Concepts and Models of Parallel and Data-centric Programming

Foundations II

Lecture, Summer 2020

Dr. Christian Terboven <terboven@itc.rwth-aachen.de>

i12 High Performance Computing
RWTH AACHEN UNIVERSITY

Outline

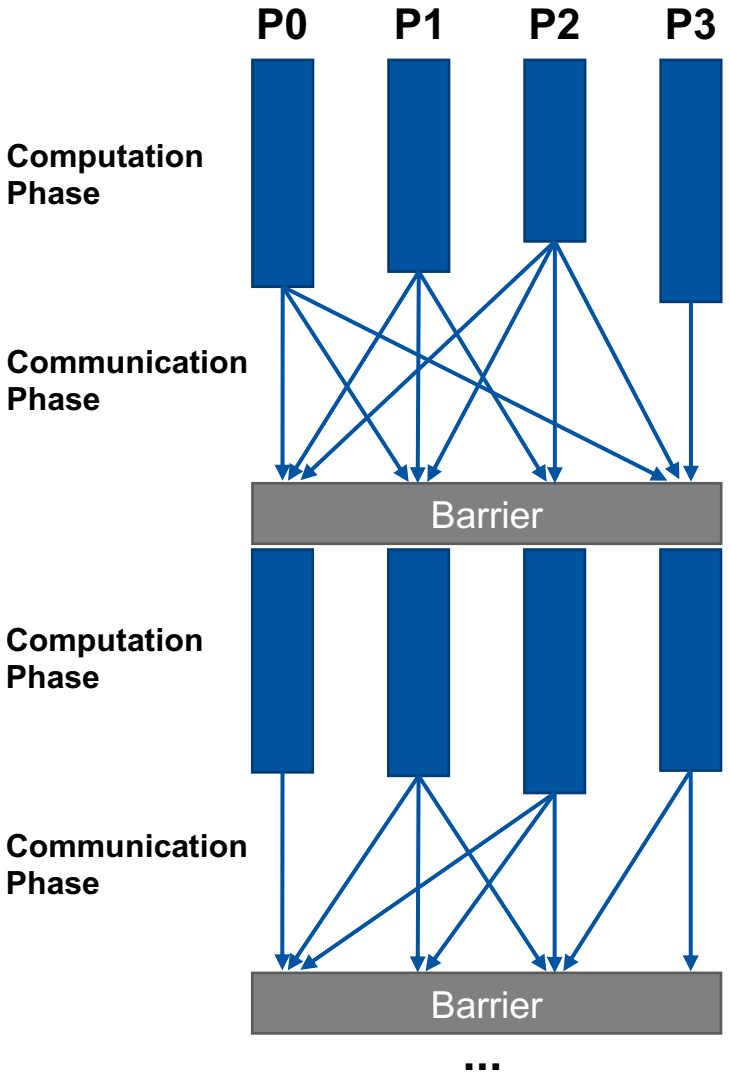
0. Organization
1. Foundations
2. Shared Memory
3. GPU Programming
4. Bulk-Synchronous Parallelism
5. Message Passing
6. Distributed Shared Memory
7. Parallel Algorithms
8. Parallel I/O
9. MapReduce
10. Apache Spark

a. Cluster Architecture
b. Convergence of HPC and Big Data
c. Parallel Programming Teasers
d. Harsh Realities

Parallel Programming Teasers: Classical HPC

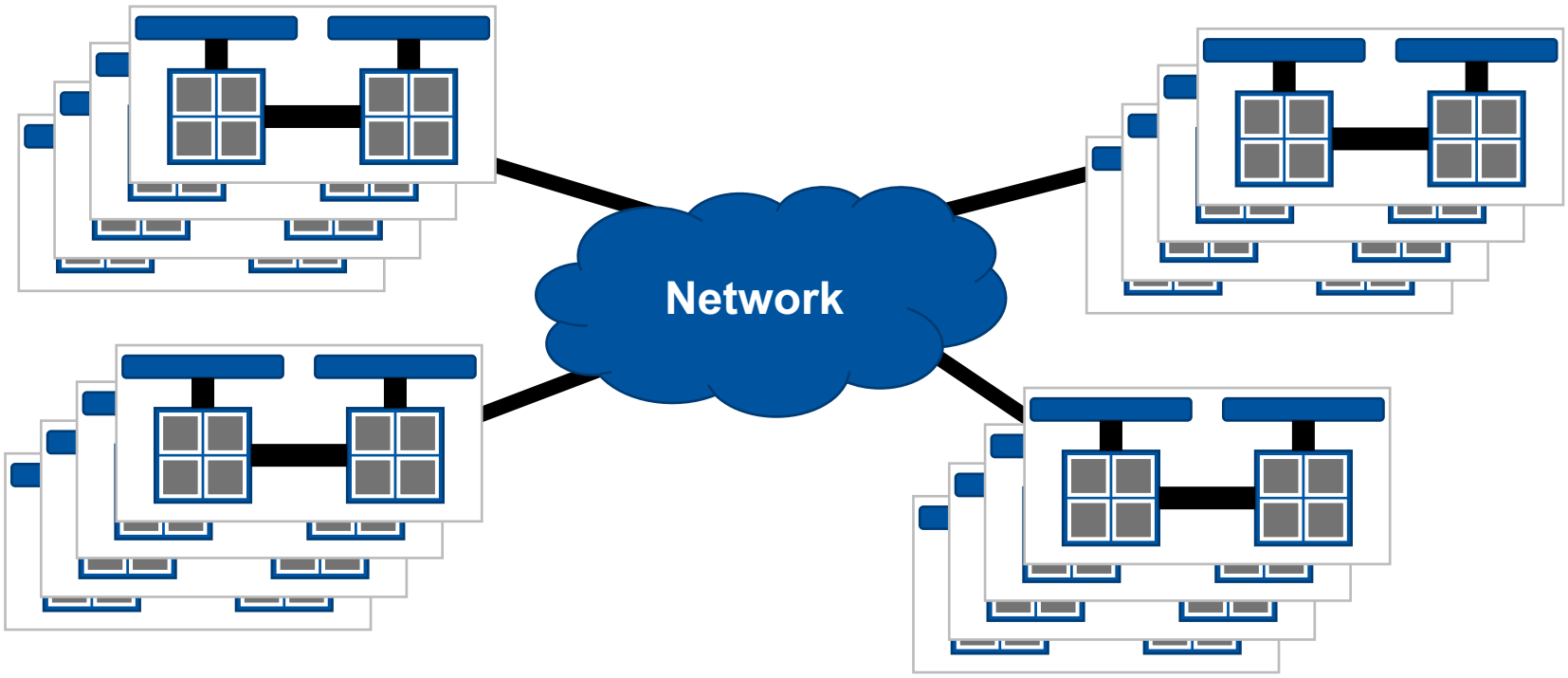
Bulk-Synchronous Parallel Computing

- Bulk-synchronous parallel (BSP) programming model
  - Computation phase: Perform local calculation with available data
  - Communication phase: Exchange results of computation between processors
- Structured design of parallel programs
- Design idea of BSP borrowed by different programming models
  - MapReduce, Google Pregel, Apache Hama



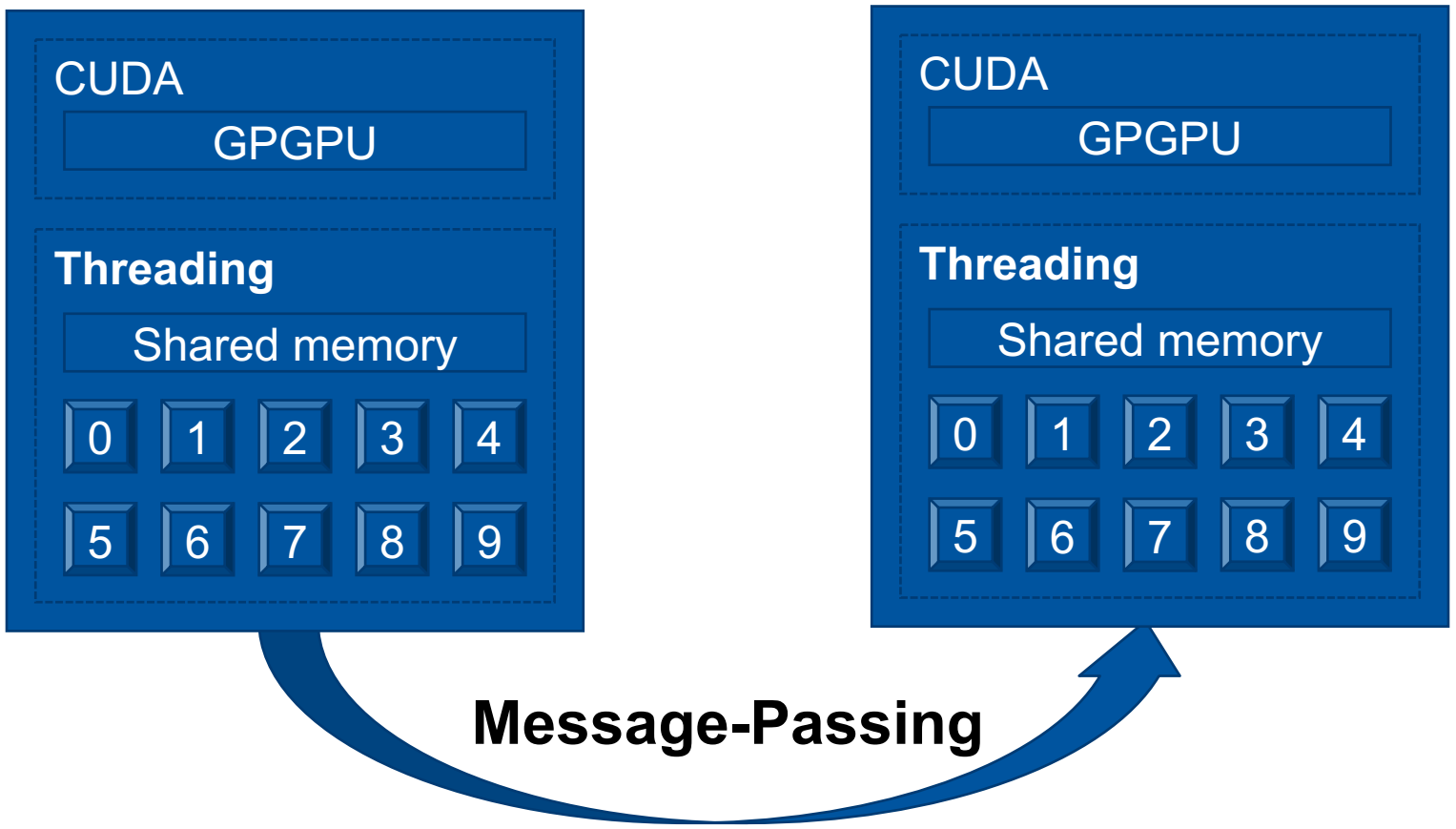
Motivation for hybrid programming

- Clusters of small supercomputers
  - Increasingly complex nodes – many cores, GPUs, Intel® Xeon Phi™, etc.



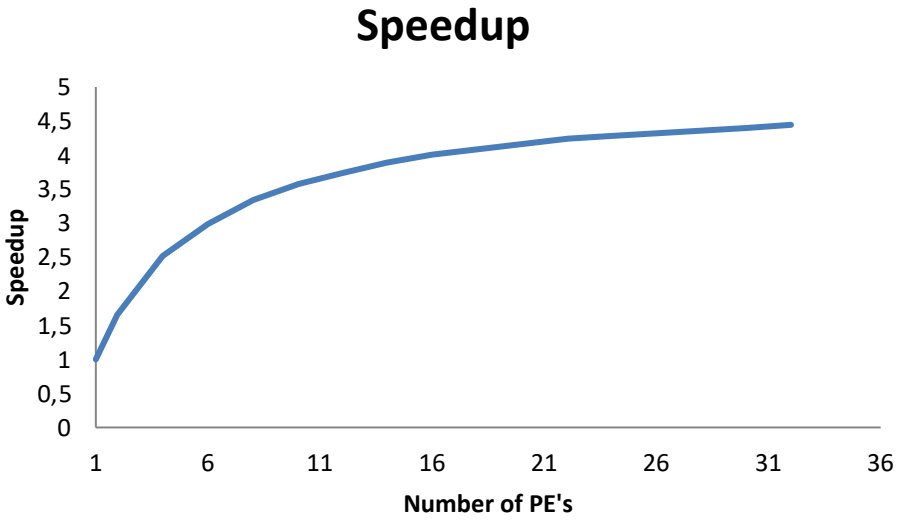
Hybrid programming

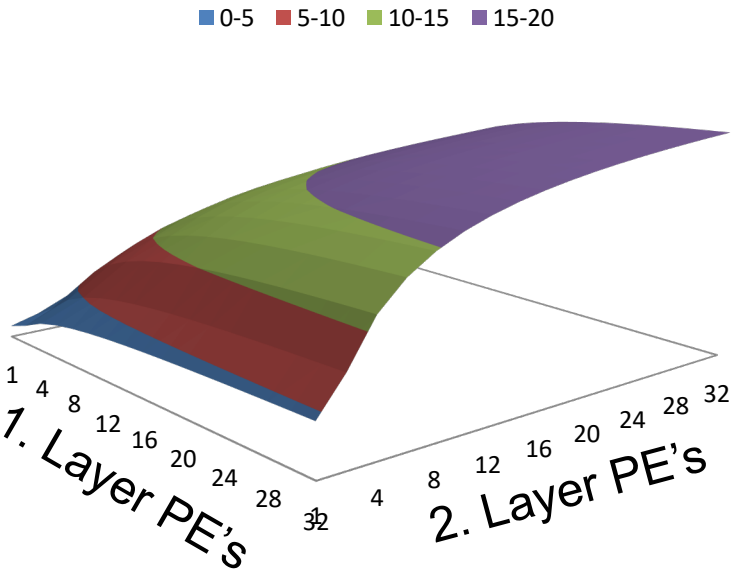
- (Hierarchical) mixing of different programming paradigms



Motivation for hybrid programming

- Speedup with Amdahl's Law:
- Hybrid parallelization adds an additional layer of possible speedup:



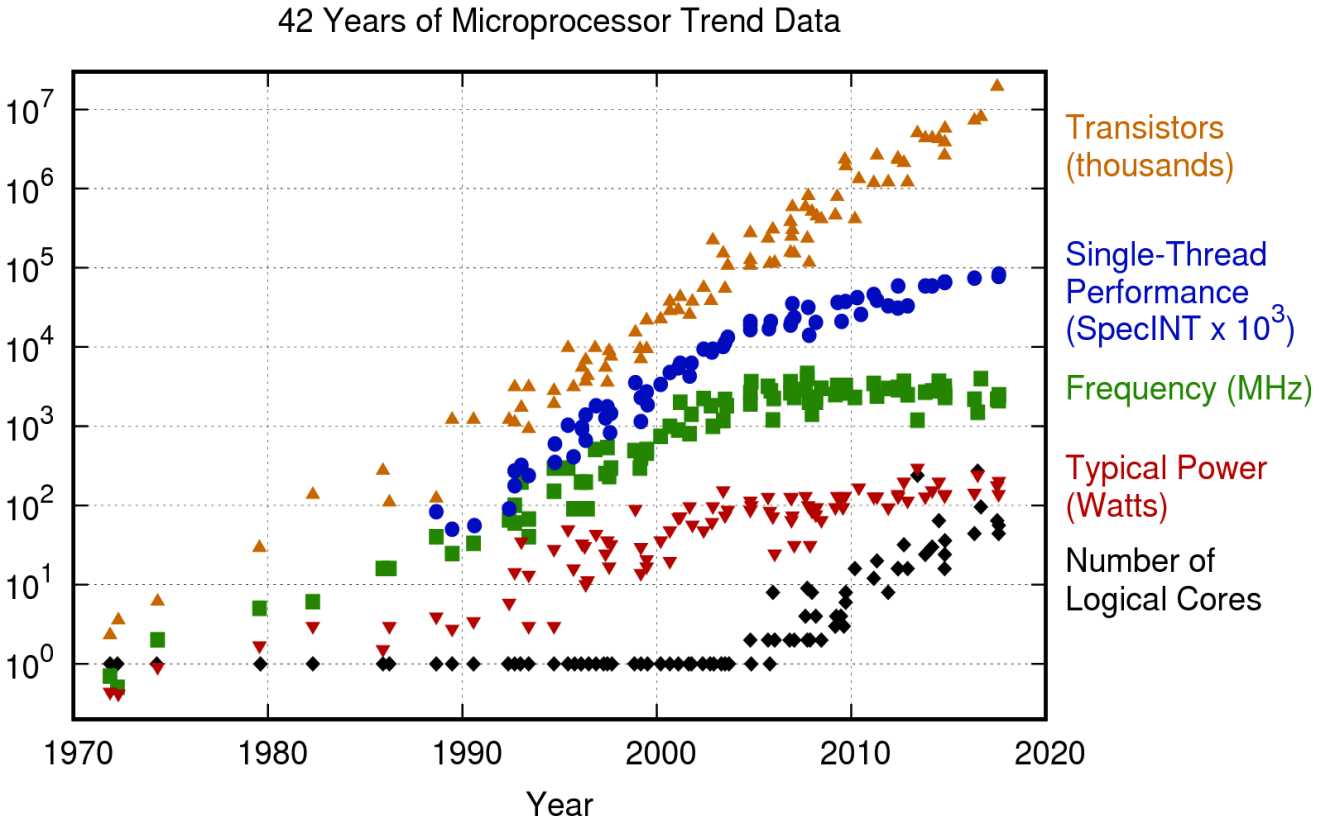


PE = Processing Element

Parallel Programming Teasers: Acceleration

Why Use Accelerators?

- CPU single core performance increases only slowly since ~2005
  - Total performance of chip increase through increasing number of logical cores (~power law)

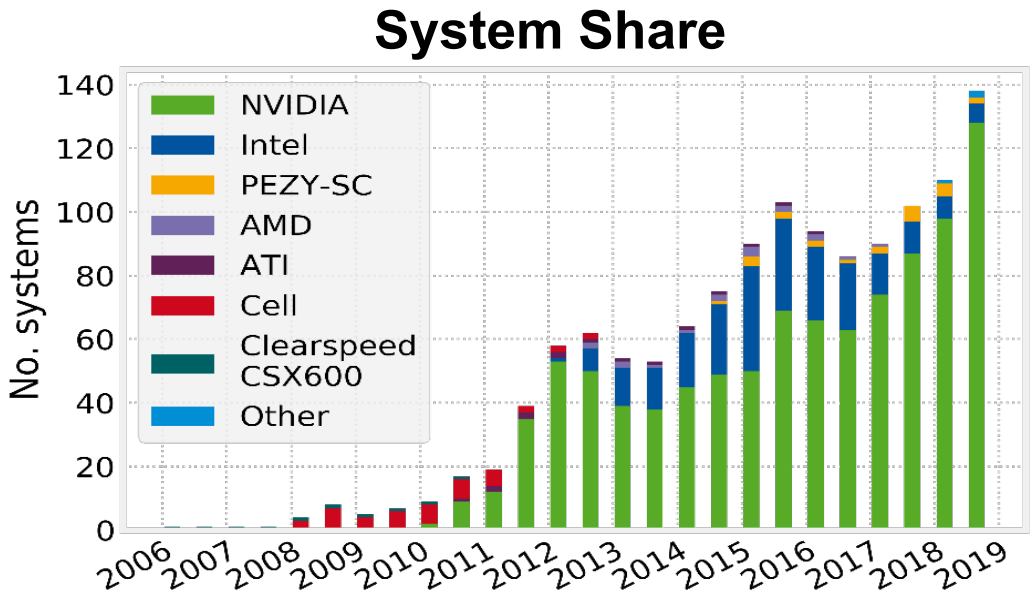


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

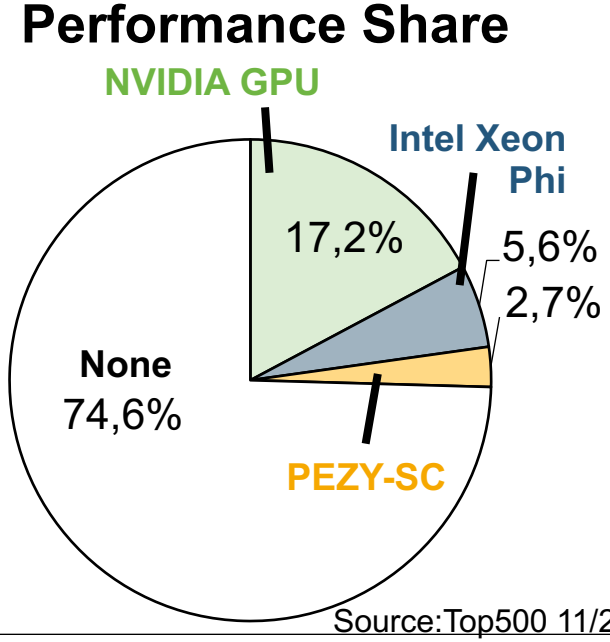
- Frequency stagnates
  - Power and cooling constraints

Why Use Accelerators?

- Demand for comp. capacity increases steadily, power is constrained
- Hardware accelerators: highly-efficient (performance per watt) for certain computing processes
  - GPGPUs (e.g. NVIDIA, AMD), FPGAs (e.g. Convey)
  - Intel Many Integrated Core (MIC) Arch. (Intel Xeon Phi)



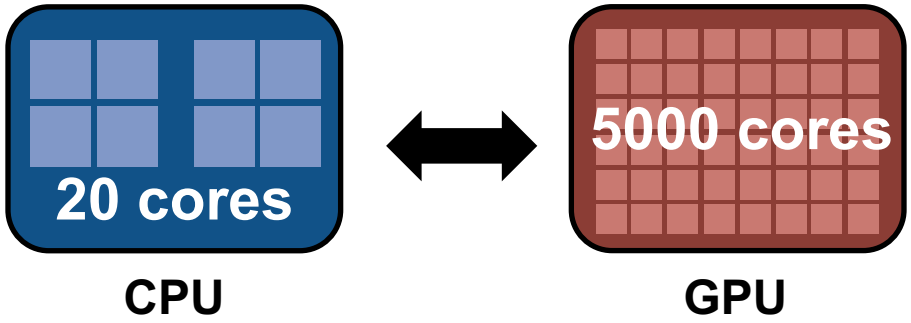
Source: Top500, 11/2018



Source:Top500 11/2017

How and When to Use Accelerators?

- How do accelerator architectures look like?
  - Why can accelerators deliver a good performance per Watt ratio?
  - Memory hierarchy
  - Execution Model
- Is any application suitable?
  - Performance modelling, leveraging parallelism
- How are accelerators programmed?
  - Programming models, libraries
- How to move data to accelerators?
  - Offloading of compute regions



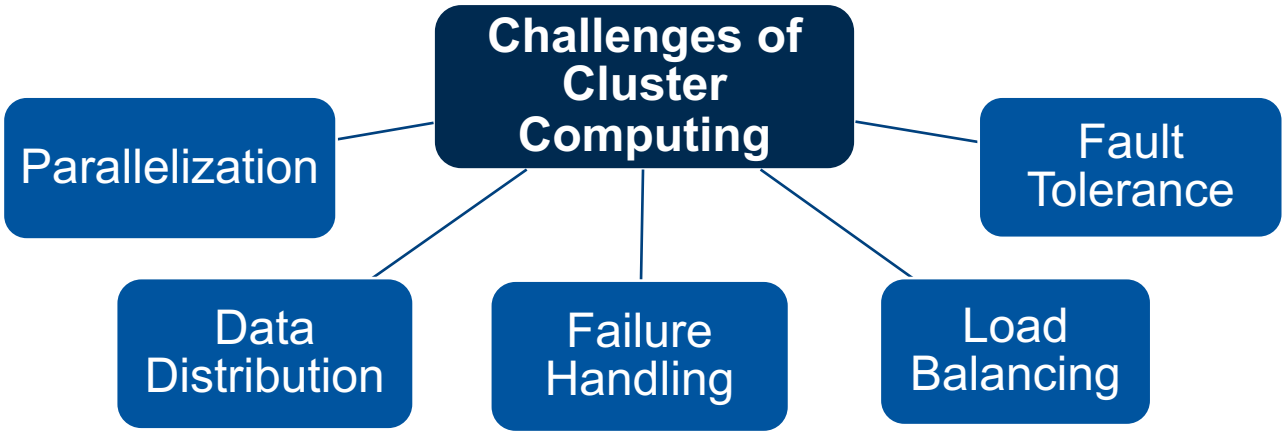
Differences CPU vs. GPU:

- CPU: optimized for low latencies
- GPU: massive but simple parallelism, high throughput

Parallel Programming Teasers: MapReduce

MapReduce

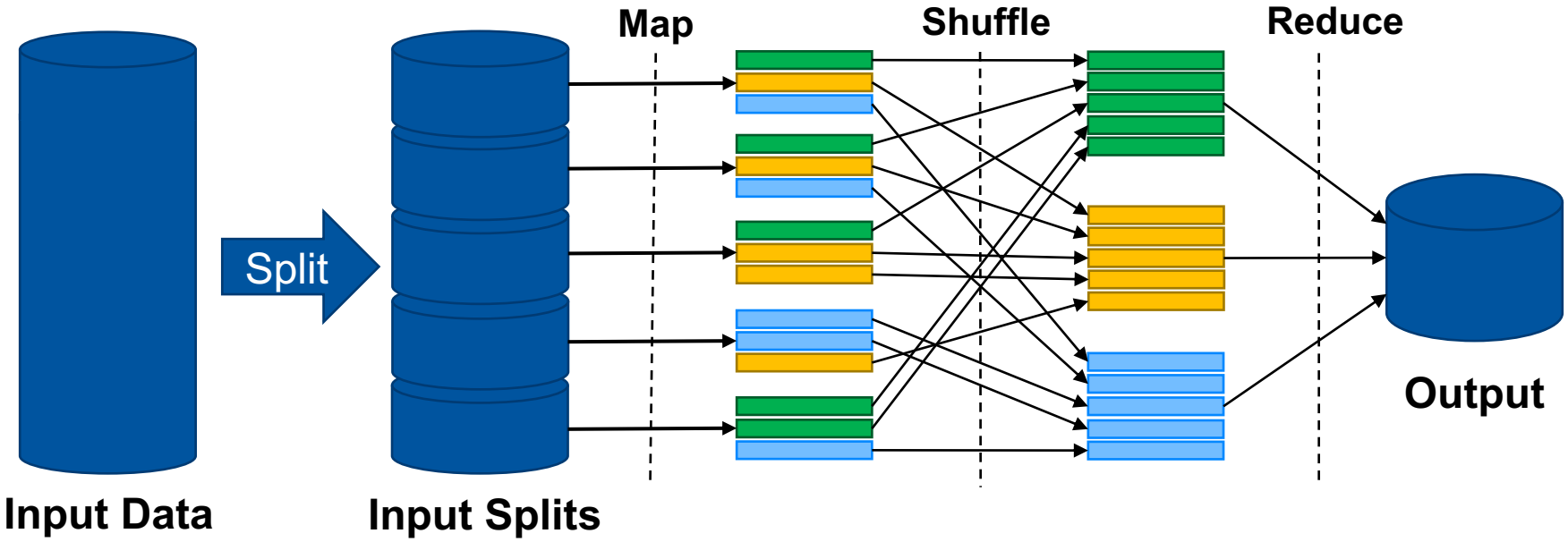
- Era of “Big Data”: Processing large input data
  - Data does not fit into main memory of single machine or small cluster
  - Distribution of computation across **huge** number of machines needed
- Problem: Cluster computing is complex



- Idea: Hide those challenges from developer by using an abstraction
- Approach: MapReduce programming model and framework

MapReduce Model

- Developer defines *Map* and *Reduce* function, rest done by framework
  - Distribute input data across worker nodes (splitting)
  - Map: Each worker node applies *Map* function to local data
  - Shuffle: Worker nodes redistribute and group data based on output keys
  - Reduce: Worker nodes apply *Reduce* function to each group of output data



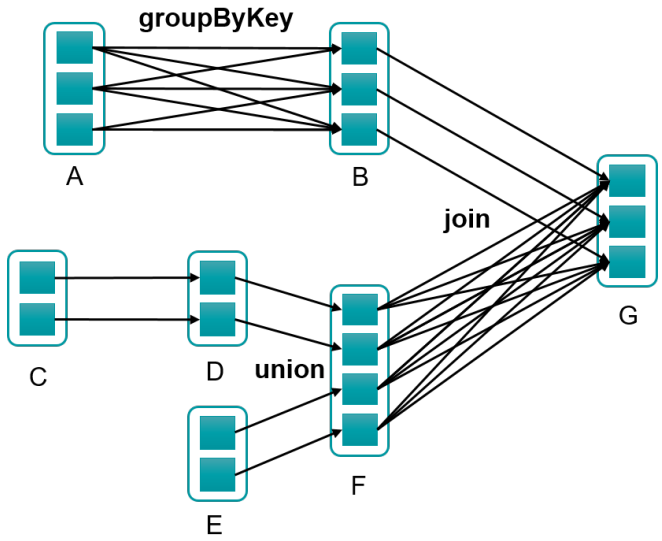
MapReduce – Applications

- Most prominent implementation: Apache Hadoop
  - Simplicity, scalability, fault tolerance
- Focus on data science: Large-scale and data-intensive tasks
  - Indexing for search engines
  - Finding popular search queries (e.g., Google Trends)
  - Clustering of news articles (e.g., Google News)
  - Machine learning: Classification, clustering, …

Parallel Programming Teasers: Apache Spark

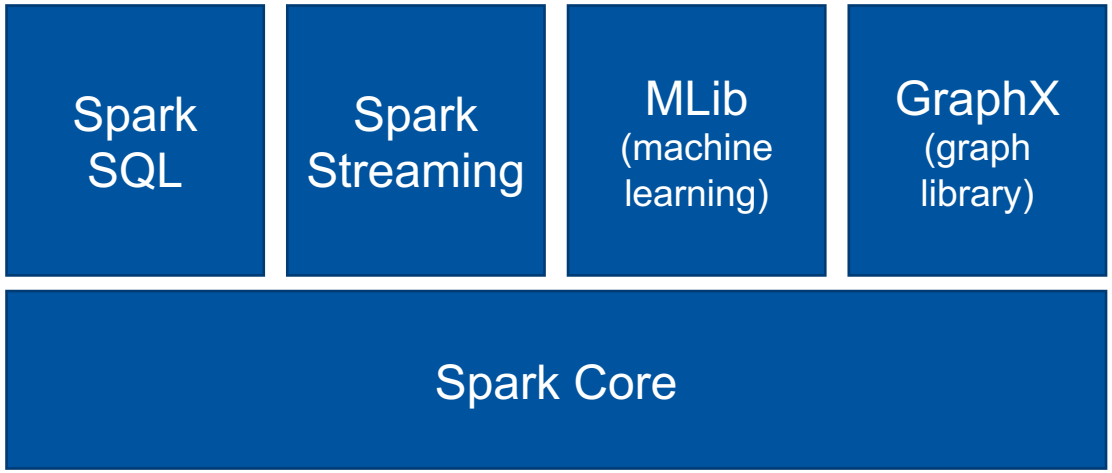
Apache Spark

- Generalization of MapReduce approach
- Focus on iterative jobs and interactive analyses on clusters
- Central concept: Resilient Distributed Datasets (RDDs)
  - Enable caching data in memory for reuse, maintain fault tolerance
- Various set of functions applicable to RDDs
  - Not limited to *Map* and *Reduce* functions, higher flexibility



Apache Spark – Framework

- Spark Core provides fundamental functionalities (RDDs, transformations, …)
- Several abstractions build upon the core functionalities
- Spark Core and abstractions form the Apache Spark project



What you have learnt

What you have learnt

- A 10 mile high view of parallel programming

- See Moodle for quiz!