High Performance Computing

Leopold Grinberg

T. J. Watson IBM Research Center, USA

High Performance Computing

Why do we need HPC?

High Performance Computing

Amazon can ship products within hours… would it be helpful if the process of processing customers orders will take 2-3 days ?

Would it be useful to predict yesterdays weather accurately ?

Oil/gas industry – it costs about $100M to drill …

but where ?

High Performance Computing

Why do we need high performance?

“ … **to compete** we need to be at least three month ahead in putting products on supermarket shelves … “ [Massimo N., Unilever]

How HPC fits into Scientific Computing

Physical process: self assembly of polymers

Mathematical model

Numerical simulation

HPC

Data analysis

HPA

Fast IO,
Fast, high-res visualization,
Fast extraction of O(10) useful values out of (O) 10^{100}

In this course:

Concepts in parallel computing

Roof-line model

How to chose right computer for an algorithm

How to chose right algorithm for a computer

Shared memory model

Distributed memory model

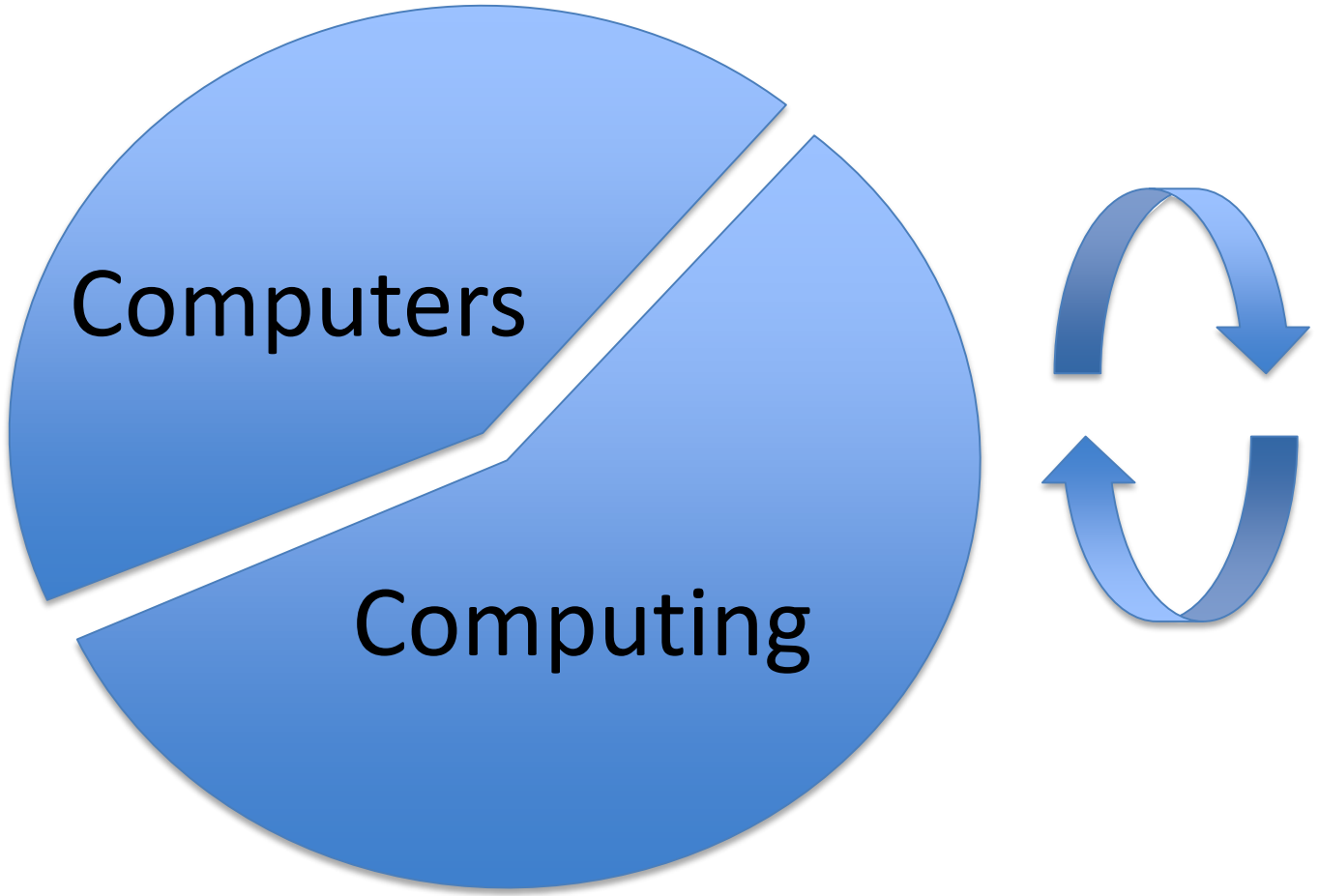
Hybrid computing

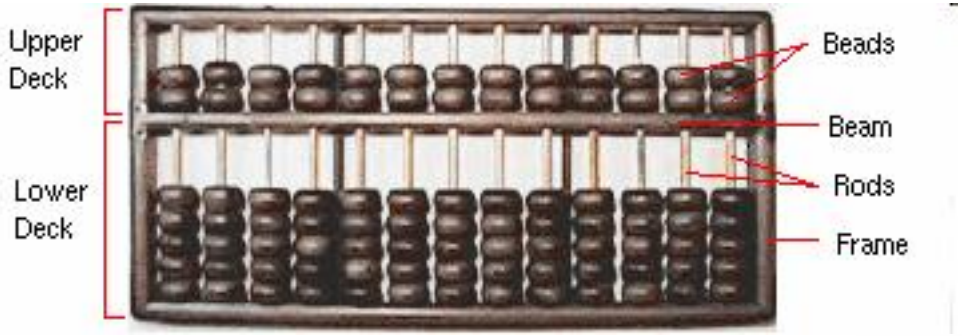
High Performance Computer

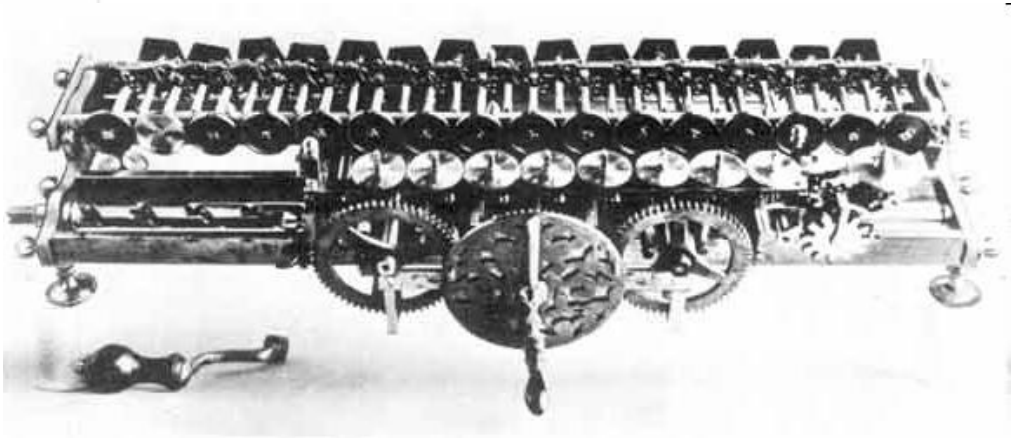
- Computer that can perform $O(10^{15})$ operations per second
- Computer that can move $O(10^{15})$ bytes per second
- Computer that consumes more energy than produced by a (nuclear) power station.
- Computer with operational cost of ¼ million dollar per hour

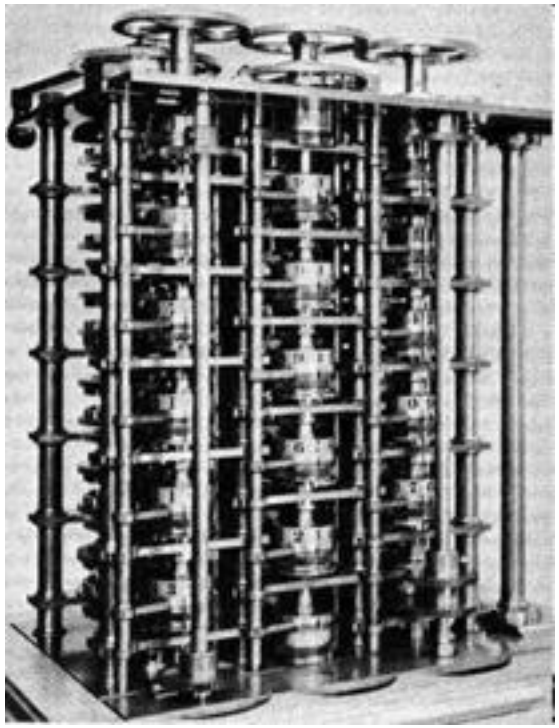
High Performance Computing

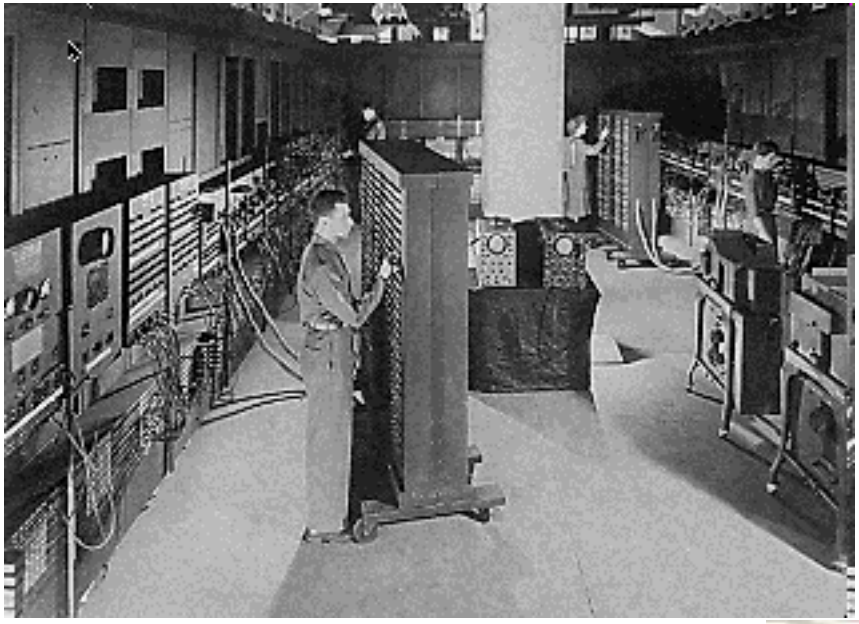
- Use of fast processors
- Use of many processors
- Smart use of many fast processors (scalable parallel algorithms)
- Use of software and hardware in a way that maximizes the output/input ratio


Computers
Computing


Upper Deck
Lower Deck
Beads
Beam
Rods
Frame

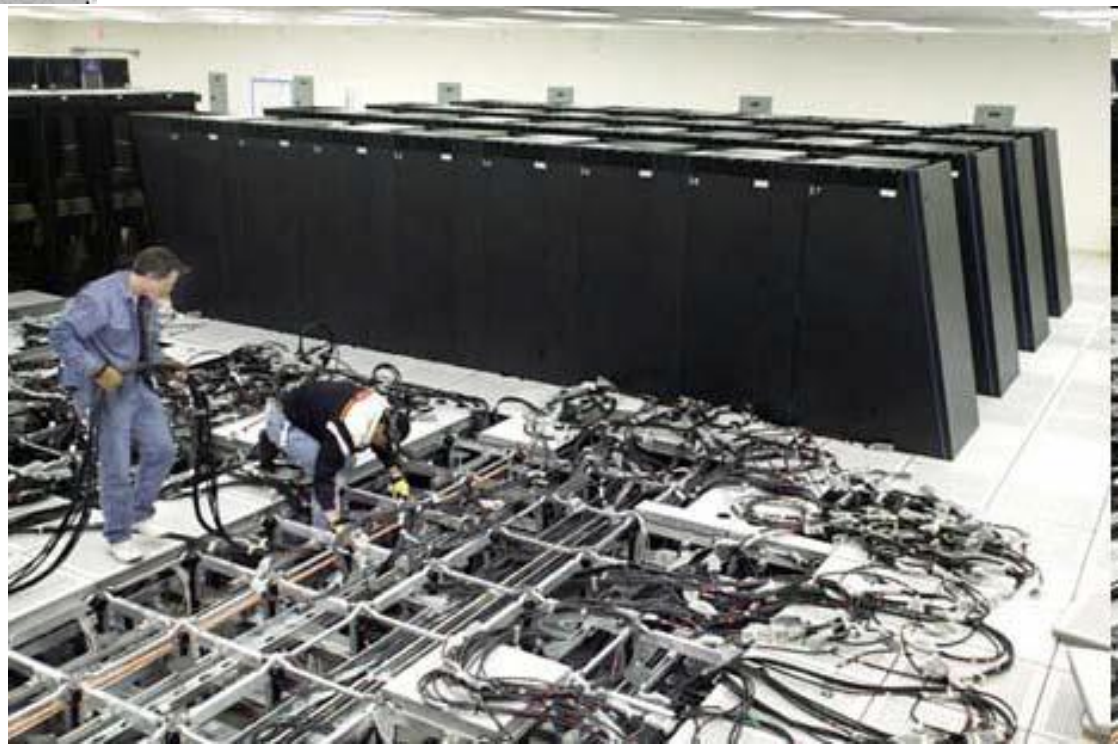






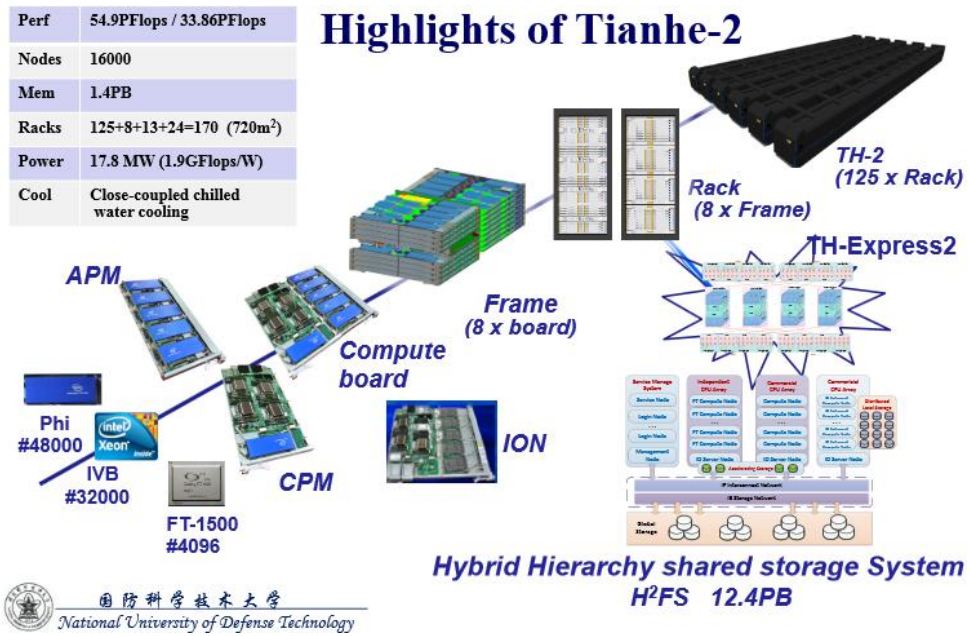
In use from 1946 to 1955, the ENIAC is commonly accepted as the first successful high-speed electronic digital computer (EDC).

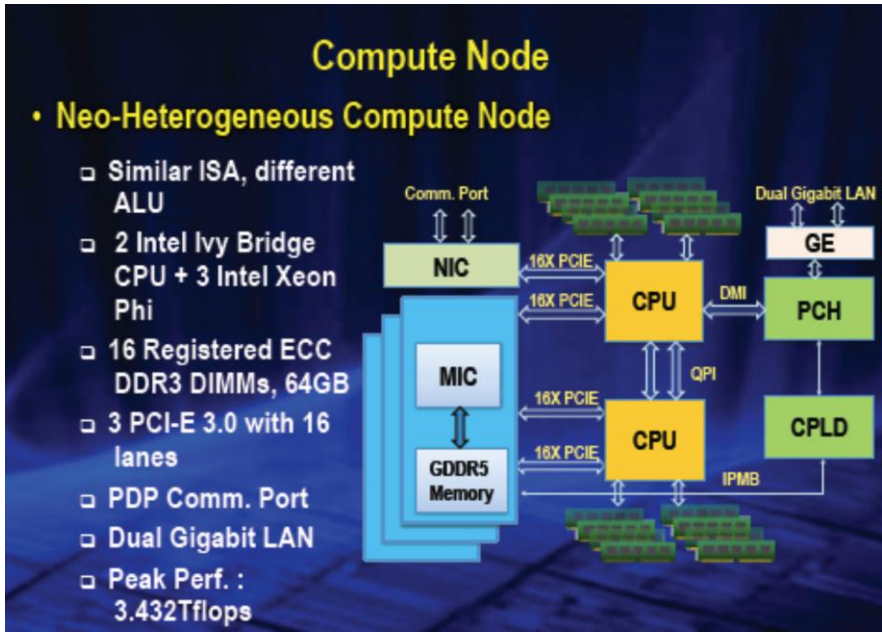
Installation of IBM BlueGene



Tianhe-2

Perf	54.9PFlops / 33.86PFlops
Nodes	16000
Mem	1.4PB
Racks	125+8+13+24=170 (720m^2)
Power	17.8 MW (1.9GFlops/W)
Cool	Close-coupled chilled water cooling





Next-generation supercomputers
100-300 PFLOPS (in 2017)

Summit is the next leap (in USA) in leadership-class computing systems for open science

ATTRIBUTE	TITAN	SUMMIT
Compute Nodes	18,688	~3,400
Processor	(1) 16-core AMD Opteron per node	(Multiple) IBM POWER 9s per node
Accelerator	(1) NVIDIA Kepler K20x per node	(Multiple) NVIDIA Volta GPUs per node
Memory per node	32GB (DDR3)	>512GB (HBM+DDR4)
CPU-GPU Interconnect	PCI Gen2	NVLINK (5-12x PCIe3)
System Interconnect	Gemini	Dual Rail EDR-IB (23 GB/s)
Peak Power Consumption	9 MW	10 MW

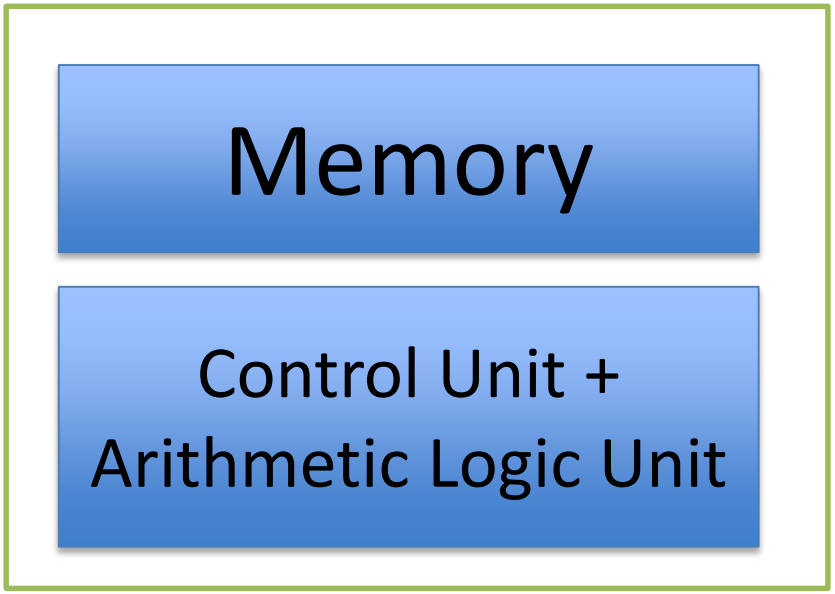
How to drive at top speed?

http://www.baixaki.com.br/imagens/galeria/1829/14970.jpg

How to drive at top speed?

To start with – know what do you drive…

High Performance Computer – what is inside?



Disks, cables, more cables, fans, water …

von Neumann Architecture

Named after the Hungarian mathematician John von Neumann who first authored the general requirements for an electronic computer in his 1945 papers.

Also known as "stored-program computer" - both program instructions and data are kept in electronic memory. Differs from earlier computers which were programmed through "hard wiring"

Four main components: Memory, Control Units, Arithmetic Logic Units, Input-Output

Blue Gene/Q packaging hierarchy

1. Chip
16 cores

2. Module
Single Chip

3. Compute Card
One single chip module,
16 GB DDR3 Memory

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

5a. Midplane
16 Node Cards

5b. I/O Drawer
8 I/O Cards
8 PCIe Gen2 slots

6. Rack
2 Midplanes
1, 2 or 4 I/O Drawers

7. System
20PF/s

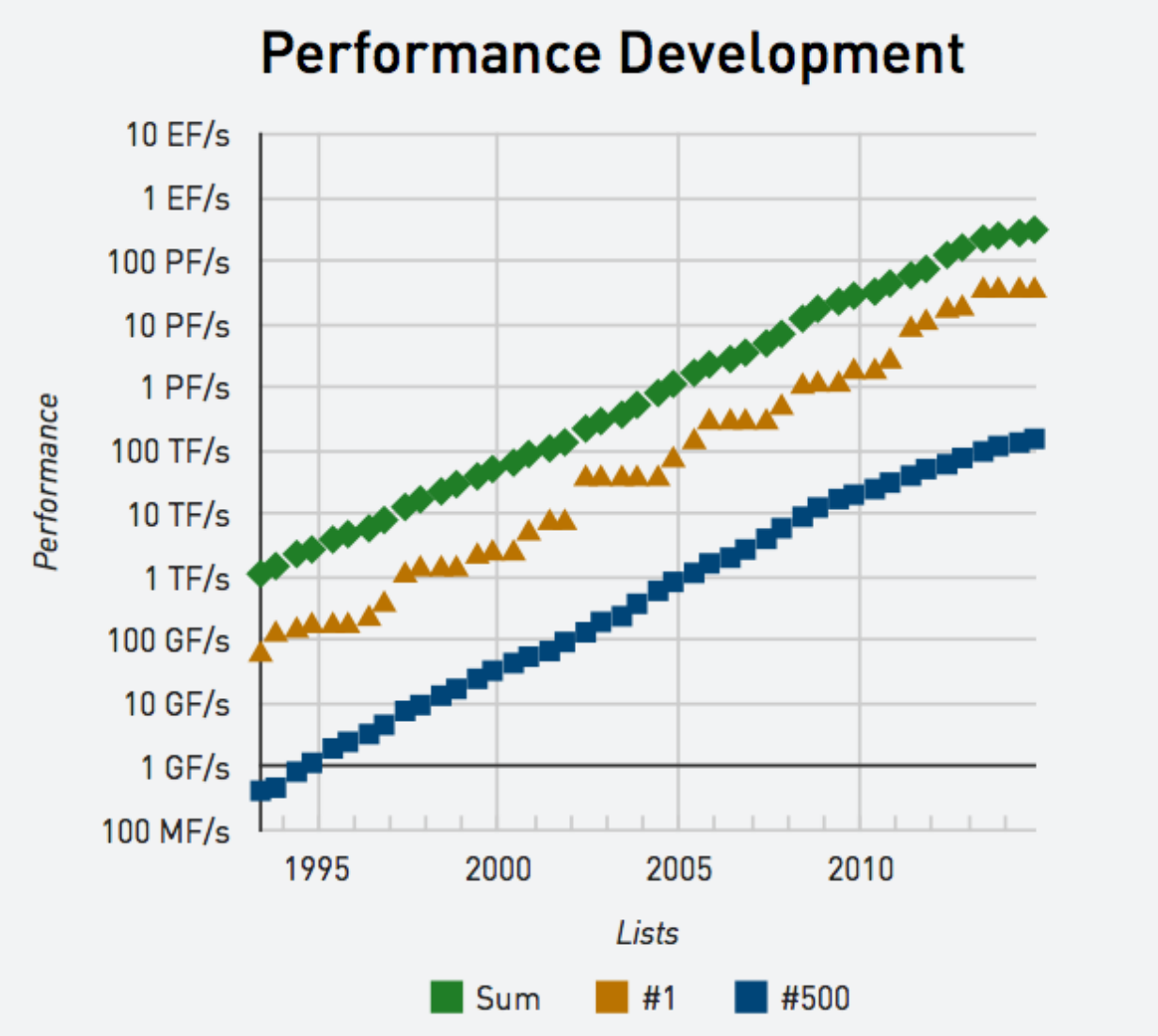
What $100M machine does?

Not much …

Add	+
Subtract	−
Multiply	×
Divide	÷
Mod	%

Why does it cost so much?

Speed (and accuracy and reliability)



How the speed is achieved ?

High clock frequency

High memory bandwidth

Parallel processing
of data and/or instructions
and **fast communication**

Not all processors and memory subsystems are the same

Some have high clock frequency

Some have high memory bandwidth

Some can compute >1 instructions per cycle

and some can not

Some need a lot of energy , and some not

….

A compute node

Processor is the heart of a compute node

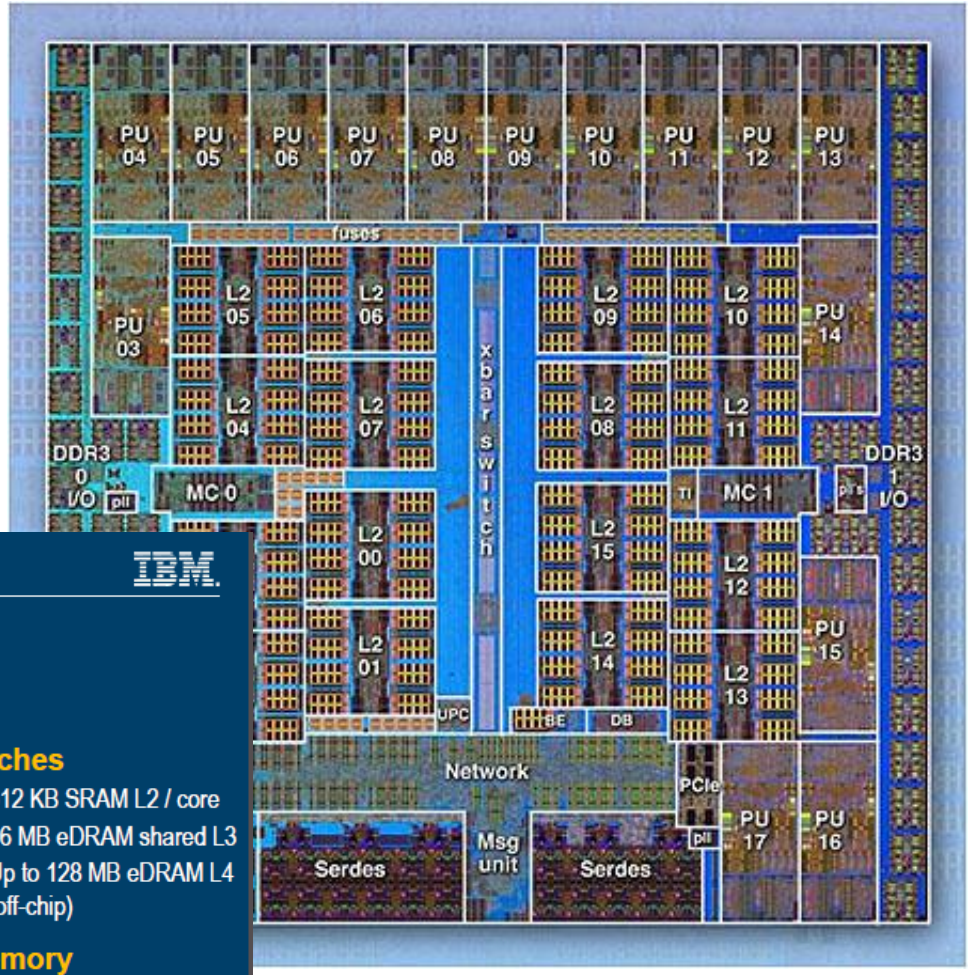
Memory subsystem is its vascular system

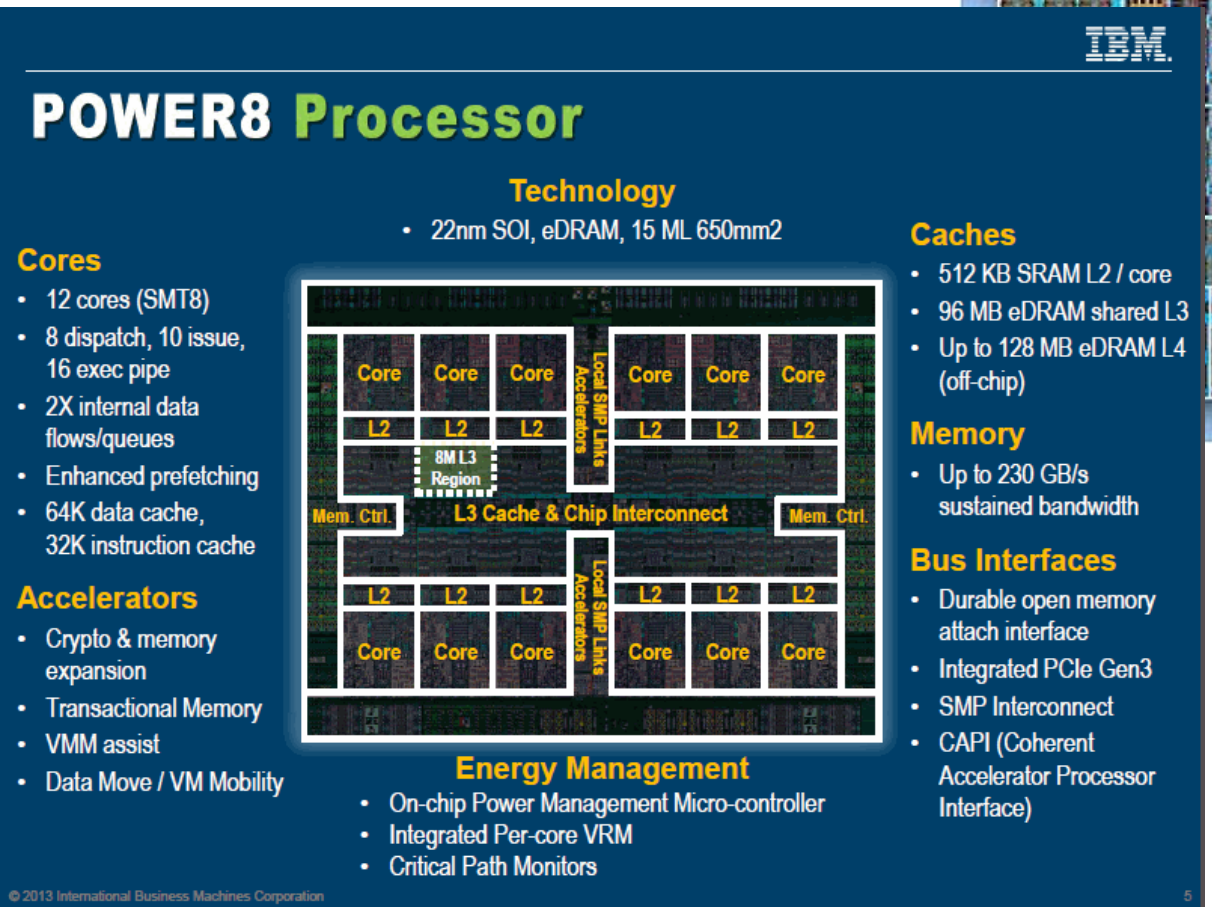
Understanding of computing starts with the understanding of memory subsystem

in order to compute $c=a+b$:

- a memory space for *a, b* and *c* should be reserved
- *a* and *b* must be initialized
- data stored in &*a* and &*b* must be moved to the processor's registers
- computed data must be stored in &*c*

On chip memory hierarchy

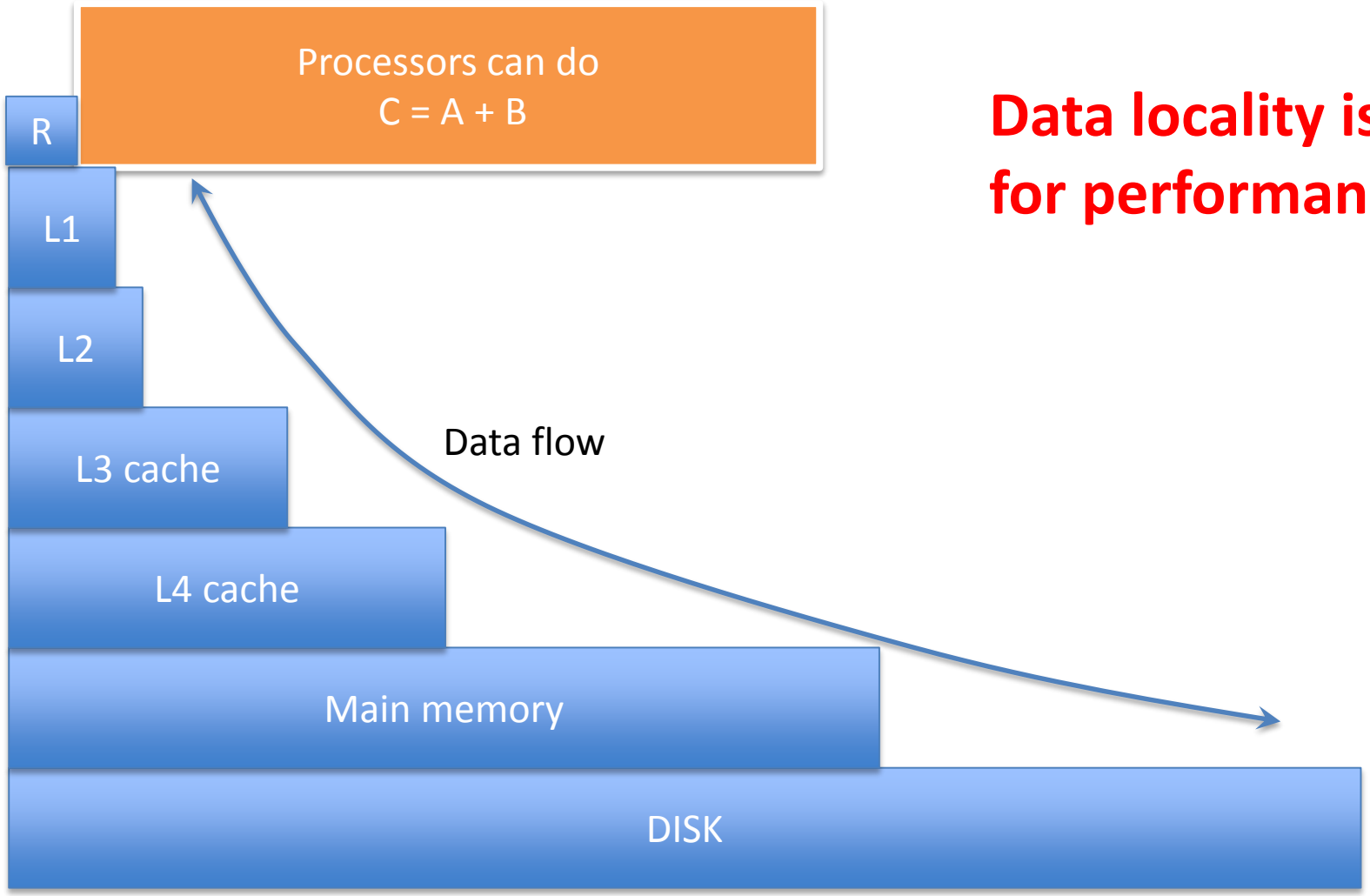




BlueGene/Q Chip (top)

Power 8 (left)

Why memory model is important?



Data locality is crucial for performance!

Bandwidth and Latency

An airplane can lift 100M hard-drives, each holding 1TB of data and fly from Beijing to Boston in 13 hours:

memory bandwidth: 100E6 *1TB/(13h*3600s/h) = 2,136.7 TB/s

latency : 13 hours ….

alternatively we can e-mail data in 2MB packets, it will take 60 seconds for each e-mail to be received:

memory bandwidth: 0.033MB/s

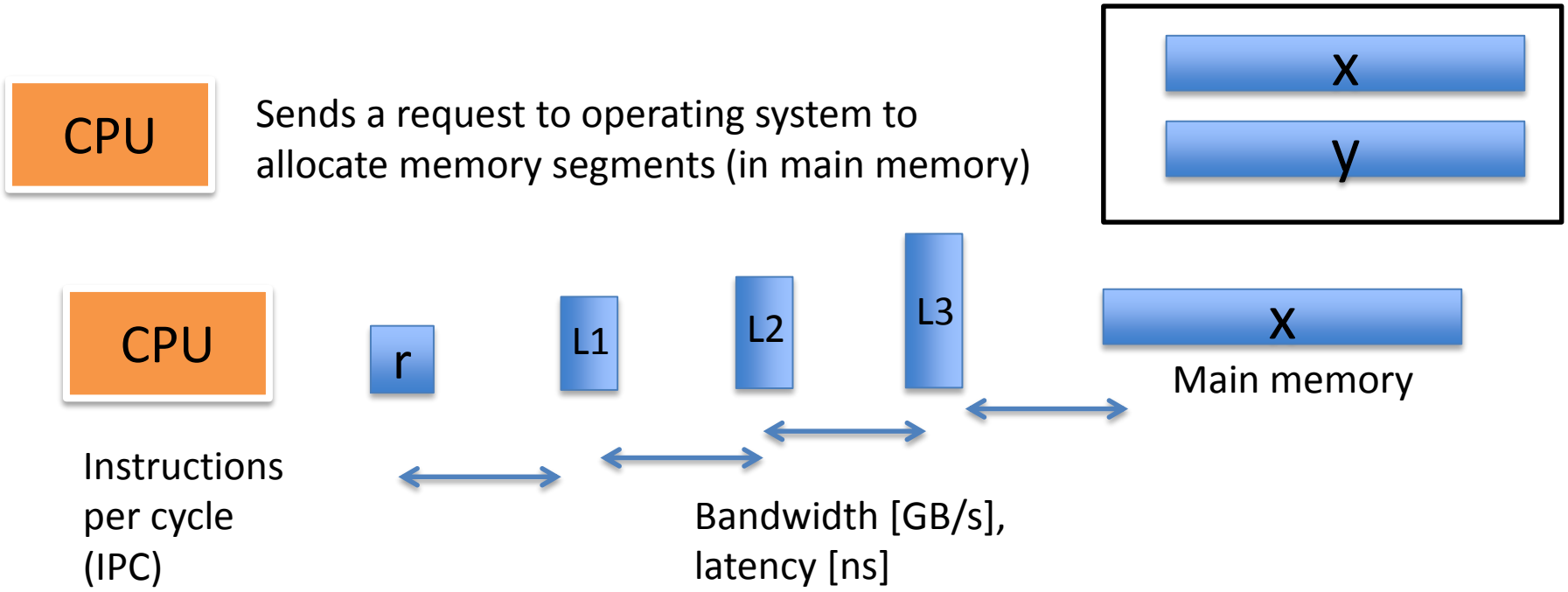
latency : 60 seconds

(882,639,730,639 hours to move 100E6*1TB)

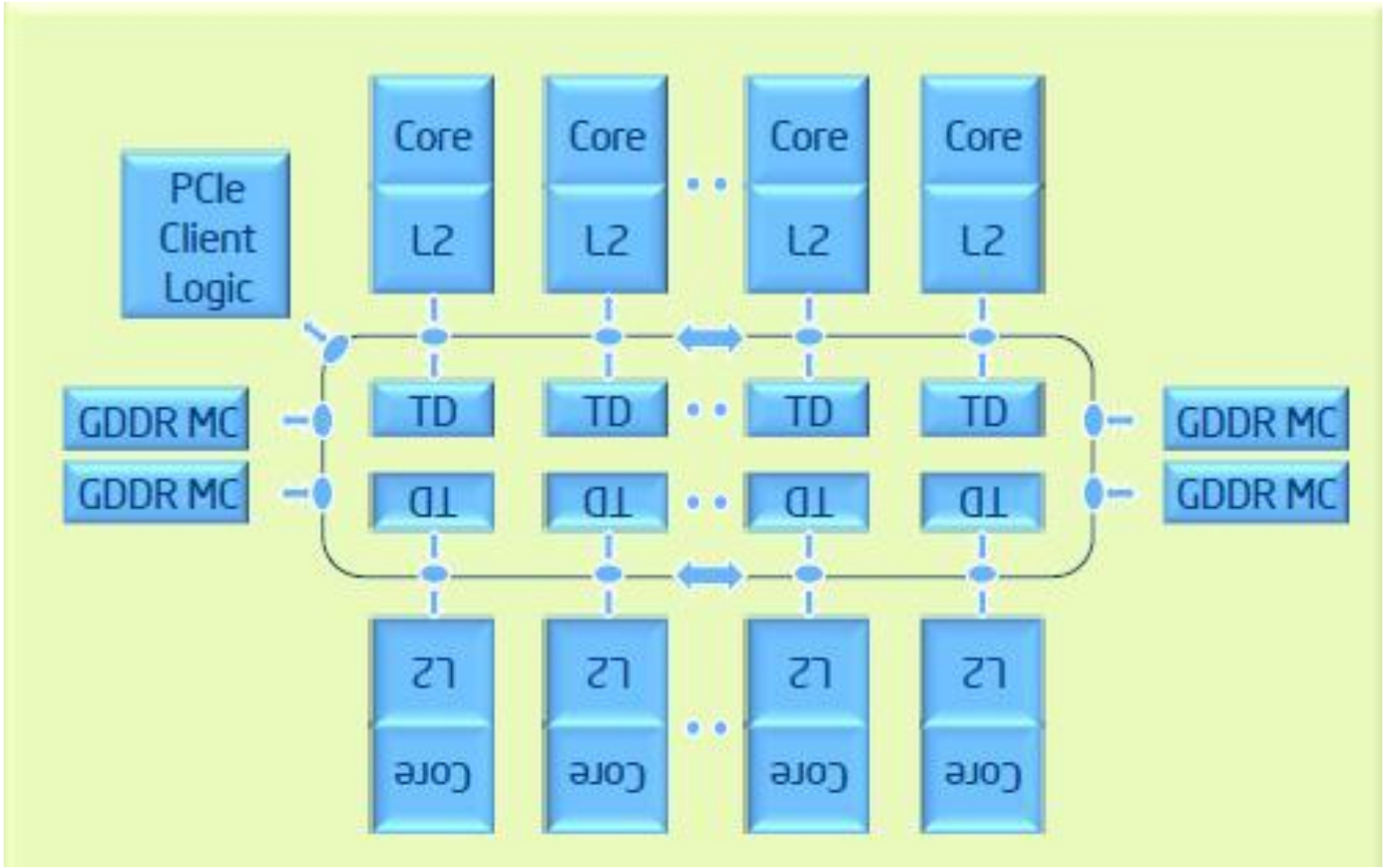
Inside the machine…

```
for (i=0; i < N; i=i+1){
    y[i] = a*x[i] + y[i];
}
```

1. Allocate memory for vectors x and y,
2. Assign values to a, y[i], x[i]
3. Perform multiply add operation.



Xeon Phi



Xeon Phi

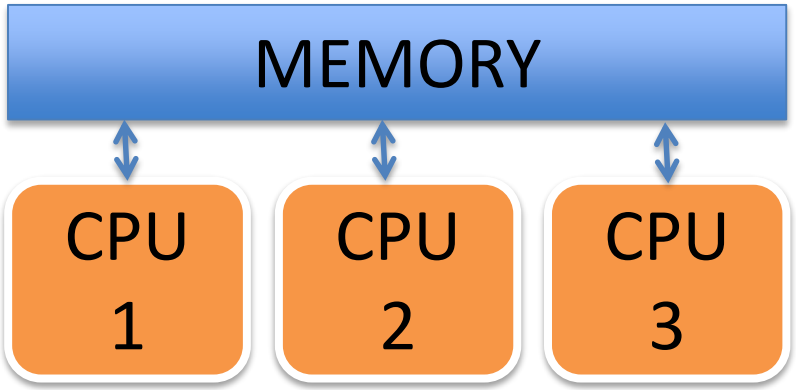
The Intel Xeon Phi coprocessor implements very high bandwidth memory subsystem. Each core is equipped with a 32KB L1 instruction cache and 32KB L1 data cache and a 512KB unified L2 cache. These caches are fully coherent and implement the x86 memory order model. **The L1 and L2 caches provide an aggregate bandwidth that is approximately 15 and 7 times, respectively, faster compared to the aggregate memory bandwidth. Hence, effective utilization of the caches is key to achieving peak performance on the Intel Xeon Phi coprocessor.** In addition to improving bandwidth, the caches are also more energy efficient for supplying data to the cores than memory.

https://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-codename-knights-corner

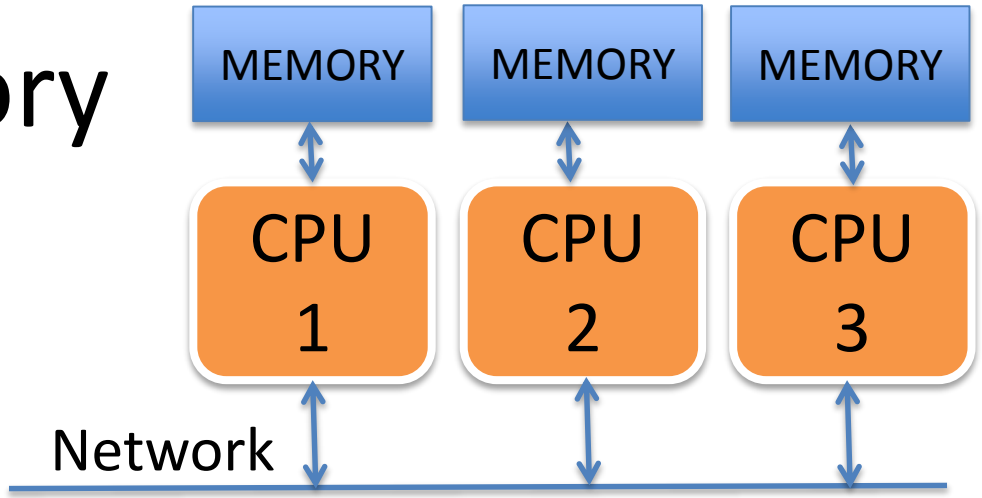
Some Theory
and
Introduction to Parallelism

Parallel Computing: **Memory Models**

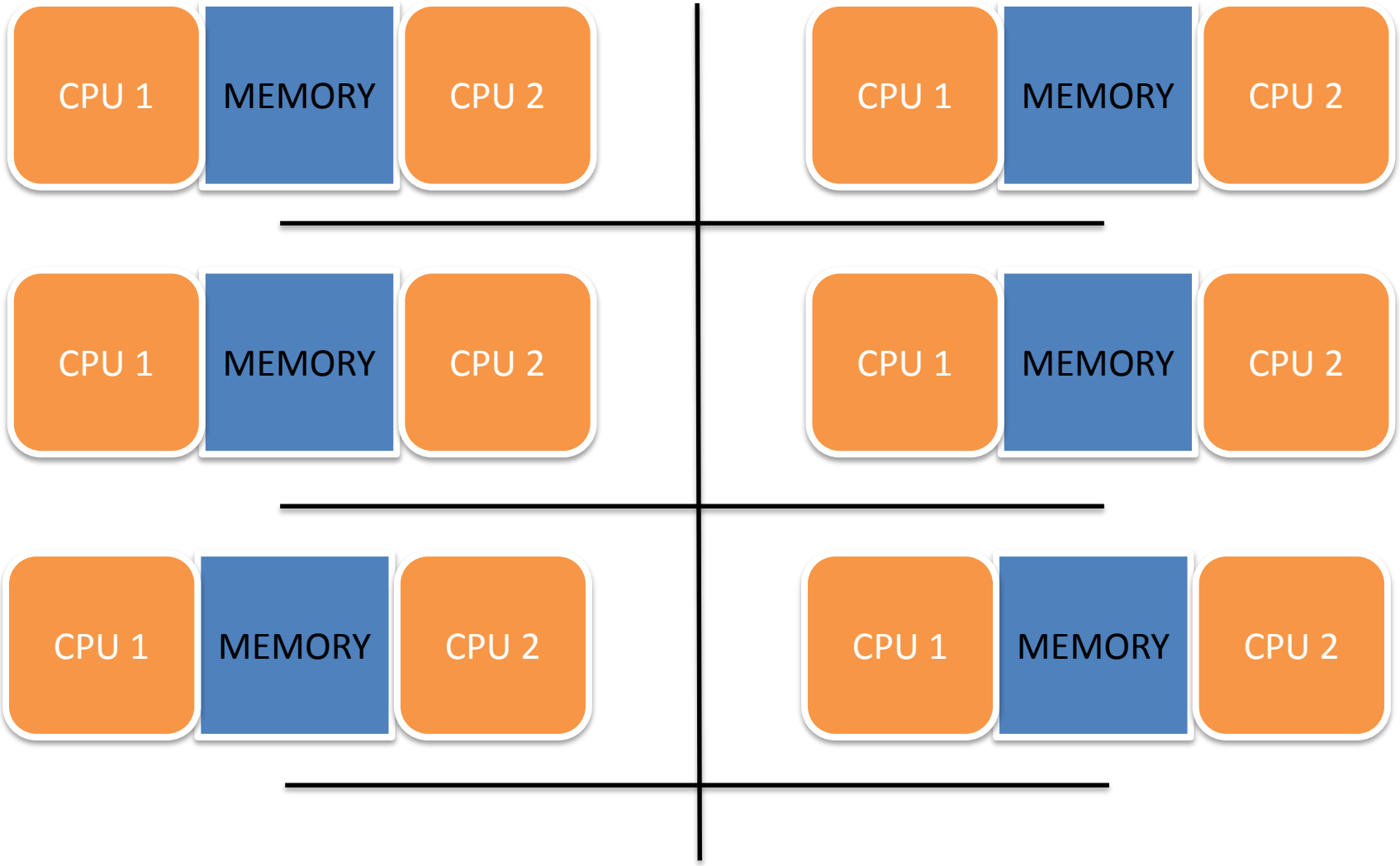
Shared Memory



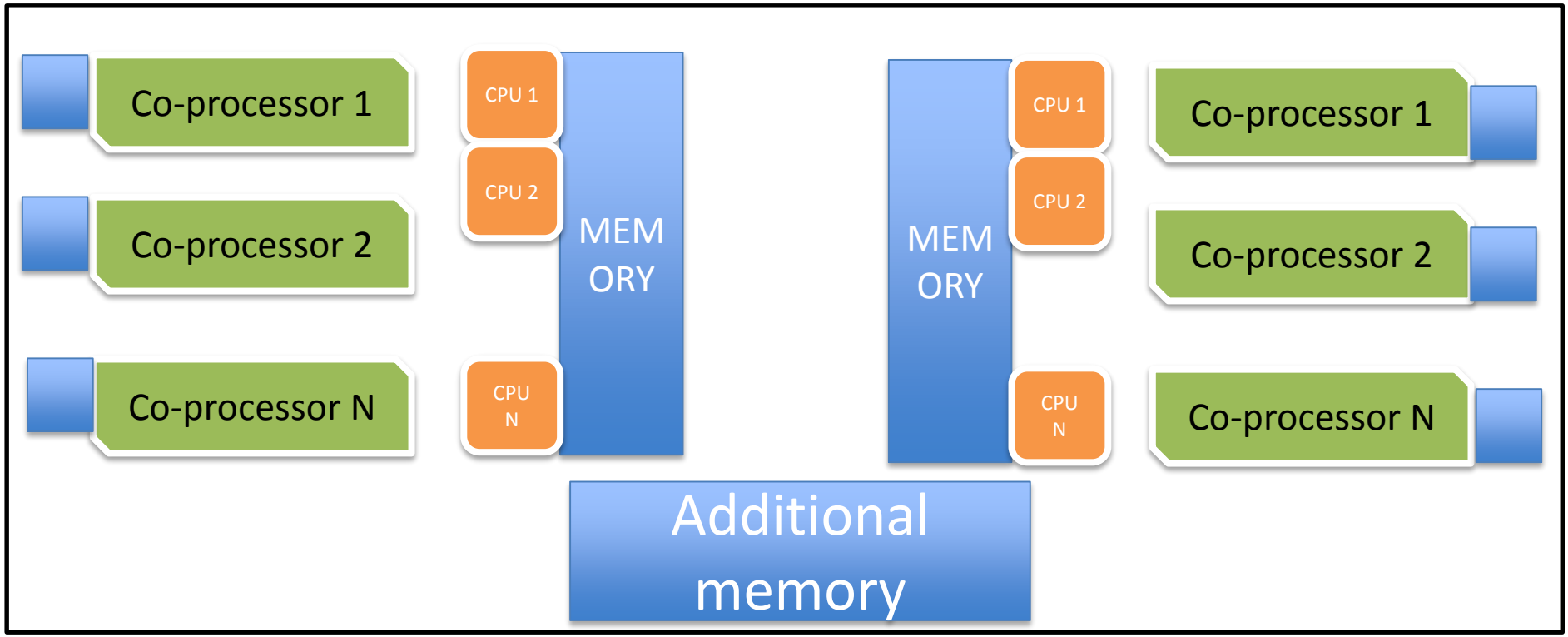
Distributed Memory



(Real) Parallel Computing: **Memory Models**



(Even more Real) Parallel Computing: **Memory Models**



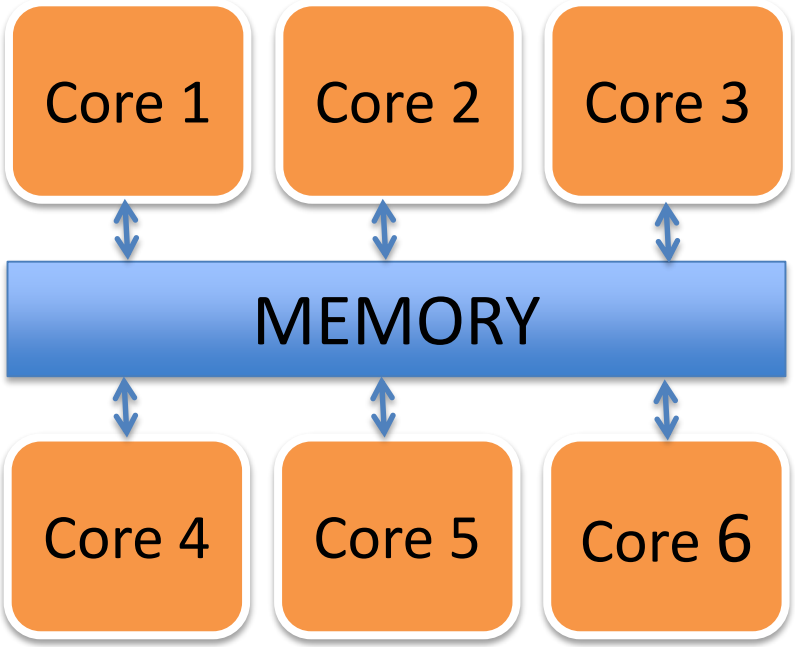
Programming Models

1. Shared Memory

2. Distributed Memory

3. Hybrid – many pools of shared and distributed memory

Shared Memory Programming Model

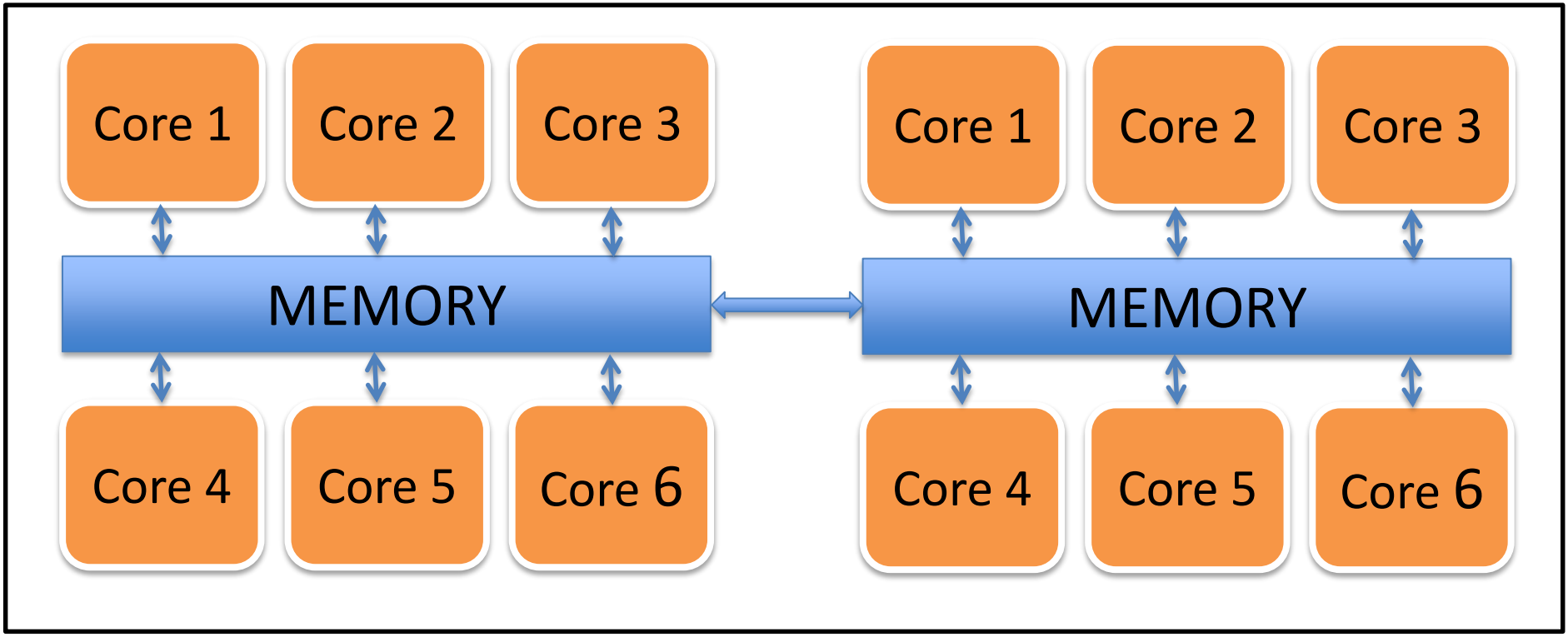


Shared memory programming model is typically implemented by using threads (OpenMP)

Example: 6 core CPU. Main memory is shared. There is a single address space. All cores can read and write from/to the same address.

Good: memory is accessible by all cores at the same speed. Many variables can be shared.
Bad: developers must be careful and prevent from different cores to write to the same address, or read and write to/from the same address at the same time (race conditions).

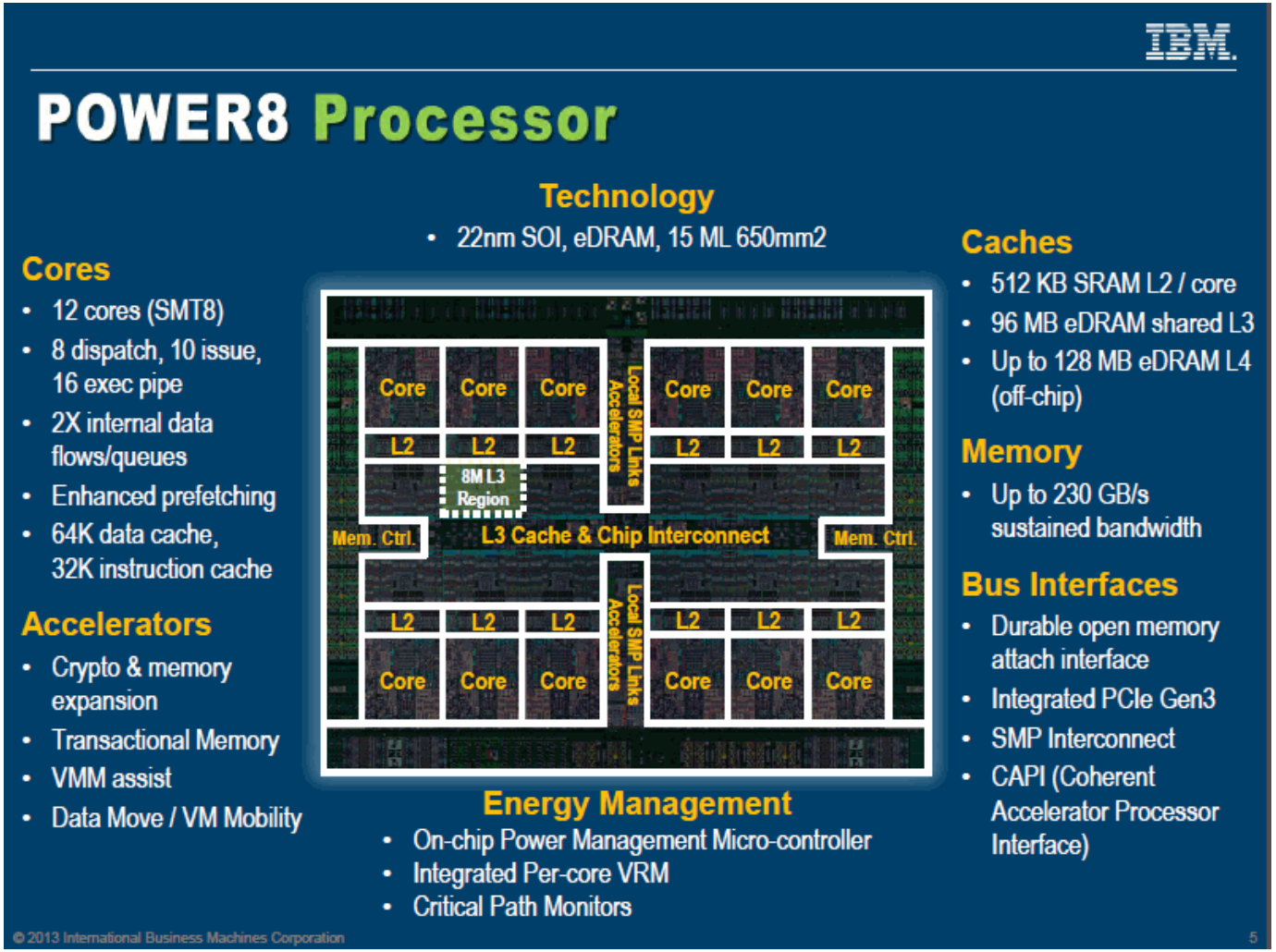
Shared Memory Programming Model



Example: two socket node, with one 6 core CPU per socket. Main memories of each socket can be mapped to a single address space and shared. Here we have Non-Uniform Memory Access (NUMA).

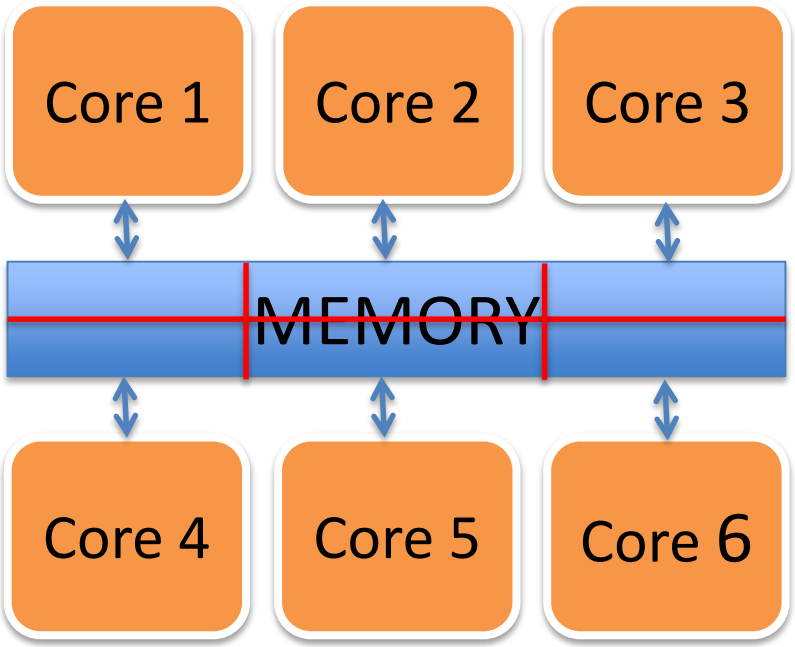
Pros: Larger memory pool.
Cons: data locality becomes and issue.

Shared Memory Programming Model



Main memory is **shared**, L4 an L3 cache are **shared**. L2 cache is **shared** only within a core.

Distributed Memory Programming Model

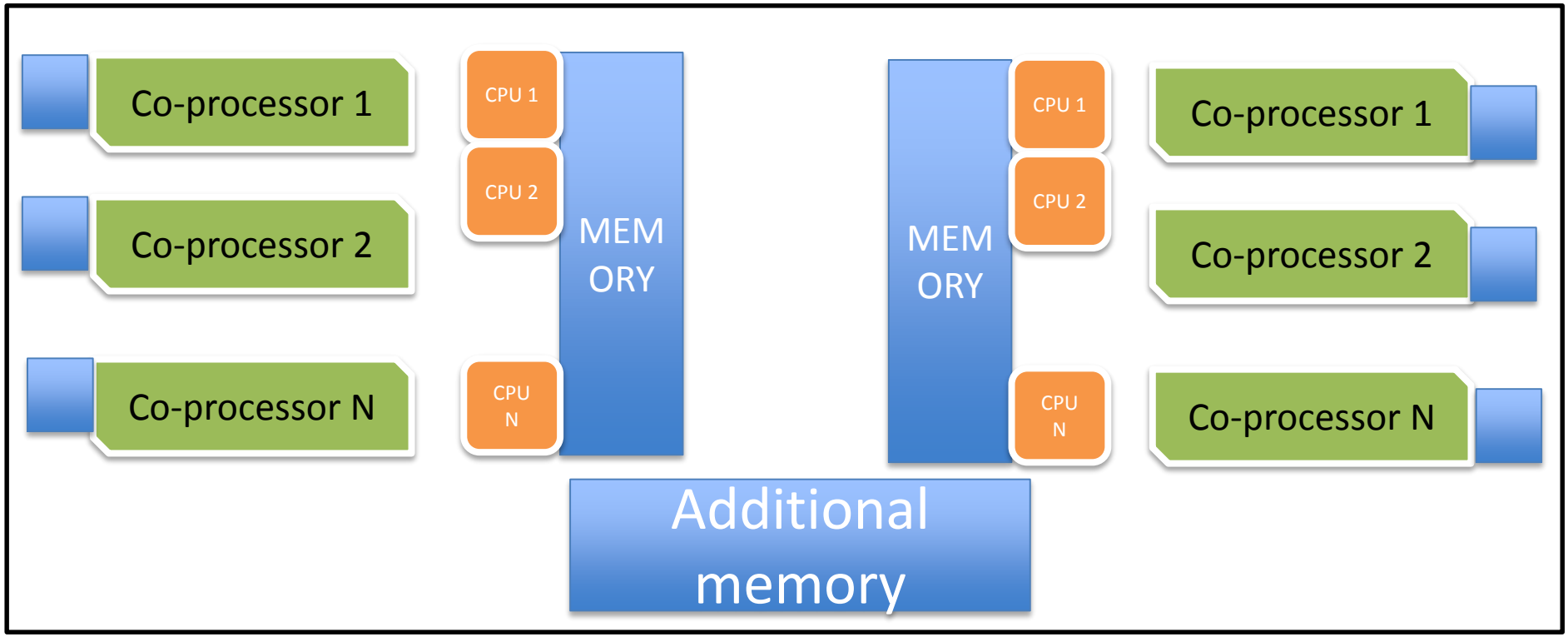


Example: 6 core CPU. Main memory is partitioned into 6 address spaces associated with each core (MPI rank).

Good: No race conditions.
Bad: effectively less memory per core is available for application. Need to perform explicit message passing to exchange data between different address spaces.

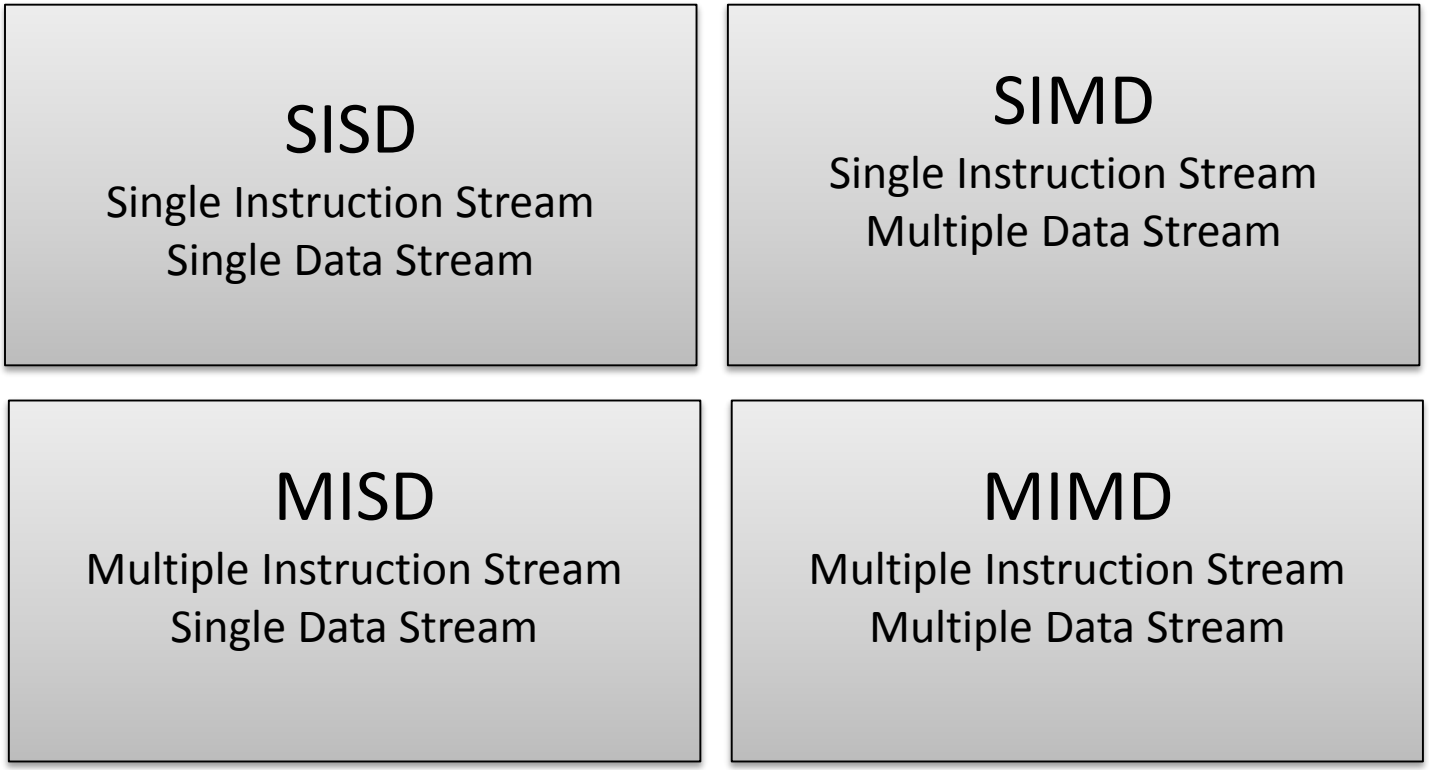
Distributed memory programming model is typically implemented using Message Passing Interface (MPI).

Hybrid Distributed-Shared Memory Programming Model, Multiple Memory Pools



Parallel computers require parallelizable algorithms

Flynn's Classical Taxonomy

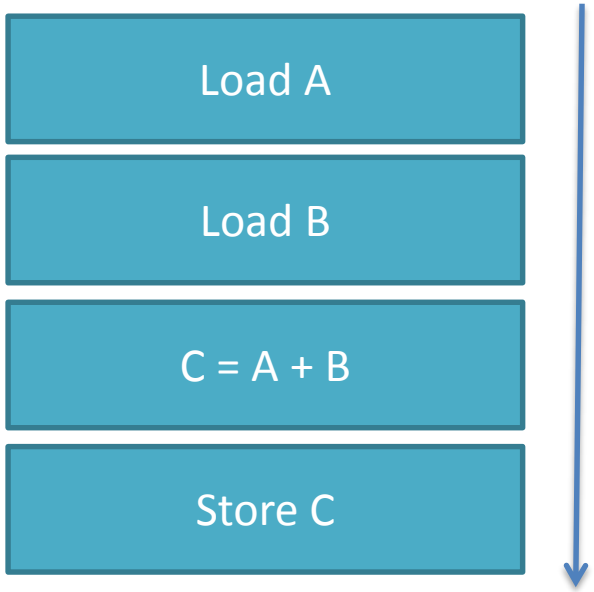


Flynn's Classical Taxonomy

SISD

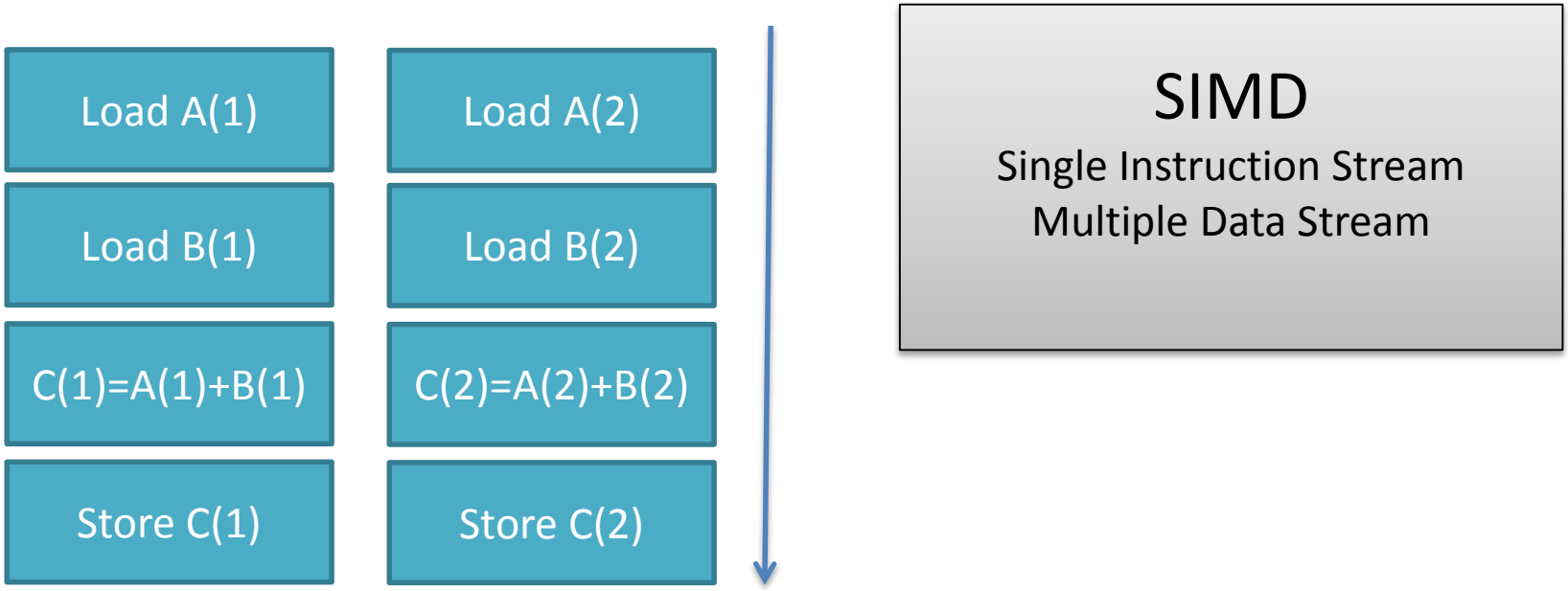
Single Instruction Stream

Single Data Stream



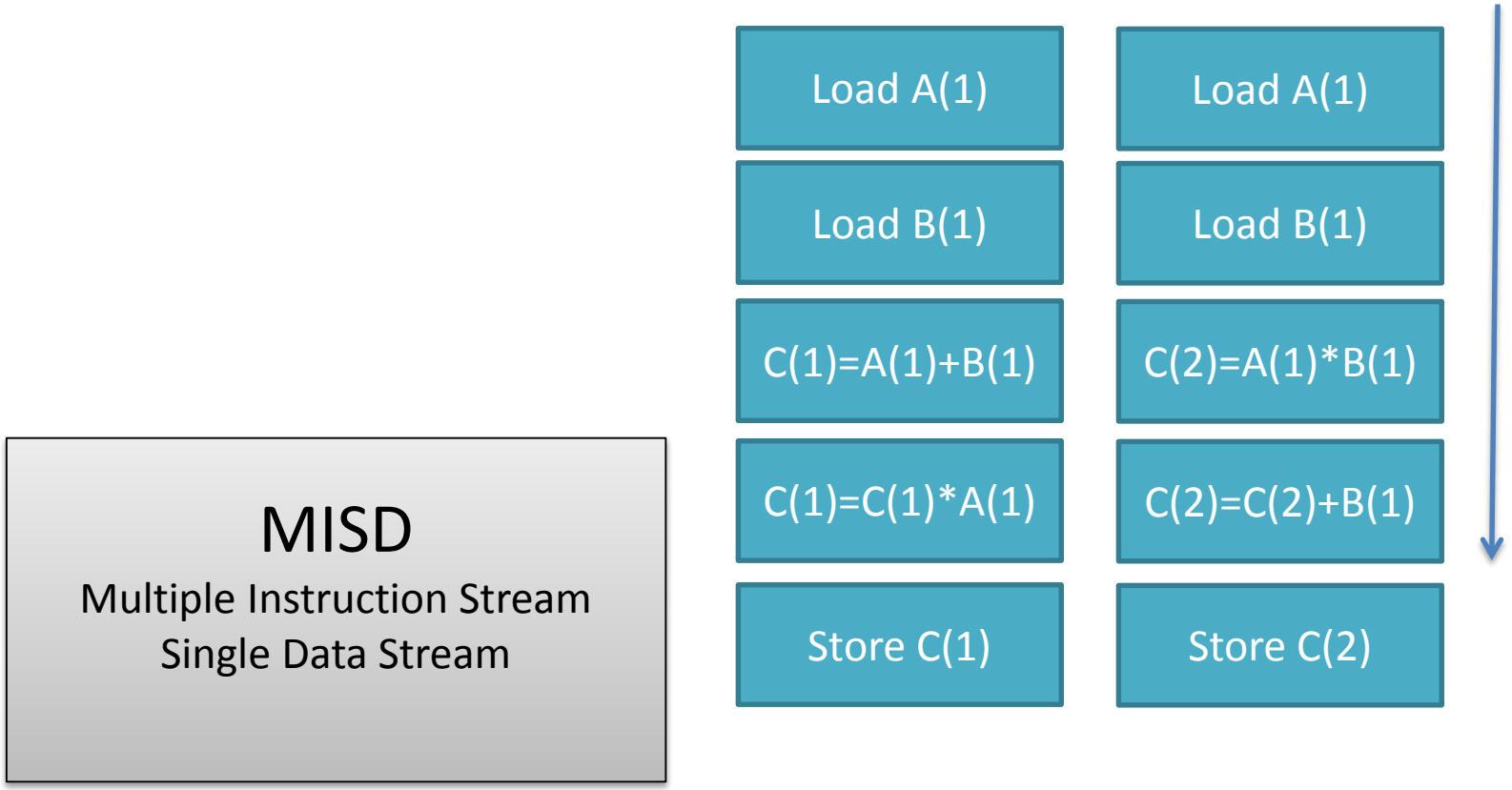
Single Instruction: Only one instruction during any one clock cycle.
Single Data: Only one data stream is being used as input/output during any one clock cycle.
Deterministic execution: (no out of order execution)
No parallelism ….

Flynn's Classical Taxonomy



Single Instruction: Only one instruction during any one clock cycle.
Multiple Data: Multiple data stream is being used as input/output during any one clock cycle.
Data parallelism

Flynn's Classical Taxonomy



Multiple Instructions: Multiple instruction during any one clock cycle.
Single Data: Single data stream is being used as input/output during any one clock cycle. (not very strict - multiple output data)
Instruction Level parallelism

Flynn's Classical Taxonomy

Multiple Instruction: Every processor may be executing a different instruction stream
Multiple Data: Every processor may be working with a different data stream
Execution can be synchronous or asynchronous, in order or out of order
Currently, **the most common type of parallel computer**

MIMD also includes SIMD and SIMT

MIMD
Multiple Instruction Stream
Multiple Data Stream

Paradigms
for
parallel computing

There are several ways to employ many-core computers

1. Use of many cores to execute related solvers concurrently. These cores do not communicate (embarrassingly parallel application)
2. Use of many core to solve one problem (or more) problem(s) at a time with communication between the cores
3. Combination of 1 and 2

Example of embarrassingly parallel application

On CPU 1 do

```
for (i = 1; i < N; i++)
   gzip file_$i.dat
```

On CPU 1 do

```
for (i = 1; i < M; i++)
   gzip file_$i.dat
```

On CPU 2 do

```
for (i = M; i < N; i++)
   gzip file_$i.dat
```

Example of embarrassingly parallel application

Solve a stochastic problem and compute the mean solution: $y(t,x) = \sum_{i=1}^{N} [f(\omega_i, t, x) \cdot w_i]$

On CPU 1 do:

$$for\ (i = 0; i < N; i++)\{$$
$$\quad y_i(t,x) = f(\omega_i, t, x)$$
$$\}$$
$$y(t,x) = \sum_{i=1}^{N} y_i(\omega_i, t, x) \cdot w_i$$

On CPU 1 do:

$$for\ (i = 0; i < M; i++)\{$$
$$\quad y_i(t,x) = f(\omega_i, t, x)$$
$$\}$$
$$y^{*}(t,x) = \sum_{i=1}^{M-1} y_i(\omega_i, t, x) \cdot w_i$$

On CPU 2 do:

$$for\ (i = M; i < N; i++)\{$$
$$\quad y_i(t,x) = f(\omega_i, t, x)$$
$$\}$$
$$y^{**}(t,x) = \sum_{i=M}^{N} y_i(\omega_i, t, x) \cdot w_i$$

$$y(t,x) = y^{*} + y^{**}$$

Paradigm for solution of tightly coupled problem

Solution of stochastic problem: $y(t,x) = \sum_{i=1}^{N} [f(\omega_i, t, x) \cdot w_i]$

On CPUs 1-10000 do:

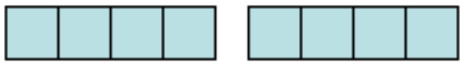
$for\ (i = 0; i < N; i++)\{$

$\quad y_i(t,x) = f(\omega_i, t, x)$

$\}$

$$y(t,x) = \sum_{i=1}^{M-1} y_i(\omega_i, t, x) \cdot w_i$$

Solve in parallel



$x_1 - x_4$ $\quad$ $x_5 - x_8$

Paradigm for solution of loosely-tightly coupled problem

Solution of stochastic problem: $y(t,x) = \sum_{i=1}^{N} [f(\omega_i, t, x) \cdot w_i]$

On CPUs 1-5000 do:

$$for\ (i = 0; i < M; i++)\{$$
$$y_i(t,x) = f(\omega_i, t, x)$$
$$\}$$
$$y^*(t,x) = \sum_{i=1}^{M-1} y_i(\omega_i, t, x) \cdot w_i$$

Solve in parallel using two non-overlapping groups of processors

On CPU 5001-10000 do:

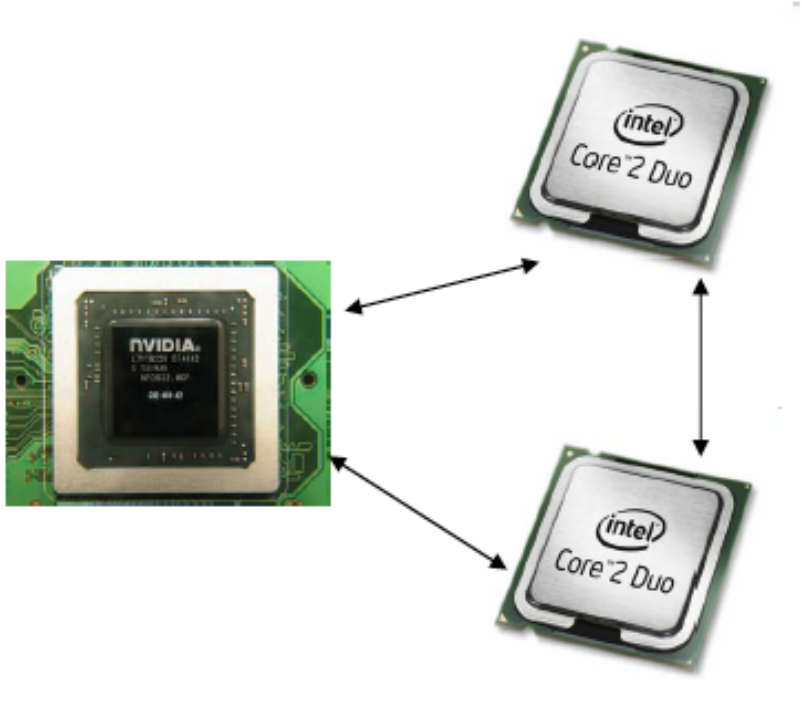
$$for\ (i = M; i < N; i++)\{$$
$$y_i(t,x) = f(\omega_i, t, x)$$
$$\}$$
$$y^{**}(t,x) = \sum_{i=M}^{N} y_i(\omega_i, t, x) \cdot w_i$$

$$y(t,x) = y^* + y^{**}$$

Here we decompose the domain in physical space (x) and in a space of the random variable.

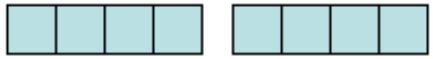
Multiple levels of parallelism

$$y(x) = Ax + Bx + Cx$$



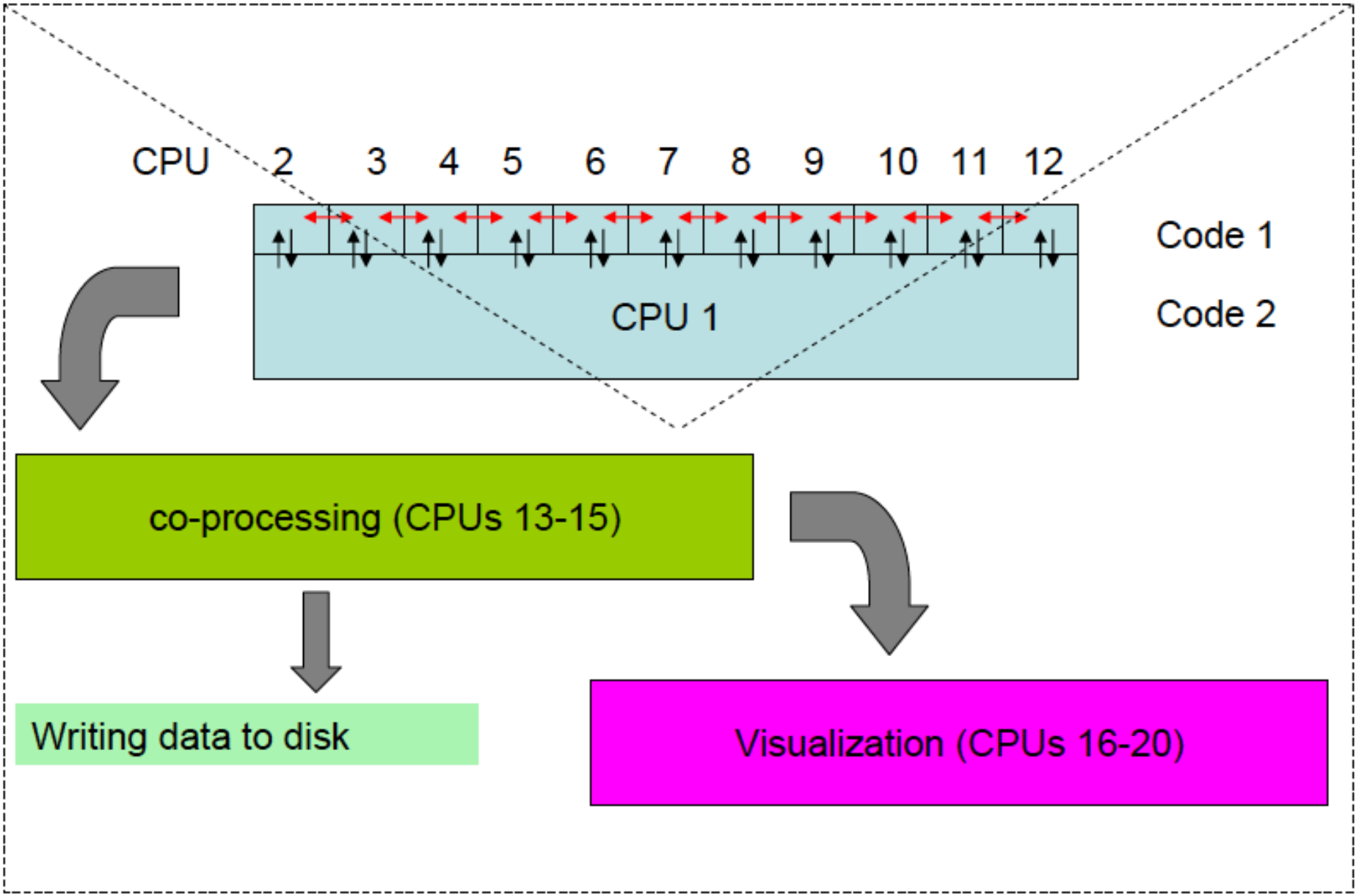
On CPU 1 do Ax; on CPU 2 do Bx; on GPU do Cx

On CPU 1 and 2 do Ax and Bx; on GPU do Cx



$x_1 - x_4$ $x_5 - x_8$

Multilayer parallelism



Black board exercise

Solve numerically

$du/dt = d^2u/dx^2$

use finite differences discretization

1st order in time, 2nd order in space

write a parallel solver