Escape

Nils Asmussen

MKC, 06/30/2016

Outline

Outline

Motivation

Beginning

- Writing an OS alone? That's way too much work!
- Port of UNIX32V to ECO32 during my studies
- Started with Escape in October 2008

Goals

- Learn about operating systems and related topics
- Experiment: What works well and what doesn't?
- What problems occur and how can they be solved?

Overview

Basic Properties

- UNIX-like microkernel OS
- Open source, available on `github.com/Nils-TUD/Escape`
- Mostly written in `C++`, some parts in `C`
- Runs on x86, x86_64, ECO32 and MMIX
- Only third-party code: `libgcc`, `libsupc++`, x86emu, inflate

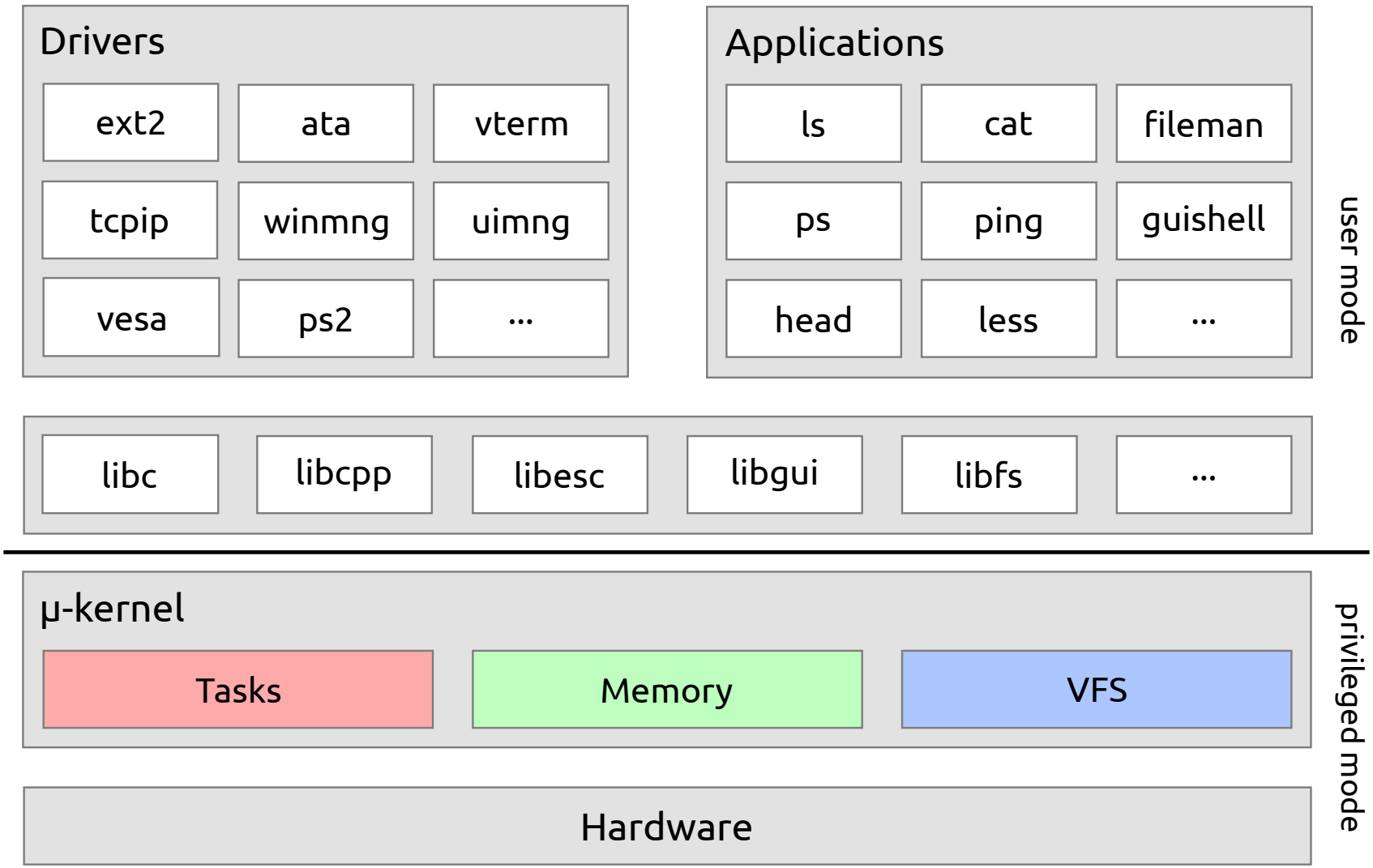
ECO32

MIPS-like, 32-bit big-endian RISC architecture, developed by Prof. Geisse for lectures and research

MMIX

64-bit big-endian RISC architecture of Donald Knuth as a successor for MIX (the abstract machine from TAOCP)

Overview


Drivers
ext2
ata
vterm
tcpip
winmng
uimng
vesa
ps2
...
Applications
ls
cat
fileman
ps
ping
guishell
head
less
...
user mode
libc
libcpp
libesc
libgui
libfs
...
µ-kernel
Tasks
Memory
VFS
privileged mode
Hardware

Outline

Processes and Threads

Process

- Virtual address space
- File-descriptors
- Mountspace
- Threads (at least one)
- ...

Thread

- User- and kernelstack
- State (running, ready, blocked, ...)
- Scheduled by a round-robin scheduler with priorities
- Signals
- ...

Processes and Threads

Synchronization

- Process-local semaphores (can also be created for interrupts)
- Global semaphores, named by a path to a file
- Userspace builds other synchronization primitives on top
  - Combination of atomic ops and process-local semaphores
  - Readers-writer-lock
  - ...

Priority Management

- Priorities are dynamically adjusted based on compute-intensity
- High CPU usage $\rightarrow$ downgrade, low CPU usage $\rightarrow$ upgrade

Outline

Memory Management

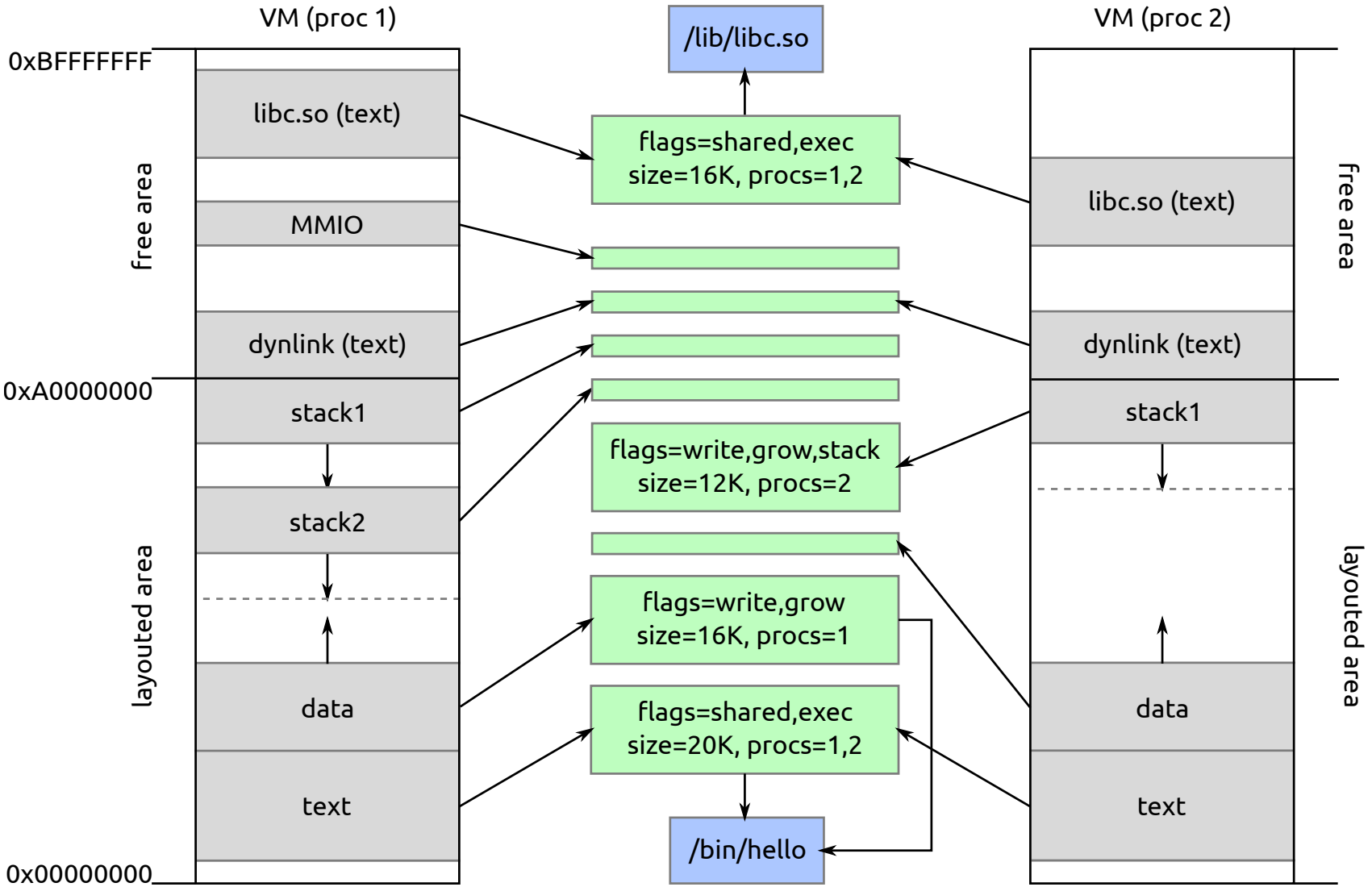
Physical Memory

- Mostly, memory is managed by a stack (fast for single frames)
- A small part handled by a bitmap for contiguous phys. memory

Virtual Memory

- Kernel part is shared among all processes
- User part is managed by a region-based concept
- `mmap`-like interface for the userspace

Virtual Memory Management



Outline

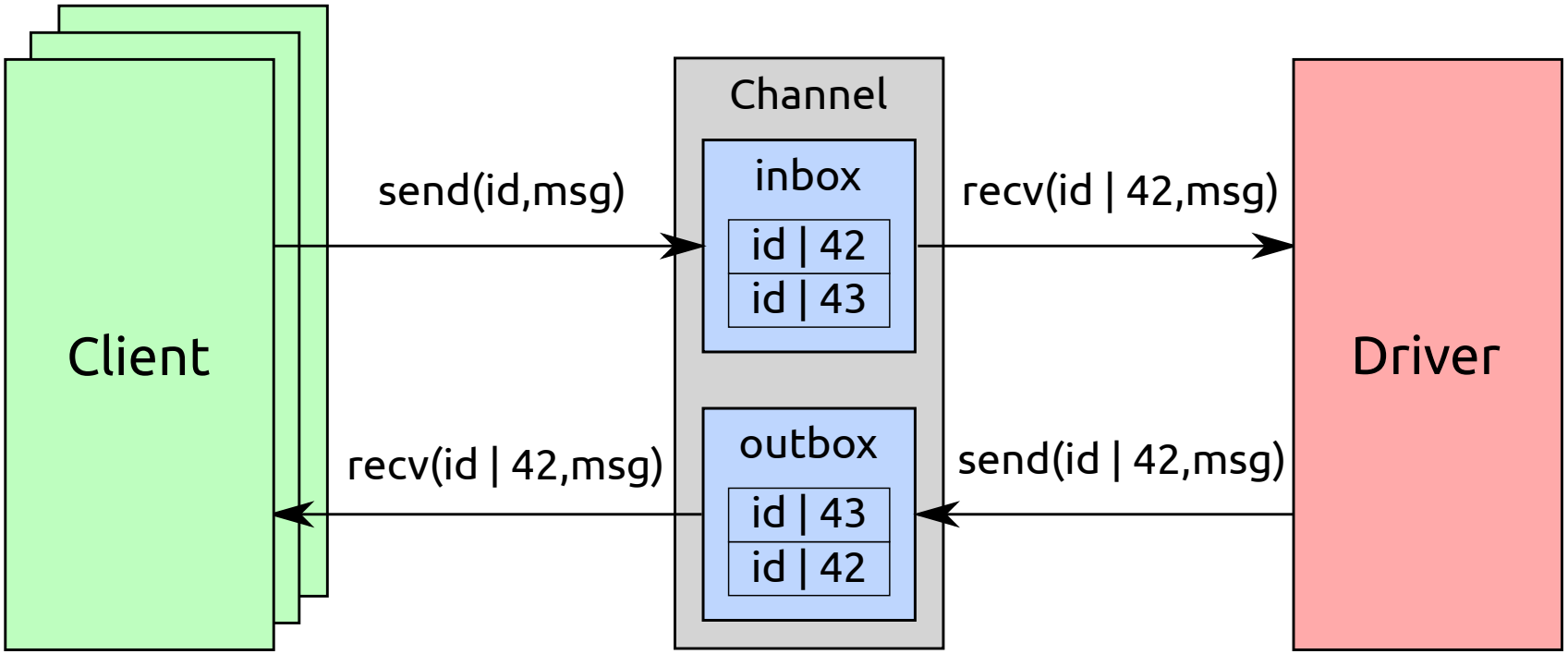
Basics

- The kernel provides the virtual file system
- System-calls: `open`, `read`, `mkdir`, `mount`, ...
- It's used for:
  1. Provide information about the state of the system
  2. Access userspace filesystems
  3. Access devices
  4. Access interrupts

Drivers and Devices

- Drivers are ordinary user-programs
- They create devices via the system-call `createdev`
- These are usually put into /dev
- Devices can also be used to implement on-demand-generated files (such as /sys/dev/hda)
- Communication is based on asynchronous message passing

Message Passing



Devices Can Behave Like Files

- As in UNIX: Devices should be accessable like files
- Messages: `FILE_OPEN`, `FILE_READ`, `FILE_WRITE`, `FILE_CLOSE`
- Devices may support a subset of these message
- Kernel handles communication for `open`/`read`/`write`/`close`
- Type of file transparent for applications

Devices Can Behave Like Filesystems

- Messages: `FS_OPEN`, `FS_READ`, `FS_WRITE`, `FS_CLOSE`, `FS_STAT`, `FS_SYNC`, `FS_LINK`, `FS_UNLINK`, `FS_RENAME`, `FS_MKDIR`, `FS_RMDIR`, `FS_CHMOD`, `FS_CHOWN`
- Kernel handles communication, if syscall refers to userspace fs
- Filesystems are mounted using the `mount` system call

Mounting Concept

- Every process has a mountspace, inherited to childs
- Mountspace is represented as a special file
- Mountspace can be cloned and joined
- Read permissions are required for `clonems`/`joinms`
- Write permissions are required for `mount`/`umount`
- Mountspace contains list of (*path*,*fs-con*) pairs
- Kernel translates fs syscalls into messages to *fs-con*

Mounting for the User

Tools

- `mount` creates a new FS for a device and makes it visible
  - `$ mount /dev/hda1 /mnt /sbin/ext2`
  - Creates `/dev/ext2-hda1`
- `bind` makes an existing FS visible at a different place
  - `$ bind /dev/ext2-hda1 /home/me/mnt`

What does bind do?

```
int fd = open("/dev/ext2-hda1", ...);
int ms = open("/sys/proc/self/ms", O_WRITE);
mount(ms, fd, "/home/me/mnt");
// open("/home/me/mnt/a/b", ...) -> FS_OPEN("/a/b")
```

Security Concerns with Mounting

- Each process has its own mountspace
- But can it shape the MS in arbitrary ways?
- What if it over-mounts system directories?
- E.g., users, groups, passwords, ...
- Process needs write permissions to the mountpoint
- Write permission to dir already allows add/remove
- Sticky directories: needs to be the owner

Interrupts

Getting IRQs to Userspace

- Drivers run in userspace; how do they get interrupts?
- Escape uses semaphores for interrupts
- Syscall `semirqcrt` creates new semaphore for given interrupt
- On an IRQ, all semaphores in the list are up'ed

Access Control

- For each interrupt, Escape creates a file /sys/irq/$irq
- Can be read to get information about this interrupt
- `semirqcrt` takes file descriptor to /sys/irq/$irq
- Exec permission is required to register for IRQs

Achieving Higher Throughput

- Copying everything twice hurts for large amounts of data
- `sharebuf` establishes shmem between client and driver
- Easy to use: just call `sharebuf` once and use this as the buffer
- Clients don't need to care whether a driver supports it or not
- Drivers need to handle `DEV_SHFILE` to support it
- In `read`/`write`, they check if SHM should be used

Achieving Higher Throughput – Code Example

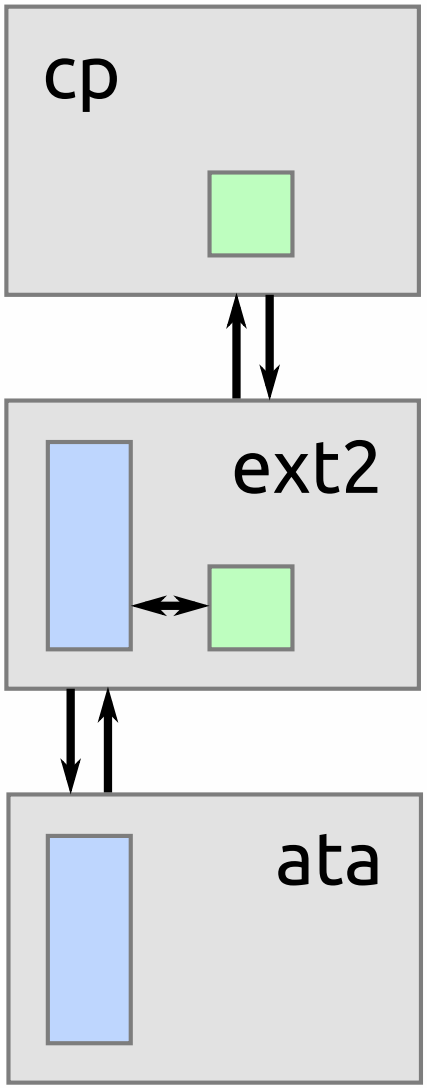
```
int fd = open("/dev/zero",IO_READ);

static char buf[SIZE];




while(read(fd,buf,SIZE)) > 0) {
    // ...
}

close(fd);
```

Achieving Higher Throughput – Code Example

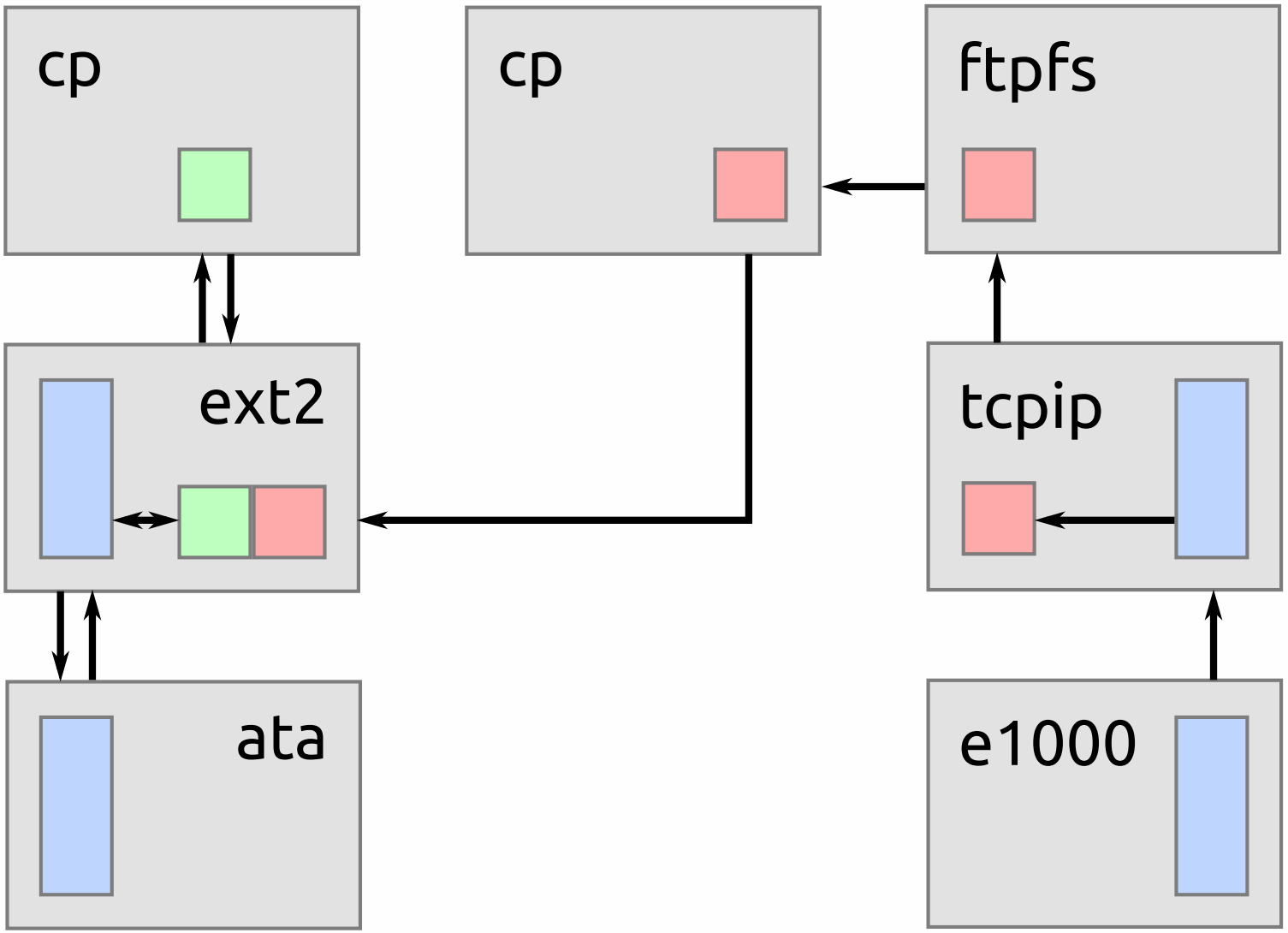
```c
int fd = open("/dev/zero",IO_READ);

static char buf[SIZE];

while(read(fd,buf,SIZE)) > 0) {
    // ...
}

close(fd);
```

```c
int fd = open("/dev/zero",IO_READ);

ulong shname;
void *buf;
if(sharebuf(fd,SIZE,&buf,&shname,0) < 0) {
    if(buf == NULL)
        error("Unable to mmap buf");
}

while(read(fd,buf,SIZE) > 0) {
    // ...
}

destroybuf(buf,shname);
close(fd);
```

Achieving Higher Throughput – Usage Example



Achieving Higher Throughput – Usage Example



Canceling Operations

Problem

- What if we want to SIGTERM a process during a read?
- An already sent read-request can't be taken back
- Channels might be shared (shared state . . .)

Canceling Operations

Problem

- What if we want to SIGTERM a process during a read?
- An already sent read-request can't be taken back
- Channels might be shared (shared state . . .)

Solution

- Introduce `cancel` syscall and message
- If a thread gets a signal, it wakes up and sends the cancel message to the driver
- The driver cancels the currently pending request, if necessary
- Race-condition: the driver might have already responded

Sibling Channels

Problem

- Suppose, you need a control and event channel per client
- Suppose, you want to implement socket's accept
- How do you do that?

Sibling Channels

Problem

- Suppose, you need a control and event channel per client
- Suppose, you want to implement socket's accept
- How do you do that?

Solution

- Introduce `creatsibl` syscall and message
- Kernel creates new channel & sends `DEV_CREATSIBL` to driver
- Driver receives it over old channel → knows both channels
- Driver can then associate both channels with each other

Integrating Networking

- Network services should be accessible like files or filesystems
- To support URLs:
  `"XYZ://foo/bar"` is translated to `"/dev/XYZ/foo/bar"`

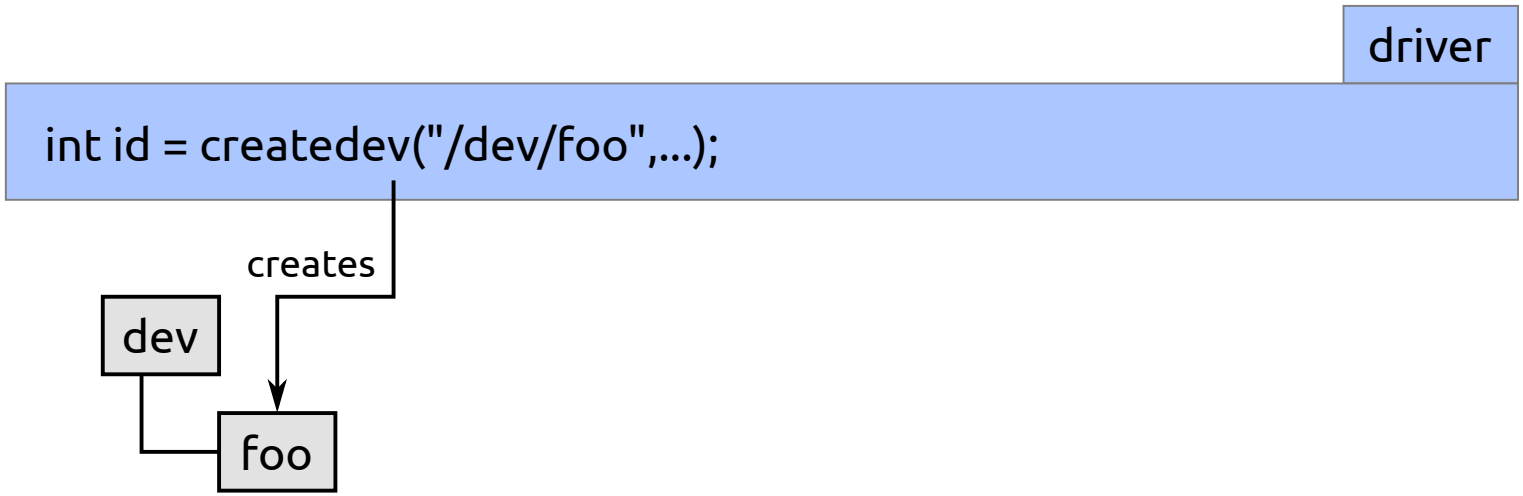
Integrating Networking

- Network services should be accessible like files or filesystems
- To support URLs:
  `"XYZ://foo/bar"` is translated to `"/dev/XYZ/foo/bar"`

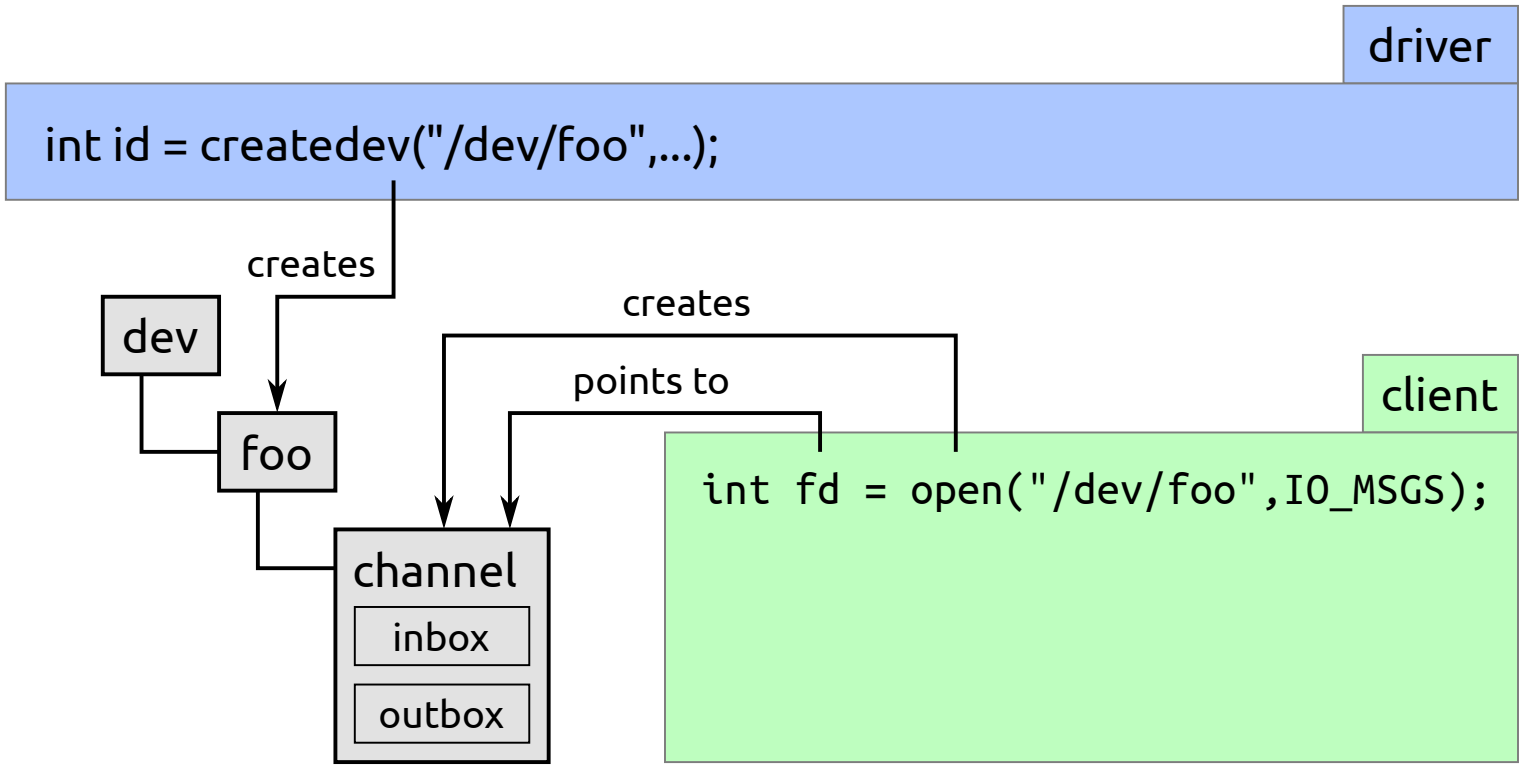
Demo!

Outline

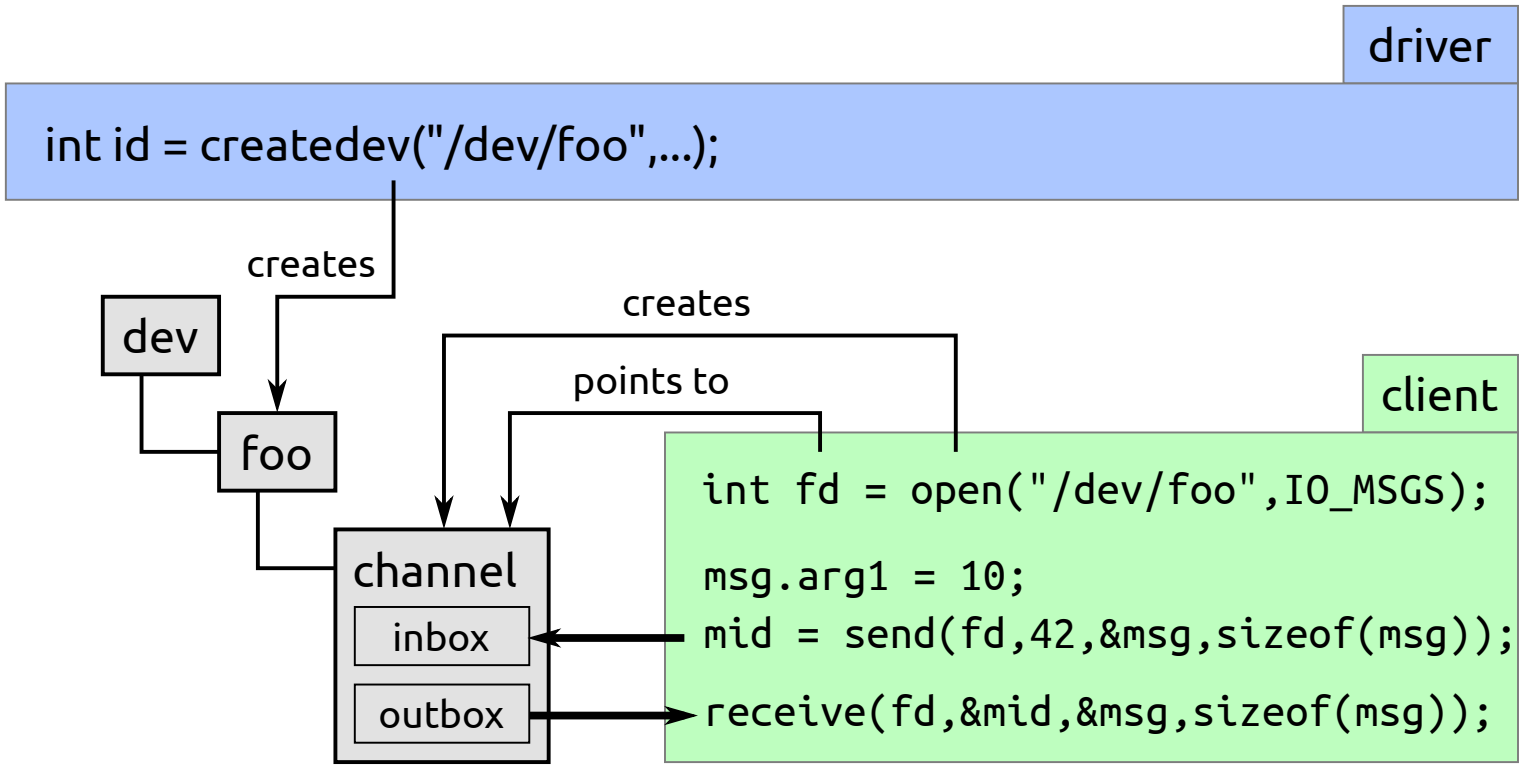
IPC between Client and Driver (Low Level)



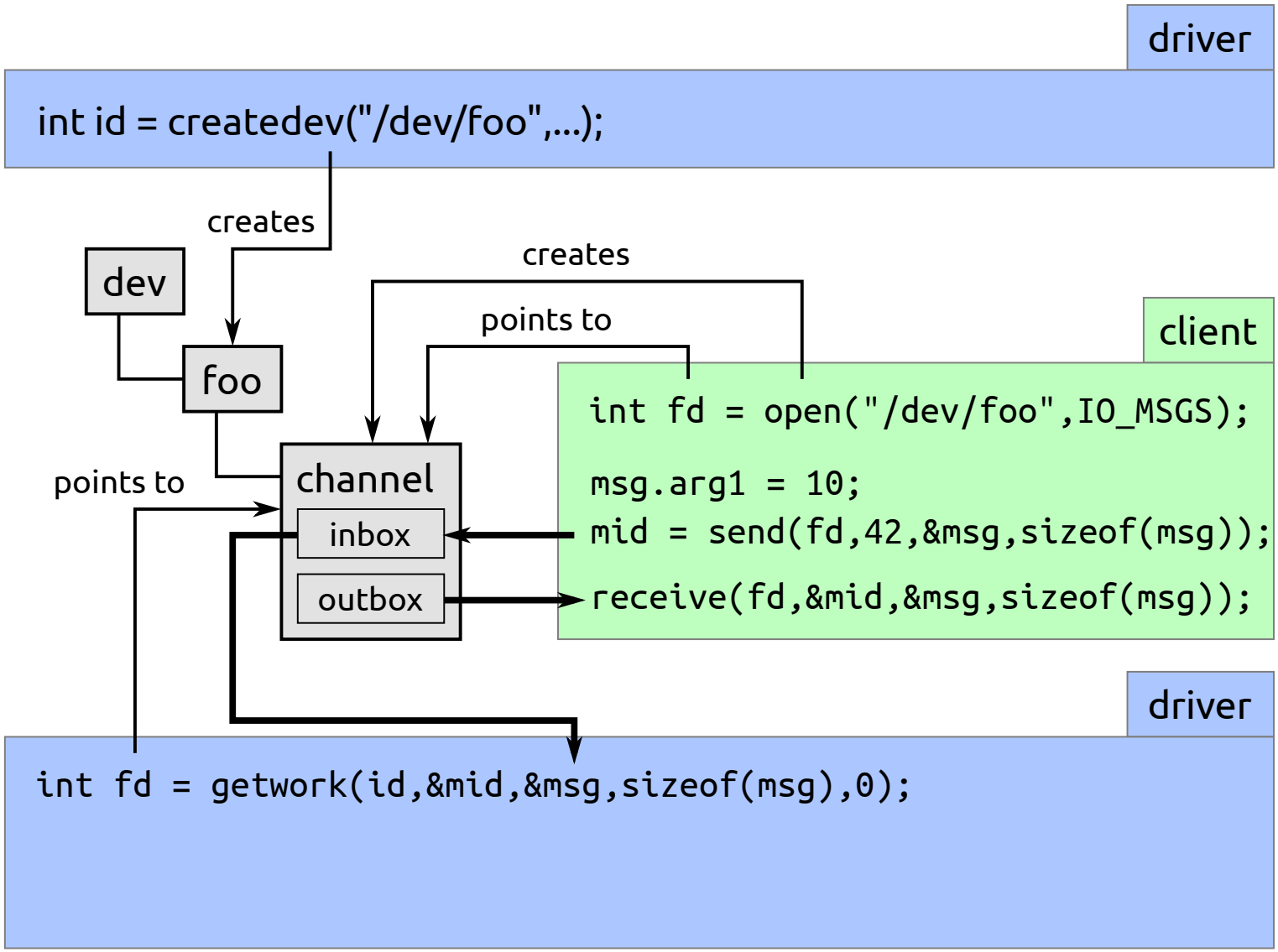
IPC between Client and Driver (Low Level)



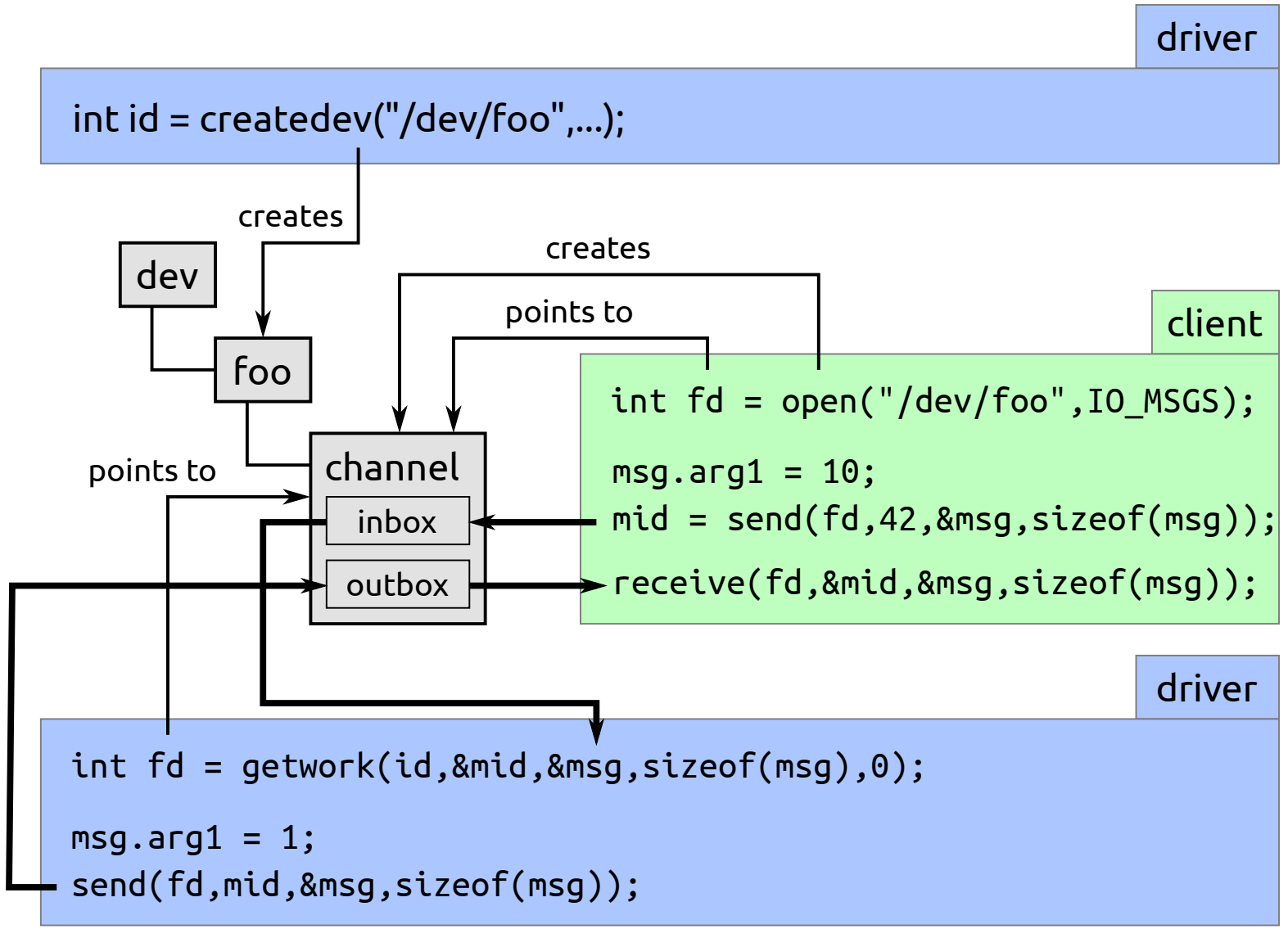
IPC between Client and Driver (Low Level)



IPC between Client and Driver (Low Level)



IPC between Client and Driver (Low Level)



Driver Example: /dev/zero

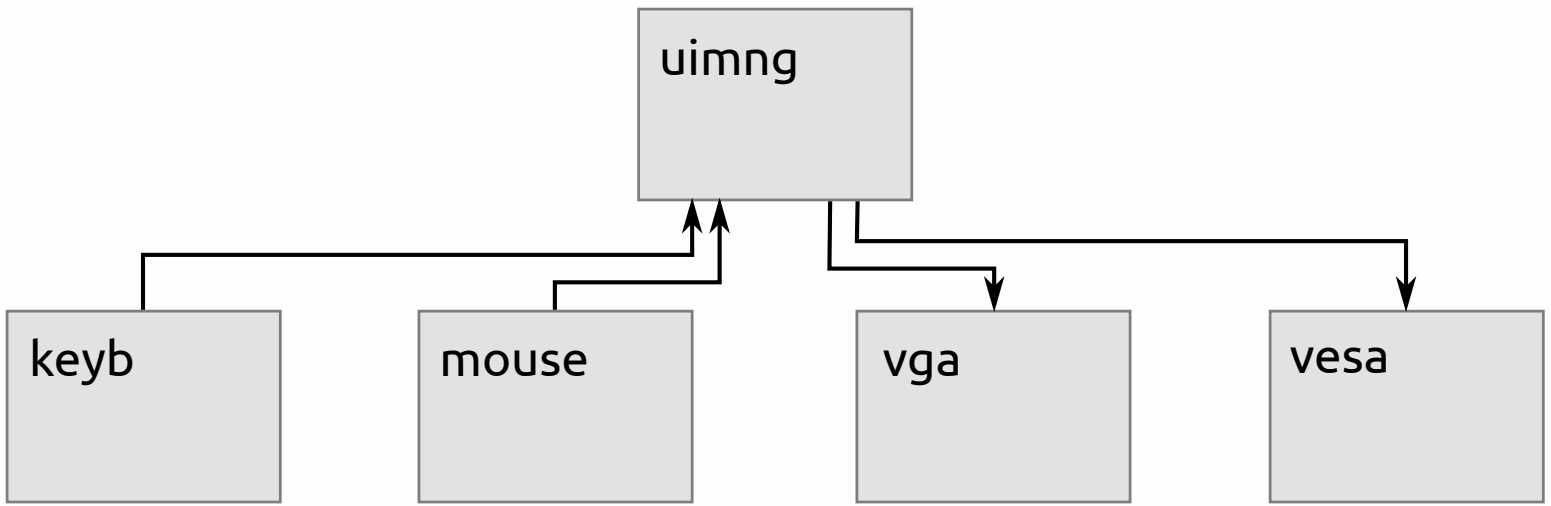
```
struct ZeroDevice : public ClientDevice<> {
    explicit ZeroDevice(const char *name, mode_t mode)
        : ClientDevice(name, mode, DEV_TYPE_BLOCK, DEV_OPEN | DEV_SHFILE |
            DEV_READ | DEV_CLOSE) {
        set(MSG_FILE_READ, std::make_memfun(this, &ZeroDevice::read));
    }

    void read(IPCStream &is) {
        static char zeros[BUF_SIZE];
        Client *c = get(is.fd());
        FileRead::Request r;
        is >> r;

        if(r.shmemoff != -1)
            memset(c->shm() + r.shmemoff, 0, r.count);
        is << FileRead::Response(r.count) << Reply();
        if(r.shmemoff == -1 && r.count)
            is << ReplyData(zeros, r.count);
    }
};

int main() {
    ZeroDevice dev("/dev/zero", 0400);
    dev.loop();
    return EXIT_SUCCESS;
}
```

Client Example: vterm

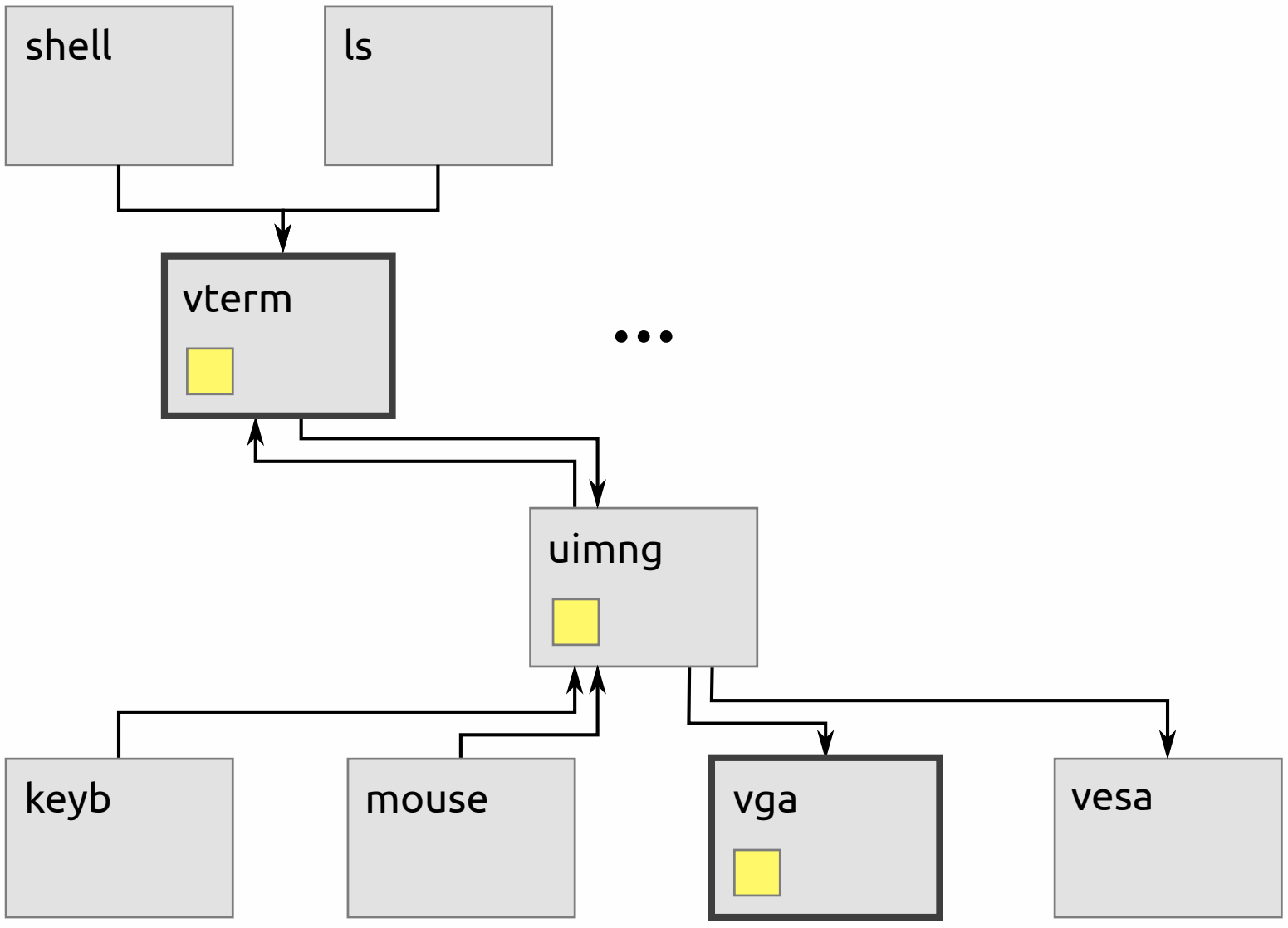
```
// get console-size
ipc::VTerm vterm(std::env::get("TERM").c_str());
ipc::Screen::Mode mode = vterm.getMode();

// implementation of vterm.getMode():
Mode getMode() {
    Mode mode;
    int res;
    _is << SendReceive(MSG_SCR_GETMODE) >> res >> mode;
    if(res < 0)
        VTHROWE("getMode()",res);
    return mode;
}
```

Outline

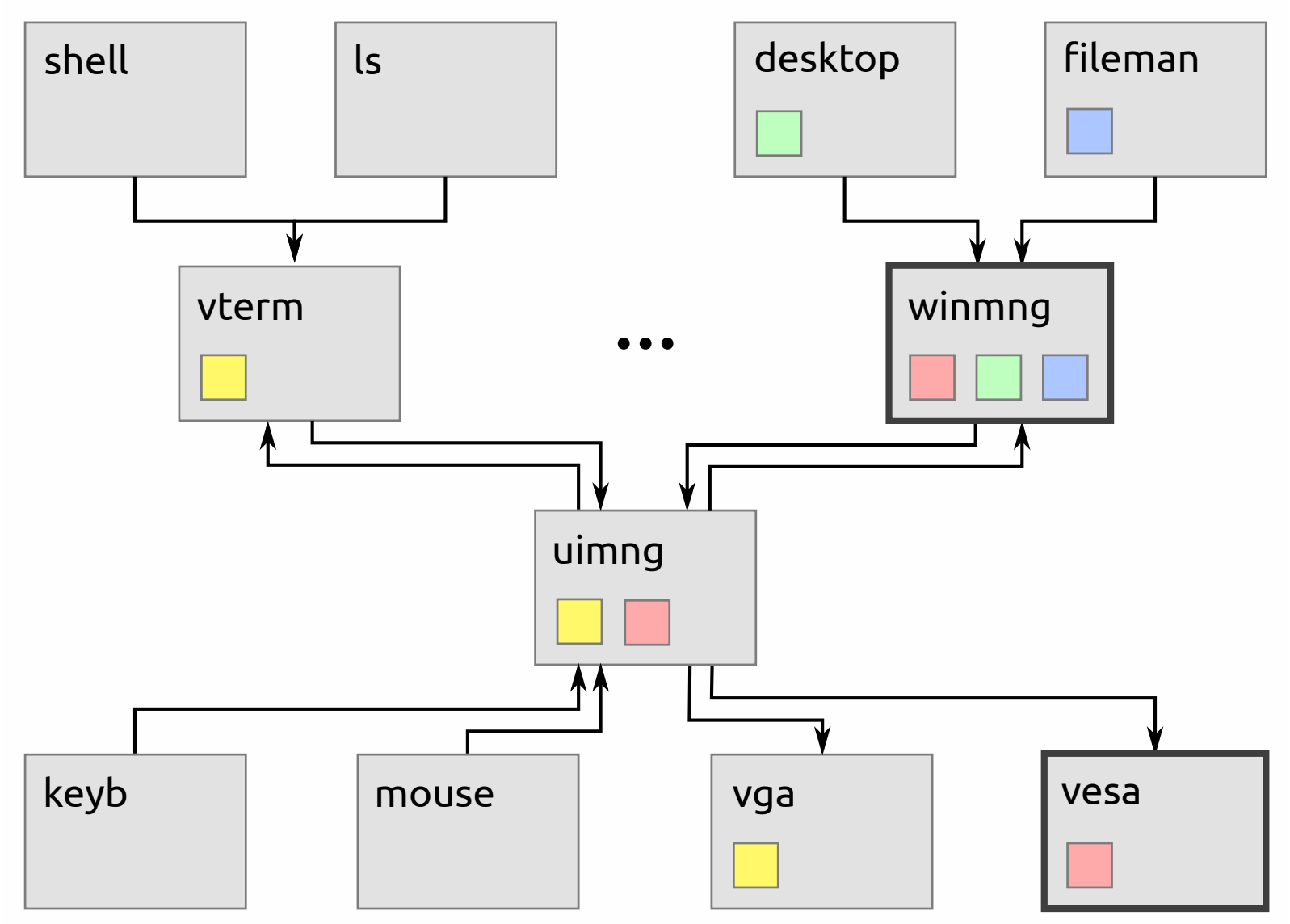
UI Concept



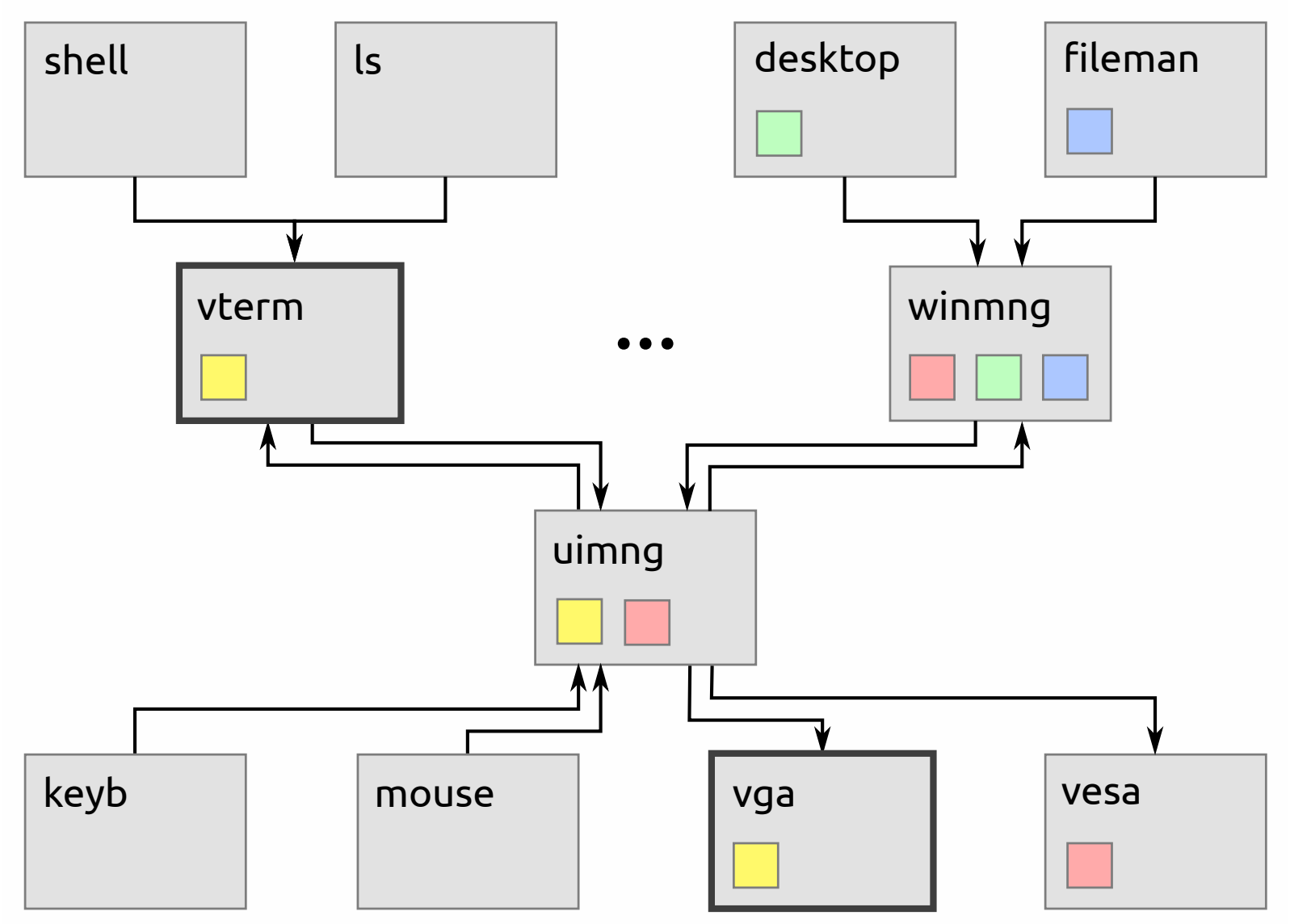
UI Concept



UI Concept


shell
ls
desktop
fileman
vterm
winmng
uimng
keyb
mouse
vga
vesa

UI Concept


shell
ls
desktop
fileman
vterm
winmng
uimng
keyb
mouse
vga
vesa

UI

Demo

Questions

Get the code, ISO images, etc. on:
`https://github.com/Nils-TUD/Escape`

Any questions to Escape?