```
   _
+-/ \-+
| (o) |
+-----+
```

Camera sensor compliance

EOSS 2023

Jacopo Mondi
jacopo.mondi@ideasonboard.com

Hello, I’m Jacopo

- Embedded camera engineer @ Ideas On Board Oy
  - *Video4Linux2*
  - *libcamera*

Hello!

Software compliance

What we are **not** talking about today

- license compliance
- law compliance (GDPR...)
- process compliance
- standard compliance

API compliance

An API specification is a formal definition of an *interface* between software components

- comparable to a *standard*
- can be expressed:
    - in documentation format
    - in formal language description
    - ..

Testing compliance

compliance validation

- static code analysis
    - code auditing
    - linters
    - some AI-buzzword

Testing compliance

compliance validation

- run time validation
  - unit-testing
  - fuzzying
  - correctness checks

v4l2-compliance

compliance validation

- run time validation: **v4l2-compliance**

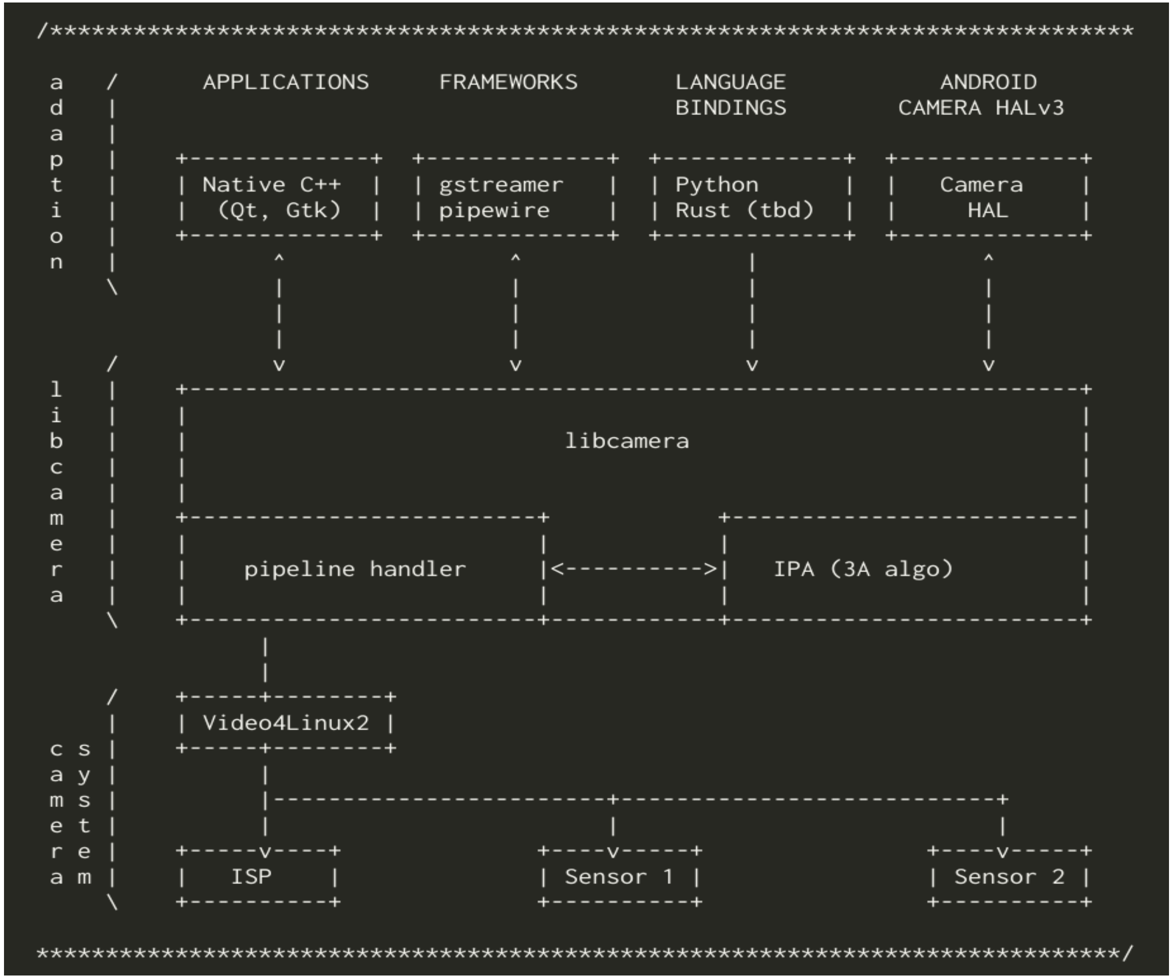
- Video4Linux2 utils suite (*v4l-utils)*

- tests for:
  - the driver supported operations
  - unit tests/fuzzer the implementation to verify correctness

v4l2-compliance

compliance validation

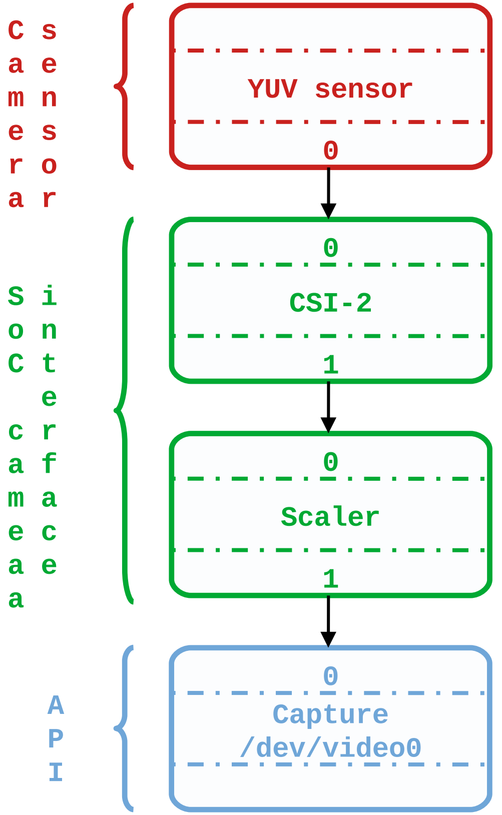
- run time validation: **v4l2-compliance**
- is API compliance enough to guarantee interoperability ?



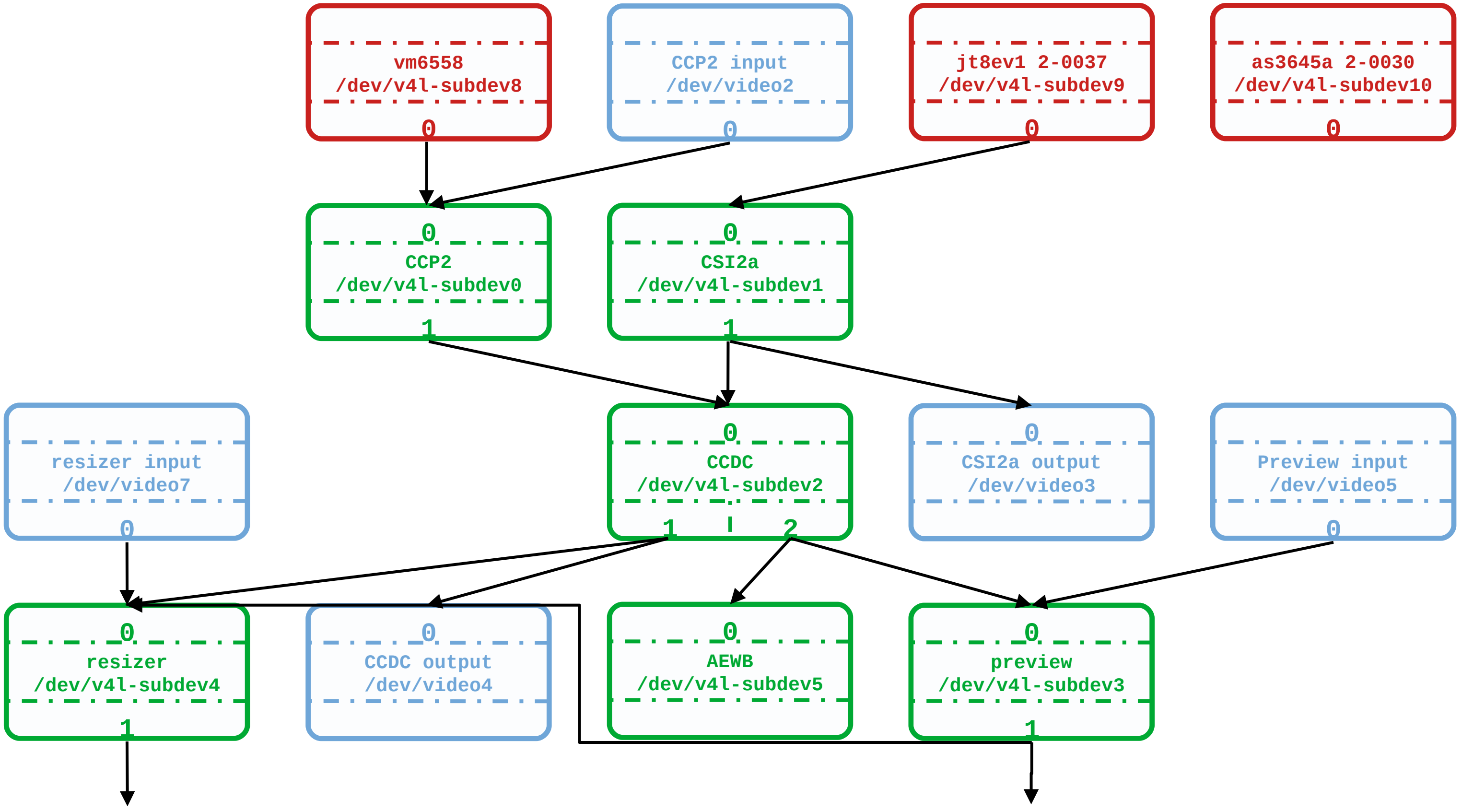
```
/******************************************************************************

 a   /      APPLICATIONS     FRAMEWORKS       LANGUAGE           ANDROID
 d   |                                        BINDINGS        CAMERA HALv3
 a   |
 p   |    +-------------+  +-------------+  +-------------+  +-------------+
 t   |    | Native C++  |  | gstreamer   |  | Python      |  |   Camera    |
 i   |    |  (Qt, Gtk)  |  | pipewire    |  | Rust (tbd)  |  |     HAL     |
 o   |    +-------------+  +-------------+  +-------------+  +-------------+
 n   |           ^                ^                |                ^
     \           |                |                |                |
                 |                |                |                |
                 |                |                |                |
     /           v                v                v                v
 l   |    +----------------------------------------------------------------+
 i   |    |                                                                |
 b   |    |                           libcamera                            |
 c   |    |                                                                |
 a   |    |                                                                |
 m   |    +-------------------------+            +-------------------------|
 e   |    |                         |            |                         |
 r   |    |    pipeline handler     |<---------->|   IPA (3A algo)         |
 a   |    |                         |            |                         |
     \    +-------------------------+------------+-------------------------+
                |
                |
     /    +-----+--------+
     |    | Video4Linux2 |
 c s |    +-----+--------+
 a y |          |
 m s |          |------------------------+---------------------------+
 e t |          |                        |                           |
 r e |    +-----v----+              +----v-----+                +----v-----+
 a m |    |   ISP    |              | Sensor 1 |                | Sensor 2 |
     \    +----------+              +----------+                +----------+

******************************************************************************/
```

libcamera: a standard consumer of the V4L2 API

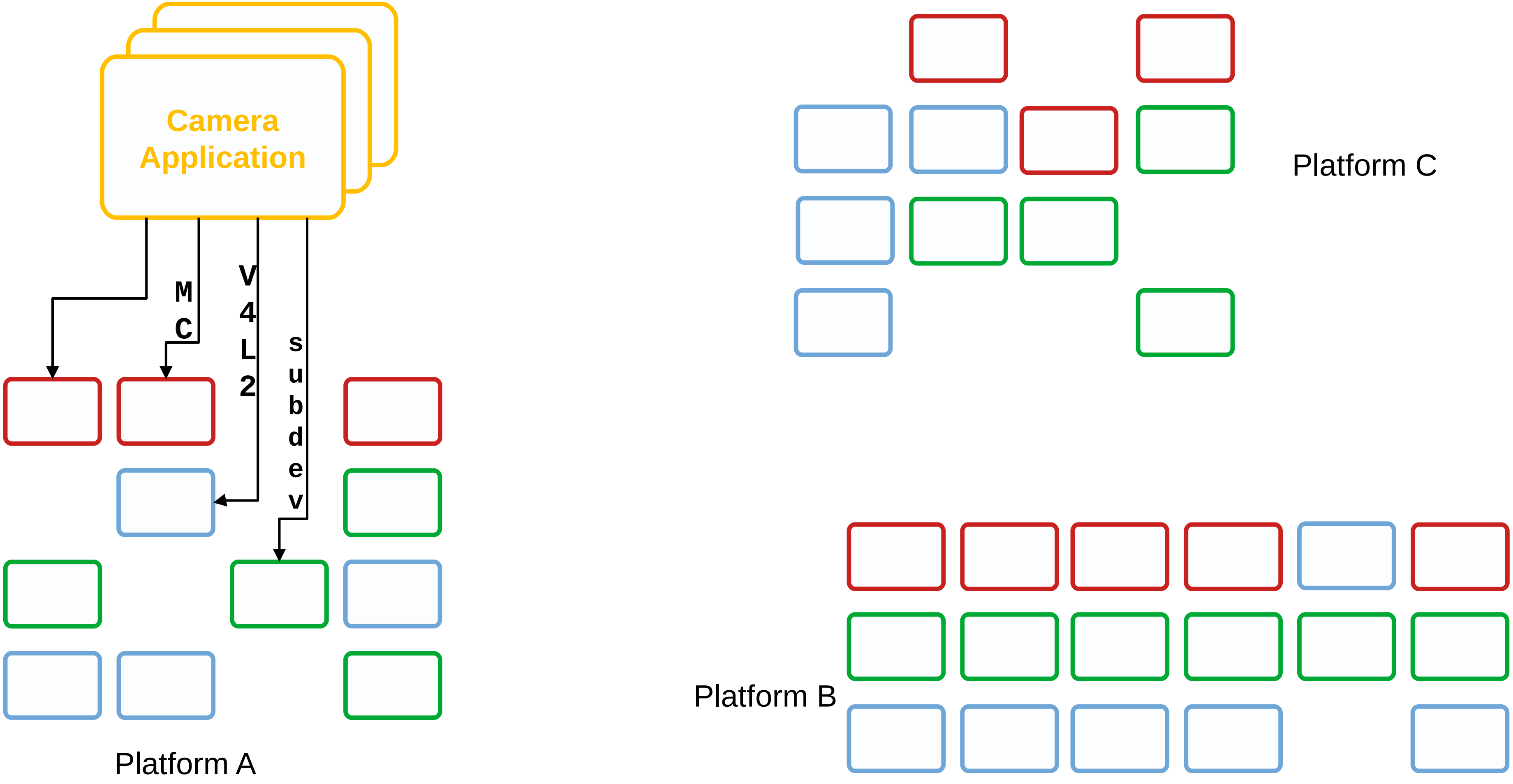
Cameras got complex (a long time ago..)



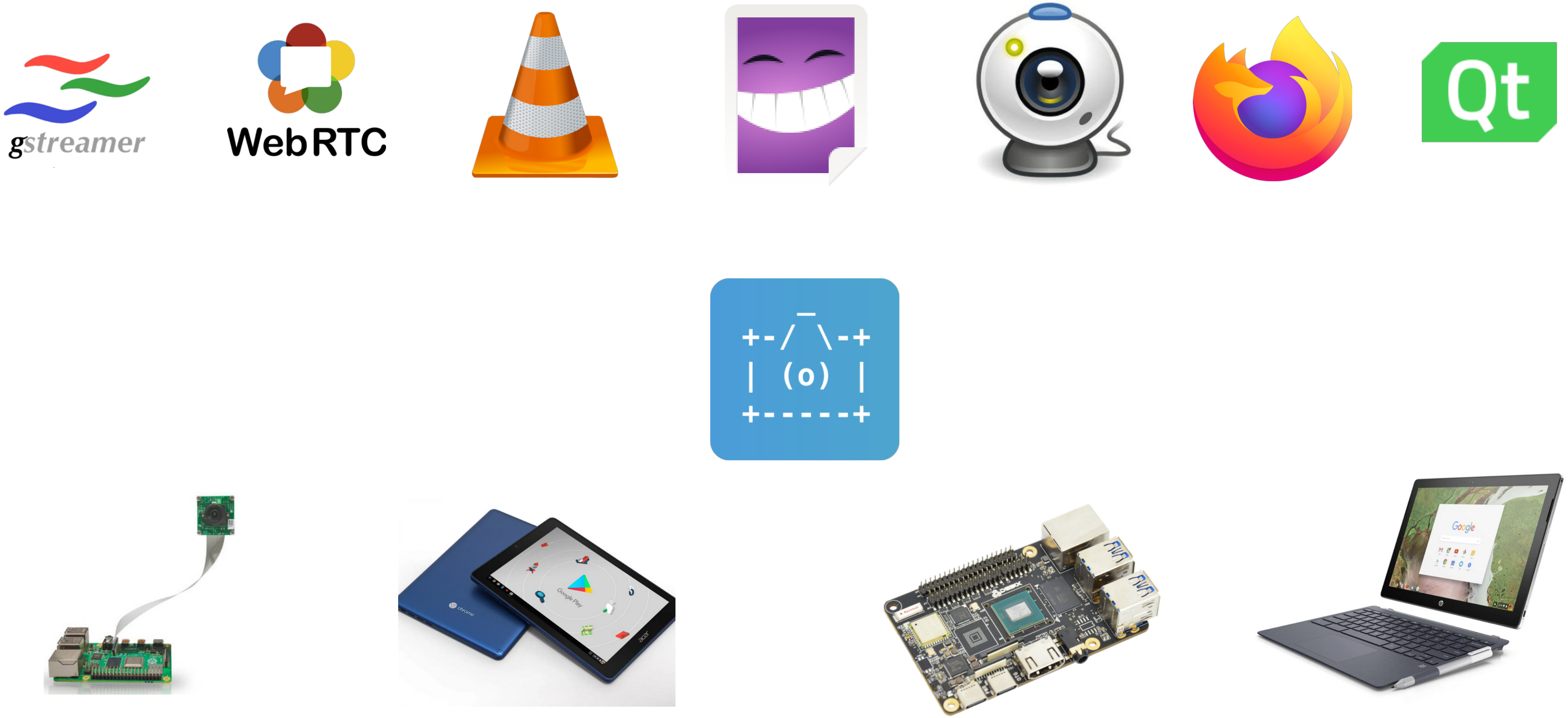
Simple camera



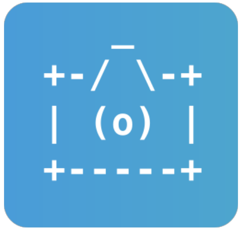
OMAP3 Camera in Nokia N900 - **2009**

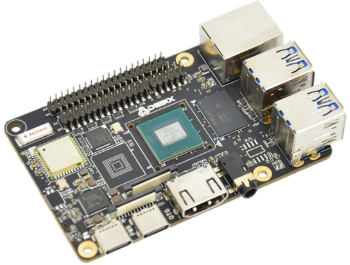


… and applications had to be platform-specific



libcamera abstracts away platform details





… but what about sensors ?

A single consumer for multiple sensor driver implementations

what could possibly go wrong ?

A single consumer for multiple sensor driver implementations

- focus is on RAW sensor with a Bayer patter color filter array
- this presentation aims to share the pain we experienced while consuming the several different interpretation of the V4L2 APIs
- it aims to provide sensor driver developers tips to avoid the most common pitfalls

Basic feature set

- exposure/gain control
- flips control
- rotation
- analog crop rectangle
- blankings controls

libcamera needs to control the sensor exposure and gain:

- computed by the AEGC algorithm (*auto* mode)
- specified by the user (*manual* mode)

Exposure time is expressed as a duration (micro-seconds)

Gain is expressed as a multiplier applied to all color channels

Exposure/gain handling

Exposure

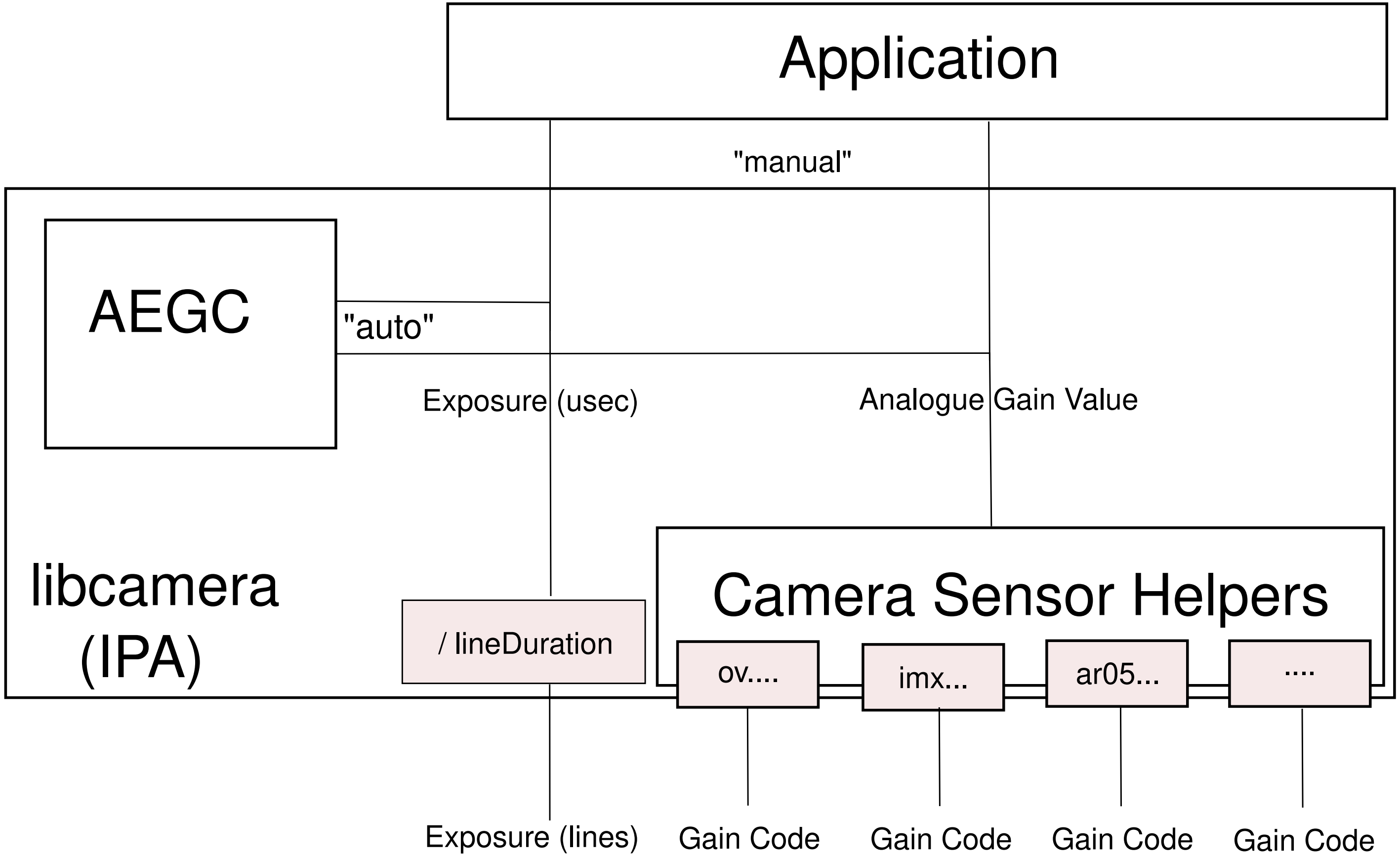
- controlled through V4L2_CID_EXPOSURE
- *typically* expressed as a number of lines
- the V4L2 specification *does not* specify a unit for the control
  - some drivers use lines
  - some other uses fraction of lines
  - they’re all technically compliant to the spec
  - but impossible to interoperate generically

Exposure/gain handling

Analogue Gain

- controlled through V4L2_CID_ANALOGUE_GAIN
- (but some sensor drivers use V4L2_CID_GAIN!)
- the control unit is *device specific*
    - *gain code:* it’s actually the register value!
- usually poorly documented

Exposure/gain handling



Exposure/gain handling

Tips for driver implementers:

- Use lines as V4L2_CID_EXPOSURE unit
    - *it's unlikely you need to control sub-line duration exposure times*

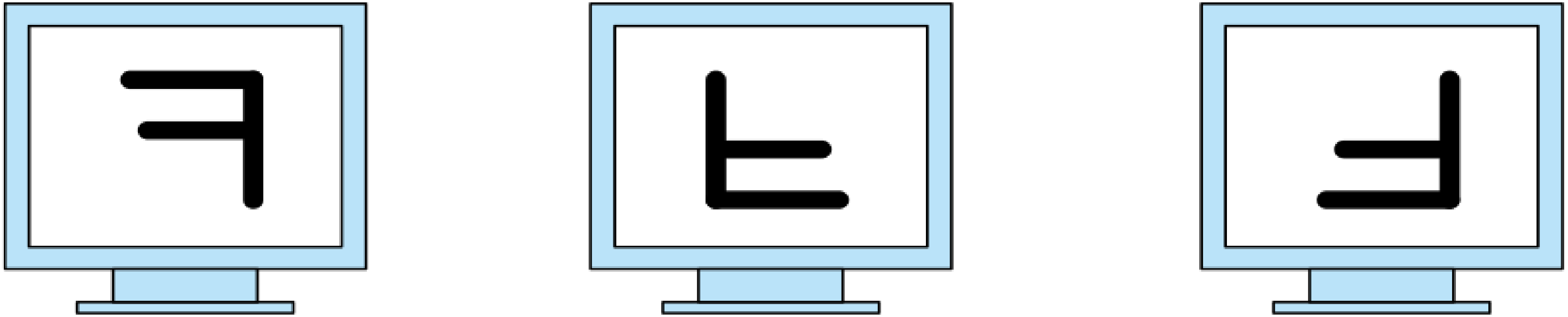
- Use the device gain code as V4L2_CID_ANALOGUE_GAIN
    - *and provide a CameraSensorHelper implementation in libcamera*

- Whenever possible split digital and analogue gain handling
    - IOW please don't use V4L2_CID_GAIN for RAW sensors

Exposure/gain handling

V/H_FLIP

- Control the pixel readout order
  - horizontal mirroring
  - vertical flip
  - 180 degrees rotation



Vertical and horizontal flips

Vertical and horizontal flips

V/H_FLIP

- Control the pixel readout order
    - horizontal mirroring
    - vertical flip
    - 180 degrees rotation

- Considered to be “simple” controls
    - but they have subtle implications for raw sensors

V/H_FLIP

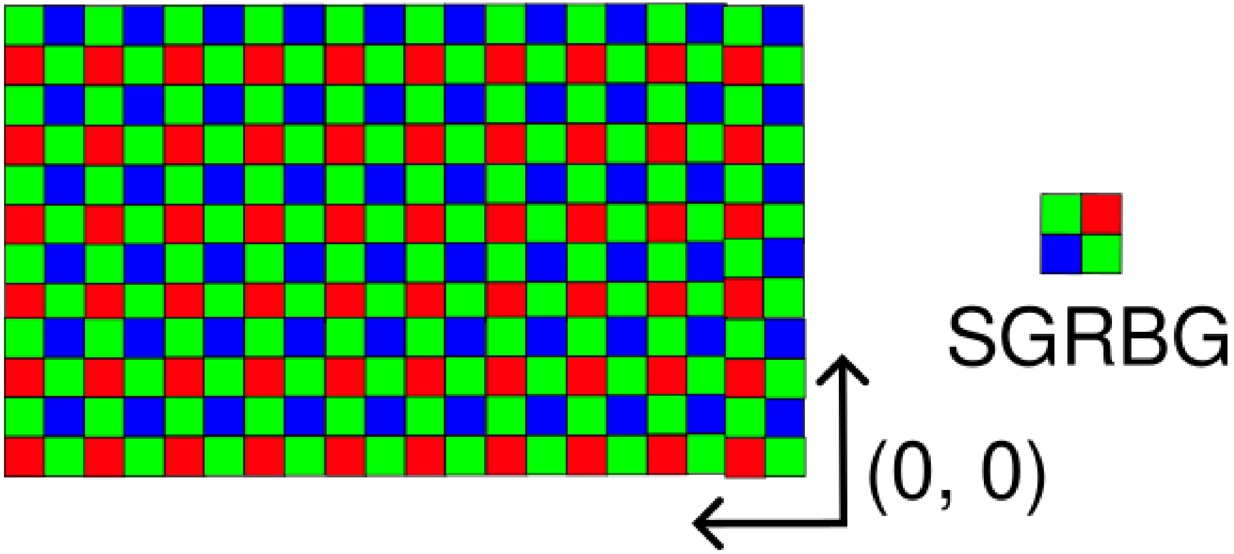
- Control the pixel readout order
  - horizontal mirroring
  - vertical flip
  - 180 degrees rotation

- Considered to be “simple” controls
  - but they have subtle implications for raw sensors

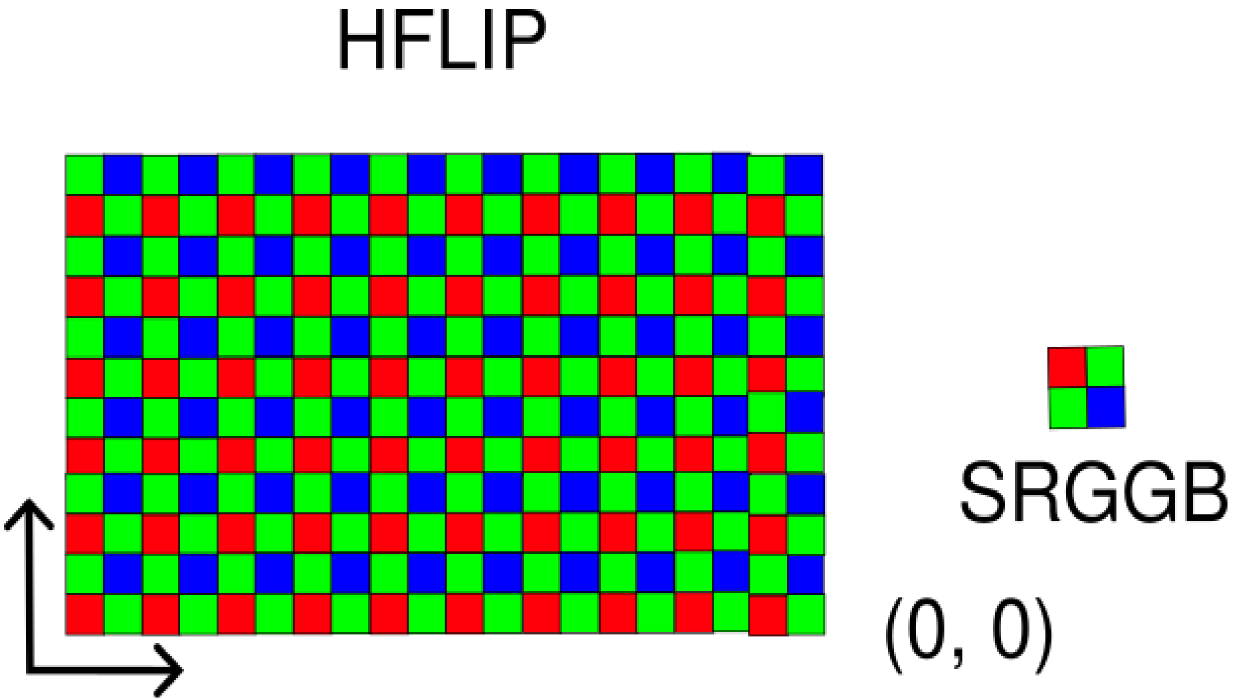
  - *they change the image format without userspace noticing it*

Vertical and horizontal flips

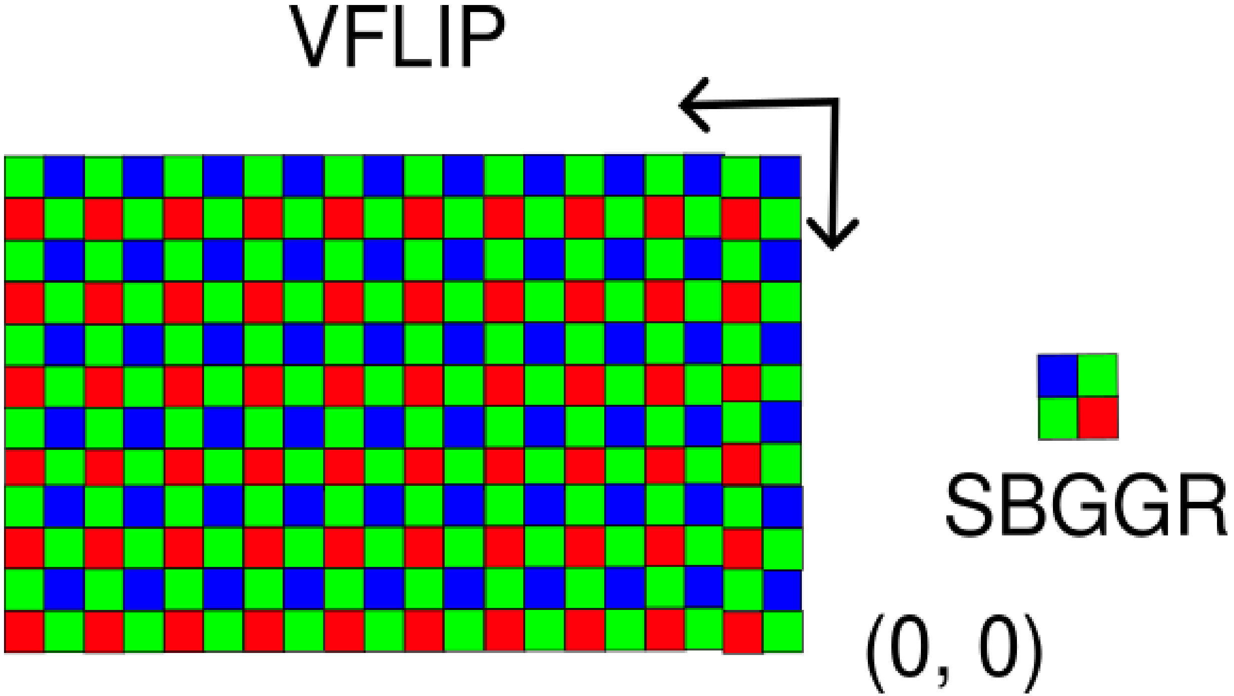
NO FLIPs



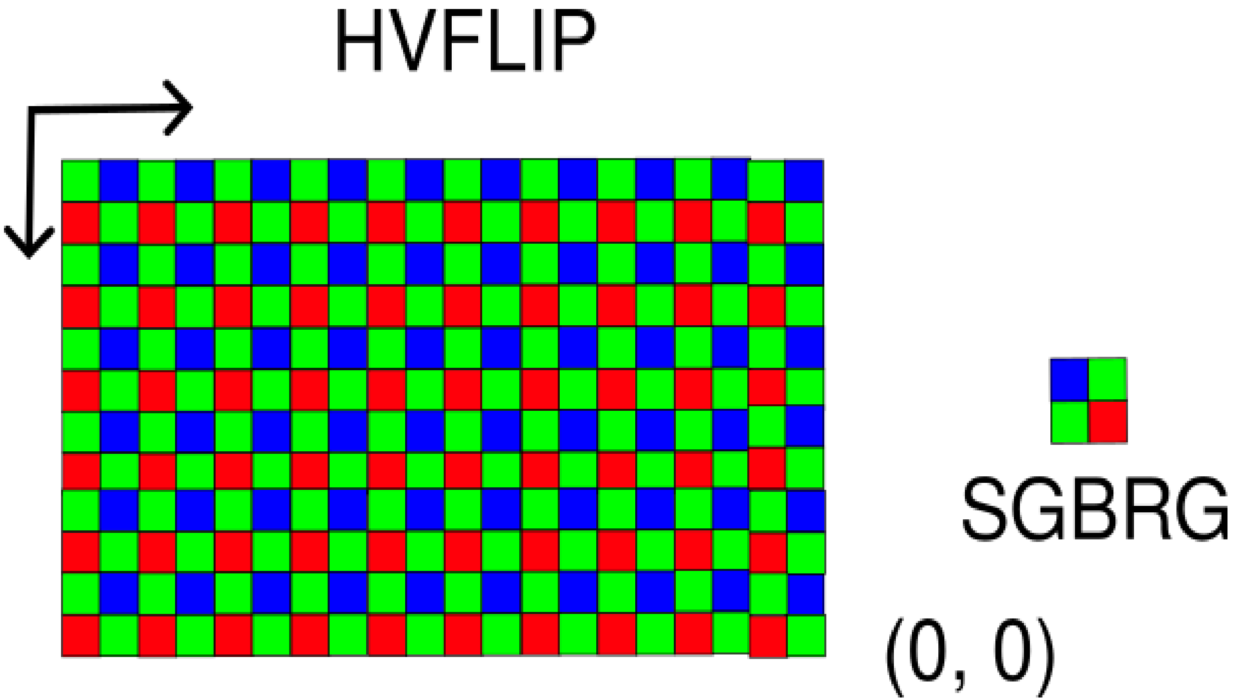
Vertical and horizontal flips



Vertical and horizontal flips



Vertical and horizontal flips



HVFLIP

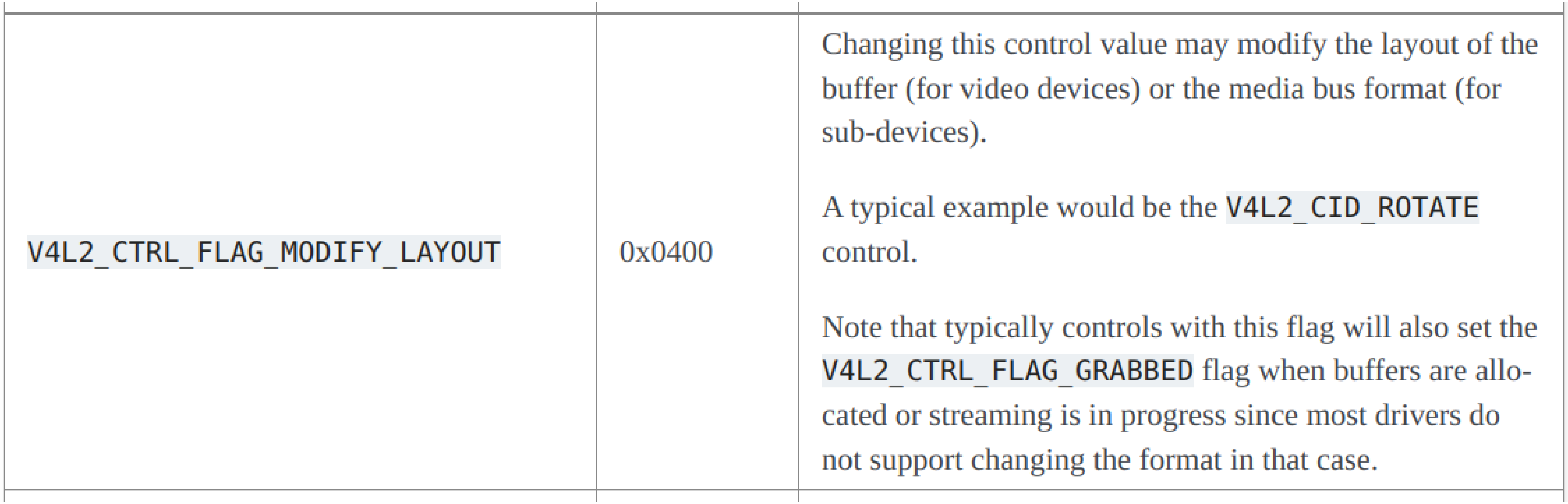
SGBRG

(0, 0)

Vertical and horizontal flips

V/H_FLIP

- V4L2 provides a control flag to signal this to userspace



`V4L2_CTRL_FLAG_MODIFY_LAYOUT`	0x0400	Changing this control value may modify the layout of the buffer (for video devices) or the media bus format (for sub-devices). A typical example would be the `V4L2_CID_ROTATE` control. Note that typically controls with this flag will also set the `V4L2_CTRL_FLAG_GRABBED` flag when buffers are allocated or streaming is in progress since most drivers do not support changing the format in that case.

Vertical and horizontal flips

Vertical and horizontal flips

V/H_FLIP

- V4L2 provides a control flag to signal this to userspace
- only 6 drivers in mainline supports it...

```
$ git grep V4L2_CTRL_FLAG_MODIFY_LAYOUT drivers/media/i2c/ | cut -f 1 | uniq | wc -l
  6
```

Camera sensor rotation

Rotation

- Expresses the camera device mounting rotation

- Device tree property *rotation*
    - *video-interface-devices.yaml*

- V4L2_CID_CAMERA_SENSOR_ROTATION

- upstreamed in 2021
- it has caused unexpected issues...

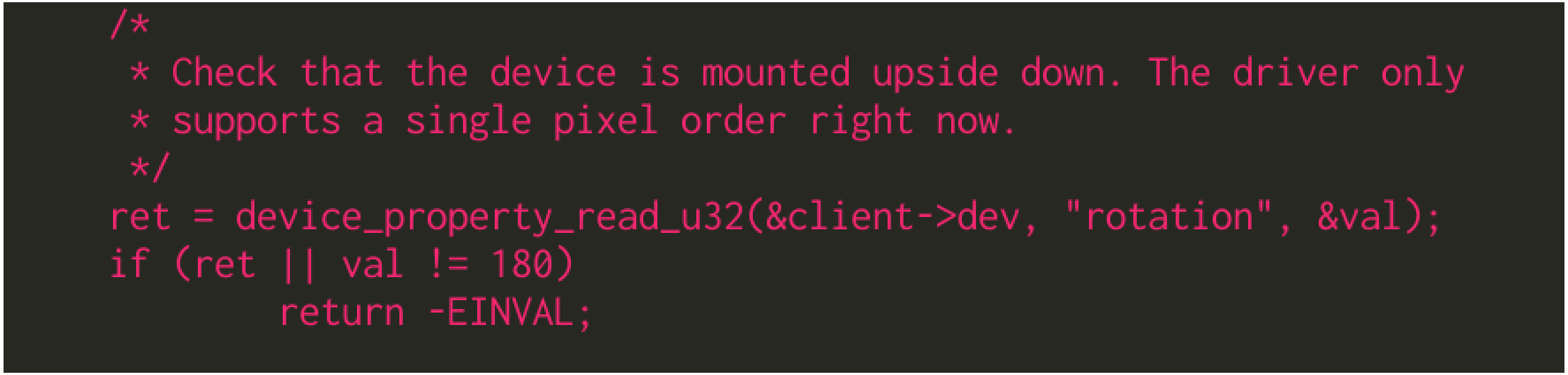
Camera sensor rotation

Rotation

- most drivers are programmed through register sequences
- those register sequences embeds v/h flips as they assume the sensor is mounted upside down to compensate for lens the inversion effect
- some drivers got confused by the default enabled flips
- some other tried to compensate for the mounting rotation by applying flips without the user noticing

Rotation

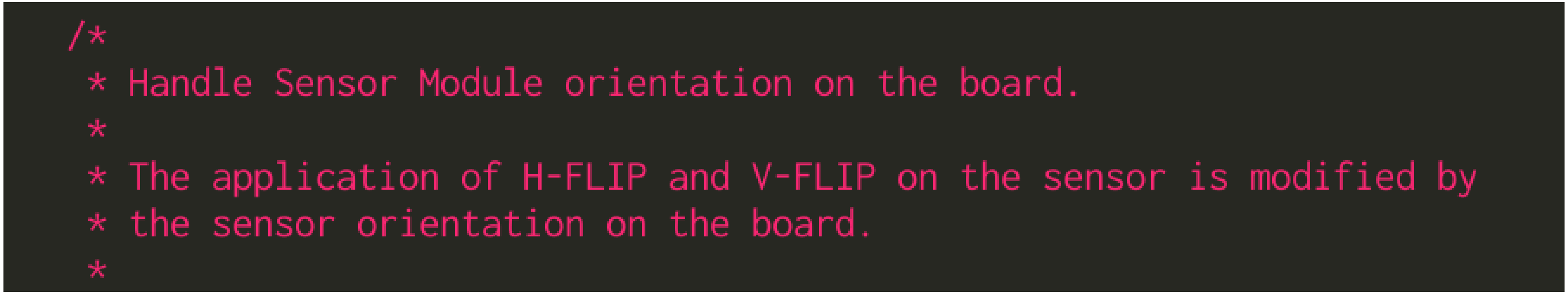
- some drivers got confused by the default enabled flips



```
/*
 * Check that the device is mounted upside down. The driver only
 * supports a single pixel order right now.
 */
ret = device_property_read_u32(&client->dev, "rotation", &val);
if (ret || val != 180)
        return -EINVAL;
```

Rotation

- some other tried to compensate for the mounting rotation by applying flips without the user noticing



```
/*
 * Handle Sensor Module orientation on the board.
 *
 * The application of H-FLIP and V-FLIP on the sensor is modified by
 * the sensor orientation on the board.
 *
```

Camera sensor rotation

Rotation

- none of the driver implementations was technically wrong
- they complied with the API specification..

- .. but their behavior was not predictable

Camera sensor rotation

Tips for drivers implementer

- Always register V4L2_CID_CAMERA_SENSOR_ROTATION with the the value associated with the *rotation* DT property

- If your driver programming sequences enable flips by default, register V4L2_CID_V/HFLIP with default value of 1

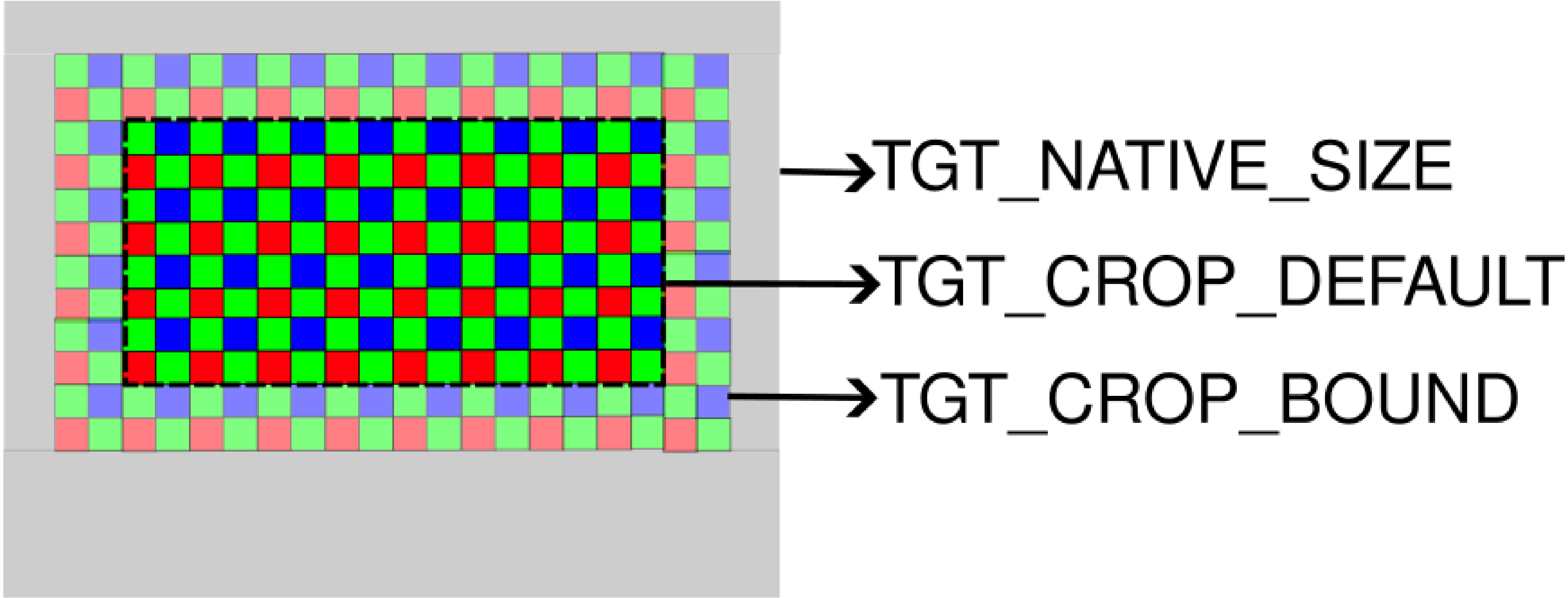
- Do not auto-compensate for rotation by silently enabling flips, let userspace deal with it!

Camera sensor rotation

Selection targets

libcamera is requesting drivers to support a few selection targets

- TGT_NATIVE_SIZE: report the full pixel array size (readable and not readable pixels)
- TGT_CROP_BOUNDS: report the readable pixel size (valid and not valid pixels)
- TGT_CROP_DEFAULT: the default analog crop
- TGT_CROP: the analog crop rectangle



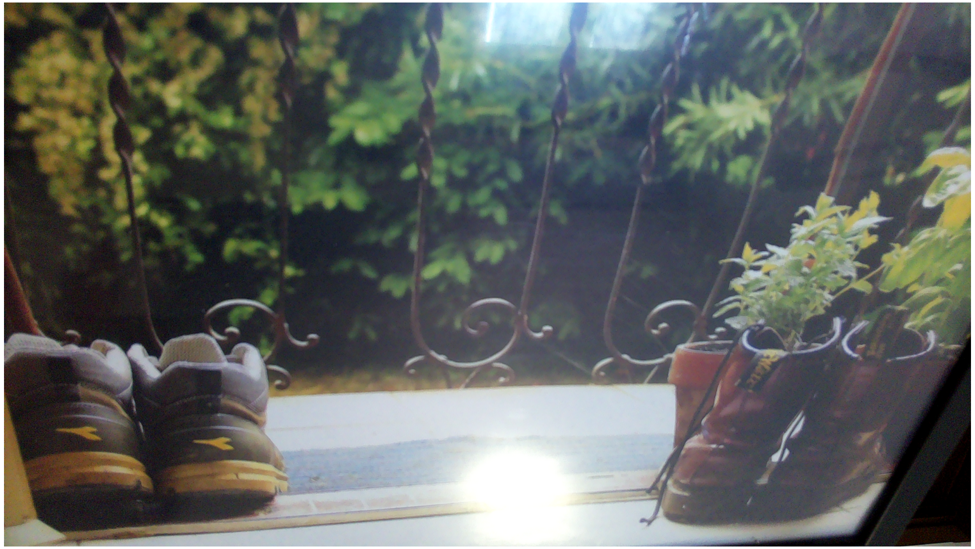
Static selection targets

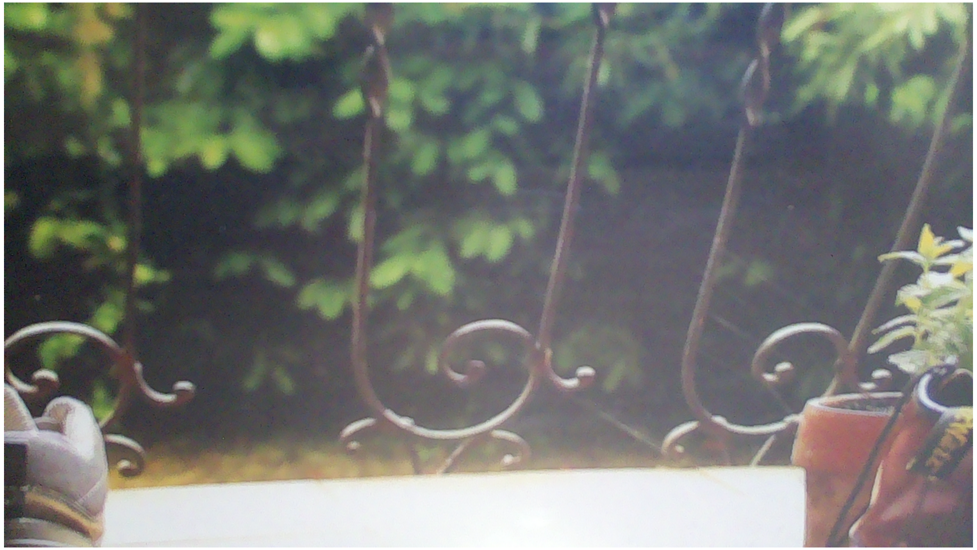
TGT_CROP: the analog crop rectangle

- the portion of the pixel array which is read out to produce the output frame
- depends on the current configuration: not a static target
- defines the image field of view
- impacts the sensor frame rate

Selection targets: analog crop

Same output resolution, different analog crop rectangle





Selection targets: analog crop

Selection targets: analog crop

TGT_CROP: the analog crop rectangle

- so far libcamera requires targets to be readable
- we’ll soon require TGT_CROP to be *writable* as well
- currently implemented by a few drivers only
- allows to dynamically change the field of view

Blankings controls

- Blankings allow to control the sensor’s frame rate
    - By enlarging or shrinking the “blank” (or inactive) time between valid image data you control the actual duration of a frame

- Horizontal blanking (*line duration)* is usually fixed
- Vertical blanking should be controllable

$$\mathit{frame\_duration} = (\mathit{width} + \mathit{hblank}) * (\mathit{height} + \mathit{vblank}) / \mathit{pixel\_rate}$$

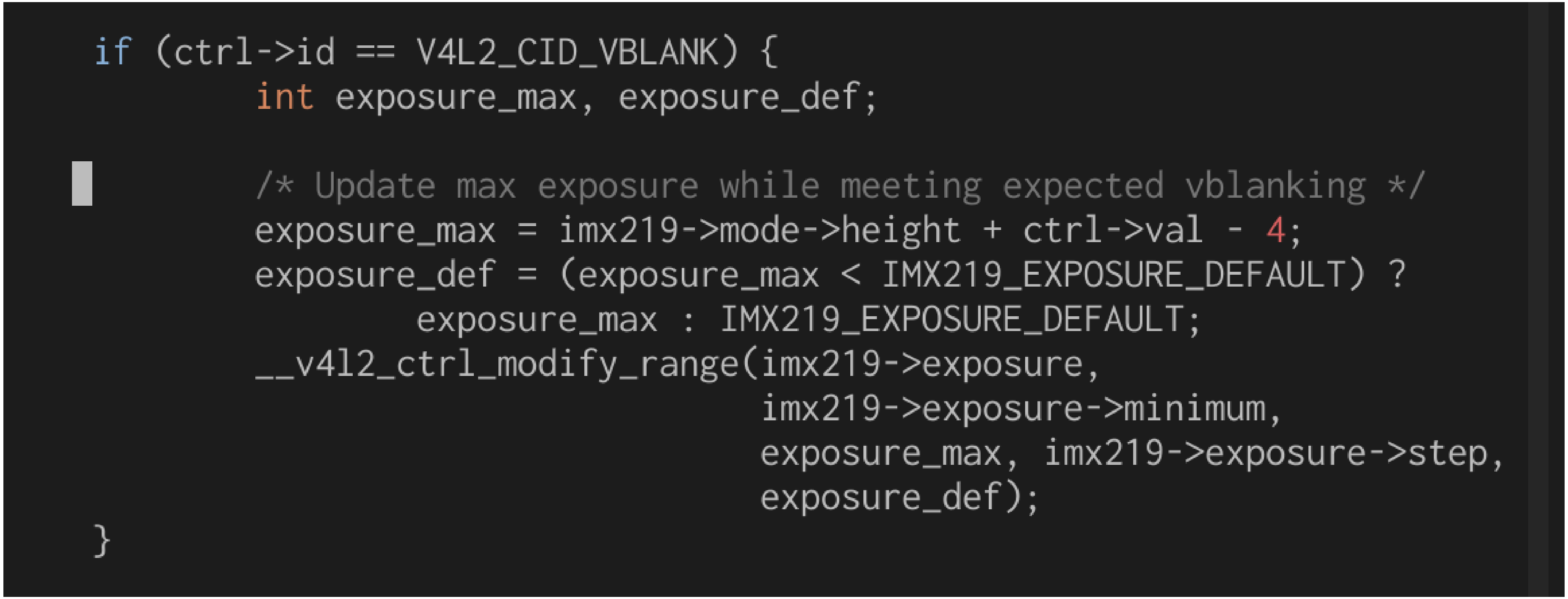
- The total frame size depends on the visible sizes as well as on the blankings size

- What happens when a new mode is applied to the sensor ?

  - some drivers resets blankings to default
  - some drivers adjust blankings only if they exceed limits

Blankings controls

You might have seen in a few places already



```
if (ctrl->id == V4L2_CID_VBLANK) {
        int exposure_max, exposure_def;

        /* Update max exposure while meeting expected vblanking */
        exposure_max = imx219->mode->height + ctrl->val - 4;
        exposure_def = (exposure_max < IMX219_EXPOSURE_DEFAULT) ?
                exposure_max : IMX219_EXPOSURE_DEFAULT;
        __v4l2_ctrl_modify_range(imx219->exposure,
                                 imx219->exposure->minimum,
                                 exposure_max, imx219->exposure->step,
                                 exposure_def);
}
```

Blankings controls

- VBLANK limits EXPOSURE
  - if you need to set both of them

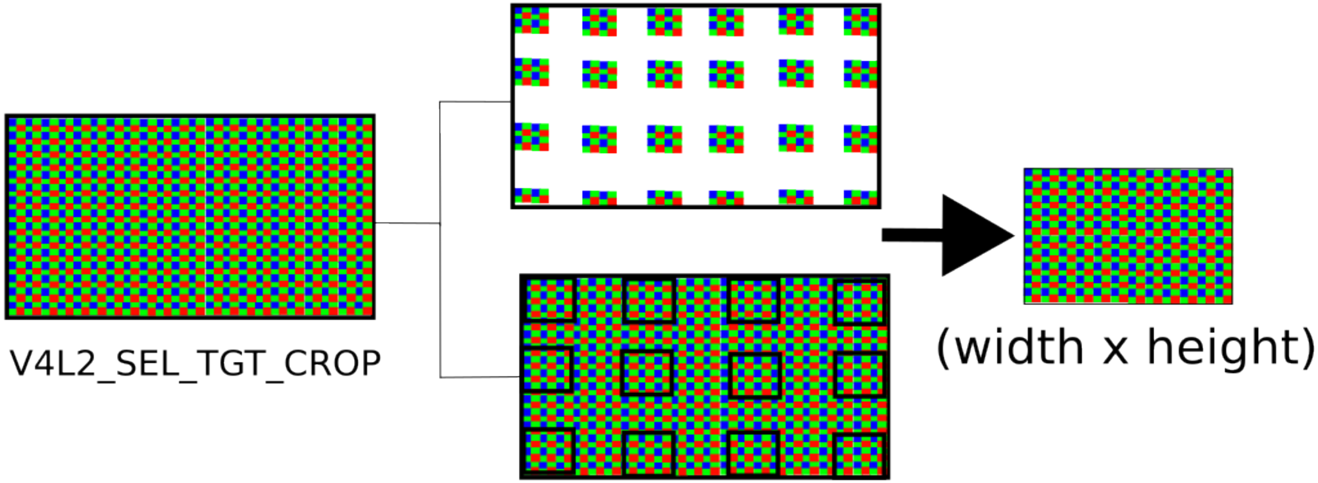
    1) Set VBLANK → driver updates EXPOSURE
    2) Set EXPOSURE

  - and you have to be careful about the order of operations

  - *[PATCH 0/2] media: uapi: Add V4L2_CID_VTOTAL control* from Benjamin Bara currently in discussion

Blankings controls

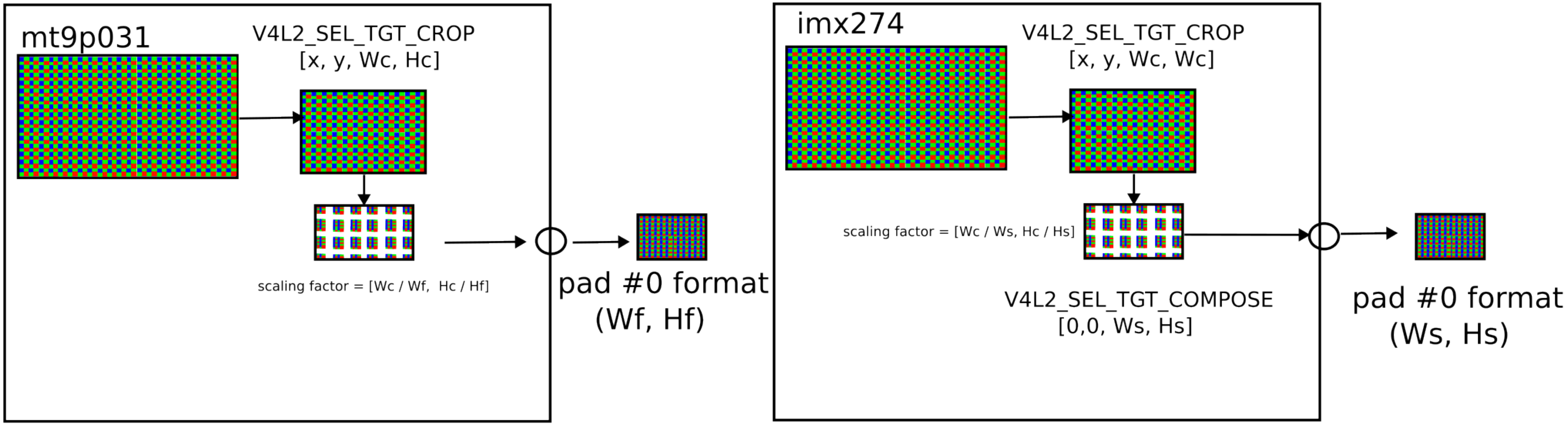
- The same output resolution can be obtain in different ways



sub-sampling

There currently is no API to know if a mode is binned or cropped

- So drivers had to find their own ways to express that



mt9p031

V4L2_SEL_TGT_CROP
[x, y, Wc, Hc]

scaling factor = [Wc / Wf, Hc / Hf]

pad #0 format
(Wf, Hf)

imx274

V4L2_SEL_TGT_CROP
[x, y, Wc, Wc]

scaling factor = [Wc / Ws, Hc / Hs]

V4L2_SEL_TGT_COMPOSE
[0,0, Ws, Hs]

pad #0 format
(Ws, Hs)

sub-sampling

*Writing applications that works generically with multiple sensor drivers is **hard***

- Abstracting away driver and device detail require a lot of effort
- Drivers might get creative when the API doesn’t help them

Conclusions

Conclusions

A standard library that abstracts away details simplifies applications development

- API compliance is not enough to guarantee interoperability
  - API might be under-specified
  - Implementation details cannot be automatically validated

- Requires a lot of review effort
- Gets easier if a reference implementation defines the expected behavior

Conclusions

A standard consumer of the kernel interfaces is the only way to validate the implementation and design of the kernel abstractions

- For a long time kernel APIs have been implemented but not exercised consistently by userspace

- A reference userspace implementation serves to validate design choices made in kernel space
- Increase consistency and completeness of kernel drivers

? !

By the way, we are hiring

jobs@ideasonboard.com