

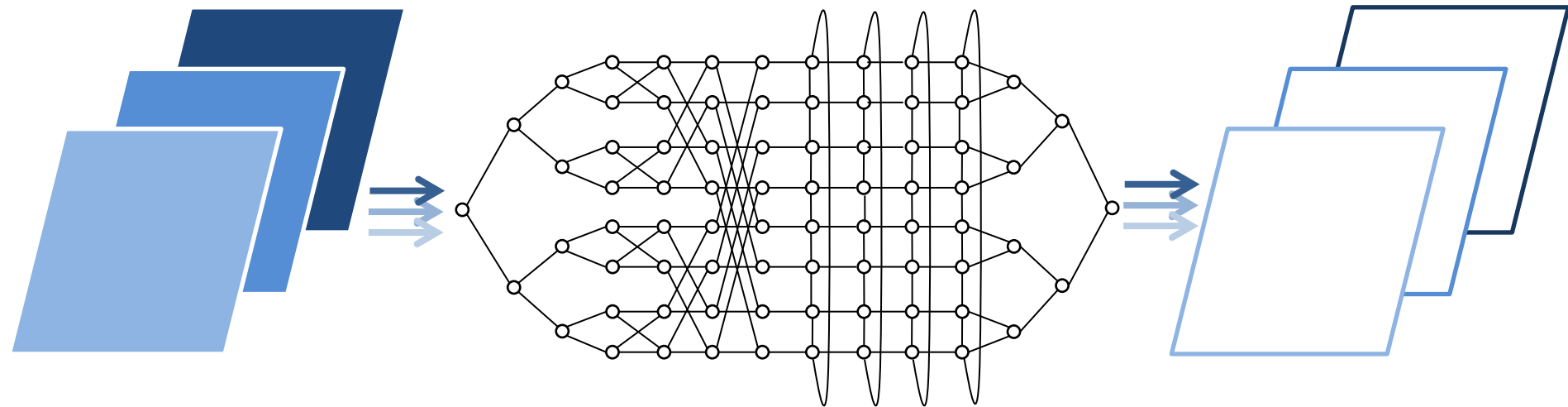
# Programmation Parallèle

## MPI: Message Passing Interface

---

---

---



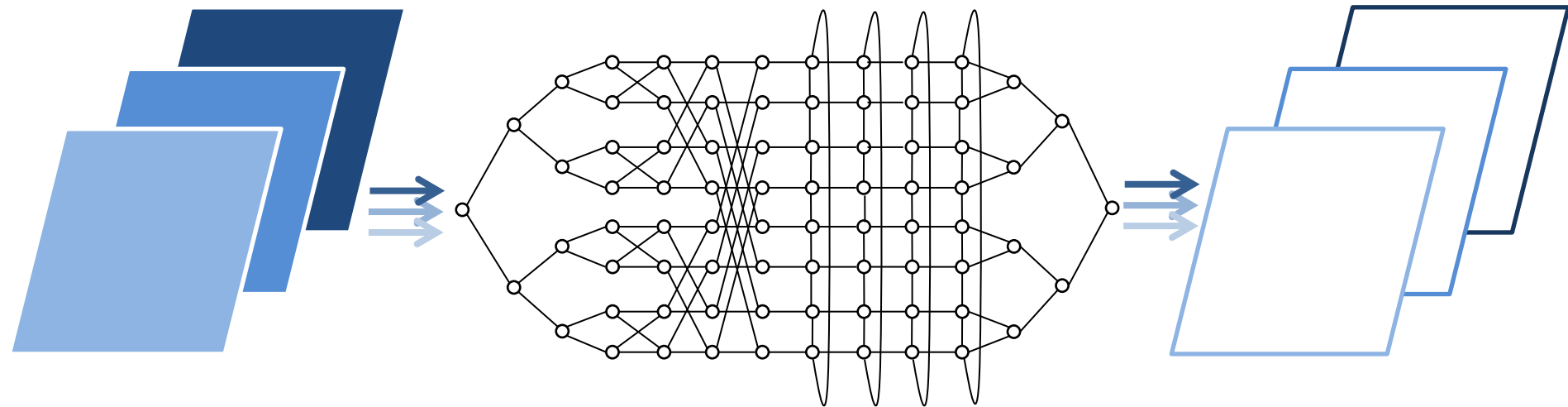
# Introduction to HPC and Parallel Programming

ENSIIE-HPC/BigData-PP-IIP-Lecture 0

---

---

---



# Context



- Why do we need parallel computing?
- How can we exploit main resources from computers?
- What are the different approaches/paradigms of parallelism?

# Context



## ■ Why parallelism?

- Parallel computing is everywhere
- From cellphone processors to supercomputers

## ■ Parallel programming paradigm

- Distributed-memory programming
- Shared-memory programming
- Hybrid programming
- Heterogeneous programming

## ■ Goal

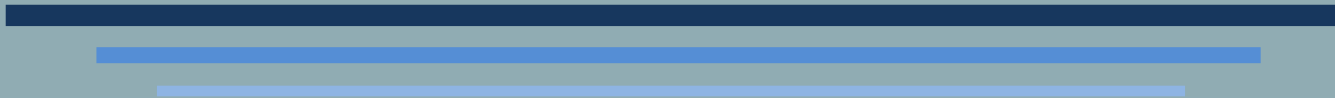
- Identify independent work to exploit parallelism

# Lecture Outline



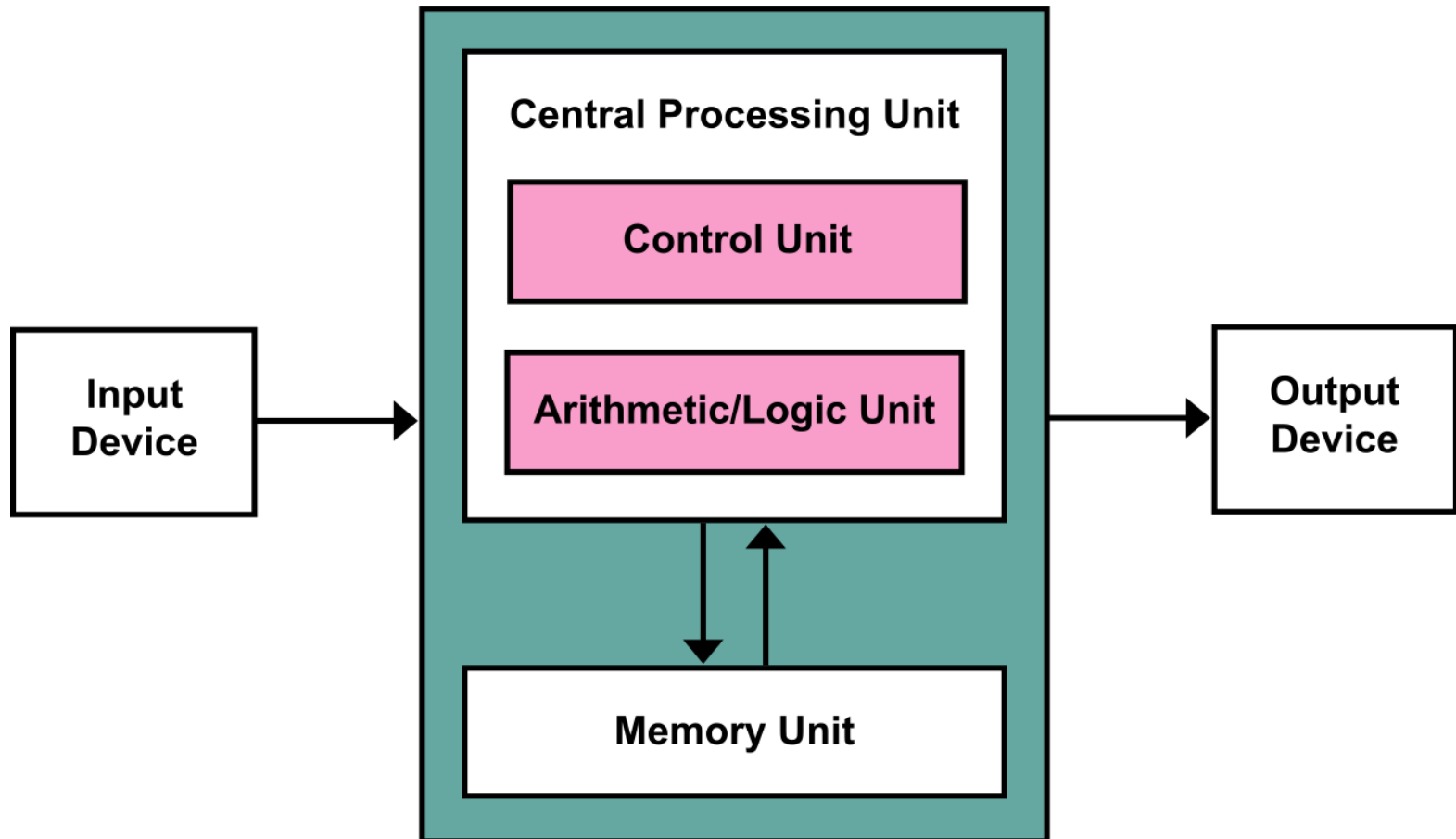
- Hardware Evolution
- Hardware Challenges
- Parallel Computing
- Outline of the course

# HARDWARE EVOLUTION

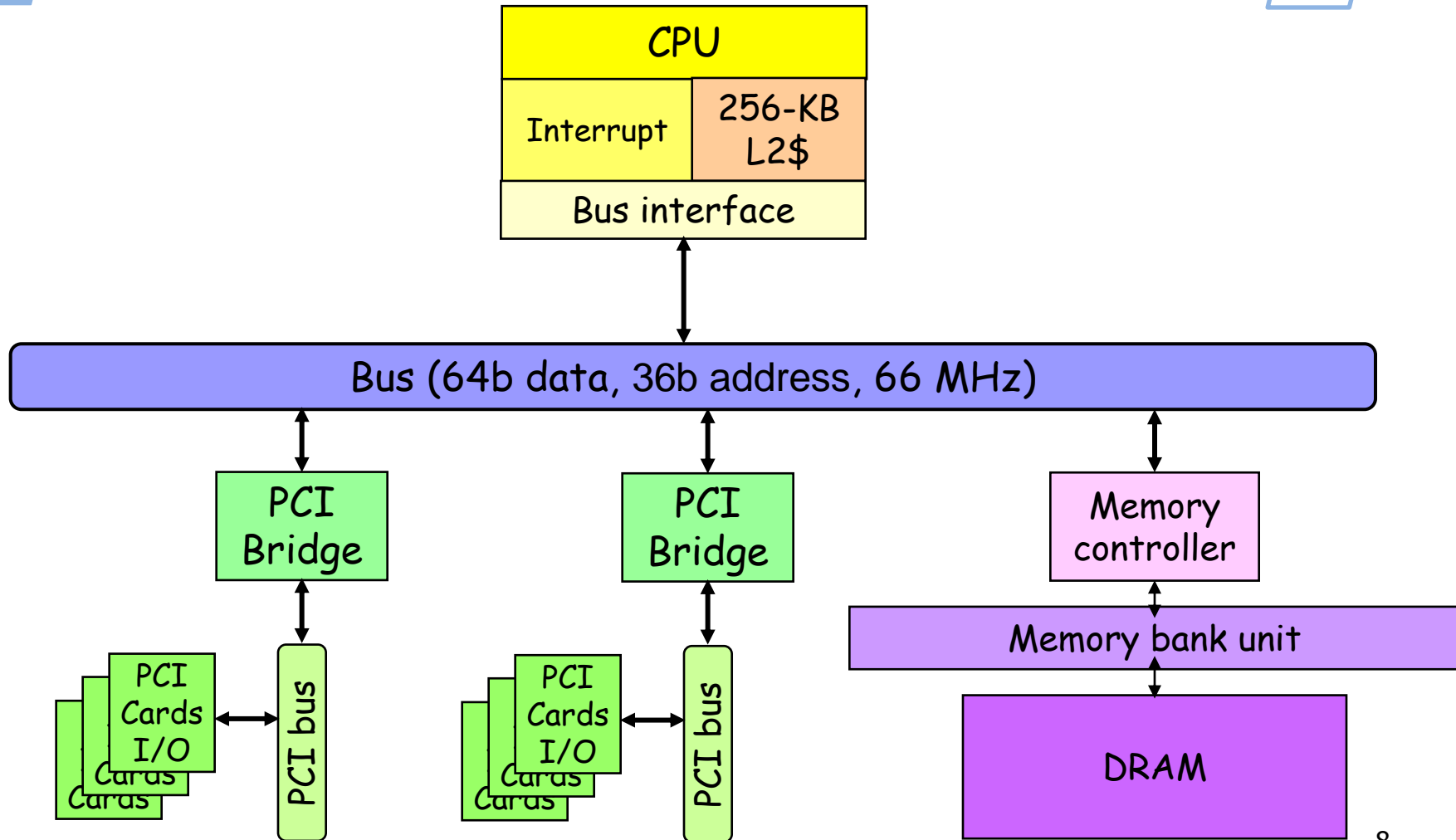


From single-core to clusters

# Von-Neumann Architecture



# More-Accurate Vision





# Processor Architecture



- Architecture
  - What is exposed to a (low-level) developer
- Examples
  - Compute cores
  - Registers
  - Memory model
    - Register-to-register, memory-register, ...
  - Stack management
  - Function call convention
  - Addressing mode
  - Assembly instructions (i.e., ISA)
  - ...

# Architecture vs. Micro-architecture



- Assembly instruction is the smallest atomic visible part executed by a processor
  - This is part of architecture
- But there are many things underneath...
- Micro-architecture is defined as
  - Implementation of ISA
    - For each instruction, various implementations are possible
  - Processor internals that helps implementing target ISA
    - Some mechanisms are mandatory
    - Some are optional and may help for performance, energy, safety...

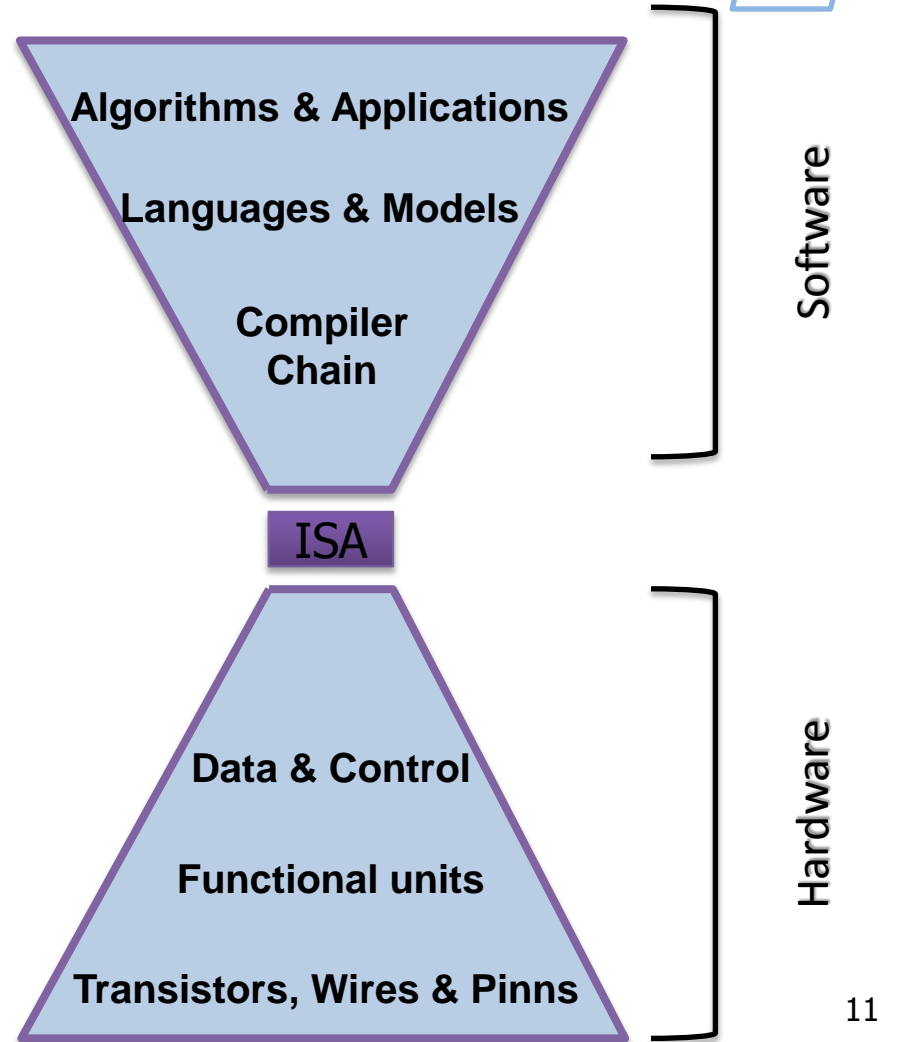
# Architecture Levels

- **Instruction Set Architecture**

- Link between human and machine
- Human readable interface instead of writing code in binary

- **Software stack**

- Offer higher layers of abstraction to efficiently program architectures



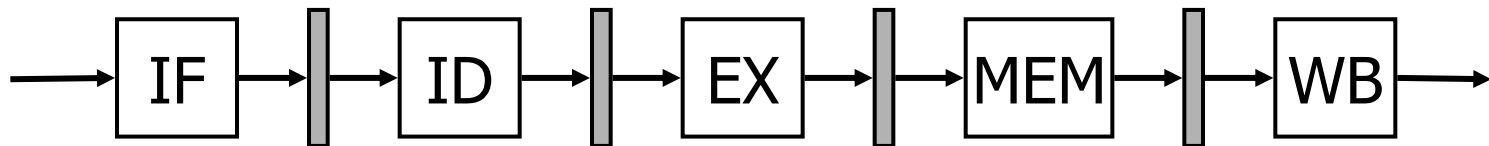
# Performance Improvement



- How to increase the performance of our simple processor?
  - First things first: frequency!
    - Double the frequency will double the overall performance
    - Up to multiple GHz
      - Billions of instructions per second
  - How?
    - Through component optimization
    - With cost reduction of CPU operations
  - Drawbacks: existing limits
    - Power consumption
    - Heat

# Pipeline

- Based on the following idea:
  - *Do not wait for an instruction to be done before starting the next one*
- Impact: split the execution steps into small stages
  - Each instruction goes through all stages
- Example from MIPS: 5 stages
  - IF: instruction fetch
  - ID: instruction decode
  - EX: execution
  - MEM: memory access
  - WB: write back



# Pipeline

## ■ Several limits

- Performance based on longest stage
- Hazards

## ■ Longest stages

- Optimize each stage
- Increase the number of stages

## ■ Hazards

- Structural hazards
  - Pipeline blocked because of hardware resources
- Data hazards
  - Pipeline blocked because of data dependencies
- Control hazards
  - Pipeline blocked because of branch instruction

Duplicate hardware

Out-of-order execution

Branch prediction

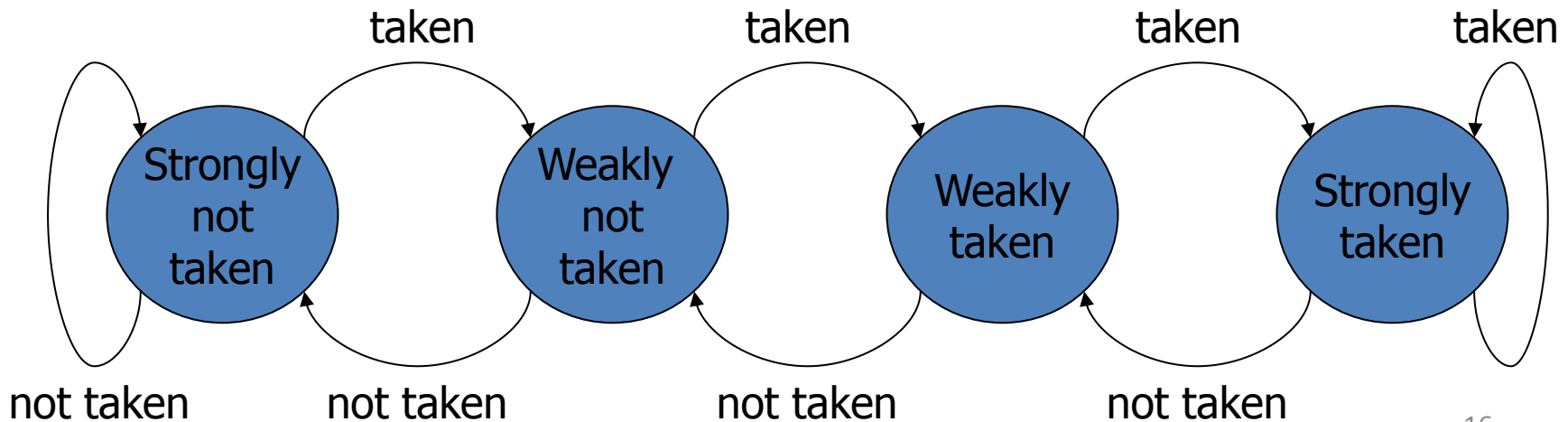
# Out-of-Order Execution



- Main idea: escape from linear assembly
  - If next instruction cannot be executed, check the one after and execute it if no dependencies remain
- Basic algorithm: Tomasulo (in 60s)
  - Extension of scoreboard technique
  - 3 stage: Issue/Execute/Write
    - Issue if reservation station available and operand update
    - Execute if operand available
    - Write when execution is done and propagate result
- Advantages
  - False-dependency removal
  - Functional-unit abstraction
  - Register renaming

# Branch Prediction

- Branch instruction
  - Need to wait for the outcome to be resolved
  - Ability to *guess* the outcome...
- Finite-state automaton
  - Example w/ 2-bit saturating up-down counter
  - More complex implementation w/ history level





# Flynn Taxonomy



---

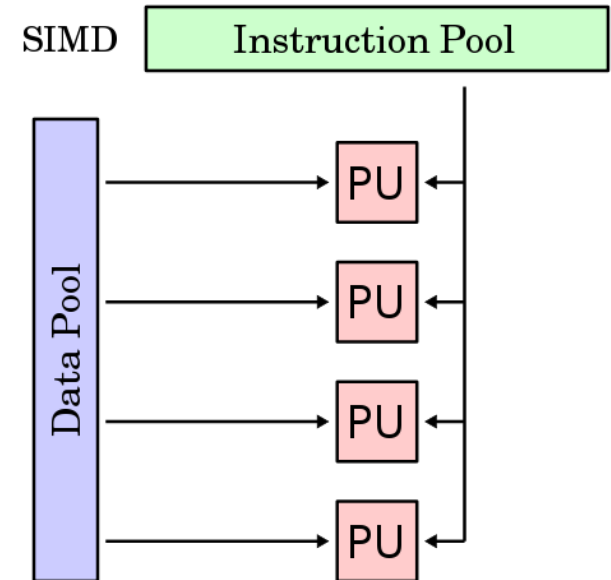
---

---

- But we consider only one specific scenario
  - We retire one instruction at a time performing one operation on one data
- What about extending that?
  - Lead to Flynn taxonomy
  - Depend on the number of concurrent instructions and data streams
- Single instruction, single data (SISD)
- Single instruction, multiple data (SIMD)
- Multiple instructions, single data (MISD)
- Multiple instructions, multiple data (MIMD)

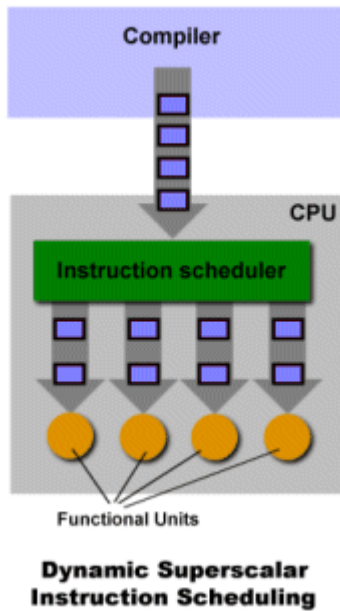
# Single Instruction, Multiple Data

- Enlarge functional units to process one instruction with multiple data
  - Notion of vectors
- Advantages
  - Improve overall peak performance with reduced design cost
  - Need more transistors, but some operations are very simple (e.g., addition)
  - No need to change the rest of the processor
- Drawbacks
  - Architectural mechanism
    - New ISA: SSE (128b), AVX (256), AVX512 (512b)
  - Need to express a large parallelism degree
    - E.g., 16 flots on Intel Xeon Phi processors
  - Suitable for regular codes

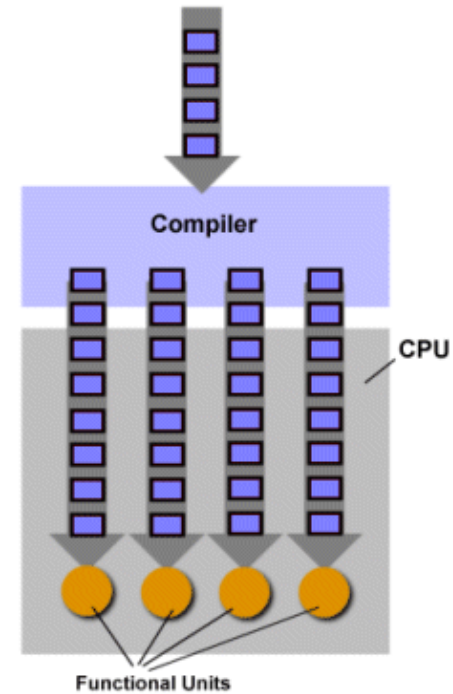


# Multiple instructions


## Superscalar



## VLIW



# Beyond Core

- 
- But we consider only one instruction stream
    - How can we go further?
  - Solution #1
    - Performance of one single regular processor is limited
    - Can use transistors to duplicate some parts or even almost the whole processor!
    - → Multicores
  - Solution #2
    - Exploit processor stalls with multiple instruction streams
    - → SMT (Intel Hyperthreading)
    - Need to duplicate only parts of the processors

# Multicore Limits



- Multicore allows performance extension
  - New transistors are dedicated to functional units
  - Double the number of cores will double the performance (in theory!)
- Total number of cores still increasing
  - 10 years ago: 2
  - Currently: 16 to 20
- But some parts of the processor do not scale very well with the number of cores!
  - Cache coherency
  - Memory access

# Intel Strategy

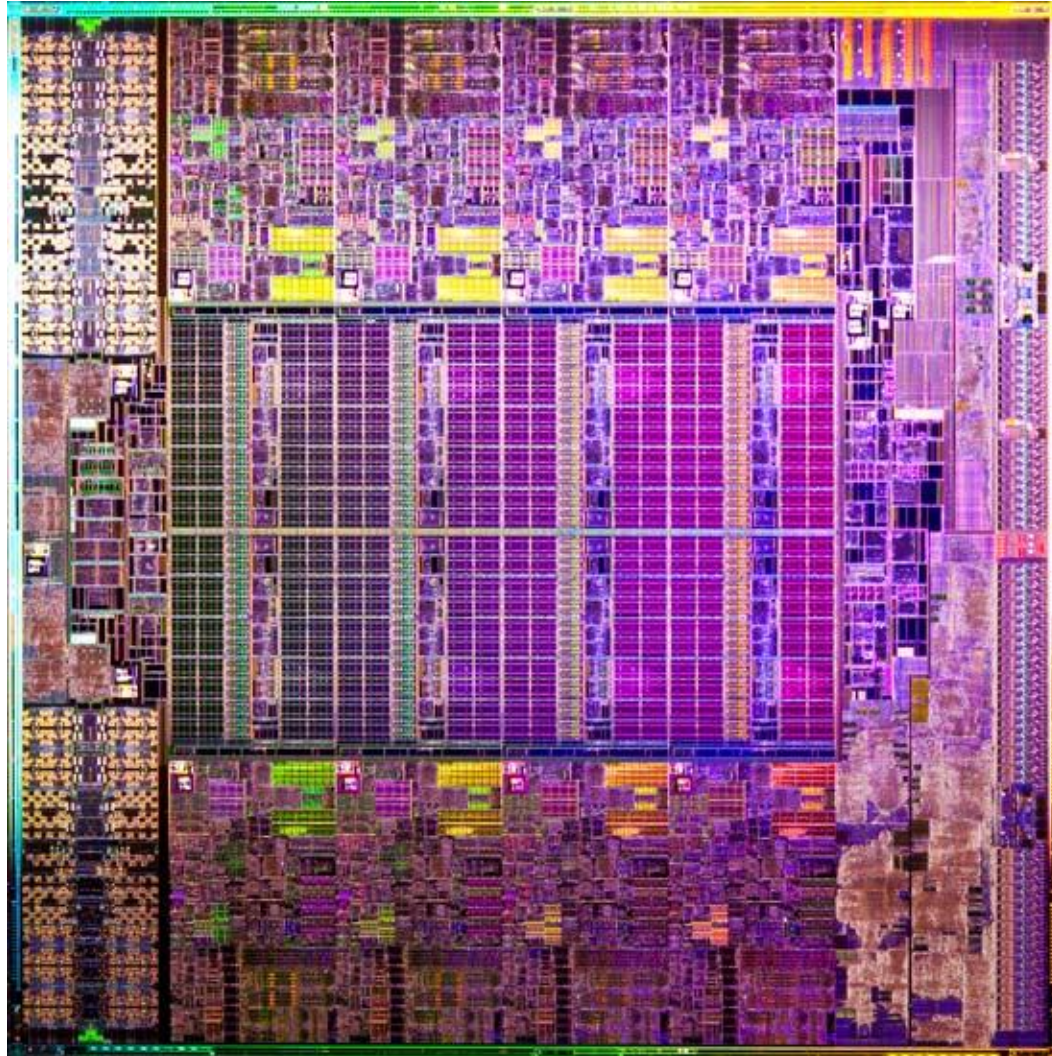
## Intel® Xeon® Processor Roadmap Plan for HPC



Potential future options, subject to change without notice. Codenames.  
All timeframes, features, products and dates are preliminary forecasts and subject to change without further notification.



# Intel Xeon Sandy Bridge



# Intel Xeon Sandy Bridge

## Xeon E5-2600 Block Diagram

- **Bi-Directional Full Ring**

- 32B/clock/agent
- 8 Core/LLC slices

- **Last Level Cache**

- 32B/clock/slice

- **Dual QPI Agent**

- **Integrated I/O**

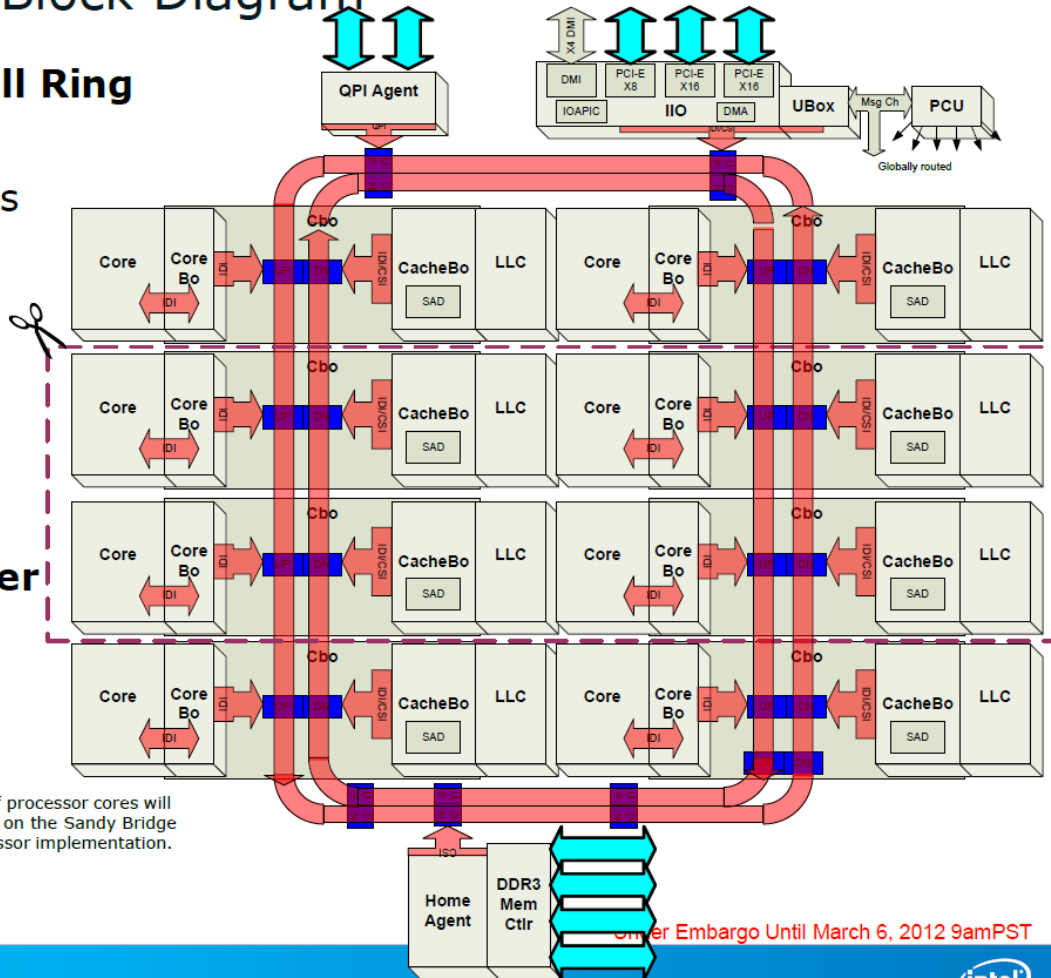
- **Home Agent**

- **Integrated Memory Controller**

- Connected to HA

- **PCU**

- **"Ubox"**



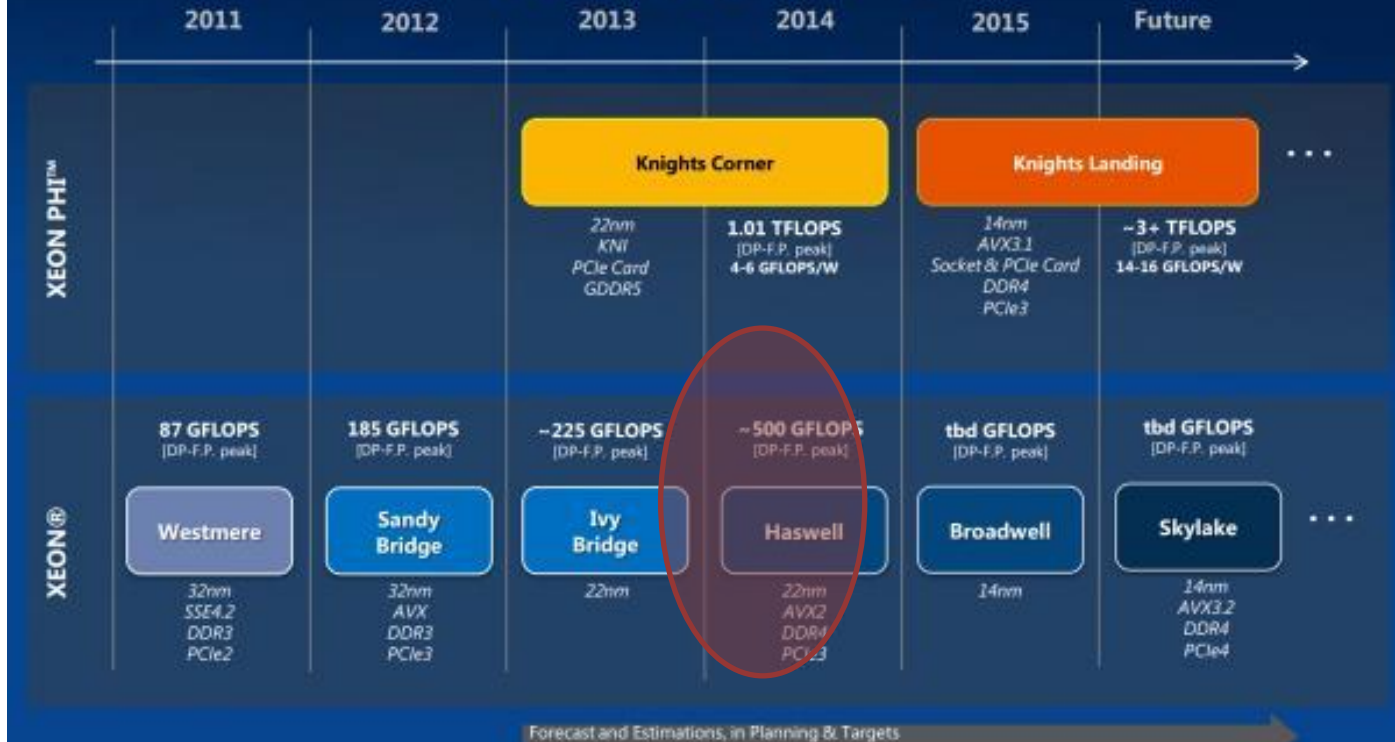
Block Diagram Illustrative only. Number of processor cores will vary with different processor models based on the Sandy Bridge Microarchitecture. Represents server processor implementation.

Under Embargo Until March 6, 2012 9am PST



# Intel Strategy

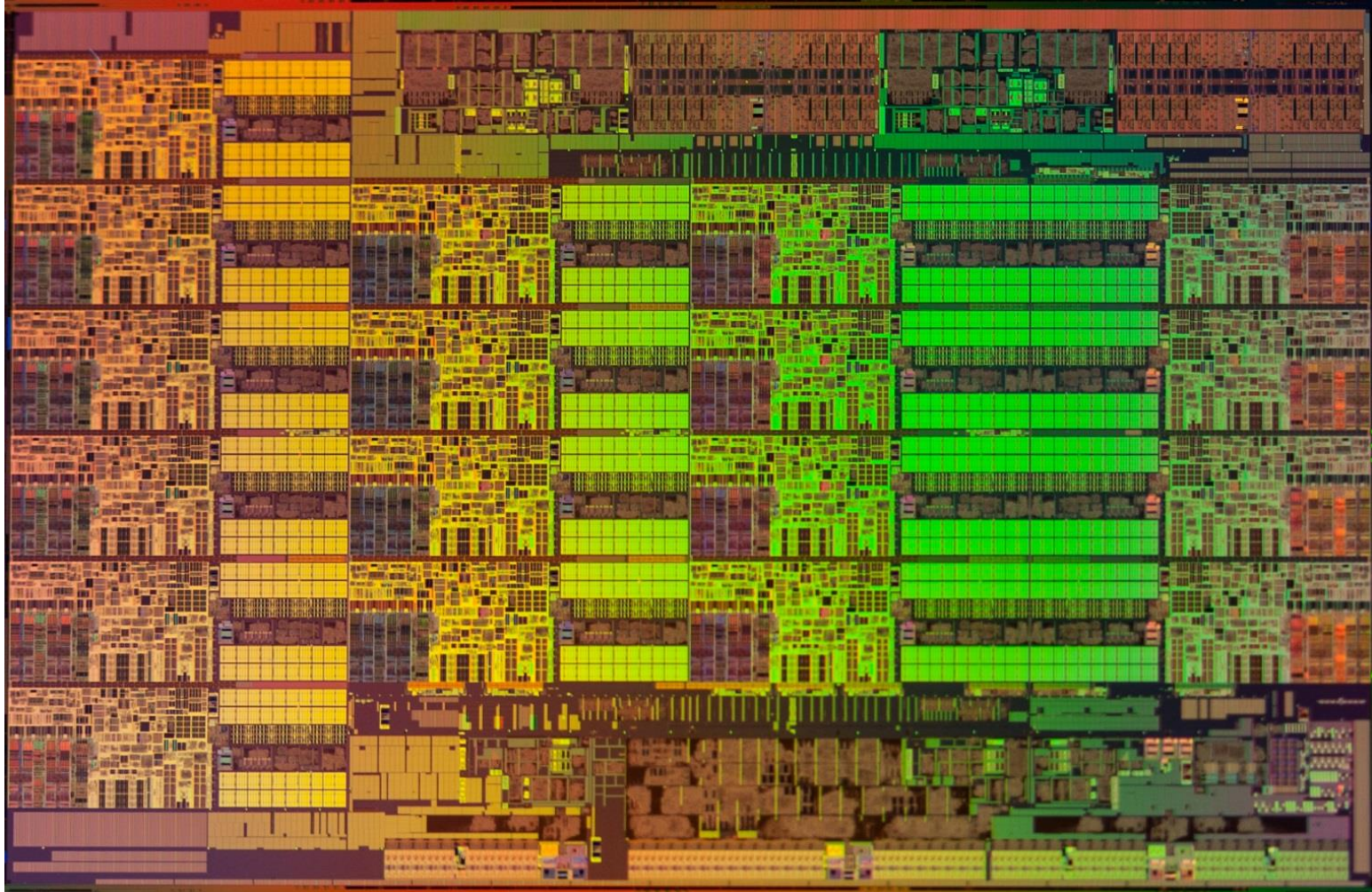
## Intel® Xeon® Processor Roadmap Plan for HPC



Potential future options, subject to change without notice. Codenames.  
 All timeframes, features, products and dates are preliminary forecasts and subject to change without further notification.

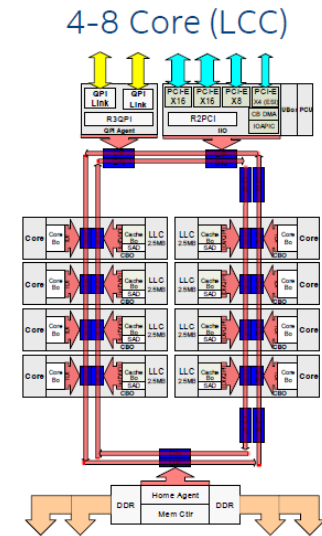
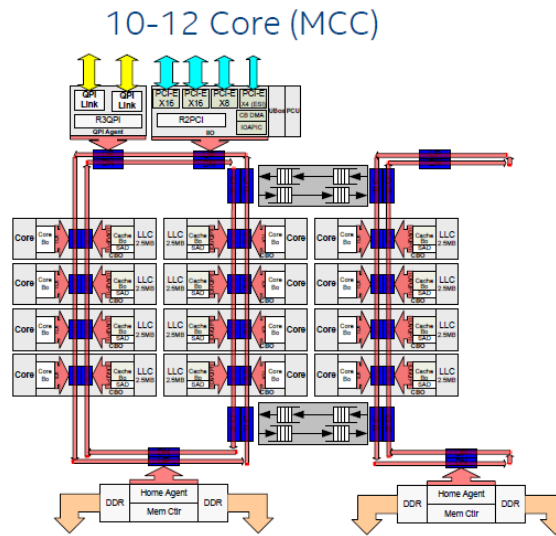
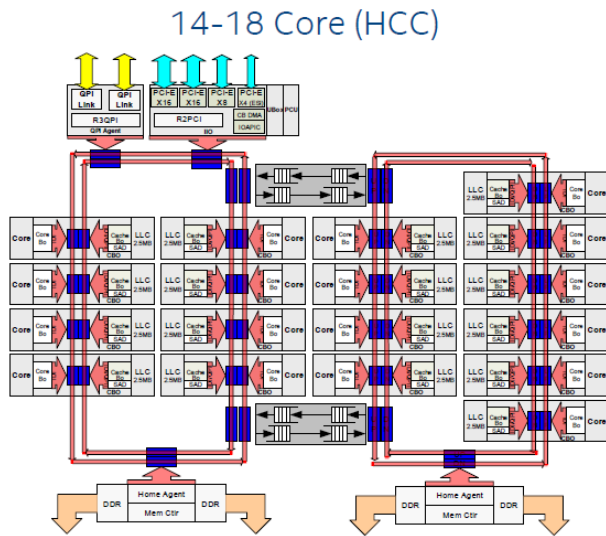


# Intel Xeon Haswell



# Intel Xeon Haswell

## Haswell EP Die Configurations



Not representative of actual die-sizes, orientation and layouts – for informational use only.

Chop	Columns	Home Agents	Cores	Power (W)	Transistors (B)	Die Area (mm <sup>2</sup> )
HCC	4	2	14-18	110-145	5.69	662
MCC	3	2	6-12	65-160	3.84	492
LCC	2	1	4-8	55-140	2.60	354

# Intel Strategy

## Intel® Xeon® Processor Roadmap Plan for HPC

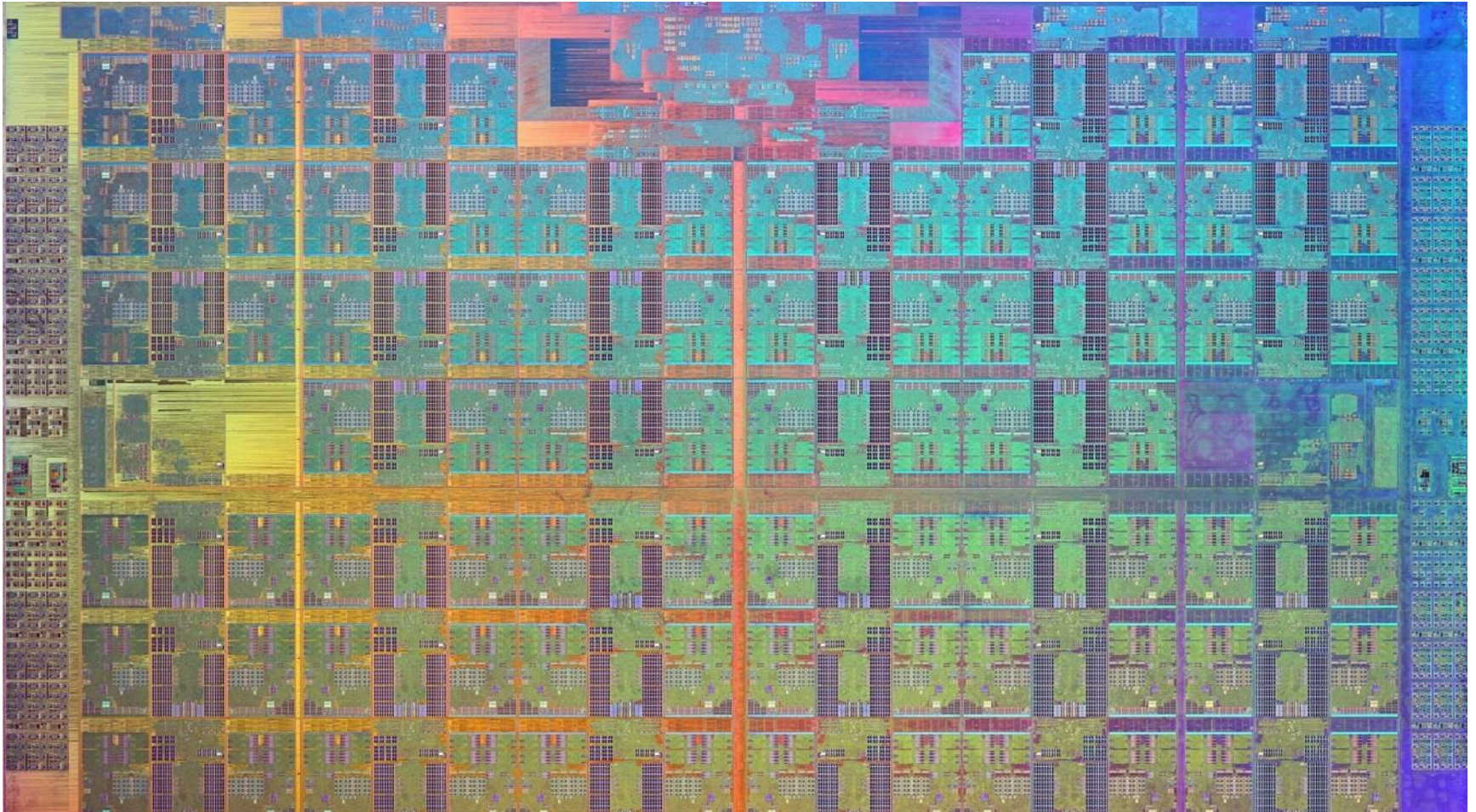


Forecast and Estimations, in Planning & Targets

Potential future options, subject to change without notice. Codenames.  
All timeframes, features, products and dates are preliminary forecasts and subject to change without further notification.



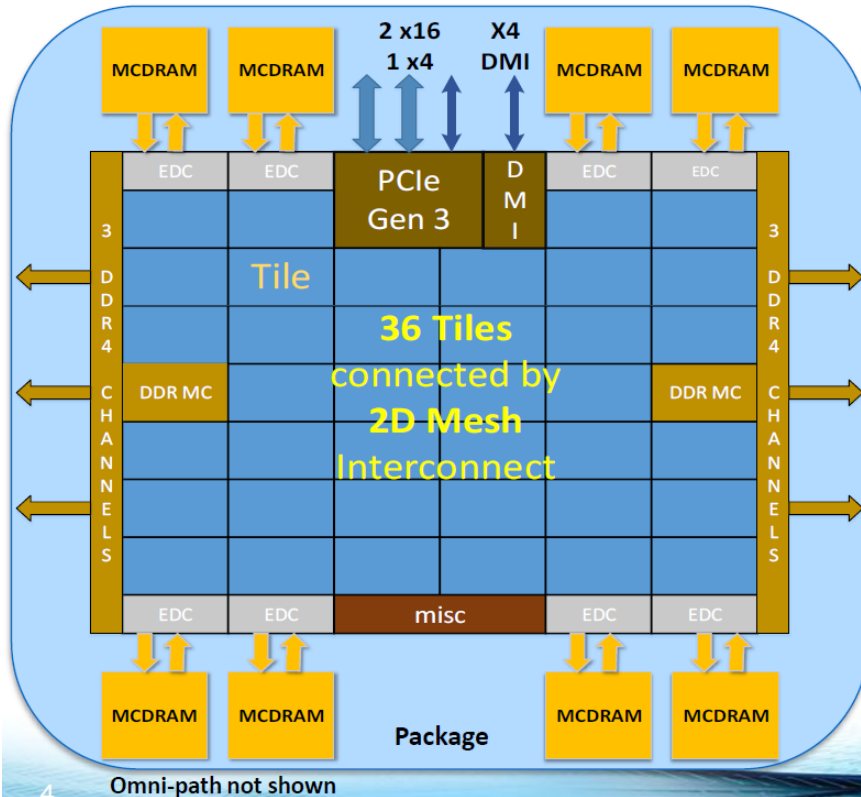
# Intel Xeon Phi



# Intel Xeon Phi

## Knights Landing Overview

<b>TILE</b>	2 VPU	CHA	2 VPU
	Core	1MB L2	Core



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

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

**Memory: MCDRAM: 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 only**

**Fabric: Omni-Path on-package (not shown)**

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

**Scalar Perf: ~3x over Knights Corner**

**Streams Triad (GB/s): MCDRAM : 400+; DDR: 90+**

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. 1 Binary Compatible with Intel Xeon processors using Haswell Instruction Set (except TSX). 2 Bandwidth 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.

# Beyond Processor



- Maintaining full performance on the same die is challenging
  - Nearly impossible to add more chips due to heat dissipation
  - How to go further to improve performance?
- Solution: put multiple processors together
  - Each processor can be multicore or manycore
  - Classical example: dual socket
- Unified shared memory but
  - Non-Uniform Memory Access (NUMA)
  - Needed to increase local performance
- Lead to the composition of one compute node
  - Can be augmented with additional compute card like NVIDIA GPGPU

# Towards Clusters

- But one node is not enough!
  - Group of multiple nodes
  - Name: cluster
  - Current structure of supercomputers
- Nodes linked by network
  - High-speed network
    - IB, BXI, OPA, Aries, Tofu...
- Need different nodes
  - Login nodes
  - IO nodes
  - Computational nodes (maybe multiple types)
  - ...
- Whole machine → distributed memory system



# TERA 100

- First European *Petaflop* machine

- Bull system installed at CEA in 2010

- Overview

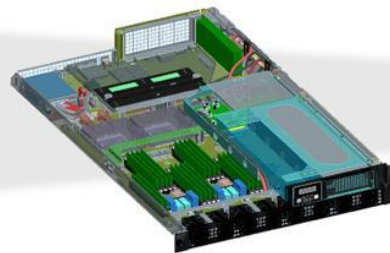
MACHINE



CABINET



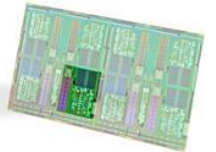
COMPUTE NODE



MULTICORE CPU



COMPUTE CORE



TERA-100

220 CABINETS

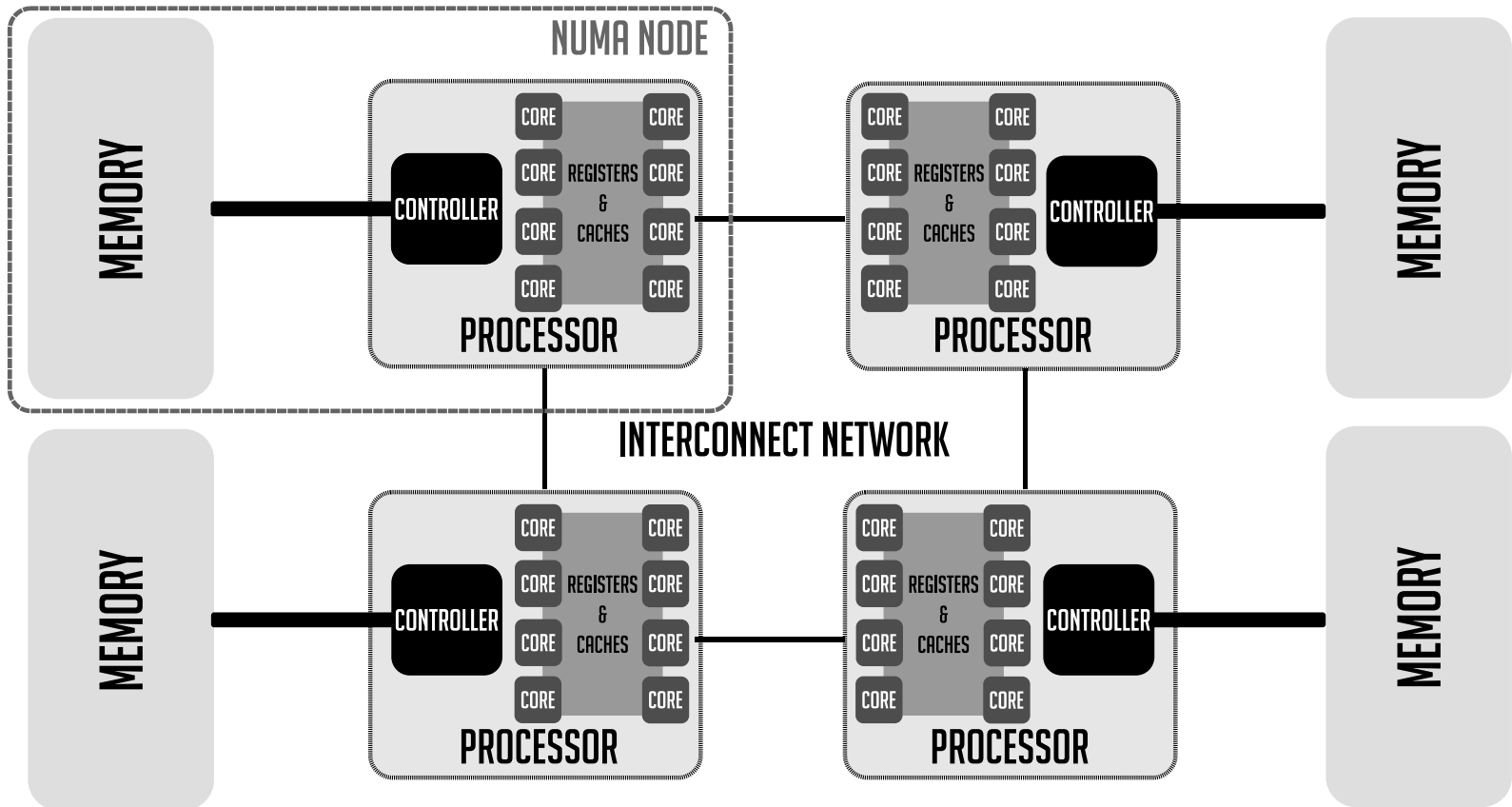
~20 NODES PER CABINET  
TOTAL OF 4,370 NODES

4 CPUS PER NODE  
TOTAL OF 17,480 CPUS

8 CORES PER CPU  
TOTAL OF 139,840 CORES

# TERA 100

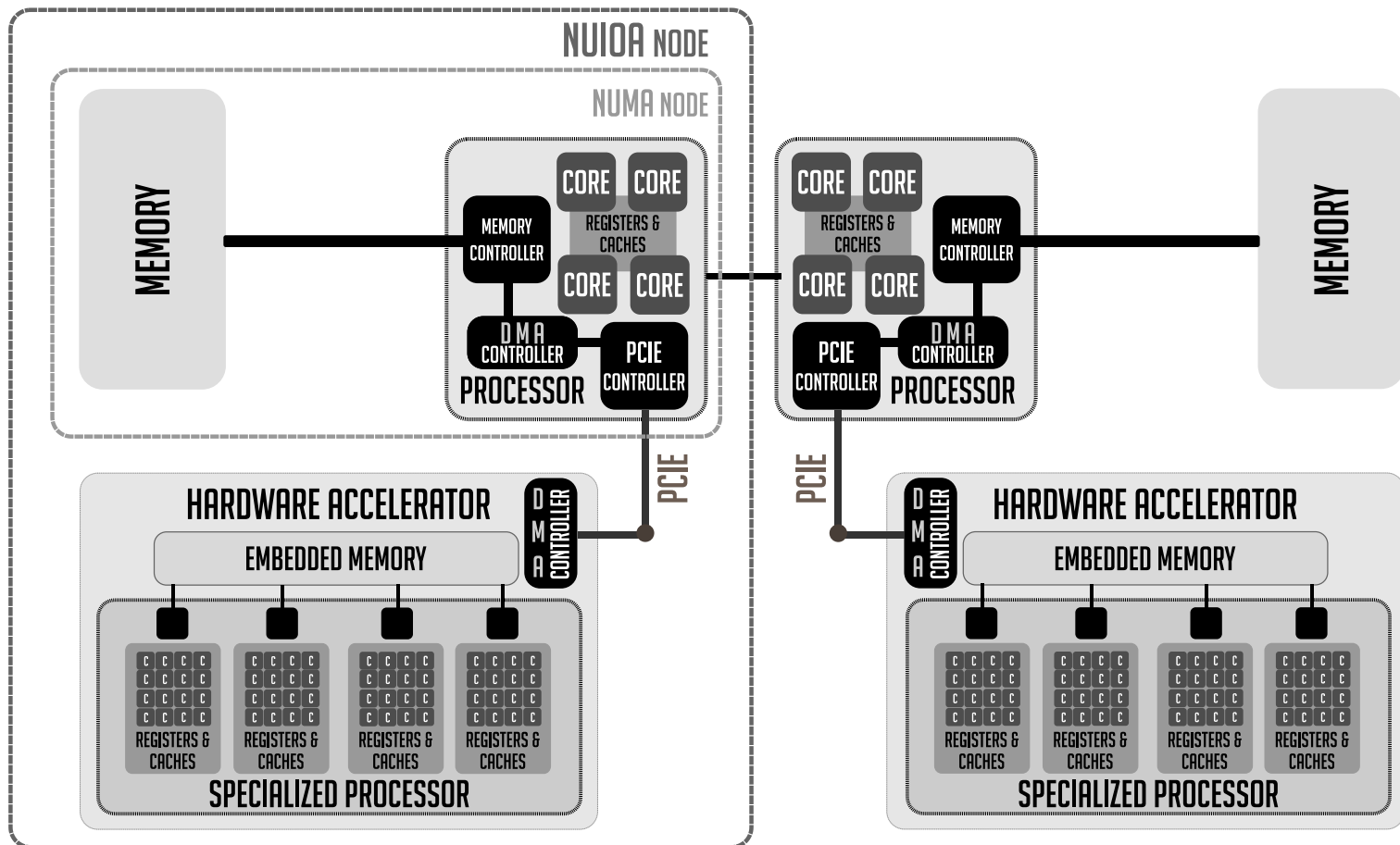
- Regular node



# TERA 100



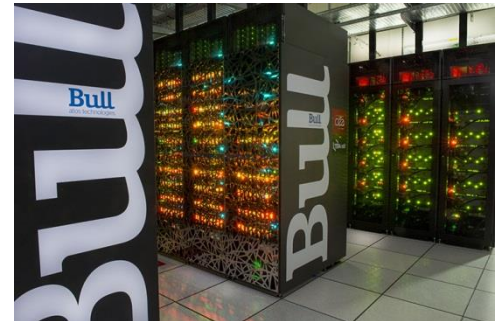
## ■ Heterogeneous node



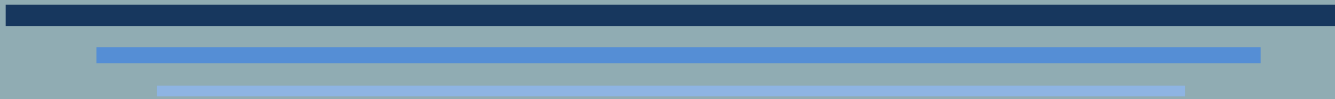
# TERA 1000

- Co-design w/ Atos & Intel
- Organized into 2 phases
  - Tera 1000-1
    - Intel Xeon Haswell
    - Dual-socket 16-core CPUs
    - IB network
  - Tera 1000-2
    - Intel Xeon Phi KNL
    - BXI network

TERA 1000



# SUPERCOMPUTERS OVERVIEW



Classification and french ecosystem

# Supercomputer Classification

- Small application to compare machines
  - Benchmark or *miniapp* or *proxyapp*
  - Results → metrics able to compare machines
- Example: **Top500**
  - Rank machines according to the computational power on regular codes
  - Homepage: <http://www.top500.org>
- Benchmark: `Linpack`
  - Linear solver based on linear algebra
  - Relies on performance of DGEMM
  - Towards HPCG (Conjugate Gradient)



# Top500

- List of 500 most powerful machines

- Measure mainly the computational power
- According to Linpack results

- Updated twice a year

- June: ISC conference in Germany
- November: SC conference in US

- Machine Information

- Main info (rank, site)
- System (name and short description)
- Number of cores
- Performance (R<sub>max</sub>, R<sub>peak</sub>)
- Power

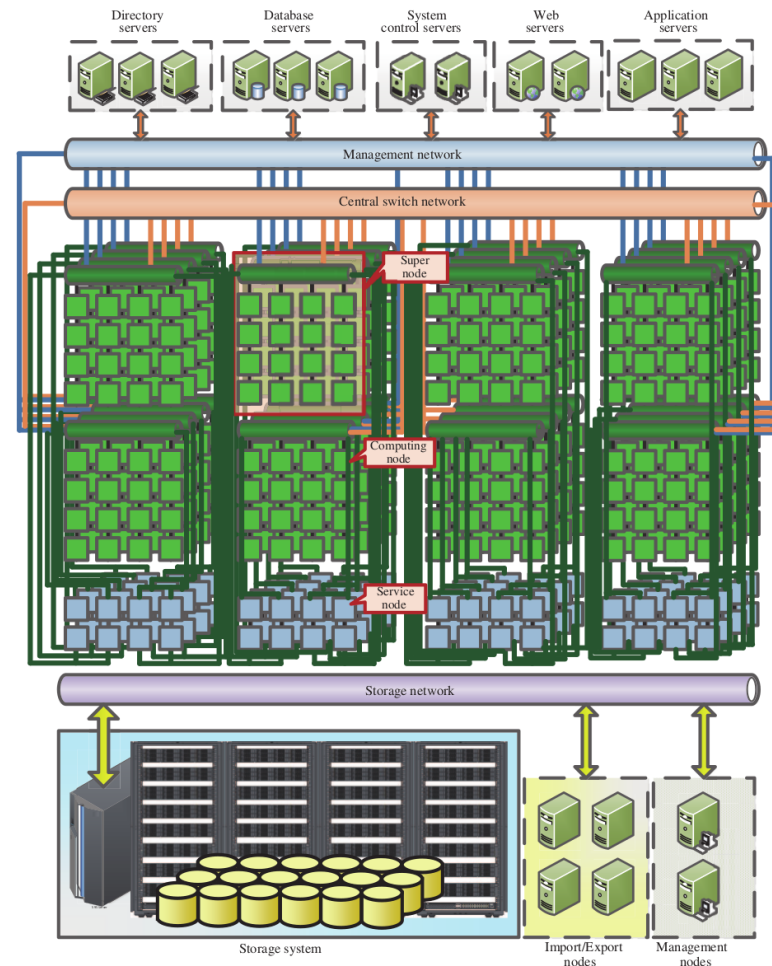
- Notes

- Performance in Tflops/s ( $10^{12}$  floating-point operations per second)
- Difference between max performance (R<sub>max</sub>) and Linpack result (R<sub>peak</sub>)
- Power measured in kW



# Top500 #1

- Sunway TaihuLight
- Total: 40,960 CPUs
  - SW26010
  - 64bit RISC processors
  - 256 cores per chip
  - 64KB scratchpad



Source: Report on the Sunway TaihuLight System by Jack Dongarra



# Top500 (#1 to #5)

Rank	Country	System	Cores	Rmax	Rpeak	Power
1	China	Sunway TaihuLight	10,649,600	93,014.6	125,435.9	15,371
2	China	Tianhe-2	3,120,000	33,862.7	54,902.4	17,808
3	United States	Titan	560,640	17,590.0	27,112.5	8,209
4	United States	Sequoia	1,572,864	17,173.2	20,132.7	7,890
5	United States	Cori	622,336	14,014.7	27,880.7	3,939



# Top500 (#6 to #10)

Rank	Country	System	Cores	Rmax	Rpeak	Power
6	Japan	Oakforest-PACS	556,104	13,554.6	24,913.5	2,719
7	Japan	K	705,024	10,510.0	11,280.4	12,660
8	Swiss	Piz Daint	206,720	9,779.0	15,988.0	1,312
9	United States	Mira	786,432	8,586.6	10,066.3	3,945
10	United States	Trinity	301,056	8,100.9	11,078.9	4,233



# Top500 Analysis



- First comments
  - Ability to reach almost 100 Pflops
    - $10^{17}$  floating-point operations per second
  - Machines with lot of *cores*
  - Power consumption up to 17 Mwatts
  - Top 10 exhibits different system architectures
- Deeper analysis
  - Big difference between *Rmax* and *Rpeak*
  - Big difference between *Rmax* and *Power*
- Ordering based on power efficiency : **Green500**
  - Sort supercomputers according to the ratio power consumption / Linpack performance

# Green500



R	Top500	System	Cores	Rmax	Rpeak	Mflops/W
1	28	DGX SaturnV	60,512	3,307	4,896.5	9,462.09
2	8	Piz Daint	206,720	9,779	15,988	7,453.51
3	116	Shoubu	1,313,280	1,001	1,533.5	6,673.84
4	1	Sunway TaihuLight	10,649,600	93,014.6	125,435.9	6,051.3
5	375	QPACE3	18,432	447.1	766.8	5,806.32



# Green500 Analysis



- Main ordering
  - First machine is not the most *powerful*
    - Rank in Top500
  - Large difference between first and second machine: 27%
  - Specific architecture seems to be more efficient
- Top500 and Green500 limits
  - Linpack is a very specific benchmark
  - Regular computation (mainly linear algebra)
  - Few communications/synchronization between parallel units
- Need different benchmarks to classify supercomputers
  - Most powerful machines on irregular codes: Graph500
  - Based on graph traversal
  - GTEPS: Billions of edges traversed per second

# Graph500



R	Top500	System	Nodes	Cores	Pb Scale	GTEPS
1	7	K	82,944	663,552	40	38,621.4
2	1	Sunway TaihuLight	40,768	105,99680	40	23,755.7
3	4	Sequoia	98,304	157,2864	41	23,751
4	9	Mira	49,152	786,432	40	14,982
5	19	JUQUEEN	16,384	262,144	38	5,848



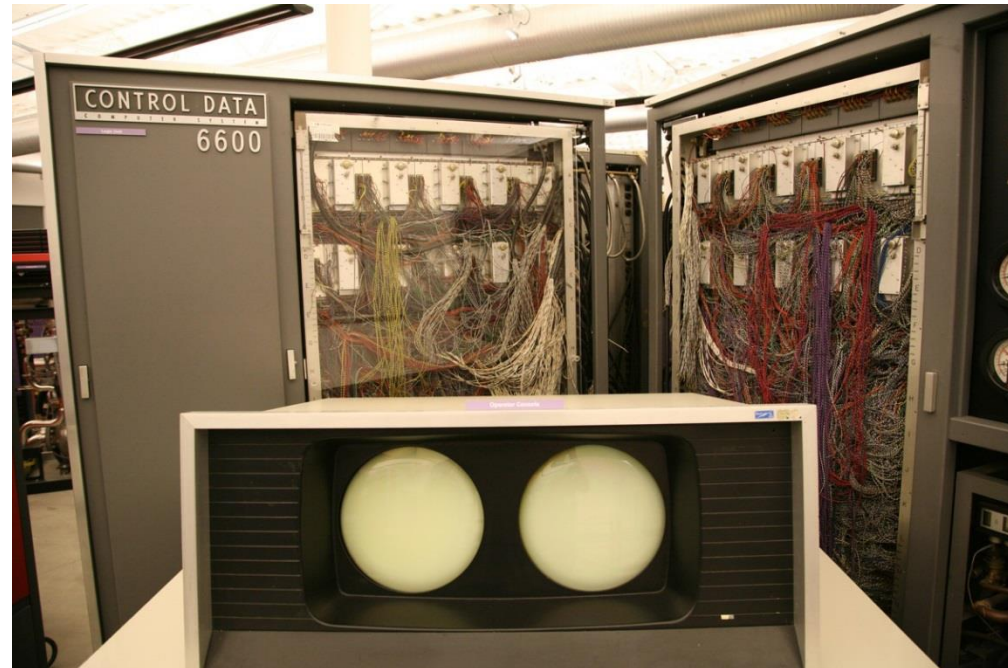
# Conclusion on Ranking



- Multiple ranking methods
  - Correspond to various needs
  - Highlight different architectures
- Where do the differences come from?
  - Various domains of applications
  - Depends on the target users
  - Impact on the design choices
  - Difference machine architectures
    - Processors, memory, network...
- How did we end up with such current lists?
  - A little bit of HPC/architecture history...

# HPC History

- CDC 6600
  - Built in 1964
  - Contain a single CPU
  - Cost: \$8 Million
  - Frequency: 40 MHz
  - Freon cooling
- Performance
  - 3 Mflops





# HPC History

## ■ Cray 1

- Built in 1976
- Designed by
  - Seymour Cray
- Cost: \$5 - \$8 million
- Frequency: 80 MHz
- Freon cooling

## ■ Performance

- ▶ **136 Mflops**



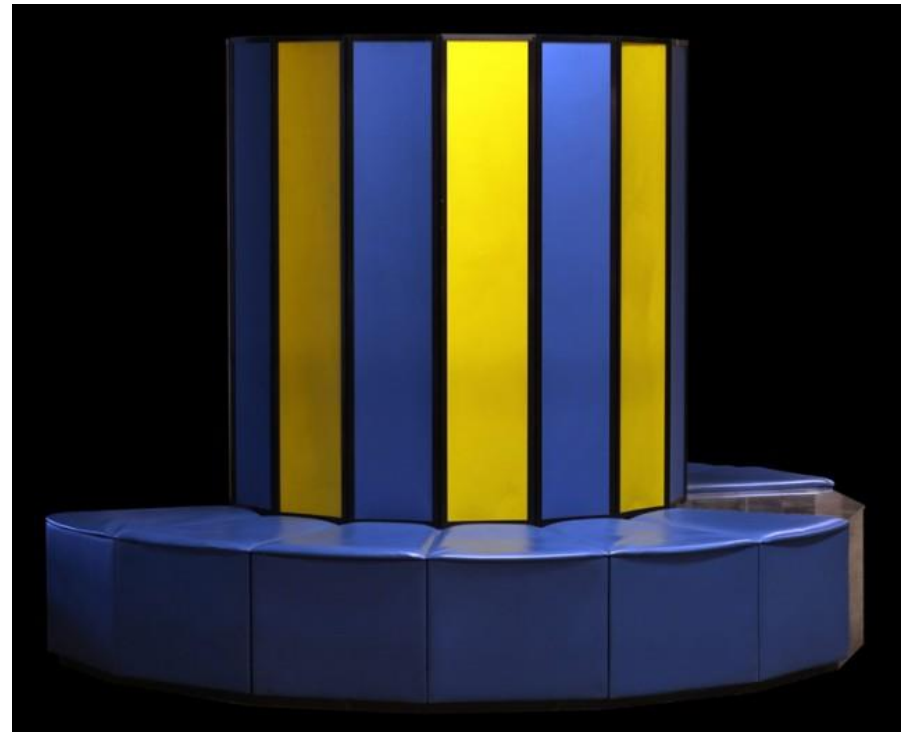
# HPC History

## ■ Cray XMP

- Built in 1982
- Up to 4 CPUs
- Frequency: 105Mhz
- Cost: \$15 million

## ■ Performance

- 200 Mflops per CPU
- **800 Mflops total!**



# HPC History

- **ASCI Red**
  - Build in 1997
  - 6,000 CPUs
  - Intel Pentium Pros
    - Regular processors
  - Frequency: 200Mhz
  - Cost: \$46 million
- **Performance**
  - **> 1 Tflops**
  - First one!



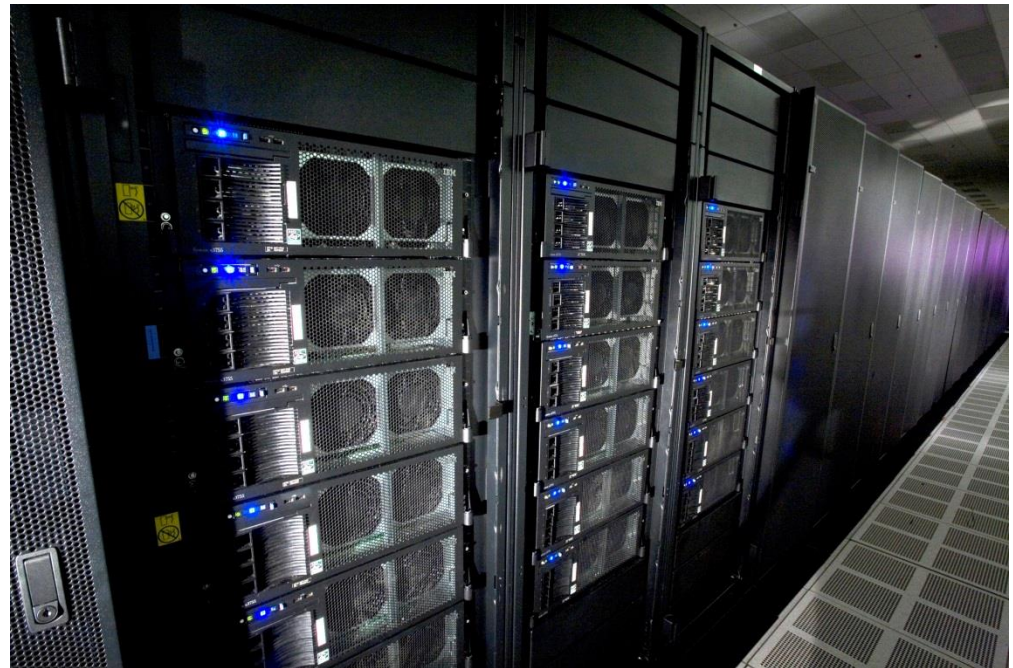
# HPC History

## ■ IBM Roadrunner

- Built in 2008
- Hybrid
  - ADM Opteron
  - IBM PowerPC
- Frequency:
  - 1.8GHz & 3.2GHz
- Cost: \$100 million

## ■ Performance

- **> 1 Pflops**
- First one!



# French Status

- Top500: 4<sup>th</sup> country w/ 20 systems
  - 4% of systems
  - 3.8% of global performance

Rank	Site	System	Cores	Rmax	Rpeak	Power
16	Total	Pangea (SGI)	220,800	5,283.1	6,712.3	4,150
50	Meteo France	Prolix2 (BULL)	72,000	2,168.0	2,534.4	830.4
51	Meteo France	Beaufix2 (BULL)	73,440	2,157.4	2,585.1	830.2
55	CEA	Tera-1000-1 (BULL)	70;272	1;871.0	2586.0	1,042
64	CINES	Occigen (BULL)	50,544	1,628.8	2,102.6	934.8

# French Ecosystem

- Teratec

- European pole of competence in high performance simulation
- Technology, research, dissemination
- Teaching & training



- Campus

- Group multiple companies & research labs
- Located in Bruyères-le-Châtel (close to CEA)
- Exascale Computing Research (Intel/CEA/UVSQ)
- InHP@CT seminars
  - <http://inhpact.hpcframework.com/>



- Forum organized each year

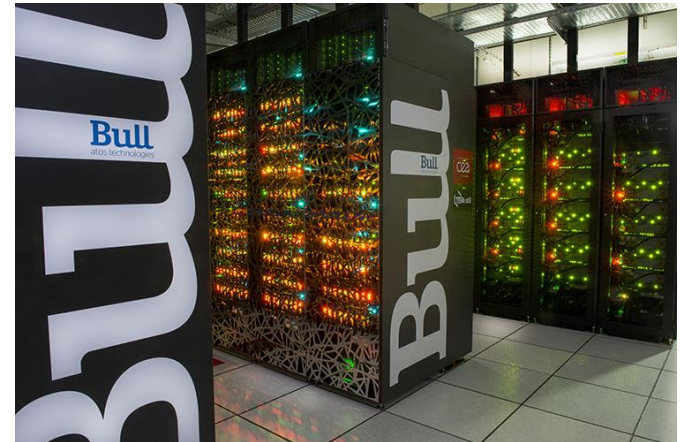
- June 19 & 20, 2018 @ Ecole Polytechnique
- Presentations & Exhibition



# French Ecosystem

- Main French Vendor:
  - Bull Atos
- Inside Top500
  - 7th vendors
  - 20 systems (4%)
  - 3.6% of global performance
- Co-design with CEA

**Bull**  
atos technologies



# French Vision: Bull & CEA

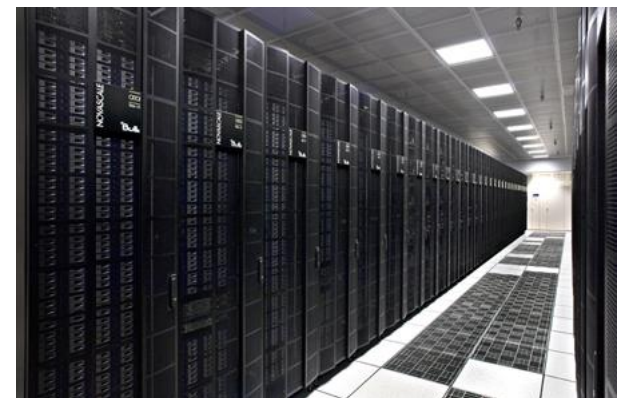


- Co-design between Atos Bull & CEA
- Multiple machines inside Top500 made by BULL and hosted by CEA
- HPC at CEA
  - Mainly CEA/DAM (Bruyères-le-Châtel)
  - Different product lines



# TERA

- Part of defense simulation program
- History
  - Program started in 1996
  - Predicted to set up 3 machines
- First machine: Tera 1 (HP/COMPAQ)
  - 2,560 cores (Alpha CPU, 1 GHz)
  - Quadrics interconnect
  - Linpack performance: 3.18 Tflop/s
  - Rank 4 in June 2002
- Second machine: Tera 10 (BULL)
  - 8,704 cores (Intel Itanium 2, 1.6GHz)
  - Quadrics interconnect
  - Linpack performance: 42.9 Tflop/s
  - Rank 5 in June 2006



# Current TERA Machine

- Tera 100 (Bull)
  - 140,000 cores (Intel Xeon Nehalem)
  - 4,300 compute nodes
  - IB QDR interconnect
  - Linpack performance: 1,050 TFlop/s
  - Rank 6 in November 2010
- Next steps
  - Tera 1000



# CCRT

- Research and Technology Computing Center

- *Centre de calcul pour la recherche et la technologie*

- French consortium

- Started in 2003

- Based on french academic & industry

- Goals

- Provide High Performance Computing resources for large scientific computations

- Foster a real synergy between research organizations, universities and industry

- Promote exchanges and scientific collaboration between partners.



# Current CCRT machine

- Cobalt (Bull)

- Total: 39,816 compute cores (Intel Xeon Broadwell)
- Node w/ dual-socket (28 cores per node)
- IB EDR interconnect
- Rank 63 in June 2016
- 1.299 Pflops



# PRACE

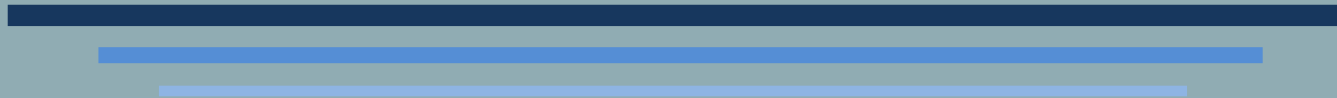
- Partnership for Advanced Computing in Europe
  - European Consortium
- 25 member countries
- 5 PRACE centers
  - BSC (Spain)
  - CINECA (Italy)
  - CSCS (Switzerland)
  - GCS (Germany)
  - GENCI (France)
- Currently
  - French machine Curie
  - Located in TGCC (Bruyères-le-Chatel)



# Current Status

- Computational power of supercomputer increases
  - How is it possible?
  - Is it specific to HPC?
- What are the main evolutions for the future?
- Need to understand the main parts to exploit such machines
  - A little bit of hardware architecture...

# HARDWARE CHALLENGES



Main challenges

# Hardware Challenges



- Conclusion from hardware presentation
  - Processors are building blocks of clusters
  - But one processor = cores + complex mechanisms
  - Clusters are made of many other components that are crucial for overall performance
- List of major components
  - Processors
  - Memory
  - Mother boards & nodes
  - ...
- What are the challenges related to these components?



# Processor Challenges



## ■ Main trends

- Increase number of cores
- Larger compute units
- General purpose or dedicated

## ■ Increase in the number of cores

- Per processor
- Per nodes

## ■ Evolution of compute units

- Less microarchitectural mechanisms
- Larger vector units

## ■ General purpose or dedicated

- Regular Intel Xeon multicore processors
- Intel Xeon Phi processors
- NVIDIA GPGPU

# Memory Subsystem Challenges



- Extended memory levels
- Evolution of caches
  - Still some private caches
  - May include scratchpad
  - Shared caches → mesh-based coherency
- New memory levels
  - High-Bandwidth Memory (HBM)
  - Non-volatile memory (NVM)

# Number of nodes



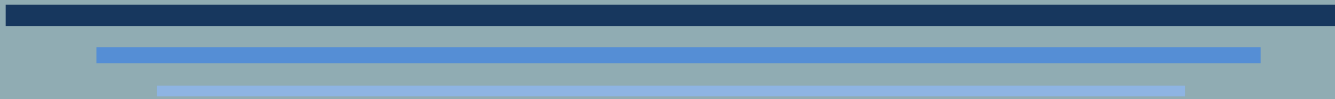
## ■ Main trend

- Include challenges from processors and memory
- Increase in number of nodes

## ■ Impacts

- Put the stress on network card (NIC)
  - Need to handle communication with more neighbors
- Imply new design for switches
  - Need to organize the network in specific topology (e.g., fat tree)

# PARALLEL COMPUTING



From cellphone to supercomputer

# Parallel Computing



- Tentative definition

- *Ability to exploit multiple compute units at the same time to solve a problem*

- Involve various domains

- Vehicule security, performance
- Chemistry (molecule interaction/reaction)
- Bio-informatics
- Energy
- Weather forecast

# Definitions

- Task
  - Work to do
- Thread
  - Implementation of a task: logical sequence of sequential actions result of the execution of a program
- Process
  - Instance of a program. A process consists of one or more threads that share a common address space. If a process has multiple threads, it is said to be multithreaded.
- Parallel computing
  - Parallel computing consists of splitting a program in several tasks that can be executed at the same time on independent computing resources to reduce execution time/compute larger problems/try multiple solutions

# What is Parallelism?



- Old idea to solve a problem more quickly and costly in time calculation
- One solution: to use several processing units (e.g. processors)
- Difficulty: organization of parallel tasks (parallel algorithmic):
  - solve the initial problem correctly: dependencies between tasks
  - the processing units must have constantly (useful) work to perform: distribution and (dynamic) load balancing

# Sequential programming



- ORDERED suite of instructions to run to resolve the initial problem
- Sequential semantics: any instruction can only begin when the previous one is completed and its result available
- FIXED ORDER in the execution of all instructions
  - Regardless of tasks dependencies



# Parallel programming

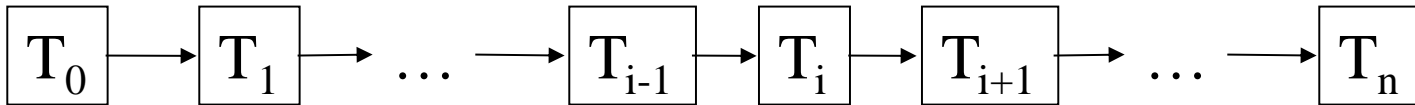


- Several execution flows (instructions + data)
- Several instructions executed simultaneously
- Multiple processors (or cores)
- Highlights the actual dependencies between statements:
  - the task T2 depends on the task T1 iff T2 needs the result of T1 (a correct result)
  - If T2 does not depend on T1 and T1 does not depend on T2, then T1 and T2 are **independent** tasks
  - → Two independent tasks can be executed in any order, or even simultaneously (e.g. in parallel)

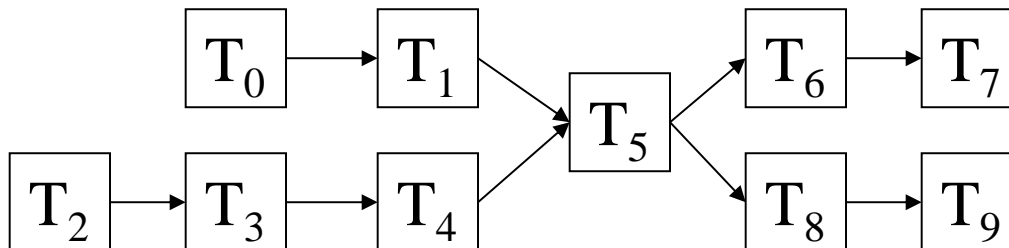
# Dependency graph

Dependency graph: highlights dependency relationships between tasks to complete an action

- $T_1 \rightarrow T_2$  means that  $T_2$  depends on  $T_1$ ;
- Depth of graph gives dependency
- Width of graph gives independency (parallelism) ;
- Sequential programming: dependency  $n$  and parallelism 1



- Ex parallel programming with 10 tasks: dependency 6 and parallelism 2



# Concurrency



- Parallel tasks are executed:
  - **simultaneously** :
  - or **alternately** (e.g. a task can be stopped to execute another one, then its execution is resumed) ;
  - or both;
- → Issue : tasks may access and modify common data (need **critical section**);
- → Solution : need to find mechanisms to ensure data coherency : **locks** and **mutex** (mutual exclusion)

# Communication



- Communication and synchronisation :
  - To ensure the consistency of a calculation, parallel tasks can have a « meeting point » before resuming their execution
- These « meeting points » are called **synchronisation** :
  - If the synchronisation concerns all parallel tasks
    - Global or collective synchronisation
  - If the tasks have different addressing space
    - Communications
- Communications
  - Global synchronisation => **global or collective communication**
  - synchronisation between 2 tasks => **point-to-point communication**

# Parallelism types



- What are the sources of parallelism in an application?
- 3 sources :
  - Control parallelism (tasks)
  - Flow parallelism (pipeline)
  - Data parallelism

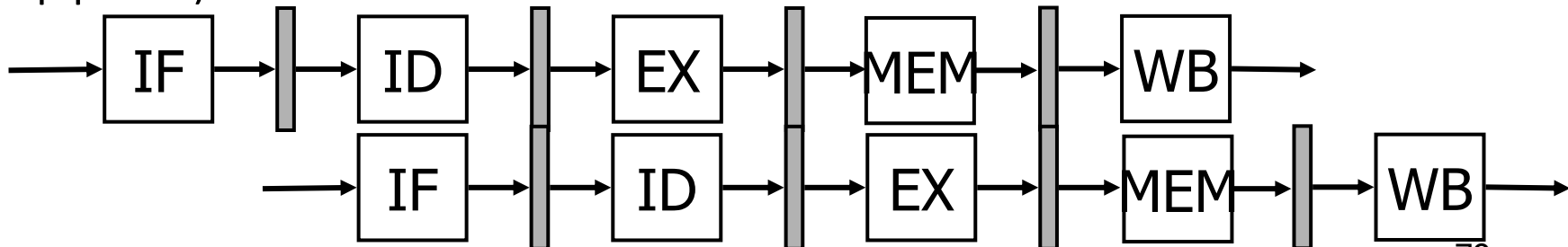
# Control parallelism



- Idea: « *Do several things simultaneously* »
- Simple constatation :
  - **An application is composed of tasks that can run simultaneously**
  - Example : the execution of a kitchen recipe with several cooks
- Exploitation of control parallelism consists in managing the dependencies between an application tasks in order to obtain an allocation of computation resources as optimal as possible
  - **Extraction of this parallelism from the dependency graph: width of the graph**
- In practice, the degree of parallelism is low and often complex to set up (ex : ILP dans les processeurs)
- Fits to the following parallel programming models:
  - **MIMD (Multiple Instruction Multiple Data)**
  - **MPMD (Multiple Program Multiple Data)** : Processors run different programs with their own data

# Flow parallelism

- Idea : " Chainwork "
- Work like a Pipeline
  - A series of operations are applied on a data stream, generally composed of similar data
  - The computing resources are associated with the actions and linked so that the results of the actions performed at time  $T$  are passed at time  $T + 1$  to the next computing resource
  - Example : vectorial machine
- Degree of parallelism depends on pipeline depth (number of stages)
- Working on vectorial data:
  - Vectors must be long enough to minimise on the cost of filling the pipeline
- Data flow must be as contiguous as possible (every stop empties the pipeline)



# Data parallelism



- Idea : « *Repeat actions on similar data* »
- Share the data and not the tasks
  - example : several kitchen clerks peeling a bunch of potatoes
- Degree of parallelism potentially very high, depending on the amount of data
- Fits the parallel programming model: **SPMD (Single Program Multiple Data)**
  - All processors run the same program with their own data
  - Viable for a large number of data
  - Very effective for many intensive computing algorithms (scientific computation, animated films): afterwards, we will focus on this programming model



# Parallel programming paradigm



- Independently of the hardware architectures of the machines, two parallel programming models emerge:
  - distributed memory programming model
  - shared memory programming model
- In theory, each model can be implemented on any type of architecture with collateral effects of varying performance

# distributed memory prog. model



- Conditions :
  - the parallel tasks work on separate memories, invisible from one another
  - the data (tables, etc ...) are split (called distributed data) on the various parallel tasks
- Consequence: to ensure the accuracy of the final result, inter-task communication becomes mandatory
- This is called programming by **message passing**
- **Fits the SPMD model**

# distributed memory prog. model



- Implementation on distributed-memory architecture
  - Easy because it is the same principle
  - on a distributed memory machine, processes that have their own address spaces must send messages over the network to exchange information
- Implementation on shared-memory architecture
  - Is not more difficult:
    - through multiple processes using the shared memory segments for communications
    - Through a multithreaded process using its memory to exchange information between threads

# distributed memory prog. model



- Encapsulation of message exchanges, implemented by libraries, such as:
  - PVM (Parallel Virtual Machine) : one of the first portable library of message passing
  - MPI (Message Passing Interface) : the actual de-facto standard, coming from the collaborations of industrial partners and universities
    - This cours is dedicated to this programming interface
  - RMA: Remote Memory Access

# shared memory prog. model



- Condition :
  - **Parallel tasks are sharing the same address space**
- Consequence :
  - Need to properly handle concurrent access to the same data (critical sections)
- Despite the apparent simplicity of programming, performance can be quickly degraded because:
  - Critical sections are not parallel (by definition)
  - Data locality and memory hierarchy are transparent to the user

# shared memory prog. model



- Implementation on distributed-memory architecture
  - difficult : how to « see » the whole memory ?
  - DSM (Distributed Shared Memory) : software mechanism allowing to simulate a unified memory on top of a physically distributed memory
  - May be very inefficient because the cost of distant accesses is hidden
- Implementation on shared-memory architecture
  - Easy because it is the same principle
  - In a multithreaded process, every thread can access the process memory (which can address the whole memory on the node)

# shared memory prog. model



- API POSIX pthread :
  - Normalised manipulation (POSIX) of threads in a process
  - Fit the MPMD model
- OpenMP :
  - Thread manipulation by directives
- At the present time, tools implementing the shared memory model offer a great deal of thought so that applications can make the best use of multicore processors
  - TBB, Cilk++, ABB, ...

# Hybrid memory prog. model



- Combine both distributed and shared memory programming models
- Explicit hybrid programming:
  - Explicitly use multiple programming models for distributed and shared memory paradigm
  - MPI+X (X= OpenMP, Pthreads, TBB, ...)
- Implicit programming model
  - Programming models working on a unified view of the memory
    - PGAS: Partitioned Global Address Space
  - Software to simulate a unified memory
    - DSM: Distributed Shared Memory



# Heterogeneous prog. model



---

---

---

- **Programmation on accelerator(s) attached to the node**
  - GPUs, Intel Knights Corner/Landing, Specialized cards
- **Specific programming language**
  - Ex: CUDA for Nvidia GPUs, Intel LEO for KNL
- **Can also use generic programming models**
  - C, C++, etc... for Intel KNL
  - OpenMP accelerator directives for Intel KNL, Nvidia...

# Parallelism benefit in cluster



- Distributed memory programming model
  - Allows to work on all nodes of a supercomputer
  - Parallelism potential benefits: tens (or hundreds) of thousands nodes
- Shared memory programming model
  - Allows to work on all cores/hyperthreads of a node
  - Parallelism potential benefits: up to 288 hyperthreads on an Intel KNL
- Vectorization
  - Allows to use all computational resources in one instruction
  - Parallelism potential benefits (depends on architecture vector length): up to 16 doubles / 32 simples

# OUTLINE OF THE COURSE



MPI: Message Passing Interface

# MPI: Message Passing Interface



- Lecture 1:
  - MPI basic principles
  - Point-to-point communications
- Lecture 2:
  - MPI collective communications
- Lecture 3:
  - Manipulation of MPI structures
- Lecture 4:
  - Advanced collective communications
- Lecture 5:
  - MPI Remote Memory Access
- Lecture 6:
  - MPI I/O
- Lecture 7:
  - MPI Forum and MPI Future Features
  - Introduction to High Speed Network