EECE 528:
Parallel & Reconfigurable Computing

Lecture #1

Motivational Background 1

- Landscape
  - Continuously shrinking transistor
  - 4GHz limit // power wall circa 2004
- Problem
  - How to get more performance? (computations per second)

Motivational Background 2

- Trivial solution: just add ALUs
  - Serial
    - R1 ← R2 + 3 @ time t1
    - R3 ← R4 + R2 @ time t2 > t1, reuses ALU
  - Parallel
    - R1 ← R2 + 3 R3 ← R4 + R2 @ time t1

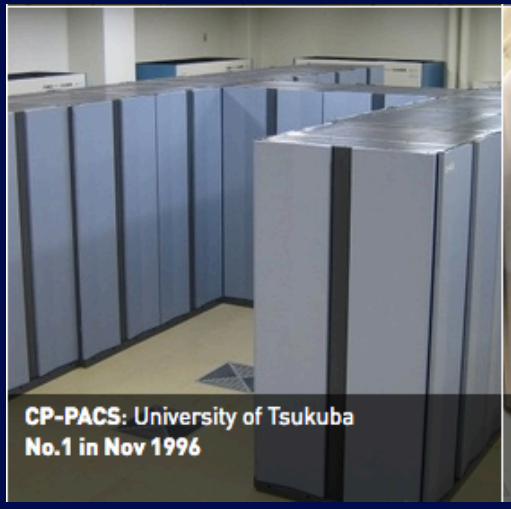
- Then, what makes it ***so hard***?

Top500 over the years…

Numerical Wind Tunnel: National Aerospace Laboratory of Japan
No.1 in Nov 1993

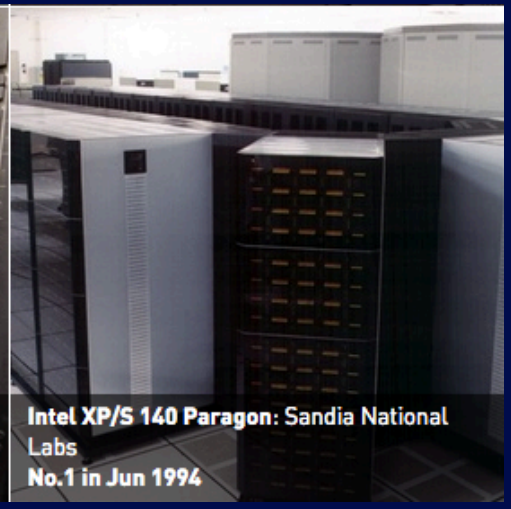
CM-5: Los Alamos National Lab
No.1 in Jun 1993

TOP 500
The List.



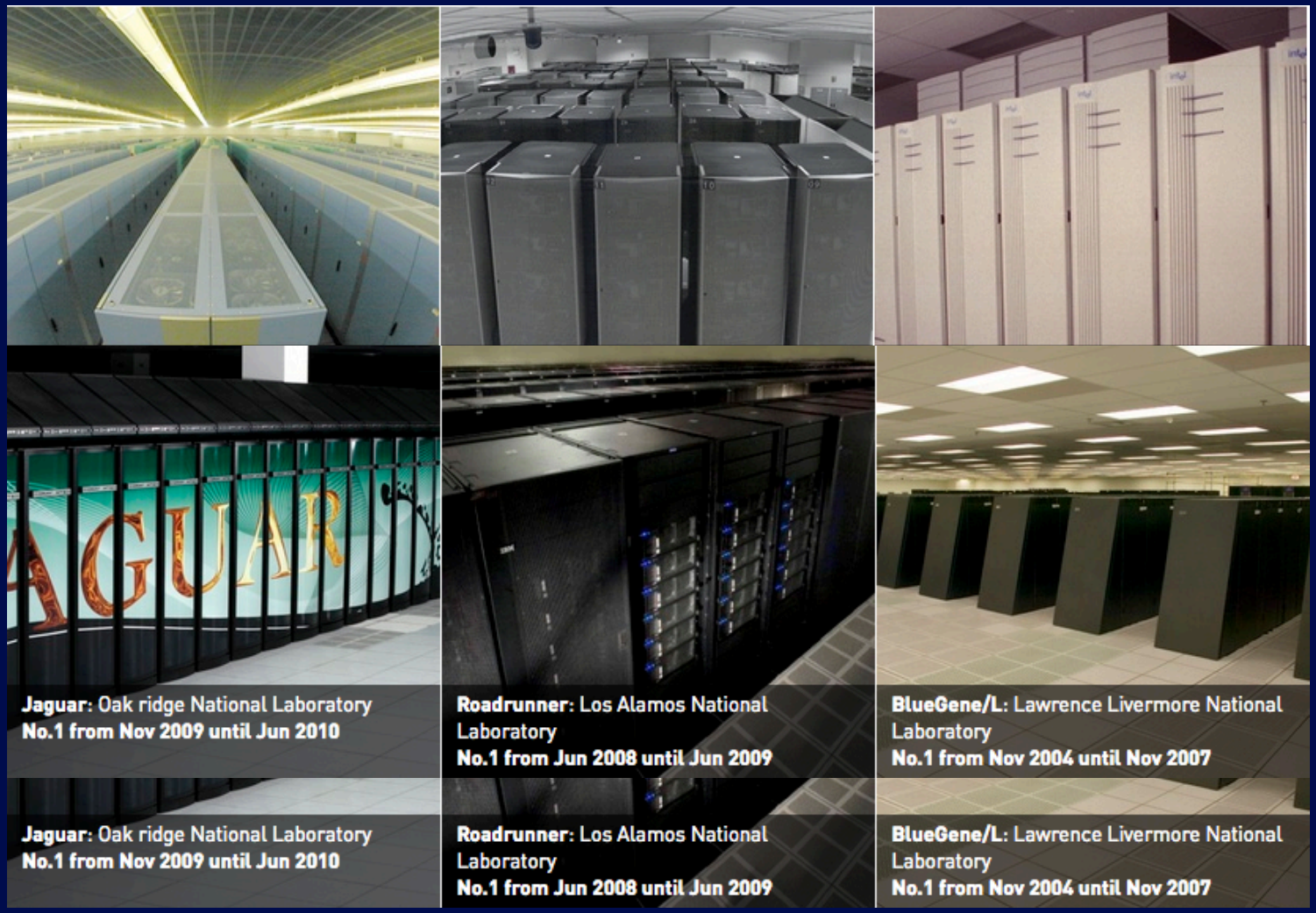
CP-PACS: University of Tsukuba
No.1 in Nov 1996

Hitachi SR2201: University of Tokyo
No.1 in Jun 1996

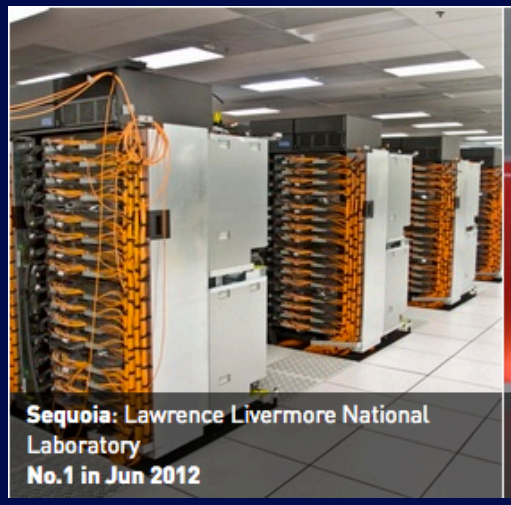


Intel XP/S 140 Paragon: Sandia National Labs
No.1 in Jun 1994

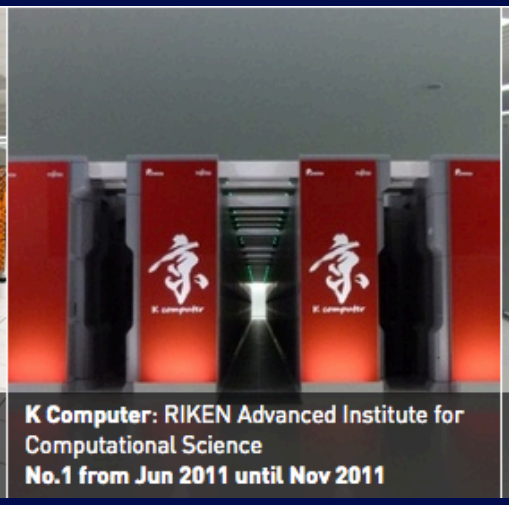
Top500 over the years…



Top500 over the years…



Sequoia: Lawrence Livermore National Laboratory
No.1 in Jun 2012

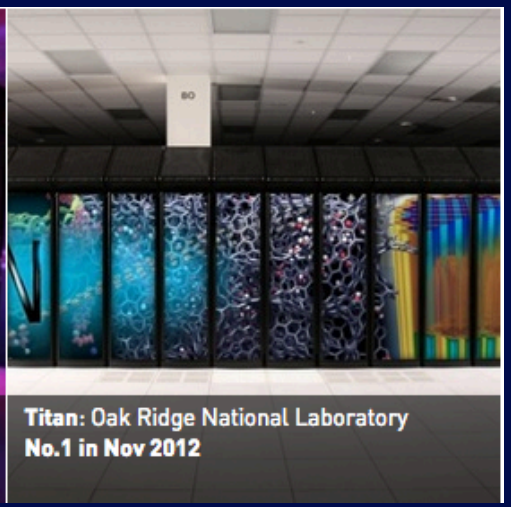


K Computer: RIKEN Advanced Institute for Computational Science
No.1 from Jun 2011 until Nov 2011

Tianhe-1A: National Supercomputing Center in Tianjin
No.1 in Nov 2010

Sunway TaihuLight: National Supercomputing Center in Wuxi
No.1 from Jun 2016 until Jun 2017

Tianhe-2 (MilkyWay-2) : National University of Defense Technology
No.1 from Jun 2013 until Nov 2015



Titan: Oak Ridge National Laboratory
No.1 in Nov 2012

China's Sunway TaihuLight (10M cores!)

www.top500.org

TOP 10 Sites for June 2017

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
2	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P , NUDT National Super Computer Center in Guangzhou China	3,120,000	33,862.7	54,902.4	17,808
3	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	361,760	19,590.0	25,326.3	2,272
4	**Titan** - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x , Cray Inc. DOE/SC/Oak Ridge National Laboratory United States	560,640	17,590.0	27,112.5	8,209
5	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom , IBM DOE/NNSA/LLNL United States	1,572,864	17,173.2	20,132.7	7,890

1.44W/core → 375W/socket

Compute Power:
17 MW Generator, only $7M

New 17100kW Wärtsilä HFO Generator

Price (USD):	$6,825,000.00
Item#:	USP008718
Wattage:	17,100 kW
Make/Model:	Wärtsilä 18V50DF
Year of Mfr:	2016
Fuel Type:	HFO
Frequency:	60 Hz

Additional Info:

TRI FUEL - Diesel, HFO, and Natural Gas !
47.2% RATED ELECTRICAL EFFICIENCY !!
7616 KJ/KWHR HEAT RATE !!
See Specs attached...
NEW 2016 Surplus Engines
6 in stock !!

www.top500.org

TOP 10 Sites for June 2017

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
2	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P , NUDT National Super Computer Center in Guangzhou China	3,120,000	33,862.7	54,902.4	17,808
3	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	361,760	19,590.0	25,326.3	2,272
4	**Titan** - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x , Cray Inc. DOE/SC/Oak Ridge National Laboratory United States	560,640	17,590.0	27,112.5	8,209
5	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom , IBM DOE/NNSA/LLNL United States	1,572,864	17,173.2	20,132.7	7,890

Top500 CPUs + Accelerators

1. Sunway SW26010 (1.45GHz, 260 cores/socket)
2. Intel Xeon E5-2692 (2.2GHz, 12 cores/socket)
   - Accelerator: Intel Xeon Phi 31S1P
3. Intel Xeon E5-2690v3 (2.6GHz, 12 cores/socket)
   - Accelerator: Nvidia Pascal P100
4. AMD Operton 6274 (2.2GHz, 16 cores/socket)
   - Accelerator: Nvidia Tesla (Kepler) K20x
5. IBM Power BlueGene/Q (1.6GHz, 16 cores/socket)

Top500 CPUs + Accelerators

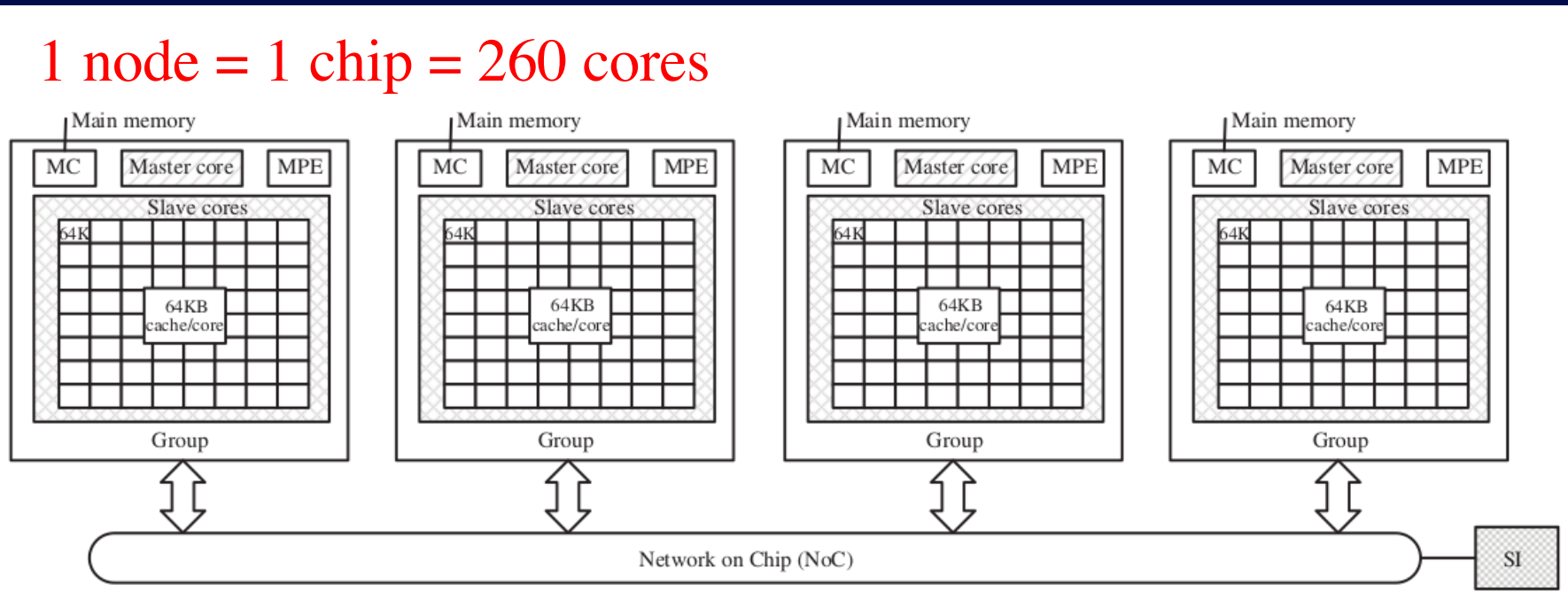
- CPUs
  - **Sunway SW26010** **(internal accelerator)**
  - **IBM BlueGene/Q** **(internal accelerator)**
  - Intel Xeon E5 (no accelerator)

- Accelerators
  - **Nvidia Pascal P100**
  - **Intel Xeon Phi 31S1P**

Sunway SW26010

http://www.netlib.org/utk/people/JackDongarra/PAPERS/sunway-report-2016.pdf

- 1 node (chip) = 260 cores
- 1 supernode (box) = 256 nodes = 66,560 cores
- 1 cabinet (rack) = 4 supernodes = 266,240 cores
- 1 system (room) = 40 cabinets = 10,649,600 cores



Sunway SW26010

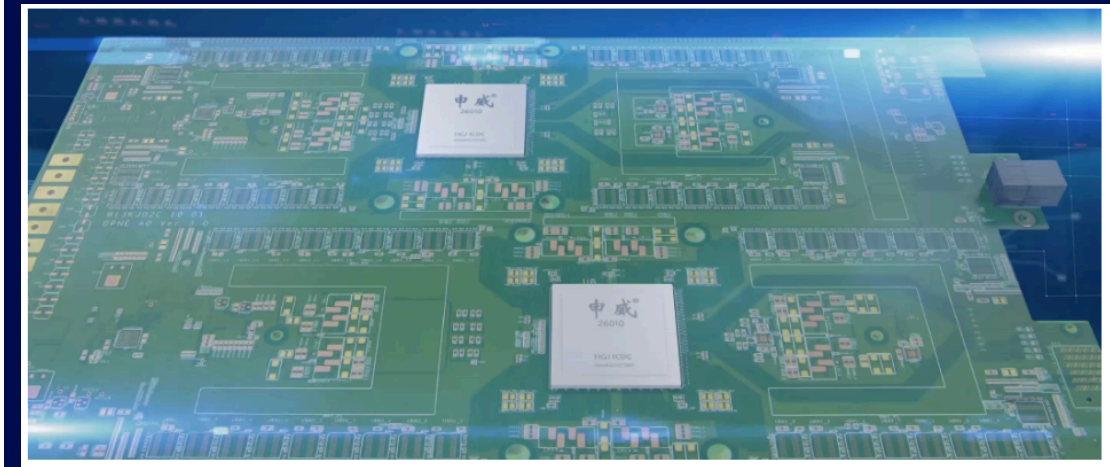
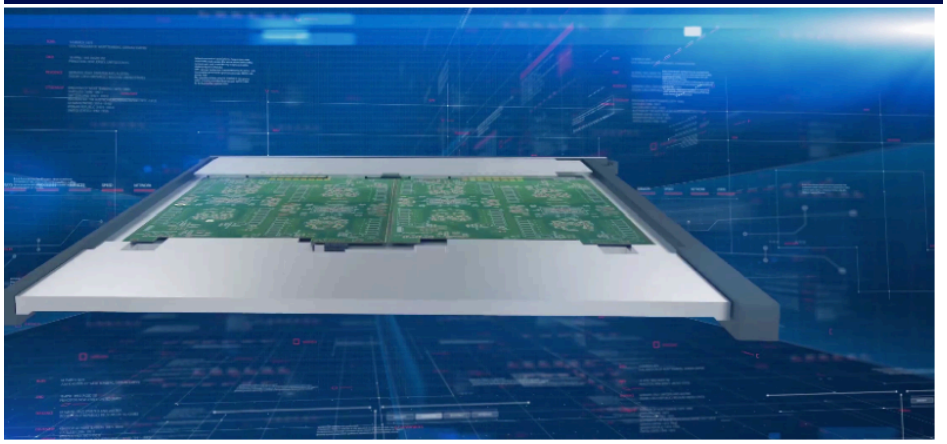


Figure 6: Two nodes on a card.



Four cards on a board, two up and two down (on the other side).

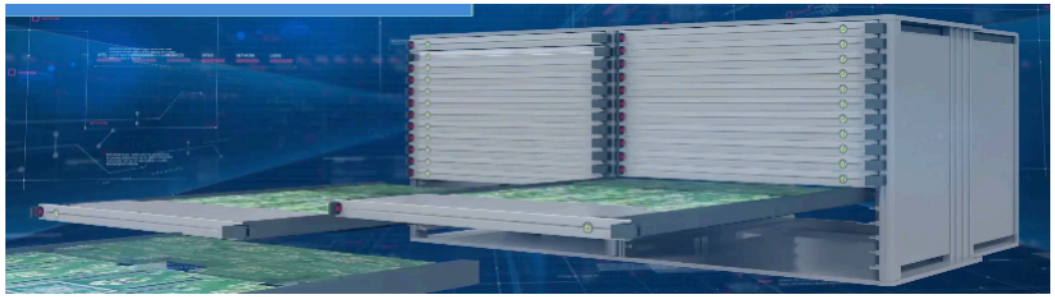


Figure 8: A Supernode composed of 32 boards or 256 nodes.

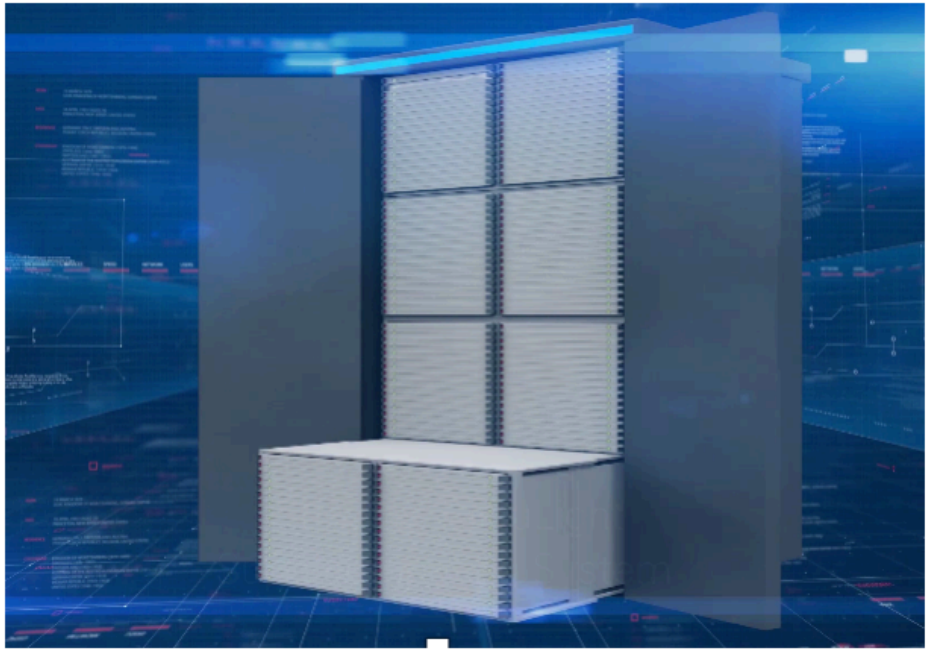
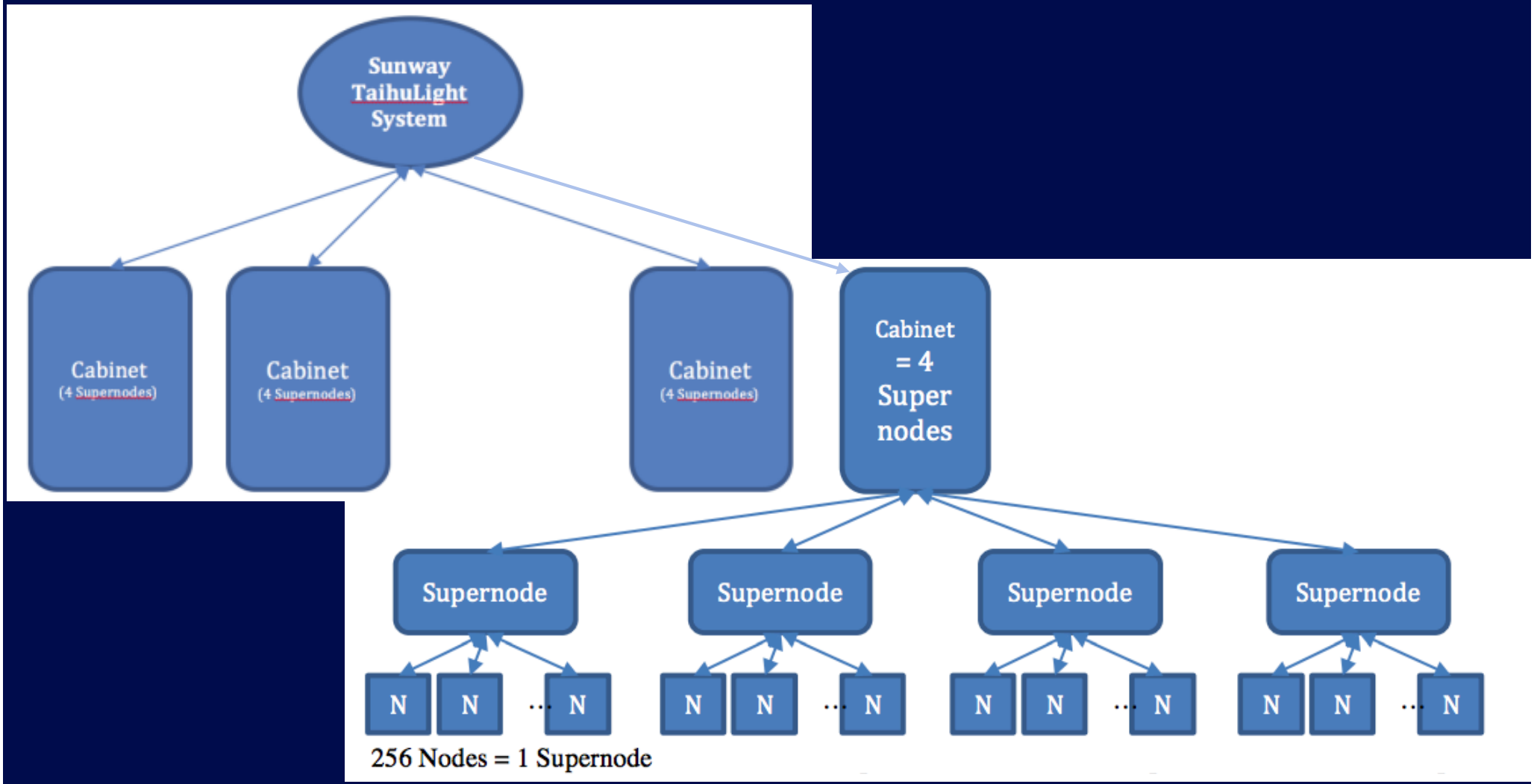


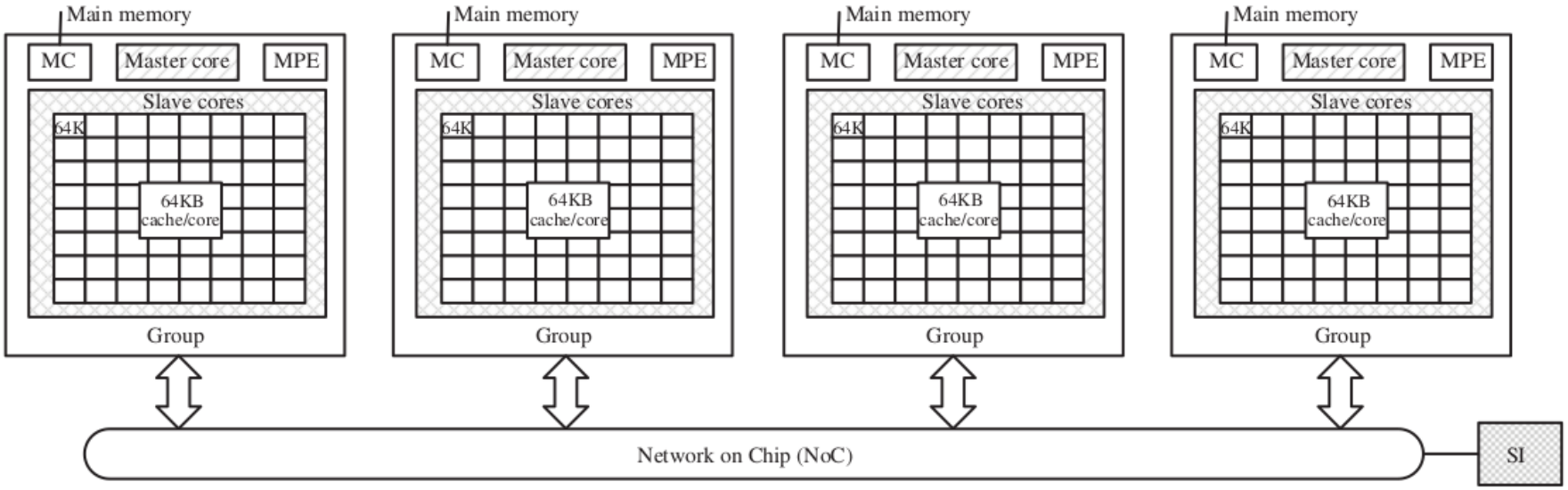
Figure 9: A cabinet composed of 4 supernodes or 1024 nodes.

Sunway SW26010



Sunway SW26010 CPU Core & Chip

1 node = 1 chip = 260 cores



- Per chip
  - 4 x 8GB DDR3 -2133 (1.3PB full system)
  - 136.5 GB/s bandwidth

IBM BlueGene/Q Compute (BQC)

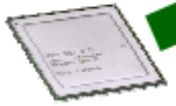
http://www.training.prace-ri.eu/uploads/tx_pracetmo/BG-Q-_Vezolle.pdf

http://www.redbooks.ibm.com/redbooks/pdfs/sg247872.pdf

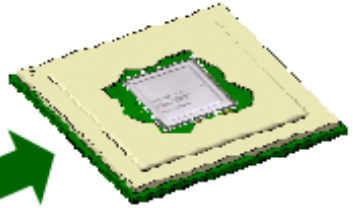
- 1 compute card (chip) = 16 cores
- 1 node card (box) = 32 compute cards = 512 cores
- 1 midplane (box) = 16 node cards = 8192 cores
- 1 cabinet (rack) = 2 midplanes = 16,384 cores
  - Compare to Sunway, with 266,240 cores per cabinet
- 1 system (room) = 96 cabinets = 1,572,864 cores
  - Compare to Sunway, with 40 cabinets @ 10M cores
- 1.6PB RAM (Sunway: 1.3PB)

IBM BlueGene/Q Compute (BQC)

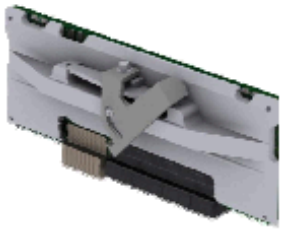
1. Chip:
16 λP cores



2. Single Chip Module



3. Compute card:
One chip module,
16 GB DDR3 Memory,



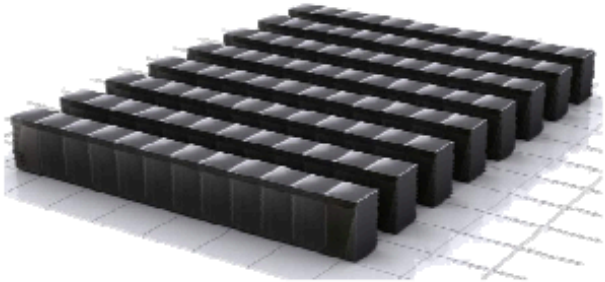
4. Node Card:
32 Compute Cards,
Optical Modules, Link Chips, Torus

5a. Midplane:
16 Node Cards

5b. IO drawer:
8 IO cards w/16 GB
8 PCIe Gen2 x8 slots

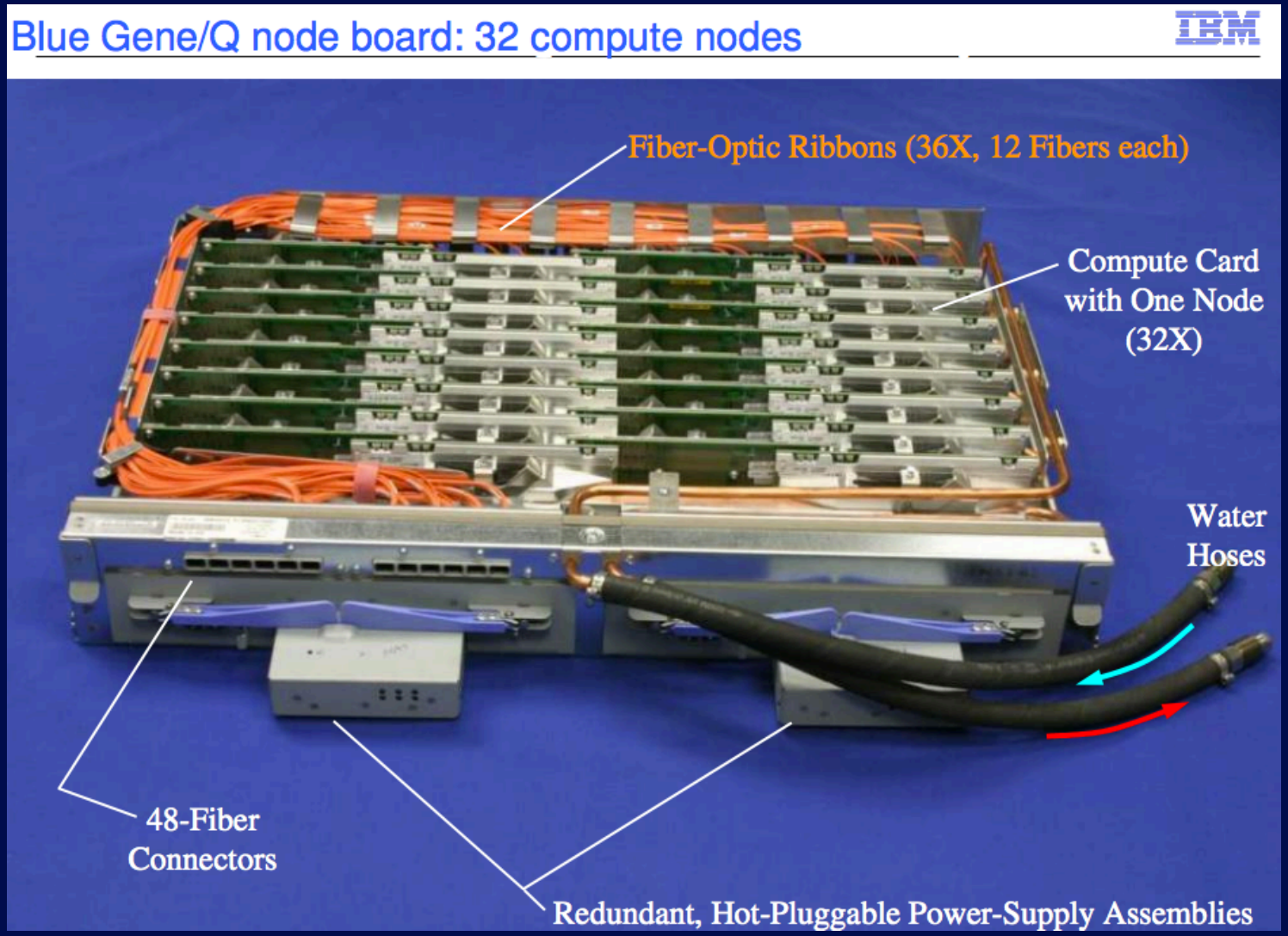
6. Rack: 2 Midplanes

7. System:
20PF/s



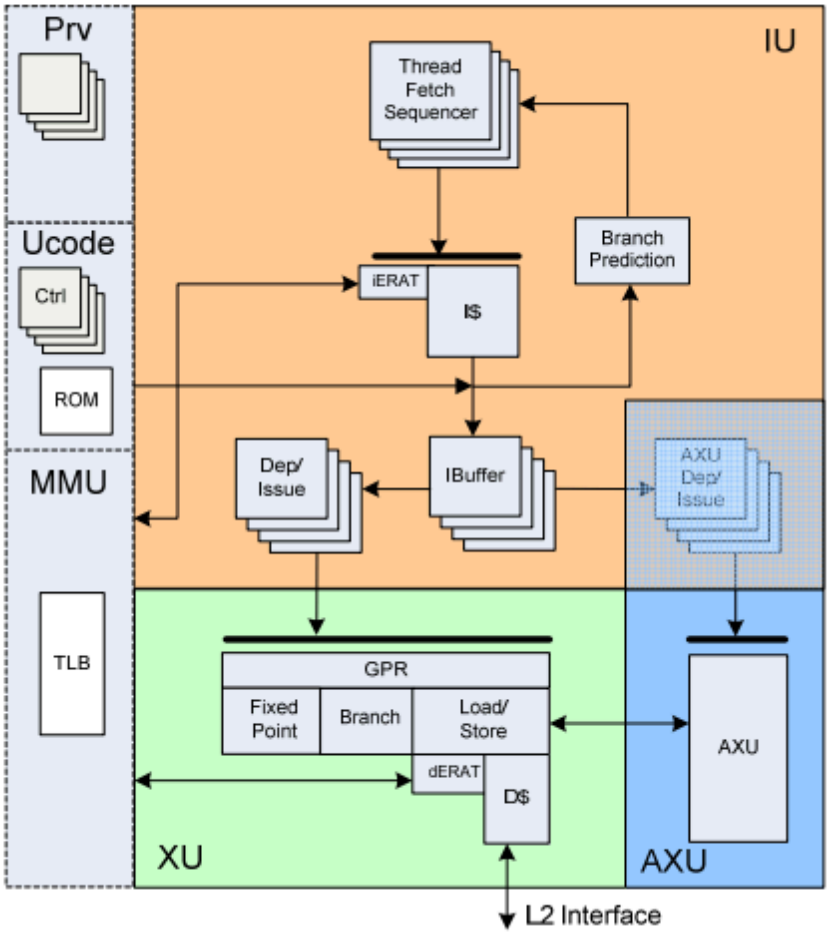
- **Sustained single node perf: 10x P, 20x L**
- **MF/Watt: (6x) P, (10x) L (~2GF/W target)**

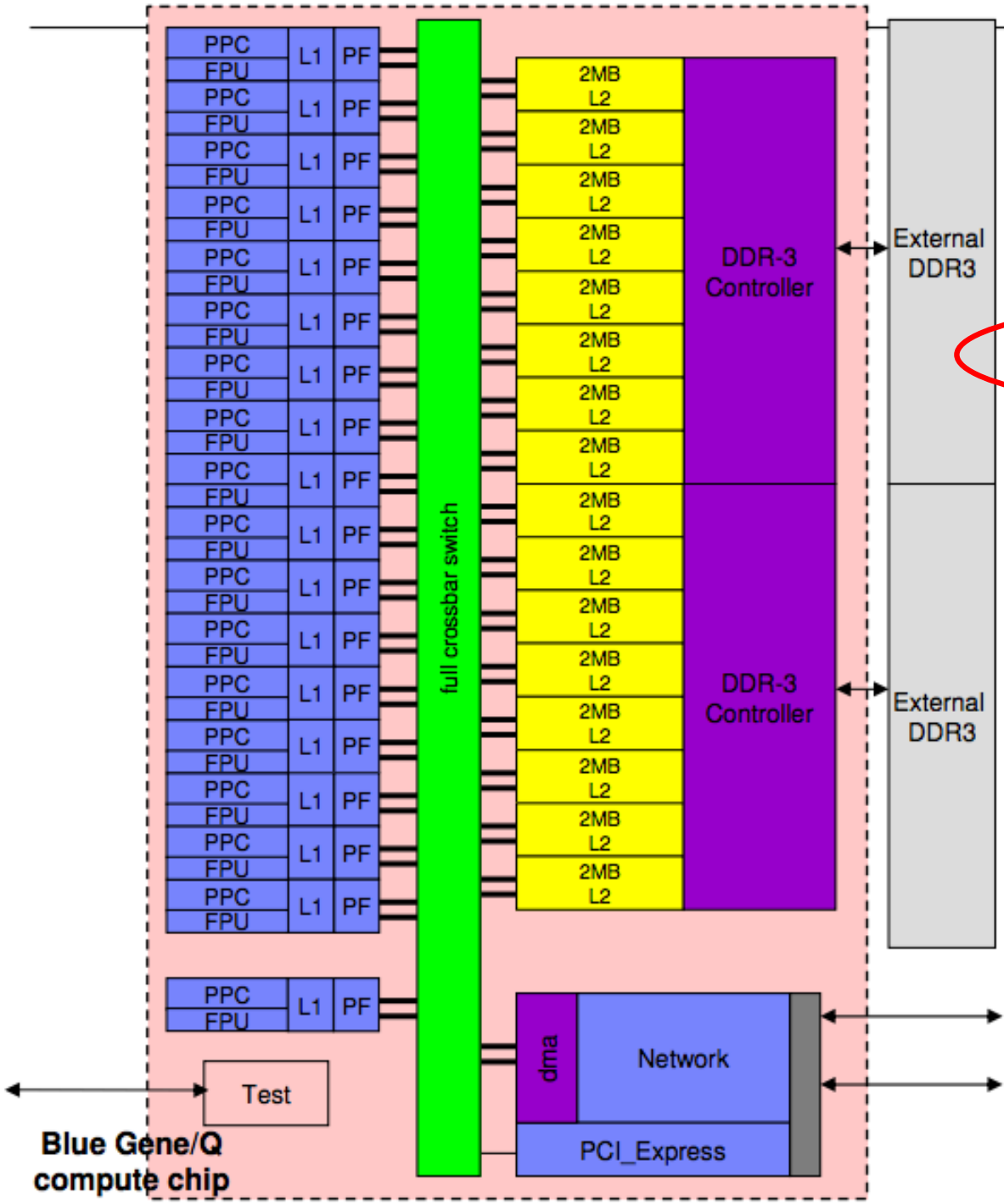
IBM BlueGene/Q Node Card



IBM BlueGene/Q Processor (A2 core)

- Simple core, designed for excellent power efficiency and small footprint.
- Embedded 64 b PowerPC compliant
- **4 SMT threads typically get a high level of utilization on shared resources.**
- Design point is 1.6 GHz @ 0.74V.
- AXU port allows for unique BGQ style floating point
- **One AXU (FPU) and one other instruction issue per cycle**
- **In-order execution**





Blue Gene/Q chip architecture

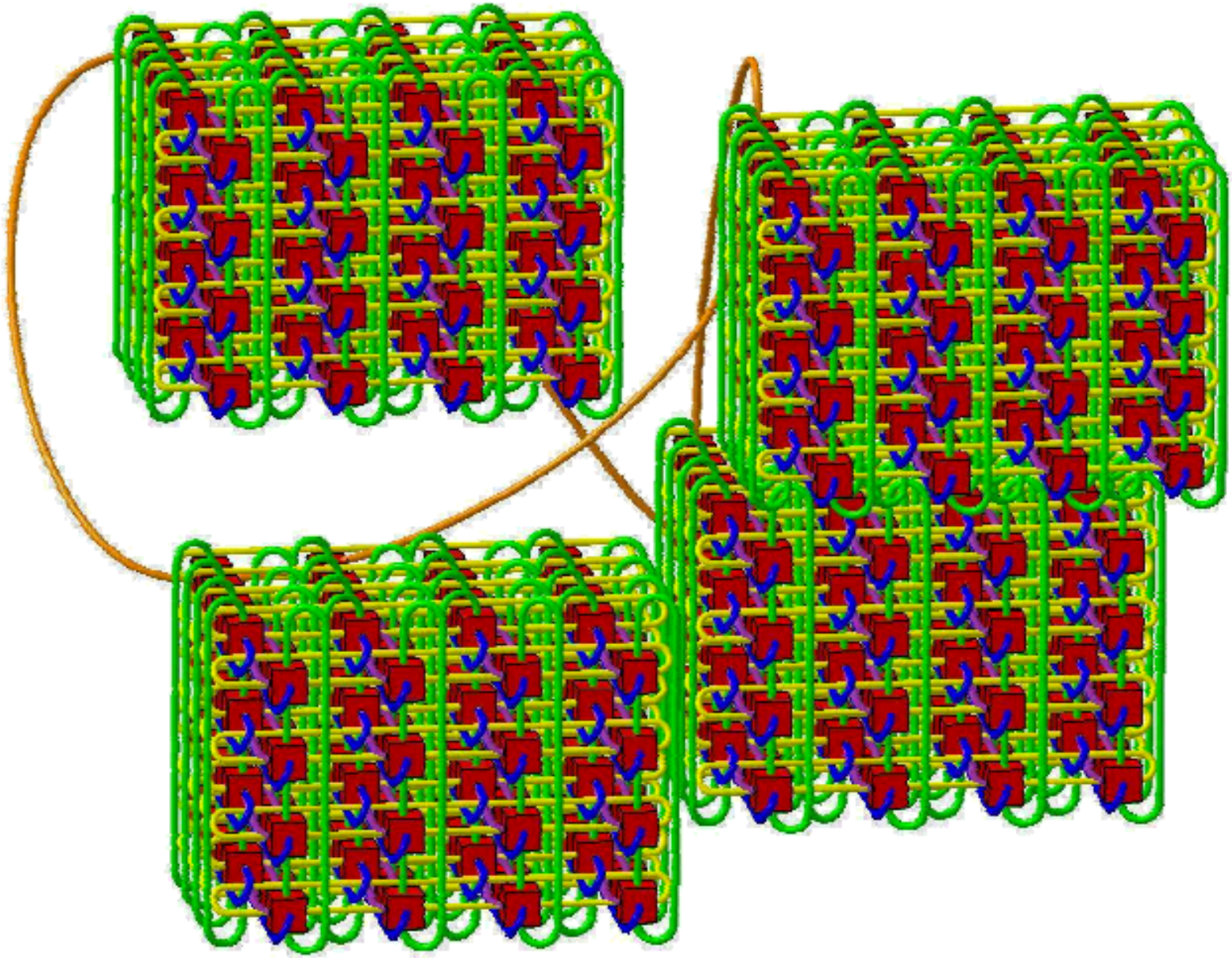
- 16+1 core SMP
  - Each core 4-way hardware threaded
- Transactional memory and thread level speculation
- Quad floating point unit on each core
  - 204.8 GF peak node
- Frequency target of 1.6 GHz
- 563 GB/s bisection bandwidth to shared L2
  - (Blue Gene/L at LLNL has 700 GB/s for system
- 32 MB shared L2 cache
- 42.6 GB/s DDR3 bandwidth (1.333 GHz DDR3)
  - (2 channels each with chip kill protection)
- 10 intra-rack interprocessor links each at 2.0GB/s
- one I/O link at 2.0 GB/s
- 8-16 GB memory/node
- ~60 watts max DD1 chip power

2 GB/s I/O link (to I/O subsystem)

10*2GB/s intra-rack & inter-rack (5-D torus)

note: chip I/O shares function with PCI_Express

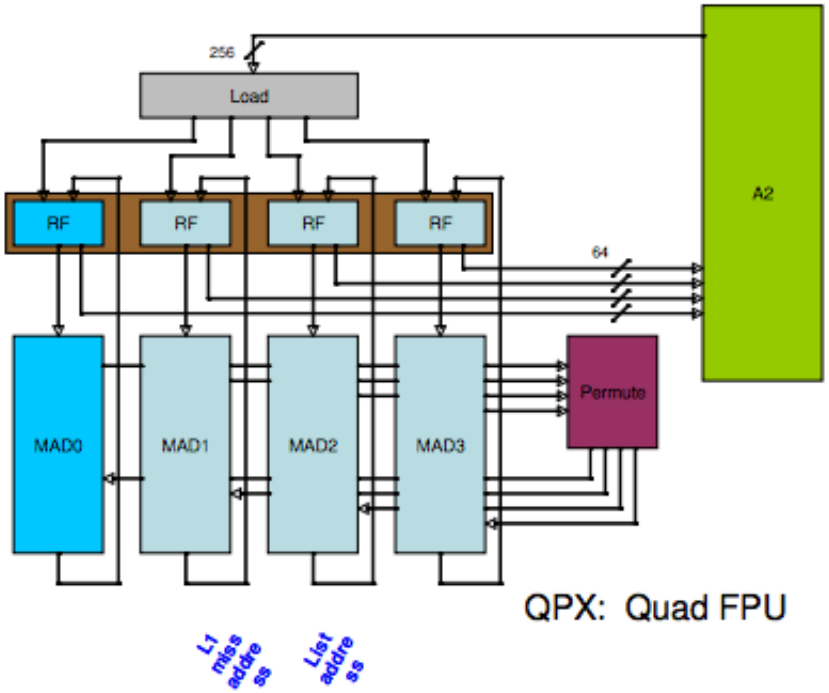
Blue Gene/Q Network: 5D Torus



IBM BlueGene/Q Quad FPU

- **Quad FPU**
  - **4-wide double precision FPU SIMD**
  - **2-wide complex SIMD**
  - **Supports a multitude of alignments**
  - **Allows for higher single thread performance for some applications**
  - **4R/2W register file**

    32x256 bits per thread
  - **32B (256 bits) datapath to/from L1 cache**



QPX: Quad FPU

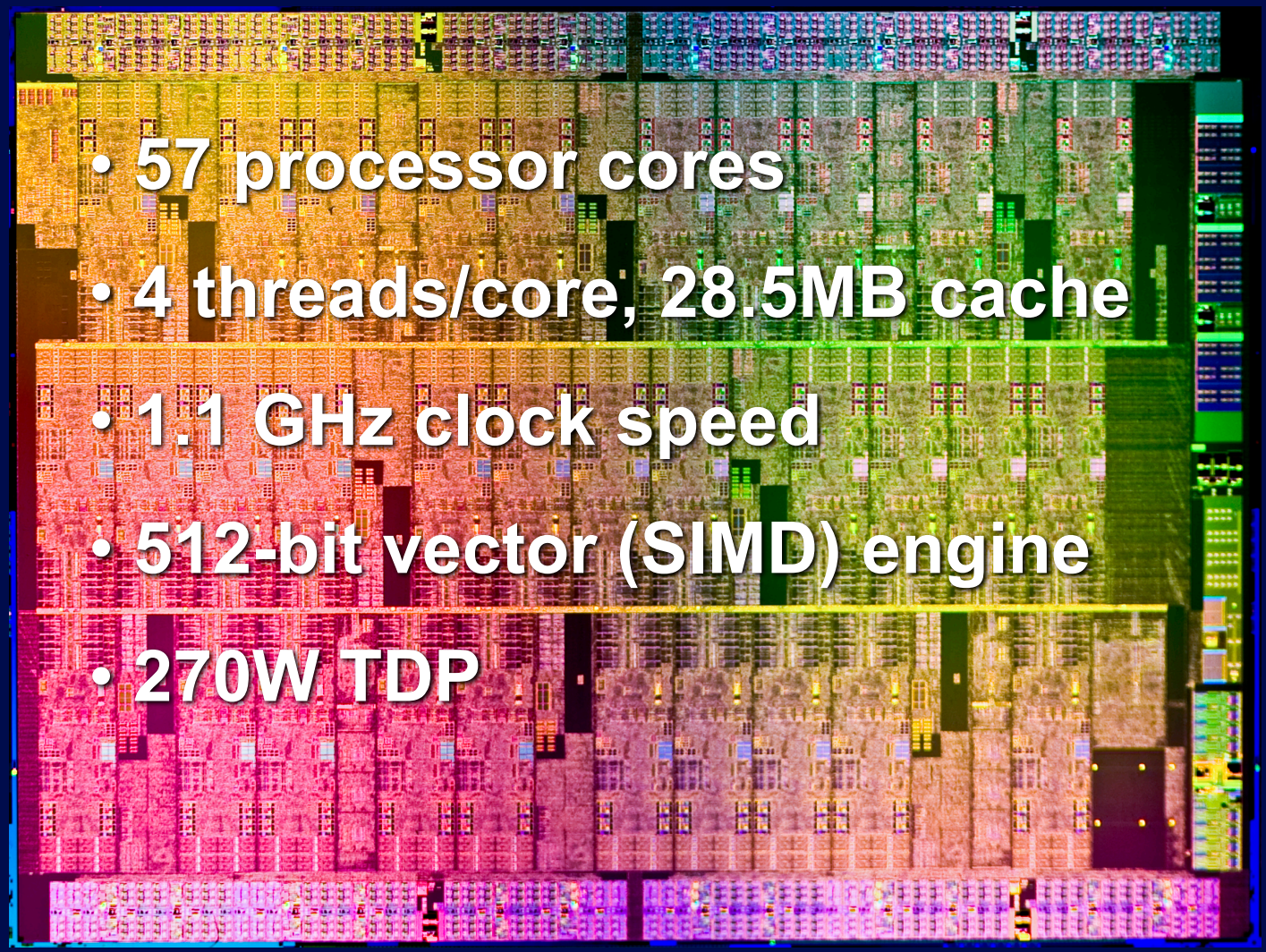
IBM BlueGene/Q: SIMD in Compiler

Auto-SIMDization Support

- Improving SIMD from BG/P/L to BG/Q
  - Moving from low level optimizer to high-level optimizer due to better loop optimization structure, and alignment analysis
- To enable SIMDization
  - Start to compile (assume appropriate –qarch=qp and –qtune=qp option):
    - -O3 (compile time, limited hot and SIMD optimizations)
  - Increase optimization Level
    - -O4 (compile time, limited scope analysis, SIMDization)
    - -O5 (link time, pointer analysis, whole-program analysis, and SIMD instruction)
  - -qhot=level=0 (enables SIMD by default)
- To disable SIMDization
  - Turn off SIMDization for the program
    - add -qhot=nosimd to the previous command line options
  - Turn off simdization for a particular loop
    - #pragma nosimd | !IBM* NOSIMD
- Tune your programs
  - Help the compiler with extra information (directives/pragmas)
  - Check the SIMD instruction generation in the object code listing (-qsource -qlist).
  - Use compiler feedback (-qreport -qhot) to guide you.

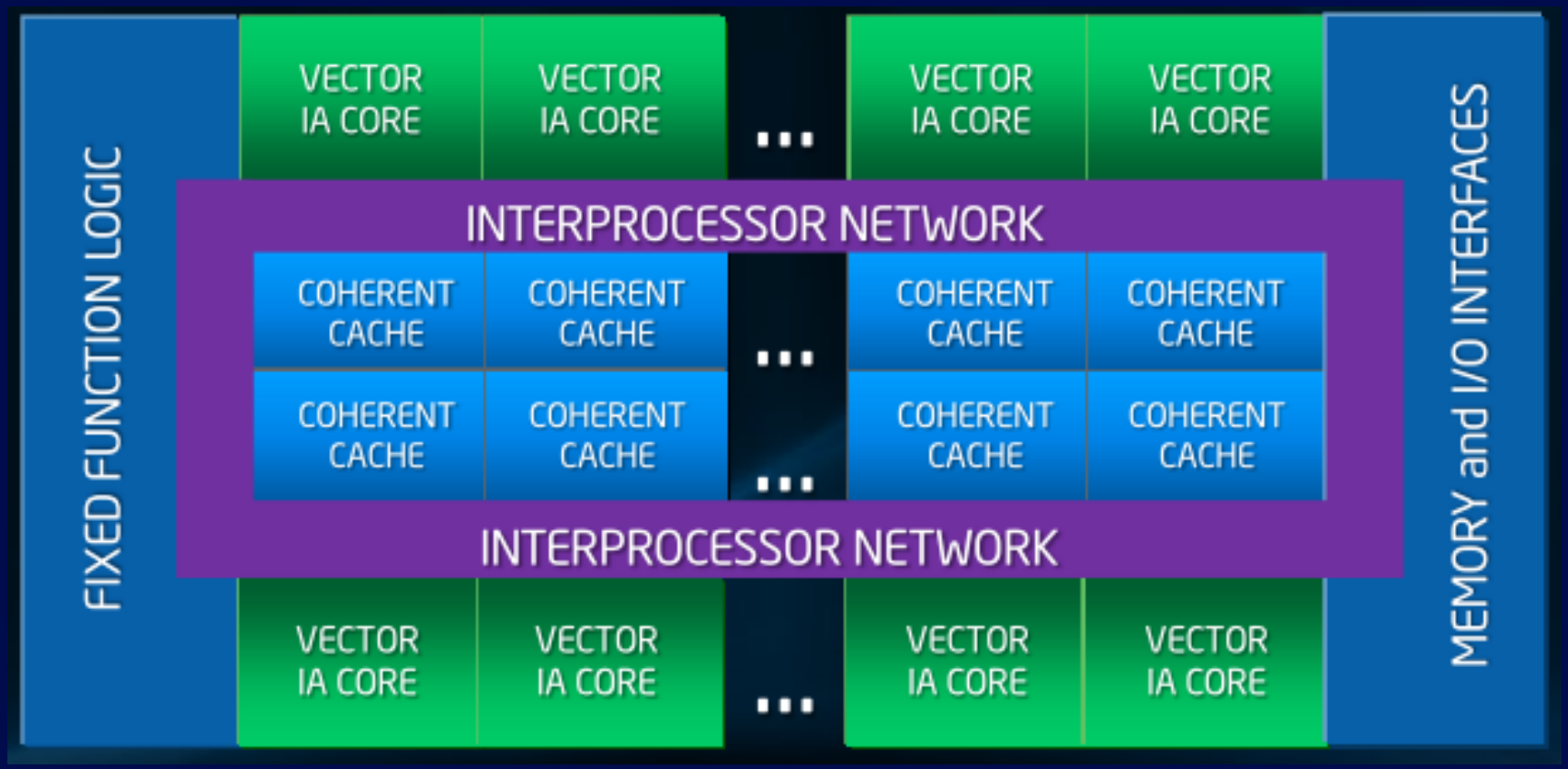
Intel: Xeon Phi (PCIe card)

Intel Xeon Phi 31S1P

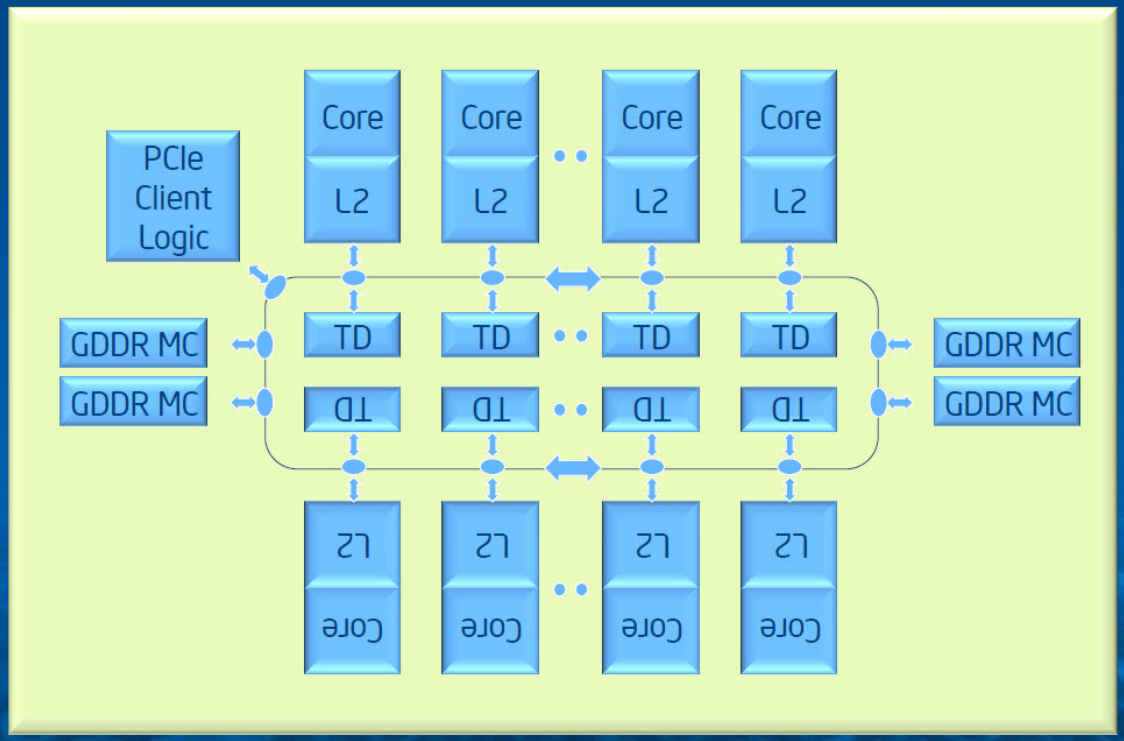


Intel Xeon Phi

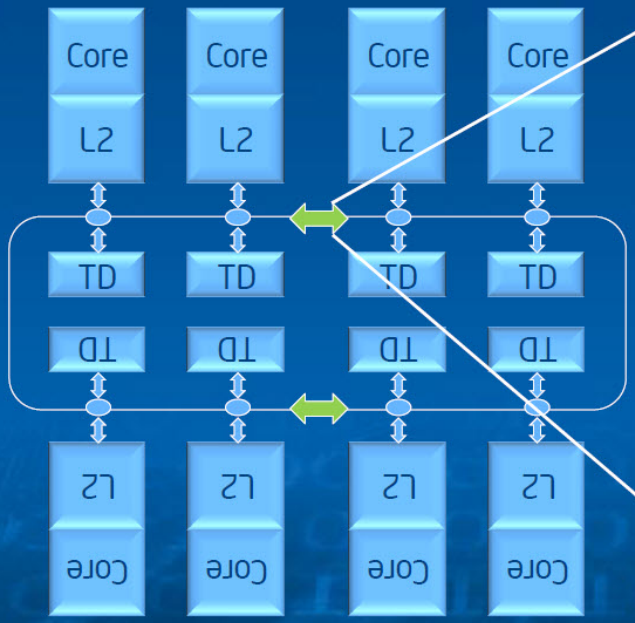
- up to 72 cores on a chip, 1.7GHz
- 1st generation “Knight’s Corner”
- 2nd generation “Knights Landing”

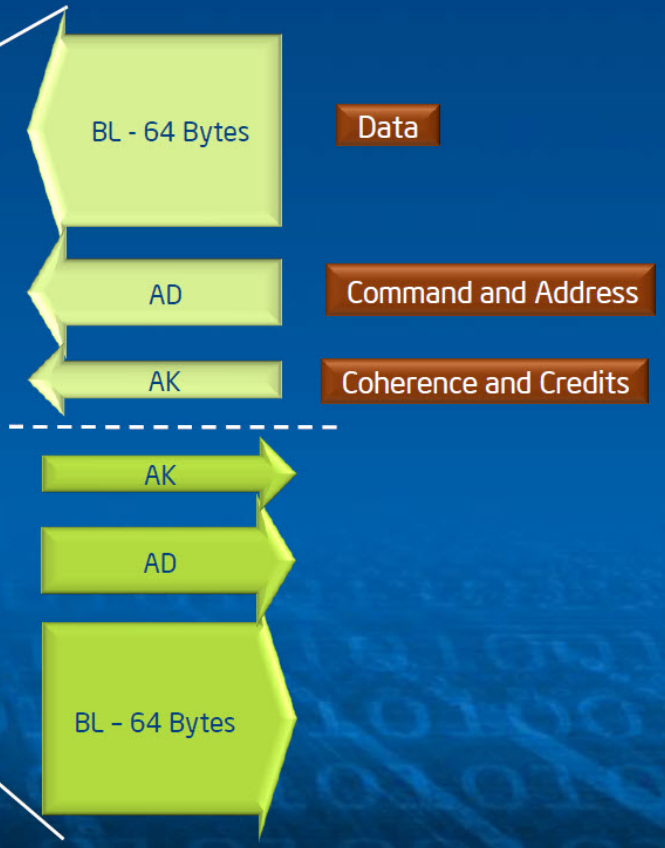


Knights Corner Micro-architecture

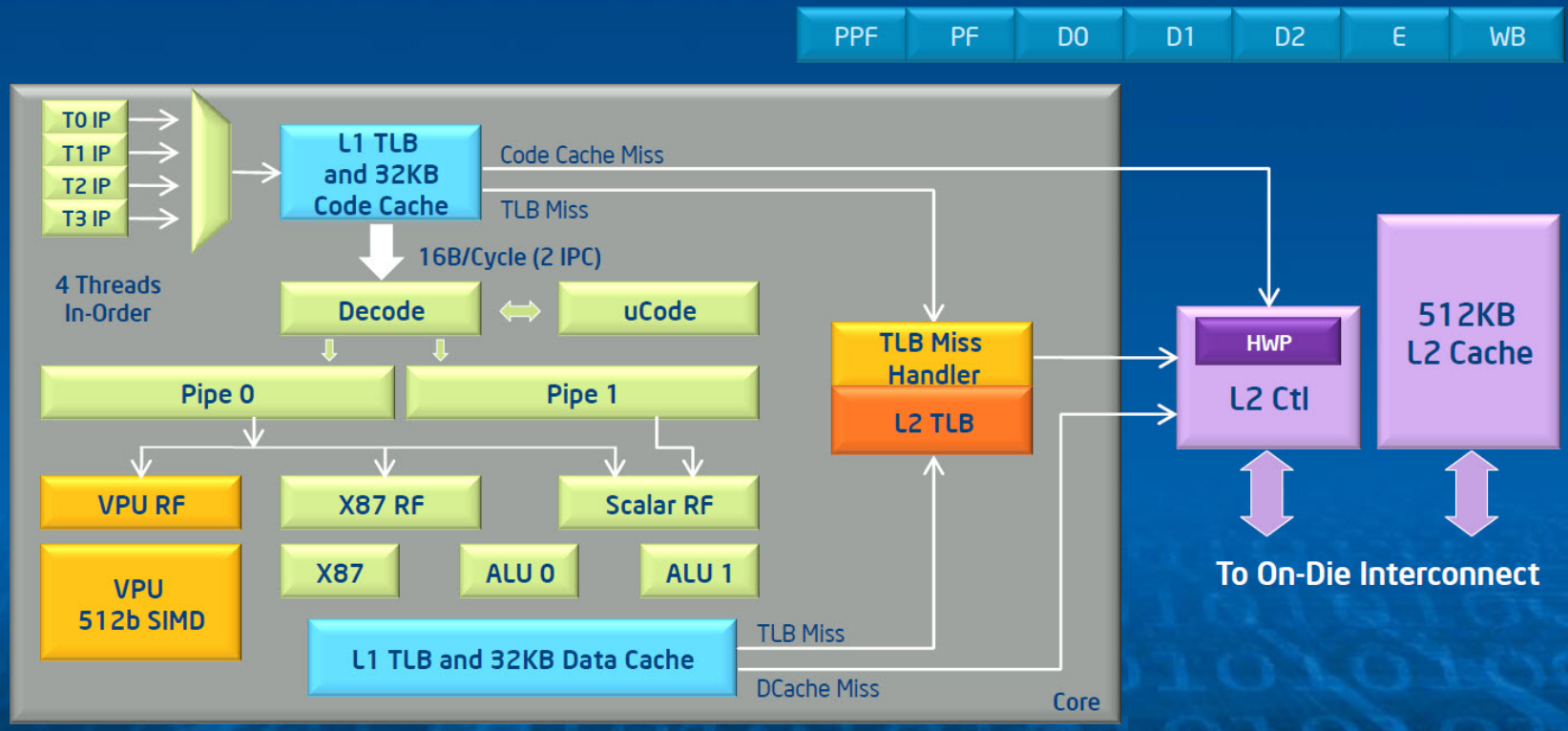


Interconnect



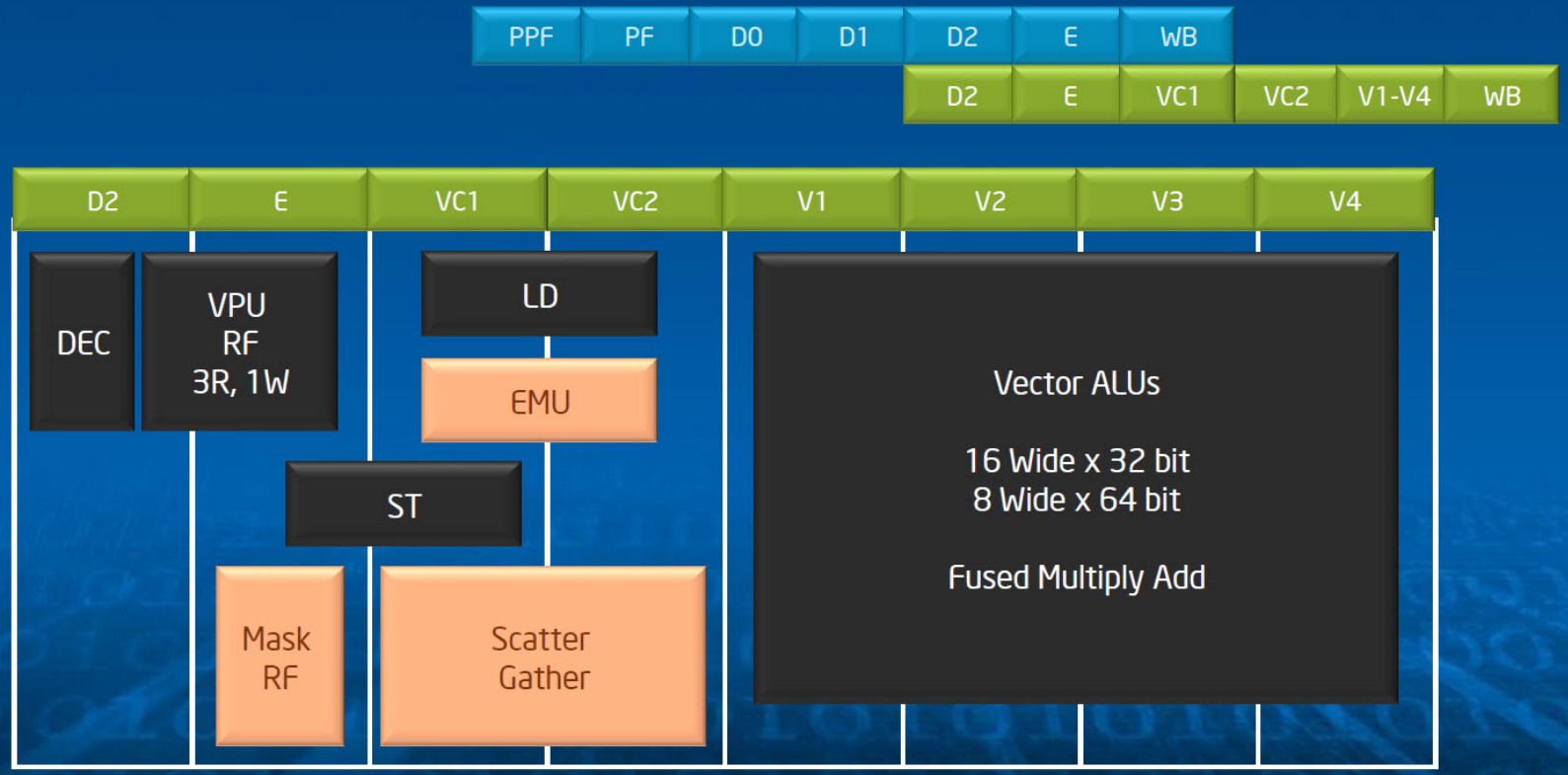


Knights Corner Core



X86 specific logic < 2% of core + L2 area

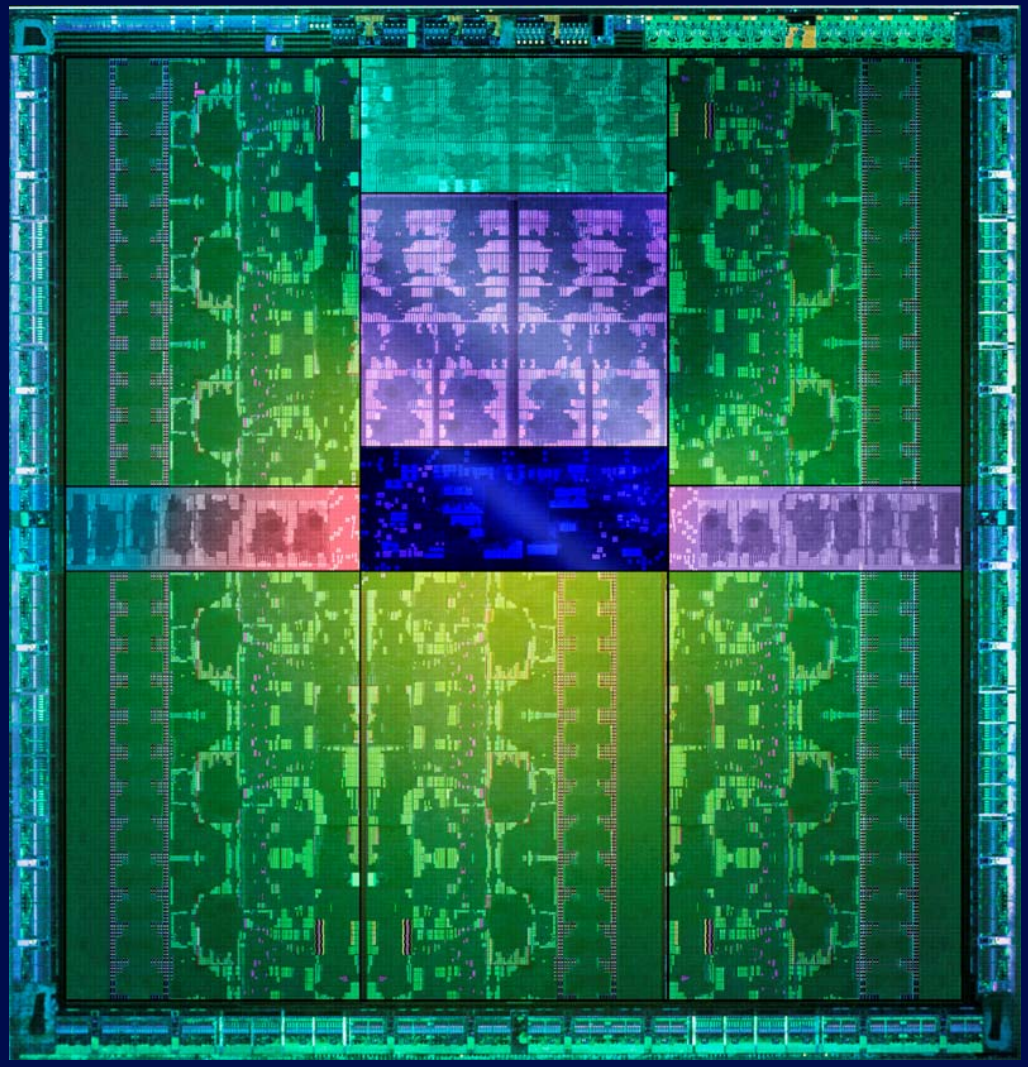
Vector Processing Unit



NVIDIA GPU Chip Architectures

- 2010 Fermi
- 2012 Kepler
- 2014 Maxwell
- 2016 Pascal
- 2017 Volta
- “Tesla” is name of card used for GPGPU compute acceleration, used since 2007

NVIDIA Tesla (Kepler) K20x (GK110)




SMX
Instruction Cache
Warp Scheduler
Dispatch
Register File (65,536 x 32-bit)
Core
DP Unit
LD/ST
SFU
Interconnect Network
64 KB Shared Memory / L1 Cache
48 KB Read-Only Data Cache
Tex
Memory Controller

NVIDIA – Kepler GK110

- 15 SMX “streaming multiprocessors”
  - 16x12=192 CUDA cores
    - (15x192 = 2880 total cores)
  - 64 double-precision units
  - 32 special function units (trancendentals)
  - 32 load/store units

Complications

- All those ALUs need to communicate!
  - Speed-of-interconnect < speed-of-light
  - Communication across chip is getting slower (RC effects)
    - See whiteboard

Overview of Lecture

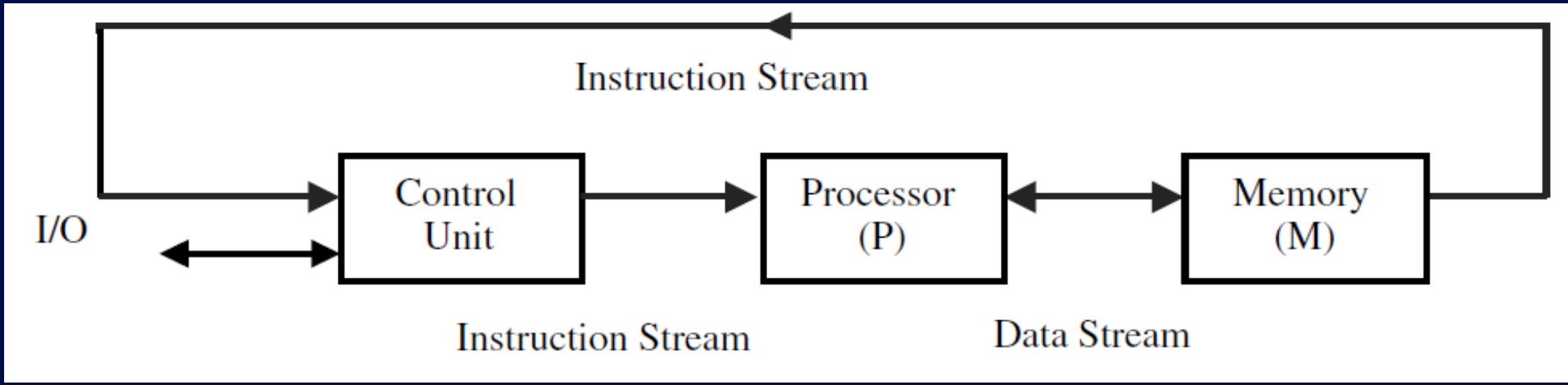
- Motivational Background
- Flynn's Taxonomy
- Types of Parallelism
- Limits: Dependence
- Limits: Amdahl

Classes of Parallel Architecture

- Flynn's Taxonomy
  - SISD
  - SIMD
  - MISD
  - MIMD
- New one?
  - SPMD

SISD

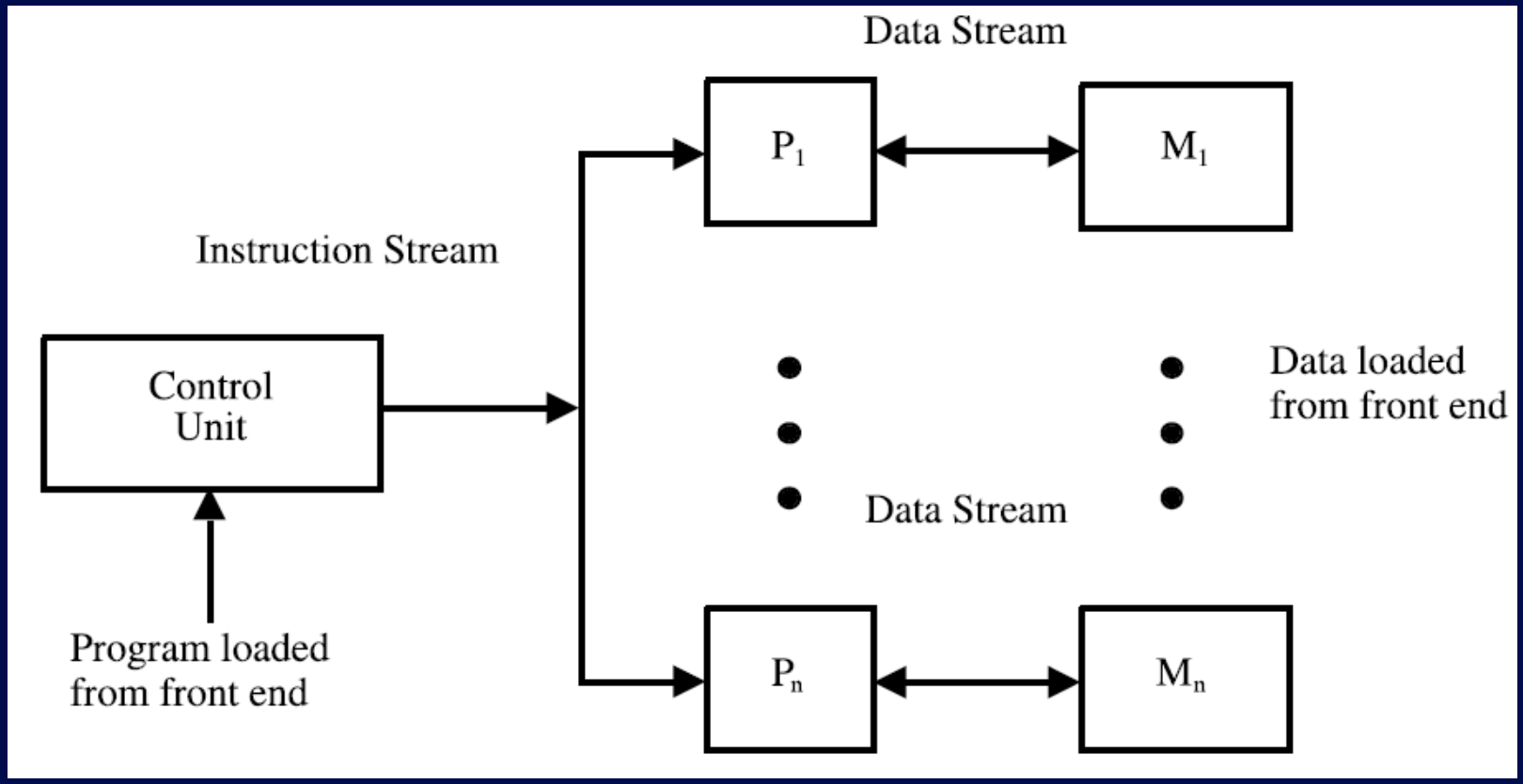
- Single Instruction – Single Data



Source: El-Rewini and Abd-El-Barr - Advanced Computer Architecture and Parallel Processing

SIMD

- Single Instruction – Multiple Data



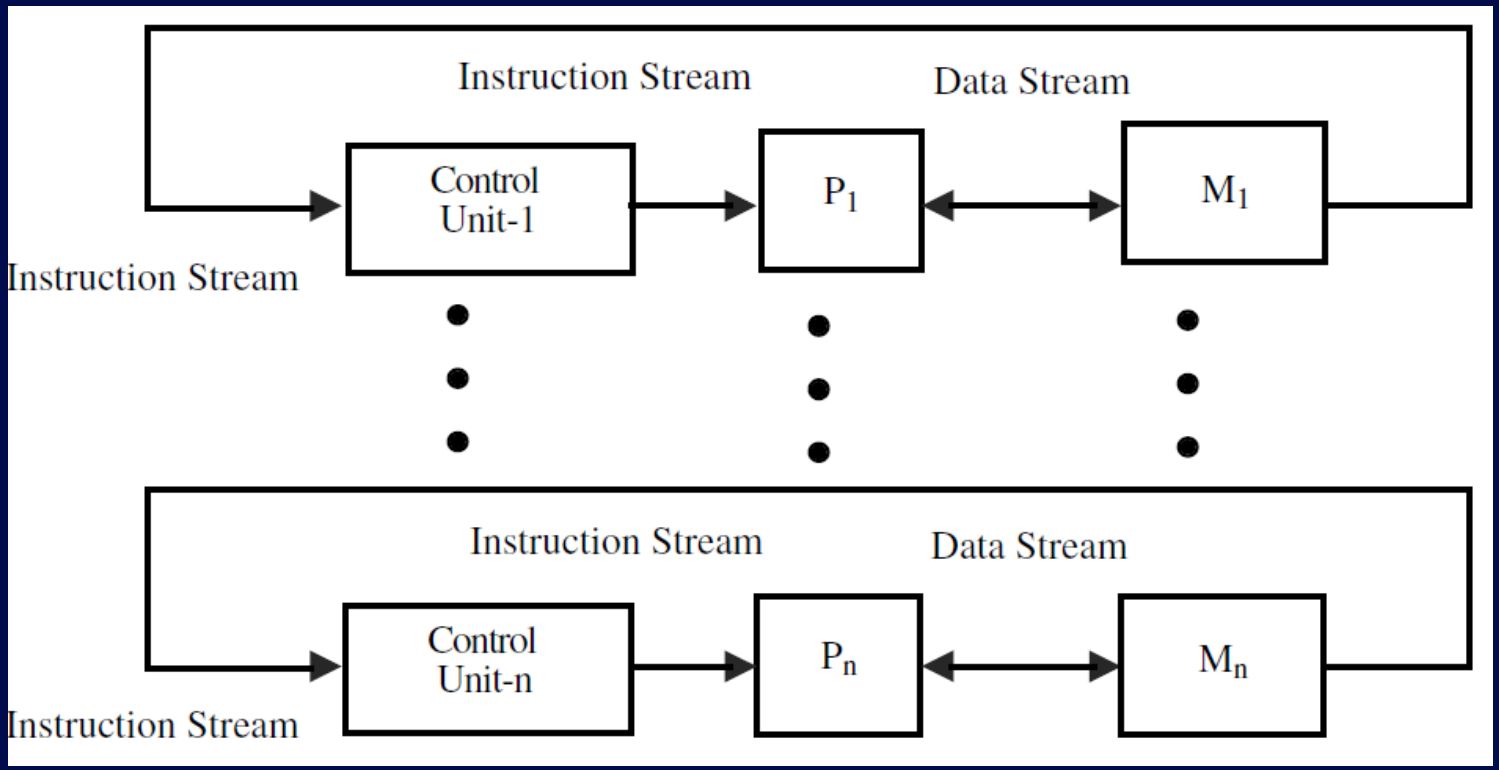
Source: El-Rewini and Abd-El-Barr - Advanced Computer Architecture and Parallel Processing

MISD

- Multiple Instruction – Single Data
- Exist in Concept, Not Implemented

MIMD

- Multiple Instruction – Multiple Data



Source: El-Rewini and Abd-El-Barr - Advanced Computer Architecture and Parallel Processing

MIMD

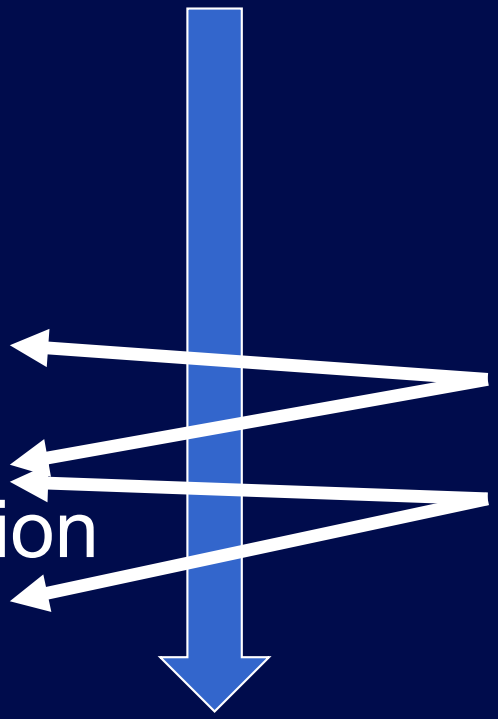
- Can either be *tightly-* or *loosely-coupled*
- Tightly-Coupled:
  - Shared Address Space
    - Symmetric Multiprocessors (SMPs)
    - Uniform Memory Access (UMA)
    - Nonuniform Memory Access (NUMA)
  - E.g. Intel i7 Quad Core
- Loosely-Coupled:
  - Disjoint Address Space
    - Distributed SISDs
    - Message Passing
  - E.g. Network Clusters

SPMD

- Single Program, Multiple Data (SPMD)
  - A special case of MIMD
- MIMD: compute nodes can run completely different programs
  - 3D physics on node 1
  - graphics rendering on node 2
- SPMD: compute nodes run identical programs
  - Free-running, out-of-sync programs
  - At any point in time, each node may run a different instruction

Parallelism Levels

- Task
- Thread
- Data
- Loop
- Instruction
- Bit



- Plus more!
- Transactional
- Memory
- Pipeline

Task-level Parallelism

- Usually at Function Level of a Program
  - Little or no communication between tasks
- Example:

  Given data A and B, find func1(A,B) and func2(A,B)
  - Two tasks
  - Assume two processors available
    - func1(A,B) on CPU1
    - func2(A,B) on CPU2
- Sometimes called “pipeline parallelism”
  - d = func1(A,B);
  - e = func2(d,C);

Thread-level Parallelism

- Similar to Task-level
  - May be finer grain
  - May be little or no communication
  - May be a lot of communication (NEW)

Data-level Parallelism

- Distribution of Data among Processors
- Example:

  Add two arrays of N x N elements
  - Divide data among P processors
  - Each processor adds N^2/P elements
- Exploited by GPU, vector, SIMD

Data-level Parallelism

- Examples

For(i=0 to 10) A[i] = B[i] + C[i];

For(i=0 to 10) A[i] = A[i] + B[i];

For(i=1 to 10) A[i] = A[i-1] + B[i];

Sometimes called Loop-level parallelism

Loop-level Parallelism

- Exploit Concurrency in Loops
- Possible examples…
  - For-loop to calculate dot product of array A and B
    - Is this really data parallelism?
  - Overlap loop iteration *i* with iteration *i+1* by starting next iteration as early as possible (but no earlier than any loop-carried dependence)
    - Is this "pipeline" parallelism?

Instruction-level Parallelism

- Machine Instruction Level
- Identify independent instructions within an instruction window
  - Superscalar: done every time at run-time by CPU
  - VLIW: done once at compile-time by compiler
  - JIT: done periodically at run-time by run-time system (eg, Java)
- Example:
  - ADD R1, R2, R3
  - LOAD R4, R2

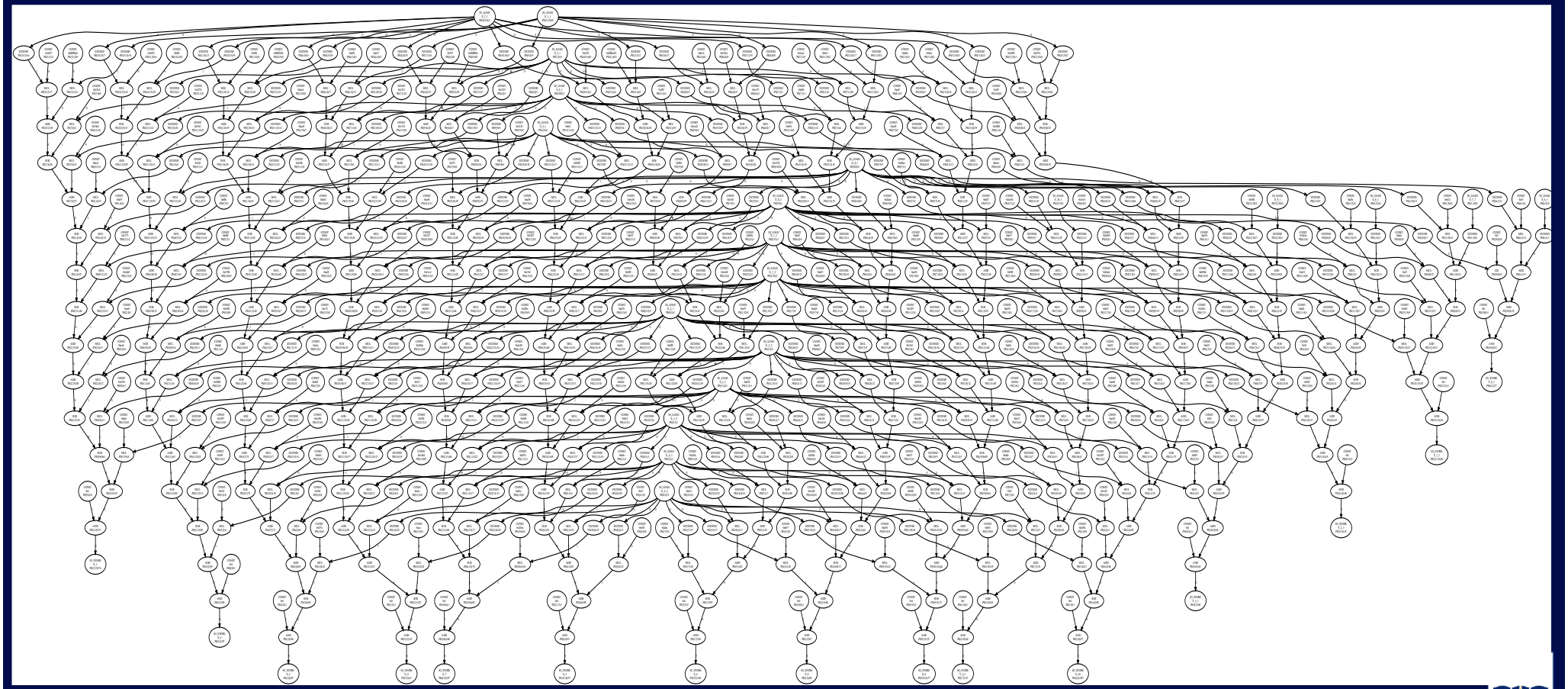
Bit-level Parallelism

- Example:
  - 32-bit bitwise AND
    - c = a & b;
  - 16-bit addition
    - c0 = a0 + b0; c1 = a1 + b1;
    - Two instructions on a 8-bit ALU
    - One instruction on a 16-bit ALU
    - But carry chains are really sequential… ?

Key Concept Surrounding Parallelism: **Dependence**

- Does result of current operation depend on result of other operation?
  - **Yes**
    - Other operation must be computed first
    - Sequential!
  - **No**
    - Operations can be computed in any order
    - Parallel!

Example Dependence Graph



Type of Dependencies

- RAR – read after read – no dependence
- RAW – read after write – true dependence
- WAR – write after read – false/anti dependence
- WAW – write after write – output dependence

RAR: no dependence

- Read after Read
  - No Dependency
- Example:
  - R2 <= R1 + 1
  - R3 <= R1 + 2

RAW: true dependence

- Read after Write
  - Producer/consumer relationship
  - Strong sequential dependence!
- Example:
  - R2 <= R1 + 1
  - R3 <= R2 + 2

WAR: false dependence

- Write after Read
  - Aka anti-dependence
- Example:
  - R2 <= R1 + 1
  - R1 <= R3 + 2
- Can 2nd op be done before 1st op?
  - Can WAR dependences be avoided?

Avoid WAR false dependence by Renaming

- Avoid by allocating new storage
  - Rename to use separate memory locations
- Example:
  - R2 <= R1 + 1
  - R1' <= R3 + 2
  - Rename all future ops to use R1'

WAW: output dependence

- Write after Write
- Example:
  - R2 <= R1 + 1
  - R2 <= R3 + 2
    - Can 2nd op be done before 1st op?
    - Can WAW dependences be avoided?

Avoid WAW output dependence

- Avoid by:
  - Optimizing away earlier computation?
    - ~~R2 <= R1 + 1~~
    - R2 <= R3 + 2
  - Renaming to use separate memory locations?
    - R2 <= R1 + 1
    - R2' <= R3 + 2
- What happens if this is output going to a printer?
  - Be careful of side effects!! Cannot always reorder/optimize

Type of Dependencies

- RAR – no dependence
- RAW – true dependence
- WAR – false/anti dependence
- WAW – output dependence

Both of these aka Name dependences
Can be resolved with renaming

The Ultimate Speed Limit

… beep, beep!

Amdahl's Law

Gene Amdahl: Famous computer systems architect at IBM in 60s and 70s.

Question:

- If you improve **part** of the system, how much faster does the **entire** system run?

Amdahl's Law gives us the speed limit!

- Given Enhancement E, define:

Speedup(E) = PerformanceAfter(E) / PerformanceBefore(E)

= ExecutionTimeBefore(E) / ExecutionTimeAfter(E)

Amdahl's Law

- More detail….
  - Enhancement E
  - results in a speedup of S
  - to only ***some fraction*** of the program F:

ExecutionTimeAfter(E) = [(1-F) + F/S] * ExecutionTimeBefore(E)

(derivation on next slide)

- Usually expressed as a speedup

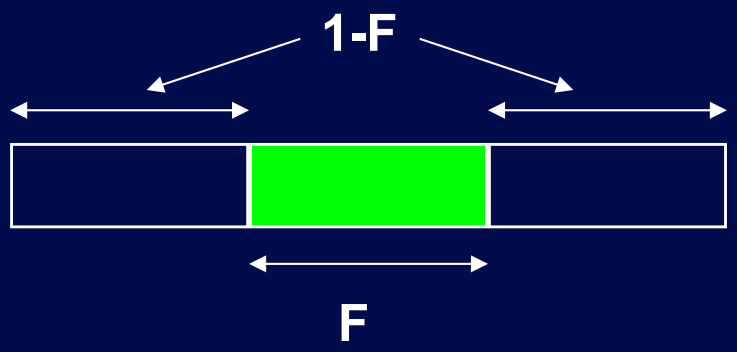
Speedup(E)

= ExecutionTimeBefore(E) / ExecutionTimeAfter(E)

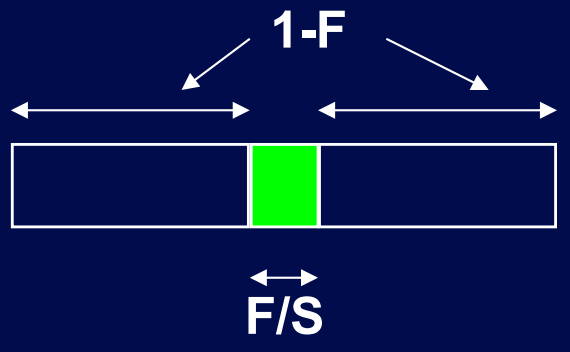
= 1 / [(1-F) + F/S]

Amdahl's Law Derivation

Before E:



After E:



(1-F) portion untouched:

ExecutionTimeBefore(E) = [(1-F) + F] = 1

F portion improved by S times, to F/S:

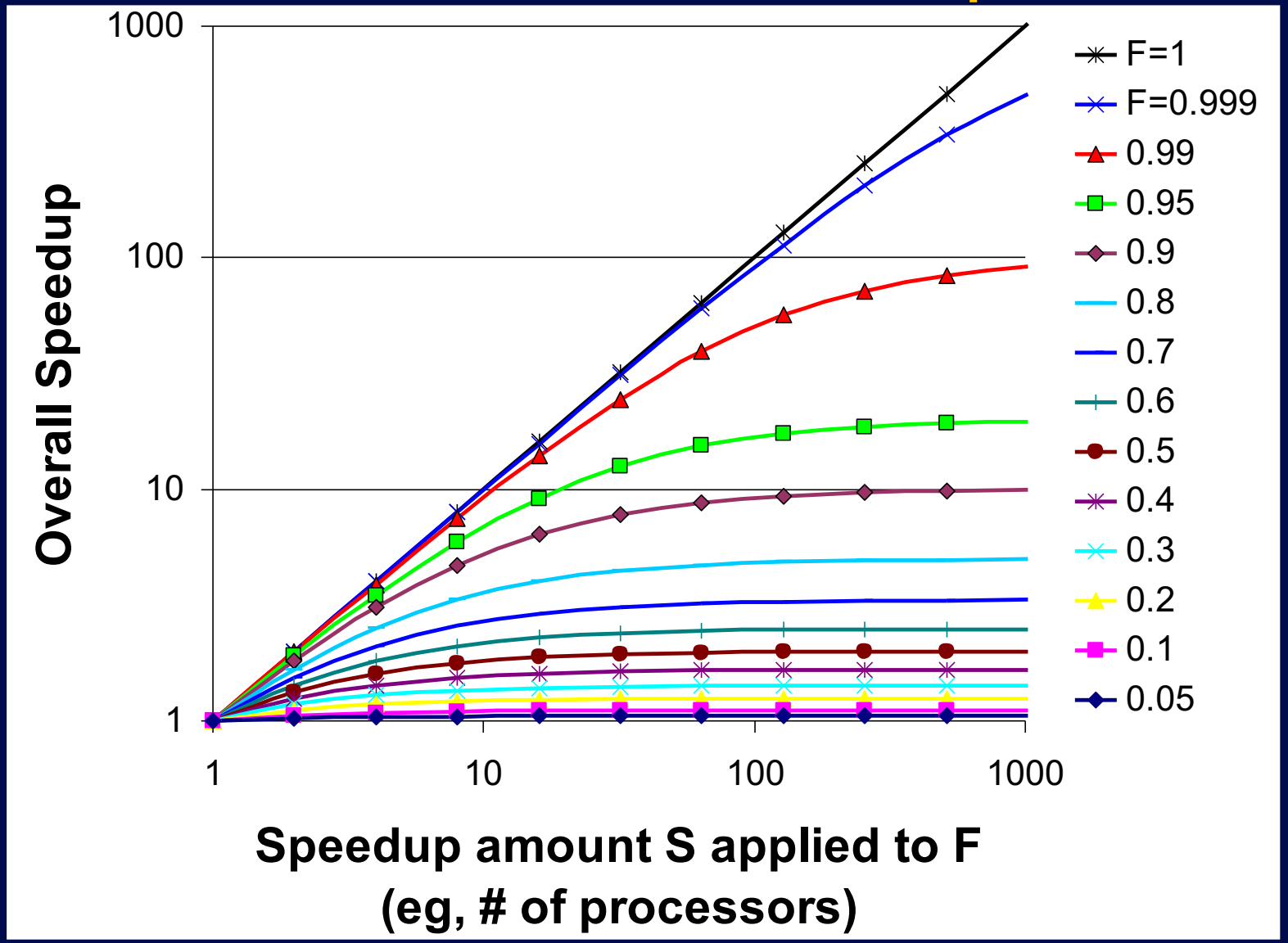
ExecutionTimeAfter(E) = [(1-F) + F/S]

Therefore:

$$\text{Speedup(E)} = \text{Before/After} = 1 \,/\, [(1-F) + F/S]$$

Lesson: **when speeding up a computer system, work on the part with the biggest F**

Amdahl's Law: Speed Limits!



F is portion of program that can be sped up.

Amdahl's Law Summary

- Amdahl's Law
  - Designer's Mantra: Make the common case fast
  - Applies to all engineering optimizations !!!
- Corollary
  - Rare cases don't matter
- Student's Corollary
  - On a test, do the easy stuff for the most marks first

Amdahl's Law Rebuttal: **Gustafson's Law**

- Does Amdahl always win?
  - According to Amdahl, the sequential part limits the maximum speedup
- Gustafson's Law
  - As the number of processors increases, you can scale the problem size
  - As the problem size grows, the sequential part should shrink (relative to the size of the parallel part)

Summary

- Concurrency: try to identify *independent* operations that can be performed in parallel
- Only parallelize the *common case*, and make sure it is *frequent enough* to matter
- Check if the problem size scales when adding parallelism

Optimizing Compilers

- Compiler converts C/Fortran/etc into machine code for CPU
- Almost all compilers capable of significant optimizations
  - Not enabled by default
  - Be sure you turn on optimizations!
    - Often, several levels to choose from
    - Minimal –O, Good –O2, Too much –O3

Organization of a Hypothetical Optimizing Compiler

– taken from Bacon et al, "Compiler Transformations for High-Performance Computing", ACM Computing Surveys, 1994

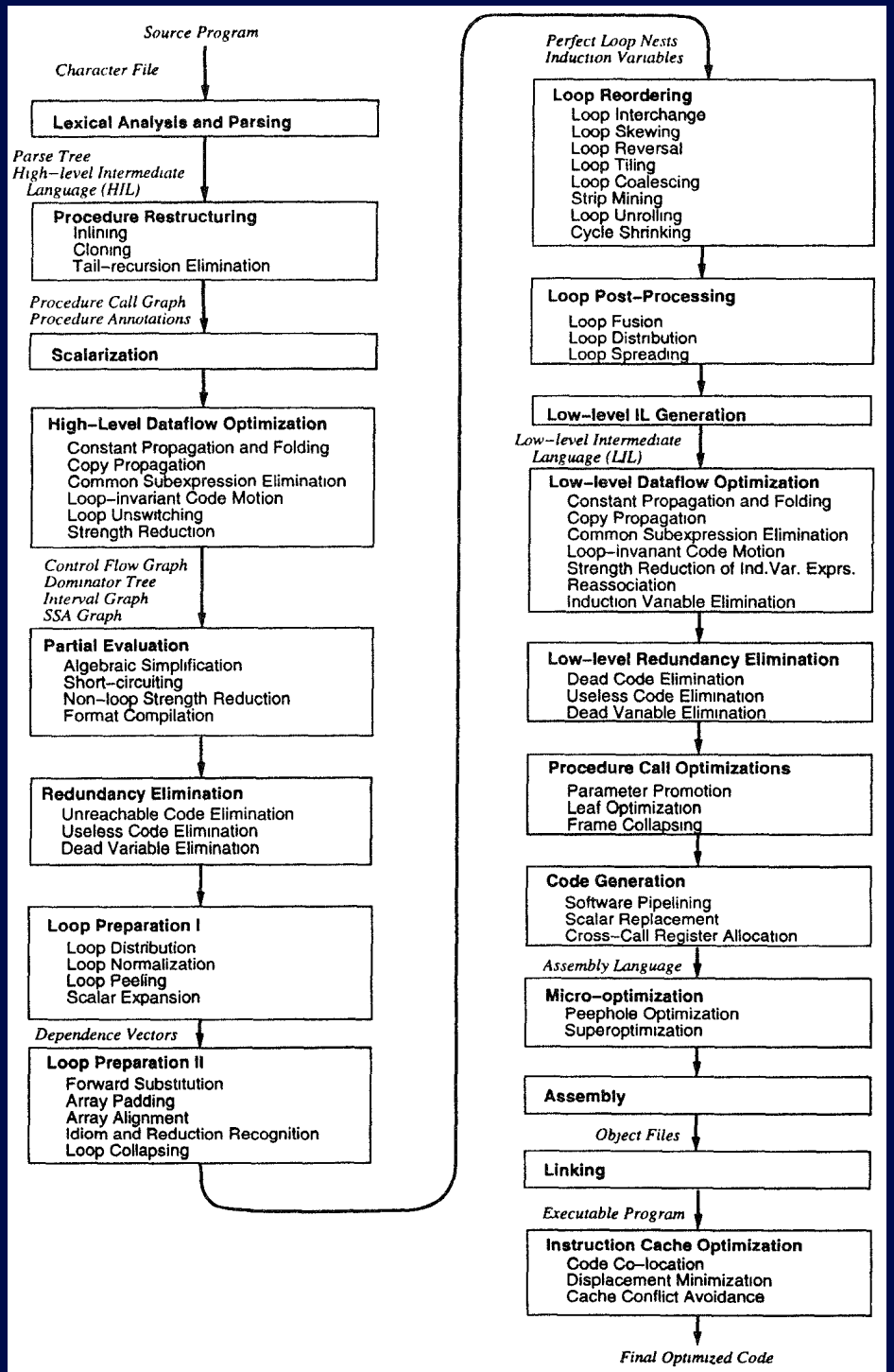


Figure 2. Organization of a hypothetical optimizing compiler.

Interesting to follow…

- Top500.org
- Top500.org/green500
- Graph500.org

- Hotchips.org
- Hot Interconnects (hoti.org)

TODO

- NOW: 3 volunteers to present next Friday
  - Only these 3 volunteers may present again to improve grade
- BEFORE FRIDAY: Send email for dropbox access
  - guy.lemieux@gmail.com
- BEFORE FRIDAY: Install dropbox
- Come to class on Friday
  - Assignment 0 (due following Friday)
  - Reading assignments
  - Presentation signups

Assignment 0

- As part of Assignment 0, please do the following…
  1. Write a “vanilla” Matrix Multiply program in C
     - Parameterize the data type to support one of {int, float, double}
     - Initialize with random data
     - Measure and display the run-time of matrix multiply only
  2. Execute 3 executable versions (int, float, double) each with 4 compiler optimization settings (none, –O, –O2, –O3). Use matrix sizes with 128x128 to 4096x4096.
  3. Rewrite your “vanilla” code by applying as many typical compiler transformations as you can; document the ones you applied in a list.
  4. Repeat execution (step 2).
  5. Write a 1-page report, including a table to show runtime results of steps 2 and 4.

TODO

- NOW: pick 3 students to present next Friday
  - Volunteers now to present first
  - May present a second time to improve grade
- BEFORE FRIDAY: Send email for dropbox access
  - guy.lemieux@gmail.com
- BEFORE FRIDAY: Install dropbox
- Come to class on Friday
  - Assignment 0 (due following Friday)
  - Reading assignments
  - Presentation signups

Readings (no reports necessary)

- Read papers on Flynn, Amdahl, Gustafson (before this lecture)
- Read (Wikipedia page, UIUC slides) on Compiler Optimizations (after this lecture)
- For further reference…
  - Bacon survey paper (common optimizations)
  - Supercomputers paper (dependence analysis)