

# Moslab – Chair of Operating Systems

## Keyboard Device Driver & Integration

Viktor Reusch and Tianhao Wang,  
original slides by Martin Küttler

# Last assignment

Any questions?

# Hotfix

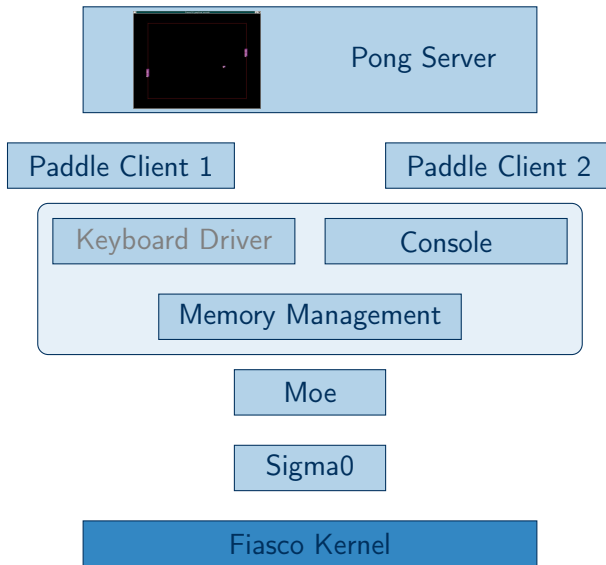
I fixed some bugs in the code base at

<https://os.inf.tu-dresden.de/repo/gitbox/teaching/moslab-code.git>:

996b758d, 66818ee4, 6ea3af57, 4ed78e3a

You should cherry-pick or rebase to get the hotfix applied.

We are here



# Let's Play Pong

You will need a Ned configuration file...

```
local pong_server = ld:new_channel();
ld:start({
  caps = { PongServer = pong_server:svr(), vesa = fbdrv },
  log = { "server", "blue" },
}, "rom/pong-server");

ld:start({
  caps = { PongServer = pong_server },
  log = { "client", "green" },
}, "rom/pong-client");
```

Look at your previous assignments for `modules.list` and frame buffer setup.

## Let's Play Pong

You will need a Ned configuration file...

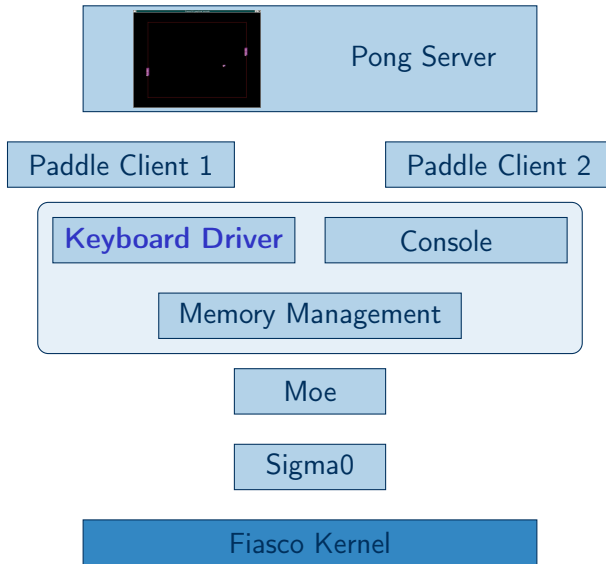
```
local pong_server = ld:new_channel();
ld:start({
  caps = { PongServer = pong_server, vesa = fbdrv },
  log = { "server", "blue" },
}, "rom/pong-server");

ld:start({
  caps = { PongServer = pong_server },
  log = { "client", "green" },
}, "rom/pong-client");
```

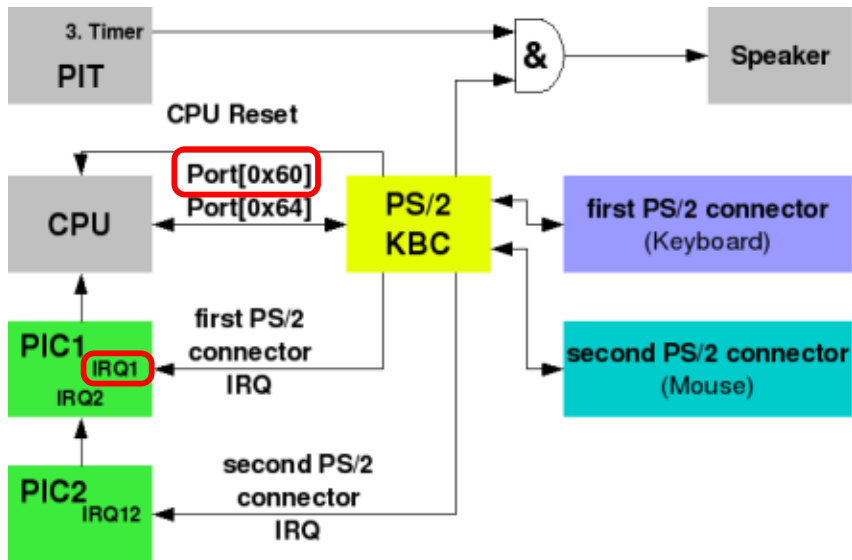
Look at your previous assignments for modules.list and frame buffer setup.

**But how to control the paddles?**

# Today's goal



## PS/2 Keyboard Controller



Source: [http://wiki.osdev.org/\"8042\"\\_PS/2\\_Controller](http://wiki.osdev.org/\)



# Driving the keyboard

- ▶ Subscribe to interrupt 0x1.
- ▶ On interrupt:
  - ▶ Read scan code from I/O port 0x60 (`inb 0x60`).
  - ▶ Translate scan code into key code and action.
- ▶ Wrap a server interface around it, and you're done.

## Getting access to the IO port

Add to x86-legacy.devs (inside outer function)

```
PS2 = Hw.Device(function()  
    Property.hid = "PNP0303";  
    Resource.iop1 = Res.io(0x60, 0x60); -- PS/2 device 1  
    Resource.iop2 = Res.io(0x64, 0x64); -- PS/2 device 2  
    Resource.irq1 = Res.irq(1, 0x000000);  
    Resource.irq2 = Res.irq(12, 0x000000);  
end);
```

## Getting access to the IO port

The following is already in x86-fb.io (and probably shouldn't be called gui, feel free to rename).

```
Io.add_vbus("gui", Io.Vi.System_bus
{
  ps2 = wrap(hw:match("PNP0[3F]??"));
})
```

Then give IO a server cap (called gui) to a gate, and give the client cap to your keyboard server (called vbus).

## How to handle irqs and ioports in C

- ▶ For irqs look at `pkg/examples/sys/isr` (it's C, you can figure out the C++ interface).
- ▶ Though, you will need `l4io_request_icu` to get the ICU.
- ▶ Request io port from vbus: `l4io_request_ioport(0x60, 1)`.
- ▶ Read value from io port (after you received an interrupt): `l4util_in8(0x60)`.

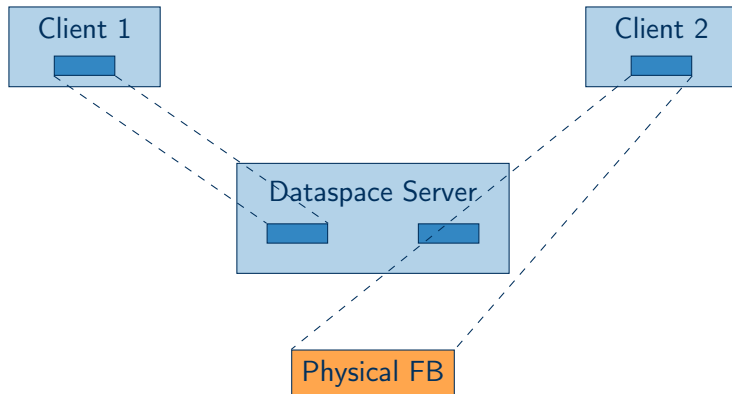
## Assignment, part 1

- ▶ Build a working keyboard driver and keyboard multiplexer.
- ▶ The keyboard driver sends the keyboard events to the multiplexer.
- ▶ The multiplexer remembers which keys are currently pressed.
- ▶ You already have working pong clients in `src/l4/pkg/pong/examples`.
- ▶ Modify the pong clients to be controllable by keyboard, with different controls.
- ▶ Let the pong clients periodically query the pressed keys from the keyboard multiplexer.

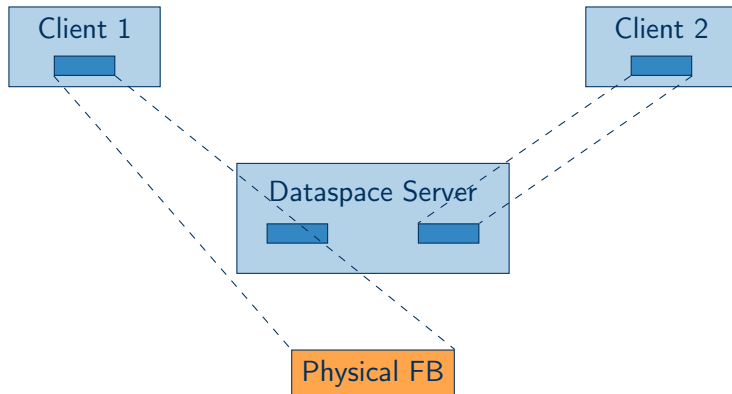
# Graphical console multiplexing

- ▶ Now there are two programs that can draw: pong and the console, so we need to multiplex graphics.
- ▶ One of them should render into physical framebuffer, while the other renders into plain memory.
- ▶ You will need a dataspace server that serves both clients.
- ▶ For switching, that server will unmap both dataspaces and remapped them in reverse order.

## Graphical console multiplexing



## Graphical console multiplexing





# Graphical console multiplexing

- ▶ Your server will need to
  - ▶ hand out two capabilities to frame buffers (i.e. to gates, that you respond on),
  - ▶ implement the frame buffer interface as defined in `src/l4/pkg/l4re-core/l4re/include/video/goos`, and
  - ▶ implement dataspace as defined in `src/l4/pkg/l4re-core/l4re/include/dataspace`.
- ▶ Have a look at `src/l4/pkg/l4re-core/l4re/util/include/dataspace_svr` for a nearly complete dataspace implementation.
- ▶ Hint: Make sure that the dataspace server has the pages that it wants to map out to clients mapped itself.

# Switching Console Clients

1. User indicates a client switch.
2. Unmap physical FB from client.
3. Make client's FB point to a virtual copy (on next page fault).
4. Unmap new client's virtual FB.
5. Copy new client's virtual data into physical FB.
6. Make new client's FB point to physical FB (on next page fault).

## Assignment, part 2

- ▶ Implement console switching, so that the user can play pong and switch to the console at any time.
- ▶ Extend your keyboard multiplexer to a full-fledged console multiplexer (keyboard + frame buffer).

## Assignment, part 3

- ▶ On real hardware you can't read pong's output: Edit `send_ipc()` in `pkg/pong/include/logging.h` to send all output to your log server.
- ▶ Prepare for question regarding this lab in the MOS exam.