Improving the current state of the DRM in-kernel client ecosystem

NVIDIA Linux Graphics Team

Brief overview of in-kernel DRM clients

Kernelspace content-blitting applications

- fbcon - Framebuffer consoles, also known as text consoles running on a framebuffer device
- bootsplash - WIP[1]
  - There is a proposal to work on building kernelspace bootsplash support using the DRM client infrastructure
- drm_panic - A Blue Screen of Death equivalent for Linux
  - It is an example of a related application in kernelspace, but it does not make use of the in-kernel DRM client infrastructure

Adopters supporting these in-kernel applications

Some quick statistics

- **57** DRM drivers implement fbdev emulation support
  - `rg -l drm_client_setup -g '!include/**' | xargs -I {} dirname {} | uniq | wc -l`
- while only **14** implement drm_panic support
  - `rg -l '\.get_scanout_buffer' -g '!**/drm_panic.c' | xargs -I {} dirname {} | uniq | wc -l`

- Needing to implement different APIs to support functionality adds both development and coverage testing costs if this needs to be done per-application
  - The ideal scenario would be the case where all DRM in-kernel client applications can share the same API
  - To understand how we can get there, we should take a look at the existing APIs from implementing DRM fbdev emulation support and drm_panic support

Setting up fbdev emulation support

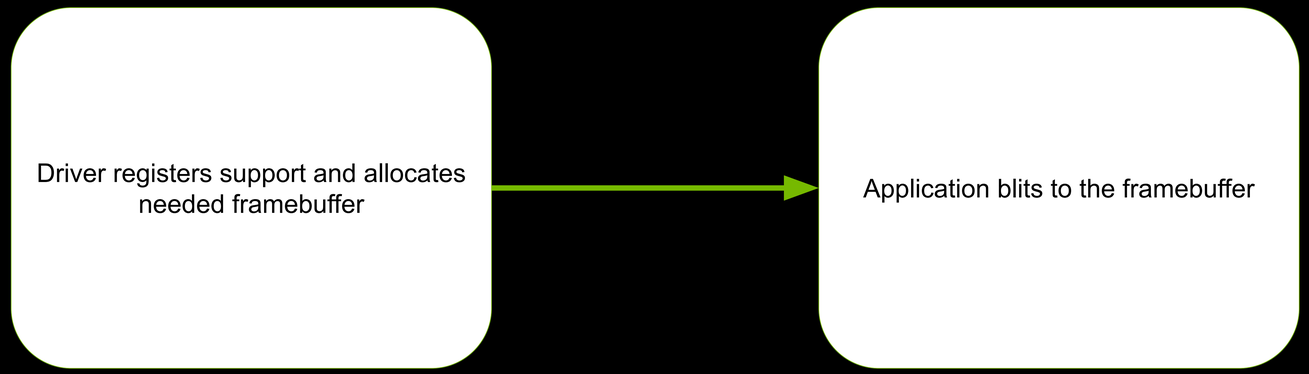
Perspective from device drivers

1. A DRM driver must implement .fbdev_probe for DRM fbdev emulation support. This callback allocates a buffer that the kernel can blit content to for presenting to the display based on the surface height, width, and fourcc pixel format

Setting up fbdev emulation support

Perspective from device drivers

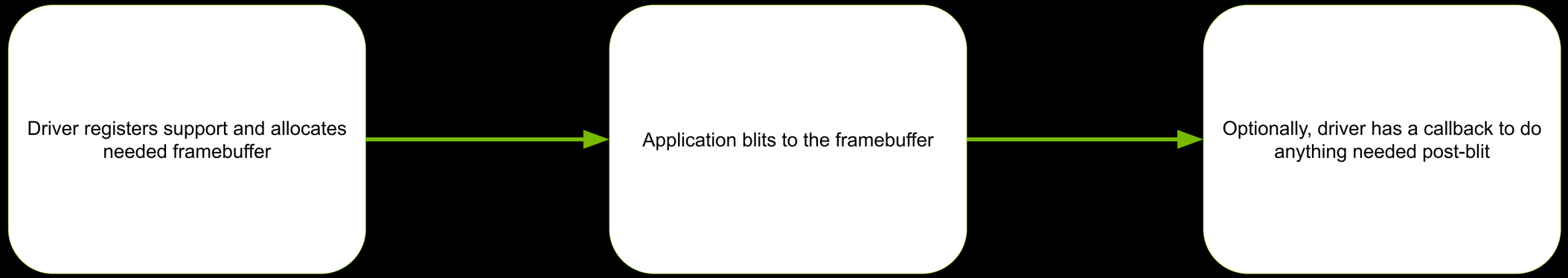
2. fbcon content is blitted to the fbdev-backed framebuffer



Setting up fbdev emulation support

Perspective from device drivers

3. An optional callback called after the framebuffer memory provided to the DRM core stack is updated with the console contents
   - This callback can be used for shadow buffering and performing real blits after some pre-processing. drm_fbdev_ttm_helper_fb_dirty provides a simple example



Setting up drm_panic support

Perspective from device drivers

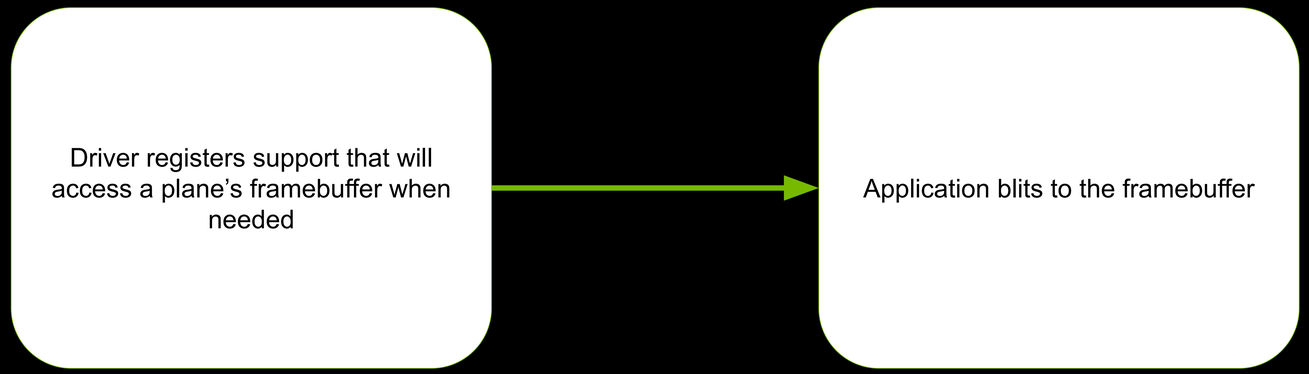
1. For a driver to register support for DRM panic, it first must call drm_dev_register AND in the call to drm_panic_register, the core stack loops through all planes of a DRM graphics device to see if the driver has set up a drm_plane_helper_funcs.get_scanout_buffer for the planes

Driver registers support that will access a plane's framebuffer when needed

Setting up drm_panic support

Perspective from device drivers

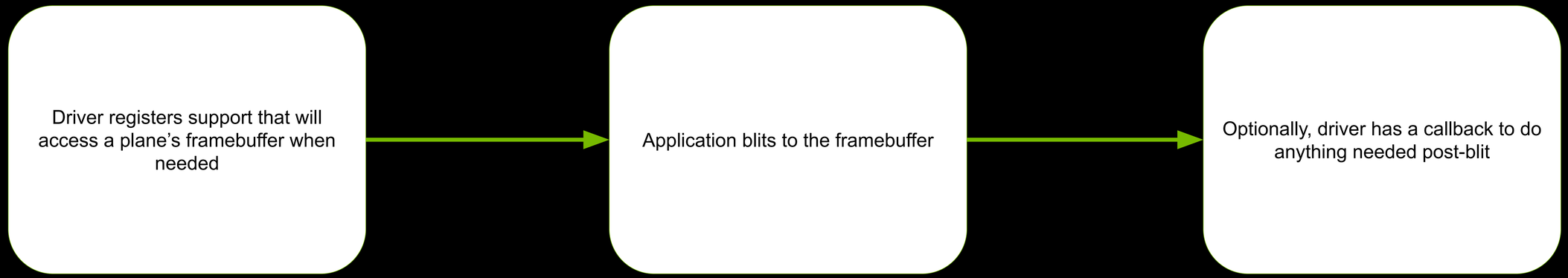
2. drm_plane_helper_funcs.get_scanout_buffer is called for all planes the callback is implemented on. This function provides a 'struct drm_scanout_buffer' that functions can use to blit to the display. A good example of this would be draw_panic_static_user in drivers/gpu/drm/drm_panic.c



Setting up drm_panic support

Perspective from device drivers

3. Optionally, a driver can implement a panic_flush callback that is called after core DRM writes content to the scanout buffer. This can be useful if a driver needs to do extra operations to make the buffer visible on the display



Comparing the two in-kernel DRM clients

A look at their data structures

- The structs that represent the scanout buffers between the two stacks are also very similar, 'struct drm_scanout_buffer' and 'struct drm_client_buffer'
- Similar to drm_client_buffer, it would be better for drm_scanout_buffer to build its structure on top of drm_framebuffer since it mostly reuses components found in that base structure
- The remaining information like the iosys_map can be common among the two structures
- Unifying the buffer representation is the first step to converging these two in-kernel applications

Comparing the two in-kernel DRM clients

Conclusions

- The flow between DRM panic and DRM fbdev emulation support is very similar
  - Write in-kernel application content to a scanout buffer. Have a callback that enables driver post-blit handling, enabling possibilities for shadow buffers, etc
- Similar to the buffer data structures, the framebuffer setup and post-blit handling flows should also be unified

Unifying the APIs

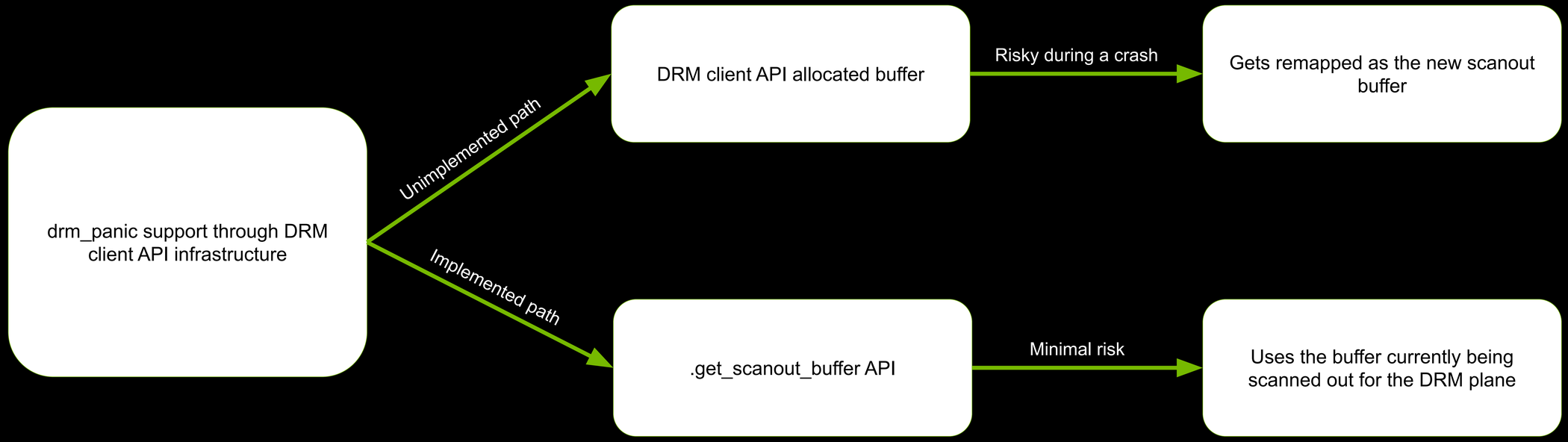
That is the goal of the drm_client API

- “drm: Provide client setup helper and convert drivers”[2]
  - This series aimed to be a building block for in-kernel applications
  - “Patch 4 adds drm_client_setup(), a client-agnostic interface to initialize the in-kernel DRM clients. It only supports the new fbdev emulation setup, but additional clients will be added here. Hopefully this will hide future changes to DRM client initialization from drivers.”
- Hopefully, unifying the similar paths between drm_panic and the common DRM client API paths will improve drm_panic adoption while minimizing the number of APIs a driver has to implement
- Needing separate driver APIs for future in-kernel applications will not scale as we add functionality ranging from bootsplash support to in-kernel tetris

Unifying the APIs

That is the goal of the drm_client API

- **NOTE:** The decision to not use DRM client infrastructure was intentional for drm_panic in order to reuse the currently presented framebuffer instead of mapping a new one[3]
  - DRM client API could probably add another callback / incorporate a “.get_scanout_buffer” equivalent for “critical” in-kernel client applications



We would like your feedback!

- Maybe you have some visions for this unification effort?
- Or some questions regarding our development roadmap?
- Please reach out!
  - Four of us are here at XDC
  - NVIDIA Forums
  - Feel free to email me with regards to development discussions
    - Rahul Rameshbabu <rrameshbabu@nvidia.com>

References

- [1] bootsplash WIP status
  - https://docs.kernel.org/gpu/todo.html#bootsplash
- [2] “drm: Provide client setup helper and convert drivers”
  - https://lore.kernel.org/dri-devel/20240924071734.98201-1-tzimmermann@suse.de/
- [3] drm_panic changing from using the DRM client API to drm_scanout_buffer
  - https://lore.kernel.org/dri-devel/9b232cab-057c-bb42-48cb-f83da3f0e938@suse.de/


NVIDIA