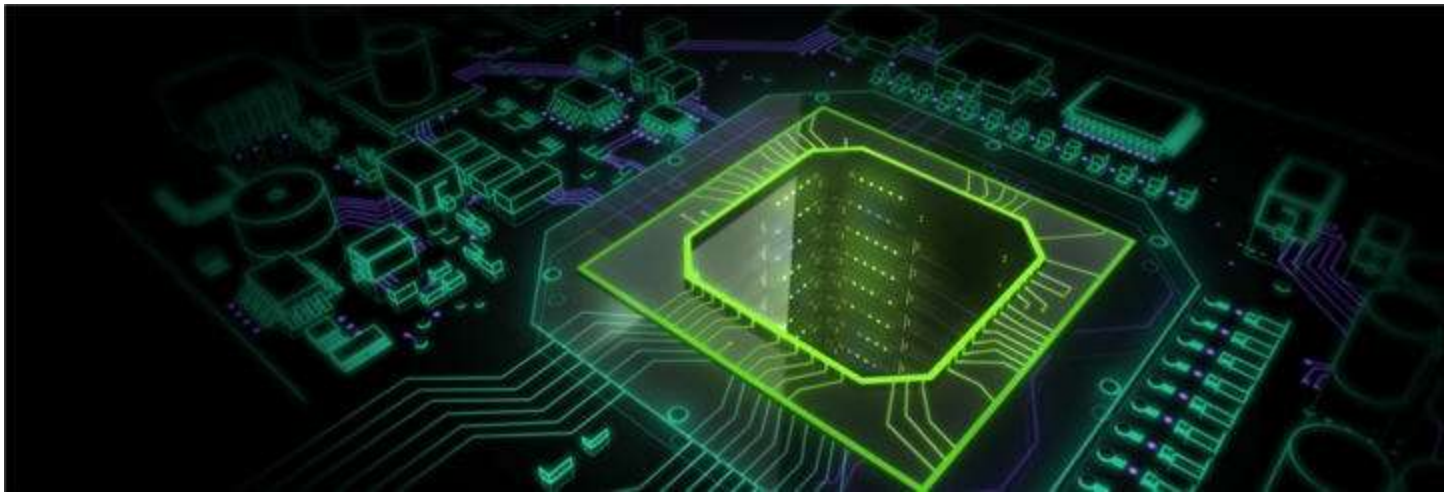
NYU
New York University

CSCI-GA.3033-012

# Graphics Processing Units (GPUs): Architecture and Programming

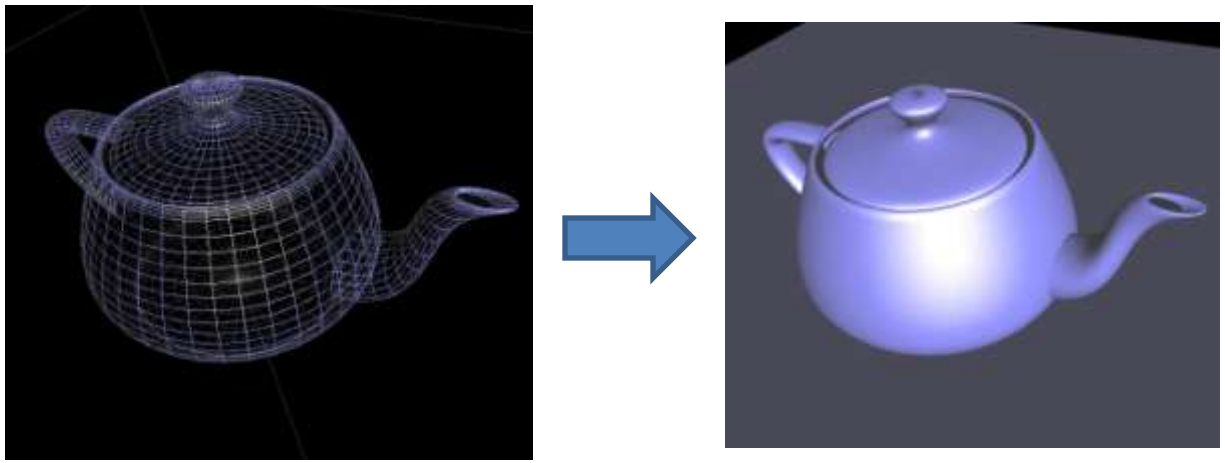
## Lecture 2: History of GPU Computing

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com



# A Little Bit of Vocabulary

- Rendering: the process of generating an image from a model
- Vertex: the corner of a polygon (usually that polygon is a triangle)
- Pixel: smallest addressable screen element



# From Numbers to Screen

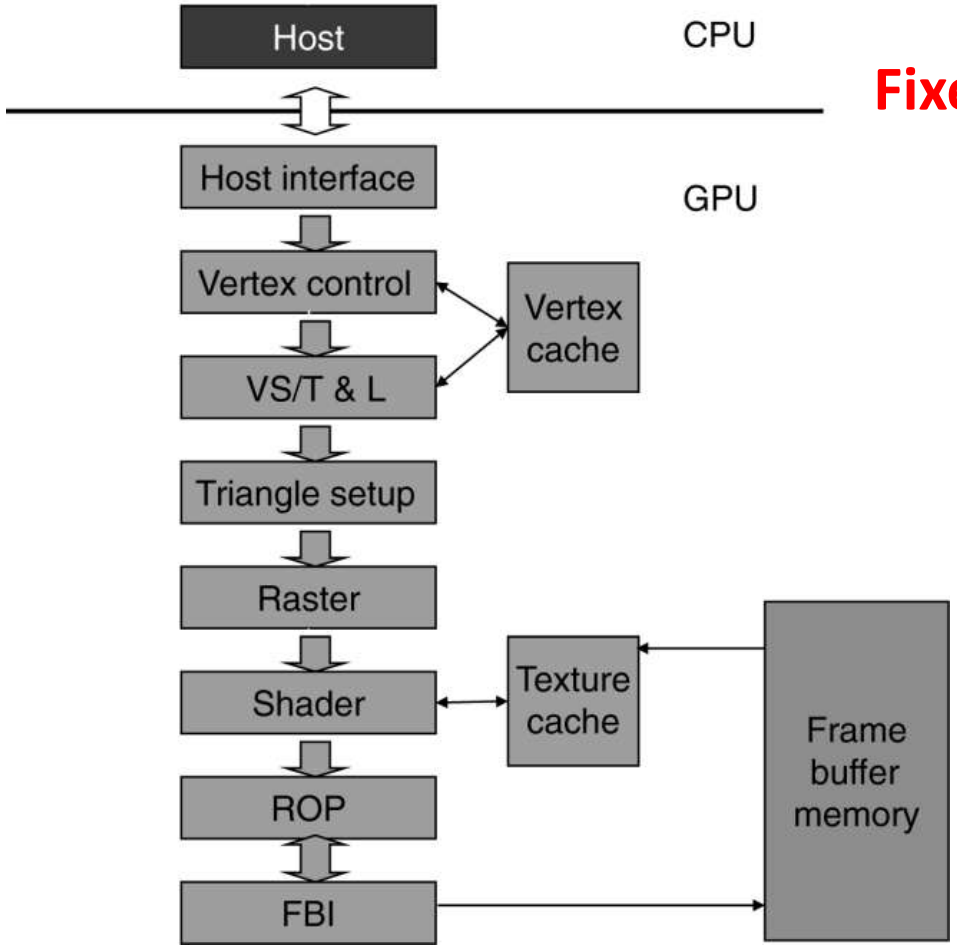
```
0.748952 -0.764952 -0.210132,
0.672245 -0.688862 -0.210132,
1.00016 -0.369596 -0.210132,
1.09939 -0.004004 -0.210132,
1.14496 0.324436 -0.210132,
1.15747 0.601712 -0.210132,
1.08016 0.793529 -0.210132,
0.98164 0.872032 -0.210132,
0.808263 0.929016 -0.210132,
0.442563 0.985585 -0.210132,
0.221794 1.00159 -0.210132,
0 1.0053 -0.210132,
-0.221794 1.00159 -0.210132,
-0.442563 0.985585 -0.210132,
-0.808263 0.929016 -0.210132,
```

# Before GPUs

- Vertices to pixels:
  - Transformations done on CPU
  - Compute each pixel "by hand", in series… **slow!**

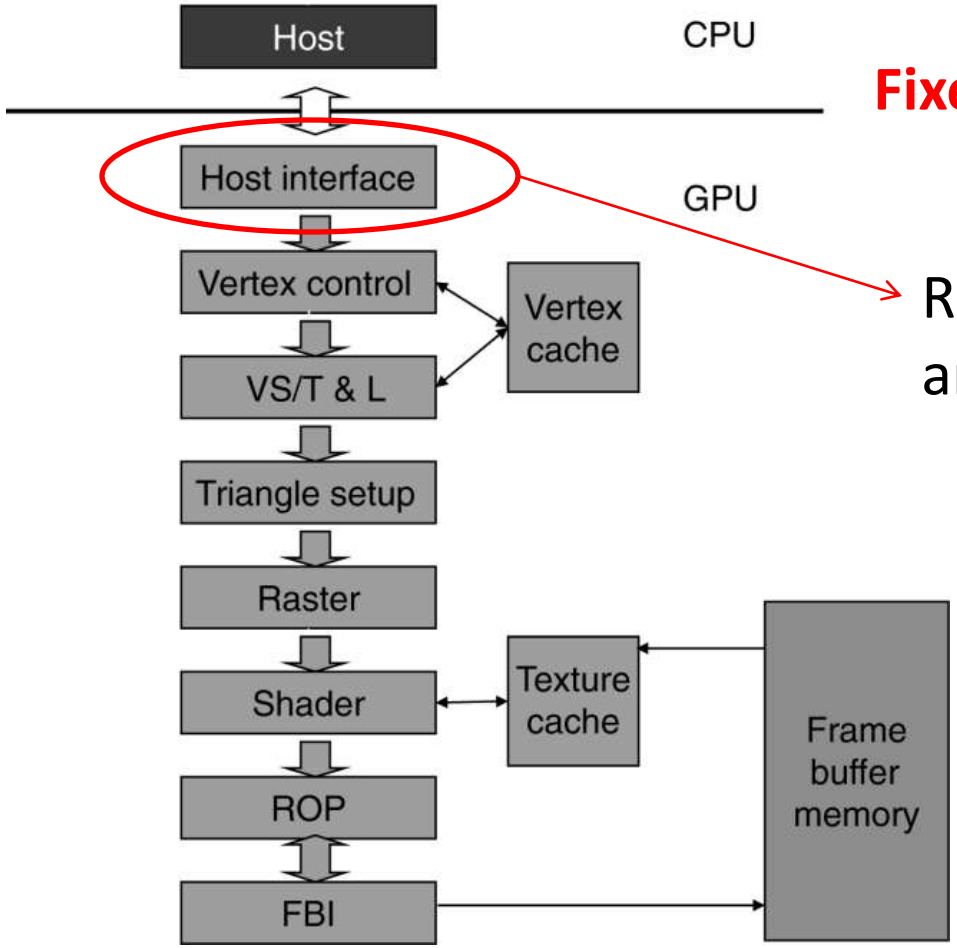
**Example:** 1 million triangles * 100 pixels per triangle * 10 lights * 4 cycles per light computation = **4 billion cycles**

# Early GPUs: Early 80s to Late 90s




**Fixed-Function Pipeline**

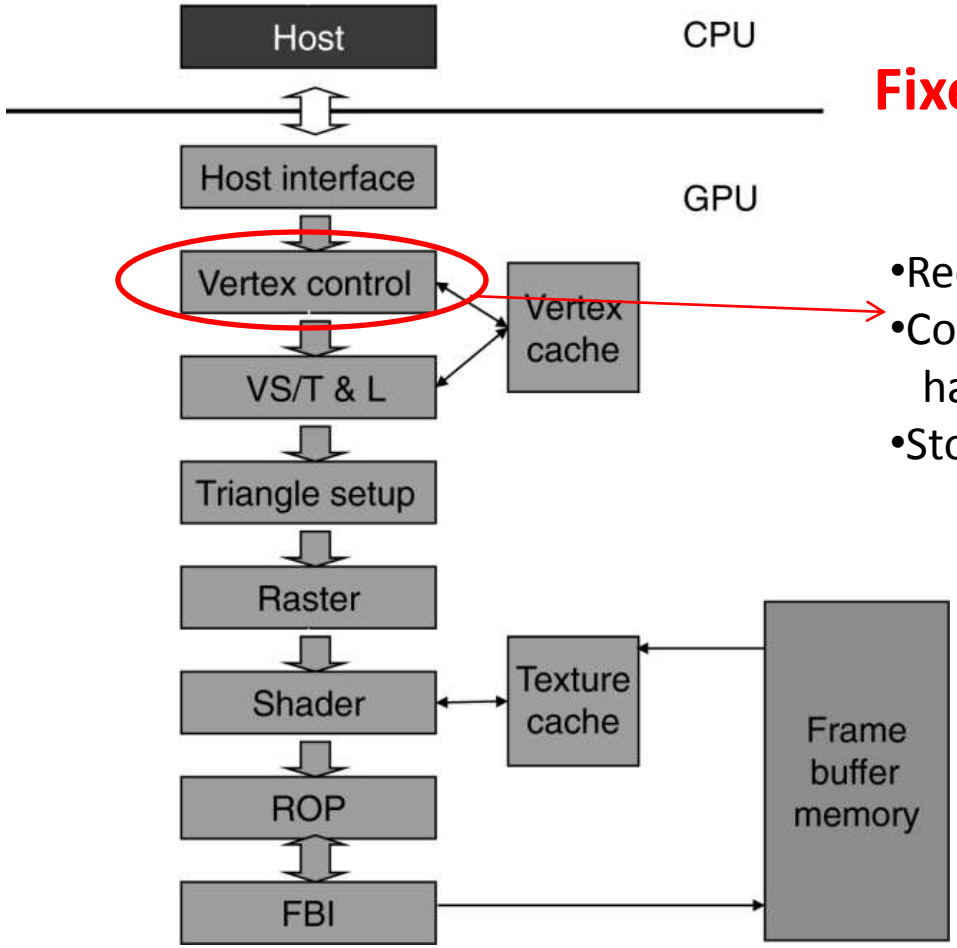
# Early GPUs: Early 80s to Late 90s




**Fixed-Function Pipeline**

Receives graphics commands and data from CPU

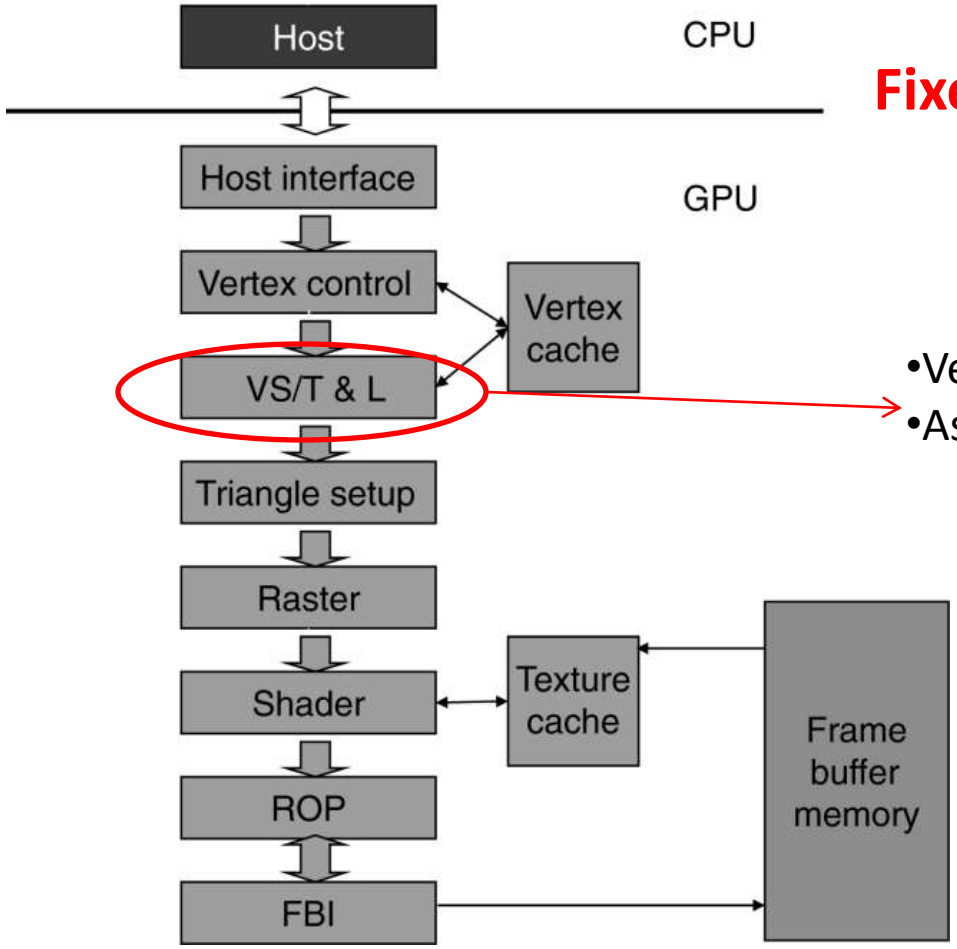
# Early GPUs: Early 80s to Late 90s




**Fixed-Function Pipeline**

- Receives triangle data
- Converts them into a form that hardware understands
- Store the prepared data in vertex cache

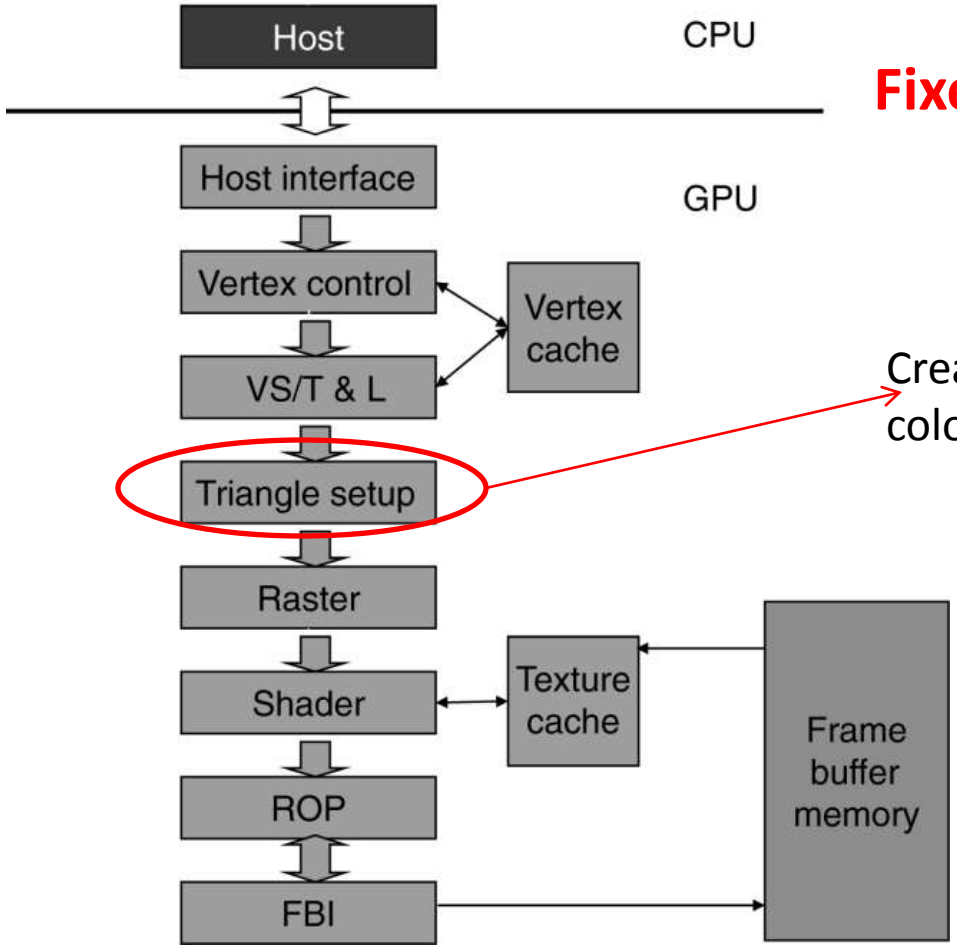
# Early GPUs: Early 80s to Late 90s




**Fixed-Function Pipeline**

- Vertex shading transform and lighting
- Assigns per-vertex value

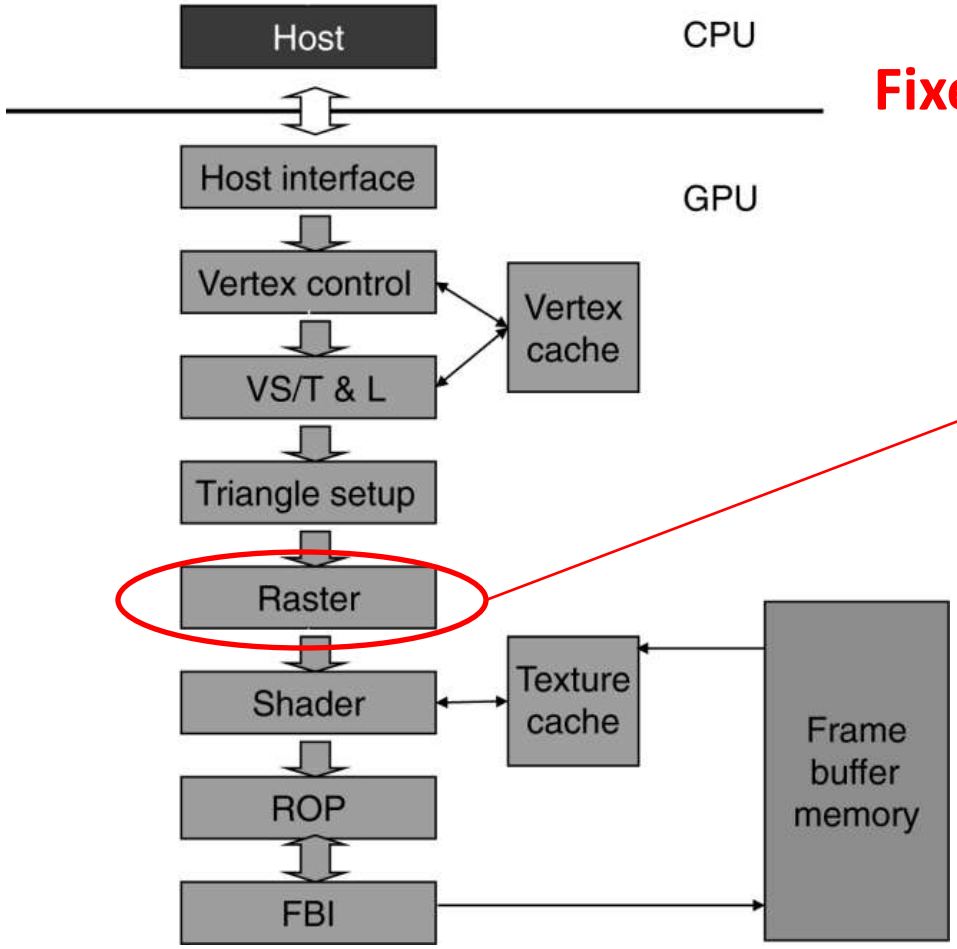
# Early GPUs: Early 80s to Late 90s




**Fixed-Function Pipeline**

Creates edge equations to interpolate colors across pixels touched by the triangle

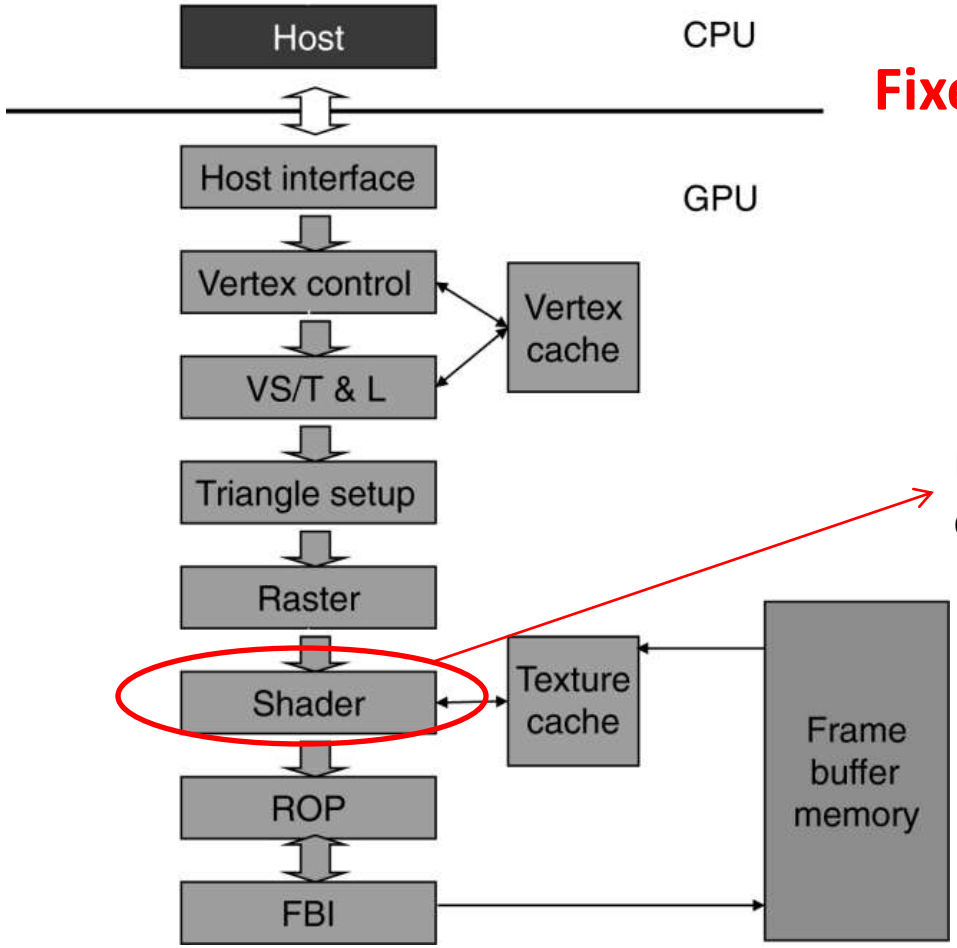
# Early GPUs: Early 80s to Late 90s




**Fixed-Function Pipeline**

- Determines which pixel falls in which triangle
- For each pixel, interpolate per-pixel values

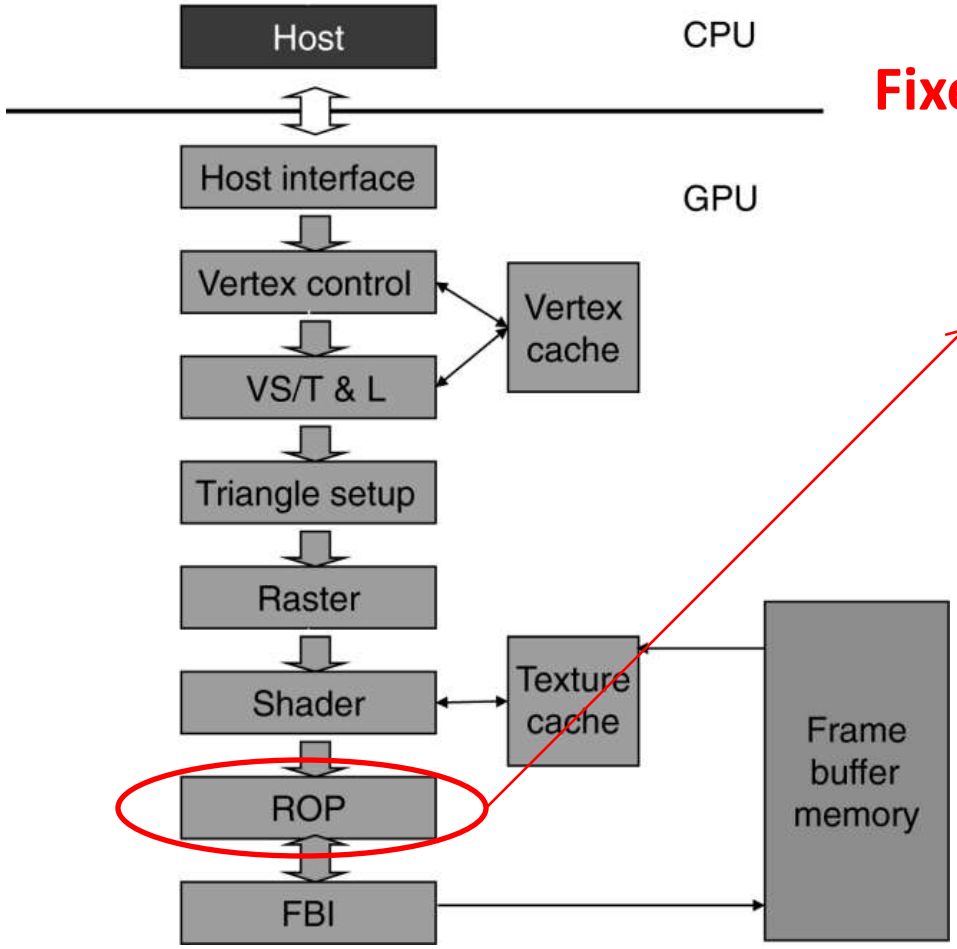
# Early GPUs: Early 80s to Late 90s




**Fixed-Function Pipeline**

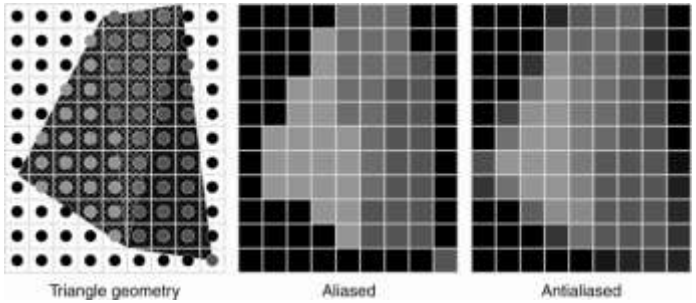
Determines the final color of each pixel

# Early GPUs: Early 80s to Late 90s

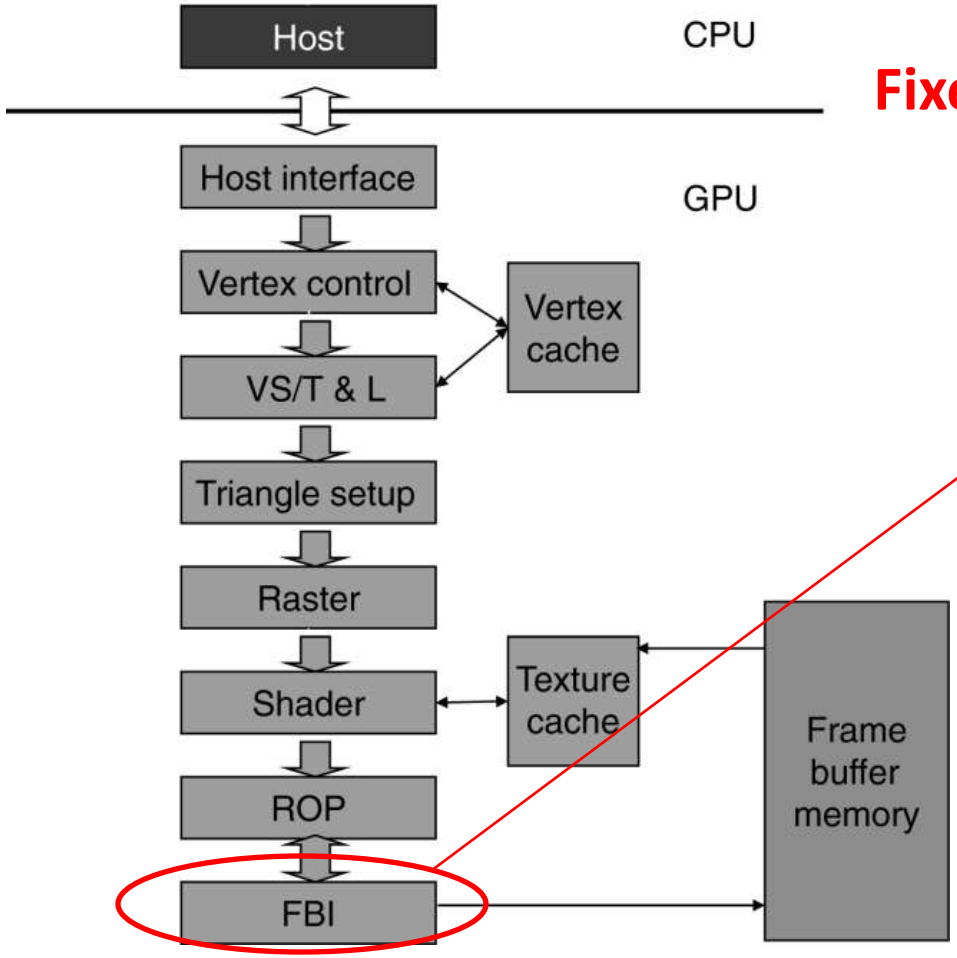



**Fixed-Function Pipeline**

The raster operation:
performs color raster operations that blend the color of overlapping objects for transparency and antialiasing



# Early GPUs: Early 80s to Late 90s

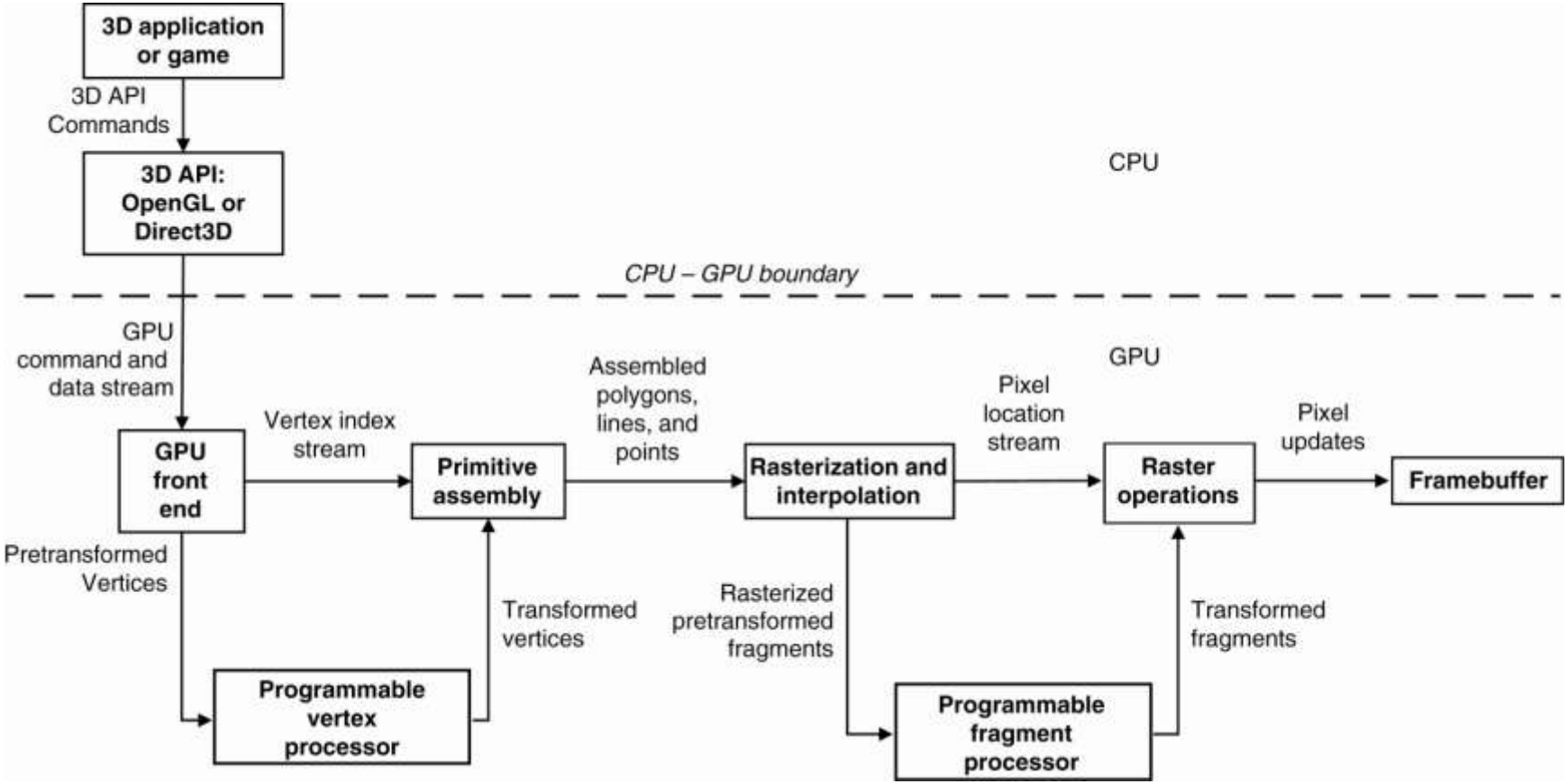



**Fixed-Function Pipeline**

The frame buffer interface manages memory reads/writes.

# Next Steps

- In 2001:
  - NVIDIA exposed the application developer to the instruction set of VS/T&L stage
- Later:
  - General programmability extended to to shader stage
  - Data independence is exploited


3D application or game
3D API Commands
3D API: OpenGL or Direct3D
CPU
CPU – GPU boundary
GPU command and data stream
GPU
Vertex index stream
Assembled polygons, lines, and points
Pixel location stream
Pixel updates
GPU front end
Primitive assembly
Rasterization and interpolation
Raster operations
Framebuffer
Pretransformed Vertices
Transformed vertices
Rasterized pretransformed fragments
Transformed fragments
Programmable vertex processor
Programmable fragment processor

# In 2006

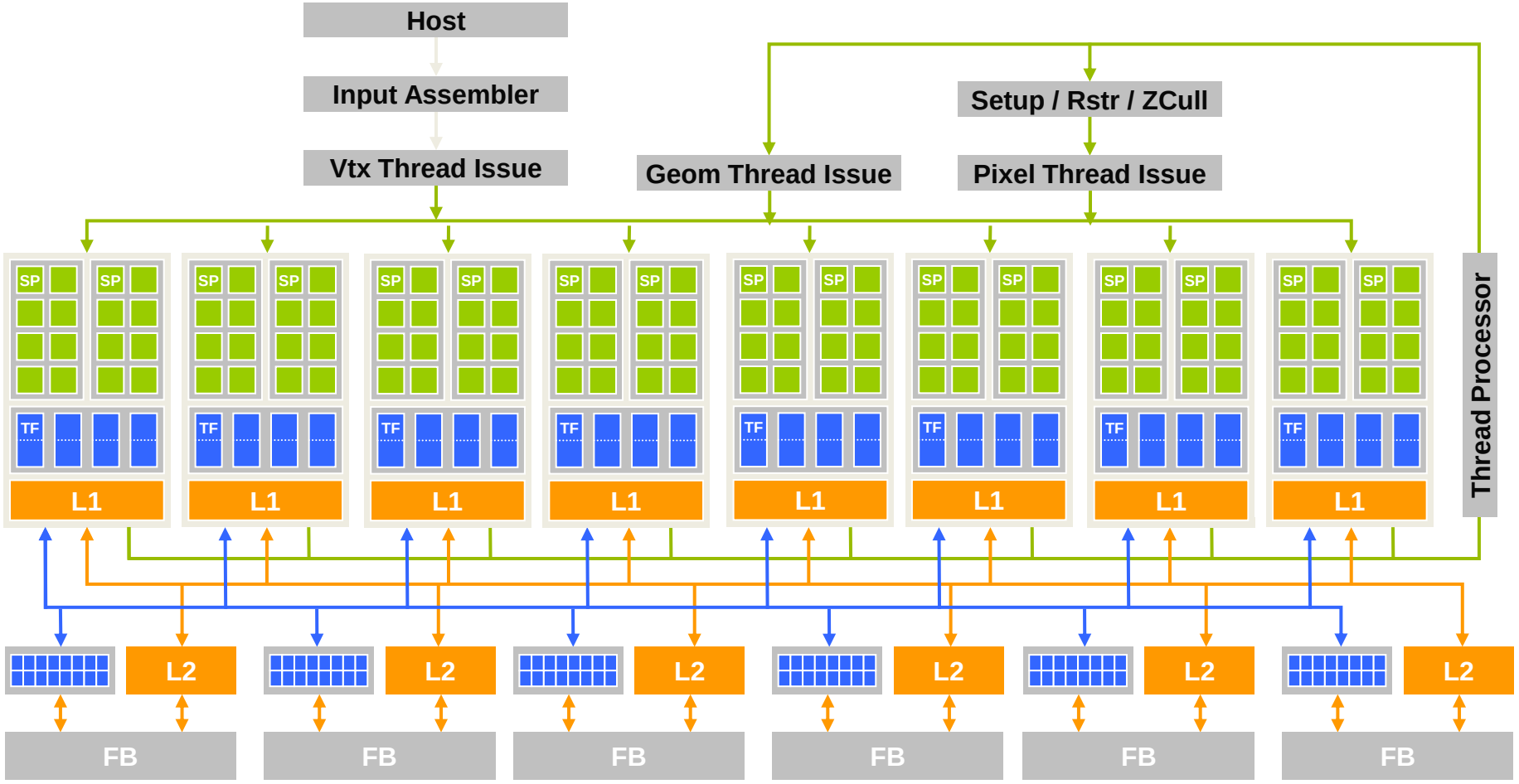
- NVIDIA GeForce 8800 mapped separate graphics stage to a unified array of processors
  - For vertex shading, geometry processing, and pixel processing
  - Allows dynamic partition

# Regularity + Massive Parallelism







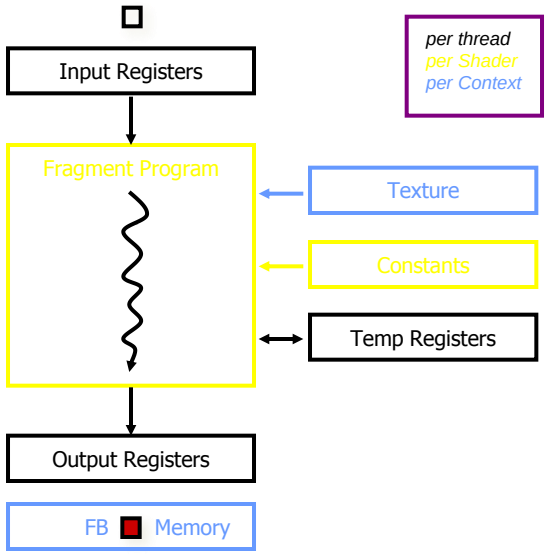

Exploring the use of GPUs to solve compute intensive problems

The birth of GPGPU but there are many constraints

GPUs and associated APIs were designed to process graphics data

# Previous GPGPU Constraints

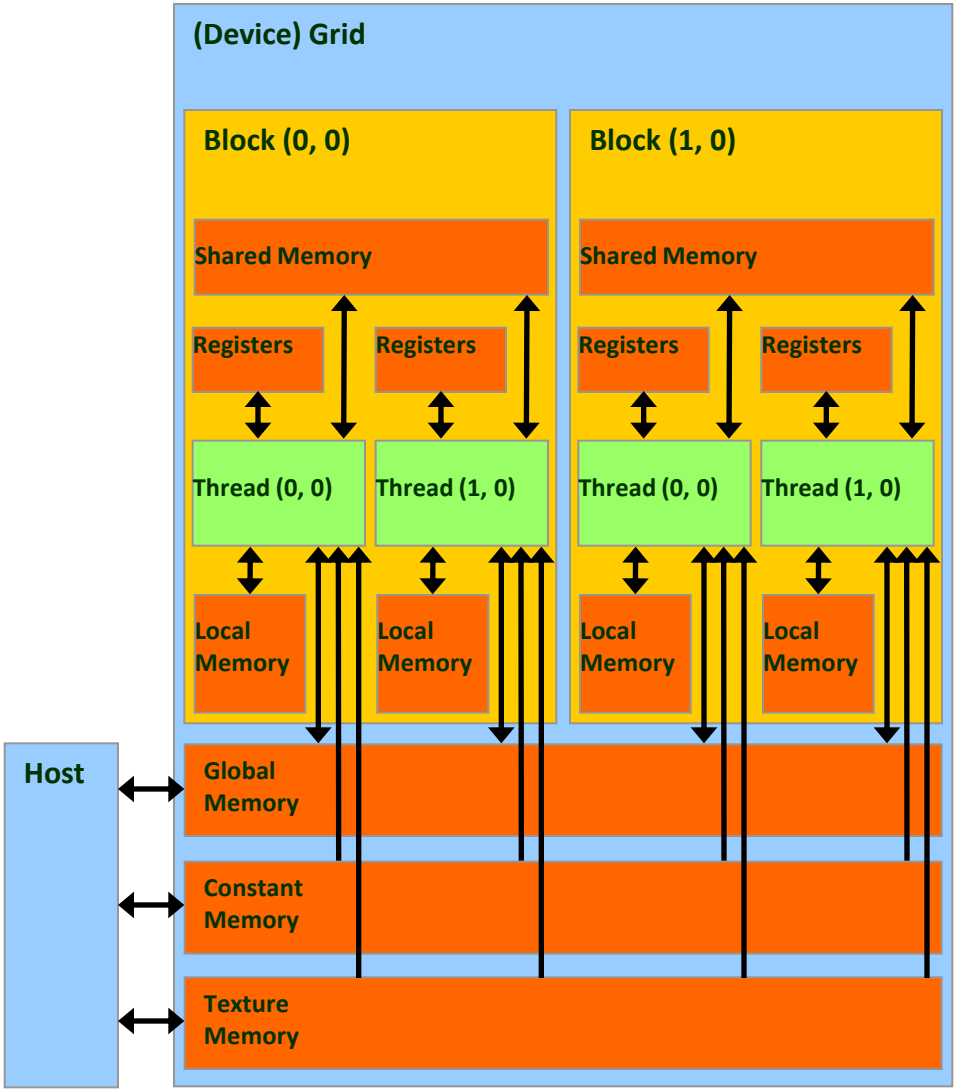
- Dealing with graphics API
  - Working with the corner cases of the graphics API
- Addressing modes
  - Limited texture size/dimension
- Shader capabilities
  - Limited outputs
- Instruction sets
  - Lack of Integer & bit ops
- Communication limited
  - Between pixels
  - Scatter a[i] = p



# The Birth of GPU Computing

- **Step 1**: Designing high-efficiency floating-point and integer processors.
- **Step 2**: Exploiting data parallelism by having large number of processors
- **Step 3**: Shader processors fully programmable with large instruction cache, instruction memory, and instruction control logic.
- **Step 4**: Reducing the cost of hardware by having multiple shader processors to share their cache and control logic.
- **Step 5**: Adding memory load/store instructions with random byte addressing capability
- **Step 6**: Developping CUDA C/C++ compiler, libraries, and runtime software models.

# A Glimpse on Memory Space

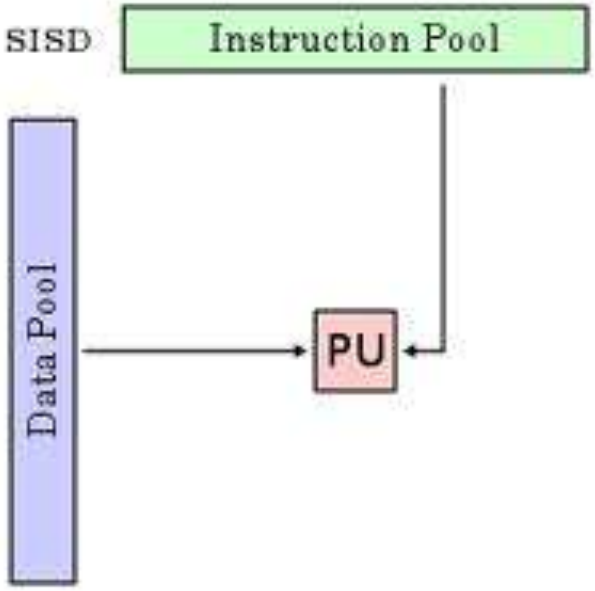


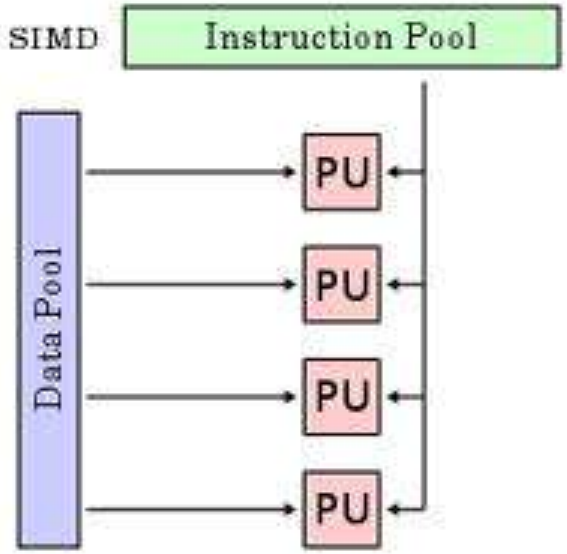

Source : "NVIDIA CUDA Programming Guide" version 1.1

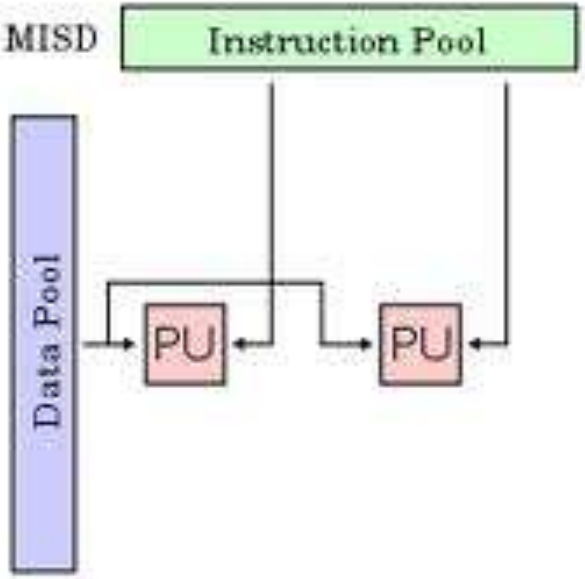
# A Quick Glimpse on: Flynn Classification

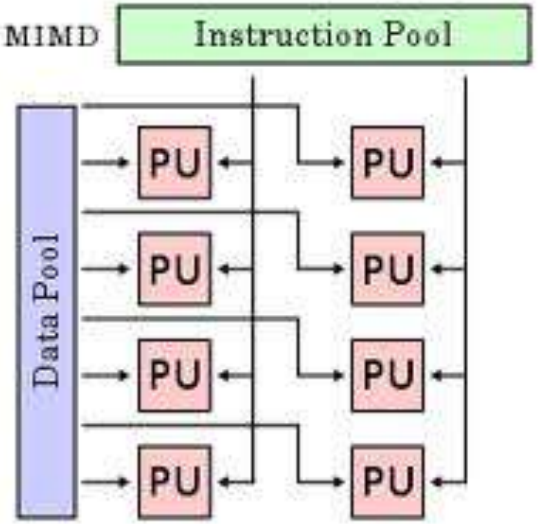
- A taxonomy of computer architecture
- Proposed by Micheal Flynn in 1966
- It is based two things:
  - Instructions
  - Data

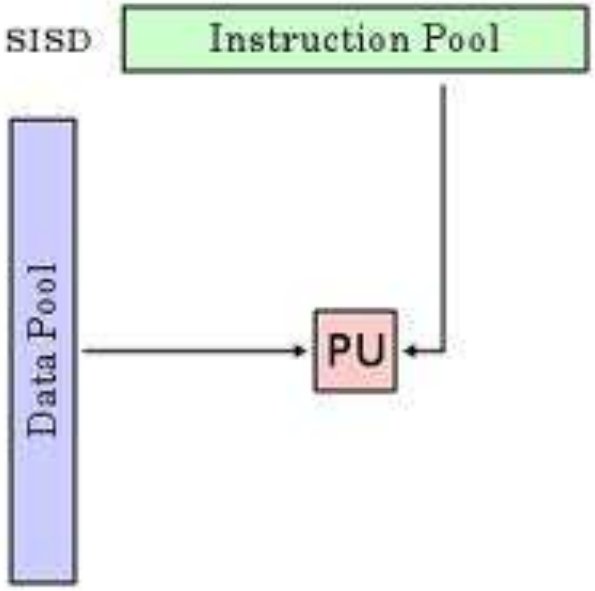
| | Single instruction | Multiple instruction |
|---|---|---|
| Single data | SISD | MISD |
| Multiple data | SIMD | MIMD |

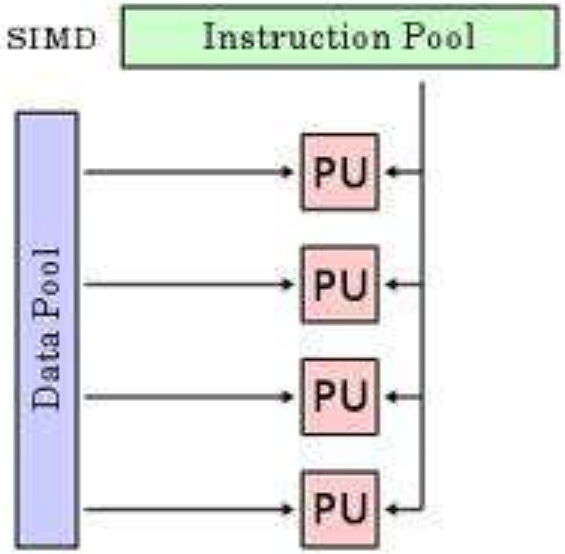

SISD
Instruction Pool
Data Pool
PU


SIMD
Instruction Pool
Data Pool
PU
PU
PU
PU

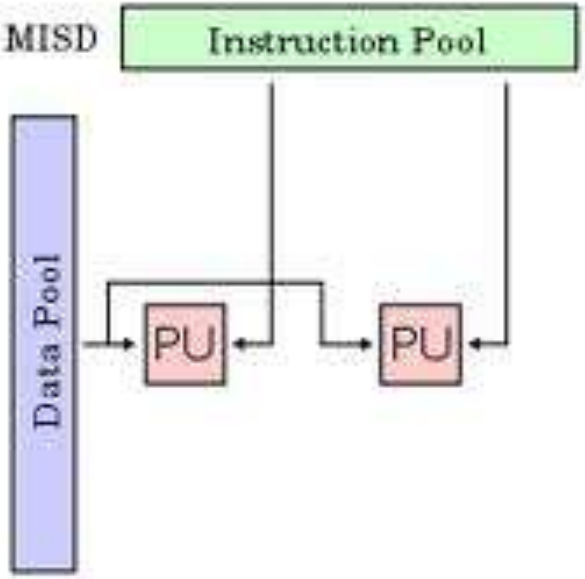

MISD
Instruction Pool
Data Pool
PU
PU

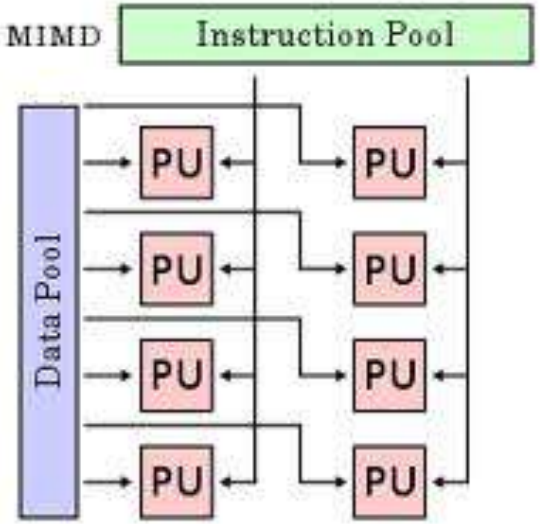

MIMD
Instruction Pool
Data Pool
PU
PU
PU
PU
PU
PU
PU
PU



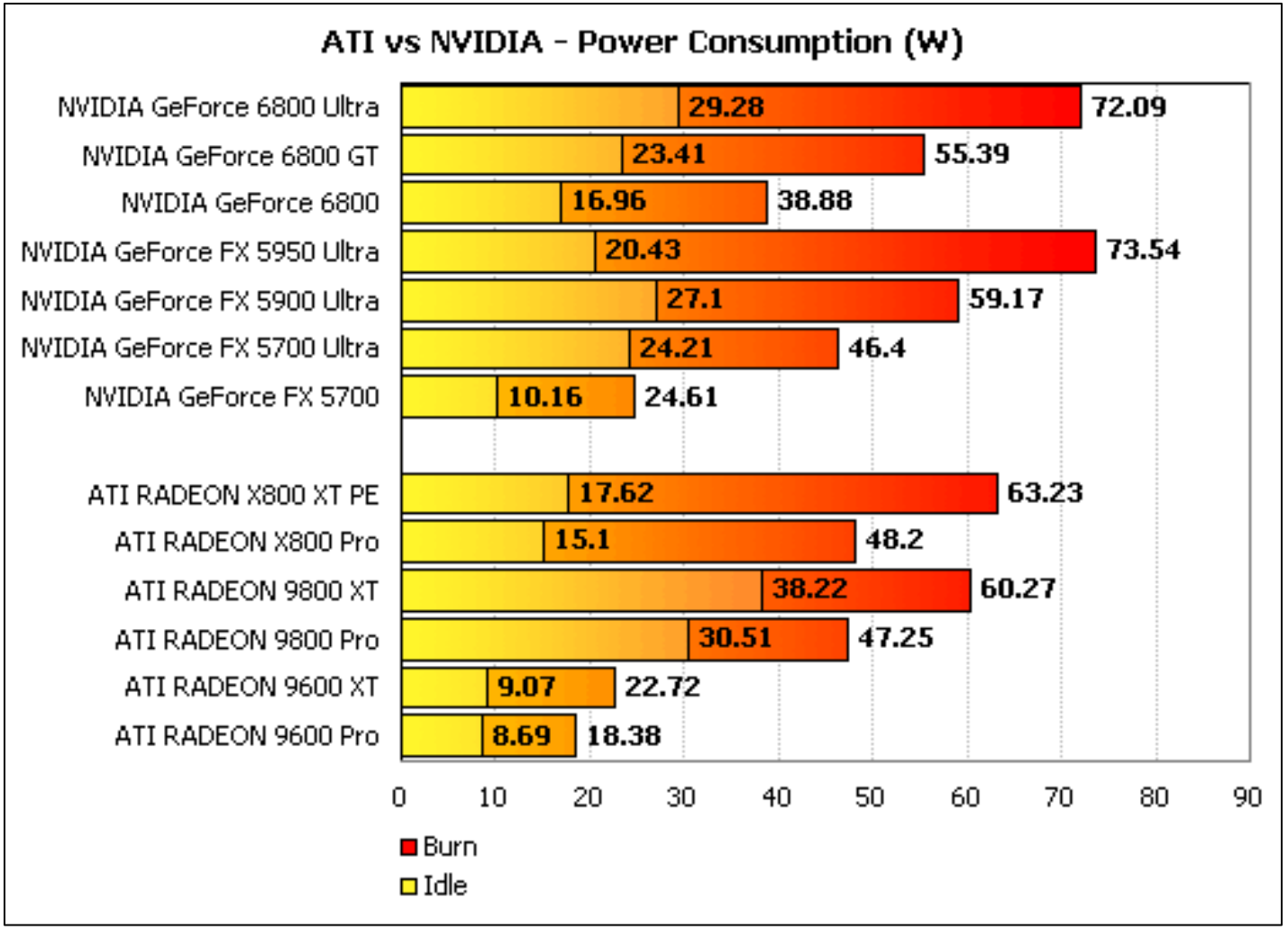




Which one is closest to GPU?



# Problem With GPUs: Power



# Problems with GPUs

- Need enough parallelism
- Under-utilization
- Bandwidth to CPU

**Still a way to go**

# Conclusions

- The design of state-of-the art GPUs includes:
  - Data parallelism
  - Programmability
  - Much less restrictive instruction set