What we learned: Idea 1

CPU

GPU

What we learned: Idea 2

Each GPU core has many ALUs

What we learned: Idea 3

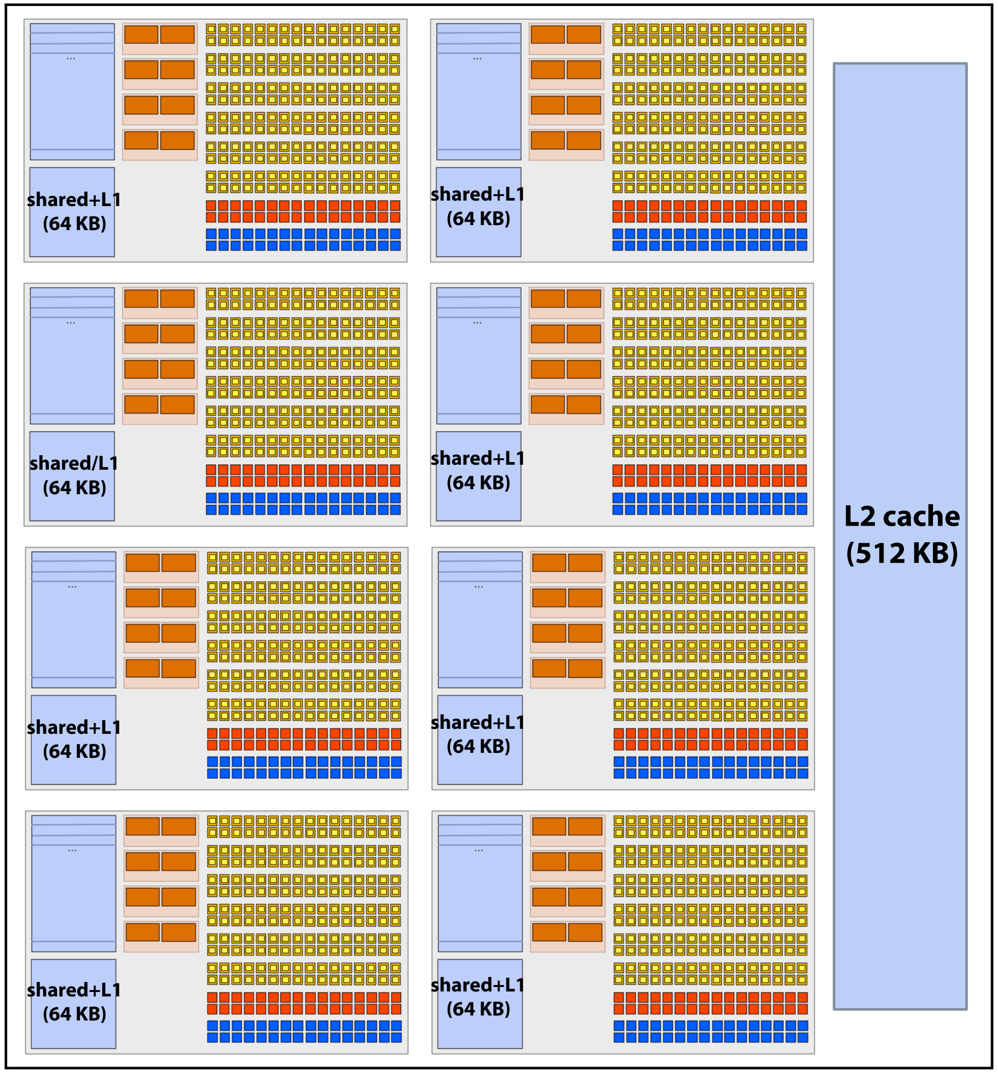
Interleave groups of threads

What we learned: processing data

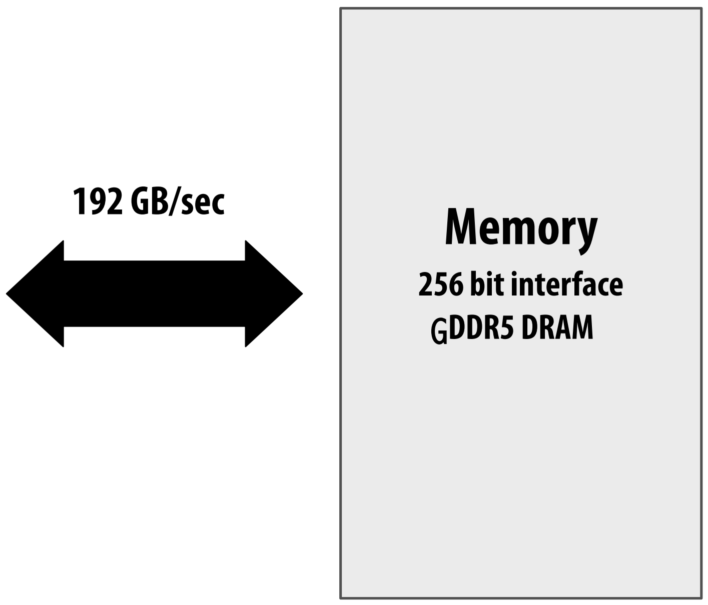
What about moving data to processors?

NVIDIA GTX 680 (2012)

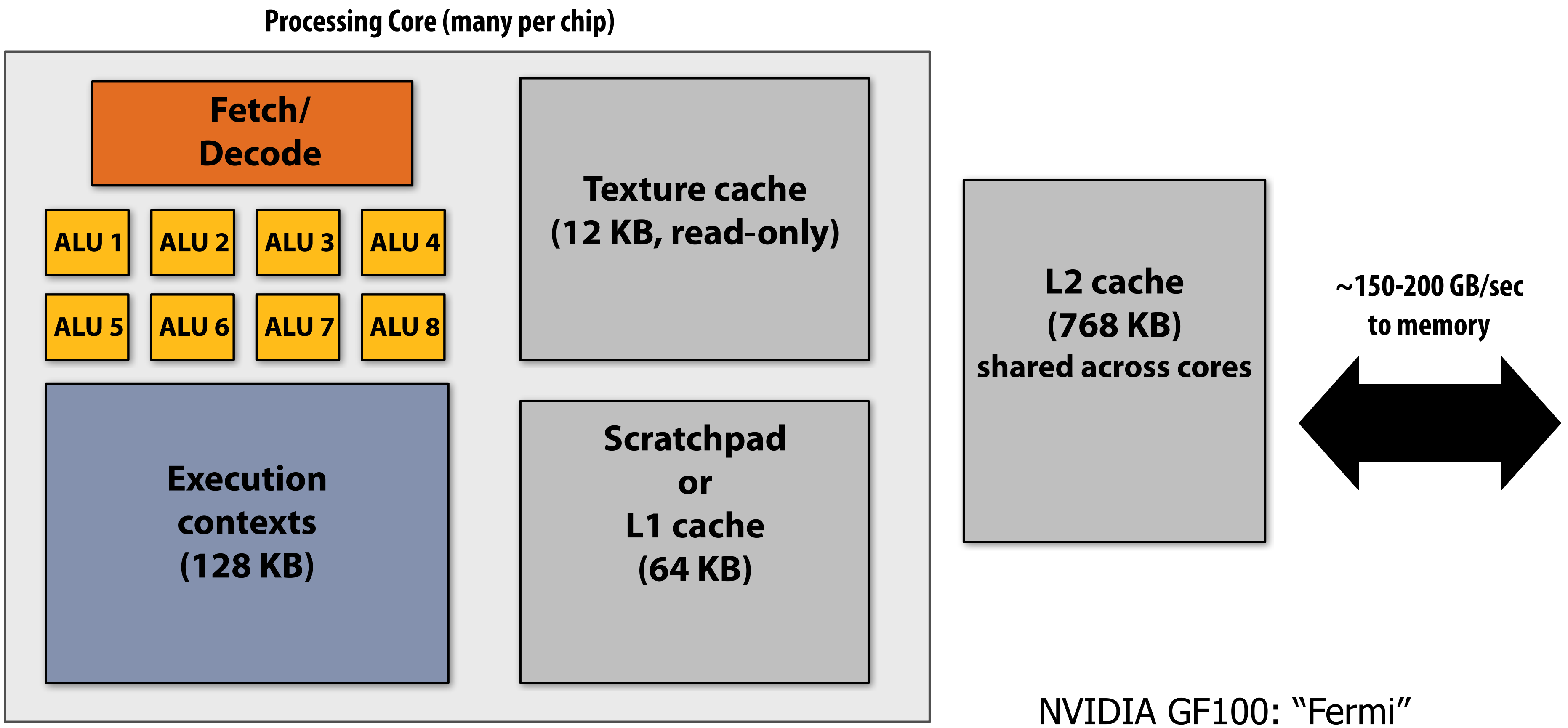
NVIDIA Kepler GK104 architecture



- 1 GHz clock
- Eight SMX cores per chip
- 8 x 192 = 1,536 SIMD mul-add ALUs = 3 TFLOPs
- Up to 512 interleaved warps per chip (16,384 CUDA threads/chip)
- TDP: 195 watts



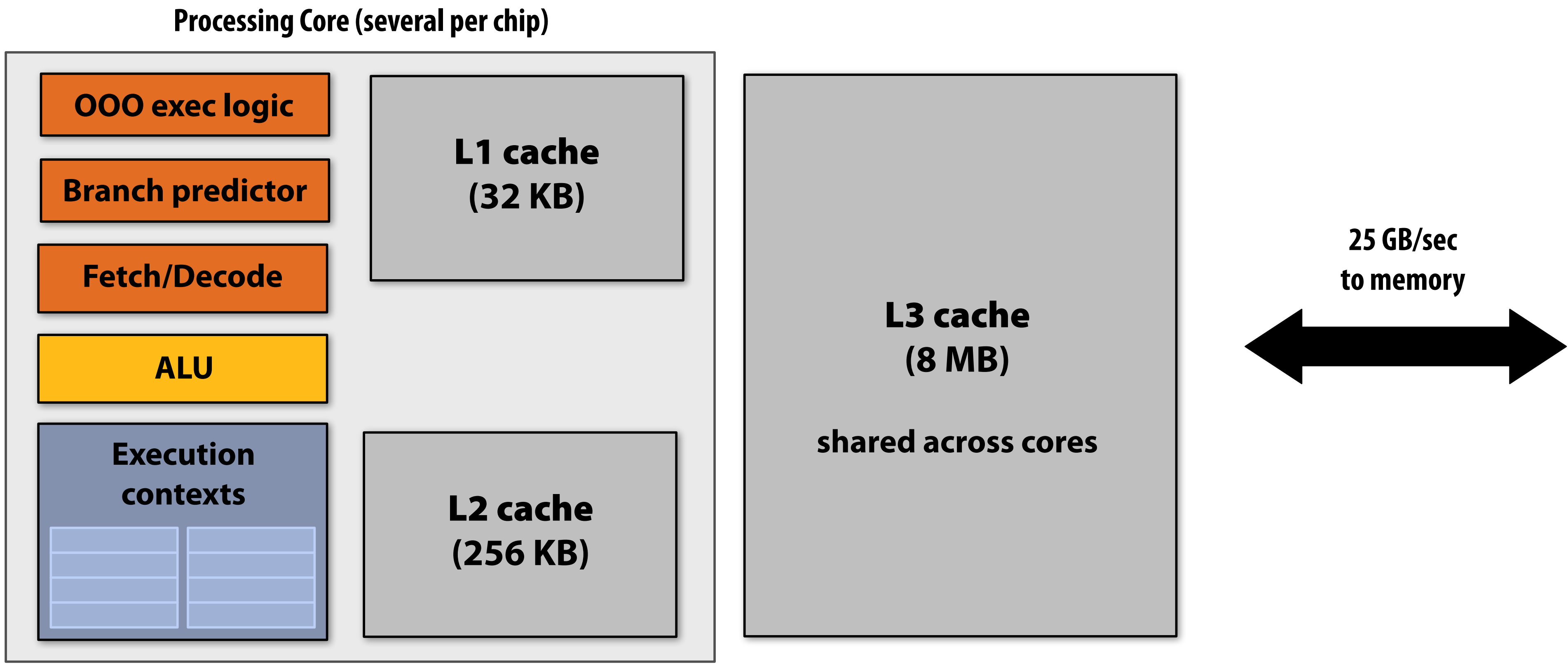
"GPU-style" memory hierarchy


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

NVIDIA GF100: "Fermi"

More cores, more ALUs, no large traditional cache hierarchy (use threads to tolerate latency) Require high-bandwidth connection to memory

Recall: "CPU-style" memory hierarchy

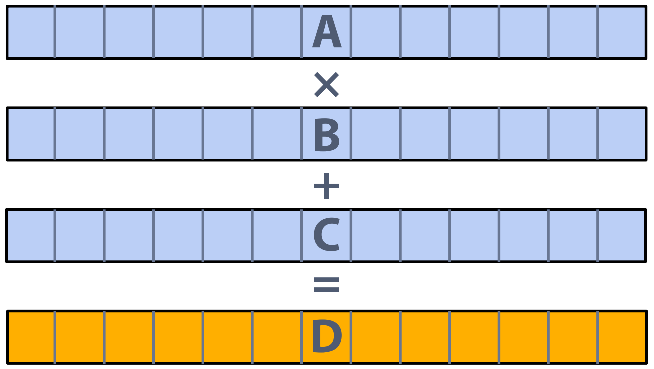


CPU cores run efficiently when data is resident in cache (caches reduce latency, provide high bandwidth)

GPUs are memory bandwidth hungry

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 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
  - GDDR: today’s version is GDDR5
  - Wide memory bus & high memory frequency
    - E.g., 384-bit memory bus
    - Bandwidth can achieve 150-200 GB/sec
  - Still, this is only six-to-eight times the bandwidth available to CPU

Bandwidth limited!

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

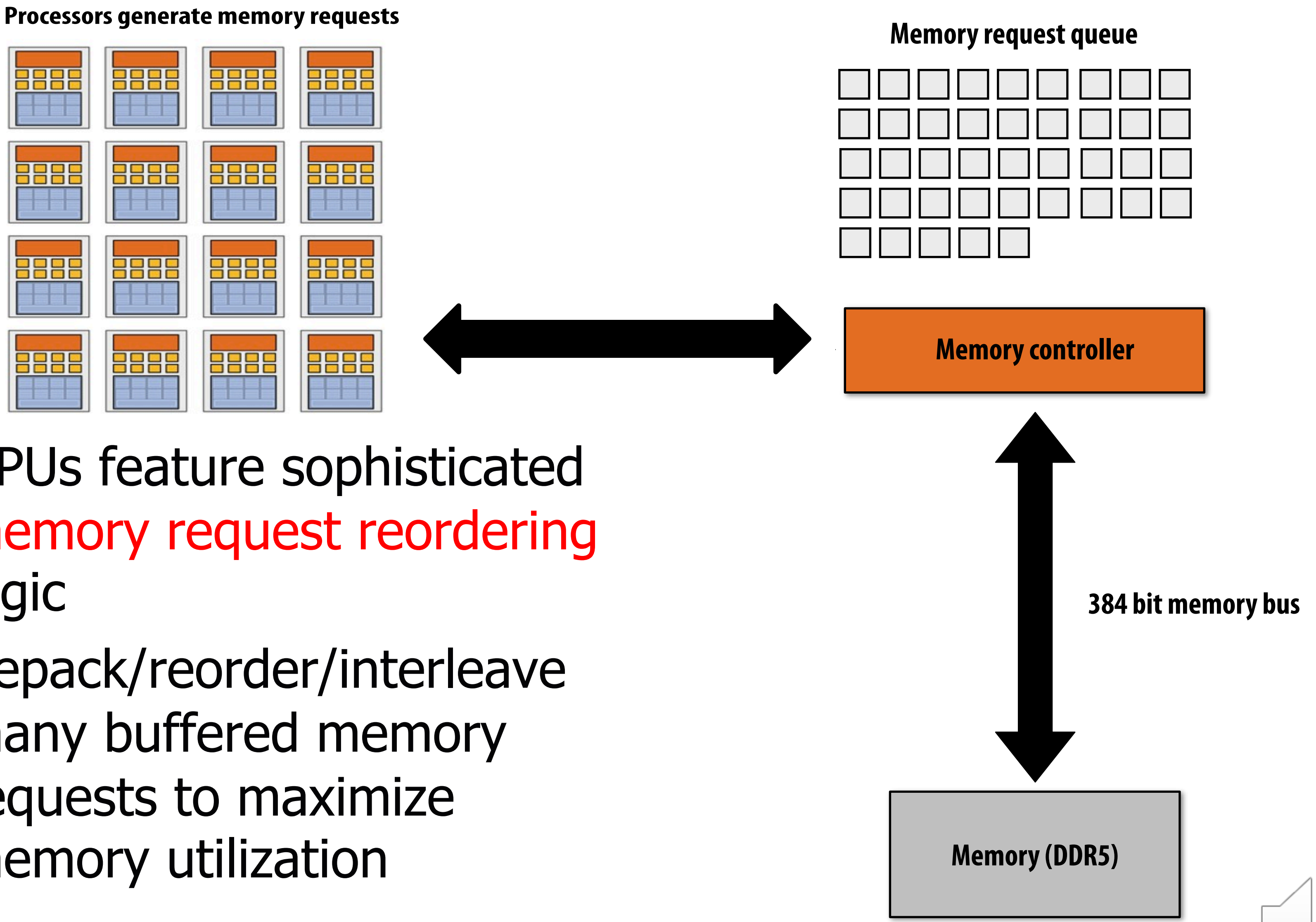
No amount of latency hiding helps this.

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

Deal with the bandwidth challenge

- 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
- New memory technologies beyond GDDRx
  - HBM (3D/2.5D integrated memory), e.g., AMD Fury X, NVIDIA Pascal and Volta

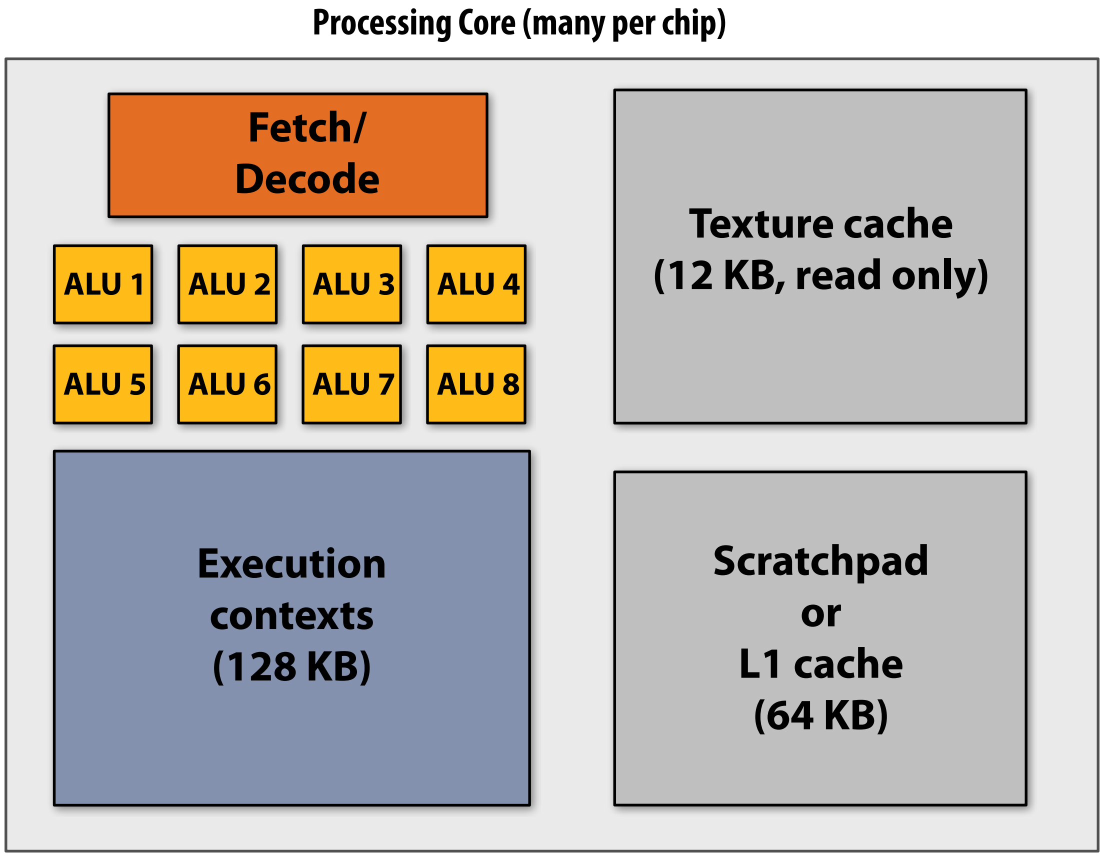
Idea 1: Using available bandwidth well



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

Idea 2: Fetch data from memory less often

- Scratchpad for reuse known at compile-time
  - Intra-fragment reuse
  - Cross-fragment reuse
- Compression


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

Load-data into scratchpad (LD addr -> scratchpad addr)
Many fragments reuse data loaded into scratchpad once

Deal with the bandwidth challenge

- 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
- New memory technologies

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

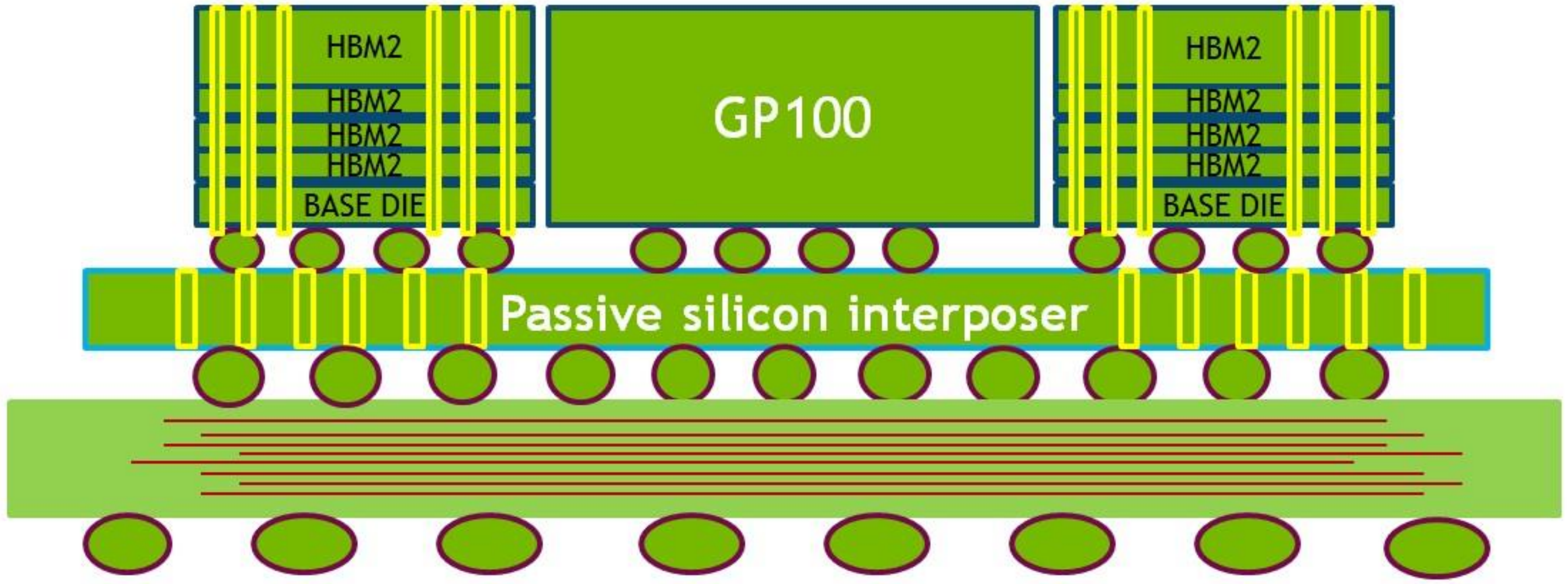
3 scalar float operations + 1 exp()
8 float3 operations + 1 clamp()
1 texture access (highlighted in red)

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

Deal with the bandwidth challenge

- 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
- New memory technologies
  - E.g., 3D integration – main memory can be "on-chip"

Idea 4: NVIDIA Tesla P100 (Pascal 2016)



Some cards feature 16 GB HBM2 in four stacks with a total of 4096bit bus with a memory bandwidth of 720 GB/s

Summary: GPU Memory Design Ideas

- 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
- New memory technologies
  - E.g., 3D integration – main memory can be "on-chip"