Vulkan – When and Why

Michael Worcester – Driver Engineer
(michael.worcester@imgtec.com)
30 Feb 2017

www.imgtec.com

Perfect World

- **When?**
  - All the time!
- **Why?**
  - Because Vulkan is pretty dope

Alas – Platform Support

- Hardware requirements – 3.1 minimum
- OEM support
- “Political reasons”

Takes time!

Alas - Maturity

- Drivers are good, but still very young
- Tools and ecosystem
- Third-party libraries and middleware

Takes time!

Alas – Required Skills

- Evolved set of skills required
- Traditional APIs hide hardware complexity
- Need an in-depth knowledge of GPU architecture

Takes time!

Maybe these aren’t negatives at all?

- **Platform support**
  - The platform of the future
- **Maturity**
  - A chance to start (partially) from scratch
- **Skills**
  - Applicable everywhere

If you *can* invest the time, the rewards are great!

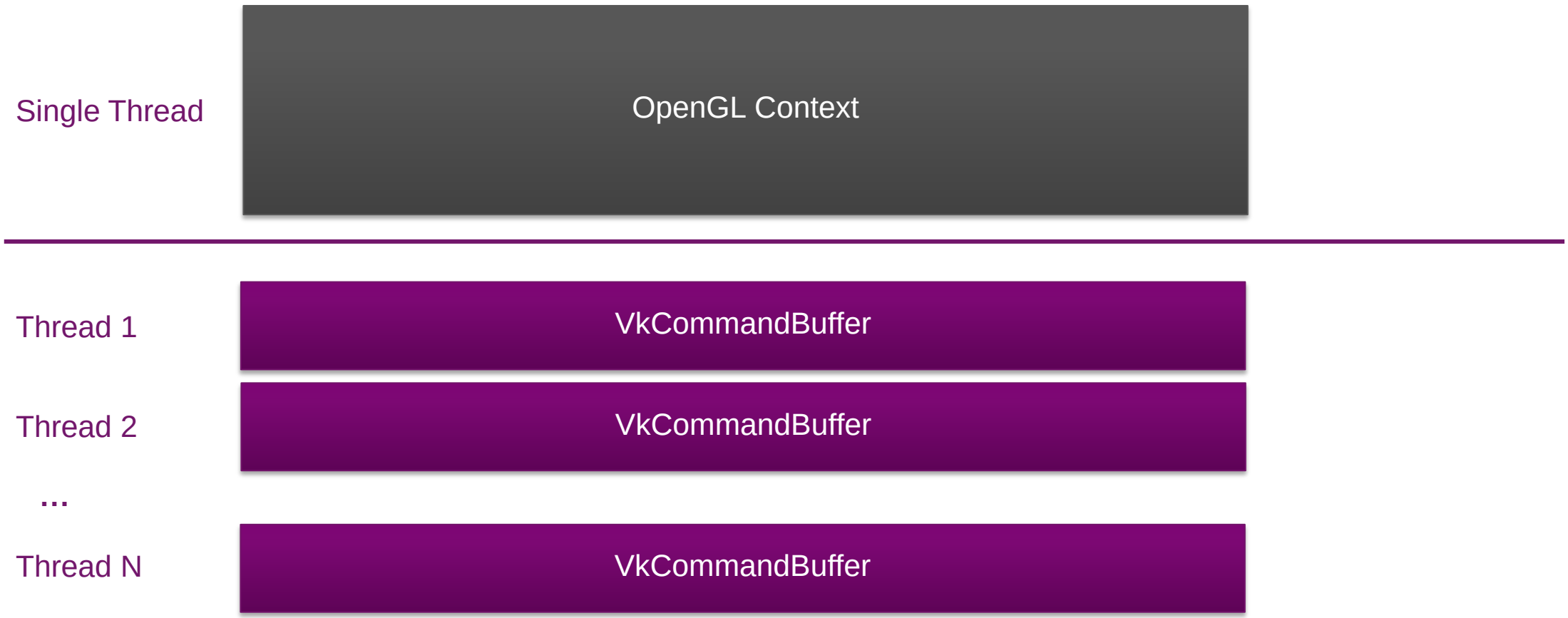
Command Buffers – Deferring the work

- **OpenGL is immediate (ignoring display lists)**
  - Driver does not know how much work is incoming
  - Has to guess
  - Bad!
- **Vulkan splits recording of work from submission of work**
  - Removes guesswork from driver
  - Reducing hitching
  - Helps eliminate unexplained resource usage

Command Buffers – Pooling Resource

- **Command Buffers always belong to a Command Pool**
  - Buffers are allocated from pools
  - Pools provide lightweight synchronisation
  - Pools can be reset, reclaiming all resources
  - Two flavours of pool:
    - Individual reset of command buffers
    - Group reset **only**

Command Buffers – Going wide



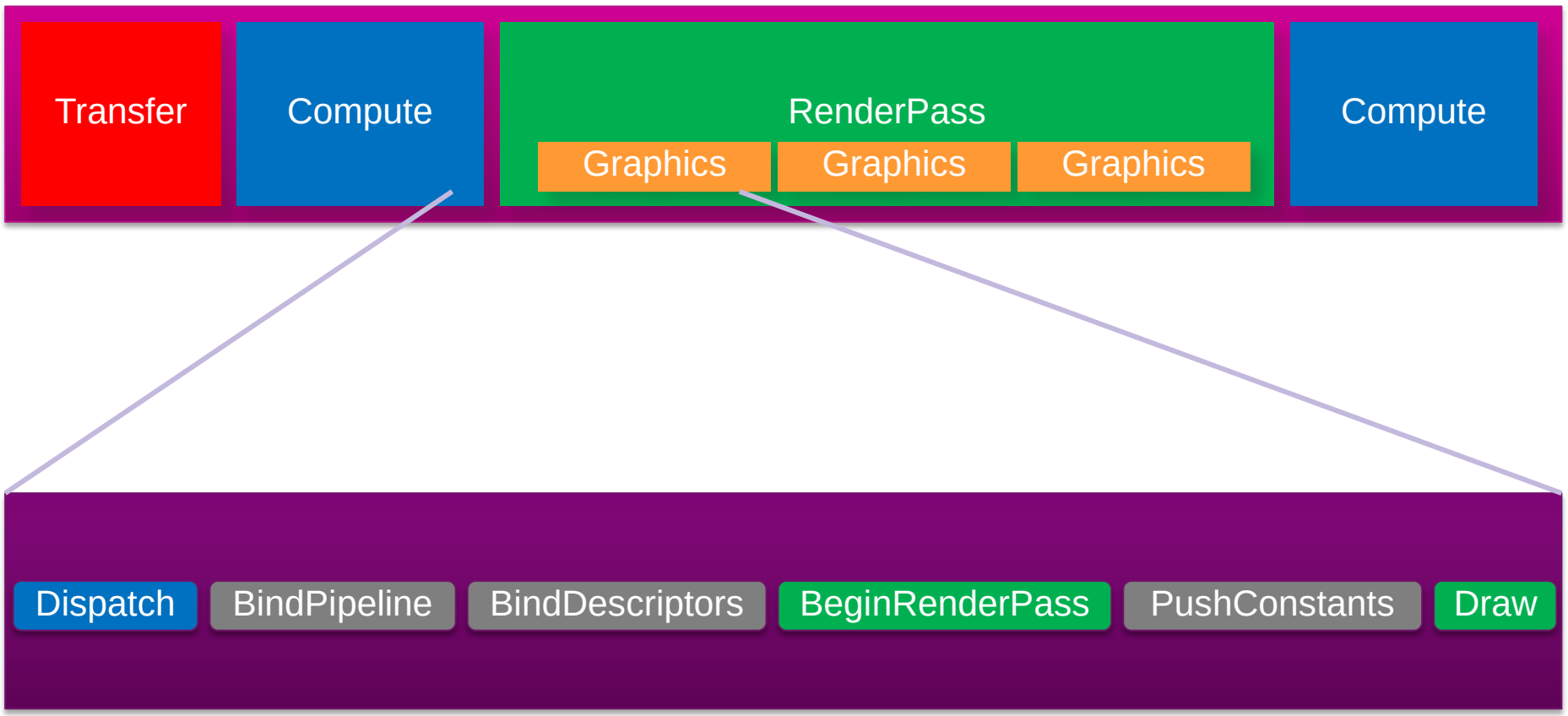
Command Buffers – Command Types

- **Deferred recording of commands**
  - Transfer
  - Graphics
  - Compute
  - Synchronisation

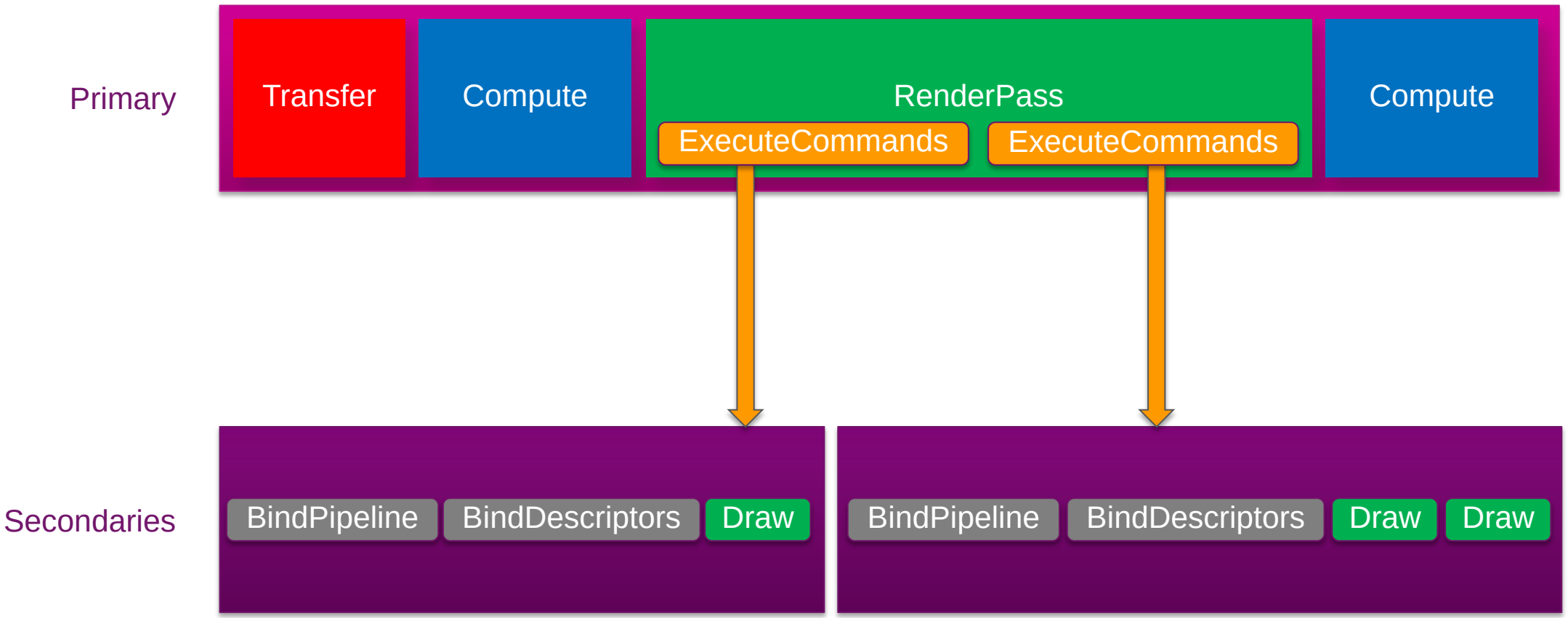
Command Buffers – Transfers

- **Transfer commands are raw copies**
  - However, they can change the *tiling* of an image (this is the only way!)
- **CPU -> GPU**
  - Texture upload
  - Static buffer data
- **GPU -> CPU**
  - Read back of data
- **GPU -> GPU**
  - Pipelined updates of data
  - Mipgen

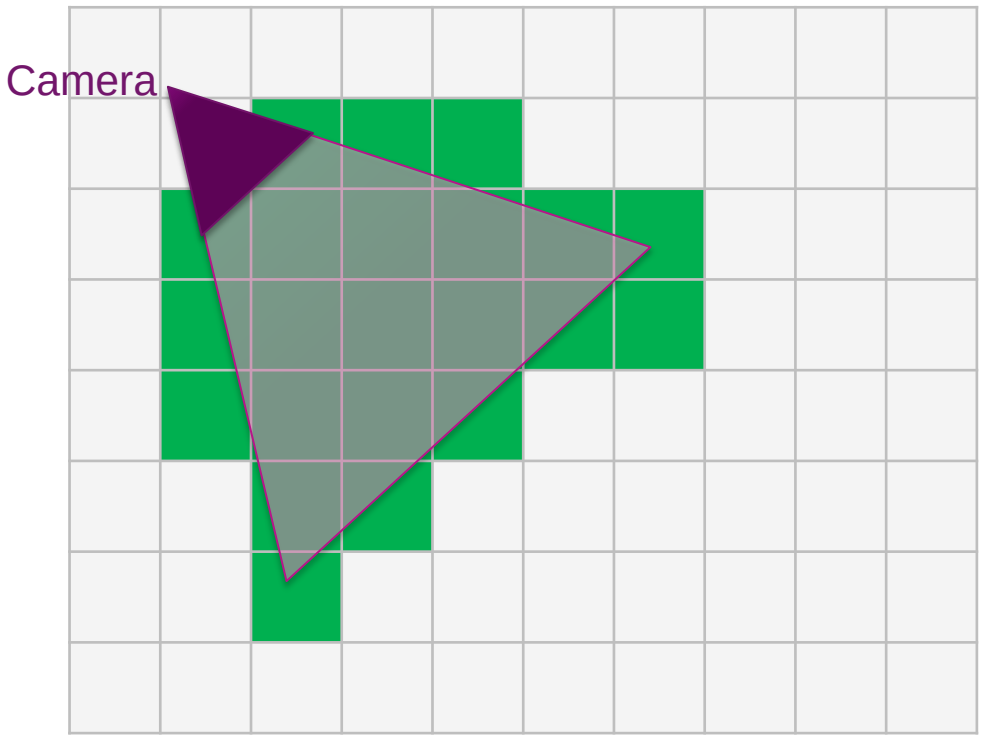
Command Buffers – “Inside” or “Out”



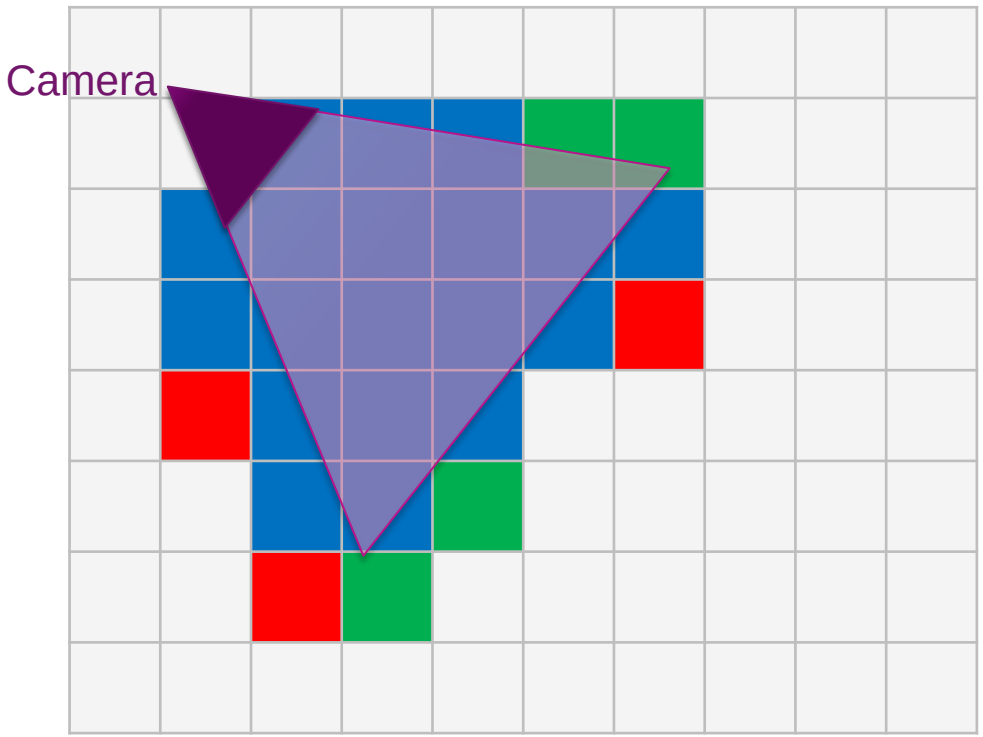
Command Buffers – Secondaries



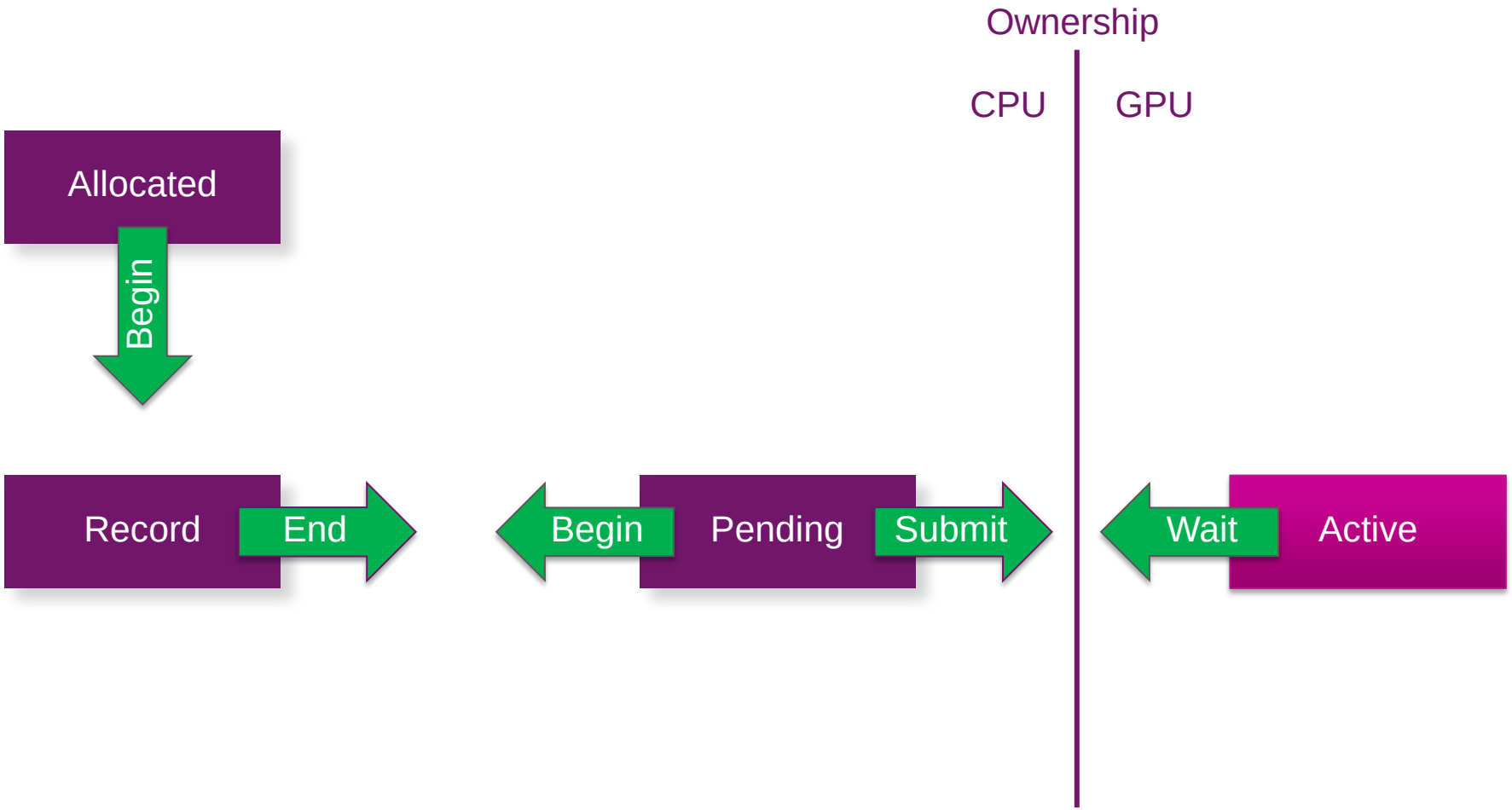
Command Buffers – Reuse



Command Buffers – Reuse



Command Buffers – Lifetime



Pipelines - An anatomy

- Fixed Function States
- Programmable Shaders
- Descriptor Layout
- Renderpass (more later)
- Dynamic State

Pipelines – Fixed Function States

- Everything that isn't a shader
- Buffer formats/layouts

- VertexInput
- InputAssembly
- Tessellation
- Viewport
- Raster
- Multisample
- DepthStencil
- ColorBlend

Pipelines – Shader Stages

- **Currently same as OpenGL**
  - Vertex
  - Control
  - Evaluation
  - Geometry
  - Fragment
- **Note: Tessellation and Geometry are optional features**

Pipelines – Descriptor Layout

Describes the set of resources that a shader can access

- Uniforms
- Storage Buffers
- Images
- Samplers
- Push Constants

Pipelines – Dynamic State

- **Per-draw state**
- **Tedious to compile each one**
  - Combinatorial explosion
- **Dynamic state!**
  - Opt-in
  - Only use when required

- **Viewport**
- **Scissor**
- **Line Width**
- **Depth Bias**
- **Blend Constant Colour**
- **Depth Bounds**
- **Stencil**
  - **Compare**
  - **Write**
  - **Reference**

Pipelines – The Cache

- Share common state
- Load/Store

The Real World, Today

- **When?**
  - As soon as you’re ready
  - Getting easier all the time
- **Why?**
  - Because Vulkan is **still** pretty dope