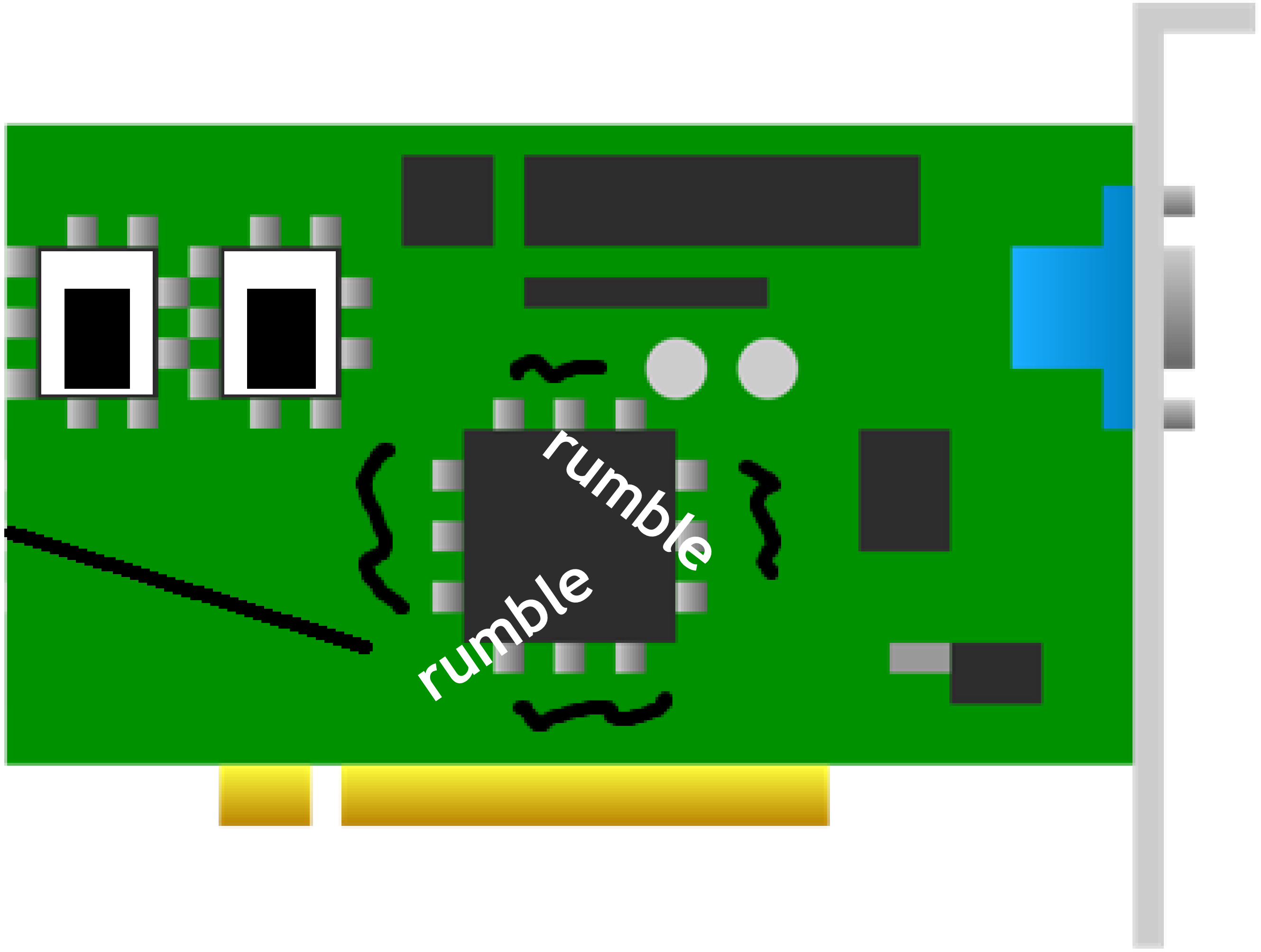
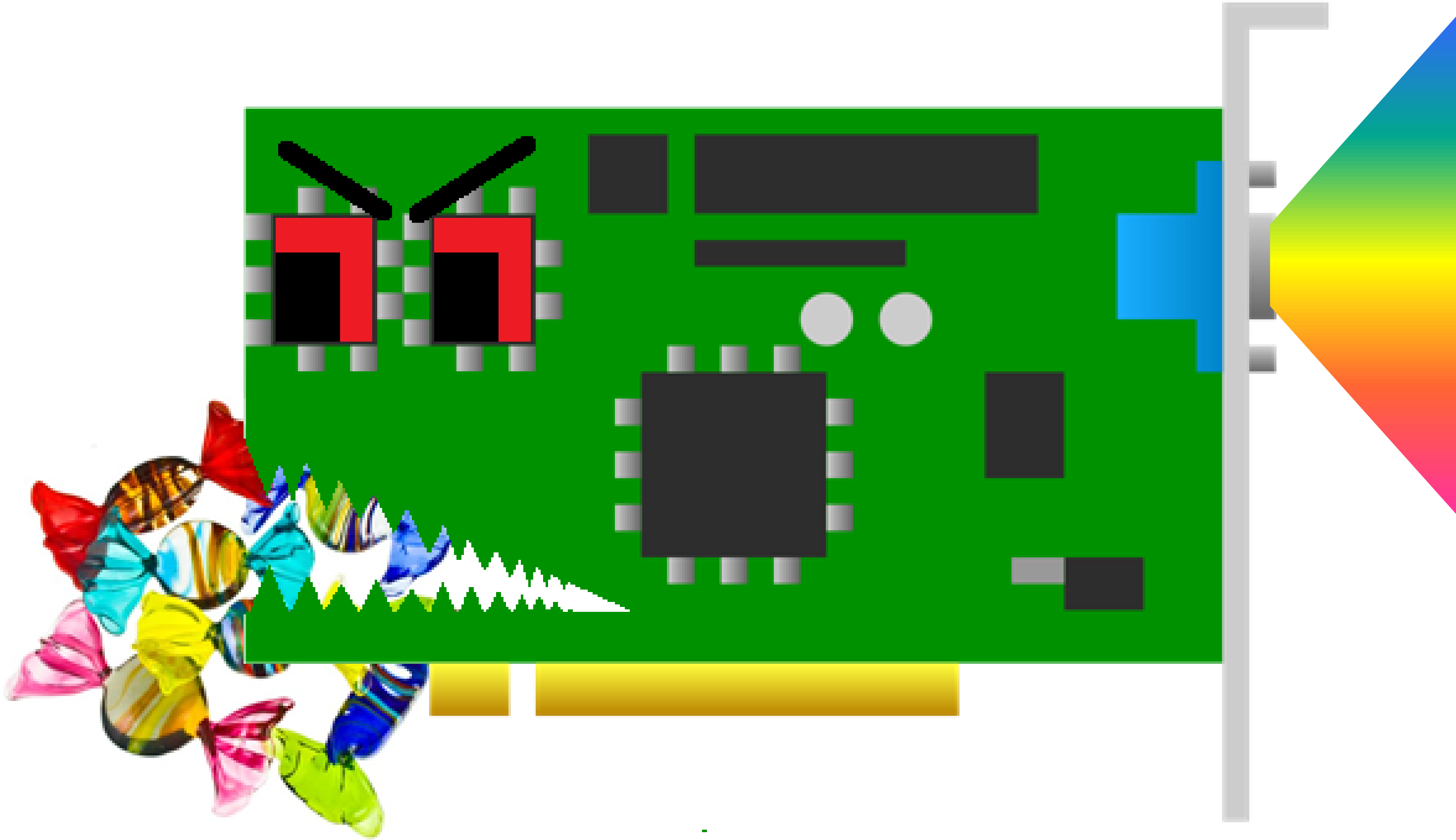
Keeping your GPU fed without getting bitten

Tobias Hector
May 2018


rumble
rumble



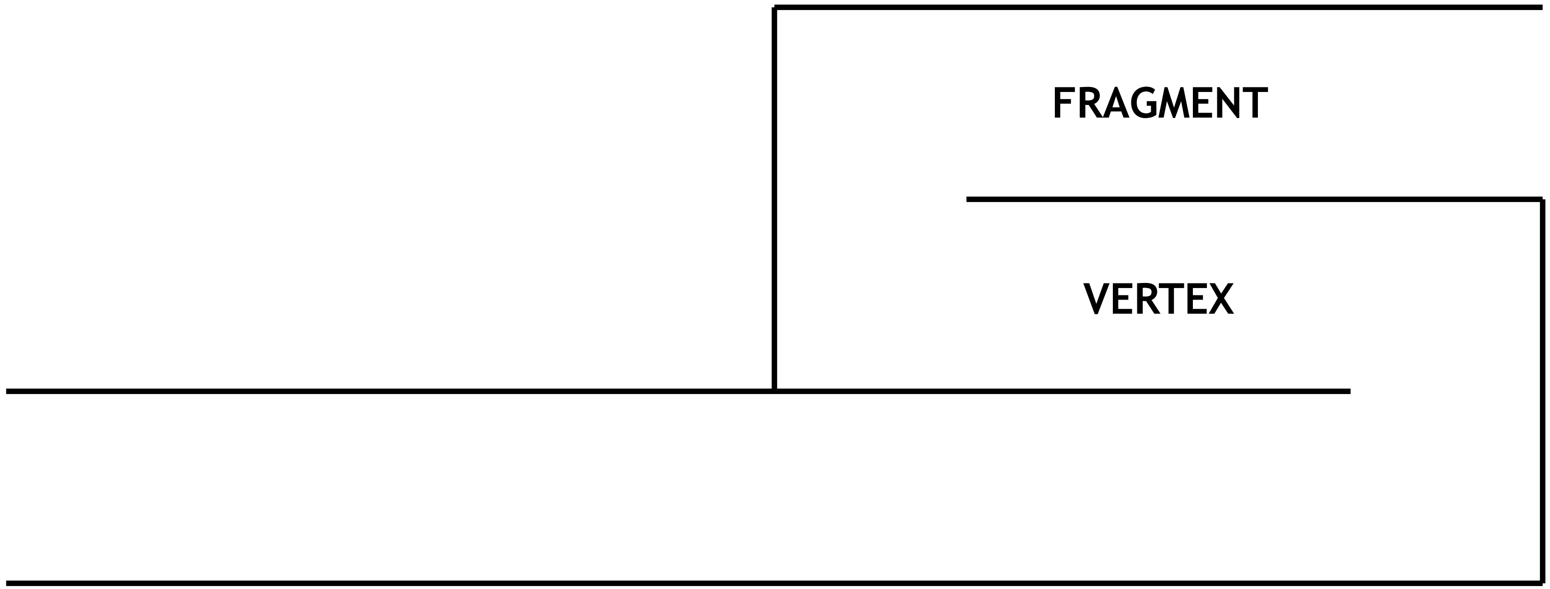
Keeping it fed

- **GPU needs a constant supply of food**
  - It doesn’t want to wait

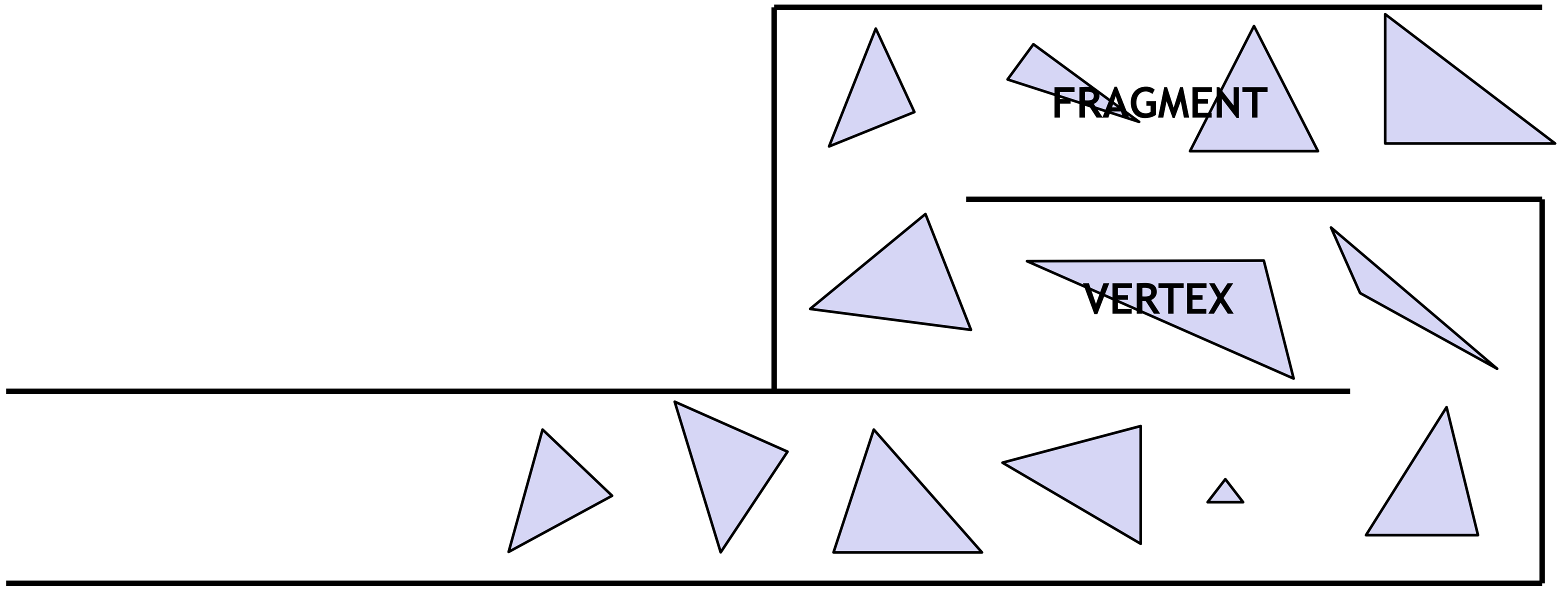
- **Certain foods are tough to digest**
  - Provide multiple operations to hide stalls

- **Your job to give it a balanced diet**
  - A healthy mix of graphics, compute and transfers is recommended

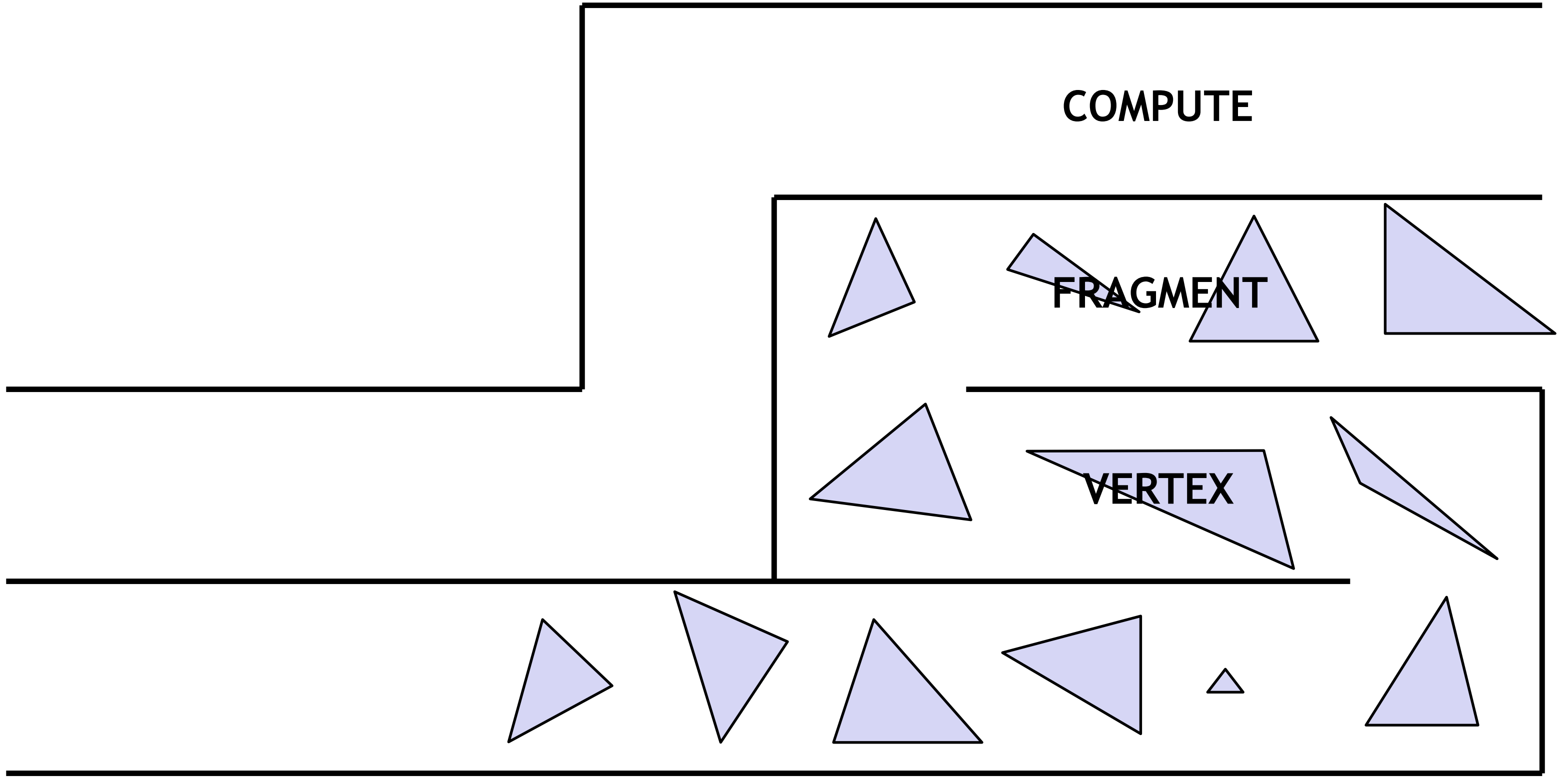
Keeping it fed



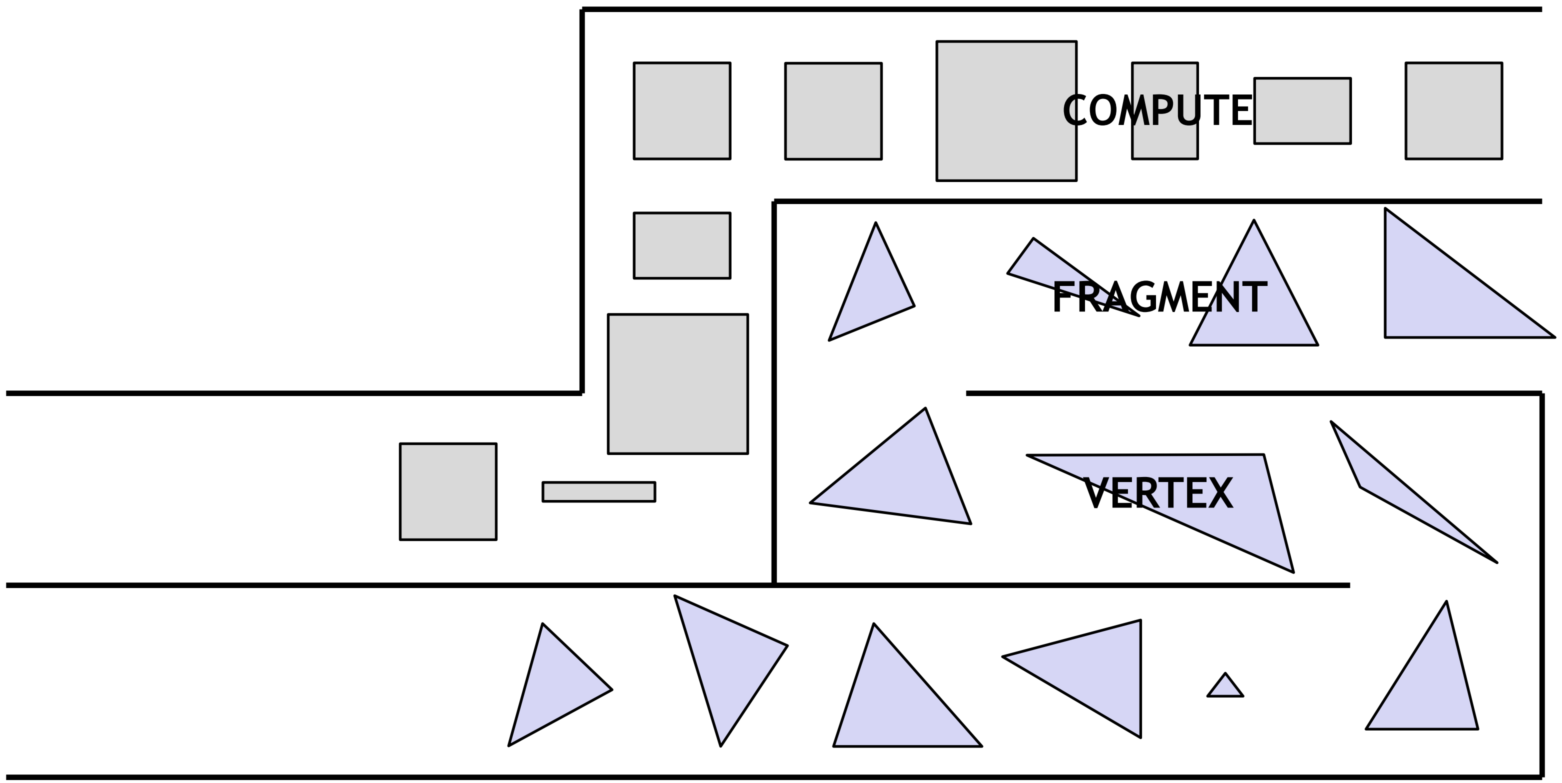
Keeping it fed



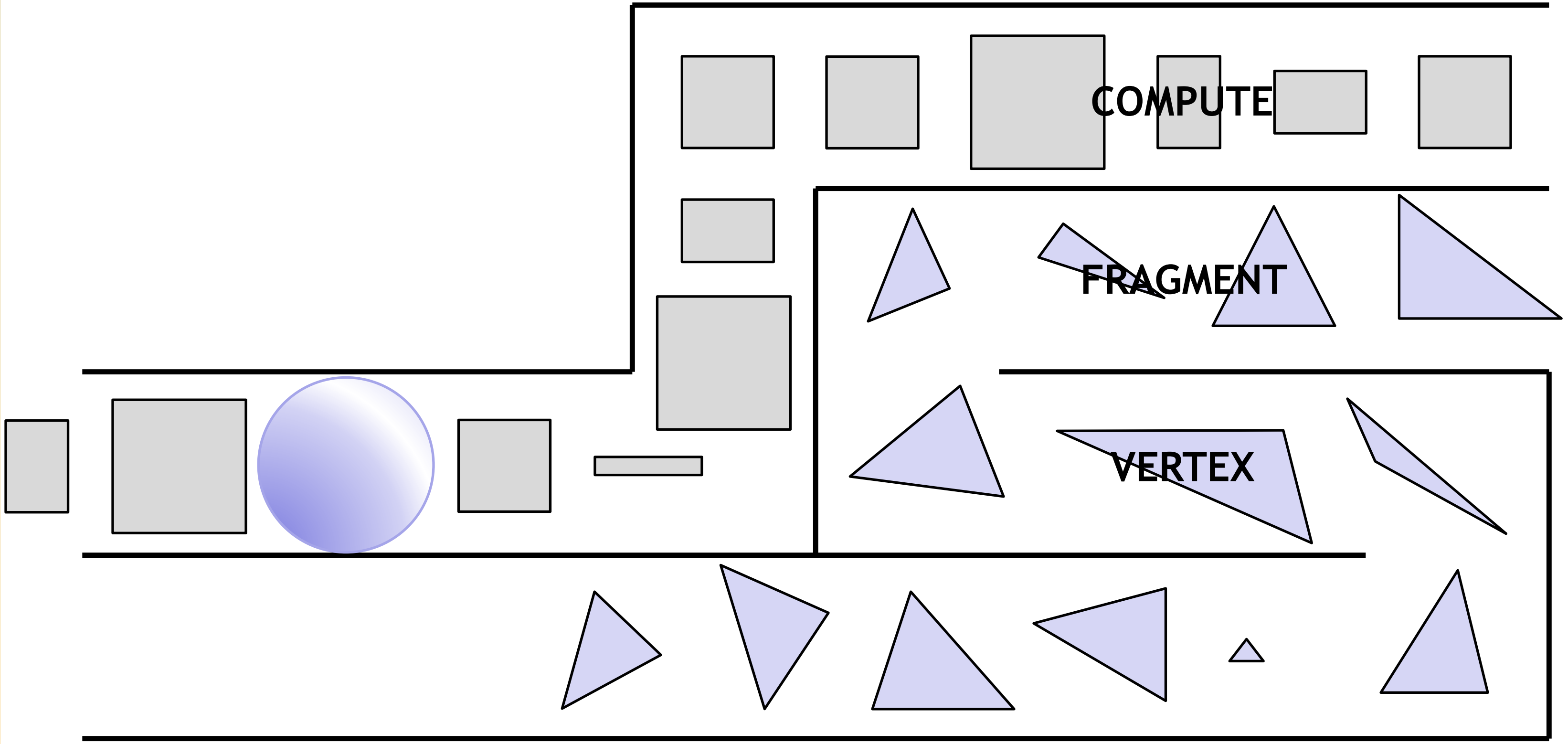
Keeping it fed



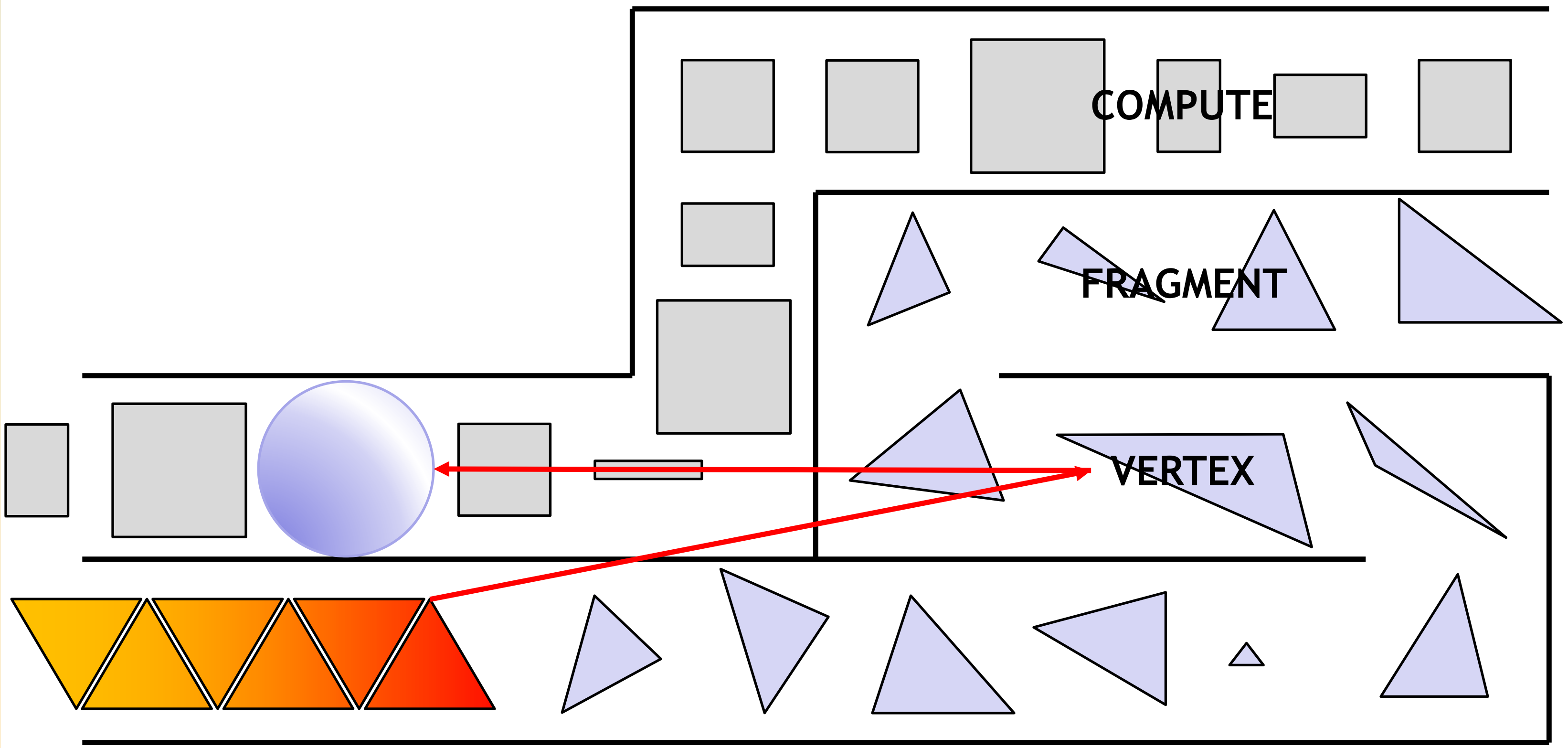
Keeping it fed



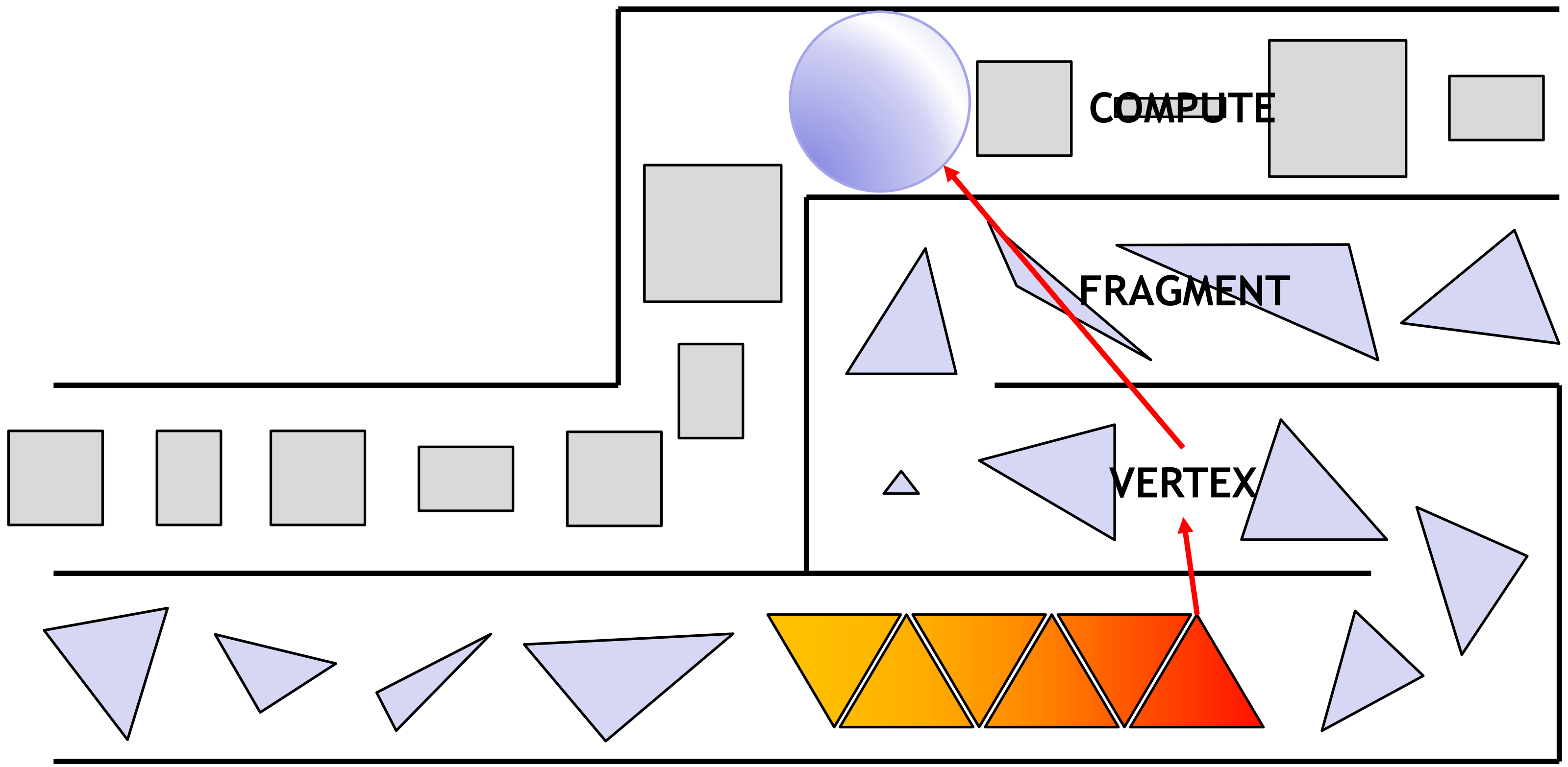
Keeping it fed



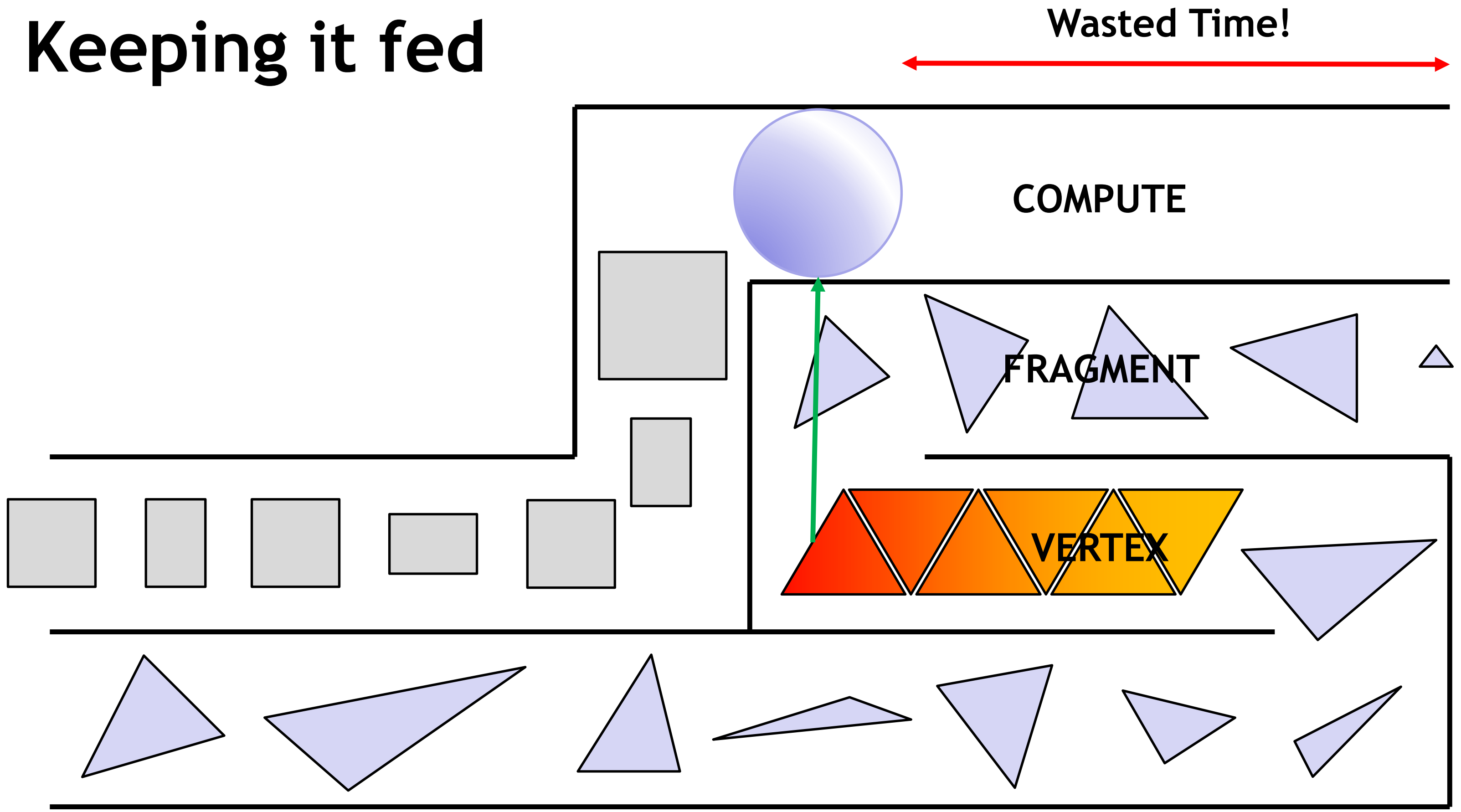
Keeping it fed



Keeping it fed

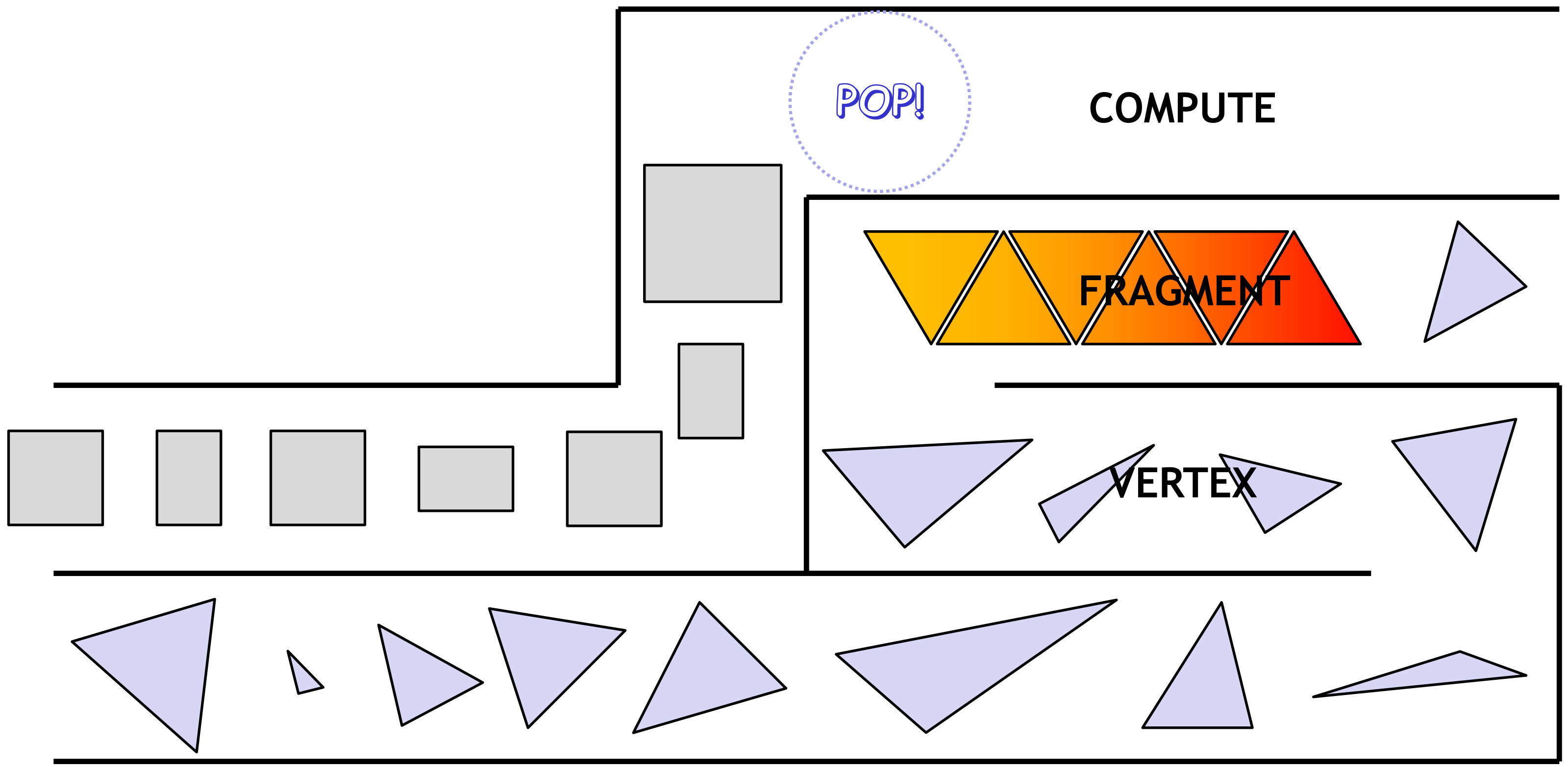


Keeping it fed



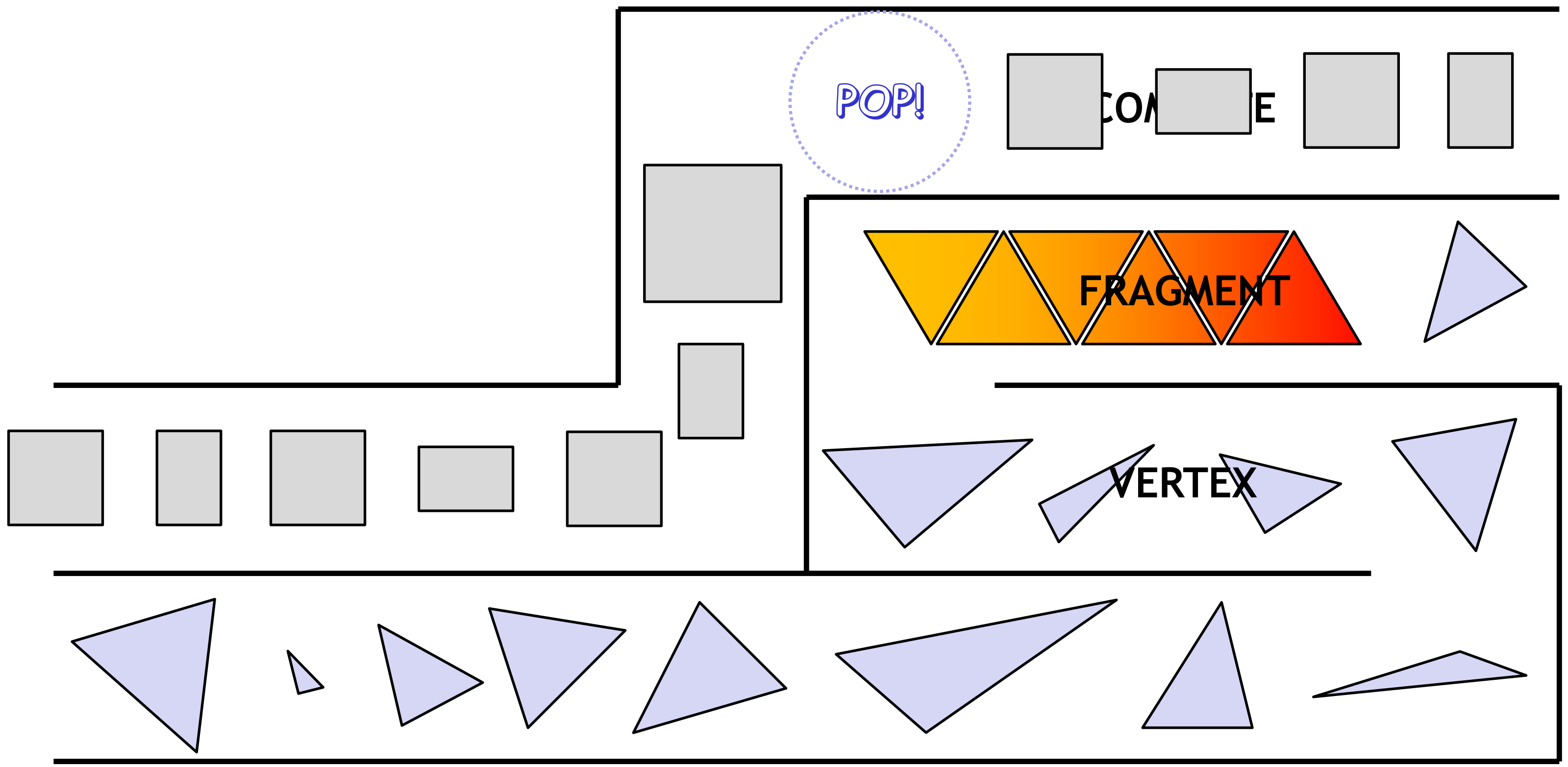
Keeping it fed

Off we go again…!



Keeping it fed

Better!

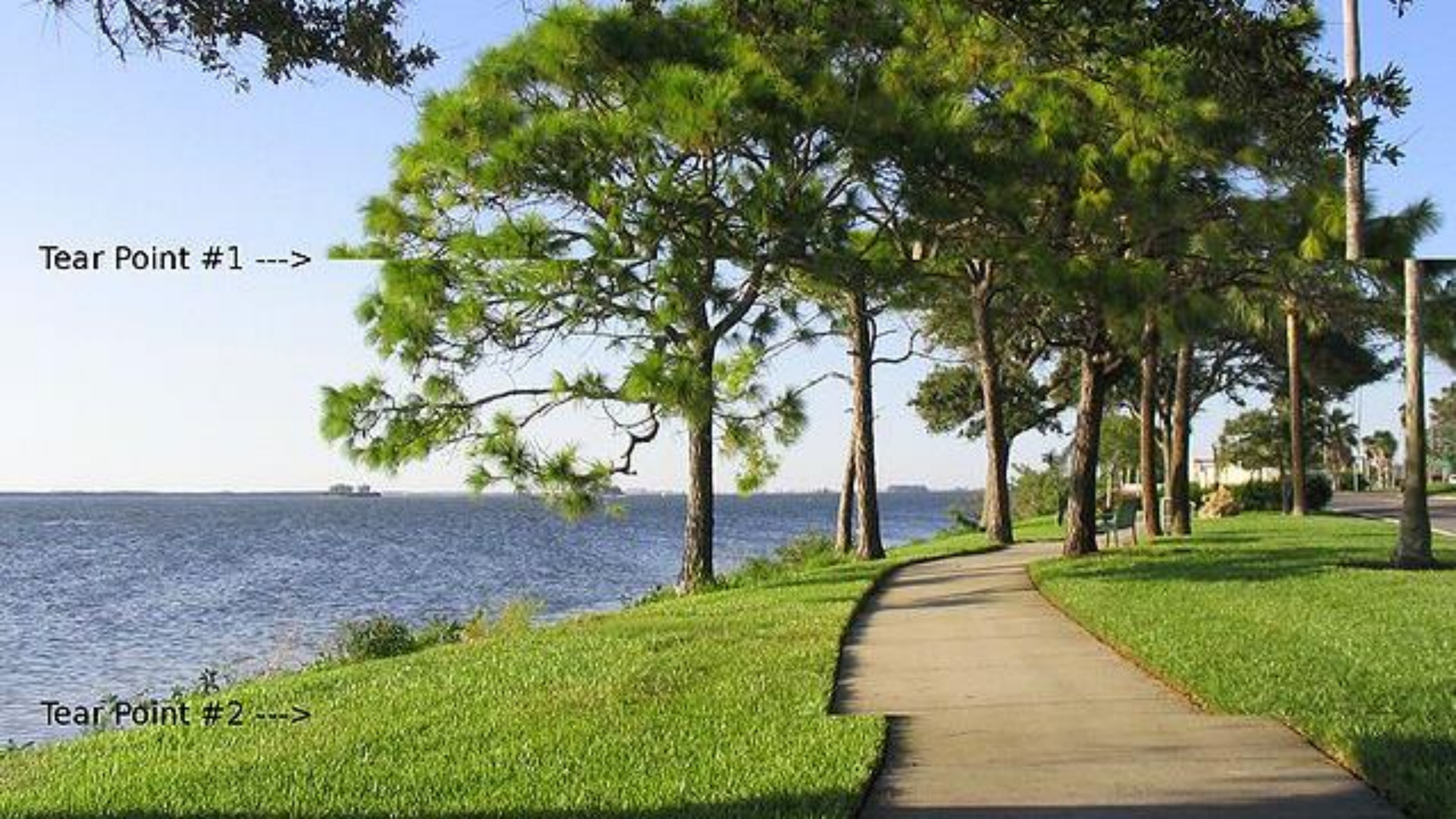


Not getting bitten

- GPU eating from lots of different plates
  - Don’t touch anything it’s using!

- It doesn’t want a mouthful of beef choc chip ice cream
  - Don’t change data whilst it’s accessing a resource

- Hey I’m eating that!
  - Don’t delete resources whilst the GPU is still using them


Tear Point #1 --->
Tear Point #2 --->

So what to do…

Synchronization Types

- **3 types of explicit synchronization in Vulkan**

- **Pipeline Barriers, Events and Subpass Dependencies**
  - Within a queue
  - Explicit memory dependencies

- **Semaphores**
  - Between Queues

- **Submission and Fences**
  - Coarse CPU-GPU synchronization

Synchronization Types

- 3 types of explicit synchronization in Vulkan

- ~~Pipeline Barriers,~~ Events ~~and Subpass Dependencies~~
  - Within a queue
  - Explicit memory dependencies

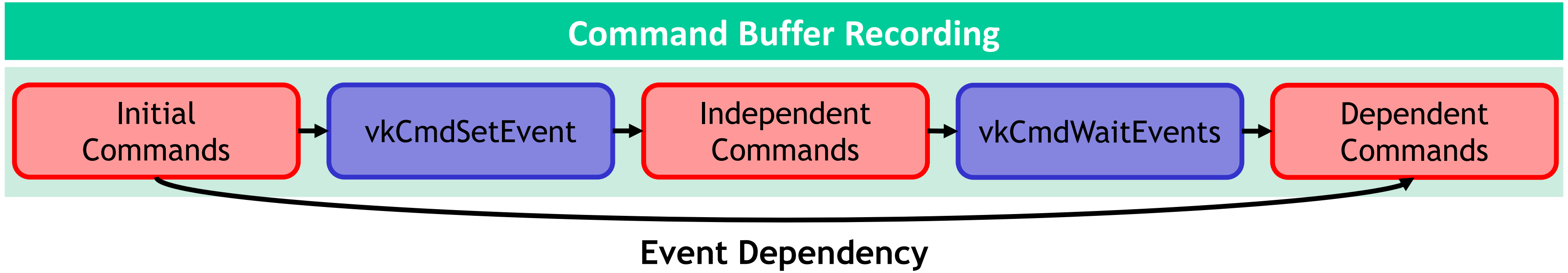
Two of these were covered in the last talk!

- Semaphores
  - Between Queues

- Submission and Fences
  - Coarse CPU-GPU synchronization

Events

- Events
  - Similar to pipeline barriers, but operate over a range
  - Use when work possible in-between

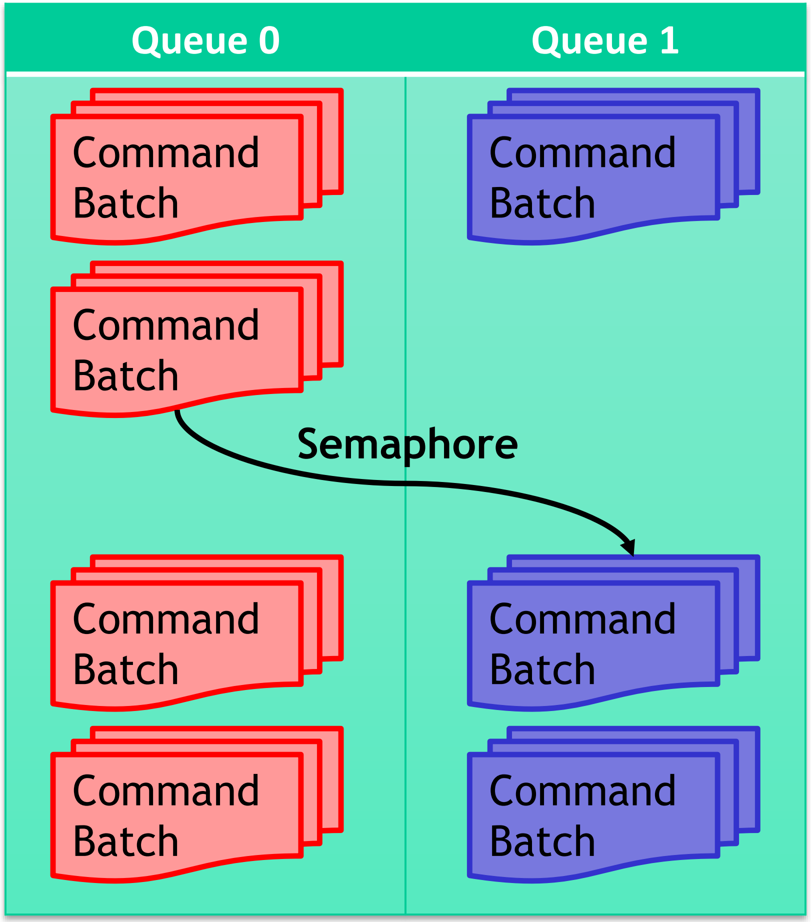


Semaphores

- **Semaphores**
  - Used to synchronize queues

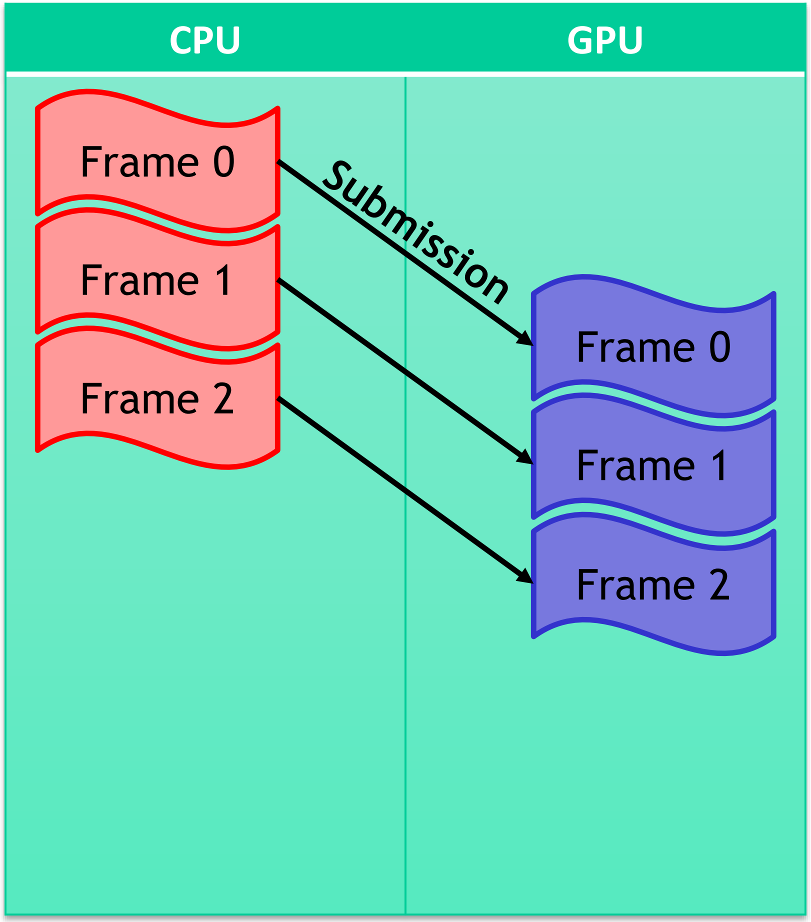
- **Fairly coarse**
  - N per batch of command buffers
  - Or per present/acquire

- **GPU caches invalidated at wait points**
  - “Available writes are made visible”



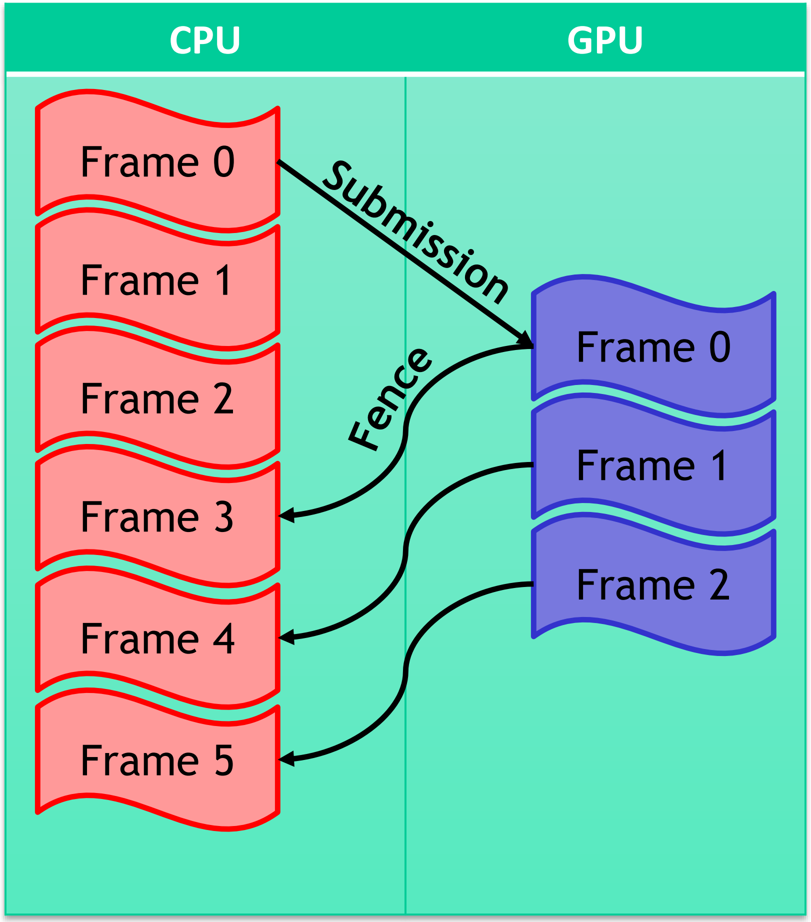
Submission

- Submission
  - Used to start GPU work
  - Triggered by the CPU



Submission & Fences

- **Submission**
  - Used to start GPU work
  - Triggered by the CPU
- **Fences**
  - Determine when GPU work complete
  - Wait/Query on the CPU
  - Key for resource management
- **Very coarse grain**
  - Several "batches" of work
  - E.g. One frame of rendering



Also enormous stall operations!

- **vkDeviceWaitIdle and vkQueueWaitIdle**
  - Huge sledgehammer – waits for GPU work to complete
  - Useful for teardown and debugging

Also enormous stall operations!

- **vkDeviceWaitIdle and vkQueueWaitIdle**
  - Huge sledgehammer – waits for GPU work to complete
  - Useful for teardown and debugging

- **Do not use these for anything else!**

Also enormous stall operations!

- **vkDeviceWaitIdle and vkQueueWaitIdle**
  - Huge sledgehammer – waits for GPU work to complete
  - Useful for teardown and debugging

- **Do not use these for anything else!**
  - Except *maybe* debugging

Programmer Guidelines

- **Specify EXACTLY the right amount of synchronization**
  - Too much and you risk starving your GPU
  - Miss any and your GPU will bite you
- **Pay particular attention to the pipeline stages**
  - Fiddly but become intuitive as you use them
- **Consider Image Layouts**
  - If your GPU can save bandwidth it will
- **Prefer render passes**
  - Driver able to plan workloads efficiently
- **Pay attention to implicit dependencies**
  - Submit and Semaphores guarantee a lot – don’t add more!
- **Different behaviour depending on implementation**
  - Test/Tune on every platform you can find!

Tooling that can help

- **Firstly, fair warning**
  - The state of tooling is not great for *debugging* general issues
  - We're aware of this being an issue…

Tooling that can help

- **Firstly, fair warning**
  - The state of tooling is not great for *debugging* general issues
  - We're aware of this being an issue…

- **However, there are some!**

Validation Layers

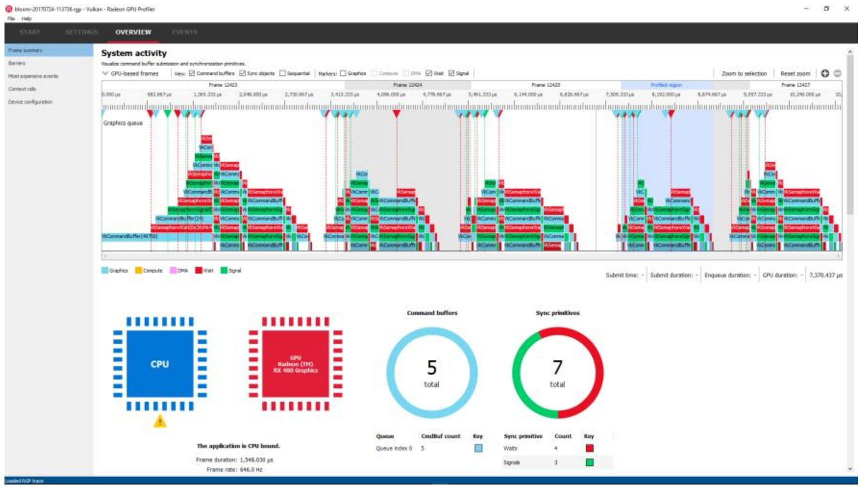
- **Figuring out the right enum combinations is hard**
  - Access flags, pipeline stages, layouts…
  - Hard to commit to memory

- **Validation will catch obvious mistakes**
  - E.g. pairing shader access flags with a non-shader stage
  - Using graphics stages on a compute queue
  - Some amount of general misuse

- **It's a useful amount, and getting better**
  - Please shout if getting more done is important!
  - Will help us raise priority

Profiling Tools

- Will help detect over-synchronization
  - Pipeline bubbles
  - Unnecessary layout transitions
  - Unexpected Stalls



View: ☑ Command buffers ☑ Sync objects ☐ Sequential | Markers: ☐ Graphics ☐ Compute ☐ DMA ☑ Wait ☑ Signal

Frame 12423 | Frame 12424 | Frame 12425 | Profiled region | Frame 12427

0.000 µs 682.667 µs 1,365.333 µs 2,048.000 µs 2,730.667 µs 3,413.333 µs 4,096.000 µs 4,778.667 µs 5,461.333 µs 6,144.000 µs 6,826.667 µs 7,509.333 µs 8,192.000 µs 8,874.667 µs 9,557.333 µs 10,240.000 µs

Submit time: - | Submit duration: - | Enqueue duration: - | GPU duration: - | 7,370.437 µs

The application is CPU bound.
Frame duration: 1,548.030 µs
Frame rate: 646.0 Hz

Queue	CmdBuf count	Key
Queue index 0	5	🟦

Sync primitive	Count	Key
Waits	4	🟥
Signals	3	🟩

- Currently mostly vendor specific
  - E.g. use RGP for AMD GPUs
  - https://gpuopen.com/gaming-product/radeon-gpu-profiler-rgp/

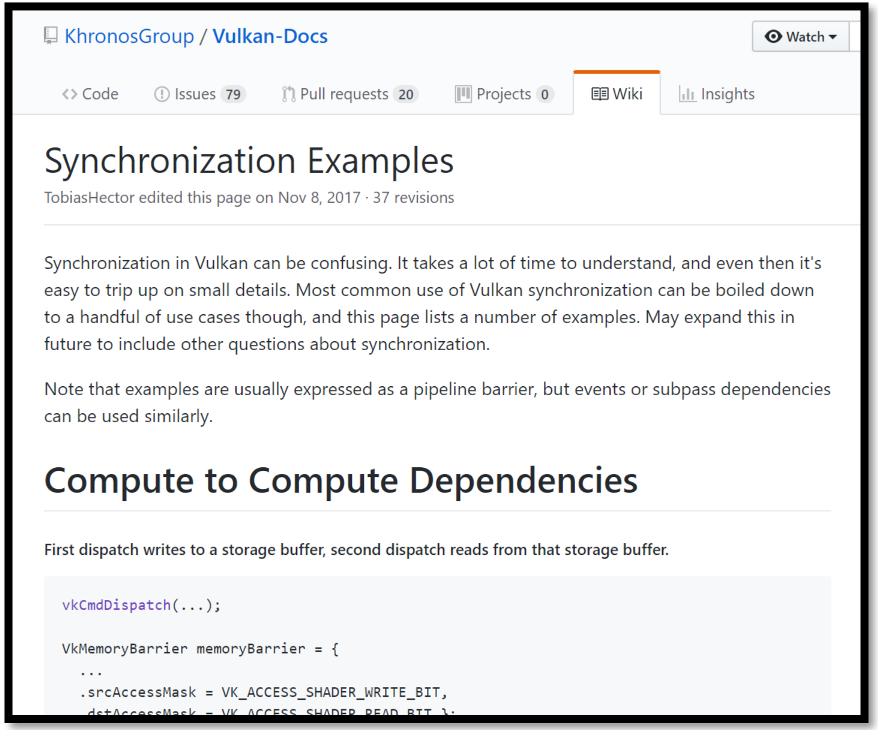
- RenderDoc has *some* vendor profiling integration
  - Can be used as a catch all, but playback may affect results
  - Still needs to run on each vendor platform

Examples!

- **Plenty of examples around**
  - Even some in the spec
  - Have a root around on the net?

Examples!

- Plenty of examples around
  - Even some in the spec
  - Have a root around on the net?



KhronosGroup / Vulkan-Docs

Synchronization Examples

TobiasHector edited this page on Nov 8, 2017 · 37 revisions

Synchronization in Vulkan can be confusing. It takes a lot of time to understand, and even then it's easy to trip up on small details. Most common use of Vulkan synchronization can be boiled down to a handful of use cases though, and this page lists a number of examples. May expand this in future to include other questions about synchronization.

Note that examples are usually expressed as a pipeline barrier, but events or subpass dependencies can be used similarly.

Compute to Compute Dependencies

First dispatch writes to a storage buffer, second dispatch reads from that storage buffer.

```
vkCmdDispatch(...);

VkMemoryBarrier memoryBarrier = {
  ...
  .srcAccessMask = VK_ACCESS_SHADER_WRITE_BIT,
  .dstAccessMask = VK_ACCESS_SHADER_READ_BIT };
```

- More helpfully…
  - There's a page full of pointed examples here:
  - https://github.com/KhronosGroup/Vulkan-Docs/wiki/Synchronization-Examples
  - Raise issues if you want something added!

Simple(r) Vulkan Synchronization

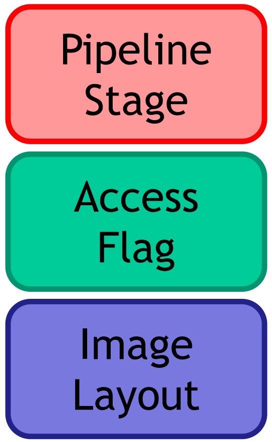
- **Single-header library**
  - Simpler synchronization API than raw Vulkan

- **Still expresses most of the flexibility**
  - With only a tiny bit of the complexity
  - (Mileage may vary ☺)

- **So easy even DX12 developers can use it!**
  - Sorry, it's still hard, but this should really help

Simple(r) Vulkan Synchronization

- So what does it do exactly?



n Synchronization

VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT
VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT
VK_PIPELINE_STAGE_VERTEX_INPUT_BIT
VK_PIPELINE_STAGE_VERTEX_SHADER_BIT
VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT
VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT
VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT
VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT
VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT
VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT
VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT
VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT
VK_PIPELINE_STAGE_TRANSFER_BIT
VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT
VK_PIPELINE_STAGE_HOST_BIT
VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT
VK_PIPELINE_STAGE_ALL_COMMANDS_BIT

Pipeline Stage

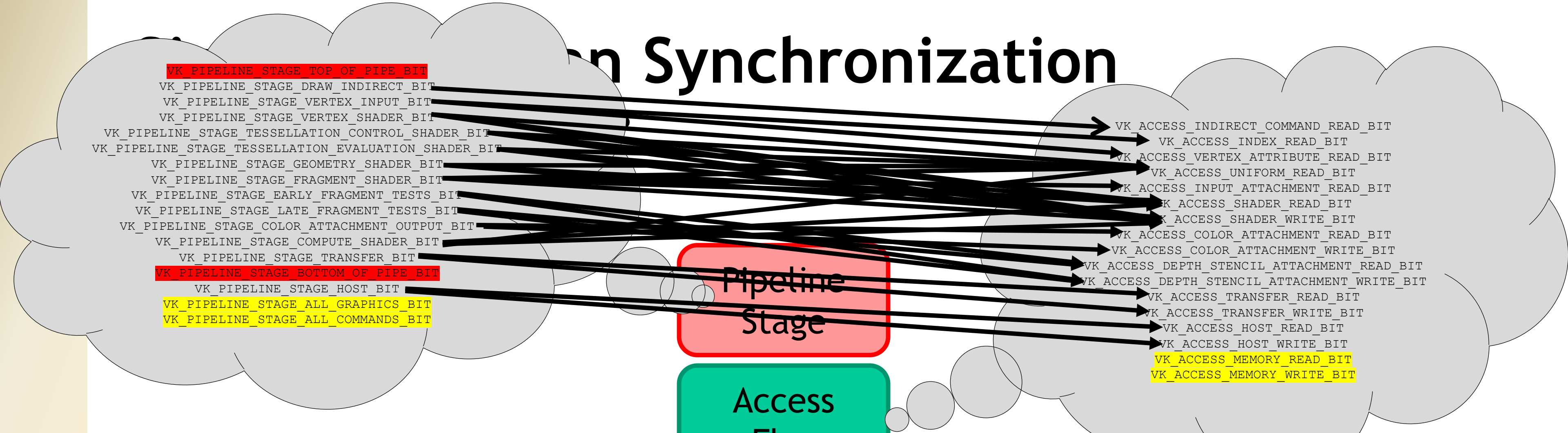
Access Flag

Image Layout

VK_ACCESS_INDIRECT_COMMAND_READ_BIT
VK_ACCESS_INDEX_READ_BIT
VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT
VK_ACCESS_UNIFORM_READ_BIT
VK_ACCESS_INPUT_ATTACHMENT_READ_BIT
VK_ACCESS_SHADER_READ_BIT
VK_ACCESS_SHADER_WRITE_BIT
VK_ACCESS_COLOR_ATTACHMENT_READ_BIT
VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT
VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT
VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT
VK_ACCESS_TRANSFER_READ_BIT
VK_ACCESS_TRANSFER_WRITE_BIT
VK_ACCESS_HOST_READ_BIT
VK_ACCESS_HOST_WRITE_BIT
VK_ACCESS_MEMORY_READ_BIT
VK_ACCESS_MEMORY_WRITE_BIT

VK_IMAGE_LAYOUT_UNDEFINED
VK_IMAGE_LAYOUT_GENERAL
VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL
VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL
VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL
VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL
VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL
VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL
VK_IMAGE_LAYOUT_PREINITIALIZED
VK_IMAGE_LAYOUT_PRESENT_SRC_KHR

n Synchronization



VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT
VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT
VK_PIPELINE_STAGE_VERTEX_INPUT_BIT
VK_PIPELINE_STAGE_VERTEX_SHADER_BIT
VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT
VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT
VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT
VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT
VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT
VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT
VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT
VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT
VK_PIPELINE_STAGE_TRANSFER_BIT
VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT
VK_PIPELINE_STAGE_HOST_BIT
VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT
VK_PIPELINE_STAGE_ALL_COMMANDS_BIT

VK_ACCESS_INDIRECT_COMMAND_READ_BIT
VK_ACCESS_INDEX_READ_BIT
VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT
VK_ACCESS_UNIFORM_READ_BIT
VK_ACCESS_INPUT_ATTACHMENT_READ_BIT
VK_ACCESS_SHADER_READ_BIT
VK_ACCESS_SHADER_WRITE_BIT
VK_ACCESS_COLOR_ATTACHMENT_READ_BIT
VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT
VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT
VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT
VK_ACCESS_TRANSFER_READ_BIT
VK_ACCESS_TRANSFER_WRITE_BIT
VK_ACCESS_HOST_READ_BIT
VK_ACCESS_HOST_WRITE_BIT
VK_ACCESS_MEMORY_READ_BIT
VK_ACCESS_MEMORY_WRITE_BIT

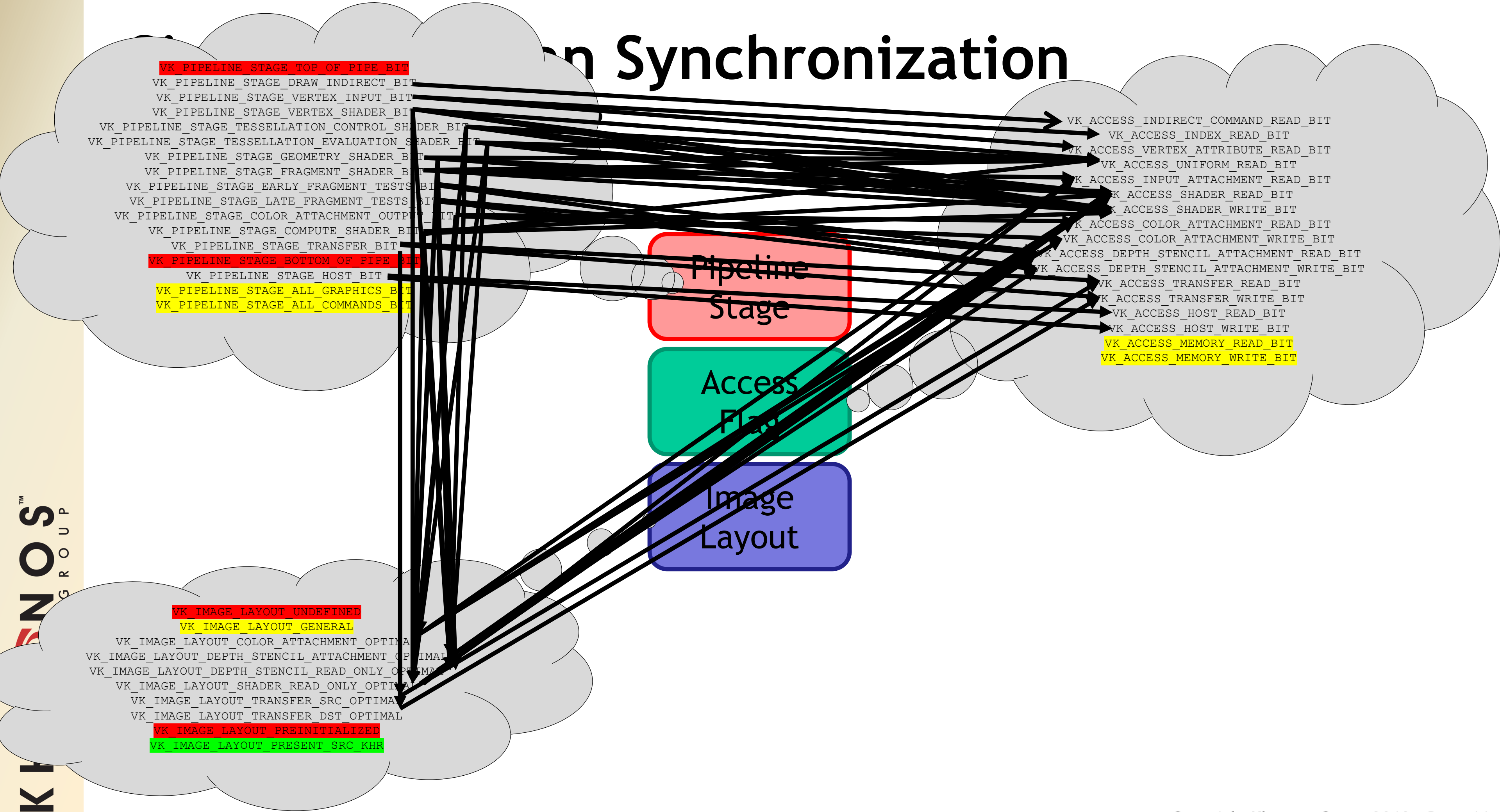
Pipeline Stage

Access Flag

Image Layout

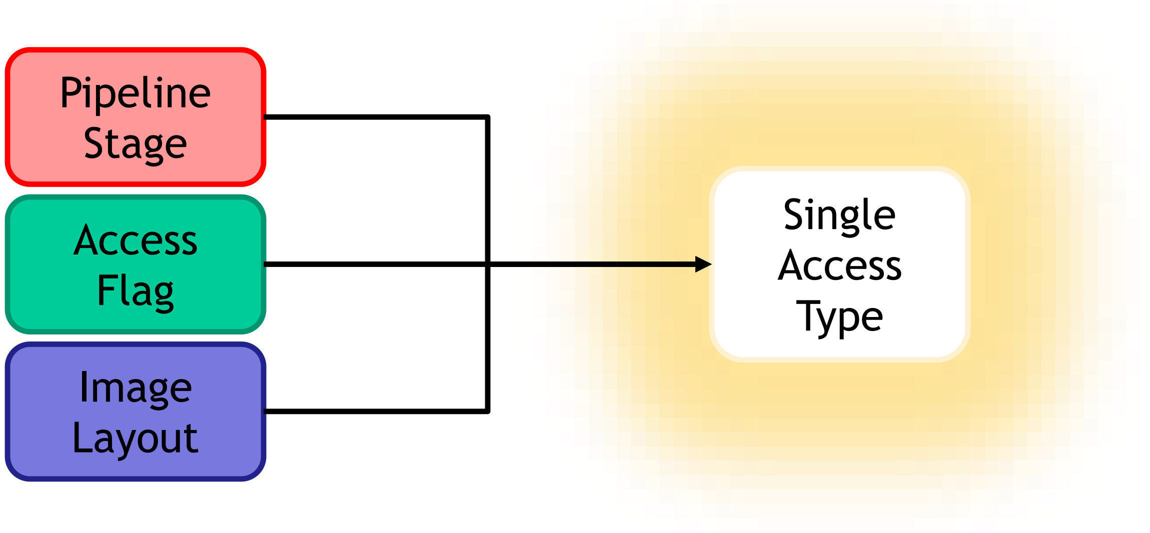
VK_IMAGE_LAYOUT_UNDEFINED
VK_IMAGE_LAYOUT_GENERAL
VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL
VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL
VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL
VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL
VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL
VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL
VK_IMAGE_LAYOUT_PREINITIALIZED
VK_IMAGE_LAYOUT_PRESENT_SRC_KHR

n Synchronization


VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT
VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT
VK_PIPELINE_STAGE_VERTEX_INPUT_BIT
VK_PIPELINE_STAGE_VERTEX_SHADER_BIT
VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT
VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT
VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT
VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT
VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT
VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT
VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT
VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT
VK_PIPELINE_STAGE_TRANSFER_BIT
VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT
VK_PIPELINE_STAGE_HOST_BIT
VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT
VK_PIPELINE_STAGE_ALL_COMMANDS_BIT
VK_ACCESS_INDIRECT_COMMAND_READ_BIT
VK_ACCESS_INDEX_READ_BIT
VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT
VK_ACCESS_UNIFORM_READ_BIT
VK_ACCESS_INPUT_ATTACHMENT_READ_BIT
VK_ACCESS_SHADER_READ_BIT
VK_ACCESS_SHADER_WRITE_BIT
VK_ACCESS_COLOR_ATTACHMENT_READ_BIT
VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT
VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT
VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT
VK_ACCESS_TRANSFER_READ_BIT
VK_ACCESS_TRANSFER_WRITE_BIT
VK_ACCESS_HOST_READ_BIT
VK_ACCESS_HOST_WRITE_BIT
VK_ACCESS_MEMORY_READ_BIT
VK_ACCESS_MEMORY_WRITE_BIT
Pipeline Stage
Access Flag
Image Layout
VK_IMAGE_LAYOUT_UNDEFINED
VK_IMAGE_LAYOUT_GENERAL
VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL
VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL
VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL
VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL
VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL
VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL
VK_IMAGE_LAYOUT_PREINITIALIZED
VK_IMAGE_LAYOUT_PRESENT_SRC_KHR

Simple(r) Vulkan Synchronization

- So what does it do exactly?



Simple(r) Vulkan Synchronization

- So what does it do exactly?

Single
Access
Type

- **Getting the right combinations of 3 independent values is <u>really painful</u>**
  - Has been a constant source of problems
  - Should save a few head-desk related incidents and a bunch of debugging…

Simple(r) Vulkan Synchronization

- Disclaimer
  - This library is still ***alpha*** software

- Status?
  - The mappings are all solid – use them as a reference
  - Interface may change
  - A few less common use cases might be missing
  - Some success with one app integration already

- **Interested in direct integration with an app/engine?**
  - Talk to me afterwards ☺

- **Get it here:**
  - https://github.com/Tobski/simple_vulkan_synchronization

Programmer Guidelines

- **Specify EXACTLY the right amount of synchronization**
  - Too much and you risk starving your GPU
  - Miss any and your GPU will bite you
- **Pay particular attention to the pipeline stages**
  - Fiddly but become intuitive as you use them
- **Consider Image Layouts**
  - If your GPU can save bandwidth it will
- **Prefer render passes**
  - Driver able to plan workloads efficiently
- **Pay attention to implicit dependencies**
  - Submit and Semaphores guarantee a lot – don’t add more!
- **Different behaviour depending on implementation**
  - Test/Tune on every platform you can find!

Programmer Guidelines

- **Specify EXACTLY the right amount of synchronization**
  - Too much and you risk starving your GPU
  - Miss any and your GPU will bite you
- **Pay particular attention to the pipeline stages**
  - Fiddly but become intuitive as you use them
- **Consider Image Layouts**
  - If your GPU can save bandwidth it will
- **Prefer render passes**
  - Driver able to plan workloads efficiently
- **Pay attention to implicit dependencies**
  - Submit and Semaphores guarantee a lot – don't add more!
- **Different behaviour depending on implementation**
  - Test/Tune on every platform you can find!
- **USE ALL OF THE TOOLS**

Programmer Guidelines

- **Specify EXACTLY the right amount of synchronization**
  - Too much and you risk starving your GPU
  - Miss any and your GPU will bite you
- **Pay particular attention to the pipeline stages**
  - Fiddly but become intuitive as you use them
- **Consider Image Layouts**
  - If your GPU can save bandwidth it will
- **Prefer render passes**
  - Driver able to plan workloads efficiently
- **Pay attention to implicit dependencies**
  - Submit and Semaphores guarantee a lot – don’t add more!
- **Different behaviour depending on implementation**
  - Test/Tune on every platform you can find!
- **USE ALL OF THE TOOLS***

* Sorry they’re a bit lacking

Keep your GPU fed without getting bitten!

Questions?