igel

Technology Consulting Company
Research, Development &
Global Standard

Using Linux Media Controller for Wayland/Weston Renderer

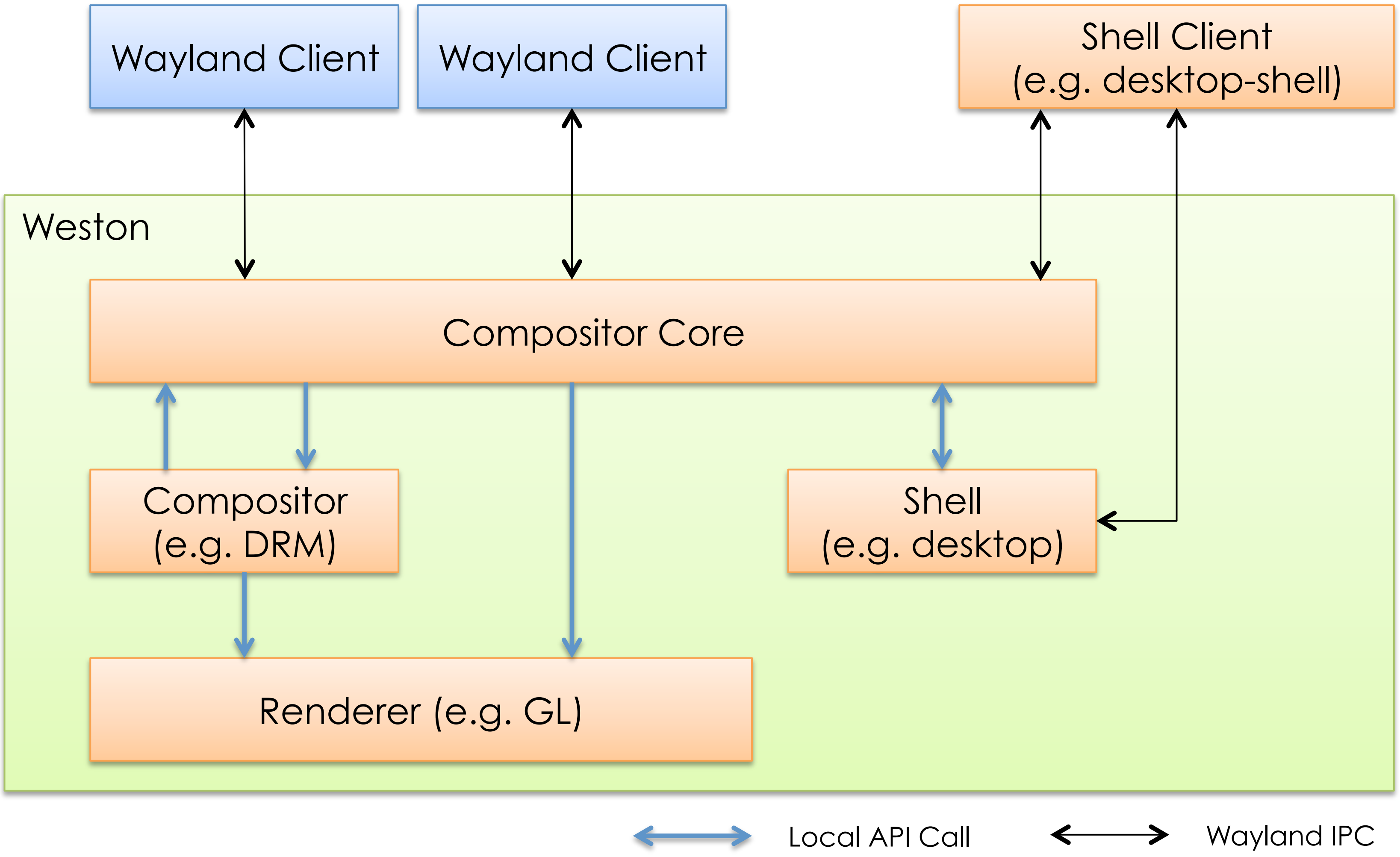
Takanari Hayama
taki@igel.co.jp
http://www.igel.co.jp/

Agenda

- Wayland/Weston Overview
- Porting Weston to R-Car
- Why Linux Media Controller Renderer?
- Linux Media Controller Framework
- V4L2 Renderer Design
- Conclusions

WAYLAND/WESTON OVERVIEW

Weston Architecture



Wayland Client

Wayland Client

Shell Client
(e.g. desktop-shell)

Weston

Compositor Core

Compositor
(e.g. DRM)

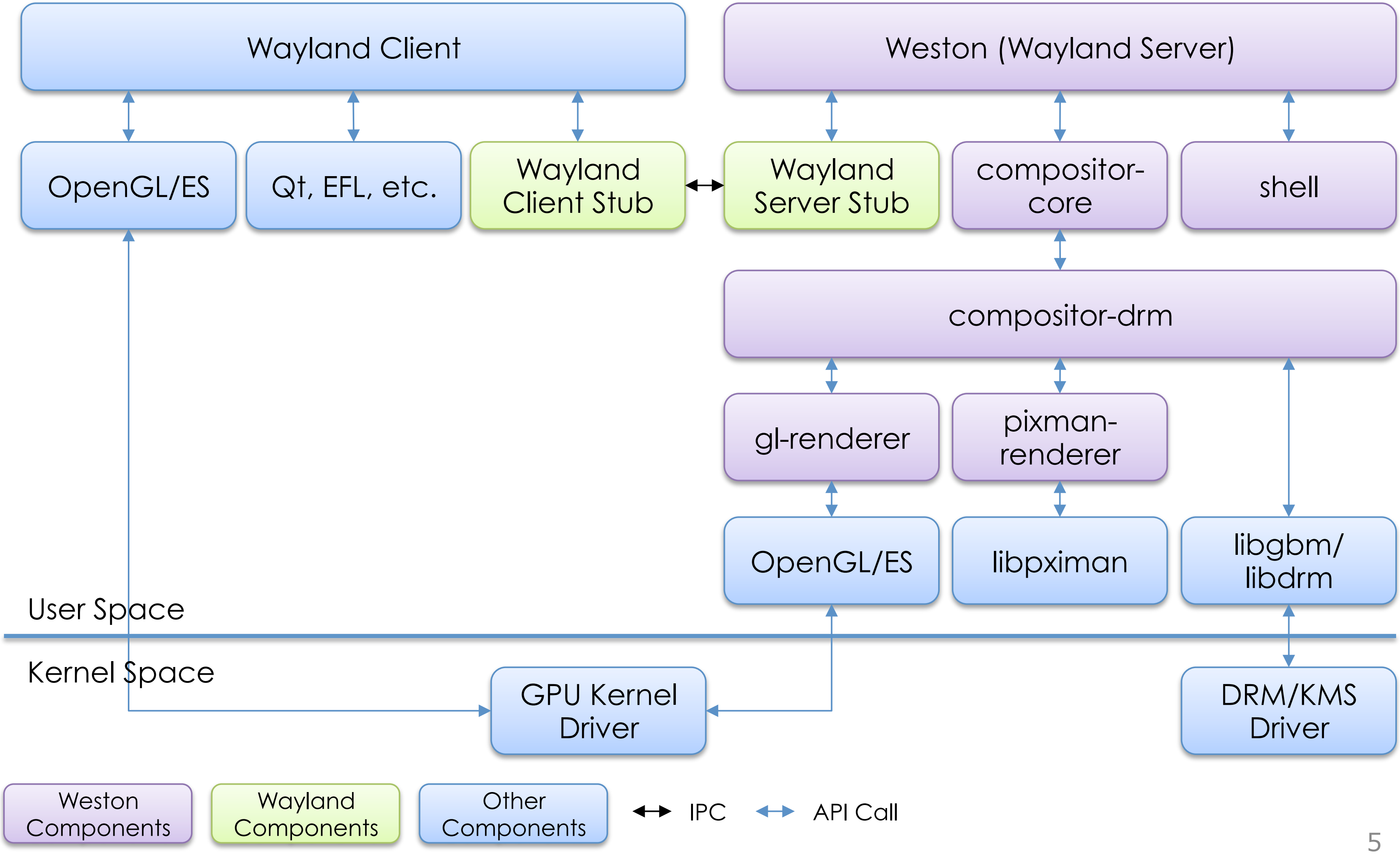
Shell
(e.g. desktop)

Renderer (e.g. GL)

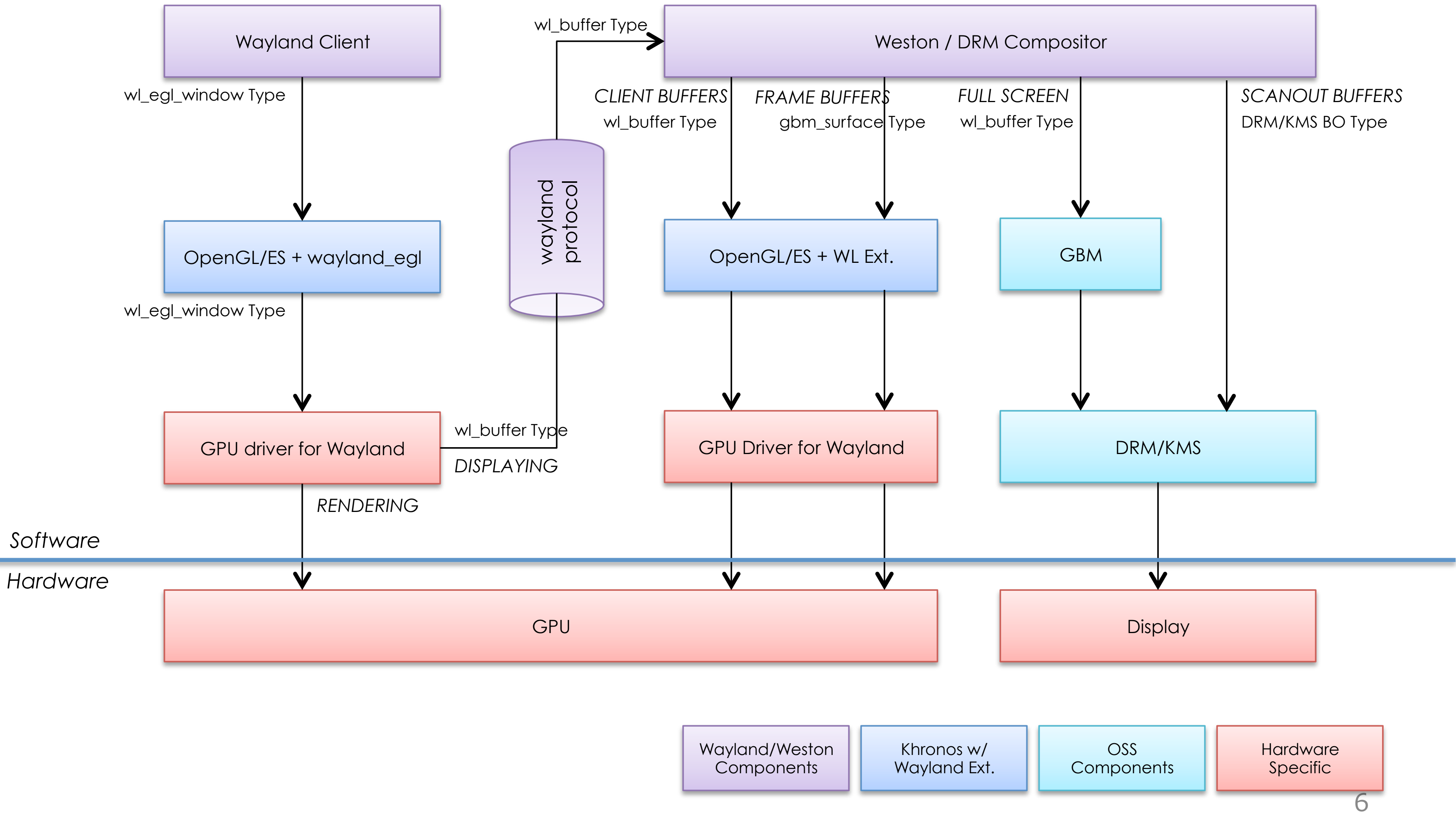
Local API Call

Wayland IPC

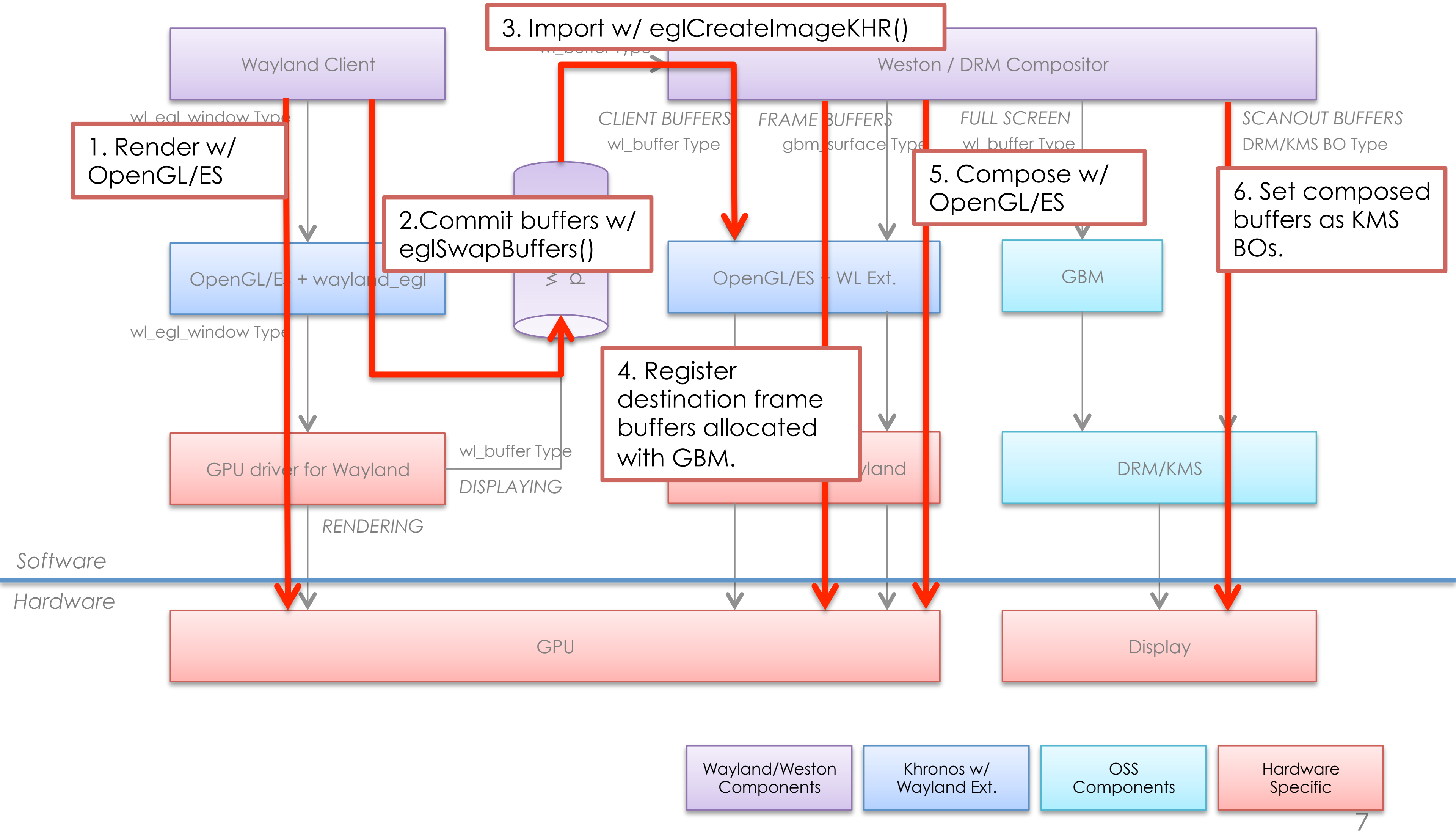
Weston w/ DRM Backend



Rendering and Composition: Overview (GL-Renderer)



Rendering and Composition: Window Composition



1. Render w/ OpenGL/ES

2.Commit buffers w/ eglSwapBuffers()

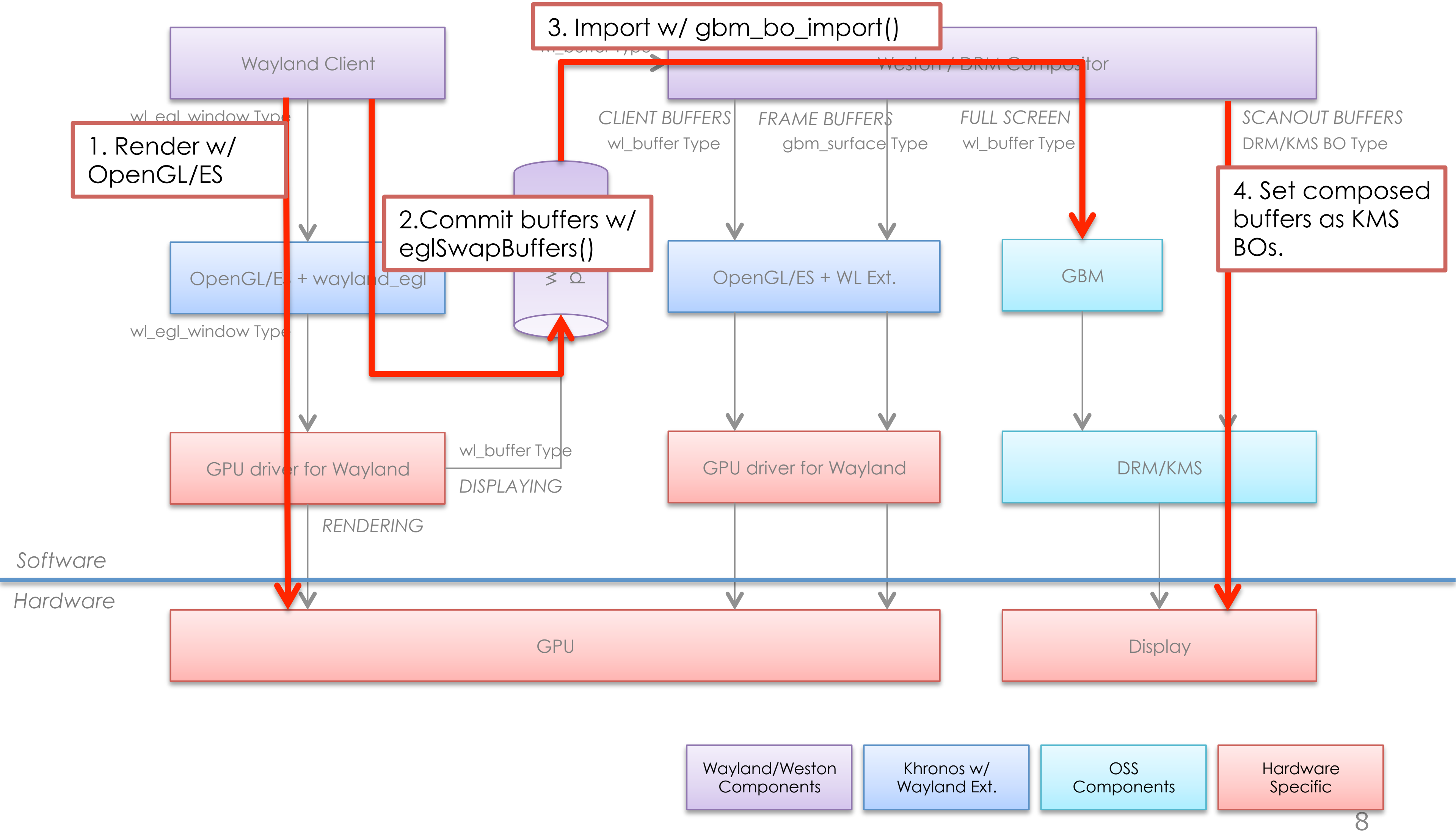
3. Import w/ eglCreateImageKHR()

4. Register destination frame buffers allocated with GBM.

5. Compose w/ OpenGL/ES

6. Set composed buffers as KMS BOs.

Rendering and Composition: Full Screen or Sprite Rendering



1. Render w/ OpenGL/ES

2.Commit buffers w/ eglSwapBuffers()

3. Import w/ gbm_bo_import()

4. Set composed buffers as KMS BOs.

PORTING WESTON TO R-CAR

What Are Required?

1. OpenGL/ES for Wayland/Weston

2. Zero Copy Mechanism for Native Buffer

"Typically, hardware enabling includes modesetting/display and EGL/GLES2. On top of that, Wayland needs a way to share buffers efficiently between processes."
http://wayland.freedesktop.org/architecture.html

Wayland Requirements for OpenGL/ES

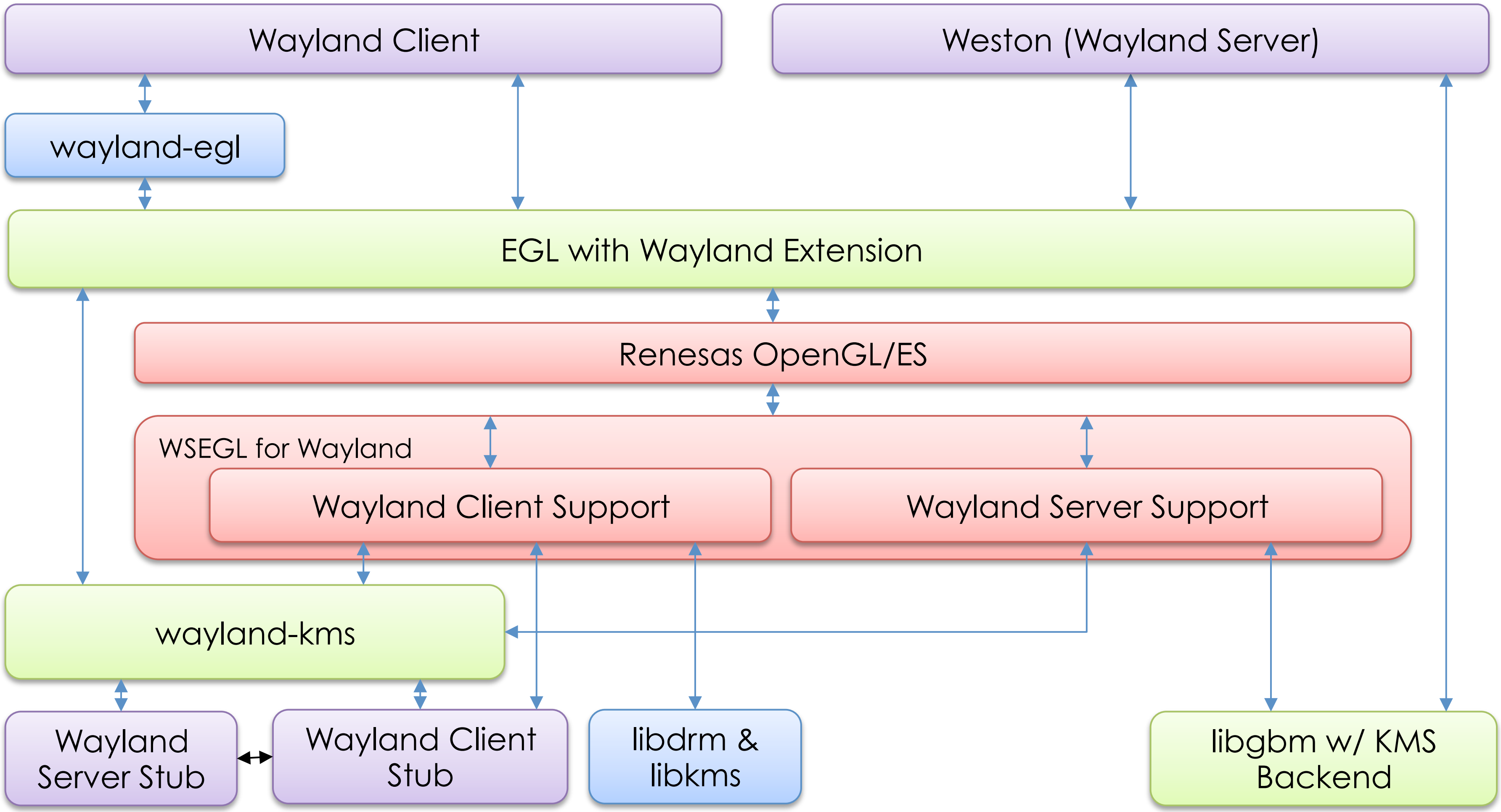
- Must support the following Native Display Types for eglGetDisplay():
  - `wl_display` for clients
  - `gbm` handle for Weston

- Must support the following EGL_EXTENSIONs:
  - `EGL_KHR_image_pixmap`
  - `EGL_WL_bind_wayland_display`

- Must support the following Native Pixmap Type for eglCreateImageKHR():
  - `EGL_WAYLAND_BUFFER_WL`

- Must support the following Wayland extension APIs:
  - `eglBindWaylandDisplayWL`
  - `eglUnbindWaylandDisplayWL`
  - `eglQueryWaylandBufferWL`

Weston for Renesas R-Car



Wayland/Weston Components	Standard OSS Components	New OSS Components	Proprietary	↔ IPC	↔ API Call

Wayland Composition Revisited

1. A client creates a *wl_surface* on the server.
2. The client attach a *wl_buffer* to the created surface.
3. The client submit the *wl_buffer* to the server.
4. The server takes the *wl_buffer* and compose to the screen.

All of above should happen in zero-copy manner!

What is wl_buffer by the way?

- An abstract data type that represents a reference to a pixel buffer.

- 2 open source implementations:
  - wl_shm : wayland standard
    - Based on Linux shared memory. Not physically contiguous.
  - wl_drm : Mesa standard
    - Based on DRI. Possibly physically contiguous.

- Weston understands wl_shm only. Wl_drm is Mesa specific. Thus, wl_drm is not handled by Weston, but by Mesa internally.

Which wl_buffer implementation to use?

- Requirements
  - End-to-end Buffer Zero Copy
  - Physically Contiguous Memory
- wl_drm?
  - Implementation is too Mesa dependent.
- Need more generic implementation.

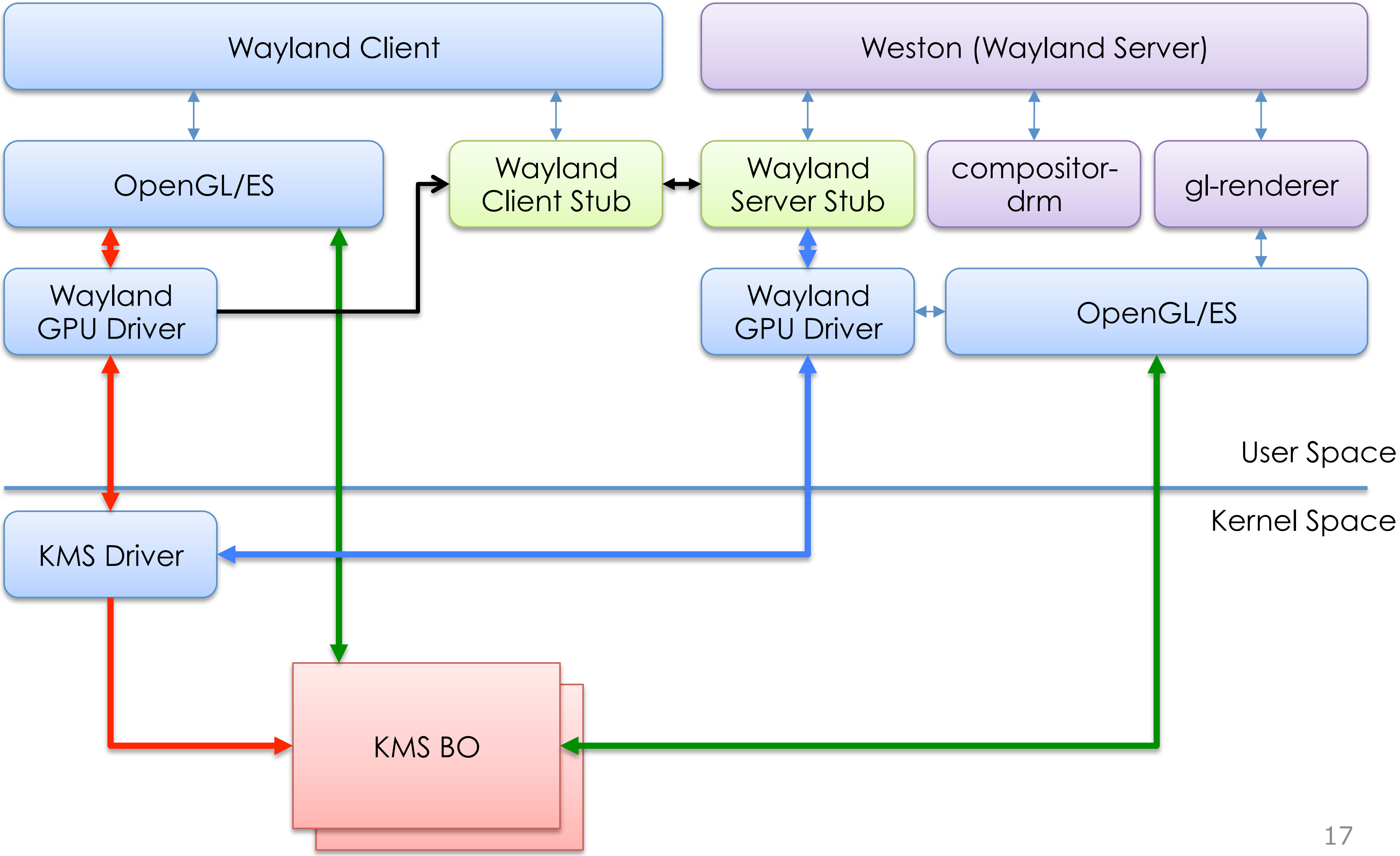
wl_kms

- KMS BO buffer type.
  - https://github.com/thayama/wayland-kms
  - Based on wl_drm in Mesa.

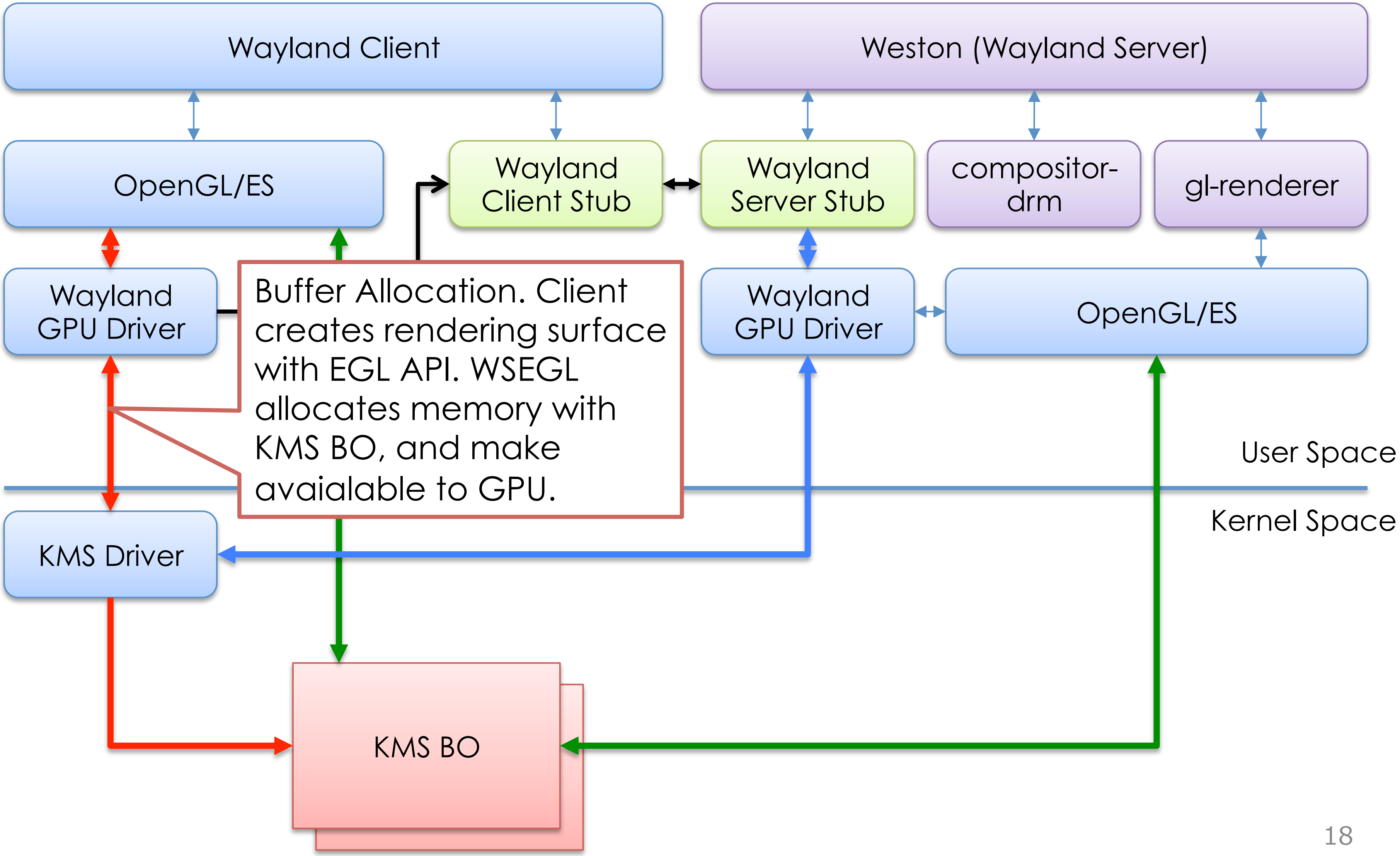
- Imports DMABUF via PRIME, a dma-buf interface layer in DRM.
  - Originally, we used DRM Handle, but we now use DMABUF instead.

- Can directly pass video output from V4L2.

Buffer Zero Copying with wl_kms

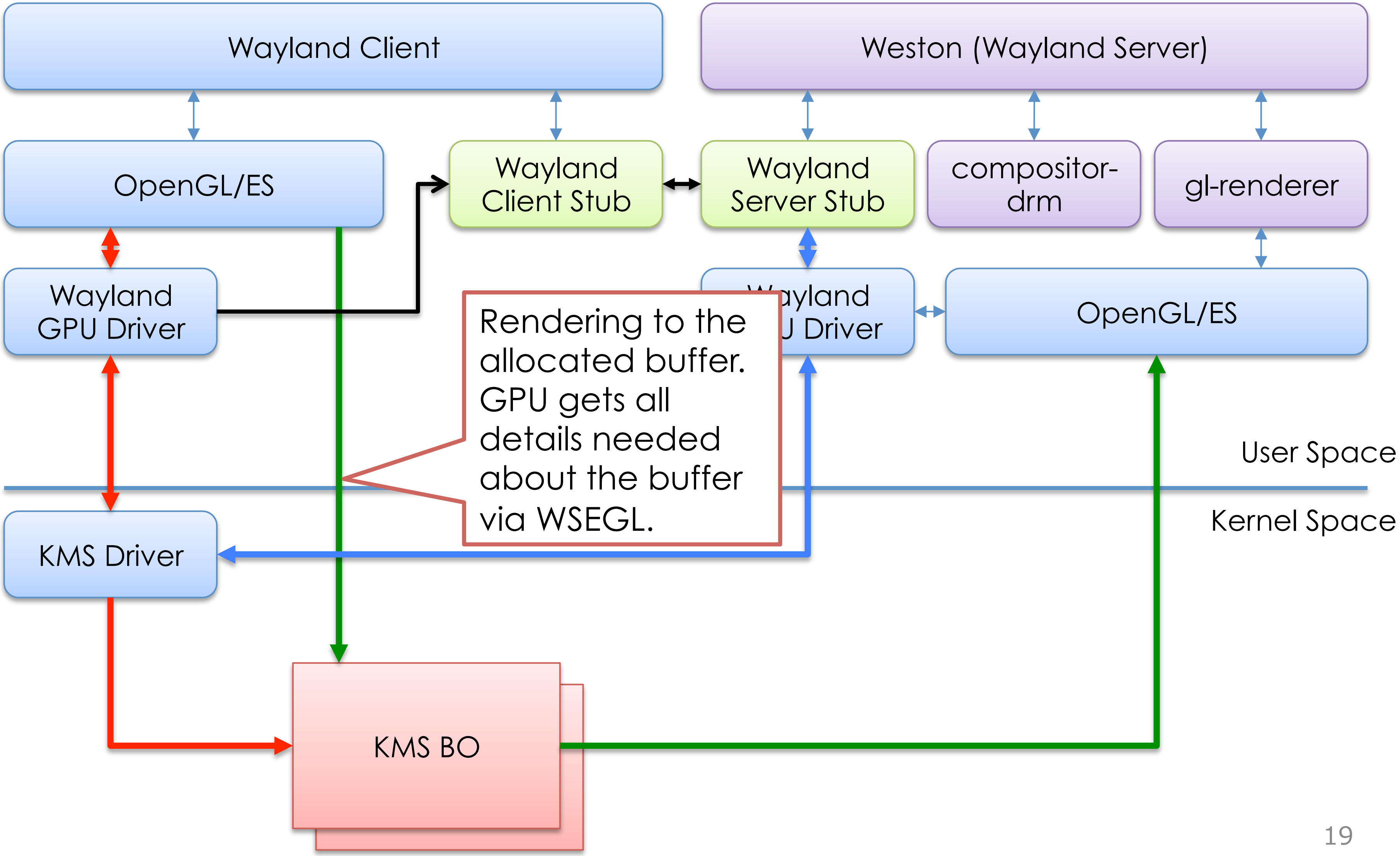


Buffer Zero Copying with wl_kms



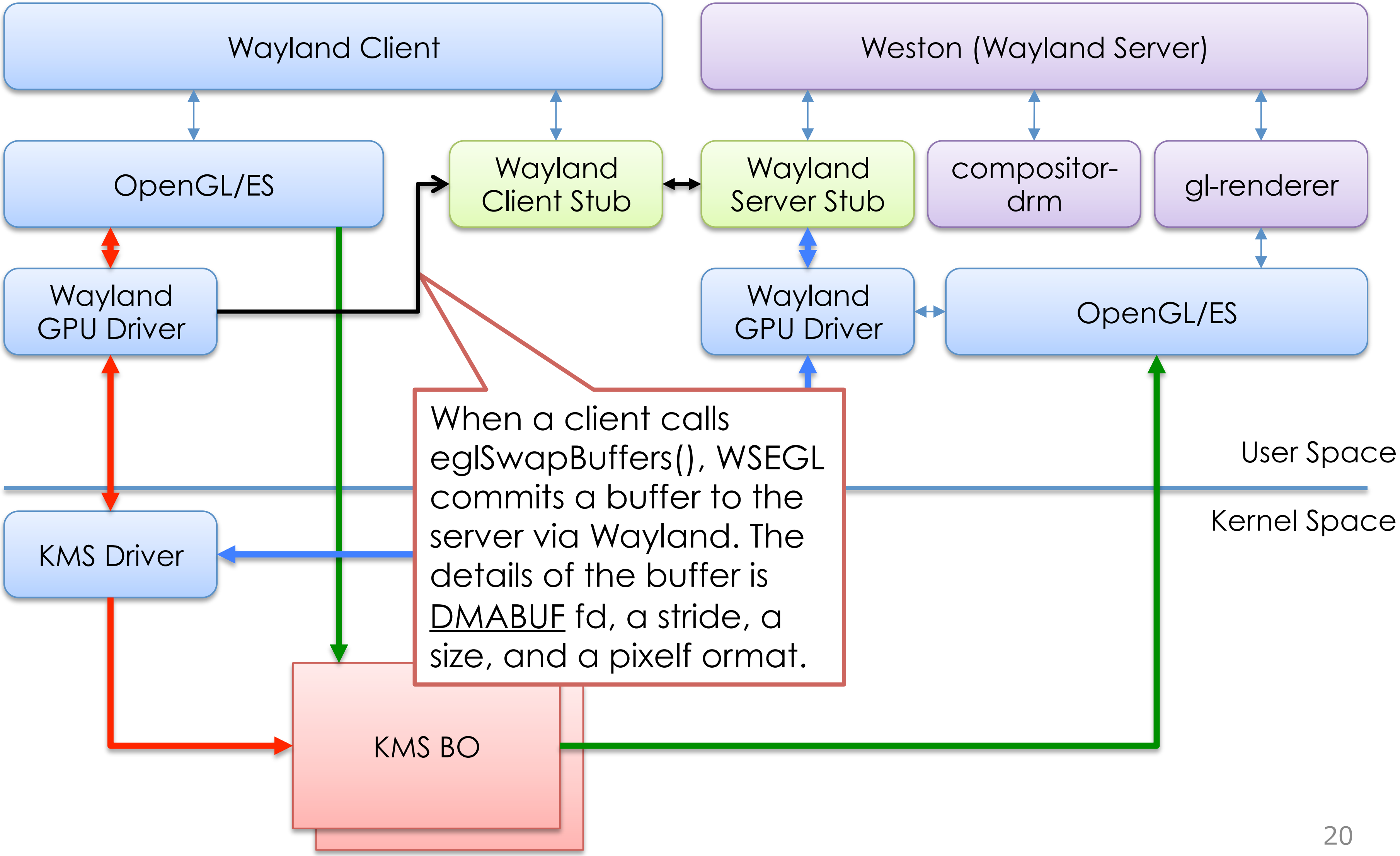
Buffer Allocation. Client creates rendering surface with EGL API. WSEGL allocates memory with KMS BO, and make avaialable to GPU.

Buffer Zero Copying with wl_kms



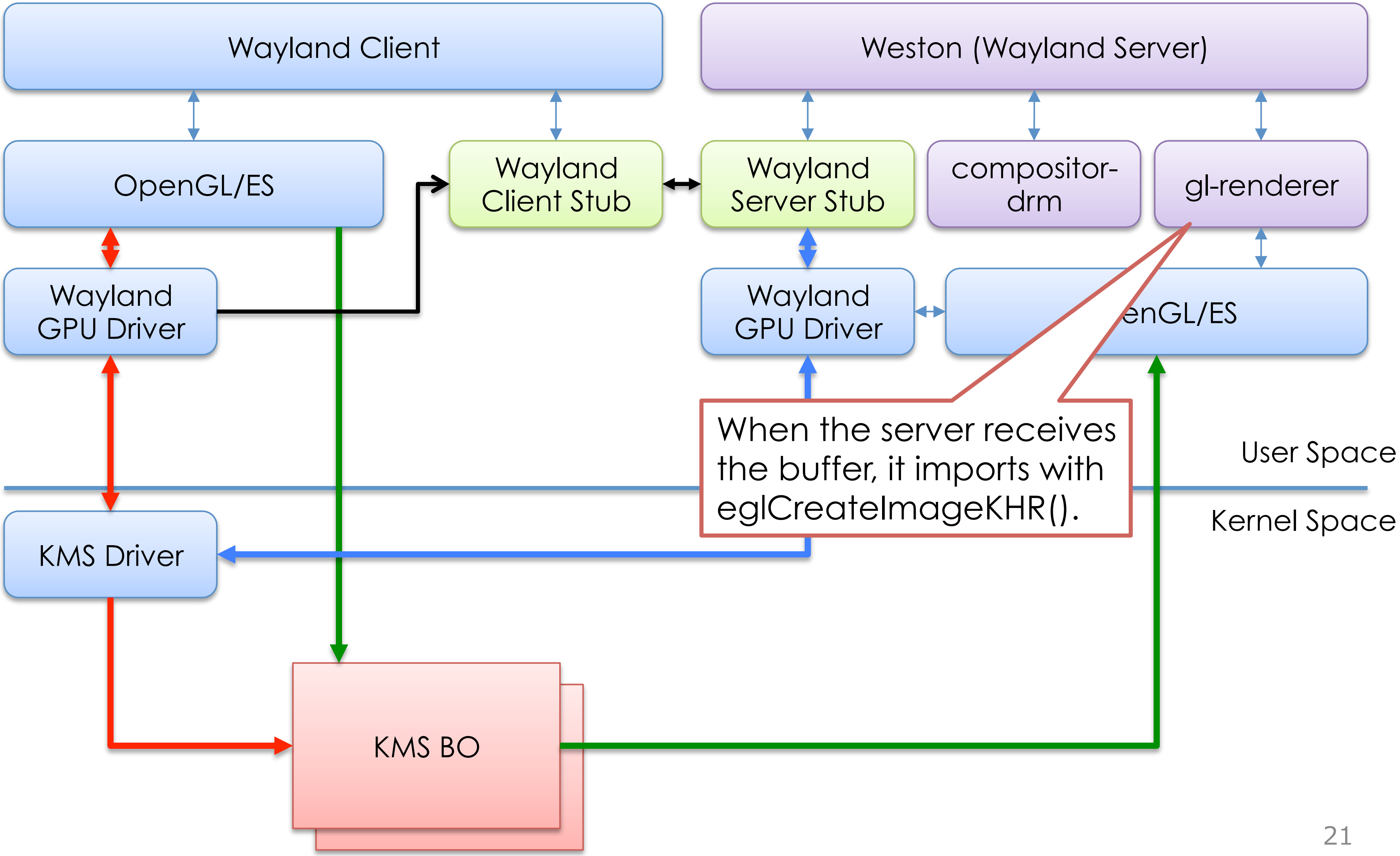
Rendering to the allocated buffer. GPU gets all details needed about the buffer via WSEGL.

Buffer Zero Copying with wl_kms



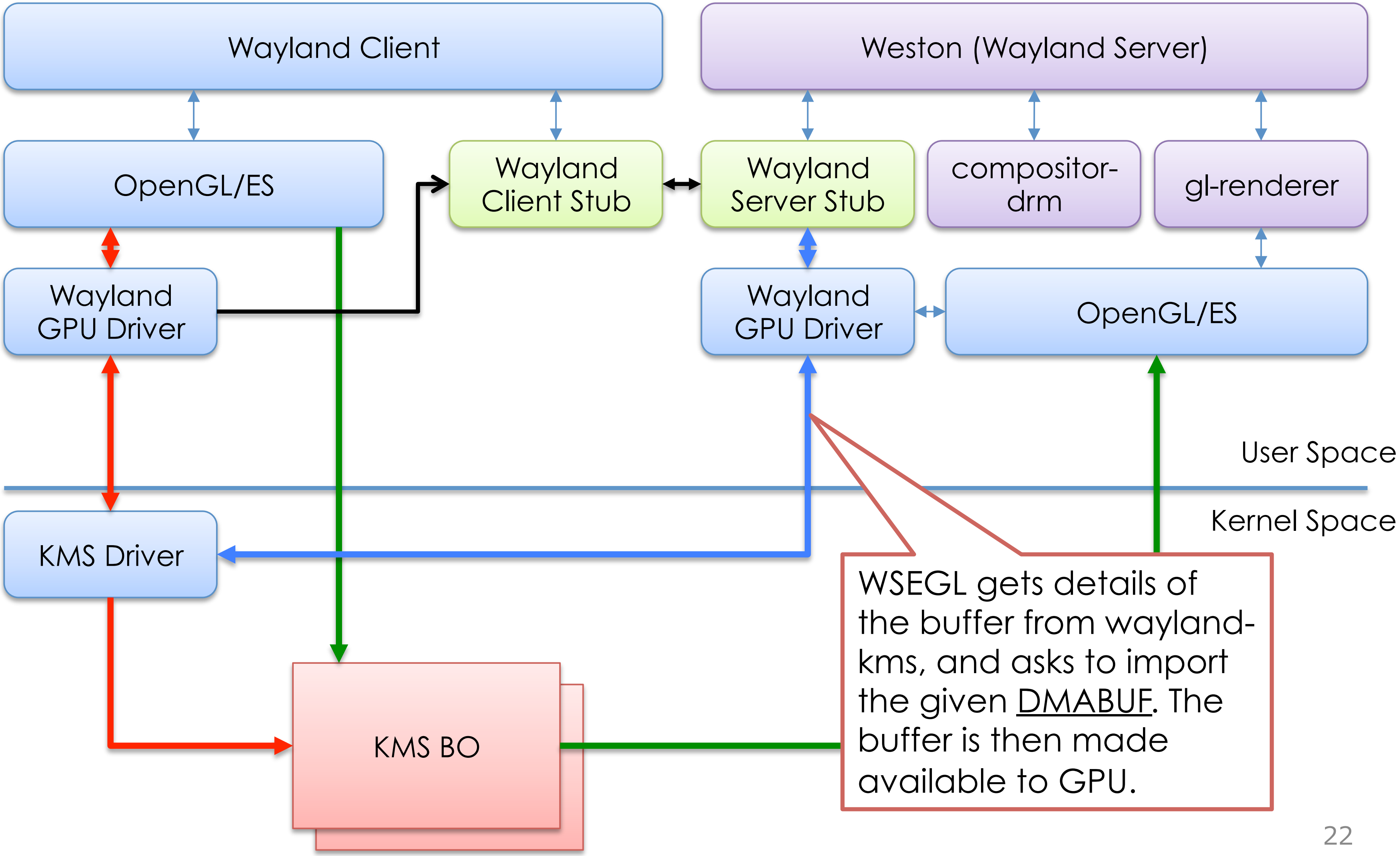
When a client calls eglSwapBuffers(), WSEGL commits a buffer to the server via Wayland. The details of the buffer is DMABUF fd, a stride, a size, and a pixelf ormat.

Buffer Zero Copying with wl_kms



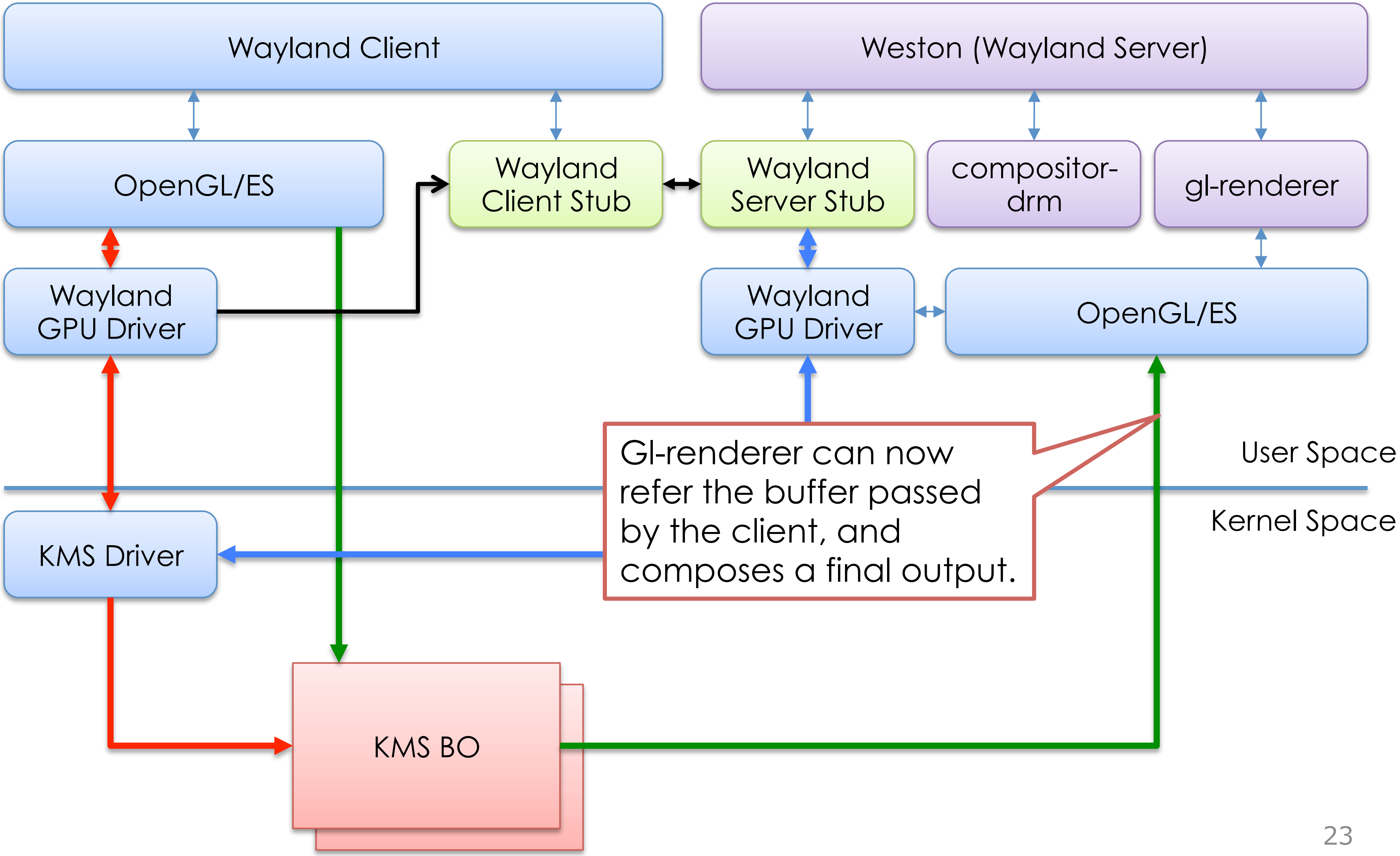
When the server receives the buffer, it imports with eglCreateImageKHR().

Buffer Zero Copying with wl_kms



WSEGL gets details of the buffer from wayland-kms, and asks to import the given DMABUF. The buffer is then made available to GPU.

Buffer Zero Copying with wl_kms



Gl-renderer can now refer the buffer passed by the client, and composes a final output.

WHY LINUX MEDIA CONTROLLER RENDERER?

Motivation

- Applications are heading towards more and more GPU intensive.
- People want to use GPU for more advanced UI, rather than a simple window composition.
  - On the other hand, some people want to do more complex composition using GPU. ☺
- GPU Offloading is one way.
  - https://archive.fosdem.org/2014/schedule/event/wayland_gpu/
  - But, still premature for real products.

Simpler Approach?

- Not many embedded SoC has multiple GPUs.

- However, they often have sophisticated hardware for video signal processing that allow to do 2D blending.

- Why not use them?

LINUX MEDIA CONTROLLER FRAMEWORK

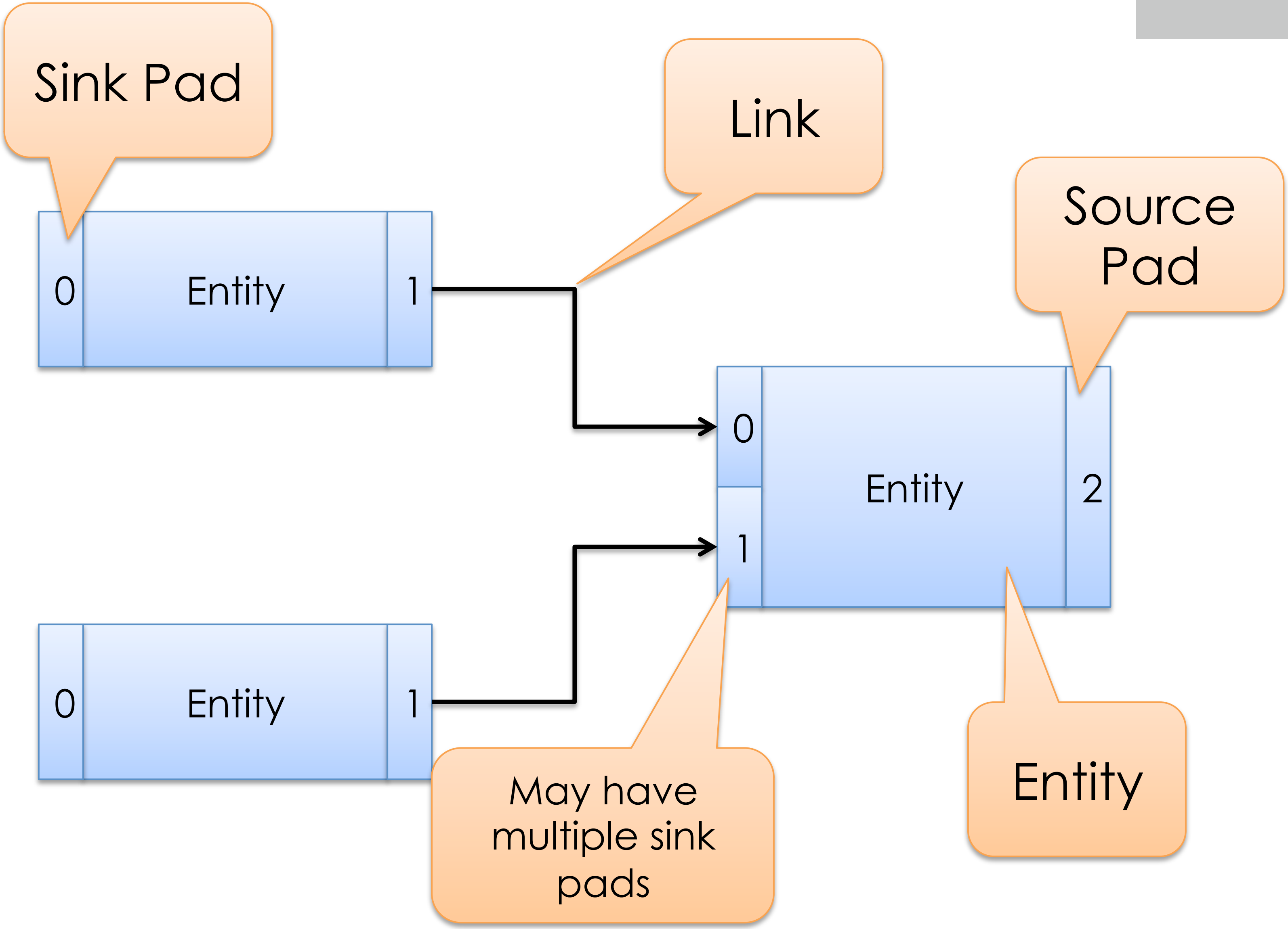
What is Media Controller?

- Just a Video4Linux2 device.

- Make V4L2 media device parameters and pipelines configurable from user space.
  - http://linuxtv.org/downloads/presentations/summit_jun_2010/20100206-fosdem.pdf

- Important keywords: Entities, Pads, and Links.

Media Controller



Why Media Controller?

- A standard Linux API to configure complicated media devices from user space.
- On Renesas R-Car, VSP1, a device for video signal processing, is exposed via Media Controller API in Linux.
- Zero-copy could be easily achieved via DMABUF. Ideal for our use case.
  - Can use wl_kms!

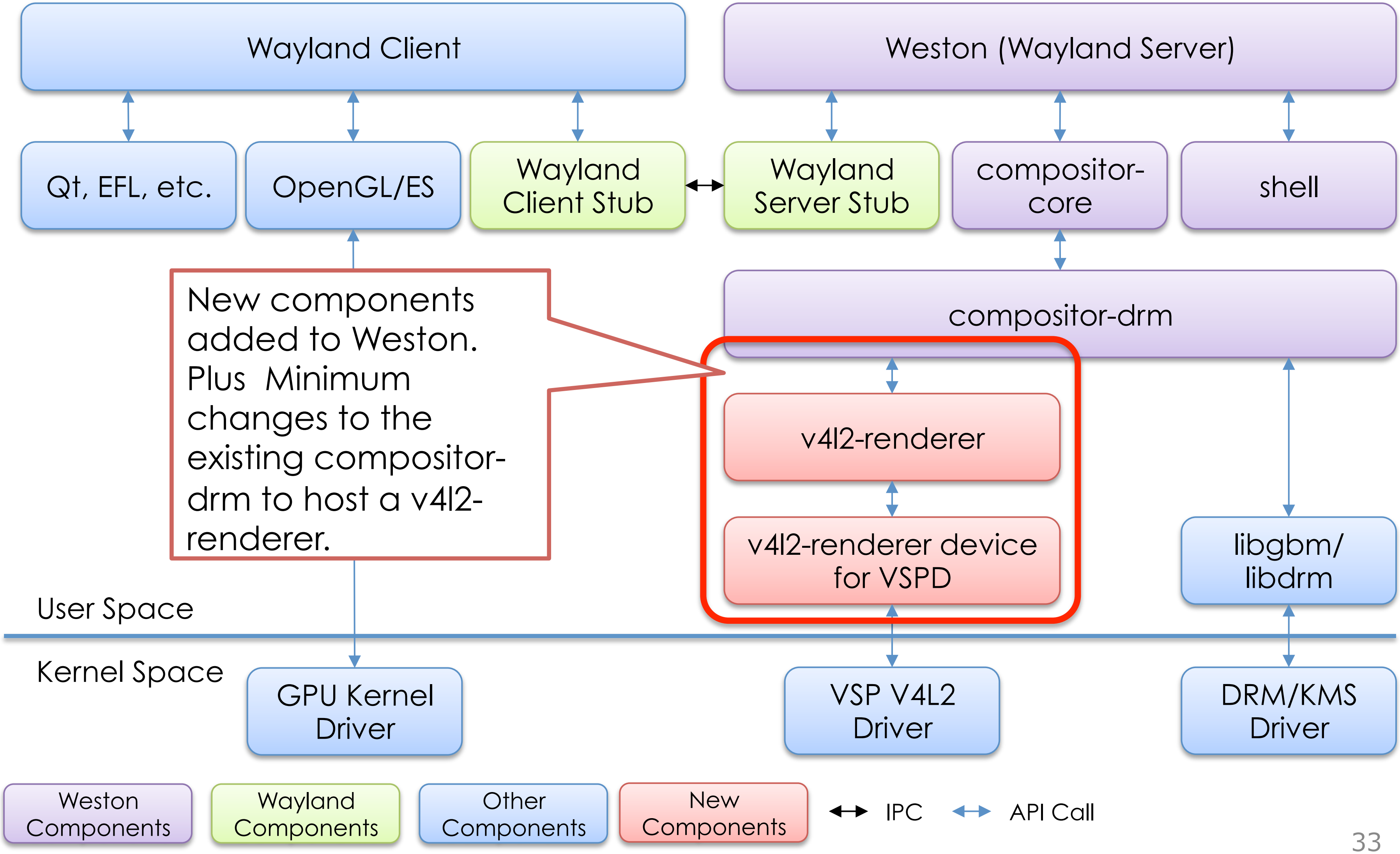
Renesas R-Car VSP1

- Supported Features
  - Scaling
  - Cropping
  - Pixel Format Conversion
  - 4 to 1 Blending

- As pipelines in VSP1 are configurable, best suits to Media Controller Framework!

V4L2 RENDERER FOR WESTON

V4L2 Renderer Support in Weston



New components added to Weston. Plus Minimum changes to the existing compositor-drm to host a v4l2-renderer.

Changes to compositor-drm

- Kept as minimum as possible.

- Changes are to load v4l2-renderer, and pass output buffers to v4l2-renderer. Almost same as those of for pixman-renderer.

- Minor changes on output buffer allocations; DMABUF export and a read permission to mmap’d output buffer are added.

v4l2-renderer

- Media device agnostic layer.
- Does everything needed to import wl_kms and wl_shm buffer to V4L2 Media Controller arena, i.e. DMABUF.
- Calculations required to figure out source regions and destination regions are done in v4l2-renderer.
- Anything that are not media device specific is handled in v4l2-renderer.

V4L2 renderer device

- Media device specific layer.

- Does everything specific to the media devices.
  - Media Controller Framework requires the background knowledge of the underlying media devices.

- Does actual job to compose surfaces specified as DMABUF from v4l2-renderer.

V4L2 Renderer Device API

API	Descriptions
init	Initialize a v4l2 media controller device.
create_output	Create an output. No buffer passed yet.
set_output_buffer	Set an output buffer for the output.
create_surface	Create a surface. No buffer passed yet.
attach_buffer	Set a buffer for the surface.
begin_compose	Begin a new composition.
finish_compose	Finish the composition.
draw_view	Compose the surface.
get_capabilities	Get capabilities of the V4L2 Renderer Device.

Current Status

- V4L2 Renderer is not official yet.
  - https://github.com/thayama/weston

CONCLUSIONS

Was Media Controller the Right Choice?

- Yes and No.

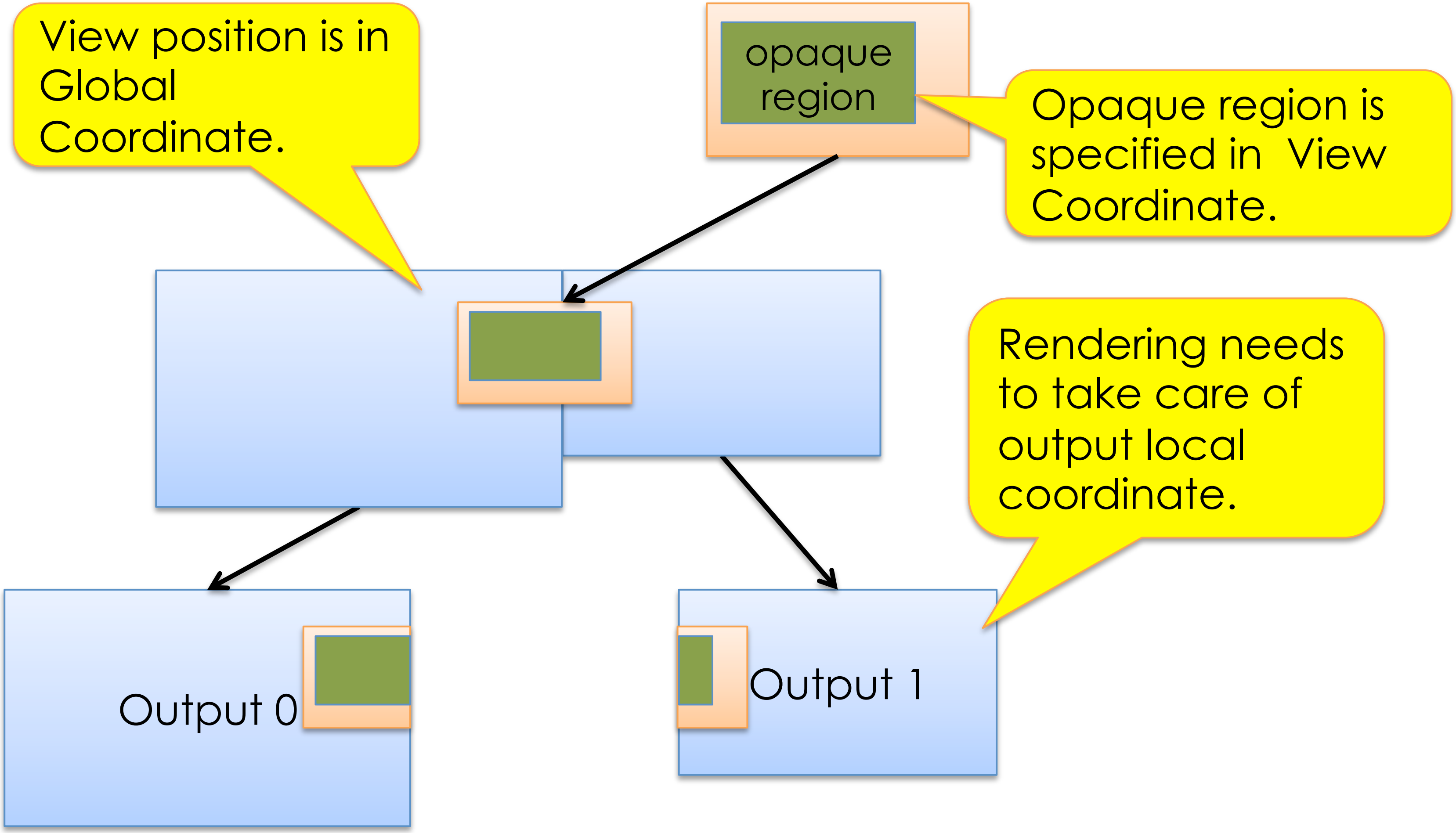
- Configuring parameters and links are not free. Not really great if we need to configure media device very often.
  - DRI could be alternative.

- On the other hand, use of the Linux standard features including DMABUF is great for extensibility and flexibility.

Was implementing new renderer easy?

- Yes and No.
- Other renderers help you implementing new renderer.
- Geometry was the most complex part of Weston.
  - Global coordinate, Output local coordinate, and view coordinate.
- Renderers are responsible for understanding these coordinates and rendering views to the correct location.
  - Pixman-renderer can give you some idea; how complicated it is.

Coordinate System in Weston



View position is in Global Coordinate.

Opaque region is specified in View Coordinate.

Rendering needs to take care of output local coordinate.

APPENDIX

Components used in Renesas OpenGL/ES for Wayland

Components	Descriptions
EGL with Wayland Extension	A thin layer to support Wayland specific EGL APIs and a native buffer type for eglCreateImageKHR(). https://github.com/thayama/libegl
wayland-kms	A subclass of wl_buffer for to pass KMS BO. Defines a wl_kms wayland protocol, and server side codes. https://github.com/thayama/wayland-kms
libgbm w/ KMS Backend	GBM frontend extracted from Mesa and used in Weston with a KMS Backend support. https://github.com/thayama/libgbm
WSEGL for Wayland	A bridge component between GPU and Wayland.