GStreamer and dmabuf

OMAP4+ graphics/multimedia update

Rob Clark

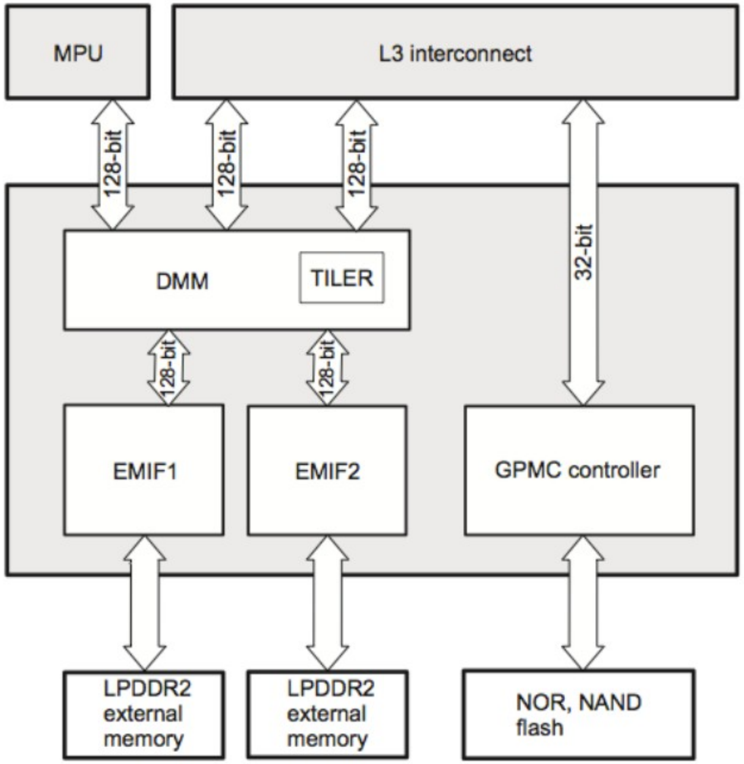
Outline

- A quick hardware overview
- Kernel infrastructure: drm/gem, rpmsg+dce, dmabuf
- Blinky s***.. putting pixels on the screen
- Bringing it all together in GStreamer

A quick hardware overview

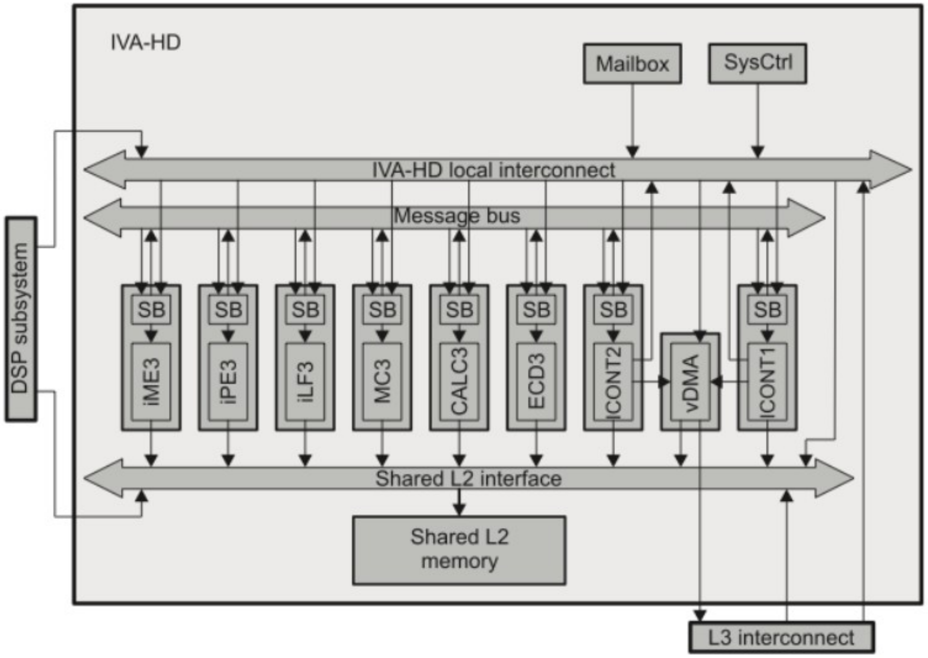
DMM/Tiler

- Like a system-wide GART
  - Provides a contiguous view of memory to various hw accelerators: IVAHD, ISS, DSS
- Provides tiling modes for enhanced memory bandwidth efficiency
  - For initiators like IVAHD which access memory in 2D block patterns
- Provides support for rotation
  - Zero cost rotation for DSS/ISS access in 0º/90º/180º/270º orientations (with horizontal or vertical reflection)



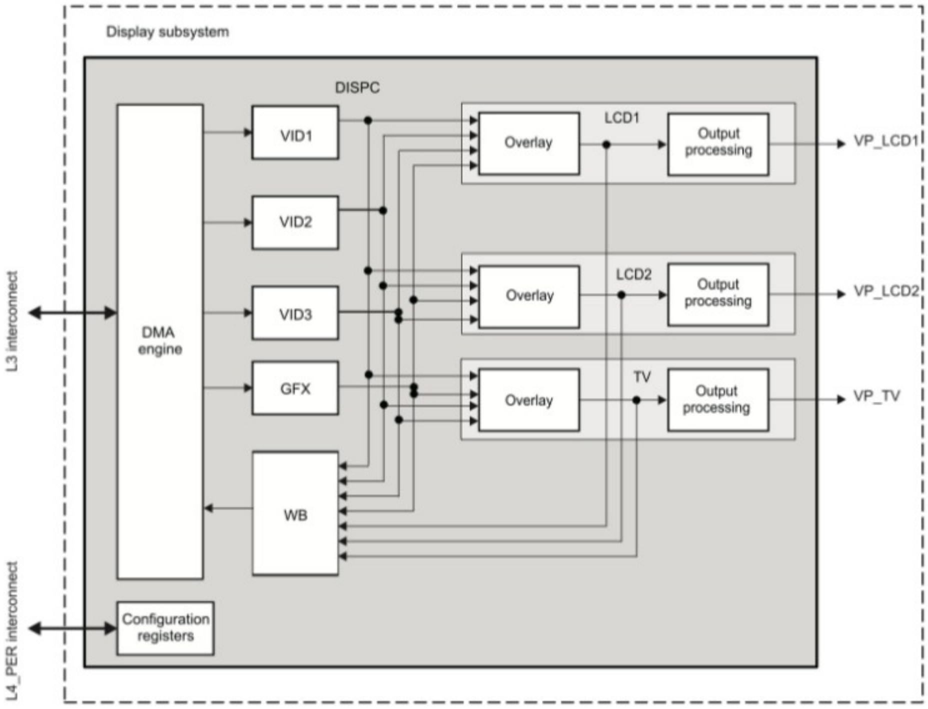
IVA-HD

- Multi-codec hw video encode/decode
  - H.264 BP/MP/HP encode/decode
  - MPEG-4 SP/ASP encode/decode
  - MPEG-2 SP/MP encode/decode
  - MJPEG encode/decode
  - VC1/WMV9 decode
  - etc



DSS – Display Subsystem

- Display Subsystem
  - 4 video pipes, 3 support scaling and YUV
  - Any number of video pipes can be attached to one of 3 “overlay manager” to route to a display



Kernel infrastructure:

drm/gem, rpmsg+dce, dmabuf

DRM Overview

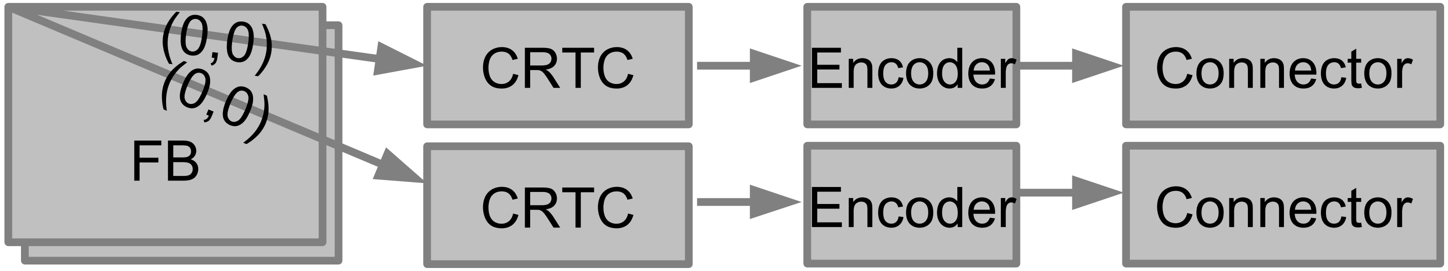
- DRM → Direct Rendering Manager
  - Started life heavily based on x86/desktop graphics card architecture
  - But more recently has evolved to better support ARM and other SoC platforms
- KMS → Kernel Mode Setting
  - Replaces fbdev for more advanced display management
  - Hotplug, multiple display support (spanning/cloning)
  - And more recently support for overlays (planes)
- GEM → Graphics Execution Manager
  - But the important/useful part here is the graphics/multimedia buffer management

DRM - KMS

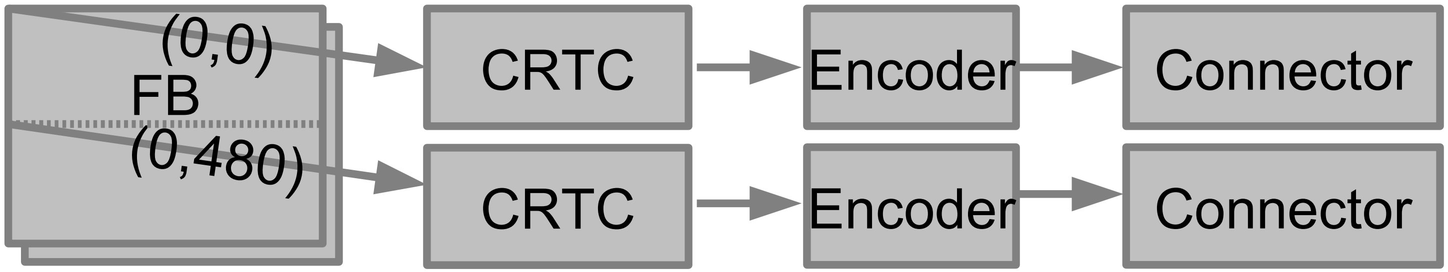
- Models the display hardware as:
  - Connector → the thing that the display connects to
    - Handles DDC/EDID, hotplug detection
  - Encoder → takes pixel data from CRTC and encodes it to a format suitable for connectors
    - ie. HDMI, DSI, DPI
  - CRTC → takes the DMA engine that scans out the framebuffer
  - Plane → an overlay
  - Framebuffer → just a piece of memory
    - A GEM object plus attribute: fourcc, width, height, pitch
- See: http://www.ideasonboard.org/media/drm/index.html

KMS - Multi-display

- Clone Mode



- Virtual Display

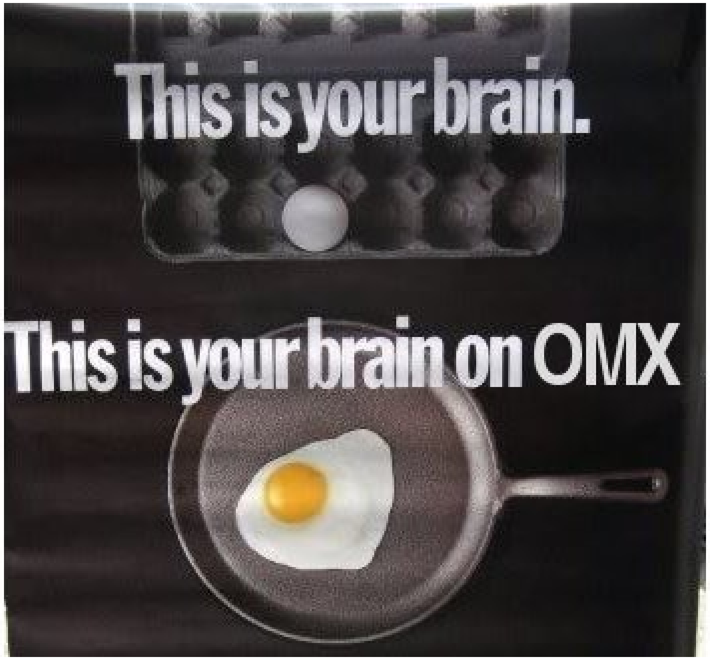


omapdrm

- DRM driver for OMAP platforms
- Supports the KMS API for multi-display, hotplug, etc
- Supports GEM buffers
  - Can be dynamically mapped to DMM on demand, for example when passing a buffer to hw decoder, or scanning out a fb
  - Handles mmap of cached buffers
    - Page faulting + PTE shootdown for tracking dirty pages
  - Handles mmap of 2D tiled buffers
    - Usergart + page faulting + PTE shootdown for giving userspace 4KiB aligned view of 2D tiled buffers at potentially odd alignments

DCE – Distributed Codec Engine

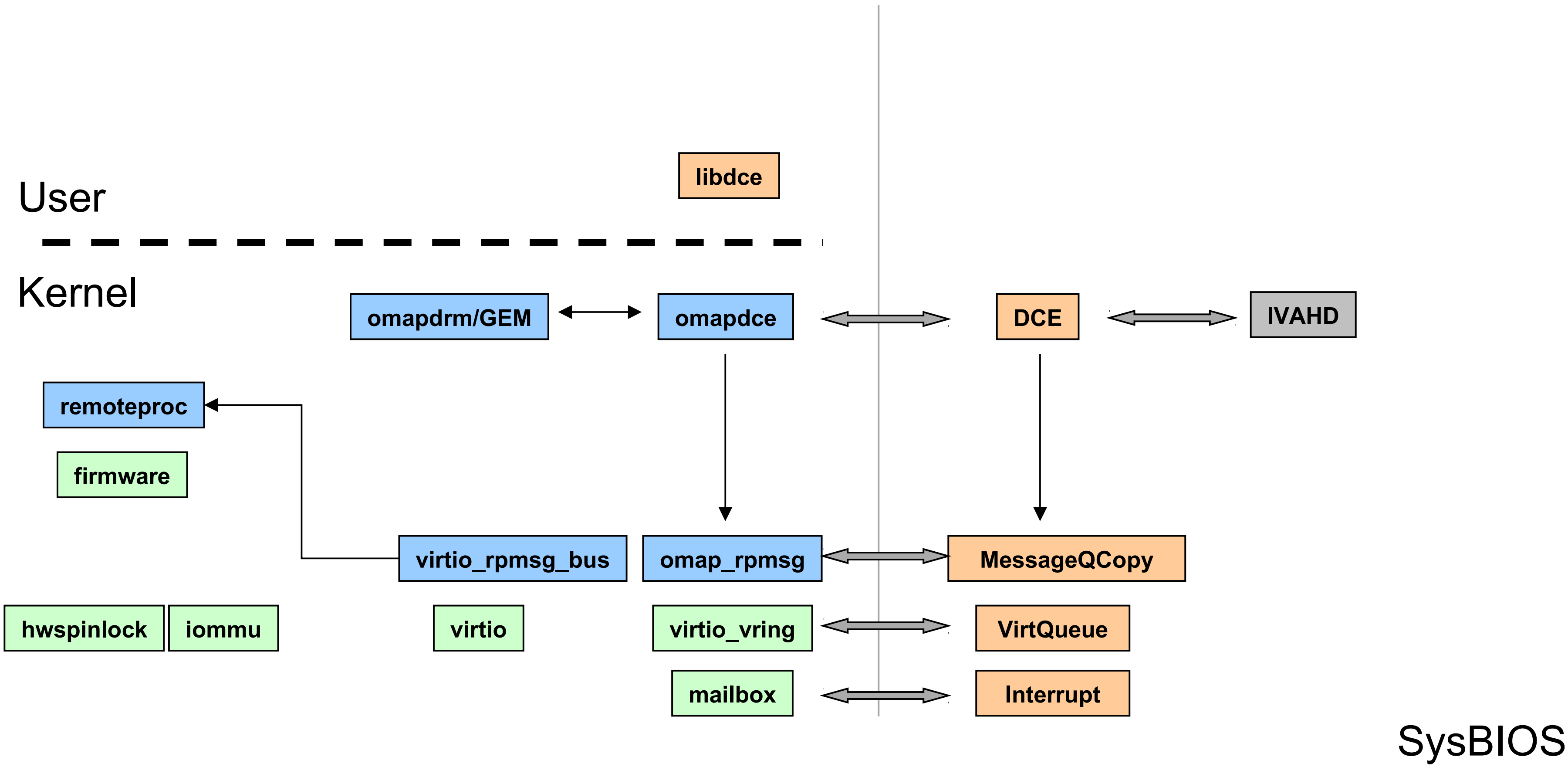
- We eventually came to our senses about a sane way to use video decode/encode accelerators: DCE
- OpenMAX → DCE
  - Removes a layer + many kloc
  - Simplified IPC, fewer IPC/frame
  - The CE engine API beneath OMX is actually a quite sensible API
    - Doesn't try to hide things like locked reference frames
    - Synchronous, gets rid of lots of possible race conditions
  - Results is fewer lines of code in gst elements working around OMX



rpmsg

- A simple kernel level framework for IPC with coprocessors
  - No userspace component
  - No userspace API
  - Considerably smaller/simpler than syslink
  - Because it is kernel level, omapdce driver can use linux kernel frameworks for IVAHD power management, dynamic buffer mapping/eviction to DMM/TILER
- Based on virtio kernel infrastructure
- Handles firmware loading
- Designed to support more than just OMAP
- Upstream
  - Core infrastructure is upstream, OMAP specific parts are waiting for some IOMMU enhancements

rpmsg+dce



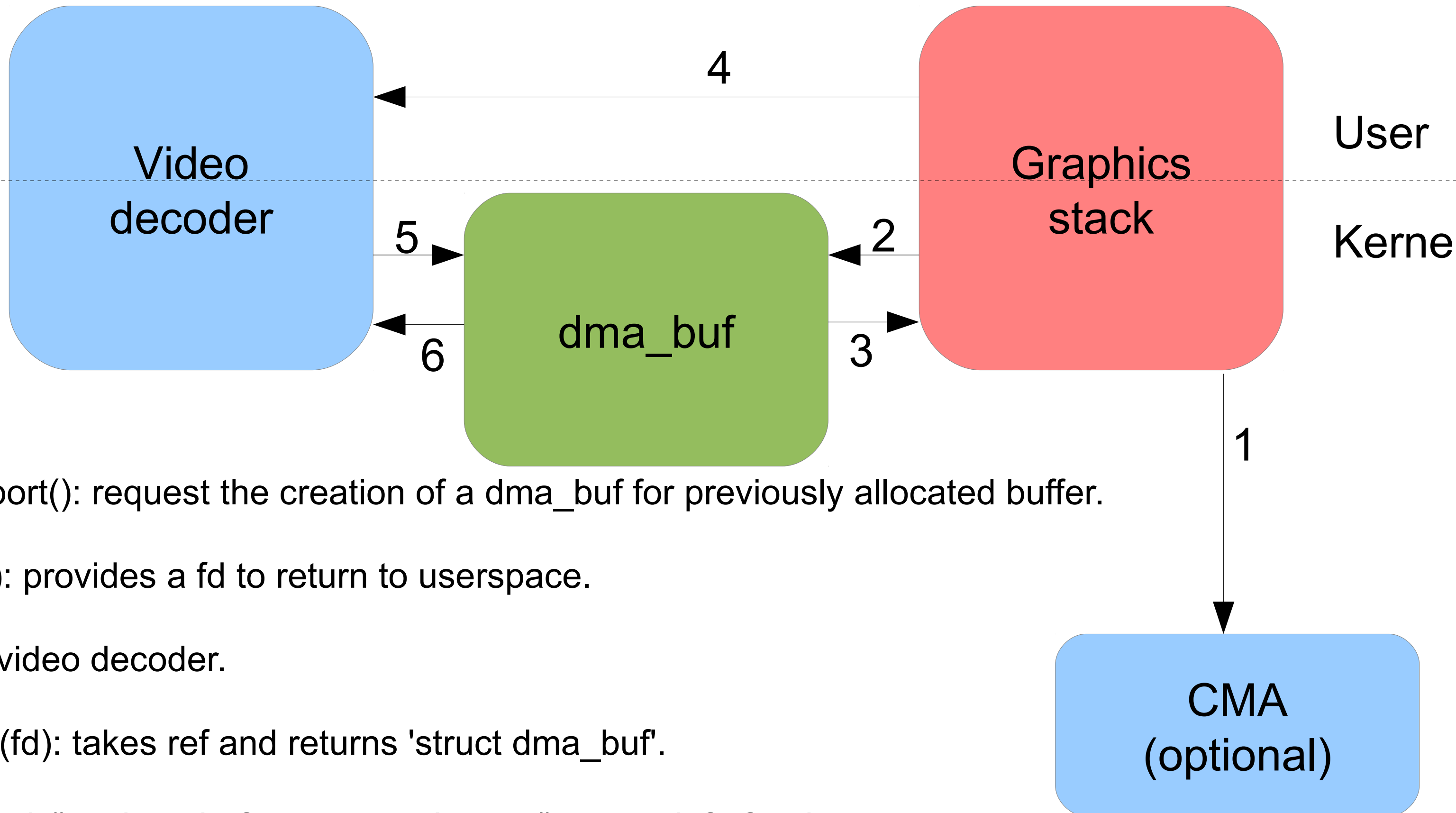
(android+openmax based solution has a similar picture with many more boxes)

dmabuf

- Kernel mechanism for sharing buffers between devices
  - Based on 'struct file'
    - Provides reference counting
    - And file descriptor, for passing between processes, and cleanup if process exits
  - Provides kernel level APIs for drivers to attach buffers, get address (scatterlist), kmap, etc

- No direct userspace API
  - Existing devices can import/export dmabuf handles (fd)
    - V4L2: V4L2_MEMORY_FD
    - DRM: DRM_IOCTL_PRIME_{HANDLE_TO_FD, FD_TO_HANDLE}
  - dmabuf fd's can be mmap()d for userspace access
    - We'll take advantage of this in GStreamer 1.0 to avoid unnecessary mmap
    - For cached buffers on non-coherent architectures, exporting device must do some magic

dmabuf usage flow (example)



1) allocation

2) dma_buf_export(): request the creation of a dma_buf for previously allocated buffer.

3) dma_buf_fd(): provides a fd to return to userspace.

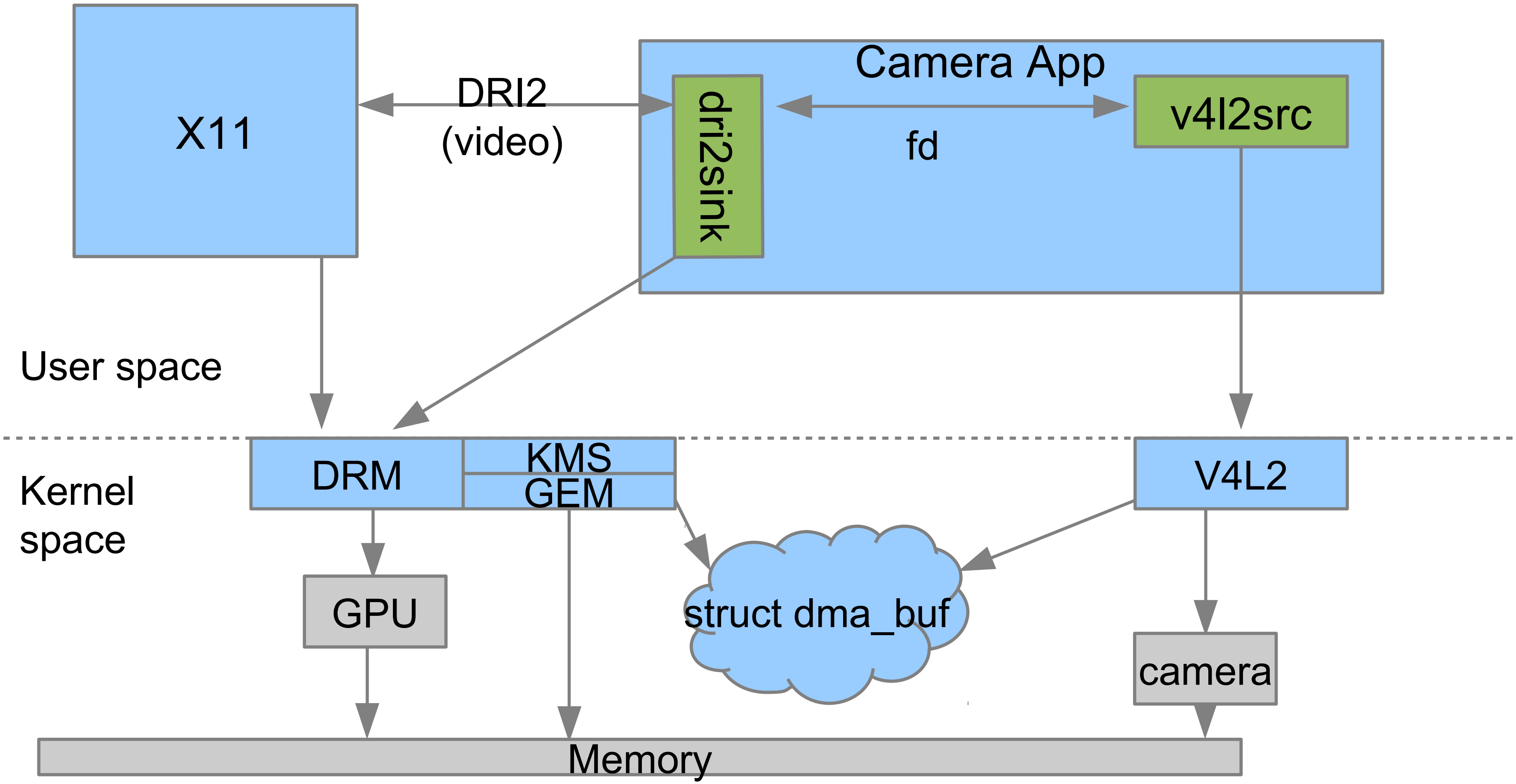
4) fd passed to video decoder.

5) dma_buf_get(fd): takes ref and returns 'struct dma_buf'.

6) dma_buf_attach() + dma_buf_map_attachment(): to get info for dma

- a) dev->dma_parms should be expanded to tell if receiving device needs contiguous memory or any other special requirements
- b) allocation of backing pages could be deferred by exporting driver until it is known if importing driver requires contiguous memory.. to make things a bit easier on systems without IOMMU

dmabuf example



Blinky s***..

putting pixels on the screen

KMS overlays – Keeping it simple

- If you don't need a display server, use hw overlays (kms planes) directly
- Support in GStreamer via kmssink
- Can attach single fb to multiple planes for multi-display
  - Use different src coords to different plane → crtc → encoder → connector to span multiple displays
  - Not yet supported in kmssink but all the kernel bits are there

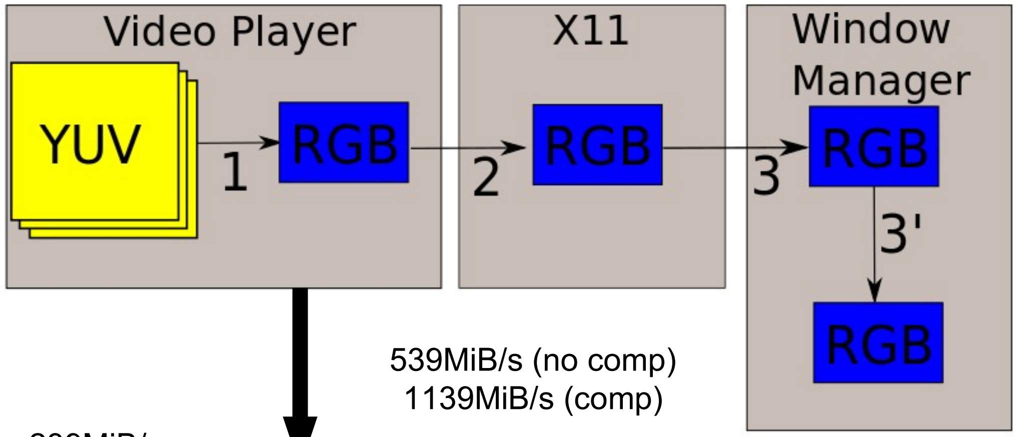
X11 – Traditional Blinky

- Traditionally Xv extension used for rendering video
  - Xshm buffers: 2x memcpy
    - Not terribly good for hw decoders that have special memory requirements
    - And not terribly good for GPUs either.. need a copy into a GPU accessible buffer or at least map/unmap on every frame
- DRI2
  - Used under the hood by VAAPI/VDPAU.. but can only support unscaled RGB buffers, so GPU blit YUV->RGB + scaling done on client side
- DRI2Video
  - Combines the ideas of Xv and DRI2
  - Xserver (DDX driver) allocates GEM buffer and passes to client process
    - Allows us to abstract DMM/TILER stuff in omapdrm kernel driver
  - But unlike DRI2, the buffer can be YUV (incl. Multi-planar), sized according to video size, not scaled drawable size, and cropped
  - Can support zero-copy overlays too: display can scanout GEM buffers
    - But not implemented yet

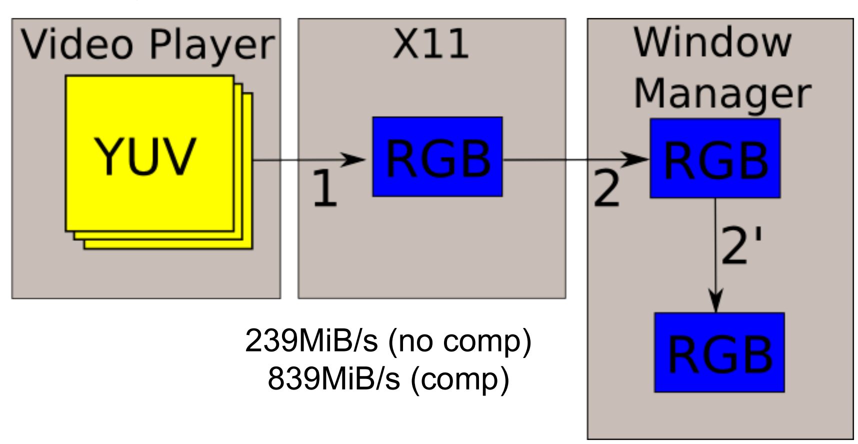
X11 – dri2video

Example memory bandwidth savings based on 1080p 30fps NV12 video rendered to nearly fullscreen window on 1280x1024 display

NV12->RGB = (1920*1080*1.5) + (1280*1024*4) → 239MiB/s
Swap/blit = (1280*1024*4) * 2 → 300MiB/s
Composite = (1280*1024*4) * 2 → 300MiB/s
Presentation blit = (1280*1024*4) * 2 → 300MiB/s

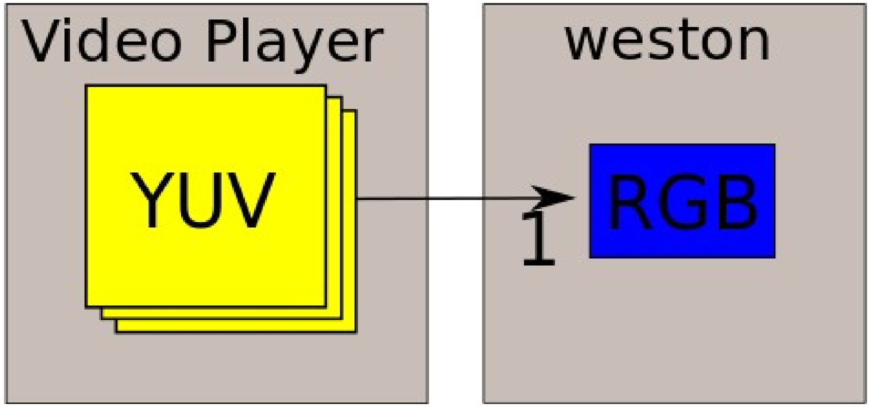


NV12->RGB = (1920*1080*1.5) + (1280*1024*4) → 239MiB/s
Swap/blit = (1280*1024*4) * 2 → 300MiB/s
Composite = (1280*1024*4) * 2 → 300MiB/s
Presentation blit = (1280*1024*4) * 2 → 300MiB/s



Wayland – Simply Blinky

- In wayland, no separation of window manager and display server
  - This makes use of overlays much easier.. which weston already supports
- With wl_drm protocol, we can push YUV buffers directly to server
  - Similar in result to dri2video.. but less copies due to window manager for compositing. And no tearing!
  - Either use overlay or do a YUV->RGB as part of the final composition



239MiB/s

Bringing it all together in GStreamer

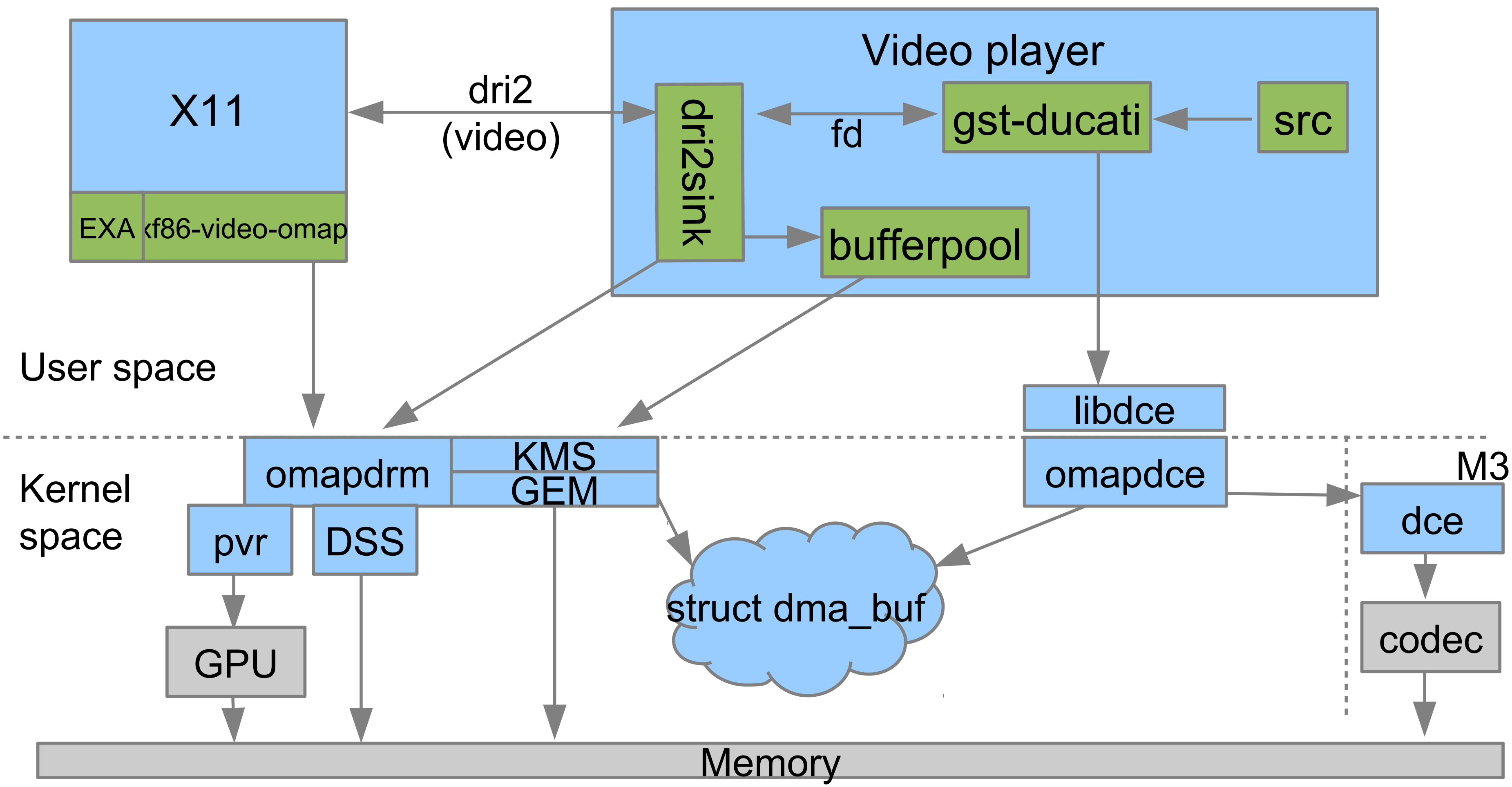
Current status in GStreamer

- Our primary supported environment is (sadly) still GStreamer 0.10
  - Customers still using 0.10
  - Apps support in distros for 1.0 is not there yet
  - And we don't have the manpower to fully test and support both 0.10 and 1.0
- Some experimental support for 1.0
  - And hopefully we can drop 0.10 and switch to 1.0 “soon”

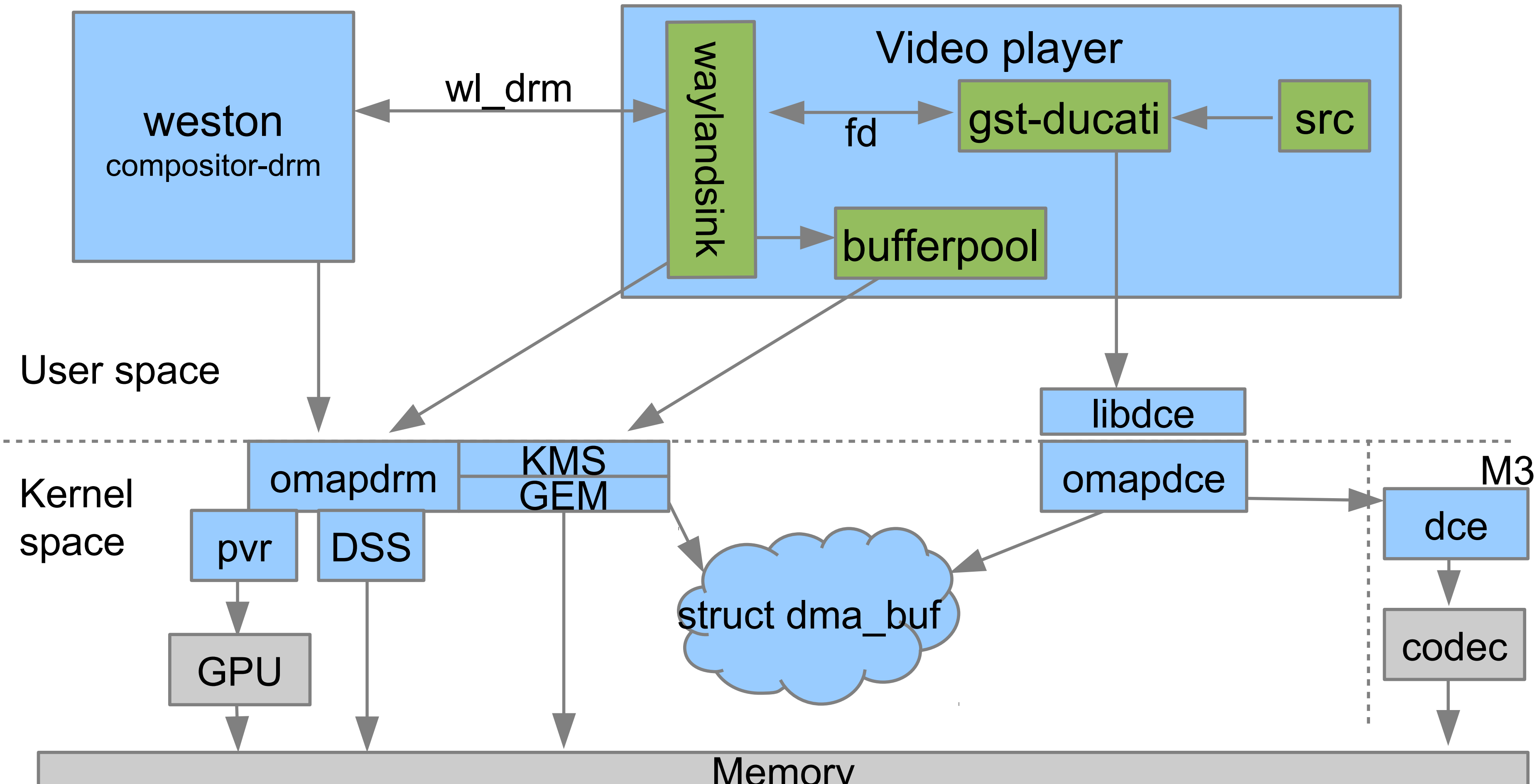
The transition to 1.0

- To better prepare for 1.0, we've made a few changes
  - Using “quark” mechanism to attach what would be GstMeta
    - Public meta:
      - dmabuf fd
      - cropping coordinates
    - Per-element private mapping data
      - GEM handles for decoders/encoders
      - DRM fb-id's for kmssink
      - DRI2 attachment point for dri2videosink
      - eglImage for GL based renders (xbmc, gst-clutter)
  - A common GstDRMBufferPool
    - Attaches GstDmaBuf quark/meta to buffers
    - Allows decoders, sinks, etc, to mostly not care who is allocating the buffer
      - dri2videosink needs to subclass GstDRMBufferPool to allocate via xserver

GStreamer + dmabuf (X11)



GStreamer + dmabuf (Wayland)



The End

(and demo, time and logistics permitting)