

@AxSaucedo

# CppCon 2022

## GPU Accelerated Computing and Optimizations on Cross-Vendor Graphics Cards with Vulkan & Kompute

Alejandro Saucedo

[@AxSaucedo](#)

IT'S BREATHTAKING.



# Hello, my name is Alejandro

 @AxSaucedo



Alejandro Saucedo  
@AxSaucedo

Engineering Director  
Seldon Technologies

Chief Scientist  
The Institute for Ethical AI & ML

Governing Council Member-at-Large  
Association for Computing Machinery



# High level Objectives

Parallel Processing

GPU Computing

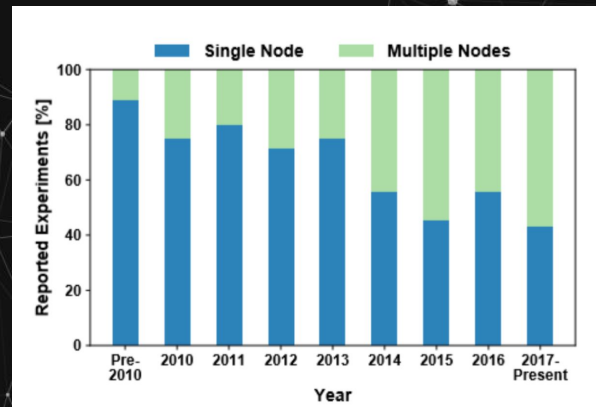
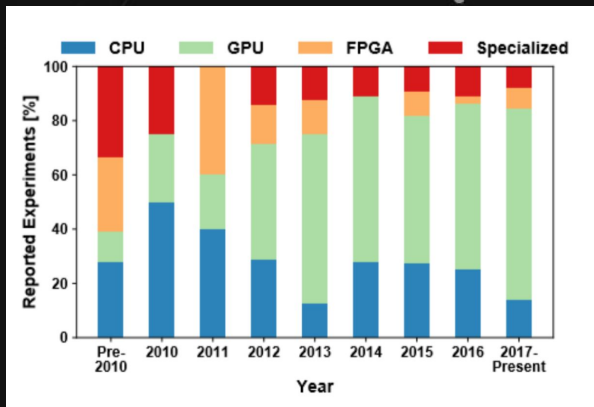
Vulkan SDK

Kompute Framework

Hands on Examples

# Why Parallel Processing?

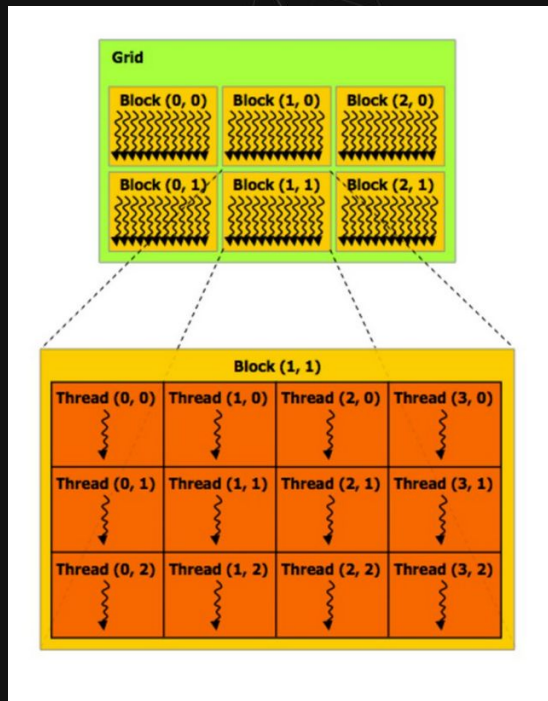
- Functions can often be reduced to highly parallelizable stages (Matrix Mult, ML Layers, etc)
- Micro-batching allows for further parallelization of multiple inputs (eg. cost instead of loss)
- Breaking up fractions of each ensemble comp. across tightly coupled hardware (eg. multi-GPU)



Ben-Nun, Tal, and Torsten Hoefler. "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis." ACM Computing Surveys (CSUR) 52.4 (2019): 1-43.

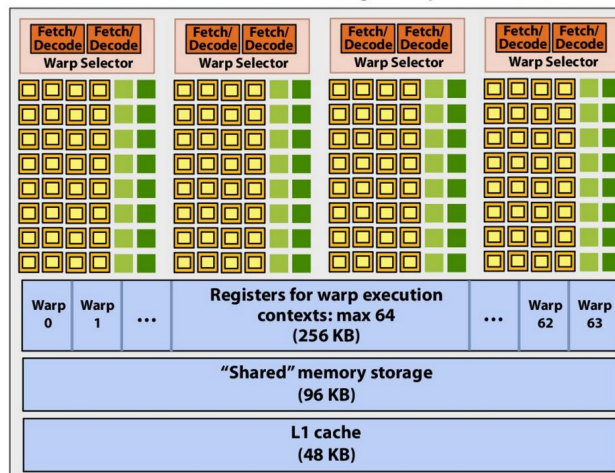





# GPU Compute: Hardware design



## NVIDIA GTX 1080 (2016)

This is one NVIDIA Pascal GP104 streaming multi-processor (SM) unit

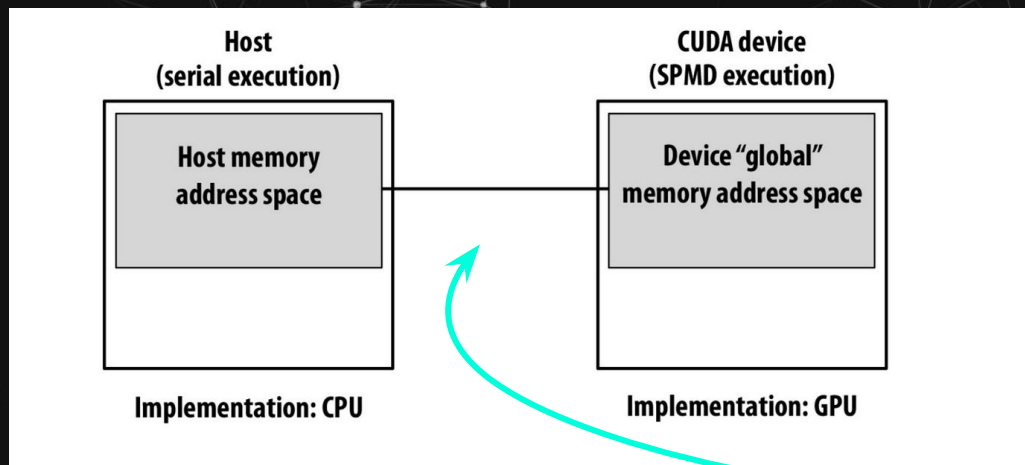


-  = SIMD functional unit, control shared across 32 units (1 MUL-ADD per clock)
-  = load/store
-  = SIMD special function unit (sin, cos, etc.)

- SM resource limits:
- Max warp execution contexts: 64 (2,048 total CUDA threads)
  - 96 KB of shared memory

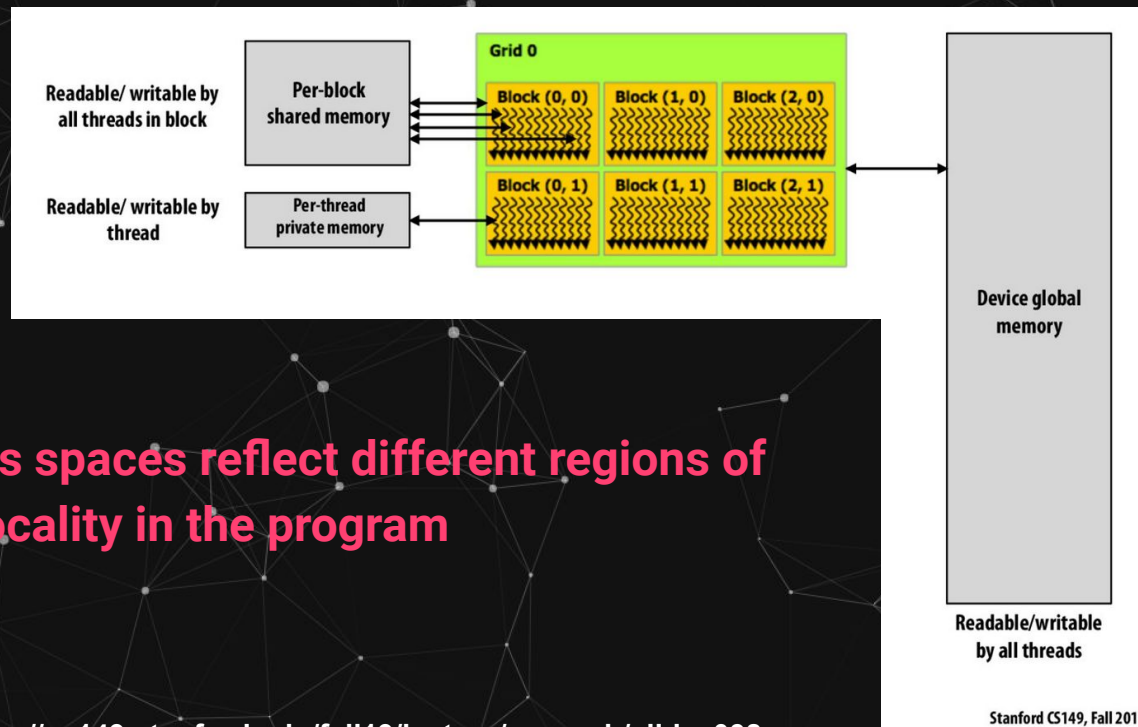
# GPU Compute: Memory Model

Distinct host and GPU device address spaces



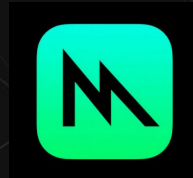
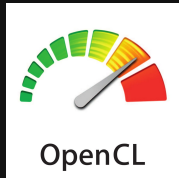
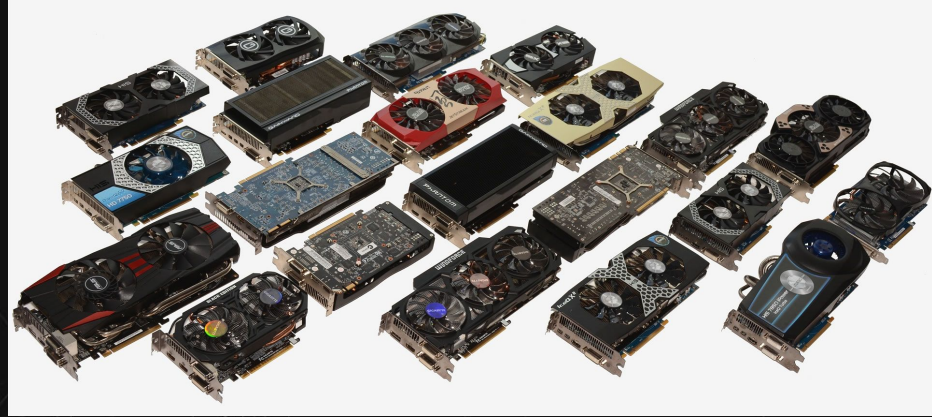
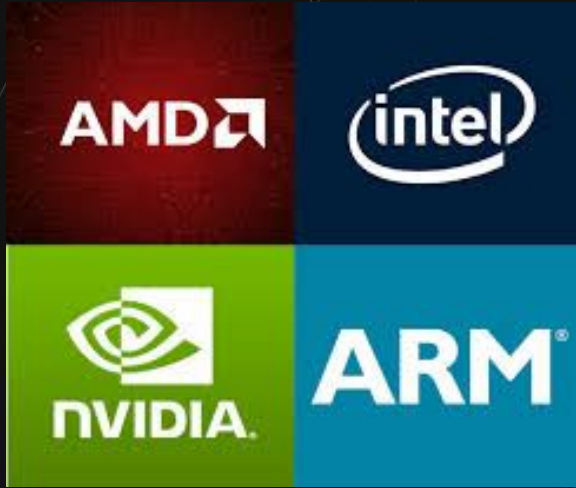
Memory is allocated  
and copied explicitly

# GPU Compute: Memory Model



Different address spaces reflect different regions of locality in the program

# Motivations: Heterogeneity



Do you  
not have  
Phones?





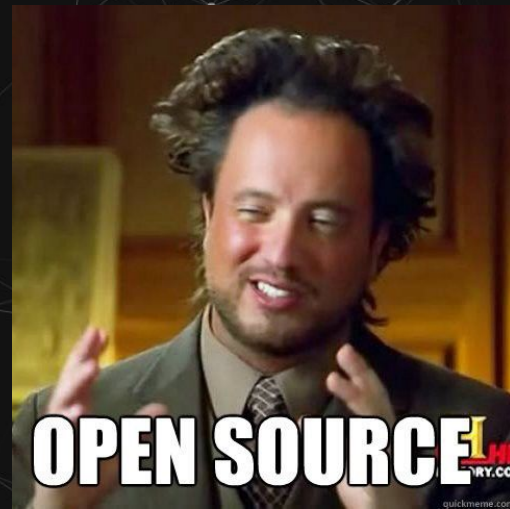
# Introducing Vulkan

## Created by the Khronos group

The Khronos Group, Inc. is a non-profit member-funded industry consortium, focused on the creation of open standard, royalty free APIs for authoring and accelerated playback of dynamic media on a wide variety of platforms and devices.

## Top Vulkan Priorities

1. Performance
2. Interoperability
3. Performance



# Khronos Members



# Vulkan SDK

## Advantages

- Low level with rich access to components
- C-style API as core interface for developing GPU applications
- A broad range of top players leading the development of the framework
- Highly compatible across different platforms, mobile, and different suppliers

## Disadvantages

- Low level with rich access to components
- C-style API as core interface for developing GPU applications
- A broad range of top players leading the development of the framework
- Highly compatible across different platforms, mobile, and different suppliers

# Architecture

Vulkan Overarching Application which owns all sub-components

Application

Instance

Physical Device

Logical Device

Queue

Command Buffer

Command Buffer

Command Buffer

Vulkan can have multiple instances of application

Physical devices specify the underlying GPU card hardware

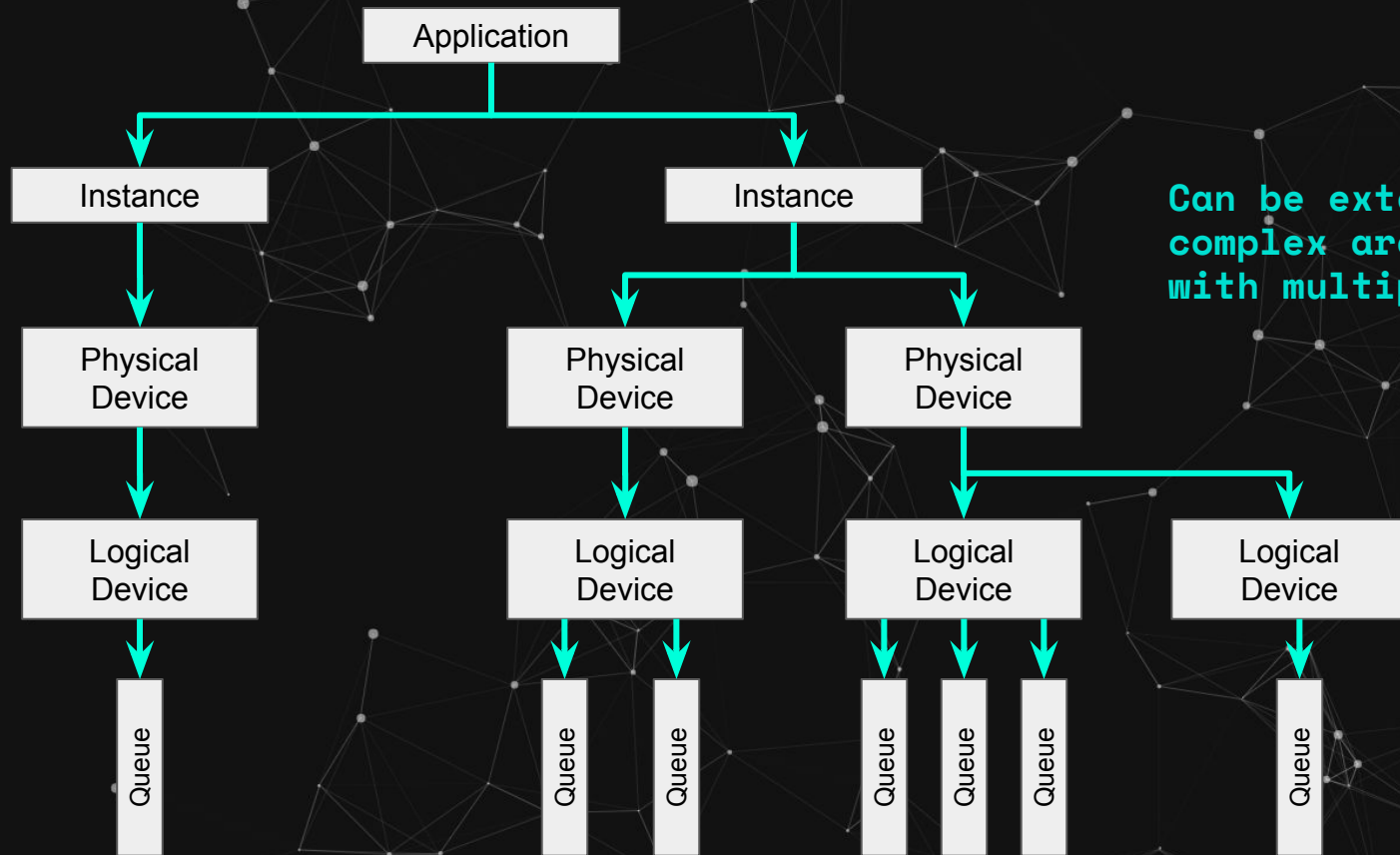
Logical devices are abstractions for physical devices with set properties

Work gets submitted via queues from CPU and gets picked up and executed from GPU

Command buffers specify work to be carried out as an individual unity (move data, run shader, copy data, etc)

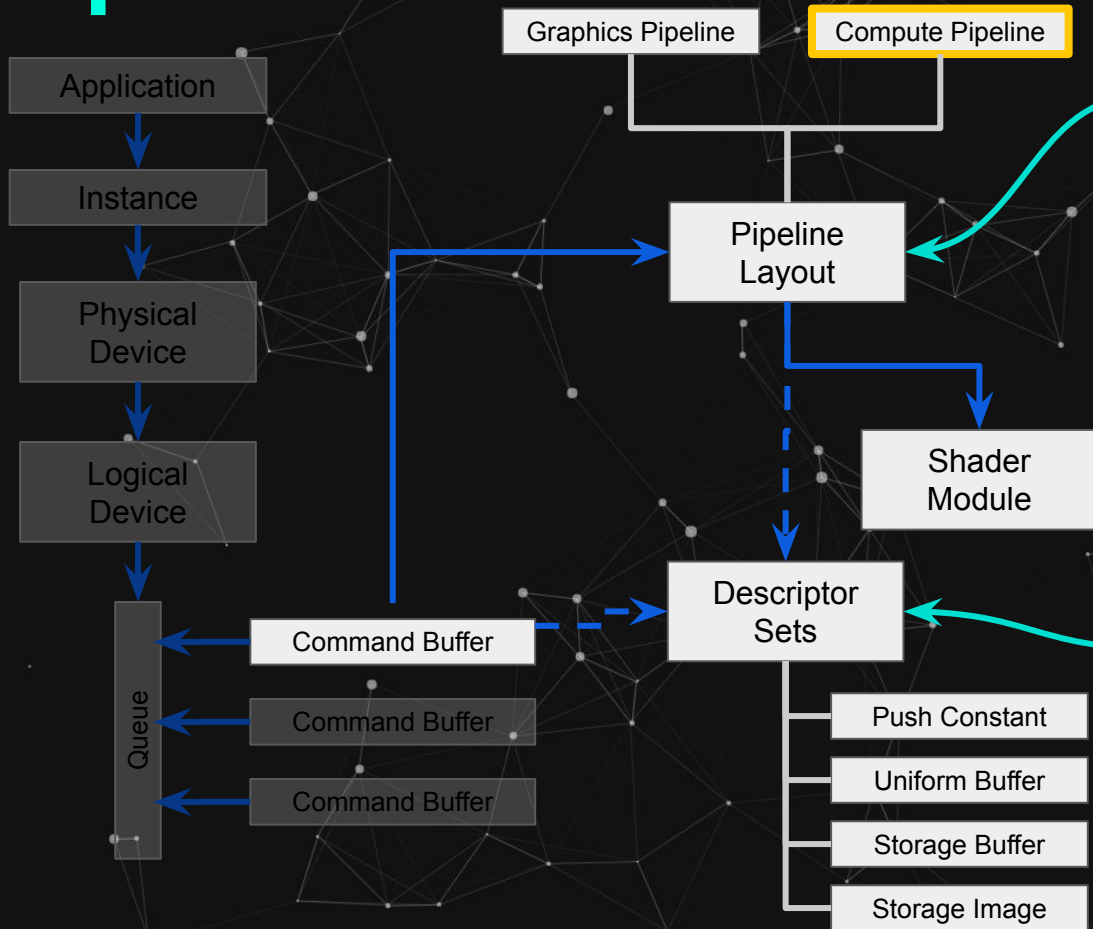


# Application



Can be extended to  
complex architectures  
with multiple devices

# Pipeline

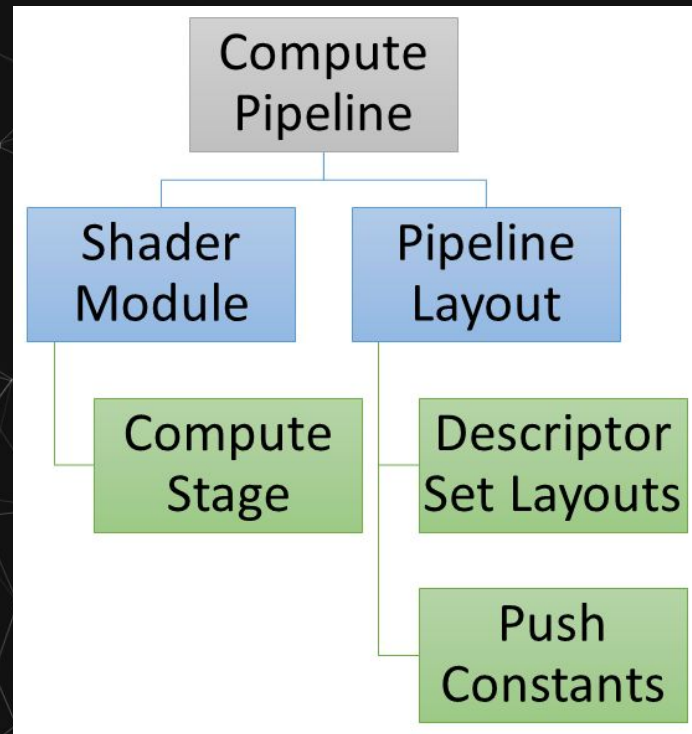
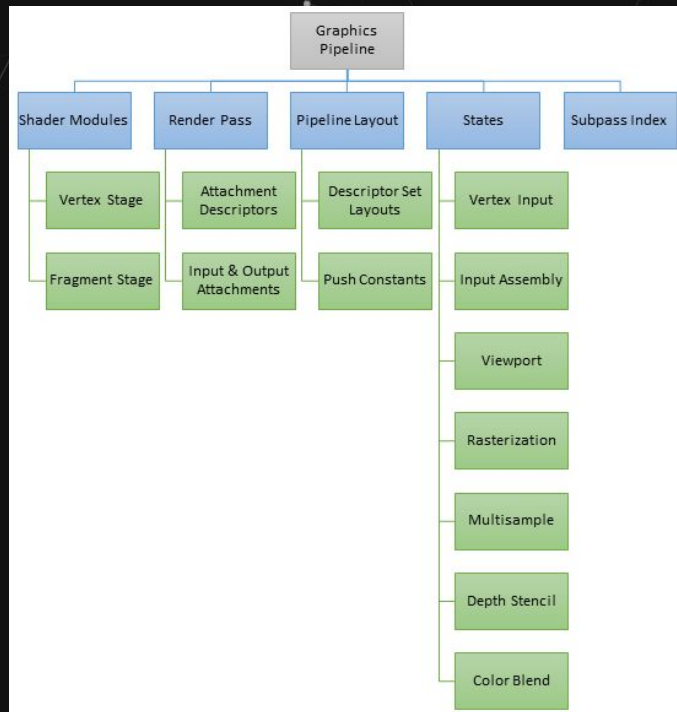


Vulkan Pipeline is core to the “processing” stages, binding the key components on each stage

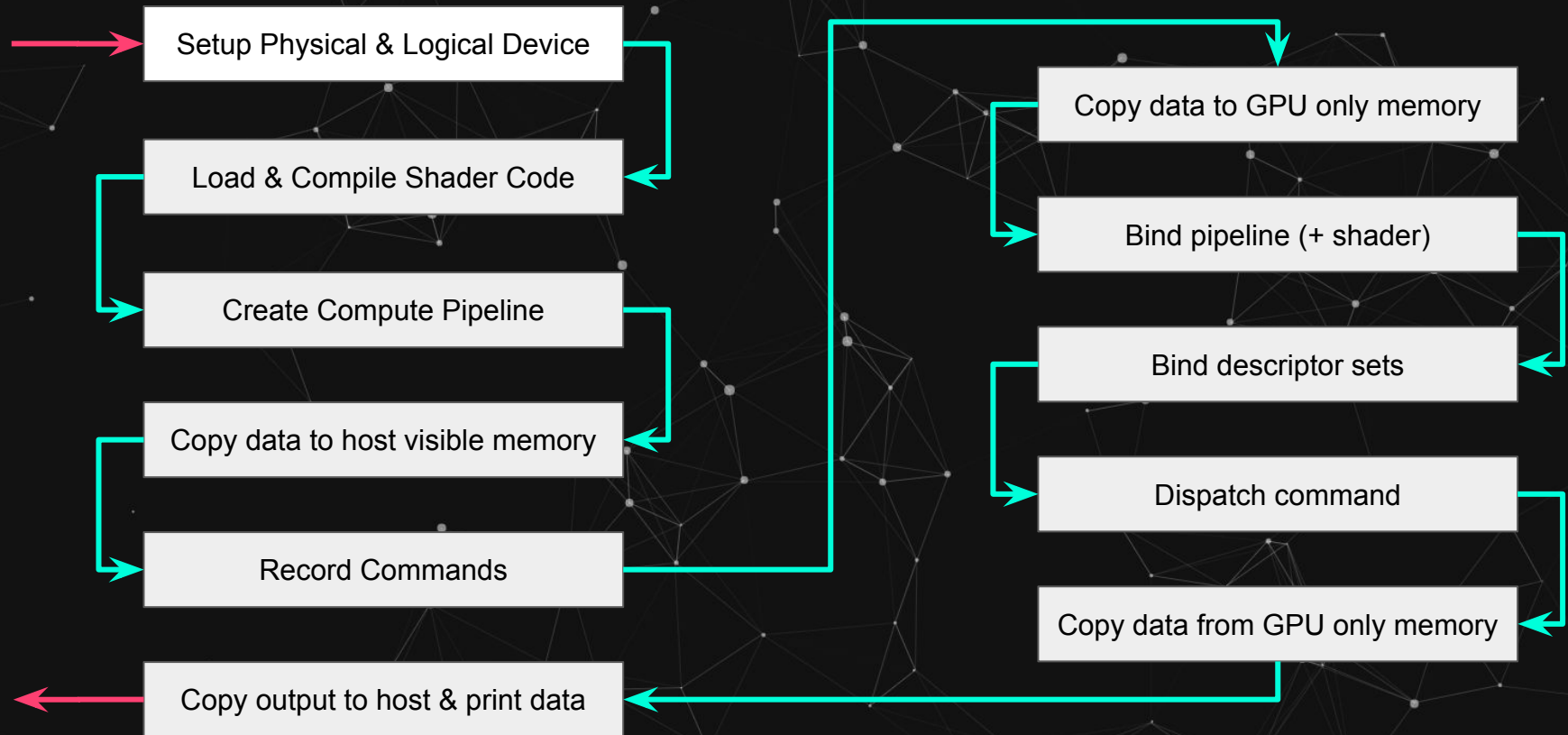
Shaders consist of the parallel code that runs in the GPU to perform any relevant processing

Descriptors are the components that can be used to bind data from GPU buffers into shaders and pipelines


# Vulkan Pipeline Types



# The Life of a Vulkan Program





The background is a dark gray to black field filled with a complex, interconnected network of thin white lines. These lines connect numerous small, light gray circular nodes, creating a web-like or molecular structure that spans the entire frame. The density of the connections varies, with some areas appearing more clustered than others.

**Only takes about  
500-2000 lines of code...**

# Enter Kompute

## The General Purpose Vulkan Computing Framework.

- **Dozens** instead of thousands of lines of code required
- **Augments** Vulkan interface instead of abstracting it
- **BYOV**: Bring-your-own-Vulkan design to play nice with existing Vulkan applications
- **Non-Vulkan name convention** to disambiguate components





Kompute is part of the Linux Foundation

[kompute.cc/overview/community.html](https://kompute.cc/overview/community.html)

## High Level Overview of Features

- C++ Interface
- Python Bindings
- Explicit (GPU/CPU) memory ownership
- Granular access to GPU queues
- Single header file available
- Integration with Mobile Apps
- Integration with Game Engine

# Vulkan Kompute: Components

Top level resource that manages Vk Device and Vk Queue

Kompute Manager

Kompute Sequence

Manages & executes operations as batch in GPU as record commands and queue submits

Core data unit component to transfer and process via GPU memory and buffers

Kompute Operation

Kompute Operation

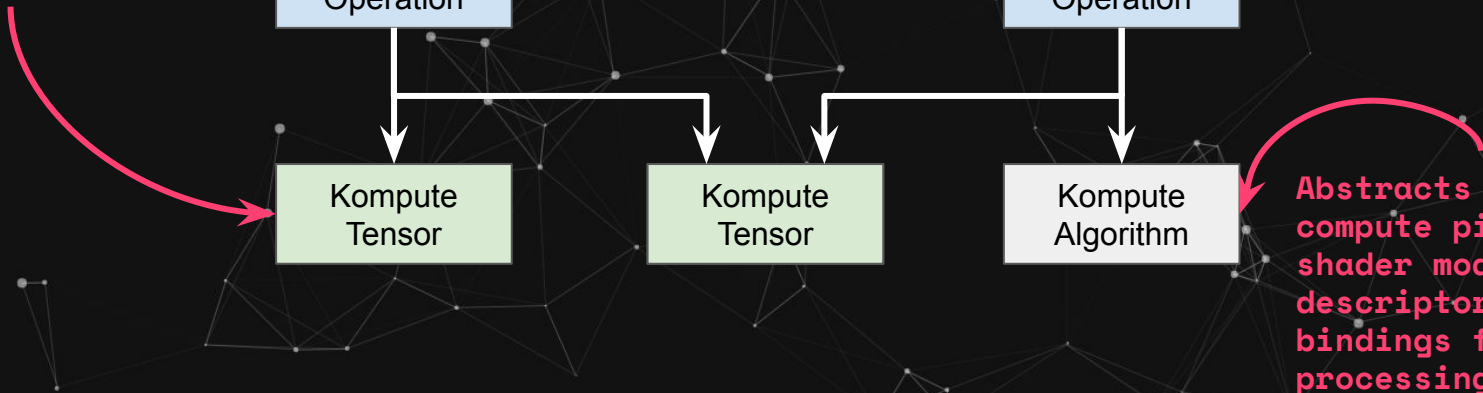
Extensible operation that performs instructions in GPU with tensor and optional shader

Kompute Tensor

Kompute Tensor

Kompute Algorithm

Abstracts the compute pipeline, shader modules and descriptor set bindings for processing





# Enter Vulkan Kompute (Simple Sum Example)

```
// 1. Create Kompute Manager  
kp::Manager mgr;
```

```
// 2. Create and initialise Kompute Tensors through manager  
auto tensorInA = mgr.tensor({ 2., 2., 2. });  
auto tensorInB = mgr.tensorT<float>({ 1., 2., 3. });  
auto tensorOut = mgr.tensorT<float>({ 0., 0., 0. });  
auto params = { tensorInA, tensorInB, tensorOut };
```

```
// 3. Run operation synchronously  
auto algo = mgr.algorithm(params, shader);
```

```
// 4. Copy Tensor and execute algorithm  
mgr.sequence()  
->record<kp::OpTensorSyncDevice>(params);  
->record<kp::OpAlgoDispatch>(algo);  
->record<kp::OpTensorSyncLocal>(params)  
->eval();
```

```
// Prints the output which is Output: { 2, 4, 6 }  
for (const float& elem : tensorOut->data())  
    std::cout << elem << " ";
```

```
static std::vector<uint32_t> shader = compileShader(R"(  
#version 450  
  
layout (local_size_x = 1) in;  
  
// The input tensors bind index is relative  
layout(binding = 0) buffer bina { float tina[]; };  
layout(binding = 1) buffer binb { float tinb[]; };  
layout(binding = 2) buffer bout { float tout[]; };  
  
void main() {  
    uint index = gl_GlobalInvocationID.x;  
    tout[index] = tina[index] * tinb[index];  
}  
)");
```

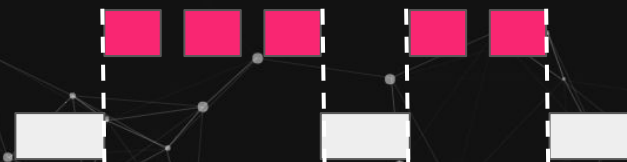
# Deeper Optimizations

CPU

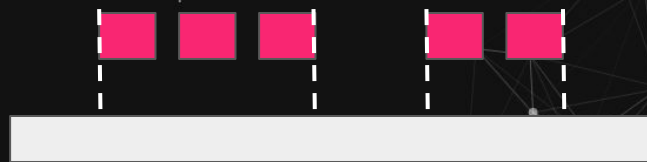
GPU



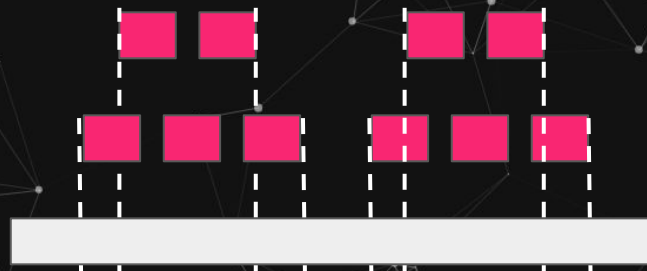
Run a single command/operation in a sequence with manager



Reuse multiple sequences in same Tensors with pre-recorded cmds

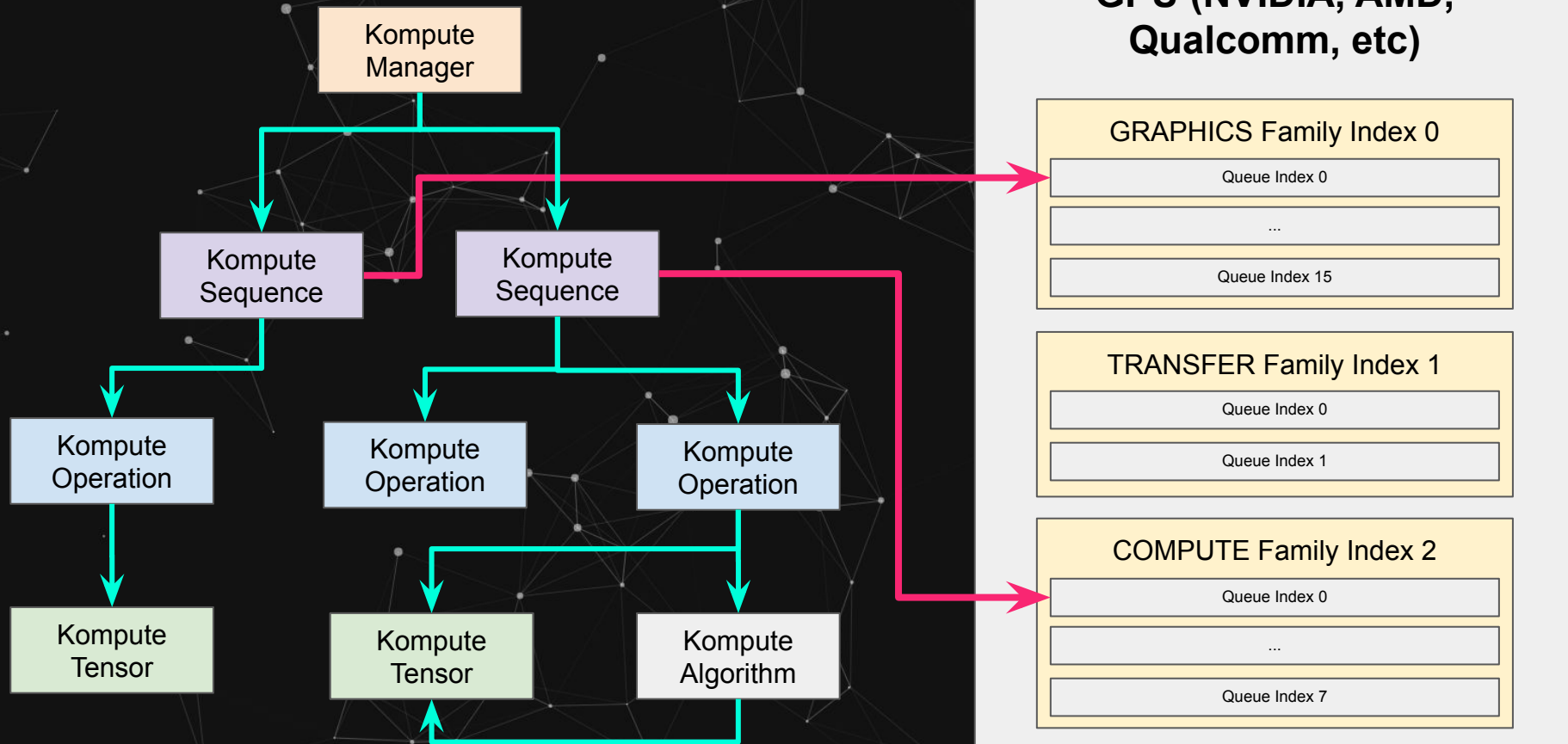


Asynchronous execution of Sequences



Concurrent execution of Sequences across GPU queues

# Kompute GPU Optimizations



# Enter Vulkan Kompute (Hardware Parallel)

```
// Kompute Manager with custom settings
uint32_t deviceId(1);
std::vector<uint32_t> queues({ 0, 2 });

kp::Manager mgr(deviceId, queues);
```

```
// Create parameters to use for each computation
std::vector<std::shared_ptr<kp::Tensor>> paramsA = { ... };
std::vector<std::shared_ptr<kp::Tensor>> paramsB = { ... };
```

```
// Create seq on relative index
auto algo1 = mgr.algorithm(paramsA, shader);
auto algo2 = mgr.algorithm(paramsB, shader);
```

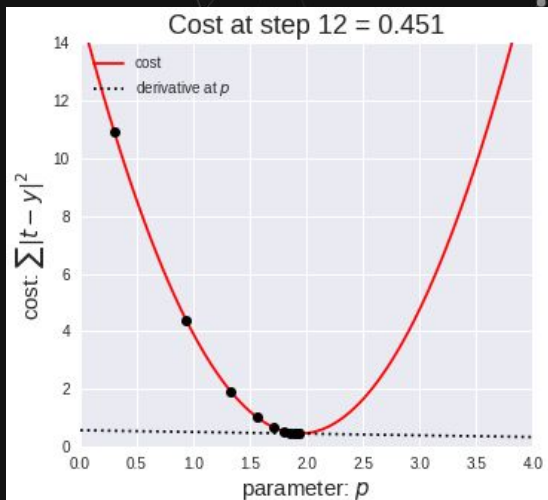
```
sq1->evalAsync<kp::OpaAlgoDispatch>(algo1);
sq2->evalAsync<kp::OpaAlgoDispatch>(algo2);
```

```
// Create seq on relative index
auto sq1 = mgr.sequence(0);
auto sq2 = mgr.sequence(1);
```

```
sq1->evalAwait();
sq2->evalAwait();
```

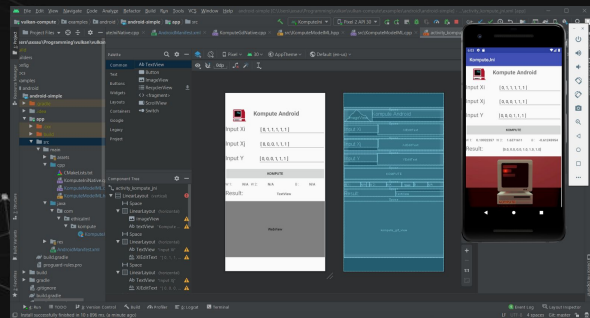


# Check out other tutorials

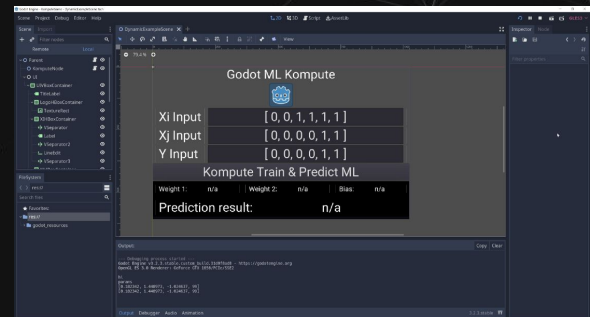


## GPU Accelerated Machine Learning

[\[Blog Post\]](#)

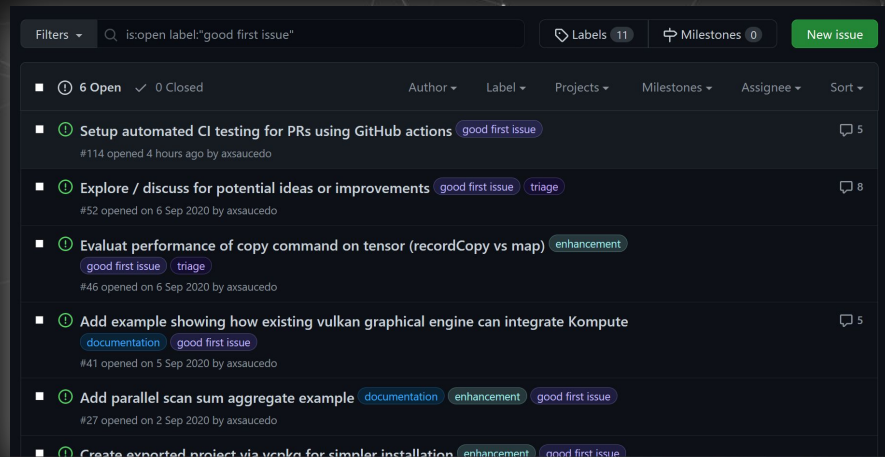


## Android App Integration [\[Blog Post\]](#)



## Godot Game Engine [\[Blog Post\]](#)

# Get Involved!



A screenshot of a GitHub repository's issue list. The search bar at the top shows the filter "is:open label:'good first issue'". There are 11 labels and 0 milestones. The list of issues includes:

- 6 Open, 0 Closed. Author, Label, Projects, Milestones, Assignee, Sort.
- Setup automated CI testing for PRs using GitHub actions (good first issue) #114 opened 4 hours ago by axsaucedo (5 comments).
- Explore / discuss for potential ideas or improvements (good first issue, triage) #52 opened on 6 Sep 2020 by axsaucedo (8 comments).
- Evaluat performance of copy command on tensor (recordCopy vs map) (enhancement, good first issue, triage) #46 opened on 6 Sep 2020 by axsaucedo (5 comments).
- Add example showing how existing vulkan graphical engine can integrate Kompute (documentation, good first issue) #41 opened on 5 Sep 2020 by axsaucedo (5 comments).
- Add parallel scan sum aggregate example (documentation, enhancement, good first issue) #27 opened on 2 Sep 2020 by axsaucedo.
- Create exported project via vcpkg for simpler installation (enhancement, good first issue).

## Explore / discuss for potential ideas or improvements #52

Open axsaucedo opened this issue on 6 Sep 2020 · 8 comments



axsaucedo commented on 6 Sep 2020

Member

Open issue to openly discuss potential ideas or improvements, whether on documentation, interfaces, examples, bug fixes, etc.



axsaucedo added the triage label on 6 Sep 2020



DTolm commented on 7 Sep 2020

Hello,  
I am Dmitrii, the creator of VkFFT and Vulkan version of Spirit. I saw your comment and I believe your project is the way to go, if Vulkan wants to be popular in compute or, specifically, scientific field. There has to be some kind of a layer that will move them as further as possible from the way I developed Vulkan Spirit. There are some important things, that have to be clarified in the absolute beginning, which are related to the architectural problems and how this layer should be designed. These things are based on my

Pick up one of the  
good-first-issues

Share thoughts and  
suggestions via #52

# High level Roadmap

Integrate as backend of ML / scientific-computing framework(s)

---

Create more default `kp::Operations` to have out of the box commands

---

Examples running Kompute across other platforms and frameworks

@AxSaucedo

# CppCon 2022

GPU computing using Vulkan & Kompute for  
Cross-vendor Graphic Cards (AMD, Qualcomm,  
NVIDIA & friends)

Alejandro Saucedo

[@AxSaucedo](#)

THE MORE YOU KNOW 