

# Parallel Programming Concepts

## Programming Models

---

Peter Tröger

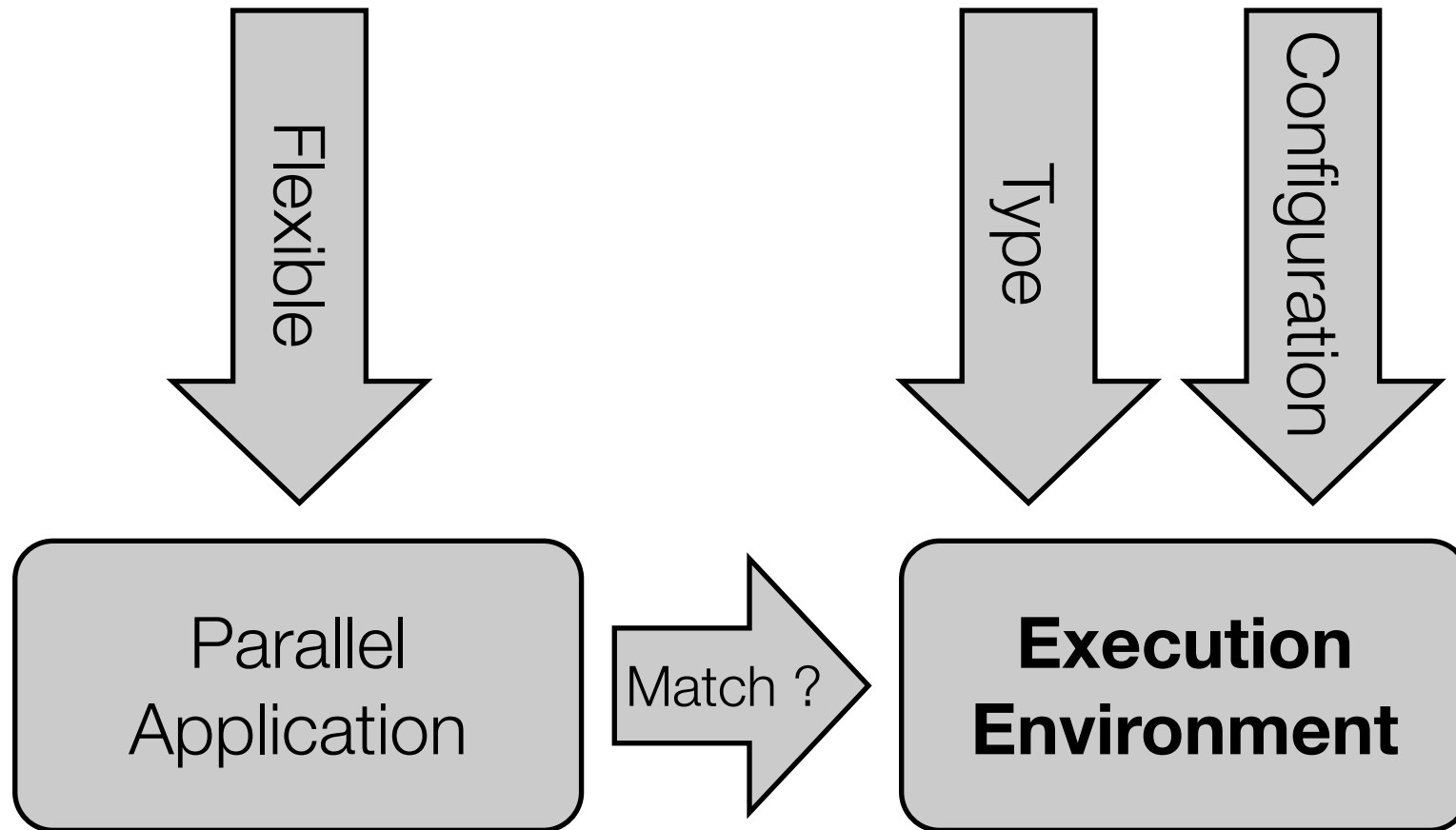
*Sources:*

*Clay Breshears: The Art of Concurrency*

*Blaise Barney: Introduction to Parallel Computing*

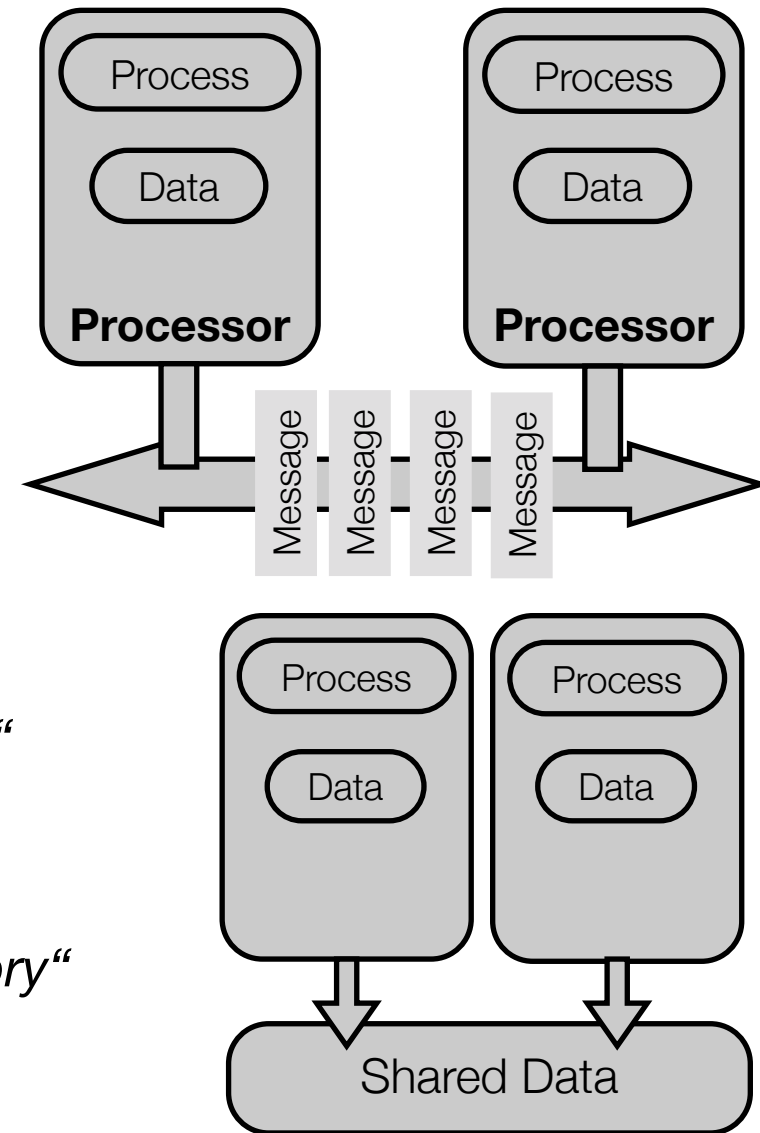
# The Parallel Programming Problem

---



# Execution Environment

- Execution environments can be distinguished in two classes
  - „*Shared memory*“: SMP hardware, distributed shared memory, virtual runtime environments, ...
  - „*Shared nothing*“: Hardware clusters, massively parallel HPC, grid computing, processor interconnect, Hadoop, ...
- Pfister: „*shared memory*“ vs. „*distributed memory*“
- Foster: „*multiprocessor*“ vs. „*multicomputer*“
- Tannenbaum: „*shared memory*“ vs. „*private memory*“
- Environment can be either hardware or software

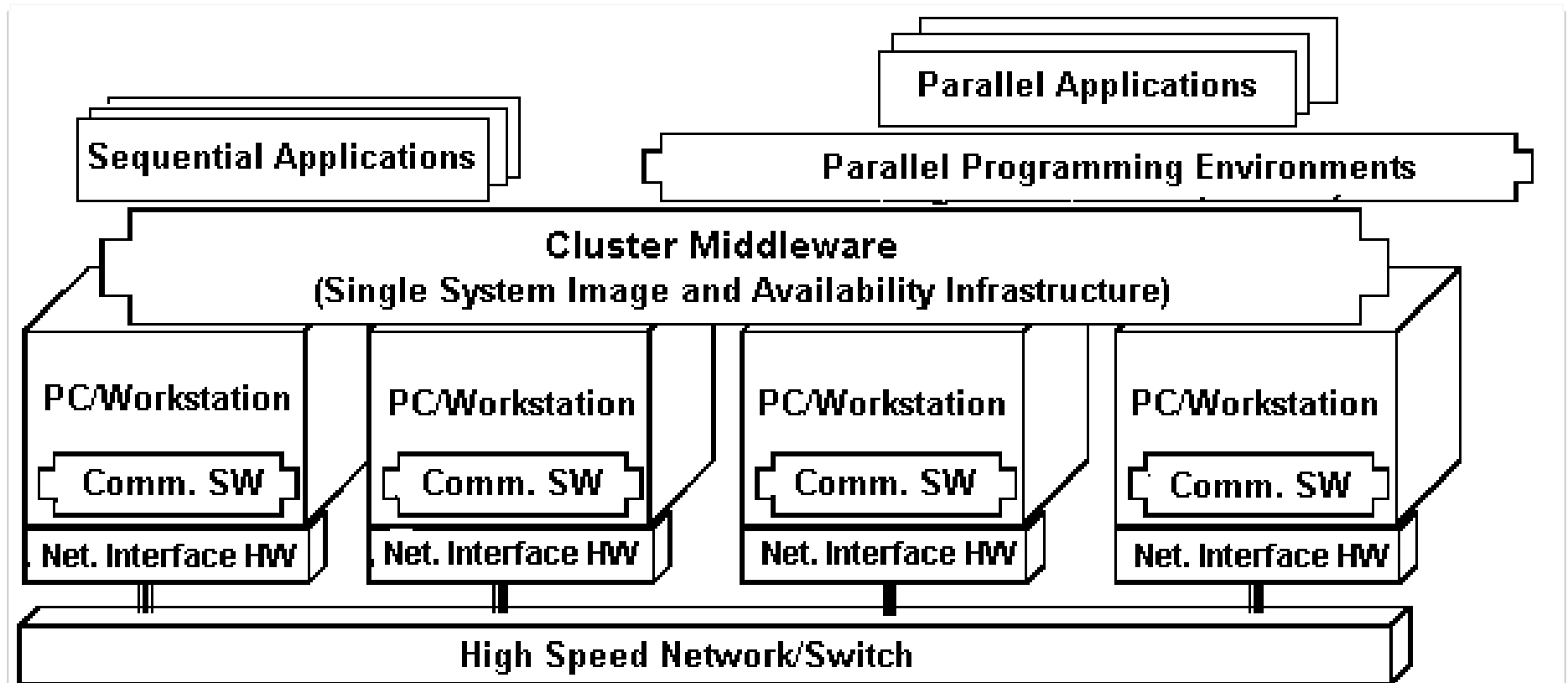


# Shared Memory

---

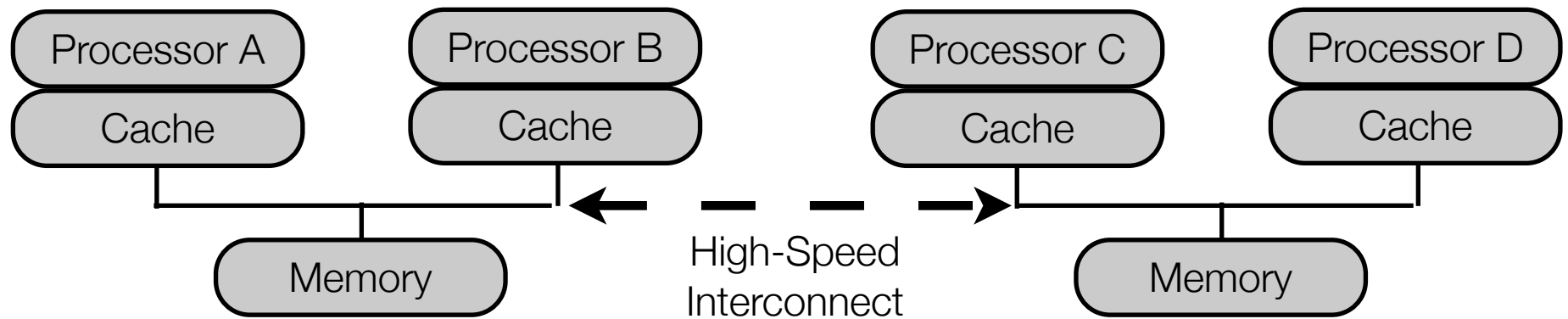
- All processors act independently, access the same global address space
- Changes in one memory location are visible for all others
- **Uniform memory access (UMA) system**
  - Equal load and store access for all processors to all memory
  - Default approach for majority of SMP systems in the past
- **Non-uniform memory access (NUMA) system**
  - Delay on memory access according to the accessed region
  - Typically realized by processor interconnection network and local memories
  - Cache-coherent NUMA (CC-NUMA), completely implemented in hardware
  - About to become standard approach with recent X86 chips

# Shared Nothing



# Hybrid Environments

---



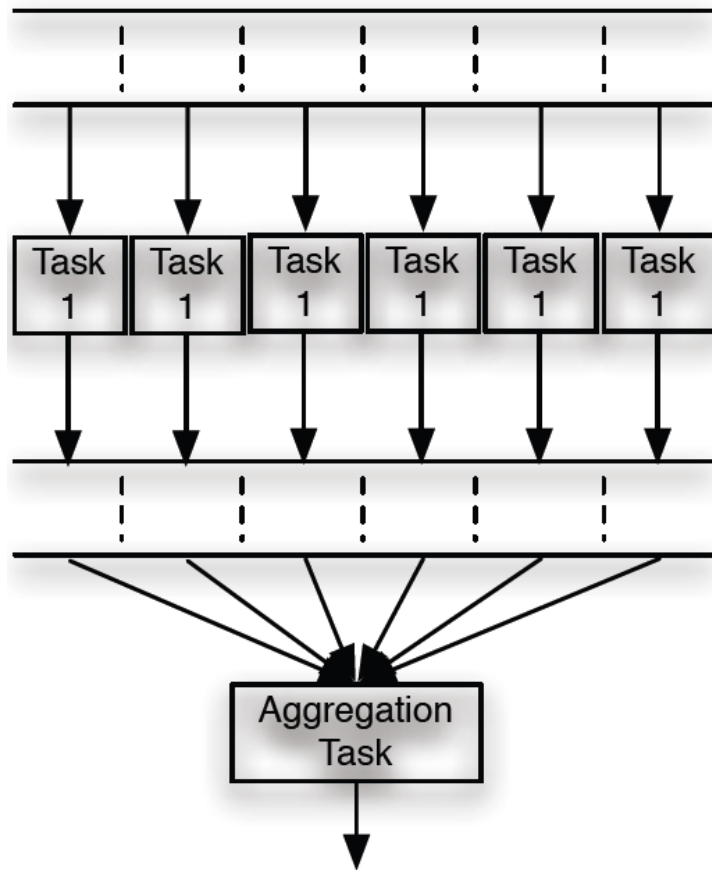
# Execution Environment

---

- Hardware / software execution environment with „*shared memory*“ resp. „*shared nothing*“ is optimized for specific workload
  - „*task parallel*“
    - Different operations being performed at the same time
    - Might originate from the same or different programs
  - „*data parallel*“
    - Parallel execution of the same operation on disjoint data sets
- Maps directly to SIMD / MIMD, orthogonal to memory semantics
- Sometimes also „*flow parallelism*“ added
  - Overlapping work on data stream (pipelines, assembly line model)

# Execution Environment

## Data Parallelism

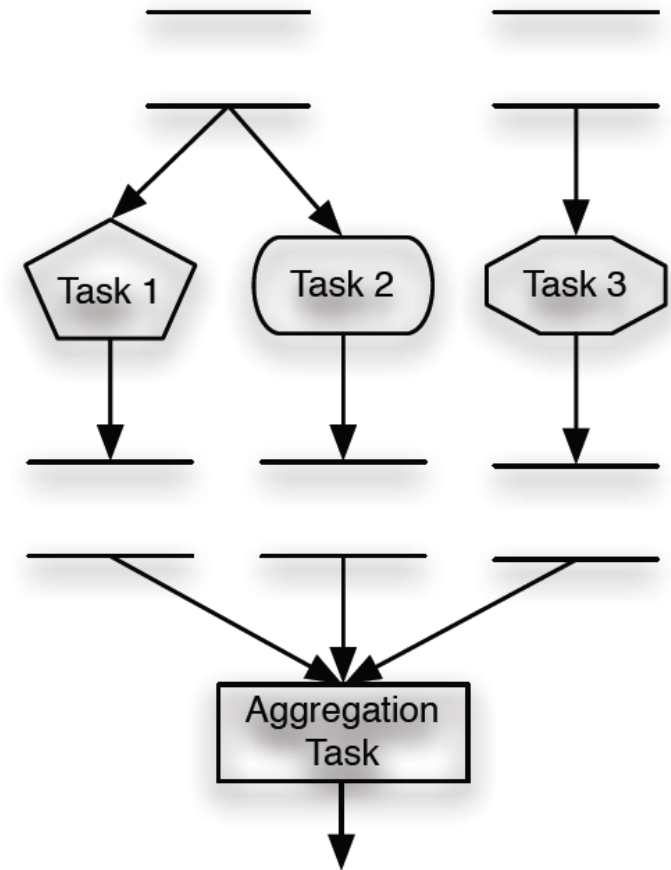


## Task Parallelism

Input Data

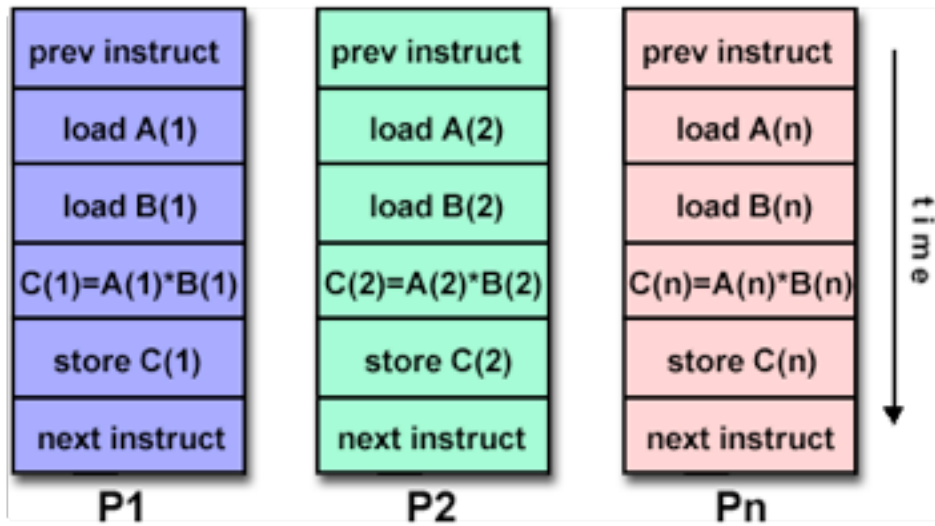
Parallel Processing

Result Data

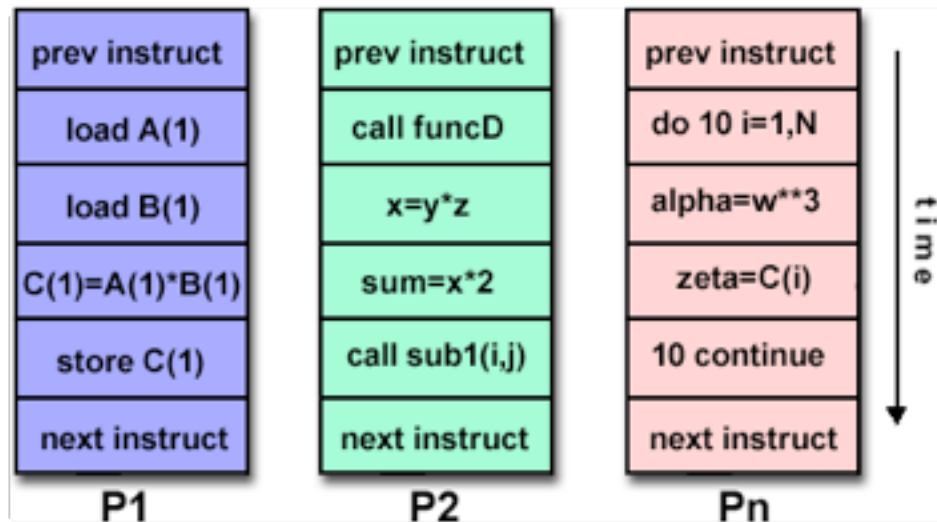
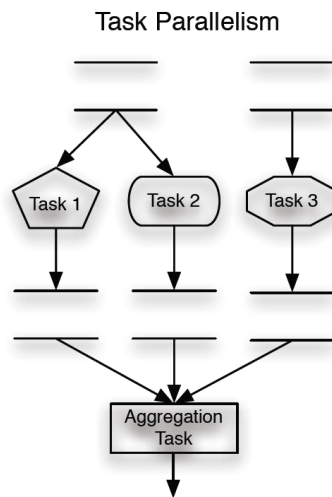
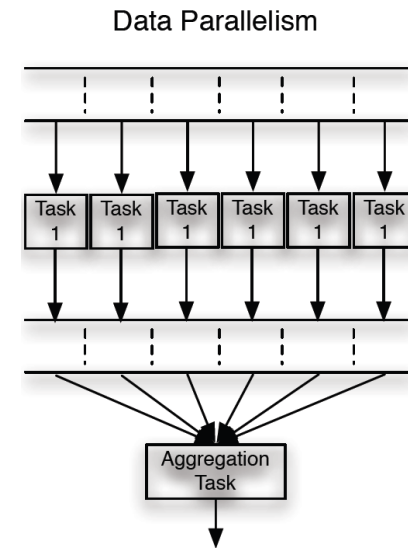


# Execution Environment

(C) Blaise Barney



**Single Instruction,  
Multiple Data (SIMD)**



**Multiple Instruction,  
Multiple Data (MIMD)**

# Execution Environment Examples

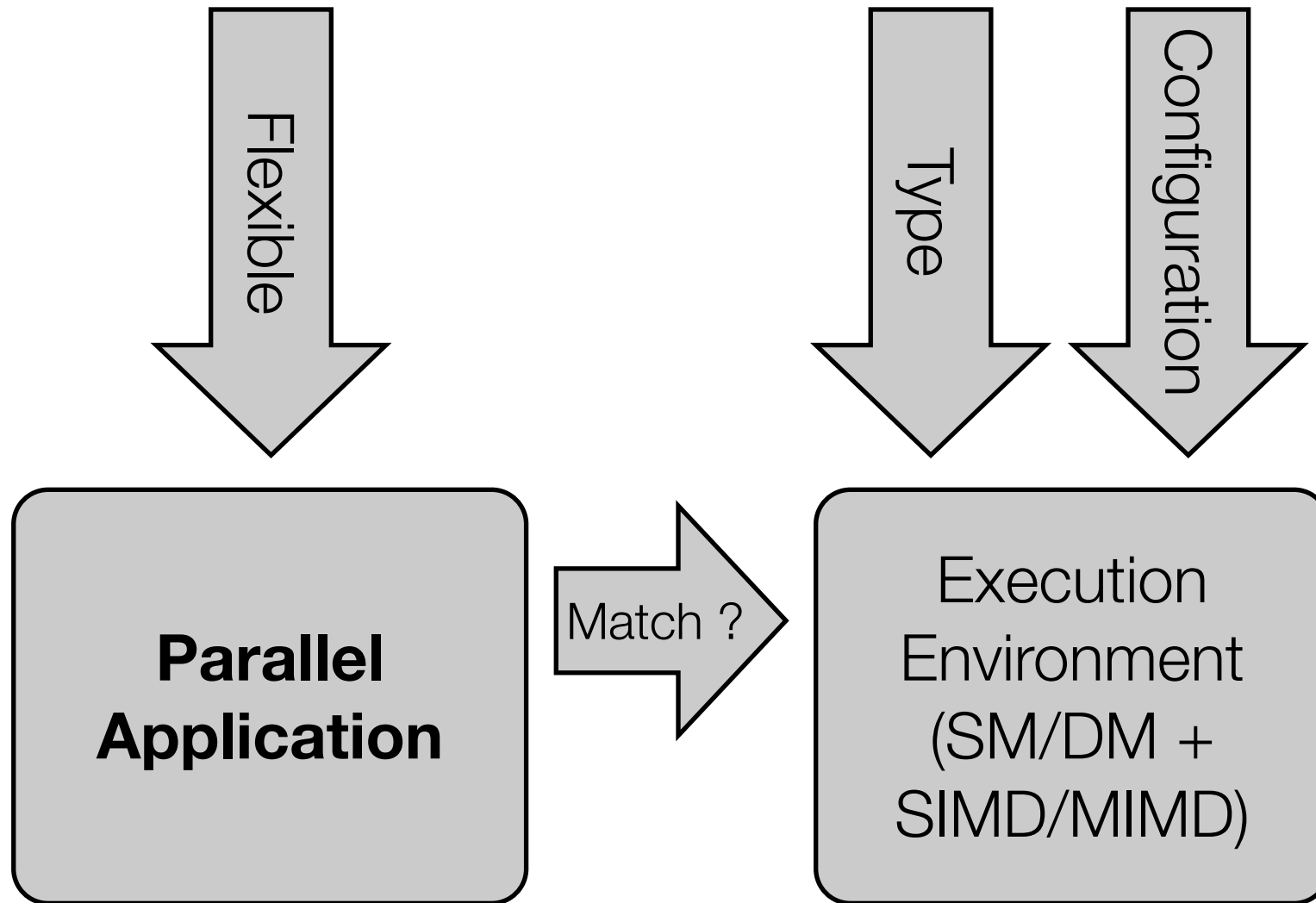
	Data Parallel / SIMD	Task Parallel / MIMD
Shared Memory (SM)	<i>SM-SIMD</i> GPU, Cell, SSE, AltiVec Vector processor ...	<i>SM-MIMD</i> ManyCore/SMP system ...
Shared Nothing / Distributed Memory (DM)	<i>DM-SIMD</i> processor-array systems systolic arrays Hadoop ...	<i>DM-MIMD</i> cluster systems MPP systems ...

- Note:

Task-parallel execution environments are easily also usable as data-parallel execution environment, but not optimized for it

# The Parallel Programming Problem

---



# Parallel Application

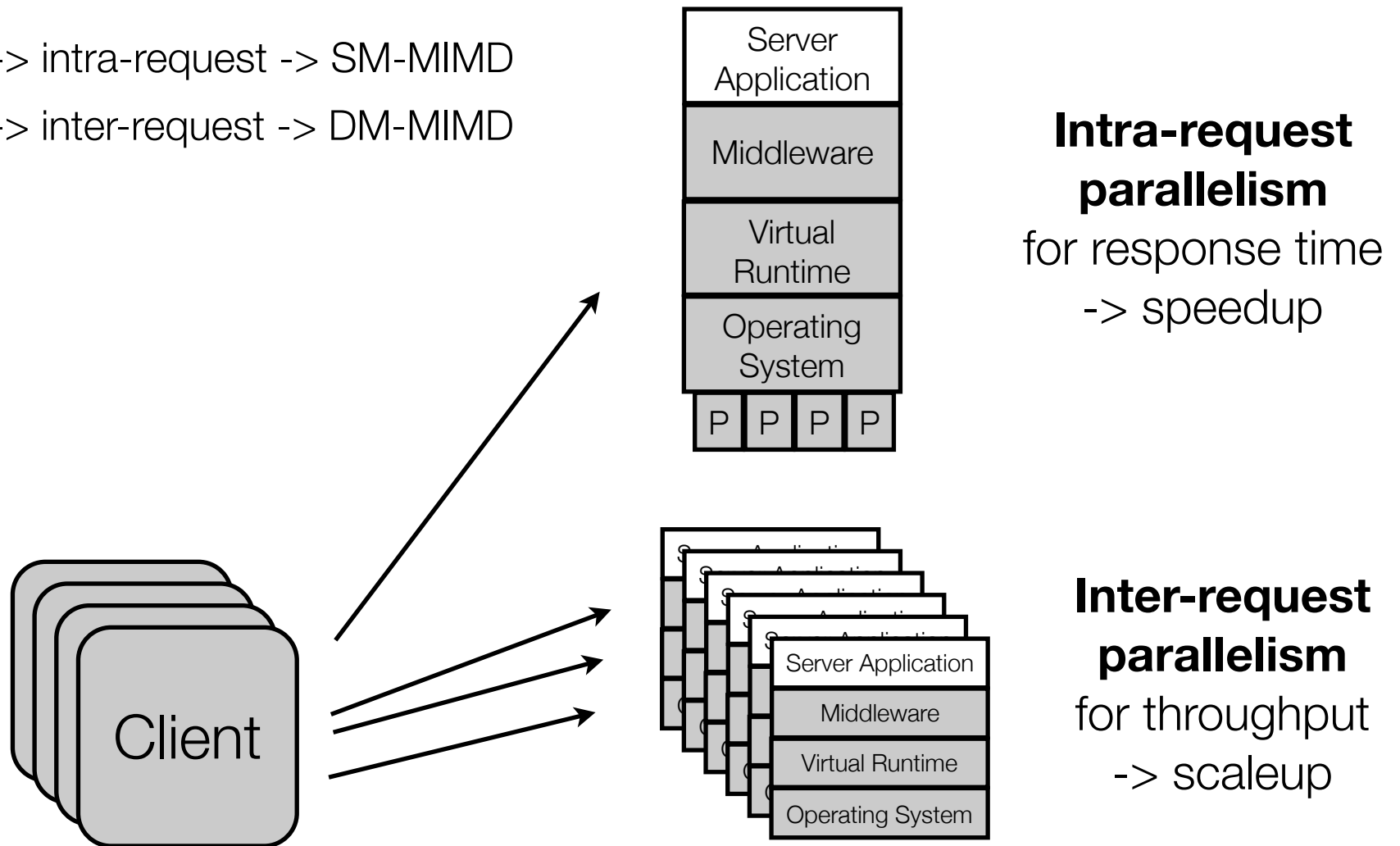
---

- Motivation for porting your application:
- **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
- Goal depends on the application, examples:
  - Transaction processing usually heads for **throughput** (scalability)
  - Decision support system usually heads for better **response time** (speed)
- ‚Parallel application‘ is a widely used term

# Example: Server-Side Parallelism

Speedup -> intra-request -> SM-MIMD

Scaleup -> inter-request -> DM-MIMD



# Parallel Application

---

- Parallel application execution
  - Supported to have two or more actions executing *simultaneously*
  - Demands execution environment
- Parallelization can be coarse-grained or fine-grained
  - Decision of algorithm design and / or configuration
- Proposal for this course (this is debatable)
  - **coarse-grained** == **inter-request parallelism** (think Apache)
  - **fine-grained** == **intra-request parallelism** (think multithreading)
  - Interpret single application start as one ,request‘
- How to formulate our parallel application ? We need a **programming paradigm**.

# Programming Paradigm

---

- Programming paradigm: coding convention or standard
  - Something a majority of people agrees upon
- Parallel programming is one of these paradigms
  - Other examples: declarative, constraint-based, structured, object-oriented
- Each paradigm can be realized by a set of programming models
  - Programming model: „*set of rules for a game*“ [Almasi, Gottlieb]
  - Point where execution environment and application meet
  - Programming models for parallel programming:

**Multi-Tasking, Message Passing, Implicit Parallelism**

# From Multi-Tasking to Implicit Parallelism

---

- Imperative multi-tasking fails to solve concurrency issues
  - At each statement, developer must decide semantically upon locks to ensure correct data access and data modification
  - For each method call, one must reason about locks being held (deadlock)
  - Locks are not fixed at compile time, new might be created during run time
  - Additional locks might remove race conditions, but also add new deadlocks
  - -> Tackle the problem from a completely different direction
- **Declarative / implicit parallelism** instead of imperative programming
- **Message passing** instead of shared memory as concurrency base

# Programming Models

---

- High-level view of the application on it's execution environment
- Intended as contract - if you follow the rules, scalability should be achievable
- Decouples software and execution environment architecture development
- Classification
  - **Multi-Tasking:** Typically used for SM-MIMD execution environments, recently also relevant for SM-SIMD environments
  - **Message Passing:** Typically used for DM-MIMD execution environments
  - **Implicit Parallelism:** globally useful, since mapping is implicit
  - No enforcement, different mappings are possible
- Programming models are implemented by languages and libraries

# Parallel Programming Languages

---

- Programming languages contain of syntax and standard library
  - Example: C + libc, Python + class library
  - Allows adding parallel programming support to sequential languages
- Often languages implement more than one parallel programming model
- Languages are often categorized by their feasibility for a particular execution environment - „*data-parallel languages*“ vs. „*task-parallel languages*“
- Most algorithmic problems obviously match to one kind of execution environment: „*data-parallel problem*“ vs. „*task-parallel problem*“

Multi-Tasking	PThreads, OpenMP, OpenCL, Linda, Cilk, ...
Message Passing	MPI, PVM, CSP Channels, Actors, ...
Implicit Parallelism	Map/Reduce, PLINQ, HPF, Lisp, Fortress, ...
Mixed Approaches	Ada, Scala, Clojure, Erlang, X10, ...