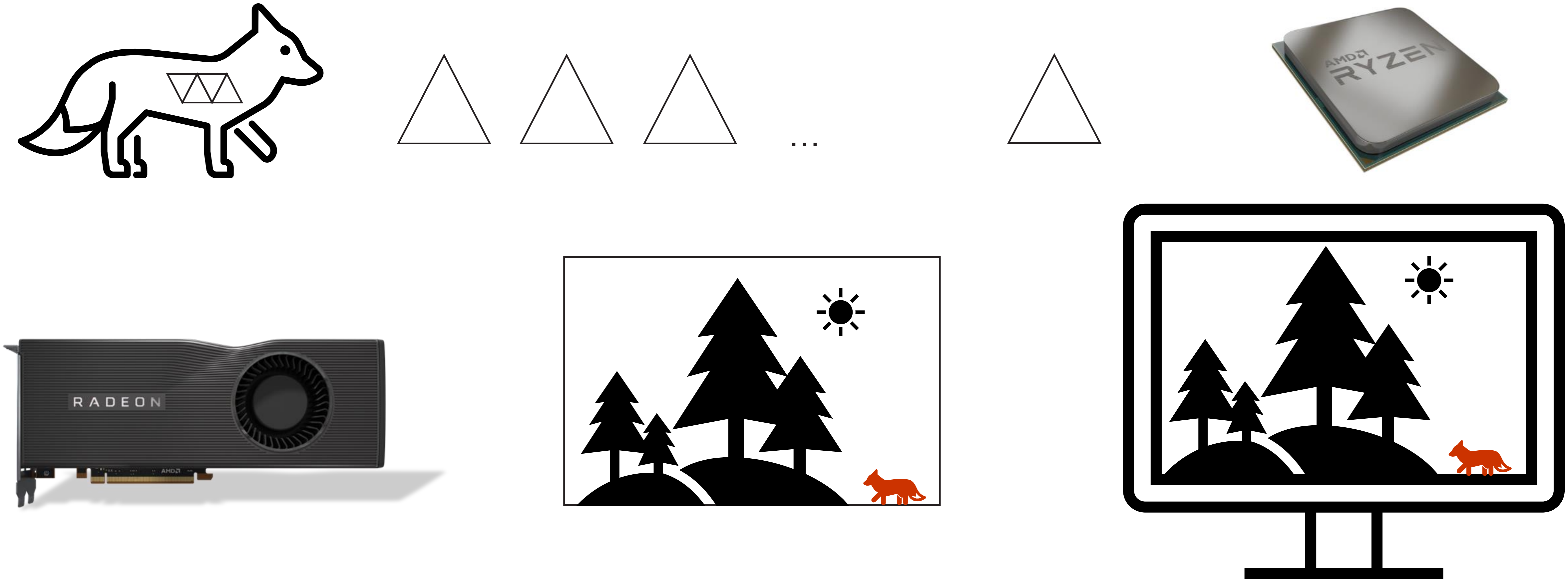
ALL THE PIPELINES – JOURNEY THROUGH THE GPU

LOU KRAMER,
DEVELOPER TECHNOLOGY ENGINEER, AMD

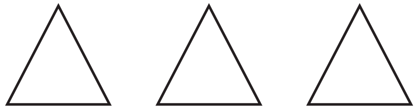
OVERVIEW

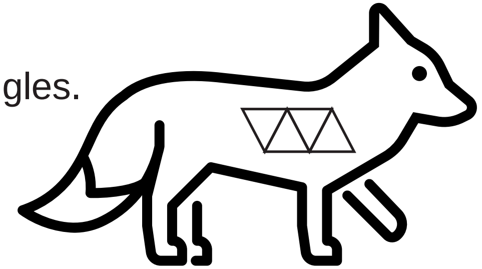


CONTENT CREATION

Some 3d model created via your software of choice (e.g., Blender - www.blender.org).

This model is represented by a bunch of triangles.

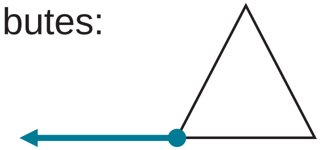


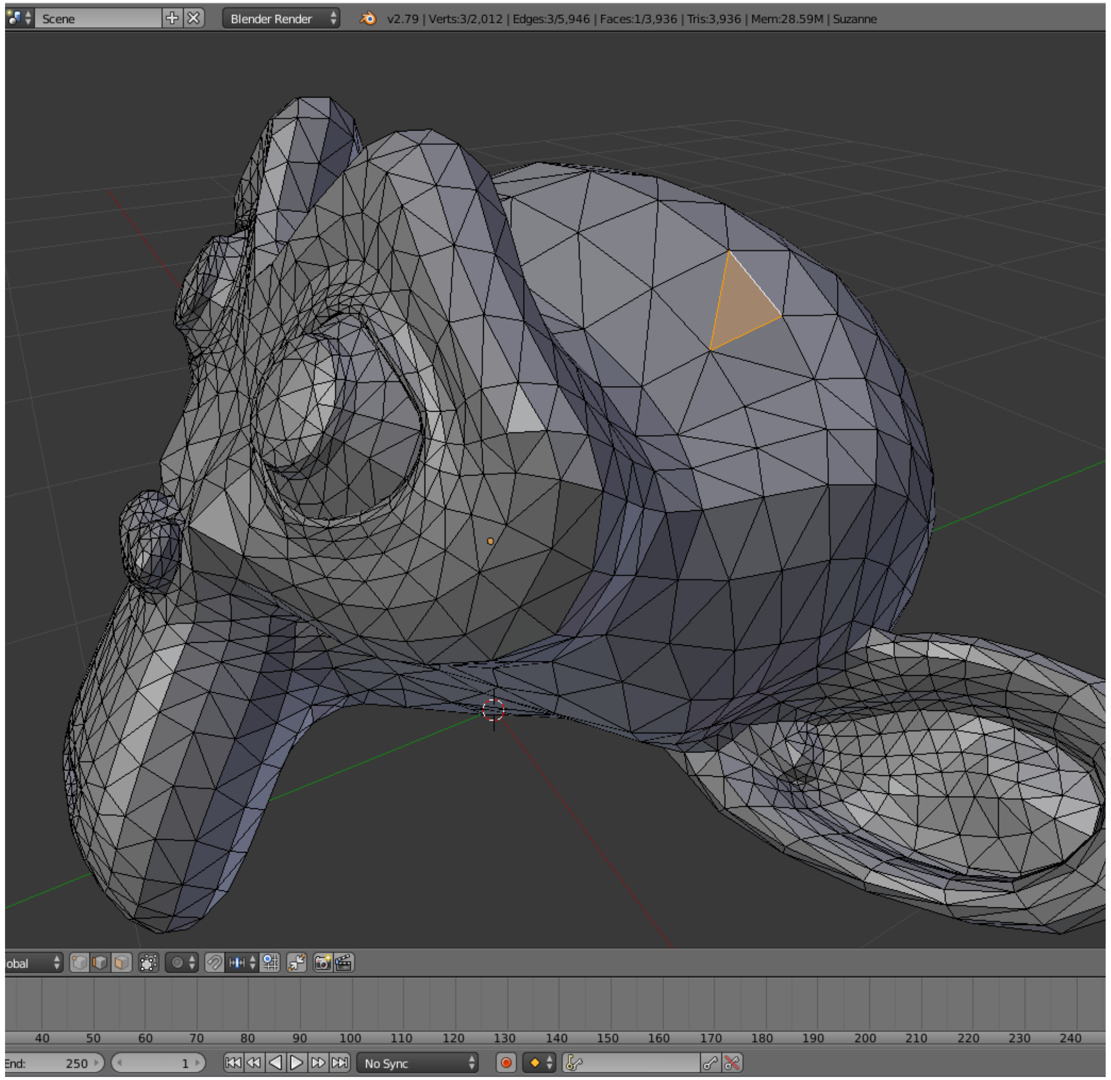


Each triangle is defined by 3 vertices.

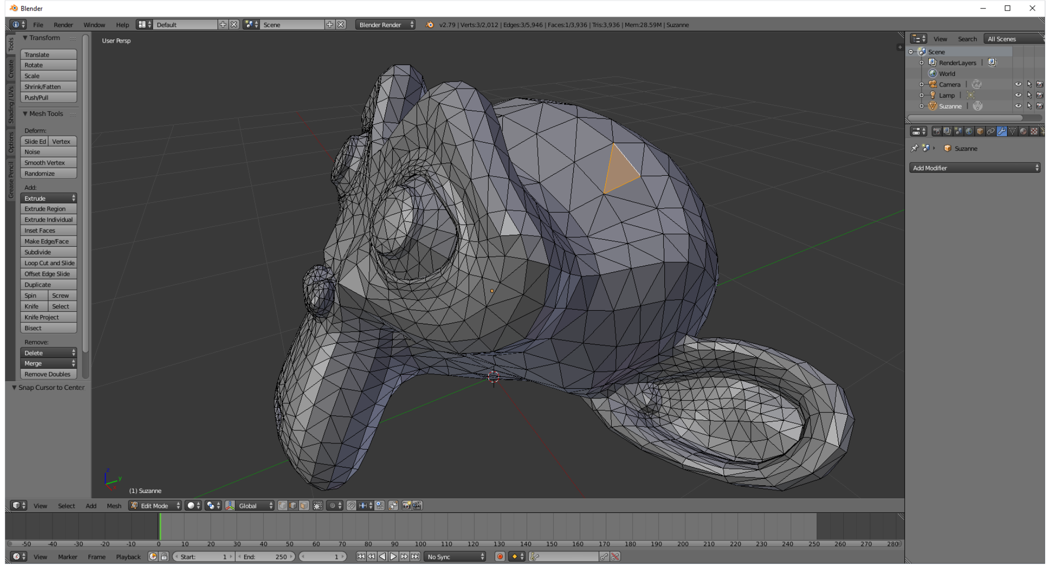
Vertices can have a number of attributes:

- Position
- Normal Vector
- Texture coordinate
- …





CONTENT CREATION

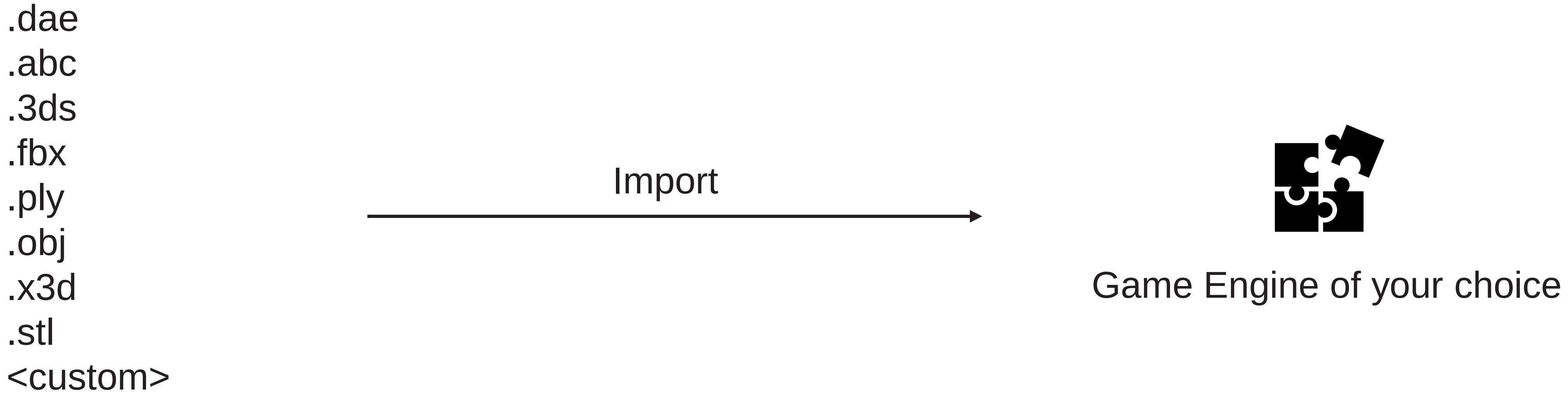


Positions
Normal Vectors
Texture Coordinates
Connectivity Information
…

Export

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

CONTENT CREATION



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

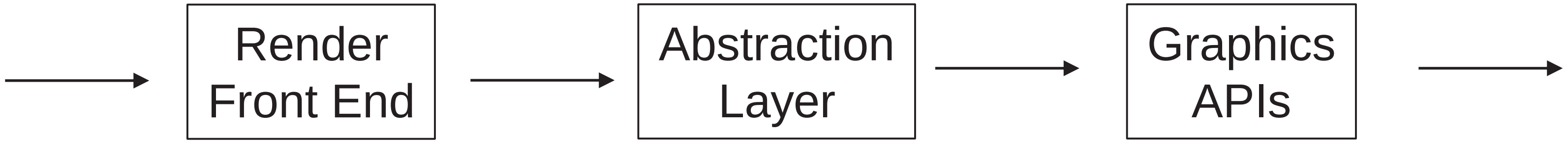
Import

Game Engine of your choice

RENDERING – PREPARATION ON THE CPU

Geometry

Engine Specific format



Render Front End

Abstraction Layer

Graphics APIs

Mesh Creation:
- Vertex Buffers.
- Index Buffers.
- Textures.
- …

Visibility Testing
- The less work the GPU needs to do the better.

…

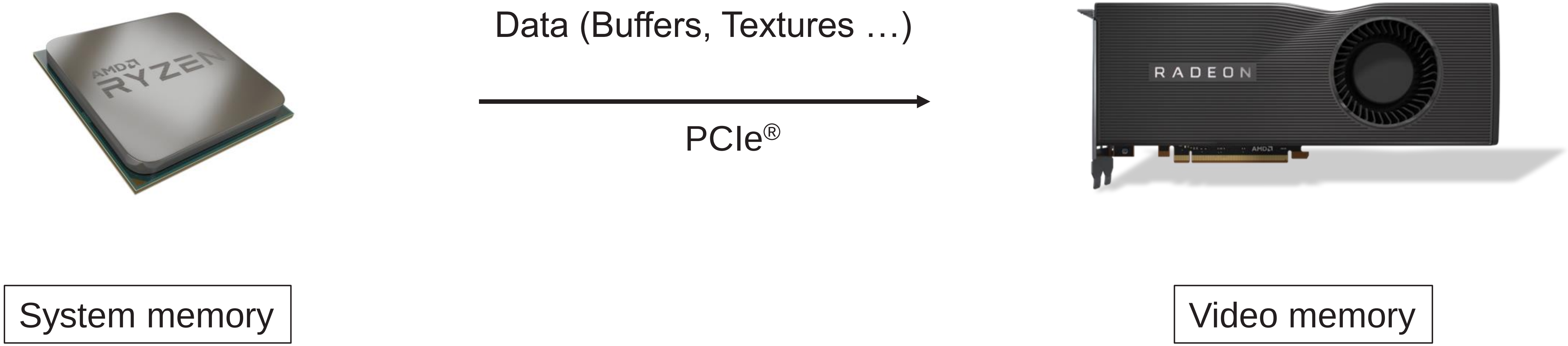
Buffers in
System Memory (CPU)

`MyDraw`
`MyDispatch`
…

Vulkan®
(`vkCmdDrawIndexed`,`vkCmdDispatch`, …)
D3D12
(`DrawIndexedInstanced`,`Dispatch`, …)
D3D11
…

List of Commands

RENDER FRONT END



Data (Buffers, Textures …)

PCIe®

System memory

Video memory

GPU COMMANDS

List of Commands

```
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer
vkCmdDrawIndexed
…
```

- Send a batch of commands to the GPU so the GPU is busy for quite a while.
- Every command list submission takes some time!

GPU COMMANDS

List of Commands

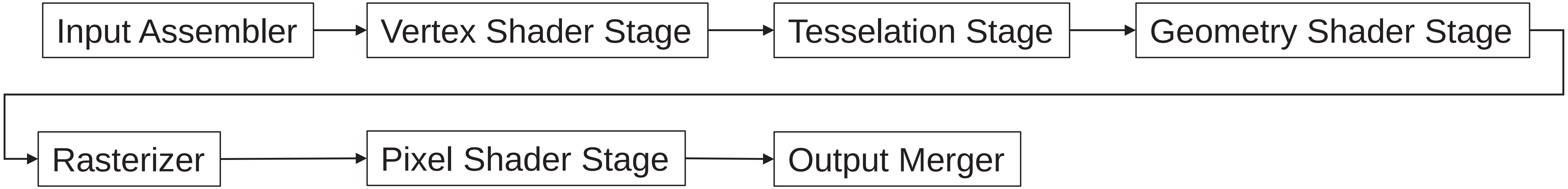
```
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer
vkCmdDrawIndexed
…
```

- Send a batch of commands to the GPU so the GPU is busy for quite a while.
- Every command list submission takes some time!

- `vkCmdBindPipeline`: Sets the GPU into the correct state.

  → Contains the settings for the different stages of the graphics pipeline.
- `vkCmdBindVertexBuffers`: Tells the GPU which vertex buffers to access for the next drawcalls.
- `vkCmdBindIndexBuffer`: Tells the GPU which index buffer to access for the next draw calls.
- `vkCmdDrawIndexed`: The actual draw call → will process the specified vertices according to the state of the GPU.

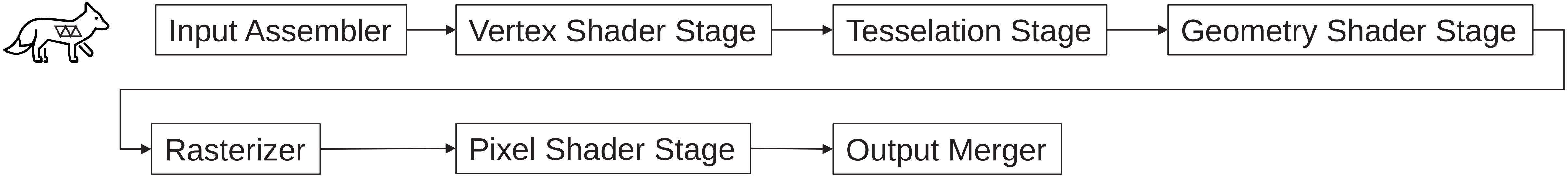
THE GRAPHICS PIPELINE



THE COMPUTE PIPELINE

Compute Shader Stage

THE LOGICAL PIPELINES

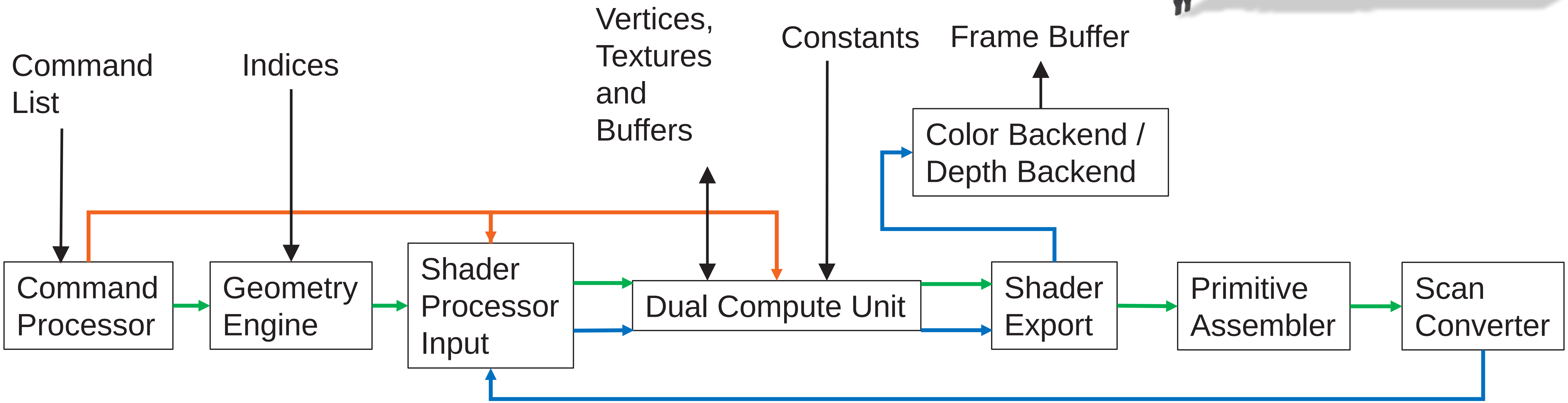


Input Assembler → Vertex Shader Stage → Tesselation Stage → Geometry Shader Stage → Rasterizer → Pixel Shader Stage → Output Merger

e.g.: Compute Lighting Shader

Compute Shader Stage

ON THE RDNA ARCHITECTURE

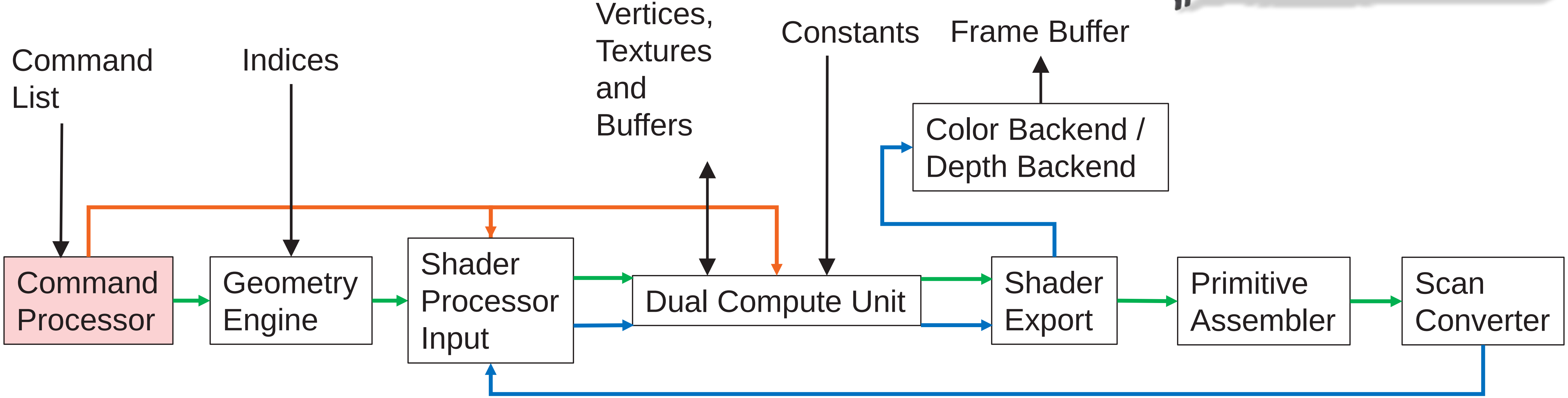


🟩 Vertex Pipeline

🟦 Pixel Pipeline

🟧 Compute Pipeline

ON THE RDNA ARCHITECTURE



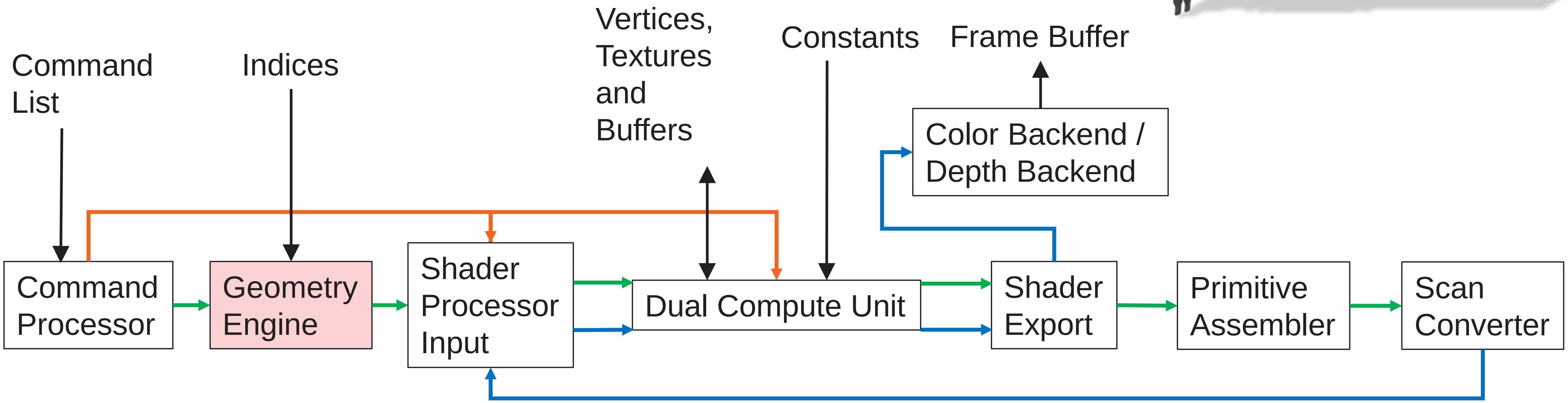
Command Processor (CP)
Processes the list of commands received by the CPU.
Sets the GPU in the correct ‚state‘ to execute the commands.

🟩 Vertex Pipeline

🟦 Pixel Pipeline

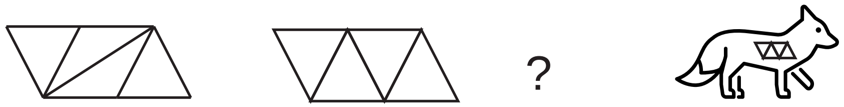
🟧 Compute Pipeline

ON THE RDNA ARCHITECTURE



Geometry Engine (GE)

Knows about topology (points, lines, triangles) / connectivity.

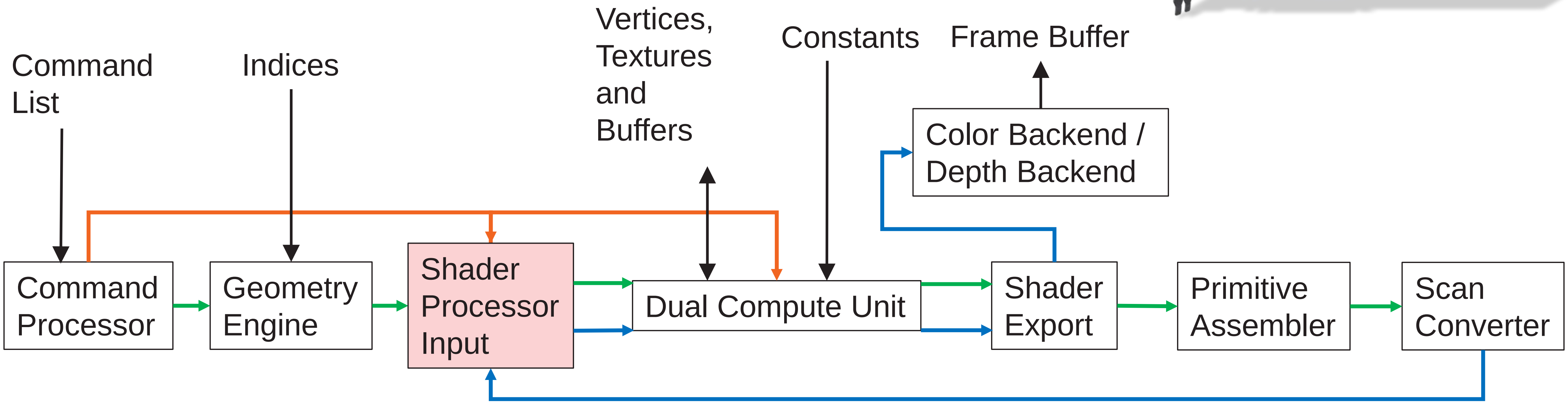


🟩 Vertex Pipeline

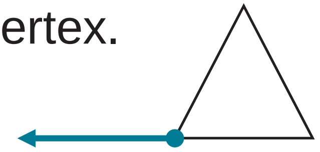
🟦 Pixel Pipeline

🟧 Compute Pipeline

ON THE RDNA ARCHITECTURE



Shader Processor Input (SPI)
Accumulates work items. For a vertex shader, one work item is one vertex.
Sends them in waves to the Dual Compute Unit.
A wave consists of 32 or 64 work items.

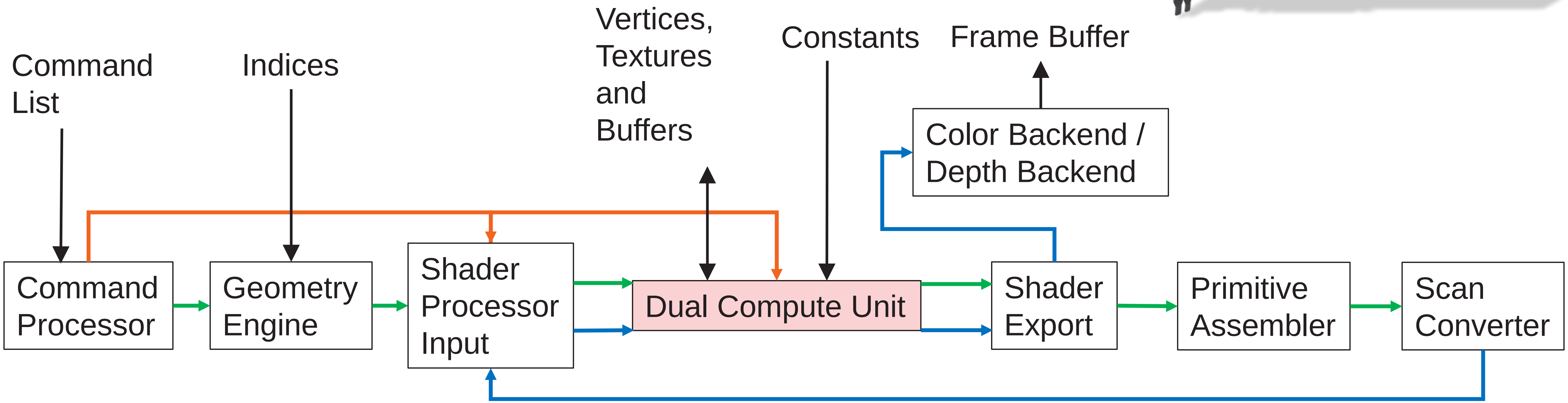


🟩 Vertex Pipeline

🟦 Pixel Pipeline

🟧 Compute Pipeline

ON THE RDNA ARCHITECTURE



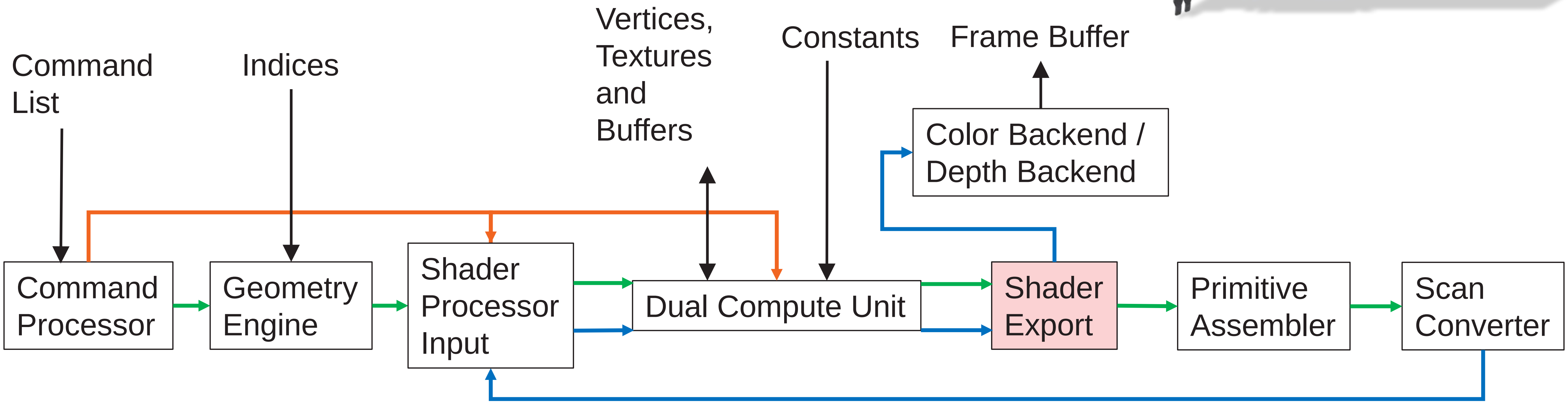
Dual Compute Unit (Dual CU)
Executes shader programs.
Can read and write to memory.
First run is through the vertex pipeline → one work item represents one vertex.

🟩 Vertex Pipeline

🟦 Pixel Pipeline

🟧 Compute Pipeline

ON THE RDNA ARCHITECTURE



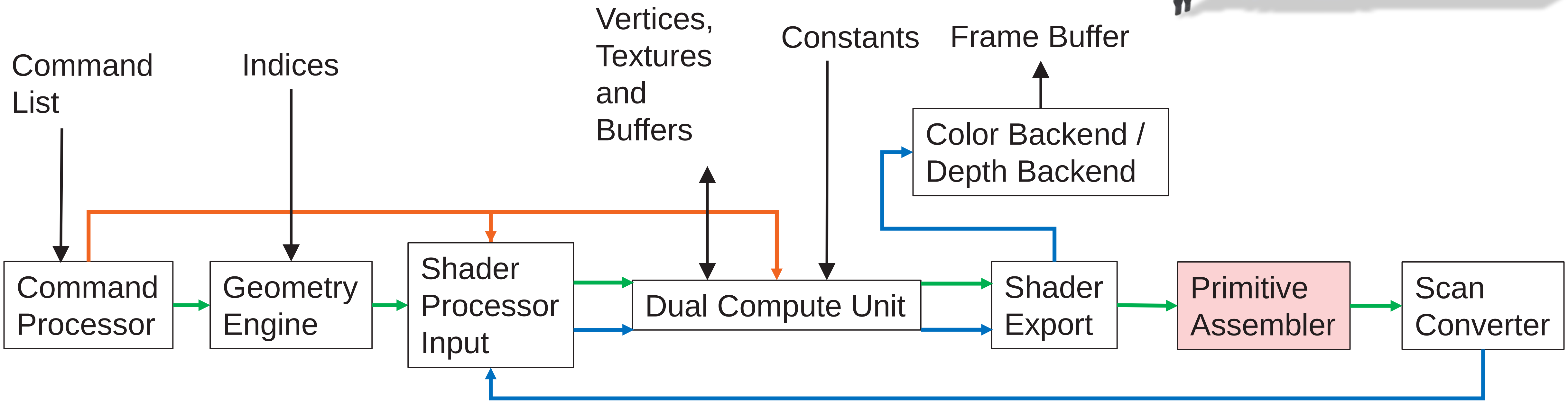
Shader Export (SX)
Handles special output from the Dual CU.

🟩 Vertex Pipeline

🟦 Pixel Pipeline

🟧 Compute Pipeline

ON THE RDNA ARCHITECTURE



Command List

Indices

Vertices, Textures and Buffers

Constants

Frame Buffer

Color Backend / Depth Backend

Command Processor

Geometry Engine

Shader Processor Input

Dual Compute Unit

Shader Export

Primitive Assembler

Scan Converter

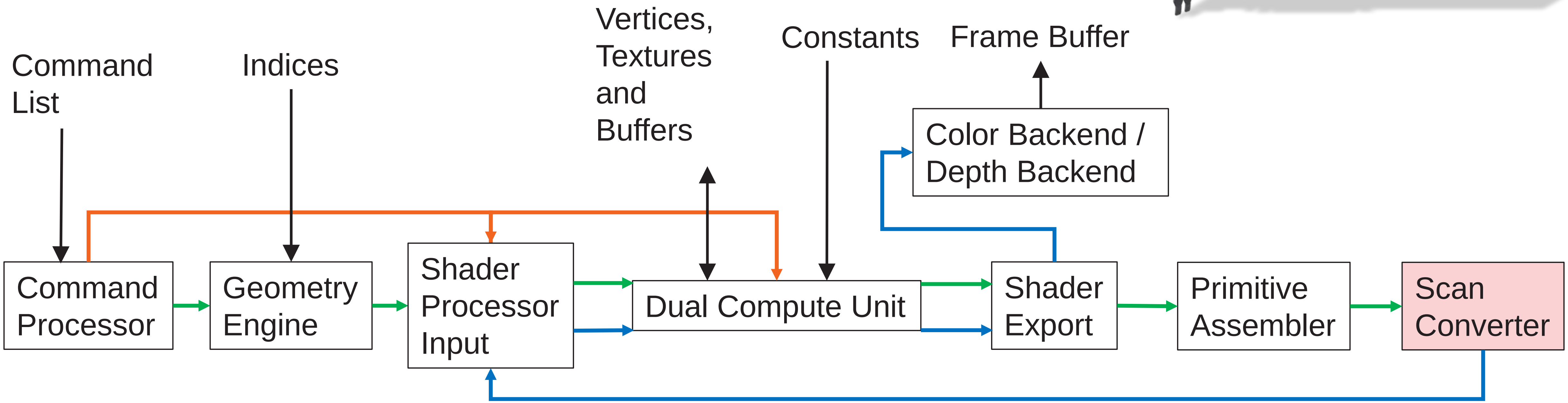
Primitive Assembler (PA)
Accumulates vertices that span a triangle.
Forwards triangles to Scan Converter.

🟩 Vertex Pipeline

🟦 Pixel Pipeline

🟧 Compute Pipeline

ON THE RDNA ARCHITECTURE



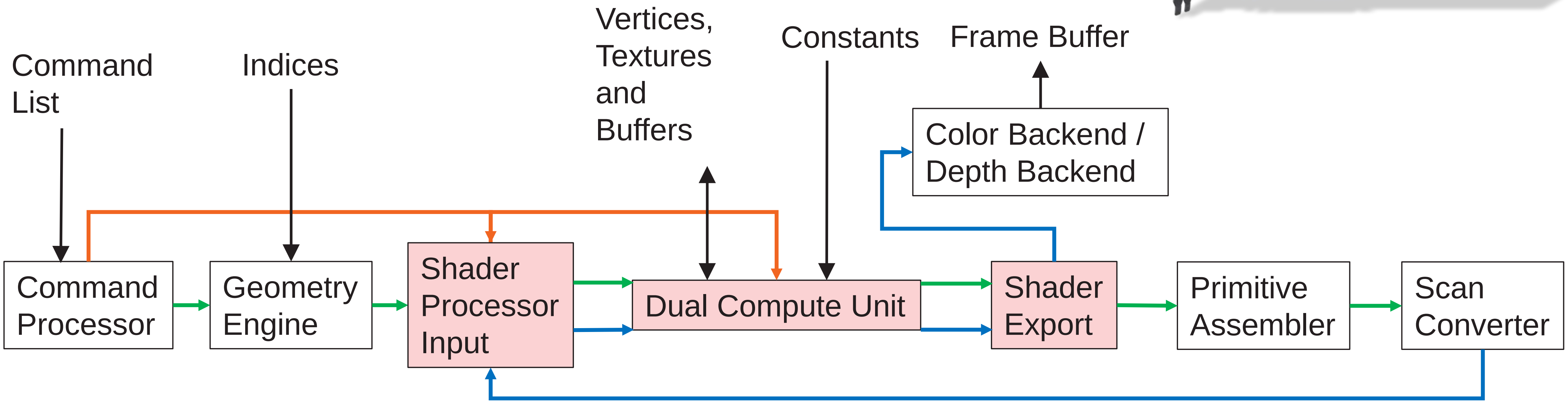
Scan Converter (SC)
Determine pixels covered by each triangle.
Forwards them to Shader Processor Input (SPI).

🟩 Vertex Pipeline

🟦 Pixel Pipeline

🟧 Compute Pipeline

ON THE RDNA ARCHITECTURE



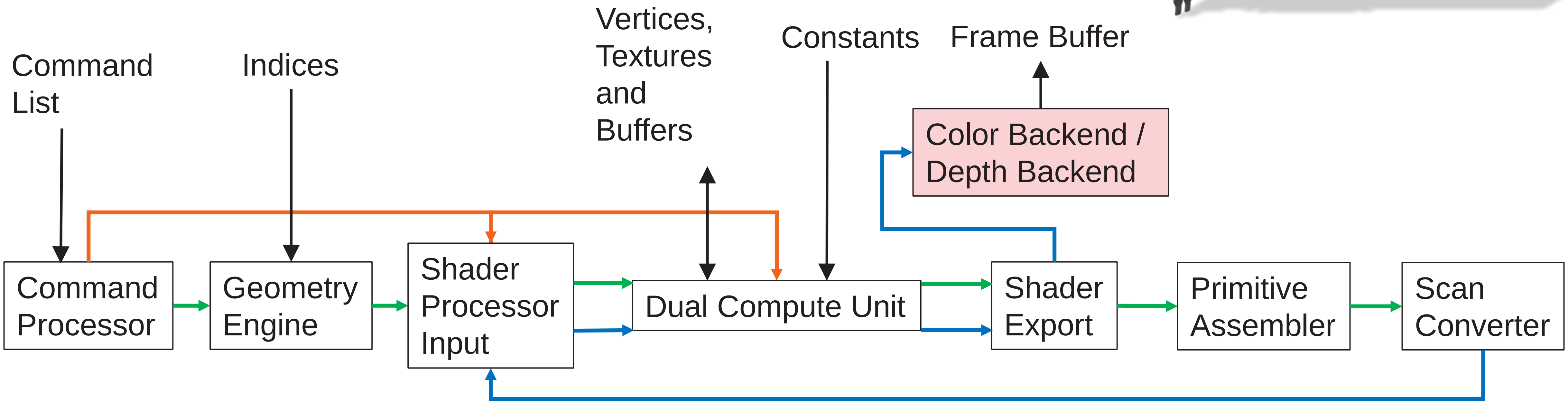
SPI, Dual CU, SX – Pixel Pipeline
One work item represents one pixel (also called fragment).

🟩 Vertex Pipeline

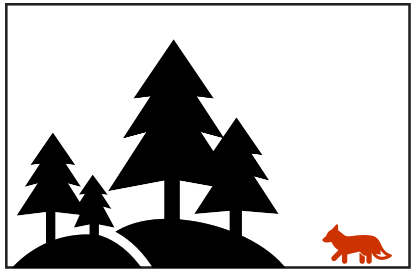
🟦 Pixel Pipeline

🟧 Compute Pipeline

ON THE RDNA ARCHITECTURE



Color Backend / Depth Backend
Discard occluded pixels based on depth / stencil.
Write colored pixels to render targets.

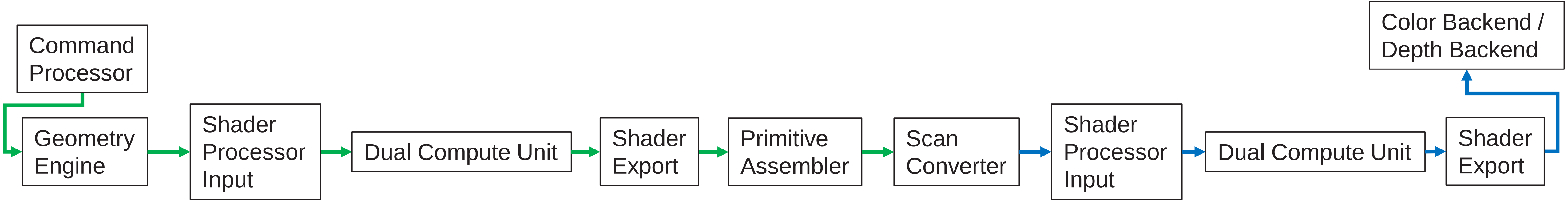


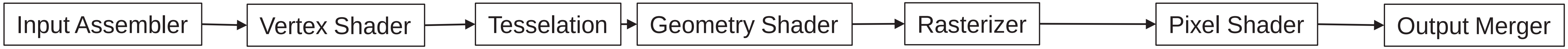
🟩 Vertex Pipeline

🟦 Pixel Pipeline

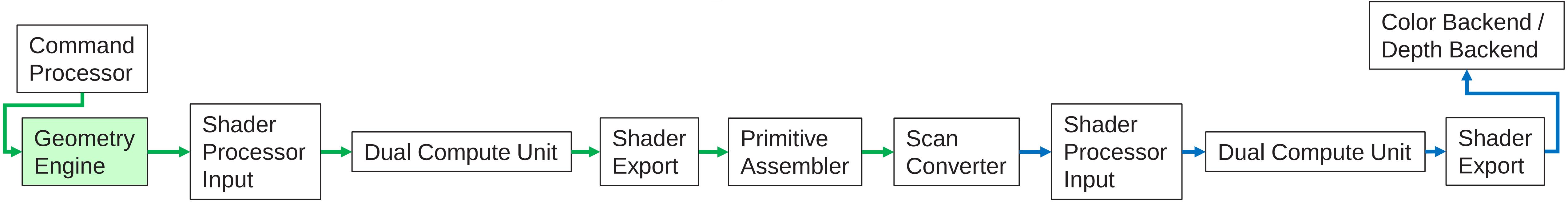
🟧 Compute Pipeline

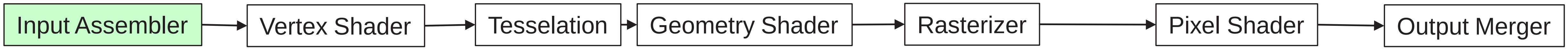
LOGICAL PIPELINE → HARDWARE





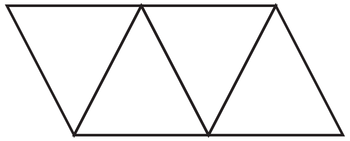
LOGICAL PIPELINE → HARDWARE



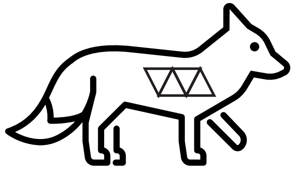


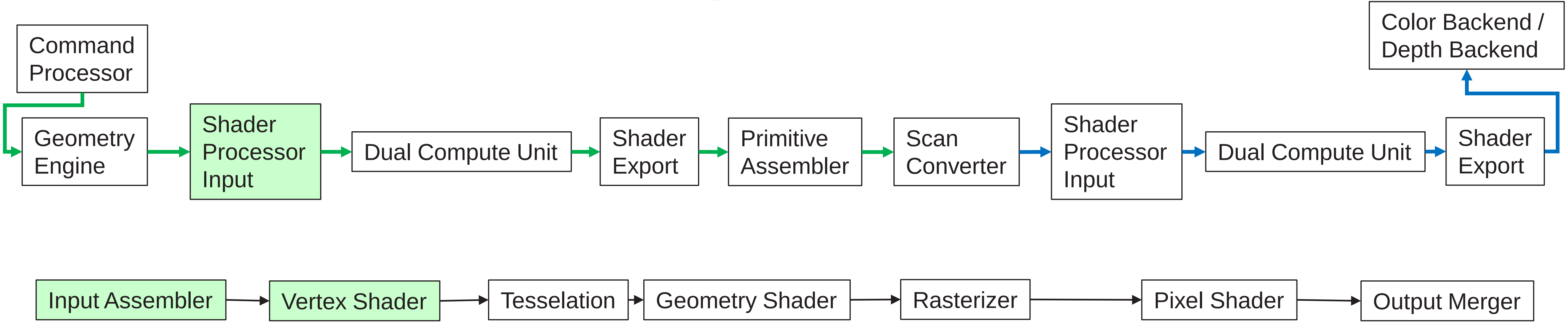
- Retrieves indices from index buffer.
- Knows about the topology.
  - Triangles, line, point.
- Holds a cache for vertex reuse.
  - Avoid shading vertices multiple times.
- Forwards index to Shader Processor Input (SPI).

0	1	2	2	3	0	1	5	6	6	2	1	7	6	5	5	4	7

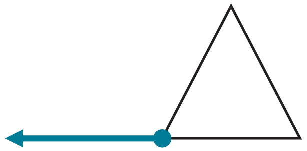


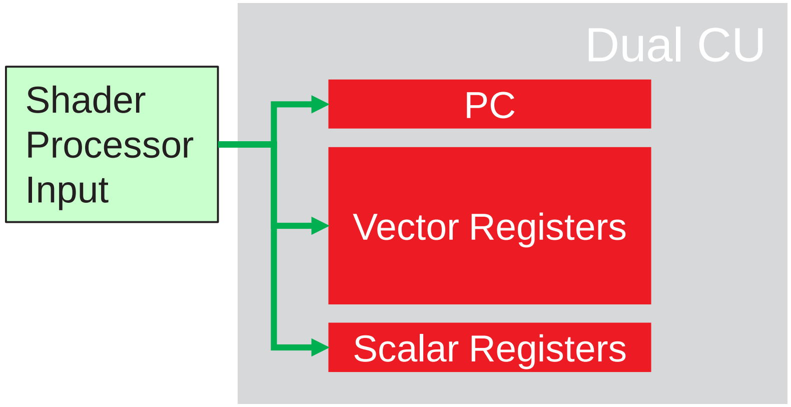
LOGICAL PIPELINE → HARDWARE



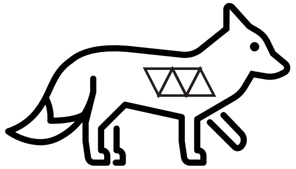


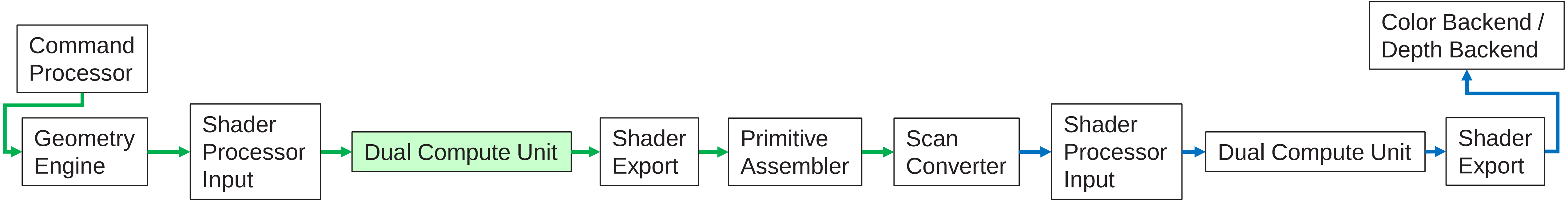
- Waits until there are enough indices before bothering the Dual Compute Unit.
- Chooses a Dual CU.
- Configures Dual CU.
  - Reserves resources in Dual CU.
  - Loads address of vertex shader program into the program counter (PC).
  - Initializes scalar and vector registers in Dual CU.
- Kicks off work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





LOGICAL PIPELINE → HARDWARE




Command Processor

Geometry Engine

Shader Processor Input

Dual Compute Unit

Shader Export

Primitive Assembler

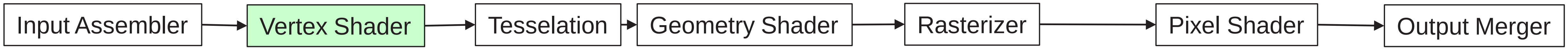
Scan Converter

Shader Processor Input

Dual Compute Unit

Shader Export

Color Backend / Depth Backend


Input Assembler

Vertex Shader

Tesselation

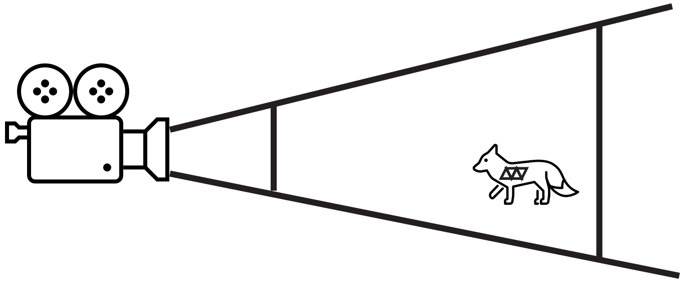
Geometry Shader

Rasterizer

Pixel Shader

Output Merger

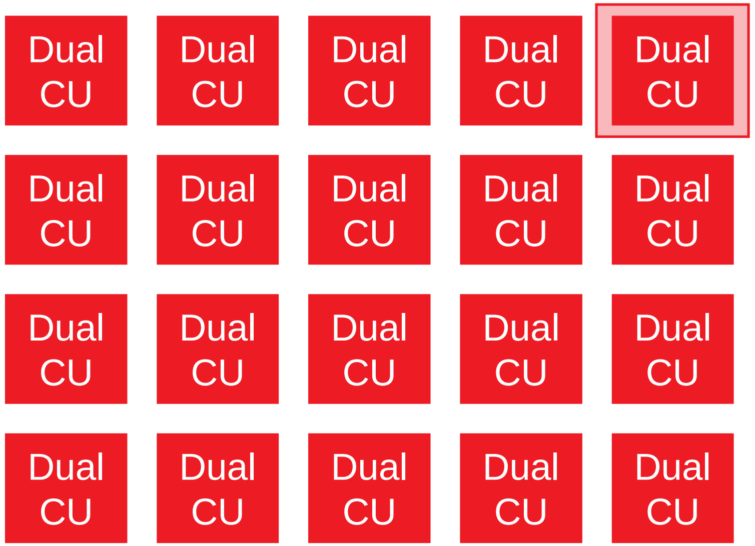
- Usually transforms vertices from object space to clip space.
- Attaches additional vertex attributes.
- Usually same transformations for every vertex in the mesh.
  → Parallelizable.

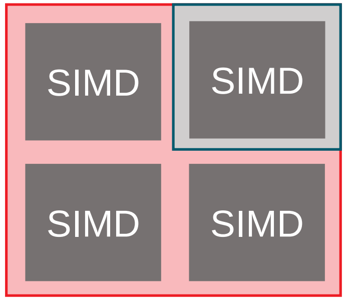


DUAL COMPUTE UNIT

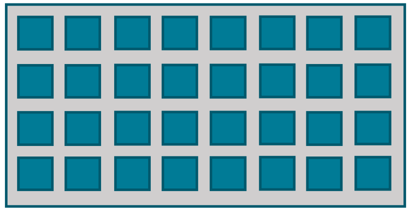
Dual Compute Units are designed to execute parallel workloads!

The number of Dual CUs depends on the card,
e.g. Radeon™ RX 5700 XT has 20 Dual CUs.





4 x 32-wide SIMDs per Dual CU



32 threads per SIMD

One vector register (VGPR) holds one value per thread.

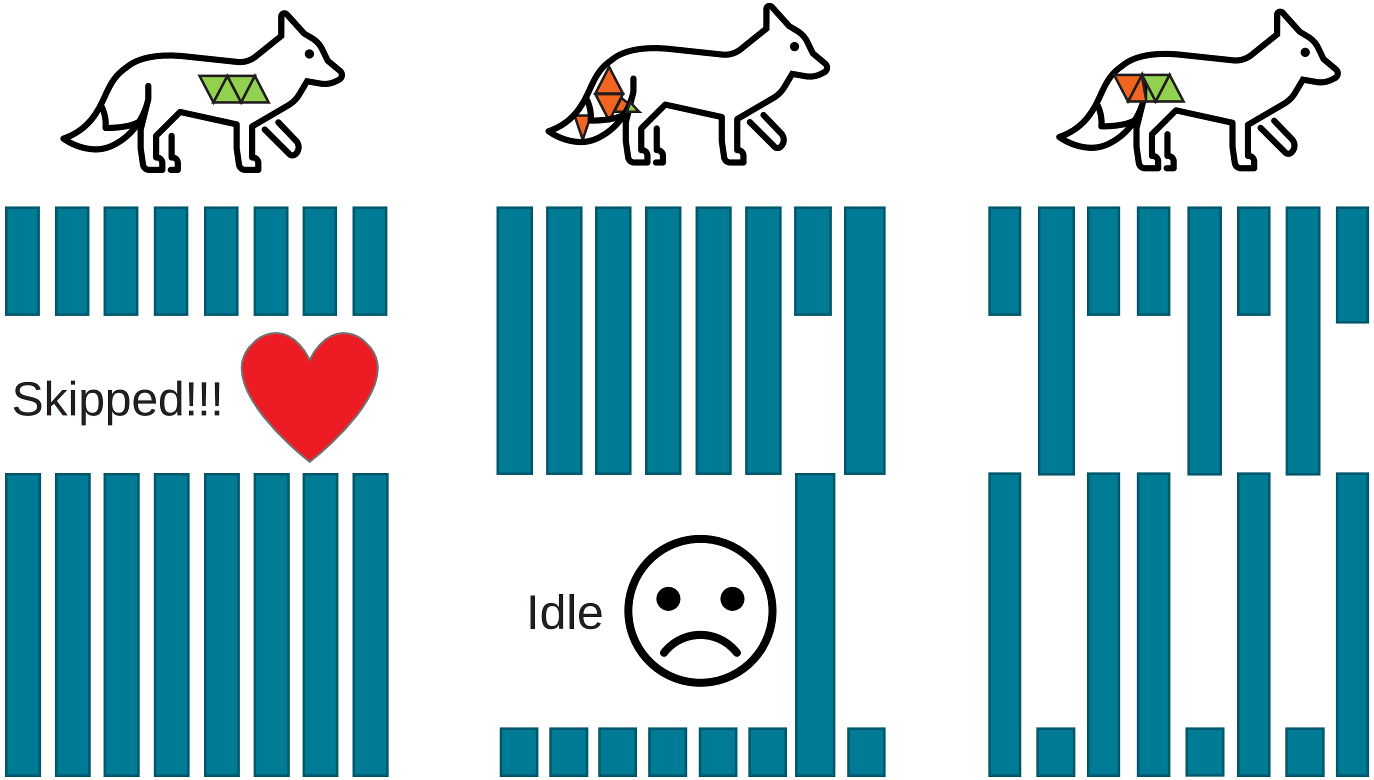
One scalar register (SGPR) holds one value per wave.

DUAL COMPUTE UNIT

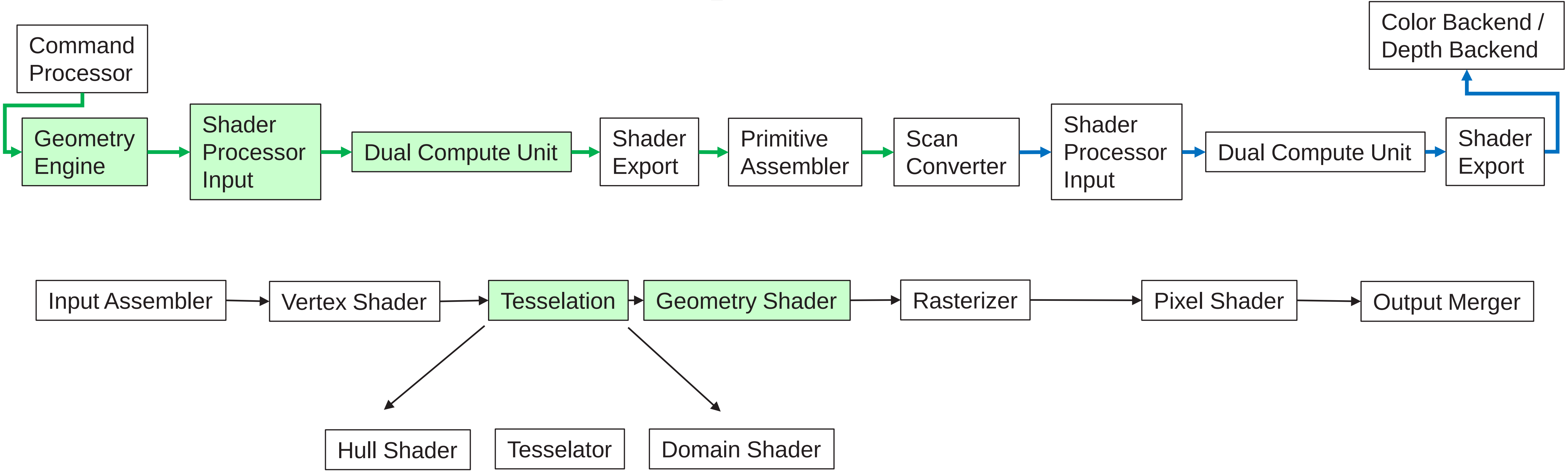
Dual Compute Units are designed to execute parallel workloads!

Example Pseudo-Code:

```
…
if (vertex == tail) {
// Do something fox tail specific
}
else {
// Do something else
}
…
```

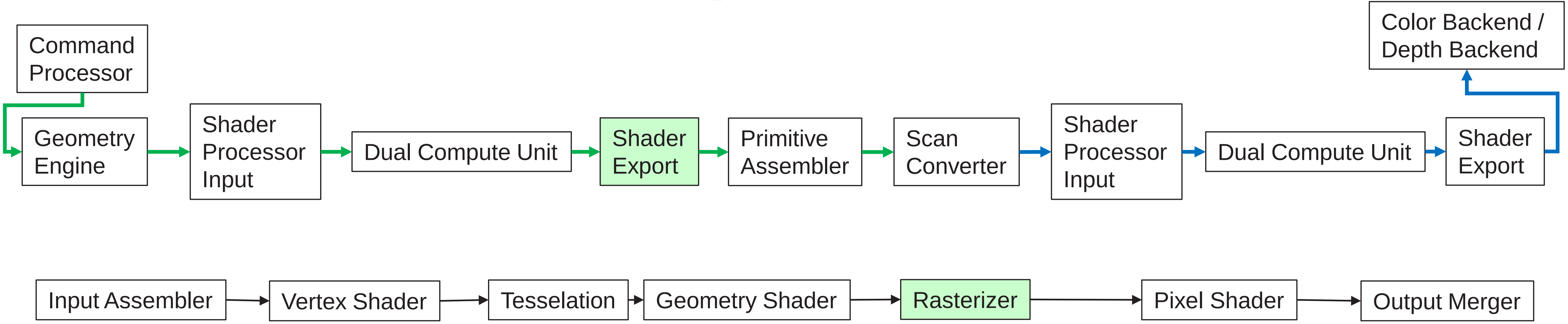


LOGICAL PIPELINE → HARDWARE



Optional. It can be used to further transform the geometry.

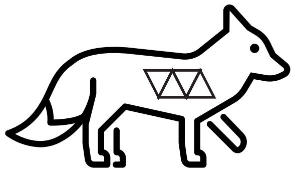
LOGICAL PIPELINE → HARDWARE

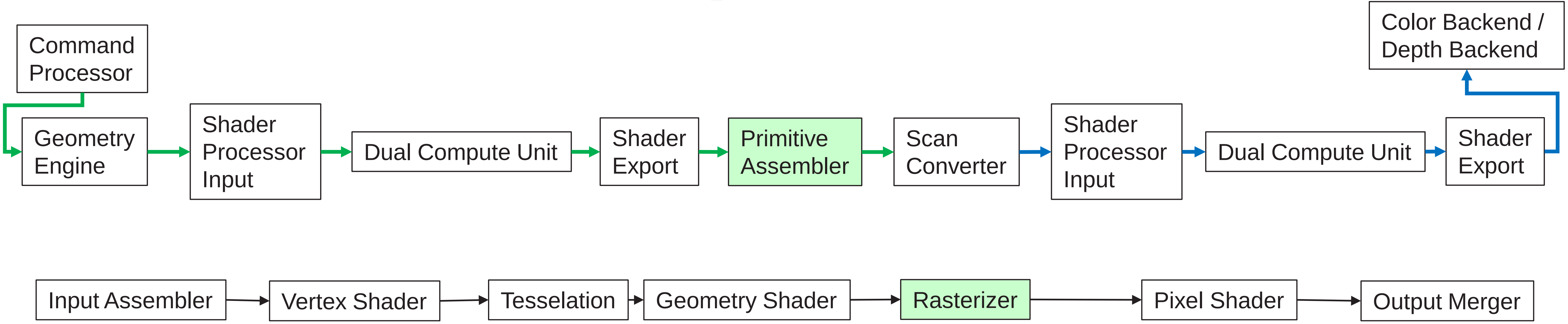


- Gets transformed vertices from the Dual CUs.
- Decides if we forward to the Color Backend / Depth Backend or to the Primitive Assembler.

  → On the vertex pipeline, it forwards to the Primitive Assembler.

LOGICAL PIPELINE → HARDWARE



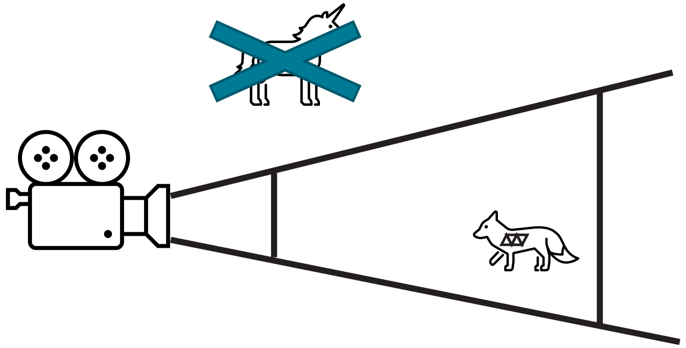


Command Processor

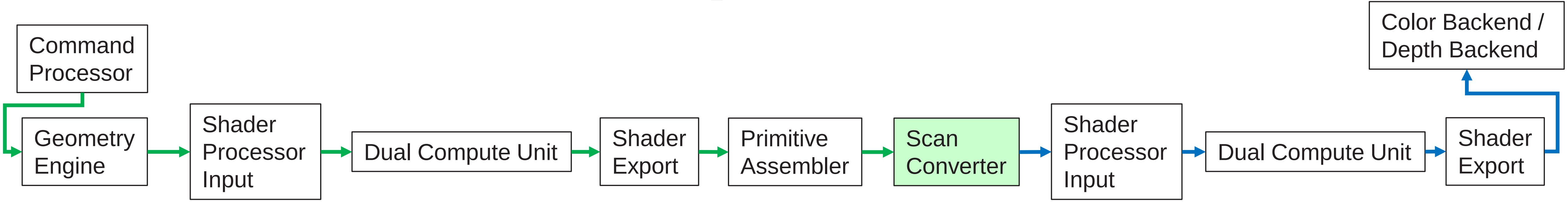
Geometry Engine → Shader Processor Input → Dual Compute Unit → Shader Export → Primitive Assembler → Scan Converter → Shader Processor Input → Dual Compute Unit → Shader Export → Color Backend / Depth Backend

Input Assembler → Vertex Shader → Tesselation → Geometry Shader → Rasterizer → Pixel Shader → Output Merger

- Uses the connectivity info stored by the Geometry Engine earlier.
- Waits for 3 connected vertex positions to appear.
- Transforms vertices to screen space.
- View Frustum Culling.
- Backface Culling.
- Forwards triangle to Scan Converter.

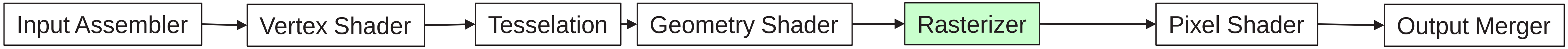


LOGICAL PIPELINE → HARDWARE



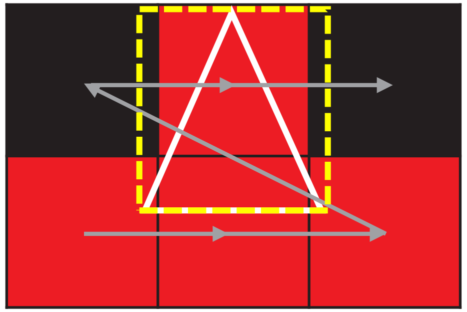
Command Processor

Geometry Engine → Shader Processor Input → Dual Compute Unit → Shader Export → Primitive Assembler → Scan Converter → Shader Processor Input → Dual Compute Unit → Shader Export → Color Backend / Depth Backend

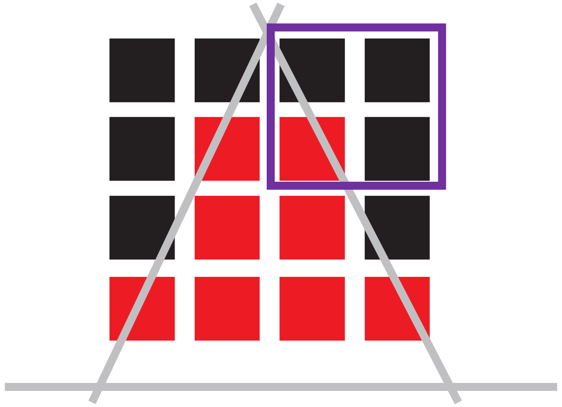


Input Assembler → Vertex Shader → Tesselation → Geometry Shader → Rasterizer → Pixel Shader → Output Merger

- Determine all pixels that overlap the triangle.
  → Rasterization
- Scan Conversion only on a coarse level.

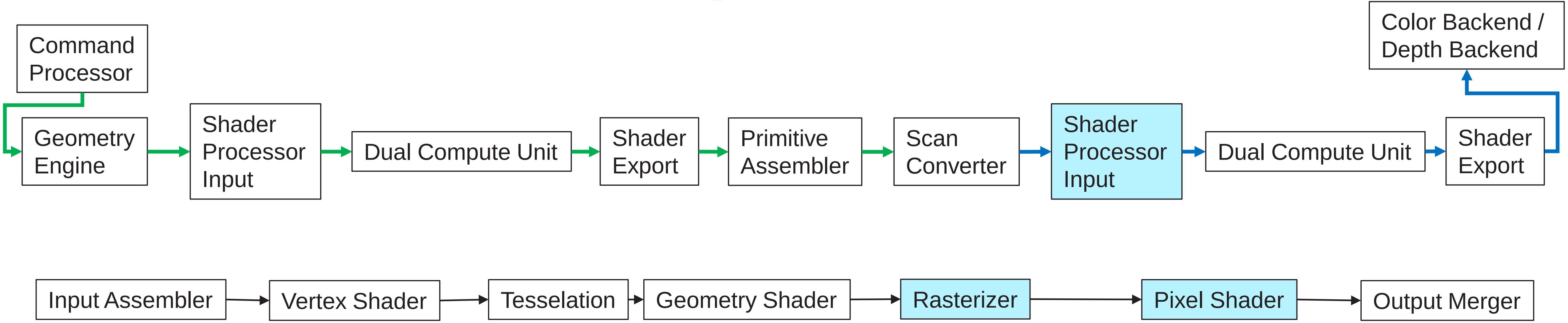


On a fine level (4x4 pixels) test against the triangle edges.



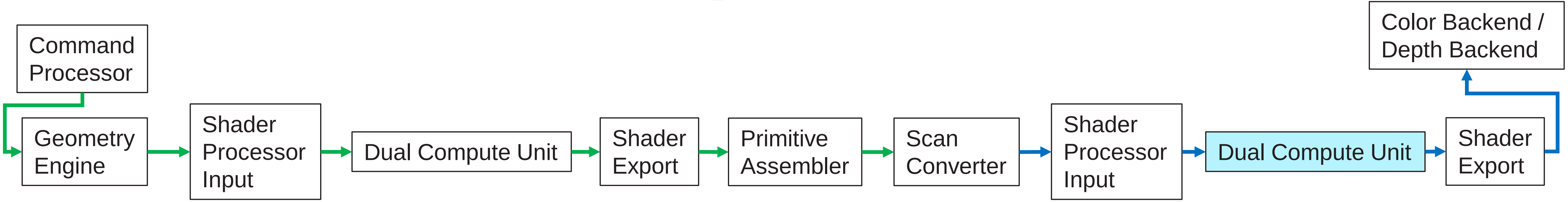
Forwards **quads** to the Shader Processor Input.

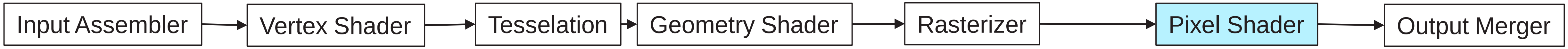
LOGICAL PIPELINE → HARDWARE



- Gets enough quads from the Scan Converter.
- Chooses Dual Compute Unit, sets it up.
- A work item is now a pixel / fragment!

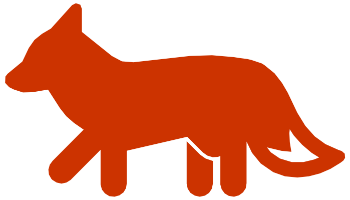
LOGICAL PIPELINE → HARDWARE



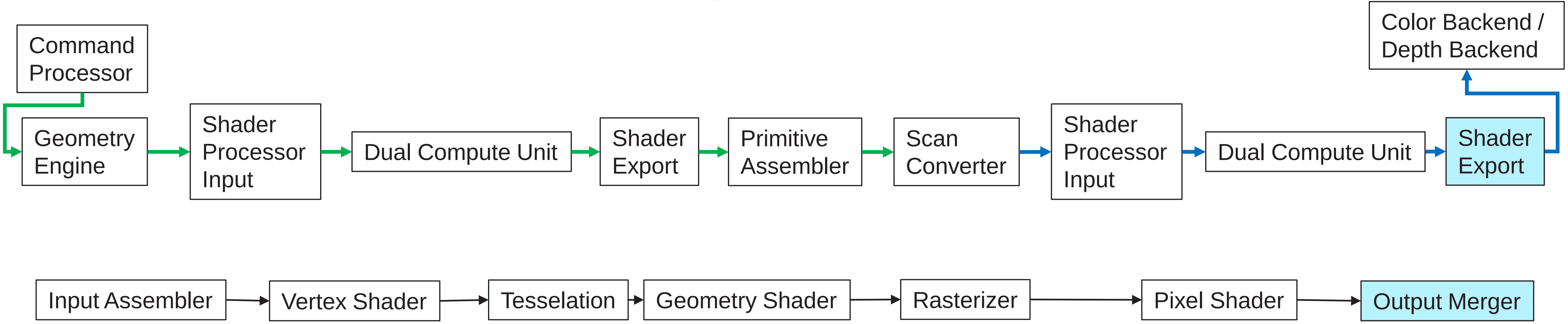


- A lot of small triangles covering only a single pixel → a lot of threads are masked out:

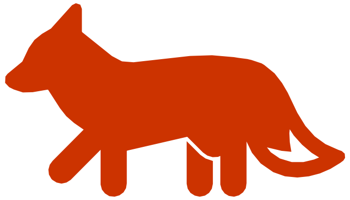
- Pixels within the quad not covered by the triangle lead to inactive threads!
- Export finished pixels / fragments via Shader Export.



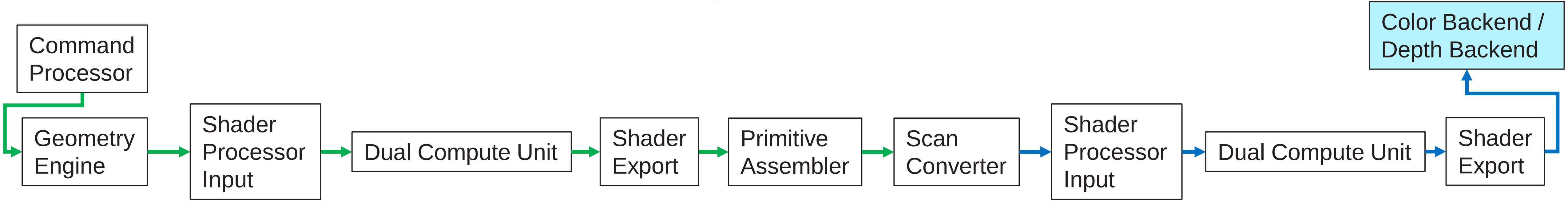
LOGICAL PIPELINE → HARDWARE

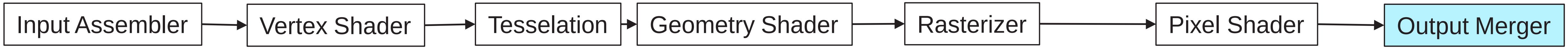


- This time, the Shader Export forwards the fragments to the Color Backend / Depth Backend instead of the Primitive Assembler.

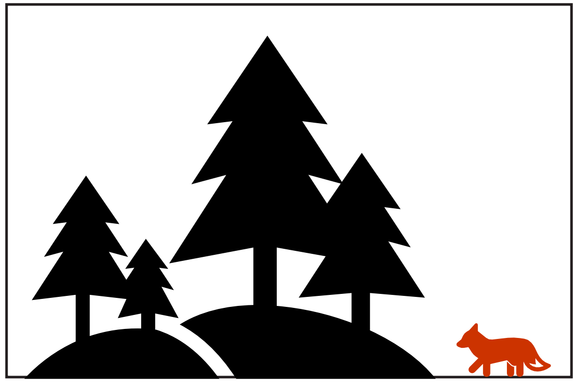


LOGICAL PIPELINE → HARDWARE

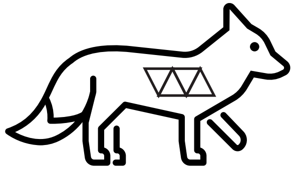


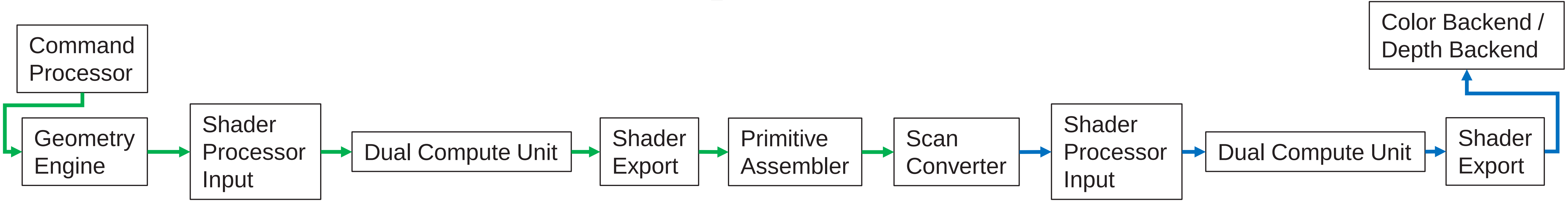


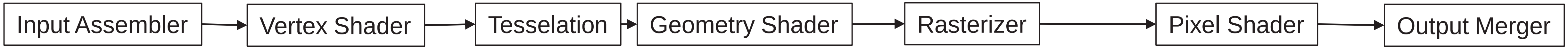
- Color Blending.
- Writes the fragment color to the bound render targets.
- MSAA resolve.
- Compression.
- …

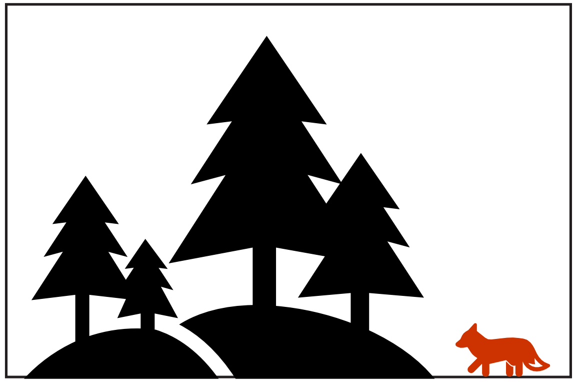


LOGICAL PIPELINE → HARDWARE

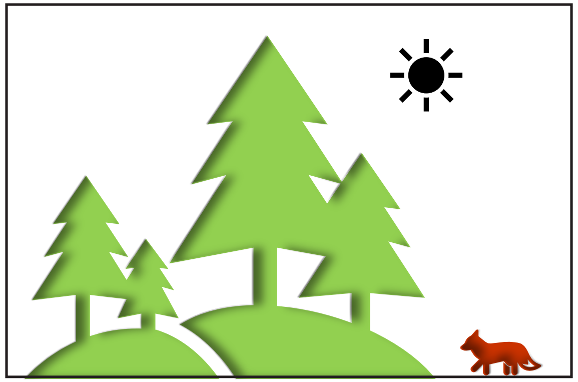




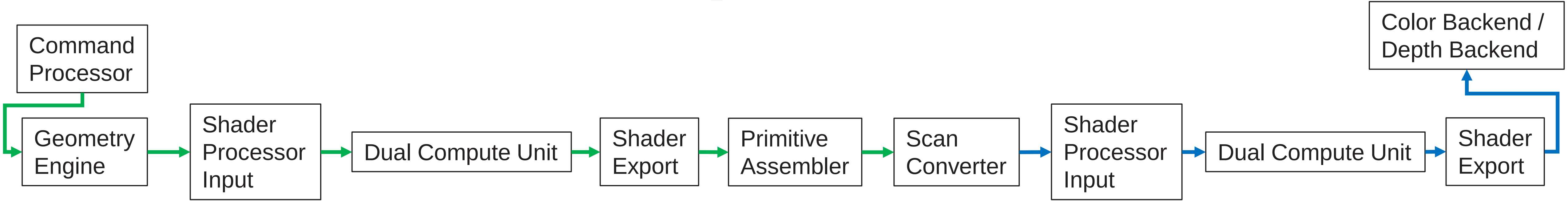




- Use this rendertarget as input for the next pass.
- Do some calculations.
- Output another rendertarget with same size.
- Every pixel of the output render target needs to be modified.



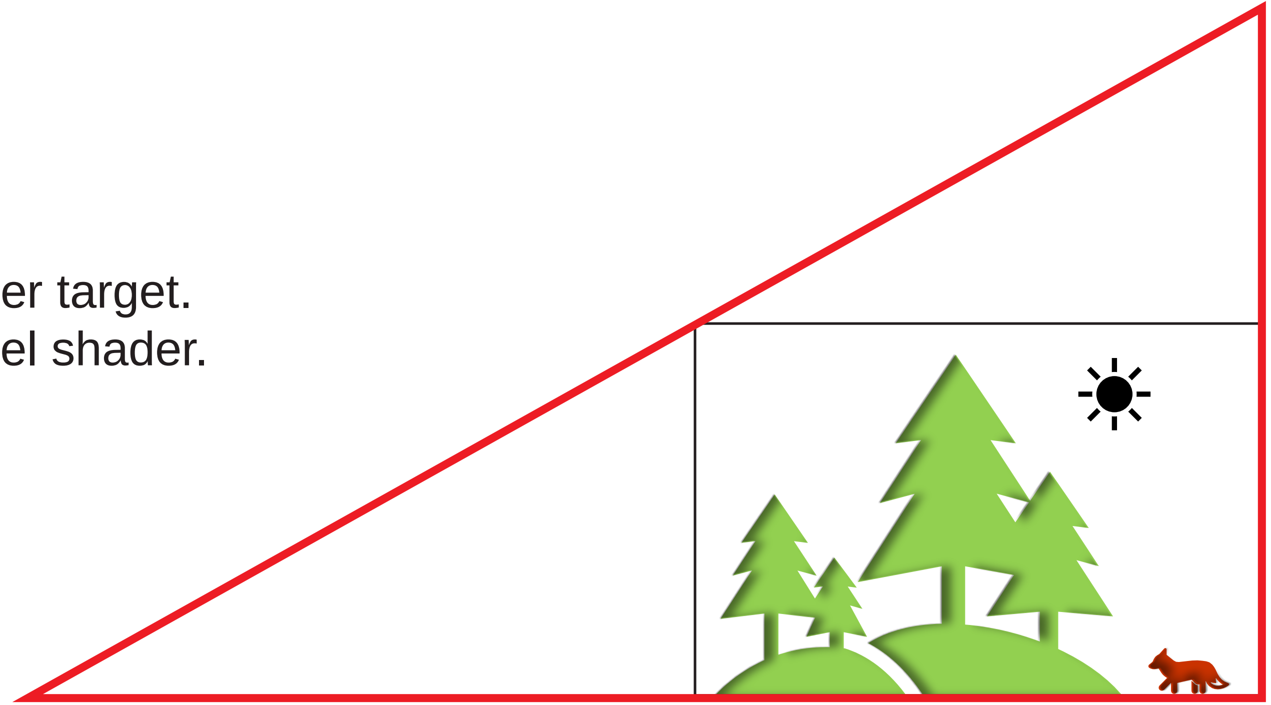
LOGICAL PIPELINE → HARDWARE



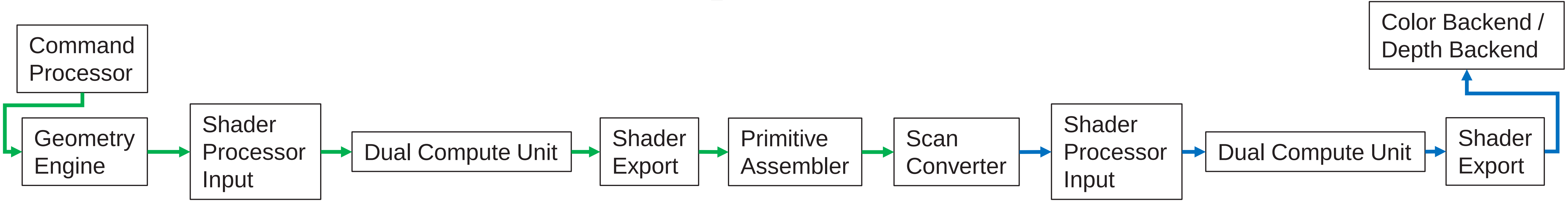
Command Processor

Geometry Engine → Shader Processor Input → Dual Compute Unit → Shader Export → Primitive Assembler → Scan Converter → Shader Processor Input → Dual Compute Unit → Shader Export → Color Backend / Depth Backend

- Input a triangle spanning over the whole output render target.
- The rasterizer will then forward every pixel to the pixel shader.

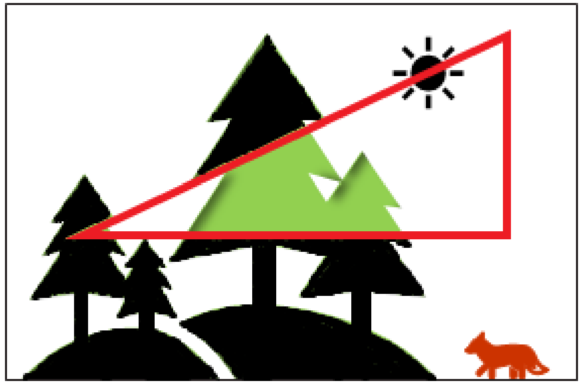


LOGICAL PIPELINE → HARDWARE

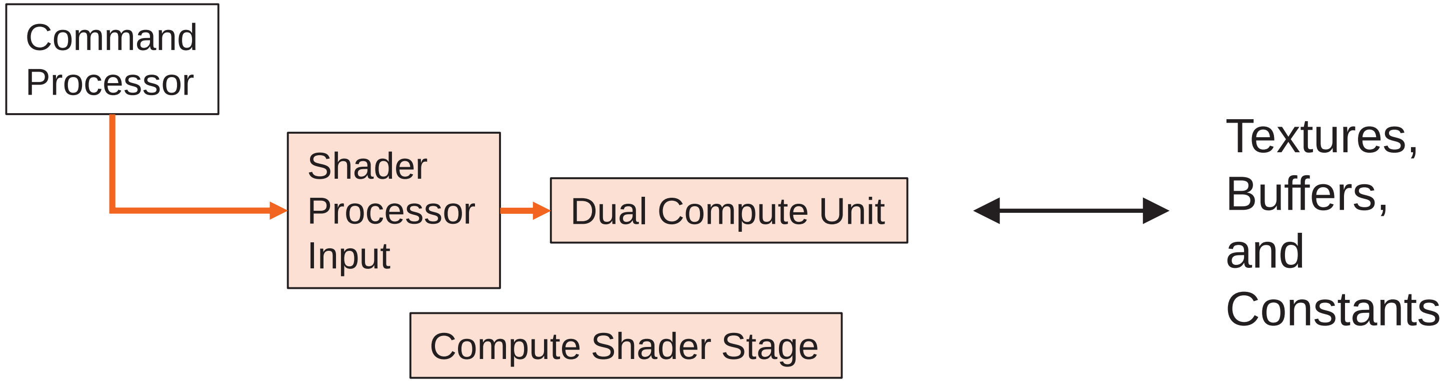


- Input a triangle spanning over the whole output render target.
- The rasterizer will then forward every pixel to the pixel shader.

- If the triangle is not spanning over the whole output render target, only a part of the output render target will be modifed!



THE COMPUTE PIPELINE



Command Processor

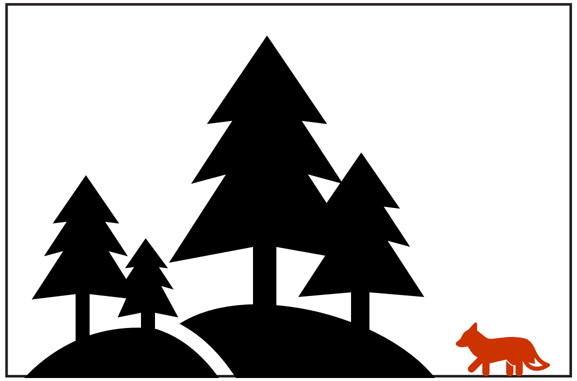
Shader Processor Input

Dual Compute Unit

Compute Shader Stage

Textures,
Buffers,
and
Constants

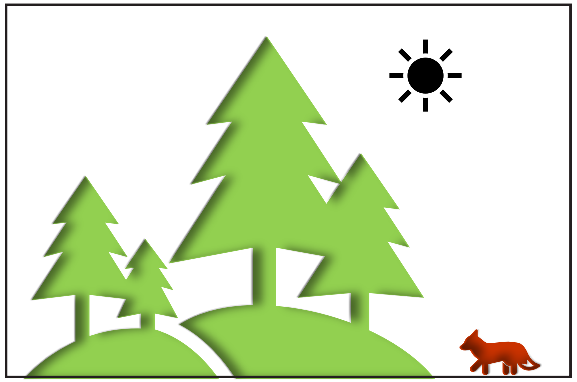
Remember: Dual Compute Units can read and write to memory.



Input: render target from previous pass.

Do some calculations using the data from the Input texture using a compute shader.

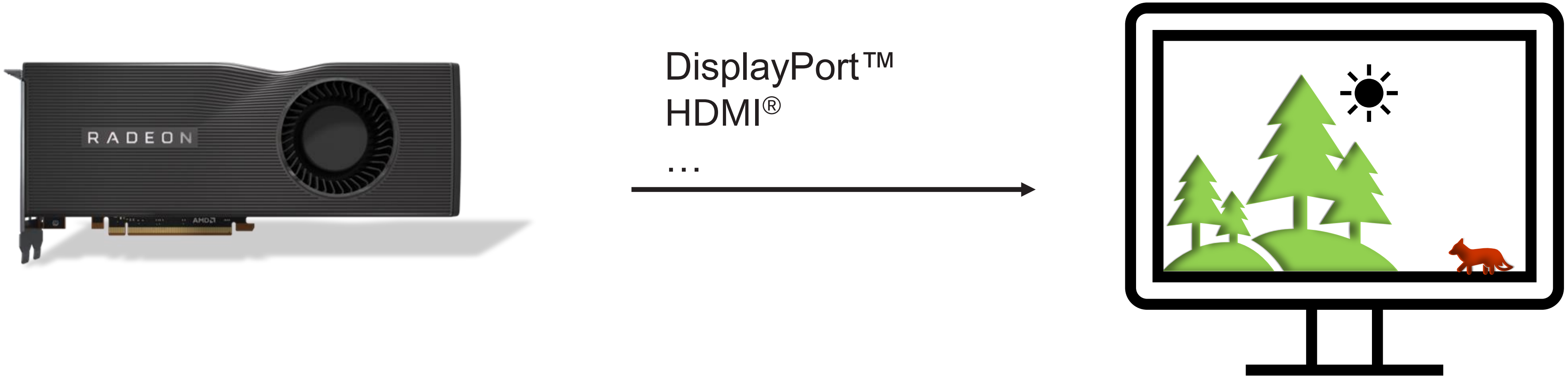
Dispatch as many threads as the texture has pixels.
→ For compute shaders, the number of threads is explicitly specified.



Output: writable texture.

PRESENTATION

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



DisplayPort™
HDMI®
…

Q&A

lou.kramer@amd.com

GPUOpen.com

Special thanks to Dominik Baumeister and his talk
"Triangles Are Precious – Let‘s Treat Them With Care".

DISCLAIMER & ATTRIBUTES

Disclaimer:

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED 'AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD