Karlsruhe Institute of Technology

# Microkernel Construction
## I.1 – Introduction, Motivation, Problems

**Lecture Summer Term 2017**
**Wednesday 15:45-17:15 R131, 50.34 (INFO)**

Jens Kehne, Marius Hillenbrand
Operating Systems Group, Department of Computer Science

# Staff

- Jens Kehne
  - PhD student since June 2012
  - Email: kehne@kit.edu

- Marius Hillenbrand
  - PhD student since July 2011
  - Email: marius.hillenbrand@kit.edu

- Meeting Times
  - By arrangement via e-mail
  - Bldg. 50.34, Room 155 / 160

# Background

- L4Ka project (http://l4ka.org)
- Fiasco.OC (http://os.inf.tu-dresden.de/fiasco/)
- seL4 (http://seL4.systems)
- OKL4 (http://www.ok-labs.com/products/okl4-microvisor)

- Kevin Elphinstone and Gernot Heiser: "FromL3 to seL4: What Have We Learnt in 20 Years of L4 Microkernels?"

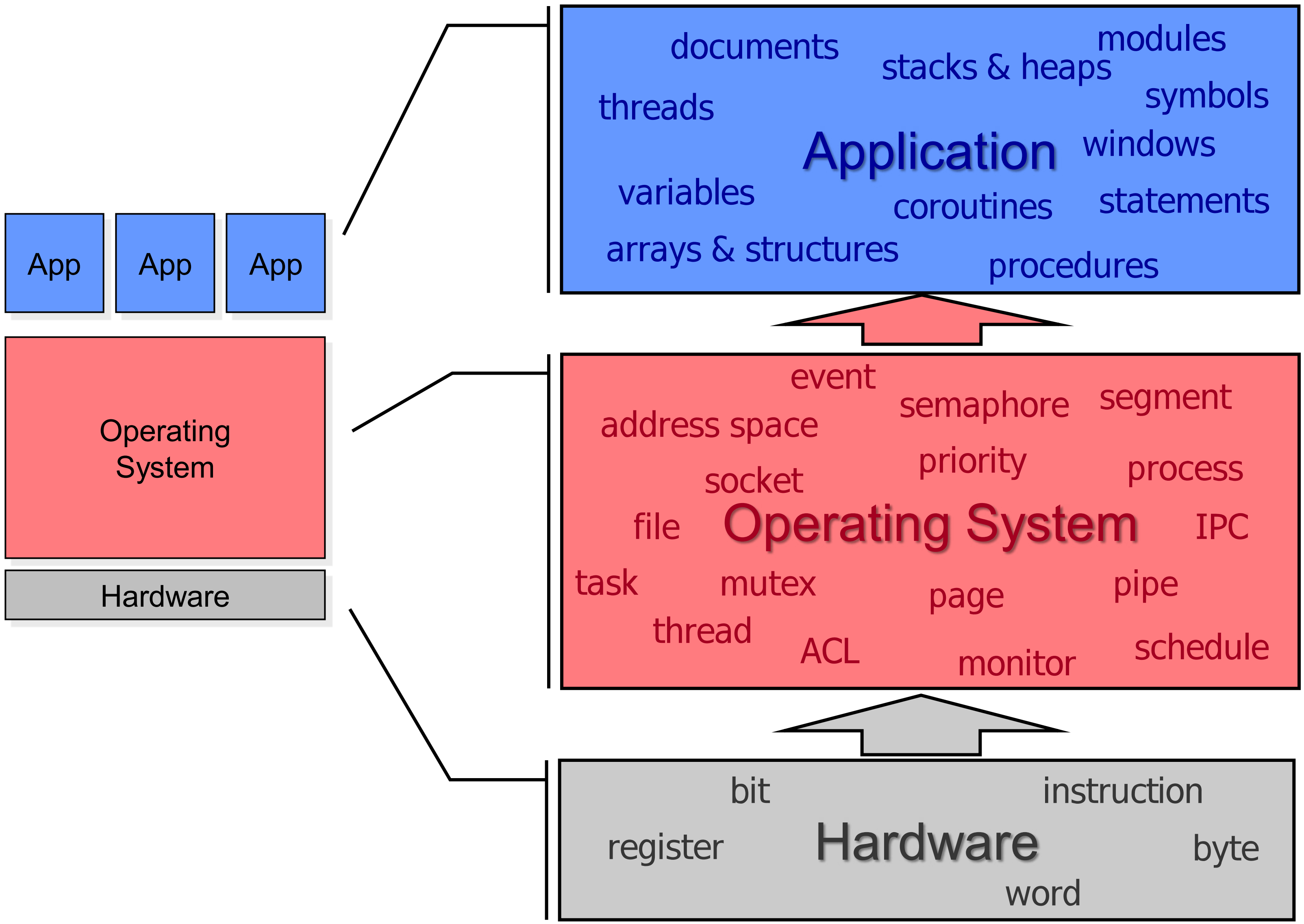
- Slides in Ilias
  - Password: MKCSS17

# Purpose of Operating Systems

- Every computer runs an OS

- Abstract from the hardware
  - Interrupts, exceptions
- Provide common services
  - Protection (process)
  - Execution (thread)
  - Device/Resource management (socket)
  - Persistence of data (file)
- Bridge semantic gap
  - Application demands vs. hardware provides

# Purpose


App
App
App
Operating System
Hardware
documents
stacks & heaps
modules
symbols
threads
Application
windows
variables
coroutines
statements
arrays & structures
procedures
event
semaphore
segment
address space
priority
process
socket
file
Operating System
IPC
task
mutex
page
pipe
thread
ACL
monitor
schedule
bit
instruction
register
Hardware
byte
word

# Operating System Designs

μ-kernel with object interfaces

- ✓ Standard interface
- ✓ User-defined interfaces
- ✓ Runs subsystems from different vendors
- ✓ Good flexibility
- ✓ Good minimalism
- ✓ Good performance
- ✗ Difficult to use
- ✗ Different paradigm

application-specific

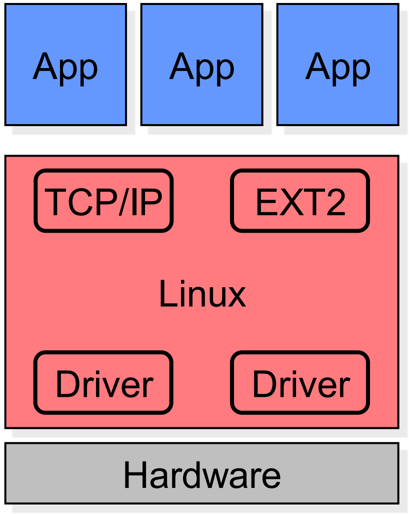
- ✓ Ultimate flexibility
- ✓ Ultimate minimalism
- ✓ Ultimate performance
- ✓ Normal programming paradigm
- ✗ Proprietary and incompatible solutions

monolithic

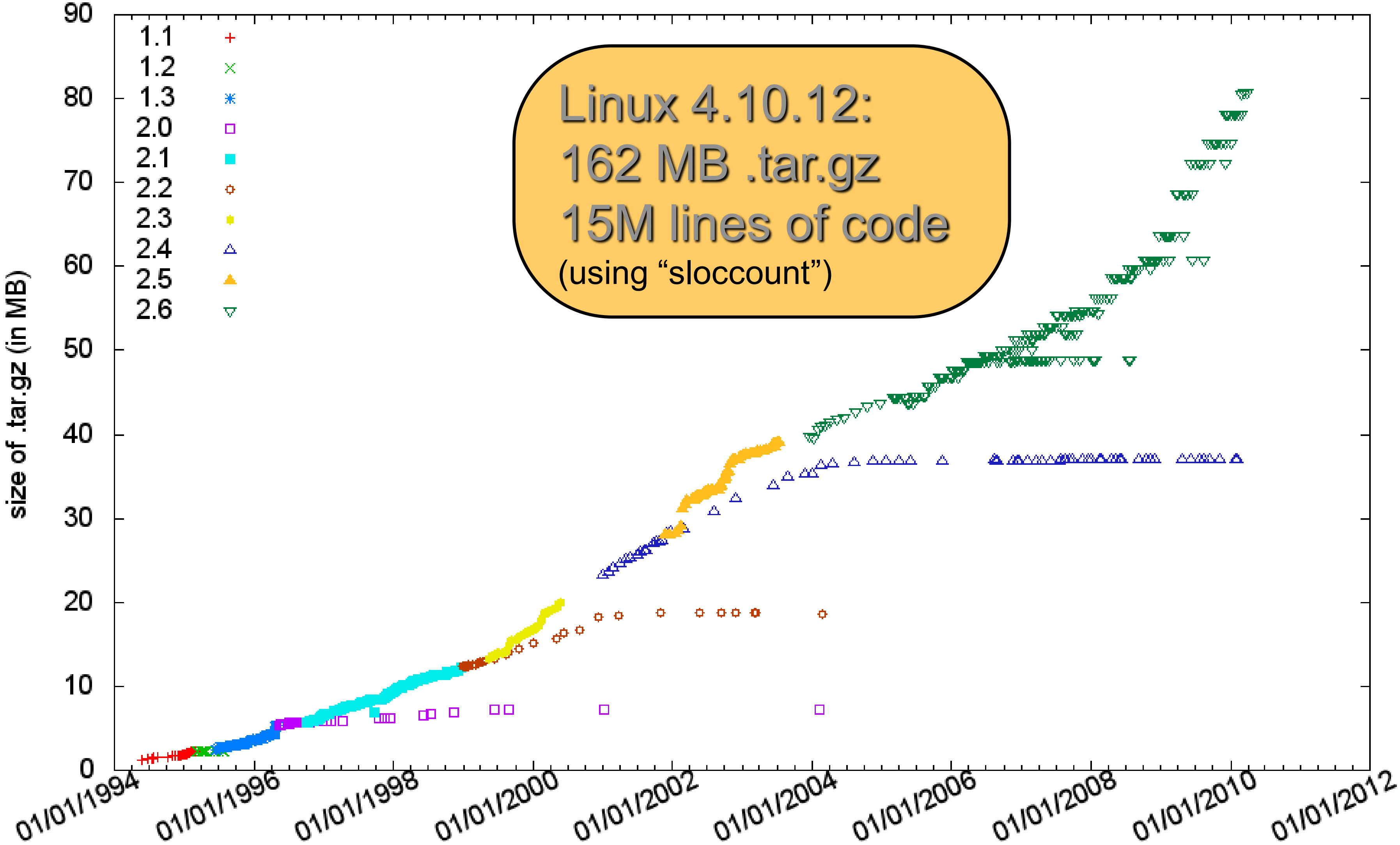
- ✓ Standard interface
- ✓ Runs programs from different vendors
- ✗ Limited extensibility
- ✗ Limited flexibility

# Monolithic Kernels – Advantages

- Kernel has access to everything
  - All optimizations are possible
  - All techniques/mechanisms/concepts can be implemented
- Kernel extended by adding more code



# Linux Kernel Evolution (.tar.gz)

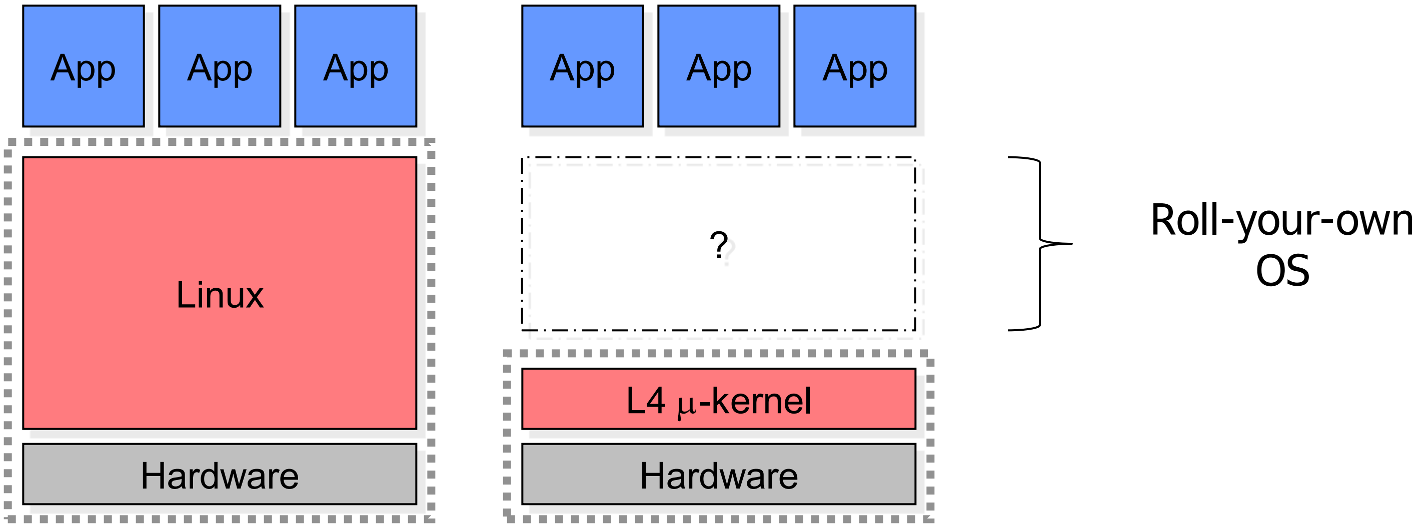


# Approaches to Tackling Complexity

- Monolithic approaches
  - Layered Kernels
  - Modular Kernels
  - Object Oriented Kernels
- Alternatives
  - Extensible Kernels
  - Managed kernels
  - Hypervisors
  - Microkernels


Operating Systems Group
Department of Computer Science

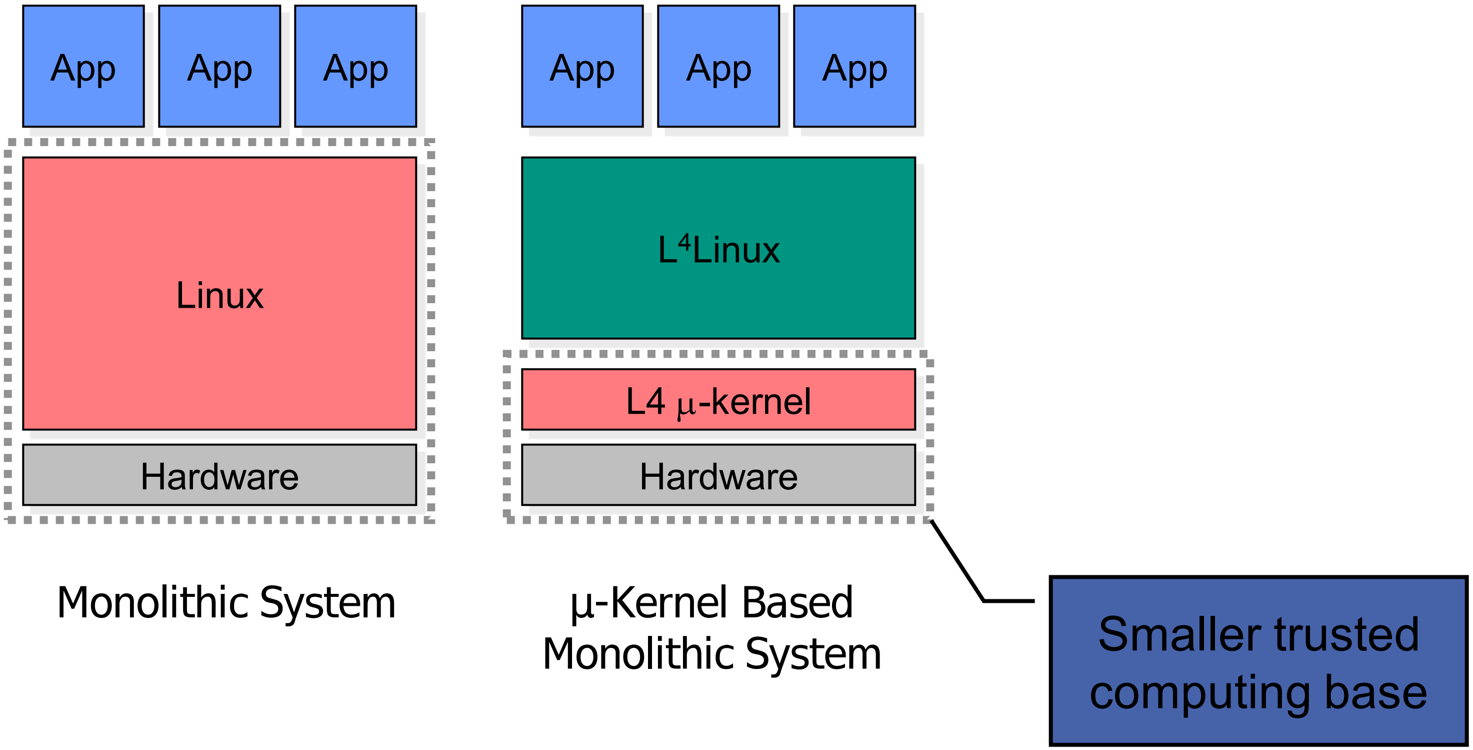
# μ-Kernel Based Systems




Monolithic System

μ-Kernel Based System

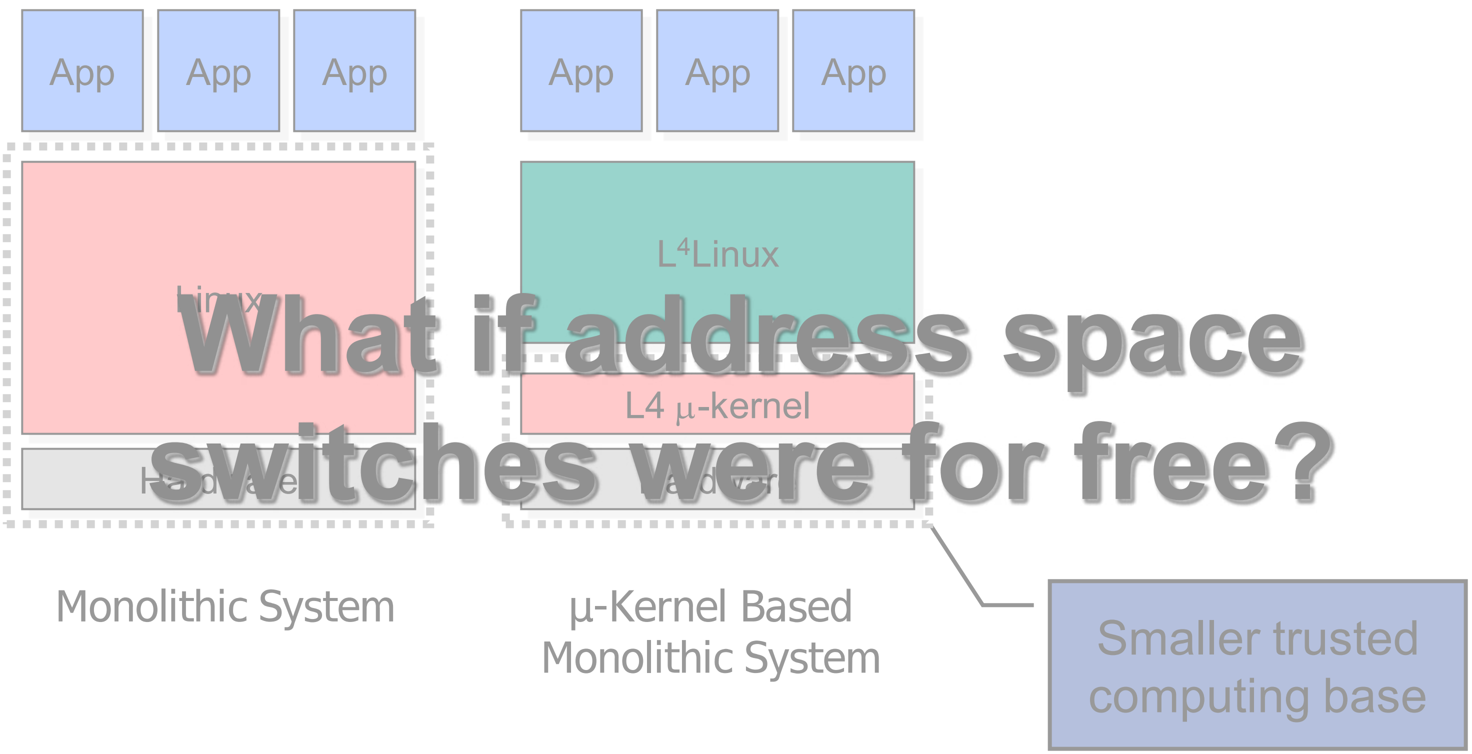
# µ-Kernel Based Systems





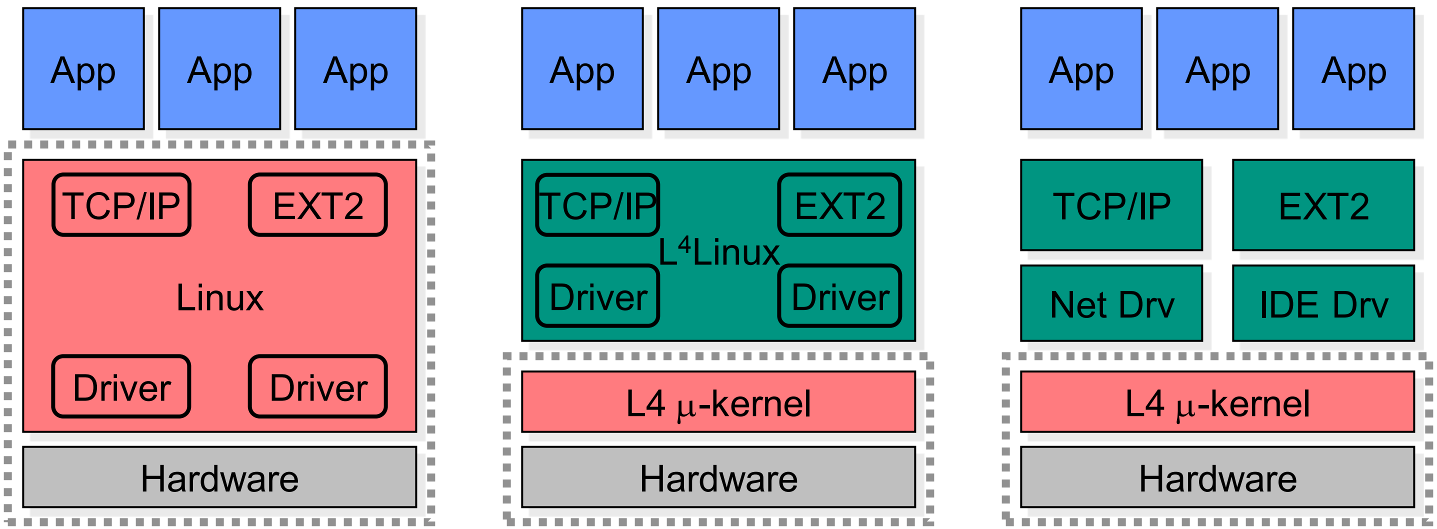

Operating Systems Group
Department of Computer Science

# µ-Kernel Based Systems




**What if address space switches were for free?**

# µ-Kernel Based Systems

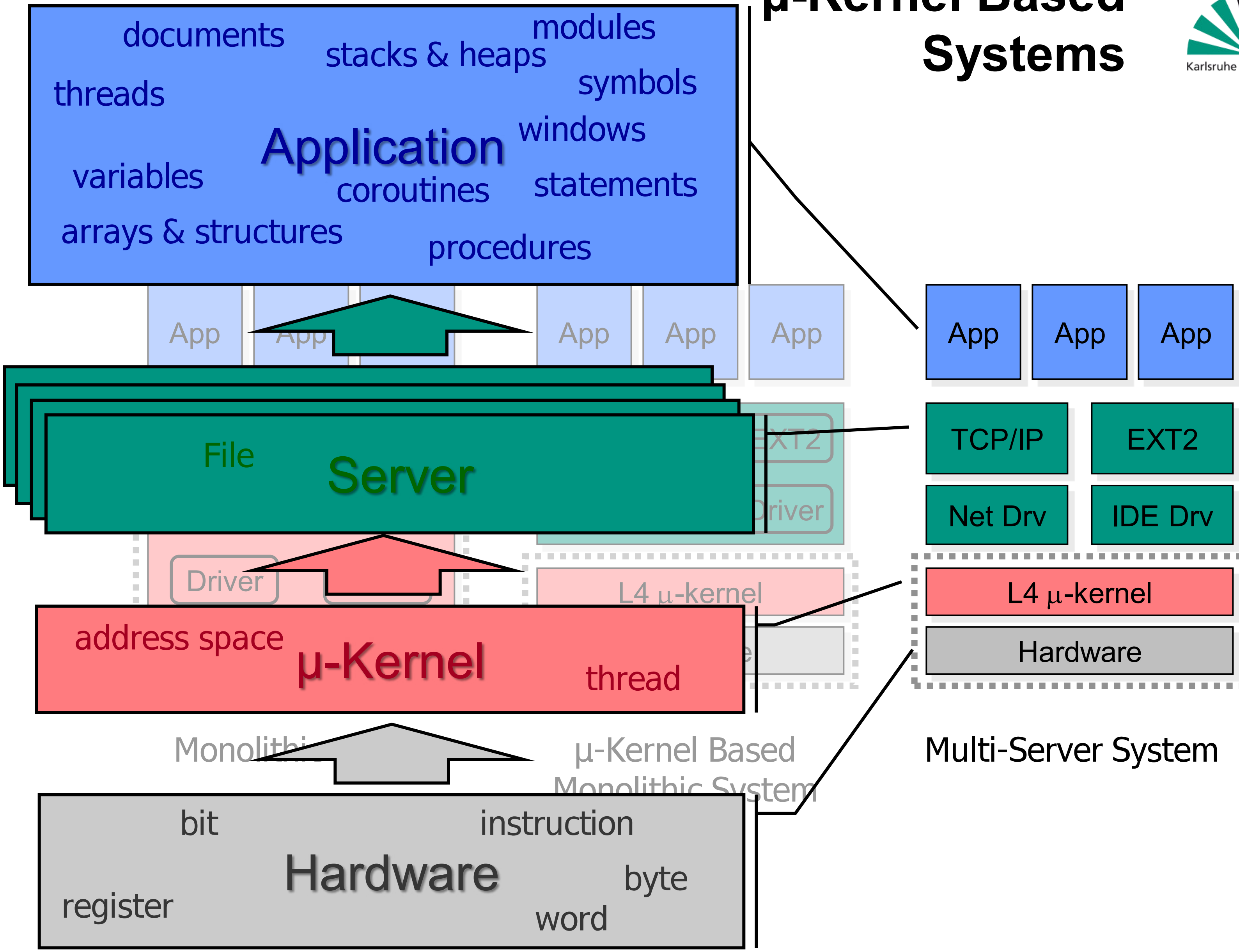



Monolithic System

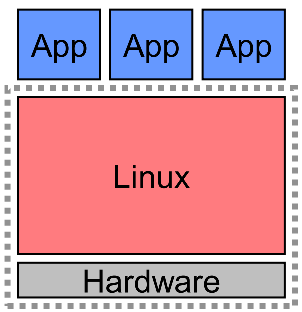
µ-Kernel Based Monolithic System

Multi-Server System

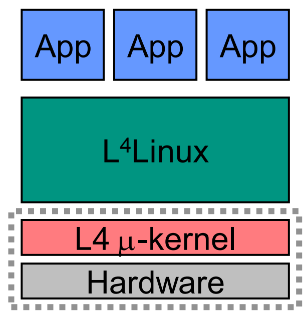
# µ-Kernel Based Systems



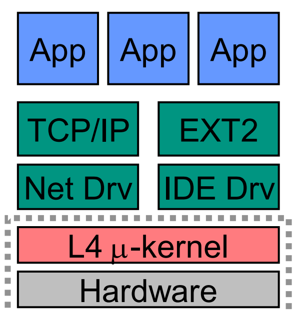
# Microkernel Based Systems



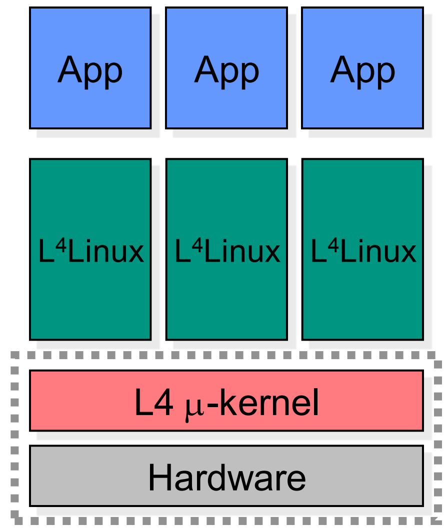

Monolithic System




μ-Kernel Based
Monolithic System

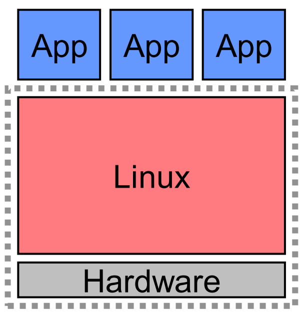



Multi-Server System

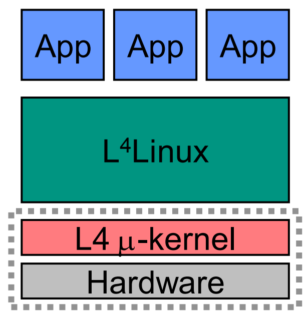



μ-Kernel Based
Server Consolidation

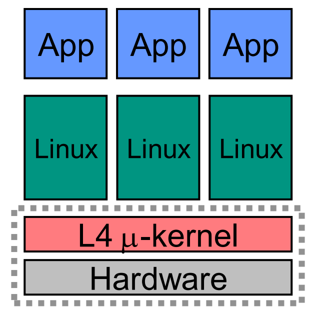
# Microkernel Based Systems



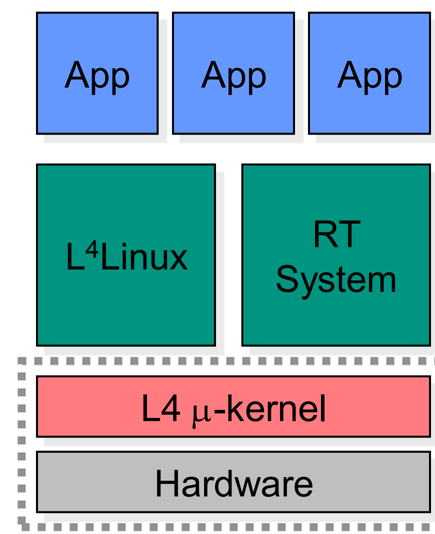

Monolithic System



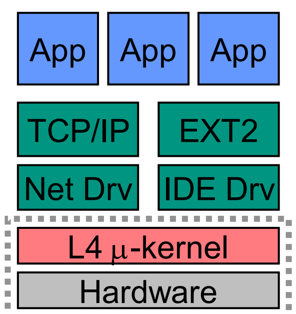

μ-Kernel Based
Monolithic System




μ-Kernel Based
Server Consolidation

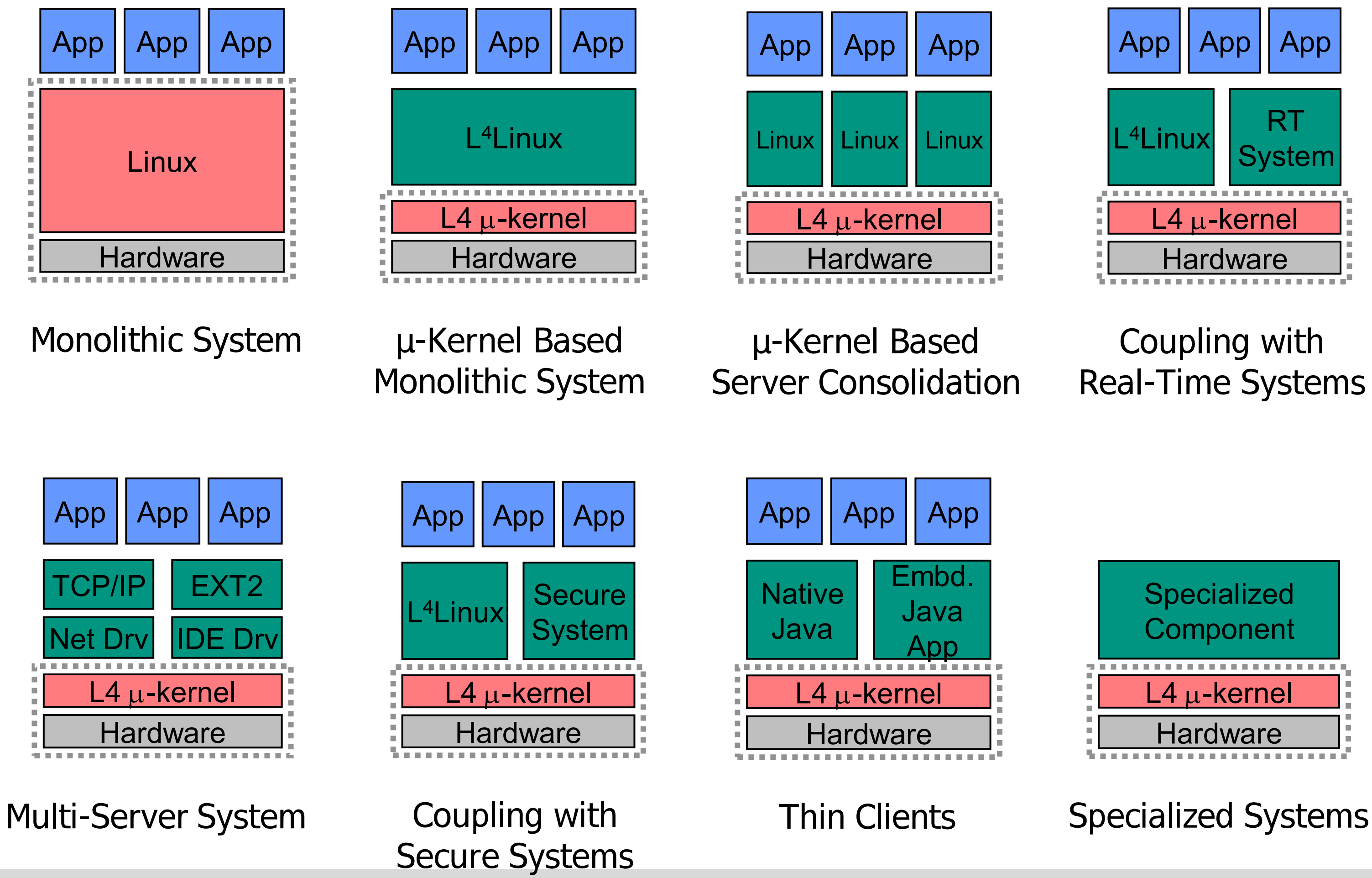



Coupling with
Real-Time Systems




Multi-Server System

# Microkernel Based Systems





Operating Systems Group
Department of Computer Science

# History

- Monolithic kernels
- 1st-generation µ-kernels
  - Mach *CMU, OSF* **External Pager**
  - Amoeba *Vrije Universiteit*
  - (L3) *GMD* **User-Level Driver**
- 2nd-generation µ-kernels
  - Exokernel *MIT*
  - L4Ka::Pistachio *GMD / IBM / UKa* **Recursive Address Spaces**
- 3rd-generation µ-kernels
  - Fiasco.OC *TU Dresden* **Object capability system**
  - seL4 *UNSW/NICTA* **Formal verification**


Operating Systems Group
Department of Computer Science

## The Great Promise

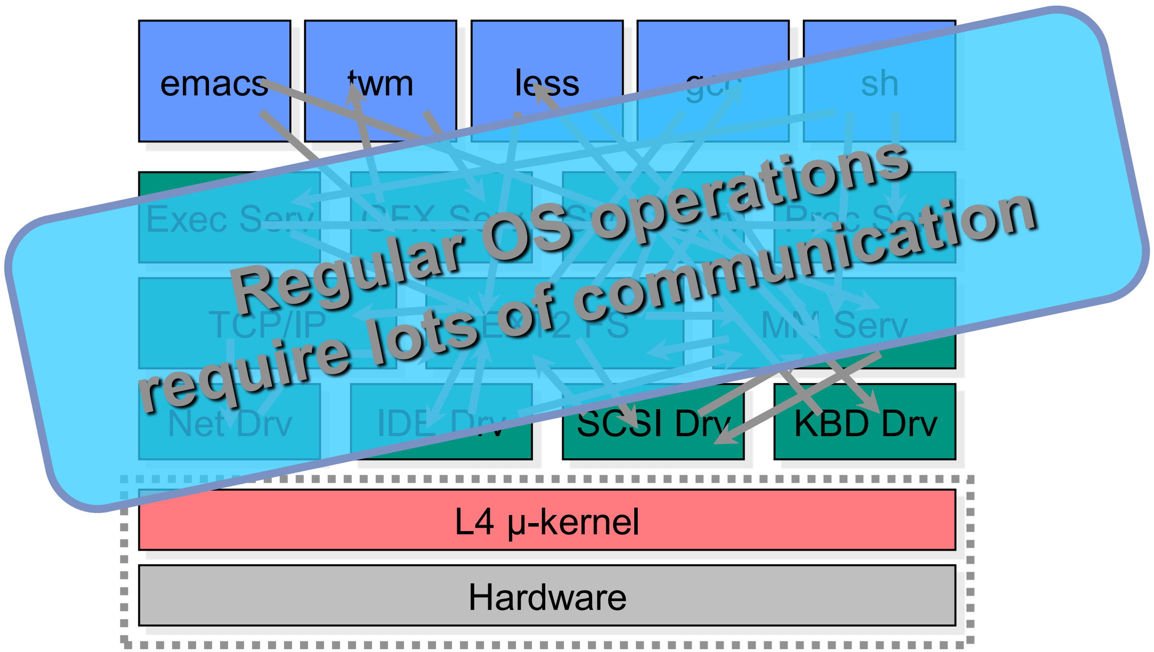
# The Big Disaster

- ~~Coexistence of different~~
  - ~~APIs~~
  - ~~File systems~~
  - ~~OS personalities~~
- ~~Flexibility~~
- ~~Extensibility~~
- ~~Simplicity~~
- ~~Maintainability~~
- ~~Security~~
- Safety

- *SLOW*
- *INFLEXIBLE*
- *LARGE*

**IBM WorkPlace OS:**
**~2,000,000,000 US$**

# Microkernel Based Systems: The Challenge

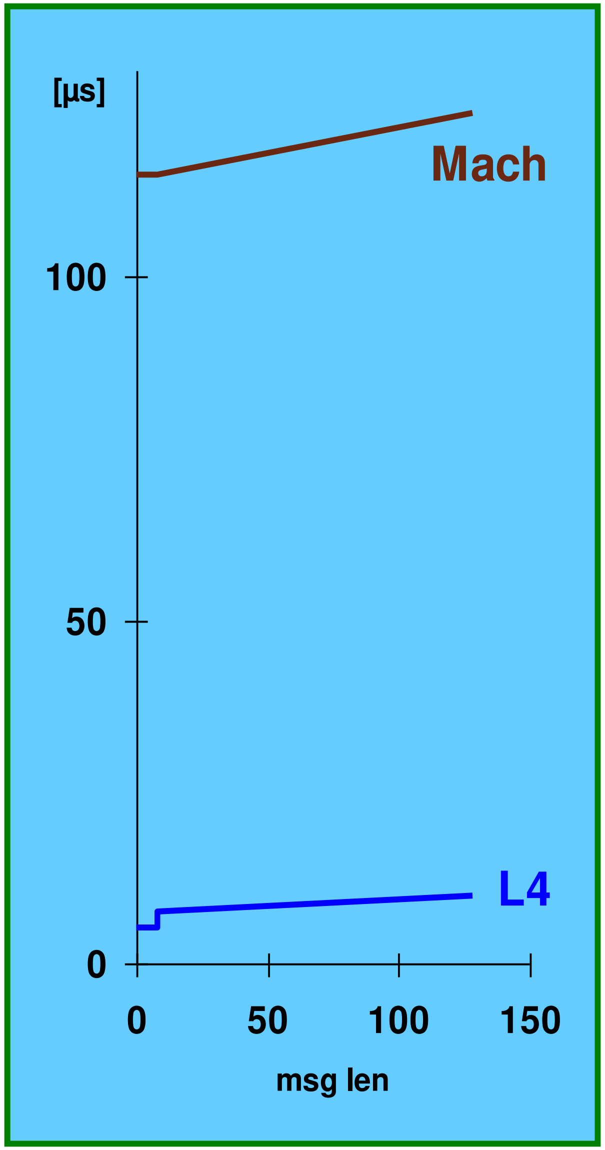


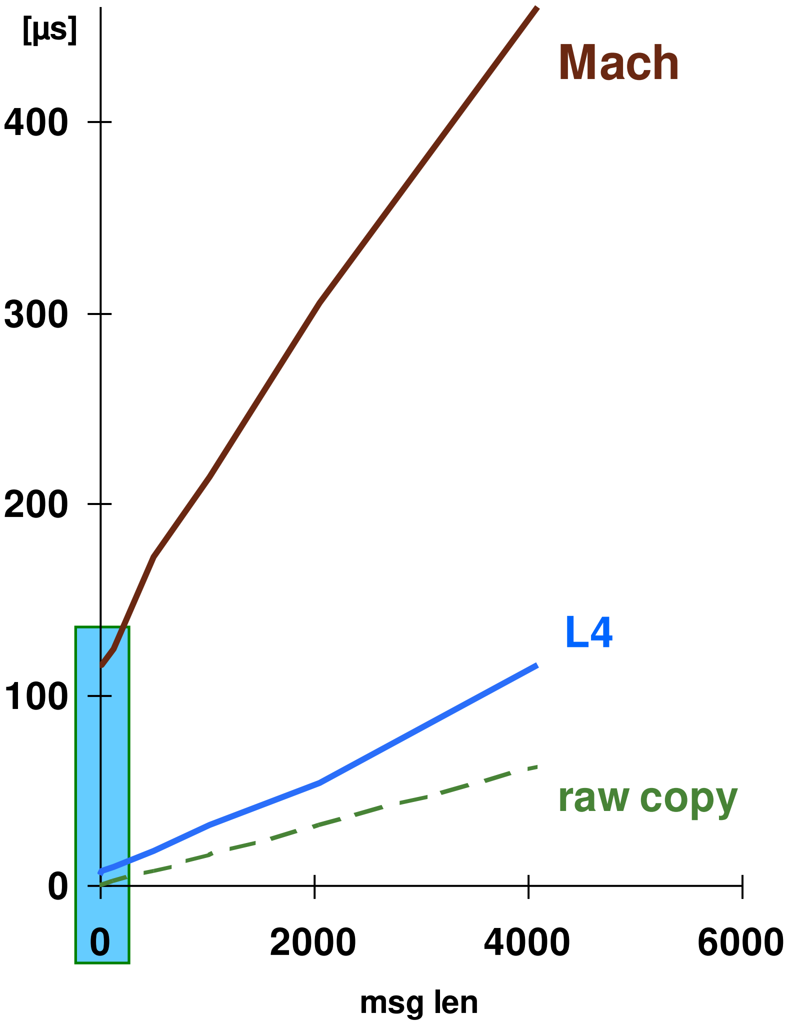
# Mach: The 100-$\mu$s Disaster

25 MHz 386 → 50 MHz 486 → 90 MHz Pentium → 133 MHz Alpha

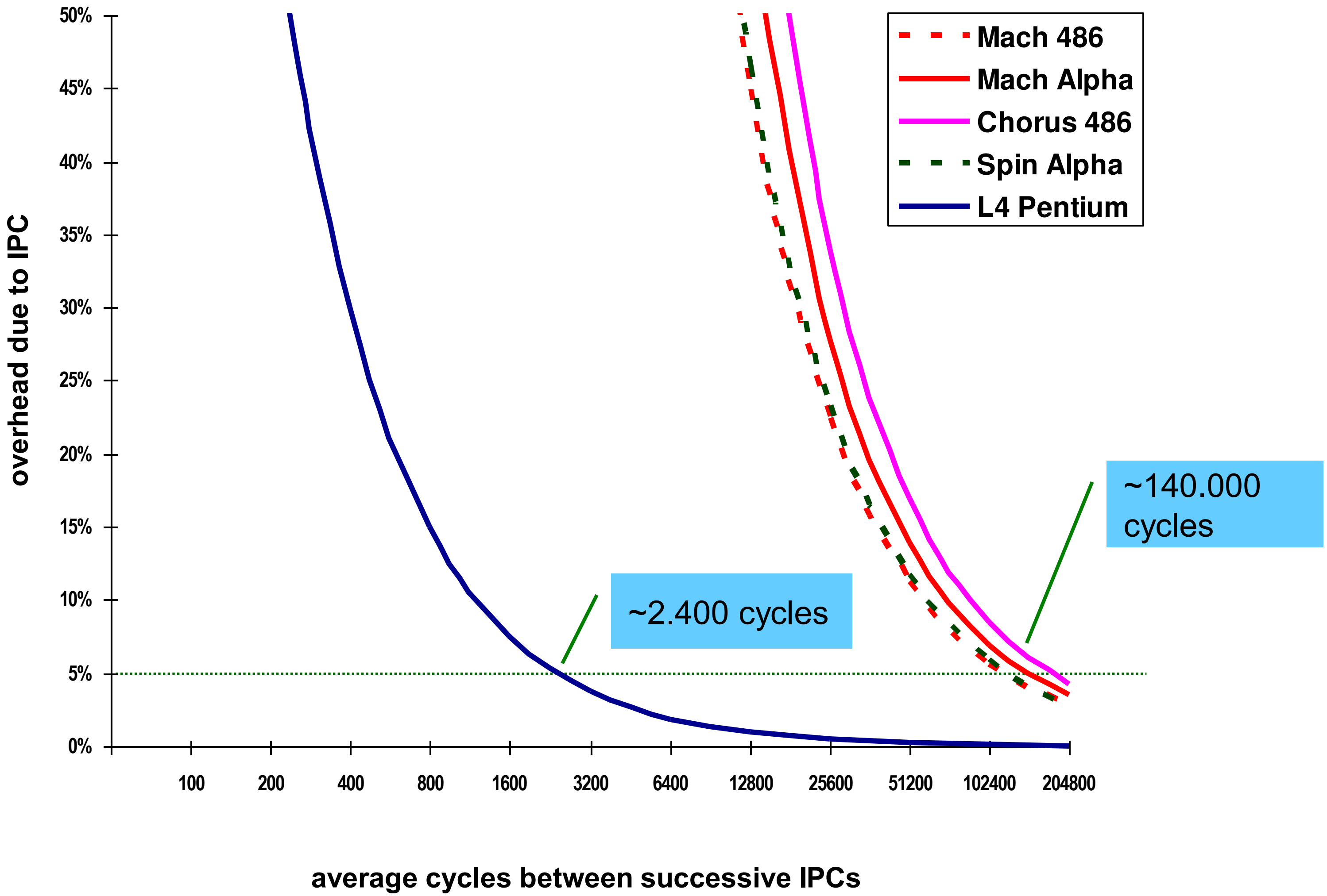

Operating Systems Group
Department of Computer Science

# IPC Costs (486, 50 MHz)



# Overhead due to IPC






Operating Systems Group
Department of Computer Science

# Thesis

A µ-kernel does the job if

- properly designed and
- carefully implemented.

- Minimality
- Architectural Integration
- Elegance

→

- Efficiency
- Flexibility

# Thesis

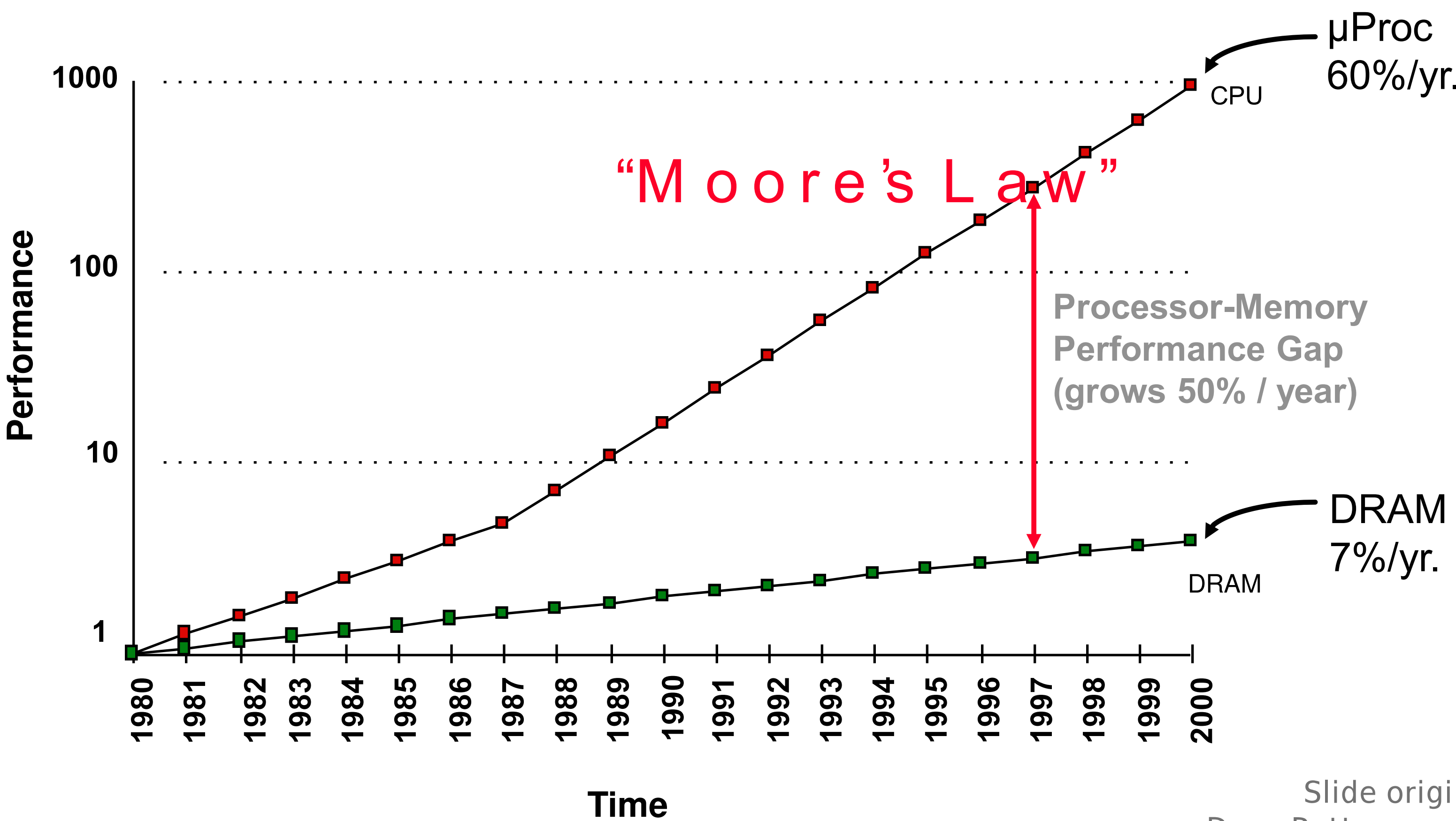
A µ-kernel does the job if

- properly designed and
- carefully implemented.

When analyzing IPC performance,

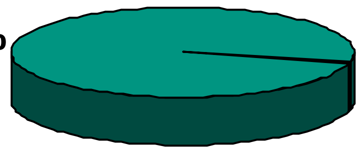
cycles are not the only thing to consider!

# Processor-DRAM Gap (Latency)




Slide originally from
Dave Patterson, Parcon 98

# Cache Working Sets

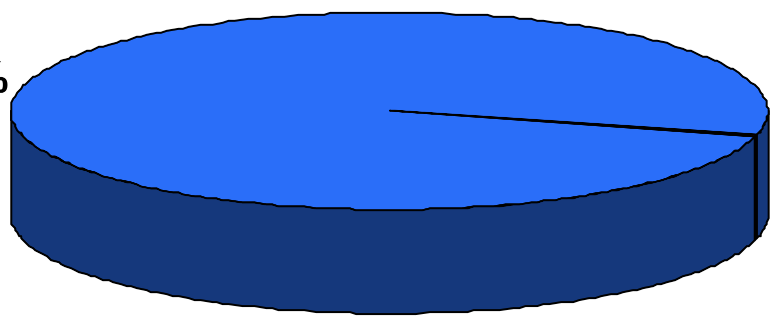



**L1 cache**

- **1024 cache lines (16K + 16K)**
- **12 lines used for IPC**

**L2 cache**

- **8192 cache lines (256K)**
- **12 lines used for IPC**



# Multi-Processor Architectures

- Synchronization
  - Bus locks
  - Inter-processor interrupts
- NUMA behavior
  - Simultaneous multithreading (HyperThreading)


Operating Systems Group
Department of Computer Science

Software engineering

Fault tolerance

Security

Performance

Memory

Quality of service

# Small Kernel ≠ Small Problem

Formal verification

Devices

Scheduling

Multiprocessor

Persistence

Portability



Operating Systems Group
Department of Computer Science

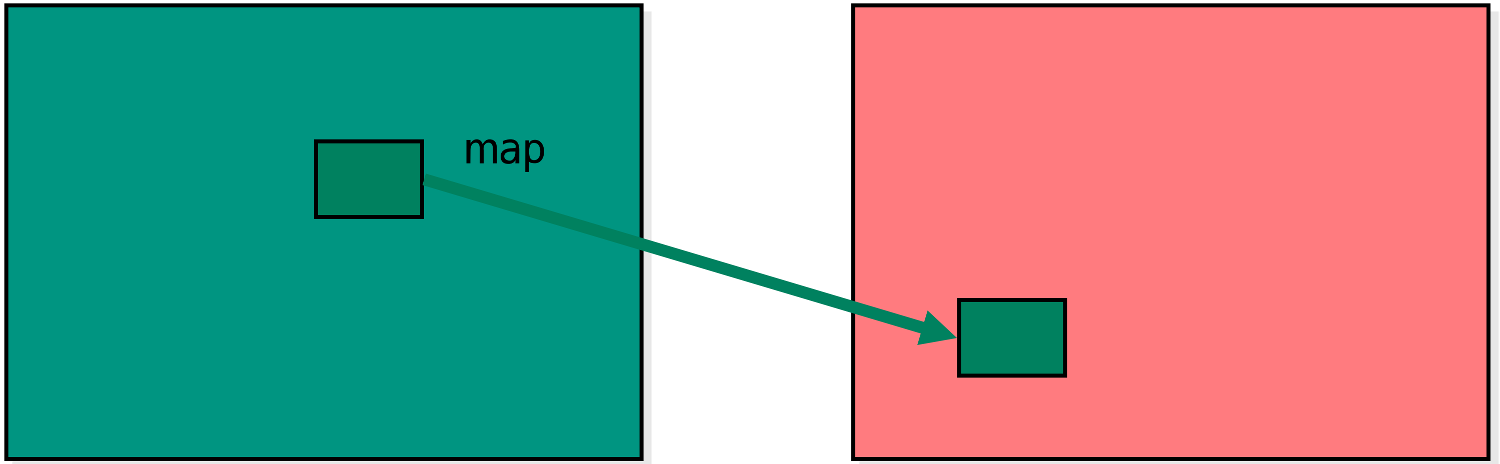
# µ-Kernel Design

- A µ-kernel does no real work
  - µ-kernel services are only required to overcome µ-kernel constraints (i.e., protection through address spaces)
- Therefore, µ-kernels have to be infinitely fast!

**Minimality is the key!**

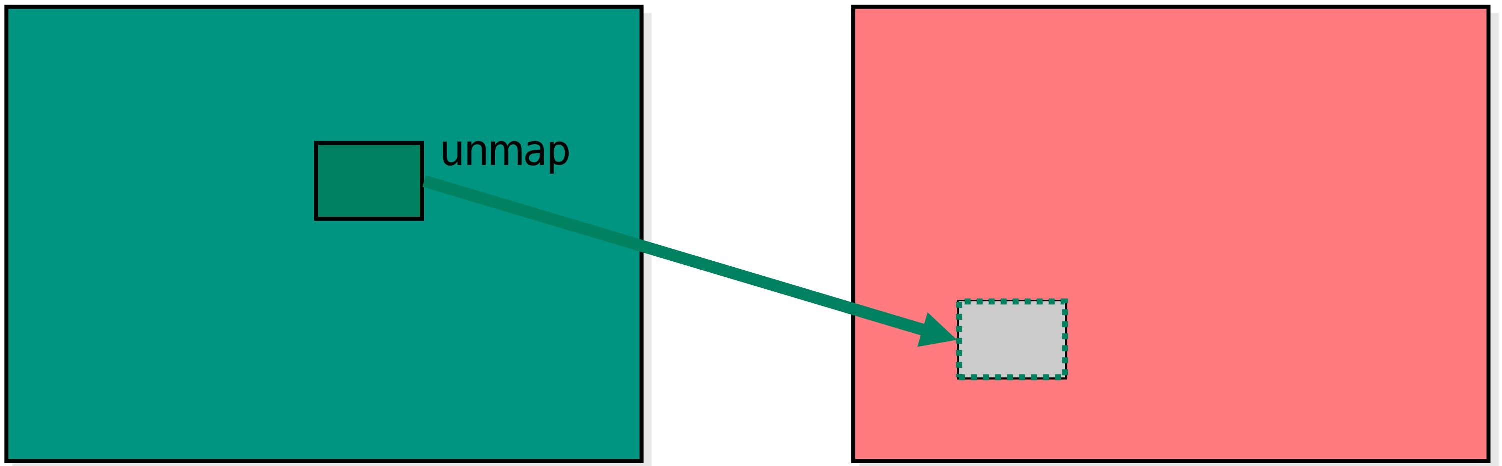
- **Address Spaces** — ***Mapping***
- **Threads** — ***IPC***

# Address Spaces – Mapping



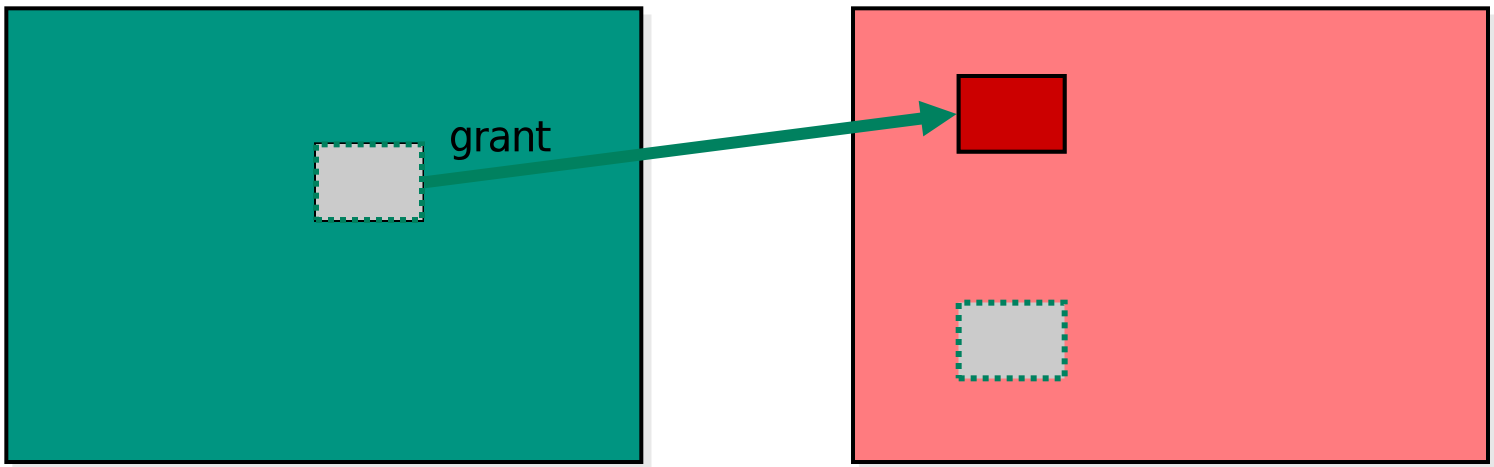

- Setup shared memory regions

# Address Spaces – Mapping



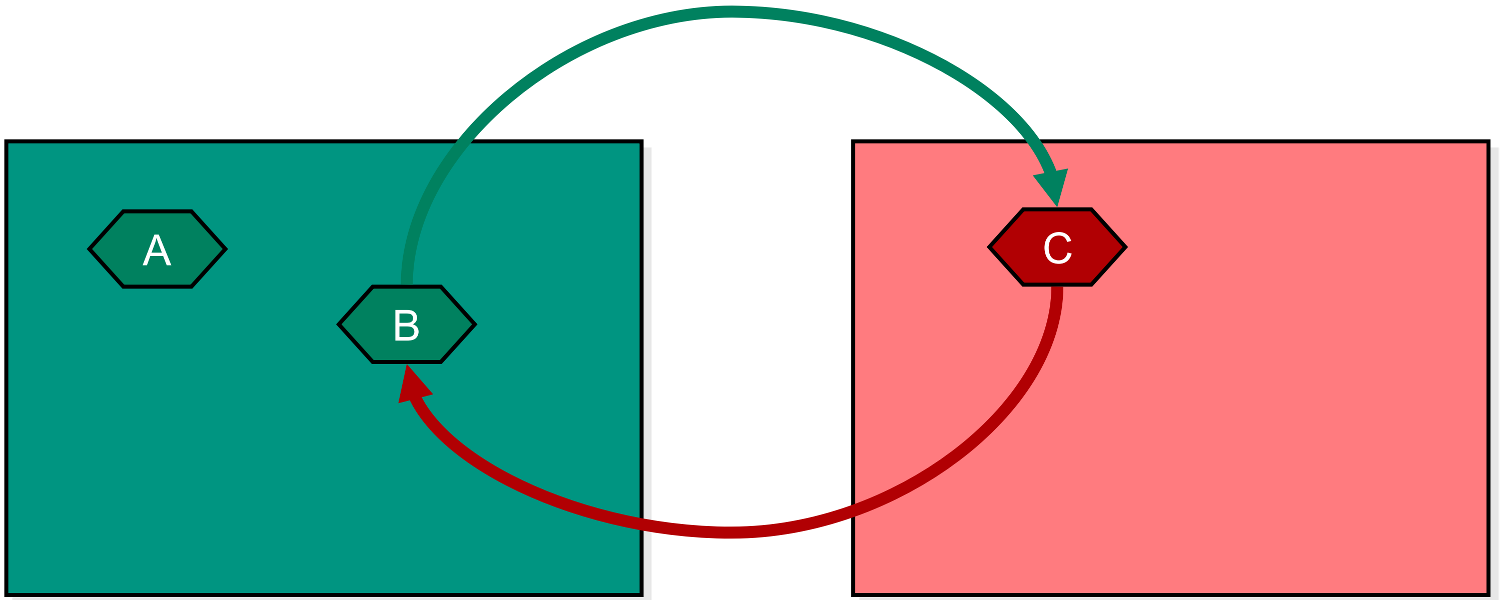

- Revoke shared memory regions

# Address Spaces – Mapping



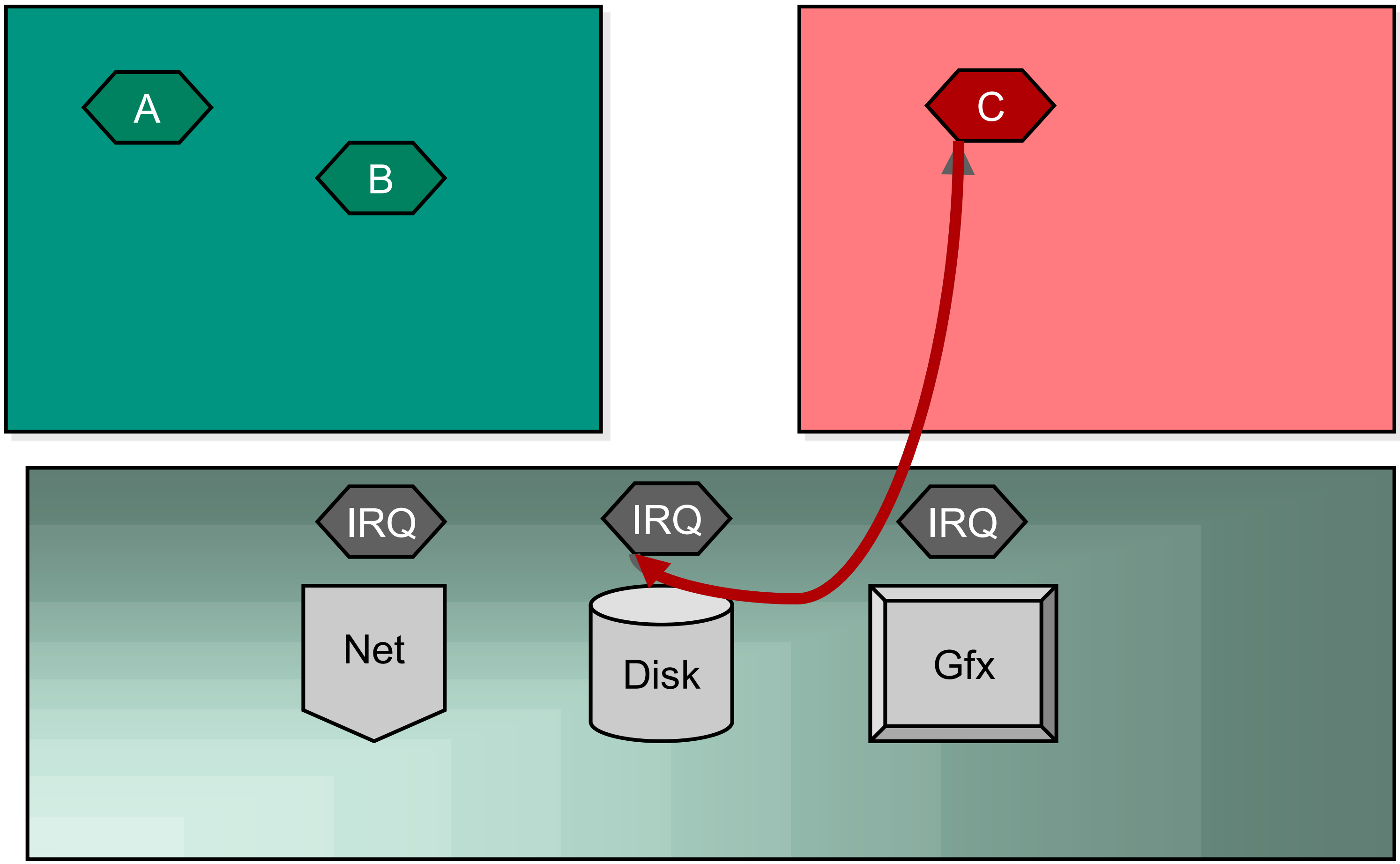

- Donate memory regions to others
- Frees up virtual memory in the granting space
  - Useful for file servers

# Threads – IPC




- Enable controlled communication across address space boundaries

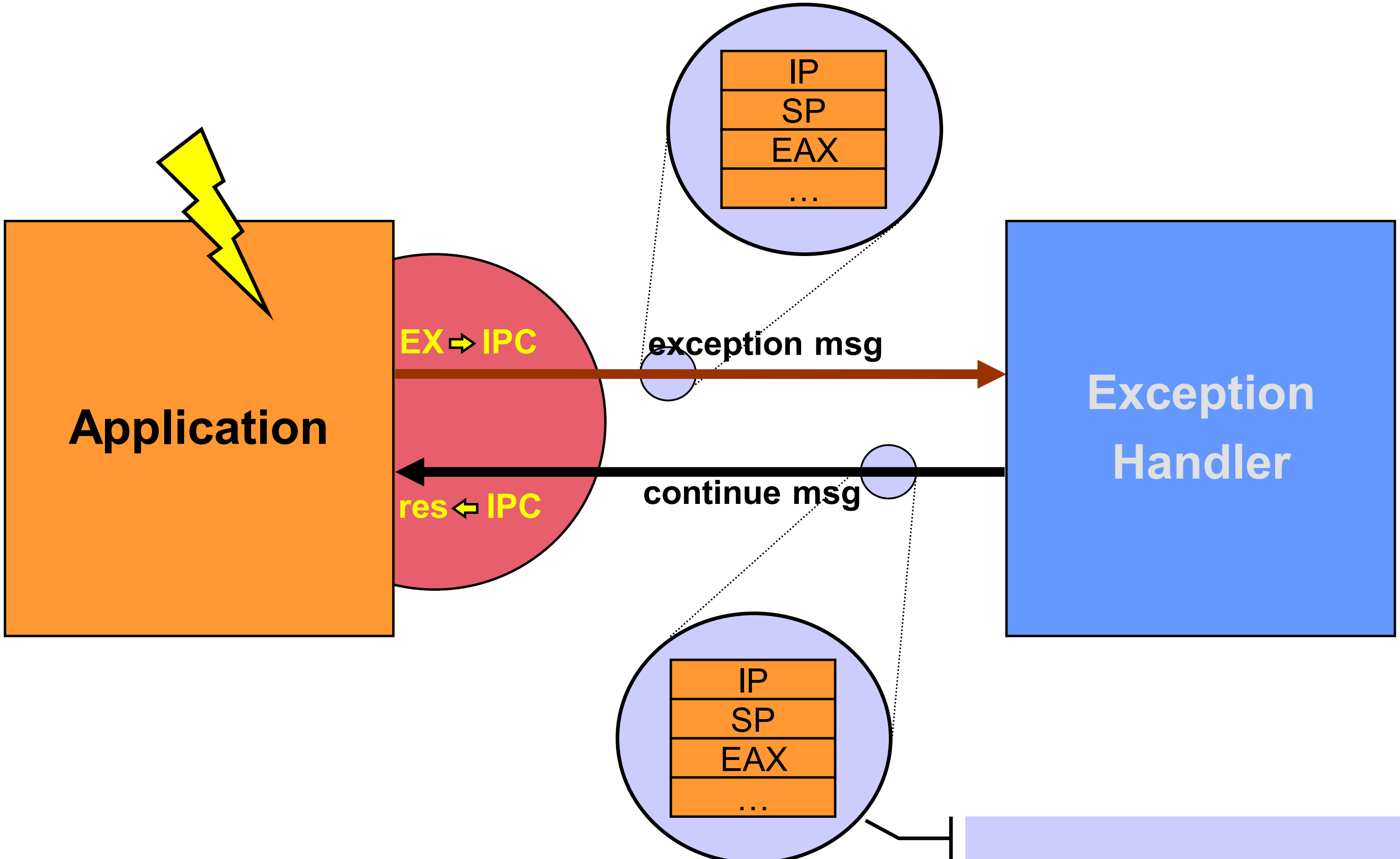
# User-Level Device Drivers





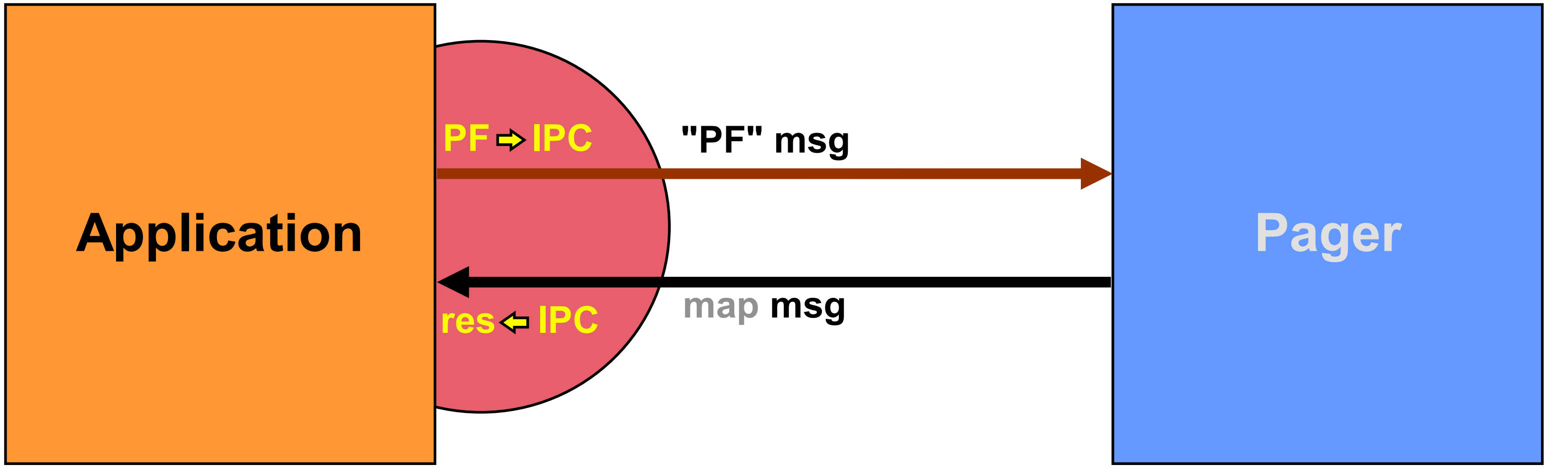

Operating Systems Group
Department of Computer Science

# Exception Handling




Kernel modifies register contents according to reply message

# Page Fault Handling



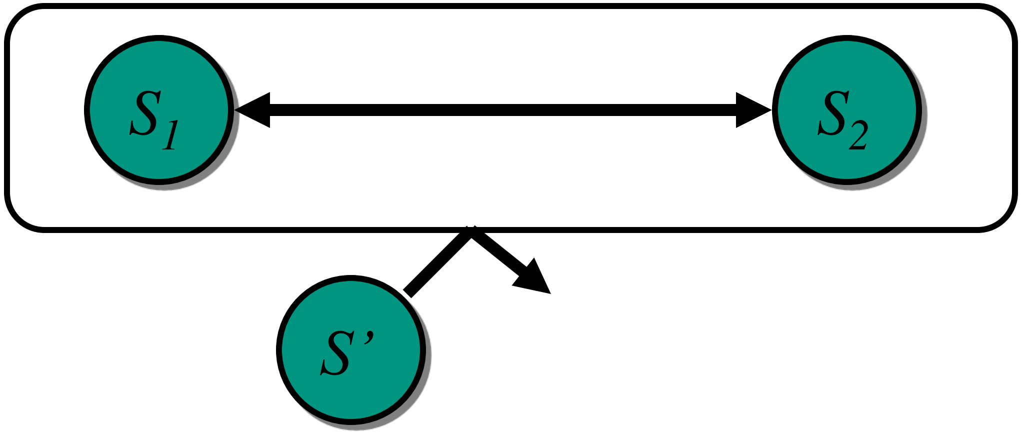
# Tolerated Concepts

A concept is tolerated in the $\mu$-kernel if…

competing user-level implementations would violate system requirements.

# Functional Requirements

- Principle of independence
  - Subsystem $S$ provides guarantees independent of $S'$
- Principle of integrity
  - Other subsystems can rely on independence guarantees
- Example: performance isolation, memory isolation

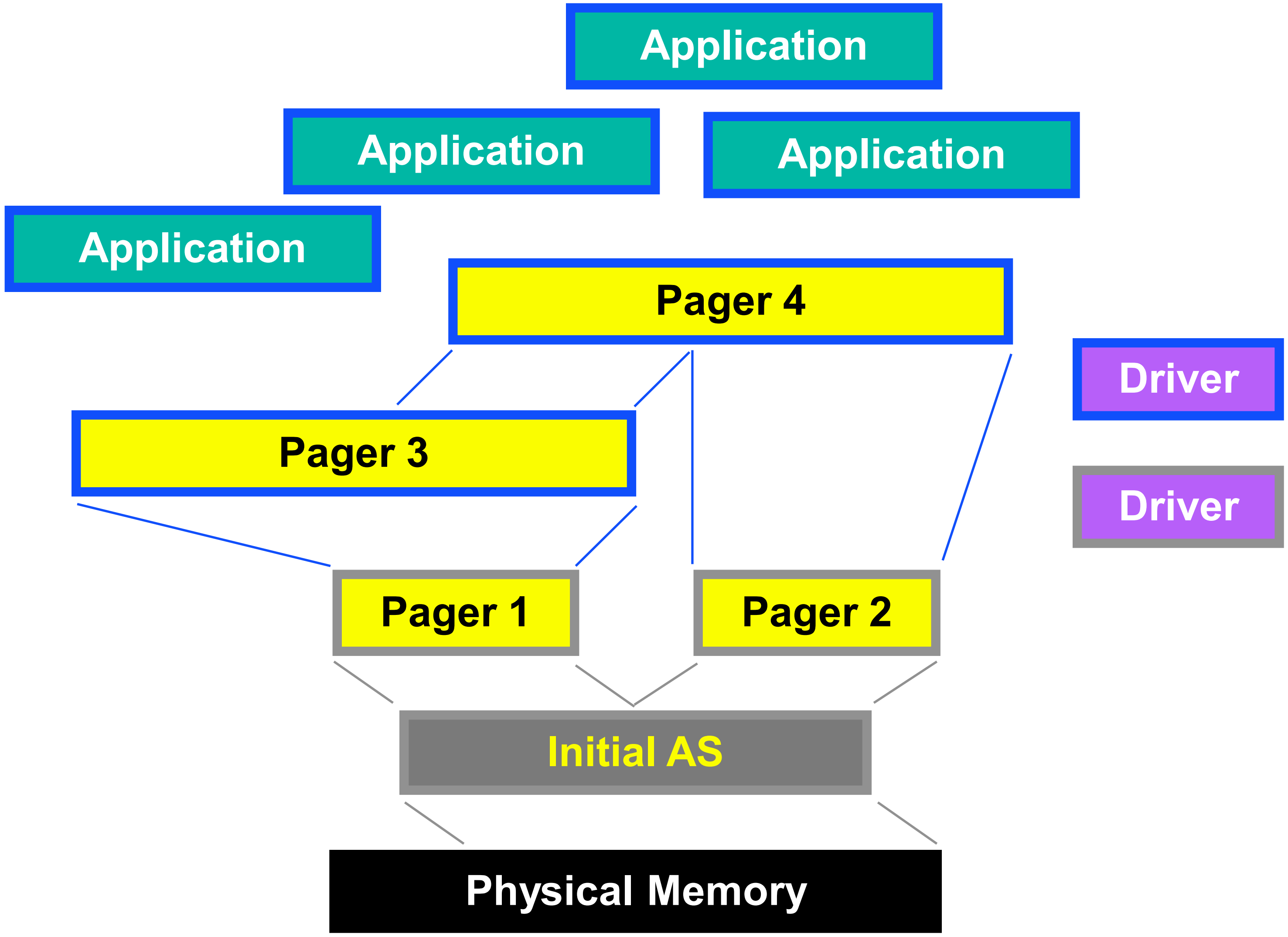


# Requirement: Address Spaces

- µ-kernel must hide hardware address spaces
  - Otherwise, integrity principle is violated
- Must permit arbitrary protection schemes
  - Including non-protection…
- Solution: recursive construction of address spaces outside the kernel


Operating Systems Group
Department of Computer Science

# Recursive Address Spaces



# Requirement: IPC

- IPC = inter-process communication
- Inherently required in µ-kernel with threads

- Transfer messages between endpoints
  - Threads (e.g., Pistachio)
  - IPC gates (e.g., Fiasco.OC)

- Contractual
  - Sender determines what to send
  - Receiver agrees to receive the information

# Requirement: Threads

- A thread $\tau$ is an activity inside an address space with
  - registers
  - instruction pointer
  - stack pointer
  - state information
- $\sigma(\tau)$ := address space of thread $\tau$

# Why Tolerate Threads in μ-Kernels?

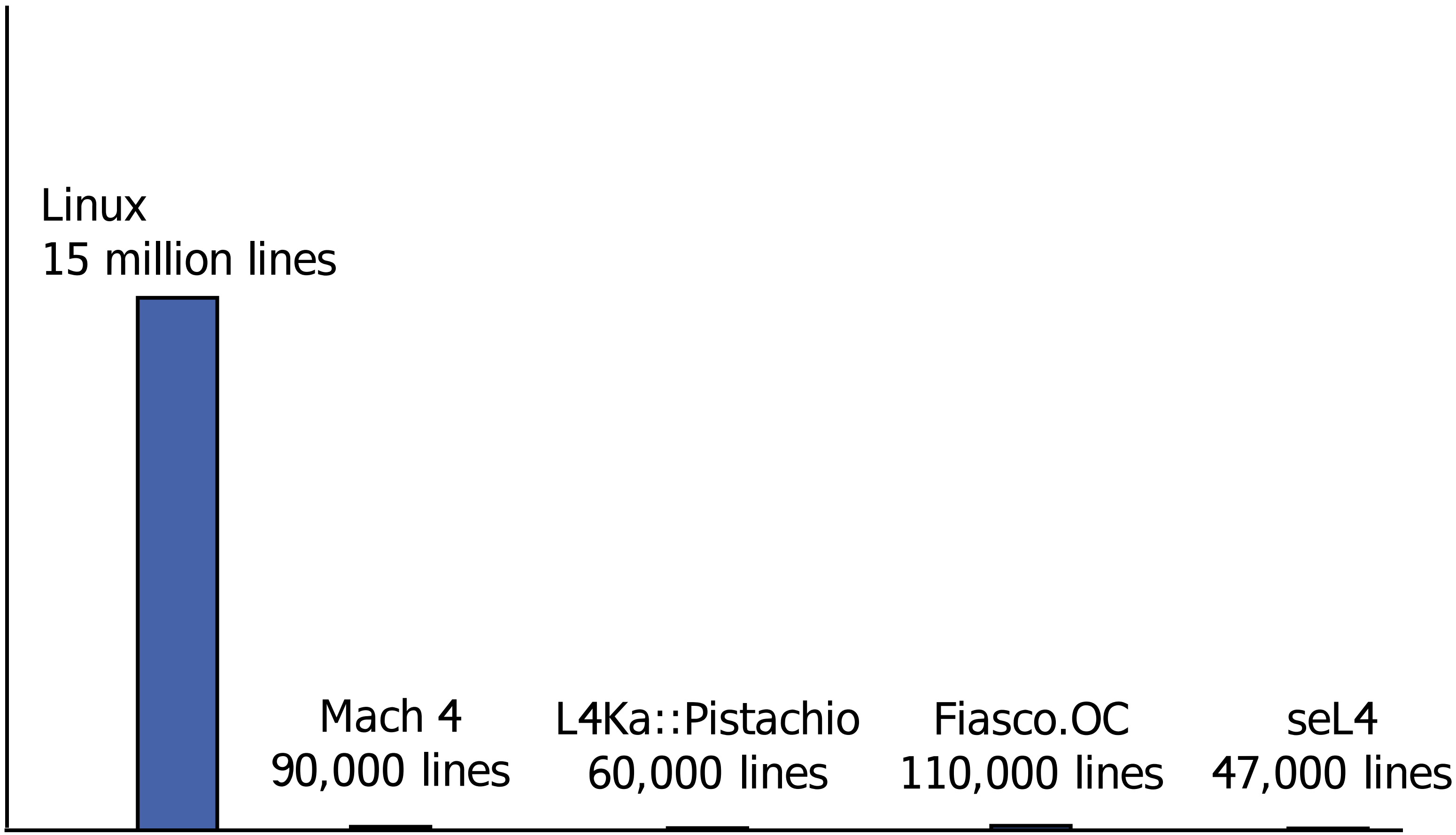
- How to guarantee "fair" CPU access for all activities?

- Need a trusted intermediary (the kernel) to multiplex
  - This means policy in the kernel, which is BAD™

- Unfortunately, no better solution exists


Operating Systems Group
Department of Computer Science

# Size Comparison



# Course Contents

## You do learn

- How to design a µK
- Why L4 is sooooo fast
- Reasons why others failed
- Things we screwed up
- Nitty-gritty details about x86
- Some OS bashing…
- More cool stuff…

## You don't learn

- How to construct a system on a µK
- Linux dos and don'ts
- Why operating system X is better than Y

# Course Overview

- Overview, Motivation, Problems
- Threads, System-calls, and Thread Switching
- TCBs and Address Space Layout
- IPC Functionality and Implementation
- Dispatching
- Virtual Memory and Mapping Database
- Interrupts, Exceptions and CPU Virtualization
- Security

Many algorithms, often influencing the system design.

# Next Lecture

- Next lecture:

Threads, System Calls, Thread Switching