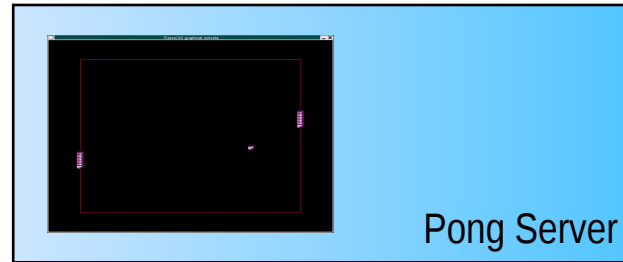




Complex Lab — Operating Systems

Graphical Console

- Any comments? Questions?



Paddle Client 1

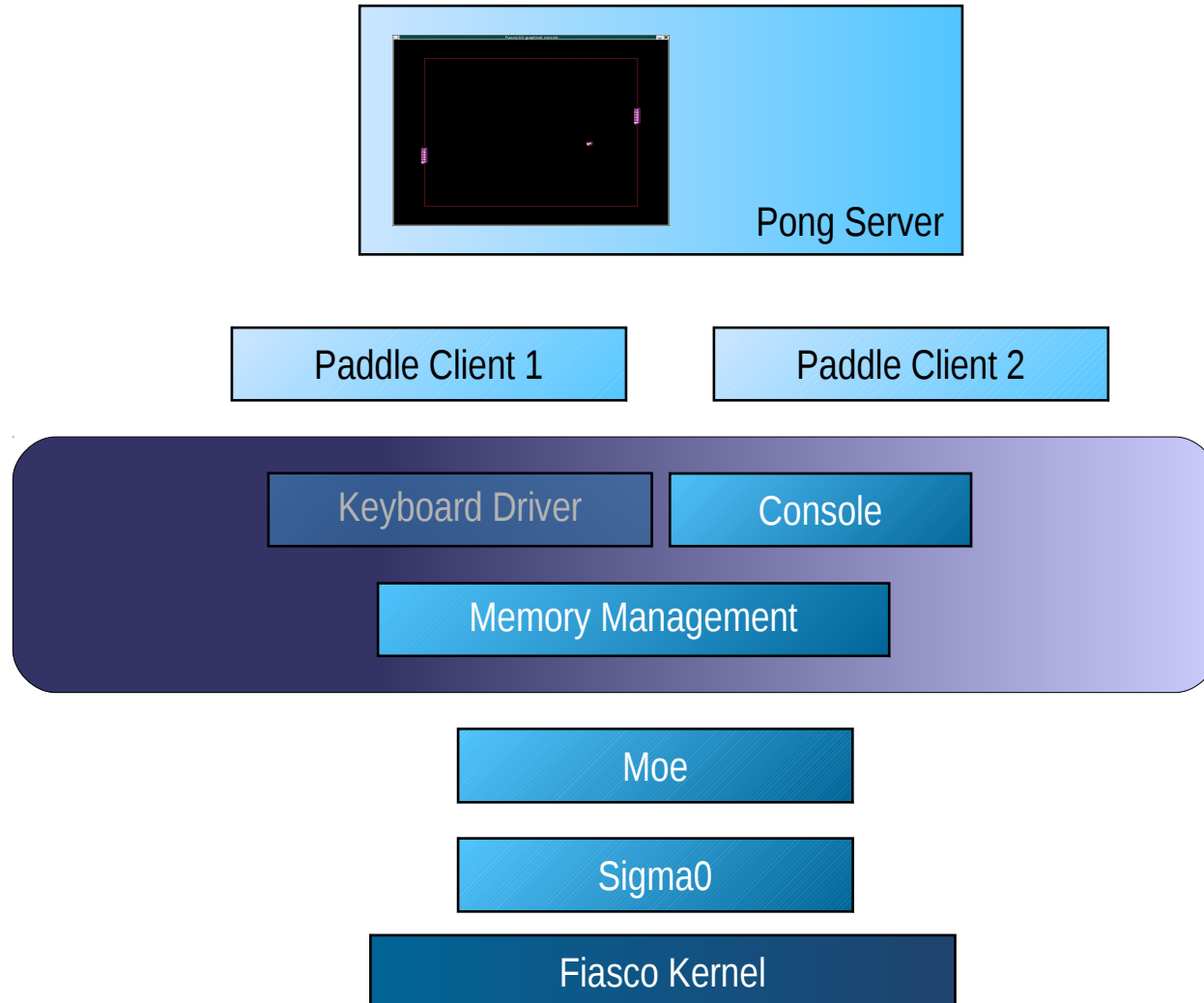
Paddle Client 2

Memory Management

Moe

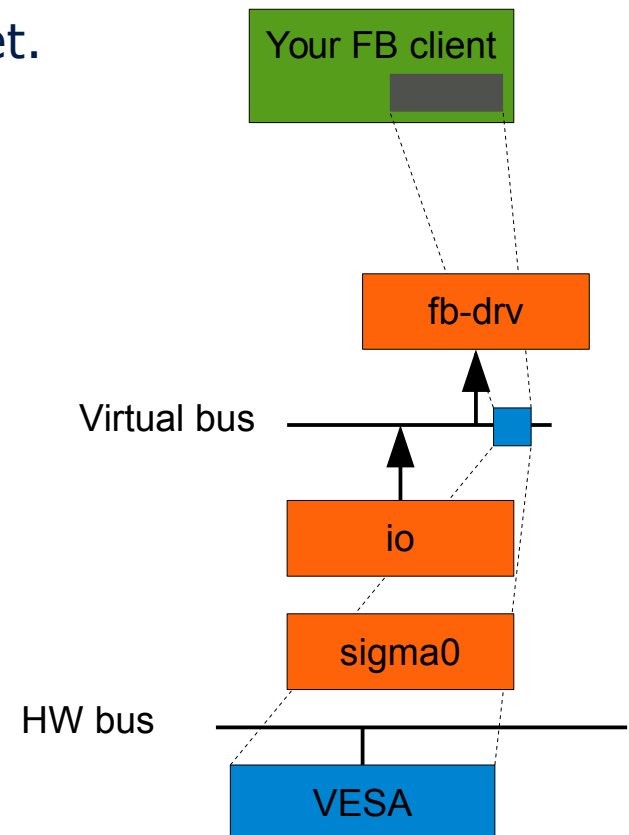
Sigma0

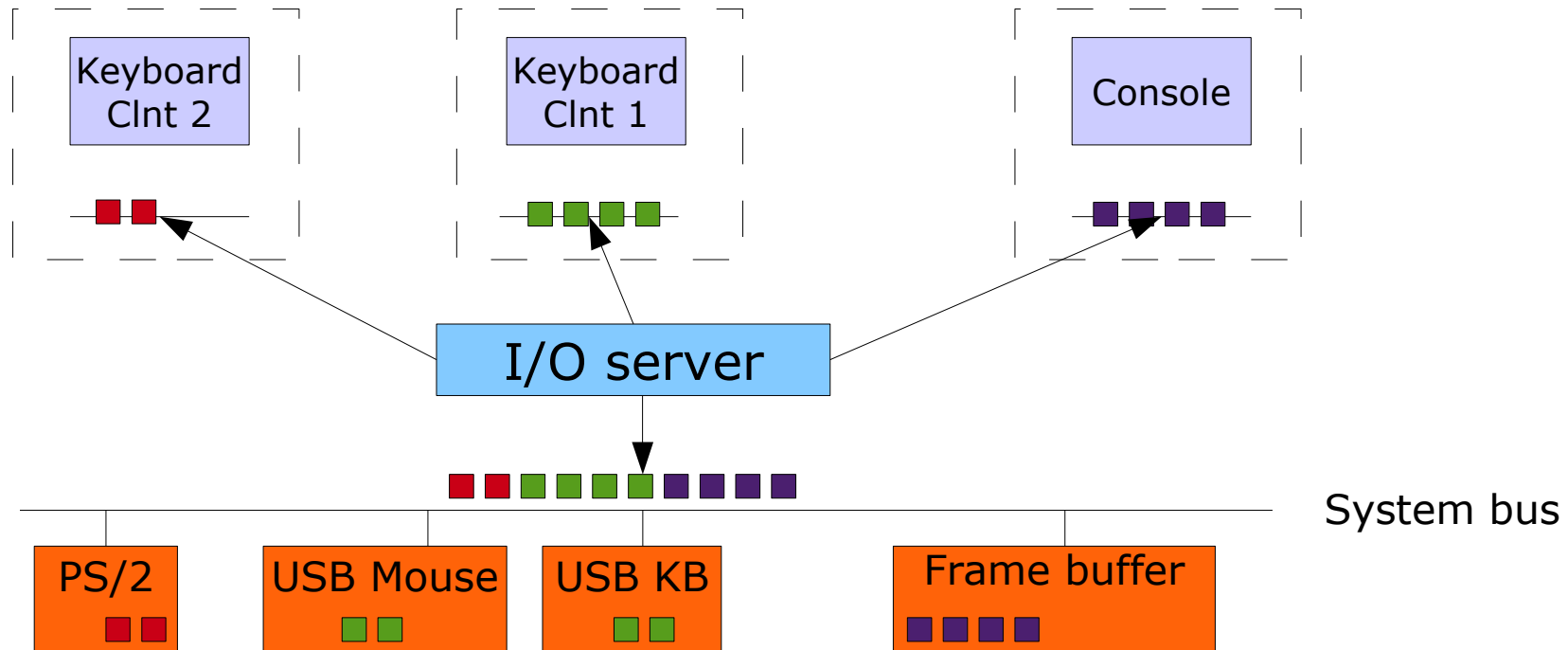
Fiasco Kernel



- VESA – Video Electronics Standards Association
- Standard: XGA BIOS extension
 - Put computer into XGA mode
 - At bootup or at runtime
 - Requires execution of wicked real-mode code
 - GRUB: `vbeset <mode>`
 - L4 FB-DRV: command line option `-m <mode>`
 - Mode: 0x100 – 0x11F → see e.g., Wikipedia on “VESA BIOS extensions”
 - Get access to HW frame buffer
 - Physical address
 - Size and color info
 - Render graphics into frame buffer

- VESA belongs to legacy x86 device set.
- IO server manages all I/O resources.
- fb-drv server provides a frame buffer interface to its clients
- In your app:
 - Query “fb” to get an L4Re::Goos_fb
 - Get FB info and FB memory
 - Directly render into FB mem





- Package in SVN: fb-drv
- IO
 - IO service
 - Configured through IO config files passed on command line
 - You'll need two:
 - src/l4/conf/x86-legacy.devs → static configuration of some common x86 devices
 - src/l4/conf/examples/x86-fb.io → description of a "virtual" PCI bus containing the VGA device
- Fb-drv
 - Maps VESA fb to an L4Re::Util::Video::Goos_fb

```
require ("L4");

local l = L4.default_loader;

fbdrv_vbus = l:new_channel(); -- for IO's virtual device bus
fbdrv_fb = l:new_channel();  -- for FB-DRV's server interface

l:start( { caps = { fbdrv = fbdrv_vbus:svr(),
                  icu = L4.Env.icu,
                  sigma0 = L4.cast(L4.Proto.Factory,
                                   L4.Env.sigma0):create(
                                   L4.Proto.Sigma0) },
          log = { "IO", "y" }},
        "rom/io rom/x86-legacy.devs rom/x86-fb.io");

l:start( { caps = { vbus=fbdrv_vbus, fb=fbdrv_fb:svr() },
          log = {"fbdrv", "B" }},
        "rom/fb-drv -m 0x117" );

l:start( {caps = {fb=fbdrv_fb}},
        "rom/your_fb_client" );
```

- `l4/pkg/l4re/include/video/view/goos`
`l4/pkg/l4re/util/include/video/goos_fb`
 - `Goos_fb(char const *name)` → setup a Goos frame buffer using a named FB capability (channel to fb-drv)
 - `Goos_fb::view_info()` → get FB info
(`struct L4Re::Video::View::Info`)
 - `Goos_fb::attach_buffer()` → get FB data space attached to memory
 - `Goos_fb::refresh()` → as the name says
 - This is **not** necessary for a physical fb. Why?

- Framebuffer base address:

```
base = fb->attach_buffer();
```

- Address of a pixel (x,y):

```
L4Re::Video::View::Info info;  
int r = fb->view_info(&info);  
if (r != 0) error(..);
```

```
addr = base  
      + y * (bytes per line :=  
              info.pixel_info.bytes_per_pixel  
              * fb_info.width)  
      + x * info.pixel_info.bytes_per_pixel
```

- Setting a pixel:
 `*addr = <color>`
- Colors:
 - `l4re/include/video/colors`
 - `Pixel (Pixel_info) := set of Color_components`
 - `Color_component`:
 - Size in bits
 - Shift within containing type

- `l4/pkg/libgfxbitmap` – C library to render strings into a frame buffer
 - At startup: call `gfxbitmap_font_init();` once
 - Render string to framebuffer:

```
gfxbitmap_font_text(void *fb_address,  
                    l4re_fb_info_t *fbinfo,  
                    gfxbitmap_font_t font,  
                    const char *text, unsigned len,  
                    unsigned x, unsigned y,  
                    gfxbitmap_color_pix_t foreground,  
                    gfxbitmap_color_pix_t background);
```
 - `fb_address` → the address the FB data space is mapped to
 - `fbinfo` → your `L4Re::Framebuffer::Info` struct, can just be cast
 - Colors are unsigned int
 - Rest clear?
- Useful enum values:
`GFXBITMAP_DEFAULT_FONT, GFXBITMAP_USE_STRLEN`

- For graphics, see your favorite Computer Graphics text book (or copy the relevant algorithms from Wikipedia...)
- There's a libpng port in our non-public SVN, contact me if you want access.
- All not necessary to solve the assignment.

- Make your echo server to render text into the physical framebuffer
 - Direct access for now, no need to have virtual screens
- Use libgfxbitmap for font rendering
- Keep track of where text is
 - Scrolling as in a terminal
 - History
 - We don't have input, so we can't scroll upward yet.
 - But we'll have input after the next assignment...

- Next meeting: January 7th 2014
- Keyboard device driver & Putting it all together