Some Reminders for a Seamless Online Class…

- Please turn on your video
- Mute yourself (press and hold spacebar when you'd like to talk)
- Don't do anything you wouldn't do in an in-person class
- I will occasionally check the chat for messages if you'd like to share there instead
- Please say your name before you speak

Recap

- Data-savviness is the future!
- “Classical” relational databases
  - Notion of a DBMS
  - The relational data model and algebra: bags and sets
  - SQL Queries, Modifications, DDL
  - Database Design
  - Views, constraints, triggers, and indexes
  - Query processing & optimization
  - Transactions
- Non-classical data systems
  - Data preparation:
    - Semi-structured data and document stores
    - Unstructured data and search engines
  - Data Exploration:
    - Cell-structured data and spreadsheets
    - Dataframes and dataframe systems
    - OLAP, summarization, and visual analytics
  - Batch Analytics:
    - Compression and column stores
    - **Parallel data processing and map-reduce**

Some material drawn from Mining Massive Data Sets, Leskovec, Rajaraman, Ullman

Parallel data processing and map-reduce

- We've studied OLAP — a specialization of relational databases targeted at business analytics and reporting at scale with data cube materialization and column stores
- Today, we're going to be studying the primitives for processing large volumes of relational, unstructured, or semi-structured data at scale
- Often, when we're trying to process really large volumes of data, we need to span across multiple nodes/machines
    - This hasn't been a focus of our class so far
- We'll start by covering what map-reduce offers, before switching over to cover parallel databases

Let's revisit search engines

- To create an inverted index, we need to read all the webpages
  - Size of the web: 20+B web pages x 20 KB = 400+ TB
  - Disk reading speed ~50MBPS
  - On a single disk drive: (400 x 1000 x 1000) / (35 x 60 x 60 x 24 x 30)
    - 4 months to read the web!
  - If each hard-drive can store 1TB, then 400 hard-drives to store the web.

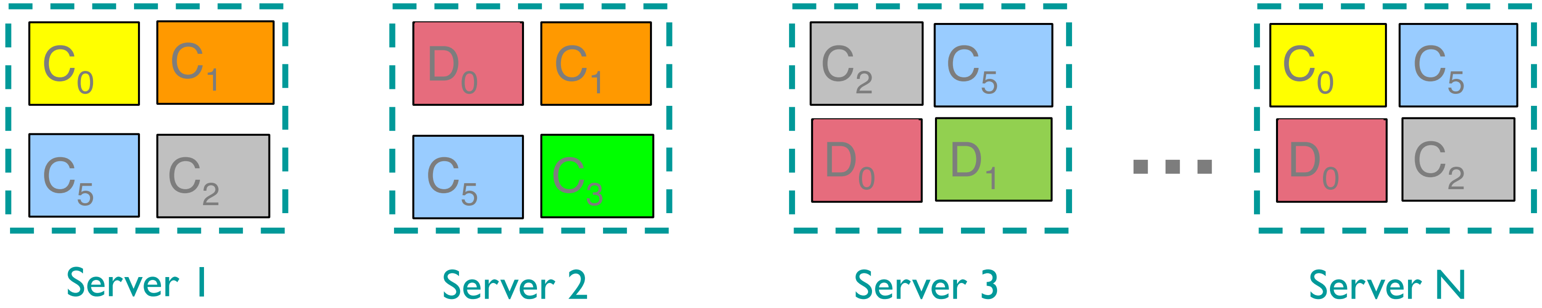
- Instead, parallelize!
  - Modern data system architectures use many cheap (commodity) machines connected by cheap network (ethernet)
- Q: If we could read from all 400 hard-drives at once, we could get the job done in how much time?
  - 120 x 24 hard-drive-hours / 400 hard-drives = 7.2 hours!

Large Scale Data Processing

- Challenges:
  - How do we specify the processing task?
  - How do we distribute this processing across these many machines?
    - Transferring data across network is expensive
  - How do we deal with failures?
    - If a server dies once in a 100 days, then if we have a 1000 servers, on average 10 will die per day

Google's Approach: Circa '00s

- First component: a distributed, replicated file system
  - GFS, modern open-source incarnation: HDFS
  - Just like a file system in your machine, but each file is replicated across machines for reliability
  - Files are very large; rarely updated, but reading and appending are more common
  - There is a "coordinator" node that keeps track of which file is where



- Second component: a distributed data processing programming paradigm
  - Map-Reduce, modern open-source incarnation
  - The processing usually happens "close" to the data if possible to avoid moving data across network (but in some cases unavoidable)

Map-Reduce

- Let’s say we have the entire web crawl and we want to count the # of times each word appears on the web
  - The web is stored across 400 machines, each with its hard-drive

- Q: How would you go about doing this?

Map-Reduce Overview

- Developer-provided Processing Step 1: **Map**
  - Read a lot of data and extract something of value

- Internal Step 2: Group by Keys (**Shuffle**)

- Developer-provided Processing Step 2: **Reduce**
  - Aggregate, summarize, filter, transform data organized by key

Map-Reduce Overview

- Developer-provided Processing Step 1: **Map**
    - Read a lot of data and extract something of value
    - *For each document, emit <word: 1> pairs — key: value pairs*

- Internal Step 2: Group by Keys (**Shuffle**)
    - *Group/sort pairs based on word, so: <word 1, 1>, <word 1, 1>, … <word 1, 1> <word2, 1> …*

- Developer-provided Processing Step 2: **Reduce**
    - Aggregate, summarize, filter, transform data organized by key
    - *For each word, sum up the total number of 1s in the value*

Provided by the programmer

MAP:
Read input and produces a set of key-value pairs

Group by key:
Collect all pairs with same key

Provided by the programmer

Reduce:
Collect all values belonging to the key and output

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. '"The work we're doing now -- the robotics we're doing -- is what we're going to need ……………………..

Document

(The, 1)
(crew, 1)
(of, 1)
(the, 1)
(space, 1)
(shuttle, 1)
(Endeavor, 1)
(recently, 1)
….

(key, value)

(crew, 1)
(crew, 1)
(space, 1)
(the, 1)
(the, 1)
(the, 1)
(shuttle, 1)
(recently, 1)
…

(key, value)

(crew, 2)
(space, 1)
(the, 3)
(shuttle, 1)
(recently, 1)
…

(key, value)

Only sequential reads

The Map-Reduce Paradigm is General!

- Simply change the map and reduce functions to fit new applications
- Semantics:
  - Programmer specifies two methods:
    - _Map(k, v) → <k’, v’>*_
      - Takes a key-value pair and outputs a set of key-value pairs
      - E.g., key is the filename, value is a single line in the file
      - There is one Map call for every (k,v) pair
    - _Reduce(k’, <v’>*) → <k’, v’’>*_
      - All values v’ with same key k’ are reduced together and processed in v’ order
      - There is one Reduce function call per unique key k’

For Our Word Count Example

- Map (key, value)
    - //key is documentname, value is contents
    - For each word in value
        - Output (word, 1)

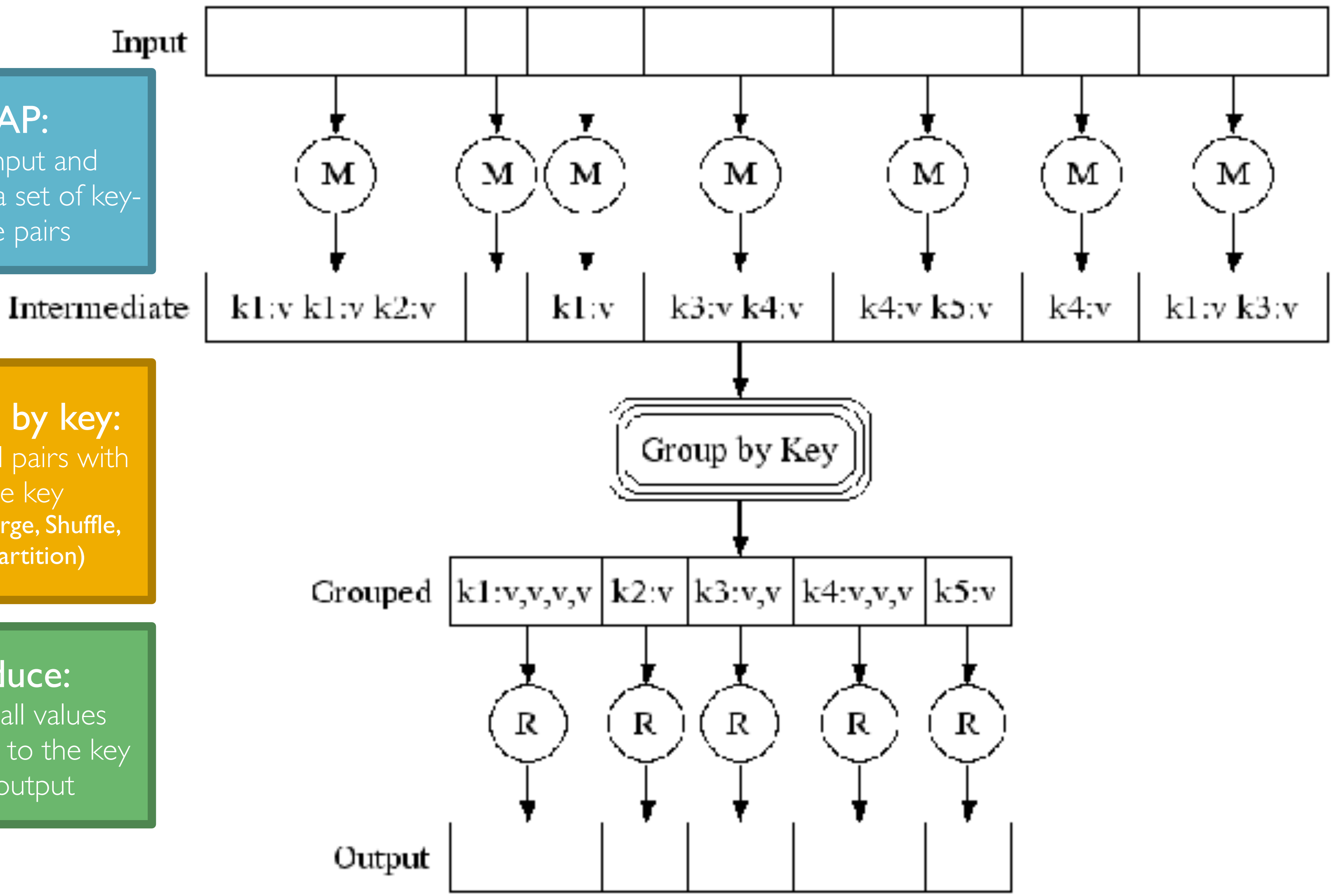
- Reduce (key, values)
    - // key is word, values is a list of 1s
    - Output (key, size (values))

What conceptually happens

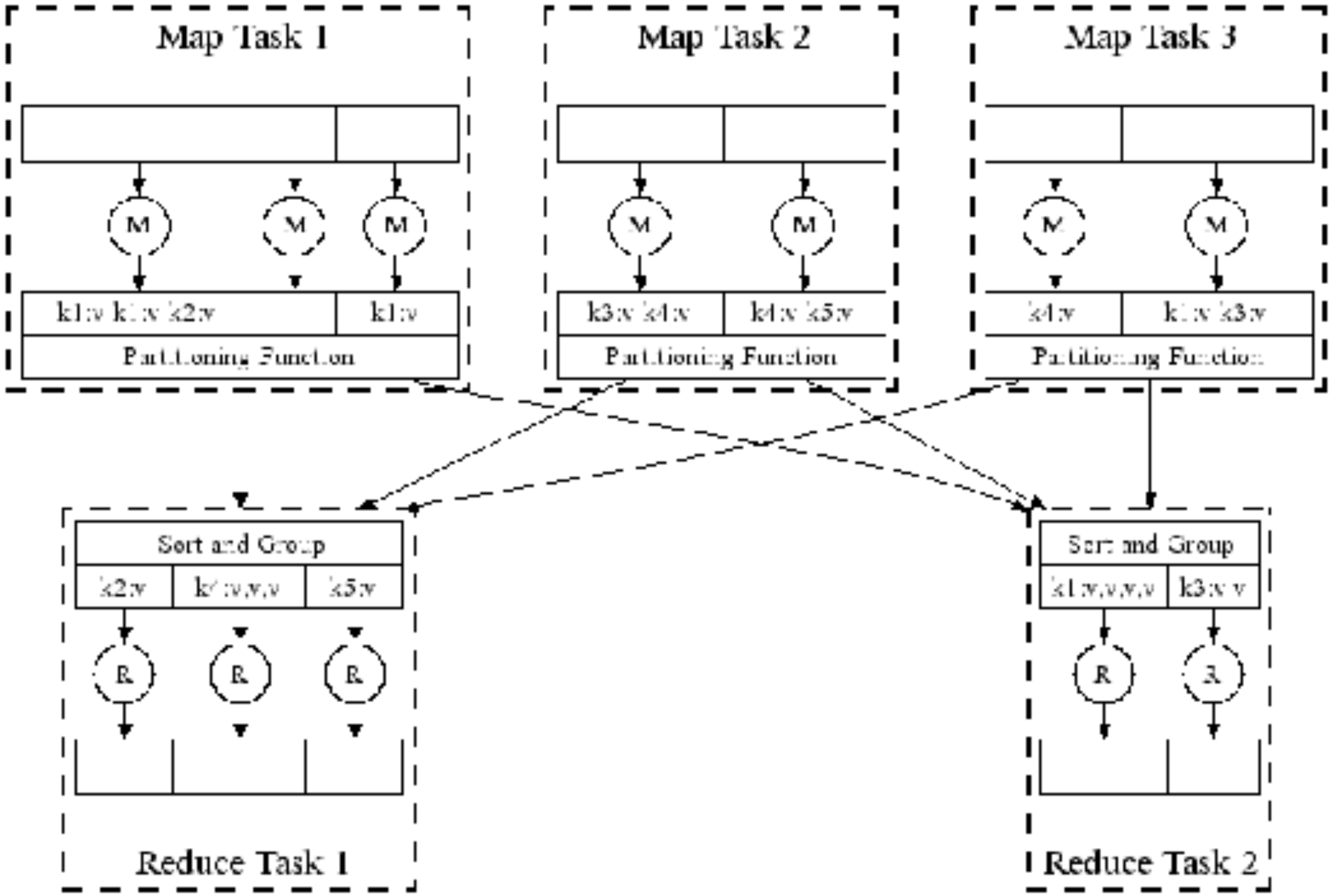
MAP:
Read input and produces a set of key-value pairs

Group by key:
Collect all pairs with same key
(Hash merge, Shuffle, Sort, Partition)

Reduce:
Collect all values belonging to the key and output



What actually happens



What does the Map-Reduce Paradigm Handle?

- Assigning tasks to machines
- Performing the "shuffle" step
- Handling failures and inter-machine communication
- In some cases, can apply a "combiner" at the map task itself to reduce the amount of data shuffled, especially if the reduce function has some nice commutative properties
  - For example, we can sum up the 1s for each word for each document before shuffling across network

A slightly harder case: inverted index construction… remember this slide?

- Phase 1:
  - For each document, generate a "canonicalized term, documentID" pair for all non-stop-word terms in the document
  - Can encode more information with the pair, e.g., list of locations, frequency of occurrence, etc.
- Phase 2:
  - Sort these pairs based on the term, documentID pair
  - May need to use two or multi-pass sort (remember query processing!)
- Phase 3:
  - Read in the sorted pairs based on term, concatenate documentID together to form a posting list for that term
  - (We may be able to merge Phase 3 with Phase 2)

Q: How would we use M-R to construct the inverted index?

- Obama -> 10, 24, 125, 259, 1025, 2314, …
- Bush -> 10, 15, 17, 259, 2001, 2547, …
- Clinton -> 10, 15, 24, 17, 125, 1005, 2001, 2347, …