Lecture 8:
The GPU Memory Hierarchy

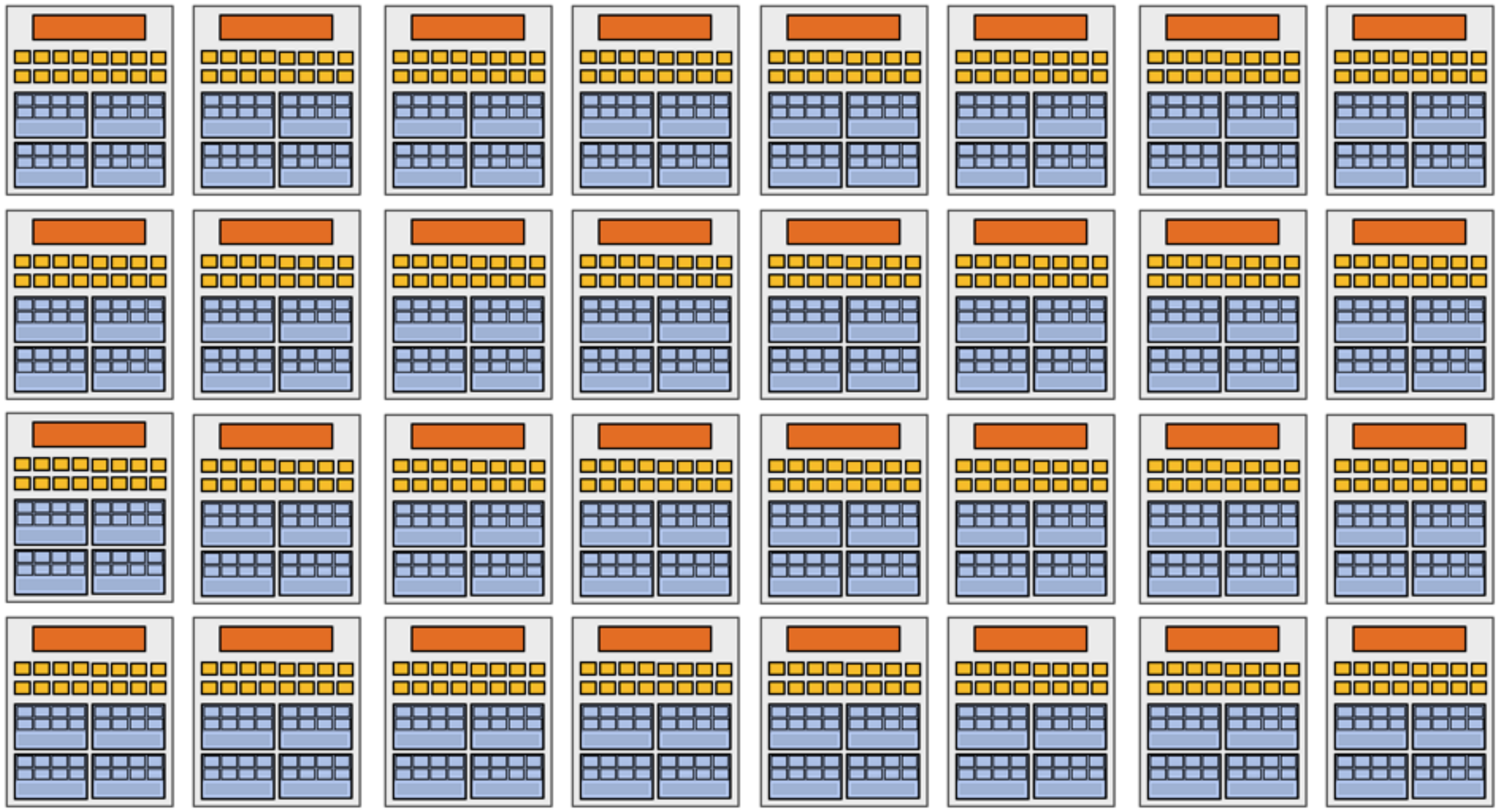
Kayvon Fatahalian

CMU 15-869: Graphics and Imaging Architectures (Fall 2011)

Last time

GPUs contain a collection of programmable processing cores: responsible for carrying out data-parallel stages of the graphics pipeline (vertex, fragment, primitive processing)

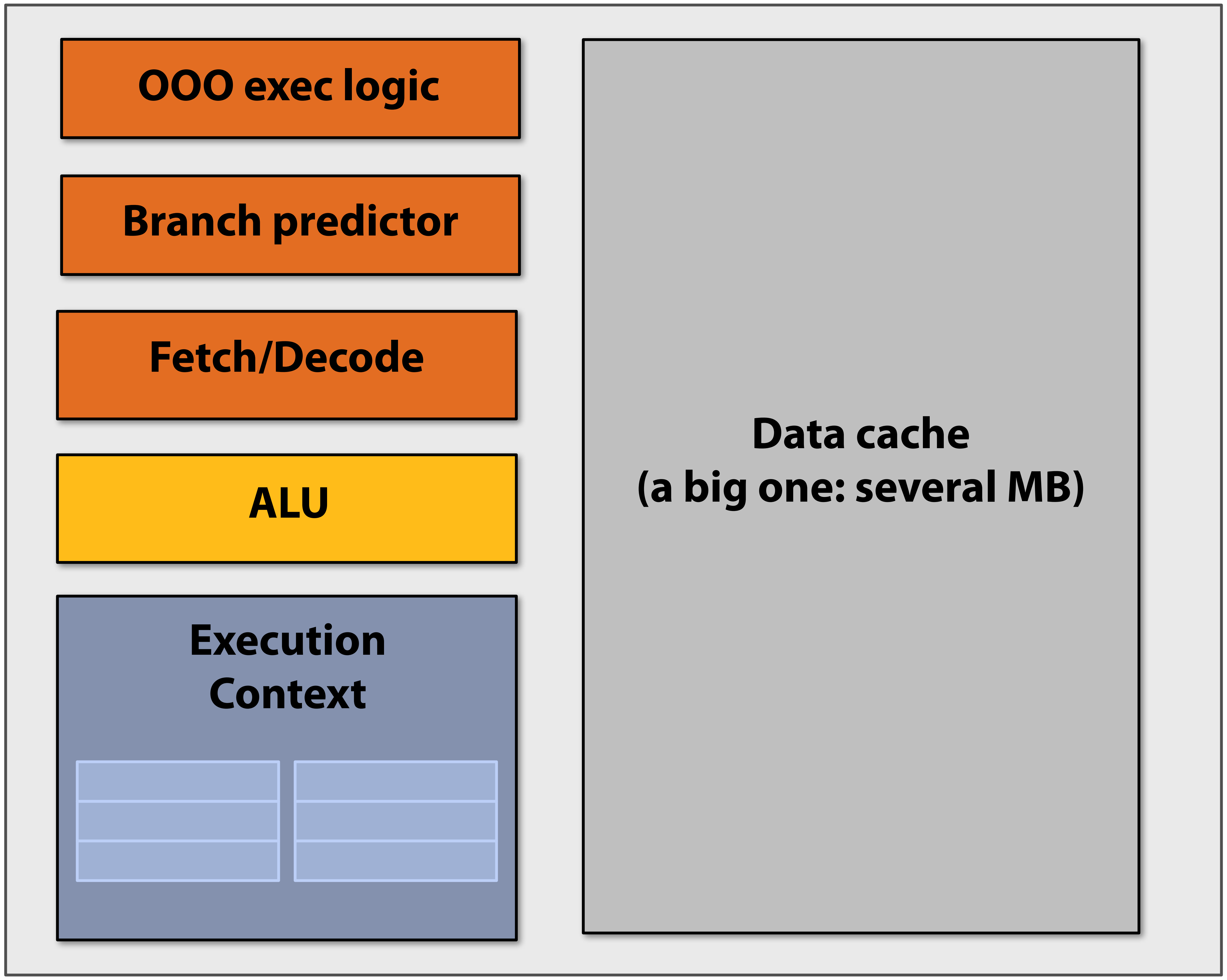
- **Many processing cores**
- **SIMD execution**
- **Hardware support for large-scale multi-threading**



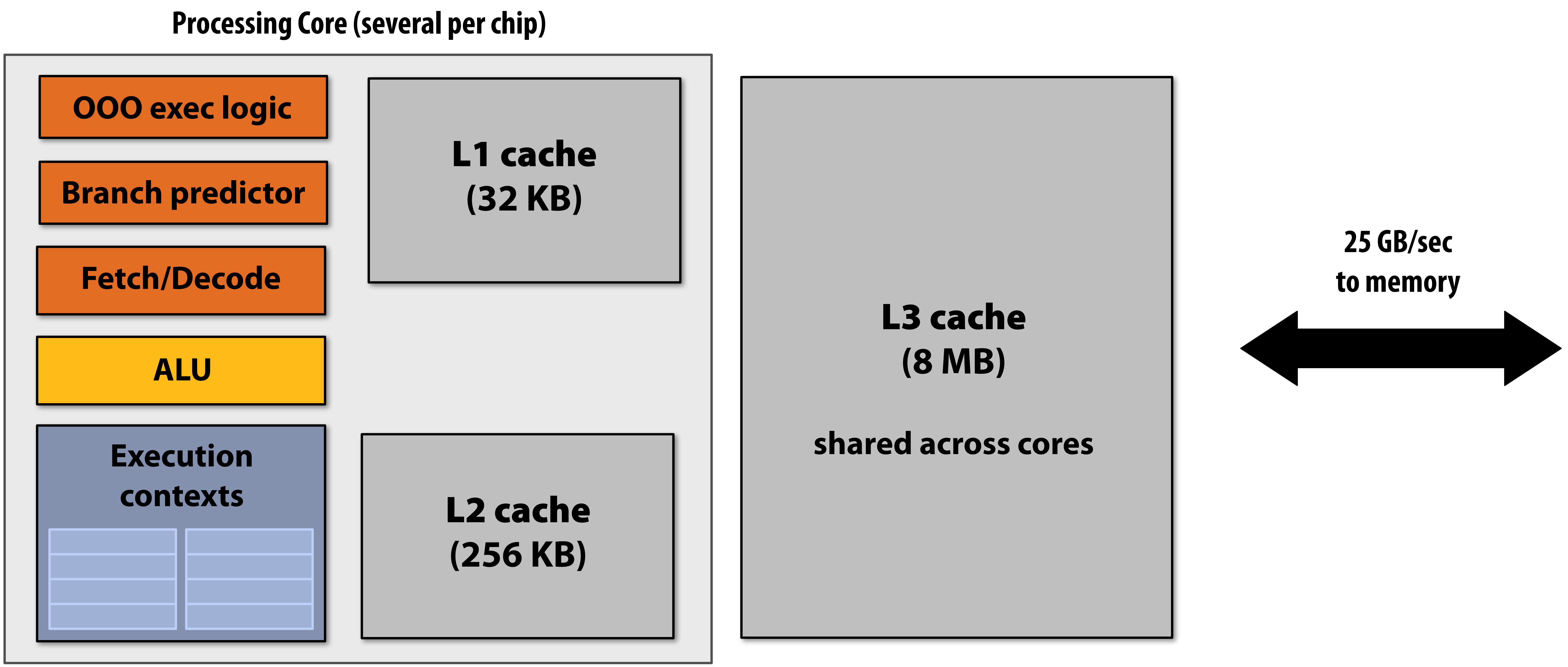
Last time: processing data

Now: moving data to processors

Recall: "CPU-style" core

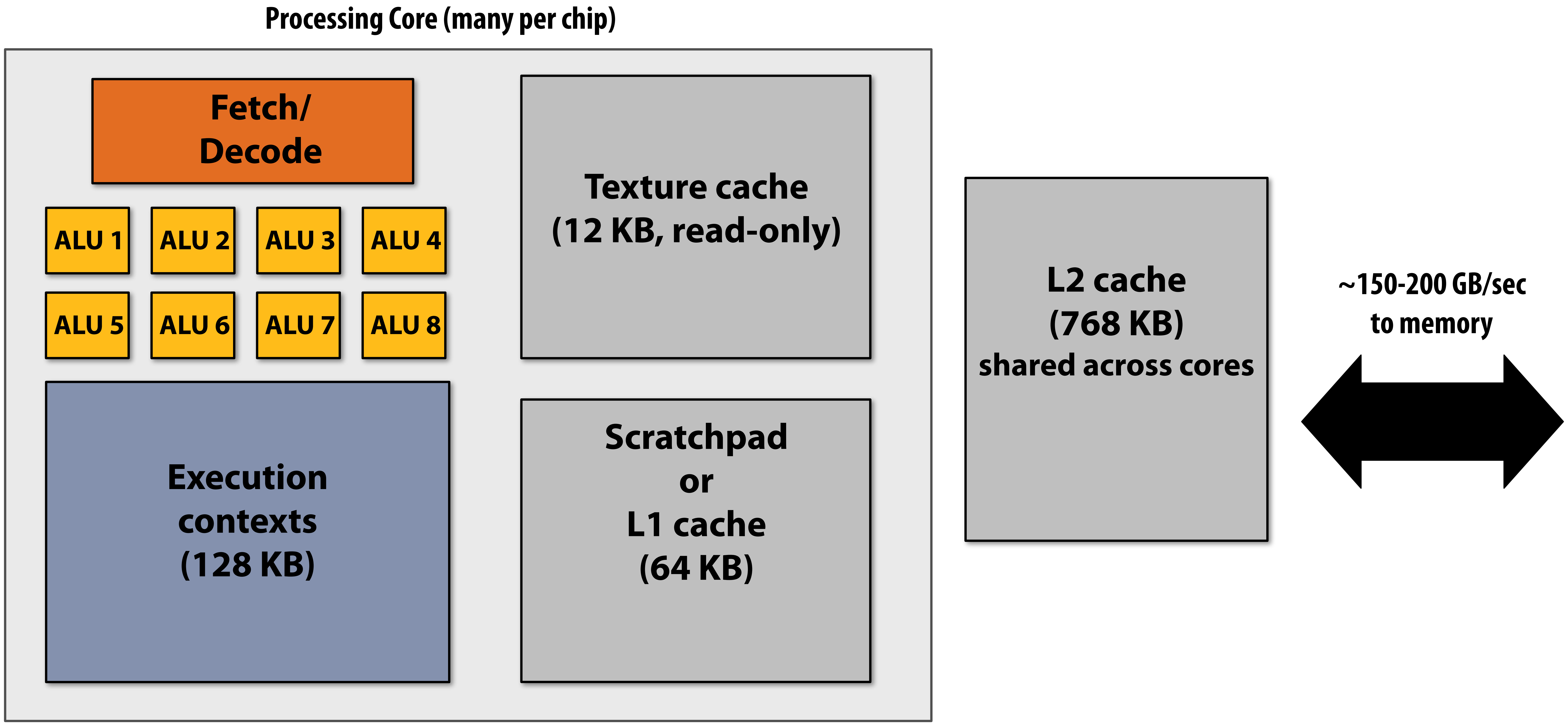


"CPU-style" memory hierarchy



CPU cores run efficiently when data is resident in cache (caches reduce latency, provide high bandwidth)

"GPU-style" memory hierarchy (data from NVIDIA GF100: "Fermi")


Processing Core (many per chip)
Fetch/ Decode
ALU 1
ALU 2
ALU 3
ALU 4
ALU 5
ALU 6
ALU 7
ALU 8
Execution contexts (128 KB)
Texture cache (12 KB, read-only)
Scratchpad or L1 cache (64 KB)
L2 cache (768 KB) shared across cores
~150-200 GB/sec to memory

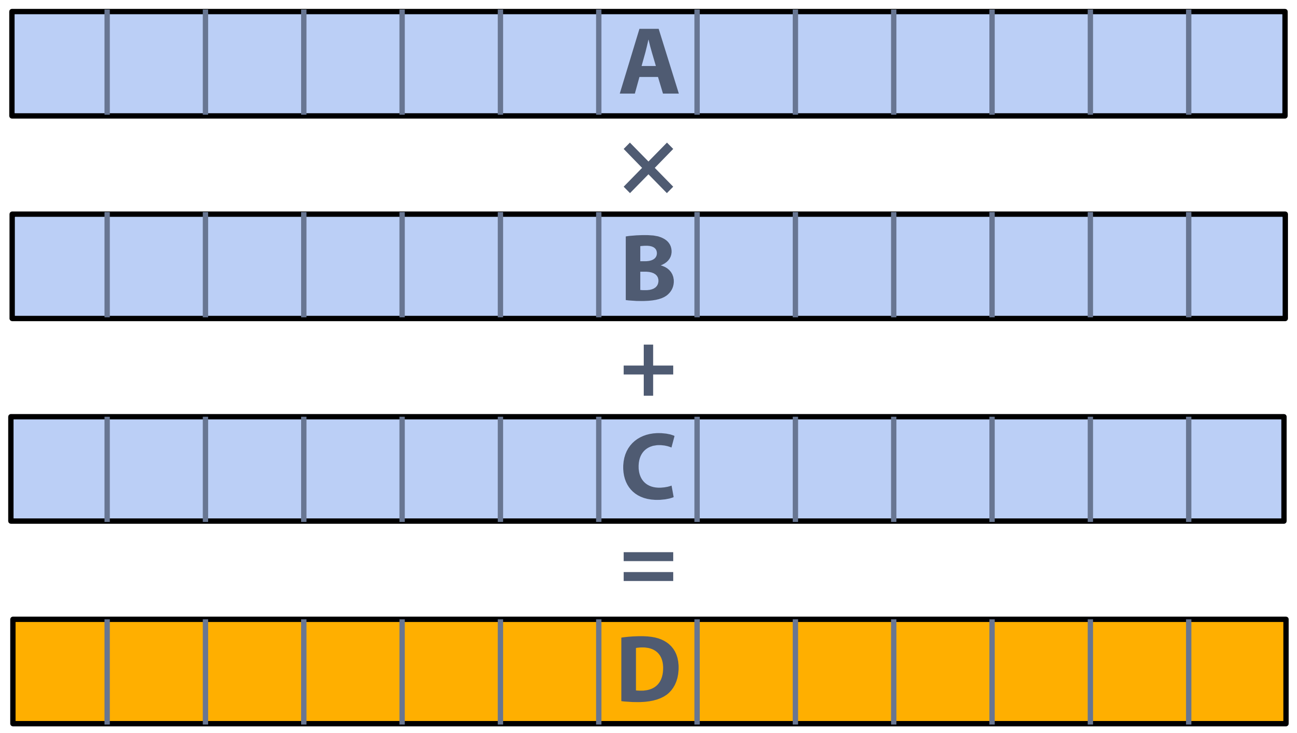
More cores, more ALUs, no large traditional cache hierarchy (use threads to tolerate latency)

Require high-bandwidth connection to memory

Thought experiment

Task: element-wise multiplication of two vectors A and B

1. Load input A[i]
2. Load input B[i]
3. Load input C[i]
4. Compute A[i] × B[i] + C[i]
5. Store result into D[i]



Four memory operations (16 bytes) for every MUL-ADD

Radeon HD 5870 can do 1600 MUL-ADDs per clock

Need ~20 TB/sec of bandwidth to keep functional units busy

Less than 1% efficiency… but 6x faster than CPU!

Bandwidth limited!

If processors request data at too high a rate, the memory system cannot keep up.

No amount of latency hiding helps this.

Overcoming bandwidth limits are a common challenge for application developers on throughput-optimized systems.

Bandwidth is a critical resource

- A high-end GPU (e.g., Radeon HD 5870) has...
  - Over **twenty times** (2.7 TFLOPS) the compute performance of quad-core CPU
  - No large cache hierarchy to absorb memory requests

- GPU memory systems are designed for throughput
  - Wide memory bus (150-200 GB/sec)
  - Still, this is only **six-to-eight times** the bandwidth available to CPU

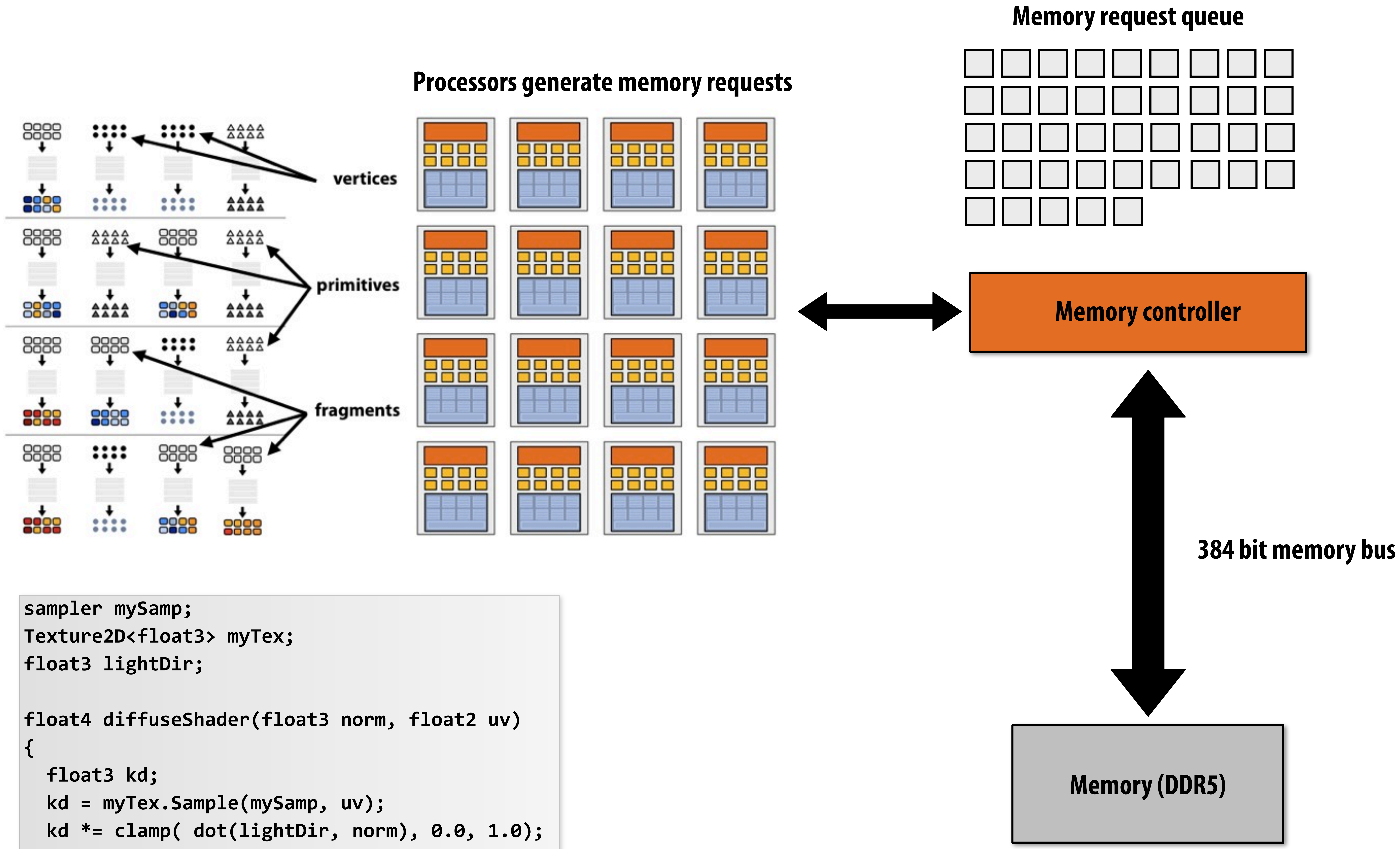
Bandwidth is a critical resource

- Use available bandwidth well

- Fetch data from memory less often (share/reuse data)

- Request data less often (instead, do more math: it's "free")
  - "arithmetic intensity" : ratio of math to data access

Using available bandwidth well



```
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);
}
```

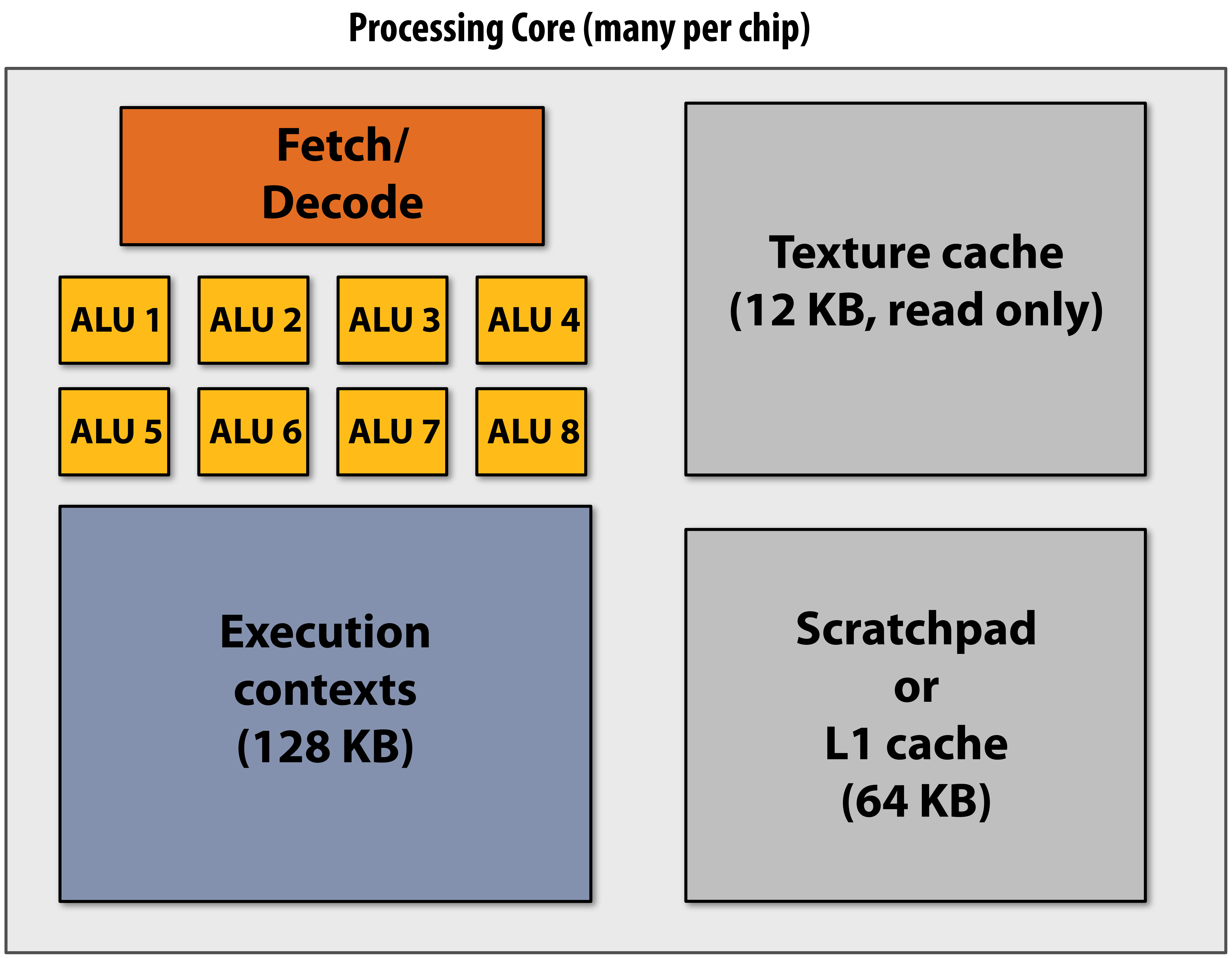
Bandwidth is a critical resource

- Use available bandwidth well
  - GPUs feature sophisticated memory request reordering logic
  - Repack/reorder/interleave many buffered memory requests to maximize memory utilization

- Fetch data from <u>memory</u> less often (share/reuse data)
  - Intra-fragment reuse
  - Cross-fragment reuse
  - Compression

- Request data less often (instead, do more math: it's "free")
  - "arithmetic intensity" : ratio of math to data access

Scratchpad for reuse known at compile-time


Processing Core (many per chip)
Fetch/ Decode
ALU 1
ALU 2
ALU 3
ALU 4
ALU 5
ALU 6
ALU 7
ALU 8
Execution contexts (128 KB)
Texture cache (12 KB, read only)
Scratchpad or L1 cache (64 KB)

Load-data into scratchpad (LD addr -> scratchpad addr)

Many fragments reuse data loaded into scratchpad once ***

*** Not in OpenGL/Direct3D shader programming model (under the hood optimization)

Bandwidth is a critical resource

- **Use available bandwidth well**
  - GPUs feature sophisticated memory request reordering logic
  - Repack/reorder/interleave many buffered memory requests to maximize memory utilization

- **Fetch data from <u>memory</u> less often (share/reuse data)**
  - Intra-fragment reuse
  - Cross-fragment reuse
  - Compression

- **Request data less often (instead, do more math: it's "free")**
  - "arithmetic intensity" : ratio of math to data access

Shading often has high arithmetic intensity

```
sampler mySamp;
Texture2D<float3> myTex;
float3 ks;
float  shinyExp;
float3 lightDir;
float3 viewDir;

float4 phongShader(float3 norm, float2 uv)
{
  float result;
  float3 kd;
  kd = myTex.Sample(mySamp, uv);
  float spec = dot(viewDir, 2 * dot(-lightDir, norm) * norm + lightDir);
  result = kd * clamp(dot(lightDir, norm), 0.0, 1.0);
  result += ks * exp(spec, shinyExp);
  return float4(result, 1.0);
}
```

Image credit: http://caig.cs.nctu.edu.tw/course/CG2007

3 scalar float operations + 1 exp()
8 float3 operations + 1 clamp()
1 texture access (highlighted in red)

Vertex processing often has higher arithmetic intensity than fragment processing (less use of texturing)

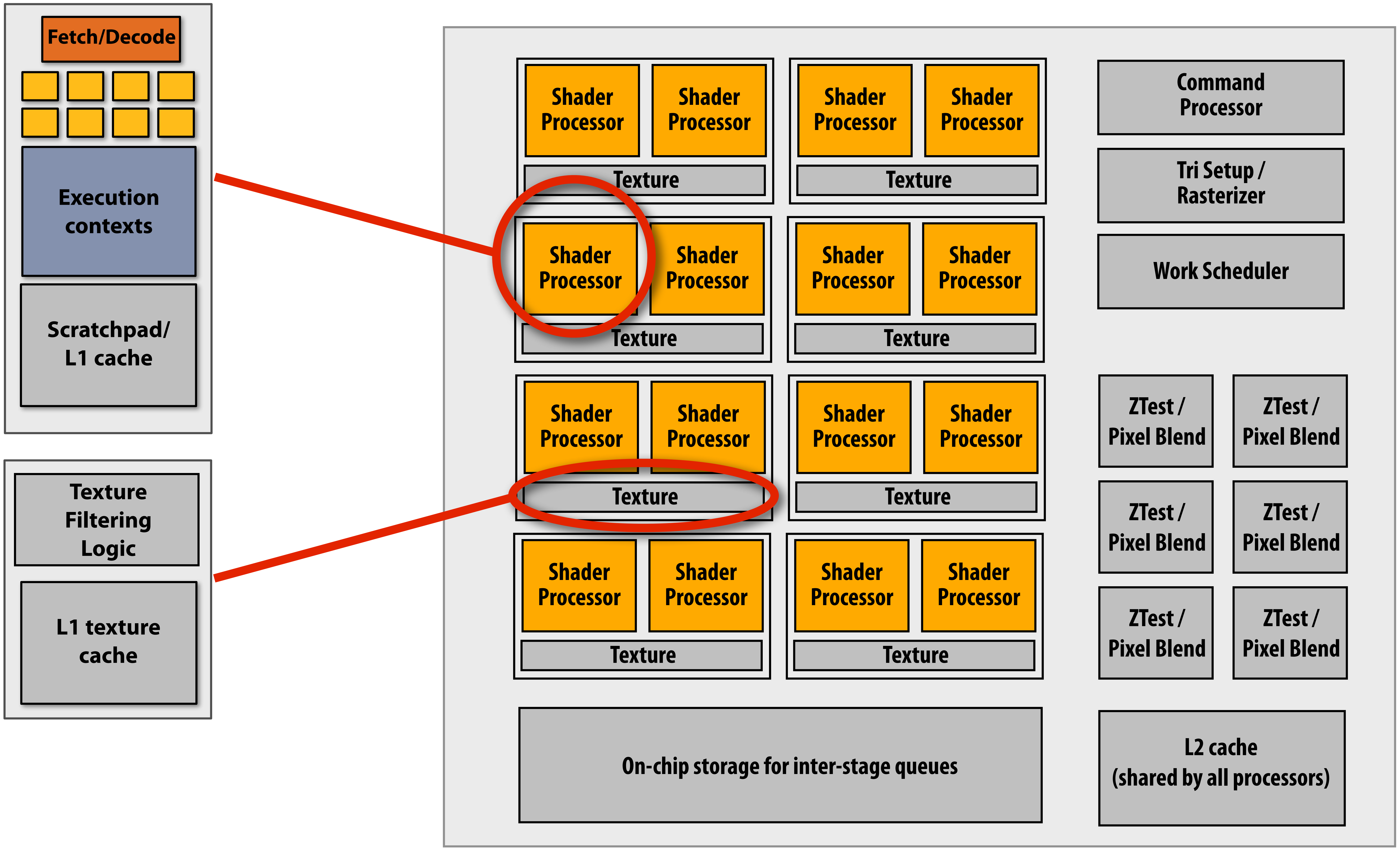
Summary: workloads that run efficiently on a GPU's programmable cores ...

- Have thousands of independent pieces of work
  - Utilizes many ALUs on many cores
  - Have <u>much more</u> parallel work than numbers of GPU ALUs, enabling large-scale interleaving as a mechanism to hide memory latency

- Are amenable to instruction stream sharing
  - Maps to SIMD execution well

- Are compute-heavy: the ratio of math operations to memory access is high
  - Not limited by memory bandwidth

Modern GPU: heterogeneous many-core



Homogeneous collection of throughput-optimized programmable processing cores

Augmented by fixed-function logic

Readings

- E. Lindholm et al., *NVIDIA Tesla: A Unified Graphics and Computing Architecture*. IEEE Micro, March 2008

  (note: parts about non-graphics computing beginning on p49 not required)

- Not required, but recommended background on the origin of the modern programmable processor:
  - E. Lindholm et al., *A User Programmable Vertex Engine*. SIGGRAPH 2001

Projects

- Project proposals due Friday (11:59pm)
  - Email documents to Kayvon
  - I will review through them over the weekend

- Relevant literature surveys are due following Friday