Vulkanised 2019

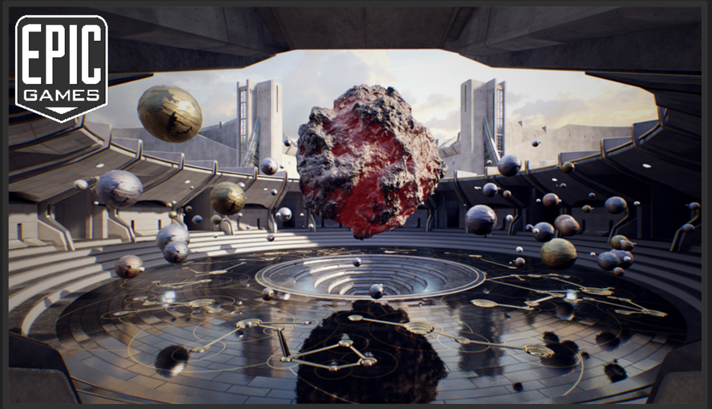
Live Long and Optimise

Michael Parkin-White, Calum Shields
m.parkin-whi@samsung.com c.shields@samsung.com

Galaxy GameDev

Vulkan®

Intro – Samsung Galaxy GameDev



Promoting use of Vulkan on Android

Support studios remotely & on-site

Help partners optimise their games through use of tools and best-practices

Galaxy GameDev

Vulkan®

Optimal Swapchain Management

An in-depth investigation - beyond the basics

Galaxy GameDev

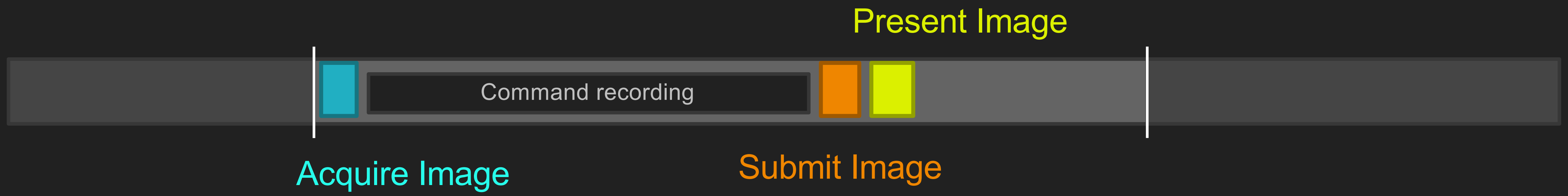
Vulkan®

Vulkan Swapchain & presentation

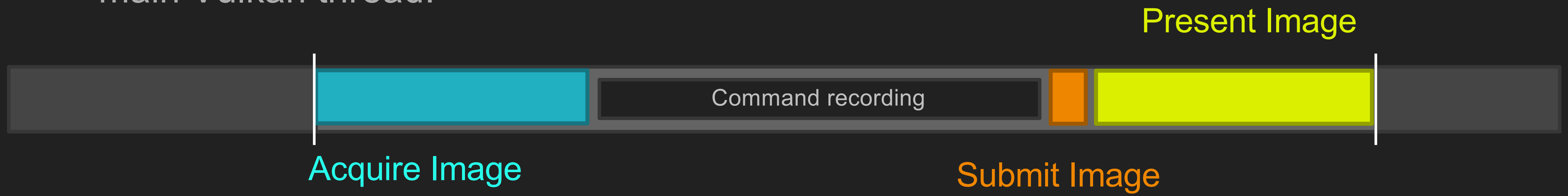
- Consists of three interactions w/ presentation engine:
  - **vkAcquireNextImageKHR(..)** – Potential blocking if no swapchain image is available
  - **vkQueueSubmit(..)** – Non blocking
  - ~~**vkQueuePresentKHR(..)** – Non blocking~~
  - **vkQueuePresentKHR(..)** – Frequently blocks on Android - in some cases for > **20ms!**

Standard swapchain coordination

- A standard approach to swapchain management:

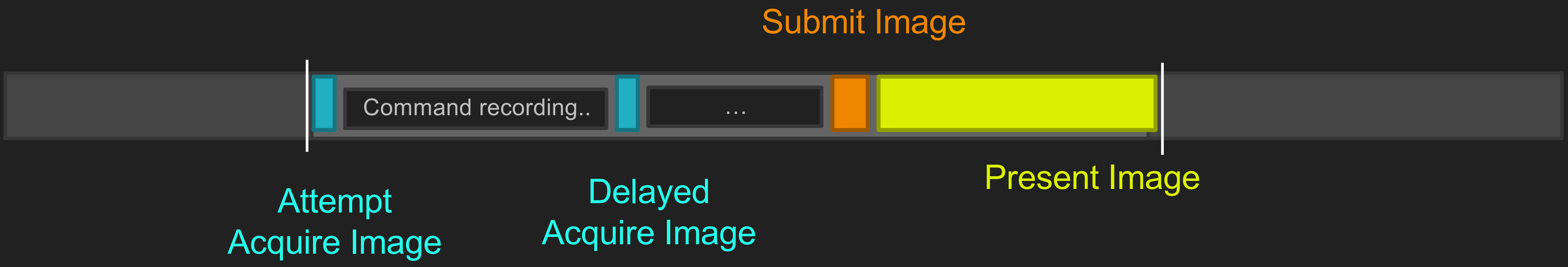


- However, we have problems when blocking behaviour emerges – increased CPU time on main Vulkan thread:



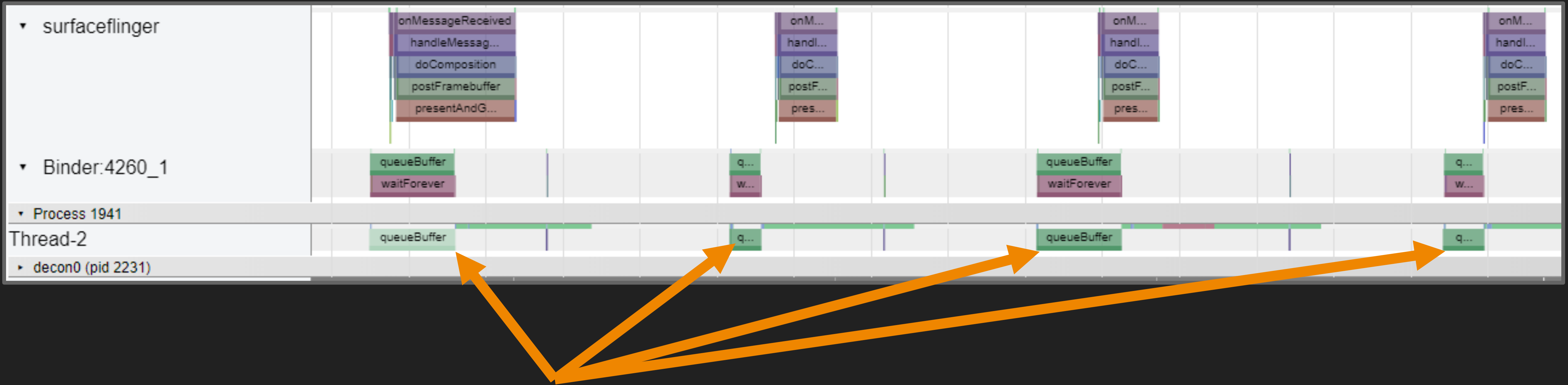
Consideration – Delayed Acquire

- In games with highly variable frame timings, there is a benefit to having two points at which we attempt **vkAcquireNextImageKHR**.



The blocking vkQueuePresentKHR issue

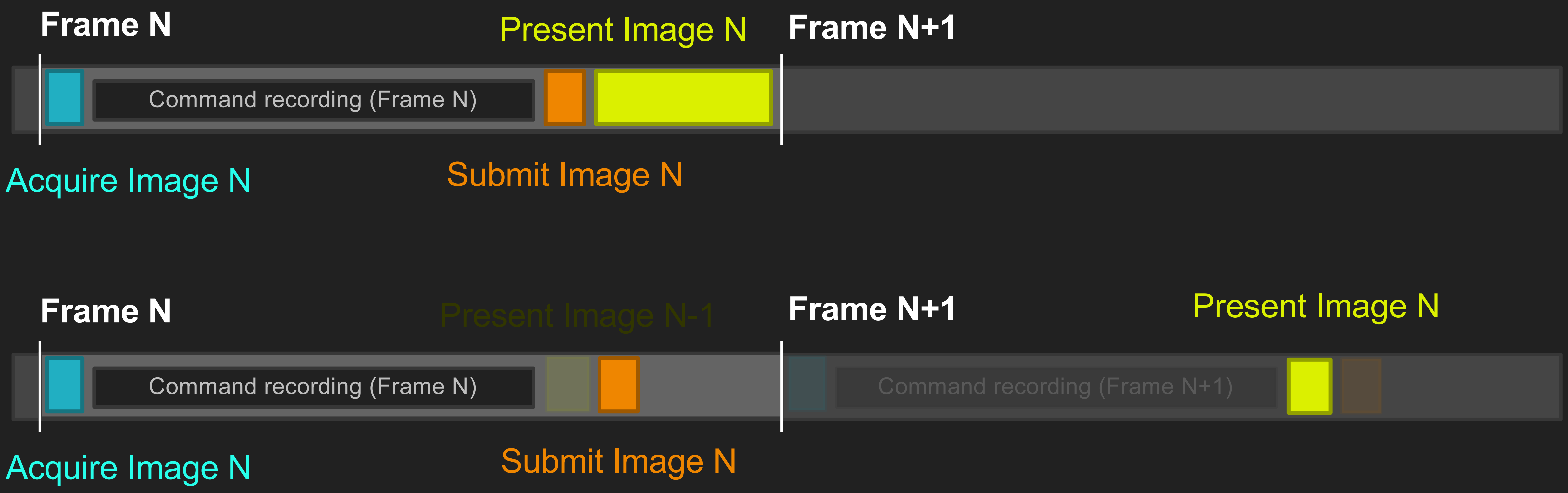
- We have observed that vkQueuePresentKHR can block for significant durations on Android



queueBuffer (called by vkQueuePresentKHR) taking an average of **12ms**

Solution #1 – Delayed Present

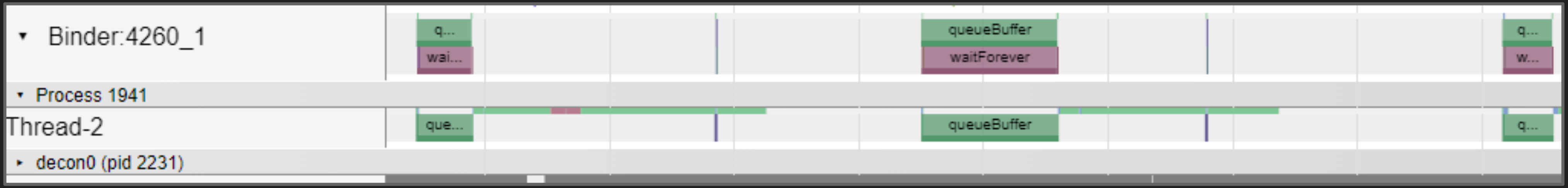
- This delay is influenced by proximity to vkQueueSubmit – So we can instead delay the call to present



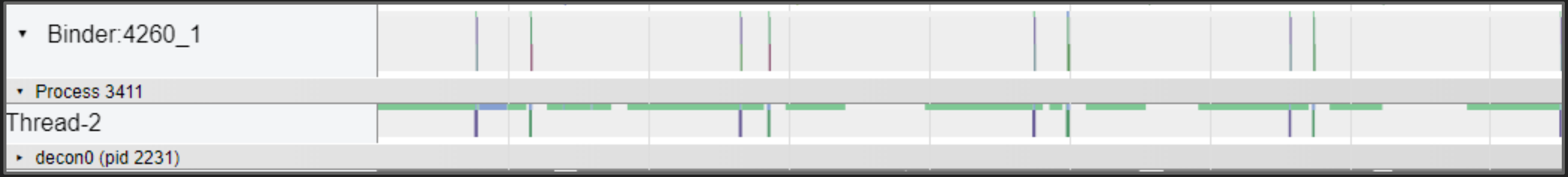
Galaxy GameDev

Vulkan®

Results – Delayed Present



Default Present: *queueBuffer takes an average of* ***12ms***



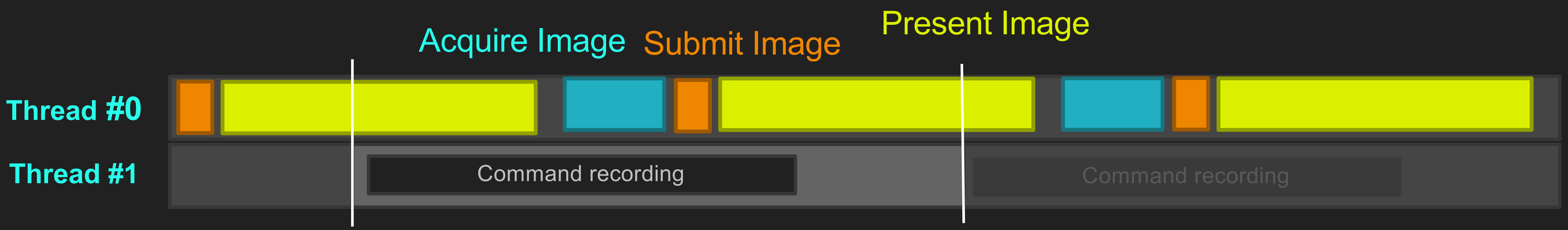
Delayed Present: *queueBuffer takes an average of* ***0.2ms***

Galaxy GameDev

Vulkan®

Solution #2 – Presentation Thread

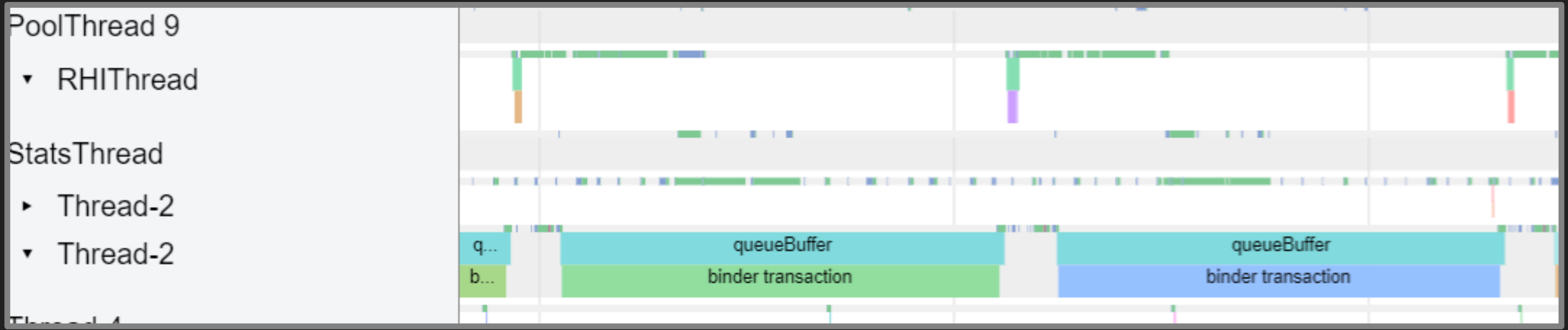
- Another solution is to defer swapchain interaction calls to a separate thread:



- In this case, we move the calls off the work/recording thread – allowing the wait to be absorbed externally
- We can continue with useful CPU work on the main thread.
  - A synchronisation check should be added to prevent the CPU from getting too many frames ahead (i.e. more than 2 ahead).

Results – presentation thread

- With the presentation thread implemented in **UE4 Sun-temple** demo:



Thread-2 is now used for presentation

RHI Thread is now free to get on with useful work!

	Standard UE4.22.0 VulkanRHI	UE4.22.0 VulkanRHI With presentation thread
S960U – Locked Frequency	FPS: 34	FPS: 41

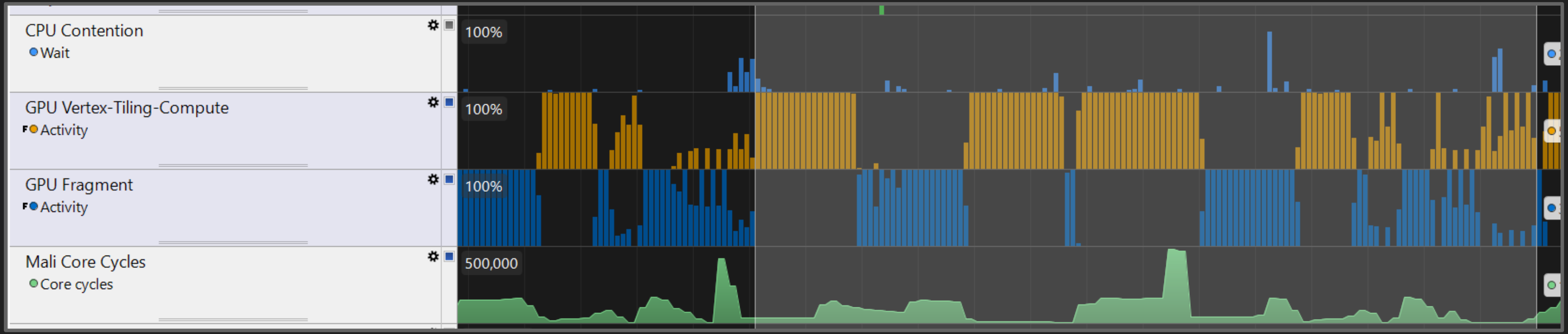
Pipeline Analysis

Optimising rendering workflow

Galaxy GameDev

Vulkan®

Example: Pipeline analysis



Poor pipelining results in pipeline bubbles – Not getting the most out of GPU

Fragment and vertex work never running in parallel

Caused by sub-optimal pipeline barrier and subpass-dependency stage masks

Current Frame Cost:
62.5ms (16 fps!)

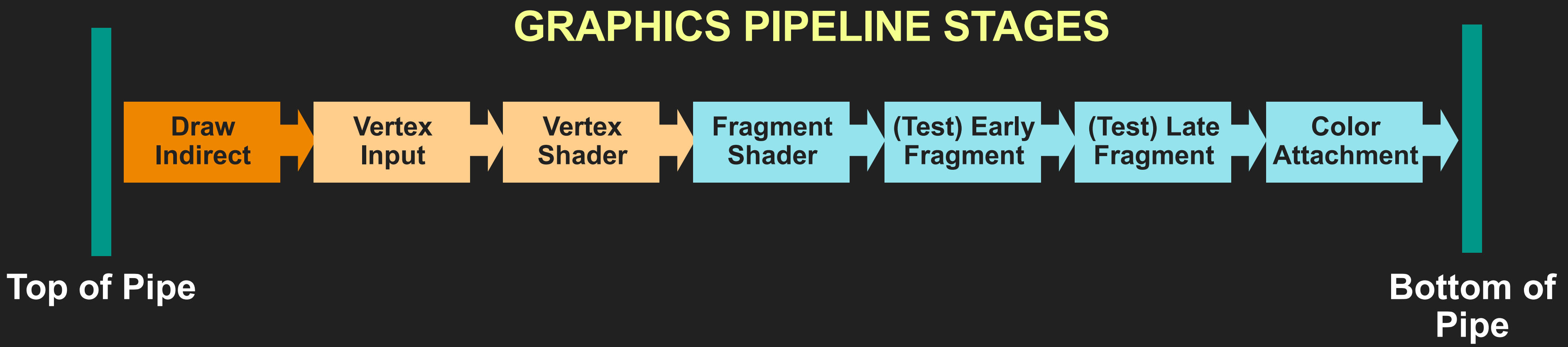
Galaxy GameDev

Vulkan

Pipeline Barriers: Quick overview

- Used to specify **execution dependencies** between specific pipeline stages in two action commands

- **Destination stage mask:** Specifies where the 2nd (next) action item will wait for the 1st (previous) action to complete its **Source stage mask** stages

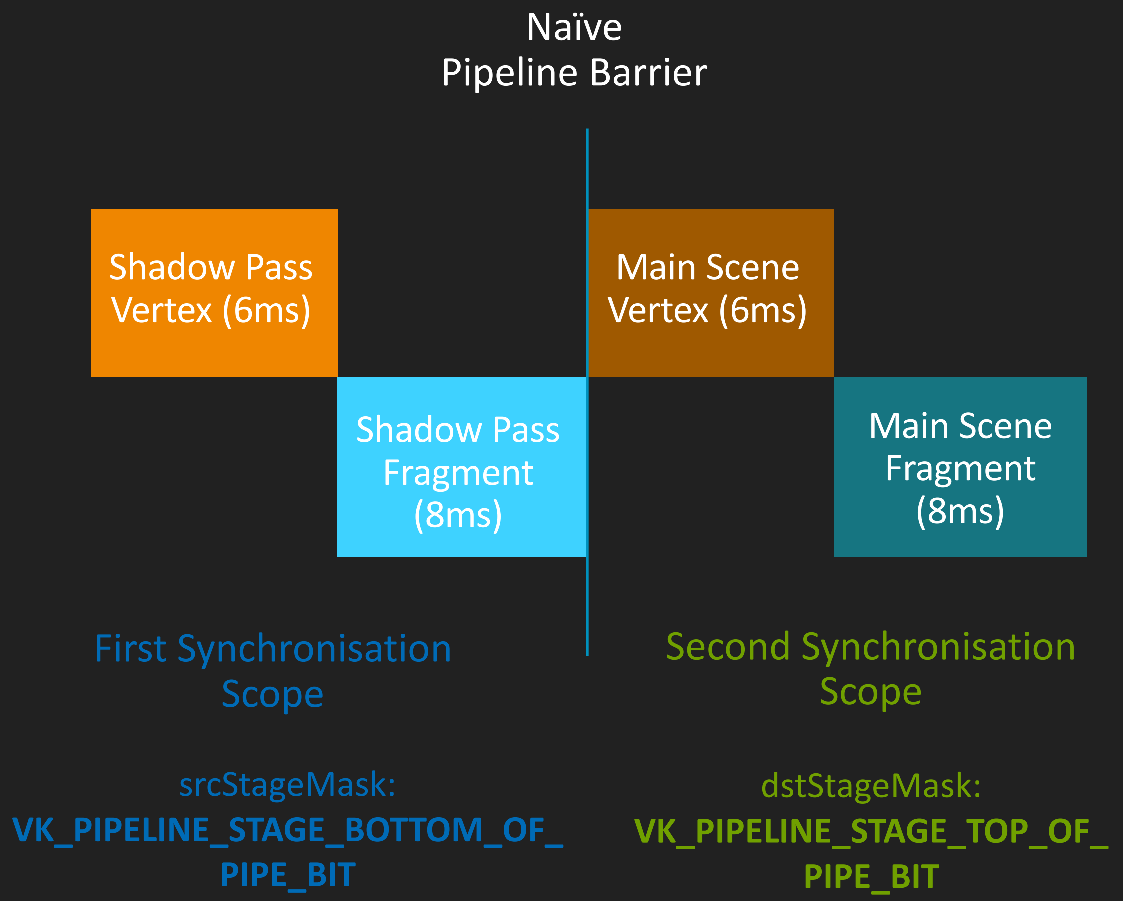


Galaxy GameDev

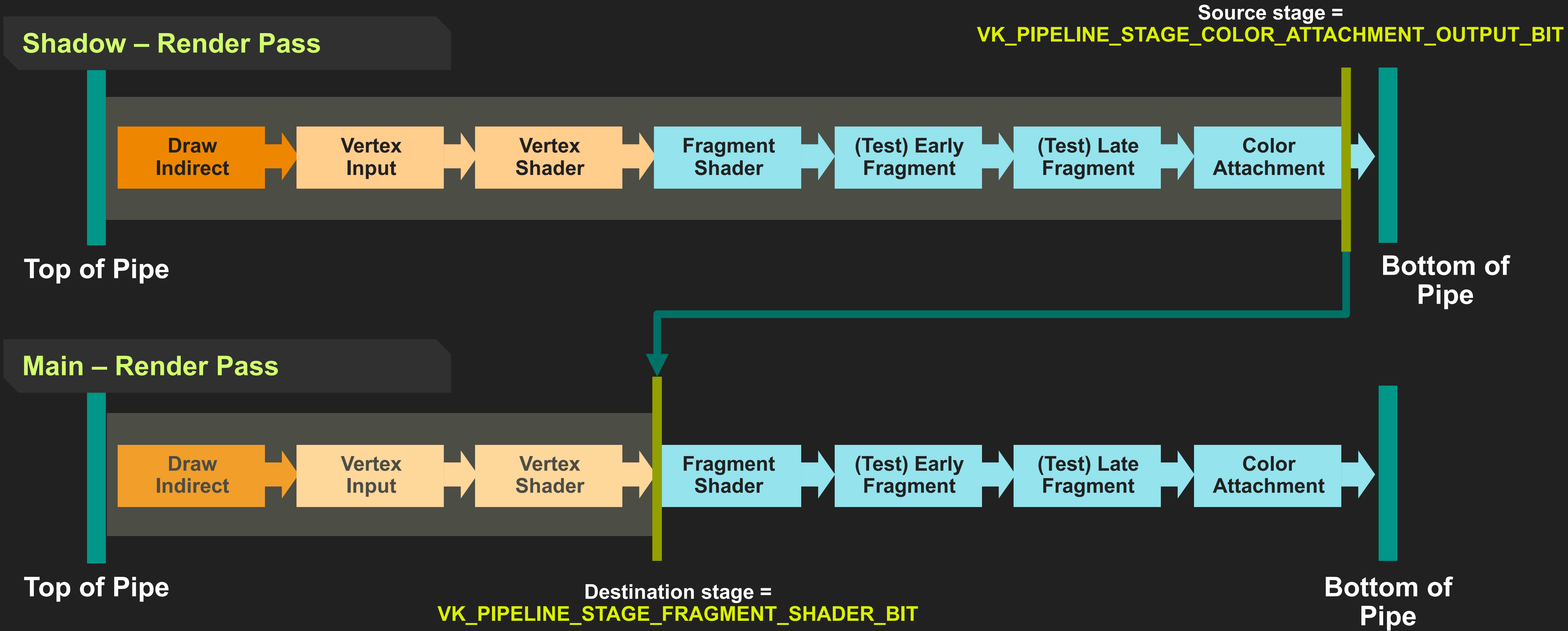
Vulkan

Pipeline barriers example

- **Simplified Example:** Render with two passes. Shadow mapping and main render
- Main scene render needs to use the shadow map rendered in the first pass
- Naïve synchronisation assumes entire shadow pass needs to complete **before** we start the main scene's rendering work


Naïve
Pipeline Barrier
Shadow Pass Vertex (6ms)
Shadow Pass Fragment (8ms)
Main Scene Vertex (6ms)
Main Scene Fragment (8ms)
First Synchronisation Scope
Second Synchronisation Scope
srcStageMask:
VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT
dstStageMask:
VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT

Pipeline Barriers: Improved case!



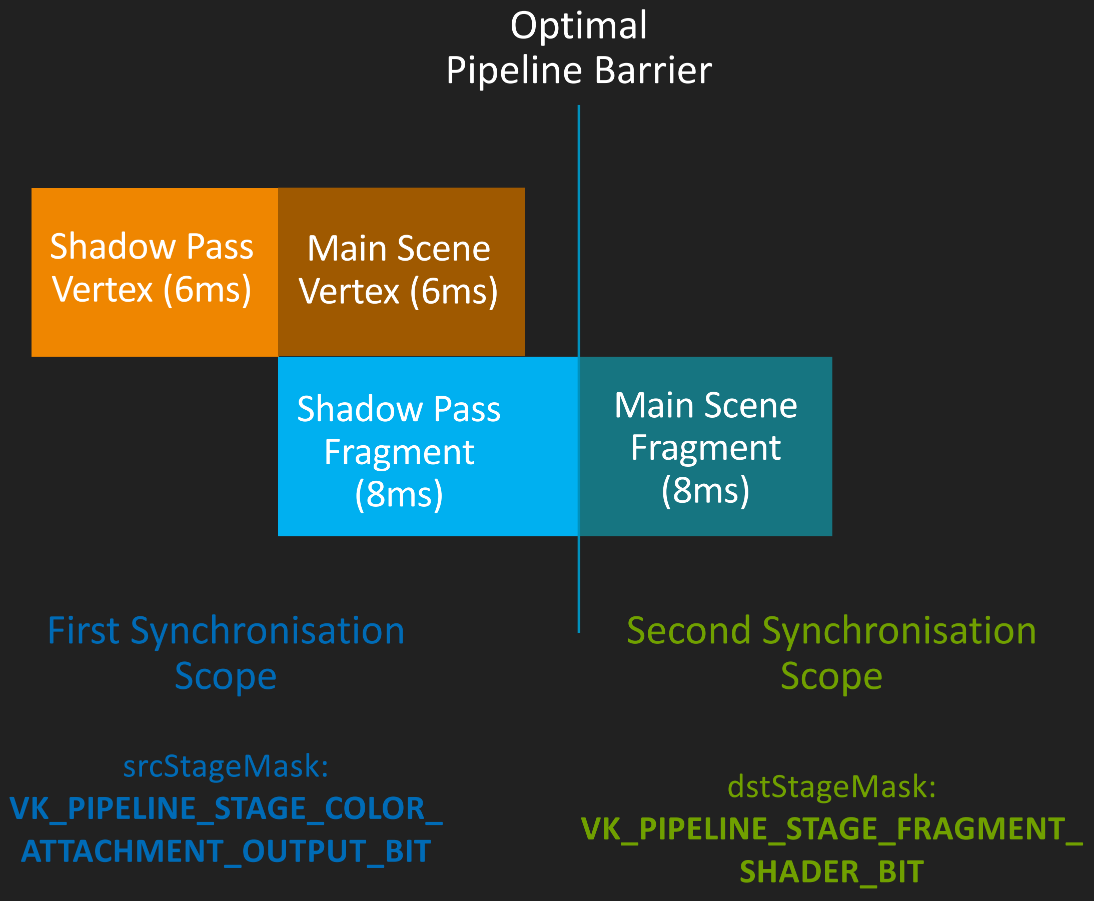
Galaxy GameDev

Vulkan

Pipeline barriers example

- **Optimal case:** We can modify the stage masks and allow the main scene vertex work to overlap the shadow-pass fragment work.

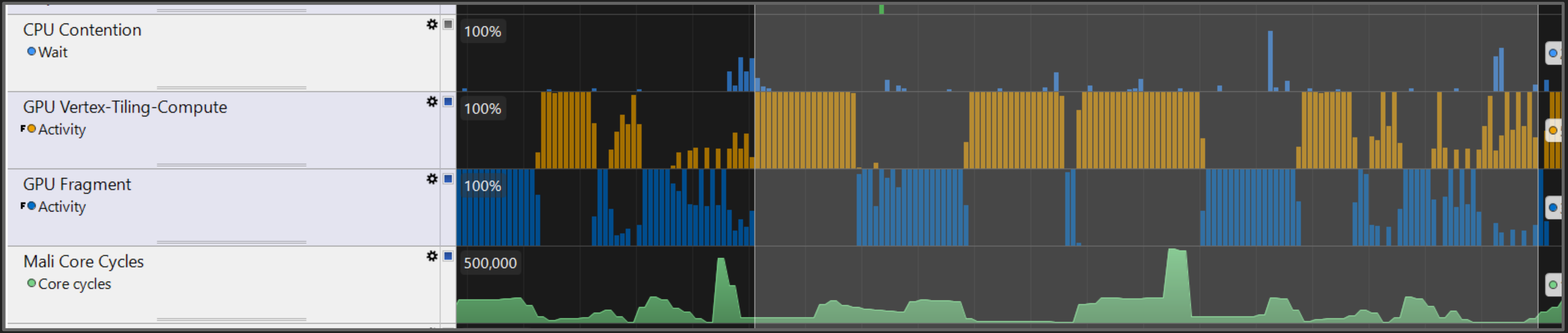
- Results in **6ms** saving! ALU-dependent vertex operations can run in parallel with Texture-dependent fragment operations


Optimal
Pipeline Barrier
Shadow Pass Vertex (6ms)
Main Scene Vertex (6ms)
Shadow Pass Fragment (8ms)
Main Scene Fragment (8ms)
First Synchronisation Scope
Second Synchronisation Scope
srcStageMask:
VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT
dstStageMask:
VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT

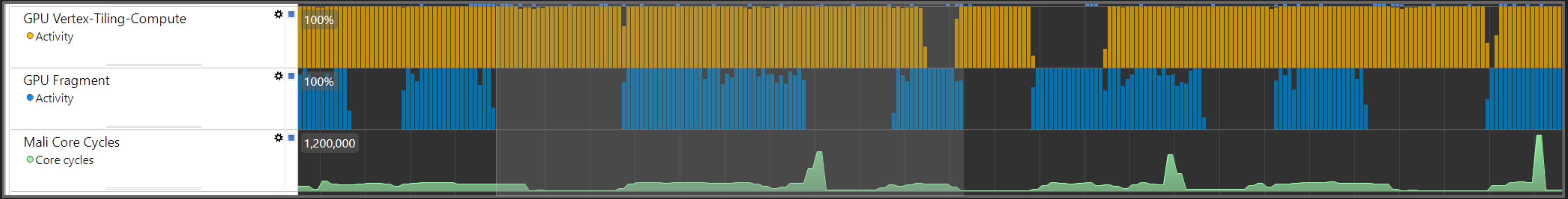
Galaxy GameDev

Vulkan®

Before – General pipeline barriers



After – Optimal per-pass pipeline barriers



***Frame-time: 40ms - 56% Performance increase -** with one line of code!*

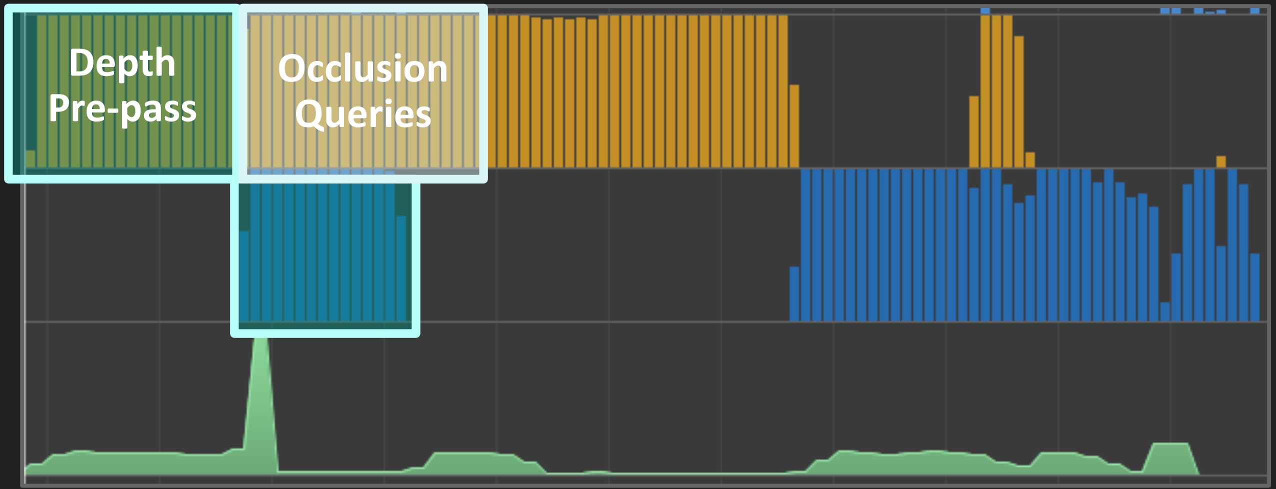
Galaxy GameDev

Vulkan®

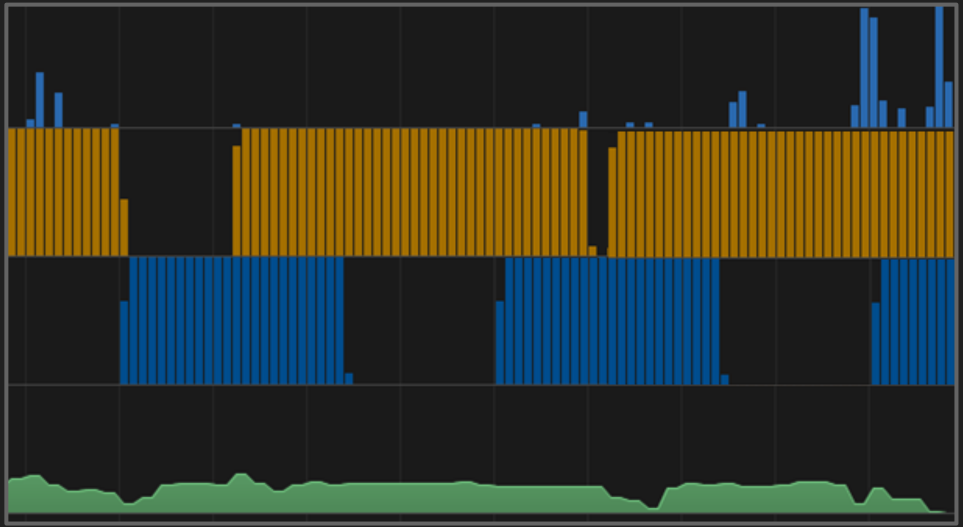
Further Pipeline Optimization – Removing Render Passes

- **High Vertex Load** – Vertex work expensive on tile-based GPUs
- AAA Engine ported to Android from PC/Console – Non-optimal for mobile HW

Standard Trace



Removing Depth pre-pass



Potential performance: **35 fps**!

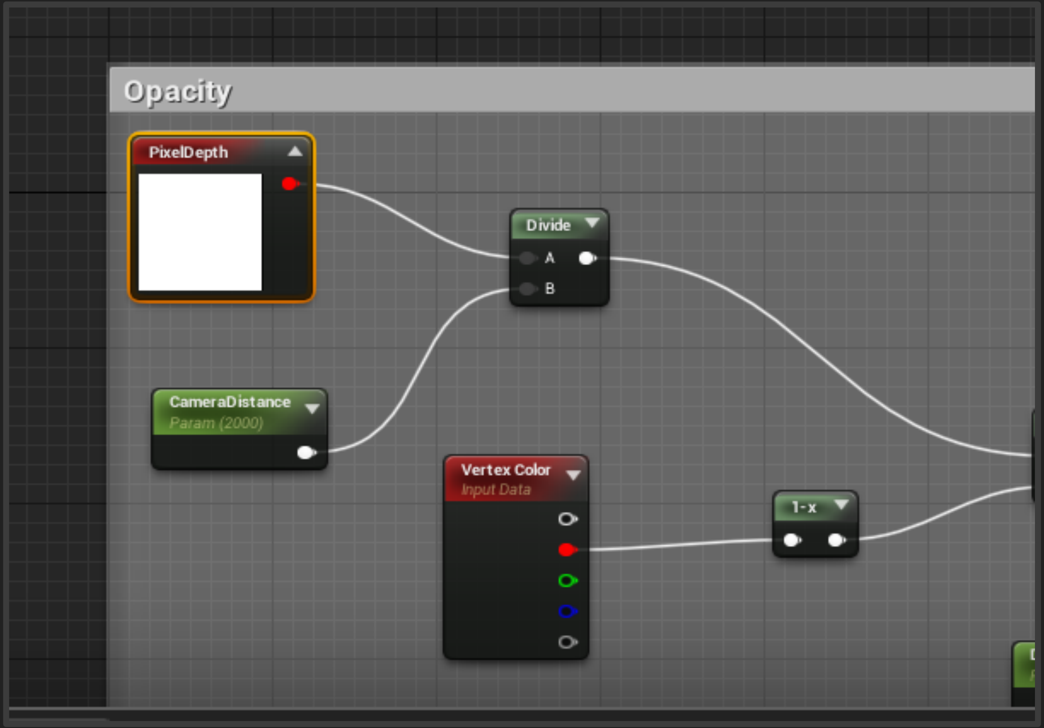
(62.5ms -> 28ms/frame)

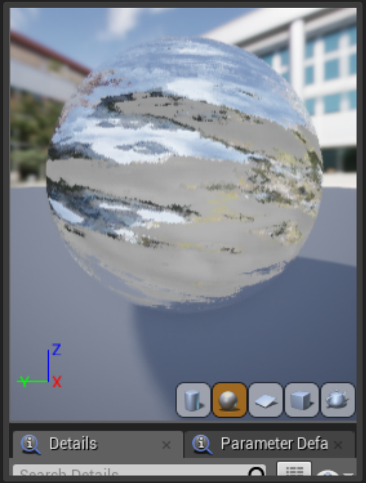
Subpasses

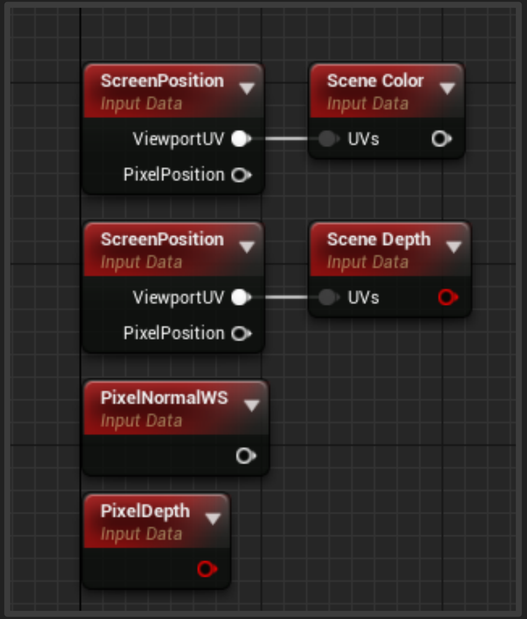
Optimising rendering for memory bandwidth

Useful Vulkan Features - Subpasses

- Allows efficient performing of additional render workload where on-tile frame contents are preserved
  - Large bandwidth savings
  - Avoid GPU Idle time spent storing and loading framebuffer data to main memory
  - Potential power saving and performance increases
  - Use of transient attachments and lazy memory allocation







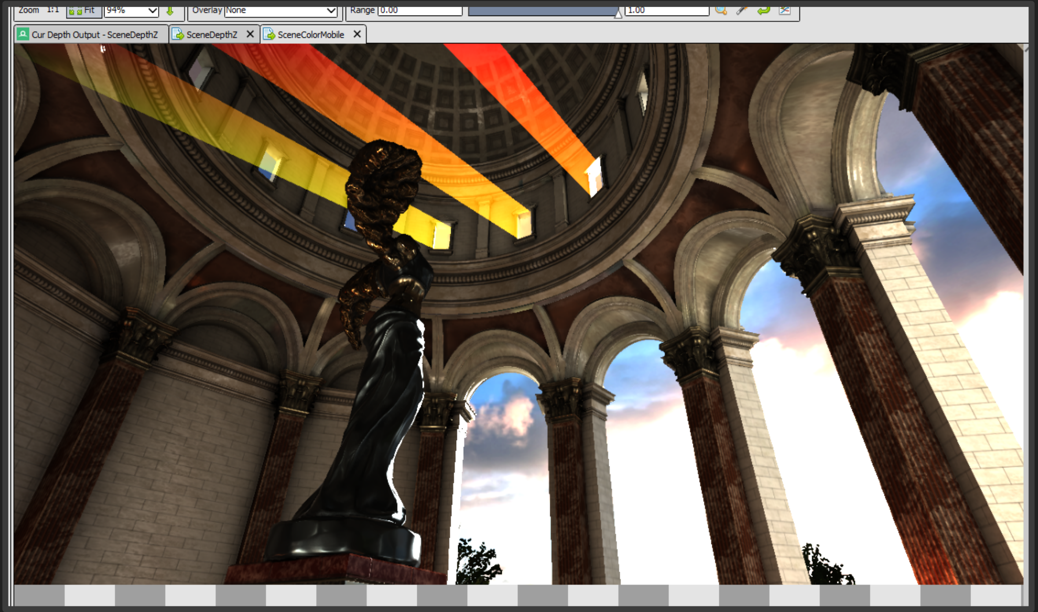
Subpass Viability

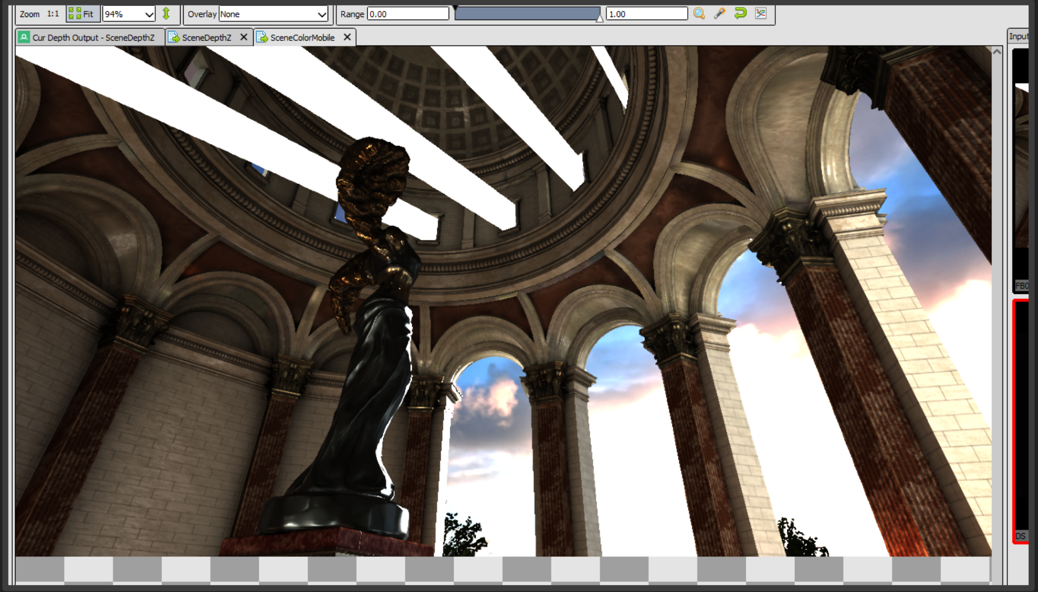
- *shaders in next render pass only sample local framebuffer data*
- *Next render pass uses the same framebuffer attachments*
- *Material nodes e.g.* ***PixelDepth, SceneDepth*** *and* ***SceneColor*** **could* imply subpass compatibility!*

Note: *If we need sparse sampling of a framebuffer, we cannot benefit from subpasses.*

Note: *Many UE4 "Translucency-pass" shaders only sample the local depth value!*

These are perfect candidates for Subpass optimisation.


Zoom 1:1 Fit 94% Overlay None Range 0.00 1.00


Zoom 1:1 Fit 94% Overlay None Range 0.00 1.00

Generic Shader Compatibility test

- Test sample coordinate against local pixel coordinate – using RenderDoc

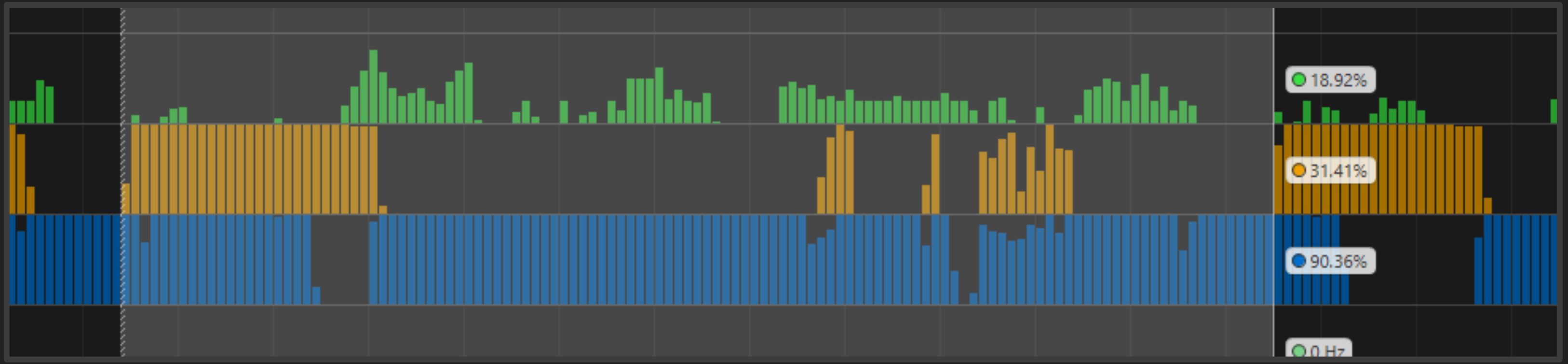
```
vec2 depth_sample_uv = ((v5.xy / v5.ww) *
_18.pu_h[12].xy) + _18.pu_h[12].wz;

float DIFF = depth_sample_uv - vec2(gl_FragCoord.x /
1376.0, gl_FragCoord.y / 720.0);

gl_FragColor.rgb = 1.0-vec3(DIFF);
```

Allows us to determine whether depth sample in “Translucency pass” is ***local***

In this case, god ray shader is ***subpass compatible!***

Galaxy GameDev

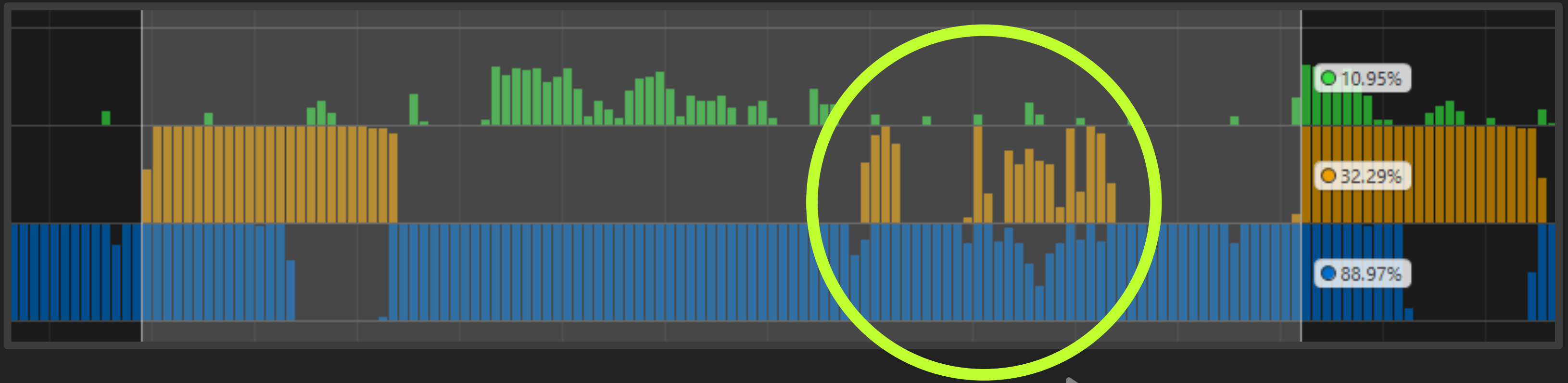
Vulkan.



Standard UE4 Render pass

Depth, colour and stencil targets stored and re-loaded

GPU spends ~1.0ms idle



Using Subpasses

Depth, colour and stencil remain on-tile

GPU spends ~0.15ms idle

Bandwidth saving of 700MB/s

Reduced Fragment Idle Time by ~1ms!

Galaxy GameDev

Vulkan.

Subpass Performance Results

	FPS	CPU	GPU
Default – No additional Subpasses	52	11.1%	97%
Using Subpasses (No Depth and Color load/store)	55	12.4%	97%

~6% Performance increase *in GPU fragment-bound use case on* ***Galaxy S9****!*

Vulkan Tips and Tricks

Quick points for optimisation

Galaxy GameDev

Vulkan®

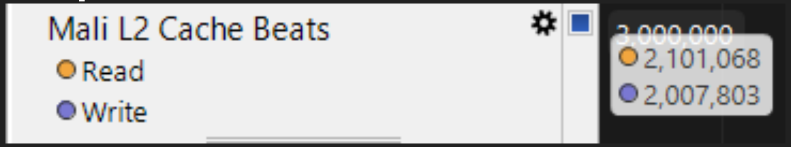
Load and Store Appropriately

- LOAD_OP_LOAD will read the attachment data in system memory into the tile buffer
  - Costs a lot of bandwidth
- LOAD_OP_CLEAR & LOAD_OP_DONT_CARE will set the clear value in the tile buffer directly
  - Costs no bandwidth
- STORE_OP_STORE will write the attachment back out to system memory
  - Costs a lot of bandwidth
- STORE_OP_DONT_CARE writes nothing out
  - Costs no bandwidth

Clear Efficiently

- Don’t ever use vkCmdClearColorImage or vkCmdClearDepthStencilImage!!
  - Wastes Bandwidth unnecessarily
  - Use loadOp Clear at beginning of renderpass
  - Use vkCmdClearAttachments mid renderpass

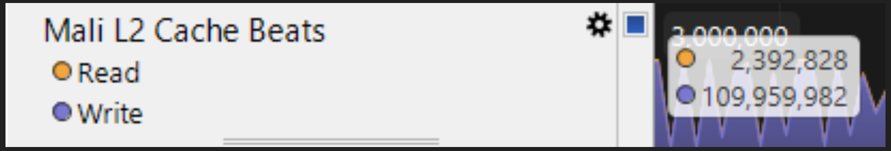
Optimal Clear 62.7MB/s



Mali L2 Cache Beats
● Read
● Write

3,000,000
● 2,101,068
● 2,007,803

Unoptimal Clear 1.7GB/s



Mali L2 Cache Beats
● Read
● Write

3,000,000
● 2,392,828
● 109,959,982

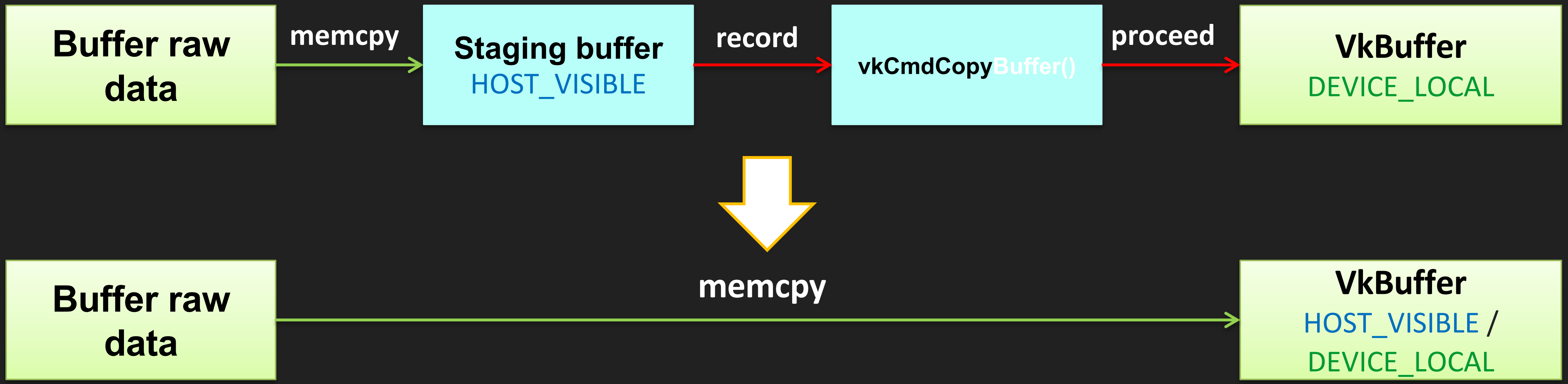
Transient Attachments

- Attachments that exist solely in tile memory
  - Doesn’t need to be backed by memory
  - Reduces memory footprint

- Required flags
  - imageUsage = VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT
  - memoryProperty = VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT

Upload buffer data to GPU

- No need to use staging buffers for copying CPU buffer data to GPU
- UMA on mobile devices
- Still required for uploading image data to GPU



Buffer raw data

memcpy

Staging buffer
HOST_VISIBLE

record

vkCmdCopyBuffer()

proceed

VkBuffer
DEVICE_LOCAL

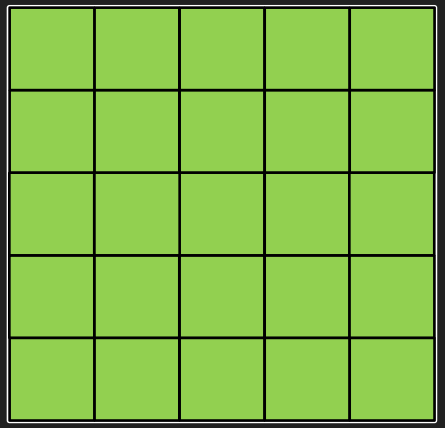
Buffer raw data

memcpy

VkBuffer
HOST_VISIBLE /
DEVICE_LOCAL

Tiling (of images)

- Raster order doesn’t usually suit textures
- TILING_LINEAR is useful for frequent updates
- Use TILING_OPTIMAL for better GPU cache access



TILING_LINEAR

TILING_OPTIMAL

New Vulkan Feature – Depth Stencil Resolve

- MSAA is cheap on mobile tile-based architectures
- Resolving MSAA depth targets is currently expensive
- New Vulkan Extension to enable efficient on-tile resolve

What this Means:

- Depth-dependent Renderpass effects such as:

  Translucency pass - decal projection - depth of field - god rays - fog

- Possible with MSAA enabled at no additional performance cost!

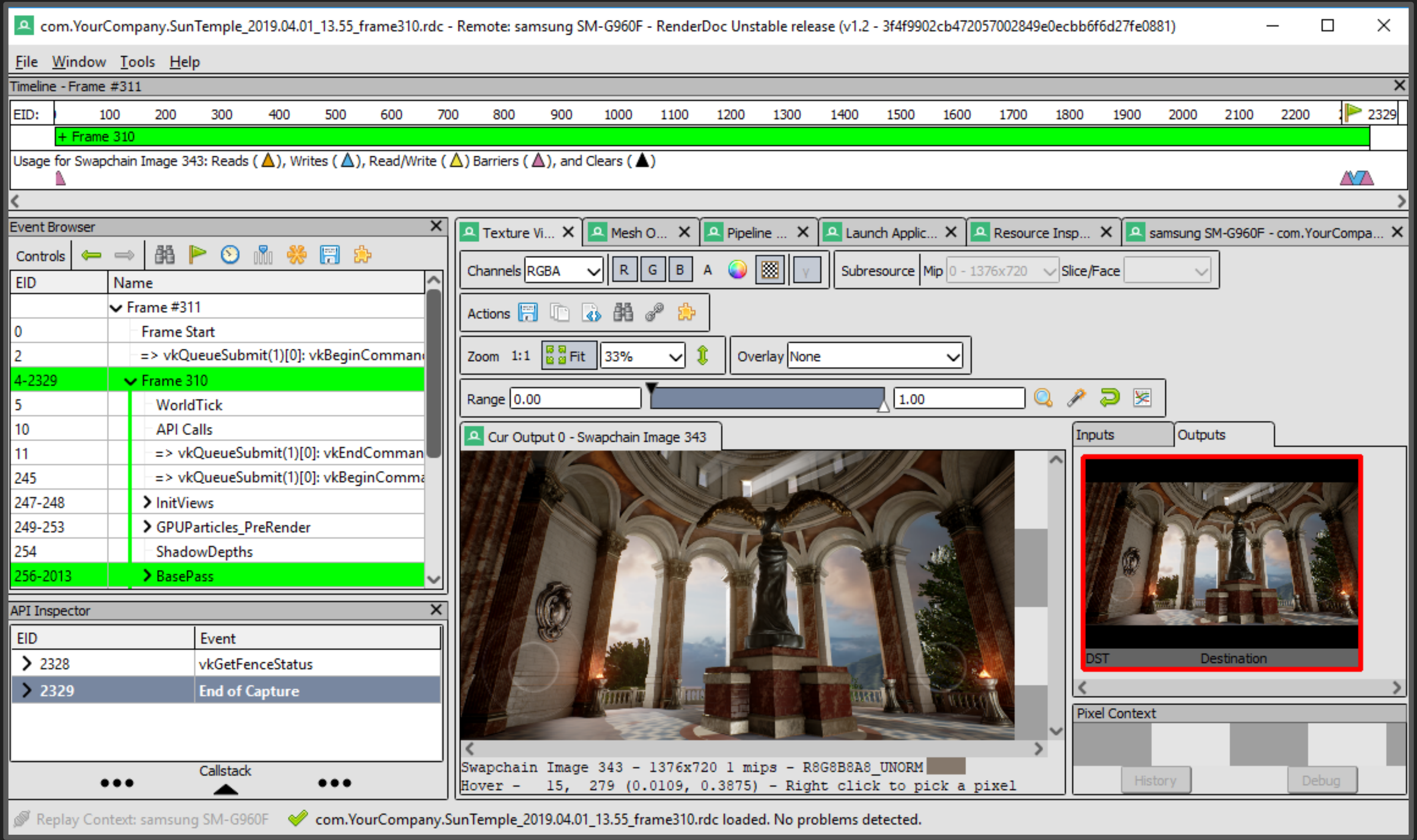
Android Tools

Best tools for the job

Galaxy GameDev

Vulkan®

Tools - RenderDoc



com.YourCompany.SunTemple_2019.04.01_13.55_frame310.rdc - Remote: samsung SM-G960F - RenderDoc Unstable release (v1.2 - 3f4f9902cb472057002849e0ecbb6f6d27fe0881)

Usage for Swapchain Image 343: Reads (▲), Writes (▲), Read/Write (▲) Barriers (▲), and Clears (▲)

EID	Name
	Frame #311
0	Frame Start
2	=> vkQueueSubmit(1)[0]: vkBeginComman
4-2329	Frame 310
5	WorldTick
10	API Calls
11	=> vkQueueSubmit(1)[0]: vkEndComman
245	=> vkQueueSubmit(1)[0]: vkBeginComma
247-248	InitViews
249-253	GPUParticles_PreRender
254	ShadowDepths
256-2013	BasePass

EID	Event
2328	vkGetFenceStatus
2329	End of Capture

Channels: RGBA

Subresource: Mip 0 - 1376x720

Slice/Face

Zoom: 33%

Overlay: None

Range: 0.00 – 1.00

```
Swapchain Image 343 - 1376x720 1 mips - R8G8B8A8_UNORM
Hover -   15,  279 (0.0109, 0.3875) - Right click to pick a pixel
```

Replay Context: samsung SM-G960F

✓ com.YourCompany.SunTemple_2019.04.01_13.55_frame310.rdc loaded. No problems detected.

Full static frame analysis.

Verify API usage:

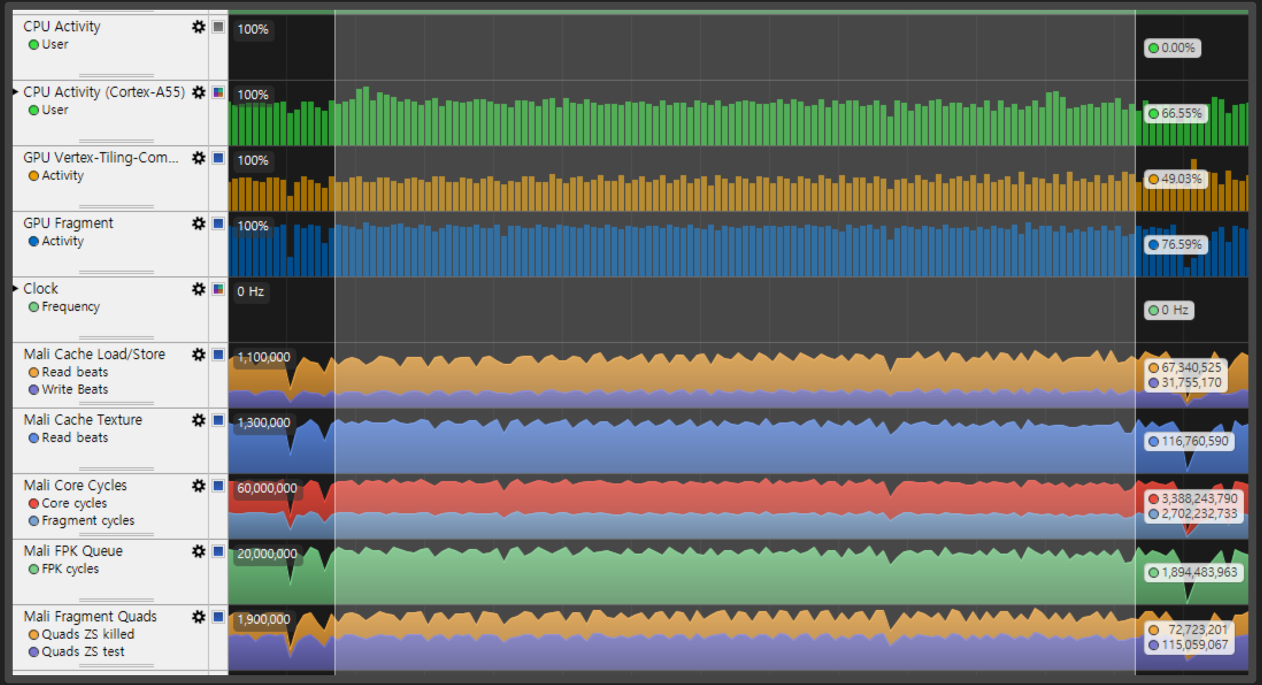
- *Draw calls*
- *Renderpasses*
- *Barriers*
- *Resources*

Step-through scene

Informed content optimisation!

Works very well with Vulkan!

Tools – Arm Streamline, Snapdragon Profiler



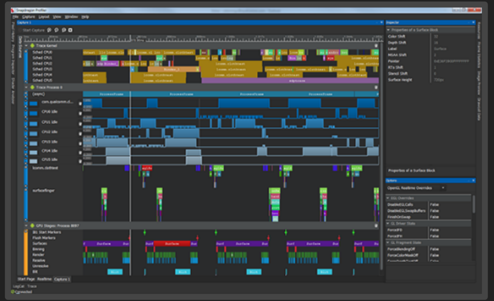
In-depth hardware analysis

Counters:

- *Vertex Activity*
- *Fragment Activity*
- *CPU core utilisation*
- *Memory analysis*

High-resolution data

Identify bottlenecks!



Visually analyse render workload execution

Improve app performance with high-quality Vulkan use

Galaxy GameDev

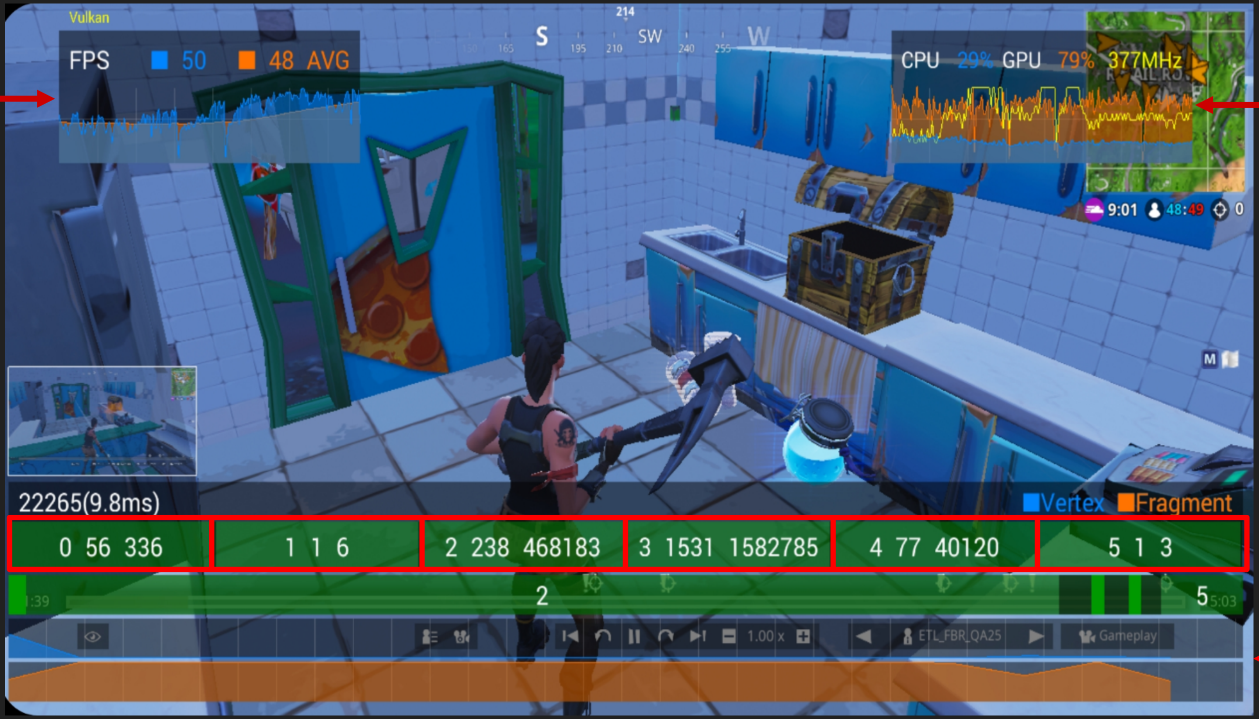
Vulkan.

GPU Watch

- Performance monitoring tool
  - Direct result on the screen
  - Support Vulkan/OpenGL ES

FPS info.

Screenshot of the captured frame



```
22265(9.8ms)
0  56  336 | 1  1  6 | 2  238  468183 | 3  1531  1582785 | 4  77  40120 | 5  1  3
```

CPU/GPU utilization

GPU Profiling information
- Renderpass count
- GPU activity

Thank You

- Correct Pipeline Barrier staging
- Use Subpasses where you can
- Load & Store Appropriately

- Use Transient Attachments
- Clear Efficiently