A look at the GPU architecture

Pre-G80: Separate vertex and fragment processors.

Hard-wired for graphics. Load balance problems.

G80: Unified architecture. More suited for GPGPU. Higher performance due to better load balancing.

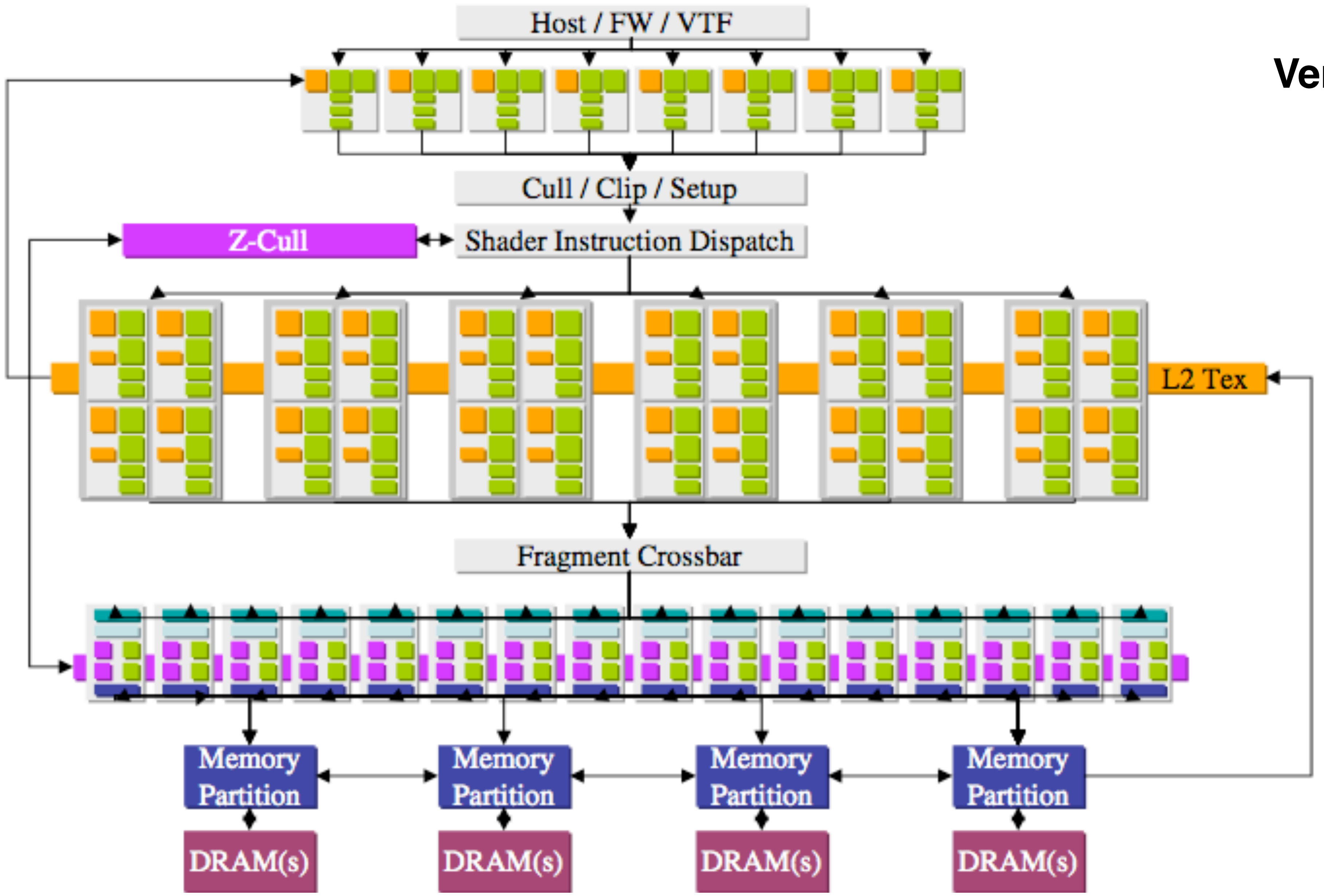
G92: Similar to G80, more cores, more cores per group.

GT100: More cores, much more double precision

GK104: More cores, more power efficient

(Similar track for AMD)

7800: High-end GPU before G80

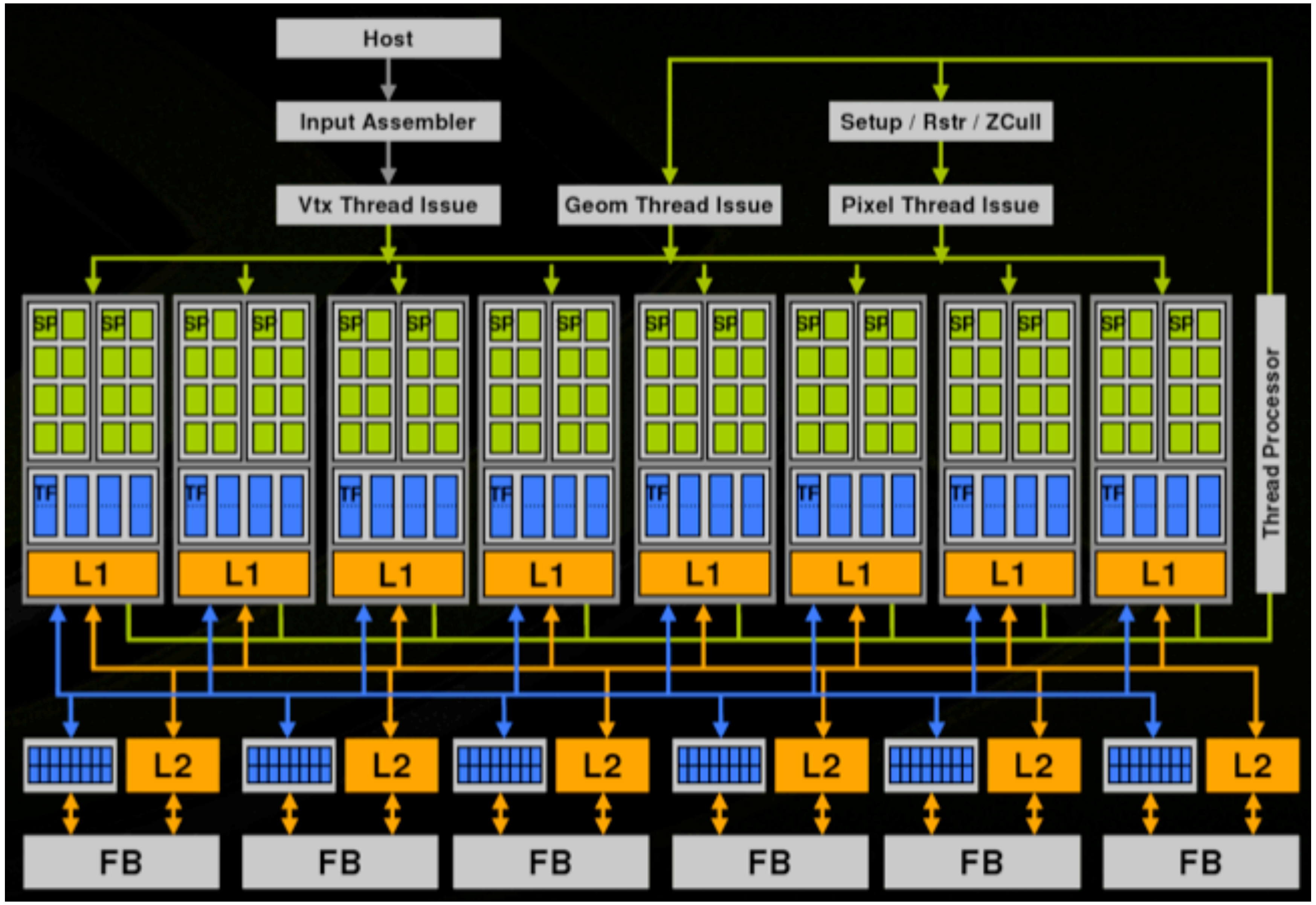


Vertex processors

Fragment processors

Framebuffer operations

G80



Hardware formerly between vertex and fragment processors

Unified processors

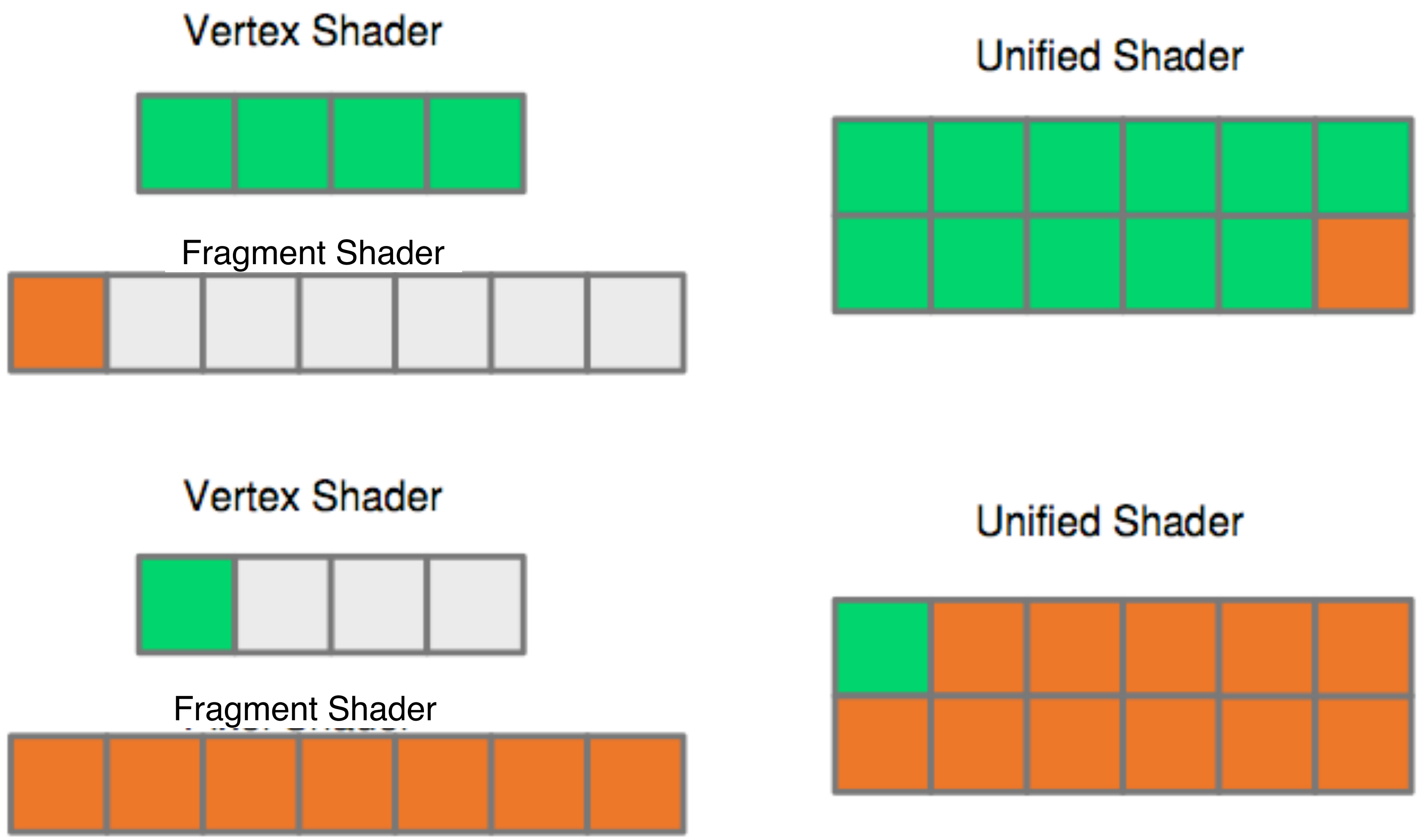
Framebuffer operations

G80: A question of *load balance*!

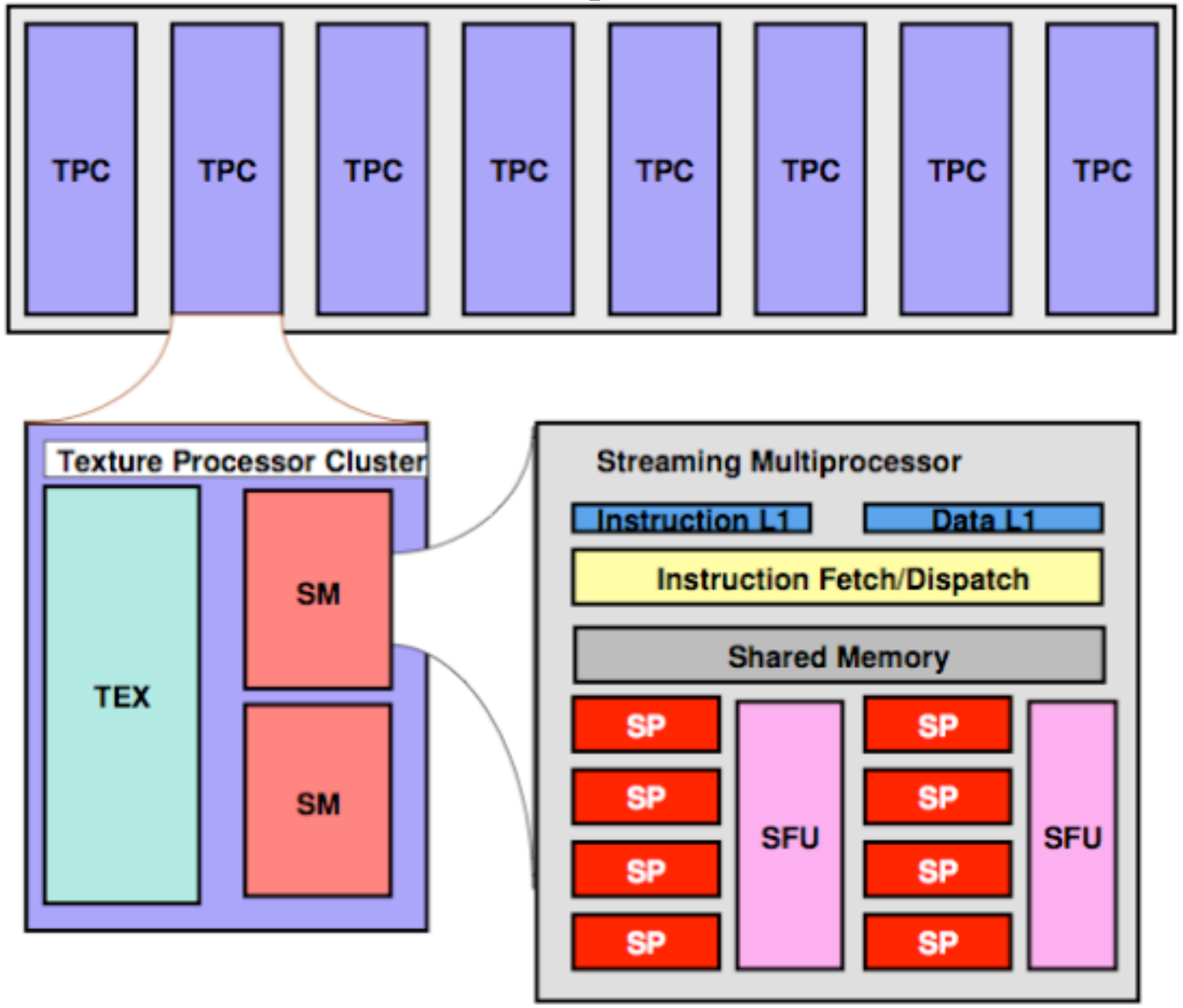
Separate vertex and fragment processors | **Unified processors**

Vertex problem (e.g. complex geometry)

Fragment problem (e.g. advanced rendering)



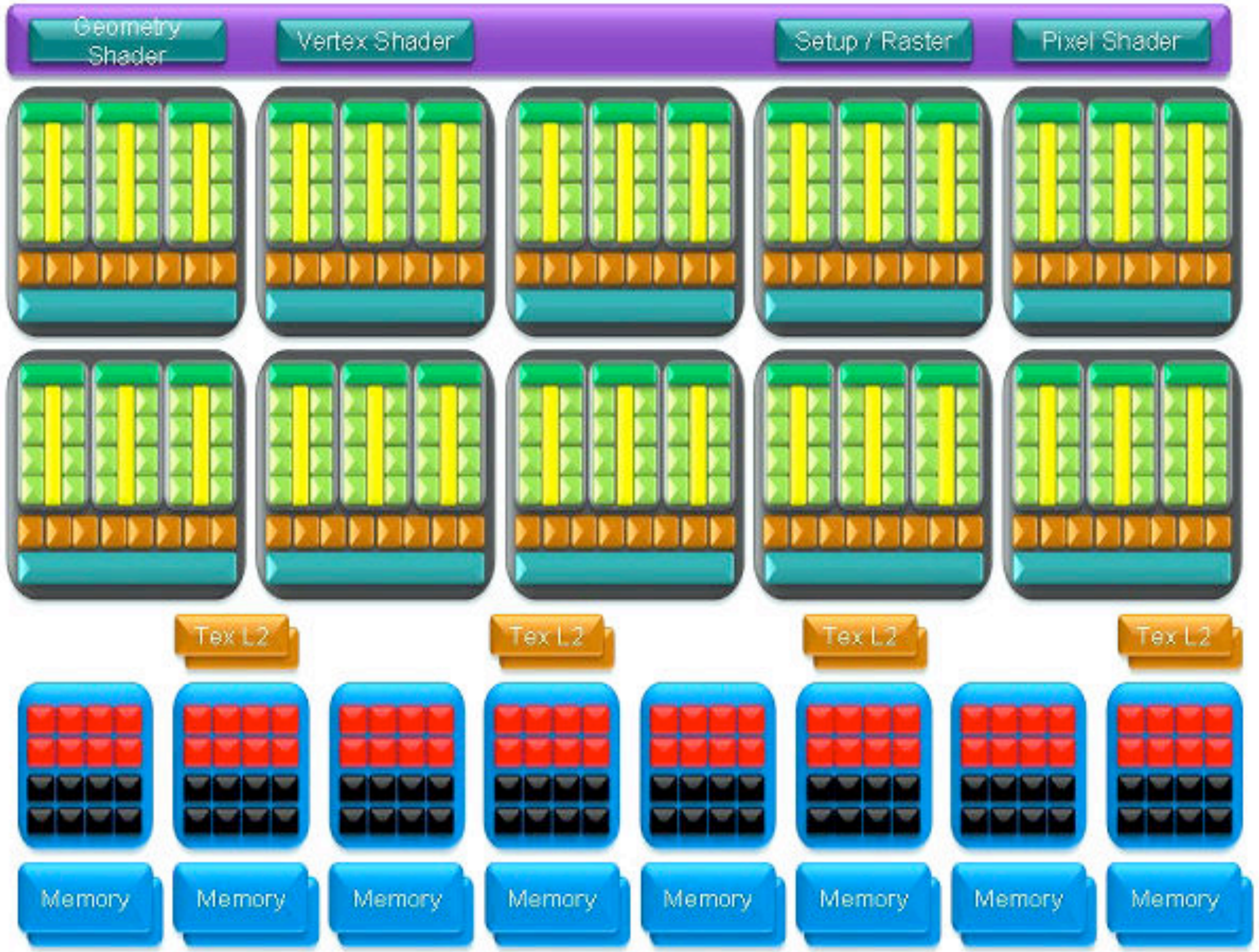
G80 processor hierarchy



8 top-level groups of TPCs

SM is a group of 8 SIMD cores

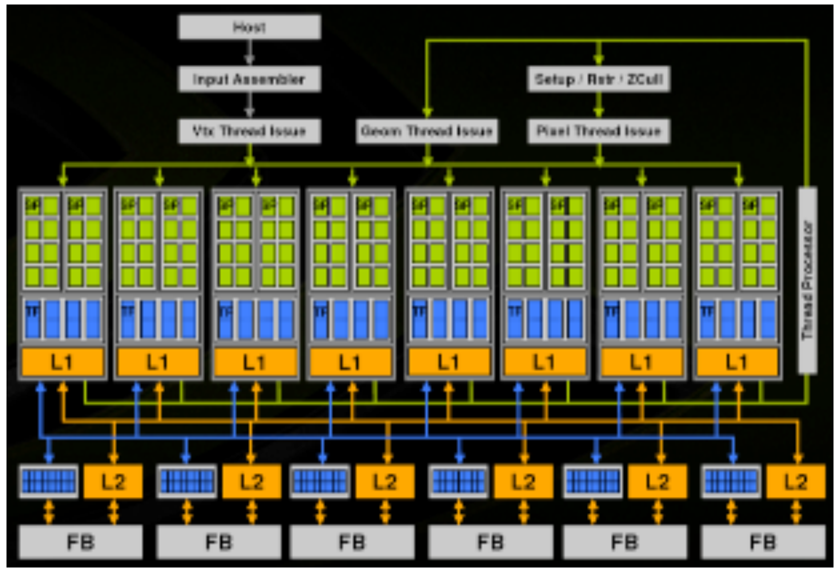
GT200

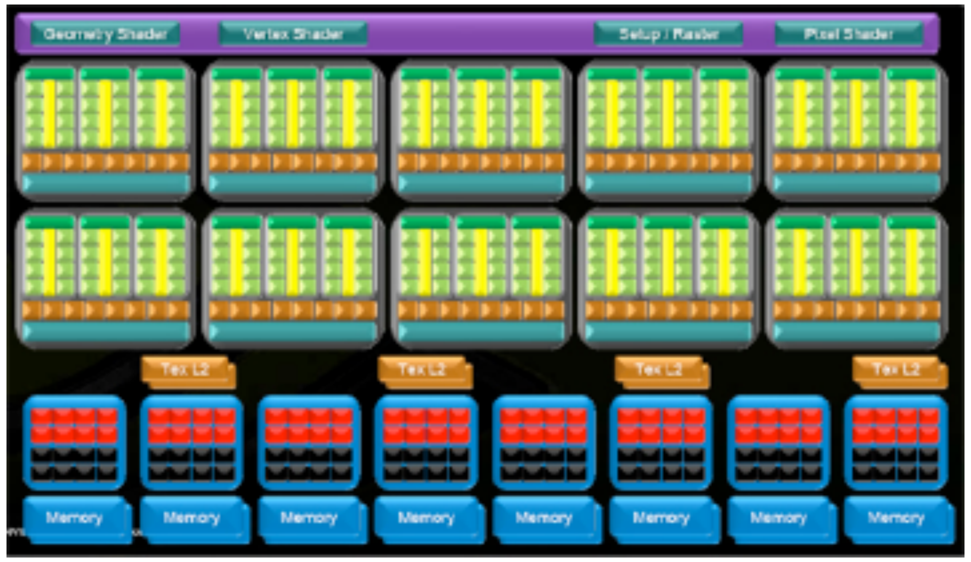


Similar but with a bit more of everything

G80 vs GT200 in numbers:

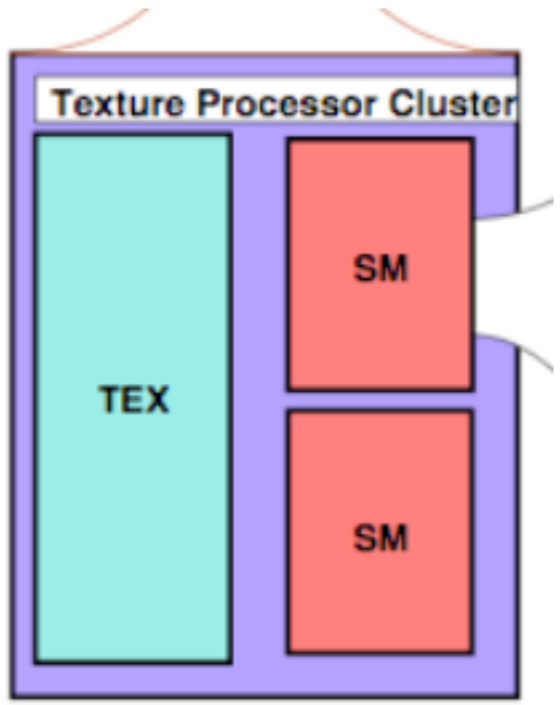
8 cores per SM 10 cores per SM
2 SMs per cluster 3 SMs per cluster
8 clusters 10 clusters





8 was *not* a magic number - more cores per SM

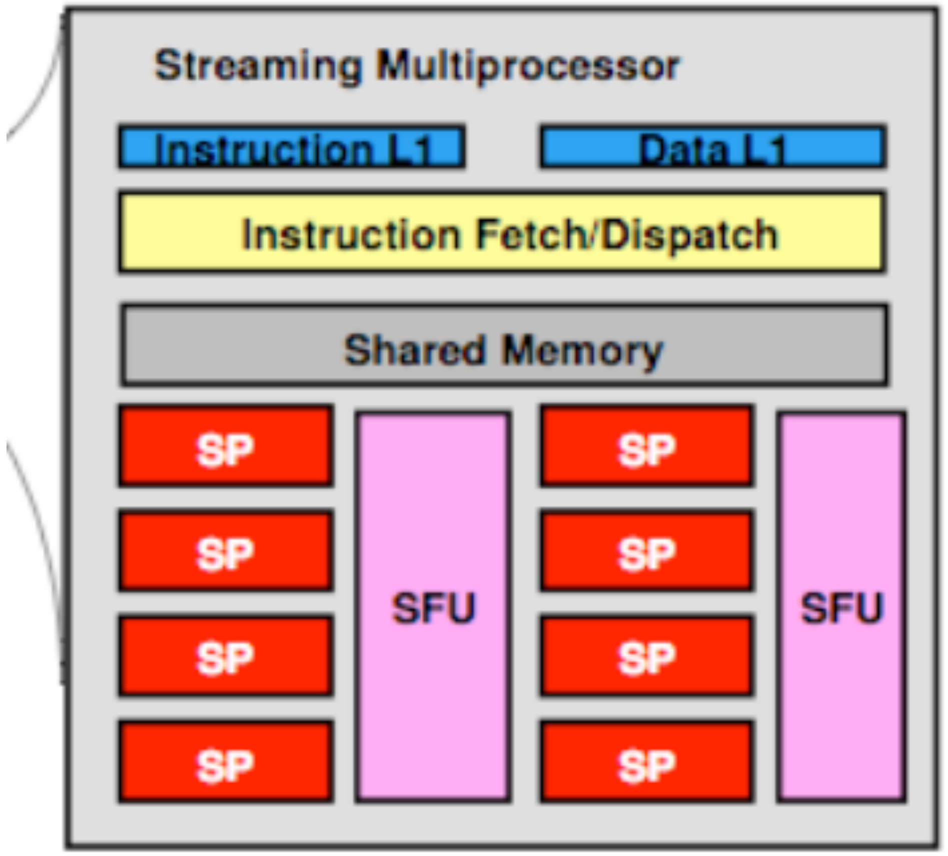
Vital components



Texture processor cluster: 2 or 3 SMs and a *texturing unit*

A texturing unit will provide texturing access with automatic interpolation - vital component for graphics

Vital components



SM: 8 cores

but also

SFU: Special functions unit

Shared memory

Register memory in each core

Instruction handling/thread management

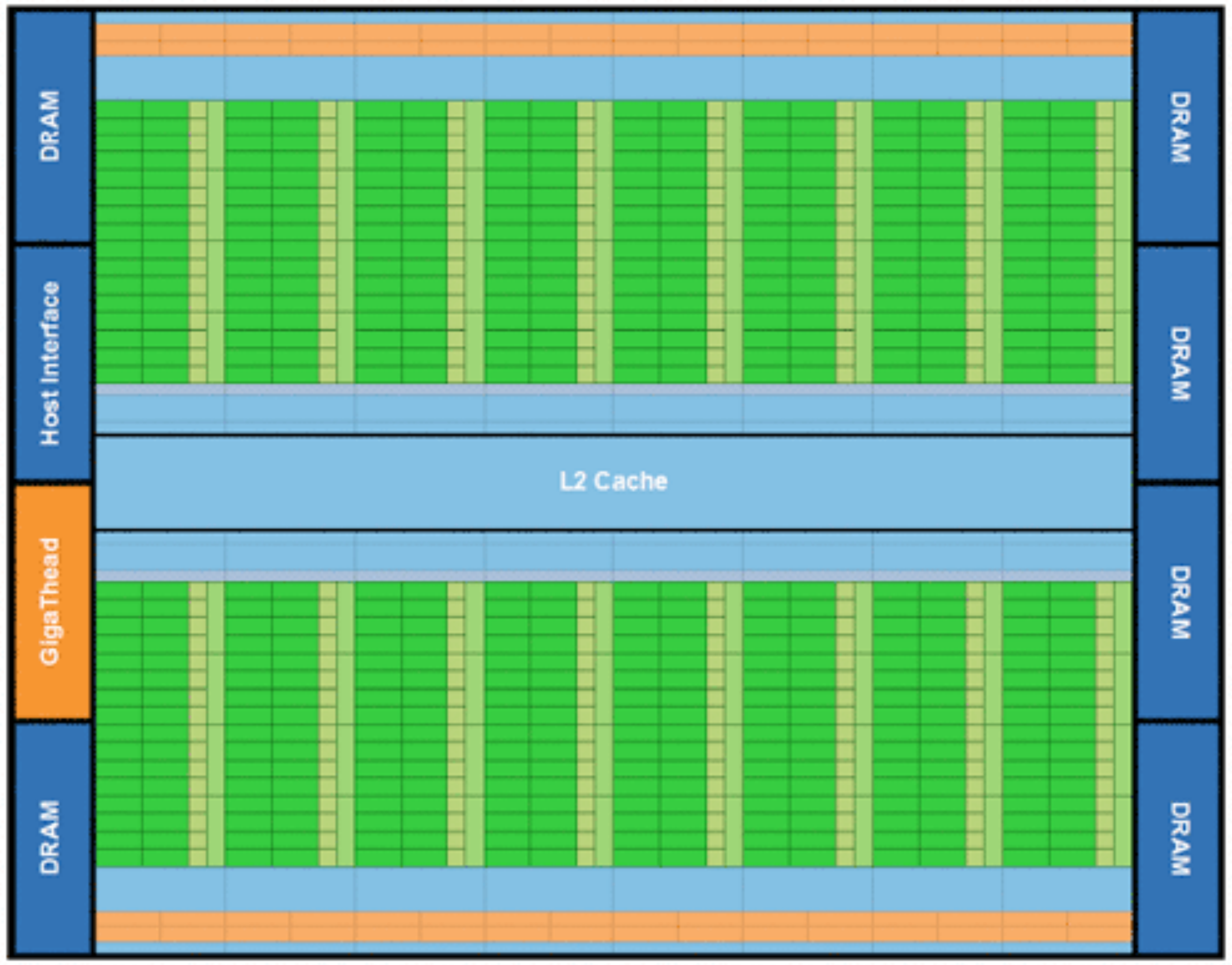
How much architecture details do we need to know?

Shaders: The architecture is mostly invisible

Cuda/OpenCL: Less so, but number of cores more or less ignored - as long as we provide more parallelism in our algorithm than the architecture has!

Memory usage is specified by the programming languages. More about that later.

2010: Fermi (GT100)



Looks like:

16 SMs

32 cores per SM

Support for 24576 threads!

Much area for L2 cache!

2010: Fermi (GT100)

Four clusters

Four SMs in each

32 cores per SM!

2010: Fermi (GT100)

Major changes in favor of general computing.

512 cores

Caching closer to the processors!

Concurrent kernels.

64-bit wide

ECC

More on Fermi

4x performance for double (64-bit FP)

More silicon space for cache! More like a CPU.

16 SMs, 512 cores (32 cores per SM)

CGPU = Computing Graphics Processing Unit

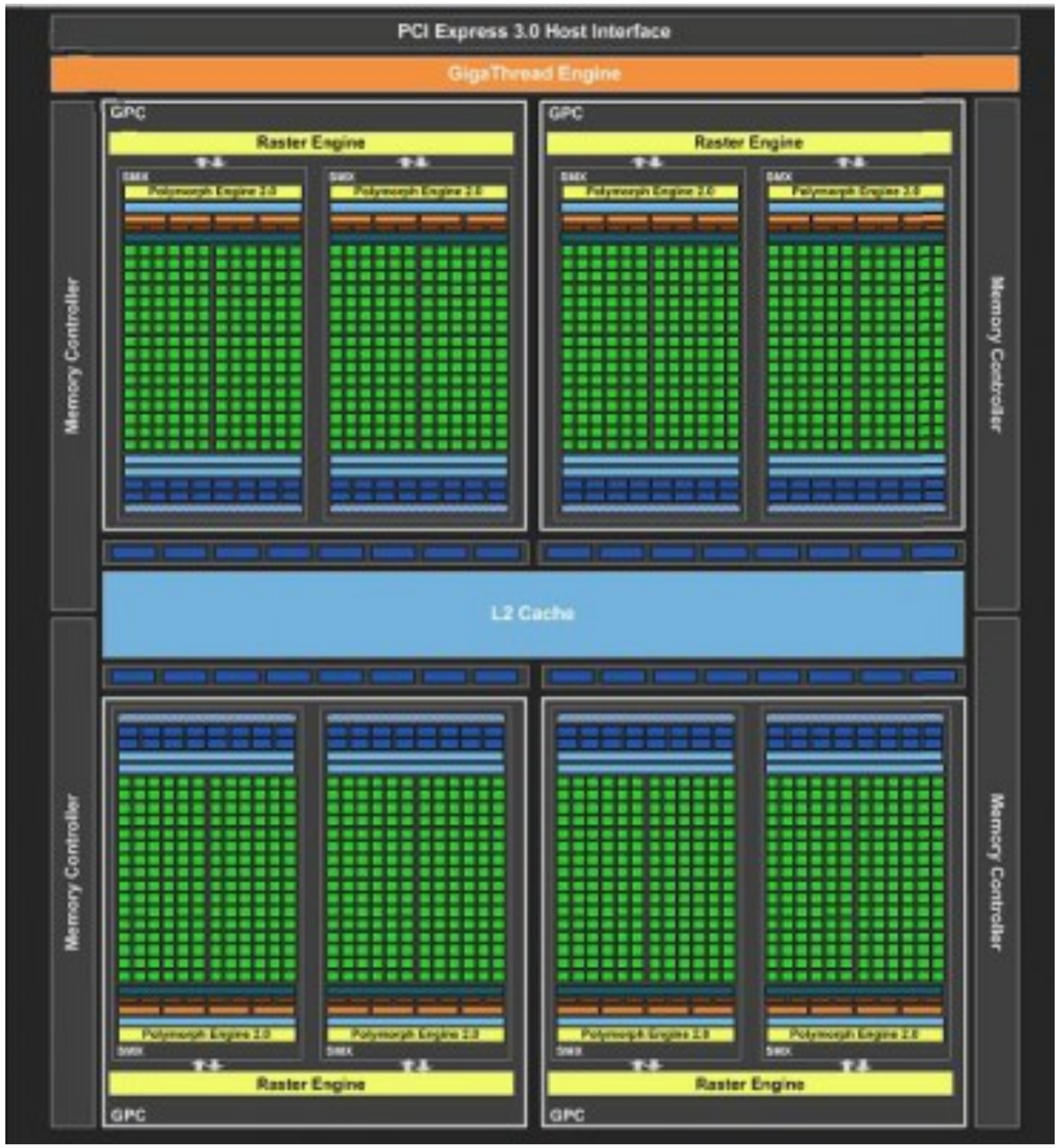
=> NVidia aims for GPGPU with Fermi!

2012: Kepler (GK104, GK110)
2014: Maxwell (GM107, GM204)

Back to graphics focus, strikes back against AMD.

Fewer SMs, double performancs lagging behind.

AMD taking the lead in GPU computing with the R9 series!



GK104 Kepler

1536 cores

8 SMs

Still a lot of cache

2016: Pascal (GP102-107)

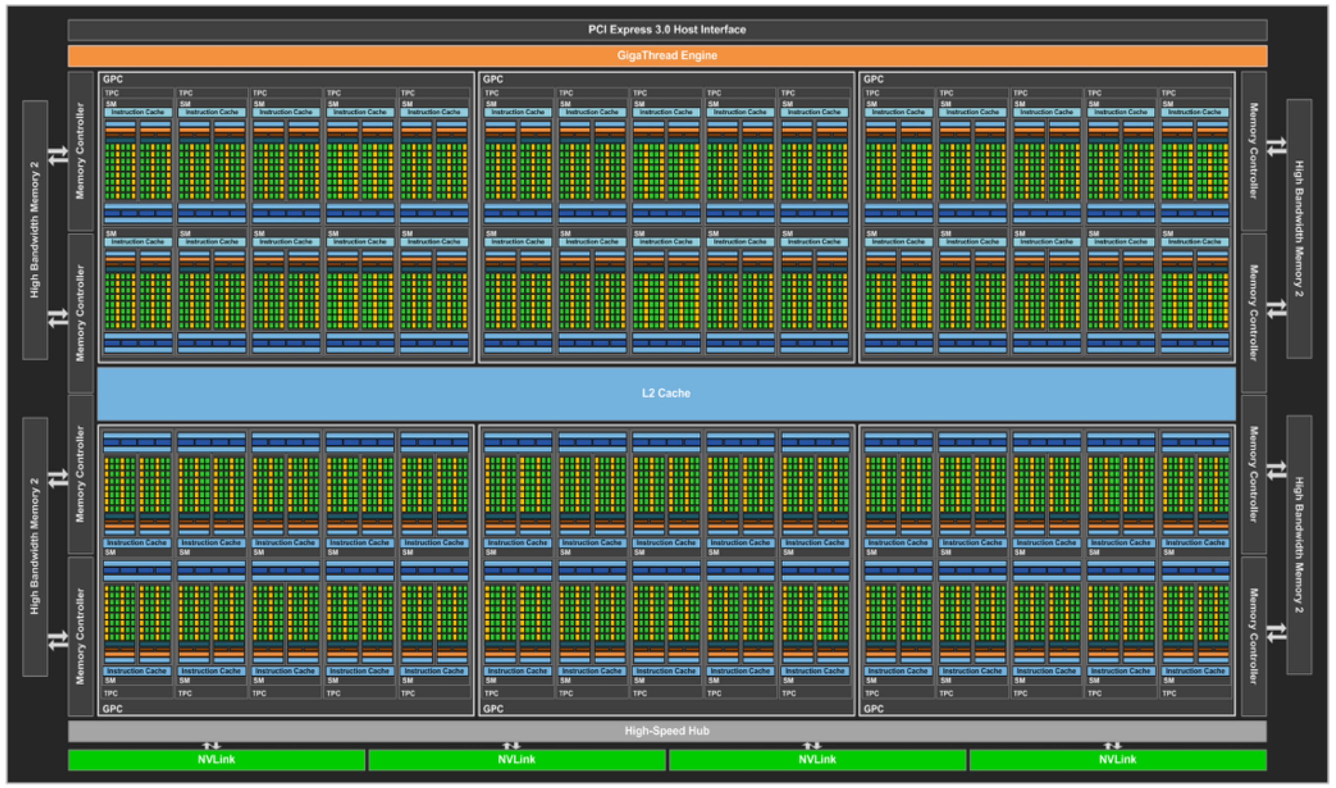
More SM's -> more independent computing!

56 SMs with 64 cores each!

Good double performance is back!

Pascal (GP102-107)

NVidia giving GPU computing attention again?



Related parallelization efforts

IBM Cell (next generation canceled!)

Intel Larabee ("put on ice" - dead)

GPUs are the clear winners so far!

But never count out Intel...

how about the more recent Xeon Phi?
(Follow-up on Larabee)

How does it compare?

	Xeon E5-2670	Xeon Phi 5110P	Tesla K20X
Cores	8	60	14 SMX
Logical Cores	16 (HT)	240 (HT)	2,688 CUDA cores
Frequency	2.60GHz	1.053GHz	735MHz
GFLOPs (double)	333	1,010	1,317
SIMD width	256 Bits	512 Bits	N/A
Memory	~16-128GB	8GB	6GB
Memory B/W	51.2GB/s	320GB/s	250GB/s
Threading	software	software	hardware

Test: Does it compete?

Paths	Sequential	Sandy-Bridge CPU[1,2]	Xeon Phi[1,2]	Tesla GPU[2]
128K	13,062ms	694ms	603ms	146ms
256K	26,106ms	1,399ms	795ms	280ms
512K	52,223ms	2,771ms	1,200ms	543ms

[1] The Sandy-Bridge and Phi implementations make use of SIMD vector intrinsics. ← Important!

[2] The MRG32K3a random generator from the cuRAND library (GPU) and MKL library (Sandy-Bridge/Phi) were used.

The GPU still wins! (Even over other SIMD!)

Conclusion comparison
SB - Xeon Phi - GPU

Even the CPU performed pretty well.

All use SIMD (at least partially) for best performance!

All require you to code in parallel!