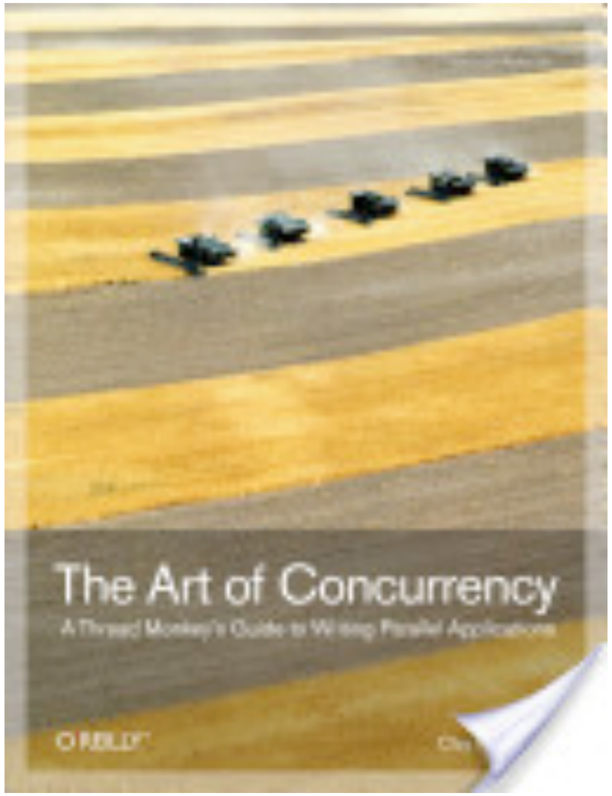
Parallel Programming Concepts

Introduction

Peter Tröger

Course Design

- Lectures covering theoretical and practical aspects of concurrency and parallelism
  - 30 minutes oral exam
  - Lectures partially given by domain experts from OSM group
- 3 big assignments
  - 2/3 must be solved correctly
  - Implementation of parallel algorithms with different programming models
- Literature list on course home page
- Good book for starters ...



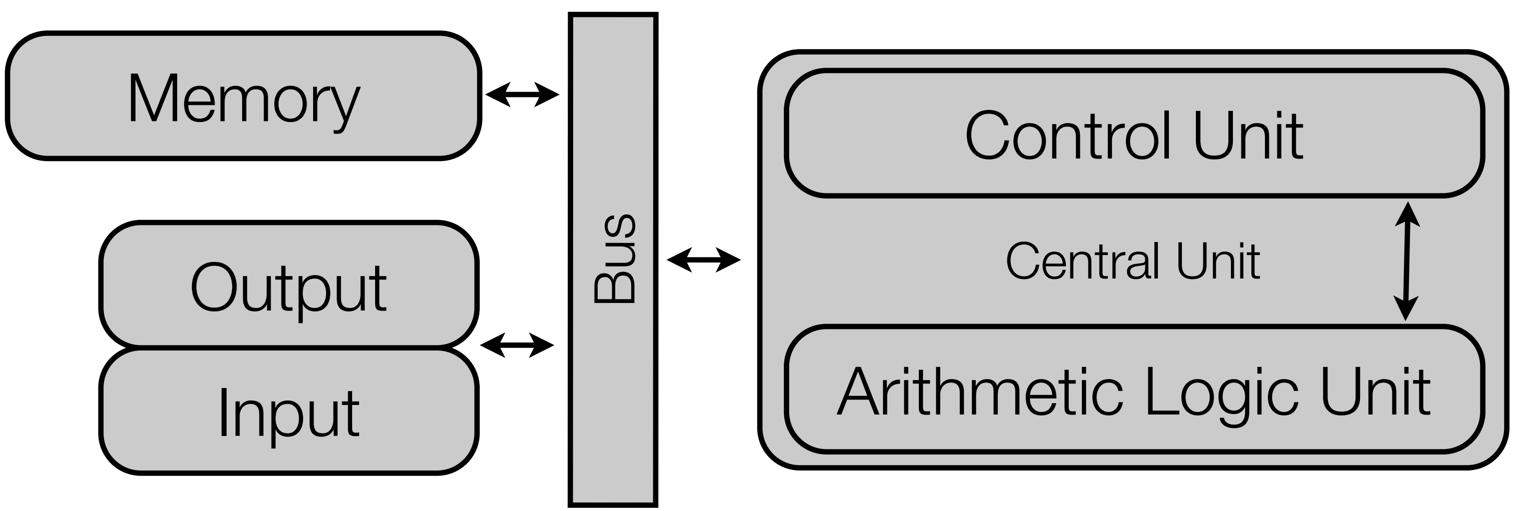
The Art of Concurrency:
A Thread Monkey's Guide to Writing Parallel Applications
Clay Breshears
O'Reilly Media, Inc., 2009

Computer Markets

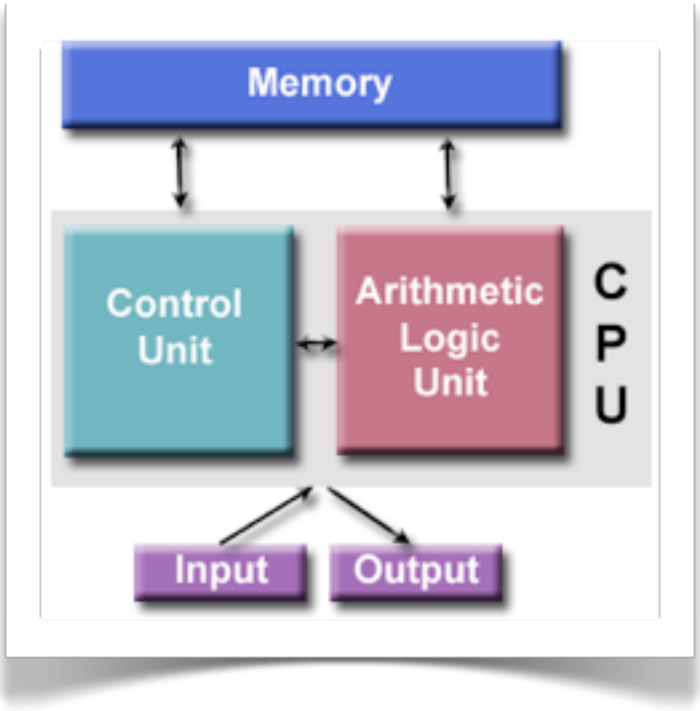
- Embedded Computing
  - Real-time systems, nearly everywhere
  - Power consumption and price as major issue
- Desktop Computing
  - Home computers
  - Best-possible performance / price ratio as major issue
- Servers
  - Performance and availability of provided business service as major issue
  - Web servers, banking back-end, order processing, ...

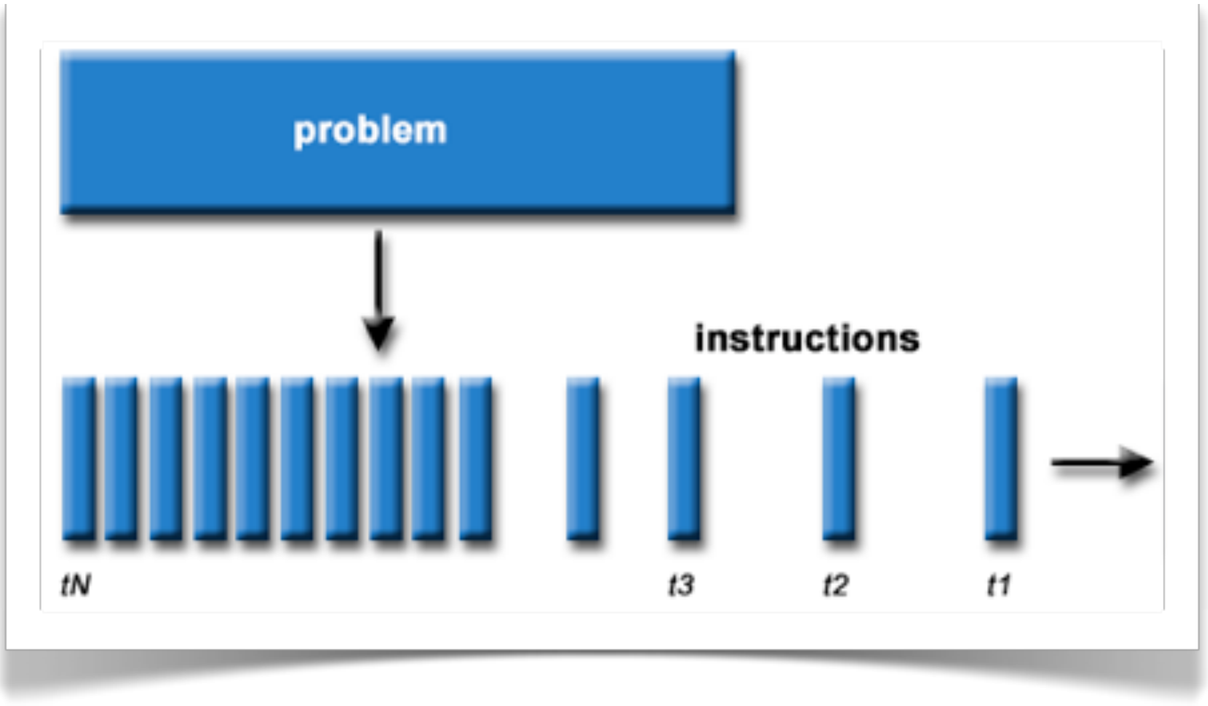
Machine Model

- First computers had fixed programs (electronic calculator)
- *von Neumann architecture* (1945, for EDVAC project)
  - Instruction set used for assembling programs stored in memory
  - Program is treated as data, which allows program exchange under program control and self-modification
  - von Neumann bottleneck



Machine Model





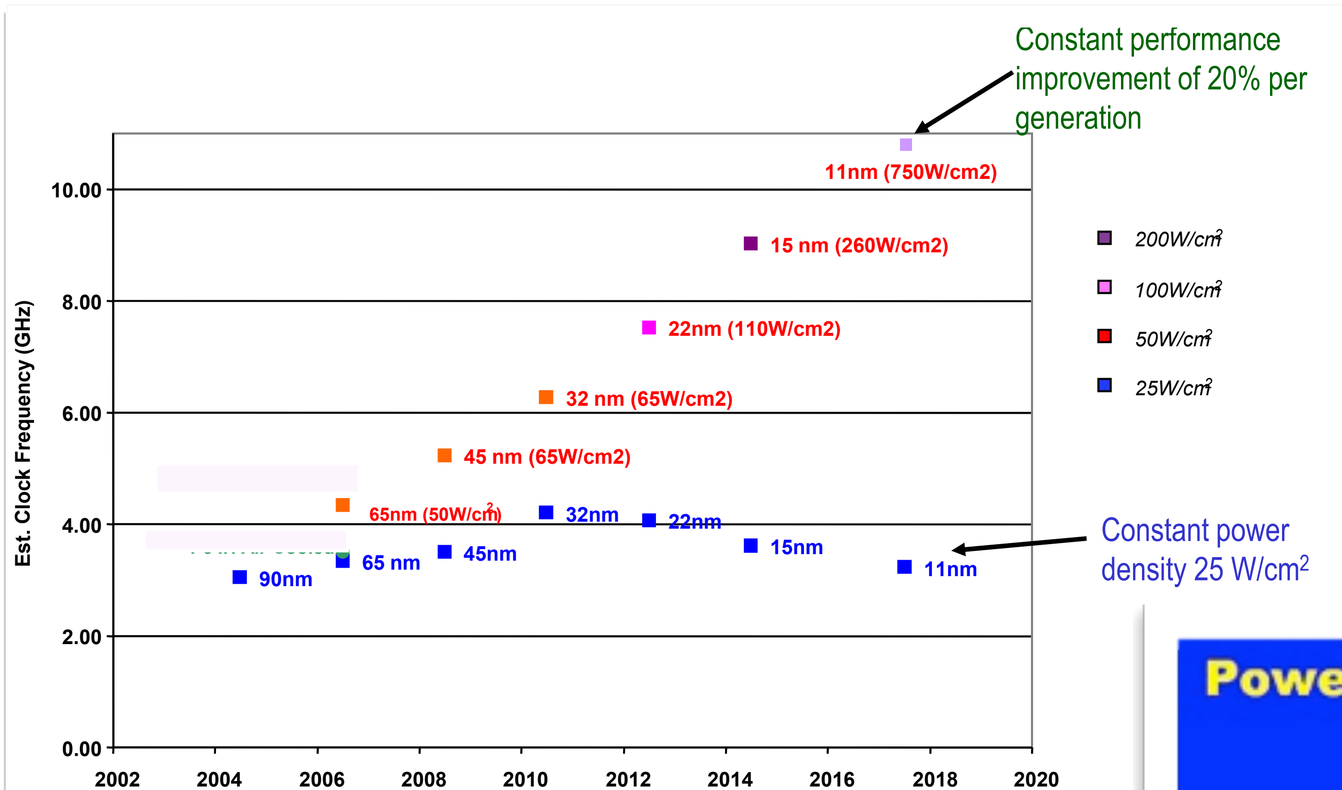
Three ways of doing anything faster (Pfister)

- **Work harder**
- Work smarter
- Get help

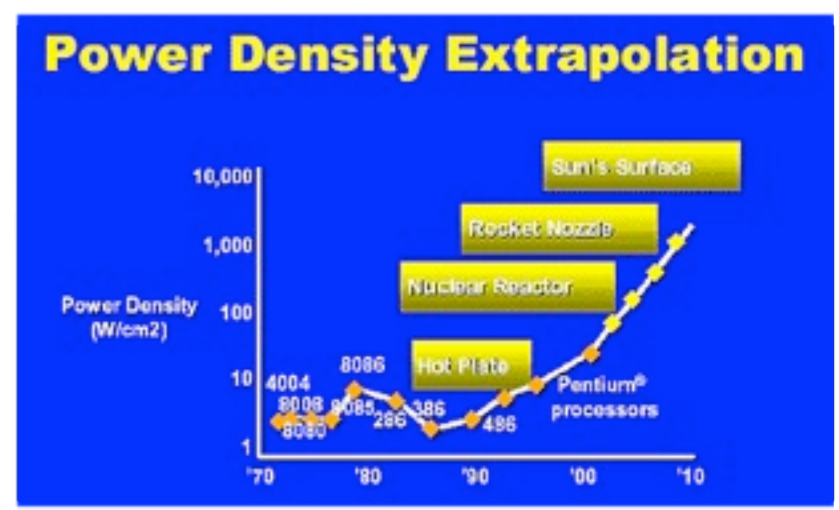
Work Harder

- *„...the number of transistors that can be inexpensively placed on an integrated circuit is increasing exponentially, doubling approximately every two years. ...“ (Moore's Law)*
  - Rule of exponential growth is applied to many IT hardware developments
  - Density rule is sometimes applied on system performance
- *„Andy giveth, and Bill taketh away.“*
- Traditional ways for making processors faster:
  - Clock speed - More cycles per time unit
  - Execution optimization - More work per cycle
  - Caching - Tackle the memory hierarchy

Power per Core [Frank & Tyberg]



- Clock speed increase is no longer an option
- More transistors at the same speed
- For some time, bigger caches was the answer



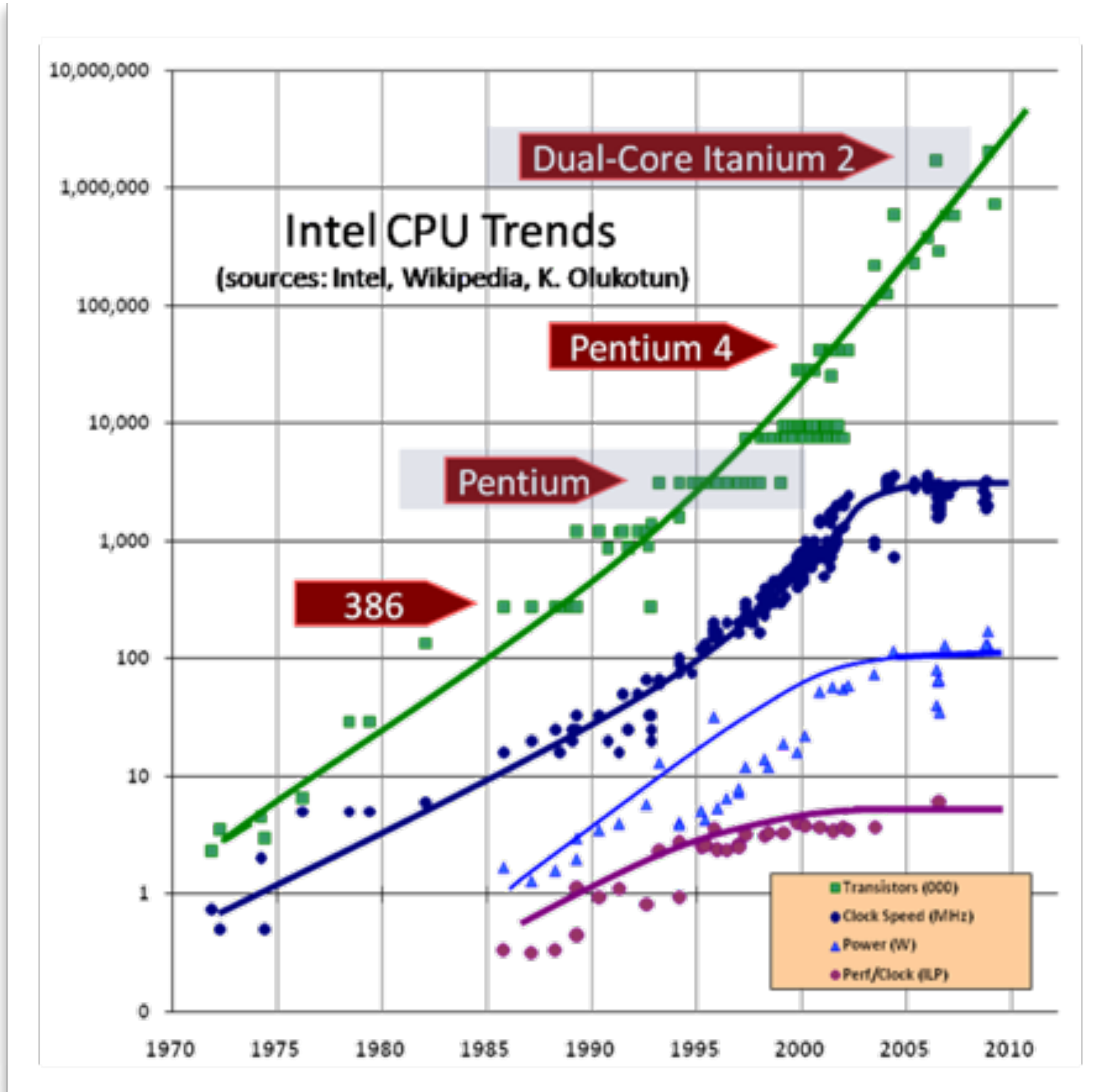
Memory Hierarchy

(C) Chevance, approx. values in 2005

Technology	Access Time	Human Scale	Capacity	Price
Processor Register	100 ps	0.1 s	64x64 Bits	part of CPU
Processor Cache	L1: ~1 ns L2-L3: 4-16 ms	16 s	kB - MB	part of CPU
RAM	~150 ns	~ 25 min	>= 1 GB	~0.1 $/MB
Disk	~6 ms	~700 days	> 70 GB /disk	~0.005 $/MB
Tape Robot	~10 s	~3200 years	~100 GB / tape	<0.001 $/MB

The Free Lunch Is Over

- Clock speed curve flattened in 2003
  - Heat
  - Power consumption
  - Leakage
- 2 GHz since 2001 (!)
- ‚Work Harder‘ no longer works
- We stumbled into the **Many-Core Era**



(C) Herb Sutter, 2009

Conventional Wisdoms Replaced

Old Wisdom	New Wisdom
Power is free, transistors are expensive	„Power wall“
Only dynamic power counts	Static leakage makes 40% of power
Multiply is slow, load-and-store is fast	„Memory wall“
Instruction-level parallelism gets constantly better via compilers and architectures	„ILP wall“
Parallelization is not worth the effort, wait for the faster uniprocessor	Performance doubling might now take 5 years due to physical limits
Processor performance improvement by increased clock frequency	Processor performance improvement by increased parallelism

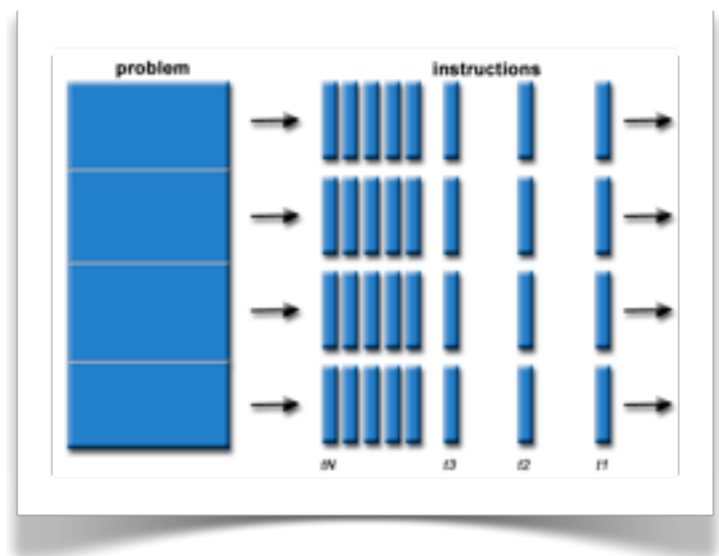
(C) Asanovic et al., Berkeley Technical Report EECS-2006-183

Three ways of doing anything faster (Pfister)

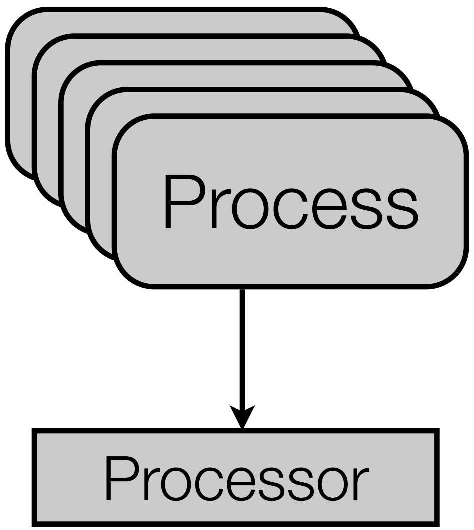
- Work harder
- Work smarter
- **Get help**

Getting Help

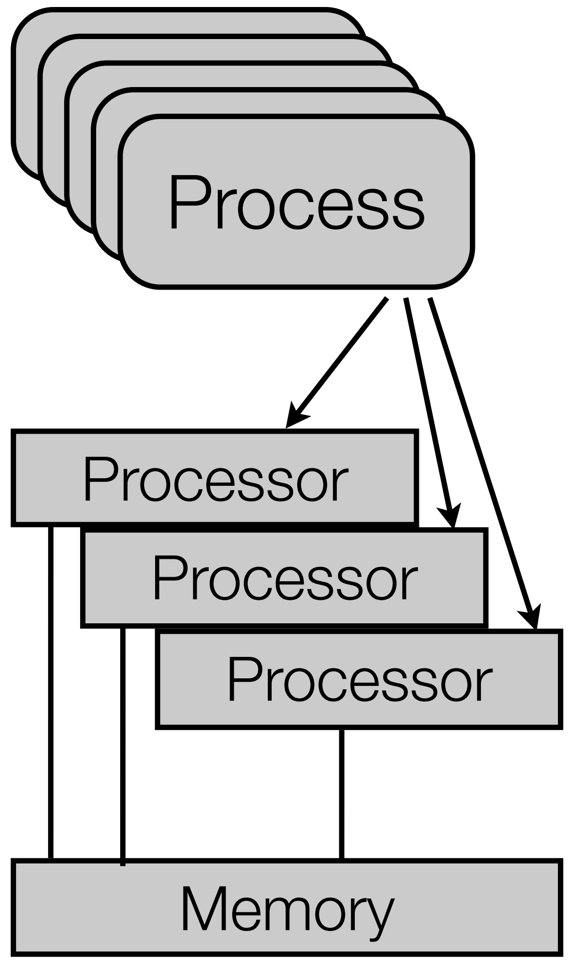
- „A parallel computer is a set of processors that are able to work cooperatively to solve a computational problem.“ (Foster 1995)
- Typical solution not only in computer science
  - Building construction, car manufacturing, every larger company
- Some problems always benefit from faster processing
  - Simulation and modeling (climate, earthquakes, airplane design, ...), Data mining, transaction processing
- Sequential code is history
  - Easy to understand, huge variety of programming languages - and now ?

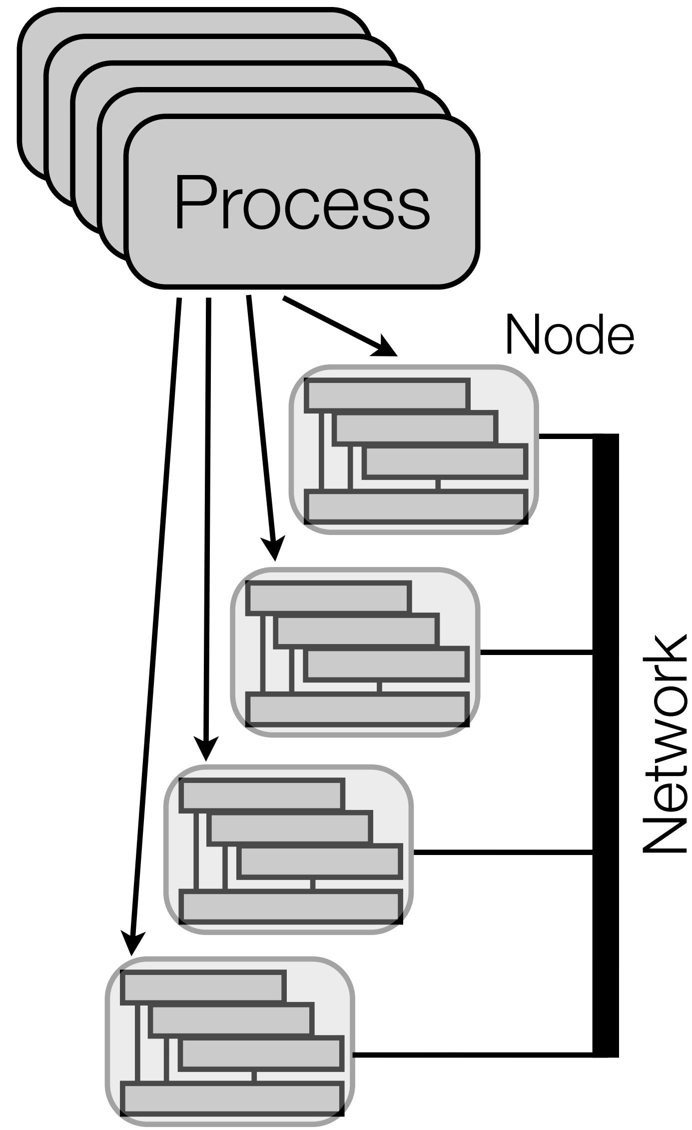


Parallel Hardware

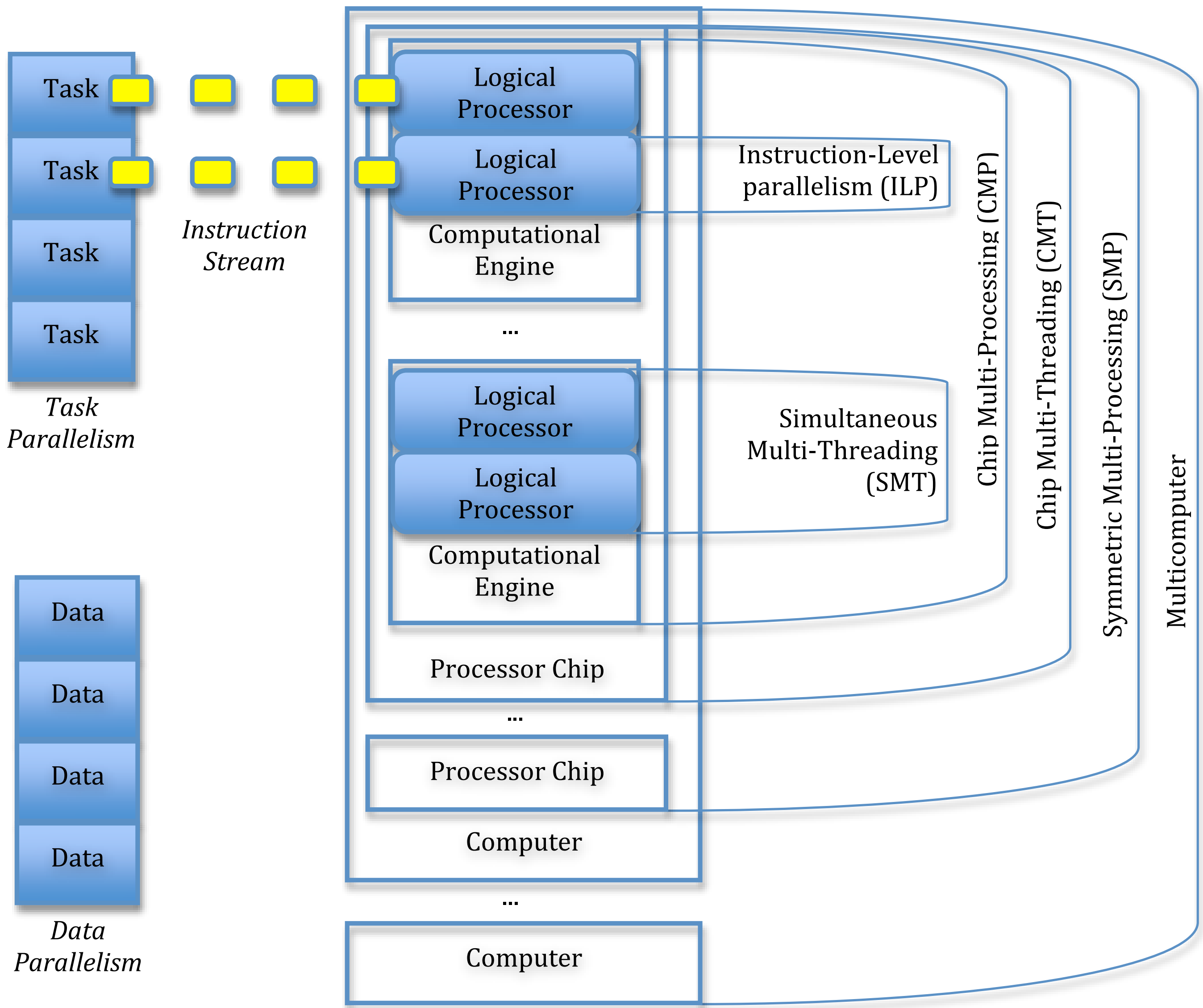


- Pipelining
- Super-scalar
- VLIW
- Branch prediction
- ...



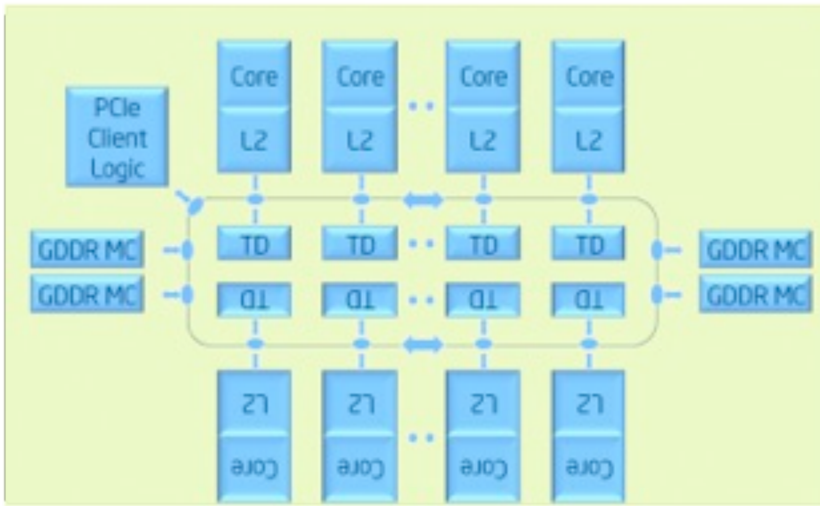


Parallel Hardware

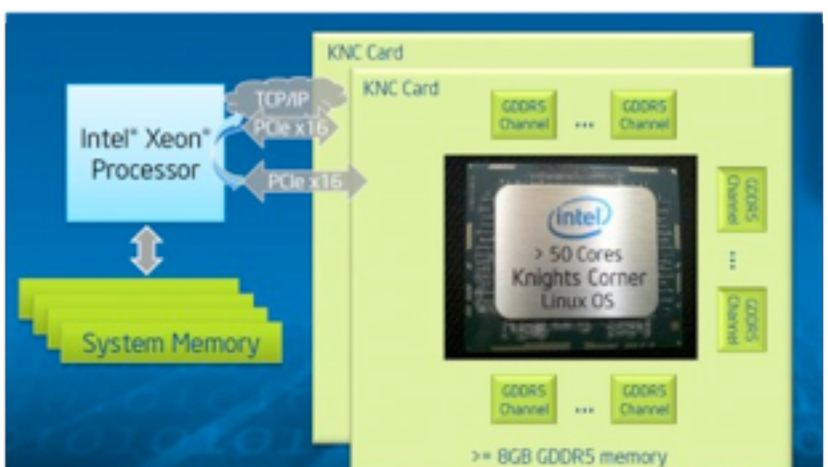


- Where ?
  - Inside the processor (instruction-level parallelism, multicore)
  - Through multiple processors in one machine (multiprocessing)
  - Through multiple machines (multicomputer)

Parallel Hardware



(George Chrysos, Intel)



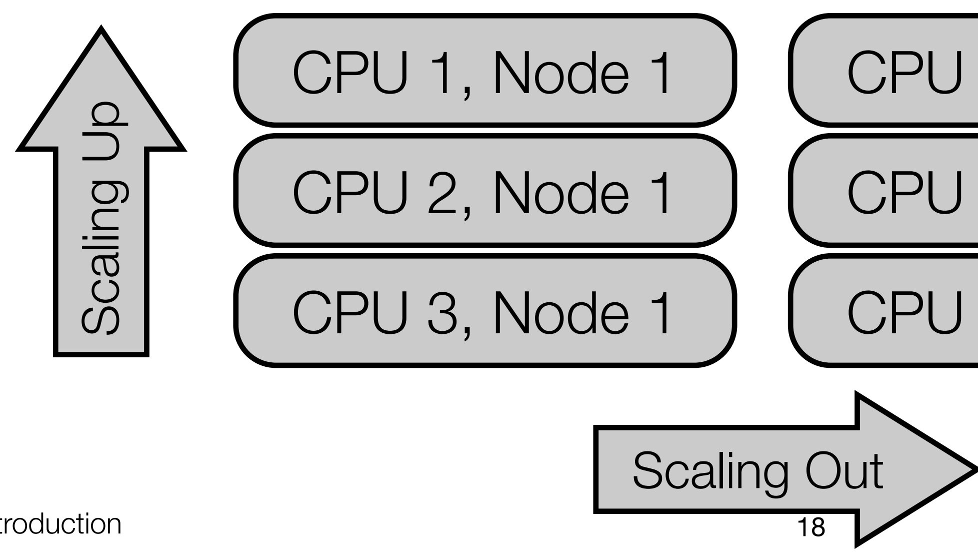
- Intel Knights Corner / Xeon Phi
  - Tag Directory (TD) per L2 cache
  - 4 groups of 16 cores,
    4 threads per core
  - 512-bit SIMD vector unit per core
  - Multiple rings (data, addresses, coherence information)
  - Gatter / scatter address machinery

Parallel Systems

- Always there, but widely ignored by the ‚average‘ developer
- Now mainstream - multi-core, hyper-threading, gaming consoles, GPU‘s
- High-End Systems
    - Toy Story (1995) - 100 dual-processor machines as render farm
    - Toy Story 2 (1999) - 1400 processor cluster
    - Monsters Inc. (2001) - 250 servers with 14 processors each = 3500 CPU‘s
    - HPI Future SOC Lab (2010) - 204 cores in 11 machines; 2.3 TB RAM
        - DL980 - 64 cores (8 x Xeon X7560), 2 TB RAM
- Clusters and custom-made MPP rules the HPC world

Reason for choosing a parallel architecture

- Performance - do it faster
- Throughput - do more of it in the same time
- Price / performance - do it as fast as possible for the given money
- Scalability - be prepared to do it faster with more resources
- Scavenging - do it with what I already have



Which One Is Faster ?

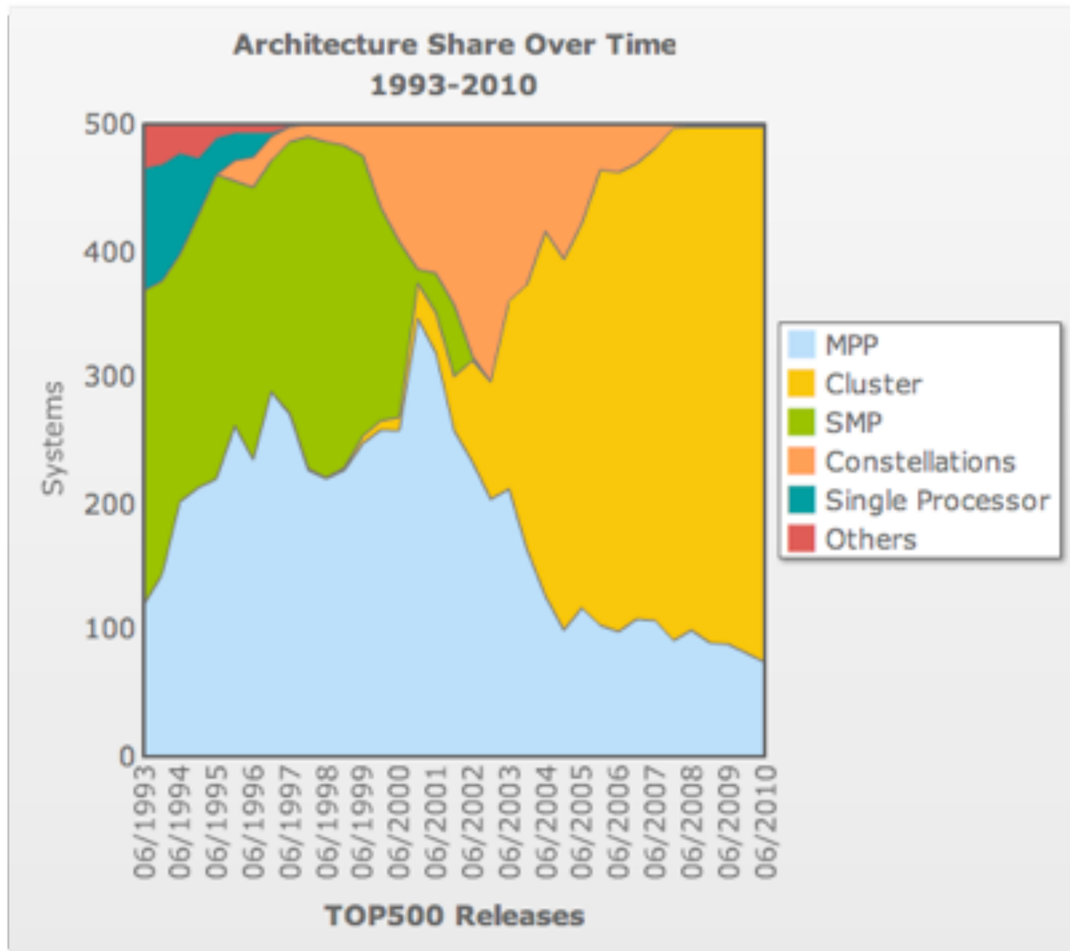
- Usage scenario
  - Transporting a fridge
- Usage environment
  - Driving through forrest
- Perception of performance
  - Maximum speed
  - Average speed
  - Acceleration

Getting Faster ?

- Sequential processing
- Parallel processing through pipeline
  - First results from previous step are already presented to next step
- Parallel processing of one task by splitting it up
  - Parallel sorting algorithms (e.g. Quicksort)
- Example: Processing of a SQL request (join of two tables)
  - Search -> Join -> Sort -> Write
- Interesting problems
  - What means „faster“ ?
  - Does „adding more processors“ automatically means „more performance“ ?

TOP 500

- It took 11 years to get from 1 TeraFLOP to 1 PetaFLOP
- Performance doubled approximately every year
- Assuming the trend continues, ExaFLOP by 2020
- TOP500 Nr.1 (2012) - IBM Sequoia:
  16,3 Petaflops,
  1.6 PB memory,
  98304 compute nodes,
  1.6 Million cores,
  7890 kW power

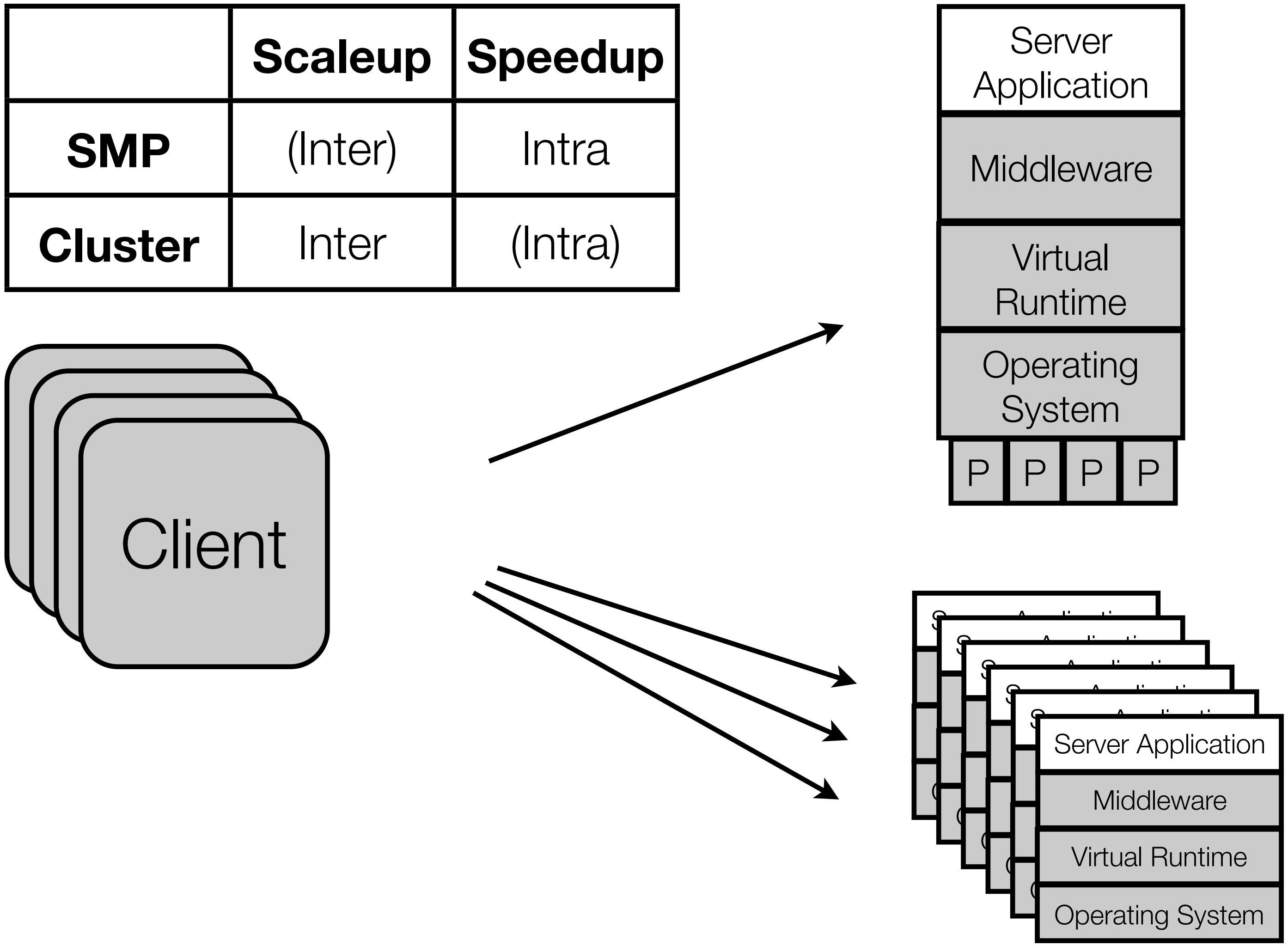


The Ideal Parallel System

- **Linear speedup**
    - n times more resources lead to n times less time for solving the same task
- **Linear scaleup**
    - n times more resources solve an n times larger problem in the same time
- Aimed goal depends on the application
    - Transaction processing usually heads for **throughput** (scalability)
    - Decision support system usually heads for better **response time** (speed)

Example: Server-Side Application Parallelism

	Scaleup	**Speedup**
SMP	(Inter)	Intra
Cluster	Inter	(Intra)



Intra-request parallelism for response time

Inter-request parallelism for throughput and fault tolerance

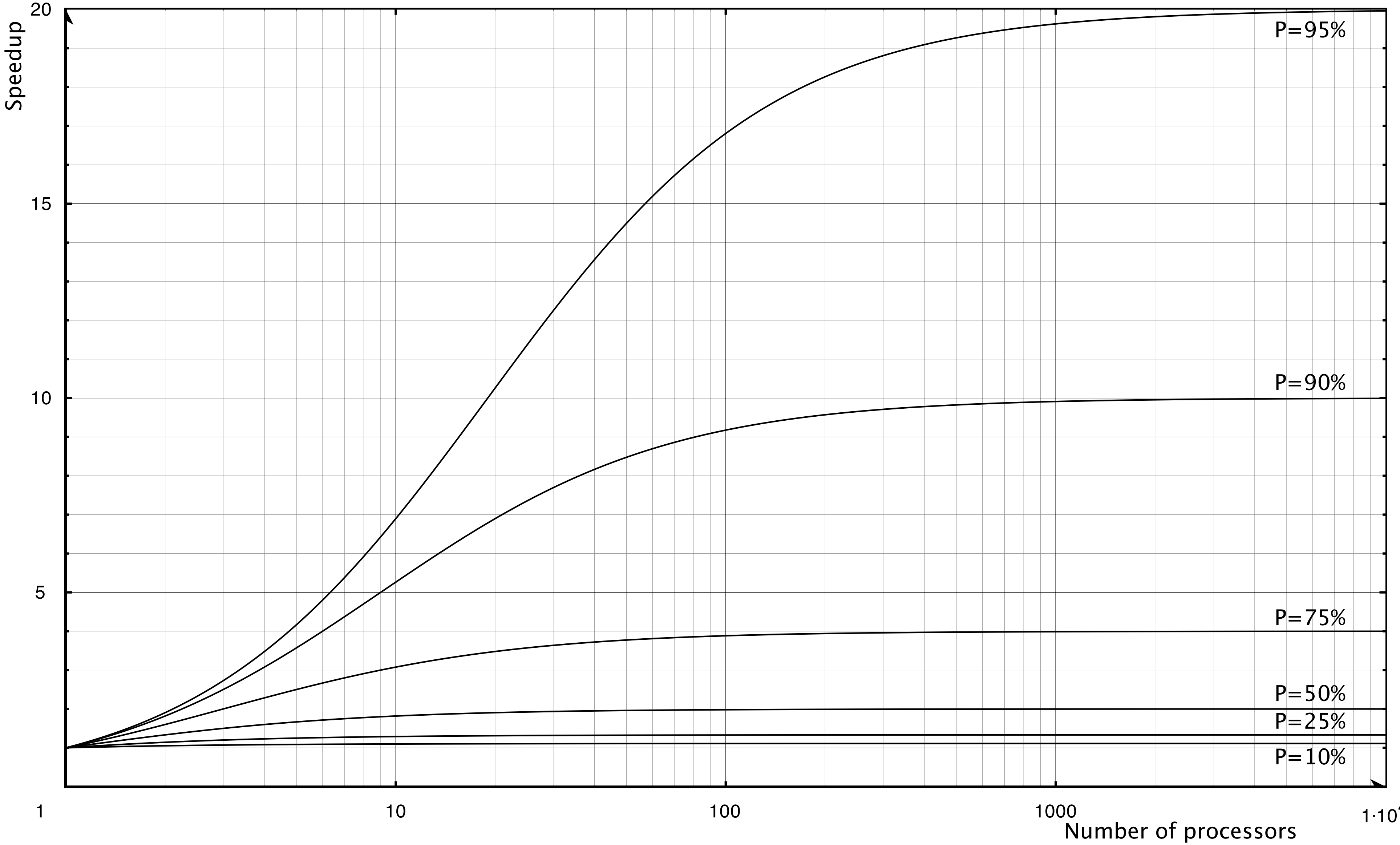
Problems with Speedup by Parallelization

- Well-researched problem in parallel databases (D. DeWitt, J. Gray)
  - **Start-Up**: Initialization of parallel activity, synchronization of results
  - **Interference**: Conflicts through access to shared data
  - **Dispersion**: Overall execution time depends on the slowest process
  - All problems increase with the number of processors
- **Amdahl‘s Law (1967)**
  - P is the portion of the program that benefits from parallelization
  - Maximum speedup by N processors:

$$s = \frac{(1-P)+P}{(1-P)+\frac{P}{N}}$$

    - Maximum speedup tends to 1 / (1-P)
    - Parallelism only reasonable with small N or small *(1-P)*

Amdahls Law



Speedup
20
15
10
5
P=95%
P=90%
P=75%
P=50%
P=25%
P=10%
1
10
100
1000
1·10^4
Number of processors

Implications

- Maximum theoretical speedup is N (linear speedup)
- BUT: Amdahl assumed fixed problem size, and looked on execution time
  - Problem size could scale with the number of processors („do more“)
  - Time spend in the sequential part usually depends on problem size
  - Run-time is typically an expected constant value („paper deadline“)
- Gustafson‘s Law
  - Let p be a measure of problem size, S(p) the time for the sequential part
  - Maximum speedup by N processors: $S(p) + N * (1 - S(p))$
  - When serial function part shrinks with increasing p, speedup grows as N
- *Everyone knows Amdahl’s law, but quickly forgets it. [Thomas Puzak, IBM]*

Terminology

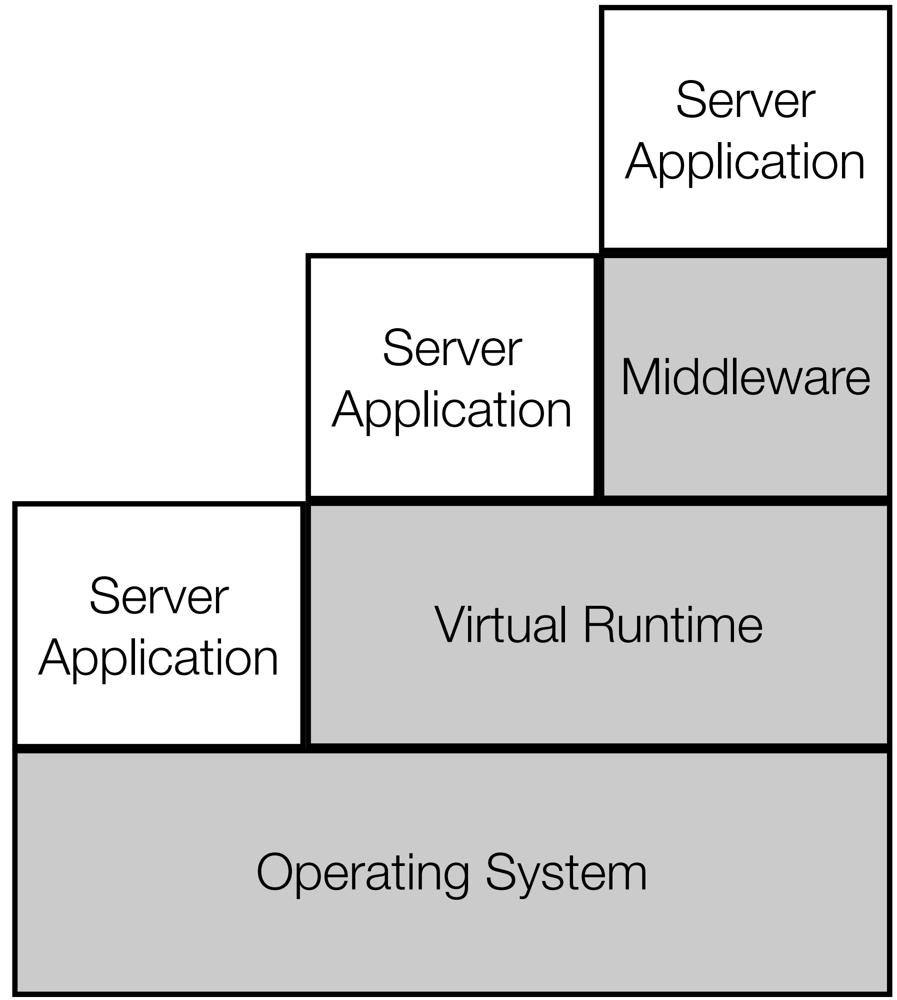
- **Concurrency**
  - Supported to have two or more actions *in progress* at the same time
  - Classical operating system responsibility
    (resource sharing for better utilization of CPU, memory, network, ...)
  - Demands **scheduling** and **synchronization**
- **Parallelism**
  - Supported to have two or more actions executing *simultaneously*
  - Demands **parallel hardware**, **concurrency support**, (and **communication**)
  - Programming model relates to chosen hardware / communication approach
- Examples: Windows 3.1, threads, signal handlers, shared memory

Terminology

- **Concurrency** vs. **parallelism** vs. **distribution**
  - Two threads started by the application
    - Are given as *concurrent* activities by the program code
    - Might (!) be executed in *parallel*
    - Concurrent code be *distributed* on different machines
  - Windows 3.1 had concurrency, but no parallelism
  - Parallelism demands parallel hardware (see last lecture)
  - Concurrency demands some scheduler
- Concurrent programming: Signal handling, thread library
- Parallel programming: Synchronization and communication

Support for Concurrent Applications

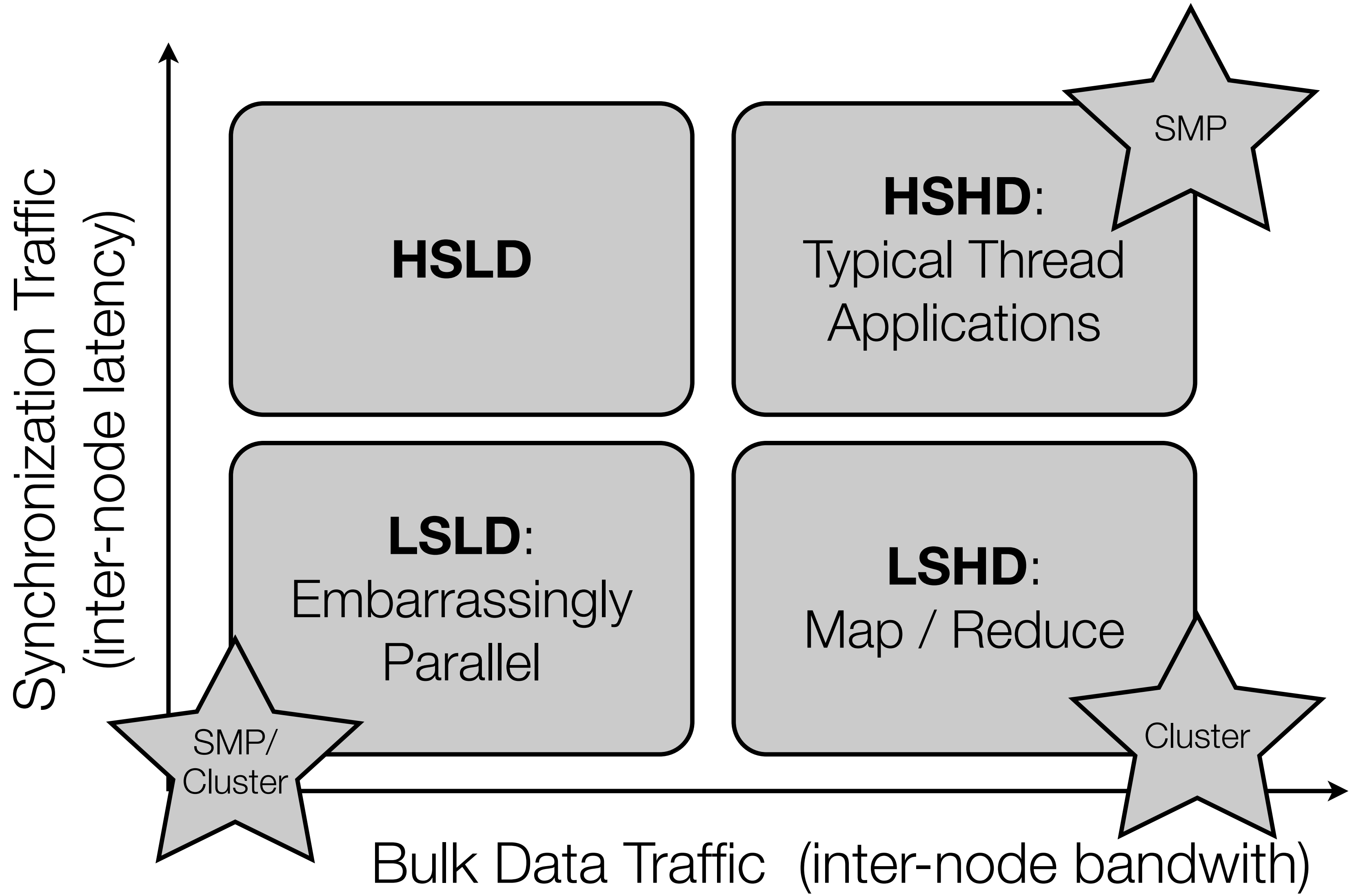
- By operating system
  - SMP-aware schedulers
- By virtual runtime
  - Java / .NET threading support
- By middleware
  - J2EE / CORBA thread pooling
- By application itself



Concurrent Programming

- Independent computations the machine can execute in any order
  - Iterations of (some) loops
  - Independent function calls
- Concurrency overhead: Create, manage, and synchronize concurrent tasks
- Threading methodology [Intel]
  - Analyze - Identify independent computations, find hotspots by profiling
  - Design and implement
  - Test for correctness - no altering of serial logic, data races, deadlocks
  - Tune for performance

Parallel Application Characteristics (Pfister)



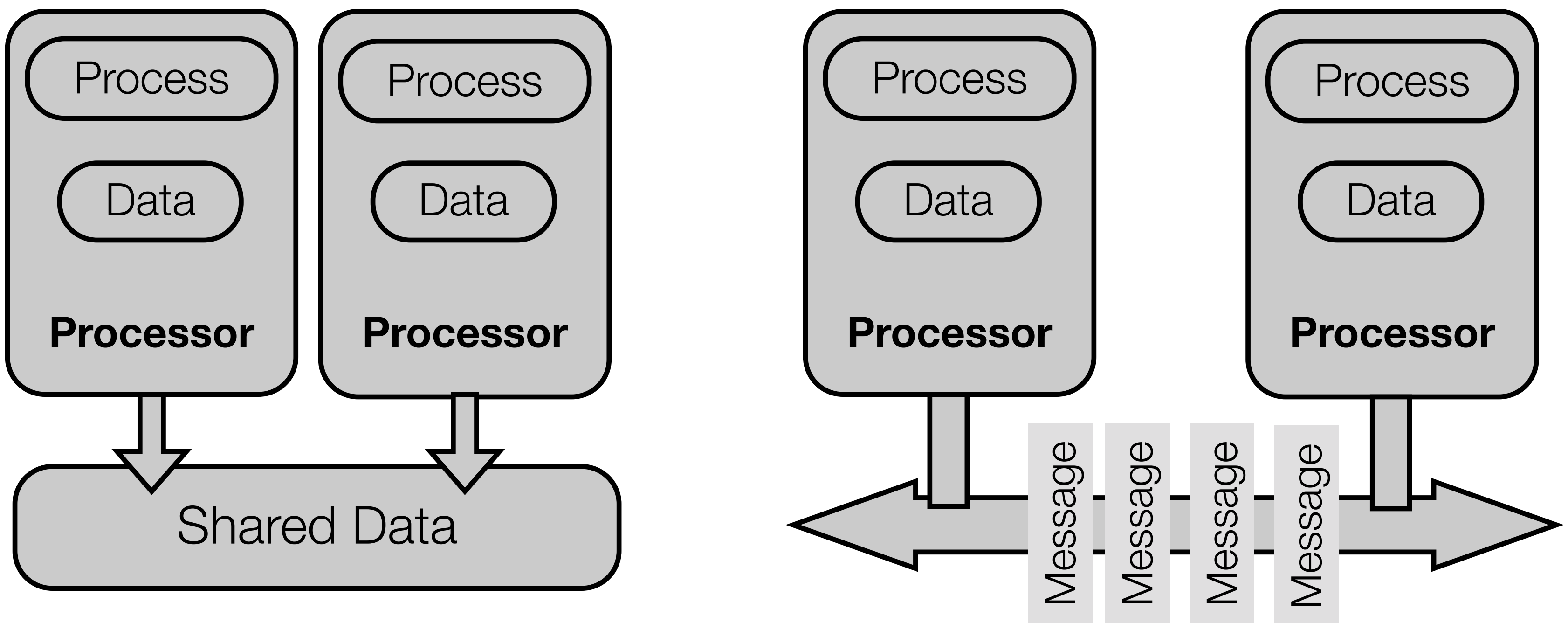
Terminology [Mattson et al.]

- **Task** - Parallel program breaks a problem into tasks
- **Execution unit** - Representation of a concurrently running task (e.g. thread)
  - Tasks are mapped to execution units during development time
- **Processing element** - Hardware element running one task
  - Depends on scenario - logical processor vs. core vs. machine
  - Execution units are mapped to processing elements by scheduling
- **Synchronization** - Mechanism to order activities of parallel tasks
- **Race condition** - Program result depends on scheduling of execution units

Programming Models

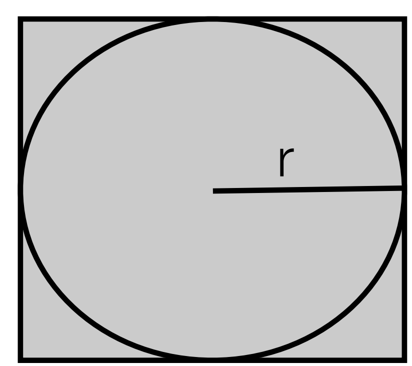
- Almasi and Gottlieb: *„set of rules for a game“*
  - Programs and algorithms as game strategies
- High-level view of the application on it‘s run time environment
  - Hardware might imply a programming model, but does not enforce it
  - Reflects on the design of the application
- For uni-processor, no question due to „von Neumann“
- For parallel architectures, **shared-memory**, **message passing** or **data parallelism** approaches
- Models in use depend on size of parallel system (**Small N** vs. **Large N**)
- Delivering performance while raising the level of abstraction

Shared Memory vs. Message Passing



Examples

- Fibonacci function $F_{K+2}=F_K+F_{K+1}$
  - Cannot be parallelized, since each computed value depends on earlier one
- Parallel search
  - Looking in a search tree for a ‚solution‘
  - New tasks for sub-trees, with channel to parent
- PI approximation by master-worker scheme (monte carlo simulation)
  - Area of the square $A_S=(2r)^2=4r^2$, area of the circle $A_C=pi*r^2$, so $pi=4*A_C / A_S$
  - Randomly generate points in the square
  - Compute A_S and A_C by counting the points inside the square / circle



„The vast majority of programmers today
don’t grok concurrency,
just as the vast majority of programmers 15 years ago
didn’t yet grok objects“

(Herb Sutter, 2005)