

Hints for L4 Microkernel

Jim Huang (黃敬群) <jserv@0xlab.org>

May 30, 2013 / NCTU, Taiwan

Rights to copy

© Copyright 2013 **0xlab**

<http://0xlab.org/>



Corrections, suggestions, contributions and translations
are welcome!

Attribution – ShareAlike 3.0

You are free

- to copy, distribute, display, and perform the work
- to make derivative works
- to make commercial use of the work

Latest update: May 16, 2013

Under the following conditions

- **BY:** **Attribution.** You must give the original author credit.
- **Share Alike.** If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.
- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

License text: <http://creativecommons.org/licenses/by-sa/3.0/legalcode>



Goals of This Presentation

- Introduce L4 microkernel principles and concepts
 - from “Microkernel Construction”
<http://os.inf.tu-dresden.de/Studium/MkK/>
 - From "Microkernel-based Operating Systems"
http://www.inf.tu-dresden.de/index.php?node_id=1314
- Understand the real world usage for microkernels
- Explore the modern implementation of modern L4 microkernel



Agenda

- Background Knowledge
- Microkernel-based Operating Systems
- Case Study



Background Knowledge



Microkernel Concepts

- Minimal kernel and hardware enforce separation
- Only kernel runs in CPU privileged mode
- Components are user!level processes
- No restrictions on component software
- Reuse of legacy software



principle of least privilege (POLA)

POSIX

operations allowed
by default

some limited
restrictions apply

ambient authority

POLA

nothing allowed by
default

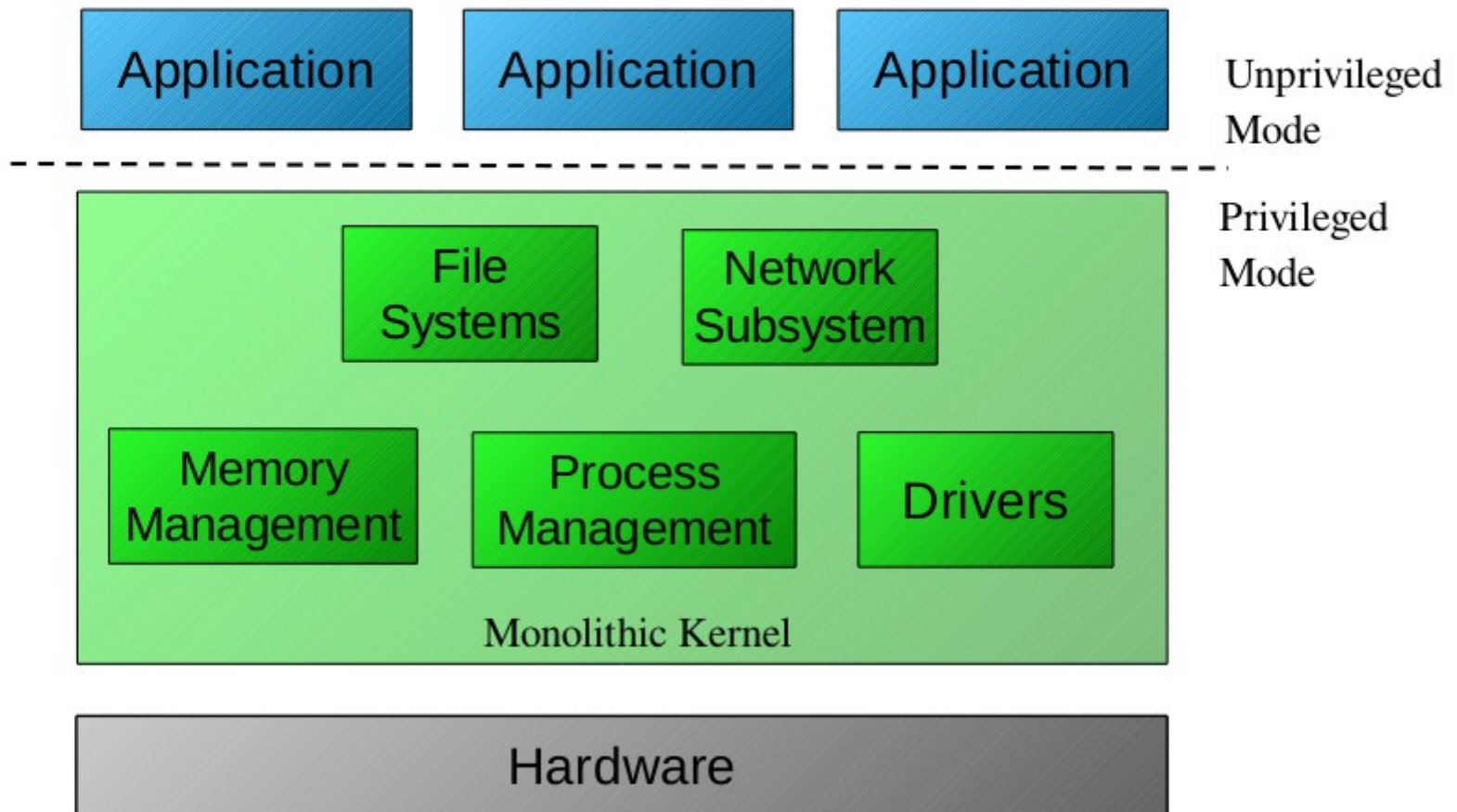
every right must
be granted

explicit authority

A capability is a communicable, unforgeable token of authority. It refers to a value that references an object along with an associated set of access rights. A user program on a capability-based operating system must use a capability to access an object.



„Monolithic“ Kernel System Design



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?

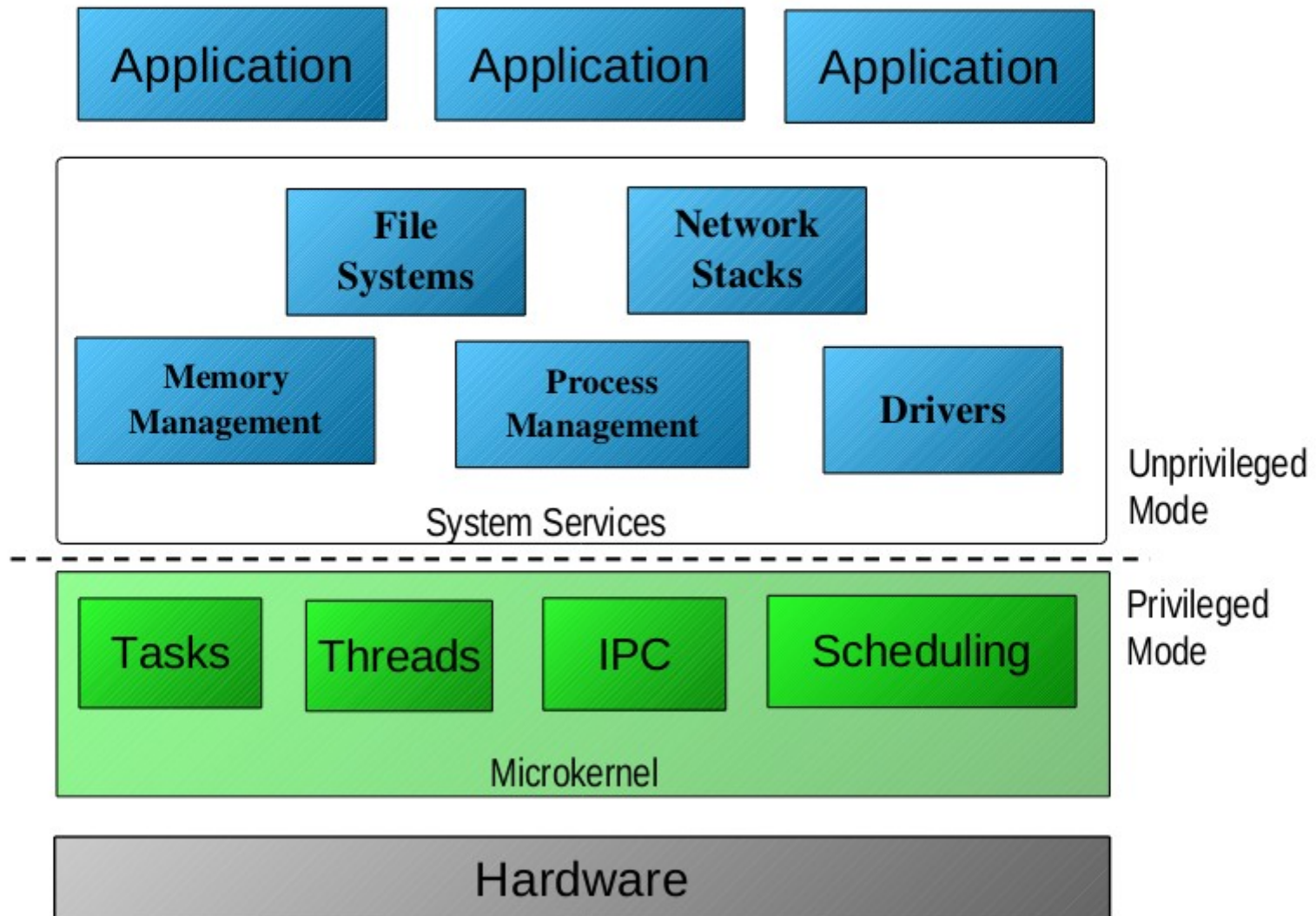
- **More and more difficult to manage increasing OS complexity**

Case Study: Bugs inside big kernels

- Drivers cause 85% of Windows XP crashes.
 - Michael M. Swift, Brian N. Bershad, Henry M. Levy: “Improving the Reliability of Commodity Operating Systems”, SOSP 2003
- Error rate in Linux drivers is 3x (maximum: 10x)
 - Andy Chou, Junfeng Yang, Benjamin Chelf, Seth Hallem, Dawson R. Engler: “An Empirical Study of Operating System Errors”, SOSP 2001
- Causes for driver bugs
 - 23% programming error
 - 38% mismatch regarding device specification
 - 39% OS-driver-interface misconceptions
 - Leonid Ryzhyk, Peter Chubb, Ihor Kuz and Gernot Heiser: “Dingo: Taming device drivers”, EuroSys 2009

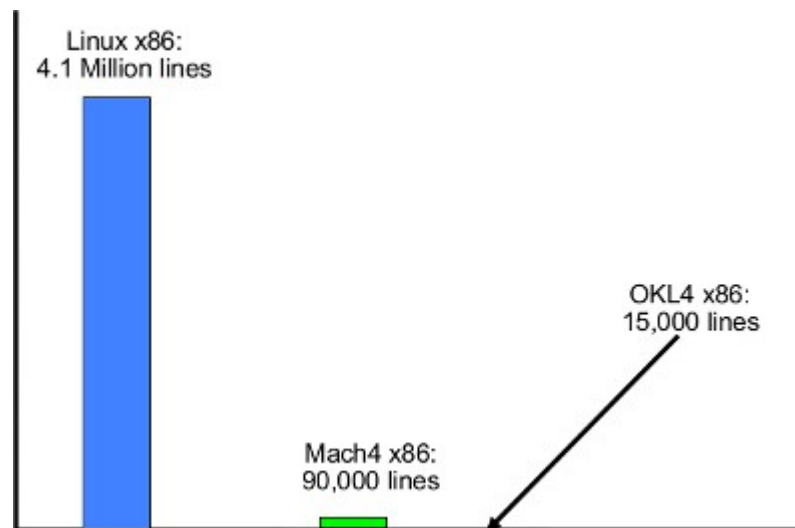


Microkernel System Design

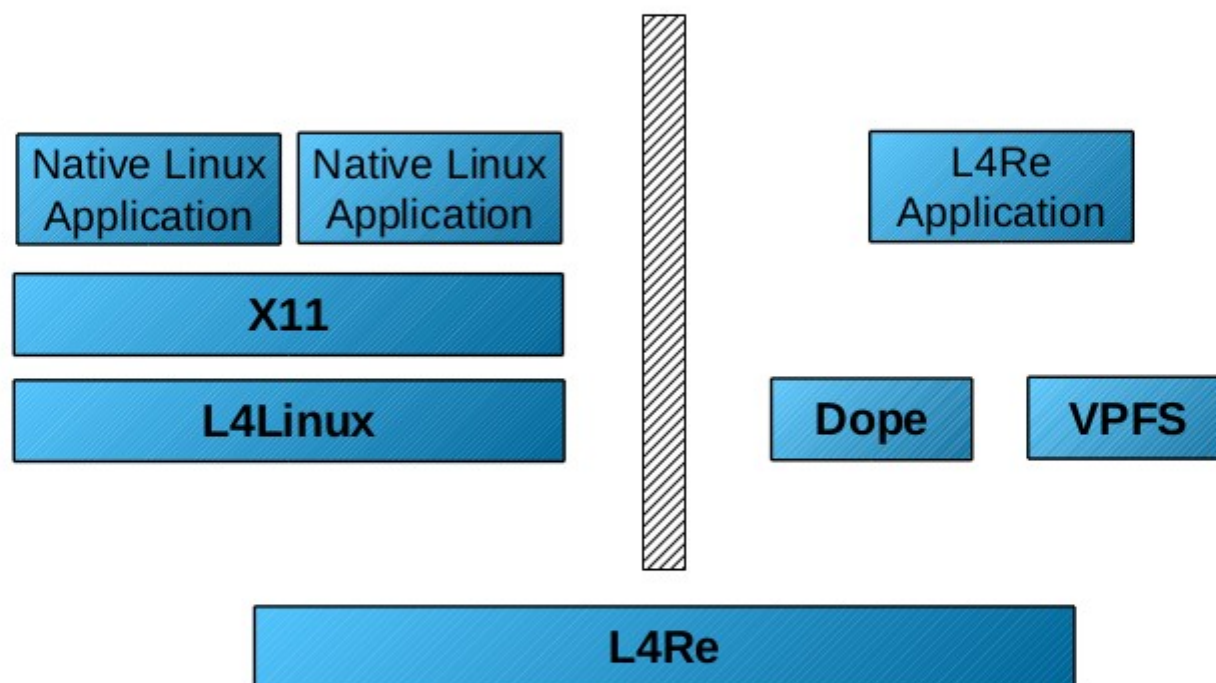


Microkernel

- Minimalist approach
 - IPC, virtual memory, thread scheduling
- Put the rest into user space
 - Device drivers, networking, file system, user interface
- Disadvantages
 - Lots of system calls and context switches
- Examples: Mach, L4, QNX, MINIX, IBM K42

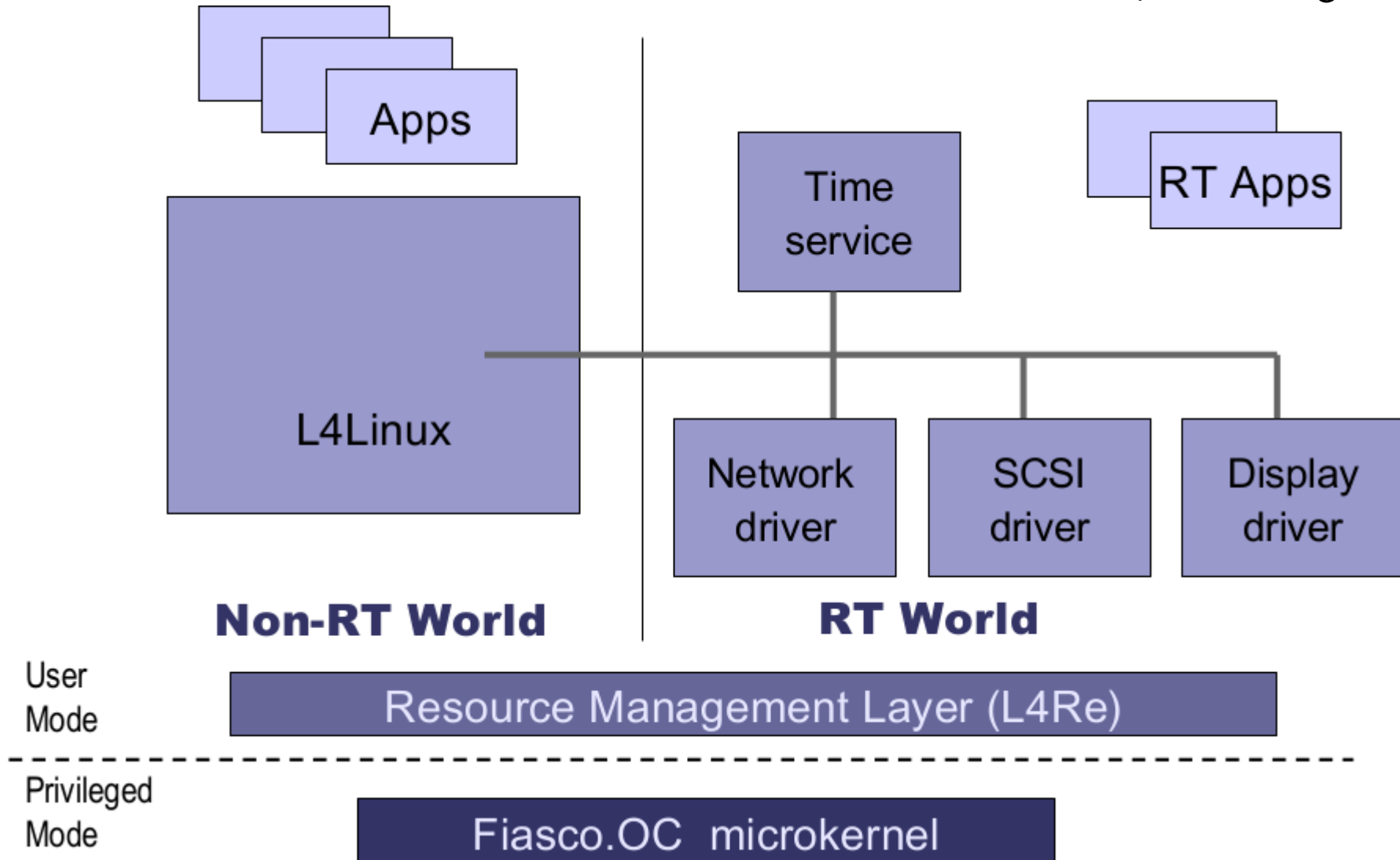


Example – Fiasco.OC



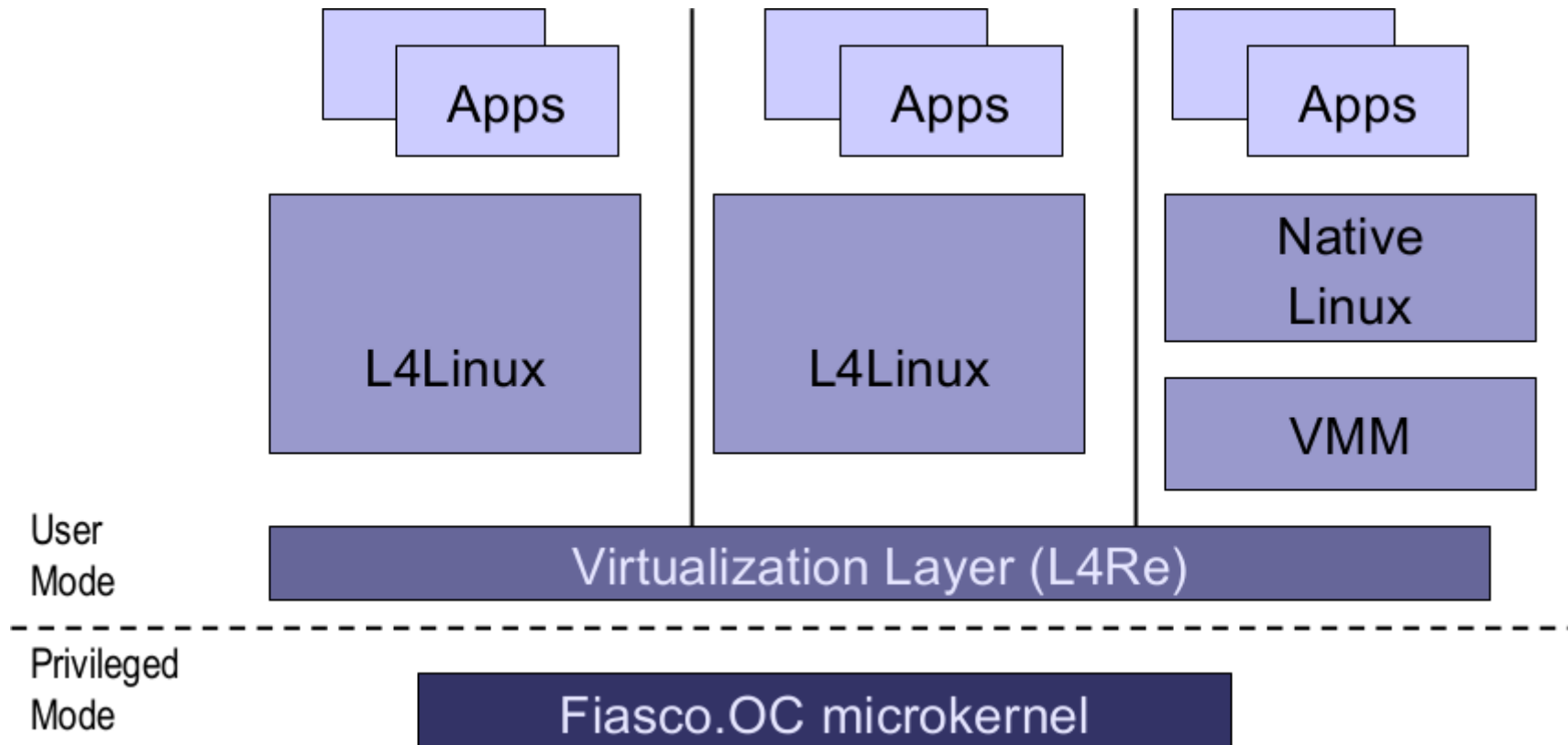
DROPS

(Dresden Real-Time Operating System)



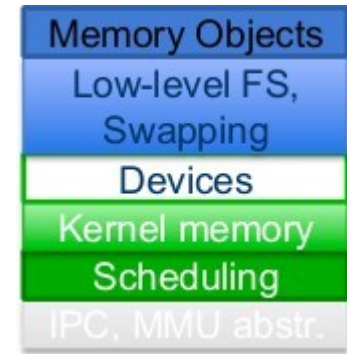
Virtualization based on L4Re

- Isolate not only processes, but also complete
- Operating Systems (compartments)
 - “Server consolidation”



3 Generations of Microkernel

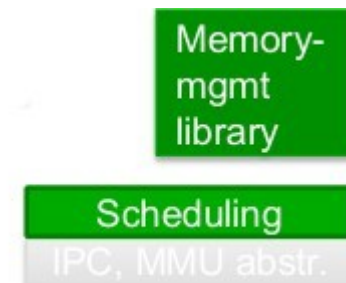
- Generation 1: Mach (1985-1994)



- Generation 2: L3 & L4 (1990-2001)



- Generation 3: seL4, Fiasco.OC, NOVA (2000-)



3 Generations of Microkernel

- Mach (1985-1994)
 - replace pipes with IPC (more general)
 - improved stability (vs monolithic kernels)
 - poor performance
- L3 & L4 (1990-2001)
 - order of magnitude improvement in IPC performance
 - written in assembly, sacrificed CPU portability
 - only synchronus IPC (build async on top of sync)
 - very small kernel: more functions moved to userspace
- seL4, Fiasco.OC, Coyotos, NOVA (2000-)
 - platform independence
 - verification, security, multiple CPUs, etc.




History about L4 Microkernels⁽¹⁾

- Eumel, L3, BirliX
- 1995 • first version of L4
- 1997 • L4Linux, the first major application
- 1998 • Fiasco, the first HLL implementation
- PikeOS: first commercial derivative
- real-time systems based on Fiasco
- Pistachio (Uni Karlsruhe)

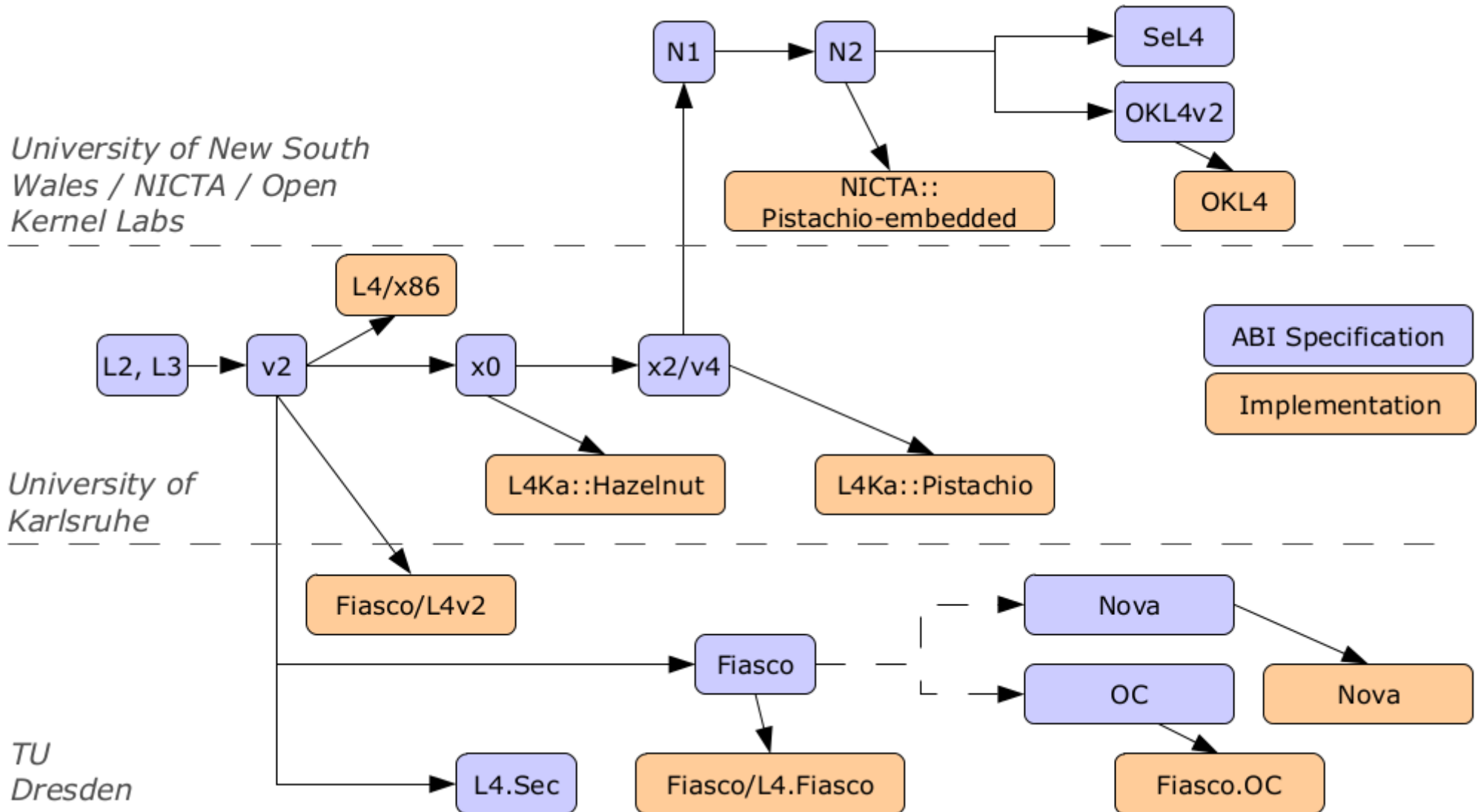


History about L4 Microkernels⁽²⁾

- 2000 • First commercial usage (Fiasco on a DRM product)
 - 2005 • Qualcomm adopts L4 kernel from NICTA (Pistachio derivative)
 - 2009 • Full formal verification of implementation in Haskell (NICTA)
 - 2009 • Fiasco.OC, L4Re
 - 2009 • NOVA
- 



L4 Family (incomplete)



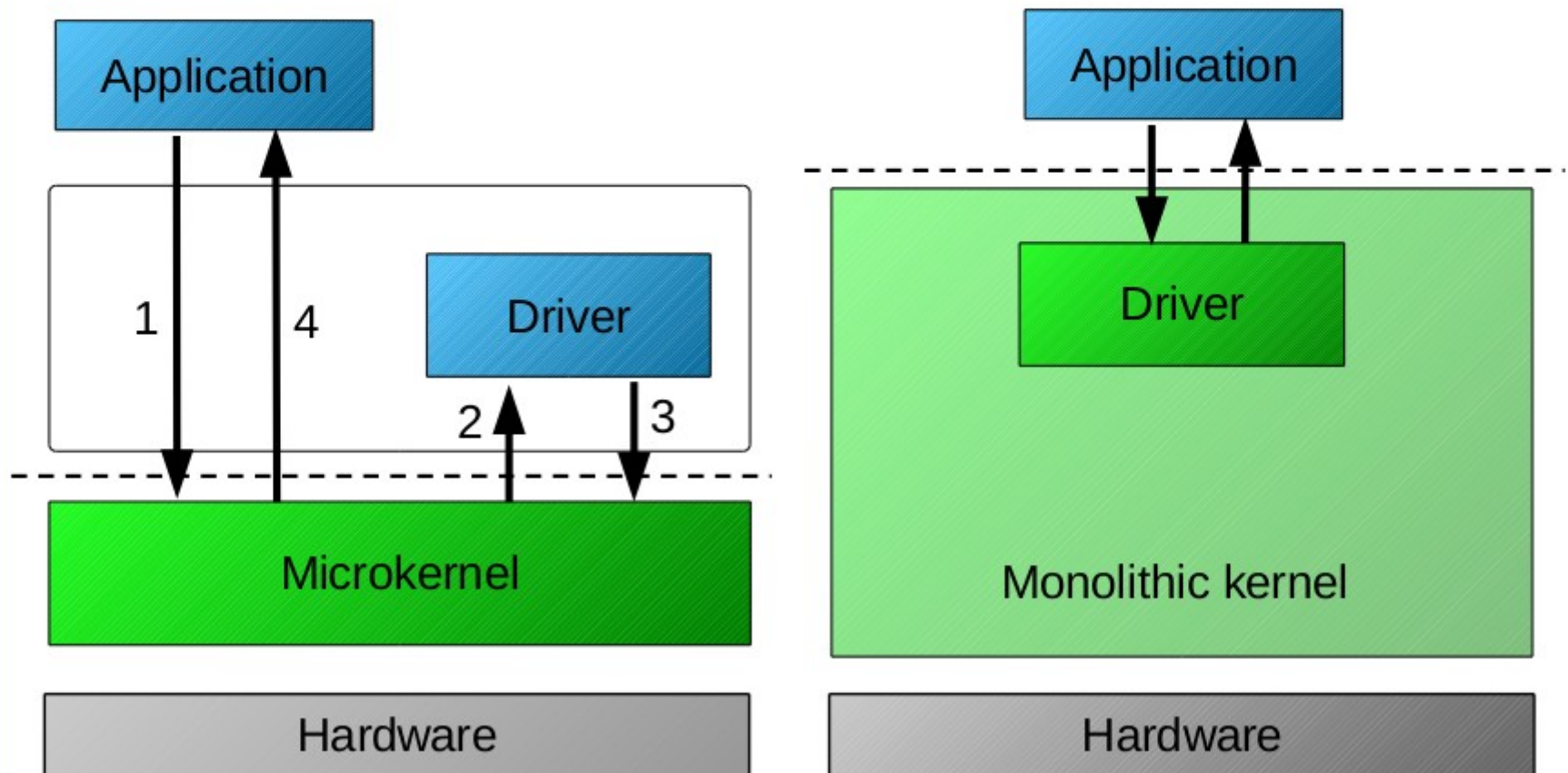
L4 API vs. ABI

- L4 project was originally established by Jochen Liedtke in the 1990s and is actively researched by the L4Ka team at the University of Karlsruhe in collaboration with NICTA / University of New South Wales and the Dresden University of Technology.
- L4 is defined by a platform-independent¹ API and a platform-dependent ABI



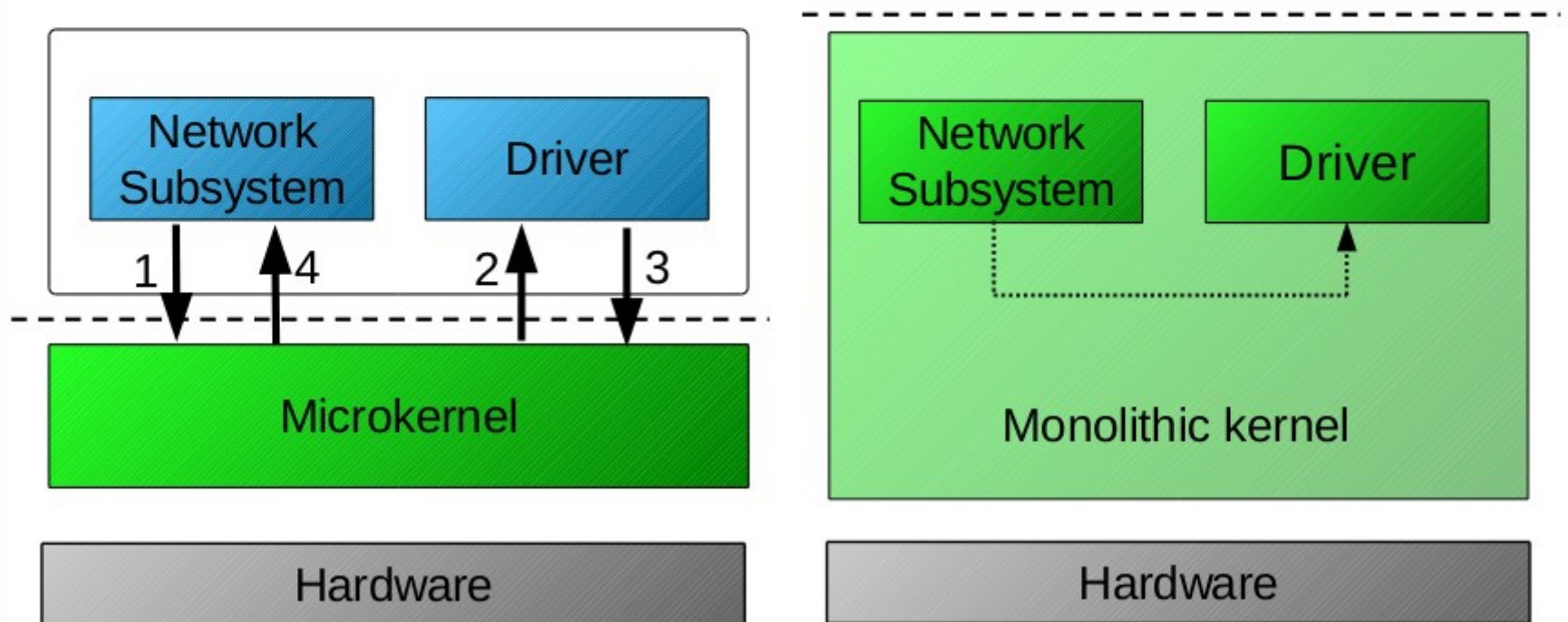
Robustness vs. Performance (1)

- 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



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)*

Microkernel-based Operating Systems

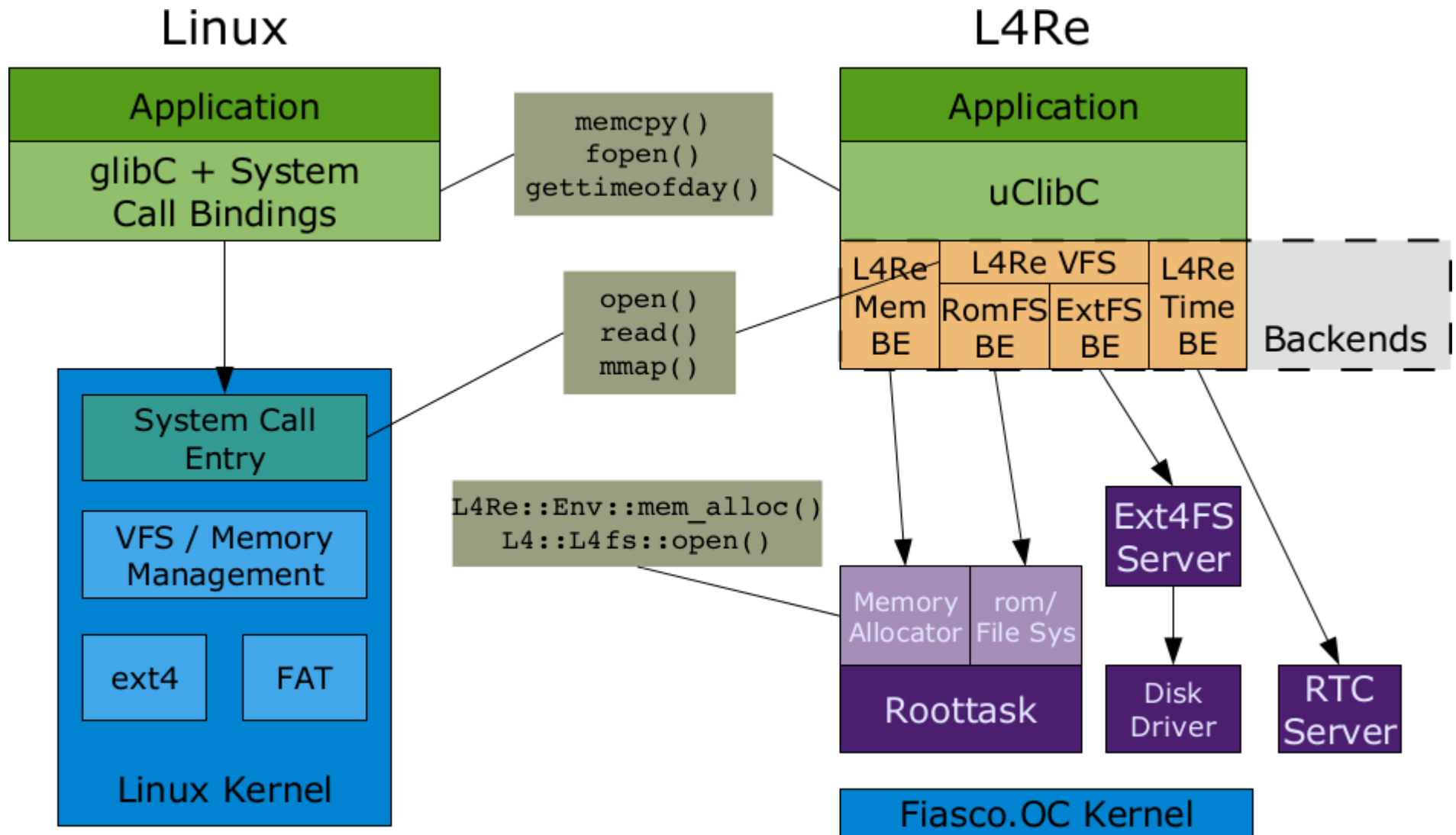


Microkernel-based Operating Systems - Introduction

- Information:
 - Carsten Weinhold
 - Dresden, Oct 09th 2012
- Filename: 01-Introduction.pdf
 - Page 22 – Page 46



POSIX on L4Re



L4Re Backend example: time()

```
time_t time(time_t *t)
{
    struct timespec a;

    libc_be_rt_clock_gettime(&a);

    if (t)
        *t = a.tv_sec;

    return a.tv_sec;
}
```

Replacement of POSIX'
time() function

```
uint64_t __libc_l4_rt_clock_offset;

int libc_be_rt_clock_gettime(struct timespec *t)
{
    uint64_t clock;

    clock = l4re_kip()->clock();
    clock += __libc_l4_rt_clock_offset;

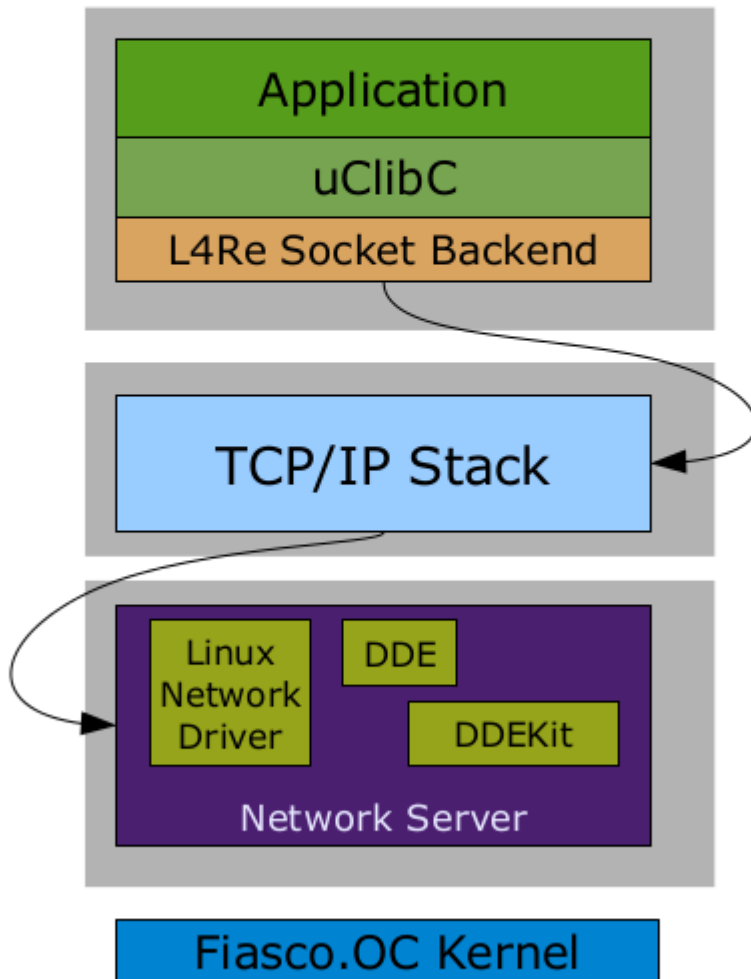
    t->tv_sec  = clock / 1000000;
    t->tv_nsec = (clock % 1000000) * 1000;

    return 0;
}
```

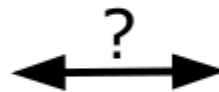
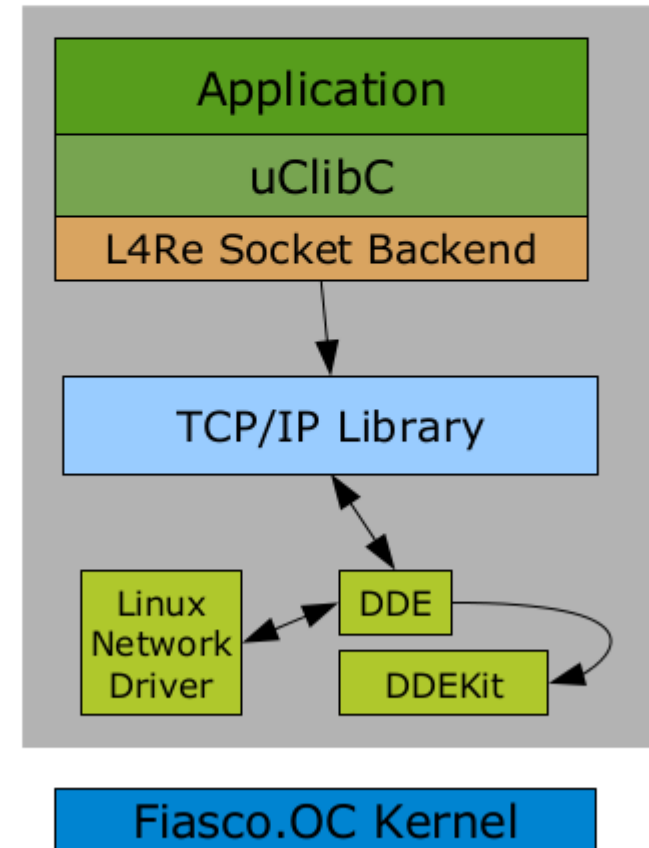
Call L4Re-specific
backend function



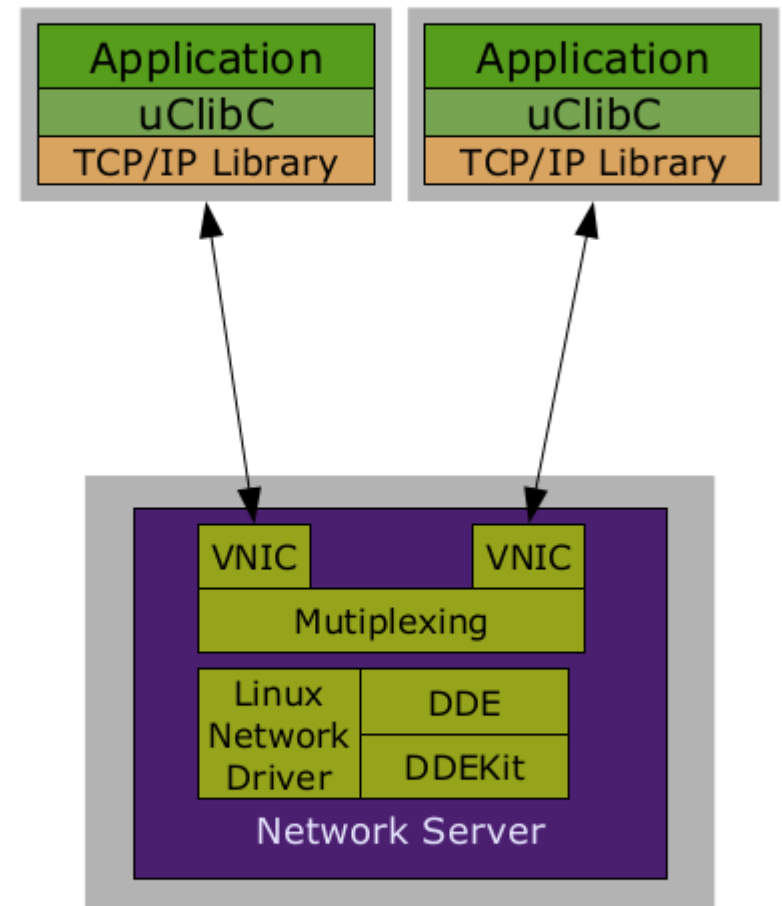
Full Protection: Isolated Address Spaces



Best performance: all in one address space



- Problem: Devices often don't support shared access from multiple applications
- Solution: Introduce “virtualized” intermediary interfaces
- Networking on L4Re: Ankh network multiplexer
 - Shared memory NIC for each client
 - Virtual MAC addresses



Case Study: Security



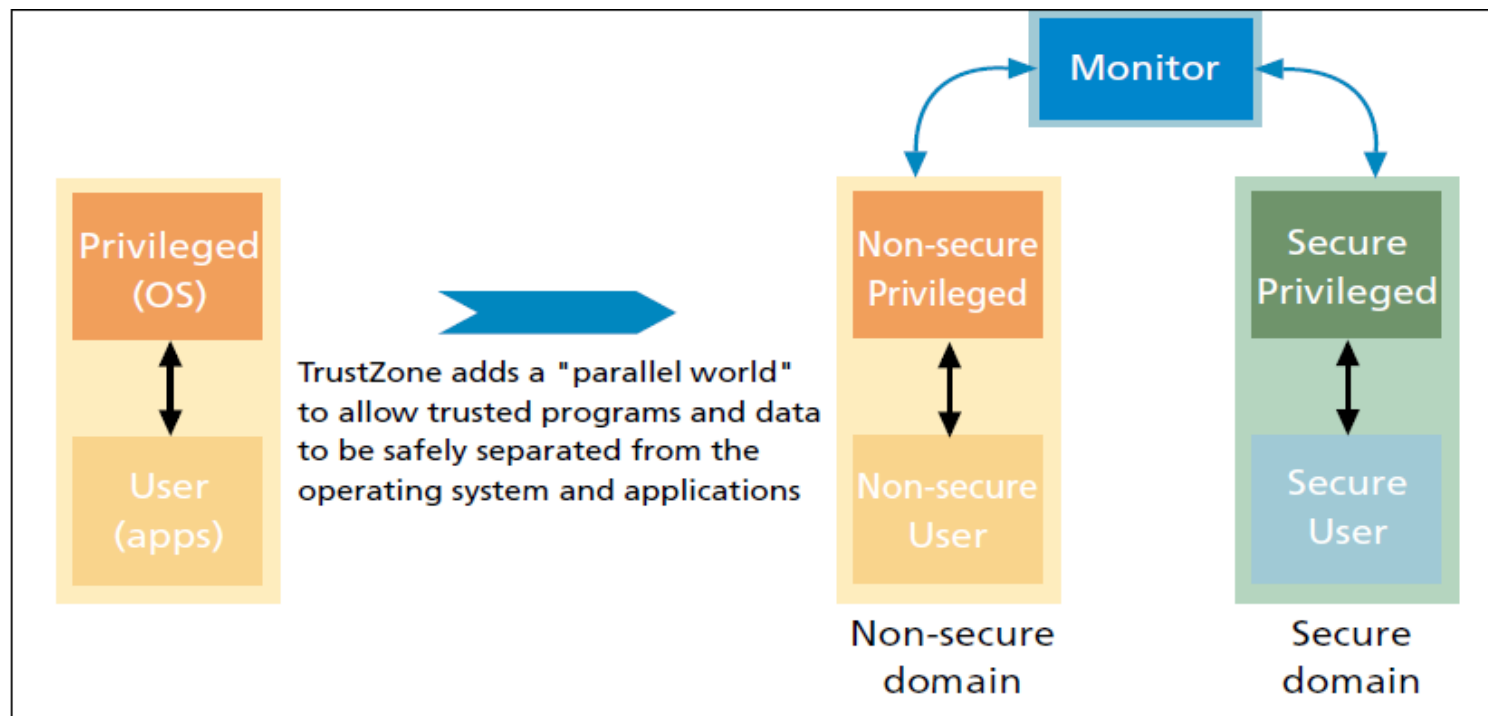
Apply Fiasco.OC for ARM

- Mission: To implement a software stack that allows the emulation of the mobile operating system to use the functionality of hardware protection mechanisms
- Essential components
 - ARM TrustZone extension
 - Fiasco.OC microkernel
 - Android



ARM TrustZone

- ARM's processor extension that allows for a software TPM implementation
- Available on all major ARM mobile phone SoC
- There is limited open source development with TrustZone



ARM TrustZone: Application Examples

- Secure PIN Entry
- Digital Rights Management
- e-Ticketing Mobile TV (Netflix)



Trust Platform Module (TPM)

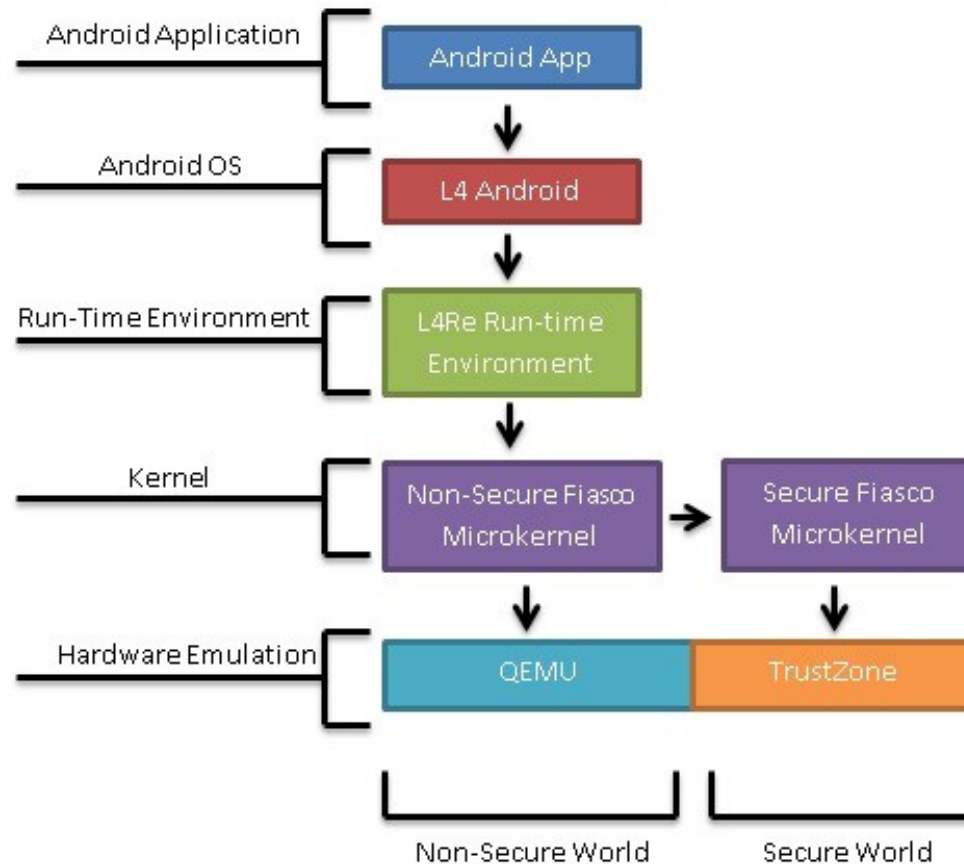
- A TPM is a chip that resides on the motherboard, and provides 4 basic functionalities
 - Secure storage and reporting of platform configurations
 - Protected private key storage
 - Cryptographic functions
 - Initialization and management functions



System Overview

The highest part of the stack will be a program we write that uses TrustZone's TPM features

Application will make TrustZone calls to the microkernel



TPM Design Goals

- use the secure world to provide two TPM services:
 - Random Number Generation
 - RSA Key Generation
- An Android application will be able to use the TPM services provided and will be able to perform the following tasks:
 - encrypt sensitive data using the secure world
 - decrypt sensitive data using the secure world



Reference

- From L3 to seL4: What Have We Learnt in 20 Years of L4 Microkernels? Kevin Elphinstone and Gernot Heiser, NICTA/UNSW
- Microkernel Construction"
<http://os.inf.tu-dresden.de/Studium/MkK/>
- Microkernel-based Operating Systems
http://www.inf.tu-dresden.de/index.php?node_id=1314
- The L4 Microkernel, Hermann Härtig, Technische Universität Dresden
- Microkernels, Arun Krishnamurthy, University of Central Florida





<http://0xlab.org>