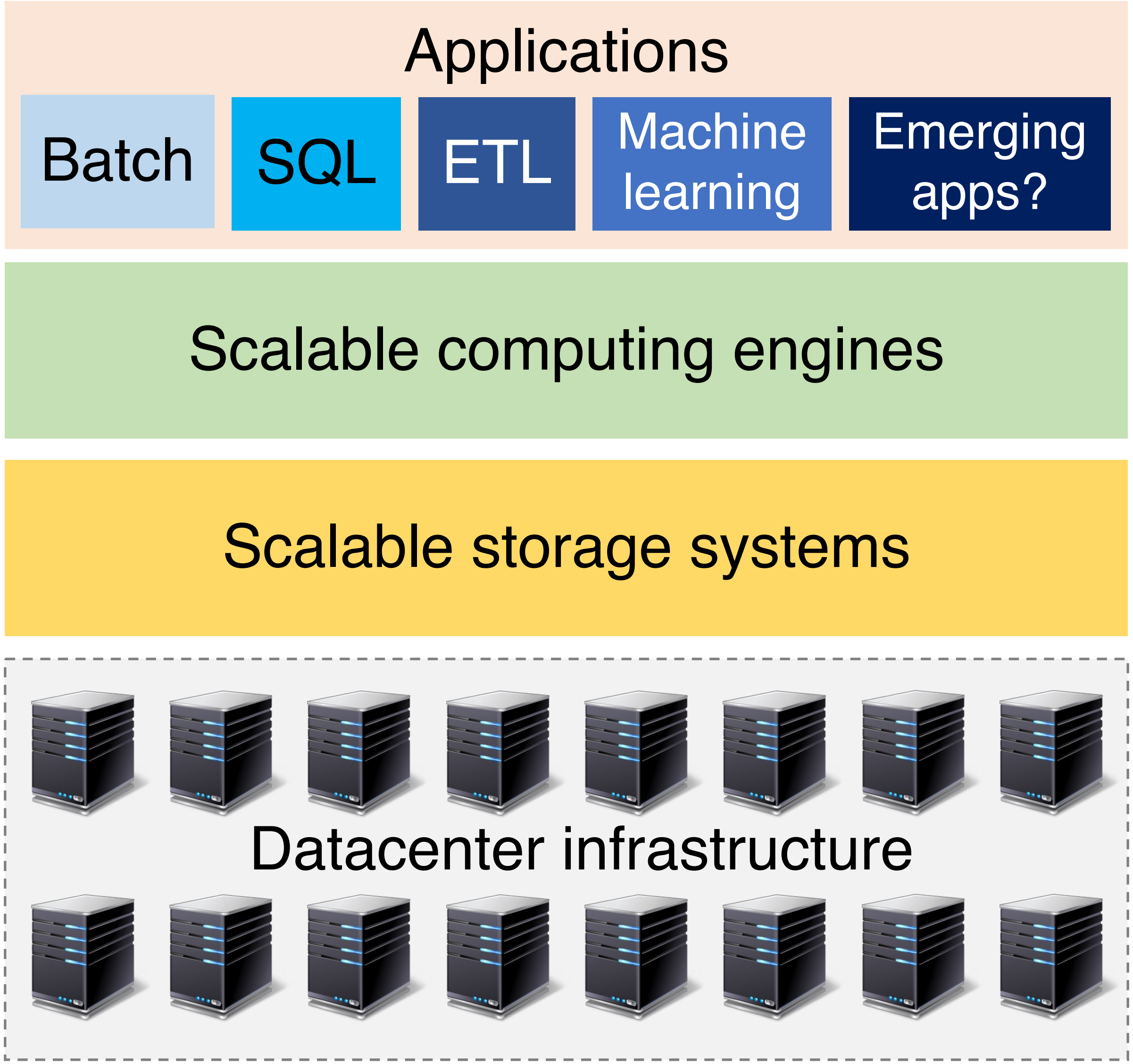
Google MapReduce

DS 5110: Big Data Systems (Spring 2023)

Lecture 3b

Yue Cheng

UNIVERSITY of VIRGINIA



Applications

Batch

SQL

ETL

Machine learning

Emerging apps?

Scalable computing engines

Scalable storage systems

Datacenter infrastructure

The big picture (motivation)

- Datasets are too big to process using a single computer

The big picture (motivation)

- Datasets are too big to process using a single computer

- Good parallel processing engines are rare (back then in the late 90s)

The big picture (motivation)

- Datasets are too big to process using a single computer

- Good parallel processing engines are rare (back then in the late 90s)

- Want a parallel processing framework that:
  - is **general** (works for many problems)
  - is **easy to use** (no locks, no need to explicitly handle communication, no race conditions)
  - can **automatically parallelize** tasks
  - can **automatically handle machine failures**

Context (Google circa 2000)

- Starting to deal with massive datasets
- But also addicted to cheap, unreliable hardware
  - Young company, expensive hardware not practical
- Only a few expert programmers can write distributed programs to process them
  - Scale so large jobs can complete before failures

Context (Google circa 2000)

- Starting to deal with massive datasets
- But also addicted to cheap, unreliable hardware
  - Young company, expensive hardware not practical
- Only a few expert programmers can write distributed programs to process them
  - Scale so large jobs can complete before failures
- **Key question:** how can every Google engineer be imbued with the ability to write parallel, scalable, distributed, fault-tolerant code?
- **Solution:** abstract out the redundant parts
- **Restriction:** relies on job semantics, so restricts which problems it works for

Application: Word Count

```
cat data.txt
     | tr –s ‘[[:punct:][:space:]]’ ‘\n’
     | sort | uniq -c
```

```
SELECT count(word), word FROM data
     GROUP BY word
```

Deal with multiple files?

Deal with multiple files?

1. Compute word counts from individual files

Deal with multiple files?

1. Compute word counts from individual files

2. Then merge intermediate output

Deal with multiple files?

1. Compute word counts from individual files
2. Then merge intermediate output
3. Compute word count on merged outputs

What if the data is too big to fit in one computer?

What if the data is too big to fit in one computer?

1. In parallel, send to worker:
   - Compute word counts from individual files
   - Collect results, wait until all finished

What if the data is too big to fit in one computer?

1. In parallel, send to worker:
   - Compute word counts from individual files
   - Collect results, wait until all finished

2. Then merge intermediate output

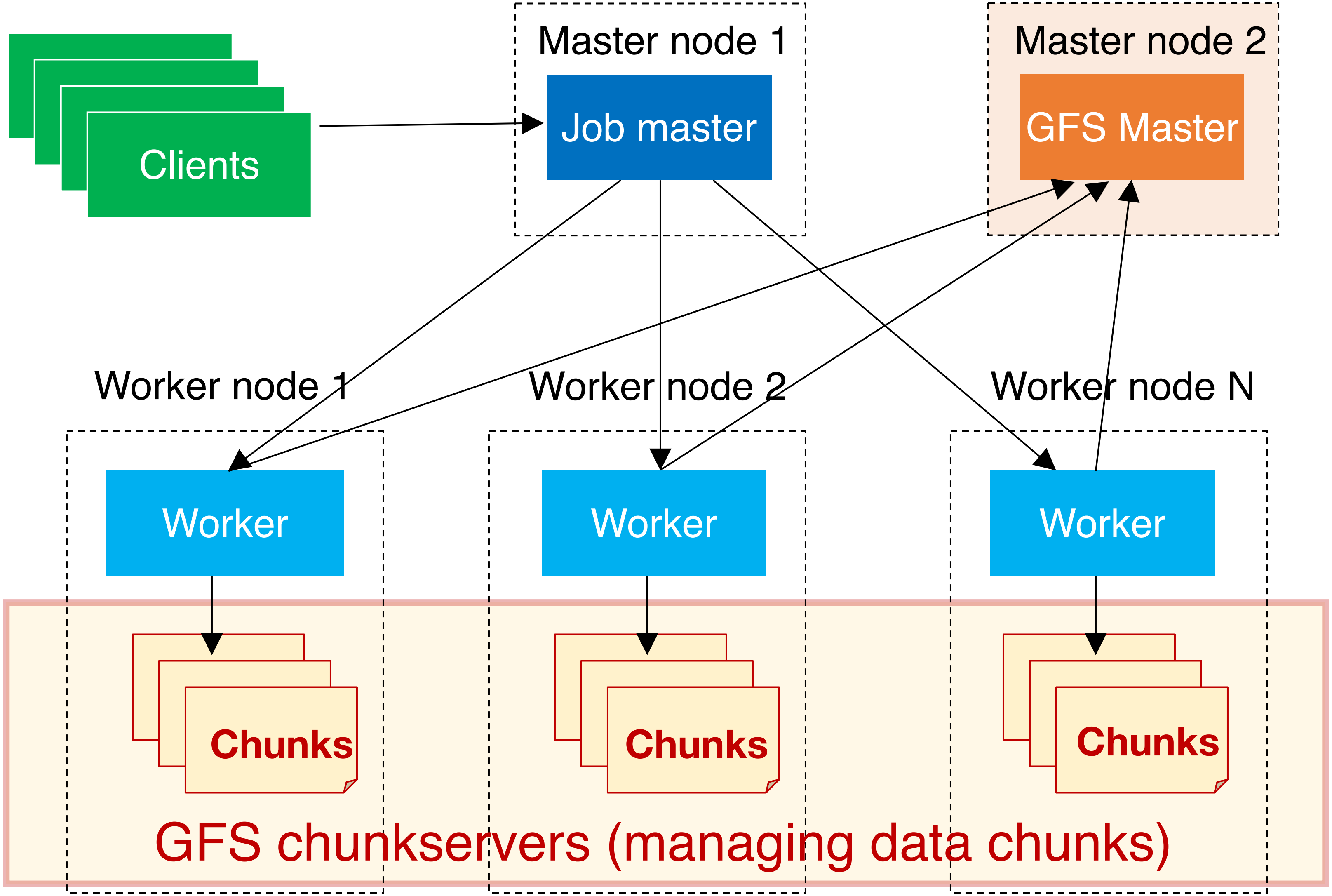
What if the data is too big to fit in one computer?

1. In parallel, send to worker:
   - Compute word counts from individual files
   - Collect results, wait until all finished

2. Then merge intermediate output

3. Compute word count on merged intermediates

MapReduce+GFS: Put everything together



MapReduce: Programming interface

- `map(k1, v1) → list(k2, v2)`
  - Apply function to (`k1, v1`) pair and produce set of intermediate pairs (`k2, v2`)

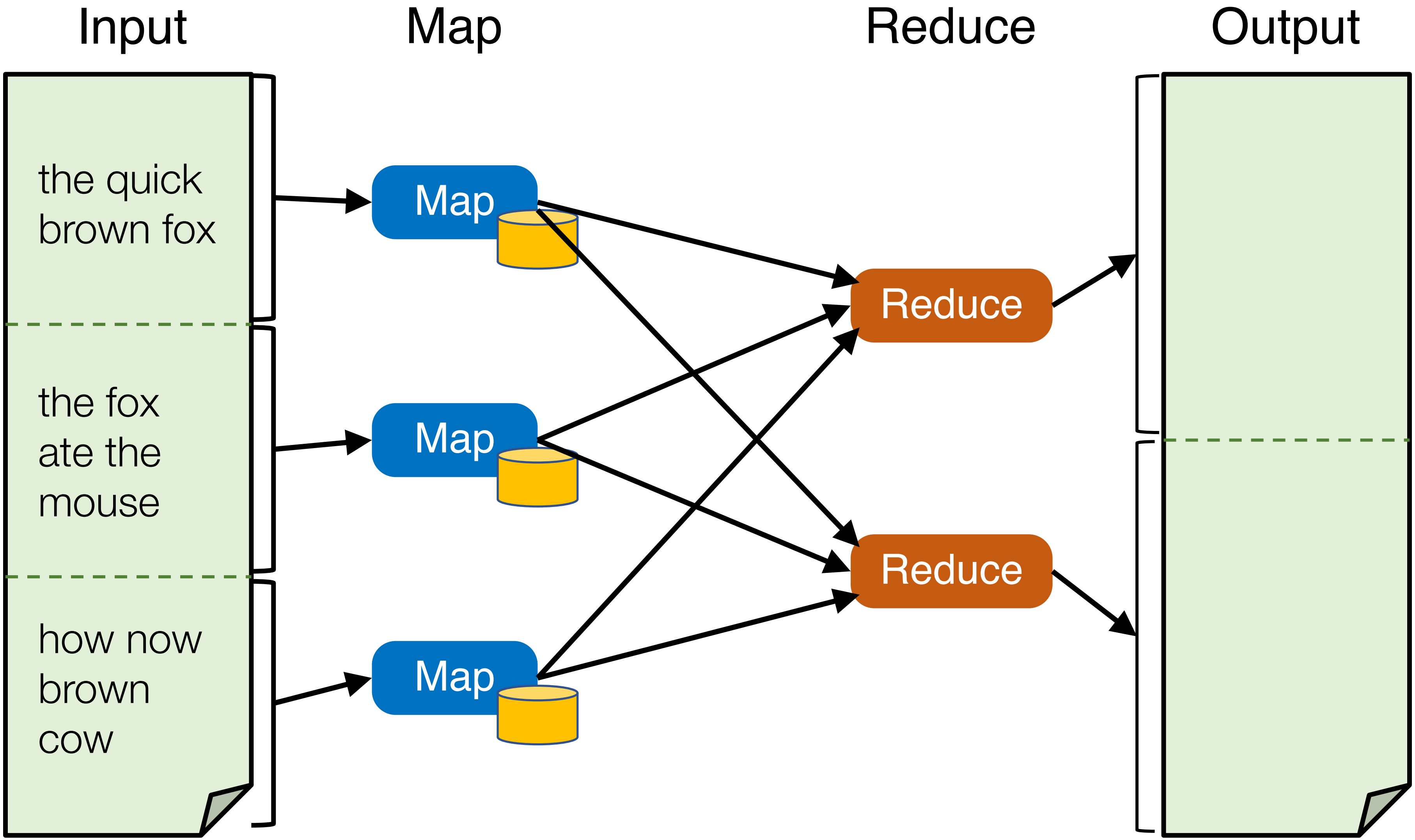
- `reduce(k2, list(v2)) → list(k3, v3)`
  - Apply aggregation (reduce) function to values
  - Output results

MapReduce: Word Count

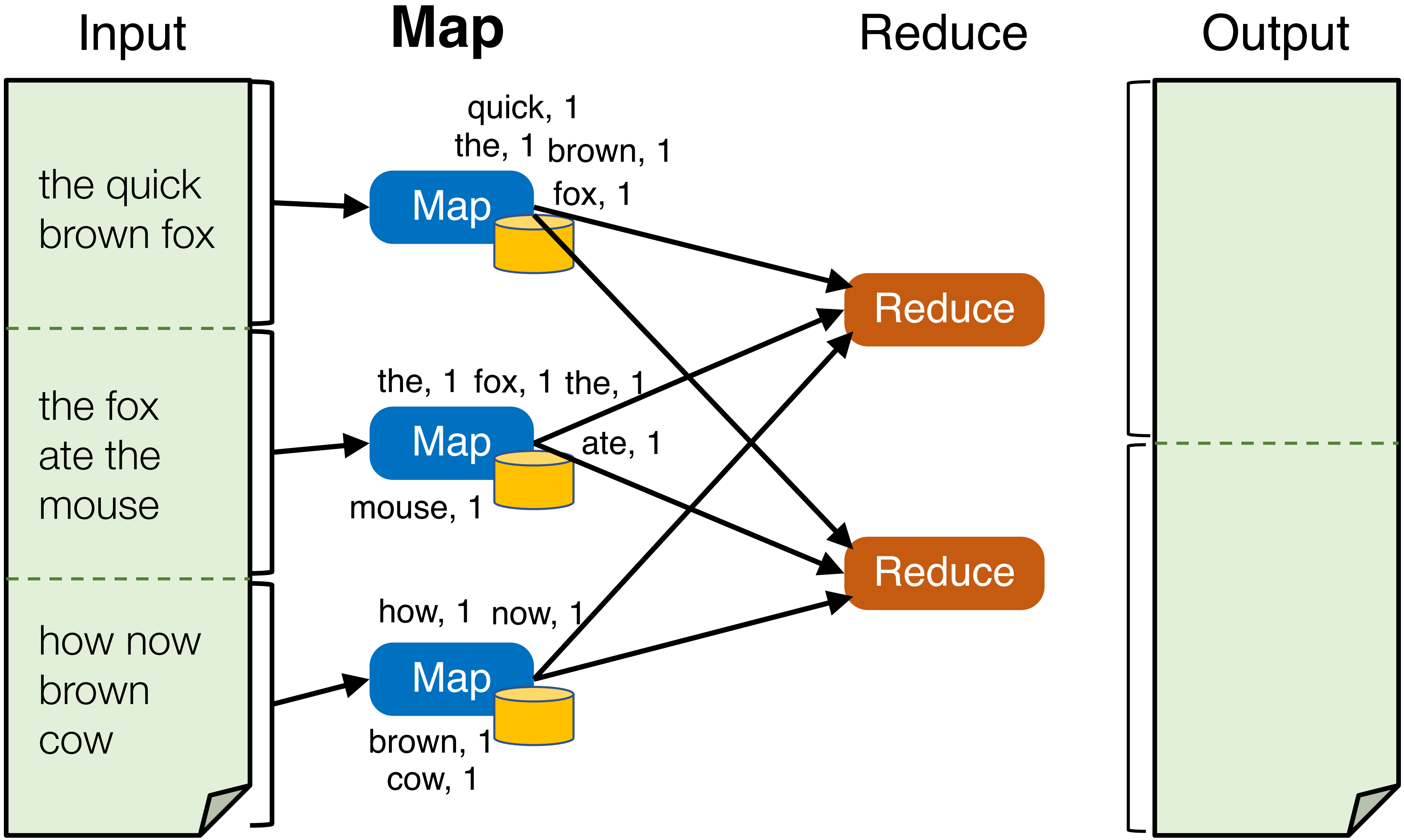
```
map(key, value):
    for each word w in value:
        EmitIntermediate(w, “1”);

reduce(key, values):
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    Emit(AsString(result));
```

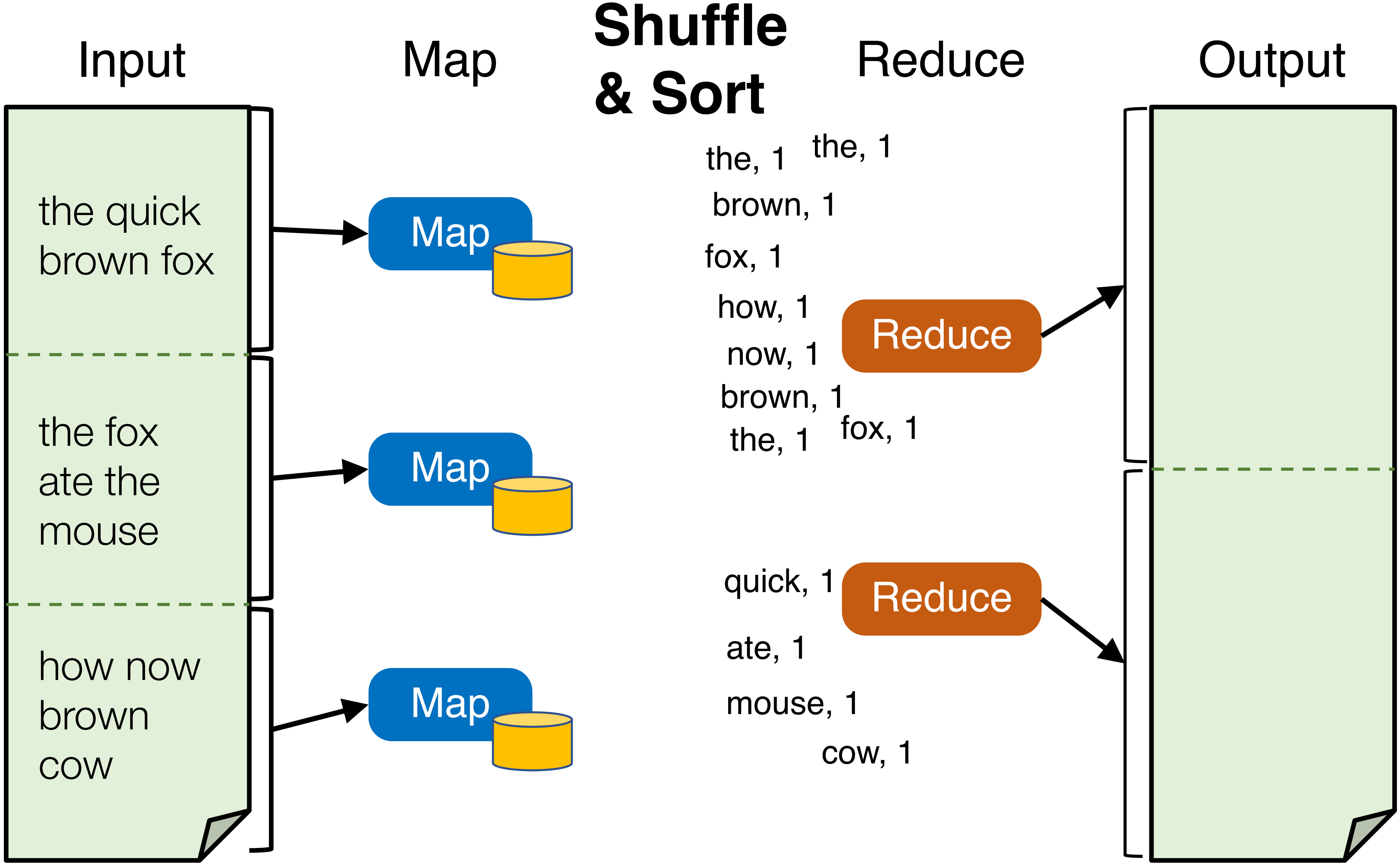
Word Count execution



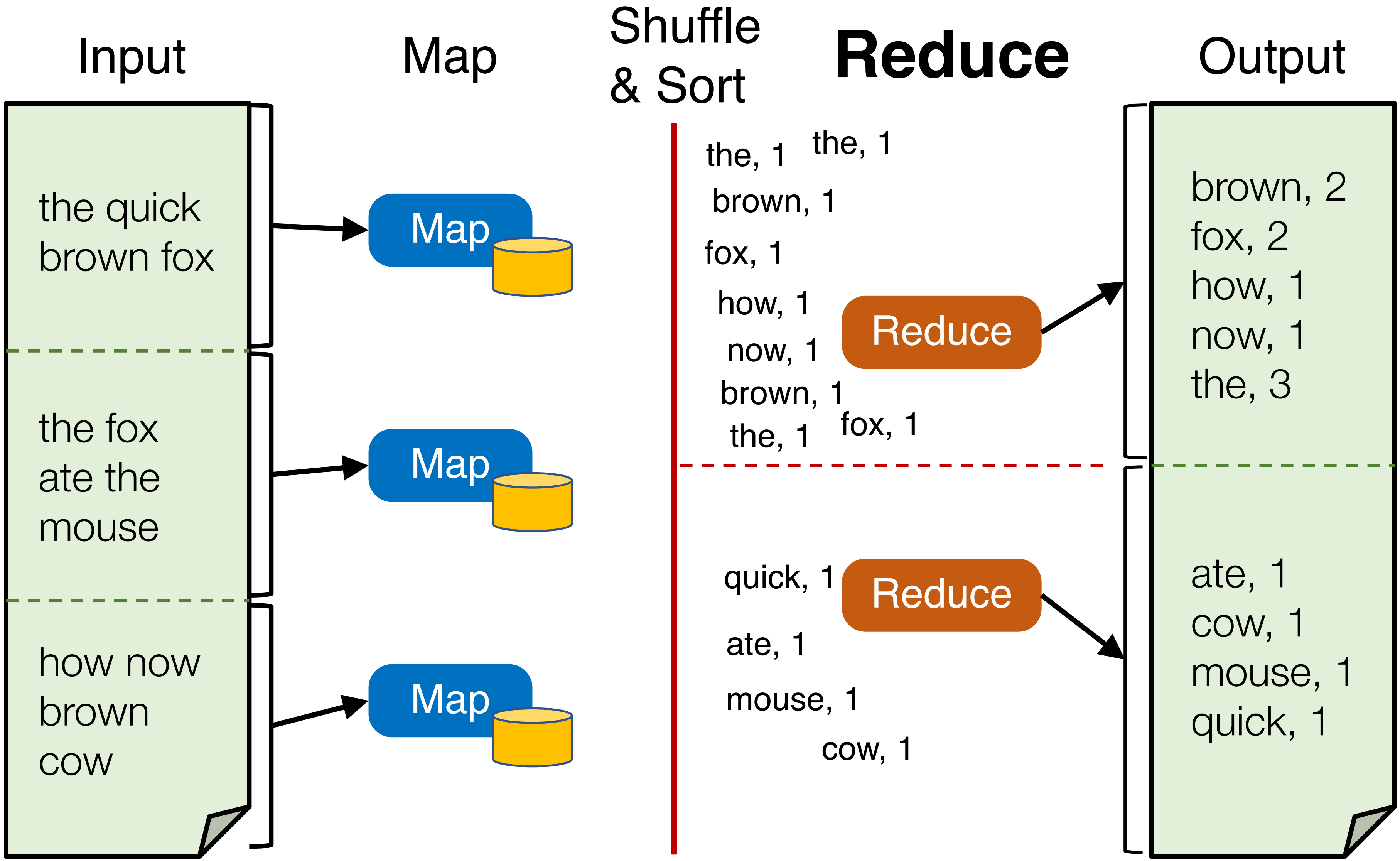
Word Count execution



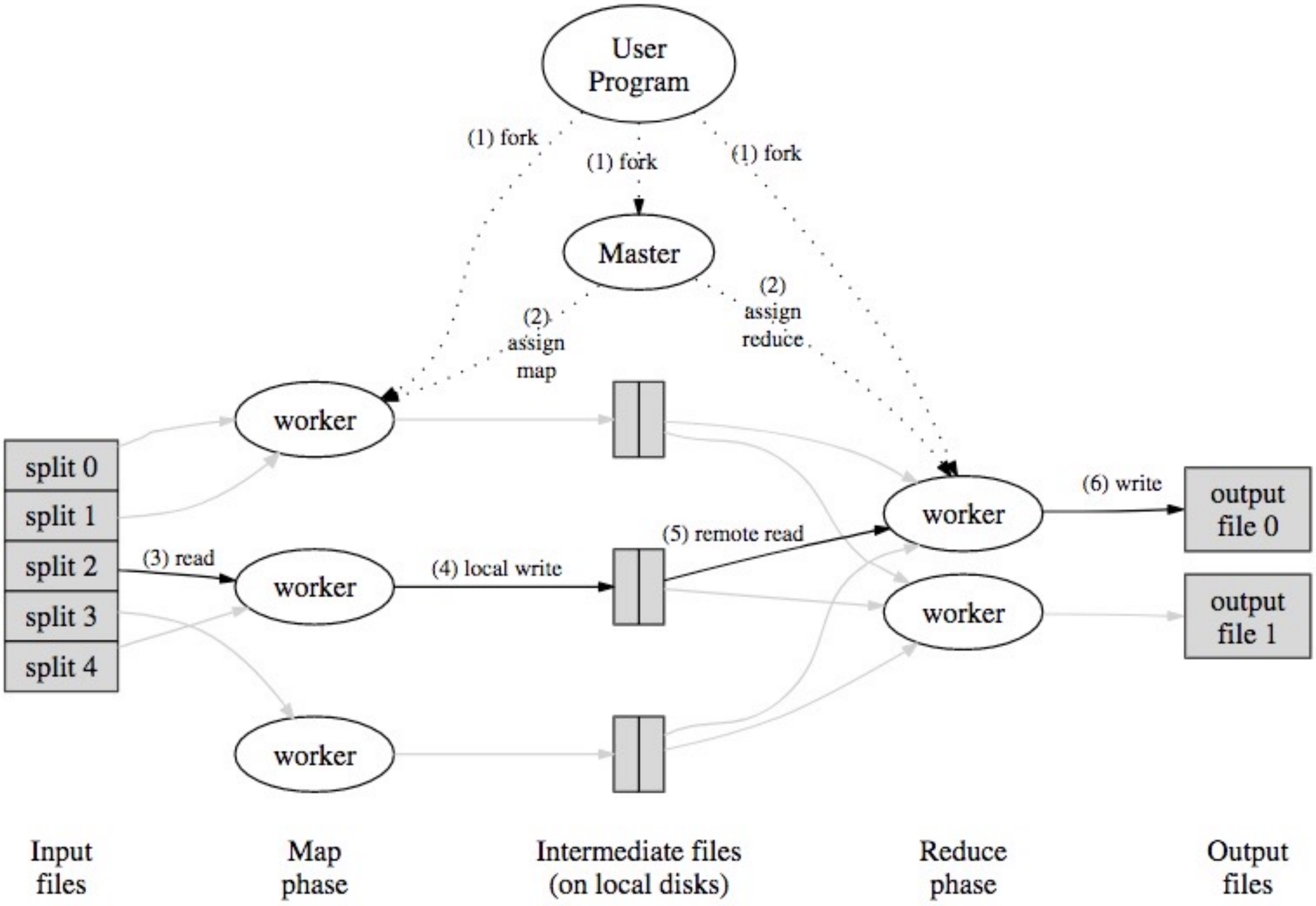
Word Count execution



Word Count execution

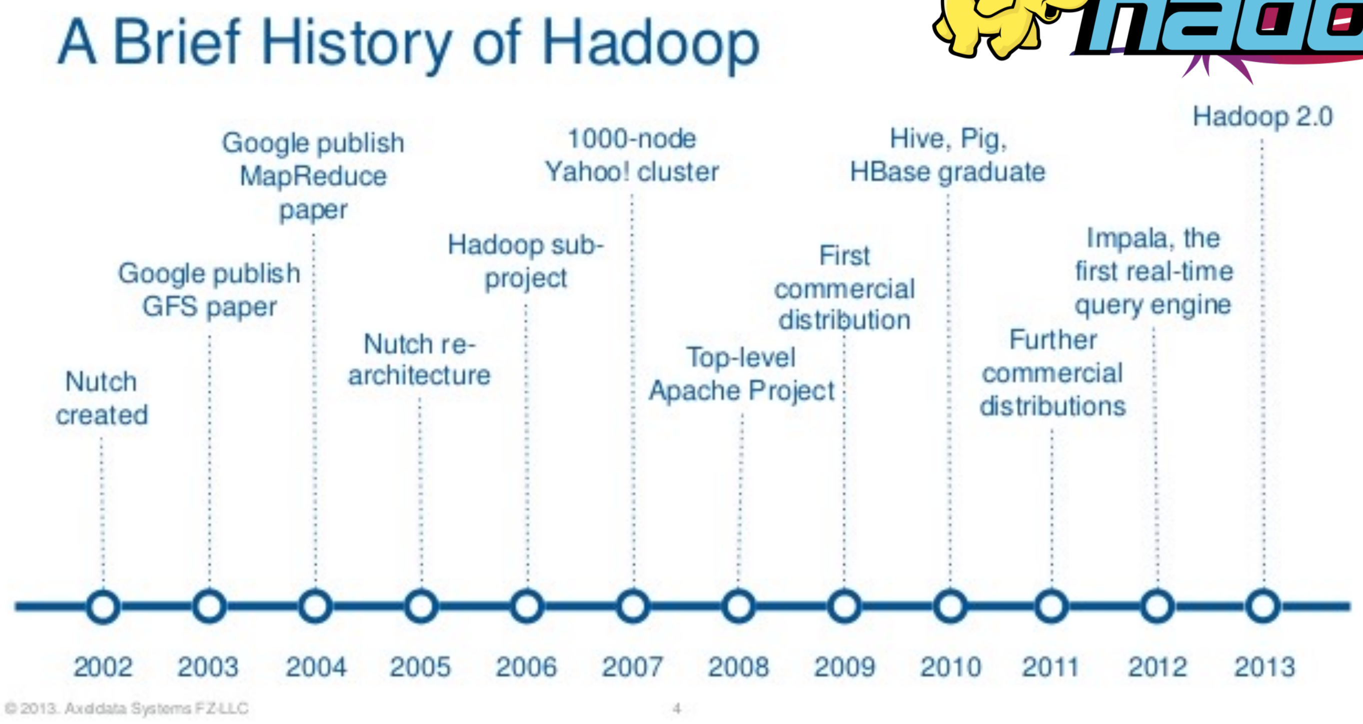


MapReduce data flows in paper



How it started: Apache Hadoop

- An open-source implementation of Google's MapReduce framework
    - Hadoop MapReduce atop Hadoop Distributed File System (HDFS)



A Brief History of Hadoop

2002 Nutch created

2003 Google publish GFS paper

2004 Google publish MapReduce paper

2005 Nutch re-architecture

2006 Hadoop sub-project

2007 1000-node Yahoo! cluster

2008 Top-level Apache Project

2009 First commercial distribution

2010 Hive, Pig, HBase graduate

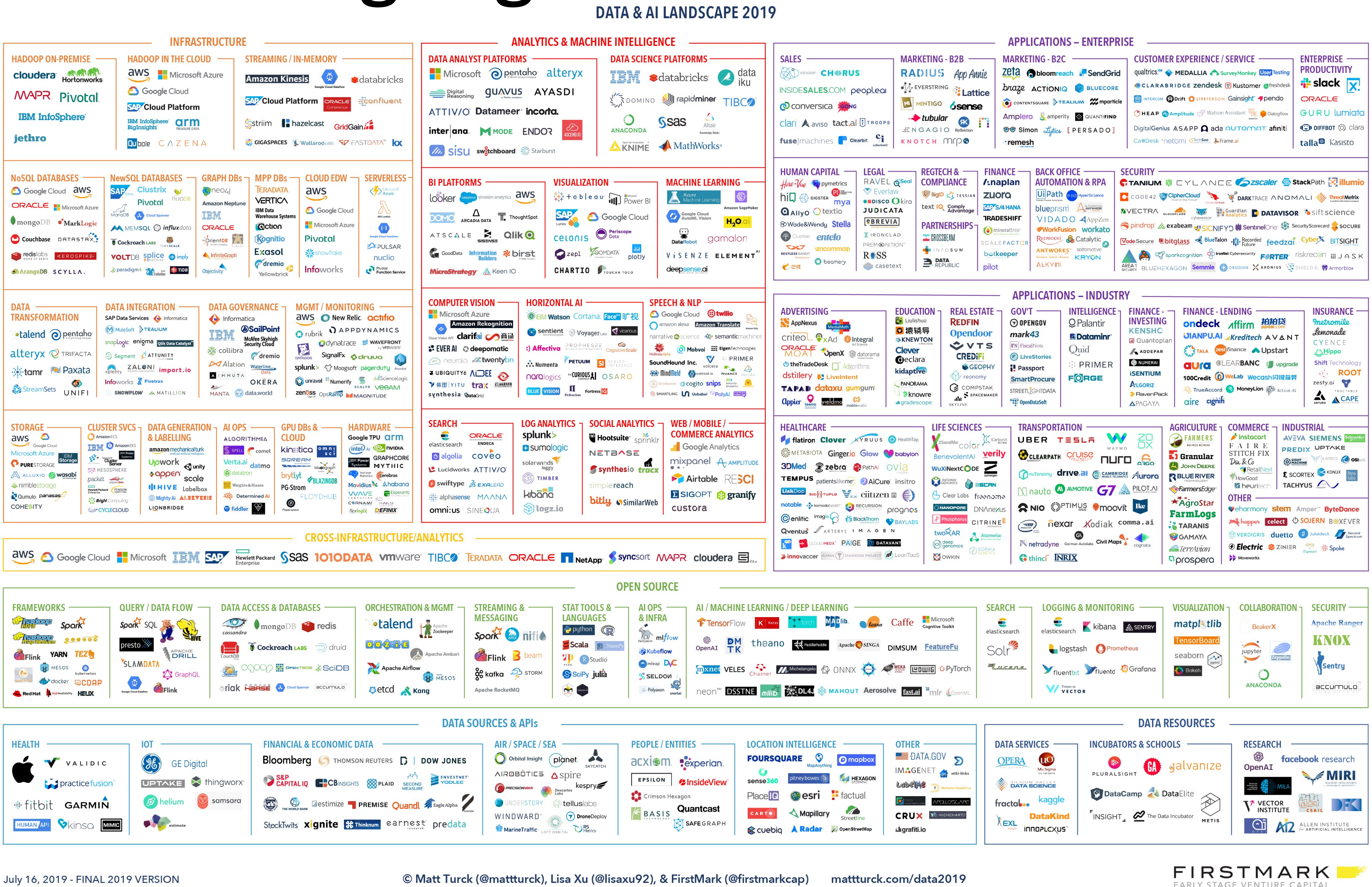
2011 Further commercial distributions

2012 Impala, the first real-time query engine

2013 Hadoop 2.0

© 2013 Axiddata Systems FZ-LLC 4

How it's going ...

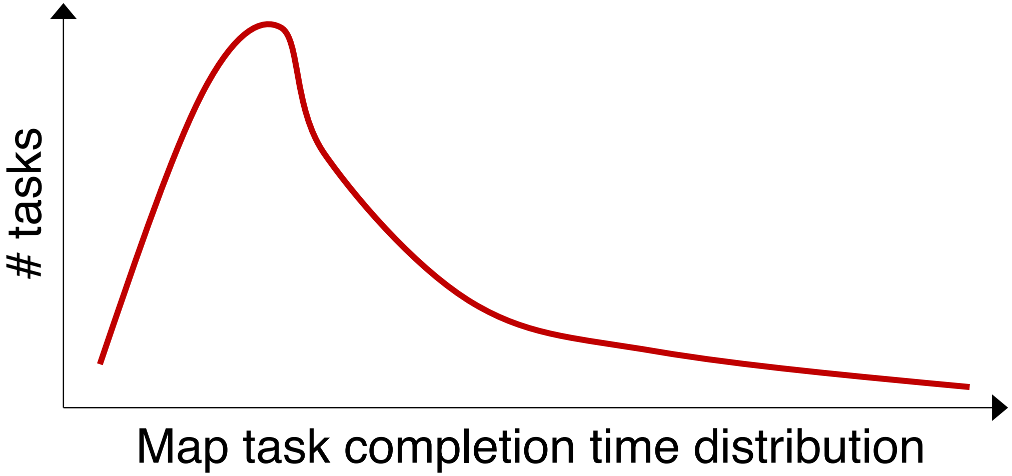


July 16, 2019 - FINAL 2019 VERSION

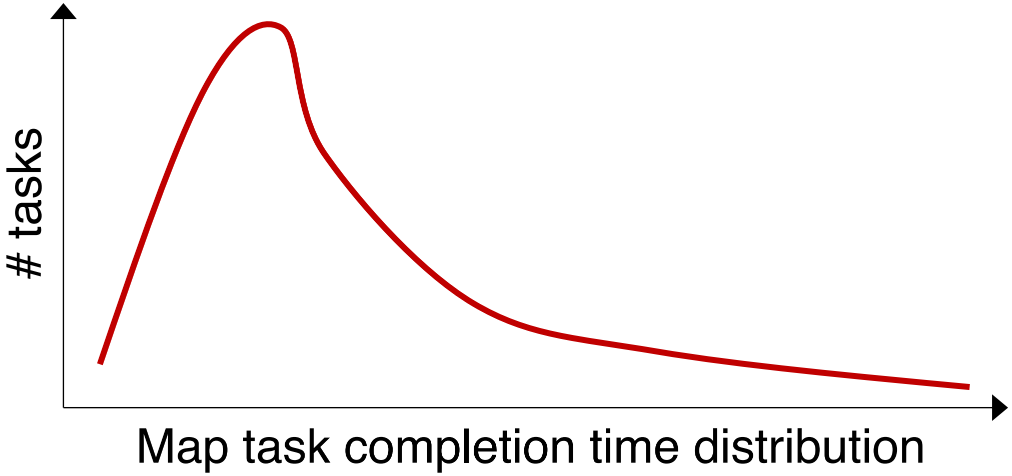
© Matt Turck (@mattturck), Lisa Xu (@lisaxu92), & FirstMark (@firstmarkcap)

mattturck.com/data2019

Stragglers



Stragglers



- **Tail execution time** means some workers (always) finish late

- Q: How can MR work around this?
  - Hint: its approach to **fault-tolerance** provides the right tool

Resilience against stragglers

- If a task is going slowly (i.e., straggler):
  - Launch second copy of task on another node
  - Take the output of whichever finishes first

More design

- Master failure

- Locality

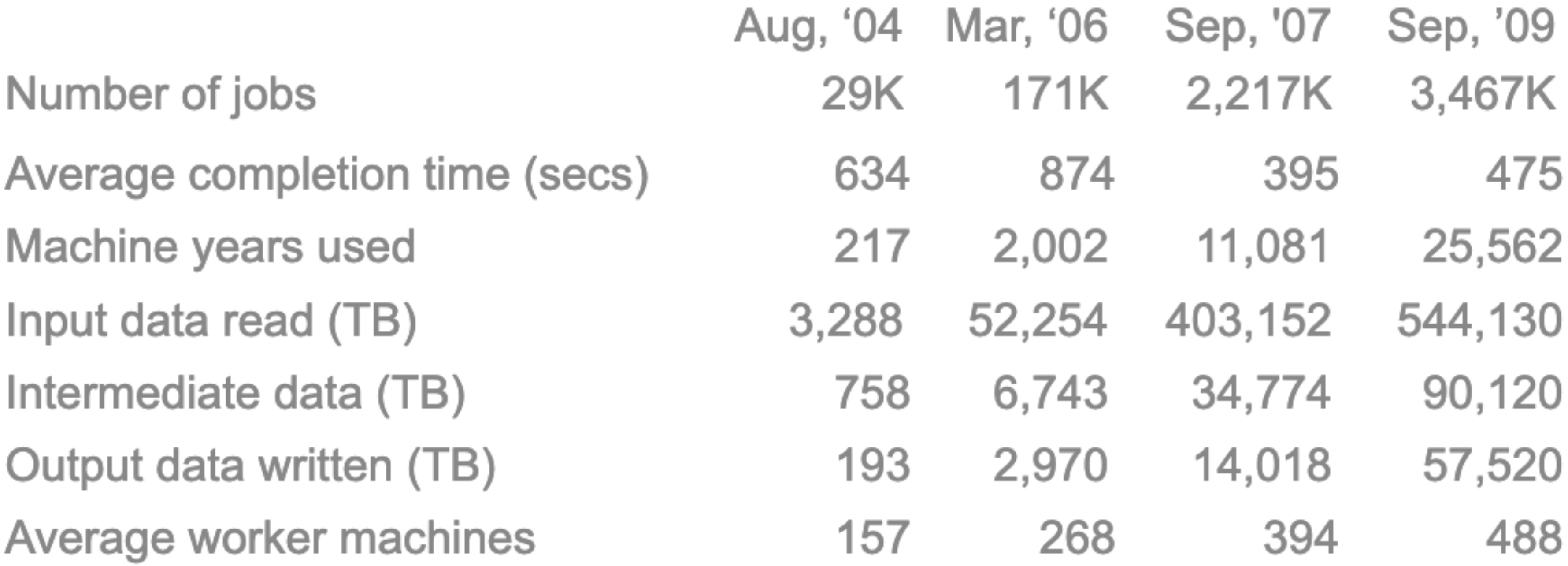
- Task granularity

GFS usage at Google

- 200+ clusters
- Many clusters of 1000s of machines
- Pools of 1000s of clients
- 4+ PB filesystems
- 40 GB/s read/write load
  - In the presence of frequent hardware failures

* Jeff Dean, LADIS 2009

MapReduce usage statistics over time



	Aug, ‘04	Mar, ‘06	Sep, '07	Sep, ’09
Number of jobs	29K	171K	2,217K	3,467K
Average completion time (secs)	634	874	395	475
Machine years used	217	2,002	11,081	25,562
Input data read (TB)	3,288	52,254	403,152	544,130
Intermediate data (TB)	758	6,743	34,774	90,120
Output data written (TB)	193	2,970	14,018	57,520
Average worker machines	157	268	394	488

* Jeff Dean, LADIS 2009

MapReduce discussion

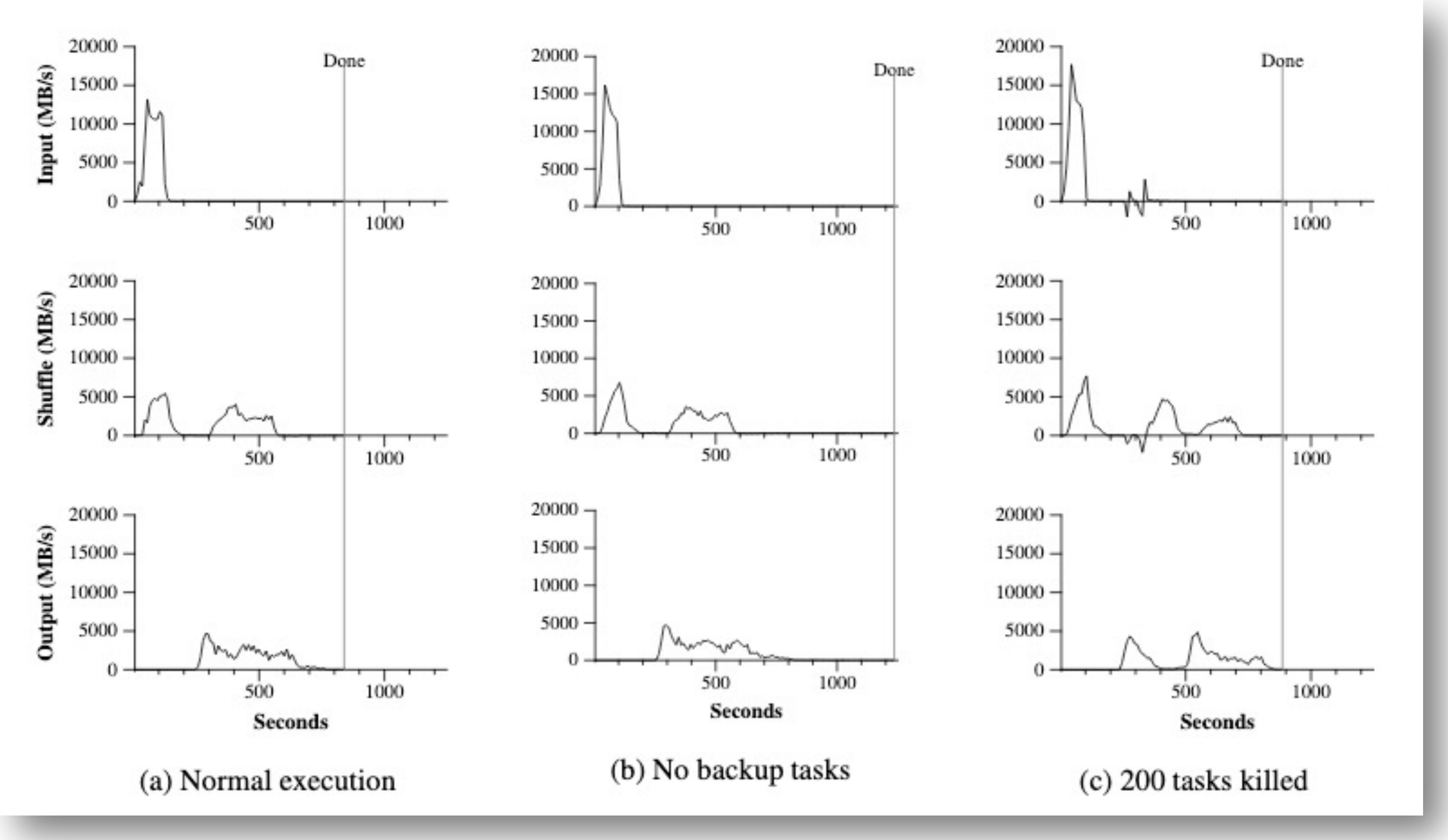
What will likely serve as a performance bottleneck for Google's MapReduce used back in 2004 (or even earlier)? CPU? Memory? Disk? Network? Anything else?

MapReduce discussion

What will likely serve as a performance bottleneck for Google's MapReduce used back in 2004 (or even earlier)? CPU? Memory? Disk? Network? Anything else?

How does MapReduce reduce the effect of slow network?

MapReduce discussion



(a) Normal execution

(b) No backup tasks

(c) 200 tasks killed

MapReduce discussion

Consider a log analytics job where you perform log-based debugging. You want to extract the timestamp info of all entries that match a keyword and then calculate the count of all matched entries:

1. Filter the entries with the keyword;
2. Calculate the count of all matched entries

What are the main shortcomings of using MapReduce to support such pipeline-like applications?

Next step

- Look out for
    - Project suggestion doc
    - Fill the team composition form
    - Project bid and team composition due by Feb 24

- Next week: Apache Spark