Embedded Linux Conference – EU 2018

# Complex Cameras on Linux

**Mauro Carvalho Chehab**

Shape the Future with Innovation and 1ntelligence

Oct, 23 2018

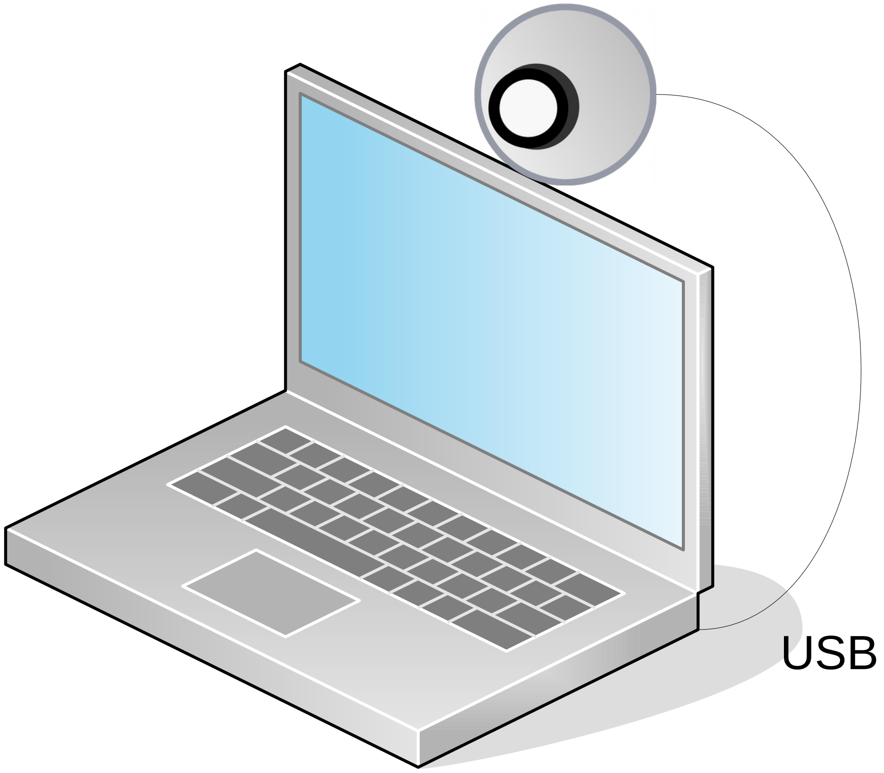
# Contents

# What is a complex camera?

# What’s a Complex Camera?

- The Linux Kernel media subsystem provide support for two types of camera:
  - “traditional” media hardware
    - supported via “standard” V4L2 API
  - Complex cameras
    - Require 3 API sets to control (V4L2, media controller, subdev API)

# Traditional cameras



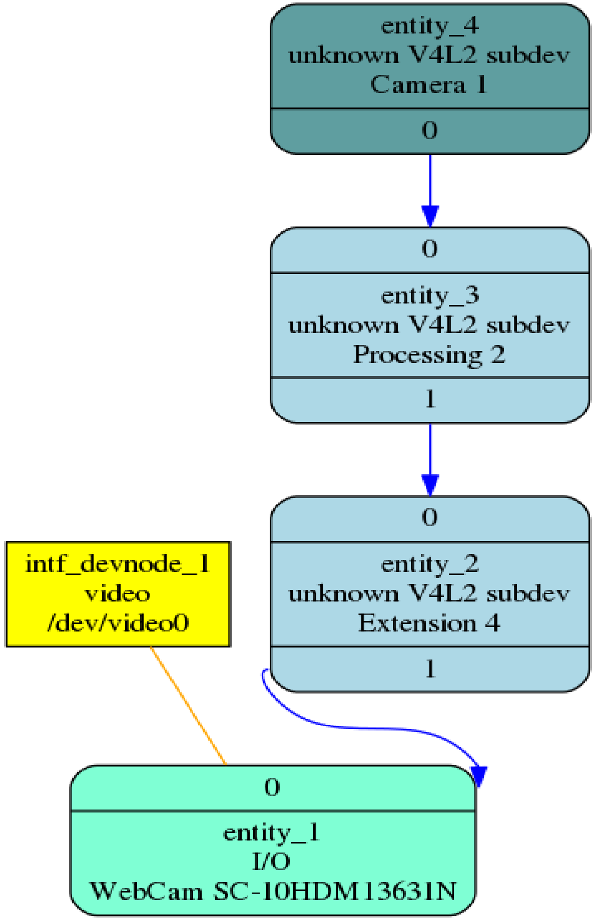
# Traditional cameras

- A single device node (like */dev/video0*) is enough  to control the entire device;
- Multiple device nodes could be opened (like *video0*, *vbi0*, *radio0*), and even multiple capture nodes, on devices with support delivering streams in parallel;
- They may eventually expose themselves via the media controller API;
- Generic apps can easily support them, as all hardware-specific details are abstracted by the Kernel;
- Yet, some hardware provide proprietary formats
- Internally, the camera hardware usually has an image signal processor or use some hardware tricks, in order to provide a images that are good enough for consumers;
- We call it **devnode-based** devices.

# Example: Samsung chromebook snow (USB UVC)

In yellow: device nodes
Visible to userspace

1 devnode is enough
to fully control the
hardware
(/dev/video0)



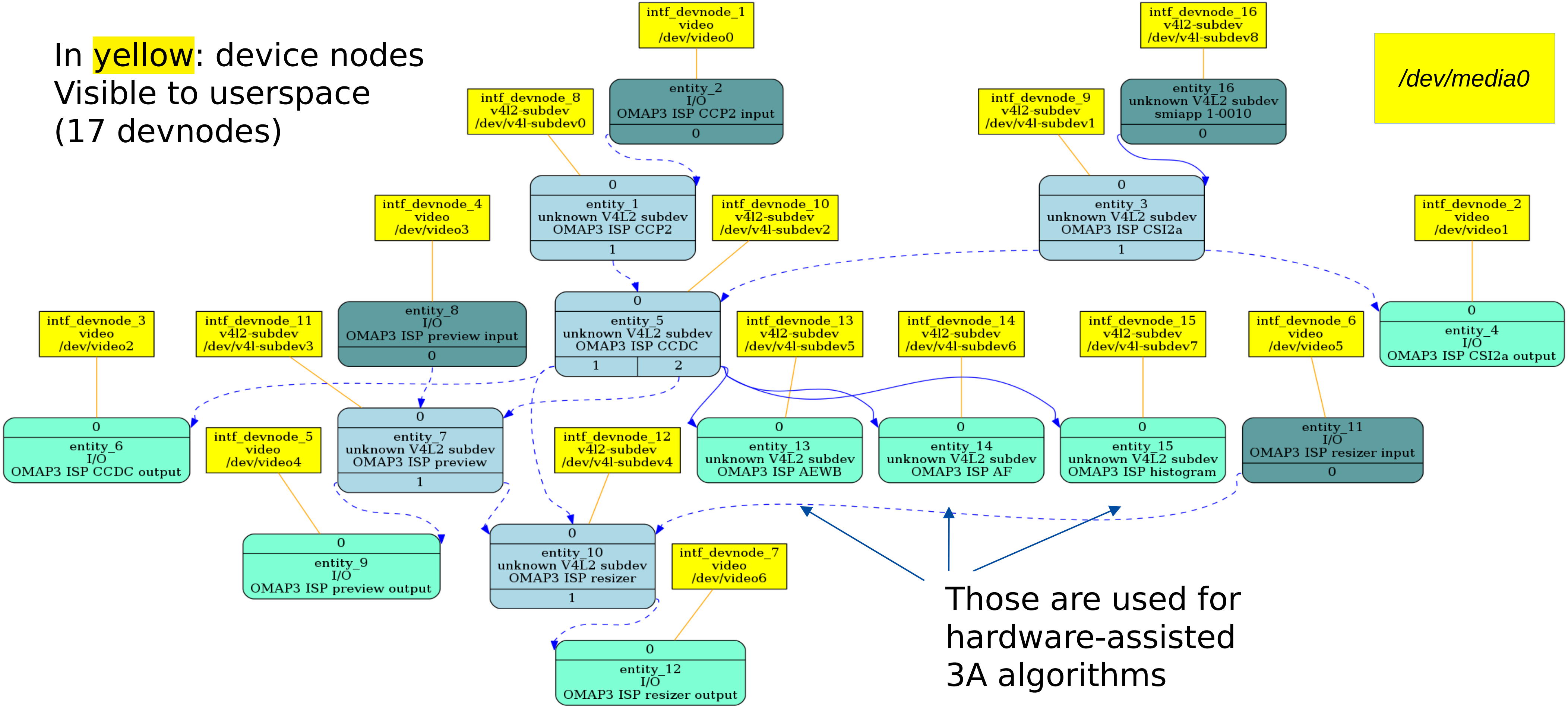

/dev/media0

# Complex Cameras

- “complex” media hardware supported by V4L2, Media controller (MC) and V4L2 subdev APIs
- There are typically multiple video capture devices (*/dev/video?*) and multiple sub-devices exposed (*/dev/v4l2-subvev?*), plus the media controller device (*/dev/media0*);
- Controlling the hardware require opening the media controller, setup the pipeline and adjust the sub-devices accordingly (by opening their /dev/v4l2-subdev?) devnodes;
- Only streaming is controlled by */dev/video?*;
- Typically, assumes that the CPU chipset (or SoC) has an Image Signal Processor (ISP) to enhance image, running 3A algorithms (auto-expose, auto-whitebalance and autofocus), scaling, format conversion, etc.
- Sometimes, the ISP is controlled via a different driver/pipeline;
- The 3A algorithm usually requires a binary only runtime at CPU;
- Applications need to know details about the hardware.
- We call it **MC-based** devices.

# Example of a complex camera: omap3isp

In yellow: device nodes
Visible to userspace
(17 devnodes)




Those are used for
hardware-assisted
3A algorithms

# The V4L2 library: libv4l – How it started

- Was designed after we added the gspca drivers
  - hundreds of different USB cameras using USB Device Vendor Class
    - No standard: each USB bridge vendor chooses how to control the camera
  - Most drivers were developed based on reverse engineering
  - Hardware for those cameras are simple, and several are USB 1.1
    - The camera bridge usually has a proprietary format, in order to reduce the USB traffic, due to bus constraints
- Most modern cameras now use USB Video Class (UVC)
  - Spec provides a standard way to control them

# The V4L2 library: libv4l – Goals

- Libv4l was designed to achieve those main goals:
  - Allow all apps to work with any camera, including closed source ones
  - Bridge format specific conversions should be part of the library
  - Provide simple fast software algorithms to adjust exposure and enhance image quality when apps are used with simpler camera hardware;
  - Adjust image, if sensors are mounted upside down
  - Provide compatibility with the old V4L1 API, with was removed from Kernel
  - Most open source apps use libv4l by default.

# Libv4l

# Summary of libv4l

- Consists of 3 sets of libraries:
  - Image processing: **libv4lconvert**
  - V4L1 compatibility: **libv4l1**/ **v4l1compat**
  - V4L2 library: **libv4l2** / **v4l2convert**

# Image processing (libv4lconvert)

- Contains format conversion routines:
  - Needed to support non-UVC cameras
    - Also works with UVC cameras
- Main focus:
  - Compressed formats decoding (like mjpeg)
  - Proprietary formats produced by the USB bridge (or by the sensor);
- Goal:
  - Apps just need to be able to handle a few video formats, like RGB 24 bits
  - A camera app that only supports RGB-24 can work with any traditional Linux Camera

# Image processing (libv4lconvert)

- Format Conversions:
  - Conversion for camera bridges: sn9c10x, sn9c20x, etc;
  - Other conversions: RGB, YUV, Bayer, mjpeg, jpeg and jpeg lite;
  - There's a BZ asking for MPEG, in order to support some MPEG-only capture devs
  - Support for Conexant HM12 format (TV and capture boards);
  - Formats emulated are tagged with **V4L2_FMT_FLAG_EMULATED**.
- Has support for flipping images and for RGB/YUV cropping;
  - Has processing algorithms: gamma control, auto-WB, and autogain
- A developer has a generic algorithm for auto-focus – pending submission
- Has a database of cameras that require special userspace hacks to work
  - E. g.: cameras mounted upside/down, the ones that don't need auto-WB; the ones that require some processing (like auto-WB) per default.

# V4L1 compatibility (libv4l1 and v4l1compat)

- Compatibility with old V4L version 1 API
  - Got deprecated a long time ago
  - Still, camorama used to depend on it
    - I started co-maintaining it and got rid of V4L1 API a couple months ago
      - Will still take some time to propagate to distros
        - Fedora 28 has the new version already
  - Maybe some other apps still rely on it
- Supporting V4L2 was important during conversion
  - Kernel support for V4L1 was moved to userspace

# V4L1 compatibility (libv4l1 and v4l1compat)

- Libv4l1 provides an emulation of the V4L version 1 API, talking to the Kernel using V4L2 version 2 API
  - This was required when we removed V4L1 backward-compatibility layer upstream, in order to not break userspace;
- Function are defined as v4l1_*func,* where *func* is *open*, *open64*, *close*, *dup*, *ioctl*, *read*, *mmap*, *mmap64* and *munmap*.
- Most known V4L1 userspace open source apps were converted to V4L2 a long time ago.
- v4l1-convert provides hooks for using standard open, open64, close, dup, ioctl, read, mmap, mmap64 and munmap via **LD_PRELOAD**, in order to allow running a V4L1 binary only application.

# V4L2 library: libv4l2 and v4l2convert

- API consists of v4l2_*func*()
  - where *func* is *open*, *close*, *mmap*, *munmap*, *ioctl*, *read,* …
- Calls libv4lconvert
  - have other features like autogain, auto-white balance, …
- Goal is to make apps independent of V4L2 features.
  - Has a *quirks* database, solving issues like sensors mounted upside down and exposing some software-based camera controls
- Used for all sort of V4L2 generic apps:
  - TV
  - Video stream capture
  - camera

# V4L2 library: libv4l2 and v4l2convert

- Libv4l2 provides a set of v4l2_*func*, where *func* is *open*, *close*, *dup*, *ioctl*, *read*, *write*, *mmap*, *munmap*, *set_control*, *get_control* and *fd_open*.
  - Converting an application to use it should be as simple as renaming the function calls
  - That limited the API, as all syscalls-like functions should remain using the same parameters as at the libc with the same syntax.
  - It was meant to support all V4L2 ioctls
  - Yet, new ioctls were added without adding their counterparts at library
    - libv4l2 author stopped working with cameras sometime ago, and nobody took his place keep developing the library;
- Right now, we apply patches only when bugs are reported or someone sends us contributions.
- v4l2-convert provides hooks for using standard glibc function calls via **LD_PRELOAD**, in order to run a V4L2 binary only application.

# Main problems with current libv4l approach

- Maintenance:
  - When video buffering code gets new features, those need to be reflected at the library;
- Performance:
  - Several applications don’t check if the format is software-emulated or not
  - Easy to fix at apps, but developers doesn’t seem to care;
  - If we change the API to better identify emulation, I suspect most apps won’t use
- Devnode-based devices:
  - There’s no support for MC-based devices
  - There is a patchset adding support for OMAP3 hardware.
  - It is somewhat hackish that a v4l2_open() would open MC + subdev + video devnodes at the same time.
  - How to handle partial failures while opening many device nodes?

# Gstreamer and libv4l

- Right now, gstreamer defaults to not enable libv4l, due to several reasons:
  - It crashes when CREATE_BUFS is being used (as it lacks support for it);
  - It crashes in the jpeg decoder, when frames are corrupted;
  - Apps exporting DMABuf need to be aware of emulation, otherwise the DMABuf exported are in the original format;
  - RW emulation only initializes the queue on first read, causing poll() to fail;
  - Signature of v4l2_mmap does not match mmap() (minor issue);
  - The colorimetry does not seem emulated when using libv4l format conversion;
  - Sub-optimal locking.
  - Gstreamer has workarounds for those, but with the lost of features.
- Most of these problems are due to the lack of an active maintainer for libv4l.
- Since 1.14, libv4l2 can be enabled in run-time using **GST_V4L2_USE_LIBV4L2=1**.

# Modern hardware on customer's devices

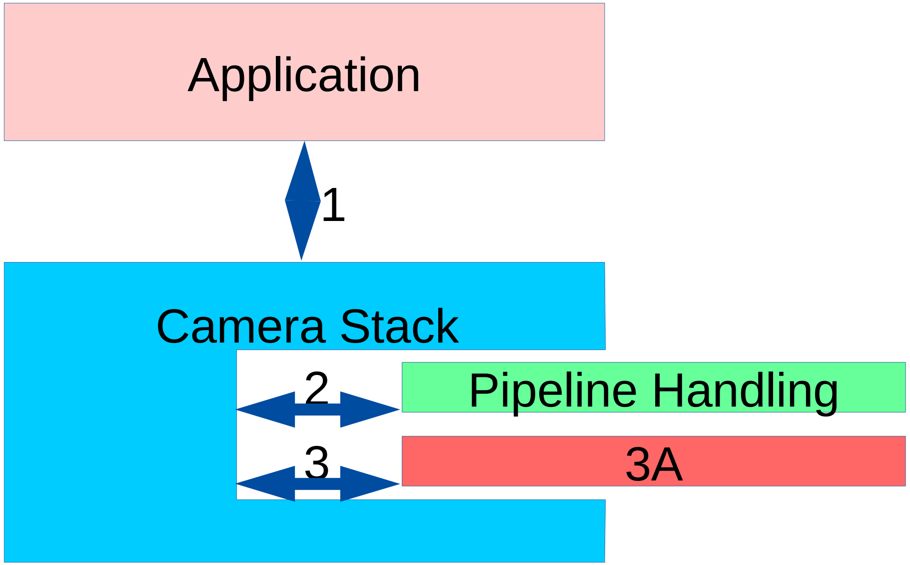
# Modern hardware and complex cameras

- The complex cameras are there since 2008 (Nokia N9/N900). So, why we're urging to solve this issue nowadays?
  - Because modern hardware used for laptops and PCs started to come with integrated ISPs;
  - It started with Intel Atom: atomisp driver
  - Now, modern Intel mobile CPU chipsets are coming with IPU3
  - Dell has notebook models with IPU3 chipset, whose cameras don't work with generic apps (like Dell Latitude 5285)
- Because we want a single solution that works with:
  - Standard PCs/notebooks running standard linux distros and Tizen;
  - Android HAL
  - ChromeOS HAL

# How to solve it?

# How to Solve it?

- As agreed at the Complex Camera Workshop, in Japan (Jun, 18):
  - Develop a Camera Stack capable of supporting all V4L2 hardware
    - Inspired on Android HALv3, but addressing some problems on it.




Should have 3 APIs:

1) Between Application and Camera Stack

2) Between Camera stack and Pipeline handler

3) Between Camera stack and 3A algorithms

Vendor specific
Framework
Framework + Vendor specific if needed

# What’s next?

- See the speech:
  *Why Embedded Cameras are Difficult, and How to Make Them Easy*
  - Wed, Oct 24 – 16:15
- Libcamera.org – Hot site for the new development
  - https://git.linuxtv.org/libcamera.git/
  - Should be available tomorrow
- More discussions will happen at Linux Media Summit on Thrusday

# Thank you