The future of high-performance graphics

KHRONOS GROUP

Welcome

Tom Olson

ARM

The story so far…

- **Prehistory**
  - Growing recognition that OpenGL needs a reboot
- **June to August 2014**
  - Next Generation OpenGL project launch
  - Unprecedented commitment from all sectors of the industry
  - Project disclosure and call for participation at SIGGRAPH
- **Since then…**
  - Intense focus and a lot of hard work
  - Vulkan unveil at GDC 2015

Outline

- Welcome - Tom Olson, ARM
- Introducing Vulkan - Johan Andersson, EA
- High Level Concepts in Vulkan - Pierre-Loup Griffais, Valve
- Going Wide: Vulkan and Many Threads - Dan Baker, Oxide Games
- Vulkan Binding Model - Niklas Smedberg, EPIC Games
- Shaders and Programs and Binary *Oh My!* - Aras Pranckevičius, Unity
- It's Dangerous to Vulkan Alone—Take These - John McDonald, Valve

But first…

- **Huge thanks to the whole Vulkan team!**
  - New members are always welcome

AMD

THANKS AMD!


INTRODUCING
VULKAN
JOHAN ANDERSSON
ELECTRONIC ARTS
FROSTBITE

STATE OF THE INDUSTRY

- Consensus around explicit low-level graphics APIs!
  - Mantle pioneered & led the way
  - Proven with multiple engines & full games
  - DX12 & Metal further contributed

NEED A STANDARD

- Work on all (modern) GPUs
- Supported on all platforms – desktop & mobile
- Developed together – not owned by a single platform / vendor
- Extensible - avoid explosion of platform- or vendor-specific APIs
- Binary intermediate representation for shaders

VULKAN

- Based on Mantle – standardizes & replaces it
- Explicit control & extremely low overhead
- Designed to be efficient on a wide set of GPUs
- Advanced concepts:
  - Multi queue & multi device
  - Tile-based passes & load/store operations
  - + Much more!
- Frostbite will transition our Mantle renderer to Vulkan

VULKAN FOUNDATION

- Unprecedented collaborative rapid development in Khronos
  - Game engine developers
  - Software developers
  - GPU vendors
  - Platform vendors
- Building a powerful foundation for the industry going forward!

High Level Concepts in Vulkan

Pierre-Loup Griffais

The soul of Vulkan

- Brand new API
- Built with multi-threading in mind
- Works everywhere
- Greater control over memory management
- Less hidden work and overhead in the driver
- → **Smaller drivers, better driver quality**

With great power…

- More developer responsibility
- No runtime error validation
- CPU synchronization around common objects
- CPU/GPU synchronization
- GPU memory hazards

With great power…

- More developer responsibility
- No runtime error validation
- CPU synchronization around common objects
- CPU/GPU synchronization
- GPU memory hazards

- Tools and validation suites to help developers
- John will talk more about this later

Scheduling work in Vulkan

- Queues and command buffers
- Command buffers are built on many threads
- Every command buffer is self-sufficient
- Scheduled into GPU queues

Shaders in Vulkan

- SPIR-V, brand new binary shading language
- Easy and fast to consume by the driver
- Makes offline shader compilation possible
- Spec and GLSL compiler available TODAY

Vulkan is here

- Valve driver for Intel GPUs developed along the spec to help ISVs bootstrap their code
- Source 2 supports Vulkan alpha today
- Spec and drivers coming later this year
- Intel/Linux driver will be open-sourced
- Vulkan supported across the board on Steam Machines

Demo

OXIDE GAMES

Going Wide: Vulkan and Many Threads

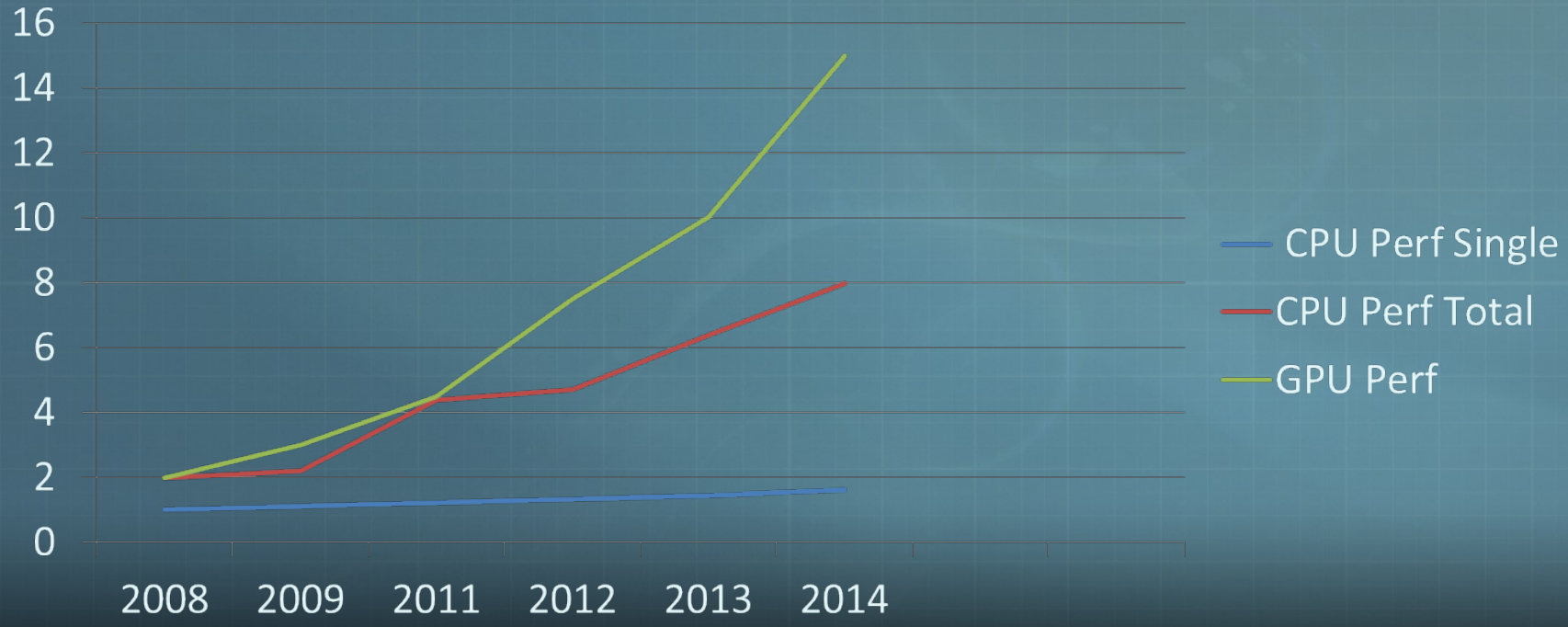
Dan Baker

Graphics Architect, Oxide Games

Inspirational Video

- Certain types of games, peak loads can get pretty big

Threading


16
14
12
10
8
6
4
2
0
2008
2009
2011
2012
2013
2014
CPU Perf Single
CPU Perf Total
GPU Perf

Observations

- GPUs are getting faster more quickly than CPUs
- Most CPU performance now comes from higher core count
- Software takes years to develop, even if your app is not CPU/API bound yet, will be soon
- Conclusion: Software must scale

CSS Threading

- Caller Side Synchronization Threading
- Previous generation APIS built on either single threaded or client/server thread designs
- Advantageous if application does not make good use of multiple cores
- Hugely problematic if application is truly threaded

CSS: Functions are "Silo"ed

```
vkTypicalCall([output],[input],[input1])
```

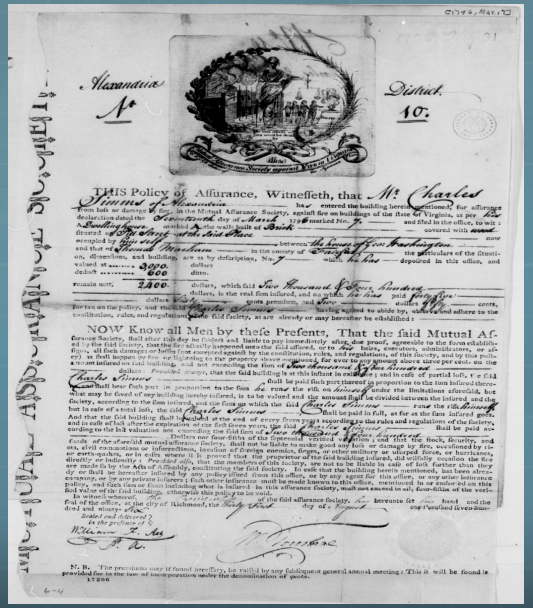
Only modifies output*

Only reads input

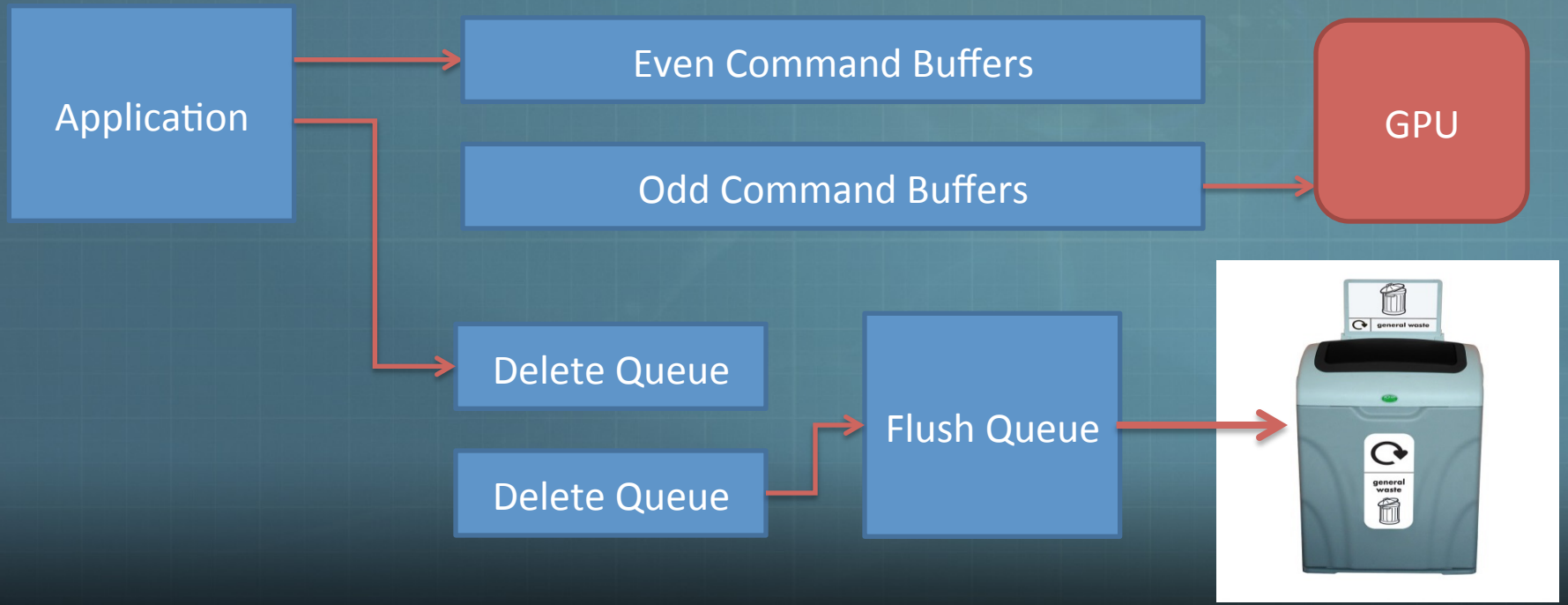
No global state side-effects!

A contract between App and API

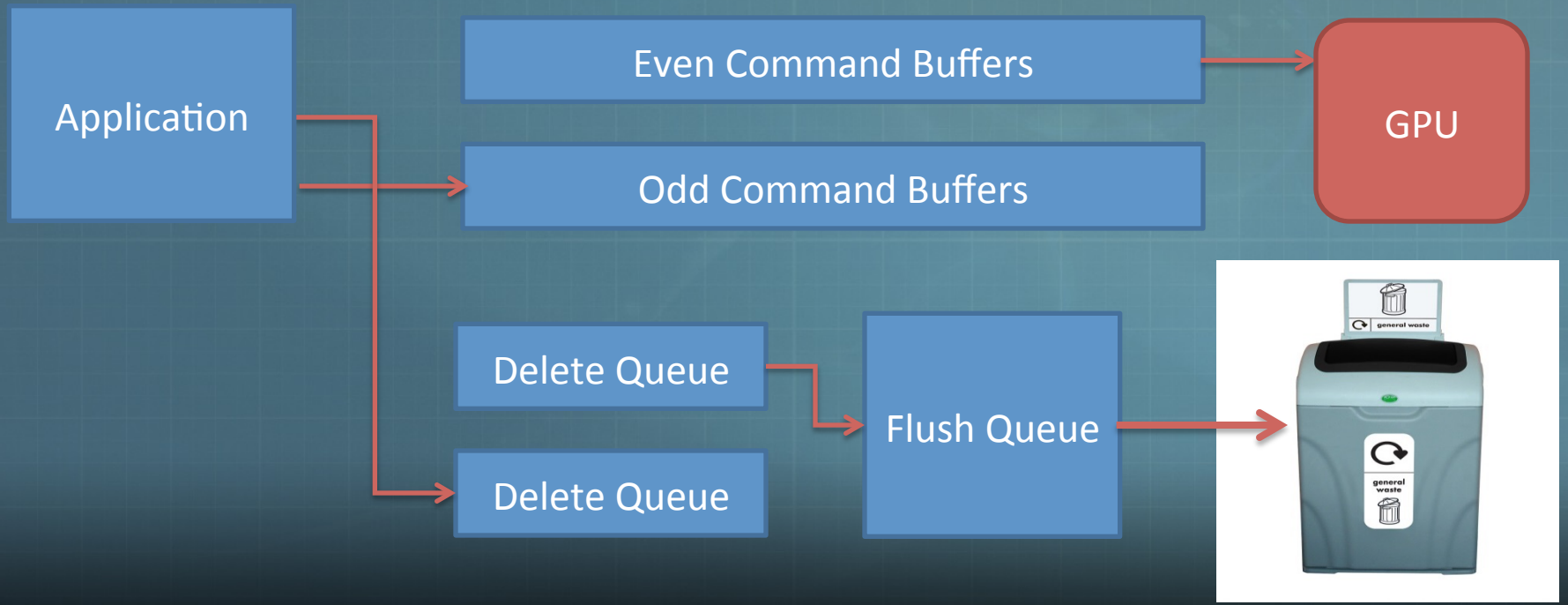
- Application will not make conflicting calls on the same objects (e.g. writing one object while another is reading it)
- Driver will generally not lock or serialize any API call
  - Context information is embedded on the object being operated on
  - With exception to occasional CPU side memory allocation (but should be rare occurrence on create calls)



Application runs parallel to GPU



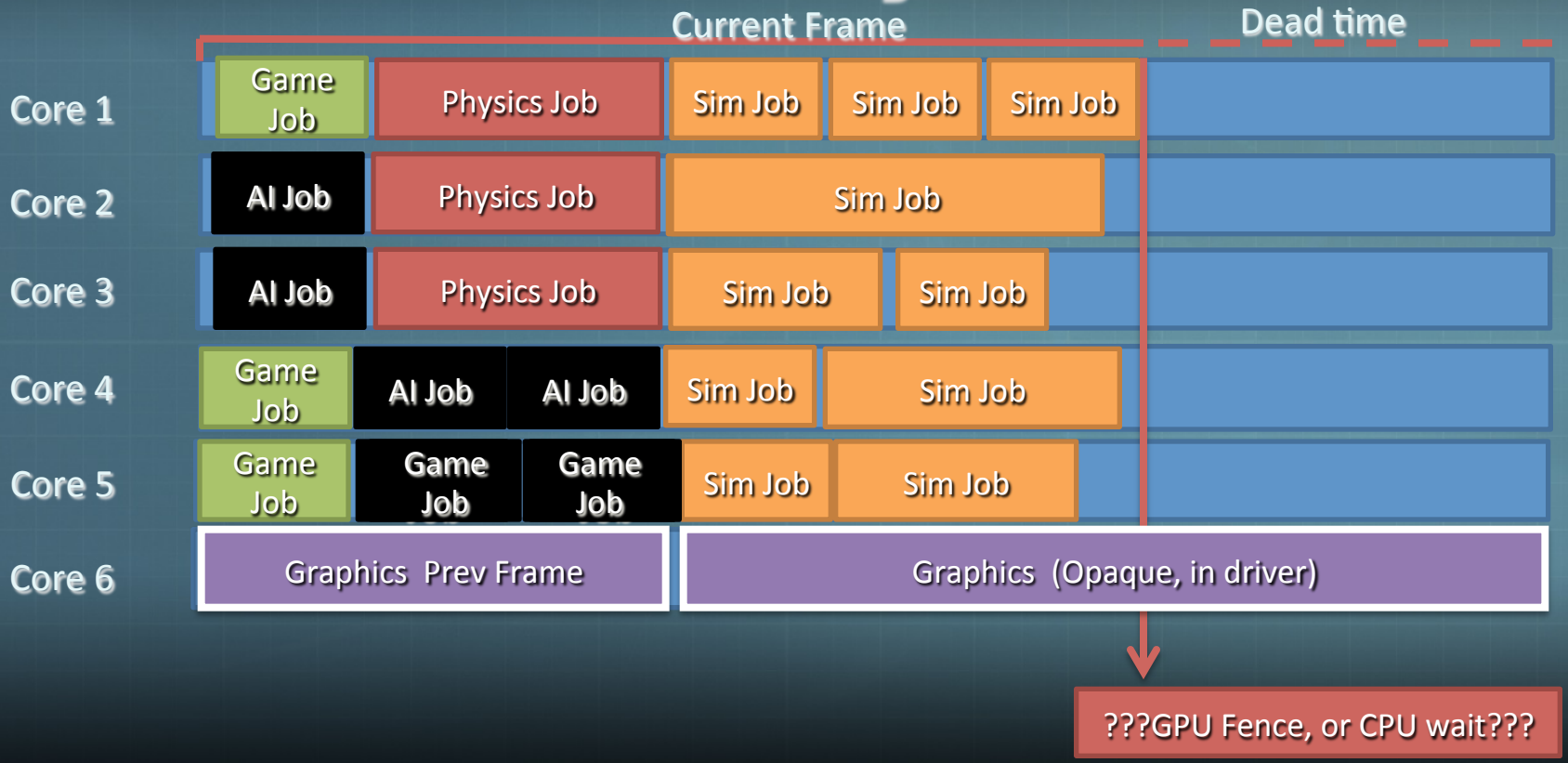
Application runs parallel to GPU



Review

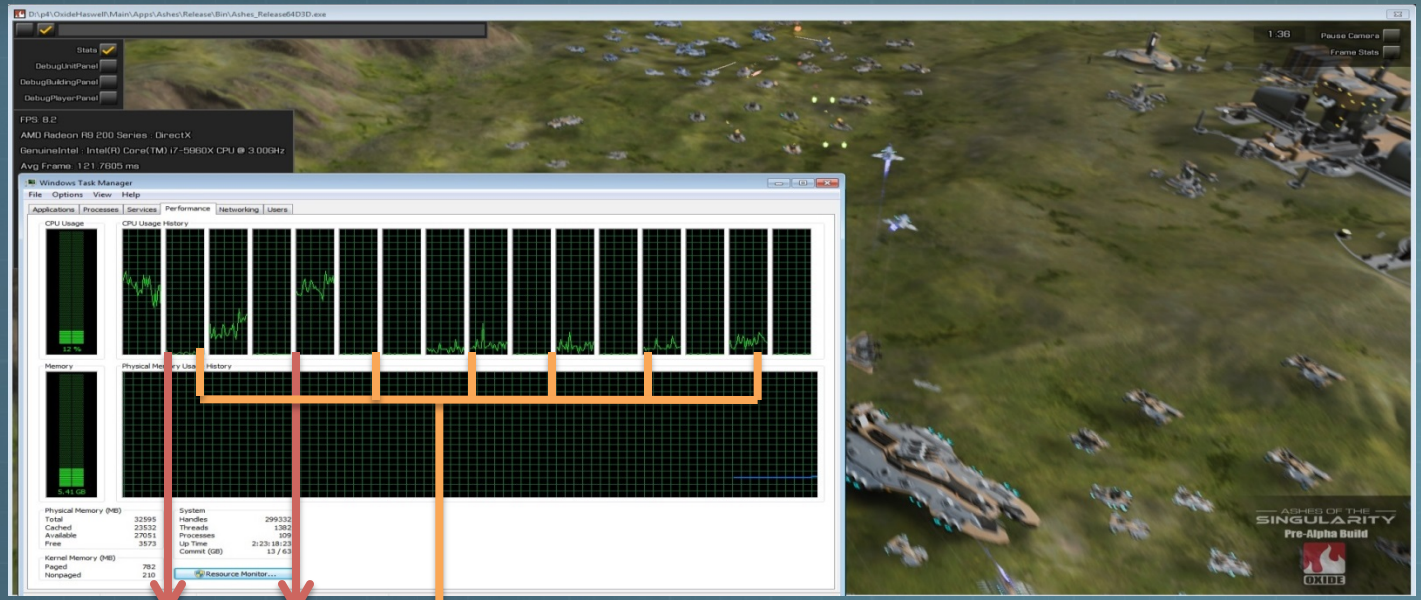
- When we say Vulkan is free threaded, we mean
  - most API function calls are *operators*. They operate only on data which is passed into them as output, and read-only the data passed on that as input
  - API function calls are transparent for thread safety: valid to call so long as the there is no read/write or write/write hazards
  - No hidden driver work!
  - GPU/CPU hazard is explicitly exposed. GPUs are read operators on data, therefore read/write hazards between CPU/GPU must also be managed by application
  - In General, API function calls will not have locks in them
    - With exception to calls which must allocate some types of memory

Old way


Current Frame
Dead time
Core 1
Game Job
Physics Job
Sim Job
Sim Job
Sim Job
Core 2
AI Job
Physics Job
Sim Job
Core 3
AI Job
Physics Job
Sim Job
Sim Job
Core 4
Game Job
AI Job
AI Job
Sim Job
Sim Job
Core 5
Game Job
Game Job
Game Job
Sim Job
Sim Job
Core 6
Graphics Prev Frame
Graphics (Opaque, in driver)
???GPU Fence, or CPU wait???

OXIDE GAMES

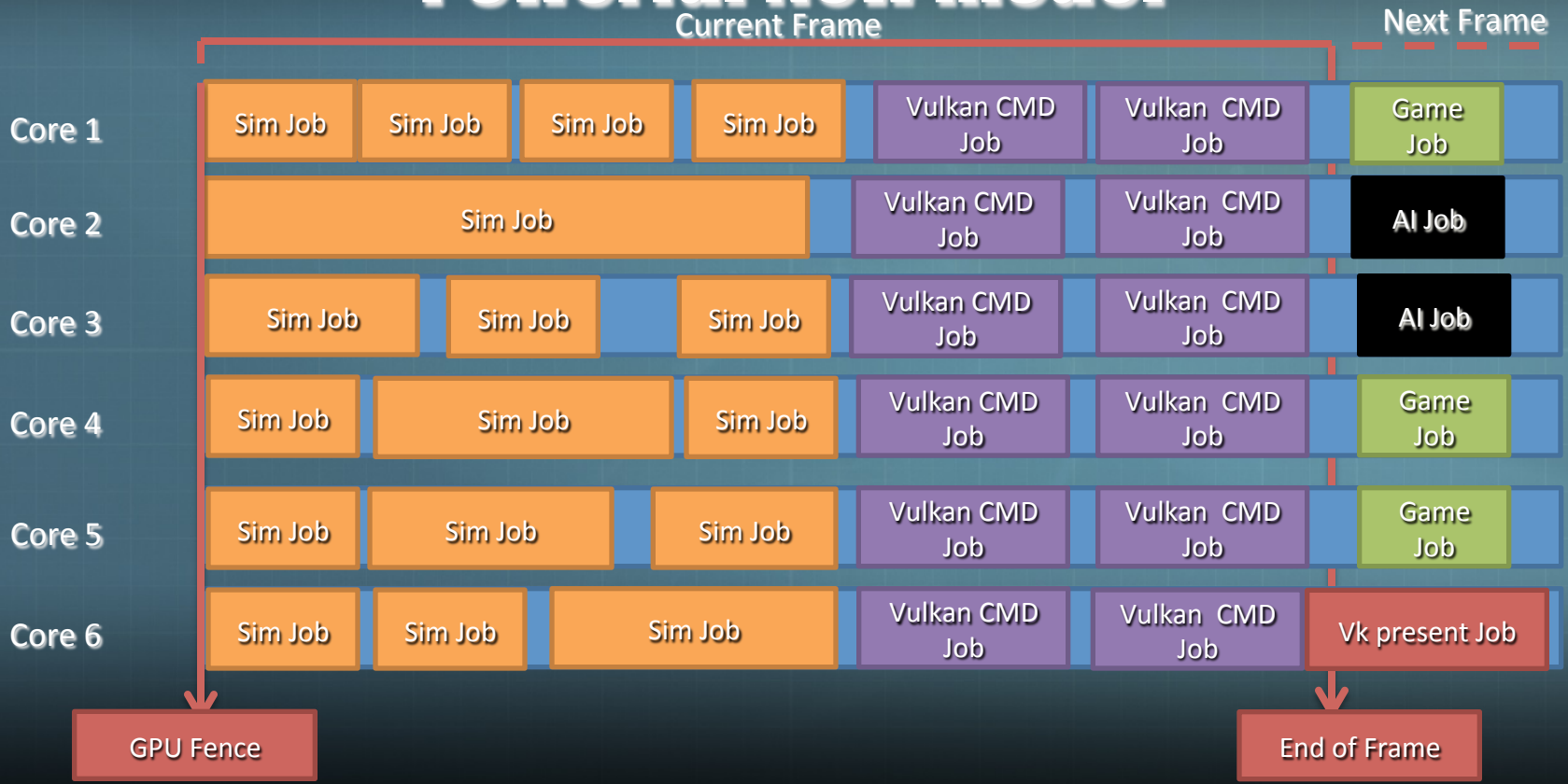
Old Way



Driver related cores. Missing time due to thread accounting and system level synchronization primitives

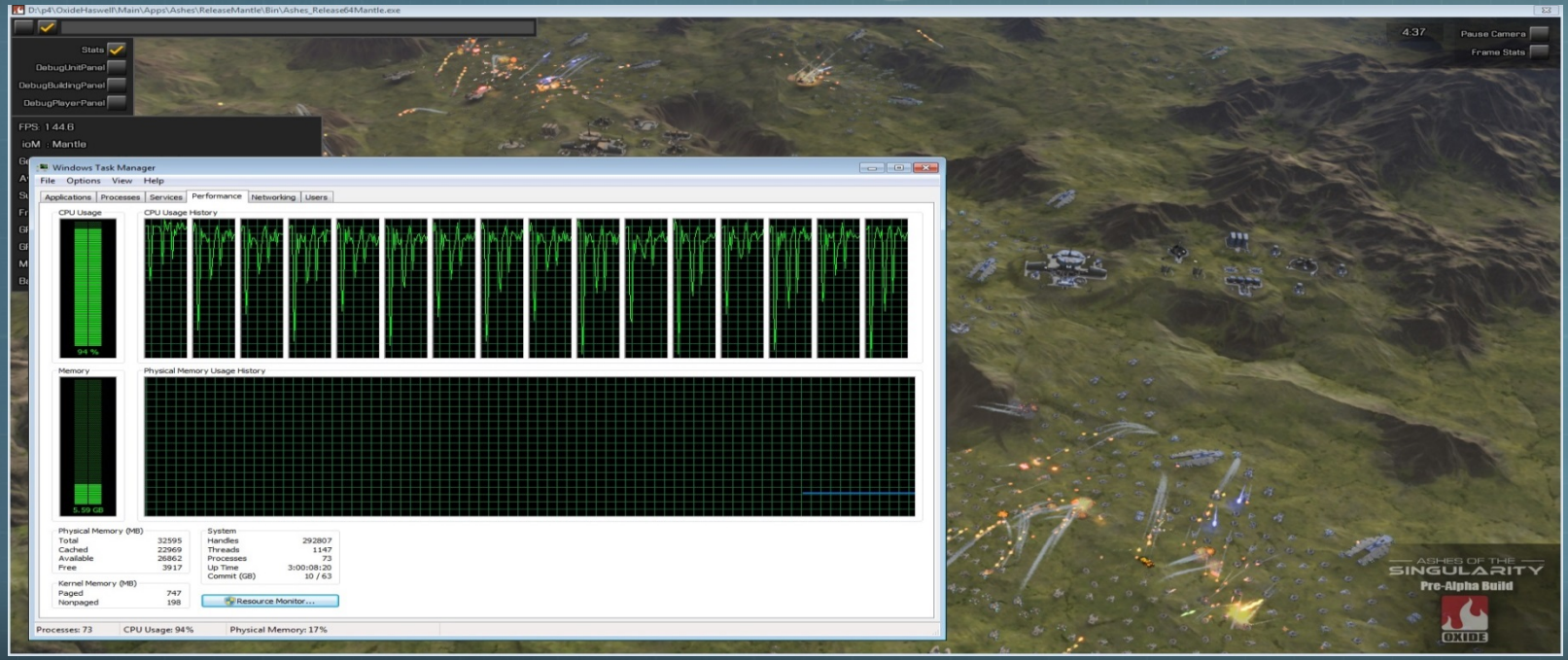
Lots of unused CPU space! Engine is just waiting for driver to be done

Powerful New model



OXIDE GAMES

New way



Vulkan simulation using a modified Mantle build to simulate infinitely fast GPU

Big performance Gains

- 8-Core/16 thread CPUs exist today for consumers – even in mobile
- Where will be in 2 years? 5? 10?
- On the high end, game genres are limited by API threading model. What types of games will Vulkan bring to us?

Vulkan Binding Model

Niklas “Smedis” Smedberg

Senior Engine Programmer, Epic Games

Goal

- Efficiently bridge the gap
  - Traditional fixed hardware bindings
  - Bindless-like operation

Overview

- Shader uses a Descriptor to access a resource
  - Resources: Sampler, Image, Buffer
- Descriptors are grouped into DescriptorSets
  - DescriptorSet is bound as a single unit
  - Shader bindings described by DescriptorSetLayout
- Shader has multiple DescriptorSet binding slots
  - Described by DescriptorSetLayoutChain in PSO

DescriptorPool

- DescriptorSets are allocated from a DescriptorPool
- One-shot:
  - Allocate until DescriptorPool is full, then clear entire pool
  - Ultra-fast allocator: great for ring buffering DescriptorSets
- Dynamic:
  - Out-of-order allocation/delete of DescriptorSets

Multithreading

- Update Descriptors on any thread
- Pre-alloc DescriptorSets of a common DescriptorSetLayout
  - Update existing DescriptorSets, avoiding alloc/free
- Update descriptors:
  - By copy (driver schedules the update)
  - Immediately (promise no synchronization is needed)

DescriptorSetLayoutChain

- Ordered by binding frequency (lowest first)
  - Fast switching between similar DescriptorSetLayoutChains
  - Low-frequency bindings are persistent
- For example, two PipelineStateObjects:
  - PSO A using Chain S: {LayoutX, LayoutY, LayoutZ}
  - PSO B using Chain T: {LayoutX, LayoutY, LayoutW}
- When binding PSO B after using PSO A:
  - Only bind Set for LayoutW
  - Sets in {LayoutX, LayoutY} persists

Extra Features

- Bind DescriptorSets to all or any subset of pipeline stage
- High-frequency buffer offsets for UBOs and SSBOs
  - Can be provided at DescriptorSet binding
- Flexibility with samplers. API supports:
  - Separate {sampler},{texture} descriptors
  - Combined {sampler,texture} descriptors
  - Immutable samplers specified in DescriptorSetLayout

Shaders and Programs and Binary *Oh My!*

Aras Pranckevičius

Graphics Plumber, Unity

TL;DR

- SPIR-V
- It’s bytecode!
- Well, that’s it :)

Improvements

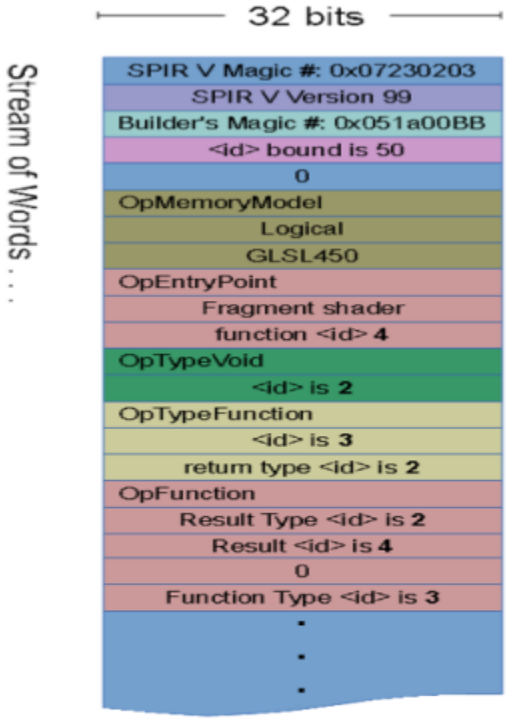
- **Half of the world is not even using GLSL!**
  - Allow more source languages
- **No need for full compiler stack in every driver**
  - Solves different frontend bugs/performance
- **Solves IP issues for some usages**

SPIR-V

- **Standard Portable Intermediate Representation**
- **Core for Vulkan**
  - Supports GLSL
- **Core for OpenCL 2.1**
  - Supports CL 1.2, 2.0, 2.1 C/C++
- **Allows conversion from/to LLVM IR**

SPIR-V

- **Highly regular binary representation**
- **Higher level than other shader IRs (D3D)**
  - No register alloc
  - No packing into float4
  - Hierarchical type info preserved
  - Structured flow control preserved
- **SSA for all intermediate results**
  - Load/Store for IO



Splitting the work

- **Offline**
  - All frontend work
  - Some optimizations
- **Load Time**
  - Conversion to GPU code, regalloc, sched
- **Vulkan**
  - No hidden recompiles (full state specified)
  - Can save/load final pipeline objects

Source Languages & IRs

- Multiple possible (GLSL & CL out of the box)
- Built-in function sets separate from core spec
- Convertible to other ILs without data loss
  - SPIR-V -> LLVM -> Optimize -> SPIR-V

Misc

- **Extensible: can import new instruction sets & semantics**
- **Debugging: can annotate anything with text/file/line**
- **All instructions encode their size**
  - Tools can skip over unknown extensions easily

Call to shader action!

- **Go write your own parser/compiler**
- **Specification and reference GLSL->SPIR-V**
  - Available right this second!
- **Other shading languages?**
  - Should be possible™

It's Dangerous to Vulkan Alone—Take These

John McDonald

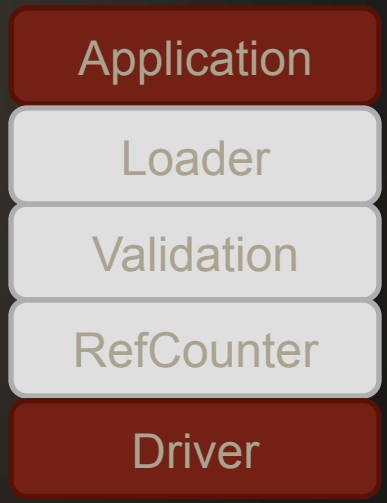
VALVE

Efficient Development

- Layered API
- Official SDK
- Other Tools

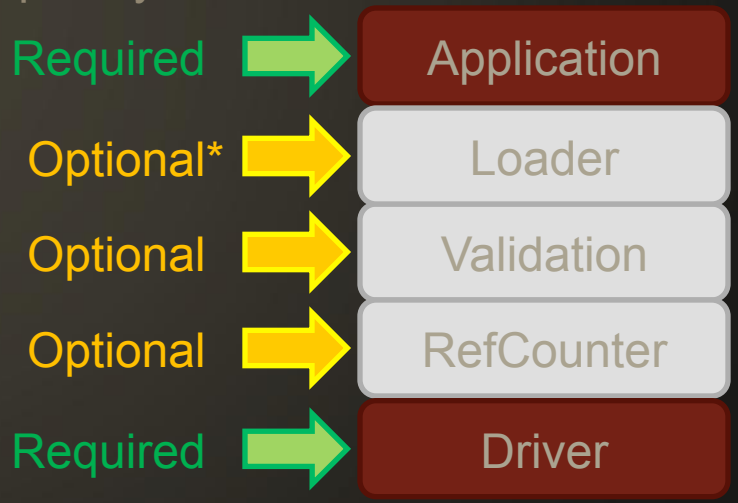
Layered API?

- Vulkan consists of multiple layers
- The bottom-most layer is driver—the top-most is the application
- Application developer chooses which layers are active
  - Proportional Taxation: Inactive Layers are completely free



Layered API?

- Vulkan consists of multiple layers
- The bottom-most layer is driver—the top-most is the application
- Application developer chooses which layers are active
  - Proportional Taxation: Inactive Layers are completely free

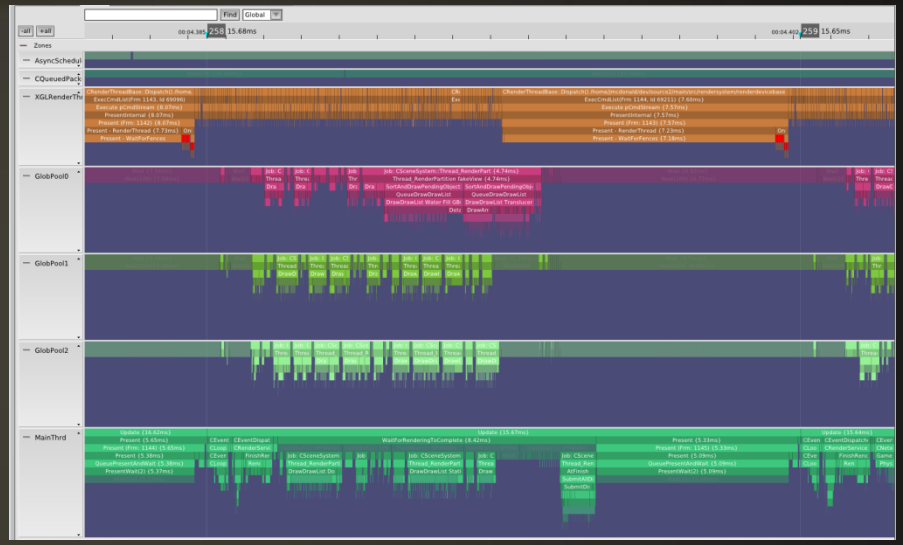


Types of Layers

- Active (application requested)
  - Debug
  - Validation
  - Performance
- Passive (injected via third party)
  - Steam Overlay
  - FRAPS

Types of Layers

- Active (application requested)
  - Debug
  - Validation
  - Performance
- Passive (injected via third party)
  - Steam Overlay
  - FRAPS
- Layers can be OSS (all Official Layers are OSS)
- Or Proprietary
  - RAD Game Tools' Telemetry Layer



Vulkan SDK

- Official SDK with BSD/MIT Open Source License directly from Khronos
- SDK Includes
  - Loader
  - Header / Build Scripts per platform
  - Official Layers

Loader?

- Expected that platforms will ship a default loader
  - You can use this if you choose
- But applications can ship a private version as well
  - Allows you to work around bugs
  - And avoids the opengl32.dll problem
- The Loader is actually just another Layer!
- You *could* opt-out of using the loader, but it's usage is highly recommended

Official Layers

- The complete list of layers is TBD, but the current expectation is roughly:
  - Loader
  - Validation
  - Reference Counting
  - Performance Linting
  - Debug
  - Thread Safety
  - Trace Capture & Replay
- Above may change, of course

Additional SDK Includes

- Scripts to trivially generate your own layers
  - Useful for tracking down app-specific bugs
  - And helping build a richer ecosystem
- Open Source Intel Driver
  - For Linux/SteamOS
  - Used in today's demos
- GLSL->BIL Compiler

Conformance Testing

- Open Source Conformance Testing!
  - Allow developers to easily verify an implementation they are running on
  - And see examples of feature usage in real code
- Ensure your application is covered by submitting usage cases back to Khronos
  - Patches Welcome

3rd Party Tools

- Introducing GLAVE, an Open Source Debugger for Vulkan
  - Effectively VOGL, but for Vulkan
- Developed in parallel with the API
- Expected to ship with the API

Performance Demo

Questions?

- Come ask questions at Khronos sessions **today**
  - SF Green Space
  - 657 Mission Street Suite 200 (5 minute walk)
    - 12-1:30 (First Session)
    - 2-3:00 (Second Session)
  - All Technical Questions are fair game!

- Tom Olson: @thekhronosgroup
- Johan Andersson: @repi
- Pierre-Loup Griffais: @plagman
- Niklas Smedberg: @EpicGames
- Dan Baker: @danbaker, @oxidegames
- Aras Pranckevicius: @aras_p
- John McDonald: @basisspace