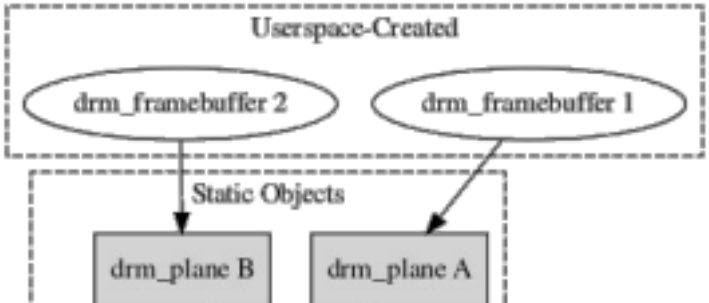
DRM KMS overview

- Framebuffers - memory object, pixel source for the scanout
  - Plane - an image source that can be blended during the scanout



- CRTC - display pipeline, blends Planes together
  - Encoder - connecting element between CRTC and external connector
- Connector - general abstraction for the display sinks

Usually each of these objects will be mapped to hardware resources.

Reminder: atomic mode setting

- Transactional model for updating display pipeline state
- Each pipeline object has a ‘state’ data

- Two steps for changing mode/properties:
  - atomic_check(), checks new state, calculates hw-specific data, [redacted]
  - atomic_commit() / atomic_update(): apply new state to the hardware, [redacted]

DRM Plane abstraction

Plane represents a single image that is blended during the display scanout process Each plane declares several properties:

- Supported image format
- Supported rotation, Z position
- Supported alpha blending, etc

Userspace (composer) selects these properties and validates them using DRM_MODE_ATOMIC_TEST_ONLY flag before setting

Frequently all planes (all hw blocks) have common set of features, making composer simple enough.

Usecase: Qualcomm display subsystem

- Several hardware blocks related to image blending
- Some of them support RGB & YUV and scaling, others only RGB and no scaling

To make things more complex:

- Under certain constraints these blocks can render two images at once instead of rendering just one
- Each block has limited max width, so rendering 4k images would require using two blocks

Downstream solution: export blocks as is and let composer decide how to use hardware features.

This requires adding non-standard plane properties

Composers become hardware-aware

Planes go Virtual

Let the kernel do composer job!

- Allocate as many planes as possible (2x hw blocks, each block can render max 2 images)
- Each plane is created using the superposition of all possible features ● Hardware-unaware composer sets up the planes as required and tests the

configuration

● The DRM driver tries to allocate hardware resources at the mode setting time (possibly returning an error if resources are oversubscribed)

Changes required

Init:

- Allocate one plane per hw block - Specify exact list of supported formats

Atomic_check:

- Check scaling and rotation.
- Calculate next state

Atomic_update:

- Apply calculated state to hardware

- [redacted]
- [redacted] -

Calculate next state

Init:

- Allocate 2x planes
- Specify a superset of the formats

Atomic_update:

- Apply calculated state to hardware

Atomic_check:

Dirty details

atomic_check might be rolled back, so resource allocation should be in the drm private object.

Release and allocate should come in two consecutive steps:

1. Release all freshly disabled blocks
2. Allocate blocks for new planes

Our implementation:

1. plane's atomic_check() callback (it is called for all planes) does release 2. CRTC's atomic_check() would allocate HW blocks for all planes bound to the CRTC 3. Then CRTC's atomic_check() would call dpu_plane_real_atomic_check()