# Advanced Metal Techniques

How to avoid common pitfalls in Metal

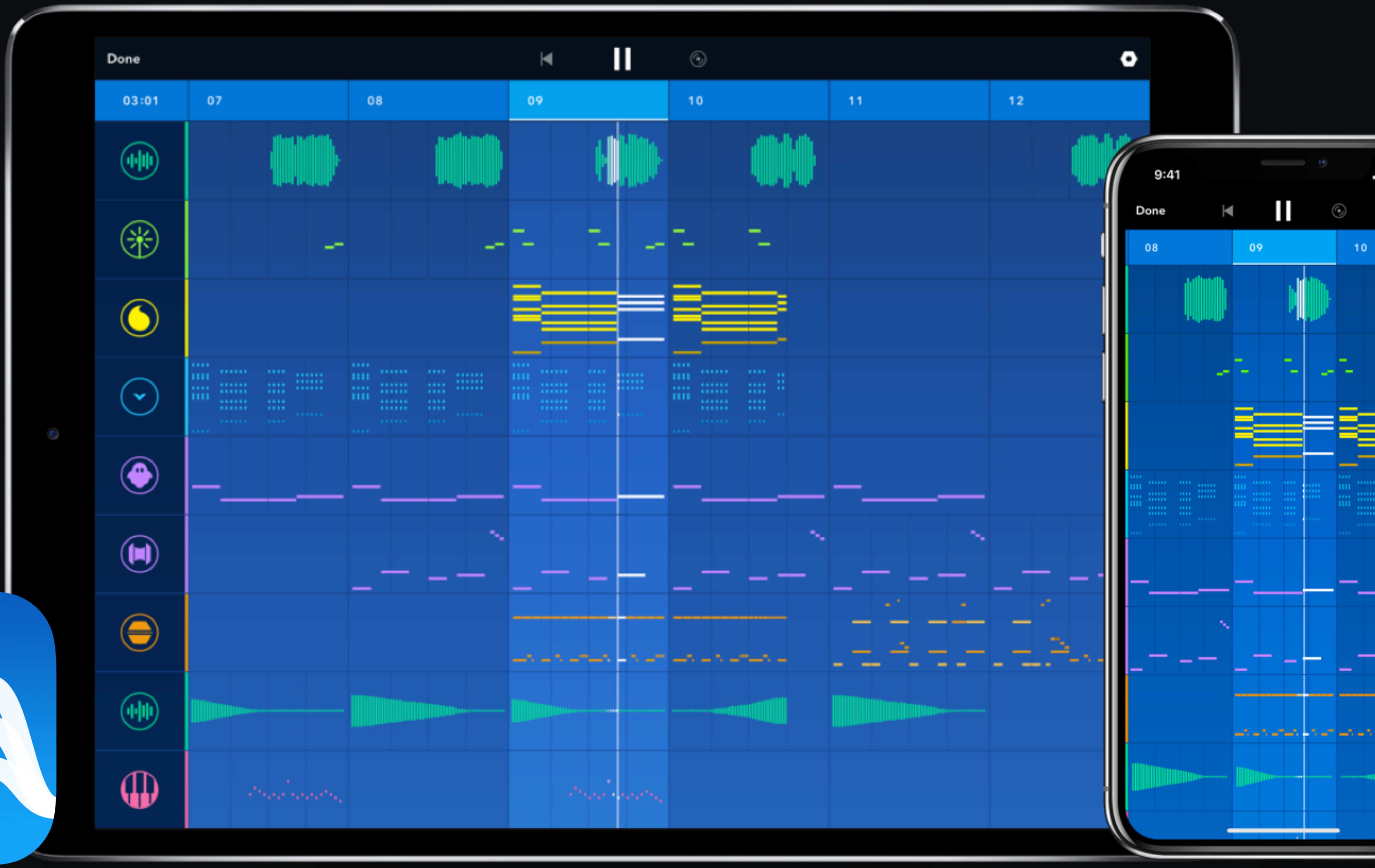
Basil Al-Dajane | Medly Labs

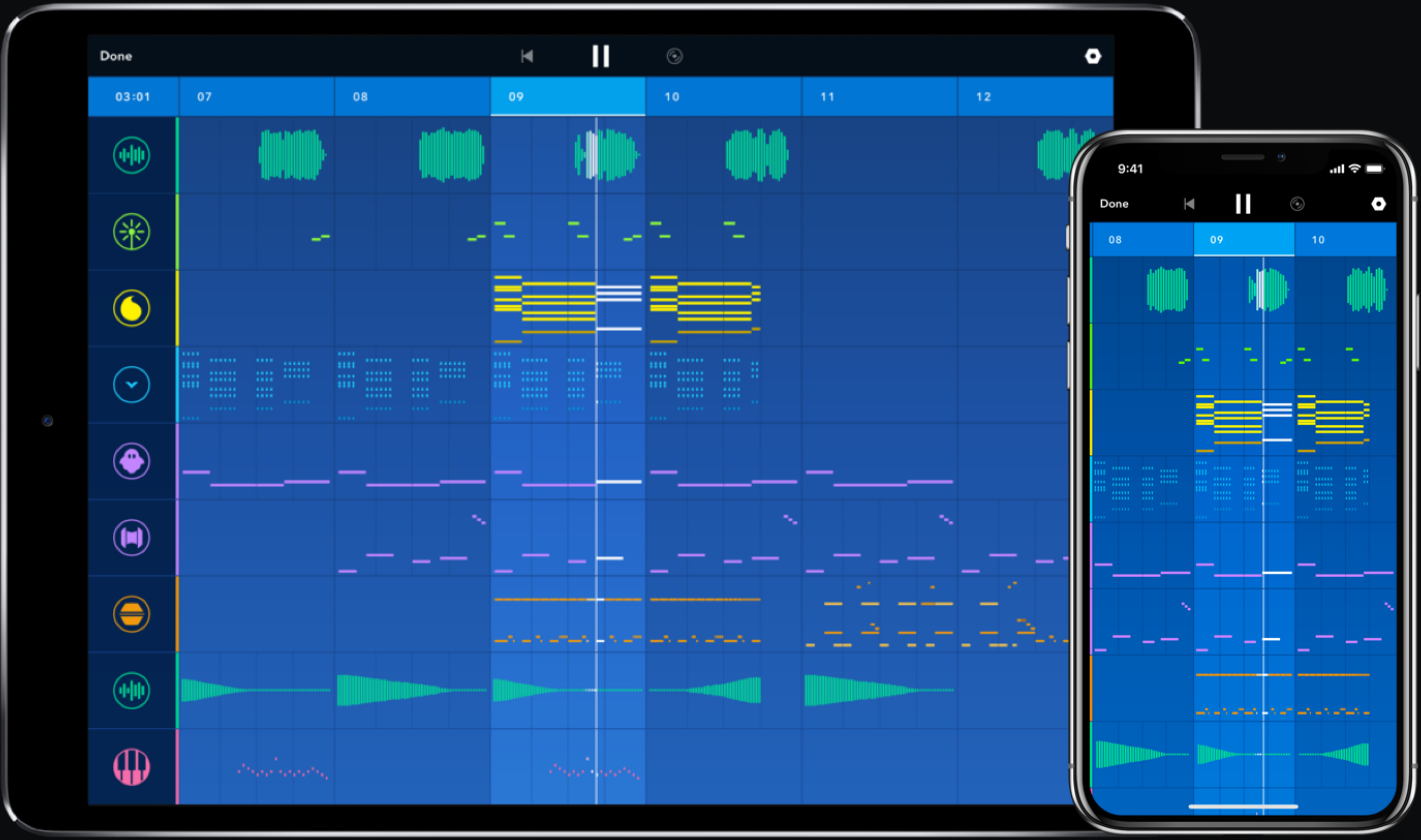
# Who Am I?

I'm Basil Al-Dajane,
Co-Founder of Medly

Medly is a music-making app for iOS

Been using OpenGL since 2012 and Metal for a year and a half now




Done
03:01
07
08
09
10
11
12
9:41
Done
08
09
10

# Last Metal Talk

Chris Feher, of Shopify, talked about use of Metal to create a dynamically updating map

Peak sales / minute
$1,138,574

# But Metal Can Do More…

While OpenGL may be cross platform, and not have as much boiler plate, it’s one big state machine

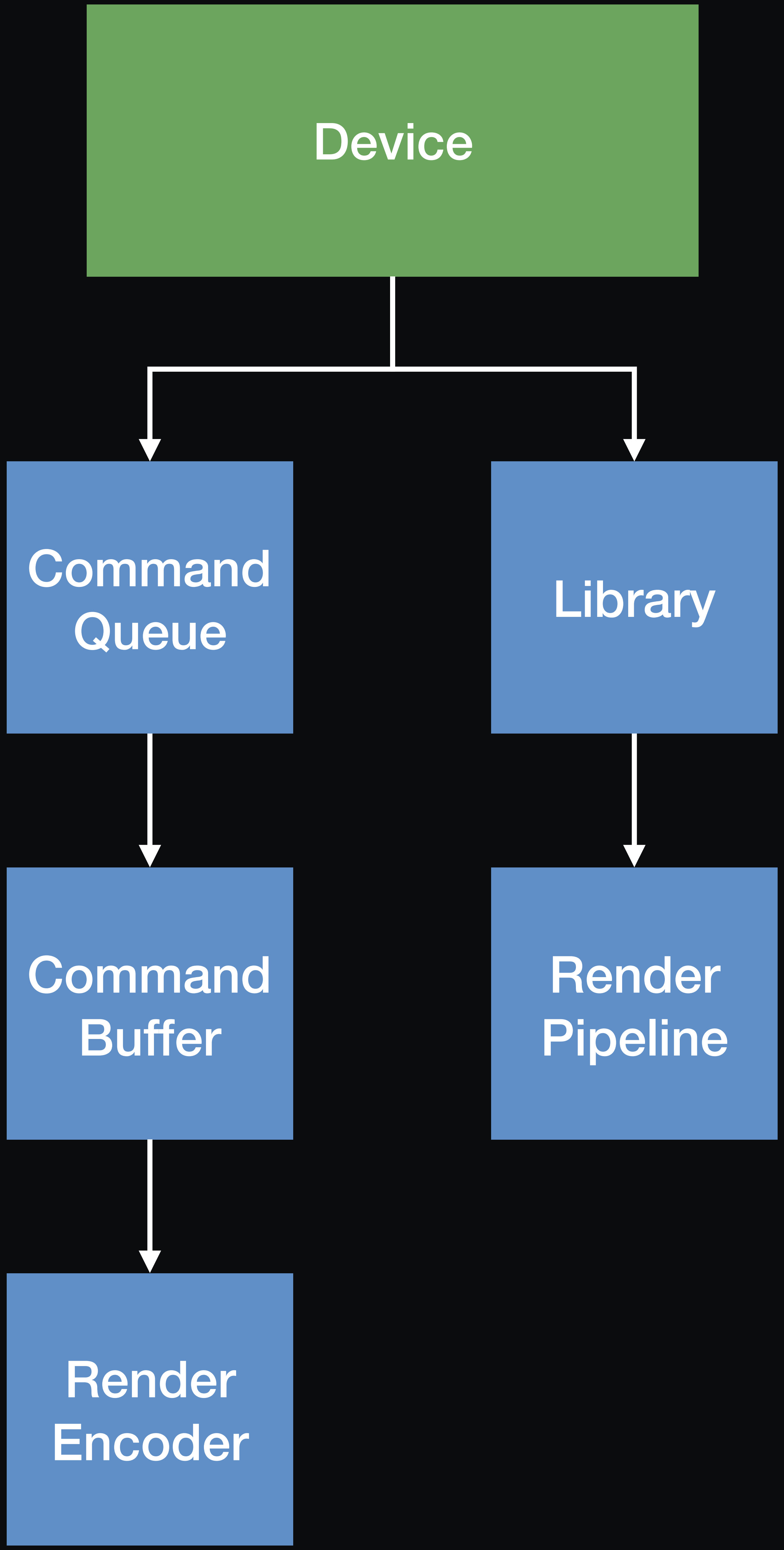
Metal is designed for multithreaded applications

Consistent in the way it draws

> *Always draws to texture*

Make sense of the boiler plate!

A lot more not in this talk

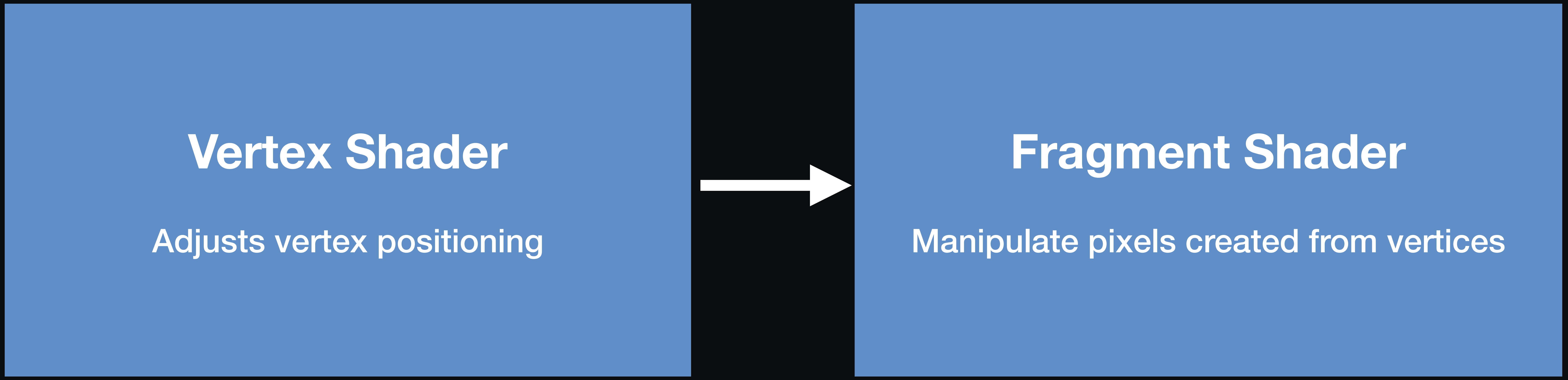

Device
Command Queue
Library
Command Buffer
Render Pipeline
Render Encoder

# Render Encoder

What you'll use to render objects
By setting up resources such as buffers and textures

And finally calling draw

# Render Pipeline

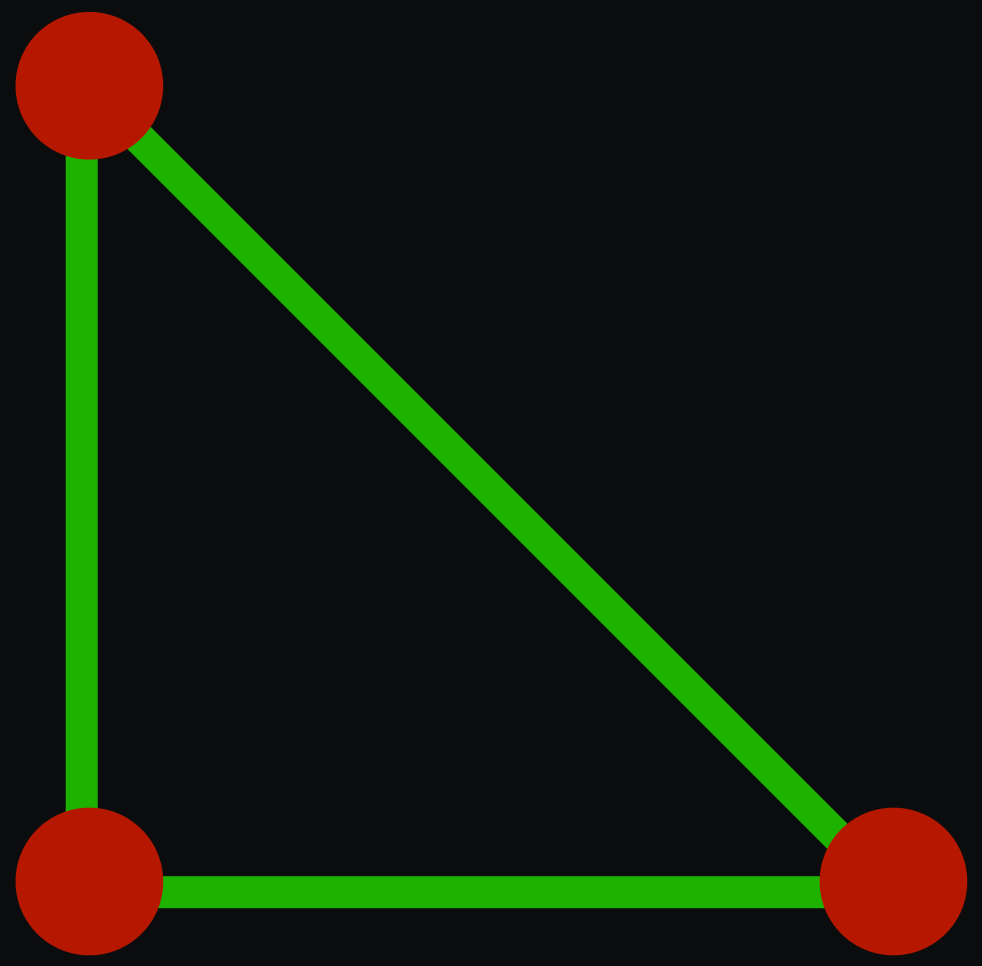


**Vertex Shader**

Adjusts vertex positioning

**Fragment Shader**

Manipulate pixels created from vertices

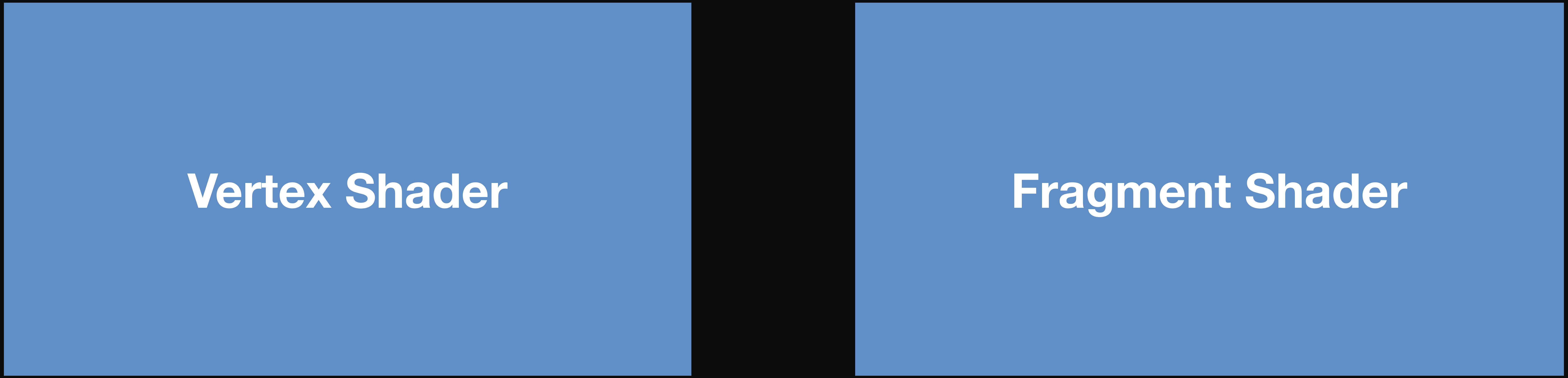


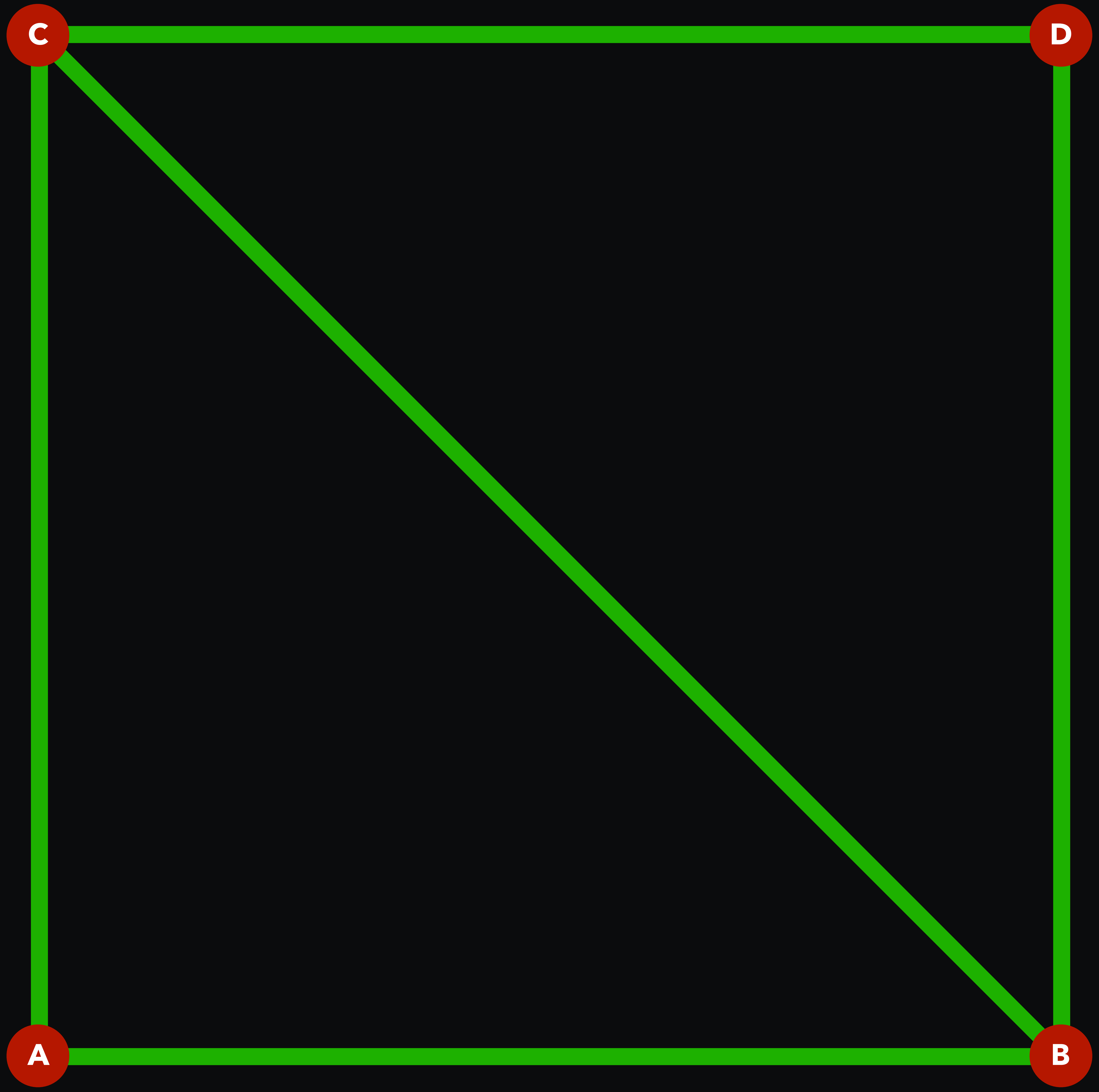
# Render Encoder

Can send arguments and resources to either shader

```
geometryData [[buffer(0)]]
constants    [[buffer(1)]]
objectData   [[buffer(2)]]
```

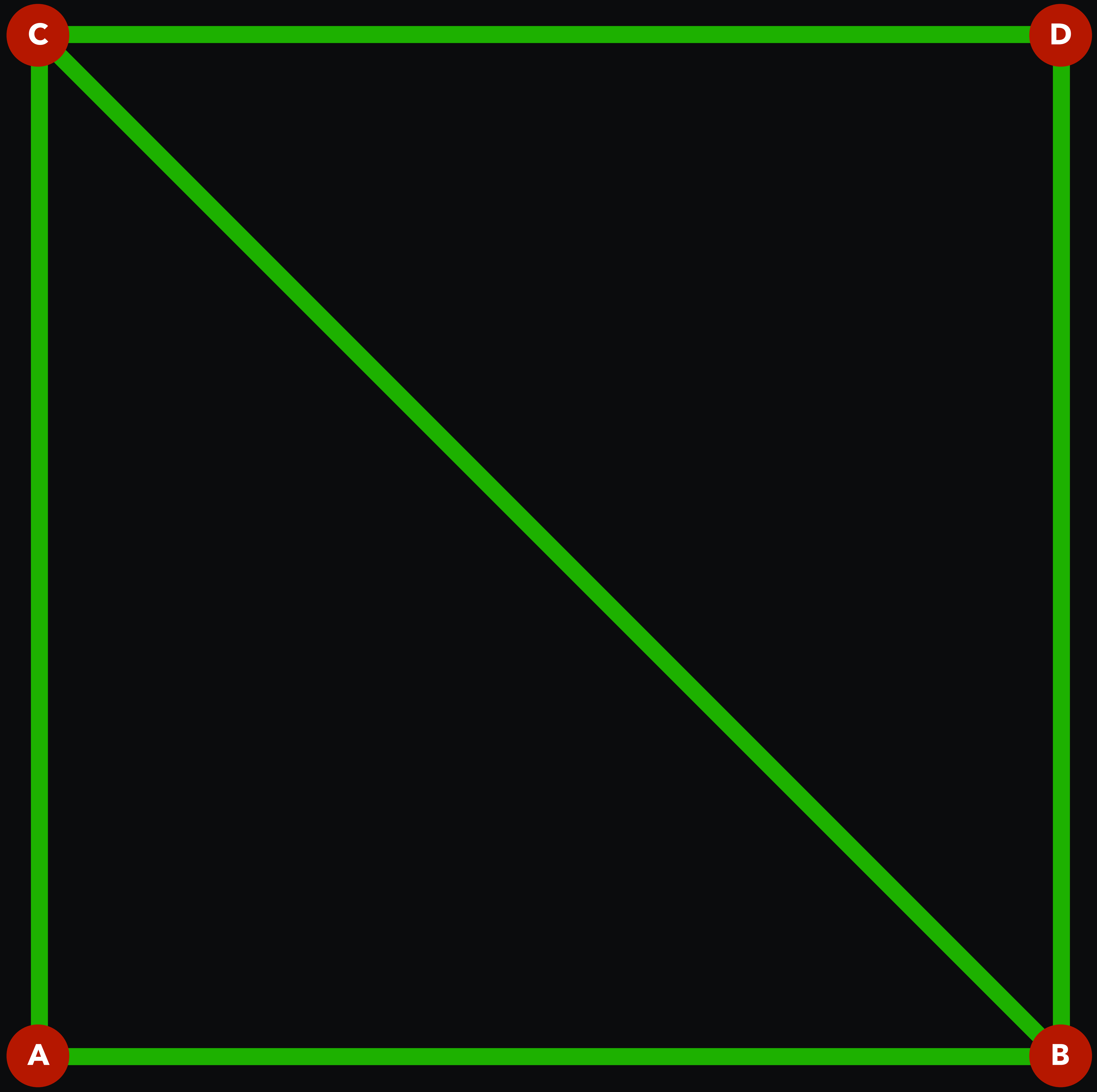
```
dataFromVertex [[stage_in]]
destination    [[color(0)]]
texture        [[texture(0)]]
screenData     [[buffer(0)]]
```





```
typedef struct ColorVertex {
    vector_float2 position;
    float r, g, b, a;
} ColorVertex;
```

|   |   |
|---|---|
| A | 0 |
| B | 1 |
| C | 2 |
| B | 1 |
| C | 2 |
| D | 3 |



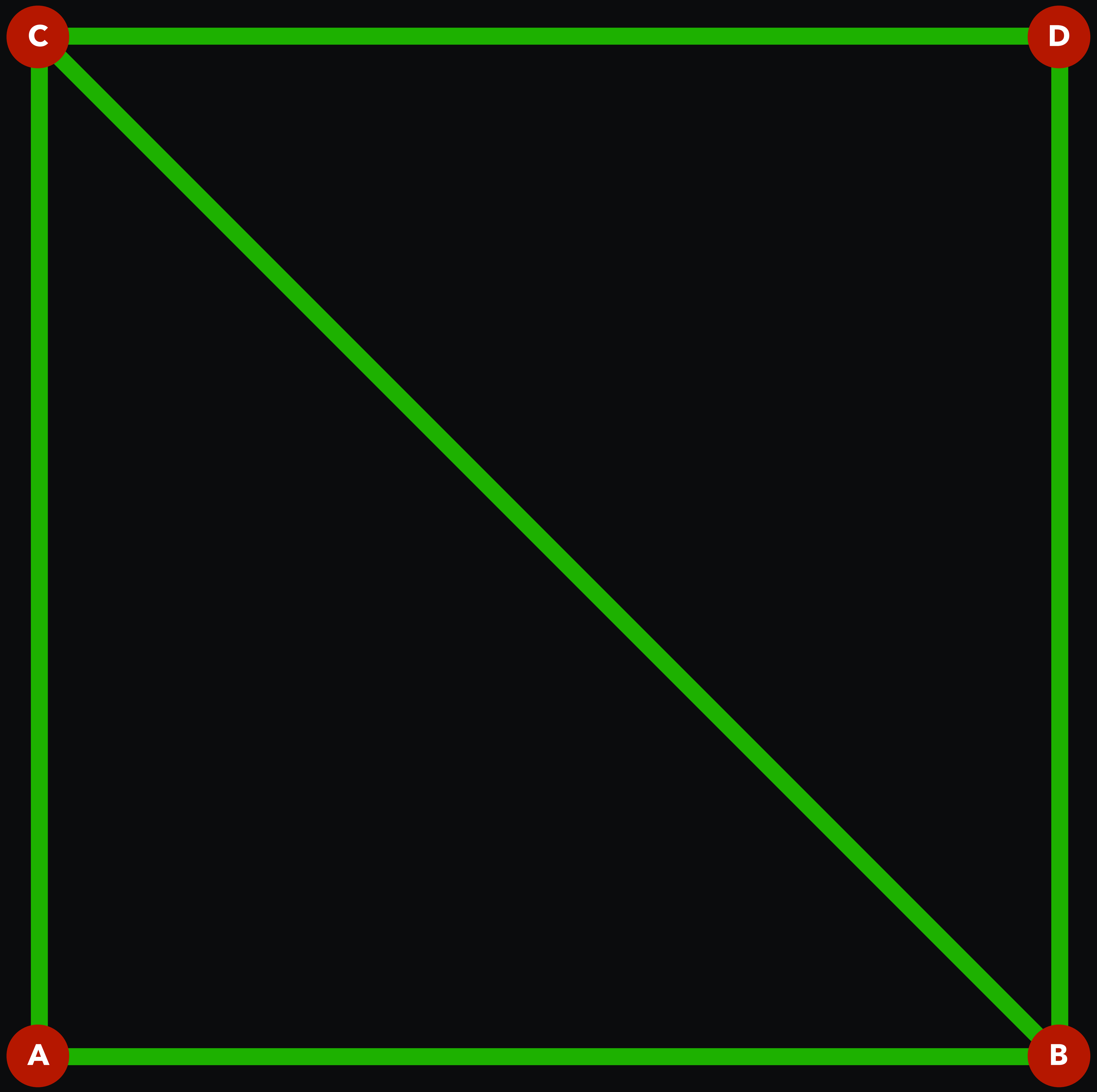
```
typedef struct ColorVertex {
    vector_float2 position;
    float r, g, b, a;
} ColorVertex;
```

| | |
|---|---|
| A | 0 |
| B | 1 |
| C | 2 |

| | |
|---|---|
| B | 1 |
| C | 2 |
| D | 3 |

Naive

6 Floats * 6 Vertices
= (6 * 4 bytes) * 6
= 144 bytes



```
typedef struct ColorVertex {
    vector_float2 position;
    float r, g, b, a;
} ColorVertex;
```

Re-use vertices

Geometry

6 Floats * 4 Vertices

= (6 * 4 bytes) * 4

= 96 bytes

Index

2 UInt16 * 6 indices

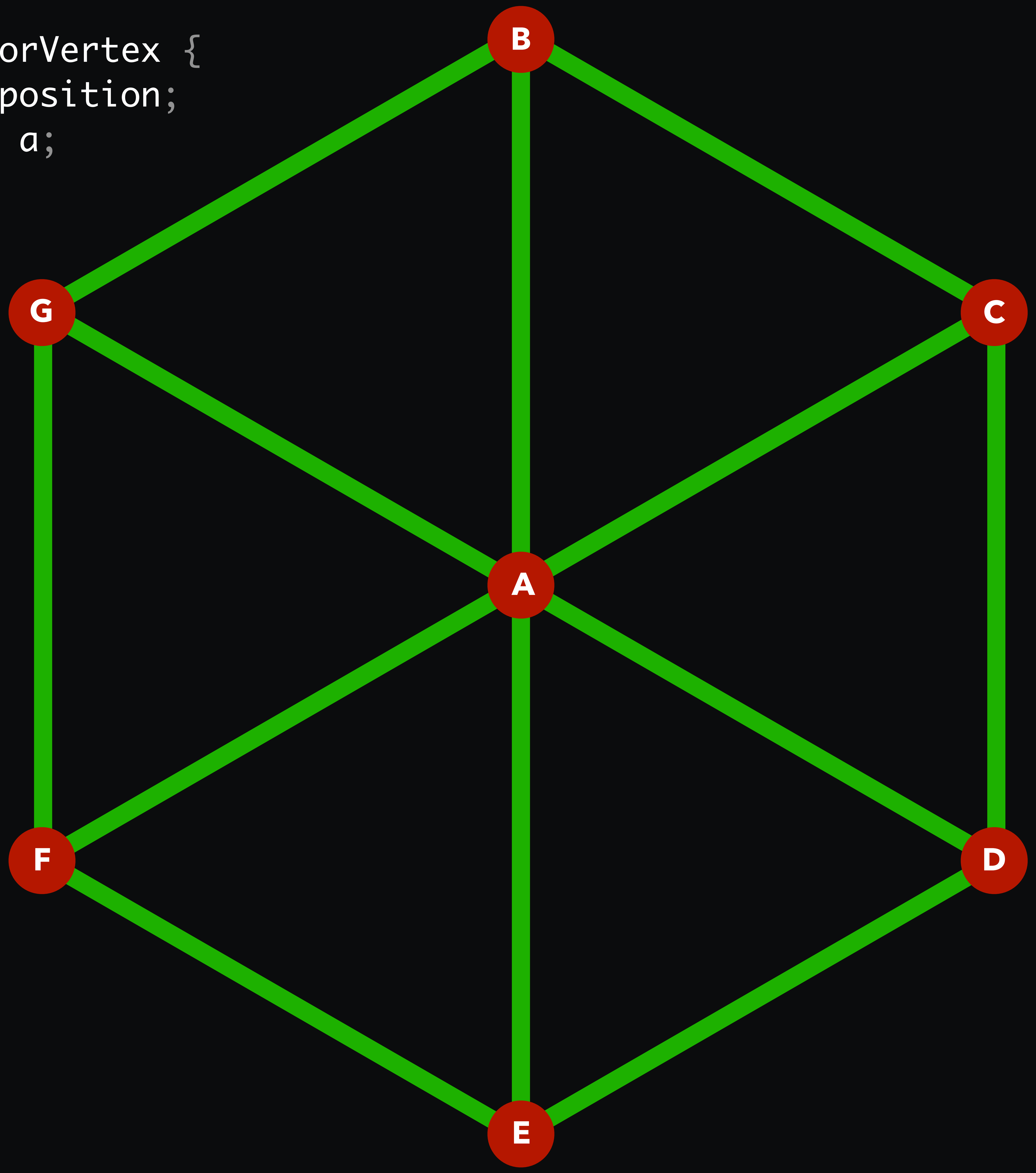
= (2 * 2 bytes) * 6

= 24 bytes

Total = 120 bytes

Versus 144 bytes

```
typedef struct ColorVertex {
    vector_float2 position;
    float r, g, b, a;
} ColorVertex;
```



## Naive

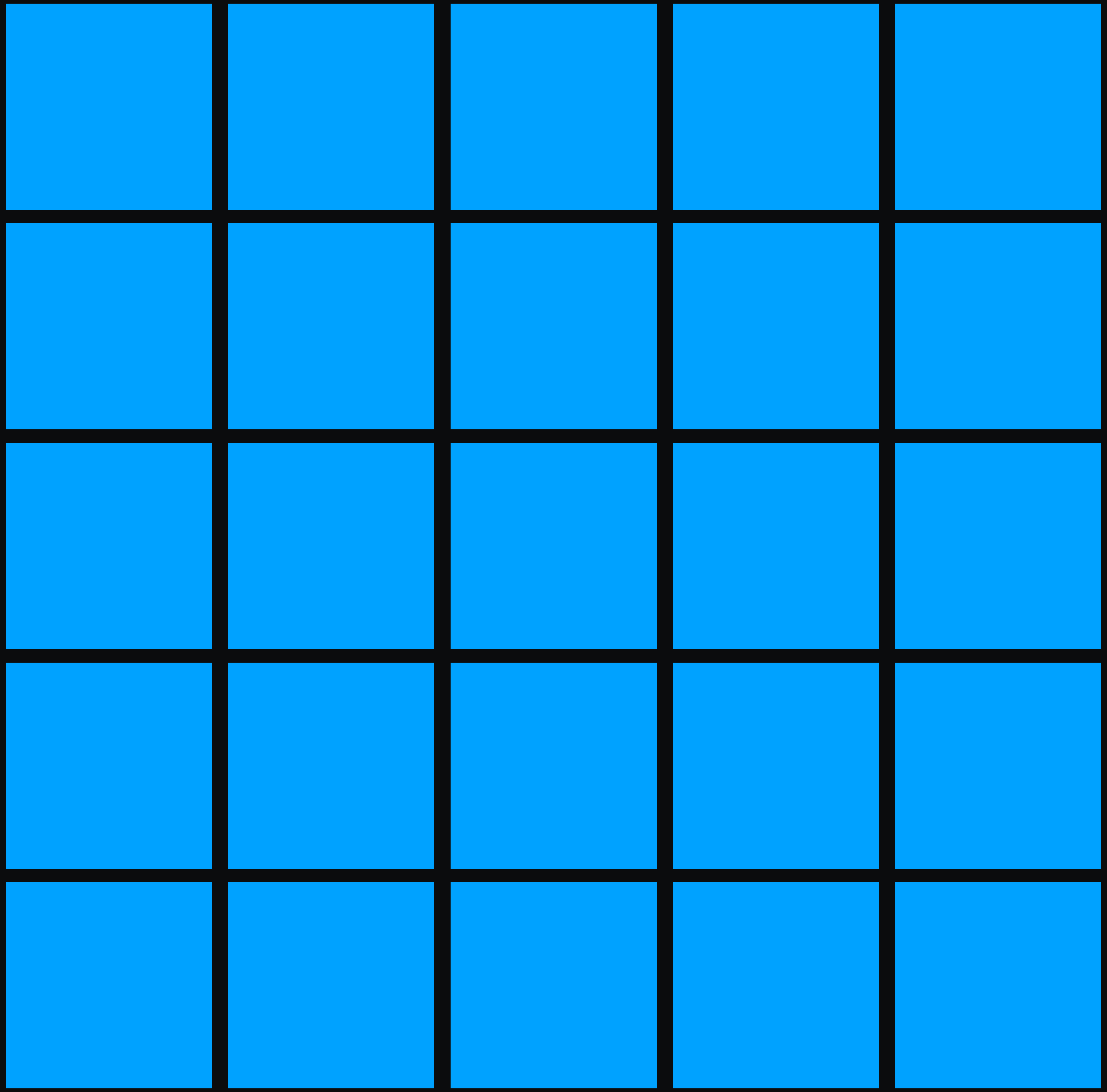
  6 Floats * (3 * 6) Vertices
= (6 * 4 bytes) * 18
= 432 bytes

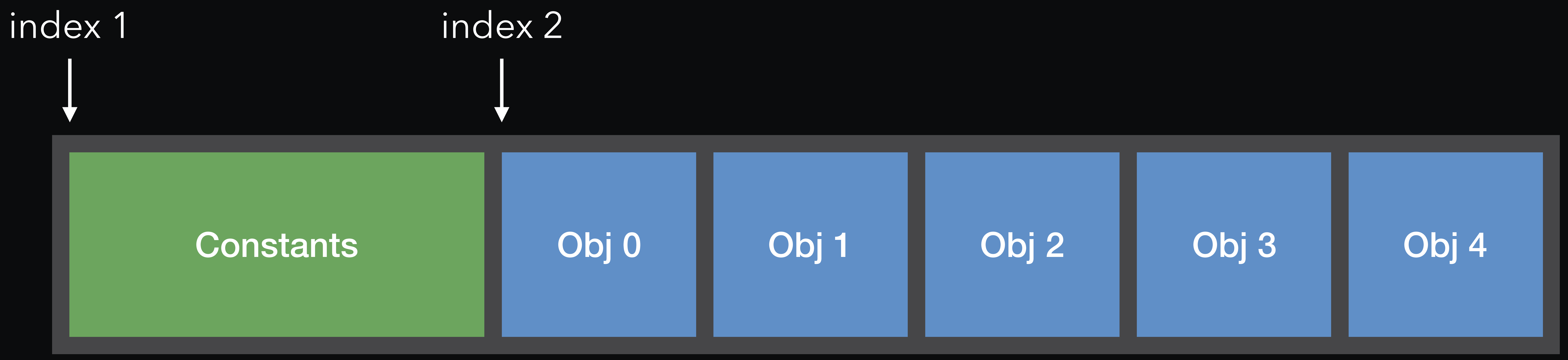
## Re-use vertices

Geometry
  6 Floats * 7 Vertices
= (6 * 4 bytes) * 7
= 168 bytes

Index
  2 UInt16 * 18 indices
= (2 * 2 bytes) * 18
= 72 bytes

Total 240 bytes, 44% less





index 1

index 2

Constants

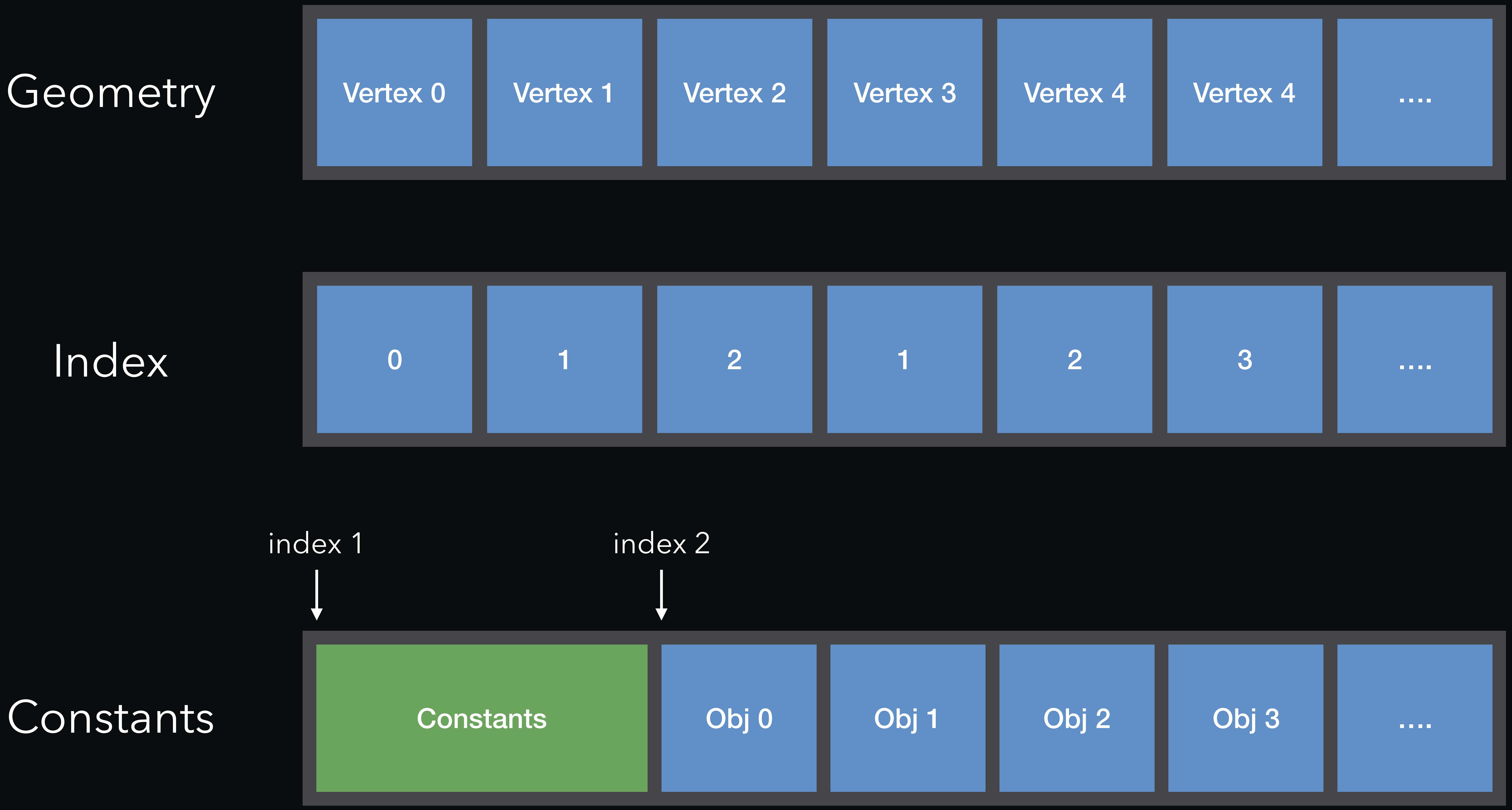
Obj 0

Obj 1

Obj 2

Obj 3

Obj 4



Geometry

Vertex 0 | Vertex 1 | Vertex 2 | Vertex 3 | Vertex 4 | Vertex 4 | ....

Index

0 | 1 | 2 | 1 | 2 | 3 | ....

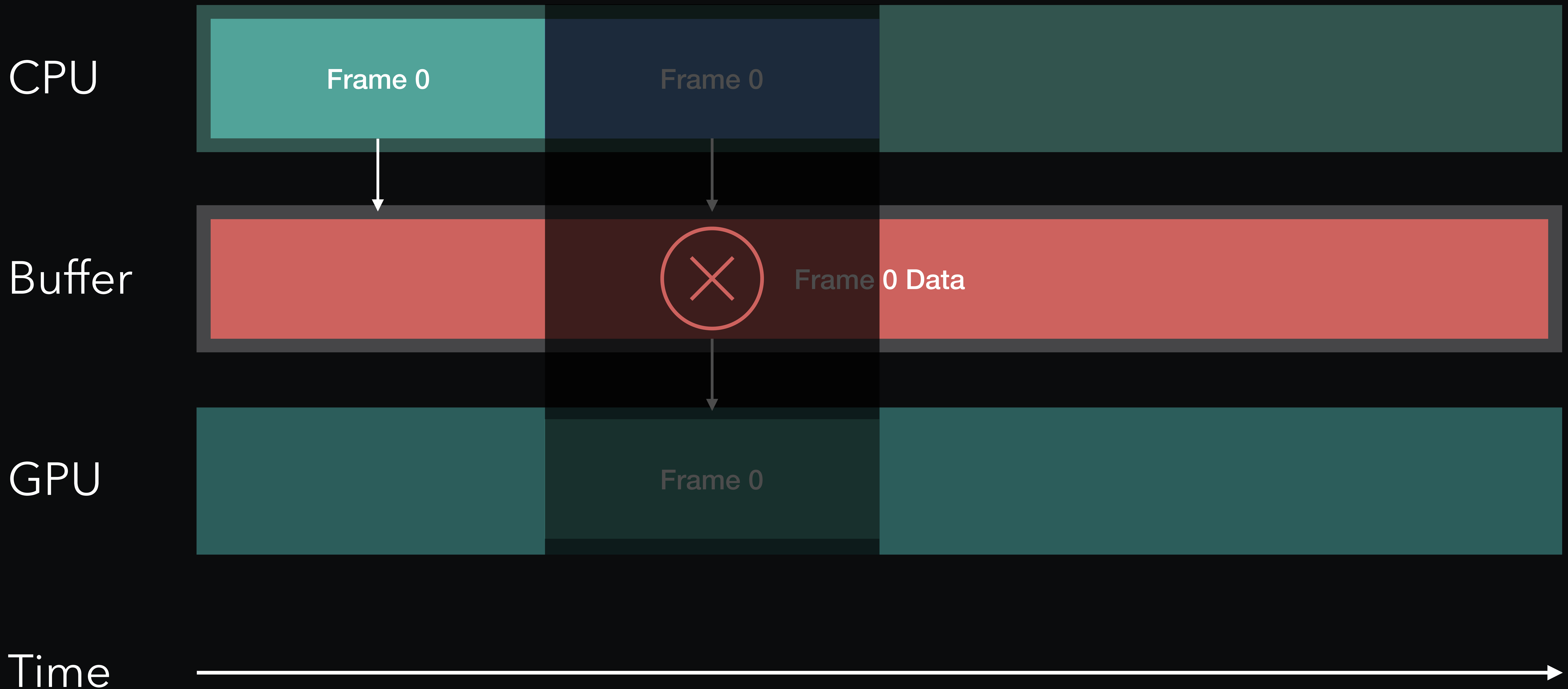
index 1

index 2

Constants

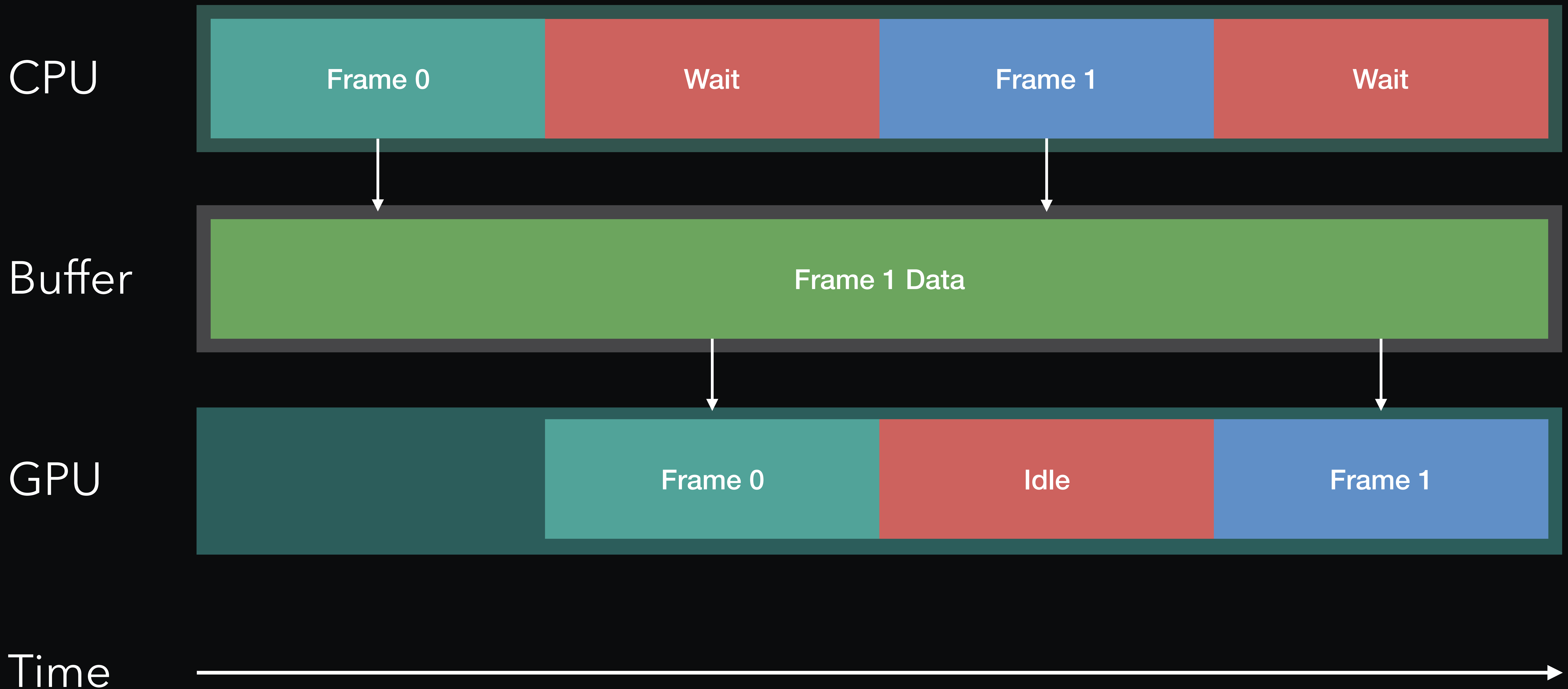
Constants | Obj 0 | Obj 1 | Obj 2 | Obj 3 | ....

# Resource Contention



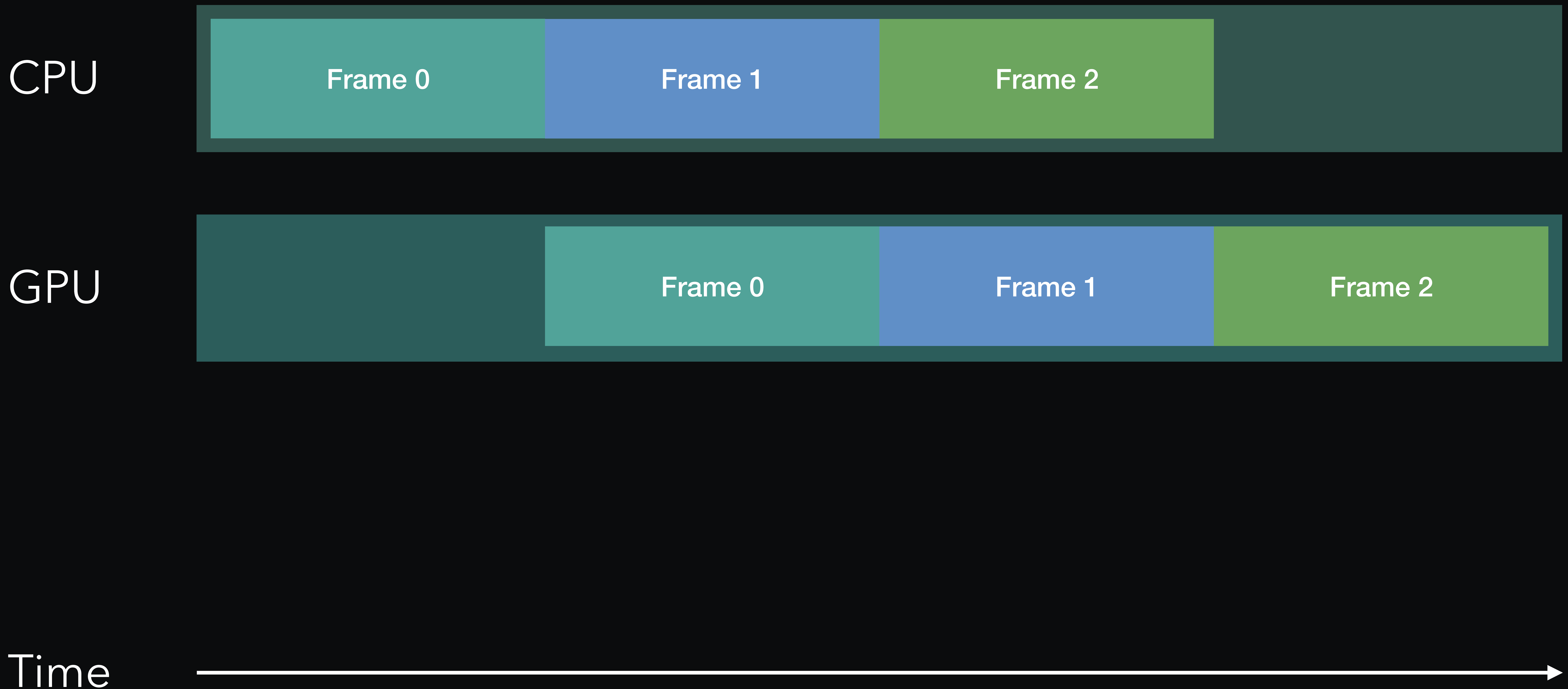
Adapted from WWDC 2016 Session 603 - Adopting Metal, Part 2

# Naive Synchronization



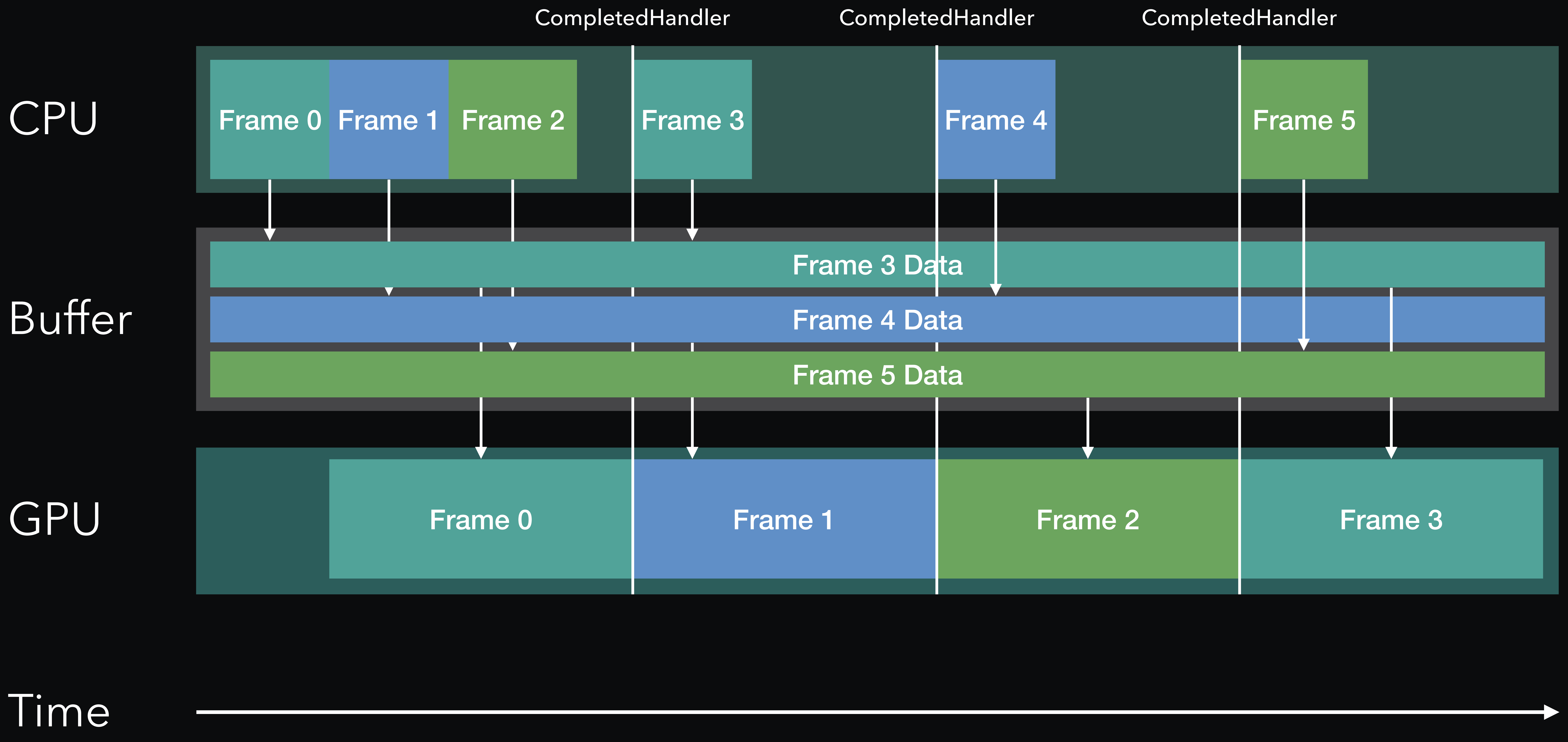
Adapted from WWDC 2016 Session 603 - Adopting Metal, Part 2

# Ideal Workload



Adapted from WWDC 2016 Session 603 - Adopting Metal, Part 2

# Triple Buffering



Adapted from WWDC 2016 Session 603 - Adopting Metal, Part 2

# Triple Buffering Implementation

Generate

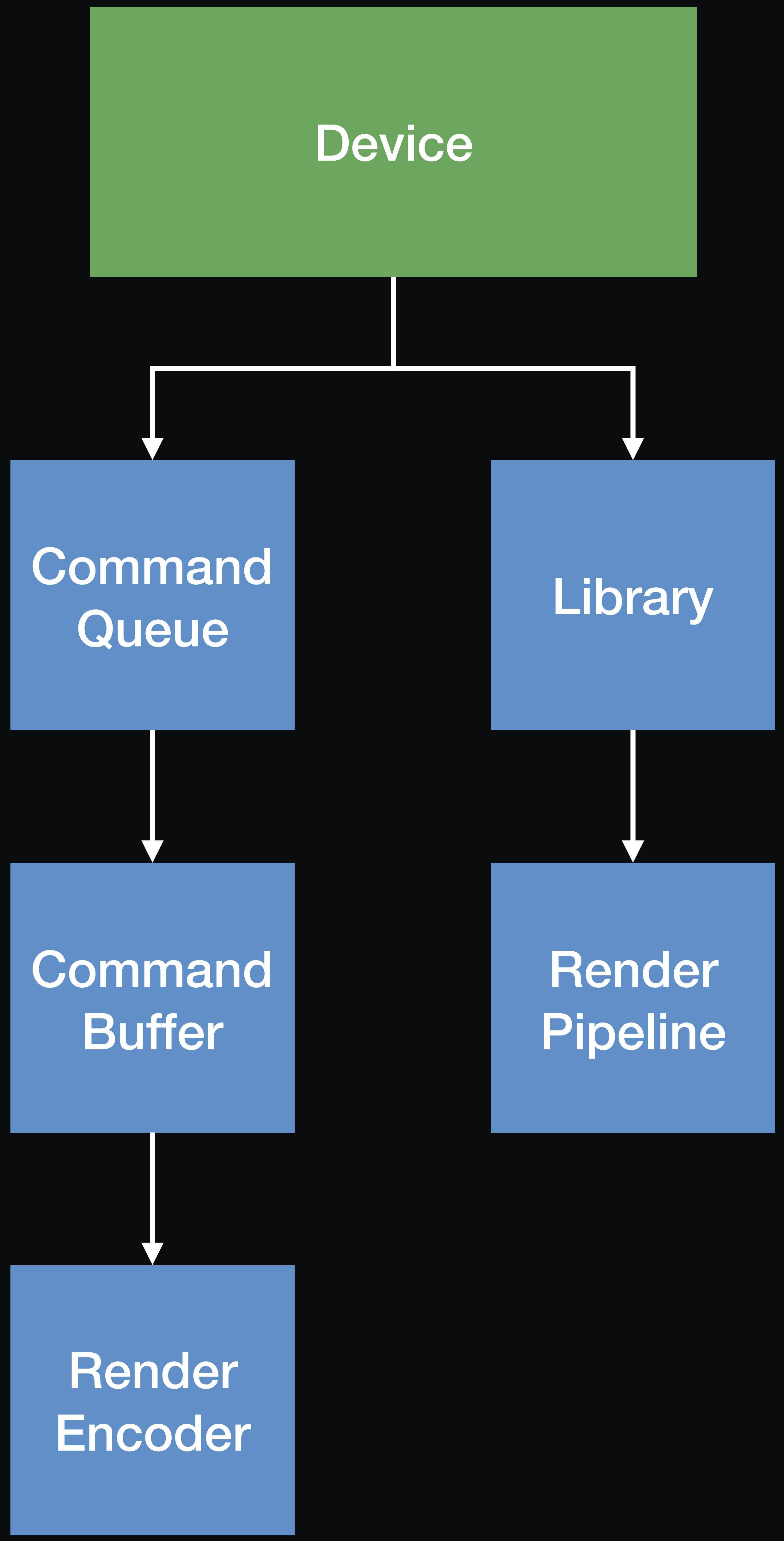
- For each frame, generate all data from scratch and write into current buffer.

Copy & Modify

- Copy previous buffer into current buffer, then apply modifications

Copy When Dirty

- When writing to a buffer, mark others as dirty, and mark current as clean
- When re-using buffer, if still clean you can just start updating it, if dirty copy from previous buffer before updating



Device

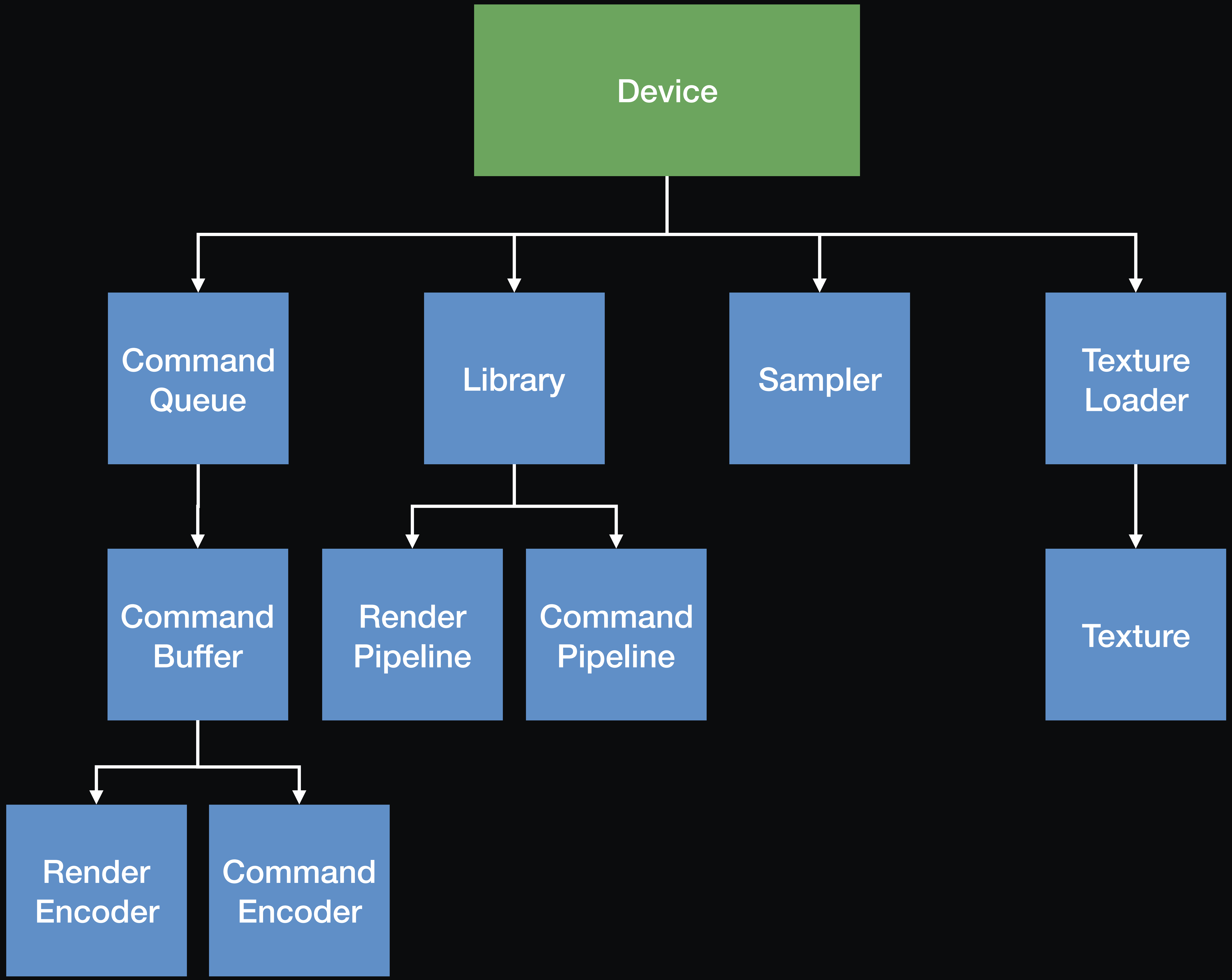
Command Queue

Library

Command Buffer

Render Pipeline

Render Encoder



Device

Command Queue

Library

Sampler

Texture Loader

Command Buffer

Render Pipeline

Command Pipeline

Texture

Render Encoder

Command Encoder

# Additional Resources

| | |
|---|---|
| Adopting Metal, Part 1 | https://developer.apple.com/videos/play/wwdc2016/602/ |
| Adopting Metal, Part 2 | https://developer.apple.com/videos/play/wwdc2016/603/ |
| Advanced Metal Shader Optimization | https://developer.apple.com/videos/play/wwdc2016/606/ |
| Introducing Metal 2 | https://developer.apple.com/videos/play/wwdc2017/601/ |
| Metal 2 Optimization and Debugging | https://developer.apple.com/videos/play/wwdc2017/607/ |
| Metal 2 on A11 Processors | https://developer.apple.com/videos/fall2017/ |

basil@medlylabs.com

Sample code and slides can be found at:
https://github.com/baldajan/MetalGrid