

Introduction to Database Systems

CSE 414

Lecture 26: Parallel Databases and MapReduce

HW8 (1)

- MapReduce (Hadoop) w/ declarative language (Pig)
- Cluster will run in Amazon's cloud (AWS)
 - Give your credit card (and the credit code you get from us that will cover the costs)
 - Click, click, click... and you have a MapReduce cluster
- We will analyze a real 0.5TB graph
- Processing the entire data takes hours
 - Problems #1,#2,#3: queries on a subset only
 - Problem #4: entire data

HW8 (2)

- HW8 due the last day of the quarter
 - **No late days** (so we can post solutions over the weekend)
 - Corollary: feel free to burn remaining late days on hw7 – but **only** if you have some left
 - Don't wait until the last minute – heavy load on machines then, slow response
- Details, demos, & more in sections this week!
 - Bring your laptop if you can

Amazon Warning

- “We **HIGHLY** recommend you remind students to turn off any instances after each class/session – as this can quickly diminish the credits and start charging the card on file. **You are responsible for the overages.**”
- “AWS customers can now use **billing alerts** to help monitor the charges on their AWS bill. You can get started today by visiting your [Account Activity page](#) to enable monitoring of your charges. Then, you can set up a billing alert by simply specifying a bill threshold and an e-mail address to be notified as soon as your estimated charges reach the threshold.”

Outline

- Today: Parallel Data Processing at Massive Scale (MapReduce)
 - Reading assignment (optional):
Chapter 2 (Sections 1,2,3 only) of Mining of Massive Datasets, by Rajaraman and Ullman
<http://i.stanford.edu/~ullman/mmds.html>

Review

- Why parallel processing?
- What are the possible architectures for a parallel database system?
- What are speedup and scaleup?

Parallel Data Processing at Massive Scale

Data Centers Today

- Large number of commodity servers, connected by high speed, commodity network
- Rack: holds a small number of servers
- Data center: holds many racks

Data Processing at Massive Scale

- Want to process petabytes of data and more
- Massive parallelism:
 - 100s, or 1000s, or 10000s servers
 - Many hours
- Failure:
 - If medium-time-between-failure is 1 year
 - Then 10000 servers have one failure / hour

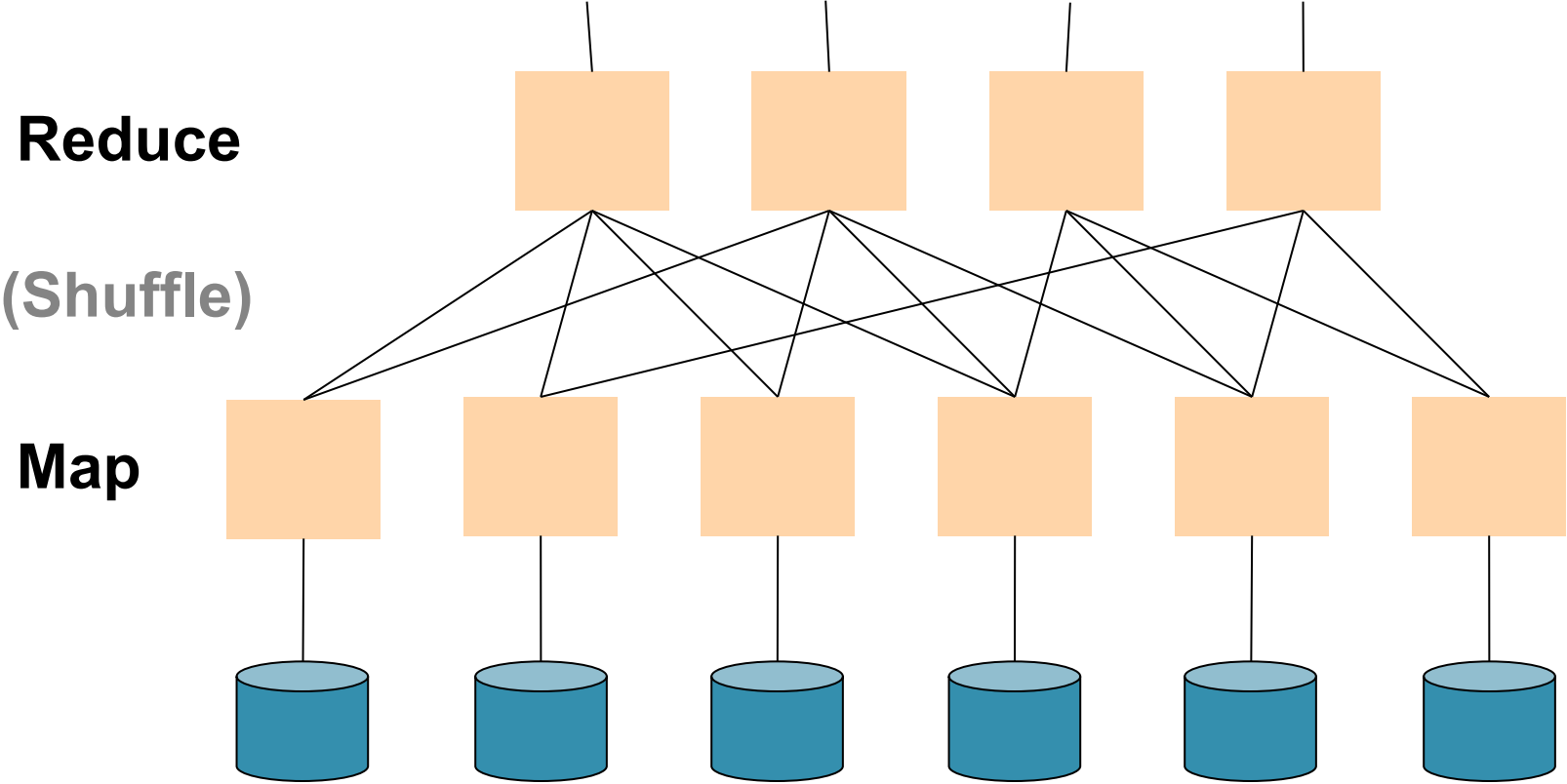
Distributed File System (DFS)

- For very large files: TBs, PBs
- Each file is partitioned into *chunks*, typically 64MB
- Each chunk is replicated several times (≥ 3), on different racks, for fault tolerance
- Implementations:
 - Google's DFS: **GFS**, proprietary
 - Hadoop's DFS: **HDFS**, open source

MapReduce

- Google: paper published 2004
- Free variant: Hadoop
- MapReduce = high-level programming model and implementation for large-scale parallel data processing

Observation: Your favorite parallel algorithm...



Typical Problems Solved by MR

- Read a lot of data
- **Map**: extract something you care about from each record
- Shuffle and Sort
- **Reduce**: aggregate, summarize, filter, transform
- Write the results

Outline stays the same,
map and reduce computations
change to fit the problem

Data Model

Files !

A file = a bag of **(key, value)** pairs

A MapReduce program:

- Input: a bag of **(inputkey, value)** pairs
- Output: a bag of **(outputkey, value)** pairs

Step 1: the **MAP** Phase

User provides the **MAP**-function:

- Input: **(input key, value)**
- Output:
bag of **(intermediate key, value)**

System applies the map function in parallel to all **(input key, value)** pairs in the input file

Step 2: the REDUCE Phase

User provides the REDUCE function:

- Input:
(intermediate key, bag of values)
- Output: bag of output (values)

System groups all pairs with the same intermediate key, and passes the bag of values to the REDUCE function

Example

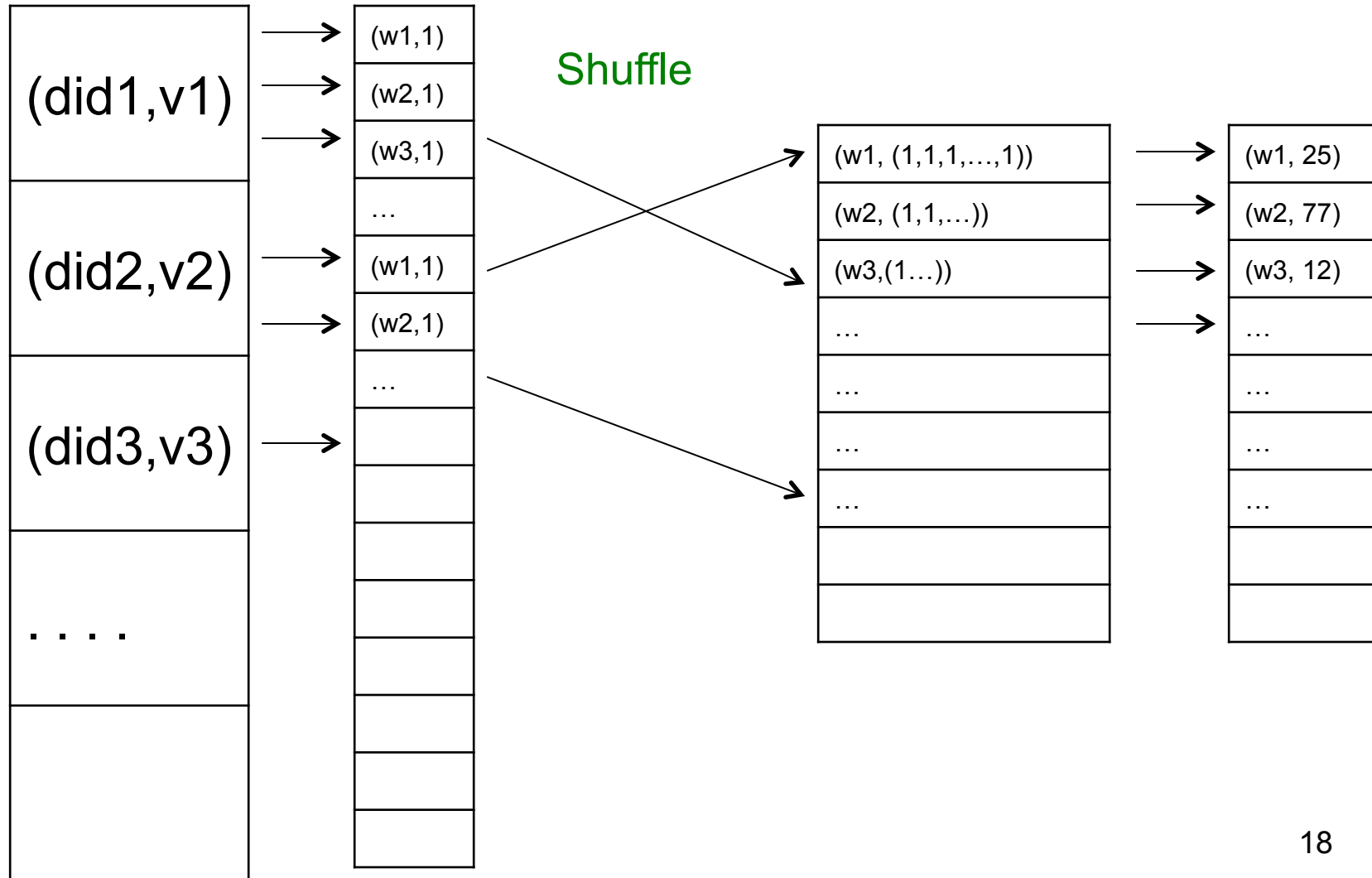
- Counting the number of occurrences of each word in a large collection of documents
- Each Document
 - The **key** = document id (**did**)
 - The **value** = set of words (**word**)

```
map(String key, String value):  
  // key: document name  
  // value: document contents  
  for each word w in value:  
    EmitIntermediate(w, "1");
```

```
reduce(String key, Iterator values):  
  // key: a word  
  // values: a list of counts  
  int result = 0;  
  for each v in values:  
    result += ParseInt(v);  
  Emit(AsString(result));
```

MAP

REDUCE



Jobs v.s. Tasks

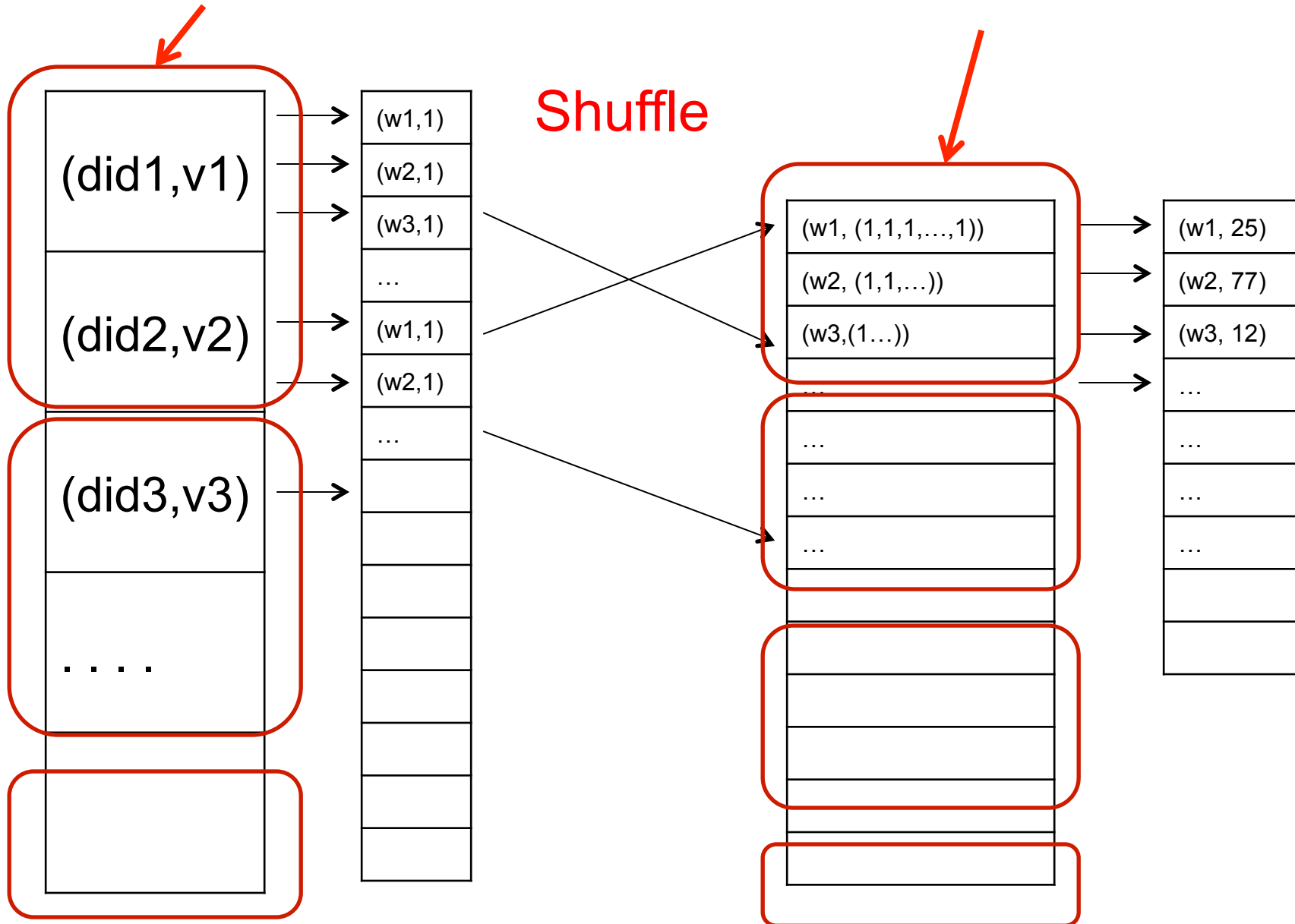
- A **MapReduce Job**
 - One single “query”, e.g. count the words in all docs
 - More complex queries may consists of multiple jobs
- A **Map Task**, or a **Reduce Task**
 - A group of instantiations of the map-, or reduce-function, which are scheduled on a single worker

Workers

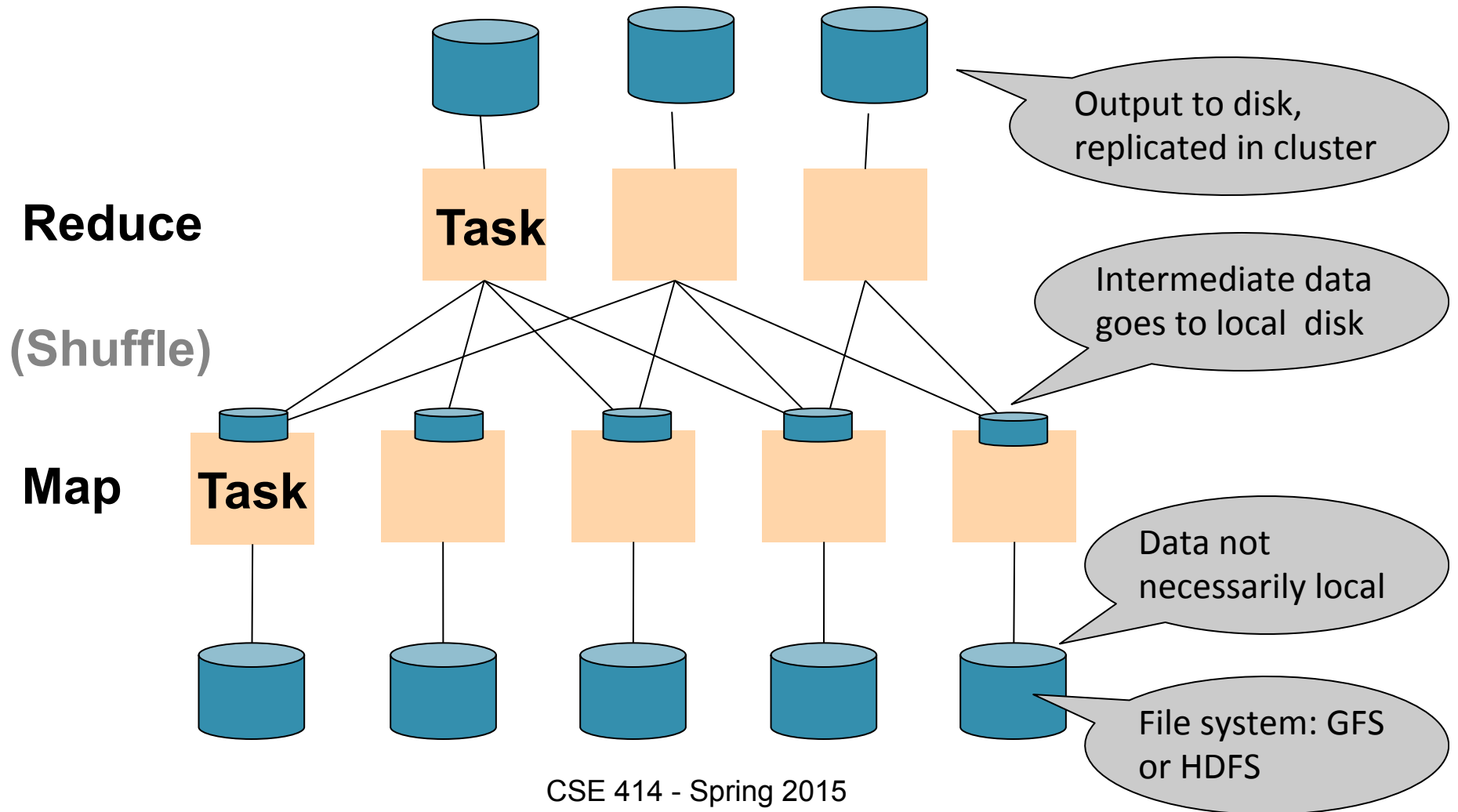
- A **worker** is a process that executes one task at a time
- Typically there is one worker per processor, hence 4 or 8 per node

MAP Tasks

REDUCE Tasks

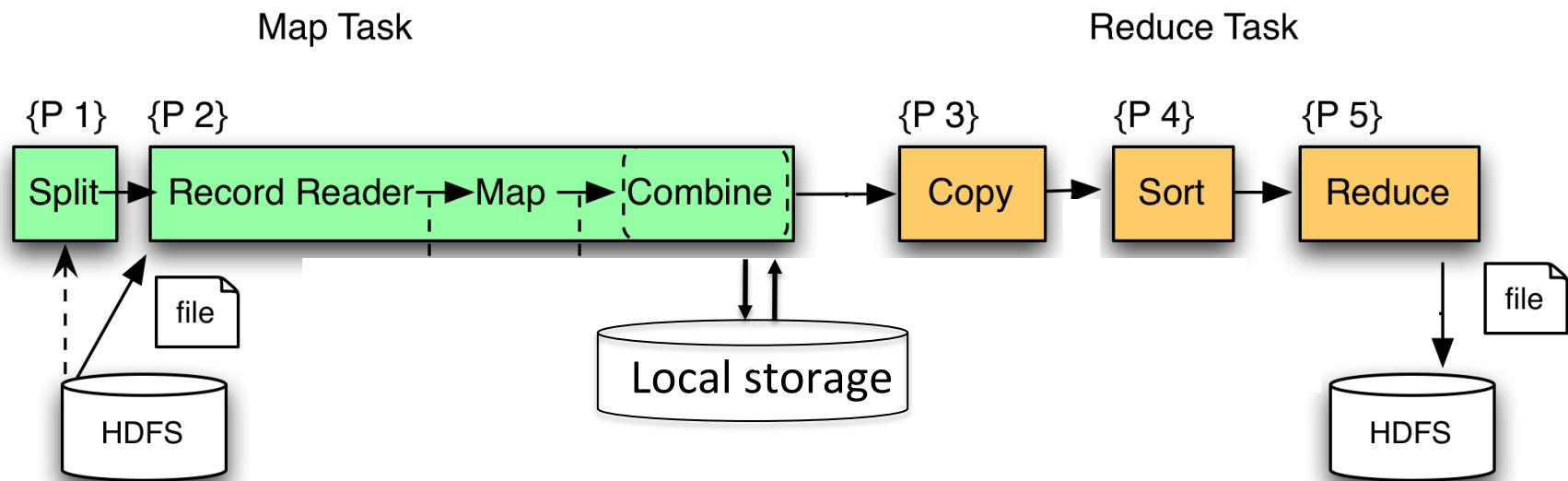


MapReduce Execution Details



MR Phases

- Each Map and Reduce task has multiple phases:



Example: CloudBurst



CloudBurst. Lake Washington Dataset (1.1GB). 80 Mappers 80 Reducers.

Implementation

- There is one master node
- Master partitions input file into *M splits*, by key
- Master assigns *workers* (=servers) to the *M map tasks*, keeps track of their progress
- Workers write their output to local disk, partition into *R regions*
- Master assigns workers to the *R reduce tasks*
- Reduce workers read regions from the map workers' local disks

Interesting Implementation Details

Worker failure:

- Master pings workers periodically,
- If down then reassigns the task to another worker

Interesting Implementation Details

Backup tasks:

- *Straggler* = a machine that takes unusually long time to complete one of the last tasks. Eg:
 - Bad disk forces frequent correctable errors (30MB/s → 1MB/s)
 - The cluster scheduler has scheduled other tasks on that machine
- Stragglers are a main reason for slowdown
- Solution: *pre-emptive backup execution of the last few remaining in-progress tasks*

MapReduce Summary

- Hides scheduling and parallelization details
- However, very limited queries
 - Difficult to write more complex queries
 - Need multiple MapReduce jobs
- Solution: declarative query language

Declarative Languages on MR

- PIG Latin (Yahoo!)
 - New language, like Relational Algebra
 - Open source
- HiveQL (Facebook)
 - SQL-like language
 - Open source
- SQL / Tenzing (Google)
 - SQL on MR
 - Proprietary

Parallel DBMS vs MapReduce

- Parallel DBMS
 - Relational data model and schema
 - Declarative query language: SQL
 - Many pre-defined operators: relational algebra
 - Can easily combine operators into complex queries
 - Query optimization, indexing, and physical tuning
 - Streams data from one operator to the next without blocking
 - Can do more than just run queries: Data management
 - Updates and transactions, constraints, security, etc.

Parallel DBMS vs MapReduce

- MapReduce
 - Data model is a file with key-value pairs!
 - No need to “load data” before processing it
 - Easy to write user-defined operators
 - Can easily add nodes to the cluster (no need to even restart)
 - Uses less memory since processes one key-group at a time
 - Intra-query fault-tolerance thanks to results on disk
 - Intermediate results on disk also facilitate scheduling
 - Handles adverse conditions: e.g., stragglers
 - Arguably more scalable... but also needs more nodes!