Faculty of Computer Science Institute for System Architecture, Operating Systems Group

Advanced Components on Top of A Microkernel

Björn Döbel

What we talked about so far

- Microkernels are cool!
- Fiasco.OC provides fundamental mechanisms:
  - Tasks (address spaces)
    - Container of resources
  - Threads
    - Units of execution
  - Inter-Process Communication
    - Exchange Data
    - Timeouts
    - Mapping of resources

Lecture Outline

- Building a real system on top of Fiasco.OC
- Reusing legacy libraries
  - POSIX C library
- Device Drivers in user space
  - Accessing hardware resources
  - Reusing Linux device drivers
- OS virtualization on top of L4Re

Reusing Existing Software

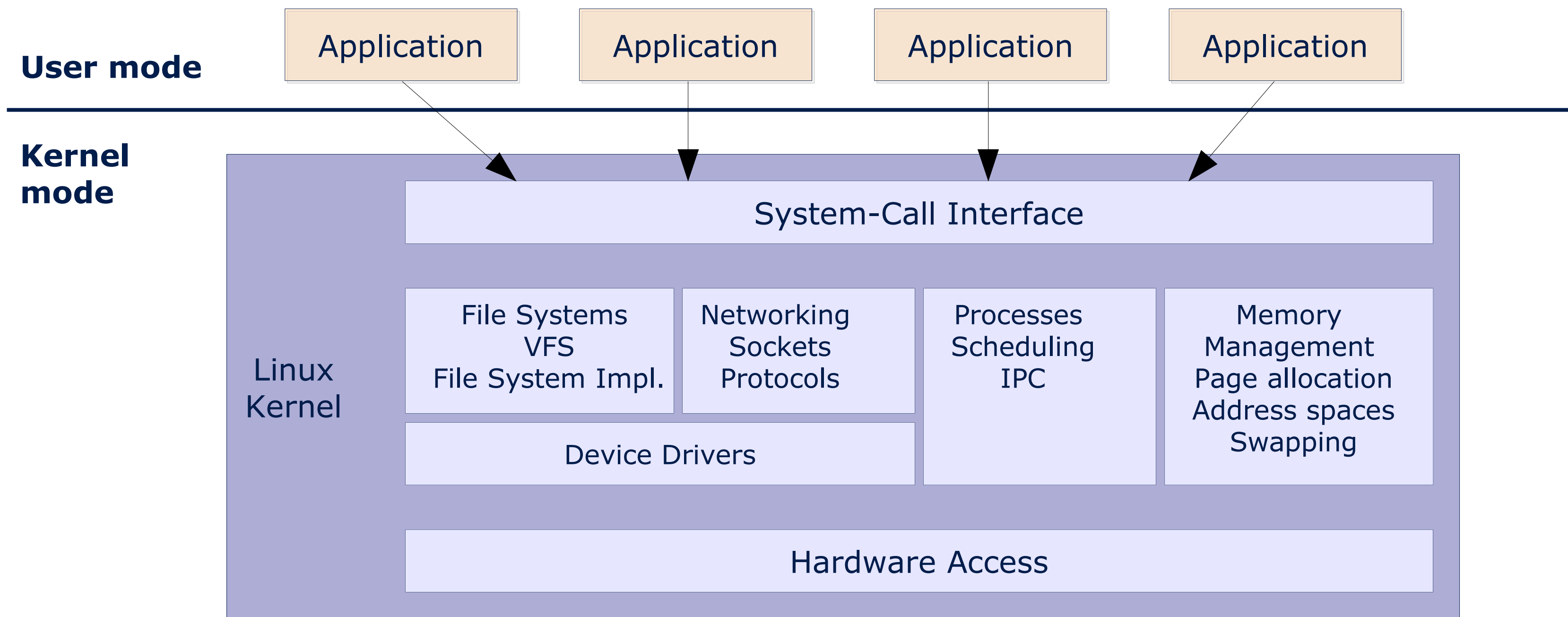
- Often used term: legacy software
- Why?
  - Convenience:
    - Users get their “favorite” application on the new OS
  - Effort:
    - Rewriting everything from scratch takes a lot of time
    - But: maintaining ported software and adaptions also does not come for free

TECHNISCHE UNIVERSITÄT DRESDEN

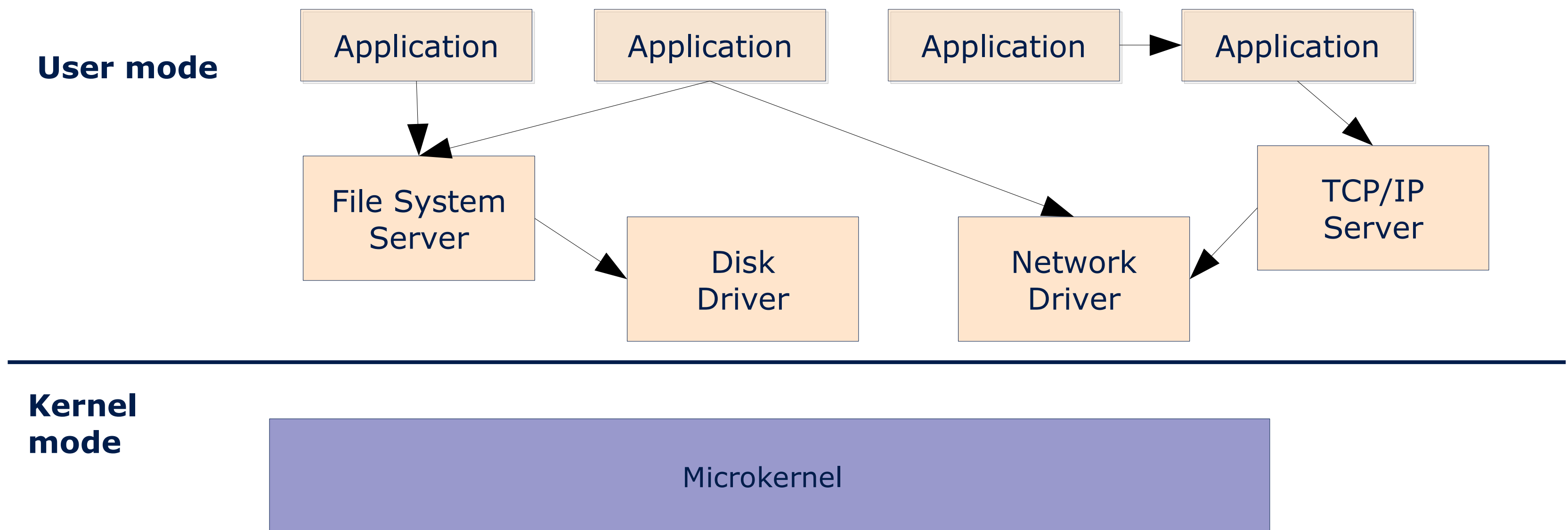
Reusing Existing Software

- How?
  - Porting:
    - Adapt existing software to use L4Re/Fiasco.OC features instead of Linux
    - Efficient execution, large maintenance effort
  - Library-level interception
    - Port convenience libraries to L4Re and link legacy applications without modification
      - POSIX C libraries, libstdc++
  - OS-level interception
    - Wine: implement Windows OS interface on top of new OS
  - Hardware-level:
    - Virtual Machines

Starting Point: Monolithic OS



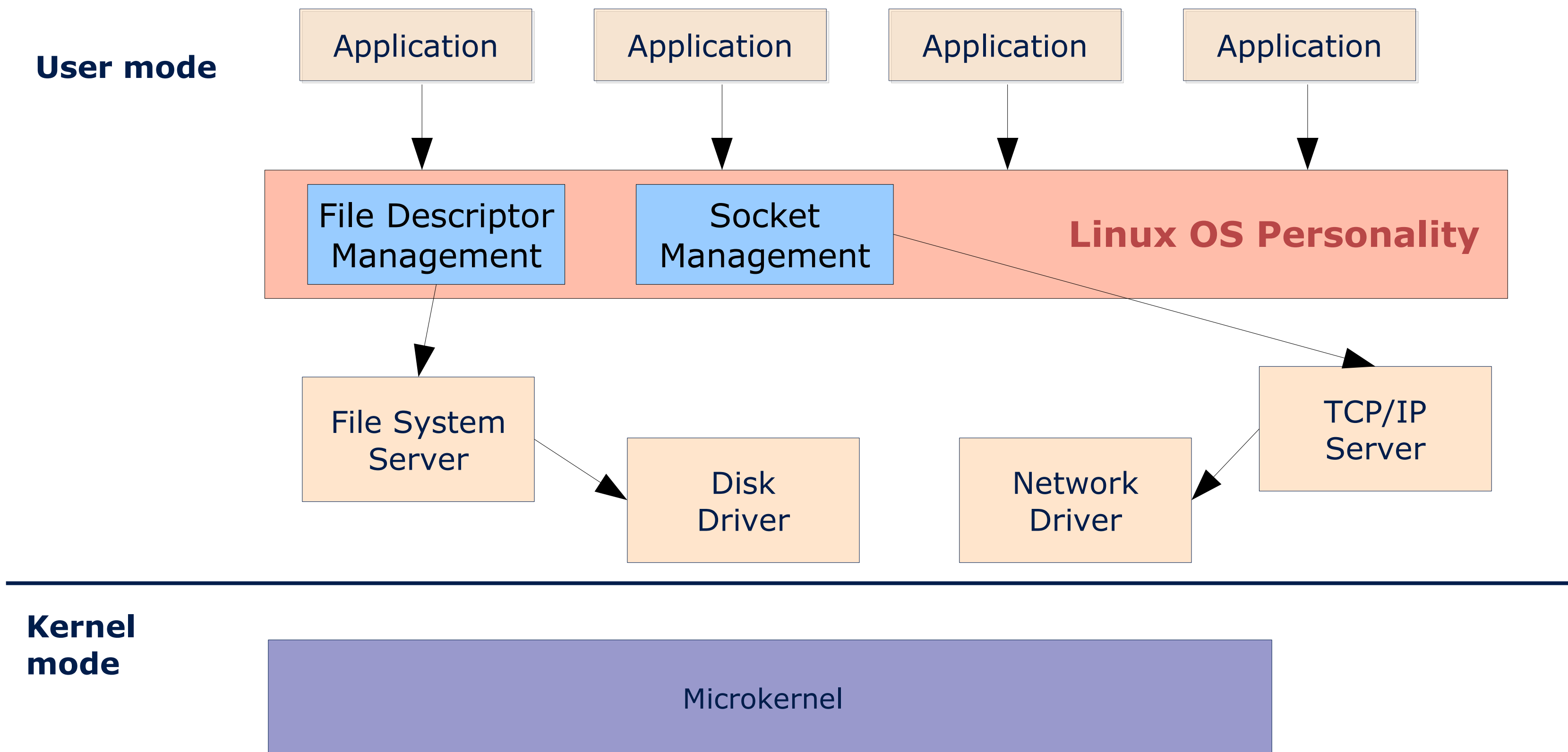
Microkernel: Decomposed System



COMPLEX!

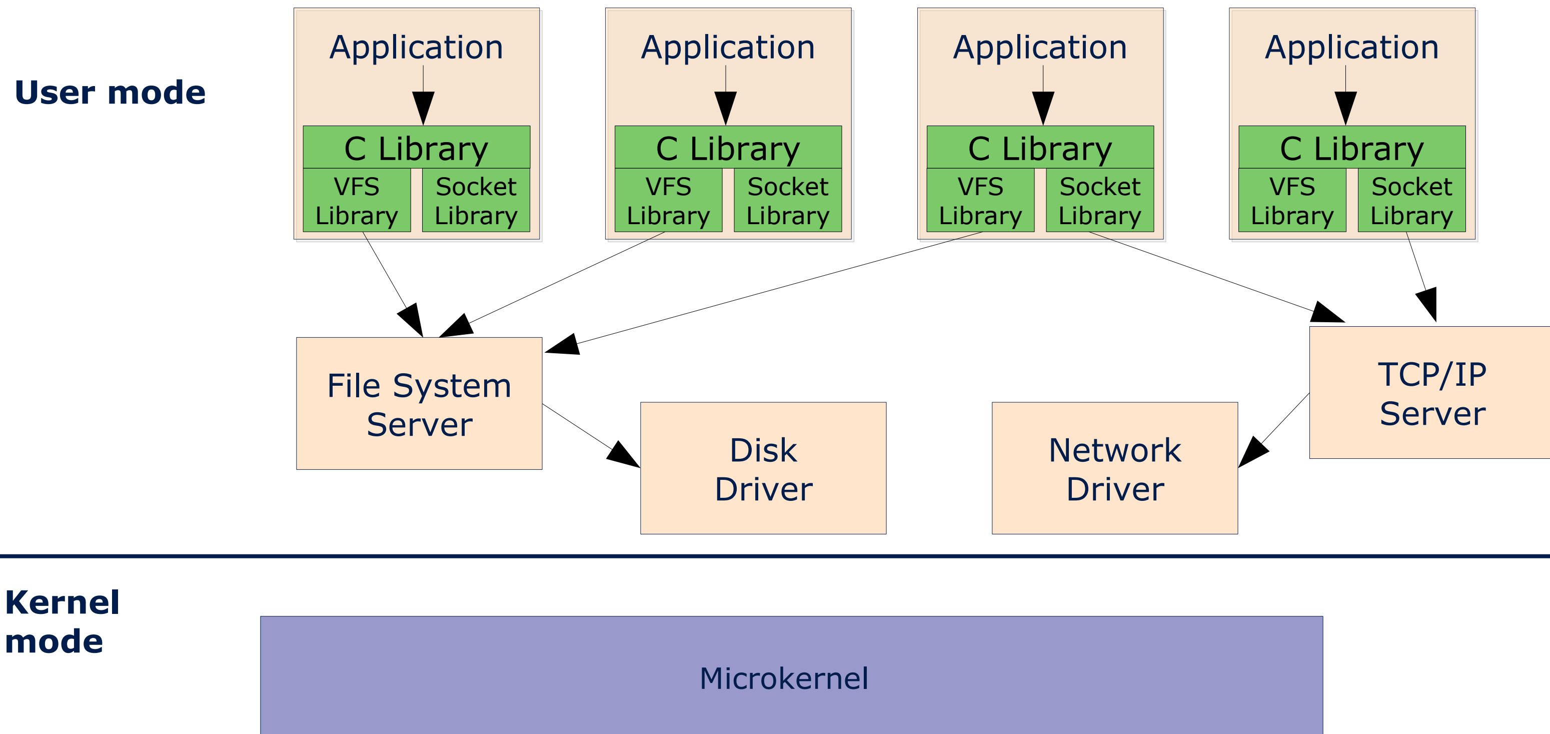
TECHNISCHE UNIVERSITÄT DRESDEN

Idea: Central Server



- Complexity only hidden in OS personality
- Personality becomes a bottleneck

L4Re: Emulation Libraries



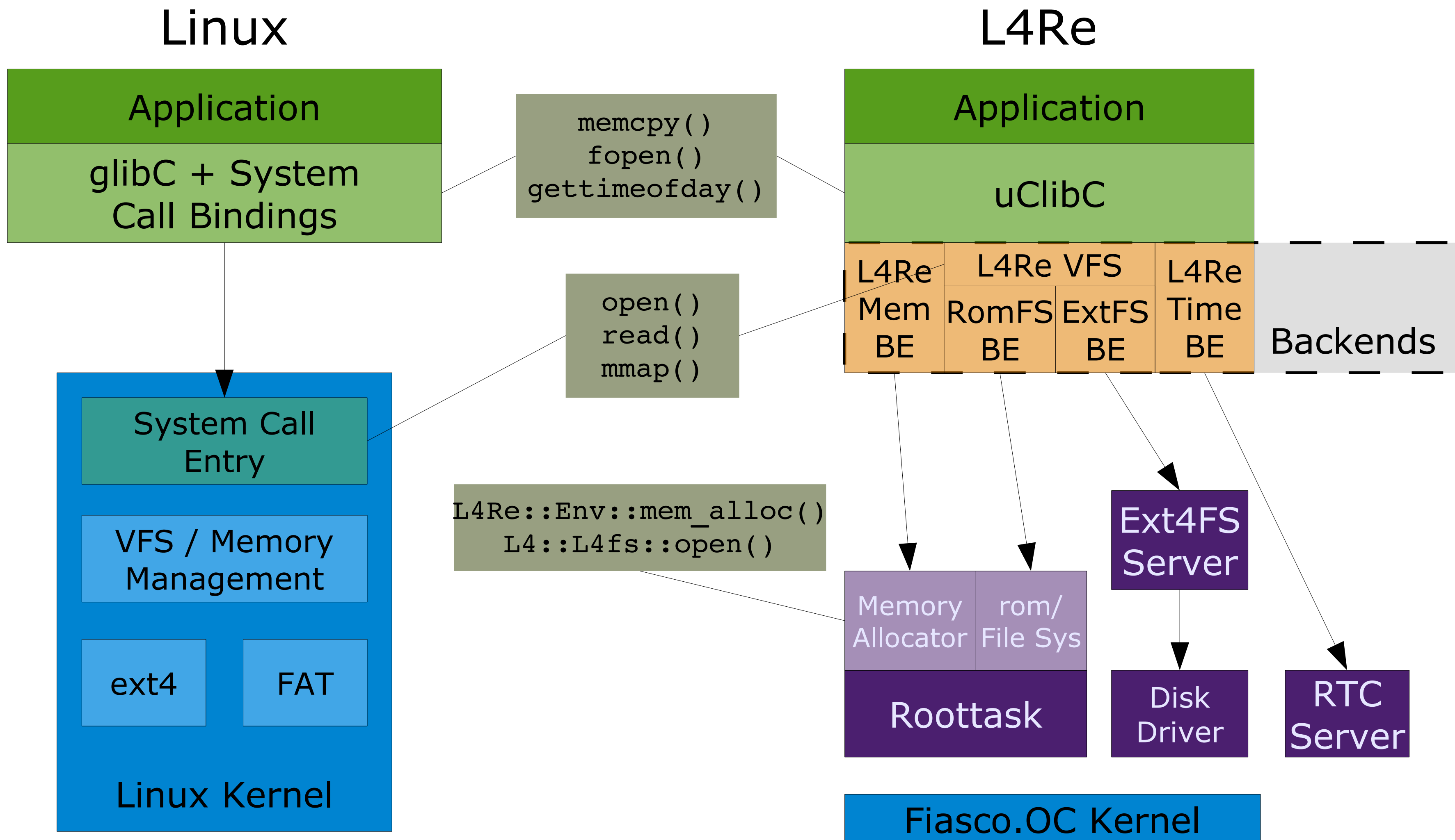
- Applications use common C interface (POSIX)
- Complexity split across “backend” libraries

POSIX

- Standard: Portable Operating System Interface (for UNIX*)
- Interfaces:
  - I/O: files, sockets, TTYs
  - Threads: libpthread
  - ...
- Usually provided by a C library
- Abstractions vs. dependencies:
  - `memcpy()`, `strcpy()` → no deps, simply reuse
  - `malloc()` → depends on mmap() / sbrk()
  - `getpwent()` → depends on file system, notion of users, Posix ACLs ...

TECHNISCHE UNIVERSITÄT DRESDEN

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

TECHNISCHE UNIVERSITÄT DRESDEN

L4Re Backend: mmap

- libC implements memory allocator
- Uses `mmap(... MAP_ANONYMOUS …)` to allocate backing memory
- Can reuse libC's allocator if we provide mmap()

- L4Re VFS library:
  - `mmap(MAP_PRIVATE|MAP_ANONYMOUS)`:
    → use dataspace as backing memory
    → attach DS via L4RM interface

Summary: Legacy Reuse

- L4Re provides C (and C++) standard library
- OS-specific functionality wrapped by backend libraries
  - Virtual file system
  - Memory allocation
  - Time
  - Signals
  - Sockets
- POSIX support nearly directly enables a wide range of other libraries:
  - libpng, freetype, libcairo, Qt, ...

Moving on: Device Drivers

- Now that I have a virtual file system, I'd actually like to access the disk …
  - … or the network
  - … or my USB webcam.
- Microkernel philosophy: run non-essential components in user space
- But: device drivers might need privileged hardware access.

Some statistics

- [Swift03]: Drivers cause 85% of Windows XP crashes.
- [Chou01]:
  - Error rate in Linux drivers is 3x (maximum: 10x) higher than for the rest of the kernel
  - Bugs cluster (if you find one bug, you're more likely to find another one pretty close)
  - Life expectancy of a bug in the Linux kernel (~2.4): 1.8 years
- [Rhyzyk09]: Causes for driver bugs
  - 23% programming error
  - 38% mismatch regarding device specification
  - 39% OS-driver-interface misconceptions

Anecdote: Linux e1000 NVRAM bug

- **Aug 8th 2008** Bug report: e1000 PCI-X network cards rendered broken by Linux 2.6.27-rc
  - overwritten NVRAM on card

Anecdote: Linux e1000 NVRAM bug

- **Aug 8th 2008** Bug report: e1000 PCI-X network cards rendered broken by Linux 2.6.27-rc
  - overwritten NVRAM on card
- **Oct 1st 2008** Intel releases quickfix
  - map NVRAM somewhere else

Anecdote: Linux e1000 NVRAM bug

- **Aug 8th 2008** Bug report: e1000 PCI-X network cards rendered broken by Linux 2.6.27-rc
  - overwritten NVRAM on card
- **Oct 1st 2008** Intel releases quickfix
  - map NVRAM somewhere else
- **Oct 15th 2008** Reason found:
  - dynamic ftrace framework tries to patch __init code, but .init sections are unmapped after running init code
  - NVRAM got mapped to same location
  - Scary cmpxchg() behavior on I/O memory

Anecdote: Linux e1000 NVRAM bug

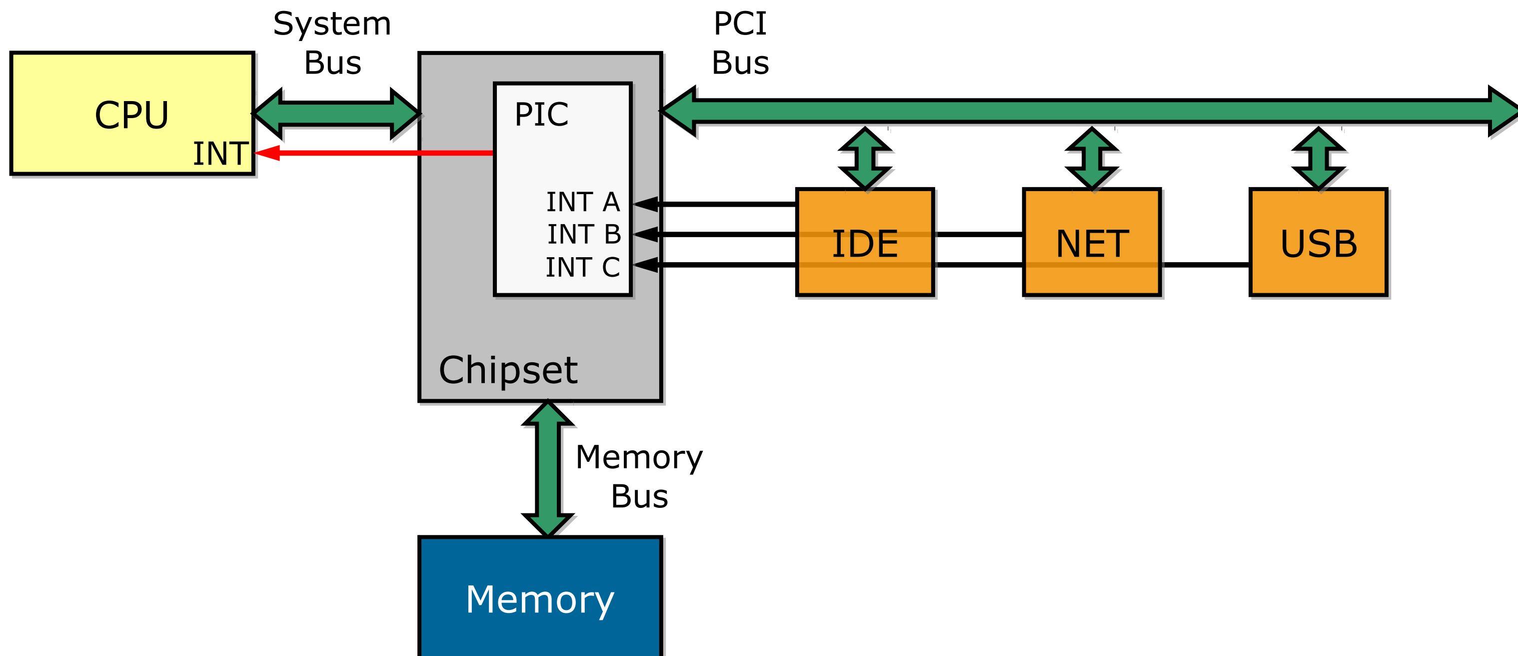
- **Aug 8th 2008** Bug report: e1000 PCI-X network cards rendered broken by Linux 2.6.27-rc
  - overwritten NVRAM on card
- **Oct 1st 2008** Intel releases quickfix
  - map NVRAM somewhere else
- **Oct 15th 2008** Reason found:
  - dynamic ftrace framework tries to patch __init code, but .init sections are unmapped after running init code
  - NVRAM got mapped to same location
  - Scary cmpxchg() behavior on I/O memory
- **Nov 2nd 2008** dynamic ftrace reworked for Linux 2.6.28-rc3

Idea: User-level Drivers

- Isolate components
  - device drivers (disk, network, graphic, USB cruise missiles, ...)
  - stacks (TCP/IP, file systems, …)
- Separate address spaces each
  - More robust components
- Problems
  - Overhead
    - HW multiplexing
    - Context switches
  - Need to handle I/O privileges

Hardware Resources: Interrupts

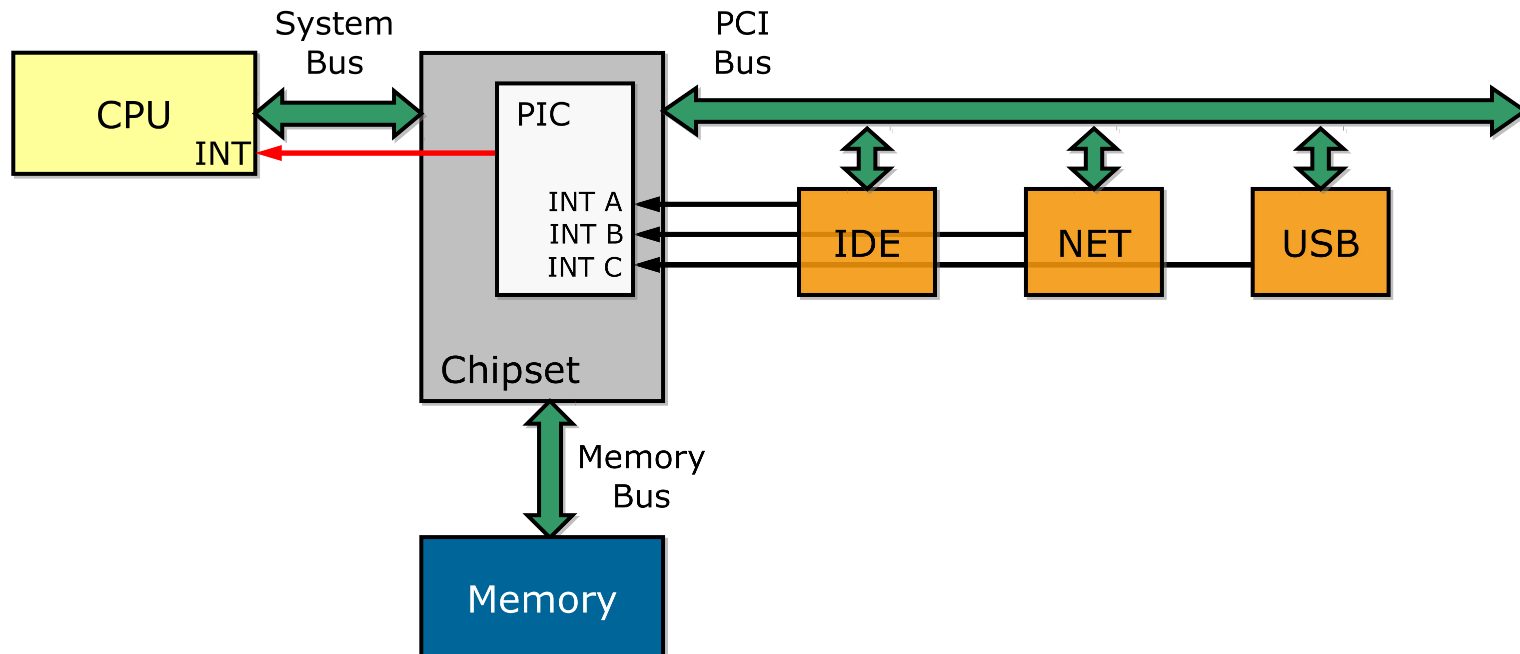
- Signal device state change
- Programmable Interrupt Controller (PIC, APIC)
  - map HW IRQs to CPU's IRQ lines
  - prioritize interrupts



TECHNISCHE UNIVERSITÄT DRESDEN

Hardware Resources: Interrupts (2)

- Handling interrupts involves
  - examine / manipulate device
  - program PIC
    - acknowledge/mask/unmask interrupts



L4: Interrupt handling

- IRQ kernel object
  - Represents arbitrary async notification
  - Kernel maps hardware IRQs to IRQ objects
- Exactly one waiter per object
  - call `l4_irq_attach()` before
  - wait using `l4_irq_receive()`
- Multiple IRQs per waiter
  - attach to multiple objects
  - use `l4_ipc_wait()`
- IRQ sharing
  - Many IRQ objects may be `chain()`ed to a master IRQ object

TECHNISCHE UNIVERSITÄT DRESDEN

Hardware Resources: I/O ports

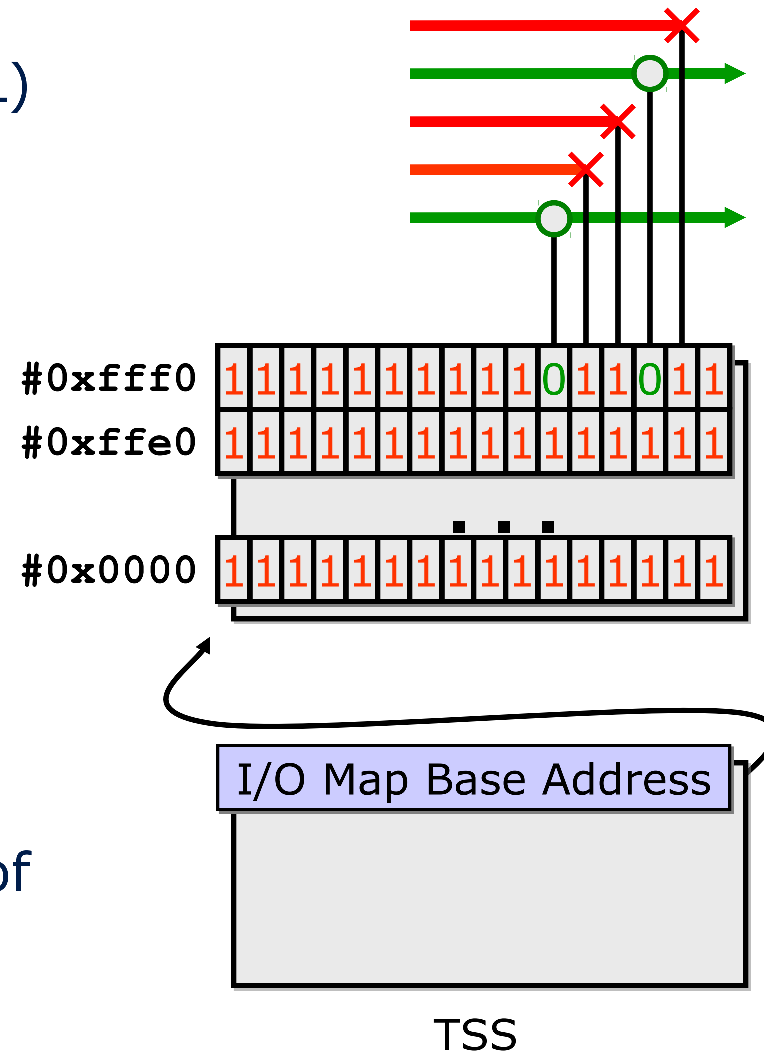
- x86-specific feature
- I/O ports define own I/O address space
  - Each device uses its own area within this address space
- Special instruction to access I/O ports
  - in / out: I/O read / write
  - Example: read byte from serial port

```
mov $0x3f8, %edx
in  (%dx), %al
```

- Need to restrict I/O port access
  - Allow device drivers access to I/O ports used by its device only

I/O Bitmap

- Per task IO privilege level (IOPL)
- If IOPL > current PL, all accesses are allowed (kernel mode)
- Else: I/O bitmap is checked
- 1 bit per I/O port
  - 65536 ports -> 8kB
- Controls port access (0 == ok, 1 == GPF)
- L4: per-task I/O bitmap
  - Switched during task switch
  - Allows per-task grant/deny of I/O port access



Fiasco.OC: I/O Flexpages

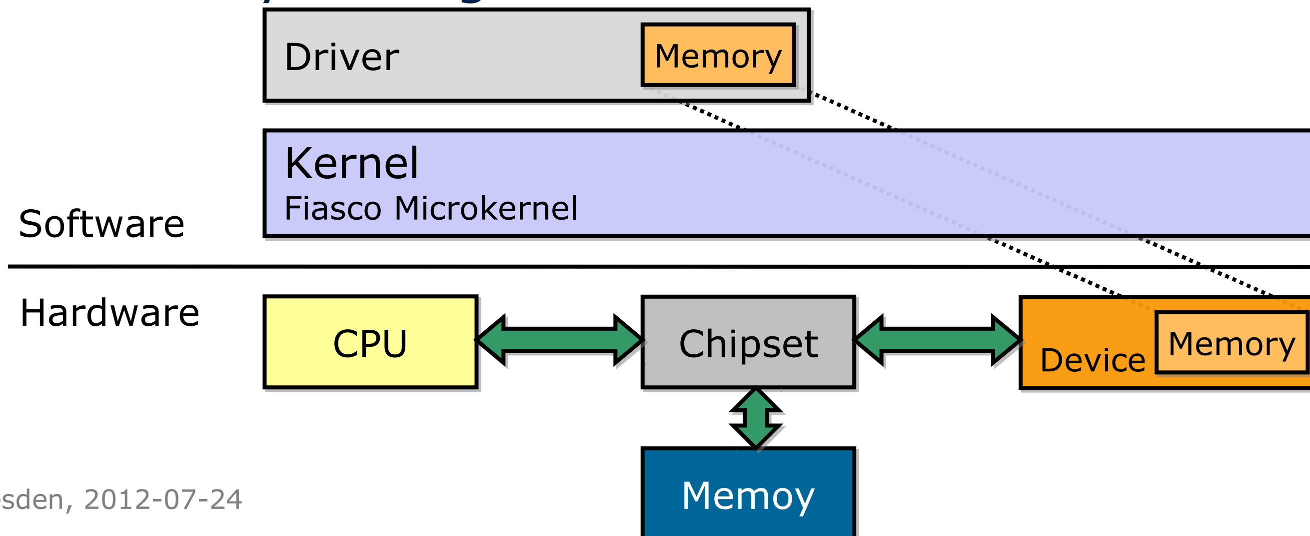
- Reuse kernel's map/grant mechanism for mapping I/O port rights -> I/O flexpages
- Kernel detects type of flexpage and acts accordingly
- Task with all I/O ports mapped is raised to IOPL 3

L4.Fiasco I/O flexpage format

TECHNISCHE UNIVERSITÄT DRESDEN

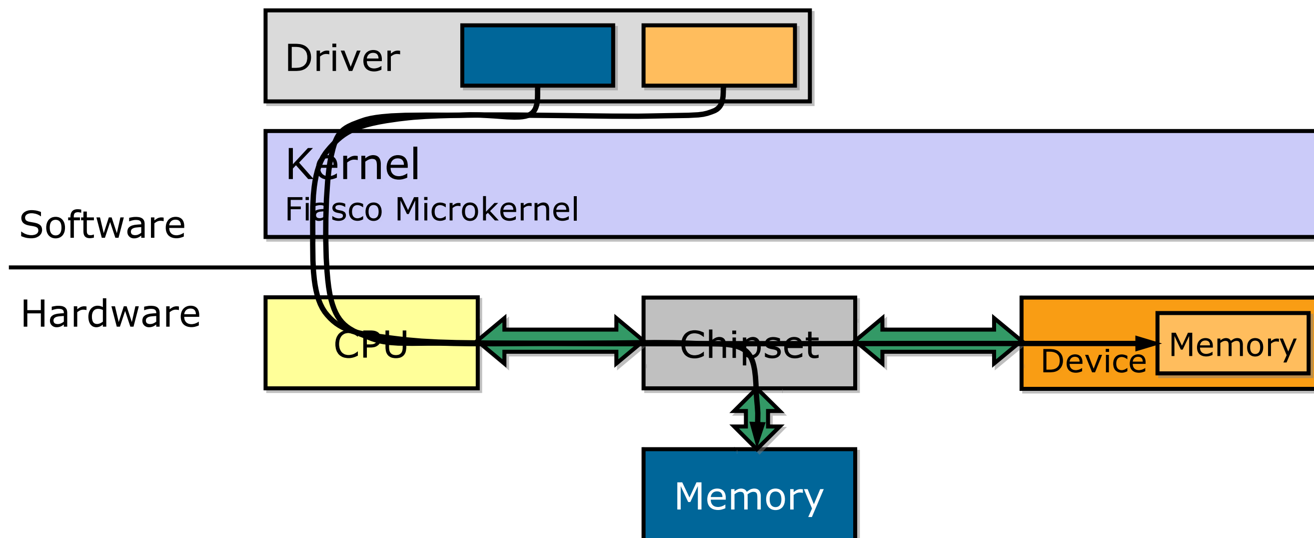
Hardware Resources: I/O Memory

- Devices often contain on-chip memory (NICs, graphcis cards, ...)
- Instead of accessing through I/O ports, drivers can map this memory into their address space just like normal RAM
  - no need for special instructions
  - increased flexibility by using underlying virtual memory management



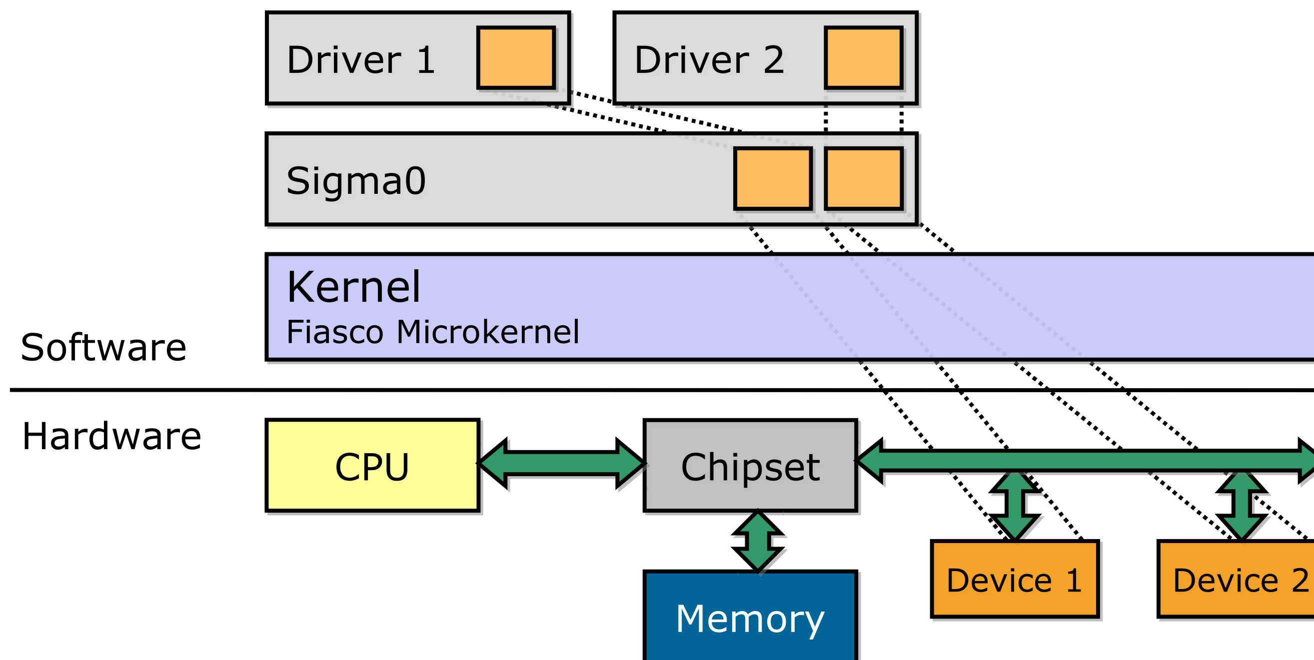
Hardware Resources: I/O memory (2)

- Device memory looks just like phys. memory
- Chipset needs to
  - map I/O memory to exclusive address ranges
  - distinguish physical and I/O memory access



I/O memory in L4

- Like all memory, I/O memory is owned by sigma0
- Sigma0 implements protocol to request I/O memory pages
- Abstraction: Dataspaces containing I/O memory

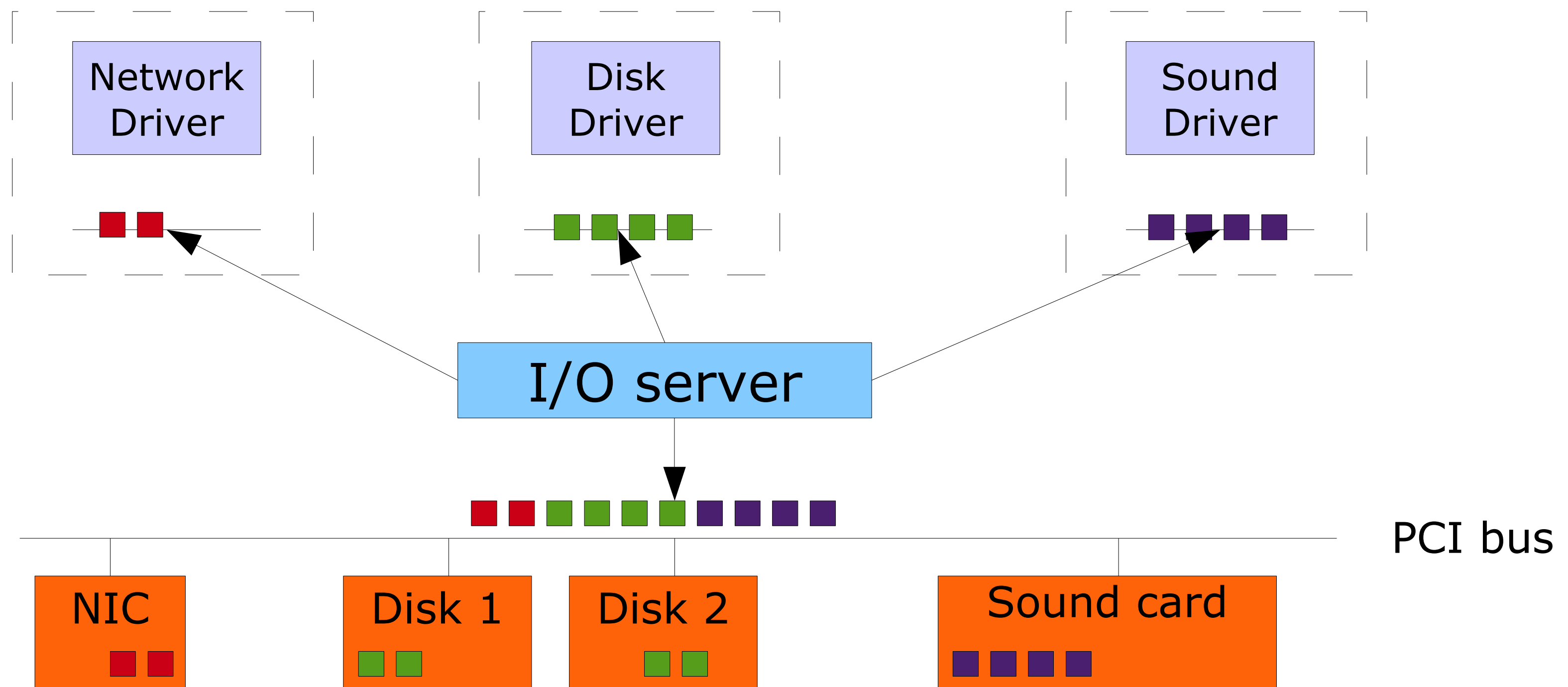


Can I trust my driver?

- Scenario: devices attached to a PCI bus
- Driver needs access to PCI configuration space
  - I/O resource discovery
  - Enable/disable device interrupts
- PCI config space access:
  - Write (bus, device, function#) to I/O port 0xCF8
  - Read/write data to/from I/O port 0xCFC
- Problem:
  - Who makes sure that my driver uses the proper (bus, device, function) tuple?

L4Re: Device Management

- Solution: only provide access to those devices, a driver is supposed to access
- I/O server + virtual buses



Summary: Fiasco.OC/L4Re Device Drivers

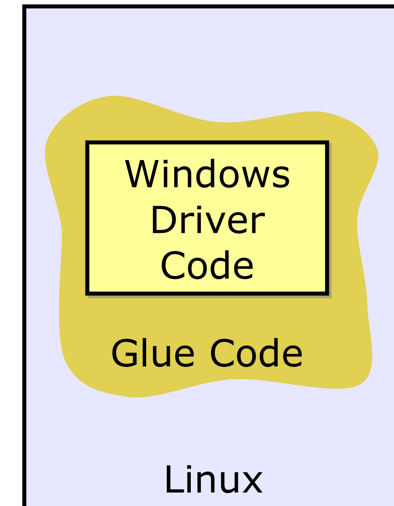
- Interrupts → Mapped to IPC
- I/O ports → Managed as kernel resource
- I/O memory → Cleanly integrates with L4 memory management
- PCI bus → Managed by separate I/O server

Implementing Device Drivers

- Just like in any other OS:
  - Specify a server interface
  - Implement interface, use the access methods provided by the runtime environment
- Highly optimized code possible
- Hard to maintain
- Implementation time-consuming
- Unavailable specifications
- Why reimplement drivers if there are already versions available?
  - Linux, BSD – Open Source
  - Windows – Binary drivers

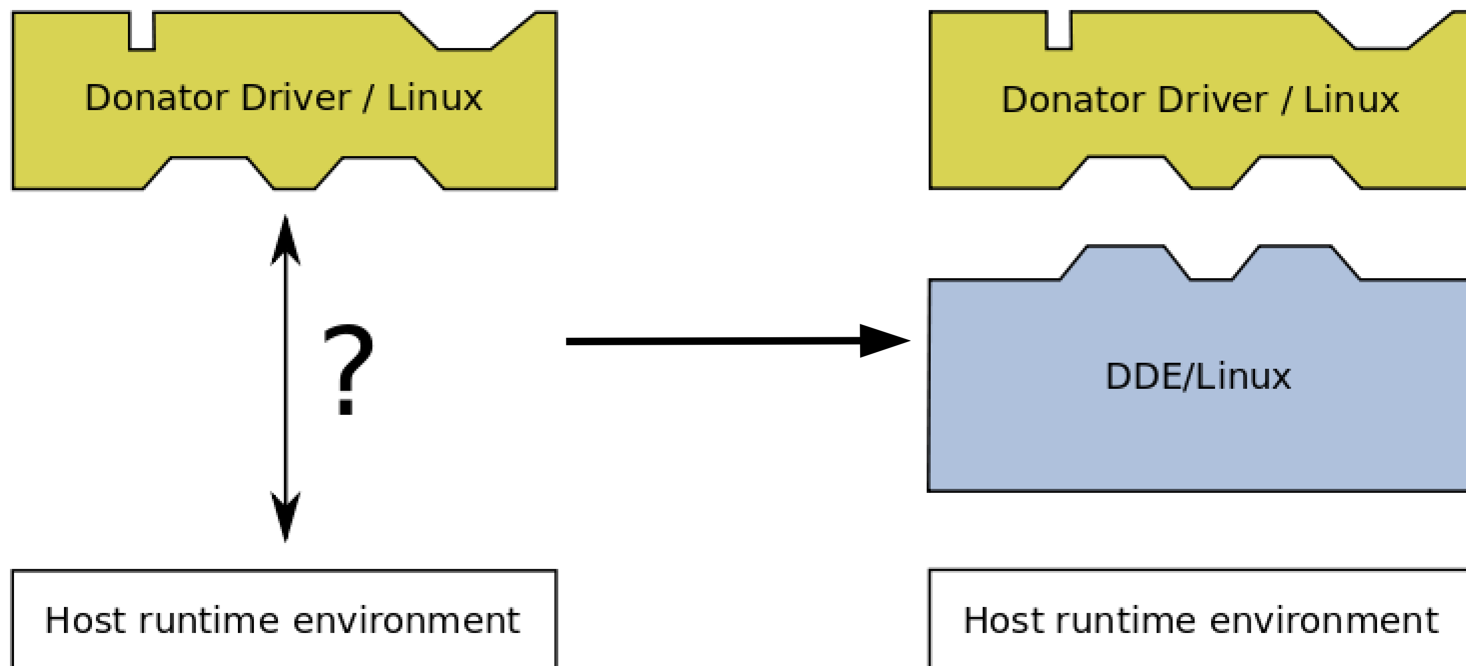
Reusing Legacy Device Drivers

- NDIS-Wrapper: Linux glue library to run Windows WiFi drivers on Linux
- Idea is simple: provide a library mapping Windows API to Linux
- Implementation is a problem.



Reusing Legacy Device Drivers (2)

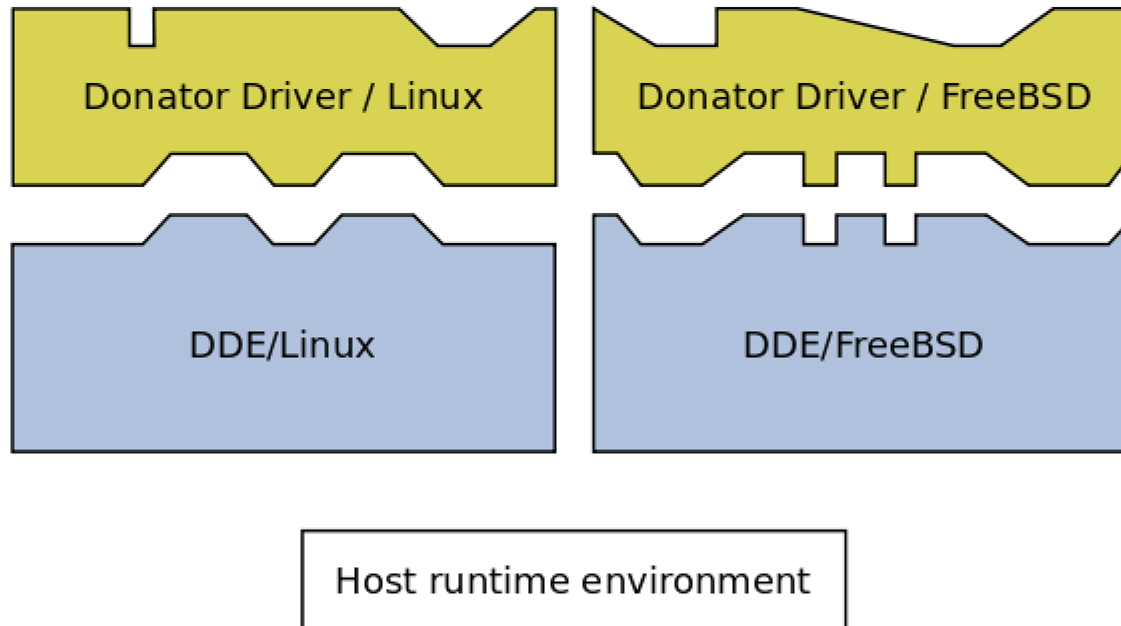
- Generalize the idea: provide a Linux environment to run drivers on L4
  → Device Driver Environment (DDE)



Emulating Linux: DDE/Linux

- Multiple L4 threads provide a Linux environment
  - Workqueues
  - SoftIRQs / Bottom Halves
  - Timers
  - Jiffies
- Emulate SMP-like system (each L4 thread assumed to be one processor)
- Wrap Linux functionality
  - kmalloc() → L4 Slab allocator library
  - Linux spinlock → pthread mutex
- Handle in-kernel accesses (e.g., PCI config space)

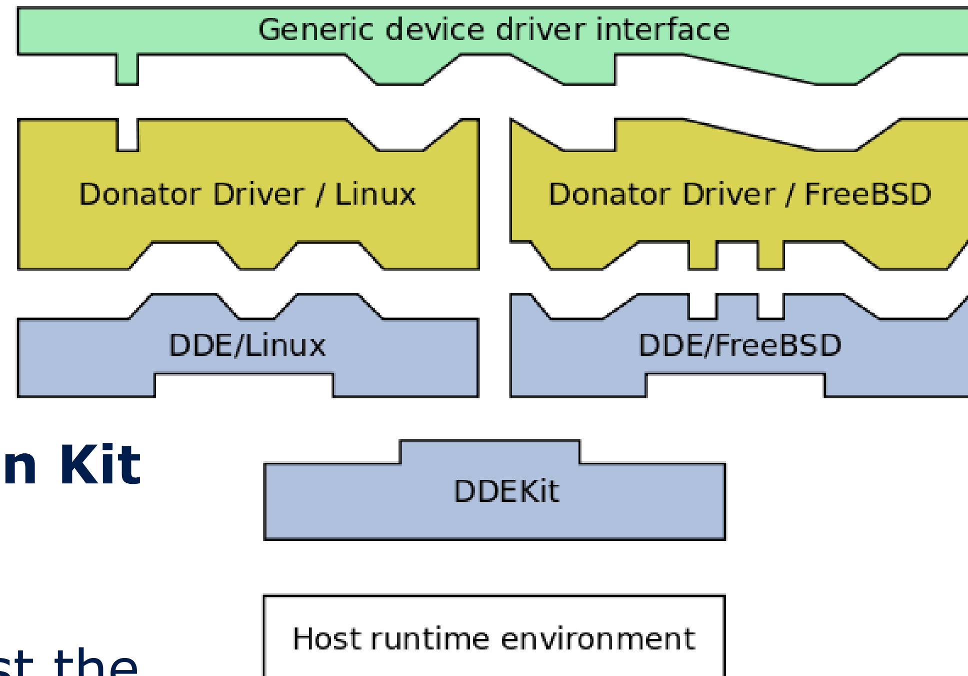
Multiple Donator OSes



DDEKit – another abstraction

- Pull common abstractions into dedicated library
  - Threads
  - Synchronization
  - Memory
  - IRQ handling
  - I/O port access

  → **DDE Construction Kit (DDEKit)**



- Implement DDEs against the DDEKit interface

DDEKit (2)

- Implementation overhead for single DDEs gets much smaller
- Performance overhead still reasonable
  - e.g., no visible increase of network latency in user-level ethernet driver
- L4-specific parts (sloccount):
  - standalone DDE Linux 2.4: **~ 3.000 LoC**
  - DDEKit **~ 2.000 LoC**
  - DDEKit-based DDE Linux 2.6: **~ 1.000 LoC**
  - Standalone Linux VM (DD/OS): **> 500.000 LoC**
- Highly customizable: implement DDE base library and support libs (net, disk, sound, ...)

TECHNISCHE UNIVERSITÄT DRESDEN

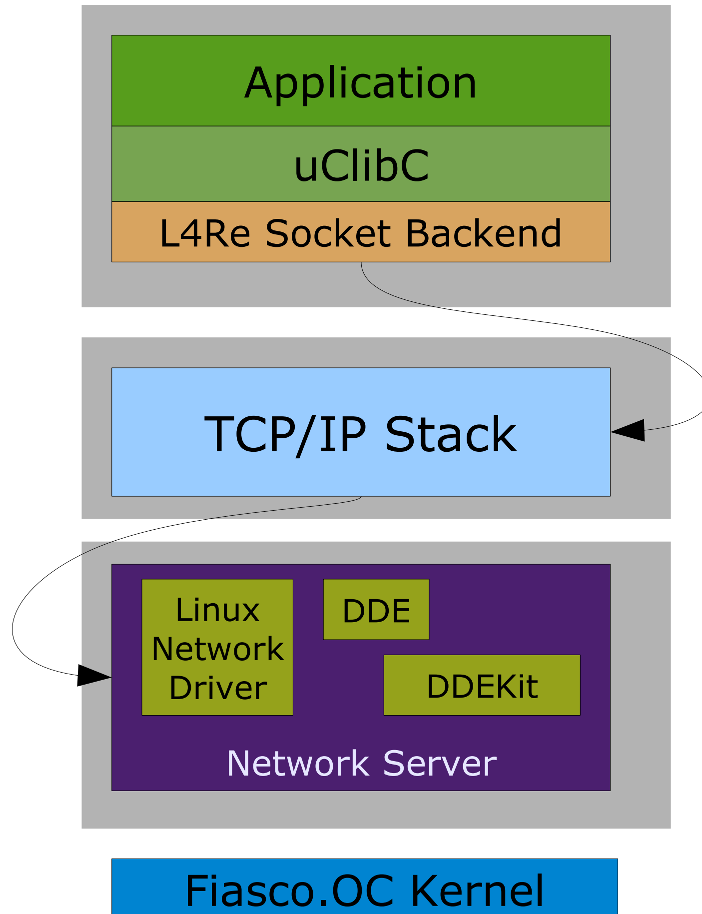
DDEKit: portability

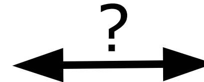
- Reversing the DDE idea: port DDEKit to host environment → reuse whole Linux support lib
- Has been done for:
  - L4Env, L4Re
  - Genode OS Framework
  - Minix 3
  - GNU/Hurd
  - Linux

Device Drivers: Isolation vs. Performance

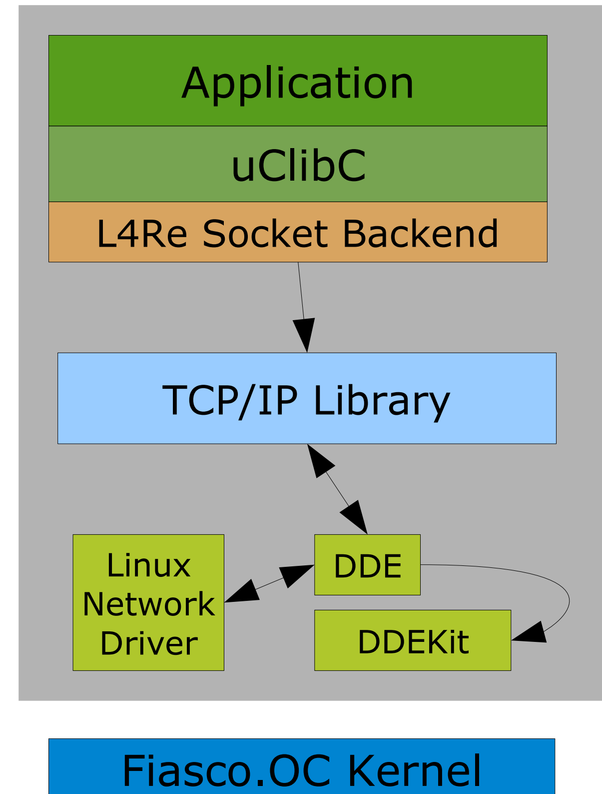
TECHNISCHE UNIVERSITÄT DRESDEN

Full Protection: Isolated Address Spaces





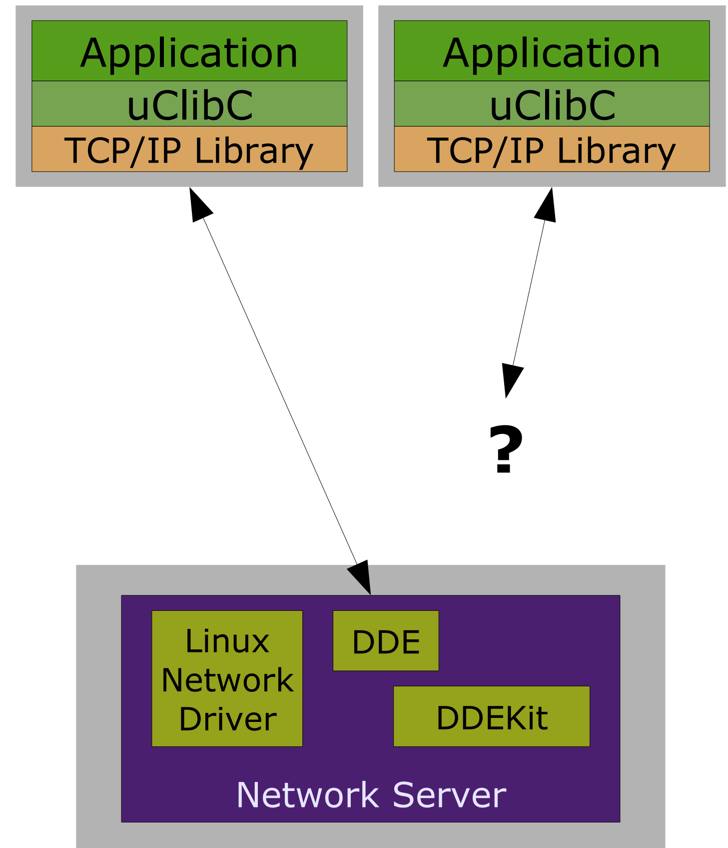
Best performance: all in one address space



TECHNISCHE UNIVERSITÄT DRESDEN

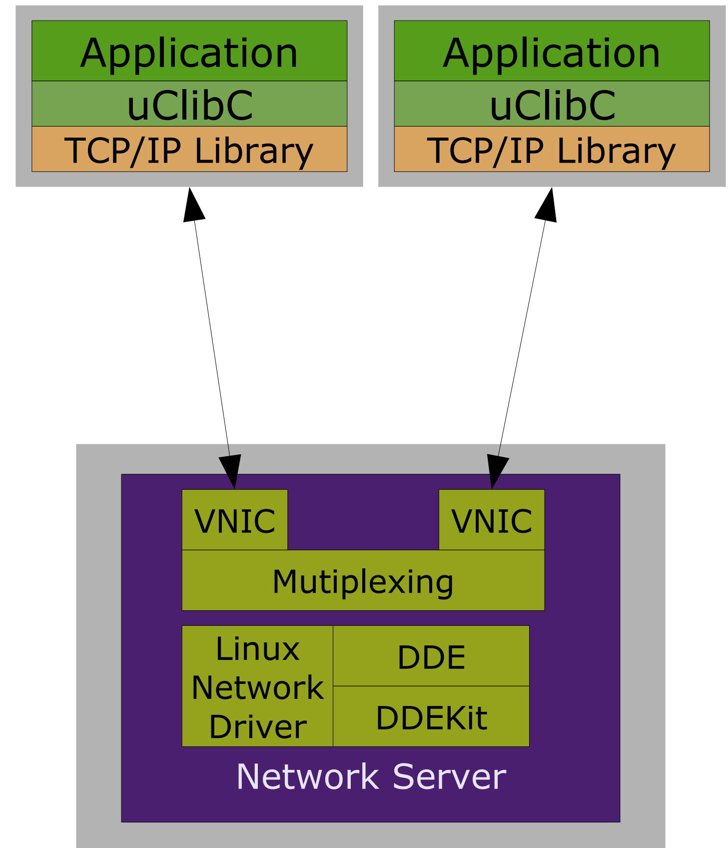
Device Sharing?

- Problem: Devices often don't support shared access from multiple applications



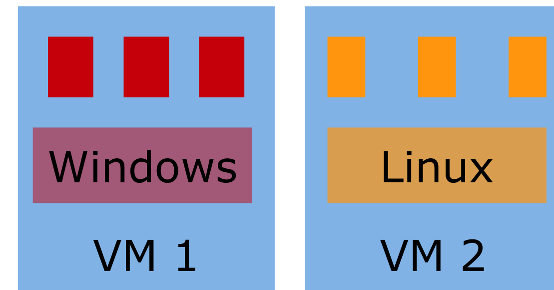
Device Sharing?

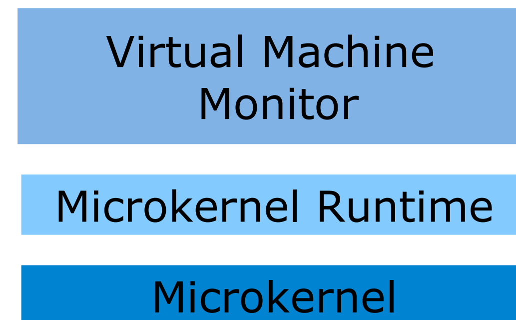
- Problem: Devices often don't support shared access from multiple applications
- Solution: Introduce “virtualized” intermediary interfaces
- Networking on L4Re: Ankh network multiplexer
  - Shared memory NIC for each client
  - Virtual MAC addresses



Virtual Machines

- Extreme approach to reusing existing software:
  - Run application inside the real OS
  - Wrap the OS in a virtual machine
  - Run the Virtual Machine on top of microkernel
- Advantages:
  - Full guest OS interface supported
  - Runs any binary that has been compiled for the guest OS without modification
- Problem: someone needs to implement a VMM!





Approaches to Virtualization

- Goal: Make guest OS think it runs on real HW
- Emulation: simulate every instruction
  - Very slow
  - Allows for incompatible guest and host hardware
- Trap&Emulate: when guest and host HW are identical
  - Execute on real processor as long as possible
  - Privileged instructions lead to processor traps
  - Handle those traps in VMM
  - Requires virtualizable instruction set!

Virtualizing Broken Instruction Sets

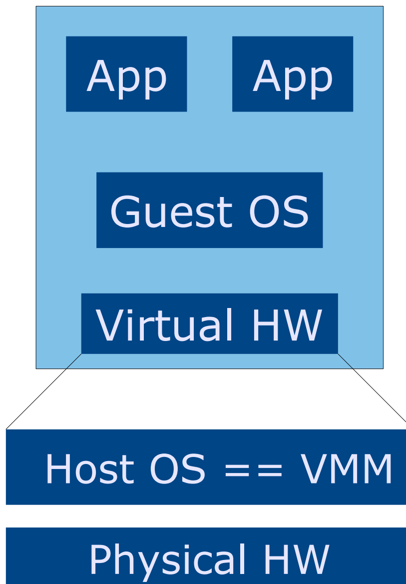
- Original x86 instruction set is not virtualizable
  - Some instructions behave differently in user and kernel mode
  - Not all such instructions cause a trap when executed in user mode
- Fix 1: Binary rewriting (VMware products)
  - Transform binary code so that broken instructions lead to a trap that can be handled by the VMM
- Fix 2: Correct the instruction set
  - Intel VT / AMD-V

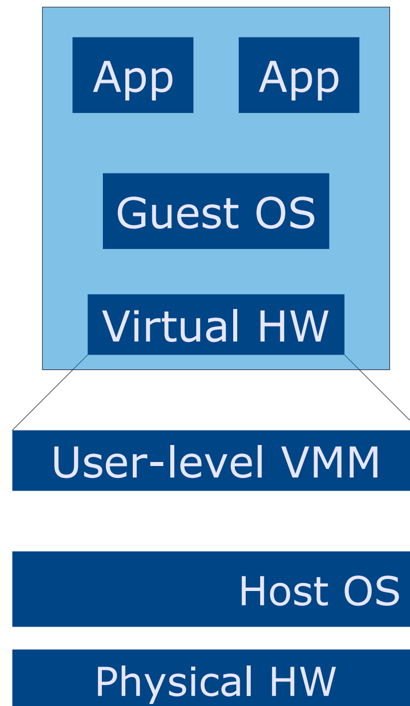
VMM Implementation Choices

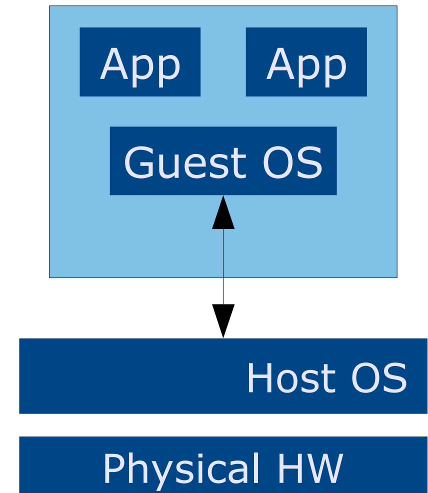
Bare-Metal VMM
- no OS overhead
- complete control
- high effort

VMM on OS
- reuse infra-structure
- may have worse performance

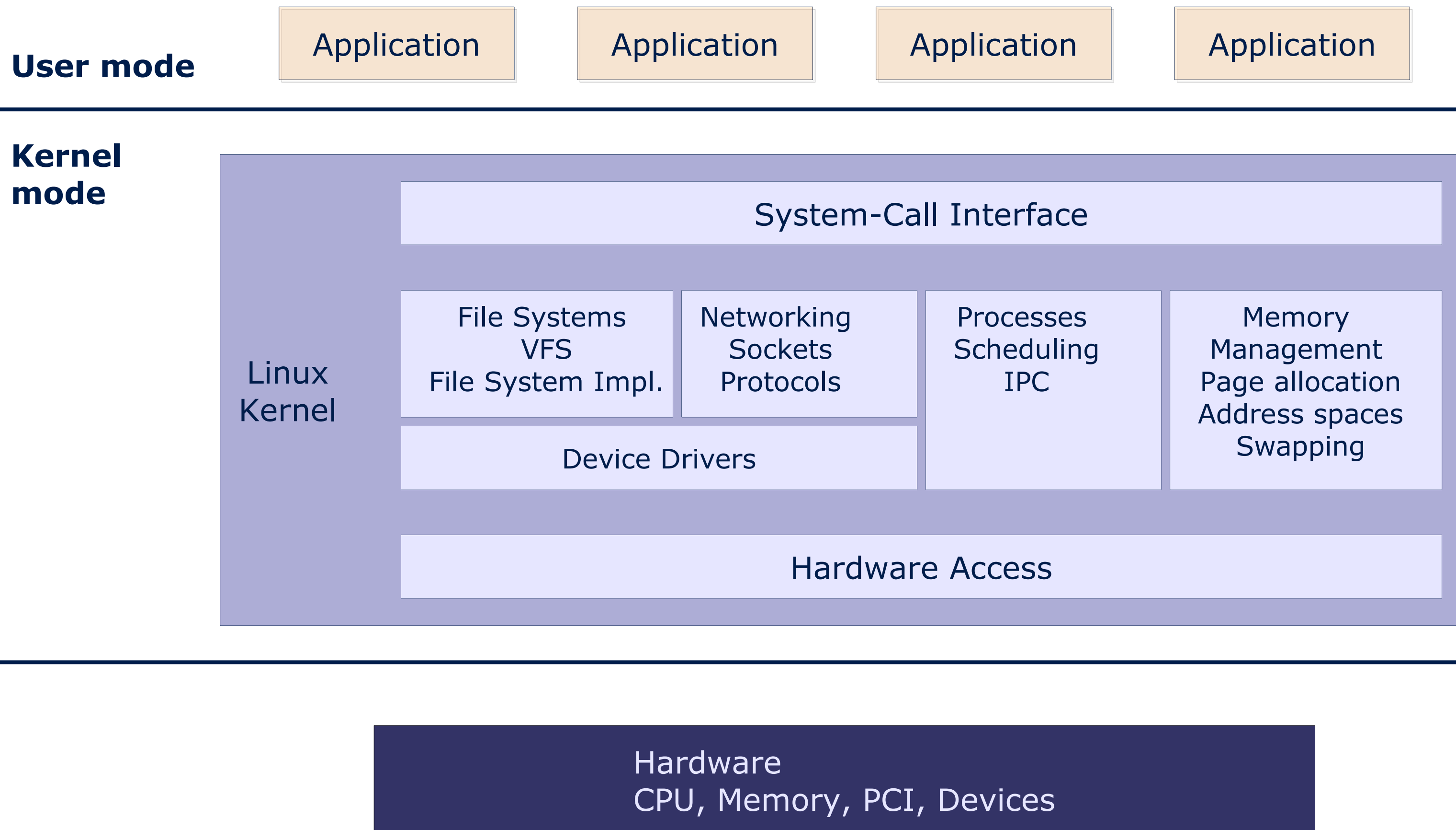
Paravirtualization
- port guest OS to host interface
- no VMM overhead
- maintenance cost





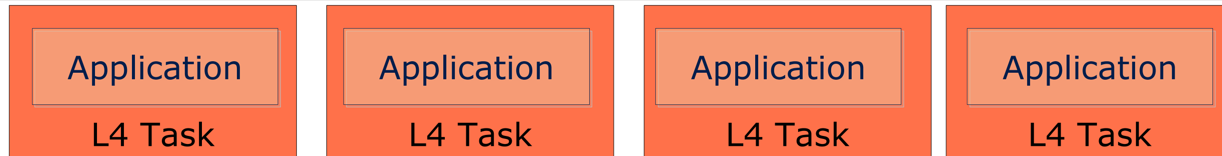


Paravirtualizing Linux



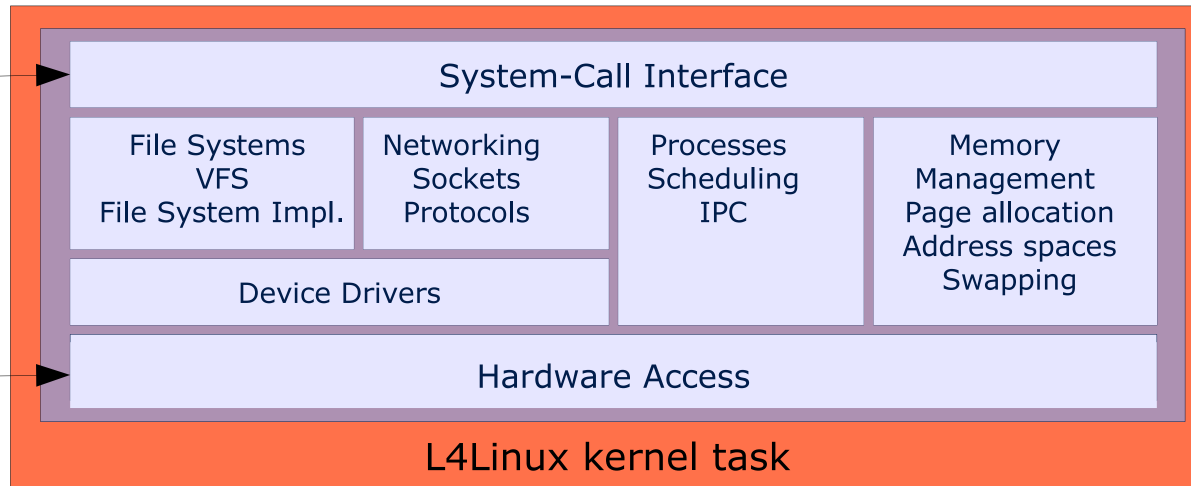
TECHNISCHE UNIVERSITÄT DRESDEN

Paravirtualizing Linux



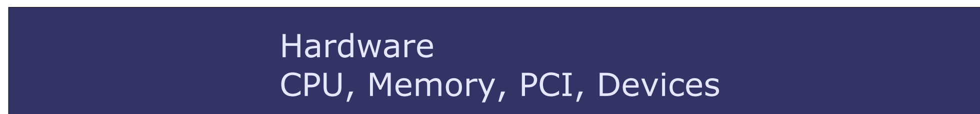
Adapt to L4 IPC

L4Re as HW architecture



User mode

Kernel mode



L4Linux

- L4 as new “hardware” platform in Linux, relay tasks to Fiasco.OC
  - Thread switching + state management
  - MMU handling
  - Use Fiasco.OC interrupts
- Linux applications are L4 tasks
  - IPC used for
    - Kernel entry
    - Signal delivery
    - Copying from/to user space

L4Linux: Kernel Entry

- Linux application executes INT 0x80 instruction

- Fiasco.OC receives CPU trap
- Fiasco.OC relays handling to user-level handler
  - Every thread has an **exception handler** assigned
  - Similar to a page fault, kernel generates exception IPC message
    - Contains whole CPU state

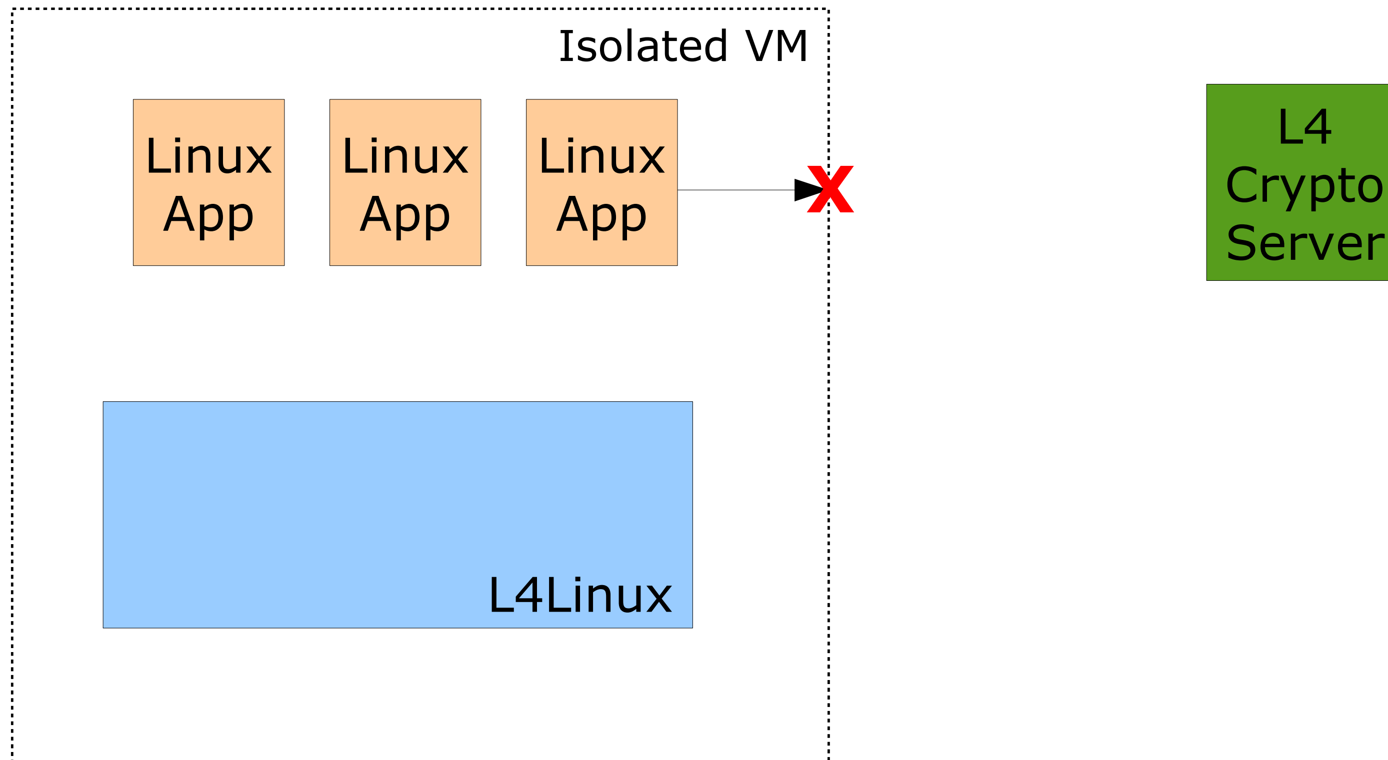
- Linux kernel task is exception handler for user applications
    - Receives exception IPC
    - Handles system call
    - Replies to let user app continue

Simplifying Exception Handling

- Fiasco.OC extension: **vCPU**
  - Thread with additional properties:
    - "kernel" task
    - "user" task
  - `l4_vcpu_resume()` → move thread to user task and let it execute
  - Upon CPU exception: move whole thread to kernel task and set it to dedicated kernel EIP/ESP
  - Kernel then calls `l4_vcpu_resume()` again
- Greatly simplifies L4Linux implementation:
  - Same mechanism used for interrupts, page faults, system calls, ...

Assignment #3

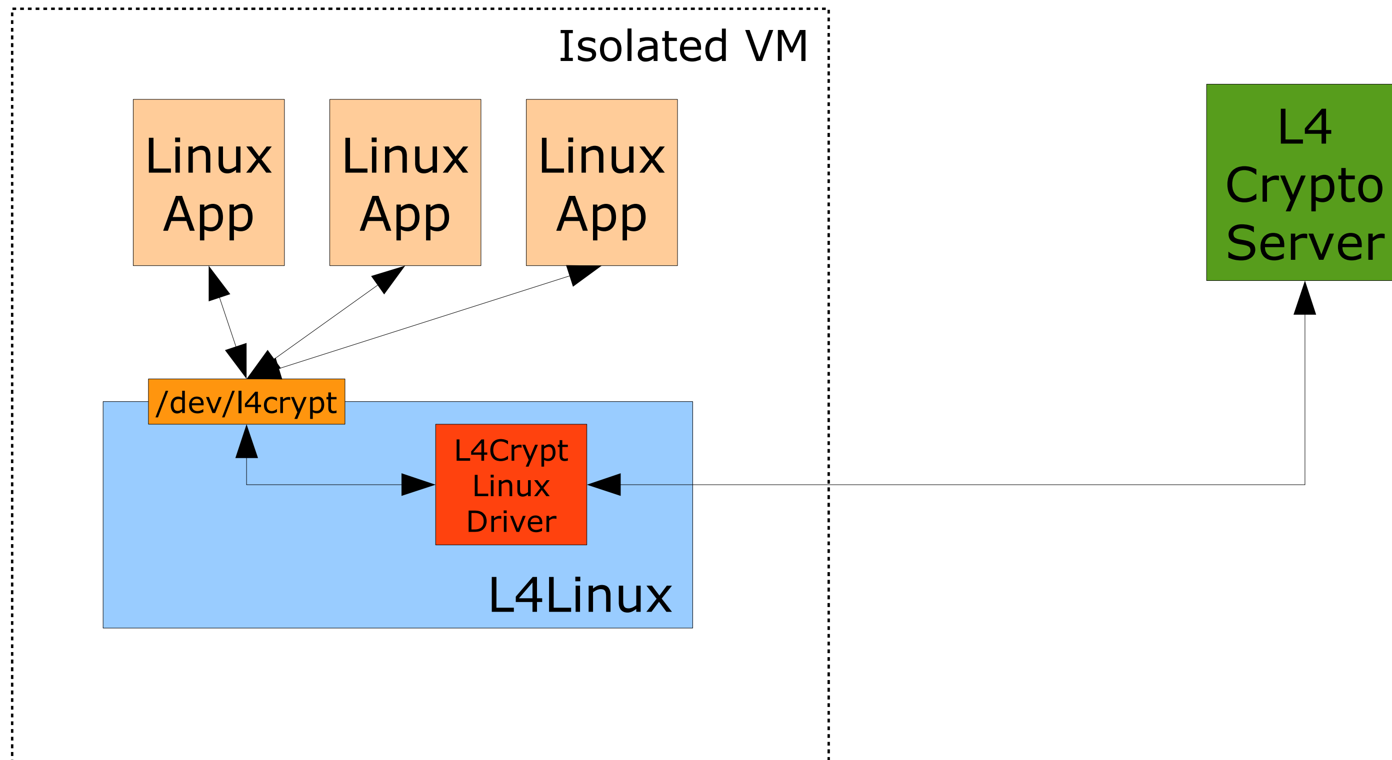
Make your encryption server (from assignment #2) accessible to an L^4Linux user application.



TECHNISCHE UNIVERSITÄT DRESDEN

Assignment #3

Make your encryption server (from assignment #2) accessible to an L4Linux user application.



Step 1: Download and build L4Linux

- Instructions: http://os.inf.tu-dresden.de/L4/LinuxOnL4/
- You will need a bunch of additional L4 packages
  - **Route A**: the build system complains about missing packages → fetch them with "svn up" in l4/pkg directory
  - **Route B**: this is the list you'll need:
    `acpica drivers examples fb-drv input io libevent libio-io libirq libvcpu lxfuxlibc mag mag-gfx rtc shmc x86emu zlib`
- When configuring L4Linux:
  - Leave L4 options as they are (only set proper L4Re build directory)
  - Disable as much HW options (PCI, SATA, SCSI, Networking, ...) as you can
    - Faster builds
    - We don't need hardware right now

Step 2: Running L4Linux

- Example files in l4/conf/examples:
  - *l4lx-x86.io* → Config file for virtual PCI bus (that will contain PS2 input device, frame buffer and any PCI devs … but you disabled PCI in step 1, remember?)
  - *l4lx-gfx.cfg* → Lua init script to launch L4Linux and a graphical console
- There's also an entry in l4/conf/modules.lst already.
- For a root file system:
  - Download an initrd, e.g., from http://os.inf.tu-dresden.de/download/
  - Build your own initrd

TECHNISCHE UNIVERSITÄT DRESDEN

Step 3: Implementing the Driver

- Linux side: use a standard chardev for communication between driver and applications

- L4 side:
  - L4Linux kernel modules are built as any other Linux kernel module
  - You may include and use L4 headers as in any other L4 application
  - For IPC: no C++ streaming – Linux is C code, so write to the UTCB yourself

Happy Hacking!

Don't perform **any** L4 system call between writing the UTCB and sending the IPC message to your server!

Log output (printk, ...) is an L4 system call!

Further Reading

- H. Weisbach et al. *"Generic User-Level PCI Drivers"*
  http://os.inf.tu-dresden.de/papers_ps/rtlws2011-dde.pdf
  Bringing the DDE approach to Linux

- Lackorzynski et al. *"Generic Virtualization with Virtual Processors"*
  http://os.inf.tu-dresden.de/papers_ps/rtlws2010_genericvirt.pdf
  Introducing the L4Linux vCPU model

- Steinberg et al.: *"NOVA: A Microhypervisor-based Secure Virtualization Architecture"*
  http://os.inf.tu-dresden.de/papers_ps/steinberg_eurosys2010.pdf
  A bare-metal microhypervisor

- Singaravelu et al.: *"Enforcing Configurable Trust in Client-side Software Stacks by Splitting Information Flow"*
  http://os.inf.tu-dresden.de/papers_ps/git-cercs-07-11.pdf
  Splitting applications into trusted and untrusted parts