

T. HOEFLER, M. PUESCHEL

# Lecture 1: Introduction

Teaching assistant: Salvatore Di Girolamo

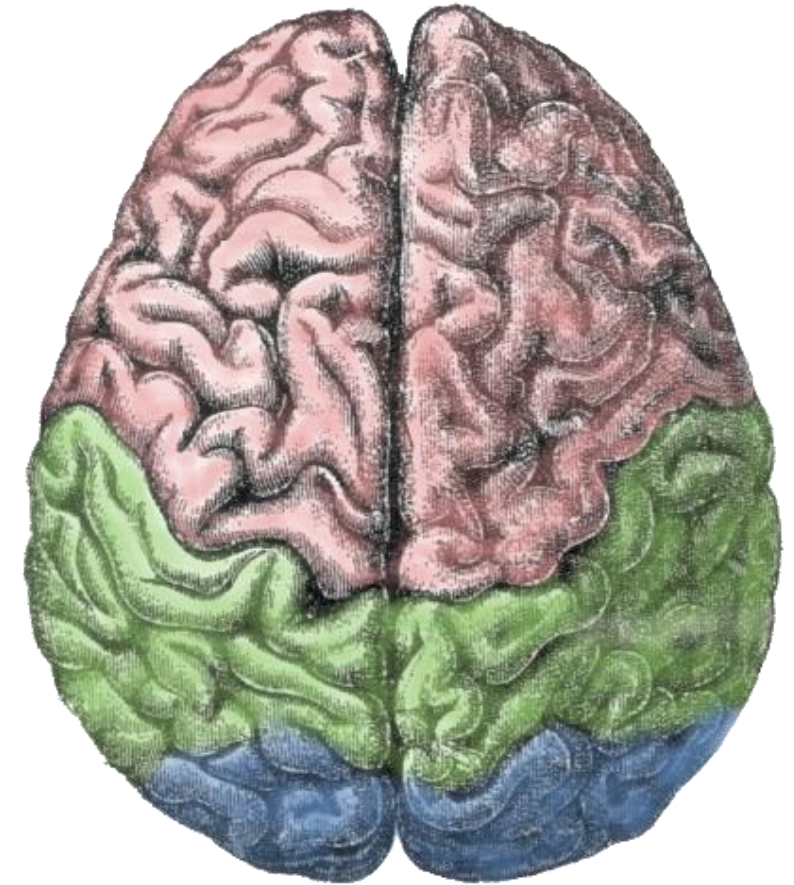


# Goals of this lecture

- **Motivate you!**
- **What is parallel computing?**
  - And why do we need it?
- **What is high-performance computing?**
  - What's a Supercomputer and why do we care?
- **Basic overview of**
  - Programming models
    - Some examples*
  - Architectures
    - Some case-studies*
- **Provide context for coming lectures**

## Let us assume ...

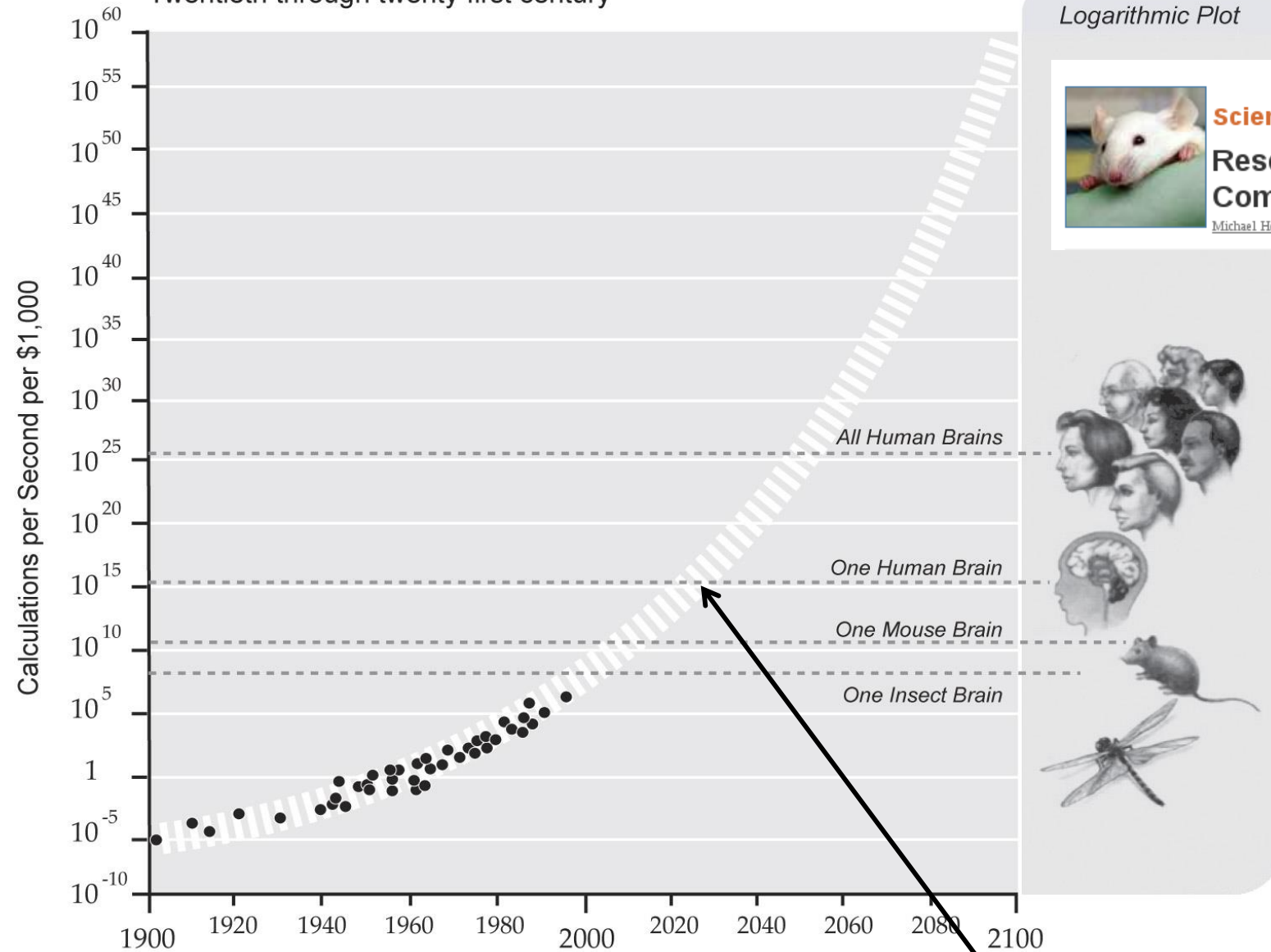
- ... you were to build a machine like this ...
  
- ... we know how each part works
  - There are just many of them!
  - Question: How many calculations per second are needed to emulate a brain?



Source: wikipedia

# Exponential Growth of Computing

Twentieth through twenty first century



Source: www.singularity.com

Year

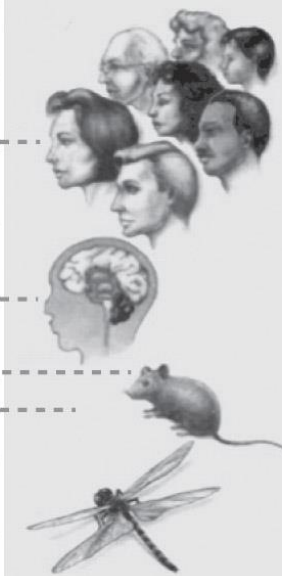
Logarithmic Plot



Science

Researchers Simulate Mouse Brain on Computer

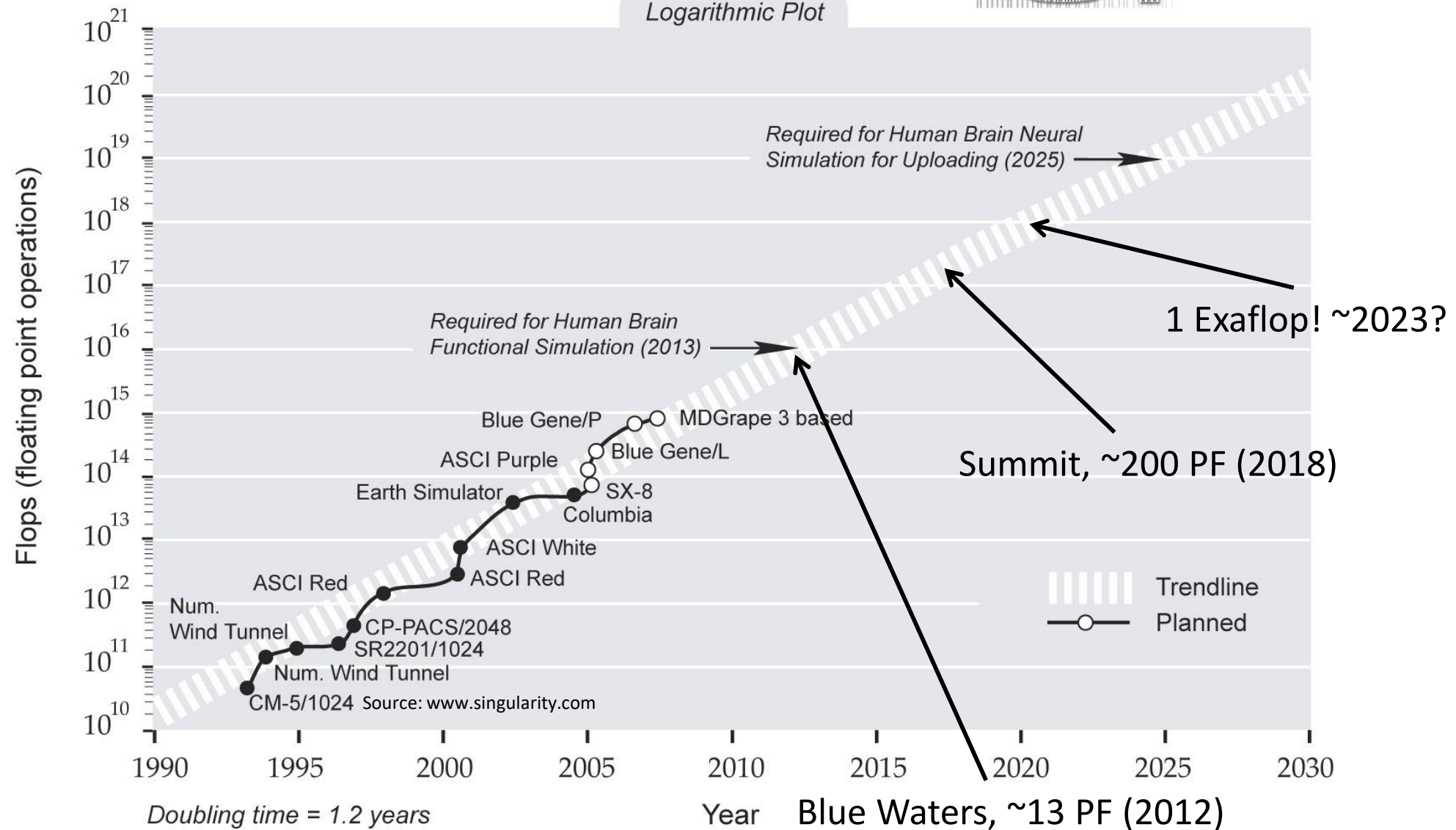
[Michael Hoffman \(Blog\)](#) - April 30, 2007 5:57 PM



Can we do this today?

# Growth in Supercomputer Power

Logarithmic Plot



# Human Brain – No Problem!

- ... not so fast, we need to understand how to program those machines ...

# Human Brain – No Problem!

## Simulating 1 second of human brain activity takes 82,944 processors

By Ryan Whitwam on August 5, 2013 at 1:34 pm | 21 Comments



*Scooped!*

### Share This Article

436	123	228	108	24
<a href="#">f Like</a>	<a href="#">t Tweet</a>	<a href="#">u 228</a>	<a href="#">g +1</a>	<a href="#">in Share</a>

The **brain** is a deviously complex biological computing device that even the fastest supercomputers in the world fail to emulate. Well, that's not entirely true anymore. Researchers at the Okinawa Institute of Technology Graduate University in Japan and

Forschungszentrum Jülich in Germany have managed to simulate a single second of human brain activity in a very, very powerful computer.

Source: extremetech.com

## Other problem areas: Scientific Computing

- **Most natural sciences are simulation driven or are moving towards simulation**
  - Theoretical physics (solving the Schrödinger equation, QCD)
  - Biology (Gene sequencing)
  - Chemistry (Material science)
  - Astronomy (Colliding black holes)
  - Medicine (Protein folding for drug discovery)
  - Meteorology (Storm/Tornado prediction)
  - Geology (Oil reservoir management, oil exploration)
  - and many more ... (even Pringles uses HPC)
- **Quickly emerging areas for HPC/parallel computing technologies**
  - Big data processing
  - Deep learning
  - HPC was always at the forefront of specialization
- **Many cloud services require HPC/parallel computing**
  - Transaction processing/analysis
  - Stock markets
  - Making movies etc.





# What can faster computers do for us?

- Solving **bigger problems** than we could solve before!
  - E.g., Gene sequencing and search, simulation of whole cells, mathematics of the brain, ...
  - The size of the problem grows with the machine power
    - *Weak Scaling*
  
- Solve today's problems **faster!**
  - E.g., large (combinatorial) searches, mechanical simulations (aircrafts, cars, weapons, ...)
  - The machine power grows with constant problem size
    - *Strong Scaling*

# High-Performance Computing (HPC)

- a.k.a. “Supercomputing”
- **Question: define “Supercomputer”!**
  - “A supercomputer is a computer at the frontline of contemporary processing capacity--particularly speed of calculation.” (Wikipedia)
  - Usually quite expensive (\$s and MW) and big (space)
- **HPC is a quickly growing niche market**
  - Not all “supercomputers”, wide base
  - Important enough for vendors to specialize
  - Very important in research settings (up to 40% of university spending)
    - *“Goodyear Puts the Rubber to the Road with High Performance Computing”*
    - *“High Performance Computing Helps Create New Treatment For Stroke Victims”*
    - *“Procter & Gamble: Supercomputers and the Secret Life of Coffee”*
    - *“Motorola: Driving the Cellular Revolution With the Help of High Performance Computing”*
    - *“Microsoft: Delivering High Performance Computing to the Masses”*

# The Top500 List

- **A benchmark, solve  $Ax=b$** 
  - As fast as possible! → as big as possible 😊
  - Reflects **some** applications, not all, not even many
  - Very good historic data!
- **Speed comparison for computing centers, states, countries, nations, continents** 😞
  - Politicized (sometimes good, sometimes bad)
  - Yet, fun to watch

# The Top500 List (June 2015)

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	DOE/SC/Oak Ridge National Laboratory United States	<b>Summit</b> - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband IBM	2,282,544	122,300.0	187,659.3	8,806
2	National Supercomputing Center in Wuxi China	<b>Sunway TaihuLight</b> - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway NRPC	10,649,600	93,014.6	125,435.9	15,371
3	DOE/NNSA/LLNL United States	<b>Sierra</b> - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband IBM	1,572,480	71,610.0	119,193.6	
4	National Super Computer Center in Guangzhou China	<b>Tianhe-2A</b> - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 NUDT	4,981,760	61,444.5	100,678.7	18,482
5	National Institute of Advanced Industrial Science and Technology (AIST) Japan	<b>AI Bridging Cloud Infrastructure (ABCI)</b> - PRIMERGY CX2550 M4, Xeon Gold 6148 20C 2.4GHz, NVIDIA Tesla V100 SXM2, Infiniband EDR Fujitsu	391,680	19,880.0	32,576.6	1,649
6	Swiss National Supercomputing Centre (CSCS) Switzerland	<b>Piz Daint</b> - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect, NVIDIA Tesla P100 Cray Inc.	361,760	19,590.0	25,326.3	2,272
7	DOE/SC/Oak Ridge National Laboratory United States	<b>Titan</b> - Cray XK7, Opteron 6274 16C 2.200GHz, Cray C... NVIDIA	560,640	17,590.0	27,112.5	8,209

# Piz Daint @ CSCS



## Blue Waters in 2009

Imagine you're designing a \$500 M supercomputer, and all you have is:

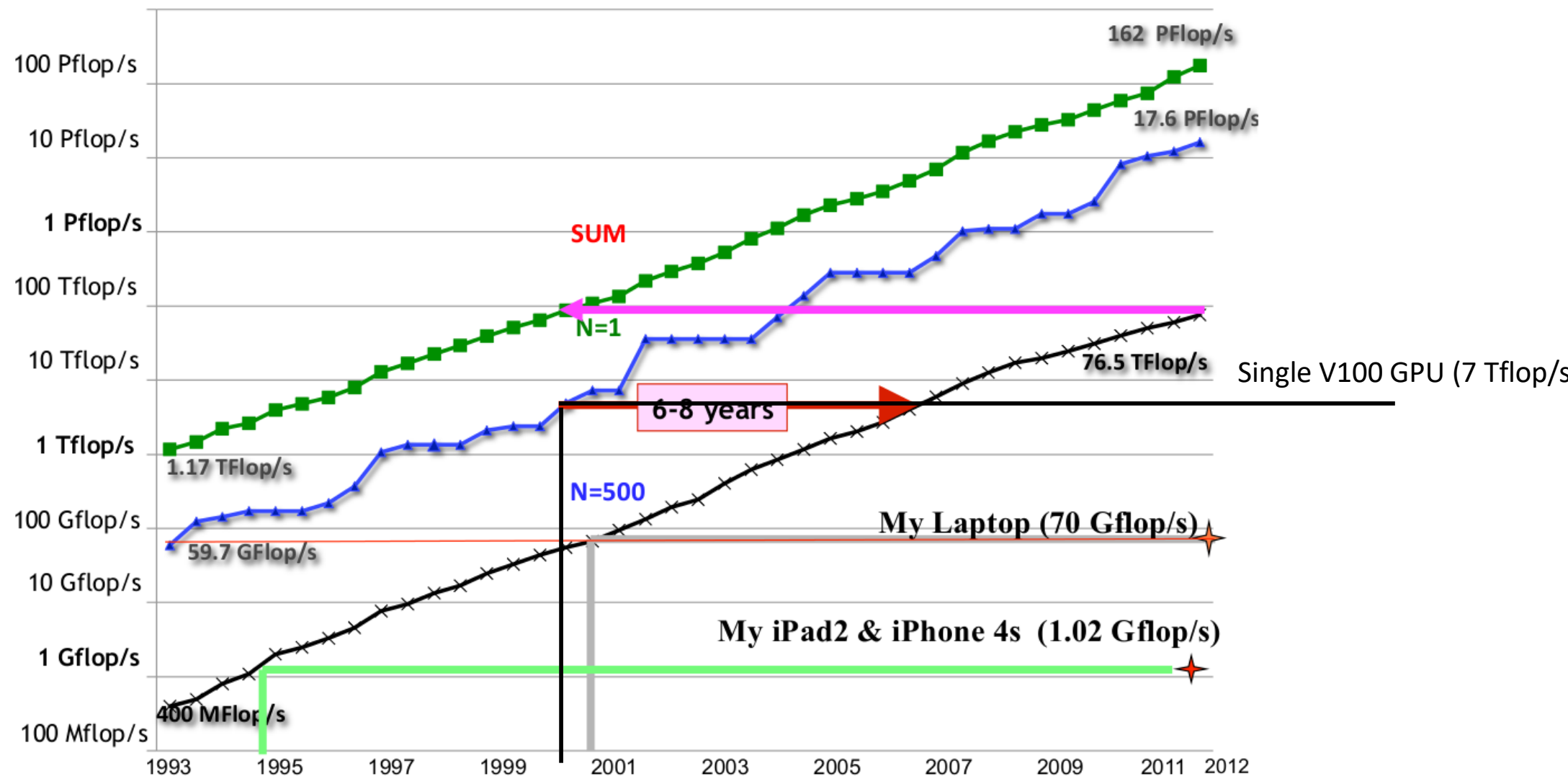


This is why you need to understand performance expectations well!

# Blue Waters in 2012



# History and Trends



Source: Jack Dongarra

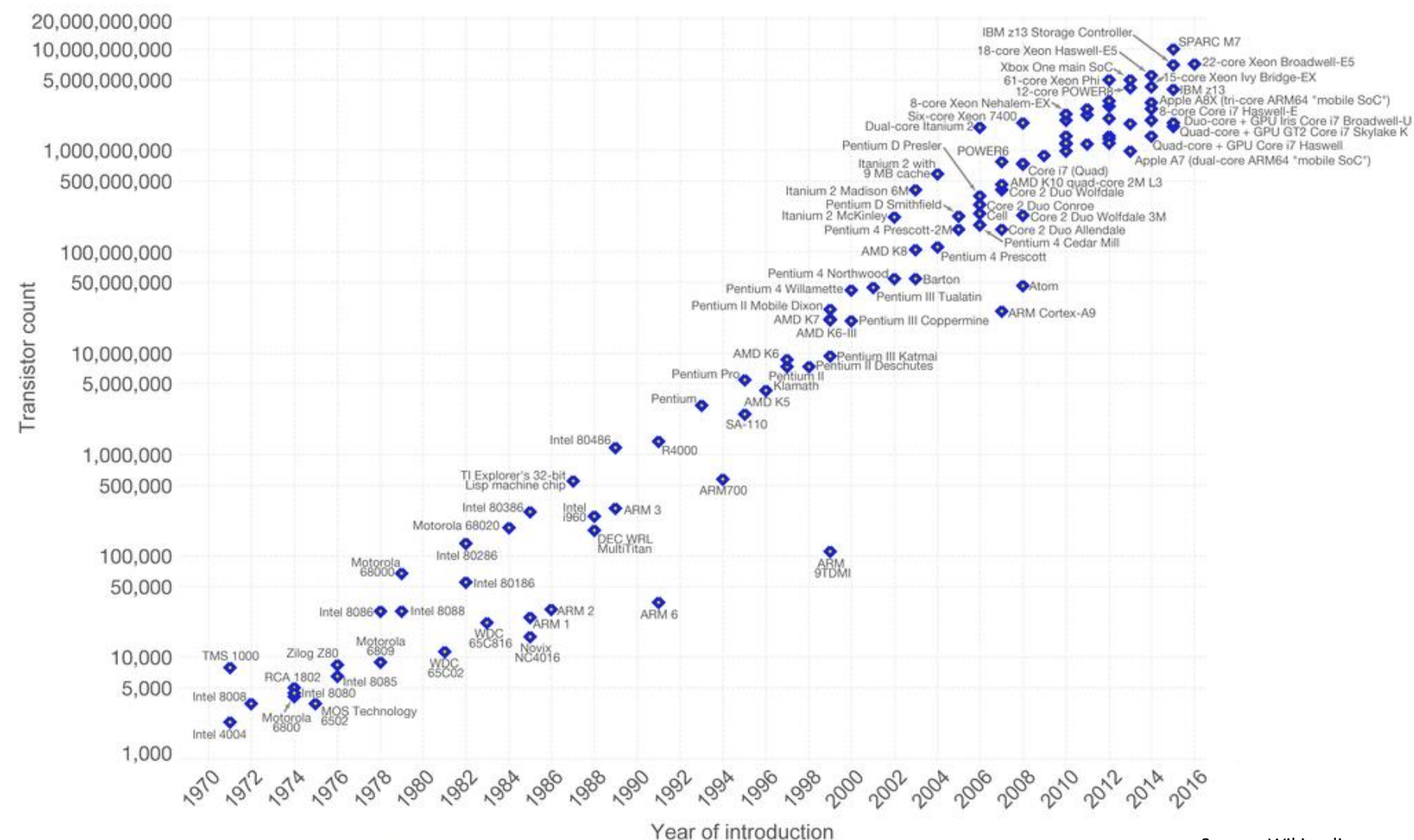




# Moore's Law – The number of transistors on integrated circuit chips (1971-2016)



Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are strongly linked to Moore's law.



Data source: Wikipedia ([https://en.wikipedia.org/wiki/Transistor\\_count](https://en.wikipedia.org/wiki/Transistor_count))

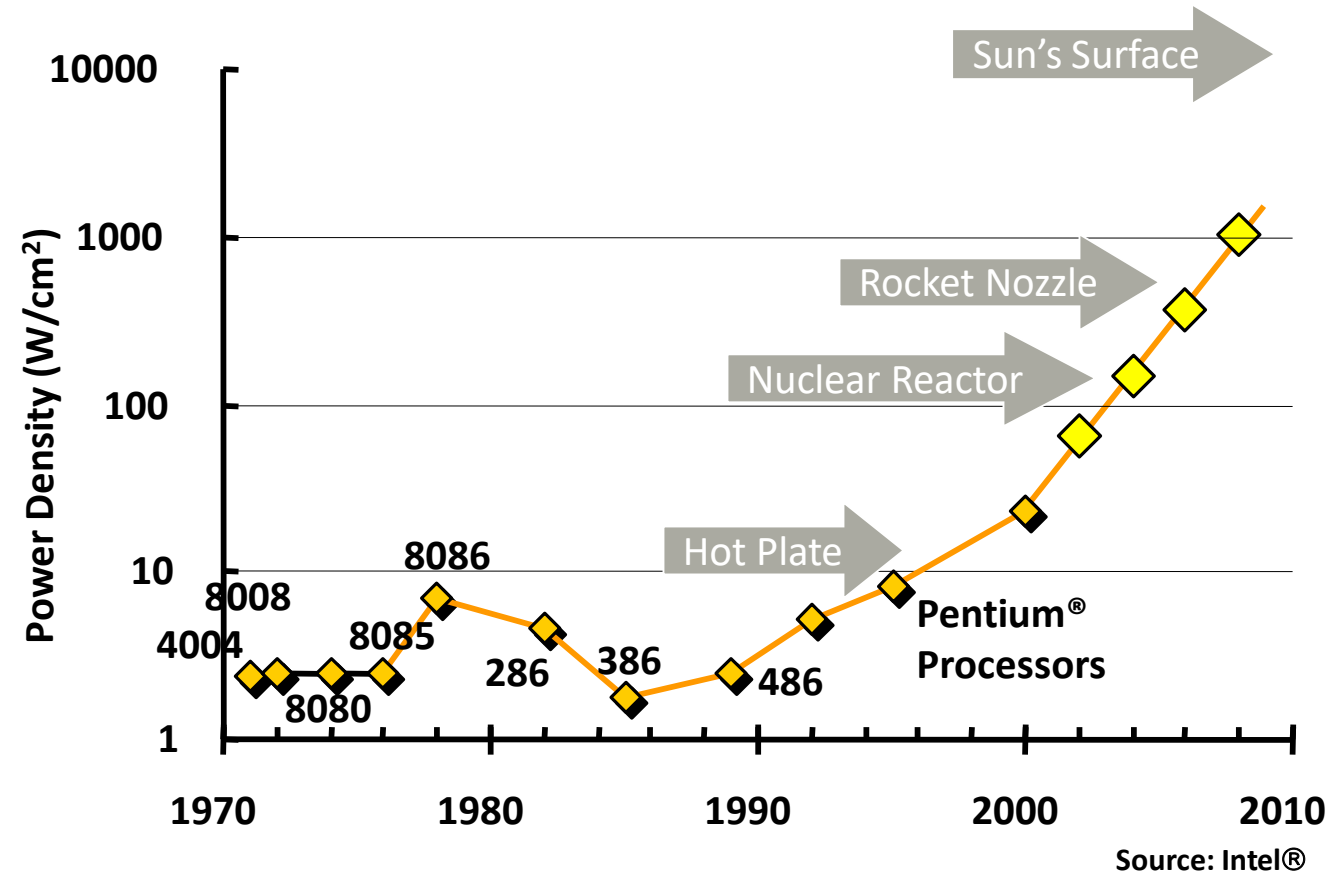
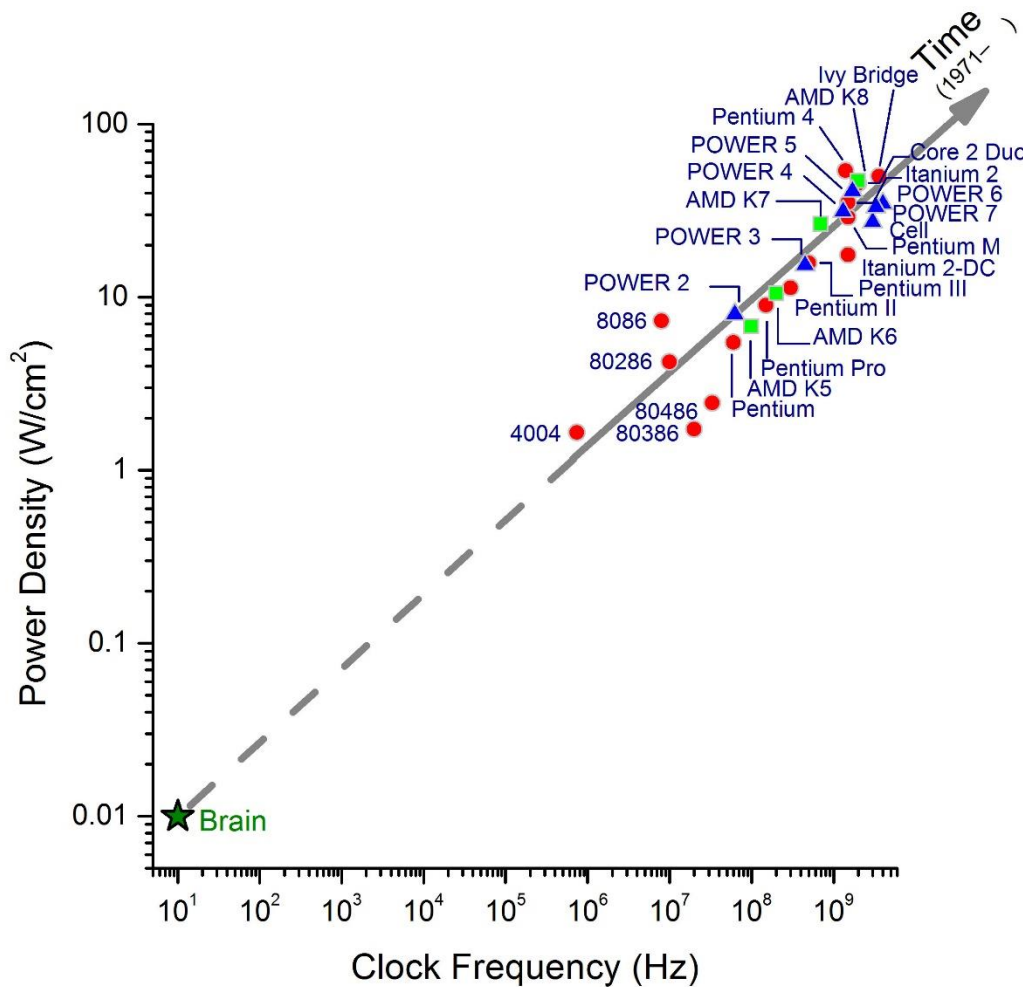
The data visualization is available at [OurWorldinData.org](http://OurWorldinData.org). There you find more visualizations and research on this topic.

Source: Wikipedia

# How to increase the compute power?

Not an option anymore!

~~Clock Speed:~~



# Computer Architecture vs. Physics (currently 0:1)

## ■ Physics (technological constraints)

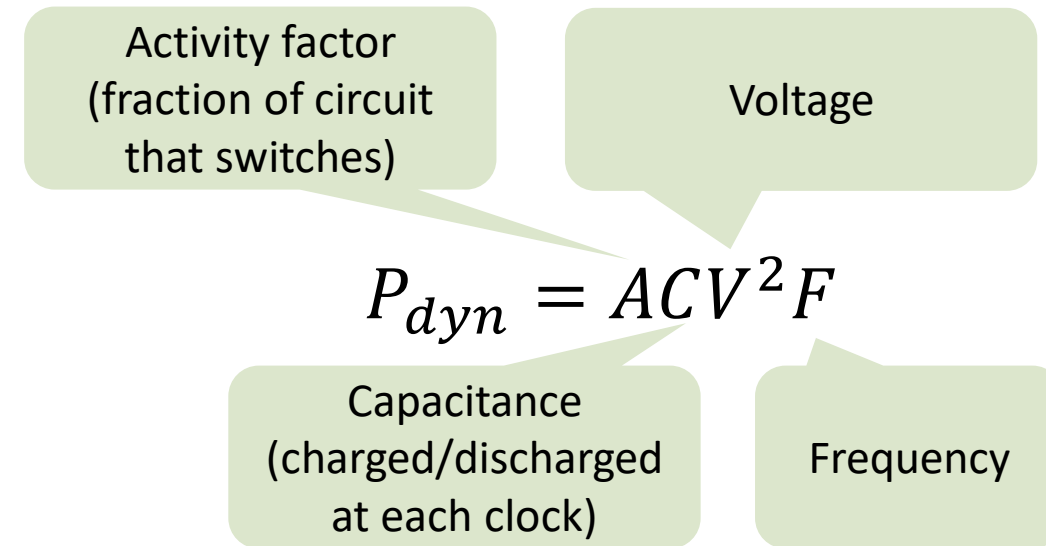
- Cost of data movement
- Capacity of DRAM cells
- Clock frequencies (constrained by end of Dennard scaling)
- Speed of Light
- Melting point of silicon

## ■ Computer Architecture (design of the machine)

- **Power management**
- ISA / Multithreading
- SIMD widths

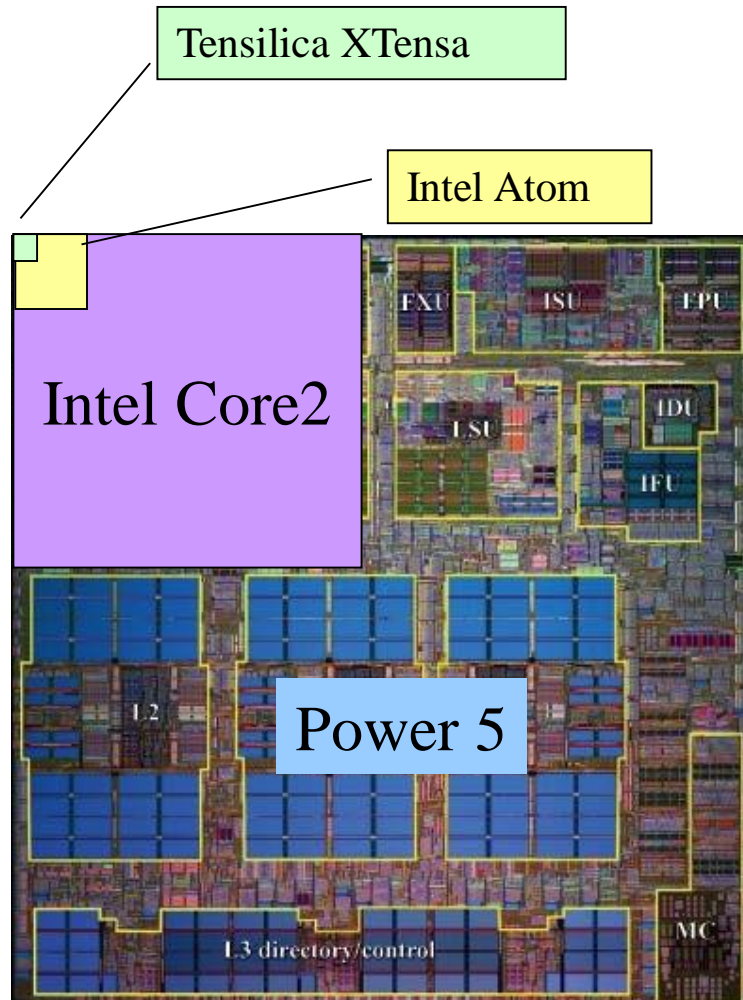
“Computer architecture, like other architecture, is the art of determining the needs of the user of a structure and then designing to meet those needs as effectively as possible within economic and technological constraints.” – *Fred Brooks (IBM, 1962)*

**Have converted many former “power” problems into “cost” problems**



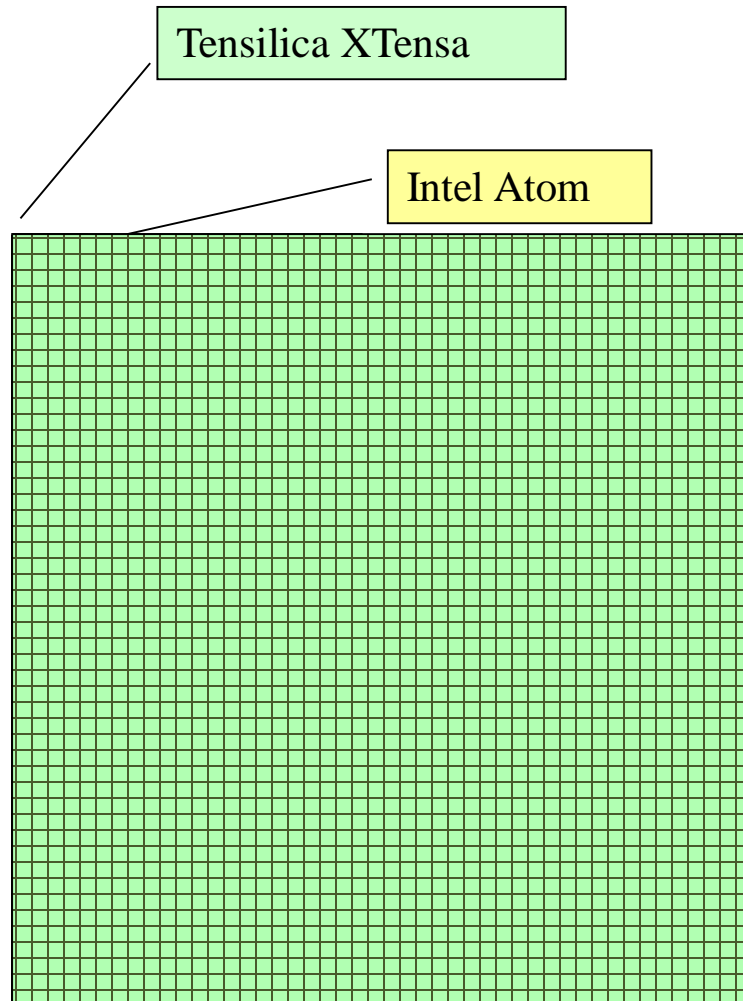
Higher voltage is needed to drive higher frequency (due to fixed capacitance). Higher voltage also increases static power dissipation (leakage).

# Low-Power Design Principles (2005)



- Cubic power improvement with lower clock rate due to  $V^2F$   
↓
- Slower clock rates enable use of simpler cores  
↓
- Simpler cores use less area (lower leakage) and reduce cost  
↓
- Tailor design to application to **REDUCE WASTE**

# Low-Power Design Principles (2005)

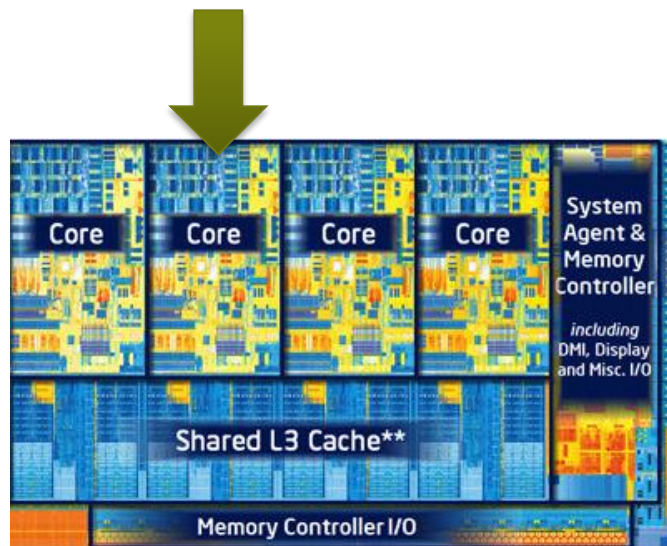


- **Power5 (server)**
  - 120W@1900MHz
  - **Baseline**
- **Intel Core2 sc (laptop) :**
  - 15W@1000MHz
  - *4x more FLOPs/watt than baseline*
- **Intel Atom (handhelds)**
  - 0.625W@800MHz
  - **80x more**
- **GPU Core or XTensa/Embedded**
  - 0.09W@600MHz
  - **400x more (80x-120x sustained)**

**Even if each simple core is 1/4th as computationally efficient as complex core, you can fit hundreds of them on a single chip and still be 100x more power efficient.**

# Heterogeneous computing on the rise!

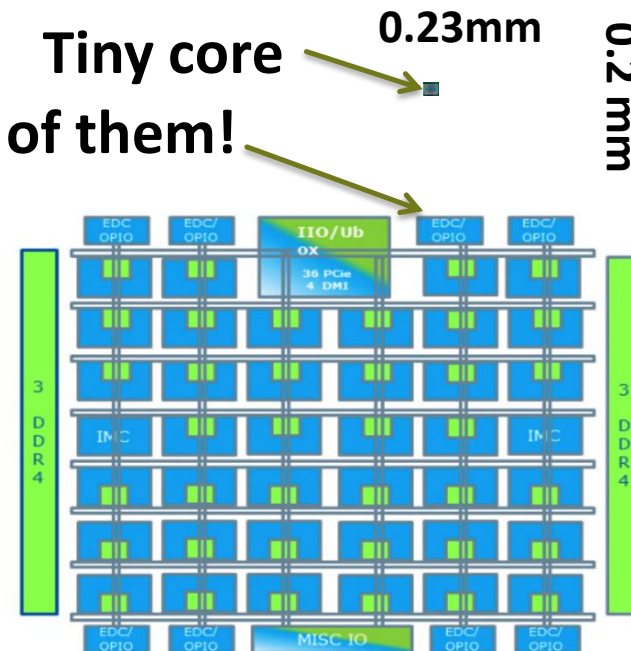
Big cores (very few)



## Latency Optimized Core (LOC)

Most energy efficient if you don't have lots of parallelism

Tiny core  
Lots of them!



## Throughput Optimized Core (TOC)

Most energy efficient if you DO have a lot of parallelism!

# Data movement – the wires

- Energy Efficiency of copper wire:
  - Power = Frequency \* Length / cross-section-area

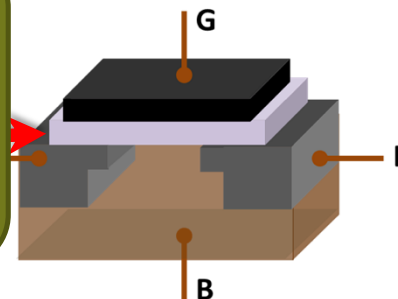


- Wire efficiency *does not improve* as feature size shrinks

- Energy Efficiency
  - Power =  $V^2 * \text{freq}$
  - Capacitance  $\sim A$
  - Transistor efficiency improves as you shrink it

Photonics could break through the bandwidth-distance limit

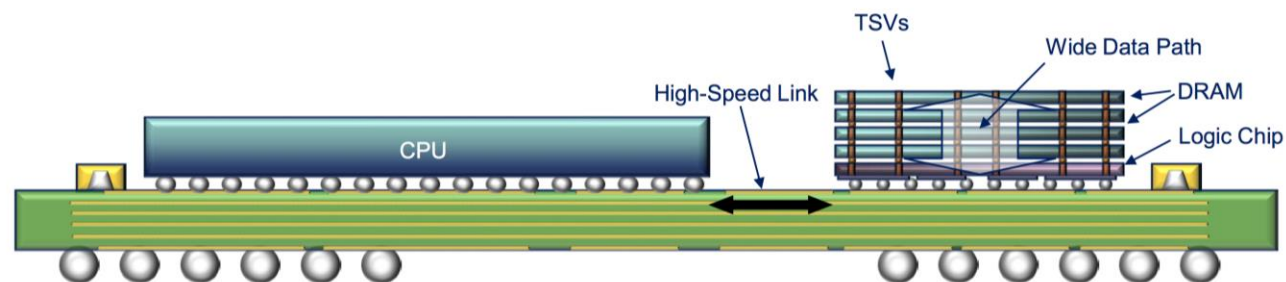
MOS Transistor



- Net result is that moving data on wires is starting to cost more energy than computing on said data (*interest in Silicon Photonics*)

# Pin Limits

- **Moore's law doesn't apply to adding pins to package**
  - 30%+ per year nominal Moore's Law
  - Pins grow at ~1.5-3% per year at best
- **4000 Pins is aggressive pin package**
  - Half of those would need to be for power and ground
  - Of the remaining 2k pins, run as differential pairs
  - Beyond 15Gbps per pin power/complexity costs hurt!
  - 10Gpbs \* 1k pins is ~1.2TBytes/sec
- **2.5D Integration gets boost in pin density**
  - But it's a 1 time boost (how much headroom?)
  - 4TB/sec? (maybe 8TB/s with single wire signaling?)



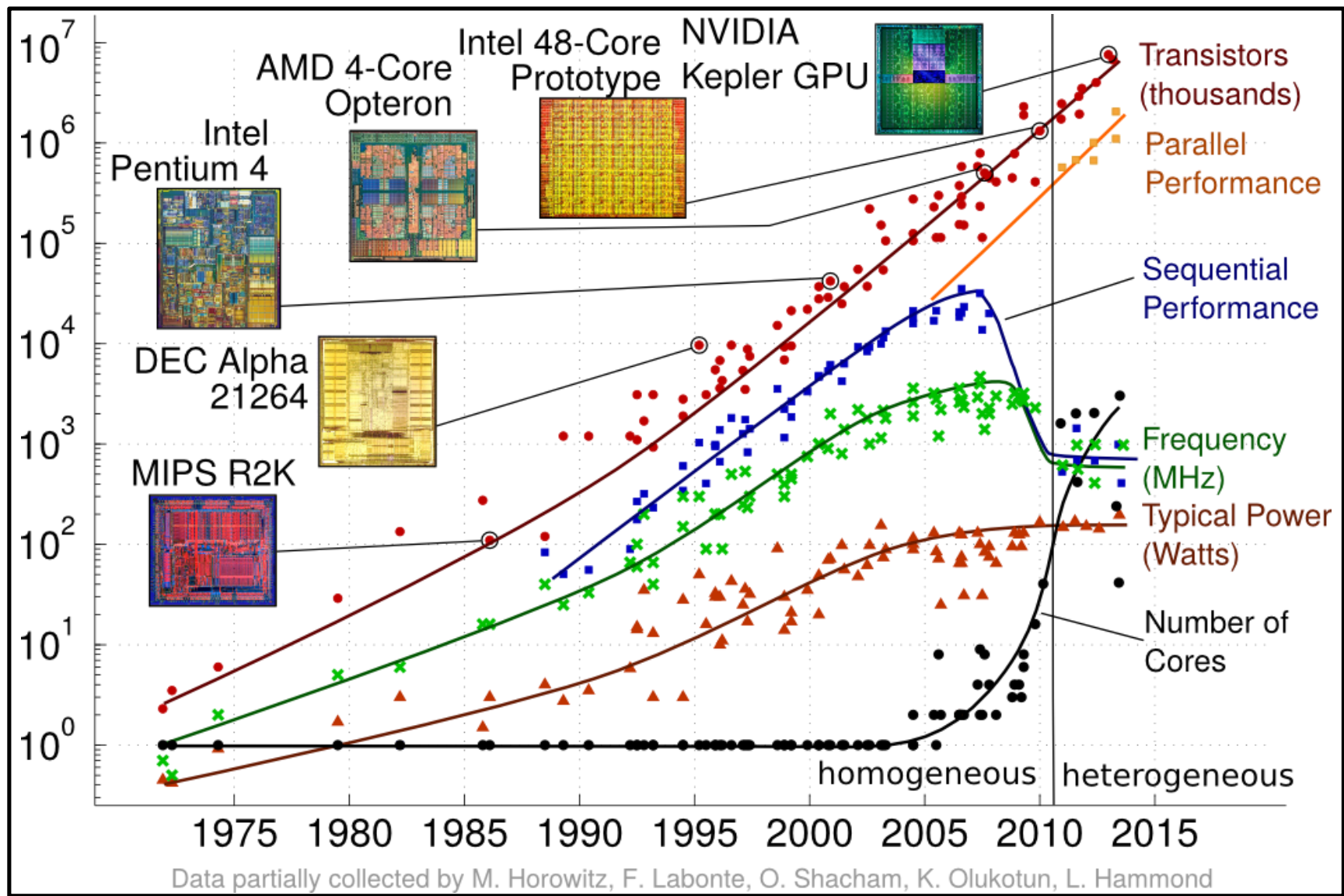


# The future?

- **Open-source CPUs?**
  - RISC-V
- **Open-source accelerators?**
  - Talk to us if interested!
  - Context of the European Processor Initiative  
*Collaboration with L. Benini (ITET)*
- **Many open research topics**
  - How to program hardware?
  - How to combine IPs into a system
  - How to build real high-performance CPUs/systems/accelerators!

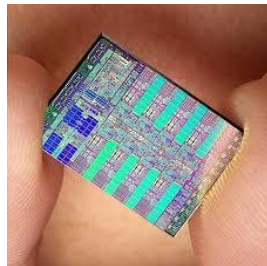


# A more complete view



# So how to invest the transistors?

- **Architectural innovations**
  - Branch prediction, out-of-order logic/rename register, speculative execution, ...
  - Help only so much 😞
- **What else?**
  - Simplification is beneficial, less transistors per CPU, more CPUs, e.g., Cell B.E., GPUs, MIC, Sunway SW26010
  - We call this “cores” these days
  - Also, more intelligent devices or higher bandwidths (e.g., DMA controller, intelligent NICs)



Source: IBM



Source: NVIDIA



Source: Intel

# Towards the age of massive parallelism

- **Everything goes parallel**
  - Desktop computers get more cores  
*2,4,8, soon dozens, hundreds?*  
*My watch has four (weak) cores ...*
  - Supercomputers get more PEs (cores, nodes)
    - > 10 million today
    - > 50 million on the horizon
    - 1 billion in a couple of years (after 2030?)
- **Parallel Computing is inevitable!**

## *Parallel vs. Concurrent computing*

Concurrent activities *may* be executed in parallel

Example:

A1 starts at T1, ends at T2; A2 starts at T3, ends at T4

Intervals (T1,T2) and (T3,T4) may overlap!

Parallel activities:

A1 is executed *while* A2 is running

Usually requires separate resources!

# Goals of this lecture

- Motivate you!
- What is parallel computing?
  - And why do we need it?
- What is high-performance computing?
  - What's a Supercomputer and why do we care?
- **Basic overview of**
  - Programming models
    - Some examples*
  - Architectures
    - Some case-studies*
- **Provide context for coming lectures**

# Granularity and Resources

## Execution Activities

- Micro-code instruction
- Machine-code instruction (complex or simple)
- Sequence of machine-code instructions:
  - Blocks*
  - Loops*
  - Loop nests*
  - Functions*
  - Function sequences*

## Parallel Resource

- Instruction-level parallelism
  - Pipelining
  - VLIW/EDGE
  - Superscalar
- SIMD operations
  - Vector operations
- Instruction sequences
  - Multiprocessors
  - Multicores
  - Multithreading

## Programming

- Compiler
  - (inline assembly)
  - Hardware scheduling
- Compiler (inline assembly)
- Libraries
- Compilers (very limited)
- Expert programmers
  - Parallel languages
  - Parallel libraries
  - Hints

# Historic Architecture Examples

## ■ Systolic Array

- Data-stream driven (data counters)
- Multiple streams for parallelism
- Specialized for applications (reconfigurable)

## ■ Dataflow Architectures

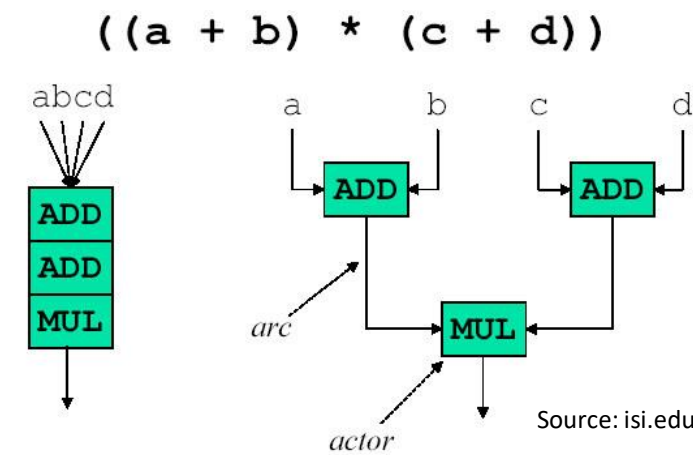
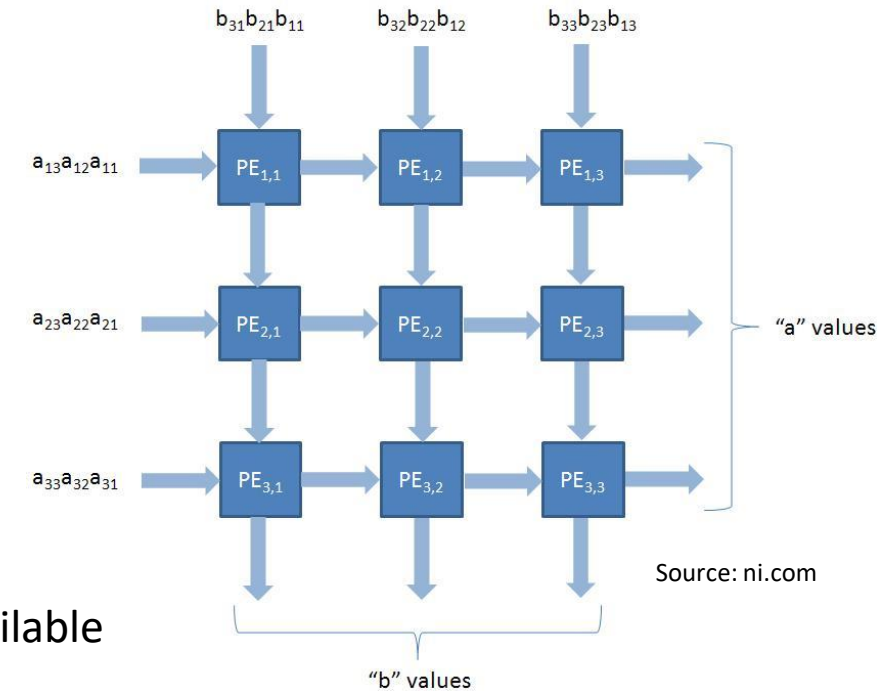
- No program counter, execute instructions when all input arguments are available
- Fine-grained, high overheads

*Example: compute  $f = (a+b) * (c+d)$*

- Both come-back in FPGA computing and EDGE architectures

- Interesting research opportunities!

*Talk to us if you're interested (i.e., how to program FPGAs easily and fast)*



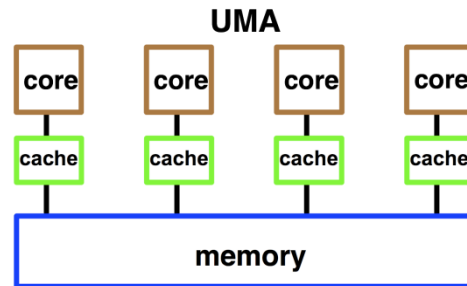
# Von Neumann Architecture (default today)

- **Program counter → inherently sequential!**  
Retrospectively define parallelism in instructions and data

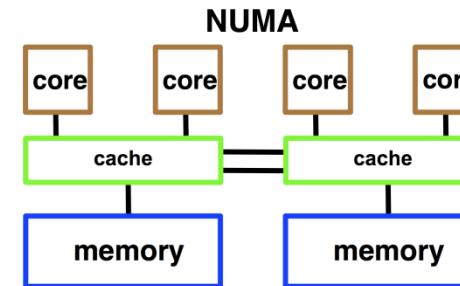
<p><b>SISD</b> Standard Serial Computer (nearly extinct)</p>	<p><b>SIMD</b> Vector Machines or Extensions (very common)</p>
<p><b>MISD</b> Redundant Execution (fault tolerance)</p>	<p><b>MIMD</b> Multicore (ubiquitous)</p>



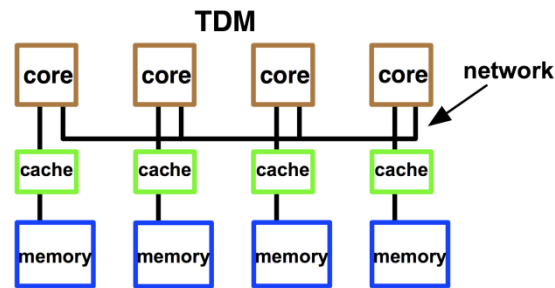
# Parallel Architectures 101 – Multiple Instruction Streams



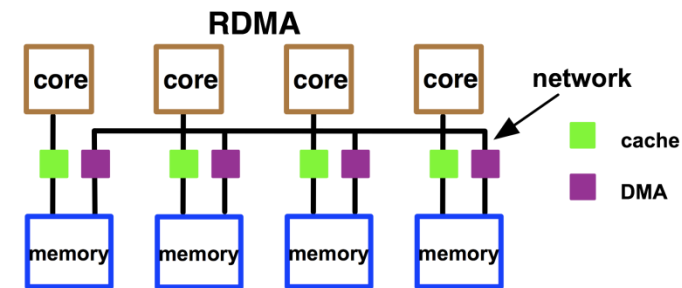
Today's laptops



Today's servers



Yesterday's clusters

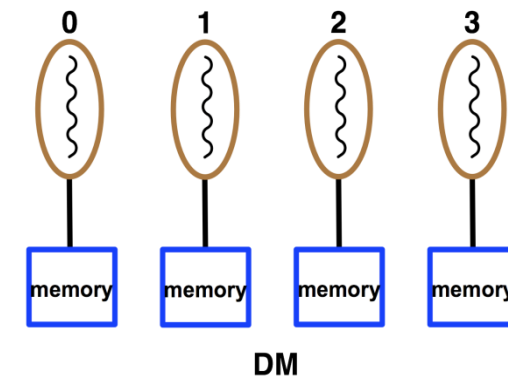
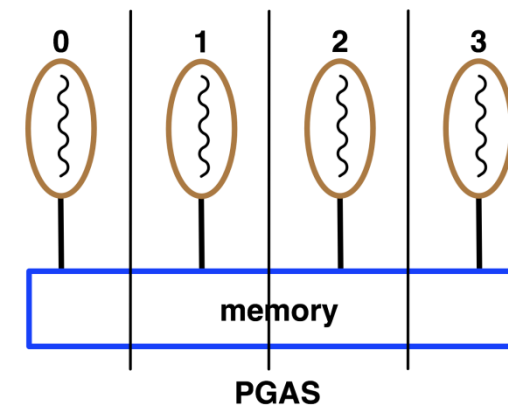
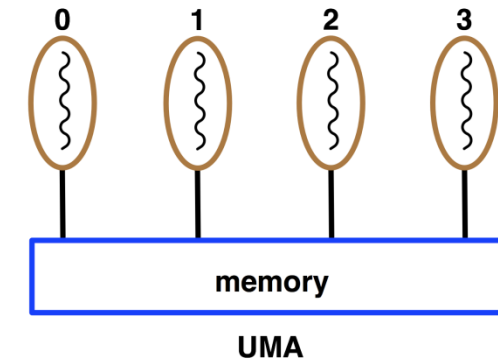


Today's clusters

- ... and mixtures of those

# Parallel Programming Models 101

- **Shared Memory Programming (SM/UMA)**
  - Shared address space
  - Implicit communication
  - Hardware for cache-coherent remote memory access
  - Cache-coherent Non Uniform Memory Access (cc NUMA)
- **(Partitioned) Global Address Space (PGAS)**
  - Remote Memory Access
  - Remote vs. local memory (cf. ncc-NUMA)
- **Distributed Memory Programming (DM)**
  - Explicit communication (typically messages)
  - Message Passing



# Shared Memory Machines

- Two historical architectures:

- “Mainframe” – all-to-all connection between memory, I/O and PEs

*Often used if PE is the most expensive part*

*Bandwidth scales with P*

*PE Cost scales with P, Question: what about network cost?*

*Answer: P<sup>2</sup>, can be cut with multistage connections (butterfly)*

- “Minicomputer” – bus-based connection

*All traditional SMP systems*

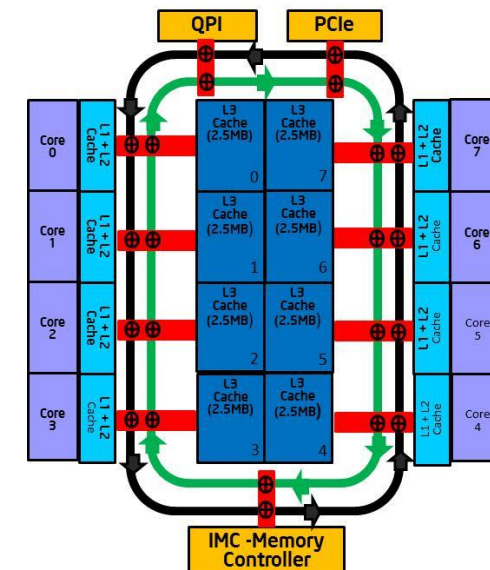
*High latency, low bandwidth (cache is important)*

*Tricky to achieve highest performance (contention)*

*Low cost, extensible*

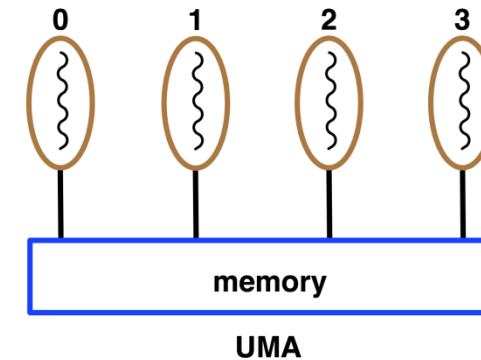


Source: IBM



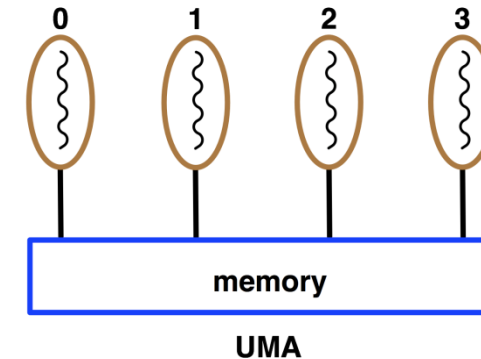
# Shared Memory Machine Abstractions

- **Any PE can access all memory**
  - Any I/O can access all memory (maybe limited)
- **OS (resource management) can run on any PE**
  - Can run multiple threads in shared memory
  - Used since 40+ years
- **Communication through shared memory**
  - Load/store commands to memory controller
  - Communication is implicit
  - Requires coordination
- **Coordination through shared memory**
  - Complex topic
  - Memory models
- **(ETH students): Most of what we covered in Parallel Programming in the 2<sup>nd</sup> semester!**



# Shared Memory Machine Programming

- **Threads or processes**
  - Communication through memory
- **Synchronization through memory or OS objects**
  - Lock/mutex (protect critical region)
  - Semaphore (generalization of mutex (binary sem.))
  - Barrier (synchronize a group of activities)
  - Atomic Operations (CAS, Fetch-and-add)
  - Transactional Memory (execute regions atomically)
- **Practical Models:**
  - Posix threads (ugs, will see later)
  - MPI-3
  - OpenMP
  - Others: Java Threads, Qthreads, ...
- **(ETH students): Most of what we covered in Parallel Programming in the 2<sup>nd</sup> semester!**

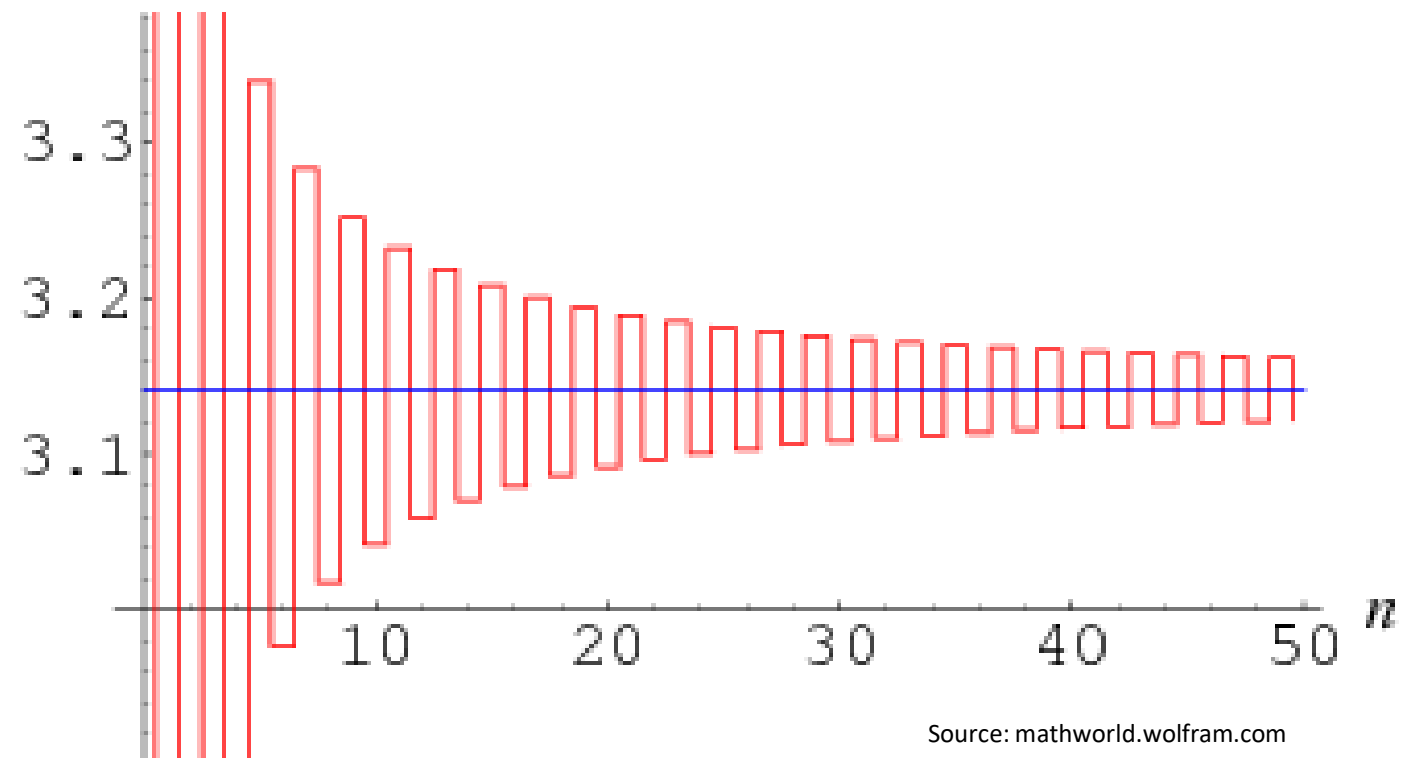


# An SMM Example: Compute Pi

- Using Gregory-Leibnitz Series:

$$4 \sum_{k=1}^n \frac{(-1)^{k+1}}{2k-1}$$

- Iterations of sum can be computed in parallel
- Needs to sum all contributions at the end



Source: mathworld.wolfram.com

# Pthreads Compute Pi Example

```
int n=10000;
double *resultarr;
pthread_t *thread_arr;
int nthreads;

void *compute_pi(void *data) {
    int i, j;
    int myid = (int)(long)data;
    double mypi, h, x, sum;

    for (j=0; j<n; ++j) {
        h = 1.0 / (double) n;
        sum = 0.0;
        for (i = myid + 1; i <= n; i += nthreads) {
            x = h * ((double)i - 0.5);
            sum += (4.0 / (1.0 + x*x));
        }
        mypi = h * sum;
    }
    resultarr[myid] = mypi;
}
```

```
int main( int argc, char *argv[] )
{
    // definitions ...
    thread_arr = (pthread_t*)malloc(nthreads *
                                    sizeof(pthread_t));
    resultarr= (double*)malloc(nthreads *
                                sizeof(double));

    for (i=0; i<nthreads; ++i) {
        int ret = pthread_create( &thread_arr[i],
                                  NULL, compute_pi, (void*) i);
    }
    for (i=0; i<nthreads; ++i) {
        pthread_join( thread_arr[i], NULL);
    }
    pi = 0;
    for (i=0; i<nthreads; ++i) pi += resultarr[i];

    printf ("pi is ~%.16f, Error is %.16f\n",
           pi, fabs(pi - PI25DT));
}
```

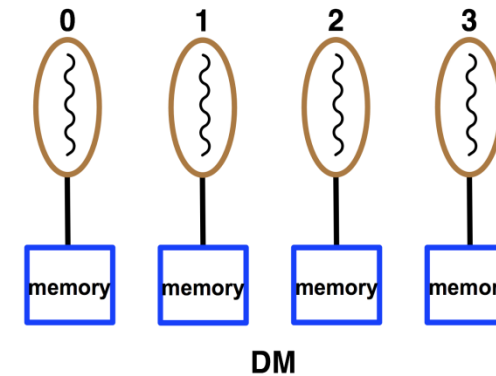
## Additional comments on SMM

- OpenMP would allow to implement this example much simpler (but has other issues)
- Transparent shared memory has some issues in practice:
  - False sharing (e.g., resultarr[])
  - Race conditions (complex mutual exclusion protocols)
  - Little tool support (debuggers need some work)
  - These issues were predominantly discussed in parallel programming in the 2<sup>nd</sup> semester  
*We will briefly repeat some but not all!*
- ***Achieving performance is harder than it seems!***

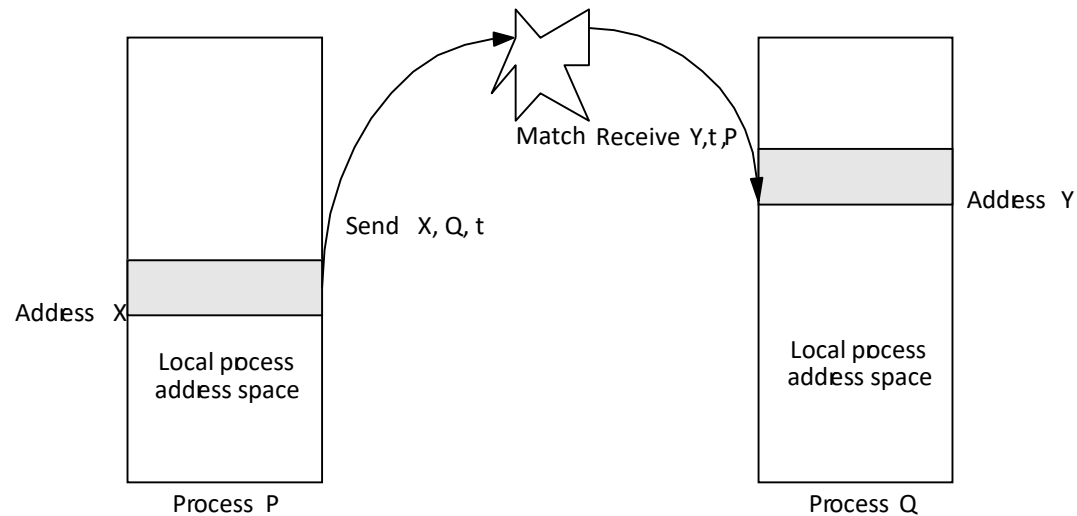


# Distributed Memory Machine Programming

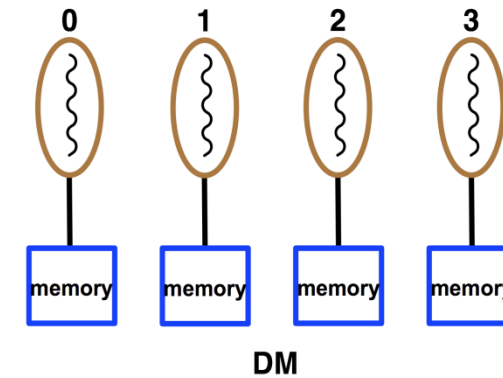
- **Explicit communication between PEs**
  - Message passing or channels
- **Only local memory access, no direct access to remote memory**
  - No shared resources (well, the network)
- **Programming model: Message Passing (MPI)**
  - Communication through messages or group operations (broadcast, reduce, etc.)
  - Synchronization through messages (sometimes unwanted side effect) or group operations (barrier)
  - Typically supports message matching and communication contexts



# DMM Example: Message Passing



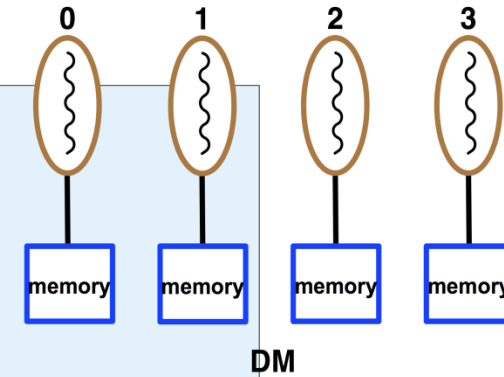
Source: John Mellor-Crummey



- Send specifies buffer to be transmitted
- Recv specifies buffer to receive into
- Implies copy operation between named PEs
- Optional tag matching
- Pair-wise synchronization (cf. happens before)

# DMM MPI Compute Pi Example

```
int main( int argc, char *argv[] ) {  
    // definitions  
    MPI_Init(&argc,&argv);  
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);  
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);  
  
    double t = -MPI_Wtime();  
    for (j=0; j<n; ++j) {  
        h = 1.0 / (double) n;  
        sum = 0.0;  
        for (i = myid + 1; i <= n; i += numprocs) { x = h * ((double)i - 0.5); sum += (4.0 / (1.0 + x*x)); }  
        mypi = h * sum;  
        MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);  
    }  
    t+=MPI_Wtime();  
  
    if (!myid) {  
        printf("pi is approximately %.16f, Error is %.16f\n", pi, fabs(pi - PI25DT));  
        printf("time: %f\n", t);  
    }  
  
    MPI_Finalize();  
}
```



## DMM Example: PGAS

- **Partitioned Global Address Space**

- Shared memory emulation for DMM

*Usually non-coherent*

- “Distributed Shared Memory”

*Usually coherent*

- **Simplifies shared access to distributed data**

- Has similar problems as SMM programming
- Sometimes lacks performance transparency

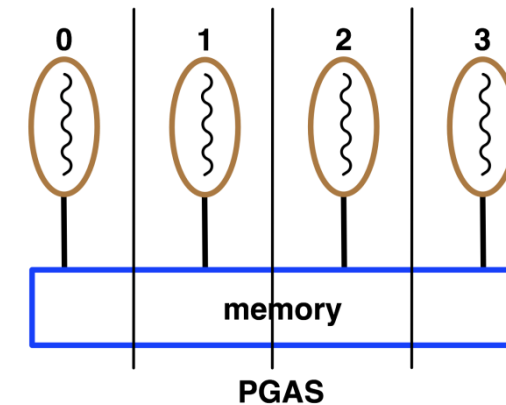
*Local vs. remote accesses*

- **Examples:**

- UPC, CAF, Titanium, X10, ...

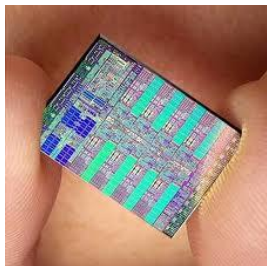
- **Interesting research question: how to exploit PGAS/RDMA in practice?**

- Cf. VLDB’17, Barthels et al.: “Distributed Join Algorithms on Thousands of Cores”



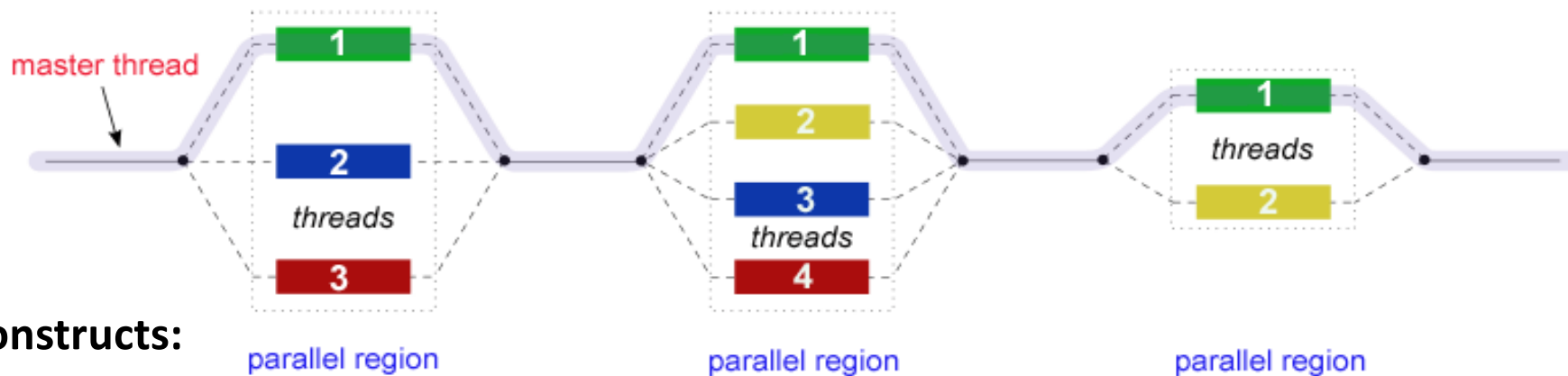
# How to Tame the Beast?

- **How to program large machines?**
- **No single approach, PMs are not converging yet**
  - MPI, PGAS, OpenMP, Hybrid (MPI+OpenMP, MPI+MPI, MPI+PGAS?, generally MPI+X), ...
- **Architectures converge**
  - General purpose nodes connected by general purpose or specialized networks
  - Small scale often uses commodity networks
  - Specialized networks become necessary at scale
- **Even worse: accelerators (not covered in this class, yet)**

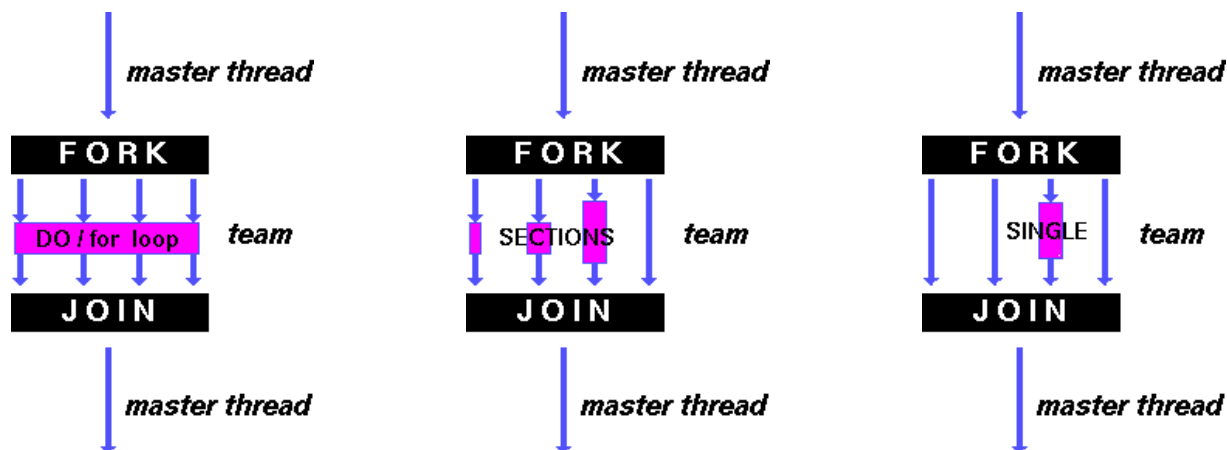


# Example: Shared Memory Programming with OpenMP

- Fork-join model



- Types of constructs:



+ Tasks

Source: Blaise Barney, LLNL

# Example: Shared Memory Programming with OpenMP

- Annotate sequential code with pragmas (introduce semantic duplication)

```
#include <omp.h>

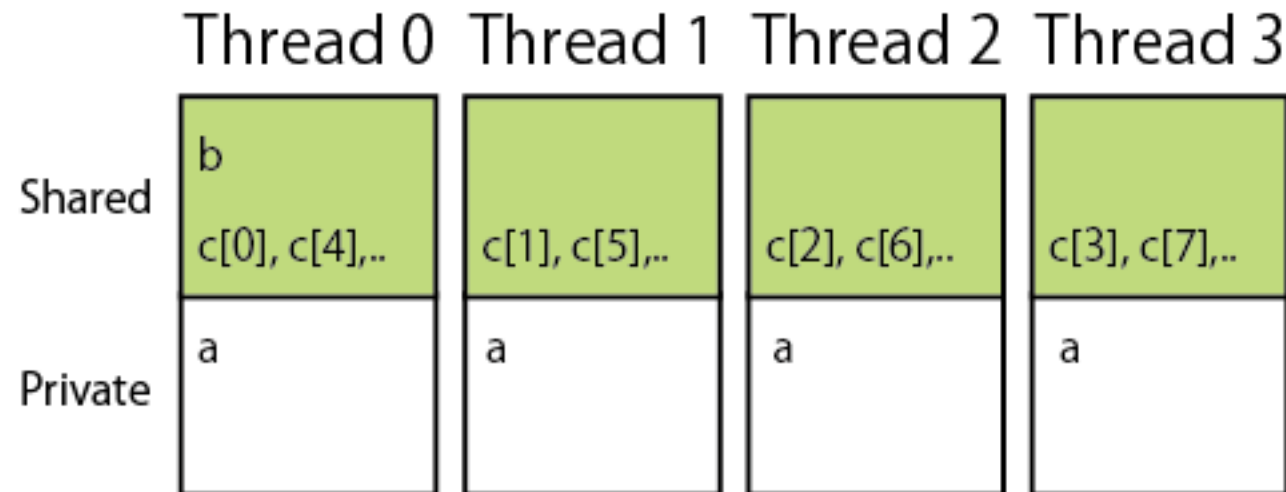
main () {
    int var1, var2, var3;
    // Serial code

    // Beginning of parallel section. Fork a team of threads. Specify variable scoping
    #pragma omp parallel private(var1, var2) shared(var3)
    {
        // Parallel section executed by all threads
        // Other OpenMP directives
        // Run-time Library calls
        // All threads join master thread and disband
    }
    // Resume serial code
}
```

Source: Blaise Barney, LLNL

# Example: Practical PGAS Programming with UPC

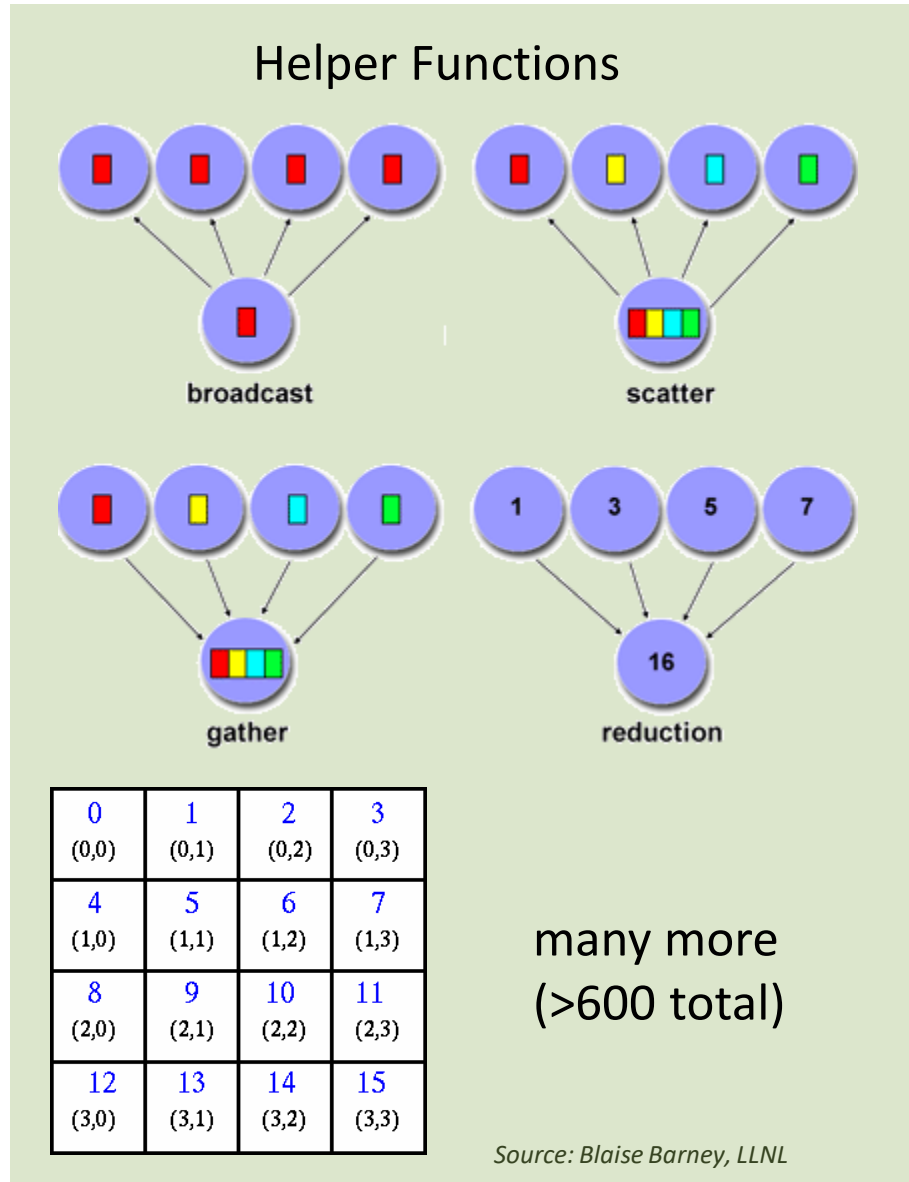
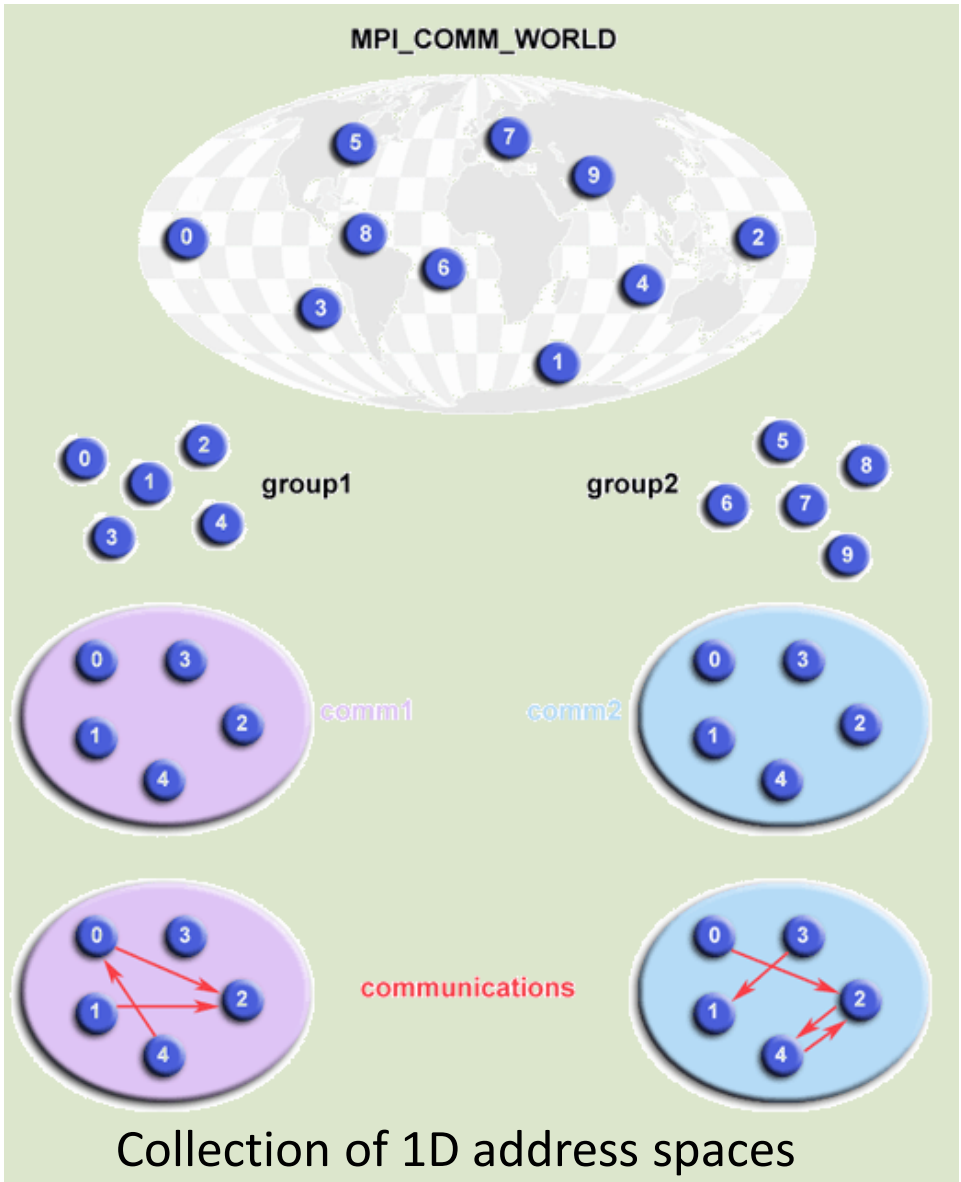
- PGAS extension to the C99 language



- Many helper library functions**
  - Collective and remote allocation
  - Collective operations
- Complex consistency model**



# Example: Practical Distributed Memory Programming: MPI-1



# Example: Practical Distributed Memory Programming: MPI-1 – Six Functions!

```
#include <mpi.h>

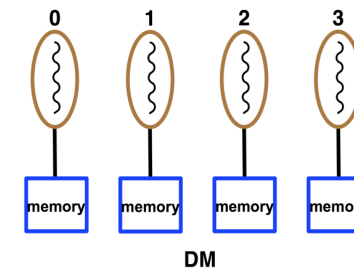
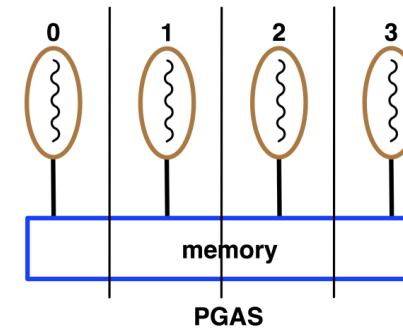
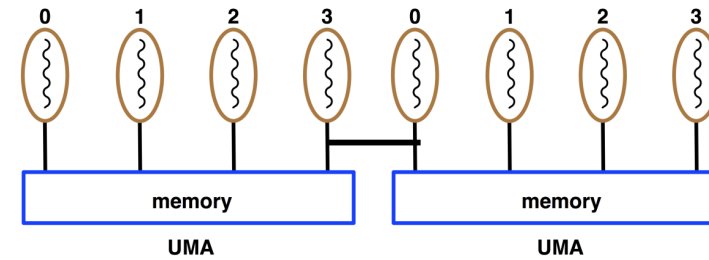
int main(int argc, char **argv) {
    int myrank, sbuf=23, rbuf=32;
    MPI_Init(&argc, &argv);

    /* Find out my identity in the default communicator */
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    if (myrank == 0) {
        MPI_Send(&sbuf,          /* message buffer */
                1,              /* one data item */
                MPI_INT,        /* data item is an integer */
                rank,           /* destination process rank */
                99,             /* user chosen message tag */
                MPI_COMM_WORLD); /* default communicator */
    } else {
        MPI_Recv(&rbuf, MPI_DOUBLE, 0, 99, MPI_COMM_WORLD, &status);
        printf("received: %i\n", rbuf);
    }

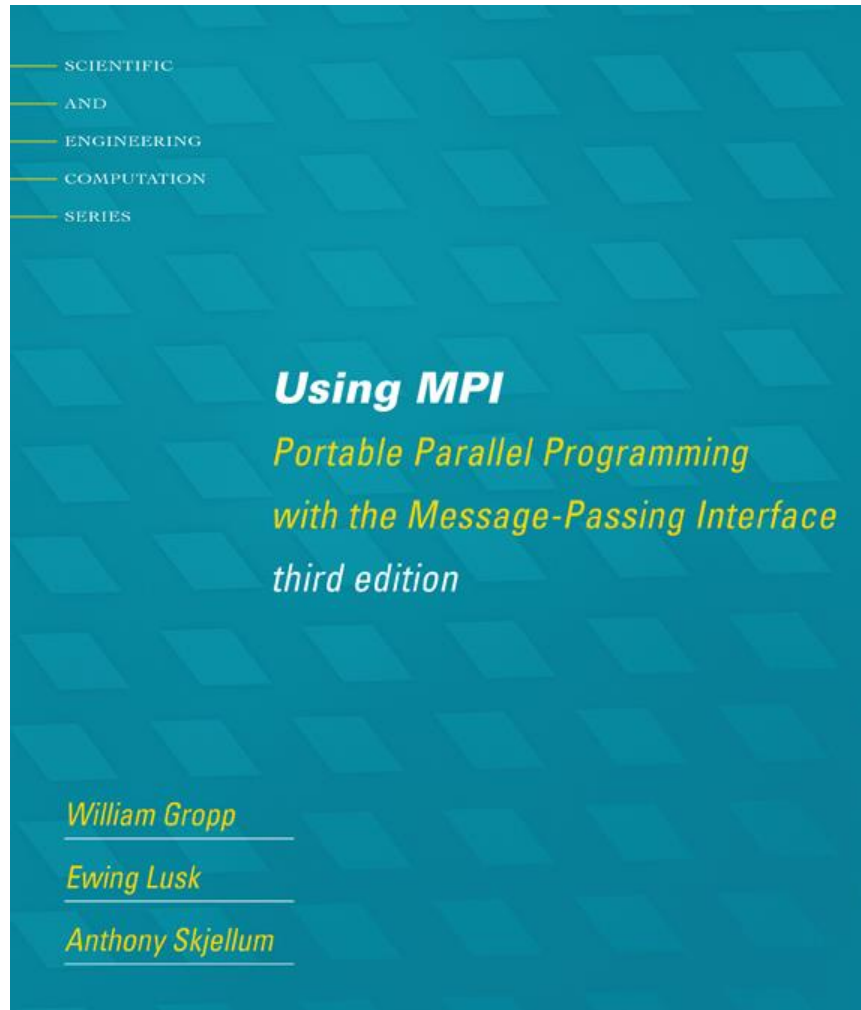
    MPI_Finalize();
}
```

# Example: MPI-2/3 supporting Shared Memory and PGAS-style!

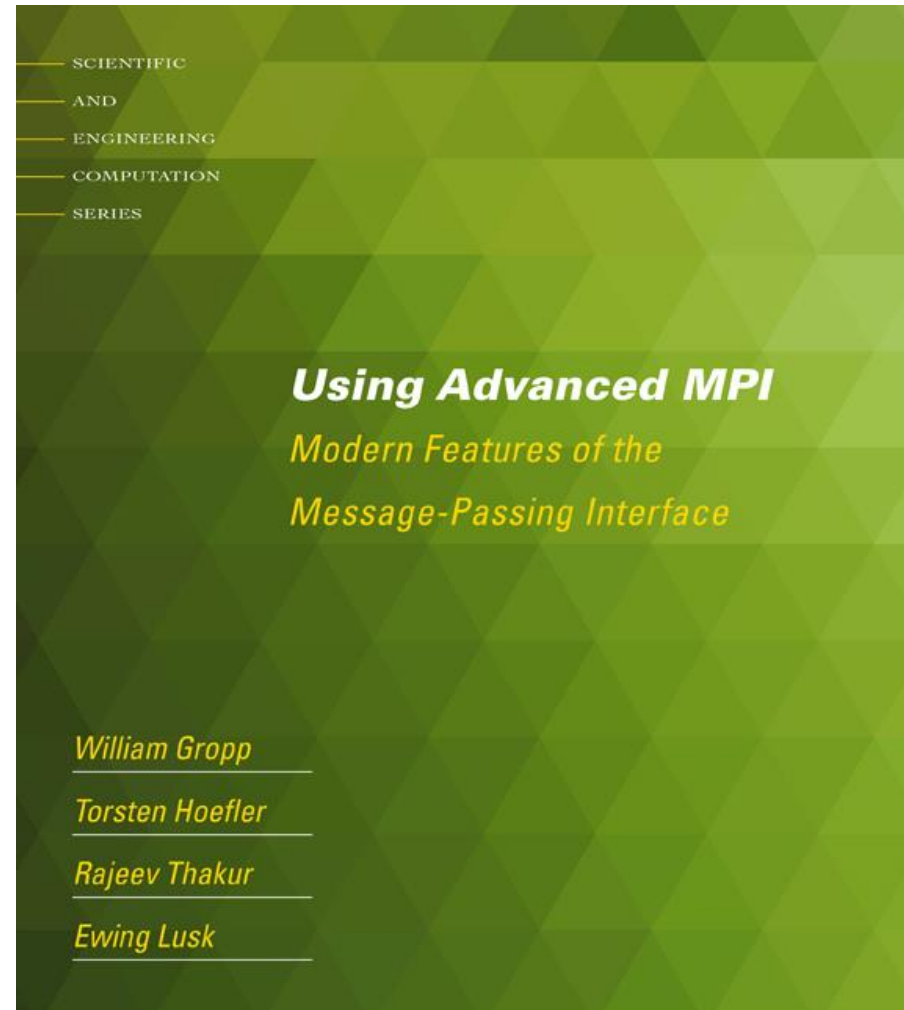
- Support for shared memory in SMM domains
  - Support for Remote Memory Access Programming
    - Direct use of RDMA
    - Essentially PGAS
- Enhanced support for message passing communication
  - Scalable topologies
  - More nonblocking features
  - ... many more



# MPI: de-facto large-scale prog. standard



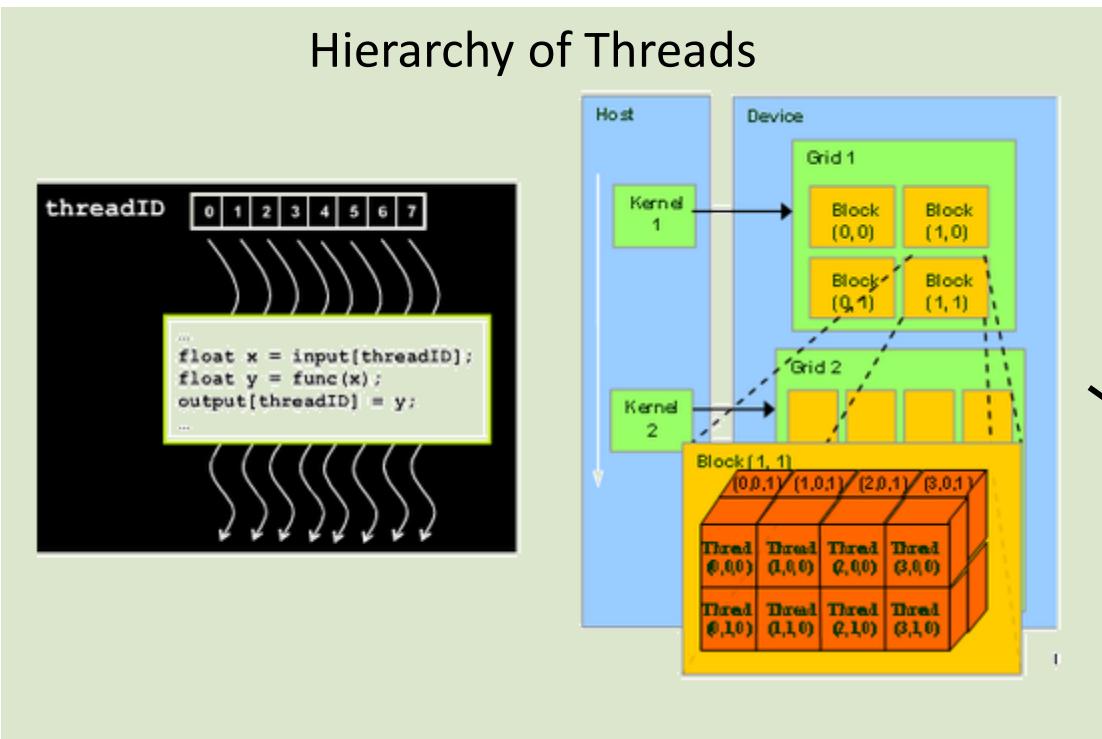
**Basic MPI**



**Advanced MPI, including MPI-3**

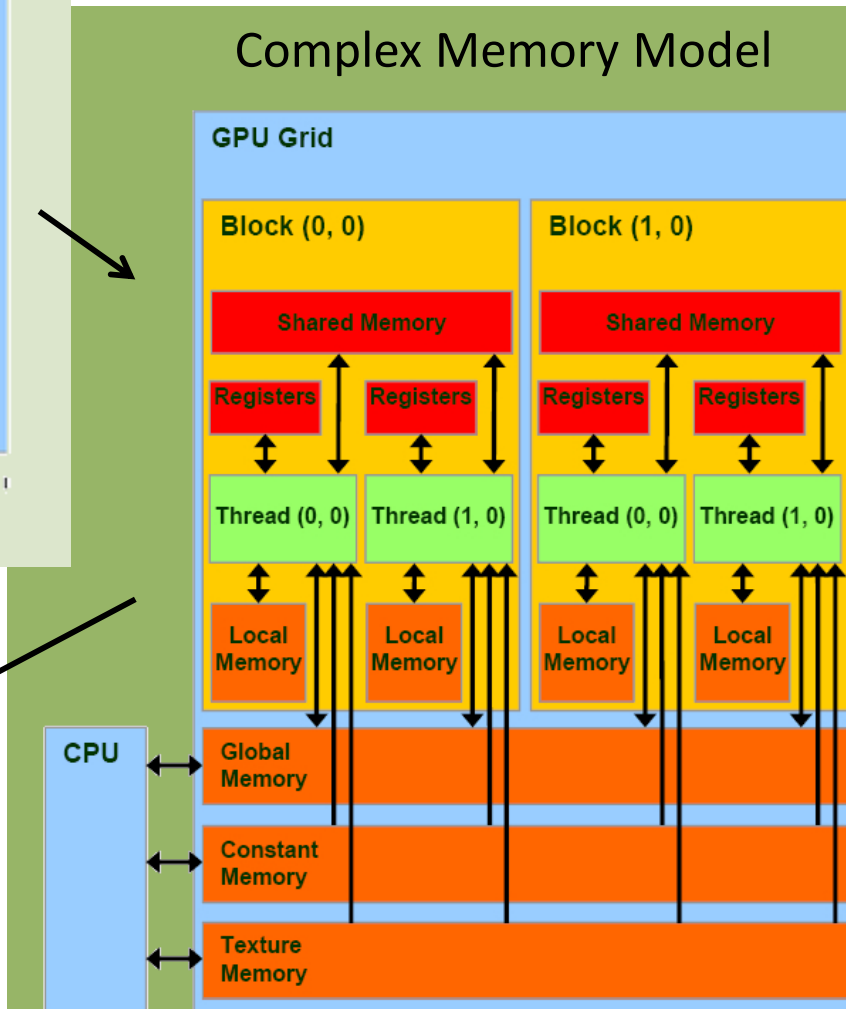
# Example: Accelerator programming with CUDA

## Hierarchy of Threads



Source: NVIDIA

## Complex Memory Model



## Simple Architecture

### Kepler Block Diagram

- 8 SMX
- 1536 CUDA Cores
- 8 Geometry Units
- 4 Raster Units
- 128 Texture Units
- 32 ROP units
- 256-bit GDDR5



# Example: Accelerator programming with CUDA

## Host Code

```
#define N 10
int main( void ) {
    int a[N], b[N], c[N];
    int *dev_a, *dev_b, *dev_c;
    // allocate the memory on the GPU
    cudaMalloc( (void**)&dev_a, N * sizeof(int) );
    cudaMalloc( (void**)&dev_b, N * sizeof(int) );
    cudaMalloc( (void**)&dev_c, N * sizeof(int) );
    // fill the arrays 'a' and 'b' on the CPU
    for (int i=0; i<N; i++) { a[i] = -i; b[i] = i * i; }
    // copy the arrays 'a' and 'b' to the GPU
    cudaMemcpy( dev_a, a, N * sizeof(int), cudaMemcpyHostToDevice );
    cudaMemcpy( dev_b, b, N * sizeof(int), cudaMemcpyHostToDevice );
    add<<<N,1>>>( dev_a, dev_b, dev_c );
    // copy the array 'c' back from the GPU to the CPU
    cudaMemcpy( c, dev_c, N * sizeof(int), cudaMemcpyDeviceToHost );
    // free the memory allocated on the GPU
    cudaFree( dev_a ); cudaFree( dev_b ); cudaFree( dev_c );
}
```

## The Kernel

```
__global__ void add( int *a, int *b, int *c ) {
    int tid = blockIdx.x;
    // handle the data at this index
    if (tid < N)
        c[tid] = a[tid] + b[tid];
}
```

## Example: OpenACC / OpenMP 4.0

- Aims to simplify GPU programming
- Compiler support
  - Annotations! More pragmas and semantic duplication

```
#define N 10
int main( void ) {
    int a[N], b[N], c[N];
    #pragma acc kernels
    for (int i = 0; i < N; ++i)
        c[i] = a[i] + b[i];
}
```

# Many many more programming models/frameworks

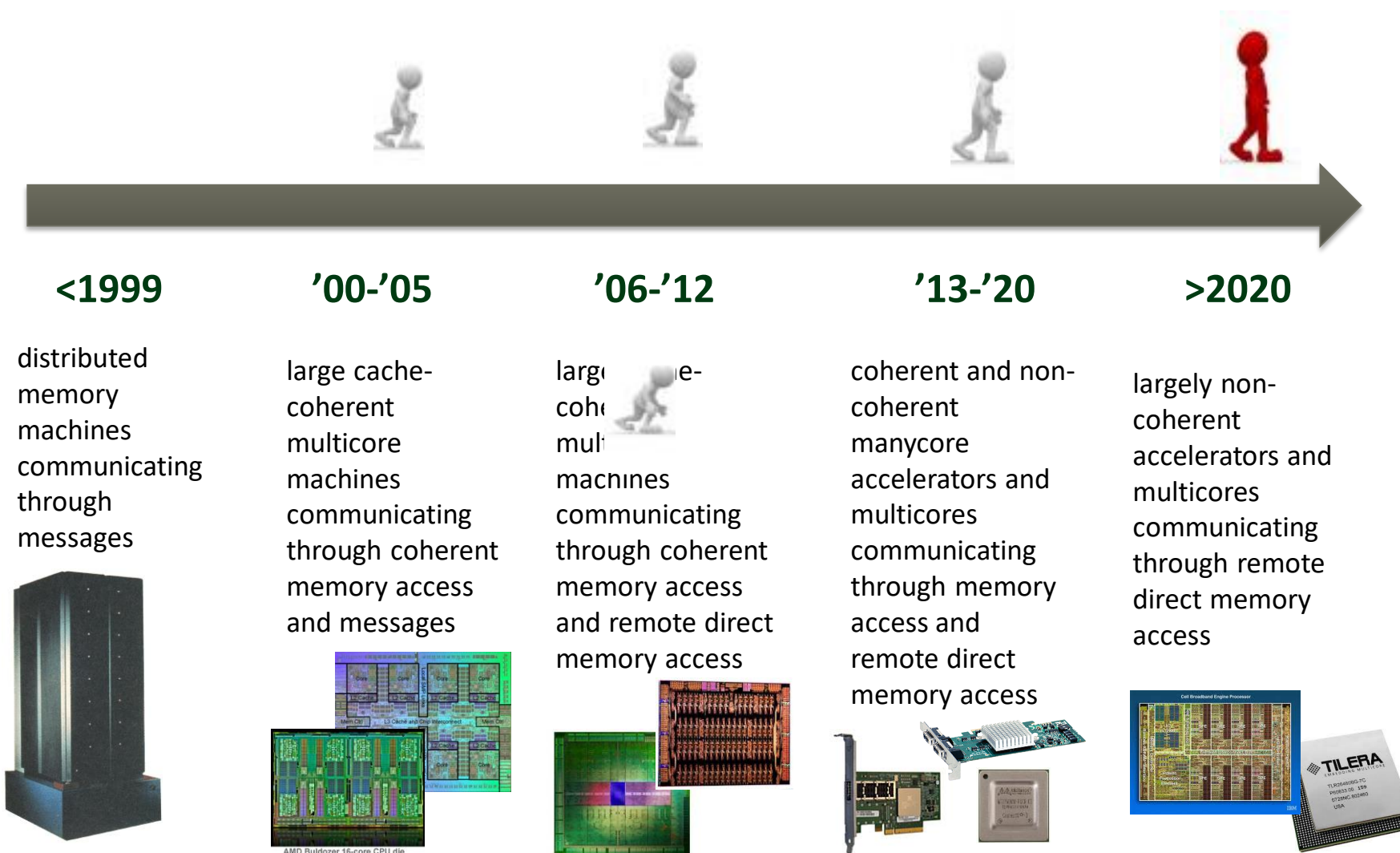
- **Not covered:**
  - SMM: Intel Cilk / Cilk Plus, Intel TBB, ...
  - Directives: OpenHMPP, PVM, ...
  - PGAS: Coarray Fortran (Fortran 2008), ...
  - HPCS: IBM X10, Fortress, Chapel, ...
  - Accelerator: OpenCL, C++AMP, ...
  - ...
- **This class will not describe any model in more detail!**
  - There are too many and they will change quickly (only MPI made it >15 yrs)
- **No consensus, but fundamental questions remain:**
  - Data movement (I/O complexity)
  - Synchronization (avoiding races, deadlock etc.)
  - Memory Models (read/write ordering)
  - Algorithmics (parallel design/thinking)
  - Foundations (conflict minimization, models, static vs. dynamic scheduling etc.)



# Goals of this lecture

- Motivate you!
- What is parallel computing?
  - And why do we need it?
- What is high-performance computing?
  - What's a Supercomputer and why do we care?
- Basic overview of
  - Programming models
    - Some examples*
  - Architectures
    - Some case-studies*
- Provide context for coming lectures

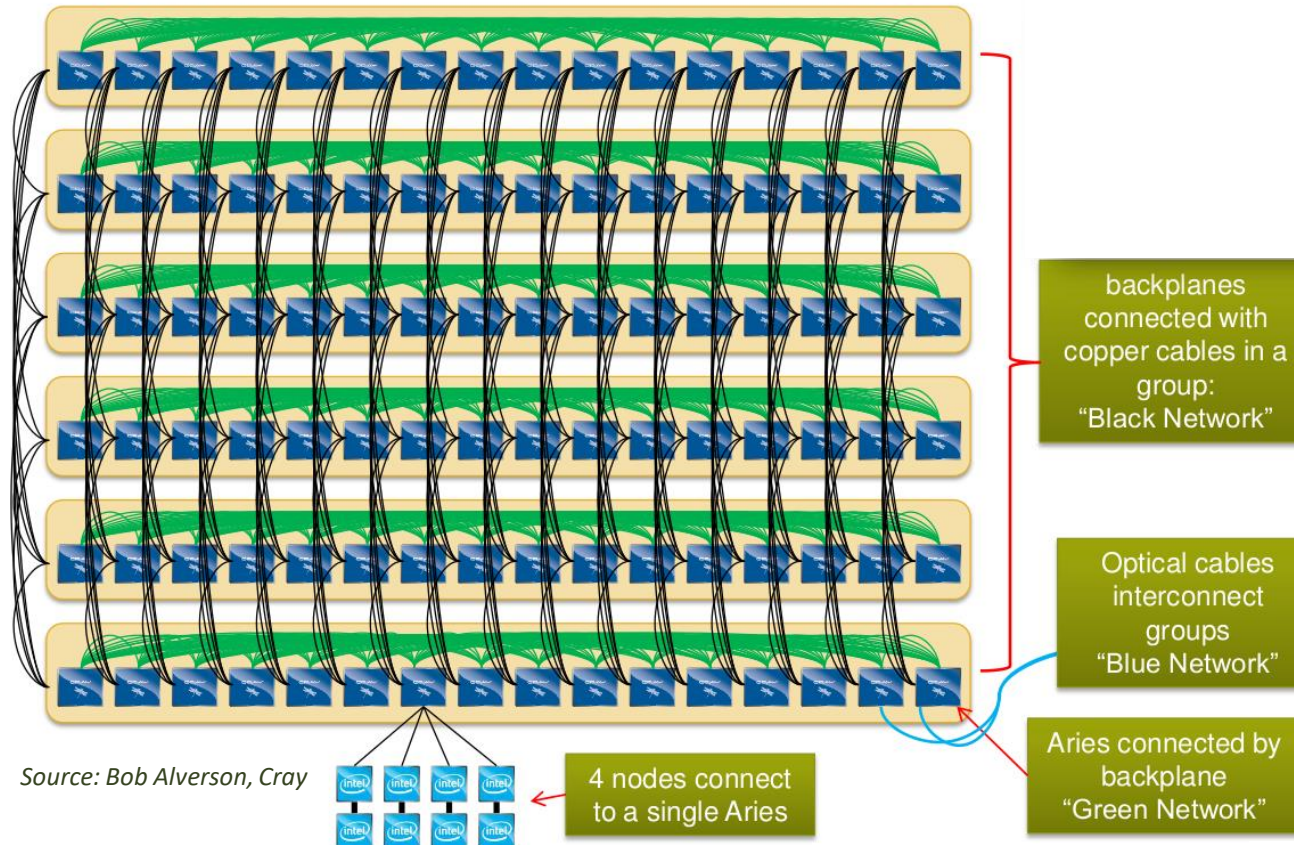
# Architecture Developments



Sources: various vendors

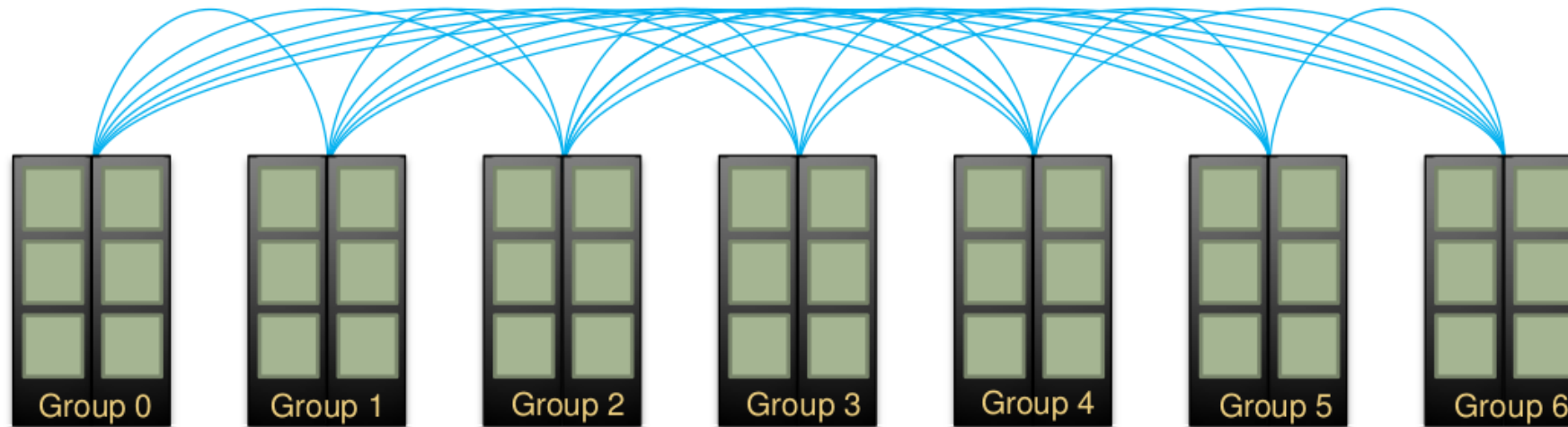
# Case Study 1: Cray Cascade (XC30) – Piz Daint!

- **Biggest current installation at CSCS! 😊**
  - >2k nodes
- **Standard Intel x86 Sandy Bridge Server-class CPUs**



# Case Study 1: Cray Cascade Network Topology

- All-to-all connection among groups (“blue network”)

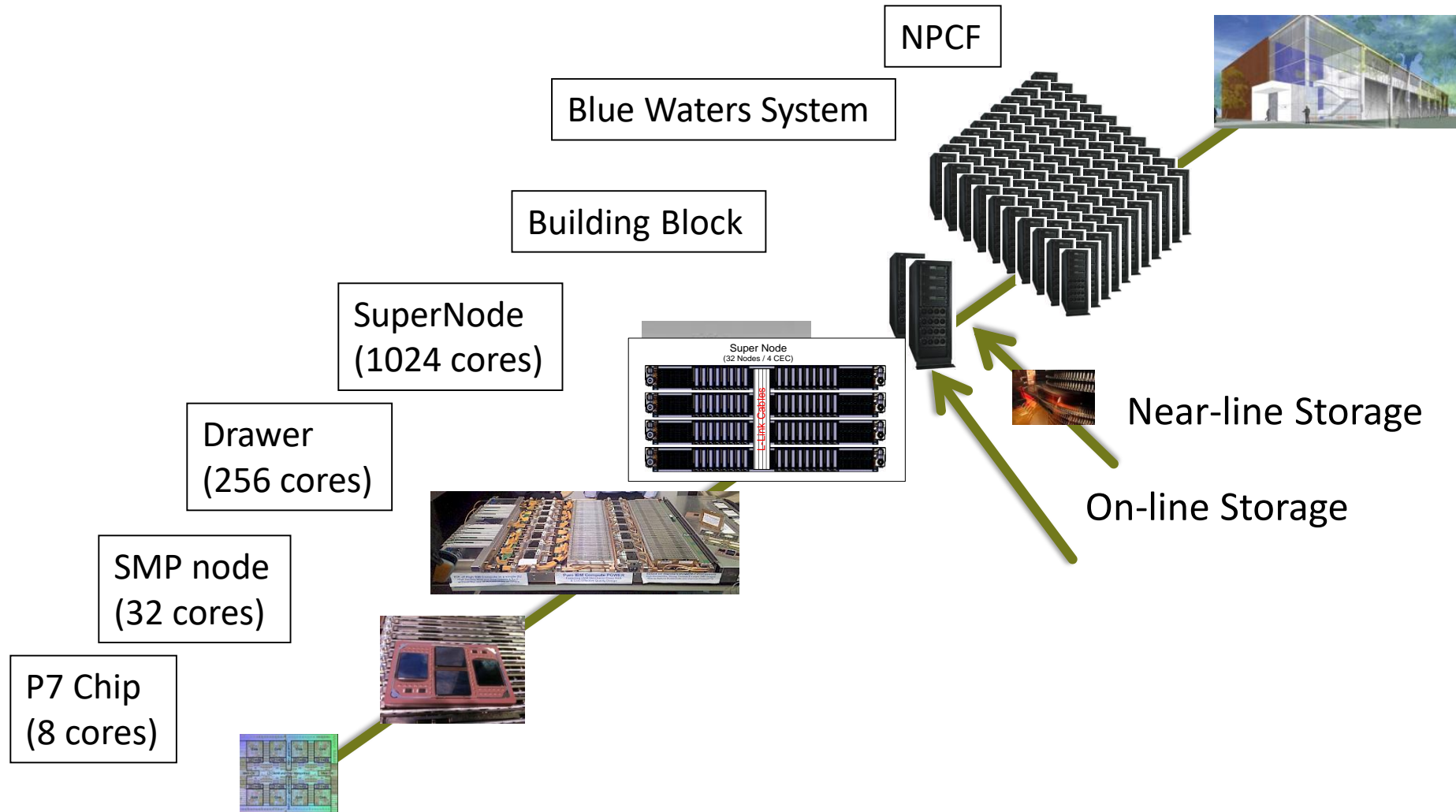


Source: Bob Alverson, Cray

- Interesting research opportunities!
- Topology design?
  - E.g., Besta, TH: Slim Fly: A Cost Effective Low-Diameter Network Topology
- Interference analysis (can we provide isolation)?
- How to route low-diameter topologies?



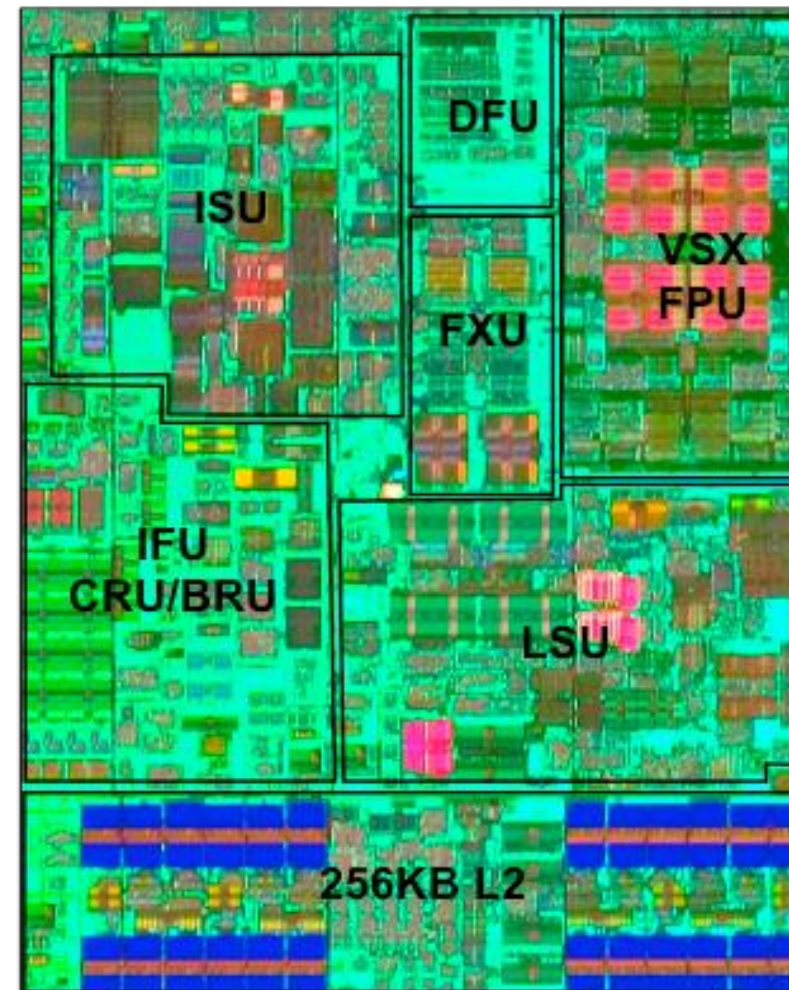
# Case Study 2: IBM POWER7 IH (BW)



Source: IBM/NCSA

# POWER7 Core

- Execution Units
  - 2 Fixed point units
  - 2 Load store units
  - 4 Double precision floating point
  - 1 Branch
  - 1 Condition register
  - 1 Vector unit
  - 1 Decimal floating point unit
  - 6 wide dispatch
- Recovery Function Distributed
- 1,2,4 Way SMT Support
- Out of Order Execution
- 32KB I-Cache
- 32KB D-Cache
- 256KB L2
  - Tightly coupled to core



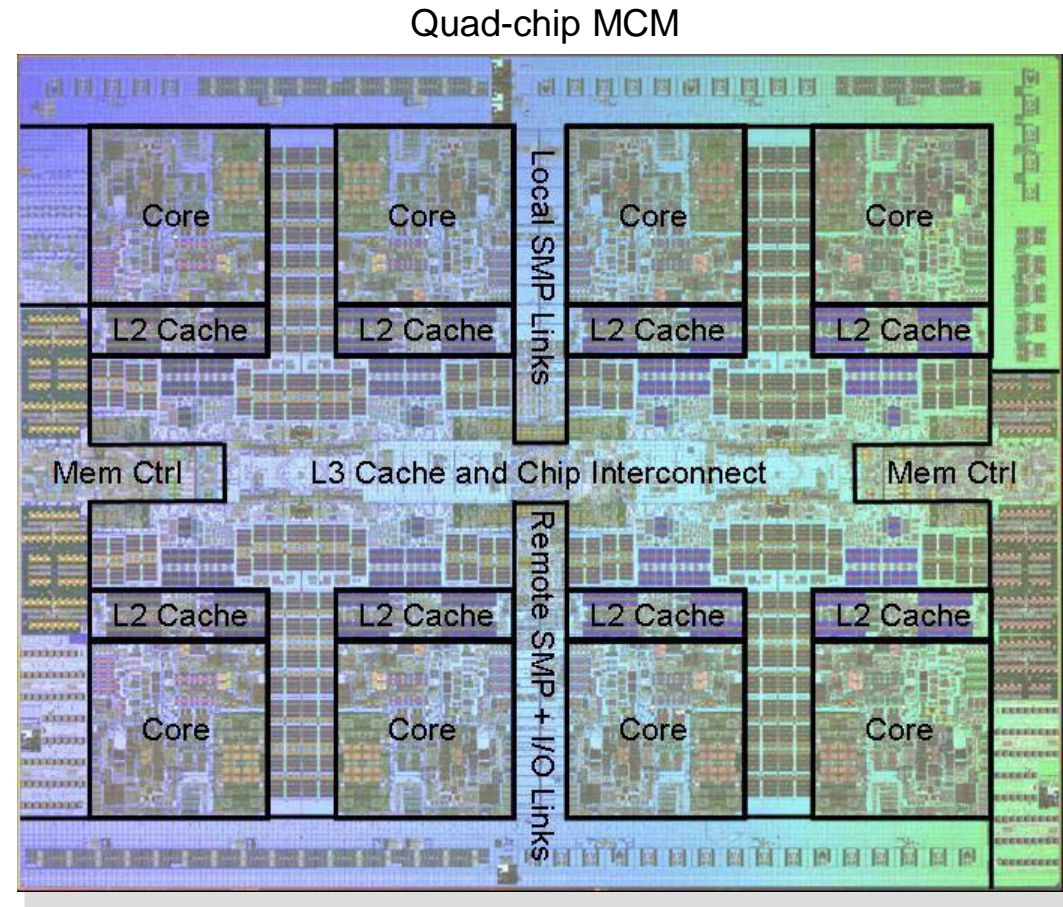
# POWER7 Chip (8 cores)

- **Base Technology**

- 45 nm, 576 mm<sup>2</sup>
- 1.2 B transistors

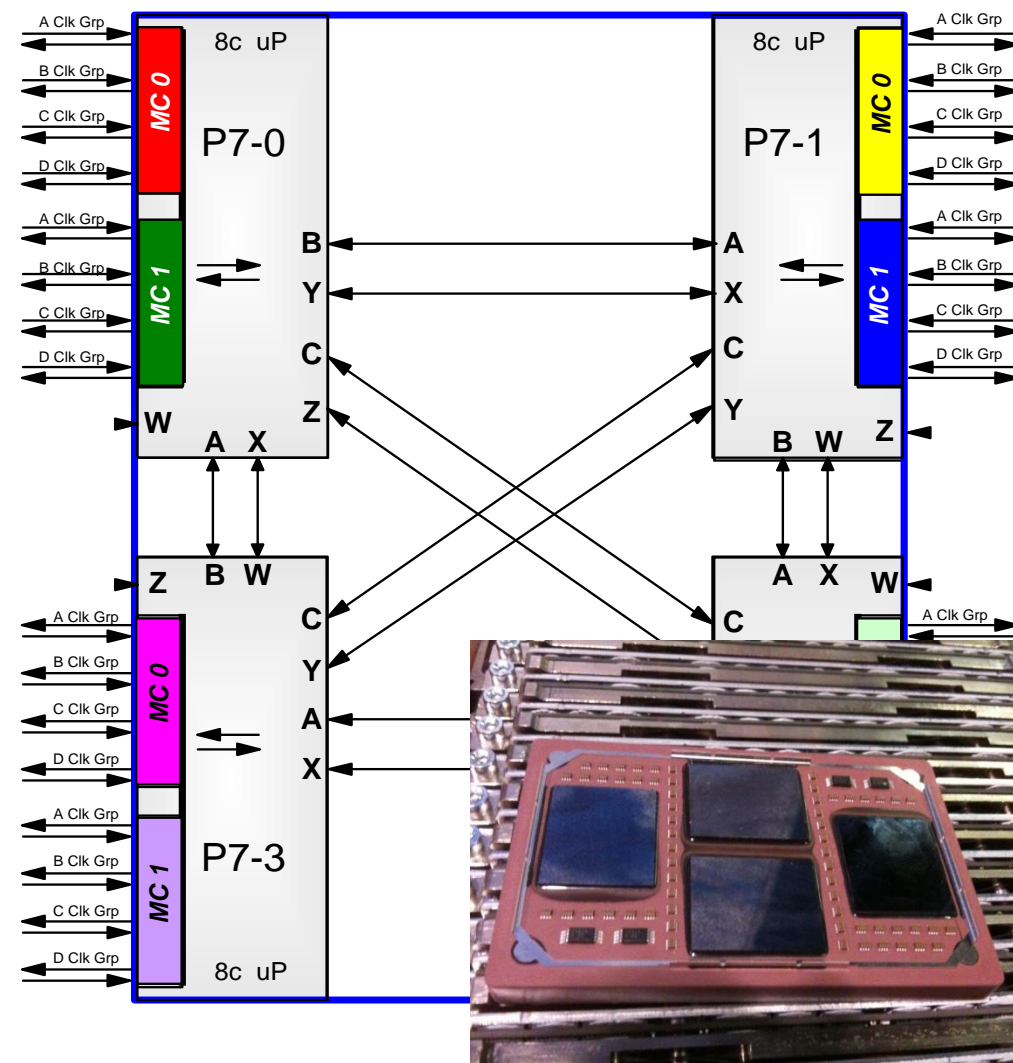
- **Chip**

- 8 cores
- 4 FMAs/cycle/core
- 32 MB L3 (private/shared)
- Dual DDR3 memory  
*128 GiB/s peak bandwidth (1/2 byte/flop)*
- Clock range of 3.5 – 4 GHz



# Quad Chip Module (4 chips)

- **32 cores**
  - 32 cores\*8 F/core\*4 GHz = 1 TF
- **4 threads per core (max)**
  - 128 threads per package
- **4x32 MiB L3 cache**
  - 512 GB/s RAM BW (0.5 B/F)
- **800 W (0.8 W/F)**

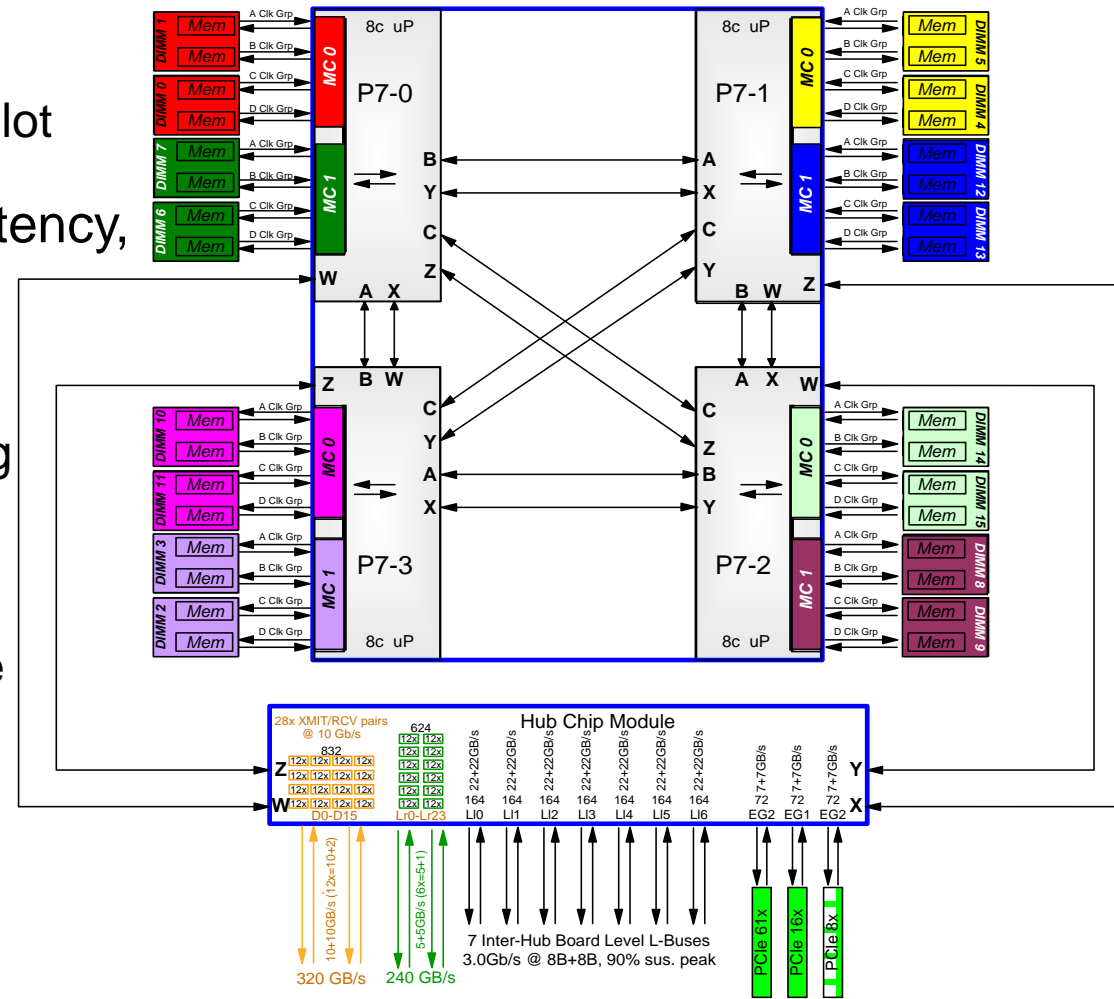


Source: IBM/NCSA



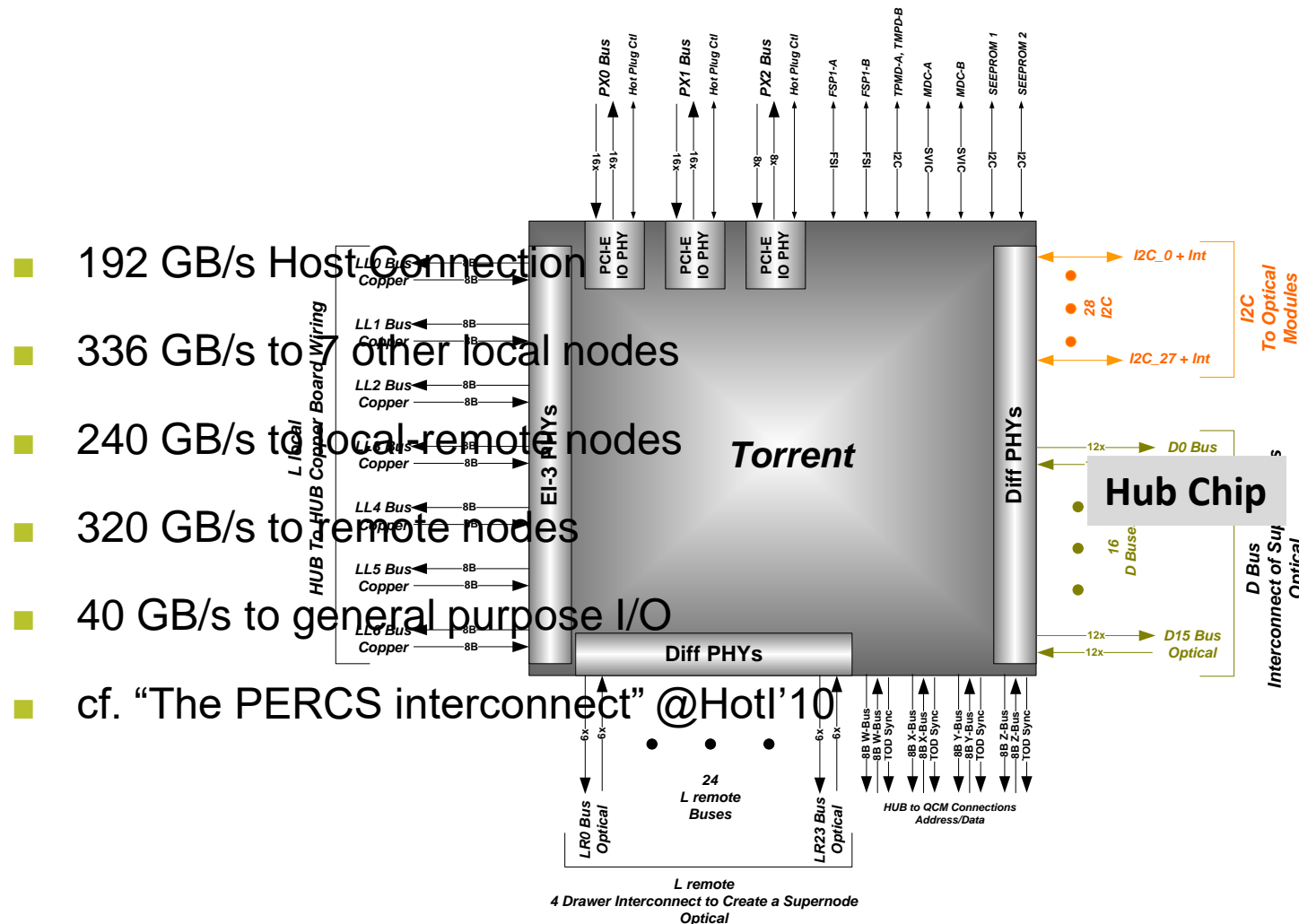
# Adding a Network Interface (Hub)

- Connects QCM to PCI-e
  - Two 16x and one 8x PCI-e slot
- Connects 8 QCM's via low latency, high bandwidth, copper fabric.
  - Provides a message passing mechanism with very high bandwidth
  - Provides the lowest possible latency between 8 QCM's



Source: IBM/NCSA

# 1.1 TB/s POWER7 IH HUB



- 192 GB/s Host Connection
- 336 GB/s to 7 other local nodes
- 240 GB/s to local-remote nodes
- 320 GB/s to remote nodes
- 40 GB/s to general purpose I/O
- cf. “The PERCS interconnect” @HotI’10

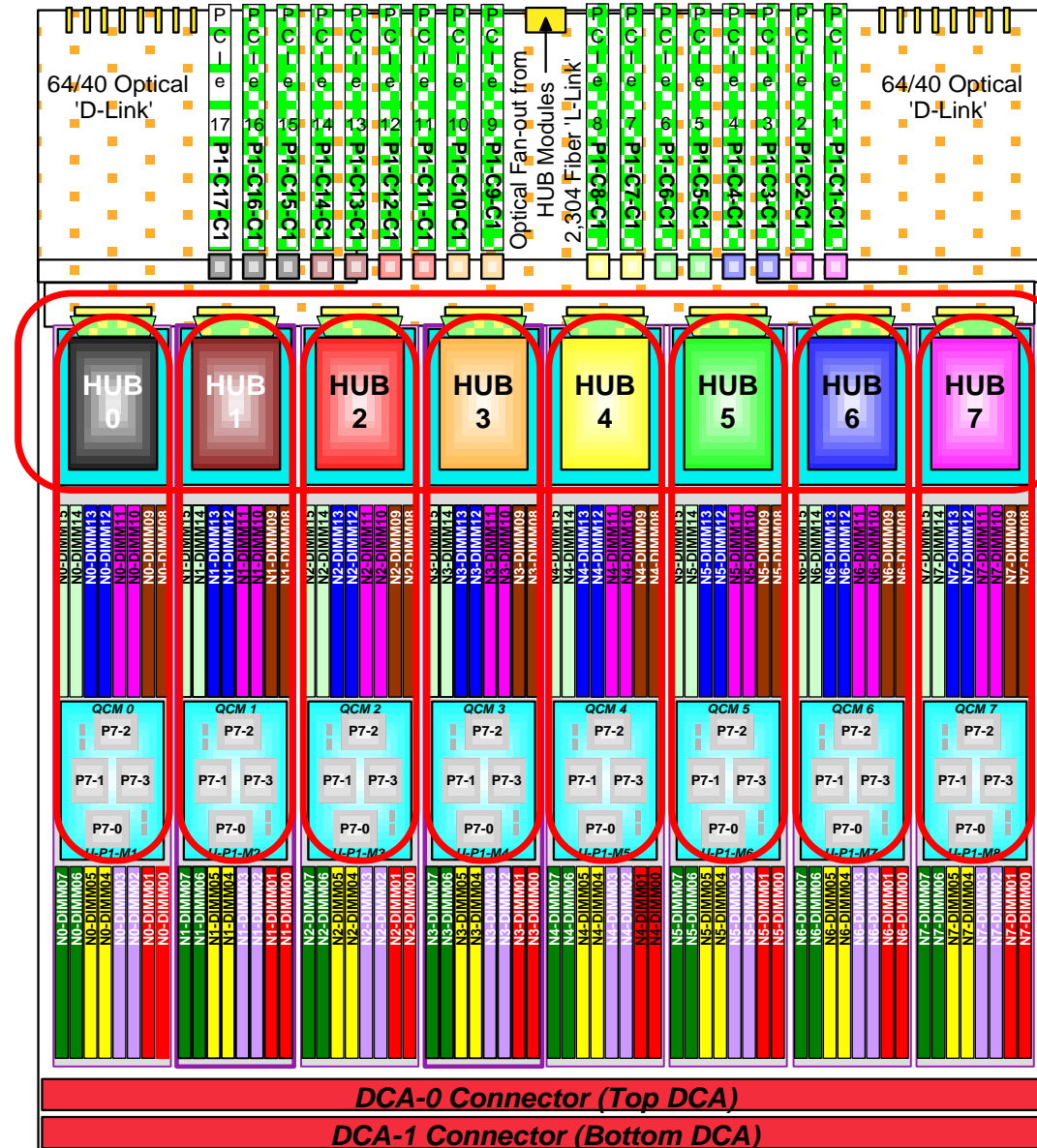
Source: IBM/NCSA

# P7 IH Drawer

- 8 nodes
- 32 chips
- 256 cores

## First Level Interconnect

- L-Local
- HUB to HUB Copper Wiring
- 256 Cores



Source: IBM/NCSA

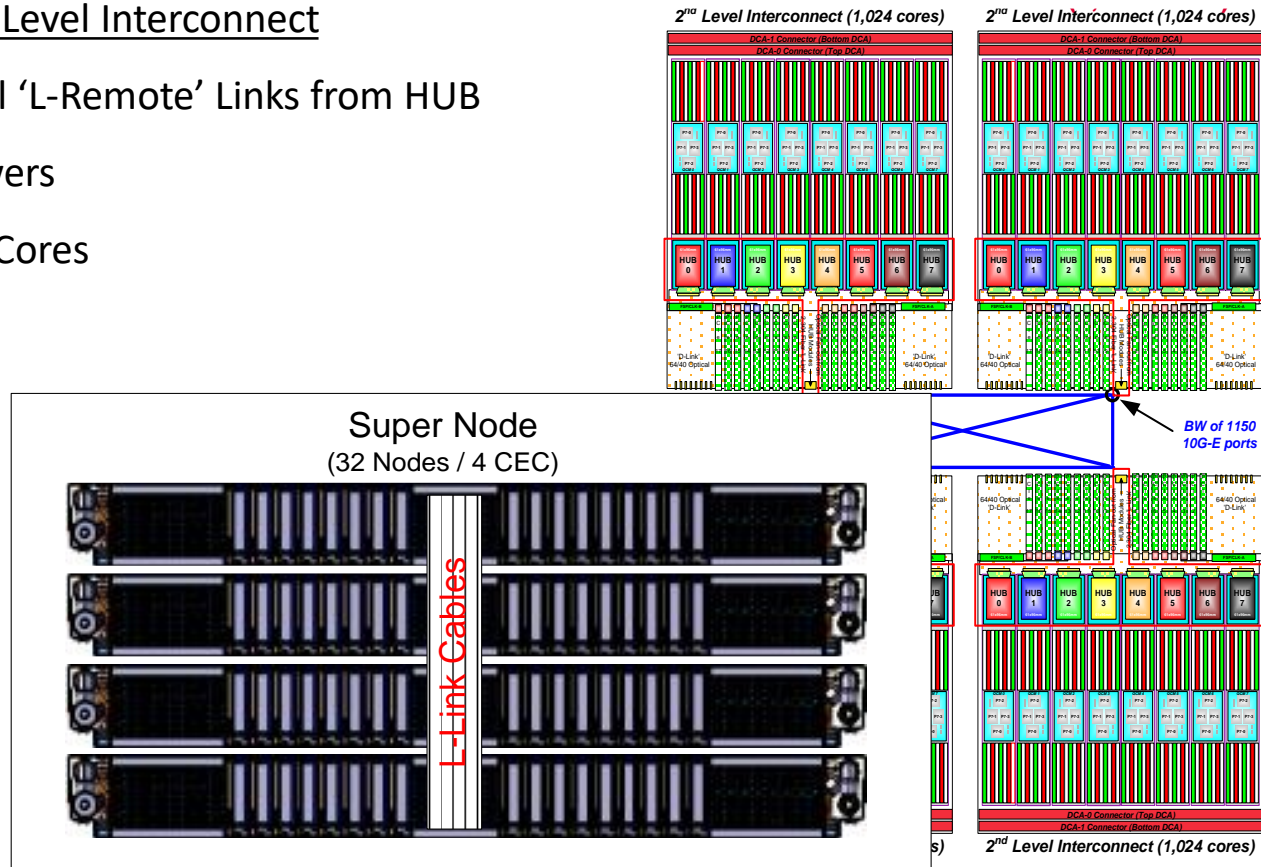
# POWER7 IH Drawer @ SC09



# P7 IH Supernode

## Second Level Interconnect

- Optical 'L-Remote' Links from HUB
- 4 drawers
- 1,024 Cores



# Goals of this lecture

- **Motivate you!**
- **What is parallel computing?**
  - And why do we need it?
- **What is high-performance computing?**
  - What's a Supercomputer and why do we care?
- **Basic overview of**
  - Programming models
    - Some examples*
  - Architectures
    - Some case-studies*
- **Provide context for coming lectures**

# DPHPC Lecture

- You will likely not have access to the largest machines (unless you specialize to HPC)
  - But our desktop/laptop will be a “large machine” soon
  - HPC is often seen as “Formula 1” of computing (architecture experiments)
- DPHPC will teach you concepts!
  - Enable to understand and use all parallel architectures
  - From a quad-core mobile phone to the largest machine on the planet!  
*MCAPI vs. MPI – same concepts, different syntax*
  - No particular language (but you should pick/learn one for your project!)  
*Parallelism is the future:*



## Related classes in the SE focus

- **263-2910-00L Program Analysis**  
<http://www.srl.inf.ethz.ch/pa.php>  
Spring 2017  
Lecturer: Prof. M. Vechev
- **263-2300-00L How to Write Fast Numerical Code**  
<http://www.inf.ethz.ch/personal/markusp/teaching/263-2300-ETH-spring16/course.html>  
Spring 2017  
Lecturer: Prof. M. Pueschel
- **This list is not exhaustive!**



# DPHPC Overview

