HPC Seminar

Introduction to HPC

Sergey Yakubov - Maxwell Team - DESY IT
Hamburg, 09.04.2018

Introduction

- What is this all about?
- For whom?
- From whom?
  - DESY IT (Y.Kemp, J.Reppin, F.Schlünzen, S.Sternberger, S.Yakubov, …) + invited experts
- Format
  - Every second Monday 15-00 – 17-00 (flexible end)
  - Lecture + hands-on (flexible shares)
  - Agenda (flexible)
  - Not individual seminars, more like building blocks
- Prerequisites
  - Basic Linux knowledge, some programming knowledge (C/C++, Fortran, Python)
  - Bring you own laptop
  - Registration – not necessary, subscribe to the mailing list
- Contact hpc-seminar@desy.de, information : https://indico.desy.de/indico/category/619/

Motivation

Well, since you’ve already came here …

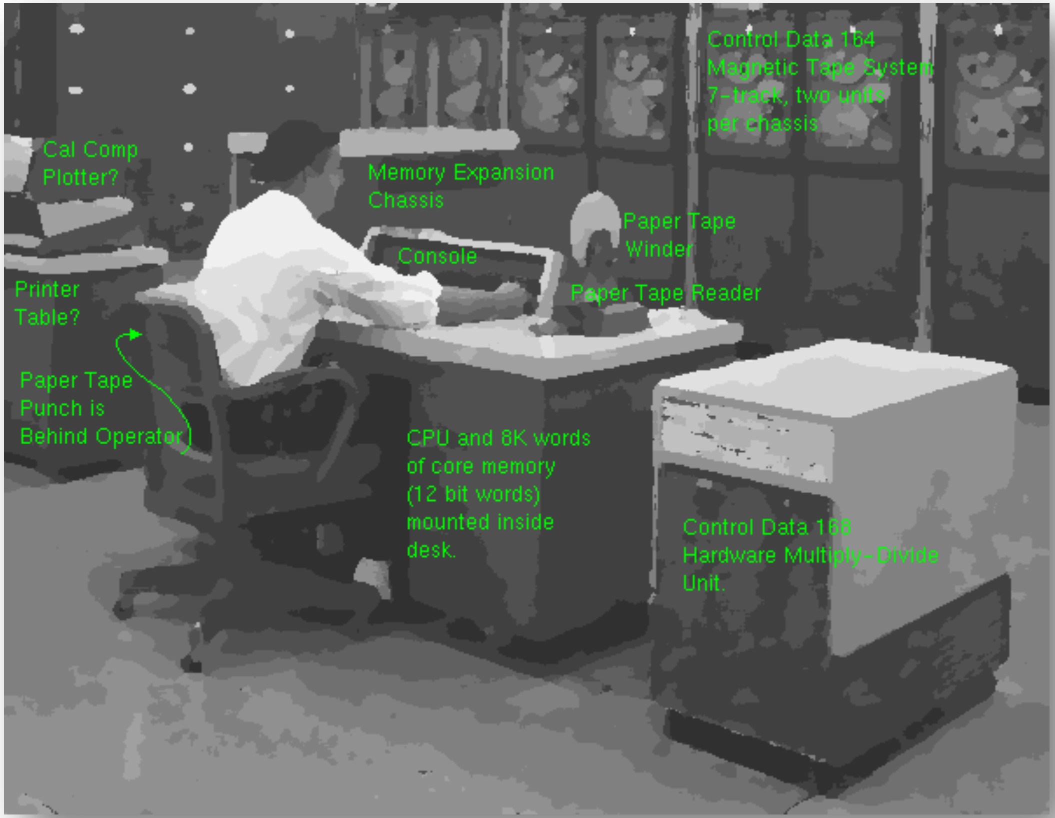
Agenda

- HPC hardware
- HPC architectures
- Parallel computing and performance
- Communications
- Software development for HPC

HPC Hardware

Brief History

- Until mid 90's only special (expensive) hardware machines from CRAY, CDC, NEC ...



cs.uiowa.edu

HPC Hardware

Brief History

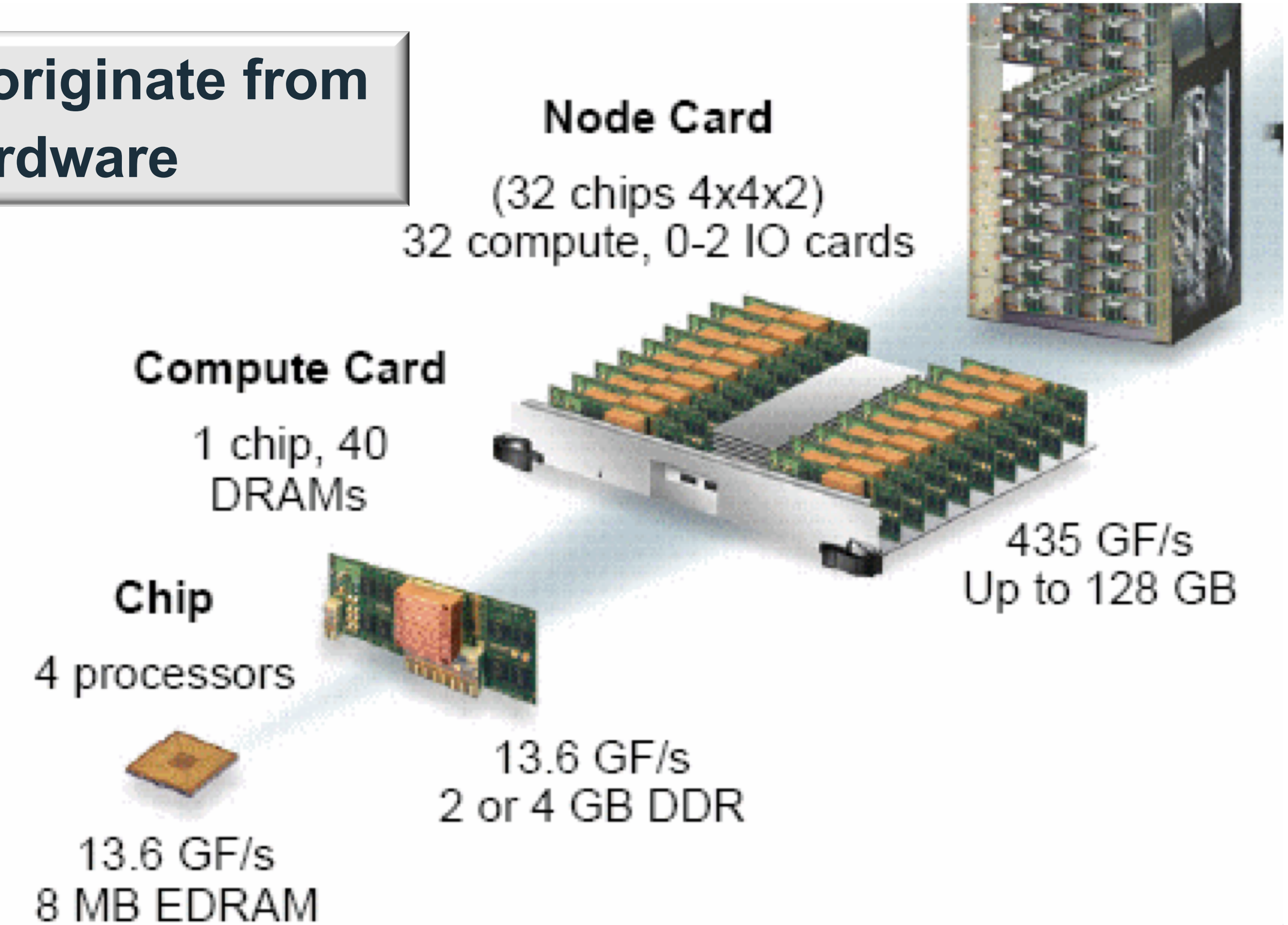
- Until mid 90‘s only special (expensive) hardware machines from CRAY, CDC, NEC ...
- Since 90s strong trend toward commodity hardware
- Today‘s computers use many of-the-shelf CPUs in parallel
  - State-of-the-art microprocessors
  - advanced network
  - cost efficient
  - present focus on energy
- Very few vector machines survived

Efficient design and use of parallel machines is of the essence

HPC Hardware

State-of-the-art

Computers originate from standard hardware

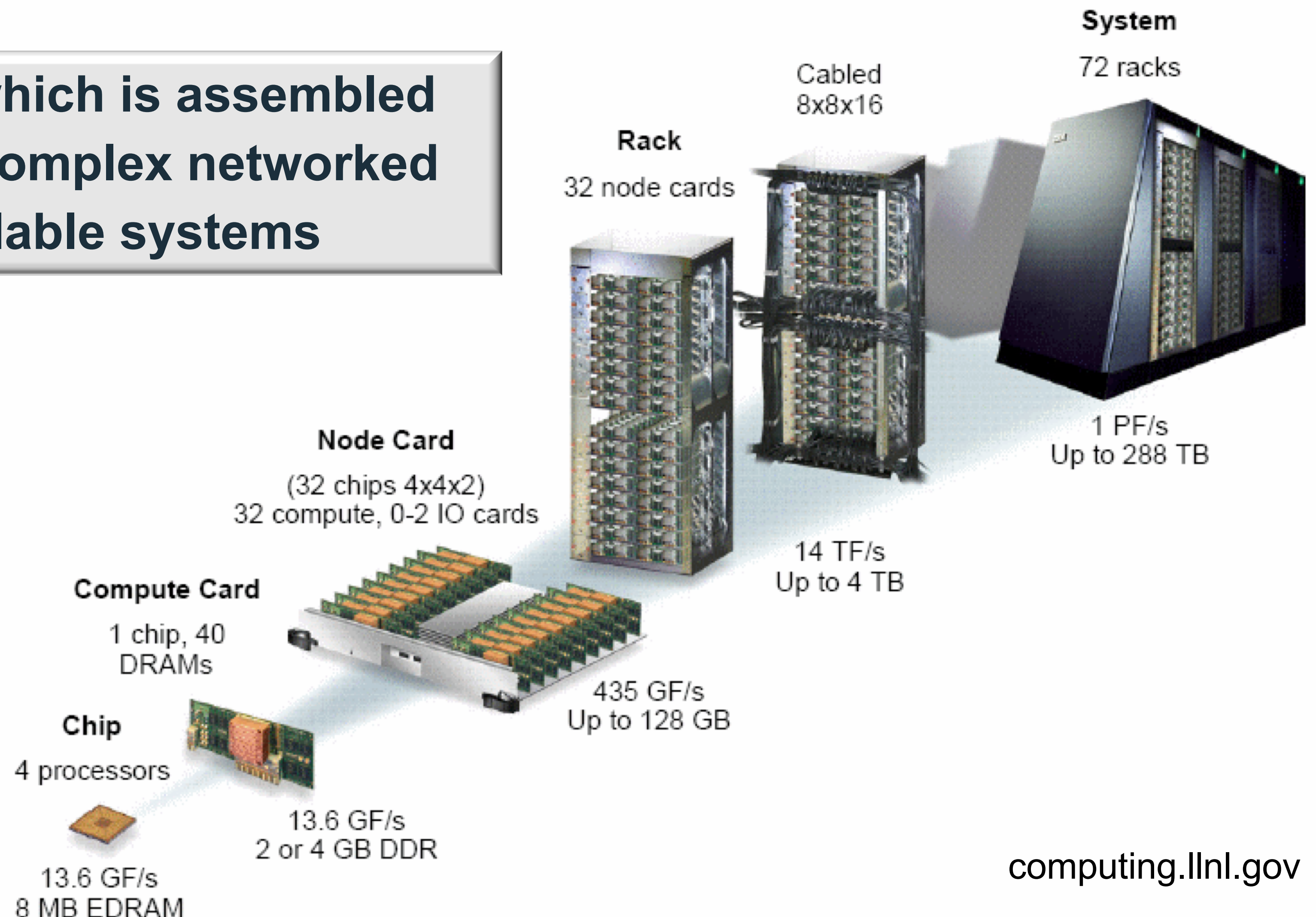


computing.llnl.gov

HPC Hardware

State-of-the-art

... which is assembled to complex networked scalable systems



computing.llnl.gov

HPC Hardware

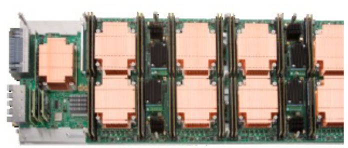
State-of-the-art

Another example

Compute node
2 sockets

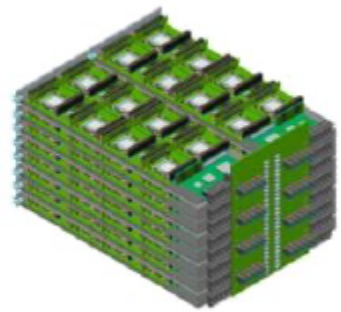
24 cores

Compute blade
4 nodes



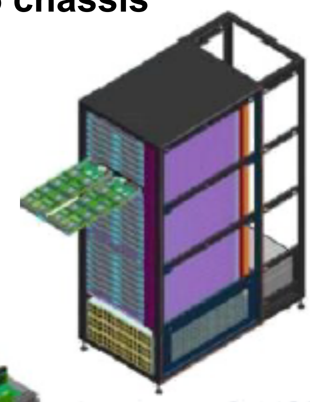
96 cores

Chassis
16 blades



1.536 cores
(64 nodes)

Group
6 chassis



9.216 cores
(384 nodes)

System

Up to 10s thousands of nodes

Cray XC40

cray.com

HPC Hardware

GPU

- GPU (Graphics Processing Unit)
  - Typically used for image processing and graphics computing (3D rendering,…)
  - Has thousands of cores to process parallel tasks
  - Has very fast memory (about 9x faster than RAM)
  - Low power consumption

Idea – use this cards for general purpose computations !

HPC Hardware

GPU

- GPGPU - advantages
  - Cheap (relatively)
  - Performance/power ratio is very high
  - Can be combined with computations on CPU

- GPGPU – disadvantages
  - (Efficient) programming is not so easy
  - Limited amount of supported languages (this can change)
  - Limited memory (32 GB max)

HPC Hardware

Networks

Technologies/standards

- Gigabit Ethernet (1 Gbit/s)
    - Slow, but enough for many applications
- InfiniBand (up to 100 Gbit/s)
    - Most used standard (26% of top500)
- 10,40,100 Gbit Ethernet
    - Alternative to InfiniBand
- Others (OmniPath, Cray Aries Interconnect, etc.)

HPC Hardware

Networks

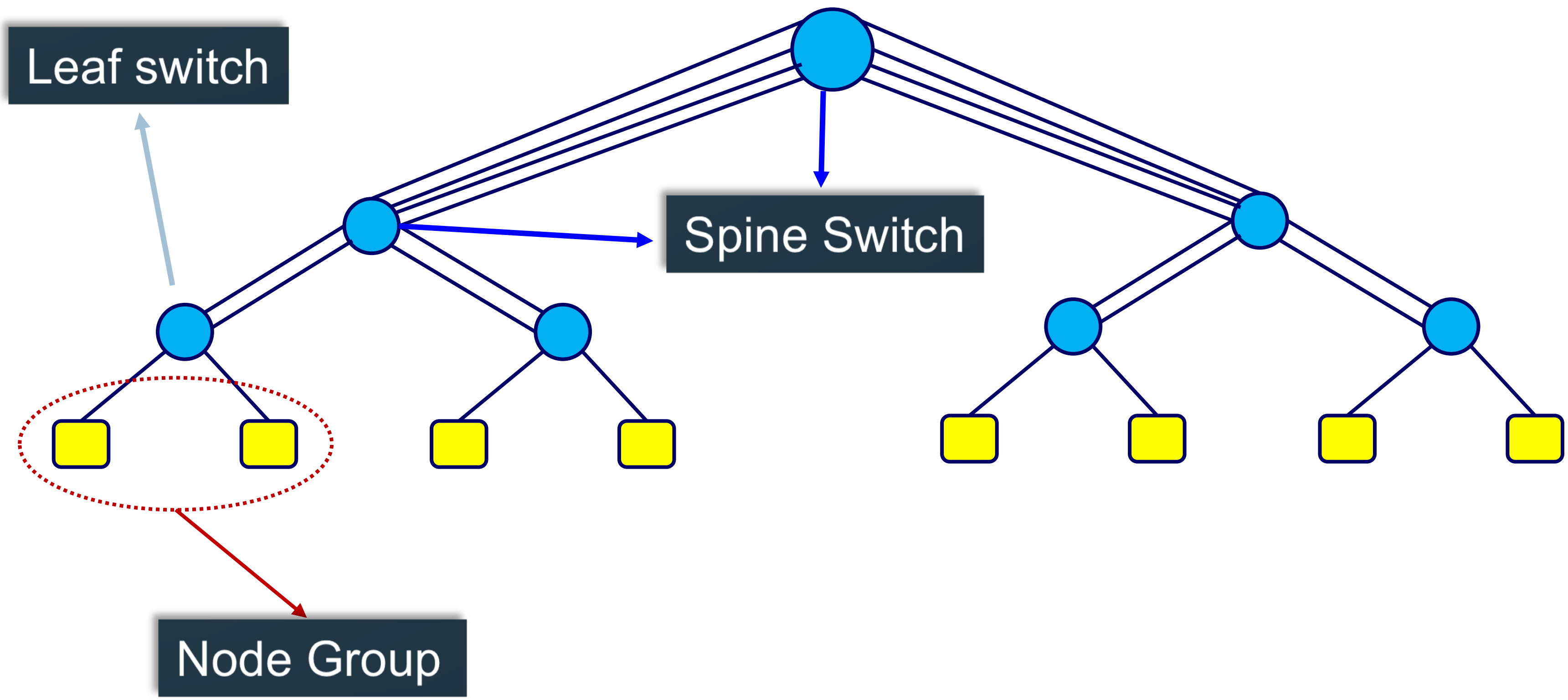
Topology

- Defines how computational nodes and other devices (IO, servers, etc.) are connected with each other
- Two basic types of connections
  - Connections via switches (Fat tree networks)
  - Direct connections between nodes (mesh, hypercube, etc)
- Routing algorithms define the path of the data to be communicated
  - Deterministic routing
  - Adaptive routing

HPC Hardware

Networks

Fat-tree topology

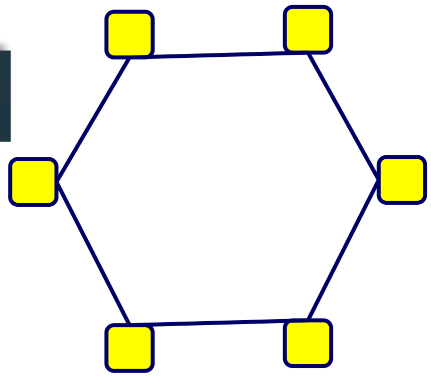


HPC Hardware

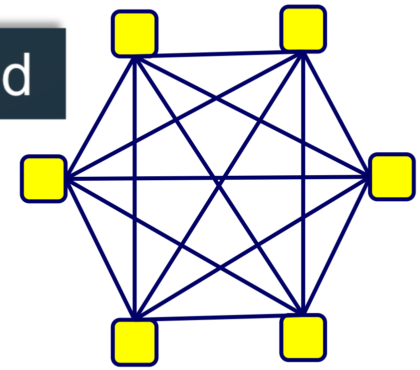
Networks

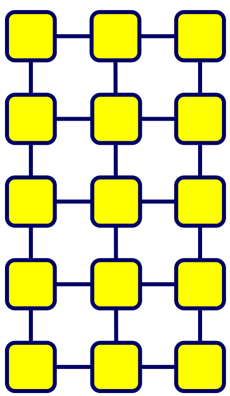
Direct connections

Ring

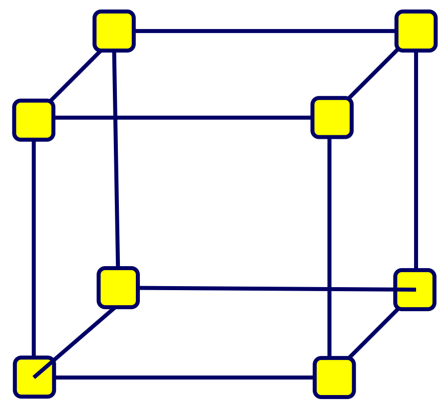


Fully connected





Mesh



Hypercube

HPC Hardware

Storage

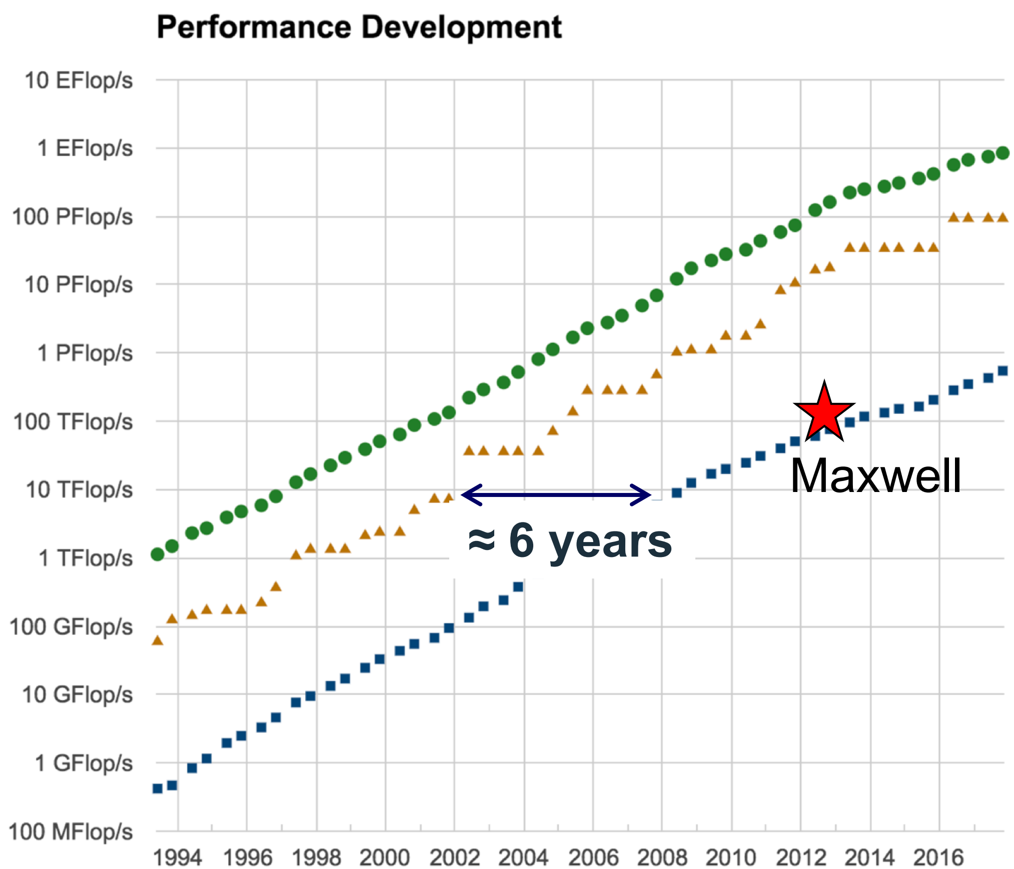
- Usually a parallel filesystem is used for data storage
  - Single file is distributed across multiple servers
  - Concurrent high-speed access
  - High performance

- There are several widely used file systems
  - GPFS (IBM Spectrum Scale) – commercial, high availability, disaster recovery, security
  - Lustre – open source, high availability
  - BeeGFS – open source, additional options for commercial users

HPC Hardware

Top 500

Computing Power (in FLoating-Point Operations per Second)



sum of 500 best computers

best system

average of 500 best computers

M(ega)=10^6

G(iga) =10^9

T(era) =10^{12}

P(eta) =10^{15}

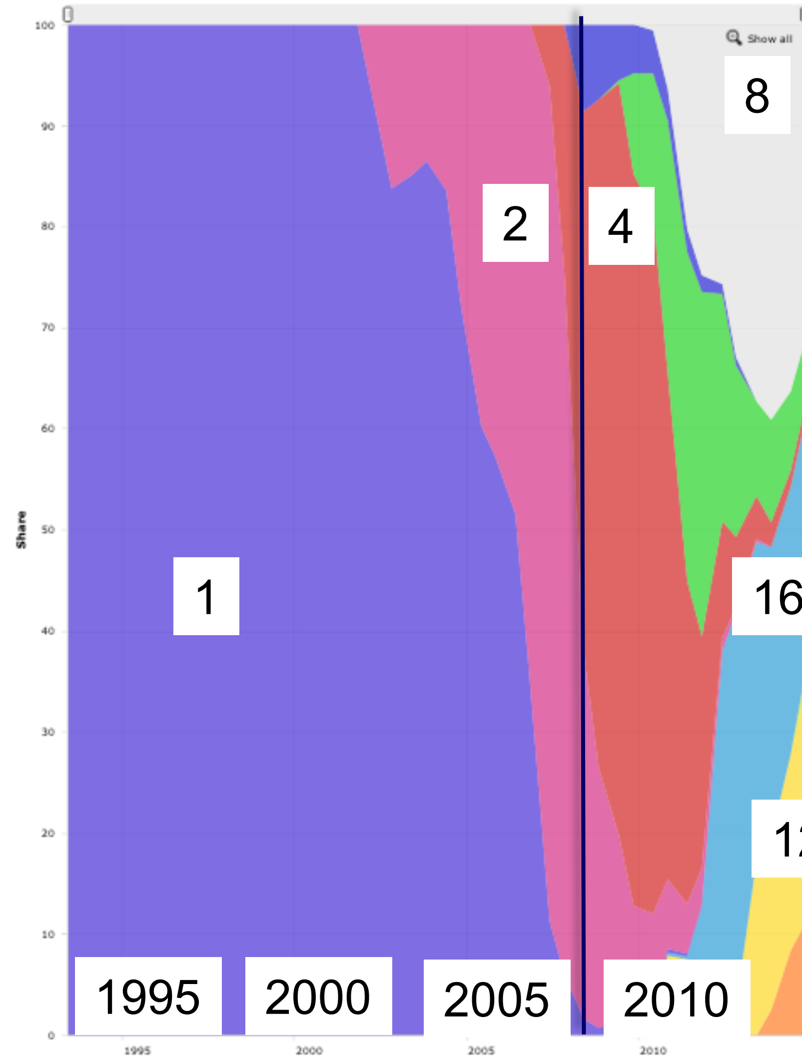
E(xa) =10^{18}

common inclination reveals a **factor of 10 every 4 years**

HPC Hardware

Top 500

- Cores per CPU Shares
  - Trend to many core systems
  - Driven by energy efficiency & cooling aspects
- Background
  - Faster clock speed = faster CPU
  - Over-clocking by increasing voltage (stand. about 1V / core)
  - Power consumption scales with V^3 , gains are sub-linearly in V
  - Reducing V by 20% cuts power by 50% and allows for 2 cores



HPC Hardware

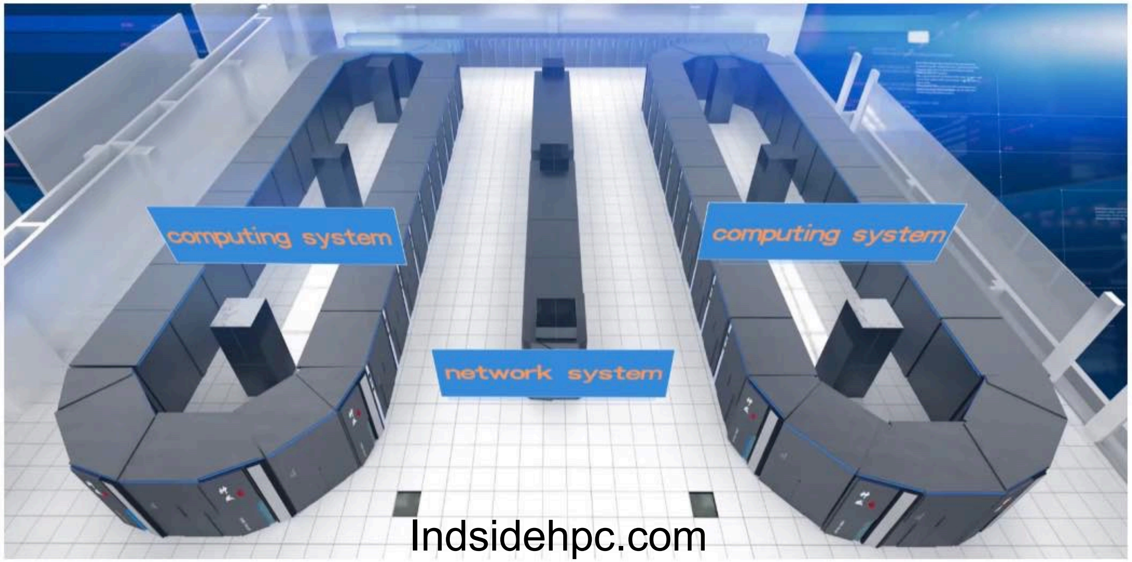
Top 500

- Operating Systems
  - almost only operated under Linux
- Architectures
  - dominated by distributed memory systems (clusters or MPP),
    virtually no shared memory systems
  - since 13 years dominated by multi-core CPUs, with
    8 cores (10%), 10 cores, (10%), 12 cores (27%), 14 cores (21%) 16 cores (21 %) per socket
- Hosting Countries (performance share)
  - dominated by the China (40%), US (29%), Japan (7%),
    Germany (4%), France (4%), UK (3%)
- Recent trend to accelerators & GPU hardware

HPC Hardware

Big machines

#1 Top500: 15 MW Power, 10 mln cores, 93 Peta Flops



#3 Top 500: 2.2 MW Power, 361,760 cores, 19 Peta Flops

HPC Architectures

Basic classification

Flynn Taxonomy (Flynn,1966)

Non-parallel single processor program execution

SISD Single Instruction Single Data	**MISD** Muiltple Instruction Single Data
SIMD Single Instruction Multiple Data	**MIMD** Multiple Instruction Multiple Data

Vector computers, GPU

Most of the modern HPC machines

HPC Architectures

Basic classification

Memory organization

MIMD
Multiple Instruction Multiple Data

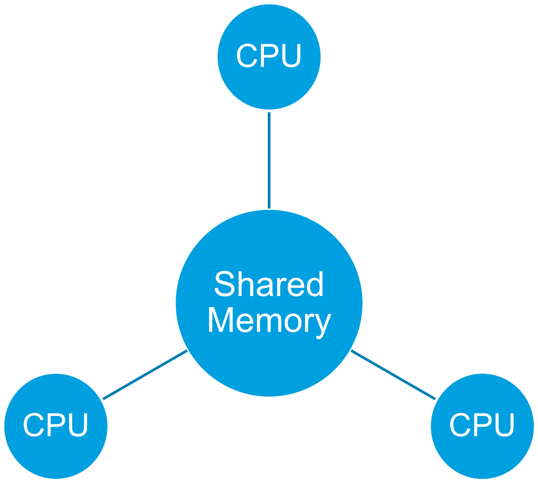
→ **Shared memory**

→ **Distributed memory**

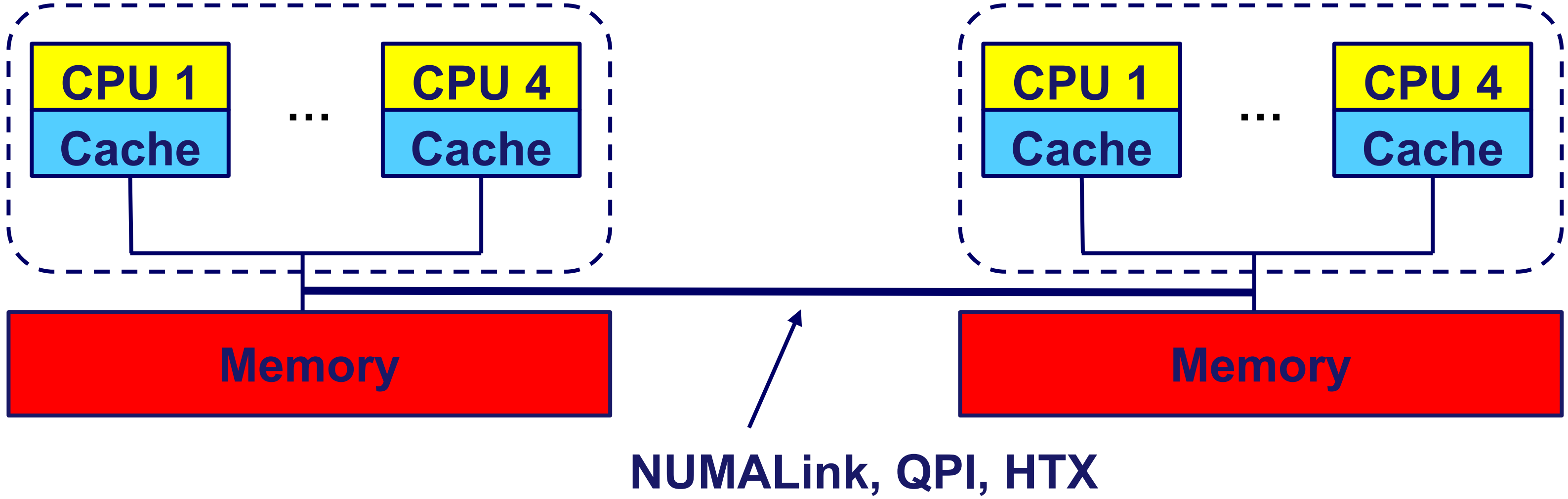
→ **Hybrid**

HPC Architectures

Shared memory



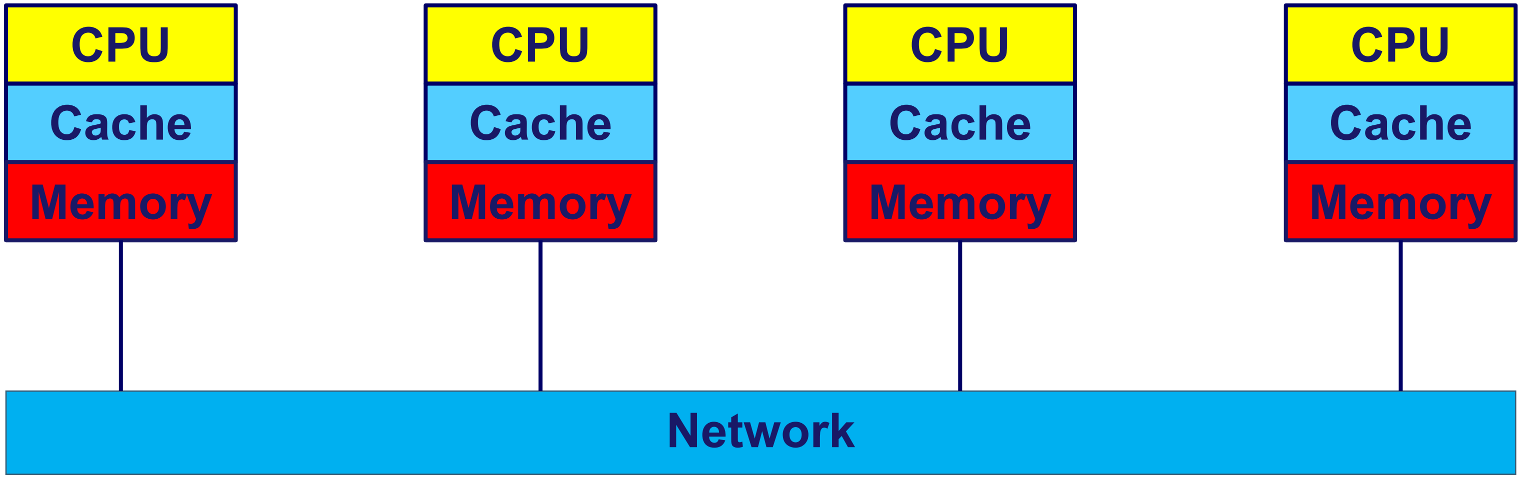
- A shared memory machine provides
  - A single shared memory address space for all CPUs
  - The address space is transparent to the user
- Two models of memory access
  - Uniform Memory Access (UMA)
  - Non-uniform Memory Access (NUMA)/cache-coherent NUMA (ccNUMA)



HPC Architectures

Distributed memory

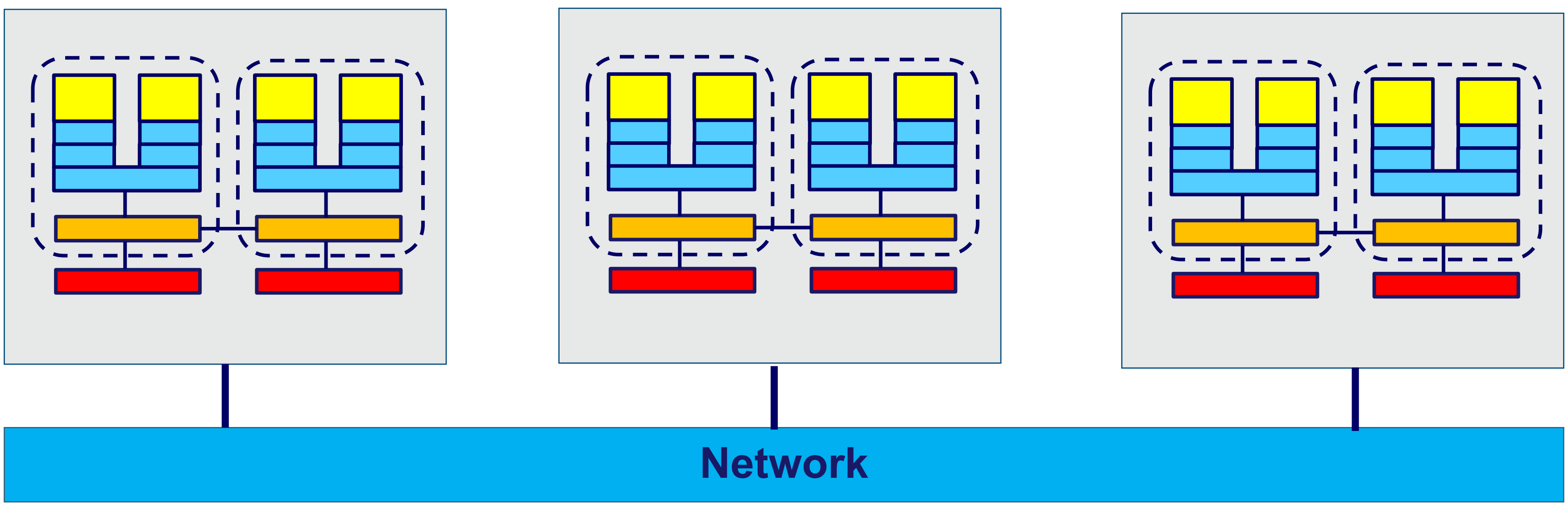
- „Pure“ distributed memory cluster
  - Each CPU has its own cache and memory, no direct access from other CPUs
  - Internode communications via network
- Ancestor of modern systems (90s)
  - A bunch of Ethernet connected desktops



HPC Architectures

Distributed memory

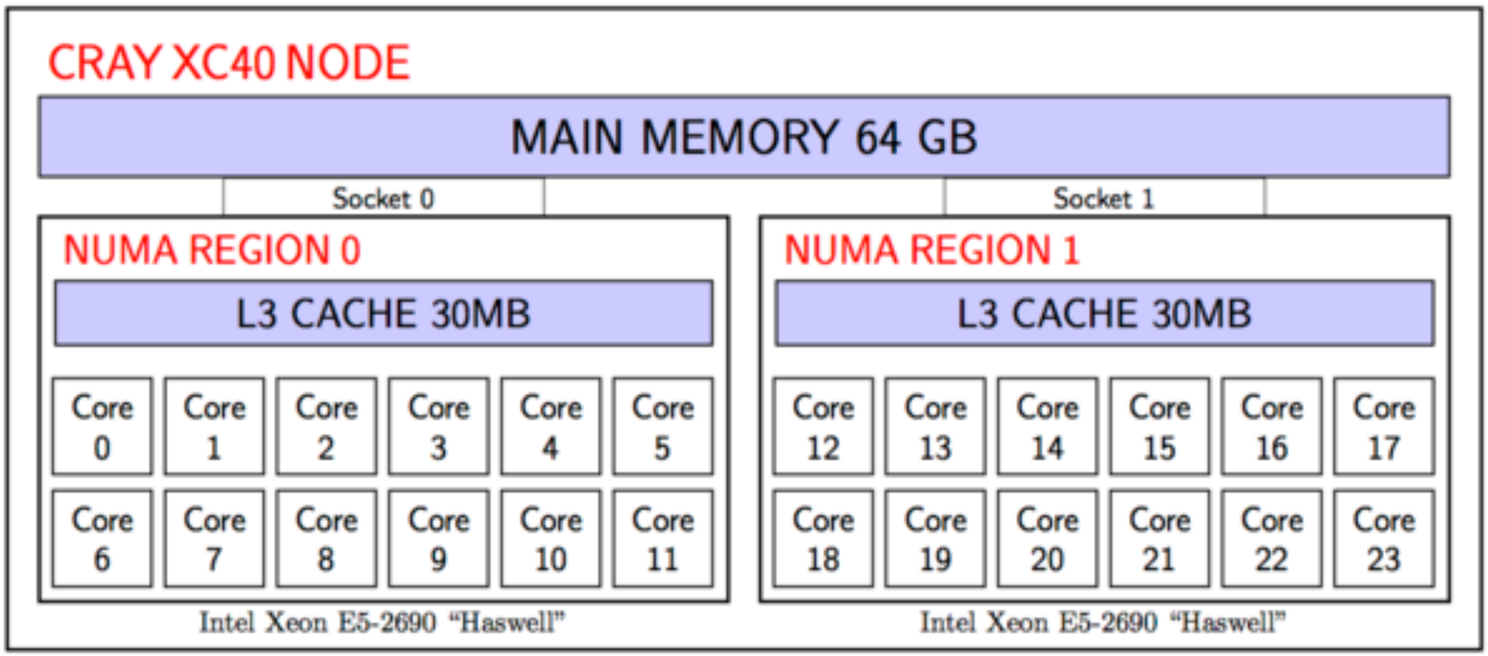
- Hybrid parallel system
  - State-of-the art compute nodes have several sockets, each has up to 16+ cores – shared memory
  - Nodes are connected via network – distributed memory
- Most of the modern HPC clusters are hybrid
- GPUs add another level of hierarchy



HPC Architectures

Distributed memory

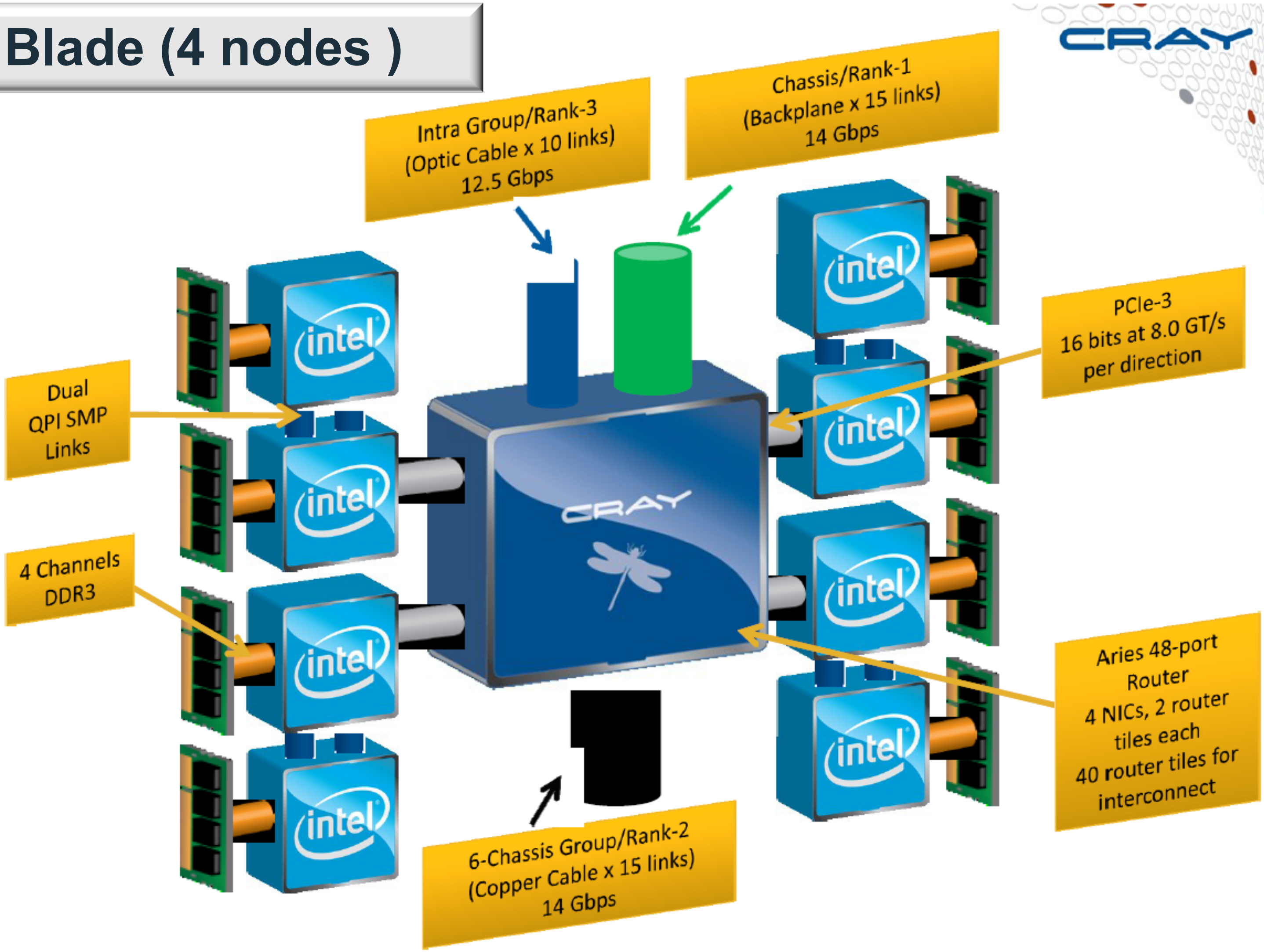
Our example - Cray XC40

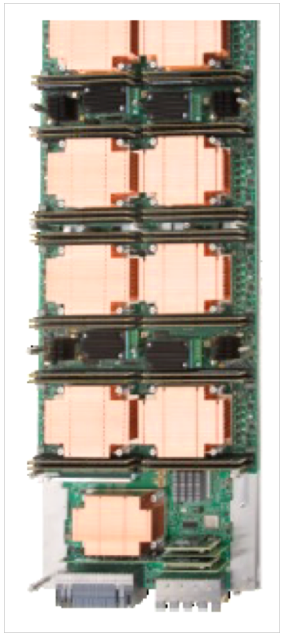


HPC Architectures

Distributed memory

Blade (4 nodes)

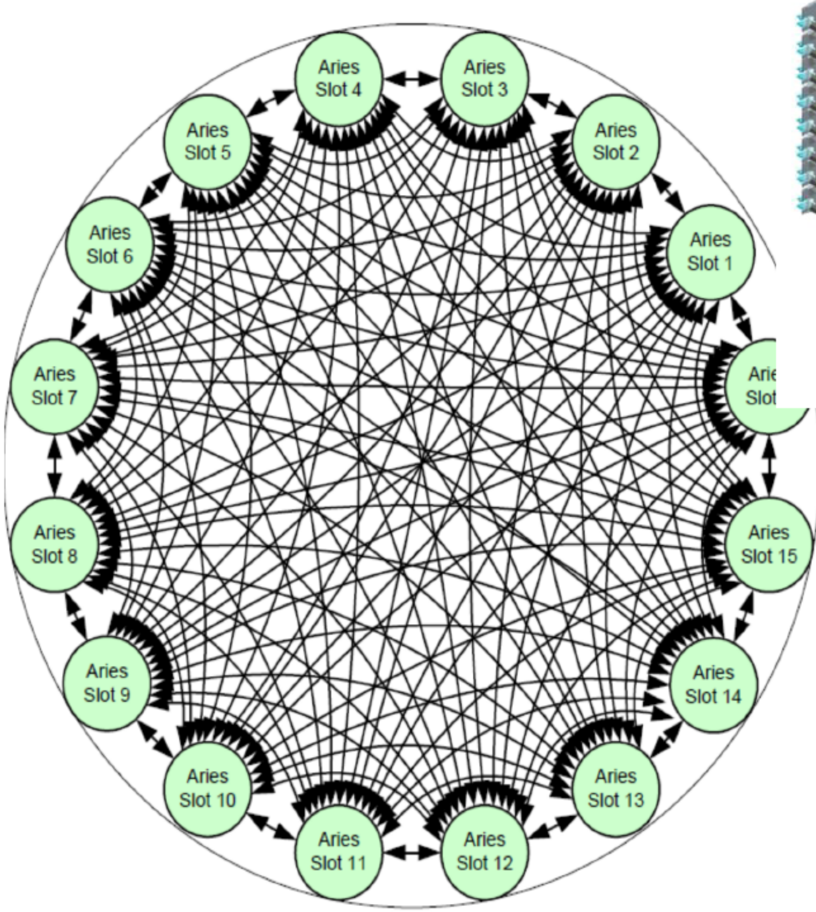


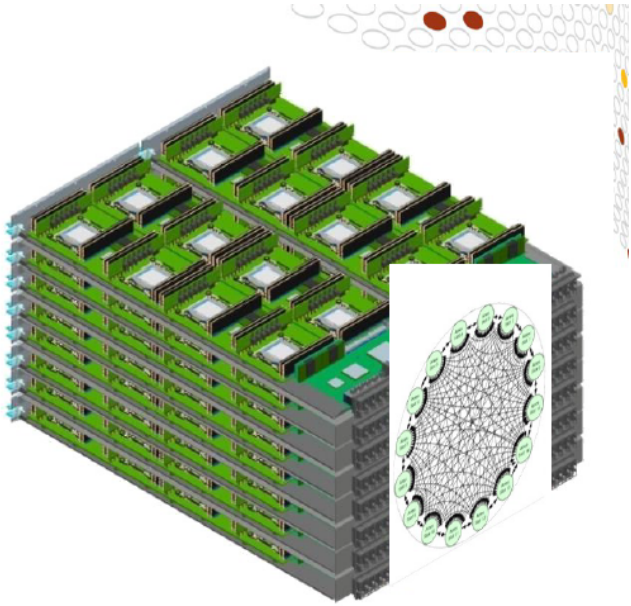


HPC Architectures

Distributed memorv

Chasis (16 blades, 64 nodes)



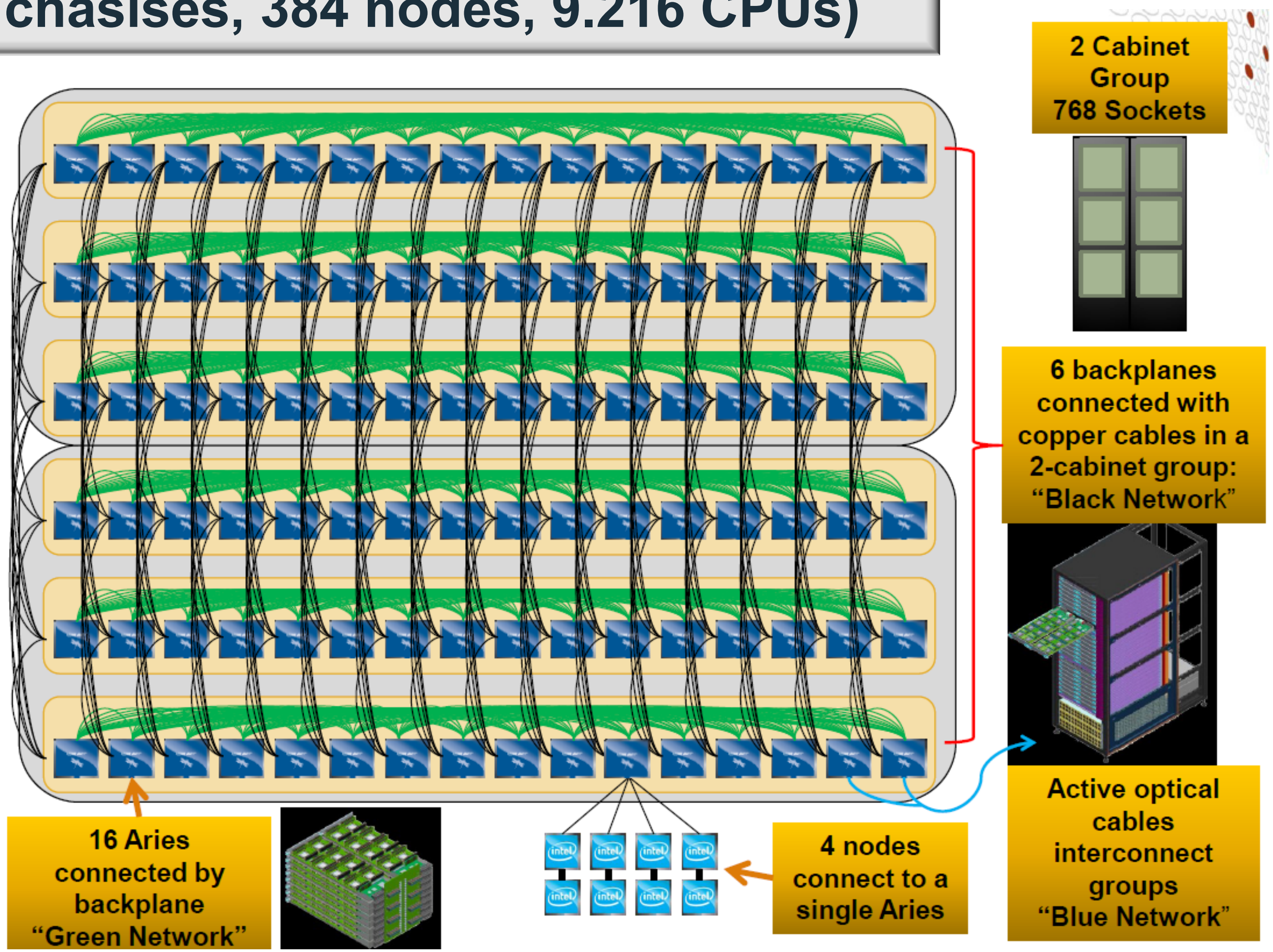


- **Chassis with 16 compute blades**
- **128 Sockets**
- **Inter-Aries communication over backplane**
- **Per-Packet adaptive Routing**

HPC Architectures

Distributed memory

Group (6 chasises, 384 nodes, 9.216 CPUs)

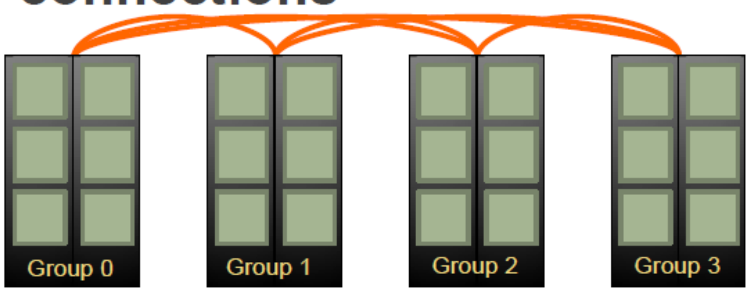


HPC Architectures

Distributed memory

System (4 groups in this example)

- An all-to-all pattern is wired between the groups using optical cables (blue network)
- Up to 240 ports are available per 2-cabinet group
- The global bandwidth can be tuned by varying the number of optical cables in the group-to-group connections



Example: An 4-group system is interconnected with 6 optical "bundles". The "bundles" can be configured between 20 and 80 cables wide

Parallel computing

Programming concepts

- **(Parallel) job/work**
  - A task to be solved on parallel machine
  - The job should be somehow (better uniformly) distributed over the available computational resources
- **Parallel program**
  - Computer program to solve a problem in parallel
- **(Parallel) process**
  - Part of a parallel program running on an assigned part of a parallel machine, usually within distributed memory architecture
- **(Parallel) thread**
  - Same as parallel process, but usually within shared memory architecture

Parallel computing

Programming concepts

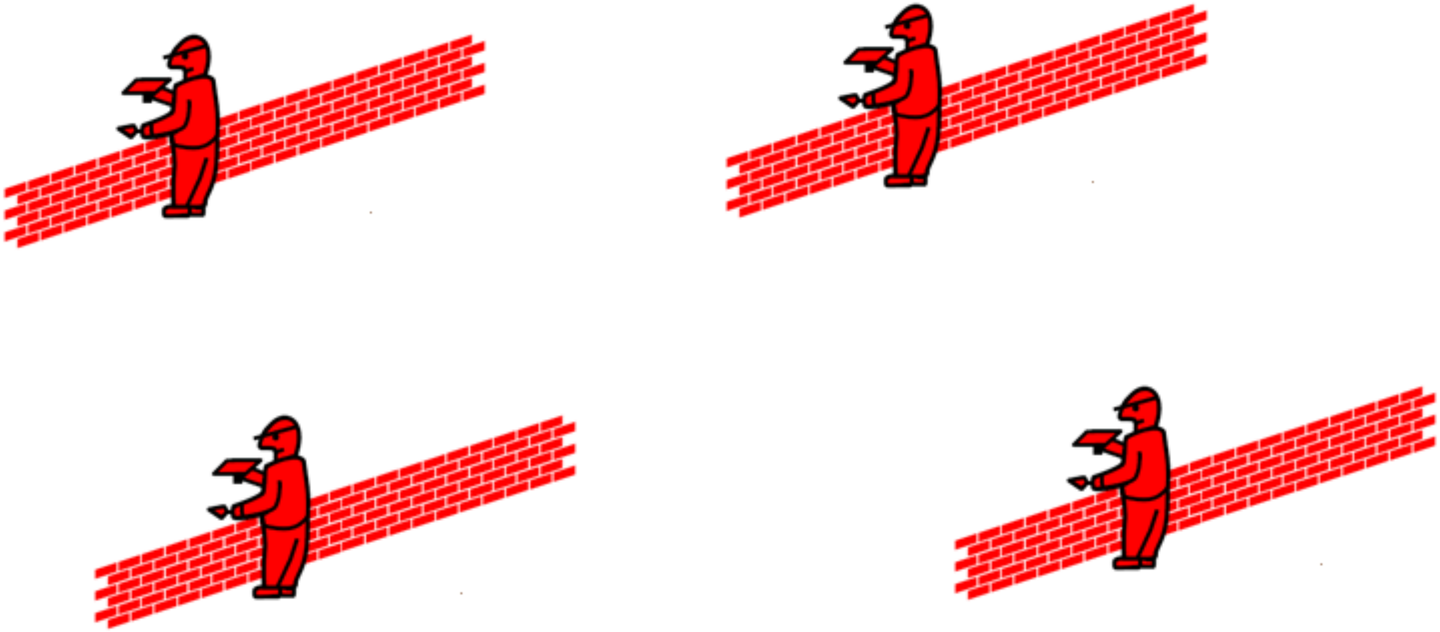
- **Single Program Multiple Data**
  - Most of the parallel programs
  - Single executable, parallelization is implemented within the application (different approaches)

- **Multiple Program Multiple Data**
  - Different executables are started in parallel on different CPUs
  - The applications can communicate with each other
  - Not used that much

Parallel Performance

Embarrassingly parallel

No communications, no resource sharing

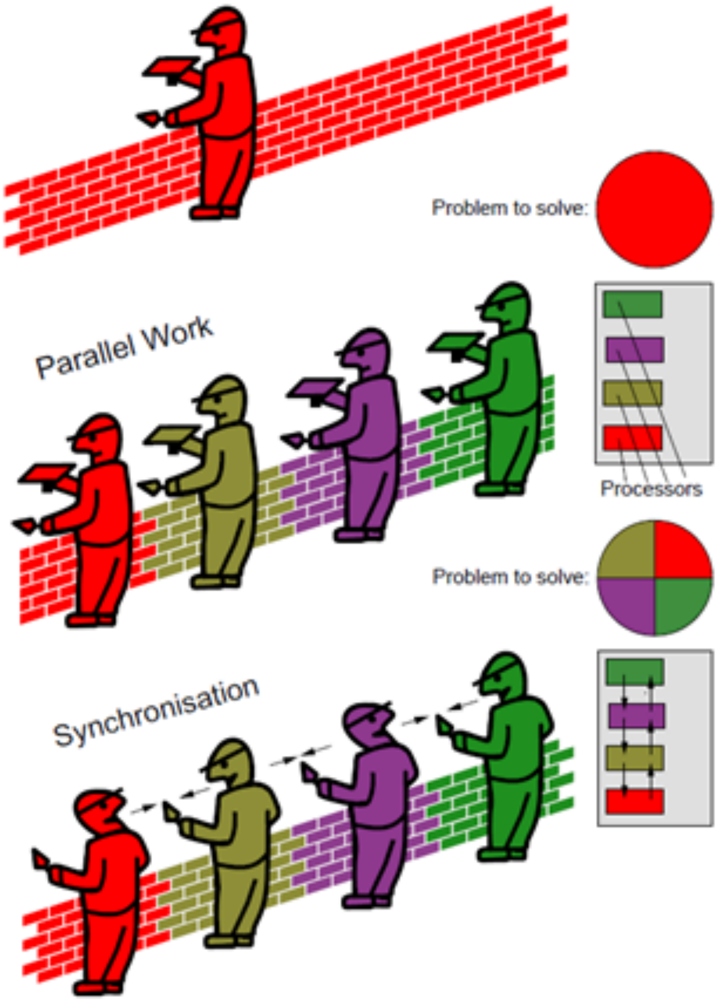


Communications and shared resources are the most crucial issues for parallel performance

Parallel Performance

“Real” parallel

Communications and resource sharing



[picture by W. Baumann]

Parallel Performance

Strong and weak scaling

Different aims require different scaling

- You could try to solve a given (fixed) problem (size) faster by using parallel machines (i.e. more and more CPUs)

  Strong Scaling

- You could try to solve a bigger and bigger problem at a fixed time to solution using parallel machines (…)

  Weak Scaling

Parallel Performance

Strong scaling

Strong scaling – number of CPUs increases, problem size is constant

- Problem size — Ω
- Number of CPUs — N
- Compute time — $T(\Omega,N)$
- Speed-up — $S(\Omega,N)=T(\Omega,1)/T(\Omega,N)$
- Efficiency — $E=S/N$

Parallel Performance

Strong scaling

Maximum speed-up is always limited due to serial part of the code in the program

$$T(1) = T_{serial} + T_{parallelizable}$$ - Single CPU time

$$T(N) = T_{serial} + T(N)_{parallel}$$ - Time on N CPUs

$$T(N)_{parallel} = \frac{T_{parallelizable}}{N}$$ - Best case

Parallel Performance

Strong scaling

Amdahl's law (1967)

$$S(N) = \frac{T_{serial} + T_{parallelizable}}{T_{serial} + \frac{T_{parallelizable}}{N}} \underset{N\to\infty}{=} \frac{T(1)}{T_{serial}} \qquad if \quad \alpha = \frac{T_{serial}}{T(1)} \Rightarrow \boxed{S_{\max} = \frac{1}{\alpha}}$$

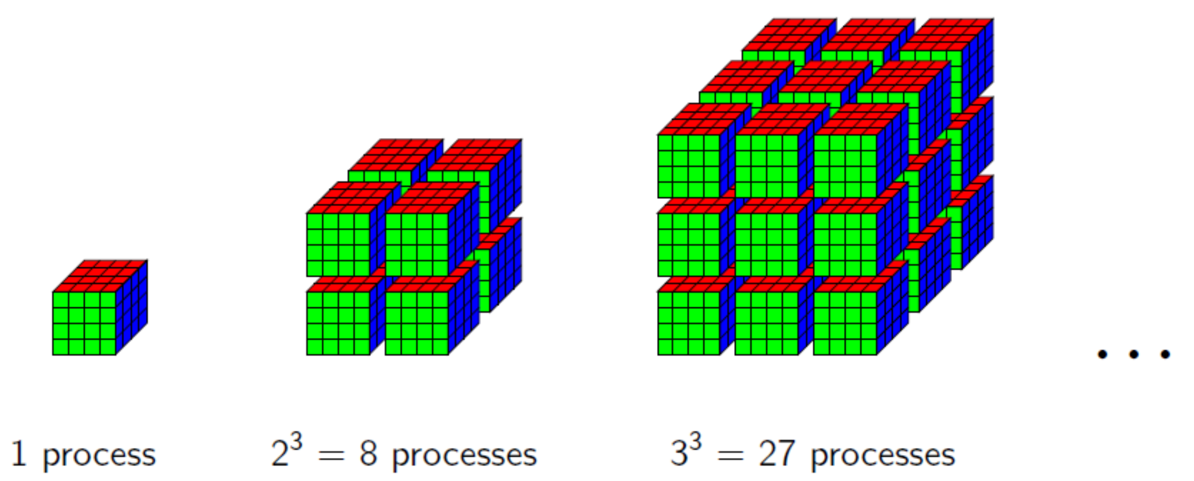
Example: Speed-up at various fractions of serial work

α/N	4	8	32	256	1024	∞
0.1	3.08	4.7	7.8	9.7	9.9	10
0.01	3.88	7.5	24	71	91	100
0.001	3.99	7.9	31	204	506	1000

Parallel Performance

Strong scaling

- Maximum speed-up is still limited even in ideal case of no serial work
  - Unbalanced load (work distribution) among CPUs
  - Communications between CPUs
- Parallel efficiency in range of 0.8 and higher is OK.
- If job efficiency is low one can increase the job size - weak scaling



Parallel Performance

Weak scaling

Weak scaling – problem size per CPU is constant

- Amount of parallel work scales linearly with number of processes

$$T(\Omega \cdot N, N) = T_{serial} + T_{parallél}$$ - Time on N CPUs

$$T(\Omega \cdot N, 1) = T_{serial} + N \cdot T_{parallel}$$ - Single CPU time

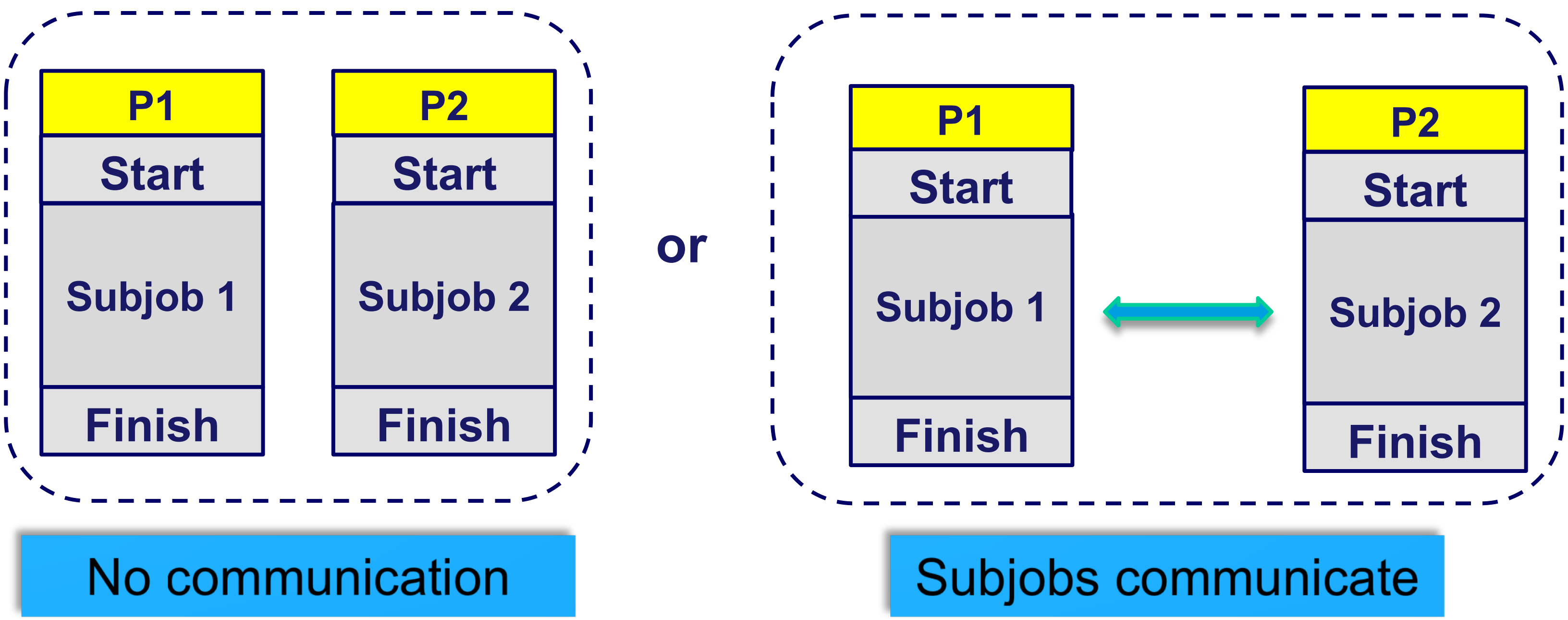
- Gustaffson law (1988) – maximum “speed-up” is unlimited

$$S(N) = \frac{T_{serial} + N \cdot T_{parallel}}{T_{serial} + T_{parallel}} = \alpha + N(1-\alpha)$$

Communications

Introduction

Communication is one of the (if not the most) crucial challenges in HPC. That is true for both hardware engineers and software developers …



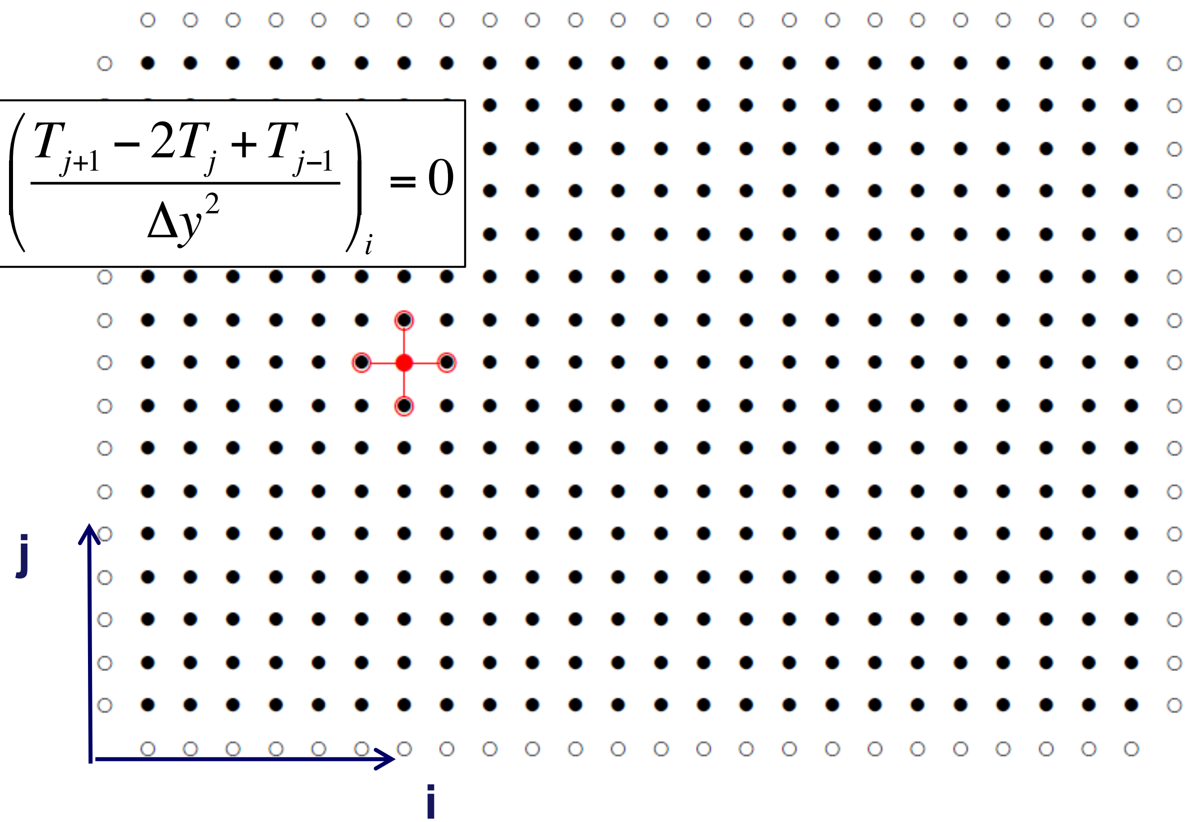
Communications

Example – Laplace equation

Data values in neighbour nodes are needed to compute local value

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0$$

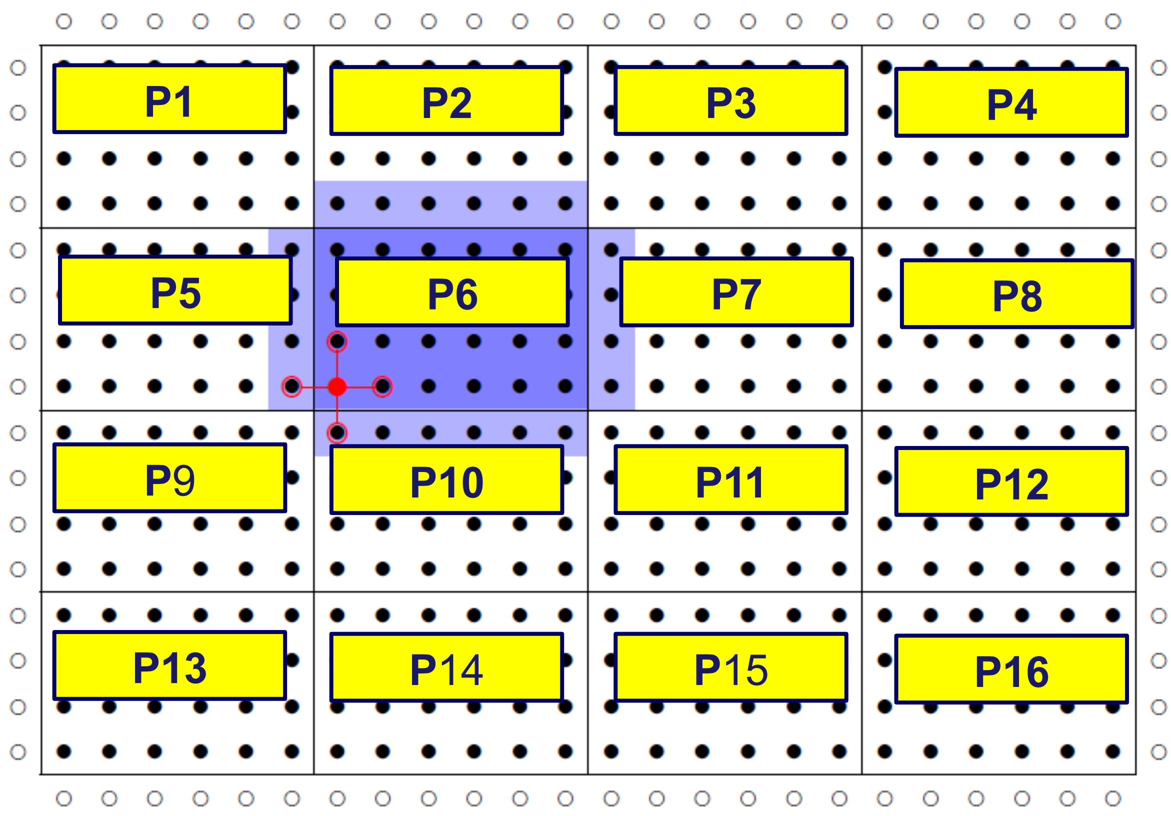
$$\boxed{\left(\frac{T_{i+1} - 2T_i + T_{i-1}}{\Delta x^2}\right)_j + \left(\frac{T_{j+1} - 2T_j + T_{j-1}}{\Delta y^2}\right)_i = 0}$$



Communications

Example – Laplace equation

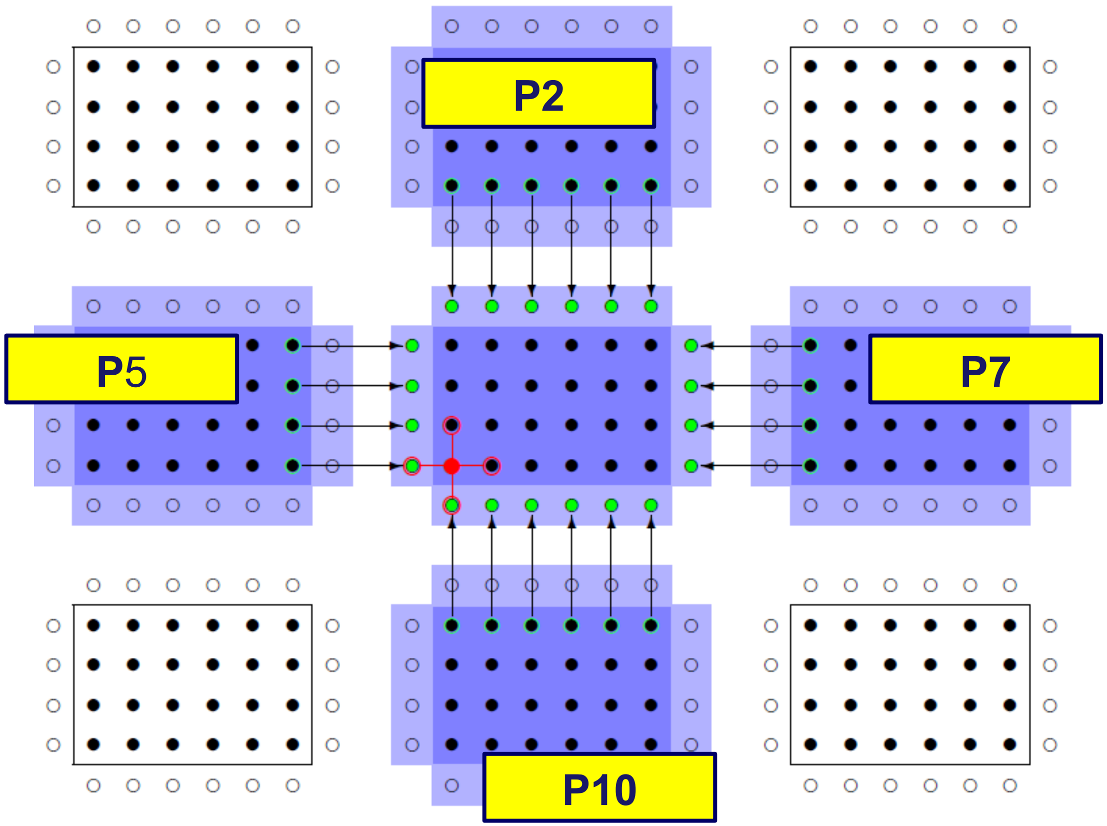
Communications occurs when nodes are distributed over processes



Communications

Example – Laplace equation

Local communications between processes



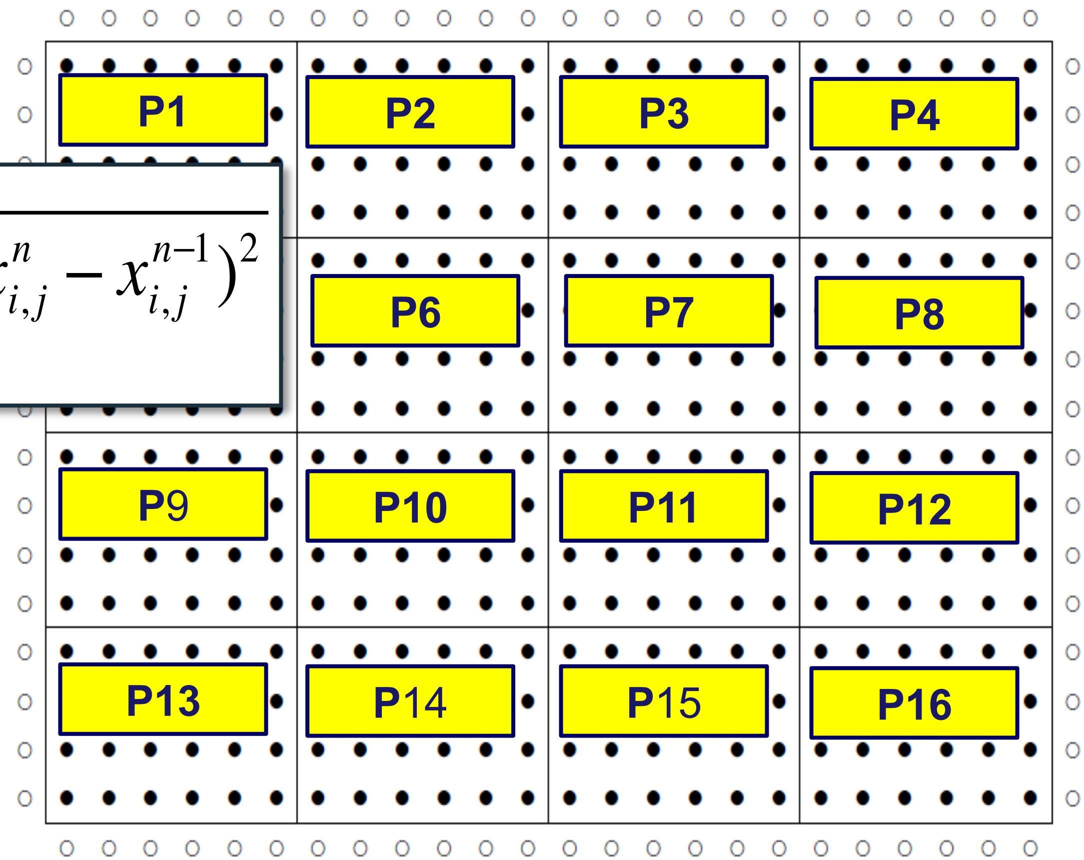
Communications

Example – Laplace equation

Global communication from all processes to all processes

$$err = \left\| \underline{x}^{n} - \underline{x}^{n-1} \right\| = \sqrt{\sum_{i,j} (x_{i,j}^{n} - x_{i,j}^{n-1})^{2}}$$

global SUM is needed

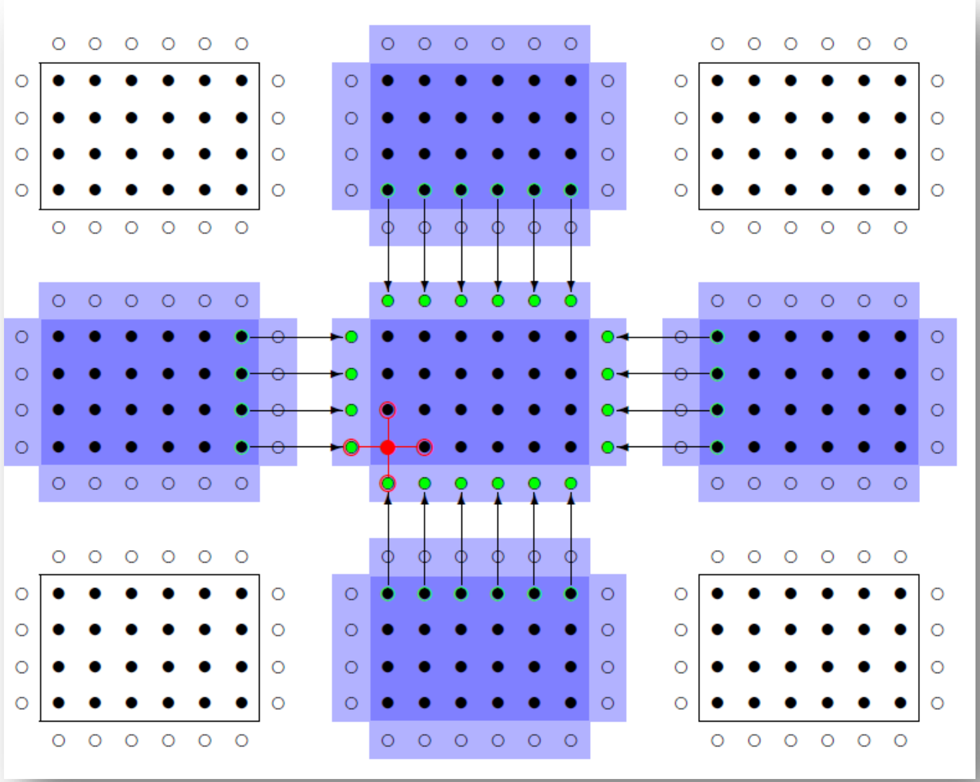


Communications

Shared and distributed memory

How are communications handled?

- Shared memory
  - Global address space
- Distributed memory
  - Data is transferred via a network



Communications

Network performance

Network basic performance characteristics

- Bandwidth
  - Amount of data transferred per time unit [G**b**/s]
- Latency
  - Time spent between start of data transfer request and arrival of the first byte [nsec or µsec]
- Scalability
  - Defines the network performance with increased number of connected nodes

Communications

Network performance

Network basic performance characteristics

- Total time for a message of N bytes with bandwidth BW

$$T = T_l + \frac{N}{BW}$$

- Effective bandwidth

$$BW_{eff} = \frac{N}{T_l + \frac{N}{BW}}$$

Communications

Network performance

Example: send 1double precision real (Mellanox 54Gb/s FDR IB)

$$\left.\begin{array}{l} T_l = 0.7\mu s = 0.7\cdot 10^{-6} s \\ BW = 6.8GB/s = 6.8\cdot 10^{9} B/s \\ N = 8B \end{array}\right\} \Longrightarrow \begin{array}{l} \dfrac{N}{BW} = 1.2\cdot 10^{-9} s \\ BW_{eff} = 11.4MB/s = 0.002BW \end{array}$$

Send more data at once!

Communications

Overhead

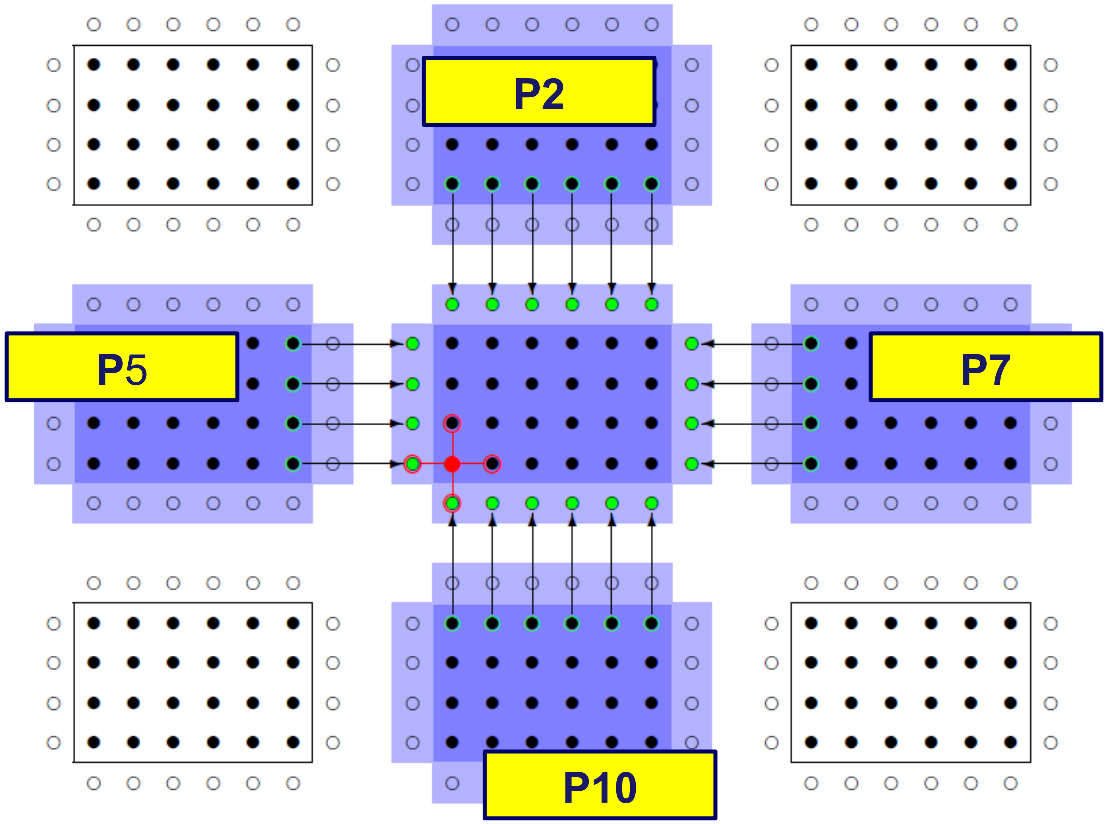
Assume there is no serial part in job

- Single CPU job time $$T(1) = T_{work}$$
- N CPUs job time $$T(N) = \frac{T_{work}}{N} + T_{comm}(N)$$
- Speed-up $$S(N) = \frac{T(1)}{T(N)} = \frac{T_{work}}{\frac{T_{work}}{N} + T_{comm}(N)} = N \cdot \frac{1}{1 + \frac{N \cdot T_{comm}(N)}{T_{work}}}$$
- Efficiency $$E(N) = \frac{S(N)}{N} = \frac{1}{1 + \frac{N \cdot T_{comm}(N)}{T_{work}}} = \frac{1}{1+\alpha}, \quad \boxed{\alpha = \frac{T_{comm}(N)}{T_{work}(N)}}$$

Communications

Overhead

Here α would be large - 16 comminucated points / 24 computing ponts per process

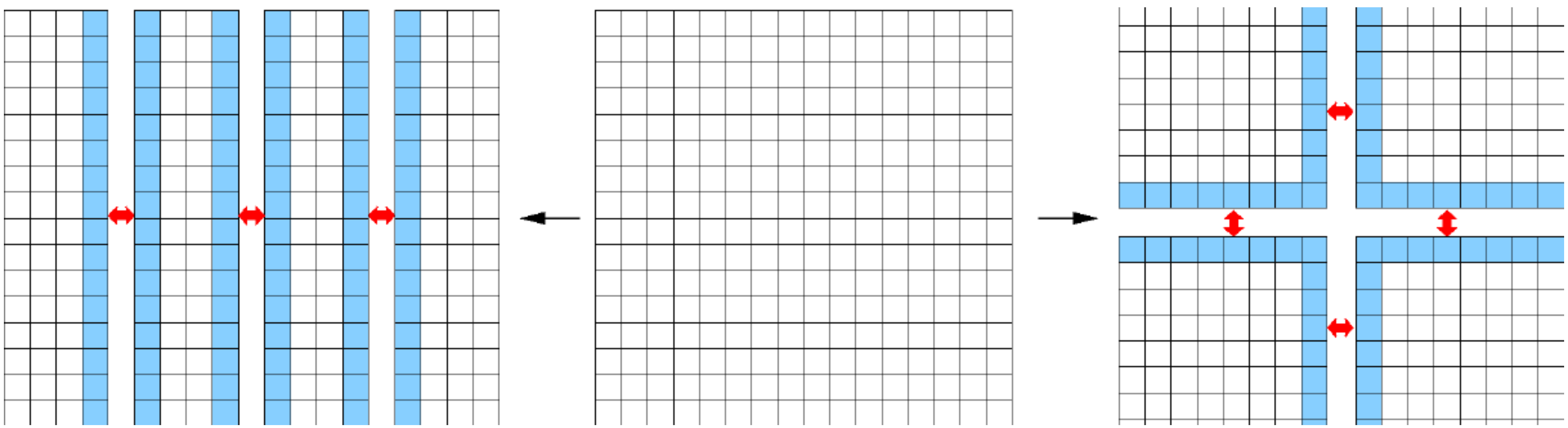


Communications

Overhead

Example: 2D regular grid with M x M nodes

Single job time $T_{work} = a \cdot M^2$



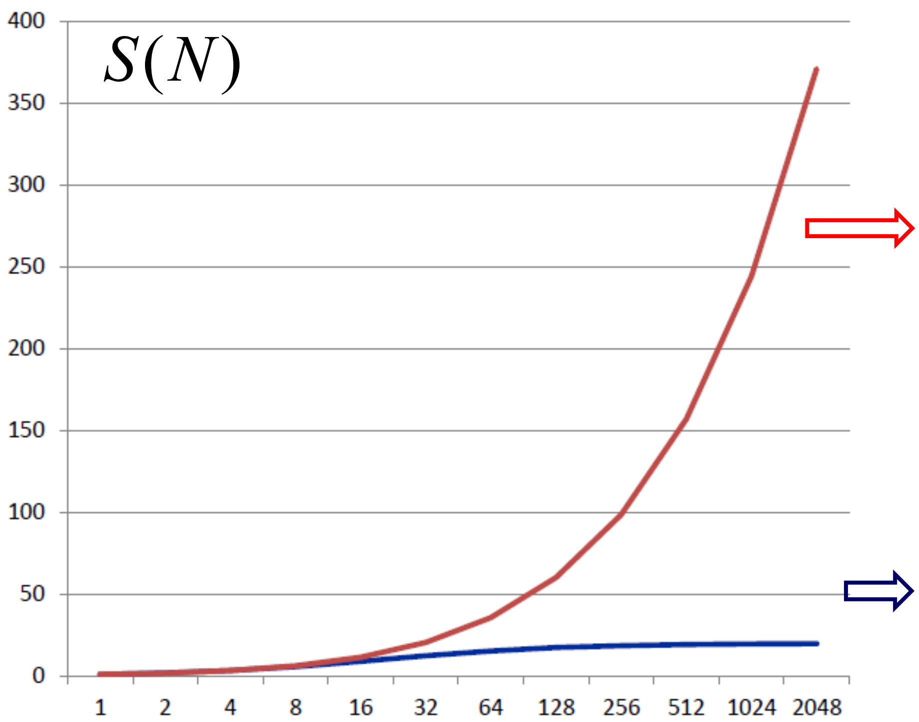
$$T_{comm_1}(N) = c \cdot 2 \cdot M$$

$$T_{comm\,2}(N) = \frac{c \cdot 4M}{\sqrt{N}}$$

Communications

Overhead

Example: 2D regular grid with M x M nodes



Squares

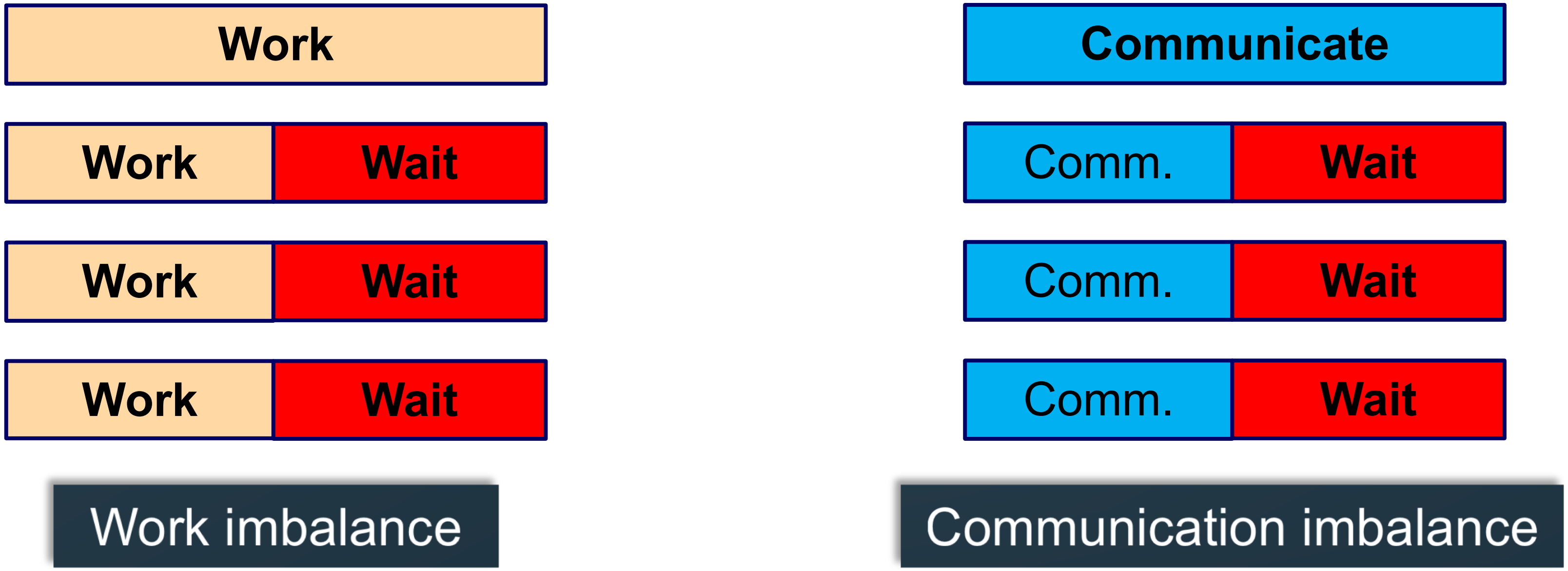
$$S_1(N) = N \cdot \frac{1}{1 + 2\tilde{a}\,\frac{\sqrt{N}}{M}}$$

Slices

$$S_2(N) = N \cdot \frac{1}{1 + \tilde{a}\,\frac{N}{M}}$$

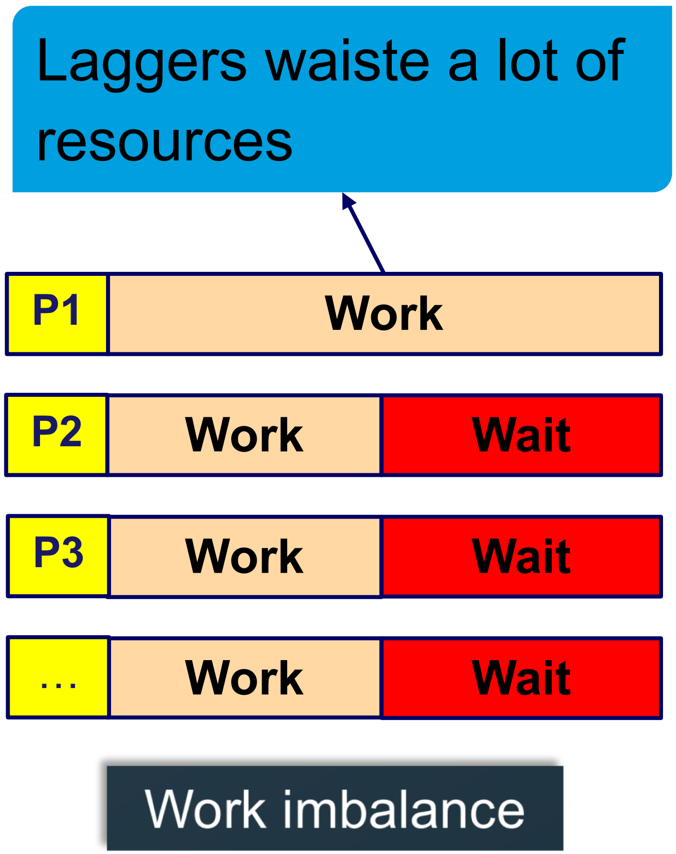
Load balance
Definition

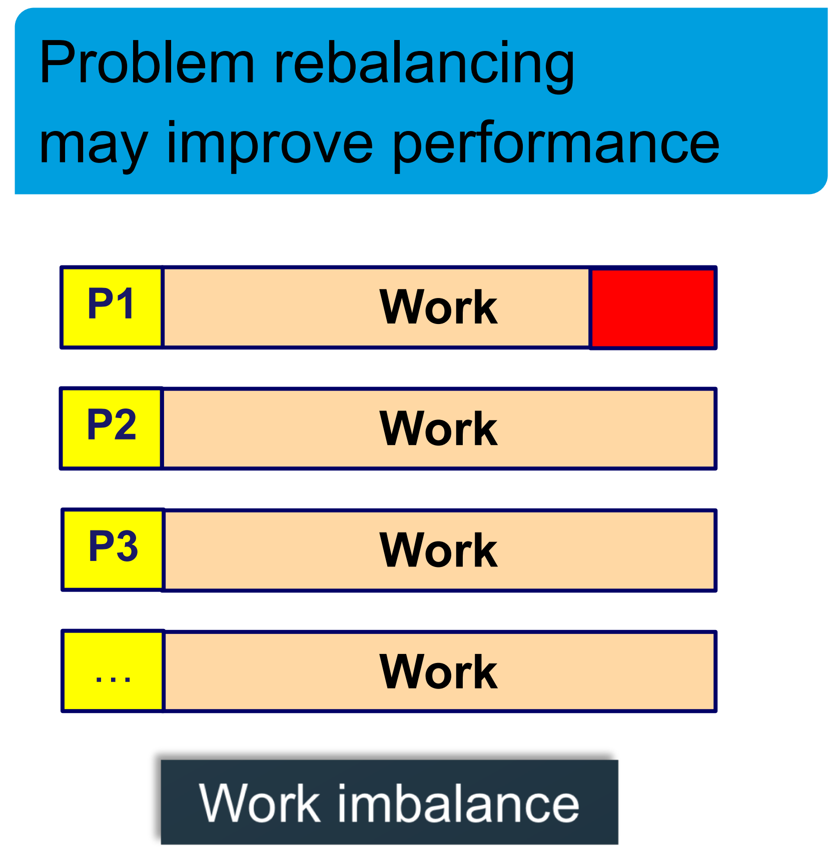
- Work distribution between processes/threads
- Communication distribution between processes/threads
- Crucial for performance



Load balance

Example



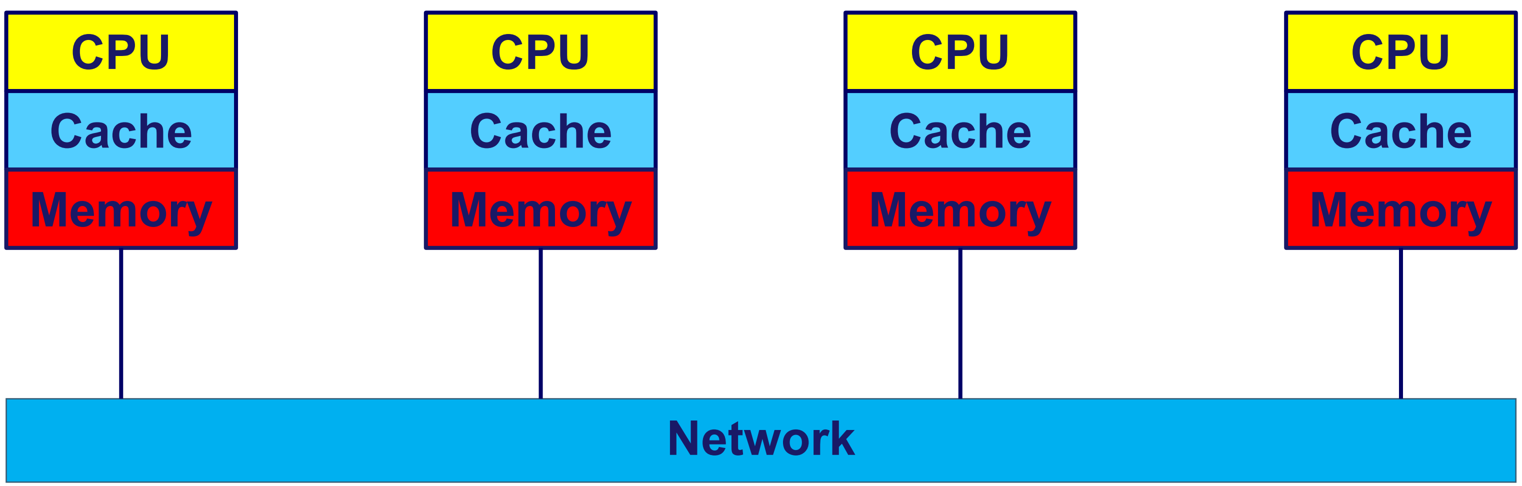


Software development

Overview

You as developer is responsible to use available hardware & software efficiently

- Distributed memory
  - Explicit message passing (MPI)
  - Implicit (Fortran Coarrays, UPC,...)

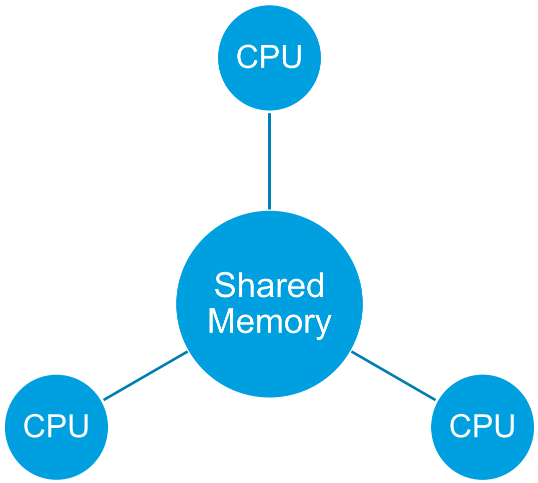


Software development

Overview

You as developer is responsible to use available hardware & software efficiently

- Shared memory - direct memory access
  - OpenMP, OpenCL, OpenAcc
  - Shmem, POSIX threads
  - All from distributed memory



Software development

Overview

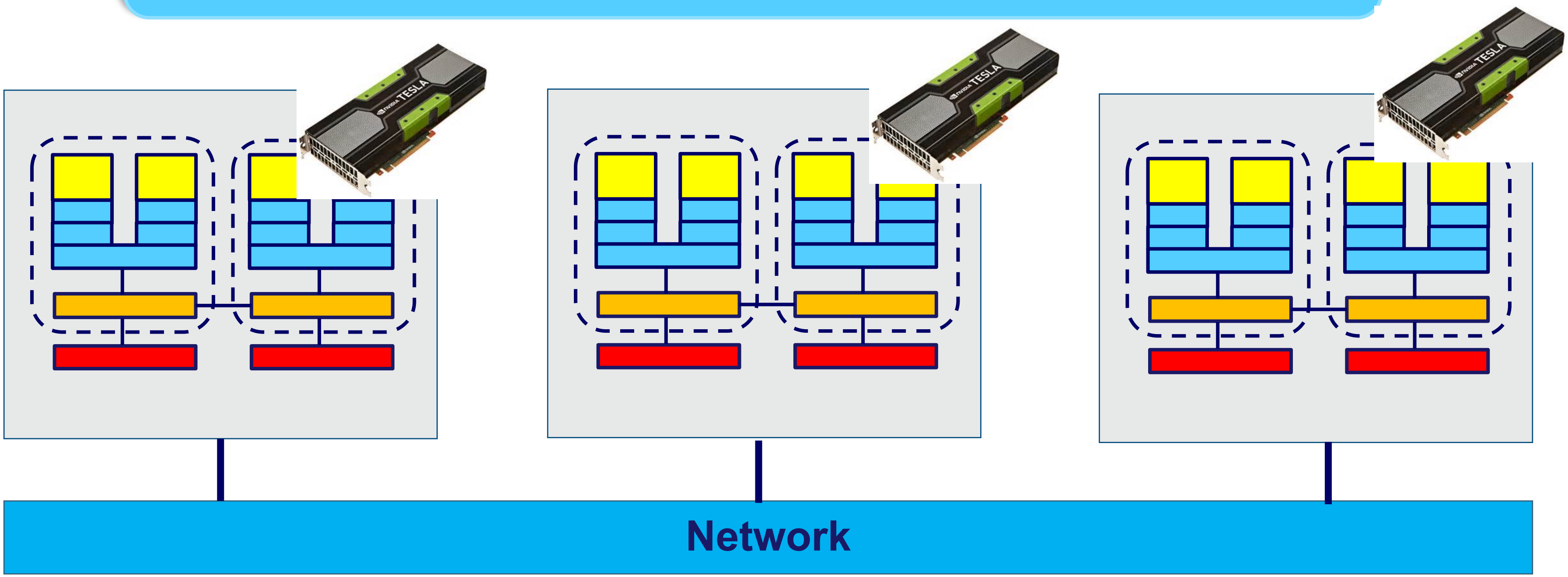
You as developer is responsible to use available hardware & software efficiently

- GPU
  - CUDA
  - OpenACC, OpenCL

Software development

Overview

You as developer is responsible to use available hardware & software efficiently



Combination of technologies makes it even harder, but at the same time more efficient

Summary

- HPC machine – complex (and expensive) hardware, many of-the-shelf CPUs, memory and storage that are assembled together and linked with a very fast network
- Usually falls into MIMD (or hybrid SIMD/MIMD) category
- Trend to increasing number of cores and accelerators
- Communications are essential and most challenging part of HPC
- Efficient parallel program should have minimum of serial computation, efficiently balance load and keep communications/computations ratio as low as possible
- Different hardware/architectures require (in general) different approaches to programming