XDC 2021

i915 TTM conversion

Thomas Hellström

intel®

Thomas Hellström - Who am I?

- Open-source graphics on spare-time starting 2003 (VIA Unichrome) XFree86, Xorg, Xine, DRM.
- Tungsten Graphics 2006 – 2008, Mesa, DRM Xorg, Consulting
- Vmware Inc. 2008 – 2020 Mesa, DRM, Xorg.
- Intel 2020 – DRM

Outline

- What is TTM?
- Why is i915 adapting TTM?
- Where are we introducing TTM and why?
- Where are we today
- What can i915 contribute back? Wishlist?
- Q & A

What is TTM?

Features and history

What is TTM? (Translation Table Maps)

- A set of utilities to manage buffer objects and their placements.
- Was originally developed to support dynamic AGP binding.
- VRAM management was added as an afterthought.
- Handles hybrid trylock evictions, command submission WW locking and memory caching.
- Utilities for asynchronous evictions.

What is TTM Ct'd

- Pioneered a lot of DRM components and functionality that Linux GPU drivers use today
  - Buffer objects.
  - Deadlock-free Wait-Die-like locking and fine-grained locking around LRU lists
  - Persistent memory mappings.
  - Synchronization objects.
  - Asynchronous or migration.
- Old, badly documented code.
- A lot of additions and clean-ups over the years. (Redhat, AMD).
- Currently maintained by Christian König, AMD.

Why is intel adapting TTM?

What else than discrete graphics?

Why is i915 adapting TTM?

- i915 was originally implementing its own VRAM / LMEM management on top of code for integrated.
- Except async migration supported mostly everything TTM supports today.
- Diverging from DRM where most if not all VRAM capable drivers used TTM.
- Upstream pressure.
- Simplifies many things like implementing asynchronous VRAM management and eviction. Well tested.
- Gives us a chance to contribute back to and take advantage of upstream development.

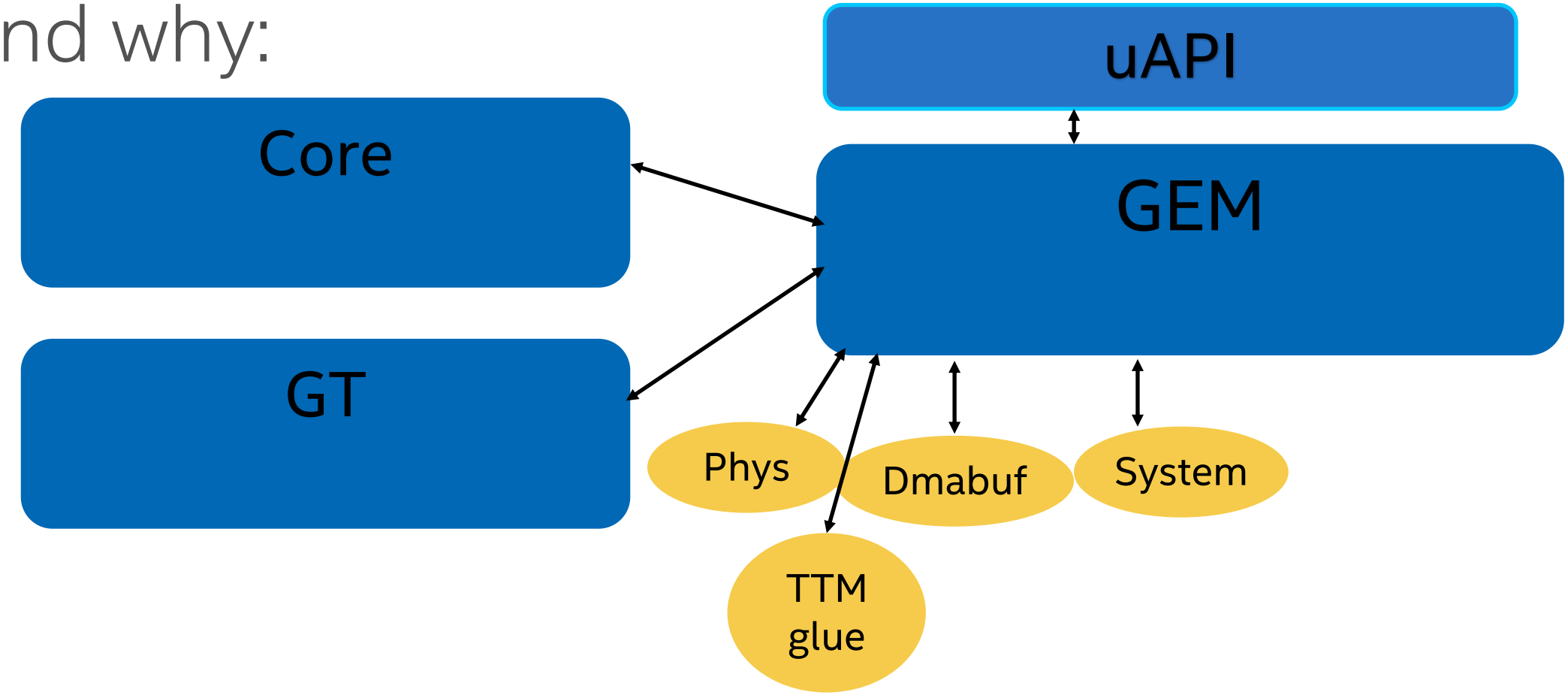
Curiosity

- i915 was once the main target driver for TTM.

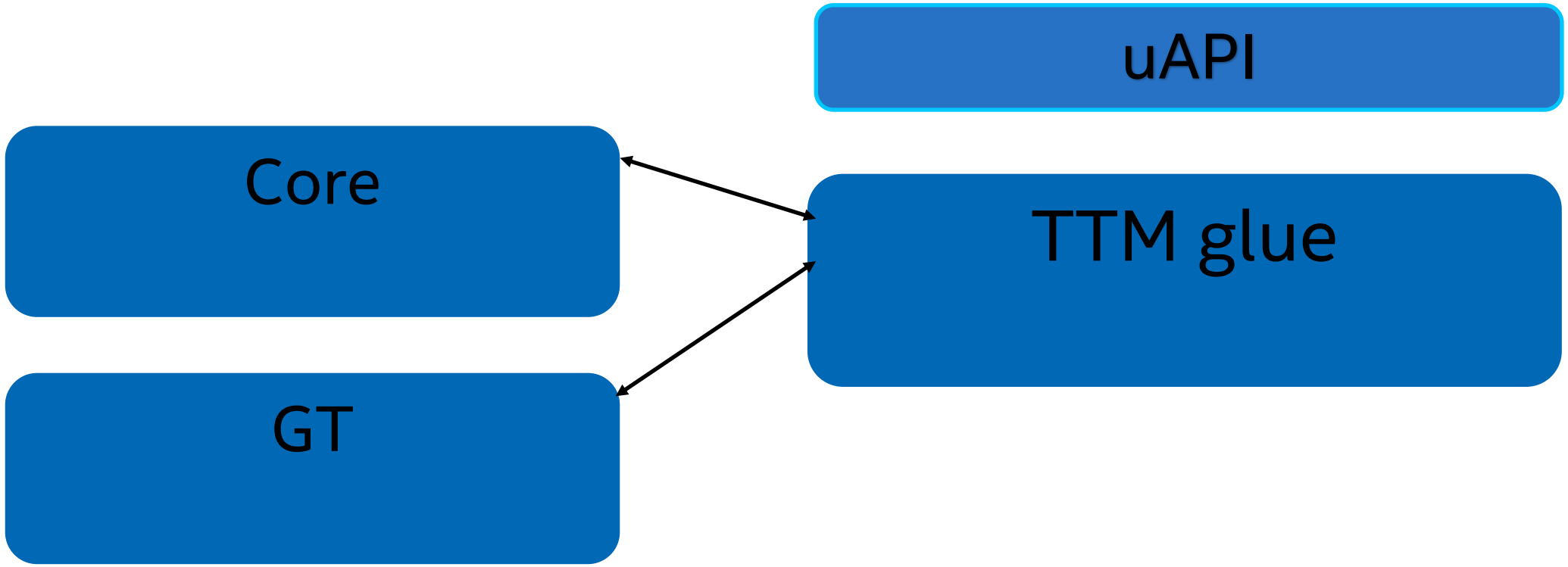
Where to introduce TTM and why?

Driver structure and TTM for integrated graphics?

i915 driver structure: Where we hook up TTM and why:



i915 driver structure: Where we aim to end up



Important uAPI changes for discrete GPUs

- CPU-maps (mmap) will use the TTM-provided address space offset.
- Caching mode is dictated by the i915 kernel driver.
- User-space gets to choose the memory region placement(s) of a gem object.
- set_caching gone and set_domain mostly gone.
- Relocations gone.
- A bunch of other uAPI clean-ups by Jason Ekstrand

Performance considerations:

- Will i915 TTM-based be better- or worse performing?
- Factors at play:
  - Caching.
  - Page-allocation.
  - Sync waits.

Where are we today?

And where do we want to go?

Where are we today?

- drm-tip i9915 for DG1 fully functional except suspend / resume under review.
- Locking clean-ups, TTM shmem support + shrinking, Fully failsafe async migration and eviction being worked on.
- DG1 support in Mesa main. Desktop and games working, occasional crashes, XWayland rendering problems.
- Need to fix up CI tests for uAPI modifications.
  Need more CI test coverage.

What about TTM for integrated?

- In its simplest form a matter of pointing also integrated to the TTM GEM backend rather than the GEM shmem – and phys backends.
- But what about CPU maps with conflicting caching?

How can we contribute back to TTM?

- CI + bugfixes,
- Reviews, developers, discussions.
- Buddy VRAM / LMEM allocator (TTM resource manager).

Wishlist for the future.

- Full Wound-Wait (cross device ?) locking during eviction.
- Shrinking rather than watermark swapout.

Q & A?


intel