Microkernel Construction

Introduction

SS2011

Torsten
Frenzel

TU Dresden
Operating
Systems Group

Lecture Goals

Provide deeper understanding of OS mechanisms

Illustrate an alternative system design concept

Promote OS research at TU Dresden

Make all of you enthusiastic kernel hackers

Torsten
Frenzel

TU Dresden
Operating
Systems Group

Administration

- Thursday, 4th DS, 2 SWS
- Theory (INF/E08) and practical exercises (INF/E046)
- Slides / Handouts available at

  http://os.inf.tu-dresden.de/Studium/MkK/
- Mailinglist:

  http://os.inf.tu-dresden.de/mailman/listinfo/mkc2011/

- In winter term:
  - Construction of Microkernel-based Systems (2 SWS)
  - Komplexpraktikum (2 SWS)

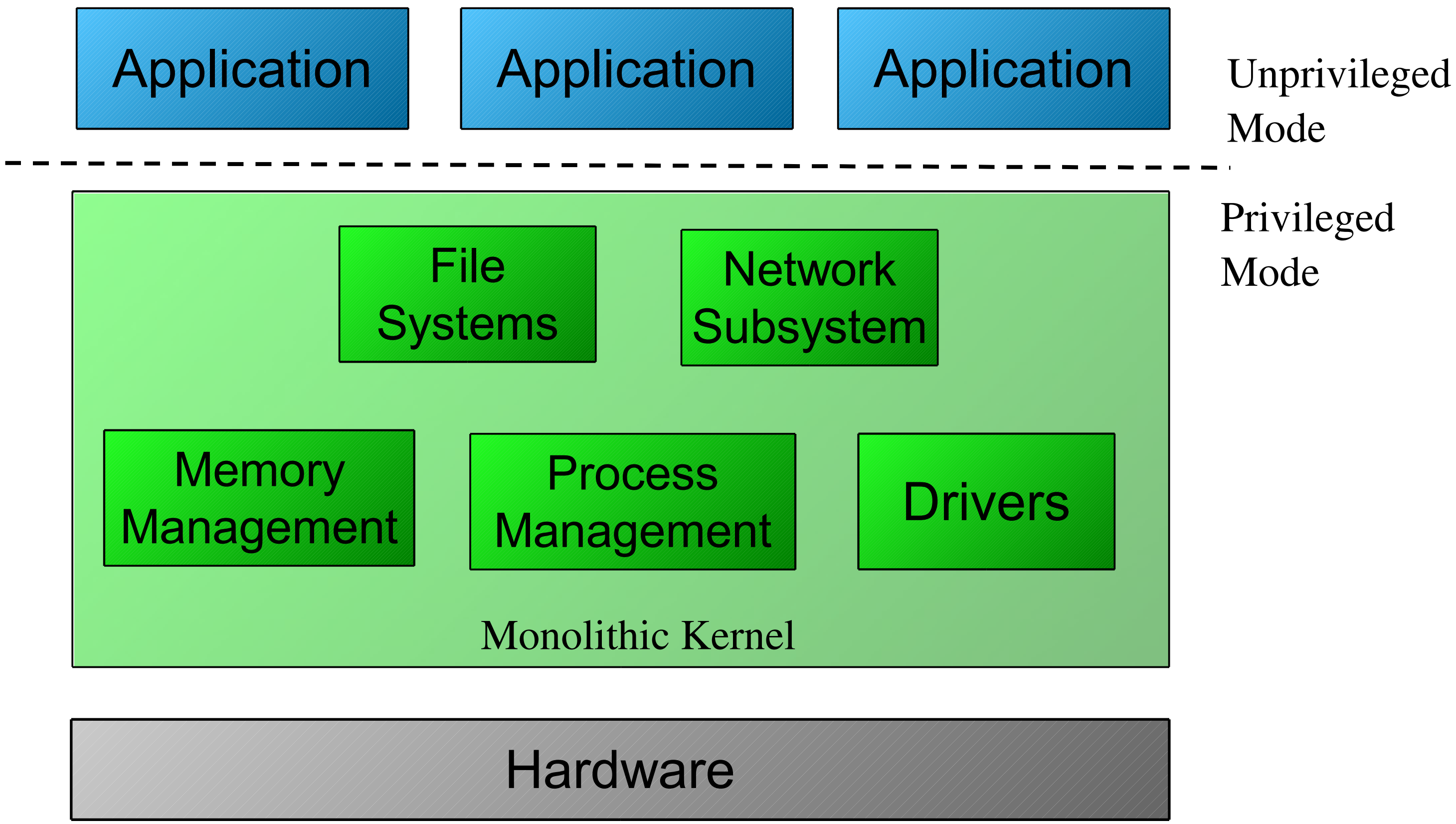
Torsten
Frenzel

TU Dresden
Operating
Systems Group

OS Design Goals

- Flexibility and Customizable
  - Tailored resource management (scheduling algorithms)
  - Scalability from embedded system to server systems
  - Applicable for real-time systems and secure systems
  - Adaptable to specific application scenarios
- Maintainability and complexity
  - Reasonable system structure
  - Well defined interfaces between components
- Robustness
  - Protection and fault isolation of system components
  - Small trusted code size (*Trusted Computing Base*)
- Performance
  - User wants tasks done as fast as possible

Monolithic Kernel System Design



Torsten
Frenzel

TU Dresden
Operating
Systems Group

Monolithic Kernel OS

- **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

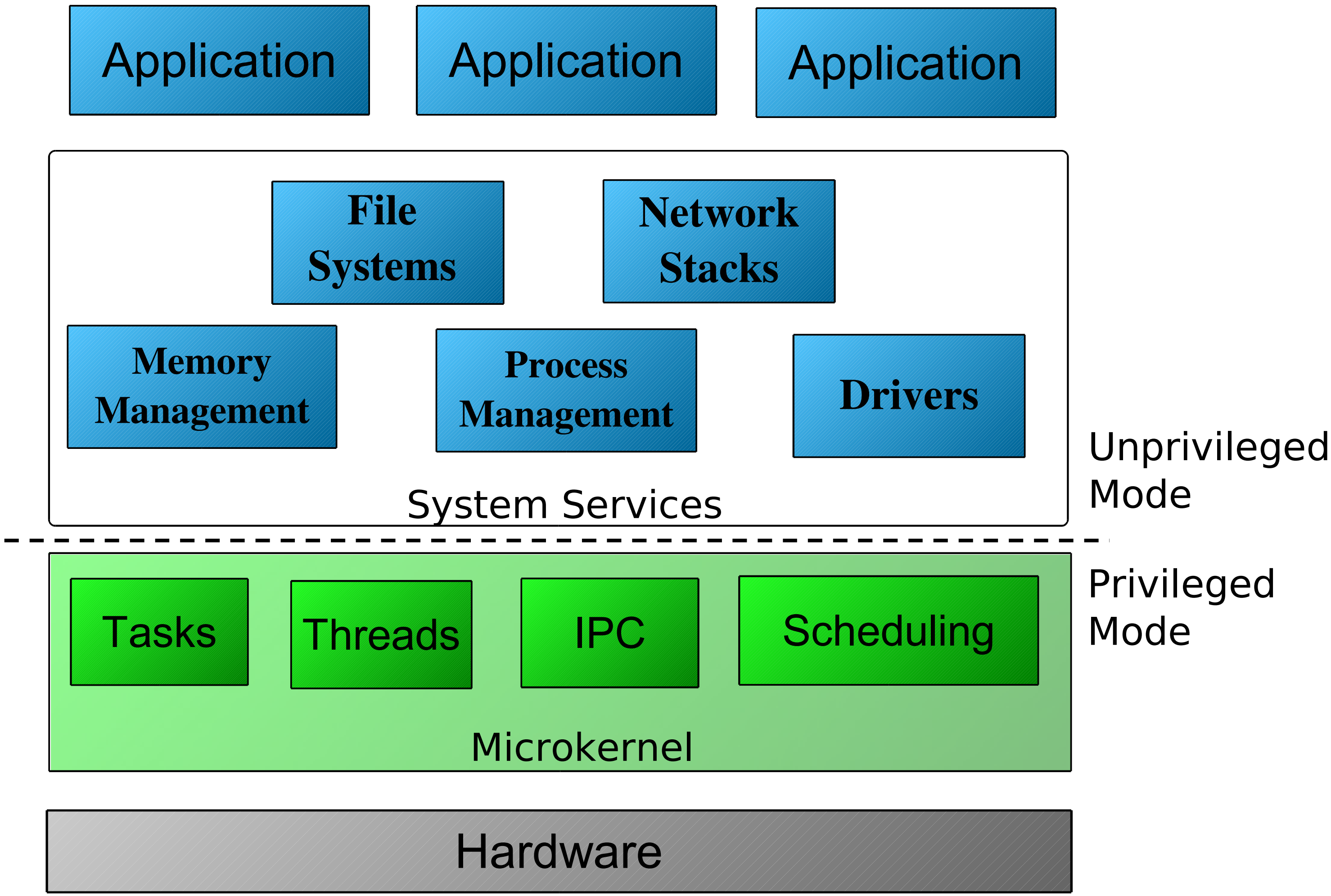
➔ No need for good system design
  - Direct access to data structures
  - Undocumented and frequently changing interfaces

➔ Big and inflexible
  - Difficult to replace system components

Why something different?

- More and more difficult to manage increasing OS complexity

Microkernel System Design



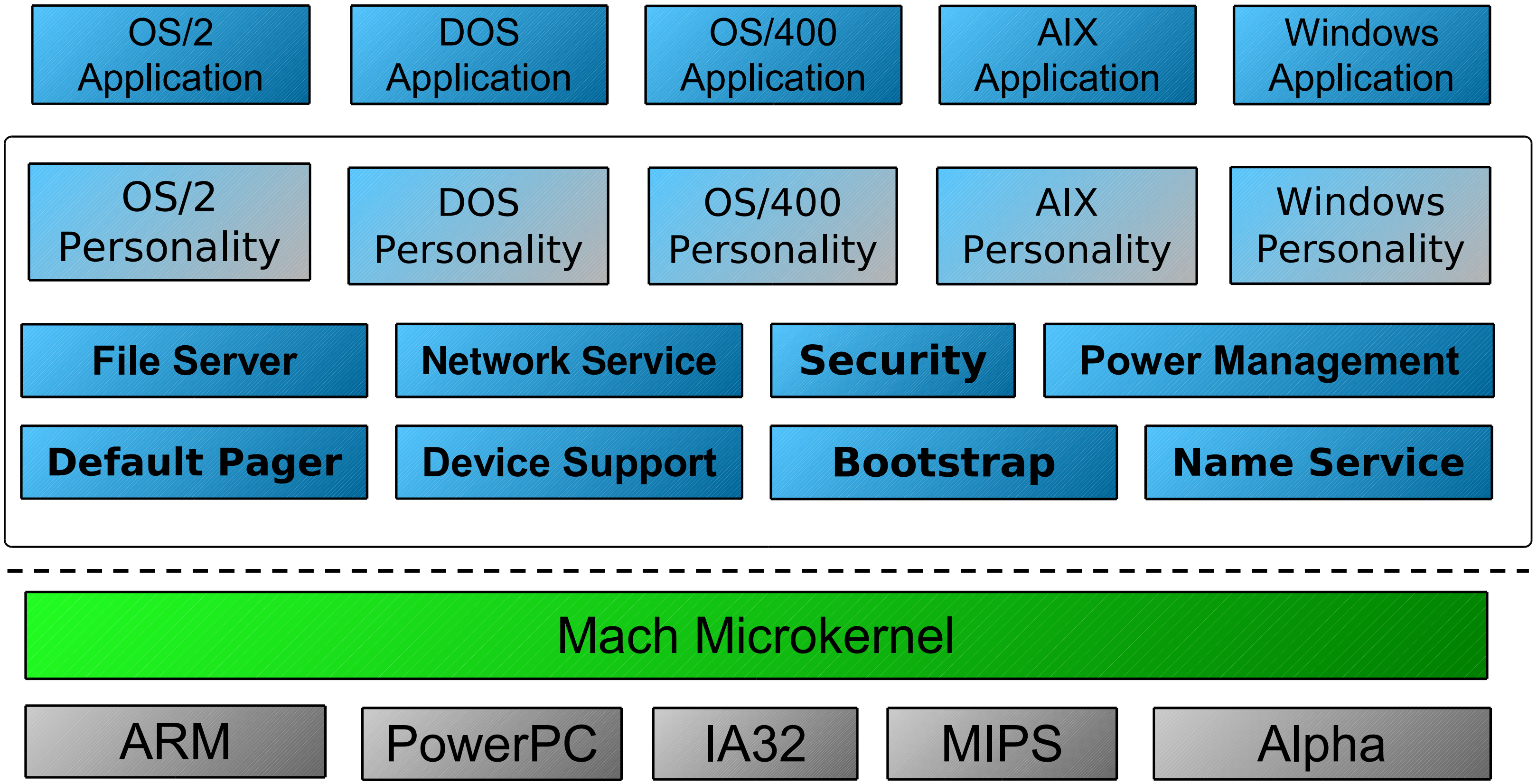
Torsten Frenzel

TU Dresden Operating Systems Group

Microkernel OS - The Vision (1)

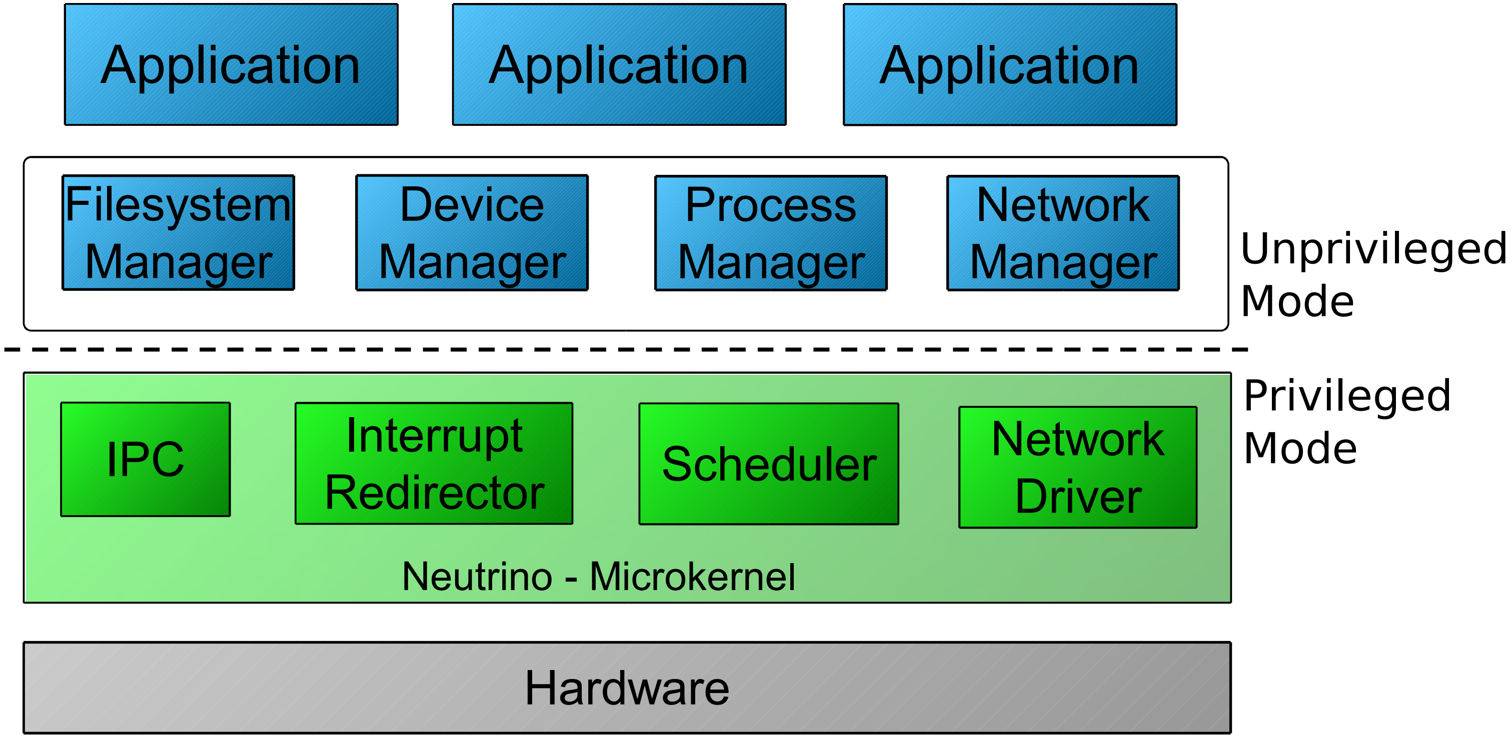
- **System components run as user-level servers**
- Protection and isolation between system components
  - More secure / safe systems
  - Less error prone
  - Small *Trusted Computing Base*
- Need for good system design
  - Well defined interfaces to system services
  - No dependencies between system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

Example – IBM Workplace OS / Mach



Example – QNX / Neutrino

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

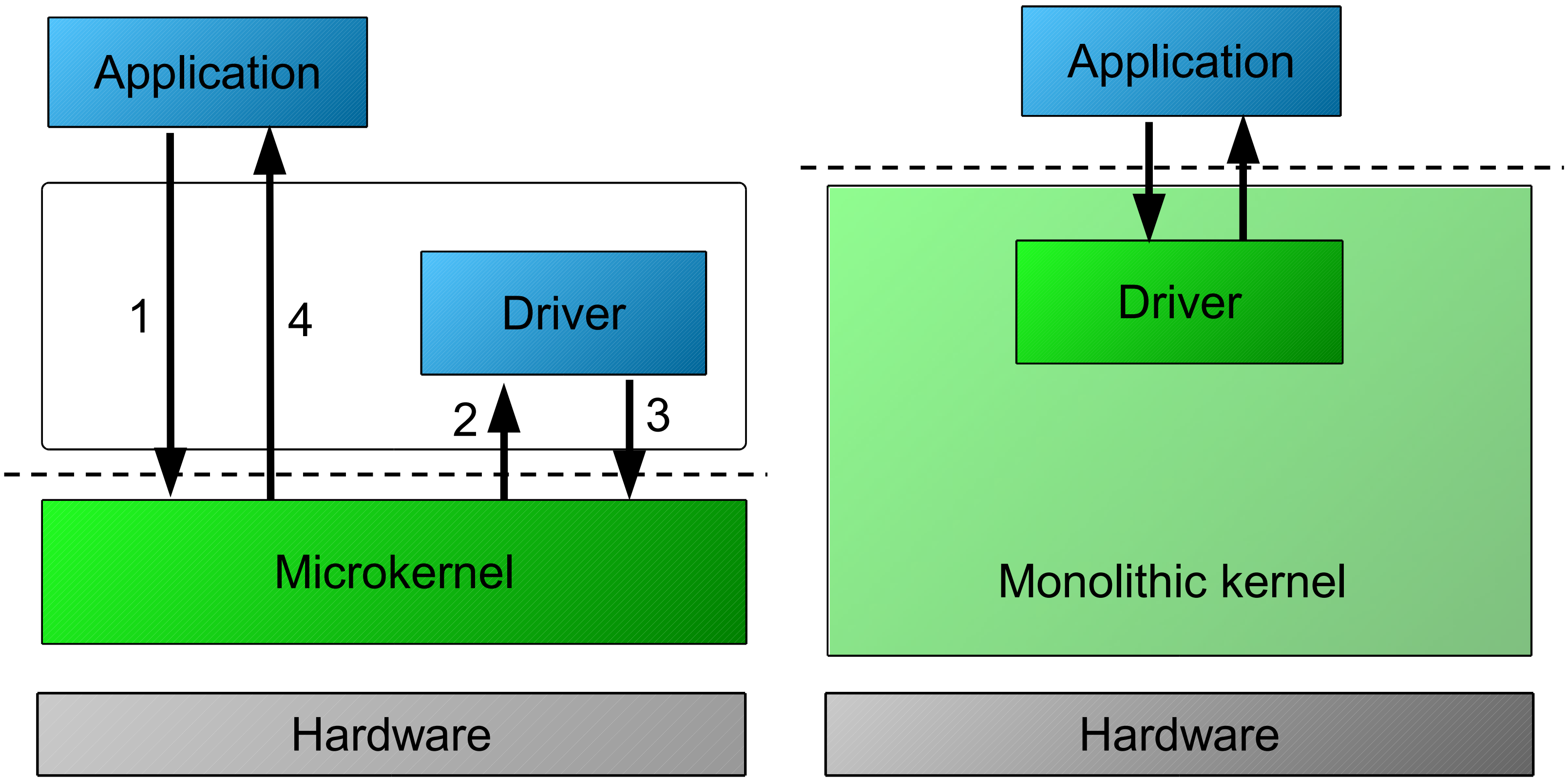


Visions vs. Reality

- 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 30.000 loc
    - Linux kernel: about 200.000 loc (without drivers)
- ✗ Performance
  - Application performance degraded
  - Communication overhead (see next slides)

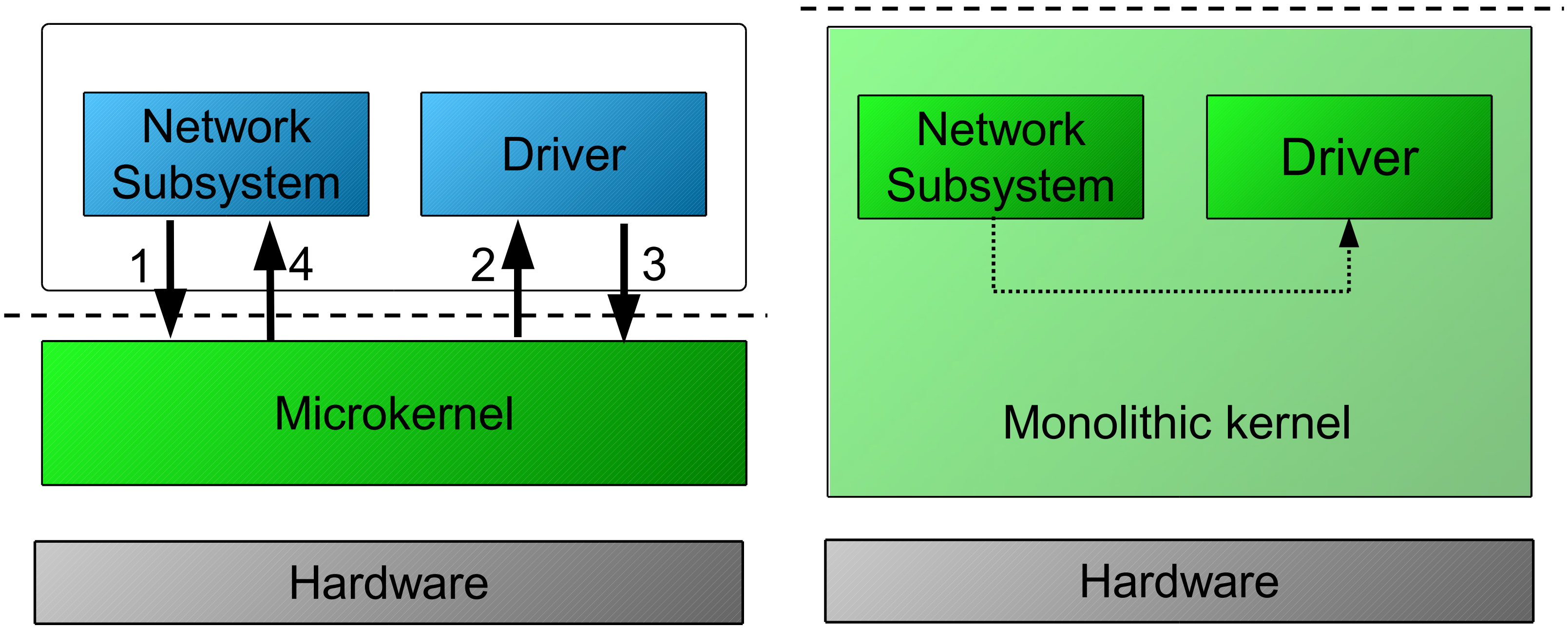
Robustness vs. Performance (1)

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



Robustness vs. Performance (2)

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



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)*

Torsten Frenzel

TU Dresden Operating Systems Group

L4 Microkernel Family

- Originally developed by Jochen Liedtke (GMD / IBM Research)
- Development continues
  - Uni Karlsruhe and UNSW Sydney (Hazelnut, Pistachio)
  - TU Dresden (Fiasco, Nova)
- Different kernel API versions:
  - V2: stable version
  - X0, X2: derived experimental versions
  - Currently many different proprietary APIs
- Support for hardware architectures:
  - **x86**: (Fiasco, Nova, Pistachio)
  - MIPS: (Pistachio)
  - ARM: (Fiasco, Pistachio)

Torsten Frenzel

TU Dresden Operating Systems Group

More Microkernels

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

Torsten
Frenzel

TU Dresden
Operating
Systems Group

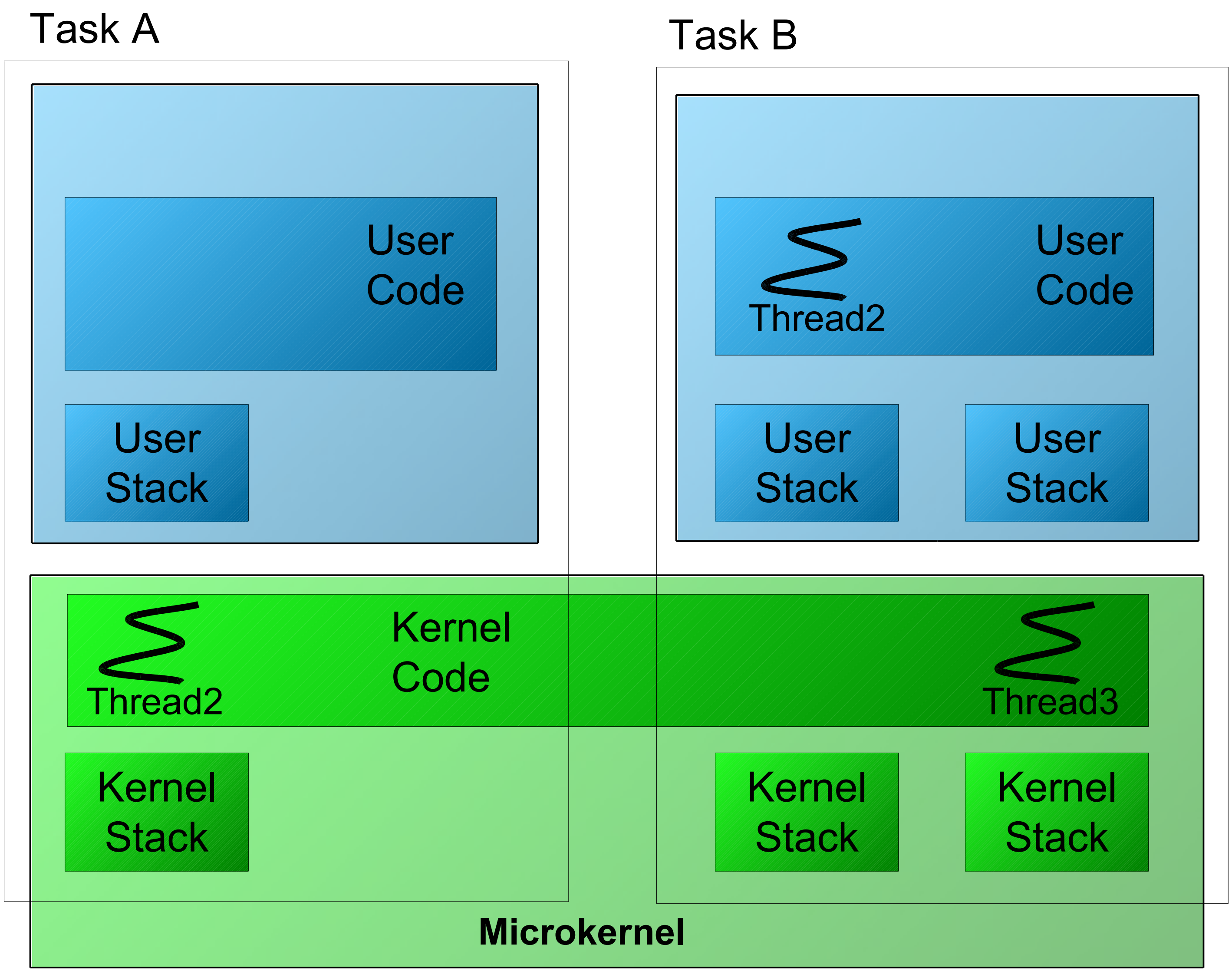
L4 - Concepts

- 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)

Torsten
Frenzel

TU Dresden
Operating
Systems Group

Threads and Tasks



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

Torsten Frenzel

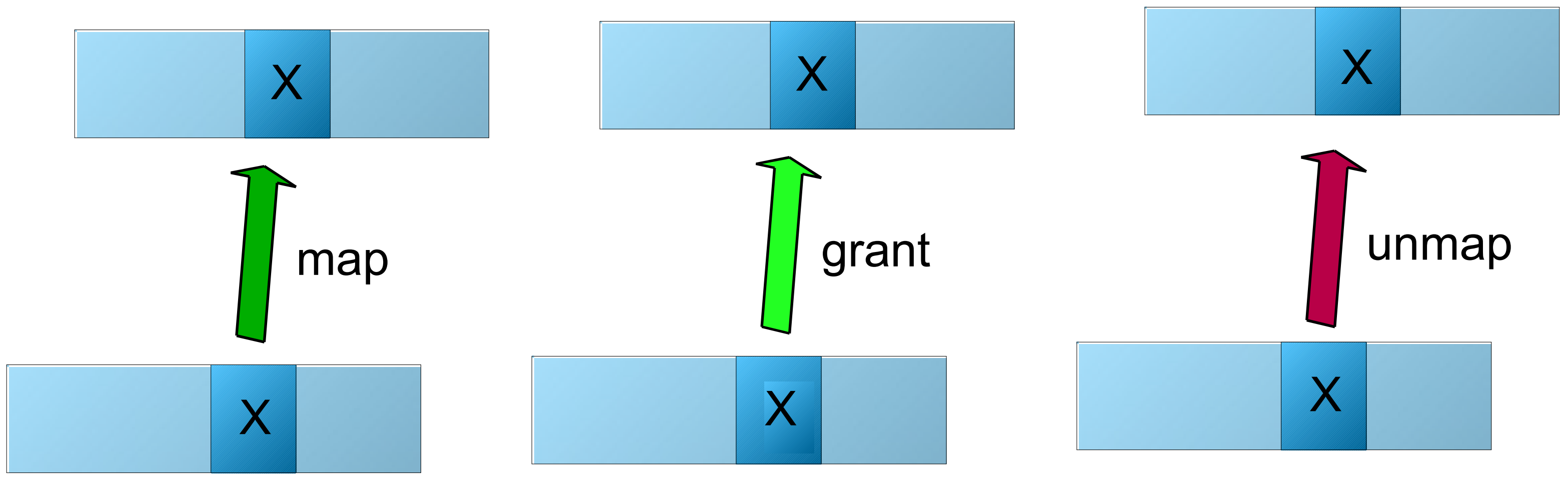
TU Dresden Operating Systems Group

Threads (2)

- Application's view:
  - Processor context (IP, SP, GPRs, FPU state) and (*user*) stack
  - Library hides implementation details
- Kernel's view:
  - Processor context (IP, SP, GPRs) and (kernel) stack
  - Object represented as Thread Control Block (TCB)
    - Saved user processor context
    - Scheduling
    - Has associated task
    - Transient state for system calls
  - Need to be created, destructed and syncronized
  - Threads can block inside the kernel and hold locks
- Basic mechanisms inside the kernel:
  - ➔ Kernel entry/exit
  - ➔ Thread switch

Tasks (1)

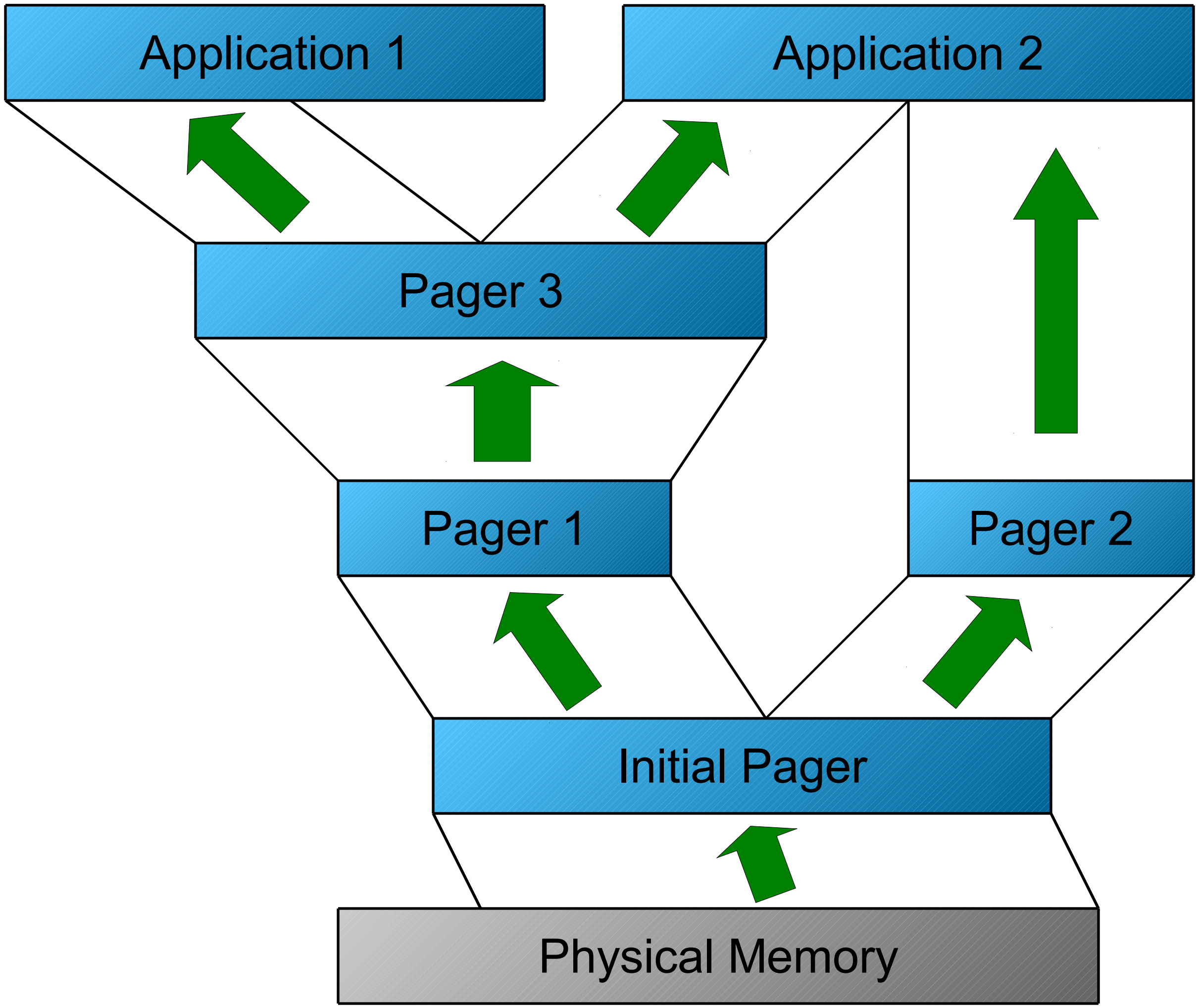
- Represent domain of protection and isolation
- Container for code, data and resources
- Address space consisting memory pages (flexpages)
- Three management operations:
  - Map: share page with other address space
  - Grant: give page to other address space
  - Unmap: revoke previously mapped page



Torsten
Frenzel

TU Dresden
Operating
Systems Group

Recursive Address Spaces



Tasks (2)

- Application's view:
  - Transparent container for code,data and resources
  - Layout is managed by the application itself or an external pager
- Kernel's view:
  - Consists of a set of page tables
  - Part is reserved for kernel code and data
  - Kernel keeps track of mapping relationship (data structure referred to as mapping database)
- Mechanisms inside the kernel
  - Insert page into an address space
  - Remove page from an address space

Torsten
Frenzel

TU Dresden
Operating
Systems Group

Communication (IPC)

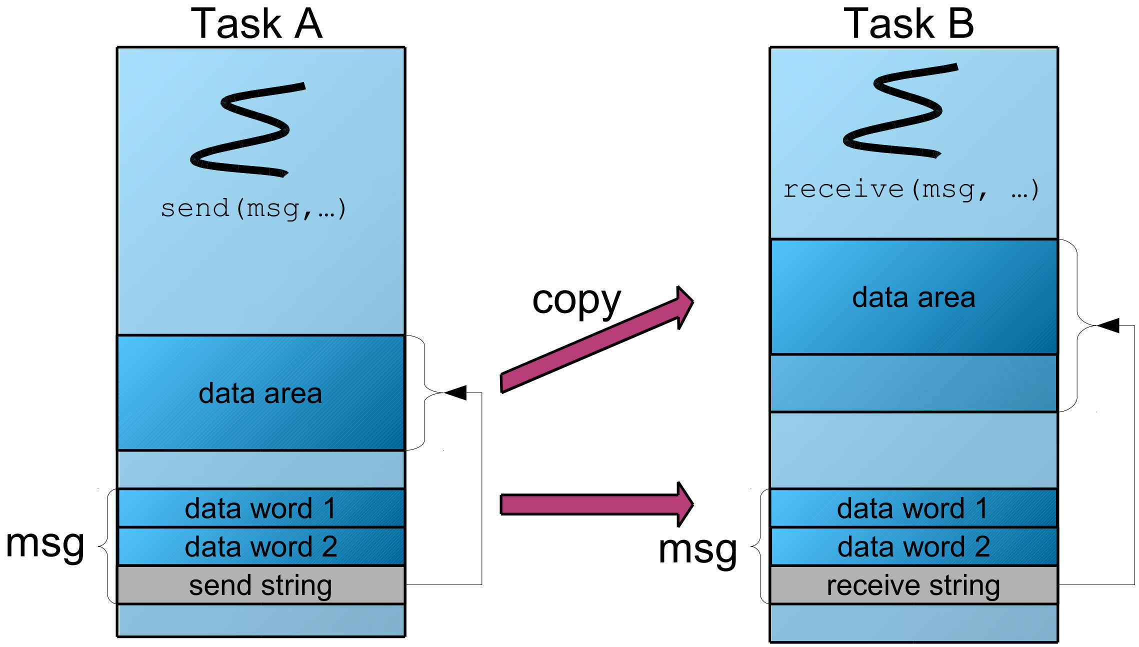
- Point-to-point reliable communication between two threads
  - Synchronous vs. asynchronous
  - Buffering vs. no buffering inside the kernel
  - Copy vs.map data
  - Direct vs. indirect IPC
  - With/without timeouts
- IPC types
  - Send (to one thread)
  - Receive from one thread (closed receive)
  - Receive from any thread (open receive)
  - Call (send and closed receive)
  - Reply and wait (send and open receive)

Torsten
Frenzel

TU Dresden
Operating
Systems Group

Copy-Data Message

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

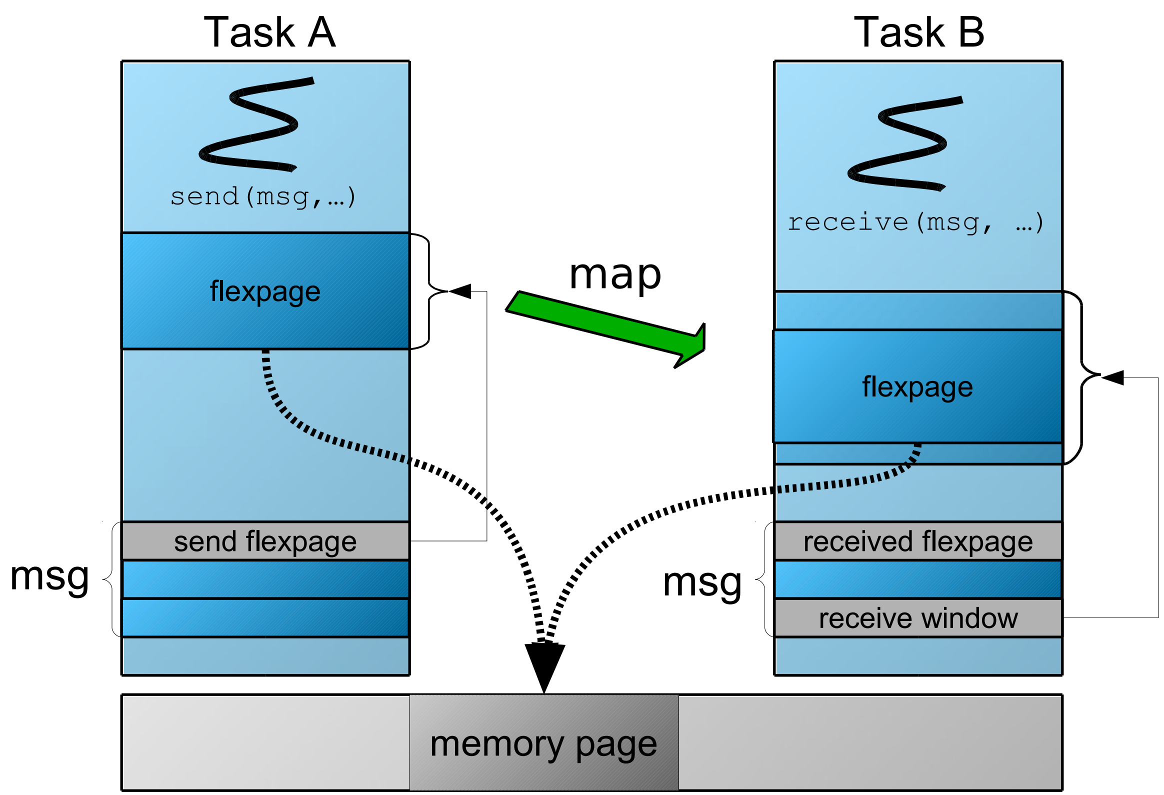


Torsten
Frenzel

TU Dresden
Operating
Systems Group

Map-Data Message

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



Scheduling

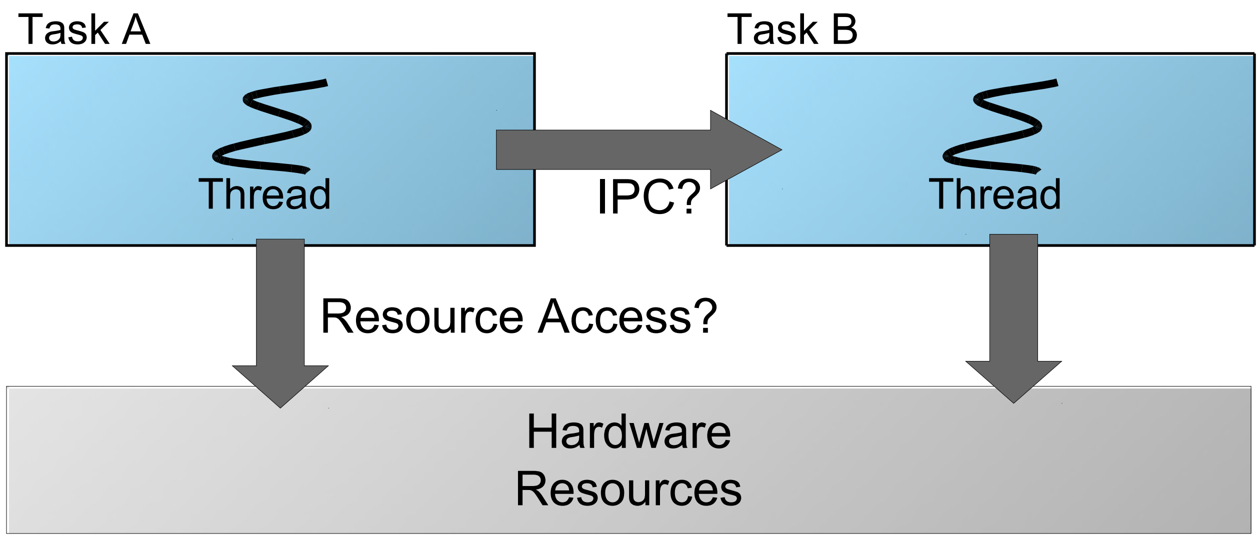
- Scheduling contexts represent scheduling entities
  - Has priority and time quantum
  - One thread can have one or more scheduling context
  - One best-effort timeslice context in system
- Scheduling mechanism
  - Round-robin scheduler with fixed priorities
  - Thread with highest priority is selected
  - L4 supports 256 priorites
  - Scheduler has complexity O(1)
- Realtime extension
  - Mechanisms to avoid priority inversion
  - Reservation scheduling contexts with periods
  - Additional syscalls

Torsten
Frenzel

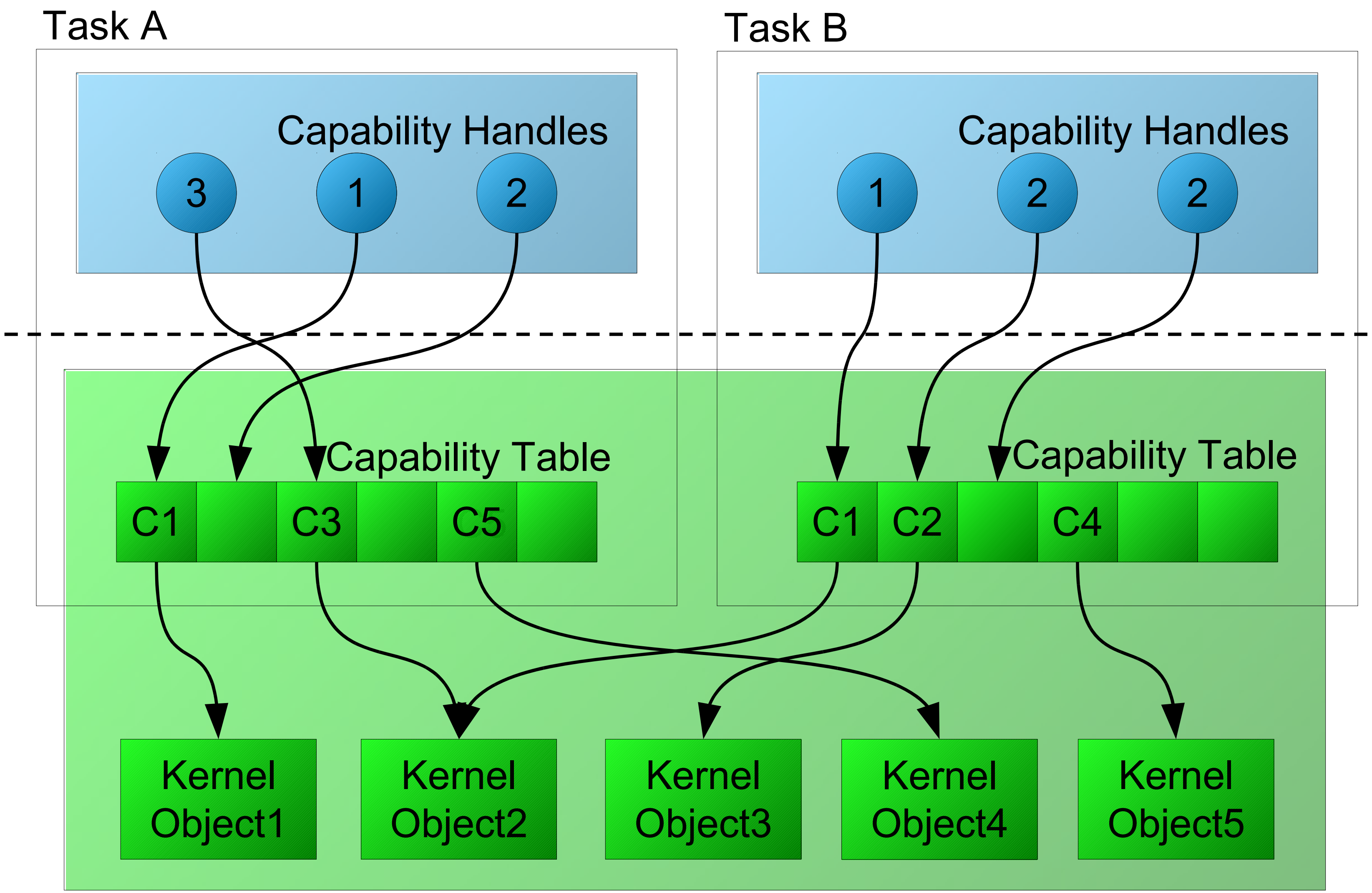
TU Dresden
Operating
Systems Group

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



Kernel-Object Capabilities



Torsten
Frenzel

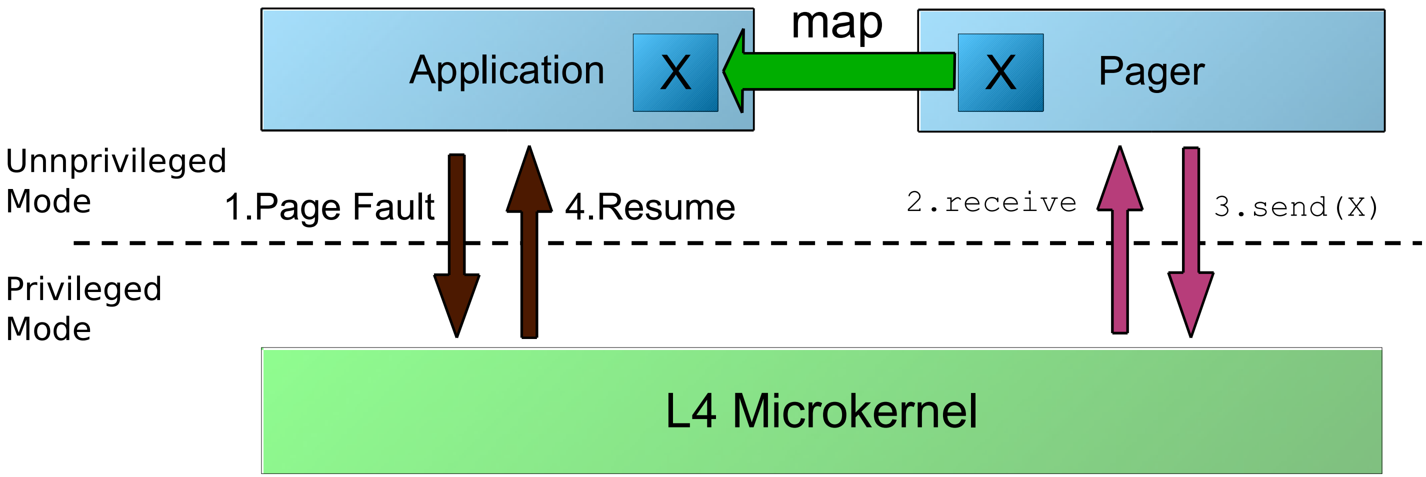
TU Dresden
Operating
Systems Group

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

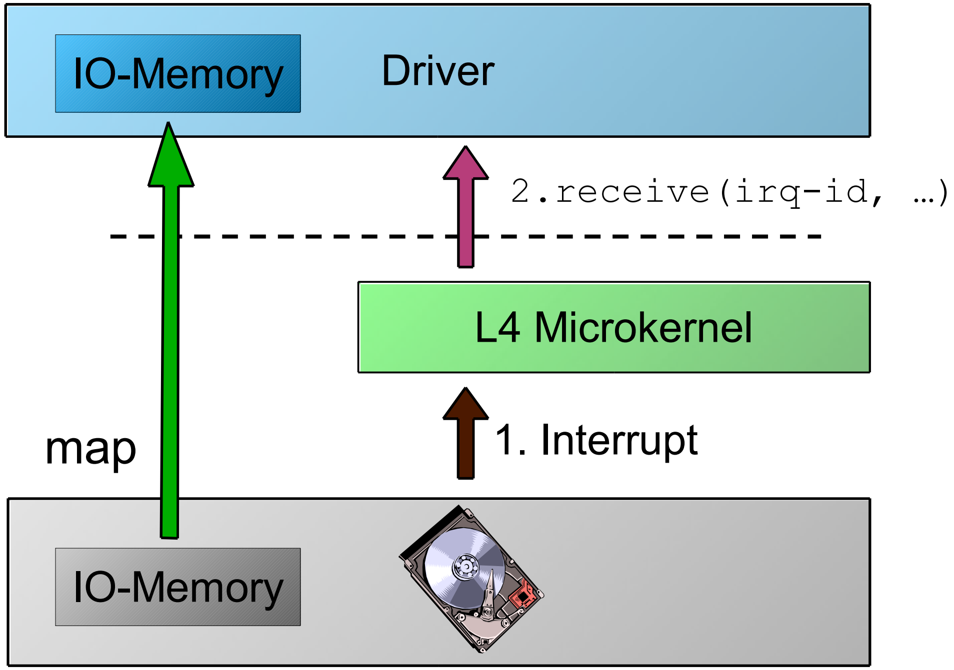


Torsten
Frenzel

TU Dresden
Operating
Systems Group

Device Drivers

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



Example: L4V2 API

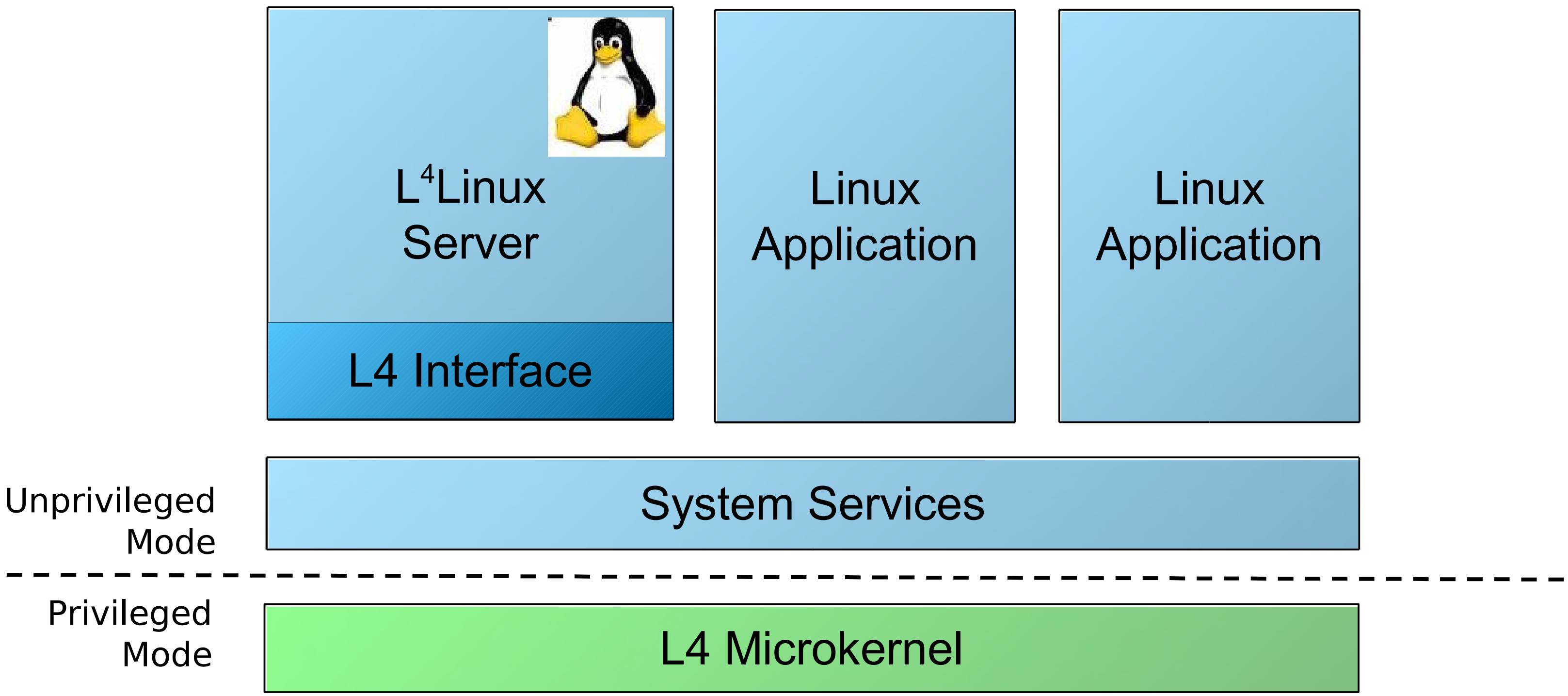
- Address Spaces
  - `l4_task_new` create / delete address spaces
- Threads
  - `l4_thread_ex_regs` create / modify threads
  - `l4_thread_schedule` modify scheduling parameter
  - `l4_thread_switch` switch to a different thread
- IPC
  - `l4_ipc` send / receive date, map flexpage
  - `l4_fpage_unmap` unmap flexpage
  - `l4_nchief` return nearest communication partner

Torsten
Frenzel

TU Dresden
Operating
Systems Group

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

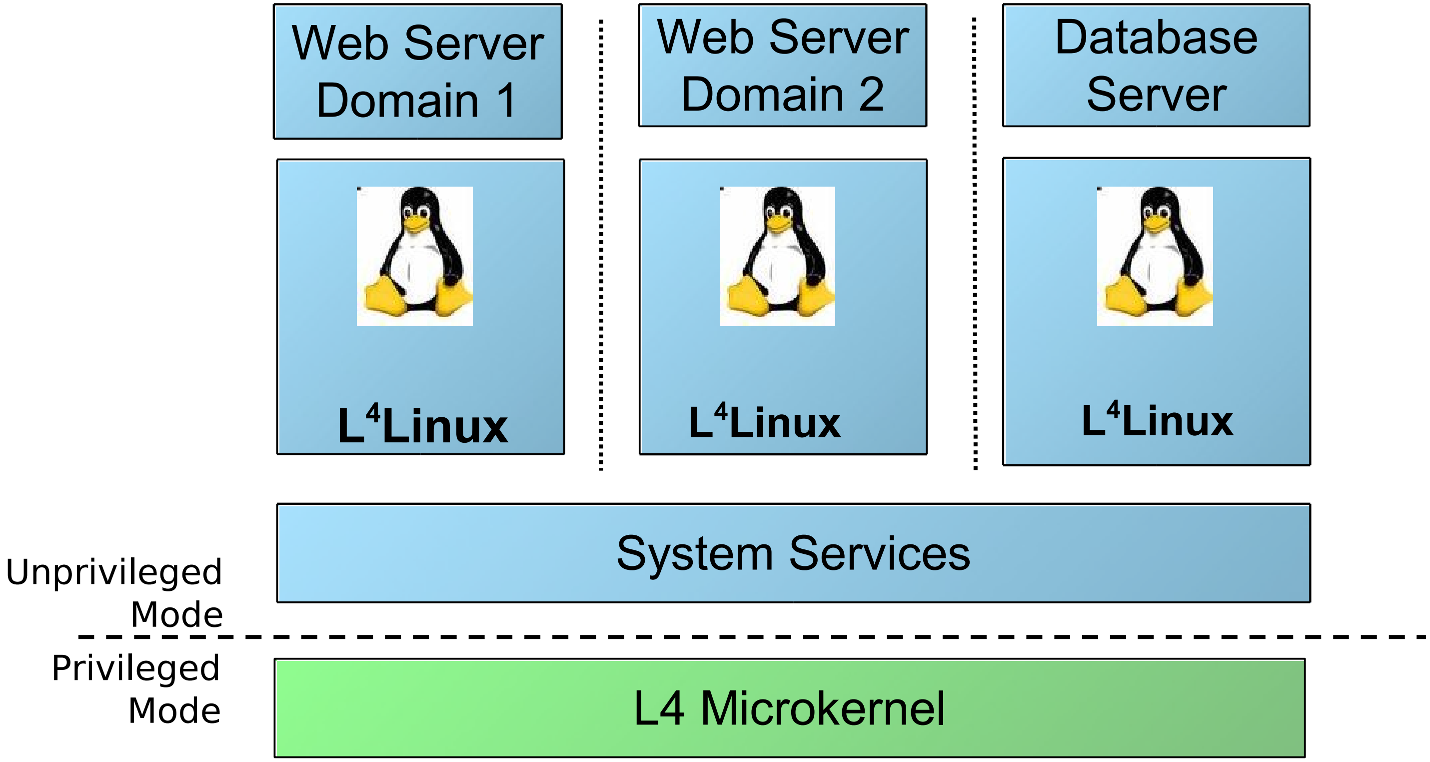


Torsten
Frenzel

TU Dresden
Operating
Systems Group

L4 Applications - Virtual Machines

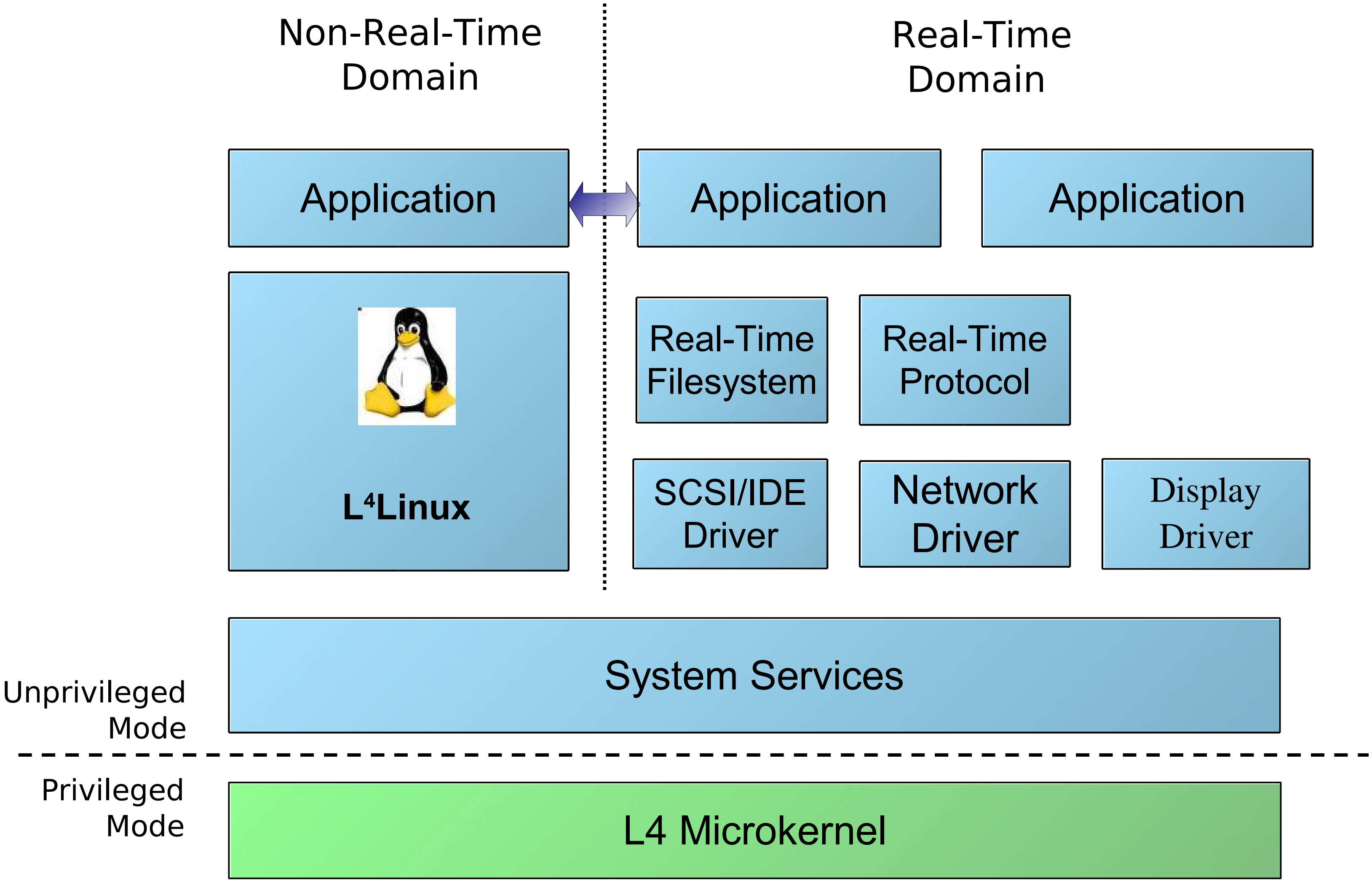
- Several isolated OSes on top of a single physical machine
- Used for server consolidation



Torsten
Frenzel

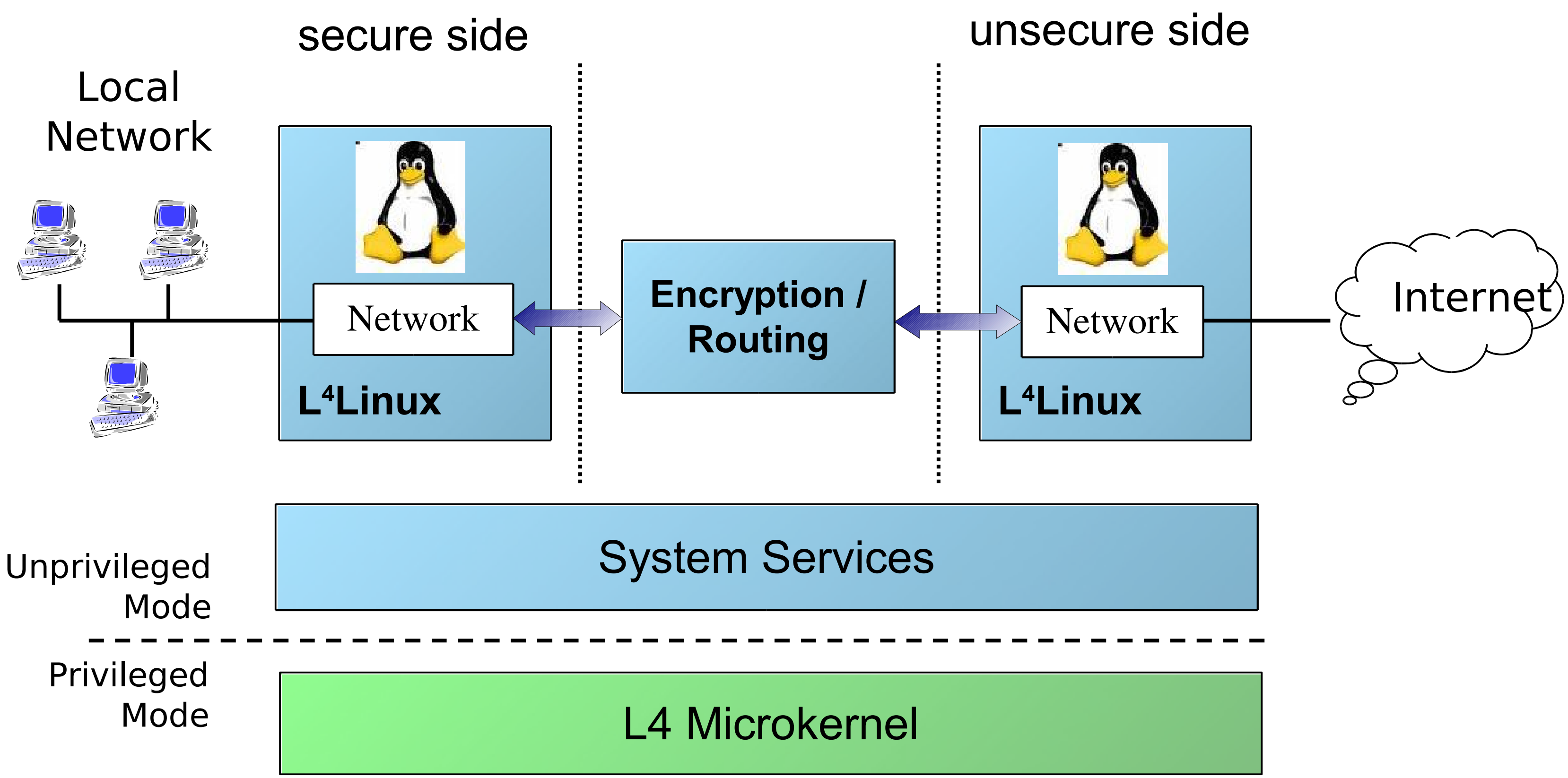
TU Dresden
Operating
Systems Group

L4 Applications - DROPS



L4 Application - µSINA

VPN Gateway



Lecture Outline

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

Torsten
Frenzel

TU Dresden
Operating
Systems Group

Practical Excercises

- Guide to build own very small kernel
- Thinking about design and implementation
  - Threads and thread switches
  - Kernel entry/exit
  - Syscalls and Interrupts
  - Address spaces and memory management
  - Device programming
- Based on x86 architecture
- Qemu as test platform

Torsten Frenzel

TU Dresden Operating Systems Group

Next: Address spaces and Threads

- Implemenation of address space
- Threads and Thread control blocks (TCBs)
- Tasks
- Page tables
- Thread and task switching
- FPU switching