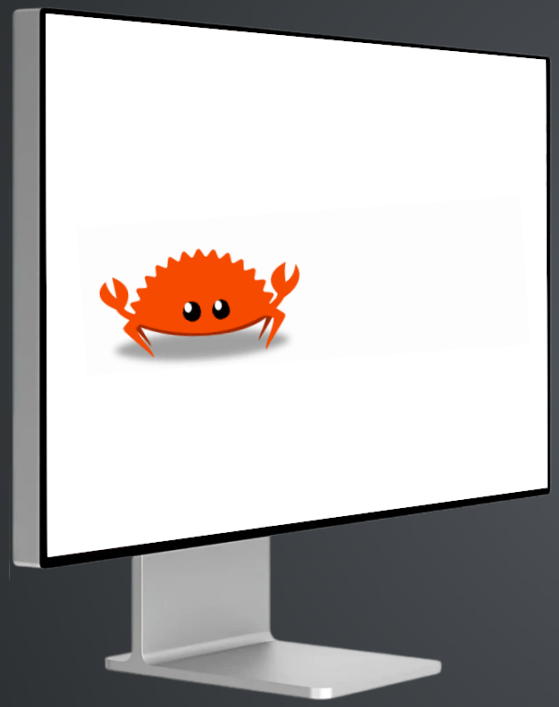
Linux support for controlling displays in Rust



Kangrejos 2025

Rahul Rameshbabu or Binary-Eater

What does "controlling a display" mean?

What does "controlling a display" mean?

Let's use this monitor to illustrate

What does "controlling a display" mean?

Change the brightness

What does "controlling a display" mean?

Adjust the volume of the speaker hw

What does "controlling a display" mean?

Selecting the input source

What does "controlling a display" mean?

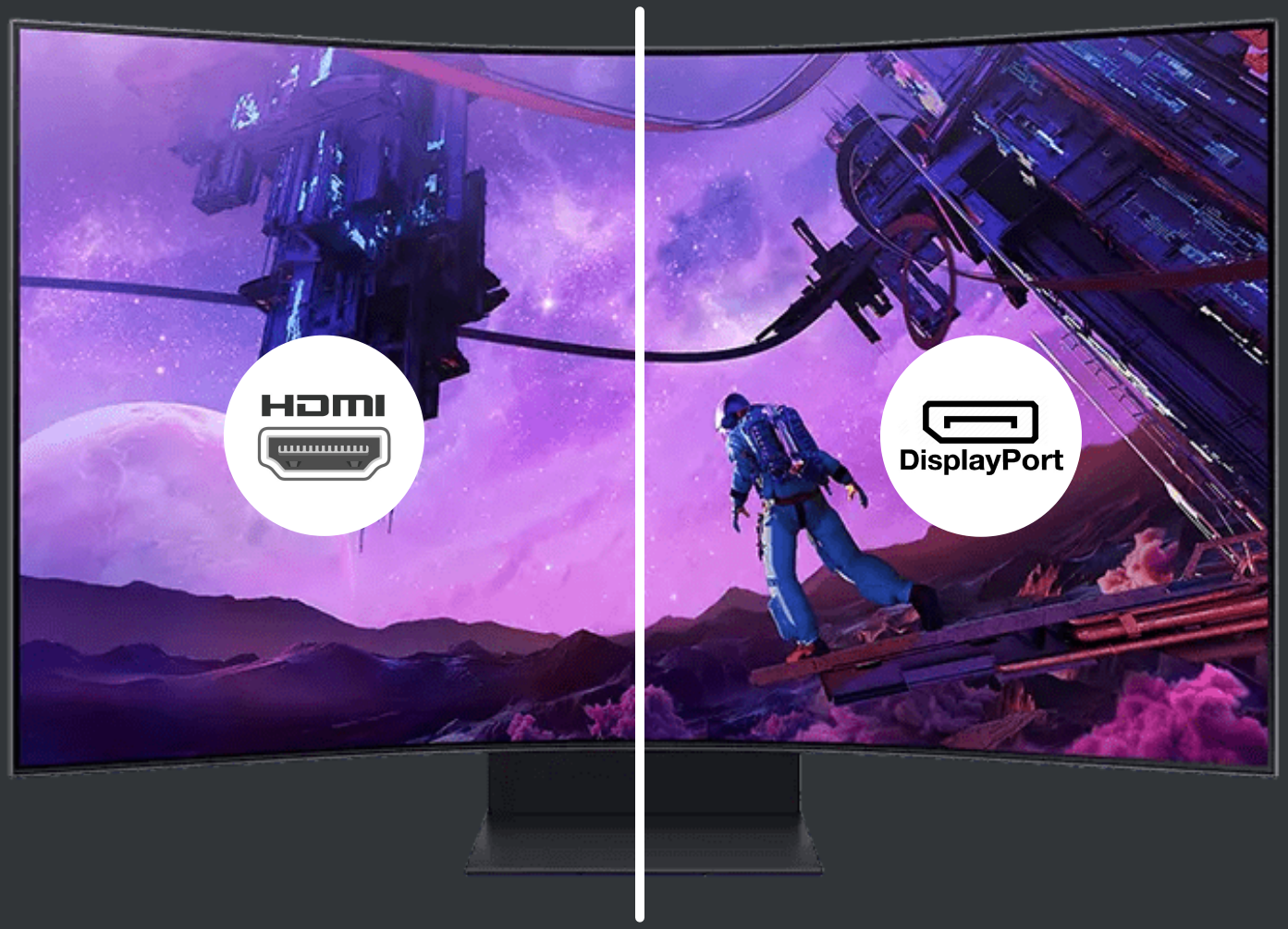
Selecting the input source

What does "controlling a display" mean?

Selecting the input source

What does "controlling a display" mean?

Selecting the input source



What does this have to do with the Linux kernel?

Isn't this what a monitor's on-screen display (OSD) is for?

Unfortunately, some monitors don't have a OSD

Introducing the Apple Studio Display and Apple Pro Display XDR

Unfortunately, some monitors don't have a OSD

Introducing the Apple Studio Display and Apple Pro Display XDR

Apple Studio Display

Unfortunately, some monitors don't have a OSD

Introducing the Apple Studio Display and Apple Pro Display XDR

Apple Studio Display

Unfortunately, some monitors don't have a OSD

Introducing the Apple Studio Display and Apple Pro Display XDR

Apple Studio Display

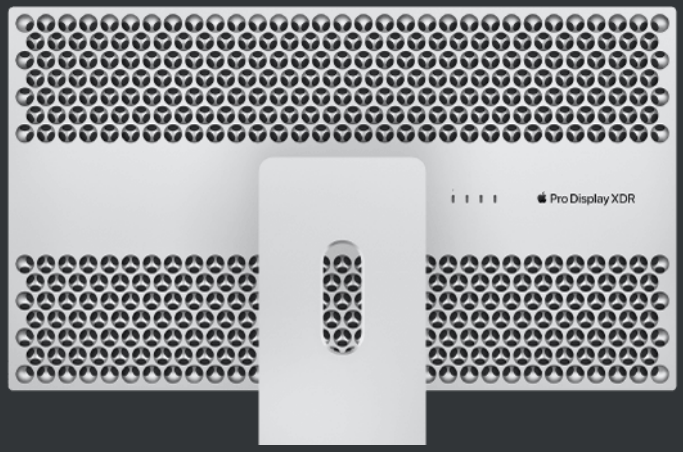
Apple Pro Display XDR

Unfortunately, some monitors don't have a OSD

Introducing the Apple Studio Display and Apple Pro Display XDR

Apple Studio Display

Apple Pro Display XDR

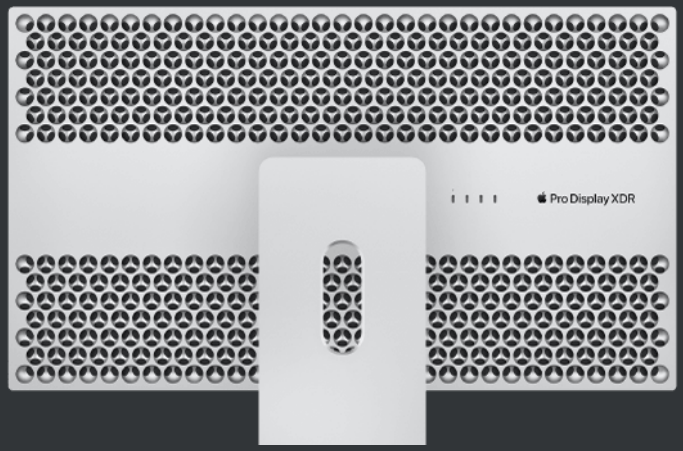


Unfortunately, some monitors don't have a OSD

Introducing the Apple Studio Display and Apple Pro Display XDR

Apple Studio Display

Apple Pro Display XDR



A computer can talk to a monitor to change these settings

Let’s delve into the mechanisms that the operating system can use to manage a monitor

Let’s delve into the mechanisms that the operating system can use to manage a monitor

- **Monitor Control Command Set (MCCS)** describes a high-level communication protocol for controlling properties of a monitor from a host system

Let’s delve into the mechanisms that the operating system can use to manage a monitor

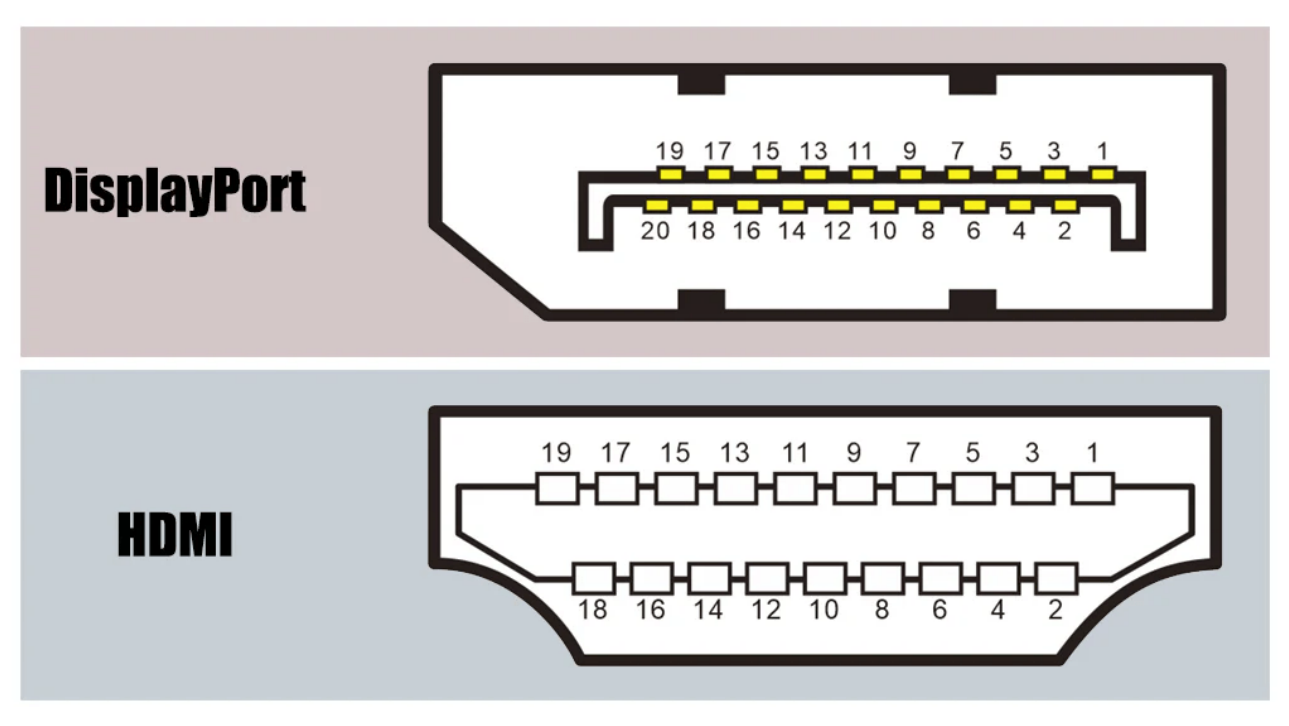
- **Monitor Control Command Set (MCCS)** describes a high-level communication protocol for controlling properties of a monitor from a host system

- A **virtual control panel (VCP)** code represents a single instruction in the **MCCS** for controlling a property of the monitor

Let’s delve into the mechanisms that the operating system can use to manage a monitor

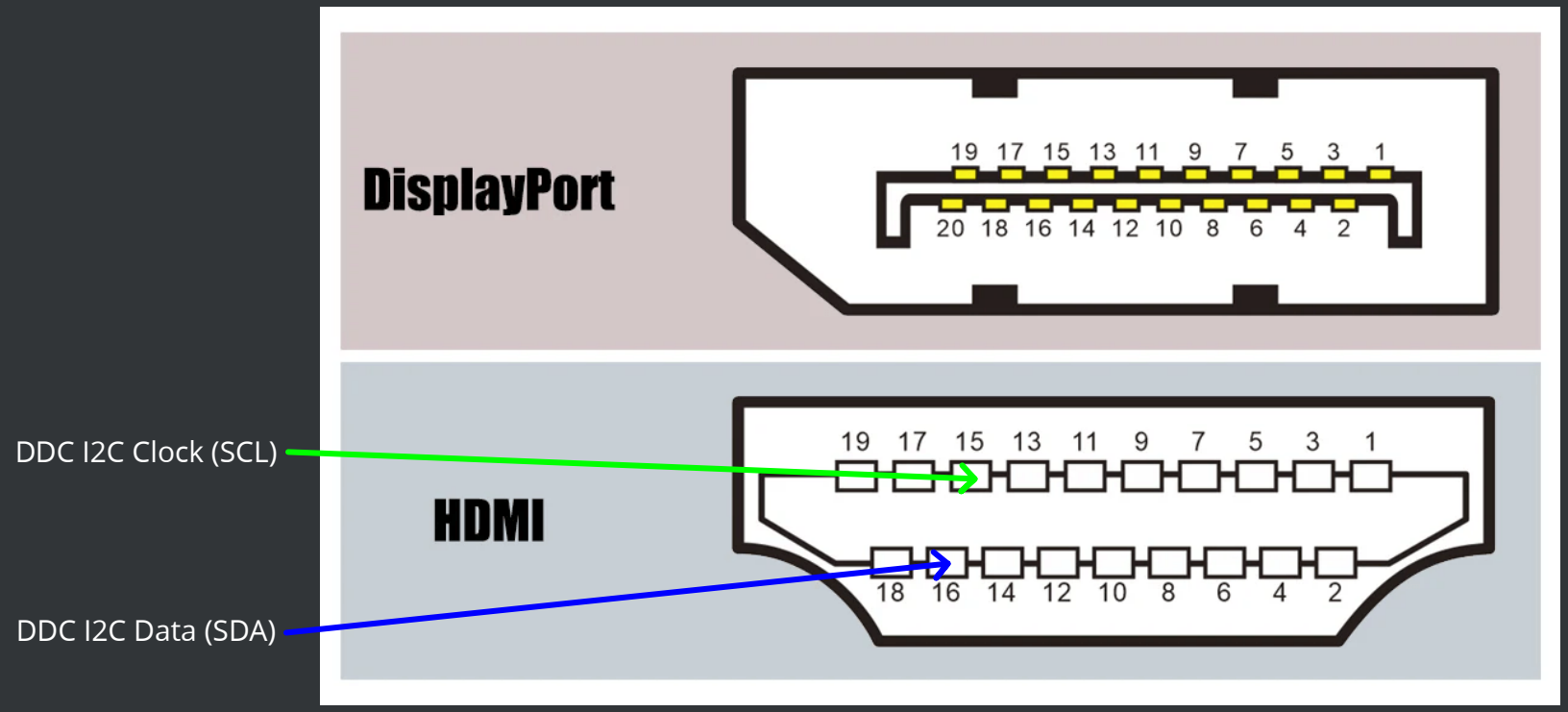
- **Monitor Control Command Set (MCCS)** describes a high-level communication protocol for controlling properties of a monitor from a host system
- A **virtual control panel (VCP)** code represents a single instruction in the **MCCS** for controlling a property of the monitor
- **MCCS** is required to be implemented over an underlying bi-directional bus

Let's understand the buses available for MCCS



Display Data Channel/Common Interface (DDC/CI) is the most common bus for implementing **MCCS**. **DDC/CI** uses I2C transactions for the communication.

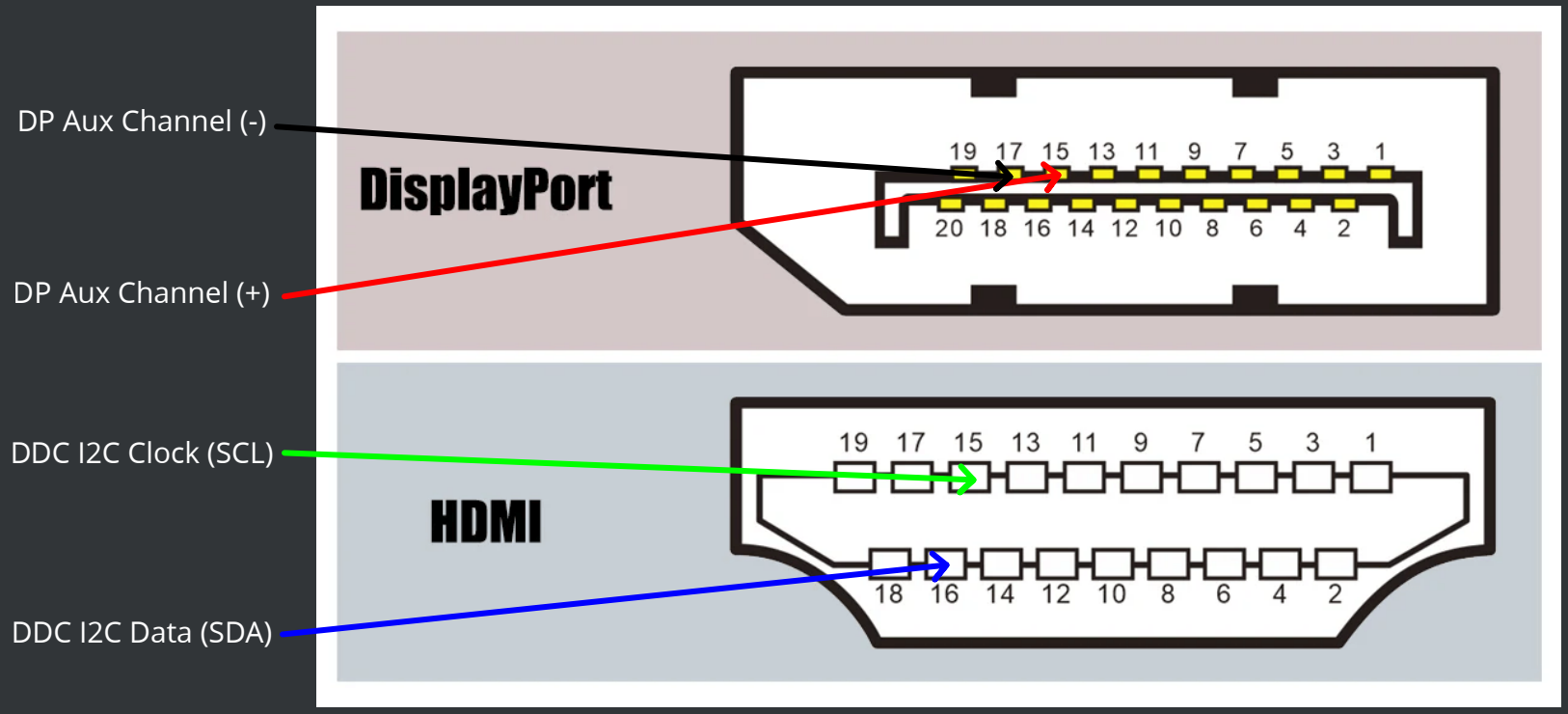
Let's understand the buses available for MCCS



Display Data Channel/Common Interface (DDC/CI) is the most common bus for implementing **MCCS**. **DDC/CI** uses I2C transactions for the communication.

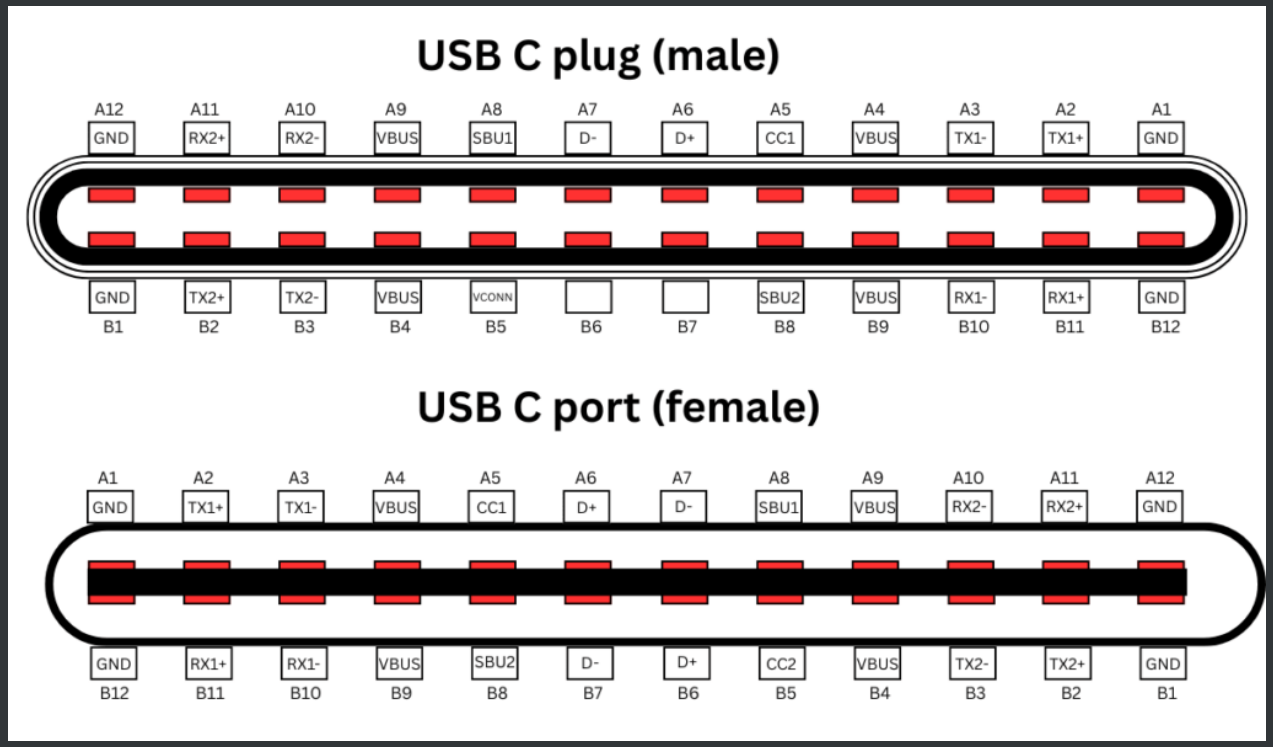
Let's understand the buses available for MCCS

NOTE: DisplayPort does not expose physical I2C, so **DDC/CI** I2C transactions are encapsulated over the DP Aux channel



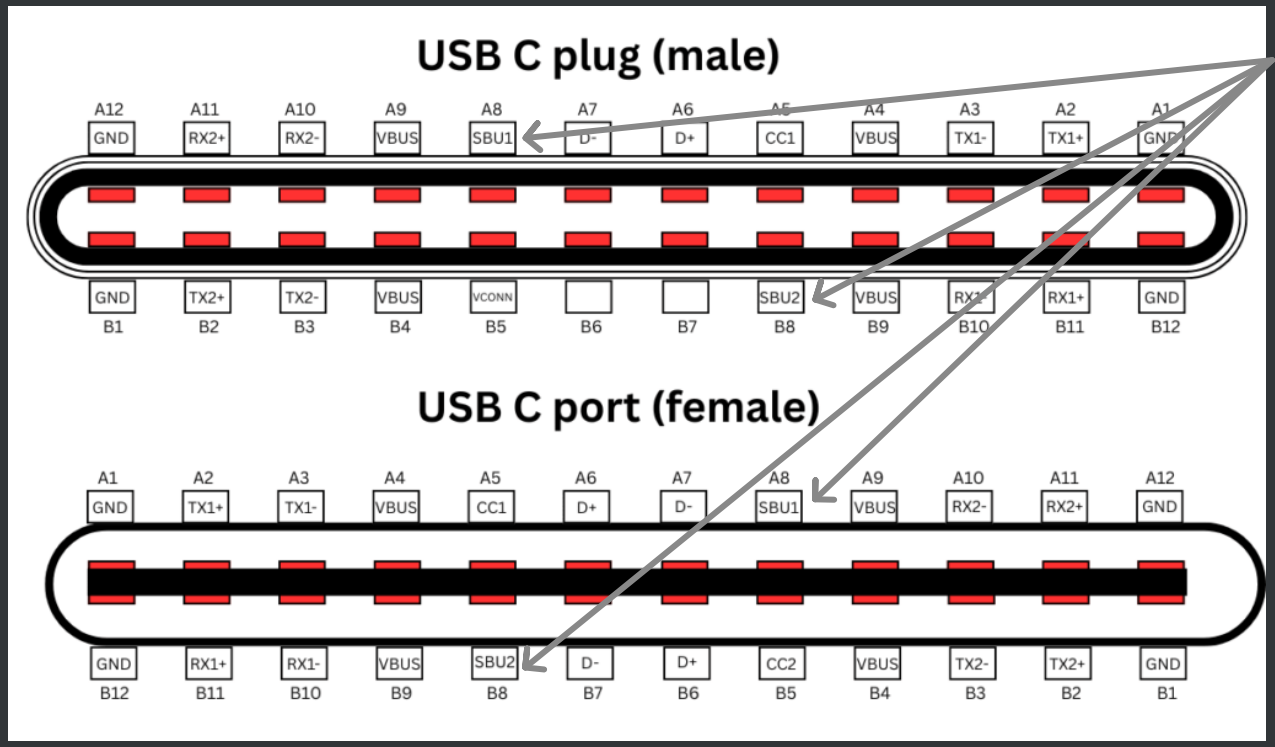
Display Data Channel/Common Interface (DDC/CI) is the most common bus for implementing **MCCS**. **DDC/CI** uses I2C transactions for the communication.

Let's understand the buses available for MCCS



DDC/CI over DP aux channel adds complications involving the abstraction of I2C. Back in 1998, the USB consortium came up with **USB Monitor Control Class** specification for **MCCS** over **HID**. USB-C monitors have made it relevant.

Let's understand the buses available for MCCS



DP Aux Channel (negotiated polarity)

DDC/CI is supported over the

DP aux channel exposed in

USB-C DisplayPort Alt Mode

DDC/CI over DP aux channel adds complications involving the abstraction of I2C. Back in 1998, the USB consortium came up with **USB Monitor Control Class** specification for **MCCS** over **HID**. USB-C monitors have made it relevant.

Delving into USB Monitor Control Class

Let's analyze a decoded HID report descriptor from an Apple Studio Display

```
Usage Page (Monitor VESA VCP),
Usage (10h, Brightness),
Logical Minimum (400),
Logical Maximum (60000),
Unit (Centimeter^-2 * Candela),
Unit Exponent (14),
Report Size (32),
Report Count (1),
Feature (Variable, Null State),
```

- USB Monitor Control Class Specification
- Device Class Definition for Human Interface Devices (HID)

Delving into USB Monitor Control Class

Let's analyze a decoded HID report descriptor from an Apple Studio Display

This is a HID usage page describing the brightness VCP code (0x10)

```
Usage Page (Monitor VESA VCP),
Usage (10h, Brightness),
Logical Minimum (400),
Logical Maximum (60000),
Unit (Centimeter^-2 * Candela),
Unit Exponent (14),
Report Size (32),
Report Count (1),
Feature (Variable, Null State),
```

- USB Monitor Control Class Specification
- Device Class Definition for Human Interface Devices (HID)

Delving into USB Monitor Control Class

Let's analyze a decoded HID report descriptor from an Apple Studio Display

This is a HID usage page describing the brightness VCP code (0x10)

```
1 Usage Page (Monitor VESA VCP),
2 Usage (10h, Brightness),
3 Logical Minimum (400),
4 Logical Maximum (60000),
5 Unit (Centimeter^-2 * Candela),
6 Unit Exponent (14),
7 Report Size (32),
8 Report Count (1),
9 Feature (Variable, Null State),
```

The unit is 10^-2 candela (luminosity) while the exponent is 10^14

- USB Monitor Control Class Specification
- Device Class Definition for Human Interface Devices (HID)

Delving into USB Monitor Control Class

Let's analyze a decoded HID report descriptor from an Apple Studio Display

```
Usage Page (Monitor VESA VCP),
Usage (10h, Brightness),
Logical Minimum (400),
Logical Maximum (60000),
Unit (Centimeter^-2 * Candela),
Unit Exponent (14),
Report Size (32),
Report Count (1),
Feature (Variable, Null State),
```

This is a HID usage page describing the brightness VCP code (0x10)

The minimum and maximum values accepted for the brightness command

The unit is 10^-2 candela (luminosity) while the exponent is 10^14

- USB Monitor Control Class Specification
- Device Class Definition for Human Interface Devices (HID)

Delving into USB Monitor Control Class

Let's analyze a decoded HID report descriptor from an Apple Studio Display

```
1 Usage Page (Monitor VESA VCP),
2 Usage (10h, Brightness),
3 Logical Minimum (400),
4 Logical Maximum (60000),
5 Unit (Centimeter^-2 * Candela),
6 Unit Exponent (14),
7 Report Size (32),
8 Report Count (1),
9 Feature (Variable, Null State),
```

This is a HID usage page describing the brightness VCP code (0x10)

The minimum and maximum values accepted for the brightness command

The unit is 10^-2 candela (luminosity) while the exponent is 10^14

There is a single brightness control that takes a 32-bit (4-bytes) value

- USB Monitor Control Class Specification
- Device Class Definition for Human Interface Devices (HID)

Delving into USB Monitor Control Class

Let's analyze a decoded HID report descriptor from an Apple Studio Display

```
1 Usage Page (Monitor VESA VCP),
2 Usage (10h, Brightness),
3 Logical Minimum (400),
4 Logical Maximum (60000),
5 Unit (Centimeter^-2 * Candela),
6 Unit Exponent (14),
7 Report Size (32),
8 Report Count (1),
9 Feature (Variable, Null State),
```

This is a HID usage page describing the brightness VCP code (0x10)

The minimum and maximum values accepted for the brightness command

The unit is 10^-2 candela (luminosity) while the exponent is 10^14

There is a single brightness control that takes a 32-bit (4-bytes) value

Has a variable (changing) value and can have null state, for when the monitor is in standby mode due to **DPMS**.

- USB Monitor Control Class Specification
- Device Class Definition for Human Interface Devices (HID)

Delving into USB Monitor Control Class

Let's analyze a decoded HID report descriptor from an Apple Studio Display

```
1 Usage Page (Monitor VESA VCP),
2 Usage (10h, Brightness),
3 Logical Minimum (400),
4 Logical Maximum (60000),
5 Unit (Centimeter^-2 * Candela),
6 Unit Exponent (14),
7 Report Size (32),
8 Report Count (1),
9 Feature (Variable, Null State),
```

This is a HID usage page describing the brightness VCP code (0x10)

The minimum and maximum values accepted for the brightness command

The unit is 10^-2 candela (luminosity) while the exponent is 10^14

There is a single brightness control that takes a 32-bit (4-bytes) value

Has a variable (changing) value and can have null state, for when the monitor is in standby mode due to **DPMS**.

- USB Monitor Control Class Specification
- Device Class Definition for Human Interface Devices (HID)

A quick peek at other OSes

- Windows has a raw Win32 api for **DDC/CI**
- Apple only implements tightly integrated support for **USB Monitor Control Class** for the external displays they sell.
- The overall state of **MCCS** usage in operating systems is not great, leading to a subpar monitor usage experience. This forces users to either have to fiddle with janky built-in monitor controls or be locked out when none exist.

What about Linux?

This question should be split between internal and external panels

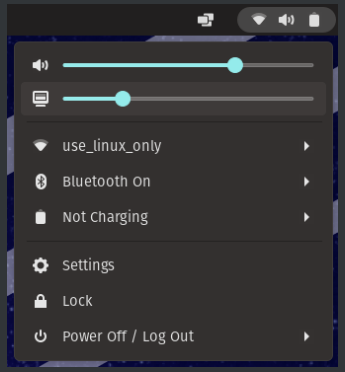
Let's start with internal panels

What does the UAPI and userspace interface look like?

```
/sys/class/backlight/
└── acpi_video0 -> ../../devices/pci0000:00/0000:00:02.0/backlight/acpi_video0
    ├── actual_brightness
    ├── bl_power
    ├── brightness
    ├── device -> ../../../0000:00:02.0
    ├── max_brightness
    ├── power
    ├── scale
    ├── subsystem -> ../../../../../class/backlight  [recursive, not followed]
    ├── type
    └── uevent
```

The sysfs interface for controlling brightness in Linux

The biggest problem with this interface is the lack of ability to identify which connector/monitor does the backlight device belong to



Let's start with internal panels

The kernel API for registering a backlight device

```
1 struct backlight_device *
2 devm_backlight_device_register(struct device *dev, const char *name,
3                   struct device *parent, void *devdata,
4                   const struct backlight_ops *ops,
5                   const struct backlight_properties *props);
```

Helper to register backlight device

Let's start with internal panels

The kernel API for registering a backlight device

```
1  struct backlight_ops {
2      /**
3       * @update_status: Operation called when properties
   have changed.
4       */
5      int (*update_status)(struct backlight_device *);
6
7      /**
8       * @get_brightness: Return the current backlight
   brightness.
9       */
10     int (*get_brightness)(struct backlight_device *);
11
12     /**
13      * @controls_device: Check against the display device
14      */
15     bool (*controls_device)(struct backlight_device *bd,
   struct device *display_dev);
16 };
```

```
1  struct backlight_device *
2  devm_backlight_device_register(struct device *dev, const char *name,
3                                 struct device *parent, void *devdata,
4                                 const struct backlight_ops *ops,
5                                 const struct backlight_properties *props);
```

Helper to register backlight device

Let's start with internal panels

The kernel API for registering a backlight device

```
1  struct backlight_ops {
2      /**
3       * @update_status: Operation called when properties
   have changed.
4       */
5      int (*update_status)(struct backlight_device *);
6
7      /**
8       * @get_brightness: Return the current backlight
   brightness.
9       */
10     int (*get_brightness)(struct backlight_device *);
11
12     /**
13      * @controls_device: Check against the display device
14      */
15     bool (*controls_device)(struct backlight_device *bd,
   struct device *display_dev);
16 };
```

Called to update the hardware state

```
1  struct backlight_device *
2  devm_backlight_device_register(struct device *dev, const char *name,
3                                 struct device *parent, void *devdata,
4                                 const struct backlight_ops *ops,
5                                 const struct backlight_properties *props);
```

Helper to register backlight device

Let's start with internal panels

The kernel API for registering a backlight device

```
1  struct backlight_ops {
2      /**
3       * @update_status: Operation called when properties
   have changed.
4       */
5      int (*update_status)(struct backlight_device *);
6
7      /**
8       * @get_brightness: Return the current backlight
   brightness.
9       */
10     int (*get_brightness)(struct backlight_device *);
11
12     /**
13      * @controls_device: Check against the display device
14      */
15     bool (*controls_device)(struct backlight_device *bd,
   struct device *display_dev);
16 };
```

Called to update the hardware state

Useful if hardware cannot support full integral range for brightness

```
1 struct backlight_device *
2 devm_backlight_device_register(struct device *dev, const char *name,
3                   struct device *parent, void *devdata,
4                   const struct backlight_ops *ops,
5                   const struct backlight_properties *props);
```

Helper to register backlight device

Let's start with internal panels

The kernel API for registering a backlight device

```
1  struct backlight_ops {
2      /**
3       * @update_status: Operation called when properties
   have changed.
4       */
5      int (*update_status)(struct backlight_device *);
6
7      /**
8       * @get_brightness: Return the current backlight
   brightness.
9       */
10     int (*get_brightness)(struct backlight_device *);
11
12     /**
13      * @controls_device: Check against the display device
14      */
15     bool (*controls_device)(struct backlight_device *bd,
   struct device *display_dev);
16 };
```

Called to update the hardware state

Useful if hardware cannot support full integral range for brightness

Check if backlight hardware is for the display

```
1  struct backlight_device *
2  devm_backlight_device_register(struct device *dev, const char *name,
3                                 struct device *parent, void *devdata,
4                                 const struct backlight_ops *ops,
5                                 const struct backlight_properties *props);
```

Helper to register backlight device

Let's start with internal panels

The kernel API for registering a backlight device

```
1  struct backlight_ops {
2      /**
3       * @update_status: Operation called when properties
   have changed.
4       */
5      int (*update_status)(struct backlight_device *);
6
7      /**
8       * @get_brightness: Return the current backlight
   brightness.
9       */
10     int (*get_brightness)(struct backlight_device *);
11
12     /**
13      * @controls_device: Check against the display device
14      */
15     bool (*controls_device)(struct backlight_device *bd,
   struct device *display_dev);
16 };
```

Called to update the hardware state

Useful if hardware cannot support full integral range for brightness

Check if backlight hardware is for the display

```
1 struct backlight_device *
2 devm_backlight_device_register(struct device *dev, const char *name,
3                   struct device *parent, void *devdata,
4                   const struct backlight_ops *ops,
5                   const struct backlight_properties *props);
```

Helper to register backlight device

The `acpi_backlight` kernel parameter

Easily one of the most configured by laptop users

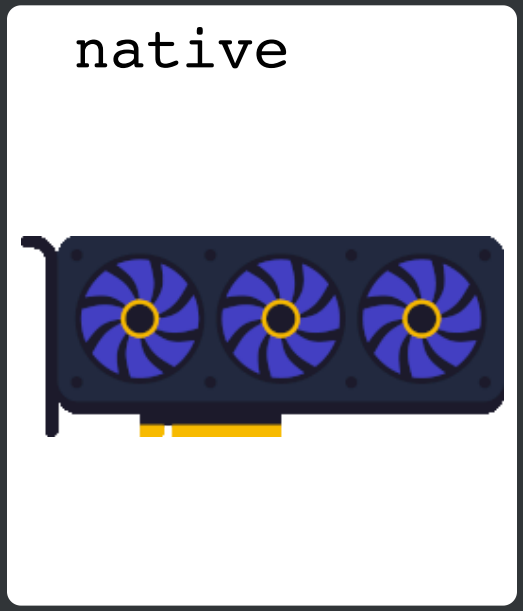
`acpi_backlight` can be configured with various values, depending on the design of the laptop

https://docs.kernel.org/admin-guide/kernel-parameters.html

The `acpi_backlight` kernel parameter

Easily one of the most configured by laptop users

`acpi_backlight` can be configured with various values, depending on the design of the laptop



https://docs.kernel.org/admin-guide/kernel-parameters.html

The `acpi_backlight` kernel parameter

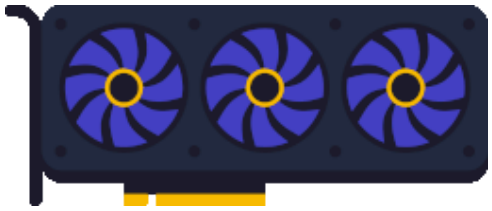
Easily one of the most configured by laptop users

`acpi_backlight` can be configured with various values, depending on the design of the laptop

`video`

ACPI `video.ko`

`native`



https://docs.kernel.org/admin-guide/kernel-parameters.html

The `acpi_backlight` kernel parameter

Easily one of the most configured by laptop users

`acpi_backlight` can be configured with various values, depending on the design of the laptop

`video`

ACPI `video.ko`

`vendor`

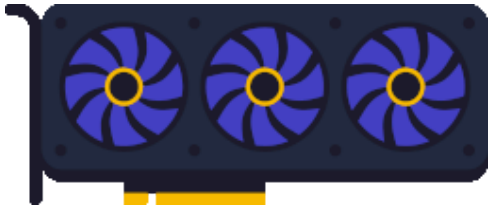
ThinkPad SONY ASUS

`thinkpad_acpi.ko`

`sony_acpi.ko`

`asus_laptop.ko`

`native`



https://docs.kernel.org/admin-guide/kernel-parameters.html

The `acpi_backlight` kernel parameter

Easily one of the most configured by laptop users

`acpi_backlight` can be configured with various values, depending on the design of the laptop

`video`

ACPI `video.ko`

`vendor`

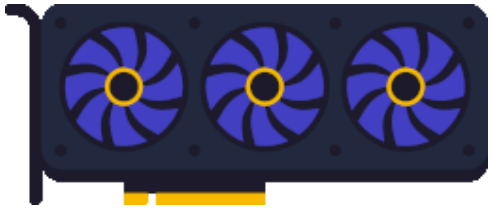
ThinkPad SONY ASUS

`thinkpad_acpi.ko`

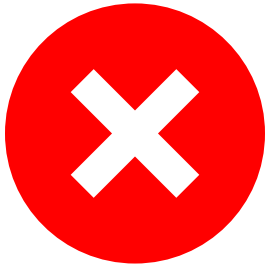
`sony_acpi.ko`

`asus_laptop.ko`

`native`



`none`



https://docs.kernel.org/admin-guide/kernel-parameters.html

Let's start with internal panels
Wrinkles with VGA Switcheroo

`vga_switcheroo` handles laptop hybrid graphics

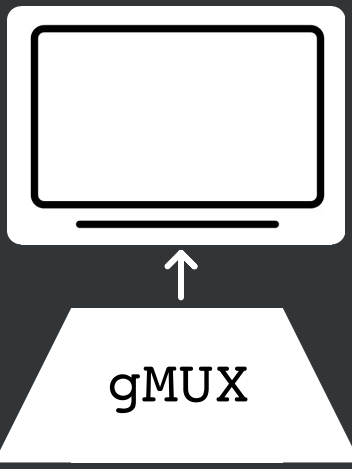
From the kernel docs:

These come in two flavors:

- `muxed`: Dual GPUs with a multiplexer chip to switch outputs between GPUs.
- `muxless`: Dual GPUs but only one of them is connected to outputs. The other one is merely used to offload rendering, its results are copied over PCIe into the framebuffer. On Linux this is supported with DRI PRIME.

Let's start with internal panels
Wrinkles with VGA Switcheroo

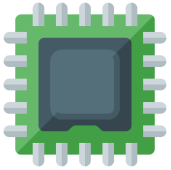
For the discussion of backlight, we only care about the `muxed` case



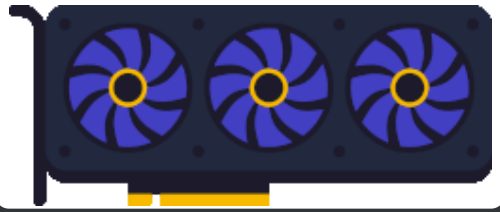

```
bool (struct backlight_ops).
(*controls_device)
```

true

laptop iGPU



laptop dGPU

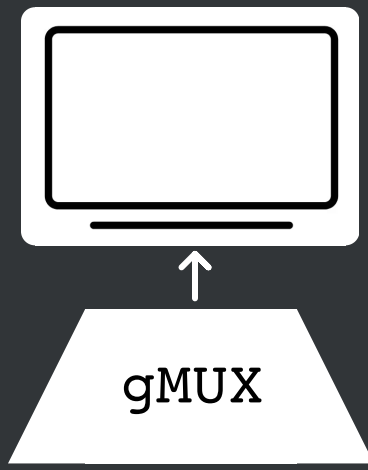


```
bool (struct backlight_ops).
(*controls_device)
```

false

Let's start with internal panels
Wrinkles with VGA Switcheroo

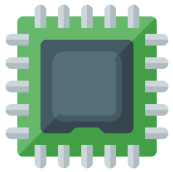
For the discussion of backlight, we only care about the `muxed` case




```
bool (struct backlight_ops).
(*controls_device)
```

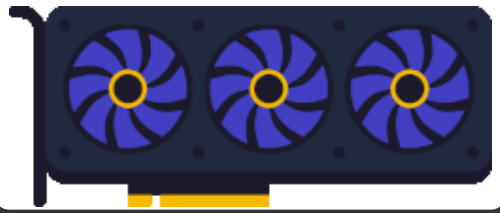
false

laptop iGPU



```
bool (struct backlight_ops).
(*controls_device)
```

laptop dGPU



true

So what makes internal internal panels so different from external panels?

Modern laptop panels use **Embedded DisplayPort** (**eDP**), replacing **low-voltage differential signaling** (**LVDS**)

The goal of **eDP** is to re-use the **DisplayPort** protocol for internal panels

eDP is not so magical

On **DisplayPort**, we can do the following with **i2c** over the **DP Aux channel**

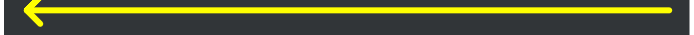
i2c address

Functionality

eDP is not so magical

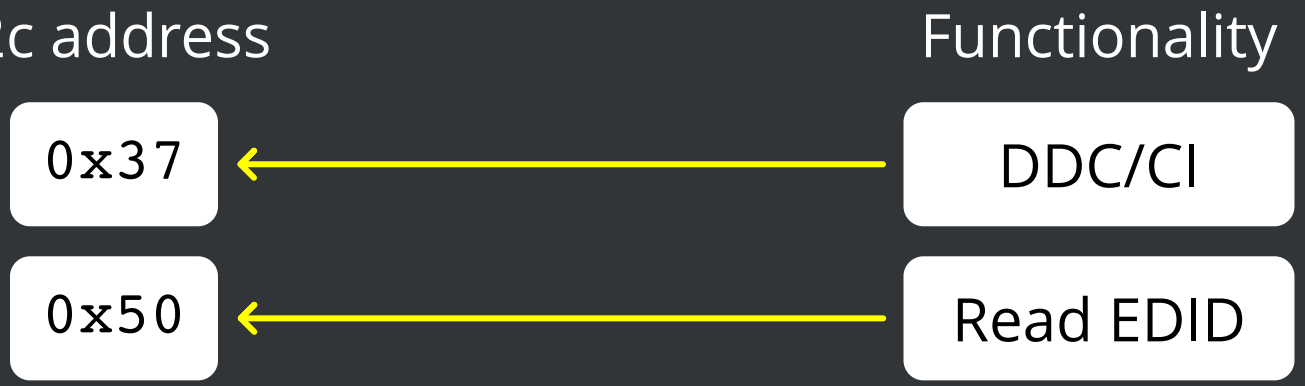
On **DisplayPort**, we can do the following with **i2c** over the **DP Aux channel**

i2c address	Functionality
`0x50`	Read EDID



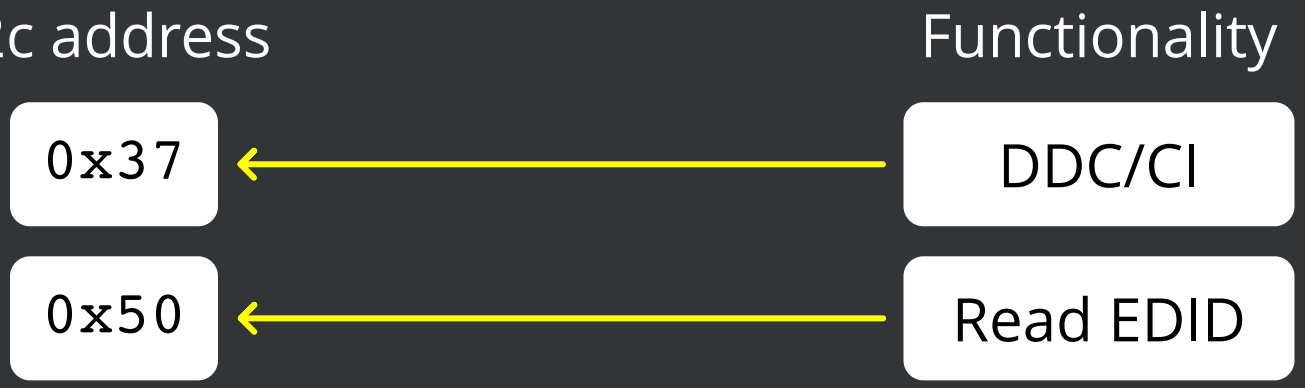
eDP is not so magical

On **DisplayPort**, we can do the following with **i2c** over the **DP Aux channel**



eDP is not so magical

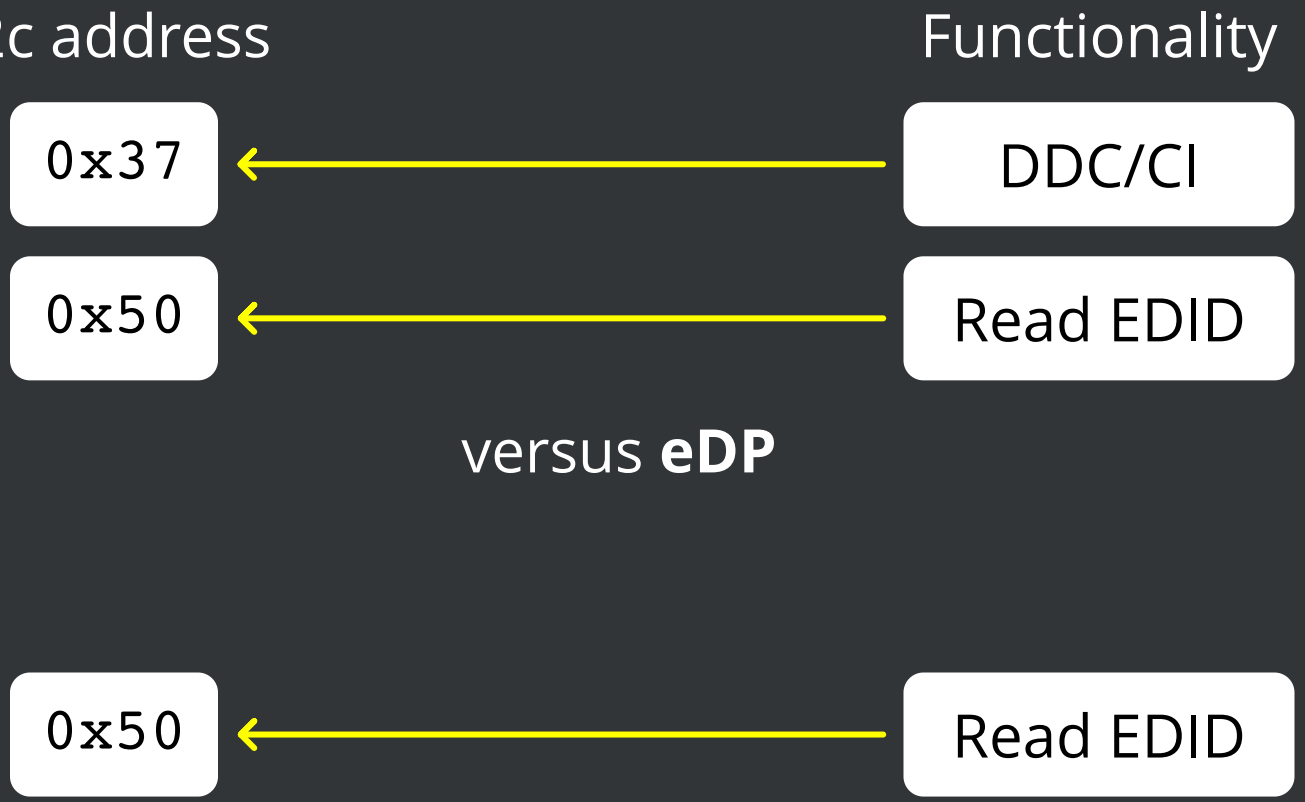
On **DisplayPort**, we can do the following with **i2c** over the **DP Aux channel**



versus **eDP**

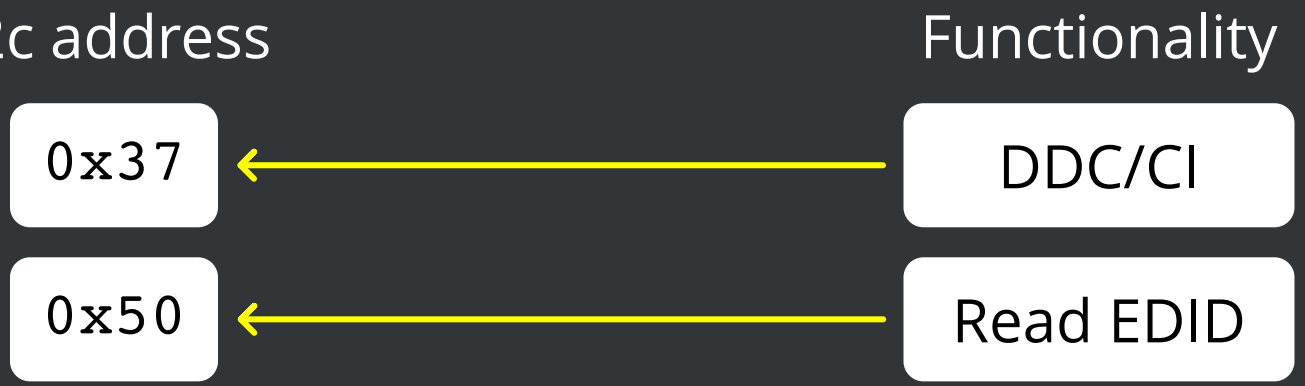
eDP is not so magical

On **DisplayPort**, we can do the following with **i2c** over the **DP Aux channel**

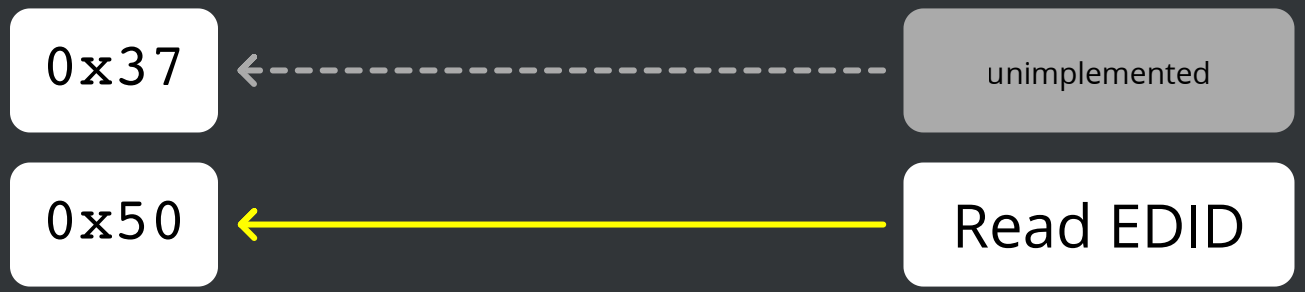


eDP is not so magical

On **DisplayPort**, we can do the following with **i2c** over the **DP Aux channel**



versus **eDP**



eDP does not implement **DDC/CI**, like **LVDS**, meaning the panel needs custom programming to control

State of Linux for controlling external panels

Only **DDC/CI** is supported in the kernel API in a very raw manner

```
/**
 * ...
 *
 * Ensures that the ddc field of the connector is correctly set.
 *
 * ...
 */
int drm_connector_init_with_ddc(struct drm_device *dev,
                                struct drm_connector *connector,
                                const struct drm_connector_funcs *funcs,
                                int connector_type,
                                struct i2c_adapter *ddc);
int drmm_connector_init(struct drm_device *dev,
                        struct drm_connector *connector,
                        const struct drm_connector_funcs *funcs,
                        int connector_type,
                        struct i2c_adapter *ddc);
```

These two helper functions just call into other DRM connector initialization helper functions.

State of Linux for controlling external panels

Only **DDC/CI** is supported in the kernel API in a very raw manner

Just gets assigned to `connector->ddc` for bookkeeping purposes

```c
/**
 * ...
 *
 * Ensures that the ddc field of the connector is correctly set.
 *
 * ...
 */
int drm_connector_init_with_ddc(struct drm_device *dev,
                                struct drm_connector *connector,
                                const struct drm_connector_funcs *funcs,
                                int connector_type,
                                struct i2c_adapter *ddc);
int drmm_connector_init(struct drm_device *dev,
                        struct drm_connector *connector,
                        const struct drm_connector_funcs *funcs,
                        int connector_type,
                        struct i2c_adapter *ddc);
```

These two helper functions just call into other DRM connector initialization helper functions.

State of Linux for controlling external panels

Let's look at how `amdgpu` handles setting up DDC/CI

```
/*
 * detect_link_and_local_sink() - Detect if a sink is attached to a given link
 *
 * link->local_sink is created or destroyed as needed.
 *
 * This does not create remote sinks.
 */
static bool detect_link_and_local_sink(struct dc_link *link,
                    enum dc_detect_reason reason)
{
    /* code omitted */

        /* From Disconnected-to-Connected. */
        switch (link->connector_signal) {
        case SIGNAL_TYPE_HDMI_TYPE_A: {
            sink_caps.transaction_type = DDC_TRANSACTION_TYPE_I2C;
            if (aud_support->hdmi_audio_native)
                sink_caps.signal = SIGNAL_TYPE_HDMI_TYPE_A;
            else
                sink_caps.signal = SIGNAL_TYPE_DVI_SINGLE_LINK;
            break;
        }
```

State of Linux for controlling external panels

Let's look at how `amdgpu` handles setting up DDC/CI

```
 4  * link->local_sink is created or destroyed as needed.
 5  *
 6  * This does not create remote sinks.
 7  */
 8 static bool detect_link_and_local_sink(struct dc_link *link,
 9                     enum dc_detect_reason reason)
10 {
11     /* code omitted */
12
13         /* From Disconnected-to-Connected. */
14         switch (link->connector_signal) {
15         case SIGNAL_TYPE_HDMI_TYPE_A: {
16             sink_caps.transaction_type =
   DDC_TRANSACTION_TYPE_I2C;
17             if (aud_support->hdmi_audio_native)
18                 sink_caps.signal = SIGNAL_TYPE_HDMI_TYPE_A;
19             else
20                 sink_caps.signal =
   SIGNAL_TYPE_DVI_SINGLE_LINK;
21             break;
22         }
23
24         case SIGNAL_TYPE_DVI_SINGLE_LINK: {
25             sink_caps.transaction_type =
   DDC_TRANSACTION_TYPE_I2C;
26             sink_caps.signal = SIGNAL_TYPE_DVI_SINGLE_LINK;
```

For a number of signal types like HDMI, `amdgpu` identifies DDC transactions to be type `i2c`.

State of Linux for controlling external panels

Let's look at how amdgpu handles setting up DDC/CI

```
62          }
63
64          case SIGNAL_TYPE_DISPLAY_PORT: {
65
66              /* wa HPD high coming too early*/
67              if (link->ep_type == DISPLAY_ENDPOINT_PHY &&
68                  link->link_enc-
    >features.flags.bits.DP_IS_USB_C == 1) {
69
70                  /* if alt mode times out, return false */
71                  if (!wait_for_entering_dp_alt_mode(link))
72                      return false;
73              }
74
75              if (!detect_dp(link, &sink_caps, reason)) {
76
77                  if (prev_sink)
78                      dc_sink_release(prev_sink);
79                  return false;
80              }
81
82              /* Active SST downstream branch device unplug*/
83              if (link->type == dc_connection_sst_branch &&
84                  link->dpcd_caps.sink_count.bits.SINK_COUNT ==
    0) {
85                  if (prev_sink)
```

DisplayPort gets complicated. We will discuss this soon.

State of Linux for controlling external panels

Let's look at how `amdgpu` handles setting up DDC/CI

```
>dpcd_caps.usb4_dp_tun_info.dp_tun_cap.bits.dpia_bw_alloc
105                     && link-
>dpcd_caps.usb4_dp_tun_info.driver_bw_cap.bits.driver_bw_allo
c_support) {
106                 if
(link_dpia_enable_usb4_dp_bw_alloc_mode(link) == false)
107                     link-
>dpcd_caps.usb4_dp_tun_info.dp_tun_cap.bits.dpia_bw_alloc =
false;
108             }
109             break;
110         }
111
112         default:
113             DC_ERROR("Invalid connector type! signal:%d\n",
114                 link->connector_signal);
115             if (prev_sink)

116                 dc_sink_release(prev_sink);
117             return false;
118         } /* switch() */
119
120         /* code omitted */
121
122         set_ddc_transaction_type(link->ddc,
123                                  sink_caps.transaction_type);
```

Finally, `amdgpu` sets the transaction type for the link itself.

State of Linux for controlling external panels

Let's explore the DisplayPort case for amdgpu

```
1  static bool detect_dp(struct dc_link *link,
2                struct display_sink_capability *sink_caps,
3                enum dc_detect_reason reason)
4  {
5      struct audio_support *audio_support = &link->dc->res_pool-
   >audio_support;
6
7      sink_caps->signal = link_detect_sink_signal_type(link,
   reason);
8      sink_caps->transaction_type =
9          get_ddc_transaction_type(sink_caps->signal);
10
11     if (sink_caps->transaction_type ==
   DDC_TRANSACTION_TYPE_I2C_OVER_AUX) {
12         sink_caps->signal = SIGNAL_TYPE_DISPLAY_PORT;
13         if (!detect_dp_sink_caps(link)) {
14             return false;
15         }
16
17         if (is_dp_branch_device(link))
18             /* DP SST branch */
19             link->type = dc_connection_sst_branch;
20     } else {
21         if (link->dc->debug.disable_dp_plus_plus_wa &&
22                 link->link_enc-
```

State of Linux for controlling external panels

Let's explore the DisplayPort case for amdgpu

```
static bool detect_dp(struct dc_link *link,
              struct display_sink_capability *sink_caps,
              enum dc_detect_reason reason)
{
    struct audio_support *audio_support = &link->dc->res_pool->audio_support;

    sink_caps->signal = link_detect_sink_signal_type(link, reason);
    sink_caps->transaction_type =
        get_ddc_transaction_type(sink_caps->signal);

    if (sink_caps->transaction_type == DDC_TRANSACTION_TYPE_I2C_OVER_AUX) {
        sink_caps->signal = SIGNAL_TYPE_DISPLAY_PORT;
        if (!detect_dp_sink_caps(link)) {
            return false;
        }

        if (is_dp_branch_device(link))
            /* DP SST branch */
            link->type = dc_connection_sst_branch;
    } else {
        if (link->dc->debug.disable_dp_plus_plus_wa &&
                link->link_enc-
```

amdgpu determines if the **i2c** transactions are native or goes over **DP Aux** based on if the signal is **DisplayPort** or is being passively converted due to features like **DP++**.

State of Linux for controlling external panels

Let's explore the DisplayPort case for amdgpu

```
5     struct audio_support *audio_support = &link->dc->res_pool->audio_support;
6
7     sink_caps->signal = link_detect_sink_signal_type(link, reason);
8     sink_caps->transaction_type =
9         get_ddc_transaction_type(sink_caps->signal);
10
11    if (sink_caps->transaction_type == DDC_TRANSACTION_TYPE_I2C_OVER_AUX) {
12        sink_caps->signal = SIGNAL_TYPE_DISPLAY_PORT;
13        if (!detect_dp_sink_caps(link)) {
14            return false;
15        }
16
17        if (is_dp_branch_device(link))
18            /* DP SST branch */
19            link->type = dc_connection_sst_branch;
20    } else {
21        if (link->dc->debug.disable_dp_plus_plus_wa &&
22                link->link_enc->features.flags.bits.IS_UHBR20_CAPABLE)
23            return false;
24
25        /* DP passive dongles */
26        sink_caps->signal = dp_passive_dongle_detection(link-
```

The logic when the signal is pure **DisplayPort**.

State of Linux for controlling external panels

Let's explore the DisplayPort case for amdgpu

```
15          }
16
17          if (is_dp_branch_device(link))
18              /* DP SST branch */
19              link->type = dc_connection_sst_branch;
20      } else {
21          if (link->dc->debug.disable_dp_plus_plus_wa &&
22                  link->link_enc-
>features.flags.bits.IS_UHBR20_CAPABLE)
23              return false;
24
25          /* DP passive dongles */
26          sink_caps->signal = dp_passive_dongle_detection(link-
>ddc,
27                                  sink_caps,
28                                  audio_support);
29          link->dpcd_caps.dongle_type = sink_caps->dongle_type;
30          link->dpcd_caps.is_dongle_type_one = sink_caps-
>is_dongle_type_one;
31          link->dpcd_caps.dpcd_rev.raw = 0;
32          link->dpcd_caps.usb4_dp_tun_info.dp_tun_cap.raw = 0;
33      }
34
35      return true;
36  }
```

DisplayPort is really complex due to **DP++** and **USB4 DP tunneling**.

State of Linux for controlling external panels

amdgpu creates an i2c_adapter interface for DDC/CI

```
1  static struct amdgpu_i2c_adapter *
2  create_i2c(struct ddc_service *ddc_service, bool oem)
3  {
4      struct amdgpu_device *adev = ddc_service->ctx-
   >driver_context;
5      struct amdgpu_i2c_adapter *i2c;
6
7      i2c = kzalloc(sizeof(struct amdgpu_i2c_adapter),
   GFP_KERNEL);
8      if (!i2c)
9          return NULL;
10     i2c->base.owner = THIS_MODULE;
11     i2c->base.dev.parent = &adev->pdev->dev;
12     i2c->base.algo = &amdgpu_dm_i2c_algo;
13     if (oem)
14         snprintf(i2c->base.name, sizeof(i2c->base.name),
   "AMDGPU DM i2c OEM bus");
15     else
16         snprintf(i2c->base.name, sizeof(i2c->base.name),
   "AMDGPU DM i2c hw bus %d",
17                 ddc_service->link->link_index);
18     i2c_set_adapdata(&i2c->base, i2c);
19     i2c->ddc_service = ddc_service;
20     i2c->oem = oem;
21
22     return i2c;
23 }
```

amdgpu creates an i2c_adapter for DDC/CI that will be either a pure i2c interface or i2c over DP Aux.

State of Linux for controlling external panels

amdgpu plumbs the i2c_adapter to the drm_connector

```
1  /*
2   * Note: this function assumes that dc_link_detect() was called
   for the
3   * dc_link which will be represented by this aconnector.
4   */
5  static int amdgpu_dm_connector_init(struct
   amdgpu_display_manager *dm,
6                      struct amdgpu_dm_connector *aconnector,
7                      u32 link_index,
8                      struct amdgpu_encoder *aencoder)
9  {
10     int res = 0;
11     int connector_type;
12     struct dc *dc = dm->dc;
13     struct dc_link *link = dc_get_link_at_index(dc,
   link_index);
14     struct amdgpu_i2c_adapter *i2c;
15
16     /* Not needed for writeback connector */
17     link->priv = aconnector;
18
19
20     i2c = create_i2c(link->ddc, false);
21     if (!i2c) {
22         drm_err(adev_to_drm(dm->adev), "Failed to create i2c
   adapter data\n");
23         return -ENOMEM;
```

State of Linux for controlling external panels

amdgpu plumbs the i2c_adapter to the drm_connector

```
19
20      i2c = create_i2c(link->ddc, false);
21      if (!i2c) {
22          drm_err(adev_to_drm(dm->adev), "Failed to create i2c
    adapter data\n");
23          return -ENOMEM;
24      }
25
26      aconnector->i2c = i2c;
27      res = i2c_add_adapter(&i2c->base);
28
29      if (res) {
30          drm_err(adev_to_drm(dm->adev), "Failed to register hw
    i2c %d\n", link->link_index);
31          goto out_free;
32      }
33
34      connector_type = to_drm_connector_type(link-
    >connector_signal);
35
36      res = drm_connector_init_with_ddc(
37              dm->ddev,
38              &aconnector->base,
39              &amdgpu_dm_connector_funcs,
40              connector_type,
41              &i2c->base);
42
```

The created `i2c_adapter` gets passed to the DRM connector initialization helper

State of Linux for controlling external panels

Userspace use of the `i2c_adapter`

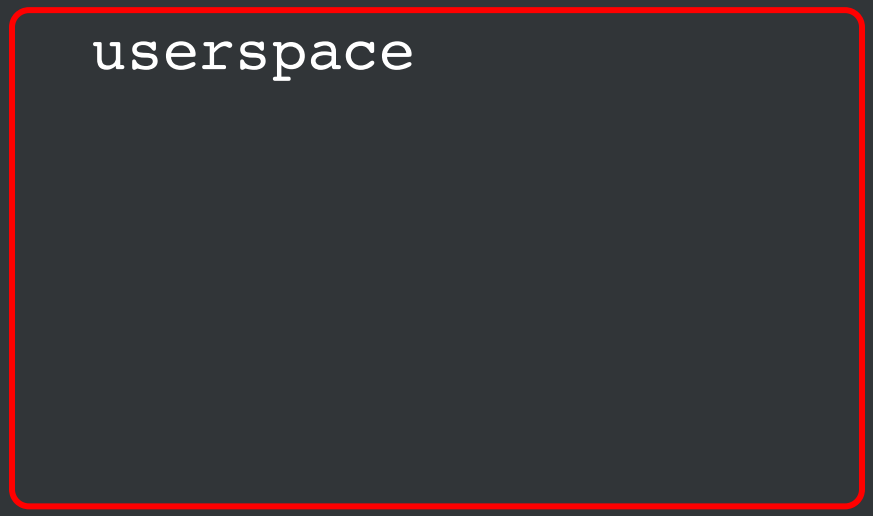
`ddcutil` reads and writes `/dev/i2c*` device nodes. Either the user can specify an explicit bus to use or `ddcutil` will crawl through the exposed nodes to see which ones are for **DDC/CI**

NOTE: `ddcutil` also support **USB Monitor Control Class**

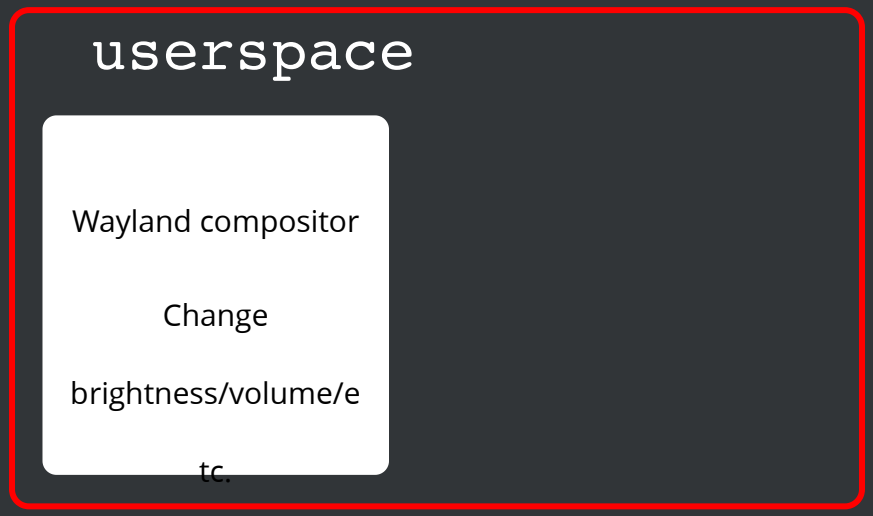
```
sudo strace -e openat ddcutil probe |& grep /dev/i2c
openat(AT_FDCWD, "/dev/i2c-5", O_RDWR)  = 3
openat(AT_FDCWD, "/dev/i2c-6", O_RDWR)  = 3
openat(AT_FDCWD, "/dev/i2c-7", O_RDWR)  = 3
openat(AT_FDCWD, "/dev/i2c-8", O_RDWR)  = 3
openat(AT_FDCWD, "/dev/i2c-9", O_RDWR)  = 3
openat(AT_FDCWD, "/dev/i2c-10", O_RDWR) = 3
openat(AT_FDCWD, "/dev/i2c-11", O_RDWR) = 3
openat(AT_FDCWD, "/dev/i2c-12", O_RDWR) = 3
openat(AT_FDCWD, "/dev/i2c-13", O_RDWR) = 3
openat(AT_FDCWD, "/dev/i2c-14", O_RDWR) = 3
openat(AT_FDCWD, "/dev/i2c-15", O_RDWR) = 3
openat(AT_FDCWD, "/dev/i2c-16", O_RDWR) = 3
openat(AT_FDCWD, "/dev/i2c-17", O_RDWR) = 3
openat(AT_FDCWD, "/dev/i2c-14", O_RDWR) = 3
```

How do we improve the situation with regards to monitor control on Linux?

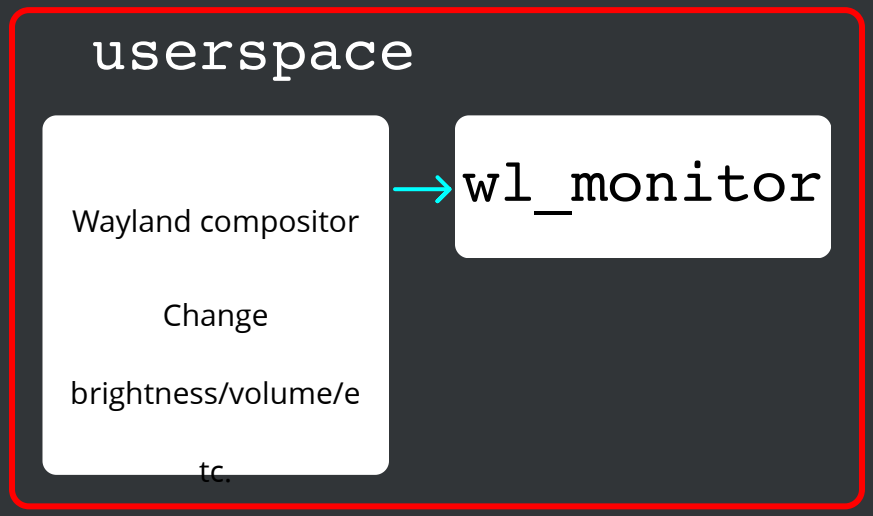
A high-level overview



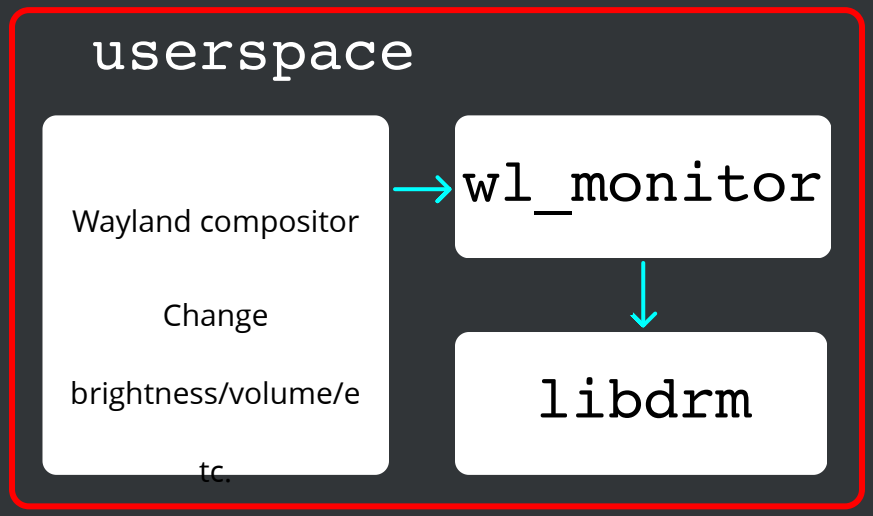
A high-level overview



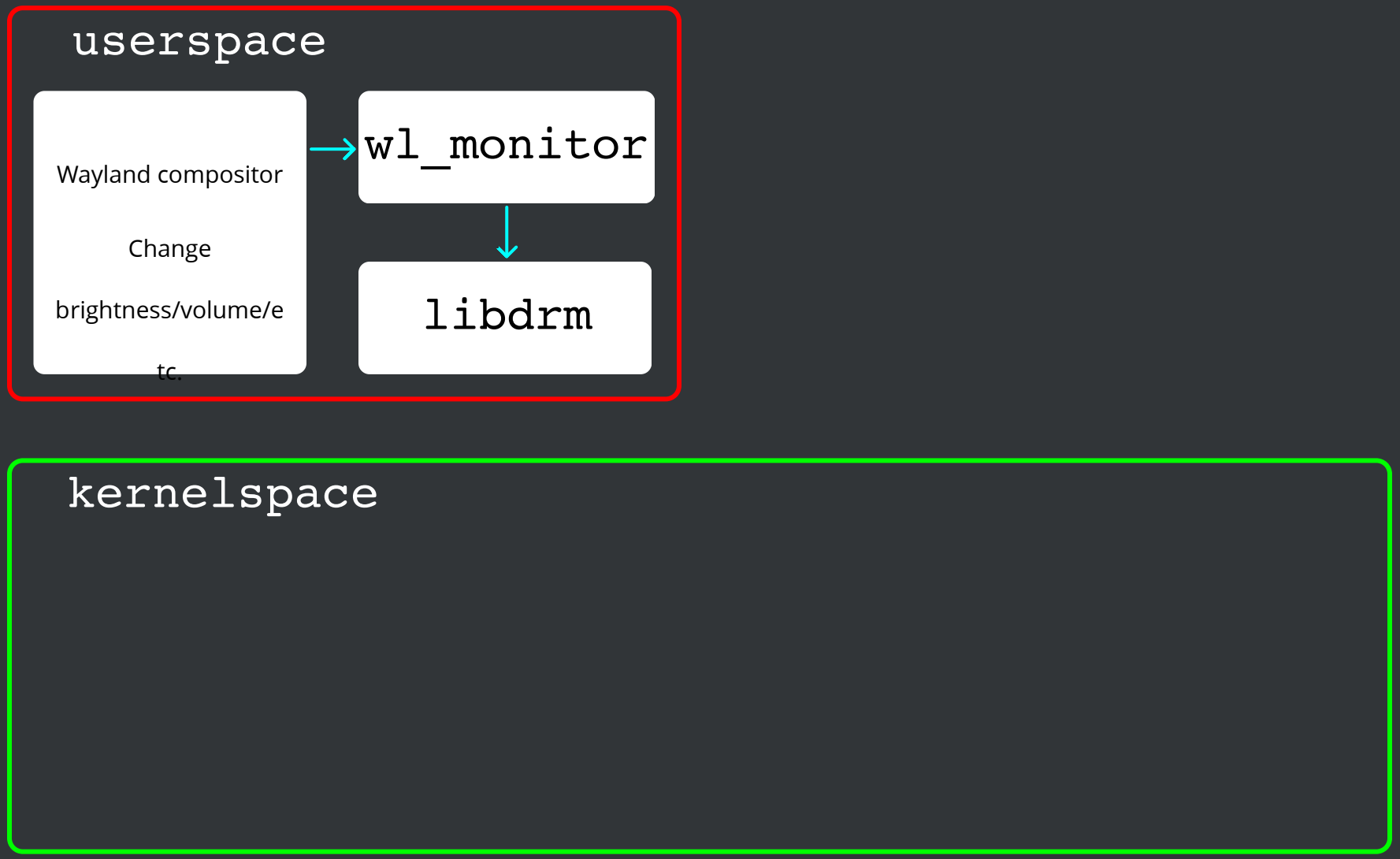
A high-level overview



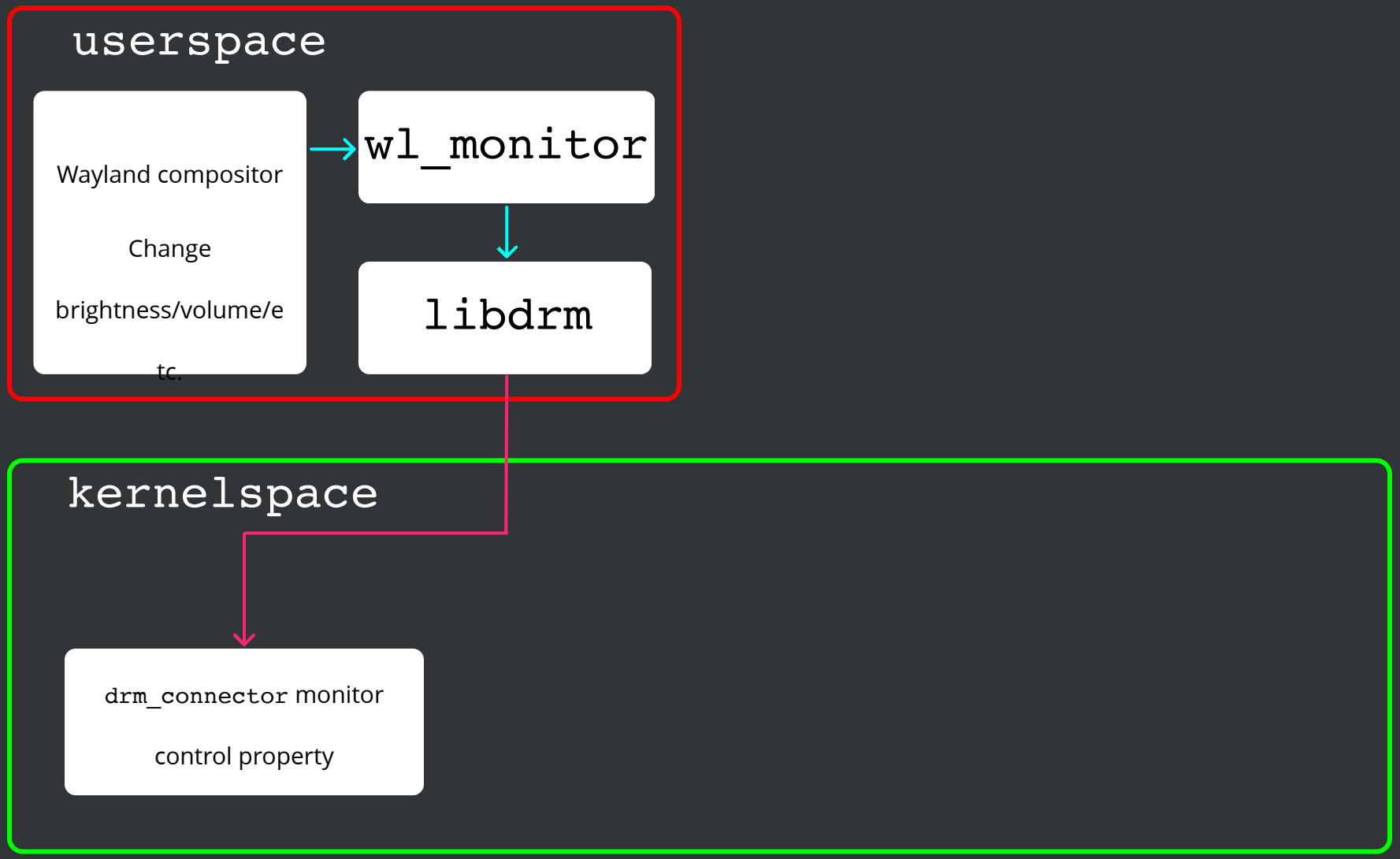
A high-level overview



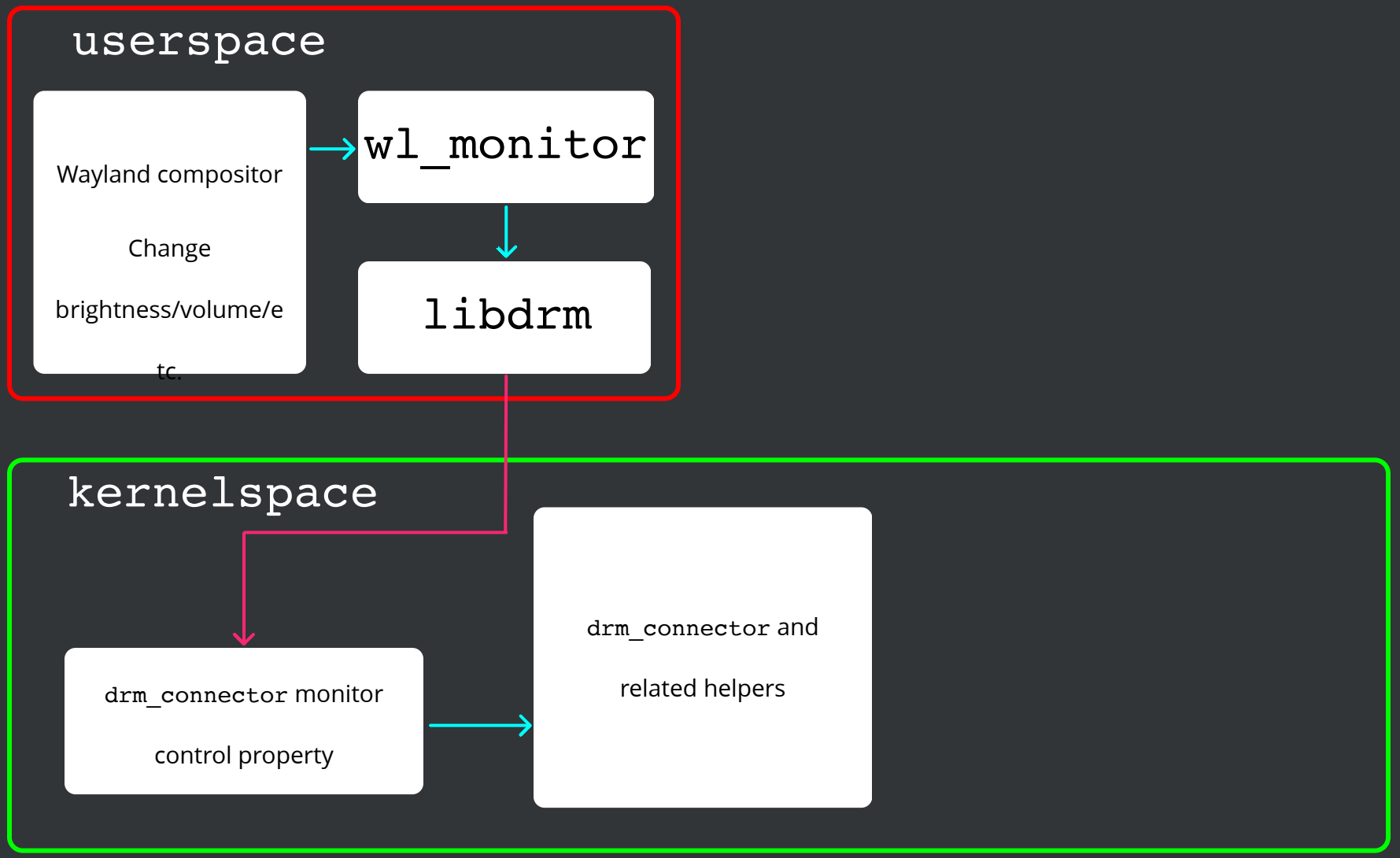
A high-level overview



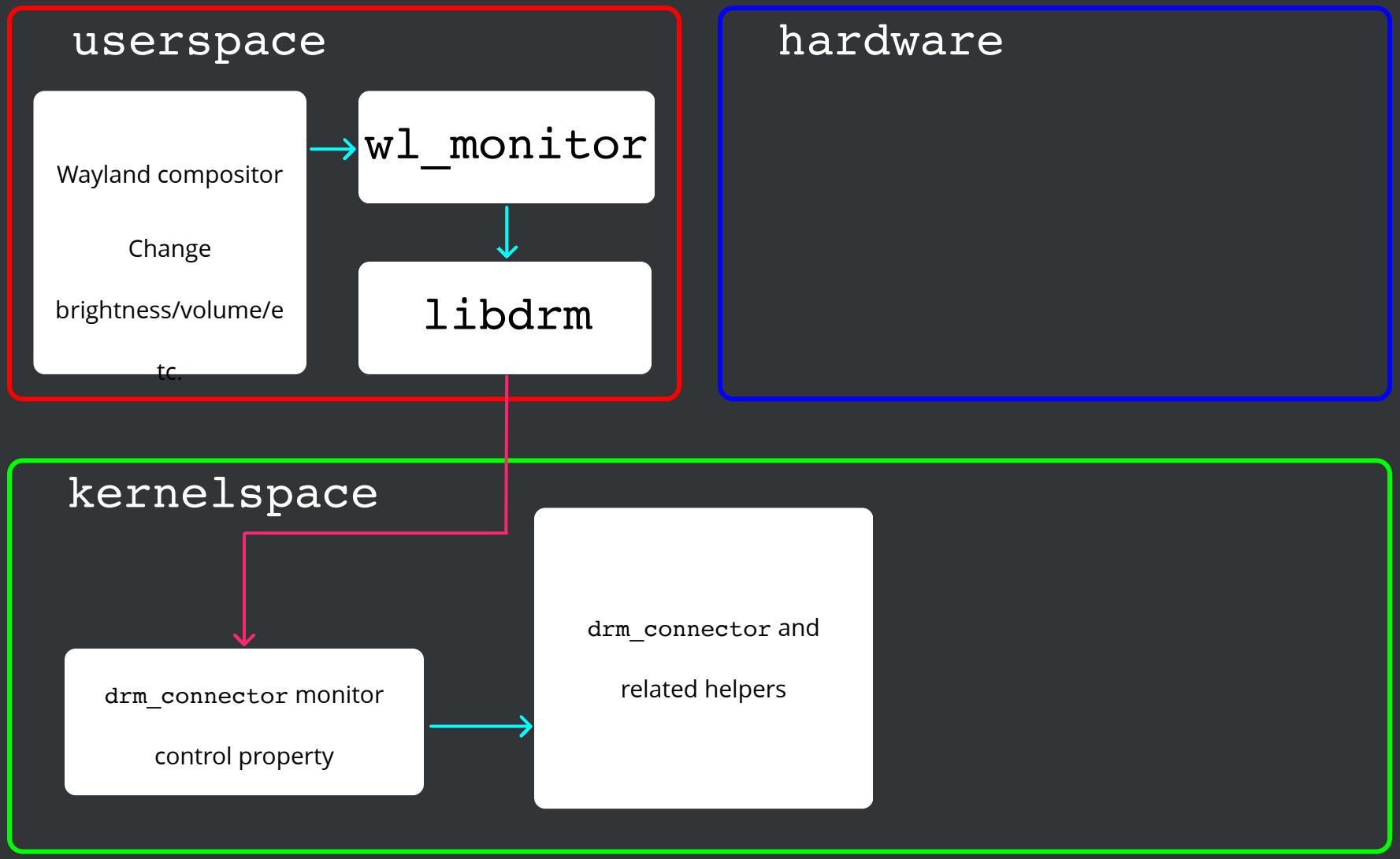
A high-level overview



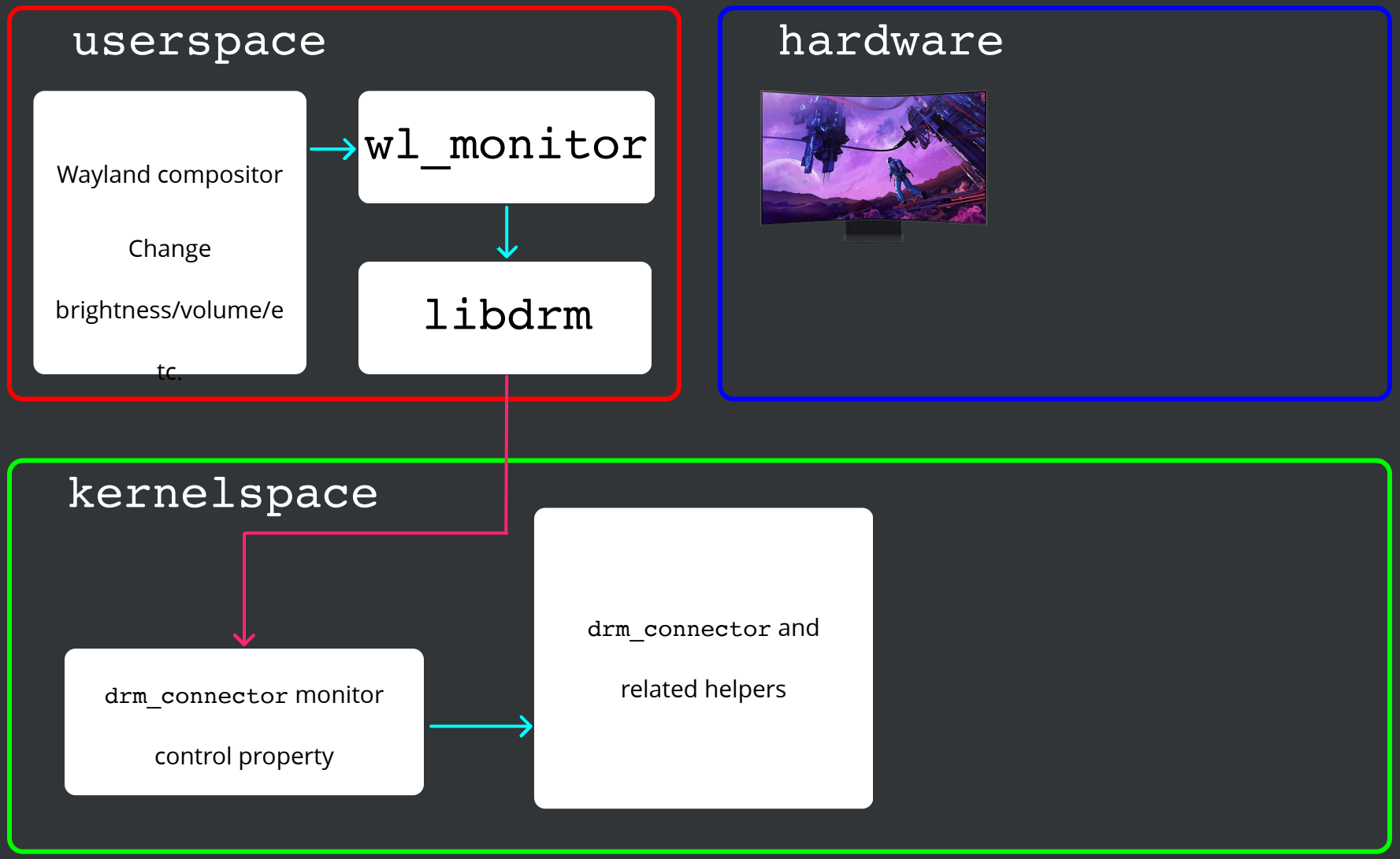
A high-level overview



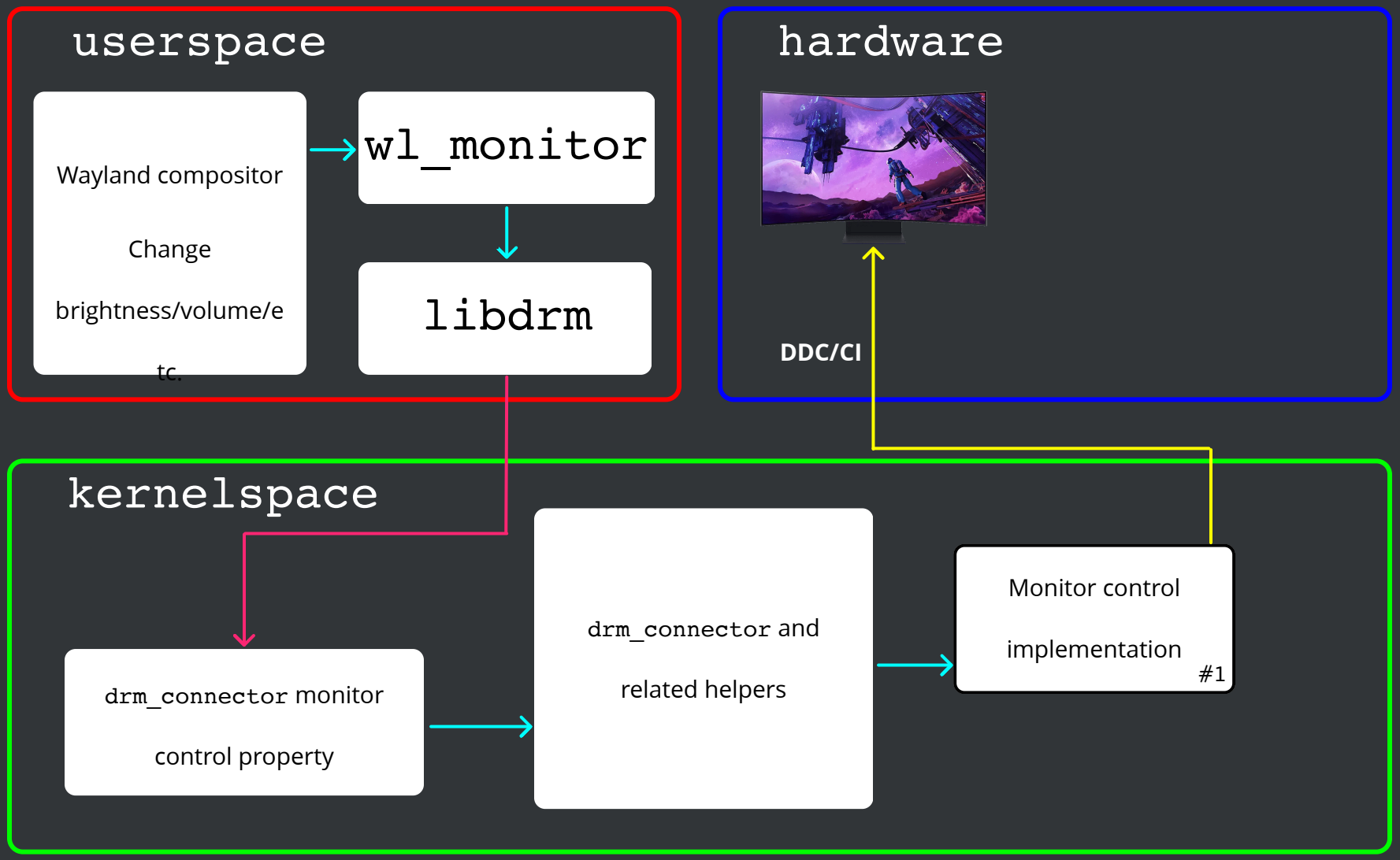
A high-level overview



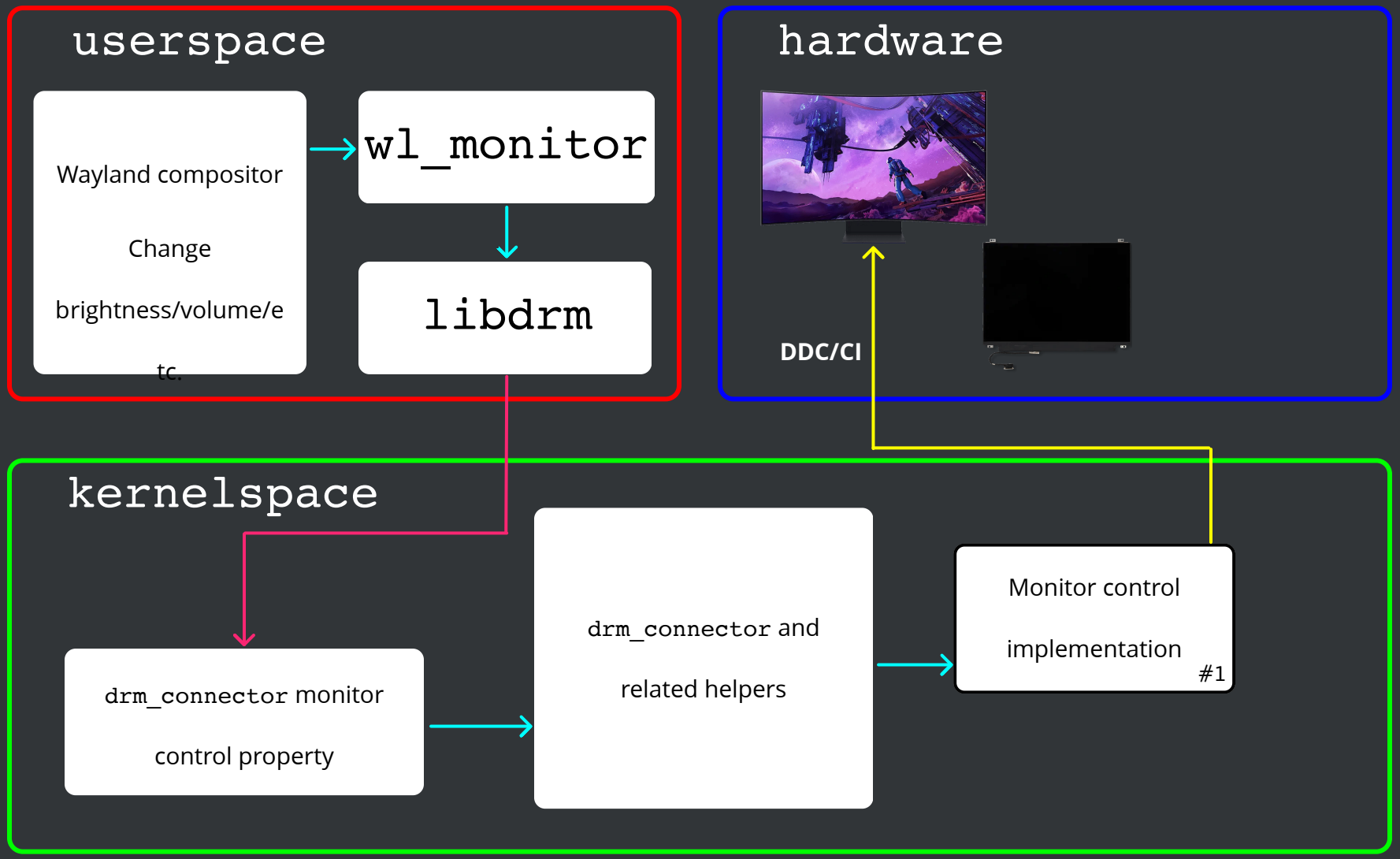
A high-level overview



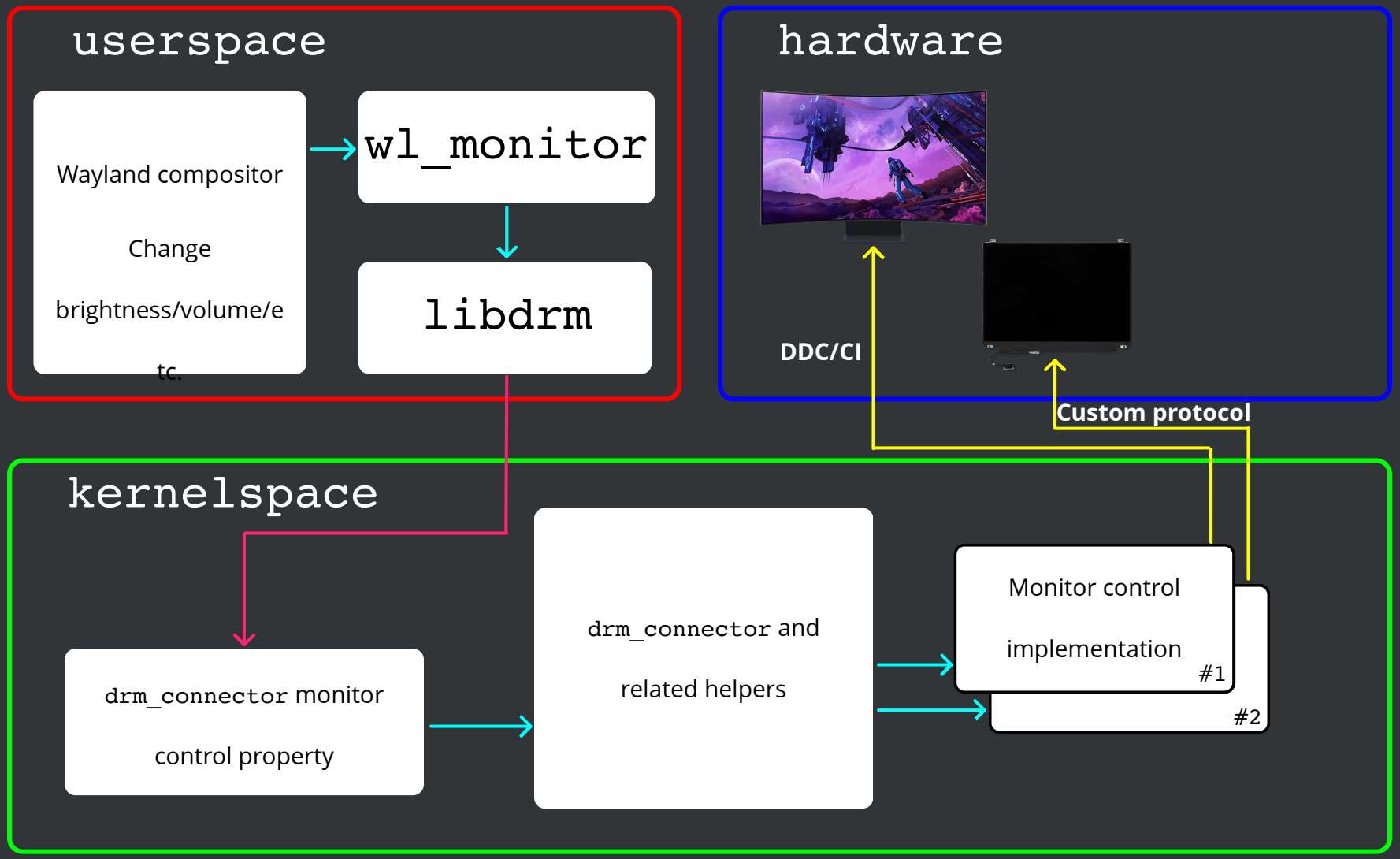
A high-level overview



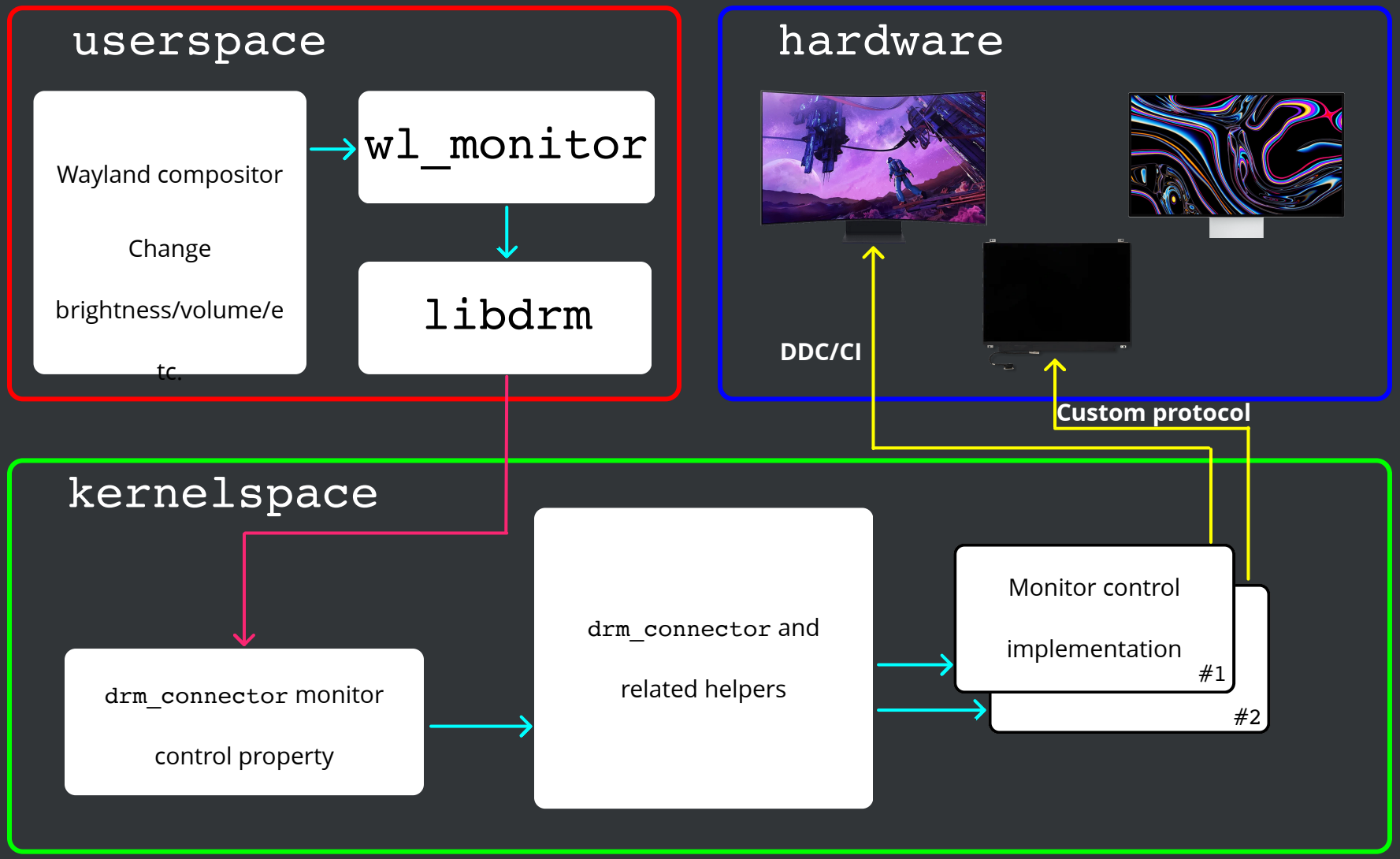
A high-level overview



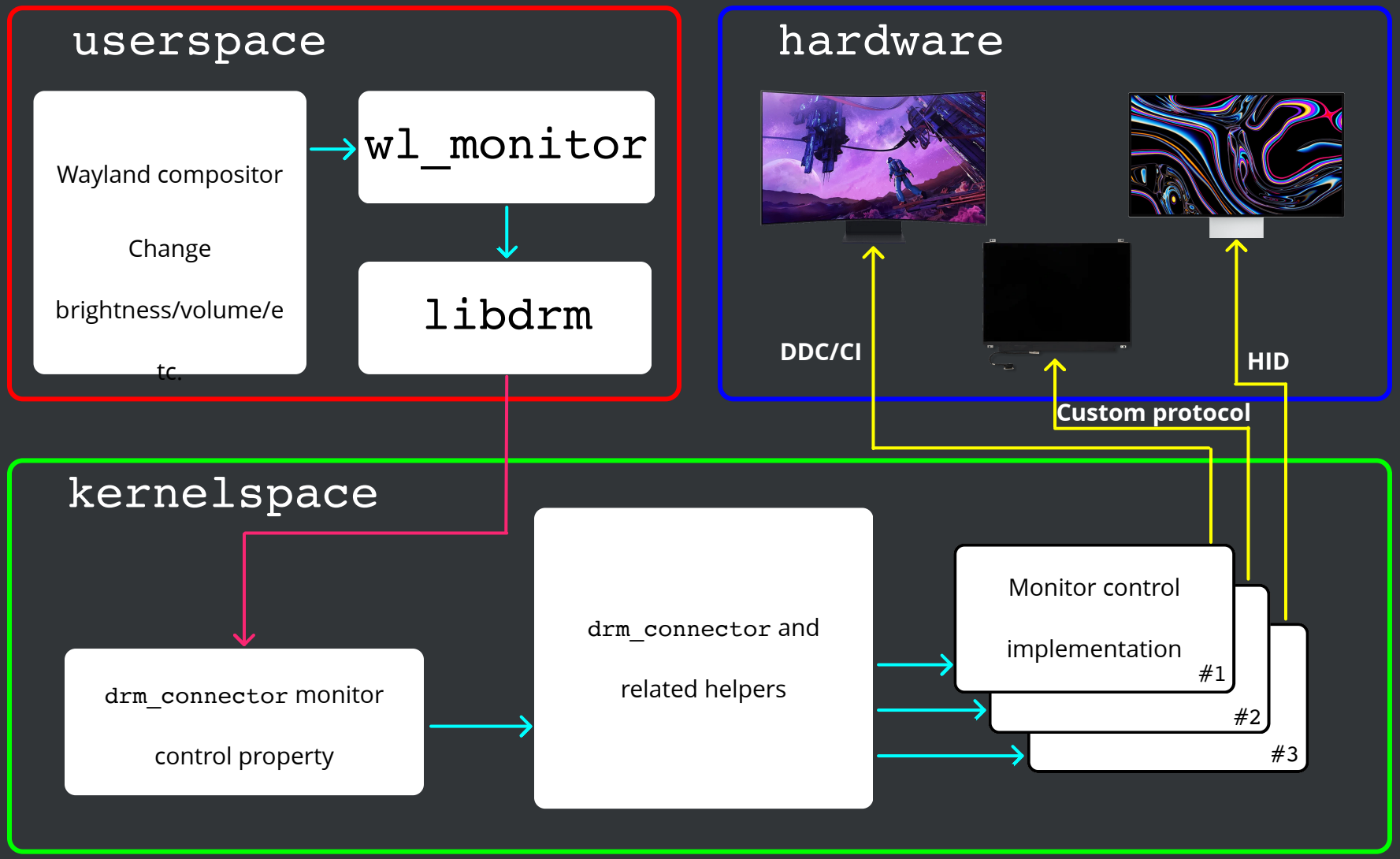
A high-level overview



A high-level overview



A high-level overview



Minor caveats

- For certain panels, the ability to have some kind of policy mechanism for control implementations will be necessary.
- In particular, laptop panels can have a variety of implementations for brightness control, and monitors can support both **USB Monitor Control Class** and **DDC/CI**.
- Scoping the API at the `drm_connector` level works well with mux-able hybrid graphics. One caveat is when the **gMUX** or an external controller controls the brightness. Being able to attach implementations to a `drm_connector` solves this issue.

Ok, what does any of this have to do with Rust for Linux?

My ambitions

The work I have posted so far to the rust-for-linux mailing list is intended as building blocks to implement the architecture I discussed in Rust.

- I plan to implement the **USB Monitor Control Class** support for USB monitors in Rust
- I have posted **HID** abstractions to use for this effort
  - https://lore.kernel.org/rust-for-linux/20250808061223.3770-1-sergeantsagara@protonmail.com/
- I have proposed a means for extending functionality of `drm_connector` in Rust
  - https://lore.kernel.org/rust-for-linux/20250818050251.102399-2-sergeantsagara@protonmail.com/
- In the next couple of slides, I will discuss these efforts

HID Abstractions in Rust

To begin with, I have created a simple Rust reference driver, `GloriousRust`, for making the Glorious Model O mouse function correctly.

Without this type of "fixup" driver, the inputs of the mouse will be ignored by the core HID handling in the kernel.

Let's delve into the Rust driver code.

HID Abstractions in Rust

Analyzing the `GloriousRust` driver

```rust
// SPDX-License-Identifier: GPL-2.0

// Copyright (C) 2025 Rahul Rameshbabu <sergeantsagara@protonmail.com>

//! Rust reference HID driver for Glorious Model O and O- mice.

use kernel::{self, bindings, device, hid, prelude::*};

struct GloriousRust;

kernel::hid_device_table!(
    HID_TABLE,
    MODULE_HID_TABLE,
    <GloriousRust as hid::Driver>::IdInfo,
    [(
        hid::DeviceId::new_usb(
            hid::Group::Generic,
            bindings::USB_VENDOR_ID_SINOWEALTH,
            bindings::USB_DEVICE_ID_GLORIOUS_MODEL_O,
        ),
        (),
    )]
);

#[vtable]
```

HID Abstractions in Rust

Analyzing the `GloriousRust` driver

1. Declare the device table for matching devices to the driver

```
5  //! Rust reference HID driver for Glorious Model O and O- mice.
6
7  use kernel::{self, bindings, device, hid, prelude::*};
8
9  struct GloriousRust;
10
11 kernel::hid_device_table!(
12     HID_TABLE,
13     MODULE_HID_TABLE,
14     <GloriousRust as hid::Driver>::IdInfo,
15     [(
16         hid::DeviceId::new_usb(
17             hid::Group::Generic,
18             bindings::USB_VENDOR_ID_SINOWEALTH,
19             bindings::USB_DEVICE_ID_GLORIOUS_MODEL_O,
20         ),
21         (),
22     )]
23 );
24
25 #[vtable]
26 impl hid::Driver for GloriousRust {
27     type IdInfo = ();
28     const ID_TABLE: hid::IdTable<Self::IdInfo> = &HID_TABLE;
29
```

HID Abstractions in Rust

Analyzing the `GloriousRust` driver

2. Use the macro that sets up the kernel module

```
36              && (rdesc[84] == 129 && rdesc[85] == 3)
37              && (rdesc[112] == 129 && rdesc[113] == 3)
38              && (rdesc[140] == 129 && rdesc[141] == 3)
39          {
40              dev_info!(
41                  hdev.as_ref(),
42                  "patching Glorious Model O consumer control report descriptor\n"
43              );
44
45              rdesc[85] = hid::MAIN_ITEM_VARIABLE | hid::MAIN_ITEM_RELATIVE;
46              rdesc[113] = hid::MAIN_ITEM_VARIABLE | hid::MAIN_ITEM_RELATIVE;
47              rdesc[141] = hid::MAIN_ITEM_VARIABLE | hid::MAIN_ITEM_RELATIVE;
48          }
49
50          rdesc
51      }
52  }
53
54  kernel::module_hid_driver! {
55      type: GloriousRust,
56      name: "GloriousRust",
57      authors: ["Rahul Rameshbabu <sergeantsagara@protonmail.com>"],
58      description: "Rust reference HID driver for Glorious Model O and O- mice",
59      license: "GPL",
60  }
```

HID Abstractions in Rust

Analyzing the `GloriousRust` driver

3. Implement the `hid::Driver` trait's `report_fixup` to make the mouse usable

```
29
30      /// Fix the Glorious Model O and O- consumer input report descriptor to use
31      /// the variable and relative flag, while clearing the const flag.
32      ///
33      /// Without this fixup, inputs from the mice will be ignored.
34      fn report_fixup<'a, 'b: 'a>(hdev: &hid::Device<device::Core>, rdesc: &'b mut
    [u8]) -> &'a [u8] {
35          if rdesc.len() == 213
36              && (rdesc[84] == 129 && rdesc[85] == 3)
37              && (rdesc[112] == 129 && rdesc[113] == 3)
38              && (rdesc[140] == 129 && rdesc[141] == 3)
39          {
40              dev_info!(
41                  hdev.as_ref(),
42                  "patching Glorious Model O consumer control report descriptor\n"
43              );
44
45              rdesc[85] = hid::MAIN_ITEM_VARIABLE | hid::MAIN_ITEM_RELATIVE;
46              rdesc[113] = hid::MAIN_ITEM_VARIABLE | hid::MAIN_ITEM_RELATIVE;
47              rdesc[141] = hid::MAIN_ITEM_VARIABLE | hid::MAIN_ITEM_RELATIVE;
48          }
49
50          rdesc
51      }
```

HID Abstractions in Rust

Rust language features that would help

1. Wrap `bindings::HID_GROUP_*` as a `Group` enum with `#[repr(u16)]`

```rust
/// HID device groups are intended to help categories HID devices based on a set
/// of common quirks and logic that they will require to function correctly.
#[repr(u16)]
pub enum Group {
    Any = bindings::HID_GROUP_ANY as u16,
    Generic = bindings::HID_GROUP_GENERIC as u16,
    Multitouch = bindings::HID_GROUP_MULTITOUCH as u16,
    SensorHub = bindings::HID_GROUP_SENSOR_HUB as u16,
    MultitouchWin8 = bindings::HID_GROUP_MULTITOUCH_WIN_8 as u16,
    RMI = bindings::HID_GROUP_RMI as u16,
    Wacom = bindings::HID_GROUP_WACOM as u16,
    LogitechDJDevice = bindings::HID_GROUP_LOGITECH_DJ_DEVICE as u16,
    Steam = bindings::HID_GROUP_STEAM as u16,
    Logitech27MHzDevice = bindings::HID_GROUP_LOGITECH_27MHZ_DEVICE as u16,
    Vivaldi = bindings::HID_GROUP_VIVALDI as u16,
}

// TODO: use `const_trait_impl` once stabilized:
//
// ```
// impl const From<Group> for u16 {
//     /// [`Group`] variants are represented by [`u16`] values.
//     fn from(value: Group) -> Self {
//         value as Self
//     }
```

RFC 3762, "Make trait methods callable in const contexts"

HID Abstractions in Rust

Rust language features that would help

2. Need to convert Group to `u16` for `bindings::hid_device_id`

```
19 //
20 // ```
21 // impl const From<Group> for u16 {
22 //     /// [`Group`] variants are represented by [`u16`] values.
23 //     fn from(value: Group) -> Self {
24 //         value as Self
25 //     }
26 // }
27 // ```
28 impl Group {
29     /// Internal function used to convert [`Group`] variants into [`u16`].
30     const fn into(self) -> u16 {
31         self as u16
32     }
33 }
34
35 /// Abstraction for the HID device ID structure `struct hid_device_id`.
36 #[repr(transparent)]
37 #[derive(Clone, Copy)]
38 pub struct DeviceId(bindings::hid_device_id);
39
40 impl DeviceId {
41     /// Equivalent to C's `HID_USB_DEVICE` macro.
42     ///
```

RFC 3762, "Make trait methods callable in const contexts"

HID Abstractions in Rust

Rust language features that would help

3. `new_usb` can be used in `const` contexts

```
37 #[derive(Clone, Copy)]
38 pub struct DeviceId(bindings::hid_device_id);
39
40 impl DeviceId {
41     /// Equivalent to C's `HID_USB_DEVICE` macro.
42     ///
43     /// Create a new `hid::DeviceId` from a group, vendor ID, and device ID
44     /// number.
45     pub const fn new_usb(group: Group, vendor: u32, product: u32) -> Self {
46         Self(bindings::hid_device_id {
47             bus: 0x3, // BUS_USB
48             group: group.into(),
49             vendor,
50             product,
51             driver_data: 0,
52         })
53     }
54 }
55
56 // ... Driver implementation
57
58 kernel::hid_device_table!(
59     HID_TABLE,
60     MODULE_HID_TABLE,
```

RFC 3762, "Make trait methods callable in const contexts"

HID Abstractions in Rust

Rust language features that would help

4. This declarative macro expands such that `new_usb` is called from a const context

```
46          Self(bindings::hid_device_id {
47              bus: 0x3, // BUS_USB
48              group: group.into(),
49              vendor,
50              product,
51              driver_data: 0,
52          })
53      }
54  }
55
56  // ... Driver implementation
57
58  kernel::hid_device_table!(
59      HID_TABLE,
60      MODULE_HID_TABLE,
61      <GloriousRust as hid::Driver>::IdInfo,
62      [(
63          hid::DeviceId::new_usb(
64              hid::Group::Generic,
65              bindings::USB_VENDOR_ID_SINOWEALTH,
66              bindings::USB_DEVICE_ID_GLORIOUS_MODEL_O,
67          ),
68          (),
69      )]
70  );
```

RFC 3762, "Make trait methods callable in const contexts"

HID Abstractions in Rust

Rust language features that would help

5. Cannot implement `From` trait due to lack of `const_trait_impl`

```
    Wacom = bindings::HID_GROUP_WACOM as u16,
    LogitechDJDevice = bindings::HID_GROUP_LOGITECH_DJ_DEVICE as u16,
    Steam = bindings::HID_GROUP_STEAM as u16,
    Logitech27MHzDevice = bindings::HID_GROUP_LOGITECH_27MHZ_DEVICE as u16,
    Vivaldi = bindings::HID_GROUP_VIVALDI as u16,
}

// TODO: use `const_trait_impl` once stabilized:
//
// ```
// impl const From<Group> for u16 {
//     /// [`Group`] variants are represented by [`u16`] values.
//     fn from(value: Group) -> Self {
//         value as Self
//     }
// }
// ```
impl Group {
    /// Internal function used to convert [`Group`] variants into [`u16`].
    const fn into(self) -> u16 {
        self as u16
    }
}

/// Abstraction for the HID device ID structure `struct hid_device_id`.
```

RFC 3762, "Make trait methods callable in const contexts"

HID Abstractions in Rust

Quickly going over the core code

A lot of it uses Danilo's common infrastructure for driver bindings, compared to my initial RFC.

```
1  // SPDX-License-Identifier: GPL-2.0
2
3  // Copyright (C) 2025 Rahul Rameshbabu <sergeantsagara@protonmail.com>
4
5  //! Abstractions for the HID interface.
6  //!
7  //! C header: [`include/linux/hid.h`](srctree/include/linux/hid.h)
8
9  use crate::{device, device_id::RawDeviceId, driver, error::*, prelude::*,
   types::Opaque};
10 use core::{
11     marker::PhantomData,
12     ptr::{addr_of_mut, NonNull},
13 };
14
15 /// Indicates the item is static read-only.
16 ///
17 /// Refer to [Device Class Definition for HID 1.11]
18 /// Section 6.2.2.5 Input, Output, and Feature Items.
19 ///
20 /// [Device Class Definition for HID 1.11]:
   https://www.usb.org/sites/default/files/hid1_11.pdf
21 pub const MAIN_ITEM_CONSTANT: u8 = bindings::HID_MAIN_ITEM_CONSTANT as u8;
22
23 /// Indicates the item represents data from a physical control.
```

I plan to discuss this topic in much greater detail at LPC 2025, including the shortcomings of this implementation.

HID Abstractions in Rust

Difficulties making progress

NOTE: This is just my own interpretation from our mailing list discussions, and I apologize if this summary is inaccurate in any way.

Miguel expressed preference that the patches go through the HID tree and make its way to Linus through HID merge requests

Jiri Kosina (HID maintainer) has expressed preference for the work to go straight to the rust-for-linux tree without his involvement

Deadlock

Benjamin Tissoires (HID maintainer) has expressed preference to be added for code reviews but not necessarily have the patches go through the linux-input mailing list or hid tree

HID Abstractions in Rust

Maintenance with being out-of-tree

```
diff --git a/rust/kernel/hid.rs b/rust/kernel/hid.rs
index a93804af8b78..588483bf7204 100644
--- a/rust/kernel/hid.rs
+++ b/rust/kernel/hid.rs
@@ -246,7 +246,7 @@ fn as_ref(&self) -> &device::Device<Ctx> {
         let dev = unsafe { addr_of_mut!((*self.as_raw()).dev) };

         // SAFETY: `dev` points to a valid `struct device`.
-        unsafe { device::Device::as_ref(dev) }
+        unsafe { device::Device::from_raw(dev) }
     }
 }

@@ -297,12 +297,6 @@ pub fn product(&self) -> u32 {
 // * `DRIVER_DATA_OFFSET` is the offset to the `driver_data` field.
 unsafe impl RawDeviceId for DeviceId {
     type RawType = bindings::hid_device_id;
-
-    const DRIVER_DATA_OFFSET: usize = core::mem::offset_of!(bindings::hid_device_id, driver_data);
-
-    fn index(&self) -> usize {
-        self.0.driver_data
-    }
 }

 /// [`IdTable`] type for HID.
```

HID Abstractions in Rust

Questions for follow-up work

HID Abstractions in Rust
Questions for follow-up work

- Should I start developing more changes on top of the patches out on the mailing list?

HID Abstractions in Rust
Questions for follow-up work

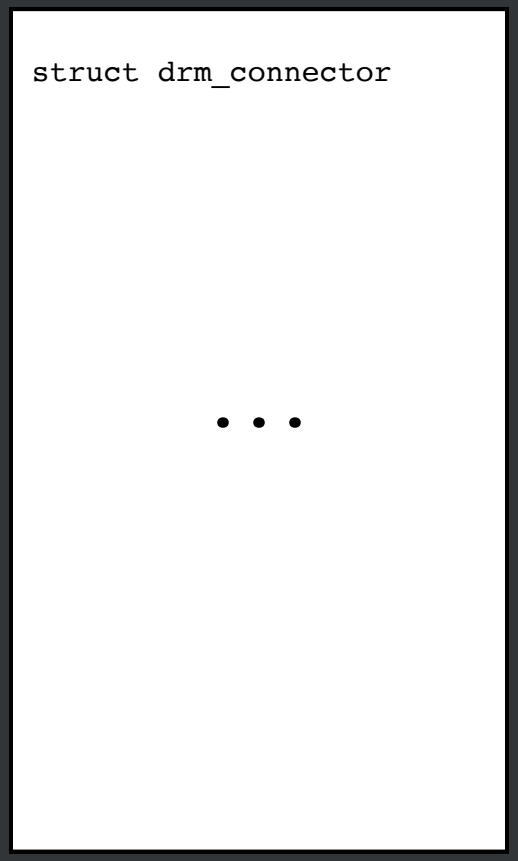
- Should I start developing more changes on top of the patches out on the mailing list?
- If so, should I start sending those out for review to the mailing list as well?

HID Abstractions in Rust
Questions for follow-up work

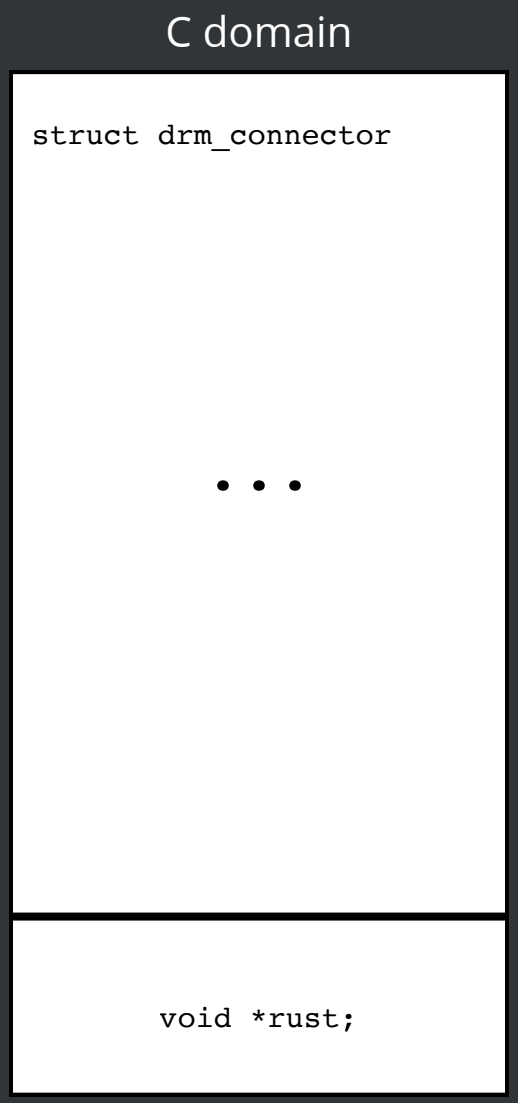
- Should I start developing more changes on top of the patches out on the mailing list?
- If so, should I start sending those out for review to the mailing list as well?
- Are there any alternative methods for handling the patches that would make all maintainers happy? Maybe I send out the patches to rust-for-linux with Benjamin CCed, and I send merge requests to Linus's tree after going through the appropriate reviews?

Extending `drm_connector` in Rust

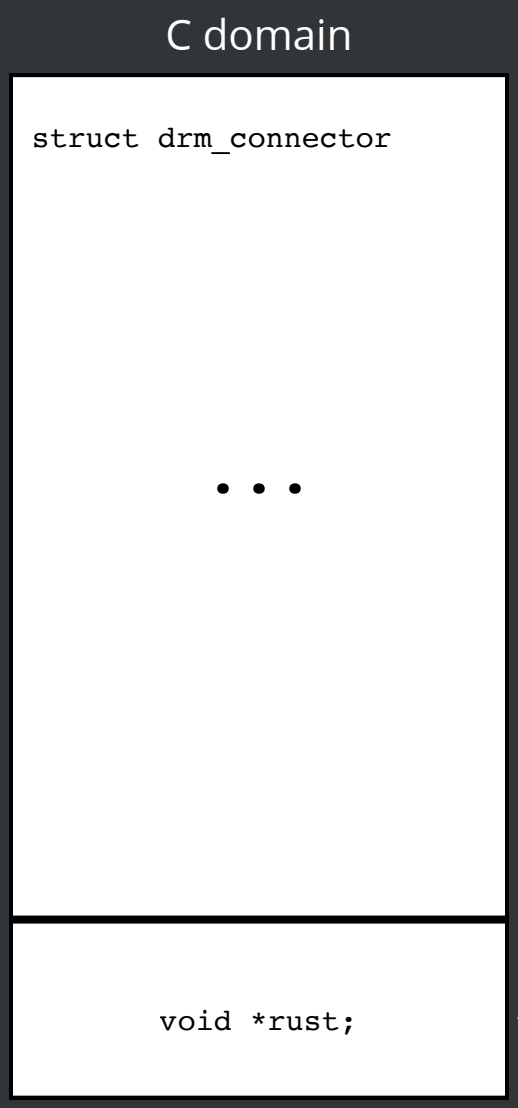
C domain



Extending `drm_connector` in Rust

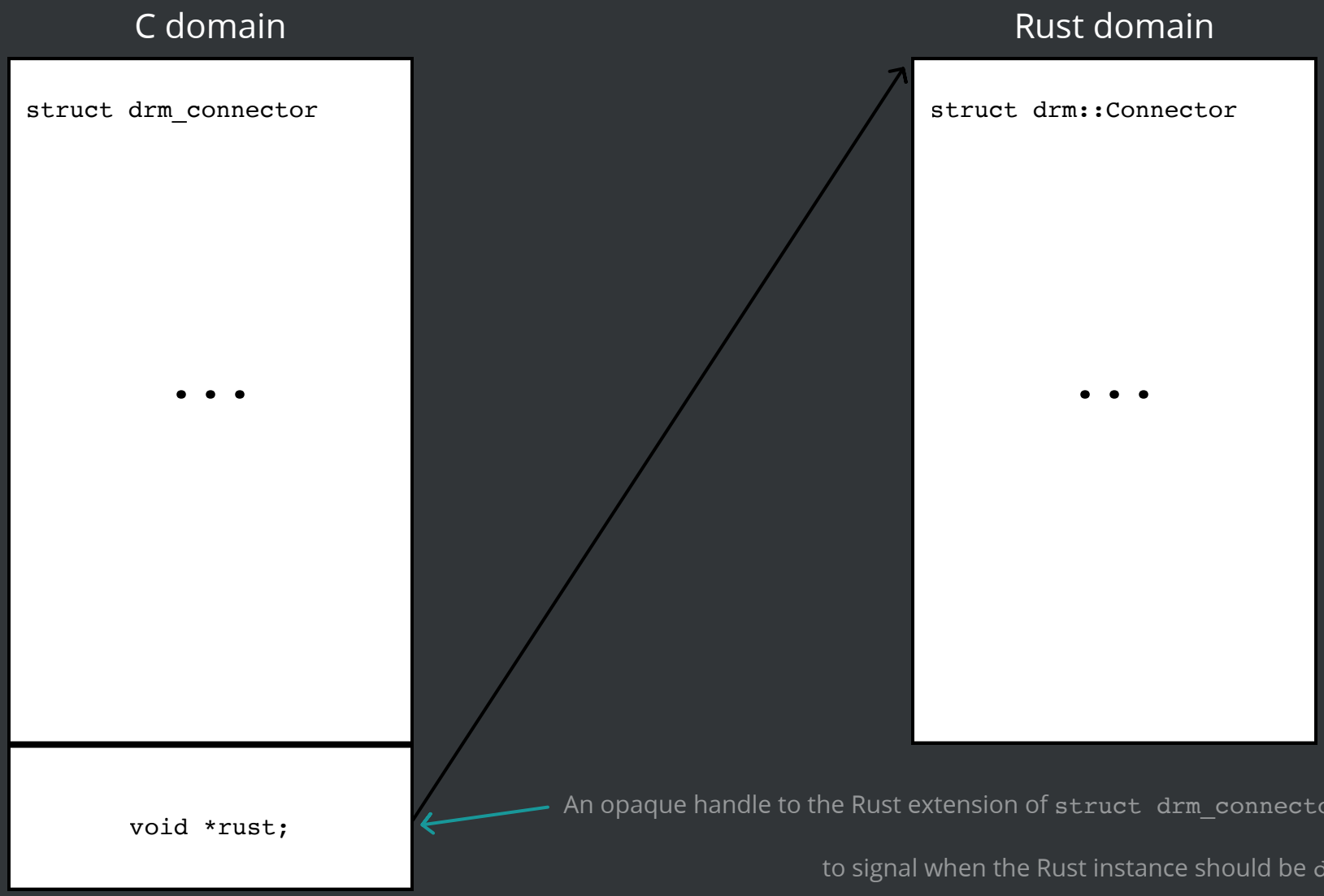


Extending `drm_connector` in Rust



An opaque handle to the Rust extension of `struct drm_connector` used to signal when the Rust instance should be `drop`-ed

Extending `drm_connector` in Rust



Extending `drm_connector` in Rust

Examining the Rust code

```rust
/// A DRM connector representation that extends `struct drm_connector`.
///
/// This connector implementation enables DRM connector API development in Rust
/// and exposing said functionality to both C and Rust DRM consumers.
#[pin_data]
pub struct Connector {
    #[pin]
    raw_connector: Opaque<*mut bindings::drm_connector>,
    rust_only_attribute: bool,

    /// A connector needs to be pinned since it is referred to using a raw
    /// pointer field `rust` in the C DRM `struct drm_connector` implementation.
    #[pin]
    _pin: PhantomPinned,
}
```

Extending `drm_connector` in Rust

Examining the Rust code

```rust
/// A DRM connector representation that extends `struct drm_connector`.
///
/// This connector implementation enables DRM connector API development in Rust
/// and exposing said functionality to both C and Rust DRM consumers.
#[pin_data]
pub struct Connector {
    #[pin]
    raw_connector: Opaque<*mut bindings::drm_connector>,
    rust_only_attribute: bool,

    /// A connector needs to be pinned since it is referred to using a raw
    /// pointer field `rust` in the C DRM `struct drm_connector` implementation.
    #[pin]
    _pin: PhantomPinned,
}
```

An appendage for C's

`struct drm_connector`

Extending `drm_connector` in Rust

Examining the Rust code

```rust
/// A DRM connector representation that extends `struct drm_connector`.
///
/// This connector implementation enables DRM connector API development in Rust
/// and exposing said functionality to both C and Rust DRM consumers.
#[pin_data]
pub struct Connector {
    #[pin]
    raw_connector: Opaque<*mut bindings::drm_connector>,
    rust_only_attribute: bool,

    /// A connector needs to be pinned since it is referred to using a raw
    /// pointer field `rust` in the C DRM `struct drm_connector` implementation.
    #[pin]
    _pin: PhantomPinned,
}
```

An appendage for C's

`struct drm_connector`

Need to make sure to pin `drm::Connector` so the memory is not moved from the handle provided to the C structure

Extending `drm_connector` in Rust

Examining the Rust code

```rust
/// A DRM connector representation that extends `struct drm_connector`.
///
/// This connector implementation enables DRM connector API development in Rust
/// and exposing said functionality to both C and Rust DRM consumers.
#[pin_data]
pub struct Connector {
    #[pin]
    raw_connector: Opaque<*mut bindings::drm_connector>,
    rust_only_attribute: bool,

    /// A connector needs to be pinned since it is referred to using a raw
    /// pointer field `rust` in the C DRM `struct drm_connector` implementation.
    #[pin]
    _pin: PhantomPinned,
}
```

An appendage for C's

`struct drm_connector`

An example field in Rust not directly

present in C's `struct drm_connector`

Need to make sure to pin `drm::Connector` so the memory is not moved from the handle provided to the C structure

Extending `drm_connector` in Rust

Examining the Rust code

```rust
/// A DRM connector representation that extends `struct drm_connector`.
///
/// This connector implementation enables DRM connector API development in Rust
/// and exposing said functionality to both C and Rust DRM consumers.
#[pin_data]
pub struct Connector {
    #[pin]
    raw_connector: Opaque<*mut bindings::drm_connector>,
    rust_only_attribute: bool,

    /// A connector needs to be pinned since it is referred to using a raw
    /// pointer field `rust` in the C DRM `struct drm_connector` implementation.
    #[pin]
    _pin: PhantomPinned,
}
```

An appendage for C's `struct drm_connector`

Pin-ing of `raw_connector` feels unnecessary to me since its a pointer to a C structure. We can move this conversation to hallway track if we are running low on time

An example field in Rust not directly present in C's `struct drm_connector`

Need to make sure to pin `drm::Connector` so the memory is not moved from the handle provided to the C structure

Extending `drm_connector` in Rust

Examining the Rust code

```rust
/// A DRM connector representation that extends `struct drm_connector`.
///
/// This connector implementation enables DRM connector API development in Rust
/// and exposing said functionality to both C and Rust DRM consumers.
#[pin_data]
pub struct Connector {
    #[pin]
    raw_connector: Opaque<*mut bindings::drm_connector>,
    rust_only_attribute: bool,

    /// A connector needs to be pinned since it is referred to using a raw
    /// pointer field `rust` in the C DRM `struct drm_connector` implementation.
    #[pin]
    _pin: PhantomPinned,
}
```

An appendage for C's `struct drm_connector`

Pin-ing of `raw_connector` feels unnecessary to me since its a pointer to a C structure. We can move this conversation to hallway track if we are running low on time

An example field in Rust not directly present in C's `struct drm_connector`

Need to make sure to pin `drm::Connector` so the memory is not moved from the handle provided to the C structure

Extending `drm_connector` in Rust

Examining the Rust code

```rust
/// C entry point for initializing the Rust extension for a DRM connector.
///
/// When a DRM connector is being initialized in the core C stack, the Rust
/// `Connector` extension needs to be allocated and initialized.
#[export]
pub unsafe extern "C" fn drm_connector_init_rust(
    raw_connector: *mut bindings::drm_connector,
) -> kernel::ffi::c_int {
    let connector = match KBox::pin_init(
        try_pin_init!(Connector{
            raw_connector <- Opaque::new(raw_connector),
            rust_only_attribute: true,
            _pin: PhantomPinned,
        }),
        GFP_KERNEL,
    ) {
        Ok(kbox) => kbox,
        Err(_) => return -ENOMEM.to_errno(),
    };

    // Provide the C `struct drm_connector` instance a handle to the Rust
    // `drm::connector:Connector` implementation for Rust connector APIs and the
    // `drm_connector_cleanup_rust` cleanup call.
    //
    // SAFETY: `raw_connector` is a valid pointer with a `rust` field that does
```

Extending `drm_connector` in Rust

Examining the Rust code

Can be called from C code in the kernel

```rust
/// C entry point for initializing the Rust extension for a DRM connector.
///
/// When a DRM connector is being initialized in the core C stack, the Rust
/// `Connector` extension needs to be allocated and initialized.
#[export]
pub unsafe extern "C" fn drm_connector_init_rust(
    raw_connector: *mut bindings::drm_connector,
) -> kernel::ffi::c_int {
    let connector = match KBox::pin_init(
        try_pin_init!(Connector{
            raw_connector <- Opaque::new(raw_connector),
            rust_only_attribute: true,
            _pin: PhantomPinned,
        }),
        GFP_KERNEL,
    ) {
        Ok(kbox) => kbox,
        Err(_) => return -ENOMEM.to_errno(),
    };

    // Provide the C `struct drm_connector` instance a handle to the Rust
    // `drm::connector:Connector` implementation for Rust connector APIs and the
    // `drm_connector_cleanup_rust` cleanup call.
    //
    // SAFETY: `raw_connector` is a valid pointer with a `rust` field that does
```

Extending `drm_connector` in Rust

Examining the Rust code

Pin-init a Rust `drm::Connector` that will be referred to by `struct drm_connector`

```
2  ///
3  /// When a DRM connector is being initialized in the core C stack, the Rust
4  /// `Connector` extension needs to be allocated and initialized.
5  #[export]
6  pub unsafe extern "C" fn drm_connector_init_rust(
7      raw_connector: *mut bindings::drm_connector,
8  ) -> kernel::ffi::c_int {
9      let connector = match KBox::pin_init(
10         try_pin_init!(Connector{
11             raw_connector <- Opaque::new(raw_connector),
12             rust_only_attribute: true,
13             _pin: PhantomPinned,
14         }),
15         GFP_KERNEL,
16     ) {
17         Ok(kbox) => kbox,
18         Err(_) => return -ENOMEM.to_errno(),
19     };
20
21     // Provide the C `struct drm_connector` instance a handle to the Rust
22     // `drm::connector:Connector` implementation for Rust connector APIs and the
23     // `drm_connector_cleanup_rust` cleanup call.
24     //
25     // SAFETY: `raw_connector` is a valid pointer with a `rust` field that does
26     // not already point to an initialized `drm::connector::Connector`
```

Extending `drm_connector` in Rust

Examining the Rust code

Provide a raw pointer to the pinned `drm::Connector` for `struct drm_connector`

```
 6 pub unsafe extern "C" fn drm_connector_init_rust(
 7     raw_connector: *mut bindings::drm_connector,
 8 ) -> kernel::ffi::c_int {
 9     let connector = match KBox::pin_init(
10         try_pin_init!(Connector{
11             raw_connector <- Opaque::new(raw_connector),
12             rust_only_attribute: true,
13             _pin: PhantomPinned,
14         }),
15         GFP_KERNEL,
16     ) {
17         Ok(kbox) => kbox,
18         Err(_) => return -ENOMEM.to_errno(),
19     };
20
21     // Provide the C `struct drm_connector` instance a handle to the Rust
22     // `drm::connector:Connector` implementation for Rust connector APIs and the
23     // `drm_connector_cleanup_rust` cleanup call.
24     //
25     // SAFETY: `raw_connector` is a valid pointer with a `rust` field that does
26     // not already point to an initialized `drm::connector::Connector`
27     unsafe { (*raw_connector).rust = connector.into_foreign() };
28
29     0
30 }
```

Extending `drm_connector` in Rust

Examining the Rust code

```rust
/// C entry point for tearing down the Rust extension for a DRM connector.
///
/// When a DRM connector is being cleaned up from the core C stack, the Rust
/// `Connector` extension instance needs to be dropped.
#[export]
pub unsafe extern "C" fn drm_connector_cleanup_rust(raw_connector: *mut
bindings::drm_connector) {
    // SAFETY: By the safety requirements of this function, the `rust` field of
    // `raw_connector`, a valid pointer, is initialized by the `into_foreign()`
    // call made by `drm_connector_init_rust()`.
    drop(unsafe { <Pin<KBox<Connector>>>::from_foreign((*raw_connector).rust) });
}
```

Extending `drm_connector` in Rust

Examining the C code

In `include/drm/drm_connector.h`, declare the Rust-implemented functions. These declarations probably could be moved into `drivers/gpu/drm/drm_connector.c`

```
#if IS_ENABLED(CONFIG_RUST)
int drm_connector_init_rust(struct drm_connector *connector);
void drm_connector_cleanup_rust(struct drm_connector *connector);
#else
static inline int drm_connector_init_rust(struct drm_connector *connector)
{
    return 0;
}

static inline void drm_connector_cleanup_rust(struct drm_connector *connector)
{
}
#endif
```

Extending `drm_connector` in Rust

Examining the C code

```
1  static int drm_connector_init_only(struct drm_device *dev,
2                     struct drm_connector *connector,
3                     const struct drm_connector_funcs *funcs,
4                     int connector_type,
5                     struct i2c_adapter *ddc)
6  {
7      struct drm_mode_config *config = &dev->mode_config;
8      int ret;
9      struct ida *connector_ida =
10         &drm_connector_enum_list[connector_type].ida;
11
12     WARN_ON(drm_drv_uses_atomic_modeset(dev) &&
13         (!funcs->atomic_destroy_state ||
14          !funcs->atomic_duplicate_state));
15
16     ret = __drm_mode_object_add(dev, &connector->base,
17                     DRM_MODE_OBJECT_CONNECTOR,
18                     false, drm_connector_free);
19     if (ret)
20         return ret;
21
22     connector->base.properties = &connector->properties;
23     connector->dev = dev;
24     connector->funcs = funcs;
```

Extending `drm_connector` in Rust

Examining the C code

Call `drm_connector_init_rust` from a core DRM connector initialization helper

```
42          goto out_put_id;
43      }
44      connector->name =
45          kasprintf(GFP_KERNEL, "%s-%d",
46                drm_connector_enum_list[connector_type].name,
47                connector->connector_type_id);
48      if (!connector->name) {
49          ret = -ENOMEM;
50          goto out_put_type_id;
51      }
52
53      ret = drm_connector_init_rust(connector);
54      if (ret)
55          goto out_put_name;
56
57      /* provide ddc symlink in sysfs */
58      connector->ddc = ddc;
59
60      INIT_LIST_HEAD(&connector->head);
61      INIT_LIST_HEAD(&connector->global_connector_list_entry);
62      INIT_LIST_HEAD(&connector->probed_modes);
63      INIT_LIST_HEAD(&connector->modes);
64      mutex_init(&connector->mutex);
65      mutex_init(&connector->cec.mutex);
```

Extending `drm_connector` in Rust

Examining the C code

```c
void drm_connector_cleanup(struct drm_connector *connector)
{
    struct drm_device *dev = connector->dev;
    struct drm_display_mode *mode, *t;

    /* The connector should have been removed from userspace long before
     * it is finally destroyed.
     */
    if (WARN_ON(connector->registration_state ==
                DRM_CONNECTOR_REGISTERED))
        drm_connector_unregister(connector);

    drm_connector_cleanup_rust(connector);

    platform_device_unregister(connector->hdmi_audio.codec_pdev);

    if (connector->privacy_screen) {
        drm_privacy_screen_put(connector->privacy_screen);
        connector->privacy_screen = NULL;
    }

    if (connector->tile_group) {
        drm_mode_put_tile_group(dev, connector->tile_group);
        connector->tile_group = NULL;
```

Extending `drm_connector` in Rust

Examining the C code

Call `drm_connector_cleanup_rust` from a core DRM connector cleanup function

```
1  void drm_connector_cleanup(struct drm_connector *connector)
2  {
3      struct drm_device *dev = connector->dev;
4      struct drm_display_mode *mode, *t;
5
6      /* The connector should have been removed from userspace long before
7       * it is finally destroyed.
8       */
9      if (WARN_ON(connector->registration_state ==
10                 DRM_CONNECTOR_REGISTERED))
11         drm_connector_unregister(connector);
12
13     drm_connector_cleanup_rust(connector);
14
15     platform_device_unregister(connector->hdmi_audio.codec_pdev);
16
17     if (connector->privacy_screen) {
18         drm_privacy_screen_put(connector->privacy_screen);
19         connector->privacy_screen = NULL;
20     }
21
22     if (connector->tile_group) {
23         drm_mode_put_tile_group(dev, connector->tile_group);
24         connector->tile_group = NULL;
25     }
```

Extending `drm_connector` in Rust

Concerns shared on the mailing list

> Thanks for the explanation.
>
> I'm not sure Rust is at the point where we can use it for the framework.
> If we want to make this work useful, we have to make it consistent and
> usable across all drivers, but we do have drivers for architectures that
> aren't supported by Rust yet (let alone tier 1).
>
> So it feels to me that it would be a bit premature for that work to be
> in Rust. If you do want to use it from a Rust driver though, feel free
> to write bindings for it, that would be a great addition.
>
> Maxime

https://lore.kernel.org/rust-for-linux/20250827-cherubic-tapir-of-wind-5cf0c4@houat/

Extending `drm_connector` in Rust

Analyzing the state of Rust support in the kernel

Architecture	Toolchain Tier	Constraints
`arm`	Tier 2	`ARMv7` Little Endian only.
`arm64`	Tier 1	Little Endian only.
`loongarch`	Tier 2	
`riscv`	Tier 2	`riscv64` and LLVM/Clang only.
`x86`	Tier 1	x86_64 only.

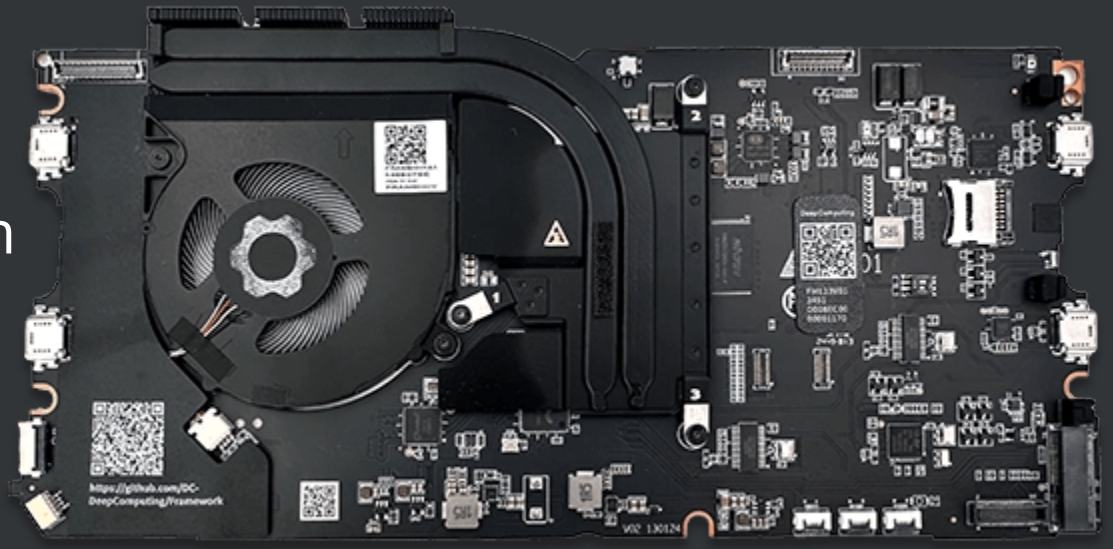
- https://docs.kernel.org/rust/arch-support.html
- https://doc.rust-lang.org/rustc/platform-support.html

Extending `drm_connector` in Rust

My take on the current Rust toolchain support

Many consumer electronics platforms will run `x86_64`, `aarch64` little endian, or `RISC-V`. I, personally, have considered running a RISC-V laptop using the DeepComputing RISC-V mainboard.

`RISC-V` is tier 2 in Rust toolchain support.



Extending `drm_connector` in Rust

My take on the current Rust toolchain support

I am not entirely sure the other architectures matter

I am having a hard time imagining someone hooking up consumer displays or panels to other architectures and wanting panel control support

Extending `drm_connector` in Rust

Rust for Linux distro availability

Rust support in the kernel is marked experimental

Having a feature like this require Rust support might conflict with the intent of marking support "experimental", given how it would be widely used for personal computing applications of Linux

`CONFIG_RUST` enabled in distro kernels

- NixOS
- Gentoo (trivial to enable)
- Arch Linux
- Ubuntu

Extending `drm_connector` in Rust

What should we do next?

First, I want to heavily discuss with folks like Maxime Ripard. I plan to start with architecture design of the feature. The topic is language-agnostic.

In parallel, I plan to share this presentation with Maxime and get a better sense of his concerns.

Based on these conversations, either I go ahead with the implementation in Rust or do the core implementation in C.

Thank yous

Miguel, Danilo, and Benno have been instrumental in my journey writing Rust patches for the Linux kernel.

I am grateful to Benjamin Tissoires and Jiri Kosina for taking the time to review my changes on the HID side of things.

Thanks, Maxime Ripard, for the feedback on the mailing list, and I am excited to share both architecture details and patches for supporting brightness control as well as other monitor control properties in the future.

I would like to thank Jason Gunthorpe for providing me help when I wanted to first contribute to the Linux kernel and for being a big inspiration for me. Jason has also been supportive of me when I want to explore these types of topics and independently contribute to the kernel.

That was a mouthful.
Any questions?