µ-Kernel Construction (1)

Overview, Motivation, Problems

Staff

- Lecturer: Raphael Neider
- PhD student

- Meeting Times
  - Tue, 14:00-15:30h
  - Bldg. 50.34, Room 161

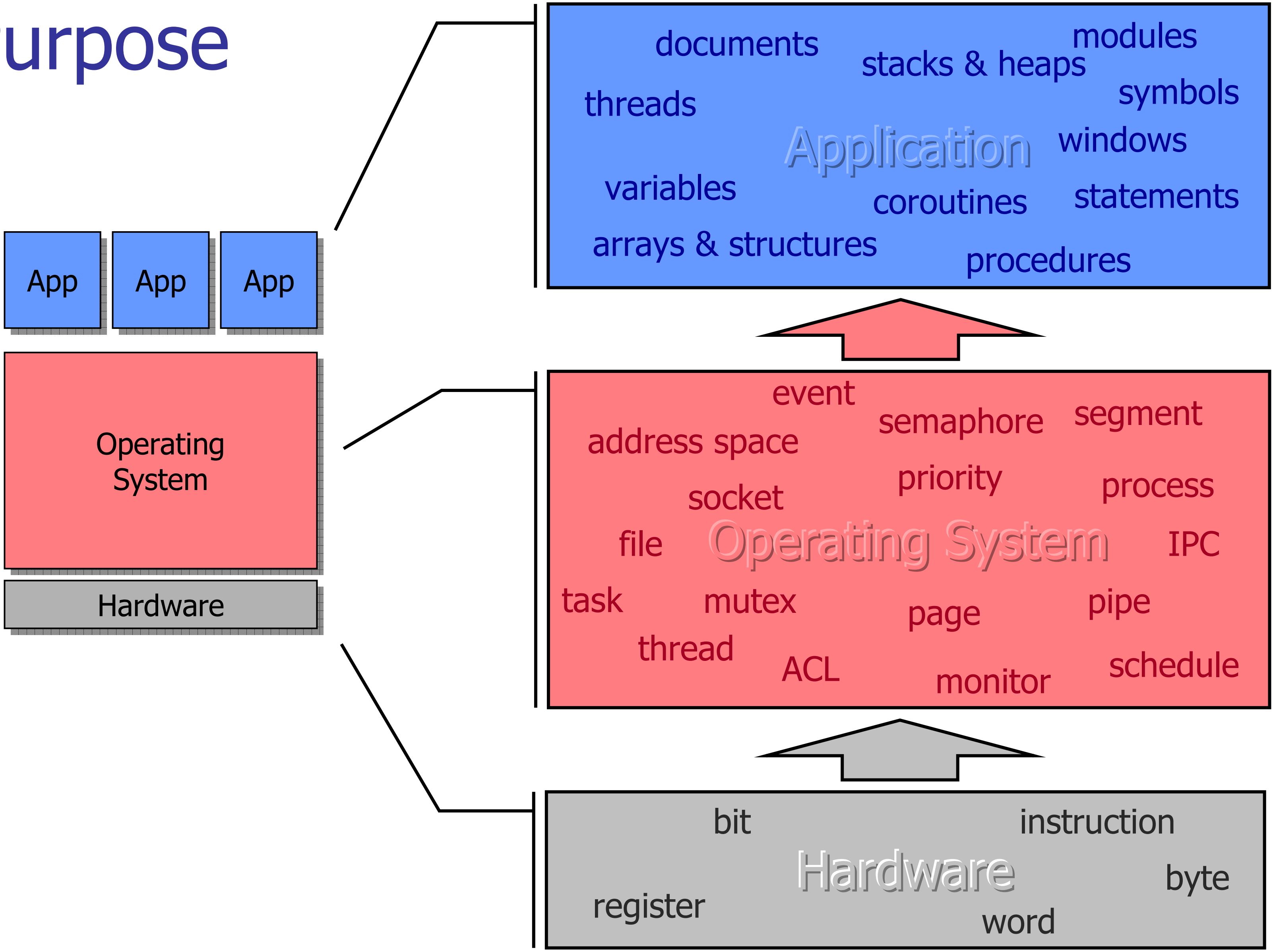
Background

- L4Ka project (http://l4ka.org)
- State-of-the-art in microkernel construction
  - L4Ka::Pistachio
    - Microkernel
  - IDL4
    - Interface compiler
  - Virtual machines
    - Target application

Purpose of Operating Systems

- Abstract from the hardware
  - Interrupts, exceptions
- Provide common services
  - Address spaces and protection
  - Threads and concurrency control
  - Persistency of data
- Bridge semantic gap
  - Application demands vs. hardware provides

Purpose



App
App
App
Operating System
Hardware
documents
stacks & heaps
modules
threads
symbols
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
Hardware
byte
register
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

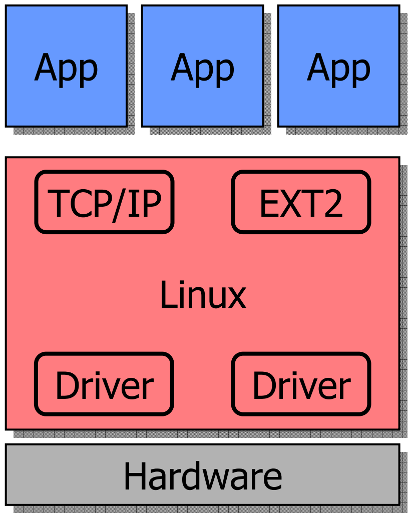
- ✓ Ultimate flexibility
- ✓ Ultimate minimalism
- ✓ Ultimate performance
- ✓ Normal programming paradigm
- ✗ Proprietary and incompatible solutions

monolithic

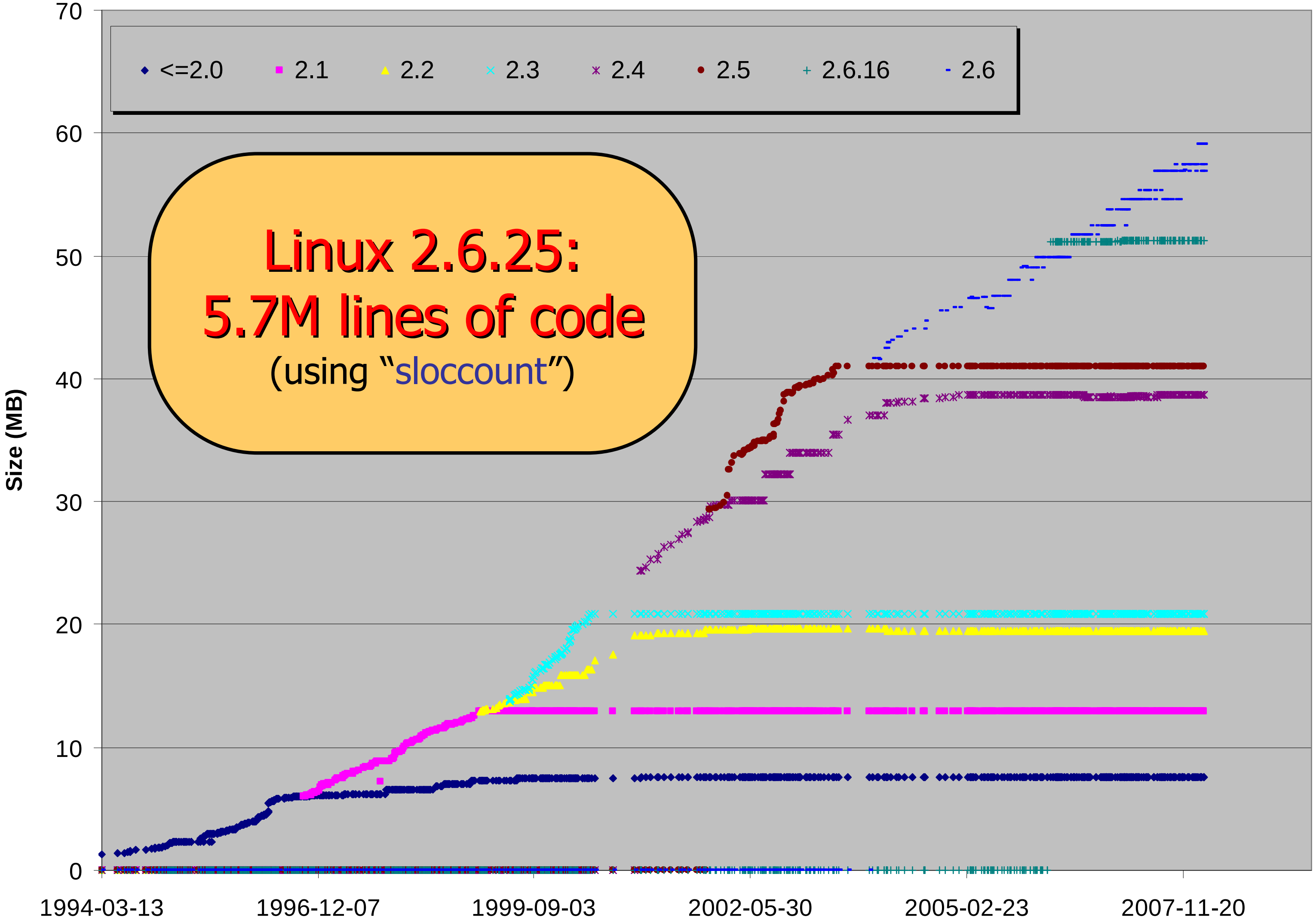
- ✓ Standard interface
- ✓ Runs programs from different vendors
- ✗ Compromised interface
- ✗ Poor performance
- ✗ Inflexible

Monolithic Kernels – Advantages

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



Linux Kernel Evolution (.tar.gz)



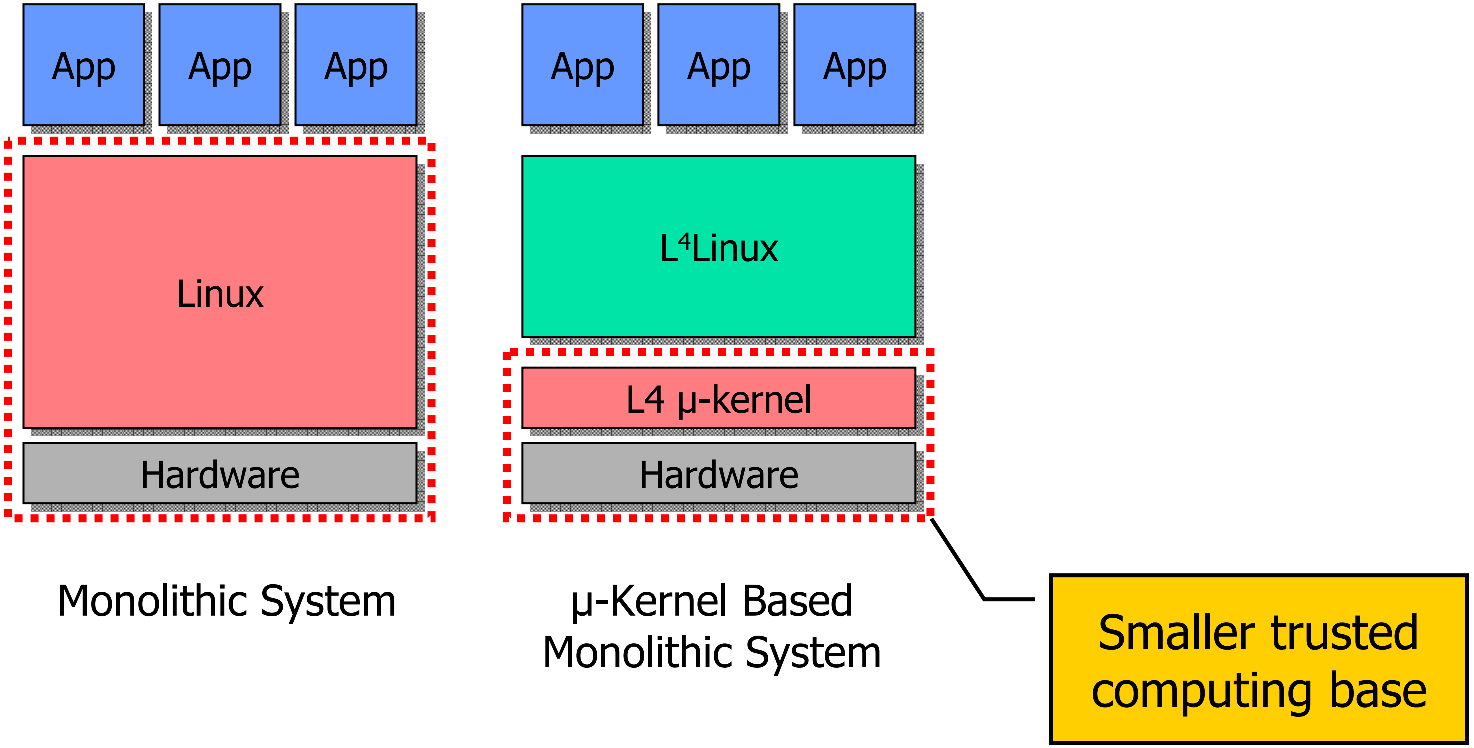
Approaches to Tackling Complexity

- Monolithic approaches
  - Layered Kernels
  - Modular Kernels
  - Object Oriented Kernels
- Alternatives
  - Extensible Kernels
  - **Microkernels**

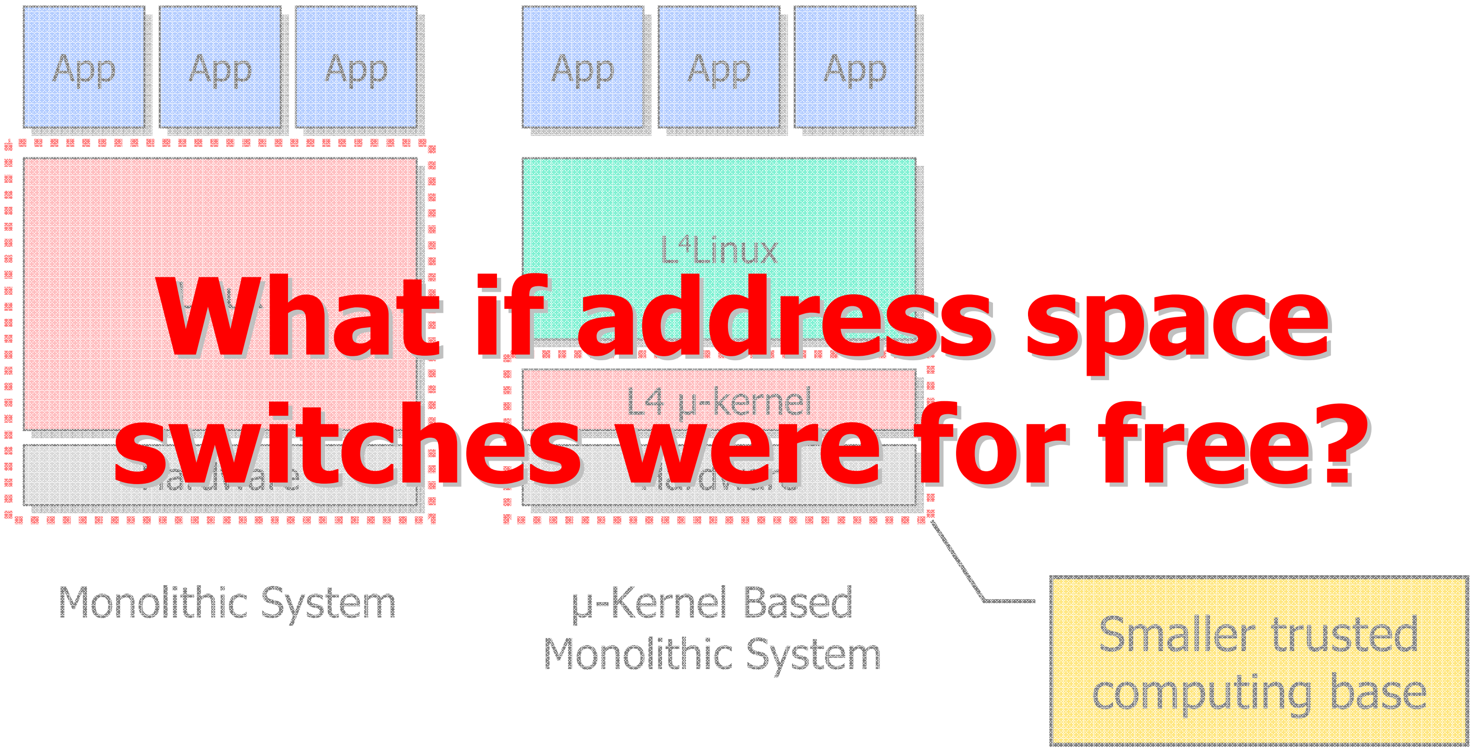
History

- Monolithic kernels
- 1st-generation µ-kernels
  - Mach *CMU, OSF* **External Pager**
  - Chorus *Inria, Chorus*
  - Amoeba *Vrije Universiteit*
  - (L3) *GMD* **User-Level Driver**
- 2nd-generation µ-kernels
  - (Spin) *U Washington*
  - Exokernel *MIT*
  - L4 *GMD / IBM / UKa* **Recursive Address Spaces**

µ-Kernel Based Systems

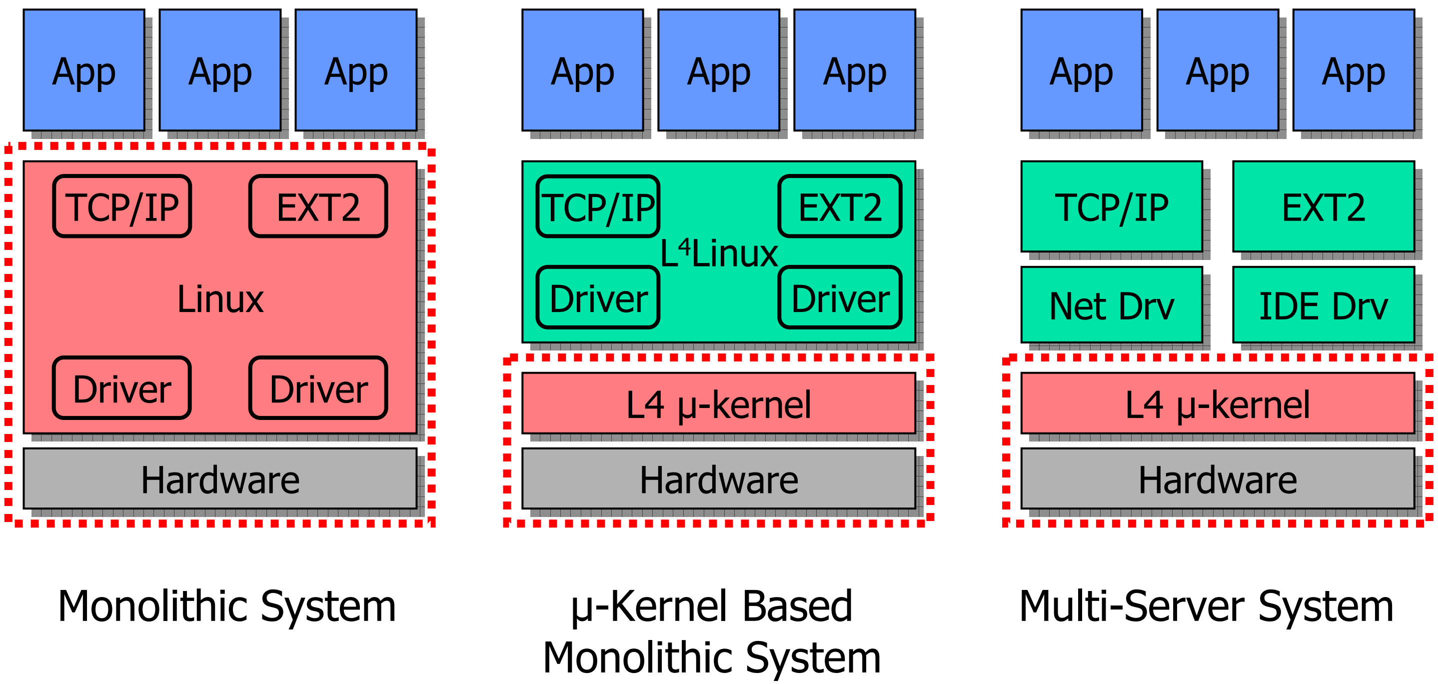


µ-Kernel Based Systems

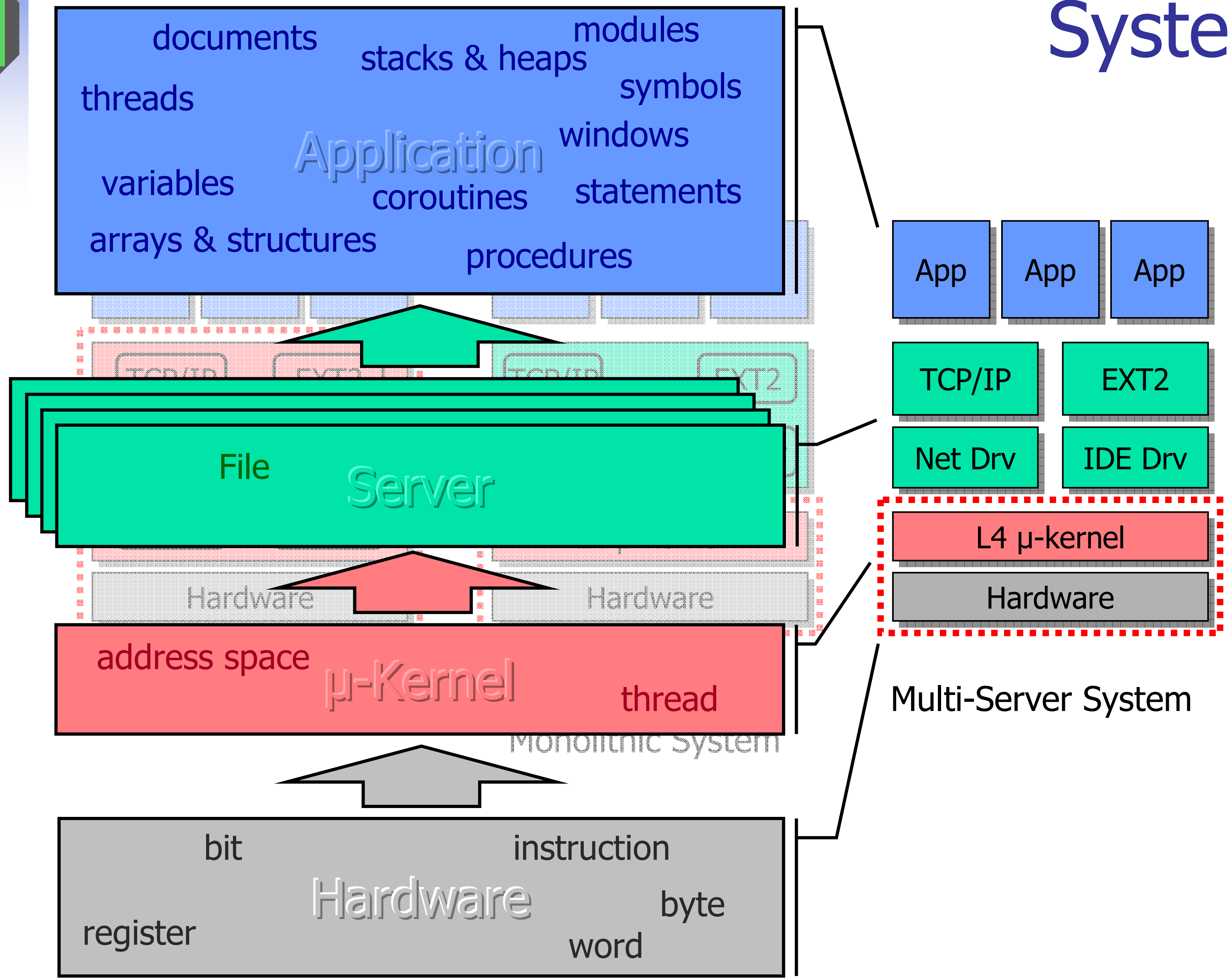


What if address space switches were for free?

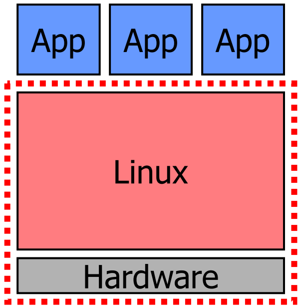
μ-Kernel Based Systems



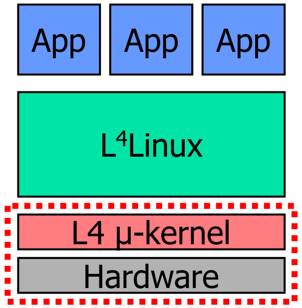
µ-Kernel Based Systems



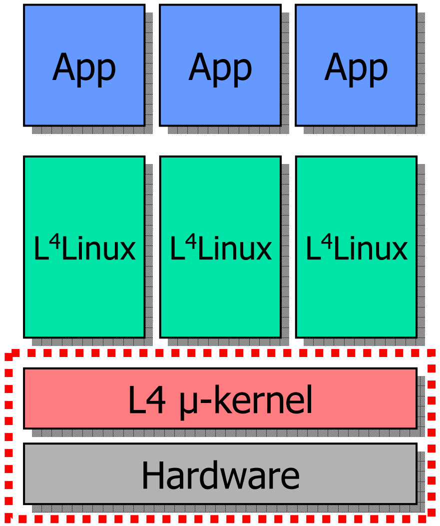
Microkernel Based Systems



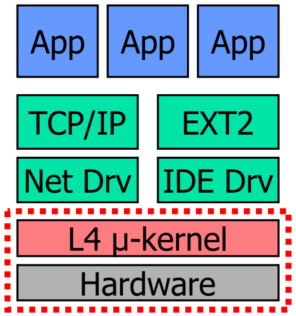
Monolithic System



µ-Kernel Based
Monolithic System

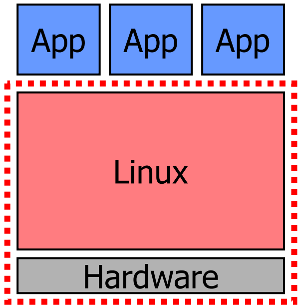


µ-Kernel Based
Server Consolidation

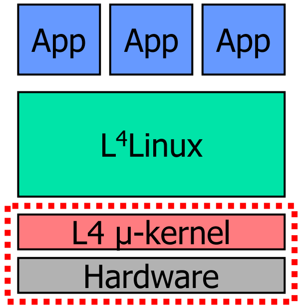


Multi-Server System

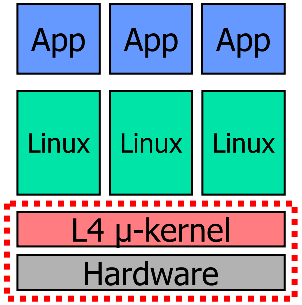
Microkernel Based Systems



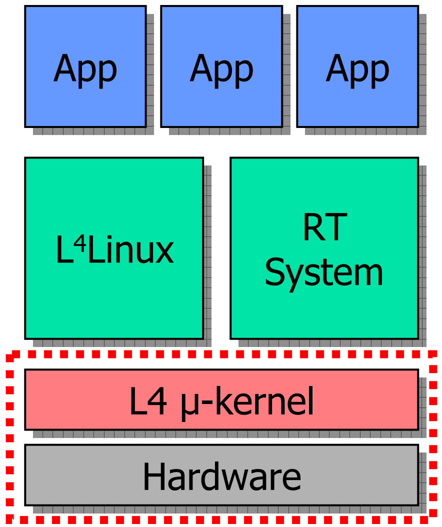
Monolithic System



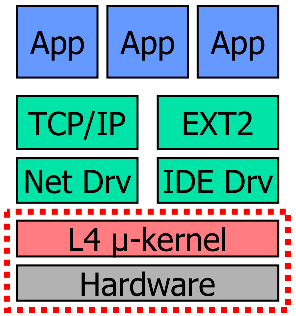
µ-Kernel Based
Monolithic System



µ-Kernel Based
Server Consolidation

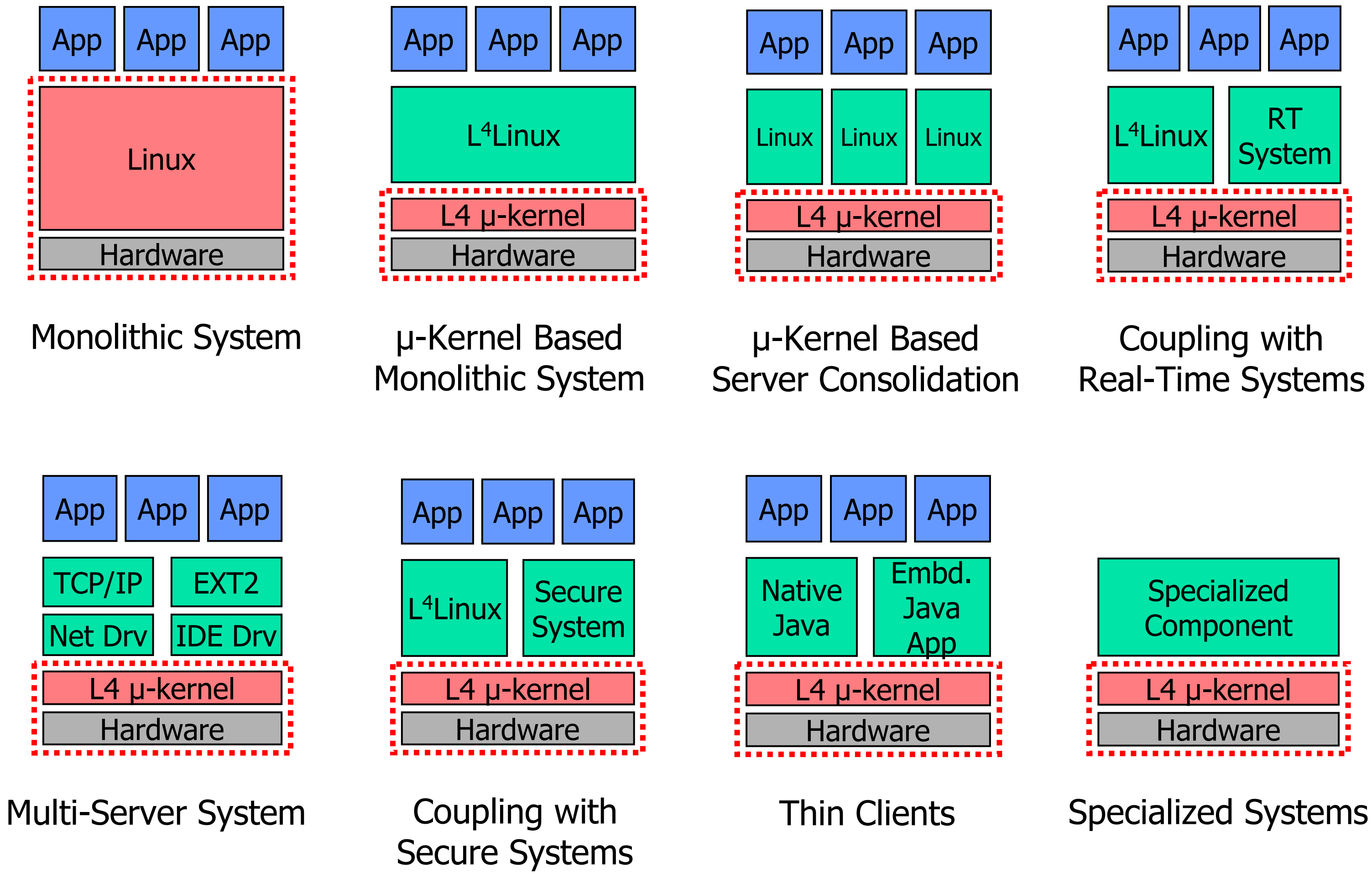


Coupling with
Real-Time Systems

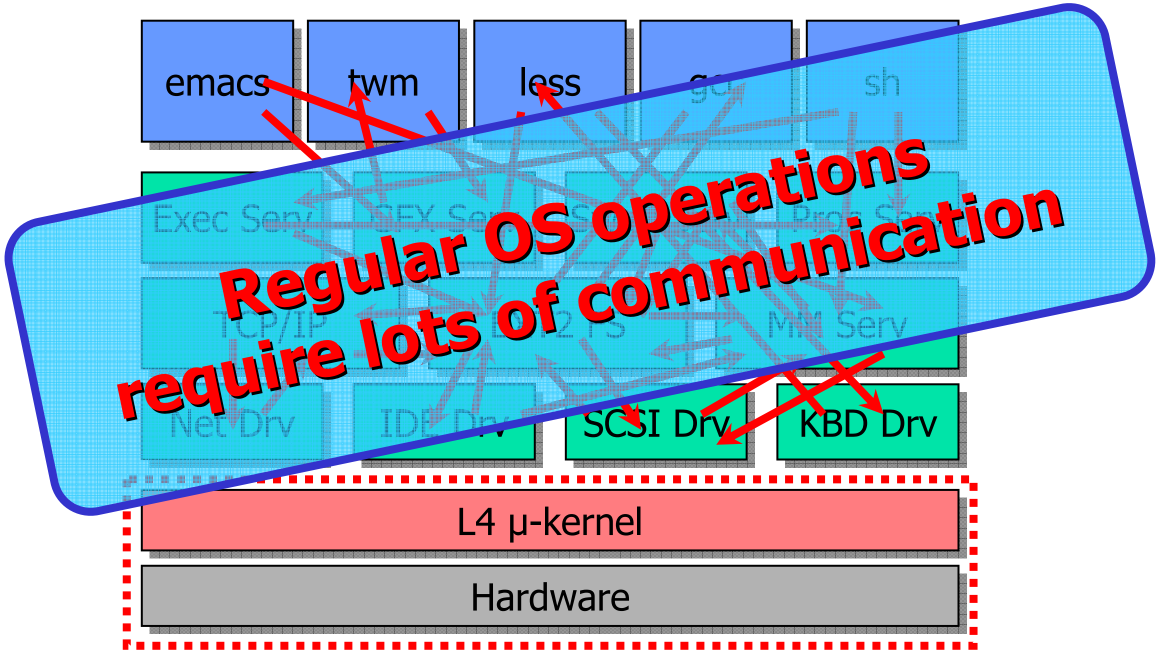


Multi-Server System

Microkernel Based Systems



Microkernel Based Systems: The Challenge



The Great Promise

The Big Disaster

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

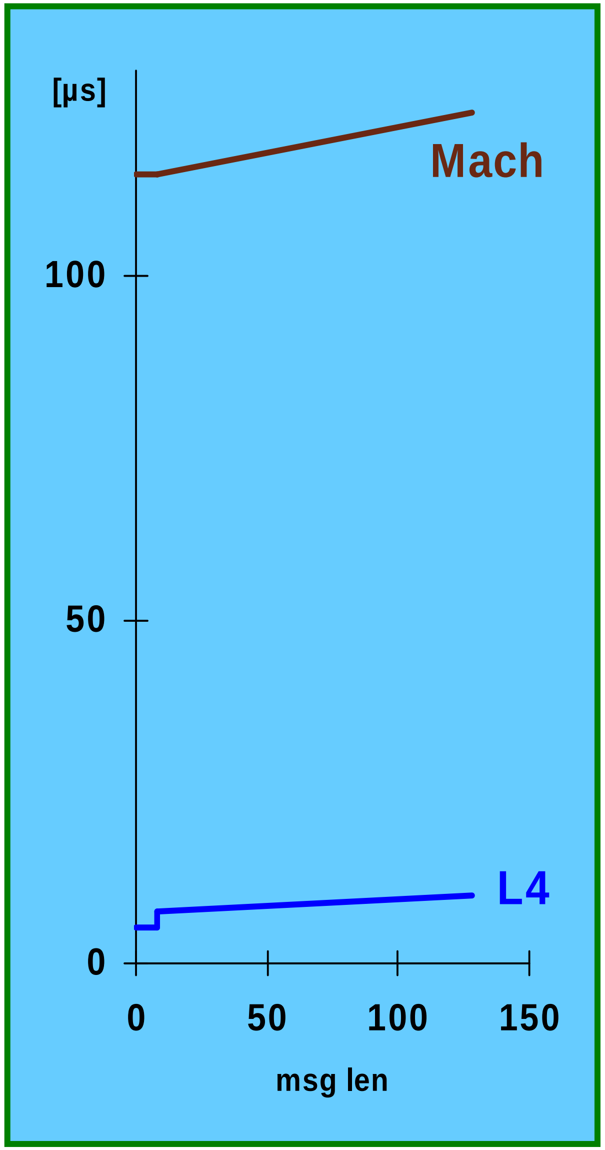
- ***SLOW***
- ***INFLEXIBLE***
- ***LARGE***

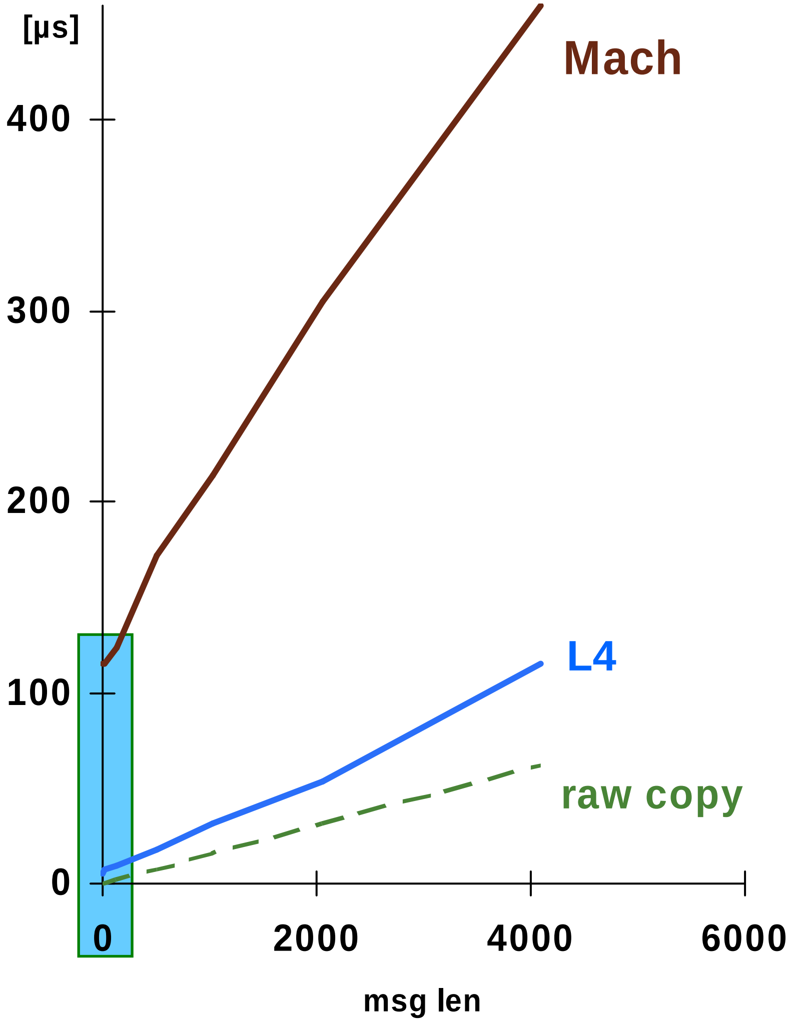
IBM WorkPlace OS:
~2,000,000,000 US$

The 100-µs Disaster

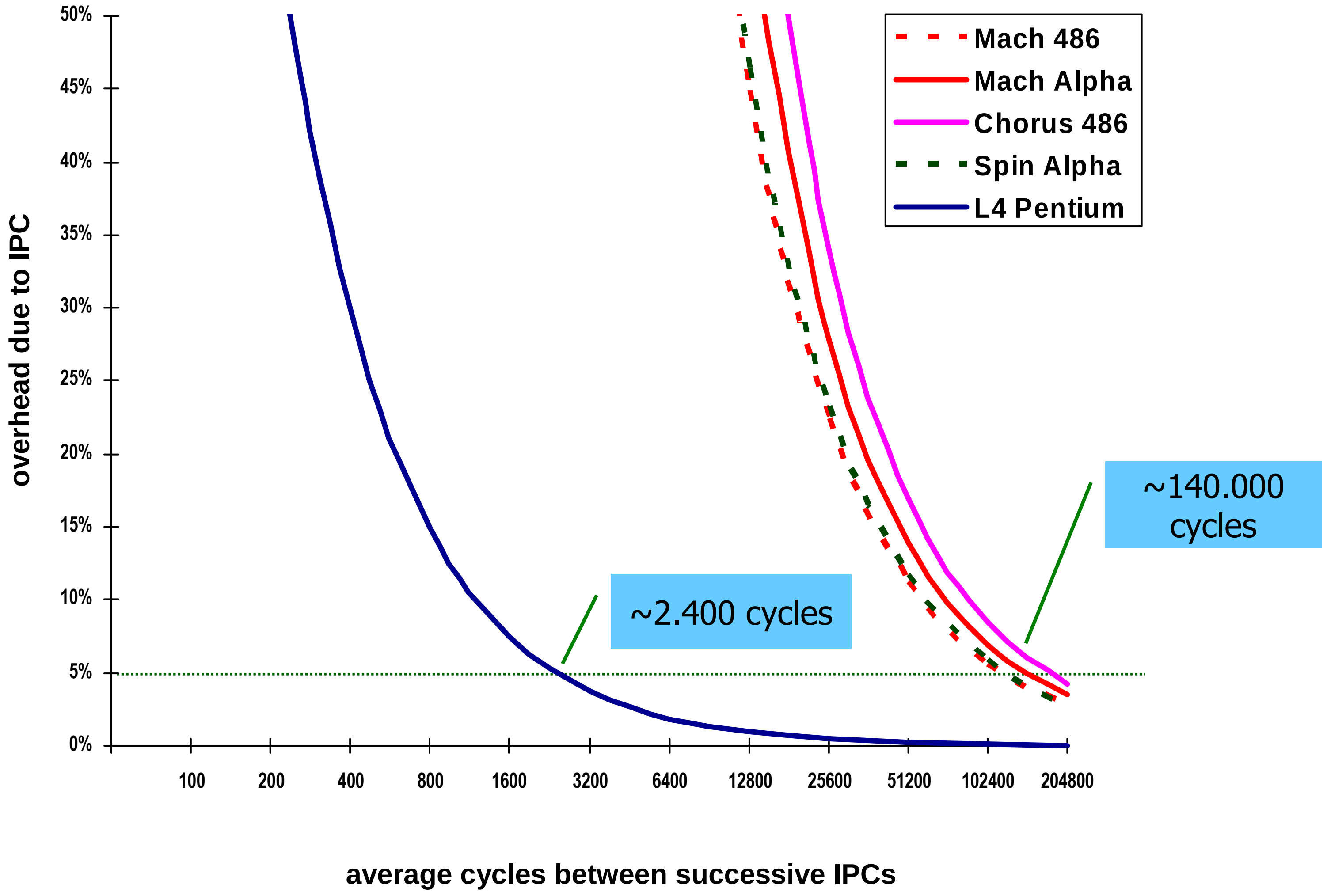
25 MHz 386 → 50 MHz 486 → 90 MHz Pentium → 133 MHz Alpha

IPC Costs (486, 50 MHz)

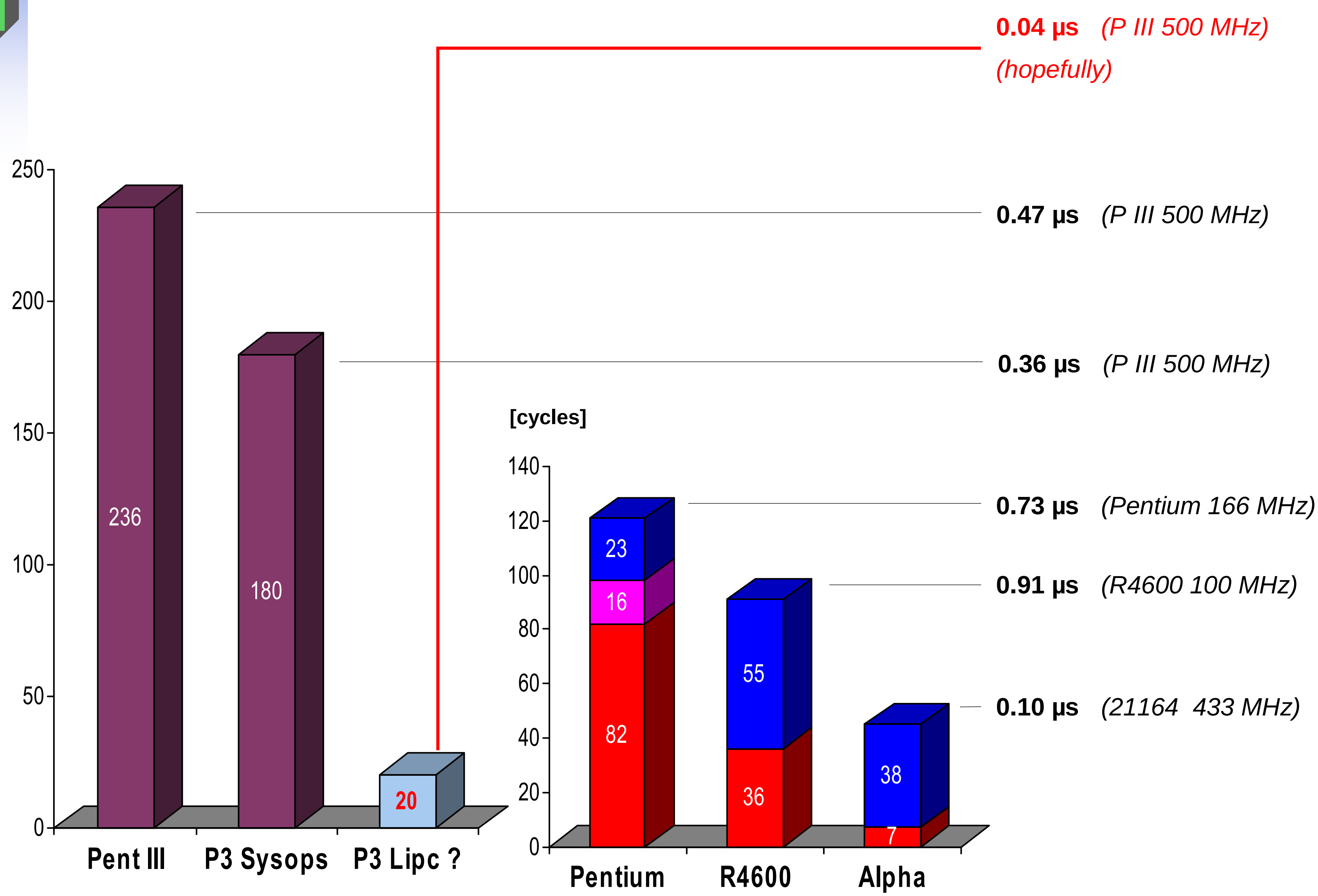




Overhead due to IPC

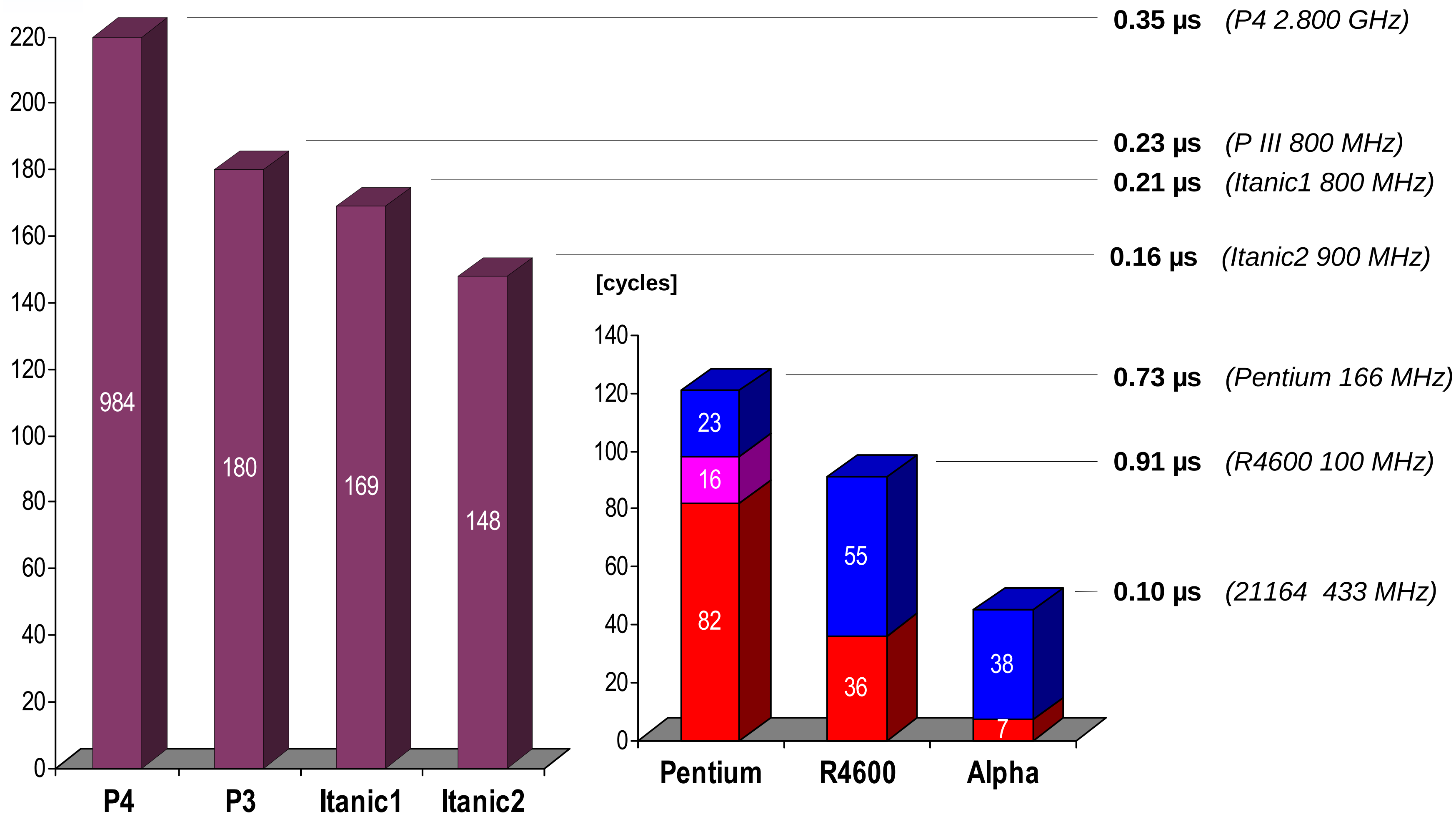


L4 IPC: '95—'00

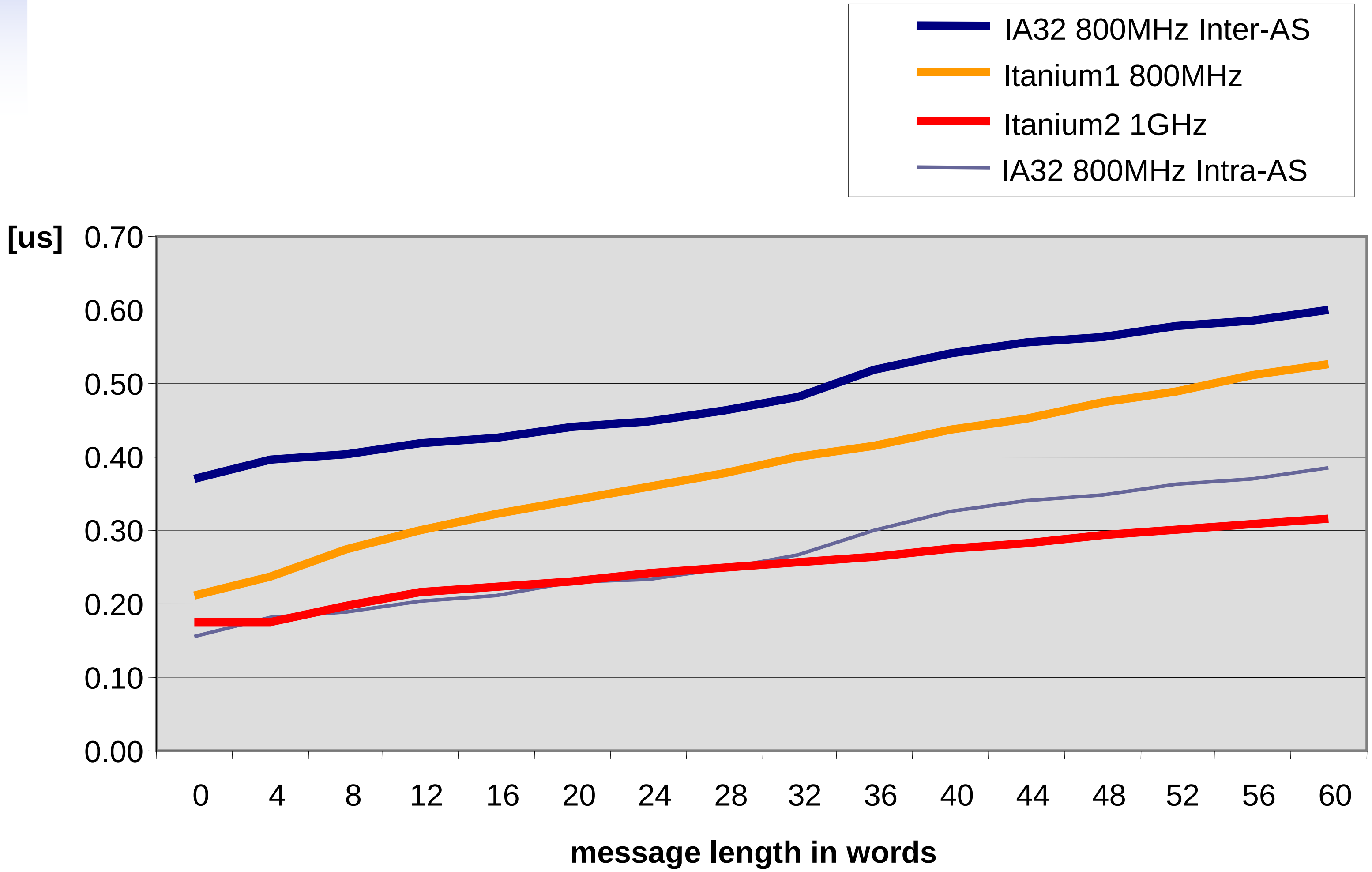


0.04 µs (P III 500 MHz)
(hopefully)
0.47 µs (P III 500 MHz)
0.36 µs (P III 500 MHz)
[cycles]
0.73 µs (Pentium 166 MHz)
0.91 µs (R4600 100 MHz)
0.10 µs (21164 433 MHz)
250
200
150
100
50
0
236
180
20
Pent III
P3 Sysops
P3 Lipc ?
140
120
100
80
60
40
20
0
23
16
82
55
36
38
7
Pentium
R4600
Alpha

L4 IPC Performance



L4Ka::Pistachio (today)



Thesis

A µ-kernel does the job if

- properly designed and
- carefully implemented.

- **Minimality**
- **Architectural Integration**
- **Elegance**

- **Efficiency**
- **Flexibility**

Thesis

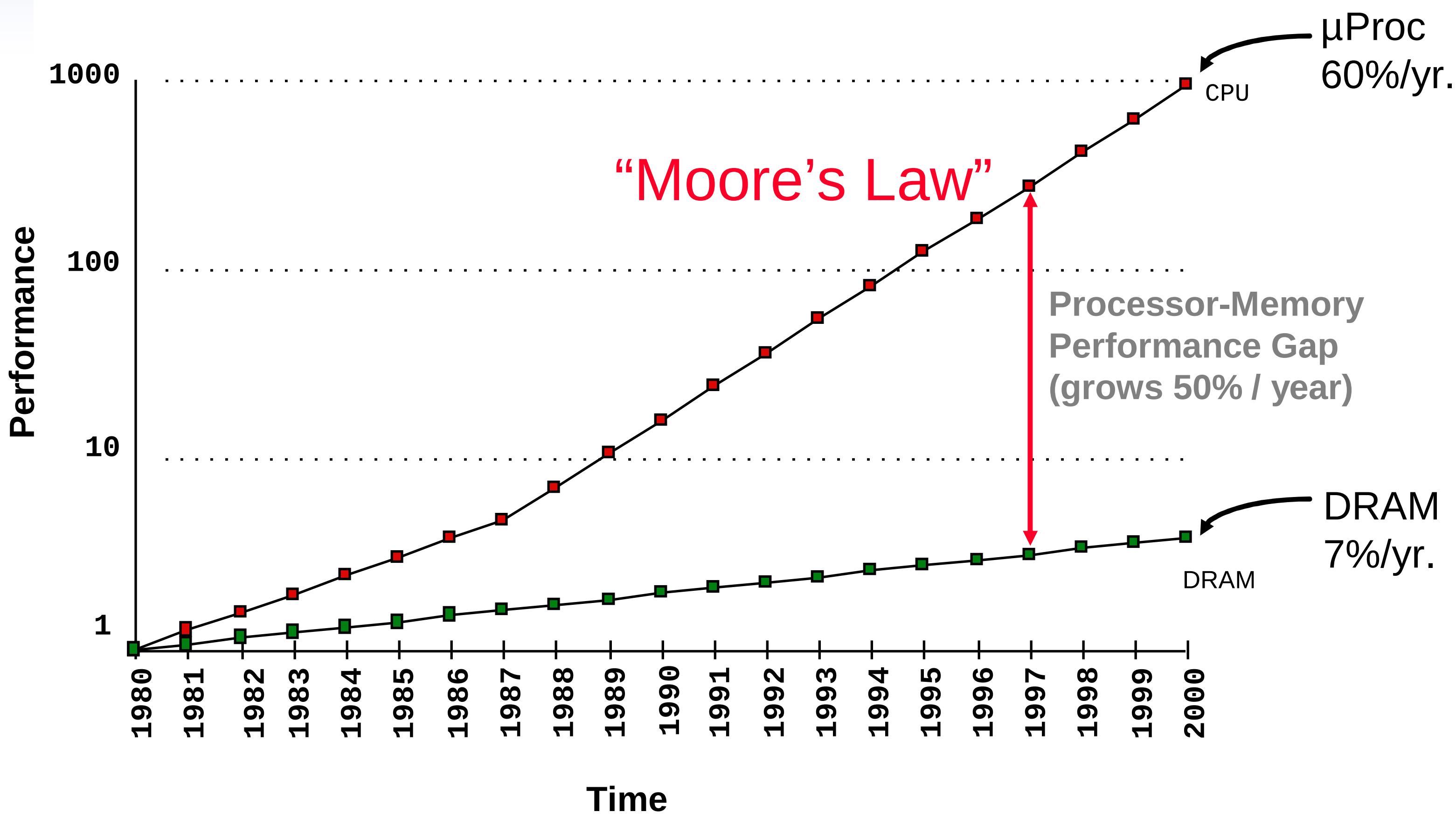
A μ-kernel does the job if

- 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

Today's Situation: Microprocessor

- Microprocessor-DRAM performance gap
  - Time of a full cache miss in instructions executed

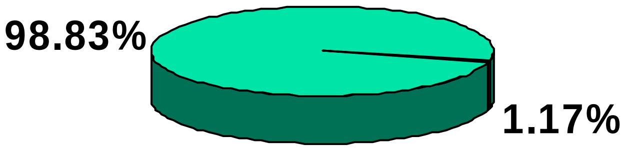
    1st Alpha (7000): 340 ns/5.0 ns = 68 clks x 2 instr. = 136 instr.

    2nd Alpha (8400): 266 ns/3.3 ns = 80 clks x 4 instr. = 320 instr.

    3rd Alpha (t.b.d.): 180 ns/1.7 ns =108 clks x 6 instr. = 648 instr.
  - 1/2X latency x 3X clock rate x 3X instr/clock $\Rightarrow$ ≈4.5X
- Minimize kernel-induced cache misses!

Slide originally from Dave Patterson, Parcon 98

Cache Working Sets

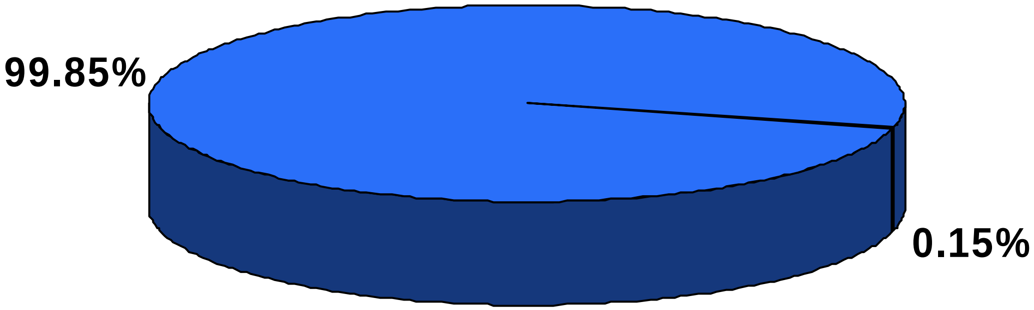


L1 cache

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

L2 cache

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



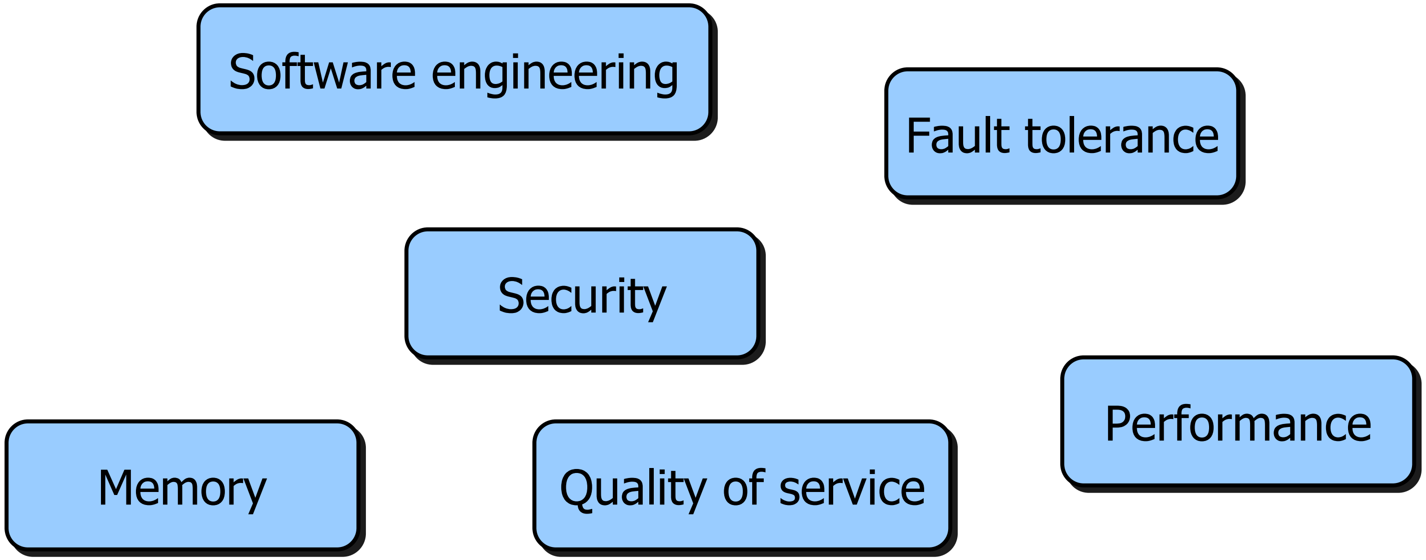
Multi-Processor Architectures

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

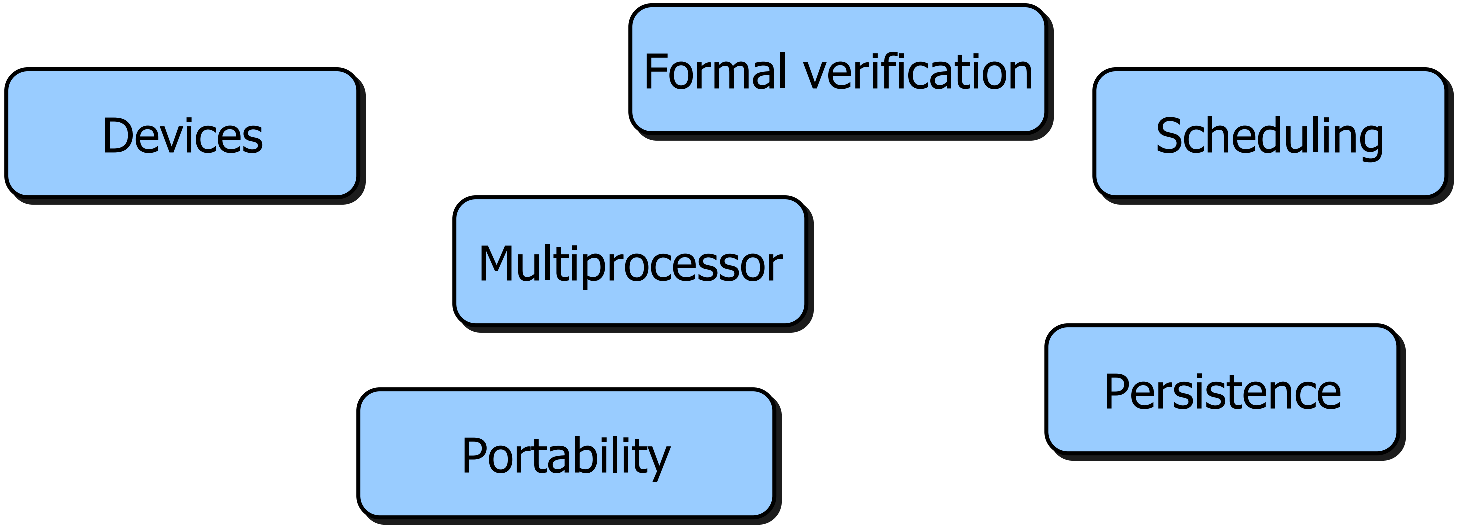
A Perfect Microkernel?

Proving minimality, necessarity and completeness would be nice but is impossible, since there is no agreed-upon metric and all is Turing-equivalent.

Jochen Liedtke, *On µ-Kernel Construction*, SOSP '95



Small Kernel ≠ Small Problem



μ-Kernel Design

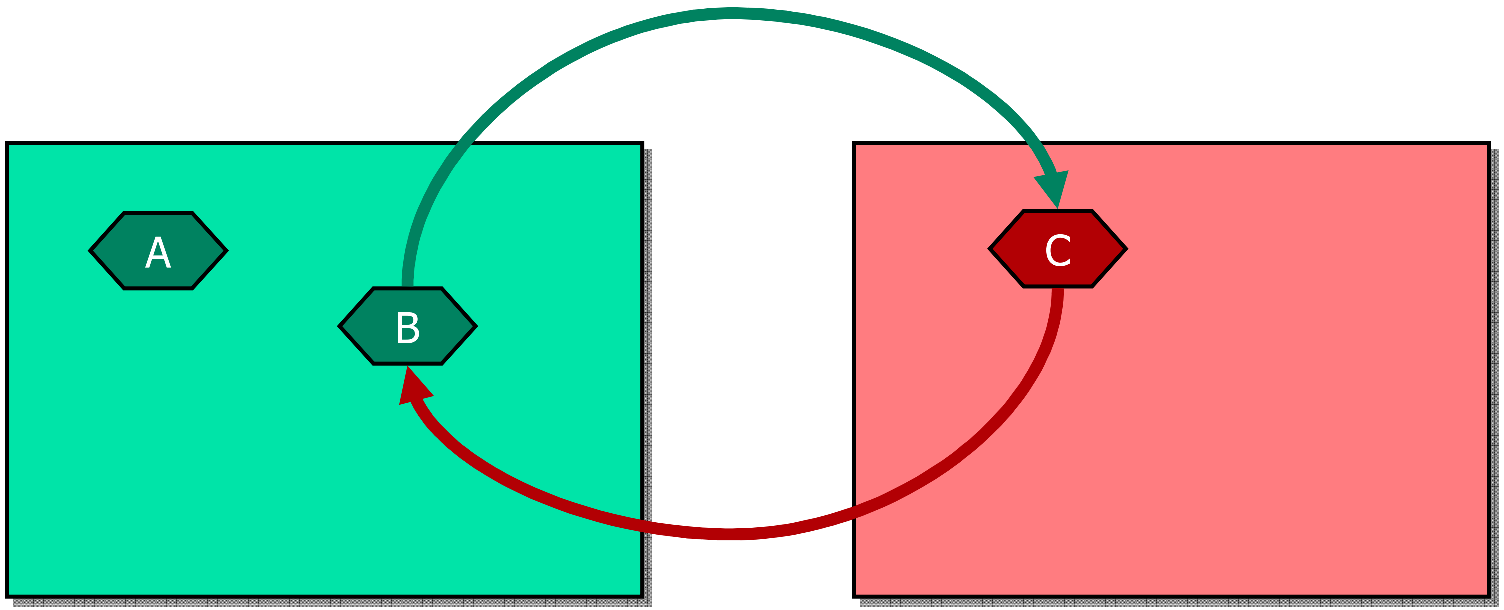
- A μ-kernel does no real work
  - μ-kernel services are only required to overcome μ-kernel constraints

- Therefore, μ-kernels have to be infinitely fast!

Minimality is the key!

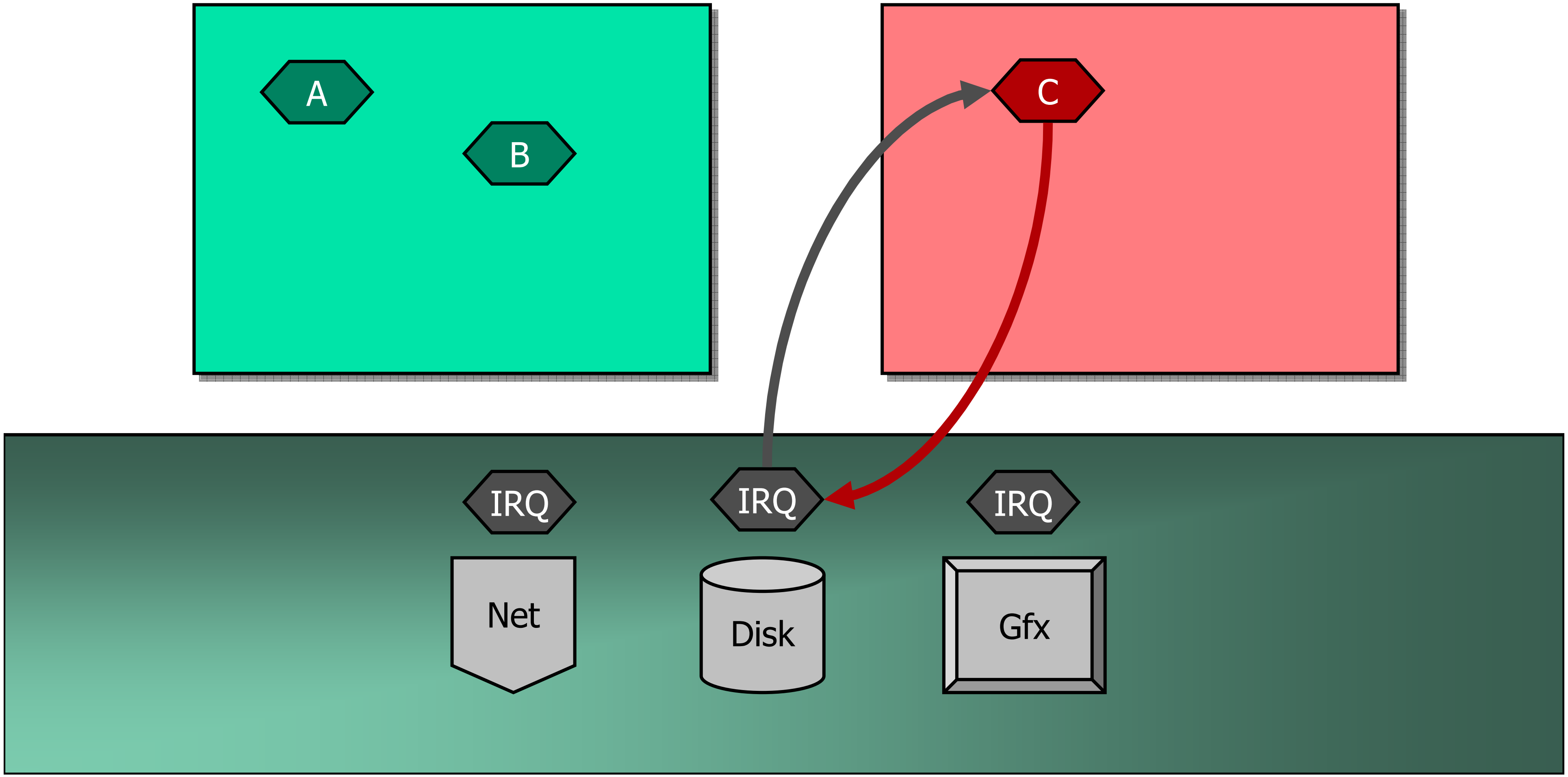
- **Threads** ***IPC***
- **Address Spaces** ***Mapping***

Threads – IPC

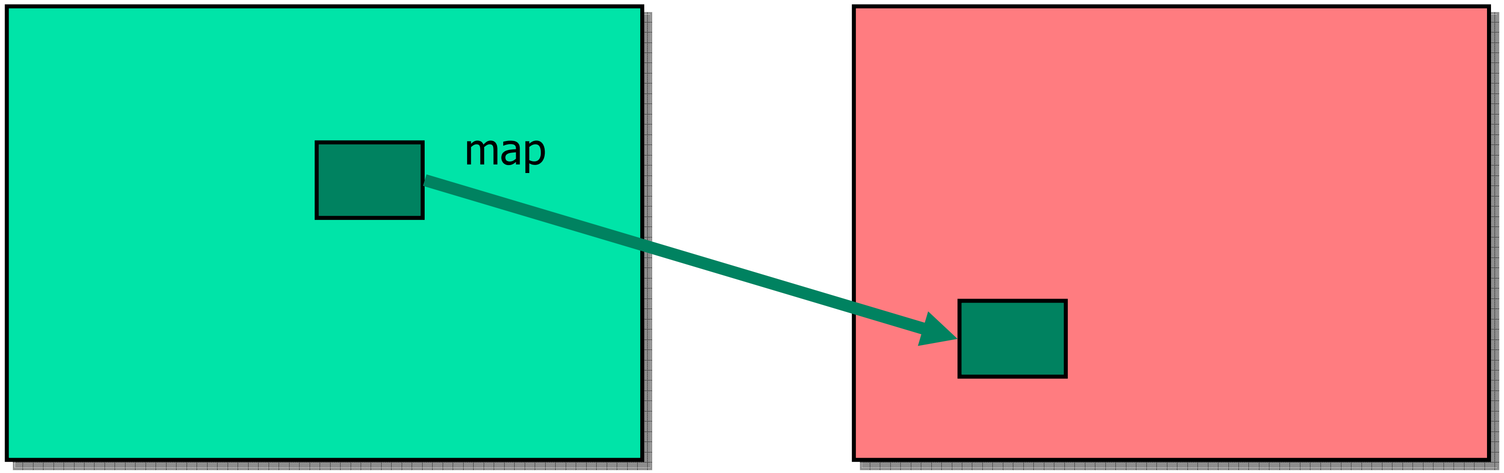


- Enable controlled communication across address space boundaries

User-Level Device Drivers

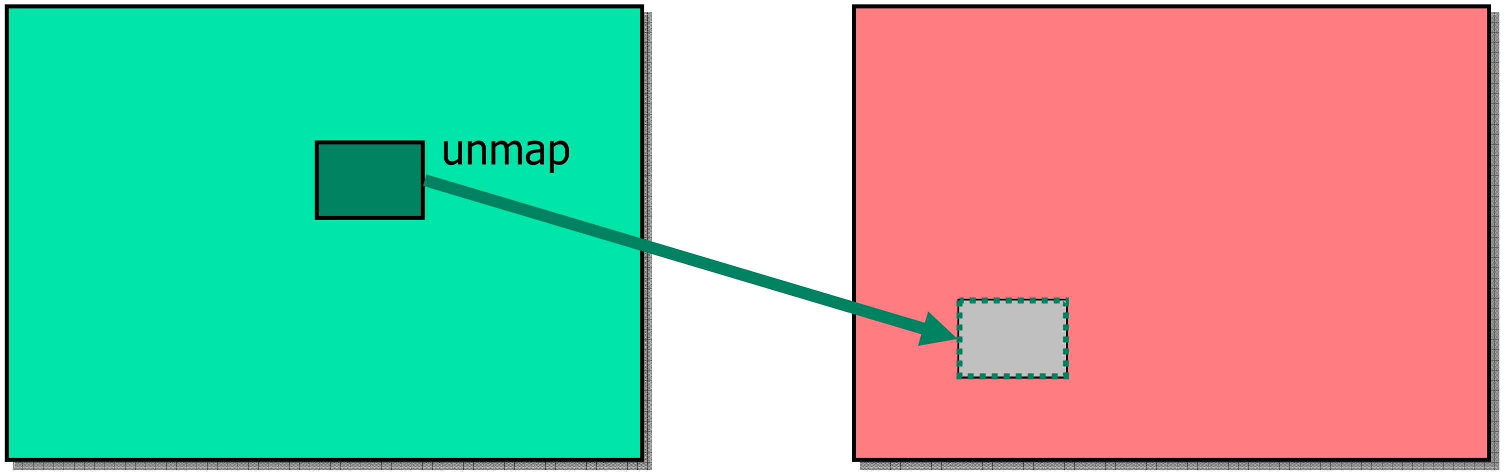


Address Spaces – Mapping



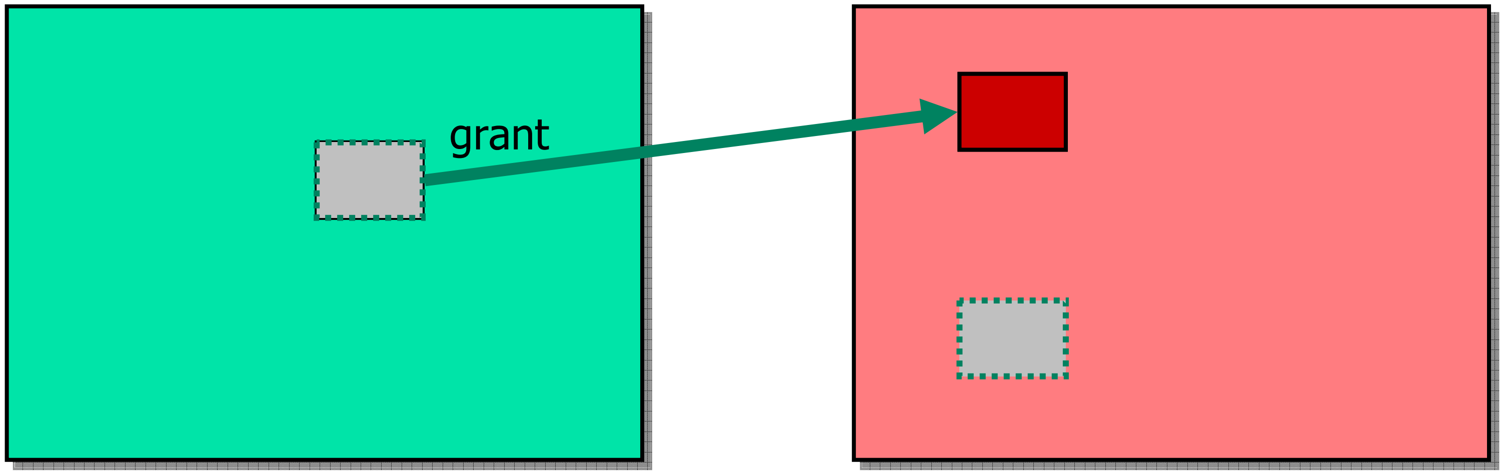
- Setup shared memory regions

Address Spaces – Mapping



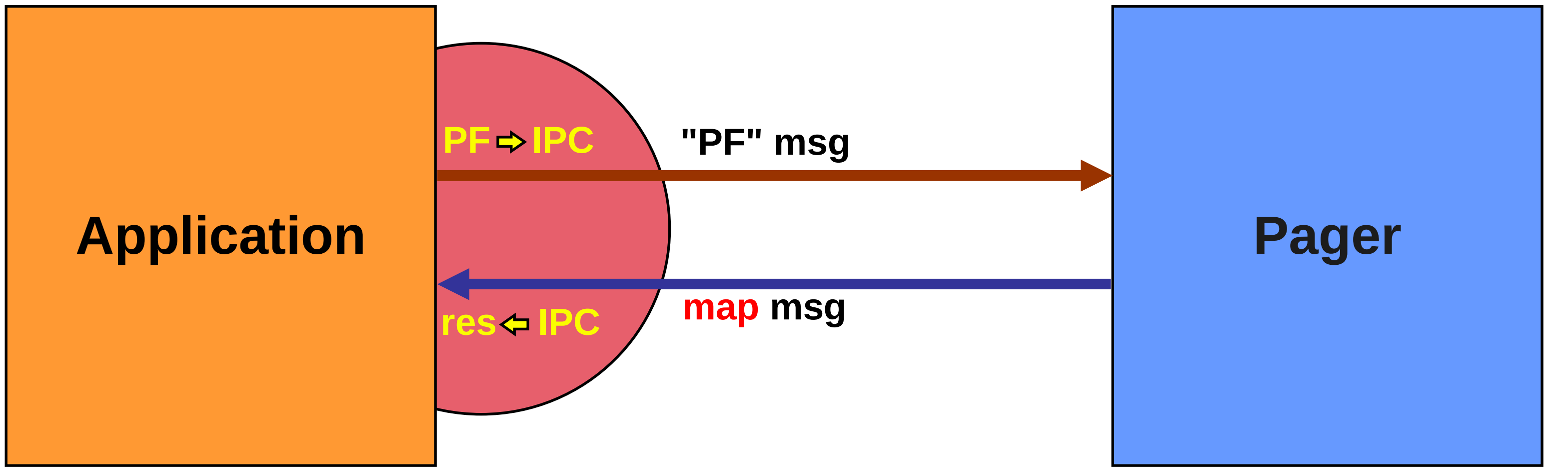
- Revoke shared memory regions

Address Spaces – Mapping

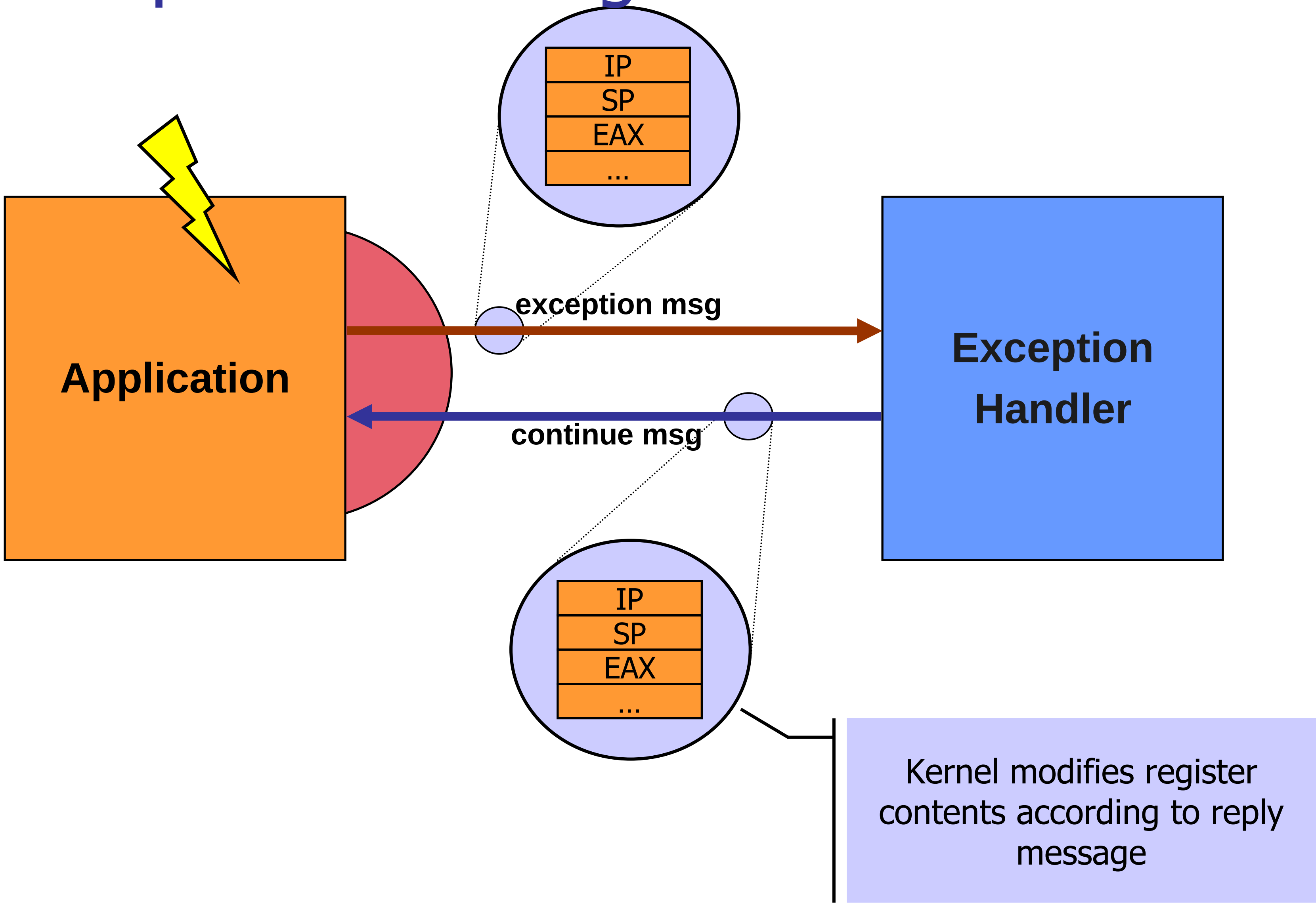


- Donate memory regions to others
- Frees up virtual memory in the granting space
  - Required for file servers, … (at least useful)

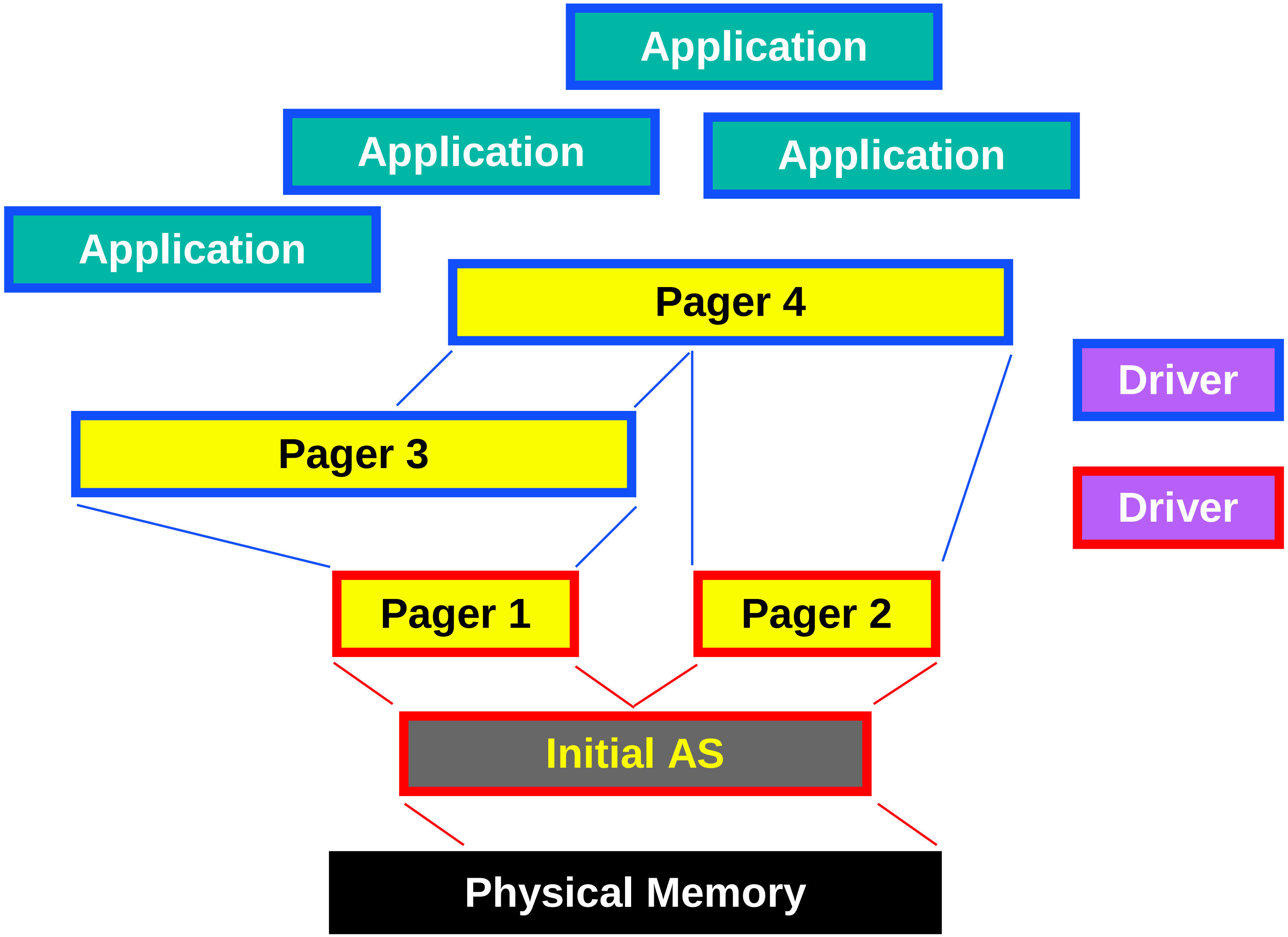
Page Fault Handling



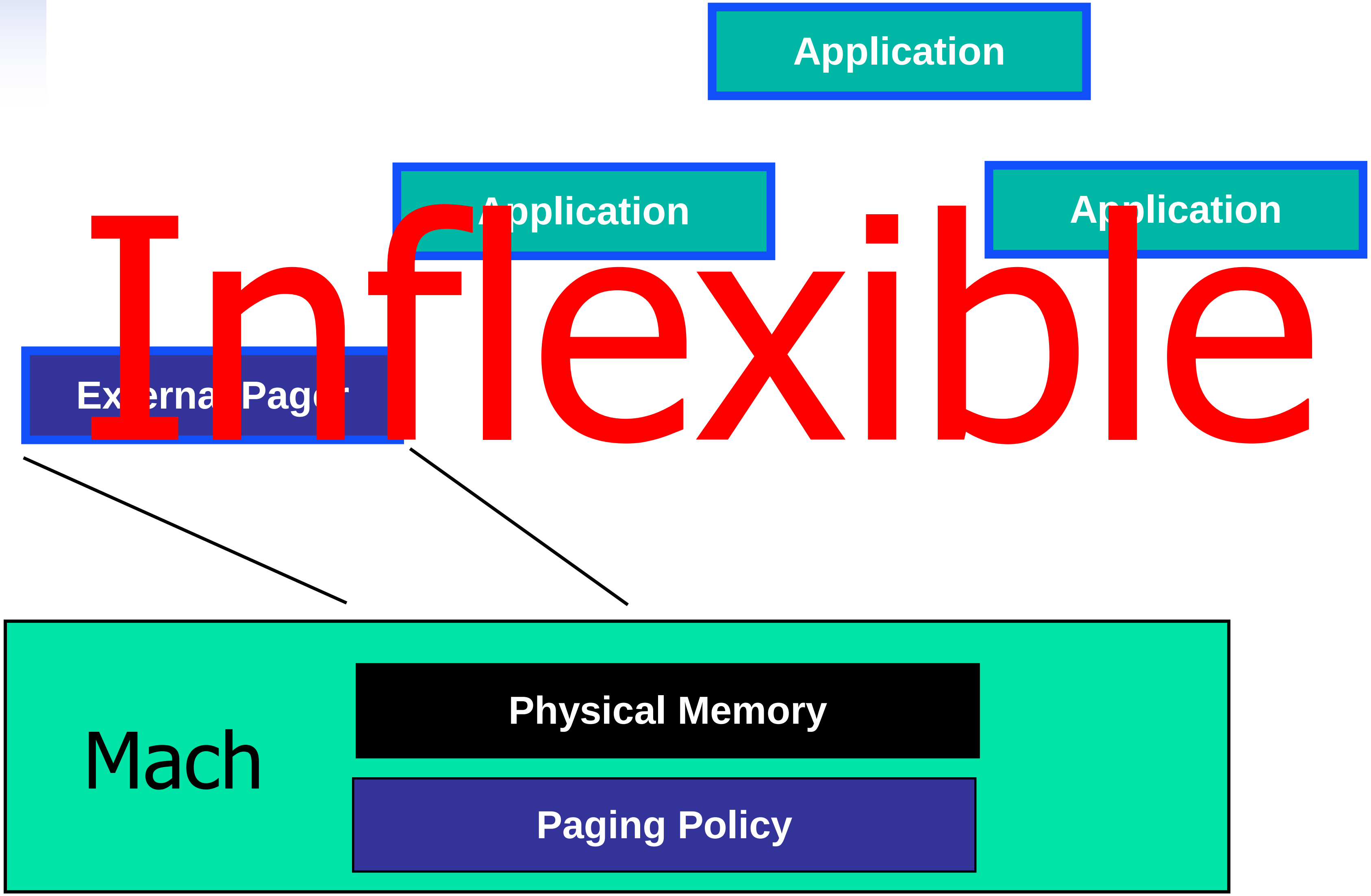
Exception Handling



Recursive Address Spaces



Mach Virtual Memory



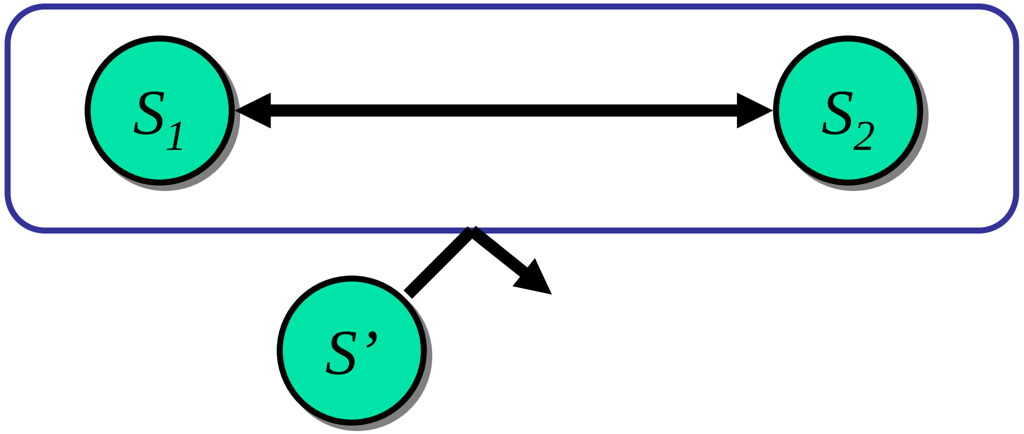
Tolerated Concepts

A concept is tolerated in the μ-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



Requirement: Address Spaces

- µ-kernel must hide hardware address spaces
  - Otherwise violates integrity principle
- But must permit arbitrary protection schemes
  [and non-protection]
- Solution: recursive construction of address spaces outside the kernel

Requirement: Threads

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

Why Tolerate Threads in μ-kernels?

- The decisive reason: $\sigma(\tau)$
- Modifications to address spaces

$$\sigma(\tau) := \sigma'$$

  must be controlled by the kernel
- Thus a notion of τ that represents the above activity

- Additionally: concurrency

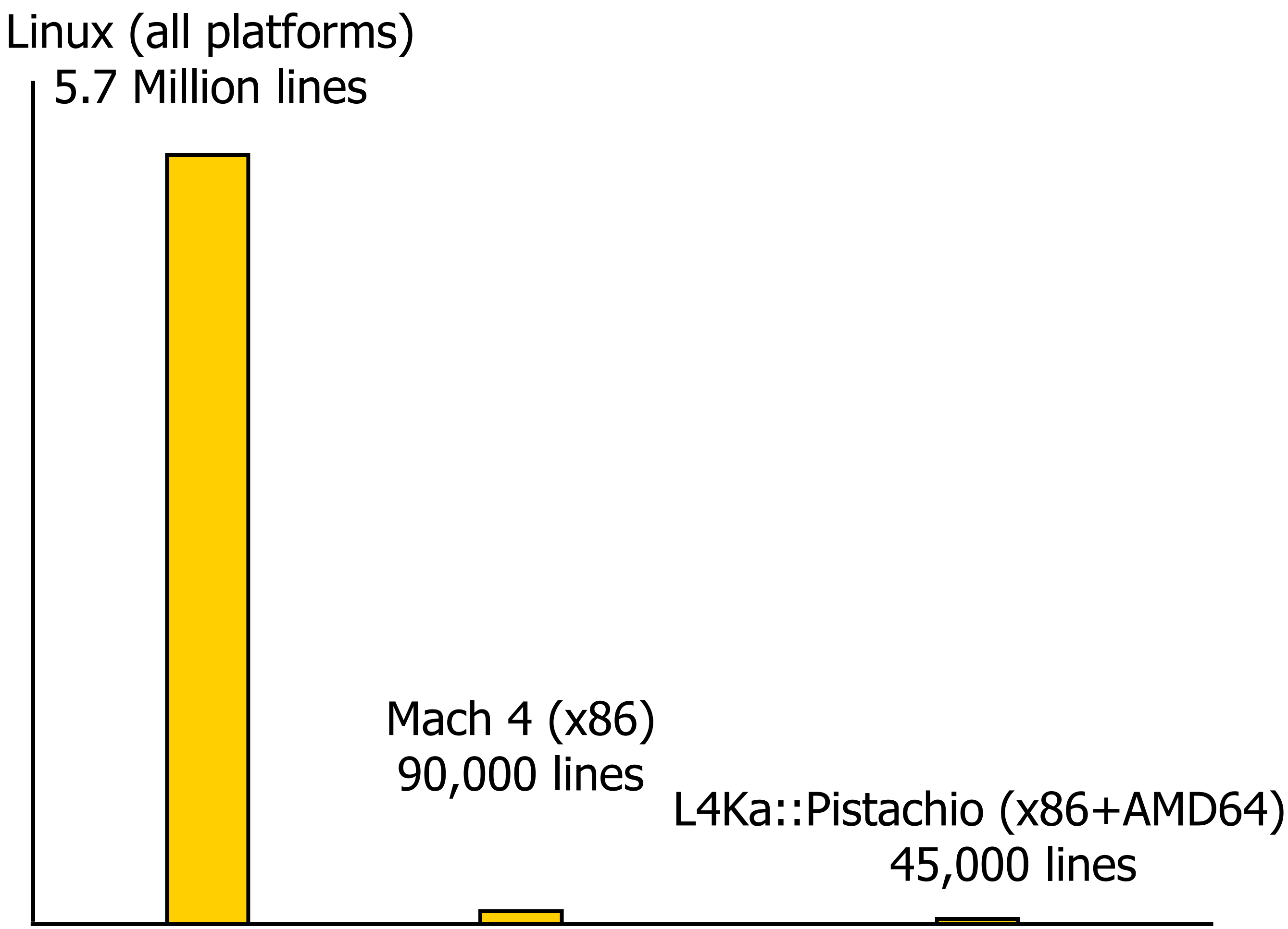
Requirement: IPC

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

- Classical approach
  - Transfer messages between threads

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

Size Comparison



In this course …

You **do** learn

- How to design a µK
- Why L4 is sooooo fast
- Reasons why others failed
- Nitty-gritty details about IA32
- Some OS bashing …
- More cool stuff …

You **don’t** learn

- How to construct a system on a µK (→SDI)
- Linux dos and don’ts
- Operating system X is better than Y

Plan

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

Many algorithms, often influencing the system design.