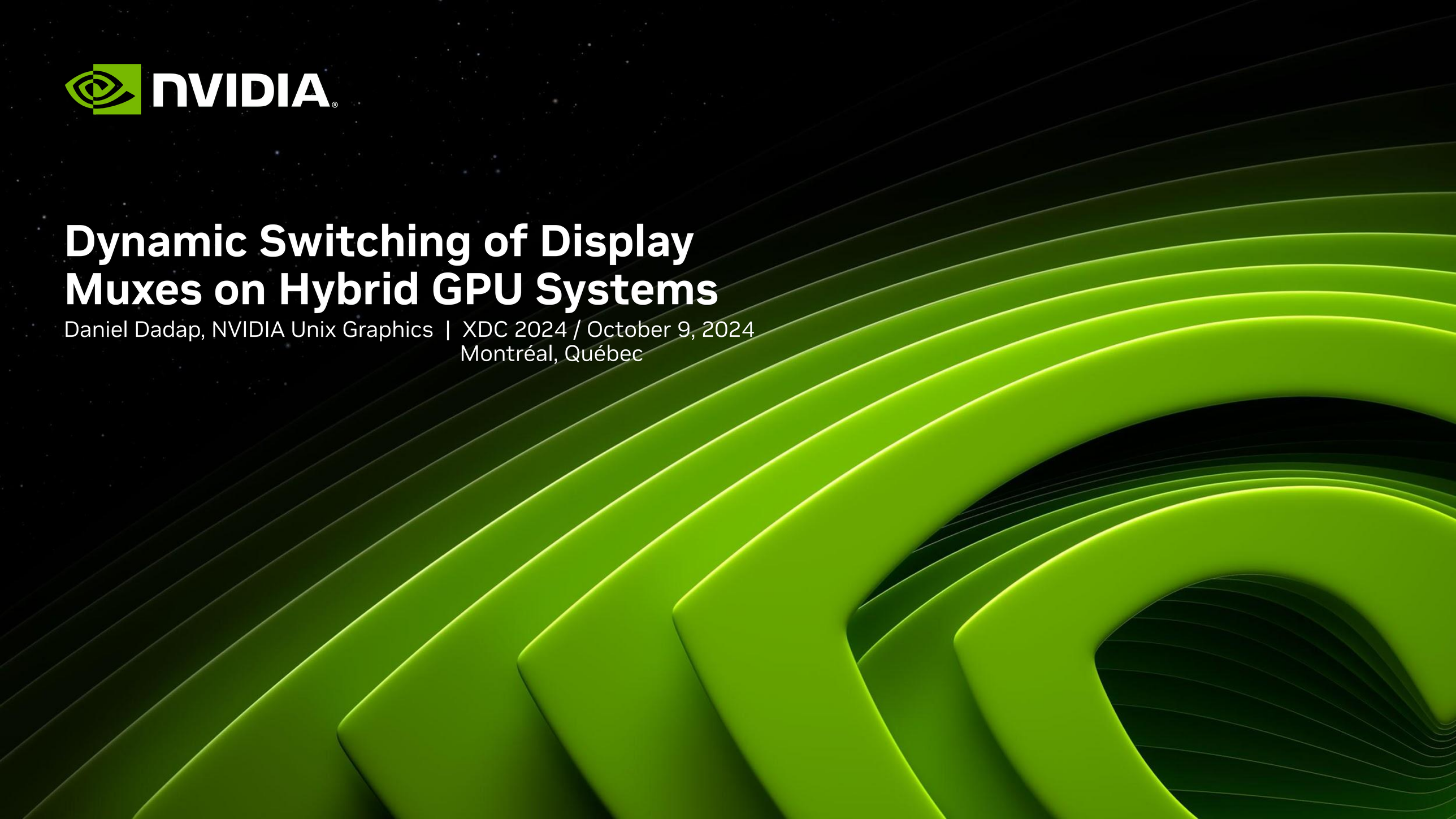


Dynamic Switching of Display Muxes on Hybrid GPU Systems

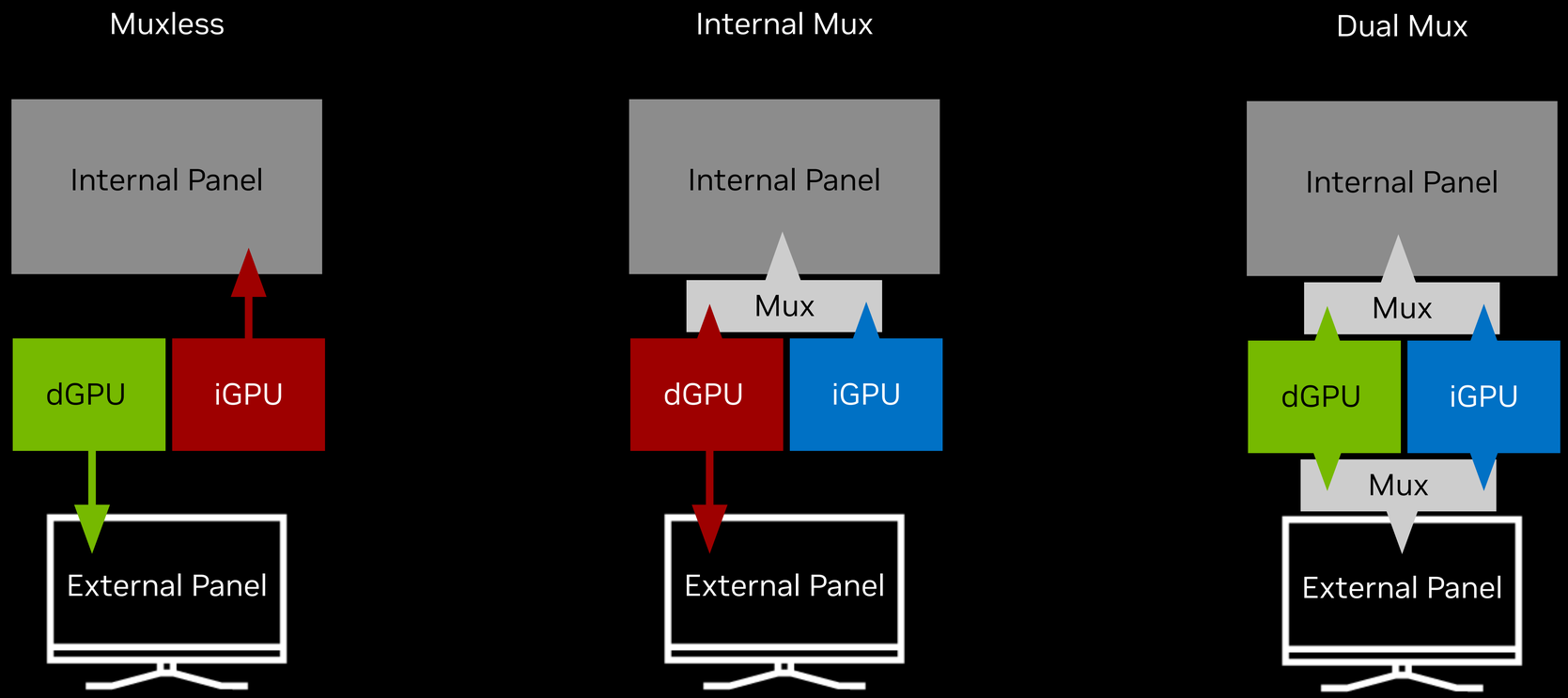
Daniel Dadap, NVIDIA Unix Graphics | XDC 2024 / October 9, 2024
Montréal, Québec

Background

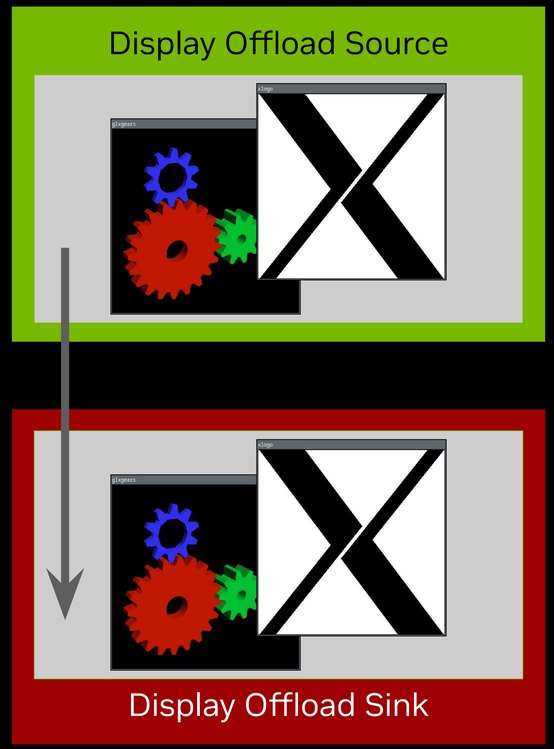
What's a "Dynamic Display Mux"?

- *Mux*: abbreviation for **mu**ltiple**x**er; selects from one of several available inputs to a single output
- *Display* **Mux**: select from one of several available display connectors to drive a display
  - Usually for notebook internal panels, between iGPU and dGPU on a hybrid GPU system
  - External display connectors usually fixed to dGPU output, but may be muxed (or fixed to iGPU) as well
  - Might be selectable at boot time (e.g. via BIOS/UEFI setup screen)
- *Dynamic* **Display Mux**: designed to be switched while the output is actively driven
  - Example: NVIDIA Advanced Optimus
  - Panel Self Refresh (PSR) during transition
  - Low switch time

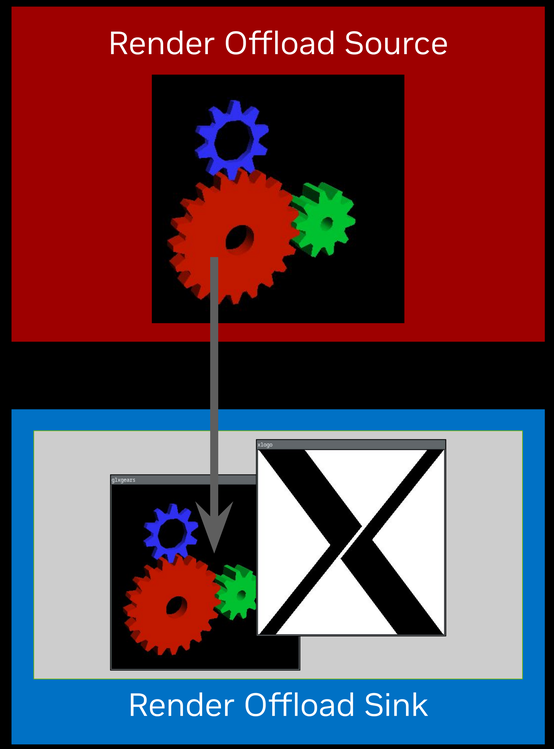
Typical Display Mux Configurations



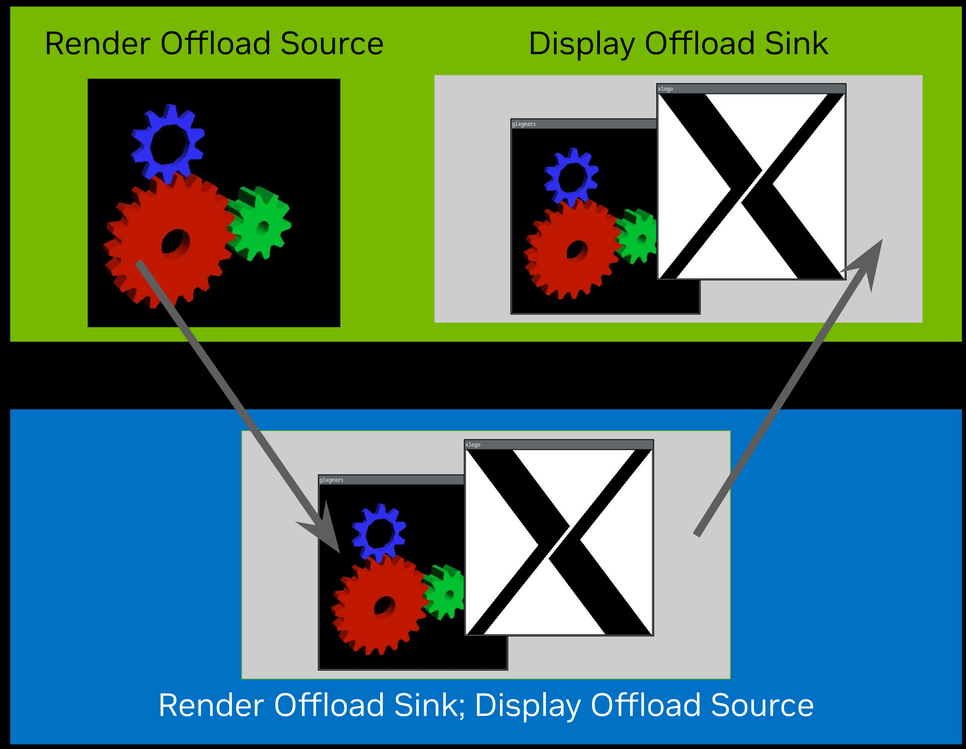
Life Without Mux: Display and Render Offload



Display Offload



Render Offload



Why not both? 😅 😭

Display Mux Benefits

- Core benefit: mux switching avoids the need for display offload
  - Both GPUs always powered even if only one is rendering
  - Continuous fullscreen copy over PCIe bus
- Core benefit possible with non-dynamic switches, but they are inconvenient
  - At a minimum, must end desktop session, lose state, etc.
- Dynamic switch could be initiated several ways:
  - Explicit, user-driven switch
  - Automatic switch for fullscreen applications (e.g., windowed render-offloaded application can unredirect when going fullscreen, bypassing need for both render and display offload)
  - Windowed applications more complicated, but prior art exists (e.g. macOS)
- Dynamic switching requires support from Wayland compositors and from GPU drivers on both sides of switch

Dynamic DIsplay Mux switch sequence of events

Switch triggered
Relevant policy determines that a switch is required

Prepare for switch
GPU drivers on both ends of the switch will need to coordinate to ensure continuity of mode.

DP AUX proxying could be useful to allow the to-be-switched-to GPU's driver to read panel state from the to-be-switched-from GPU.

Enable PSR
Panel Self Refresh maintains continuity of the current frame during the switch

Switch mux
Mux hardware will have a device-specific minimum duration before the link is ready to use.

Respond to switch
Switched-to GPU will need to, at a minimum, retrain (e)DP link and set a new mode

Disable PSR
Allow the panel to be driven by the newly switched-to GPU

GPU Driver Support

What about vga-switcheroo?

- Infrastructure for Display Mux switching in Linux [1]
  - Introduced in 2010 with Linux v2.6.34
  - Architecturally, “mux handler” driver and “client” (i.e., GPU) drivers register with subsystem; in many cases “mux handler” is owned by kernel module of one of the GPU drivers
  - “Client” drivers implement optional callbacks to be run before/after switch event
  - At the time, X did not support multiple concurrent GL implementations
- Functional challenges:
  - Design assumes switches can’t happen while output is driven
  - Forced “mux-only” switch possible, but it bypasses driver callbacks
  - Only supports a single mux with exactly two inputs, which are assumed to be exactly one iGPU and one dGPU; on existing multi-mux systems, all muxes are driven together as a single logical device
- Userspace API challenges:
  - Single mux, dual output limitations are encoded into the UAPI
  - No natural synchronization points between mux switching and modesetting
  - UAPI is in debugfs, which might be disabled in Kconfig when switcheroo might otherwise be enabled

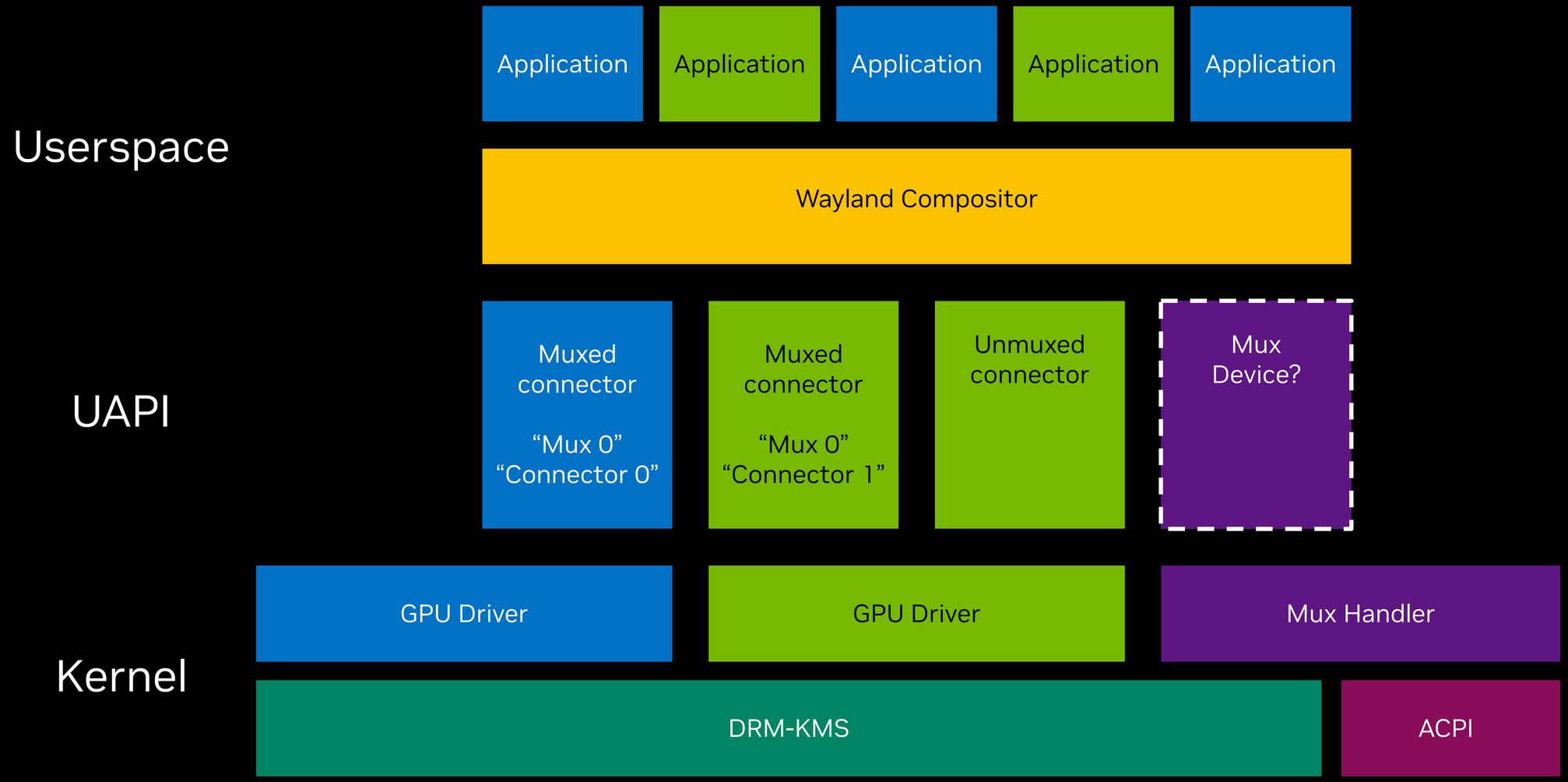
What about DRM-KMS?

- A *Dynamic* Display Mux switch is going to need to be tightly coordinated with whichever DRM-KMS client is driving output, so might as well extend the DRM-KMS API to support Dynamic Display Mux switching
- Independently suggested by NVIDIA (2022, dri-devel mailing list [2]) and AMD (2024, DisplayNext Hackfest [3,4])
  - Some minor differences between the proposals, but there is fundamental agreement that DRM-KMS is a sensible place to drive Dynamic Display Mux switching from
- The mux is ultimately wired to connectors on the source GPUs on the input end, and the sink display on the output end - and we already have `drm_connector` objects!
  - Connectors can advertise properties which expose relationships to muxes on the system
- Extending the DRM-KMS UAPI to support mux switching solves the problems with vga-switcheroo UAPI:
  - No more artificial debugfs dependency: depending on `CONFIG_DRM` is much more sensible
  - Support for *n* muxes and *m* muxed connectors
  - No assumptions about GPU types (e.g. dGPU vs iGPU); we only care about the connectors
  - The mux switch could be performed as part of an atomic modeset that the compositor may already need to do
  - Mux switch naturally synchronizes with events driven by DRM-KMS clients

Open Design Questions: Dynamic Mux in DRM-KMS

- Dedicated Mux class and IOCTL vs. setting connector properties in KMS atomic commit
- Cross-device authentication
  - If mux is addressed explicitly, client needs to prove DRM Master on devices owning connectors on both sides of switch
  - If mux is addressed as part of atomic commit, client needs to prove DRM Master on opposite side device
- Discovery of mux topology
  - Easier to solve for the “internal panel only” case, still need to differentiate which GPU corresponds to which mux state
- Querying attributes on disconnected connector
  - DP AUX proxying?
- Switching a disconnected (external) mux
  - Probably not a problem if there is a dedicated mux class with its own device node

DRM Mux Switching Architecture



Compositor Support

What Will Compositors Need to Do?

- Wayland compositors are the natural owner for mux-switching policy
  - Compositor has full view of all windows from all GPUs
  - Compositor has context to help decide “tricky” situations, e.g. alt-tab switcher display, desktop notification popups
- Compositors will need more robust multi-GPU support
  - Ability to directly scan out from any GPU - this has utility beyond mux applications, recall muxless systems where different displays are permanently connected to disparate GPUs; see prototype implementation of secondary GPU direct scanout [5]
  - More advanced functionality could depend upon ability to do rendered composition on any GPU, e.g., desktop notification during directly scanned out fullscreen app, don’t switch mux when going back to redirection
  - Supporting rendered composition on multiple GPUs would also avoid need for round trips of render + display offload regardless of mux: just always do rendered composition on whichever GPU is driving each display
  - Could be helpful to track buffer residency in dmabuf protocol, to make it easier to make policy decisions about when to switch; see proposal [6]

Recommended reading

[1] https://docs.kernel.org/gpu/vga-switcheroo.html

[2] https://lists.freedesktop.org/archives/dri-devel/2022-November/379107.html

[3] https://cloud.igalia.com/s/7Z5xdAQiN5pz6GC

[4] https://hackmd.io/TjzGMsOTSSeBSO0f4-BOqA#Display-Mux

[5] https://gitlab.freedesktop.org/wlroots/wlroots/-/merge_requests/4055

[6] https://gitlab.freedesktop.org/wayland/wayland-protocols/-/merge_requests/268


NVIDIA