The L4 ecosystem

Berlin, Dec 2004

System Issues

- Convential systems are
  - Complex
    - Linux kernel at least 500,000 LoC
  - Prone to errors
    - Drivers
  - All system components run in privileged mode
  - Inflexible
    - Global policies
  - Large Trusted Computing Base (TCB)

Insights

Observation:

Most kernel functionality does not need CPU privileges, like:

- Filesystems
- Driver functionality
- User management

What is really needed

Jochen Liedtke: “A microkernel does no real work”

⇨ Kernel provides only inevitable mechanisms

⇨ No policies enforced by the kernel

What is inevitable?

Abstractions	Mechanisms
▪ Threads	▪ Communication
	▪ Scheduling
▪ Address Spaces	▪ Safe construction

⇨ This should be sufficient for everything

The Marred Perception of Microkernels

- Supposed to be slow
  - Not true any more
- No obvious benefit
  - Infamous dispute Torvalds vs. Tannenbaum
  - How much worth is manageability of complexity?
- GNU Hurd
  - Late
  - Slow
  - Constantly lagging behind other OS in functionality

The Case for Microkernels

- Complexity needs to be handled
  - Structure beyond monolithic kernels are needed
  - Several candidates
    - Virtualisation
    - Paravirtualisation
    - Microkernel
- Implementation of some functionality is even simplified
  - Real time
    - DROPS
    - RTLinux
  - Security
    - Substantially smaller trusted computing base

Agenda

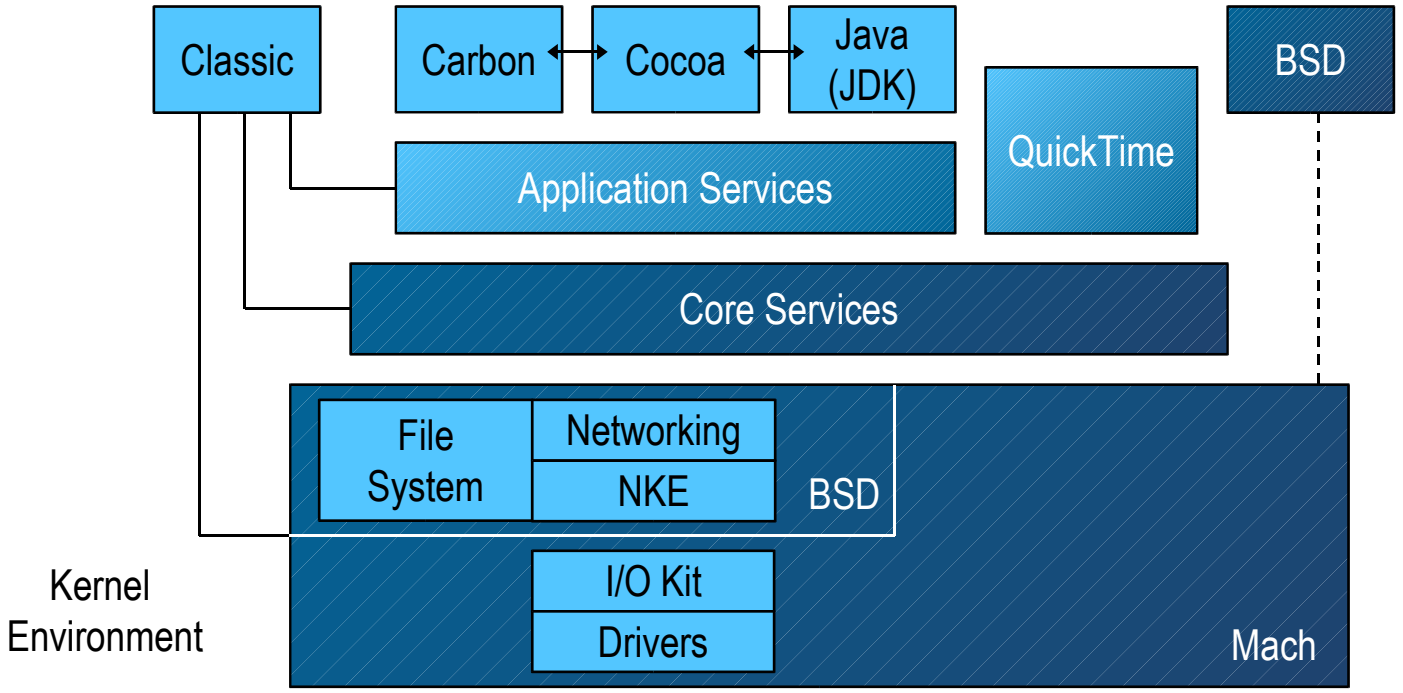
- Introduction
- Agenda
- L4 µ-kernels
- Legacy support
- Security

The Rise and Demise of the First Microkernel

- First interest in mid-eighties
- Mach
  - Started with stripped down UNIX kernels
  - High level abstractions
    - Asynchronous messages
    - Ports
    - Virtual memory management
  - Adopted by IBM for future OS development
    - Disastrous results
  - None of the ambitious goals achieved
    - The idea seemed to be a failure
- **BUT**: still alive in MacOS X

Mac OS X

- Mac OS X Kernel (Darwin) based on Mach/BSD
- Drivers / BSD services run in kernel mode



Getting it Right the 2nd Time

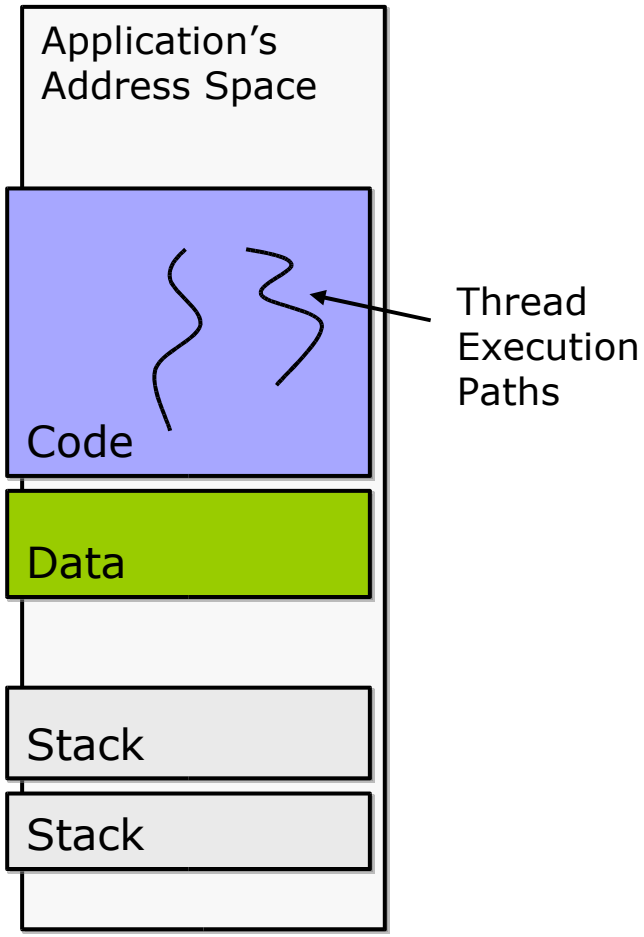
- Jochen Liedtke published ground-breaking results in the mid-90ies
- Bottom-up approach
  - Mach started with a UNIX kernel
- Kernel provides only minimal functionality
  - Address spaces with threads
  - Inter-Process Communication (IPC)
  - Hierarchical memory management
- Under active development
  - V.2, X.0, X.2
  - 9 supported architectures (L4Ka::Pistachio)
    - Alpha, ARM, IA32, AMD64, IA64, Mips64, PPC32, PPC64, SPARCv9

Threads

Abstraction and unit of execution

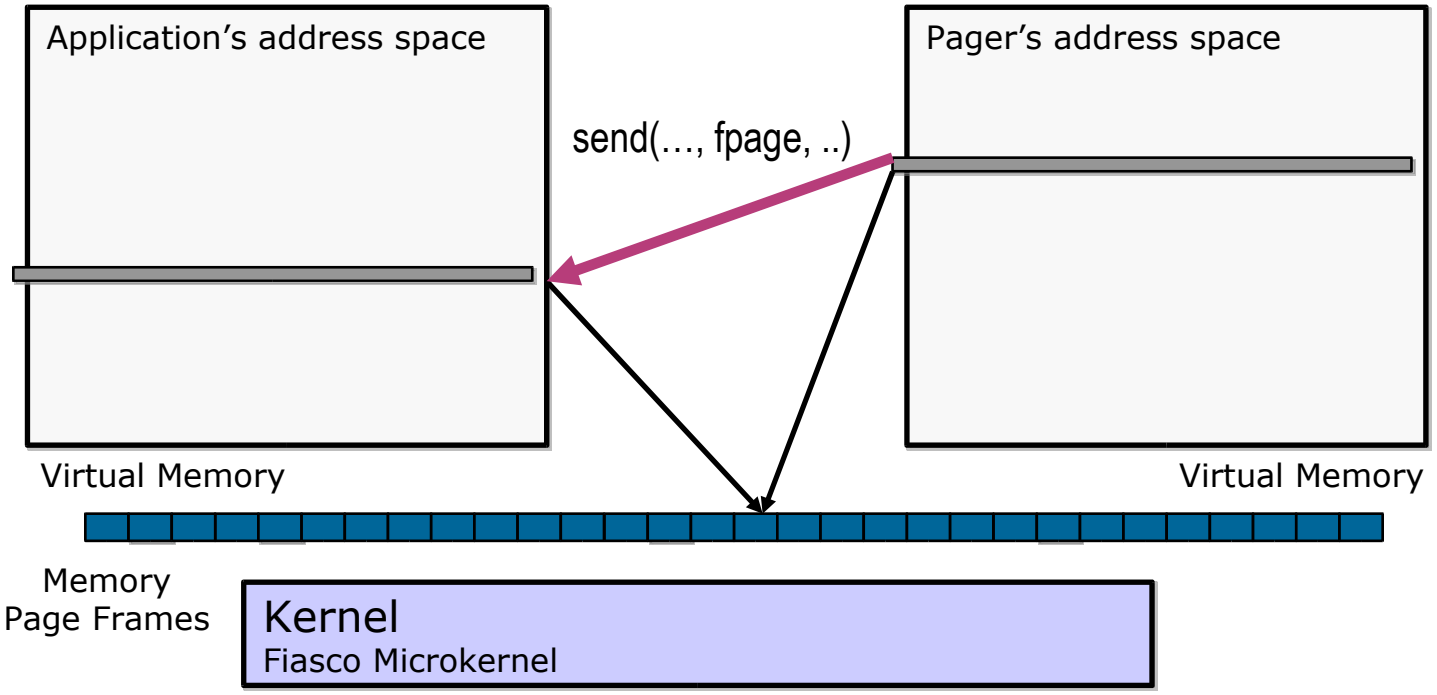
- Identified by thread id
- Consists of
  - Instruction pointer
  - Stack
  - Registers, flags, …
  - ⇨ *Thread state*
- L4 only manages (preserves) IP, SP and registers

⇨ Entry point, stack allocation (size, location) and memory is managed by user-level applications

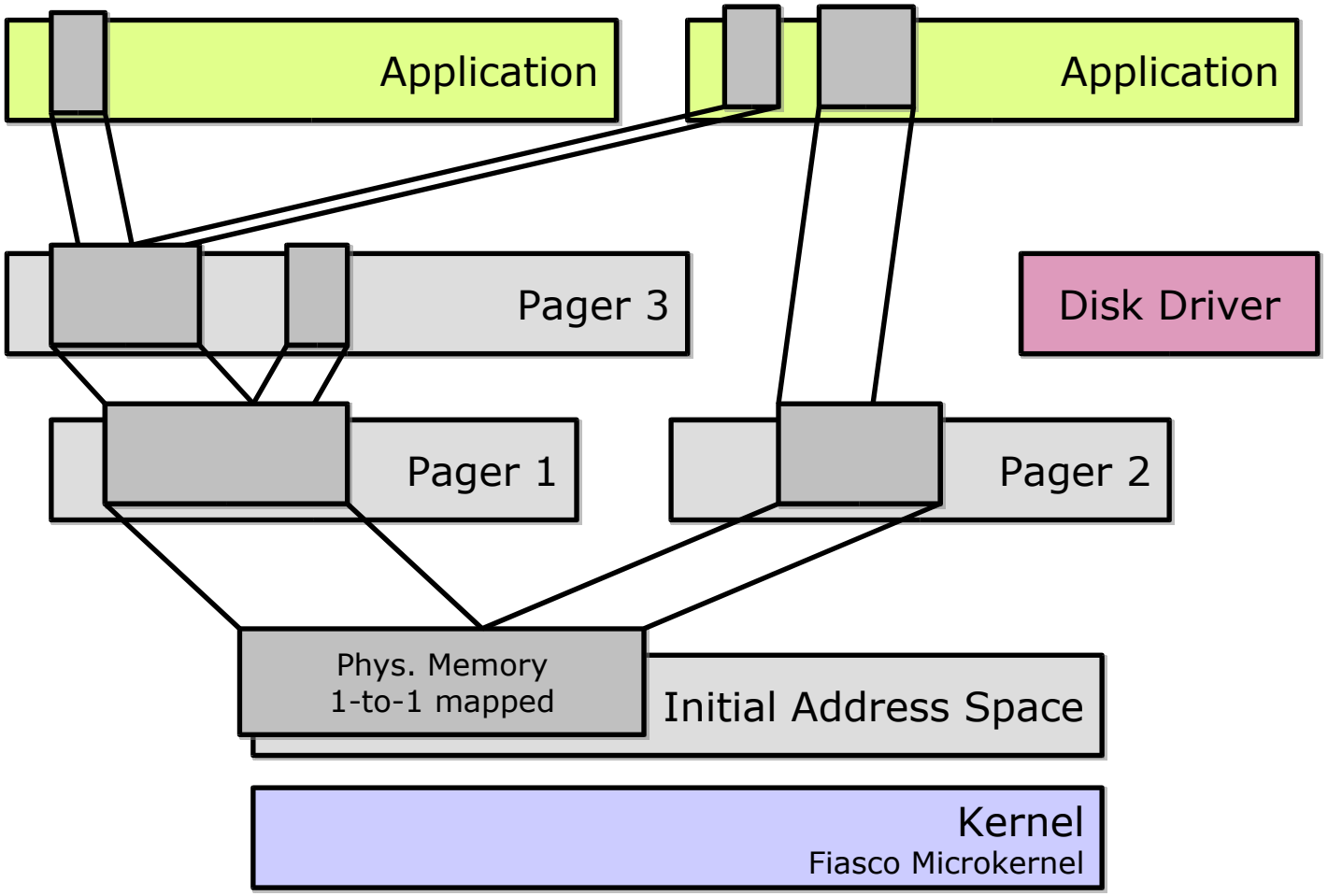


Page Mapping

- Entry in virtual memory points to page frame in phys. memory
- ⇨ Map creates an entry in the receiver's address space pointing to the same page frame
- ⇨ Only valid entries in pager's address space can be mapped to clients



L4 Hierarchical Memory Management

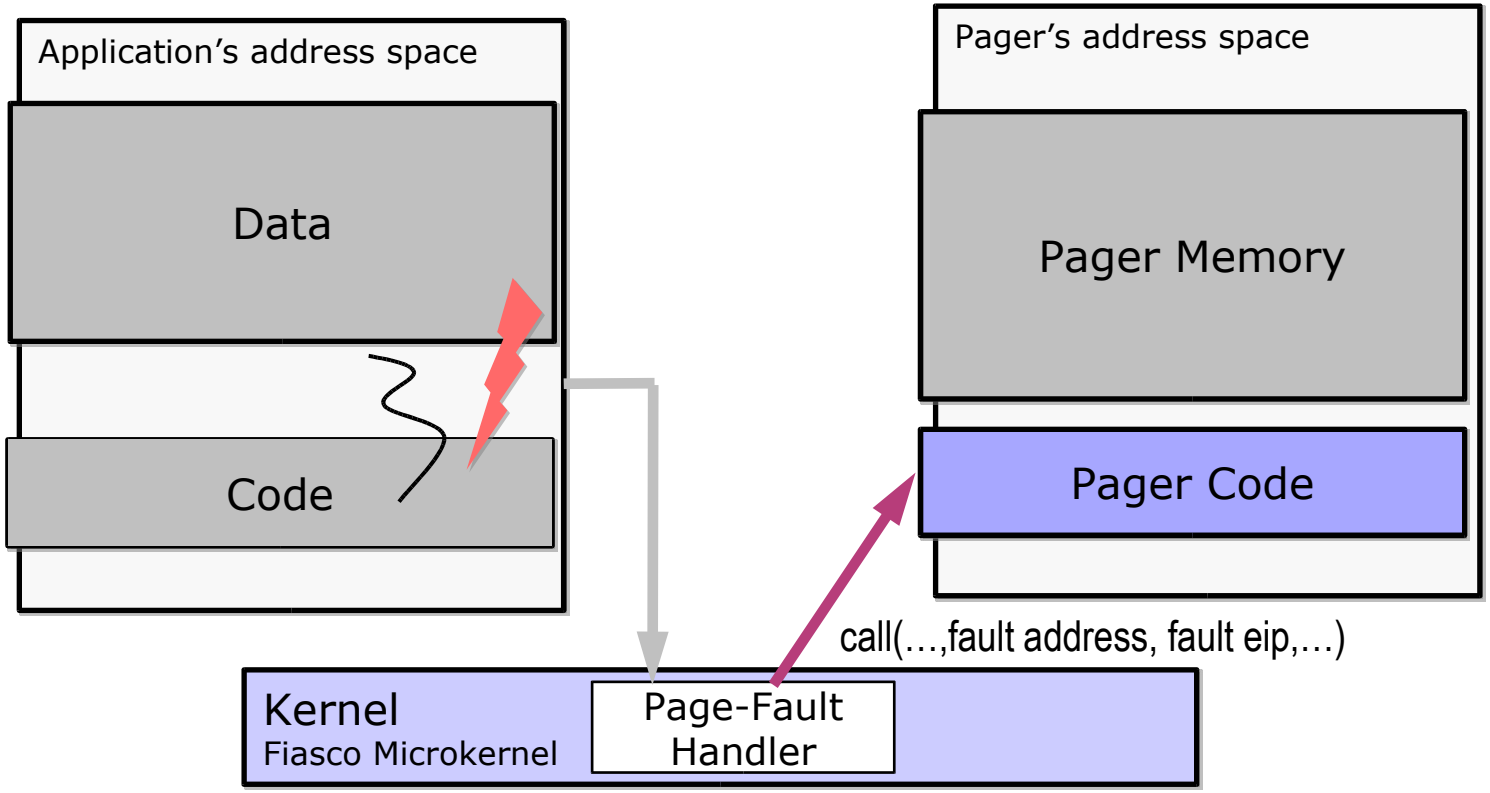


Page-Fault Handling

- Communication with pager thread ⇨ IPC

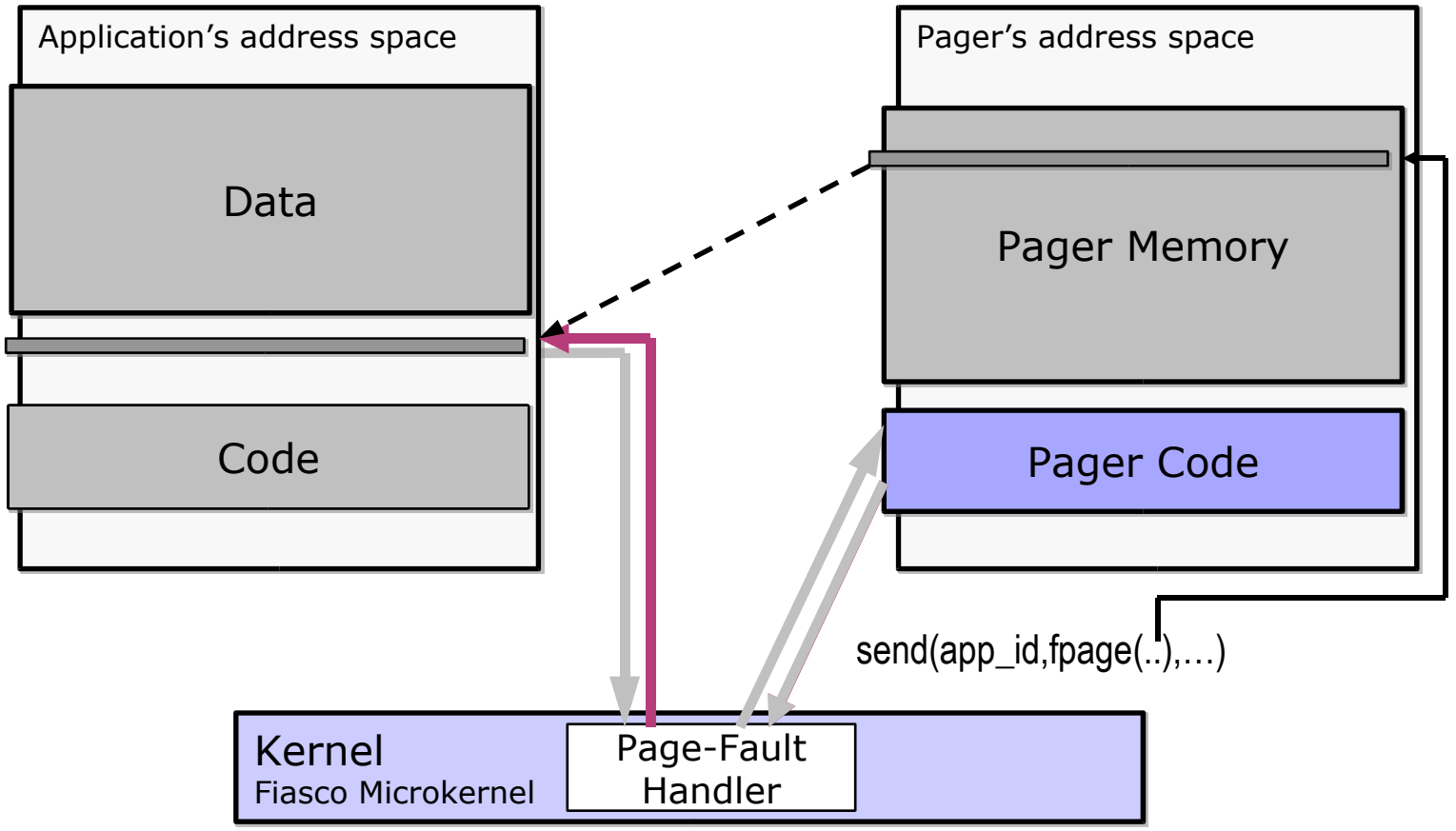
⇨ Kernel page-fault handler sets up IPC to pager

- Pager sees faulting thread as sender of IPC



Page-Fault Resolution

- Pager *maps* pages of his own address space to the address space of the client

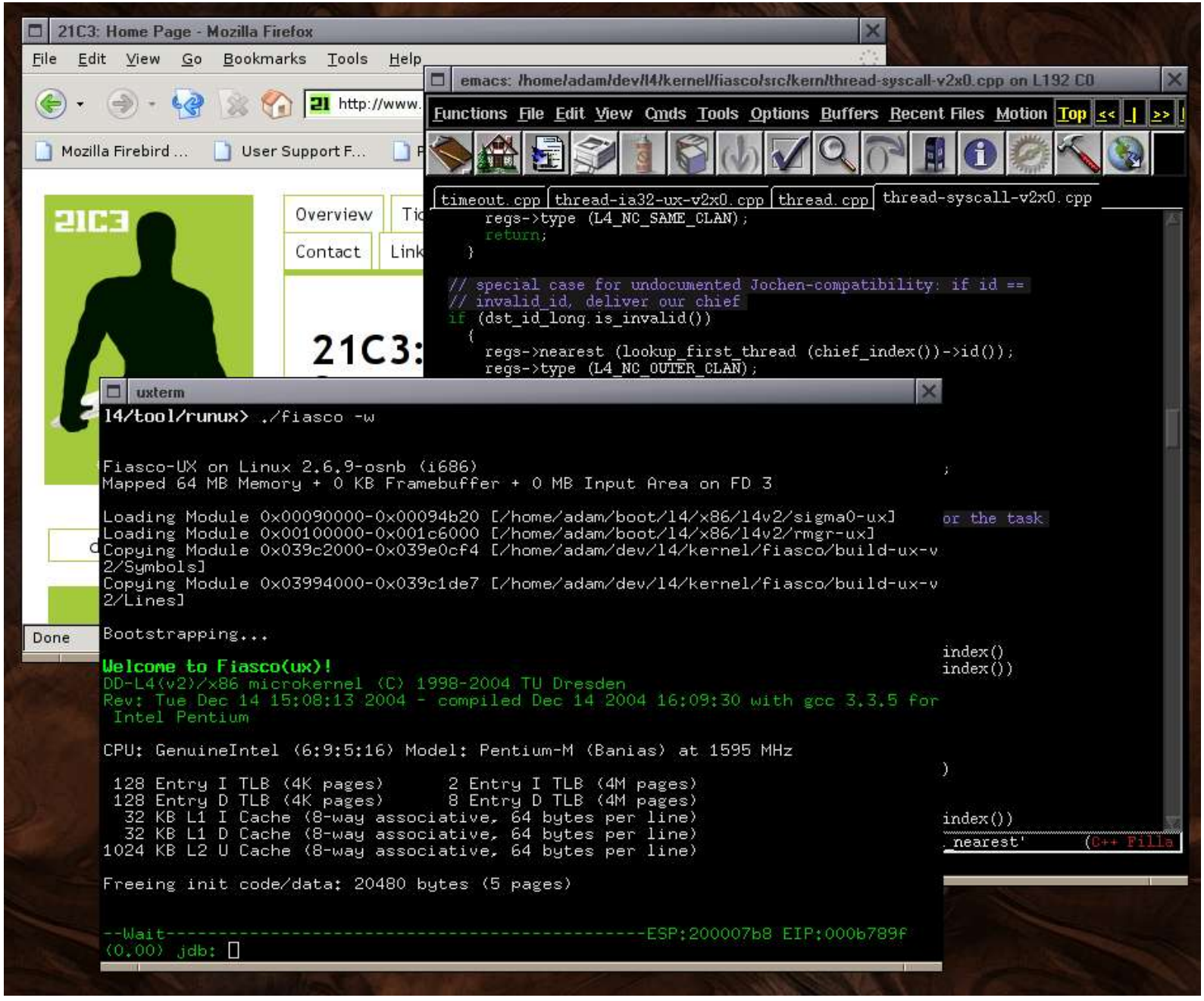


The Fiasco µ-kernel

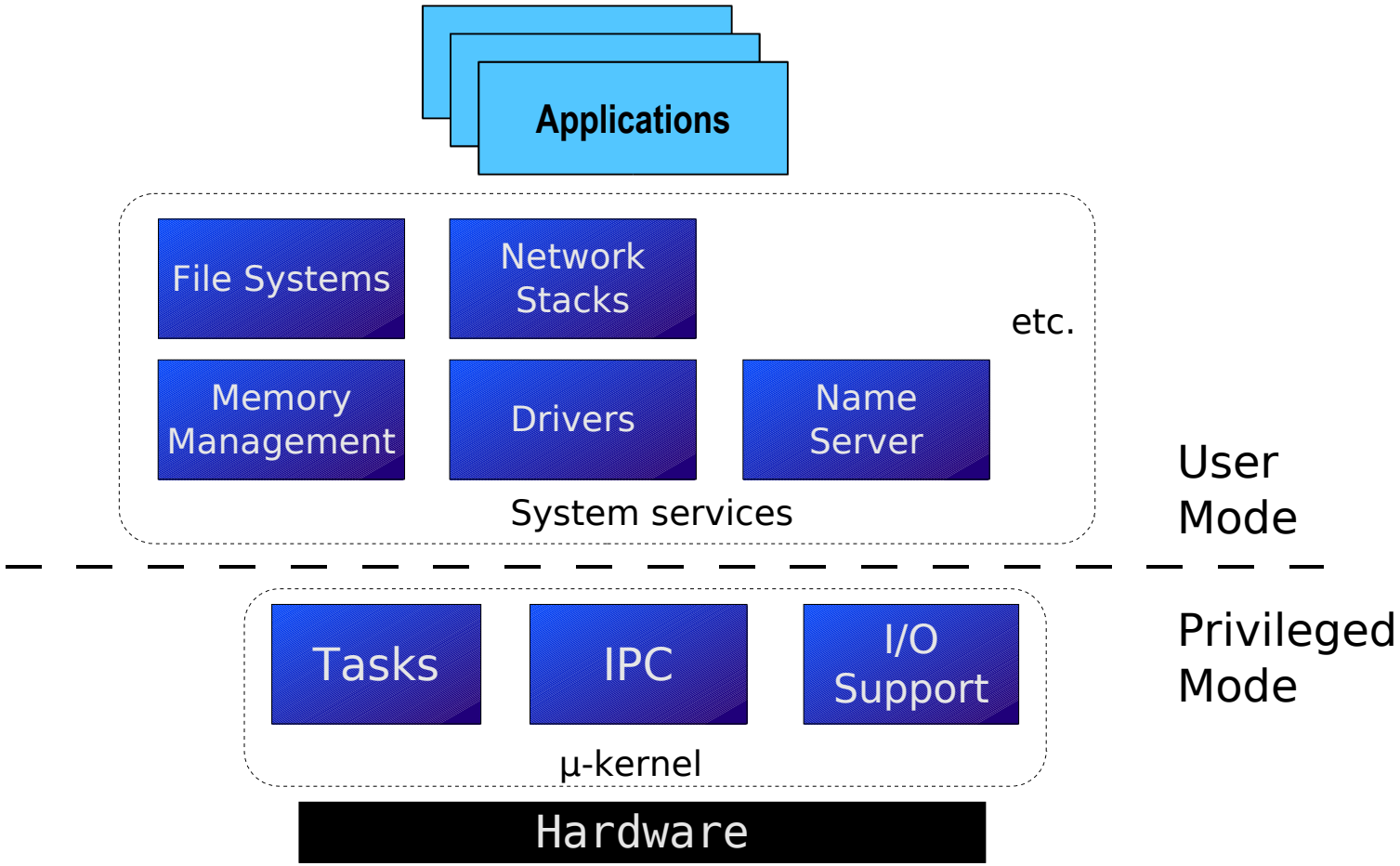
- Started by Michael Hohmuth in 1997
  - no free L4 implementation was available
  - Kernel for TUD OS projects
- Uses lock- and wait-free synchronization to be fully preemptible
  - → Prerequisite for Real-Time
- Written in C++
- Available for x86 and ARM
  - AMD64 to come

Fiasco-UX

- Port of Fiasco to Linux
  - similar to UML
- Easy test and development
  - No test hardware required
- Supports other L4 projects
  - DoPE, the L4 native GUI
  - L^4Linux


21C3: Home Page - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Mozilla Firebird ...
User Support F...
21C3
Overview
Contact
21C3:
Done
emacs: /home/adam/dev/l4/kernel/fiasco/src/kern/thread-syscall-v2x0.cpp on L192 C0
Functions File Edit View Cmds Tools Options Buffers Recent Files Motion Top
timeout.cpp
thread-ia32-ux-v2x0.cpp
thread.cpp
thread-syscall-v2x0.cpp
regs->type (L4_NC_SAME_CLAN);
return;
// special case for undocumented Jochen-compatibility: if id ==
// invalid_id, deliver our chief
if (dst_id_long.is_invalid())
regs->nearest (lookup_first_thread (chief_index())->id());
regs->type (L4_NC_OUTER_CLAN);
uxterm
l4/tool/runux> ./fiasco -w
Fiasco-UX on Linux 2.6.9-osnb (i686)
Mapped 64 MB Memory + 0 KB Framebuffer + 0 MB Input Area on FD 3
Loading Module 0x00090000-0x00094b20 [/home/adam/boot/l4/x86/l4v2/sigma0-ux]
Loading Module 0x00100000-0x001c6000 [/home/adam/boot/l4/x86/l4v2/rmgr-ux]
Copying Module 0x039c2000-0x039e0cf4 [/home/adam/dev/l4/kernel/fiasco/build-ux-v2/Symbols]
Copying Module 0x03994000-0x039c1de7 [/home/adam/dev/l4/kernel/fiasco/build-ux-v2/Lines]
Bootstrapping...
Welcome to Fiasco(ux)!
DD-L4(v2)/x86 microkernel (C) 1998-2004 TU Dresden
Rev: Tue Dec 14 15:08:13 2004 - compiled Dec 14 2004 16:09:30 with gcc 3.3.5 for Intel Pentium
CPU: GenuineIntel (6:9:5:16) Model: Pentium-M (Banias) at 1595 MHz
128 Entry I TLB (4K pages)
2 Entry I TLB (4M pages)
128 Entry D TLB (4K pages)
8 Entry D TLB (4M pages)
32 KB L1 I Cache (8-way associative, 64 bytes per line)
32 KB L1 D Cache (8-way associative, 64 bytes per line)
1024 KB L2 U Cache (8-way associative, 64 bytes per line)
Freeing init code/data: 20480 bytes (5 pages)
--Wait----------ESP:200007b8 EIP:000b789f
(0,00) jdb:

Building systems with L4 – system design



System Core Services

- µ-kernel alone doesn't do much
- Need several basic services
  - Initial task
  - Name server
  - Memory management
  - Loader
  - File provider
- Programming support libraries
  - libc
  - Thread handling
  - Synchronization
  - ...

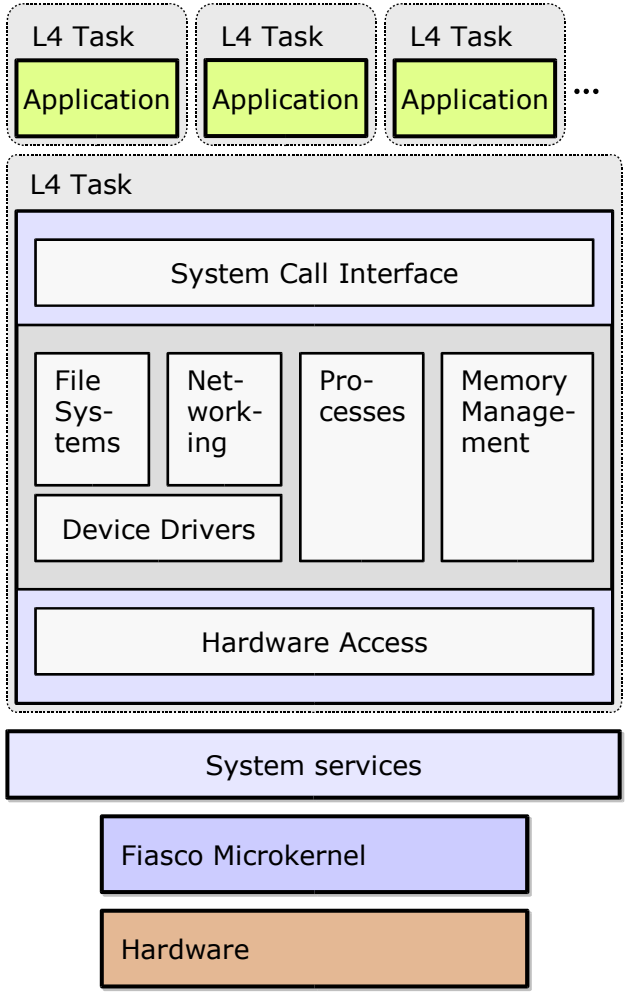
This basic functionality is called L4 environment, **L4Env**, and provides a higher-level abstraction of the kernel API.

Linux on L4

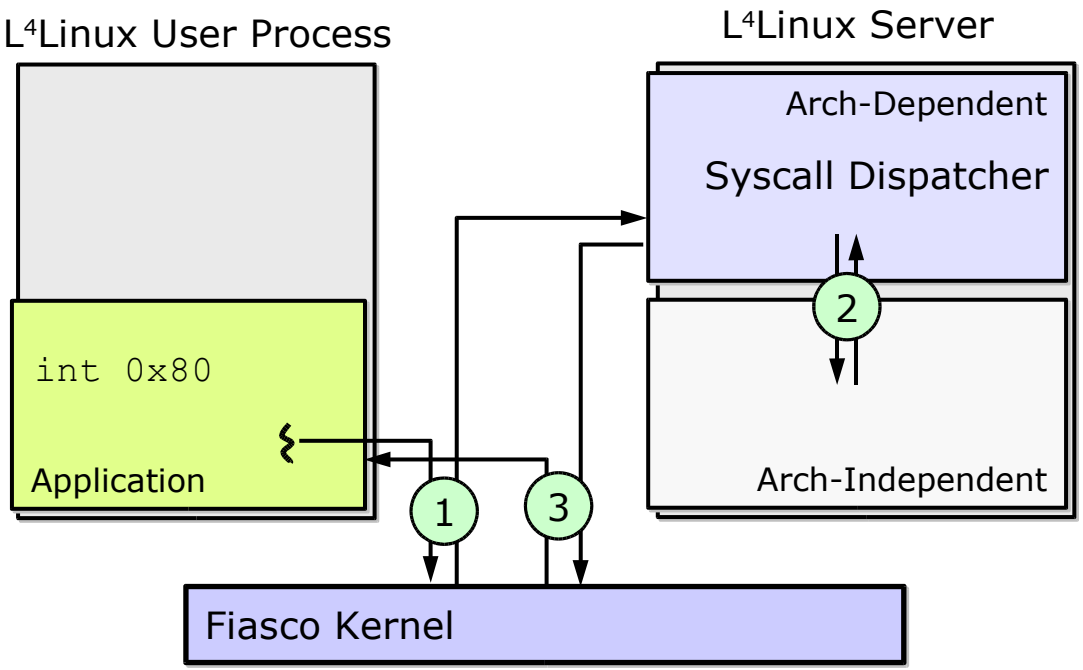
- A port of the Linux kernel to L4
- Support legacy operating systems on L4
  - Binary compatible with Linux applications
  - Runs standard distributions
- Started in 1996 with Linux 2.0
- Latest: 2.6, based on L4Env

L4Linux

- Linux kernel runs in an L4 tasks
- The Architecture-dependent part uses L4 primitives
  - Threads
  - Mappings
  - IRQ IPC



System Call Handling



1. Exception IPC to exception handler
2. Server handles system call
3. Exception reply to resume execution

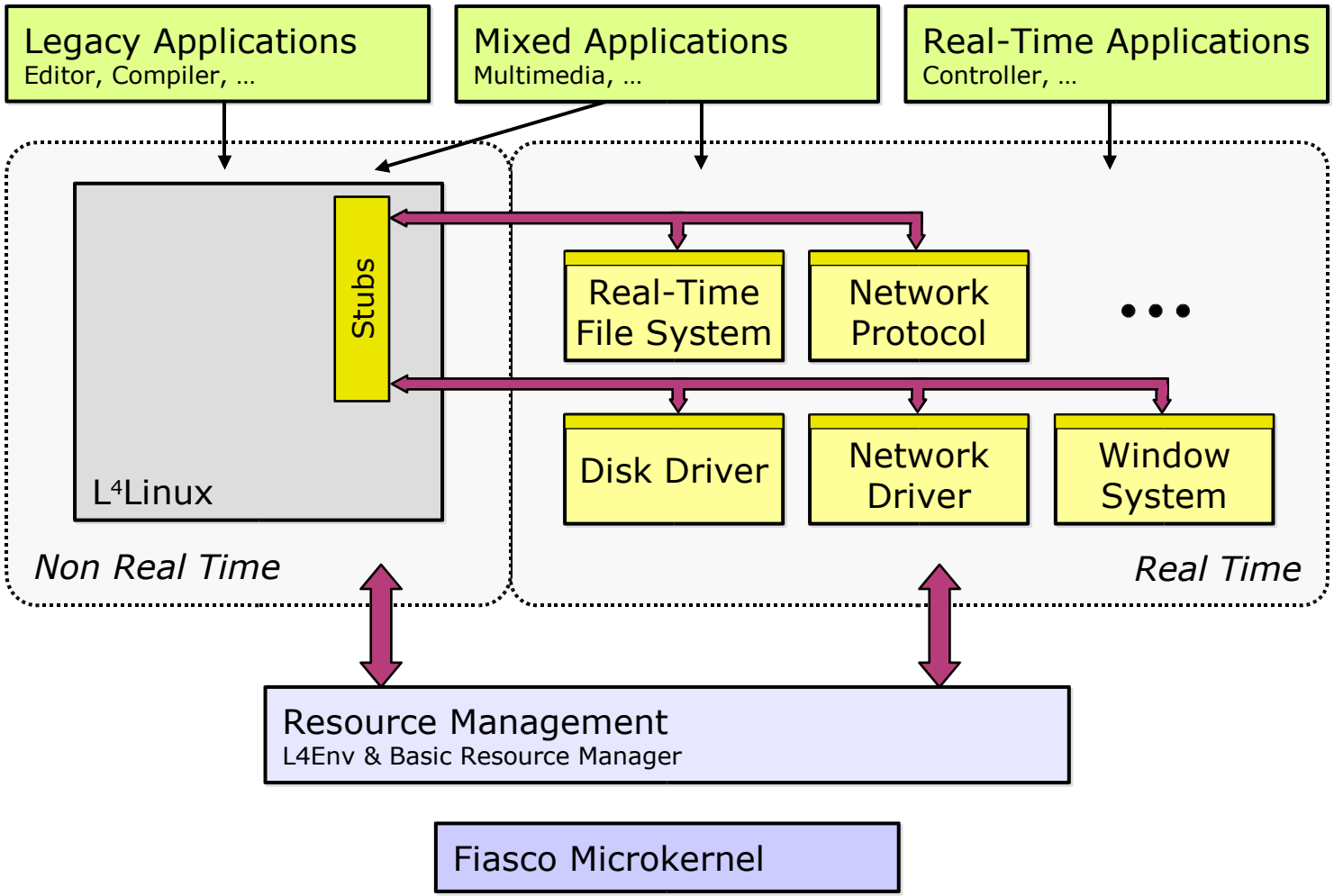
- On x86, the kernel is entered for system calls via „int 0x80“
- On L4, this will result in an exception
- Exceptions are delivered as an IPC to the exception handler

DROPS, the **D**resden **R**ealtime **OP**erating **S**ystem

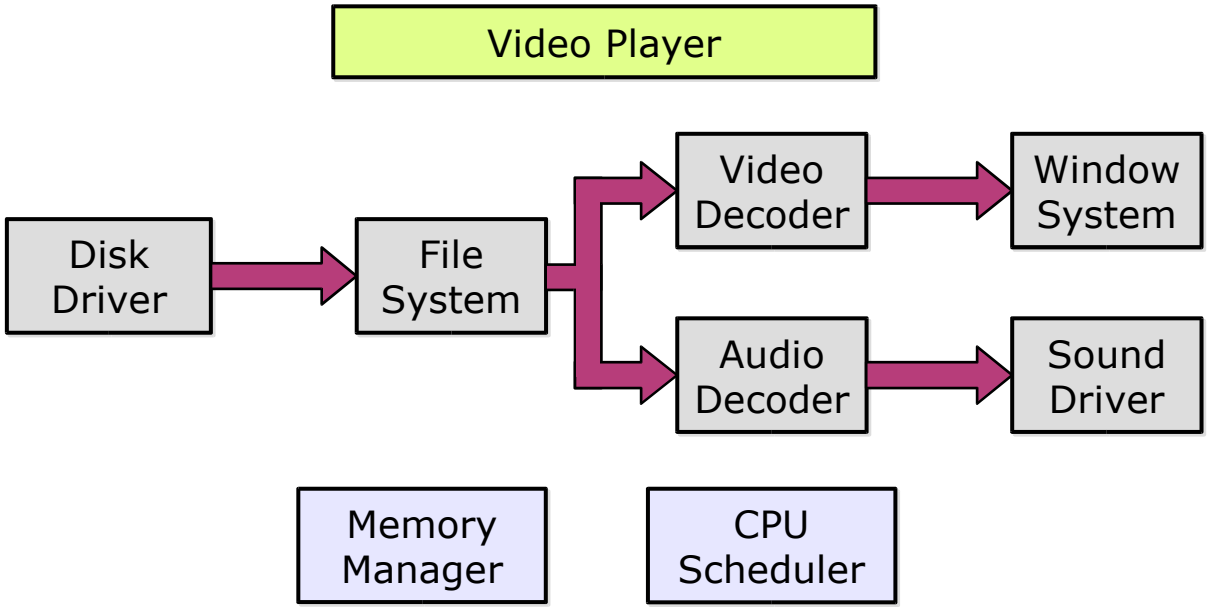
- Allow the coexistence of real-time and non-real-time applications
  - Common property of current applications, e.g. multimedia
  - Requires proper resource management
- Provide real-time guarantees using standard hardware
  - Build real-time systems using standard PC and network hardware

  ⇨ Make behavior predictable

DROPS - Architecture



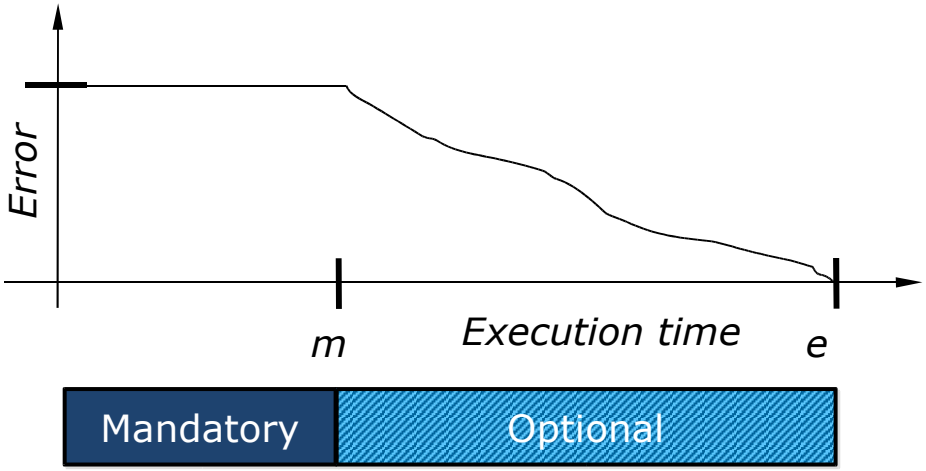
DROPS – Real-Time Application Model



- Applications are constructed from several real-time components
  - Application sets up and controls chain of components
  - Components process data streams
  - Data transfer between components e.g. using DSI

Imprecise Computations

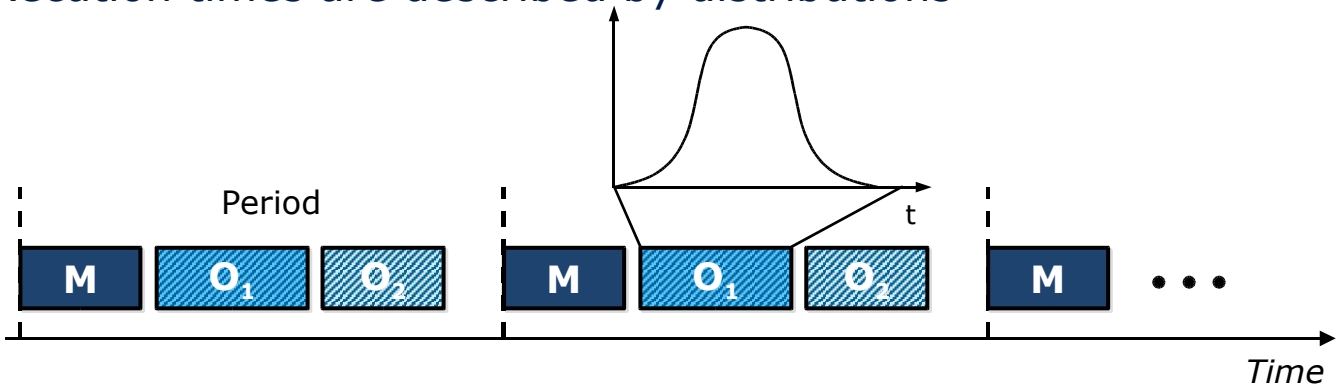
- Idea:
  - Split application in mandatory and optional part
  - Mandatory part computes necessary result
  - Optional part improves the result



- Example: Radar target tracking
  - Results of mandatory part exact enough to be able to follow target
  - Optional part improves accuracy of coordinates

DROPS: Quality-Assuring Scheduling

- Combines several ideas
  - Reservation-based scheduling
  - Splitting of applications into several parts (imprecise scheduling)
  - Probabilistic guarantees of deadlines (stochastic rate monotonic scheduling)
  - ⇨ Guarantee that a requested percentage of the optional parts reach their deadline
- Application Model
  - Periodic
  - Split into one mandatory and at least one optional part
  - Execution times are described by distributions



Nizza – Security Principles

- Minimal Trusted Computing Base (TCB) per application / service
  - Small security kernel – microkernel
  - Small set of small compontents (servers, ...)
  - Well-defined interfaces
  - Application-specific selection of platform components
- Split applications / services
  - Sensitive part on trusted platform
  - Less-sensitive (convenient) part of legacy OS

Nizza – Security Objectives

Confidentiality No unauthorized access to information

Integrity No unauthorized, unnoticed modification of information

Recoverability No permanent damage of information

Availability Timeliness of service

Nizza – System Security Objectives

- Secure and unsecure applications (trusted vs. untrusted)
  - Secure / trusted booting
  - Trusted path from/to user – Secure Graphical User Interface
    - Protection against Trojan Horses
  - Storage of sensitive information
    - Cryptographic keys, personal data
- Compatibility
  - Legacy applications / Operation systems
  - Standard hardware plus up-to-date enhancements (e.g. TPM)
  - User-friendliness

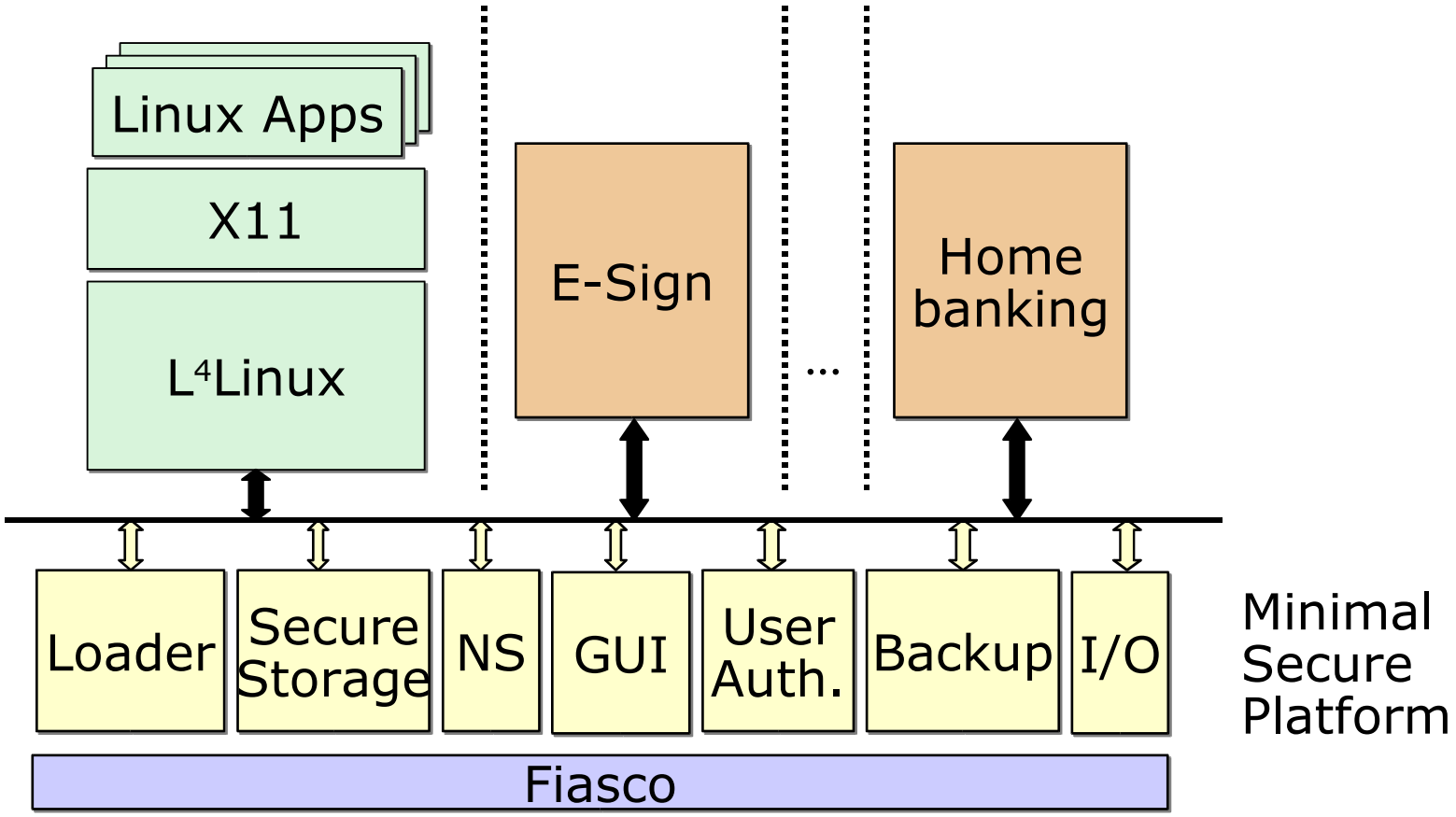
Nizza - Features

- Fine-grained isolation between applications
- Minimal TCB for trusted applications / services

  Reuse of untrusted components via **Trusted Wrappers**

  - Sandboxing
  - Perimeter Wrapping
- Support for trusted computing hardware
- Open Source alternative to Microsoft NGSCB

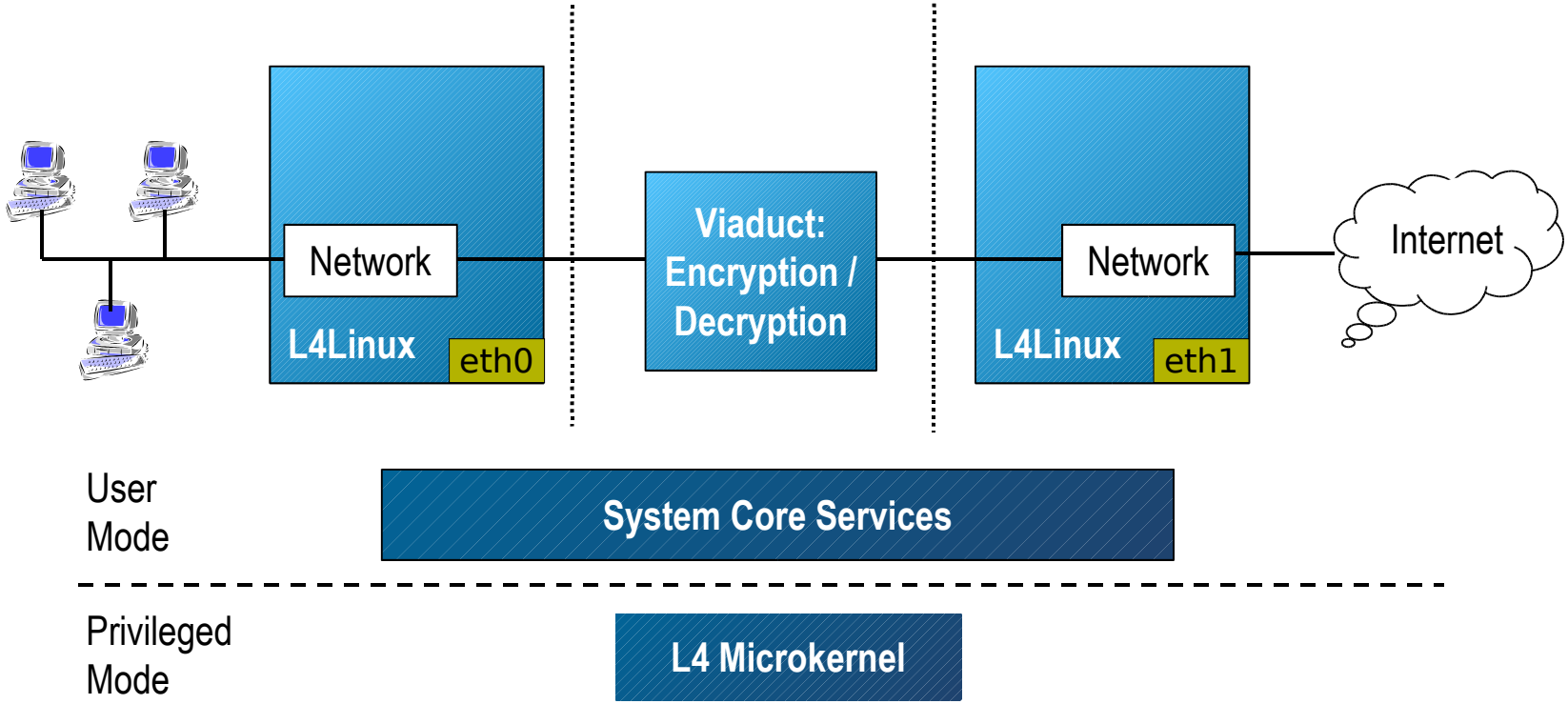
The NIZZA Security Architecture



µsina - Secure Microkernel-based System Architecture

- Build an IPsec VPN gateway with microkernel technology
- Reduce complexity of underlying platform (TCB)
- Run security sensitive components separately
  - (Re-)Use other software for untrusted parts
- Viaduct: IPsec component for en-/decryption
- Encrypted and unencrypted traffic are handled by different L^4Linux instances

µsina - Secure Microkernel-based System Architecture



Future

- Embedded systems
- Virtualization
- Advanced kernel features
  - communication control
  - kernel memory management

L4 Related Projects

- Mungi
- DD/OS
- NomadBIOS
- L4Hurd

Q/A?

http://os.inf.tu-dresden.de/
http://os.inf.tu-dresden.de/fiasco/
http://os.inf.tu-dresden.de/fiasco/ux/
http://os.inf.tu-dresden.de/L4/
http://os.inf.tu-dresden.de/drops/
http://os.inf.tu-dresden.de/L4/LinuxOnL4/

http://l4linux.org/

http://l4ka.org/