Parallel Architectures

Frédéric Desprez

INRIA

Some references

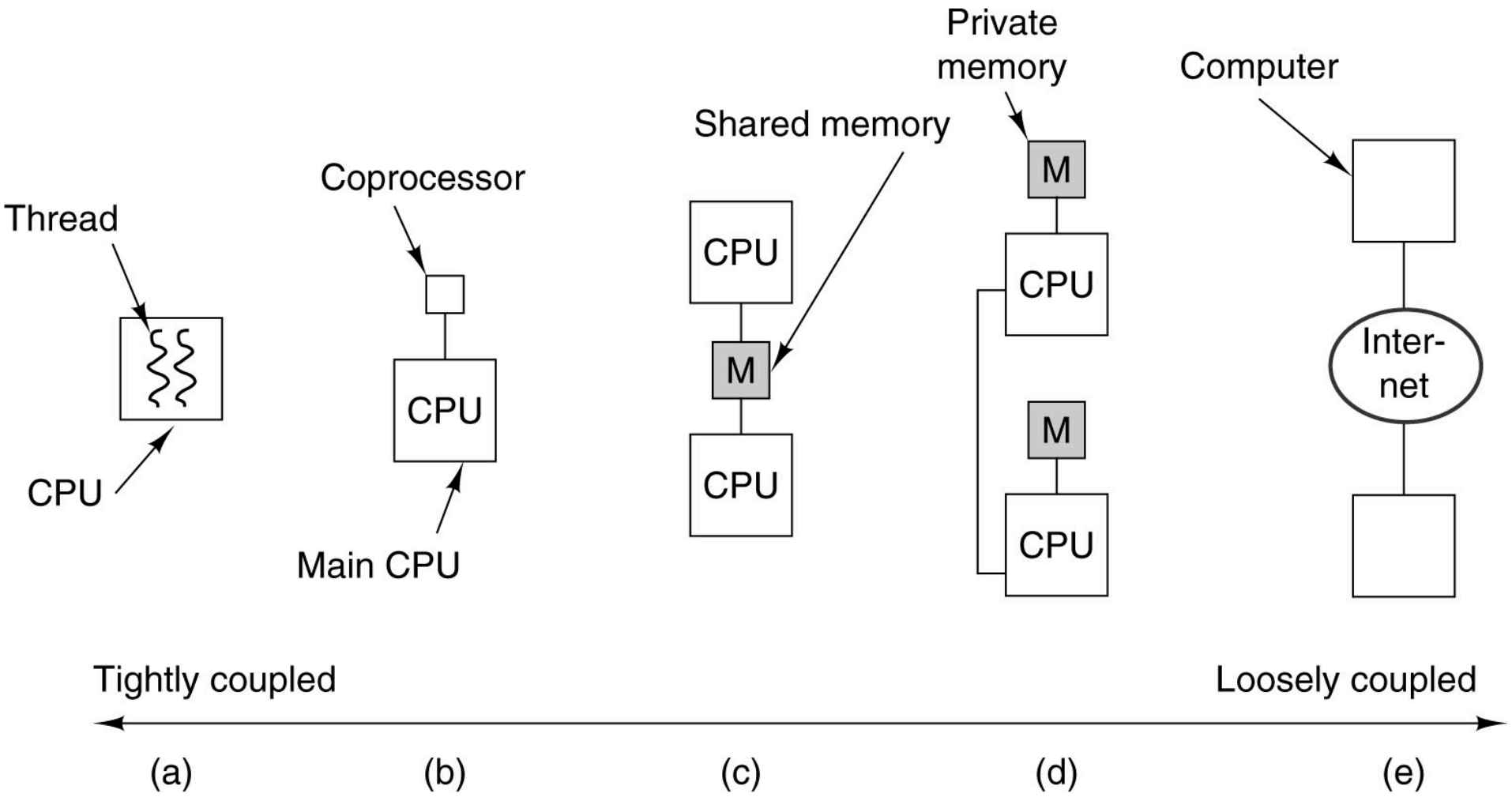
- Lecture "**Calcul hautes performance – architectures et modèles de programmation**", Françoise Roch, Observatoire des Sciences de l'Univers de Grenoble Mesocentre CIMENT
- **4 visions about HPC - A chat**, X. Vigouroux, Bull
- **Parallel Programming – For Multicore and Cluster System,** T. Rauber, G. Rünger

Lecture summary

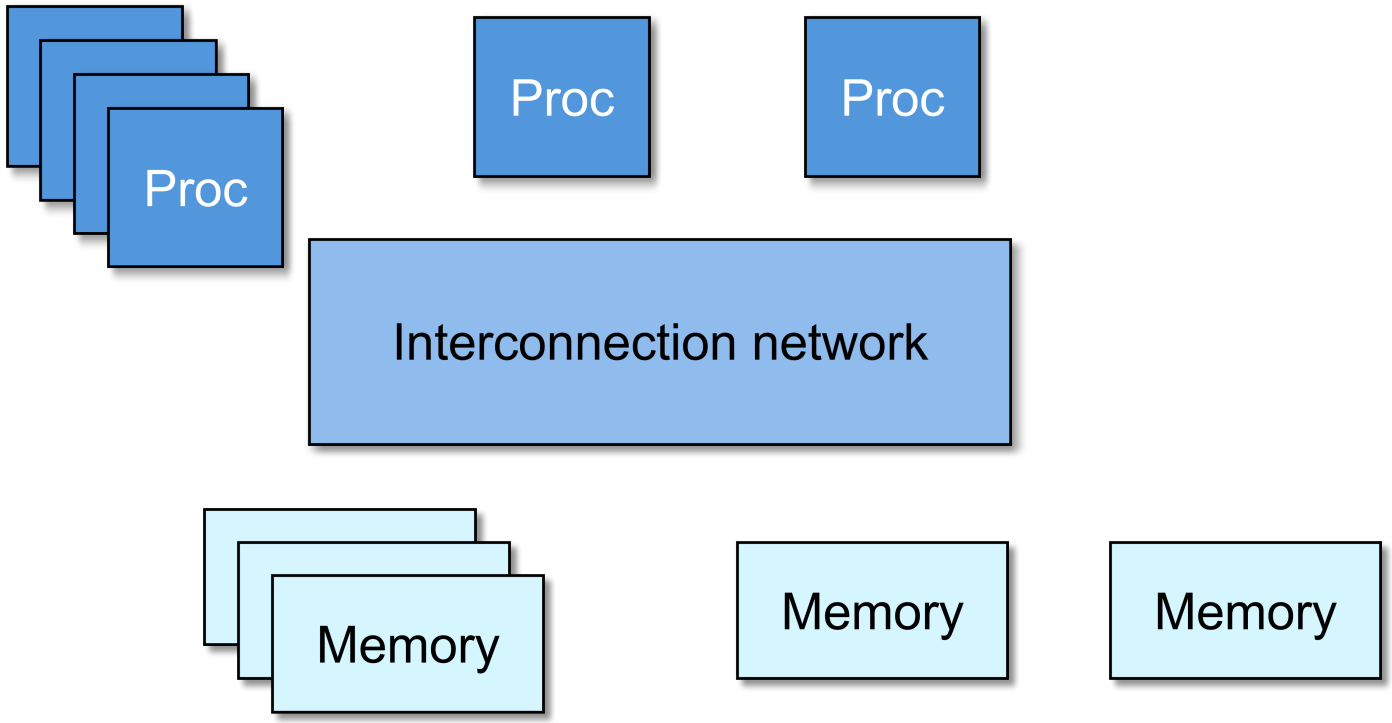
- Introduction
- Models of parallel machines
- Multicores/GPU
- Interconnection networks

MODELS OF PARALLEL MACHINES

Parallel architectures



A generic parallel machine

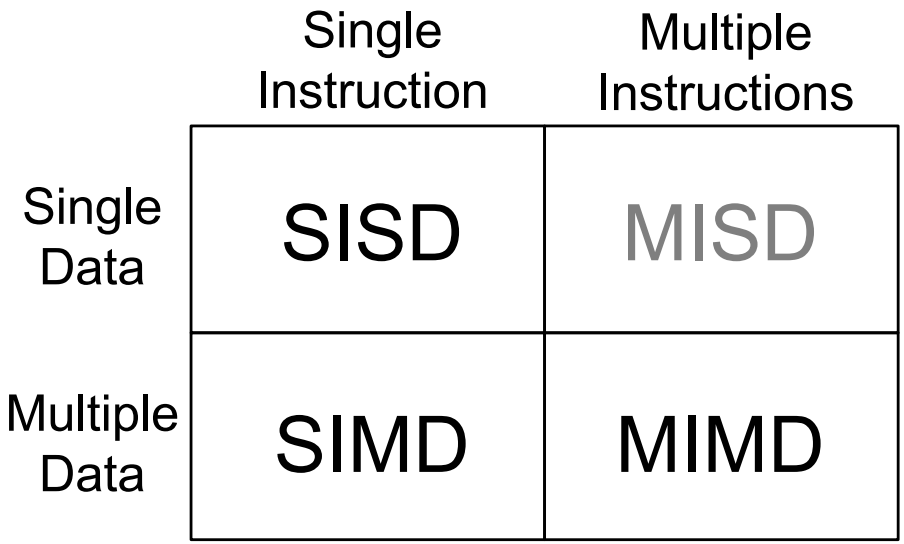


- Where is the memory?
- Is it connected directly to the processors?
- What is the processor connectivity?

Parallel machines models

Flynn's classification

- Characterizes machines according to their flow of data and instructions

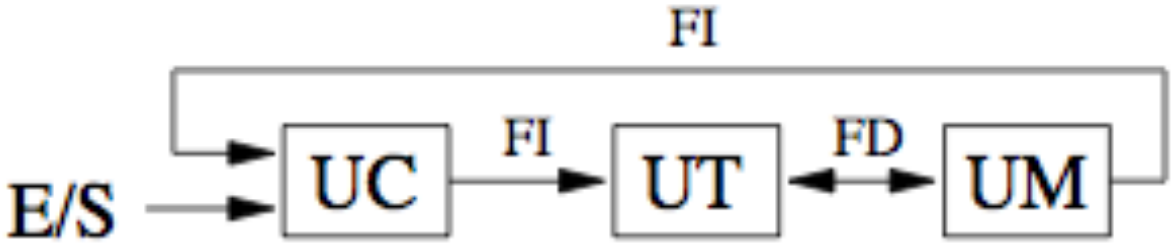


Flynn, M., "**Some Computer Organizations and Their Effectiveness**". IEEE Trans. Comput. C-21: 948., 1972.

SISD: Single Instruction, Single Data stream

"Classical" sequential machines

Each operation is performed on one data at a time



UC = Control Unit (responsible for the sequencing of instructions)

UT = Processing Unit (performs the operations)

FI = Instructions Flow

UM = Memory Unit (contains instructions and data)

FD = Data Flow

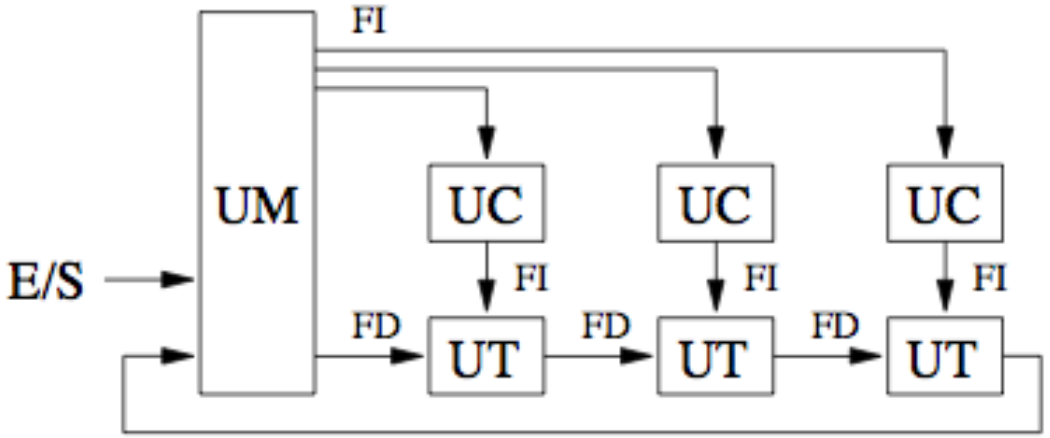
Von Neuman's model (1945)

MISD: Multiple Instruction stream, Single Data stream

Specialized "systolic" type machines

Processors arranged with a fixed topology

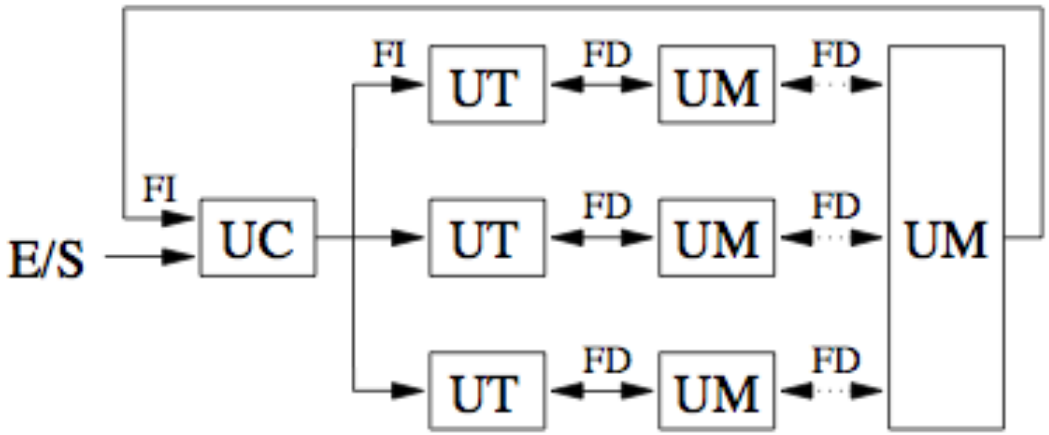
Strong synchronization



SIMD: Single Instruction stream, Multiple Data stream

Totally **synchronized** calculation units

Conditional execution with masking flag



• Machines adapted to very regular processing (matrix operations, FFT, image processing)

• Not adapted at all to irregular operations

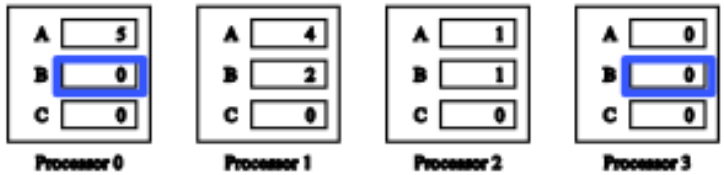
Conditionals in SIMD

- **Masking flag**
  - Used to prevent processors from performing some operations

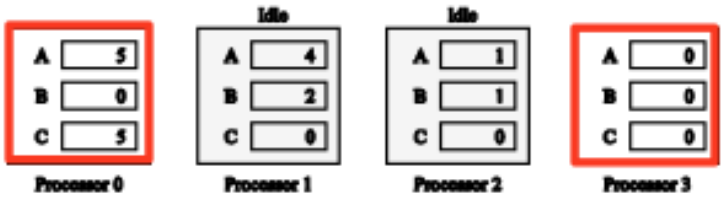
conditional statement

```
if (B == 0)
then C = A
else C = A/B
```

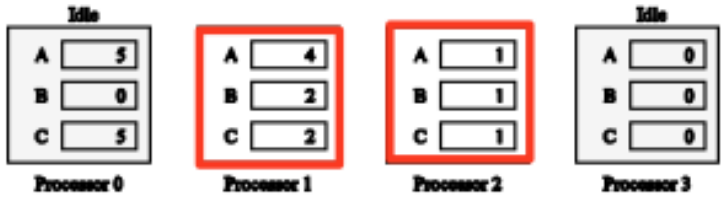
initial values



execute "then" branch



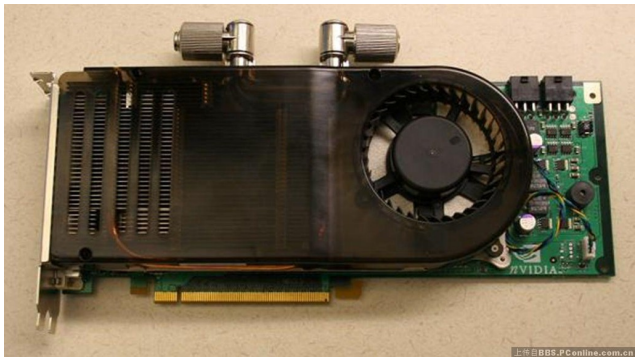
execute "else" branch



Some examples of SIMD machines

- **80's/90's parallel machines**
  - Illiac IV, MPP, DAC, Connection Machine CM-1/2, MasPar MP-1/2

- **A great return today**
  - Intel processors and SSE / SSE-2 mode (vector units)
  - 128-bit vector registers
    - 16 floats (8 bits), 8 short integers (16 bits), 4 integers (32 bits)
    - 2 floats (64 bits) for SSE-2
  - Altivec (Velocity Engine, VMX)
  - Co-processors
    - GPGPU nVidia G80
    - ClearSpeed array processor (2 control processors + 192 processors)



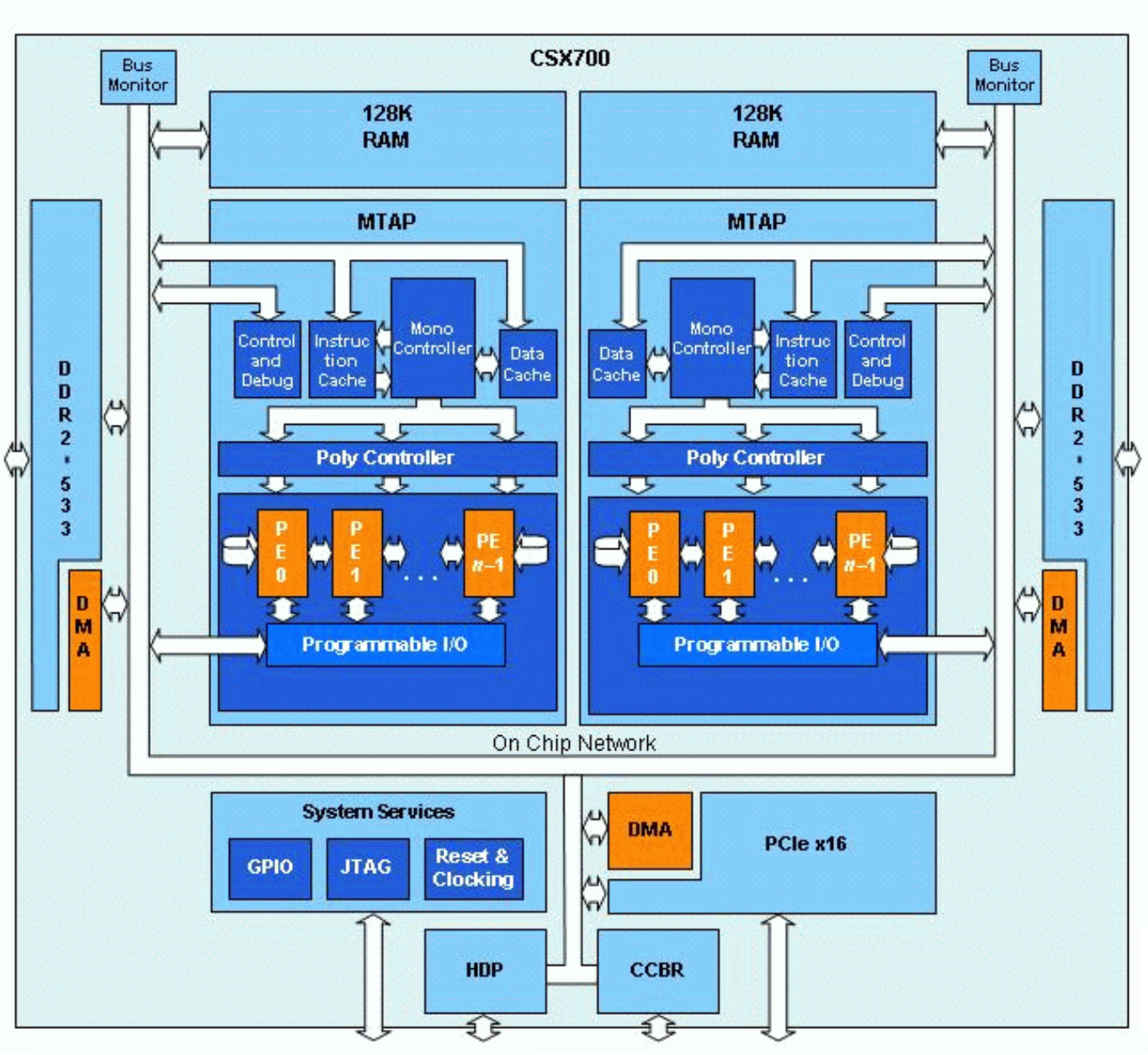
ClearSpeed CSX700 Processor

- **Dual independent Multi-threaded SIMD array processors (MTAP); each containing 96 processing elements**
  - PCIe x16 host interface
  - 2 x 64-bit DDR2 DRAM interface with ECC support
  - 256 Kbytes on-chip scratchpad memory
  - On-chip instruction and data caches
  - 64-bit virtual, 48-bit physical addressing
  - On-chip DMA controller

- **192 high-performance processing elements, containing**
  - 32/64-bit FP Multiplier
  - 32/64-bit FP Adder
  - 128-byte register file
  - 6 Kbytes of high bandwidth SRAM
  - High speed I/O channel
  - Integer ALU and 16-bit integer MAC

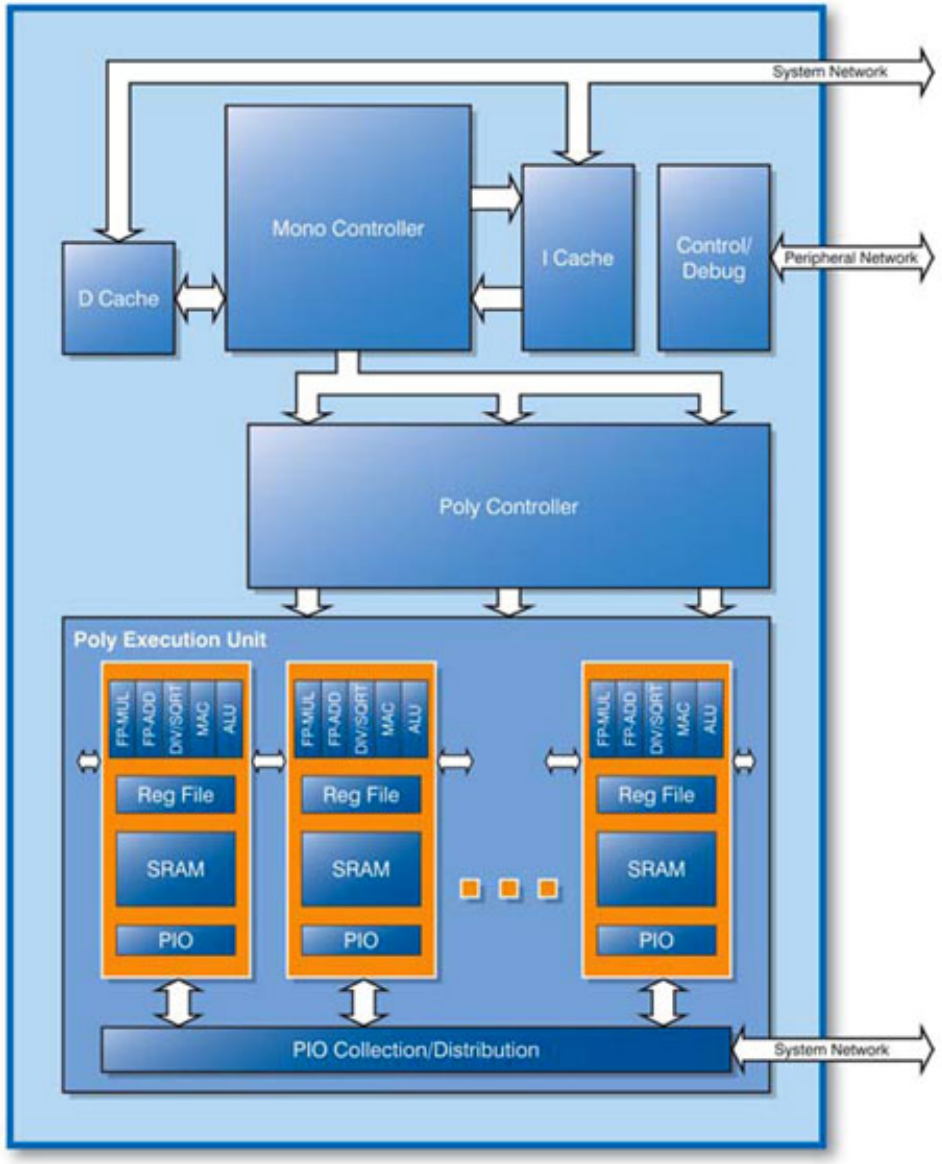
ClearSpeed™

ClearSpeed CSX700 Processor, contd.



ClearSpeed™

ClearSpeed CSX700 Processor, contd.



ClearSpeed™

ClearSpeed CSX700 Processor, contd.

- **Performances**
  - 250MHz core clock frequency
  - 96 GFLOPS single or double precision
  - 75 GFLOPS sustained double precision DGEMM
  - 48 GMAC/s integer performance
  - 192 Gbytes/s internal memory bandwidth
  - 2 x 4 Gbytes/s external memory bandwidth
  - 4 Gbytes/s chip-to-chip bandwidth

ClearSpeed™

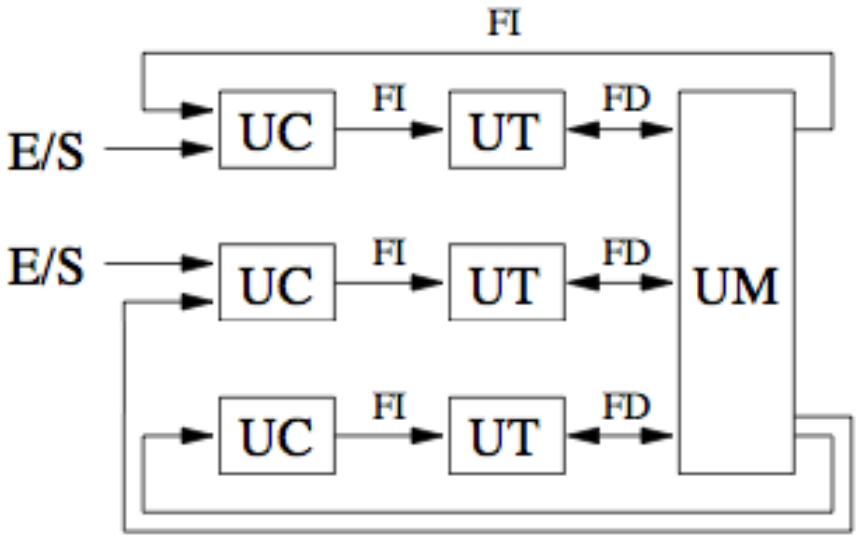
MIMD: Multiple Instructions stream, multiple data stream

Multi-Processor Machines

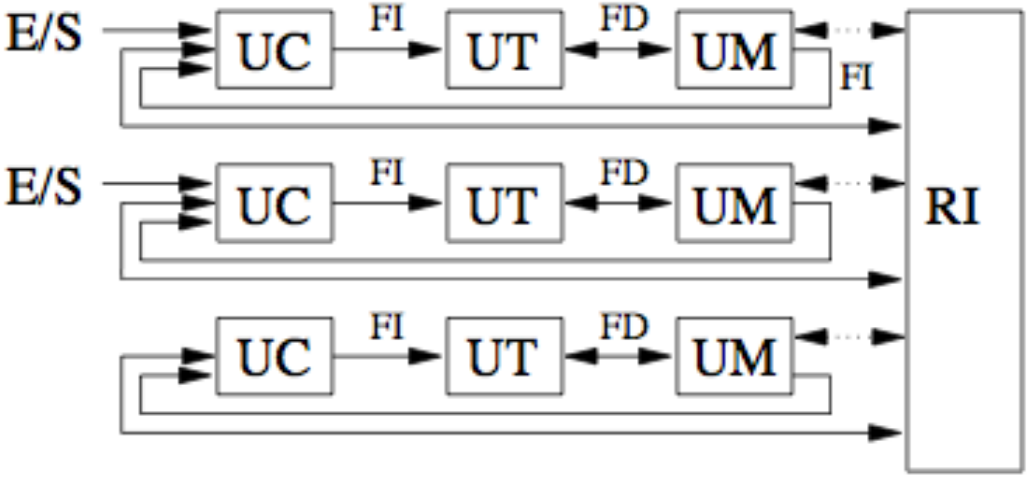
Each processor runs its own code asynchronously and independently

Two sub-classes

Shared memory



Distributed memory

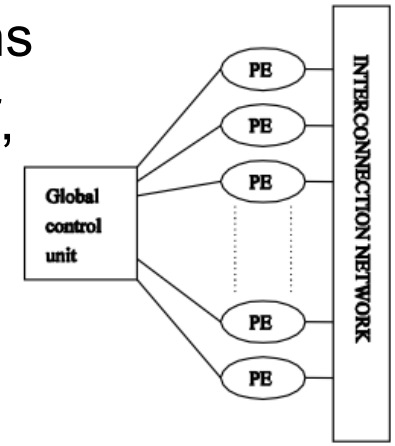


A mix between SIMD and MIMD: SPMD (**Single Program, Multiple Data**)

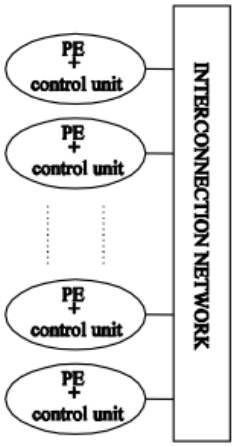
SIMD vs MIMD

- **SIMD Platforms**
  - Designed for specific applications
  - Complicated (and long) design, no "on-shelf" processors
  - Less equipment (one control unit)
  - Need less memory for instructions (single program)
  - Used heavily for current co-processors

- **MIMD Platforms**
  - Works for a wide variety of applications
  - Less expensive (components on shelf, short design)
  - Need more memory (OS and program on each processor)



SIMD architecture



MIMD architecture

Raina's classification

Taking into account the address space

- **SASM** (Single Address space, Shared Memory)
  Shared memory

- **DADM** (Distributed Address space, Distributed Memory)
  Distributed memory, without access to remote data. The exchange of data between processors is necessarily effected by passing messages, by means of a communication network

- **SADM** (Single Address space, Distributed Memory)
  Distributed memory, with global address space, possibly allowing access to data located on other processors

Raina's classification, contd.

The type of memory access implemented

NORMA (No Remote Memory Access)

No means of access to remote data, requiring the message passing

UMA (Uniform Memory Access)

Symmetric access to memory, identical cost for all processors

NUMA (Non-Uniform Memory Access)

The access performance depends on the location of the data

CC-NUMA (Cache-Coherent NUMA)

Type of NUMA architecture integrating caches

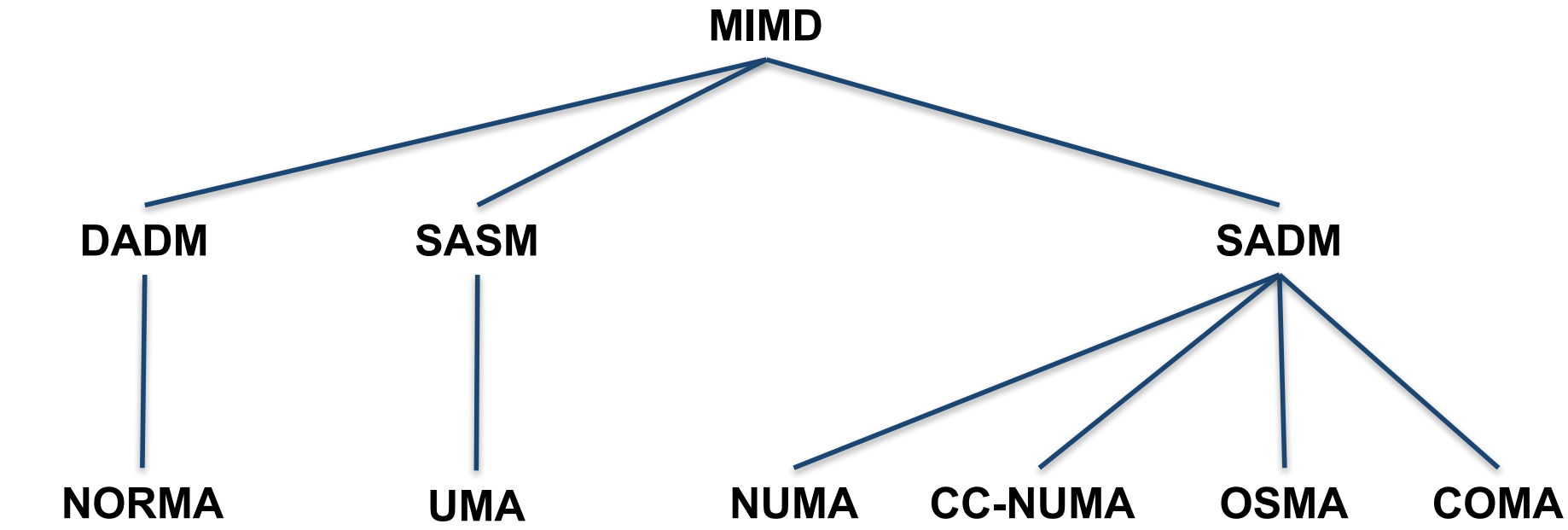
OSMA (Operating System Memory Access)

The remote data accesses are managed by the operating system, which handles page faults at the software level and handles remote copy/send requests

COMA (Cache Only Memory Access)

The local memories behave like caches, so that a data item has neither a proprietary processor nor a determined location in memory

Raina's classification, contd.



Parallel Programming Models

The **programming model** consists of the languages and libraries that will allow to have an **abstraction** of the machine

Control

- How is parallelism created (implicit or explicit)?
- What are the sequences between operations (synchronous or asynchronous)?

Data

- What are the private and shared data?
- How are these data accessed and / or communicated?

Synchronization

- What operations can be used to coordinate parallelism?
- What are atomic (indivisible) operations?

Cost

- How can we calculate the cost of each previous item?

A simple example: the sum

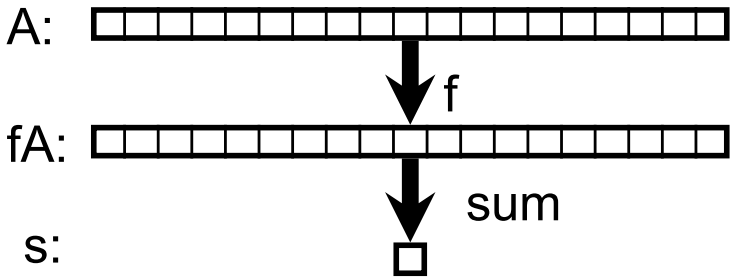
A function f is applied to the elements of an array A and the sum

$$\sum_{i=0}^{n-1} f(A[i])$$

Questions

- Where is A? In a central memory? Distributed?
- What will be the work done by the processors?
- How will they coordinate themselves to achieve a single outcome?

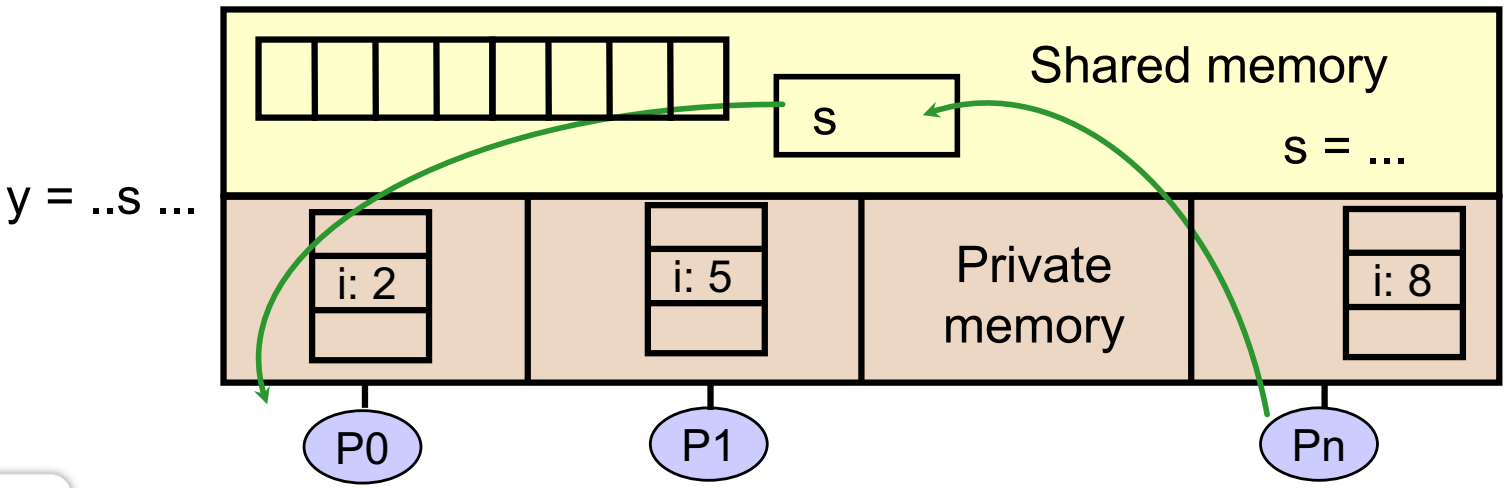
A = data array
fA = f(A)
s = sum(fA)



Shared memory

The program is a set of control threads

- They can sometimes be created dynamically during execution in some languages
- Each thread has its own private data set (local stack variables)
- Set of shared variables (static variables, shared blocks, global stack)
- Threads communicate by writing and reading shared variables
- They synchronize on shared variables



Parallelization strategy

$$\sum_{i=0}^{n-1} f(A[i])$$

Shared Memory strategy

- Small number of processors (p << n = size(A))
- Connected to a single central memory

Parallel decomposition

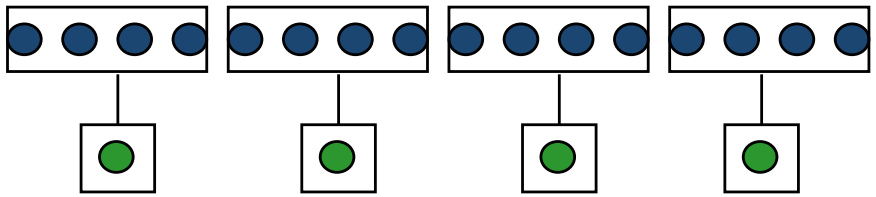
- Each evaluation and each partial sum is a task

Assign n / p numbers to each processor p

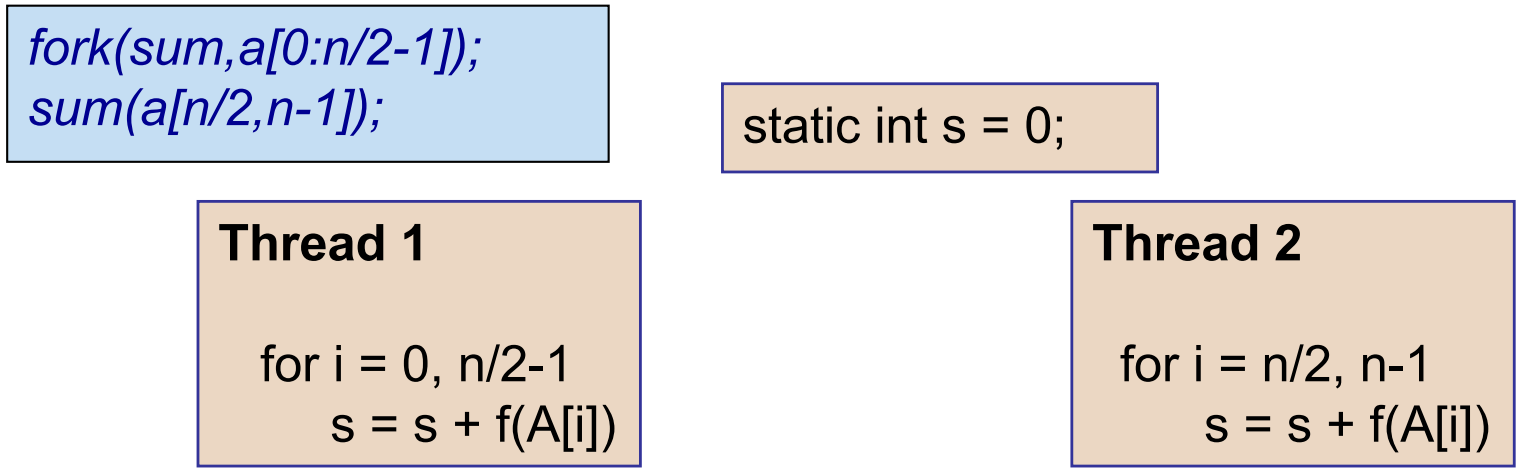
- Each of them calculates private results and a partial sum
- Gather the p local sums and calculate the total sum

Two classes of data

- Shared (logically)
  - The n numbers, the global sum
- Private (logically)
  - Local evaluations of functions



Shared memory "code" for the computation of the sum



- **What is the problem with this program?**
- **A race condition occurs when**
  - Two processors (or two threads) access the same variable (and at least one of them performs a write)
  - The accesses are competing (not synchronized) and they can appear at the same time

Shared memory "code" for the computation of the sum, contd.

A= | 3 | 5 |

$f(x) = x^2$

```
static int s = 0;
```

Thread 1		Thread 2	
….		…	
compute f([A[i]) and put in reg0	9	compute f([A[i]) and put in reg0	25
reg1 = s	0	reg1 = s	0
reg1 = reg1 + reg0	9	reg1 = reg1 + reg0	25
s = reg1	9	s = reg1	25
…		…	

- Suppose that A = [3,5], $f(x) = x^2$ and s=0 at the start
- For the result to be correct we need to have s = $3^2 + 5^2 = 34$ at the end
  - But here it can be 34, 9, or 25
- Atomic operations are read and write
  - We will not see a mixture of numbers but the operation + = is not atomic
  - All computations take place in private registers

Improved code for the sum

```
static int s = 0;
static lock lk;
```

Thread 1

```
local_s1= 0
for i = 0, n/2-1
    local_s1 = local_s1 + f(A[i])
lock(lk);
s = s + local_s1
unlock(lk);
```

Thread 2

```
local_s2 = 0
for i = n/2, n-1
    local_s2= local_s2 + f(A[i])
lock(lk);
s = s +local_s2
unlock(lk);
```

- Since the addition is associative, one can change the order
- Most computations take place on private variables
  - The frequency of sharing is also reduced, which can improve the speed
  - But there is always competition for updating s
  - It can be deleted with locks (only one thread can have a lock at one time, the other waits)

Shared memory machine model

Processors are connected to a large shared memory

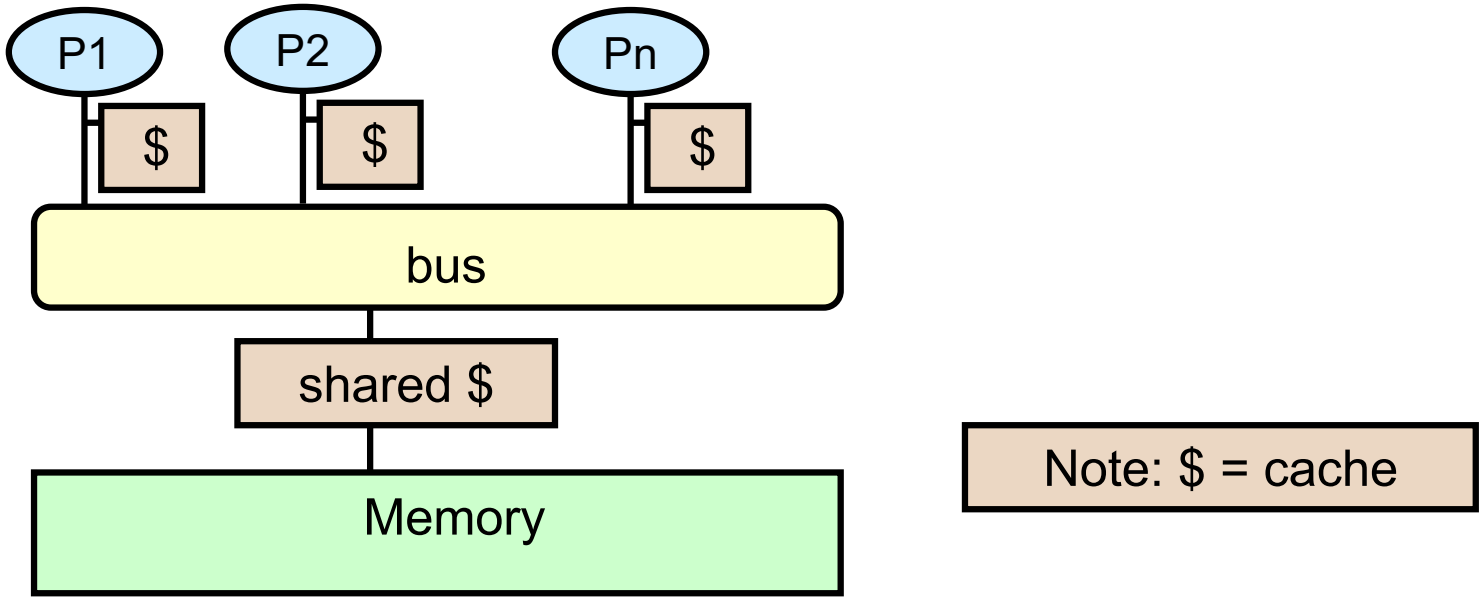
- Also known as *Symmetric Multiprocessors* (SMPs)
- SGI, Sun, HP, Intel, SMPs IBM
- Multicore processors (except that caches are shared)

Scalability issues for large numbers of processors

- Usually <= 32 processors

Advantage: Uniform memory access (*Uniform Memory Access*, UMA)

Access code: lower cost for caches compared to the main memory



Extensibility Issues for Shared Memory Architectures

Why not put more processors (with larger memory)?

- Memory bus becomes a bottleneck
- Caches must remain consistent

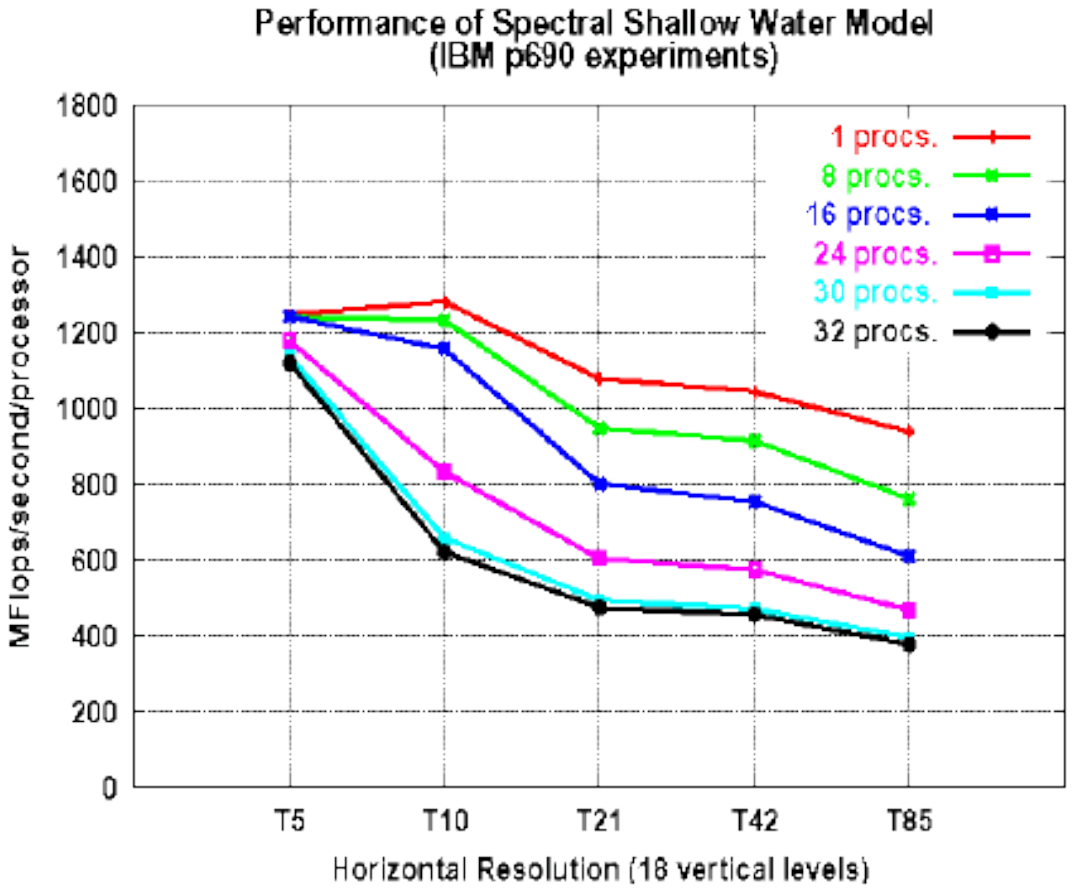
Example: Parallel Spectral Transform Shallow Water Model (PSTSWM)

- Experimental results of Pat Worley (ORNL)
- Important core of atmospheric models
  - 99% of the floating operations are additions or multiplications
  - But the code uses data on all the memory with low re-use of the loaded data (bus use and frequent shared memory)
- Experiments with sequential performance (a copy of the code running independently by increasing the number of processors used)
  - Normally the best case for shared memory: no sharing
  - But the data do not all fit in the registers and caches

Scalability Issues for Shared Memory Architectures, contd.

- Performance degradation is a function of the number of processors involved
- No data sharing between codes so perfect parallelism

- Code executed for 18 vertical levels with several horizontal sizes



Process scaling on IBM p690

OAK RIDGE NATIONAL LABORATORY
U. S. DEPARTMENT OF ENERGY

UT-BATTELLE

28

Crédits: Pat Worley, ORNL

Distributed Shared Memory

Memory is logically shared but physically distributed

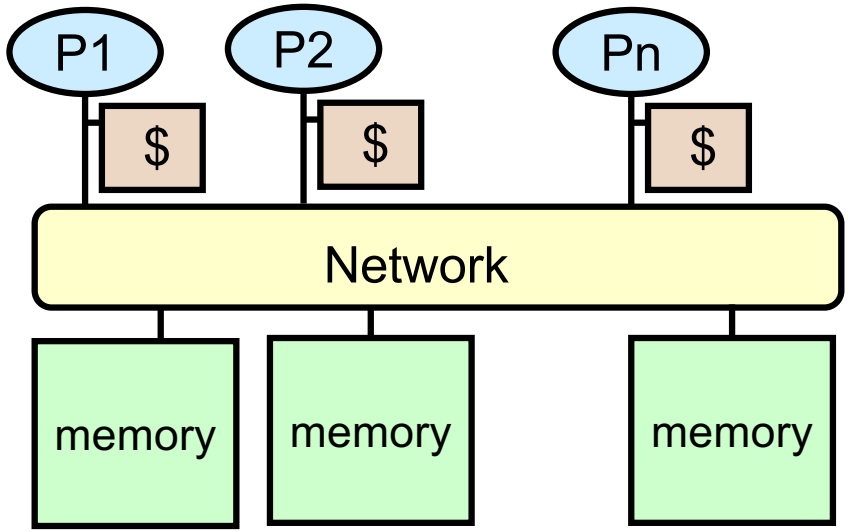
- Any processor can access any address in memory
- The lines (or pages) of cache lines are exchanged in the machine

Example: SGI platforms

- Scalable to 512 nodes (SGI Altix (Columbia) @ NASA / Ames)

Problem

- Cache Coherence Protocols
- How to maintain consistency between copies of the same memory area



The cache lines (or pages) must be large enough to cushion the overhead

➔ Locality of data critical for performance

NUMA

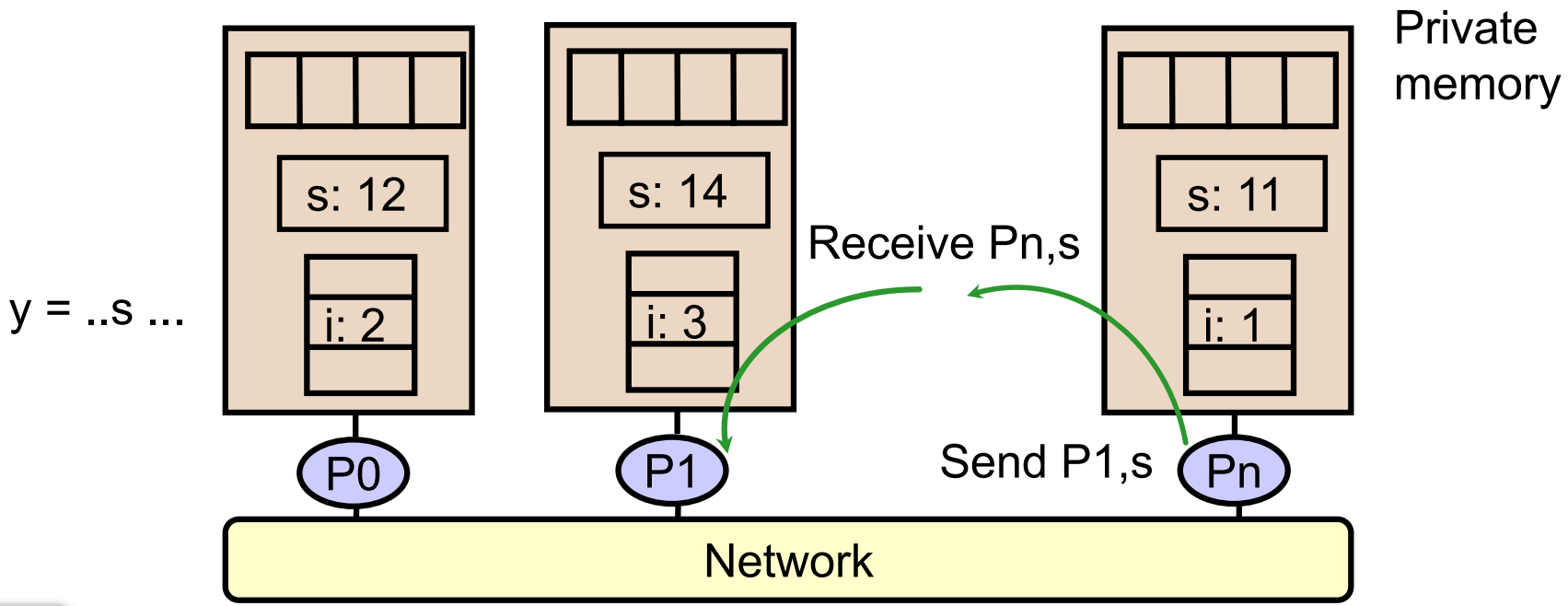
Programming model: message passing

The program consists of a set of named processes

- Generally at the start of the program
- No data sharing: a control thread and a local address space
- Data is partitioned between local processes

Processes communicate with explicit send / receive pairs

- Coordination is implicit in each communication event
- MPI (Message Passing Interface) is the most used API



Compute s = A[1]+A[2] on each processor

° First possible solution - what can crash?

```
Processor 1
    xlocal = A[1]
    send xlocal, proc2
    receive xremote, proc2
    s = xlocal + xremote
```

```
Processor 2
    xlocal = A[2]
    send xlocal, proc1
    receive xremote, proc1
    s = xlocal + xremote
```

° If the send / receive behave like the telephone system?
° Like the surface mail system?

° Second possible solution

```
Processor 1
    xlocal = A[1]
    send xlocal, proc2
    receive xremote, proc2
    s = xlocal + xremote
```

```
Processor 2
    xlocal = A[2]
    receive xremote, proc1
    send xlocal, proc1
    s = xlocal + xremote
```

° What happens if we have more processors?

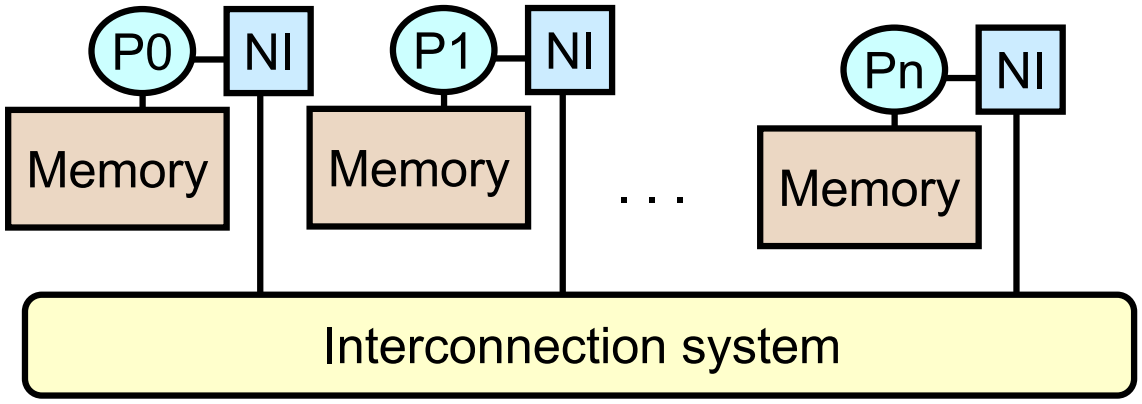
Distributed memory

Examples

- Cray XT4, XT 5
- PC clusters (Berkeley NOW, Beowulf)

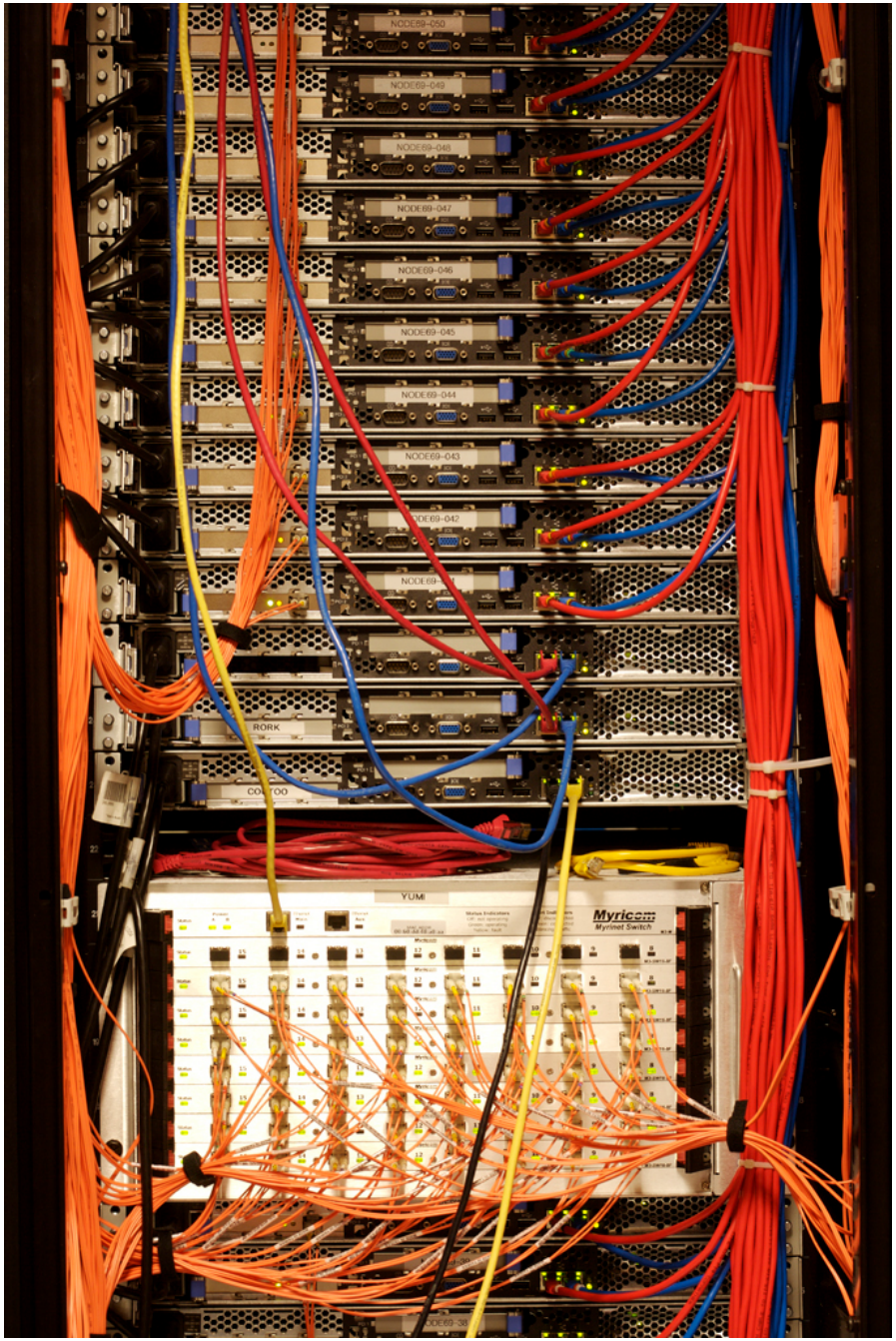
• Each processor has its own memory and cache, but can not access the memory of others

• Each "node" has its own network interface (NI) for all communications and synchronizations



Beowulf (T. Sterling)

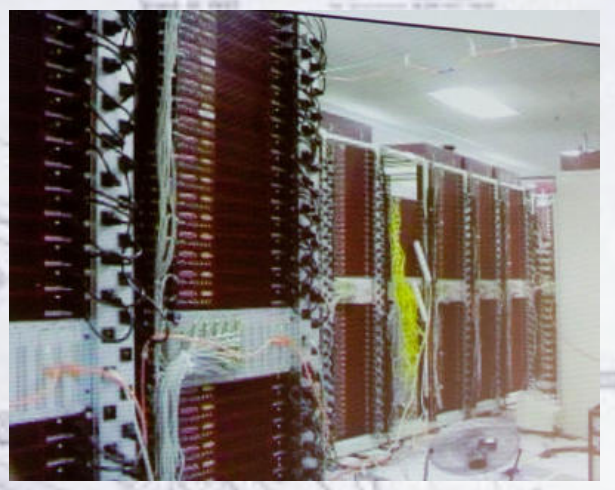

A24
10.69.137.23
OS booting
ID
select
A23
10.69.137.22
Main power on
ID
select
A22
10.69.137.21
OS booting
ID
select

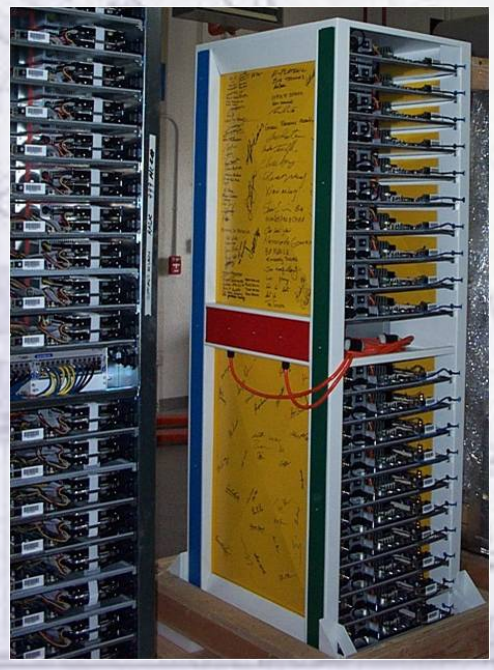

YUMI
Myricom
Myrinet Switch

Google cluster 1997

Google Data centers

- ~ 20 data centers containing more than one million servers around the world
- 40 servers / rack





facebook

Adresse électronique | Mot de passe | Se conn

Garder ma session active | Mot de passe oublié ?

Inscription Facebook vous permet de rester en contact et d'échanger avec les personnes qui vous entourent.

Articles de Facebook Engineering

Articles sur Facebook Engineering

Abonnement

Articles de Facebook Engineering

Building Efficient Data Centers with the Open Compute Project

par Jonathan Heiliger, jeudi 7 avril 2011, 10:45

OPEN Compute Project

A small team of Facebook engineers spent the past two years tackling a big challenge: how to scale our computing infrastructure in the most efficient and economical way possible.

Working out of an electronics lab in the basement of our Palo Alto, California headquarters, the team designed our first data center from the ground up; a few months later we started building it in Prineville, Oregon. The project, which started out with three people, resulted in us building our own custom-designed servers, power supplies, server racks, and battery backup systems.

Because we started with a clean slate, we had total control over every part of the system, from the software to the servers to the data center. This meant we could:

- Use a 480-volt electrical distribution system to reduce energy loss
- Remove anything in our servers that didn't contribute to efficiency
- Reuse hot aisle air in winter to both heat the offices and the outsi the data center.
- Eliminate the need for a central uninterruptible power supply.

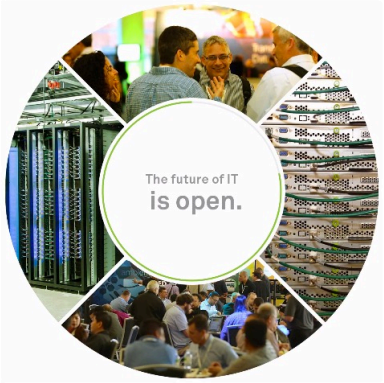
The result is that our Prineville data center uses 38 percent less energy as Facebook's existing facilities, while costing 24 percent less.

OPEN Compute Project

About | Learn | Buy | Participate | Projects | News | Contact | Sign In

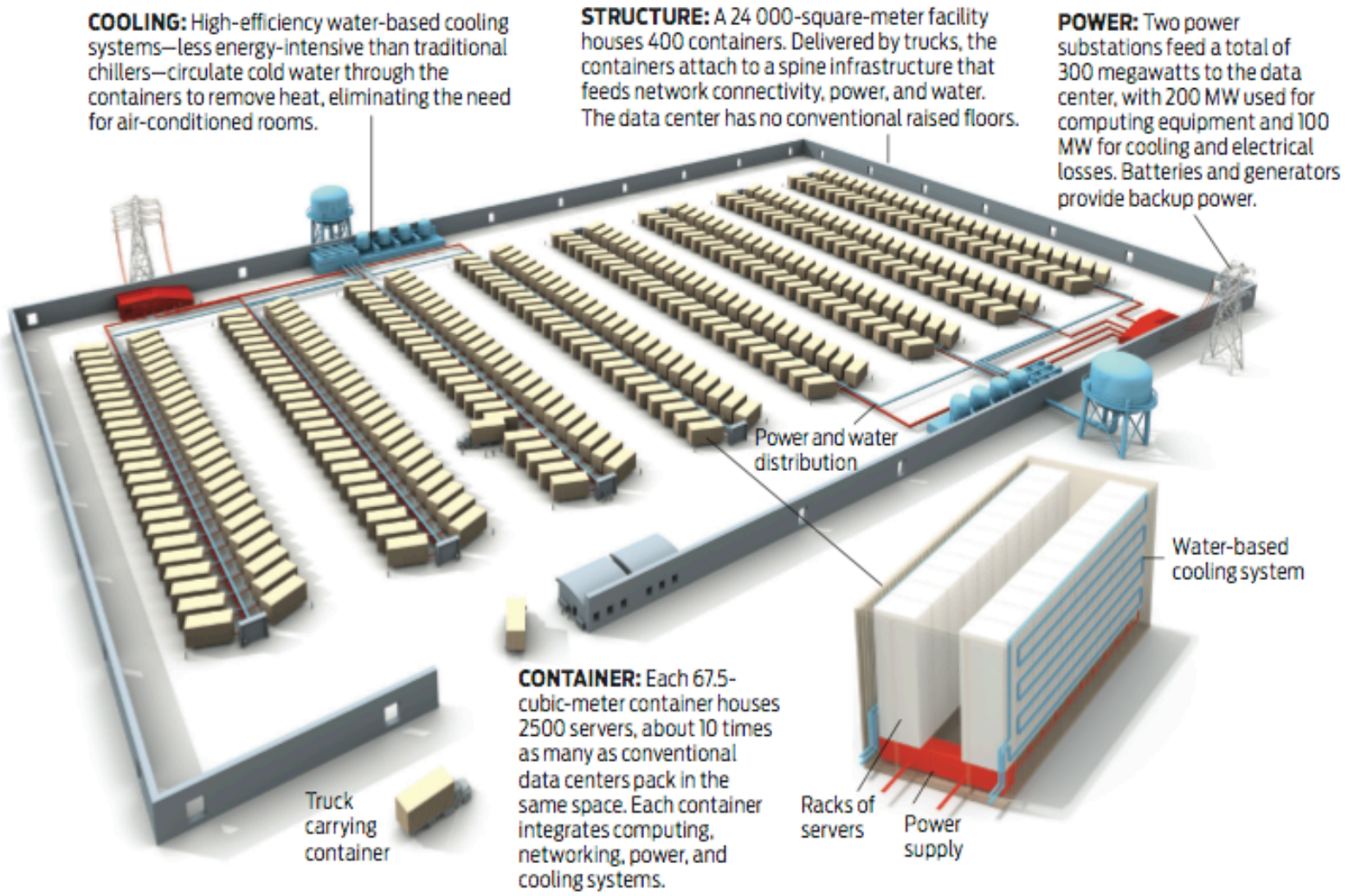
Take control of your technology future

The Open Compute Project (OCP) is reimagining hardware, making it more efficient, flexible, and scalable. Join our global community of technology leaders working together to break open the black box of proprietary IT infrastructure to achieve greater choice, customization, and cost savings.



http://opencompute.org/

The Million-Server Data Center



http://spectrum.ieee.org/tech-talk/semiconductors/devices/what-will-the-data-center-of-the-future-look-like

IBM Roadrunner (2008)

First computer to reach the Petaflops (10^{15} flops)

Roadrunner runs on

- 6,948 dual-core AMD Opteron chips on IBM Model LS21 blade servers
- 12,960 Cell engines (same as PS3) on IBM Model QS22 blade servers

With 80 terabytes of memory, the Roadrunner system and is housed in 288 IBM BladeCentre racks occupying 6,000 square feet.

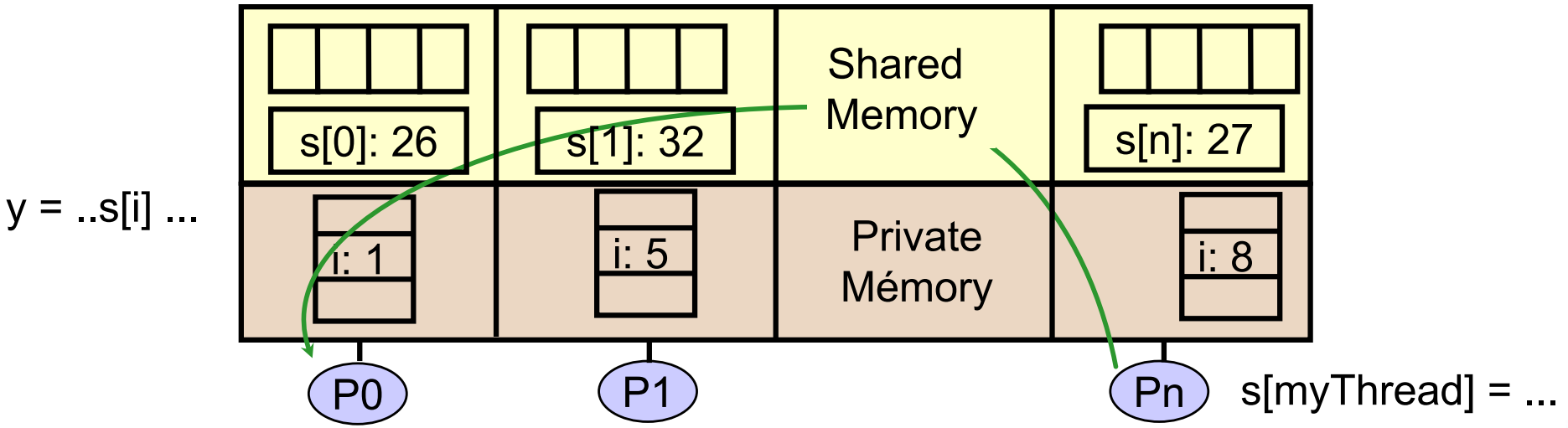
10,000 connections, both Infiniband and gigabit Ethernet, with 57 miles of fiber-optic cable.

Global address space

The program consists of a collection of named threads

- Generally set at program startup
- Local and shared data as in the shared memory model
- But the shared data is partitioned between local processors (more expensive remote access costs)
- **Examples:** UPC, Titanium, Co-Array Fortran

- Intermediate between shared memory and message passing



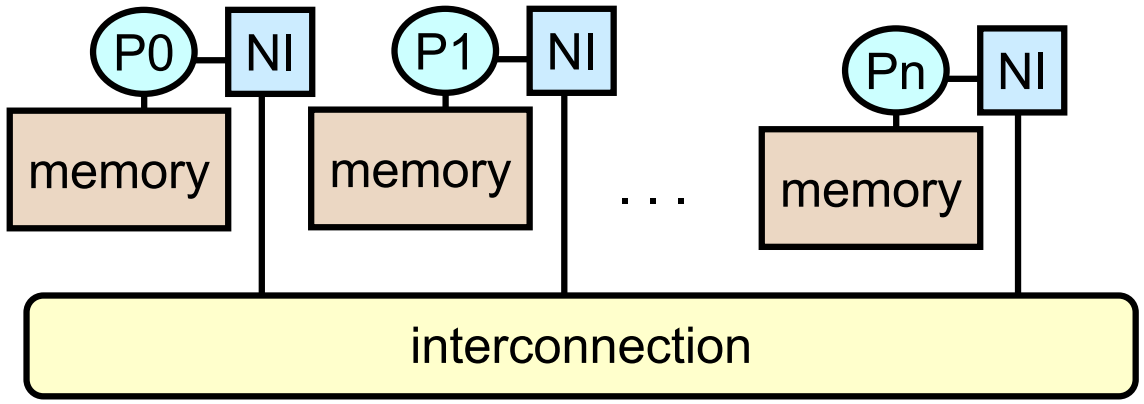
Global address space, contd.

Examples

- Cray T3D, T3E, X1 and HP Alphaserver clusters
- Clusters built with Quadrics, Myrinet, or Infiniband networks

The network interface supports RDMA (Remote Direct Memory Access)

- NIs can directly access the memory without interrupting the CPU
- A processor can read / write to memory with one-sided (put / get) operations,
- Not just a load / store on a shared memory machine
  - Continue computing until memory operation completes
- The "remote" data is usually not cached locally



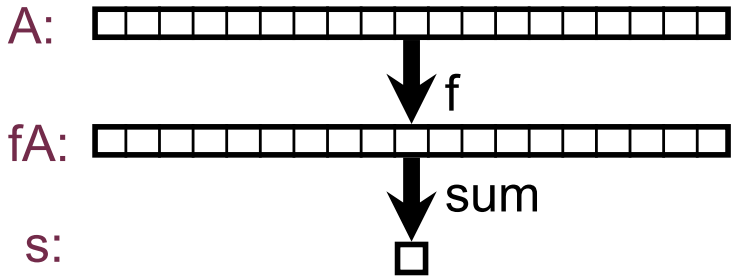
Data-parallel programming models

Data-parallel programming model

- Implicit communications in parallel operators
- Easy to understand and model
- Implicit coordination (instructions executed synchronously)
- Close to Matlab for array operations

- **Drawbacks**
  - Does not work for all models
  - Difficult to port on coarse-grained architectures

A = data array
fA = f(A)
s = sum(fA)



Vector machines

Based on a single processor

- Several functional units
- All performing the same operation

- Exceeded by MPP machines in the 1990s

Come-back since the last ten years

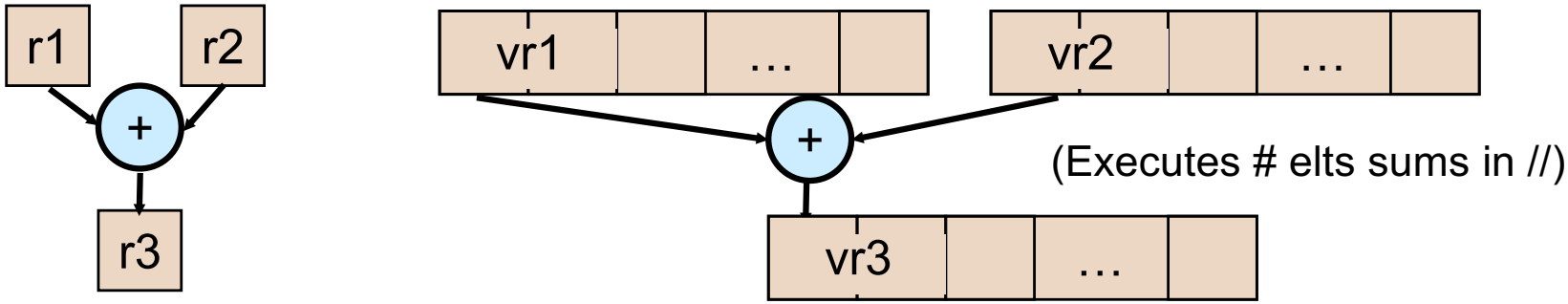
- On a large scale (Earth Simulator (NEC SX6), Cray X1)
- On a smaller scale, processor SIMD extensions
  - SSE, SSE2: Intel Pentium / IA64
  - Altivec (IBM / Motorola / Apple: PowerPC)
  - VIS (Sun: Sparc)
- On a larger scale in GPUs

Key idea: the compiler finds parallelism!

Vector processors

Vector instructions execute on an element vector

- Specified as operations on vector registers

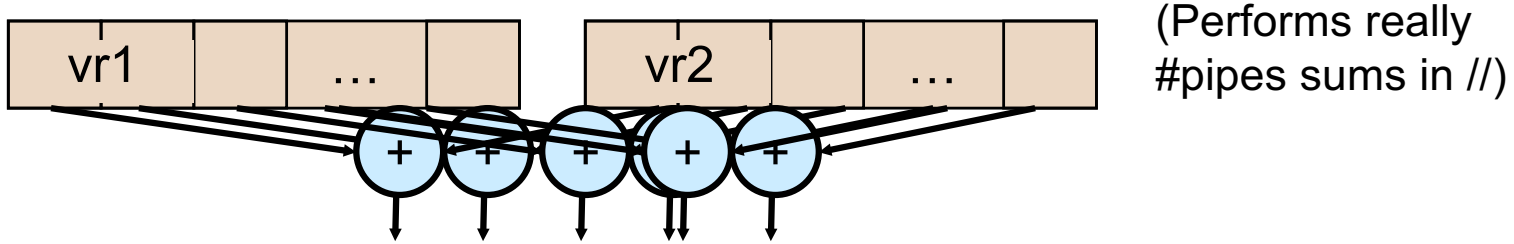


A register contains ~ 32-64 elements

- The number of elements is greater than the number of parallel units (vector pipes/lanes, 2-4)

The speed for a vector operation is

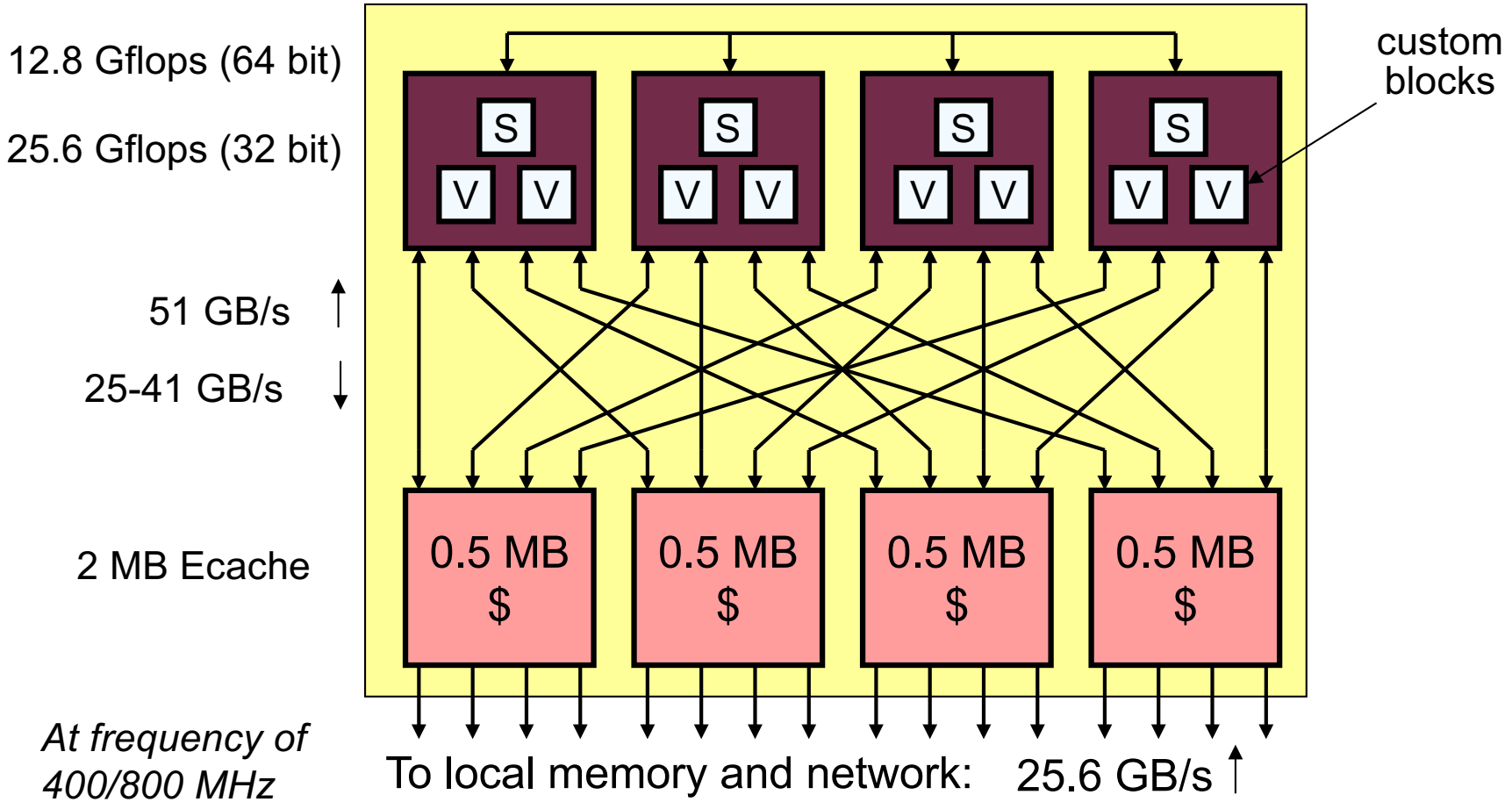
#elements-per-vector-register / #pipes



Credits: J. Levesque, Cray

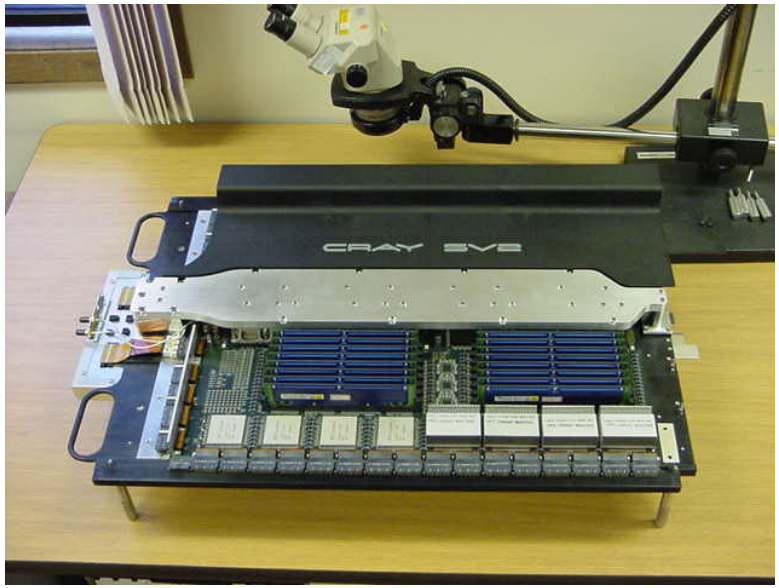
Cray X1 node

- Cray X1 builds a larger “virtual vector”, called an MSP
  - 4 SSPs (each a 2-pipe vector processor) make up an MSP
  - Compiler will (try to) vectorize/parallelize across the MSP



Cray X1: Parallel Vector Architecture

- **Cray combines several technologies in the X1**
  - 12.1 Gflop / s Vector Processors
  - Shared Caches
  - Nodes with 4 processors sharing up to 64 GB of memory
  - Single System Image for 4096 processors
  - Put / get operations between nodes (faster than MPI)



Hybrid machines

Multicore / SMPs nodes used as LEGO elements to build machines with a network

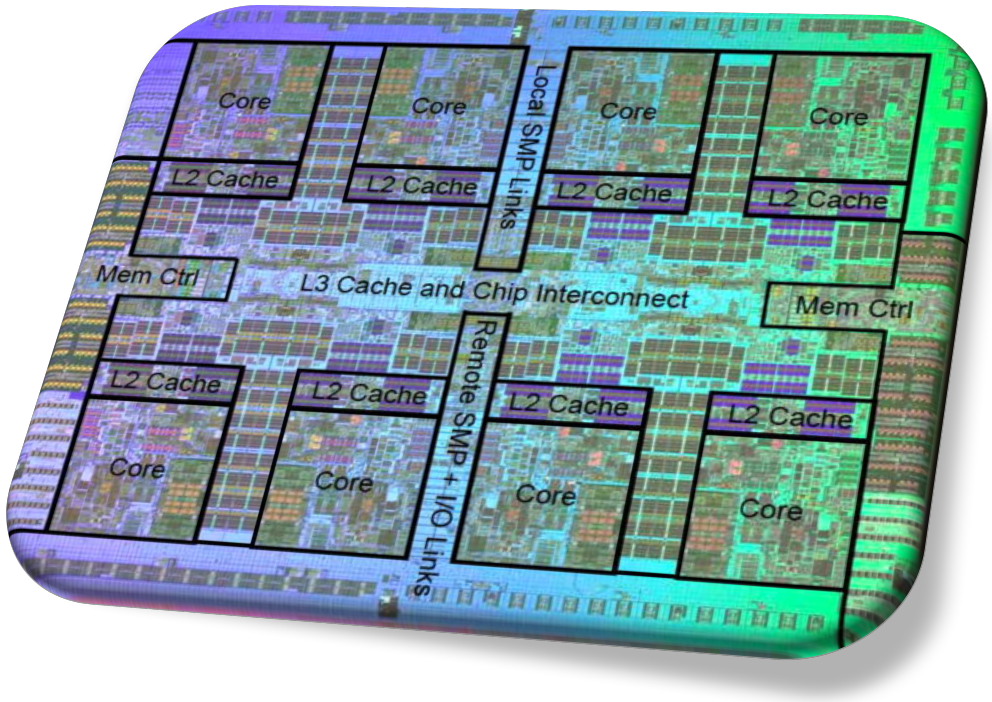
Called CLUMP (*Cluster of SMPs*)

Examples

- Millennium, IBM SPs, NERSC Franklin, Hopper

- Programming Model
  - Program the machine as if it was on a level with MPI (even if there is SMP)
  - Shared memory within an SMP and passing a message outside of an SMP

- Graphic (co) -processors can also be used



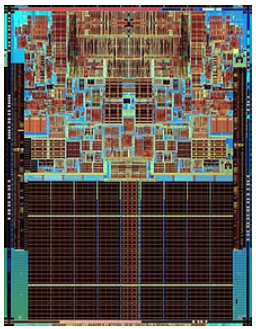
MULTICORES/GPU

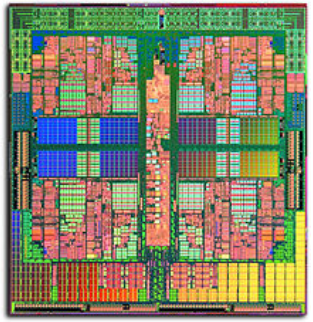
Multicore architectures

- A processor composed of at least 2 central processing units on a single chip
- Allows to increase the **computing power** without increasing the **clock speed**
- And therefore **reduce heat dissipation**
- And to **increase the density**: the cores are on the same support, the connectors connecting the processor to the motherboard does not change compared to a single core

Why multicore processors?

Quelques ordres de grandeur			
	Single Core Engraving generation 1	**Dual Core** Engraving generation 2	**Quad Core** Engraving generation 3
Core area	A	~ A/2	~ A/4
Core power	W	~ W/2	~ W/4
Chip power	W + O	W + O’	W + O”
Core performance	P	0.9P	0.8P
Chip performance	P	1.8 P	3.2 P





Nehalem-EP architecture (Intel)

4 cores

On-chip L3 cache shared (8 Mo)

3 cache levels

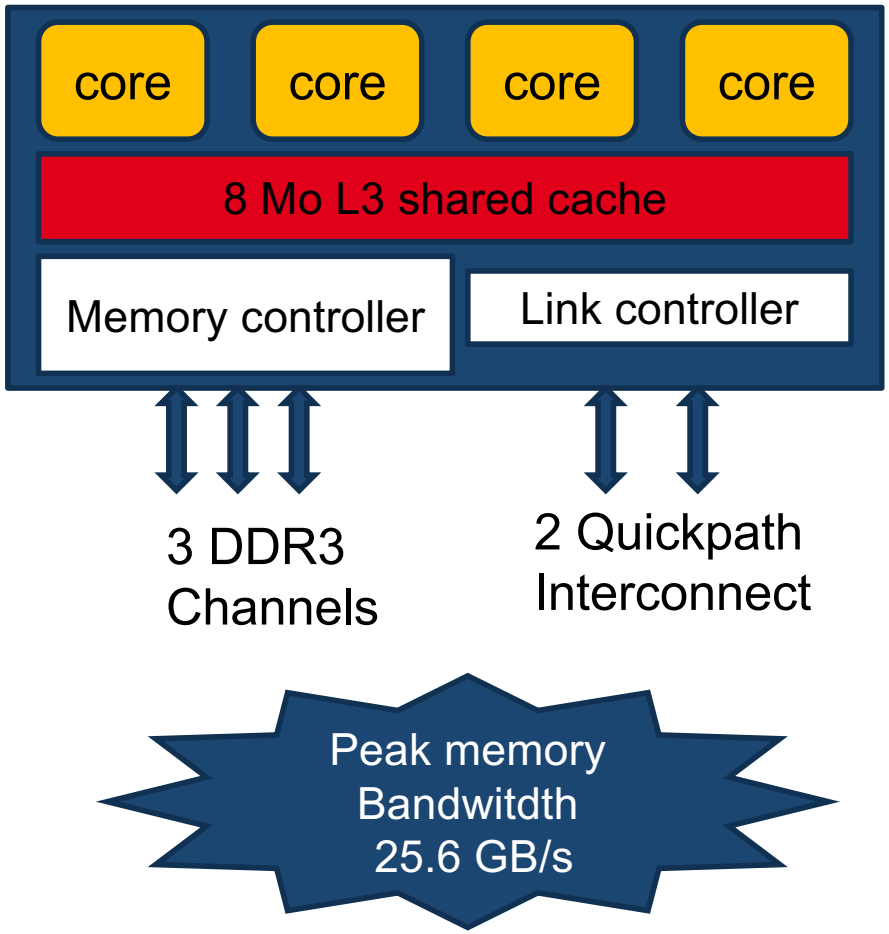
- Cache L1 : 32k I-cache + 32k D-cache
- Cache L2 : 256 k per core
- Inclusive cache: on-chip cache coherency (SMT)

732 M transistors, 1 single die (263 mm^2)

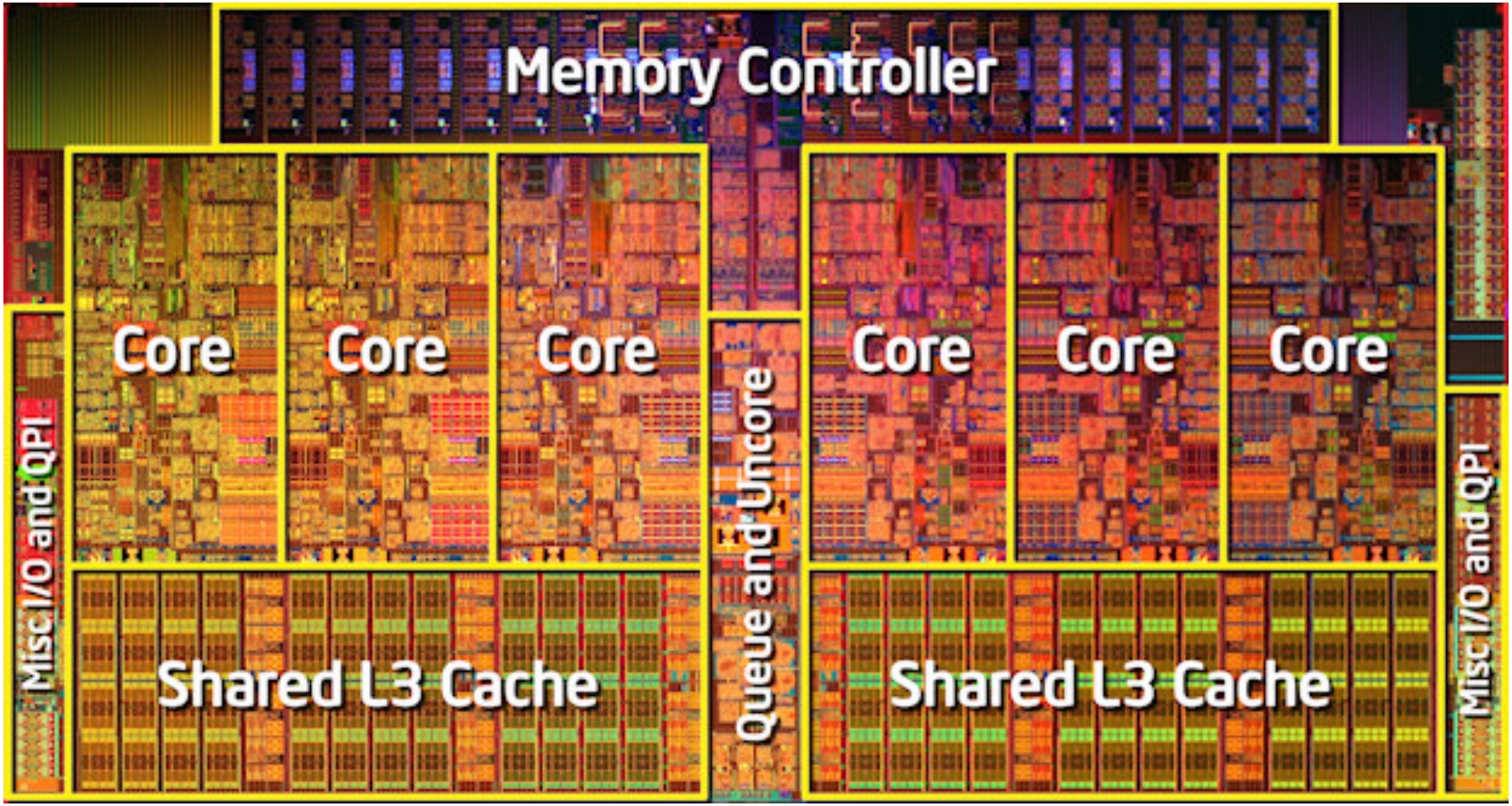
QuickPathInterconnect

- Point-to-point
- 2 links per CPU socket
- 1 for the connection to the other socket
- 1 for the connection to the chipset

Integrated QuickPath Memory controller (DDR3)

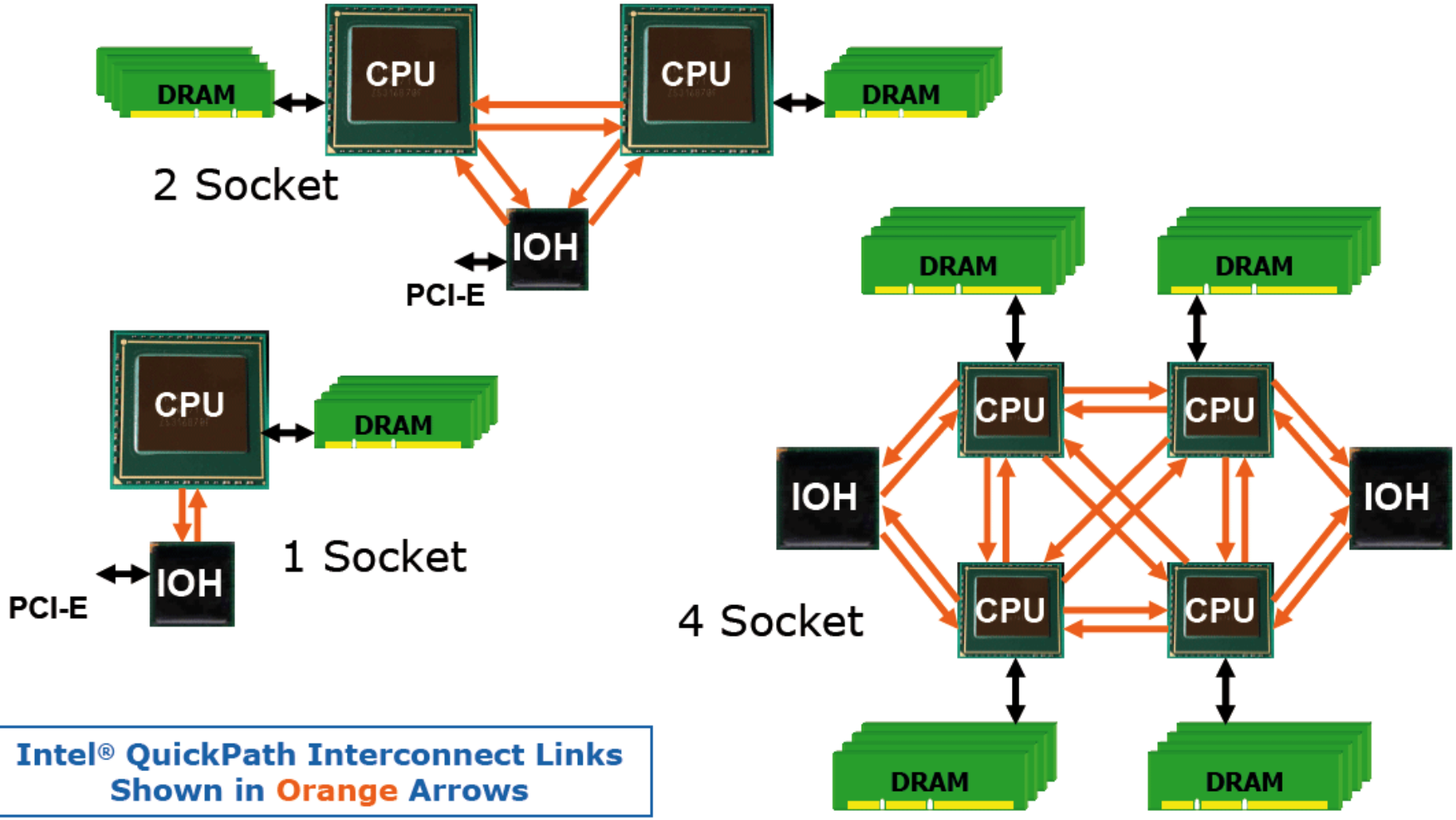


Nehalem 8 cores



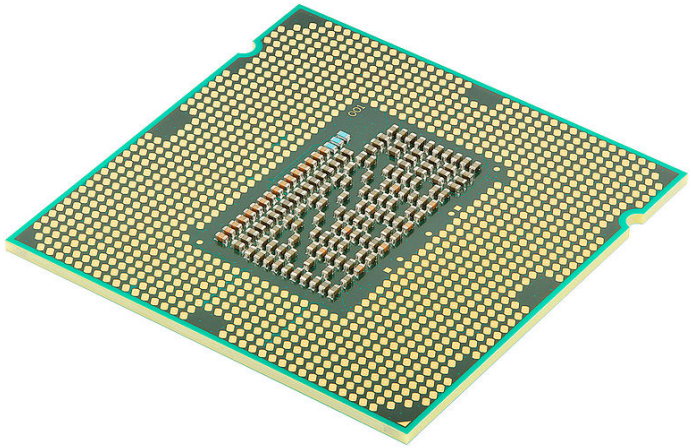
Nehalem

Example Platform Topologies



Intel® QuickPath Interconnect Links Shown in Orange Arrows

Sandy Bridge-EP architecture



Early 2012 with

- 8 cores per processor
- 3 cache levels

  L1 cache: 32k I-cache + 32k D-cache

  L2 cache: 256 k / core, 8 voies associative

  L3 cache: shared and inclusive (16 Mo on-chip)
- 4 DDR3 memory controller
- AVX instructions → 8 flop DP/cycle (twice of the Nehalem)
- 32 lines PCI-e 3.0
- QuickPathInterconnect

  2 QPI per proc

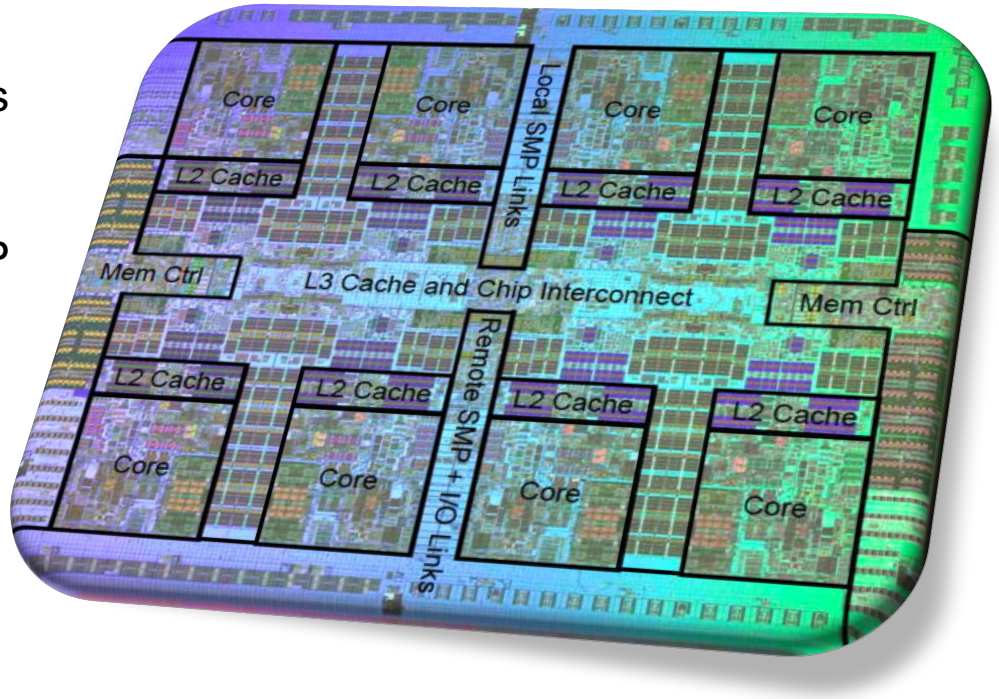
Power7 Architecture

- Cache controller L3 and memory on-chip
- Up to 100 Go/s of memory bandwidth

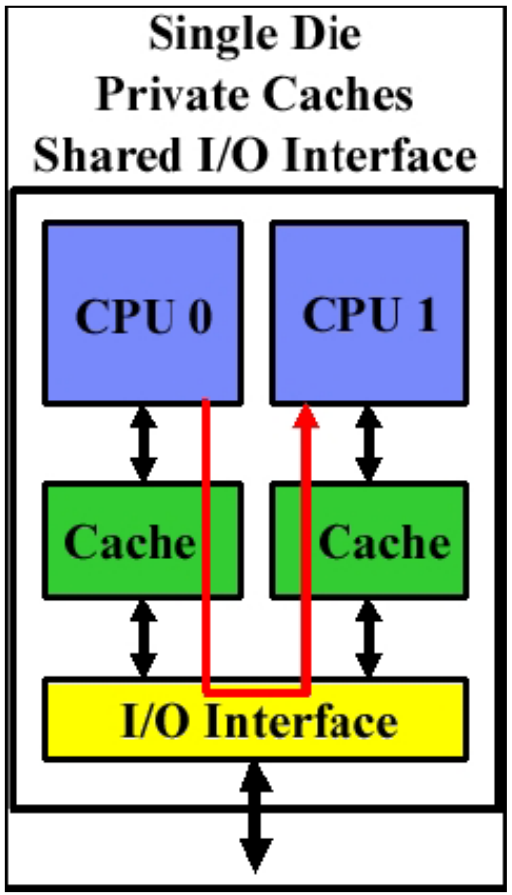
- 1200 M transistors, 567 mm^2 per die
- up to 8 cores
- 4 way SMT ⇨ up to 32 simultaneous threads
- 12 execution units, including 4 FP
- Scalability: up to 32 8-cores sockets per SMP system , ↗ 360 Go/s of chip bandwidth

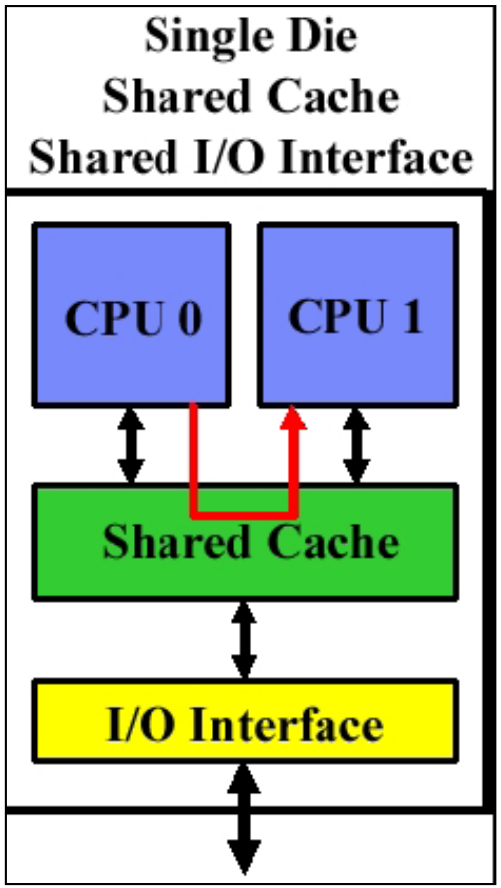
⇨ Up to 1024 threads /SMP

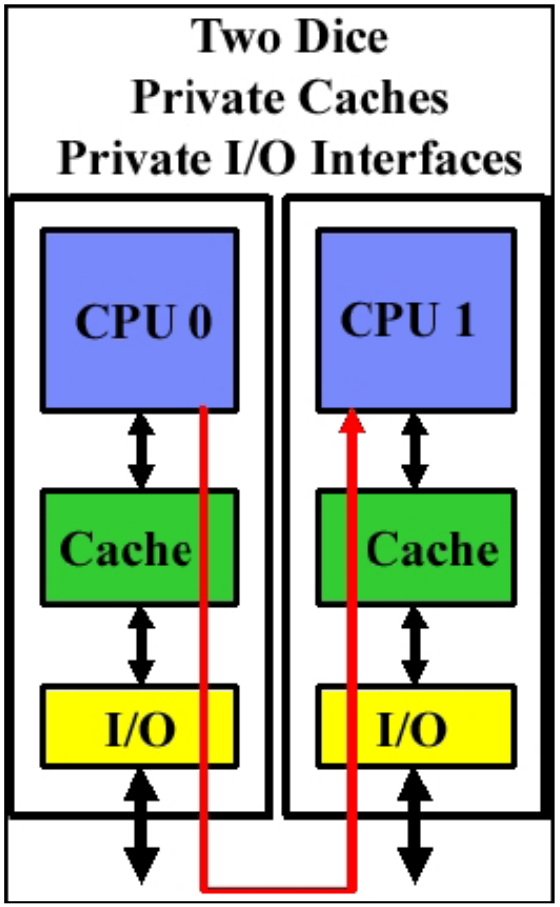
- 256Ko L2 cache /core
- L3 cache shared using partagé in eDRAM technology (embeddedDRAM)



Caches architectures







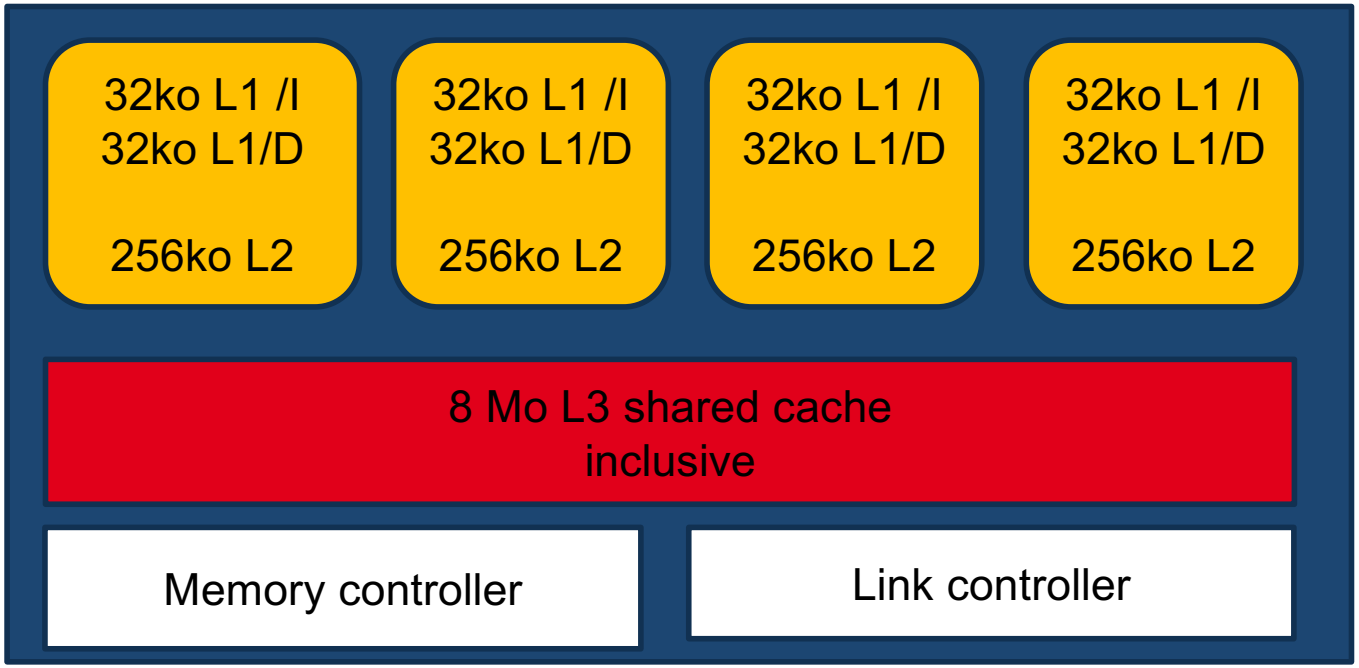
Sharing L2 and L3 caches

- **Sharing the L2 cache (or L3)**
  - ✓ ☺ Faster communication between cores,
  - ✓ ☺ better use of space,
  - ✓ ☺ thread migration easier between cores,
  - ✓ ☹ contention at the bandwidth level and the caches (space sharing),
  - ✓ ☹ coherency problem.

- **No cache sharing**
  - ✓ ☺ no contention,
  - ✓ ☹ communication/migration more costly, going through main memory.

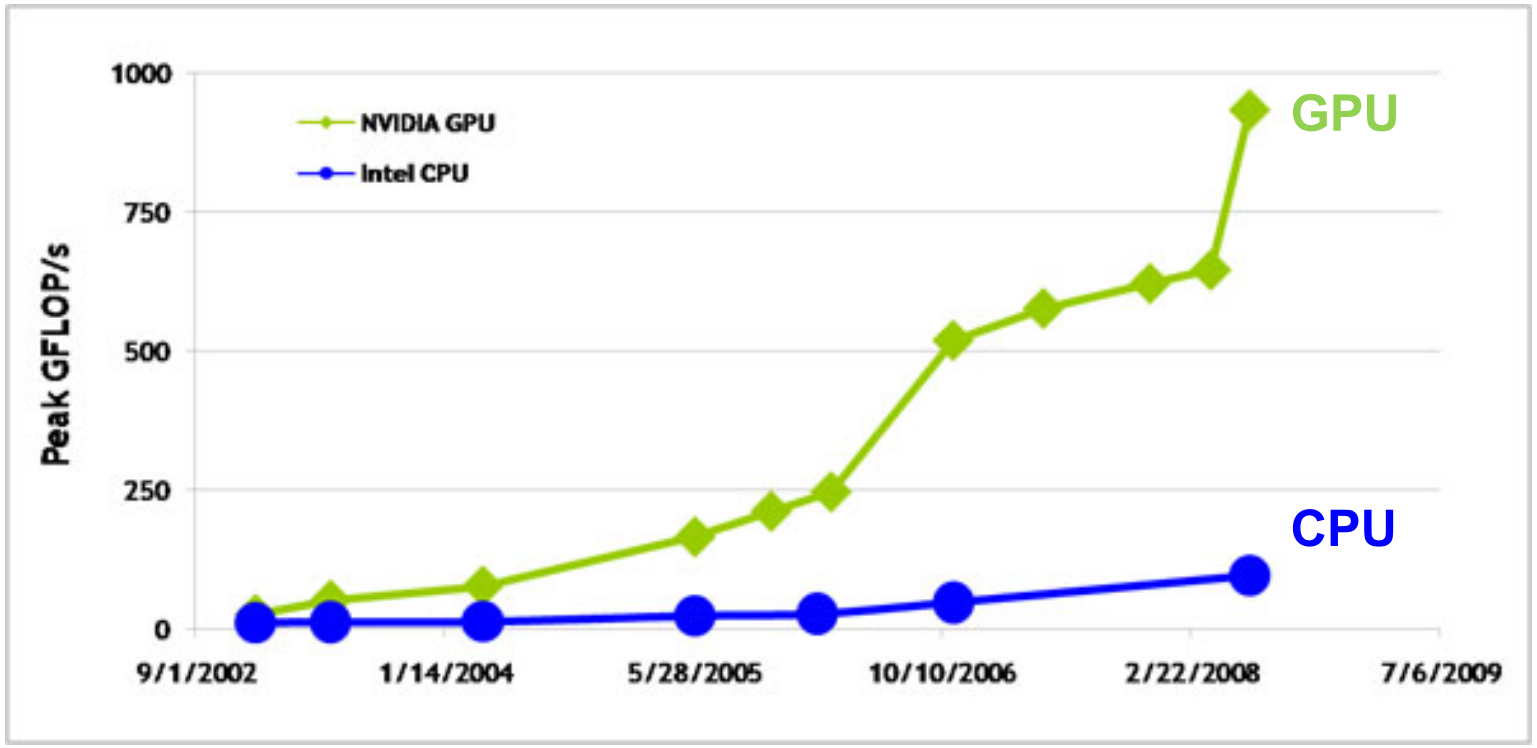
    - Private L2, shared L3 cache: IBM Power5+ / Power6, Intel Nehalem
    - All private: Montecito

Nehalem example: A 3 level cache hierarchy



- L3 cache inclusive of all other levels
  - 4 bits allow to identify in which processor's cache the data is stored
  - ✓ ☺ traffic limitation between cores
  - ✓ ☹ Waste of one part of the cache memory

Performance evolution: CPU vs GPU

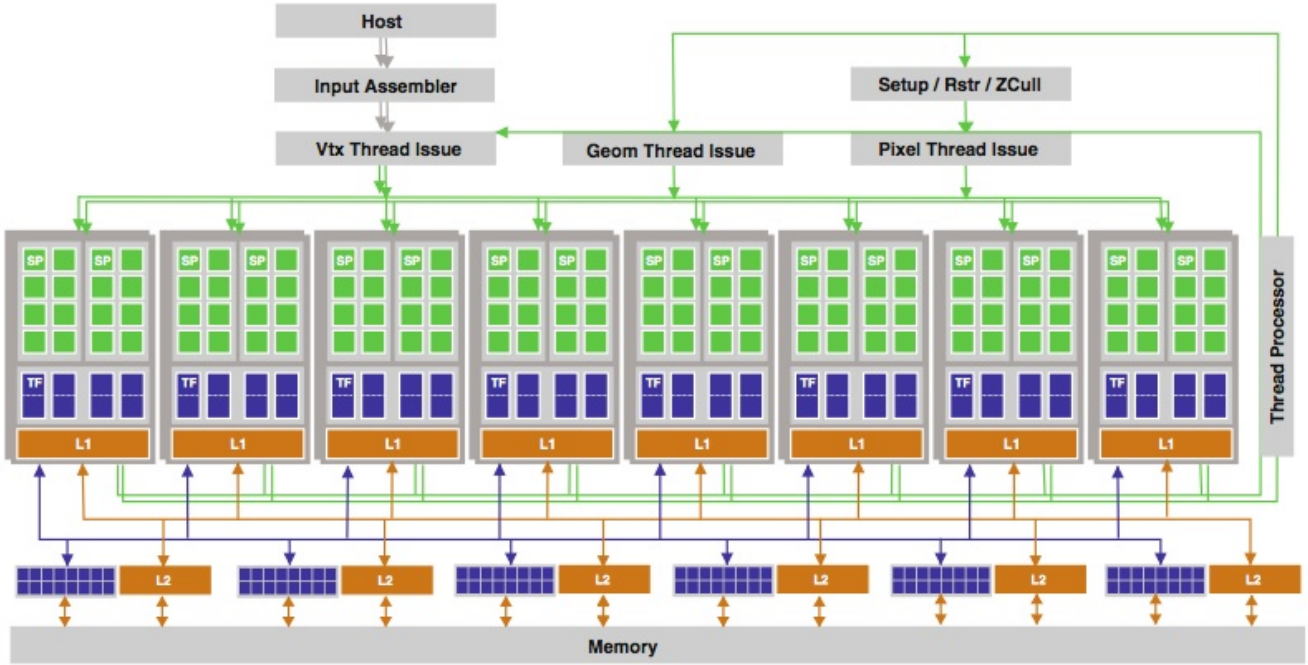


"classical" processors' speed increase *** 2 every 16 months**

GPU processors' speed increase ***2 every 8 months**

GPU

- Theoretical performance GeForce 8800GTX vs Intel Core 2 Duo 3.0 GHz:
  367 Gflops / 32 GFlops
- Memory bandwidth: 86.4 GB/s / 8.4 GB/s
- Available in every workstations/laptops: mass market
- Adapted to massive parallelism (thousands of threads per application)
- Until 10 years, only programmed using graphic APIs
- Now many programming models available
  - CUDA , OpenCL, HMPP, OpenACC



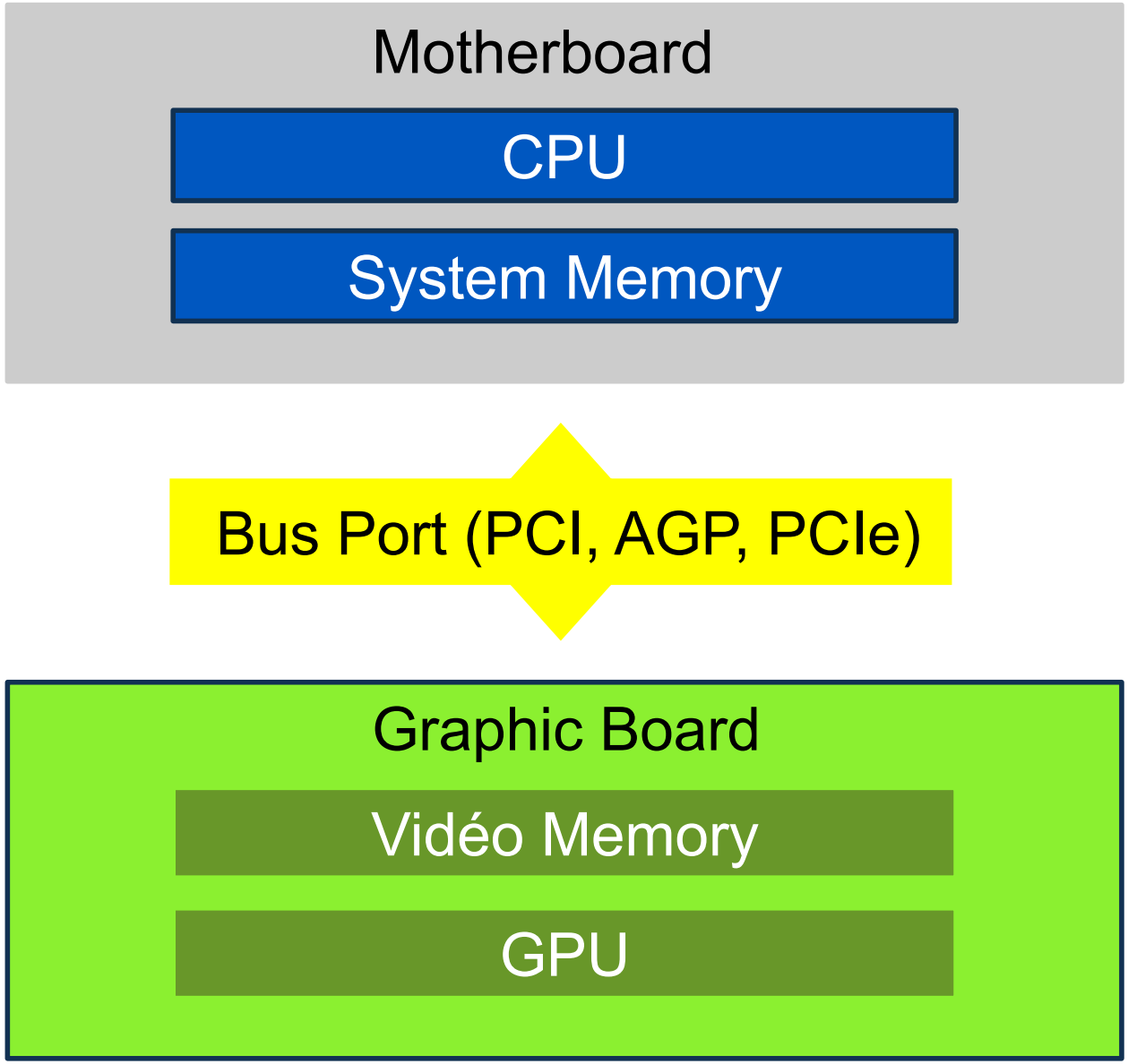
Fermi graphic processor

Major evolutions for HPC

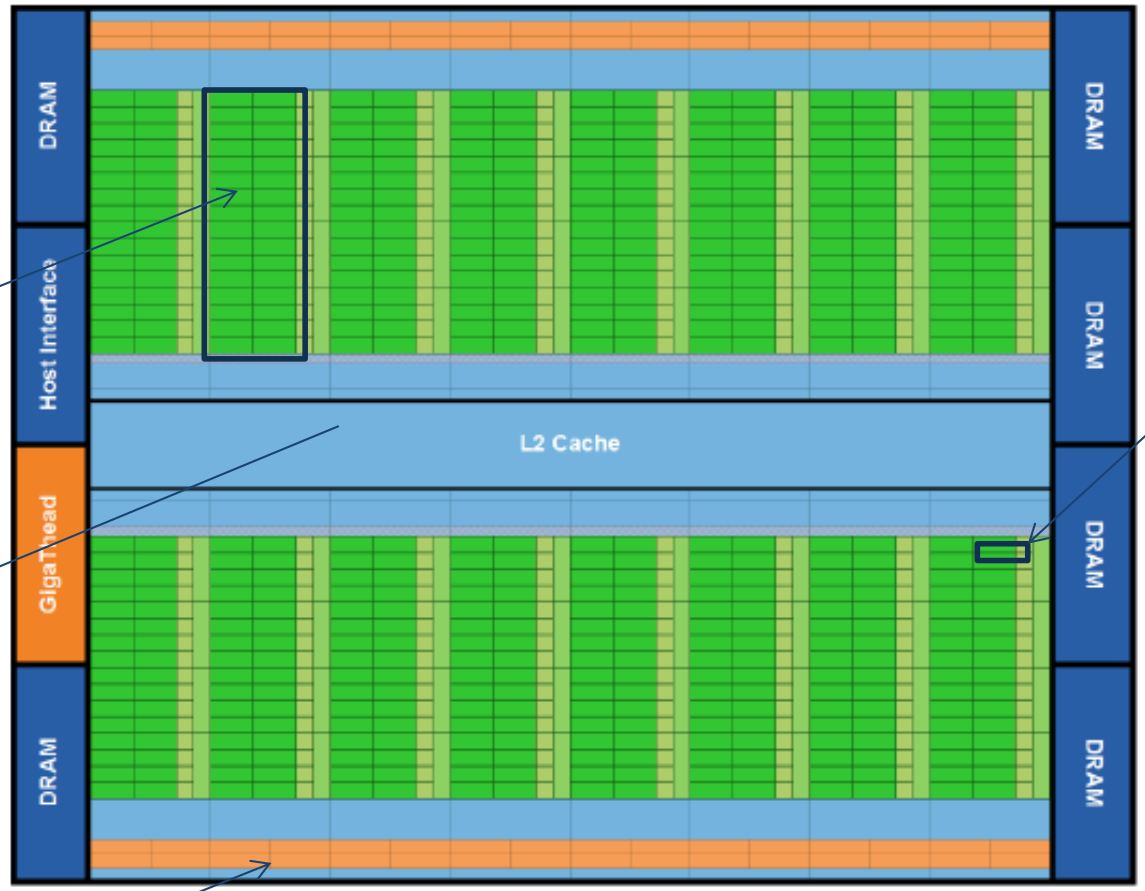
- Floating point operations: IEEE 754-2008 SP & DP
- ECC support (Error Correction Coding) on every memory

- 256 FMAs DP/cycle
- 512 cores
- L1 et L2 cache memory hierarchy
- 64 KB of L1 shared memory (on-chip)
- Up to 1 TB of GPU memory

Classical PC architecture



NVIDIA Fermi processor architecture



SM
Multiprocessor

core

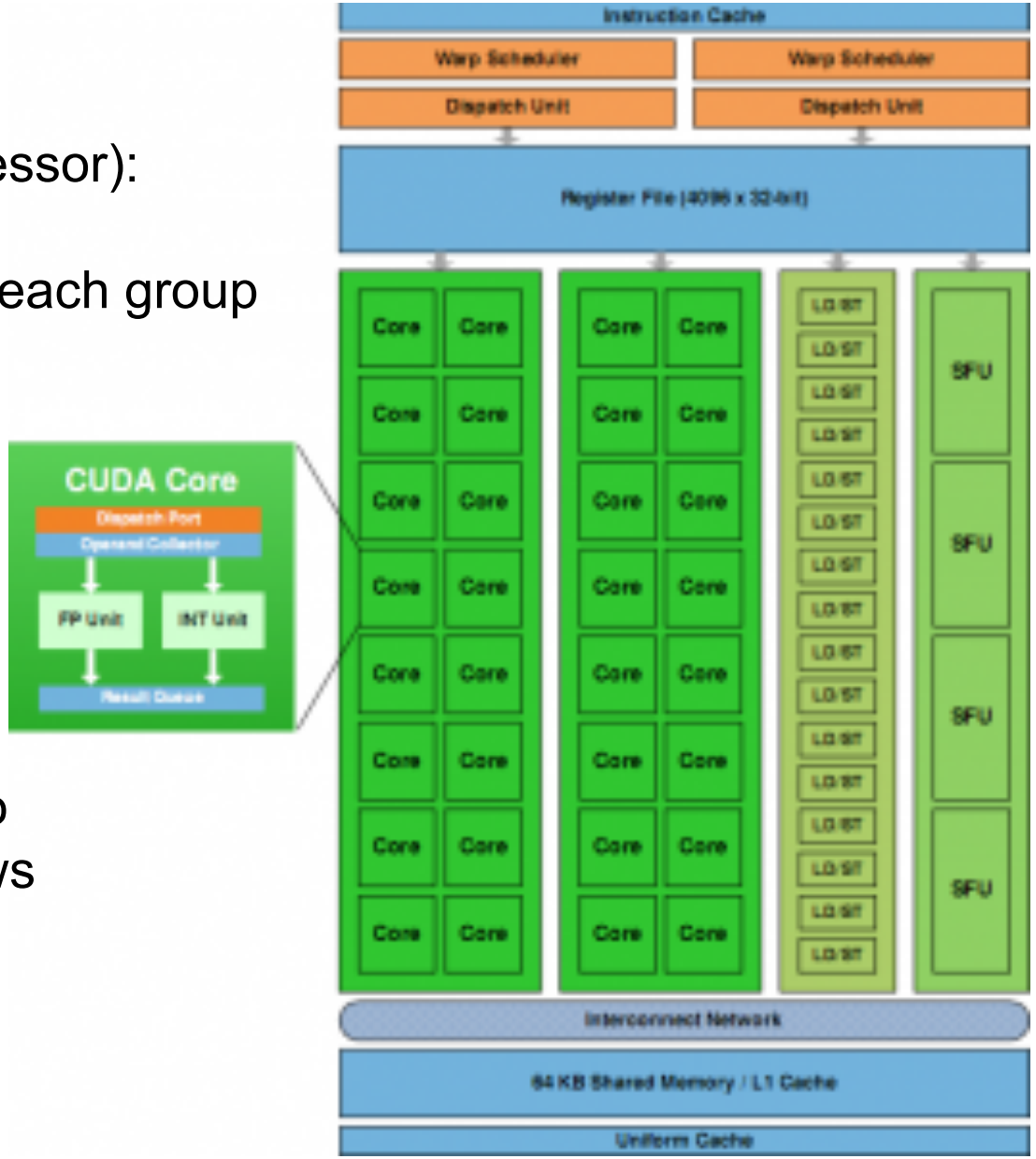
768 Ko L2 Cache

Shared memory / L1 cache
64 Ko/SM

512 Compute units
16 (32) partitions

NVIDIA Fermi processor architecture

Fermi *SM* (Streaming Multiprocessor):
Each *SM* has 32 cores
A *SM* schedules the threads for each group of 32 threads //



An important evolution
64 Ko of on-chip memory (48 ko shared mem + 16ko L1). It allows threads of a same block to cooperate.
64 bit units

GPU /CPU Comparaison

With equal performance, platforms based on GPUs

- Occupy less space
- Are cheaper
- Consume less energy

But

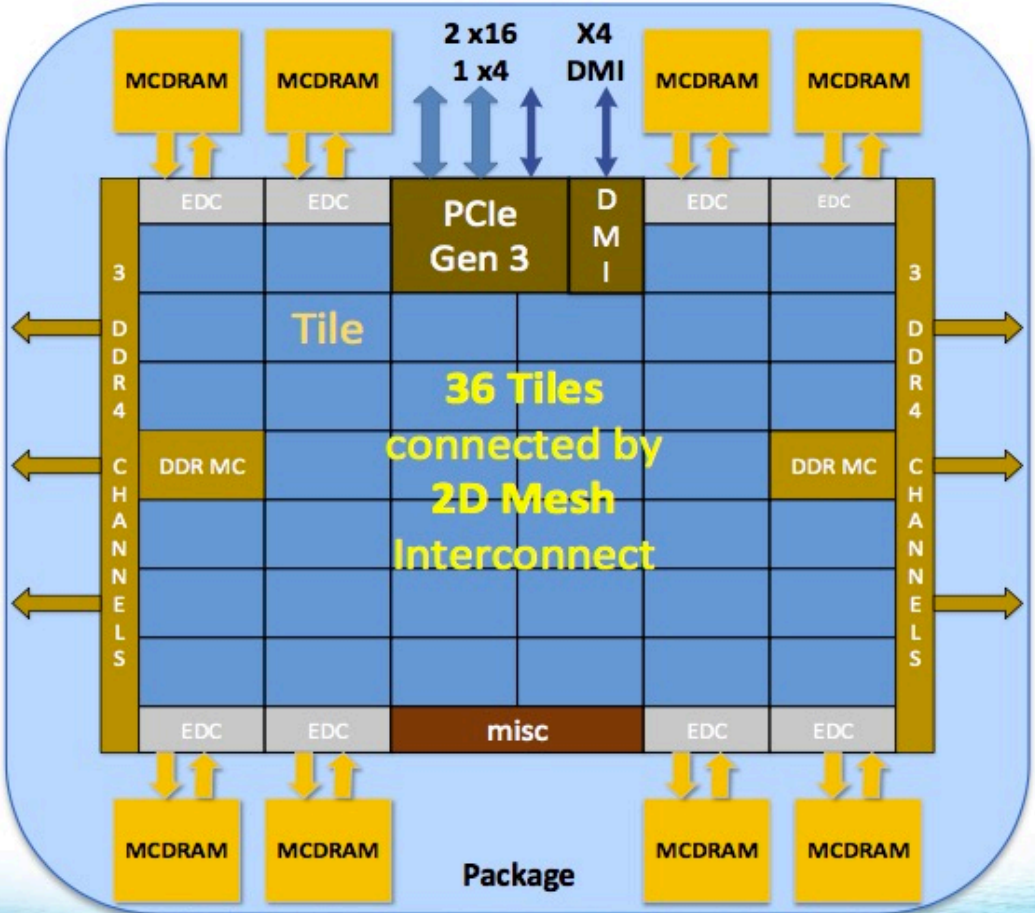
- Are reserved for massively parallel applications
- Require to learn new tools
- What is the guarantee of the durability of the codes and therefore of the investment in terms of application port?

Intel's Many Integrated Core processors: A response to the GPU?

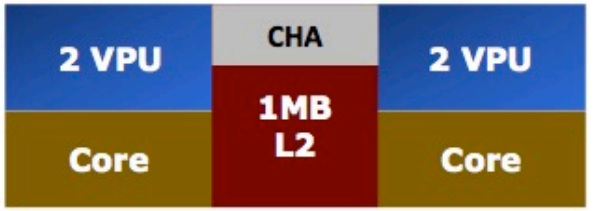
- Manycores processors « », ≥ 50 cores on the same chip
- X86 Compatibility
  - Intel software support
- Xeon Phi in June 2012
  - 60 cores/1.053 GHz/240 threads
  - 8 GB memory and 320 GB/s of bandwidth
  - 1 teraflops !

Knights Landing Intel Xeon Phi

Knights Landing Overview



Avinash Sodani CGO PPoPP HPCA Keynote 2016



Chip: up to **36 Tiles** interconnected by **2D Mesh**

Tile: 2 Cores + 2 VPU/core + 1 MB L2

Memory: MCDRAM: up to 16 GB on-package; High BW

DDR4: 6 channels @ 2400 up to 384GB

IO: 36 lanes PCIe Gen3. 4 lanes of DMI for chipset

Node: 1-Socket

Fabric: Intel® Omni-Path Fabric on-package (not illustrated)

Vector Peak Perf: 3+TF DP and 6+TF SP Flops

Scalar Perf: ~3x over Knights Corner

Streams Triad (GB/s): MCDRAM : 450+; DDR: ~90

Note: not all specifications shown apply to all Knights Landing SKUs

Source Intel: All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice. KNL data are preliminary based on current expectations and are subject to change without notice. 1Binary Compatible with Intel Xeon processors using Haswell Instruction Set (except TSX). 2Bandwidth numbers are based on STREAM-like memory access pattern when MCDRAM used as flat memory. Results have been estimated based on internal Intel analysis and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.

Kalray MPPA-256 overview

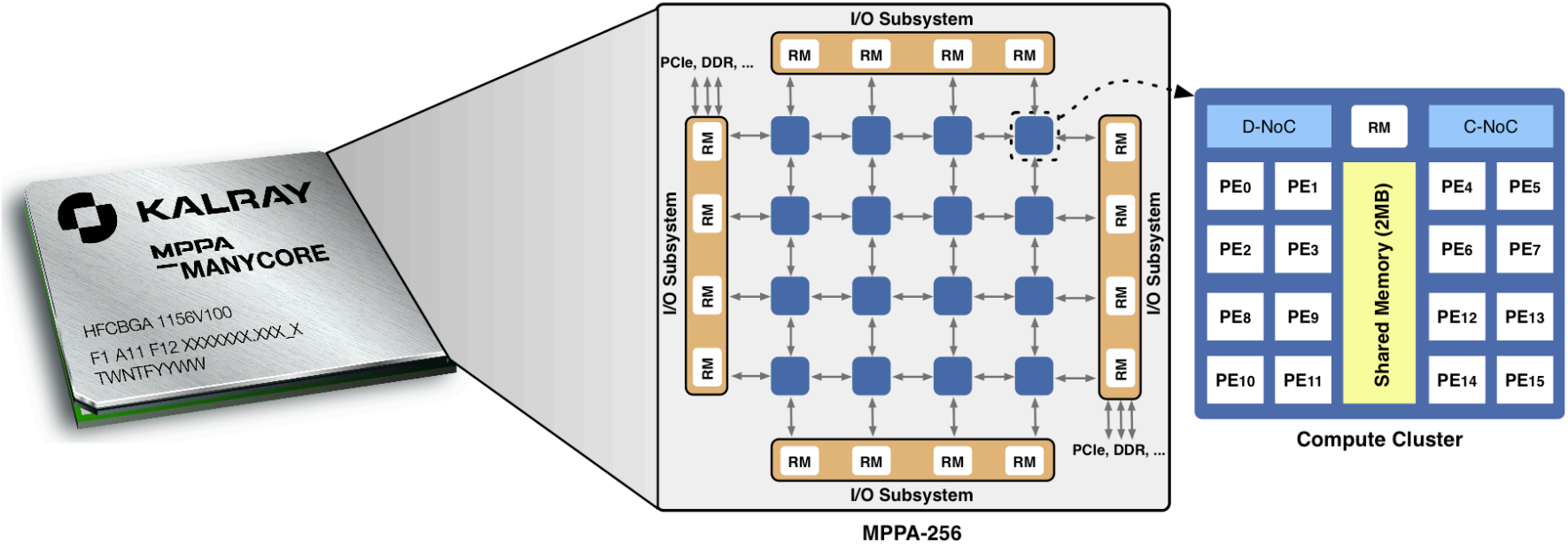
Kalray

- French semiconductor and software company developing and selling a new generation of manycore processors for HPC

MPPA-256

- Multi-Purpose Processor Array (MPPA)
- Manycore processor: 256 cores in a single chip
- Low power consumption (5W - 11W)

Kalray MPPA-256 overview



256 cores (PEs) @ 400 MHz: 16 clusters, 16 PEs per cluster

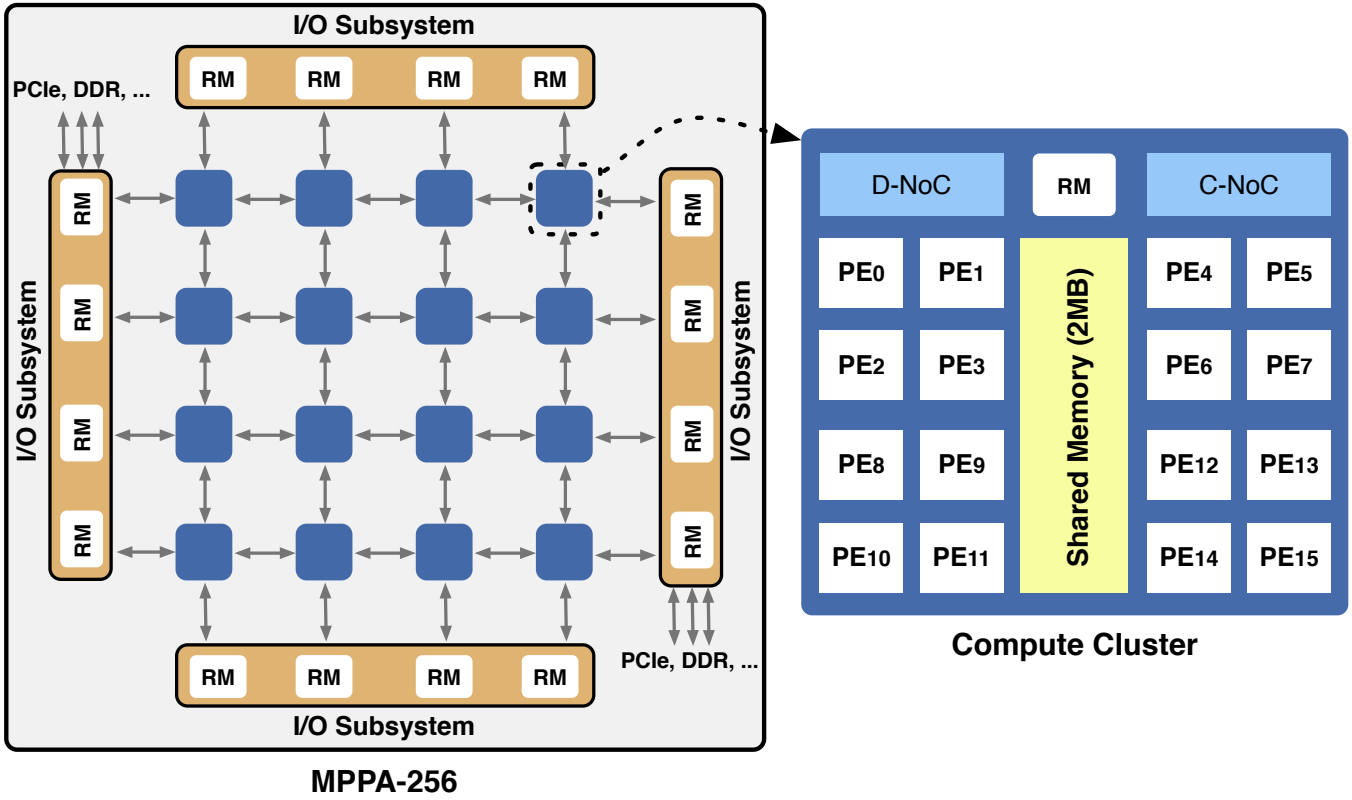
PEs **share 2 MB of memory**

Absence of cache coherence protocol inside the cluster

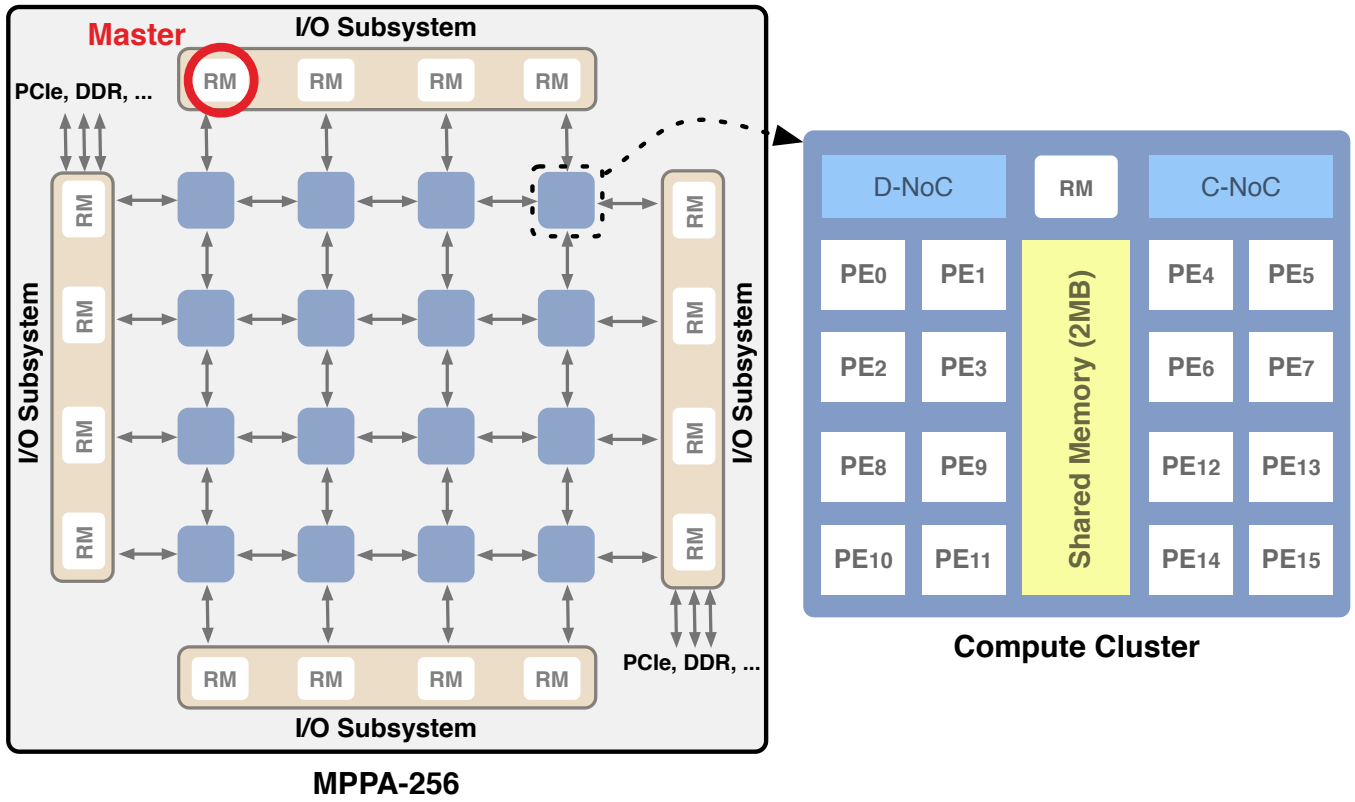
Network-on-Chip (NoC): communication between clusters

4 I/O subsystems: 2 connected to external memory

Kalray MPPA-256 overview

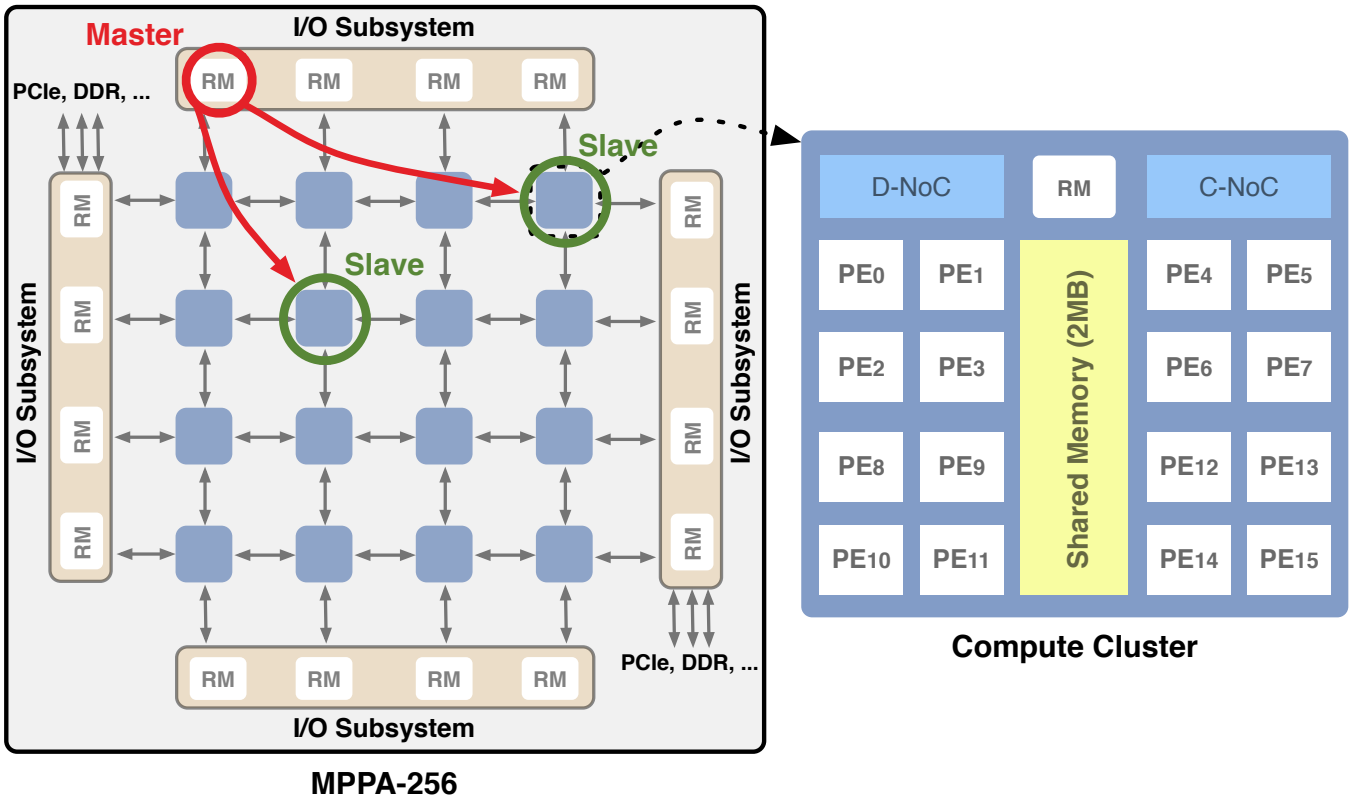


Kalray MPPA-256 overview



A **master** process runs on an **RM** of one of the **I/O subsystems**

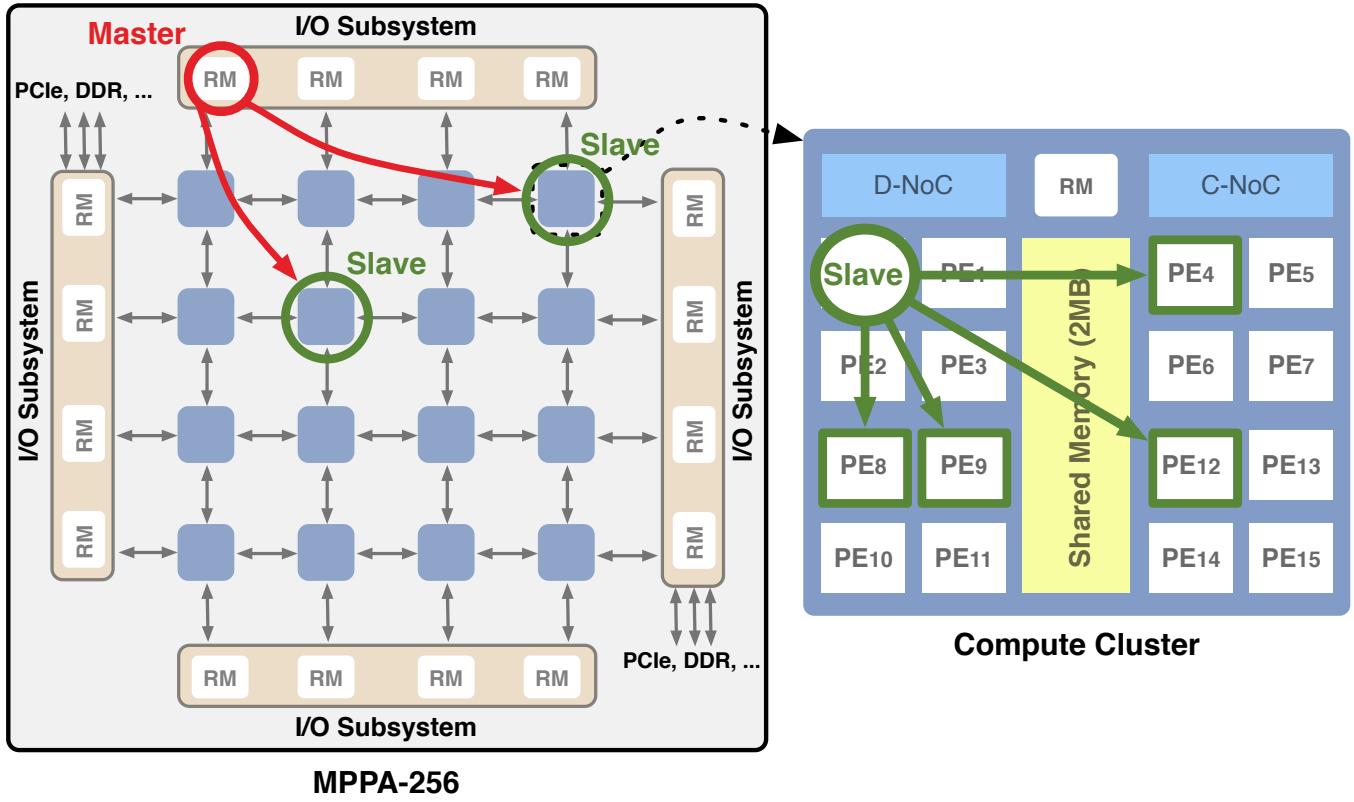
Kalray MPPA-256 overview



The **master** process spawns **worker processes**

One worker process per cluster

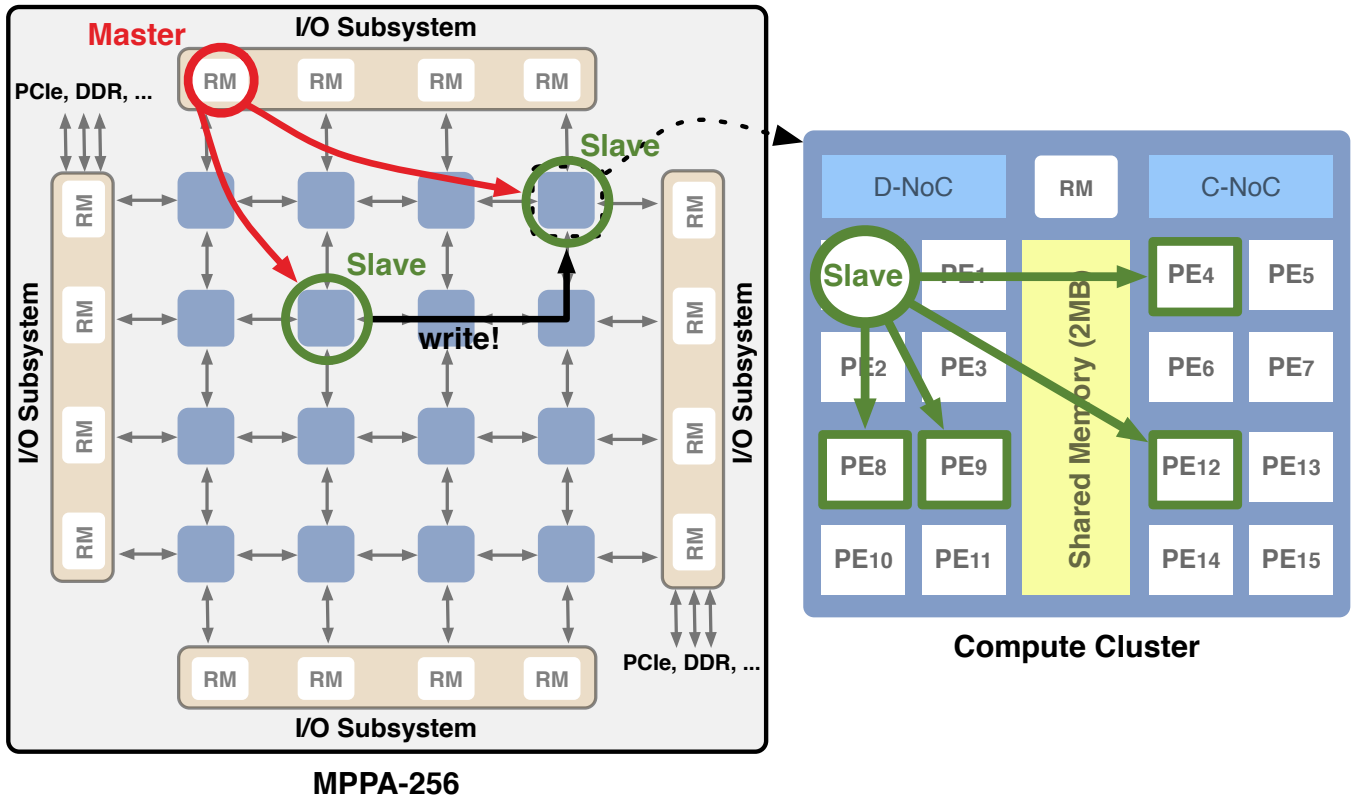
Kalray MPPA-256 overview



The **worker process** runs on the **PE0** and may create up to **15 threads**, one for each PE

Threads share 2 MB of memory

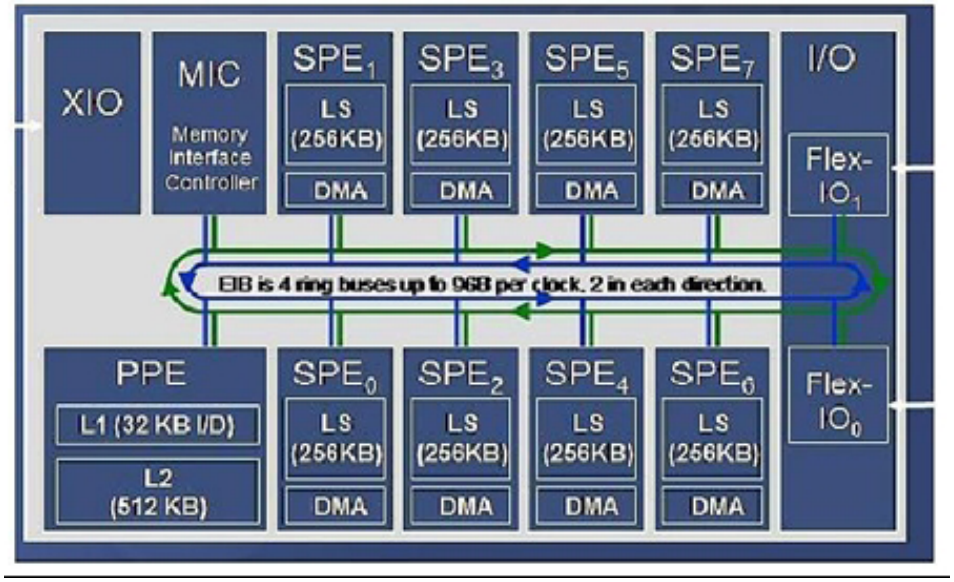
Kalray MPPA-256 overview



Communications take the form of **remote writes**

Data travel through the **NoC**

Specialized processor: CELL



- Developed by Sony, Toshiba and IBM: PlayStation 3 processor
- A processor is composed of a **main core** (PPE) and 8 **specific cores** (SPE)
- The PPE: classic PowerPC processor, without optimization, "in order", it affects the tasks to the SPEs
- SPEs: consisting of a local memory (LS) and a vector computation unit (SPU). Very fast access to their LS but to access the main memory they must perform an asynchronous transfer request to an interconnect bus. The SPEs perform the computational tasks.
- The optimization work is the **responsibility of the programmer**

CELL parallelism

- SPUs allow to process 4 32 bits operations / cycle (128 b register)
- Explicit programming of independent threads for each core
- Explicit memory sharing: the user must manage the data copy between cores

⇨ Harder to program than GPUs (because for GPUs, threads do not communicate between different multiprocessors, except at the beginning and at the end)

CELL processor: peak performance (128b registers, SP)

4 (SP SIMD) x 2 (FMA) x 8 SPUs x 3.2 GHz = 204.8 GFlops/socket (in SP)

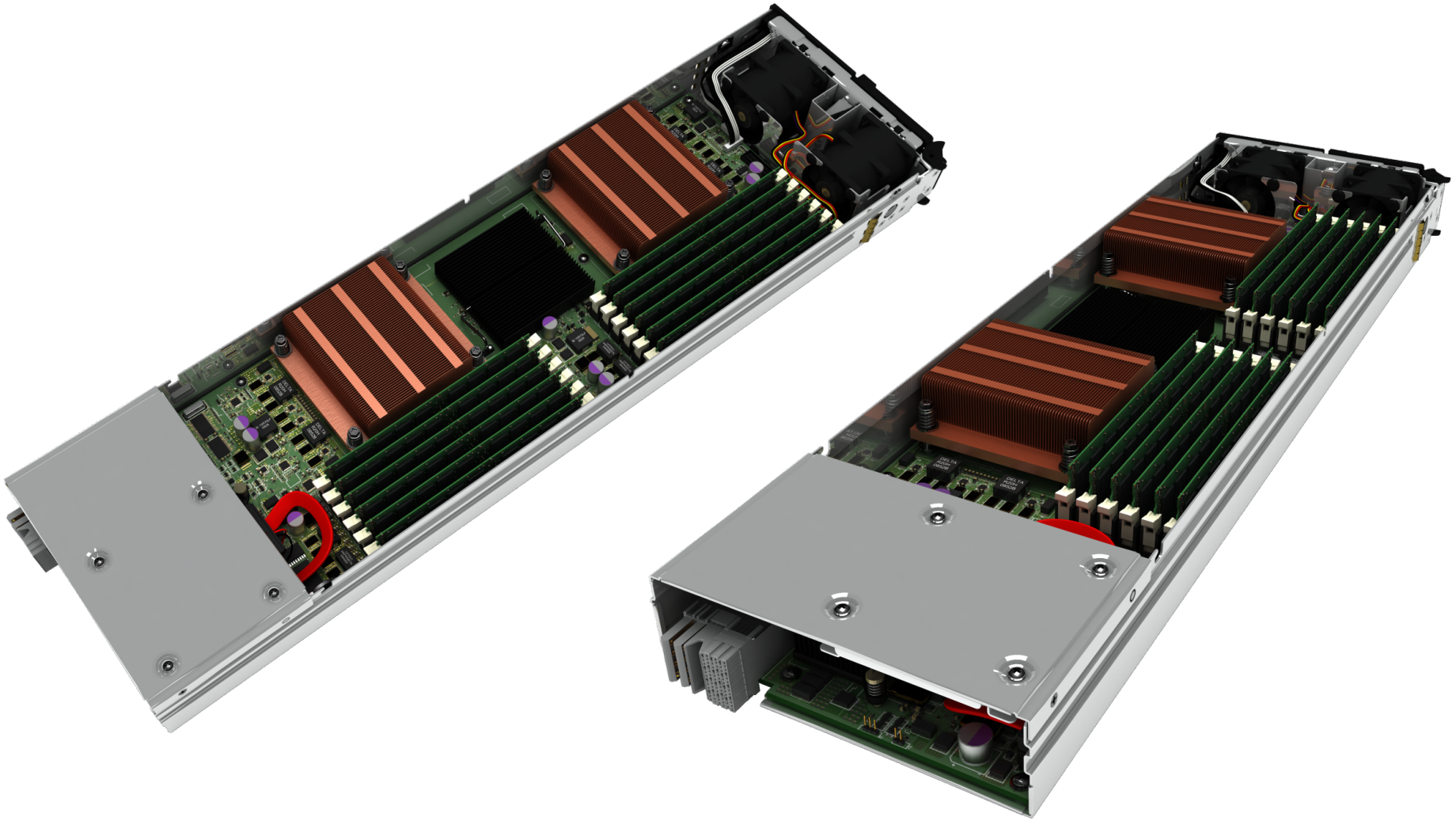
Specialized processors – hybrid programming

- **FPGA (Field Programmable Gate Array)**
  - ✓ adapted to specific problems
- **CELL**
  - ✓ interesting architecture but difficult to program
- **GPU**
  - ✓ More and more efficient
  - ✓ Better suited to HPC
  - ✓ Tools to program them being developed
  - ✓ Available anywhere, cheap

  • But adapted to a massive parallelism
  • PCI-e transfers greatly limit performance
  •The GPU as a co-processor (hybrid architecture) offers new perspectives, introduces new programming models

HOW TO BUILD A PETAFLOP MACHINE?

How to build a petaflop machine?



1 node, 2 sockets, 16 cores

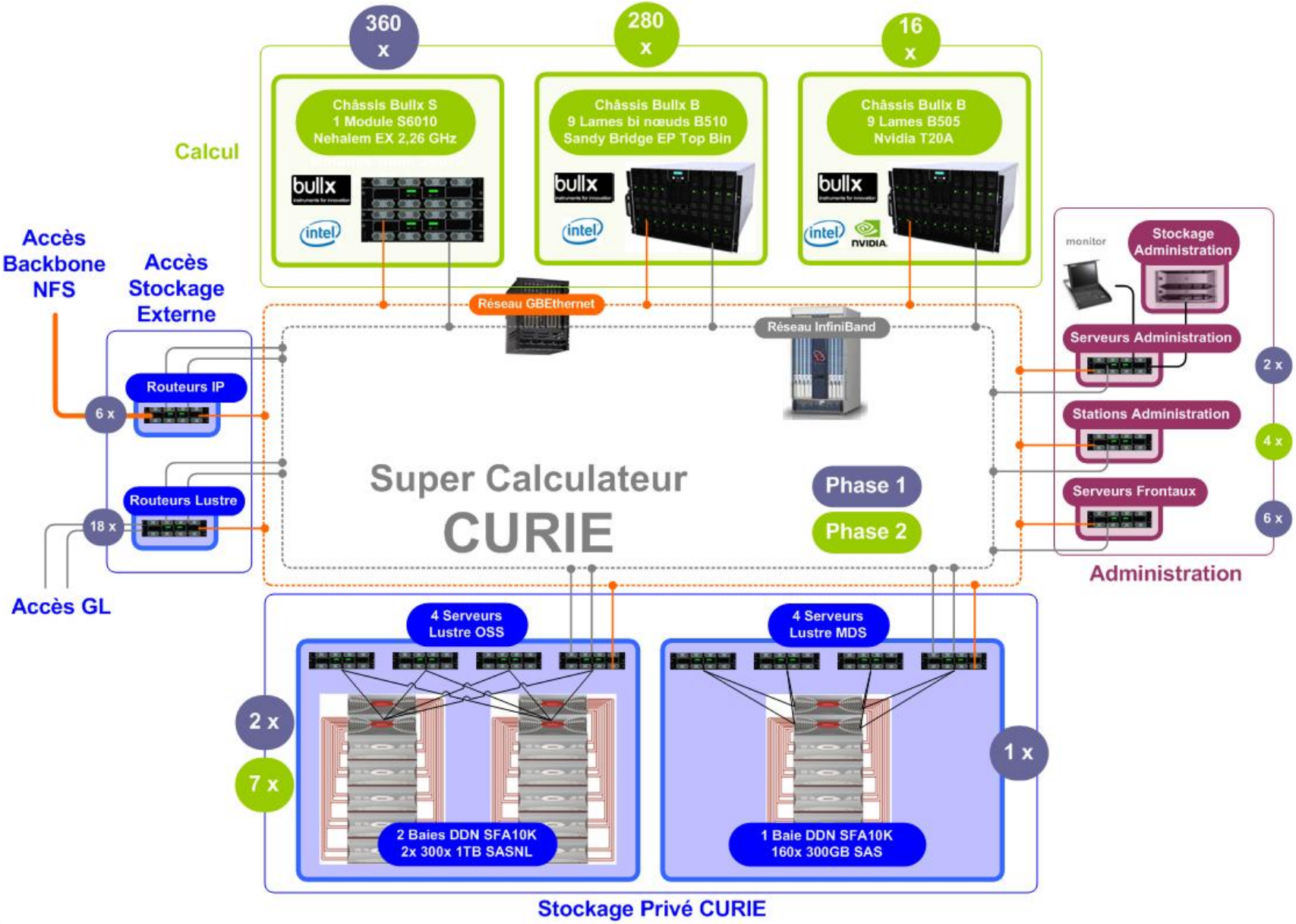
How to build a petaflop machine? Contd.

18 nodes, 36 sockets, 288 cores

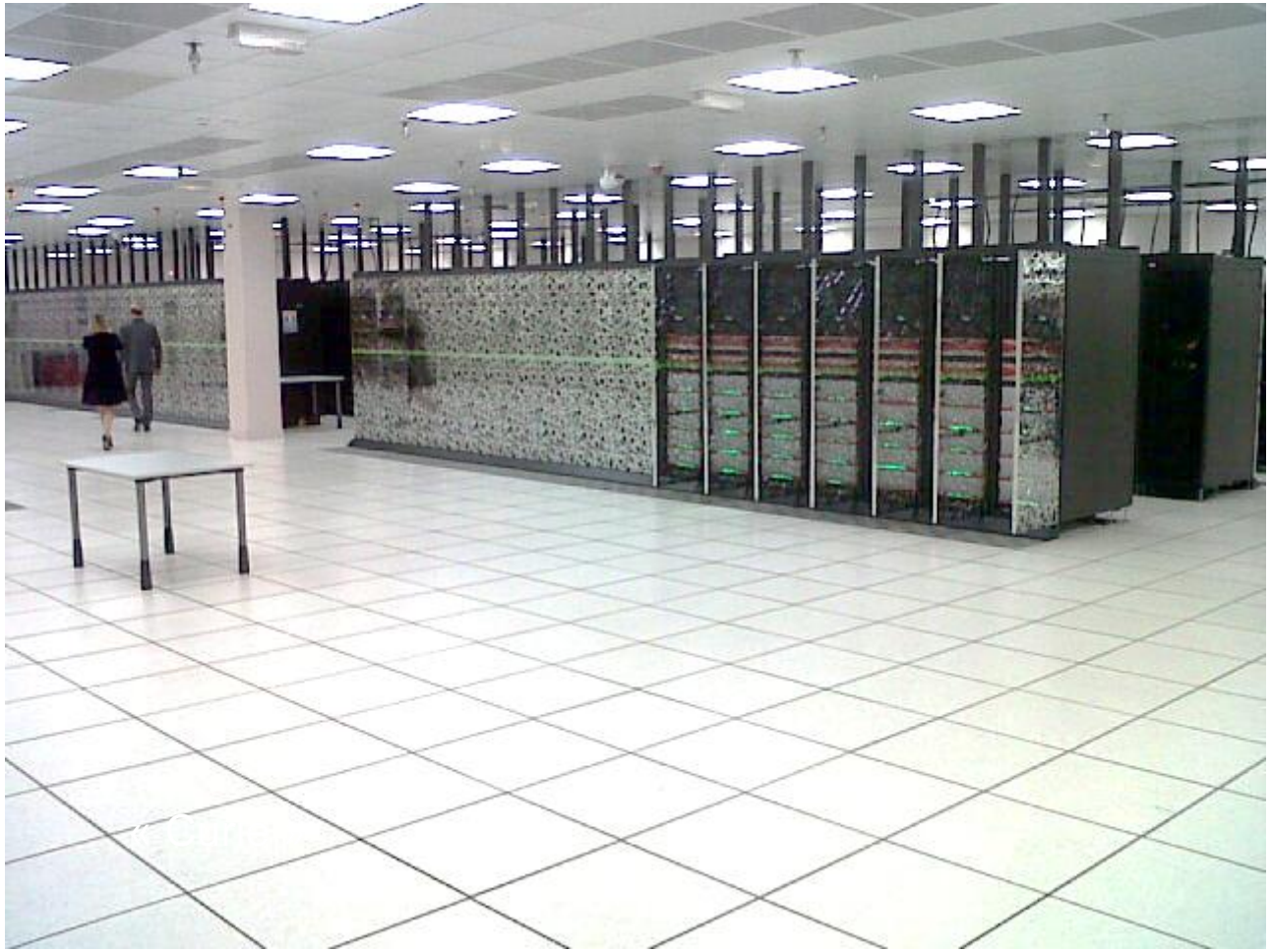
How to build a petaflop machine? Contd.

108 nodes, 216 sockets, 1728 cores

Connecting everything



How to build a petaflop machine? Contd.



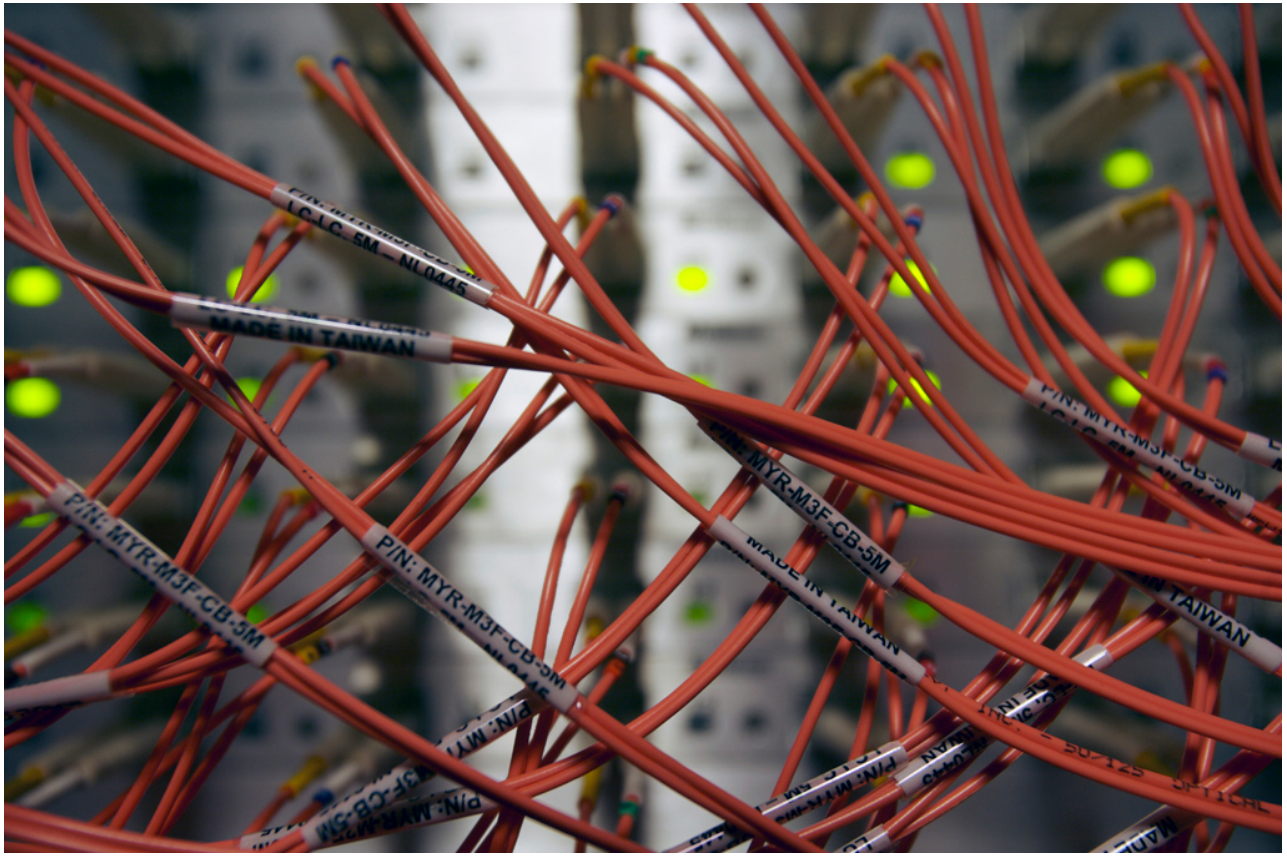
90 000 cores

360 To memory

10 Po storage

250 Go/s IO

200 m²



INTERCONNECTION NETWORKS

Formalism

- **Graph** G=(V,E)
  - V : switches and nodes
  - E: communication links e $\subseteq$ V ´ V
- ***Route***: (v_0, ..., v_k) path of length k between node 0 and node k,
  where (v_i,v_{i+1}) $\in$ E
- ***Routing distance***
- ***Diameter****:* maximum length between two nodes
- ***Average distance***
- ***Degree****:* number of input (output) channels of a node
- ***Bisection width:*** Minimum number of parallel connections that must be removed to have two equal parts

What characterizes a network?

Bandwidth (available bandwidth) $b = wf$

- Where w is the width (in bytes) and f is the send frequency f = 1 / t (in Hz)

Latency

- Time taken by a message to go from one node to another

Throughput (delivered bandwidth)

- How much bandwidth offered can be truly used

What characterizes a network? Contd.

Topology

- Physical network interconnection structure

Routing Algorithm

- Restricts all paths that messages can follow
- Many algorithms with different properties

Switching strategy

- How a message crosses a path
- Switching circuit vs. Packet switching

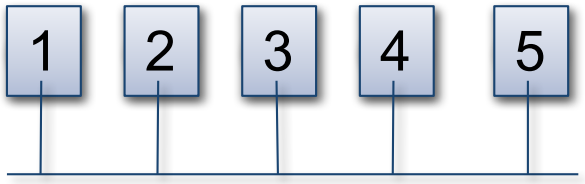
Flow control mechanism

- When a message (or piece of message) crosses a path, what happens when there is traffic?

Goals

- Latency must be as small as possible
- High throughput
- As many concurrent transfers as possible
  - The bisection width gives the potential number of parallel connections
- Lowest possible cost

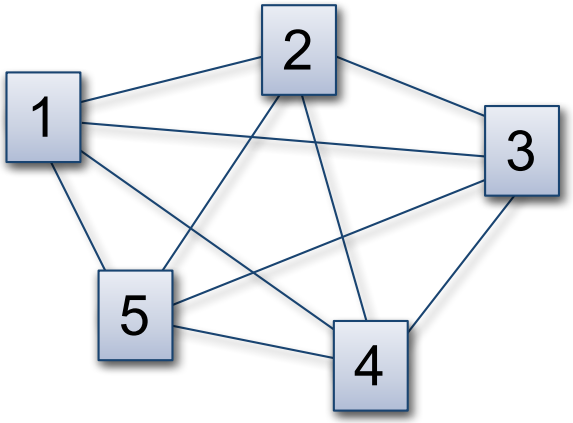
Bus (e.g. Ethernet)



- Degree = 1
- Diameter = 1
- No routing
- Bisection width = 1
  - CSMA/CD protocol
  - Limited bus length

Dynamic network
Simplest one
Lower cost

Complete network



Degree = n-1

too costly for large networks

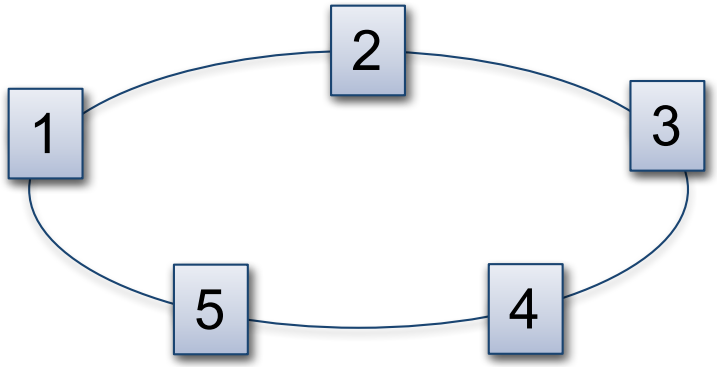
Diameter = 1

Bisection width = $\lfloor n/2 \rfloor \lceil n/2 \rceil$

When the network is cut in two parts, each node has a connection to n / 2 other nodes. There are n / 2 nodes like that.

Static network
Connection between every pair of nodes

Ring



Degree = 2

Diameter = $\lfloor n/2 \rfloor$

slow for big networks

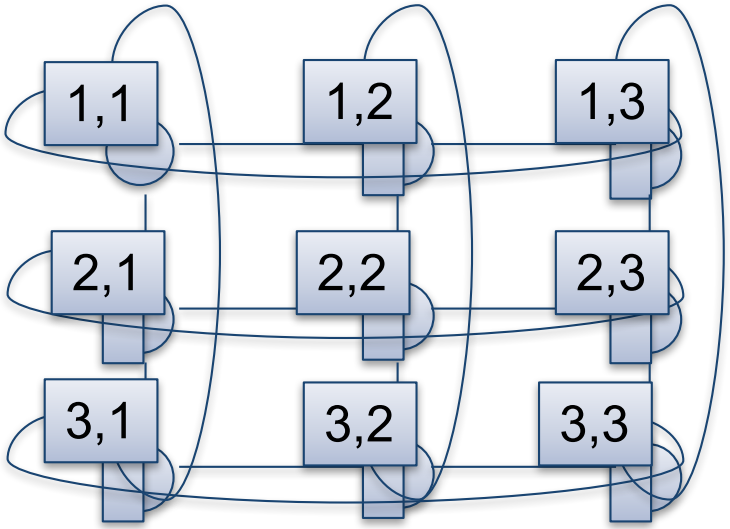
Bisection width = 2

Static network

A node i is connected to nodes
i+1 and i-1 modulo n.

– Examples: FDDI, SCI, FiberChannel Arbitrated Loop, KSR1

d-dimensional torus



For d dimensions

Degree = d

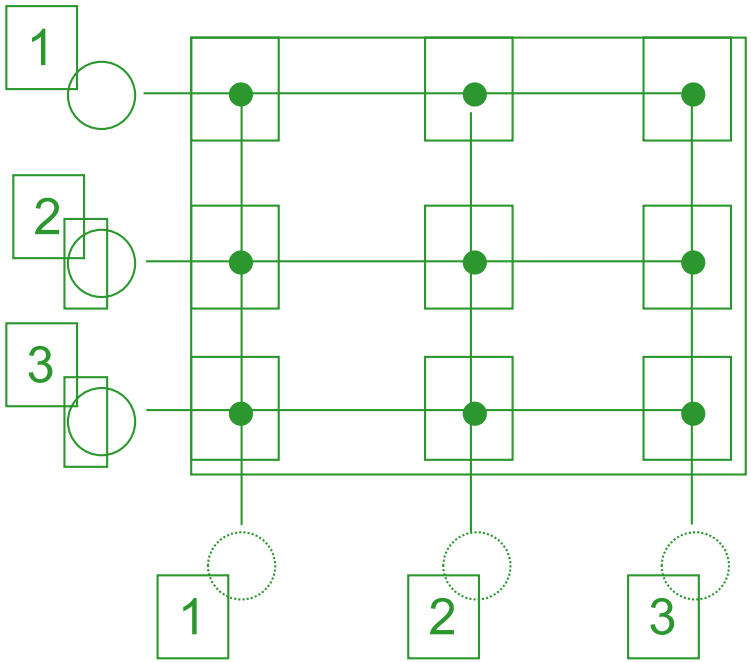
Diameter = d ($\sqrt[d]{n} - 1$)

Bisection width = ($\sqrt[d]{n}$) d–1

Static network

Cray T3D et T3E

Crossbar



Fast and costly (n^2 switches)

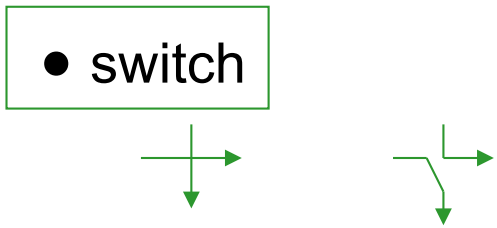
Processor x memory

Degree = 1

Diameter = 2

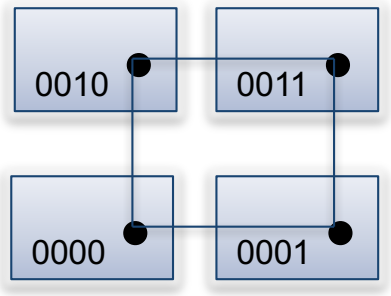
Bisection width = n/2

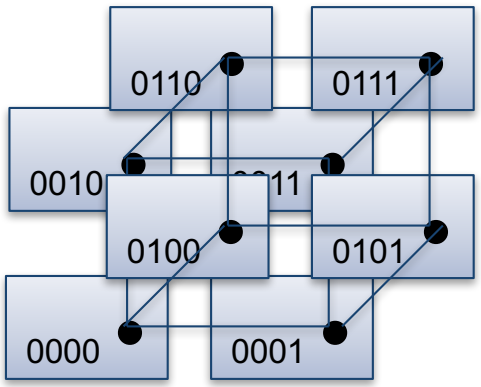
Ex: 4x4, 8x8, 16x16



Dynamic network

Hypercube



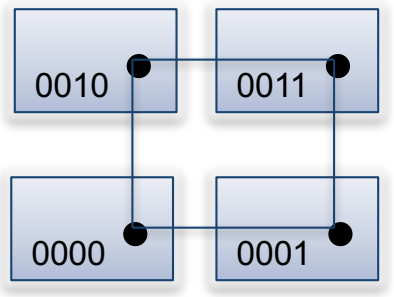


- **Hamming distance** =
  - Number of bits that differ in the representation of two numbers
  - Two nodes are connected if their Hamming distance is 1
  - Routing from x to y reduces the Hamming distance

Static network

Hypercube, contd

k dimensions, n= 2^k nodes

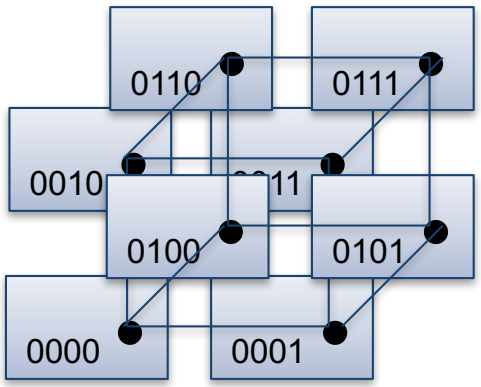


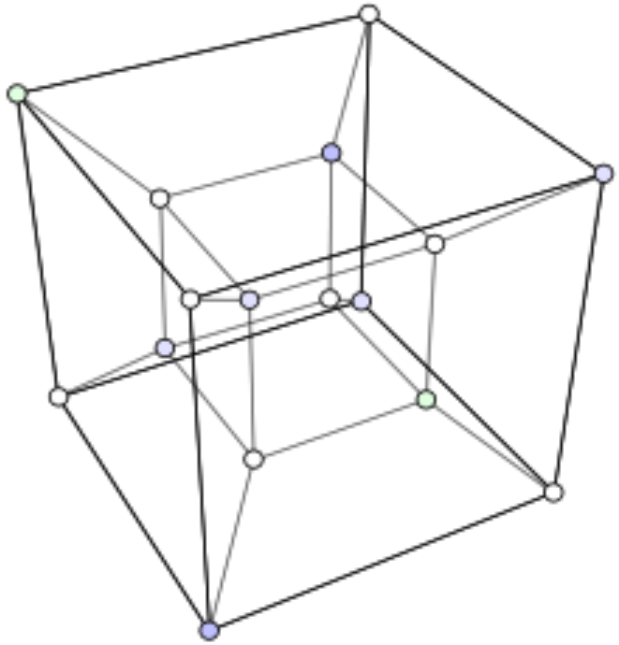
Degree = k

Diameter = k

Bisection width = n/2

Two (k-1)-hypercubes are connected through n/2 links to produce a k-hypercube

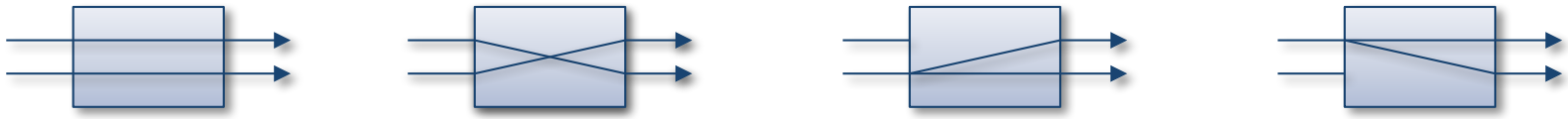




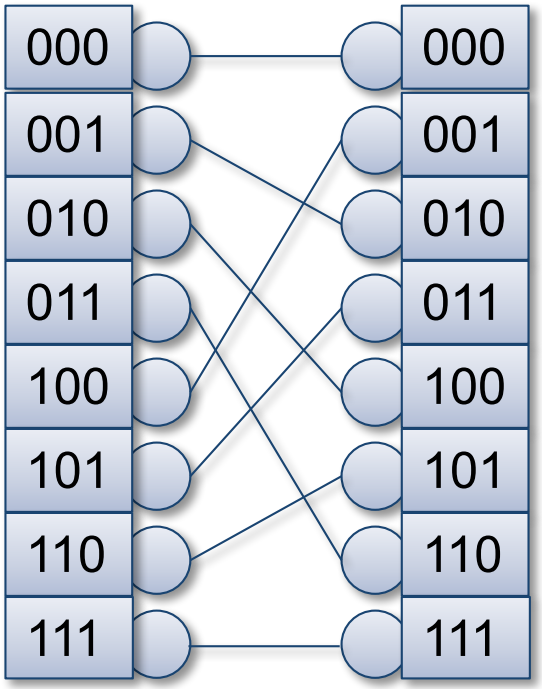
Intel iPSC/860,
SGI Origin 2000

Omega network

Basic block: 2x2 Shuffle



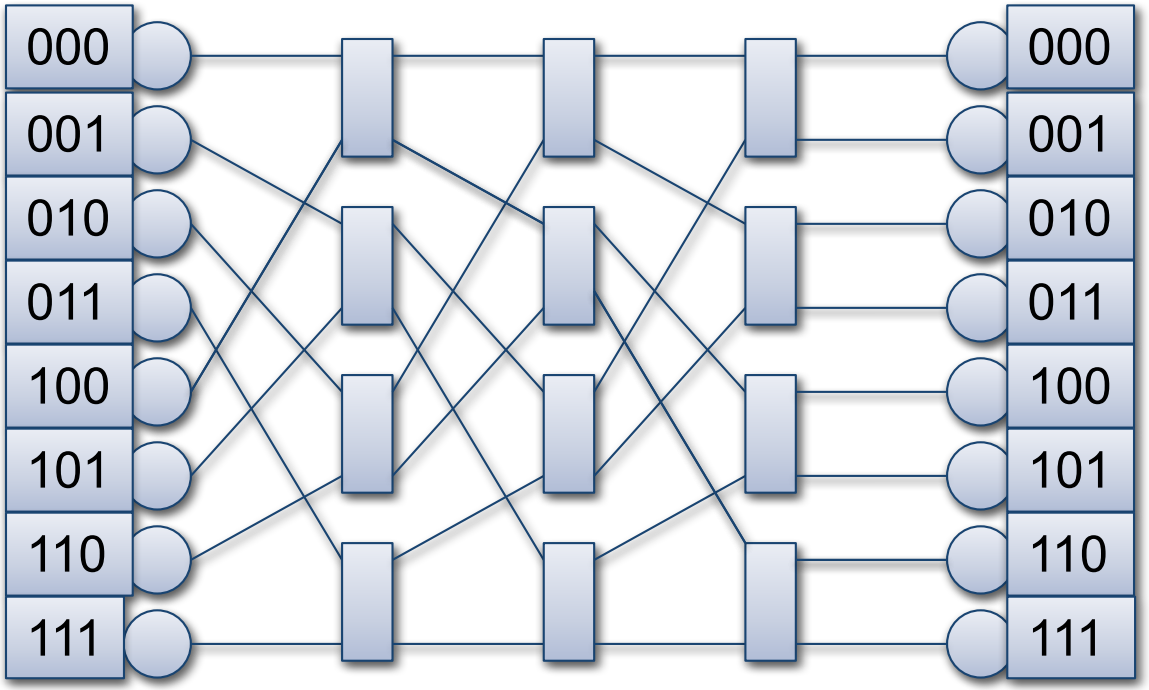
Perfect Shuffle



Omega network, contd.

Log_2n levels of 2x2 shuffle blocks

Dynamic network

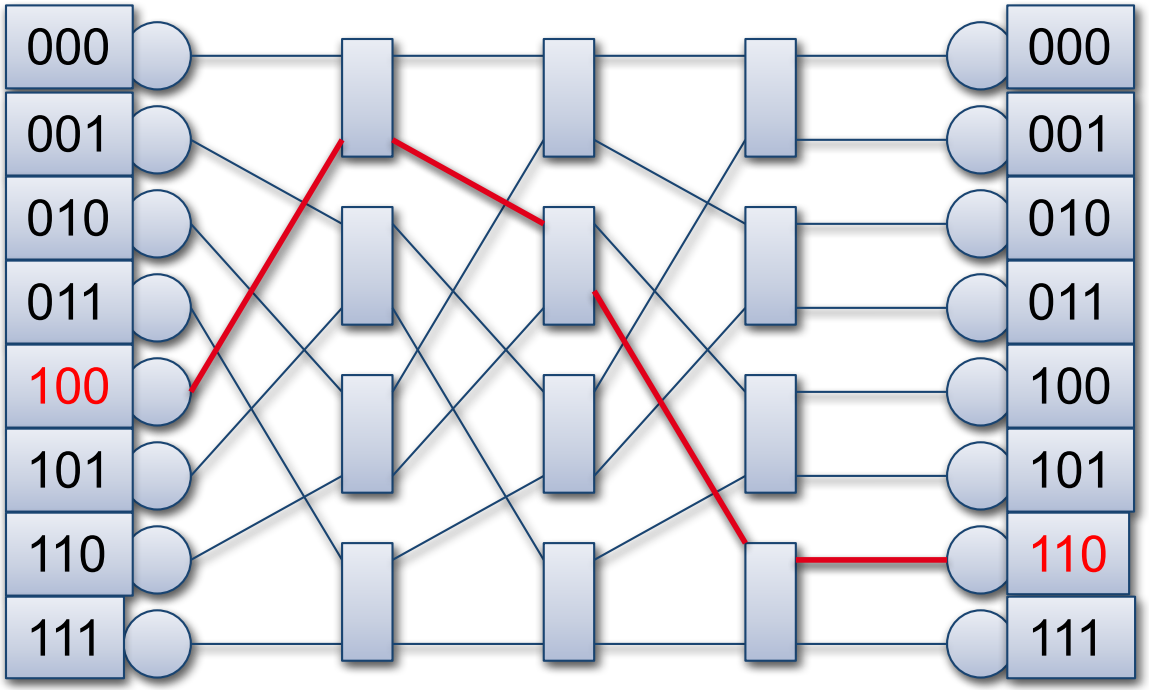


Level i looks for bit i
If 1 then go up
If 0 then go down

Omega network, contd.

Log_2n levels of 2x2 shuffle blocks

Dynamic network



Level i looks for bit i
If 1 then go up
If 0 then go down

Example 100 sends to 110

Omega network, contd.

n nodes

(n/2) $\log_2 n$ blocks

Degree = 2 for the nodes, 4 for the blocks

Diameter = $\log_2 n$

Bisection width = n/2

- For a random permutation, n / 2 messages are supposed to cross the network in parallel
- Extreme cases
  - If all the nodes want to go to 0, a single message in parallel
  - If each node sends a message, n parallel messages