TECHNISCHE UNIVERSITÄT DARMSTADT

Advanced Parallel Programming

Prof. Dr. Felix Wolf

FUNDAMENTALS

Outline

- Performance metrics
- Locality
- Amdahl's law
- Speedup & parallel efficiency
- Scaling
- Law of Gustafson
- Abstract models of parallel machines
- Asymptotic complexity
- Processes & threads

Primary performance metrics

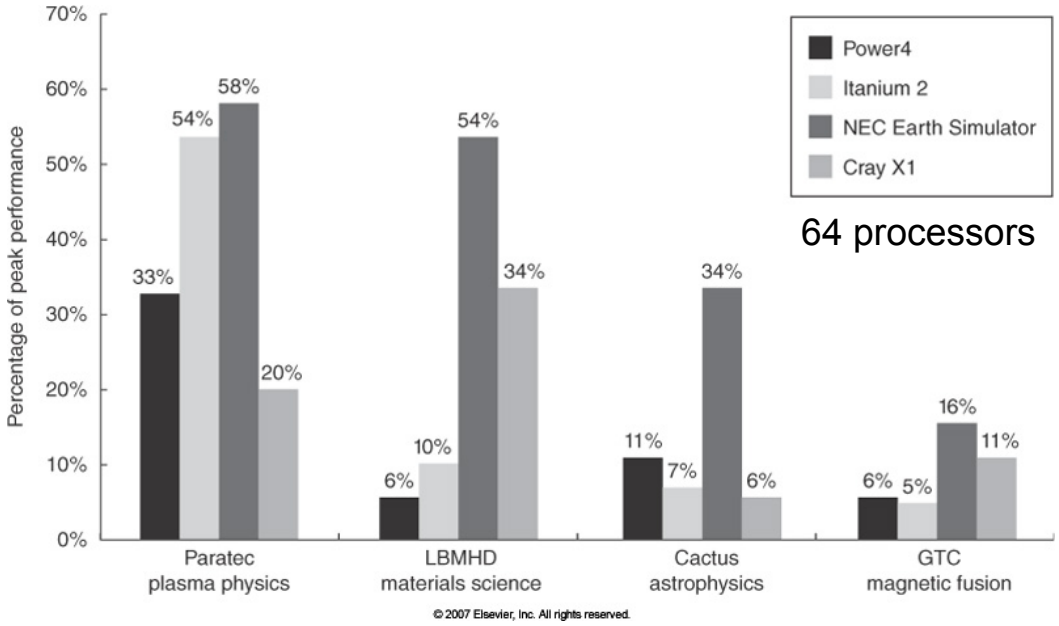
TECHNISCHE UNIVERSITÄT DARMSTADT

- Response time, execution time
  - Time between start and completion of an event or program
- Throughput
  - Total amount of work done in a given time
- Energy (to solution)

- Performance = $\frac{1}{\text{Resources to solution}}$

Peak performance

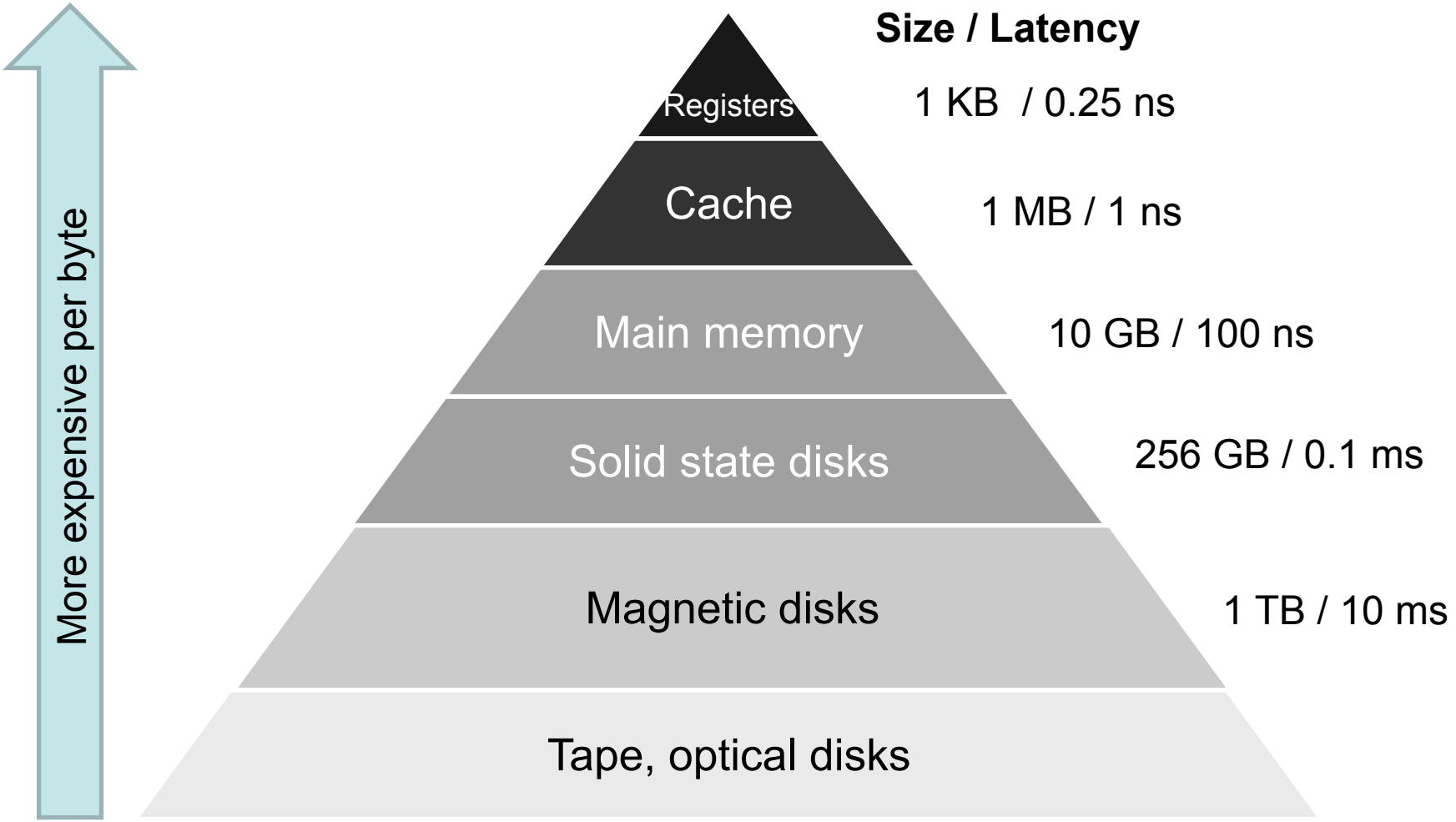
- Peak performance is the performance a computer is guaranteed not to exceed



Source: Hennessy, Patterson: Computer Architecture, 4th edition, Morgan Kaufmann

Memory hierarchy

TECHNISCHE UNIVERSITÄT DARMSTADT

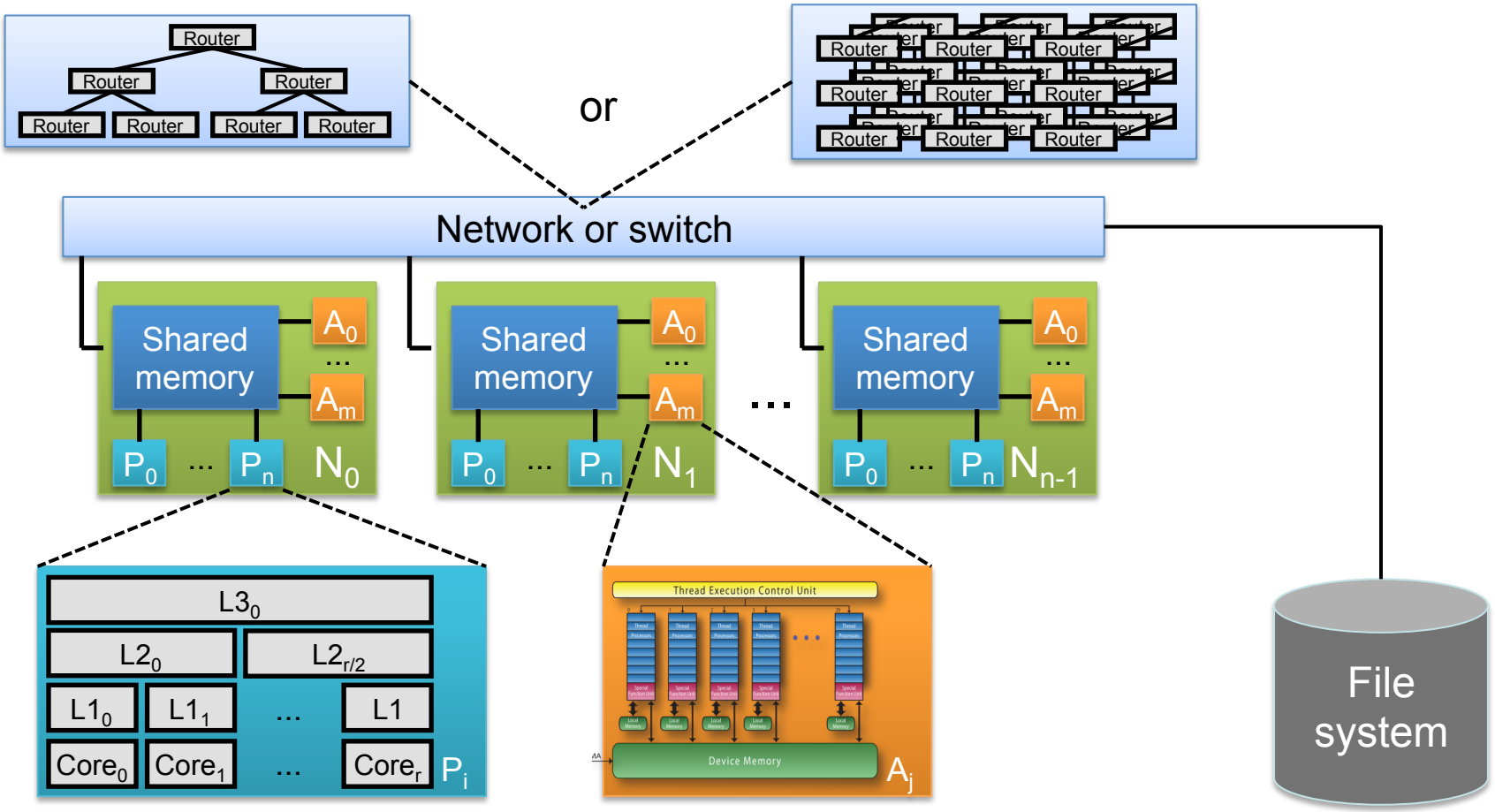


Locality

- Programs tend to reuse data and instructions they have used recently
  - Allows to estimate what instructions and data a program will use in the near future
- A program typically spends 90% of the time in 10% of its code
- Locality also applies to data accesses, but usually not as much as to code accesses

Temporal locality	Spatial locality
Recently accessed items are likely to be accessed in the near future	Items with addresses close to each other tend to be referenced close together in time

Typical supercomputer architecture



Amdahl's law

TECHNISCHE UNIVERSITÄT DARMSTADT

- The improvement in execution time to be gained from using some faster mode of execution is limited by the fraction of the time that faster mode can be used
- Speedup = $\dfrac{\text{Execution time for the entire task without using the enhancement}}{\text{Execution time for the entire task using the enhancement when possible}}$
- Depends on two factors
  - Fraction of the original execution that can benefit from the enhancement
  - Improvement gained through enhancement during that fraction

Amdahl's law (2)

$$\text{Speedup}_{\text{overall}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}} < \frac{1}{(1 - \text{Fraction}_{\text{enhanced}})}$$

Example

- Function foo() of a program takes 20 % of the overall time
- How is the speedup if the time needed for foo() can be halved?

$$\text{Speedup}_{\text{overall}} = \frac{1}{(1 - 0.2) + \frac{0.2}{2}} = \frac{10}{9}$$

- How is the speedup if the time needed for foo() can almost be eliminated?

$$\text{Speedup}_{\text{overall}} = \frac{1}{(1 - 0.2) + \frac{0.2}{\infty}} = \frac{10}{8}$$

Parallelism

TECHNISCHE UNIVERSITÄT DARMSTADT

System / application	• Server throughput can be improved by spreading workload across multiple processors or disks • Ability to add memory, processors, and disks is called scalability
Processor	• Instruction-level parallelism (e.g., pipelining) • Depends on the fact that many instructions do not depend on the results of their predecessor
Detailed digital design	• Set-associative caches use multiple banks of memory • Carry-lookahead or prefix adder • Parallelism exploited to speed up the calculation of sums from linear to logarithmic in the number of bits

Amdahl's law for parallelism

TECHNISCHE UNIVERSITÄT DARMSTADT

- Assumption – program can be parallelized on p processors except for a sequential fraction f with

$$0 \le f \le 1$$

$$\text{Speedup}_{\text{overall}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}} < \frac{1}{(1 - \text{Fraction}_{\text{enhanced}})}$$

$$Speedup(p) = \frac{1}{(1-(1-f)) + \frac{1-f}{p}} = \frac{1}{f + \frac{1-f}{p}} < \frac{1}{f}$$

- Speedup limited by sequential fraction

Available parallelism

TECHNISCHE UNIVERSITÄT DARMSTADT

Amdahl's Law

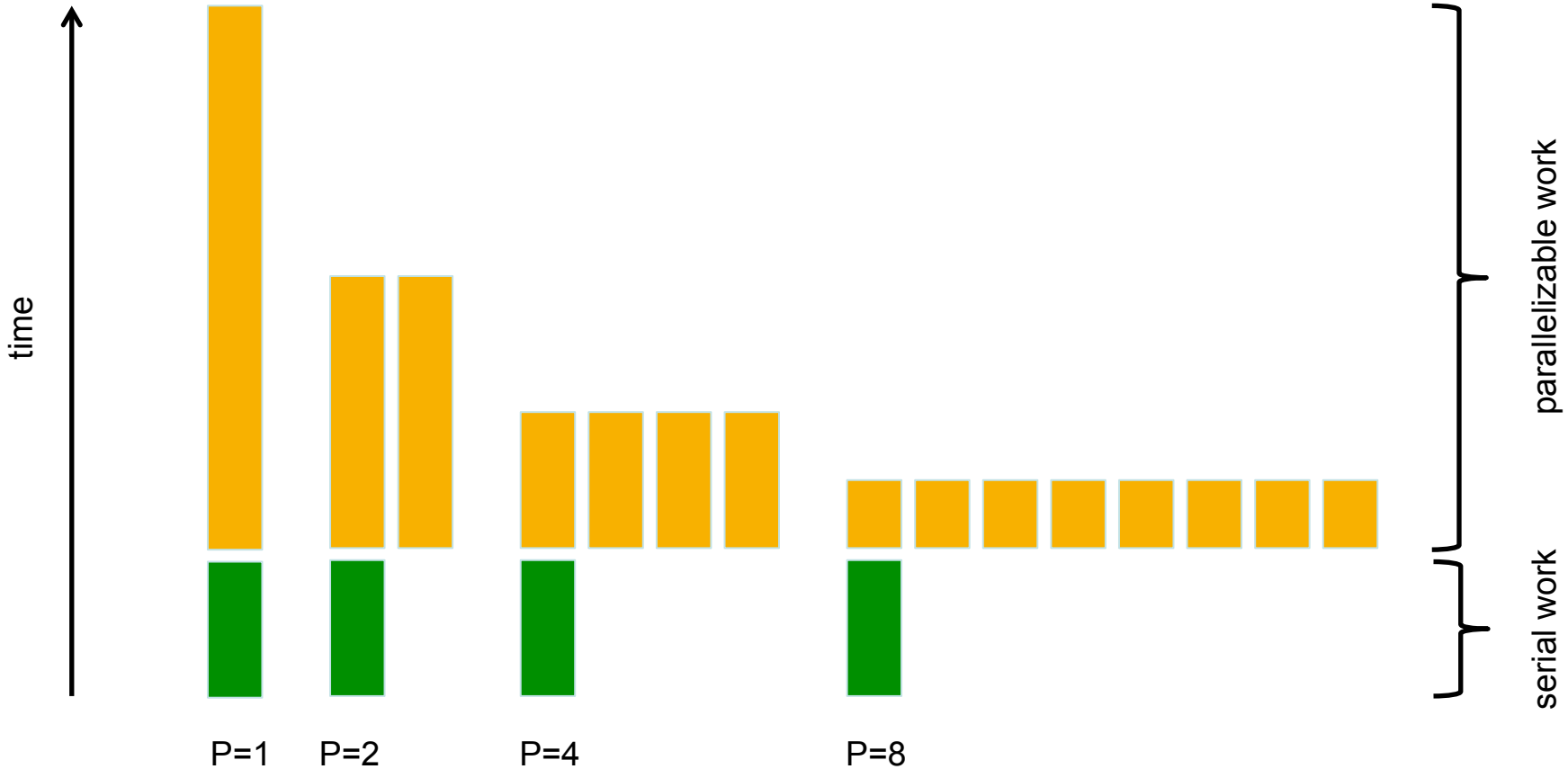
$$Speedup(p) = \frac{1}{f + \frac{1-f}{p}}$$

Overall speedup of 80 on 100 processors

$$80 = \frac{1}{f + \frac{1-f}{100}}$$

$$f = 0.0025$$

Amdahl's law (3)



Parallel efficiency

TECHNISCHE UNIVERSITÄT DARMSTADT

$$\text{Efficiency}(p) = \frac{\text{Speedup}(p)}{p}$$

- Metric for cost of parallelization (e.g., communication)
- Without super-linear speedup

$$\text{Efficiency}(p) \leq 1$$

Super-linear speedup possible

- Critical data structures may fit into the aggregate cache

Law of Gustafson

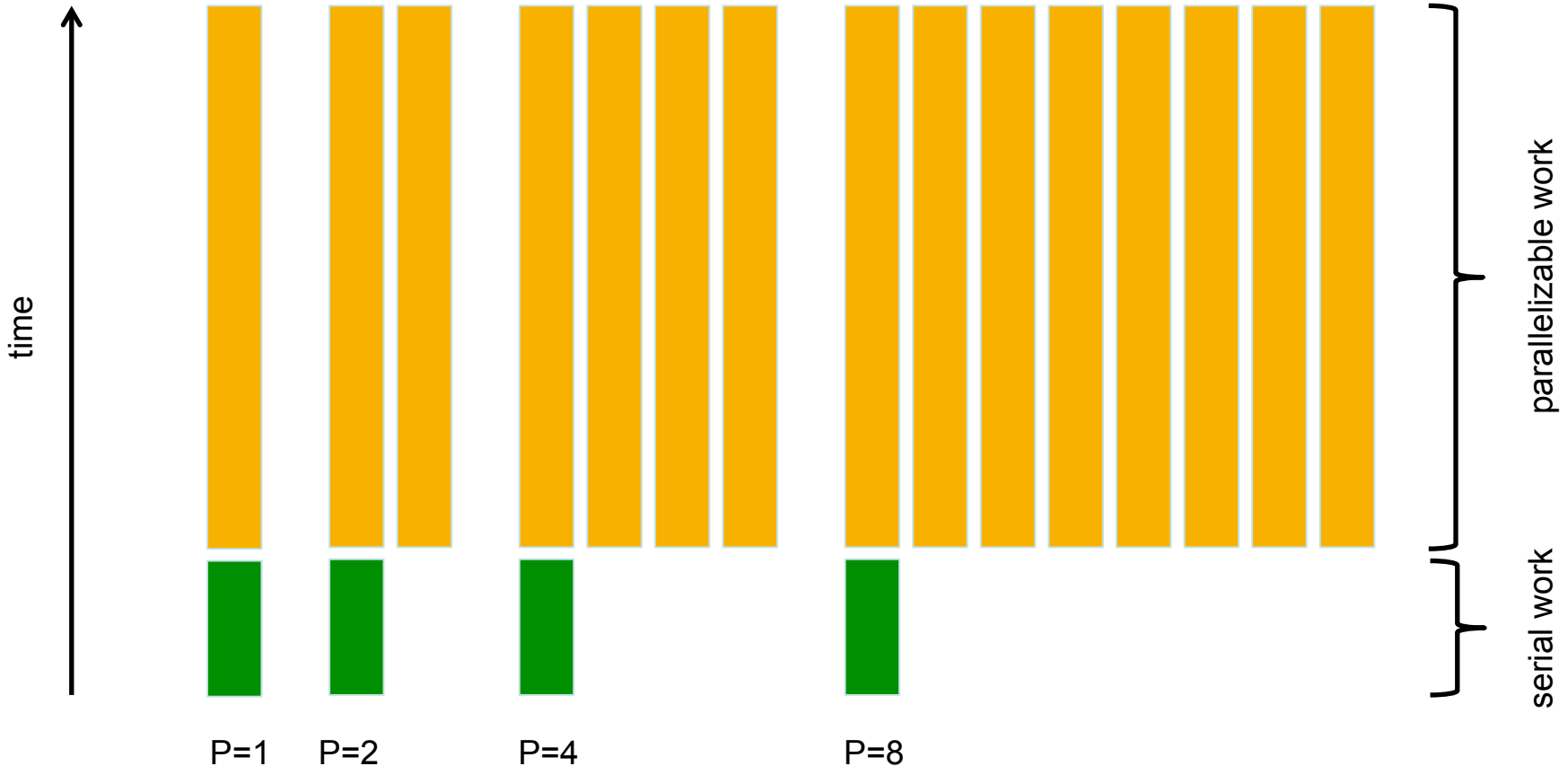
TECHNISCHE UNIVERSITÄT DARMSTADT

- Amdahl's Law ignores increasing problem size
  - Parallelism often applied to calculate bigger problems instead of calculating a given problem faster
- Fraction of sequential part may be function of problem size
- Assumption
  - Sequential part has constant runtime τ_f
  - Parallel part has runtime $\tau_v(n,p)$
- Speedup

If parallel part can be perfectly parallelized

$$\text{Speedup}(n,p) = \frac{\tau_f + \tau_v(n,1)}{\tau_f + \tau_v(n,p)}$$

Law of Gustafson (2)



Scalability

TECHNISCHE UNIVERSITÄT DARMSTADT

Weak scaling

- Ability to solve a larger input problem by using more resources (here: processors)
- Problem size per processor remains constant
- Example: larger domain, more particles, higher resolution

Strong scaling

- Ability to solve the same input problem faster as more resources are used
- Usually more challenging
- Limited by Amdahl's law and communication demand

Bandwidth and latency

TECHNISCHE UNIVERSITÄT DARMSTADT

Bandwidth or throughput

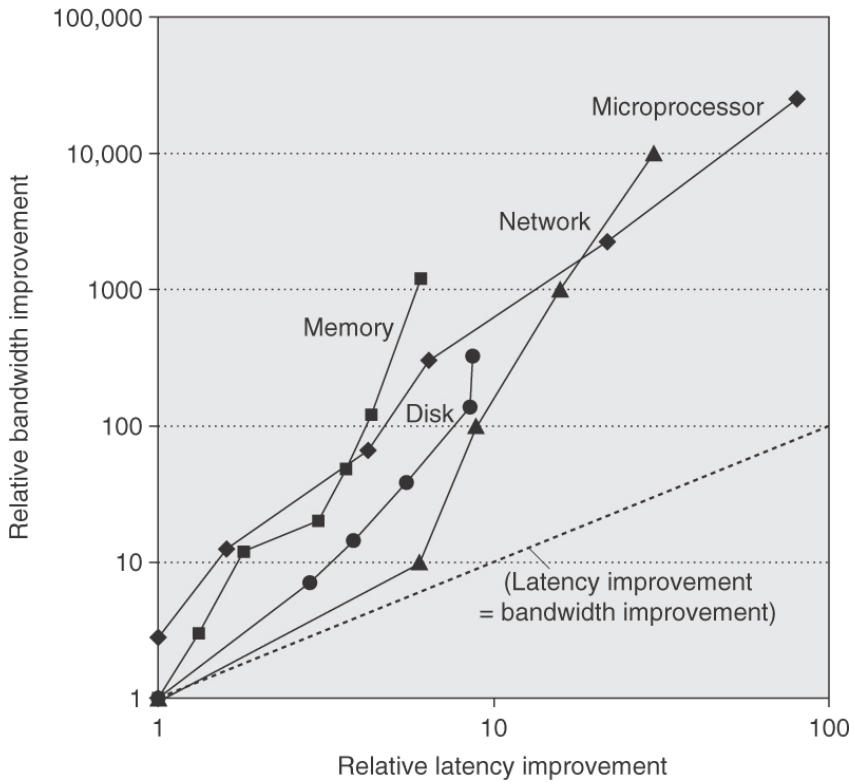
- Total amount of work done in a given time (e.g., disk transfer rate)

Latency or response time

- Time between start and completion of an event (e.g., disk access time)

Rule of thumb

- Bandwidth grows by at least the square of the improvement in latency



Source: Hennessy, Patterson: Computer Architecture, 5th edition, Morgan Kaufmann

Abstract models of parallel machines

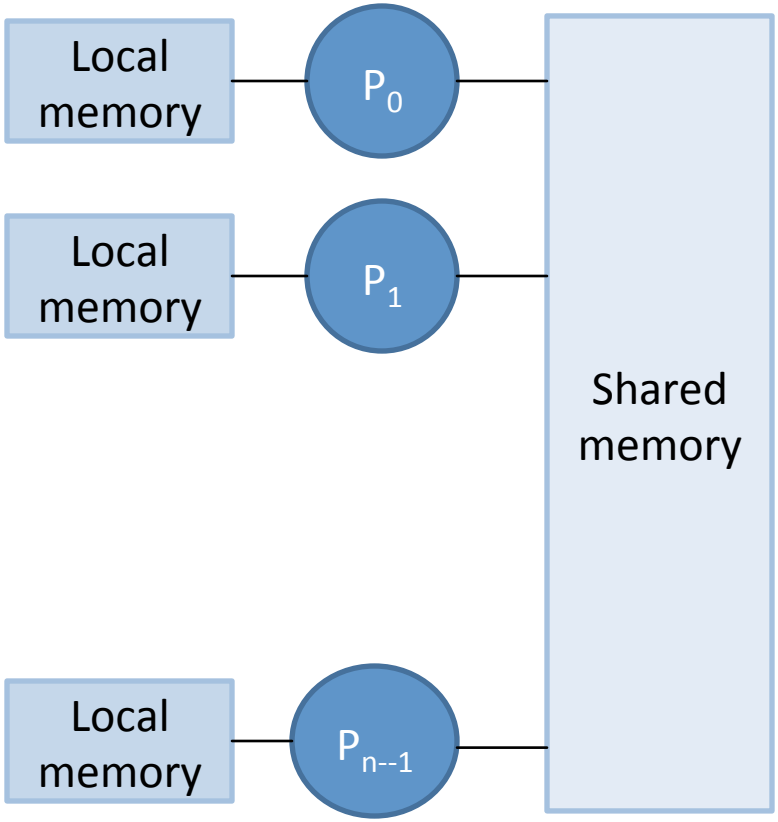
TECHNISCHE UNIVERSITÄT DARMSTADT

- Used by parallel algorithms designers to estimate algorithm performance independent of specific
  - Hardware
  - Programming language
- Define on abstract level
  - Basic operations
  - When the corresponding actions take place
  - How to access data
  - How to store data
- Objective
  - Abstract from unnecessary details
  - But cover essential characteristics of a broad class of systems

Parallel Random Access Machine (PRAM)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Any (problem-size dependent) number of processors
- Access to shared memory in unit time
- Input and output stored in shared memory
- Processors execute instructions synchronously
- Cost of an algorithm specified in terms of the number of required PRAM instructions
  - Usually asymptotic behavior as function of problem size
  - Example: $O(n^2)$



Parallel Random Access Machine (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Suitable for the theoretical analysis of algorithms
- Impractical for prediction of actual runtimes on given machine because of too many simplifying assumptions
  - No limit on the number of processors
  - Any memory location uniformly accessible from any processor
  - There is no limit on the amount of shared memory
  - Resource contention is absent

LogP

TECHNISCHE UNIVERSITÄT DARMSTADT

- Alternative machine model for parallel computation
  - Arbitrary number of processing units with distributed memory
  - Processing units are connected through an abstract communication medium which allows point-to-point communication
  - Still simple enough to allow the creation of performance models
- Defined by four parameters

L = latency of the communication medium

o = overhead of sending and receiving a message

g = the gap required between two send/receive operations

P = the number of processing units

Asymptotic complexity

TECHNISCHE UNIVERSITÄT DARMSTADT

- Example: dot product of two vectors of length $N \geq P$
  - Split vector into pieces of length N/P
  - Calculate subproducts in parallel
  - Calculate global sum in tree-like fashion

$$\begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} * \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

- Asymptotic running time

$$T(N,P) = \Theta(N/P + \lg P)$$

Asymptotic complexity notation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Big **O** notation – denotes a set of functions with an upper bound. *O(f(N))* = set of all functions *g(N)* such that there exists positive constants *c*, N_0 with $|g(N)| \leq c*f(N)$ for $N \geq N_0$.

- Big **Omega** notation – denotes a set of functions with a lower bound. *Ω(f(N))* = set of all functions *g(N)* such that there exists positive constants *c*, N_0 with $|g(N)| \geq c*f(N)$ for $N \geq N_0$.

- Big **Teta** notation – denotes a set of functions with both lower and upper bounds. *θ(f(N))* = set of all functions *g(N)* such that there exists positive constants c_1, c_2, N_0 with $c_1*f(N) \leq |g(N)| \leq c_2*f(N)$ for $N \geq N_0$.

Asymptotic speedup

TECHNISCHE UNIVERSITÄT DARMSTADT

- Asymptotic speedup

$$\frac{T_1}{T_P} = \frac{\Theta(N)}{\Theta(N/P + \lg P)} = \Theta\left(\frac{N}{N/P + \lg P}\right)$$

- Asymptotic efficiency

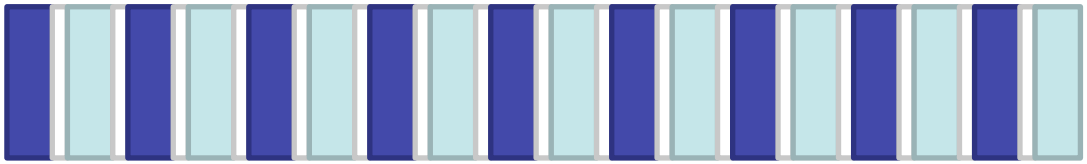
$$\frac{T_1}{P \cdot T_P} = \Theta\left(\frac{N}{N + P \lg P}\right)$$

Time sharing

TECHNISCHE UNIVERSITÄT DARMSTADT

- Also called multitasking
- Logical extension of multiprogramming
- CPU executes multiple processes by switching among them
- Switches occur frequently enough so that users can interact with each program while it is running
- On a multiprocessor, processes can also run truly concurrently, taking advantage of additional processors

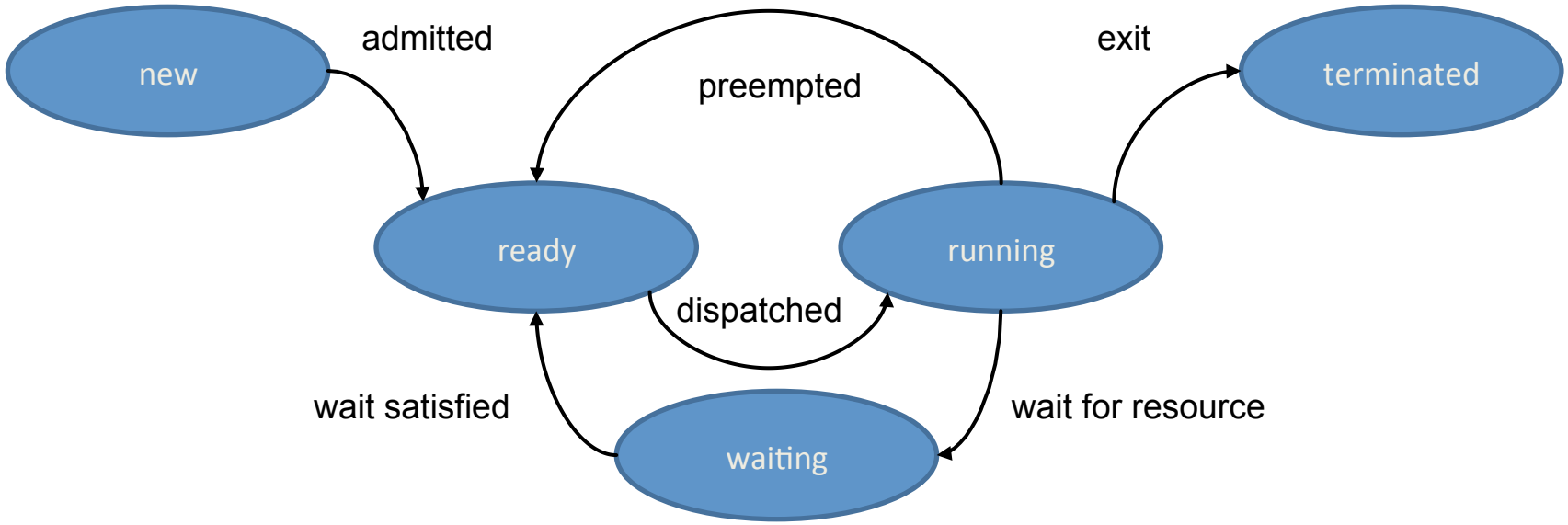
Single core



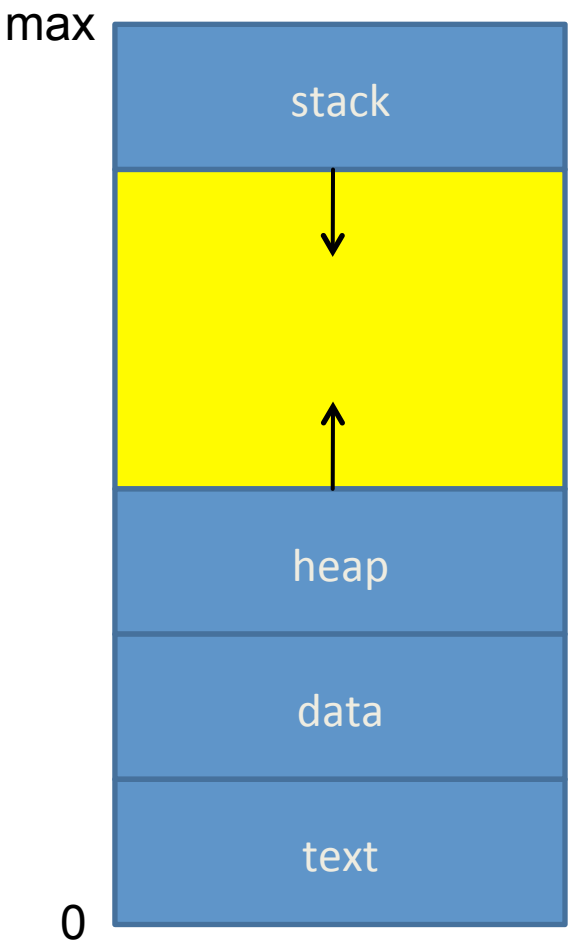
Process

TECHNISCHE UNIVERSITÄT DARMSTADT

- A process is a program in execution
- States of a process



Processes in memory



Temporary data: function parameters, return addresses, local variables

Dynamically allocated memory

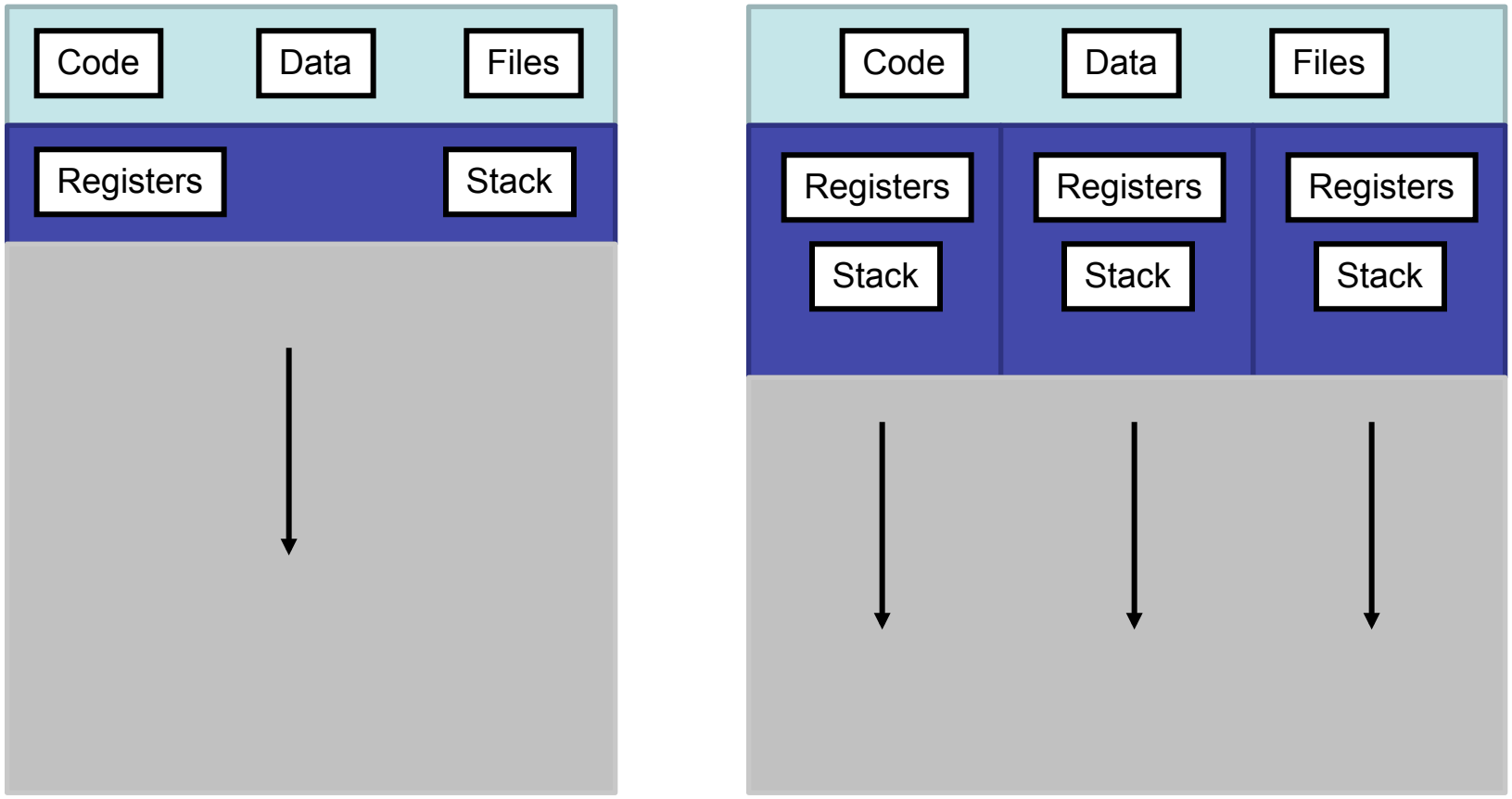
Global variables

Program code

Thread

- Basic unit of CPU utilization
  - Flow of control within a process
- A thread includes
  - Thread ID
  - Program counter
  - Register set
  - Stack
- Shares resources with other threads belonging to the same process
  - Text (i.e., code) section
  - Data section (i.e., address space)
  - Other operating system resources
  - E.g., open files, signals

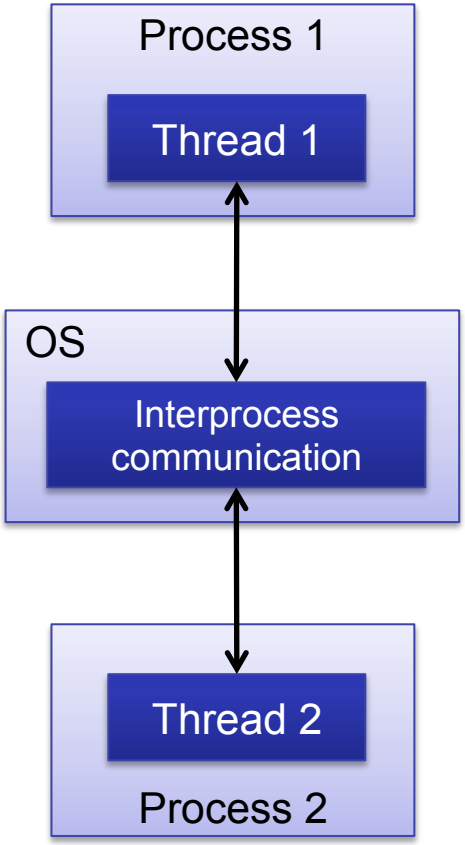
Single-threaded vs. multi-threaded

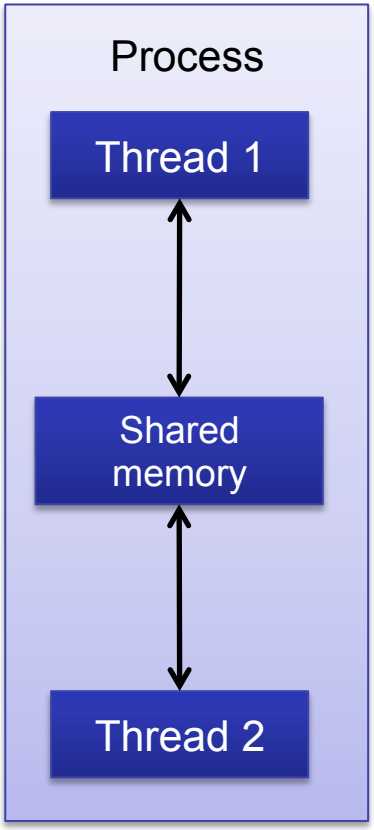


Single-threaded

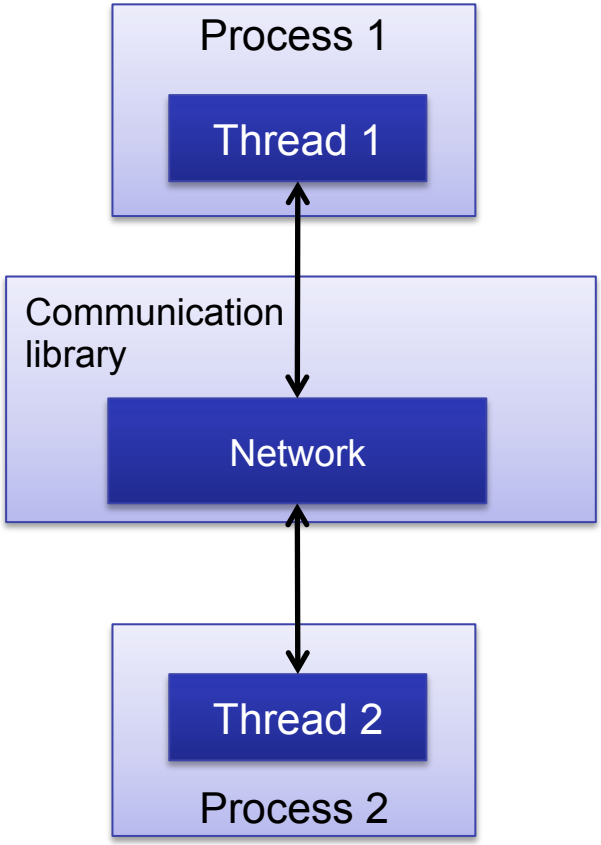
Multi-threaded

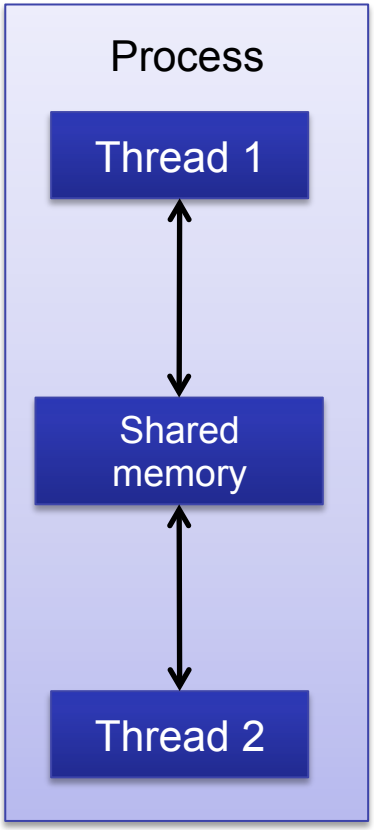
Concurrency – processes vs. threads





Concurrency – processes vs. threads (2)





Concurrency – processes vs. threads (3)

TECHNISCHE UNIVERSITÄT DARMSTADT

Processes

- Communication explicit
- Often requires replication of data
- Address spaces protected
- Parallelization usually implies profound redesign
- Writing debuggers is harder
- More scalable

Threads

- Convenient communication via shared variables
- More space efficient - sharing of code and data
- Context switch cheaper
- Incremental parallelization easier
- Harder to debug – race conditions

Summary

TECHNISCHE UNIVERSITÄT DARMSTADT

- Peak performance usually never reached
- A memory hierarchy assumes locality of accesses
- Amdahl's law applies to strong scaling
- Law of Gustavson takes weak scaling into account
- LogP abstract machine model designed for distributed memory
- Asymptotic complexity is an effective instrument to reason about an algorithm's scalability