

GWDG – Kurs  
Parallel Programming with MPI

# Parallel Computing: Basic Principles

Oswald Haan  
oahan@gwdg.de

# Parallel Computing

Wikipedia:

“Parallel computing is a type of computation in which many calculations or processes are carried out simultaneously”

Why

How

# Overview

- **Why**
  - Evolution of Computing Power
- **How**
  - Hardware Parallelism
  - Data Dependency
  - Programming Models
  - Parallel Efficiency

# Demand for more Computing Power

- Simulating complex systems in different areas at all scales:  
cosmology, climate, engineering, drugdesign, biochemistry,  
elementary particles, . . .
- Analysing huge datasets from experiments and observations:  
particle physics, genomes, internet, . . .
- Artificial Intelligence  
training and using AI systems, . . .

# Delivering more Computing Power

computing power of a computing system is defined as

$r$  [flop/s, Kilo-, Mega-, Giga-, Tera-, Peta-, Exa-flop/s] =  
maximal number of floating point operations per second  
delivererable by the system

$r$  depends on computer system parameters:

$N$  : number of computing elements in the system

$n$  : number of circuits involved in one floating point operation

$\tau$  : cycle time of a circuit in the system

$f = \tau^{-1}$  : frequency or clock rate of the system

$$r = N \times n^{-1} \times f$$

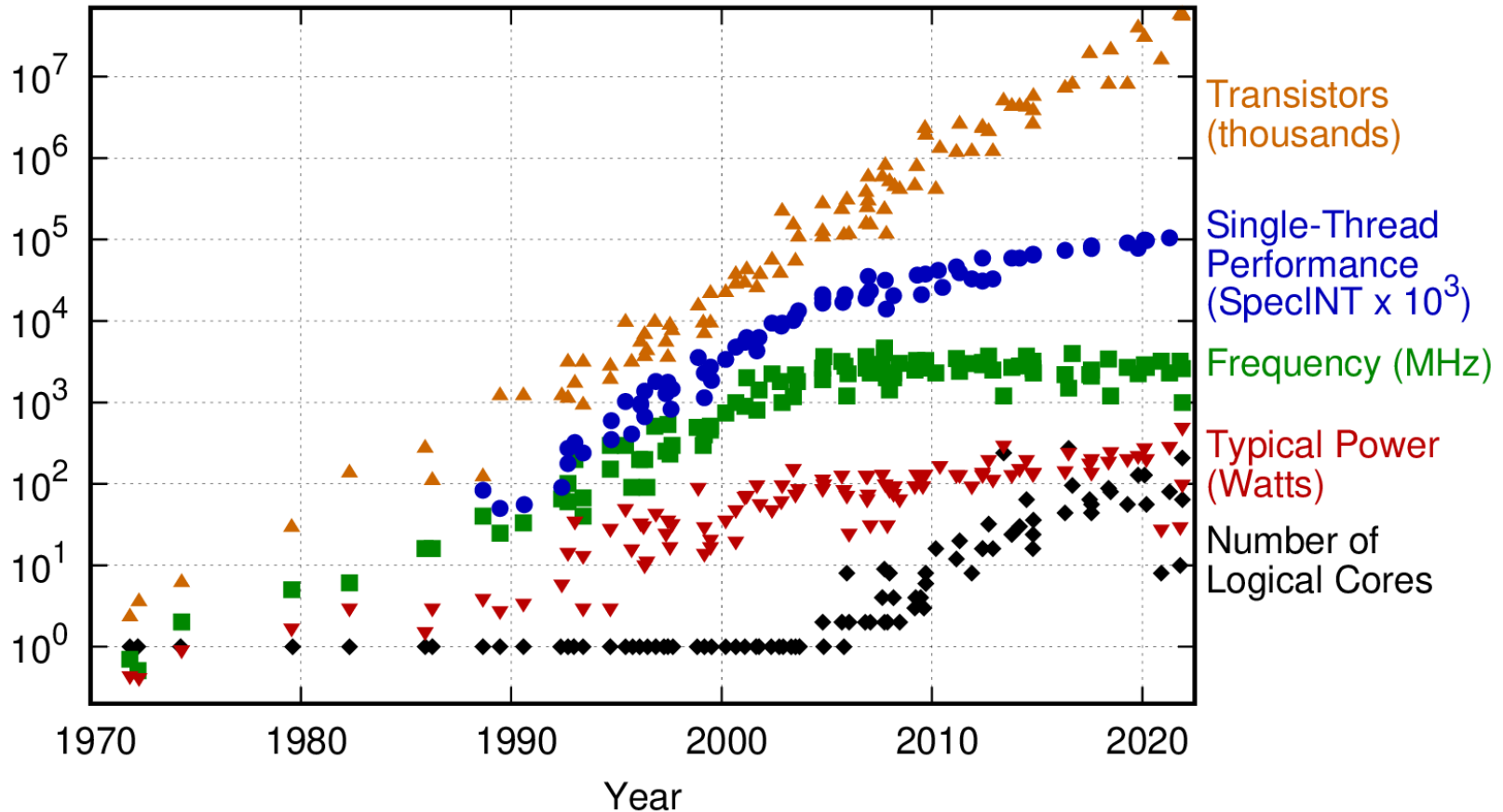
more computing elements

higher clock rate

**more computing power**

# Evolution of Microprocessors

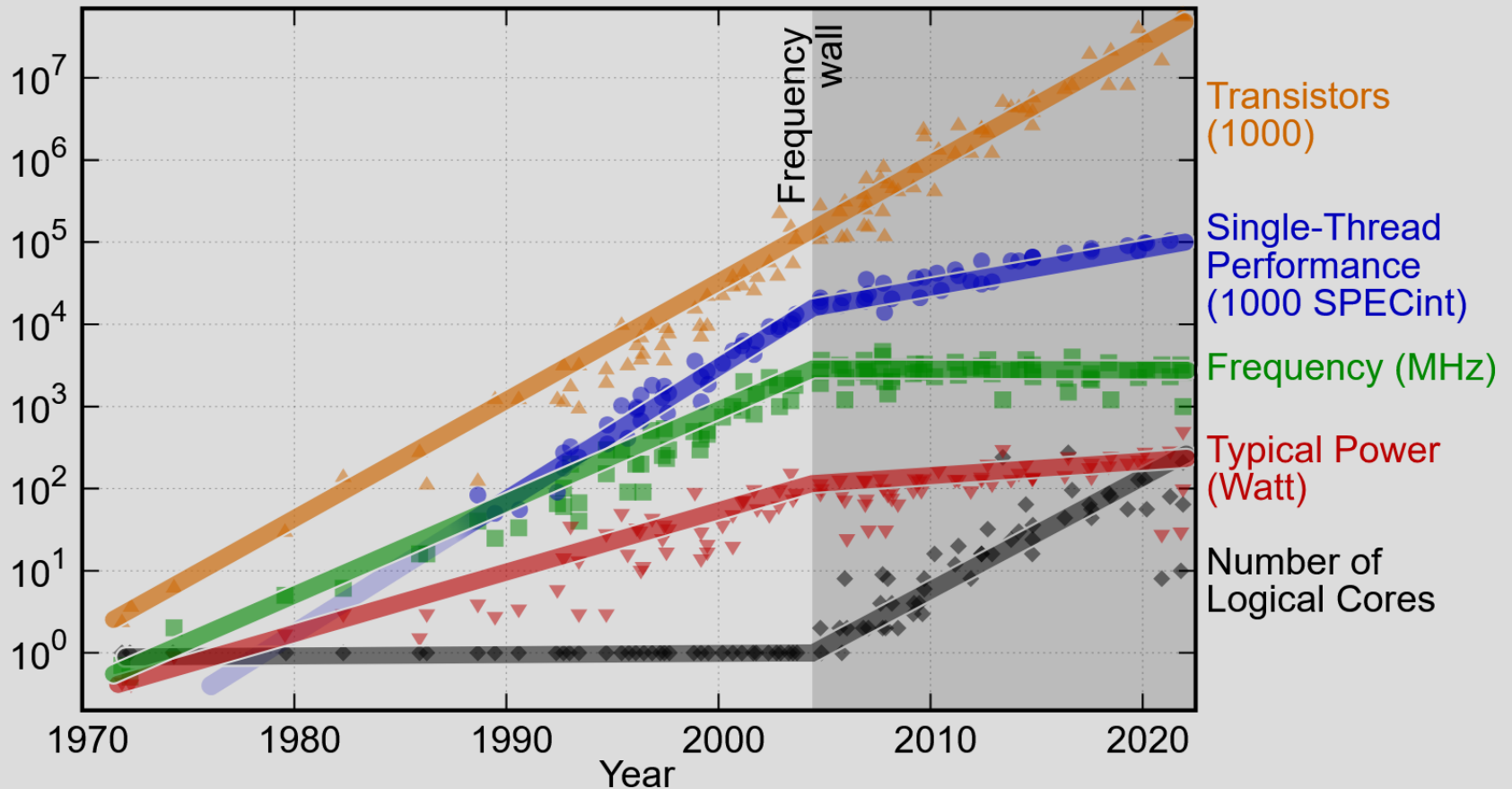
## 50 Years of Microprocessor Trend Data



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten  
New plot and data collected for 2010-2021 by K. Rupp

# Evolution of Microprocessors

50 Years of Microprocessor Trend Data



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond und C. Batten  
New plot and data collected for 2010-2021 by Karl Rupp, CC BY-4.0  
Trend lines, frequency wall (inspired by The Economist), slight touch-up by Marcel Waldvogel, 2024-10-12, CC BY-4.0

# (Not-)Moore's Law

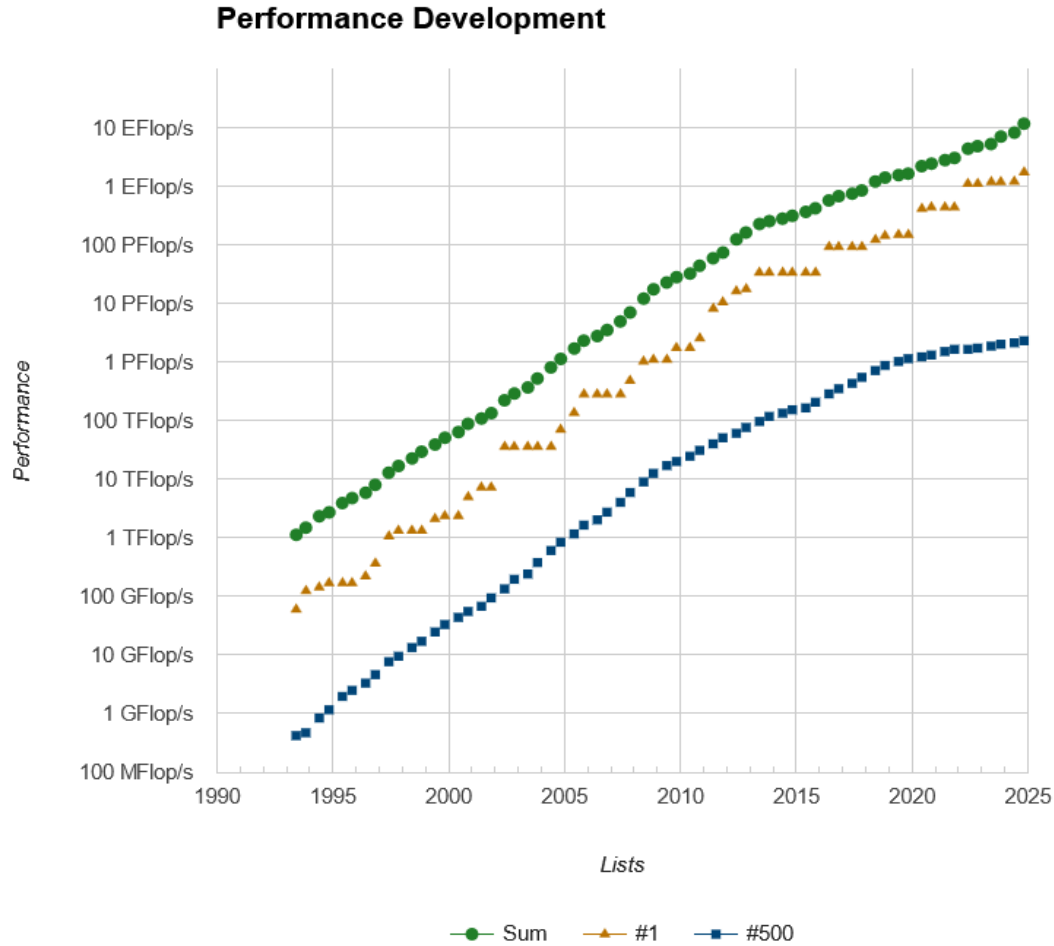
- **Moore's Law**

(Intel co-founder Gordon E. Moore, 1965) :

The number of transistors on integrated circuits doubles approximately every two years (or every 18 months).

- **Not-Moore's Law** is that clock rates do, too
- Moore's Law holds (and will for some time)
- Not-Moore's Law held until  $\approx$  2003, then broke down

# Computing Power of TOP500 Supercomputers



# Conclusion from the Why of Parallel Computing

Single compute nodes have  $O(10-100)$  computing elements

Compute clusters have  $O(1\ 000\ 000)$  computing elements

Using the computing power of nodes and clusters for complex simulations and for analysing large datasets is possible ,  
and requires

## Parallel computing

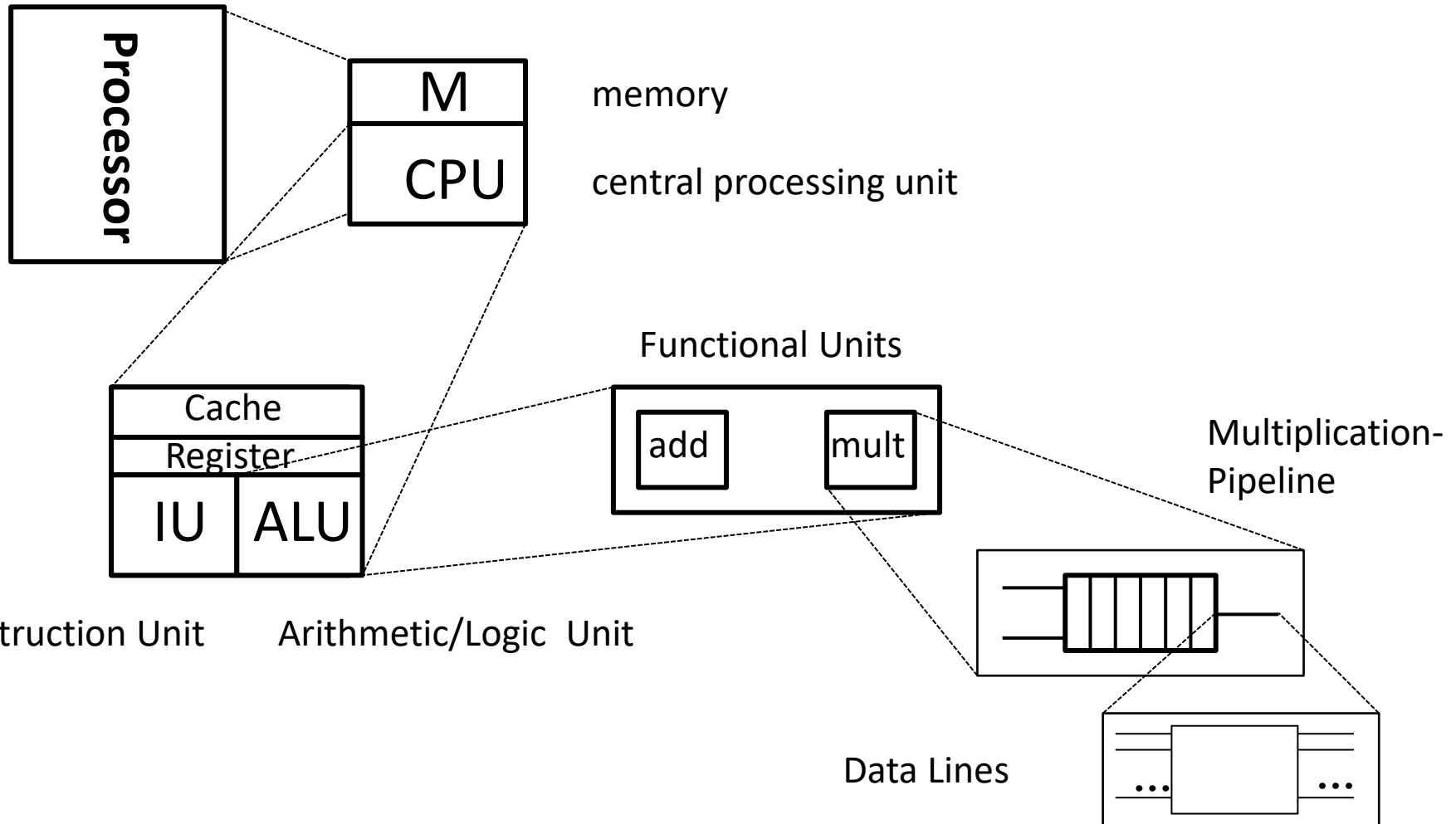
# Faster Problem Solving with Parallel Computing?

- If hardware supports simultaneous execution of several independent operations
  - ➔ **Parallel Hardware**
- If problem can be decomposed into independent pieces
  - ➔ **Parallel Algorithm**
- If software maps parallel algorithm to parallel hardware
  - ➔ **Parallel Programming Language**
- If resulting application solves the problem faster
  - ➔ **Parallel Efficiency**

# Overview

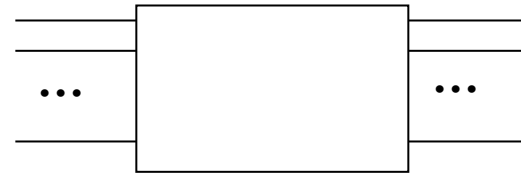
- Why
  - Evolution of Computing Power
- How
  - Hardware Parallelism
  - Data Dependency
  - Programming Models
  - Parallel Efficiency

# Components of a Processor



# Parallelism in Hardware Components

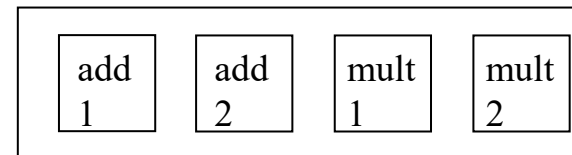
parallel bit-processing  
(64bit-architecture)



parallel segments in arithmetic pipelines  
(assembly line processing)



parallel functional units  
(superscalar architecture)

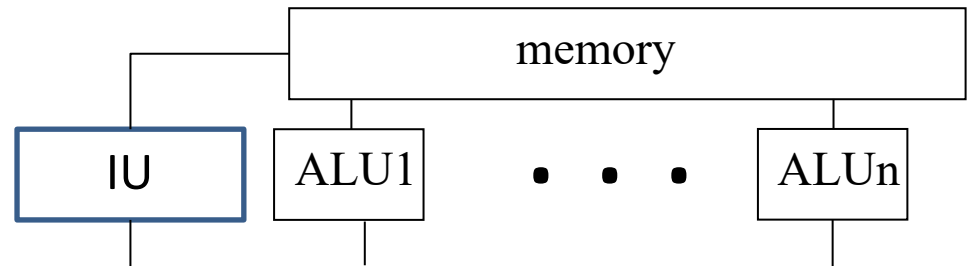


Maximal 4 double precision floatingpoint operations per cycle time:  
10 GigaFlop/s for 2,5 GHz cycle frequency

# Parallelism with Multiple Computing Elements

**SIMD** (Single Instruction Stream-Multiple Data Streams)

parallel functional units  
(graphics processor)



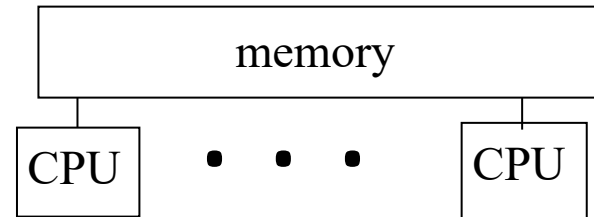
# Parallelism with Multiple Computing Elements

## **MIMD** (Multiple Instruction Streams-Multiple Data Streams)

parallel CPUs

**shared memory multiprocessor**

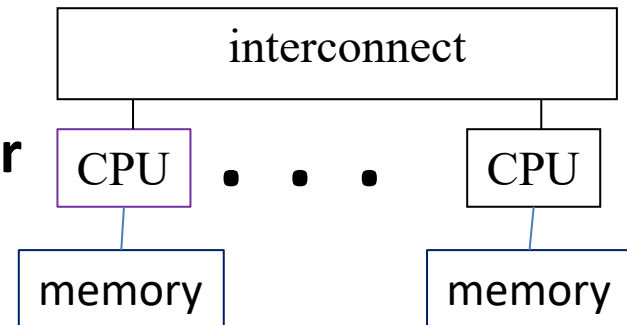
e.g. multi-core processor



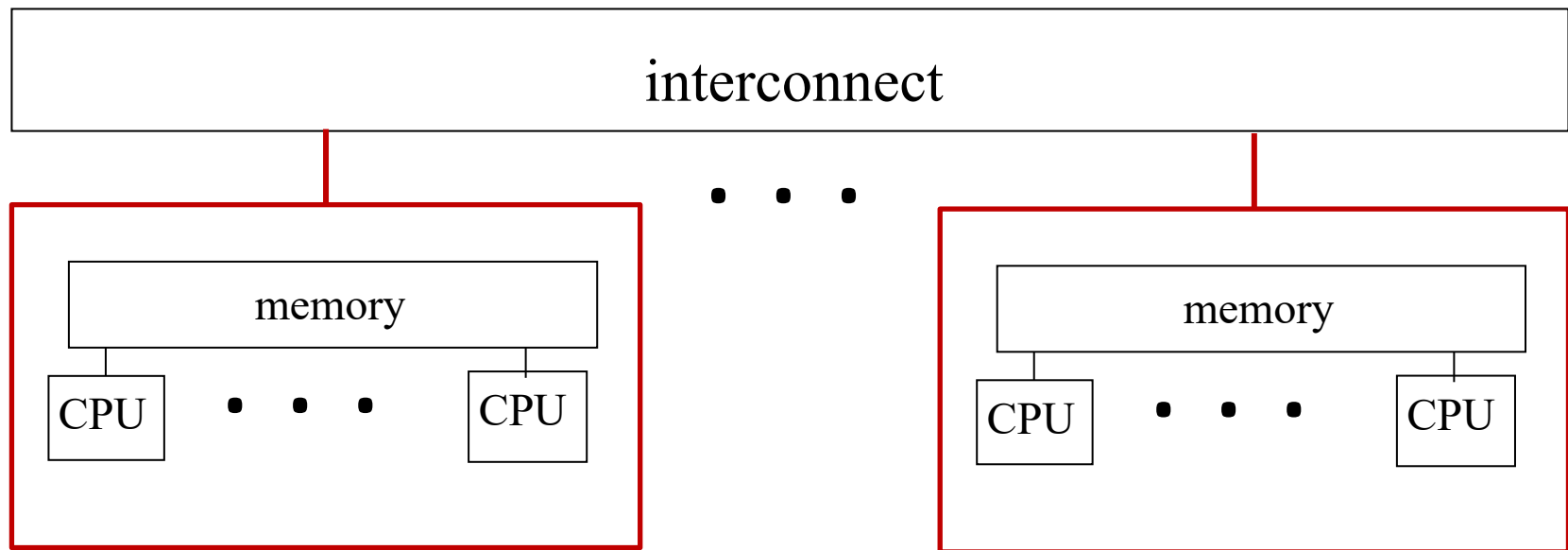
parallel processors

**distributed memory multicomputer**

e.g. cluster of single-core processors

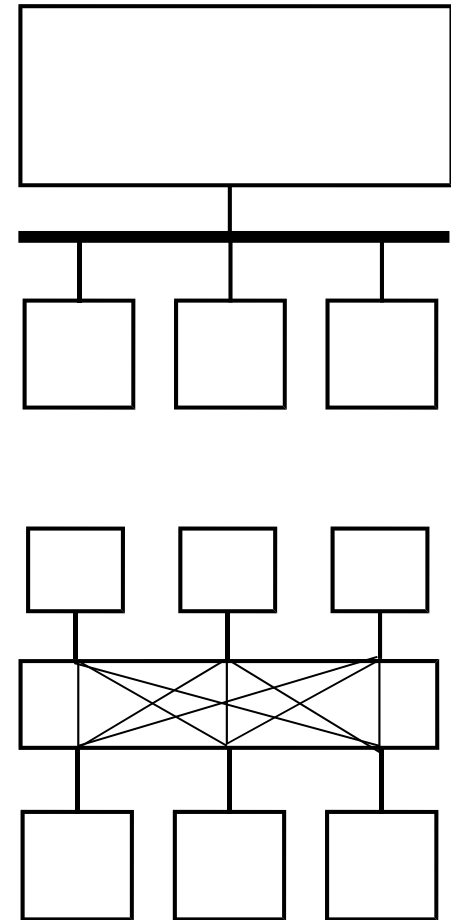


# Hybrid-Systems: Cluster of Shared-Memory Nodes



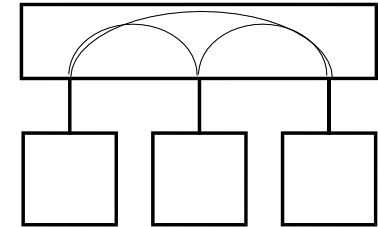
# Connection in Shared Memory Systems

- **Bus**
  - Serial memory access
  - non-scalable
  - low latency
  
- **Switch**  
(Hypertransport, Quick Path)
  - parallel memory access possible  
(for advantageous data layout)
  - scalable
  - Larger latency

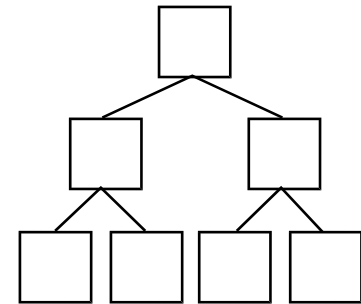


# Connection in Systems with distributed memory

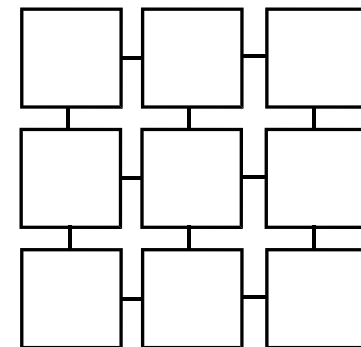
- Switch



- Tree



- 2d Lattice



# Parameters of Network-Connections

- latency  $t_{lat}$   
arrival time of the first bit

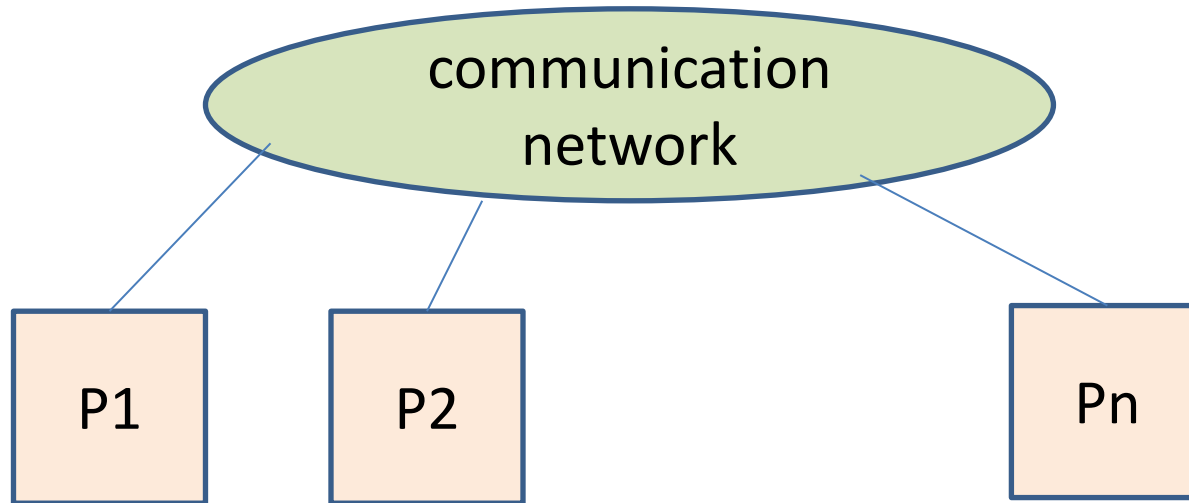
- bandwidth  $c$   
rate of data transfer

- effective bandwidth

$$c_{eff} = c \cdot \frac{1}{1 + \frac{t_{lat} \cdot c}{n_{dat}}}$$

- bisection width  
minimal number of connections between two equally sized parts of the network

# Relevance of Network Parameters



Every processor has computing power  $r$   
network has latency  $t_{lat}$  und bandwidth  $c$

$t_{lat} \cdot r$  number of flops, which could be executed during latency

$64 \cdot r / c$  number of flops, which could be executed during transfer of 8 Bytes (floating point number)

# Typical Network Parameters

	Latency $t_{lat}$ [ $\mu\text{sec}$ ]	Bandwidth $c$ [ $Gbit/s$ ]	Computing Power $r$ [ $GFlop/s$ ]	$t_{lat} \cdot r$	$64 \cdot r / c$
Intel Broadwell Single Node	0,23	64	1 core: 8,8	2 024	9
Intel Cascade Lake Single Node	0,34	75	1 core: 9,2	3 032	11
Intel Broadwell Infiniband FDR	1,3	50	24 cores: 211	274 300	270
Intel Cascade Lake Intel Omni-Path	1,2	90	96 cores: 883	1 060 000	628



# The future was (always) massively parallel

---

Connection Machine  
CM-1 (1983)

12-D Hypercube

65536 1-bit cores  
(AND, OR, NOT)

Rmax: 20 GFLOP/s

Today's notebook PC



# TOP500(2019) #1: Summit (Oak Ridge National Laboratory)

9216 IBM Power 9  
(22 cores each)

27648 NVIDIA GPUs  
(Volta GV100)

Rmax : 146 PetaFlop/s



# TOP500(2020) #1: FUGAKU (RIKEN Center for Computational Science (R-CCS) in Kobe, Japan)

158,976 nodes ARM A64FX v8.2-A  
( 48 cores each)

Rmax : 416 PetaFlop/s



# TOP500(2022) #1: FRONTIER (Lawrence Livermore National Laboratory)

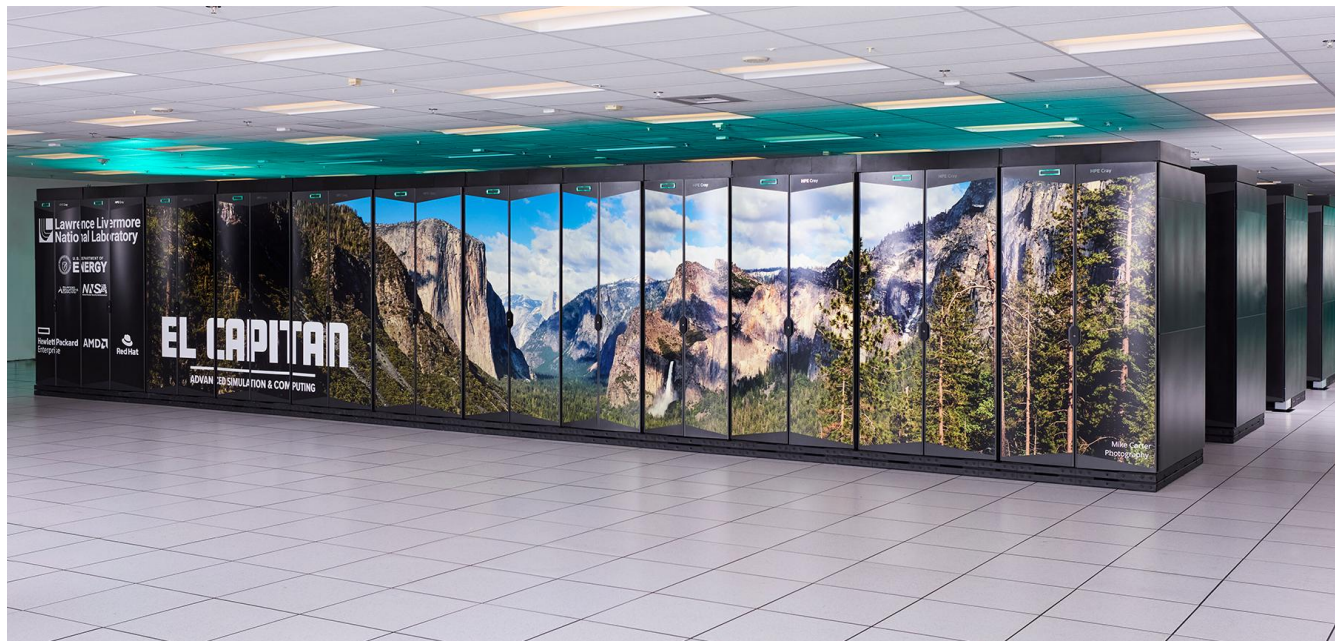
9472 nodes with  
1 AMD-EPYC-7xx3 ( 64 cores )  
4 AMD-Instinct MI250X-GPUs

Rmax : 1.353 ExaFlop/s

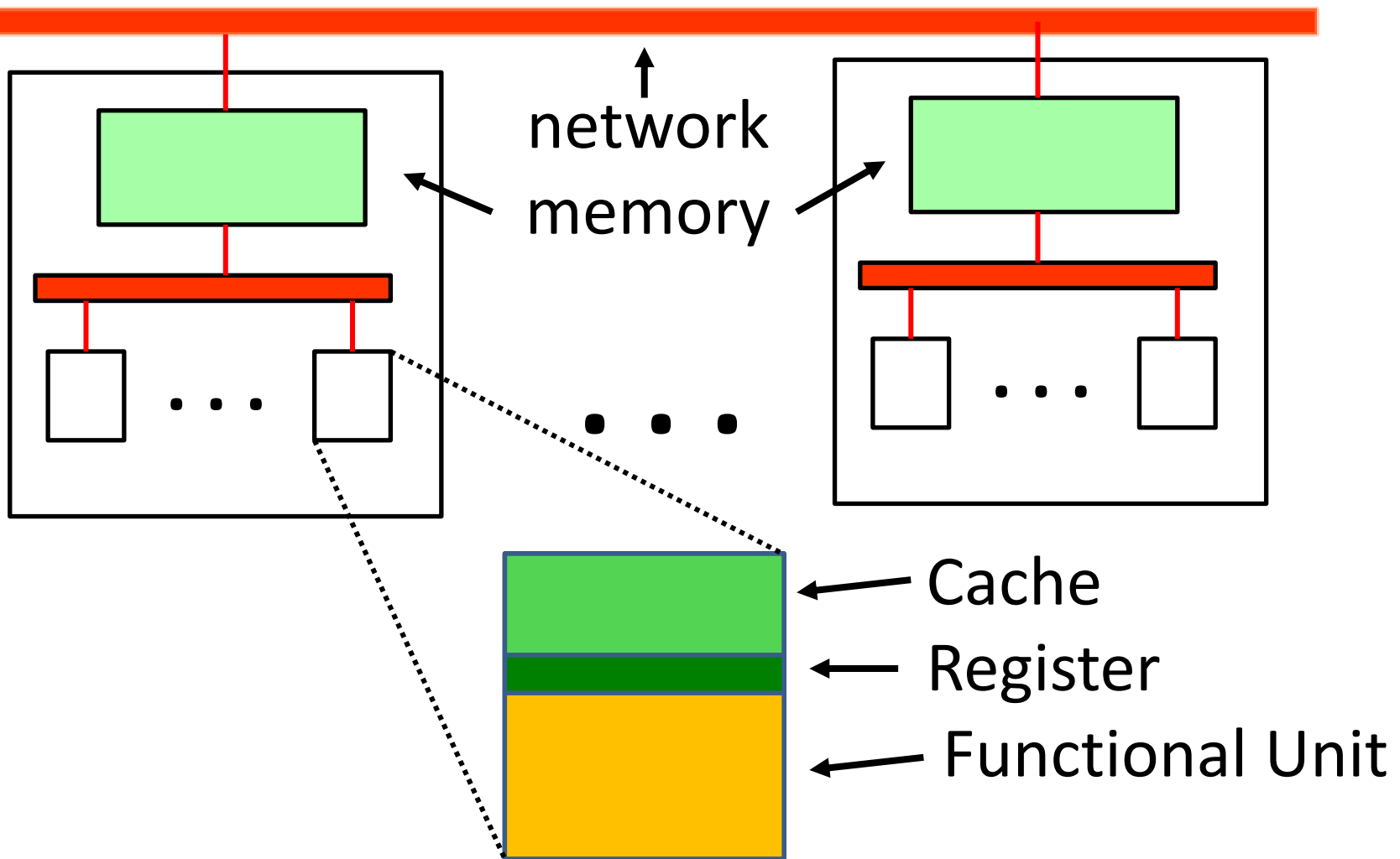


# TOP500(2024) #1: EL CAPITAN (DOE/SC/Oak Ridge National Laboratory)

11 136 nodes with 4 AMD Instinct M1300A (CPU/GPU) Systems  
~1 million CPU cores  
~10 million GPU cores  
Rmax : 1 742 ExaFlop/s

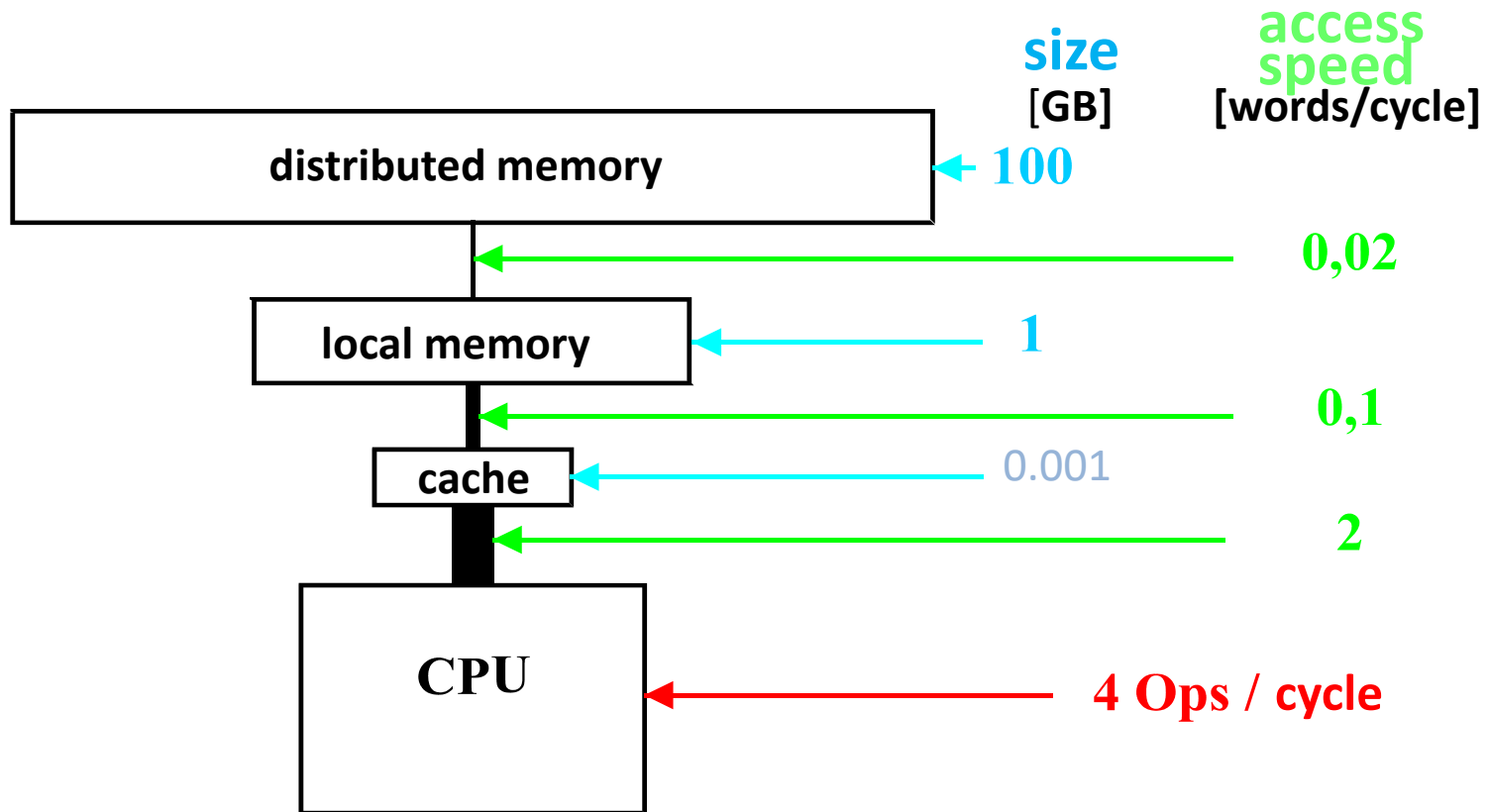


# Memory-Hierarchy in Parallel Computers



# Speed of Memory Accesses

## Memory Hierarchy



# Golden Rule for Data Access

The maximal computing power of a parallel system

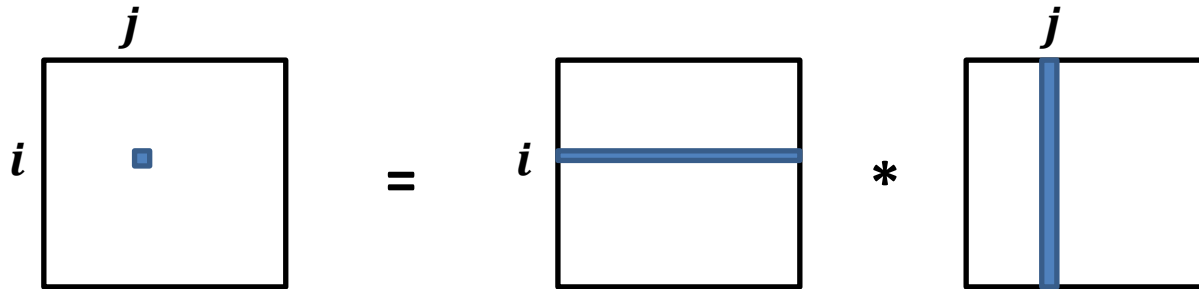
$$R_{\max} = \text{clockrate} * \text{\#CPUs} * \text{\#Operations per CPU}$$

can be realized exclusively for operating on data in cache.

In all other cases:

memory bandwidth limits the usable computing power

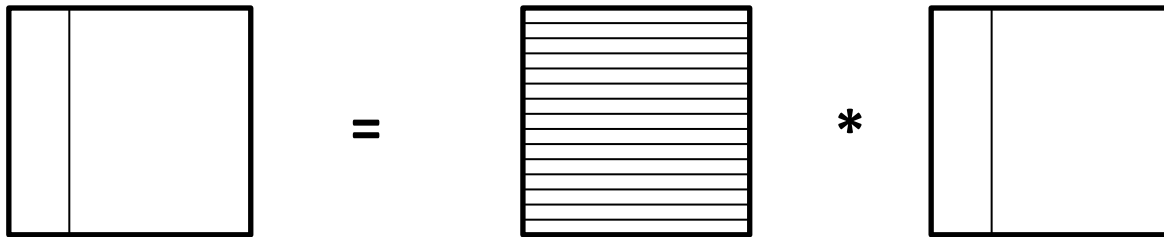
# Memory Access Patterns for Matrix Multiplication



$$a_{ij} = \sum_{k=1}^n b_{ik} c_{kj}, i, j = 1, \dots, n$$

$$N_{op} = 2n^3$$

# Multiplication Column by Column



Calculation of one column:

$2n^2$  operations

$n^2$  data to be read from memory

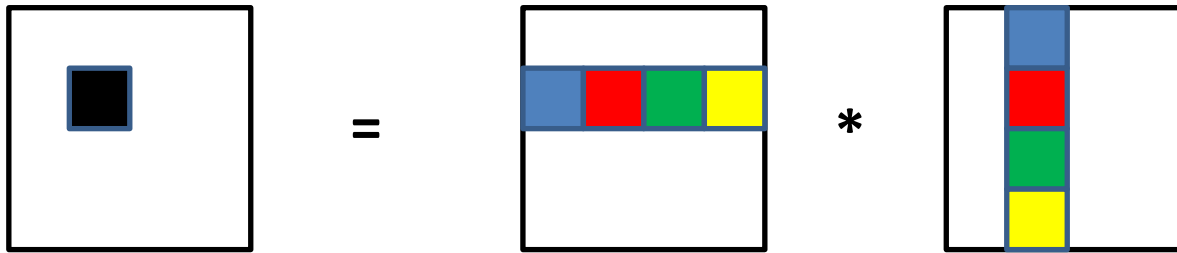
If  $n^2$  x size of numbers  $>$  Cachesize



slowing down due to Golden Rule

# Multiplication Block by Block

block-size  $bsz^2$



Computing a block multiplication  $a_{ij}^{(b)}$  :

$$N_{op} = 2bsz^3 \quad N_{in} = 2bsz^2$$

$2bsz^2 < \text{Cachesize}$  obeys Golden Rule

# Overview

- Why
  - Evolution of Computing Power
- How
  - Hardware Parallelism
  - Data Dependency
  - Programming Models
  - Parallel Efficiency

# Parallel Algorithms

- Identify data dependencies

vector addition:  $c(i) = a(i) + b(i)$ ,  $i = 1, 4$

$$c(1) = a(1) + b(1) \quad c(2) = a(2) + b(2) \quad c(3) = a(3) + b(3) \quad c(4) = a(4) + b(4)$$

sum of vector elements  $s = a(1) + a(2) + a(3) + a(4)$

$$s1 = a(1) + a(2) \quad s2 = a(3) + a(4)$$

$$s = s1 + s2$$

generating random numbers  $z(i) = a * z(i-1) \bmod m$ ,  $i = 1, 4$

$$z(1) = a * z(0) \bmod m$$

$$z(2) = a * z(1) \bmod m$$

$$z(3) = a * z(2) \bmod m$$

$$z(4) = a * z(3) \bmod m$$

# Types of Parallel Algorithms

- **embarrassingly parallel**  
Simultaneous execution of independent tasks using multiple processes
- **simulation of extended domains**  
Splitting a domain into subdomains, which are mapped to different processes.  
Every process simulates the degrees of freedom in its subdomain. Communication of boundaries.
- **Algorithms for data fields (e.g. matrices):**  
Partitioning the fields into subfields, which are mapped to different processes.  
Every process manipulates data from its subfield, using data communicated from other subfields

# Overview

- Why
  - Evolution of Computing Power
- How
  - Hardware Parallelism
  - Data Dependency
  - Programming Models
  - Parallel Efficiency

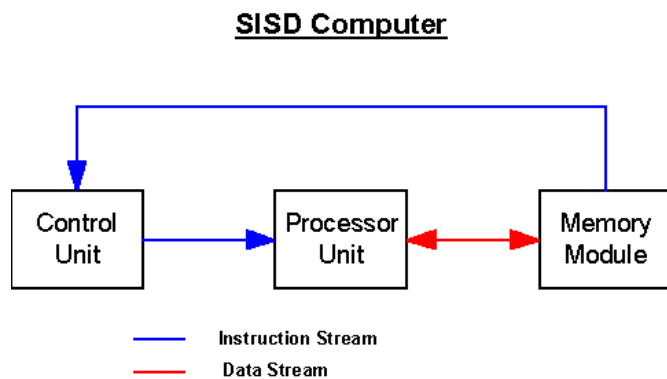
# Flynn-Taxonomie (1966)

## SISD

Single Instruction Stream

Single Data Stream

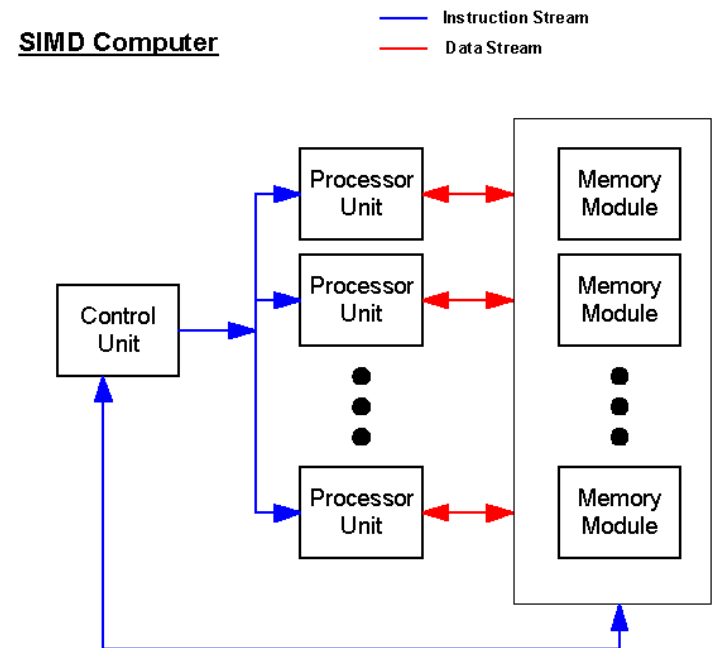
(Von Neumann architecture (1945),  
-stored program computer)



## SIMD

Single Instruction Stream

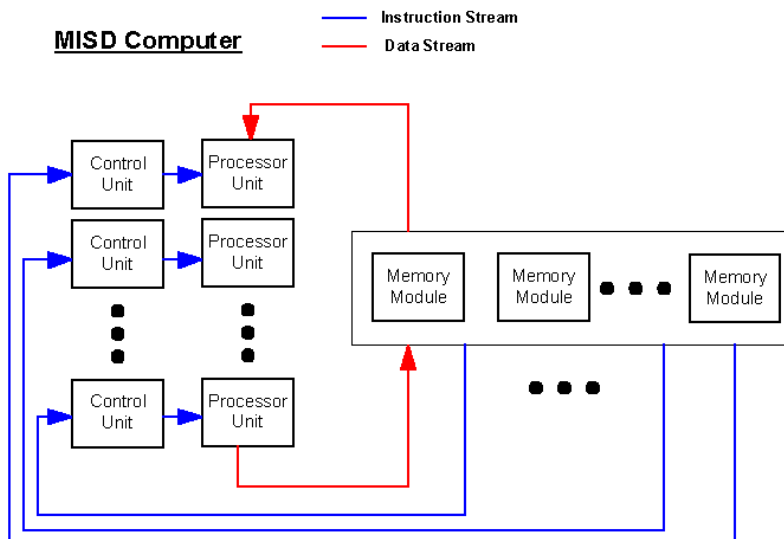
Multiple Data Streams



# Flynn-Taxonomie (1966)

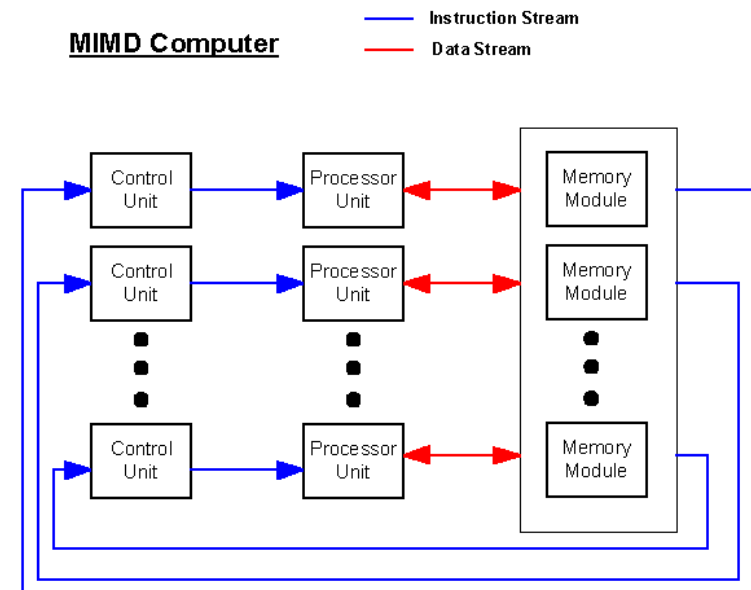
## MISD

**M**ultiple **I**nstruction Streams  
**S**ingle **D**ata Stream



## MIMD

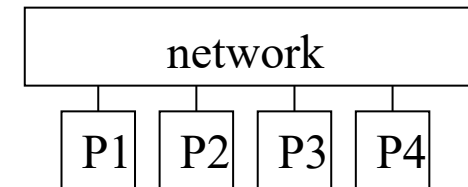
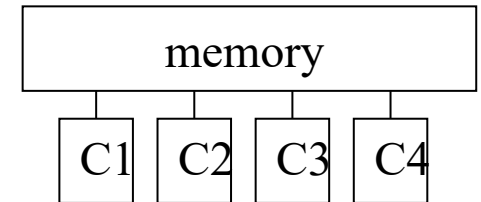
**M**ultiple **I**nstruction Streams  
**M**ultiple **D**ata Streams



# Programming Models for MIMD

Modelling the coordination of **parallel instruction streams** and **parallel data streams** has to respect the memory organization of the computing system

- **Shared Memory:**
  - Distribution of instructions to cpus
  - Synchronization of accesses to data
- **Distributed Memory:**
  - Distribution of instructions to processors,
  - Distribution of data to local memories
  - Communication of data between local memories



Two different programming models for parallel processing :

- [shared memory](#)
- [message passing](#)

Comparison to programming model for sequential processing

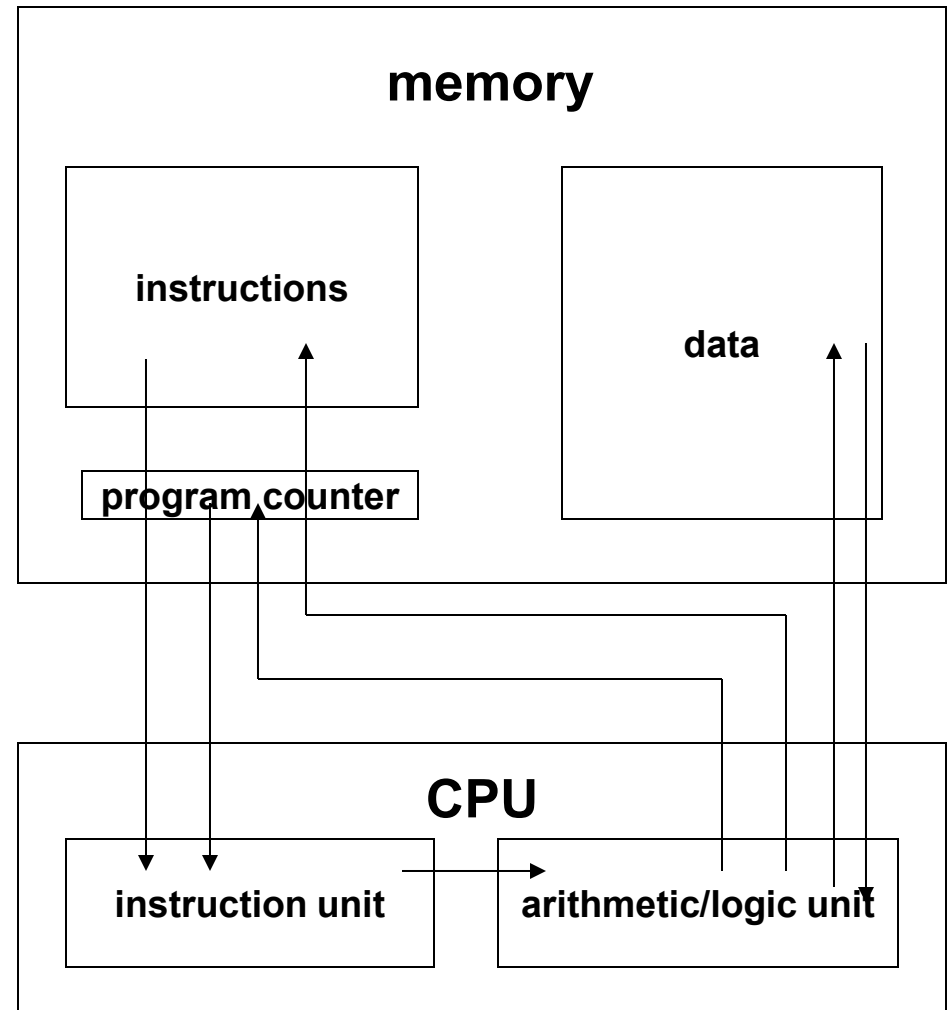
- [Von Neumann](#)

# Programming Model: Sequential (von Neumann)

objects:  
data, instructions, program counter

instructions:  
opcode op1, op2,...,re1,re2

order:  
sequential



# Programming Model: Shared Memory

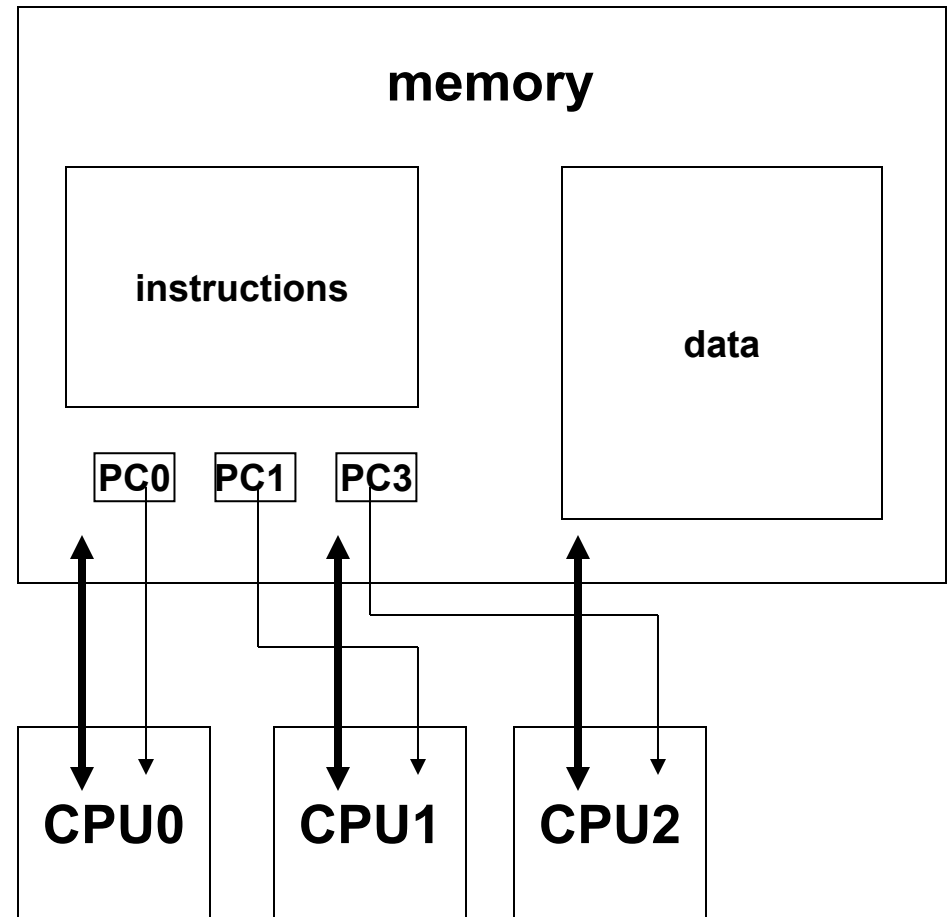
multiple instruction streams  
(threads)

objects:  
global data, instructions,  
local PCs, thread-ID

instructions:  
opcode  $op_1, op_2, \dots, re_1, re_2$   
atomic (uninterruptable) ops.

parallelism:  
mapping instructions to streams

synchronization:  
CREW (concurrent read,  
exclusive write)



# Programming Model: Message Passing

Multiple processors connected to a communication network

objects:

local data + instructions,

local program counters (pc)

unique task identification (tid)

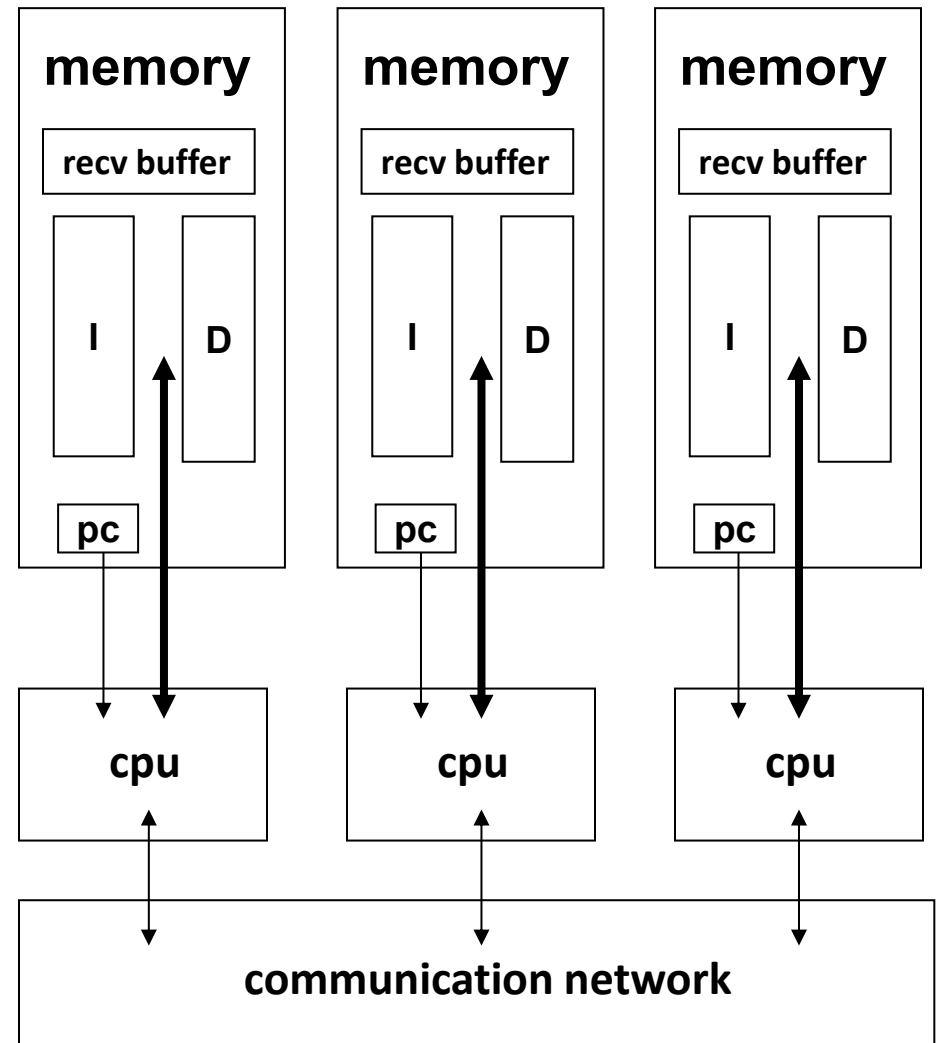
operations:

opcode (op1, op2,...,re1,re2)

send(ad,n,tid), recv(ad,n,tid)

synchronization:

recv is blocking



# Languages for Parallel Programming

Use of established programming languages to control execution within individual threads or tasks :

Fortran 77, Fortran 90, ... , C, C++, Python, ...

- **Shared Memory Programming Model:**  
creation and coordination of threads
  - POSIX Threads Library
  - OpenMP (Open Multi Processing)
- **Message Passing Programming Model:**  
creation and coordination of tasks  
communication between tasks:
  - MPI ( Message Passing Library)
- **SIMD Programming Model** (Graphics Processors):
  - CUDA (Compute Unified Device Architecture)

# Overview

- Why
  - Evolution of Computing Power
- How
  - Hardware Parallelism
  - Data Dependency
  - Programming Models
  - Parallel Efficiency

# Efficiency of Parallel Computers

Unit of computing power:

[flop/s] floating point operations per second

$$\text{maximal power} \quad r_{max} = p \cdot n_{fl} \cdot \tau^{-1}$$

$p$  number of processors (=Cores)

$n_{fl}$  number of functional units per processor

$\tau$  cycle time for one segment of pipeline

$f = \tau^{-1}$  frequency (cycle rate) of processor

GWDG's amp-Cluster (Intel Cascade Lake Platinum 9242)

$$p = 92 * 96 = 8832, n_{fl} = 32, s = 2,3 \text{ GHz}$$

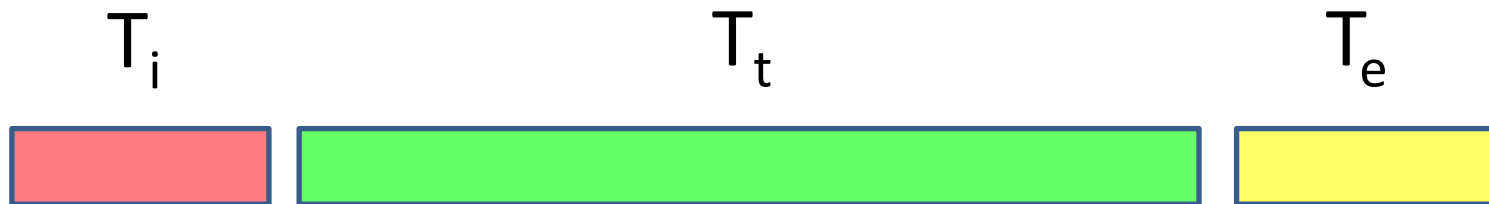
$$r_{max} = 650 \text{ Teraflop/s}$$

# Efficiency in Real Applications

- Reduction of parallel efficiency
  1. Sequential parts of application
  2. Load imbalance
  3. Communication of data
  4. Synchronization
- Reduktion of single processor efficiency
  1. Memory accesses
  2. Pipeline disruption
  3. Unused parallel pipelines

# Sequential Parts

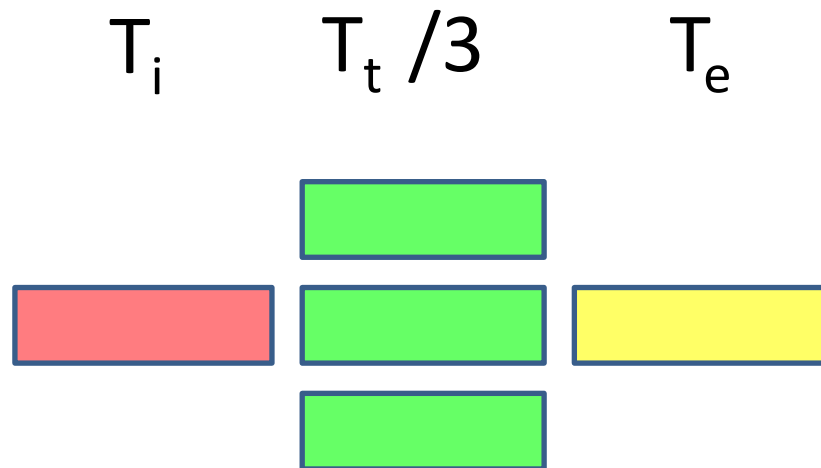
Phases of an application:    data initialization  
   data transformation  
   extraction of results



Execution time:

$$T = T_i + T_t + T_e$$

# Parallel Execution with Sequential Parts



Execution time:

$$T = T_i + T_t / 3 + T_e$$

# Amdahl's Law

$N_s$  sequential und  $N_p$  parallel ops:  $N = N_s + N_p$

Execution time on  $p$  prozessors:

$$T_p = \tau \cdot \left( N_s + \frac{N_p}{p} \right) = \tau \cdot N \left( \sigma + \frac{\pi}{p} \right)$$

Speed Up:  $S = \frac{T_1}{T_p} = p \cdot \frac{1}{1 + (p-1) \cdot \sigma} \xrightarrow{p \rightarrow \infty} \frac{1}{\sigma}$

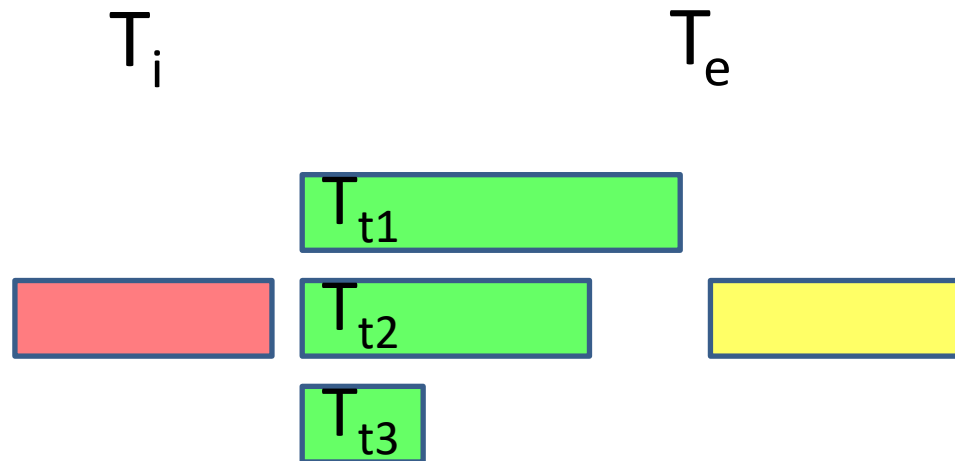
Efficiency:  $e = \frac{S}{p} = \frac{1}{1 + (p-1) \cdot \sigma}$

$\sigma_{1/2}$ : sequential part leading to 50% efficiency ( $e=0.5$ )

$$\sigma_{1/2} = \frac{1}{p-1}$$

$$\text{for } p = 8832: \quad \sigma_{1/2} \approx 0,00011$$

# Parallel Execution with Load imbalance



Execution time:

$$T = T_i + T_{t1} + T_e$$

# Generalization: Load Imbalance

process  $i, i = 1, \dots, p$  executes  $N_i$  operations,

$$N_p = \sum_{i=1}^p N_i$$

average load per processor

$$N_{av} = \frac{N_p}{p}$$

maximal deviation

$$\Delta = \max_i N_i - N_{av} = \delta \cdot N_p$$

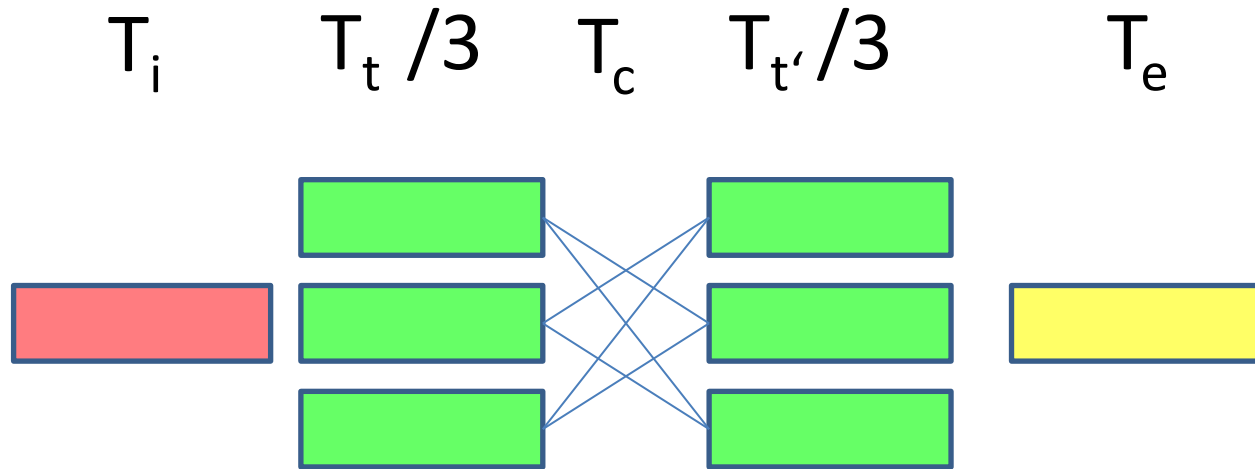
execution time

$$T_p = \tau \cdot (N_s + \max N_i) = \tau N \left( \sigma + \delta + \frac{\pi}{p} \right)$$

efficiency

$$e = \frac{T_1}{pT_p} = \frac{1}{1 + (p-1)\sigma + p\delta}$$

# Parallel Execution with Communication



Execution time:

$$T = T_i + T_e + T_c + (T_t + T_{t'})/3$$

$T_c$  may depend on the number of parallel processes

# Synchronization

Coordination of different processes:

Waiting for completion of all partial results

Sequential synchronization

$$t_{sync} = p \cdot t_{lat}$$

Cascade synchronization

$$t_{sync} = \ln(p) \cdot t_{lat}$$

Parallel granularity  $n_g$  :

number of parallel operations between synchronization points

Condition for high efficiency:  $r^{-1} \cdot n_g/p \gg t_{sync}$

# Example: Matrix-Vector-Multiplication

$$y = A \cdot x \quad y_i = \sum_j A_{ij} \cdot x_j, i = 1, \dots, n$$

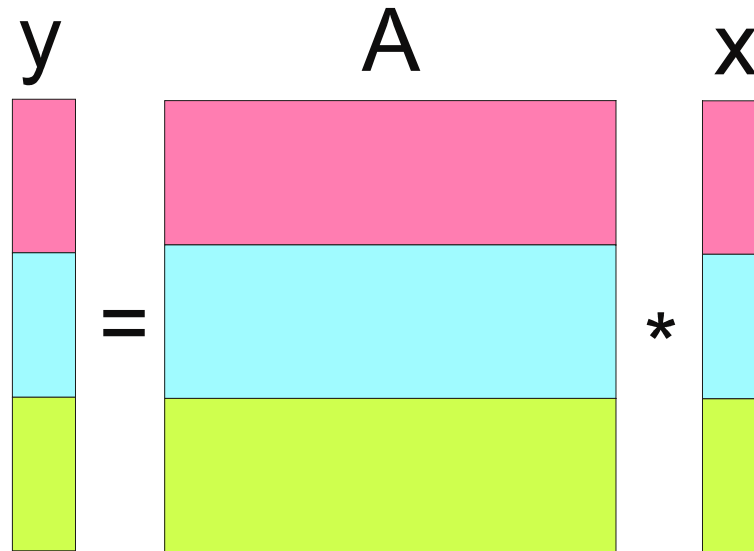
distribution of data and work:

every of  $p$  processors has

$n/p$  elements from  $x$  and  $n/p$  rows from  $A$ ,

every of  $p$  processors calculates

$n/p$  elements von  $y$



# Matrix-Vector-Multiplication

Every processor calculates  $n/p$  elements of  $y$ ,  
executing  $2 \cdot n^2/p$  operations

Every processor receives  $n/p$  elements of  $x$  from each of  
( $p - 1$ ) remaining processors

$$T_p = r^{-1} \cdot 2 \cdot \frac{n^2}{p} + (p - 1) \cdot (t_{lat} + c^{-1} \cdot n/p)$$

$$e = \frac{T_1}{p \cdot T_p} = \frac{1}{1 + p \cdot \frac{(p - 1)}{2n^2} \cdot r \cdot t_{lat} + \frac{p - 1}{2n} \cdot r/c}$$

# Matrix-Vector-Multiplication

Conditions for good efficiency:

- $p \cdot \frac{(p-1)}{2n^2} \cdot r \cdot tlat < 1$

- $\frac{p-1}{n} \cdot \frac{r}{c} < 1$

gwdg-Cluster:

- $r \cdot tlat = 1\,060\,000$

- $\frac{r}{c} = 628$

$\Rightarrow$

$$\frac{n}{p} > 728$$

$\Rightarrow$

$$\frac{n}{p} > 628$$