Data Integration and Large Scale Analysis

10 - Distributed Data-Parallel Computation

Dr. Lucas Iacono. 2025

Institute of
HUMAN-CENTRED COMPUTING

Agenda

- Announcements
- Data-Parallel Collection & Processing
- RDD, DataFrames, Datasets

Announcements

HCC
Institute of
HUMAN-CENTRED COMPUTING

Announcements

Course Evaluation and Exam

- Course evaluation: January 2026
- Exam date: Jan 30 (60 min written exam)
- Oral Exam for Erasmus Students
  - Schedule available in TeachCenter **(23/12/2024)**

Motivation and Terminology

Motivation and Terminology

Recap: Distributed Collections

Logical multi-set (bag) of key-value pairs (unsorted collection)

Different physical representations key-value pairs can be stored in various ways (e.g., database, across files, or in memory).

Easy Distribution via Horizontal Partitioning. Data divided into "chunks" (shards or partitions) based on the keys. Chunks stored on different machines (easier to handle large-scale data).

How collections are created: from single file with data or a folder of files (even if they're messy and unsorted).

Key	Value
13:00:01	12.1
14:00:05	16.0
13:00:03	12.5
13:00:05	13.0
14:00:04	15.7
14:00:06	16.3
13:00:00	12.1

Motivation and Terminology

Recap: Files and Objects

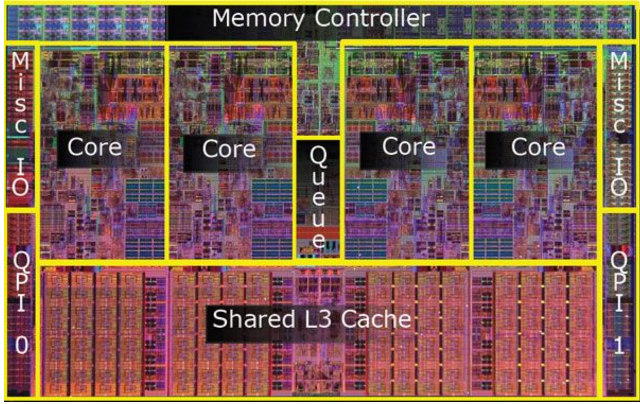
- **File:** large and continuous block of data saved in a specific format (CSV, Binary, etc.).
- **Object:** like a file, but binary and it comes with metadata (Images on S3)

Motivation and Terminology

Recap: Object Storage (e.g. AWS S3):

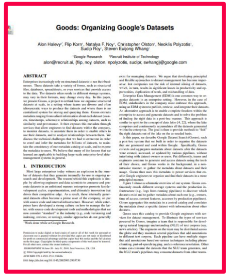
1. Data stored as objects (data, metadata, and UID).
2. Ideal for storing unstructured data like media files, backups, or large datasets.
3. Objects of a limited size (e.g., 5TB in AWS S3).

Motivation and Terminology



Nehalem Architecture

- **Integrated Memory Controller:** Integrated in chip, -- latency and ++ memory performance.
- **Support for DDR3 Memory:** Higher memory bandwidth (compared to DDR2).
- **Enhanced Hyper-Threading:** Each core supports two threads (+++ performance)
- **Multi-Core Scalability:** 2 to 8 cores per processor (2 threads / core)
- **Improved Cache Design:** Dedicated L1 and L2 cache p/core shared L3 cache



Goods: Organizing Google's Datasets

Alon Halevy[2], Flip Korn[1], Natalya F. Noy[1], Christopher Olston[1], Neoklis Polyzotis[1], Sudip Roy[1], Steven Euijong Whang[1]

[1]Google Research [2]Recruit Institute of Technology

alon@recruit.ai, {flip, noy, olston, npolyzotis, sudipr, swhang}@google.com

ABSTRACT

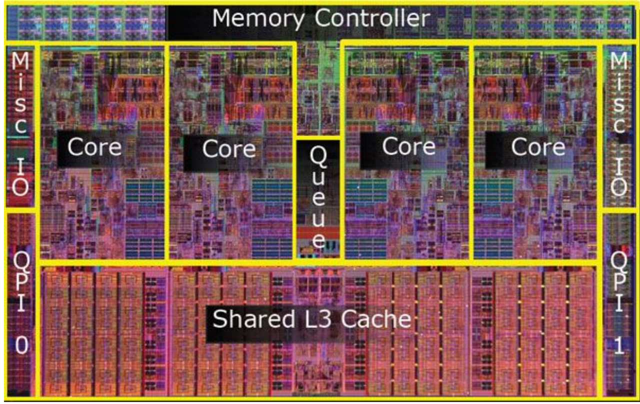
Michael E. Thomadakis: The Architecture of the Nehalem Processor and NehalemEP SMP Platforms, Report, 2010

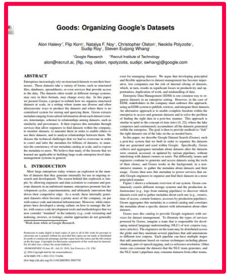
Institute of HUMAN-CENTRED COMPUTING

Motivation and Terminology

Nehalem Architecture

- **Energy Efficiency:** Turbo Boost for dynamic clock speed adjustments.
- **Advanced Manufacturing Process:** Higher transistor density and better efficiency.
- **Integrated Graphics (in later models):** Some models included integrated GPUs.
- **Foundation for Modern Architectures:** Established the groundwork for subsequent Intel architectures like **Sandy Bridge and Skylake.**
- **128-bits Floating-point multiplication**
- **128-bits floating-point addition**





Goods: Organizing Google's Datasets

Michael E. Thomadakis: The Architecture of the Nehalem Processor and NehalemEP SMP Platforms, Report, 2010

Motivation and Terminology

	Single Data	Multiple Data
Single Instruction	SISD	SIMD
Multiple Instruction	MISD	MIMD

Flynn's Classification

Computer architectures based on **how they handle instructions and data**.

- **SISD:**
  - One task at time - one data chunk (e.g. PC running a single program)
- **SIMD:**
  - One task at time - multiple data chunks (e.g. GPUs rendering)
- **MISD:**
  - Multiple tasks - one data chunk (e.g. fault-tolerant computers)
- **MIMD:**
  - Multiple tasks - multiple data chunks (multi-core CPUs 1 Core -> Program)

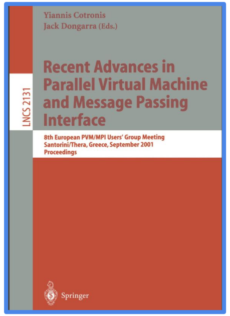
Michael J. Flynn, Kevin W. Rudd: Parallel Architectures.
ACM Comput. Surv. 28(1) 1996

Institute of
HUMAN-CENTRED COMPUTING

Motivation and Terminology

Distributed, Data-Parallel Computation

- Parallel computation of **function** foo() ➔ **single instruction (single function** applied to **all data items** in parallel).
- Collection X of data items (**key-value** pairs) ➔ **multiple data** (foo() operates on multiple **key-value** pairs).
- **Data parallelism** similar to **SIMD** but more coarse-grained notion of “instruction” and “data” ➔ **SPMD** (single program, multiple data)
- **Y** = **X.map**(**X** → **foo(x)**)
    - **X** = Data Items (key-value pairs)
    - **.map** (function/operation to each element in **X**)
    - **Y** = Output

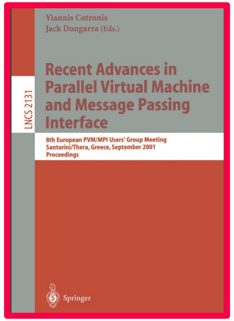


[Frederica Darema: The SPMD Model : Past, Present and Future. PVM/MPI 2001]

Motivation and Terminology

SPMD (single program, multiple data)

- **Dynamic Work Assignment.** Processes can **self-schedule**, ++ **flexibility & efficiency**.
- **More General than SIMD.** SPMD allows **different instruction streams for different data**. It can handle more complex tasks.
- **Efficient Control.** Performed at the **application level** rather than the OS level (less costly and more efficient than **Fork & Join**.
- **Applications:**
  - **MPI (Message Passing Interface)**
  - **Grid Computing**



[Frederica Darema: The SPMD Model : Past, Present and Future. PVM/MPI 2001]

Motivation and Terminology

Model	Key Features	Pros	Cons
BSP (Bulk Synchronous)	Global barriers enable synchronization after each phase	+++ Correctness and consistency; simple to implement	Overhead due to waiting at barriers Slow for stragglers
ASP (Asynchronous Parallel)	Processes run independently	Faster execution (no waiting)	Accuracy issues from outdated data
SSP (Stale-Synchronous Parallel)	Controlled staleness allows fastest processes to proceed within a limit	Balances efficiency and consistency; reduces waiting time compared to BSP	Small inaccuracies

Data-Parallel Collection & Processing

HCC
Institute of
HUMAN-CENTRED COMPUTING

Hadoop

Brief Hadoop History

- Google’s GFS + MapReduce [ODSI’04] **->** Apache Hadoop (2006).
- Apache Hive (SQL), Pig (ETL), Mahout (ML), Giraph (Graph)

Hadoop Architecture / Ecosystem

- Management (Ambari)
- Coordination / workflows (Zookeeper, Oozie)
- Storage (HDFS)
- Resources (YARN)

Hadoop

Hadoop Ecosytem: Apache Hive

- **What it is:**
  - Data warehouse **(OLAP)** infrastructure built on top of Hadoop.
  - Query and analyze large datasets stored in Hadoop using a SQL-like language called HiveQL.
- **Main Purpose:**
  - Querying and analysis of big data using SQL.
  - Suitable for batch processing and data analysis.

Hadoop

Hadoop Ecosytem: Apache Pig (ETL)

- **What it is:**
    - **High-level** platform for creating **data processing programs** in Hadoop.
- **Main Purpose:**
    - **ETL** operations. Cleaning, transforming, and preparing large datasets for analysis.
- **Use Case:**
    - Processing raw web logs into structured formats for further analysis.

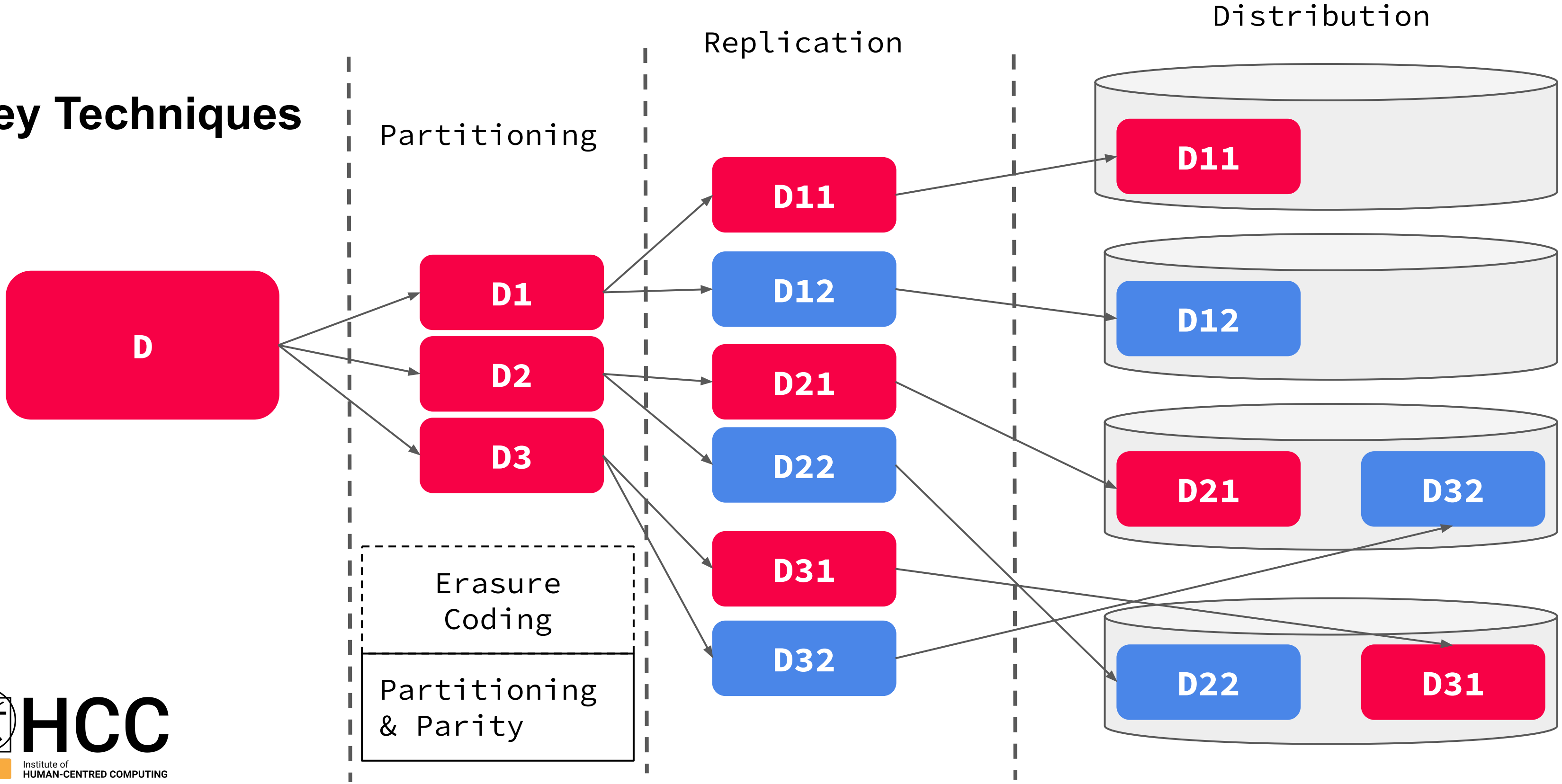
Hadoop

Hadoop Ecosytem: Apache Mahout (ML)

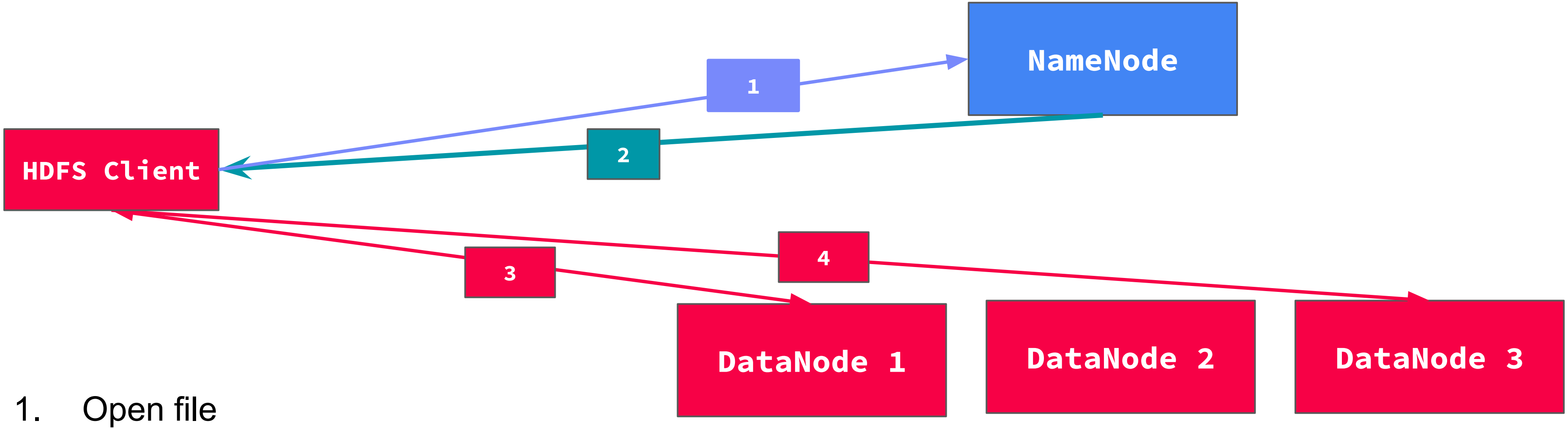
- **What it is:**
  - A library for building scalable **machine learning** algorithms on top of Hadoop.
  - Focused on distributed or scalable implementations of common ML algorithms.
- **Main Purpose:**
  - Implementing machine learning algorithms like clustering, classification, and recommendation systems on large datasets.

Recap: HDFS

Key Techniques



Recap: HDFS Read



1. Open file
2. Metadata (Block Locations, Replicas)
3. Read Block 1 (DataNode 1)
4. Read Block 2 (Data Node 3)

HCC
Institute of
HUMAN-CENTRED COMPUTING

MapReduce – Programming Model

Overview

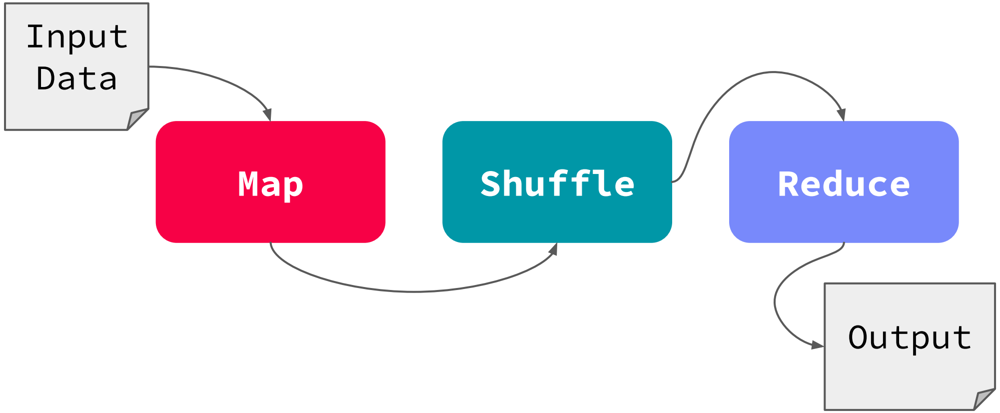
- MapReduce is a programming model for processing large datasets in parallel, distributed across multiple nodes.
- Developed by Google; popularized by Apache Hadoop.

Map

Processes and transforms input data into intermediate **key-value** pairs.

Reduce

Aggregates intermediate data to produce the final result.



MapReduce I

Why MapReduce?

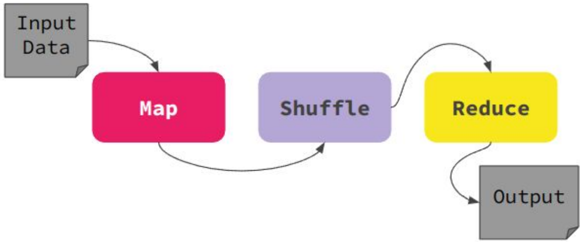
- Handles large-scale data processing efficiently.
- Works on commodity hardware.
- Built-in fault tolerance.
- Suitable for structured, semi-structured, and unstructured data.

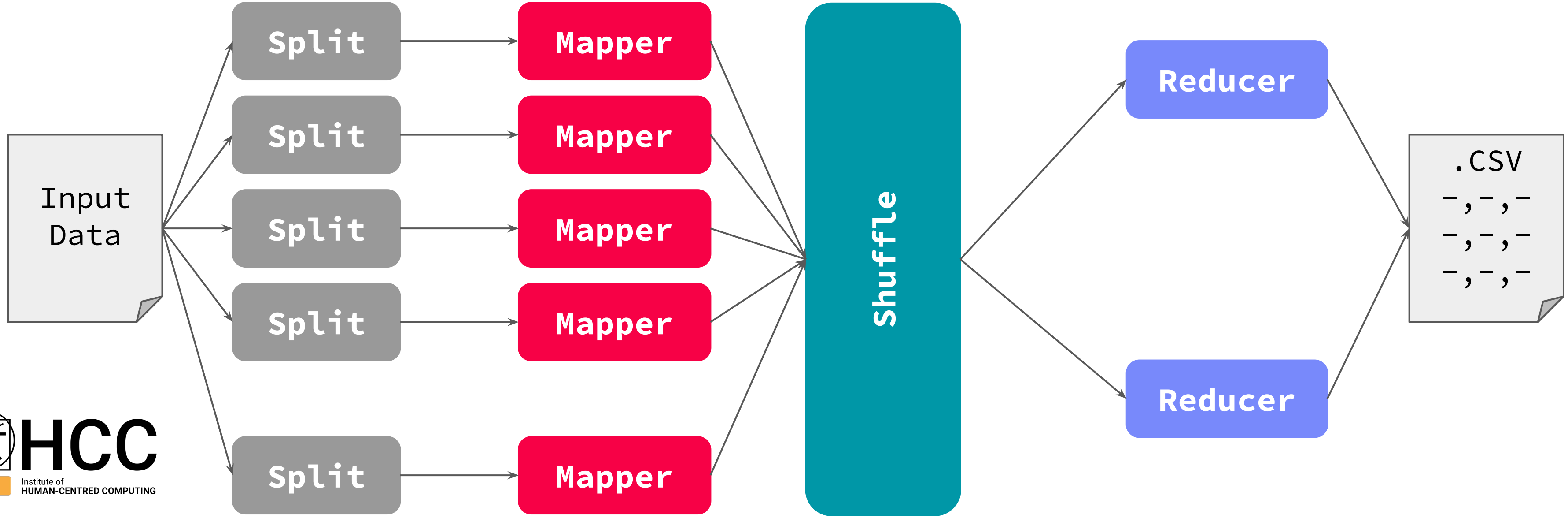
MapReduce II

Key Concepts

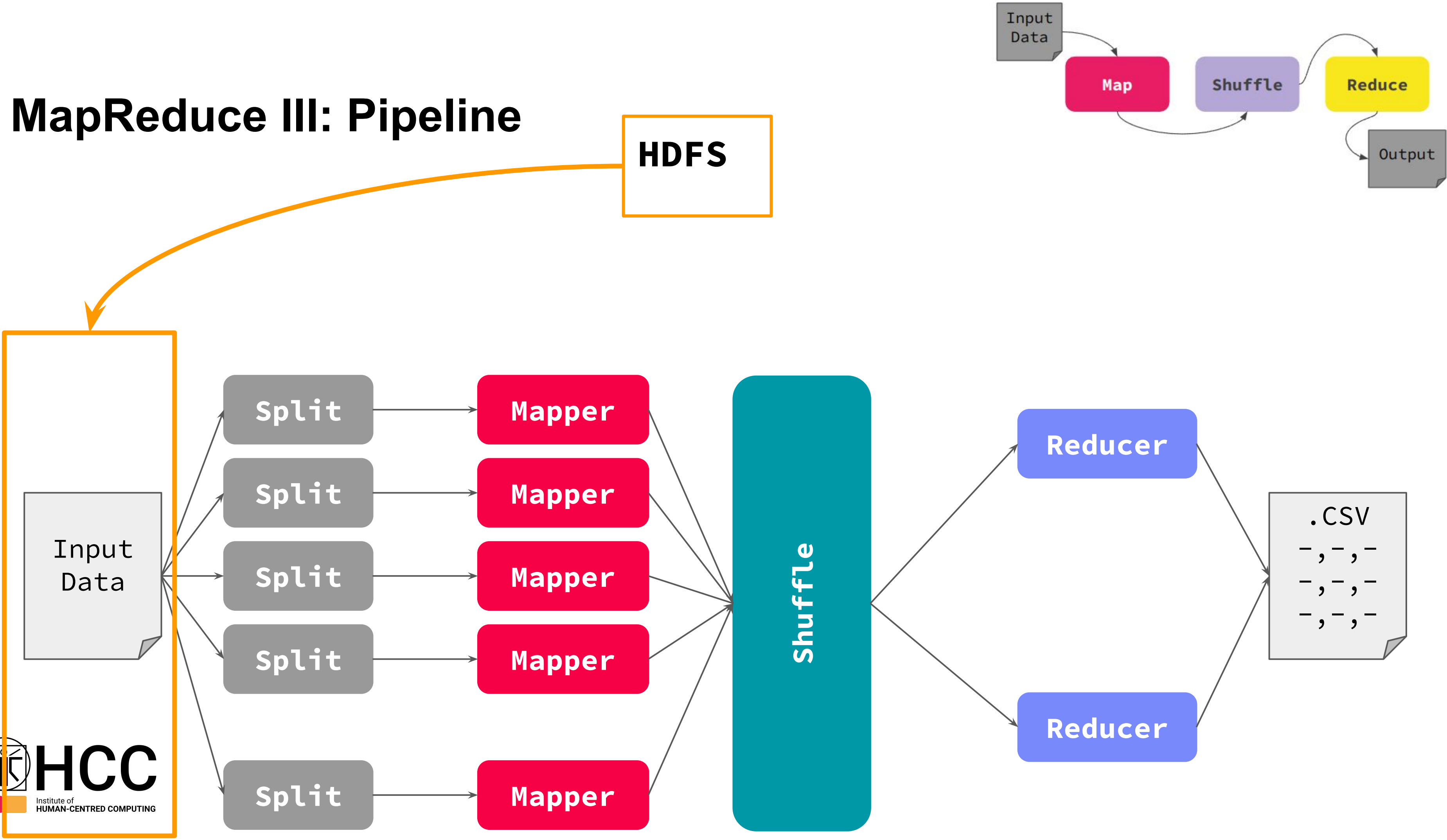
- **Distributed Processing:** Data is split across multiple nodes for parallel execution.
- **Key-Value Pairs:** Core data structure in MapReduce.

MapReduce III: Pipeline



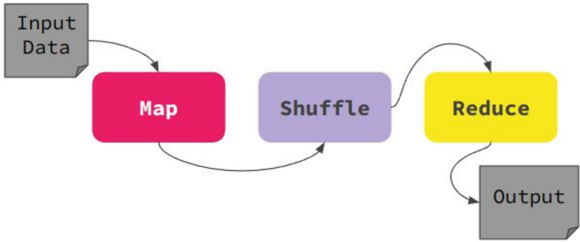


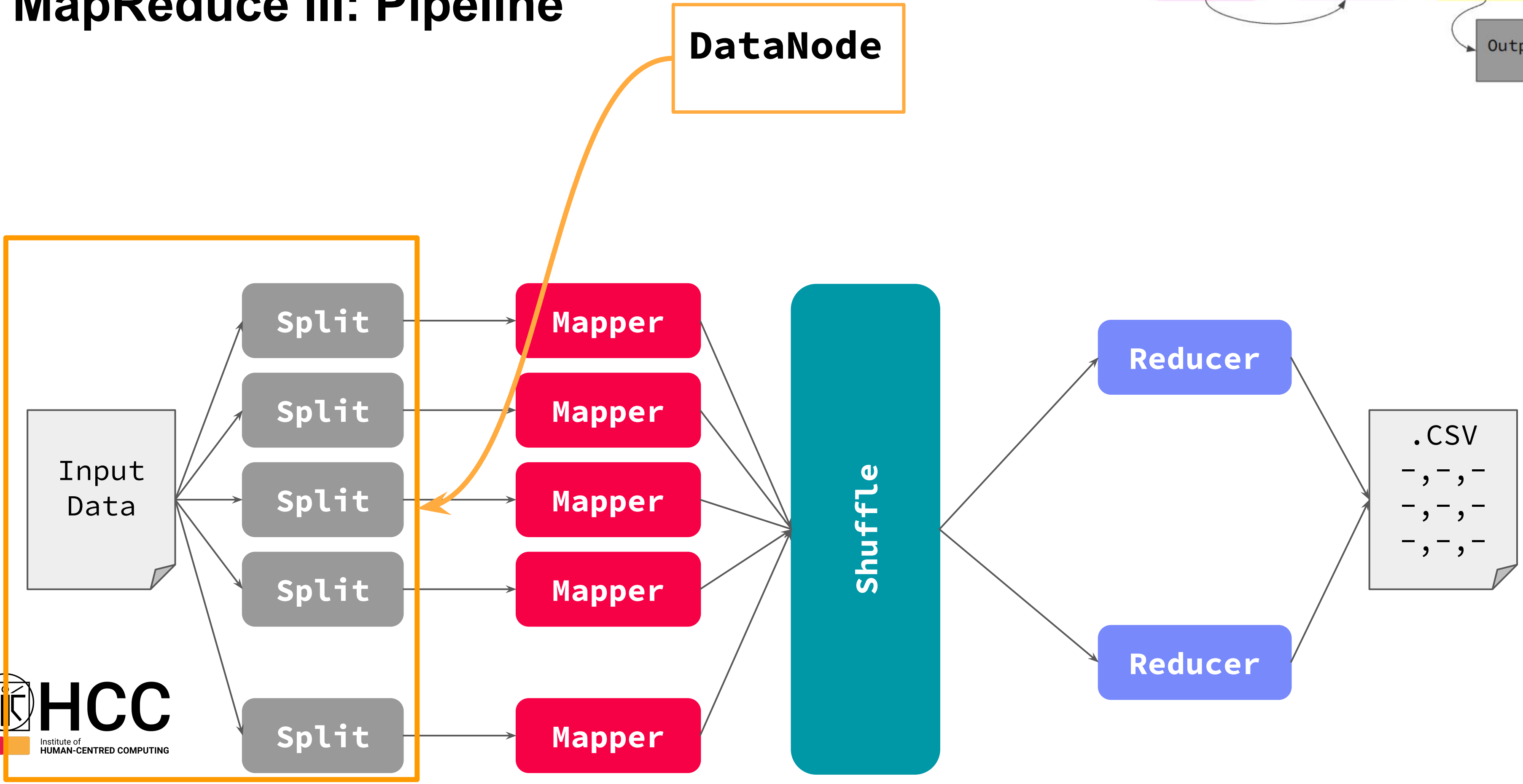
MapReduce III: Pipeline



HCC
Institute of HUMAN-CENTRED COMPUTING

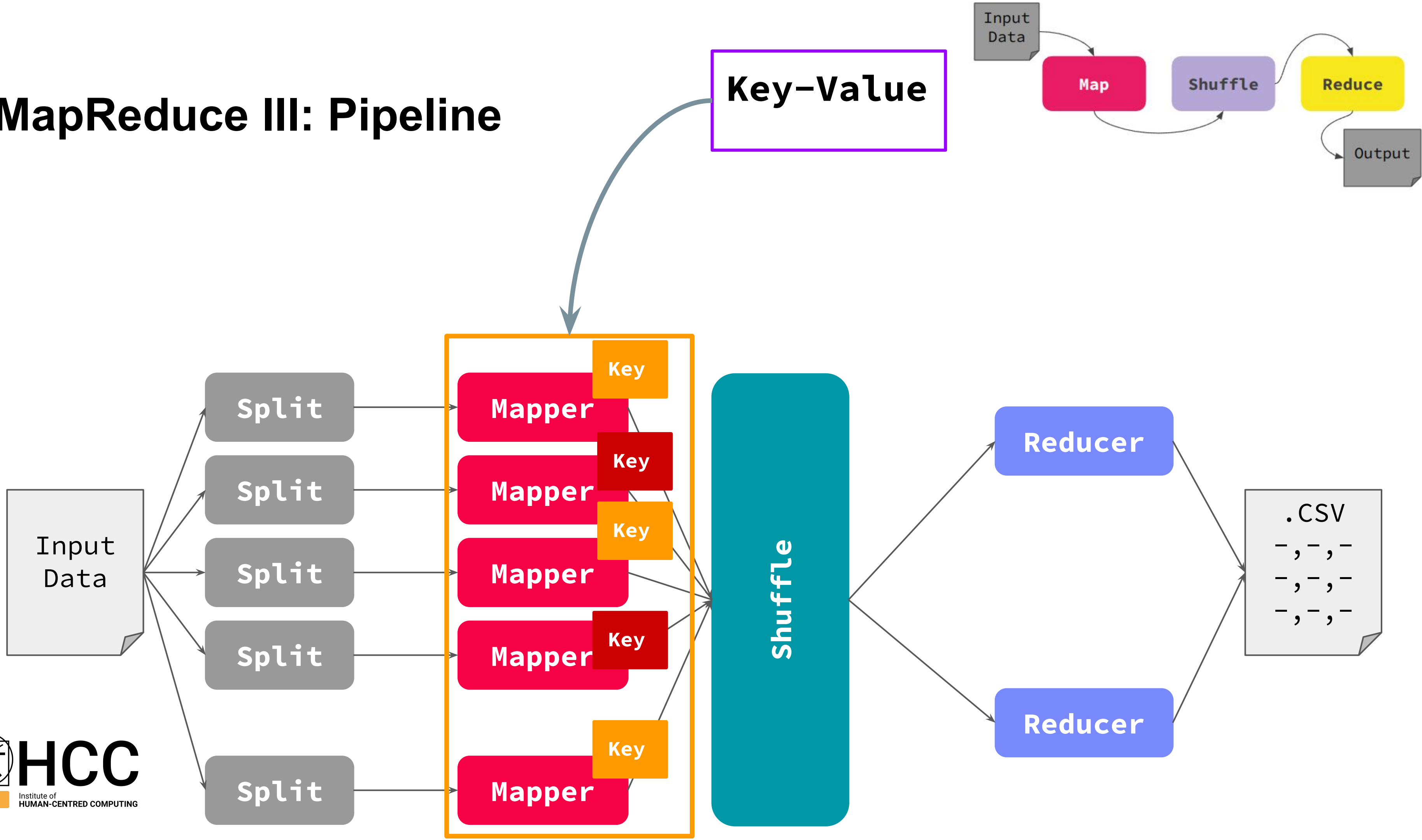
MapReduce III: Pipeline



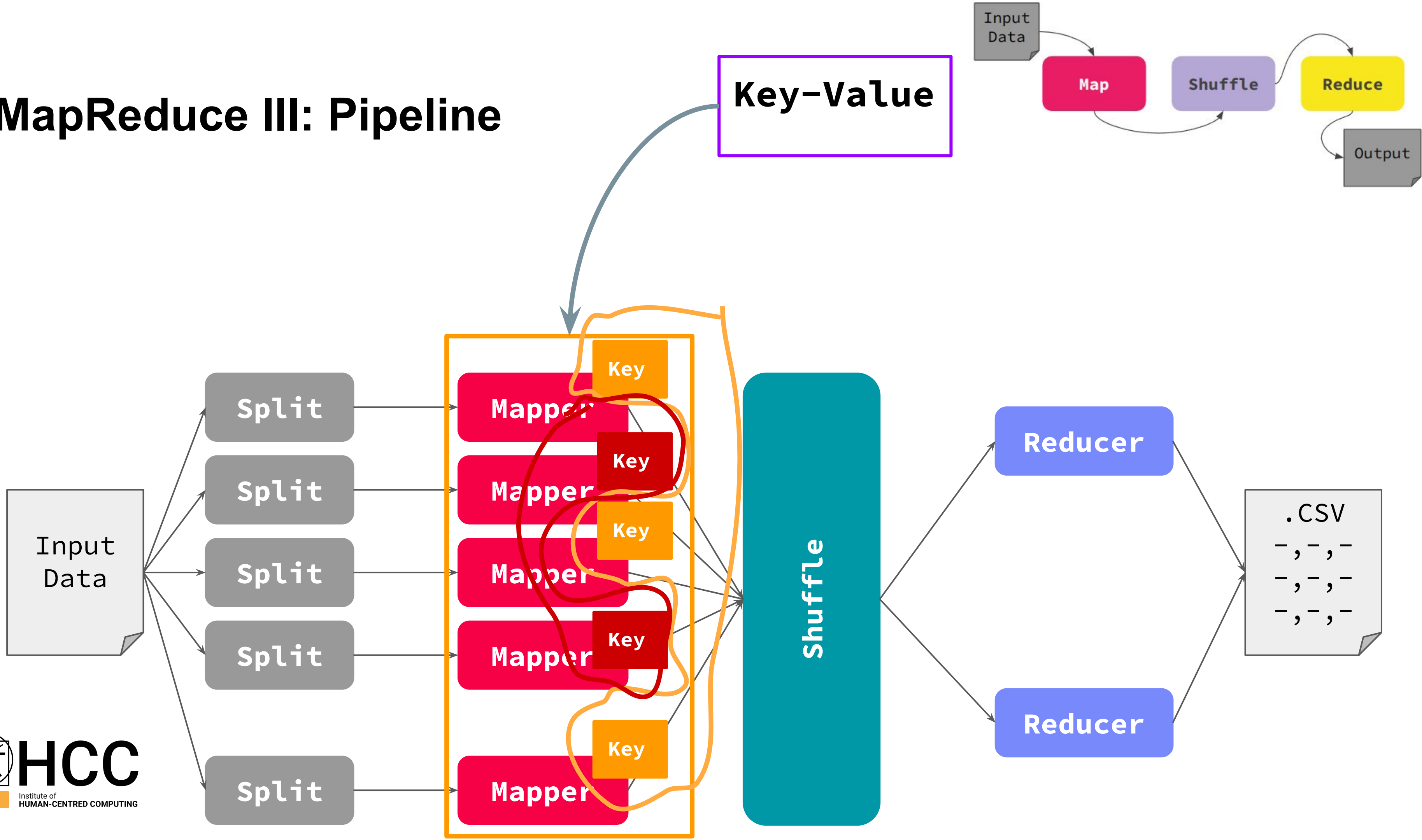


HCC
Institute of
HUMAN-CENTRED COMPUTING

MapReduce III: Pipeline

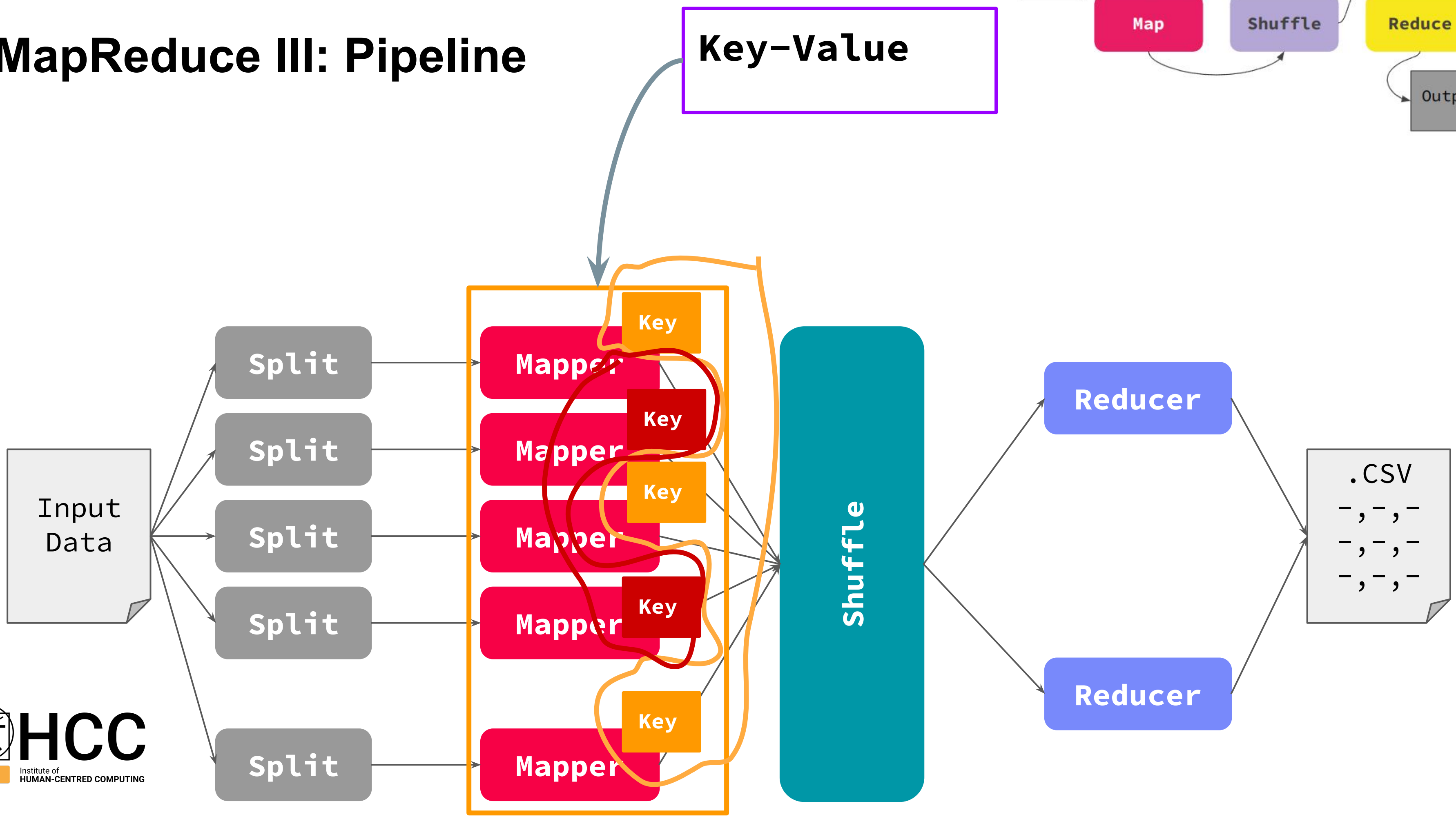


MapReduce III: Pipeline



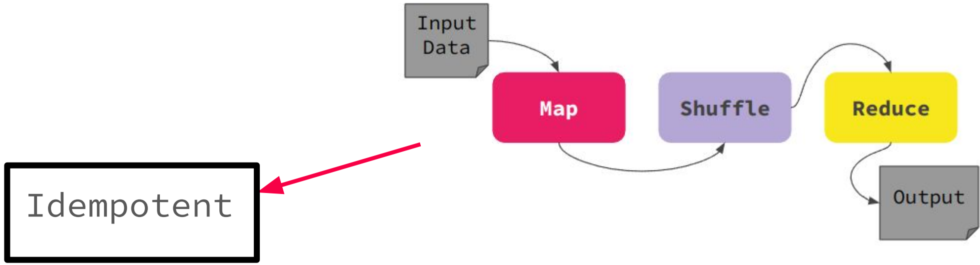
HCC
Institute of
HUMAN-CENTRED COMPUTING

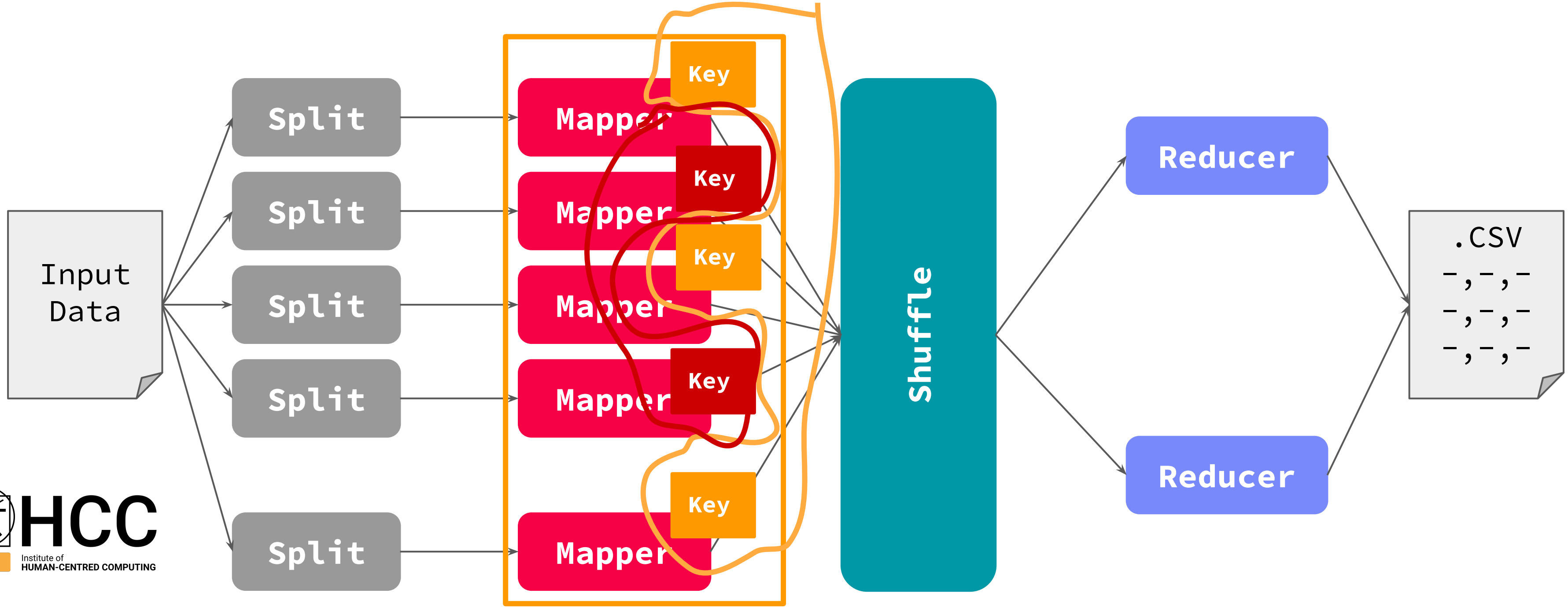
MapReduce III: Pipeline



HCC
Institute of HUMAN-CENTRED COMPUTING

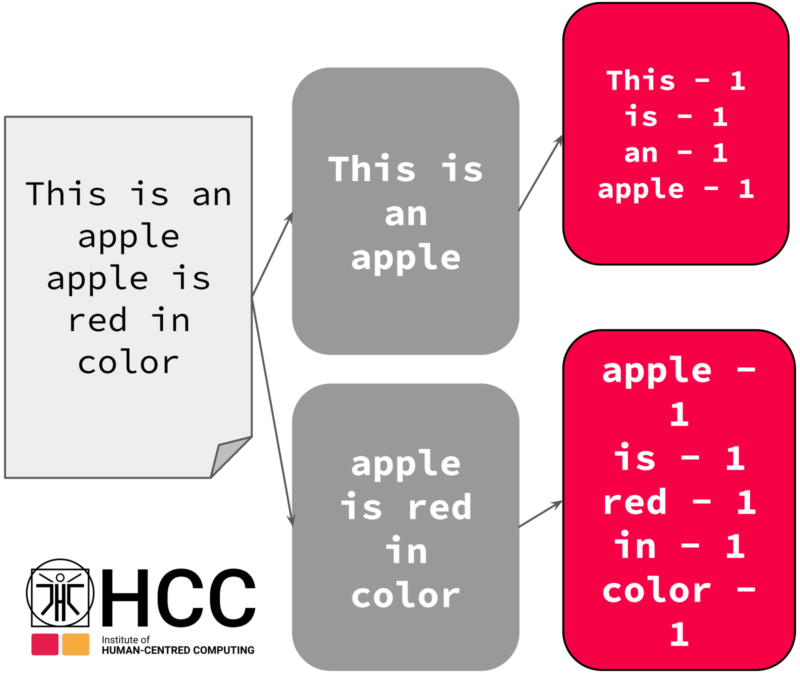
MapReduce III: Pipeline





HCC
Institute of
HUMAN-CENTRED COMPUTING

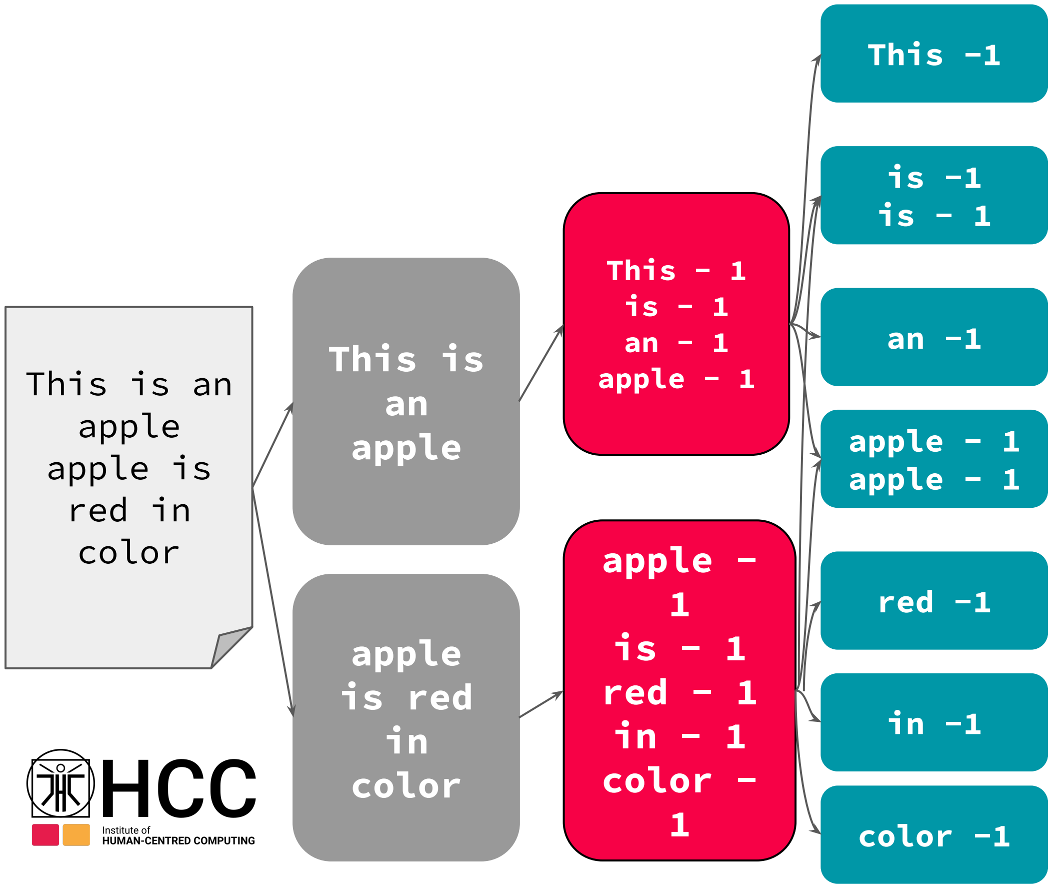
MapReduce IV: Example from Google Paper



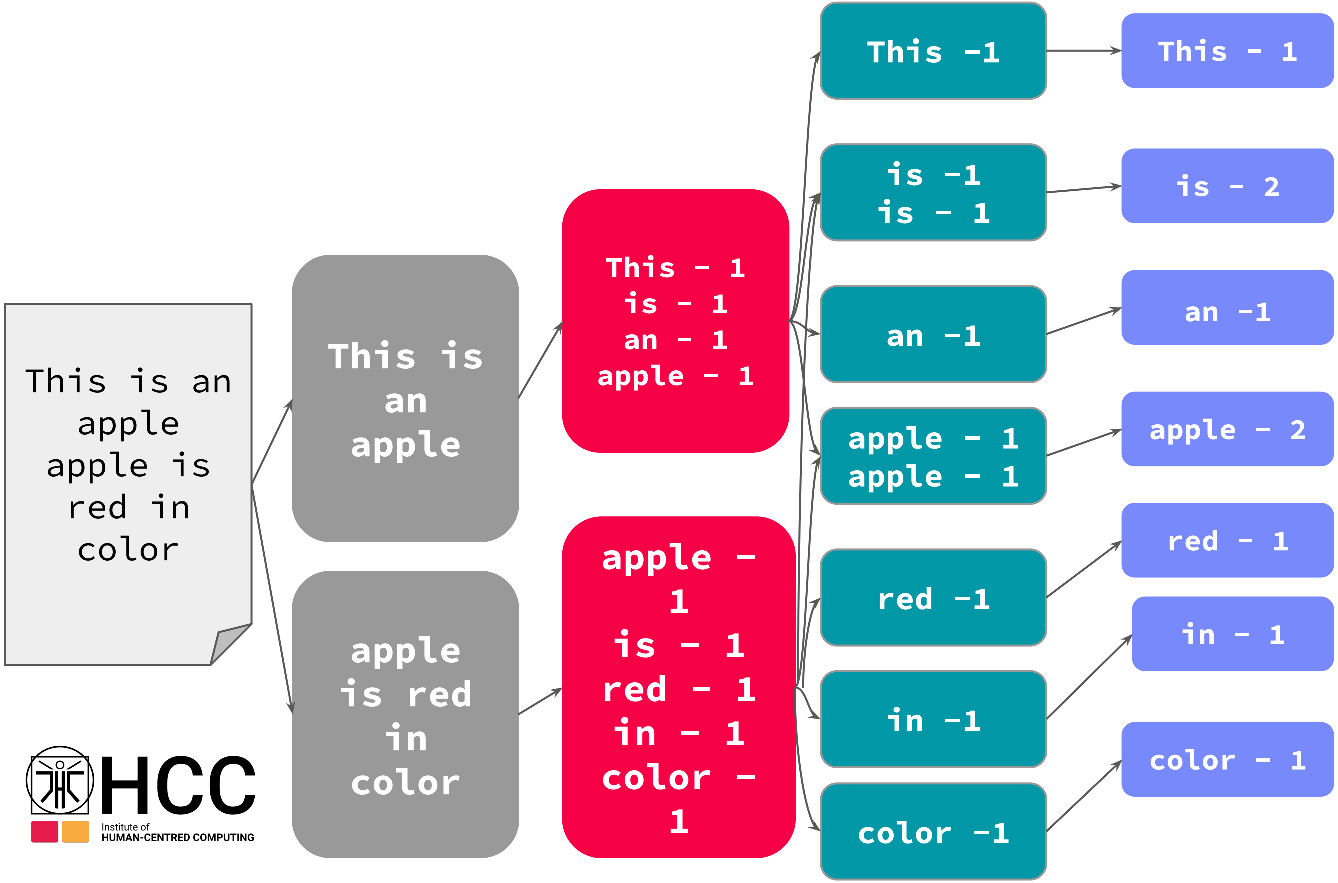

MapReduce: Simplified Data Processing on Large Clusters
by Jeffrey Dean and Sanjay Ghemawat

[Jeffrey Dean, Sanjay Ghemawat: MapReduce: Simplified Data Processing on Large Clusters. OSDI 2004]
]

MapReduce IV: Example from Google Paper

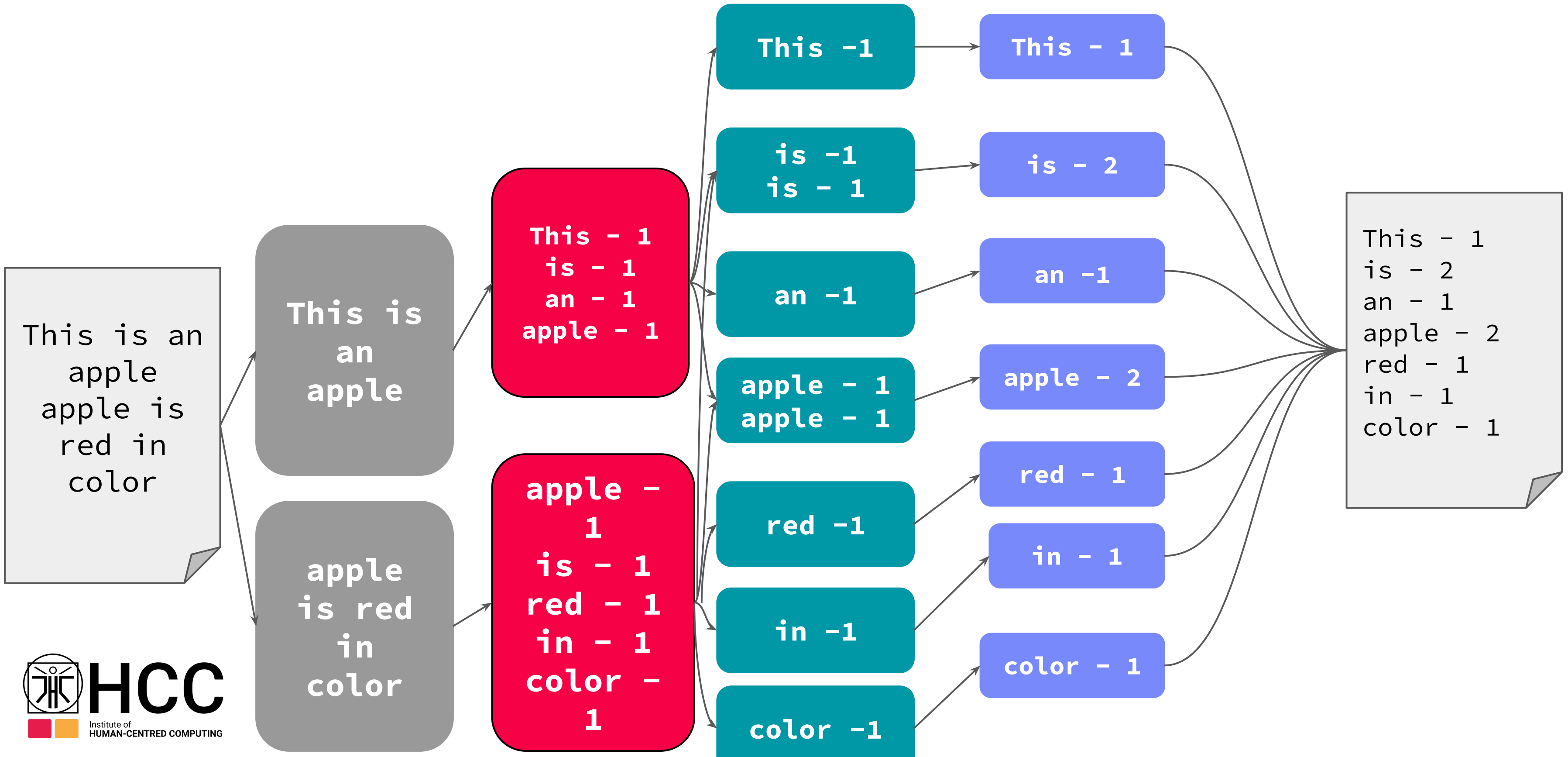


MapReduce IV: Example from Google Paper

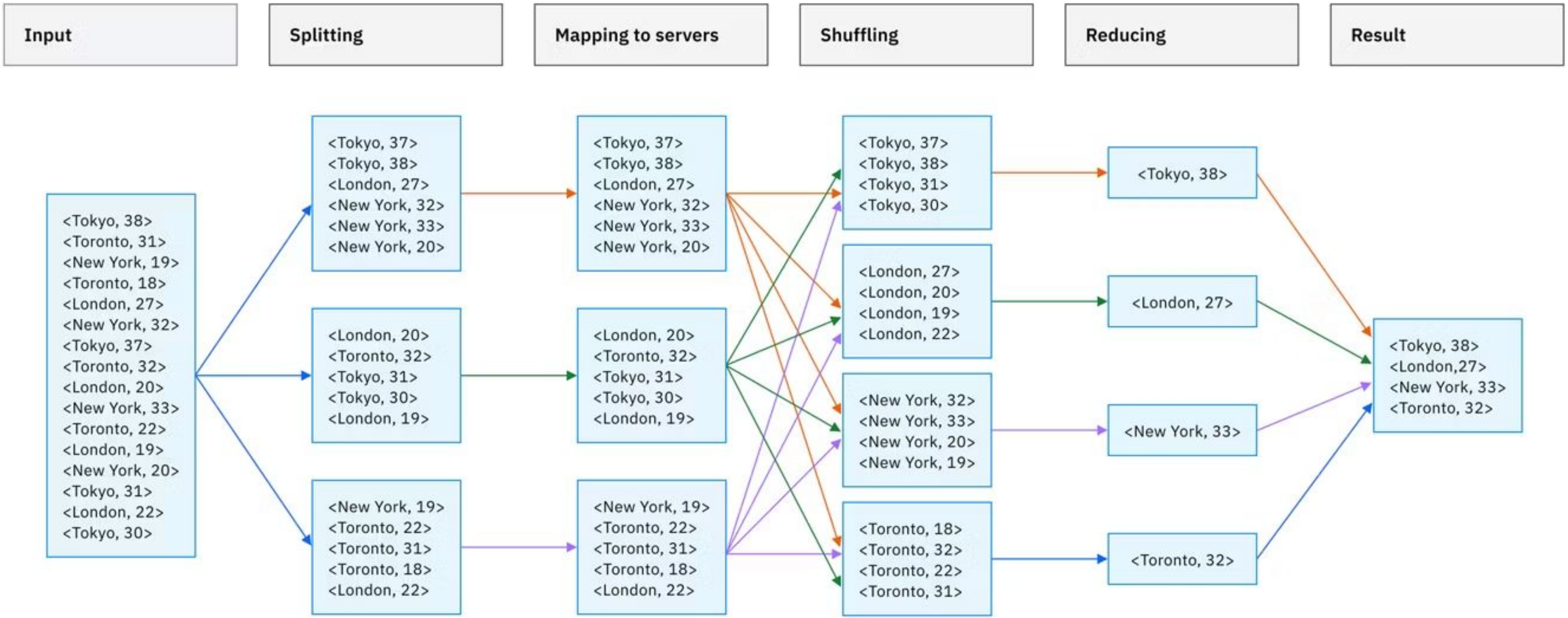


Institute of HUMAN-CENTRED COMPUTING

MapReduce IV: Example from Google Paper



MapReduce VI: City - Temperature Example (Source: IBM)



MapReduce: Summary (Pros)

- **Large-scale processing.** Large amounts of data distributed across multiple nodes in a cluster.
- **Fault-tolerant.** If a node fails, the system can recover and reassign tasks to other nodes.
- **User Defined Functions and files.** Developers can define their own **custom processing logic through UDFs**, and the **model relies on files** to store intermediate and final results.
- **Flexibility.** Developers can **customize processing logic** while the system manages **distribution** and **fault recovery automatically**.
- **Restricted functional APIs.** MapReduce relies on a limited set of functional primitives:
  - **Map**: Transforms input data into key-value pairs.
  - **Reduce**: Aggregates values associated with the same keys to produce results.
- **Implicit parallelism**. Developers only need to implement the Map and Reduce functions; the distribution of workload across nodes relies on the system.

MapReduce: Summary (Cons)

- **Performance:** its performance can suffer in complex workloads due to heavy reliance on I/O (writing and reading intermediate data to/from disk).
- **Low-level APIs:** The API is relatively basic, requiring a lot of manual effort to implement more sophisticated workflows.
- **Many different systems:** Specialized systems (**e.g., Apache Spark**, Apache Flink, or distributed database systems) have emerged as alternatives, often being more efficient and user-friendly.

Spark History and Architecture

Evolution to Spark (and Flink)

- Spark [HotCloud'10] + Resilient Distributed Datasets (RDDs) [NSDI'12] → **Apache Spark (2014)**
- **Design 1. Standing executors with in-memory storage:**
  - Spark keeps **long-running worker processes** (executors) active, enabling tasks to run faster by avoiding repeated setup costs.
  - **Data is stored in memory** whenever possible, **minimizing disk I/O** for iterative and interactive jobs.
- **Design 2. Lazy evaluation:**
  - **Directed Acyclic Graph of transformations** rather than executing them immediately.
  - Actions (e.g., collect, save, count) trigger DAG's execution, allowing workflow optimization by reordering and combining operations.

Spark History and Architecture

- **Design 3:** Fault tolerance via RDD **lineage**
  - Data partition lost → Spark can recompute using **lineage** graph of transformations applied to the data (reliability without heavy replication).
- **Performance:**
  - **In-memory storage.** Spark significantly reduces disk I/O, and it is faster for iterative tasks (e.g. machine learning) than Hadoop.
  - **Fast job scheduling.** Spark's scheduler operates with **low overhead**, enabling tasks to be scheduled in milliseconds (~100ms), compared to Hadoop's ~10 seconds per job.

Spark History and Architecture

- **APIs:**
  - **Richer functional.** Wide range of functional operators (e.g., map, reduce, filter, groupByKey, flatMap) compared to Hadoop -> **easier to write complex workflows.**
  - **General computation DAGs.** Unlike MapReduce, which forces jobs into two rigid phases (map and reduce), Spark supports general **DAGs for more flexible computation flows.**
  - **High-level DataFrame/Dataset.** data abstractions that simplify working with structured data and enable **query optimization.**

Spark History and Architecture

- **Unified Platform.** Multiple workloads into a single platform:
  - Batch processing (similar to MapReduce)
  - Streaming (real-time data)
  - Machine learning (MLlib)
  - Graph processing (GraphX)
  - SQL queries (Spark SQL)

Spark Functionality: Core components

Resilient Distributed Datasets (RDDs):

- Distributed collections **[see Motivation and Terminology]** (foundation for fault tolerance and parallelism.). See IBM definition

DataFrames and Datasets:

- Higher-level abstractions for structured and semi-structured data (Optimized via Spark's Catalyst engine).

Spark SQL:

- Query structured data using SQL.

MLlib:

- Machine learning library for scalable algorithms.

GraphX:

- Graph processing library.

Spark Functionality: Architecture

Driver Program:

- Defines the **application** and **coordinates tasks**.

Cluster Manager:

- Allocates resources (YARN, Kubernetes).

Executors:

- Workers that **execute tasks** and store data partitions.

DAGs:

- Spark builds a **logical execution plan** before running tasks.

Spark Functionality: Workflow

- **Create RDD/DataFrame:** Load data into Spark from HDFS, S3, or other sources.
- **Transformations:** Apply operations (e.g., map, filter, groupBy).
- **Actions:** Trigger execution (e.g., collect, save).
- **Execution:**
    - Splits tasks across nodes
    - Uses DAG to optimize execution.

RDD, DataFrames, Datasets

Origins of DataFrames

Recap: Data Preparation Problem

- **80% Argument:** 80-90% time for finding, integrating, cleaning data

- Data scientists prefer scripting languages and in-memory libraries

Python DataFrames:

- Python pandas DataFrame for seamless data manipulations (most popular packages/features)
- DataFrame: table with a schema
- Descriptive stats and basic math, reorganization, joins, grouping, windowing
- Limitation: Only in-memory, single-node operations

```
import pandas as pd

df = pd.read_csv(‘data/tmp1.csv’,
index_col=2)

df.head()

df = pd.concat(df, df[[‘A’,’C’]],
axis=0)
```

Spark RDD, DataFrames, and Datasets

Overview Spark DataFrame

- DataFrame is distributed collection of rows with named/typed columns
- Relational operations (e.g. select, joins, group, aggregations)
- DataSources (e.g., json, jdbc, parquet, hdfs, s3, csv)

DataFrame and Dataset APIs

- DataFrame was introduced as basis for Spark SQL
- Datasets allow more customization and compile-time analysis errors (Spark 2)

Spark RDD, DataFrames, and Datasets

Resilient Distributed Dataset (RDD):

Immutable distributed collection of data, distributed across cluster nodes

Use cases:

- Low-level transformation, actions, and control on dataset;
- Data is unstructured (media streams or text)
- Data manipulation with functional programming
- No need for a schema

Spark RDD, DataFrames, and DataSets

Dataframes:

Immutable distributed collection of **structured** data, distributed across cluster nodes

Use cases:

- Structured data
- Non-expert in spark technologies

Spark RDD, DataFrames, and Datasets

Datasets:

Strongly typed collection of domain-specific objects that can be transformed in parallel using functional or relational operations. Each Dataset also has an untyped view (DataFrame).

Use cases:

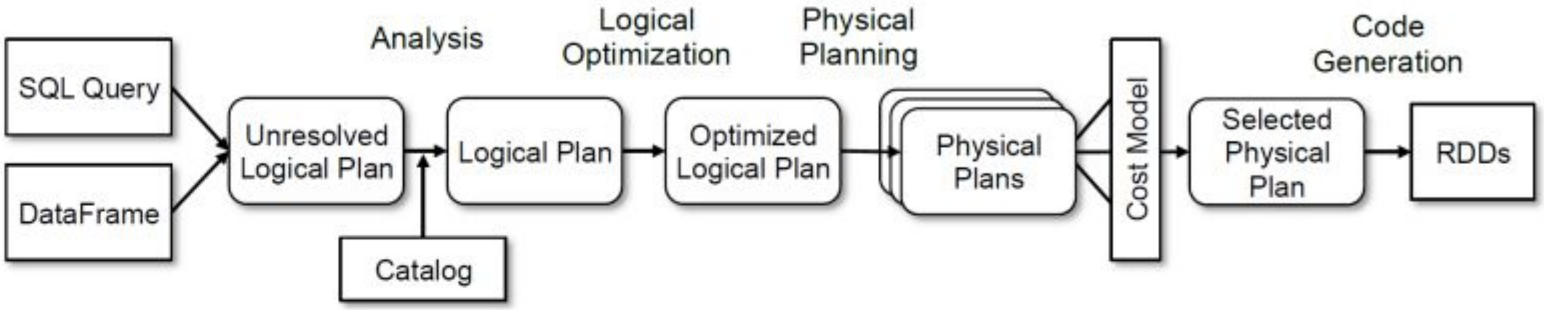
- Performance-critical pipelines (syntax and analysis) error detection at compile time)
- Rich semantics, high-level code expressions, filters, maps, aggregations, semi-structured data

SparkSQL and DataFrame/Dataset

Overview SparkSQL

- Shark (~2013): academic prototype for SQL on Spark
- SparkSQL (~2015): reimplementation from scratch
- Common IR and compilation of SQL and DataFrame operations

Catalyst: Query Planning



Summary and Q&A

Summary and Q&A

- **Summary and Q&A**
  - Motivation and Terminology
  - Data-Parallel Collection Processing
  - RDD, DataFrames, Datasets
- **Next Lectures**
  - Distributed Stream Processing **[Jan 09]**

Vielen Dank!

MapReduce VI: Hands on Lab

Servers Log

- Use the **MapReduce** programming model to:
  - Count how many times each page was accessed.
  - Identify the most popular page.
- Calculate the Average Using **MapReduce**
  - Given a list [4, 8, 15, 16, 23, 42], compute the average using MapReduce.