



TECHNISCHE
UNIVERSITÄT
DRESDEN

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

LEGACY REUSE

CARSTEN WEINHOLD

- **So far ...**

- Basic concepts, resources, ...

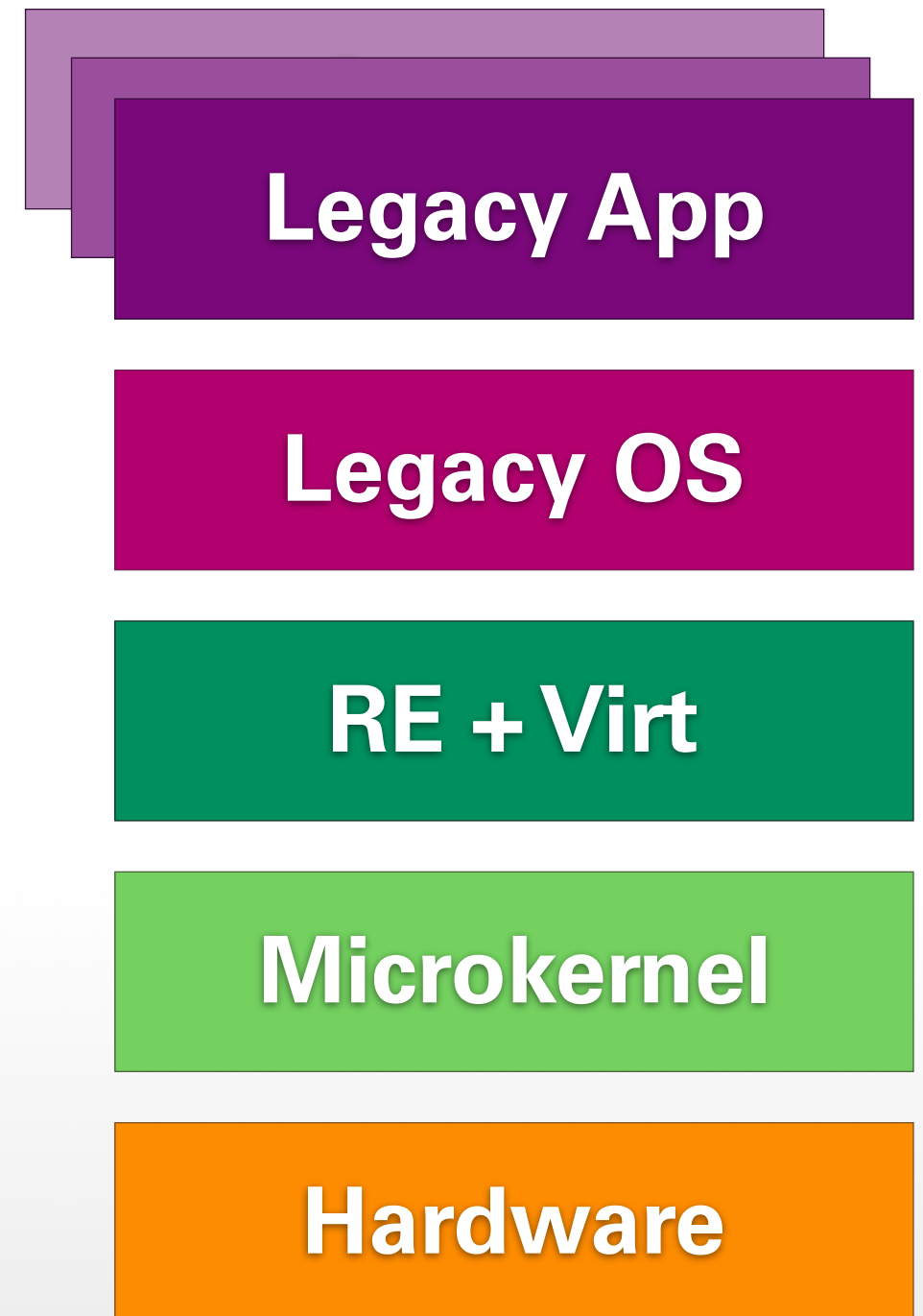
- ~~Virtualization~~ **Next week**

- **Today:**

- Operating System Personalities
- How to reuse existing infrastructure
- How to make applications happy

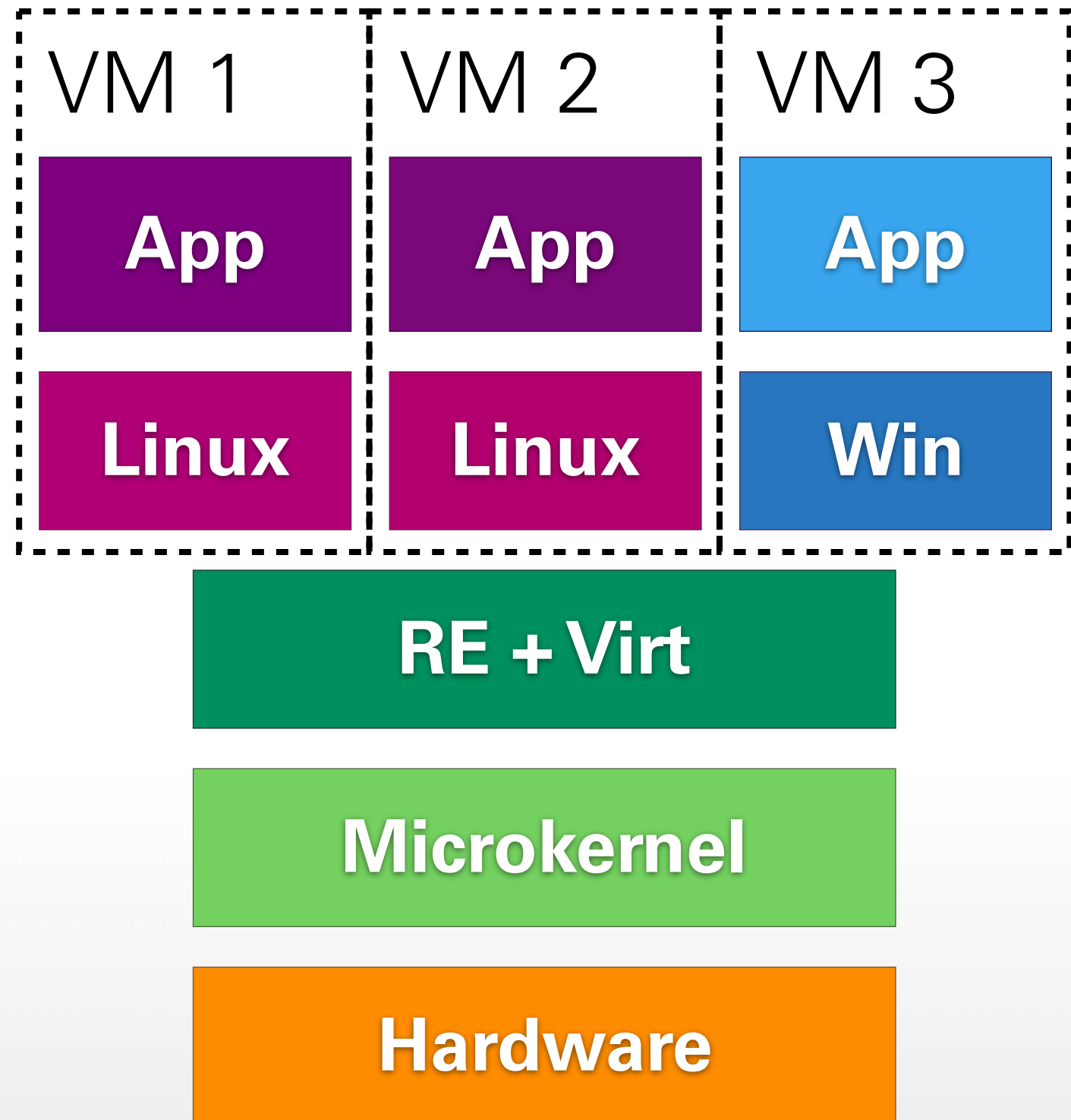
Getting applications the easy way:

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



Multiple VMs:

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



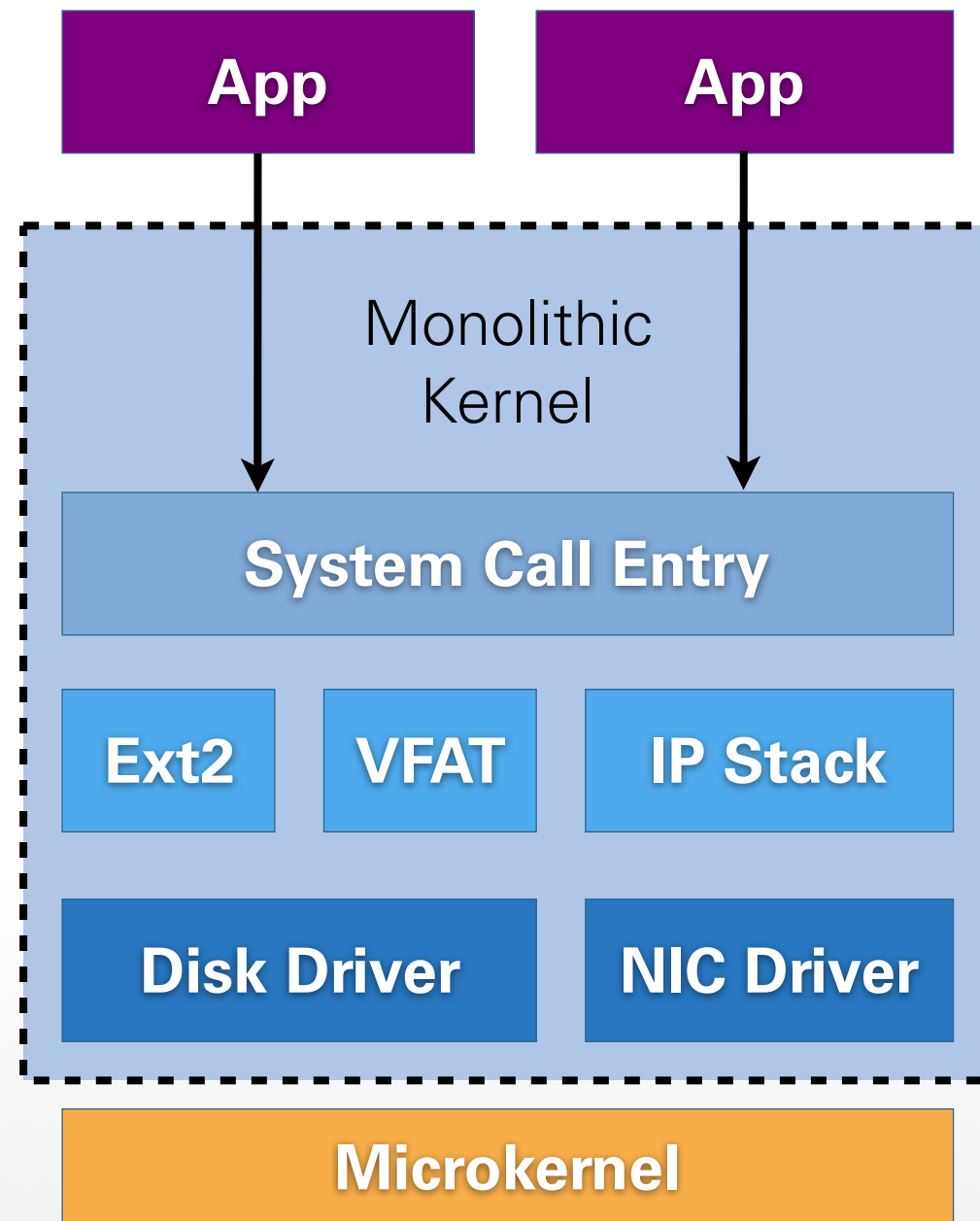
- **Virtualization:**
 - Reuses legacy OS + applications
 - Applications run in their natural environment
- **Problem:** Applications trapped in VMs
 - Different resource pools, namespaces
 - Cooperation is cumbersome (network, ...)
 - Full legacy OS in VM adds overhead
 - Management overhead, multiple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

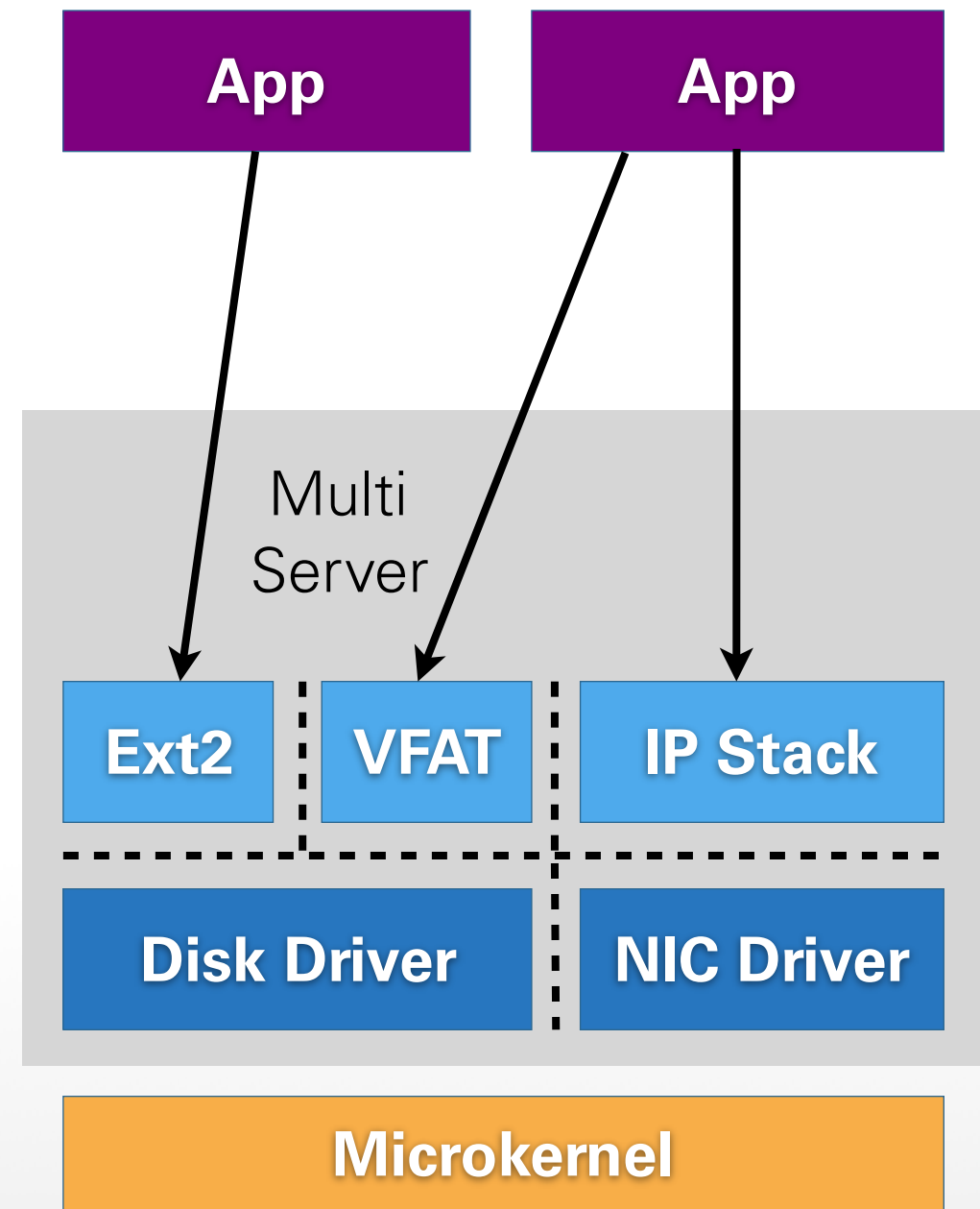
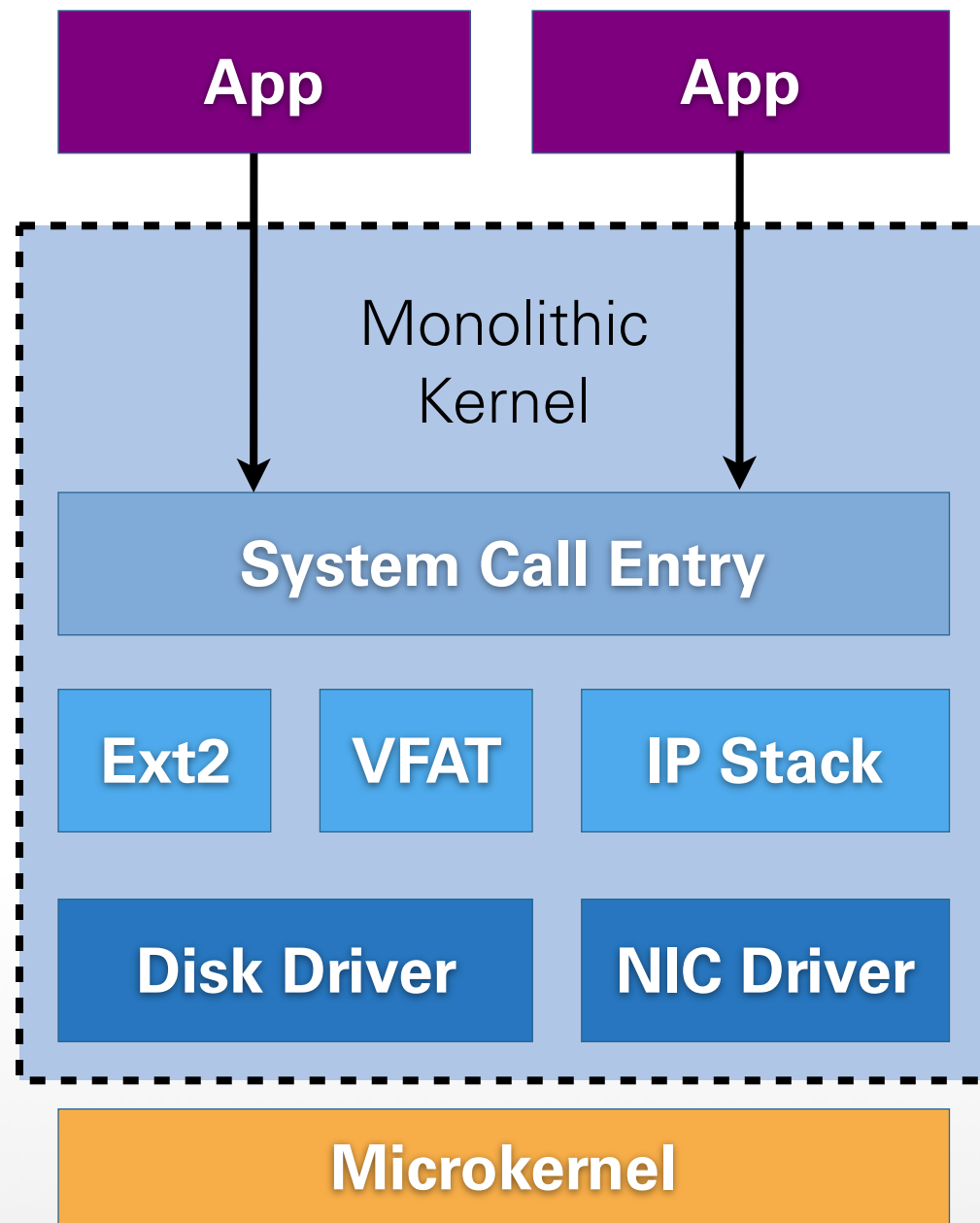
- **Hardware level** (virtualization)
 - Legacy OS + applications on top of new OS
- **Operating system level** (e.g., Wine)
 - Legacy OS's interfaces reimplemented on top of new OS
- **Application level**
 - Applications, Toolkits, frameworks, libraries ported to new OS

OPERATING SYSTEM PERSONALITIES

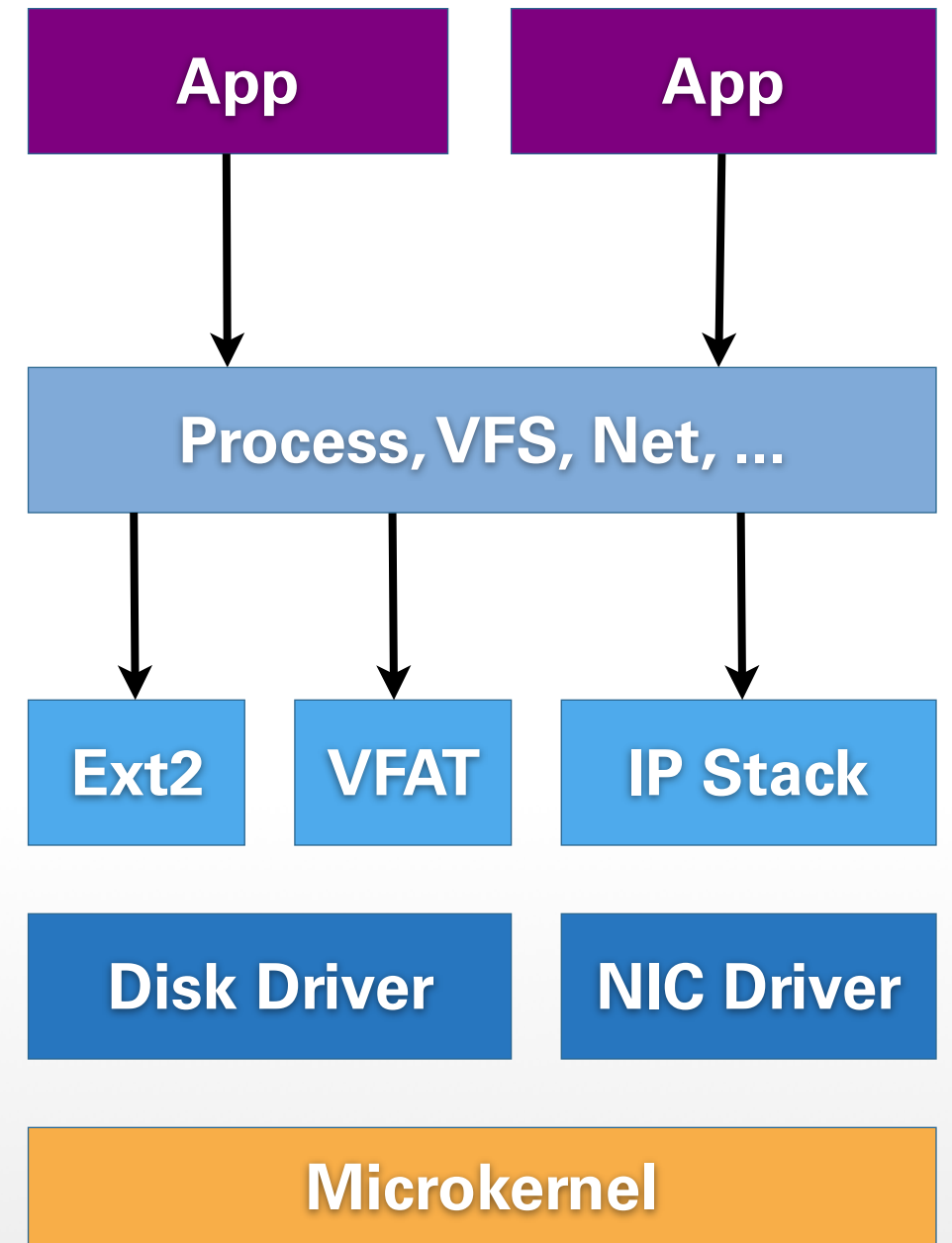
- **Idea:** Adapt at OS / application boundary
 - (Re-)Implement legacy APIs, not whole OS
 - May need to recompile application
- **Benefits:**
 - Get desired application, established APIs
 - More flexible, configurable, easier to achieve
 - Better integration (namespaces, files, ...)

SINGLE SERVER

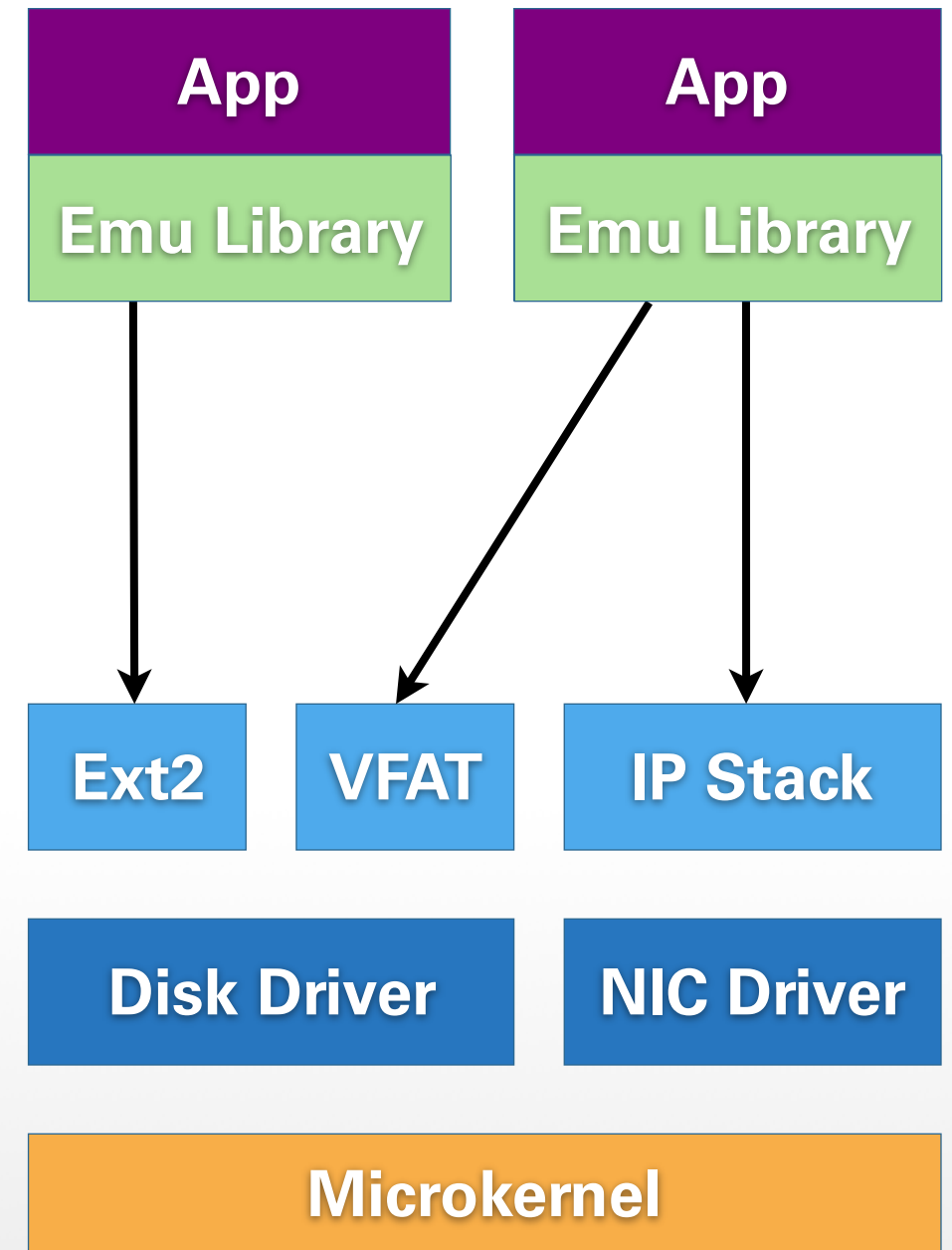




- Central server provides consistent view for both:
 - **Servers:** client state (e.g., file tables)
 - **Applications:** system resources (e.g., files)
- Potential issues:
 - Scalability
 - High complexity
 - Single point of failure



- Emulation library:
 - Linked into applications
 - Interacts with servers
 - Provides consistent view
- Each server keeps its own client state
- **Compatibility:** emulation library hidden below libc



- **What to do:**

- Determine what application needs
- Provide the needed APIs

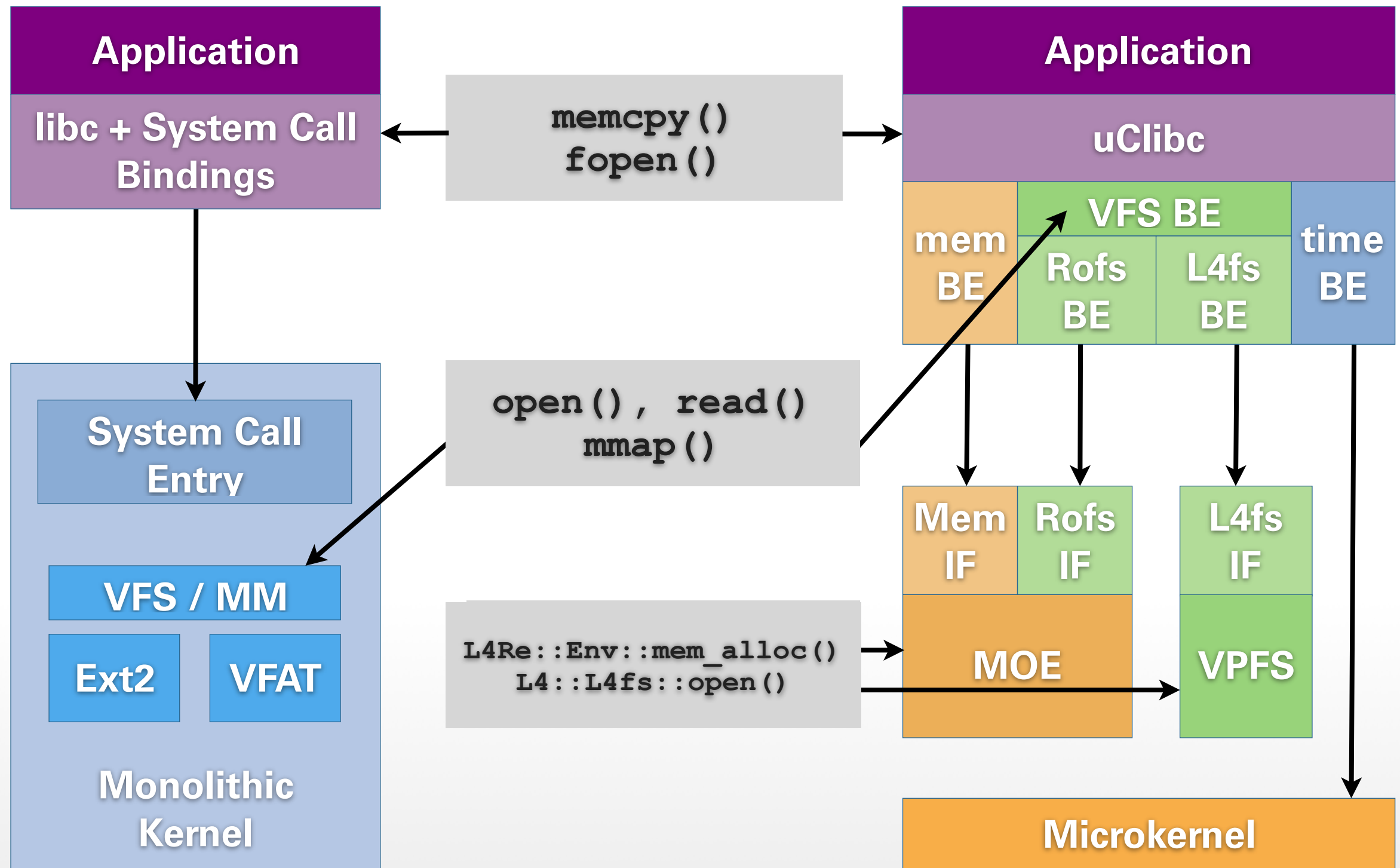
- **Approach:**

- Do not reinvent the wheel!
- Reuse libraries, map to existing APIs
- Better compatibility, more complete
- Make everything modular

- „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

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



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;
}
```

Example 2: memory management

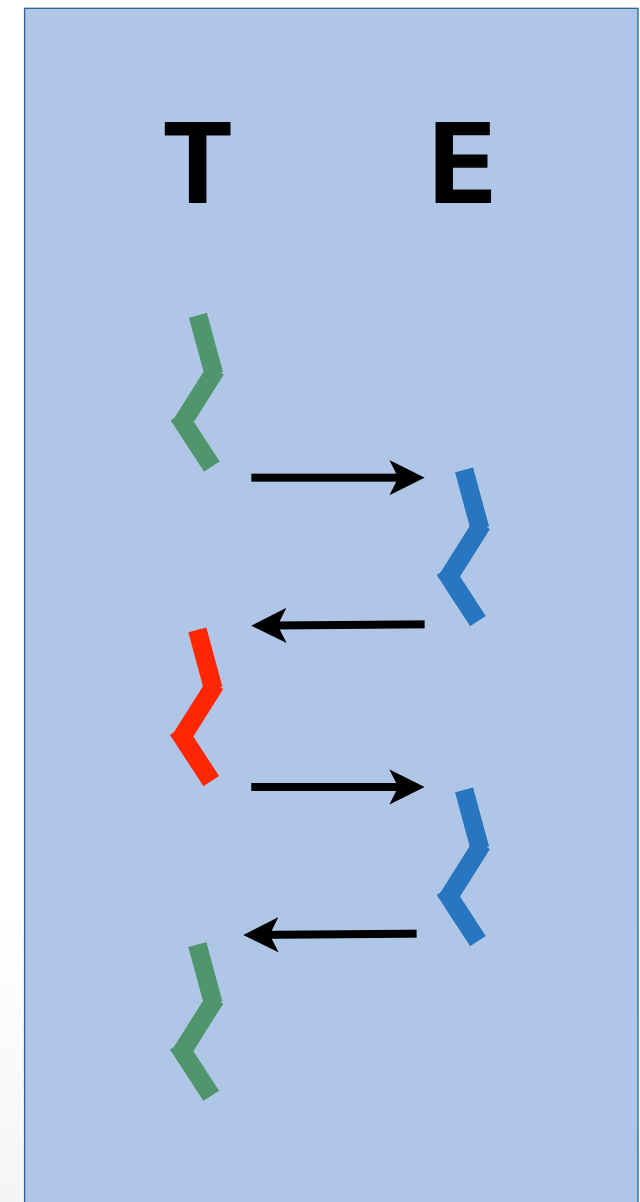
- uClibc implements heap allocator
- Requests memory pages via `mmap()`
- Can be reused, if we provide **`mmap()`**
 - Minimalist: use static pages from BSS
 - `l4re_file`:
 - Supports **`mmap()`**, **`munmap()`** for anon memory
 - Based on dataspace and L4Re region manager
 - Usually gets memory from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

- **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

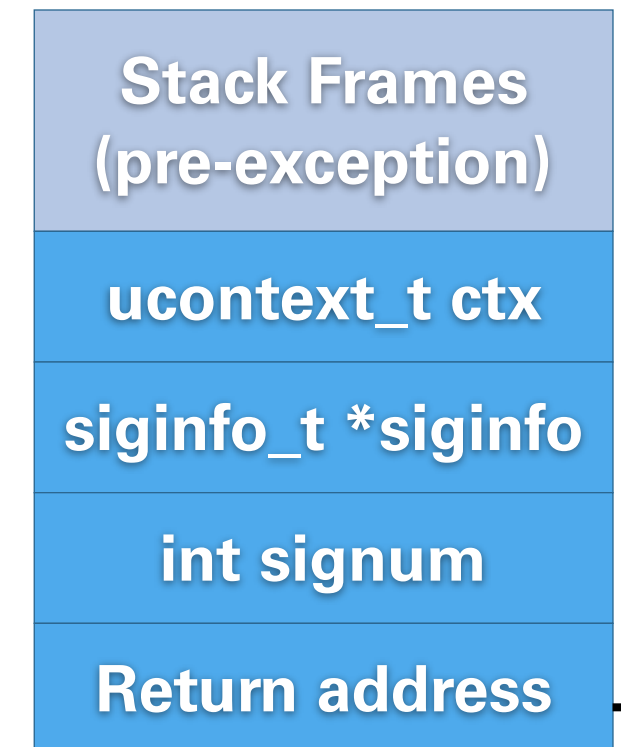
Example 3: POSIX signals

- 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?**

- 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



- **E**: handles exceptions:
 - Set up signal handler context:
 - Save **T**'s context
 - Push pointer to `siginfo_t`, signal number
 - Push address of return trap
 - `14 _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 */
}
```

- Basic mechanism for Fiasco.OC + L4Re:
 - Exceptions are mapped to IPC messages
 - E waits for exception IPCs in server loop
- Timers 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**

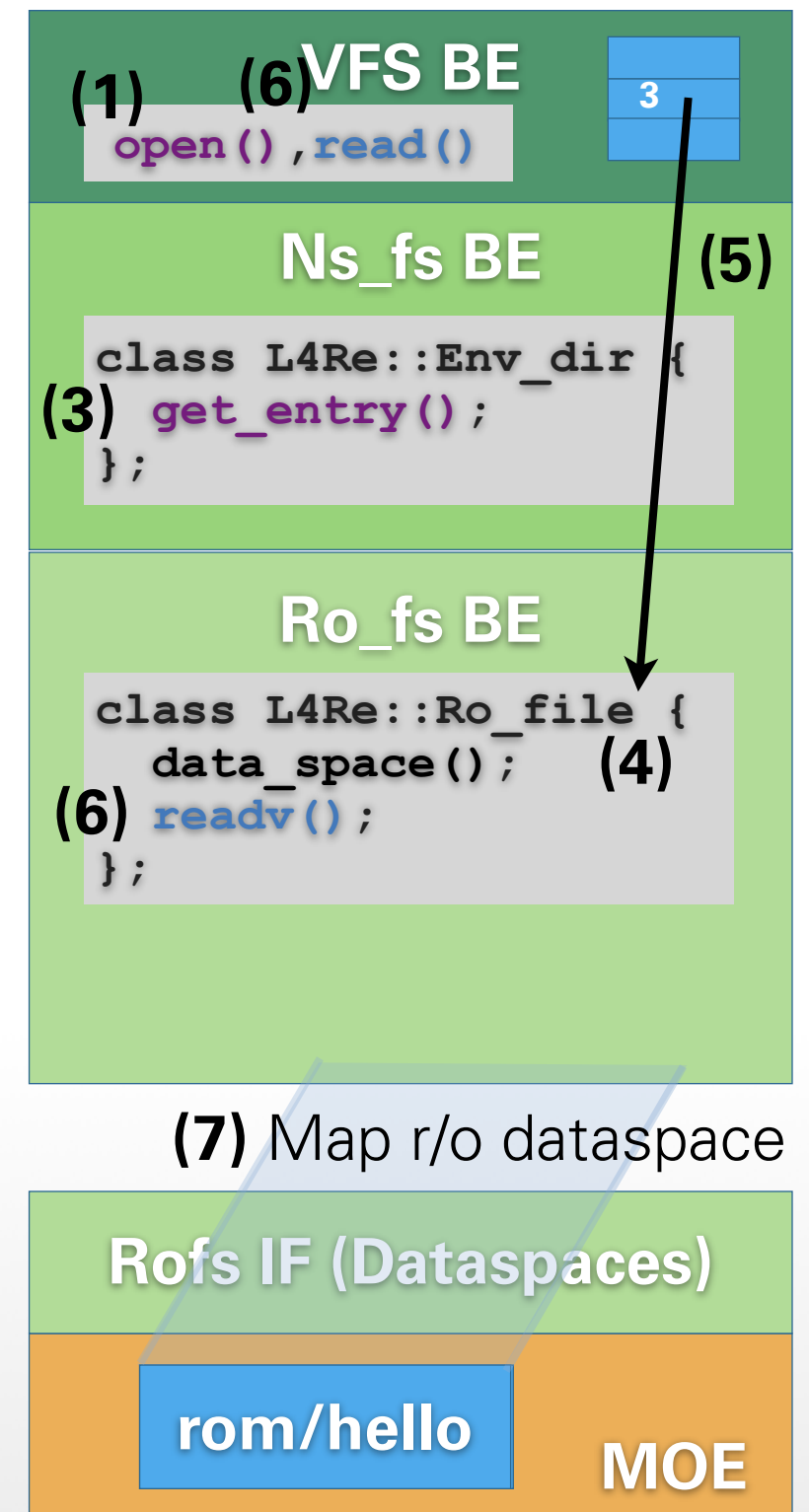
What is needed for file I/O?

- 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;
}
```

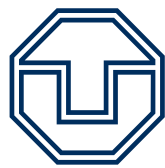
- (1) Application calls `open(„rom/hello“)`
- (2) VFS traverses mount tree, finds `Ro_fs` mounted at „rom“
- (3) VFS asks `Ro_fs` to provide a file for name „hello“, `Ro_fs` calls its `get_entry()` method
- (4) `Ro_fs::get_entry()` creates new `Ro_file` object from read-only dataspace provided by MOE
- (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

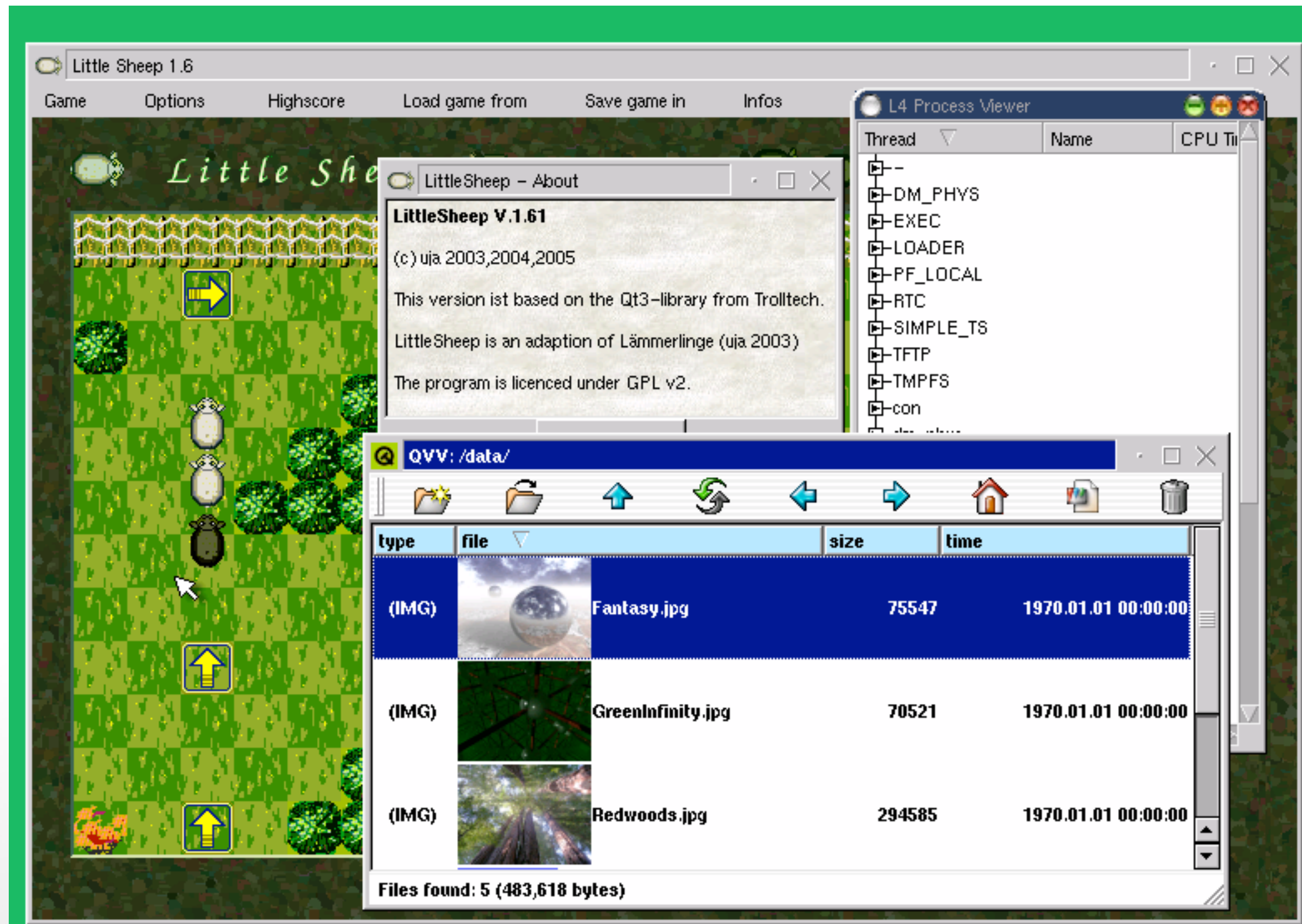


- L4Re offers most important POSIX APIs
 - C library: strcpy(), ...
 - Dynamic memory allocation:
 - malloc(), free(), mmap(), ...
 - Based on L4Re dataspaces
 - Threads, synchronization: Pthread
 - Signal handling
 - I/O support: files, terminal, time, (sockets)
- POSIX is enabler: sqlite, Cairo, SDL,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

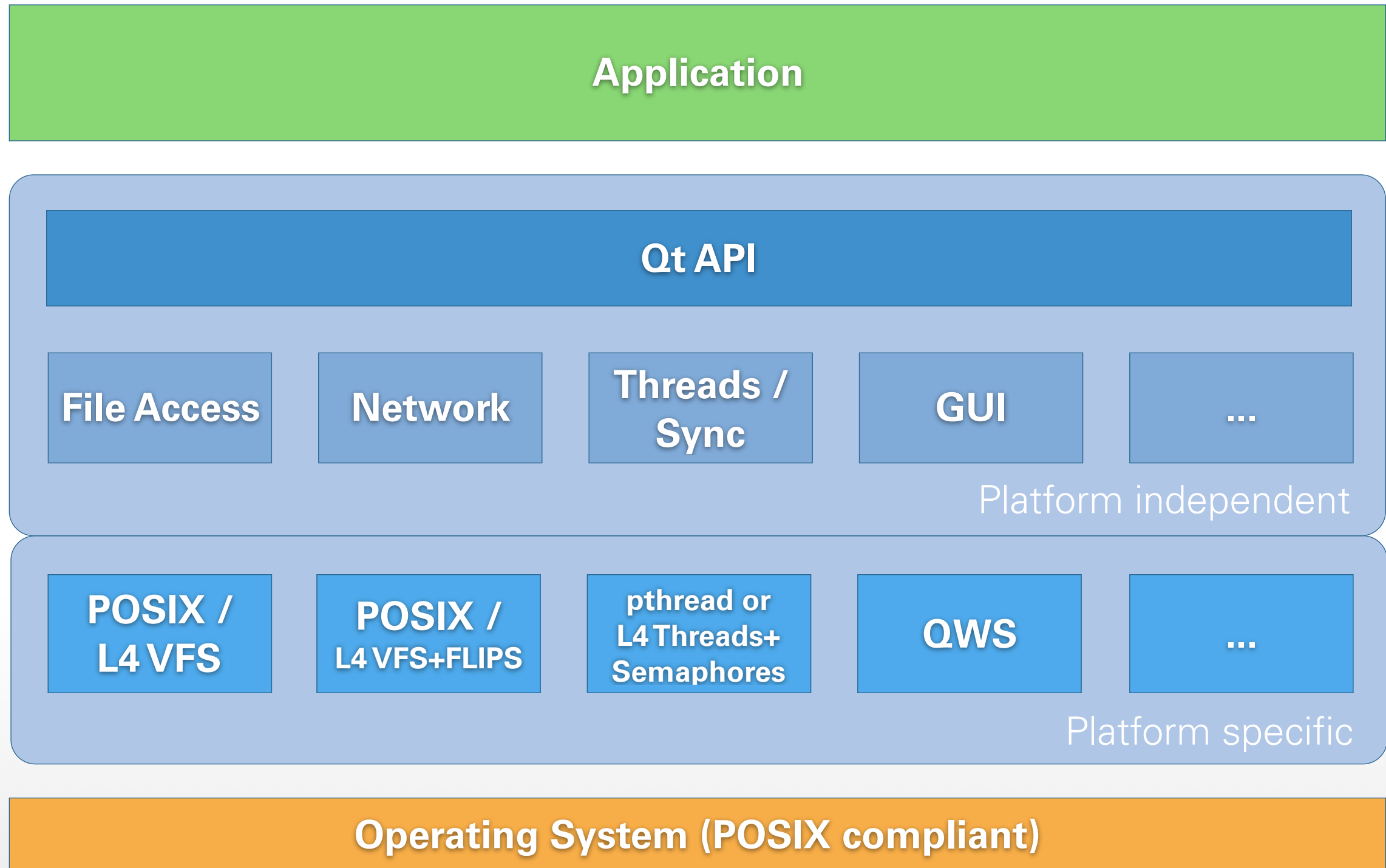
APPLICATION-LEVEL VIRTUALIZATION

- 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

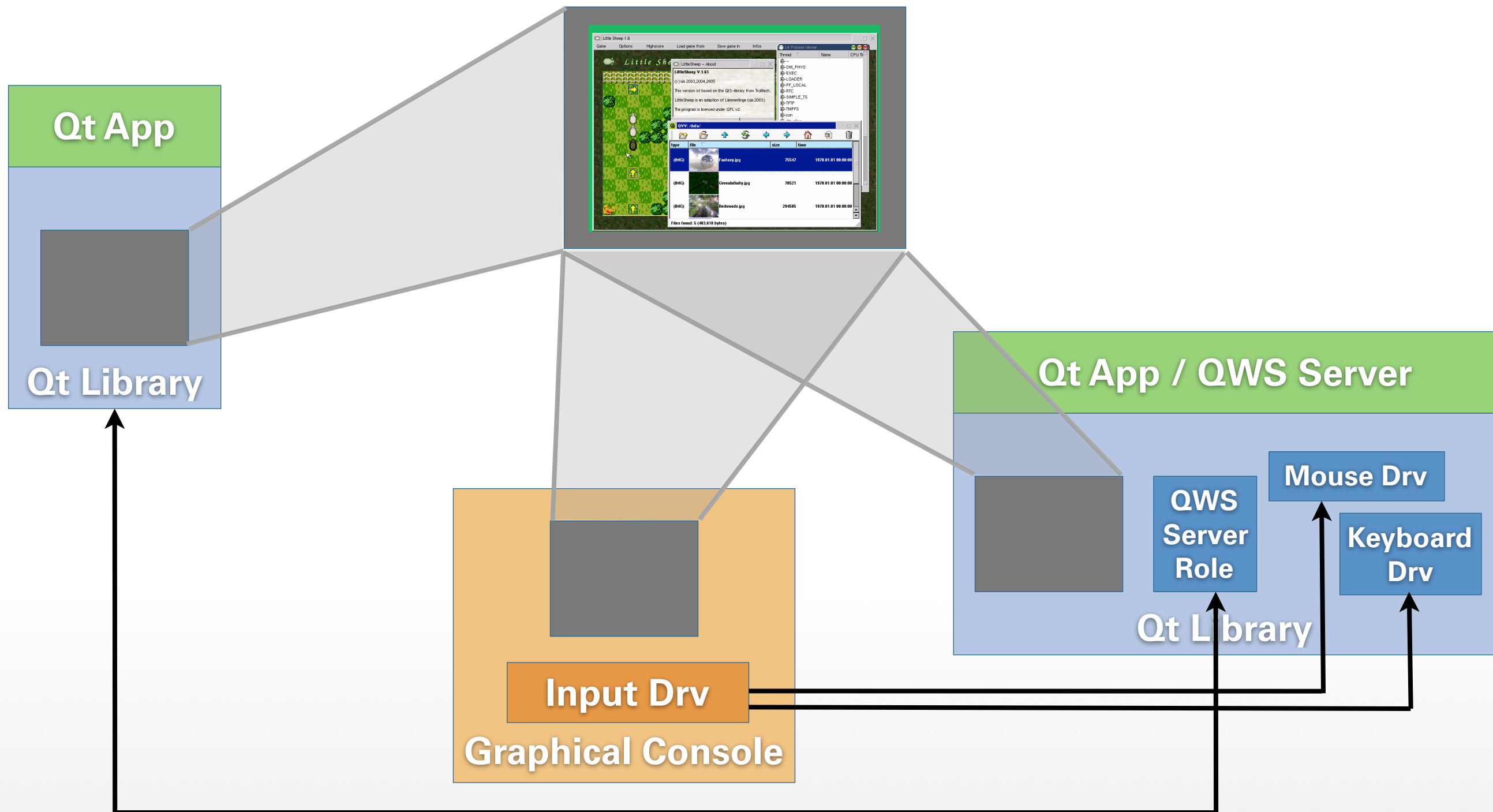




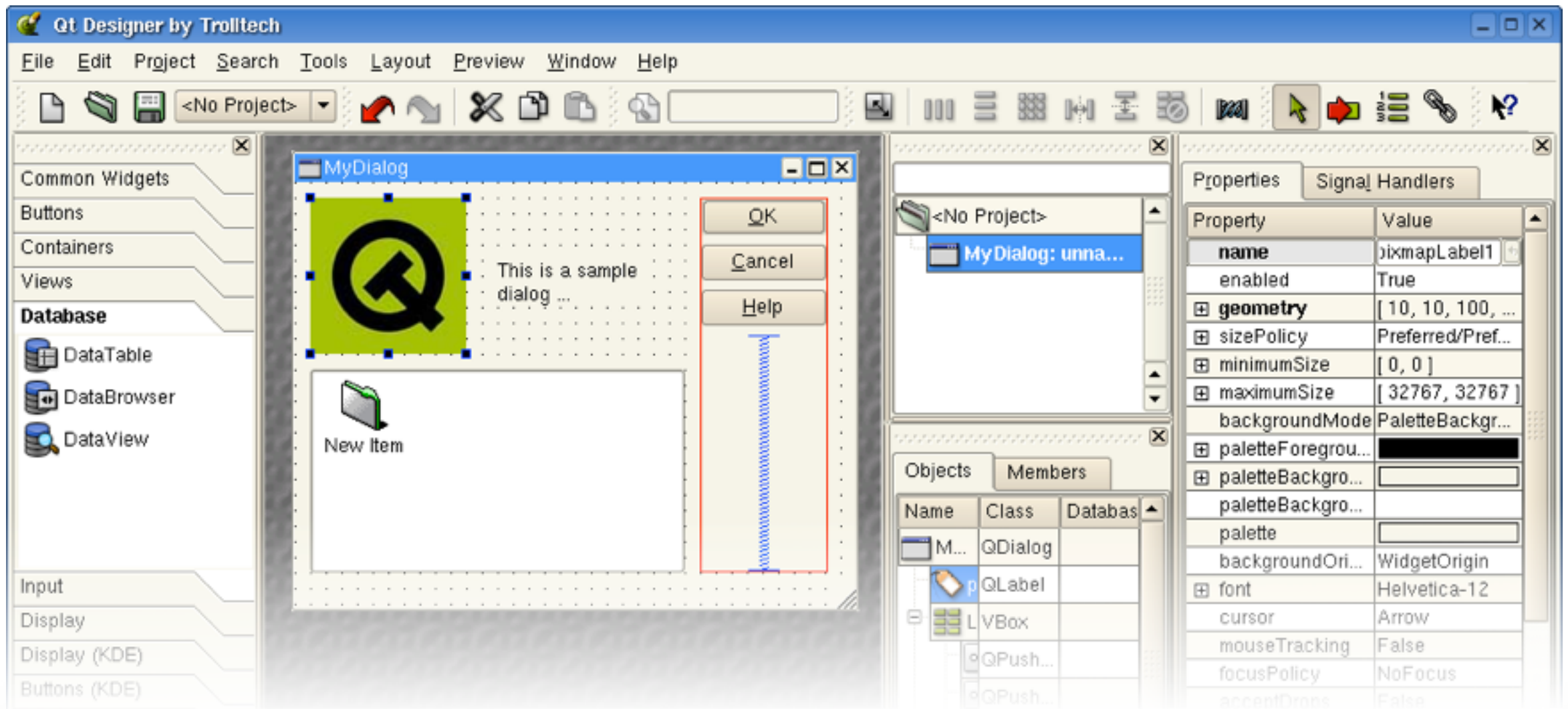
- **Qt** is a multi-platform toolkit library
- Compile & run same application on multiple platforms:
 - UNIX/X11, Embedded Linux
 - Windows, Mac OS X
 - Also ported to *L4Env*, *L4Re*, *Genode*
- L4 port based on Qt3 for Embedded Linux:
 - Brings its own windowing system
 - Relies on POSIX-like OS



DEMO



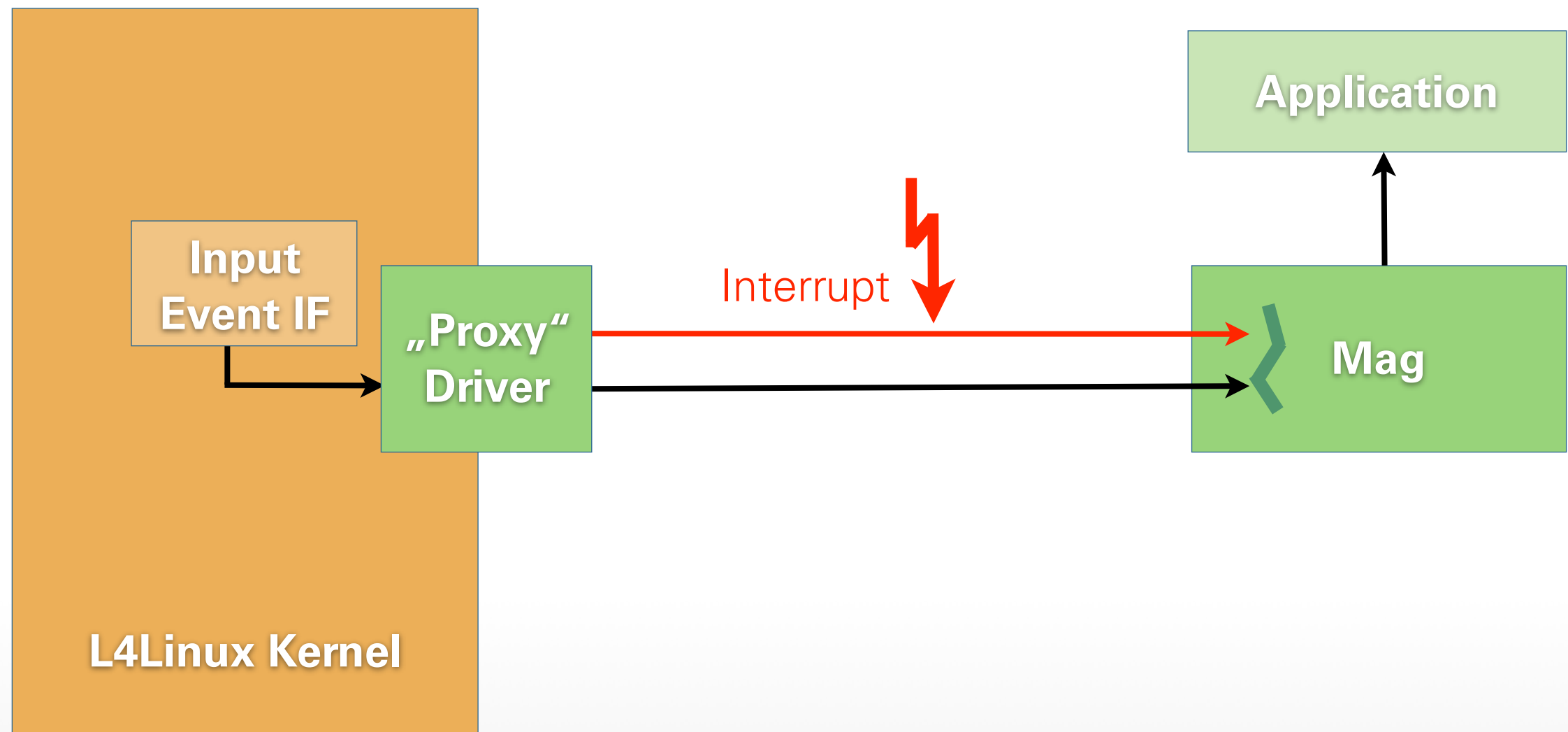
- Thanks to POSIX APIs available on L4:
 - Complete Qt GUI framework
 - Threads, synchronization
 - File access (L4 VFS backends / servers)
 - Limited network access:
 - Tool support: GUI builder



- Tools: Qt Designer, UI compiler (uic), moc
- Test Applications (or parts of them) on Linux

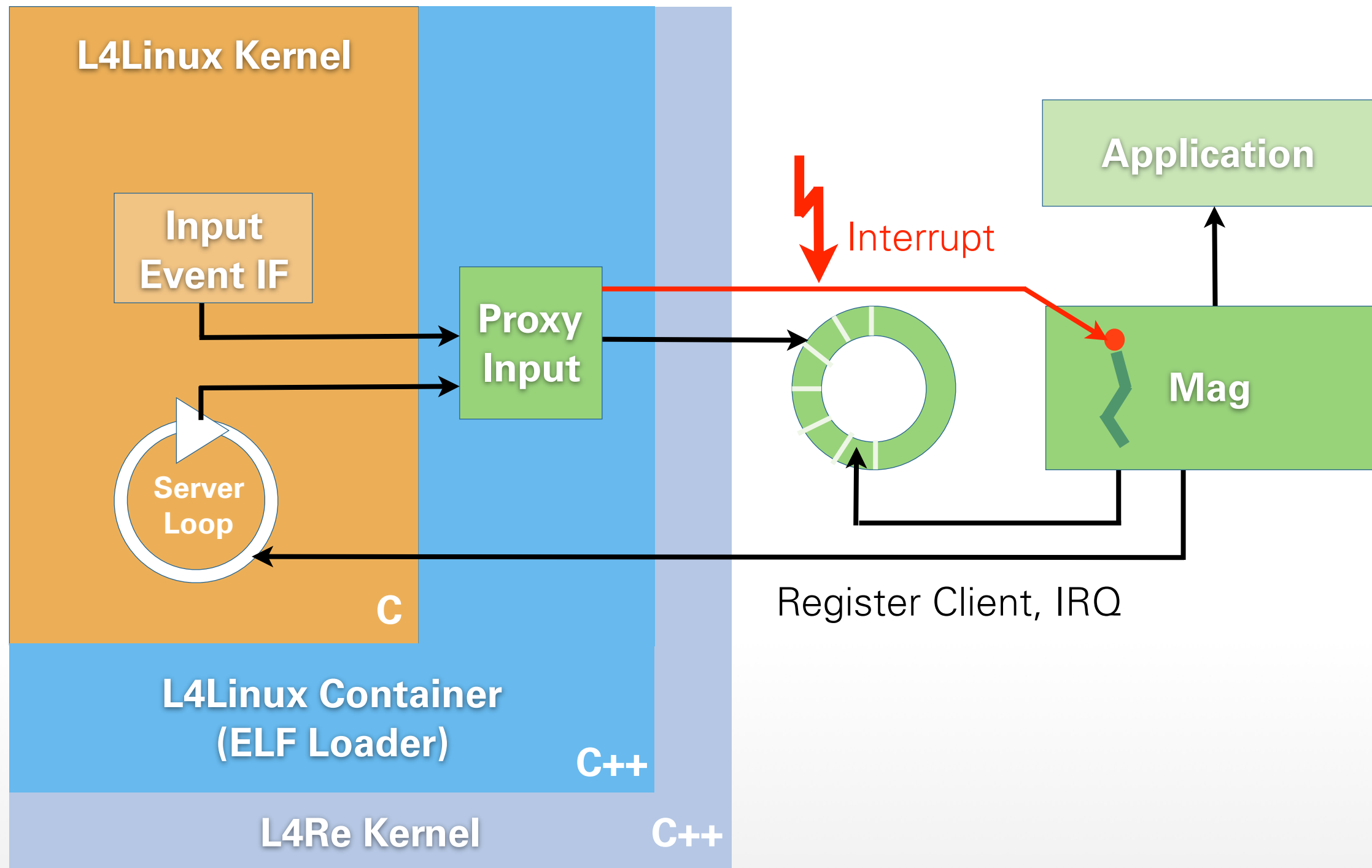
LEGACY OPERATING SYSTEM AS A TOOLBOX

- 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

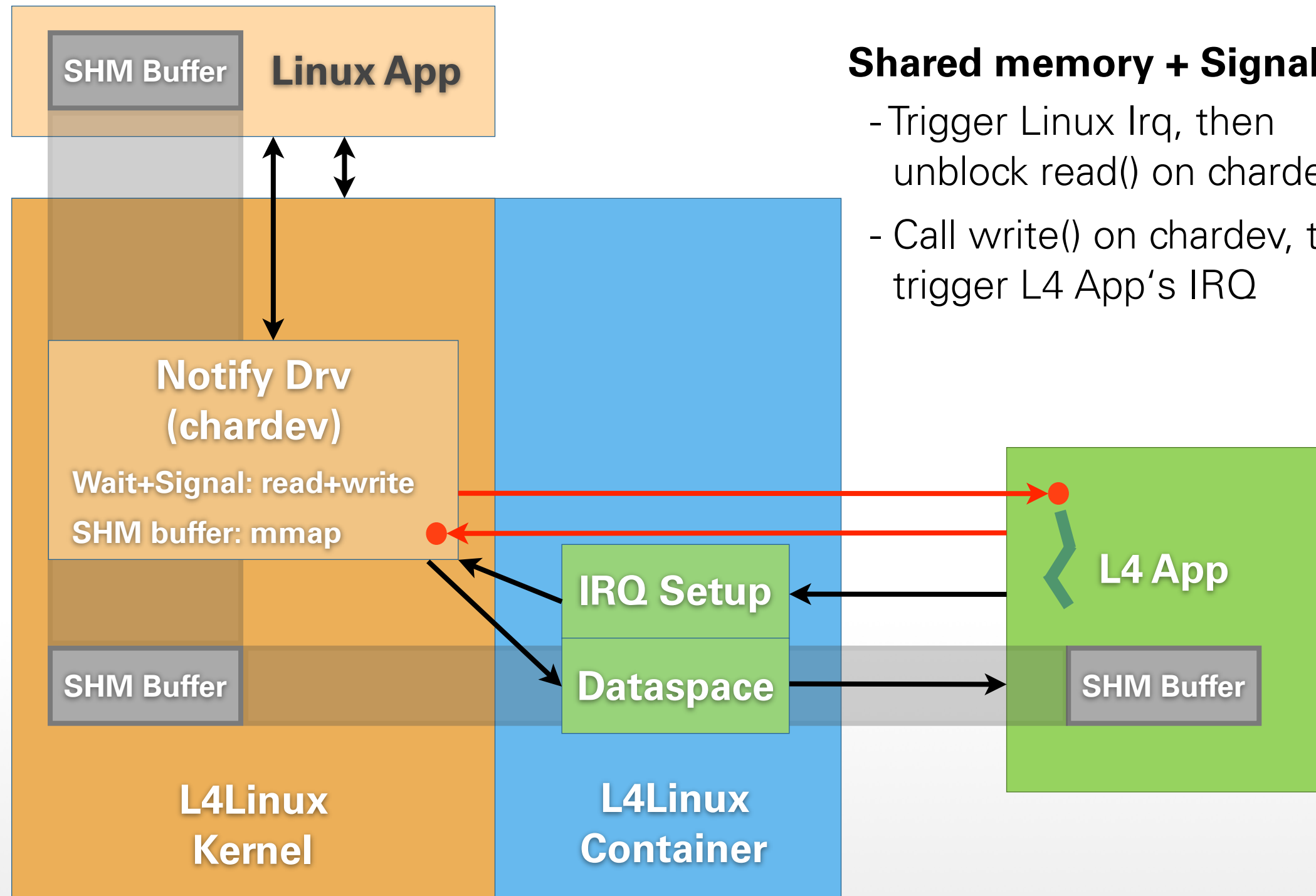


- L4Linux has drivers
- L4Re has great infrastructure for servers:
 - IPC framework
 - Generic server loop
- **Problem:** C vs. C++, calling conventions
- **Bridge:** allow calls from Linux to L4Re
 - L4Re underneath L4Linux export C

INPUT DRIVER



- **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



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

Next week:

- **Lecture:** „*Virtualization*“
- **No more exercises (this year)**

- [1] Carsten Weinhold: „**Portierung von Qt auf DROPS**“, TU Dresden, Großer Beleg 2005, http://os.inf.tu-dresden.de/paper_ps/weinhold-beleg.pdf
- [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