RASTERGRID

Going explicit with your GPU workloads using Vulkan®

Dániel Rákos
Máté Ferenc Nagy-Egri

Who are we?

- SWE consulting and development since 2020
  - Everything from applications to drivers
  - Specialized in middleware and technology enablers
  - Supporters and contributors of open standards
  - Some of our projects:
    - **Vulkan® SC™** ecosystem
    - **Vulkan® Video** specs and tooling

Dániel Rákos - Founder & CEO

- graphics and hardware enthusiast
- co-, co-, … co-creator of **Vulkan®**

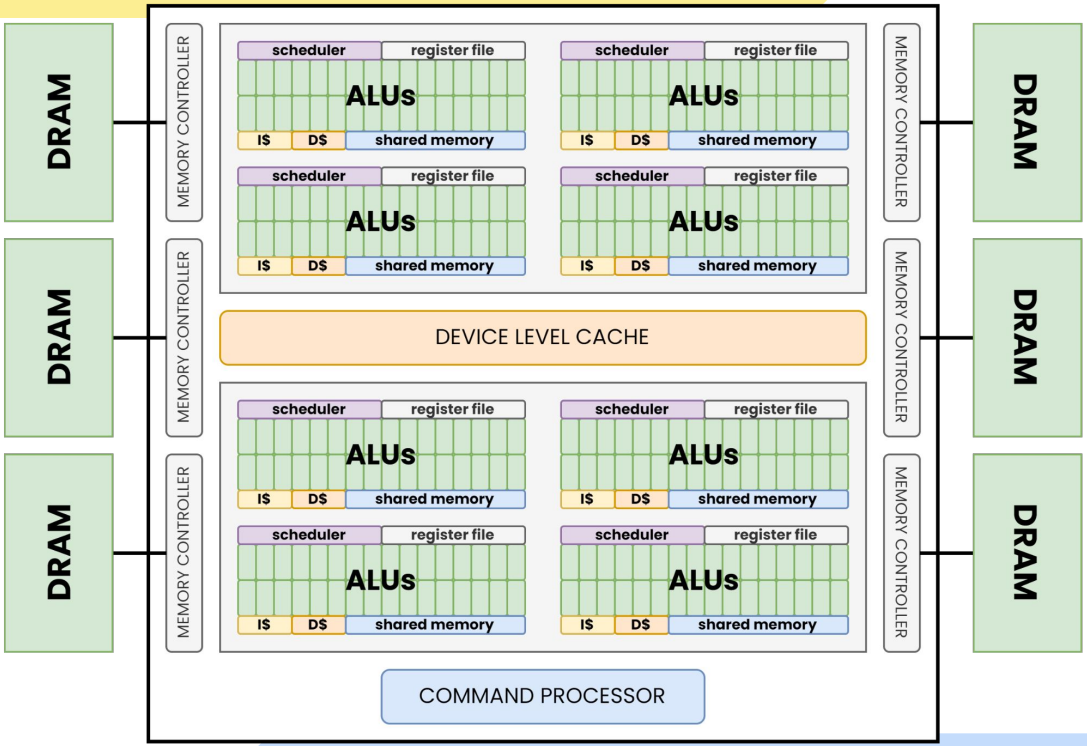
Máté Ferenc Nagy-Egri - Senior SWE

- HPC and scientific compute expert
- Hungarian GPU community builder

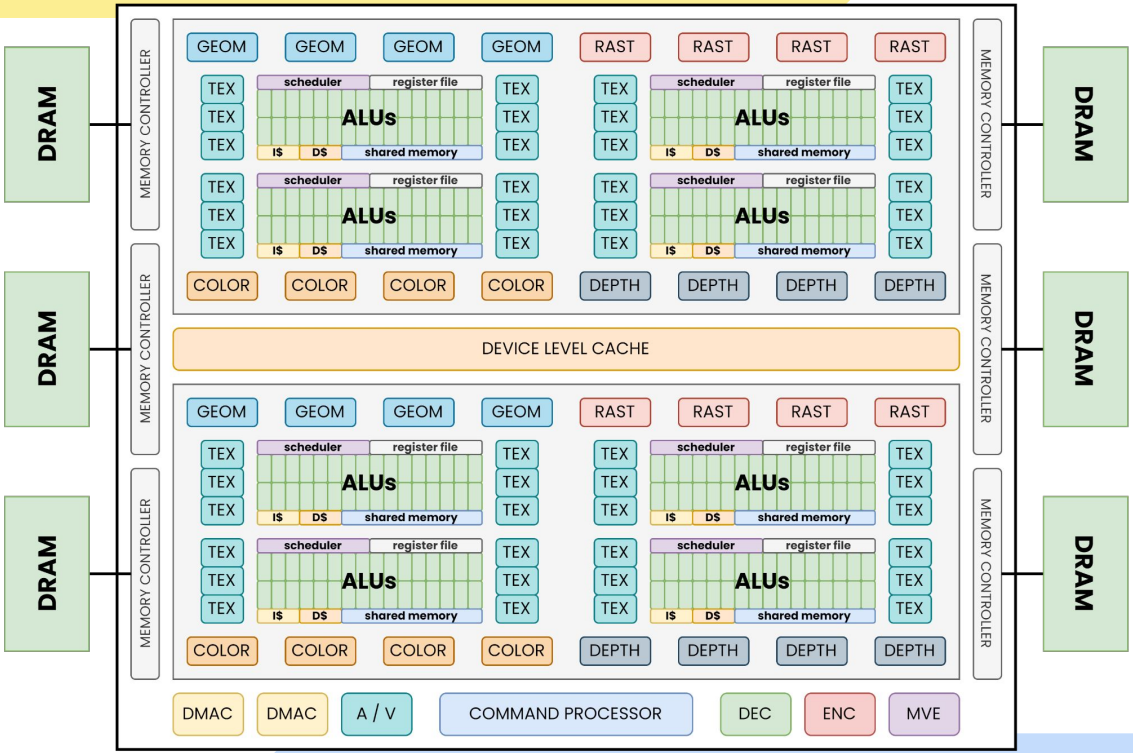
RASTERGRID

Why are we talking about a graphics API here?

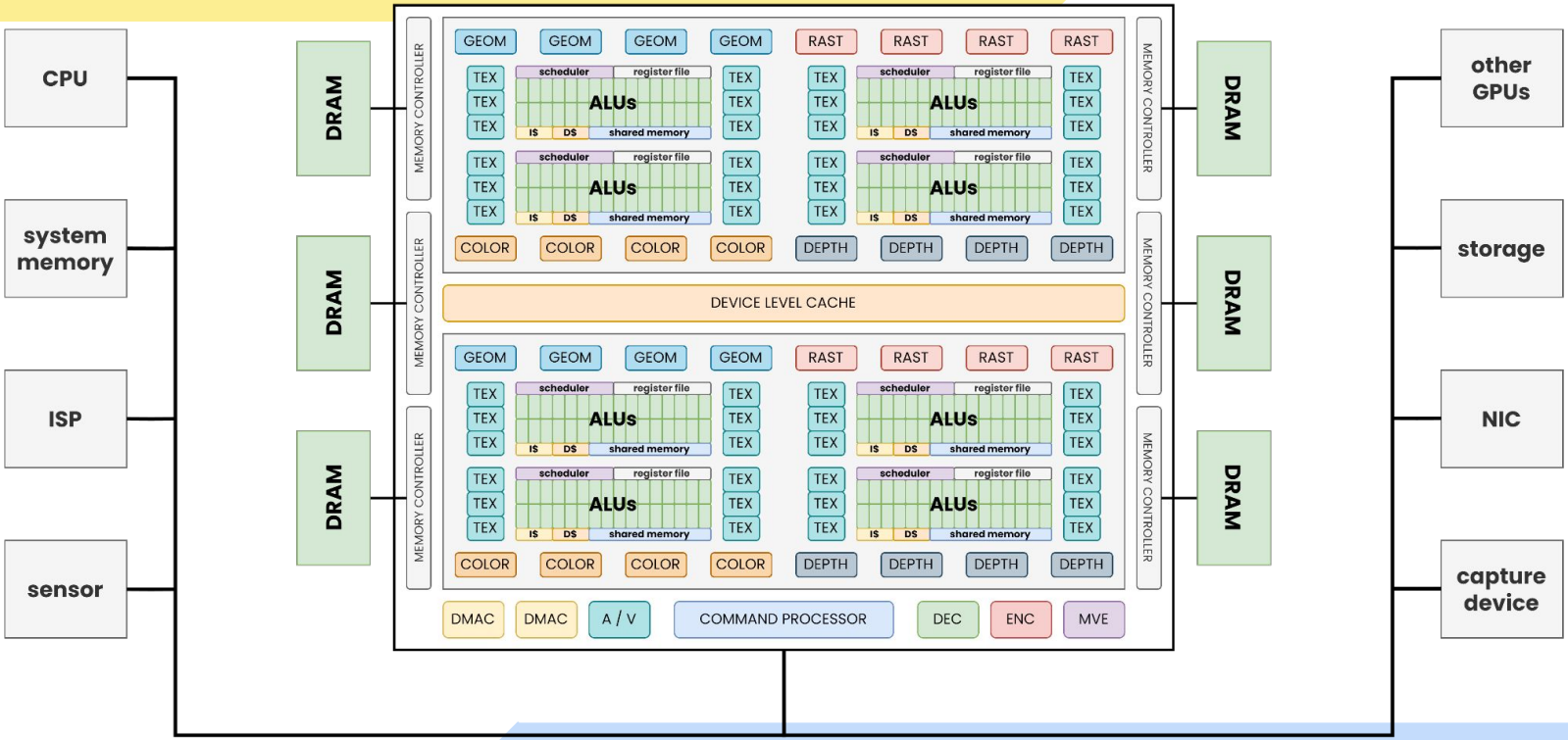
It’s GPU Day after all, right?



It’s GPU Day after all, right?



It’s GPU Day after all, right?

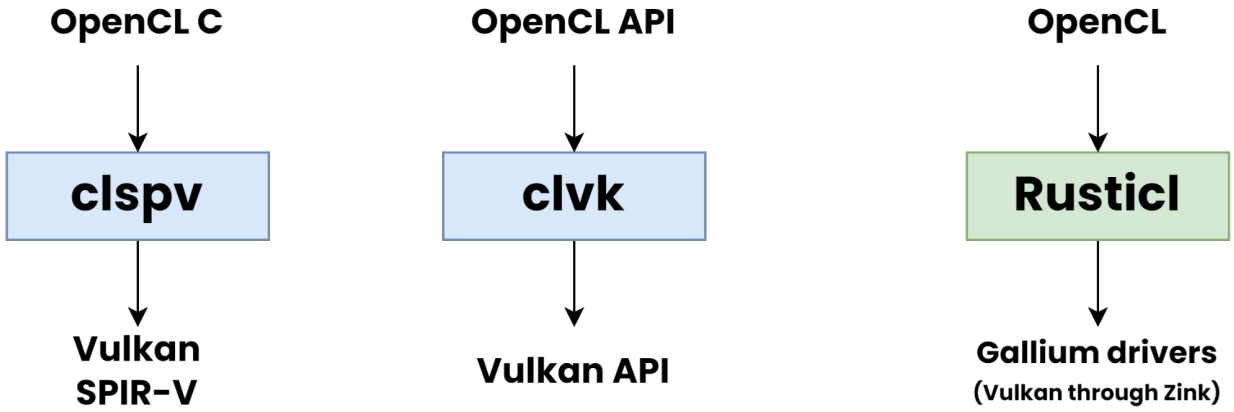


Current compute landscape with Vulkan®

- Vulkan® has compute shaders (kernels) since day zero, but...
  - uses another flavor of SPIR-V with some capability and semantic differences
  - does not have the same precision requirements (graphics can get away with less)

Current compute landscape with Vulkan®

- Vulkan® has compute shaders (kernels) since day zero, but...
  - uses another flavor of SPIR-V with some capability and semantic differences
  - does not have the same precision requirements (graphics can get away with less)



RASTERGRID

Warning: explicit content

Not that kind of explicit

Not that kind of explicit

Not that kind of explicit

So what is an explicit API?

explicit != low level

So what is an explicit API?

explicit != **low level**

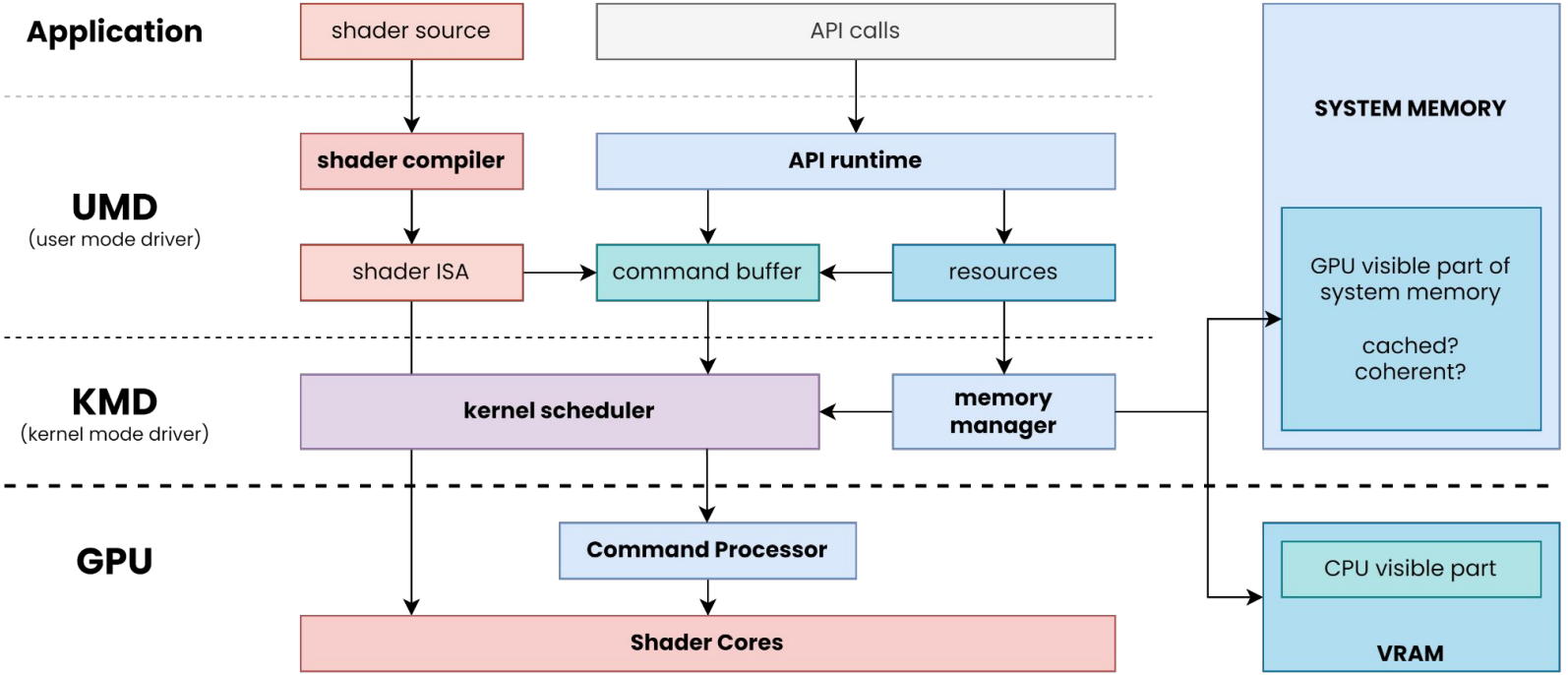
Behavior transparency & predictability

Better application control through expressiveness

RASTERGRID

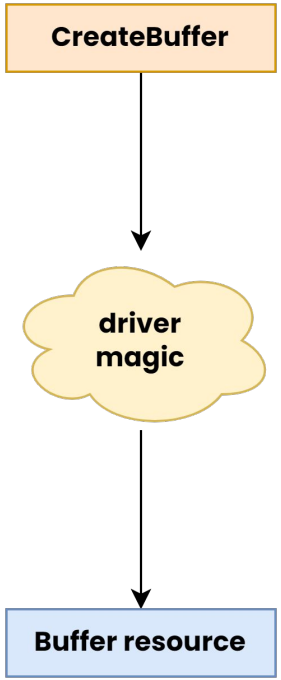
Driver magic vs application control

Basic operation of a GPU stack

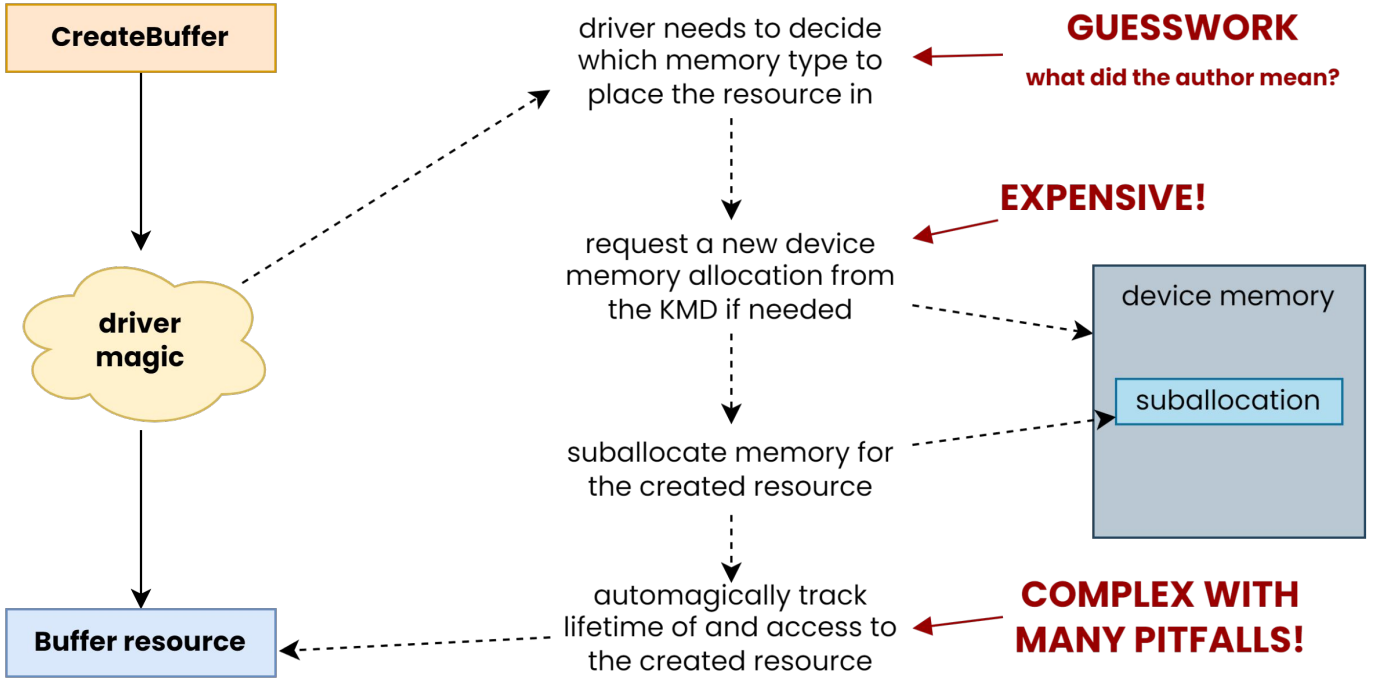


* Read more about memory types of discrete GPUs

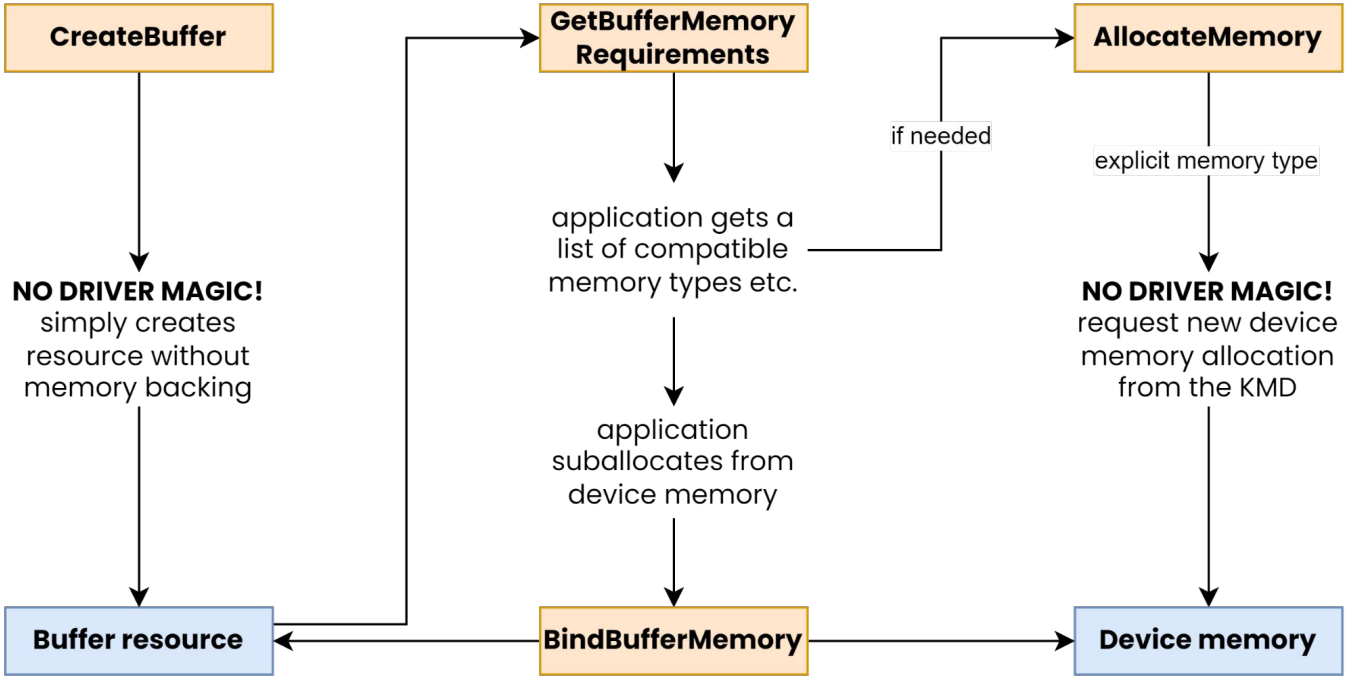
Case study #1:
Buffer allocation



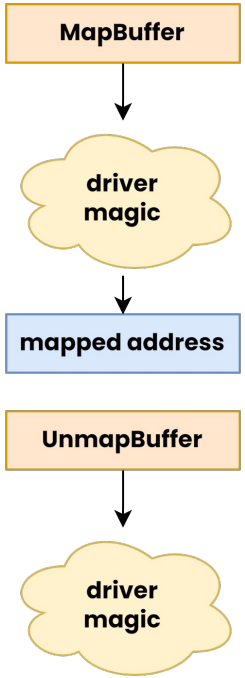
Case study #1:
Buffer allocation



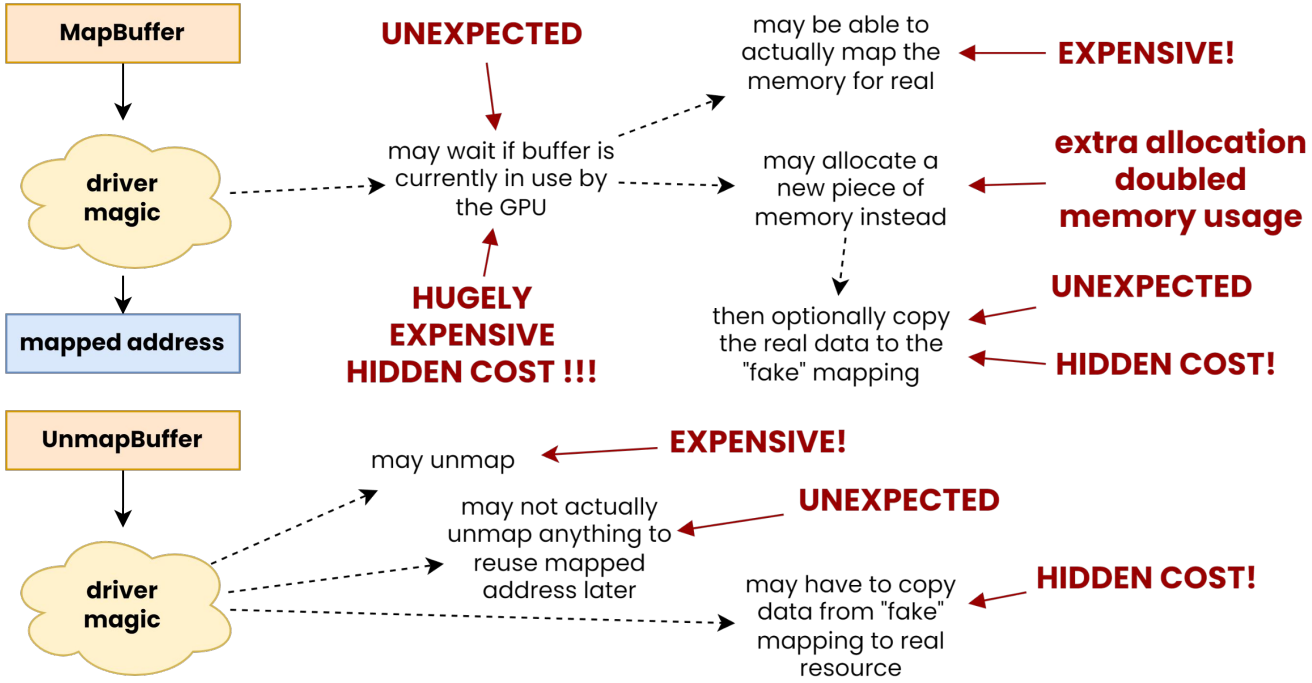
Case study #1:
Buffer allocation



Case study #2:
Buffer mapping



Case study #2:
Buffer mapping



Case study #2:
Buffer mapping

- Vulkan® is explicit about the memory types
  - If it is host-visible then you can map it to CPU address space - **NO DRIVER MAGIC!**
  - Otherwise you cannot map it
- Caching and coherency behavior is explicit
  - No accidental 100x slowdown on CPU reads of uncached data
  - Application decides when to flush/invalidate non-coherent memory
- No implicit synchronization
  - No unexpected GPU or CPU stalls
- Copying is not necessarily bad in all cases
  - It's the only option for non-host-visible VRAM anyway
  - Uploading/downloading "GPU-only" resources on dGPU - async DMA engine
  - But the application chooses when to do a copy and how to manage the memory

Case study #3:
Workload submission

1. Launch kernel
2. Launch kernel
3. Launch kernel
4. Launch kernel
5. Flush
6. Launch kernel
7. Do a copy
8. Flush
9. Launch kernel
10. Launch kernel

Case study #3:
Workload submission

1. Launch kernel
2. Launch kernel
3. Launch kernel
4. Launch kernel
5. Flush
6. Launch kernel
7. Do a copy
8. Flush
9. Launch kernel
10. Launch kernel

driver may submit at this point because it feels the command buffer is large enough

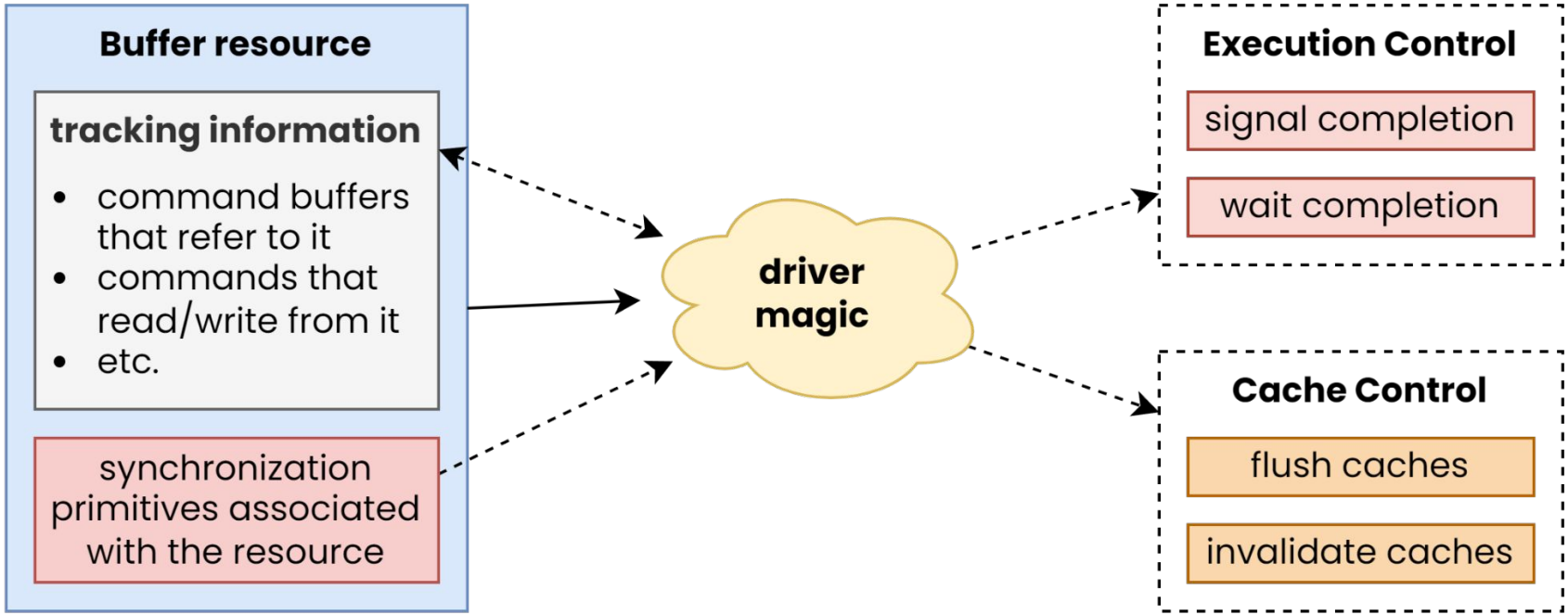
driver may ignore this flush because the workload is too small
KERNEL SUBMISSIONS ARE EXPENSIVE!

large copies are better suited for DMA transfer but involve cross-queue synchronization
DRIVER MAGIC - unclear what happens behind the scenes

Case study #3:
Workload submission

- Command buffers are directly exposed in Vulkan®
  - Application decides how large it makes them
  - … and when it submits them (potentially multiple times as they are reusable)
- Vulkan® exposes HW engines explicitly
  - Separate queues for graphics, compute, transfer (DMA), video coding, etc.
  - Application decides what it submits and where
- Need to be careful about the number of submissions
  - Involves a kernel request and thus it is expensive
  - Prefer to do only a few submissions / frame
- Command buffer size is something the application should tune
  - More commands - lower cumulative submission cost
  - Fewer commands - may be able to achieve lower latency

Case study #4:
Synchronization

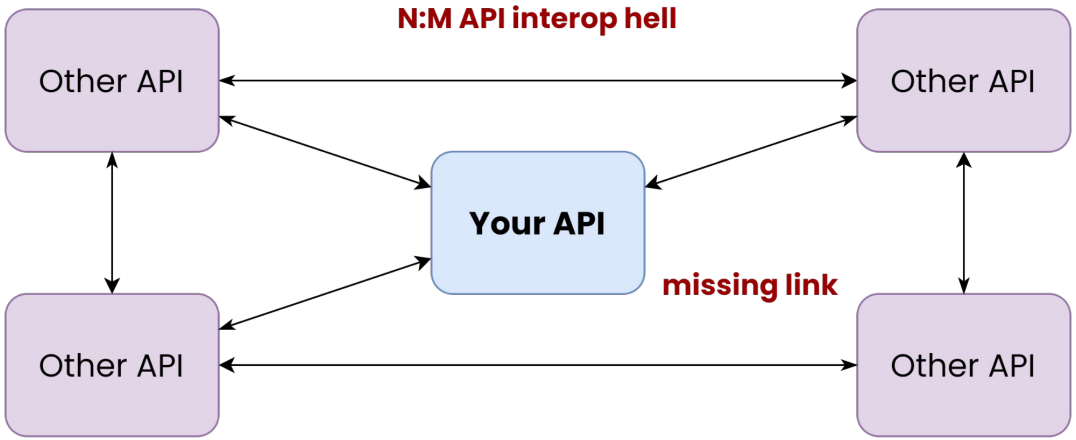


Case study #4:
Synchronization

- Traditional APIs only offer limited control over synchronization
  - There are some controls but there's a lot of implicit magic sync behind the scenes
- Drivers track synchronization requirements with resources
  - Granularity of synchronization is typically tied with them
- Too many things are bundled together into a "resource"
  - **Resource view** - as seen by GPU
  - **Resource storage** - the memory
  - **Execution control** - what to wait for and where
  - **Cache control** - what caches to flush / invalidate and when
- Compute is simple
  - Single-stage pipeline with shader (kernel) read/write access
  - Gets complicated as you have to interop with other workloads

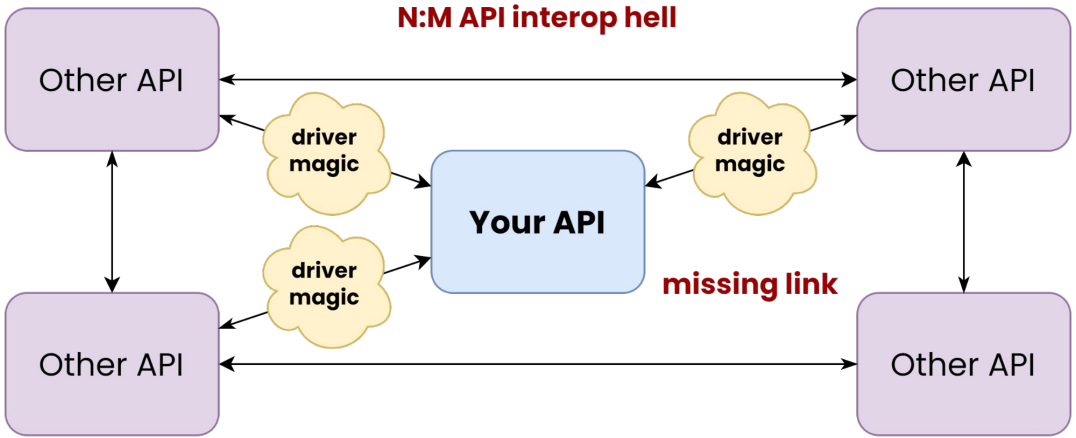
Case study #5:
Interop and resource sharing

- Traditional interop is defined between 2 APIs



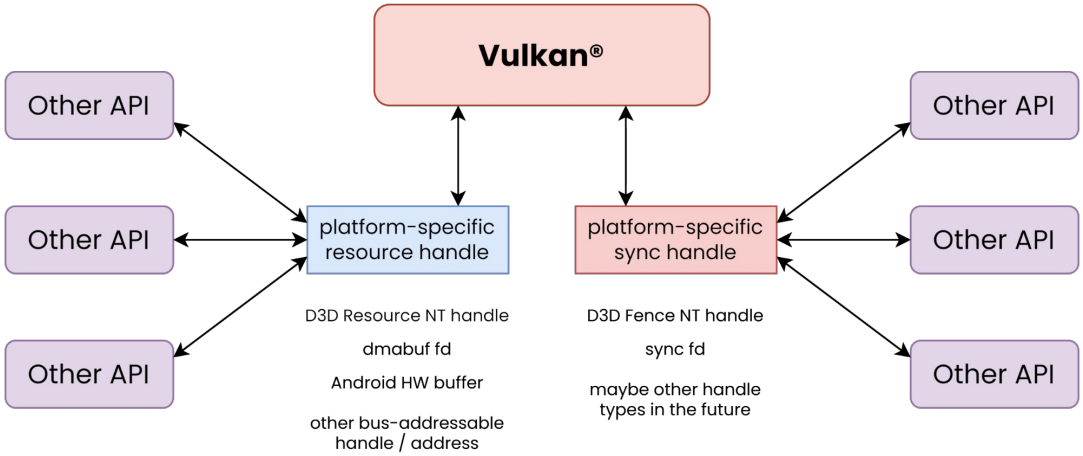
Case study #5:
Interop and resource sharing

- Traditional interop is defined between 2 APIs
- May involve expensive driver magic
  - Additional copies (no “real” sharing) and overly conservative synchronization
  - Often unclear lifetimes and transference semantics



Case study #5:
Interop and resource sharing

- Vulkan® does not try to kid itself
  - External sharing semantics depend on the platform - don't try to hide it
- Avoids interop API hell and enables sharing with a wider set of peers



With great power comes great responsibility

- In general, Vulkan® enables extracting that last 10% of performance
- In some cases no traditional API can even compete with it
  - When you need
    - full control over where, in memory, your resources are placed
    - more control over your memory budget
  - When predictability is of utmost importance
    - no unexpected allocations, copies, synchronization, etc.
- It's not for everybody and not for every job
  - Certainly not the most pleasant prototyping tool
  - It also makes it easier to shoot yourself in the foot
    - even performance-wise
- But if you're a good cook you can make the best meal with it

RASTERGRID

Thank you!
Questions?

read more about such topics on our blog
we're always looking for passionate people