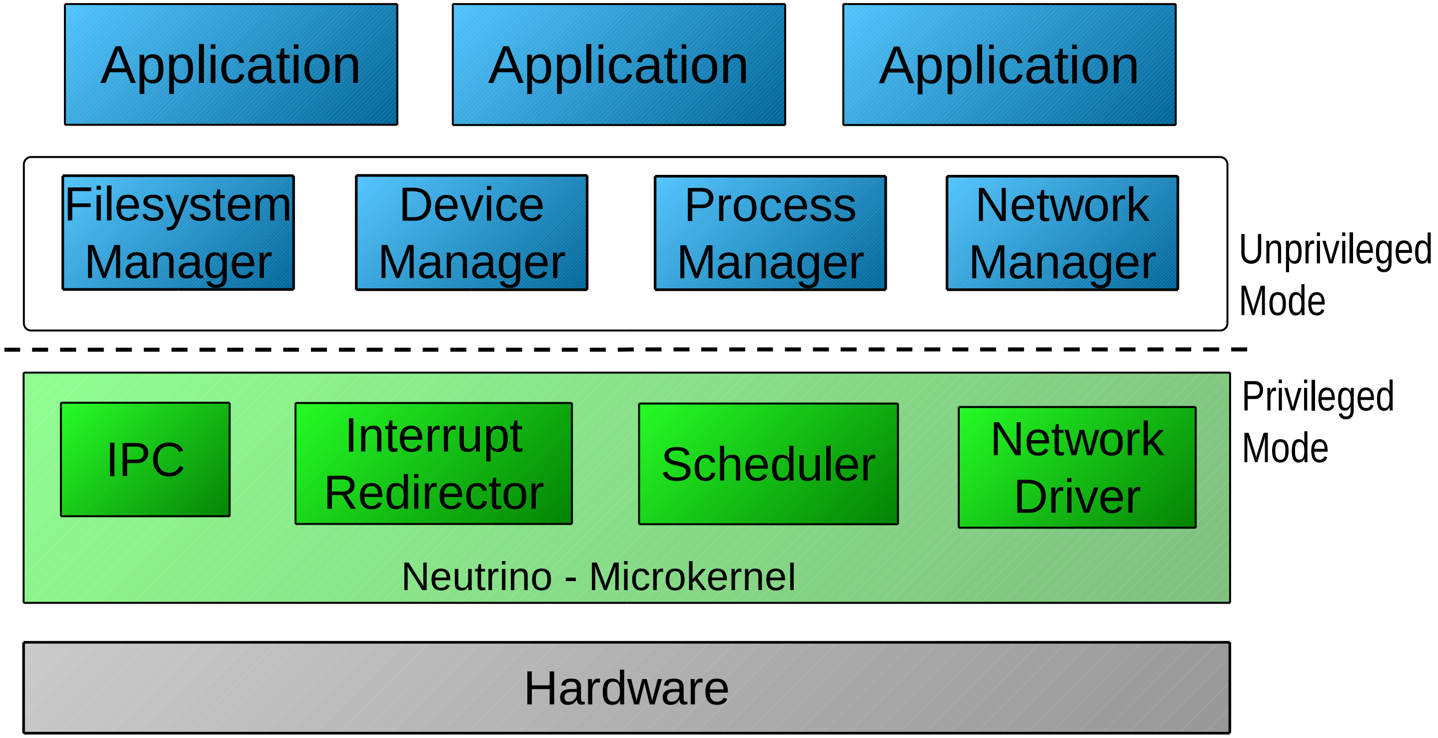


Hermann Härtig
Torsten Frenzel

TU Dresden
Operating Systems Group

Example – QNX / Neutrino

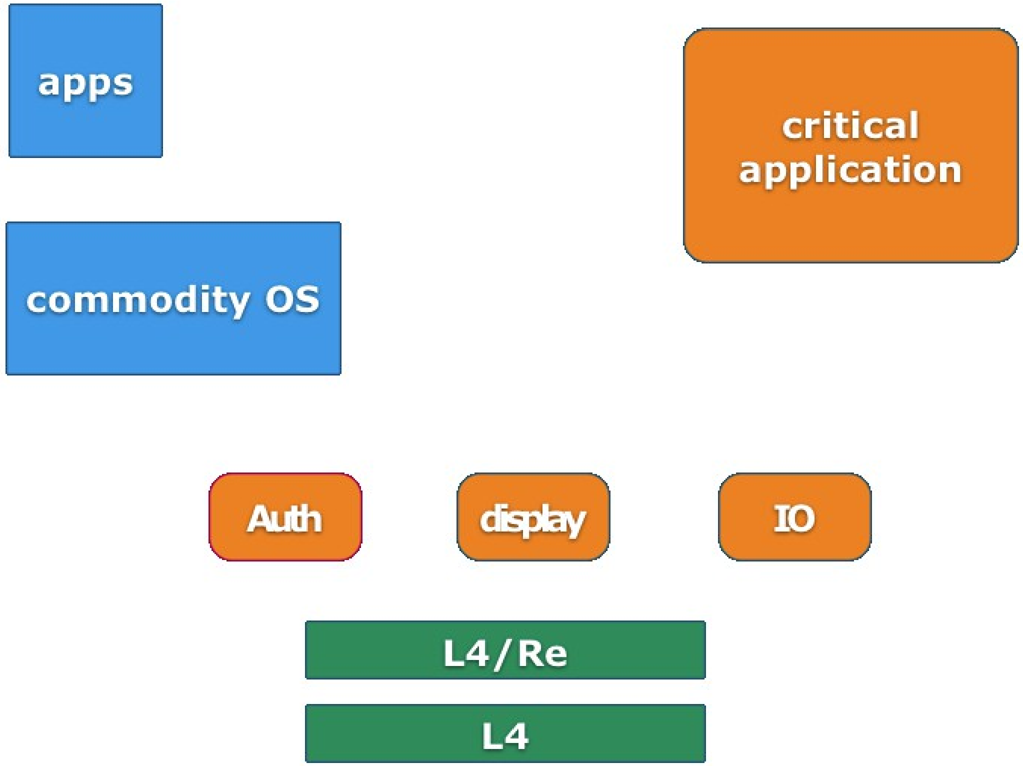
- Embedded systems
- Message passing system (IPC)
- Network transparency



Hermann Härtig
Torsten Frenzel

TU Dresden
Operating Systems Group

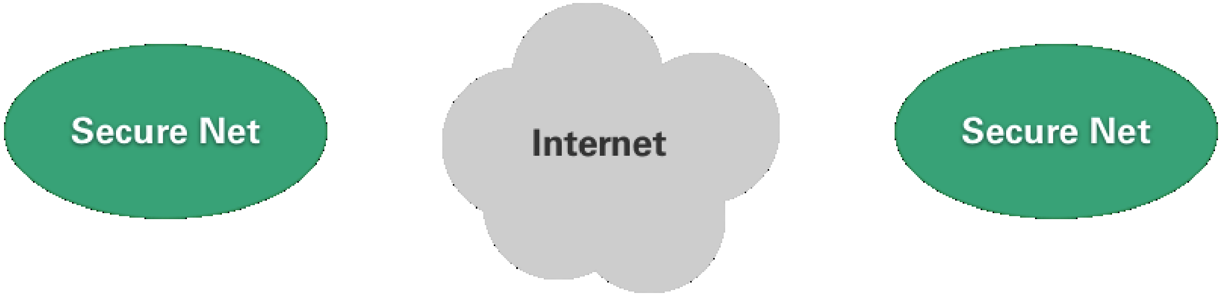
More Interesting



Hermann Härtig
Torsten Frenzel

TU Dresden
Operating Systems Group

Example: VPN

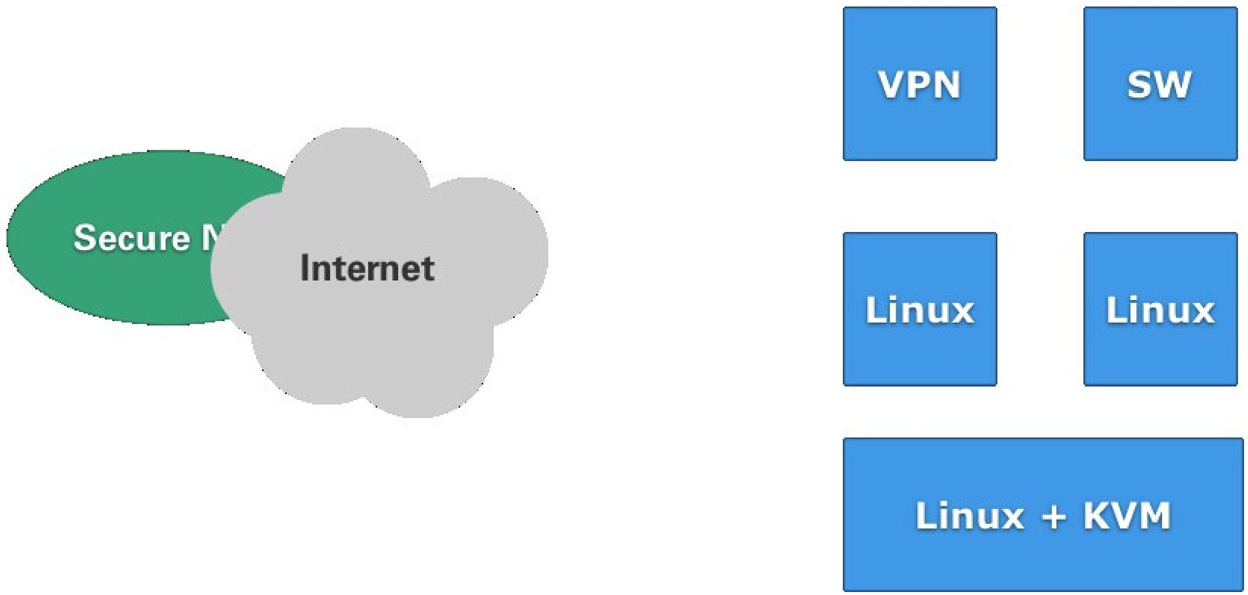


Hermann
Härtig
Torsten
Frenzel

TU Dresden
Operating
Systems Group

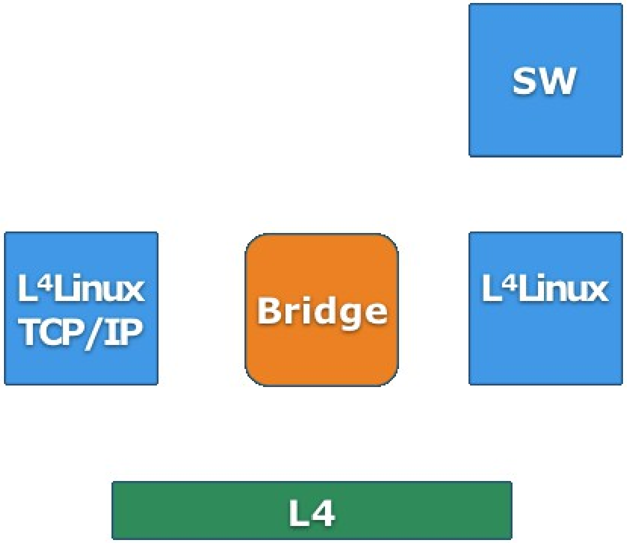
Hermann
Härtig
Torsten
Frenzel

TU Dresden
Operating
Systems Group



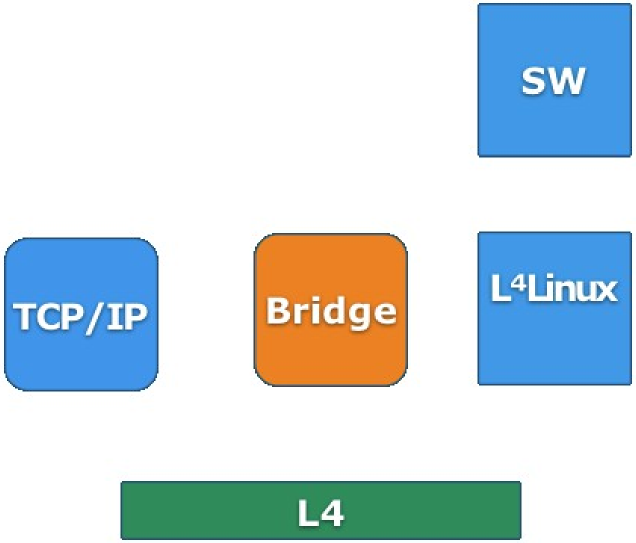
Hermann Härtig
Torsten Frenzel

TU Dresden
Operating Systems Group



Hermann
Härtig
Torsten
Frenzel

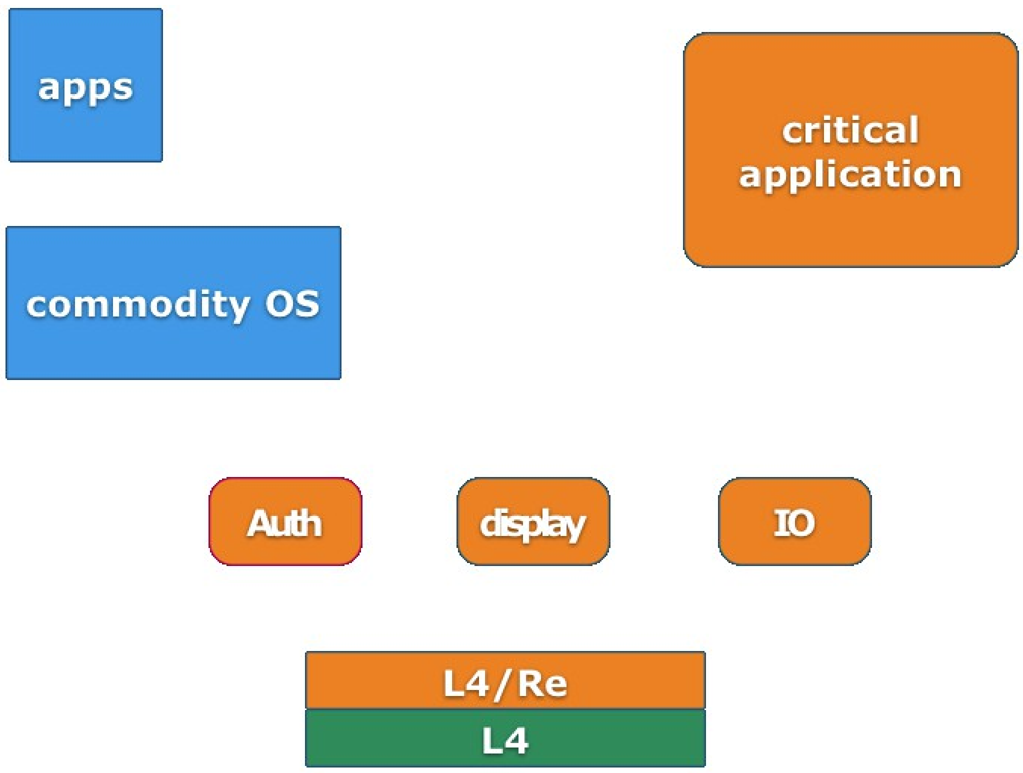
TU Dresden
Operating
Systems Group



Hermann
Härtig
Torsten
Frenzel

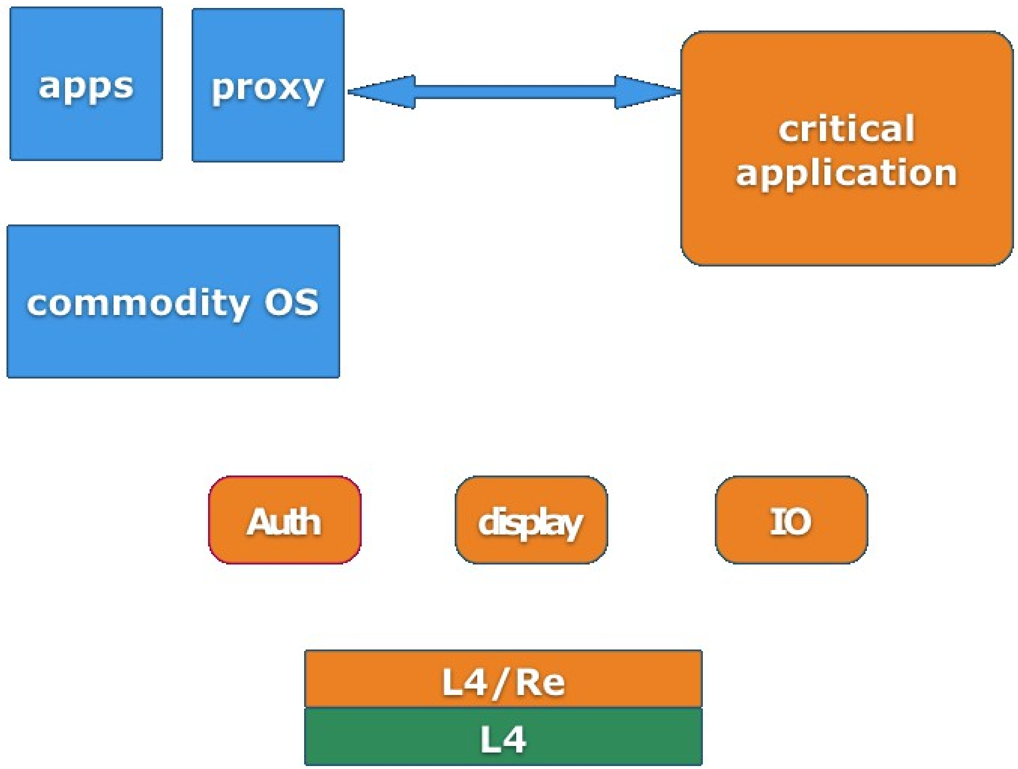
TU Dresden
Operating
Systems Group

More On Critical Applications



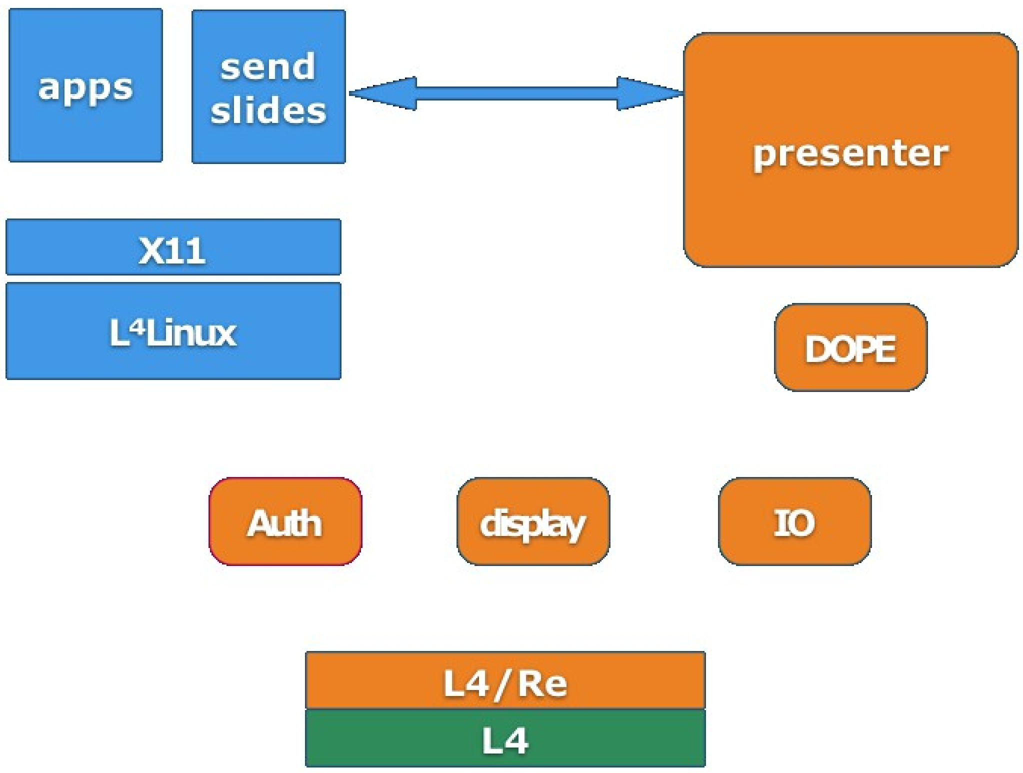
Hermann Härtig
Torsten Frenzel

TU Dresden
Operating Systems Group



Hermann
Härtig
Torsten
Frenzel

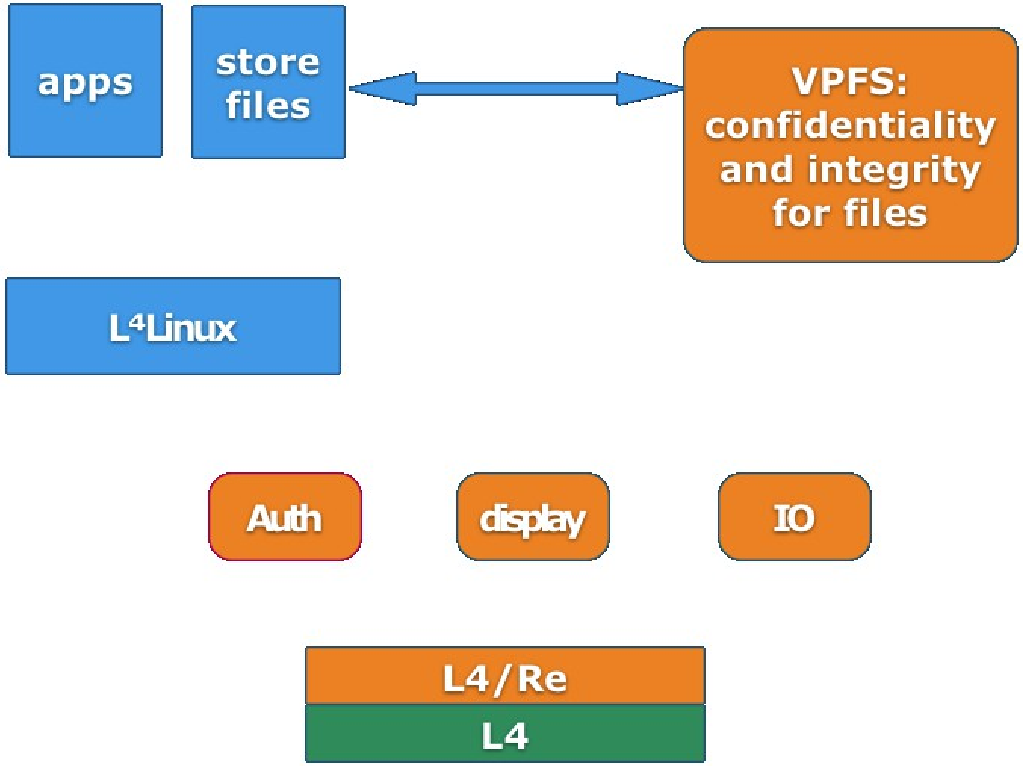
TU Dresden
Operating
Systems Group



Hermann Härtig
Torsten Frenzel

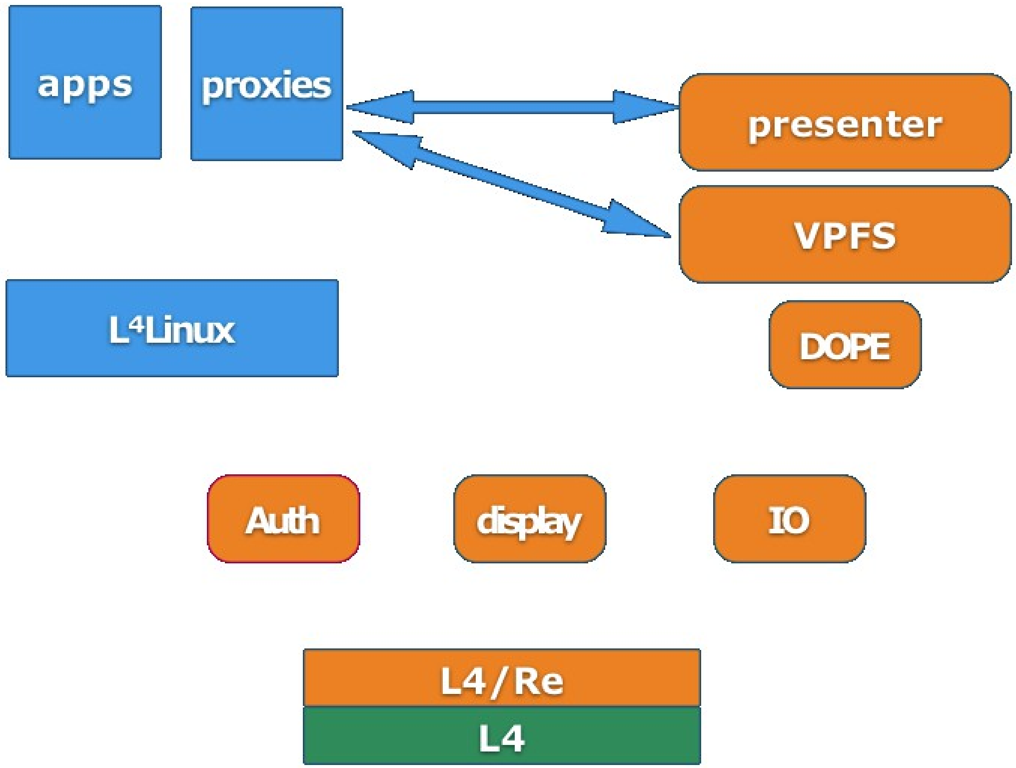
TU Dresden
Operating Systems Group

Virtual Private File System

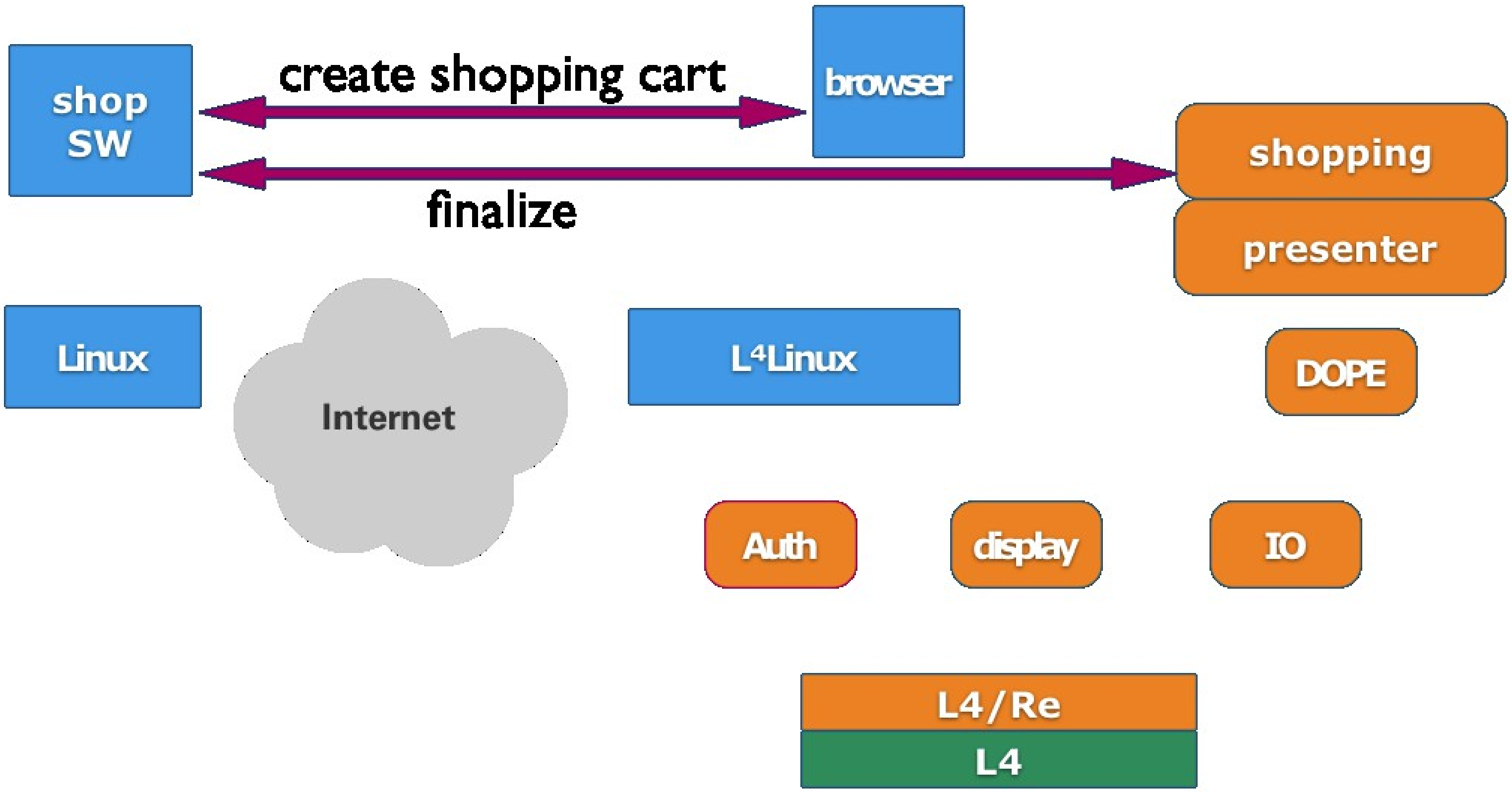


Hermann
Härtig
Torsten
Frenzel

TU Dresden
Operating
Systems Group

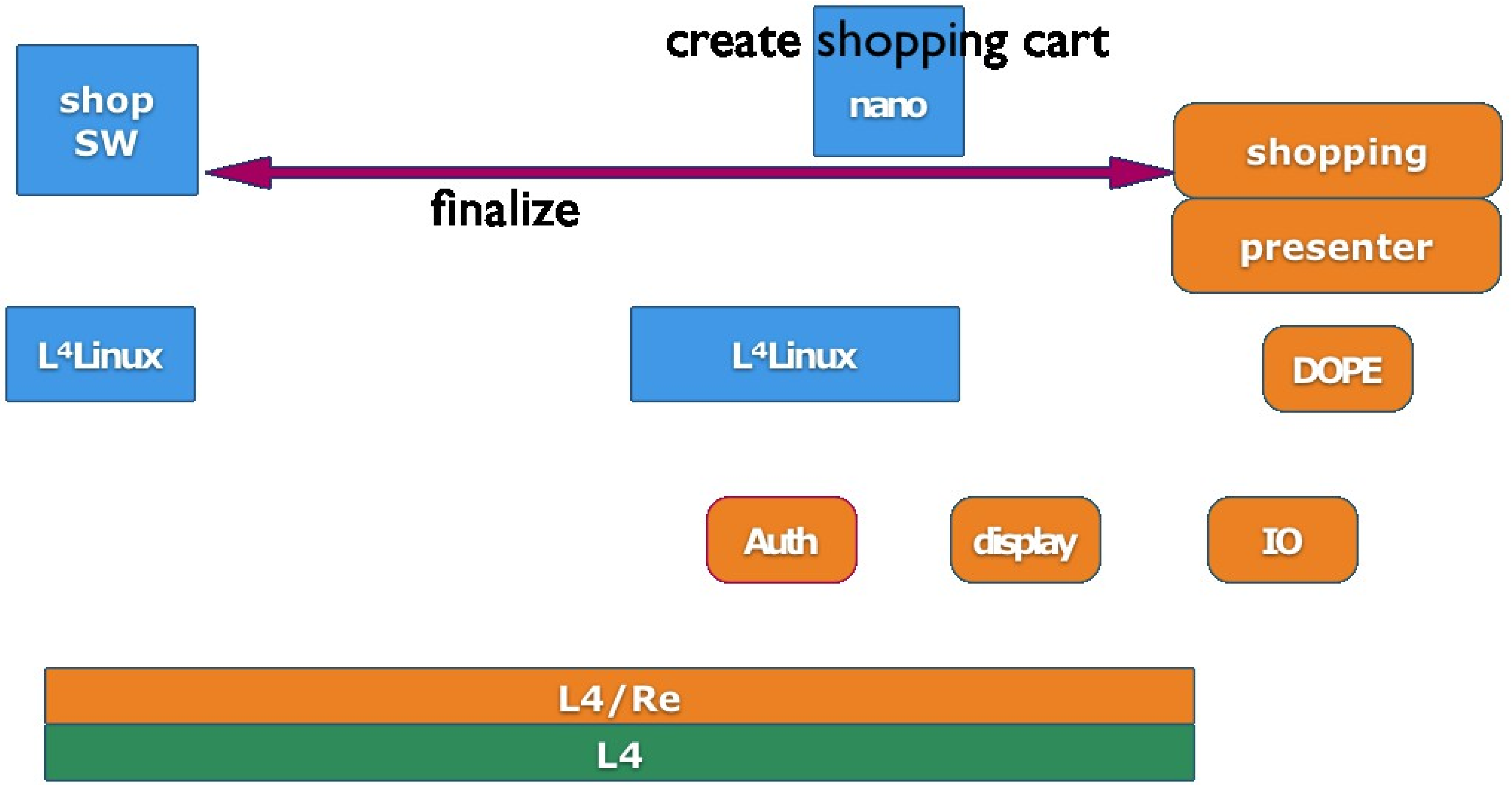

apps
proxies
presenter
VPFS
L⁴Linux
DOPE
Auth
display
IO
L4/Re
L4

Shopping



Hermann Härtig
Torsten Frenzel

TU Dresden
Operating Systems Group



Hermann
Härtig
Torsten
Frenzel

TU Dresden
Operating
Systems Group

Microkernel OS - The Vision (1)

- **System components run as user-level servers**
- Protection and isolation between system components
  - More secure / safe systems
  - Less error prone
  - Small *Trusted Computing Base*
- „Enforces“ clear system design
  - Well defined interfaces to system services
  - No dependencies between system services other than explicitly specified through service interfaces
- Small and flexible
  - Small OS kernel
  - Easier to replace system components

Hermann Härtig
Torsten Frenzel

TU Dresden
Operating Systems Group

Visions vs. Reality

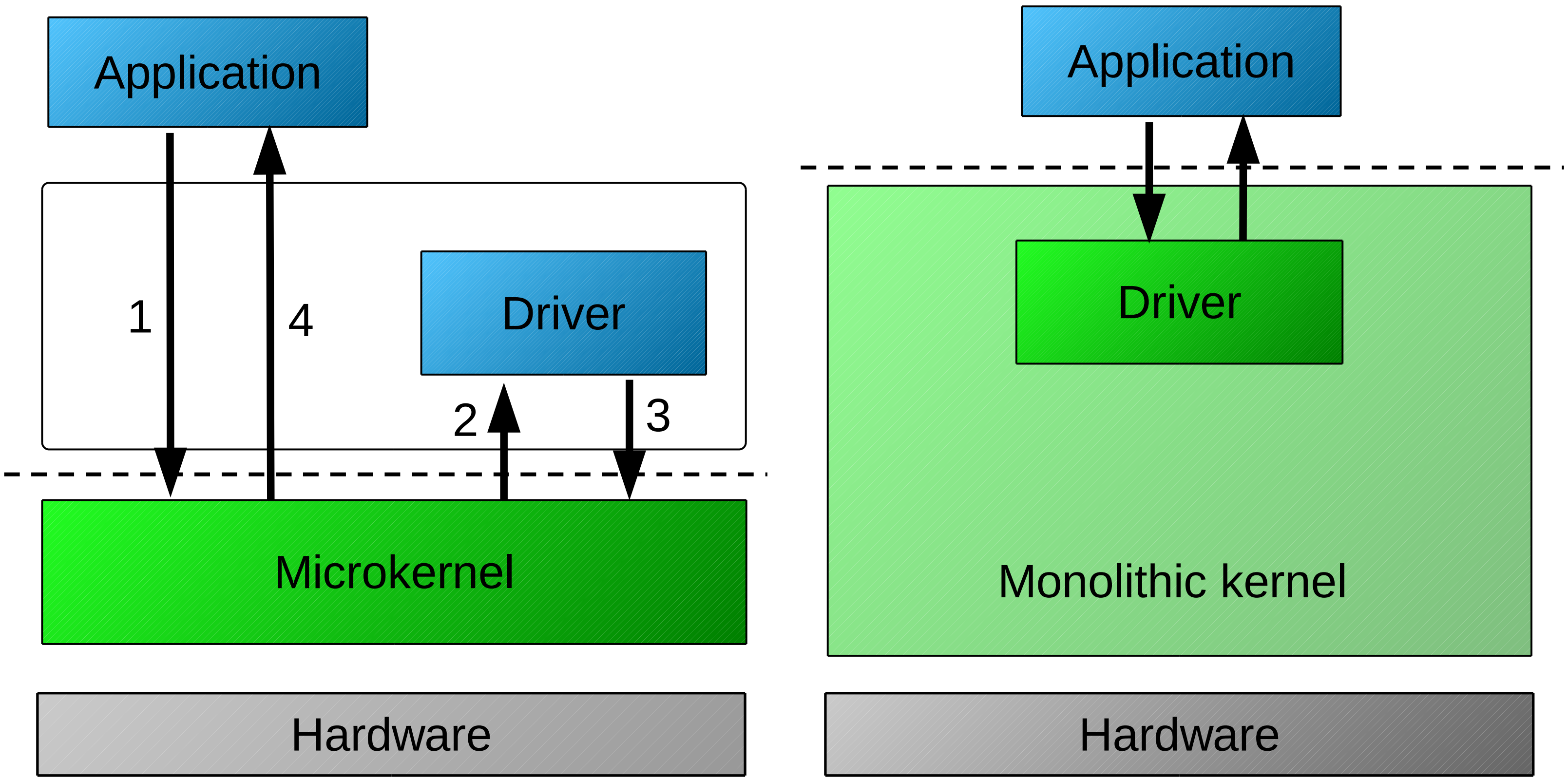
- Flexibility and Customizable
  - Monolithic kernels are modular
- Maintainability and complexity
  - Monolithic kernel have layered architecture
- ✔ Robustness
  - Microkernels are superior due to isolated system components
  - Trusted code size (i386)
    - Fiasco kernel: about 15.000 loc
    - Linux kernel: about 300.000 loc (without drivers)
- ✗ Performance
  - Application performance degraded
  - Communication overhead (see next slides)

Hermann Härtig
Torsten Frenzel

TU Dresden
Operating Systems Group

Robustness vs. Performance (1)

- System calls
  - Monolithic kernel: 2 kernel entries/exits
  - Microkernel: 4 kernel entries/exits + 2 context switches

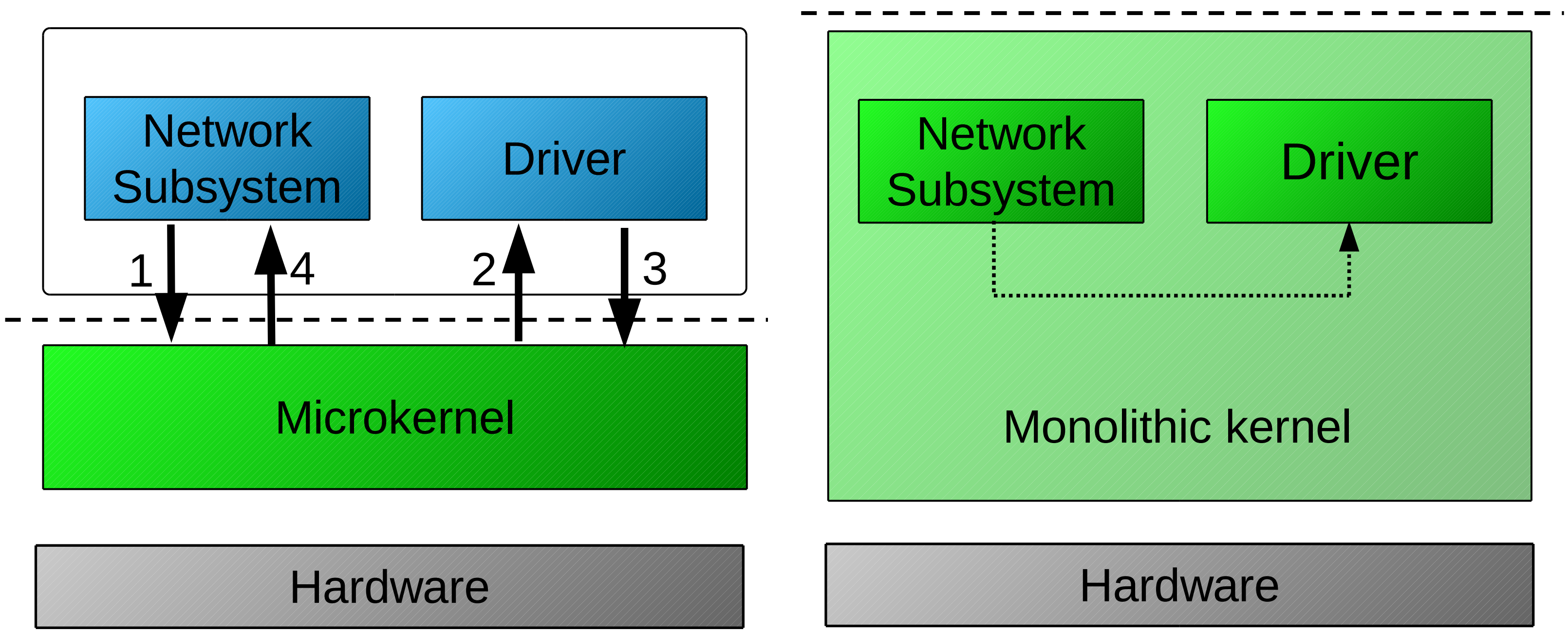


Hermann Härtig
Torsten Frenzel

TU Dresden
Operating Systems Group

Robustness vs. Performance (2)

- Calls between system services
  - Monolithic kernel: 1 function call
  - Microkernel: 4 kernel entries/exits + 2 context switches



Hermann
Härtig
Torsten
Frenzel

TU Dresden
Operating
Systems Group

Challenges

- Build functional 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
  - Synchronization
  - Device Drivers
  - File Systems
  - Communication Interfaces
- ➔ *Subject of lecture “Construction of Microkernel-based systems” (in winter term)*

Hermann
Härtig
Torsten
Frenzel

TU Dresden
Operating
Systems Group

L4 Microkernel Family

- Originally developed by Jochen Liedtke (GMD / IBM Research)
- Current development:
  - Uni Karlsruhe: Pistachio
  - UNSW/NICTA/OKLABS: OKL4, SEL4, L4Verified
  - TU Dresden: Fiasco.OC, Nova
- Support for hardware architectures:
  - **X86, ARM, ...**

More Microkernels (Incomplete list)

- Commercial kernels
  - Singularity @ Microsoft Research
  - K42 @ IBM Research
  - velOSity/INTEGRITY @ Green Hills Software
  - Chorus/ChorusOS @ Sun Microsystems
  - PikeOS @ SYSGO AG
  - OKL4
- Research kernels
  - EROS/CoyotOS @ John Hopkins University
  - Minix @ FU Amsterdam
  - Amoeba @ FU Amsterdam
  - Pebble @ Bell Labs
  - Grasshopper @ University of Sterling
  - Flux/Fluke @ University of Utah

Hermann Härtig
Torsten Frenzel

TU Dresden
Operating Systems Group

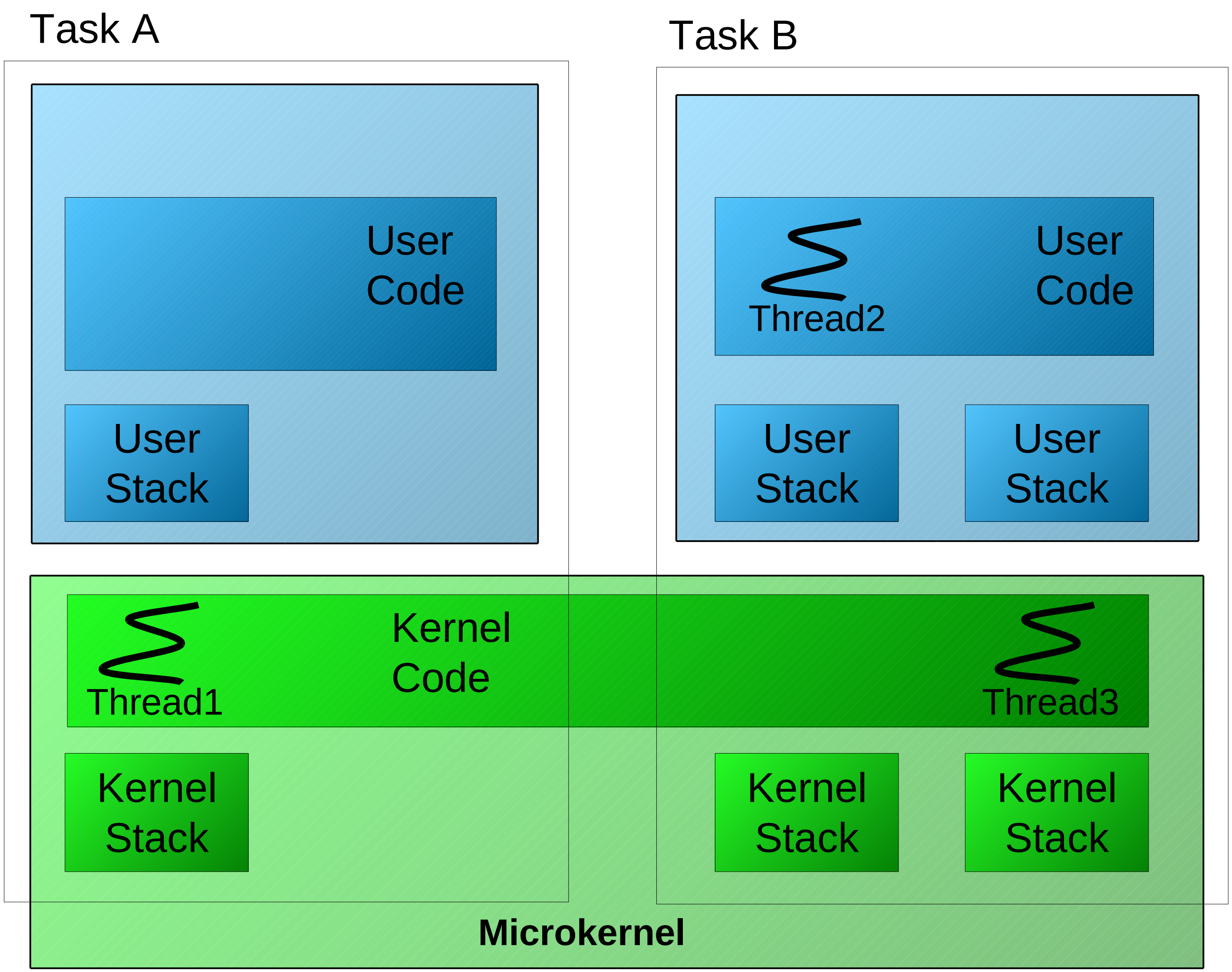
L4 - Concepts

- 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)

Hermann Härtig
Torsten Frenzel

TU Dresden
Operating Systems Group

Threads and Tasks



Hermann
Härtig
Torsten
Frenzel

TU Dresden
Operating
Systems Group

Threads (1)

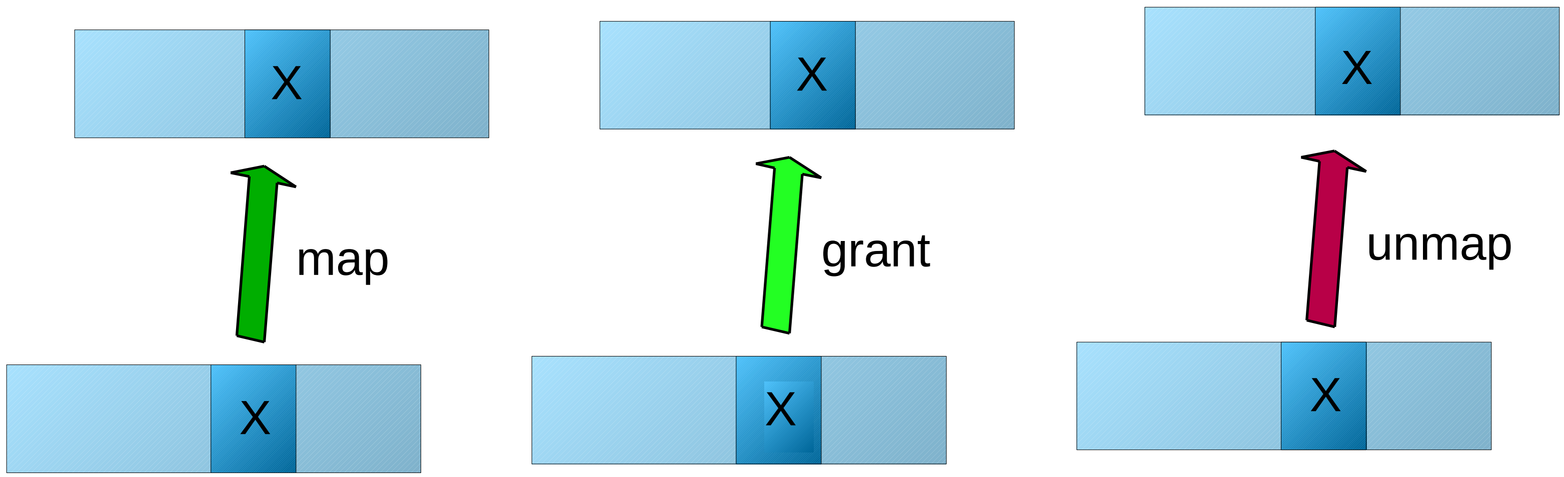
- Represent unit of execution
  - Execute user code (application)
  - Execute kernel code (system calls, page faults, interrupts, exceptions)
- Subject to scheduling
  - Quasi-parallel execution on one CPU
  - Parallel execution on multiple CPUs
  - Voluntarily switch to another thread possible
  - Preemptive scheduling by the kernel according to certain parameters
- Associated with an address space
  - Executes code in one task at one point in time
    - Migration allows threads move to another task
  - Several threads can execute in one task

Hermann Härtig
Torsten Frenzel

TU Dresden
Operating Systems Group

Tasks (1)

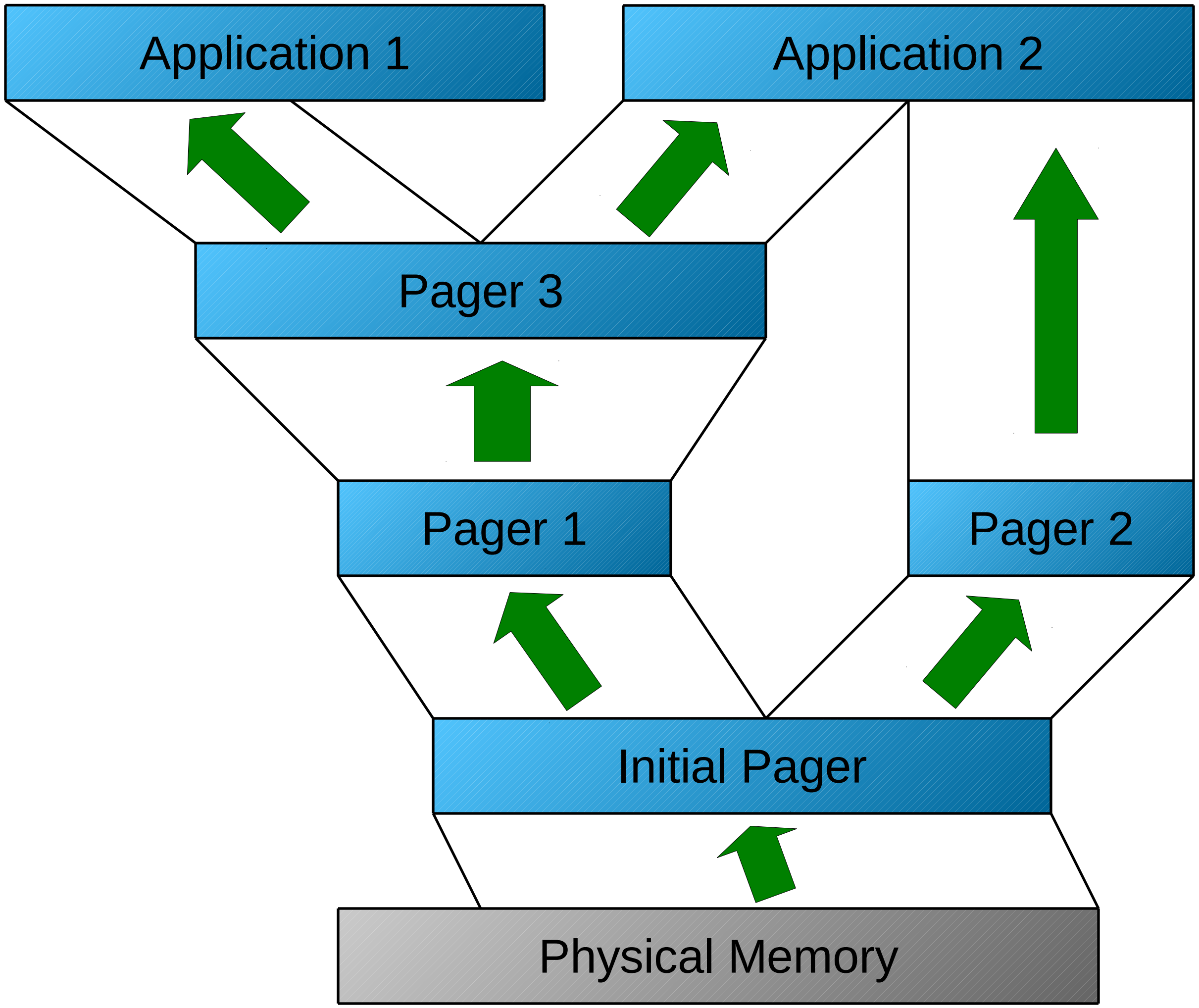
- Represent domain of protection and isolation
- Container for code, data and resources
- Address space: capabilities + memory pages
- Three management operations:
  - Map: share page with other address space
  - Grant: give page to other address space
  - Unmap: revoke previously mapped page



Hermann
Härtig
Torsten
Frenzel

TU Dresden
Operating
Systems Group

Recursive Address Spaces

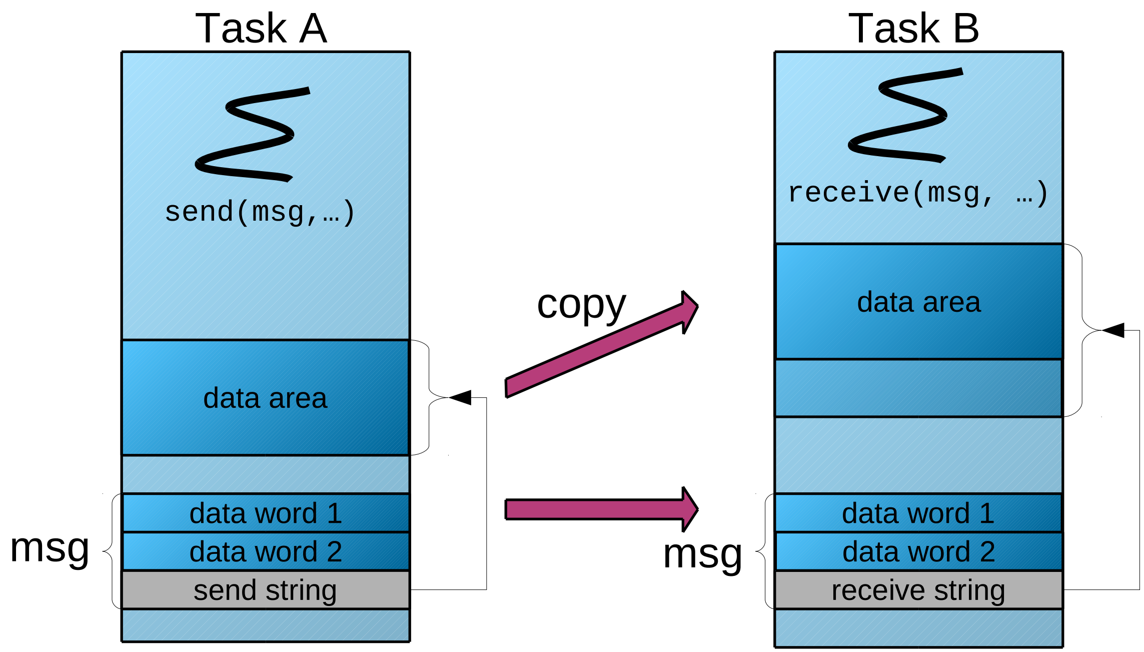


Hermann
Härtig
Torsten
Frenzel

TU Dresden
Operating
Systems Group

Messages: Copy Data

- Direct and indirect data copy
- UTCB message (special area)
- Special case: register-only message
- Pagefaults during user-level memory access possible

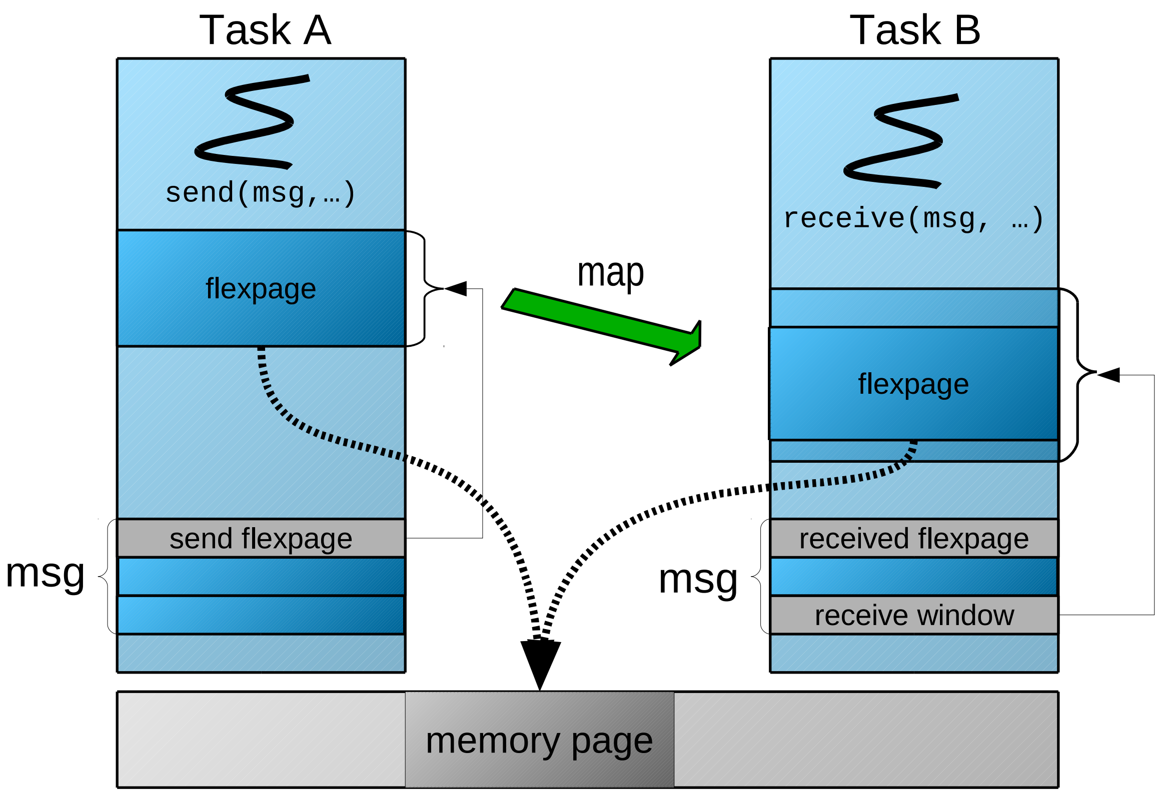


Hermann
Härtig
Torsten
Frenzel

TU Dresden
Operating
Systems Group

Message: Map References

- Used to transfer memory pages and capabilities
- Kernel manipulates page tables
- Used to implement the map/grant operations

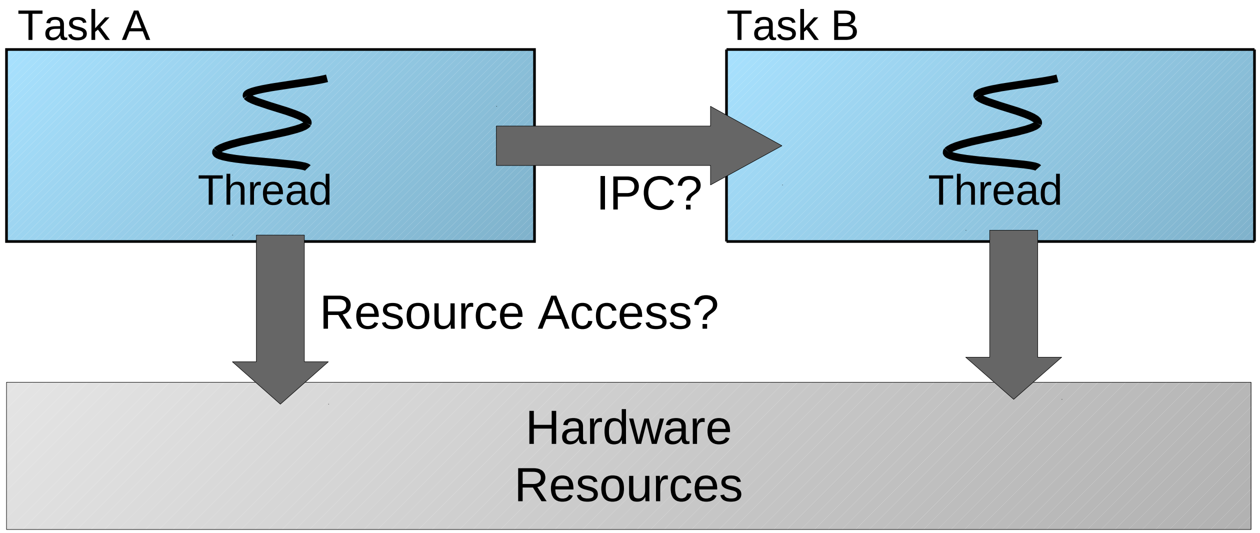


Hermann Härtig
Torsten Frenzel

TU Dresden
Operating Systems Group

Communication and Resource Control

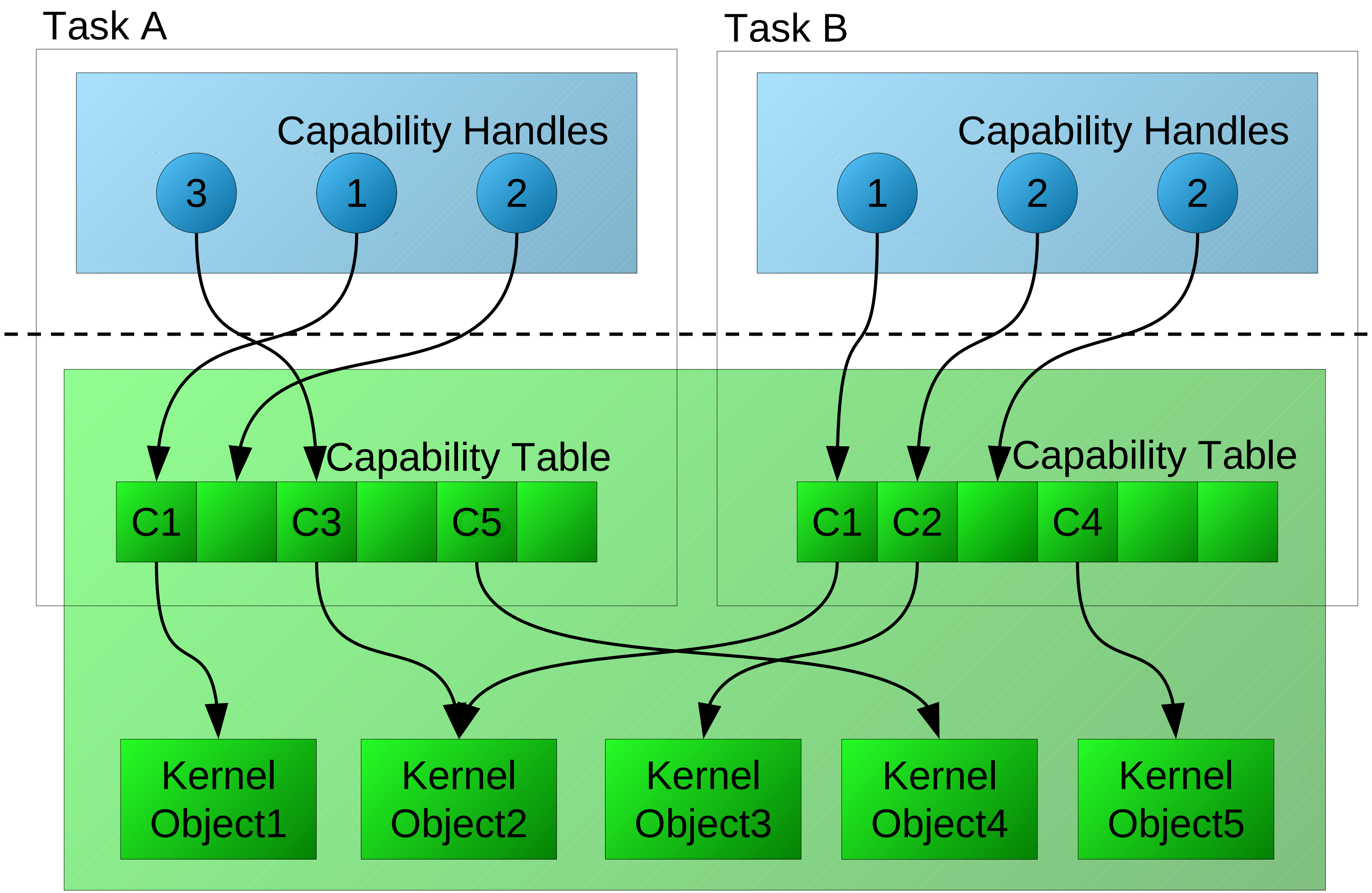
- Need to control who can send data to whom
  - Security and isolation
  - Access to resources
- Approaches
  - IPC-redirection/introspection
  - Central vs. Distributed policy and mechanism
  - ACL-based vs. capability-based



Hermann
Härtig
Torsten
Frenzel

TU Dresden
Operating
Systems Group

Capabilities



Hermann
Härtig
Torsten
Frenzel

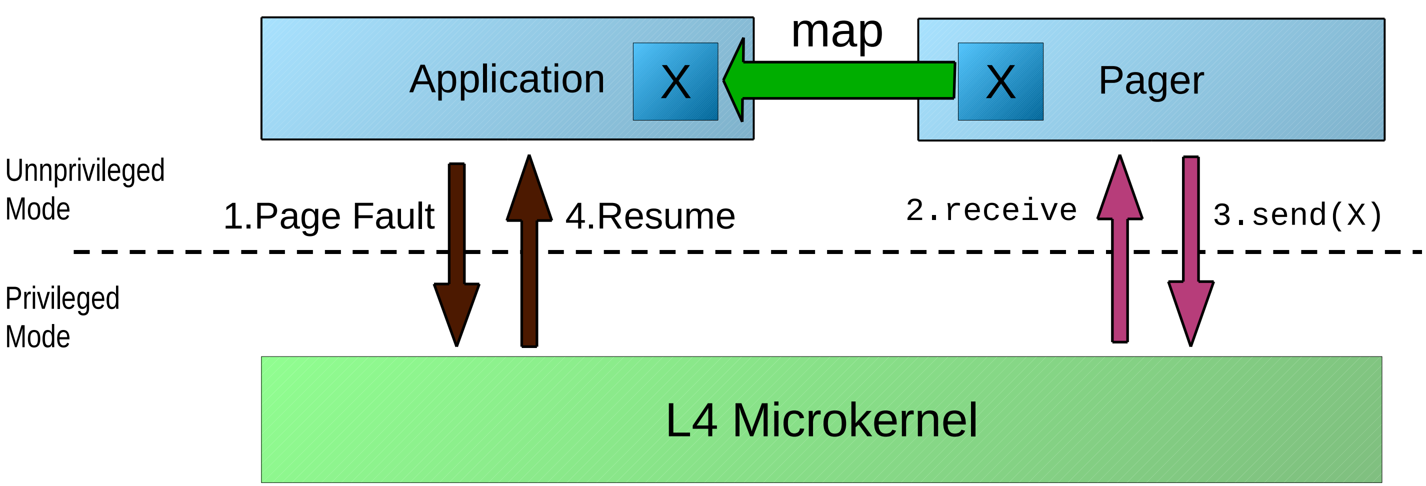
TU Dresden
Operating
Systems Group

Capabilities - Details

- Kernel objects represent resources and communication channels
- Capability
  - Reference to kernel object
  - Associated with access rights
  - Can be mapped from task to another task
- Capability table is task-local data structure inside the kernel
  - Similar to page table
  - Valid entries contain capabilities
- Capability handle is index number to reference entry into capability table
  - Similar to file handle (in POSIX)
- Mapping capabilities establishes a new valid entry into the capability table

Page Faults and Pagers

- Page Faults are mapped to IPC
  - Pager is special thread that receives page faults
  - Page fault IPC cannot trigger another page fault
- Kernel receives the flexpage from pager and inserts mapping into page table of application
- Other faults normally terminate threads

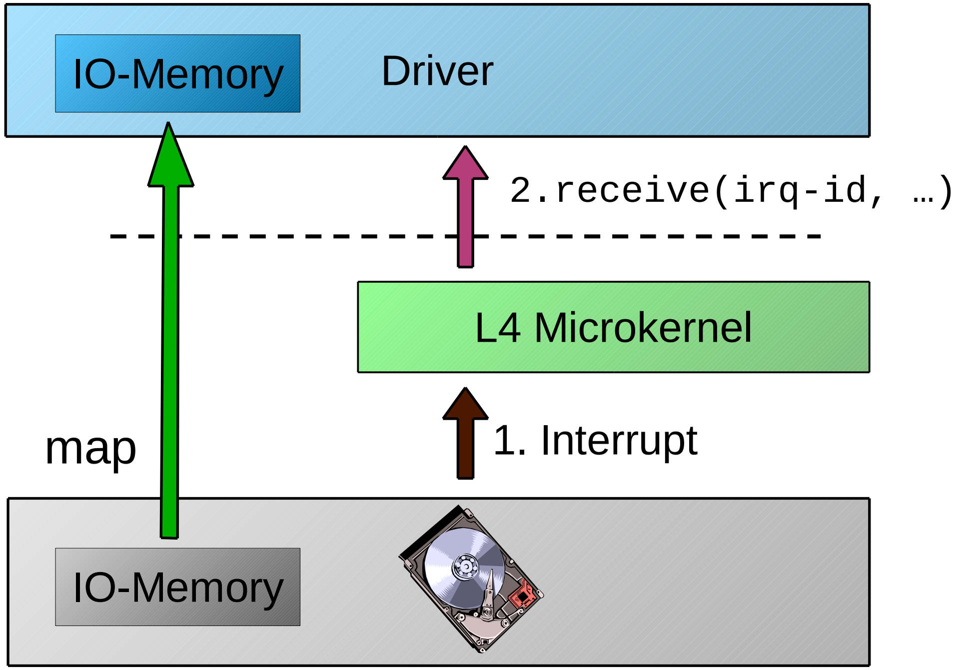


Hermann Härtig
Torsten Frenzel

TU Dresden
Operating Systems Group

Device Drivers

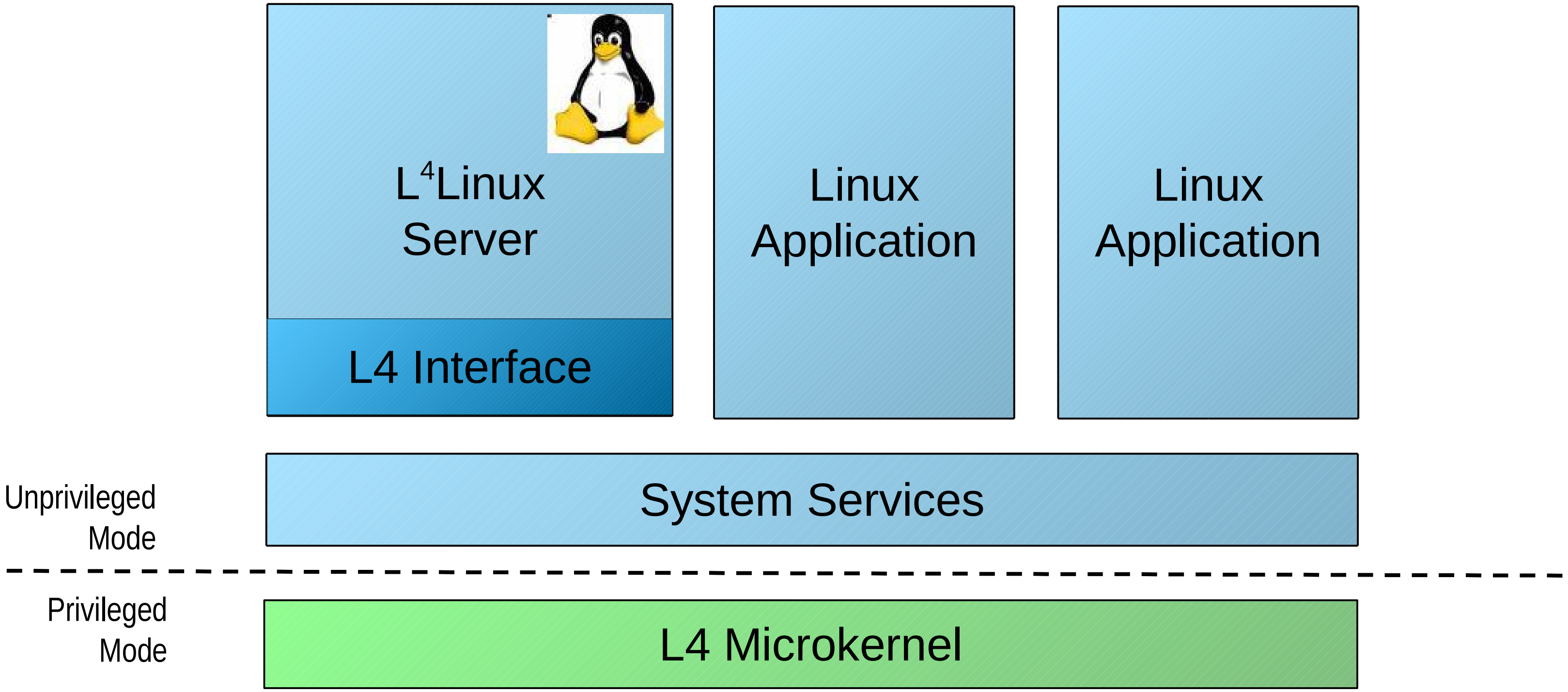
- Hardware interrupts: mapped to IPC
- I/O memory & I/O ports: mapped via flexpages



Hermann
Härtig
Torsten
Frenzel

L4 Applications - L⁴Linux

- Paravirtualized Linux kernel and native Linux applications run as user-level L4 tasks
- System calls / page faults are mapped to L4 IPC



Hermann Härtig
Torsten Frenzel

TU Dresden
Operating Systems Group

Lecture Outline

- **Introduction**
- Address spaces, threads, thread switching
- Kernel entry and exit
- Thread synchronization
- IPC
- Address space management
- Scheduling
- Portability
- Platform optimizations
- Virtualization

Hermann Härtig
Torsten Frenzel

TU Dresden
Operating Systems Group