Map-Reduce; Hadoop; Apache Spark

Amol Deshpande
CMSC424

Spring 2020 – Online Instruction Plan

- Week 1: File Organization and Indexes
- Week 2: Query Processing
- Week 3: Query Optimization; Parallel Databases 1
- Week 4: Parallel Databases; Mapreduce; Transactions 1
  - ★ Map-reduce and Apache Spark (Posted early for Project 5)
  - ★ Parallel Databases 2: Execution and Other Issues
  - ★ Transactions 1
  - ★ Homework Due April 24
- Week 5: Transactions 2 (Homework Due May 1)
- Week 6: Miscellaneous Topics (Homework Due May 8)

”Big Data”

- Very large volumes of data being collected
  - Driven by growth of web, social media, and more recently internet-of-things
  - Web logs were an early source of data
    - Analytics on web logs has great value for advertisements, web site structuring, what posts to show to a user, etc

- Big Data: differentiated from data handled by earlier generation databases
  - **Volume**: much larger amounts of data stored
  - **Velocity**: much higher rates of insertions
  - **Variety**: many types of data, beyond relational data

The FOUR V's of Big Data

From traffic patterns and music downloads to web history and medical records, data is recorded, stored, and analyzed to enable the technology and services that the world relies on every day. But what exactly is big data, and how can these massive amounts of data be used?

As a leader in the sector, IBM data scientists break big data into four dimensions: **Volume, Velocity, Variety and Veracity**

Depending on the industry and organization, big data encompasses information from multiple internal and external sources such as transactions, social media, enterprise content, sensors and mobile devices. Companies can leverage data to adapt their products and services to better meet customer needs, optimize operations and infrastructure, and find new sources of revenue.

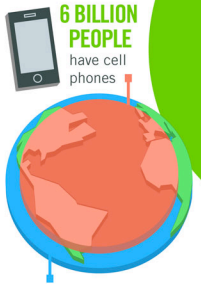
By 2015
4.4 MILLION IT JOBS
will be created globally to support big data, with 1.9 million in the United States

Volume
SCALE OF DATA

40 ZETTABYTES
[43 TRILLION GIGABYTES]
of data will be created by 2020, an increase of 300 times from 2005

It's estimated that
2.5 QUINTILLION BYTES
[2.3 TRILLION GIGABYTES]
of data are created each day

6 BILLION PEOPLE
have cell phones



WORLD POPULATION: 7 BILLION

Most companies in the U.S. have at least
100 TERABYTES
[100,000 GIGABYTES]
of data stored

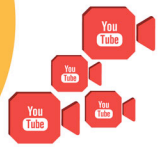
Variety
DIFFERENT FORMS OF DATA

As of 2011, the global size of data in healthcare was estimated to be
150 EXABYTES
[161 BILLION GIGABYTES]

30 BILLION PIECES OF CONTENT
are shared on Facebook every month

By 2014, it's anticipated there will be
420 MILLION WEARABLE, WIRELESS HEALTH MONITORS

4 BILLION+ HOURS OF VIDEO
are watched on YouTube each month



400 MILLION TWEETS
are sent per day by about 200 million monthly active users

Velocity
ANALYSIS OF STREAMING DATA

The New York Stock Exchange captures
1 TB OF TRADE INFORMATION
during each trading session

Modern cars have close to
100 SENSORS
that monitor items such as fuel level and tire pressure

By 2016, it is projected there will be
18.9 BILLION NETWORK CONNECTIONS
– almost 2.5 connections per person on earth

Veracity
UNCERTAINTY OF DATA

1 IN 3 BUSINESS LEADERS
don't trust the information they use to make decisions

in one survey were unsure of how much of their data was inaccurate

Poor data quality costs the US economy around
$3.1 TRILLION A YEAR

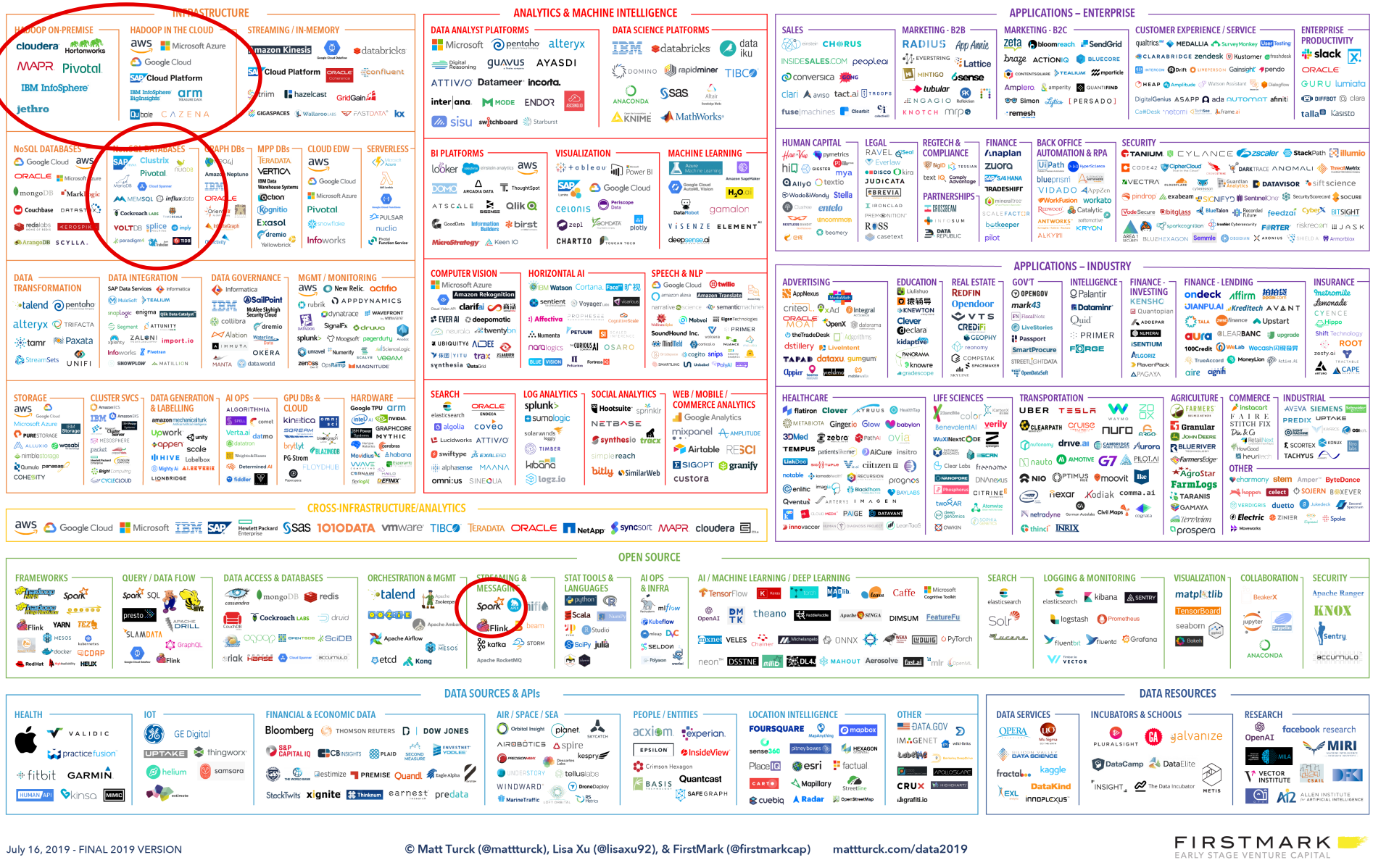
Sources: McKinsey Global Institute, Twitter, Cisco, Gartner, EMC, SAS, IBM, MEPTEC, QAS

IBM

Why (Parallel) Databases Don't Work

- The data is often not relational in nature
  - E.g., images, text, graphs
- The analysis/queries are not relational in nature
  - E.g., Image Analysis, Text Analytics, Natural Language Processing, Web Analytics, Social Network Analysis, Machine Learning, etc.
  - Databases don't really have constructs to support this
    - User-defined functions can help to some extent
  - Need to interleave relational-like operations with non-relational (e.g., data cleaning, etc.)
  - Domain users are more used to procedural languages
- The operations are often one-time
  - Only need to analyse images once in a while to create a "deep learning" model
  - Databases are really better suited for repeated analysis of the data
- Much of the analysis not time-sensitive
- Parallel databases too expensive given the data volumes
  - Were designed for large enterprises, with typically big budgets

DATA & AI LANDSCAPE 2019



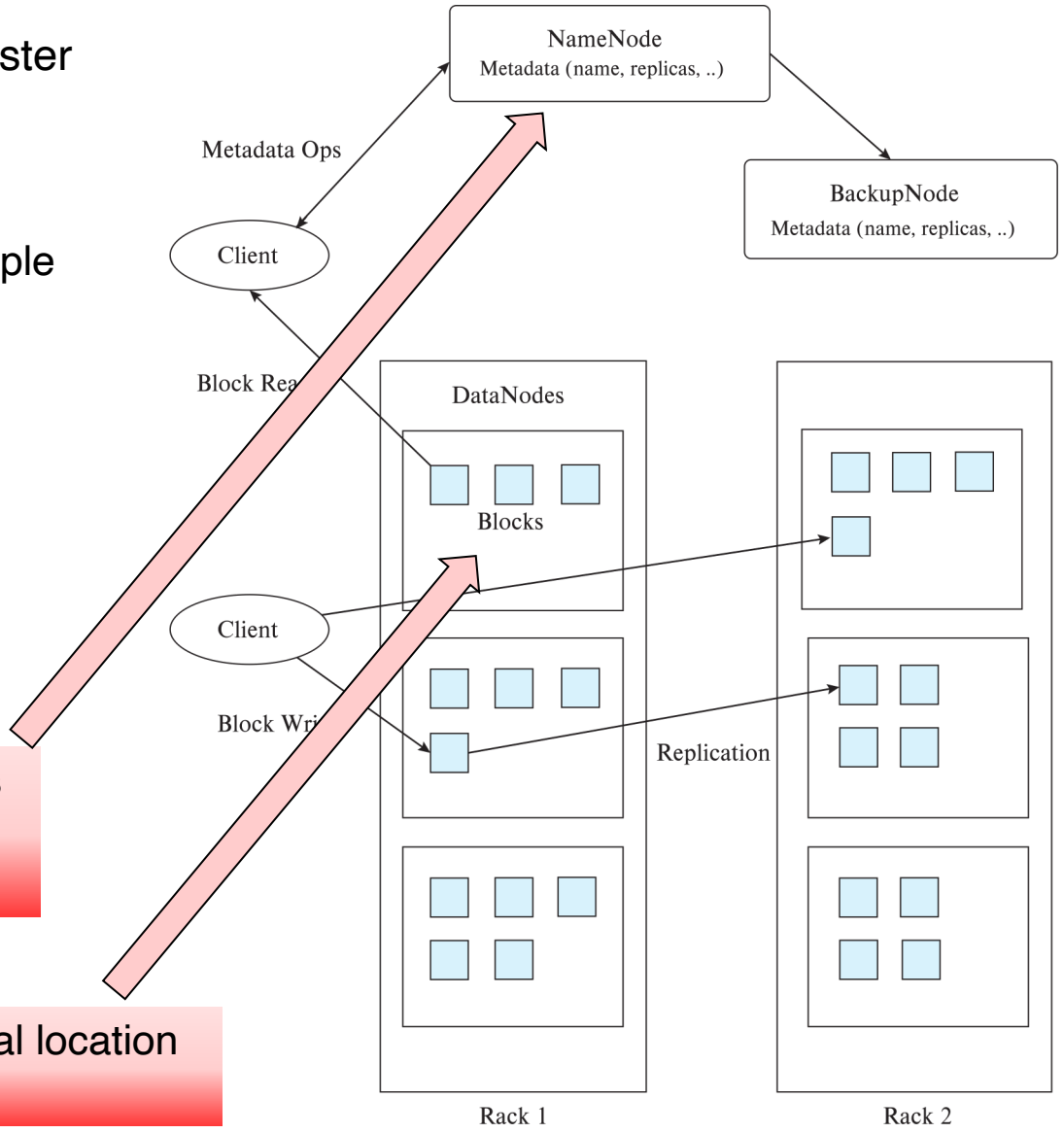
FIRSTMARK
EARLY STAGE VENTURE CAPITAL

Distributed File Systems

- A distributed file system stores data across a large collection of machines, but provides single file-system view
- Highly scalable distributed file system for large data-intensive applications.
  - E.g., 10K nodes, 100 million files, 10 PB
- Provides redundant storage of massive amounts of data on cheap and unreliable computers
  - Files are replicated to handle hardware failure
  - Detect failures and recovers from them
- Examples:
  - Google File System (GFS)
  - Hadoop File System (HDFS)

Hadoop File System Architecture

- Single Namespace for entire cluster
- Files are broken up into blocks
  - Typically 64 MB block size
  - Each block replicated on multiple DataNodes
- Client
  - Finds location of blocks from NameNode
  - Accesses data directly from DataNode



- Maps a filename to list of Block IDs
- Maps each Block ID to DataNodes containing a replica of the block

Maps a Block ID to a physical location on disk

Key-Value Storage Systems

- Unlike HDFS, focus here on storing large numbers (billions or even more) of small (KB-MB) sized records
  - **uninterpreted bytes**, with an associated key
    - E.g., Amazon S3, Amazon Dynamo
  - **Wide-table** (can have arbitrarily many attribute names) with associated key
    - Google BigTable, Apache Cassandra, Apache Hbase, Amazon DynamoDB
    - Allows some operations (e.g., filtering) to execute on storage node
  - JSON
    - MongoDB, CouchDB (document model)
- Records **partitioned** across multiple machines
  - Queries are routed by the system to appropriate machine
- Records **replicated** across multiple machines for fault tolerance as well as efficient querying
  - Need to guarantee "consistency" when data is updated
  - **"Distributed Transactions"**

Key-Value Storage Systems

- Key-value stores support
  - ***put***(key, value): used to store values with an associated key,
  - ***get***(key): which retrieves the stored value associated with the specified key
  - ***delete***(key) -- Remove the key and its associated value
- Some support ***range queries*** on key values
- Document stores support richer queries (e.g., MongoDB)
  - Slowly evolving towards the richness of SQL
- Not full database systems (increasingly changing)
  - Have no/limited support for transactional updates
  - Applications must manage query processing on their own
- Not supporting above features makes it easier to build scalable data storage systems
  - Also called **NoSQL** systems

Replication and Consistency

- **Availability** (system can run even if parts have failed)
  - Typically via replication
- **Consistency**
  - All live replicas have same value, and each read sees latest version
  - Often implemented using majority protocols
    - E.g., have 3 replicas, reads/writes must access 2 replicas
- **Network partitions** (network can break into two or more parts, each with active systems that can't talk to other parts)

- Distributed systems will "partition" at some point – must choose consistency or availability
  - Brewer's CAP "Theorem"
  - Traditional database choose consistency
  - Most Web applications choose availability

The MapReduce Paradigm

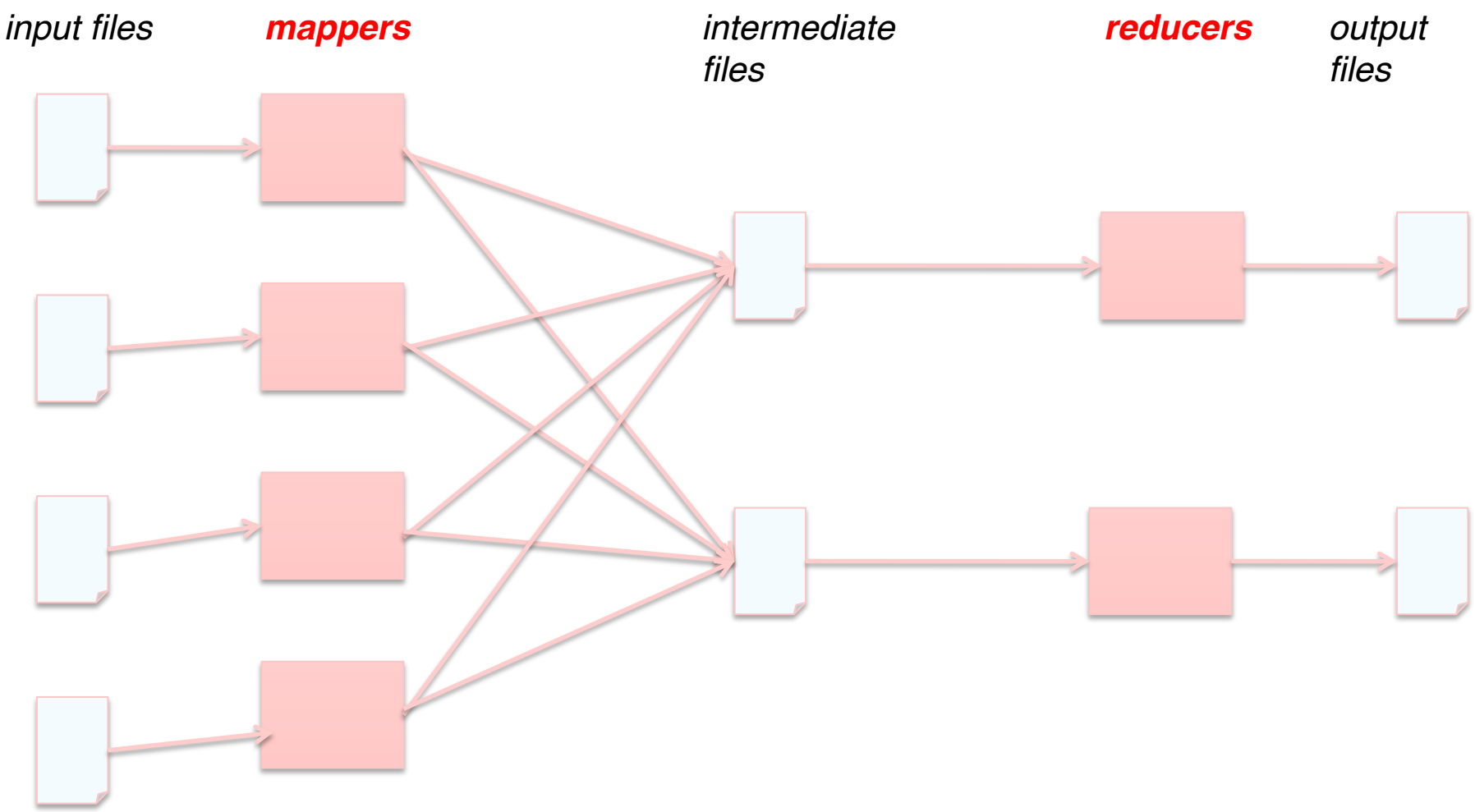
- Platform for reliable, scalable parallel computing
- Abstracts issues of distributed and parallel environment from programmer
  - Programmer provides core logic (via map() and reduce() functions)
  - System takes care of parallelization of computation, coordination, etc.
- Paradigm dates back many decades
  - But very large scale implementations running on clusters with 10^3 to 10^4 machines are more recent
  - Google Map Reduce, Hadoop, ..
- Data storage/access typically done using distributed file systems or key-value stores

MapReduce Framework

- Provides a fairly restricted, but still powerful abstraction for programming

- Programmers write a pipeline of functions, called *map* or *reduce*
  - **map programs**
    - inputs: a list of “records” (record defined arbitrarily – could be images, genomes etc…)
    - output: for each record, produce a set of “(key, value)” pairs
  - **reduce programs**
    - input: a list of “(key, {values})” grouped together from the mapper
    - output: whatever
  - Both can do arbitrary computations on the input data as long as the basic structure is followed

MapReduce Framework

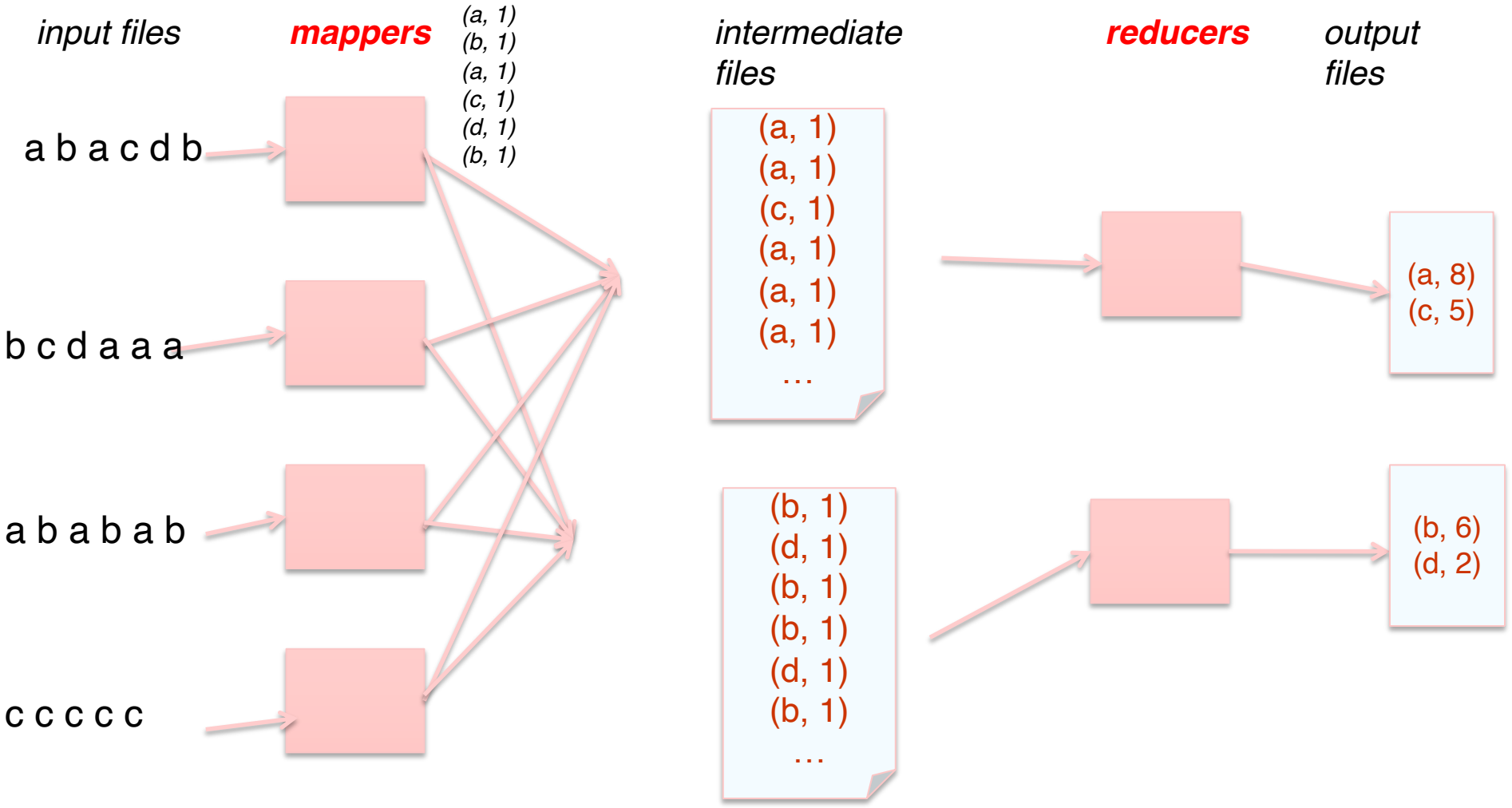


Word Count Example

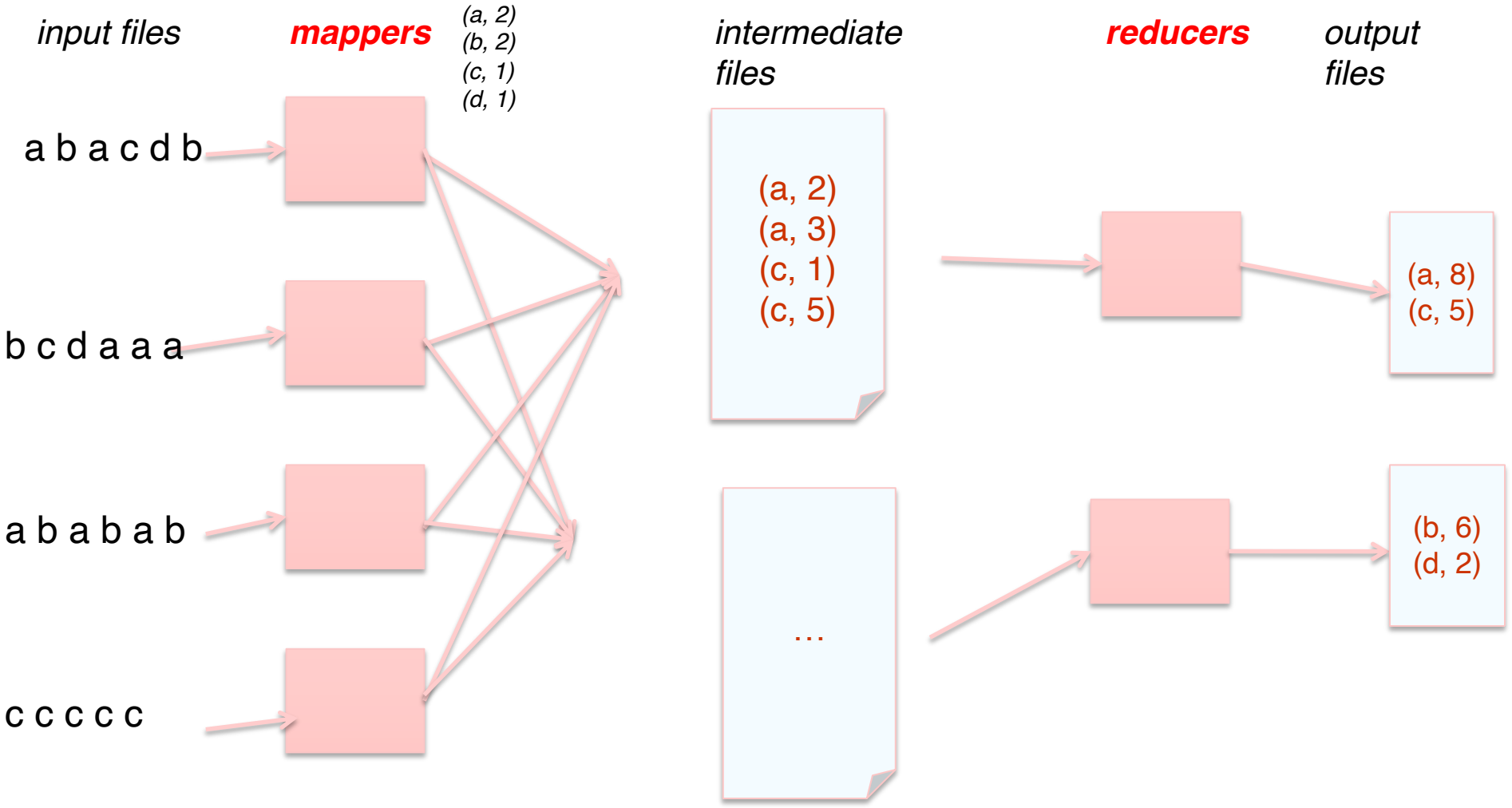
```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

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));
```

MapReduce Framework: Word Count



More Efficient Word Count



Called “mapper-side” combiner

Hadoop MapReduce

- Google pioneered original map-reduce implementation
  - For building web indexes, text analysis, PageRank, etc.
- Hadoop -- widely used open source implementation in Java
- Huge ecosystem built around Hadoop now, including HDFS, consistency mechanisms, connectors to different systems (e.g., key-value stores, databases), etc.
- Apache Spark a newer implementation of Map-Reduce
  - More user-friendly syntax
  - Significantly faster because of in-memory processing
  - SQL-like in many ways (“DataFrames”)

Spark

- **Resilient Distributed Dataset** (**RDD**) abstraction
  - Collection of records that can be stored across multiple machines
- RDDs can be created by applying algebraic operations on other RDDs
  - Or from loading data from HDFS, key-value stores, etc.
- RDDs can be lazily computed when needed
- "DataFrames" is an abstraction built on top of RDDs
  - Not unlike "relations"
  - Supports relational operations like Joins, Aggregates, etc.
- Incorporates "Query Optimization" today as well

Spark

- Walk through Spark Programming Guides and the Jupyter Notebook

Summary

- Traditional databases don't provide the right abstractions for many newer data processing/analytics tasks
- Led to development of NoSQL systems and Map-Reduce (or similar) frameworks
  - Easier to get started
  - Easier to handle ad hoc and arbitrary tasks
  - Not as efficient
- Over the last 10 years, seen increasing convergence
  - NoSQL stores increasingly support SQL constructs like joins and aggregations
  - Map-reduce frameworks also evolved to support joins and SQL more explicitly
  - Databases evolved to support more data types, richer functionality for ad hoc processing
- Think of Map-Reduce systems as another option
  - Appropriate in some cases, not a good fit in other cases