Amazon Elastic MapReduce

Architectures for Big Data Science, Spring 2016

What is MapReduce?

- Distributed programming model used to process and generate large amounts of data on a cluster, invented by Jeffrey Dean and Sanjay Ghemawat at Google in 2004.
- Motivation: Not everyone has a supercomputer to process Big Data quickly
- Allows fast processing of Big Data on a network of inexpensive machines
- Read the MapReduce Paper: http://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf

Big Picture of MapReduce

- Key Idea: “Divide and conquer by **distributing** across multiple machines”
- MapReduce describes a distributed computing **framework** consisting of:
  - Network of machines
  - Distributed file-system
  - Resource manager
- User specifies a Map() and a Reduce() function
  - The framework takes care of everything else:
    - fault-tolerance
    - data distribution
    - load-balancing

Big Picture of MapReduce

- Distributed system has two types of machines:
  - Mapper: executes the Map function on a data split
  - Reducer: executes the Reduce function on output from a Mapper
- Map function model
  - Convert an input key-value pair into a list of intermediate key-value pairs
  - Map(key, value) → List(key, value)
- Reduce function model
  - Merge all intermediate values associated with the same key
  - Reduce(key, List(value)) → List(key, value)

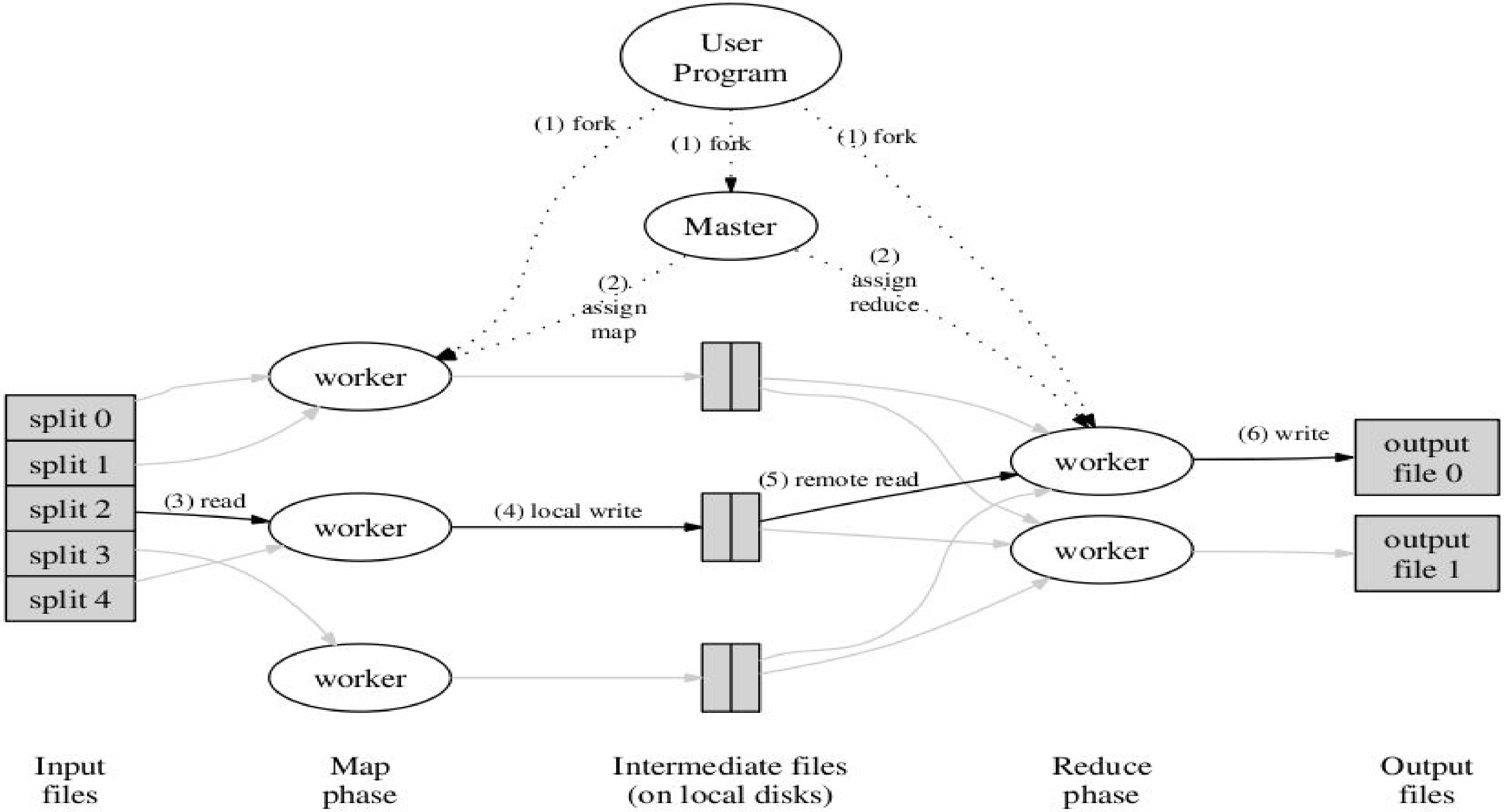
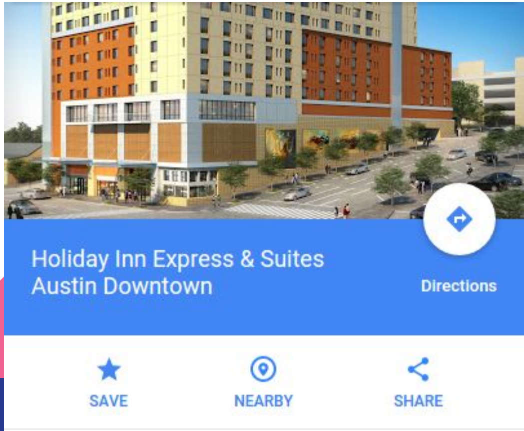


Figure 1: Execution overview

Taken from: Dr. David Franke's Lectures from CS378: Big Data Programming

Example: Google Geographical Data

- Objective: Given GPS coordinates of several locations, find the nearest Taco shop to each location.
- May seem trivial for a small number of data points, but for Google millions of people are requesting nearby locations every second on their Google Maps app
- Servicing these requests one by one would be inefficient
- Map function? Reduce function?



Map Function

- Input key-value pair is <Name, GPS coordinates>
- Map should use the GPS coordinates to compute the nearest Taco Shop
    - Perform a search in the subset of locations within a 5 mile radius of the given GPS coordinates and computes the distance to that Taco Shop
- Output is a list of <Name, (Taco Shop, Distance)>

Map Function

- Example: Mapper receives the key-value pair <Holiday Inn Express, (101.3, 58.5)>
- Map function finds all Taco Shops within 5 miles of (101.3, 58.5)
- Output is a list of <key, value> pairs:
  - <Holiday Inn Express, (Taco Joint, 1.2 miles)>
  - <Holiday Inn Express, (Torchy’s Tacos, 2.5 miles)>
  - <Holiday Inn Express, (Tamale House, 3.2 miles)>

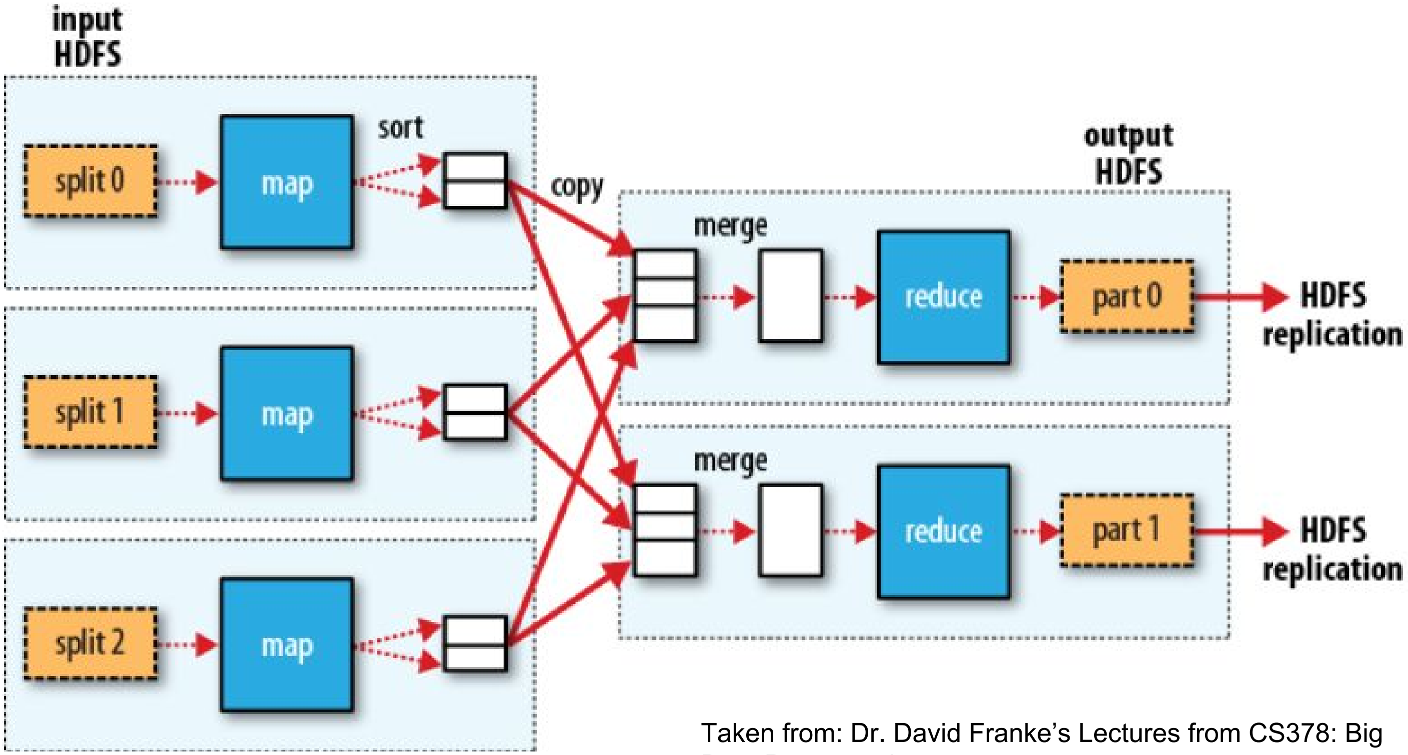
Reduce Function

- Input is list of <Name, (Taco Shop, Distance)>
- Reduce should find the minimum distance (value) of all Taco Shops nearby the same location (key)
    - Note: The reduce function here is trivial
- Output is <Name, Taco Shop>

Reduce Function

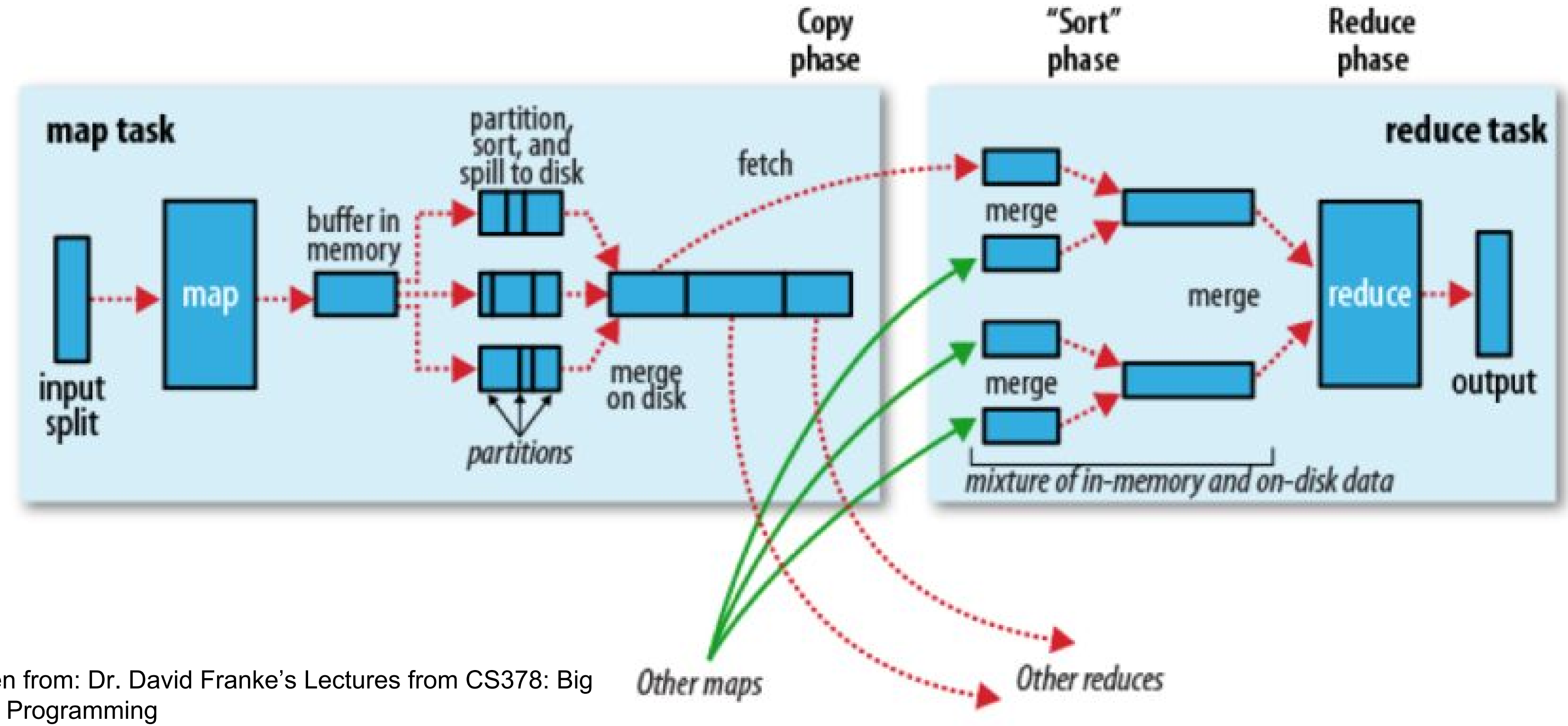
- Example: Output from Mapper is list of <key, value> pairs:
  - <Holiday Inn Express, (Taco Joint, 1.2 miles)>
  - <Holiday Inn Express, (Torchy's Tacos, 2.5 miles)>
  - <Holiday Inn Express, (Tamale House, 3.2 miles)>
- Minimum = Taco Joint
- Reducer Emits as output <Holiday Inn Express, Taco Joint>
- Done!

Architectural Overview of Hadoop MapReduce



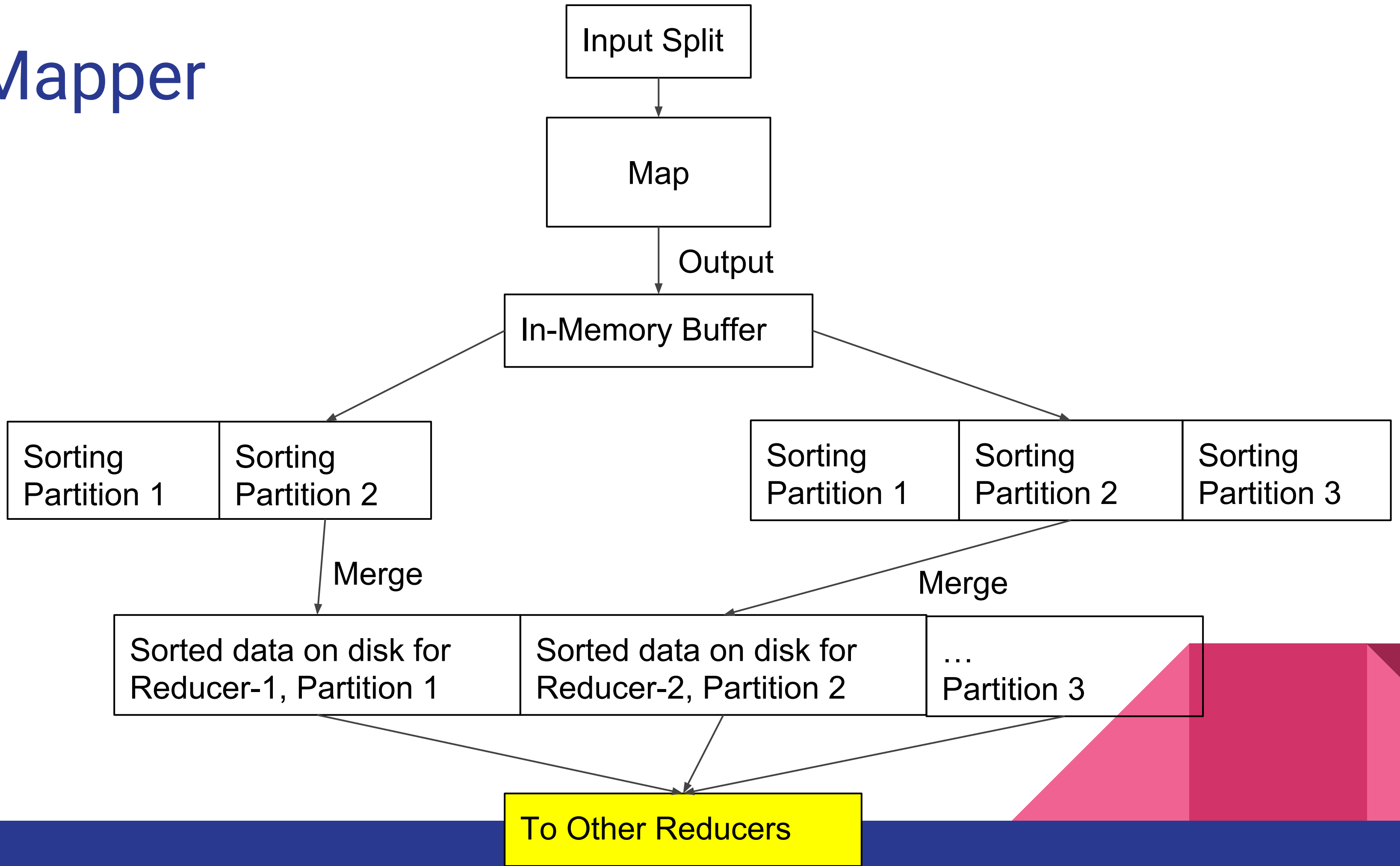
Taken from: Dr. David Franke's Lectures from CS378: Big Data Programming

The 6 Steps of MapReduce



Taken from: Dr. David Franke's Lectures from CS378: Big Data Programming

Mapper


Input Split
Map
Output
In-Memory Buffer
Sorting Partition 1
Sorting Partition 2
Sorting Partition 1
Sorting Partition 2
Sorting Partition 3
Merge
Merge
Sorted data on disk for Reducer-1, Partition 1
Sorted data on disk for Reducer-2, Partition 2
… Partition 3
To Other Reducers

Step 1: Split the Data

- Big data must be split so that multiple mappers can be executed in parallel over smaller subsets.
- Each split is processed by one mapper
- Each split is further divided into <key, value> pairs
- Map function is invoked on each <key, value> pair

Step 2: Combiner

- “mini-reducers”
- optional and implemented like a reduce function
- decreases the effort that would go into the shuffle and sort phases later
- a reduce function that is invoked on the same machine that performed the map operation

Step 3: Partitioner

- Built-in but can be customized
- Map tasks on different machines may produce output with the same keys that should be processed by a single reducer.
- Must have a way to decide upon map outputs with which key should be forwarded against which reducer for reduction
- Default Implementation:
    - N = total number of reducers
    - Reducers are given IDs numbered from 0 to N-1
    - Reducer ID to assign map output with key value = K is:
        - hash(K) % N

Step 4: Shuffle and Sort

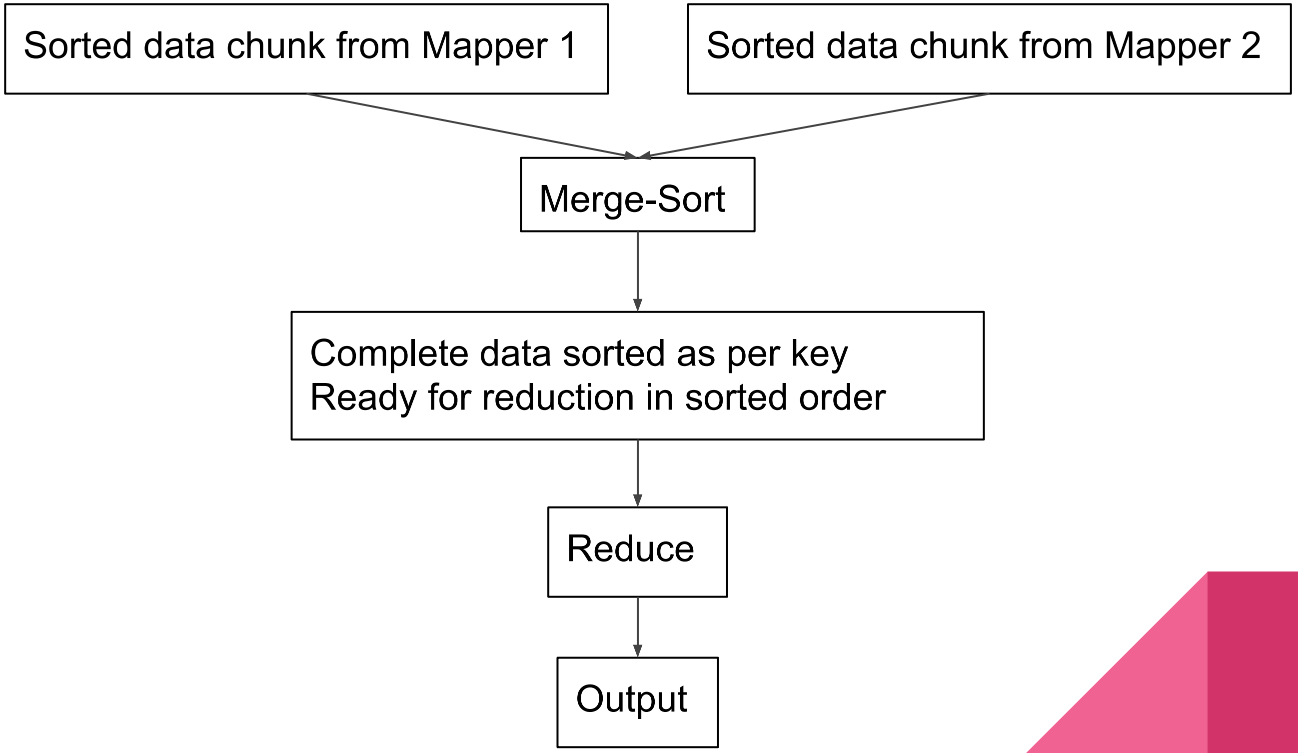
1. Map outputs with the same keys can be emitted by various map tasks running on separate machines.
    a. Partitioner decides which reducer an output with a specific map output record should be sent
2. Shuffling - process of moving around the map output records between different machines
3. Each map task writes output to a memory buffer and spills the overflow data to a local disk in round-robin fashion

Step 5: Reducer

Must perform following operations:

1. Fetch the sorted map data from various machines having data for this reducer.
2. Perform a merge-sort on the data
3. Execute reduce function over sorted data in order

Reducer


Sorted data chunk from Mapper 1
Sorted data chunk from Mapper 2
Merge-Sort
Complete data sorted as per key
Ready for reduction in sorted order
Reduce
Output

Apache Hadoop

- Open-source version of Google’s MapReduce framework
- Inspired by two papers, MapReduce and Google File System
- 4 components:
    - Hadoop Common
    - Hadoop Distributed File System (HDFS)
    - Hadoop Yarn
    - Hadoop MapReduce

Apache Hadoop MapReduce

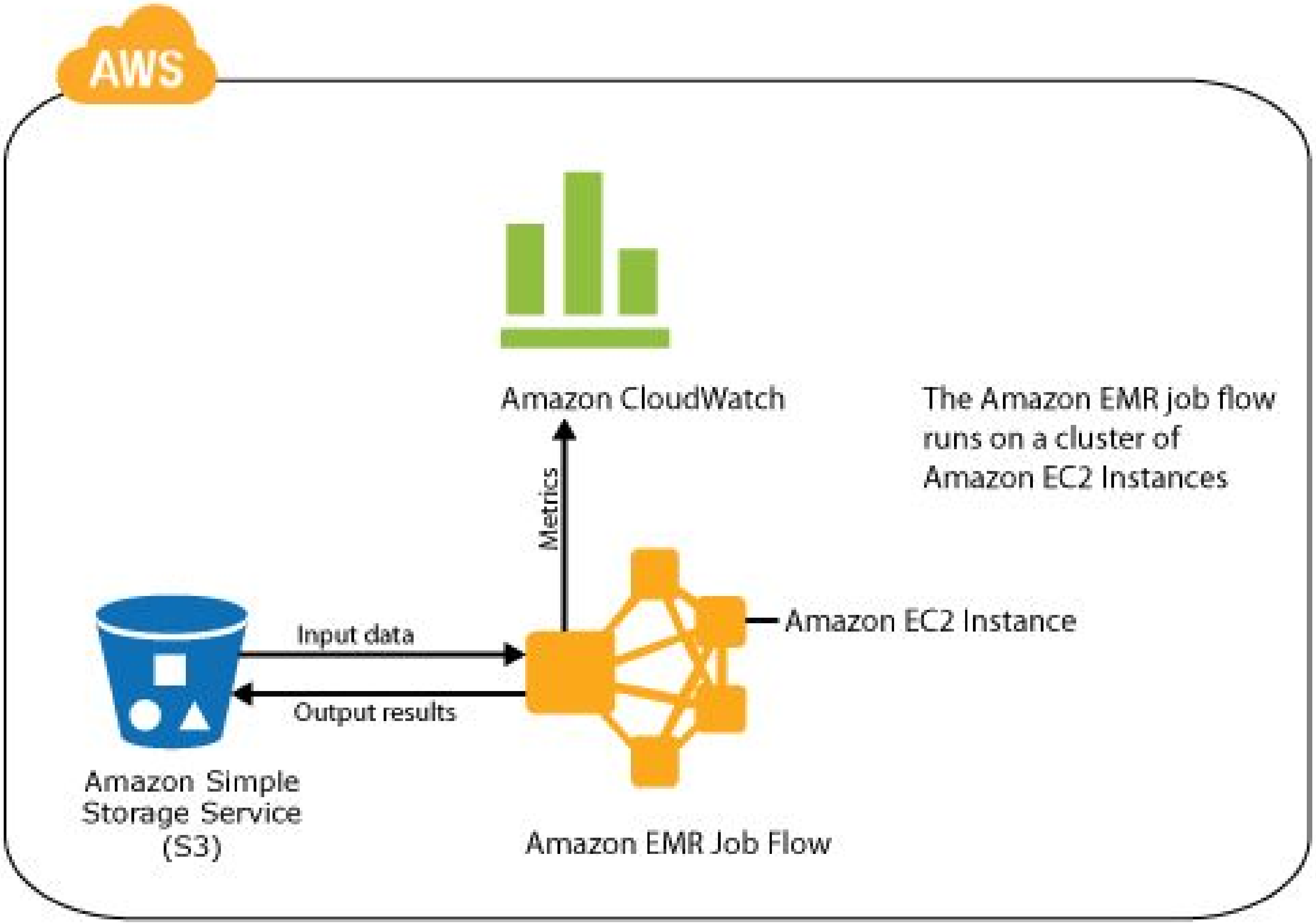
- 3 components:
  - MapReduce API
    - set of libraries for end-users, where you define map and reduce functions
  - MapReduce Framework
    - takes care of all phases of MapReduce (e.g. map phase, sort/shuffle/merge, reduce)
  - MapReduce cluster management system
    - job scheduling, resource management

Hadoop Distributed File System

- Key Ideas:
  - Fault-tolerance through block replication
  - Meant for storing LARGE files (gigabytes or terabytes in size)
  - Write-once-read-many access model
    - no append or random writes
  - Each files is divided into 128 MB blocks are are replicated across 3 different DataNodes
- For more info on how distributed filesystems work, read the GFS paper: http://static.googleusercontent.com/media/research.google.com/en//archive/gfs-sosp2003.pdf
- More on this next lecture

Amazon Elastic MapReduce

- Instead of configuring and setting up your own cluster manually, use Amazon's!
- Elastic - can scale up or down as needed
  - Add or subtract nodes in a cluster
- Underlying framework is still Hadoop
- Can integrate with other Amazon Web Services such as S3 (Storage)


AWS
Amazon CloudWatch
The Amazon EMR job flow
runs on a cluster of
Amazon EC2 Instances
Metrics
Amazon EC2 Instance
Input data
Output results
Amazon Simple
Storage Service
(S3)
Amazon EMR Job Flow

EMR Architecture

- 3 roles of servers (nodes) in an EMR cluster:
  - Master node: distributes MapReduce tasks to nodes in the cluster and monitors the status of task execution
  - Core nodes: execute MapReduce tasks and provide HDFS for storing the data related to task execution. Cannot be removed once instantiated.
  - Task nodes: only execute MapReduce tasks, do not hold data blocks. These can be removed, allowing for elastic scale-down