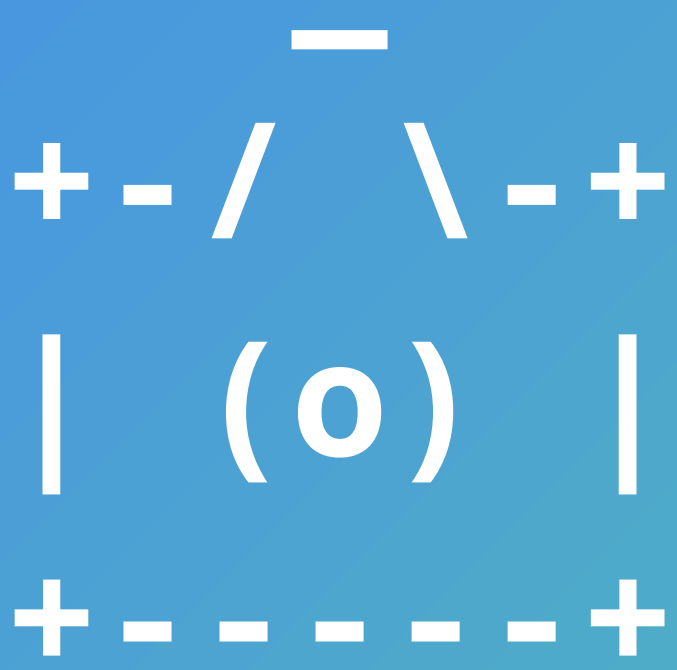


Multi-context support in V4L2

Media-summit 2024
Vienna

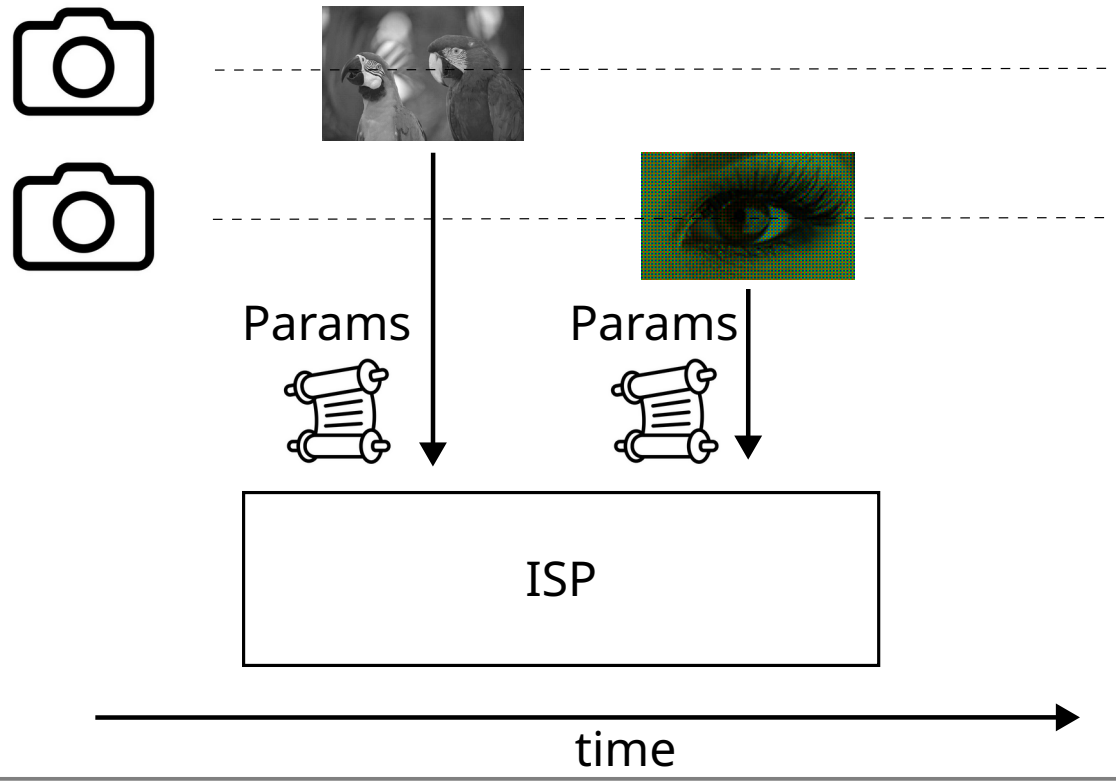
Jacopo Mondi
Laurent Pinchart

jacopo.mondi@ideasonboard.com

(Some) ISPs are time-multiplexed devices

- Resources are multiplexed at the HW or FW level and some ISP processes images in tiles
- ISPs can handle streams from different camera inputs by alternating ‘contexts’

ISP time multiplexing

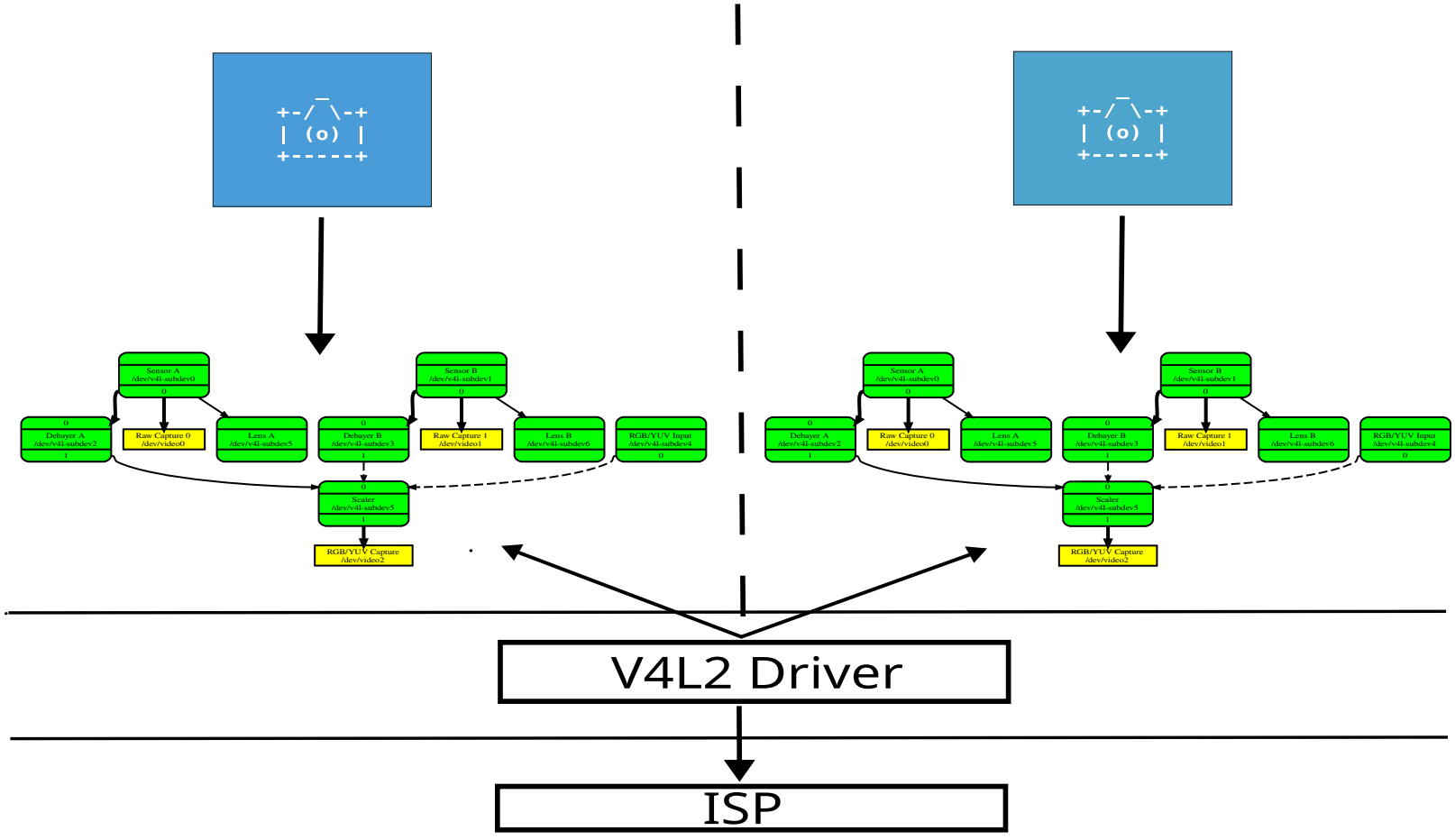


ISP time multiplexing

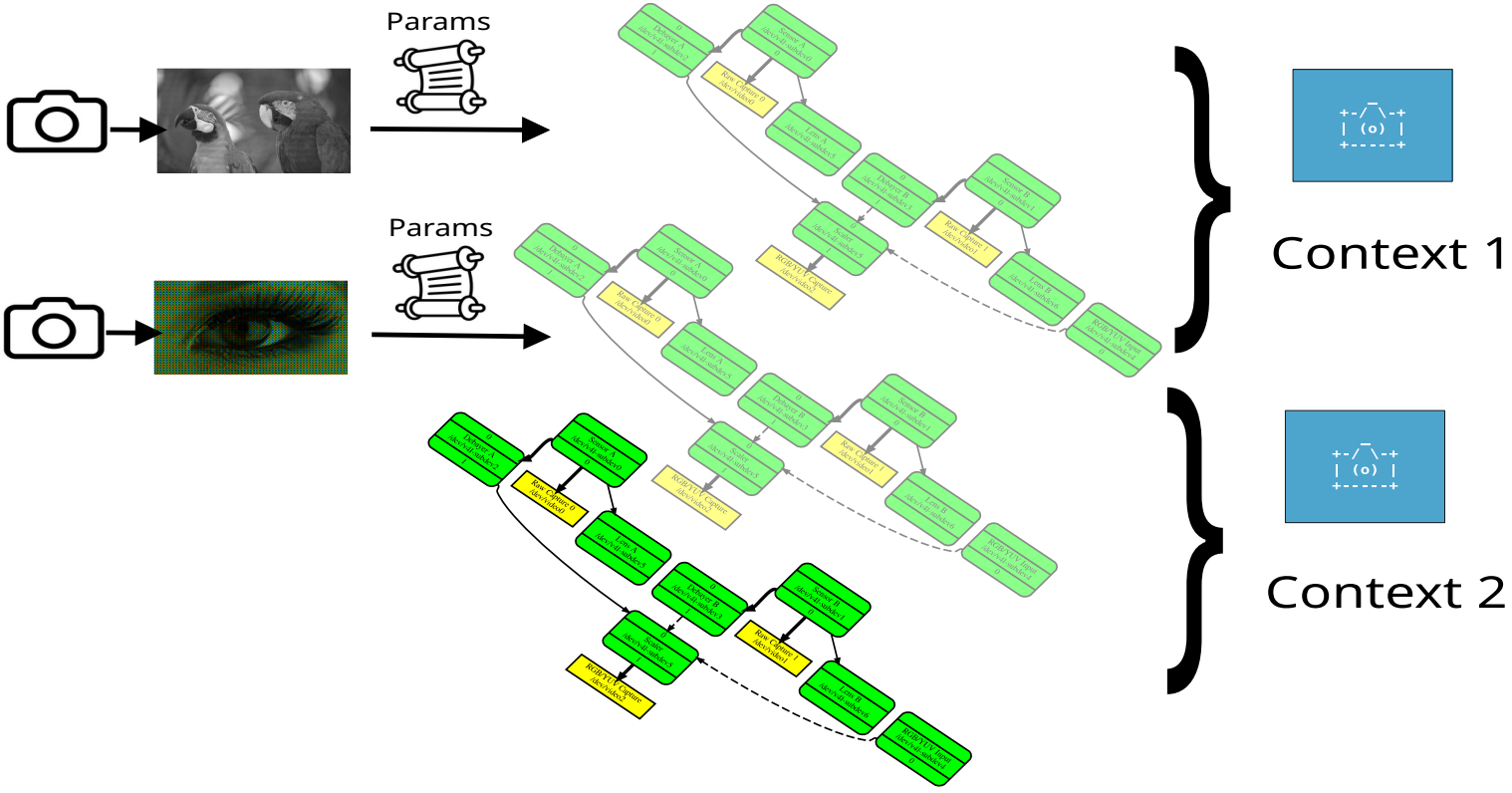
The problem has traditionally been solved by registering multiple instances of the same media graph

- One ISP instance in the system, one driver instance bound to it, multiple media graph instances (with many video device and subdevice devnodes)
- Applications are ‘tricked’ into thinking of dealing with a dedicated instance of the ISP
- Can this scale ?

Media graph multiplexing



Media graph multiplexing



Introducing contexts

Contexts:

- Execution contexts stacked on a single instance of a media graph
- Isolated at the process level
- Associates in an isolated environment:
  - Video Devices
  - V4L2 Subdevices (todo)
  - Media device links state (todo)

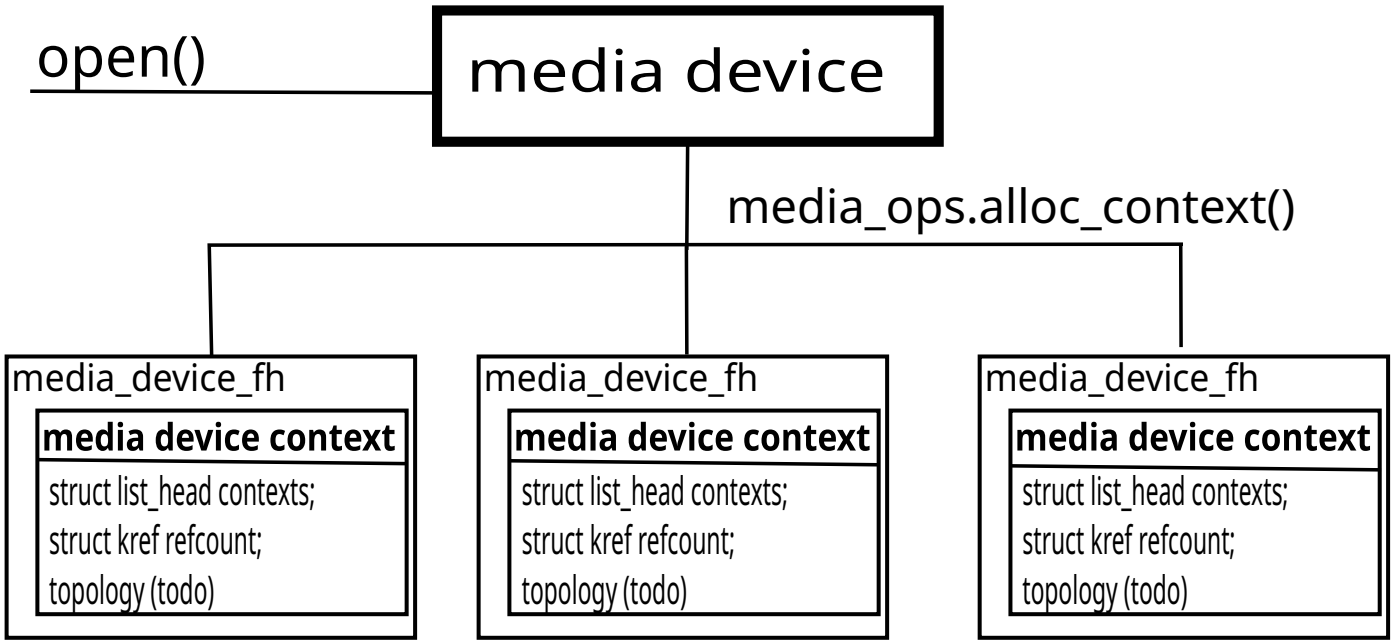
Media contexts

- **Types:**
  - Media device context (MC)

  - Media entity context (MC)
    - Video device context (V4L2)
    - V4L2 subdevice context (V4L2)

- **IOCTLs**
  - VIDIOC_BIND_CONTEXT
  - VIDIOC_SUBDEV_BIND_CONTEXT (todo)

Media contexts: key design elements



Media device context

Media device context

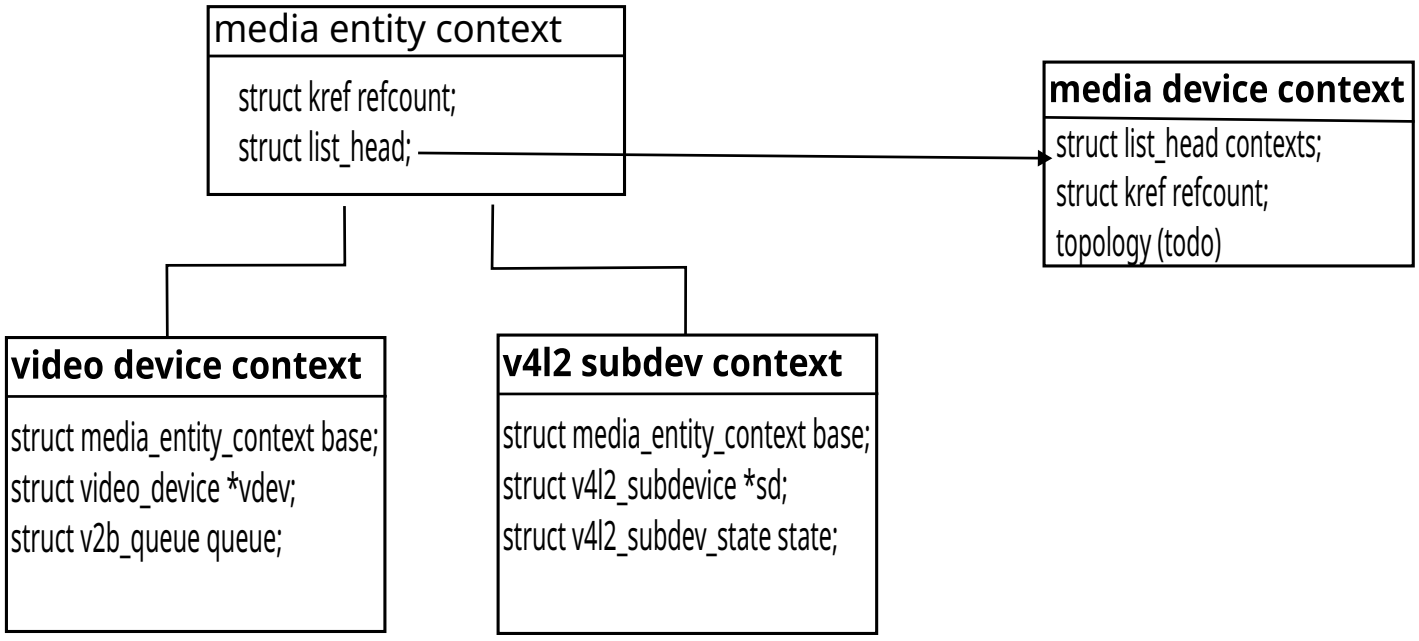
- Created at media device open time
- Refcounted
- Referenced by media-fh

- Stores a list of *struct media_entity_contexts*

- Drivers can extend it

Drivers can extend the media_device_context

```
struct media_device_ops {
        int (*link_notify)(struct media_link *link, u32 flags,
                           unsigned int notification);
        struct media_request *(*req_alloc)(struct media_device *mdev);
        void (*req_free)(struct media_request *req);
        int (*req_validate)(struct media_request *req);
        void (*req_queue)(struct media_request *req);
        void (*release)(struct media_device *mdev);
        int (*alloc_context)(struct media_device *mdev,
                             struct media_device_context **ctx);
        void (*destroy_context)(struct media_device_context *ctx);
};
```



IDEAS ON BOARD

Media entity context

Media entity context
- Refcounted
- Linked in the contexts list of *struct media_device_context*

Video device context
- Referenced by *struct v4l2-fh*
- Extends *media entity context*
- Stores *struct vb2_queue*

V4L2 subdevice context (todo)
- Referenced from *struct v4l2-subdev-fh*
- Extends *media entity context*
- Stores v4l2_subdev_state

Media entity context

Drivers can extend the media entity context

```
struct media_entity_operations {
        int (*get_fwnode_pad)(struct media_entity *entity,
                              struct fwnode_endpoint *endpoint);
        int (*link_setup)(struct media_entity *entity,
                          const struct media_pad *local,
                          const struct media_pad *remote, u32 flags);
        int (*link_validate)(struct media_link *link);
        bool (*has_pad_interdep)(struct media_entity *entity, unsigned int pad0,
                                 unsigned int pad1);
        int (*alloc_context)(struct media_entity *entity,
                             struct media_entity_context **context);
        void (*destroy_context)(struct media_entity_context *context);
};
```

VIDIOC_BIND_CONTEXT(media_fd)

- Create a *video_device_context*
- Add it to the list of contexts in *media_device_context*
- Stores a reference to in the *struct v4l2-fh*

VIDIOC_BIND_CONTEXT(media_fd)

```
media_fd = open("/dev/mediaX", ...);
video_fd = open("/dev/videoX", ...);

struct video_device_context c = {
        .context_fd = media_fd;
};

ioctl(video_fd, VIDIOC_BIND_CONTEXT, &c);

/*
 * Operate the video device as usual.
 * 'video_fd' is bound to a specific context.
 */
ioctl(video_fd, VIDIOC_..., ...);
```

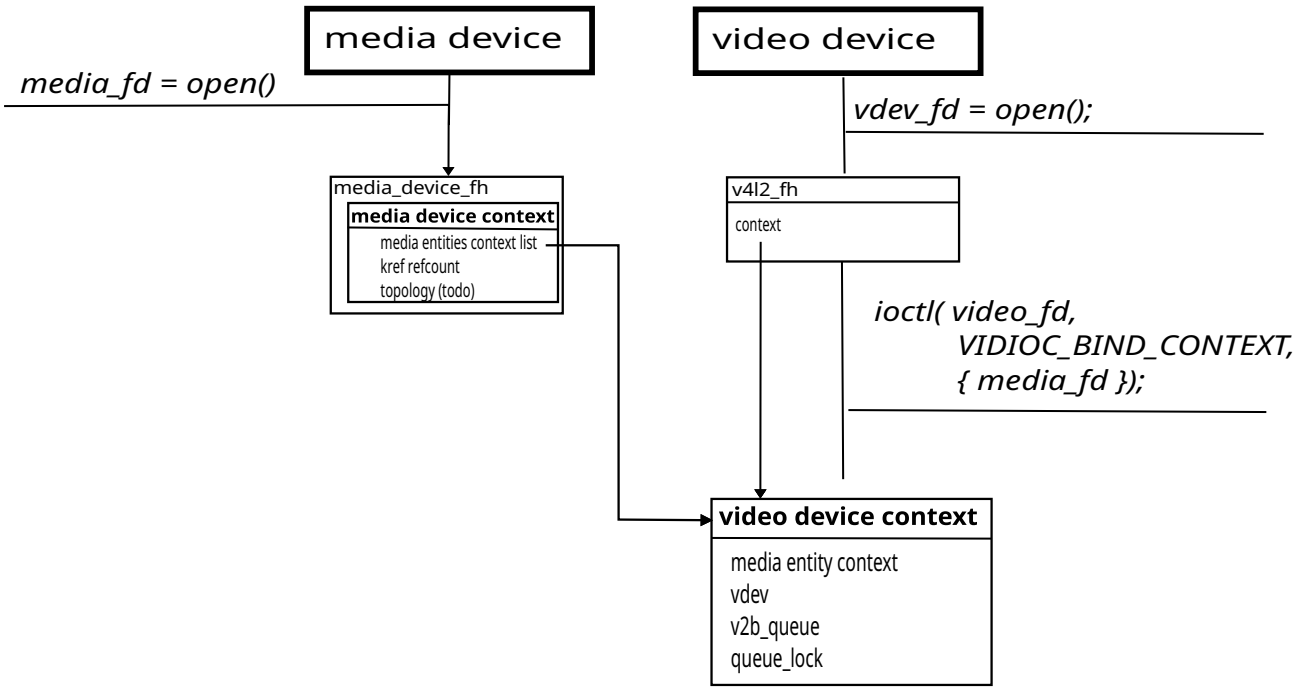
VIDIOC_BIND_CONTEXT(media_fd)

```
/* Create a different context. */
media_fd2 = open("/dev/mediaX", ...);
video_fd2 = open("/dev/videoX", ...);

struct video_device_context c2 = {
        .context_fd = media_fd2;
};

ioctl(video_fd2, VIDIOC_BIND_CONTEXT, &c2);

/*
 * 'video_fd' and 'video_fd2' operates on different
 * contexts.
 */
ioctl(video_fd, VIDIOC_..., ...);
ioctl(video_fd2, VIDIOC_..., ...);
```



Context binding

Video device context: multiplexing a video device

- Move the **vb2_queue** from *struct video_device* to *struct video_device_context*

- The format and the configuration of the video device is moved to the driver-specific video device context

- Drivers extend *struct video_device_context* to store the context-specific information

videobuf2: get the vb2_queue from the context

```
/*
 * Helper to get the vb2 queue either from:
 * 1) The video context bound to the open file handle
 * 2) The default context for context-aware drivers if userspace has not bound
 *    a context to the file handle
 * 3) From the video device for non-context aware drivers
 */
static struct vb2_queue *get_vb2_queue(struct file *file,
                                       struct video_device *vdev)
{
        struct video_device_context *ctx =
                                video_device_context_from_file(file, vdev);

        return ctx ? &ctx->queue
                   : vdev->default_context ? &vdev->default_context->queue
                   : vdev->queue;
}
```

videobuf2: get the vb2_queue from the context

```
/* vb2 ioctl helpers */

int vb2_ioctl_reqbufs(struct file *file, void *priv,
                          struct v4l2_requestbuffers *p)
{
        struct video_device *vdev = video_devdata(file);
-       int res = vb2_verify_memory_type(vdev->queue, p->memory, p->type);
+       struct vb2_queue *q = get_vb2_queue(file, vdev);
+       int res = vb2_verify_memory_type(q, p->memory, p->type);
        u32 flags = p->flags;

-       fill_buf_caps(vdev->queue, &p->capabilities);
-       validate_memory_flags(vdev->queue, p->memory, &flags);
+       fill_buf_caps(q, &p->capabilities);
+       validate_memory_flags(q, p->memory, &flags);
```

vb2_ops callbacks

- receive a *vb2_queue* as argument
- get context from the queue (*container_of)*

v4l2-ioctl-ops callbacks

- receive a *file* * as argument
- get context from the open file handle

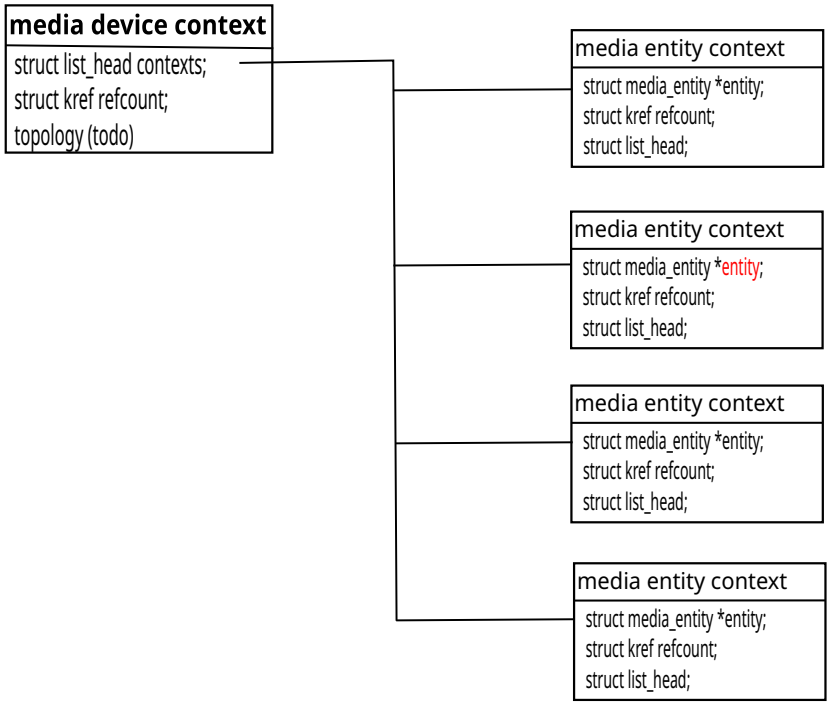
Internal driver ops

- get contexts for the media entities registered by the driver associated to the same *media device context*

video_device_context_get()

- Drivers register multiple video devices and subdevices
- They need to get contexts associated to the same *struct media_device_context **
- *video_device_context_get(mdev_context, vdev)*
- *v4l2_subdev_context_get(mdev_context, sd) (todo)*
  - Wrappers around *media_device_get_entity_context()*
  - Walk the list of contexts associated in a media device context and find the one with a matching *struct media_entity **

```
struct video_device_context *video_device_context_get(mdev, vdev)
{
        return containre_of(media_device_get_entity_context(mdev,
                                                            &vdev->entity),
                            struct video_device_context, base);
}
```



Drivers API: video_device_context_get()

Default contexts:

- Allows context-aware drivers work with context-aware and non-context aware userspace

Opt-in:

- Context support is totally opt-in: drivers that do not support multiplexing does not modified

Additional notes

Per-context media topology

- Move links state to *struct media_device_context*

Subdevs

- Same design as video devices
  - Store reference in *stuct subdev-fh*
  - Associate with a *struct media_device_context*

- Reuse *struct subdev_state* as much as possible

todos

How to make sure userspace does not mix-up contexts ?

V4l2 Controls

- I’m no expert there and I don’t know if this is possible or even desirable

m2m context

- Should the two be unified ?

Open questions