PMPP 2015/16

Design Patterns (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

GCC
Graphics, Capture and Massively Parallel Computing

Course Schedule

12.10.2015	Introduction to PMPP
13.10.2015	Lecture CUDA Programming 1
19.10.2015	Lecture CUDA Programming 2
20.10.2015	Lecture CUDA Programming 3
26.10.2015	Lecture Parallel Basics, Exercise 1 assigned
27.10.2015	Questions and Answers (Q&A), S3|19, Room 2.8
2.11.2015	Intro Final Proj., Ex. 1 due, Ex. 2 assigned, Lecture PRAM
3.11.2015	Lecture PRAM (2)
9.11.2015	Final Projects assigned, L. Parallel Sort., Exercise 2 due
10.11.2015	Questions and Answers (Q&A)
16.11.2015	Questions and Answers (Q&A)
17.11.2015	Questions and Answers (Q&A)
23.11.2015	1st Status Presentation Final Projects
24.11.2015	1st Status Presentation Final Projects (continued)
30.11.2015	Lecture Design Patterns
1.12.2015	Questions and Answers (Q&A)

(Preliminary) Course Schedule

you are here

7.12.2015	Lecture Design Patterns (2), Performance Tuning
8.12.2015	Questions and Answers (Q&A)
14.12.2015	Lecture
15.12.2015	Questions and Answers (Q&A)
11.1.2016	2nd Status Presentation Final Projects
12.1.2016	2nd Status Presentation Final Projects (continued)
18.1.2016	
19.1.2016	
25.1.2016	
26.1.2016	
1.2.2016	
2.2.2016	
8.2.2016	Final Presentation Final Projects
9.2.2016	Final Presentation Final Projects (continued)

Final Projects – Second Presentation

TECHNISCHE UNIVERSITÄT DARMSTADT

- second presentation on 11.01.2016 and 12.01.2016
  - present the current status of your project
  - discuss planned next steps
  - discuss open issues
  - DO NOT present the topic and the approach you plan to use once more
  - 7 minutes per group (!)

- slides must be submitted no later than 10 am on 11.01.2016
  - one presentation per group (not per topic)

- everybody should give a portion of the presentation

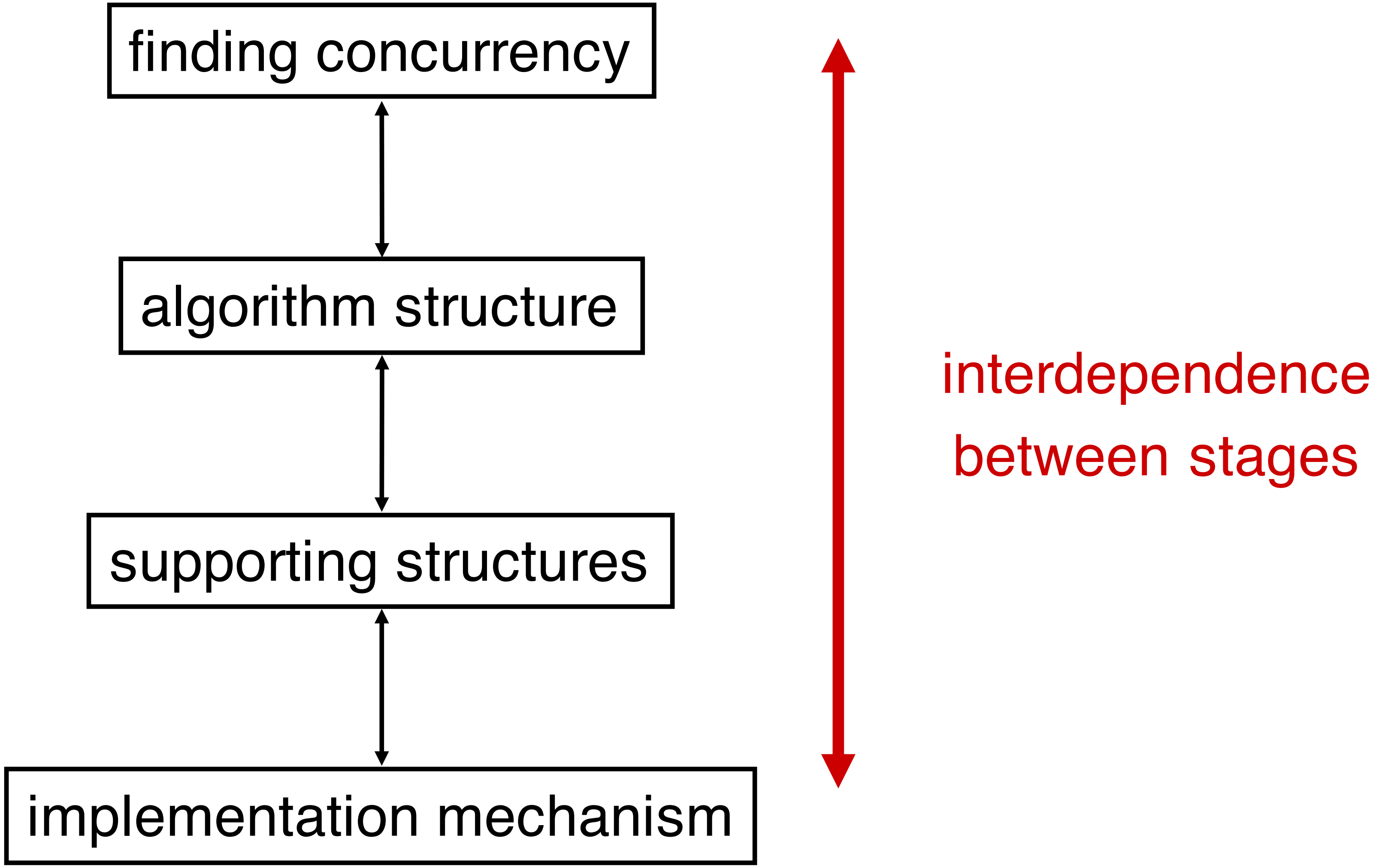
- mandatory (talk to us if this is a problem)

Written Exam

- Date for the exam will be 02.03.2016, starting at 2pm
  - do not forget to register in TuCAN!
- Rooms: S101/A1 and S101/A01
- The exam will be in English only
- overlap with exam "Communication Networks 2" will be resolved by an addition exam slot for CN2
  - exceptional, one-time solution only
  - thanks to Prof. Steinmetz and his team

!

Design Spaces



from Patterns for Parallel Programming by Mattson, Sanders, and Massingill

Design Spaces and Design Patterns

TECHNISCHE UNIVERSITÄT DARMSTADT

- finding concurrency
    - programmer working in the problem domain to identify available concurrency and expose it in the algorithm design

➔task decomposition pattern

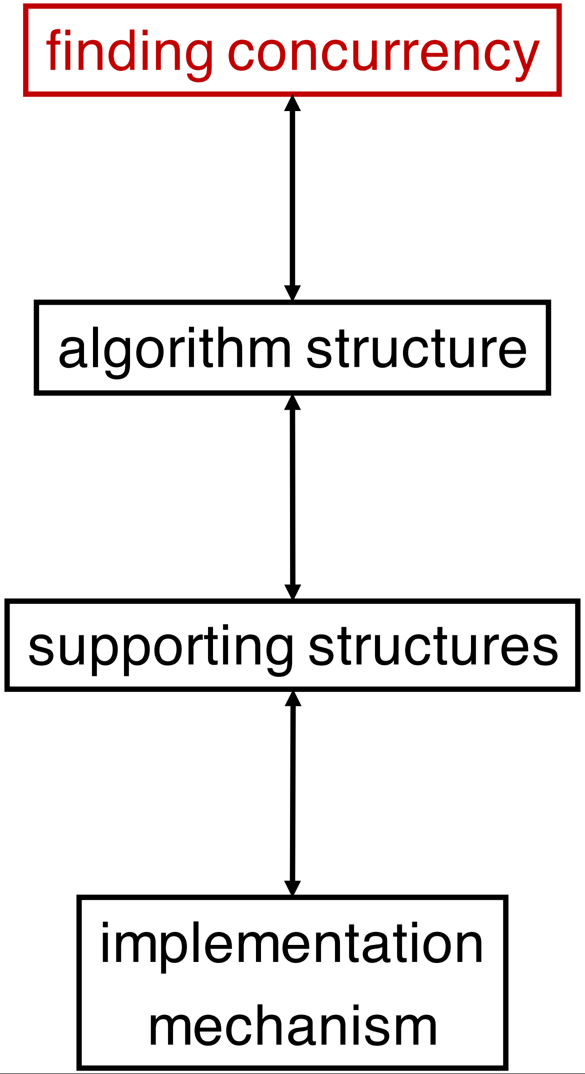
➔data decomposition pattern

➔group tasks pattern

➔order tasks pattern

➔data sharing pattern

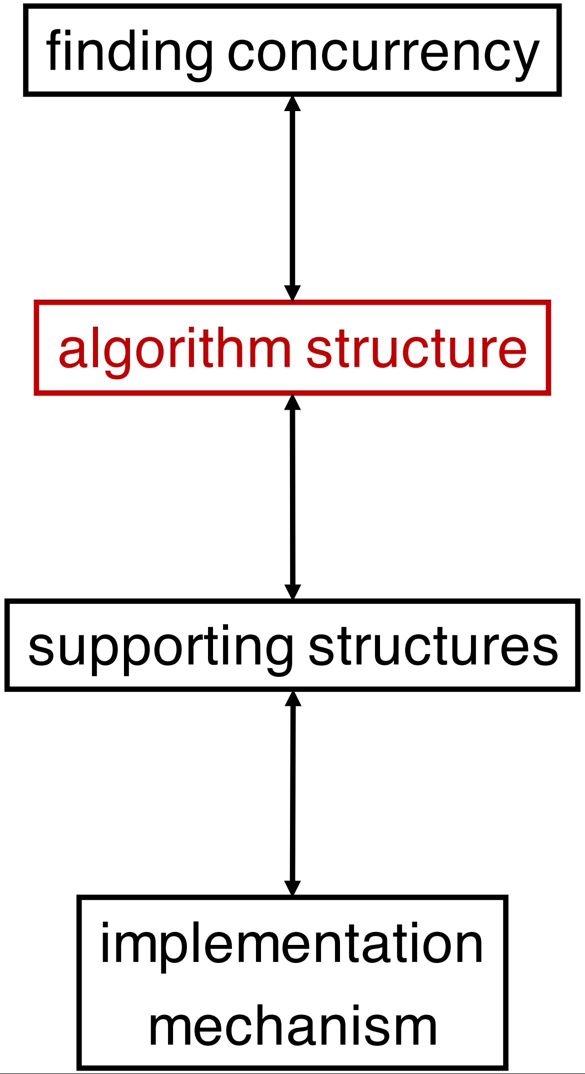
➔design evaluation pattern



Design Spaces and Design Patterns

- algorithm structure
  - programmer working with high-level structures for organizing a parallel algorithm

➔task parallelism pattern
➔divide and conquer pattern
➔geometric decomposition pattern
➔recursive data pattern
➔pipeline pattern
➔event-based coordination pattern

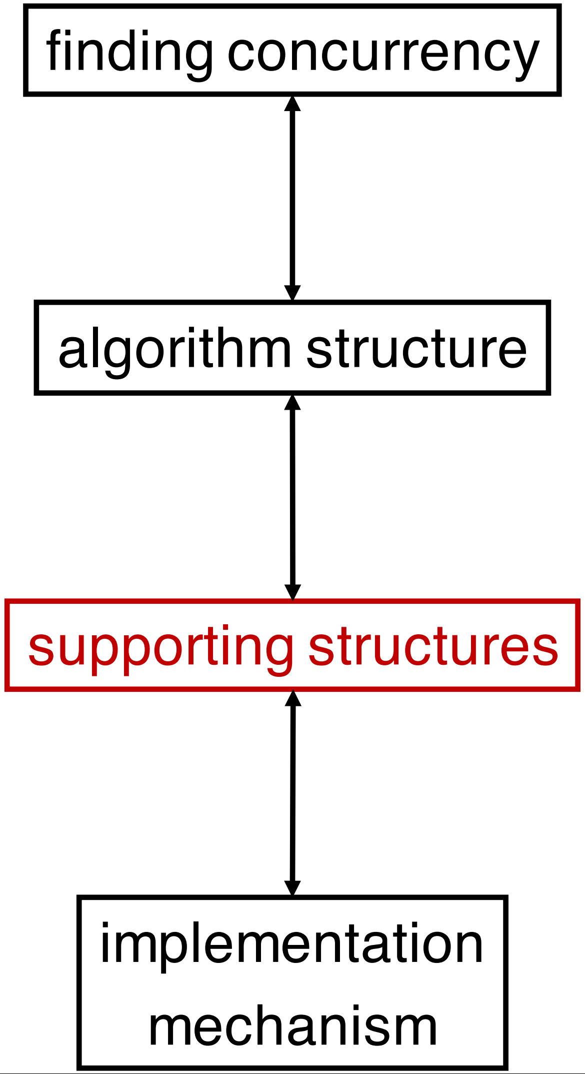


Design Spaces and Design Patterns

TECHNISCHE UNIVERSITÄT DARMSTADT

- supporting structures
  - shift from algorithms to source code
  - organization of parallel program
  - techniques to manage shared data

➔SPMD pattern
➔master/worker pattern
➔loop parallelism pattern
➔fork/join pattern
➔shared data pattern
➔shared queue pattern
➔distributed array pattern
➔…



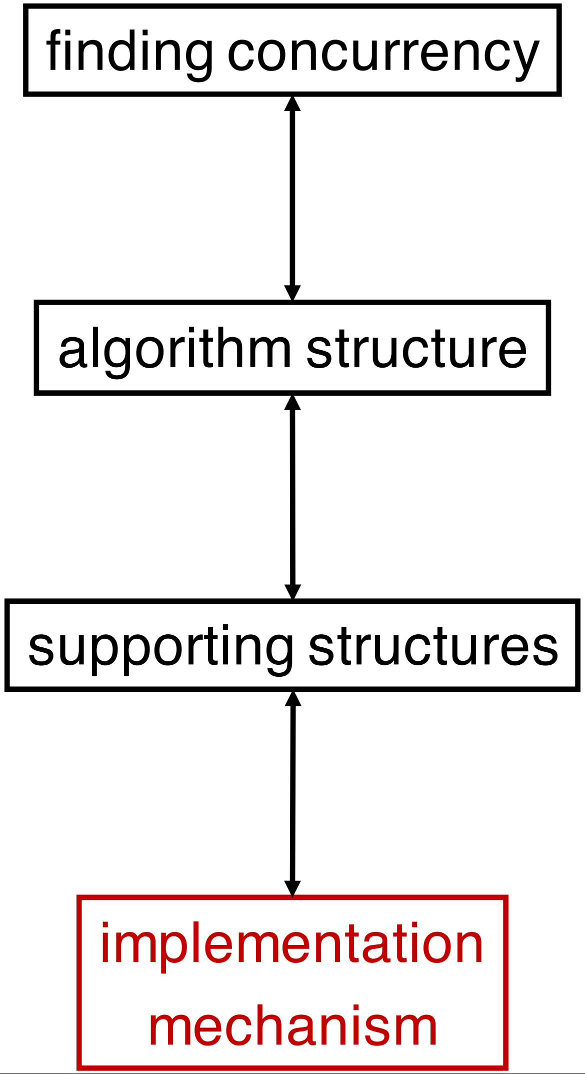
GCC Graphics, Capture and Massively Parallel Computing

Design Spaces (and Design Patterns)

TECHNISCHE UNIVERSITÄT DARMSTADT

- implementation mechanisms
  - specific software constructs for implementing a parallel program

➔UE management
➔synchronization
➔communication



GCC Graphics, Capture and Massively Parallel Computing

Design Spaces

finding concurrency

algorithm structure

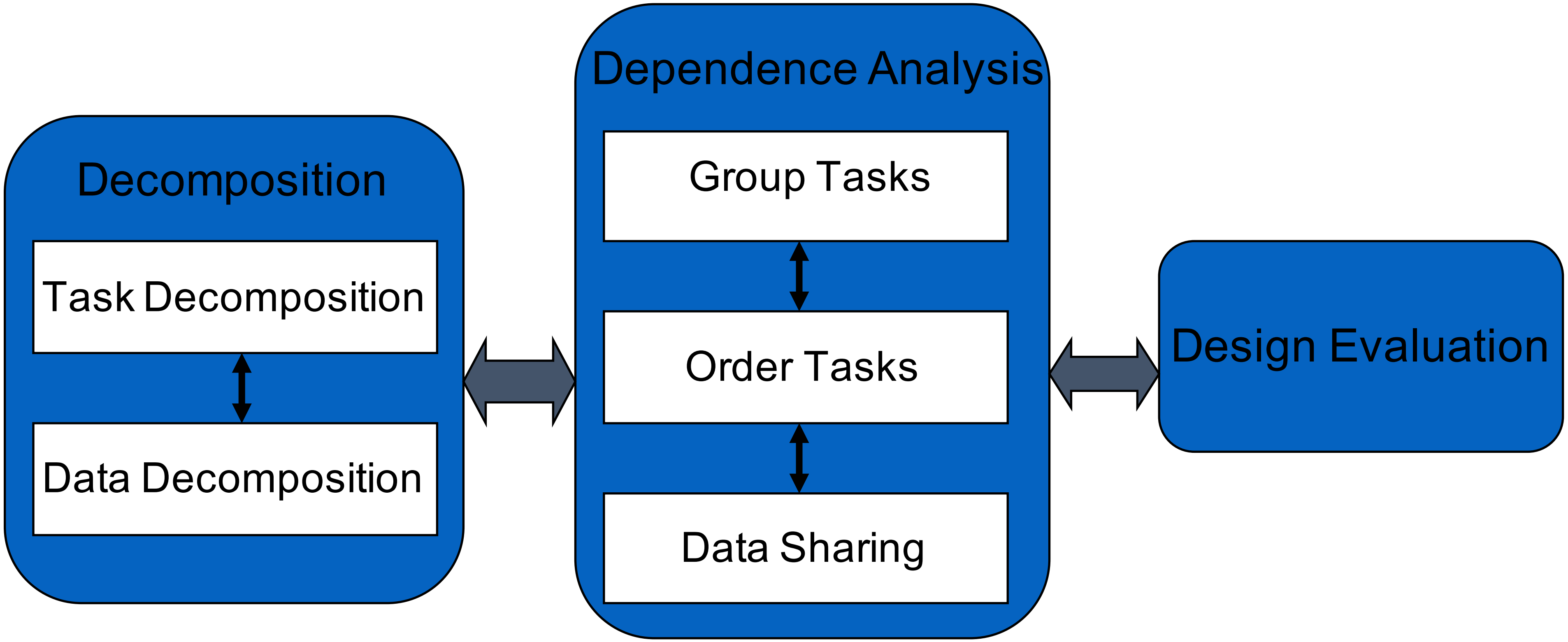
supporting structures

implementation mechanism

problem domain

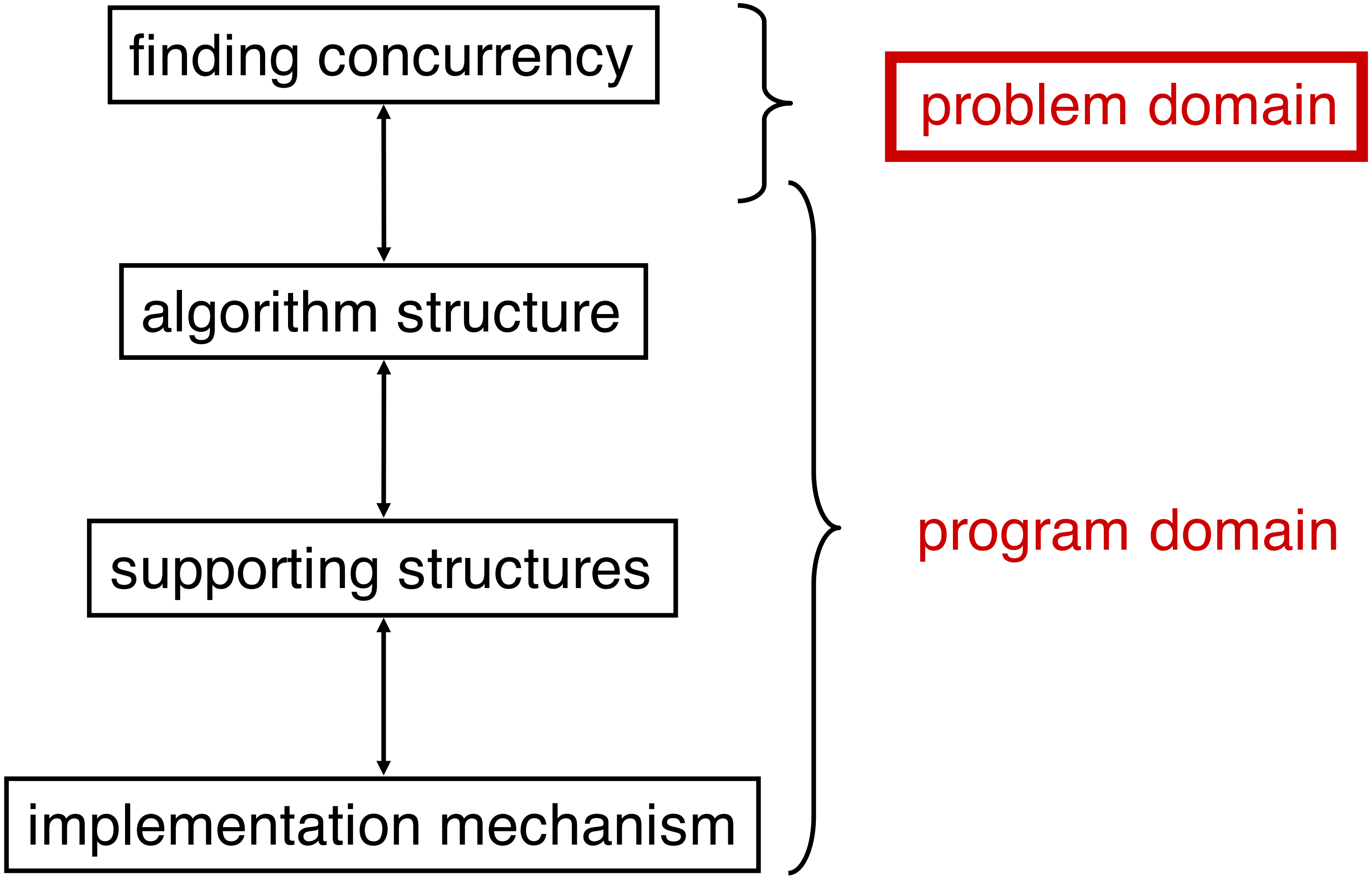
program domain

Finding Concurrency



- typically an iterative process
- opportunities exist for dependence analysis to play earlier role in decomposition

Design Spaces



Algorithm

TECHNISCHE UNIVERSITÄT DARMSTADT

- a step by step procedure that is guaranteed to terminate, such that each step is precisely stated and can be carried out by a computer
  - definiteness: the notion that each step is precisely stated
  - effective computability: each step can be carried out by a computer
  - finiteness: the procedure terminates

- multiple algorithms can be used to solve the same problem
  - some require fewer steps and some exhibit more parallelism

Algorithm Structure

Organize by Tasks
- Task Parallelism
- Divide and Conquer

Organize by Data
- Geometric Decomposition
- Recursive Data Decomposition

Organize by Flow
- Pipeline
- Event Condition

- important to re-evaluate the design
  - especially suitability for the target platform

Algorithm Structure – Main Forces

- efficiency
  - program should run quickly and make good use of resources
- simplicity
  - easy to understand code (develop, debug, verify, modify)
- portability
  - should run on a wide range of computers
  - lifetime of program typically longer than lifetime of computer
  - protects investment in software
- scalability
  - effective for a wide range of processing elements (PE)

Algorithm Structure – Main Forces

- possible conflicts
- efficiency vs. portability
  - using special hardware features yields efficient but not portable code
- efficiency vs. simplicity
  - e.g., task parallelism may require complicated scheduling

➔overall goals

➔balance between abstraction, portability and

➔suitability for a particular target platform

Algorithm Structure – Considerations

- target platform
  - should ideally not be necessary at that point in the development
  - but required to get efficient code

- order of magnitude of UEs
  - e.g., 10s or 1000s
- cost of sharing information between UEs
  - shared memory?
- programming environment
  - often multiple environments available for the platform

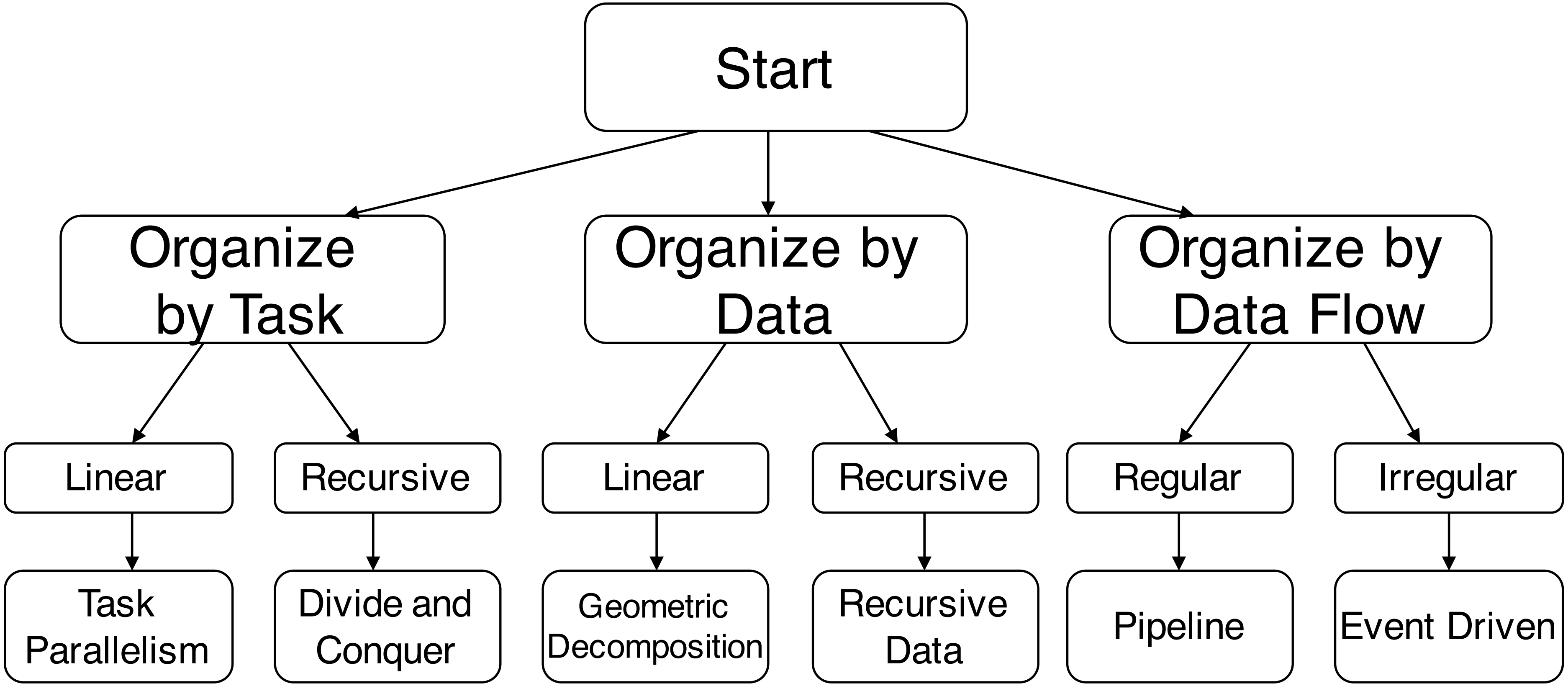
Algorithm Structure – Considerations

- major organizing principle
  - should emerge from the finding concurrency design space

- organization by tasks
  - only one group of tasks active at a time
  - interaction between tasks is major feature
  - e.g., embarrassingly parallel programs
- organization by data decomposition
  - e.g., update of a large data structure as main feature of the program
- organization by flow of data
  - e.g., continuous or discrete flow of data

➔ or a combination of the above

Choosing Algorithm Structure



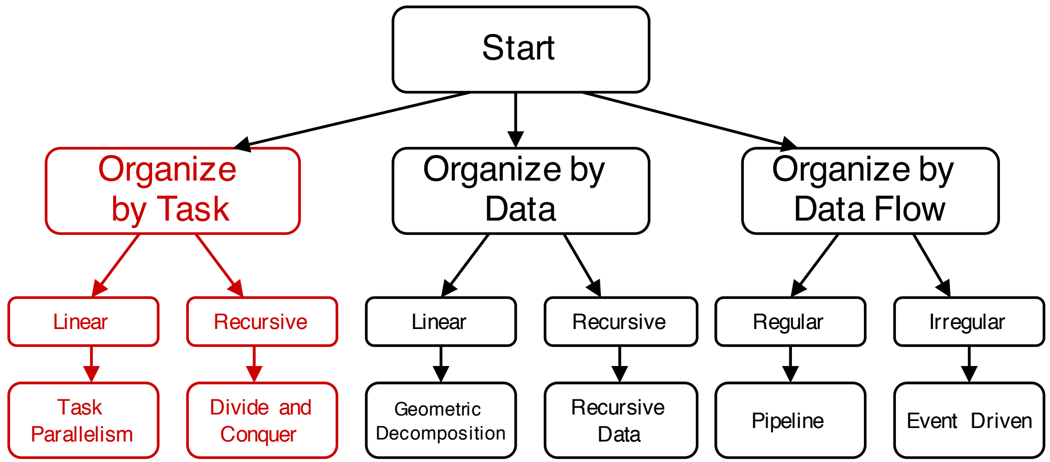
Choosing Algorithm Structure

- organize by task
- execution of tasks best organizing principle
- set of tasks enumerated linear in any number of dimensions

  ➔ task parallelism

- recursive task enumeration

  ➔ divide and conquer

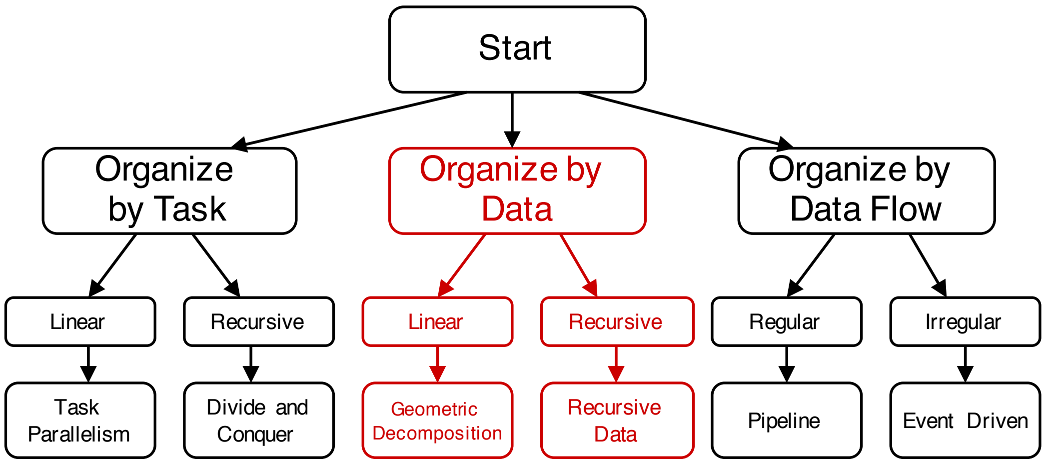


Choosing Algorithm Structure

- organize by data decomposition
- data decomposition best organizing principle
- program decomposed into discrete subspaces, solutions computed independently interacting with a small number of neighbors

  ➔ geometric decomposition
- problem defined as following links in a recursive data structure

  ➔ recursive data

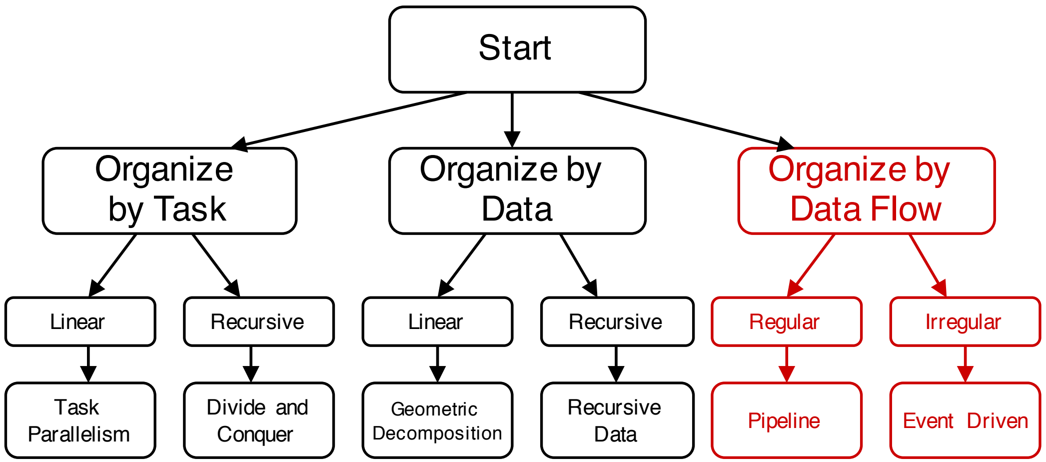


Choosing Algorithm Structure

- organize by flow of data
- flow of data imposes an ordering on groups of tasks
- regular, one-way flow of data which doesn't change during execution

  ➔ pipeline
- irregular, dynamic, not predictable flow of data

  ➔ event driven

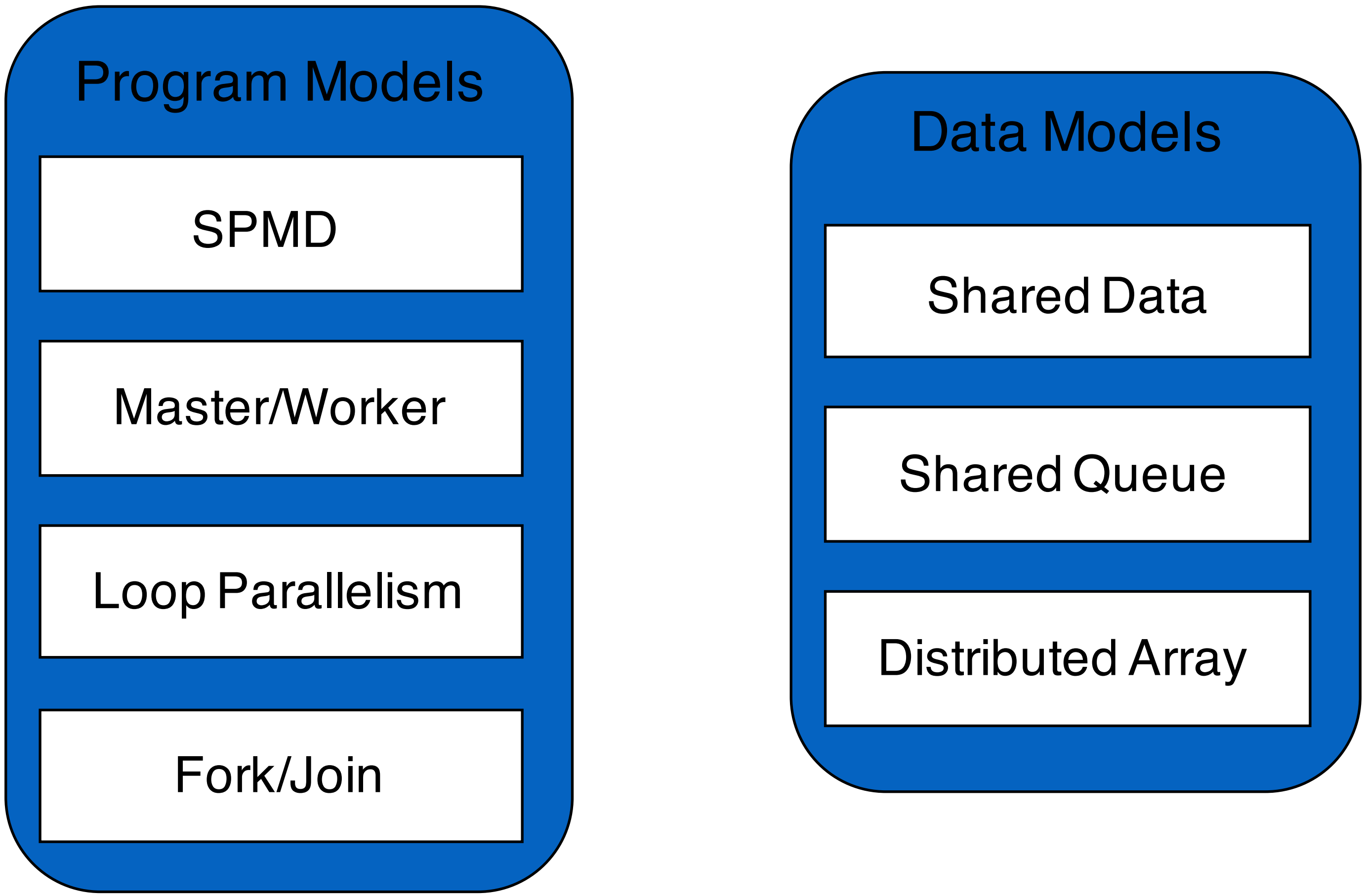


Algorithm Structure

Organize by Tasks	Organize by Data	Organize by Flow
Task Parallelism	Geometric Decomposition	Pipeline
Divide and Conquer	Recursive Data Decomposition	Event Condition

- important to re-evaluate the design
  - especially suitability for the target platform

Supporting Structures



- supporting structures not necessarily mutually exclusive

Program Models

- SPMD (Single Program, Multiple Data)
  - all PE's execute the same program in parallel, but each has its own data
  - each PE uses a unique ID to access its portion of data
  - different PE can follow different paths through the same code
  - essentially the CUDA grid model
  - SIMD/SIMT are special cases, SIMT corresponds to a CUDA warp

- master/worker
  - master creates a set of worker processes/threads and a bag of tasks
  - tasks are processed by the workers

Program Models

- loop parallelism
  - runtime of serial program dominated by a set of compute-intensive loops
  - different iterations of the loop are executed in parallel

- fork/join
  - main UE forks off other UEs working in parallel
  - often forking UE waits for other UEs to terminate and join

➔ master/worker can be implemented using the SPMD or fork/join pattern

➔ patterns not exclusive, not unique

➔ describe major idioms used by experienced programmers

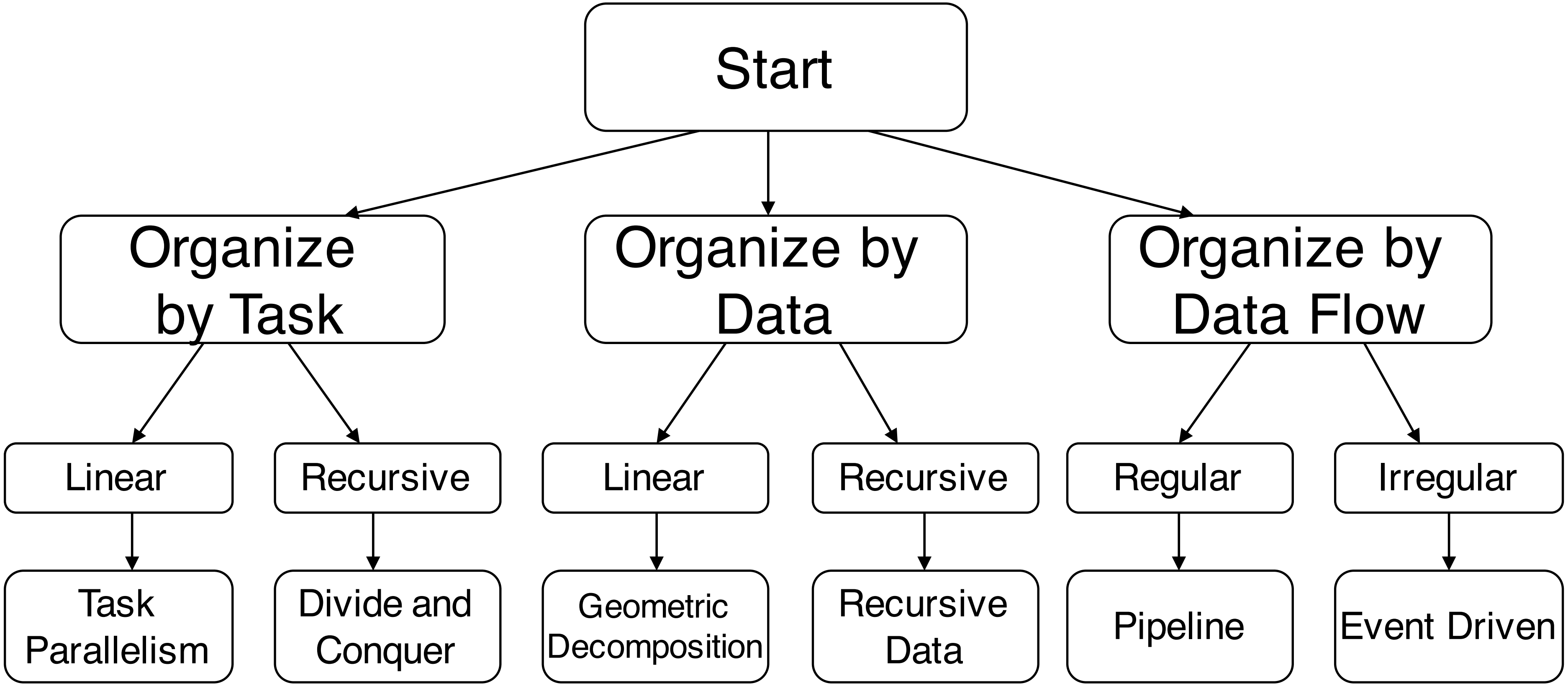
Data Structures

- shared data
  - all threads share a major data structure
  - this is what CUDA supports
  - general problem of handling shared data
  - correctness and performance issues
- shared queue
  - all threads see a “thread safe” queue that maintains ordering of data communication
- distributed array
  - decomposed and distributed among threads
  - limited support in CUDA Shared Memory

Supporting Structures: Forces

- clarity of abstraction
  - algorithm clearly apparent from the source code?
- scalability
- efficiency
- maintainability
- environment affinity
  - does the program fit the hardware and programming environment
- sequential equivalence
  - equivalent results to sequential execution if executed on many UEs?
  - relationship sequential/parallel clear?

Review: Algorithm Structure



Algorithm Structures vs. Coding Styles

	SPMD	loop parallelism	master/ worker	fork/join
task parallelism	☺☺☺☺	☺☺☺☺	☺☺☺☺	☺☺
divide and conquer	☺☺☺	☺☺	☺☺	☺☺☺☺
geometric decomposition	☺☺☺☺	☺☺☺	☺	☺☺
recursive data	☺☺		☺	
pipeline	☺☺☺		☺	☺☺☺☺
event-based coordination	☺☺		☺	☺☺☺☺

CUDA

Source: Mattson et al.

Design Spaces (and Design Patterns)

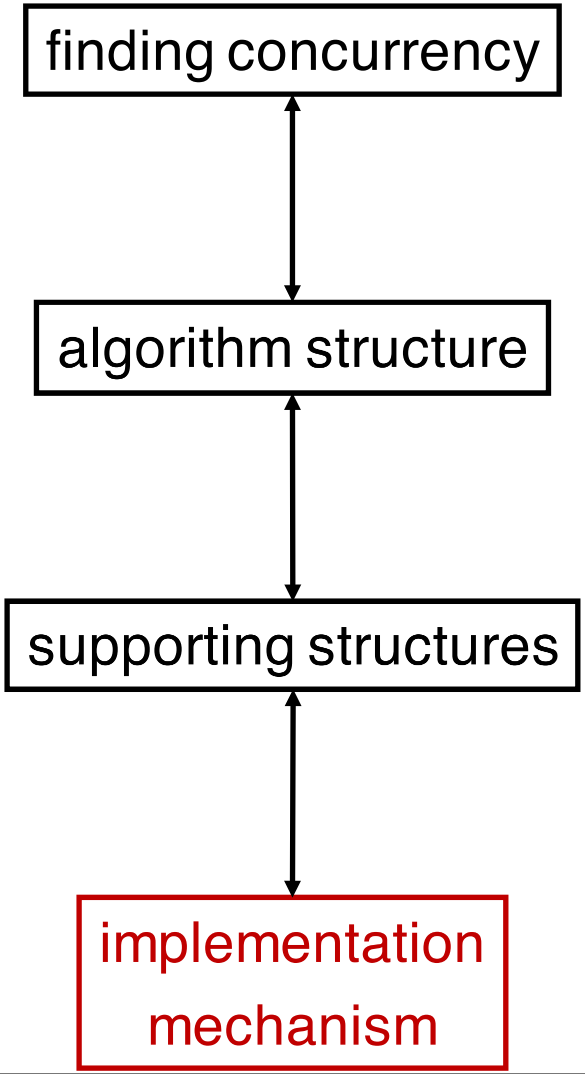
TECHNISCHE UNIVERSITÄT DARMSTADT

- implementation mechanisms
  - specific software constructs for implementing a parallel program

➔UE management

➔synchronization

➔communication



GCC Graphics, Capture and Massively Parallel Computing

Implementation Mechanisms

TECHNISCHE UNIVERSITÄT DARMSTADT

- low level operations unique to parallel programming
- only small subset used by most programmers

- UE management
  - how processes and threads are created, managed, destroyed
- synchronization
  - enforce ordering between events
  - correct access to shared data structures
- communication
  - information exchange between UEs

➔ depends highly on the target platform

GCC Graphics, Capture and Massively Parallel Computing

Implementation Mechanisms

- UE management
  - process: heavyweight object with its own state/context
  - thread: lightweight object, part of a process
  - CUDA threads: extremely lightweight

- device threads handled in CUDA as kernel
- host threads in CUDA using, e.g., cutStartThread
  - maps to native host threads

Implementation Mechanisms

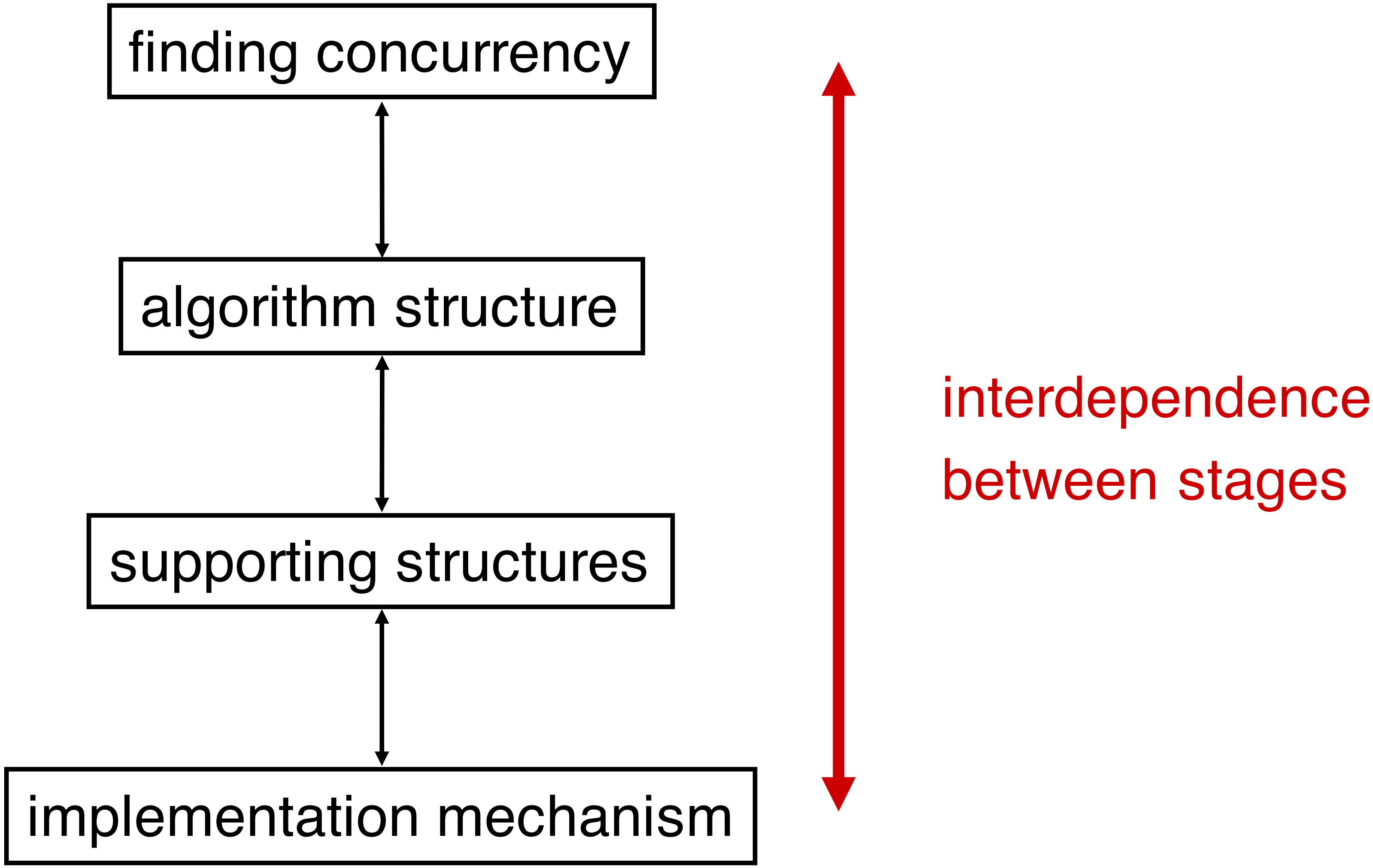
- synchronization
- memory synchronization for shared memory access
  - ensure that different threads see the same memory content
  - fences
  - CUDA:
    - volatile shared memory
    - syncthreads command (barrier synchronization)
- barrier synchronization
  - all UEs must arrive at this point before proceeding with computation
  - CUDA: e.g., syncthreads command
- mutual exclusion
  - only one UE can process a critical section
  - CUDA: e.g., atomic operations

Implementation Mechanism

- communication
- message passing between pairs of UEs
- collective communication
  - broadcast
    - single message sent to all UEs
  - barrier
    - synchronization mechanism that can be implemented using collective communiation
  - reduction
    - combine collection of results from multiple UEs

Design Spaces

TECHNISCHE UNIVERSITÄT DARMSTADT



GCC
Graphics, Capture and Massively Parallel Computing

Recommended Reading

- Chapters 3, 4, 5, and 6 of
  Patterns for Parallel Programming
  by Mattson, Sanders, and Massingill
  - finding concurrency
  - algorithm structure
  - supporting structures
  - implementation mechanisms