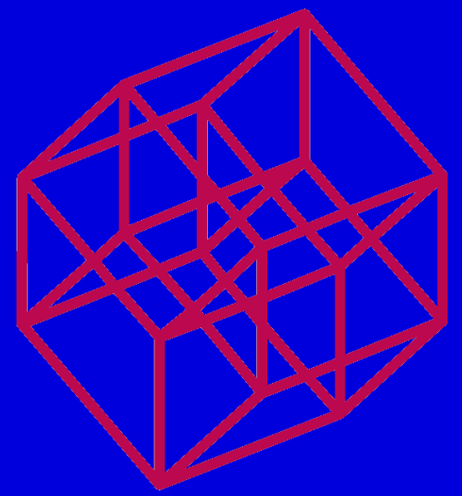
MUNI
ÚVT

Apache Hadoop & others general introduction

Open-Source Solution for Huge Data Sets



Tomáš Rebok

Masaryk University

Slides inspired by: Zheng Shao

What is Apache Hadoop?

- Open-source software framework designed for storage and processing of large-scale data on clusters of commodity hardware

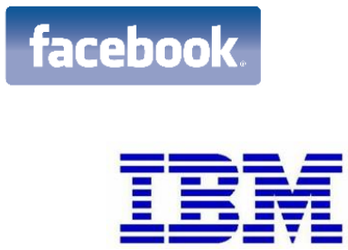
- Created by Doug Cutting and Mike Carafella in 2005.

- Cutting named the program after his son’s toy elephant.

Uses for Hadoop

- Data-intensive text processing
- Assembly of large genomes
- Graph mining
- Machine learning and data mining
- Large scale social network analysis

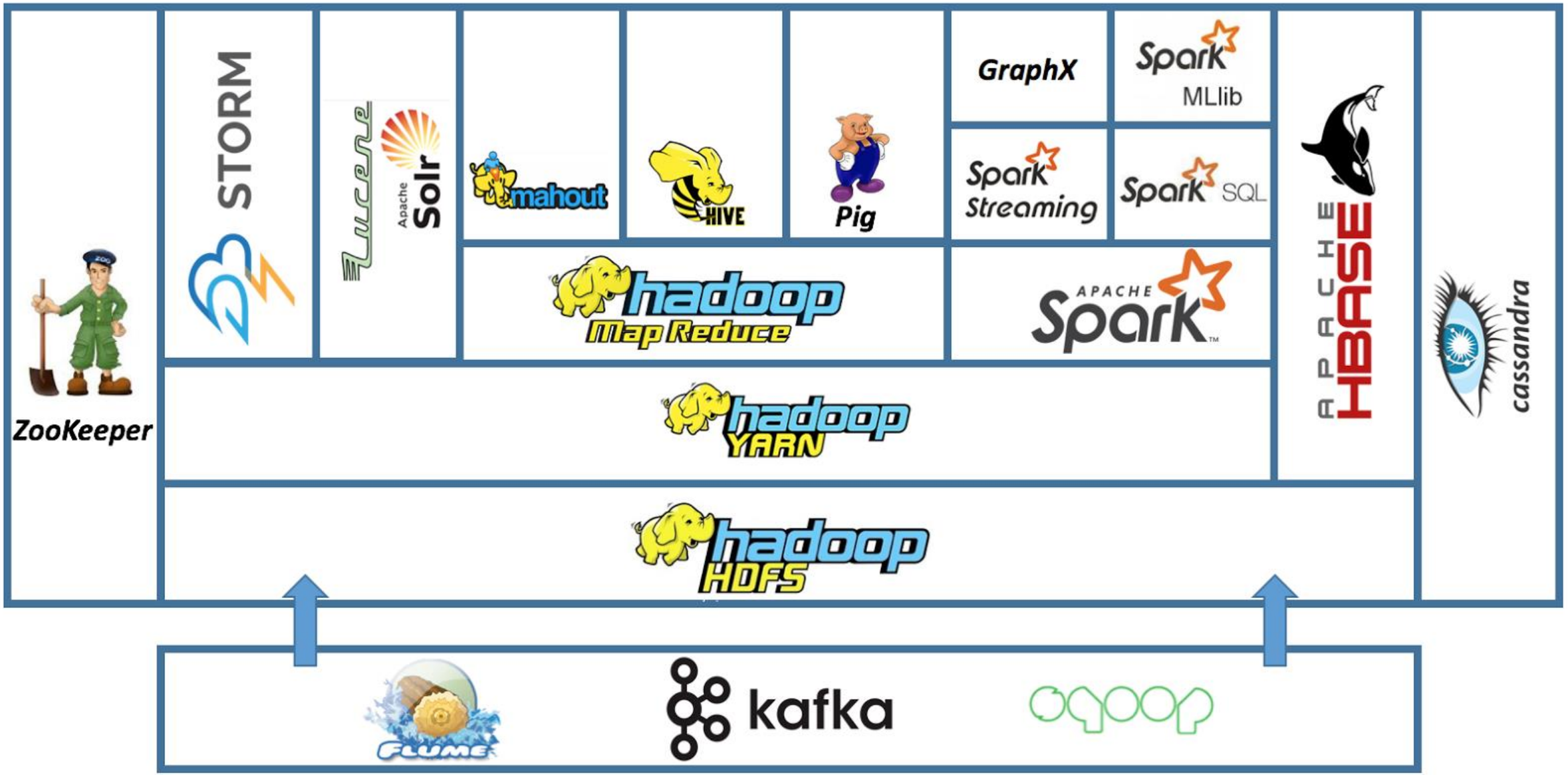
Who Uses Hadoop?



The Hadoop Ecosystem

Hadoop Common	• Contains Libraries and other modules
HDFS	• Hadoop Distributed File System
Hadoop YARN	• Yet Another Resource Negotiator
Hadoop MapReduce	• A programming model for large scale data processing

The Hadoop Ecosystem



Motivations for Hadoop

What considerations led to its design

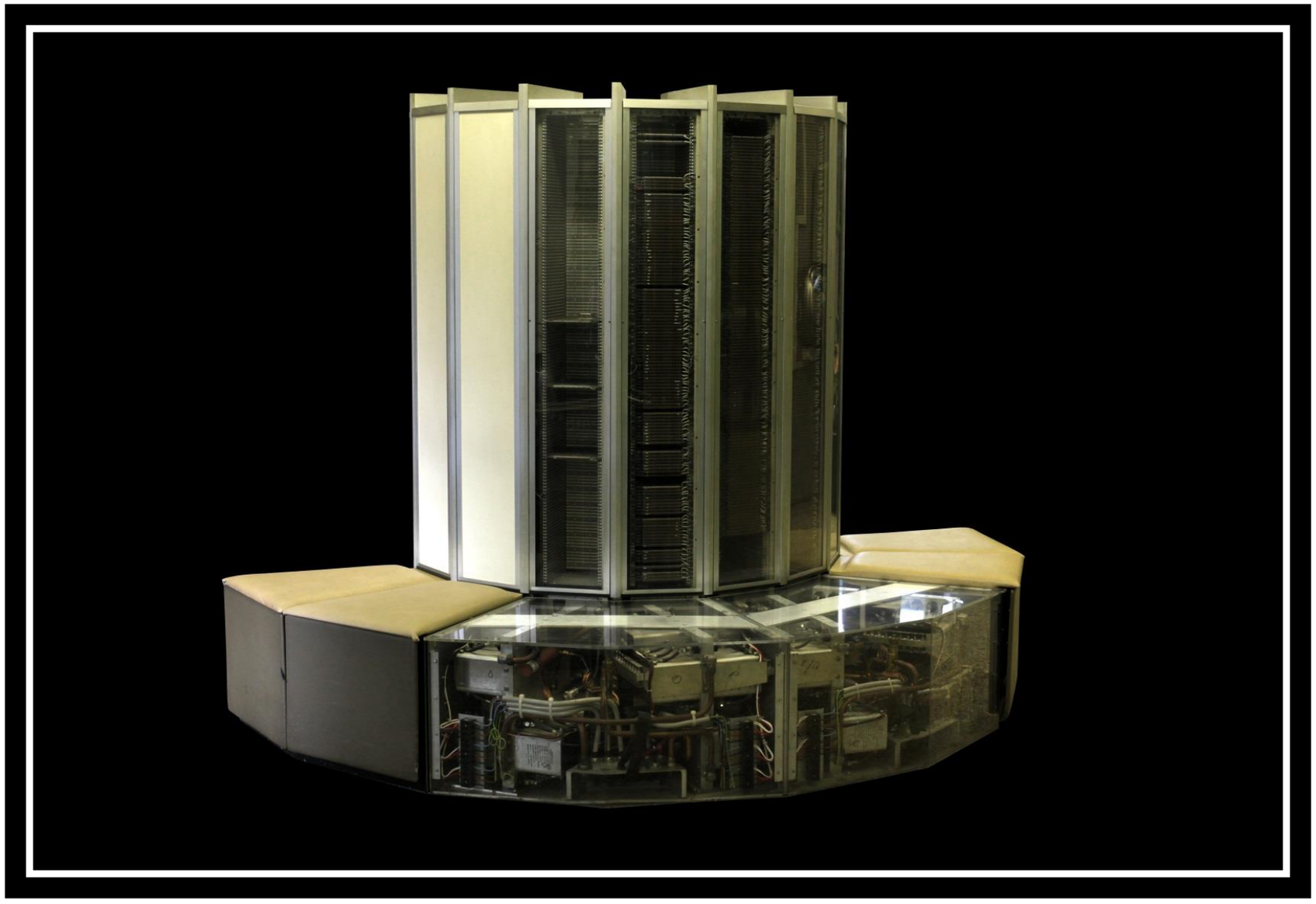
Motivations for Hadoop

- What were the limitations of earlier large-scale computing?
- What requirements should an alternative approach have?
- How does Hadoop address those requirements?

Early Large Scale Computing

- Historically computation was processor-bound
  - Data volume has been relatively small
  - Complicated computations are performed on that data
- Advances in computer technology has historically centered around improving the power of a single machine

Cray-1



Advances in CPUs

- Moore’s Law
  - The number of transistors on a dense integrated circuit doubles every two years
- Single-core computing can’t scale with current computing needs

Single-Core Limitation

- Power consumption limits the speed increase we get from transistor density

Distributed Systems

- Allows developers to use multiple machines for a single task

Distributed System: Problems

- Programming on a distributed system is much more complex
  - Synchronizing data exchanges
  - Managing a finite bandwidth
  - Controlling computation timing is complicated
  - Communication overhead
  - …

Distributed System: Problems

MUNI
ÚVT

"You know you have a distributed system when the crash of a computer you've never heard of stops you from getting any work done." – Leslie Lamport

- Distributed systems must be designed with the expectation of failure

Distributed System: Data Storage

- Typically divided into Data Nodes and Compute Nodes
- At compute time, data is copied to the Compute Nodes
- Fine for relatively small amounts of data

- Modern systems deal with far more data than was gathering in the past

How much data?

- Facebook
  - 500 TB per day
- Yahoo
  - Over 170 PB
- eBay
  - Over 6 PB

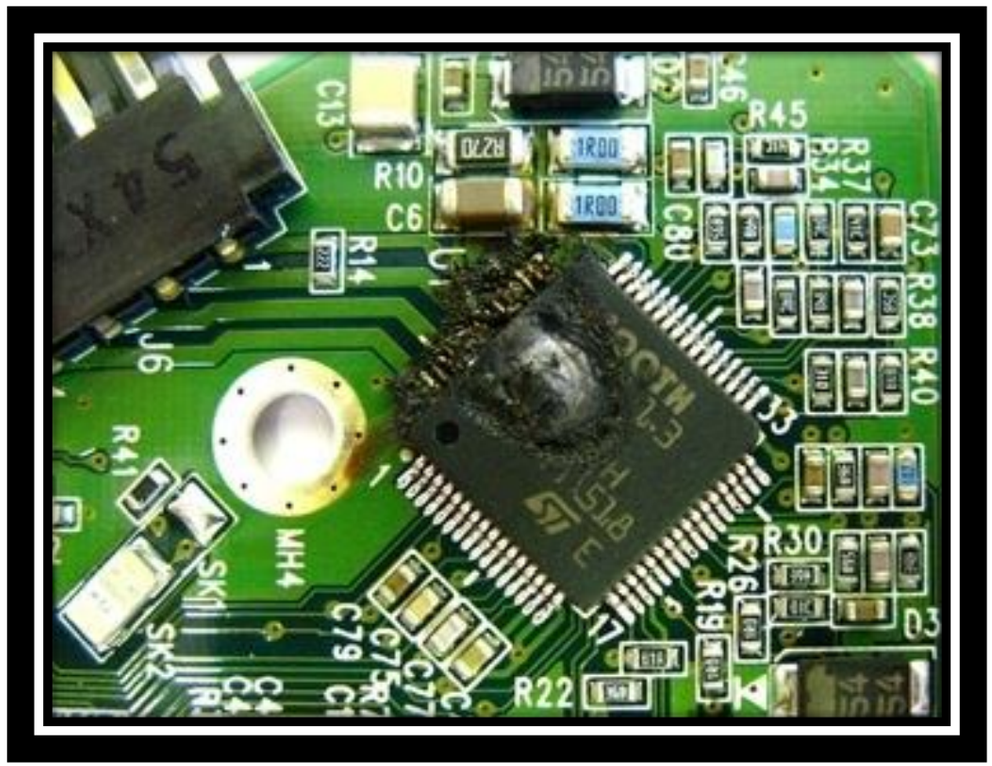
- Getting the data to the processors becomes the bottleneck

Requirements for Hadoop

- Must support partial failure
- Must be scalable

Partial Failures

- Failure of a single component must not cause the failure of the entire system only a degradation of the application performance
  - Failure should not result in the loss of any data



Component Recovery

- If a component fails, it should be able to recover without restarting the entire system
- Component failure or recovery during a job must not affect the final output

Scalability

- Increasing resources should increase load capacity
- Increasing the load on the system should result in a graceful decline in performance for all jobs
  - Not system failure

Hadoop

- Based on the work done by Google in the early 2000s
  - “The Google File System” in 2003
  - “MapReduce: Simplified Data Processing on Large Clusters” in 2004
- The core idea was to distribute the data as it is initially stored
  - Each node can then perform computation on the data it stores without moving the data for the initial processing

Core Hadoop Concepts

- Applications are written in a high-level programming language
  - No network programming or temporal dependency
- Nodes should communicate as little as possible
  - A “shared nothing” architecture
- Data is spread among the machines in advance
  - Perform computation where the data is already stored as often as possible

High-Level Overview

- When data is loaded onto the system it is divided into blocks
  - Typically 64MB or 128MB
- Tasks are divided into two phases
  - Map tasks which are done on small portions of data where the data is stored
  - Reduce tasks which combine data to produce the final output
- A master program allocates work to individual nodes

Fault Tolerance

- Failures are detected by the master program which reassigns the work to a different node
- Restarting a task does not affect the nodes working on other portions of the data
- If a failed node restarts, it is added back to the system and assigned new tasks
- The master can redundantly execute the same task to avoid slow running nodes

Hadoop Overview

MUNI
ÚVT

- **Open Source Apache Project**
  - http://hadoop.apache.org/
  - Book: http://oreilly.com/catalog/9780596521998/index.html
- **Written in Java**
  - Does work with other languages
- **Runs on**
  - Linux, Windows and more
  - Commodity hardware with high failure rate

Current Status of Hadoop

MUNI
ÚVT

- **Largest Cluster** (Facebook)
  - 2000 nodes (1200 nodes with 8 cores + 800 nodes with 16 cores)
  - 21 PB HDFS storage (12 TB per node)
  - 32 GB of RAM per node
- **Used by 40+ companies / universities over the world**
  - Yahoo, Facebook, etc
  - Cloud Computing Donation from Google and IBM
- **Startup focusing on providing services for Hadoop**
  - Cloudera

Hadoop Components

MUNI
ÚVT

- **Hadoop Distributed File System (HDFS)**
- **Hadoop Map-Reduce**
- **Contributes**
  - Hadoop Streaming
  - Pig / JAQL / Hive
  - HBase
  - Hama / Mahout
  - …

MUNI
ÚVT

Hadoop Distributed File System (HDFS)

A distributed file system designed to run on commodity hardware

Overview

- Responsible for storing data on the cluster
- Data files are split into blocks and distributed across the nodes in the cluster
- Each block is replicated multiple times

HDFS Basic Concepts

MUNI
ÚVT

- HDFS is a file system written in Java based on the Google's GFS
- Provides redundant storage for massive amounts of data

HDFS Basic Concepts

- HDFS works best with a smaller number of large files
  - Millions as opposed to billions of files
  - Typically 100MB or more per file

- Files in HDFS are write once

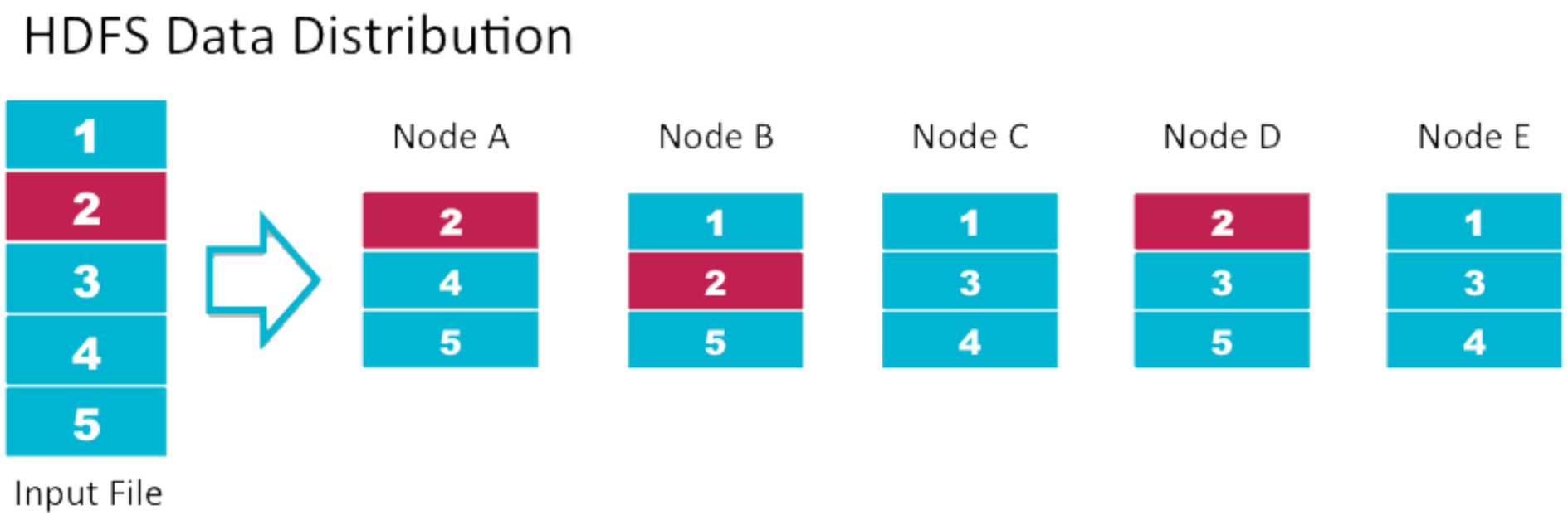
- Optimized for streaming reads of large files and not random reads

How are Files Stored

- Files are split into blocks
  - 128 MB by default
  - can be adapted based on requirements
- Blocks are split across many machines at load time
  - Different blocks from the same file will be stored on different machines
- Blocks are replicated across multiple machines
- The NameNode keeps track of which blocks make up a file and where they are stored

Data Replication

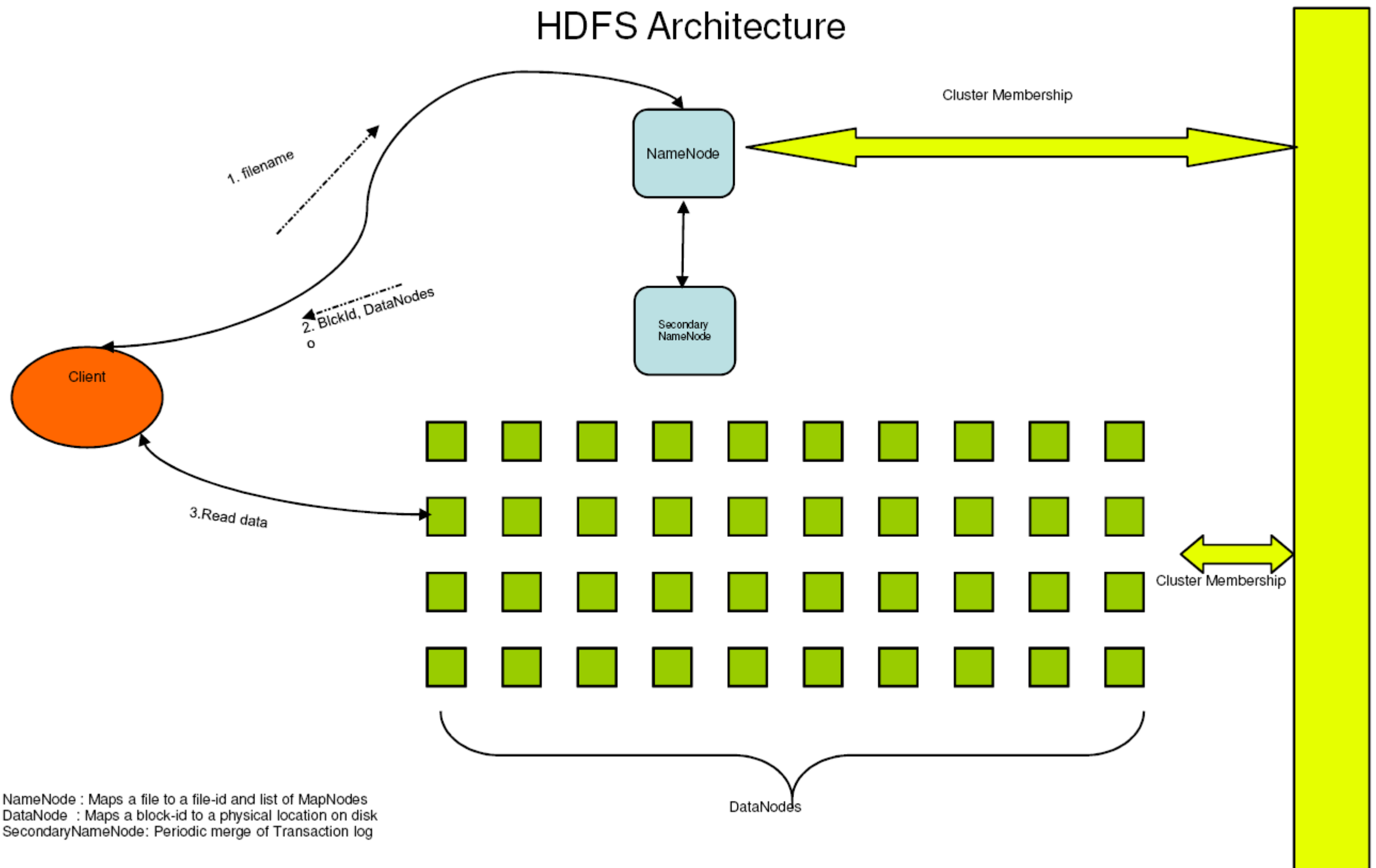
- Default replication is 3-fold



Data Retrieval

- When a client wants to retrieve data
  - Communicates with the NameNode to determine which blocks make up a file and on which data nodes those blocks are stored
  - Then communicated directly with the data nodes to read the data

HDFS Architecture



NameNode : Maps a file to a file-id and list of MapNodes
DataNode : Maps a block-id to a physical location on disk
SecondaryNameNode: Periodic merge of Transaction log

HDFS User Interface (client)

- **Java API**
- **Command Line**
  - bin/hdfs dfs -mkdir /foodir
  - bin/hdfs dfs -cat /foodir/myfile.txt
  - bin/hdfs dfs -cp /foodir/myfile.txt /foodir/myfile-backup.txt
  - bin/hdfs dfs -mv /foodir/myfile-backup.txt /foodir/myfile.txt
  - bin/hdfs dfs -rm /foodir/myfile.txt
  - bin/hdfs dfsadmin -report
  - bin/hdfs dfsadmin -decommission datanodename
- **Web Interface**
  - http://host:port/dfshealth.jsp

More about HDFS

MUNI
ÚVT

- http://hadoop.apache.org/core/docs/current/hdfs_design.html

- Hadoop FileSystem API
  - HDFS
  - Local File System
  - Kosmos File System (KFS)
  - Amazon S3 File System

MUNI
ÚVT

MapReduce

Distributing computation across nodes

MapReduce Overview

- A method for distributing computation across multiple nodes
  - Fits a lot of batch processing applications
    - Log processing
    - Web index building
- Each node processes the data that is stored at that node
- Consists of two main phases
  - Map
  - Reduce

MapReduce Features

- Automatic parallelization and distribution
- Fault-Tolerance
- Provides a clean abstraction for programmers to use

The Mapper

- Reads data as key/value pairs
  - The key is often discarded
- Outputs zero or more key/value pairs

Shuffle and Sort

- Output from the mapper is sorted by key
- All values with the same key are guaranteed to go to the same machine

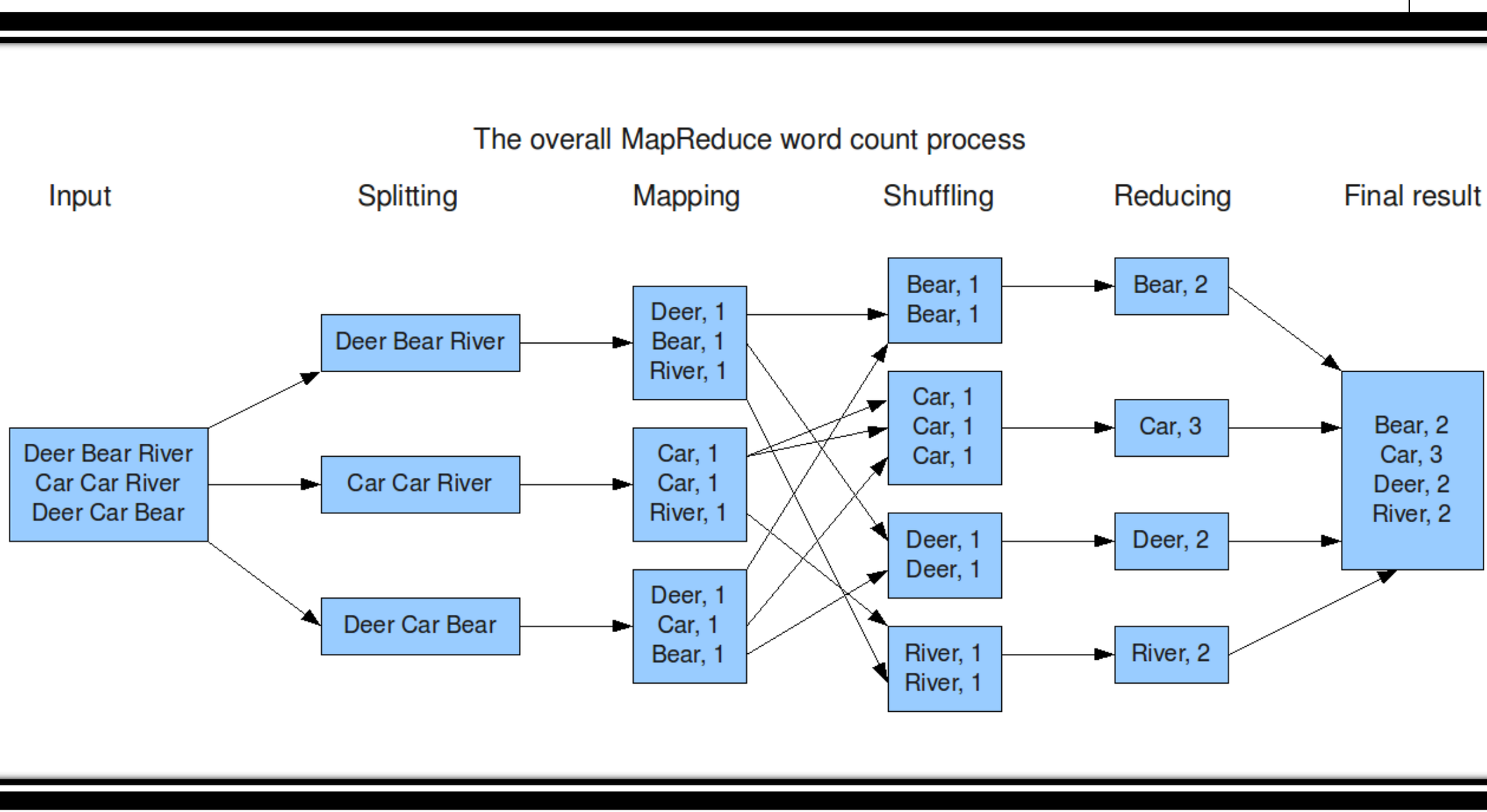
The Reducer

- Called once for each unique key

- Gets a list of all values associated with a key as input

- The reducer outputs zero or more final key/value pairs
  - Usually just one output per input key

MapReduce: Word Count



Map/Reduce Usage Examples

- When employed, **two principal questions have to be answered**:

1. **Which key-value pairs should be provided by Mappers?**

2. **How to combine values with the same keys during the Reduce stage?**

Map/Reduce Usage Examples

MUNI ÚVT

Example tasks:

1. From a CSV file, extract all the records having X value lower than 5.
2. Combine (sets union) entries of 2 CSV files into single one.
3. Perform set intersection of entries of 2 CSV files.
4. Given 2 CSV files X and Y. Perform set difference, ie. X-Y.

Hadoop Streaming

MUNI
ÚVT

- **Allow to write Map and Reduce functions in any languages**
  - because Hadoop Map/Reduce only accepts Java
  - makes development easier

- **Example: Word Count**
  - hadoop streaming
    -input /user/zshao/articles
    -mapper ‘tr “ ” “\n”’
    -reducer ‘uniq -c‘
    -output /user/zshao/
    -numReduceTasks 32

Hadoop Usage

Other available tools

Hadoop Usage in e-INFRA CZ

MUNI ÚVT

Czech infrastructures (presented earlier) provide access to **prepared Hadoop cluster**:

- https://wiki.metacentrum.cz/wiki/Kategorie:Hadoop
- current resources:
  - 27 nodes × 16 cores
  - 27 nodes × 128 GB RAM
  - 1 PB HDFS storage
- documentation: https://wiki.metacentrum.cz/wiki/Hadoop