

18-447 Lecture 21:

Parallel Architecture Overview

James C. Hoe
Department of ECE
Carnegie Mellon University

Housekeeping

- Your goal today
 - see the diverse landscape of parallel computer architectures/organizations
 - set the context for focused topics to come
- Notices
 - Lab 4, **due week 14**
 - HW6, **due Monday 5/2 noon**
 - Midterm 2 Regrade, **due Monday, 4/25**
 - **All known final conflicts have been settled!!**
- Readings
 - P&H Ch 6
 - *Synthesis Lecture: Parallel Processing, 1980 to 2020*

Parallelism Defined

- T_1 (work measured in time):
 - time to do work with 1 PE
- T_∞ (critical path):
 - time to do work with infinite PEs
 - T_∞ bounded by dataflow dependence
- Average parallelism:

$$P_{avg} = T_1 / T_\infty$$

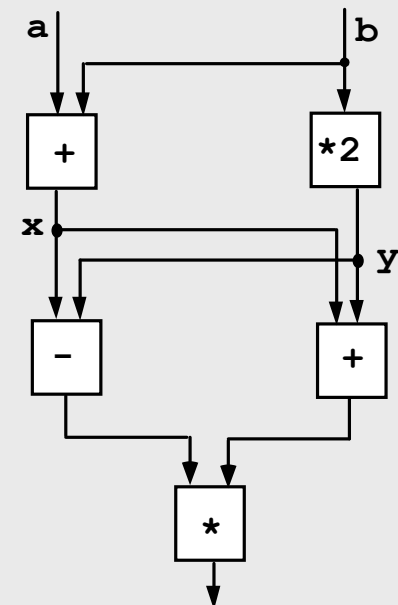
- For a system with p PEs

$$T_p \geq \max\{T_1/p, T_\infty\}$$

- When $P_{avg} \gg p$

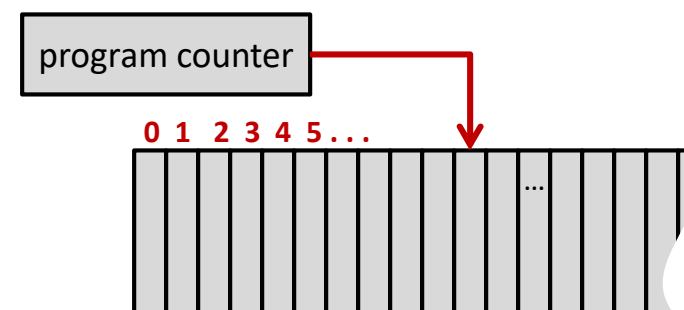
$$T_p \approx T_1/p, \text{ aka "linear speedup"}$$

```
x = a + b;
y = b * 2
z = (x-y) * (x+y)
```



A Non-Parallel “Architecture”

- Memory holds both program and data
 - instructions and data in a linear memory array
 - instructions can be modified as data
- Sequential instruction processing
 1. **program counter (PC)** identifies current instruction
 2. fetch instruction from memory
 3. update some state (e.g. **PC** and memory) as a function of current state (according to instruction)
 4. repeat



Review

dominant paradigm since its invention

Inherently Parallel Architecture

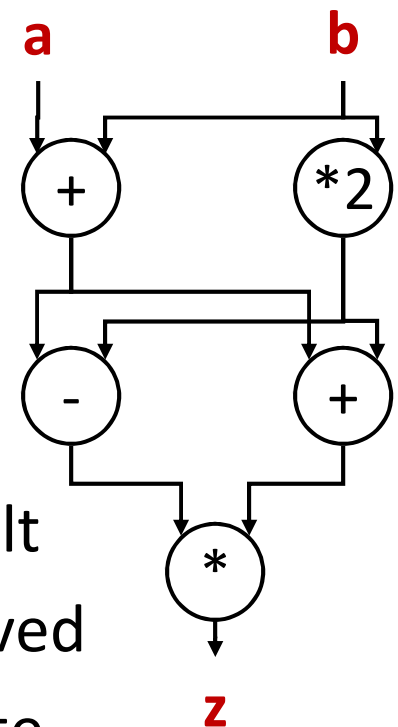
- Consider a von Neumann program
 - What is the significance of the program order?
 - What is the significance of the storage locations?

```

v := a + b ;
w := b * 2 ;
x := v - w ;
y := v + w ;
z := x * y ;

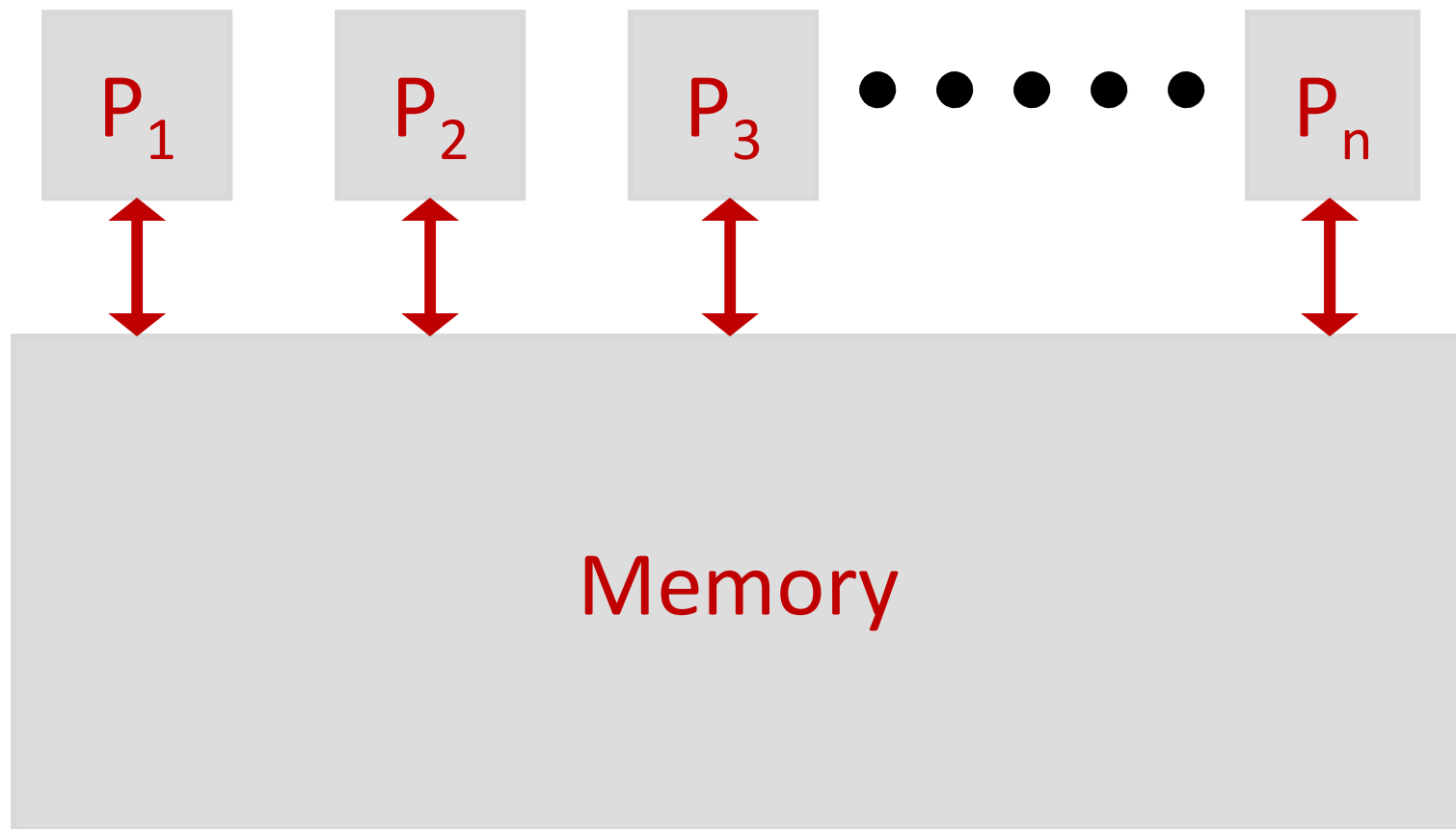
```

- Dataflow program instruction ordering implied by data dependence
 - instruction specifies who receives the result
 - instruction executes when operands received
 - no program counter, no* intermediate state



[dataflow figure and example from Arvind]

More Conventionally Parallel



Review

Do you naturally think parallel or sequential?

Simple First Look: Data Parallelism

- Abundant in matrix operations and scientific/numerical applications
- Example: AXPY (from Level 1 Basic Linear Algebra Subroutine)

$$\mathbf{Y} = \mathbf{a} * \mathbf{X} + \mathbf{Y} = \left\{ \begin{array}{l} \text{for } (i=0; i < N; i++) \{ \\ \quad \mathbf{Y}[i] = \mathbf{a} * \mathbf{X}[i] + \mathbf{Y}[i] \\ \} \end{array} \right.$$

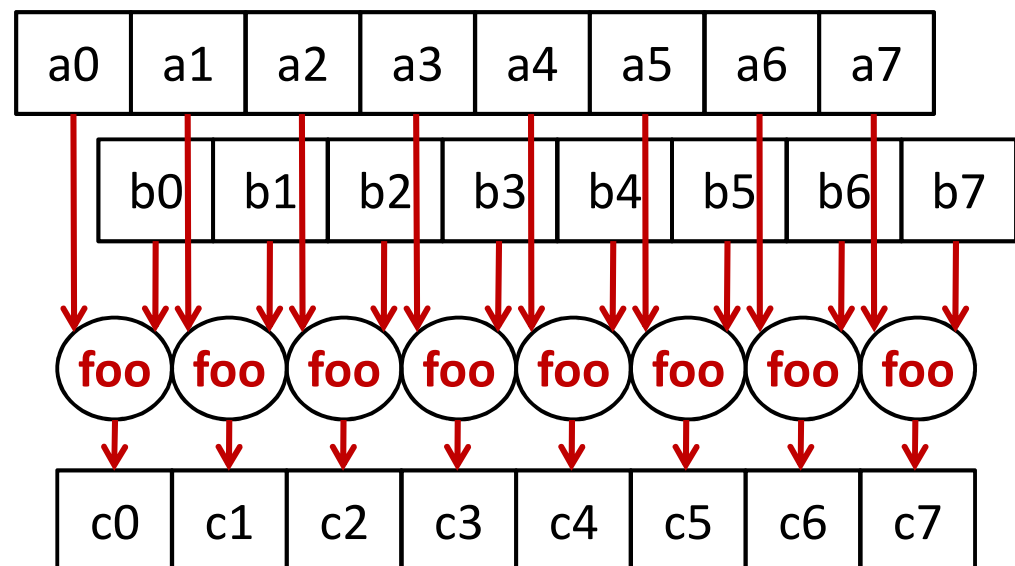
- \mathbf{Y} and \mathbf{X} are vectors
- same operations repeated on each $\mathbf{Y}[i]$ and $\mathbf{X}[i]$
- no data dependence across iterations

How to exploit data parallelism?

Parallelism vs Concurrency

```
for (i=0; i<N; i++) {  
    C[i]=foo(A[i], B[i])  
}
```

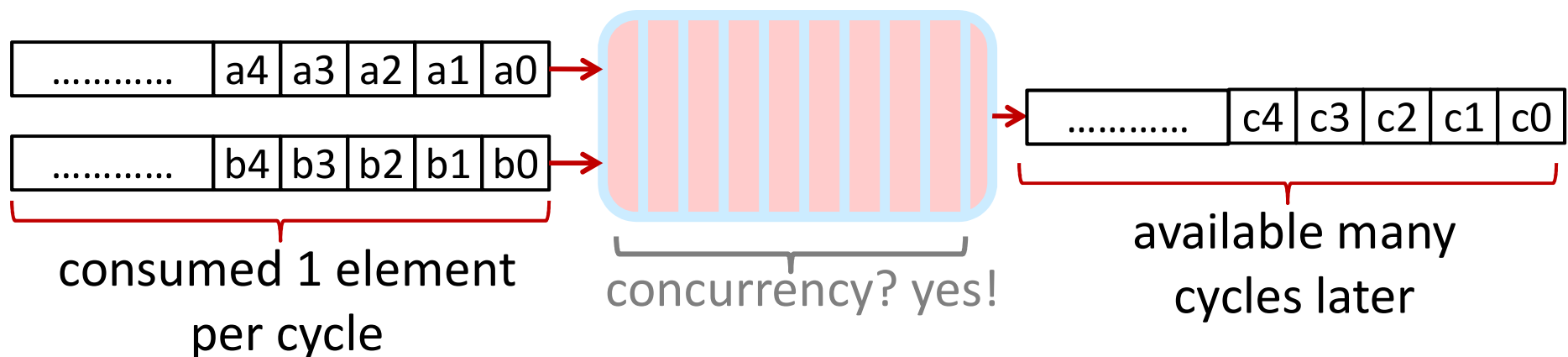
- Instantiate **k** copies of the hardware unit **foo** to process **k** iterations of the loop in parallel



Parallelism vs Concurrency

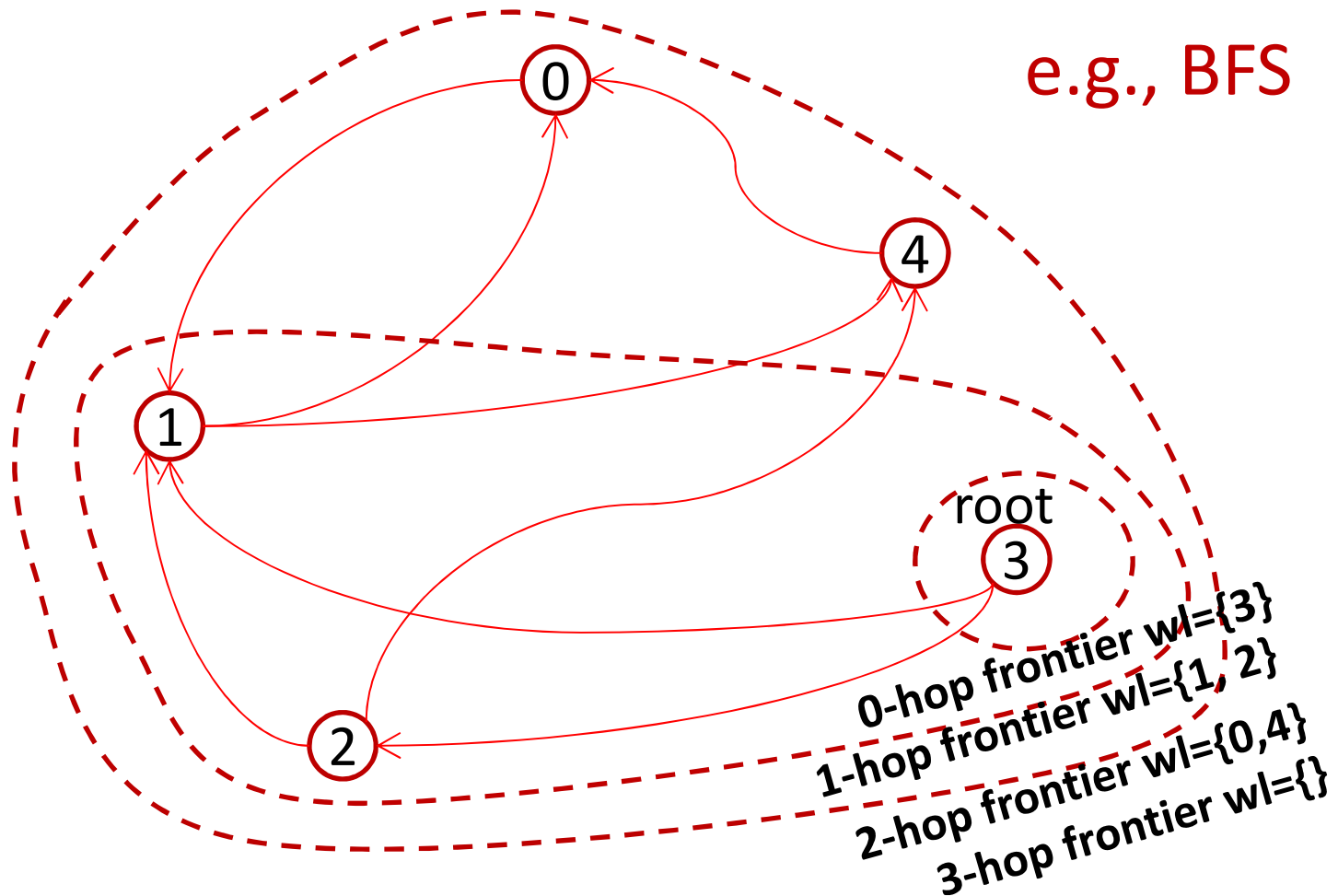
```
for (i=0; i<N; i++) {  
    C[i]=foo(A[i], B[i])  
}
```

- Build a deeply (super)pipelined version of **foo()**



Can combine concurrency and
pipelining at the same time

Harder Kind of Parallelism: Irregular and Data Dependent



Neighbors can be visited concurrently, usually without conflict

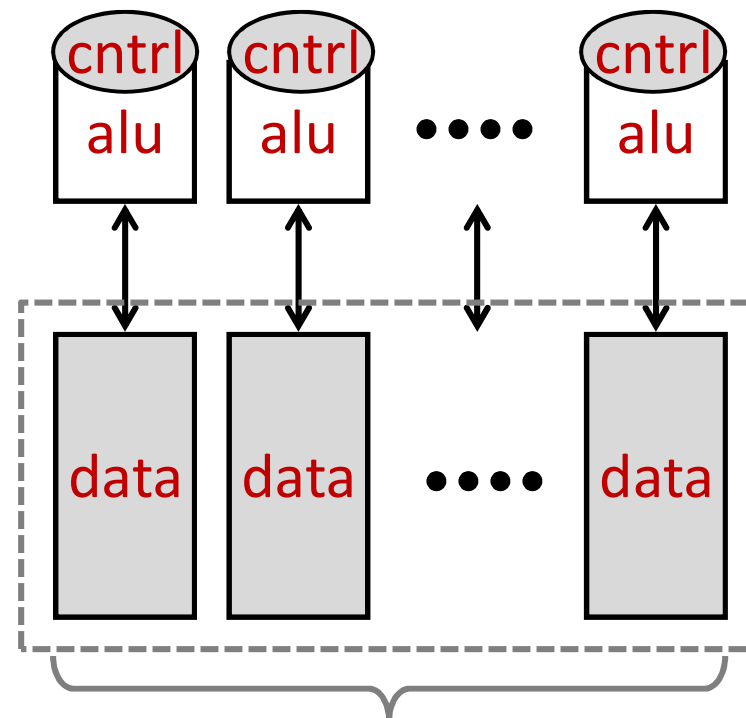
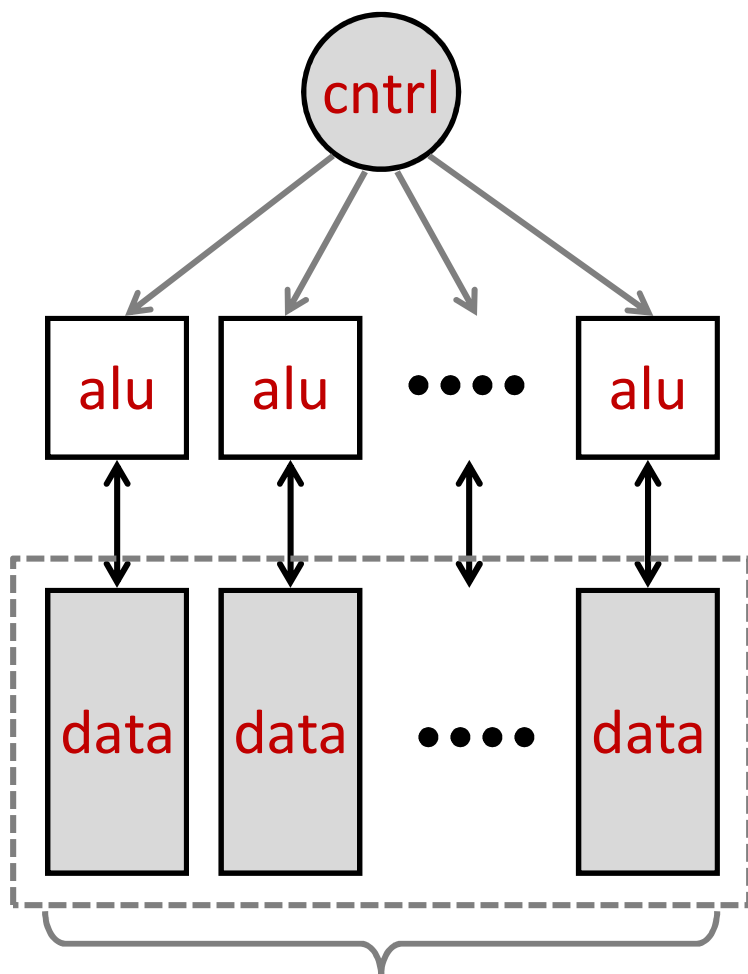
A Spotty Tour of the MP Universe

Classic Thinking: Flynn's Taxonomy

	S ingle I nstruction Stream	M ultiple I nstruction Stream
S ingle D ata Stream	SISD: your vanilla uniprocessor	MISD: DB query??
M ultiple D ata Stream	SIMD: many PEs following common instruction stream/control-flow on different data	MIMD: fully independent programs/control-flows working in parallel (collaborating SISDs ?)

SIMD vs. MIMD

(an abstract and general depiction)

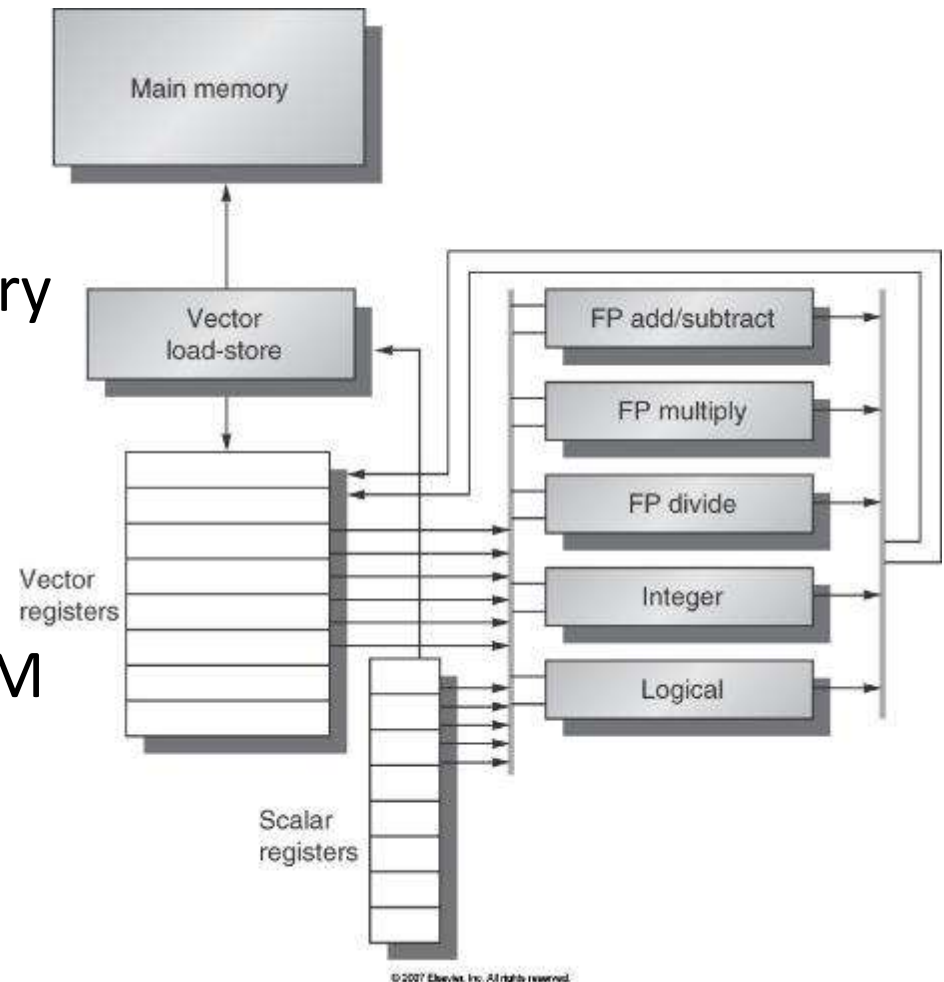


Variety in the details

- Scale, technology, application
- Concurrency
 - granularity of concurrency (how finely is work divided)—*whole programs down to bits*
 - regularity—*all “nodes” look the same and look out to the same environment*
 - static vs. dynamic—*e.g., load-balancing*
- Communication
 - message-passing vs. shared memory
 - granularity of communication—*words to pages*
 - interconnect and interface design/performance

SIMD: Vector Machines

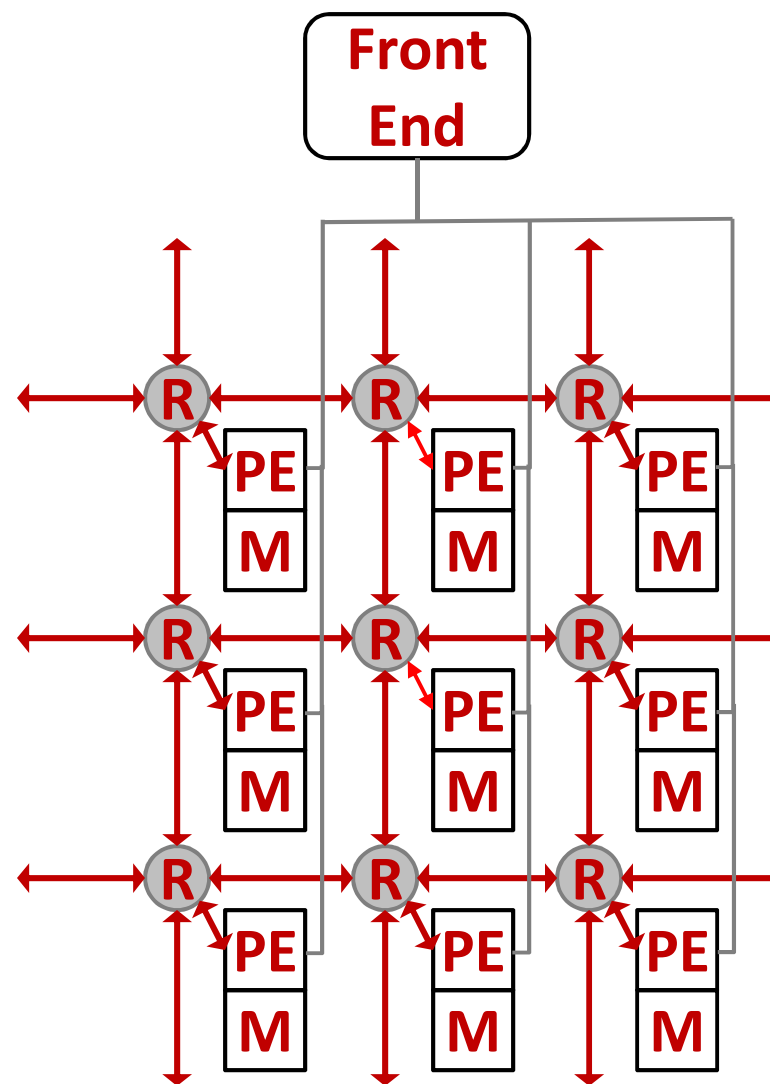
- Vector data type and regfile
- Deeply pipelined fxn units
- Matching high-perf load-store units and multi-banked memory
- E.g., Cray 1, circa 1976
 - 64 x 64-word vector RF
 - 12 pipelines, 12.5ns clk
 - ECL 4-input NAND and SRAM (no caches!!)
 - 2x25-ton cooling system
 - 250 MIPS peak for ~10M 1970\$



[Figure from H&P CAaQA, COPYRIGHT 2007 Elsevier.
ALL RIGHTS RESERVED.]

SIMD: Big-Irons

- Sea of PEs on a regular grid
 - synchronized common cntrl
 - direct access to local mem
 - nearest-neighbor exchanges
 - special support for broadcast, reduction, etc.
- E.g., Thinking Machines CM-2
 - 1000s of bit-sliced PEs lock-step controlled by a common sequencer
 - “hypercube” topology
 - special external I/O nodes



SIMD: Modern Renditions, e.g.,

- Intel SSE (Streaming SIMD Extension), 1999
 - 16 x 128-bit “vector” registers, 4 floats or 2 doubles
 - SIMD instructions: ld/st, arithmetic, shuffle, bitwise
 - SSE4 with true full-width operations

Core i7 does upto 4 sp-mult & 4 sp-add
per cyc per core, (24GFLOPS @3GHz)

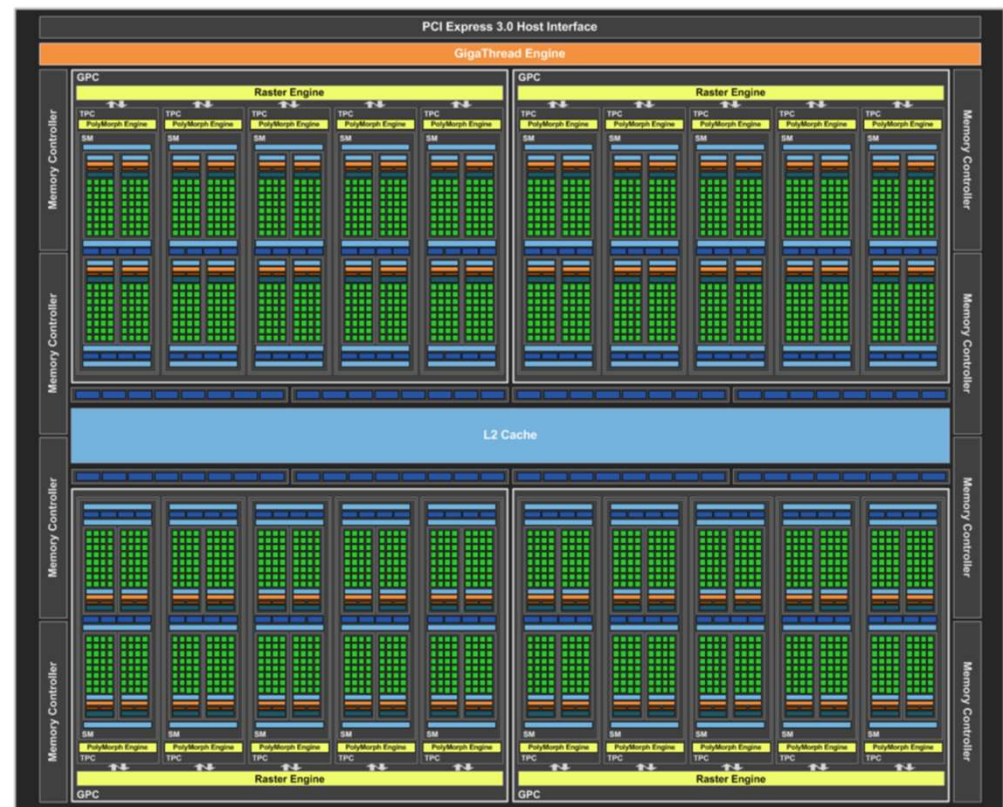
- AVX 2 doubles the above (over 1TFLOPS/chip)
- “GP” GPUs . . . (next slide)

*Simple hardware, big perf numbers but
only if massively data-parallel app!!*

E.g., 8+ TFLOPs Nvidia GP104 GPU

- 20 Streaming Multiproc
 - 128 SIMD lane per SM
 - 1 mul, 1 add per lane
 - 1.73 GHz (boosted)
- Performance
 - 8874 GFLOPs
 - 320GB/sec
 - 180 Watt

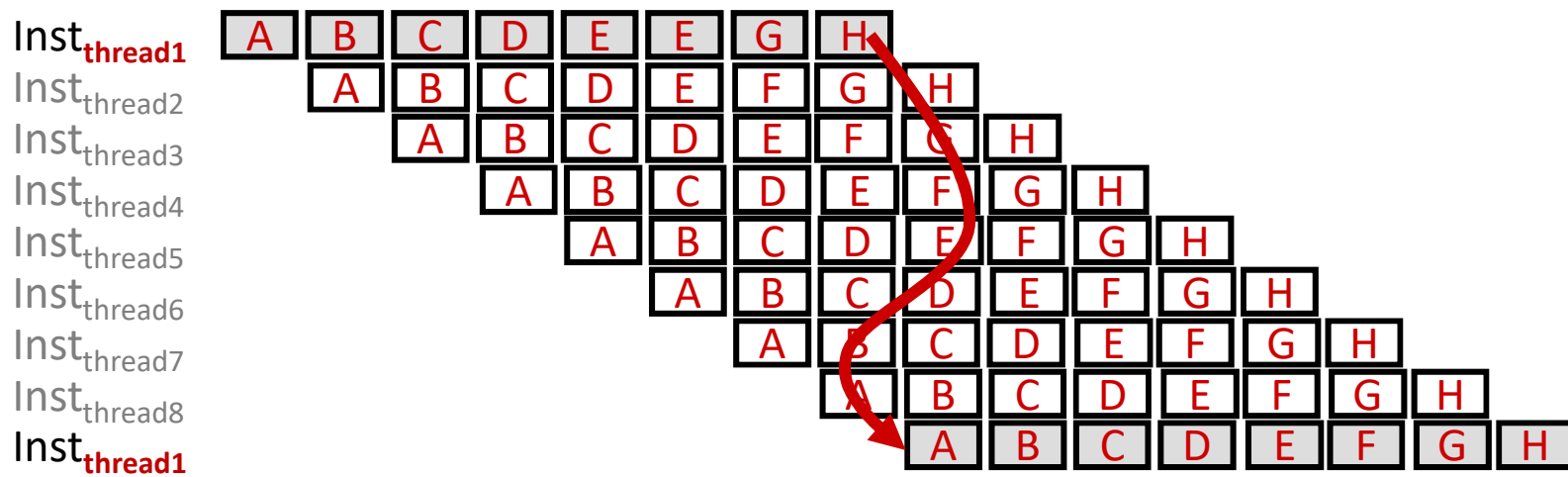
*How many FLOPs per
Watt? How many FLOPs
per DRAM byte accessed?*



[NVIDIA GeForce GTX 1080 Whitepaper]

Aside: IPC, ILP, and TLP

- Each cycle, select a “ready” thread from scheduling pool
 - only one instruction per thread in flight at once
 - on a long stall (DRAM), remove thread from scheduling
- Simpler and faster pipeline implementation since
 - no data dependence, hence no stall or forwarding
 - no penalty in making pipeline deeper



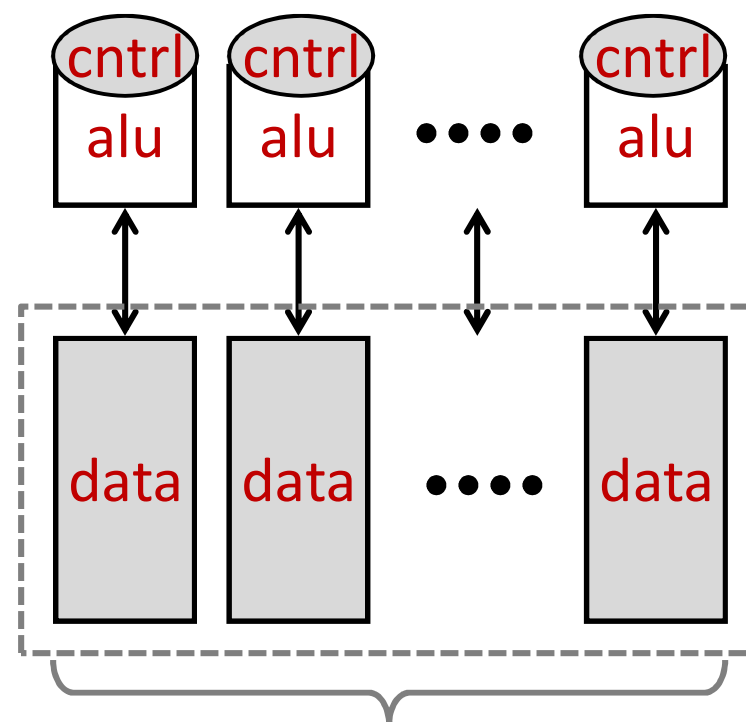
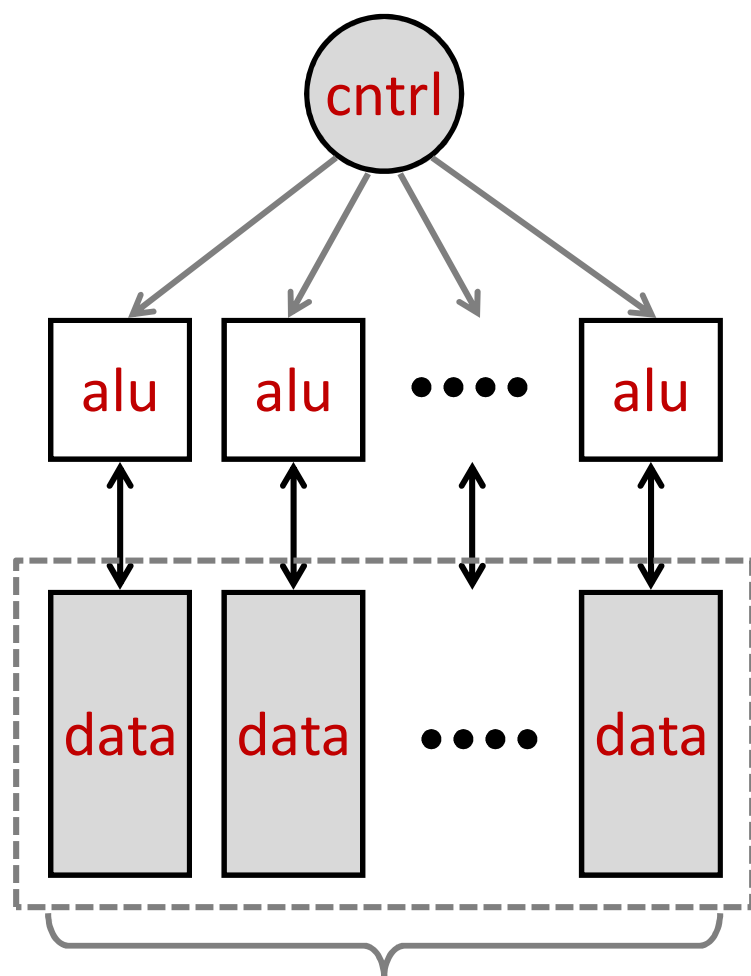
e.g., Barrel Processor [HEP, Smith]

Historical: what 1 TFLOP meant in 1996

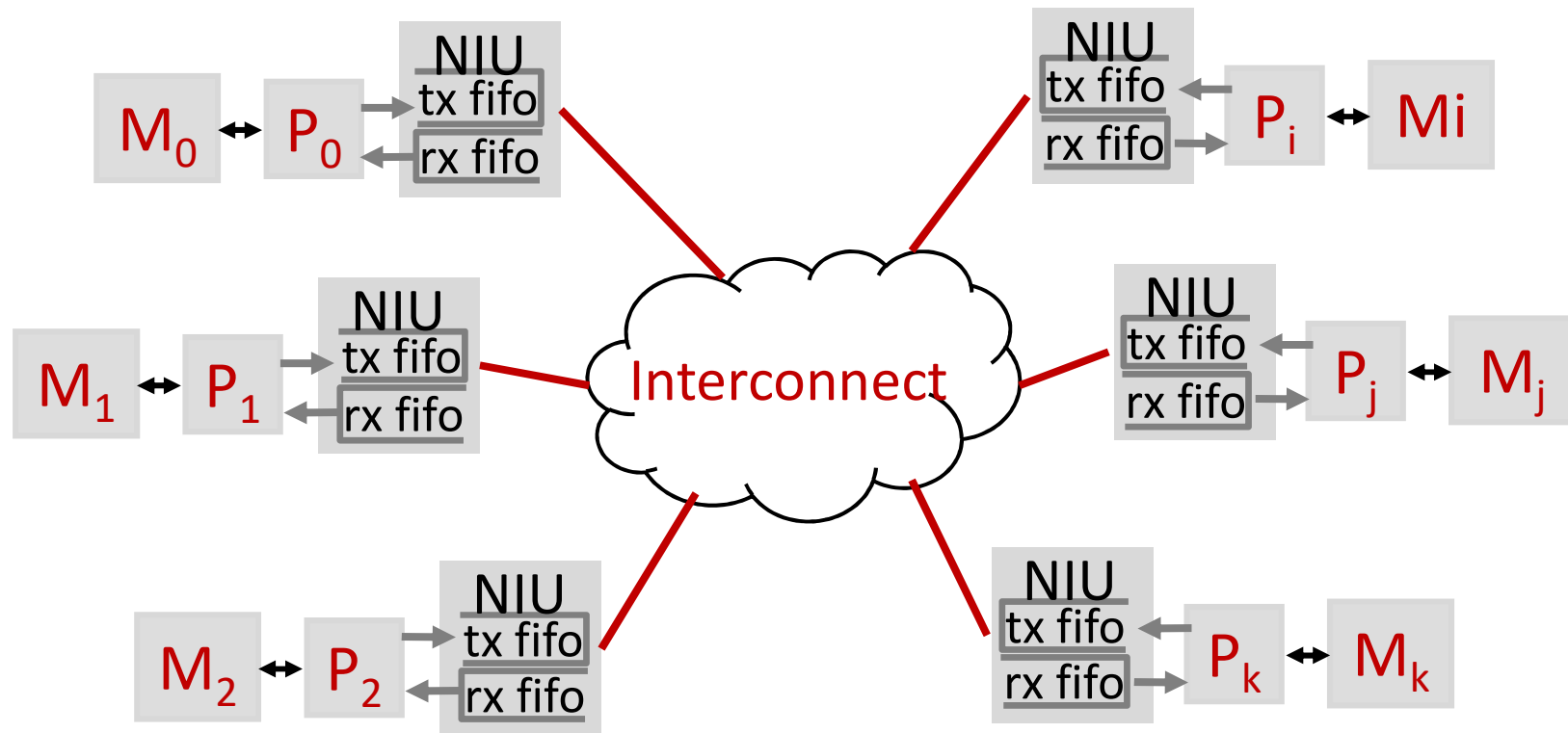
- ASCI Red, 1996—*World's 1st TFLOP computer!!*
 - \$50M, 1600ft² system
 - ~10K 200MHz PentiumPro's
 - ~1 TByte DRAM (total)
 - 500kW to power + 500kW on cooling
- Advanced Simulation and Computing Initiative
 - how to know if nuclear stockpile still good if you can't blow one up to find out?
 - require ever more expensive simulation as stockpile aged
 - Red 1.3TFLOPS 1996; Blue Mountain/Pacific 4TFLOPS 1998; White 12TFLOPS 2000; Purple 100TFLOPS 2005; . . . IBM Summit 200**Peta**FLOPS

SIMD vs. MIMD

(an abstract and general depiction)



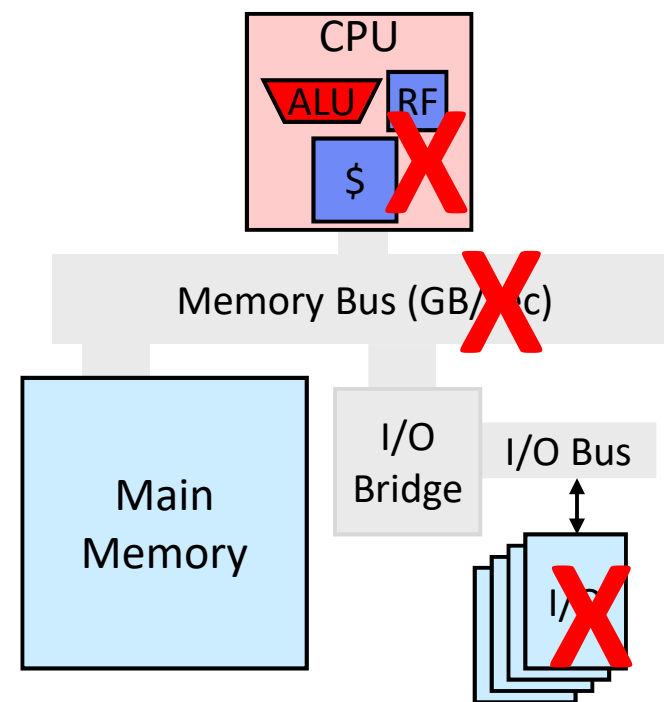
MIMD: Message Passing



- Private address space and memory per processor
- Parallel threads on different processors communicate by explicit sending and receiving of messages

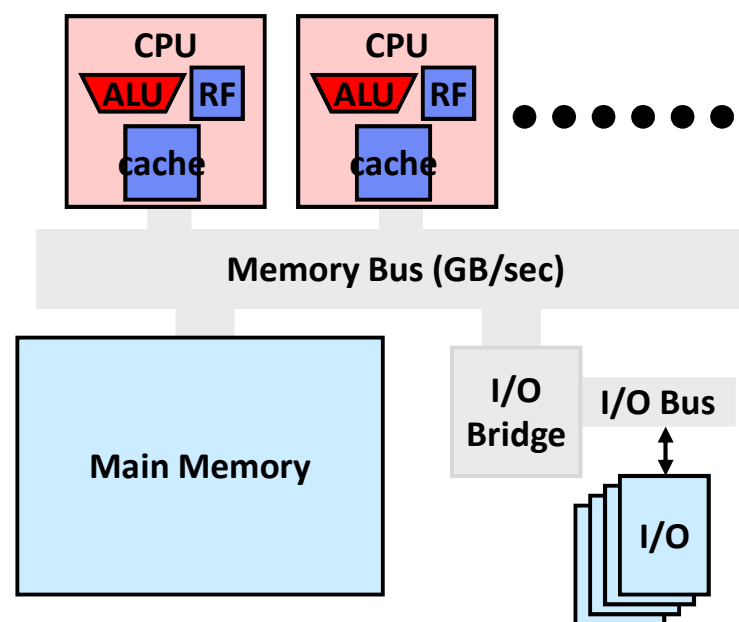
MIMD Message Passing Systems (by network interface placement)

- Any Clusters (e.g., data centers, Beowulf) (*I/O bus*)
 - Linux PCs connected by Ethernet
- “High-Performance Computing” Clusters (*I/O bus*)
 - stock workstations/servers but exotic interconnects, e.g., Myrinet, HIPPI, Infiniband, etc.
- Supers (*memory bus*)
 - stock CPUs on custom platform
 - e.g., Cray XT5 (“fastest” in 2011 224K AMD Opteron)
- Inside the CPU
 - single-instruction send/receive
 - e.g., iAPX 432 (1981), Transputers (80s), . . . (now?)



MIMD Shared Memory: Symmetric Multiprocessors (SMPs)

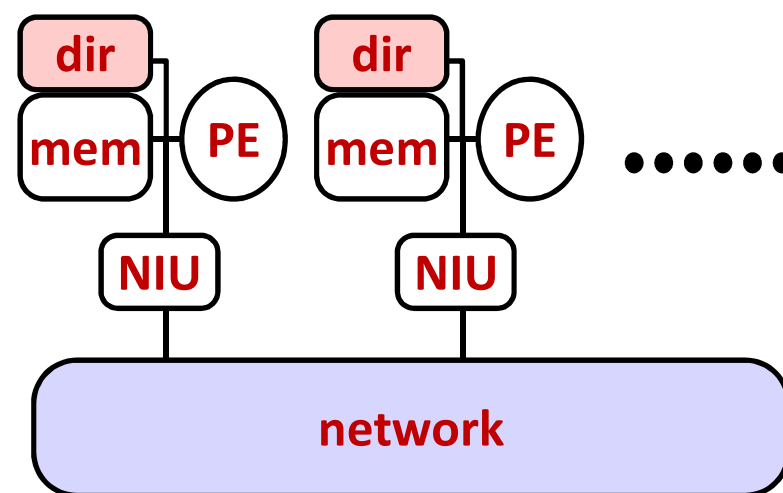
- Symmetric means
 - identical procs connected to common memory
 - all procs have equal access to system (mem & I/O)
 - OS can schedule any process on any processor
- Uniform Memory Access (UMA)
 - processor/memory connected by bus or crossbar
 - all processors have equal memory access performance to all memory locations
 - caches need to stay coherent



MIMD Shared Memory: Big Irons

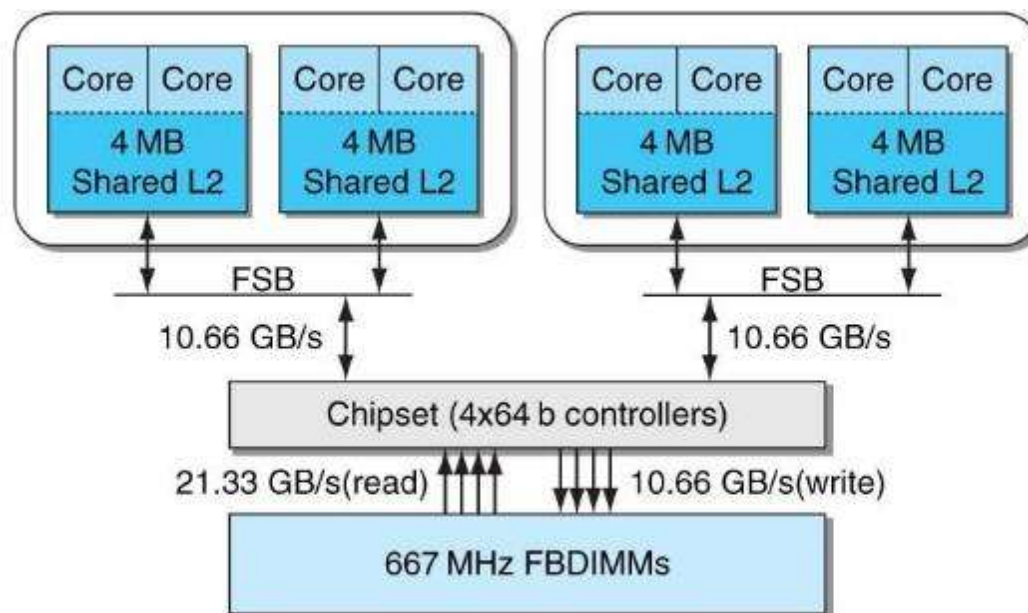
Distributed Shared Memory

- UMA hard to scale due to concentration of BW
- Large scale SMPs have distributed memory with non-uniform memory (NUMA)
 - “local” memory pages (faster to access)
 - “remote” memory pages (slower to access)
 - cache-coherence still possible but complicated
- E.g., SGI Origin 2000
 - upto 512 CPUs and 512GB DRAM (\$40M)
 - 48 128-CPU system was collectively the 2nd fastest computer (3TFLOPS) in 1999



MIMD Shared Memory: it is everywhere now!

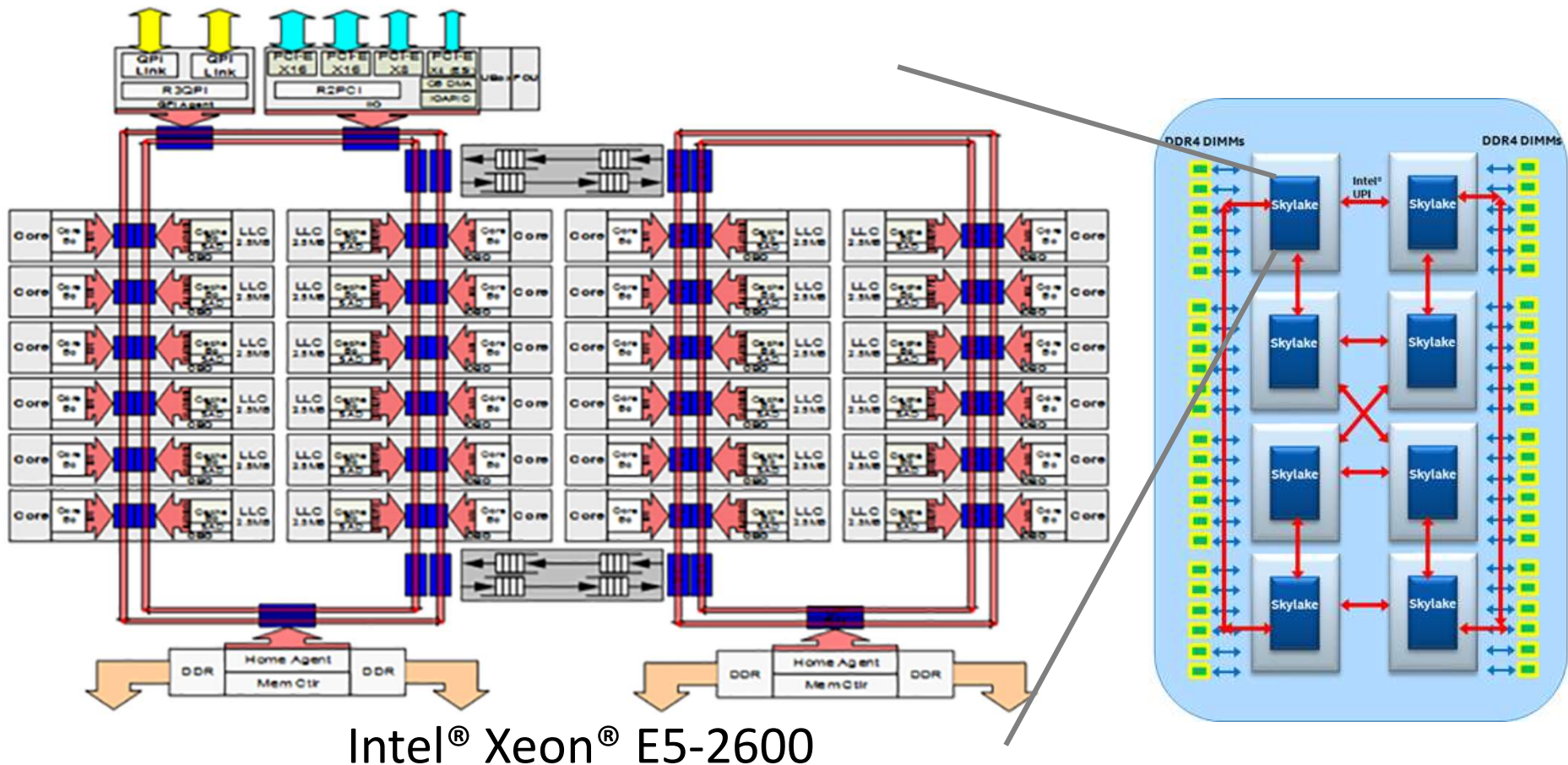
- General-purpose “multicore” processors implement SMP (not UMA) on a single chip
- Moore’s Law scaling in number of core’s



Intel Xeon e5345

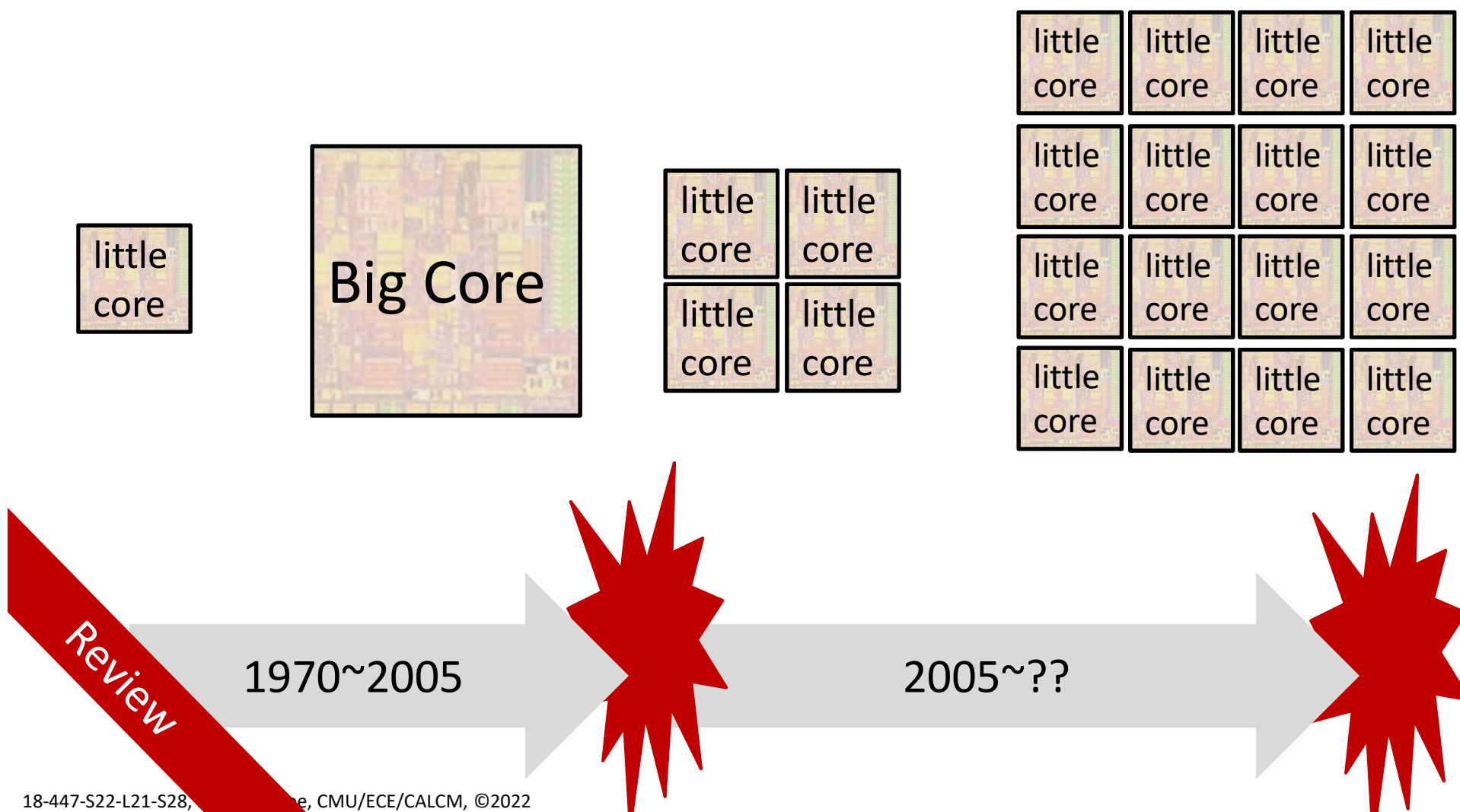
[Figure from P&H CO&D, COPYRIGHT 2009 Elsevier. ALL RIGHTS RESERVED.]

Today's New Normal

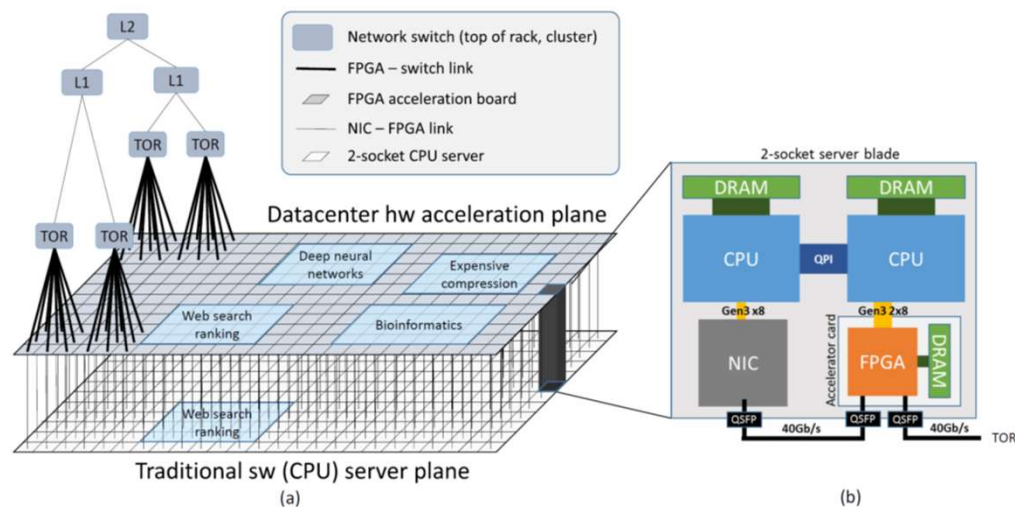


[<https://software.intel.com/en-us/articles/intel-xeon-processor-scalable-family-technical-overview>]

Remember how we got here

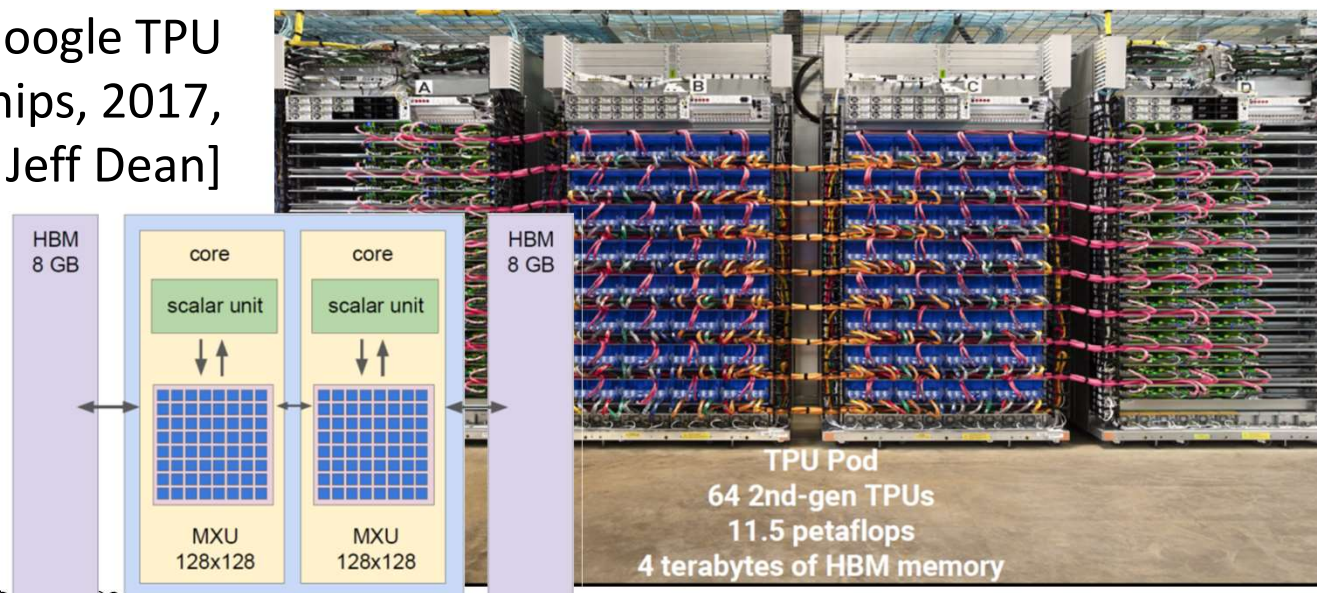


Today's Exotic

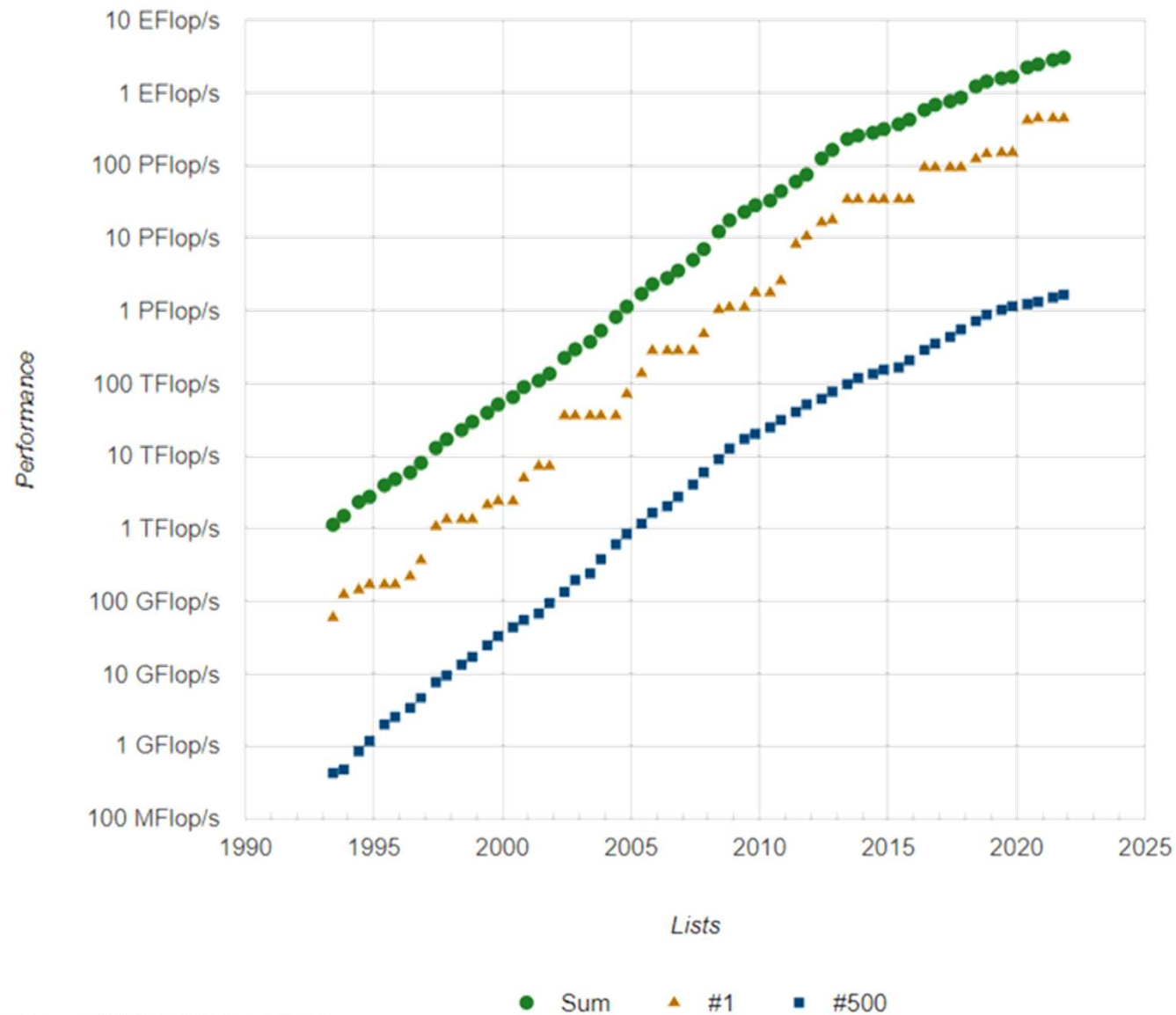


Microsoft Catapult
[MICRO 2016,
Caulfield, et al.]

Google TPU
[Hotchips, 2017,
Jeff Dean]



March Toward Exascale (10^{18}) HPC



www.top500.org

Top 500 Nov 21

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442,010.0	537,212.0	29,899
2	Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
3	Sierra - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
4	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
5	Perlmutter - HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10, HPE DOE/SC/LBNL/NERSC United States	761,856	70,870.0	93,750.0	2,589
6	Selene - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia NVIDIA Corporation United States	555,520	63,460.0	79,215.0	2,646