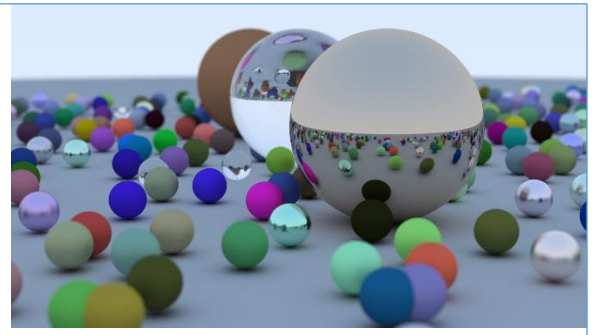
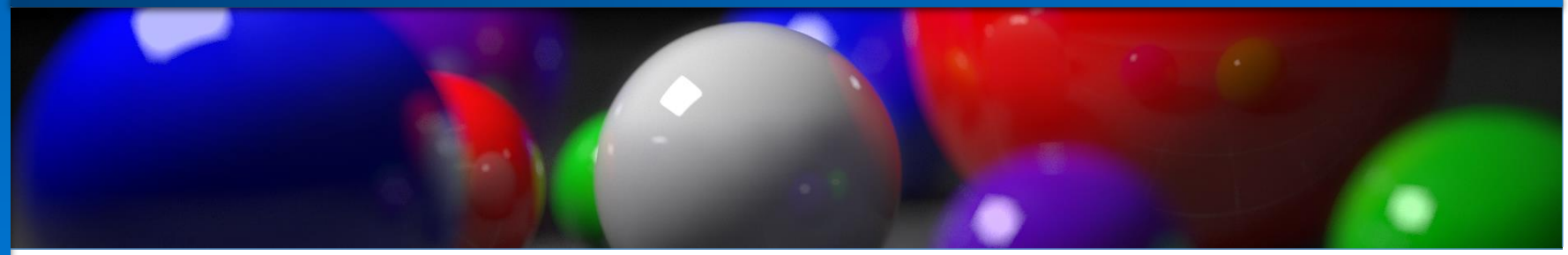
# Comp4422

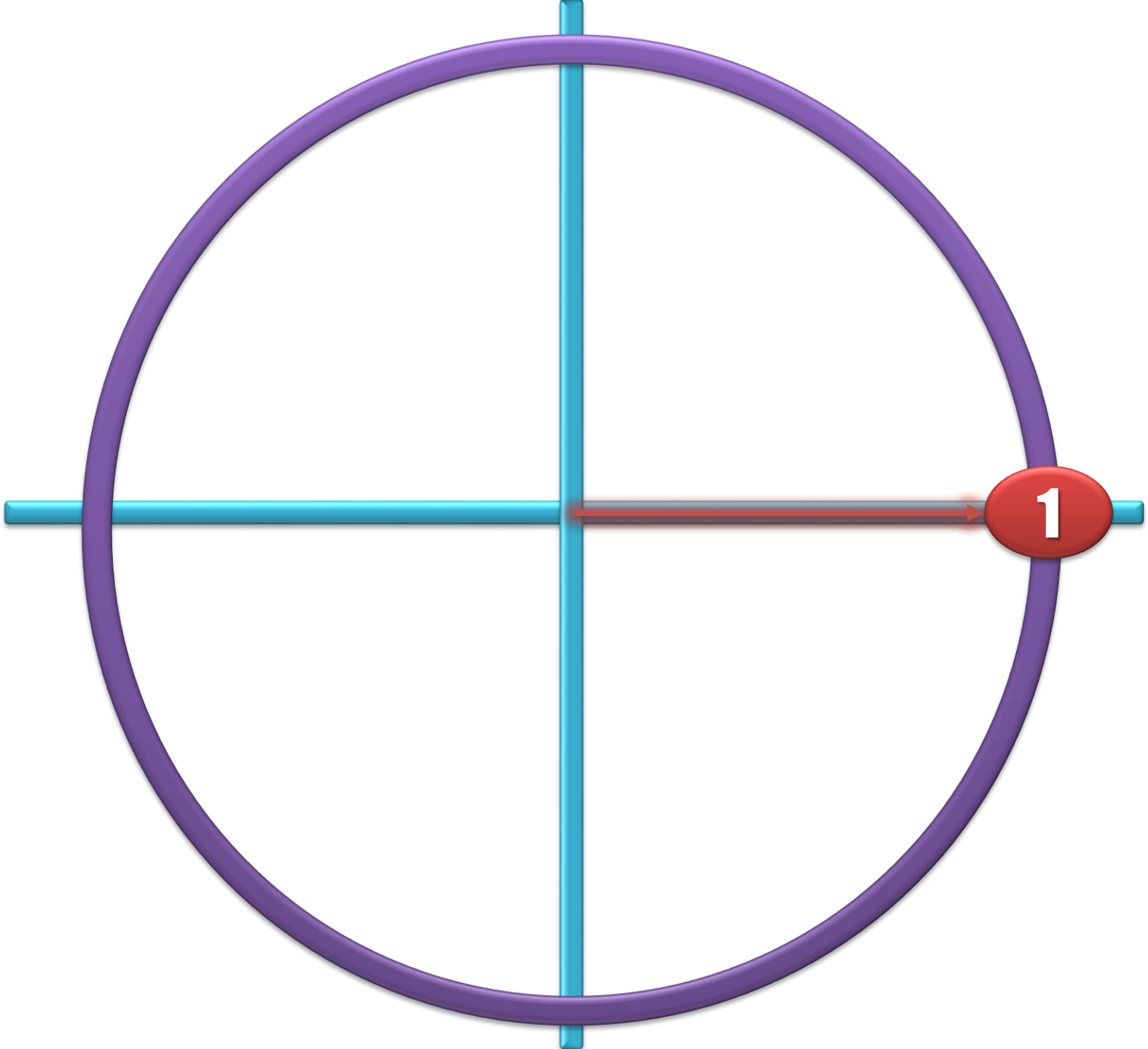


# Computer Graphics

## Lecture 02:

## GPUs




1

# Data Flow

CPU 1

- Application
- Modeling Transformations

GPU 2

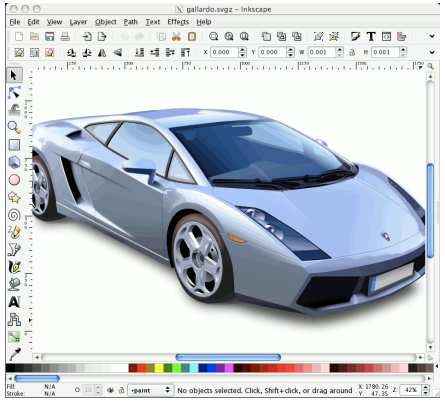
- Viewing, Projection, Clipping
- Vertices and Fragments

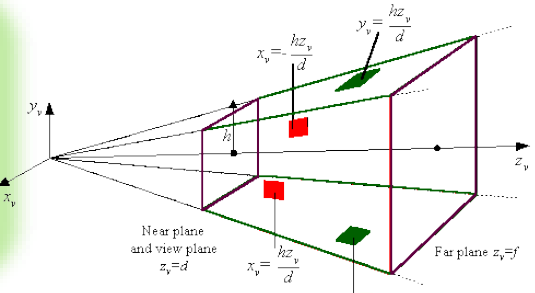
3

- GPU VRAM
- Framebuffer

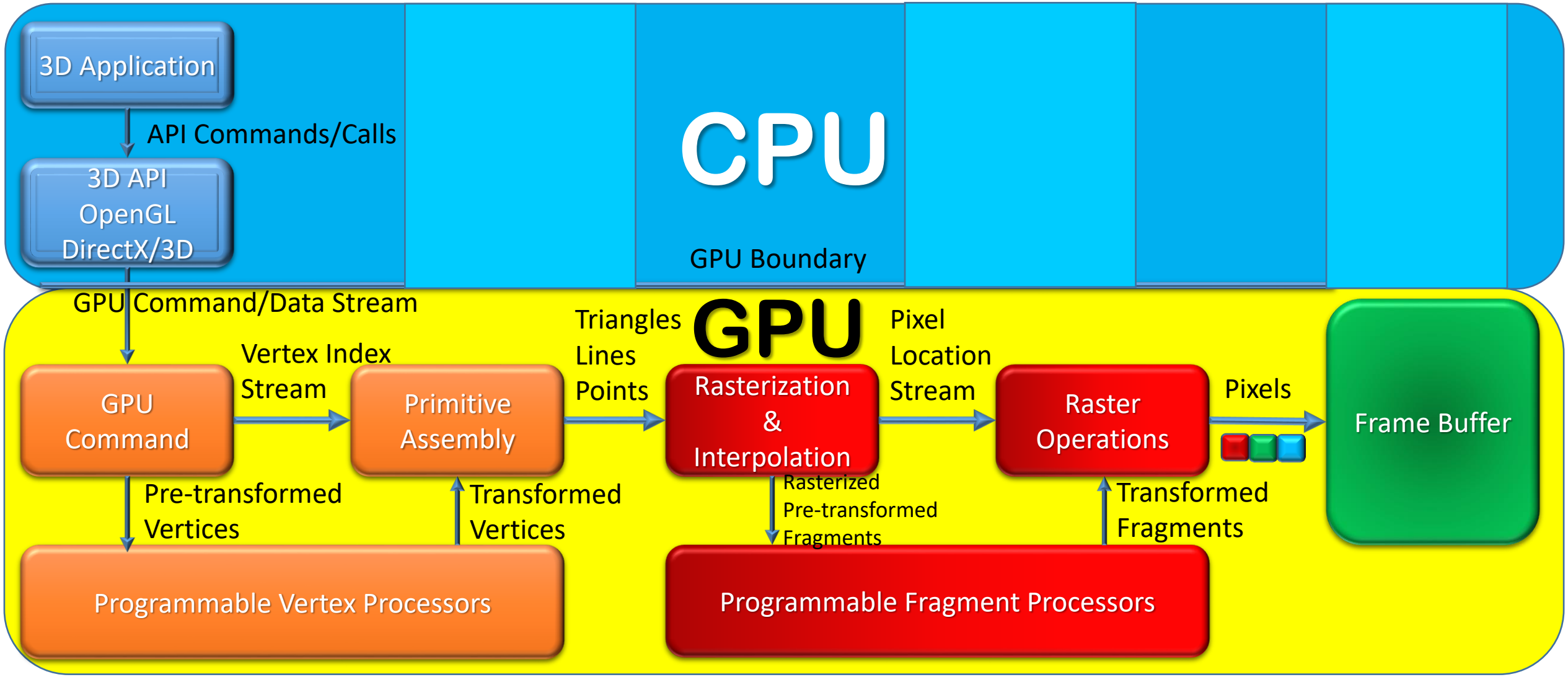
Display

- Displays

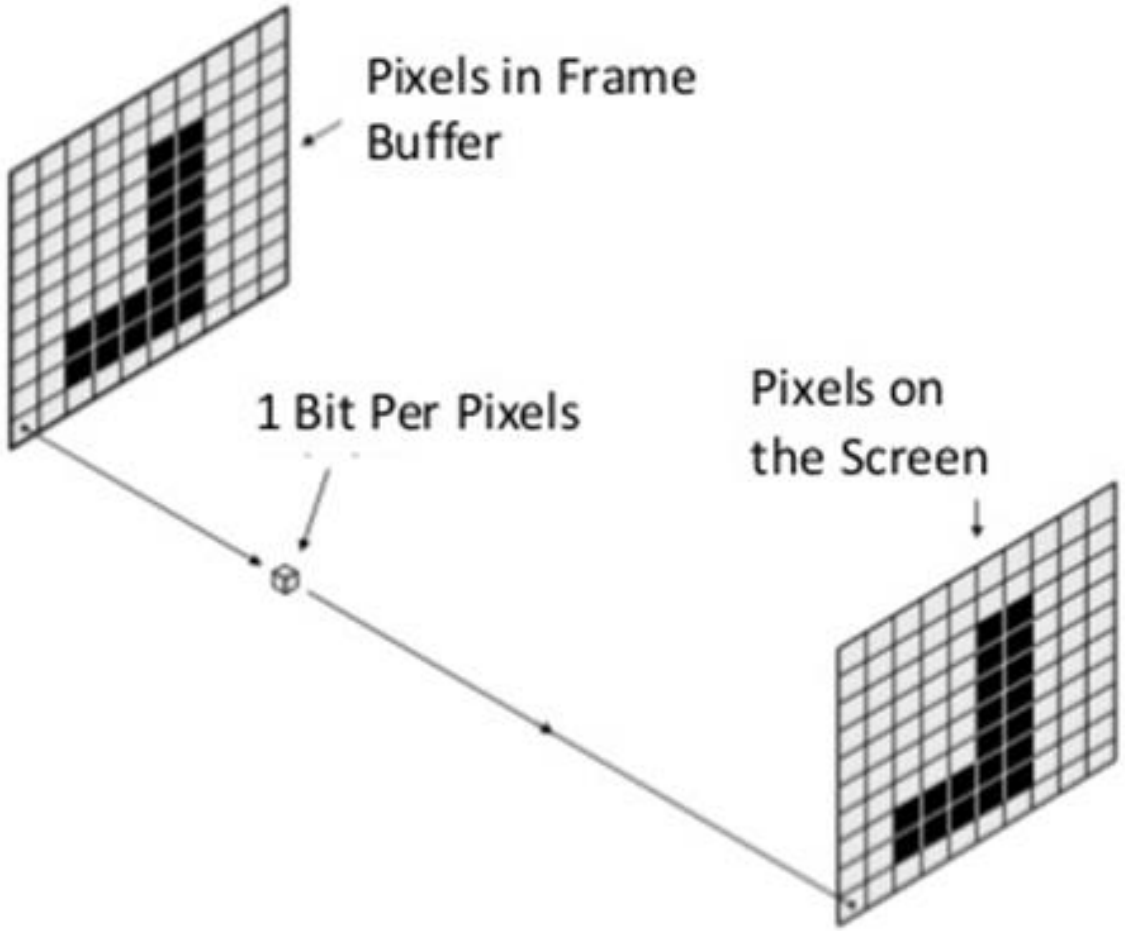





# CPU-GPU Boundary

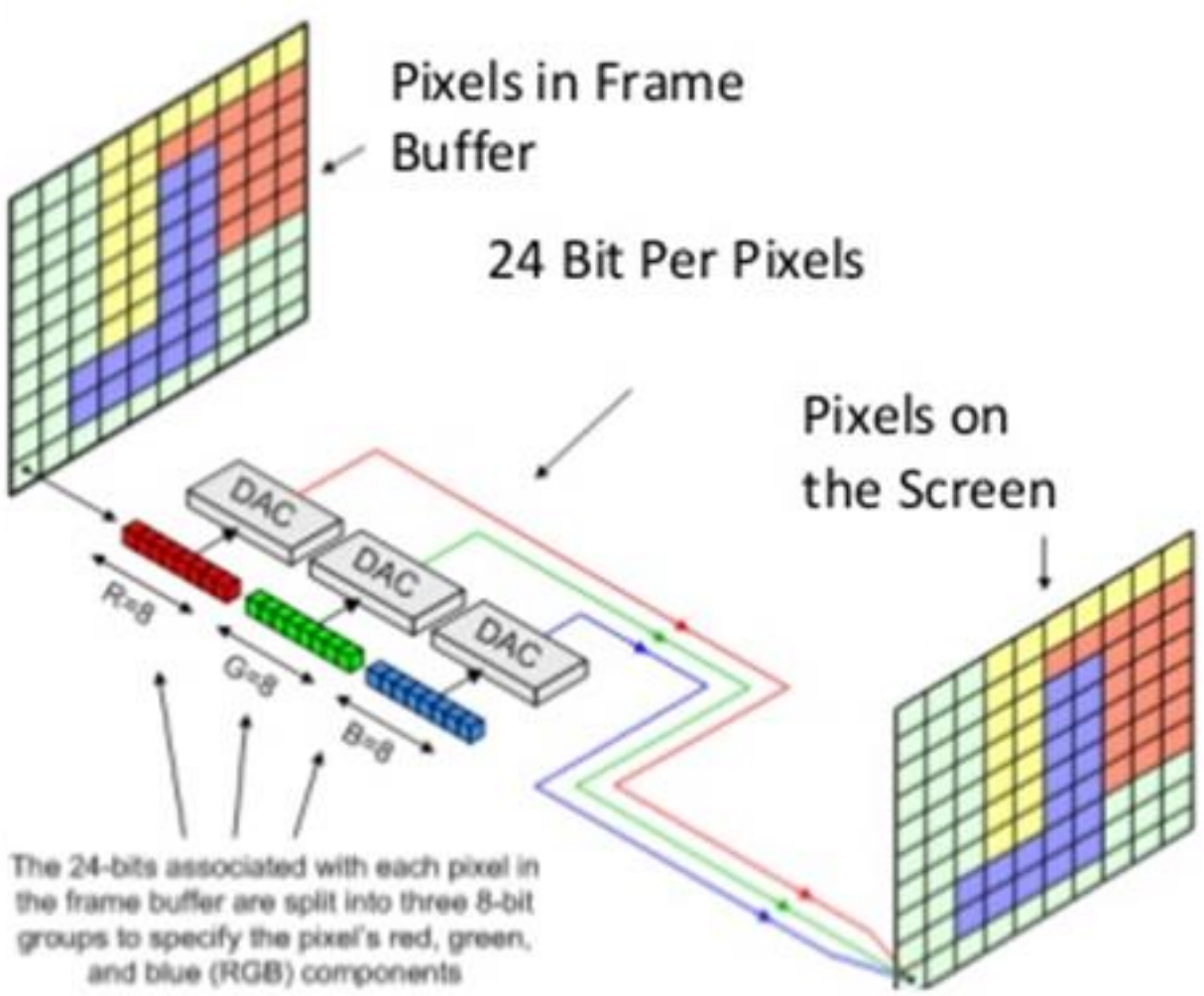


# 1-Bit FrameBuffer



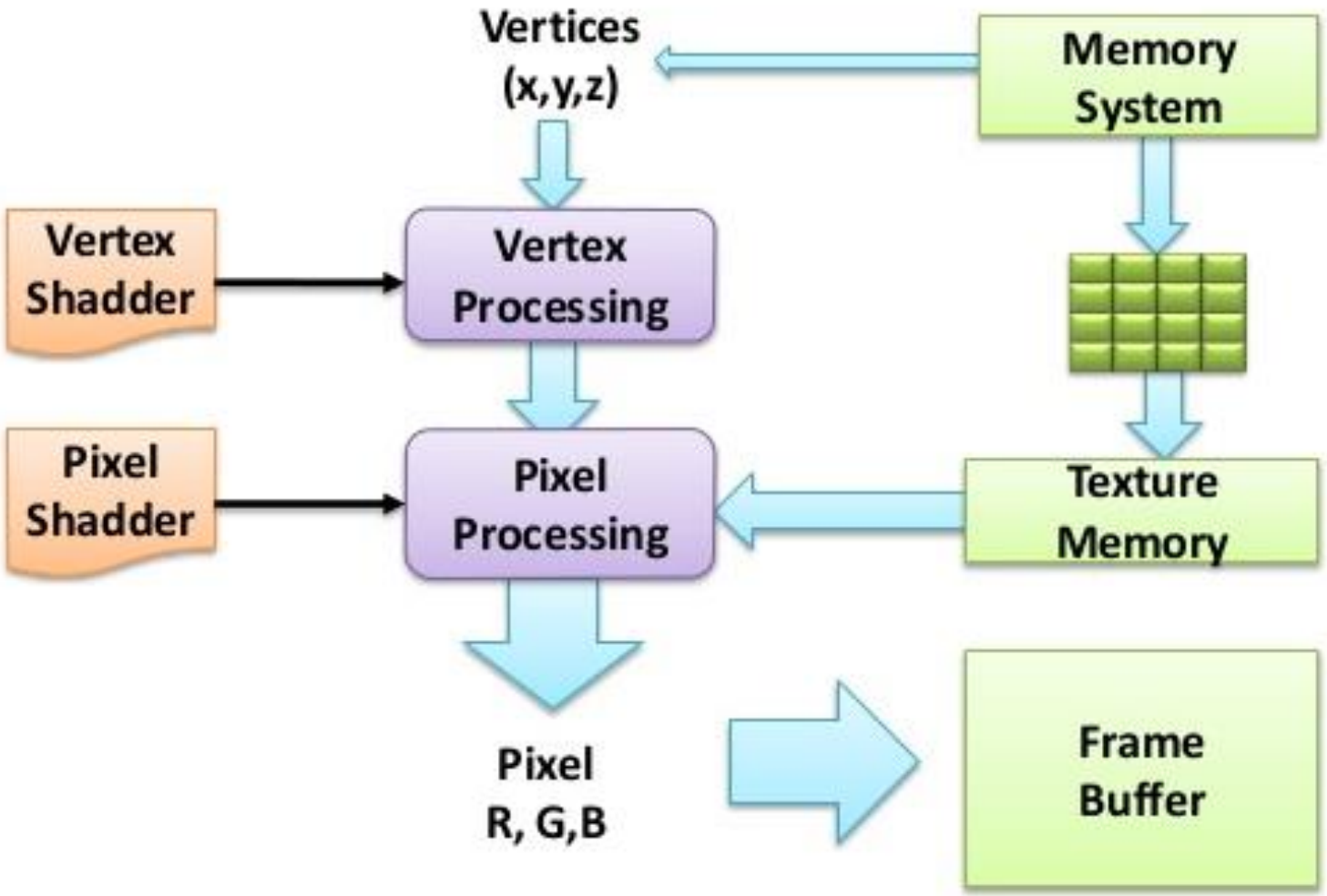

Graphical representation of 1 bit color

# 24-Bit FrameBuffer

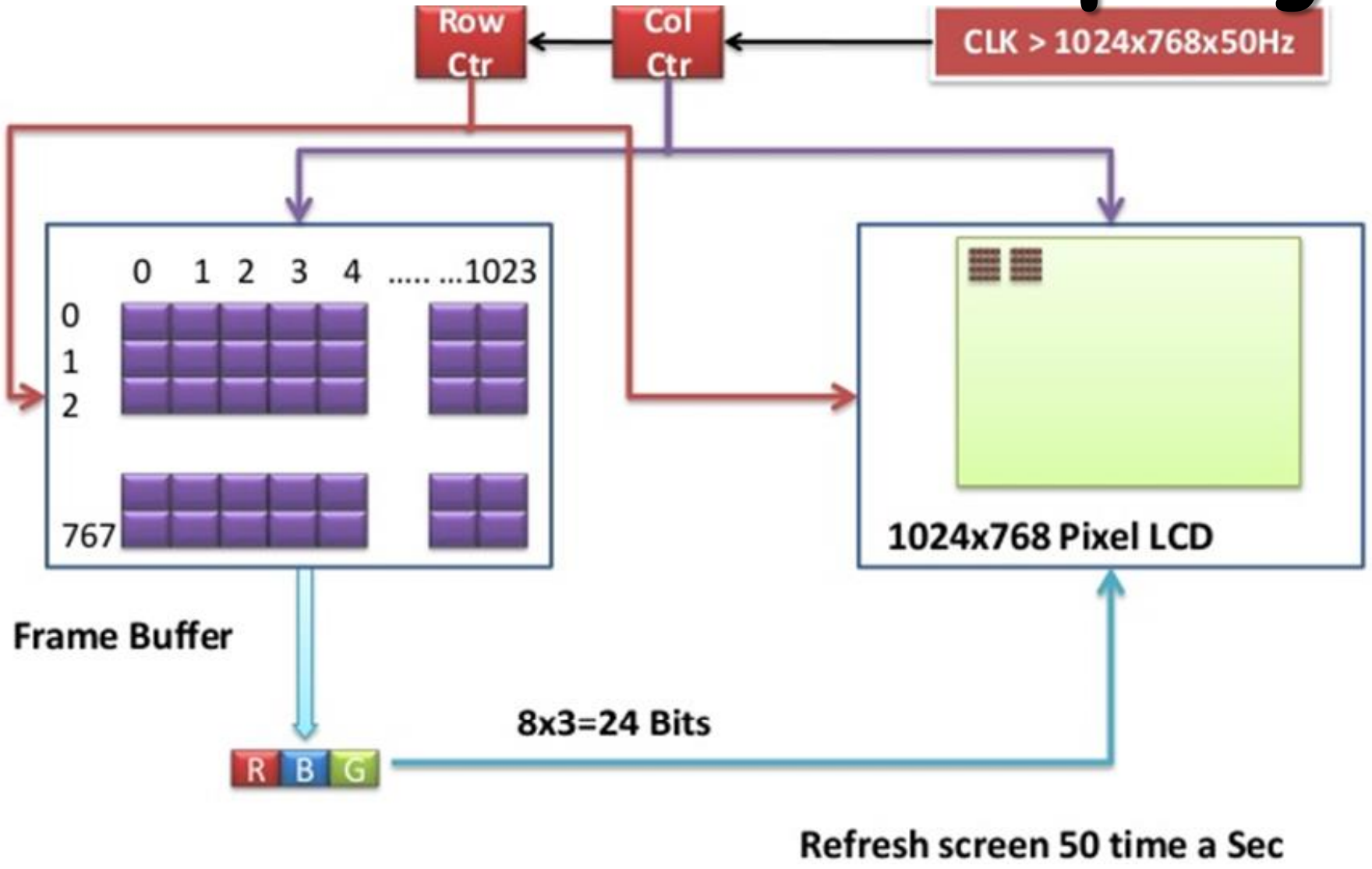


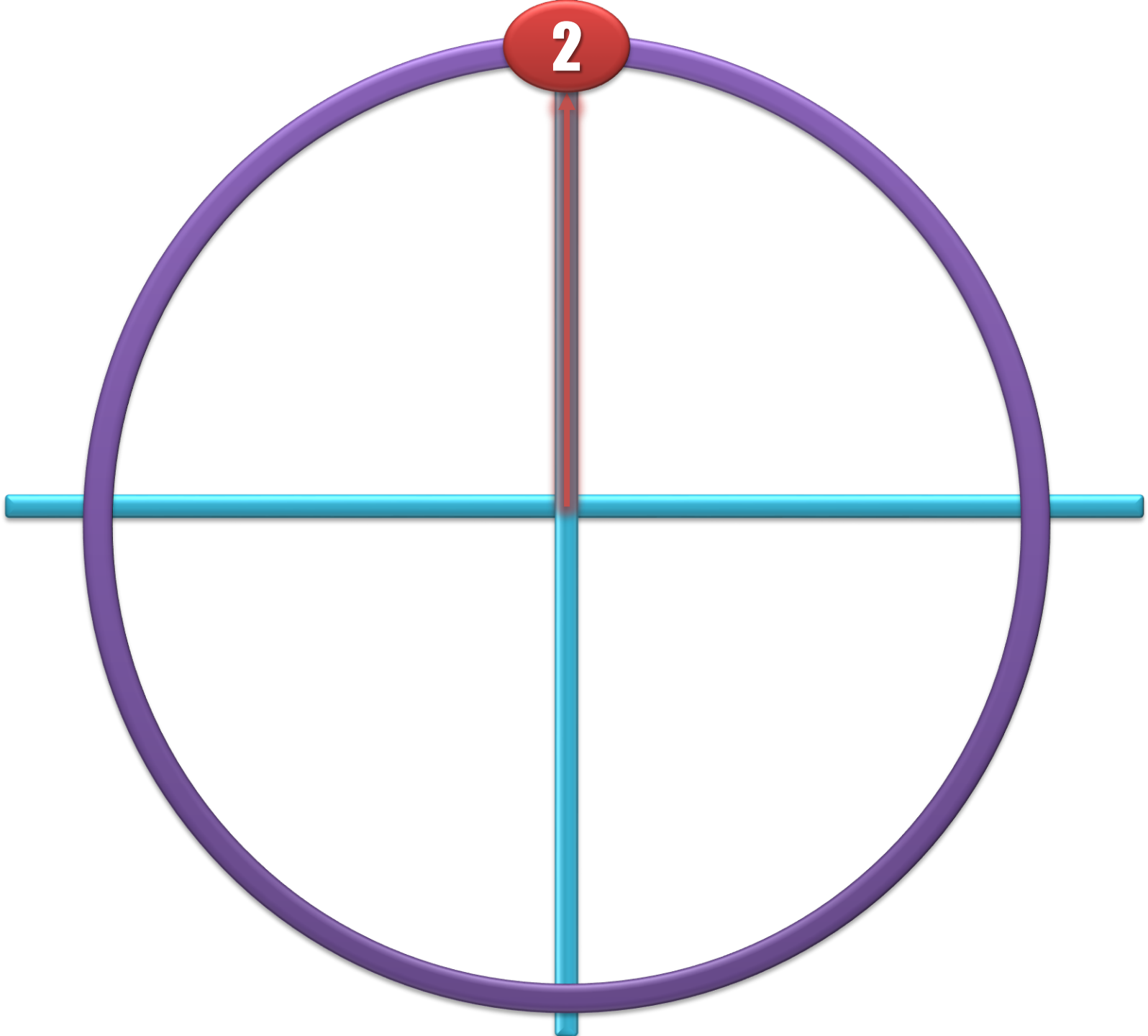

**Graphical representation of 24 bit color**

# Graphics Sub-System

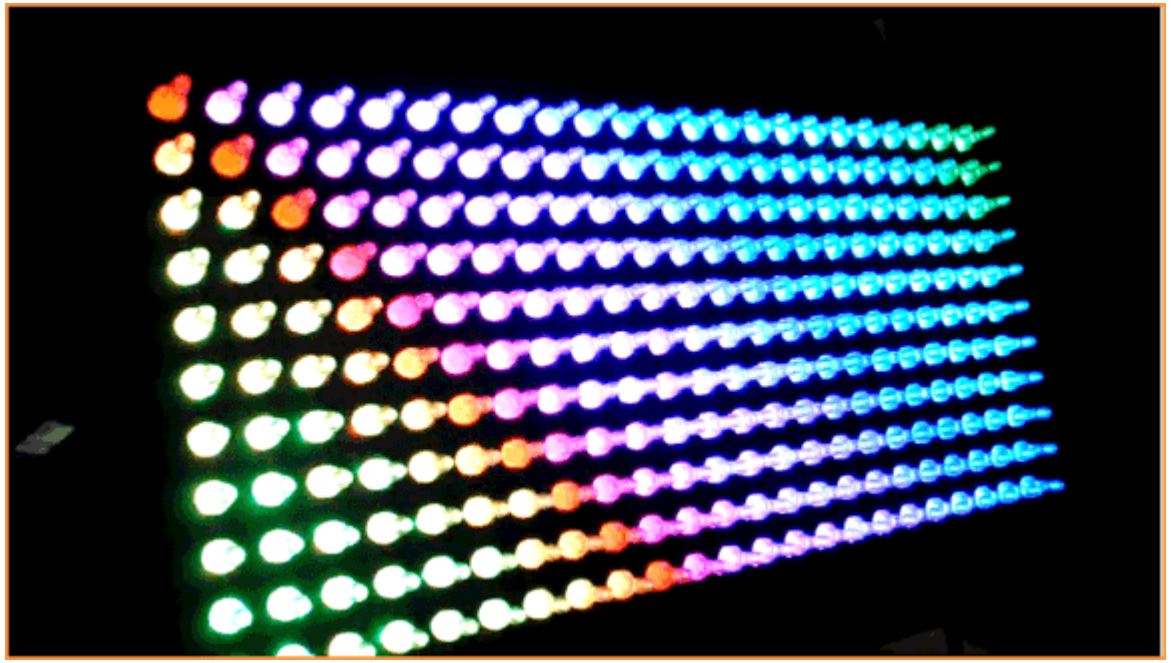


# Framebuffer-Display




2

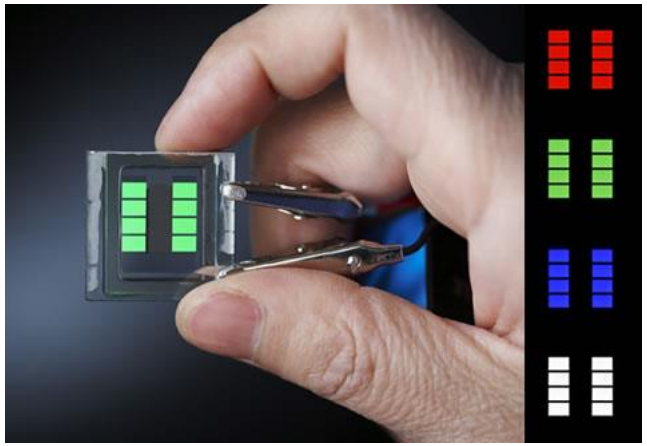
# Displays



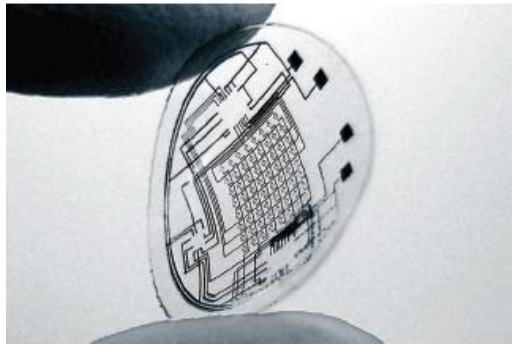
# Large Displays

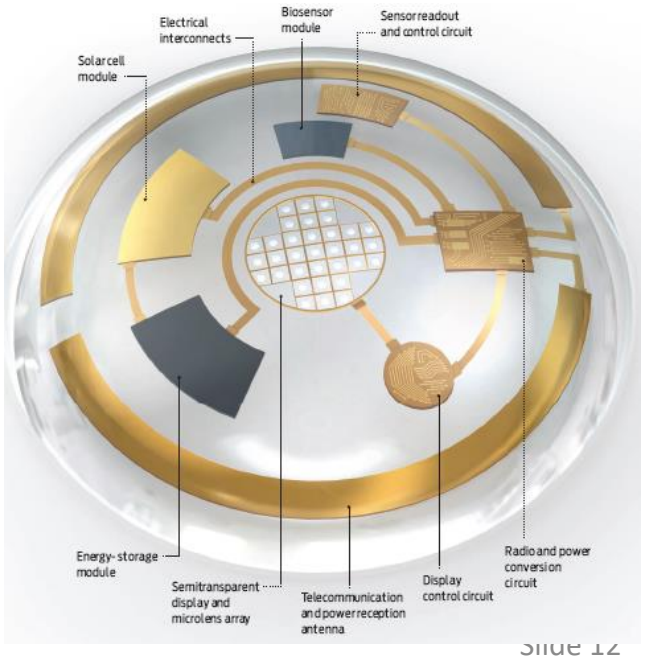

# Small Displays

- Small



- And very small

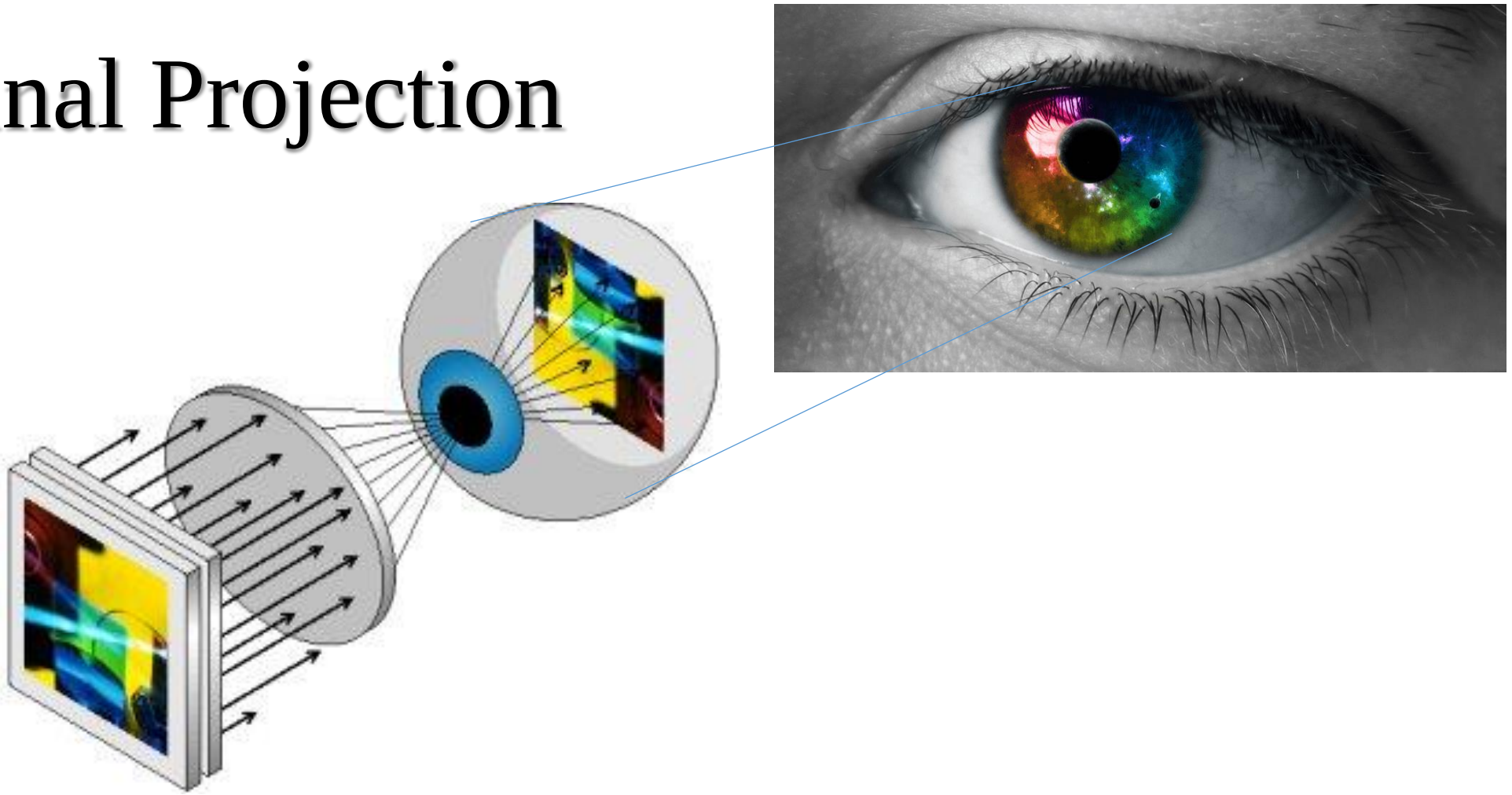





IEEE Spectrum, 2009
Babak A. Parviz
Univ. Washington, Seattle

# The Ultimate Display

- Retinal Projection



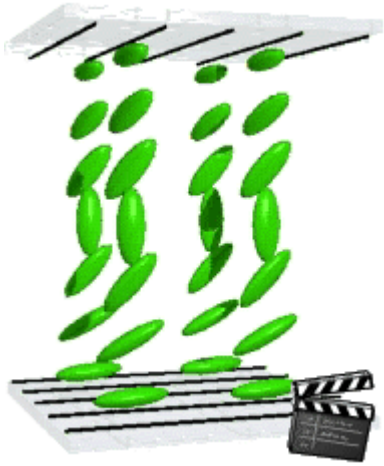
# LCD Displays


- Most common
- High production volume
- Lower costs
- Energy efficient
- Space efficient

# LC = Liquid Crystals

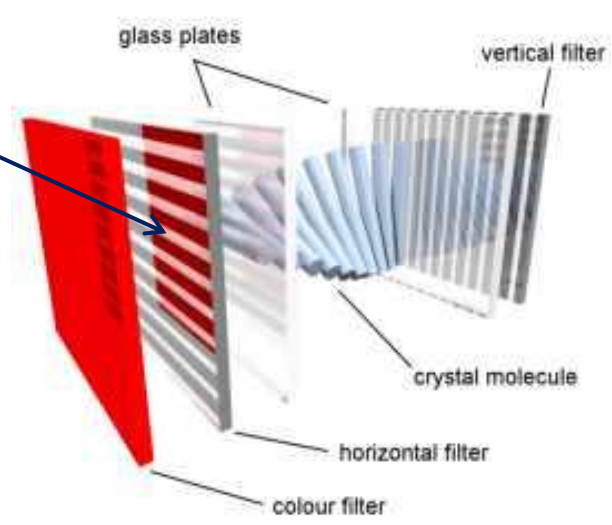
- What are LC's?
  - Substance exhibiting states between a liquid and a solid.
  - Crystals re-align when an electric field is created by applying a voltage between two electrode plates.

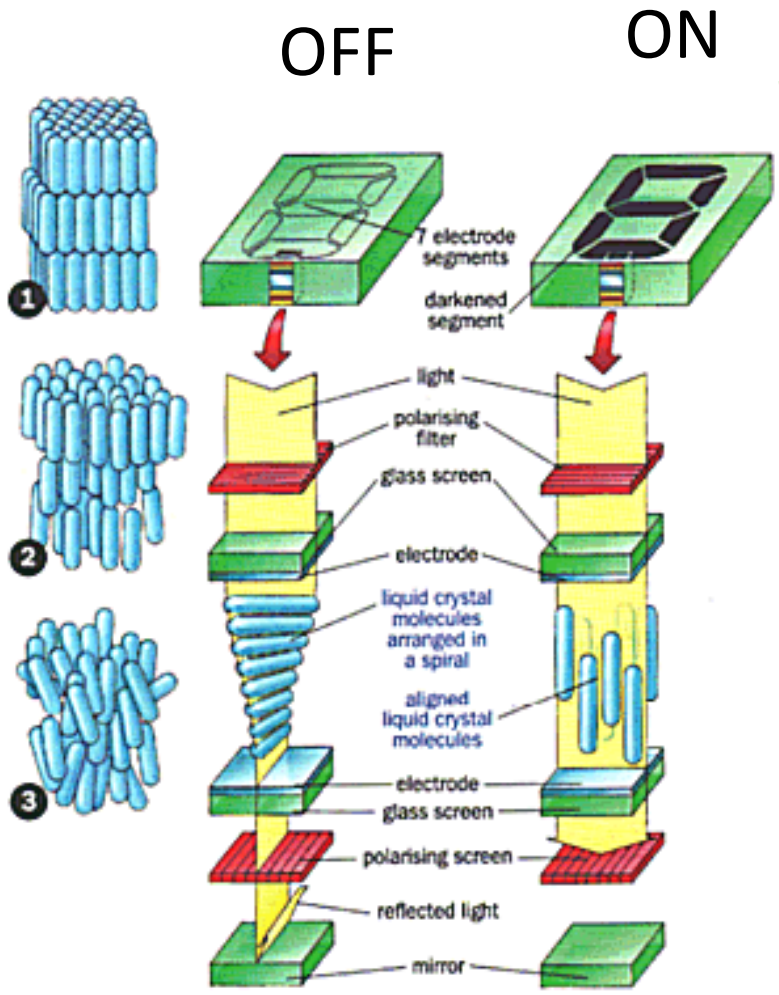
Two different States of LC's



# Basic Principle

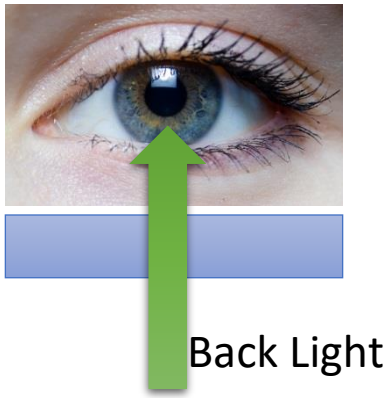
- Nematic  Liquid Crystals
- Natural state is to align with grooves



# Enhancements

- Transmissive = light from a **backlight** passes once thru the pixels out to your eye. Notebook computers use transmissive displays.

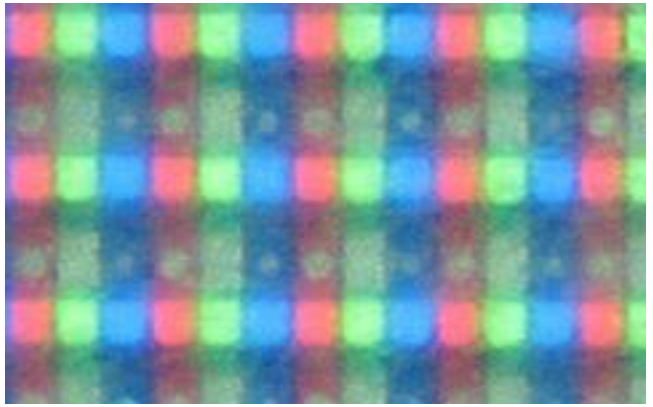



- Reflective = light from the **environment** is reflected at the back of the display and passes back thru the pixels out to your eye.


Transflective = transmissive + reflective.

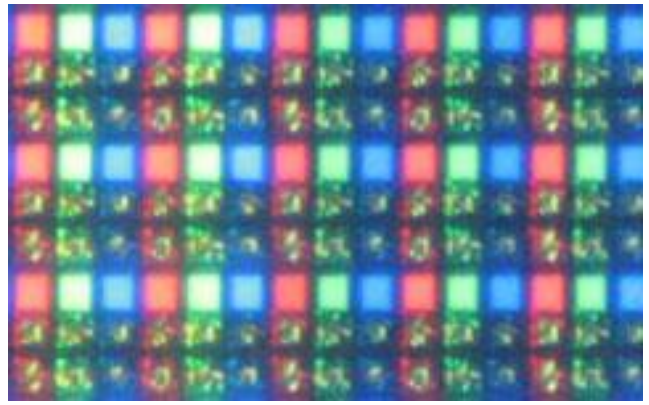
# How it looks…

- Different screens under the microscope.



iPod

2.5” 320 x 240 transflective TFT

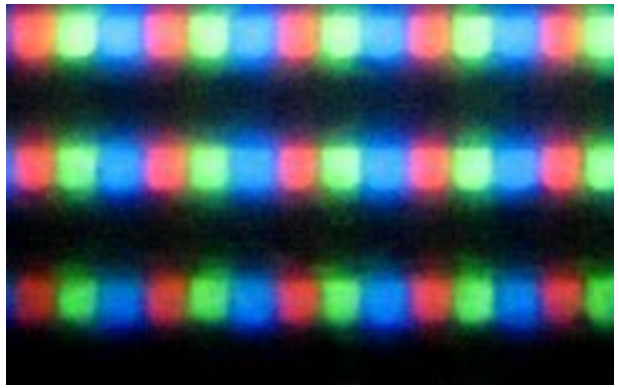


Nokia 6230i

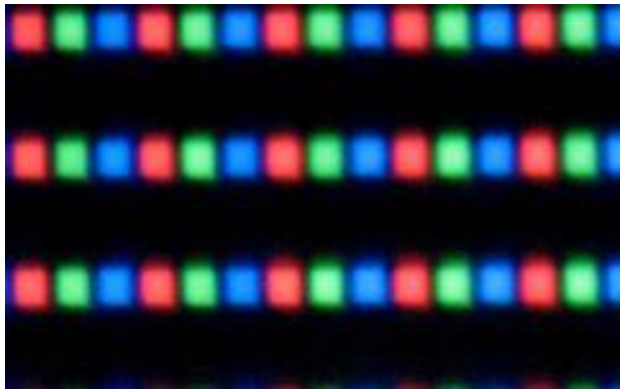
1.7” 208 x 208 transflective TFT

Both TFT LCD displays captured
with a Canon PowerShot SD200 thru a 30X microscope.

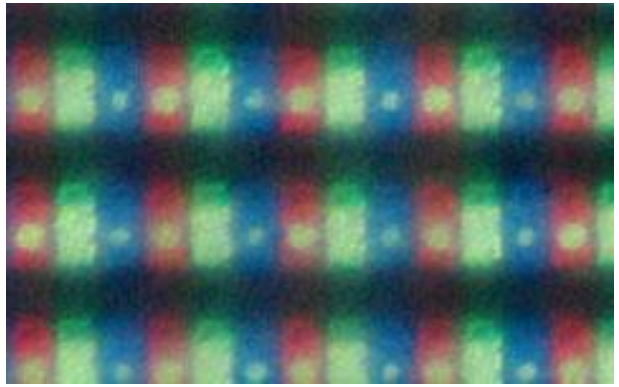
# How it looks…



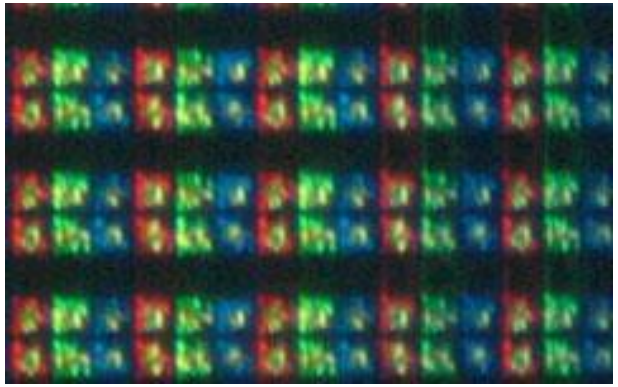
iPod transmissive rows only



Nokia 6230i transmissive only



iPod reflective rows only



Nokia 6230i reflective only

# Transflective TFT Display

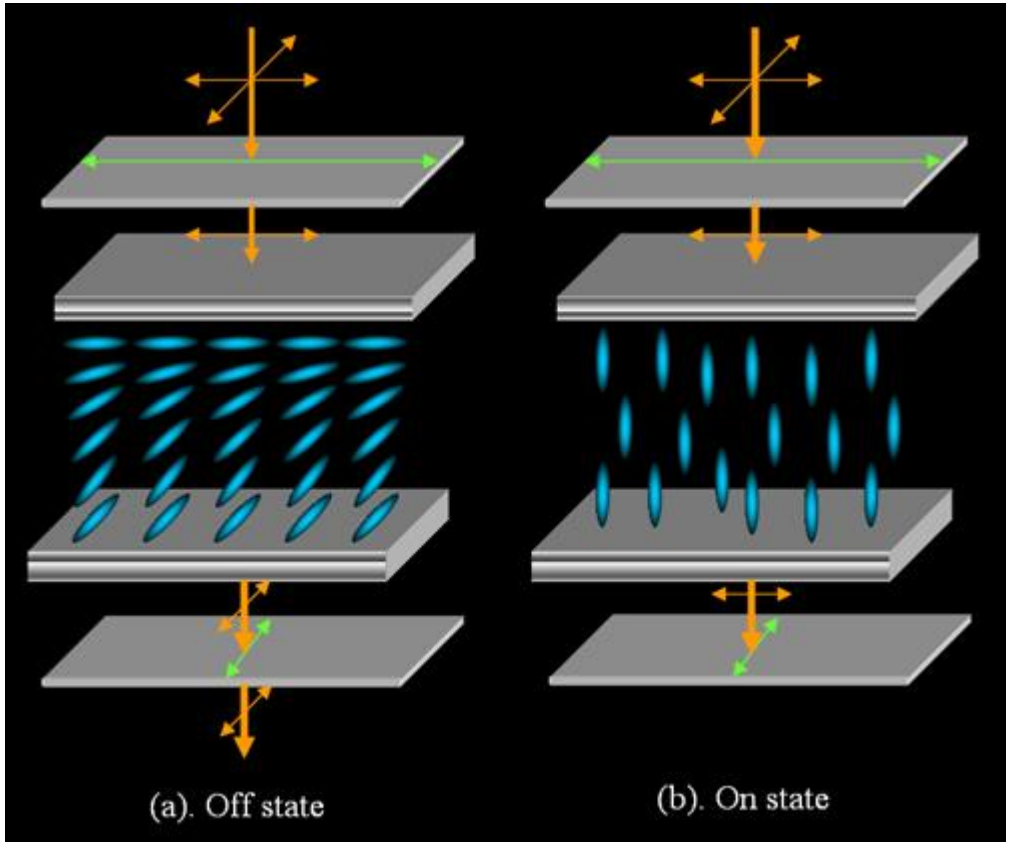
- Employ two complimentary display techniques to improve visibility across a wider range of ambient light conditions.
- Each pixel has separate transmissive and reflective areas can be seen in the close-up views of a single pixel.

# LCD Types

1. TN: Twisted Nematic (1971)
2. MVA: Multi-domain Vertical Alignment (1999)
3. IPS: In-Plane Switch (1992-2002)

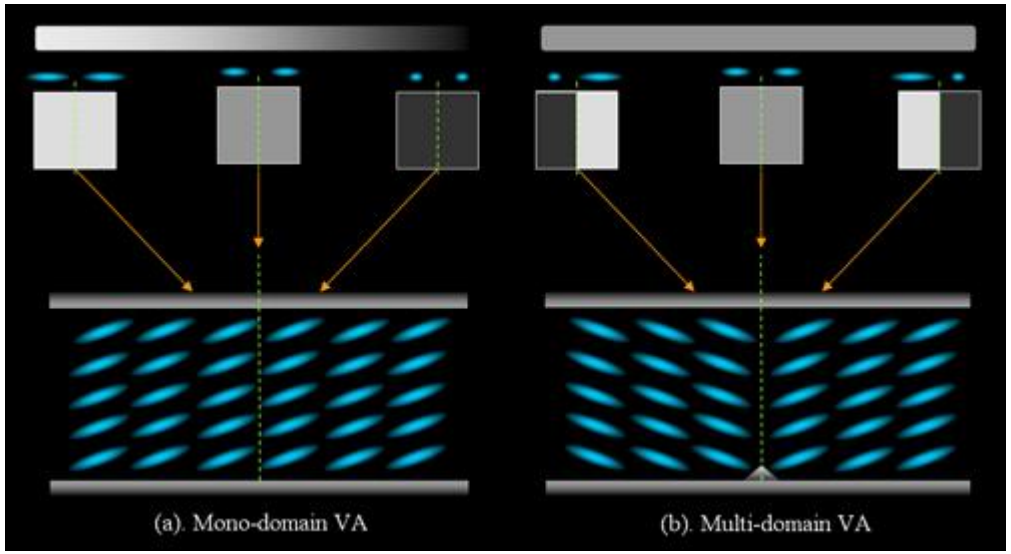
# Twisted Nematic

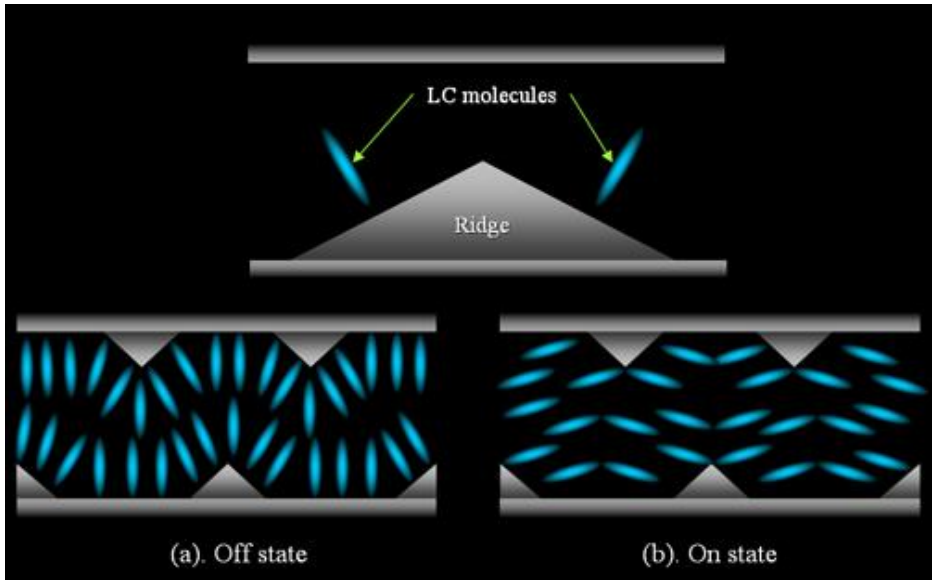
- TN first introduced Schadt, Helfrich and Fergason in 1971
- Traditional LCD



# Multi-domain Vertical Alignment

1999 Fujitsu.

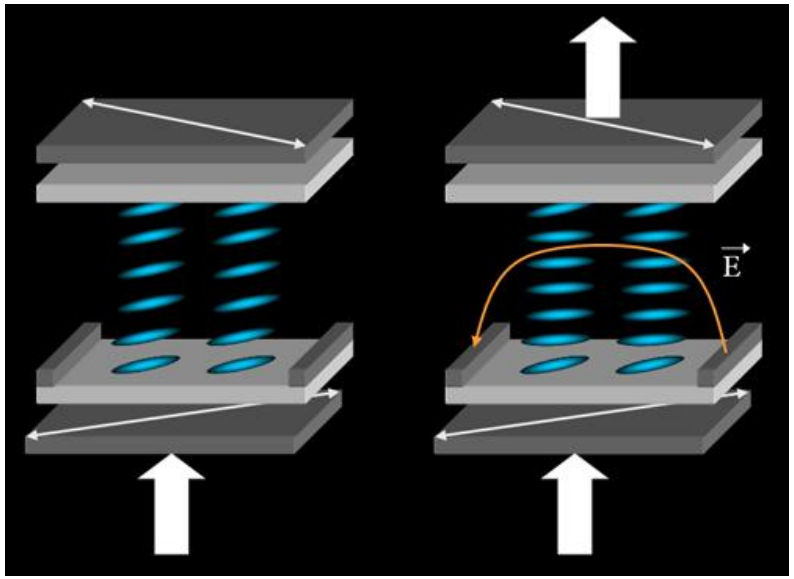



Can give very good viewing angles
Simpler manufacturing – no rubbing

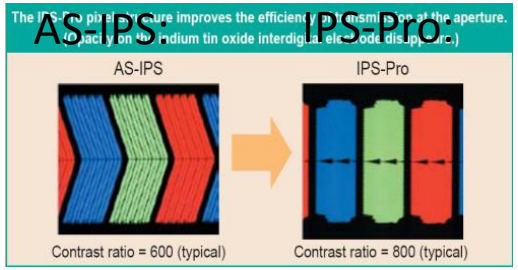
# In-Plane Switching

- Voltage is applied in the same plane as the substrate
- Hitachi 1992 and 2002.




- High Image Quality
- Very good contrast
- High brightness
- Wide aperture
- Variations:

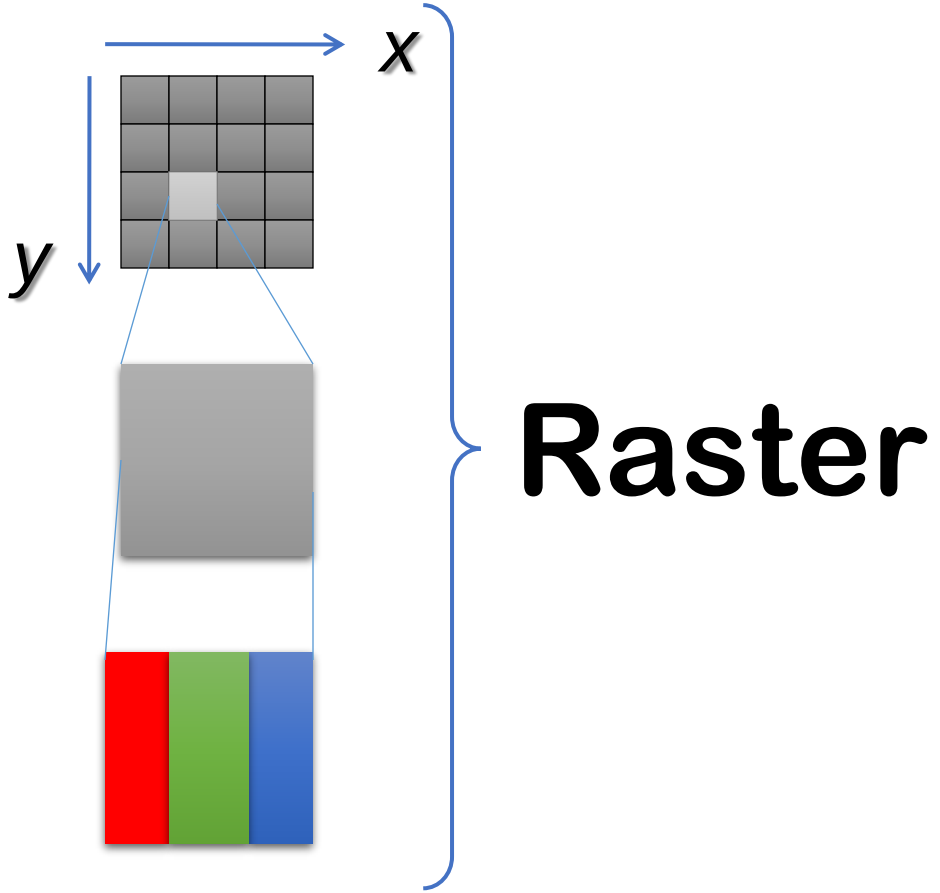
AS-IPS: IPS-Pro:

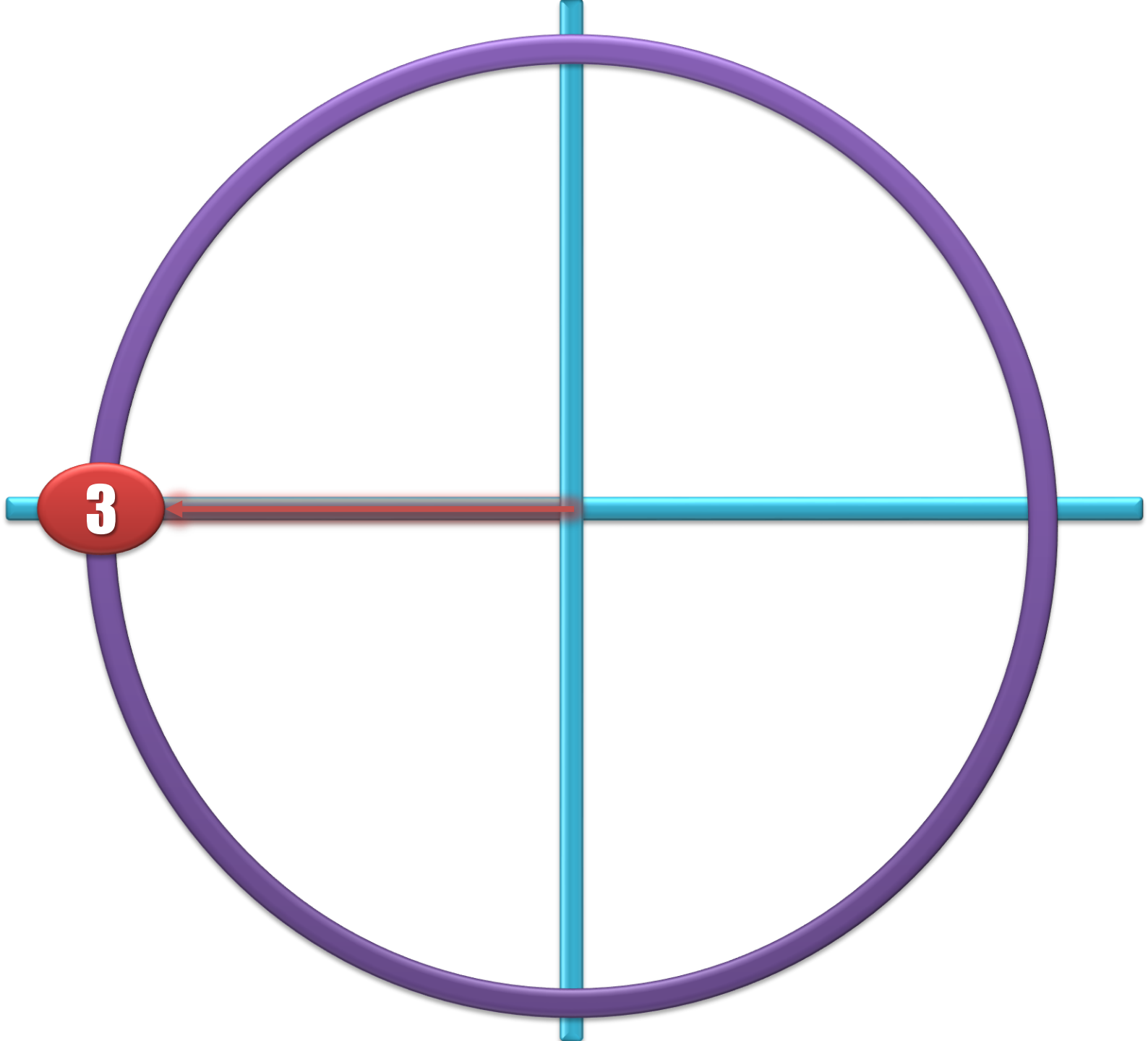


# In Summary…

# All FP Displays: Raster

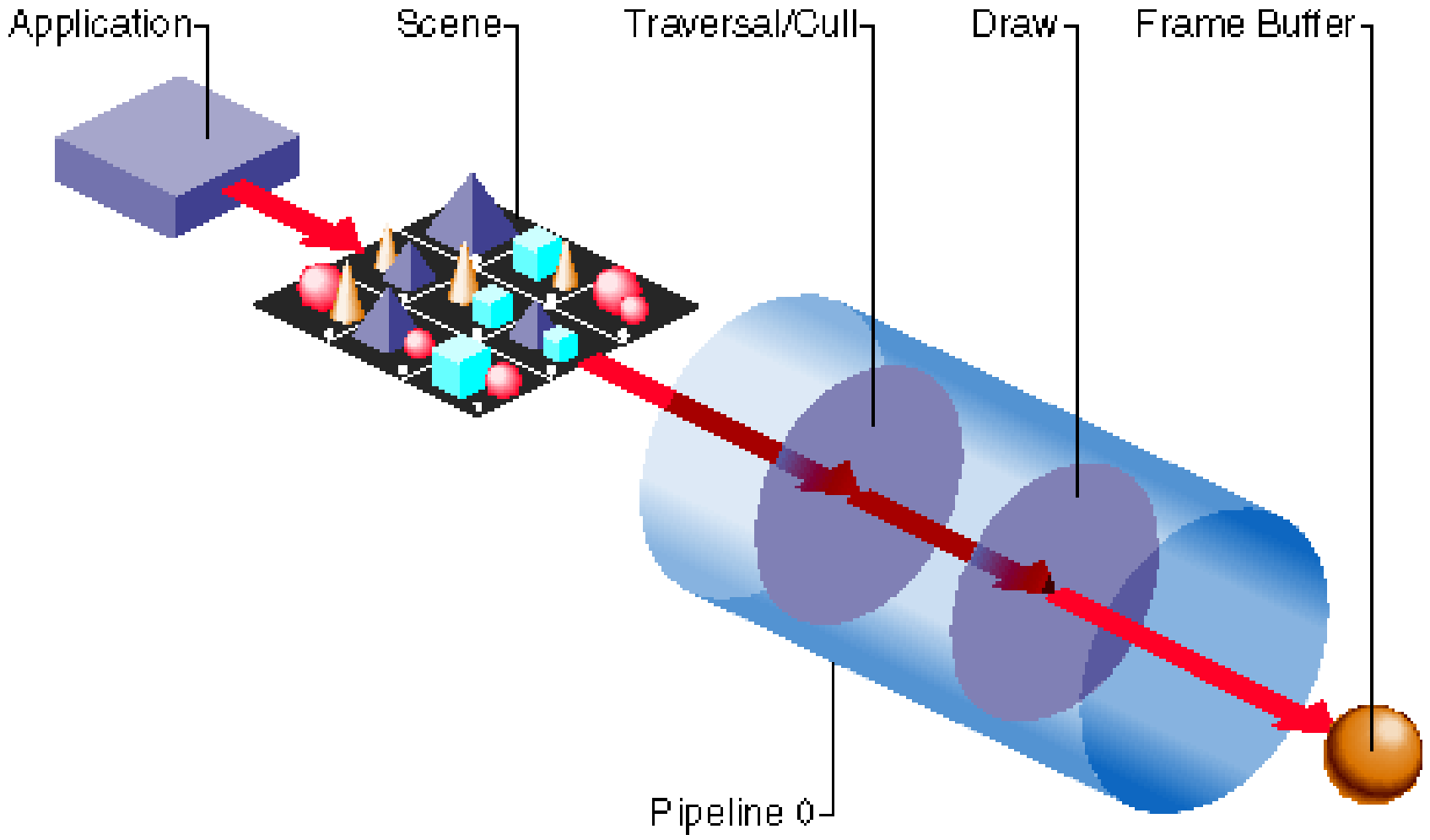
- All flat panel displays have the same basic characteristics:
  - Active Matrix = address
  - Picture Element = Pixel
  - Sub-pixel elements: R, G, B



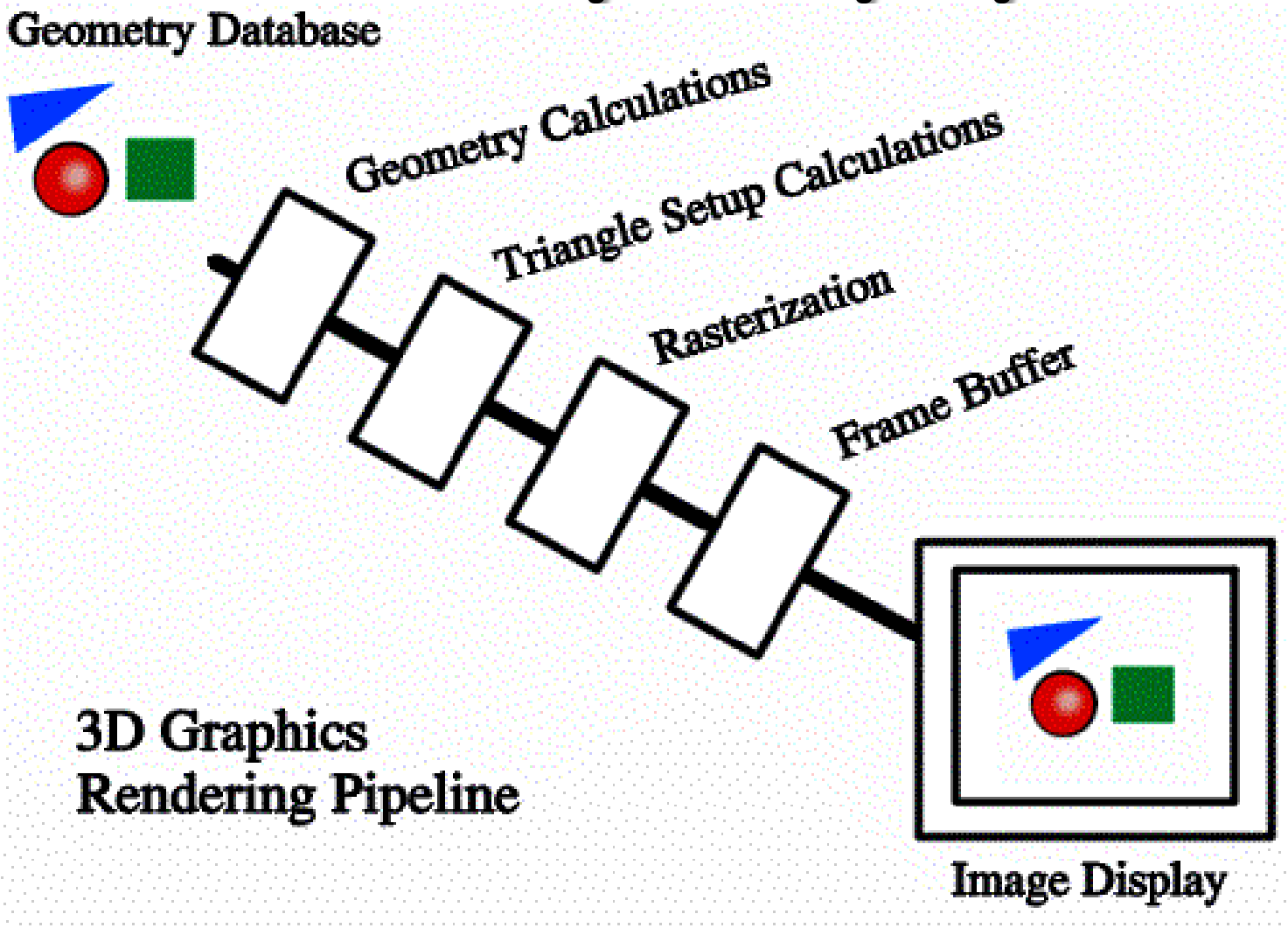

3

# GPUs

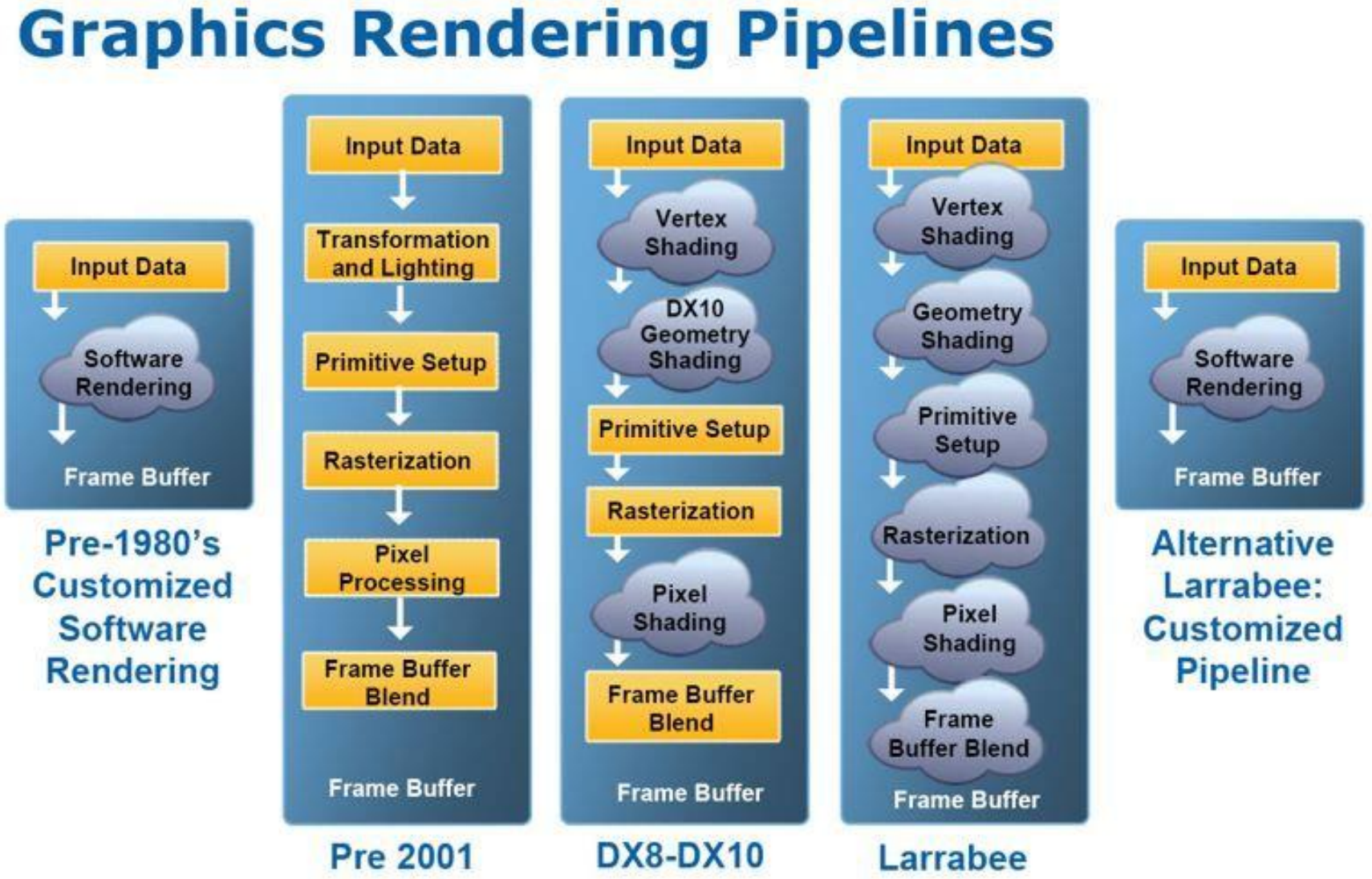
# Rendering Pipeline



# Inside a simple pipeline



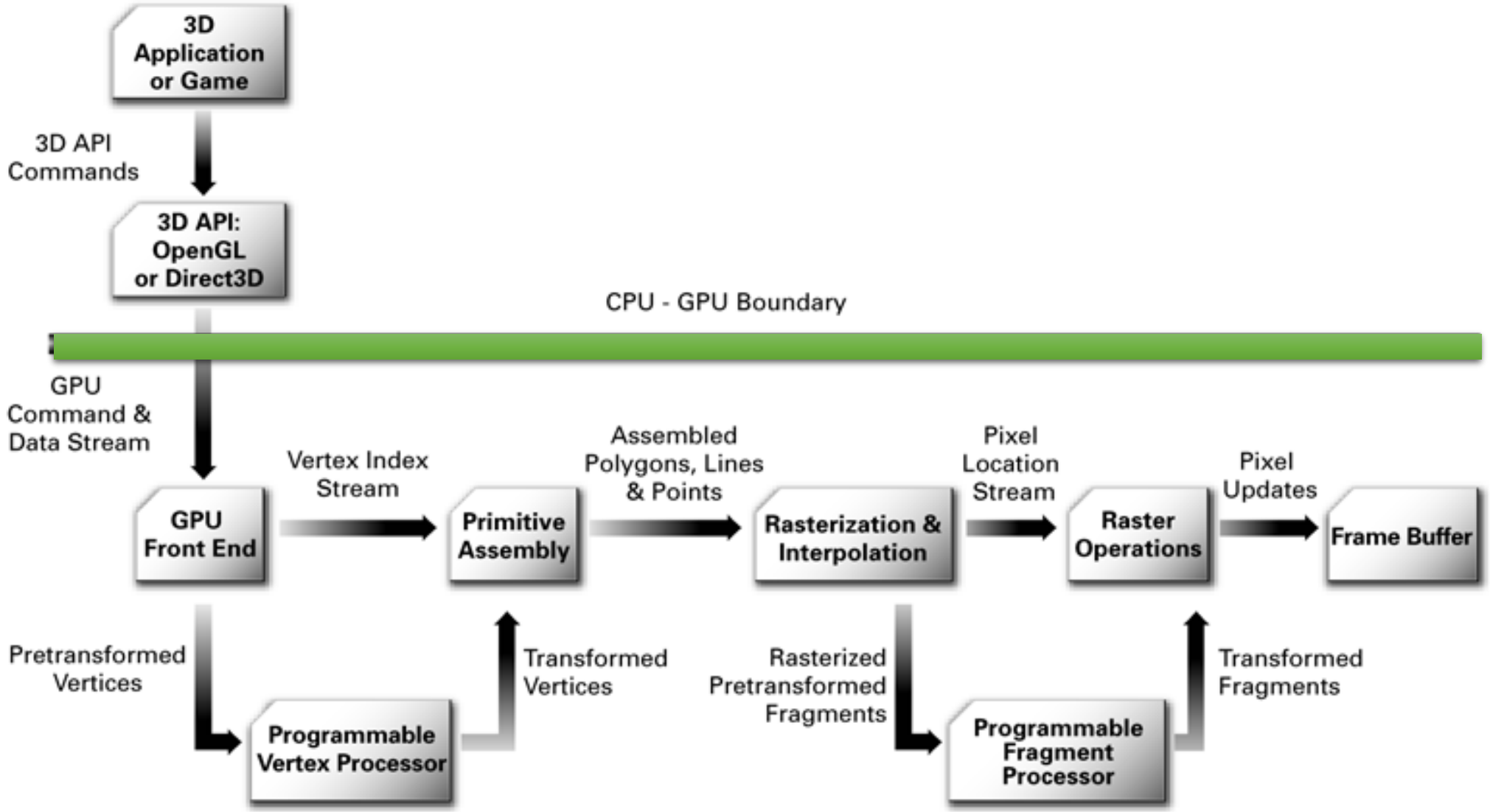
# Short History




SGI

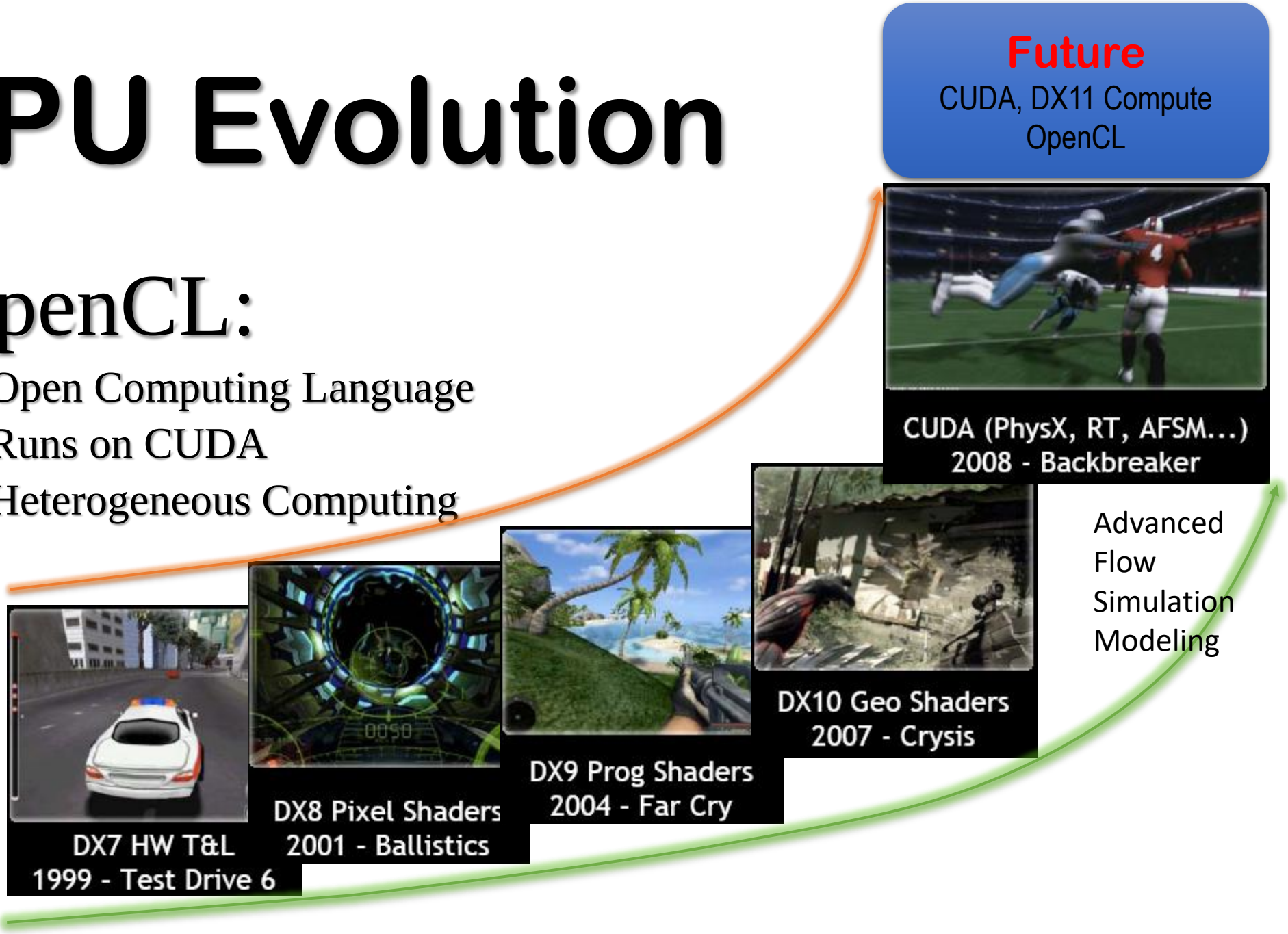
Sun Microsystems

# CPU-GPU Boundary



# GPU Evolution

- OpenCL:
  - Open Computing Language
  - Runs on CUDA
  - Heterogeneous Computing



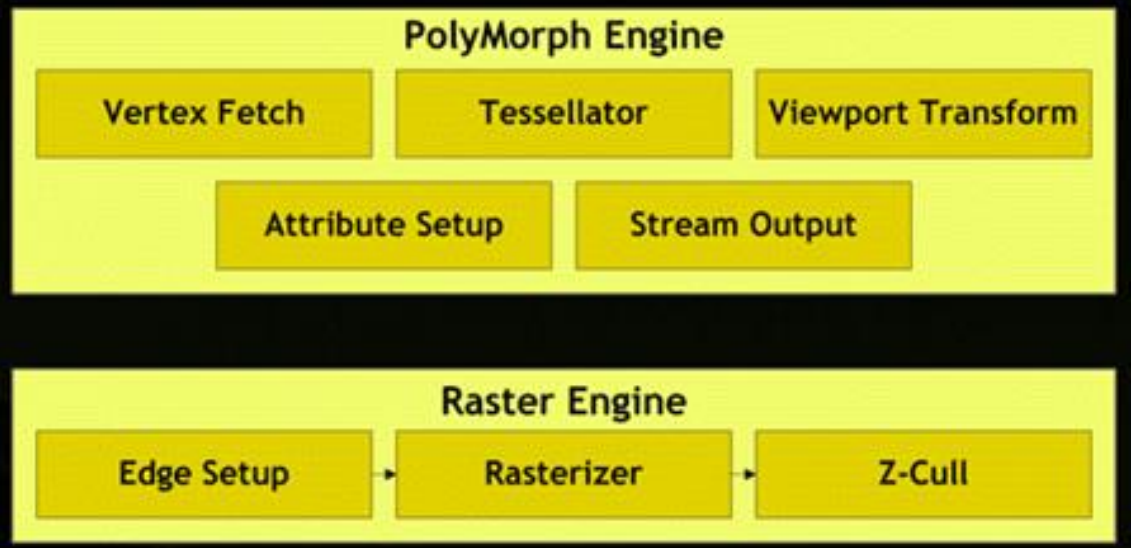
# Example GPU: NVIDIA Fermi

- NVIDIA Fermi GF100:
  - 512 CUDA stream processors
  - 16 Geometry Units
  - 4 Raster Units
  - 64 Texture Units,
  - 40nm fabrication process
  - 3.2 billion transistors
  - 700MHz core clock
  - 1.5GB GDDR5 memory
  - 384-bit memory interface
  - Up to 406W power consumption
  - 98° C

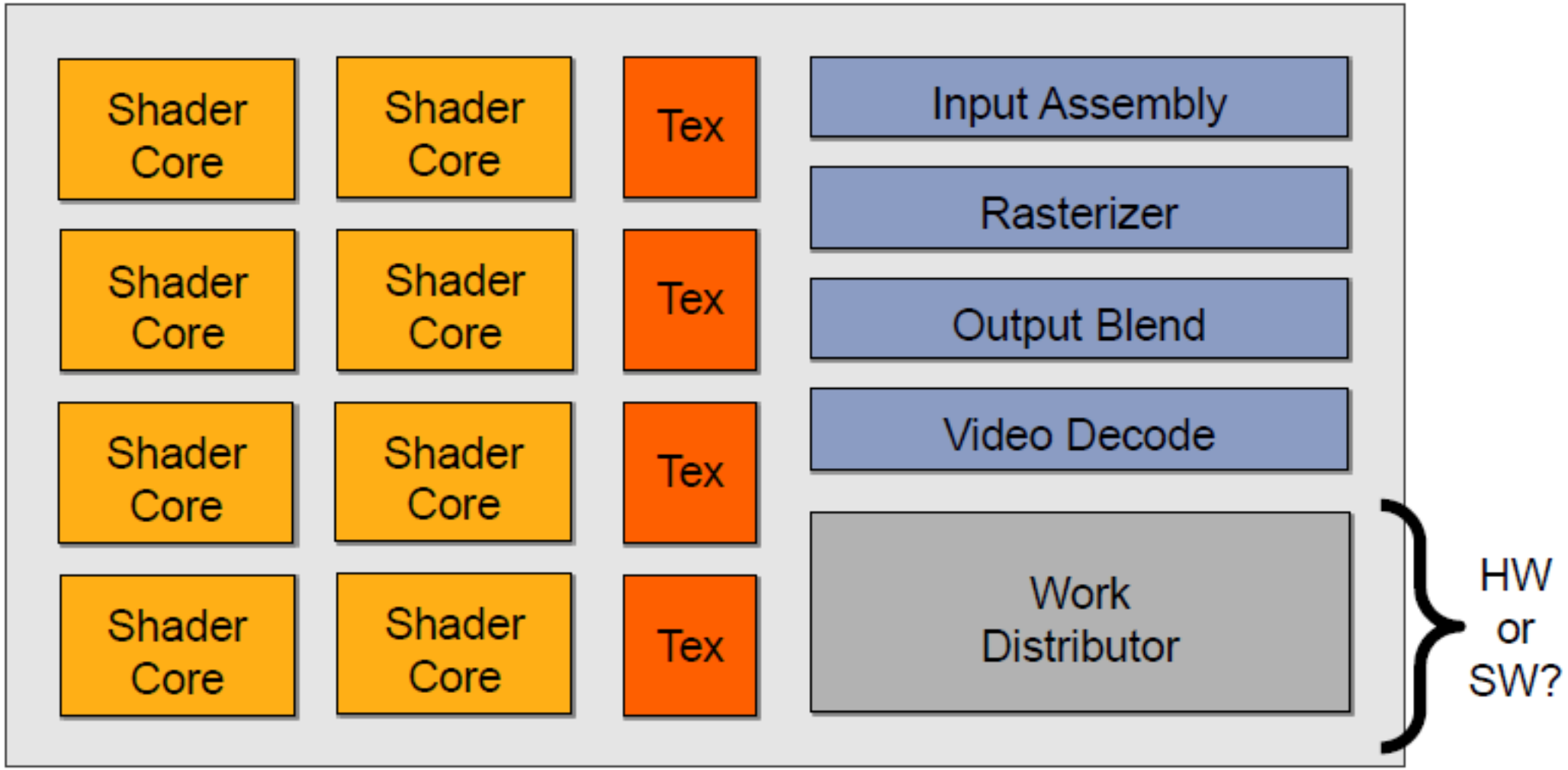
# PolyMorph Engine

## New GF100 PolyMorph and Raster Engines

- Dedicated graphics hardware
- PolyMorph Engine
  - World space processing
- Raster Engine
  - Screen space processing



# GPUs Basics


Shader Core
Shader Core
Tex
Shader Core
Shader Core
Tex
Shader Core
Shader Core
Tex
Shader Core
Shader Core
Tex
Input Assembly
Rasterizer
Output Blend
Video Decode
Work Distributor
HW or SW?

# A diffuse reflectance shader

```
sampler mySamp;
Texture2D<float3> myTex;
float3 lightDir;

float4 diffuseShader(float3 norm, float2 uv)
{
  float3 kd;
  kd = myTex.Sample(mySamp, uv);
  kd *= clamp( dot(lightDir, norm), 0.0, 1.0);
  return float4(kd, 1.0);
}
```

Shader programming model:

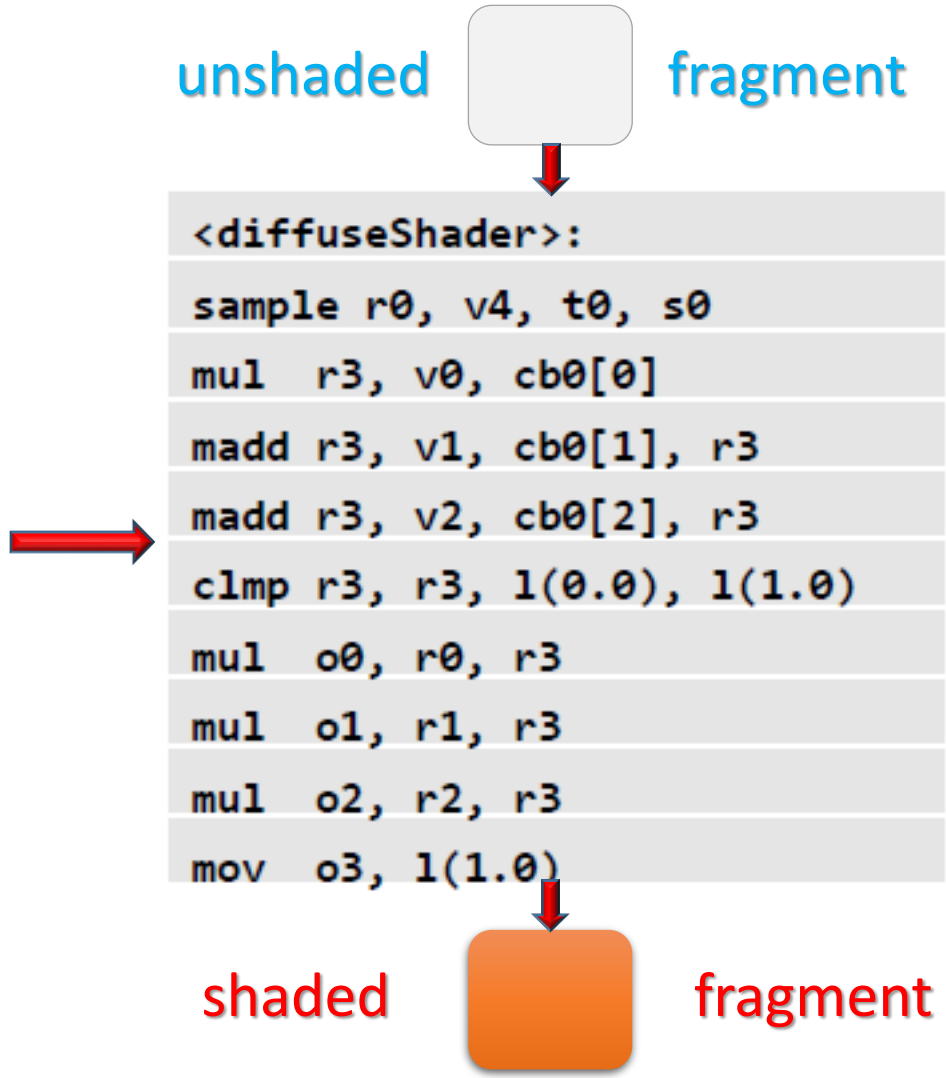
Fragments are processed independently

But, there is no explicit parallel programming

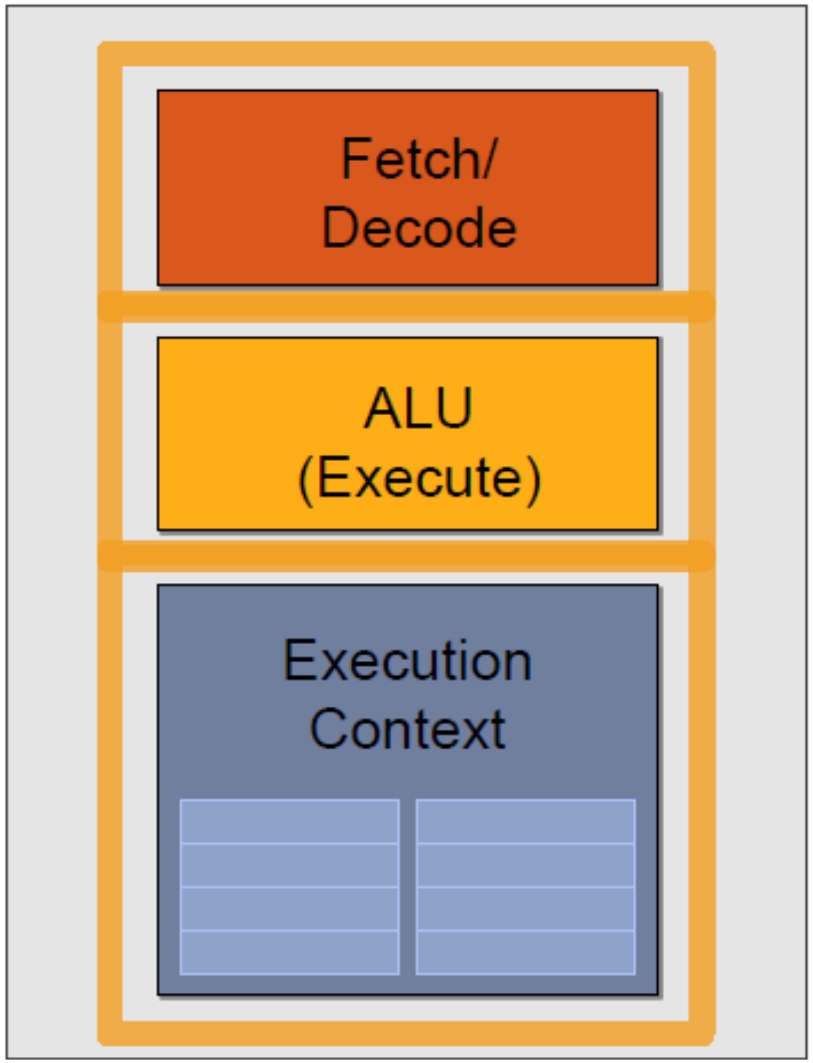
# Compile the shader program

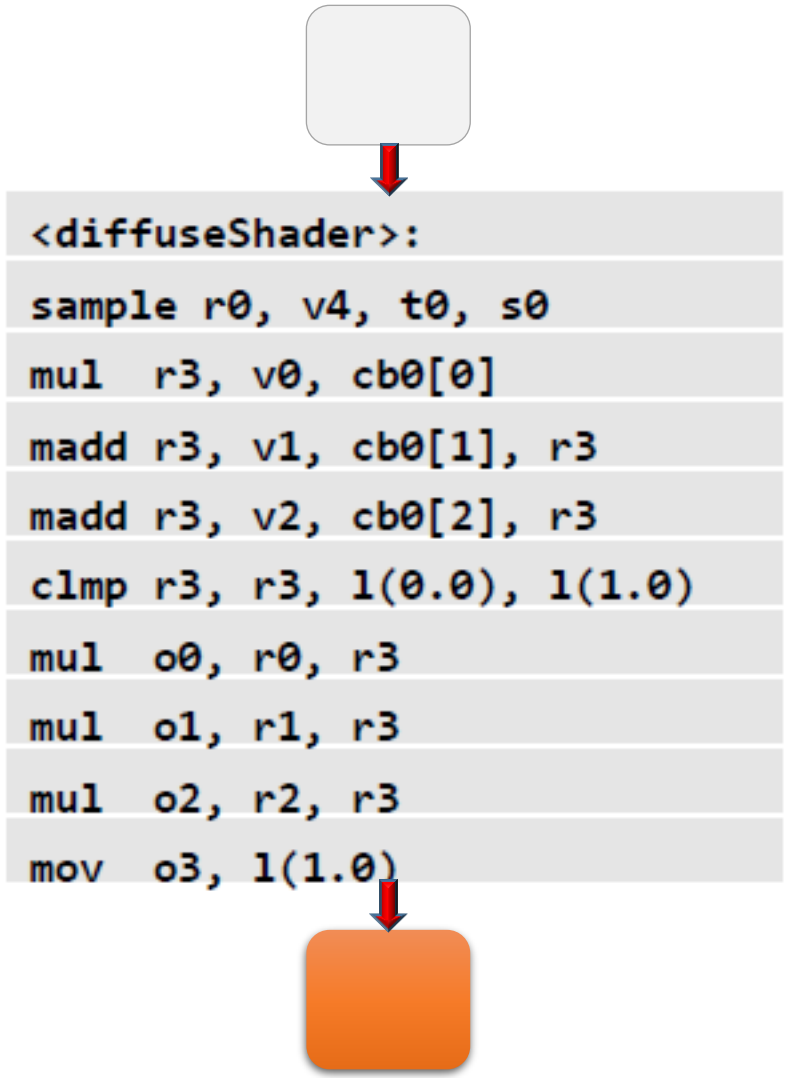
```
sampler mySamp;
Texture2D<float3> myTex;
float3 lightDir;

float4 diffuseShader(float3 norm, float2 uv)
{
  float3 kd;
  kd = myTex.Sample(mySamp, uv);
  kd *= clamp( dot(lightDir, norm), 0.0, 1.0);
  return float4(kd, 1.0);
}
```

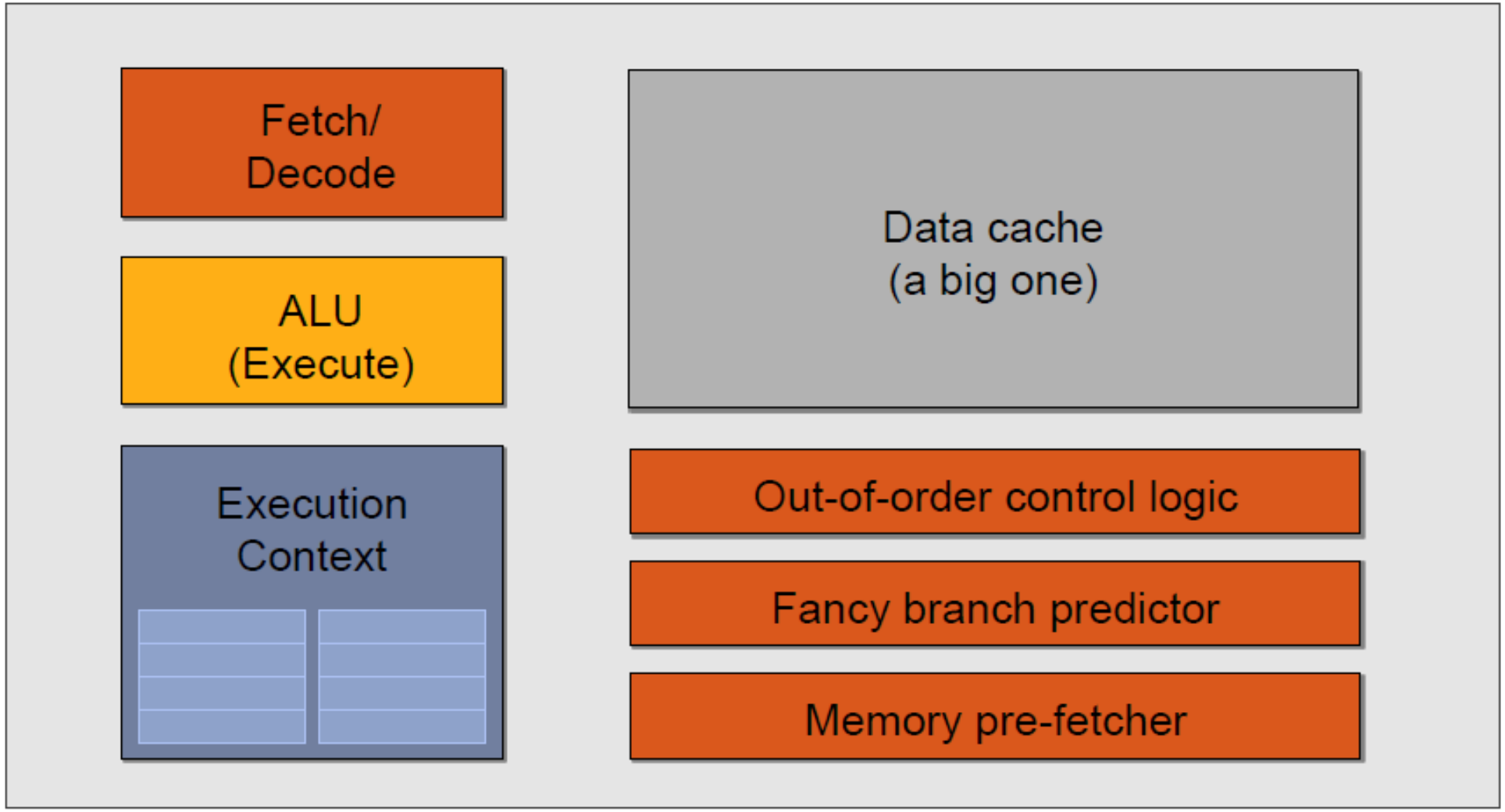


# Execute the shader

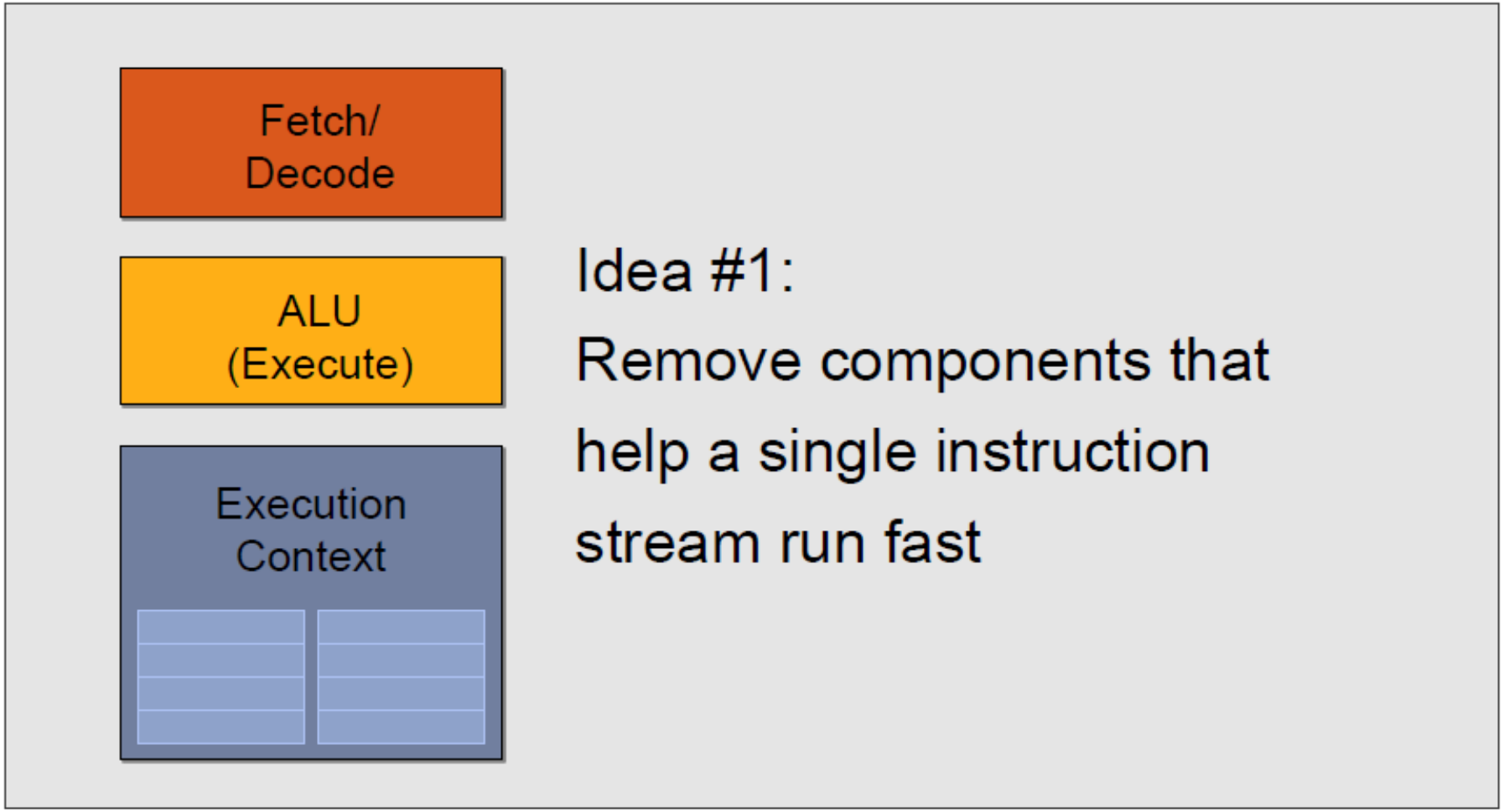


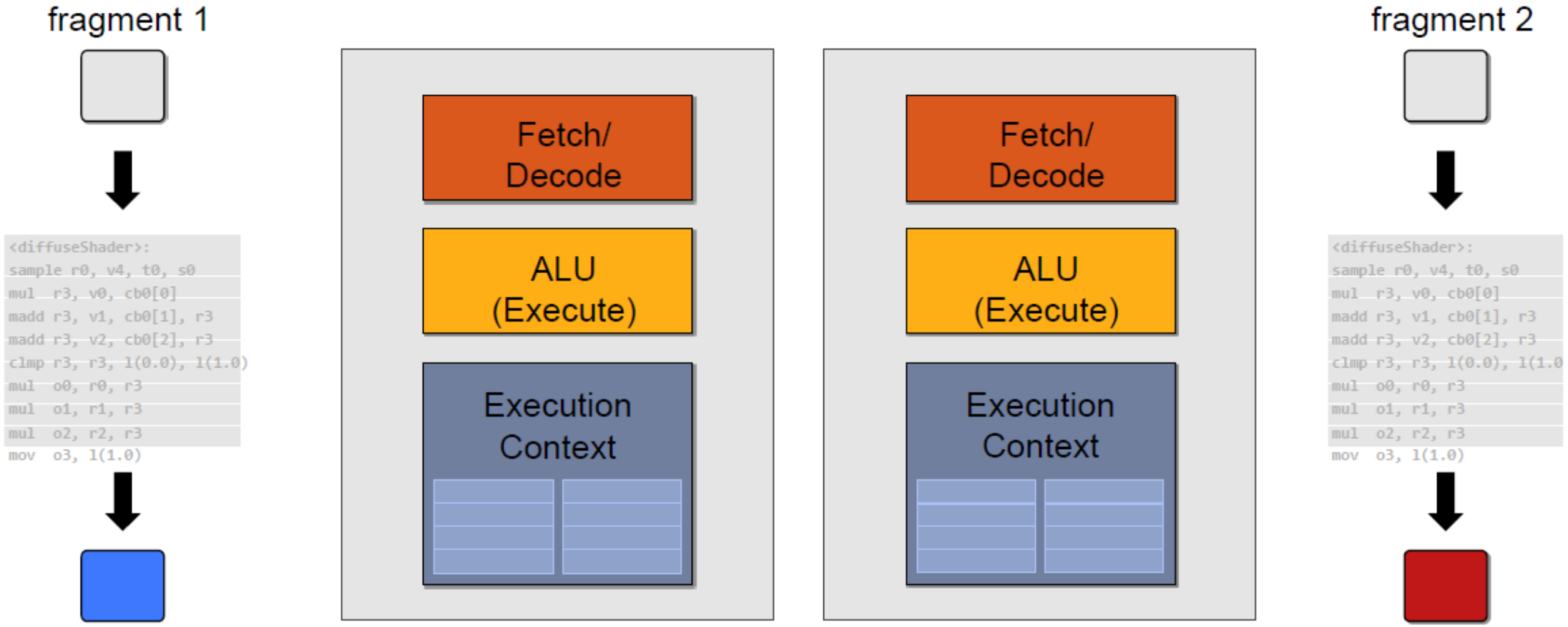
# CPU core style execution



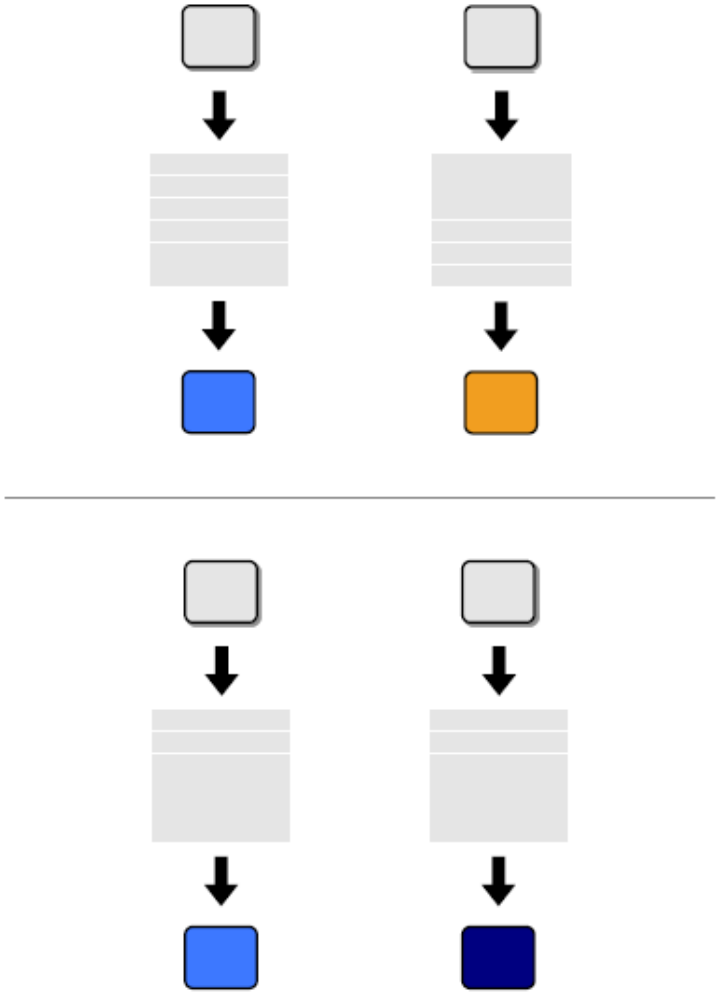
# Slimming down

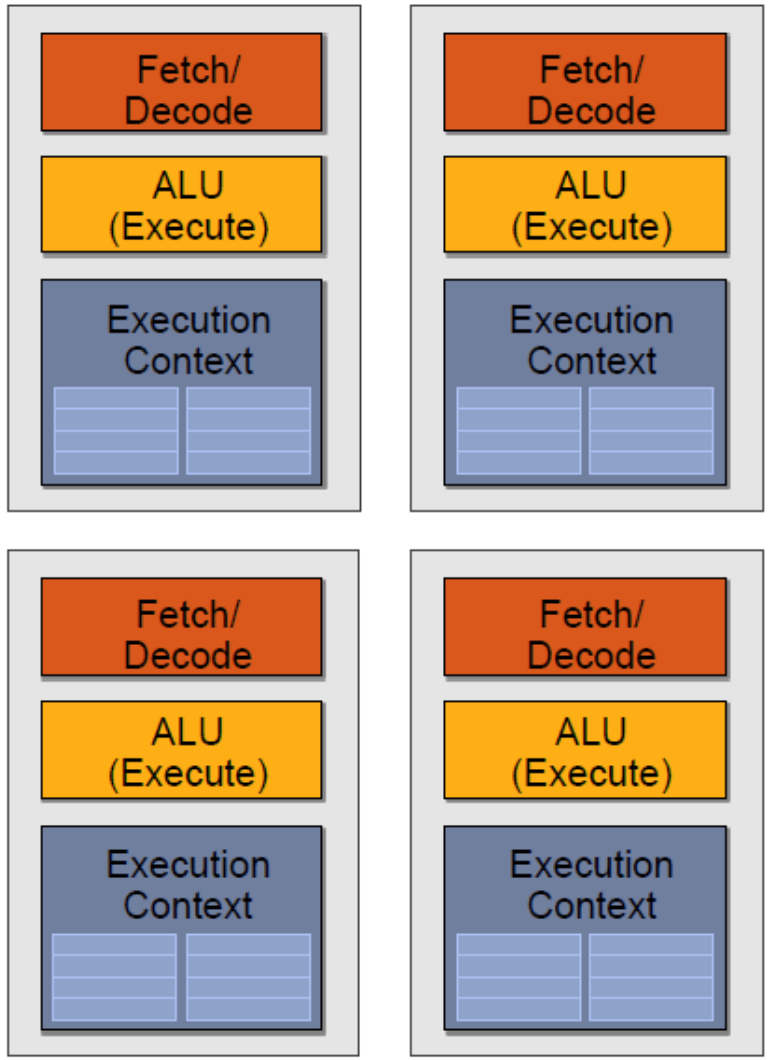


# Two core – two fragments

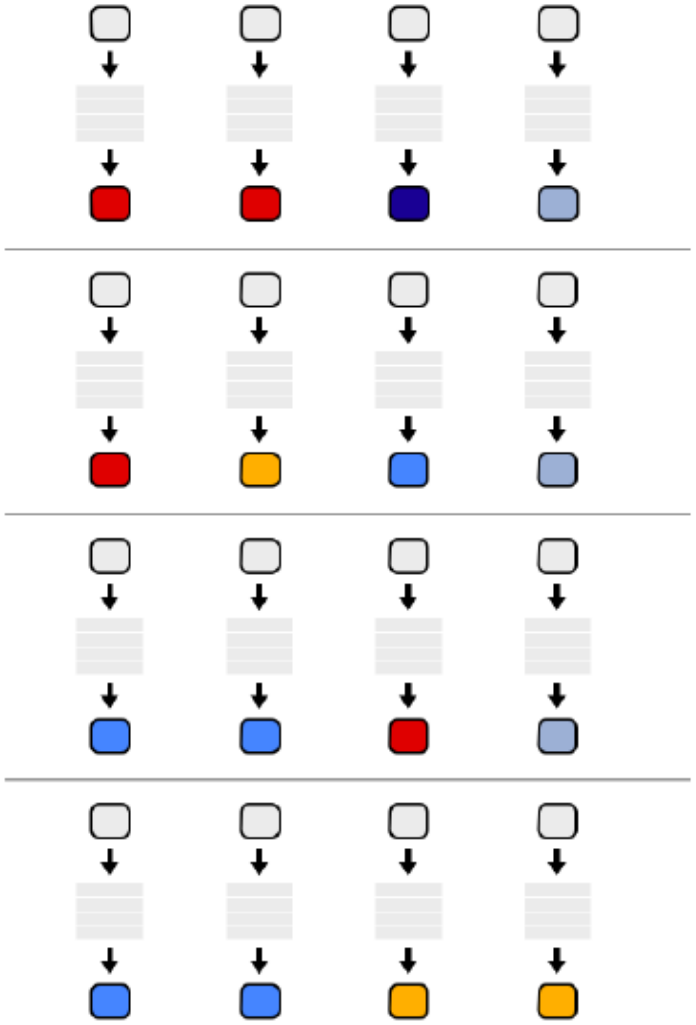


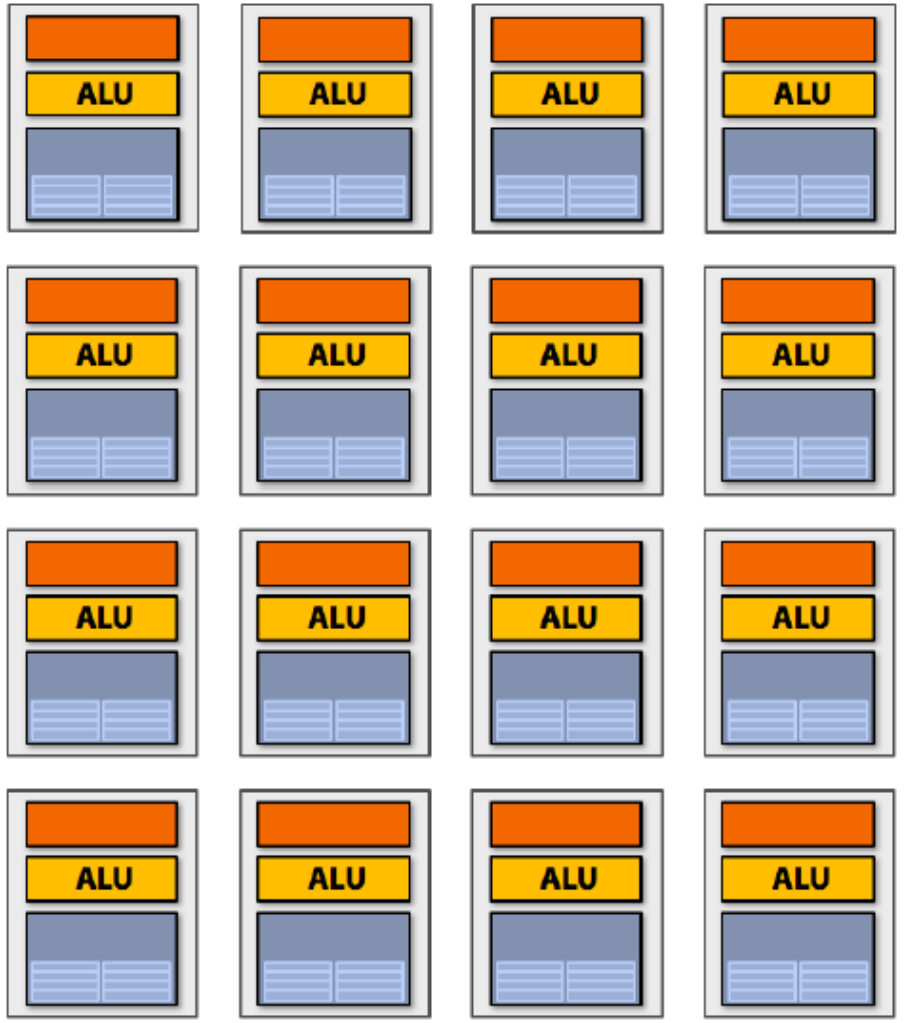
# Four cores – Four fragments





# 16 cores – 16 fragments

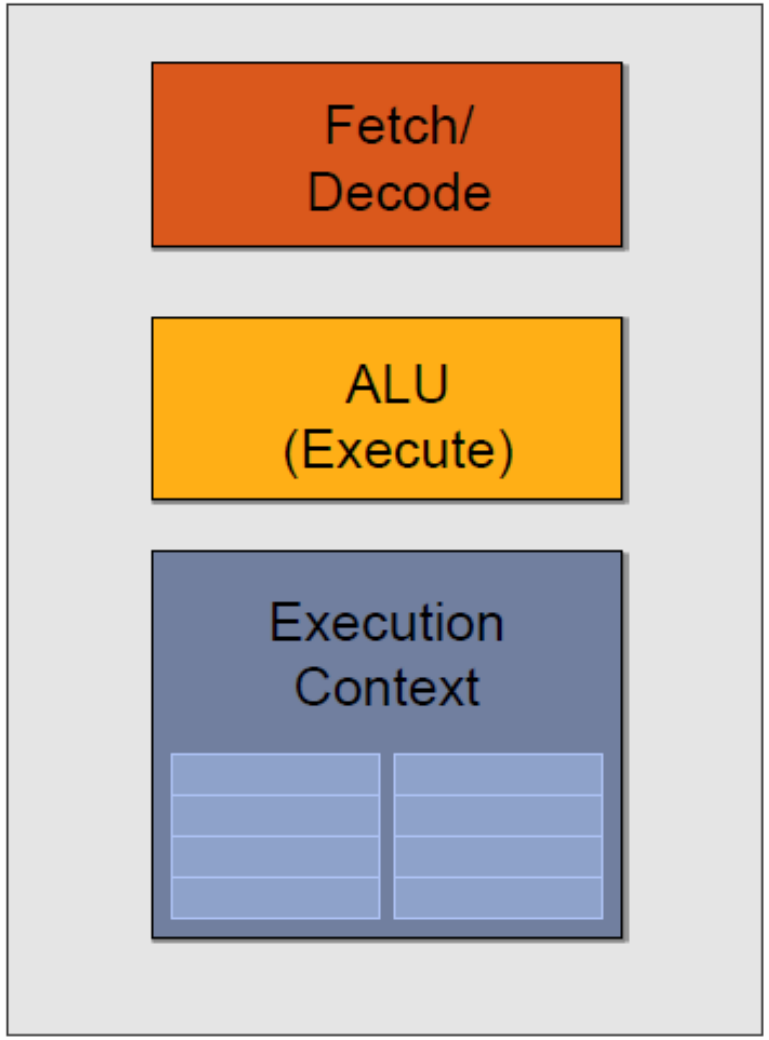




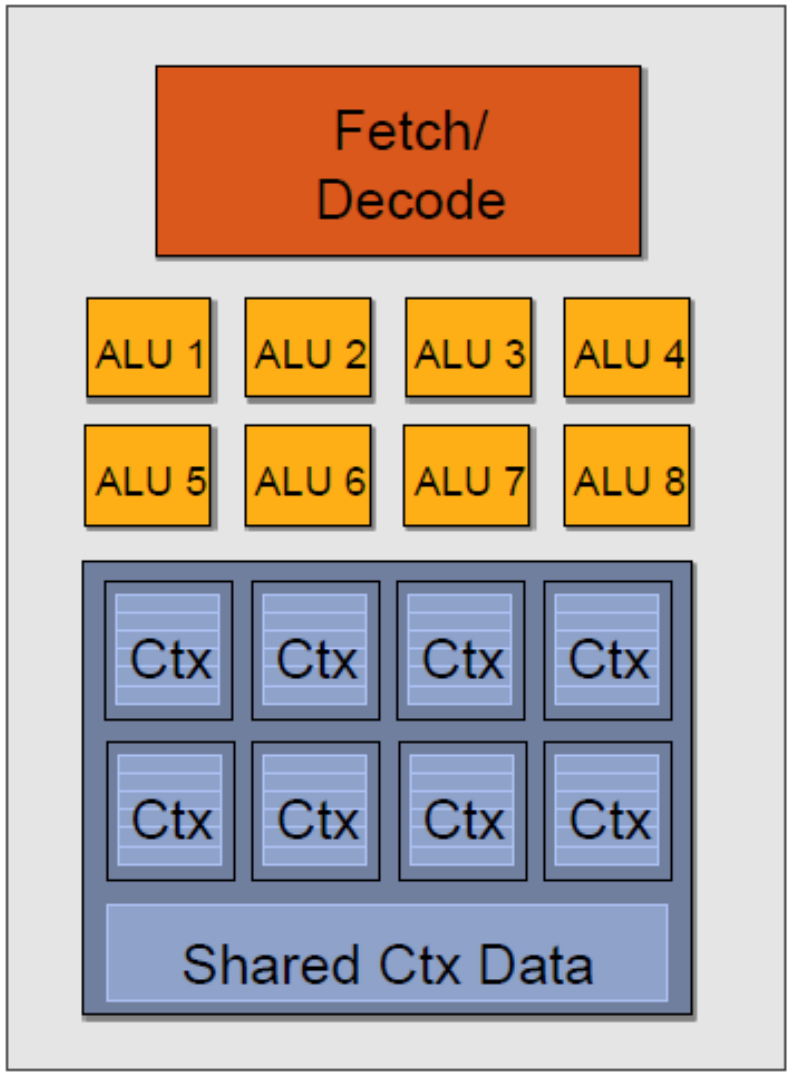
# Share an instruction stream

- Many fragments should be able to share an instruction stream

# A simple core



# Add ALUs

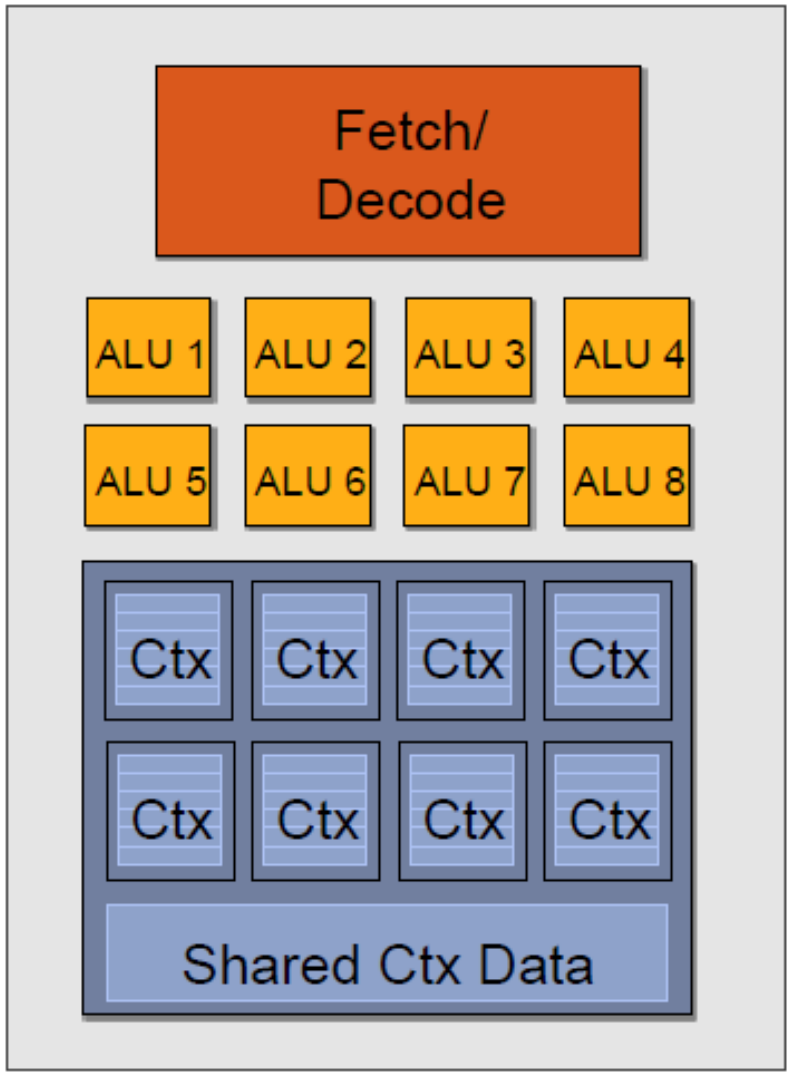



Amortize cost/complexity of managing an instruction stream across many ALUs

SIMD

Single Instruction Multiple Data

# One Frag

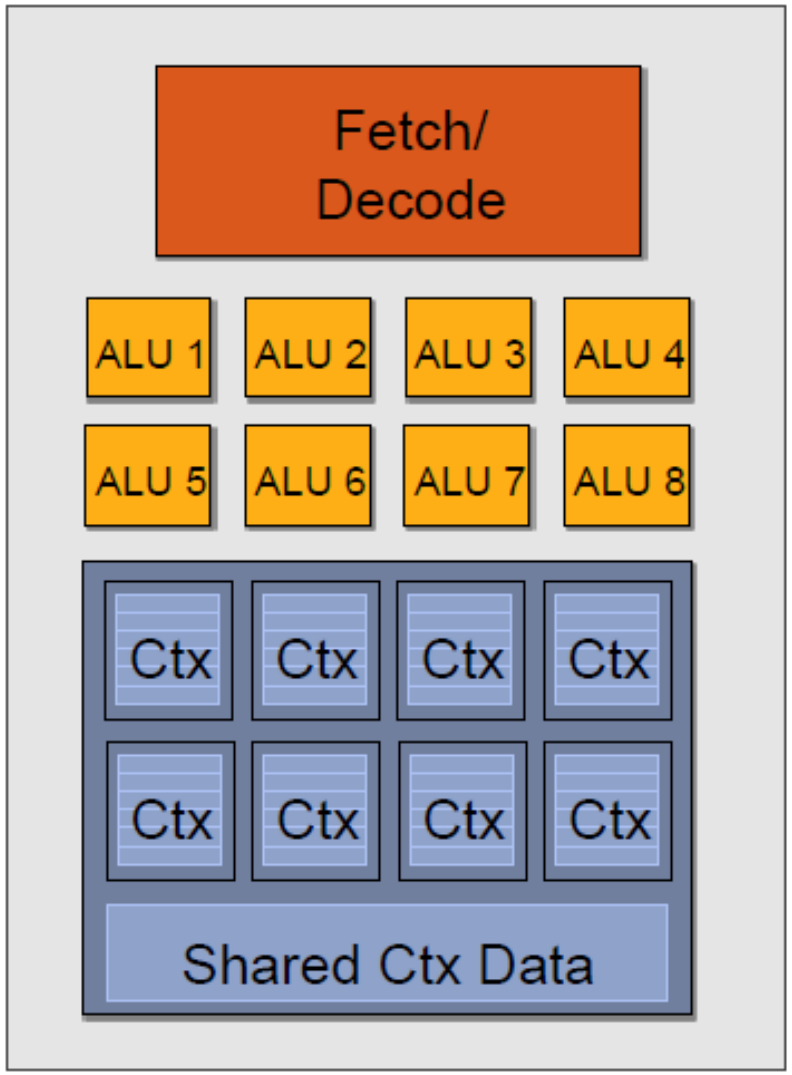



## Original Compiled Shader

```
<diffuseShader>:
sample r0, v4, t0, s0
mul  r3, v0, cb0[0]
madd r3, v1, cb0[1], r3
madd r3, v2, cb0[2], r3
clmp r3, r3, l(0.0), l(1.0)
mul  o0, r0, r3
mul  o1, r1, r3
mul  o2, r2, r3
mov  o3, l(1.0)
```

One fragment using scalar ops

# Vectorized Shader

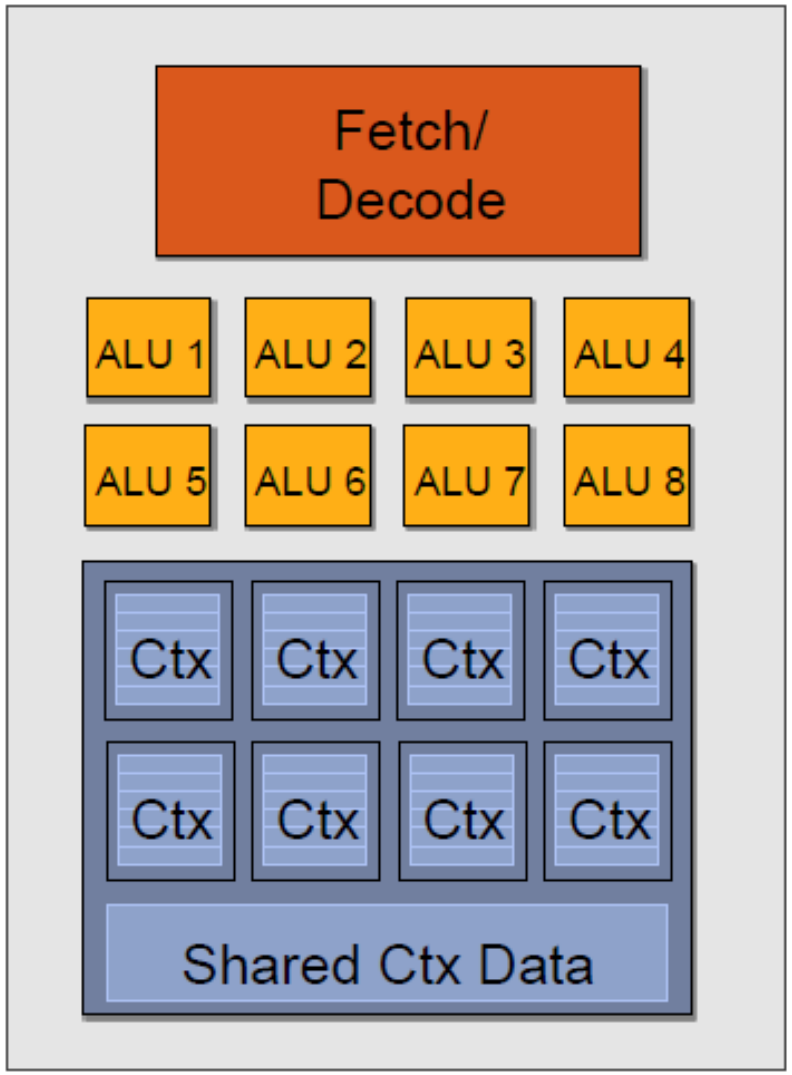


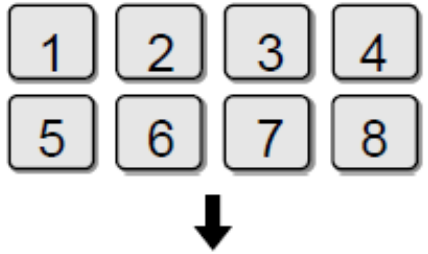

## New Compiled Shader

```
<VEC8_diffuseShader>:
VEC8_sample vec_r0, vec_v4, t0, vec_s0
VEC8_mul    vec_r3, vec_v0, cb0[0]
VEC8_madd   vec_r3, vec_v1, cb0[1], vec_r3
VEC8_madd   vec_r3, vec_v2, cb0[2], vec_r3
VEC8_clmp   vec_r3, vec_r3, 1(0.0), 1(1.0)
VEC8_mul    vec_o0, vec_r0, vec_r3
VEC8_mul    vec_o1, vec_r1, vec_r3
VEC8_mul    vec_o2, vec_r2, vec_r3
VEC8_mov    o3, 1(1.0)
```

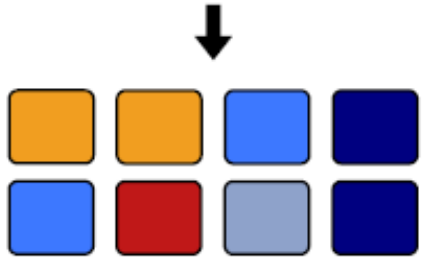
8 fragments using vector ops

# Multiple Frags

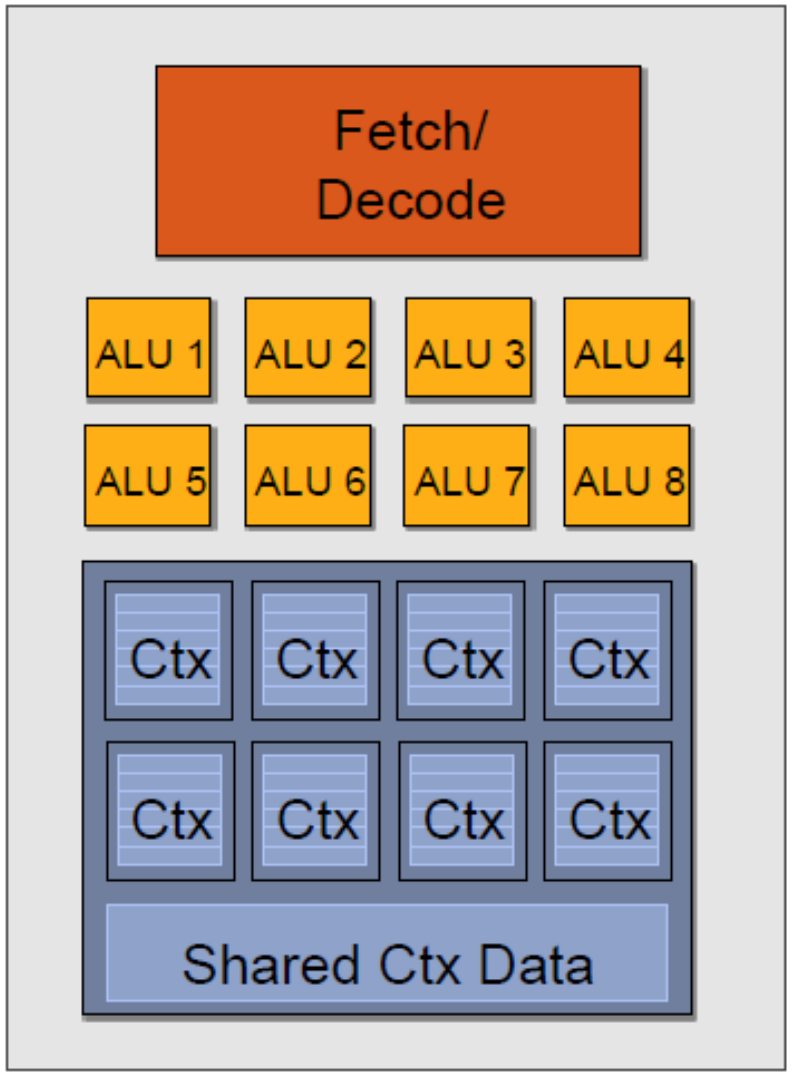


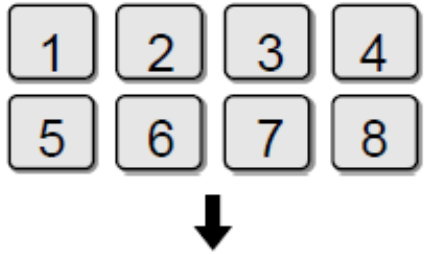




```
<VEC8_diffuseShader>:
VEC8_sample vec_r0, vec_v4, t0, vec_s0
VEC8_mul  vec_r3, vec_v0, cb0[0]
VEC8_madd vec_r3, vec_v1, cb0[1], vec_r3
VEC8_madd vec_r3, vec_v2, cb0[2], vec_r3
VEC8_clmp vec_r3, vec_r3, 1(0.0), 1(1.0)
VEC8_mul  vec_o0, vec_r0, vec_r3
VEC8_mul  vec_o1, vec_r1, vec_r3
VEC8_mul  vec_o2, vec_r2, vec_r3
VEC8_mov  o3, 1(1.0)
```

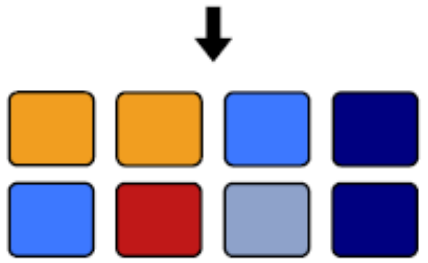


# 8 Fragments

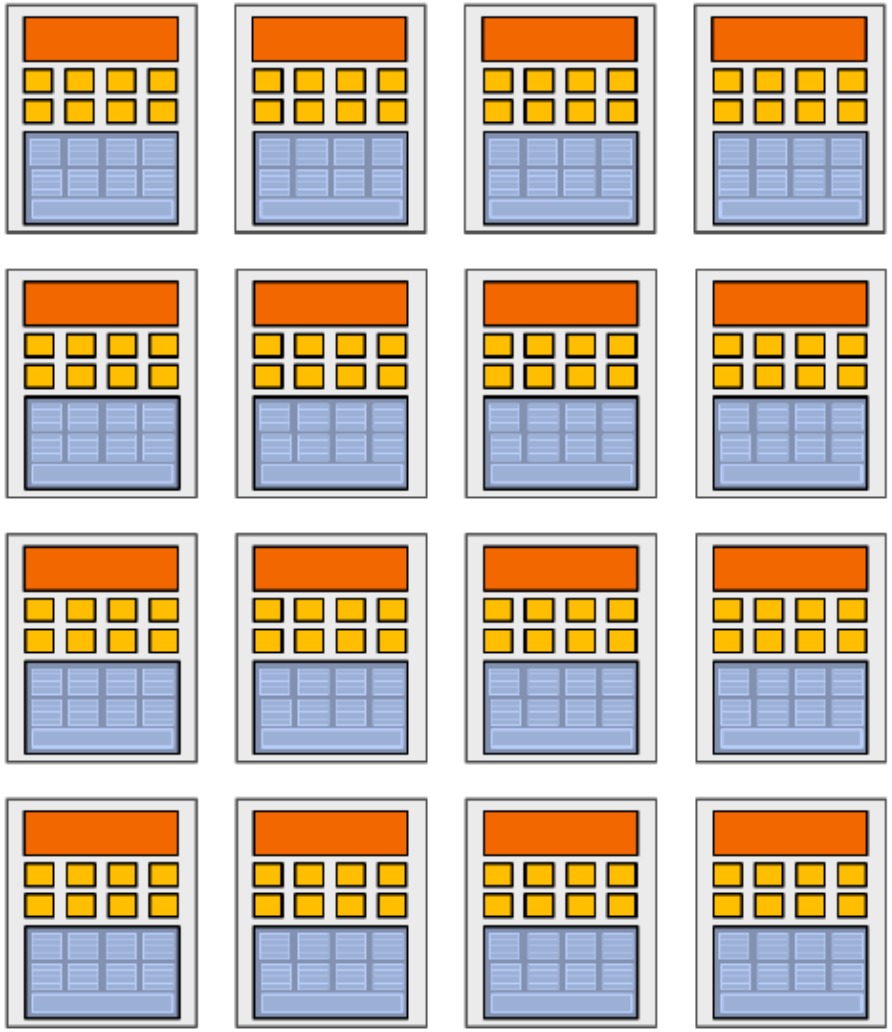



```
<VEC8_diffuseShader>:
VEC8_sample vec_r0, vec_v4, t0, vec_s0
VEC8_mul  vec_r3, vec_v0, cb0[0]
VEC8_madd vec_r3, vec_v1, cb0[1], vec_r3
VEC8_madd vec_r3, vec_v2, cb0[2], vec_r3
VEC8_clmp vec_r3, vec_r3, 1(0.0), 1(1.0)
VEC8_mul  vec_o0, vec_r0, vec_r3
VEC8_mul  vec_o1, vec_r1, vec_r3
VEC8_mul  vec_o2, vec_r2, vec_r3
VEC8_mov  o3, 1(1.0)
```

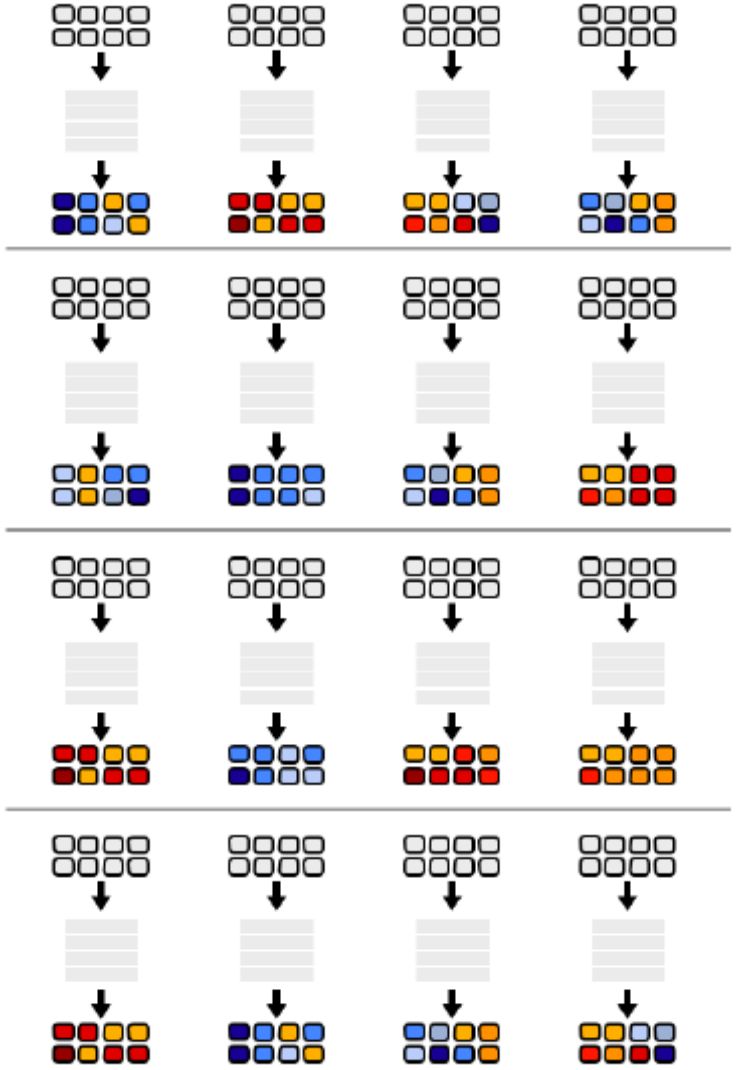


# 128 Fragments in II

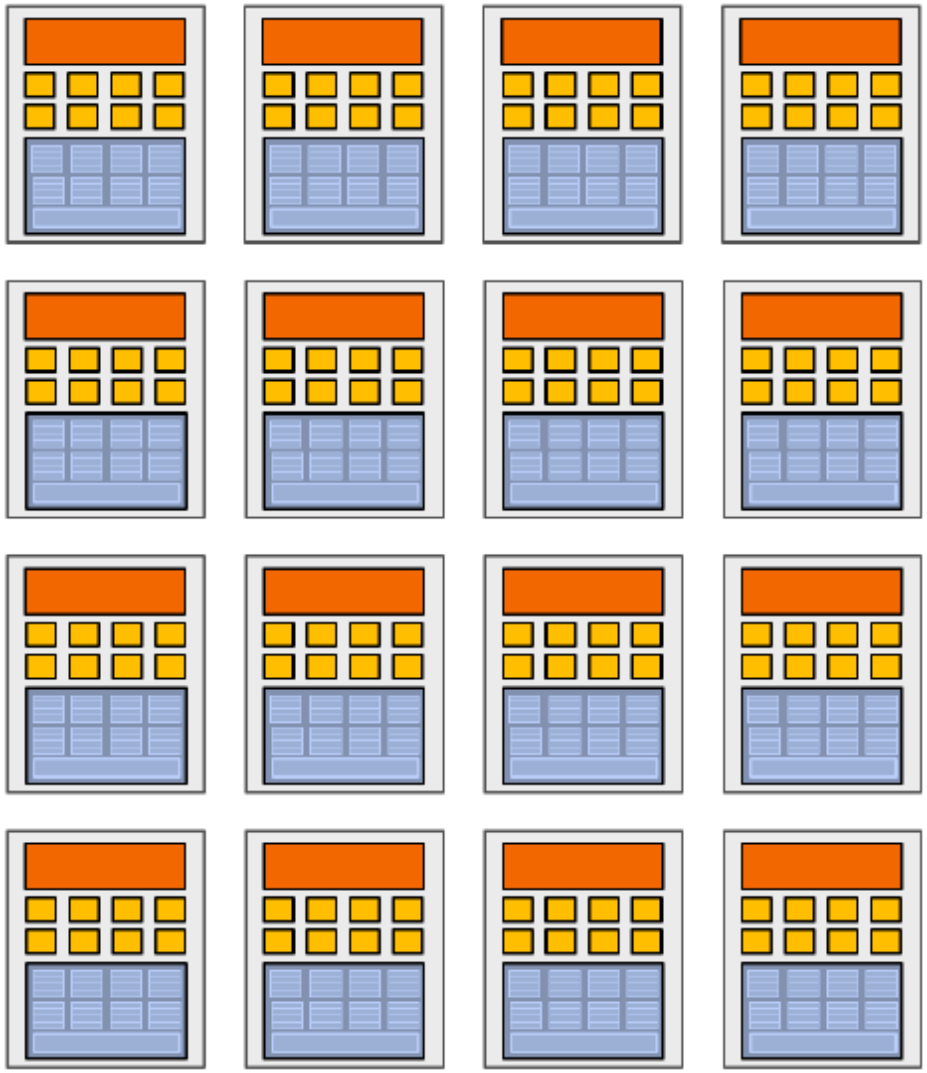


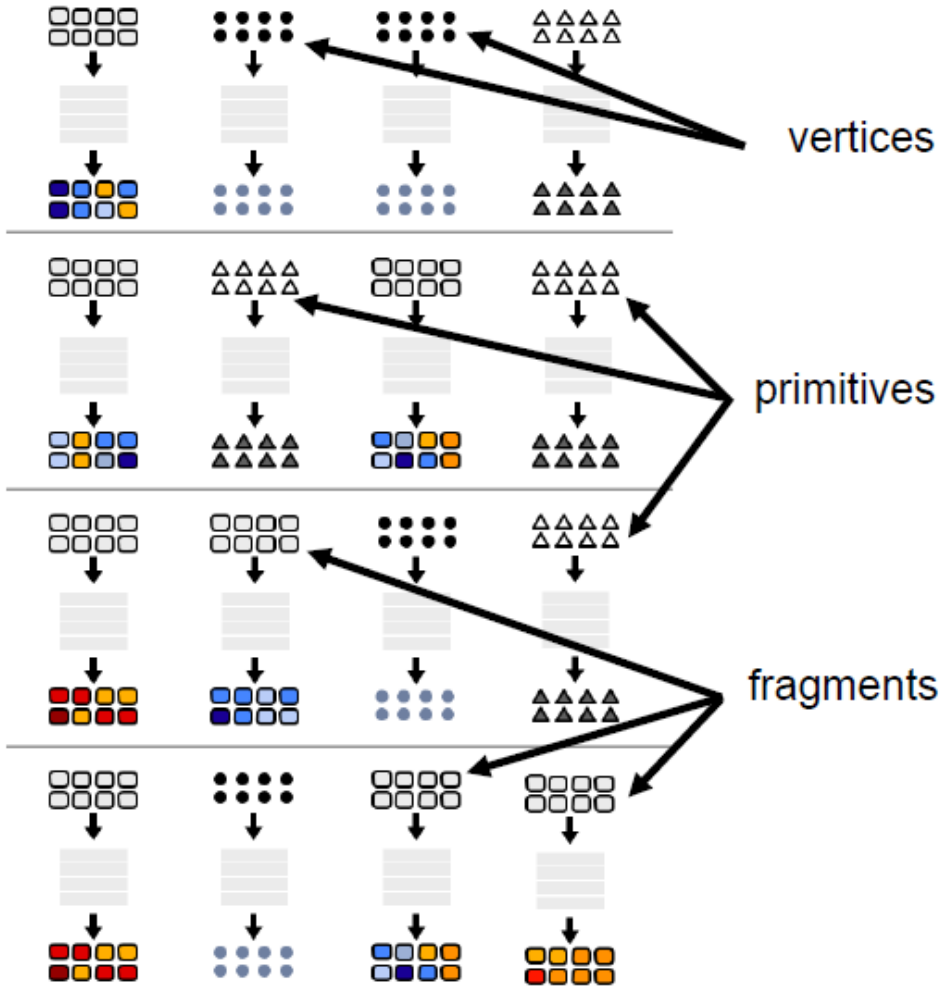
16 cores = 128 ALUs

16 instruction streams

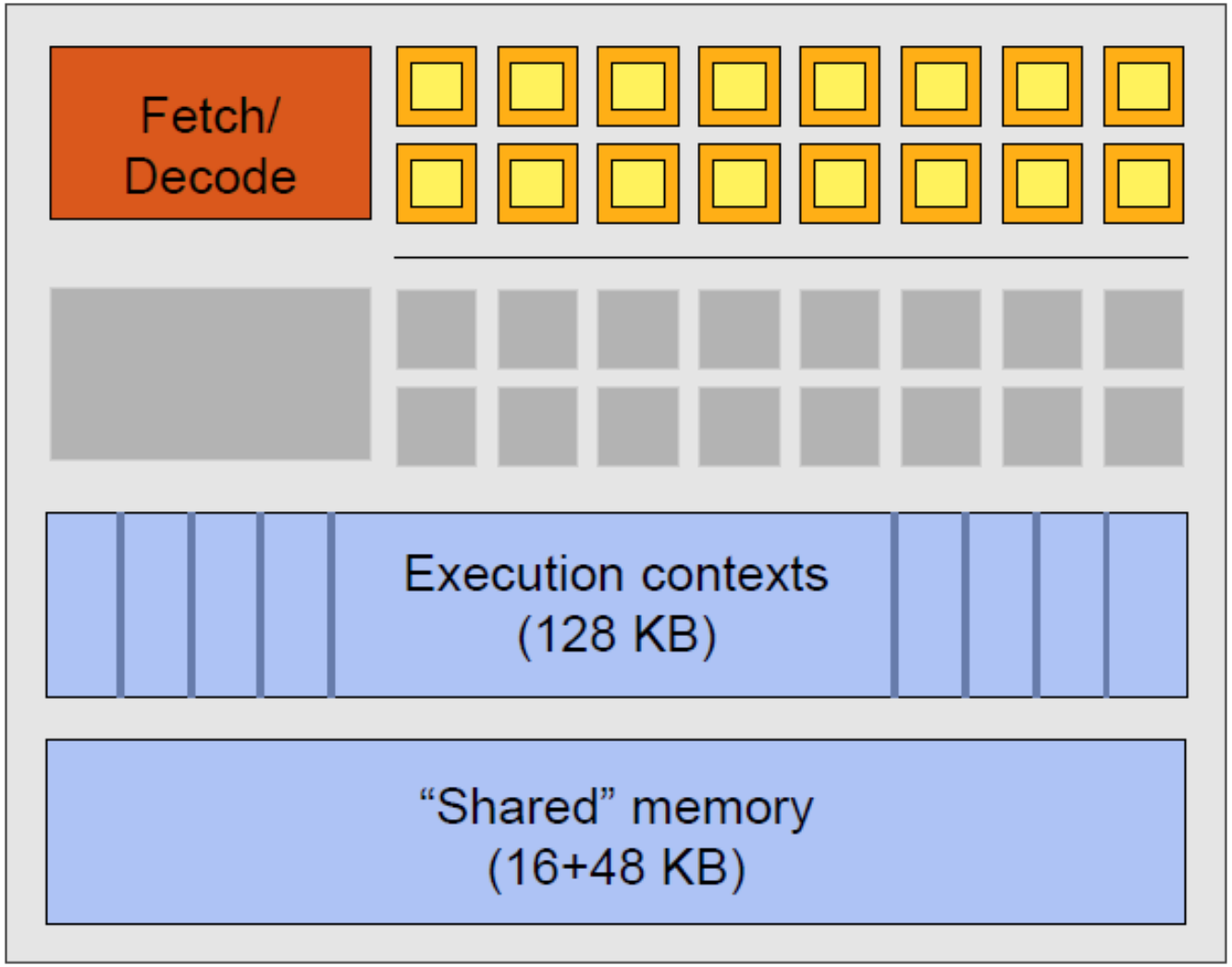


# 128 Vertices/Fragments Primitives





# Example: NVIDIA GTX 580




Groups of 32 frags/vertices/CUDA threads share an instruction stream

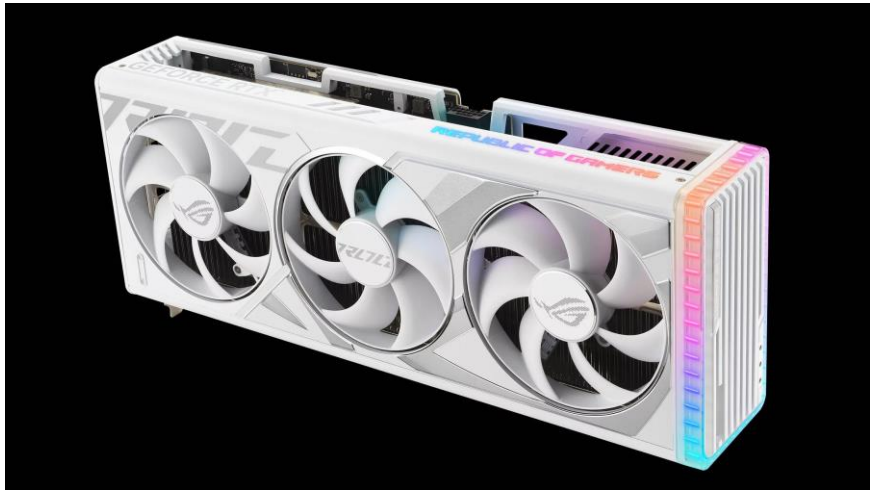
Up to 48 groups are simultaneously interleaved

Up to 1536 individual contexts can be stored

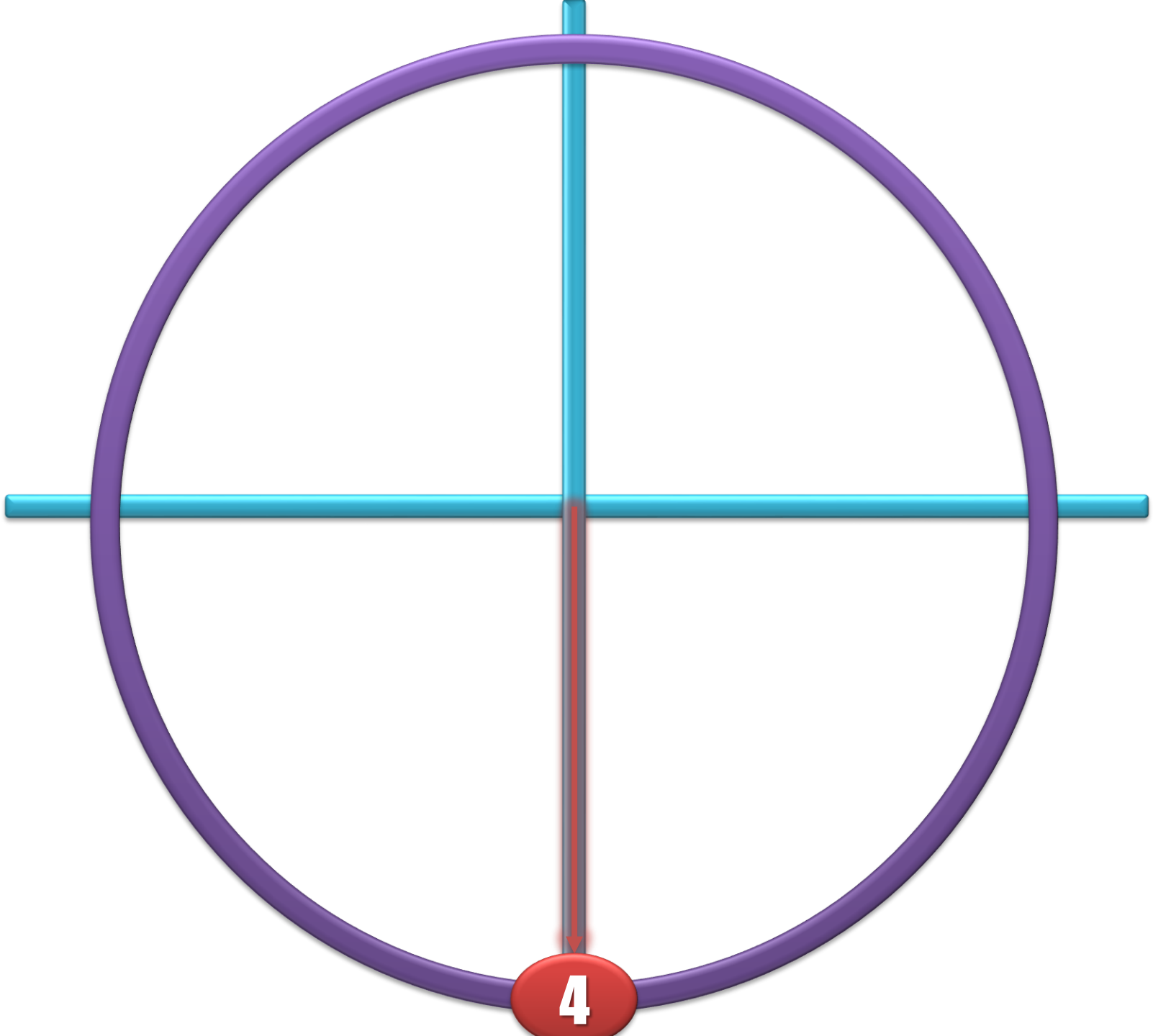
# Final Product - GTX 580

# Final Product - Latest

Nvidia GeForce RTX™ 4090, State-of-the-Art

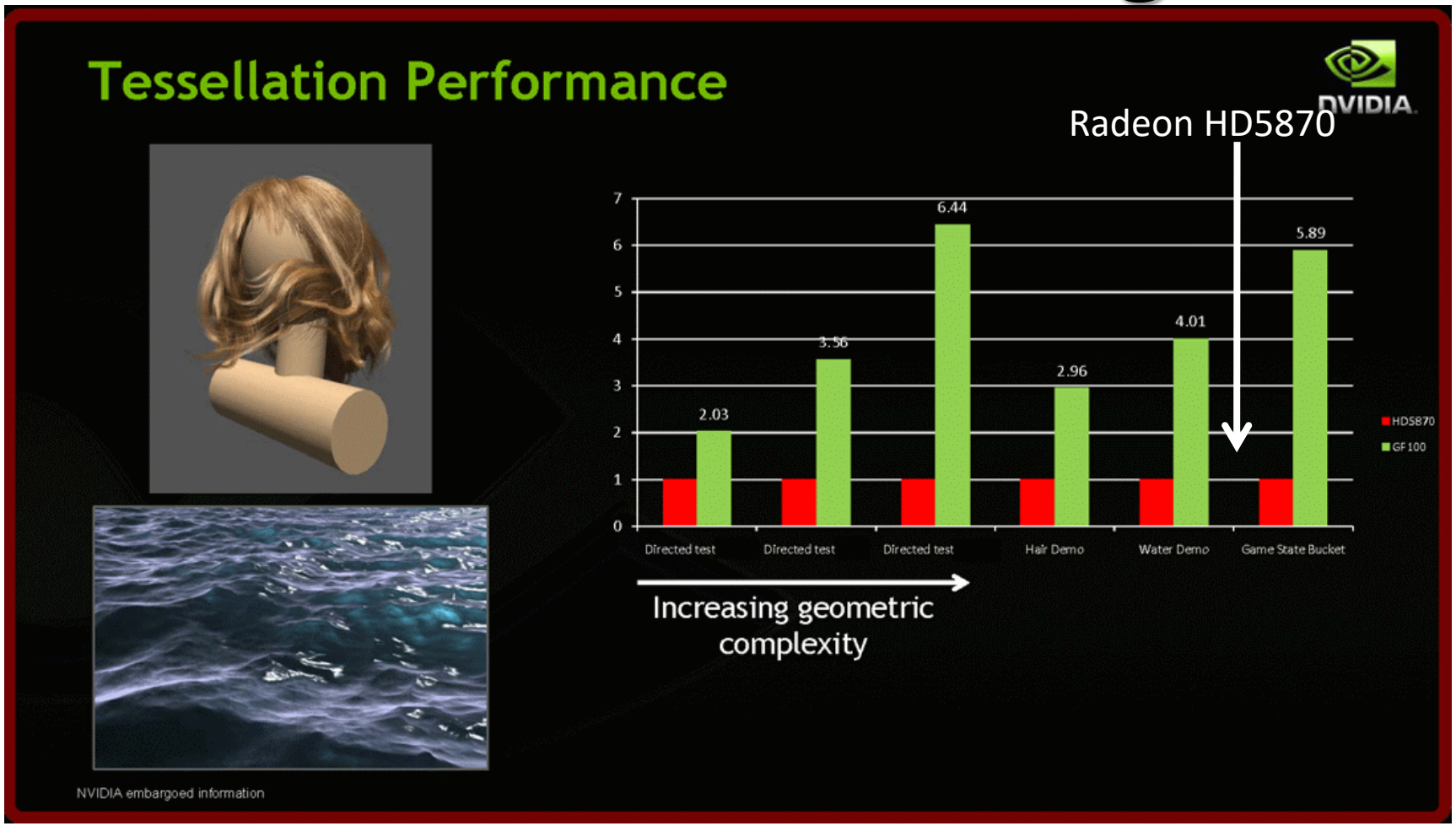



4

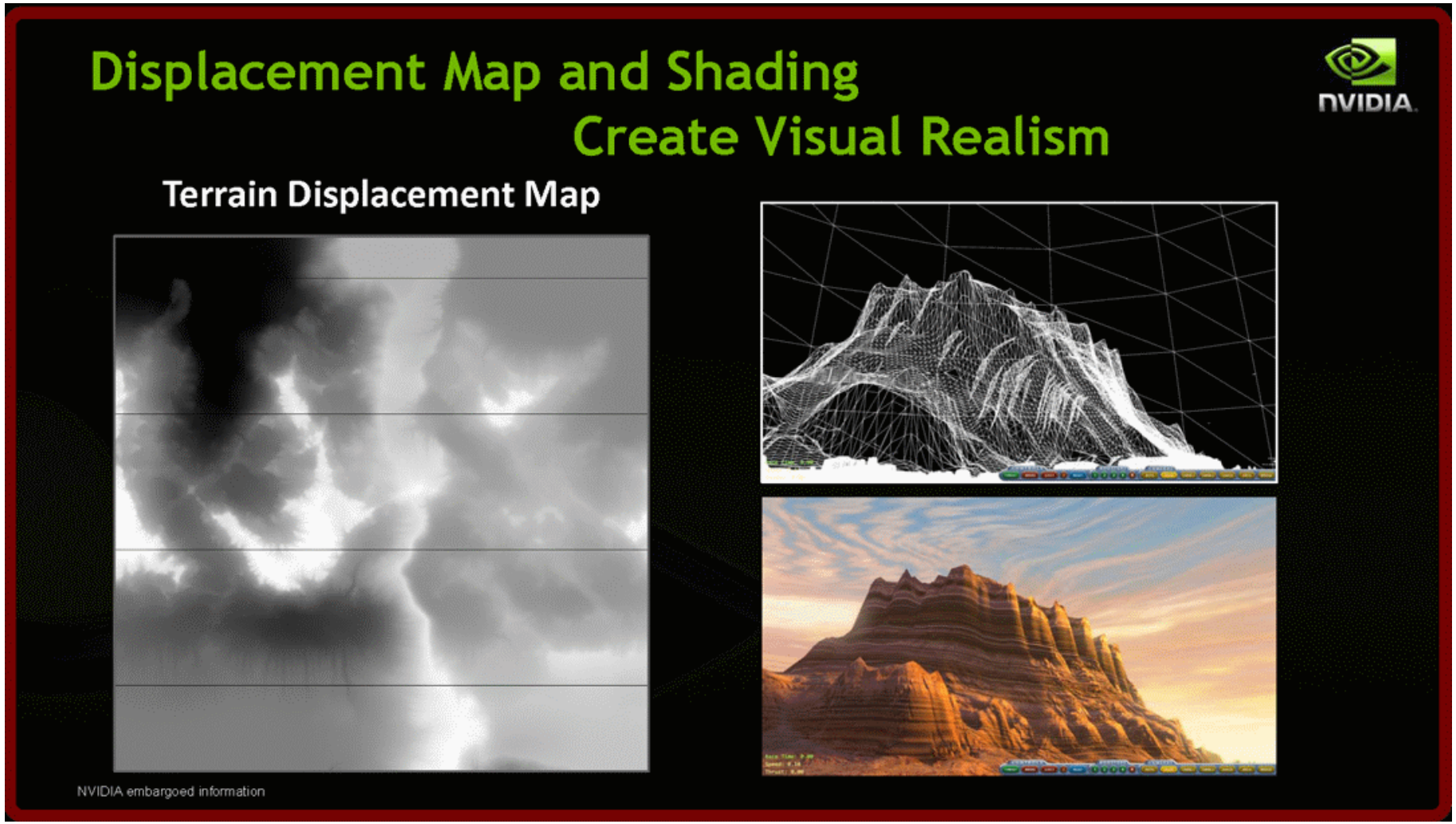
# Performance

Current top of the Line: Radeon HD5870

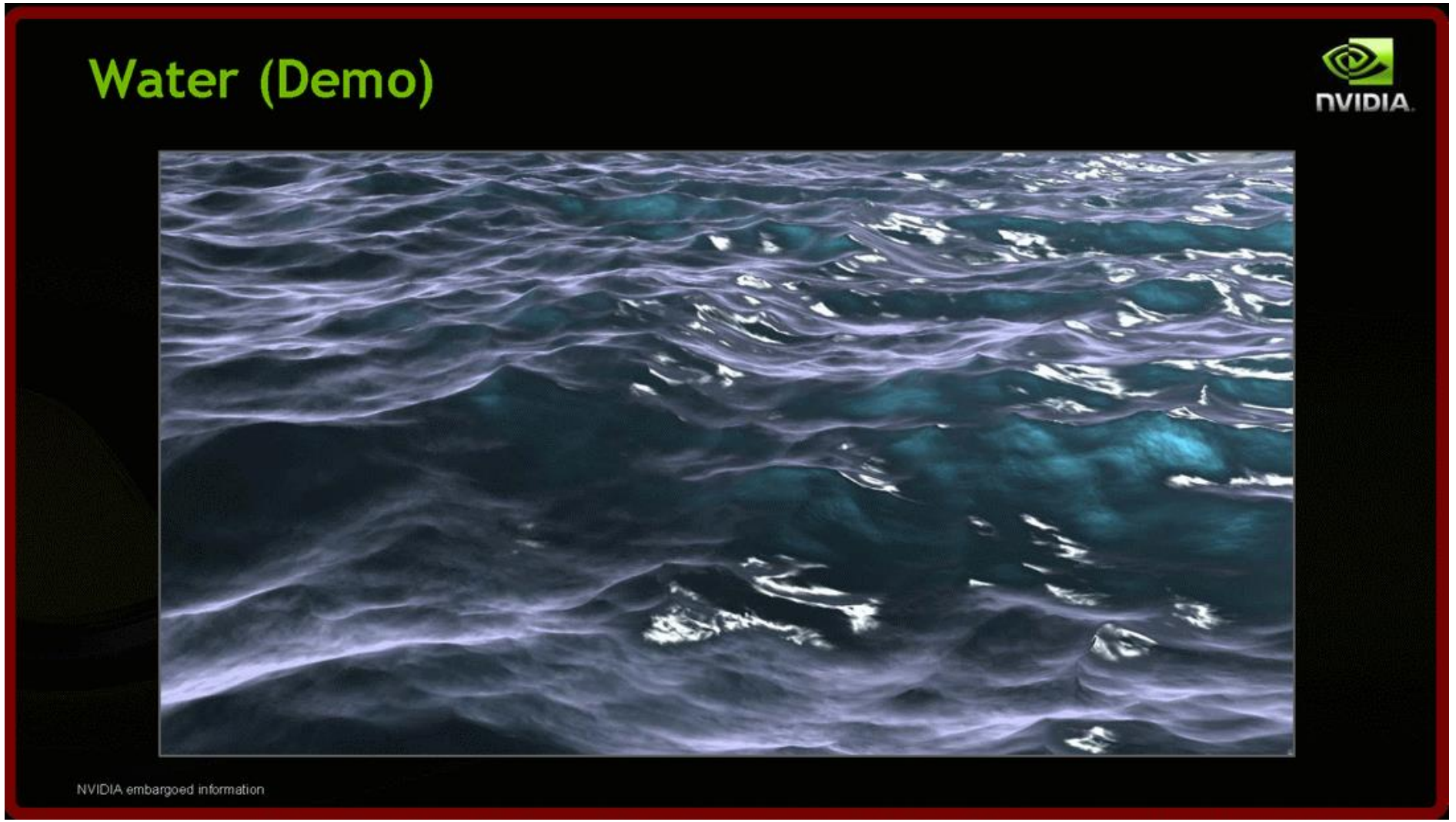
- Tessellation = no. of triangles



# Display Map and Shading



# Real-Time Water Flow



# And…Hair

## Hair (Demo)

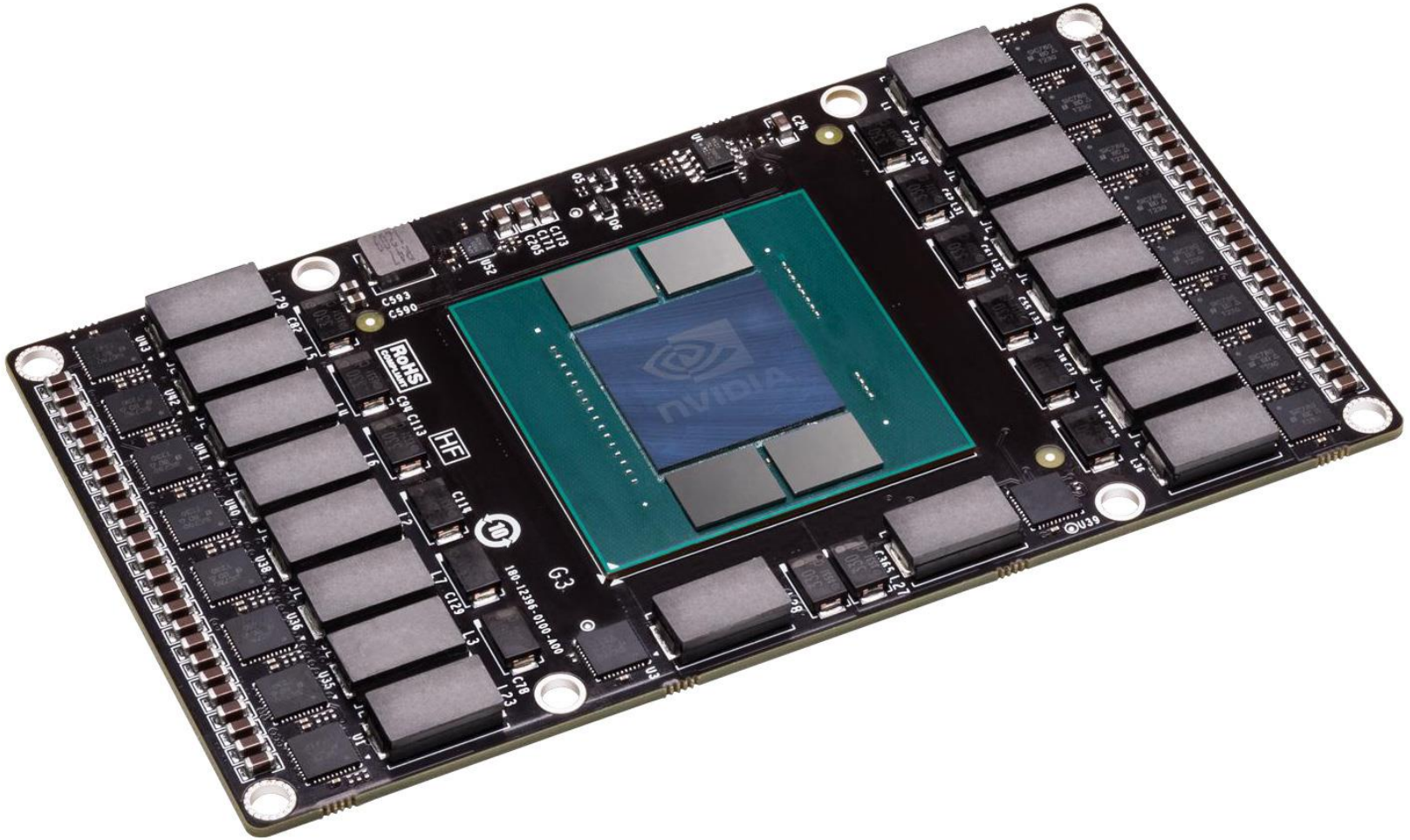
NVIDIA

- Combine tessellation, geometry shading and compute to generate hair

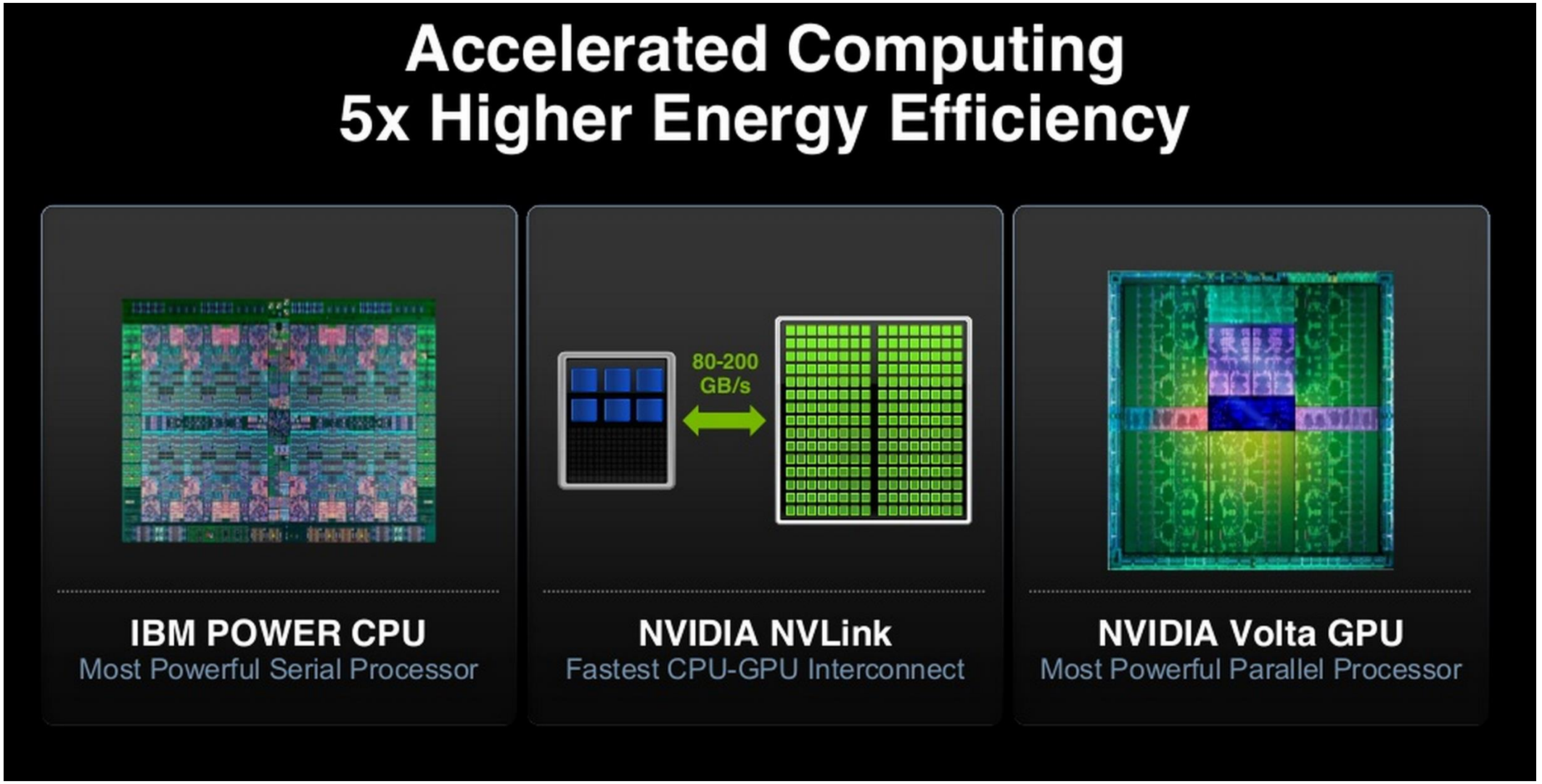
NVIDIA embargoed information

(demos)

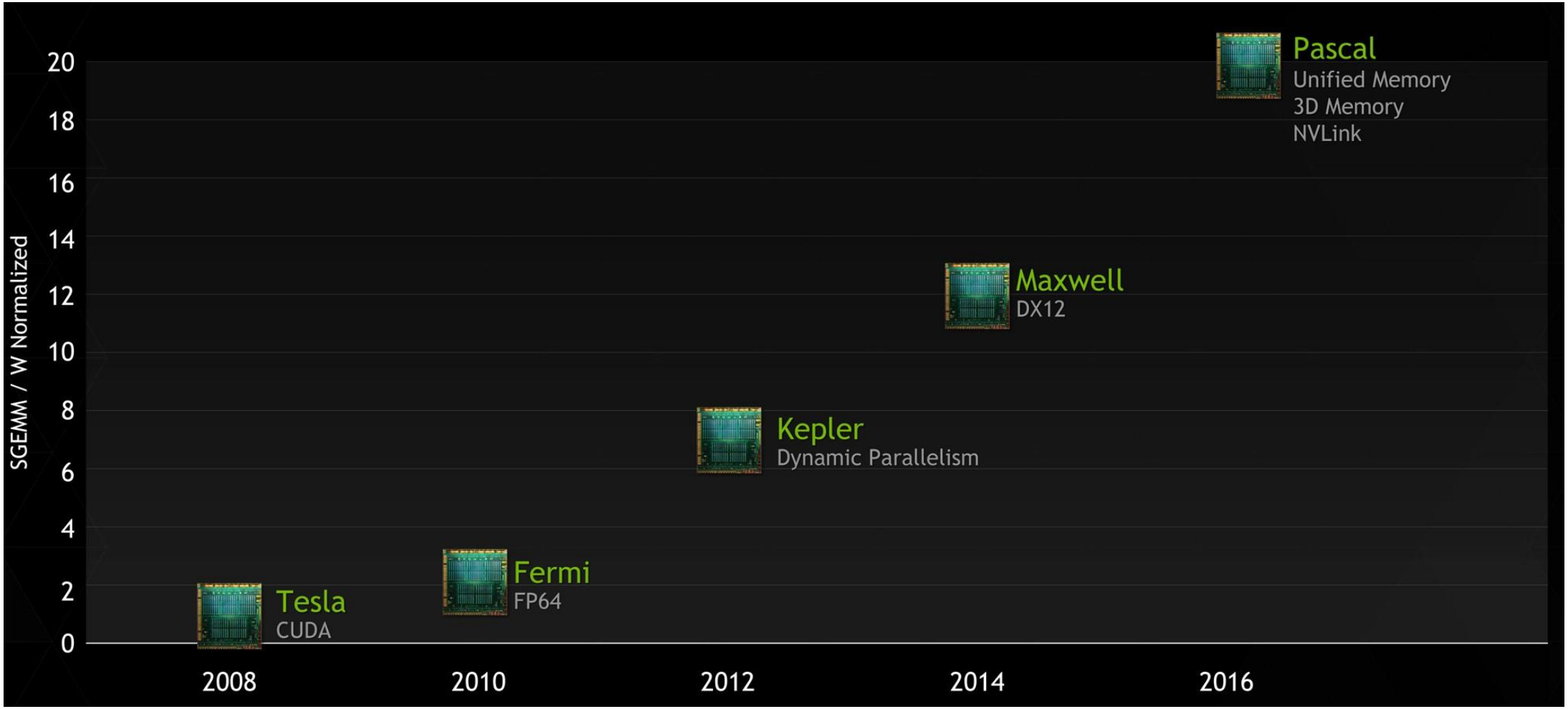
# Next: Pascal = 10 x MAXWELL

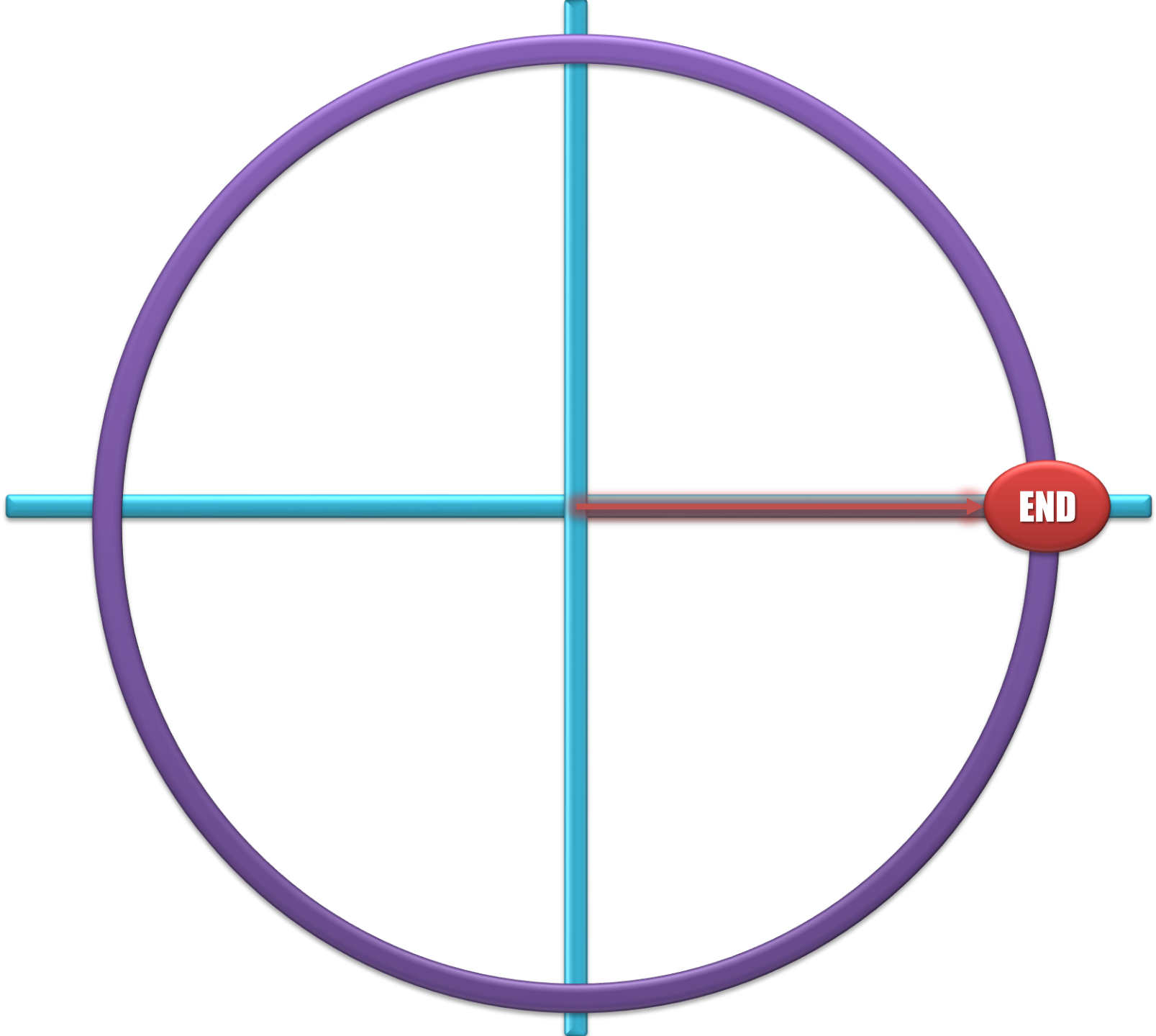


# Next: Pascal = 10 x MAXWELL



# SGEMM = Single Precision General Matrix Multiply




END

# Next:

- Javascript
- Setting up the DOM
- Basic WebGL API
- Graphics data flow