UNC CHARLOTTE

The WILLIAM STATES LEE COLLEGE of ENGINEERING

TeCSAR

Transformative Computer Systems and Architecture Research

Real-time AI
Lecture 2:Intro to GPUs

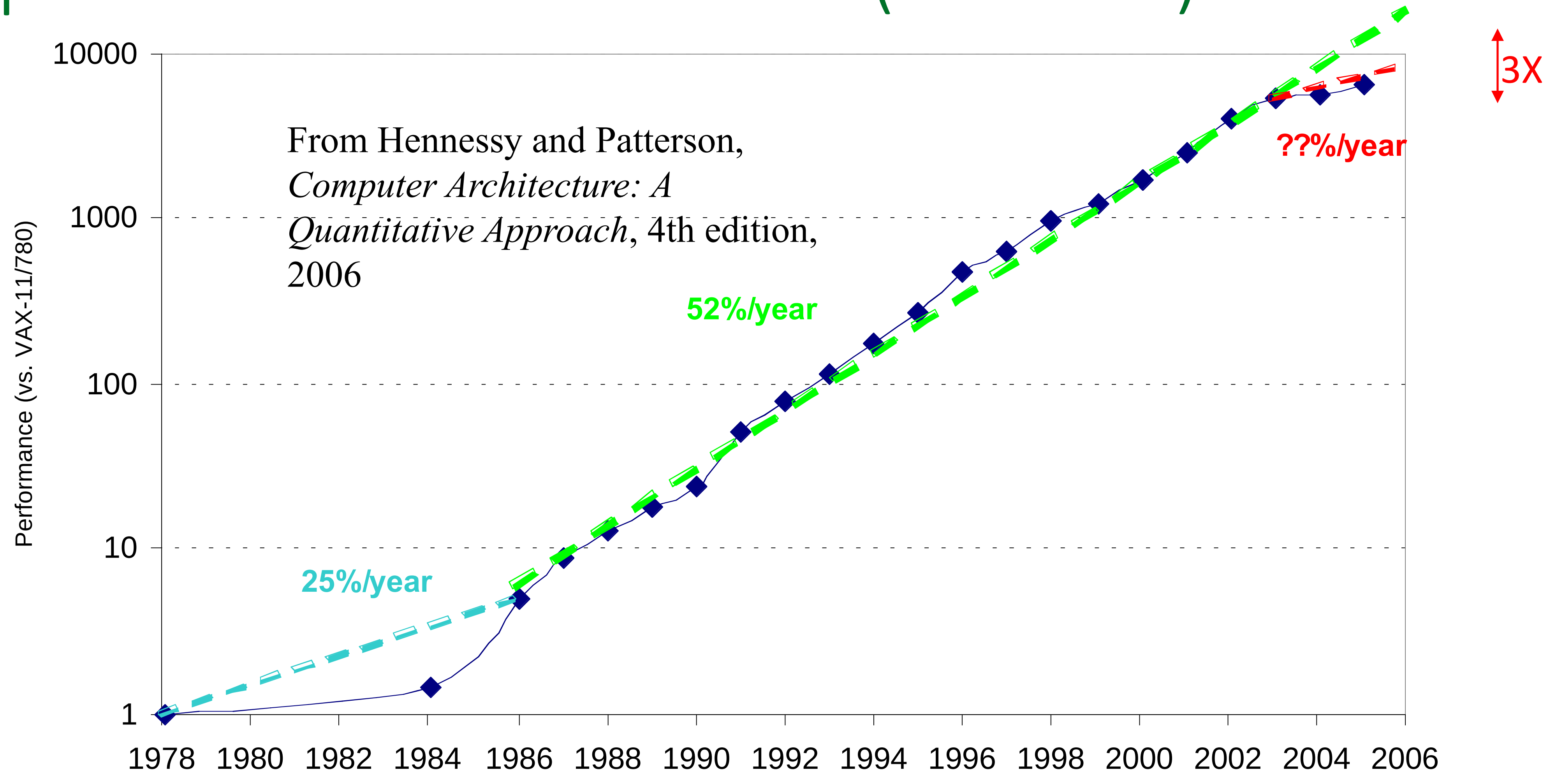
Hamed Tabkhi

Department of Electrical and Computer Engineering,

University of North Carolina Charlotte (UNCC)

htabkhiv@uncc.edu

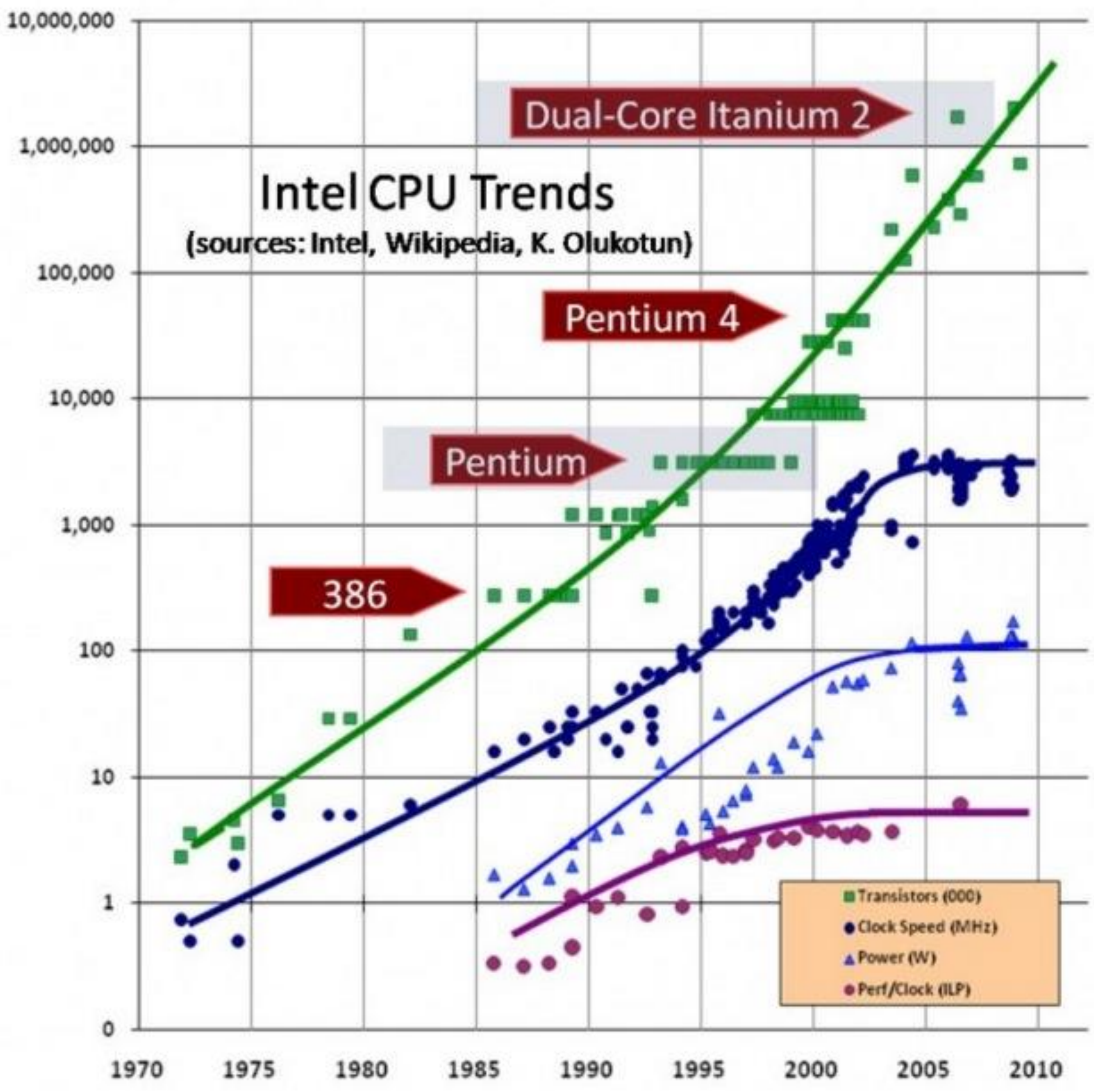
Uniprocessor Performance (SPECint)



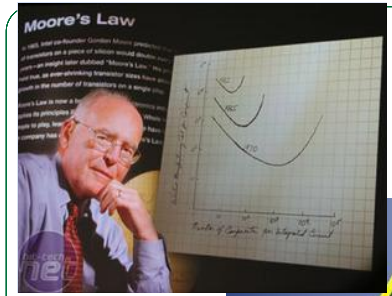
- **VAX : 25%/year 1978 to 1986**
- **RISC + x86: 52%/year 1986 to 2002**
- **RISC + x86: ??%/year 2002 to present**

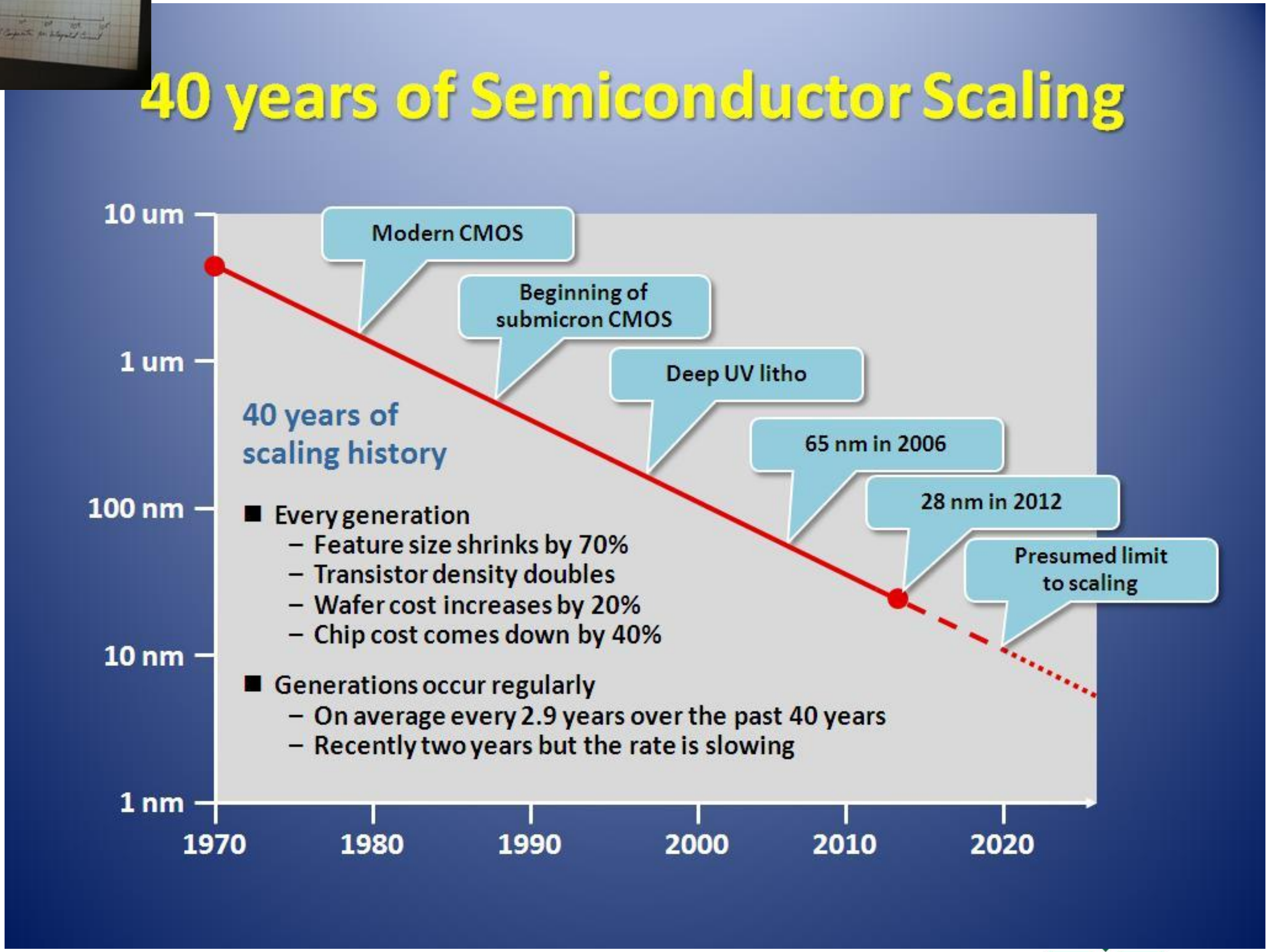
The WILLIAM STATES LEE COLLEGE *of* ENGINEERING
UNC CHARLOTTE

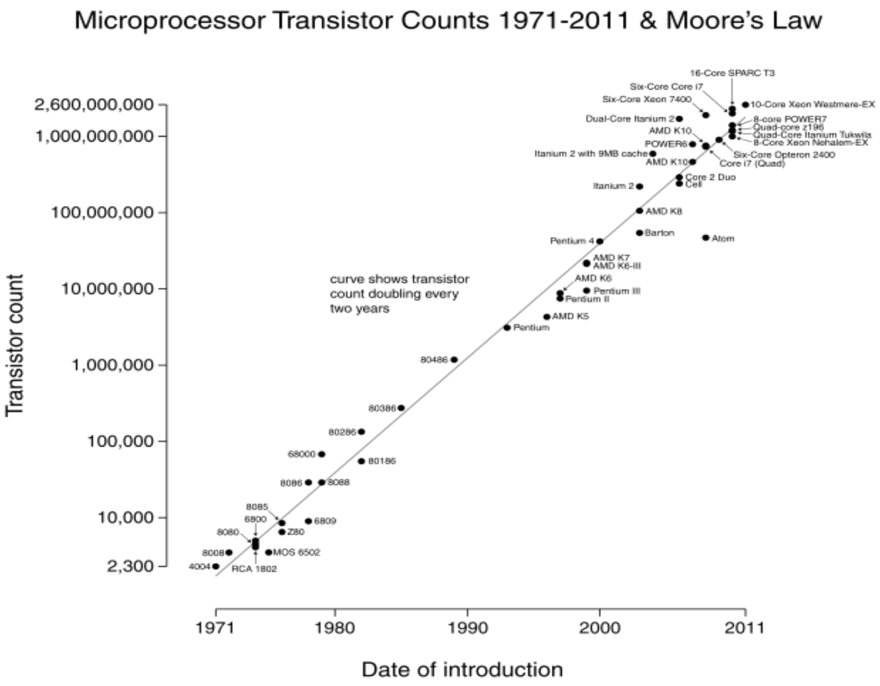
Power, Frequency, ILP



The WILLIAM STATES LEE COLLEGE of ENGINEERING
UNC CHARLOTTE



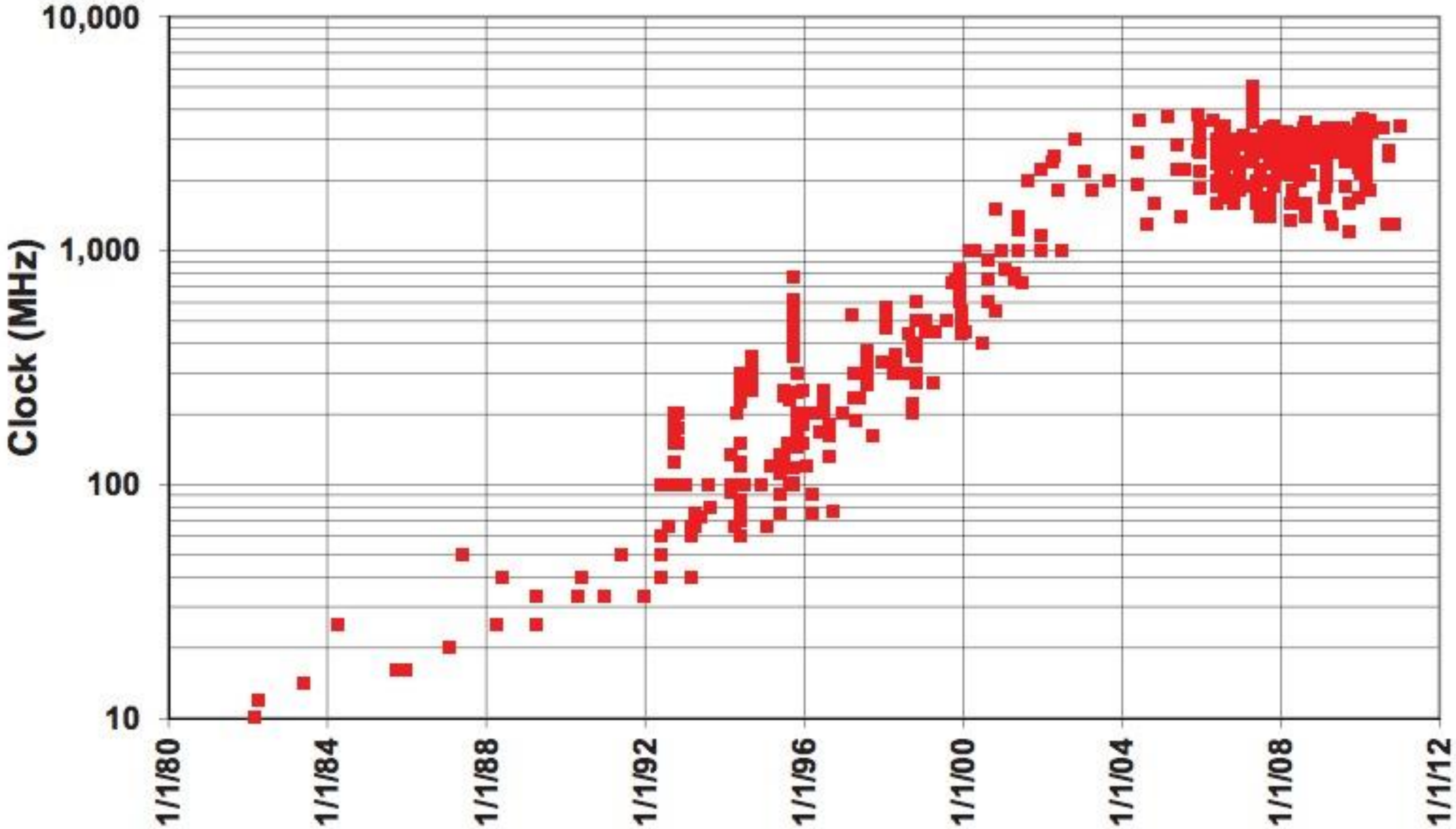




2027?
5nm

The WILLIAM STATES LEE COLLEGE *of* ENGINEERING
UNC CHARLOTTE

Frequency Has Stopped Scaling Too



The WILLIAM STATES LEE COLLEGE of ENGINEERING
UNC CHARLOTTE

Heterogeneous Architecture for Domain–Specific Computing

History

2007 - NVIDIA CUDA

- First GPGPU solution, restricted to NVIDIA GPUs

2007 - AMD Stream SDK (previously CTM)

2009 - OpenCL, Direct Compute

2011 - OpenCL revision 1.2

2012 - NVIDIA Kepler Architecture

2013 – OpenCL revision 2.0

2014 – NVIDIA Maxwell Architecture

2016 – OpenCL revision 2.2, NVIDIA Pascal

2017 – NVIDIA OpenCL 2.0 beta

GPU in comparison with CPU

- CPU

  - Few cores per chip
  - General purpose cores
  - Processing different threads
  - Huge caches to reduce memory latency
    - Locality of reference problem

- GPU

  - Many cores per chip
  - Cores specialized for numeric computations
  - SIMT thread processing
  - Huge amount of threads and fast context switch
    - Results in more complex memory transfers

NVIDIA Fermi

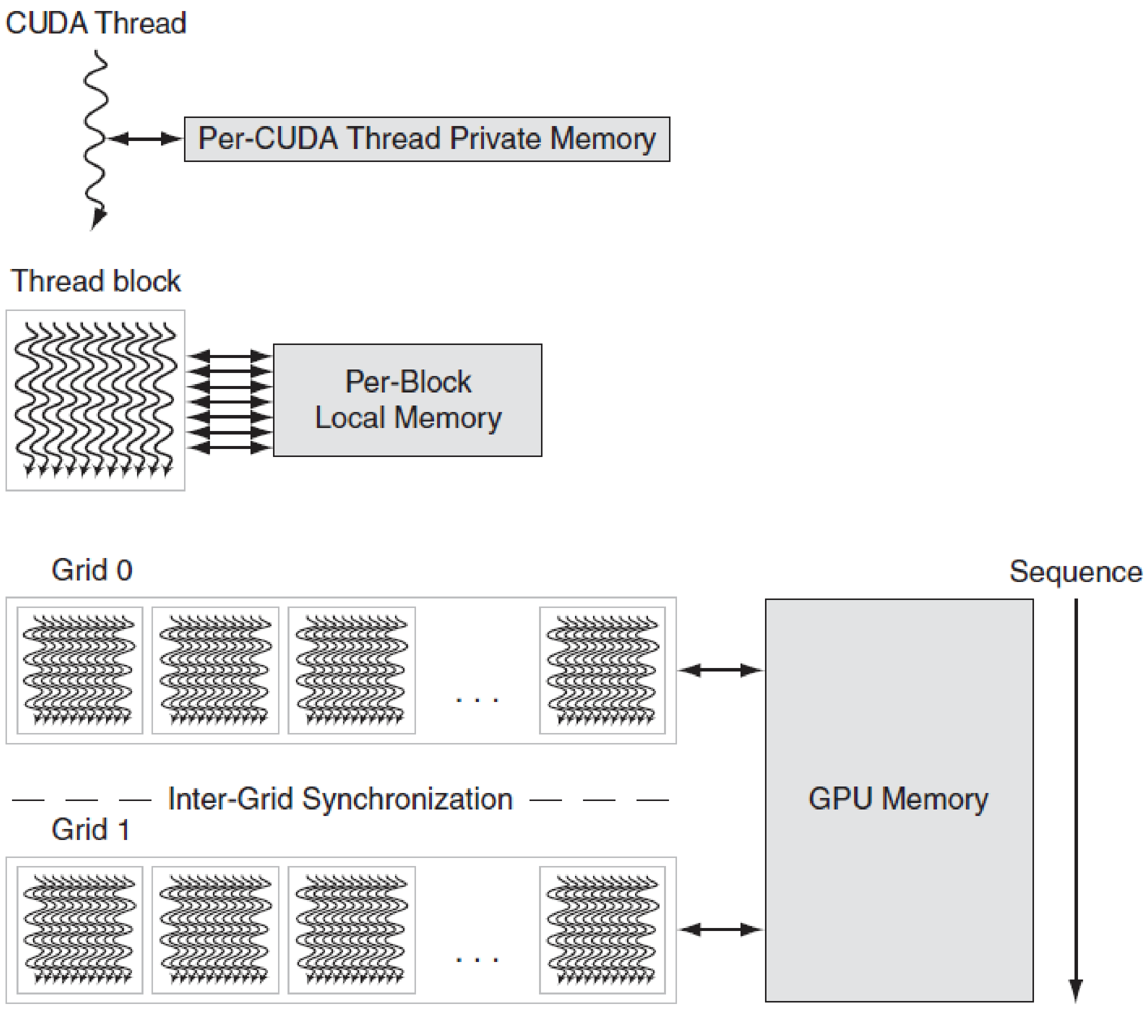
- GPU Architecture
  - 16 SMP units
  - 512 CUDA cores
  - 786kB L2 cache

Note that one CUDA core corresponds to one 5D AMD Stream Processor (VLIW5). Therefore Radeon 5870 has 320 cores with 4-way SIMD capabilities and one SFU.

GPU Execution Model

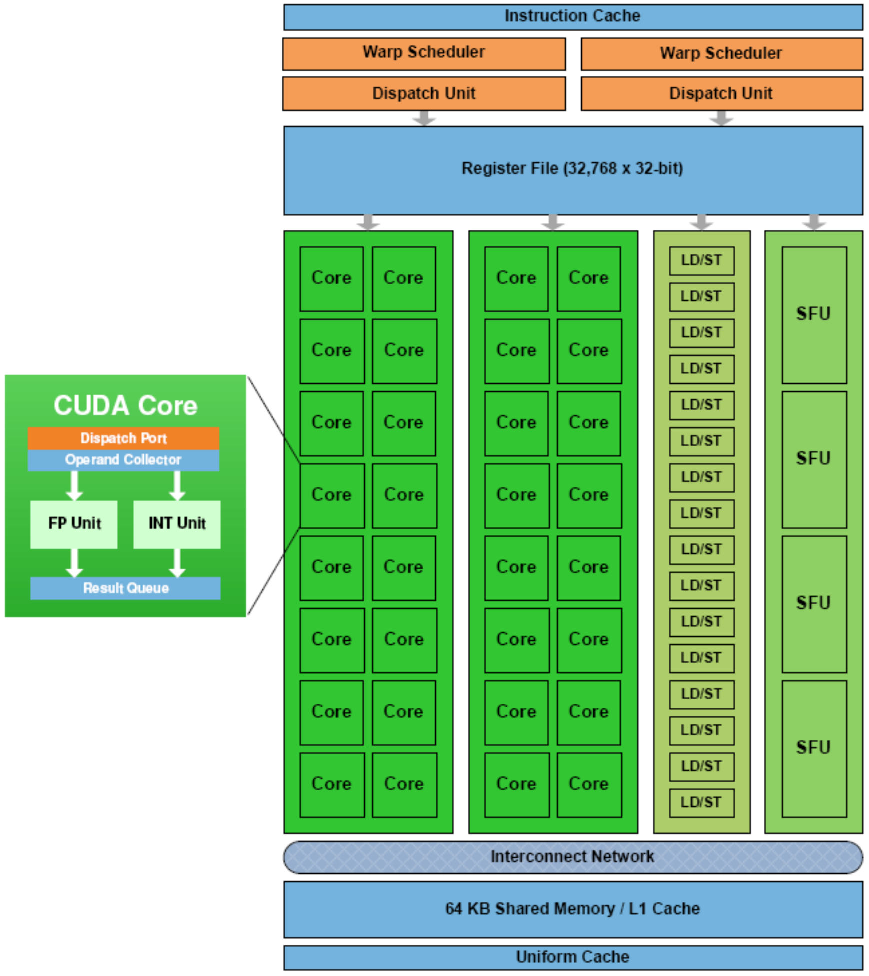
- Data Parallelism
  - Many data elements are processed concurrently by the same routine
  - GPUs are designed under this particular paradigm
    - Also have limited ways to express task parallelism
- Threading Execution Model
  - One function (kernel) is executed in many threads
    - Much more lightweight than the CPU threads
  - Threads are grouped into blocks/work groups of the same size

GPU Memory Structures



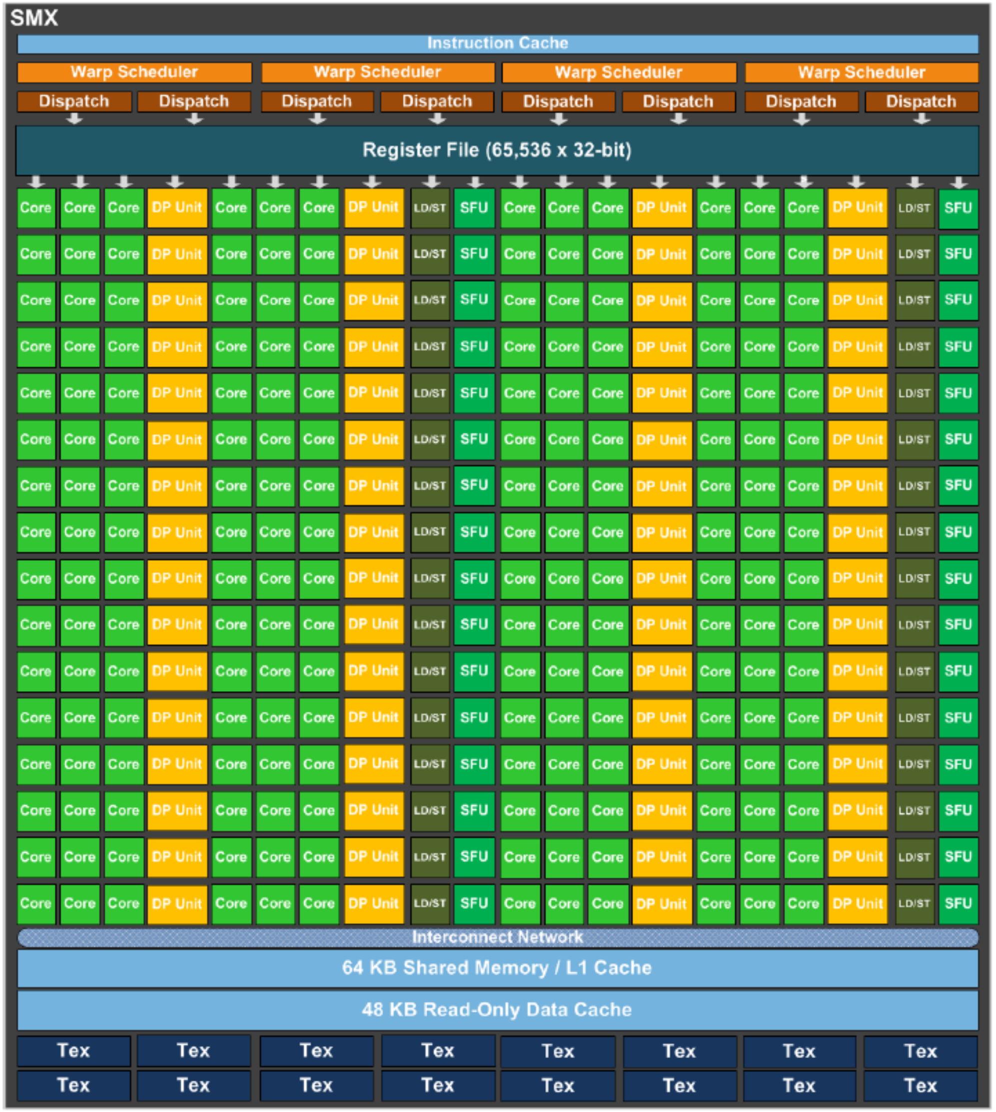
The WILLIAM STATES LEE COLLEGE *of* ENGINEERING
UNC CHARLOTTE

NVIDIA Fermi



- Streaming Multiprocessor
  - 32 CUDA cores
  - 64kB shared memory (or L1 cache)
  - 1024 registers per core
  - 16 load/store units
  - 4 special function units
  - 16 double precision ops per clock
  - 1 instruction decoder
    - All cores are running in lockstep

Kepler Architecture



- Kepler's Major Improvements
  - Streaming Processors Next Generation (SMX)
    - 192 cores, 32 SFUs, 32 load/store units
    - 3 cores share a DP unit, 6 cores share LD and SFU
  - Dynamic Parallelism
    - Kernel may spawn child kernels (to depth of 24)
    - Implies the work group context-switch capability
  - Hyper-Q
    - Up to 32 simultaneous GPU-host connections
    - Better throughput if multiple processes/threads use the GPU (concurrent connections are managed in HW)

The WILLIAM STATES LEE COLLEGE of ENGINEERING
UNC CHARLOTTE

Maxwell Architecture

- Maxwell's Major Improvements
  - Maxwell Symmetric Multiprocessor (SMM)
    - Many internal optimizations, better power efficiency
    - Improved scheduling, increased occupancy
    - Reduced arithmetic instruction latency
  - Larger L2 cache (2MB)
  - Dedicated shared memory (separate L1 cache)
  - Native shared memory atomics
  - Better support for dynamic parallelism

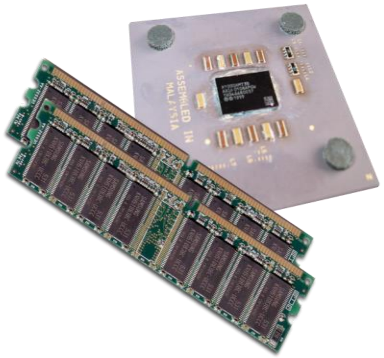
Volta Architecture

Pascal Architecture

The WILLIAM STATES LEE COLLEGE *of* ENGINEERING
UNC CHARLOTTE

Heterogeneous Computing

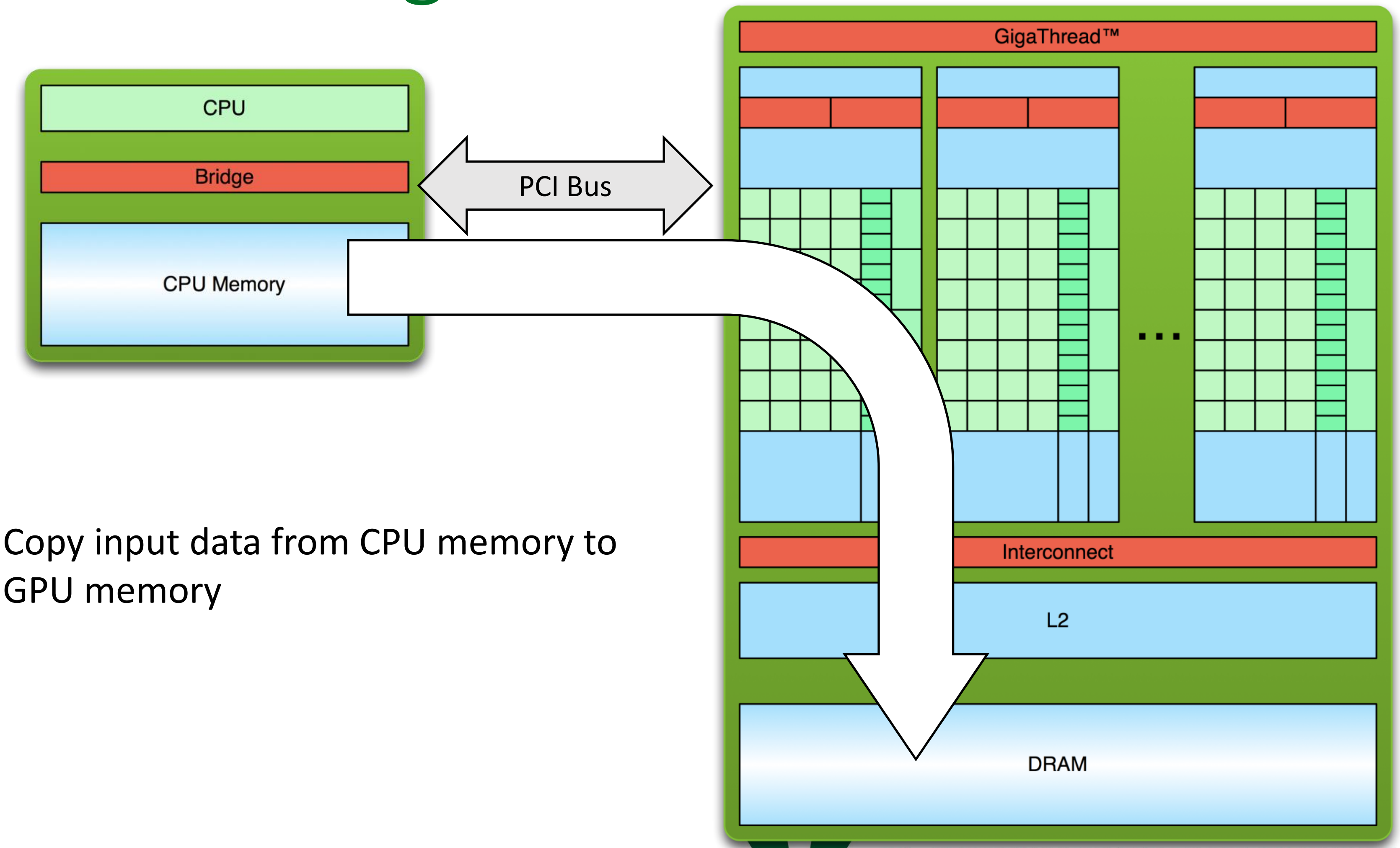
- Terminology:
  - *Host* The CPU and its memory (host memory)
  - *Device* The GPU and its memory (device memory)



Host

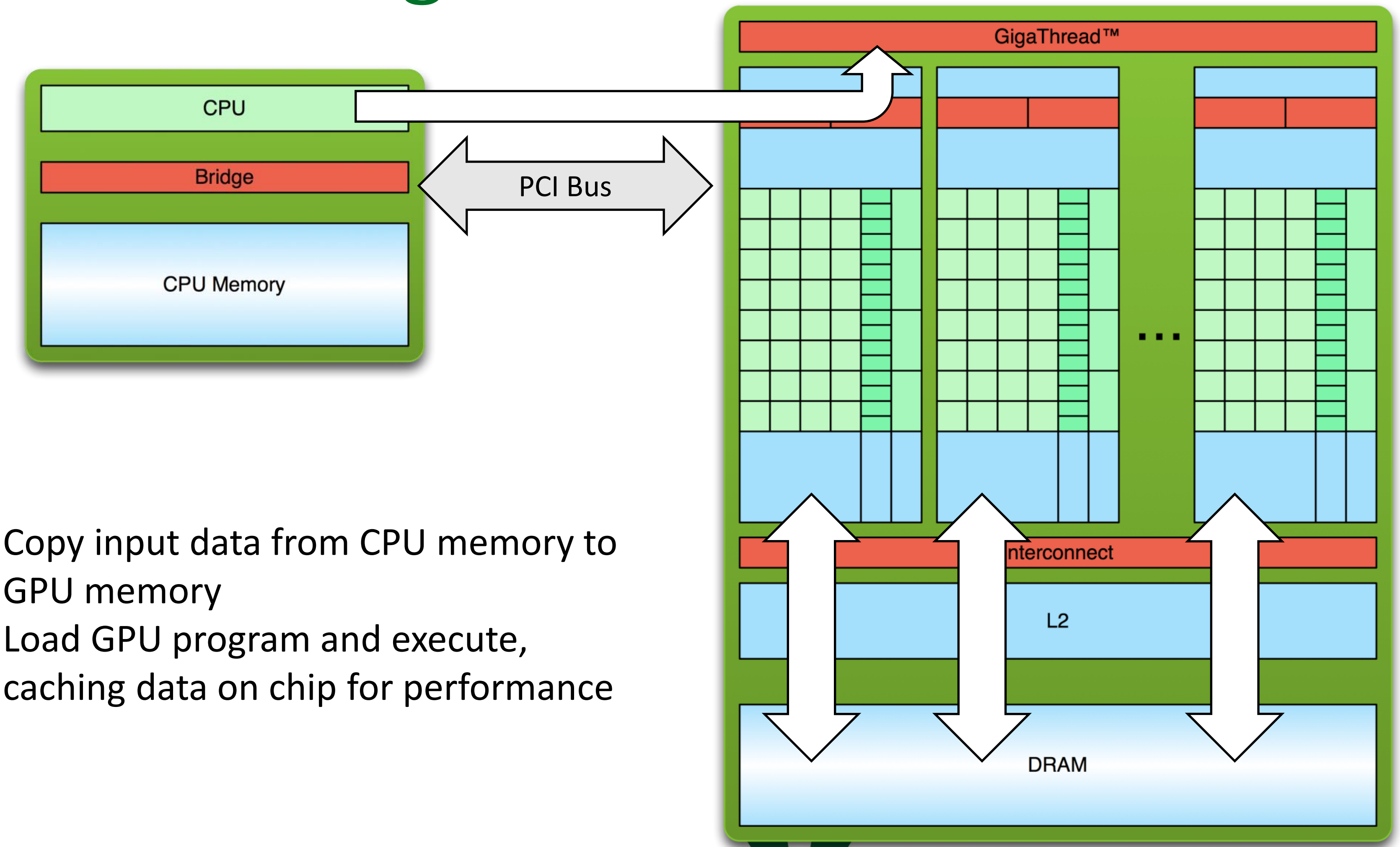
Device

Simple Processing Flow



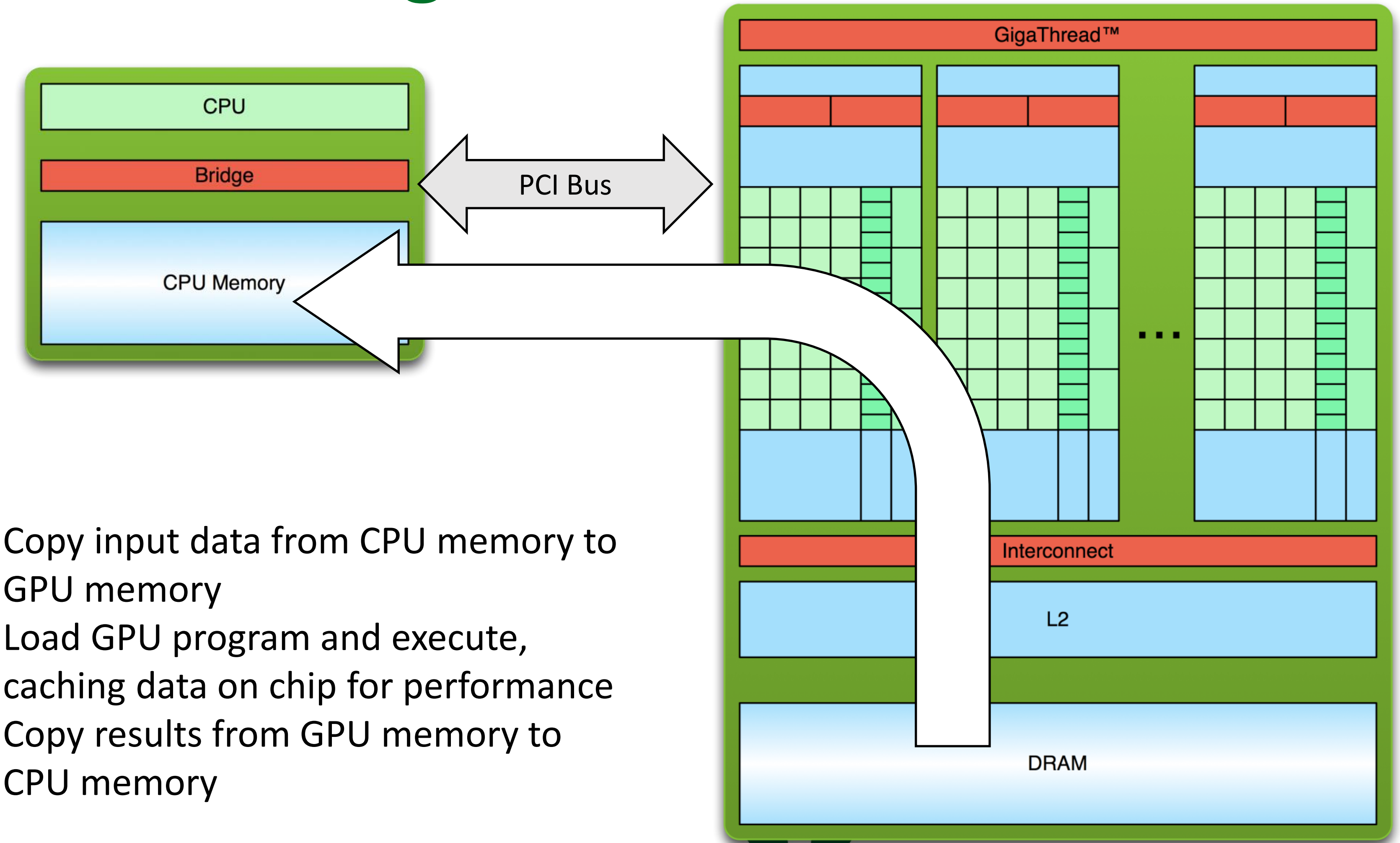
1. Copy input data from CPU memory to GPU memory

Simple Processing Flow



1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance

Simple Processing Flow



1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance
3. Copy results from GPU memory to CPU memory