

GPU Job Scheduling: Current Development

Daniel Almeida (Collabora)
Philipp Stanner (Red Hat)

What's the GPU Scheduler?

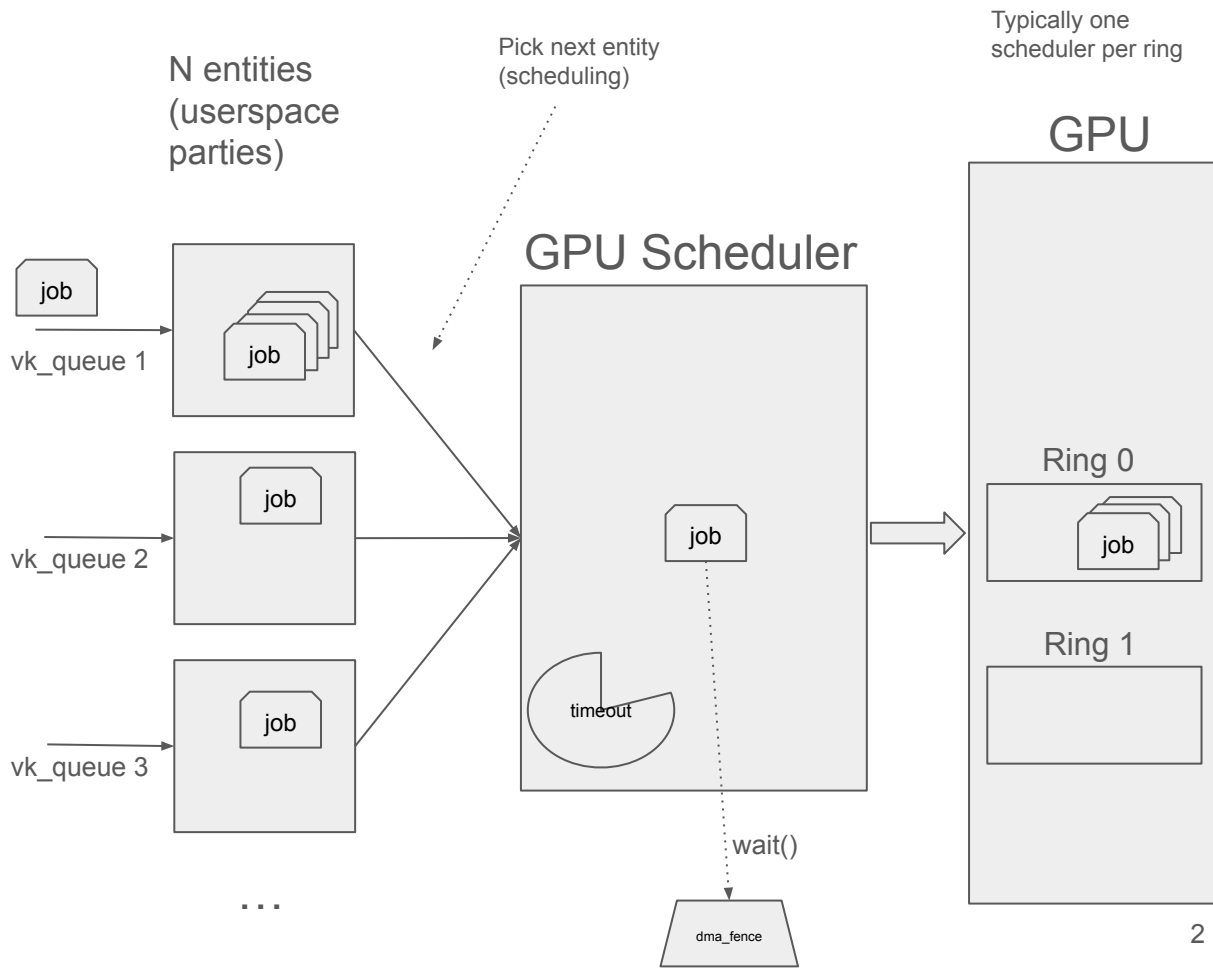
Load-Balancer: GPU doesn't have infinite capacity for jobs (graphics command buffers).

Dependency-Manager: runs jobs in (per-entity) submission order, but waits until a job's (external) dependencies are fulfilled (e.g., a VM_EXEC job only runs after an associated VM_BIND is finished).

"Scheduler" because it schedules between entities (job containers), each of which can correspond to a userspace counterpart. Can't interrupt jobs like a CPU scheduler can.

Timeout handler: Guarantees forward progress in case a job takes too long / forever.

Depicted is the classic usecase, relevant e.g. for amdgpu (steam deck)



drm_sched codebase is in bad shape

“My God, it’s full of race conditions!”
– 2024: A Kernel Odyssey

- In general, scheduler issues are / were:
 - Race conditions
 - Inconsistent (and even missing) locking
 - dma_fence signalling issues
 - Refcounting issues
 - Legacy code
 - Missing documentation
 - Often no canonical solutions (job resubmissions on GPU hang)
 - Unclear / unenforced lifetimes of scheduler objects
- In case you’re searching for work: Code base contains many FIXMEs :)

Non-Scheduling “Scheduler”

Many GPUs don't have *hardware rings* anymore, but an arbitrary number of *firmware rings*.

Scheduler now is not a scheduler anymore (for many drivers).

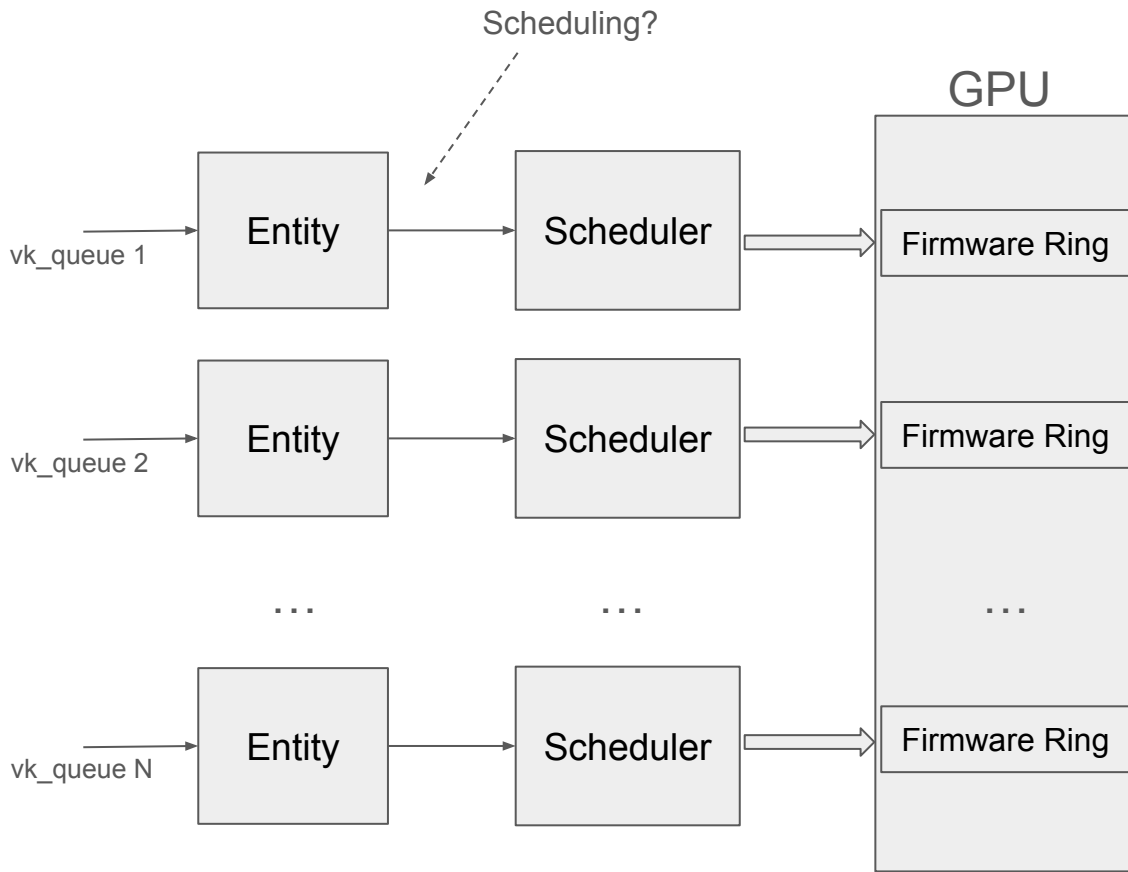
Consequence:

Linux kernel has almost no control anymore over task prioritization.

Future??

Preemptible firmware rings? Priority levels per ring through firmware?

New component: `drm_dep`?



drm::JobQueue so far:

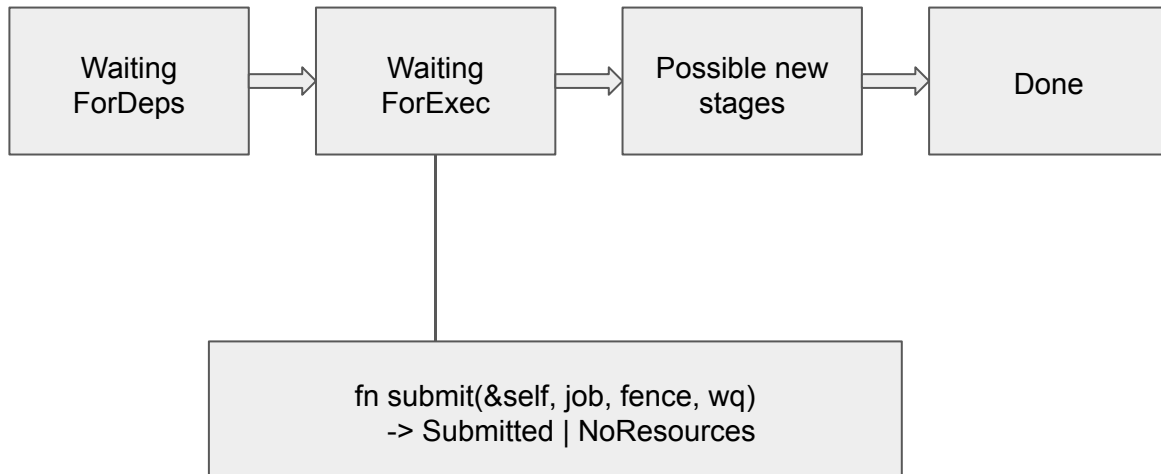
Not a scheduler: it tracks dependencies and lifecycle. The firmware decides what runs.

One XArray: every job lives in it. Each stage owns a cursor, so a stage transition is an $O(1)$ advance with no data copies.

One callback: a driver implements `submit()`. New stages may be easily implemented.

Teardown drains: dropping the queue signals every fence and guarantees no callback fires afterwards.

jobs live in one XArray — a stage transition is an $O(1)$ cursor advance



the only callback a driver must write

One queue, one ring

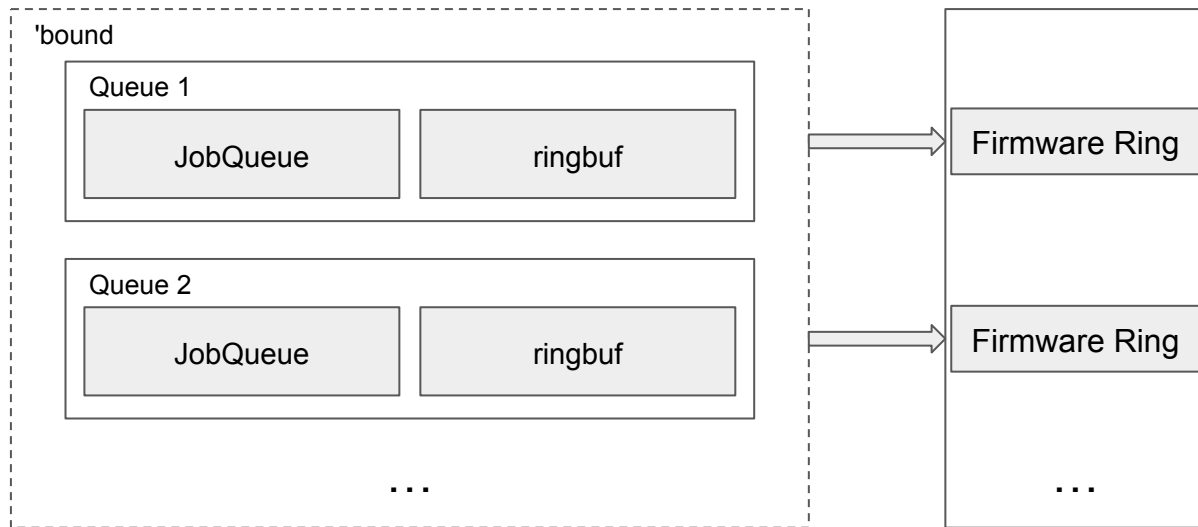
Colocated: each `vk_queue` gets its own `JobQueue` and its own ring, inside one `Queue`.

Few responsibilities: resolve dependencies, notify the driver when job is eligible to run.

Extensible: another component can implement policies on top of the `JobQueue`.

With 'bound: containment becomes enforced by the type system. A ring cannot outlive its `JobQueue`, and neither outlives the bound device.

Open questions: Is the “stage” model worth it? What data structure to use?



one `vk_queue` → one `Queue` → one `JobQueue` → one ring

a job outliving its `JobQueue`: does not compile