idc16

Imagination Developers Connection

Efficient rendering with Vulkan on PowerVR

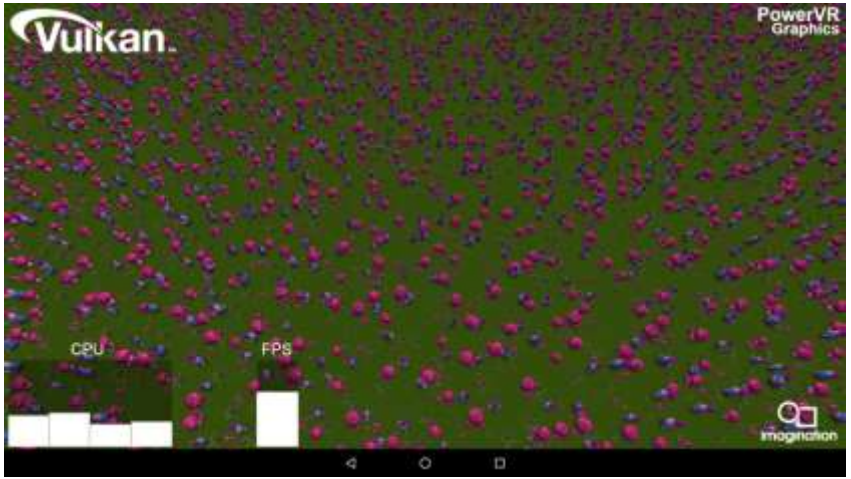
March 2016

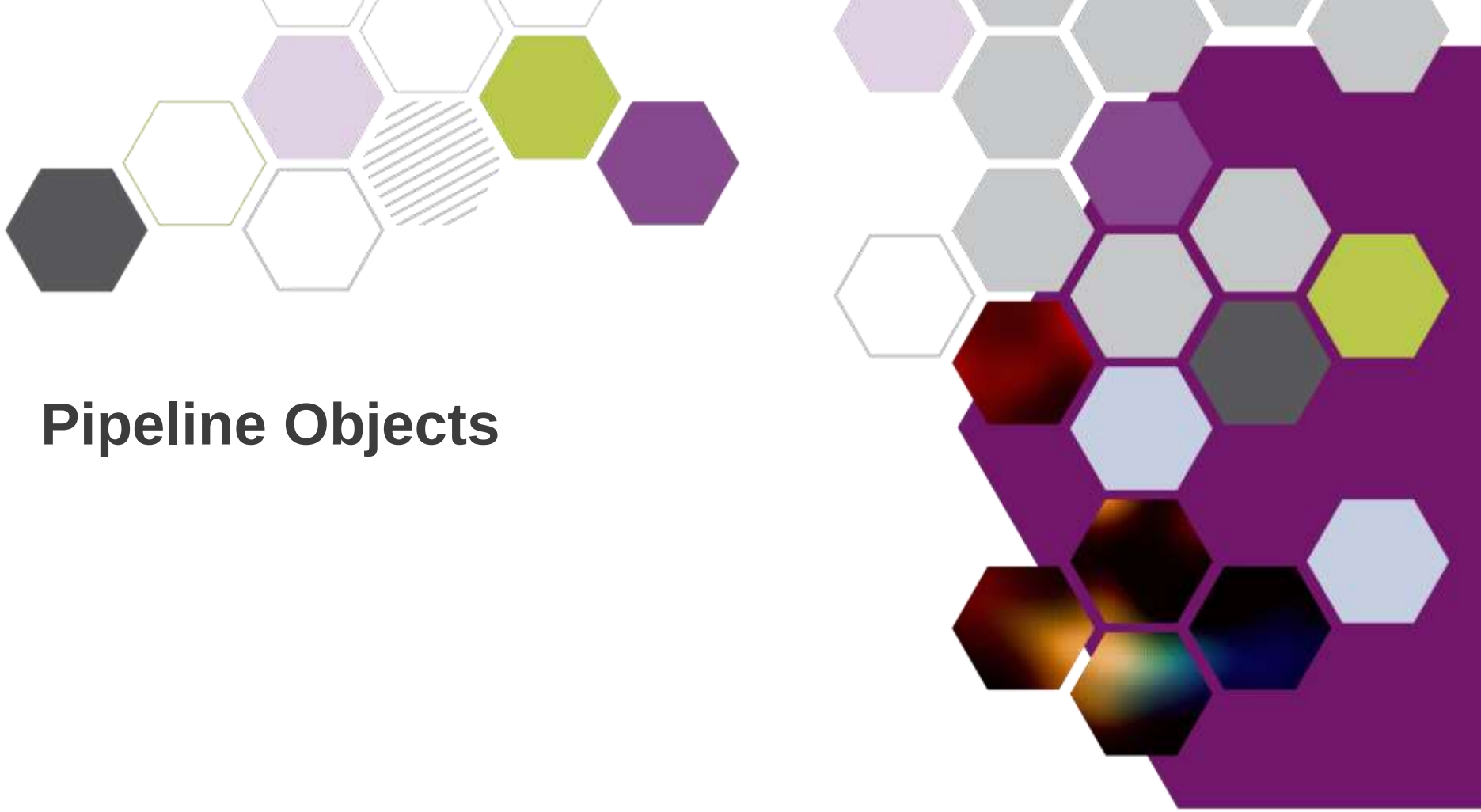
Introduction

- **In-depth look at some features of Vulkan with:**
  - Focus on usage patterns

- **Examples using Gnome Horde**

- **Pipeline Objects**
- **Command Buffers**
- **Explicit Synchronization**
- **Render Passes**





Pipeline Objects

Pipeline Objects

What are they?

- **Describes execution of a draw command**
  - Contains shaders for programmable stages
  - Contains configuration of fixed function stages

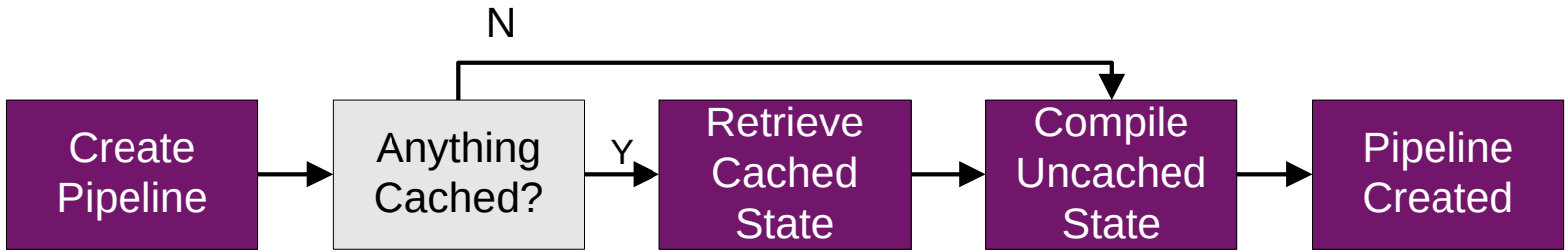
- **Allows state to be fully optimised up front**
  - State baked into HW consumable format
  - No need to do late patching/recompilation
  - E.g. Blending modes become shading instructions

Pipeline Objects

Pipeline Caches

- **Applications may need many variants**
  - Same code, different state
  - Creating those variants could take a long time

- **Pipeline Caches allow re-use of already baked state**
  - Creation checks cache first



Pipeline Objects

Creation

```
VkVertexInputBindingDescription vertexBindings[];
VkVertexInputAttributeDescription vertexAttributes[];
VkPipelineVertexInputStateCreateInfo vertexInput;
VkPipelineInputAssemblyStateCreateInfo inputAssembly;
VkViewport viewports[];
VkRect2D scissors[];
VkPipelineViewportStateCreateInfo viewport;
VkPipelineRasterizationStateCreateInfo rasterisation;
VkPipelineDepthStencilStateCreateInfo depthStencil;
VkPipelineColorBlendAttachmentState blends[];
VkPipelineColorBlendStateCreateInfo colourBlend;
VkPipelineShaderStageCreateInfo stages[];
```

Pipeline Objects

Creation and binding

```
auto gnomeMeshes = loadPodLodMeshes(renderer, "assets/gnome", "body", 7)
vertexBindings = gnomeMeshes[0].vertexBindings

vkCreateGraphicsPipeline(pipelineCache, createInfo)

vkCmdBindPipeline(cmdb, pipeline)
```

Pipeline Objects

Summary

- No big state machine
- Better maps to how a modern GPU works

- Use pipeline caches

Command Buffers

Command Buffers

What are they?

- **Records a set of graphics commands**
  - Draw commands are not immediately submitted to the GPU
  - Dispatches, Copies, Barriers are all recorded first

- **Stores commands until reset or freed**
  - Can be created once and re-used many times
  - Immutable, but can modify resources
  - Explicit control over discarding commands

Command Buffers

Multi-threaded recording

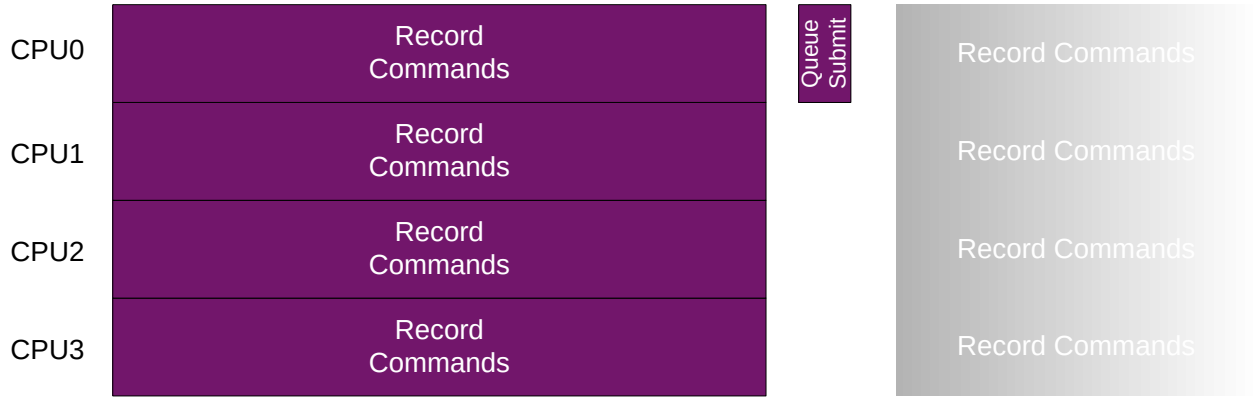
- **Commands can be recorded on multiple threads**
  - Take advantage of multi-core CPUs
  - Further ensures keeping to your frame budget

- **Re-use command buffer memory**
  - Recording requires memory
  - Reset existing memory – don't reallocate

Command Buffers

Submission

- **No further processing**
  - Just kicks work to the GPU
  - Relatively cheap operation



Command Buffers

Recording

```
vkBeginCommandBuffer(cmdb)
vkCmdBindDescriptorSet(cmdb, cameraData)

for each object:
        vkCmdBindPipeline(cmdb, object.pipeline)
        vkCmdBindIndexBuffer(cmdb, object.ibo)
        vkCmdBindVertexBuffer(cmdb, object.vbo)
        vkCmdBindDescriptorSet(cmdb, object.texture)
        vkCmdBindDescriptorSet(cmdb, perObjectData, objectOffset)
        vkCmdDrawIndexed(cmdb)

vkEndCommandBuffer(cmdb)
```

Command Buffers

Summary

- **Separation of creation and submission**

- **Spread recording over multiple threads**
- **Re-use command buffers where possible**

Explicit Sychronization

Explicit Synchronization

Overview

- **Three types of synchronization**
  - Events and Barriers
  - Semaphores
  - Fences

Explicit Synchronization

Synchronization granularity

- **Execution and memory dependencies**
  - Dependencies between tasks
  - Dependencies on memory access

- **Vulkan has much finer granularity**
  - Specify pipeline stages for execution dependencies
  - Specify memory dependencies per-resource
  - Specify usage for resources

Explicit Synchronization

Events and Pipeline Barriers

- **Events and Pipeline Barriers**
  - Within and between command buffers
  - Fine grained synchronization
- **Pipeline barriers act at a specific point**
  - vkCmdPipelineBarrier
- **Events have a beginning and end**
  - vk{Cmd}SetEvent/vk{Cmd}WaitEvent
  - Can be waited on or triggered by CPU

Explicit Synchronization

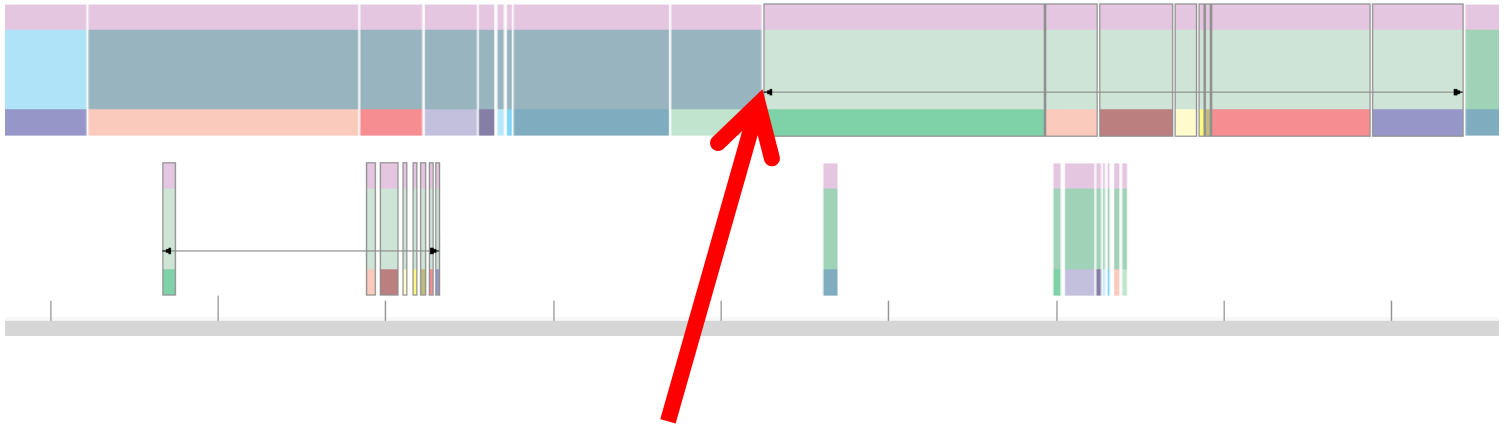
Semaphores, Fences

- **Semaphores**
  - Synchronize different queues
  - No CPU interaction
  - Describes coarse dependencies

- **Fences**
  - Triggered when a queue operation completes
  - Can be waited on the CPU
  - Describes coarse dependencies
  - Useful to know when “a frame” is completely done.

Explicit Synchronization

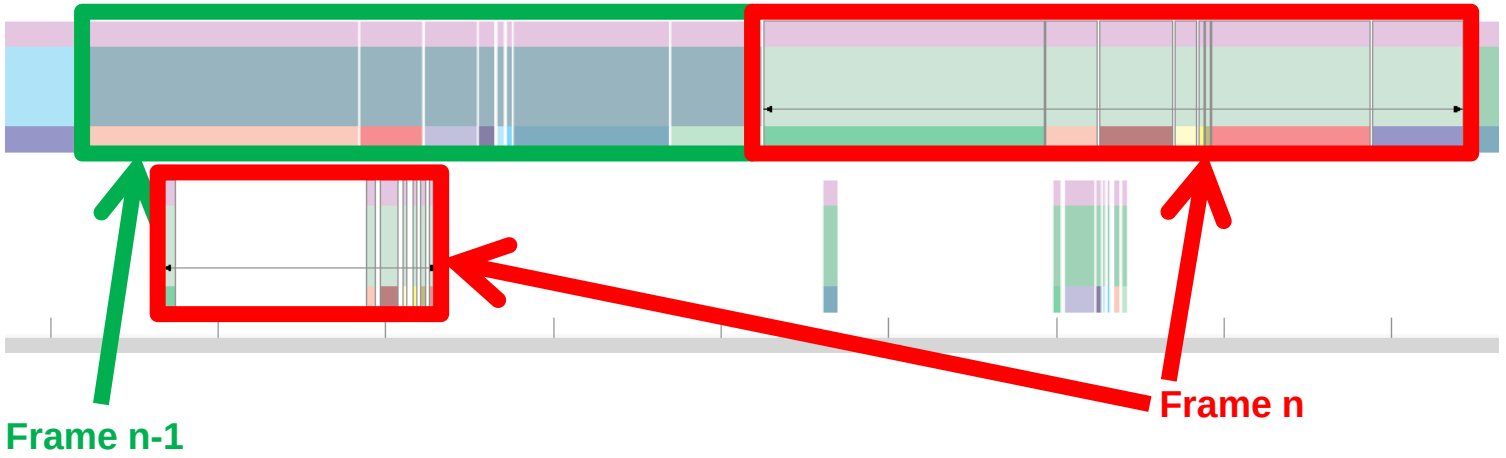
Tiling/Rasterization Overlap



Barrier allows vertex processing to occur in parallel

Explicit Synchronization

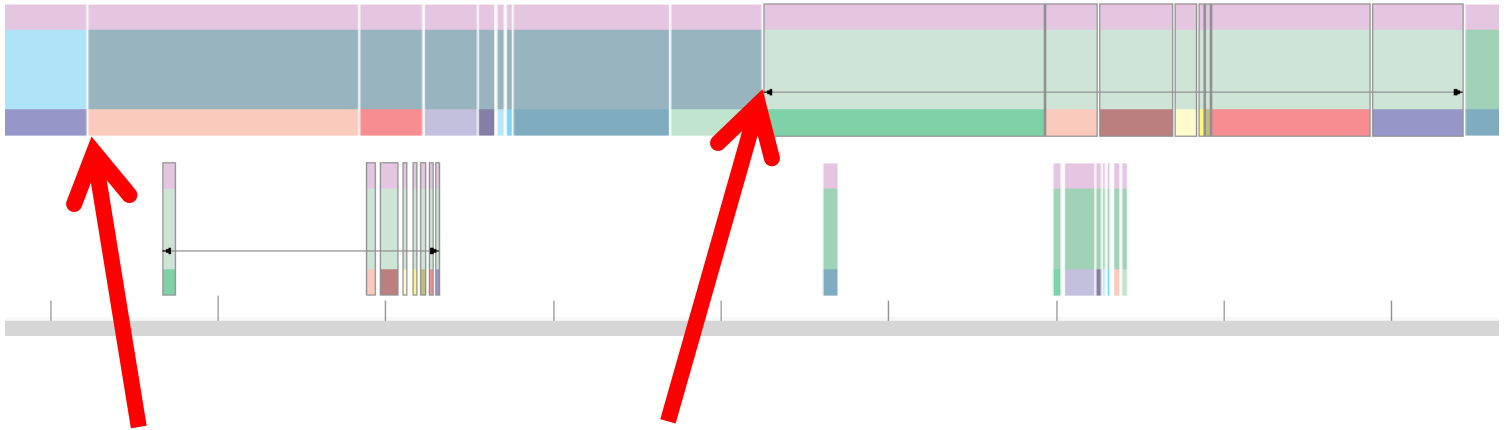
Tiling/Rasterization Overlap



We can process the vertices from frame n-1 while processing the fragments from frame n

Explicit Synchronization

Resource updates



Barrier is also required for memory dependencies

Explicit Synchronization

Resource updates

```
uint32_t currentBuffer = numSwaps % swapchainFences.size()
vkQueueSubmit(submitInfo, swapchainFences[currentBuffer])
if(numSwaps > swapchainFences.size() - 1) {
    uint32_t fenceToWaitFor = (numSwaps + 1) % swapchainFences.size()
    vkWaitForFences(swapchainFences[fenceToWaitFor])
    vkResetFences(swapchainFences[currentBuffer])
}
```

Explicit Synchronization

Resource updates

```
uint32_t currentBuffer = numSwaps % swapchainFences.size()
vkQueueSubmit(submitInfo, swapchainFences[currentBuffer])
if(numSwaps > swapchainFences.size() - 1) {
    uint32_t fenceToWaitFor = (numSwaps + 1) % swapchainFences.size()
    vkWaitForFences(swapchainFences[fenceToWaitFor])
    vkResetFences(swapchainFences[currentBuffer])
}
```

Explicit Synchronization

Summary

- **More difficult to use than GL**
- **More possibilities to optimize**

- **Synchronize exactly what you need to**
  - No more
  - No less

Render Passes

Render Passes

What are they?

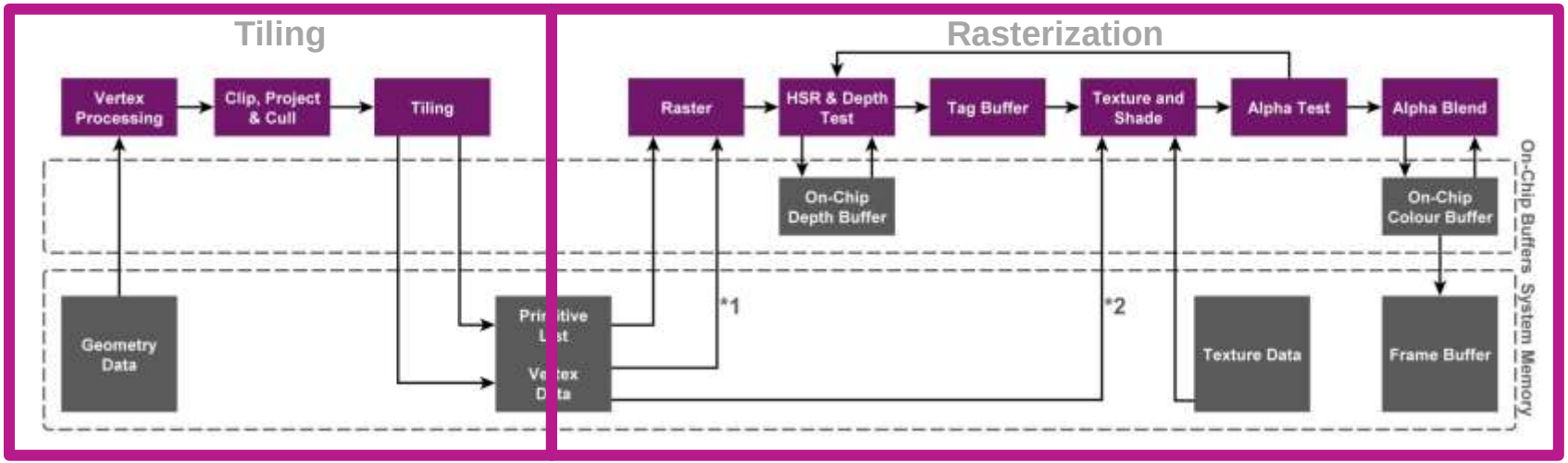
- **Describes a batch of rendering tasks**
  - A number of draw calls
  - Multiple subpasses to multiple render targets
  - Dependencies between subpasses
  - How attachments are loaded/stored

- **Avoids pipeline bubbles**
  - Describes exact dependencies between subpasses
  - No implicit ordering of pass execution
  - Forward dependencies only

Render Passes

Tile Based Architectures

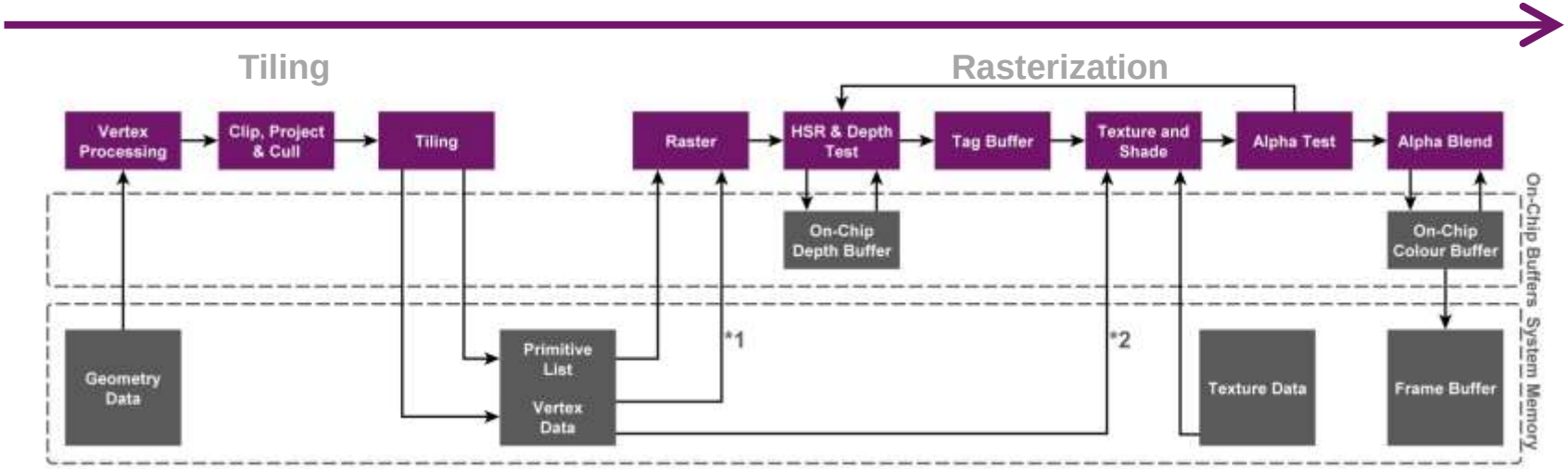
- **Two distinct stages**
  - Tiling (geometry processing and binning)
  - Rasterization (fragment processing)



Render Passes

Forwards Dependencies

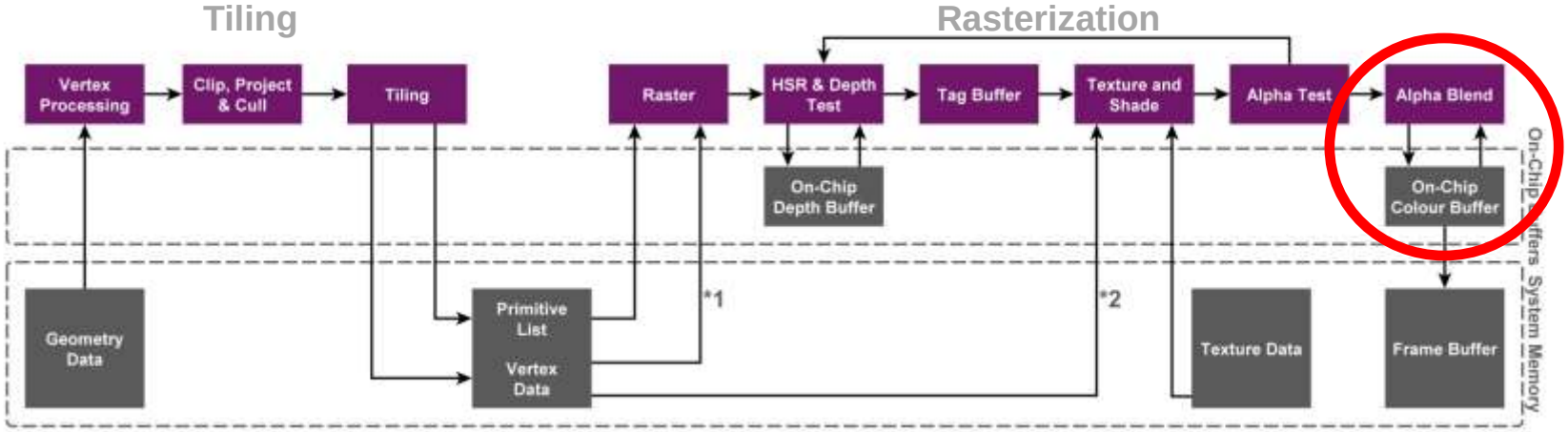
- **Dependencies between subpasses must move forward**
  - Cannot disrupt pipelined flow of hardware
  - E.g. can't wait in tiling pass for rasterization work



Render Passes

Pixel local access

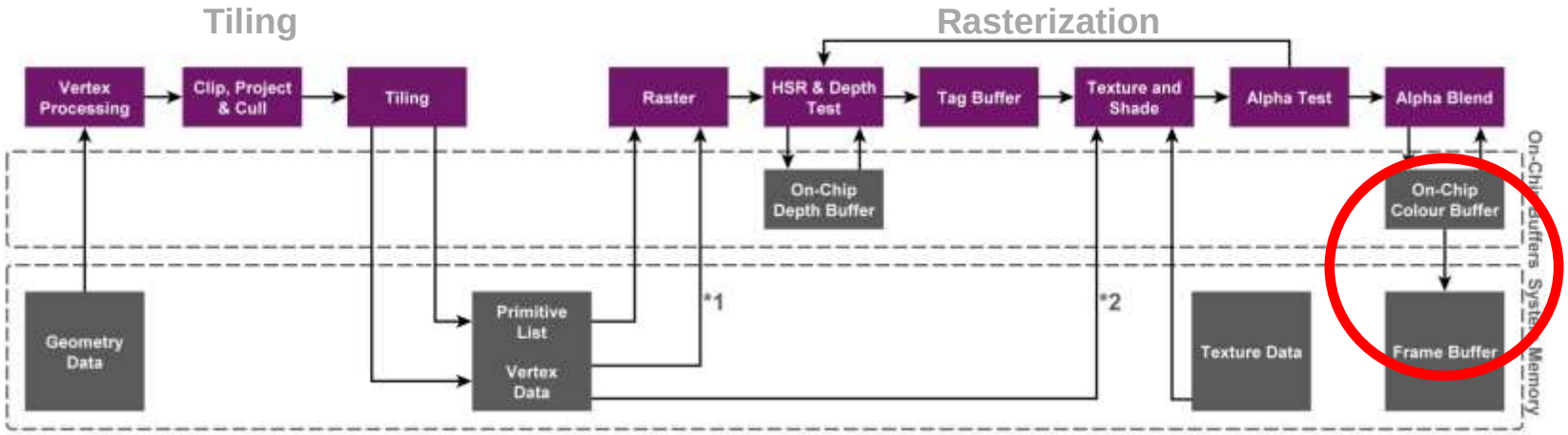
- **Access pixels previously written at same location**
  - Per-pixel location, progress is always forwards



Render Passes

Load/Store Operations

- **Explicitly state what to do with attachment data**
  - Tilers don't have to write out data
  - Reduces bandwidth if not used



Render Passes

Load store ops

No more weird driver hints:

```
glClear()
glInvalidateFramebuffer()
```

Now it's explicit:

```
VkAttachmentDescription
    .loadOp =  VK_ATTACHMENT_LOAD_OP_CLEAR,
    .storeOp = VK_ATTACHMENT_STORE_OP_STORE,
    .stencilLoadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE,
    .stencilStoreOp = VK_ATTACHMENT_STORE_OP_DONT_CARE
```

Render Passes

Subpass loads

```
layout(input_attachment_index = 0, set = 0, binding = 0) uniform subpassInput
localMemTexture;


void main() {
        vec4 tex = subpassLoad(localMemTexture)
}
```

Render Passes

Subpass loads

Render Passes

Summary

- **More explicit as to what the GPU is doing**
- **Big advantages for TBDRs**
  - Some advantages for all architectures
- **Use them as much as possible!**

Conclusion

Vulkan on PowerVR

- https://imgtec.com/vulkan
  - Nexus Player Vulkan Android image + Vulkan examples + Vulkan SDK
- **Gnome Horde video:**
  https://www.youtube.com/watch?v=P_I8an8jXuM

Conclusion

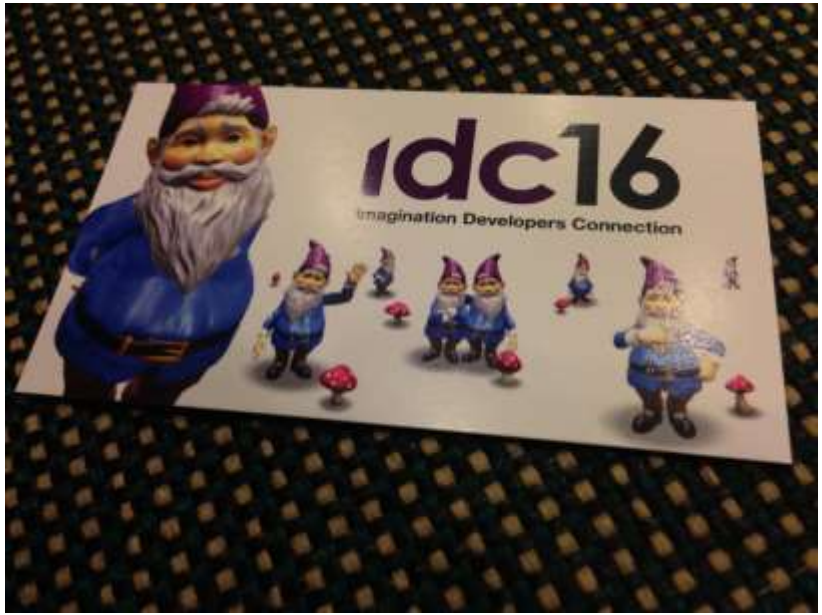
- **Vulkan gives you a lot of control**
  - With great power comes great responsibility
- **Understand GPU architectures**
- **Different way of thinking to older APIs**
- **Follow usage recommendations**

Questions?

Get in touch:

- *Tobias.Hector@imgtec.com*, *Ash.Smith@imgtec.com*
- @TobskiHectov, @sumisuav

Please come and visit us at booth #1902 in the South Hall to see our demos and to collect your very own Vulkan™ Gnome t-shirt!



www.imgtec.com/idc