Graphics in QEMU.

How the guest display shows up in your desktop window.

Gerd Hoffmann <kraxel@redhat.com>

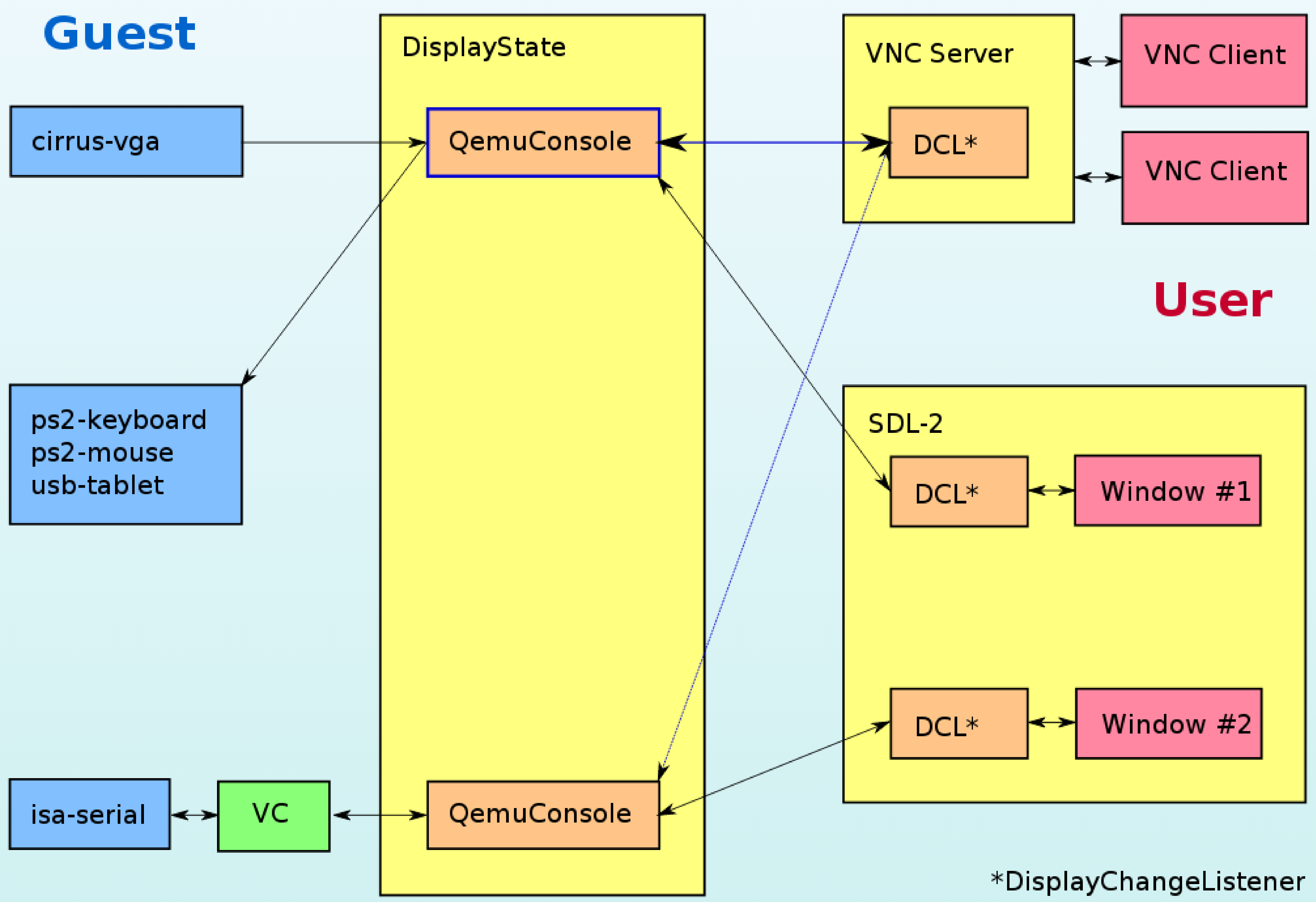
KVM Forum 2014, Düsseldorf, Germany

Outline.

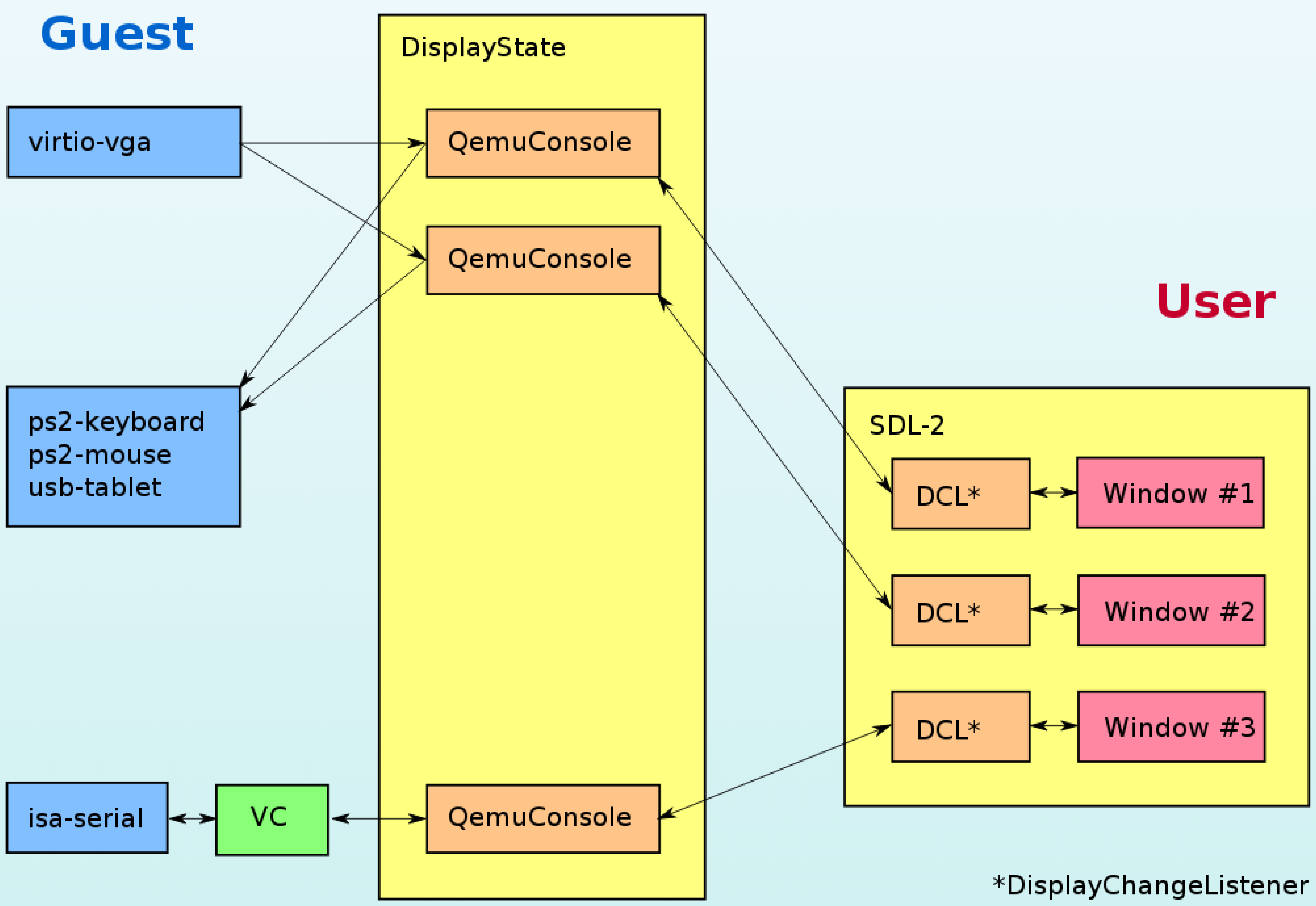
- The big picture.
- Peek into the code.
- Accelerated graphics with opengl.
- Demo.

The big picture.

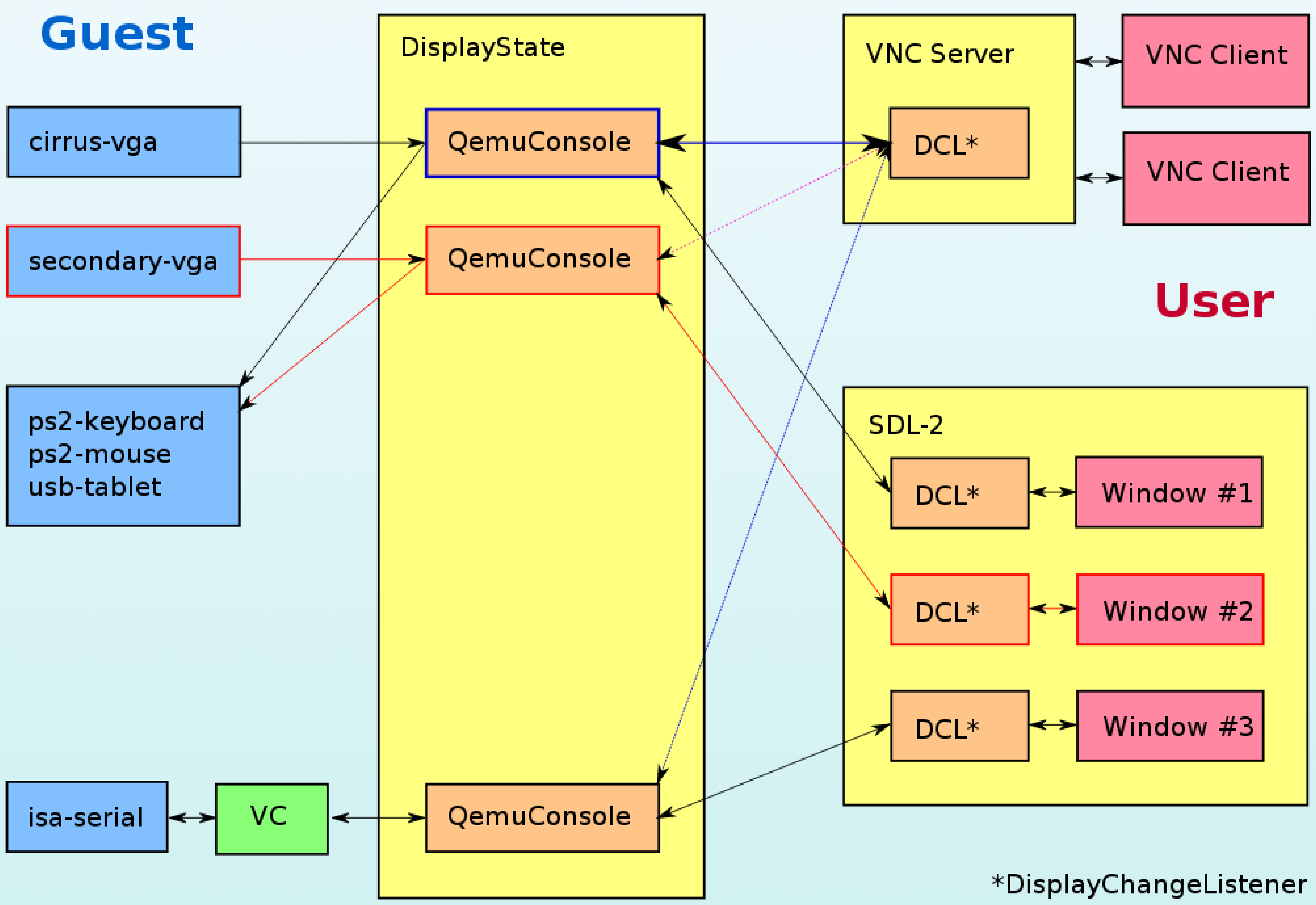
Default x86 guest setup.



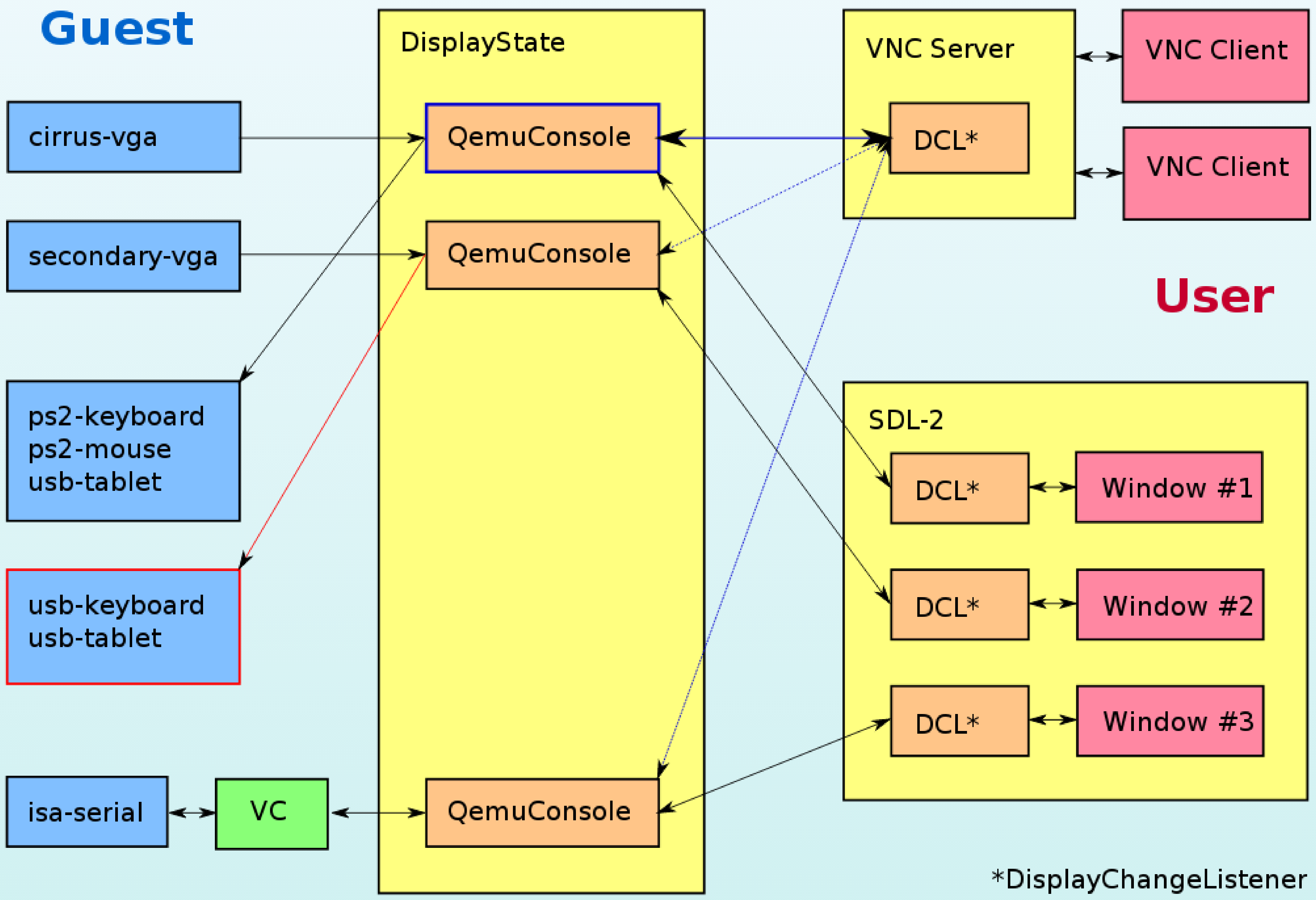
Multihead setup with virtio-gpu.



Multiseat: adding a second display.



Multiseat: adding input devices.



Configure qemu for multiseat.

```
qemu  -enable-kvm $memory $disk $whatever \
      -display gtk \
      -vga std -usb -device usb-tablet \
      -device pci-bridge,addr=12.0,chassis_nr=2,id=head.2 \
      -device secondary-vga,bus=head.2,addr=02.0,id=video.2 \
      -device nec-usb-xhci,bus=head.2,addr=0f.0,id=usb.2 \
      -device usb-kbd,bus=usb.2.0,port=1,display=video.2 \
      -device usb-tablet,bus=usb.2.0,port=2,display=video.2
```

In the guest:

```
[root@fedora ~]# cat /etc/udev/rules.d/70-qemu-autoseat.rules
SUBSYSTEMS=="pci", DEVPATH=="*/0000:00:12.0", TAG+="seat", ENV{ID_AUTOSEAT}="1"
```

More documentation is in docs/multiseat.txt.

Peek into the code.

hw: Initialize the virtual vga.

```
static const GraphicHwOps qxl_ops = {
    .gfx_update  = qxl_hw_update, // called by graphic_hw_update();
};

static int qxl_init_primary(PCIDevice *dev)
{
    QemuConsole *con;

    /* ... */
    con = graphic_console_init(DEVICE(dev), 0, &qxl_ops, qxl);
    /* ... */
}
```

ui: Register DisplayChangeListener.

vnc, following active_console.

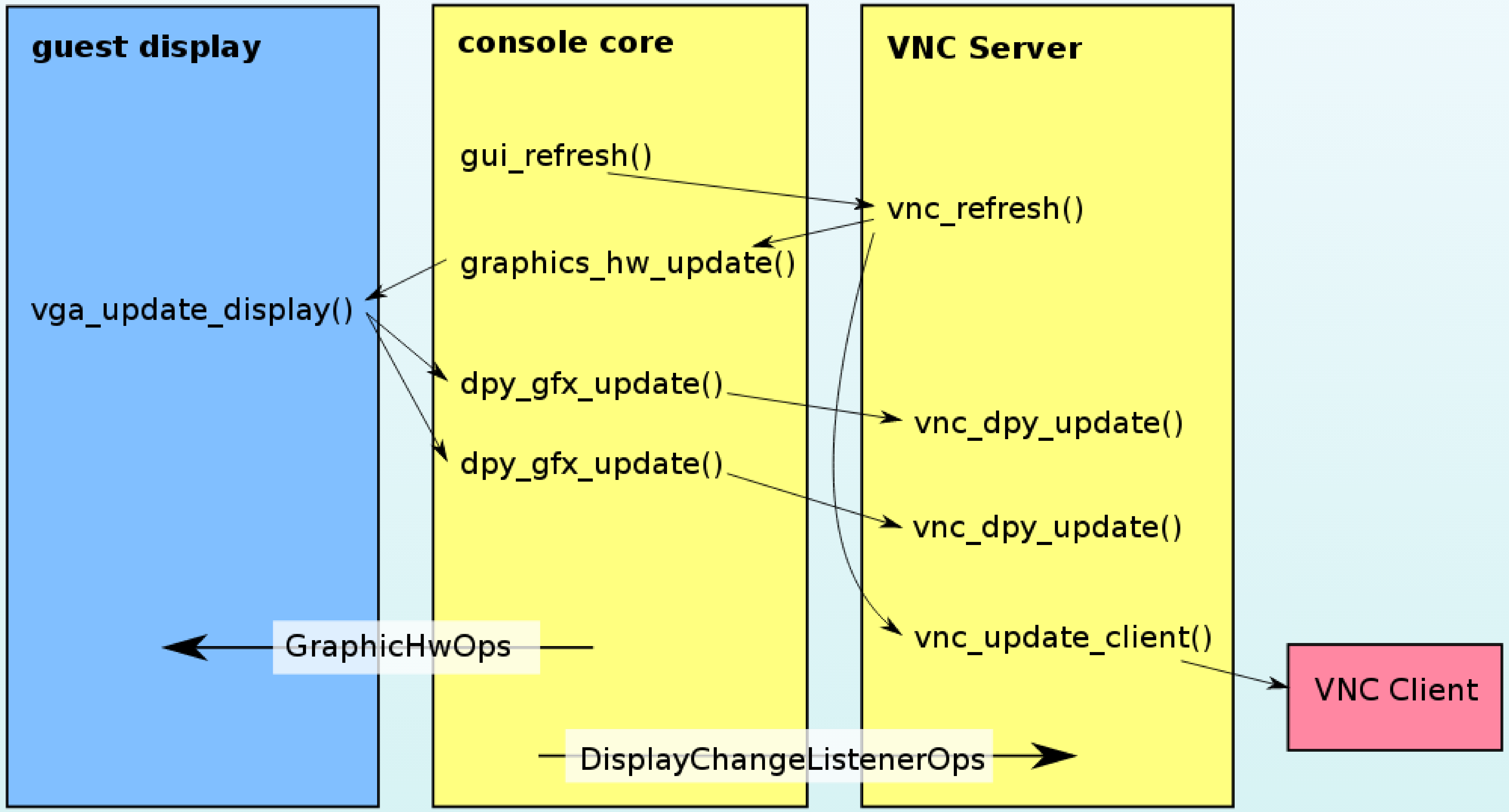
```
static const DisplayChangeListenerOps dcl_ops = {
    .dpy_name          = "vnc",
    .dpy_refresh       = vnc_refresh,    // called by timer
    .dpy_gfx_switch    = vnc_dpy_switch, // dpy_gfx_replace_surface();
    .dpy_gfx_update    = vnc_dpy_update, // dpy_gfx_update();
    /* ... */
};

void vnc_display_init(DisplayState *ds)
{
    VncDisplay *vs = g_malloc0(sizeof(*vs));

    /* ... */
    vs->dcl.ops = &dcl_ops;
    register_displaychangelistener(&vs->dcl);
}
```

ui: Register DisplayChangeListener

sdl2, one window per QemuConsole.

```
void sdl_display_init(/* ... */)
{
    /* ... */
    for (i = 0; i < sdl2_num_outputs; i++) {
        QemuConsole *con = qemu_console_lookup_by_index(i);
        if (!qemu_console_is_graphic(con)) {
            sdl2_console[i].hidden = true;
        }
        sdl2_console[i].idx = i;
        sdl2_console[i].dcl.ops = &dcl_ops;
        sdl2_console[i].dcl.con = con;
        register_displaychangelistener(&sdl2_console[i].dcl);
    }
    /* ... */
}
```

Display update cycle.



The DisplaySurface (where the data lives).

```
struct DisplaySurface {
    pixman_format_code_t format;
    pixman_image_t *image;
    uint8_t flags;
};

void dpy_gfx_replace_surface(QemuConsole *con,
                             DisplaySurface *surface);
```

Creating a DisplaySurface.

```
/* backed by host memory (vga text mode) */
DisplaySurface *qemu_create_displaysurface(int width, int height);

/* backed by device (vga) memory */
DisplaySurface *qemu_create_displaysurface_from
    (int width, int height, pixman_format_code_t format,
     int linesize, uint8_t *data);

/* backed by guest main memory */
DisplaySurface *qemu_create_displaysurface_guestmem
    (int width, int height, pixman_format_code_t format,
     int linesize, uint64_t addr);
```

Input event routing.

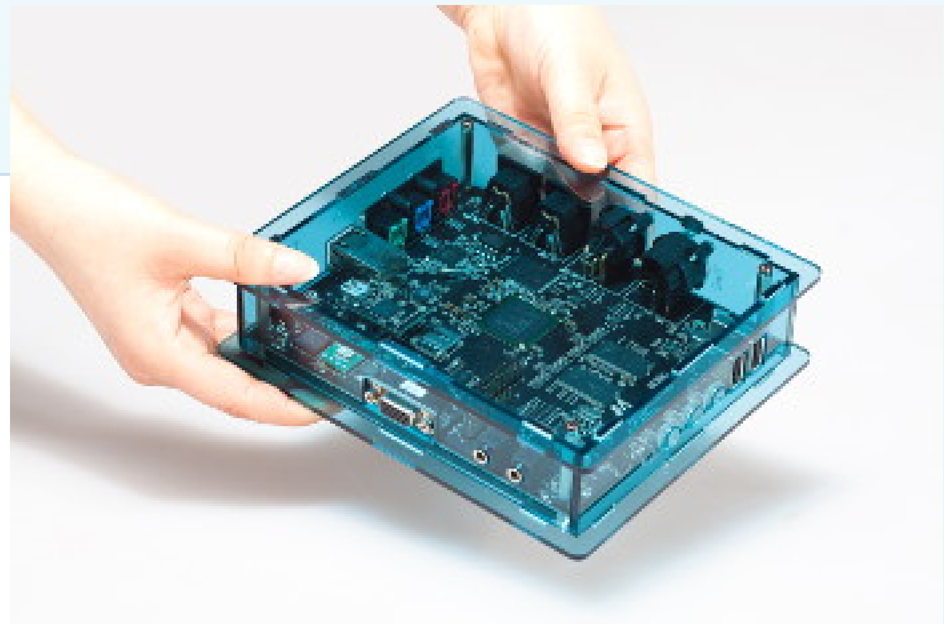
```
/* setup input routing (hw) */
void qemu_input_handler_bind(QemuInputHandlerState *s,
                             const char *device_id, int head,
                             Error **errp);

/* core input event function (ui) */
void qemu_input_event_send(QemuConsole *src, InputEvent *evt);

/* various helper functions for specific events (ui) */
void qemu_input_event_send_key_qcode(QemuConsole *src, QKeyCode q, bool down);
/* ... */
```

- Console core code is not involved in input event delivery.
- Input layer uses QemuConsole pointers to tag the event source.

Accelerated graphics with opengl.

Reason #1: Milkymist One.

Quoting m-labs.hk/m1.html:



> The Milkymist One is an experimental hardware appliance for live video effects. [...]
> The LM32 microprocessor is assisted by a texture mapping unit and a programmable floating point VLIW coprocessor [...]

QEMU emulates the texture mapping unit today by rendering into a texture using opengl, then copy back the data from the texture. Requires X11 server access for glx.

Reason #2: virgl (virtio-gpu with opengl support).

- Every modern desktop uses opengl for rendering.
- Browsers do it too.
- So we want offload that to the hardware, even when running in a virtual machine.

Reason #3: vGPU (gpu virtualization).

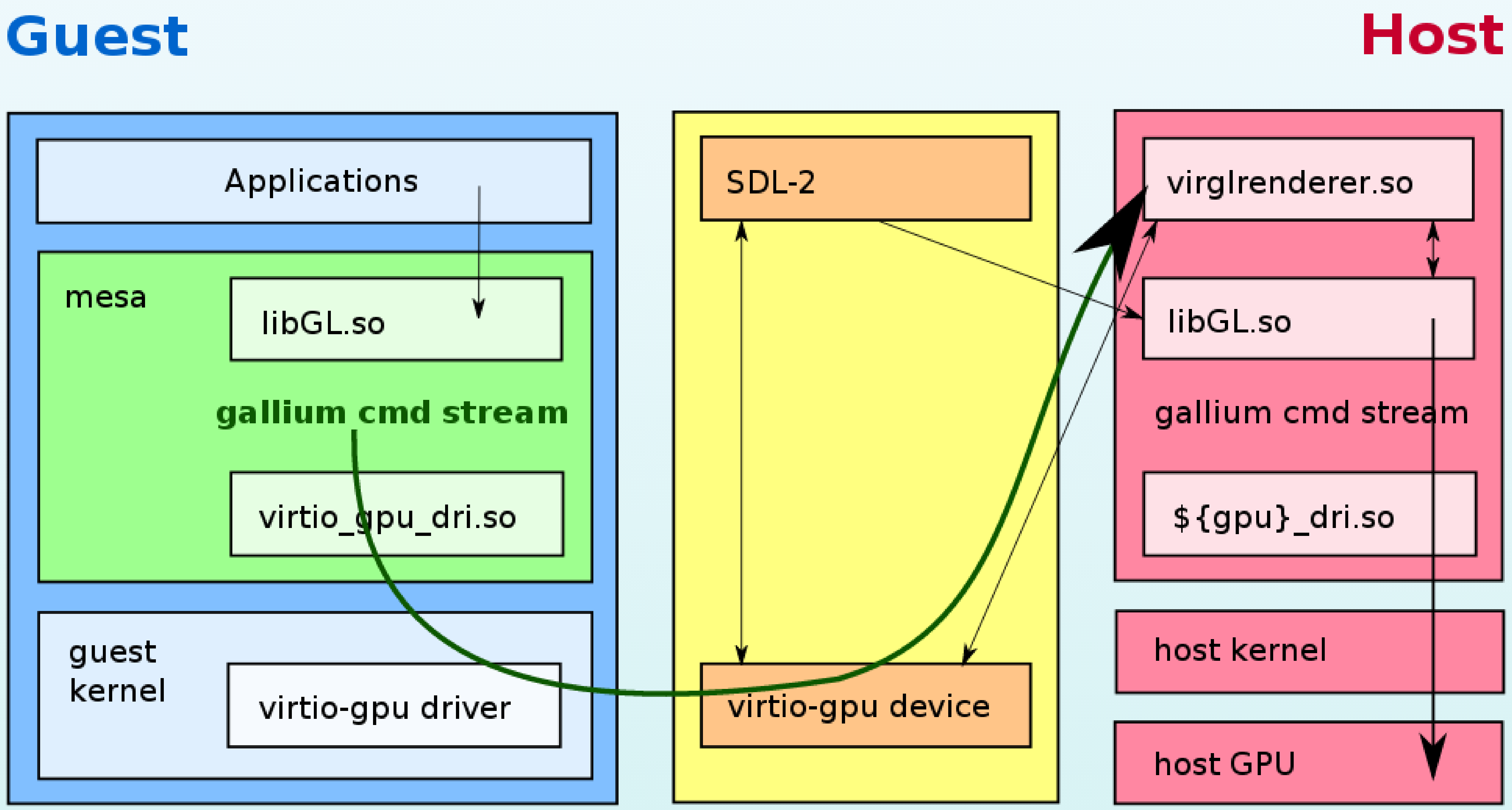
- Emulate real GPU (unlike paravirtual virtio-gpu), with the help of the host GPU.
- Vendor specific, i.e. emulating a Intel GPU for the guest requires a Intel GPU on the host.
- Intel (see KvmGT talk) and Nvidia are working on this.

Adding opengl bits to console core.

```
/* opengl context management */
qemu_gl_context dpy_gl_ctx_create(QemuConsole *con, bool shared);
void dpy_gl_ctx_destroy(QemuConsole *con, qemu_gl_context ctx);
int dpy_gl_ctx_make_current(QemuConsole *con, qemu_gl_context ctx);
qemu_gl_context dpy_gl_ctx_get_current(QemuConsole *con);

/* define and update guest display */
void dpy_gl_scanout(QemuConsole *con,
                    uint32_t backing_id, bool backing_y_0_top,
                    uint32_t x, uint32_t y, uint32_t w, uint32_t h);
void dpy_gl_update(QemuConsole *con,
                   uint32_t x, uint32_t y, uint32_t w, uint32_t h);
```

- DisplayConsoleListenerOps is likewise extended.
- backing_id is a opengl texture id.
- Might change as milkymist & vGPU are added to the picture, only virtio-gpu works today.

virtio-gpu rendering workflow.



Working today.

With "working" as in "demoable patches exist"

- hw: virtio-gpu (with 3D mode).
- ui: SDL-2.

.

To be implemented.

Hardware emulation:

- Integrate milkymist one.
- Integrate vGPU.

In the ui code:

- Add opengl support to gtk ui.
- Render without X11 display, into dma-bufs (using drm render nodes).
- Add viewer app, accepting those dma-bufs.
- Spice integration: new display channel type, basically passing dma-buf handles (only for the local case, i.e. spice-client + qemu running on the same machine).
- Allow blitting classic DisplaySurfaces using the opengl code paths.

To be investigated: remote display.

Simple approach:

- Just read from rendered texture, like we read from DisplaySurface today.
- Not exactly most efficient way ...
- We'll probably do that anyway for compatibility with older spice clients and vnc.

Offload to the GPU (better sapproach?):

- Encode guest display as video stream (one more spice display channel type ...).
- Problem: Hardware tends to support H.264 only, which is a patent minefield.

Other ideas?

Demo

Resources

Slides

- Online: https://www.kraxel.org/slides/qemu-gfx/
- As pdf: https://www.kraxel.org/slides/qemu-gfx.pdf

git repos

- kernel: https://www.kraxel.org/cgit/linux/log/?h=virtio-gpu-rebase
- qemu: https://www.kraxel.org/cgit/qemu/log/?h=rebase/vga-wip
- mesa: http://cgit.freedesktop.org/~airlied/mesa/log/?h=renderer-1-wip
- xorg: http://cgit.freedesktop.org/~airlied/xf86-video-virgl/

Documentation

- Dave's build instructions: https://docs.google.com/document/d/1CNiN0rHdfh7cp9tQ3coebNEJtHJzm4OCWvF3qL4nucc/pub. They are a bit dated, but still work when you ignore the (bitrotted) spice bits and use the repos listed above to get SDL-2 going.