Triangles Are Precious - Let's Treat Them With Care

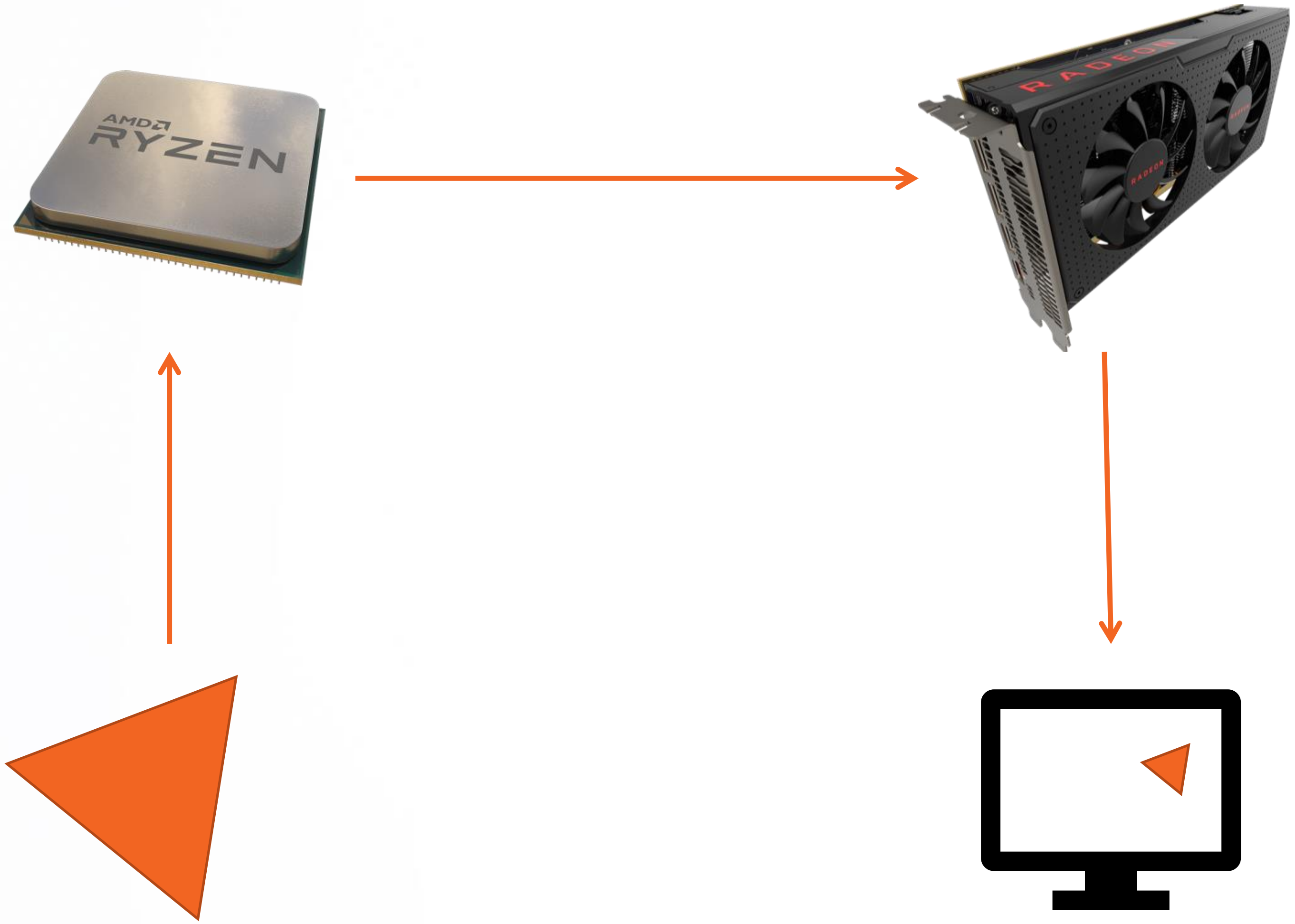
Dominik Baumeister
Developer Technology Engineer
Advanced Micro Devices, Inc.

This talk aims to provide an in-depth look at the **journey a triangle takes on a modern graphics card** before it can be displayed as shiny pixels on the screen.

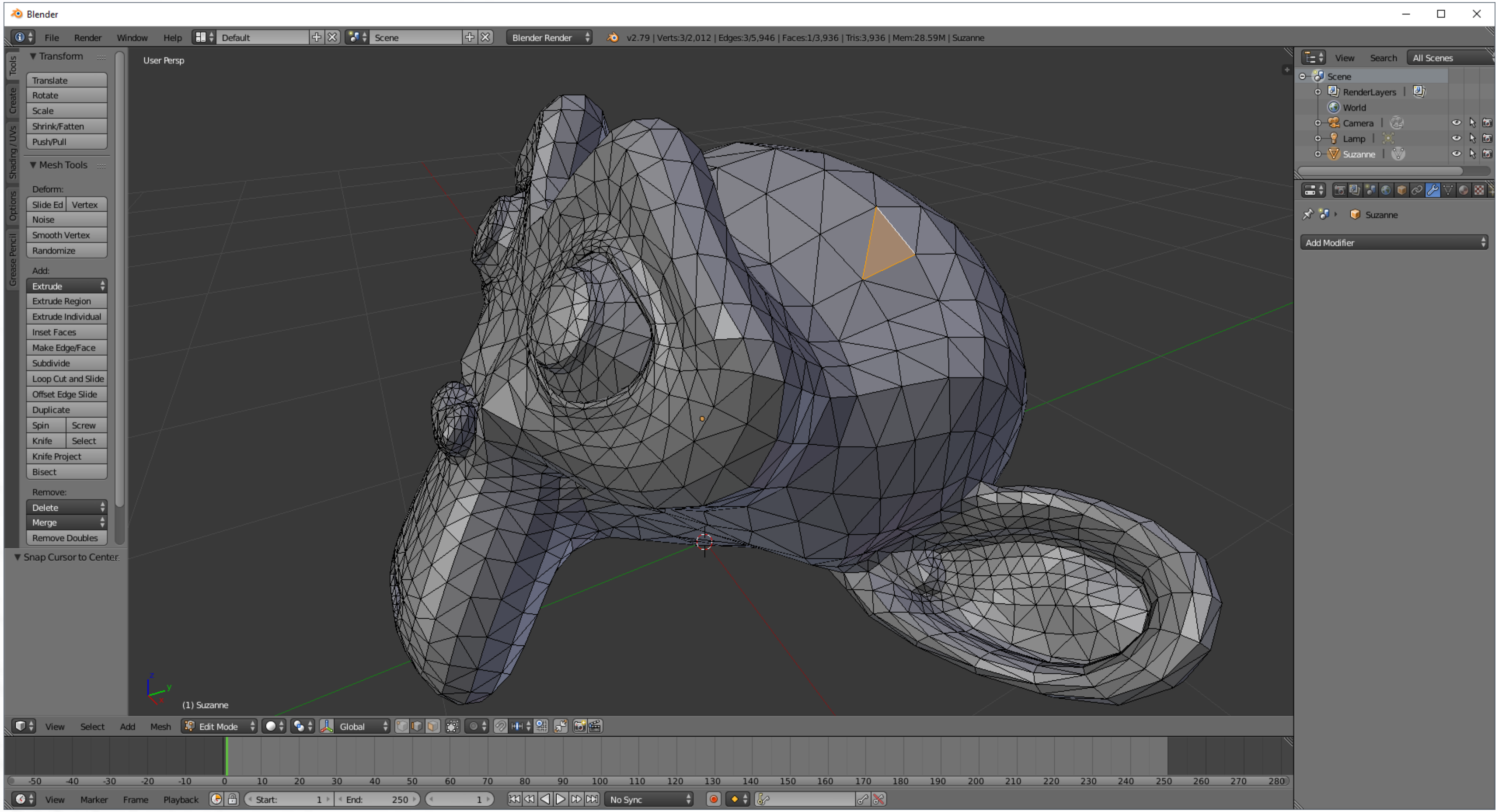
Preface

- Everything here was produced by the brilliant engineers of AMD.
- I **will** use lots of pictures.
- I **won't** use any math.
- I **will** show some code.
- I will barely scratch the surface.

The Whole Journey at a Glance

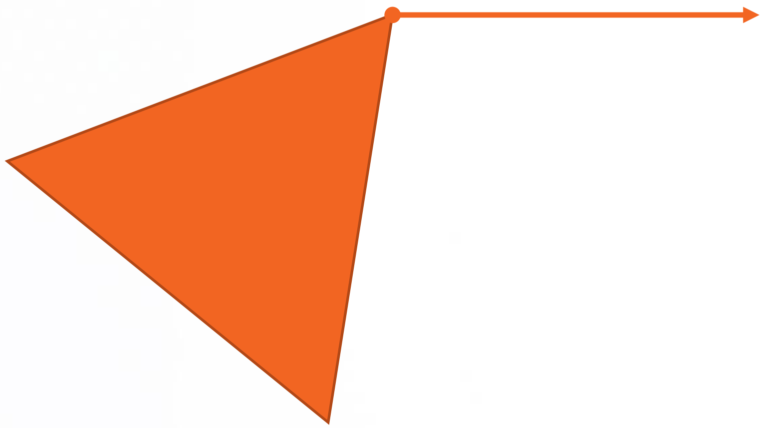


Creation Process – 3D Modelling



https://www.blender.org/

Creation Process – Vertex



Vertex

Position

Normal Vector

Texture Coordinates

…

AMD

Creation Process - Export

Blender

Positions
Normal Vectors
Texture Coordinates
Connectivity Information
…

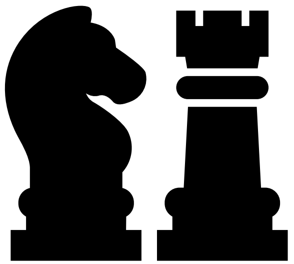
Export

.dae
.abc
.3ds
.fbx
.ply
.obj
.x3d
.stl
<custom>

Creation Process - Import

.dae
.abc
.3ds
.fbx
.ply
.obj
.x3d
.stl
<custom>

Import



Game Engine of your Choice
usually some custom format to represent geometry

Rendering – In Your Game Engine

Geometry

Engine
Specific
Format

→ Render Front End → Hardware Abstraction Layer → Graphics APIs →

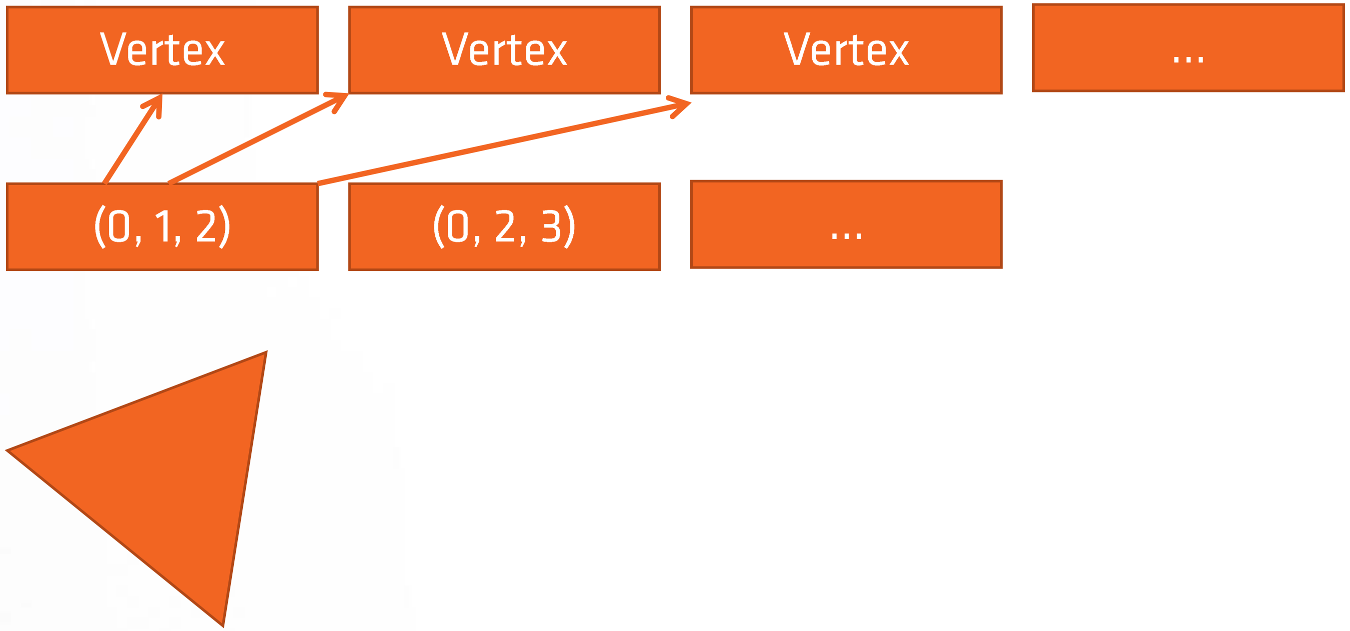
Mesh Creation
Visibility Testing
Bookkeeping

Vulkan®
D3D12
D3D11
OpenGL®
Metal
…

AMD

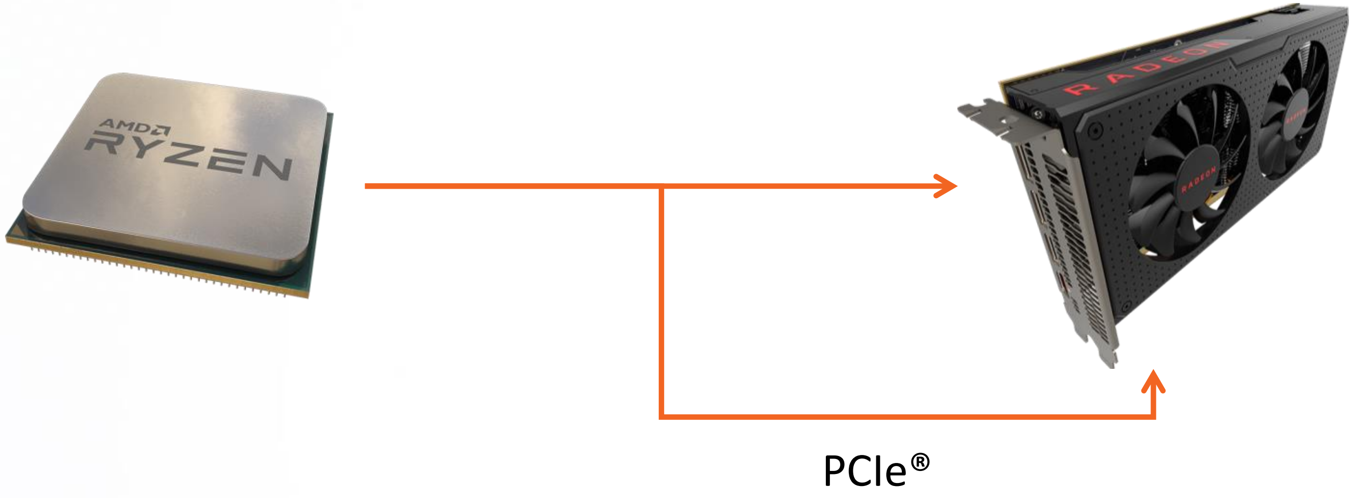
Rendering – Mesh Creation

- Similar procedure on all APIs.
- Create buffer for vertices.
- Create buffer for indices.
  This is an optimization!
  Remember the connectivity information attached to our mesh.



Rendering – Mesh Creation

- At some point: Ask Hardware Abstraction Layer to send these buffers to the GPU.
  Eventually triggers physical copies from local memory to video memory!



Rendering – In Your Game Engine

Geometry

Engine
Specific
Format

→ Render Front End → Hardware Abstraction Layer → Graphics APIs →

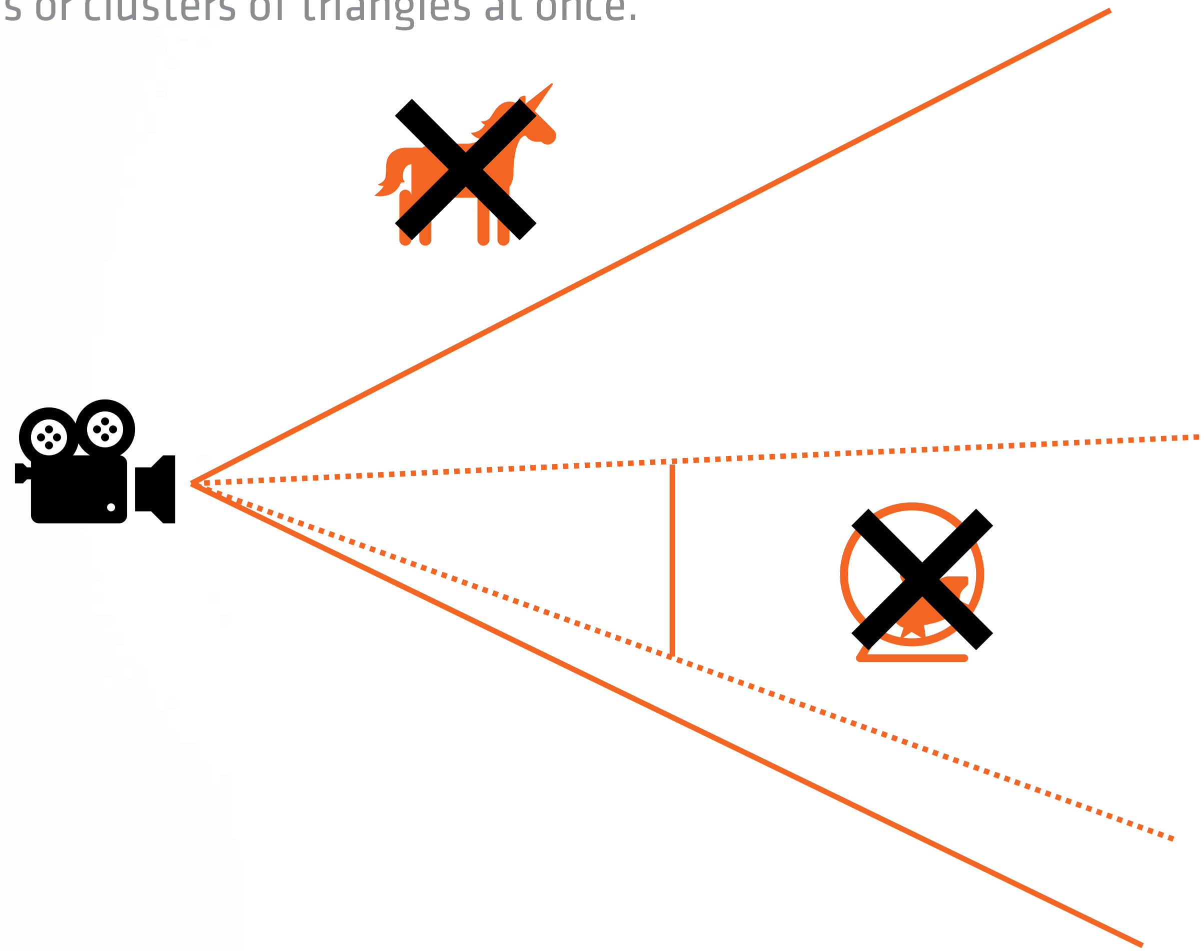
Mesh Creation
Visibility Testing
Bookkeeping

Vulkan®
D3D12
D3D11
OpenGL®
Metal
…

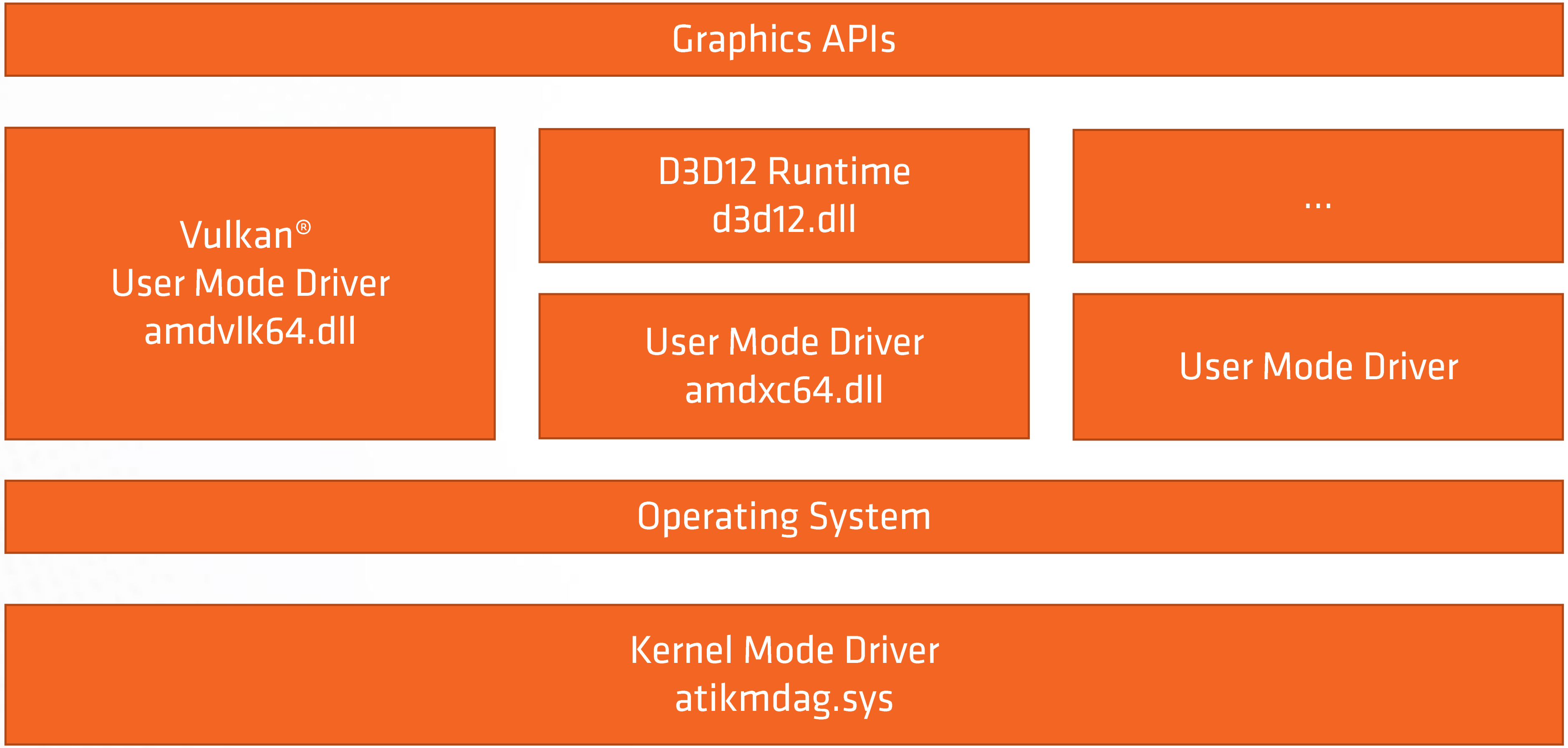
AMD

Rendering – Visibility Testing

- Avoid asking the graphics card to do superfluous work.
  In other words: Don't render objects you would not see anyways.
  Usually performed on whole meshes or clusters of triangles at once.
- Frustum Culling
- Occlusion Culling



Rendering – In the Graphics Driver



Rendering – In the Graphics Driver

Vulkan®

User Mode Driver
amdvlk64.dll

Operating System

Kernel Mode Driver

Concept of Command Buffers

Prepare a bunch of commands that tell the GPU what to do.

vkCmdBindPipeline

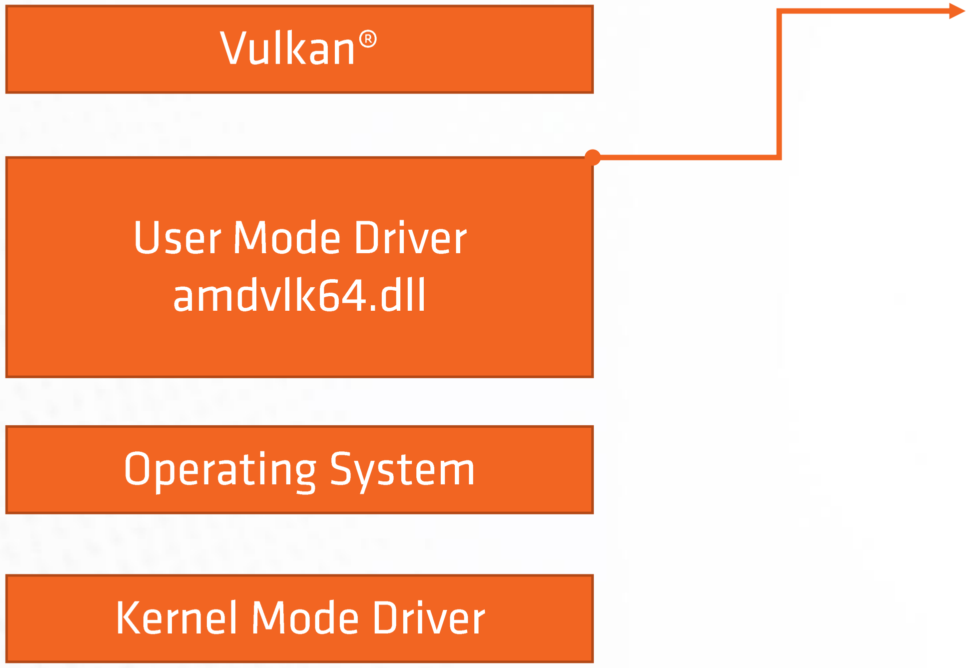
vkCmdBindVertexBuffers

vkCmdBindIndexBuffer

vkCmdDrawIndexed

AMD

Rendering – In the Graphics Driver

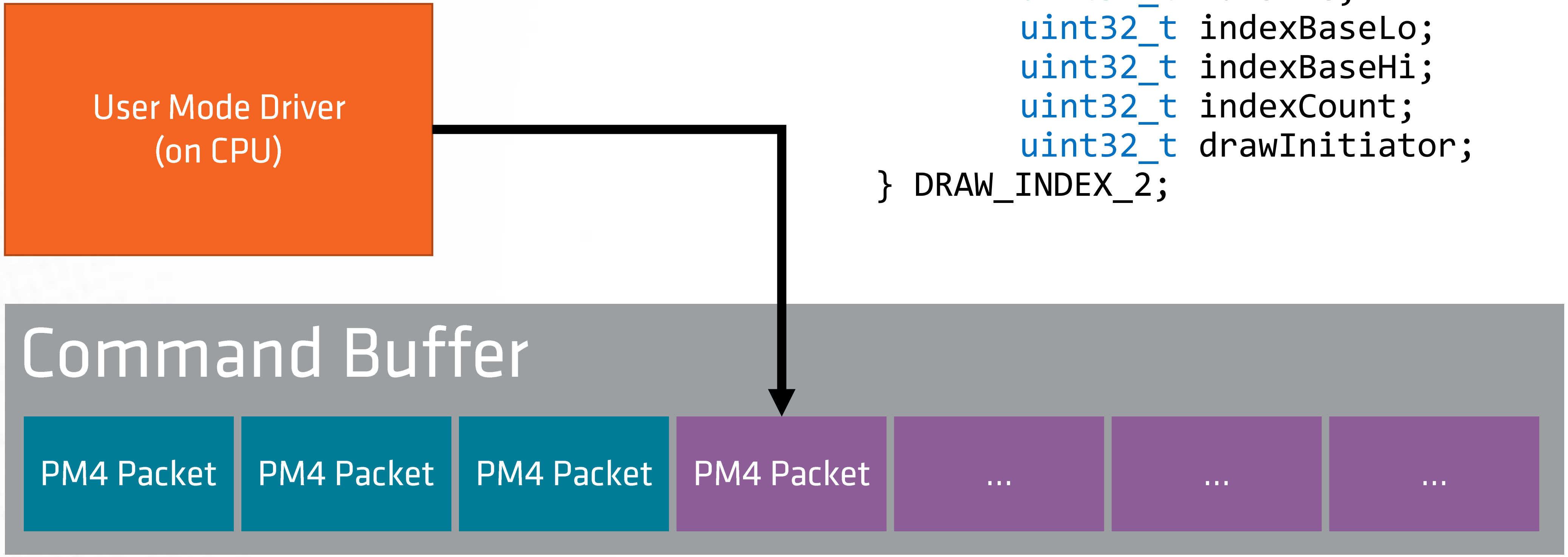


PM4 Packets

Represent the API commands in a way that can be executed by the GPU.
Linux® driver is open source on Github!

Rendering – PM4

```
// Example PM4 packet.
typedef struct DRAW_INDEX_2 {
        uint32_t header;
        uint32_t maxSize;
        uint32_t indexBaseLo;
        uint32_t indexBaseHi;
        uint32_t indexCount;
        uint32_t drawInitiator;
} DRAW_INDEX_2;
```



Rendering – In the Graphics Driver

Vulkan®

User Mode Driver
amdvlk64.dll

Operating System

Kernel Mode Driver

Security Guard

Gets the prepared command buffer from the user mode driver.
Hands it over to the kernel mode driver.
Ensures that the user mode driver cannot hang the whole system.

Command Buffer

PM4 Packet	PM4 Packet	PM4 Packet	PM4 Packet	…	…	…

AMD

Rendering – In the Graphics Driver

Vulkan®

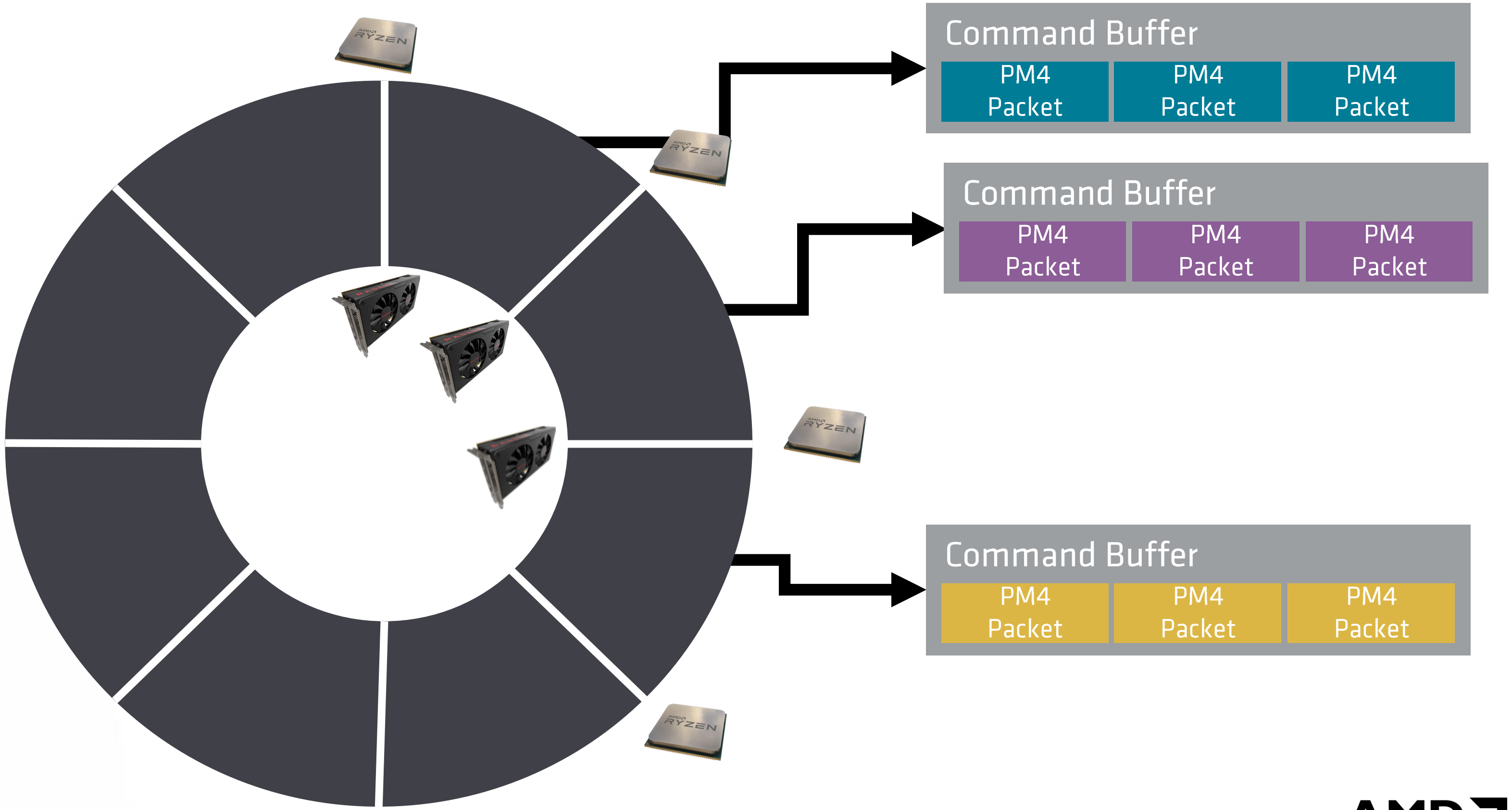
User Mode Driver
amdvlk64.dll

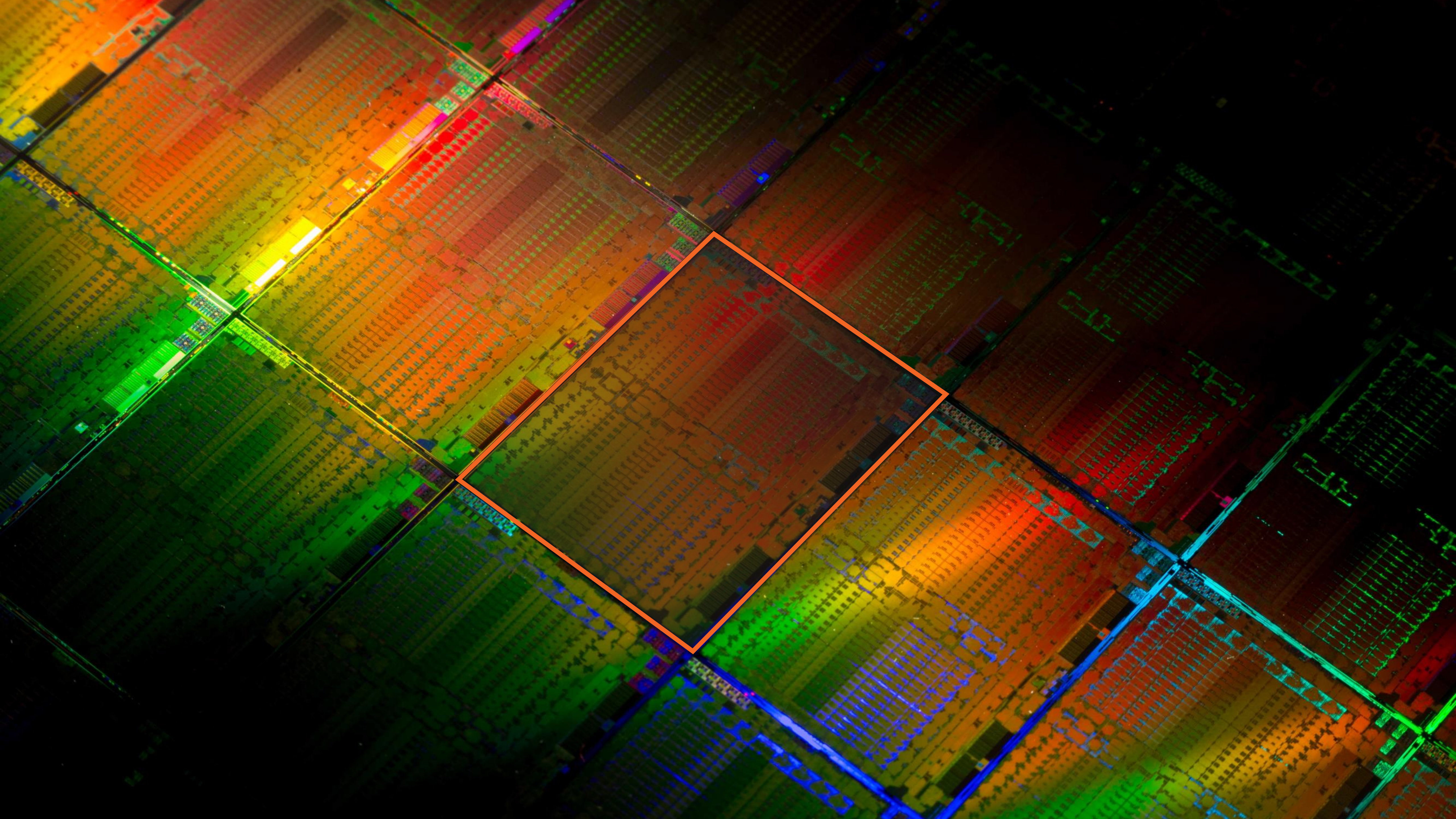
Operating System

Kernel Mode Driver

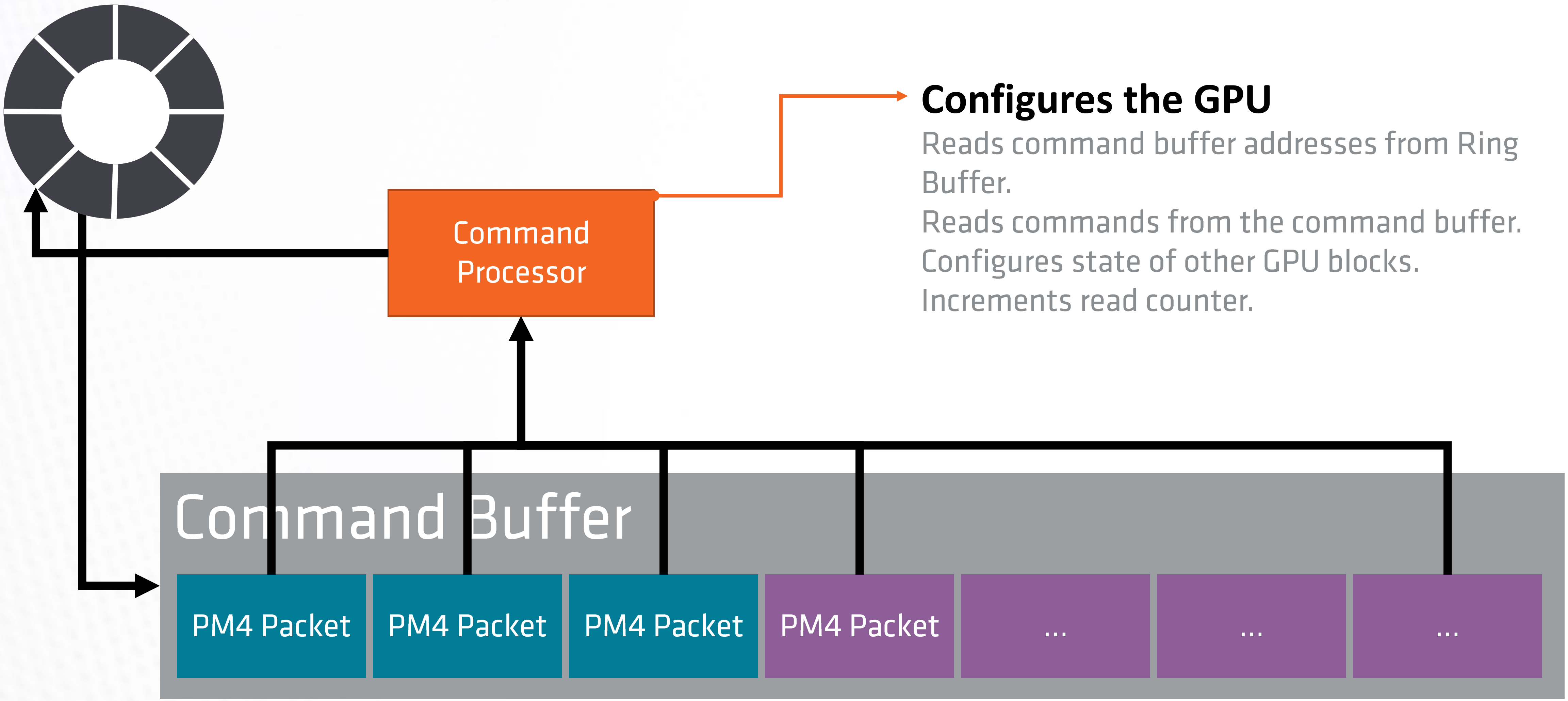
Ring Buffer

Single point of communication with the GPU.
Contains addresses to the command buffers.
CPU increments write pointer.
GPU increments read pointer.





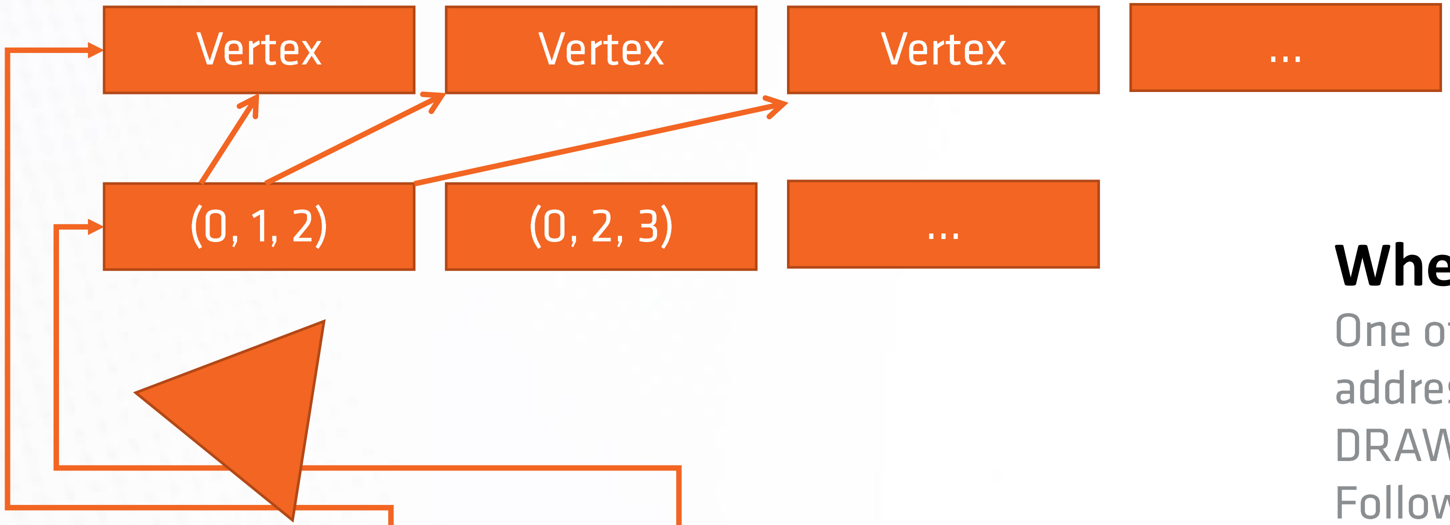
On the GPU – The Command Processor



Configures the GPU

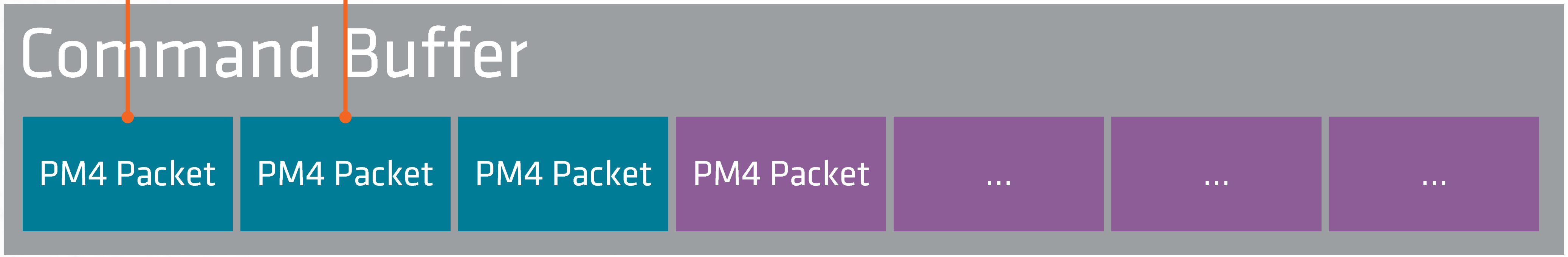
Reads command buffer addresses from Ring Buffer.
Reads commands from the command buffer.
Configures state of other GPU blocks.
Increments read counter.

On the GPU – Find the Triangle



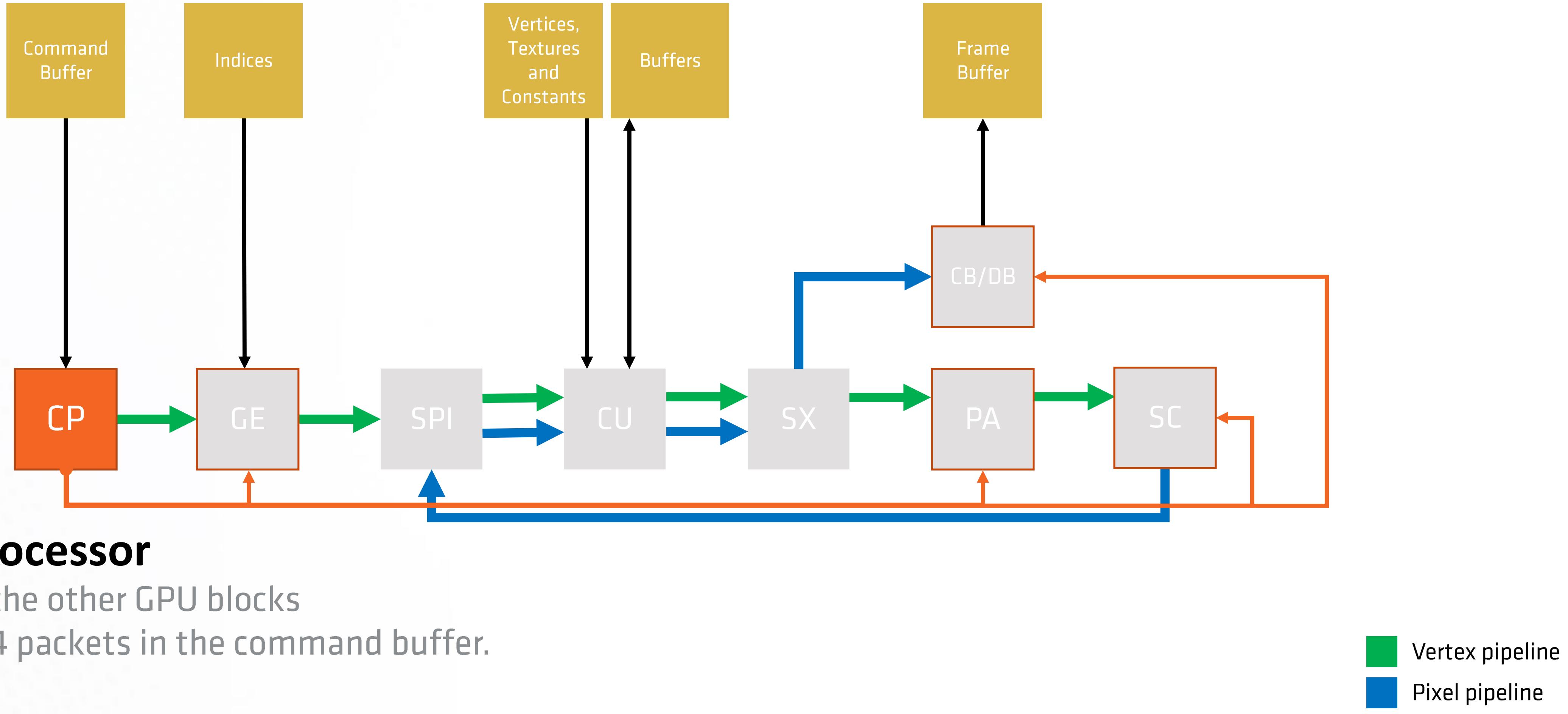
Where is our triangle?

One of the packets sets the vertex buffer addresses.
DRAW_INDEX_2 contains the index buffer address.
Follow the pointers!



AMD

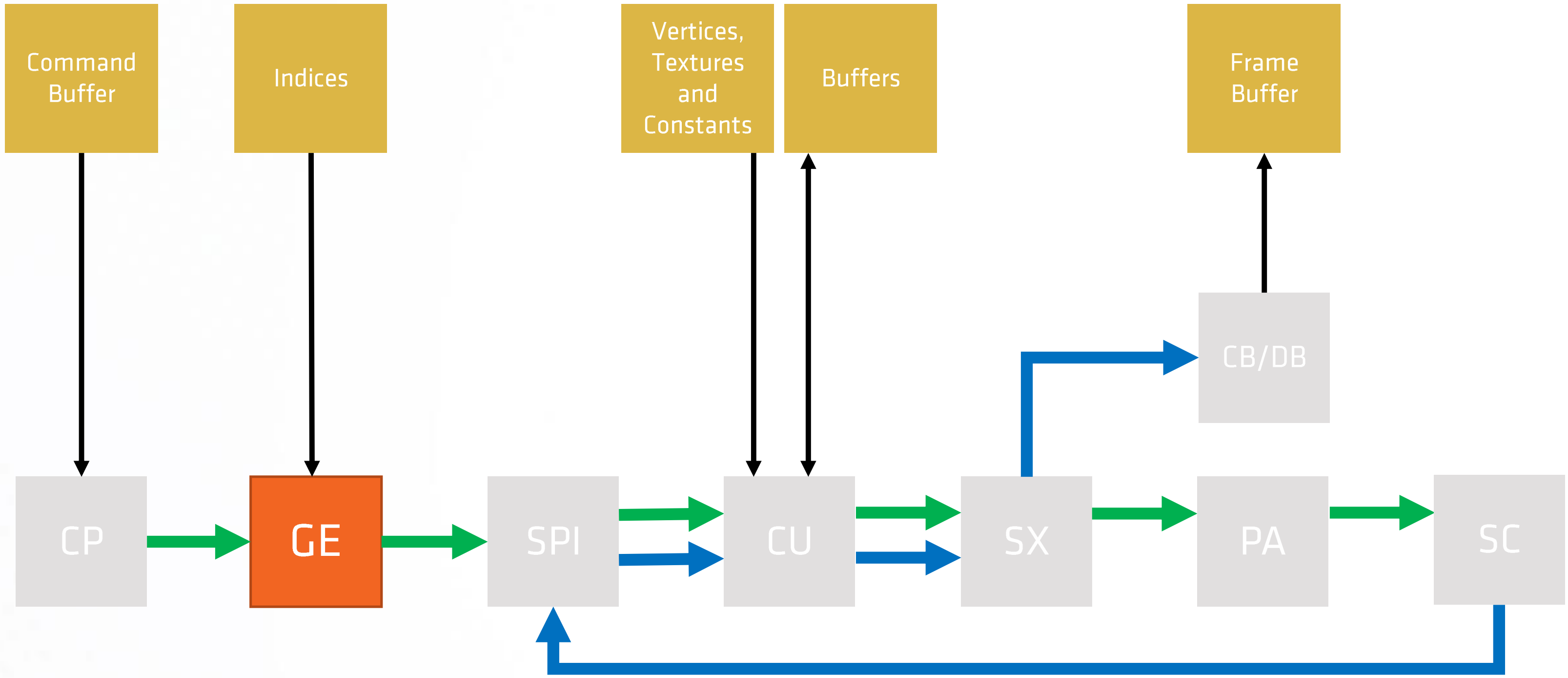
On the GPU – The Big Picture



Command Processor

Sets registers of the other GPU blocks
based on the PM4 packets in the command buffer.

On the GPU – The Big Picture



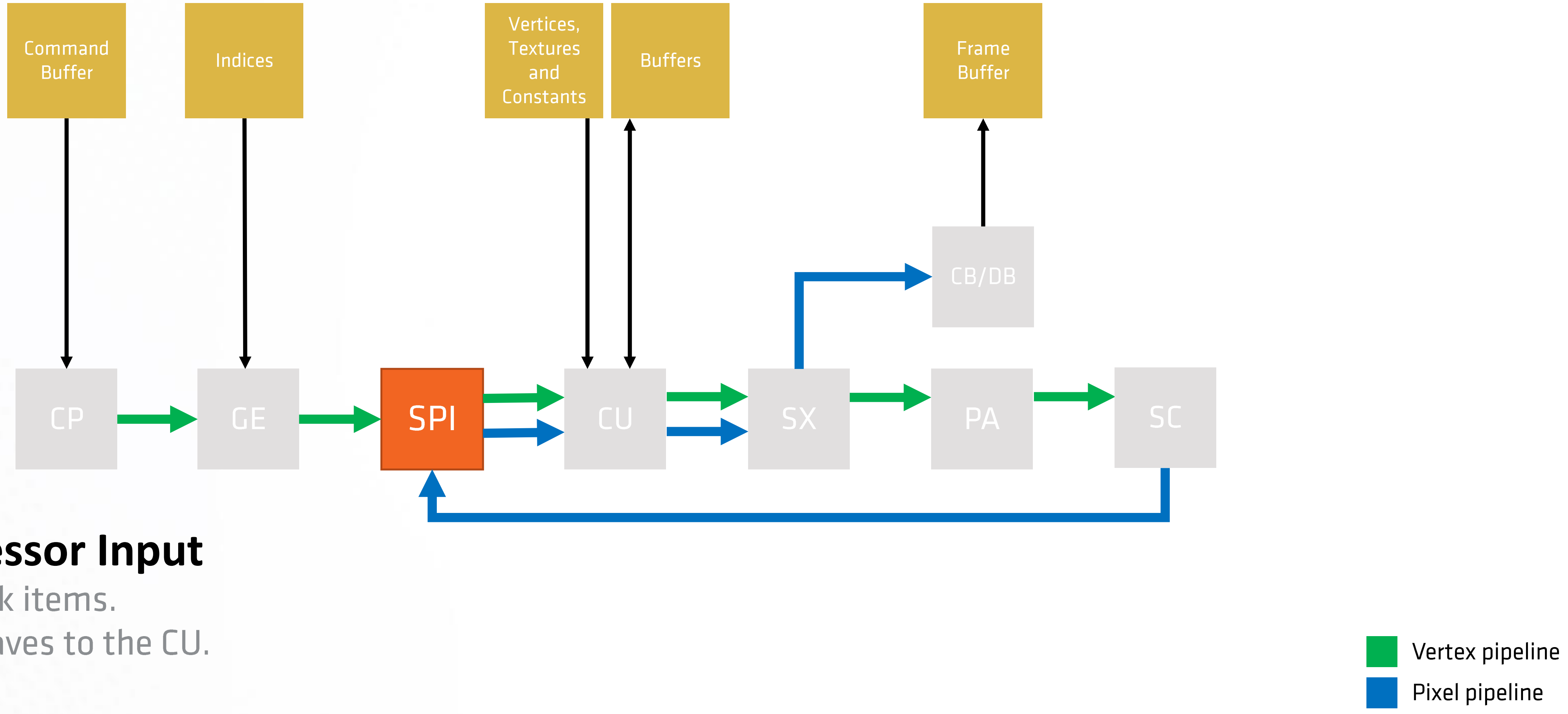
Geometry Engine

Knows about topology / connectivity.
Instructs SPI to generate work.

Vertex pipeline
Pixel pipeline

AMD

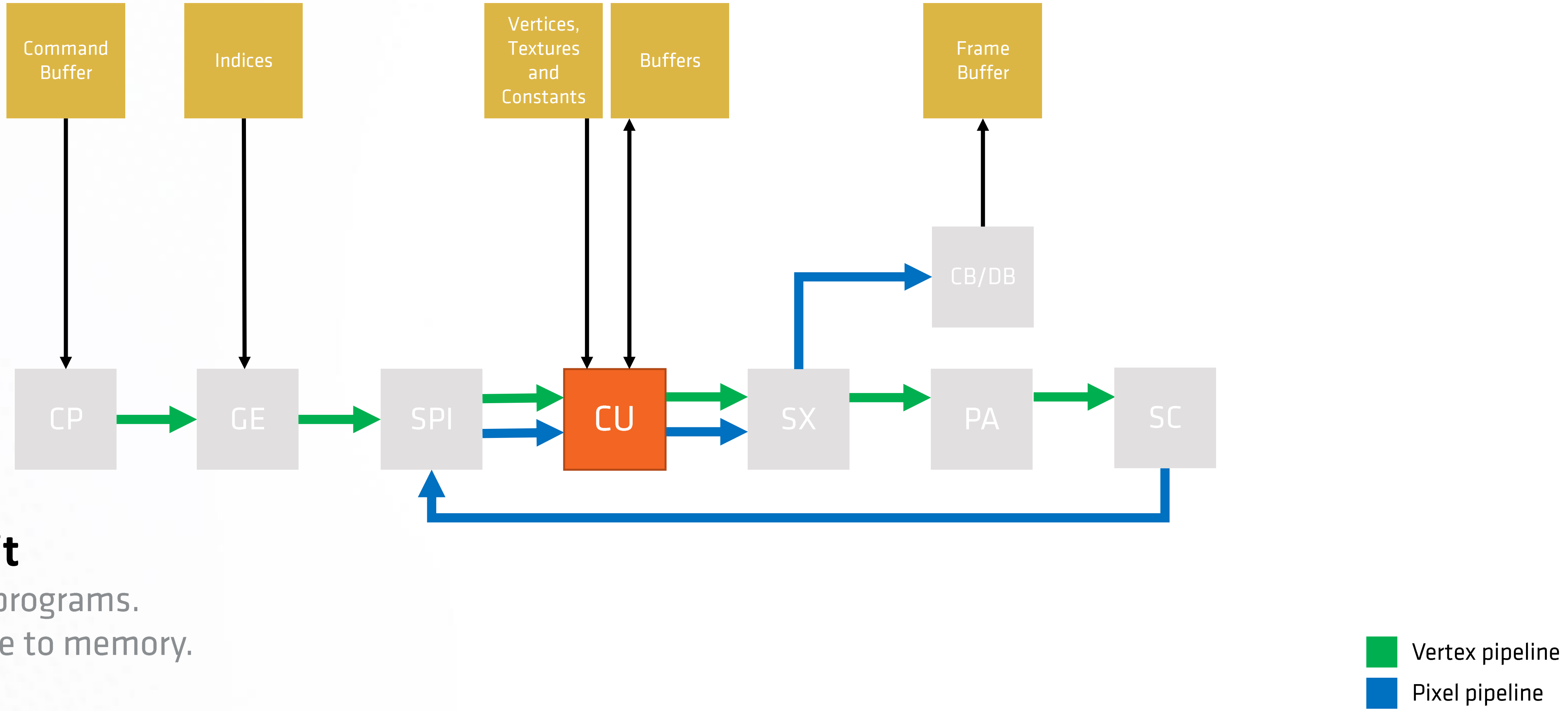
On the GPU – The Big Picture



Shader Processor Input

Accumulates work items.
Sends them in waves to the CU.

On the GPU – The Big Picture

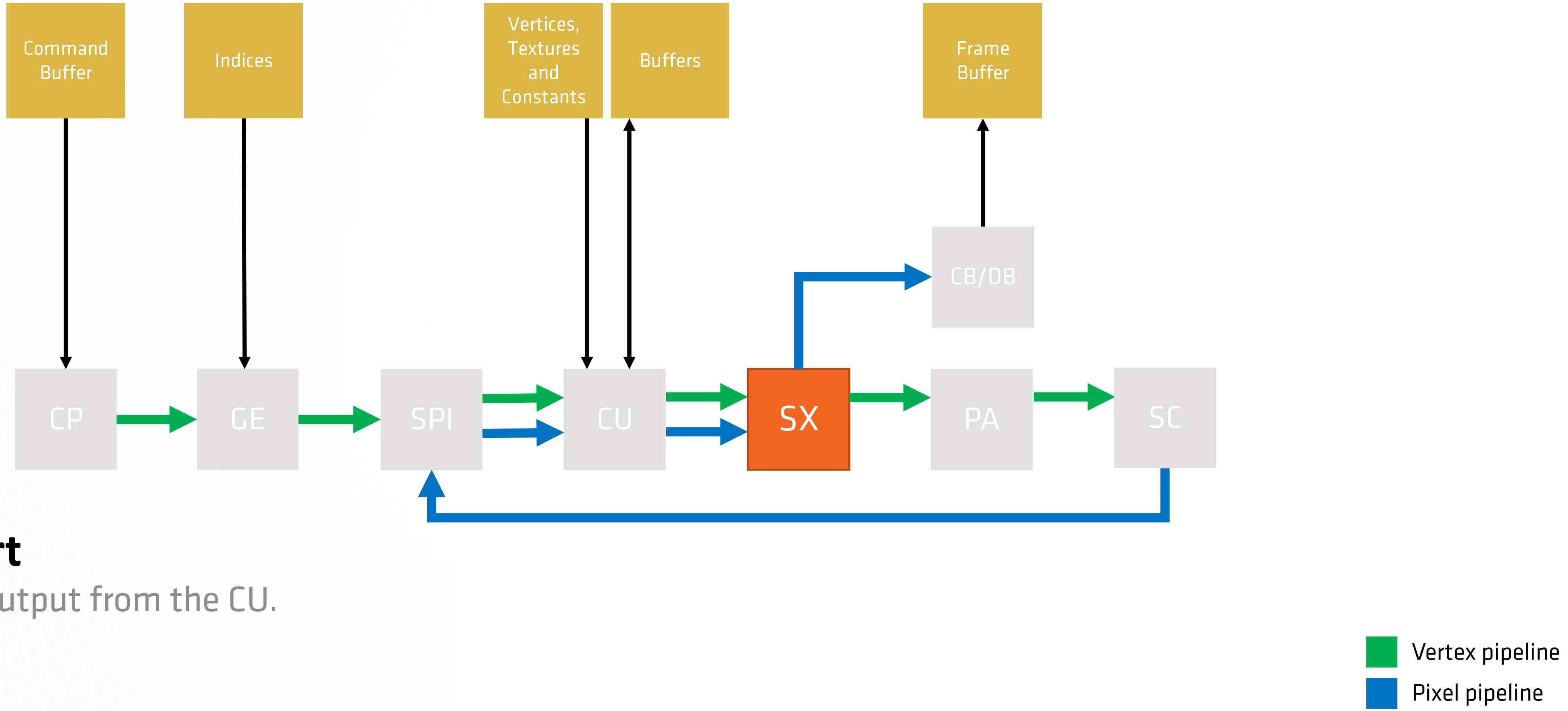


Compute Unit

Executes shader programs.
Can read and write to memory.

AMD

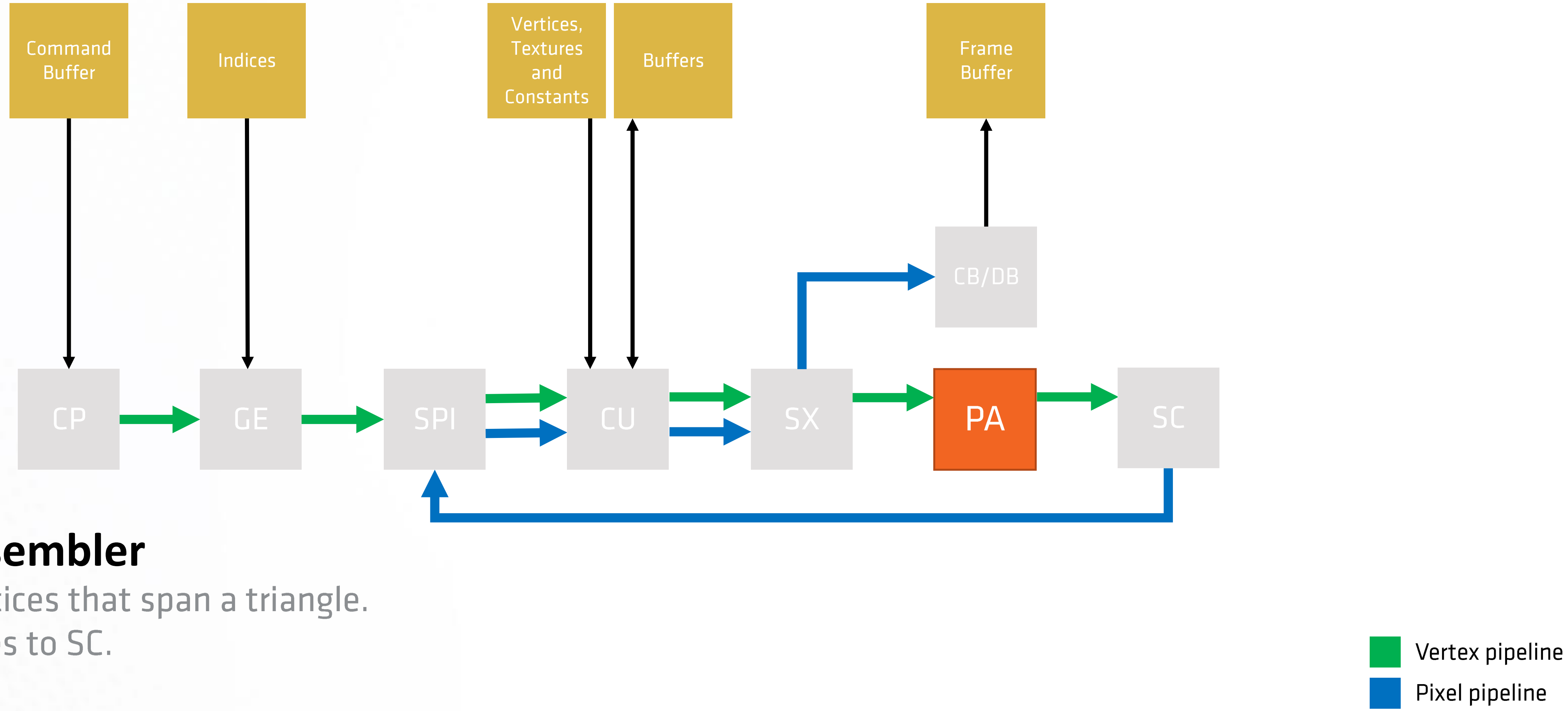
On the GPU – The Big Picture



Shader Export

Handles special output from the CU.

On the GPU – The Big Picture

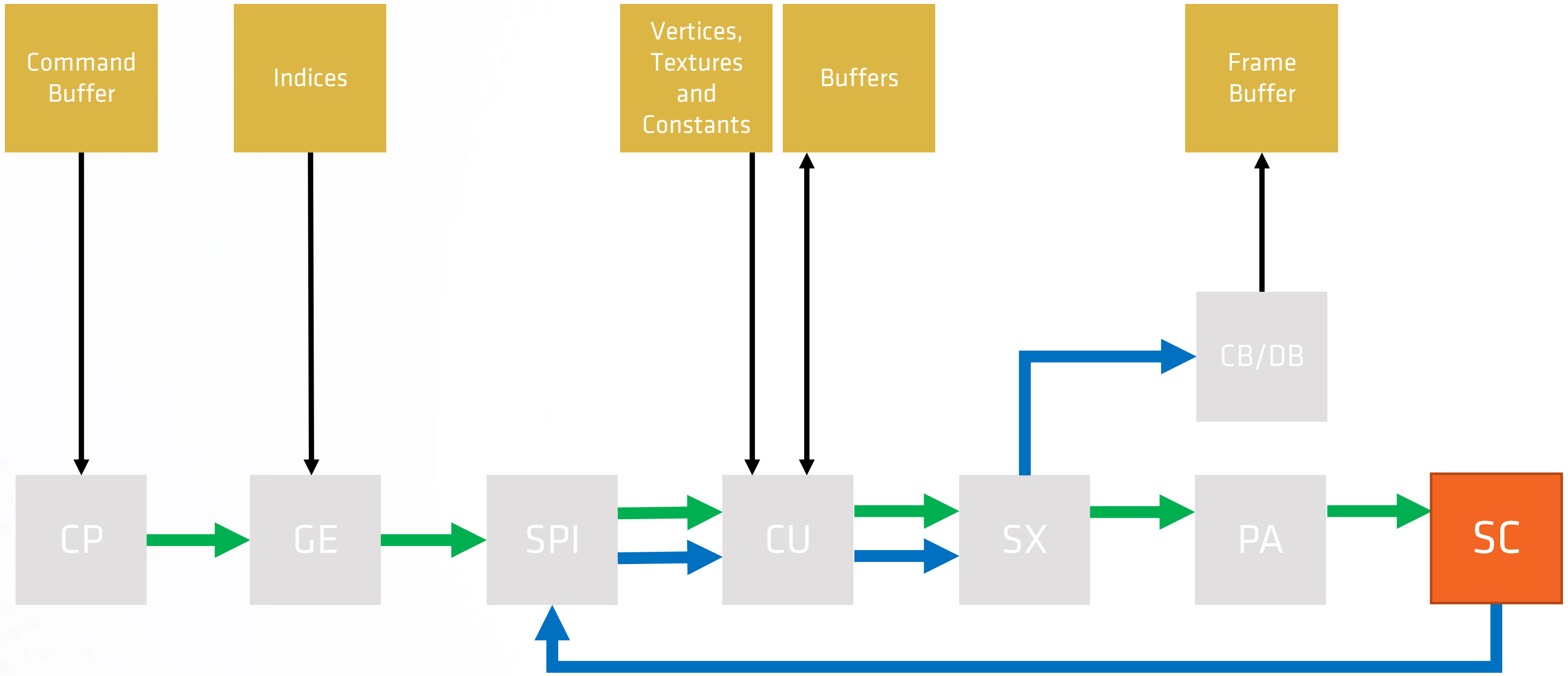


Primitive Assembler

Accumulates vertices that span a triangle.
Forwards triangles to SC.

AMD

On the GPU – The Big Picture

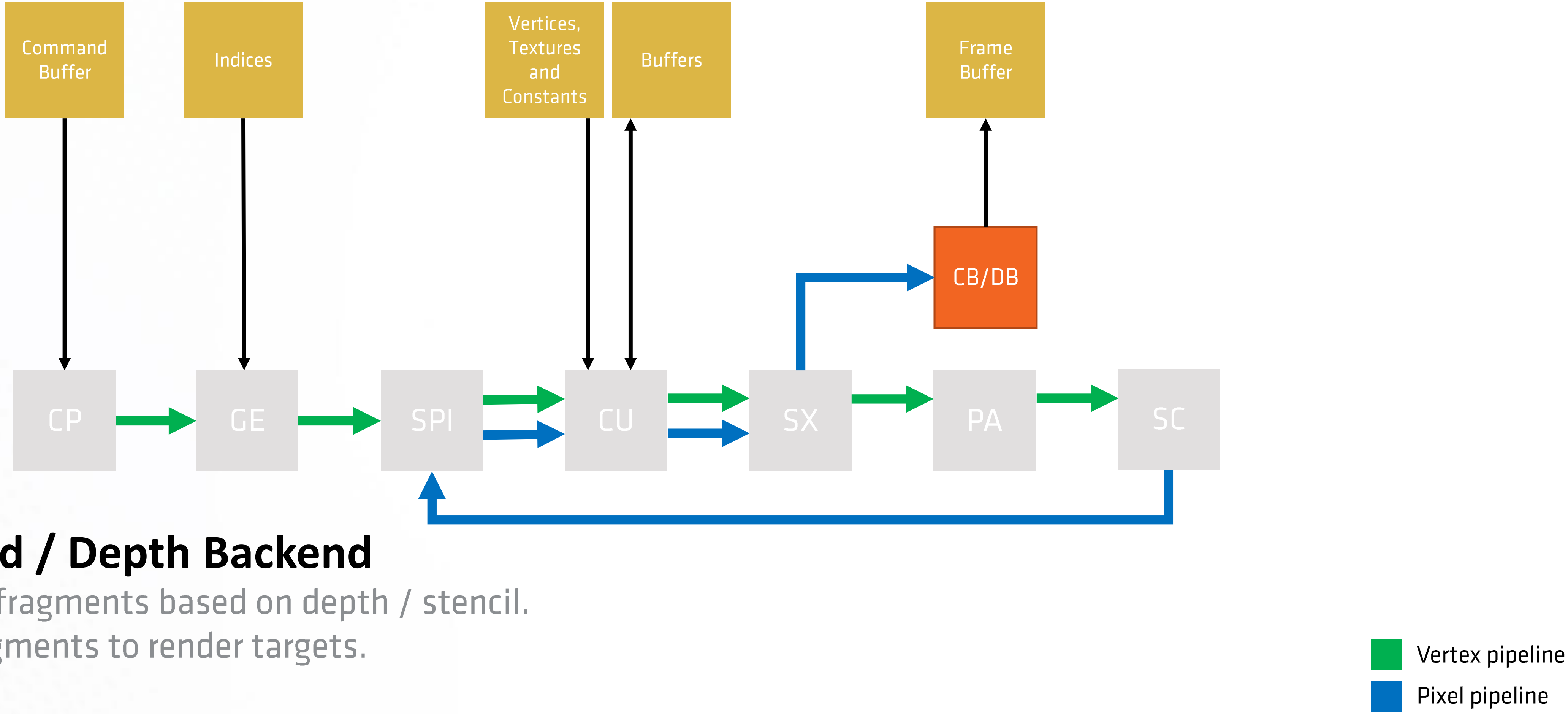


Scan Converter

Determine pixels covered by each triangle.
Forwards them to SPI.

Vertex pipeline
Pixel pipeline

On the GPU – The Big Picture



Color Backend / Depth Backend

Discard occluded fragments based on depth / stencil.
Write colored fragments to render targets.

AMD

On the GPU – Unrolled

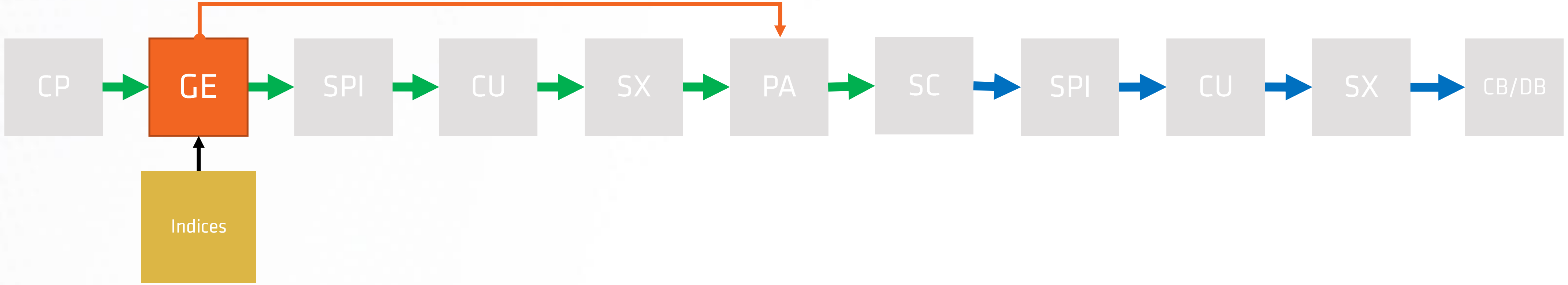
CP → GE → SPI → CU → SX → PA → SC → SPI → CU → SX → CB/DB

Vertex pipeline

Pixel pipeline

AMD

On the GPU – Geometry Engine

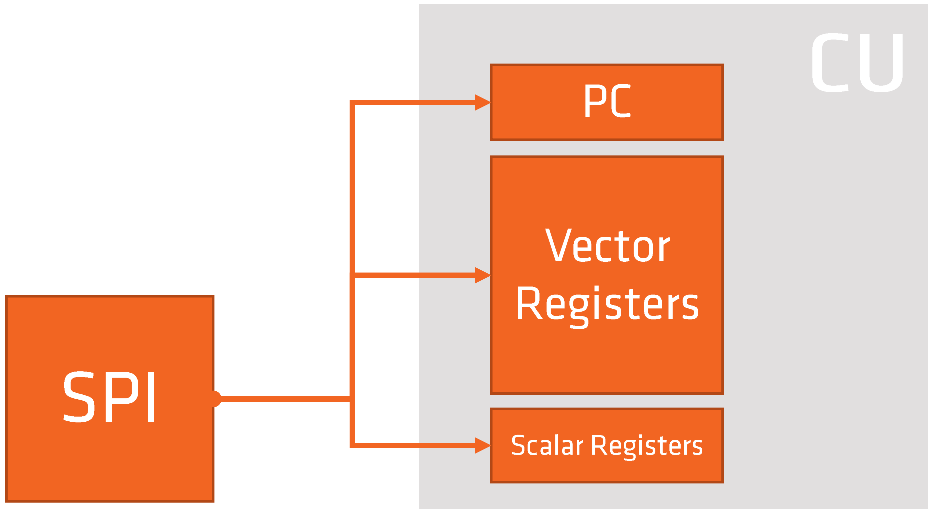


- Retrieves indices from index buffer.
- Knows about the topology.
  I.e. interprets the indices as list of triangles in our case.
- Holds a cache for vertex reuse.
  Index buffer is not just designed as an optimization to store less vertices but is also designed to avoid shading vertices multiple times.
- Reserves queue entries
  to (re)assemble triangle later in the PA.
  Vertex reuse breaks implicit connectivity.
- Forwards index to SPI.

AMD

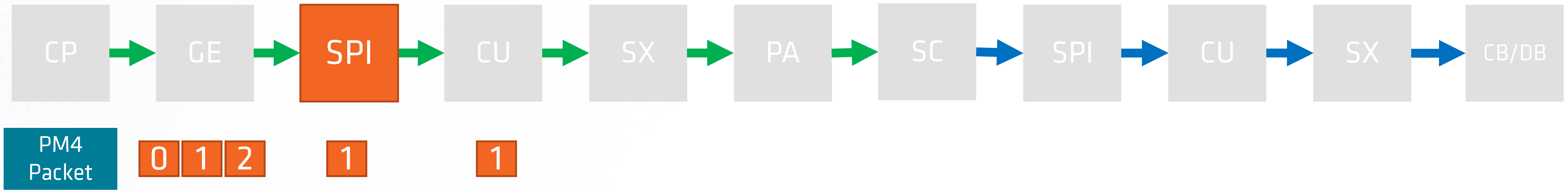
On the GPU – Shader Processor Input (Vertex Pipeline)

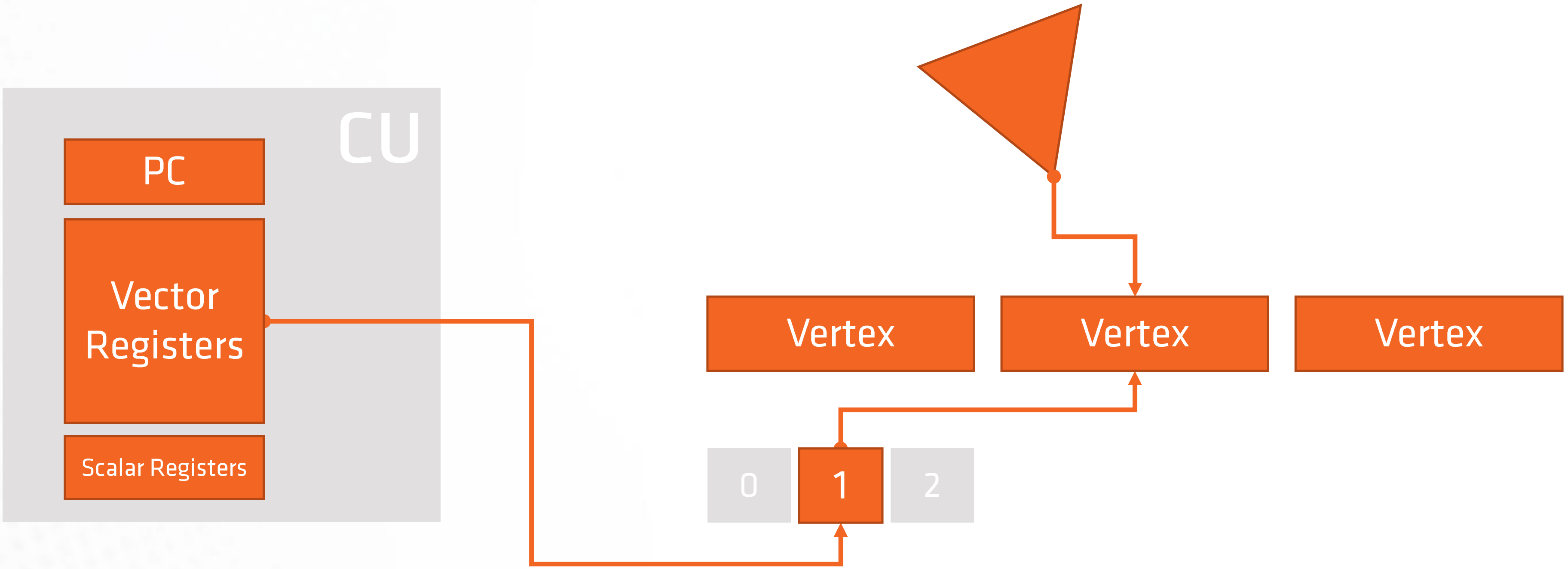
- Waits until there are enough indices before bothering the CU.
- Chooses a CU.
- Configures CU.
  Reserves resources in CU.
  Loads address of vertex shader program into the program counter.
  Initializes scalar and vector registers in CU.
- Kicks off work for CU



AMD

On the GPU – Find the Triangle #2

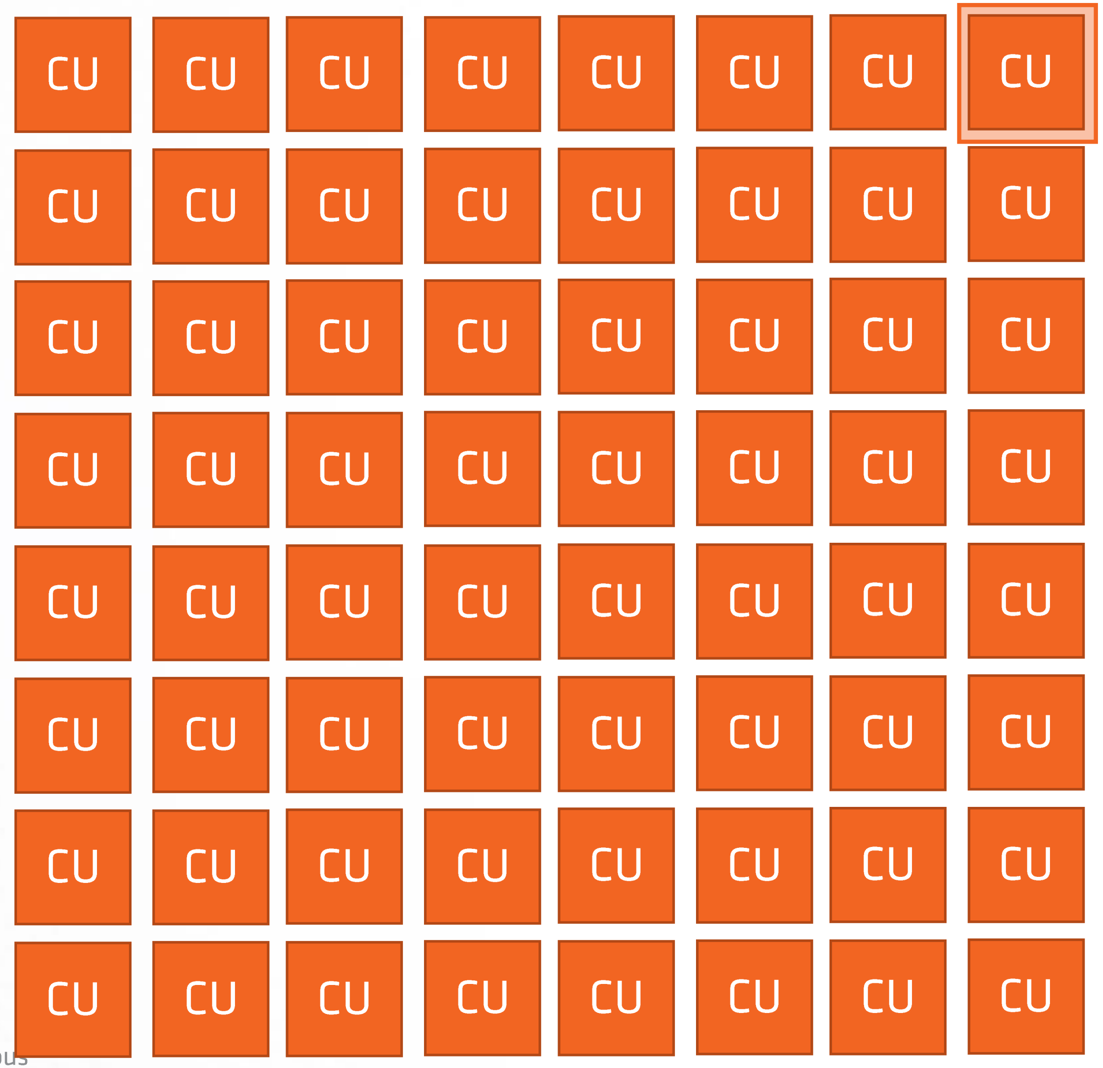


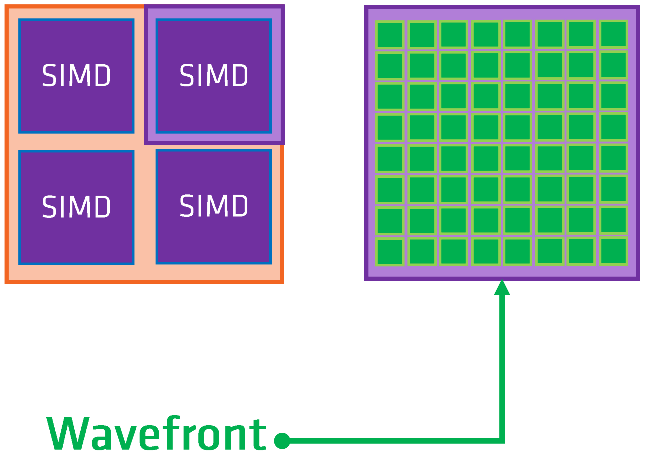


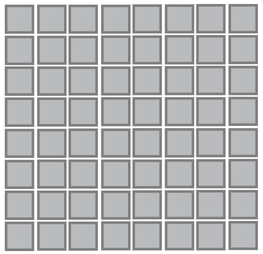
AMD

On the GPU – Compute Unit (Vertex Pipeline)

AMD Radeon™
RX Vega 64
64 CUs
4 SIMDs per CU
64 threads per SIMD



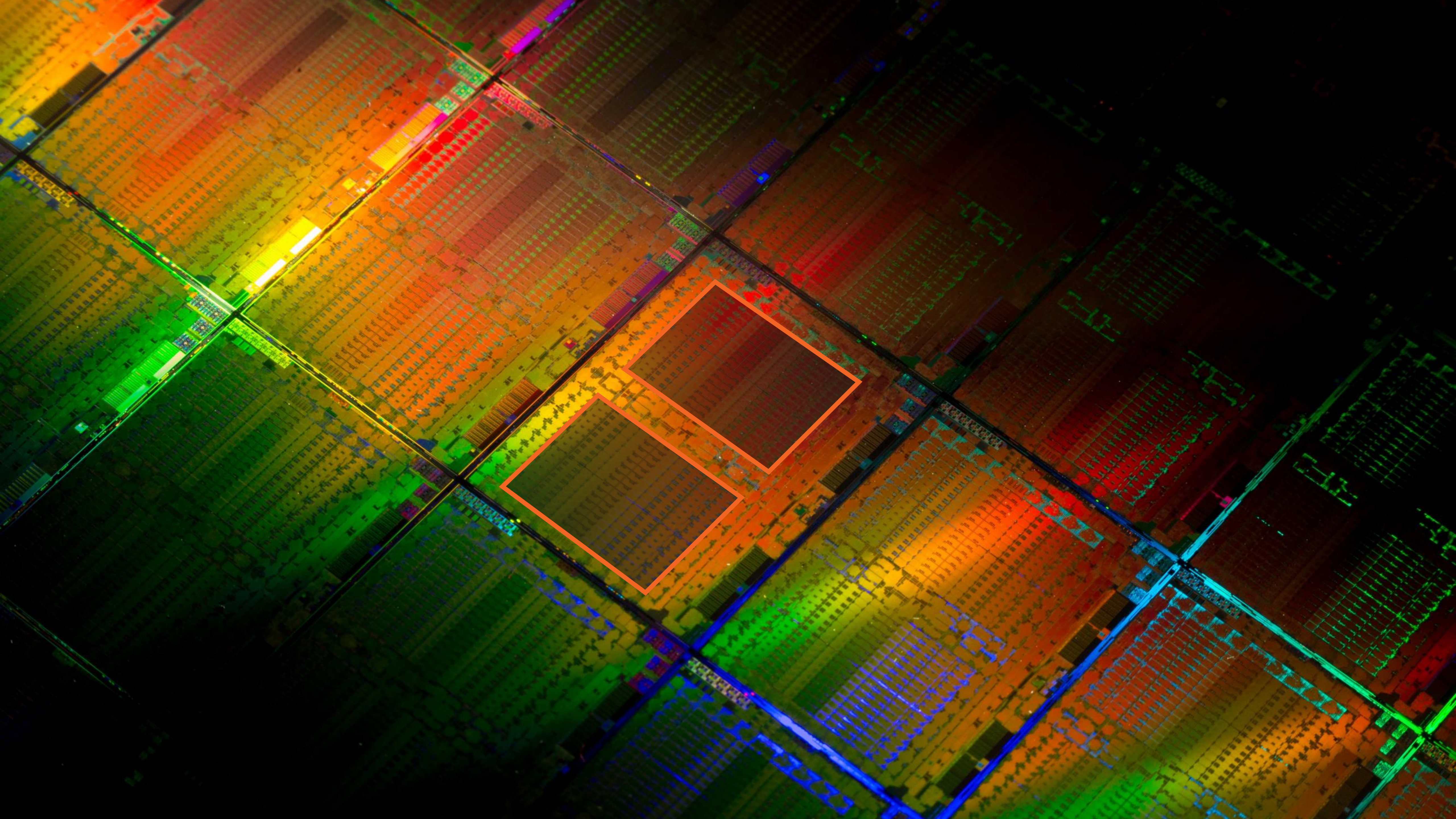




One vector register (VGPR) holds one number per **thread.**

One scalar register (SGPR) holds one number per **wavefront.**

AMD



On the GPU – Compute Unit (Vertex Pipeline)

CP → GE → SPI → **CU** → SX → PA → SC → SPI → CU → SX → CB/DB

EXAMPLE

- Either do an image lookup
- Or use constant red color
- Forward positions to SX
- Forward parameters to SX

```
  s_mov_b64 s[0:1], exec
  v_cmpx_lt_f32 s[2:3], v0, 0.5
  s_cbranch_execz label_0098
  image_sample_lz v[4:7], v[8:10], s[8:15], s[16:19] dmask:0xf
label_0098:
  s_andn2_b64 exec, s[0:1], exec
  v_mov_b32 v4, 1.0
  v_mov_b32 v5, 0
  v_mov_b32 v6, 0
  v_mov_b32 v7, 1.0
  s_mov_b64 exec, s[0:1]
  exp pos0, v0, v1, v2, v3 done
  s_waitcnt vmcnt(0)
  exp param0, v4, v5, v6, v7
  exp param1, v8, v9, off, off
```

- Executes the vertex shader program.
  Originally written by your fellow graphics programmer.
  Usually transforms vertices from object space to clip space.
  Attaches additional vertex attributes.

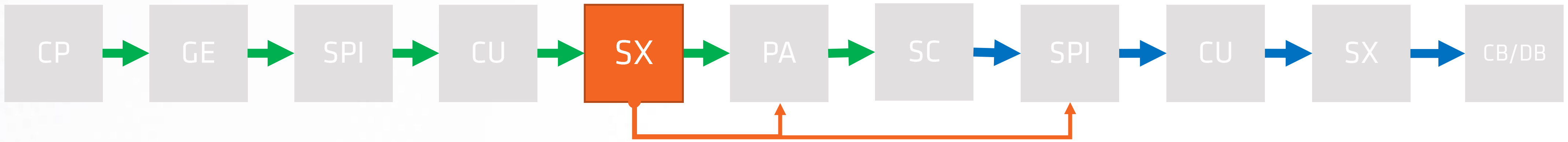
On the GPU – Compute Unit (Vertex Pipeline)

```
  s_mov_b64 s[0:1], exec
  v_cmpx_lt_f32 s[2:3], v0, 0.5
  s_cbranch_execz label_0098
  image_sample_lz v[4:7], v[8:10], s[8:15], s[16:19] dmask:0xf
label_0098:
  s_andn2_b64 exec, s[0:1], exec
  v_mov_b32 v4, 1.0
  v_mov_b32 v5, 0
  v_mov_b32 v6, 0
  v_mov_b32 v7, 1.0
  s_mov_b64 exec, s[0:1]
  exp pos0, v0, v1, v2, v3 done
  s_waitcnt vmcnt(0)
  exp param0, v4, v5, v6, v7
  exp param1, v8, v9, off, off
```

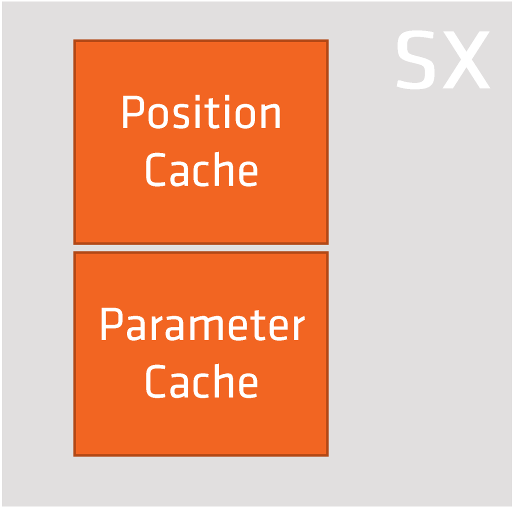
- Execution happens in lock-step.
  Each thread is designed to work on one vertex.
  A wavefront can only work on one instruction at a time.
  Inactive threads are masked out.

On the GPU – Compute Unit (Vertex Pipeline)

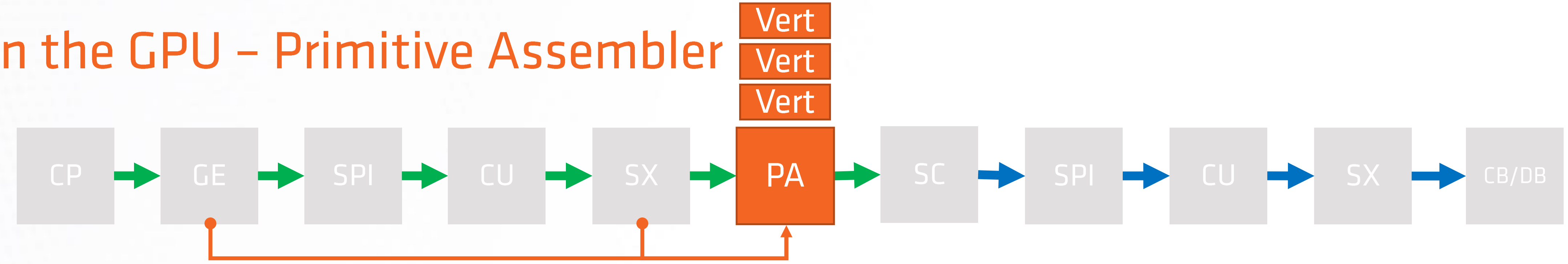
On the GPU – Shader Export (Vertex Pipeline)



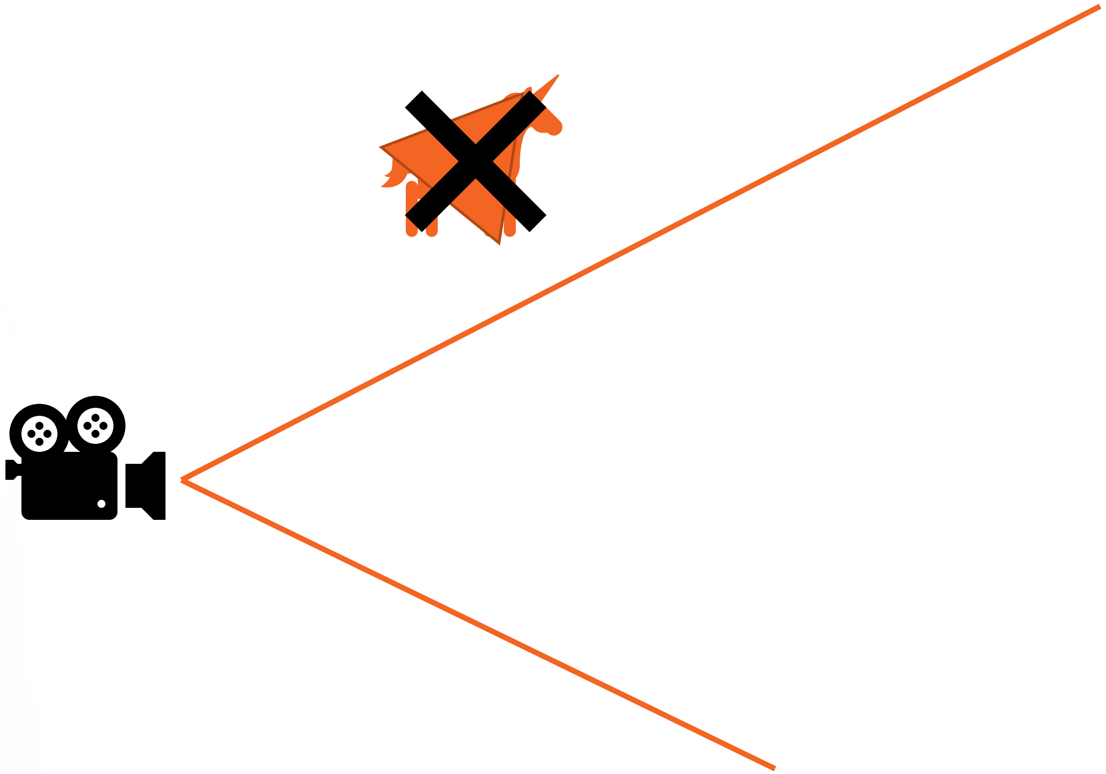
- Gets transformed vertices from CU.
- Decides if we forward data to CB/DB or PA.
  When on the vertex pipeline the SX forwards to PA.
- Caches positions for later use in PA.
  Overarching goal is to not shade the same vertices twice.
- Caches parameters for later use in SPI.
  Adding more parameters to your transformed vertices means you can keep fewer vertices around.



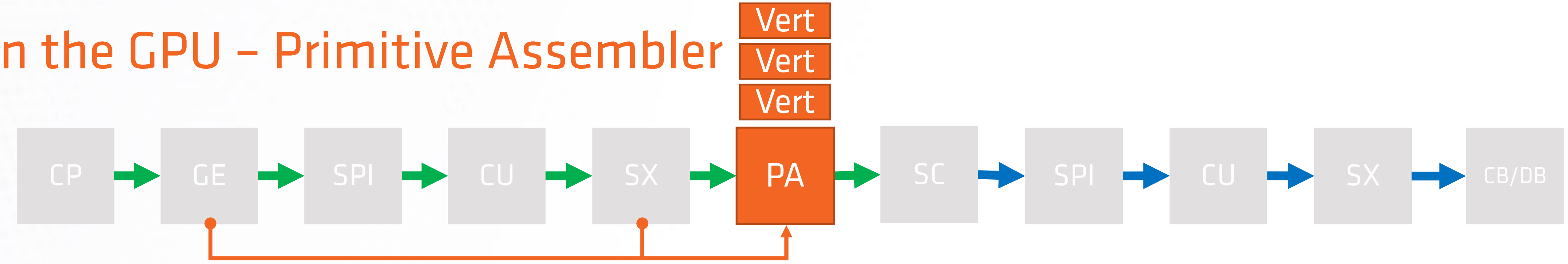
On the GPU – Primitive Assembler



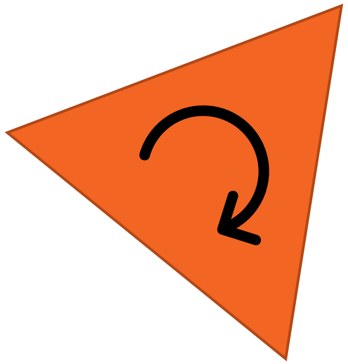
- Uses the connectivity info it got from GE earlier.
- Waits for 3 connected vertex positions to appear.
  Here is our triangle again!
- Transforms vertices to screen space.
- View Frustum Culling

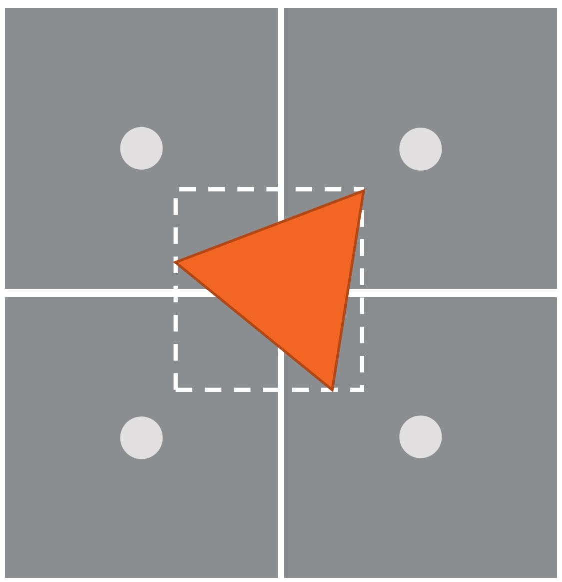


On the GPU – Primitive Assembler



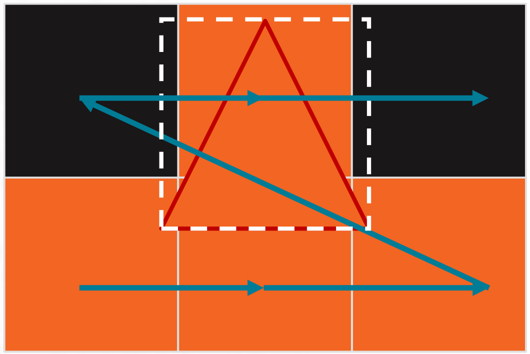
- Uses the connectivity info it got from GE earlier.
- Waits for 3 connected vertex positions to appear.
  Here is our triangle again!
- Transforms vertices to screen space.
- View Frustum Culling
- Small Primitive Filter
- Zero Area Culling
- Backface Culling
- Forwards triangle with barycentric gradients and bounding box to SC.



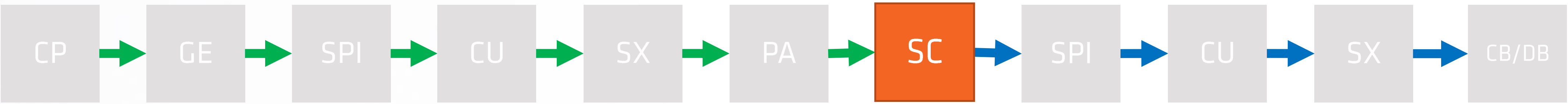


On the GPU – Scan Converter

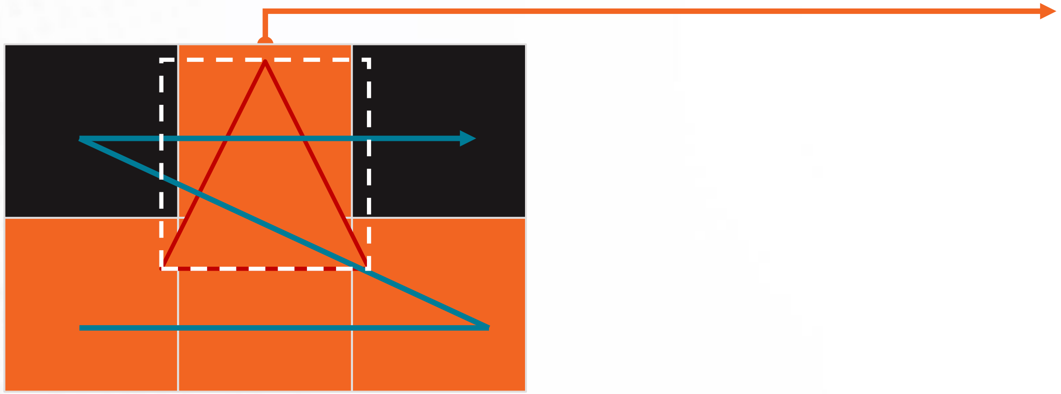
- Determine all pixels that overlap the triangle.
  Commonly known as "Rasterization".
- Scan Conversion only on a coarse level.

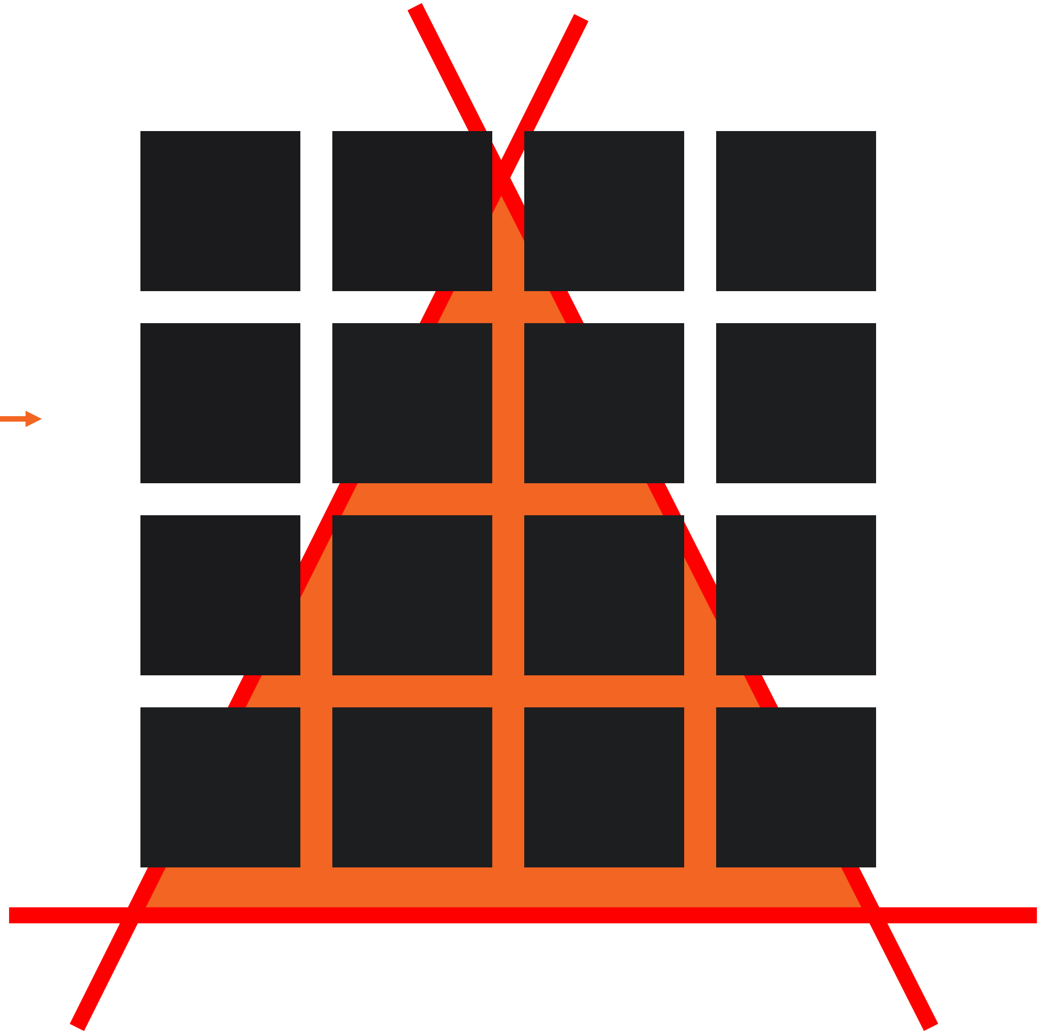


On the GPU – Scan Converter



- Determine all pixels that overlap the triangle. Commonly known as "Rasterization".
- Scan Conversion only on a coarse level.
- On a fine level (4x4 pixels) test against the triangle edges.

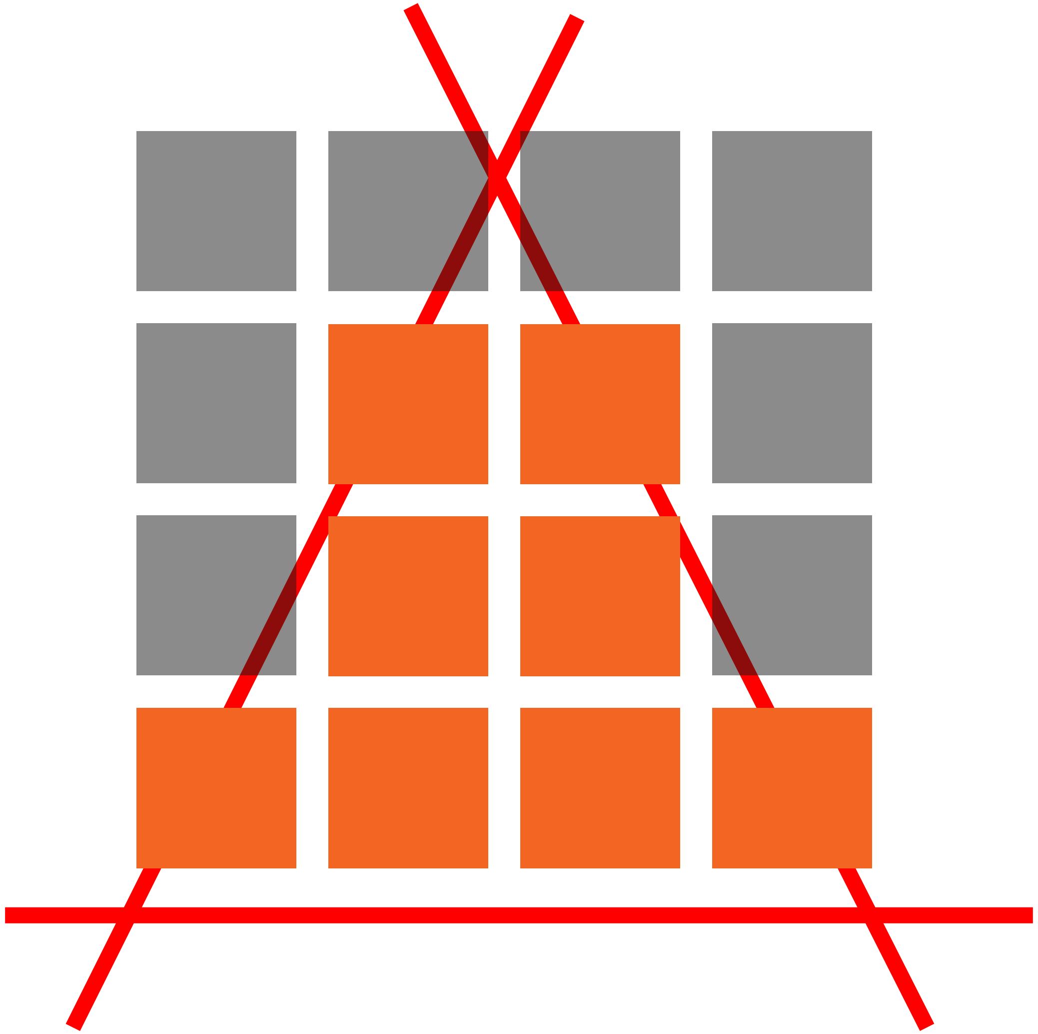




AMD

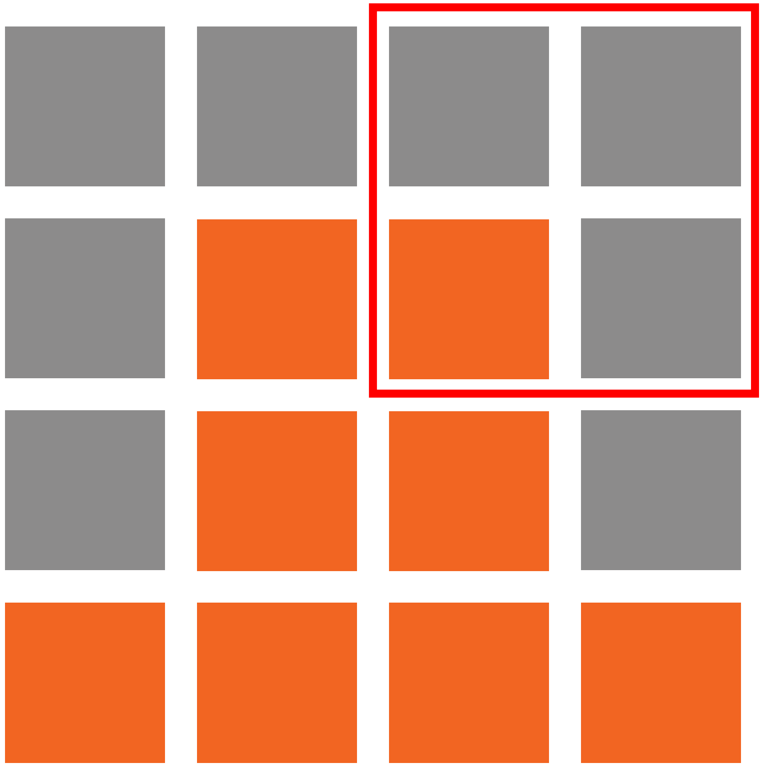
On the GPU – Scan Converter

- Determine all pixels that overlap the triangle.
  Commonly known as "Rasterization".
- Scan Conversion only on a coarse level.
- On a fine level (4x4 pixels) test against the triangle edges.



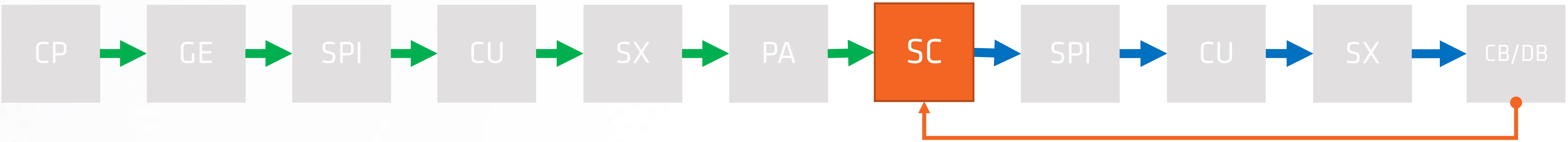
On the GPU – Scan Converter

- Determine all pixels that overlap the triangle.
  Commonly known as "Rasterization".
- Scan Conversion only on a coarse level.
- On a fine level (4x4 pixels) test against the triangle edges.
- Forwards **quads** to the SPI.
  Allows to access derivatives in the PS.

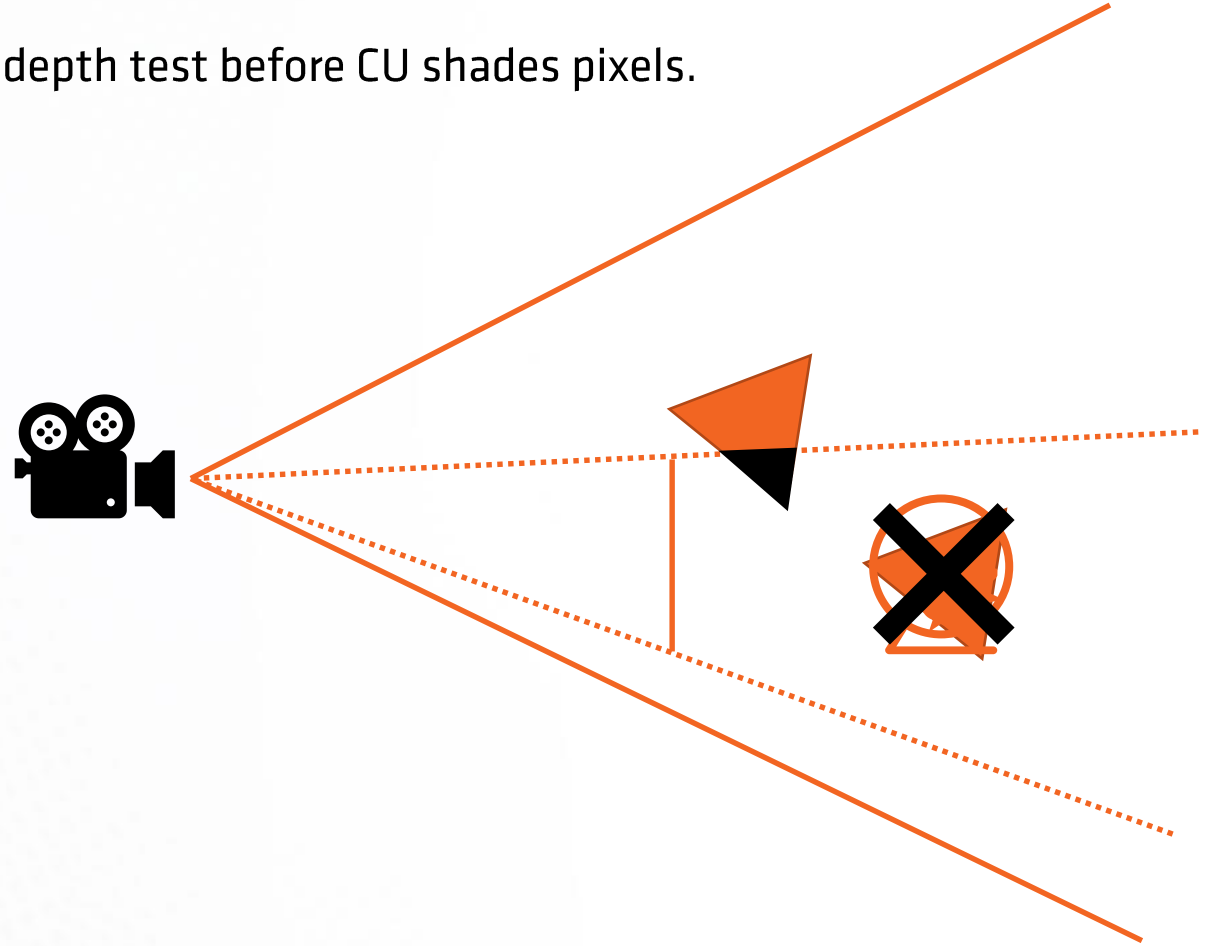


AMD

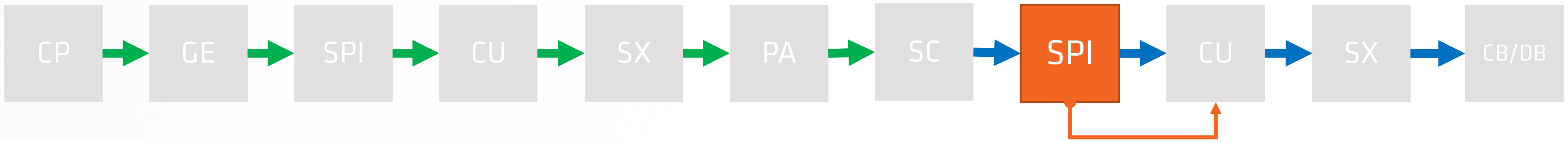
On the GPU – Scan Converter



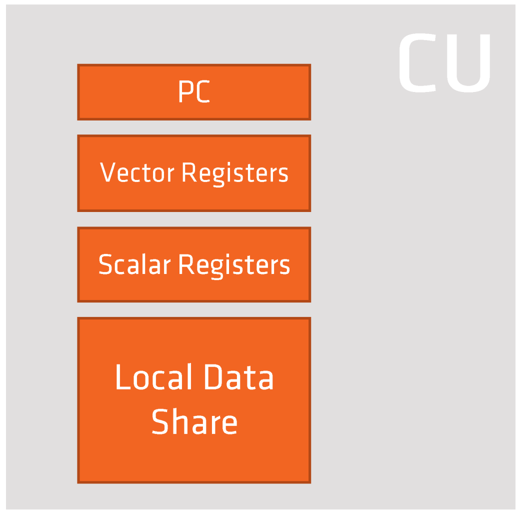
- Can do an early depth test before CU shades pixels.



On the GPU – Shader Processor Input (Pixel Pipeline)

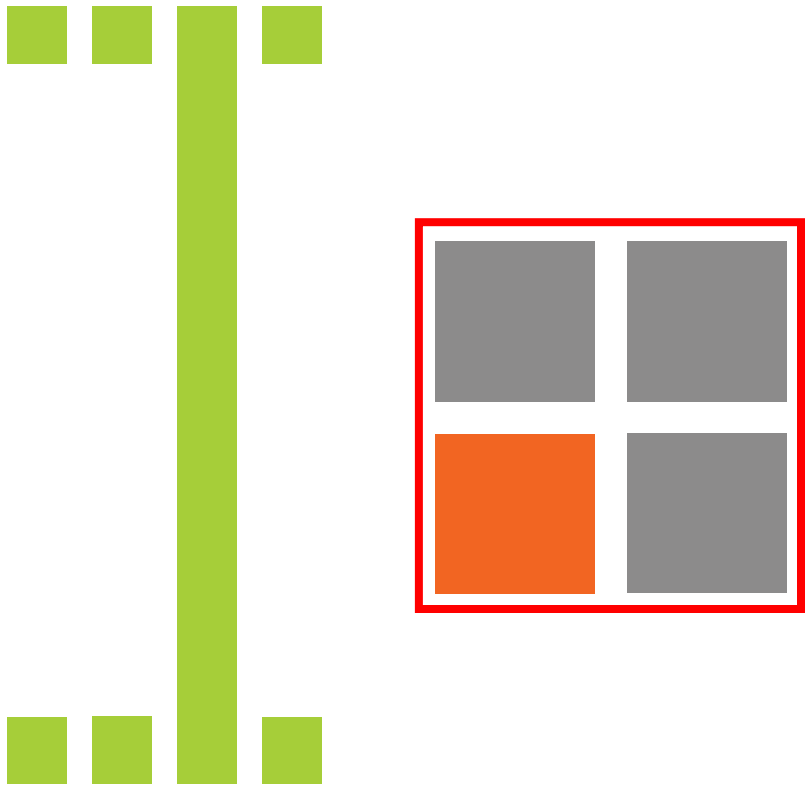


- Get enough quads from SC.
- Same as before: Choose CU, set PC & setup registers.
  Each thread with its own barycentric coordinates from SC.
- Additionally put vertex parameters into local data share.
  Allows to easily share data with other threads.



On the GPU – Compute Unit (Pixel Pipeline)

- This time each thread works on one fragment.
- With small triangles a lot of work is masked out!
- Export finished fragments to SX.

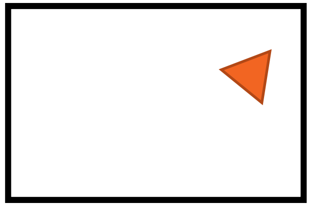


On the GPU – Shader Export (Pixel Pipeline)

- On the pixel pipeline so SX forwards fragments to CB/DB instead of PA.

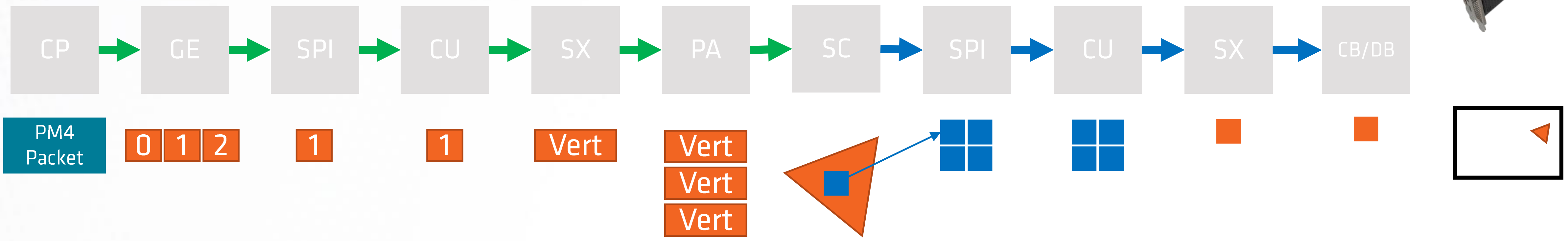
On the GPU – Color Backend / Depth Backend

- Potentially has to perform late depth test.
- Fragment Reordering
- Color Blending
- Writes the fragment colors to bound render targets.
- Does lots of other stuff:
  MSAA resolve
  Compression
  …



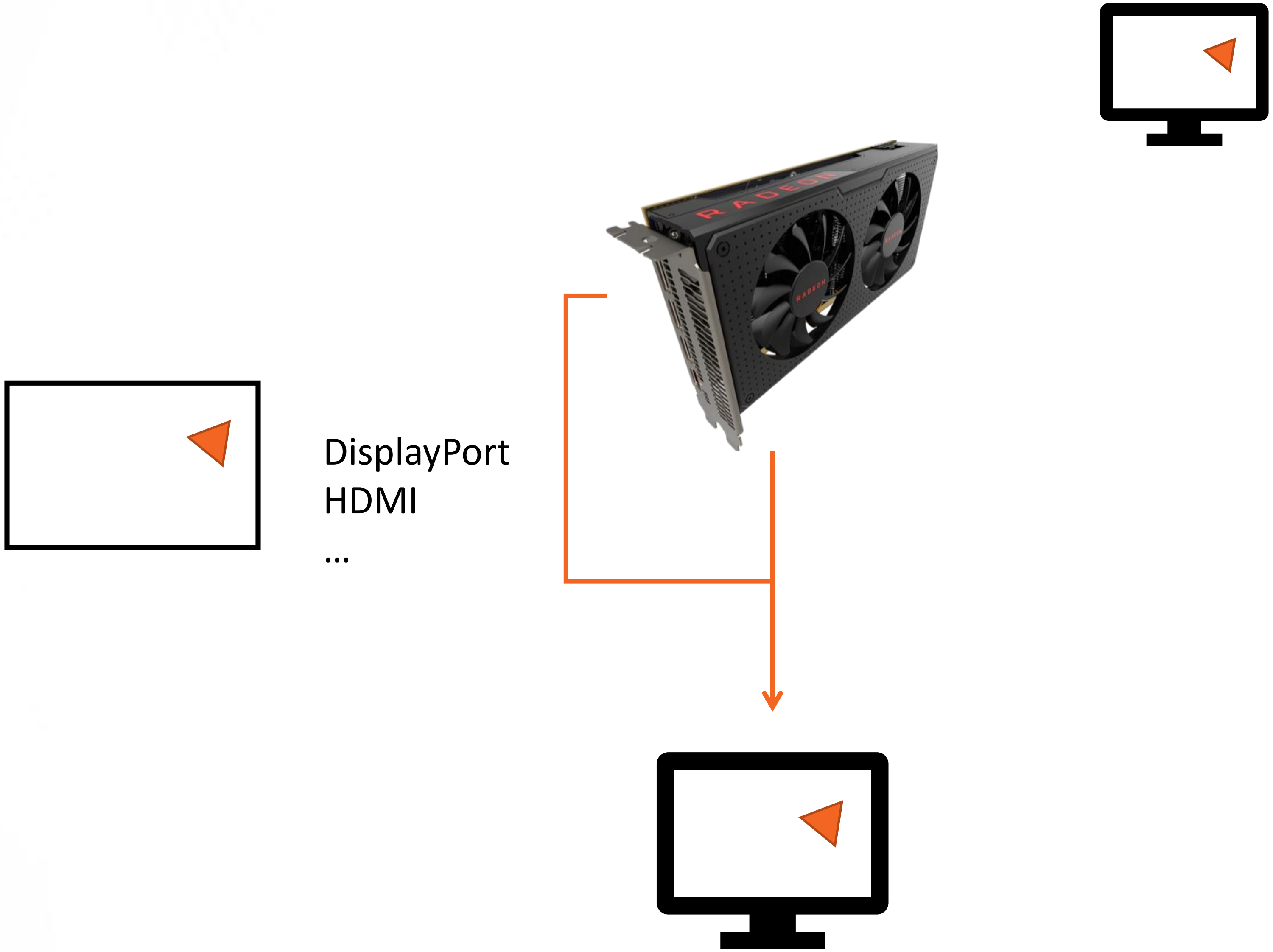
AMD

On the GPU – Looking Back



Presentation

- Wait until the render target is done and send it over to the screen.



Further Reading

- AMD Linux Open Source Driver
  https://github.com/GPUOpen-Drivers/AMDVLK
- A trip through the Graphics Pipeline 2011 – Fabian Giesen
  https://fgiesen.wordpress.com/2011/07/09/a-trip-through-the-graphics-pipeline-2011-index/
- Vega Instruction Set Architecture
  https://gpuopen.com/amd-vega-instruction-set-architecture-documentation/
- The AMD GCN Architecture – Layla Mah
  https://de.slideshare.net/DevCentralAMD/gs4106-the-amd-gcn-architecture-a-crash-course-by-layla-mah
- Radeon Southern Islands Acceleration (PM4)
  https://developer.amd.com/wordpress/media/2013/10/si_programming_guide_v2.pdf
- Optimizing the Graphics Pipeline with Compute – Graham Wihlidal
  https://frostbite-wp-prd.s3.amazonaws.com/wp-content/uploads/2016/03/29204330/GDC_2016_Compute.pdf

AMD

Thanks!

dominik.baumeister@amd.com

https://gpuopen.com/

AMD

DISCLAIMER AND ATTRIBUTIONS