TECHNISCHE UNIVERSITÄT DRESDEN

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

LEGACY REUSE

CARSTEN WEINHOLD

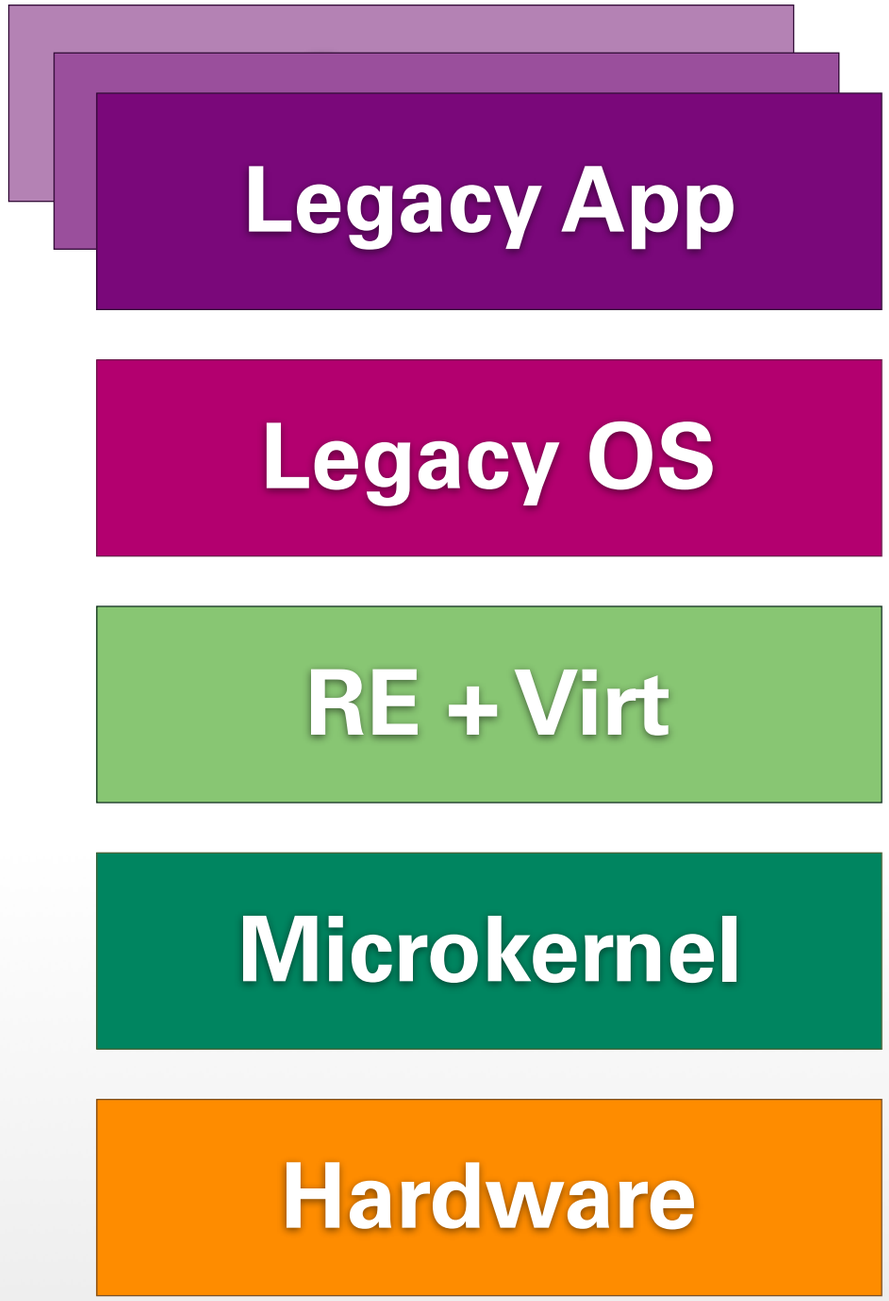
THIS LECTURE ...

- **So far ...**
  - Basic microkernel concepts
  - Drivers, resource management
- **Today:**
  - How to provide legacy OS personalities
  - How to reuse existing infrastructure
  - How to make applications happy

SINGLE SERVER

Getting applications the easy way:

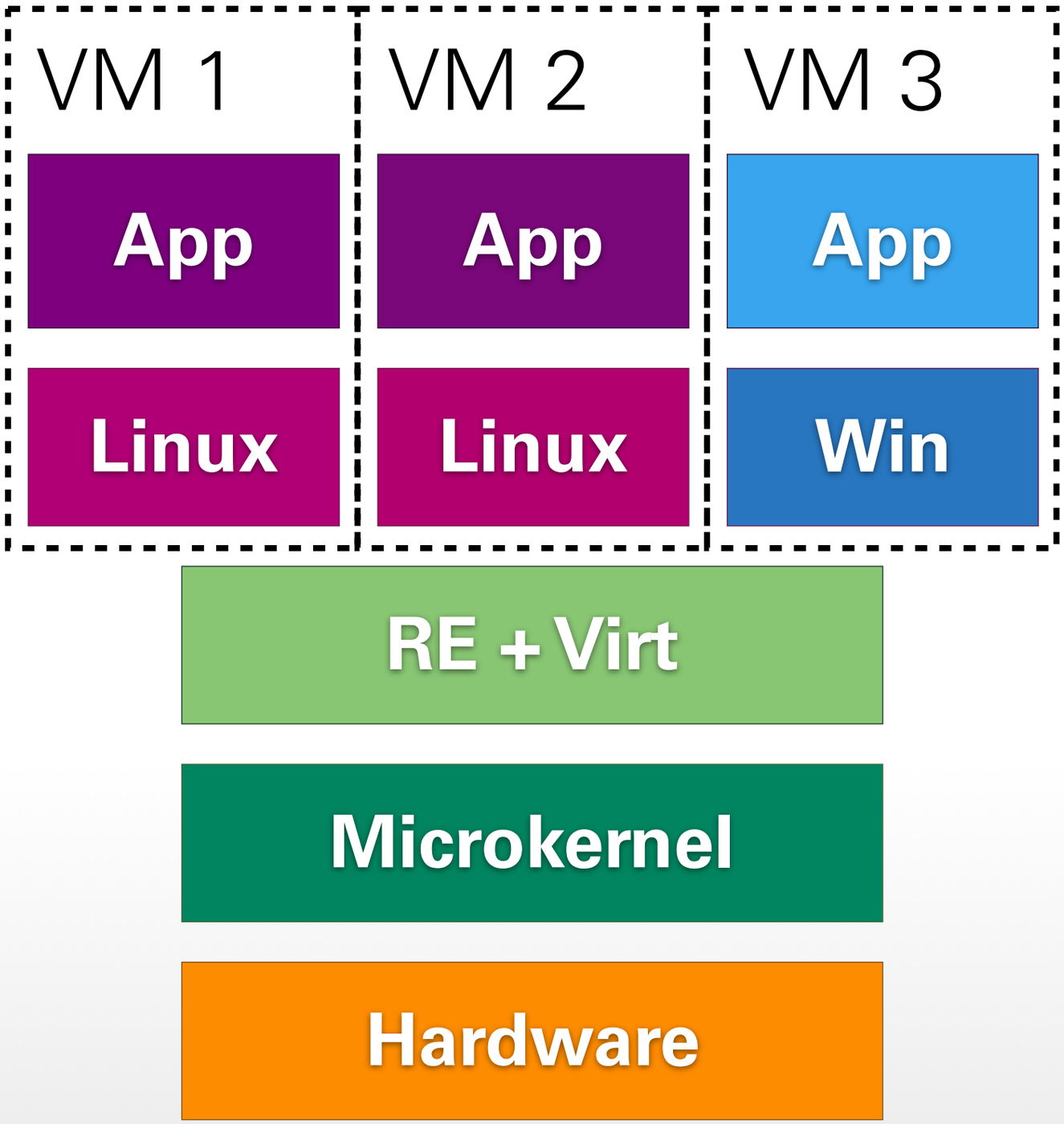
- Virtualize OS + app
- All services provided legacy OS (e.g., Linux)
- Original implementation, good app compatibility
- Limited isolation



MULTI-VM SETUP

Multiple VMs:

- Improved isolation
- Communication via virtual network
- Run same OS multiple times
- Run different OSes concurrently



VIRTUALIZATION

- **Virtualization:**
  - Reuses legacy OS + applications
  - Applications run in their natural environment
- **Problem:** Applications trapped in VMs
  - Different resource pools, namespaces
  - Cooperation is cumbersome (network, ...)
  - Full legacy OS in VM adds overhead
  - Management overhead, multiple desktops?

MAKING THE CUT

- **Hardware level** Next week
  - Virtualize legacy OS on top of new OS
- **Operating System Personality** Today
  - Legacy OS's interfaces reimplemented on top of – or ported to – new OS
- **Hybrid operating systems**
  - Run legacy OS virtualized …
  - … but tightly integrate it with new OS running underneath

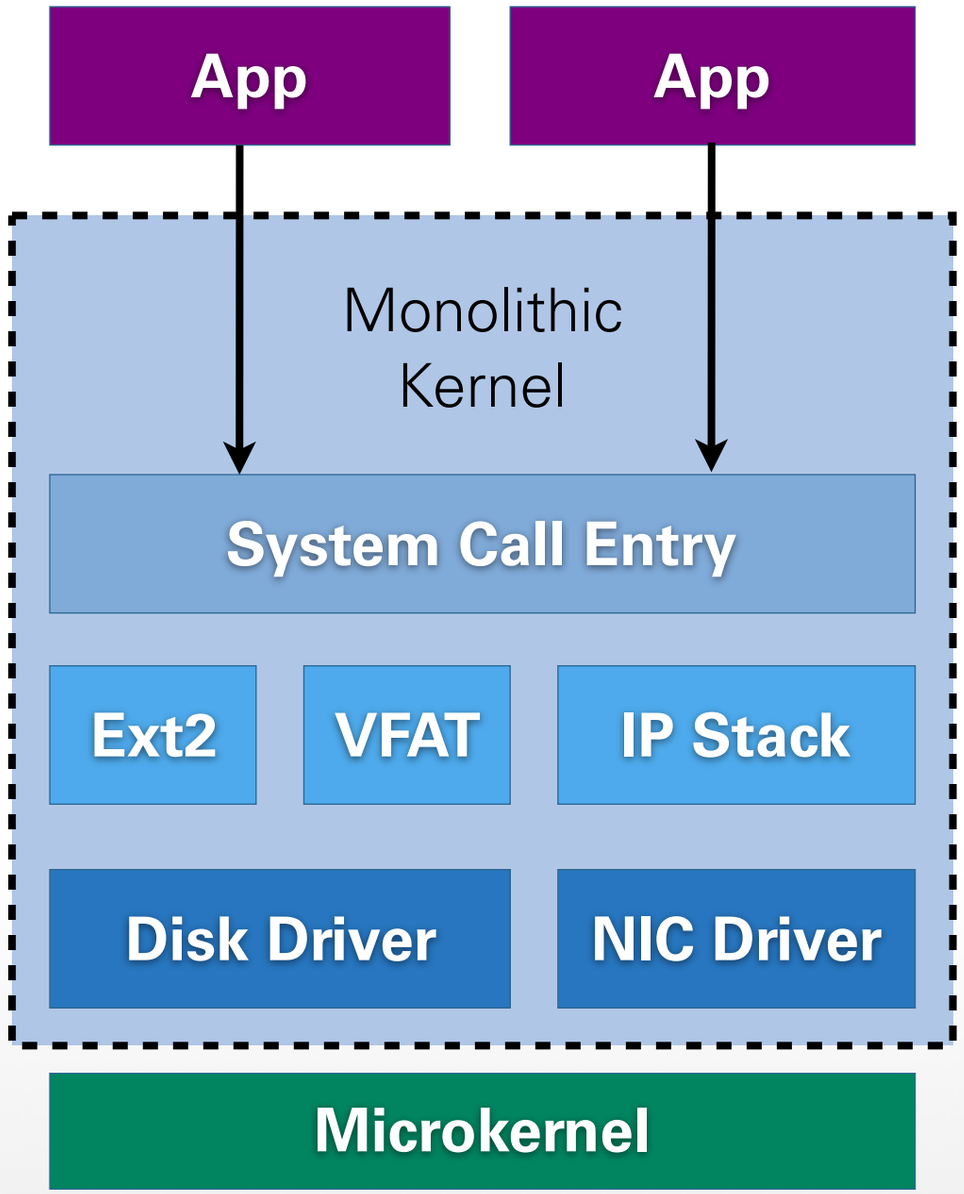
TECHNISCHE UNIVERSITÄT DRESDEN

OPERATING SYSTEM PERSONALITIES

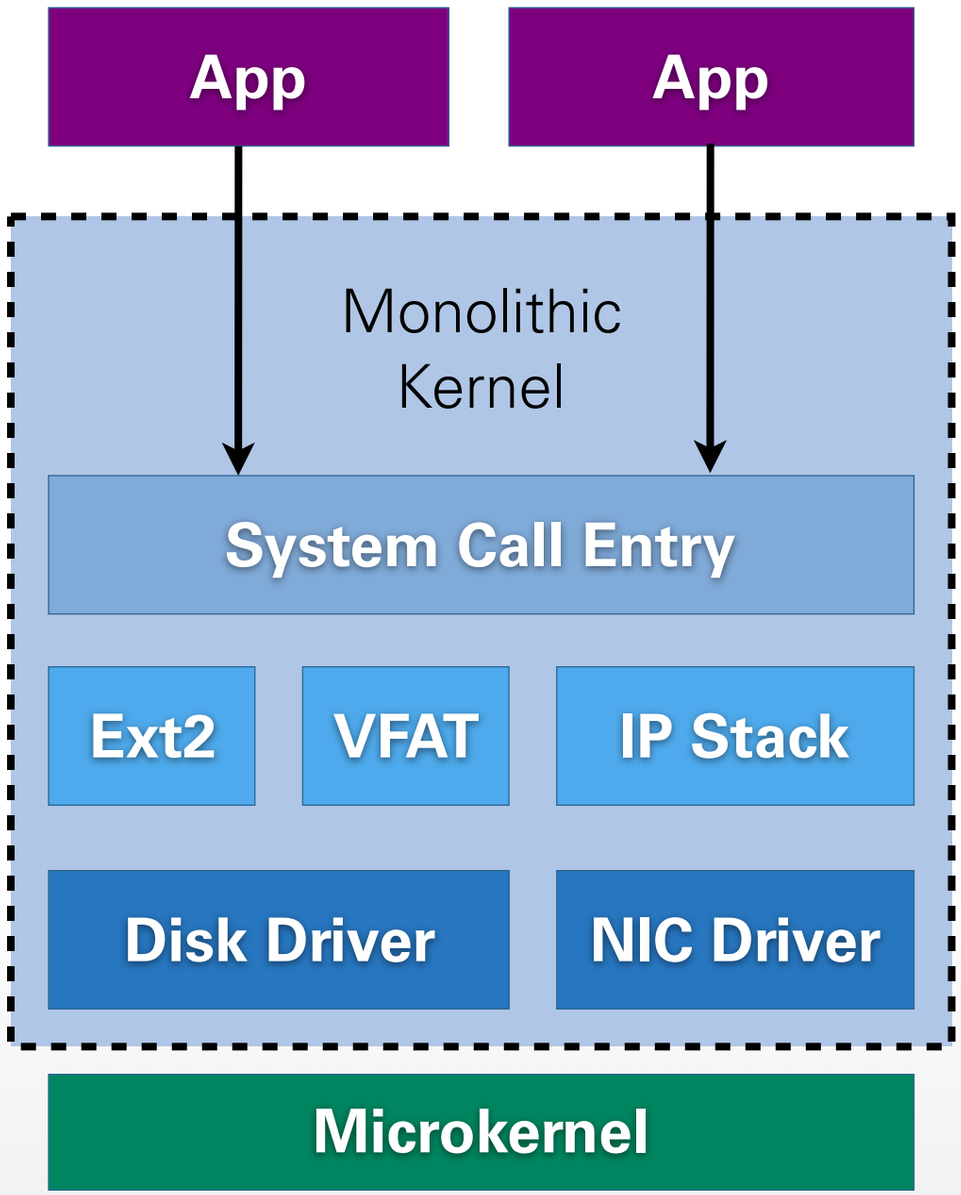
OS PERSONALITY

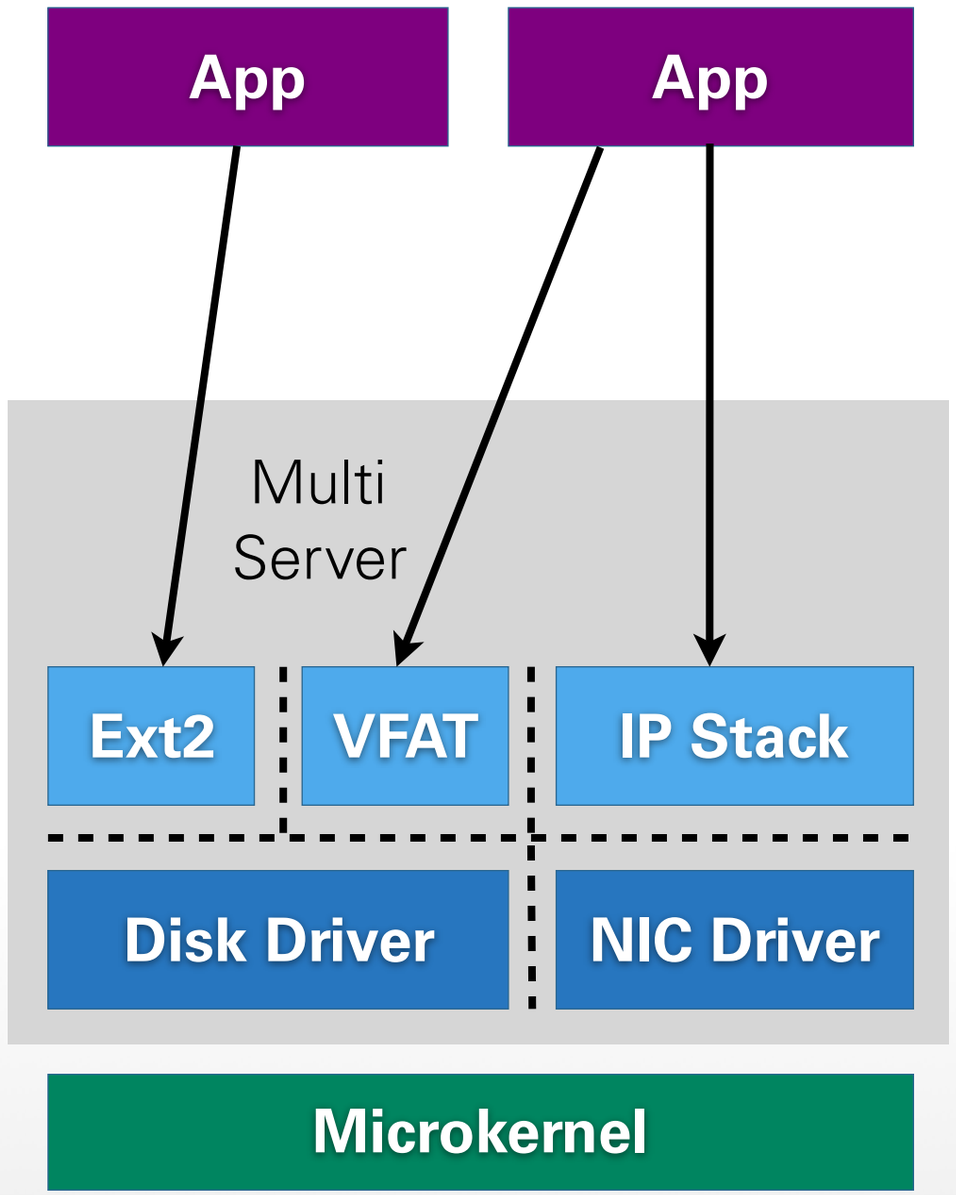
- **Idea:** Adapt at OS / application boundary
  - (Re-)Implement legacy APIs, not whole OS
  - May need to recompile application
- **Benefits:**
  - Get desired application, established APIs
  - Good integration (namespaces, files, ...)
  - Smaller overhead than virtualization
  - Flexible, configurable, but … more effort?

SINGLE SERVER



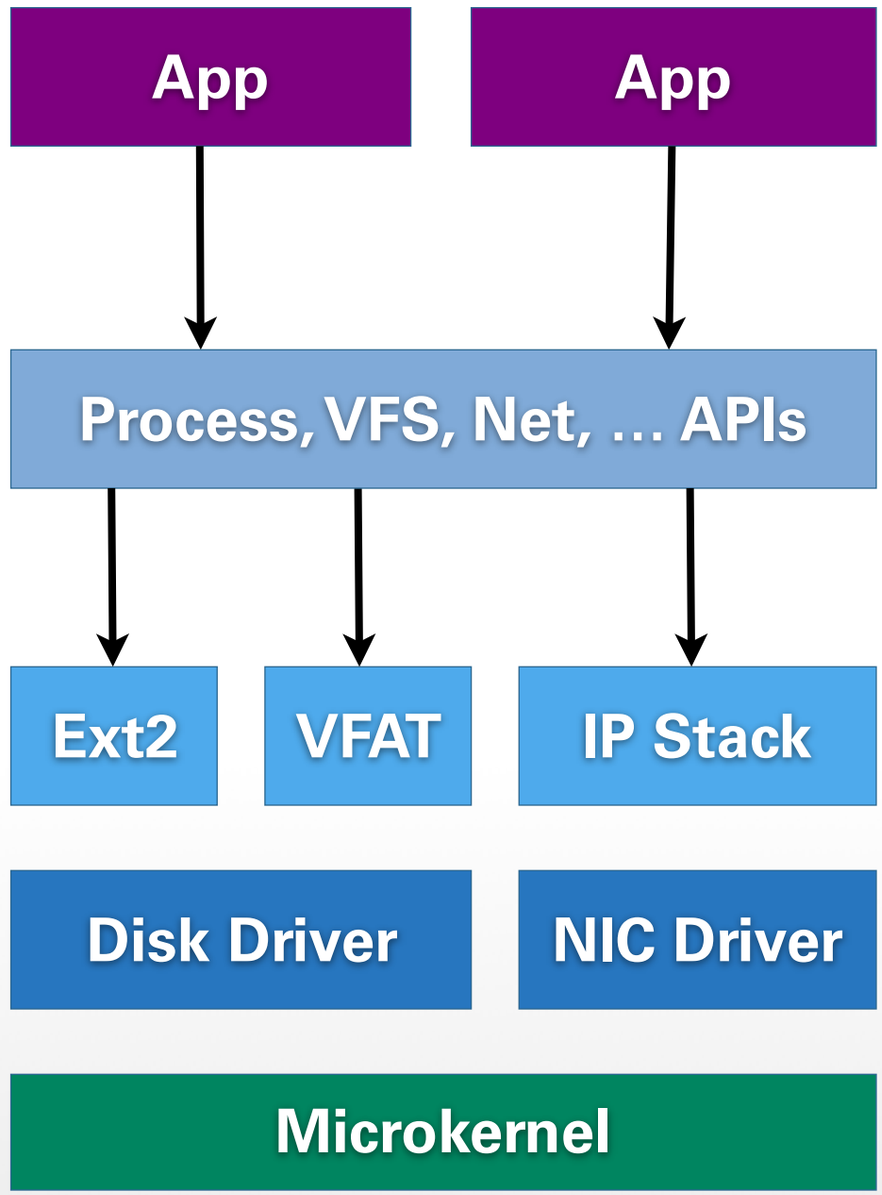
DECOMPOSITION





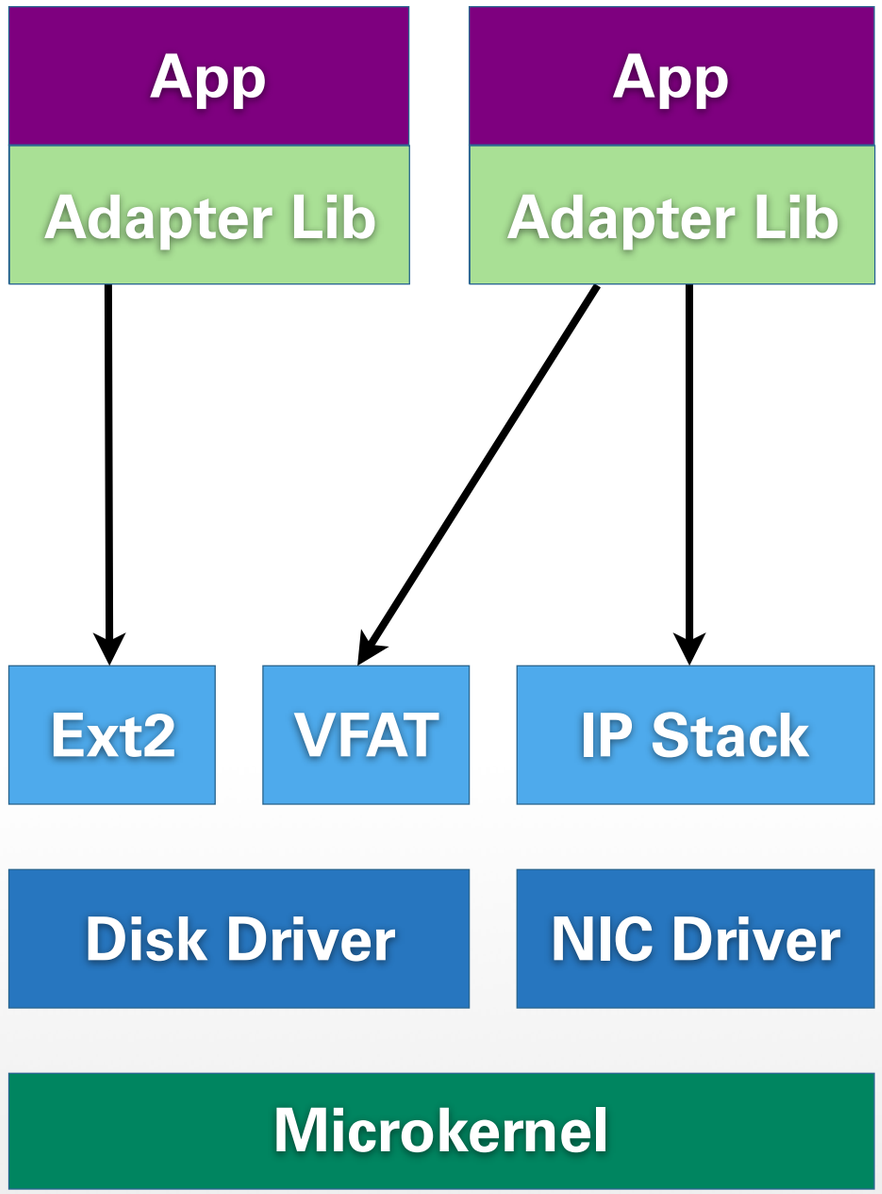
SINGLE ADAPTER

- Central adapter provides consistent view for:
  - **Servers:** client state (e.g., file tables)
  - **Applications:** system resources (e.g., files)
- Potential issues:
  - Bottleneck
  - Single point of failure
  - Little flexibility, no isolation



ADAPTER LIB

- Adapter library:
  - Linked into applications
  - Interacts with servers
  - Provides consistent view (per application)
- Each server keeps its own client state
- Compatibility: adapter library hidden below libc



METHODOLOGY

- **What to do:**
  - Determine what application needs
  - Provide the needed APIs
- **Approach:**
  - Do not reinvent the wheel!
  - Reuse libraries, map to existing APIs
  - Make everything modular
  - Keep compatibility (may drop some APIs)

POSIX STANDARD

- „Portable Operating System Interface“ is a family of standards (POSIX 1003.*)
- POSIX makes UNIX variants source-code compatible (also introduced in Windows NT)
- Defines interfaces and properties:
  - I/O: files, sockets, terminal, ...
  - Threads, synchronization: Pthread
  - System tools
- Accessible through C library

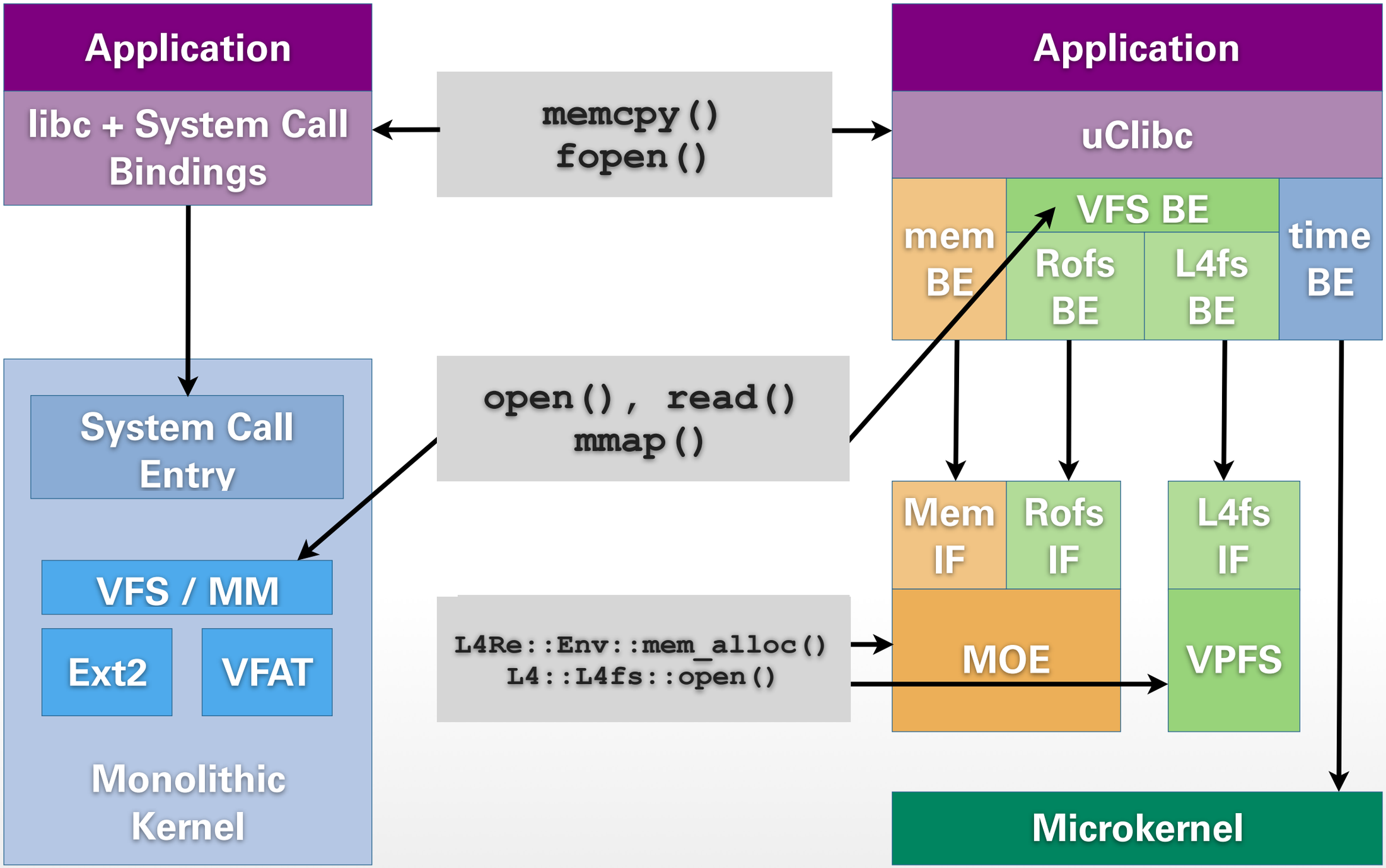
WHAT IS LIBC?

- C library abstracts underlying OS
- Collection of common functionality
- Abstraction level varies:
  - low level: **`memcpy()`**, **`strlen()`**
  - medium level: **`fopen()`**, **`fread()`**
  - high level: **`getpwent()`**
- ... and so do dependencies:
  - none (freestanding): **`memcpy()`**, **`strlen()`**
  - small: **`malloc()`** depends on **`mmap()`**
  - strong: **`getpwent()`** needs file access, name service, ...

LIBC ON L4RE

- libc support on L4Re: uClibc
  - Compatible to GNU C library „glibc“
  - Works well with libstdc++
  - Small and portable
  - Designed for embedded Linux
- But: Fiasco.OC + L4Re != Linux
- How to port a low-level library?

MULTI-SERVER LIBC



LIBC BACKENDS

- Four examples:
  - Time
  - Memory
  - Signals
  - I/O

LIBC BE: TIME

Example 1: POSIX time API

```
uint64_t __libc_l4_rt_clock_offset;

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

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

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

  return 0;
}
```

L4Re-specific backend function (called by **`time()`** and other POSIX functions)

Replacment of POSIX function **`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;
}
```

LIBC BE: MEMORY

Example 2: memory management

- uClibc implements heap allocator
- Requests memory pages via mmap()
- Can be reused, if we provide **`mmap()`**
  - Minimalist: use static pages from BSS
  - l4re_file:
    - Supports **`mmap()`**, **`munmap()`** for anon memory
    - Based on dataspaces and L4Re region manager
    - Usually gets memory from MOE

ANONYMOUS MEM

- **`malloc()`** calls **`mmap()`** with flags **`MAP_PRIVATE|MAP_ANONYMOUS`**
  - Pages taken from large dataspace
  - Attached via L4RM interface
  - Reference counter tracks mapped regions
- **`munmap()`** detaches dataspace regions
  - **if** (region_split) refs++; **else** refs--;
  - Dataspace released on zero references

LIBC BE: SIGNALS

Example 3: POSIX signals

- Used for asynchronous event notification:
  - Timers: **`setitimer()`**
  - Exceptions: **`SIGFPE`**, **`SIGSEGV`**, **`SIGCHLD`**, ...
  - Issued by applications: **`SIGUSR1`**, **`SIGUSR2`**
- Signals on Linux:
  - Built-in kernel mechanism
  - Delivered upon return from kernel
- **How to implement signals in L4Re?**

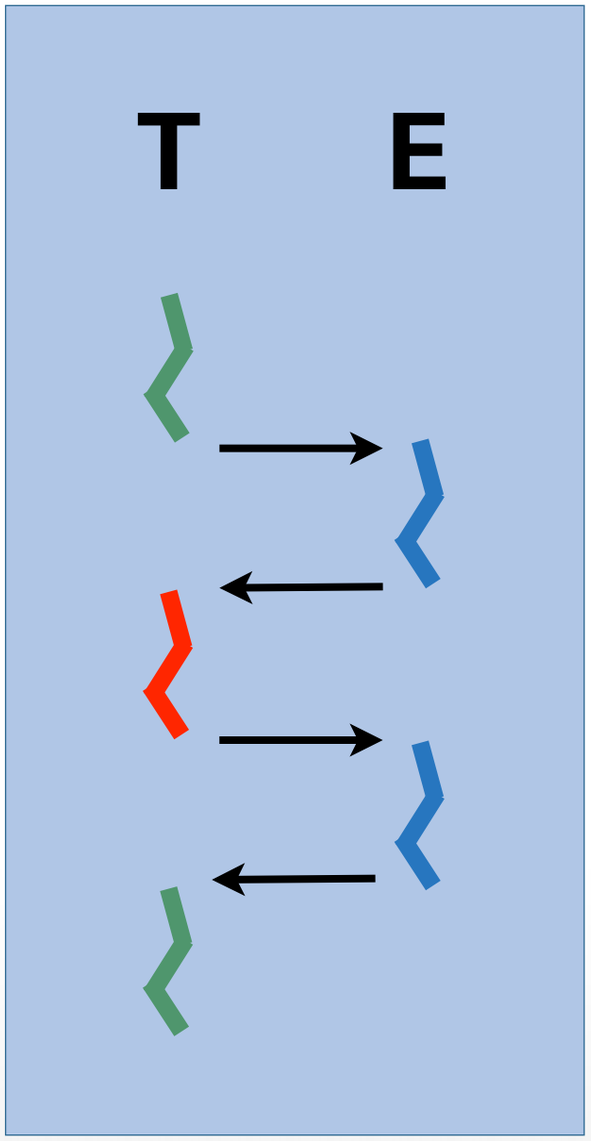
SIGNALS ON L4?

- Use exception handler mechanism:
  - Start exception handler thread, which waits in a loop for incoming exceptions
  - Set this exception handler for all user threads
  - Let kernel forward exceptions as IPC messages

- Timers can be implemented as IPC timeouts:
  - **`sigaction()`** / **`setitimer()`** called by **T**
  - **T** communicates time to wait to **E**
  - **E** waits for IPC timeout
  - **E** raises exception in T to deliver **`SIGALRM`**

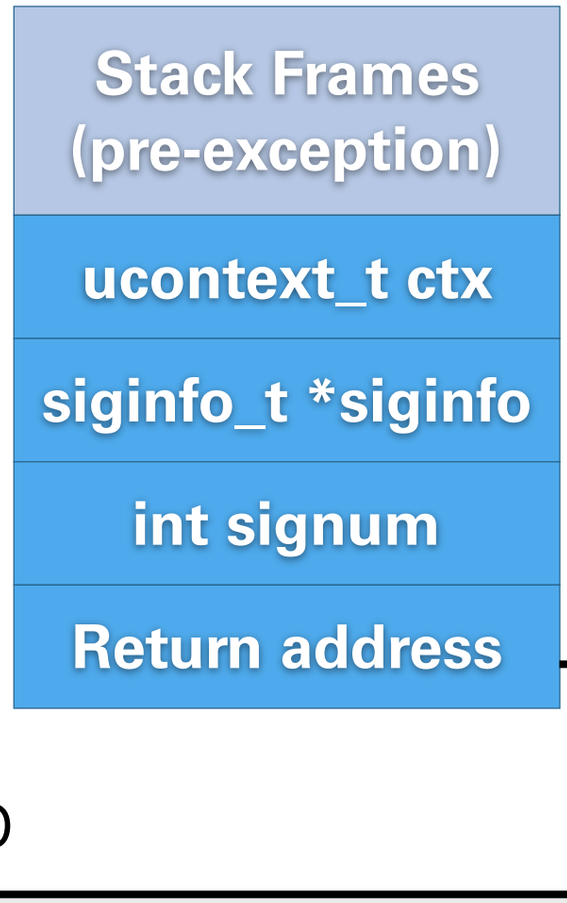
SIGNALS: L4RE

- Dedicated thread **E** handles exceptions and timers
- **E** is exception handler of thread **T**
- Exceptions in **T** are reflected to **E**
- If app configured signal handler:
  - **E** sets up signal handler context
  - **E** resets **T**'s program counter to start of signal handler
  - **T** executes signal handler, returns
- If possible, **E** restarts **T** where it had been interrupted



SIGNALS: MECHANICS

- **E**: handles exceptions:
  - Set up signal handler context:
    - Save **T**'s context
    - Push pointer to siginfo_t, signal number
    - Push address of return trap
  - **`l4_utcb_exc_pc_set(ctx, handler)`**
- **T**: execute signal handler, „returns“ to trap
- **E**: resume thread after signal:
  - Exception generated, reflected to E
  - Detects return by looking at T's exception PC
  - Restore T's context saved on stack, resume




```
void libc_be_sig_return_trap()
{
  /* trap, cause exception */
}
```

I/O SUPPORT?

Example 4: Simple I/O support:

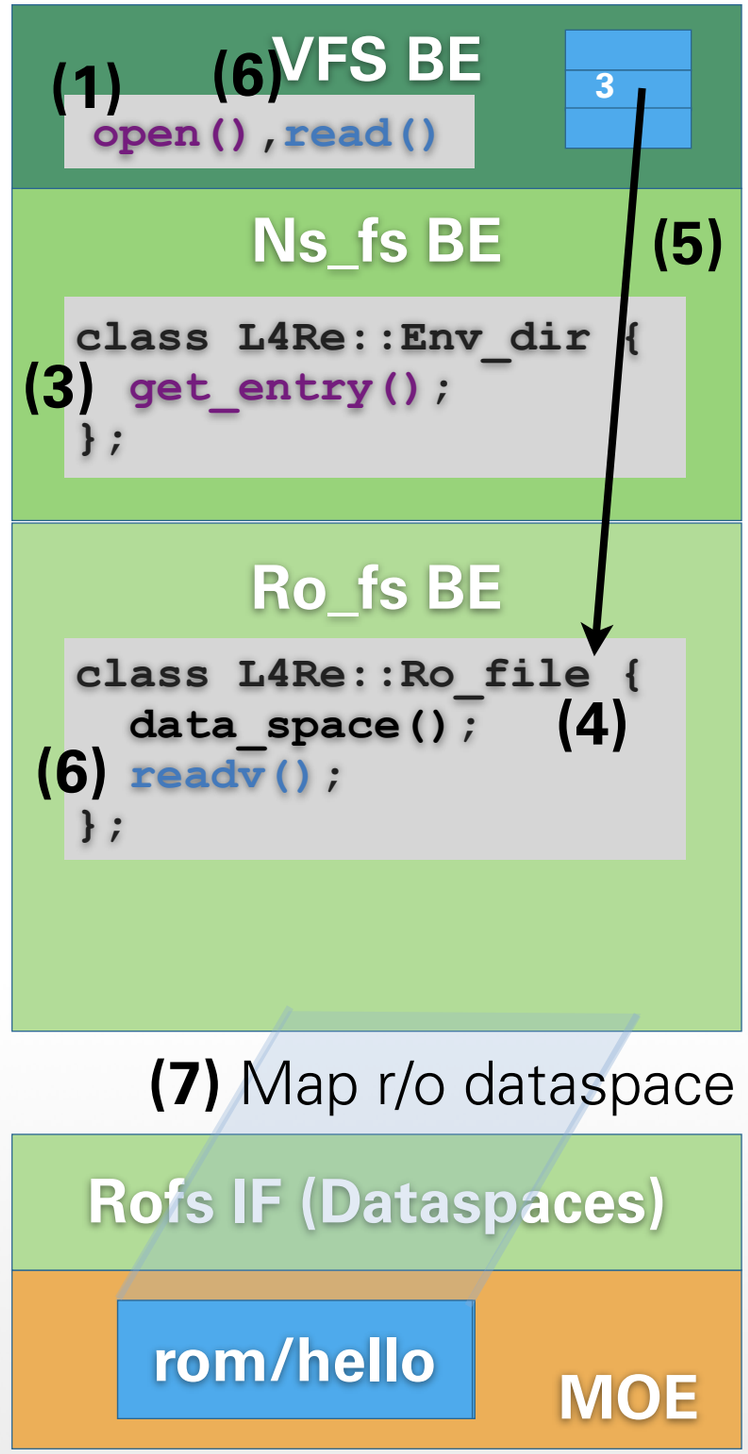
- fprintf() support: easy, just replace write()
- Minimalist backend can output text

```
#include <unistd.h>
#include <errno.h>
#include <l4/sys/kdebug.h>

int write(int fd, const void *buf, size_t count) __THROW
{
  /* just accept write to stdout and stderr */
  if ((fd == STDOUT_FILENO) || (fd == STDERR_FILENO))
    {
      l4kdb_outnstring((const char*)buf, count);
      return count;
    }
  /* writes to other fds shall fail fast */
  errno = EBADF;
  return -1;
}
```

ROM FILE SYSTEM

- (1) Application calls open(„rom/hello“)
- (2) VFS traverses mount tree, finds Ro_fs mounted at „rom“
- (3) VFS asks Ro_fs to provide a file for name „hello“, Ro_fs calls its get_entry() method
- (4) Ro_fs::get_entry() creates new Ro_file object from read-only dataspace provided by MOE
- (5) VFS registers file handle for Ro_file object
- (6) Application calls read(): ends in Ro_file::readv()
- (7) Ro_file::readv() attaches dataspace, copies requested data into read buffer



POSIX: SUMMARY

- L4Re offers most important POSIX APIs
  - C library: strcpy(), ...
  - Dynamic memory allocation:
    - malloc(), free(), mmap(), ...
      - Based on L4Re dataspaces
  - Threads, synchronization: Pthread
  - Signal handling
  - I/O support: files, terminal, time, (sockets)
- POSIX is enabler: sqlite, Cairo, SDL, MPI, ...

TECHNISCHE UNIVERSITÄT DRESDEN

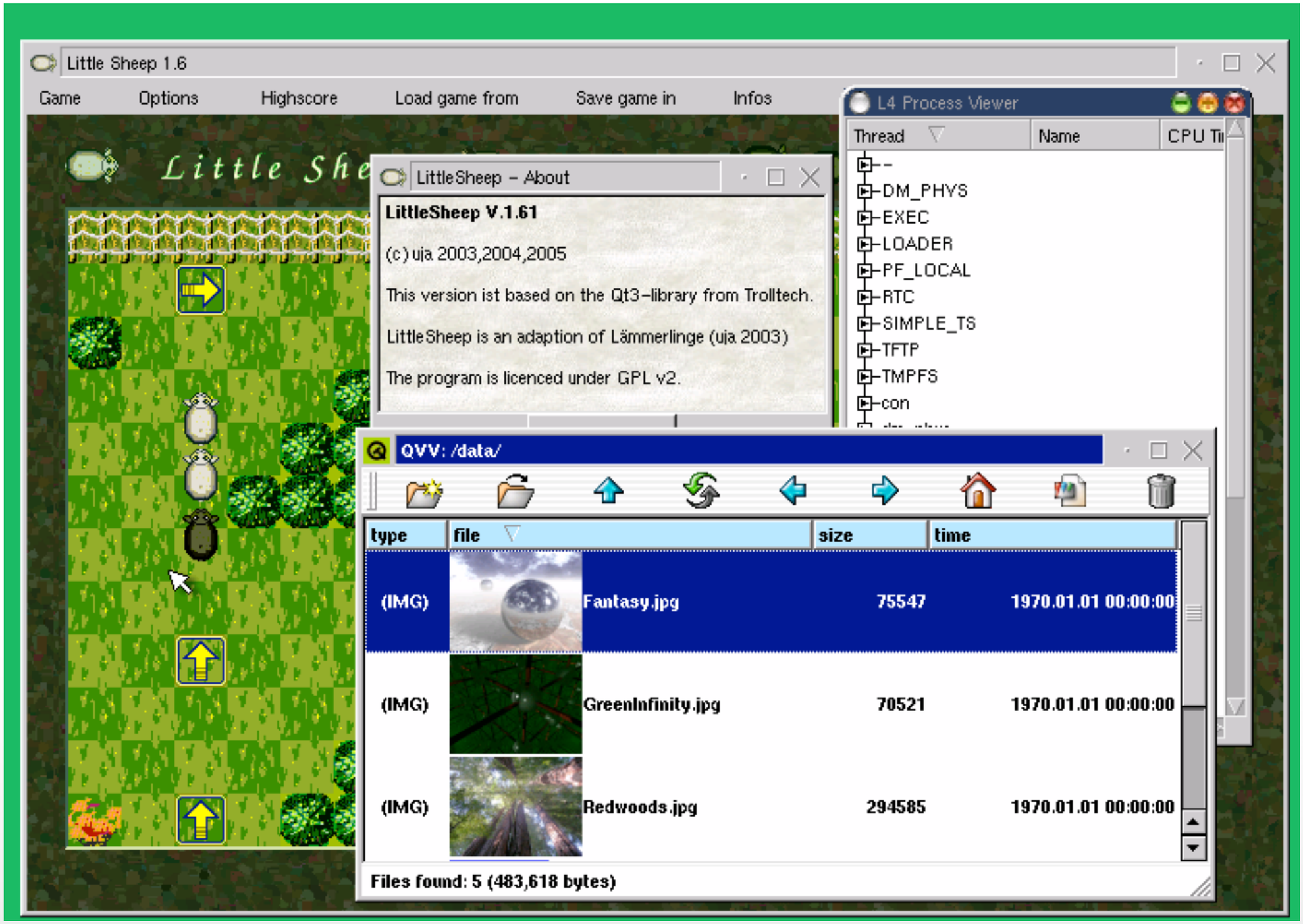
APPLICATION-LEVEL VIRTUALIZATION

RICH FRAMEWORKS

- POSIX is limited to basic OS abstractions
  - No graphics, GUI support
  - No audio support
- Examples for more powerful APIs:
  - SDL „Simple Direct Media Layer“:
    - Multimedia applications and games
  - Qt toolkit:
    - Rich GUIs with tool support
    - Fairly complete OS abstractions

SOFTWARE ON L4

SOFTWARE ON L4



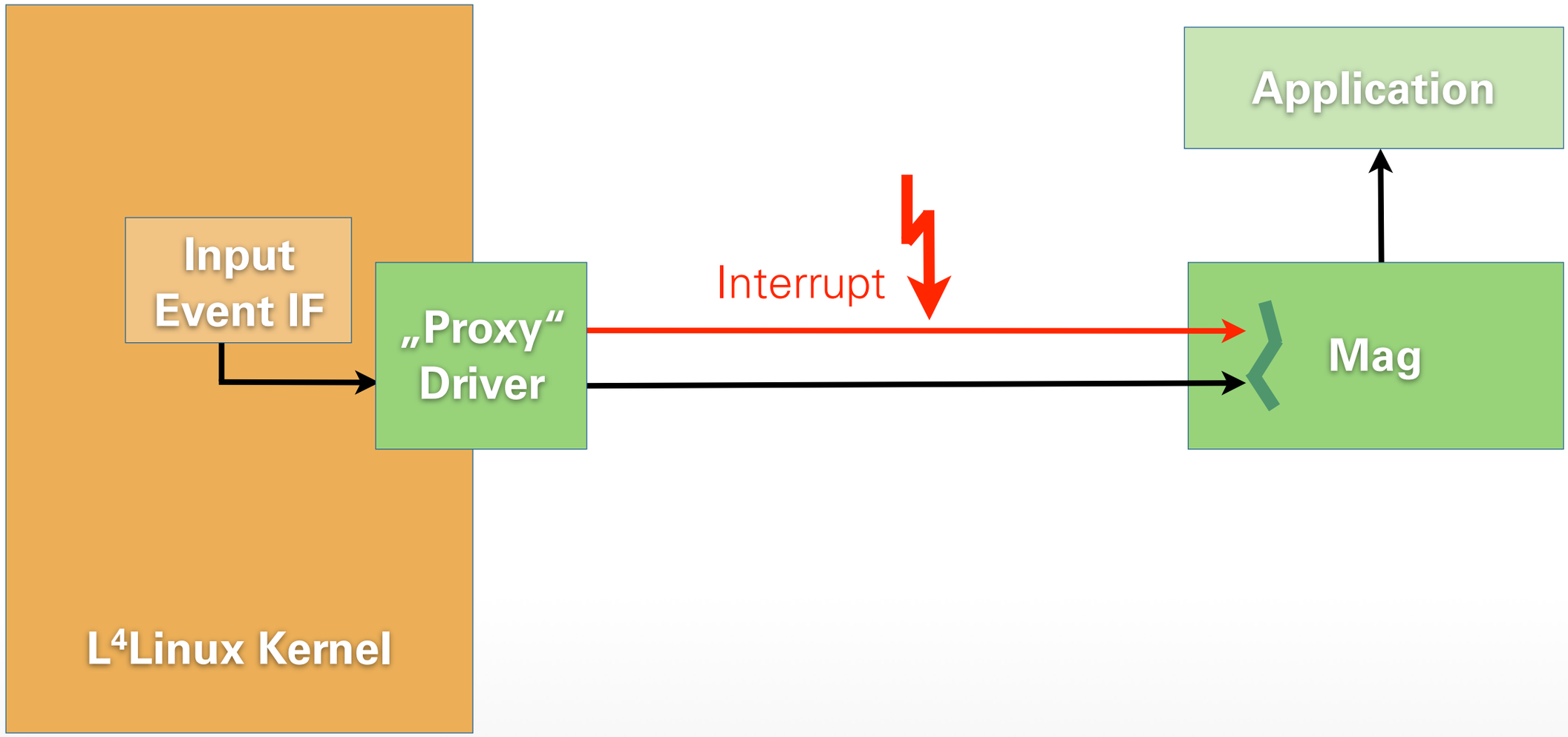
TECHNISCHE UNIVERSITÄT DRESDEN

LEGACY OPERATING SYSTEM AS A TOOLBOX

LINUX TOOLBOX

- Applications are nice, but there's more ...
- Legacy OSes have lots of:
  - Device drivers
  - Protocol stacks
  - File systems
- Reuse drivers in natural environment
  - Also see paper: *„Unmodified Device Driver Reuse and Improved System Dependability via Virtual Machines“* , by LeVasseur, Uhlig, Stoess, Götz)
- L4Linux:
  - **Hybrid applications:** access legacy OS + L4Re
  - **In-kernel support:** bridge Linux services to L4Re

GENERAL IDEA

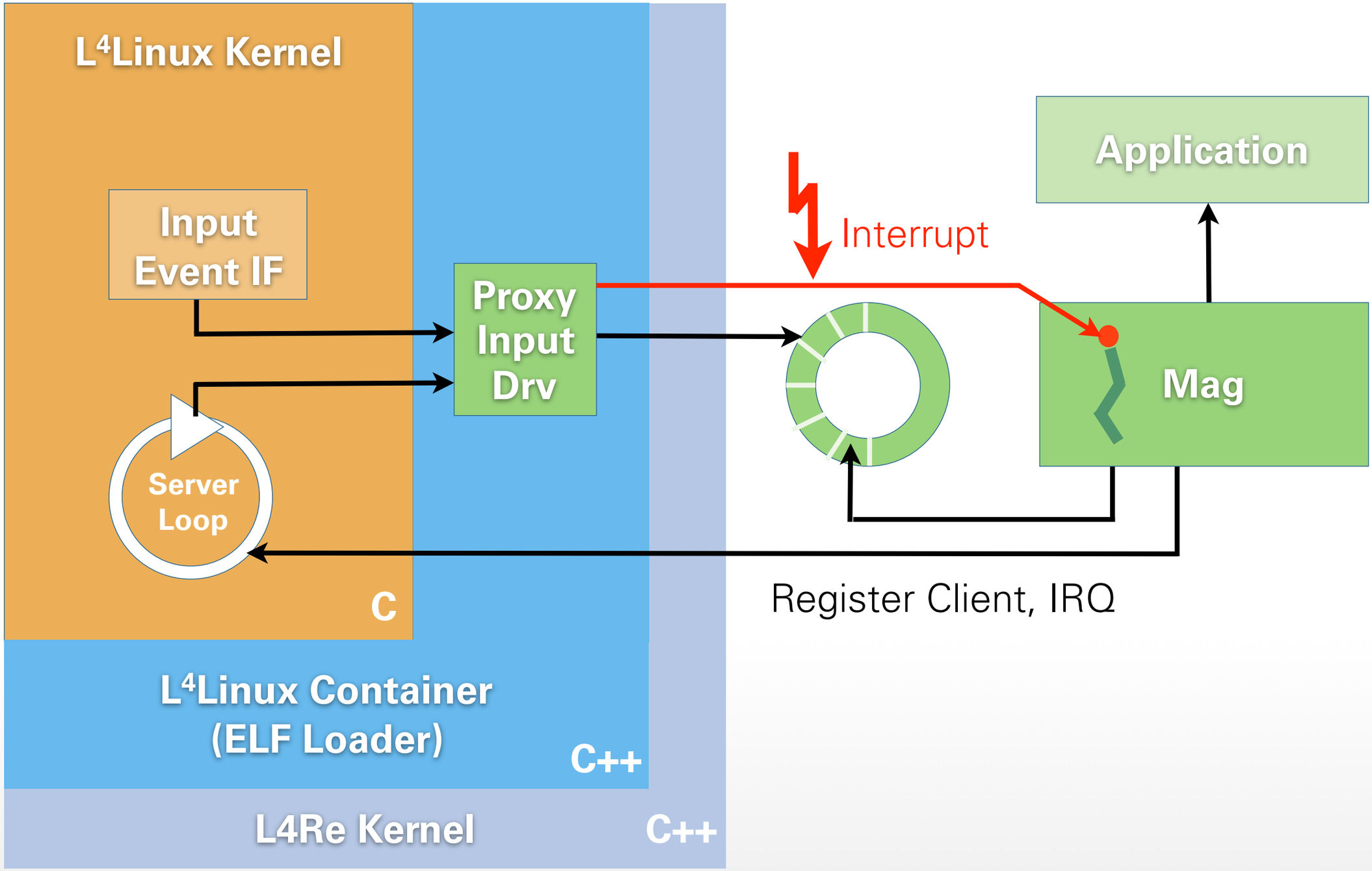


ACCESS L⁴LINUX

- L⁴Linux has drivers
- L4Re has great infrastructure for servers:
  - IPC framework
  - Generic server loop

- **Problems:** C vs. C++, symbol visibility
- **Bridge:** allow calls from L⁴Linux to L4Re
  - L4Re exports C functions to L⁴Linux
  - L⁴Linux kernel module calls them

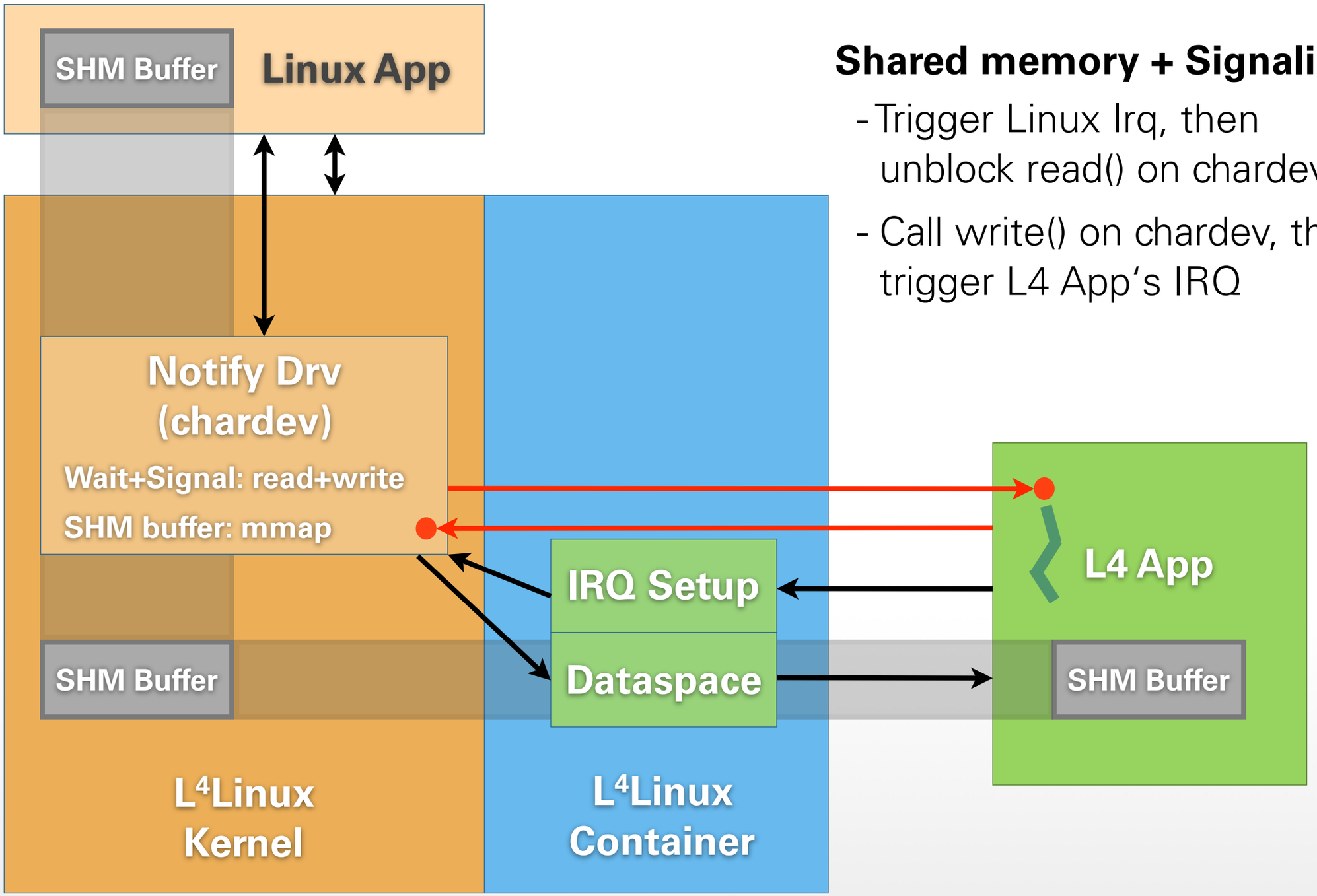
INPUT DRIVER



PRISON BREAK

- **Idea:** „enlightened“ applications
  - Know that they run on L4Re
  - Talk to L4Re servers via guest OS
- **Proxy driver** in guest provides:
  - Shared memory: Linux app + L4Re server
  - Signaling: Interrupt objects
  - Enables synchronous and asynchronous zero-copy communication (e.g., ring buffer)

PROXY DRIVER



Shared memory + Signaling:

- Trigger Linux Irq, then unblock read() on chardev
- Call write() on chardev, then trigger L4 App's IRQ

SPLIT SERVICES

- Proxy driver suitable for many scenarios:
  - Producer/consumer (either direction)
  - Split applications:
    - Reuse application on either side
    - Trusted / untrusted parts
  - Split services:
    - Block device / file system / database / ...
    - Network stack
  - Split device drivers

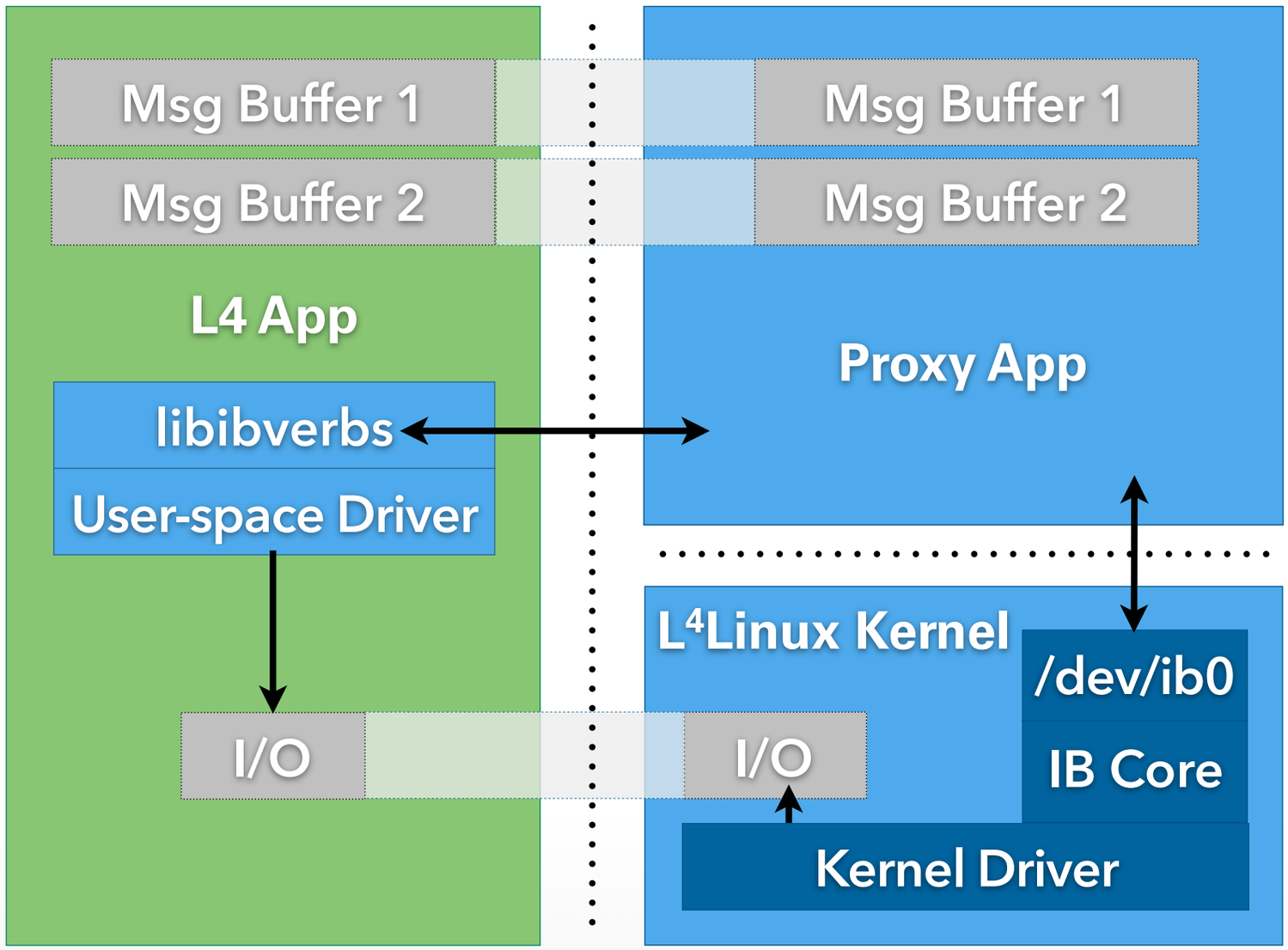
INFINIBAND

InfiniBand Stack:

- Kernel driver
- User-space driver
- Generic verbs interface

Proxy process:

- Forwards calls to kernel driver on behalf of user-space driver on L4
- Maps message buffers



TECHNISCHE UNIVERSITÄT DRESDEN

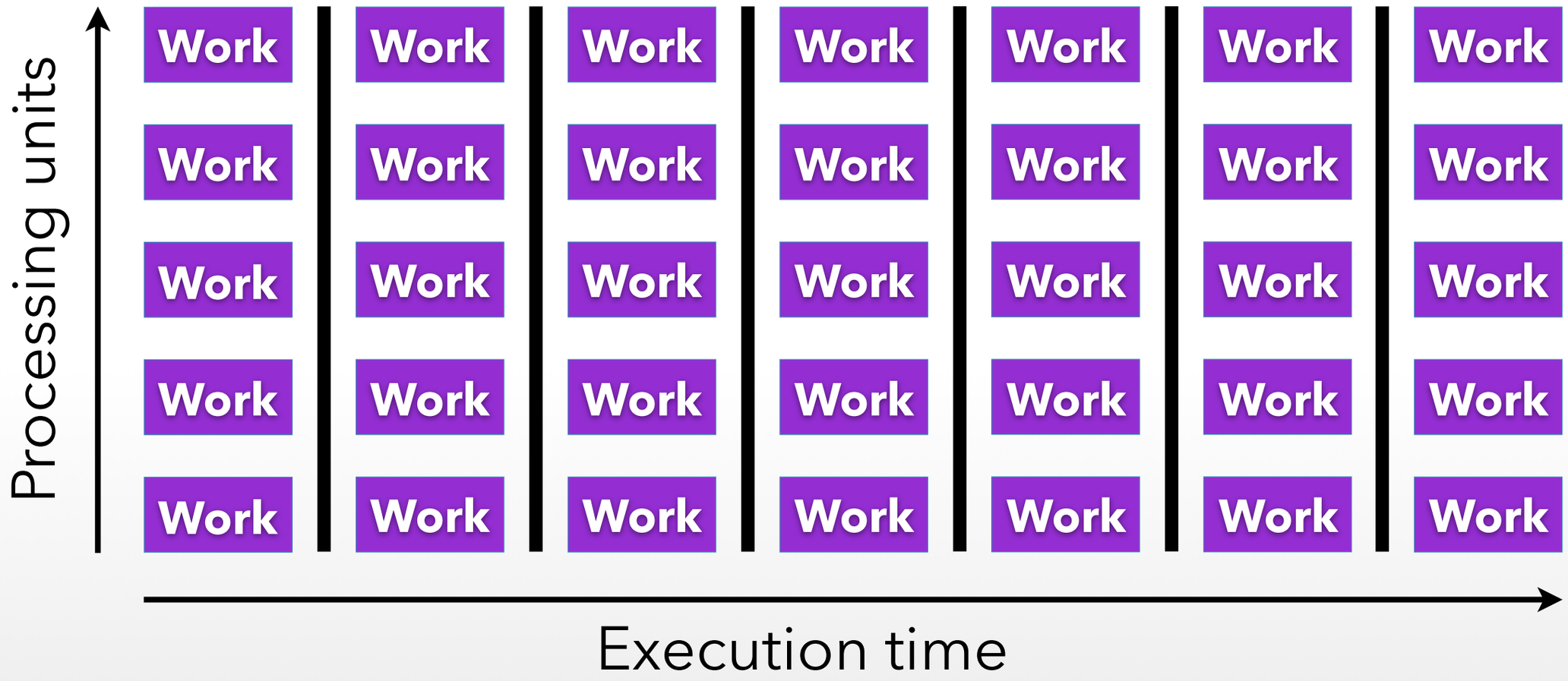
HYBRID OPERATING SYSTEMS

WHY HYBRID OS?

- **Problem:**
  - Some applications need a lot of functionality from a legacy OS like Linux …
  - … and a few strong guarantees that Linux cannot provide due to its complexity
- **Examples:**
  - Security-critical applications
  - Real-time & high-performance computing
- **Solution:** Combine Microkernel and Linux

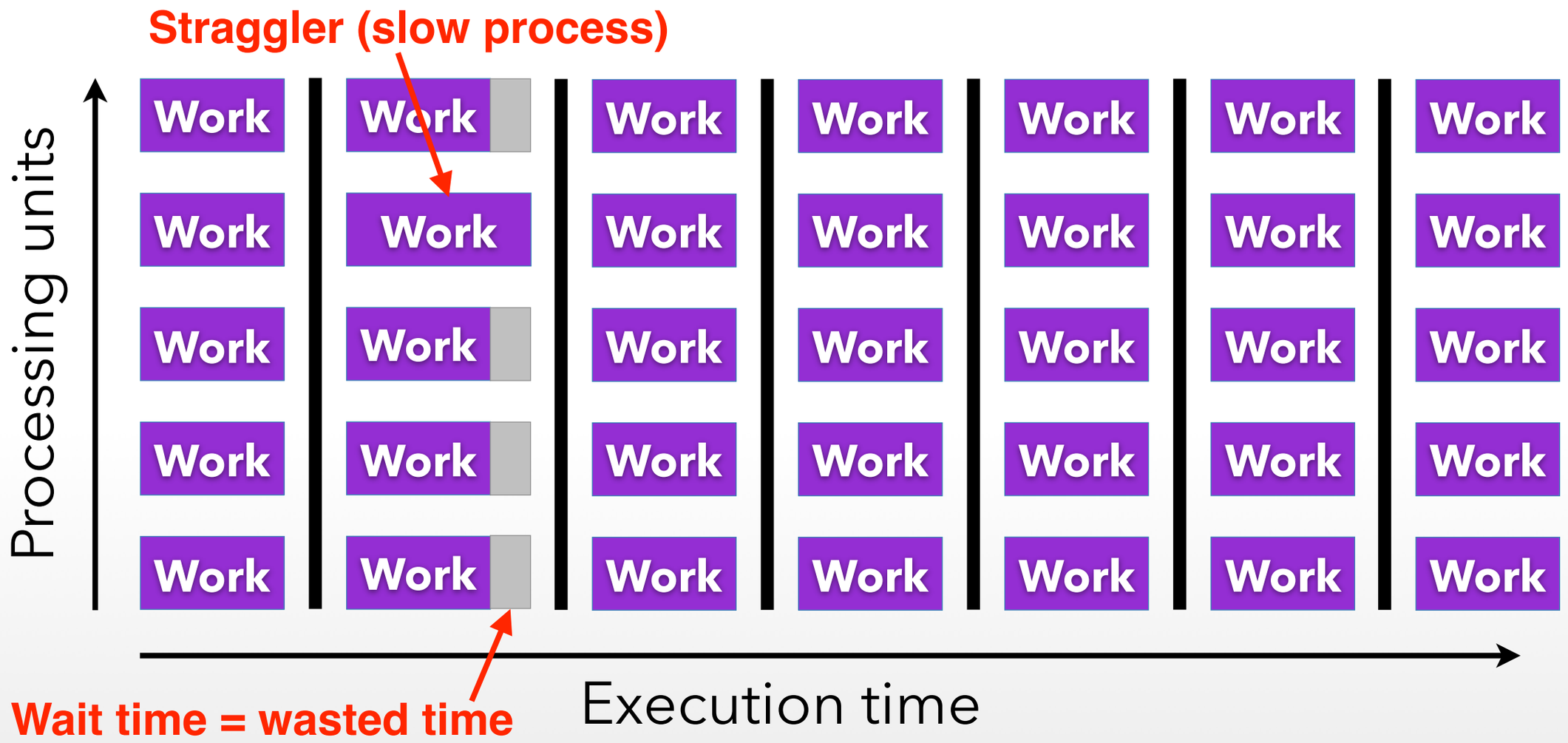
NO DELAYS!!!

- Real-time: Prevent deadline miss
- Bulk-synchronous programs: Avoid straggler



NO DELAYS!!!

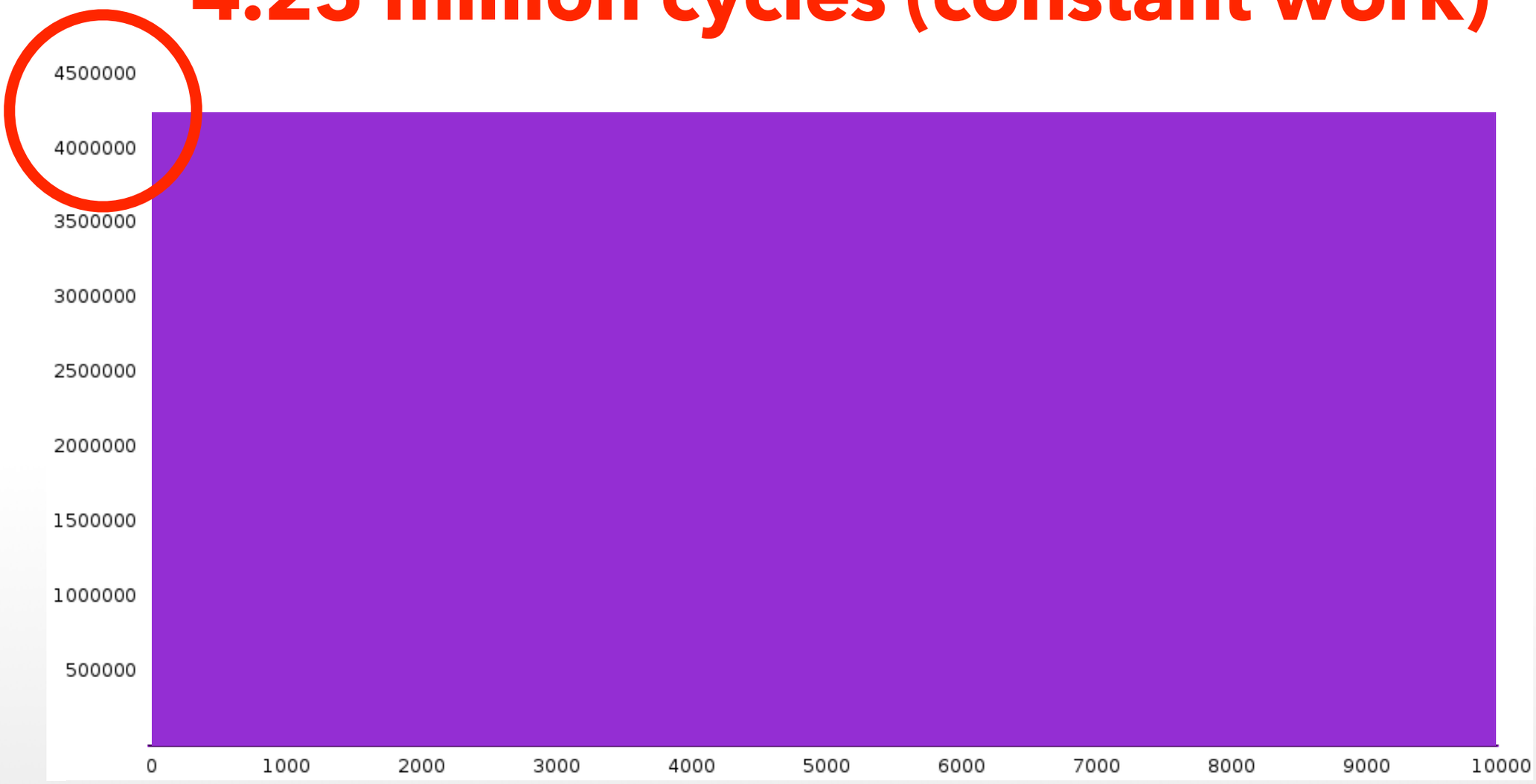
- Real-time: Prevent deadline miss
- Bulk-synchronous programs: Avoid straggler



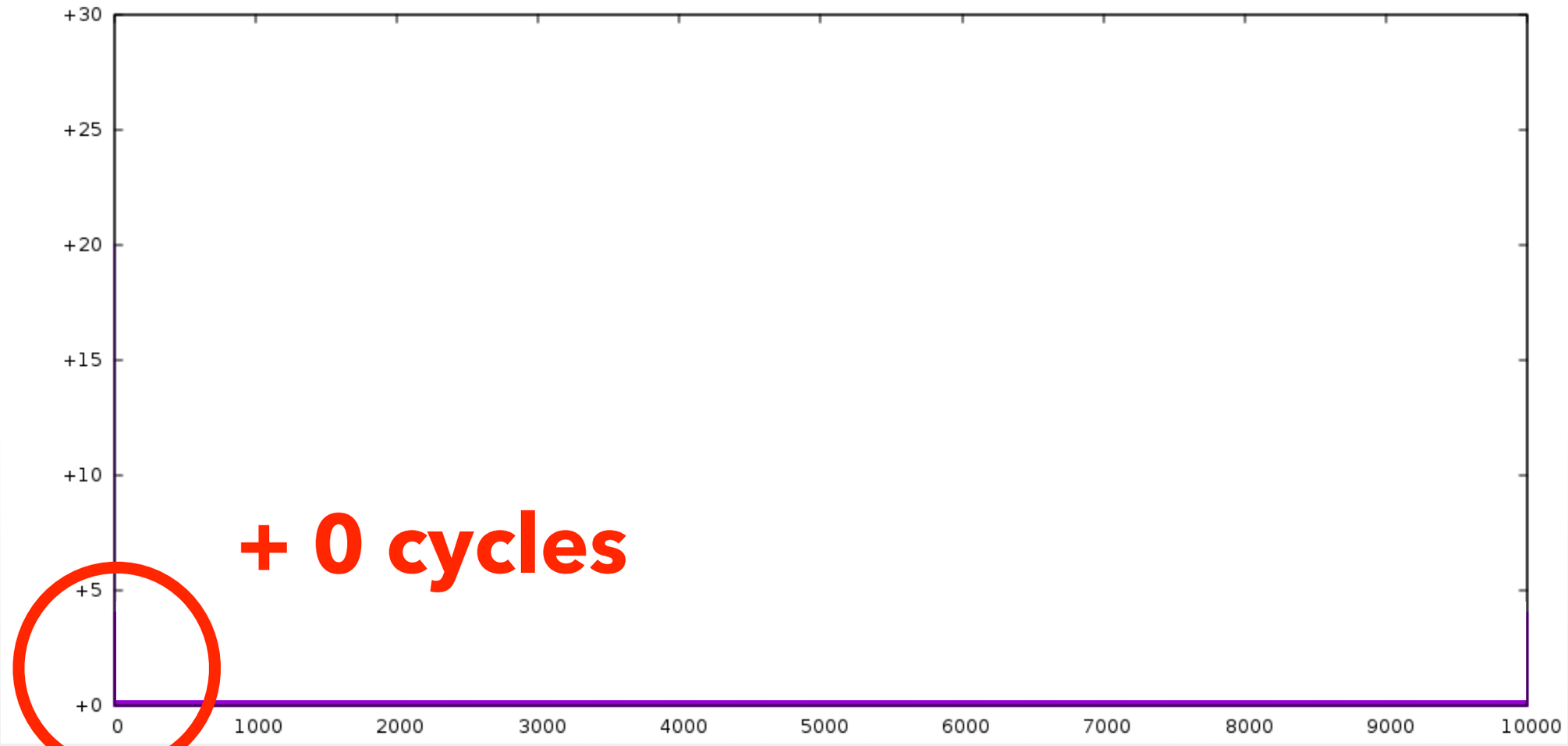
FWQ BENCHMARK

Fixed work quantum (FWQ): repeatedly measure execution time for same work

4.25 million cycles (constant work)

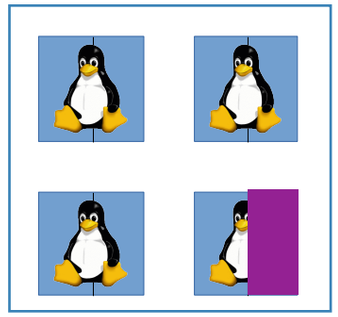


Ideal: zero extra cycles

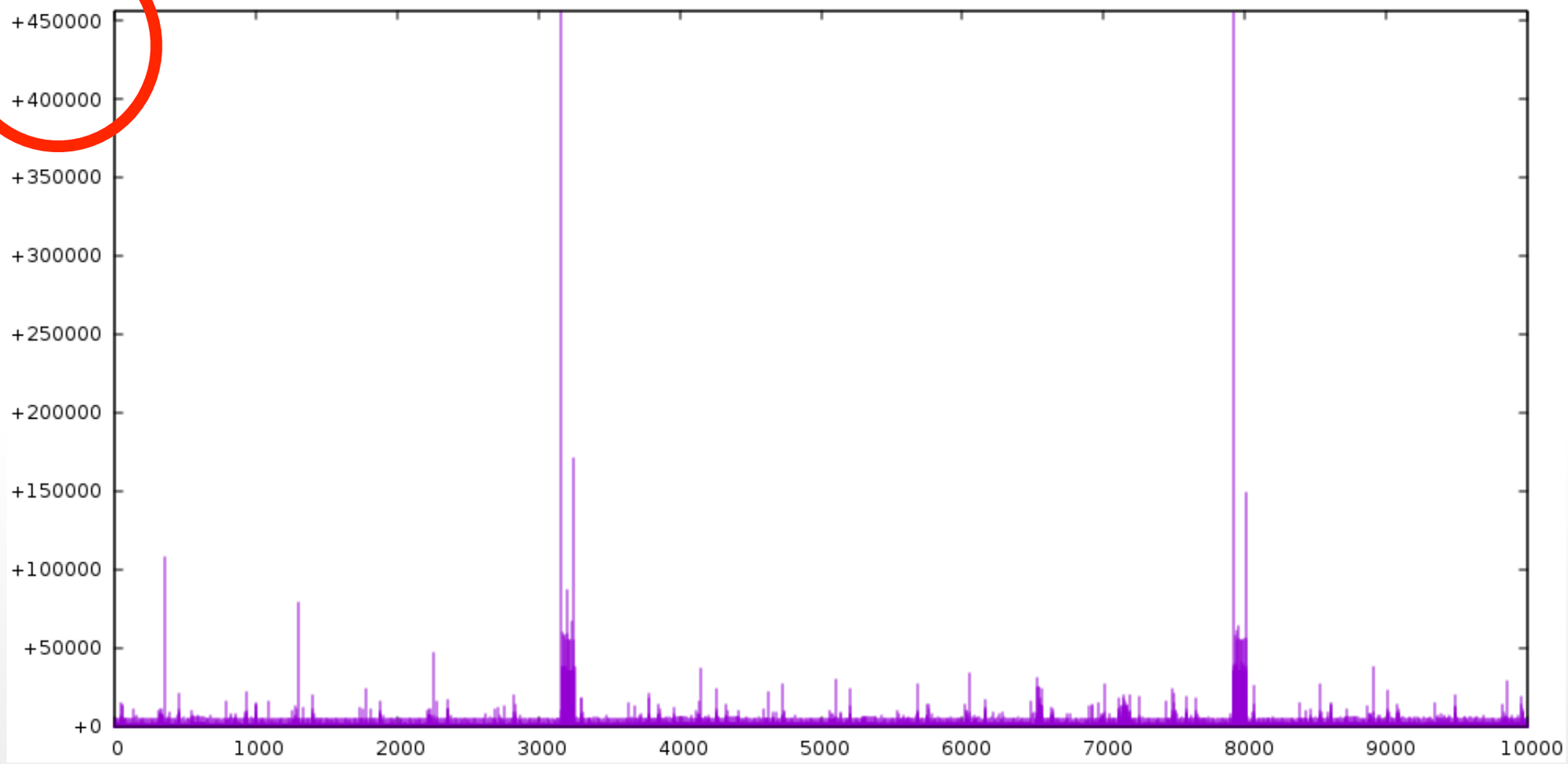


NOISY LINUX?

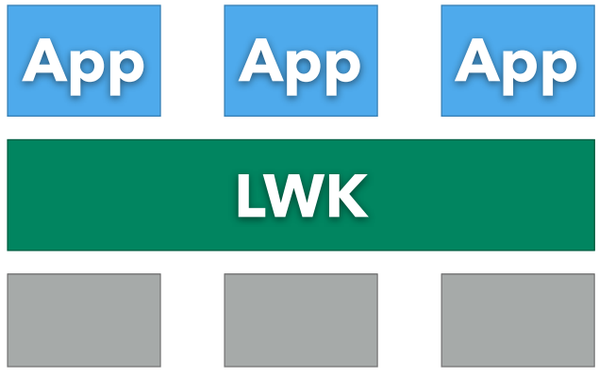
Real-World HPC Linux



+450,000 cycles ≈ 10%



APPROACHES

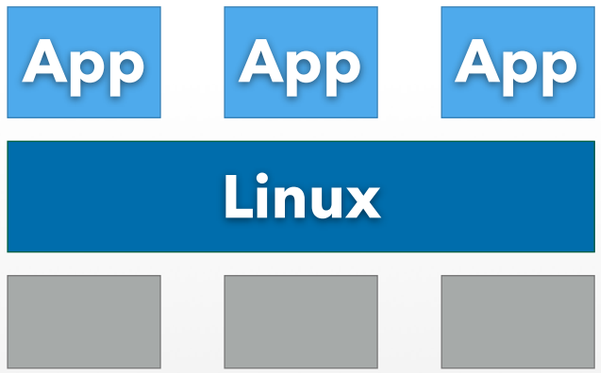


Light-Weight Kernel (LWK)

⊕ No Noise

⊖ Compatibility

⊖ Features



Tweaked Linux

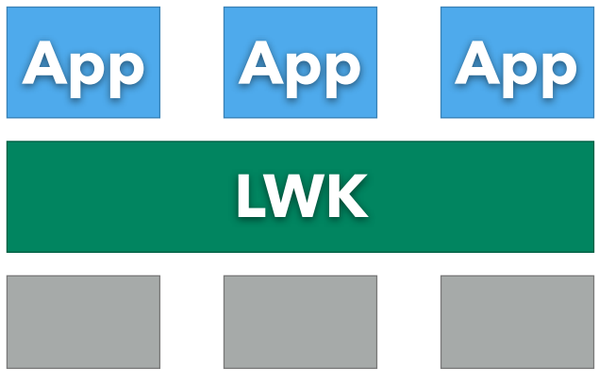
⊙ Low Noise

⊕ Compatibility

⊕ Features

⊖ Fast moving target

APPROACHES

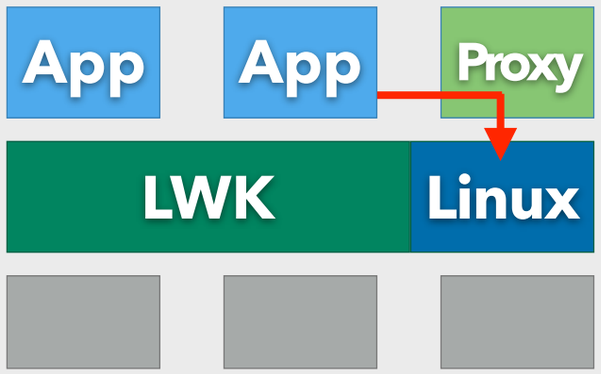


Light-Weight Kernel (LWK)

⊕ No Noise

⊖ Compatibility

⊖ Features



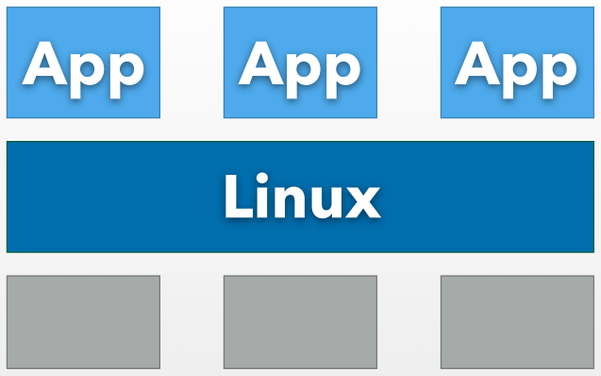
Light-Weight Kernel + Linux

⊕ No Noise

⊕ Compatibility

⊕ Features

⊖ **Much effort? Not if we can reuse a lot …**



Tweaked Linux

⊙ Low Noise

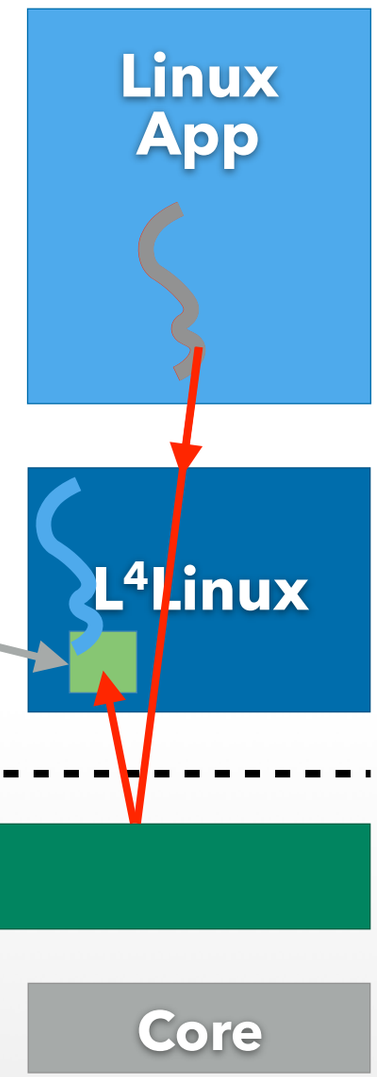
⊕ Compatibility

⊕ Features

⊖ Fast moving target

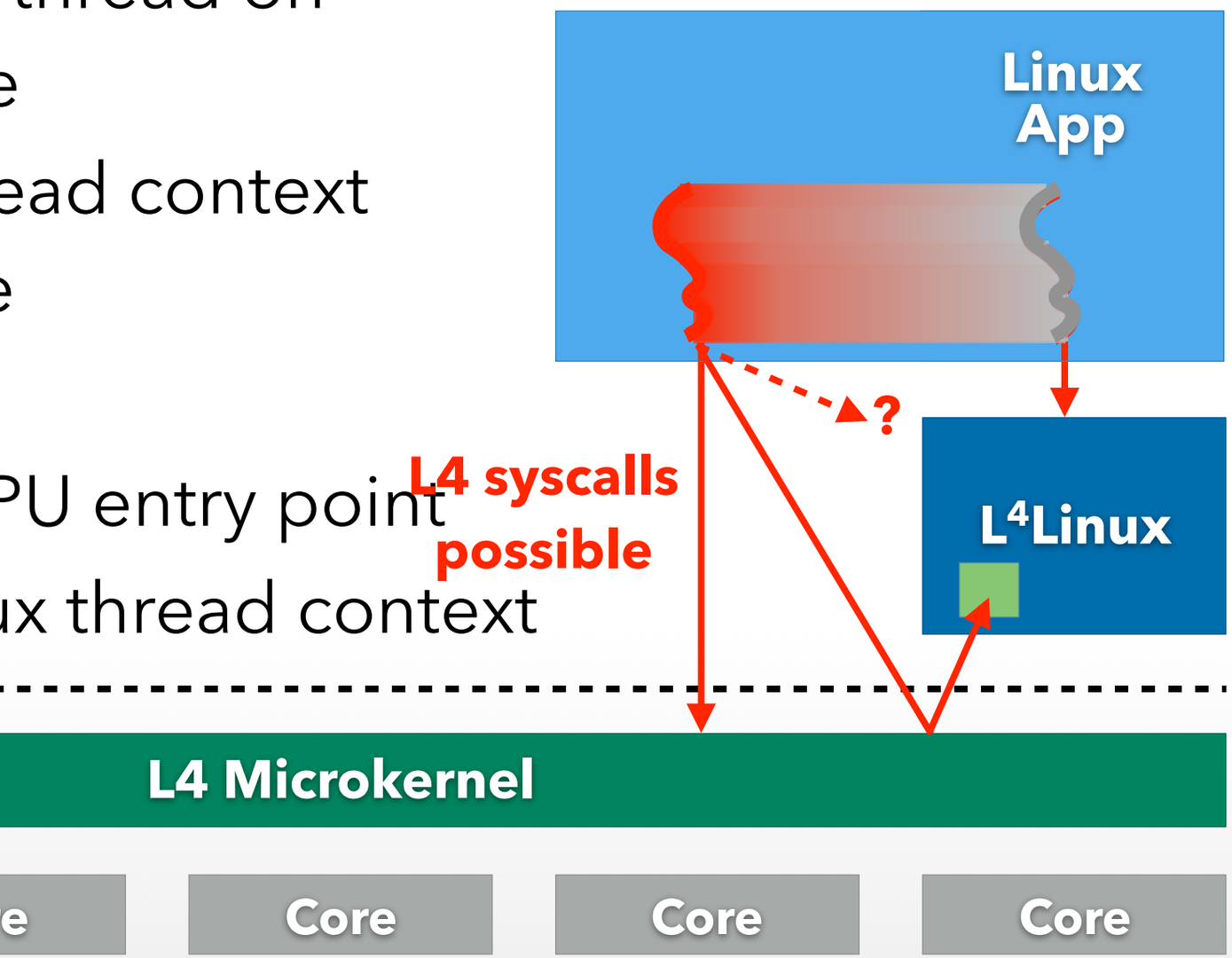
L4 + L⁴LINUX

- L⁴Linux is paravirtualized: **`arch/l4`**
- Tight integration with L4 microkernel
- Linux processes are L4 Tasks
- Threads multiplexed onto vCPU
- Linux syscalls / exceptions: reflected to vCPU entry point
- Handle syscall + resume user thread

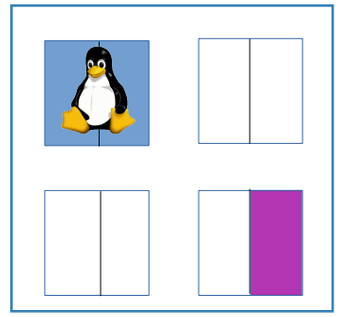


DECOUPLING

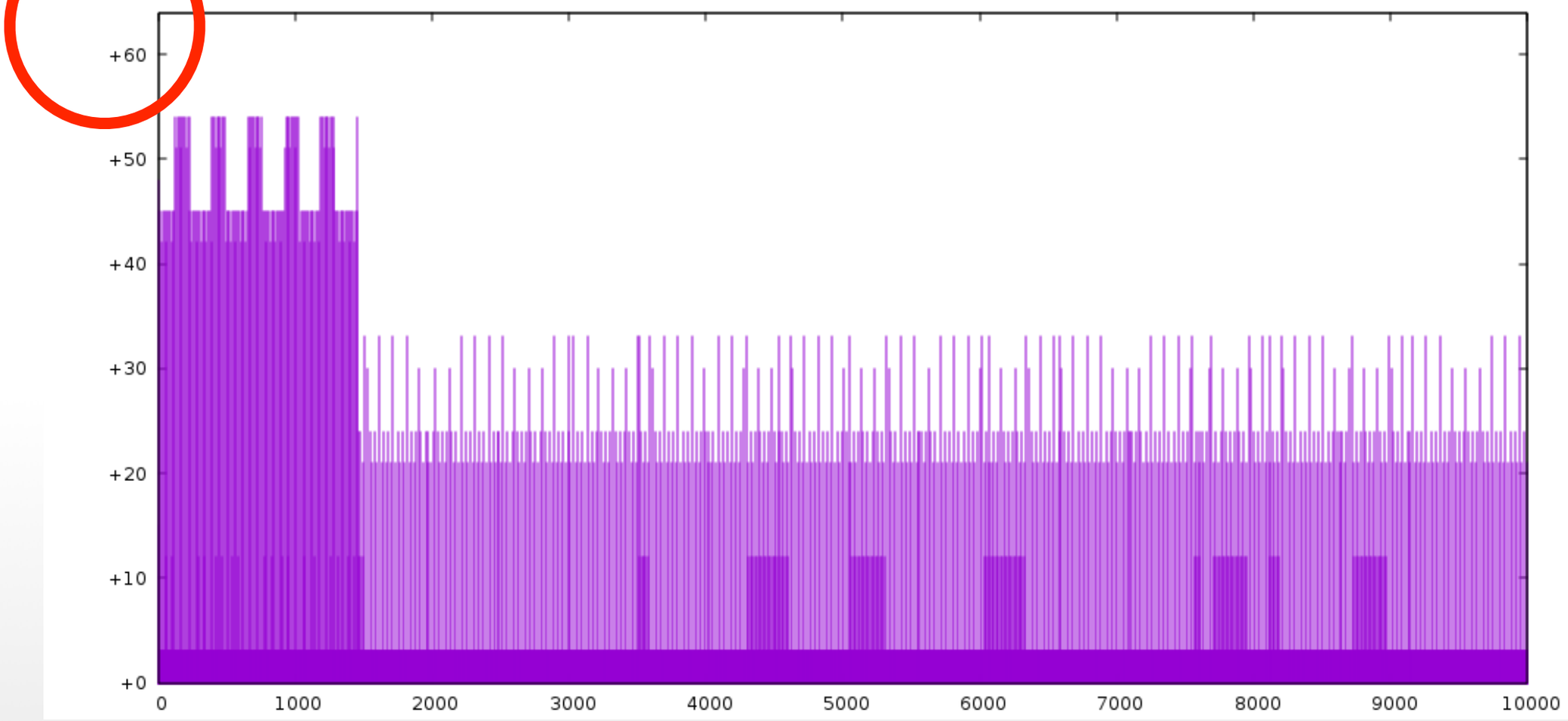
- Decoupling:
  - Create new L4 thread on dedicated core
  - Mark Linux thread context uninterruptible
- Linux syscall:
  - Forward to vCPU entry point
  - Reactivate Linux thread context



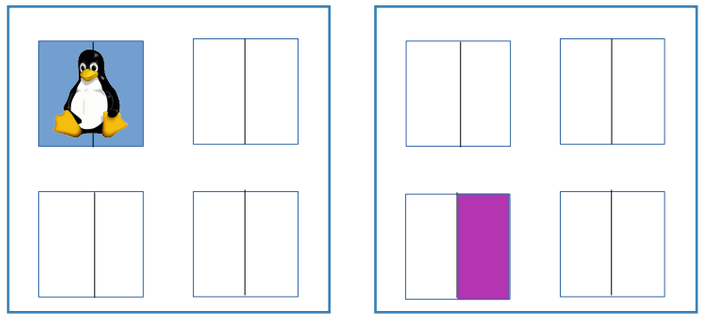
Decoupled Linux thread

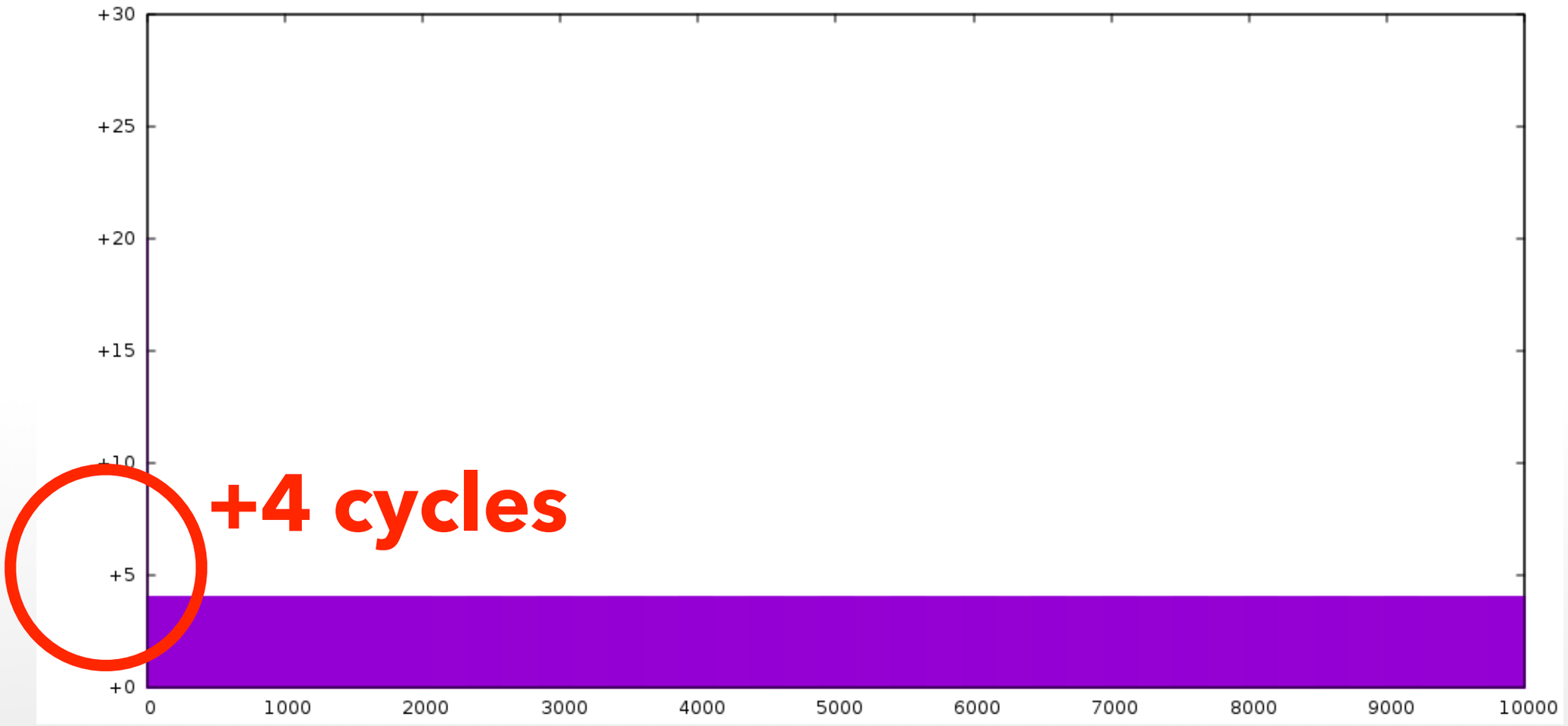


+60 cycles



Decoupled Linux thread

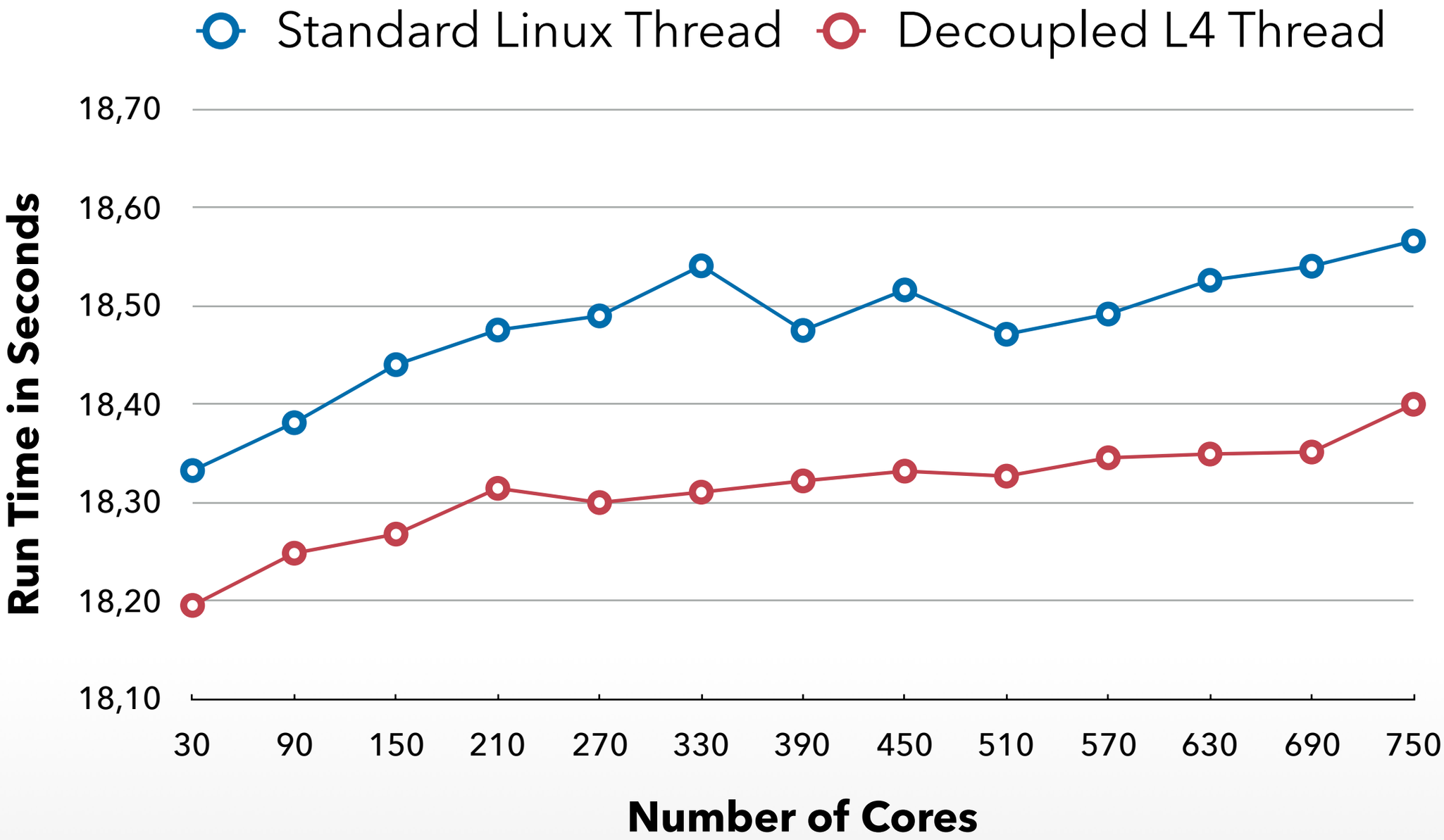




MPI-FWQ

- **MPI-FWQ:**
  - Simulates bulk-synchronous high-performance application
  - Alternates between: constant work on each processor and global barrier (wait-for-all)

MPI-FWQ RESULT



[1] „Decoupled: Low-Effort Noise-Free Execution on Commodity Systems", Adam Lackorzynski, Carsten Weinhold, Hermann Härtig, ROSS'16, Kyoto, Japan

UP NEXT ...

Next week:

- **Lecture:** *„Virtualization“*
- **No exercise**

REFERENCES

- [1] **„Decoupled: Low-Effort Noise-Free Execution on Commodity Systems“**, Adam Lackorzynski, Carsten Weinhold, Hermann Härtig, Runtime and Operating Systems for Supercomputers (ROSS 2016), Kyoto, Japan, June 2016

- [2] Resources on POSIX standard: *http://standards.ieee.org/regauth/posix/*

- [3] **„Unmodified Device Driver Reuse and Improved System Dependability via Virtual Machines“**, by J. LeVasseur, V. Uhlig, J. Stoess, S. Götz, OSDI 2004