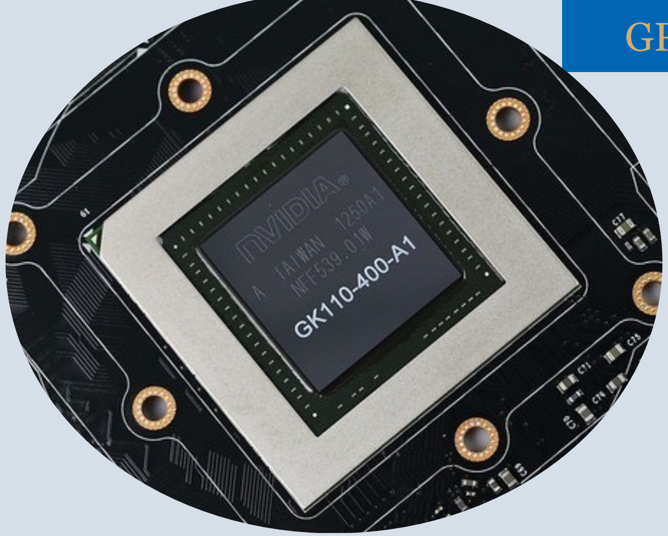
Lec 1 Hello GPU

Tonghua Su
School of Software
Harbin Institute of Technology

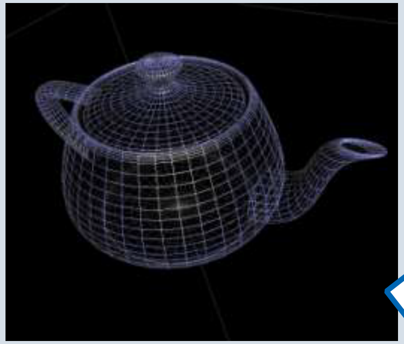
Outline

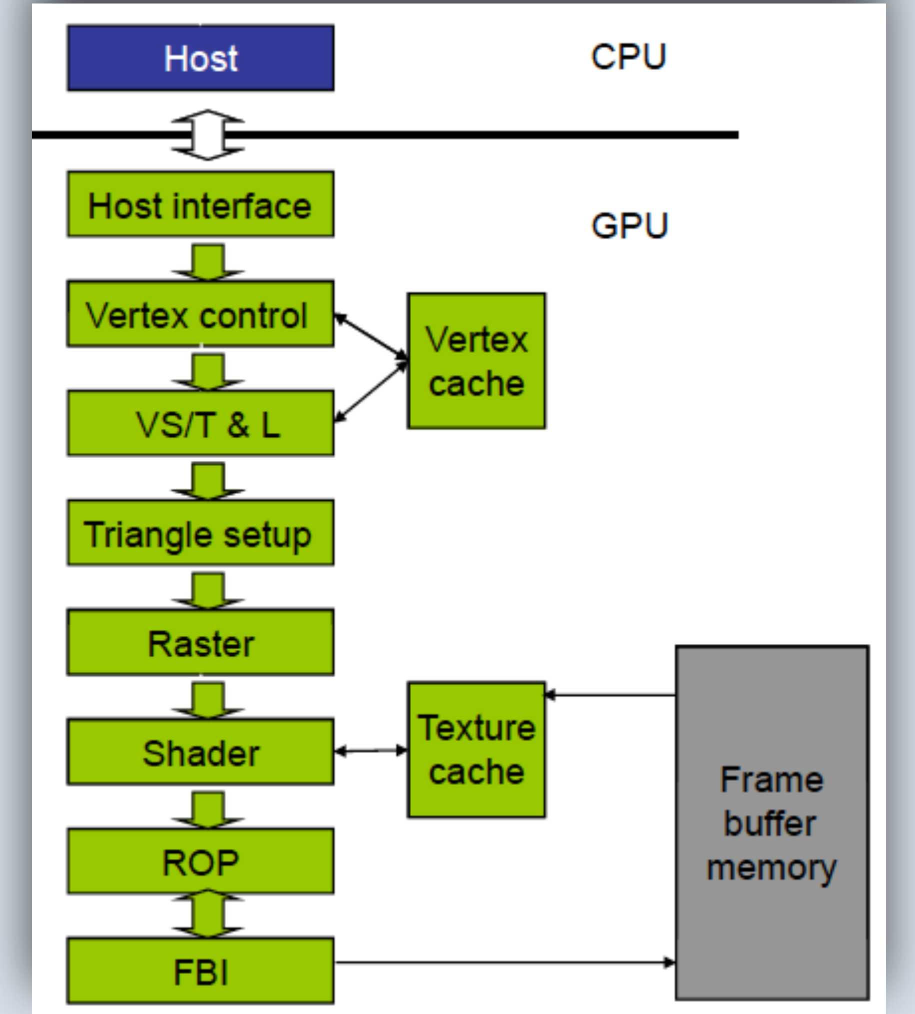
1. What is GPU?
2. GPU Architecture (briefly)
3. First GPU Program
4. Amdahl's Law
5. HW1



What is GPU

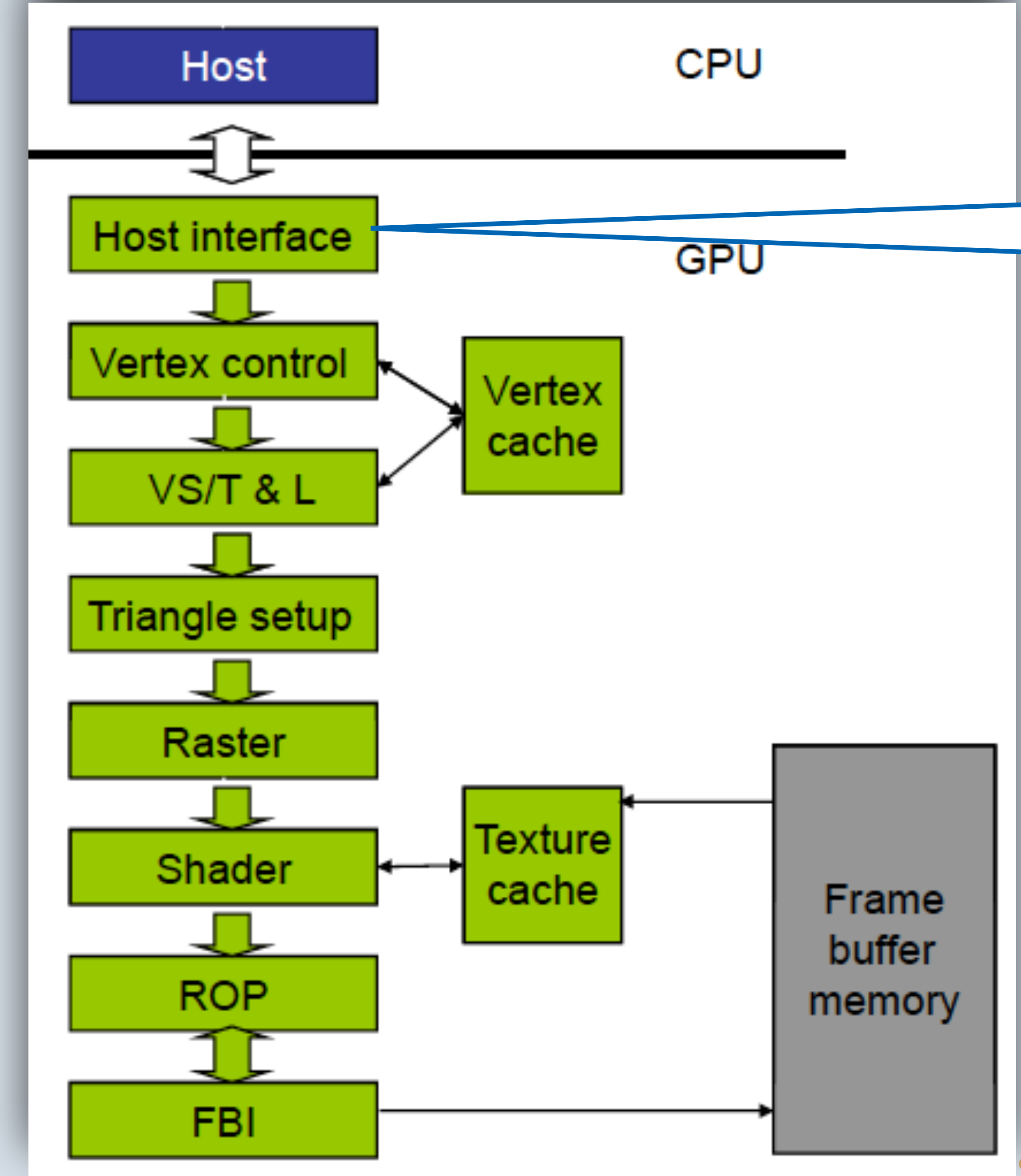
- **Specialized electronic circuit**
  - ✓ Accelerate the building of images intended for display





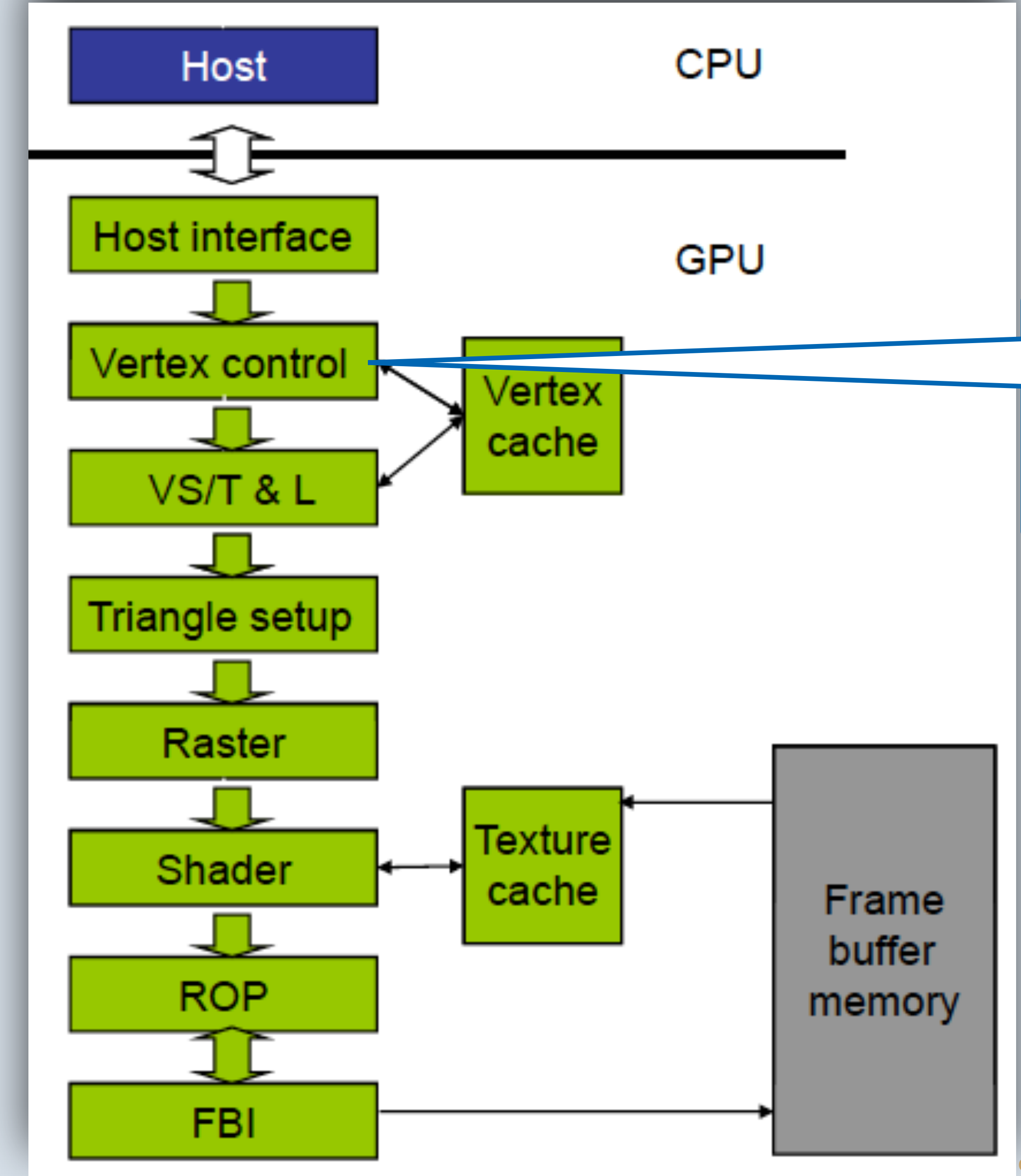

```
0.748952 -0.764952 -0.210132,
0.872246 -0.600062 -0.210132,
1.00016 -0.369696 -0.210132,
1.00939 -0.004004 -0.210132,
1.14496 0.324436 -0.210132,
1.15747 0.601712 -0.210132,
1.08016 0.793529 -0.210132,
0.98164 0.872032 -0.210132,
0.808263 0.929016 -0.210132,
0.442563 0.985585 -0.210132,
0.221794 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,
```

What is GPU



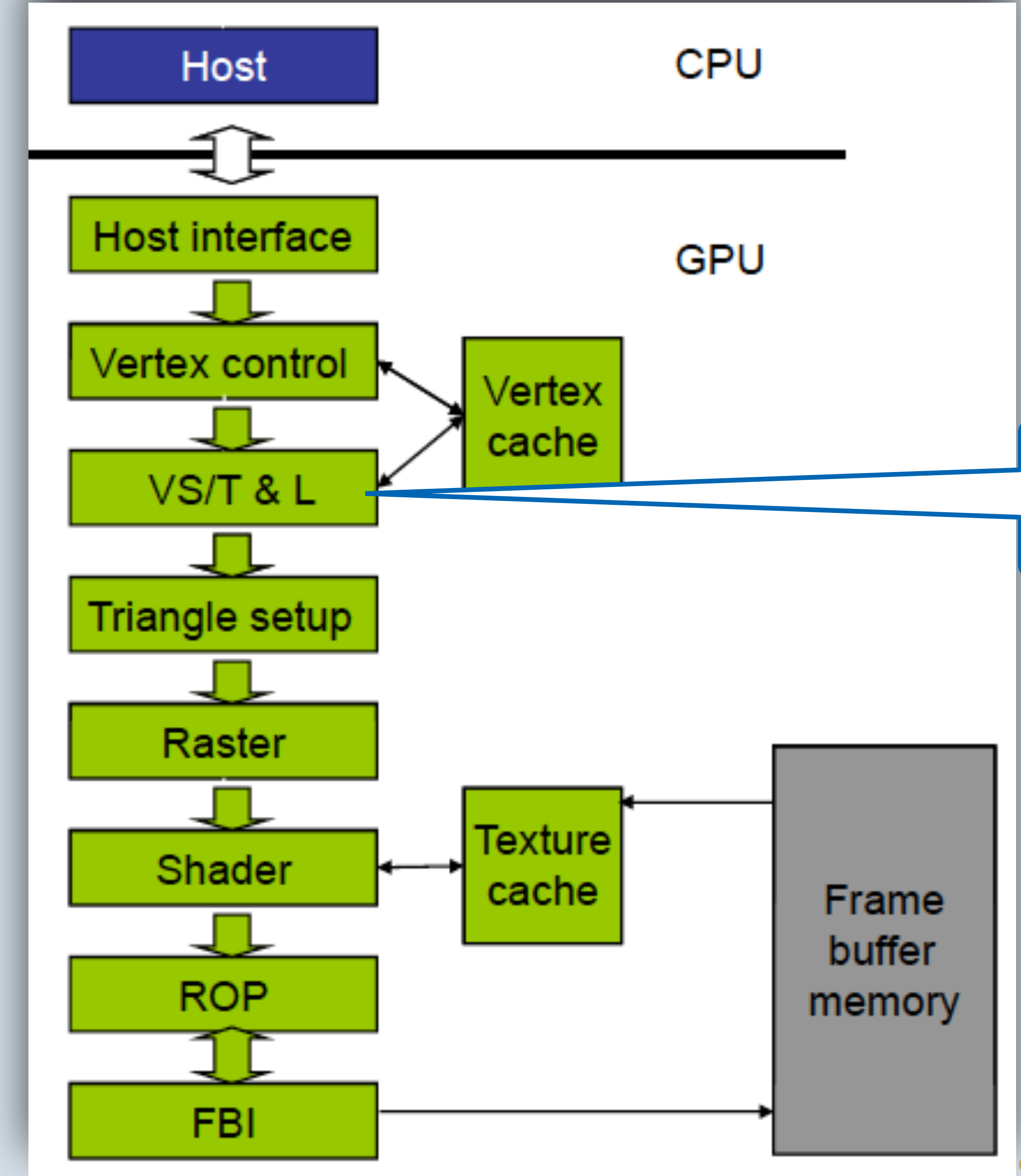
Receives graphics commands and data from CPU

What is GPU



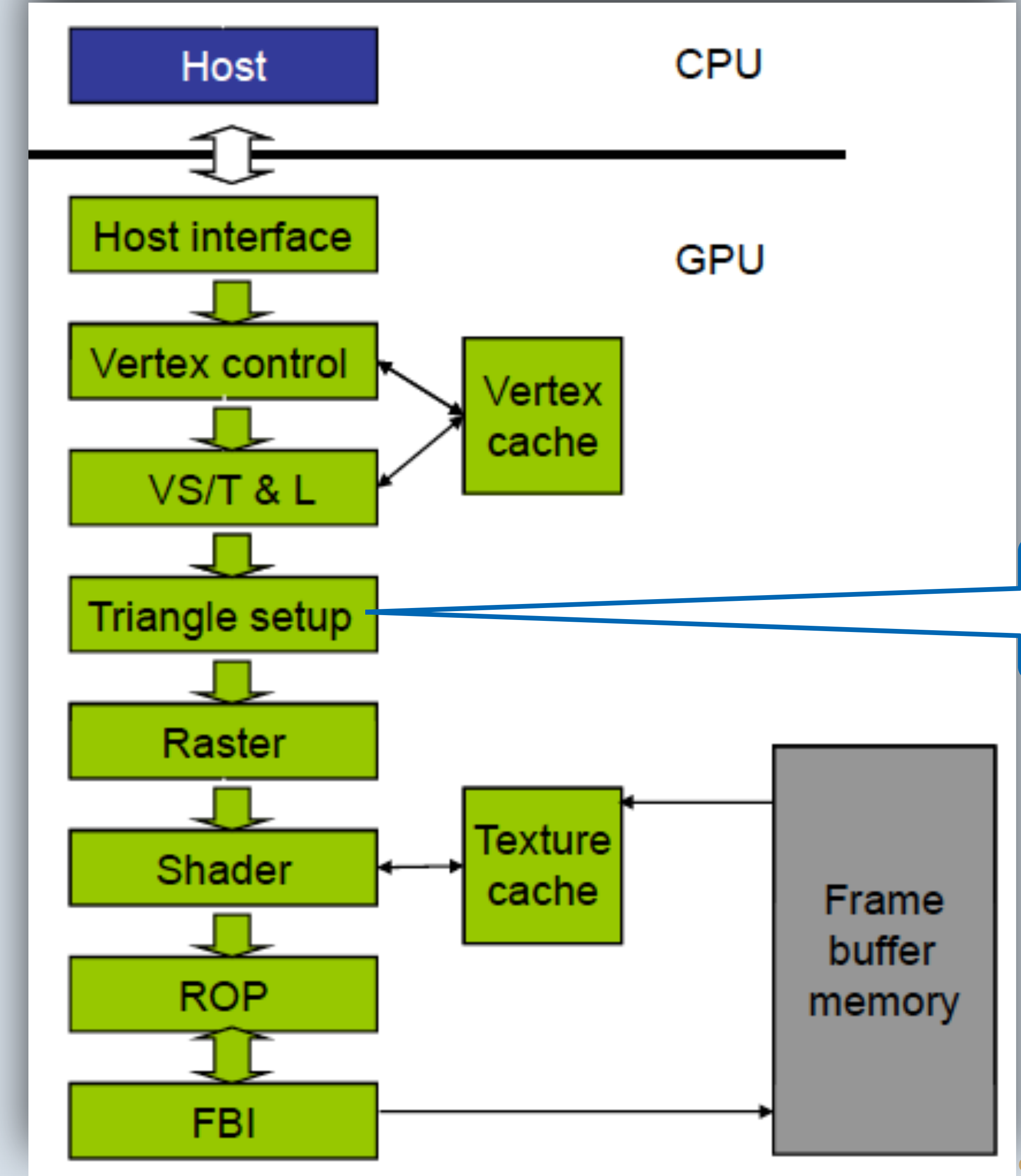
Receives triangle data
Converts them into a form that hardware understands
Store the prepared data in vertex cache

What is GPU



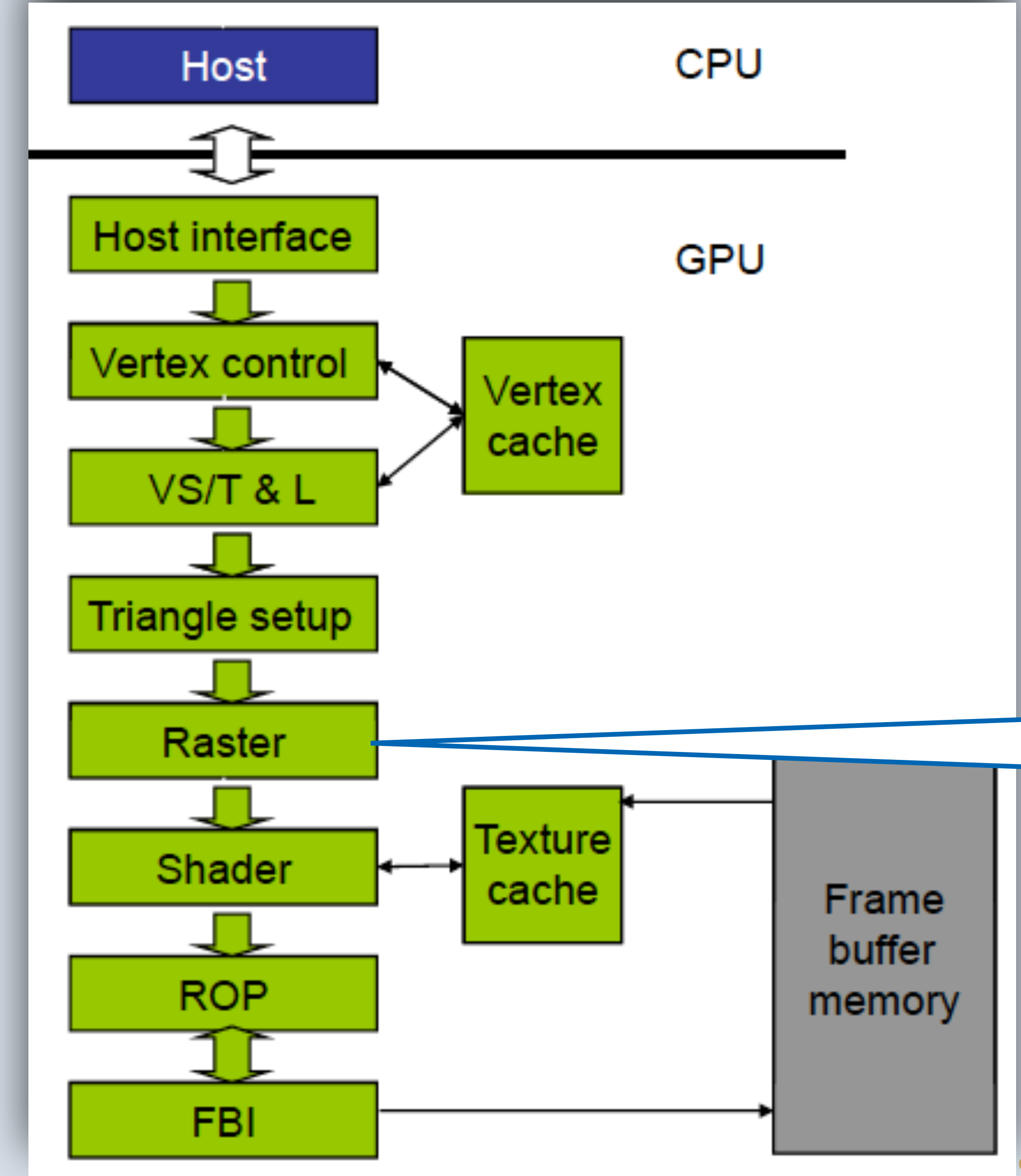
- Vertex shading transform and lighting
- Assigns per-vertex value

What is GPU



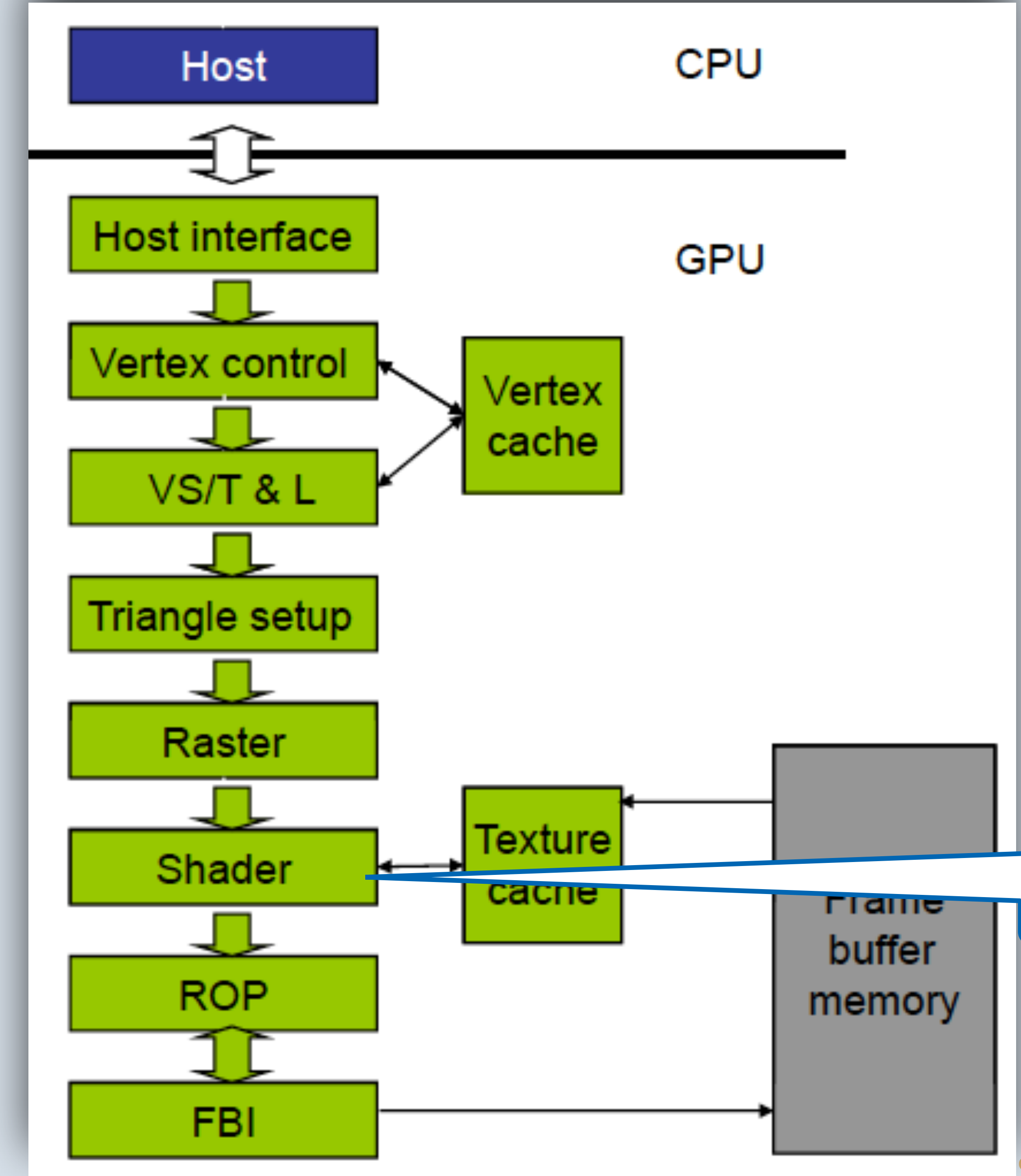
Creates edge equations to interpolate colors across pixels touched by the triangle

What is GPU



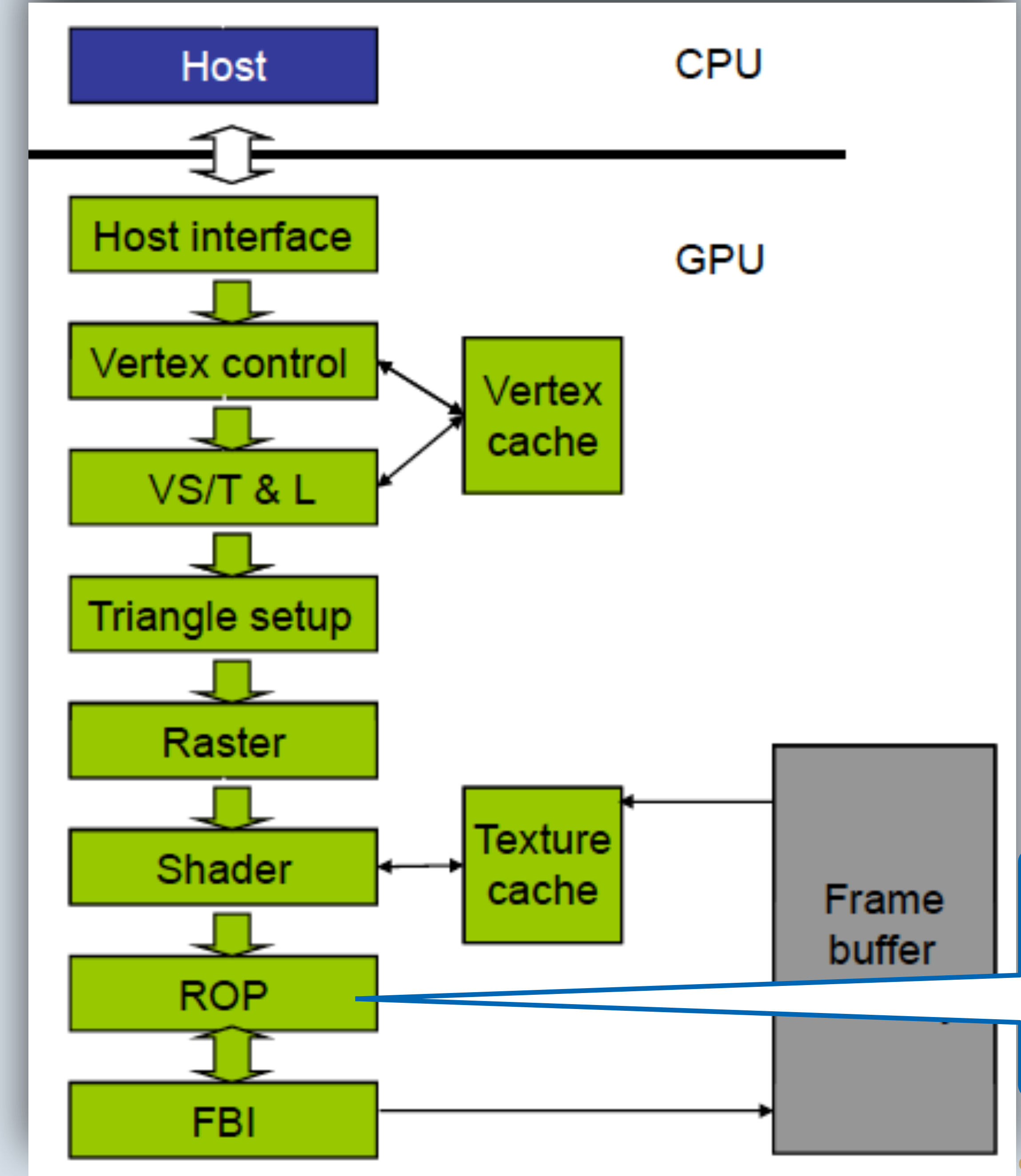
- Determines which pixel falls in which triangle
- For each pixel, interpolate per-pixel values

What is GPU



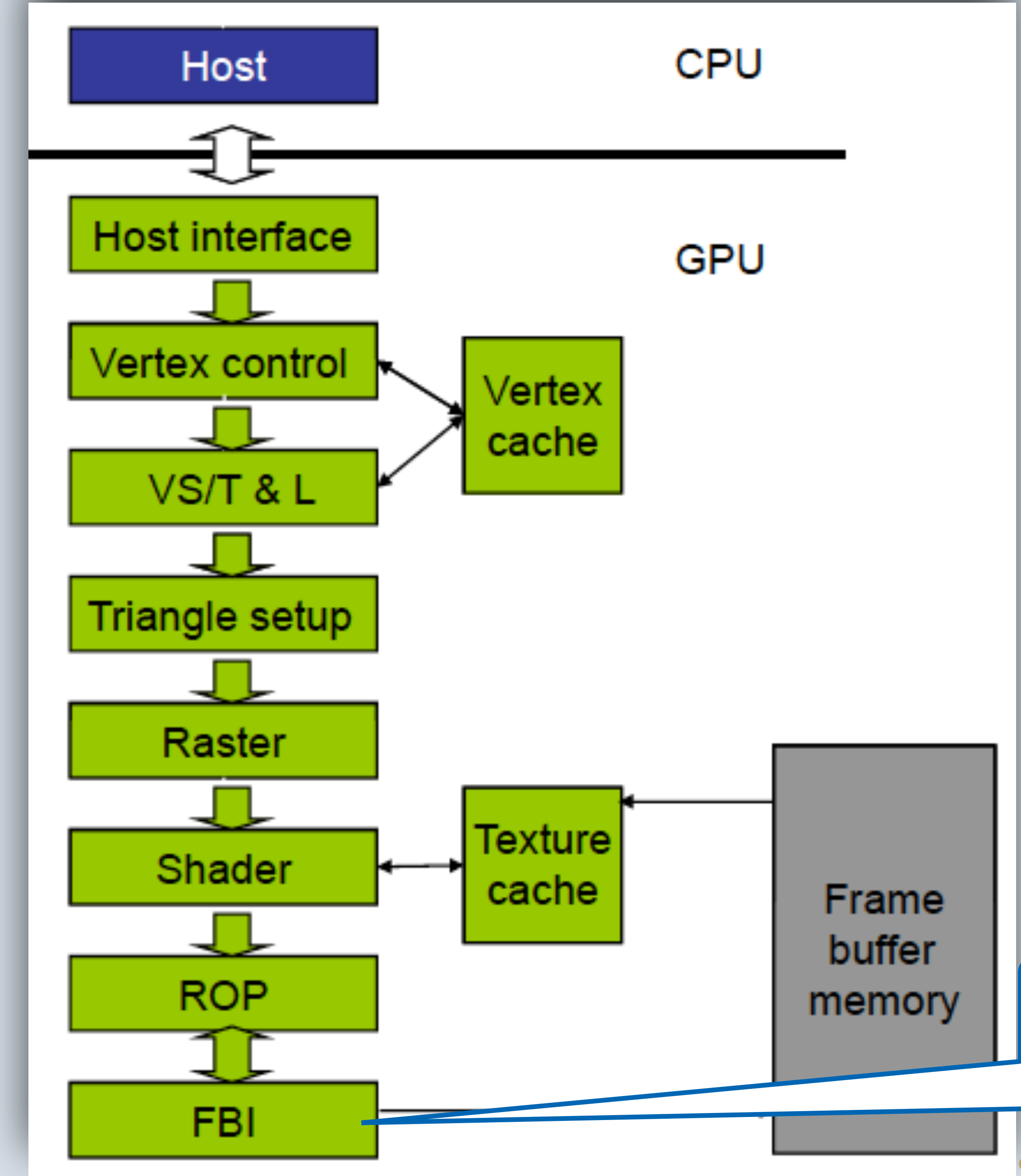
Determines the final color of each pixel

What is GPU



Determine The raster operation: performs color raster operations that blend the color of overlapping objects for transparency and antialiasings the final color of each pixel

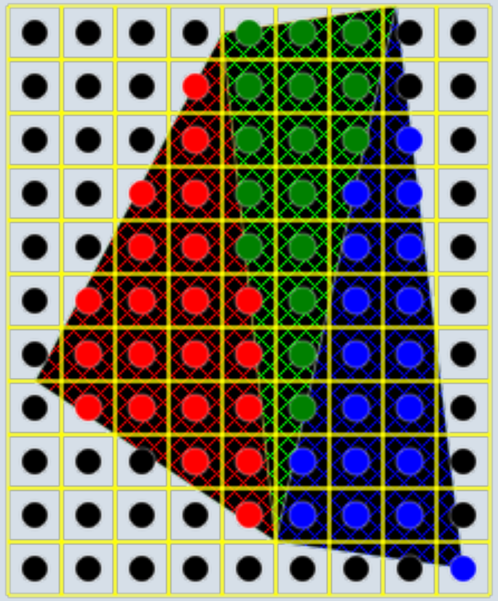
What is GPU



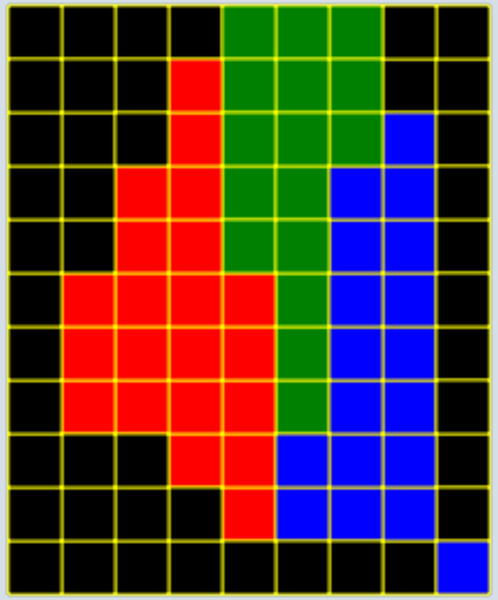
The frame buffer interface manages memory reads/writes

What is GPU

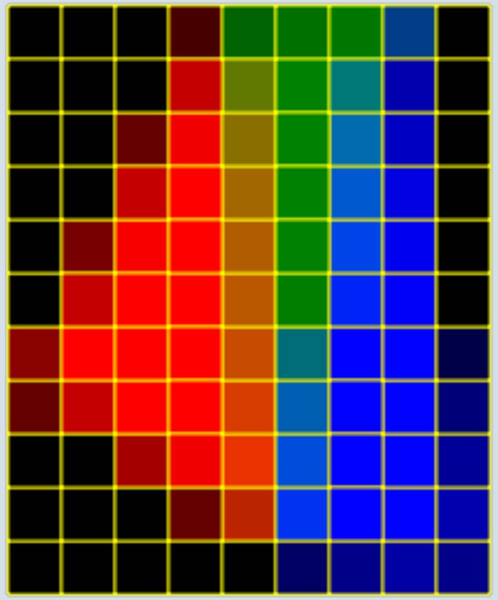
- **Specialized electronic circuit**
  - ✓ Accelerate the building of images intended for display



Triangle Geometry



Aliased

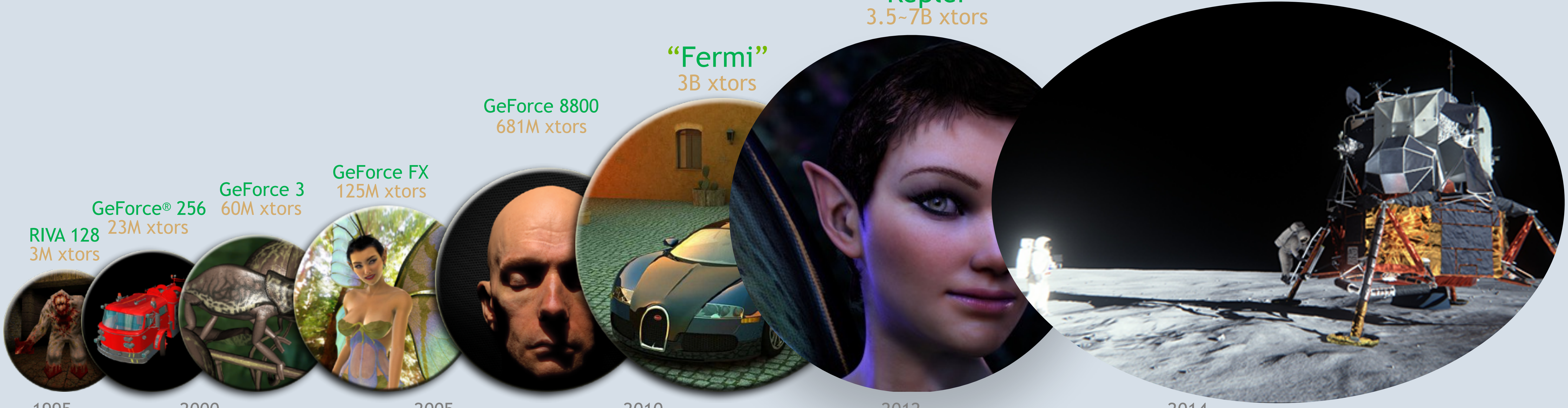


Anti-Aliased

- ✓ But now, it is used for general-purpose computation!

What is GPU

- **High throughput computation**
  - ✓ GeForce GTX 980: 4,612 GFLOP/s
- **High bandwidth memory**
  - ✓ GeForce GTX 980: 224 GB/s
- **High availability to all**
  - ✓ 500+ million CUDA-capable GPUs in the wild



Flynn Taxonomy

- **Flynn分类法(行为特征)**

S I S D ***Single Instruction, Single Data*** 串行计算机*(von Neumann*计算机*)*	***S I M D*** ***Single Instruction, Multiple Data*** 特定领域的加速器*(GPU*、向量处理机等*)*
M I S D ***Multiple Instruction, Single Data*** 比较少见，多用于容错计算机	***M I M D*** ***Multiple Instruction, Multiple Data*** 常见的并行计算机都可归入此类 *MPP/Cluster/SMP/*当前基于*Cache*的 *Multi-core (Intel*、*AMD)*

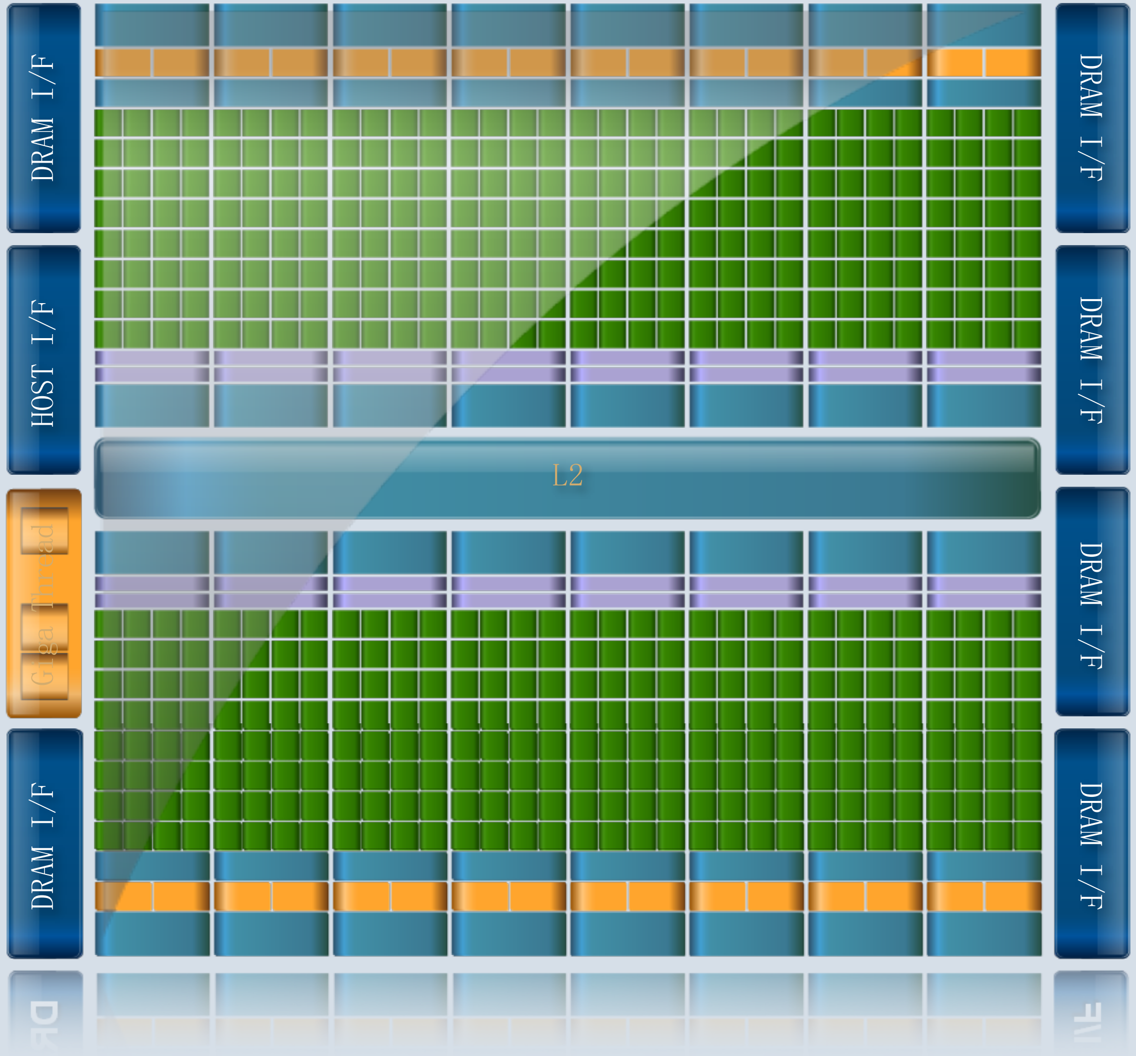
Quiz

- **Pascal 架构、Volta架构与Turing架构有多少晶体管?**
- **对比CPU与GPU浮点计算能力**
  - **CPU以E5 2640 V3为例，~300GFLOPS**
  - **GPU以GTX 980为例**

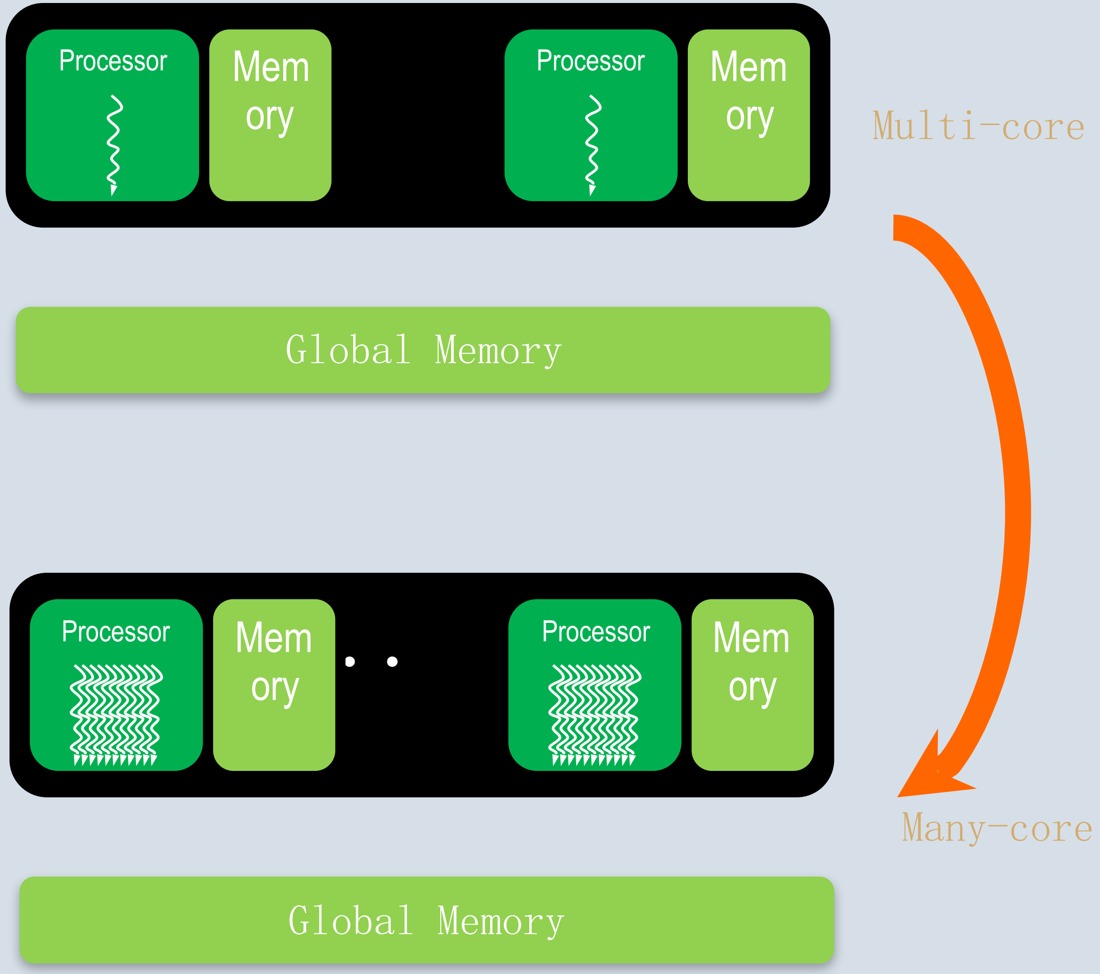
Outline

1. What is GPU?
2. GPU Architecture (briefly)
3. First GPU Program
4. Amdahl’s Law
5. HW 1

NVIDIA GPU Architecture

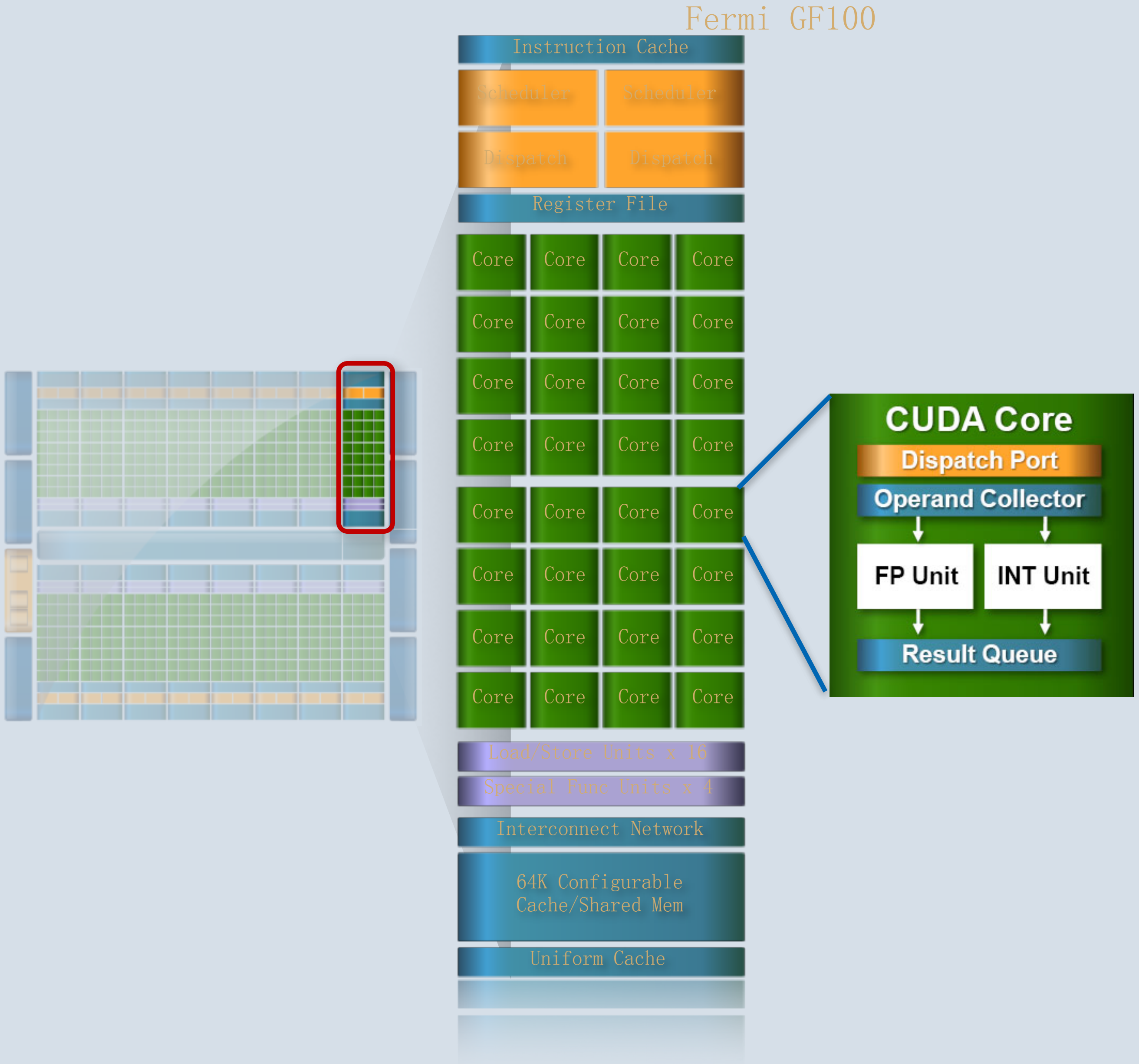


Fermi GF100



Streaming Multi-processor (SM)

- **32 CUDA Cores per SM (512 total)**
- **Direct load/store to memory**
  - √ Usual linear sequence of bytes
  - √ High bandwidth (Hundreds GB/sec)
- **64KB of fast, on-chip RAM**
  - √ Software or hardware-managed
  - √ Shared amongst CUDA cores
  - √ Enables thread communication



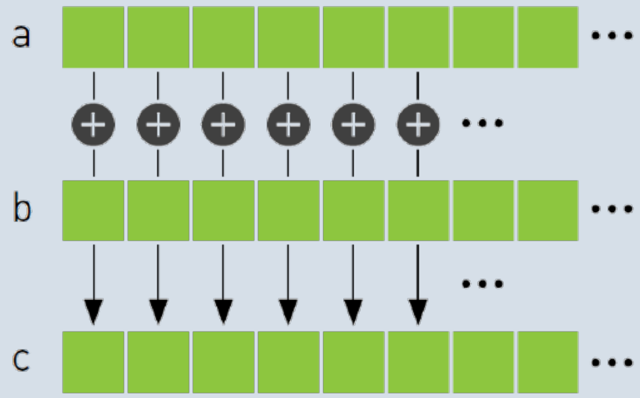
HW 1.1 Device Query

- **查询你机器上GPU设备的参数**
  - ✓ 新建.cu文件
  - ✓ 调用cudaGetDeviceCount()得到GPU设备的数量
  - ✓ 调用cudaGetDeviceProperties()函数得到GPU设备的属性结构体
  - ✓ 解释关键属性的含义，至少包括设备名称、计算能力为多少、设备可用全局内存、每线程块最大线程数、设备可用全局内存容量、每线程块可用共享内存容量、每线程块可用寄存器数量、每线程块最大线程数、每个处理器簇最大驻留线程数、设备中的处理器簇数量等
  - ✓ 可参考WILT 3.2节

Outline

Hello CUDA

- **Vector Sum**



- **Serial code**

```
for (i=0;i<128;i++)
{
        c[i] = a[i] + b[i];
}
```

- **Translate into CUDA threads**

```
__global__ void addKernel(int * const a, const int * const b, const int * const c)
{
        c[i] = a[i] + b[i];
}
```

Hello CUDA

●Vector Sum

✓ Identify thread id

```
__global__ void addKernel(int * const a, const int * const b, const int * const c)
{
        const unsigned int i = threadIdx.x;
        c[i] = a[i] + b[i];
}
```

✓ Invoke CUDA kernel

- kernel_function<<<num_blocks, num_threads>>>(param1, param2, .)
- E.g. addKernel<<< 1, 128 >>>(a, b, c);

Hello GPU

- **Demo 1**

Hello CUDA

```
__global__ addKernel(int * const a, const int * const b, const int * const c)
{
        const unsigned int i = threadIdx.x;
        c[i] = a[i] + b[i];
}
```

线程ID，同时索引数据元素

```
void main(){
        ......
        int *dev_a,*dev_b,*dev_c;
        // Allocate GPU buffers for three vectors (two input, one output)    .
        cudaMalloc((void**)&dev_c, 128* sizeof(int));
        ......
        // Copy input vectors from host memory to GPU buffers.
        cudaMemcpy(dev_a, a, 128* sizeof(int), cudaMemcpyHostToDevice);
        cudaMemcpy(dev_b, b, 128* sizeof(int), cudaMemcpyHostToDevice);

        // Launch a kernel on the GPU with one thread for each element.
        addKernel<<<1, 128>>>(dev_c, dev_a, dev_b);

        // Copy output vector from GPU buffer to host memory.
        cudaMemcpy(c, dev_c, 128* sizeof(int), cudaMemcpyDeviceToHost);

        cudaFree(dev_c);
        ......
}
```

分配显存

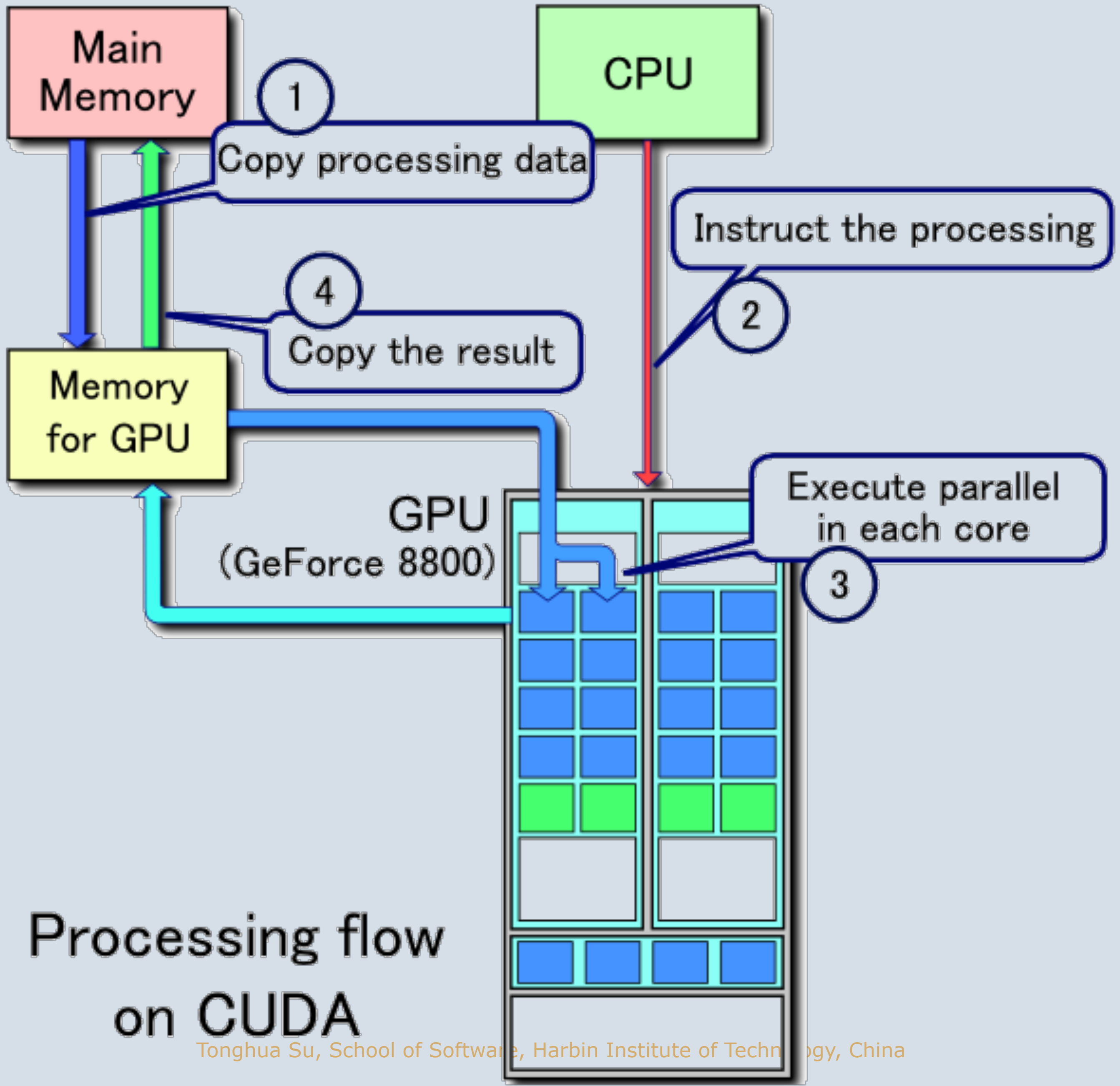
数据从主机复制到GPU

调用内核函数addKernel

数据从GPU复制回主机

释放显存

Hello CUDA


Main Memory
CPU
1
Copy processing data
Instruct the processing
2
4
Copy the result
Memory for GPU
GPU (GeForce 8800)
Execute parallel in each core
3
Processing flow on CUDA

HW 1.2 第一个GPU程序

- **自己从头编写并运行VectorSum内核**
-

Outline

1. What is GPU?
2. GPU Architecture (briefly)
3. First GPU Program
4. Amdahl’s Law
5. HW 1

Amdahl's Law

- **Gene Amdahl in 1967:**

$$Speedup = \frac{1}{r_s + \frac{r_p}{N}}$$

where $r_s + r_p = 1$ and r_s represents the ratio of the sequential portion.

✓ **Consider: 30% portion of time with 100X speedup through paralleling vs 99% portion with 100X**

Amdahl, Gene (1967). "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities". *AFIPS Conference Proceedings* (30): 483–485.

Quiz

- **请使用Amdahl 的公式计算下面程序的加速比**
  - ✓ 程序99%的部分可以使用无限多个核心计算，剩余的1%无法并行

Outline

1. What is GPU?
2. GPU Architecture (briefly)
3. First GPU Program
4. Amdahl’s Law
5. HW 1

HW 1

- **1.1 查询你机器上GPU设备的参数**
  - ✓ 新建**.cu**文件
  - ✓ 调用**cudaGetDeviceCount()**得到**GPU**设备的数量
  - ✓ 调用**cudaGetDeviceProperties()**函数得到**GPU**设备的属性结构体
  - ✓ 解释关键属性的含义，至少包括设备名称、计算能力为多少、设备可用全局内存、每线程块最大线程数、设备可用全局内存容量、每线程块可用共享内存容量、每线程块可用寄存器数量、每线程块最大线程数、每个处理器簇最大驻留线程数、设备中的处理器簇数量等
  - ✓ 可参考**WILT 3.2**节
- **1.2 自己从头编写并运行VectorSum内核**