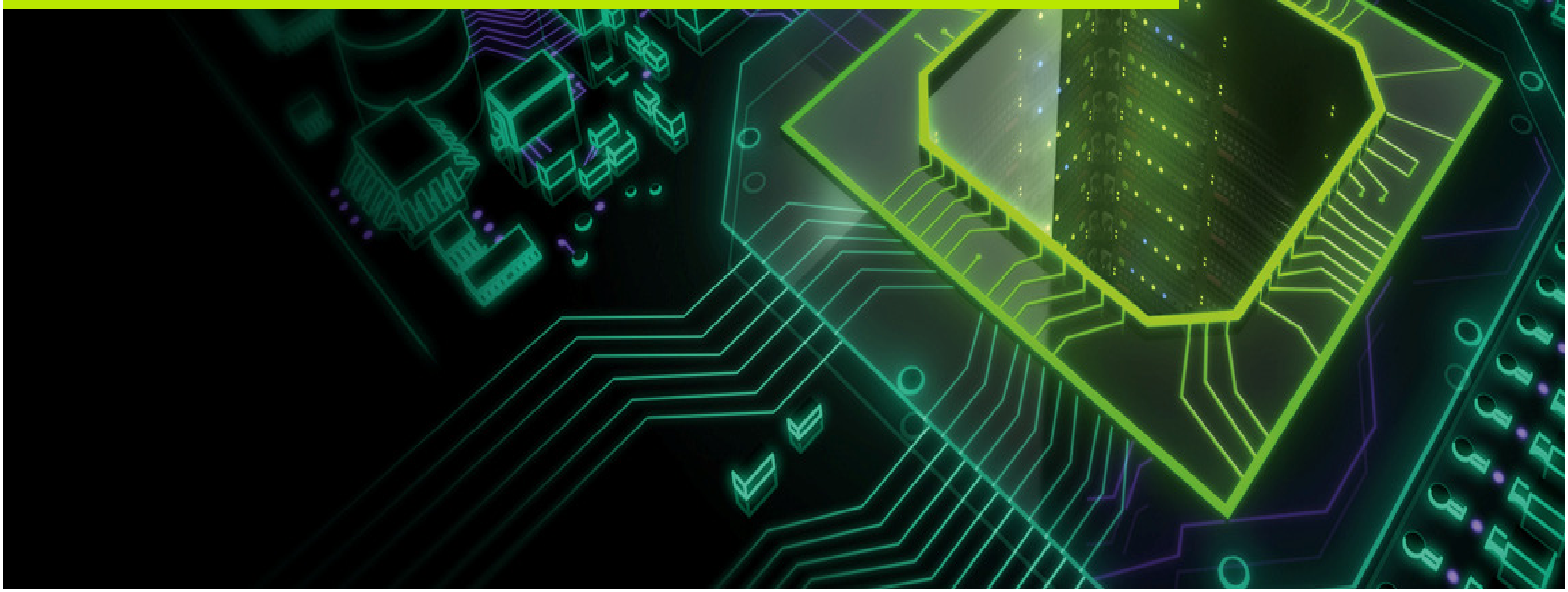
GPU Computing and the Road to Extreme-Scale Parallel Systems

STEVE KECKLER
DIRECTOR OF ARCHITECTURE RESEARCH



Agenda

- GPU Computing Overview
- Contemporary GPU Architectures
  - Fermi GPU Architecture
  - GPU systems
- Challenges for Extreme-Scale Parallel Systems
- Echelon – An NVIDIA HPC Research Project

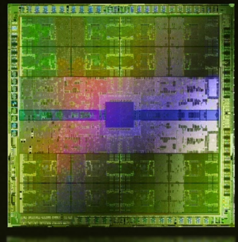
Evolution of GPUs

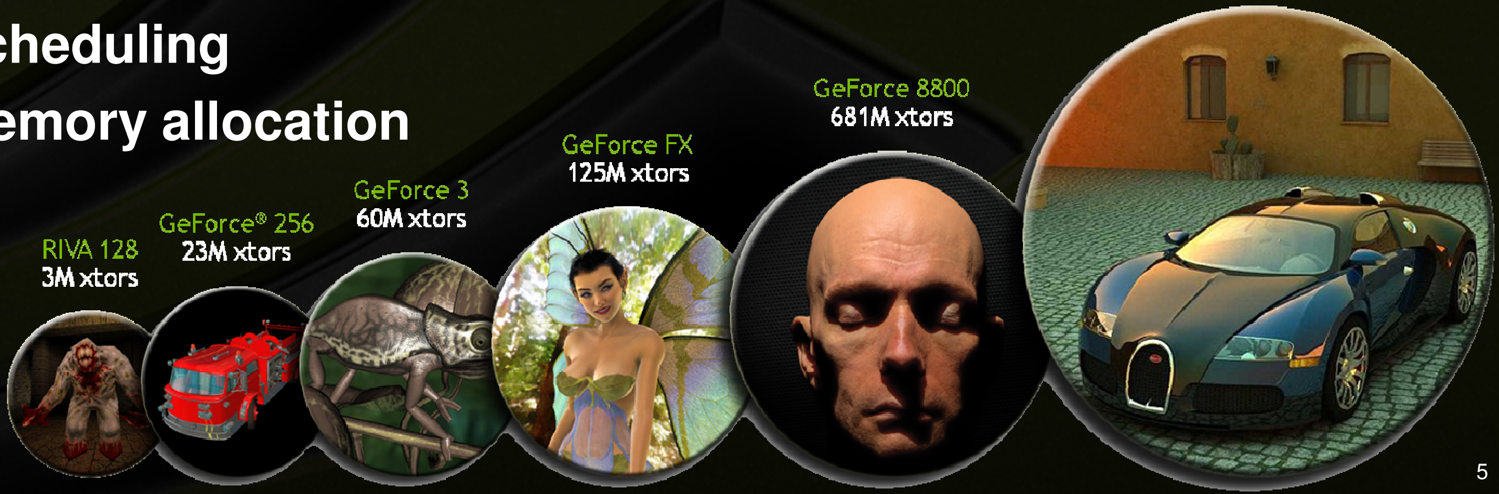
History of GPU Computing

- 1.0: Compute pretending to be graphics (early 2000s)
  - Disguise data as textures or geometry
  - Disguise algorithm as render passes
  - Trick graphics pipeline into doing your computation!
- 2.0: Program GPU directly – end of "GPGPU"
  - No graphics-based restrictions
  - 2006: Introduction of *CUDA* – general purpose compute language for hybrid GPU systems
- 3.0: GPU computing ecosystem (today)
  - 100,000+ active CUDA developers
  - Libraries, debuggers, performance tools, HPC/consumer applications, ISV applications and support
  - Education and research (350 universities teaching CUDA)

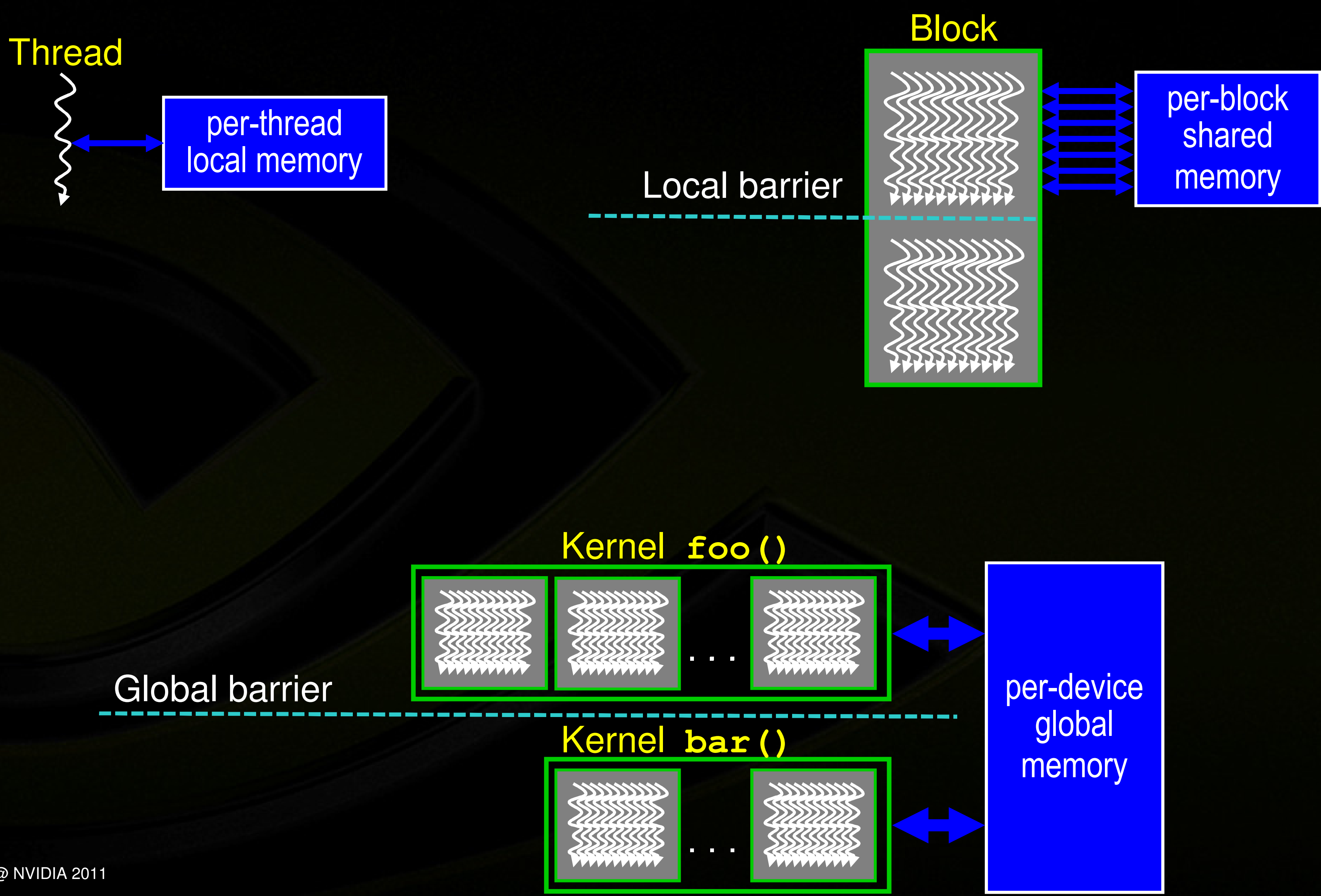
Throughput Processor Ingredients

- High arithmetic and memory bandwidth
- Throughput more important than latency
  - Hide DRAM latency with multithreading
- Explicit parallelism via fine-grained threads
  - Architecture
  - Programming system
- Hardware thread management
  - Thread creation/sync
  - Scheduling
  - Memory allocation





CUDA (Today) In One Slide



CUDA C Example

```
void saxpy_serial(int n, float a, float *x, float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}

// Invoke serial SAXPY kernel
saxpy_serial(n, 2.0, x, y);
```

Serial C Code

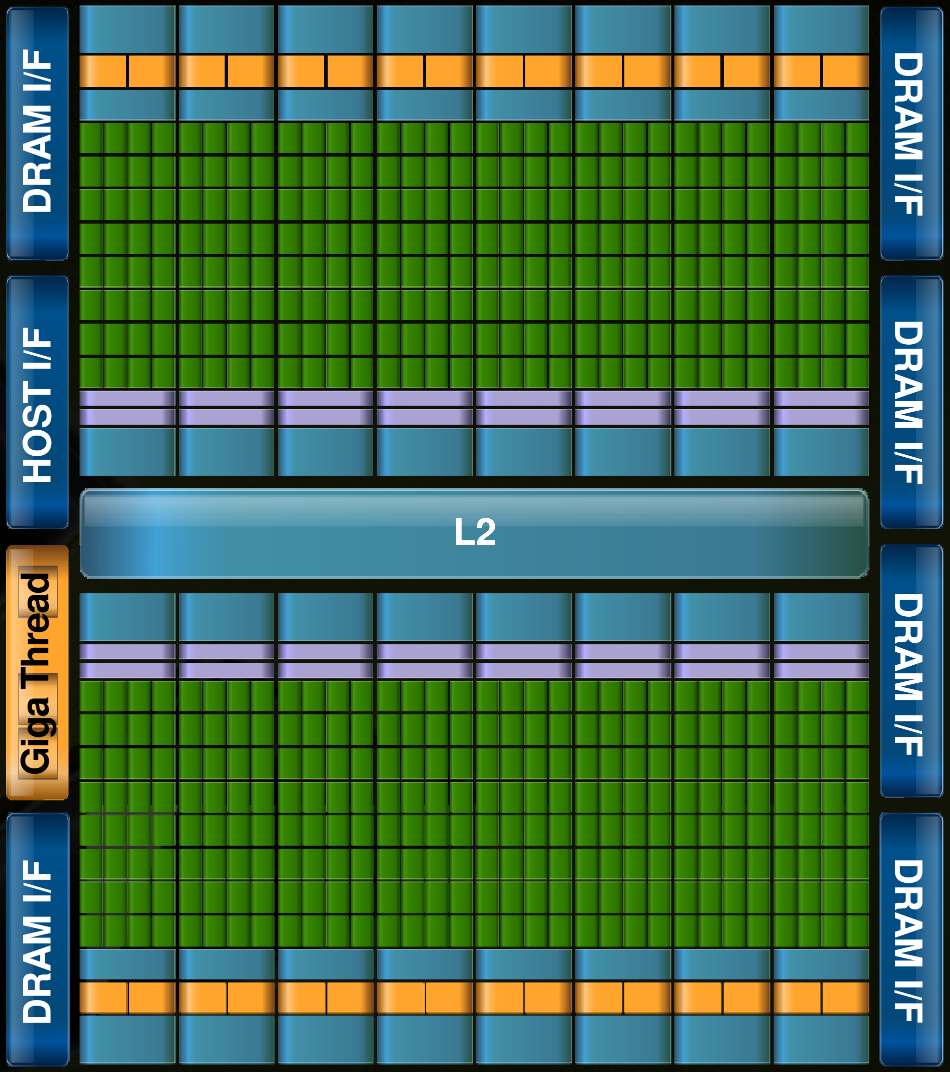
```
__global__ void saxpy_parallel(int n, float a, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n)  y[i] = a*x[i] + y[i];
}

// Invoke parallel SAXPY kernel with 256 threads/block
int nblocks = (n + 255) / 256;
saxpy_parallel<<<nblocks, 256>>>(n, 2.0, x, y);
```

Parallel C Code

NVIDIA Fermi GPU and Systems

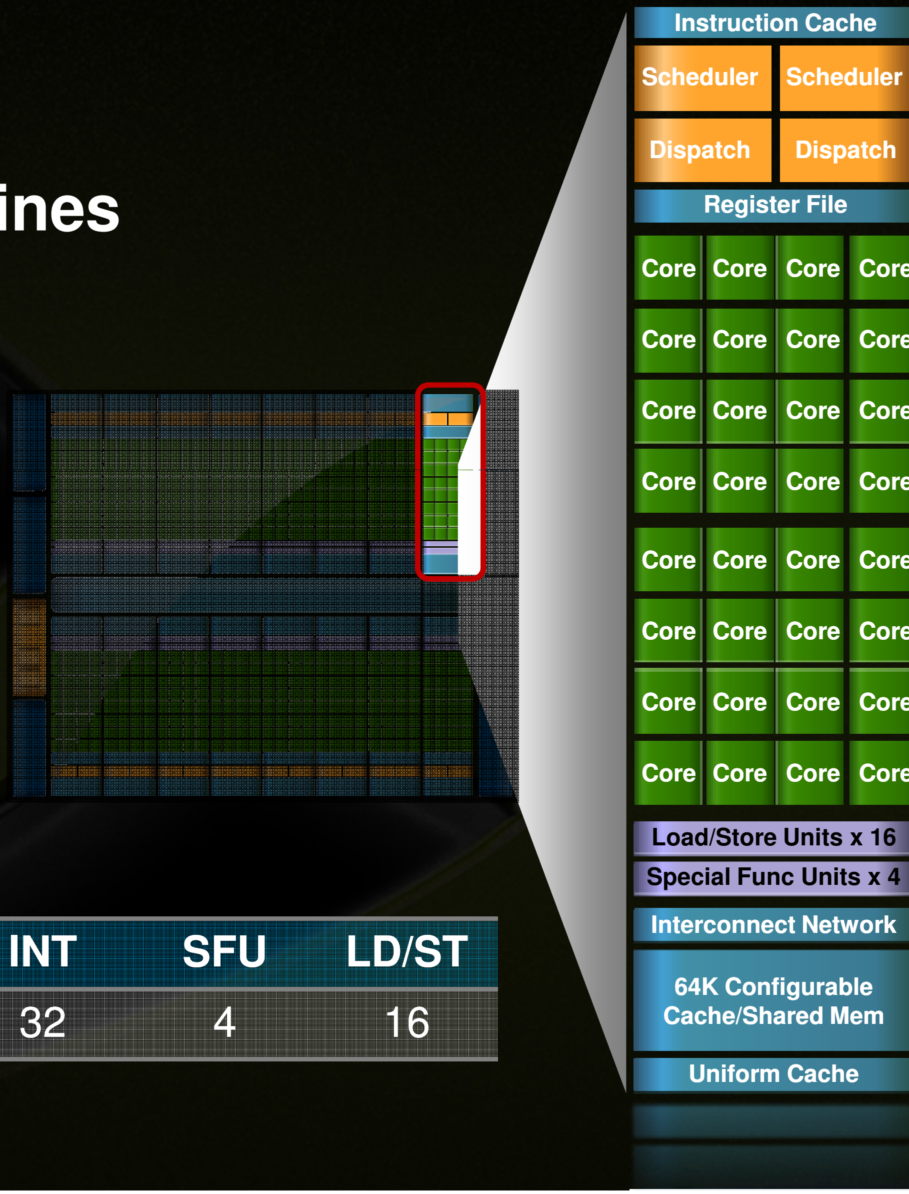
Fermi Focus Areas

- Expand performance sweet spot of the GPU
  - Caching
  - Concurrent kernels
  - FP64
  - More cores
  - More memory BW
- Bring more users, more applications to the GPU
  - C++
  - Visual Studio Integration
  - ECC



Streaming Multiprocessor (SM)

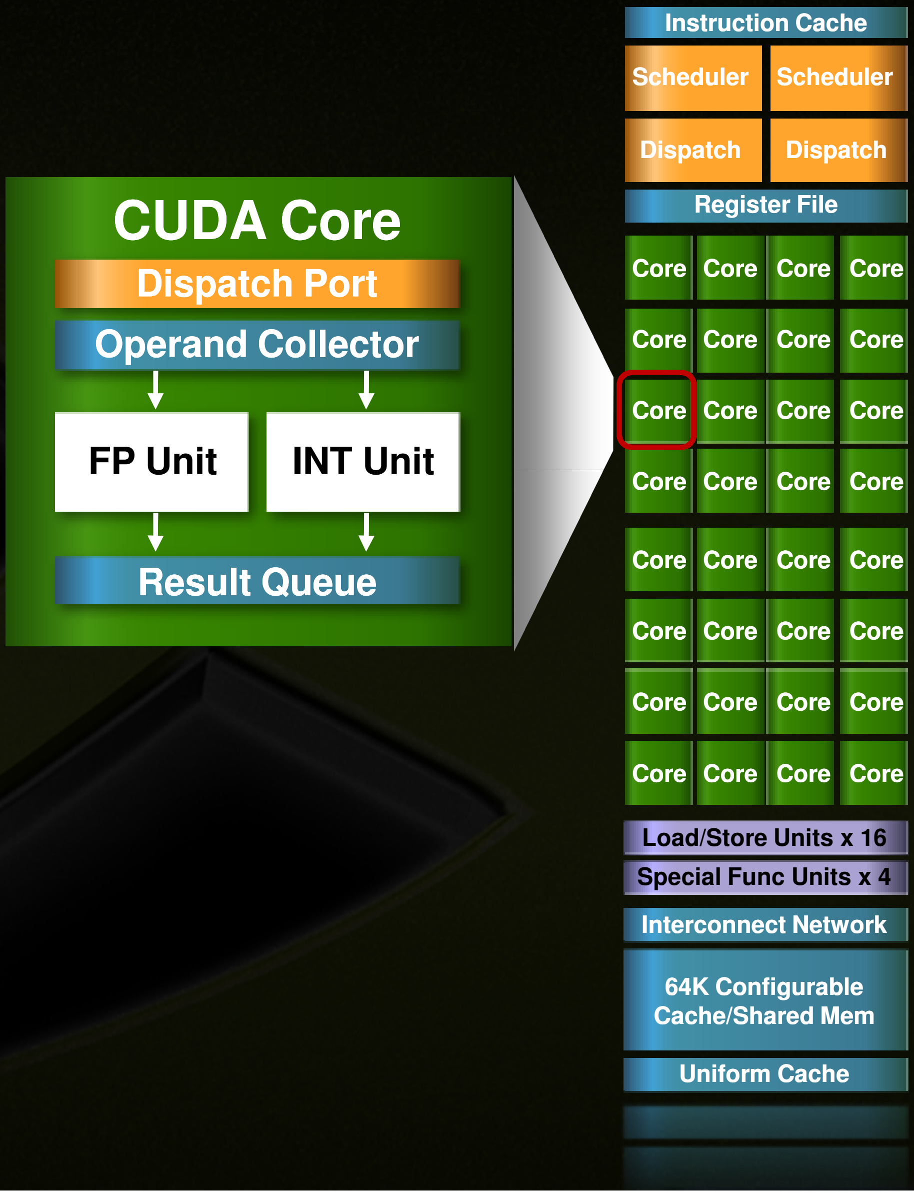
- Main computation engines
  - 16 SMs per Fermi chip
  - 32 “CUDA cores” per SM (512 total)



	FP32	FP64	INT	SFU	LD/ST
Ops / clk	32	16	32	4	16

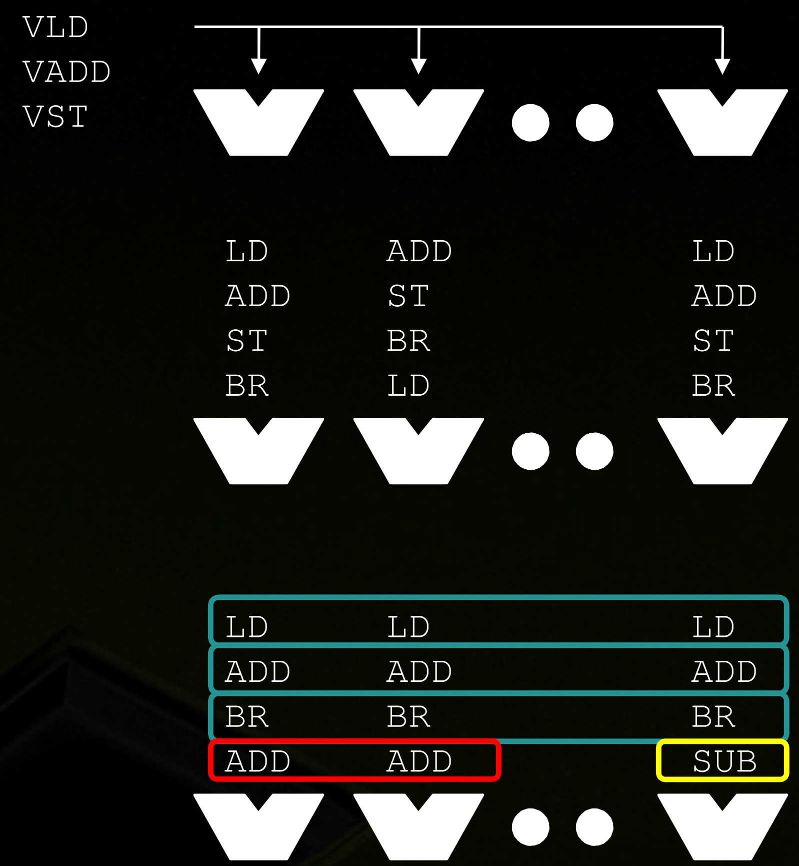
SM Microarchitecture

- Math Operations
  - IEEE 754-2008 arithmetic standard
  - Fused Multiply-Add (FMA) for SP & DP
  - Integer ALU optimized for 64-bit and extended precision ops
- Large local register file
- 64KB configurable local memory
  - Scratch and Cache
- SIMT microarchitecture



SIMD versus MIMD versus SIMT?

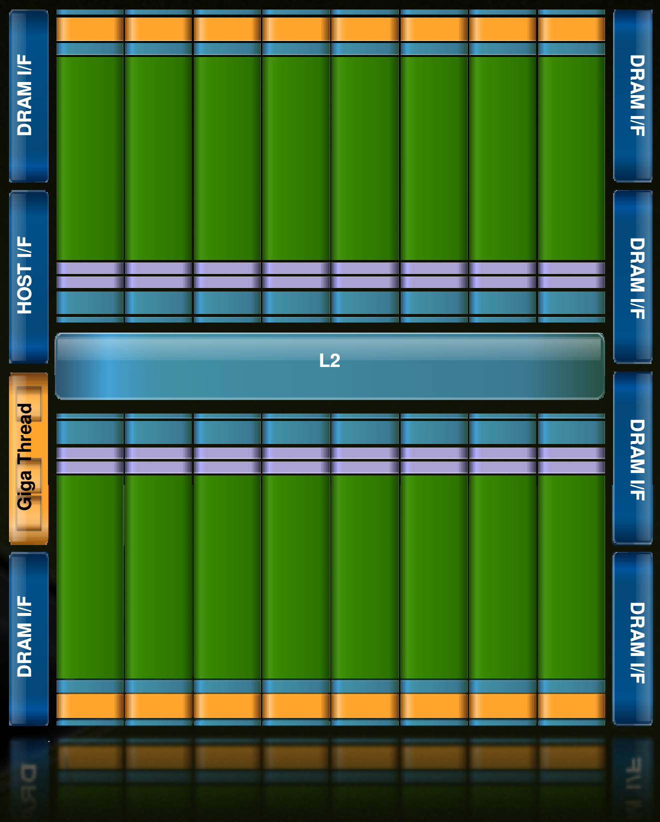
- **SIMD: Single Instruction Multiple Data**
- **MIMD: Multiple Instruction Multiple Data**
- **SIMT: Single Instruction Multiple Thread**



SIMT = MIMD Programming Model w/
SIMD Implementation Efficiencies

Memory Hierarchy

- True cache hierarchy + on-chip shared RAM
  - On-chip shared memory: regular memory access
    - dense linear algebra, image processing, …
  - Caches: irregular /unpredictable memory access
    - ray tracing, sparse matrix multiply, physics …
- Unified L2 Cache for all SMs (768 KB)
  - Fast, coherent data sharing across all cores in the GPU
- GDDR5 memory interface
  - 2x peak speed over GDDR3



	G80	GT200	Fermi
Transistors	681 million	1.4 billion	3.0 billion
CUDA Cores	128	240	512
Double Precision Floating Point	-	30 FMA ops/clock	256 FMA ops/clock
Single Precision Floating Point	128 MAD ops/clock	240 MAD ops/clock	512 FMA ops/clock
Special Function Units (per SM)	2	2	4
Warp schedulers (per SM)			2
Shared Memory (per SM)			Configurable 48/16 KB
L1 Cache (per SM)			Configurable 16/48 KB
L2 Cache			768 KB
ECC Memory Support	-	-	Yes
Concurrent Kernels	-	-	Up to 16
Load/Store Address Width	32-bit	32-bit	64-bit

Tesla C2050 Performance

515 DP GFlops
1.03 SP TFlops
144 GB/sec memory BW

NVIDIA Tesla GPUs Power 3 of Top 5 Supercomputers

#1 : Tianhe-1A

7168 Tesla GPUs 2.5 PFLOPS

#3 : Nebulae

4650 Tesla GPUs 1.2 PFLOPS

#4 : Tsubame 2.0

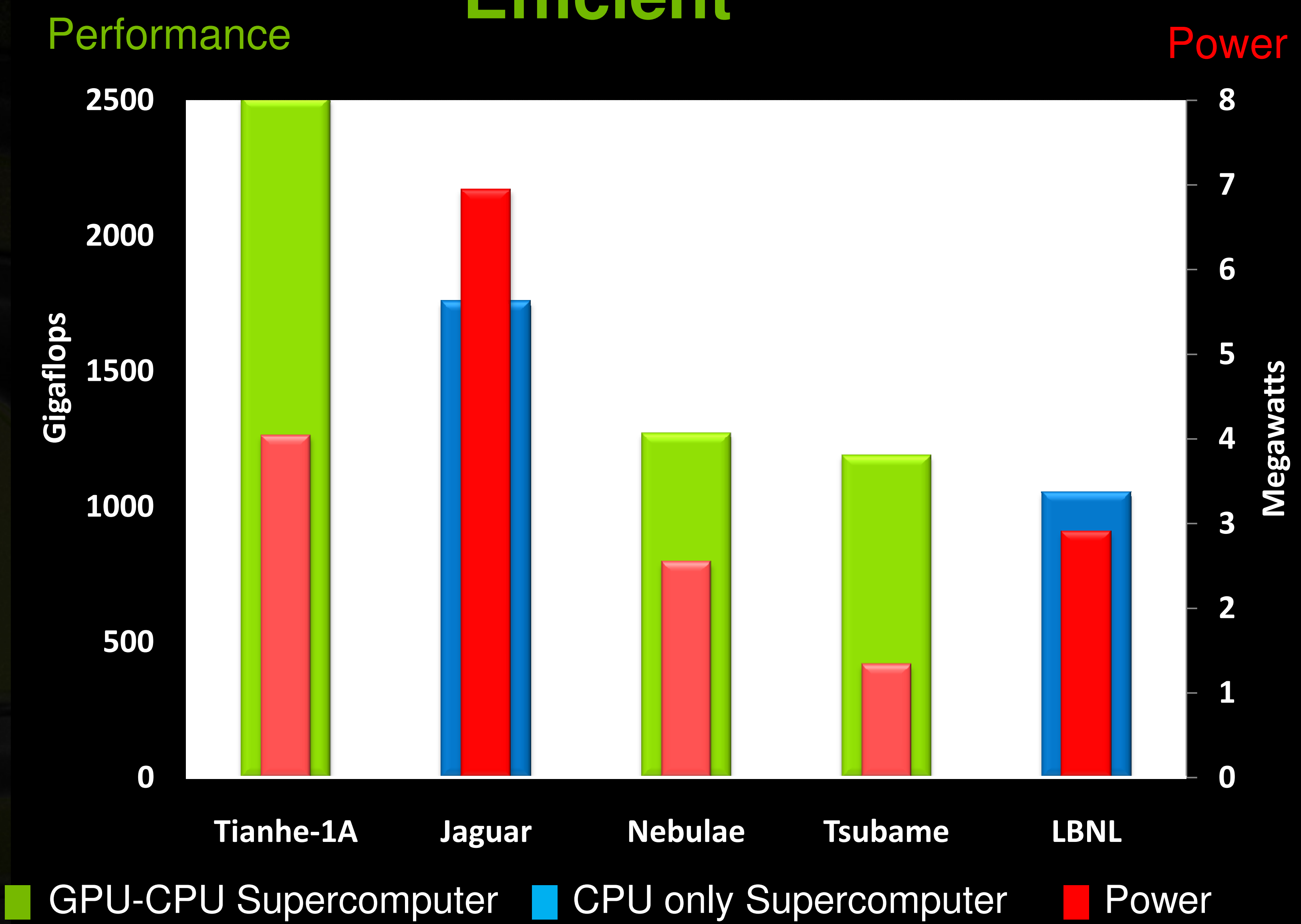
4224 Tesla GPUs 1.194 PFLOPS

8 more GPU accelerated machines in the November Top500

NVIDIA M2070 Module

NVIDIA C2070 PC Card

GPU Supercomputers: More Power Efficient



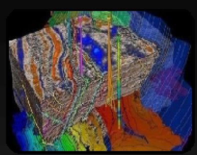
Sustained Performance (Optimized)

	Metric	CPU + GPU (Tesla 2050)	1 CPU Socket (3+ GHz 4-core Nehalem)
Linpack	GFlops	300+	~40
Sparse Matrix-Vector Multiply	GFlops	8	2
	Bandwidth (GB/sec)	100-140 (of 145)	--
Radix Sort	Million Keys/sec	800+	240

	Metric	CPU + GPU (Tesla 2050)	2 CPU Sockets (3+ GHz 4-core Nehalem)
Breadth-First Search	Billion Edges/sec	~1700	800-1000

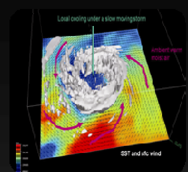
Wide Adoption of Tesla GPUs

Oil and gas



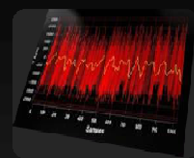
Reverse Time Migration
Kirchoff Time Migration
Reservoir Sim

Edu/Research



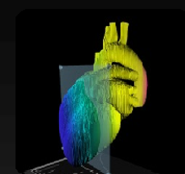
Astrophysics
Molecular Dynamics
Weather / Climate Modeling

Government



Signal Processing
Satellite Imaging
Video Analytics
Synthetic Aperture Radar

Life Sciences

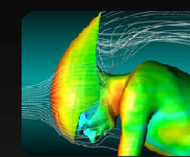


Bio-chemistry
Bio-informatics
Material Science
Sequence Analysis
Genomics

Finance

Risk Analytics
Monte Carlo
Options Pricing
Insurance modeling

Manufacturing



Structural Mechanics
Computational Fluid Dynamics
Machine Vision
Electromag.

Key Challenges for Parallel Systems

Power

Programming

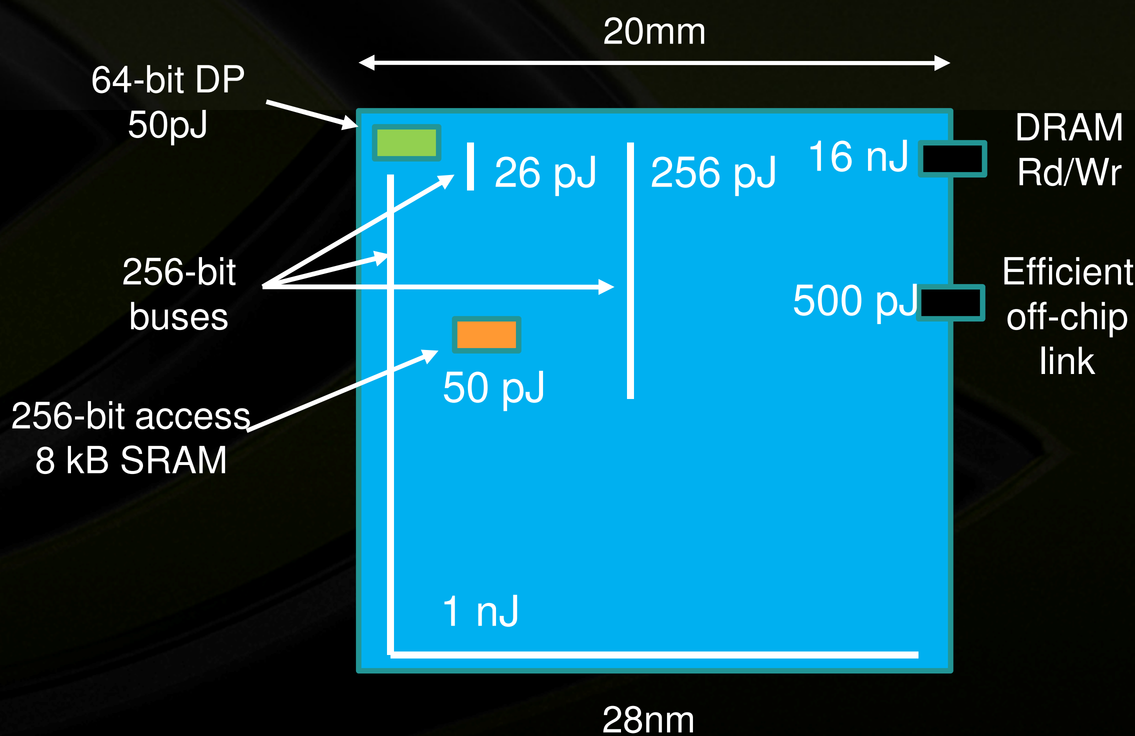
Power Constrained Computers

Energy Efficiency

- Today's chip-level efficiency (40nm)
  - CPUs: ~2nJ/FLOPS (DP FLOPS sustained)
  - GPUs: ~300pJ/FLOPS
- Future systems (e.g. ExaScale at 20MW)
  - 20pJ/FLOPS sustained across entire system
  - Similar efficiencies required at other envelopes
- Process scaling 40nm to 10nm will get us ~4x
- Need another 4x
  - Lower voltage and lower energy circuits
  - Energy-optimized architecture
  - Software

Where is the energy going?

- **Per-instruction overheads (speculation, OOO execution, etc.)**
  - **FP operation is just ~50pJ of 2nJ instruction**
- **Communication energy**



Processor Technology Projections

Processor Technology	28 nm (2011)	10nm High Perf (2017)	10nm Low Power (2017)
Vdd (nominal)	0.9 V	0.75 V	0.6 V
Frequency Target	1.5 GHz	2.5 GHz	2 GHz
DFMA energy	47 pJ	11.7 pJ (0.25x)	7.5 pJ (0.16x)
64b 8 KB SRAM Rd	14 pJ	5.4 pJ (0.25x)	2.3 pJ (0.16x)
Wire energy (Standard P&R)	486 fJ/trans/mm	303 fJ/trans/mm (0.61x)	194 fJ/trans/mm (0.39x)
Wire energy target (Engineered Channel)	111 fJ/trans/mm	69 fJ/trans/mm (0.61x)	44 fJ/trans/mm (0.39x)

Strategies for Energy Reduction

- Improve (physical) locality
  - Move bits less far: registers, memory
  - Drag fewer bits across the I/O pins
- Simplify architectures
  - Reduce per-instruction overheads
  - Push work from dynamic to static
- Reduce waste
  - Speculation/mis-speculation, prefetching, overfetching
- Push voltage down further
  - Dennard scaling is over, now an optimization process
  - More research in low-voltage circuits (e.g. RAMs)

Lots of interesting research problems here

Fundamental and Incidental Obstacles to Programmability

- Fundamental
  - Expressing 10^9 way parallelism
  - Expressing locality to deal with >100:1 global:local energy
  - Balancing load across 10^9 cores
- Incidental
  - Dealing with multiple address spaces
  - Partitioning data across nodes
  - Aggregating data to amortize message overhead

How will thread count scale?

For GPU-based systems with threads/SM chosen for memory latency tolerance

	2010: 4640 GPUs	**2018: 90K GPUs**
Threads/SM	1.5 K	~10^3
Threads/GPU	21 K	~10^5
Threads/Cabinet	672 K	~10^7
Threads/Machine	97 M	~10^9-10^{10}

Billion-fold parallel fine-grained threads for Exascale

Very simple hardware can provide

- Shared global address space (PGAS)
  - No need to manage multiple copies with different names
- Fast and efficient small (4-word) messages
  - No need to aggregate data to make Kbyte messages
- Efficient global block transfers (with gather/scatter)
  - No need to partition data by “node”
  - Vertical locality is still important

A Layered approach to Fundamental Programming Issues

- Hardware mechanisms for efficient communication, synchronization, and thread management
  - Programmer limited only by fundamental machine capabilities
- A programming model that expresses all available parallelism and locality
  - hierarchical thread arrays and hierarchical storage
- Compilers and run-time auto-tuners that selectively exploit parallelism and locality

What about legacy codes?

- Will continue to run – faster than they do now
- But…
  - They don't have enough parallelism to begin to fill the machine
  - Their lack of locality will cause them to bottleneck on global bandwidth
- As they are ported to the new model
  - The constituent equations will remain largely unchanged
  - The solution methods will evolve to the new cost model

Echelon

Extreme-scale Computer Hierarchies with Efficient Locality-Optimized Nodes

A DARPA UHPC-sponsored research project

Echelon Team

Penn

NVIDIA

CRAY

Oak Ridge National Laboratory

The University of California 1868

The University of Utah

The University of Texas at Austin

Georgia Institute of Technology

Leland Stanford Junior University 1891

Lockheed Martin

The University of Tennessee UT

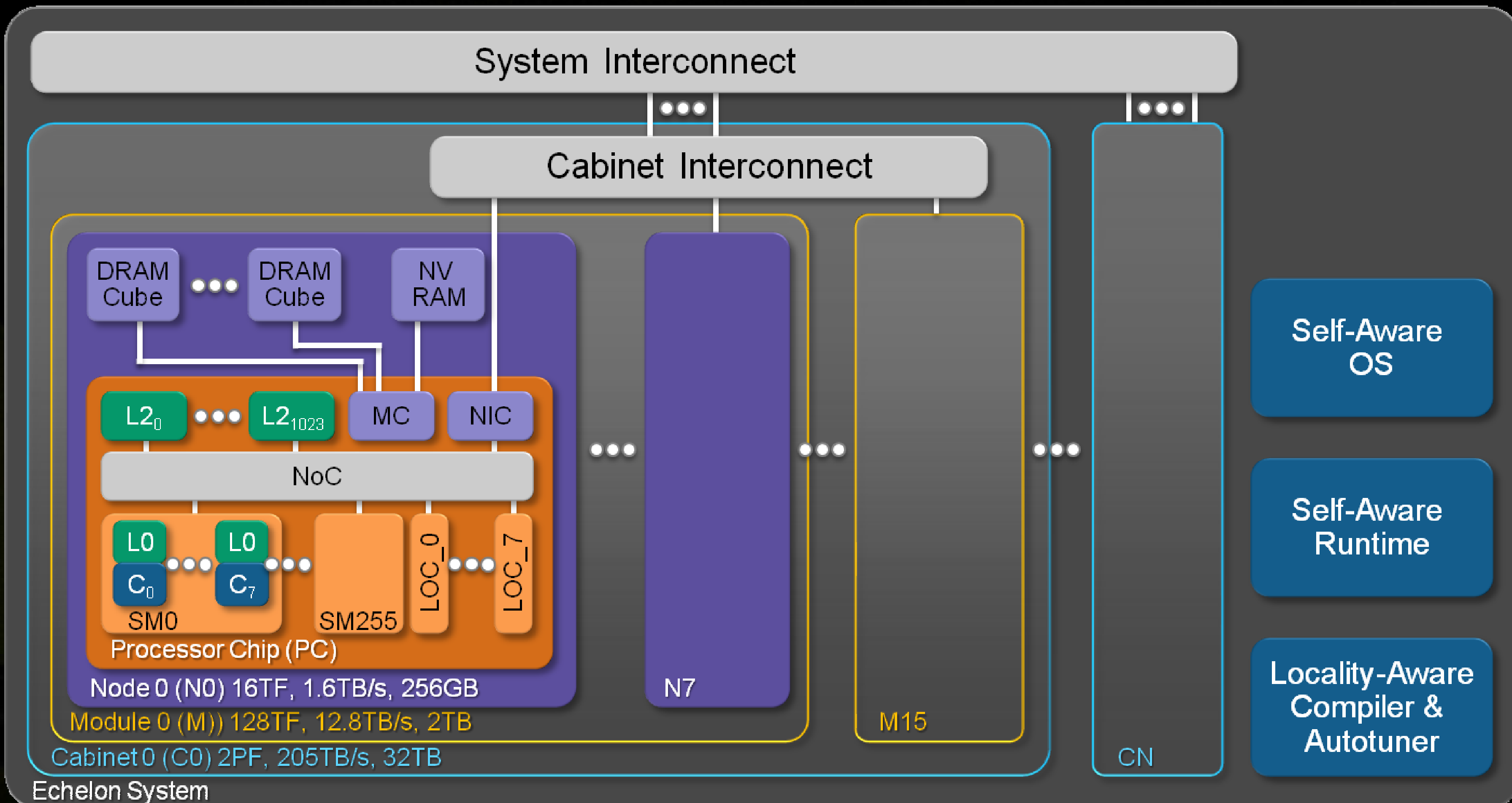
Objectives

- 100x better application energy efficiency over today's CPU systems.
- Improved programmer productivity
  - Time required to write a parallel program achieving a large fraction of peak efficiency is comparable to the time required to write a serial program today
- Strong scaling for many applications
  - Tens of millions of threads in rack, billions in Exascale
- High application mean-time to interrupt (AMTTI)
  - Low overhead, matched to application needs
- Machines resilient to attack

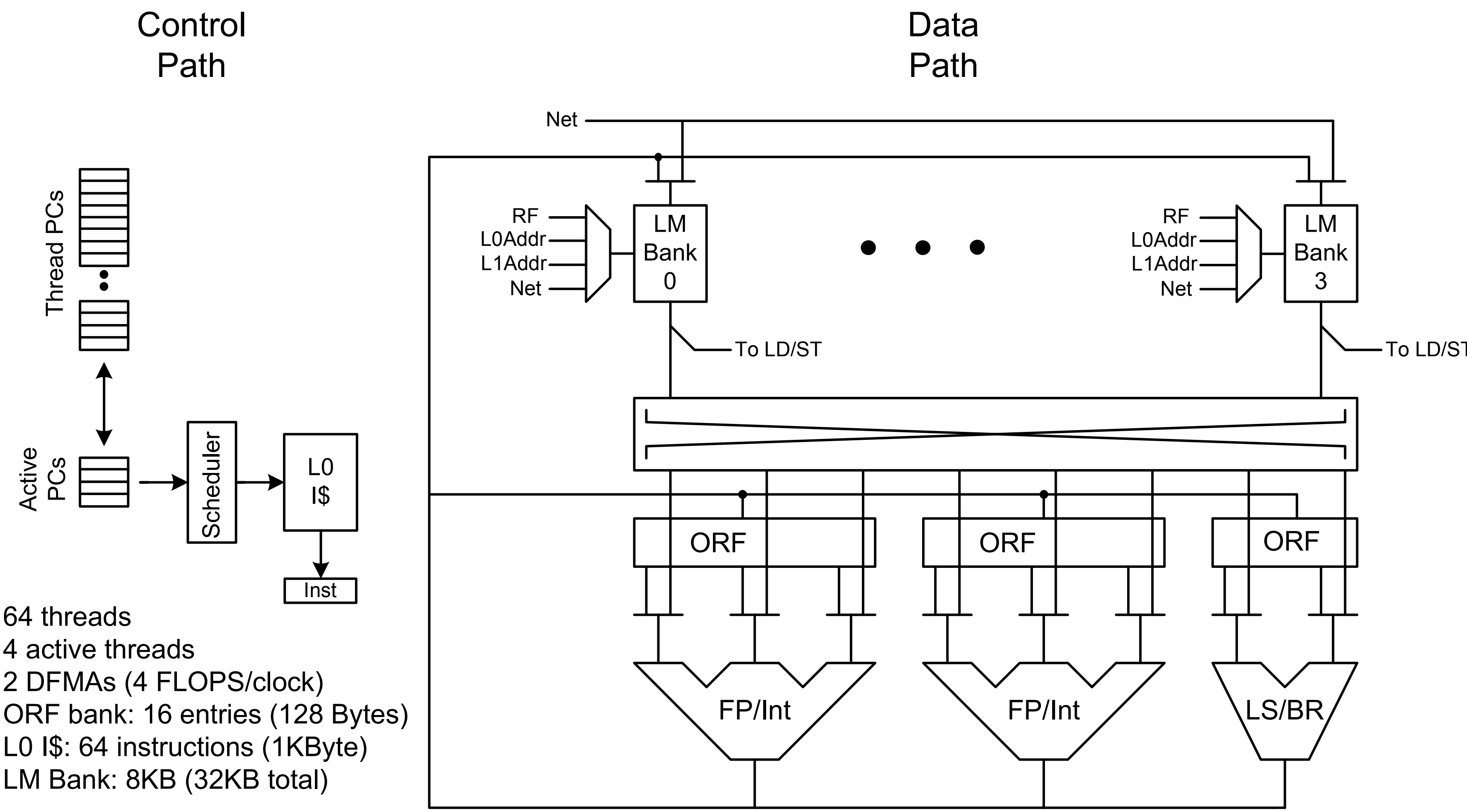
Approach

- **Energy challenge**
  - **Fine-grained parallel system with heterogeneous cores**
  - **Exposed and optimized vertical memory hierarchy**
- **Programming challenge**
  - **Global address space**
  - **Programs express concurrency/locality abstractly**
  - **Autotuning for hardware mapping**
  - **Software selective memory hierarchy configuration; selective coherence for non-critical data**
- **Resilience challenge**
  - **HW/SW cooperative resilience for energy- and performance-efficient fault protection**
  - **Guarded pointers for memory safety**

Echelon Node and System

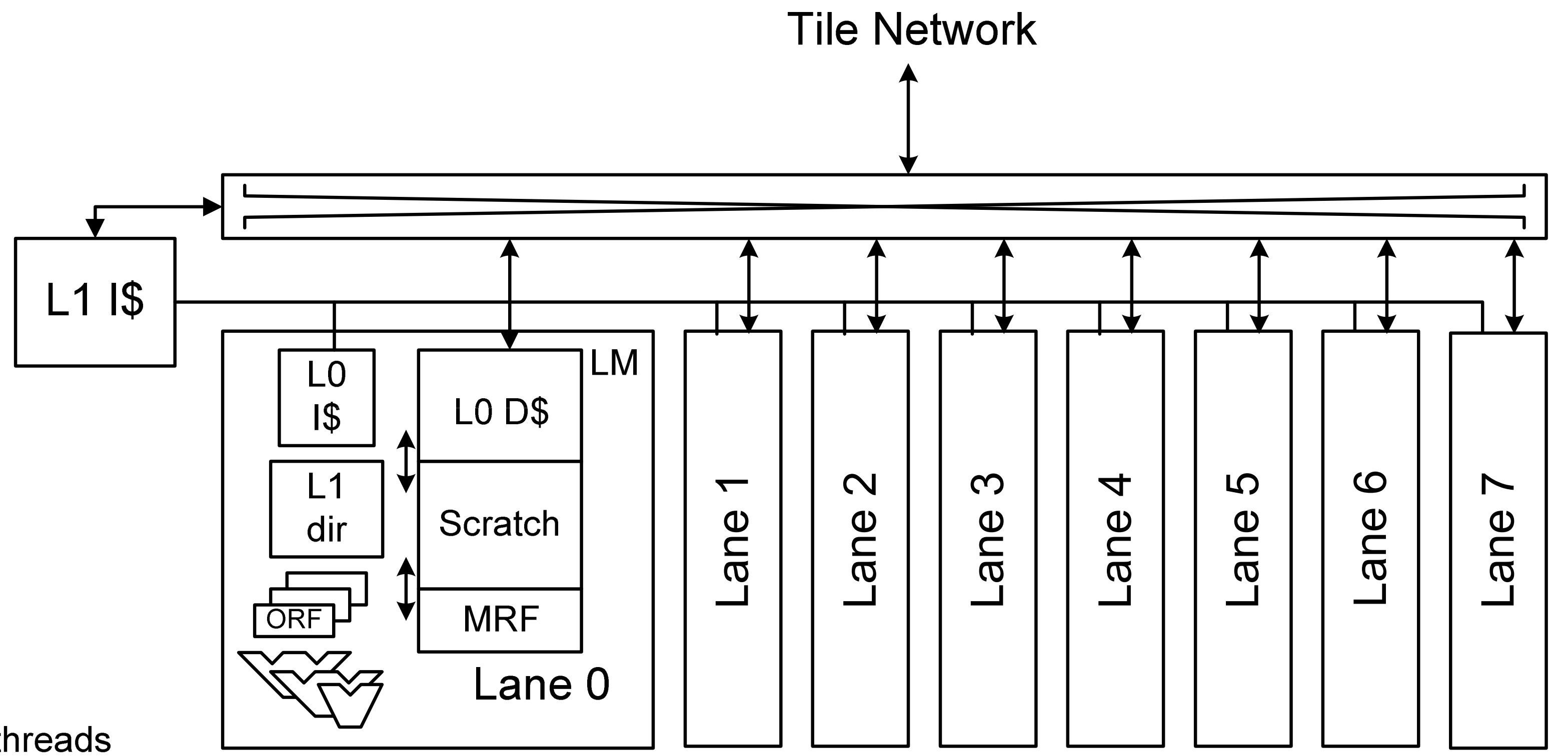


SM Lane Architecture



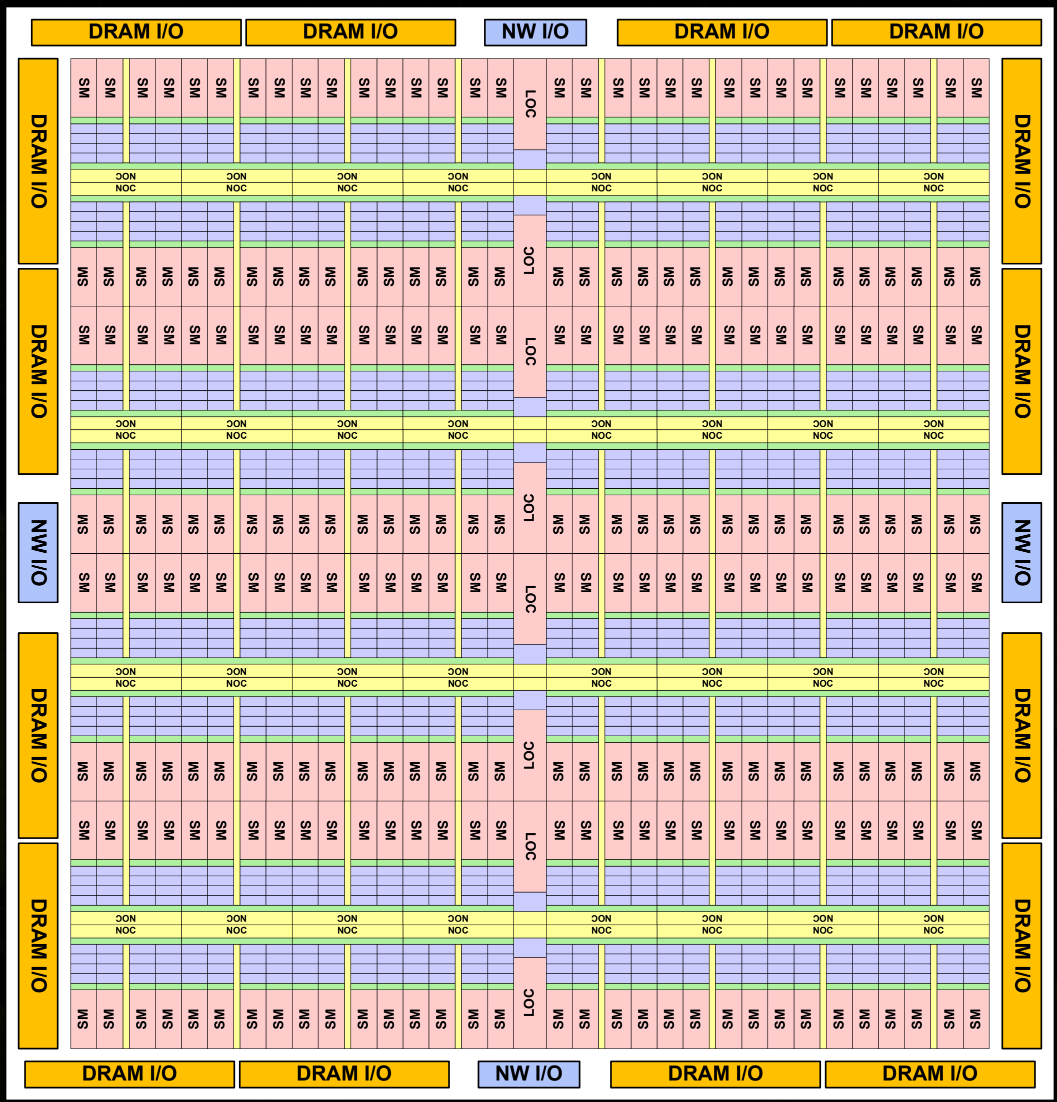
64 threads
4 active threads
2 DFMAs (4 FLOPS/clock)
ORF bank: 16 entries (128 Bytes)
L0 I$: 64 instructions (1KByte)
LM Bank: 8KB (32KB total)

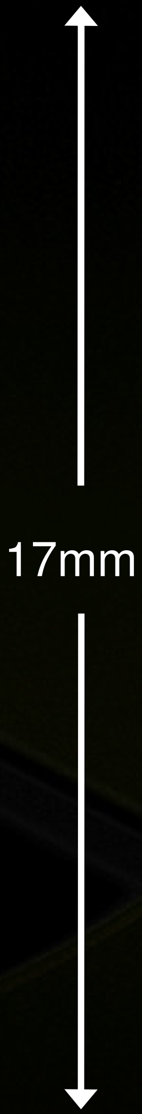
Streaming Multiprocessor (SM) Architecture

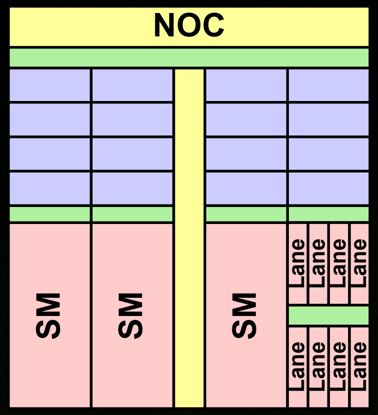


512 threads
32 active threads
16 DFMAs (32 FLOPs/clock)
L1 I$: 2K instructions (32KB)
RF/Scratch/D$: 256KB
L0 caches in other lanes form L1 cache

Echelon Chip Floorplan

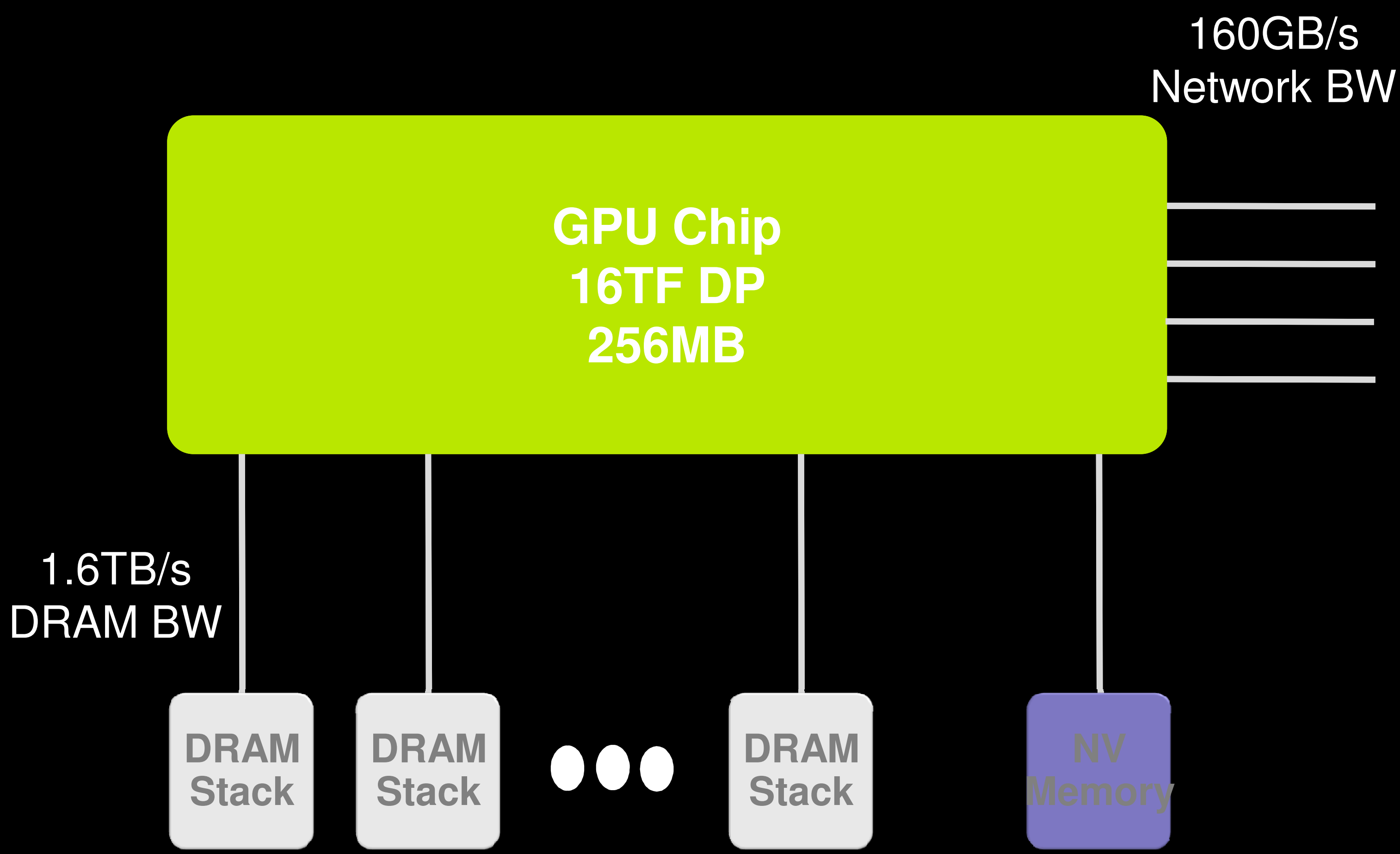




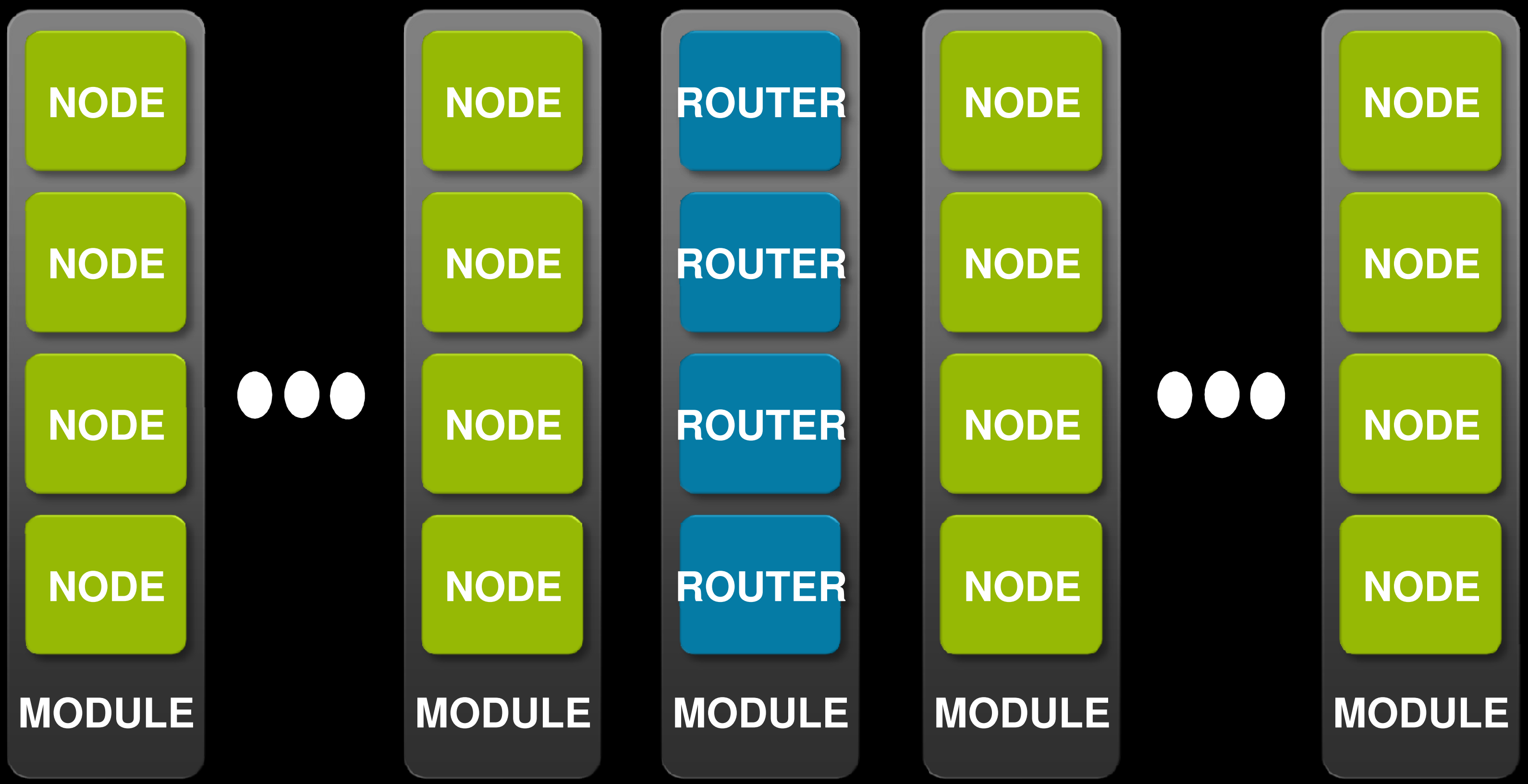


10nm process
$290mm^2$

Node MCM – 16 TF + 256GB



Cabinet: 128 Nodes, 2 PF, 38 kW



32 Modules, 4 Nodes/Module, Central Router Module(s), Dragonfly Interconnect

Exascale System

Dragonfly Interconnect
500 Cabinets is ~1EF and ~19MW

The Future of High Performance Computing

- Power constraints dictate extreme energy efficiency
- Programming systems are the long-pole in the tent
- All future interesting problems will be cast as throughput workloads
- GPUs are evolving to be *the* general-purpose throughput processors
- CPUs
  - Latency-optimized cores: important for Amdahl's law mitigation
  - But CPUs as we know them will become (already are?) "good enough", and shrink to a corner of the die/system

Questions?