Big Data Analytics

Map-Reduce and Spark

Slides by:

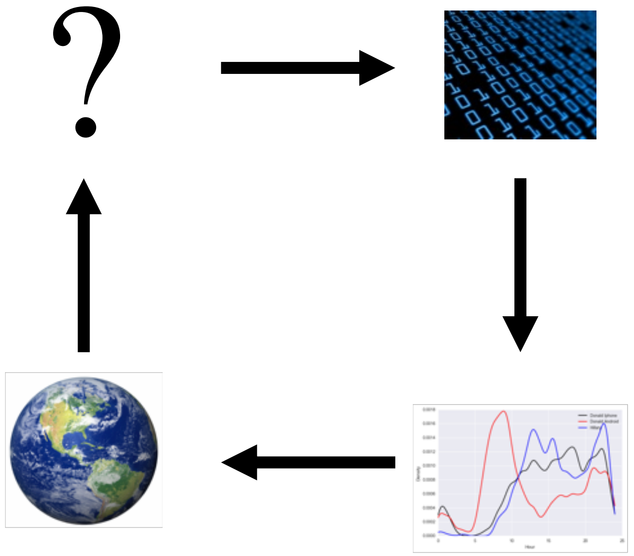
Joseph E. Gonzalez

jegonzal@cs.berkeley.edu

With revisions by:

Josh Hug

hug@cs.Berkeley.edu



From SQL to Big Data (with SQL)

- Last week…
  - Databases
  - (Relational) Database Management Systems
  - SQL: Structured Query Language
- Today
  - More on databases and database design
  - Enterprise data management and the data lake
  - Introduction to distributed data storage and processing
  - Spark

Operational Data Store

CUBE

Data Warehouse

ROLLUP

Drill Down

ETL (Extract, Transform, Load)

Snowflake Schema

Schema on Read

Data in the Organization

A little bit of buzzword bingo!

Star Schema

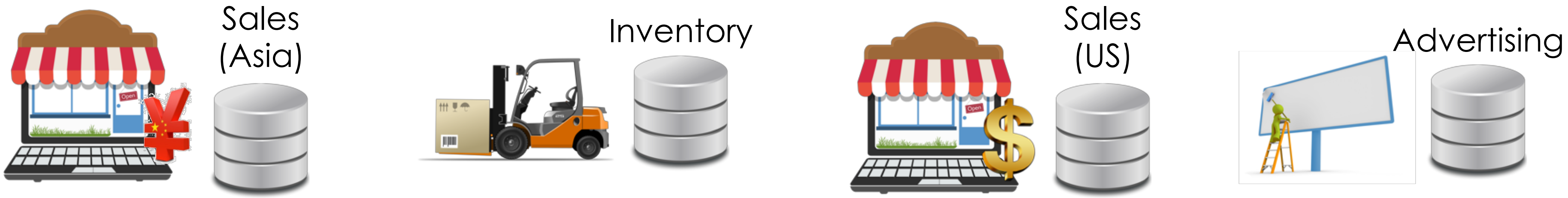
Data Lake

OLAP (Online Analytics Processing)

Inventory

How we like to think of data in the organization

The reality…

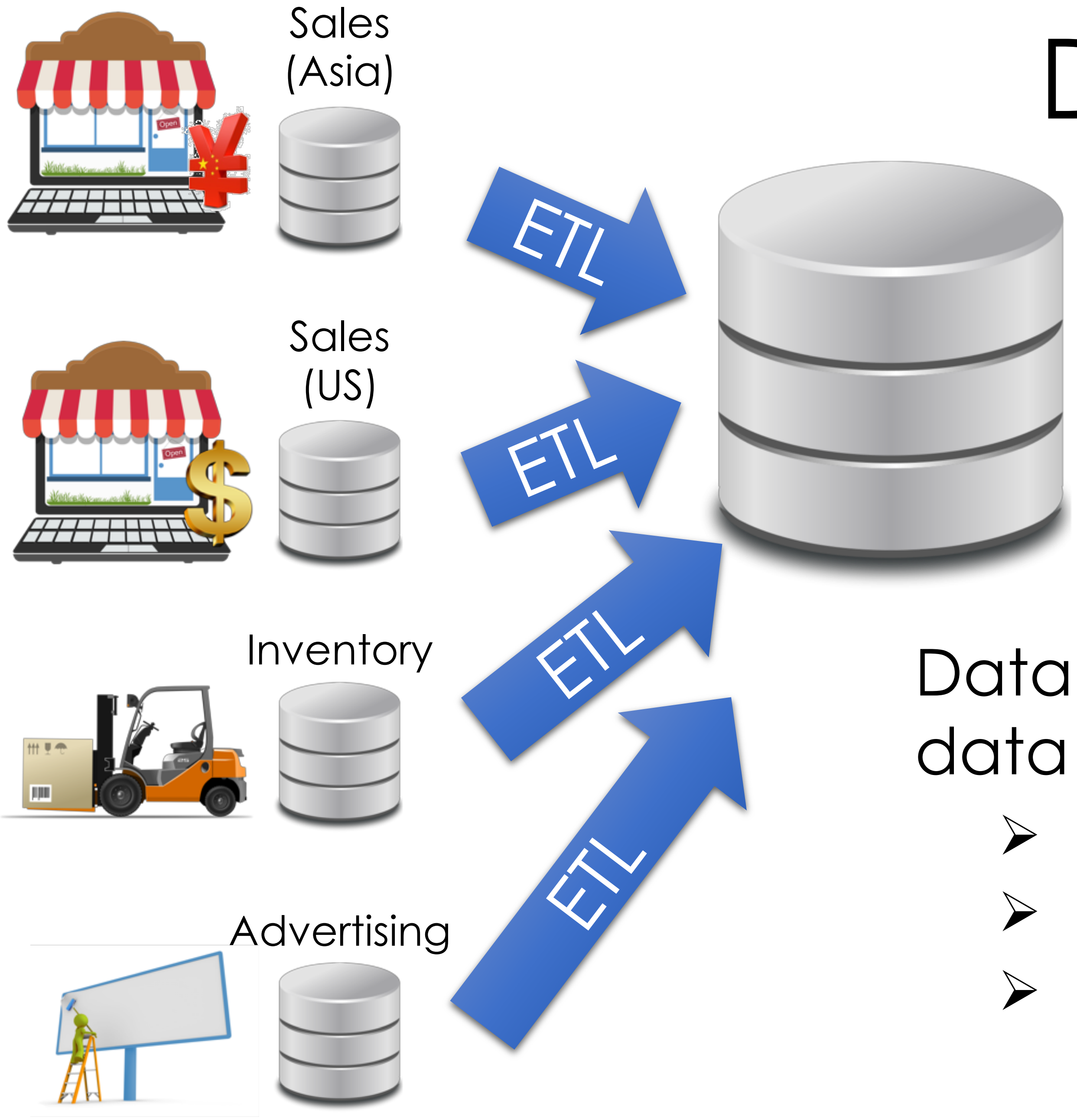


Operational Data Stores

- Capture **the now**
- Many different databases across an organization
- Mission critical… be careful!
  - Serving live ongoing business operations
  - Managing inventory
- Different formats (e.g., currency)
  - Different schemas (acquisitions ...)
- Live systems often don’t maintain history

We would like a consolidated, clean, historical snapshot of the data.

Data Warehouse



Collects and organizes historical data from multiple sources

Data is *periodically* **ETL**ed into the data warehouse:

- **Extracted** from remote sources
- **Transformed** to standard schemas
- **Loaded** into the (typically) relational (SQL) data system

<u>**E**</u>xtract → <u>**T**</u>ransform → <u>**L**</u>oad (ETL)

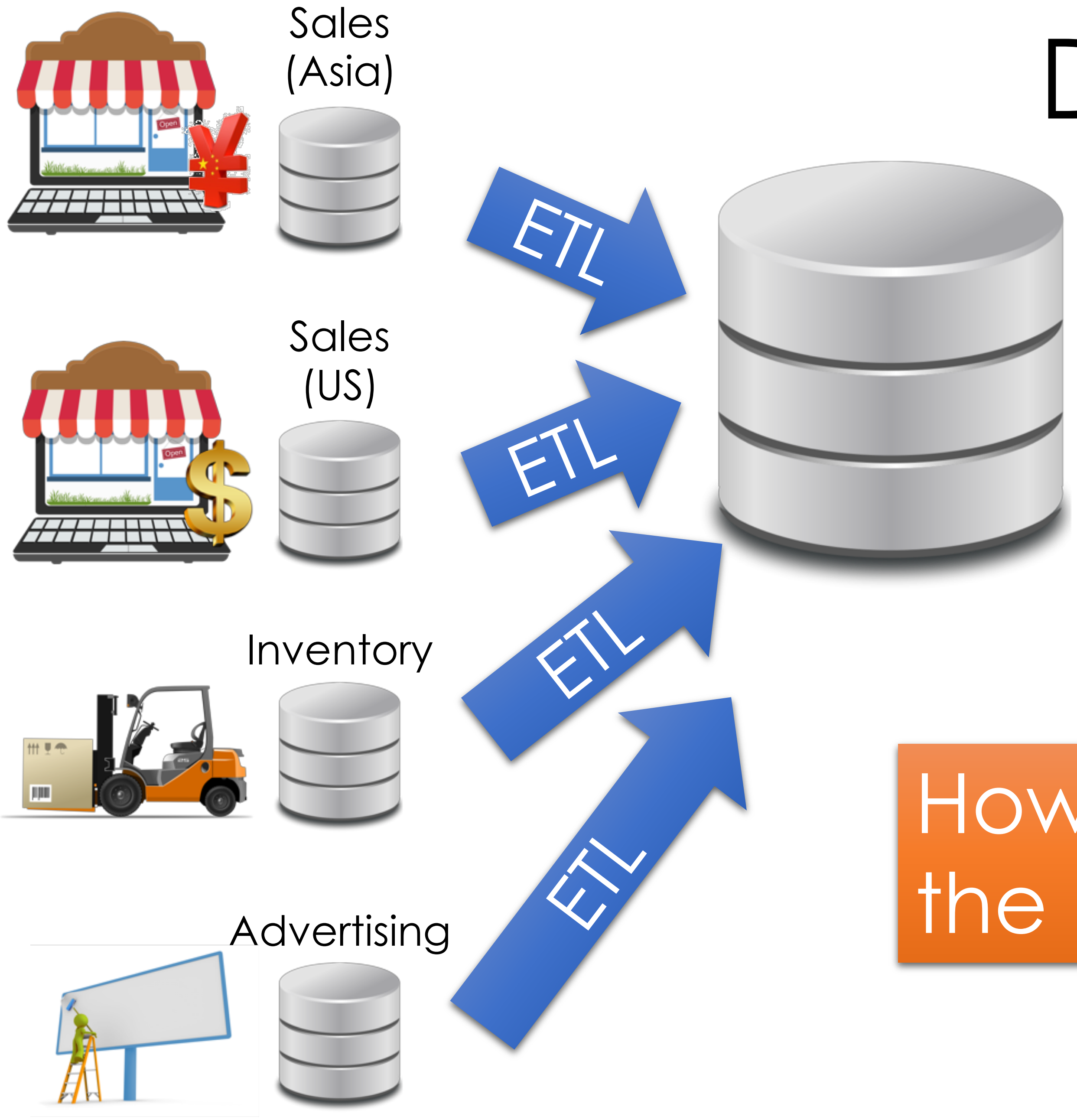
Extract & Load: provides a snapshot of operational data

- Historical snapshot
- Data in a single system
- Isolates analytics queries (e.g., Deep Learning) from business critical services (e.g., processing user purchase)
- Easy!

Transform: clean and prepare data for analytics in a unified representation

- **Difficult →** often requires specialized code and tools
- Different schemas, encodings, granularities

Data Warehouse



Collects and organizes historical data from multiple sources

How is data organized in the Data Warehouse?

Example Sales Data

pname	category	price	qty	date	day	city	state	country
Corn	Food	25	25	3/30/16	Wed.	Omaha	NE	USA
Corn	Food	25	8	3/31/16	Thu.	Omaha	NE	USA
Corn	Food	25	15	4/1/16	Fri.	Omaha	NE	USA
Galaxy	Phones	18	30	1/30/16	Wed.	Omaha	NE	USA
Galaxy	Phones	18	20	3/31/16	Thu.	Omaha	NE	USA
Galaxy	Phones	18	50	4/1/16	Fri.	Omaha	NE	USA
Galaxy	Phones	18	8	1/30/16	Wed.	Omaha	NE	USA
Peanuts	Food	2	45	3/31/16	Thu.	Seoul		Korea

- **Big** table: many *columns* and *rows*
  - Substantial redundancy → expensive to store and access
  - Make mistakes while updating
- Could we organize the data more efficiently?

Multidimensional Data Model

Sales **Fact Table**

pid	timeid	locid	sales
11	1	1	25
11	2	1	8
11	3	1	15
12	1	1	30
12	2	1	20
12	3	1	50
12	1	1	8
13	2	1	10
13	3	1	10
11	1	2	35
11	2	2	22
11	3	2	10
12	1	2	26

Locations

locid	city	state	country
1	Omaha	Nebraska	USA
2	Seoul		Korea
5	Richmond	Virginia	USA

Dimension Tables

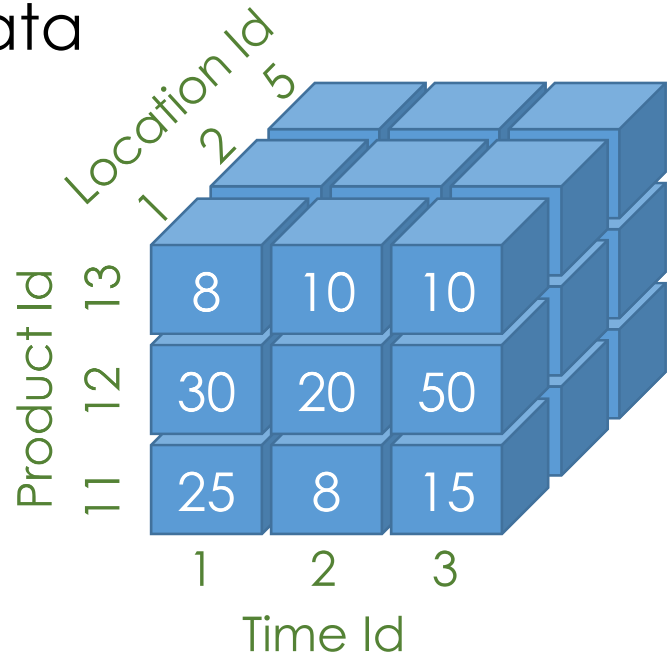
Products

pid	pname	category	price
11	Corn	Food	25
12	Galaxy 1	Phones	18
13	Peanuts	Food	2

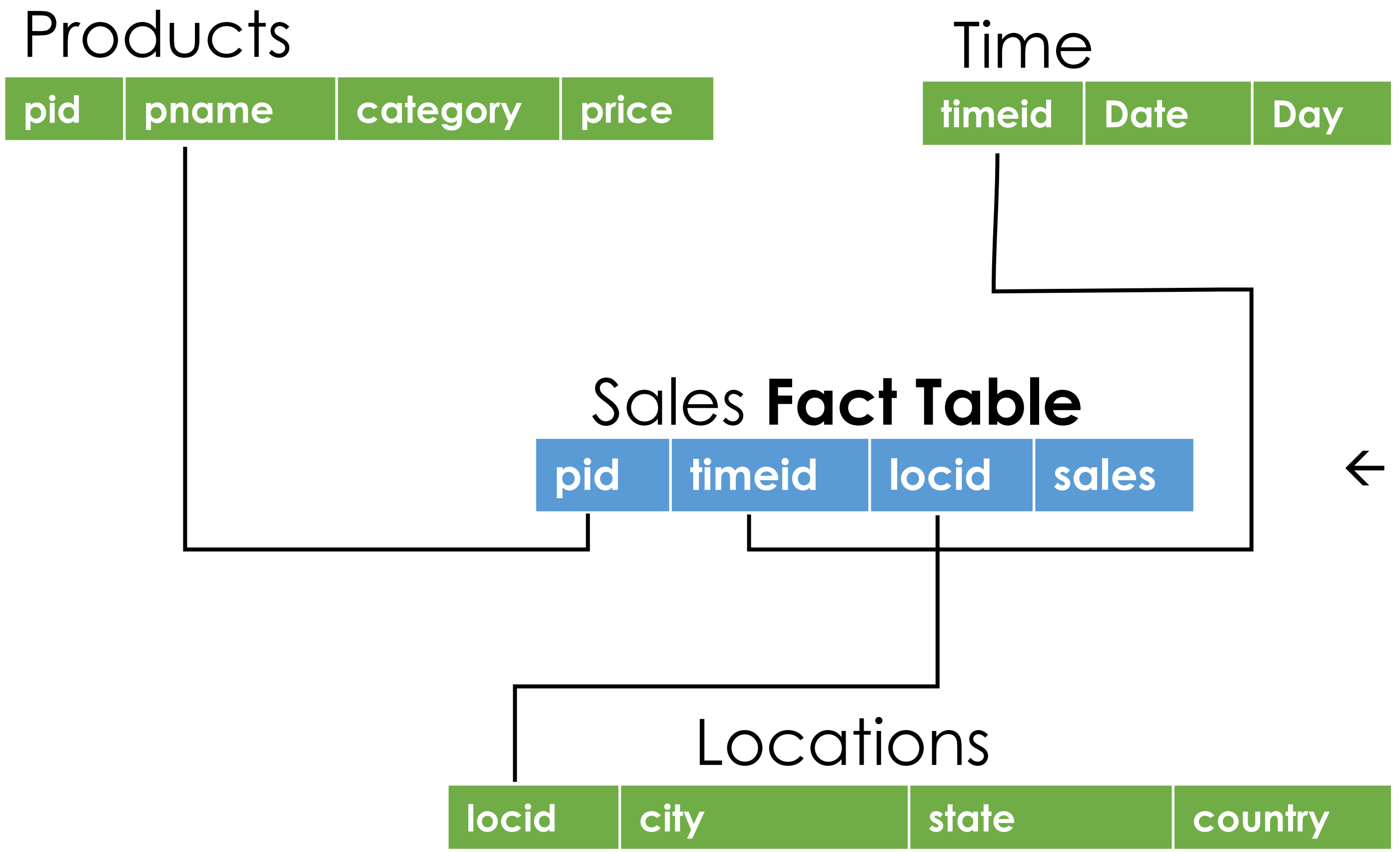
Time

timeid	Date	Day
1	3/30/16	Wed.
2	3/31/16	Thu.
3	4/1/16	Fri.

- Multidimensional "Cube" of data

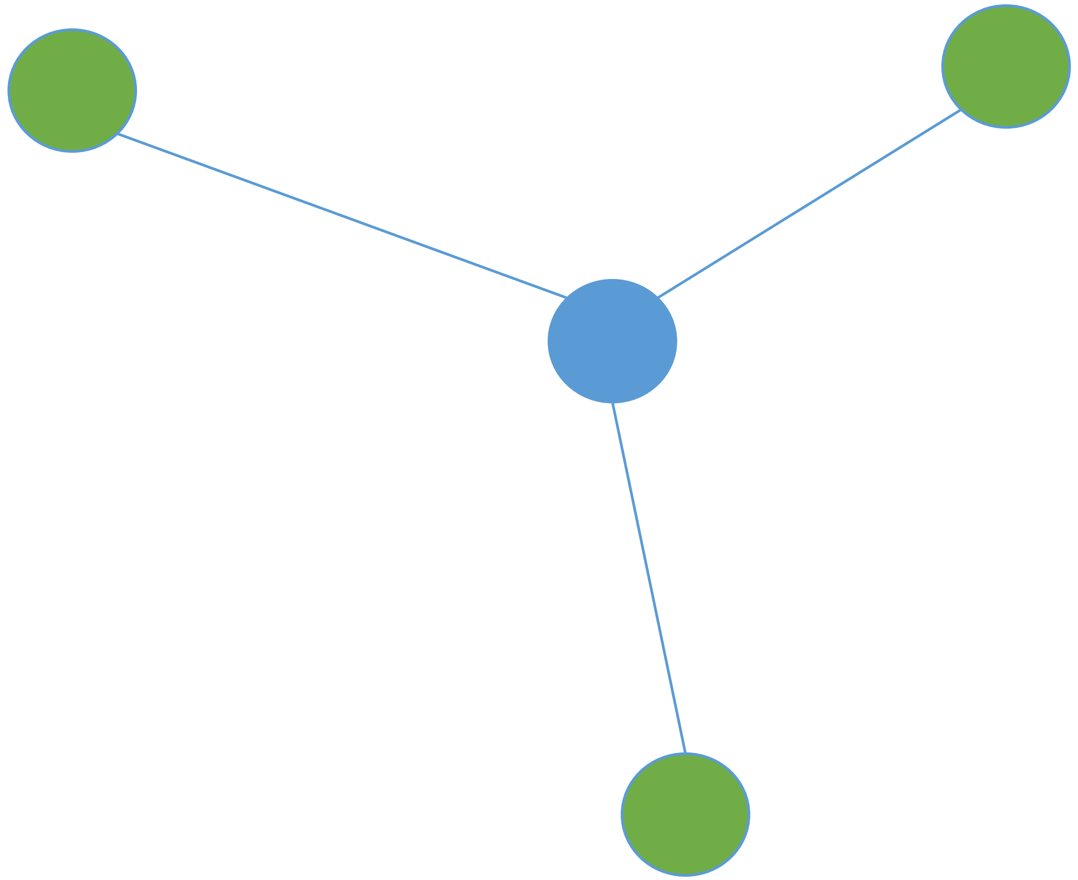


The Star Schema



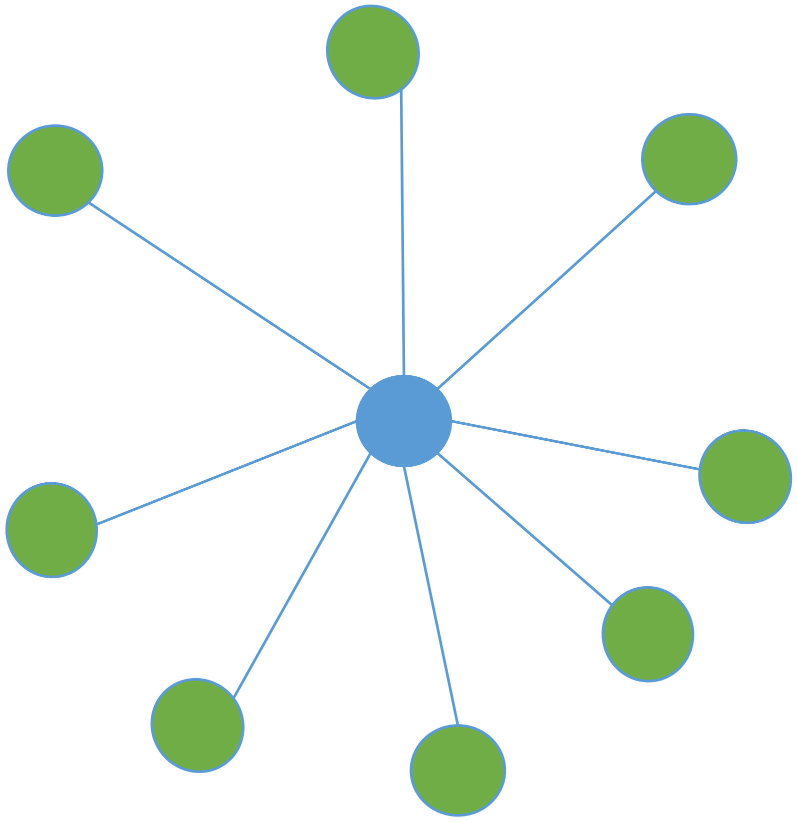
← This looks like a star …

The Star Schema



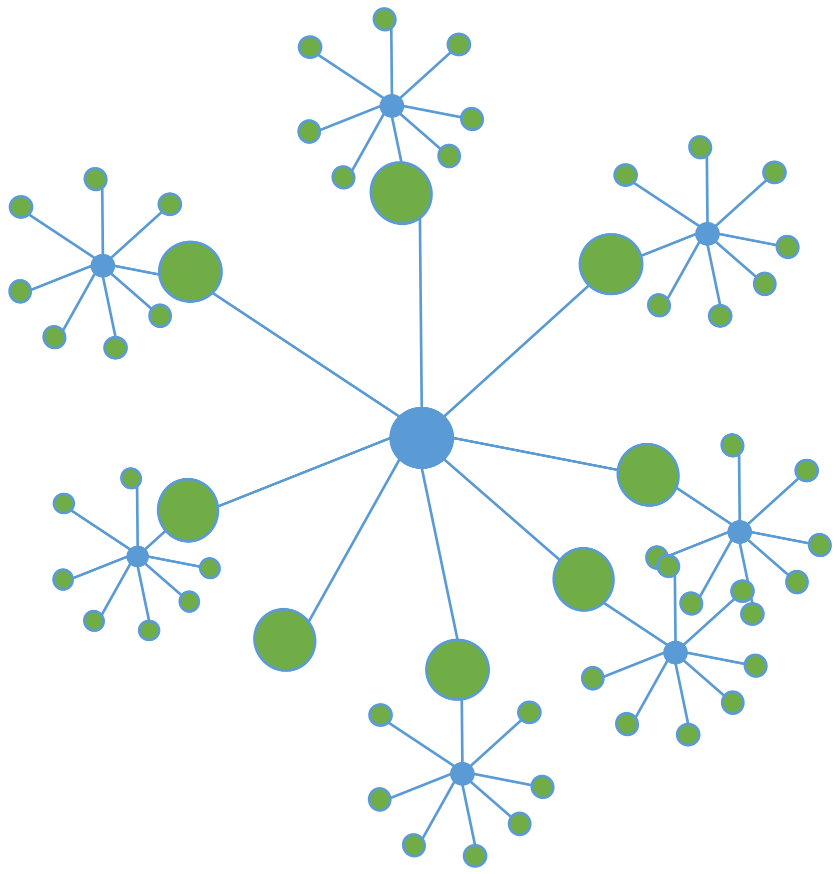
← This looks like a star …

The Star Schema



← This looks like a star …

The Snowflake Schema



← This looks like a snowflake …

See CS 186 for more!

Online Analytics Processing (**OLAP**)

Users interact with multidimensional data:

- Constructing ad-hoc and often complex SQL queries
- Using graphical tools that to construct queries
  - e.g. Tableau

Let's discuss some common types of queries used in OLAP.

Cross Tabulation (Pivot Tables)

Item	Color	Quantity
Desk	Blue	2
Desk	Red	3
Sofa	Blue	4
Sofa	Red	5

		Item		
		Desk	Sofa	***Sum***
Color	Blue	2	4	***6***
	Red	3	5	***8***
	Sum	***5***	***9***	***14***

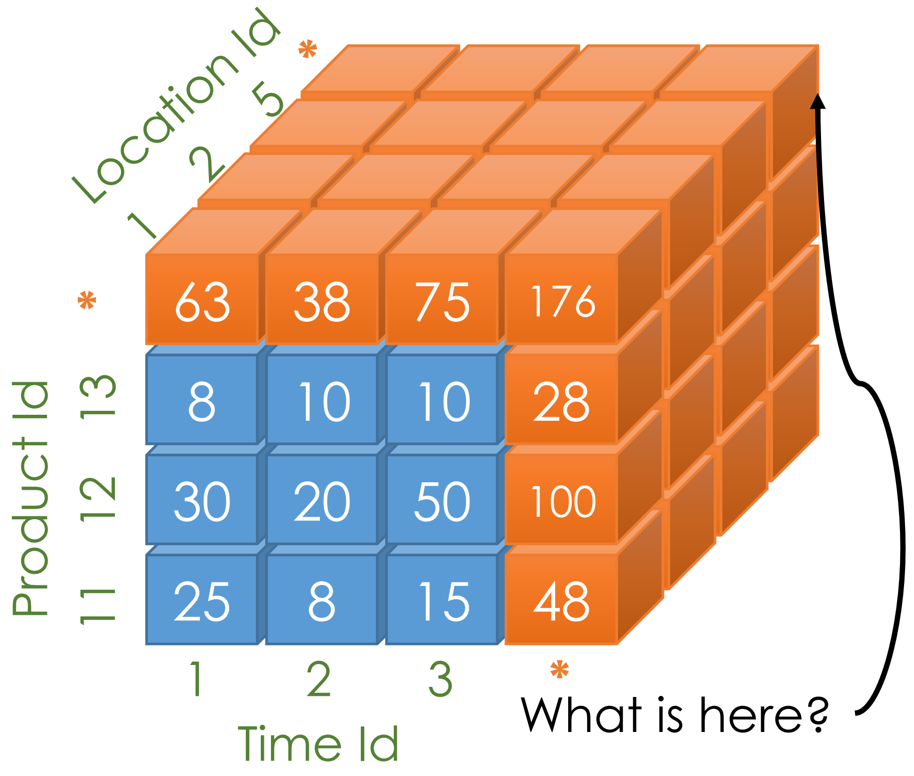
- Aggregate data across pairs of dimensions
  - **Pivot Tables:** *graphical interface* to select dimensions and aggregation function (e.g., SUM, MAX, MEAN)
  - **GROUP BY** queries
- Related to contingency tables and marginalization in stats.
- What about many dimensions?

Cube Operator

- ➢ Generalizes cross-tabulation to higher dimensions.

➢In SQL:

SELECT Item, Color, **SUM**(Quantity) **AS** QtySum
FROM Furniture
GROUP BY ***CUBE*** (Item, Color);

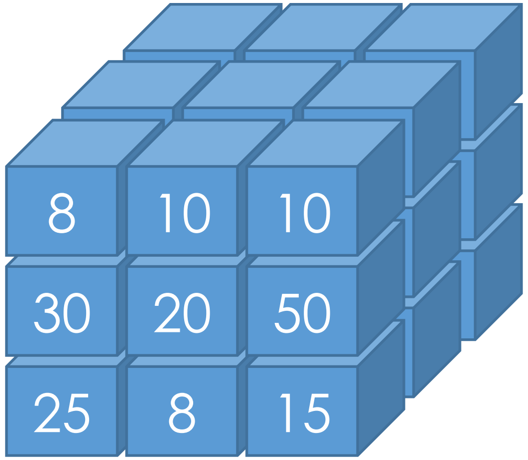


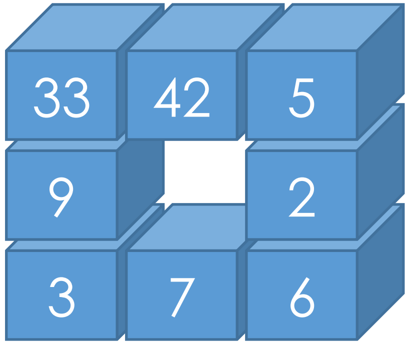
Item	Color	Quantity
Desk	Blue	2
Desk	Red	3
Sofa	Blue	4
Sofa	Red	5

Item	Color	QtySum
Desk	Blue	2
Desk	Red	3
Desk	*	5
Sofa	Blue	4
Sofa	Red	5
Sofa	*	9
*	*	14
*	Blue	6
*	Red	8

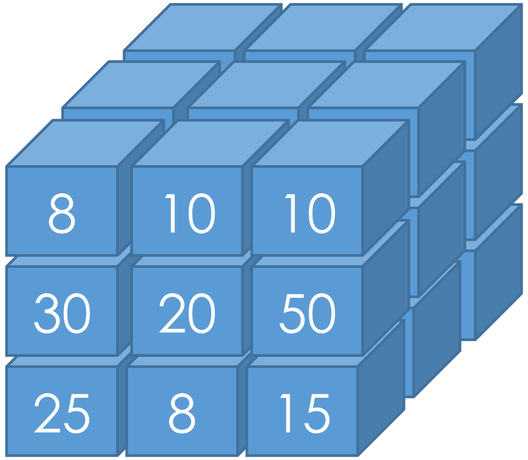
OLAP Queries

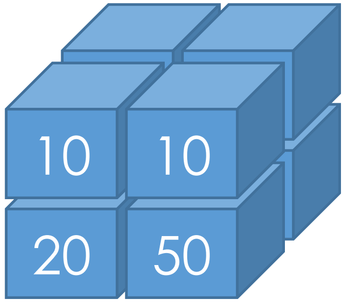
- **Slicing:** *selecting a value for a dimension*





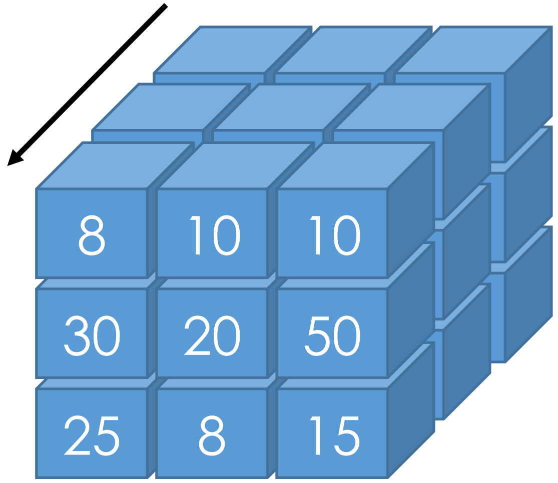
- **Dicing:** *selecting a range of values in multiple dimension*

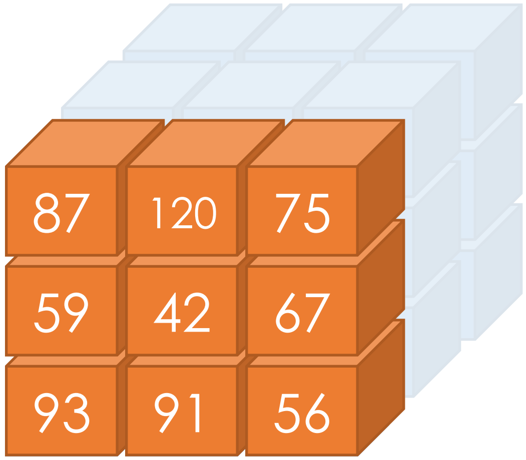




OLAP Queries

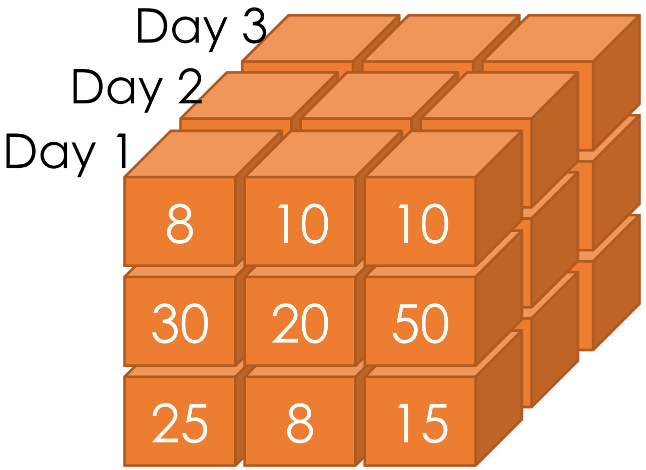
- **Rollup:** *Aggregating along a dimension*

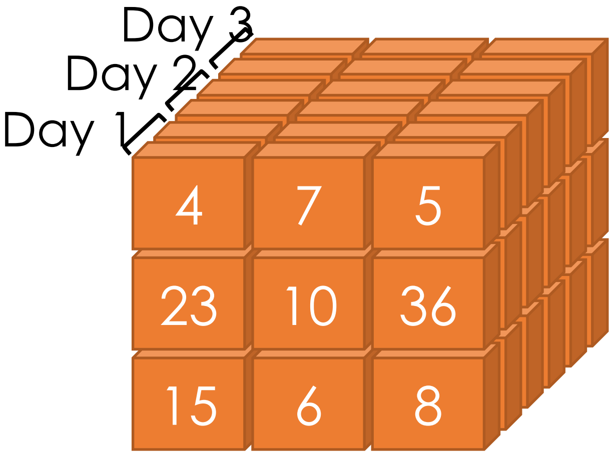




Similar to CUBE

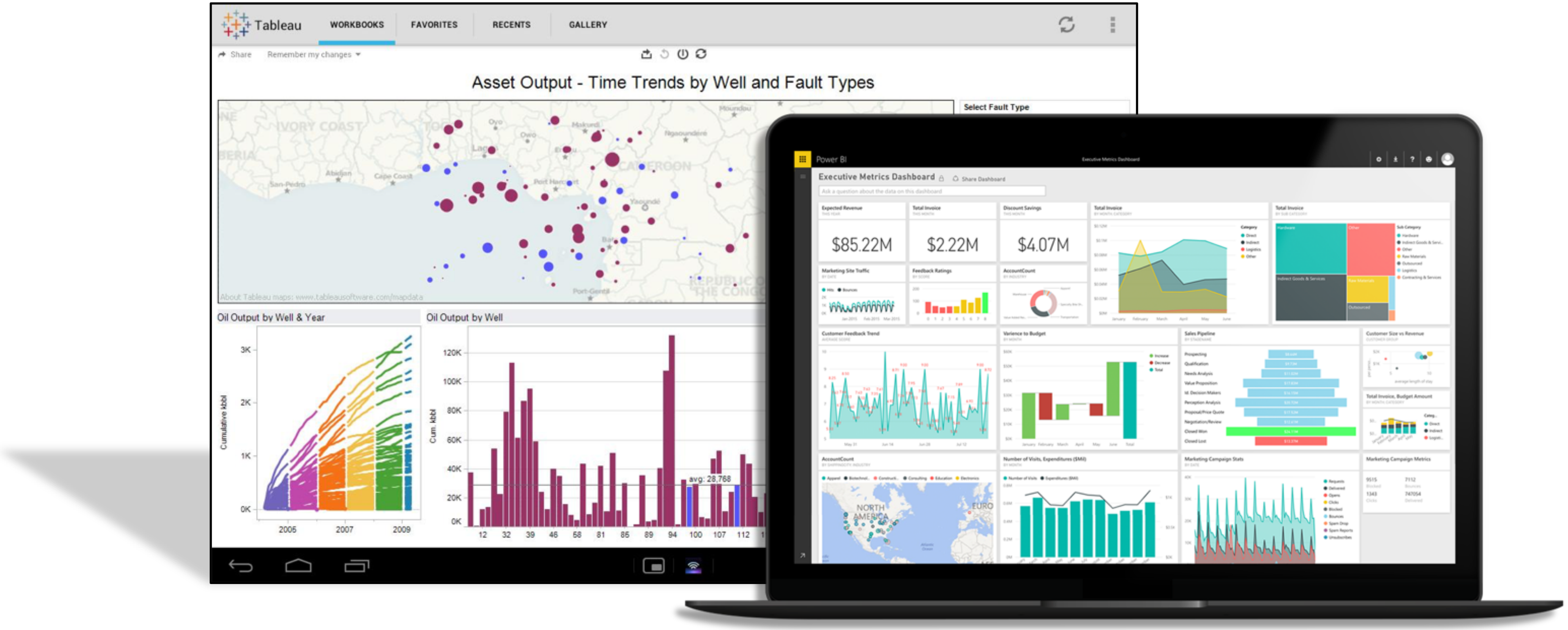
- **Drill-Down:** *de-aggregating along a dimension*



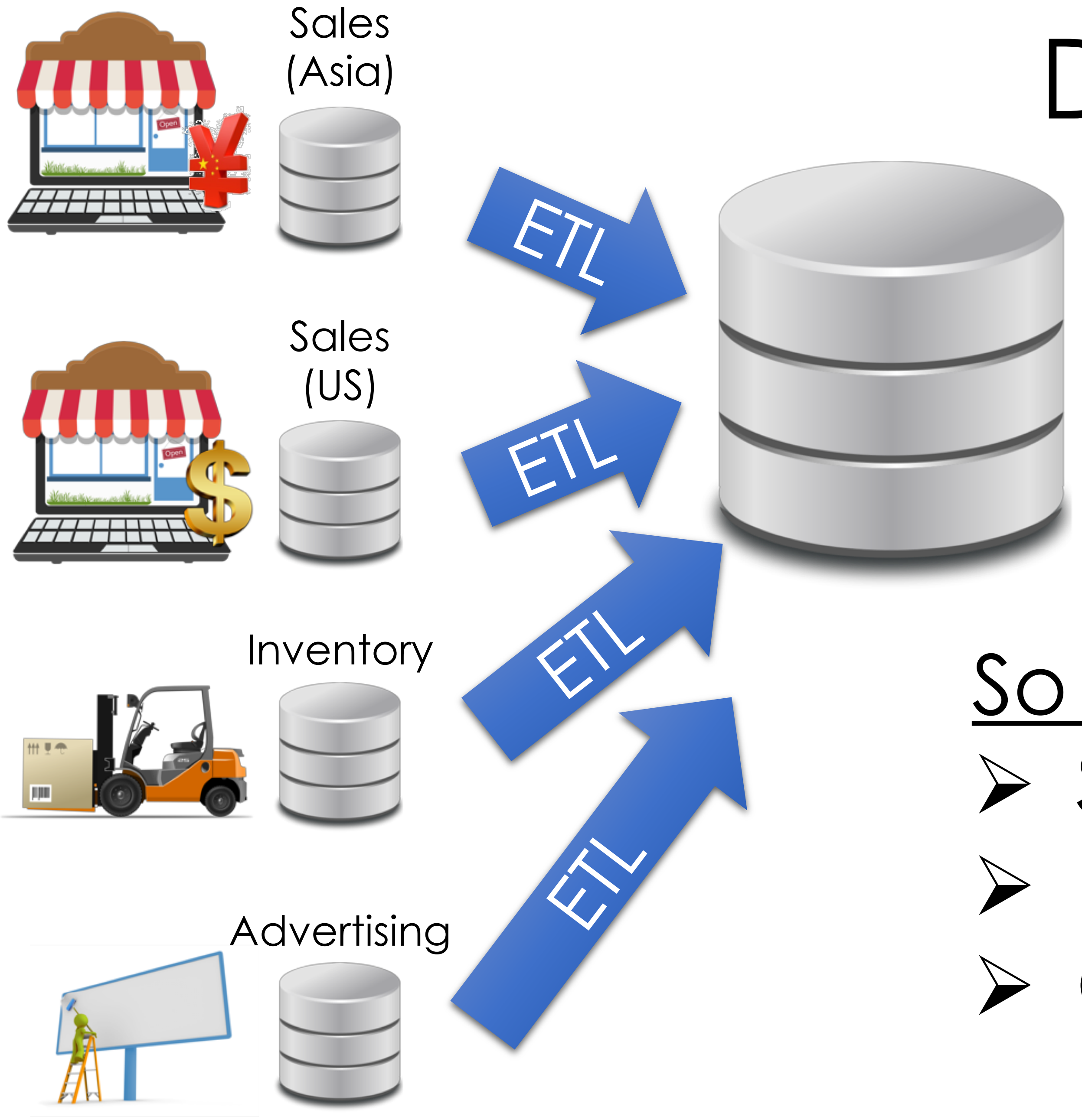


Reporting and Business Intelligence (BI)

- Use high-level tools to interact with their data:
  - Automatically generate SQL queries
    - Queries can get big!
- Common!



Data Warehouse



Collects and organizes historical data from multiple sources

<u>So far …</u>

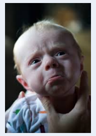
- Star Schemas
- Data cubes
- OLAP Queries

Data Warehouse

Collects and organizes historical data from multiple sources

- How do we deal with semi-structured and unstructured data?
- Do we really want to force a schema on load?

Data Warehouse

iid	date_taken	is_cat	is_grumpy	image_data
451231333	01-22-2016	1	1	
472342122	06-17-2017	0	1	
571827231	03-15-2009	0	0	
238472733	05-18-2018	0	0	

Unclear what a good schema for this image data might look like. Something like above will work, but it is inflexible!

Collects and organizes historical data from multiple sources

- How do we deal with semi-structured and unstructured data?
- Do we really want to force a schema on load?

Data Lake*

Store a copy of all the data

- in one place
- in its original "natural" form

Enable data consumers to choose how to transform and use data.

- *Schema on Read*

What could go wrong?

*Still being defined…[Buzzword Disclaimer]

The Dark Side of Data Lakes

- Cultural shift: *Curate* → *Save Everything!*
  - Noise begins to dominate signal
- Limited data governance and planning

  Example: *hdfs://important/joseph_big_file3.csv_with_json*
  - **What** does it contain?
  - **When** and **who** created it?
- No cleaning and verification → lots of dirty data
- New tools are more complex and old tools no longer work

Enter the data scientist

A Brighter Future for Data Lakes

Enter the data scientist

- Data scientists bring new skills
  - Distributed data processing and cleaning
  - Machine learning, computer vision, and statistical sampling
- Technologies are improving
  - SQL over large files
  - Self describing file formats (e.g. Parquet) & catalog managers
- Organizations are evolving
  - Tracking data usage and file permissions
  - New job title: data engineers

How do we **store** and **compute** on large unstructured datasets

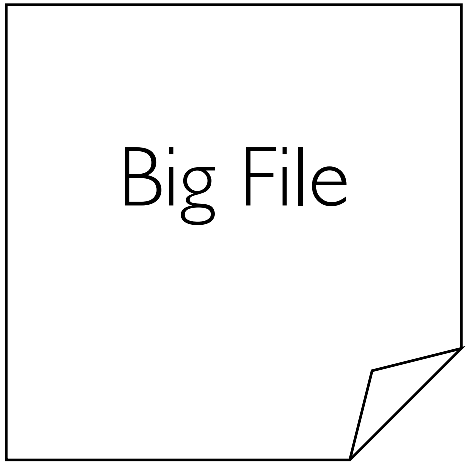
- Requirements:
  - Handle very **large files** spanning **multiple computers**
  - Use **cheap** commodity devices that **fail frequently**
  - **Distributed data processing** quickly and **easily**

- Solutions:
  - **Distributed file systems →** spread data over multiple machines
    - Assume machine **failure** is common → **redundancy**
  - **Distributed computing →** load and process files on multiple machines concurrently
    - Assume machine **failure** is common → **redundancy**
    - **Functional programming** computational pattern → **parallelism**

Distributed File Systems

Storing very large files

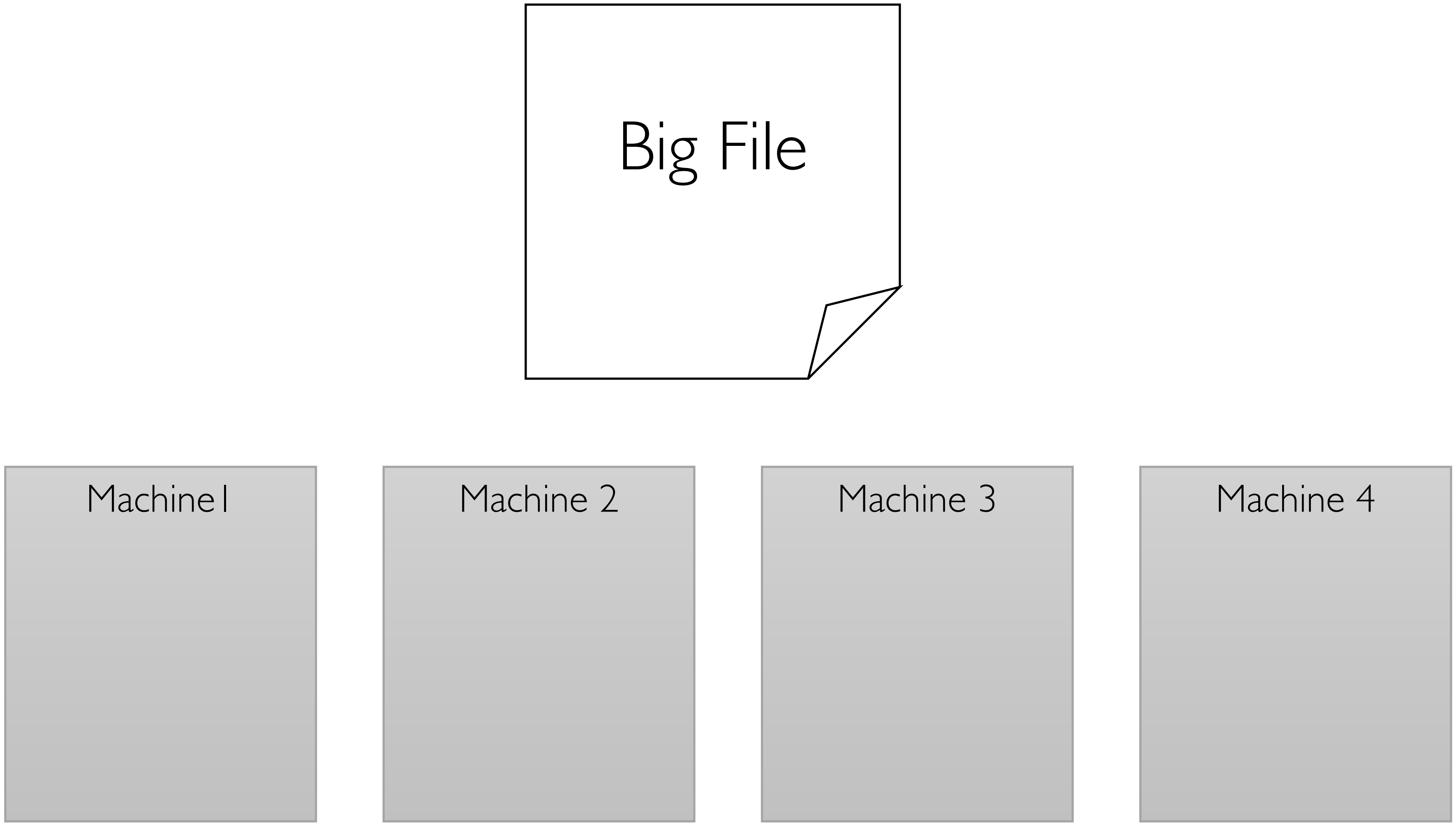
Fault Tolerant Distributed File Systems



How do we **store** and **access** very **large files** across **cheap** commodity devices ?

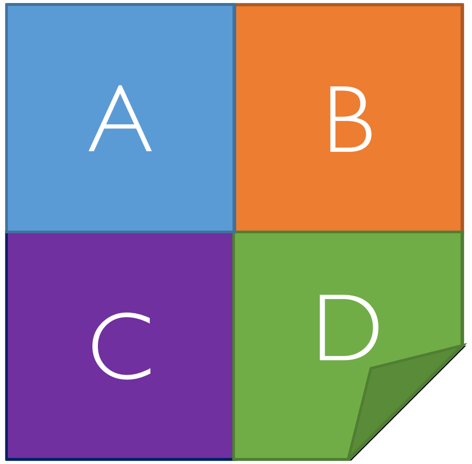
[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems

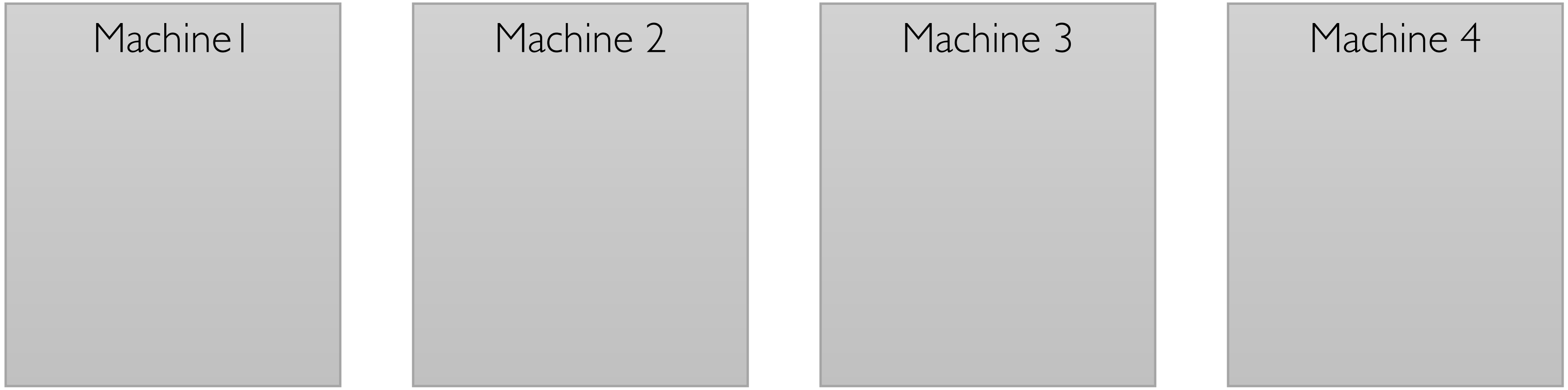


[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems

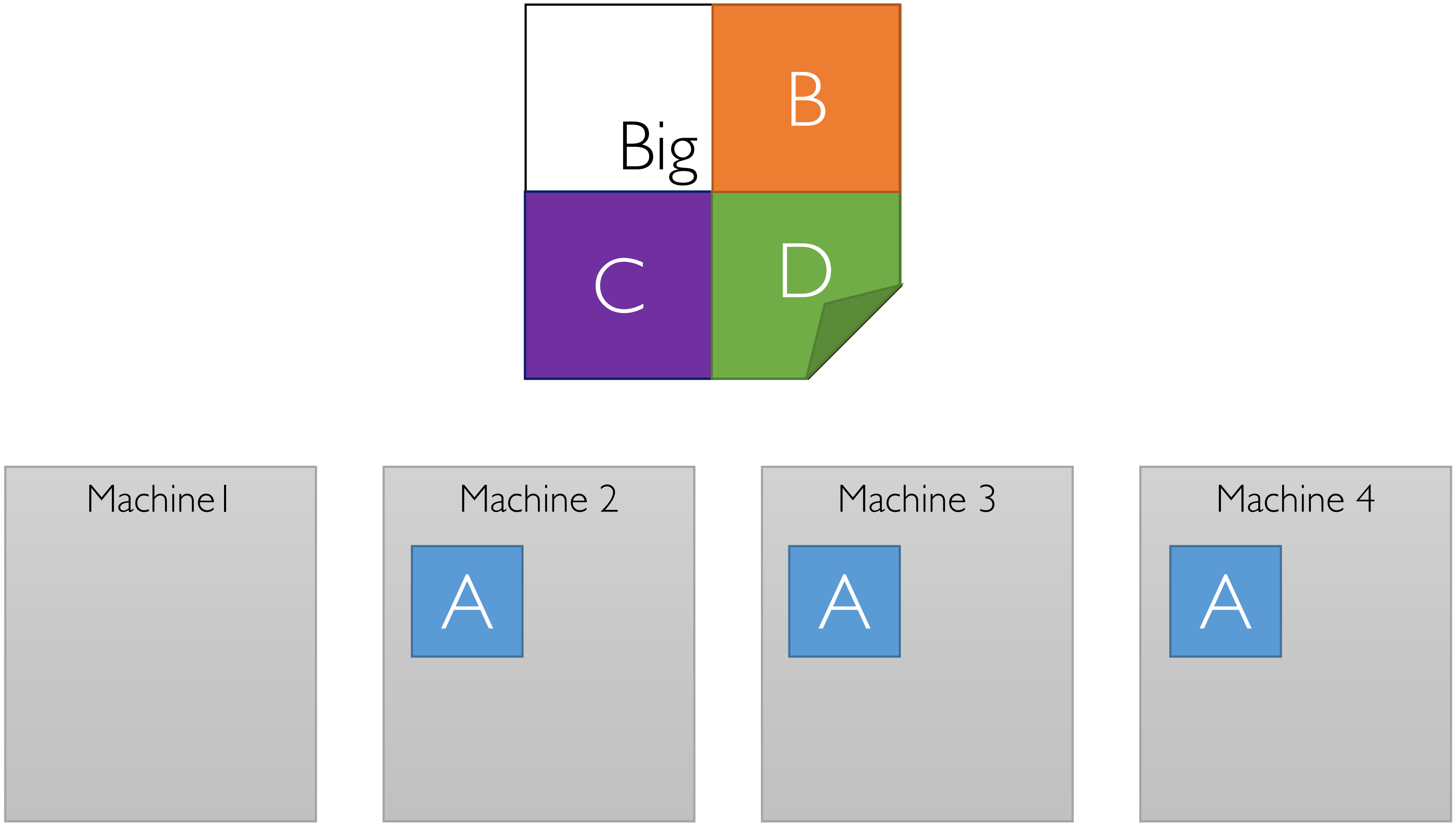


- Split the file into smaller parts.
- How?
  - Ideally at record boundaries
  - What if records are big?



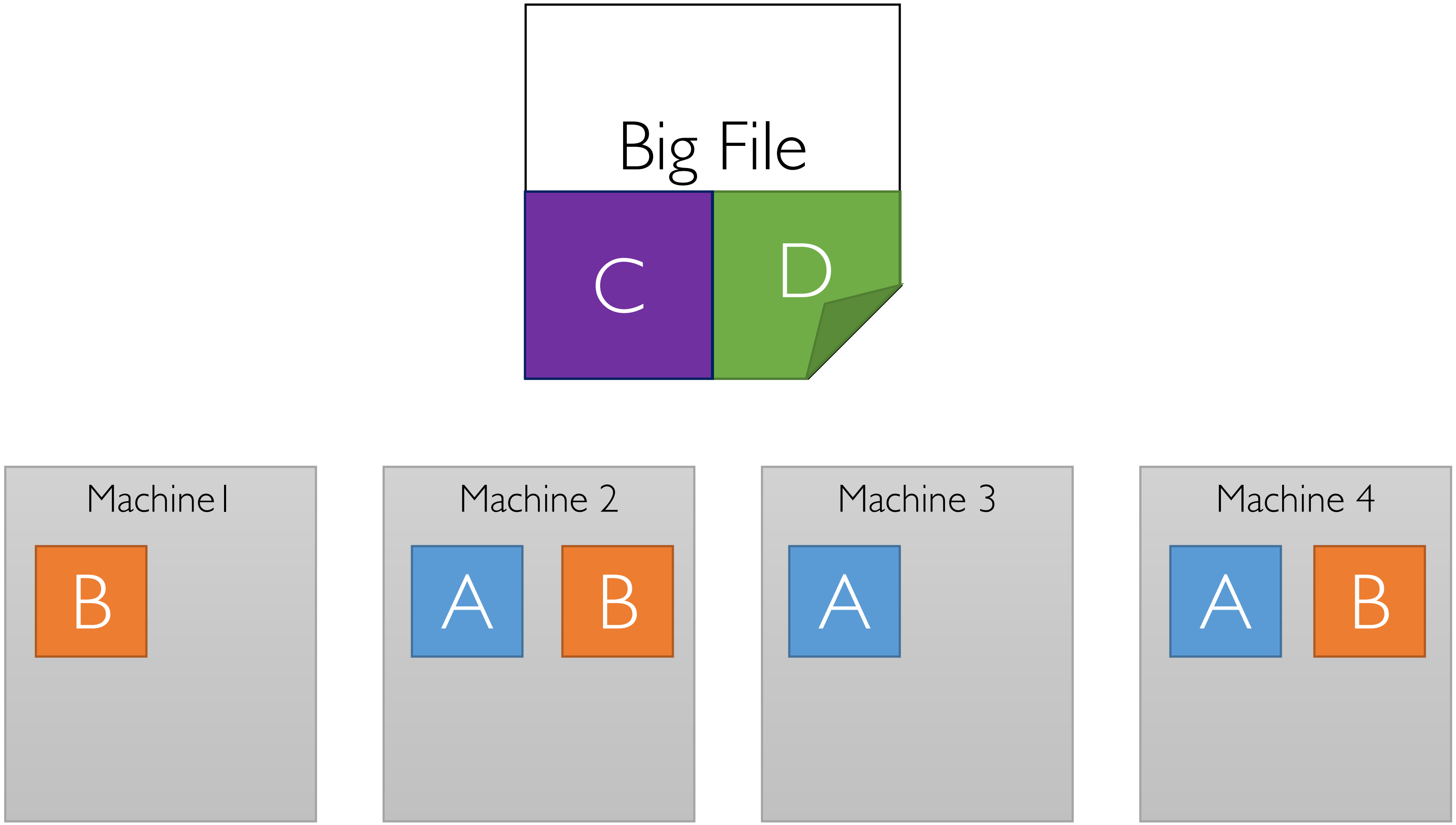
[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems



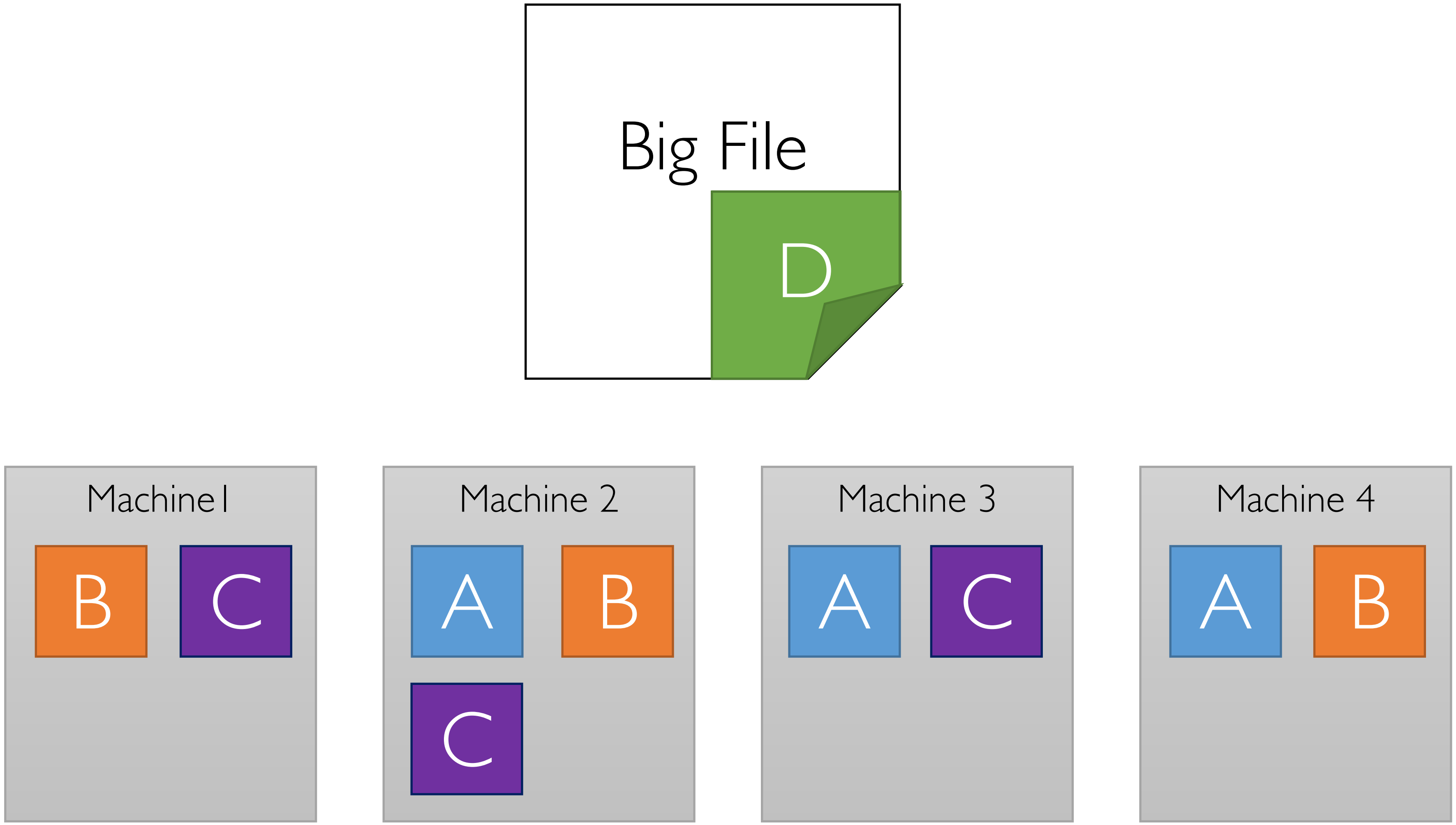
[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems



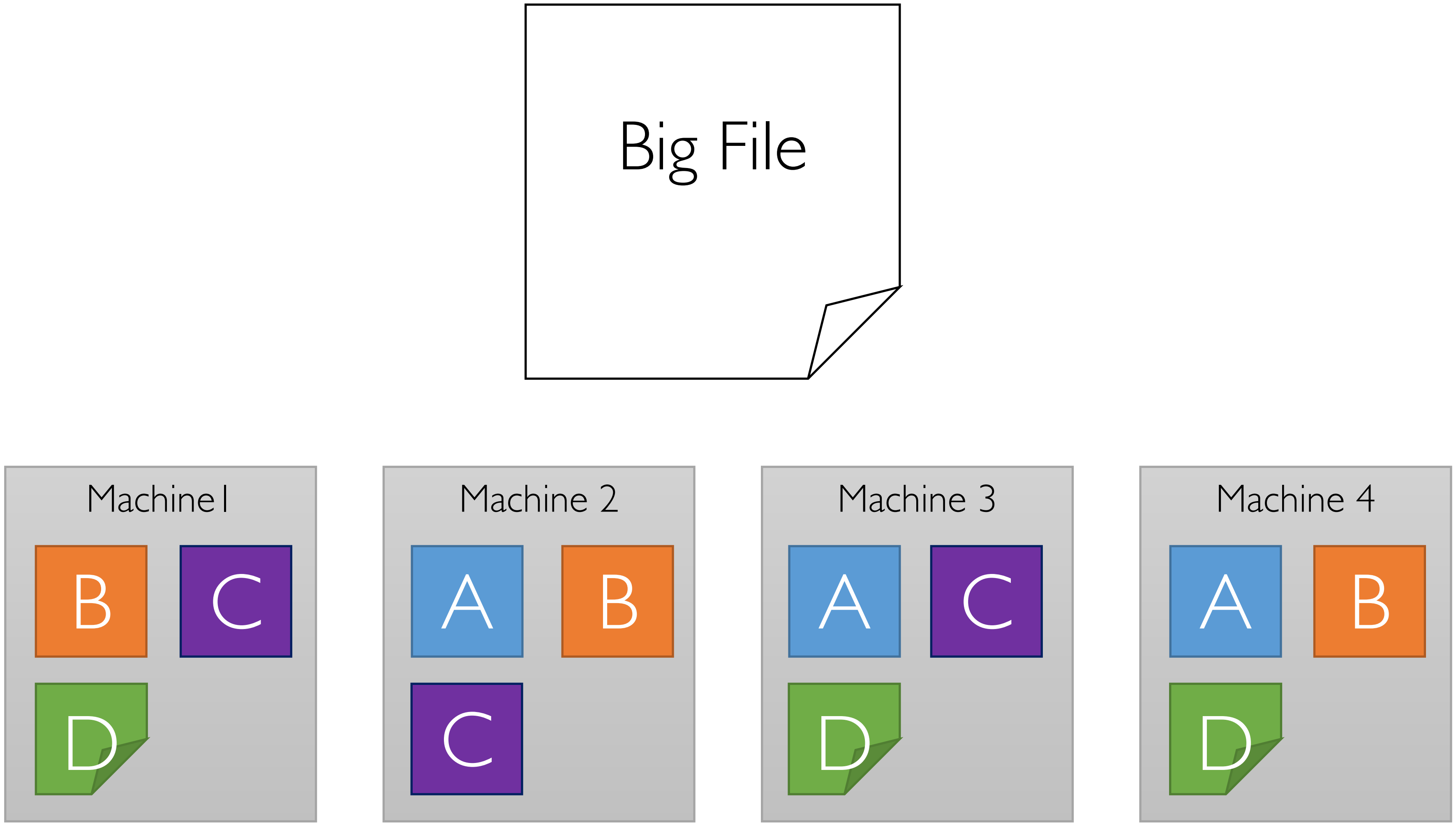
[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems



[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems

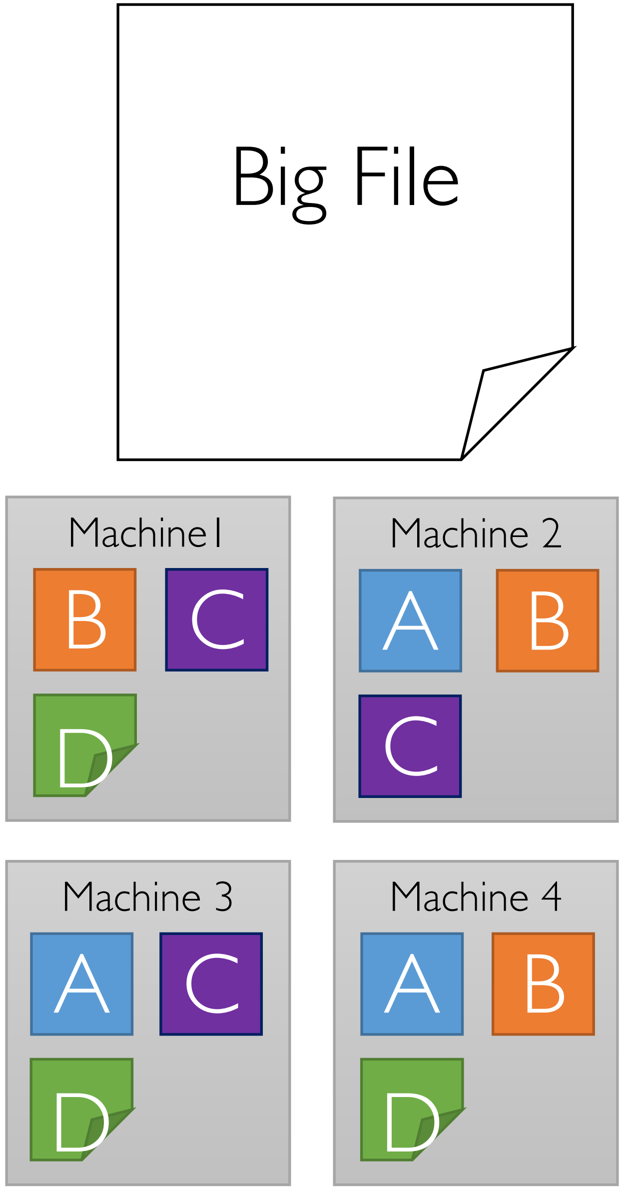


[Ghemawat et al., SOSP'03]

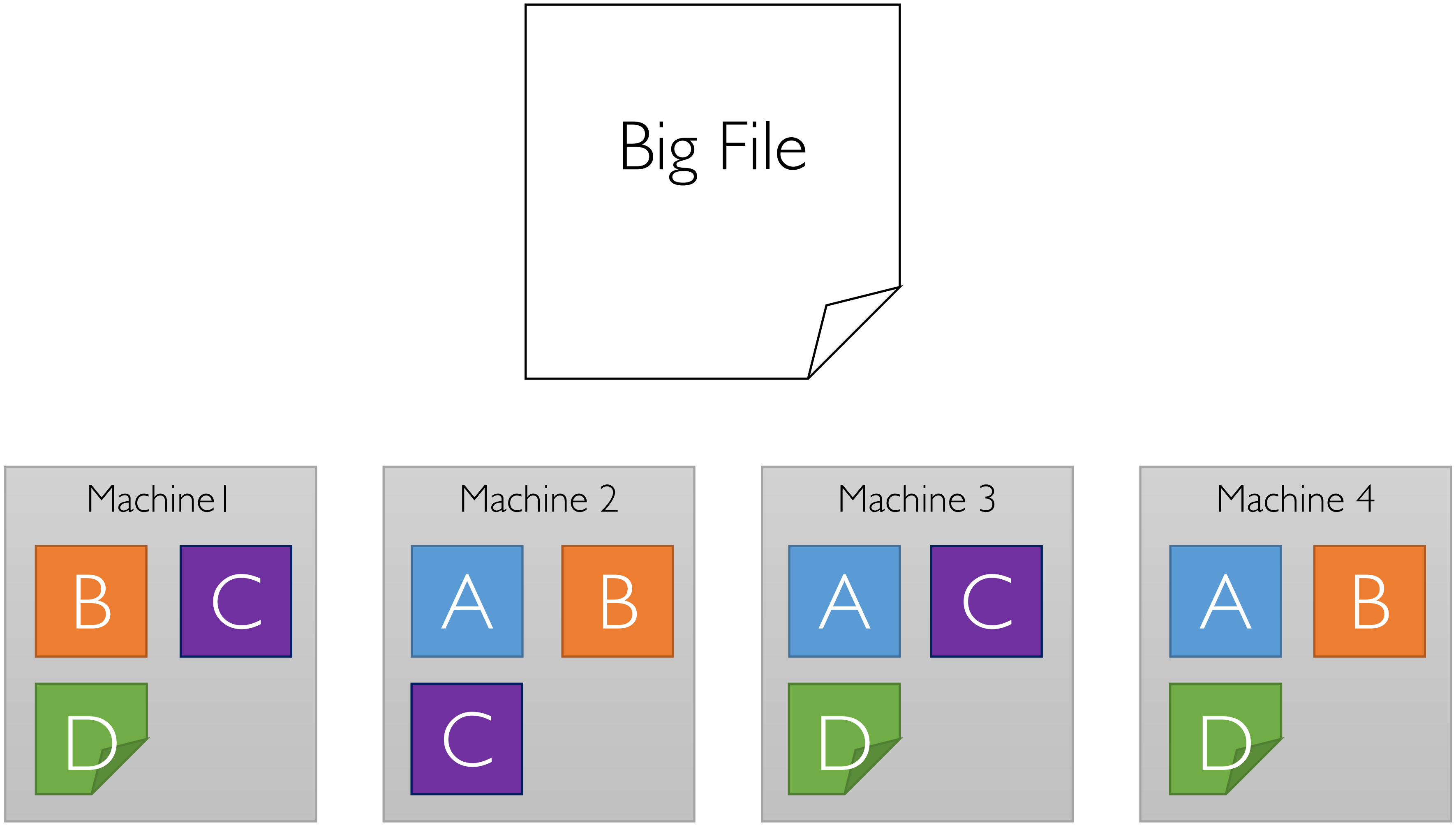
Fault Tolerant Distributed File Systems

- Split large files over multiple machines
  - Easily support massive files spanning machines
- Read parts of file in parallel
  - Fast reads of large files
- Often built using cheap commodity storage devices

Cheap commodity storage devices will fail!

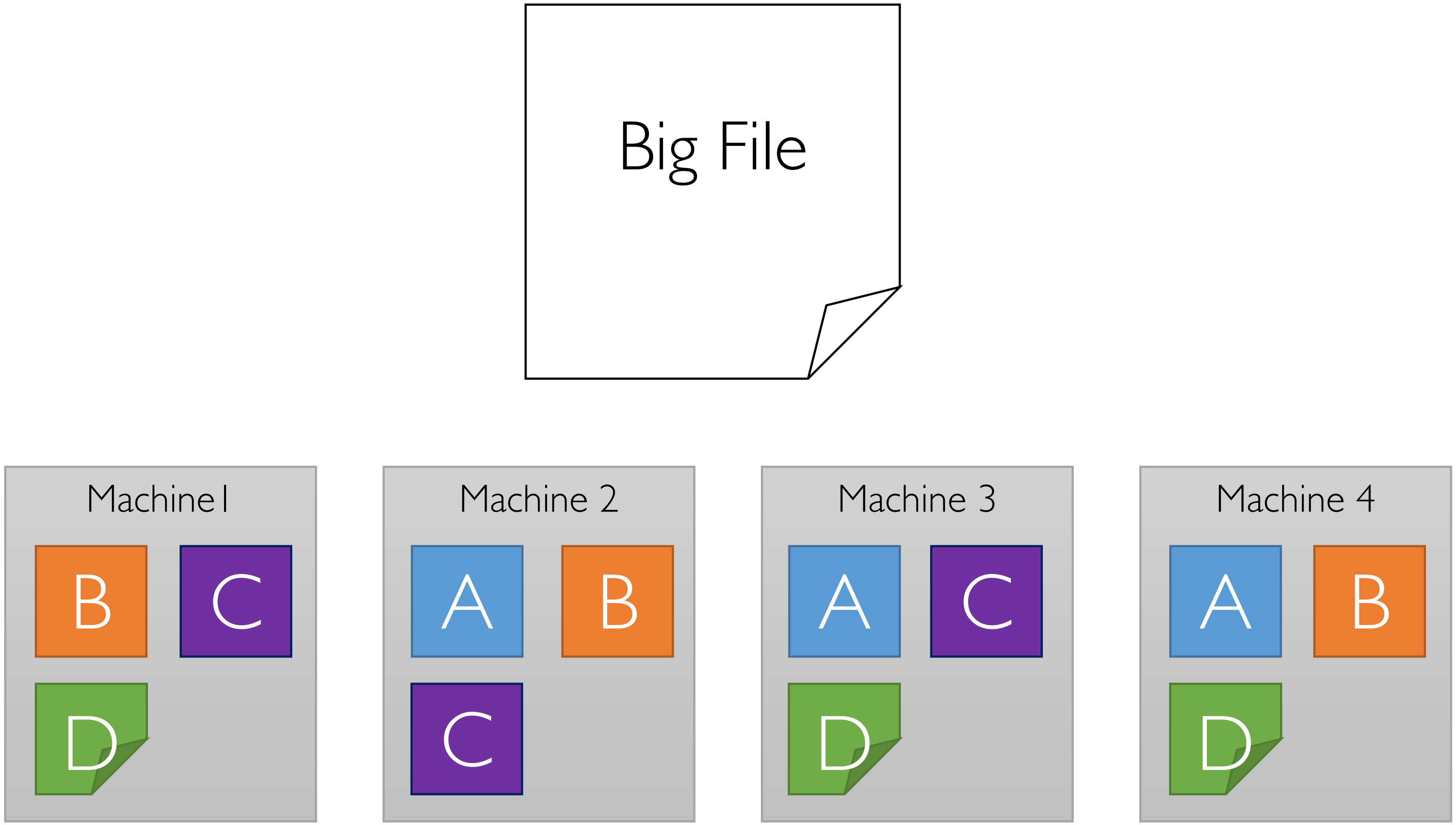


Fault Tolerant Distributed File Systems
Failure Event



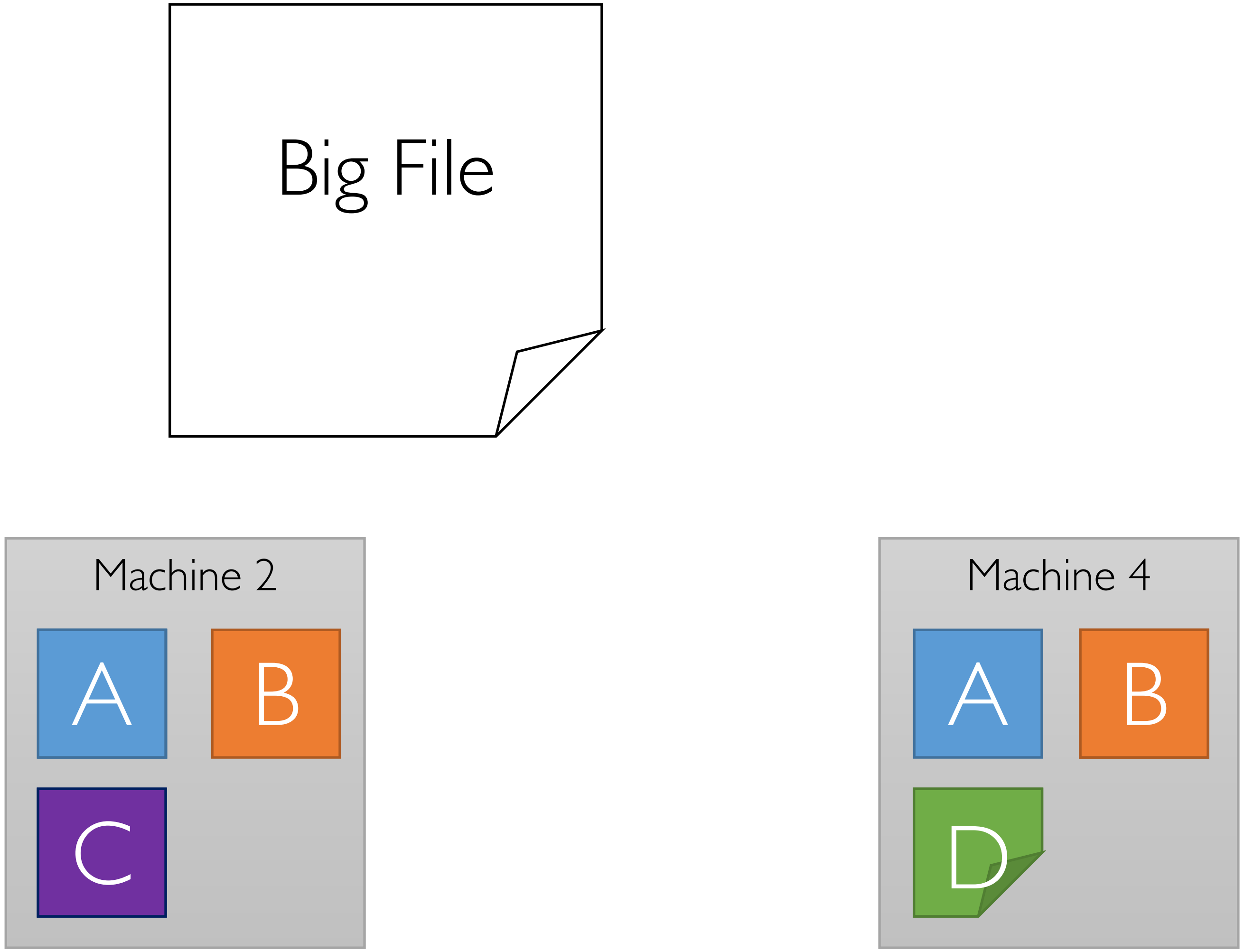
[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems
Failure Event



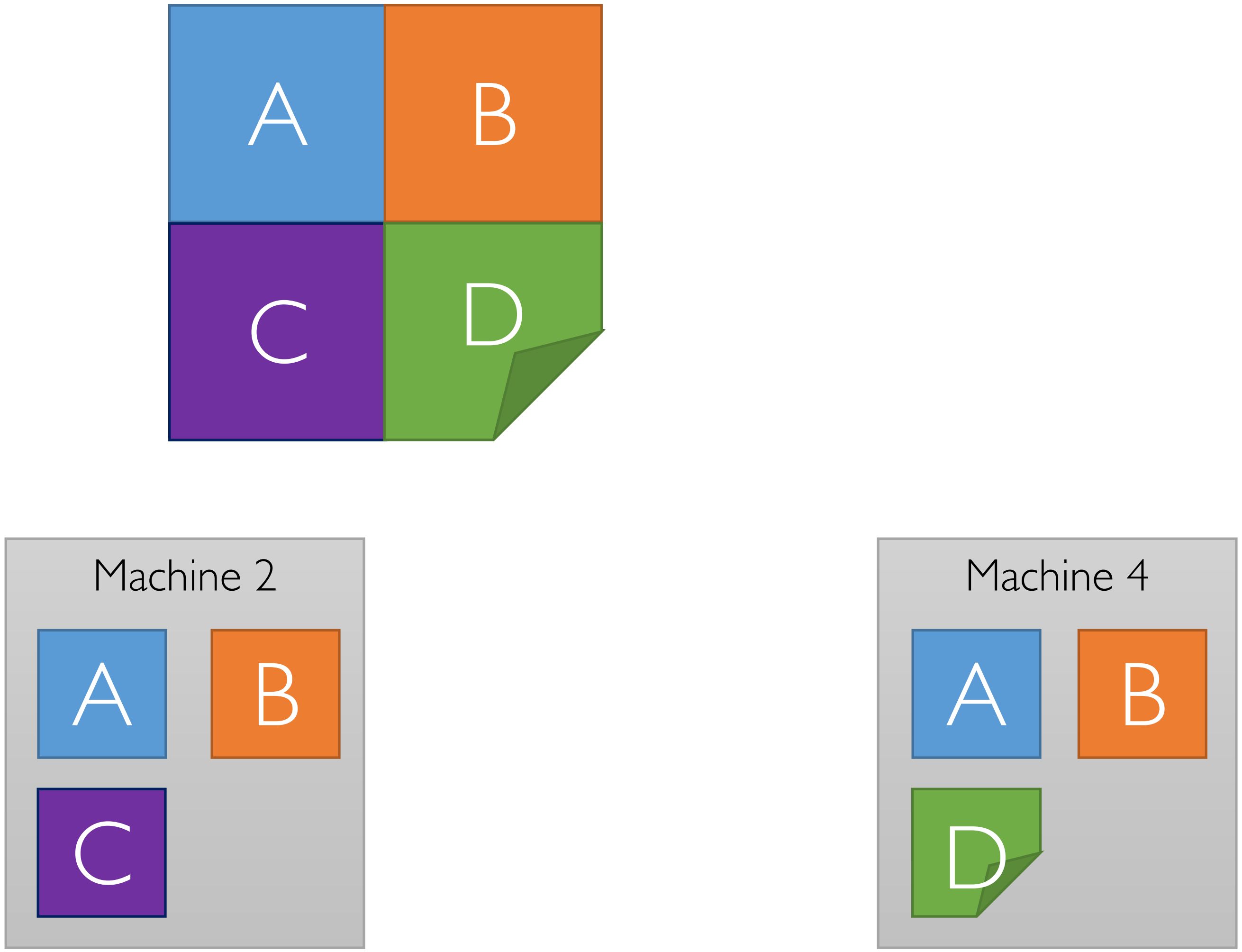
[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems
Failure Event



[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems
Failure Event



[Ghemawat et al., SOSP'03]

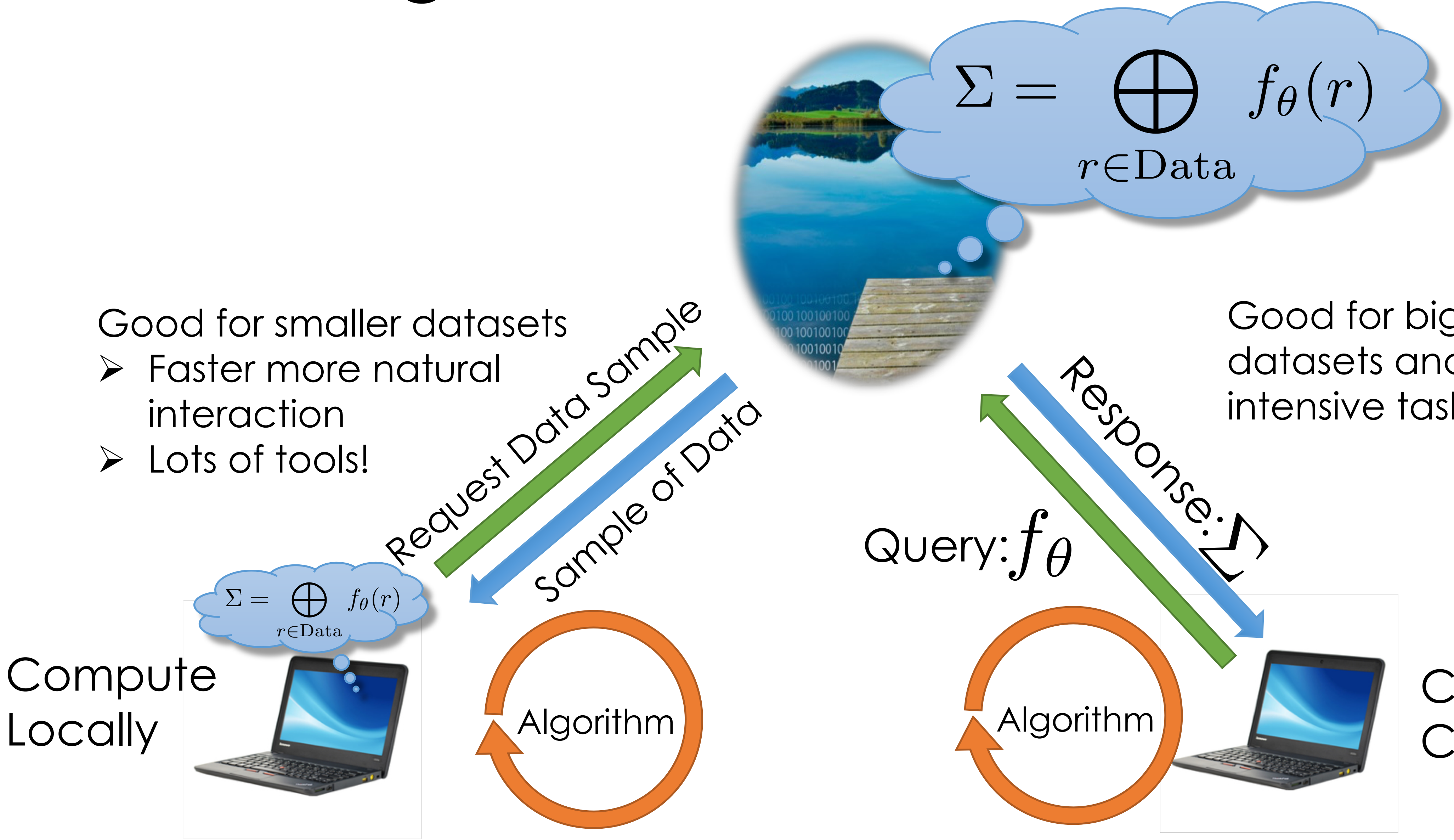
Map-Reduce Distributed Aggregation

Computing across very large files

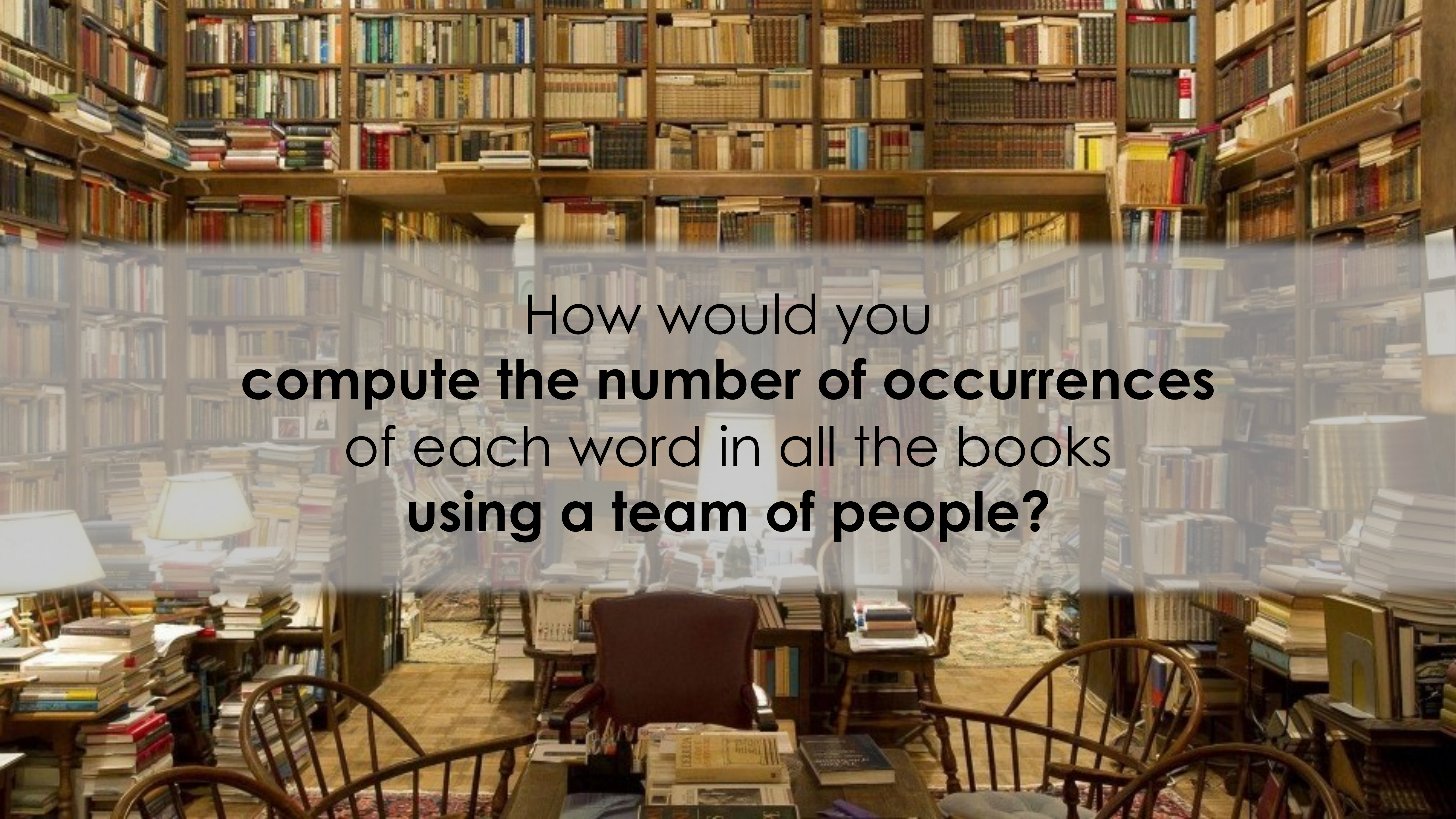
Interacting With the Data

Good for smaller datasets

- Faster more natural interaction
- Lots of tools!



Good for bigger datasets and compute intensive tasks

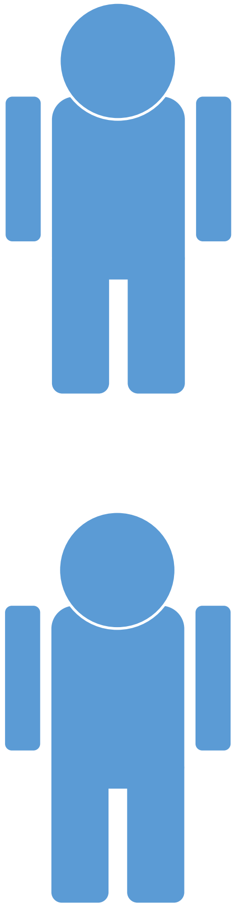


How would you
compute the number of occurrences
of each word in all the books
using a team of people?

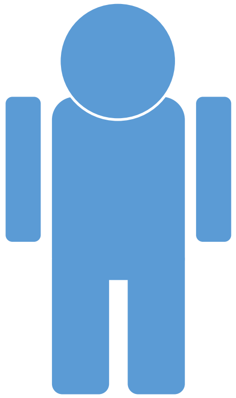
Simple Solution

Simple Solution

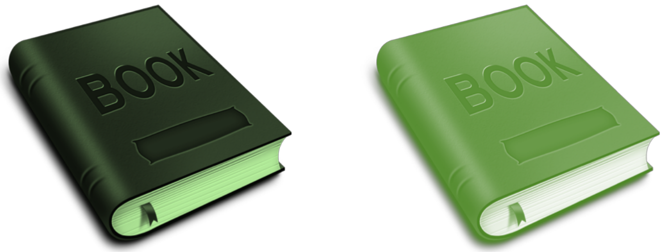
1) Divide Books Across Individuals



Simple Solution

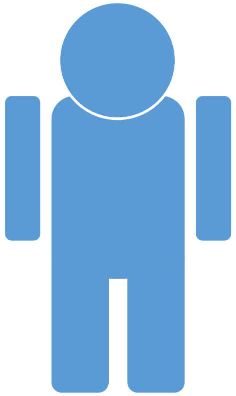


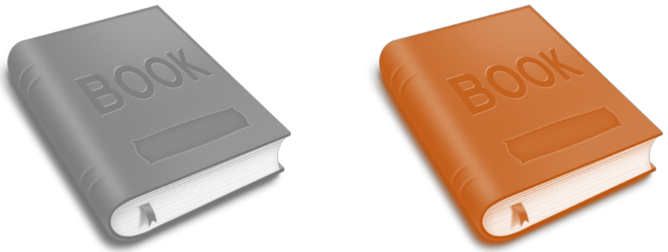
1) Divide Books Across Individuals



2) Compute Counts Locally

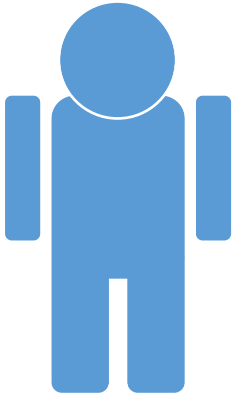
Word	Count
Apple	2
Bird	7
…	



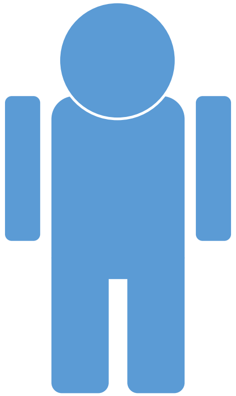


Word	Count
Apple	0
Bird	1
…	

Simple Solution



1) Divide Books Across Individuals



2) Compute Counts Locally

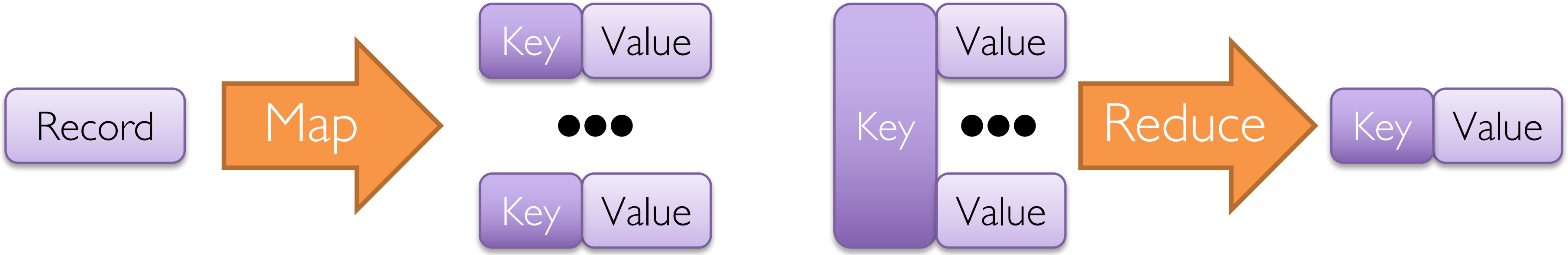
Word	Count
Apple	2
Bird	7
…	

Word	Count
Apple	0
Bird	1
…	

3) Aggregate Tables

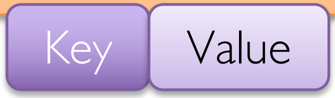
Word	Count
Apple	2
Bird	8
…	

The Map Reduce Abstraction



Example: *Word-Count*

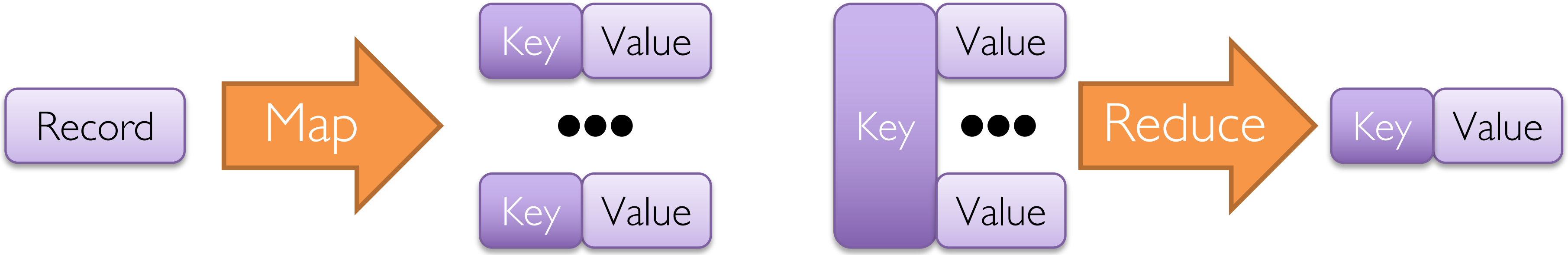
```
Map(book):
  for (word in set(book)):
    emit (word, book.count(word))
```




```
Reduce(word, counts) {
  sum = 0
  for count in counts:
     sum += count
  emit (word, SUM(counts))
}
```

[Dean & Ghemawat, OSDI'04]

The Map Reduce Abstraction (Simpler)



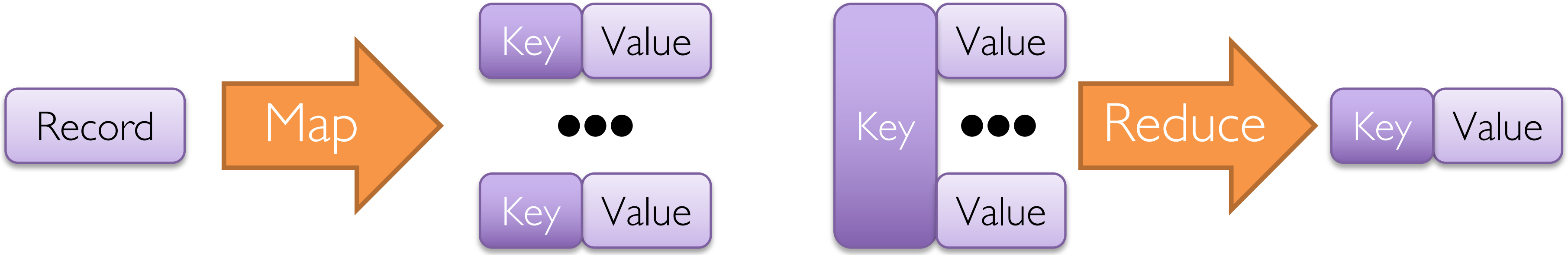
Example: *Word-Count*

```
Map(book):
  for (word in book):
    emit (word, 1)
```


```
Reduce(word, counts) {
  sum = 0
  for count in counts:
     sum += count
  emit (word, SUM(counts))
}
```

[Dean & Ghemawat, OSDI'04]

The Map Reduce Abstraction (General)



Example: *Word-Count*

```
Map(record, f):
  for (key in record):
    emit (key, f(key))
```


```
Reduce(key, values, f) {
  agg = f(values[0], values[1])
  for value in values[2:]:
      agg = f(agg, value)
  emit (word, agg)
}
```

Map: Deterministic
Reduce: Commutative and Associative

[Dean & Ghemawat, OSDI'04]

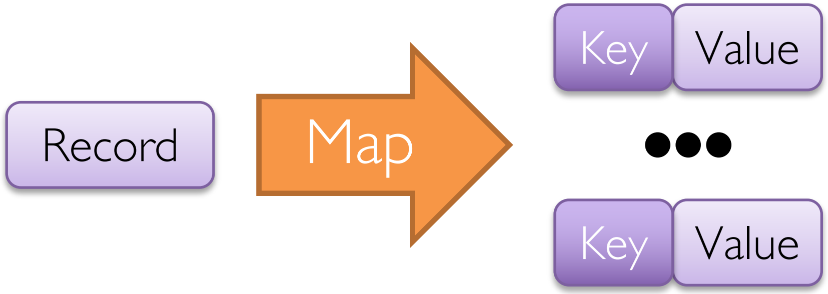
Key properties of Map And Reduce

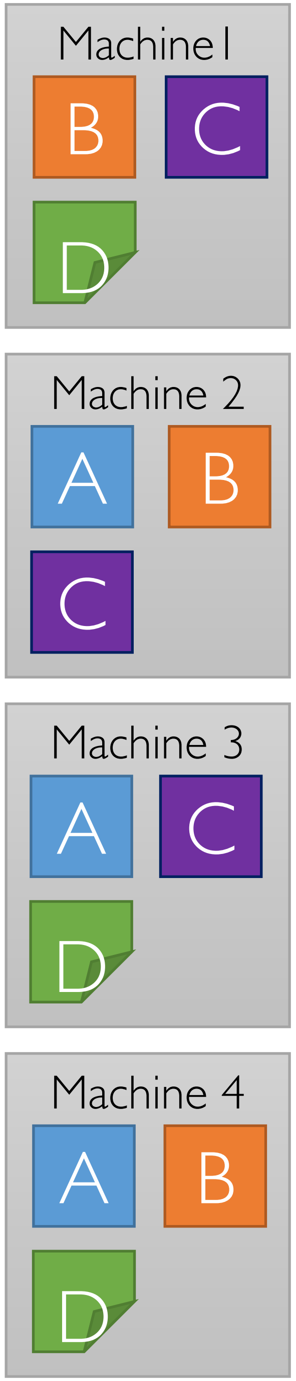
- **Deterministic Map:** allows for re-execution on failure
  - If some computation is lost we can always re-compute
  - Issues with samples?
- **Commutative Reduce:** *allows for re-order of operations*
  - Reduce(A,B) = Reduce(B,A)
  - Example (addition): A + B = B + A
- **Associative Reduce:** *allows for regrouping of operations*
  - Reduce(Reduce(A,B), C) = Reduce(A, Reduce(B,C))
  - Example (max): max(max(A,B), C) = max(A, max(B,C))
    - Warning: Floating point operations (e.g. addition) are not guaranteed associative.

Question

- Suppose our reduction function computes a*b + 1.
- Suppose we have 3 values associated with the key 'cat'. What is the result of the reduction operation?

Executing Map Reduce

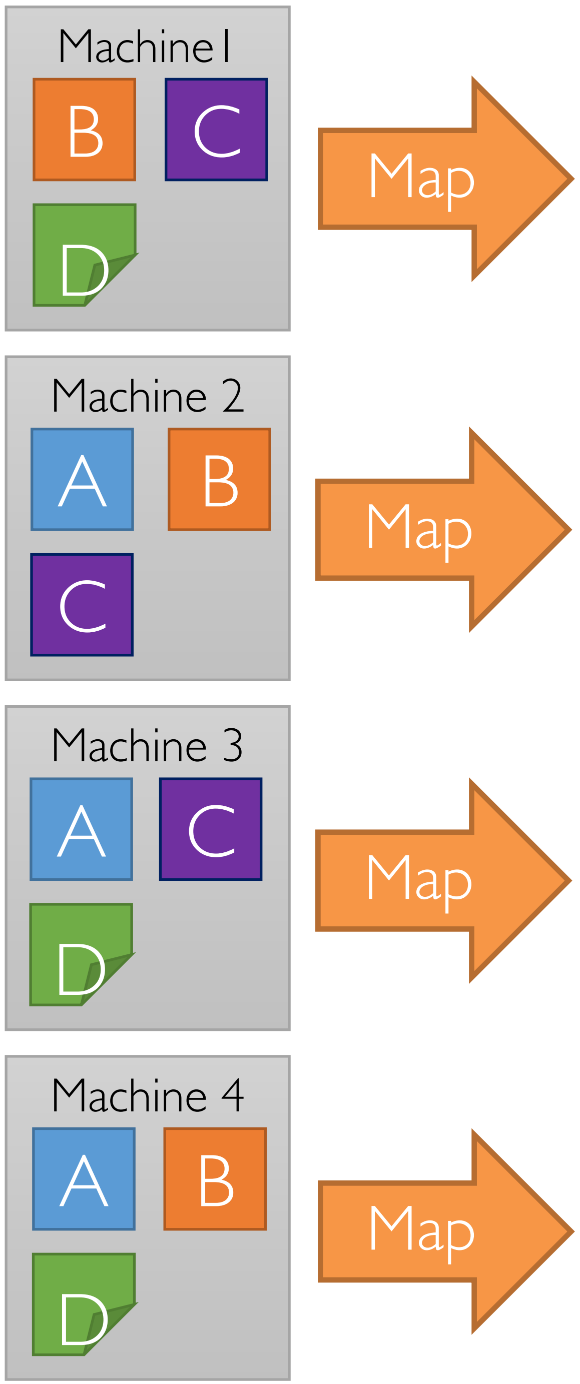




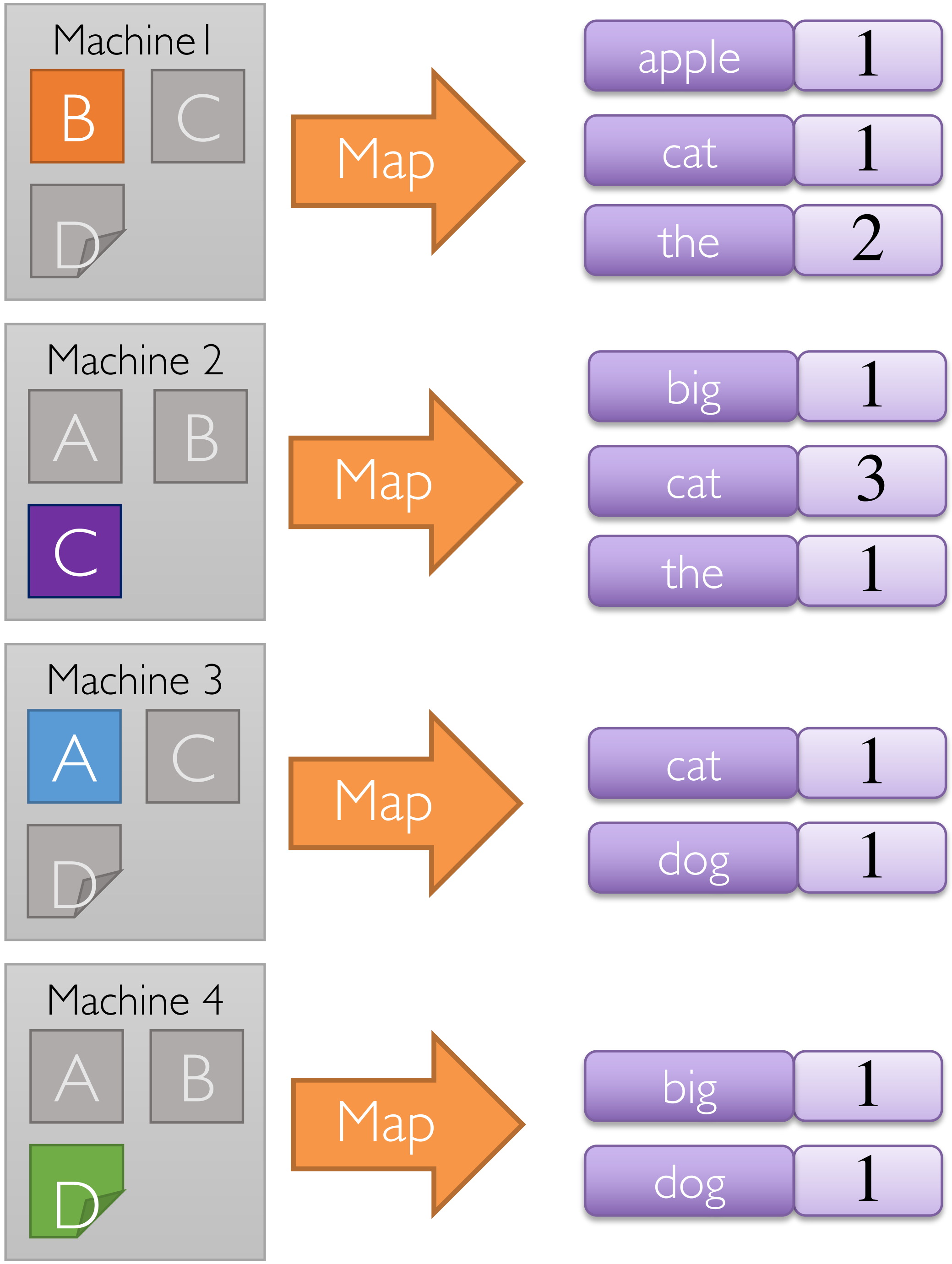
Executing Map Reduce

Distributing the Map Function

Executing Map Reduce



Distributing the Map Function



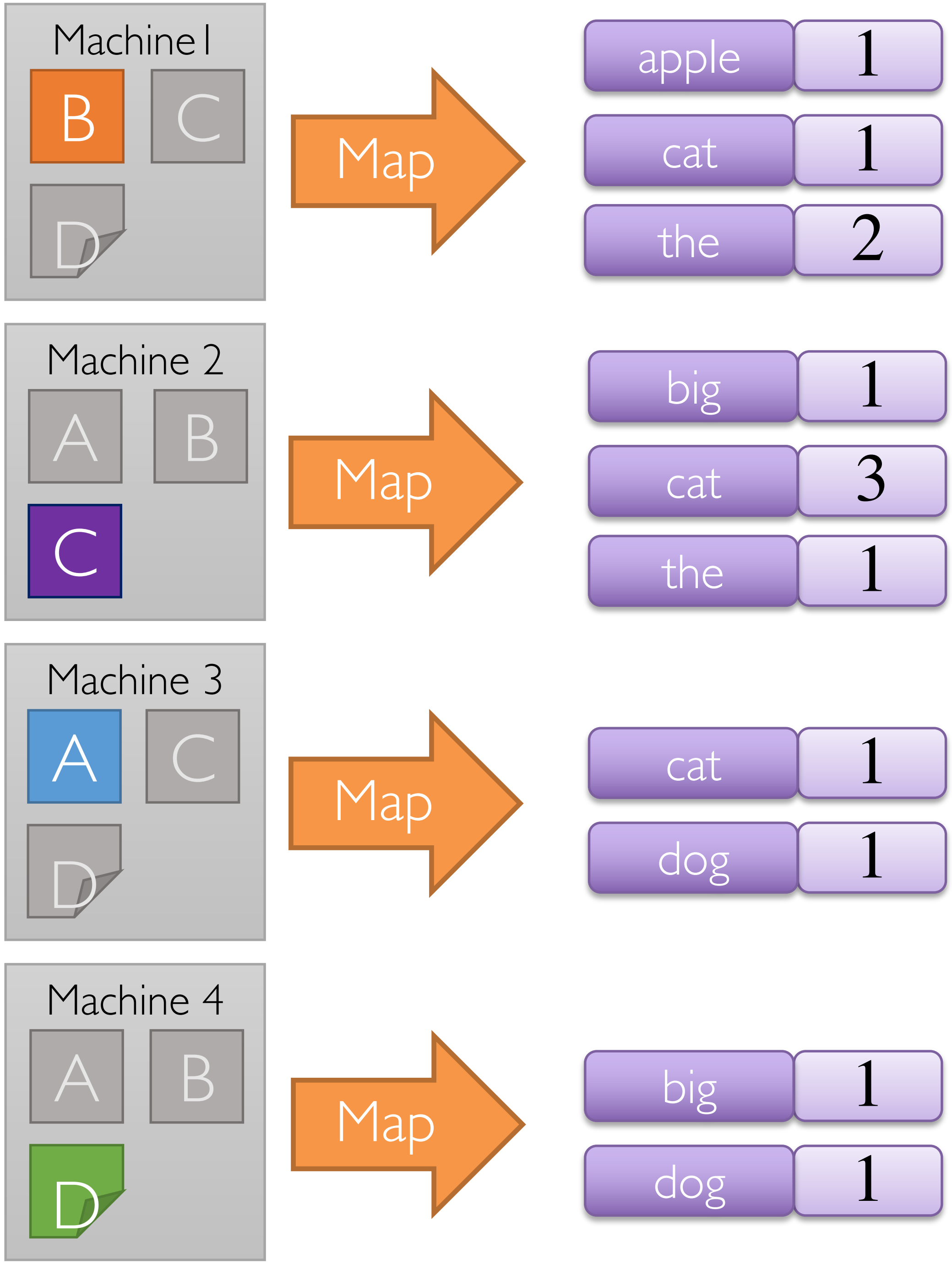
Executing Map Reduce

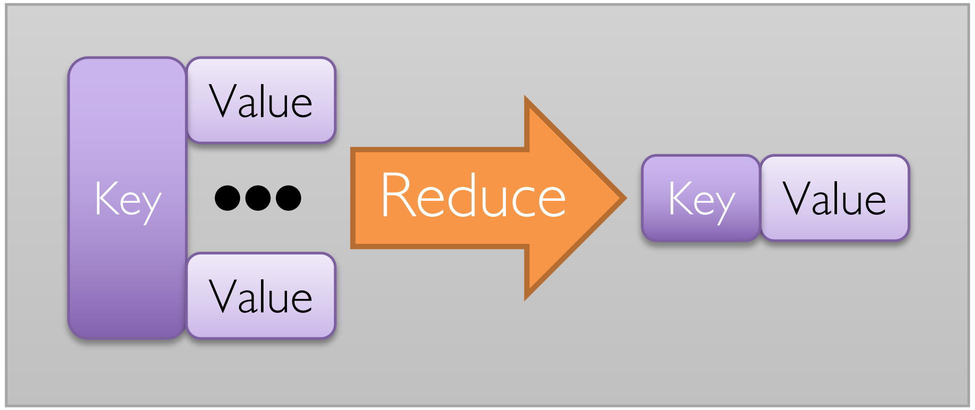
The map function applied to a local part of the big file.

Run in Parallel.

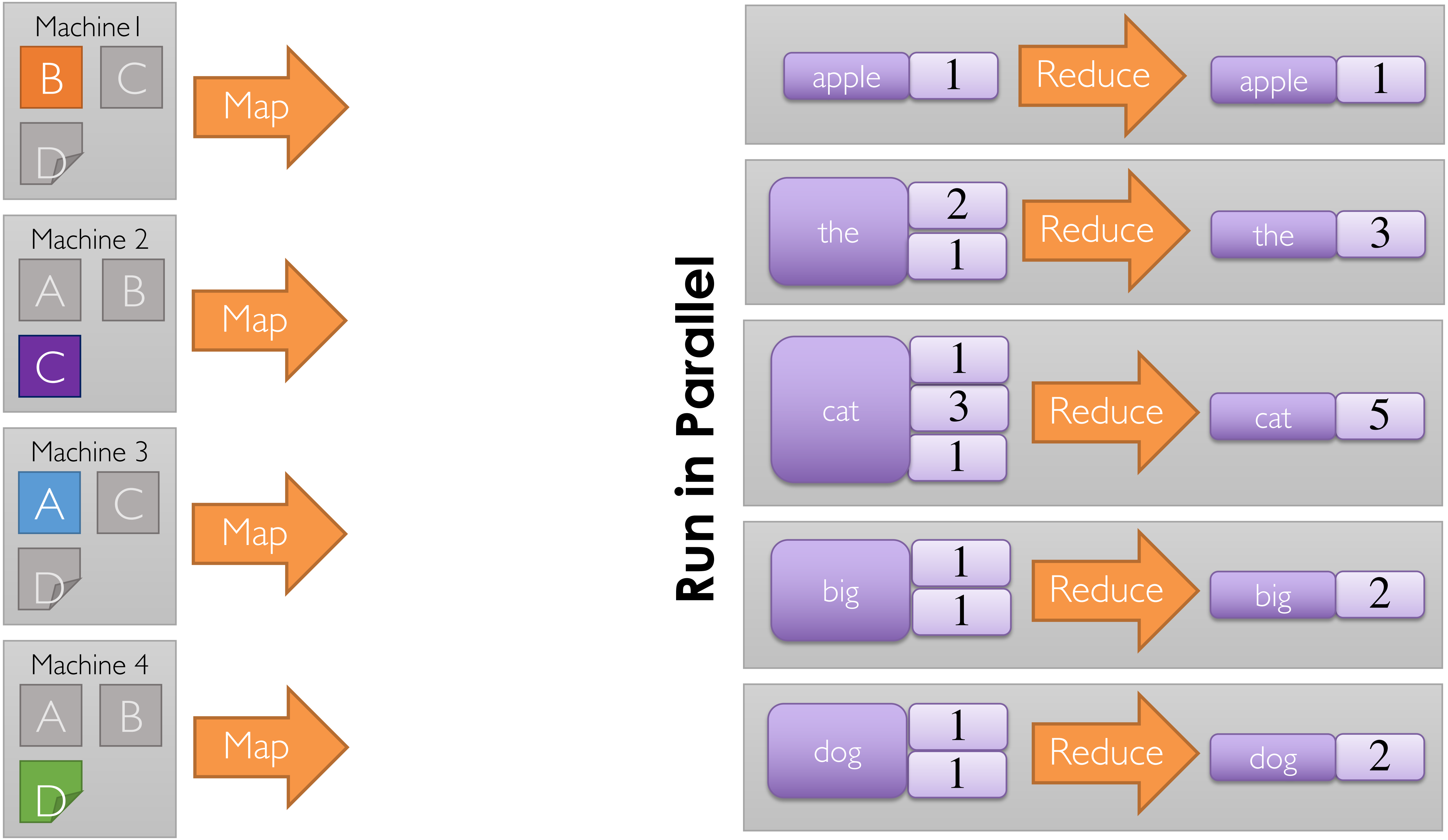
Output is cached for fast recovery on node failure

Executing Map Reduce

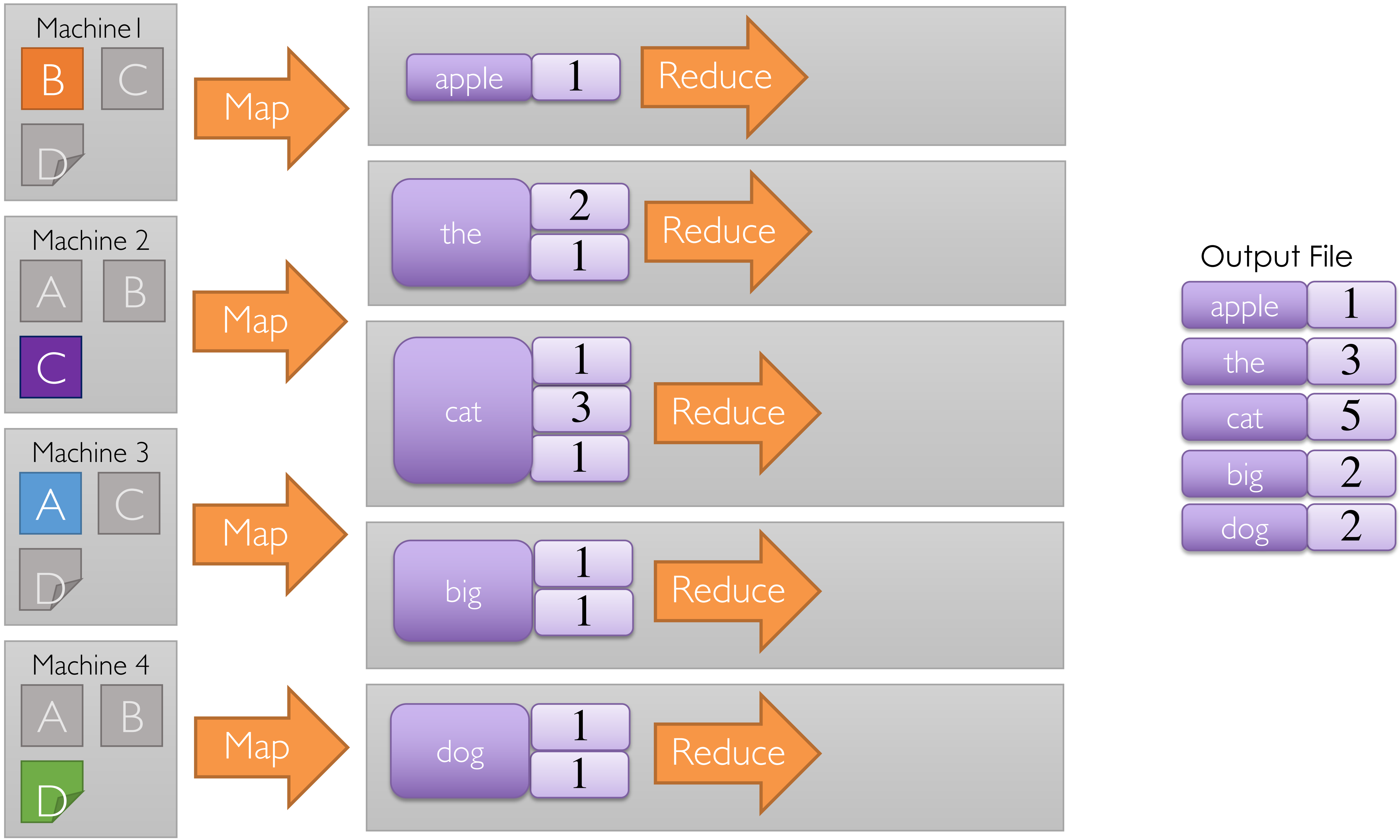




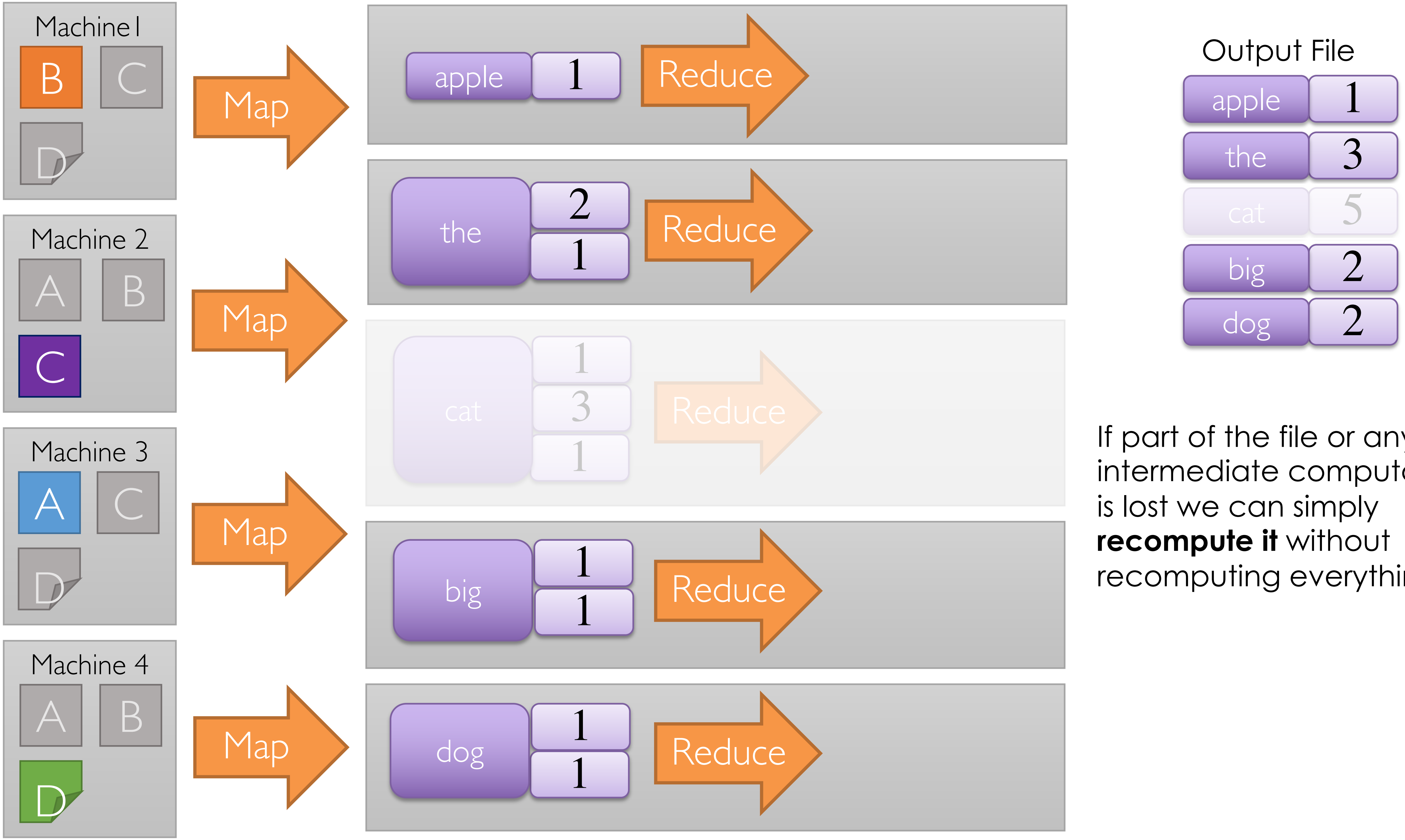
Reduce function can be run on many machines …



Machine 1
B
C
D
Map
Machine 2
A
B
C
Map
Machine 3
A
C
D
Map
Machine 4
A
B
D
Map
Run in Parallel
apple
1
Reduce
apple
1
the
2
1
Reduce
the
3
cat
1
3
1
Reduce
cat
5
big
1
1
Reduce
big
2
dog
1
1
Reduce
dog
2



Machine 1
B
C
D
Machine 2
A
B
C
Machine 3
A
C
D
Machine 4
A
B
D
Map
Map
Map
Map
apple
1
Reduce
the
2
1
Reduce
cat
1
3
1
Reduce
big
1
1
Reduce
dog
1
1
Reduce
Output File
apple
1
the
3
cat
5
big
2
dog
2



If part of the file or any intermediate computation is lost we can simply **recompute it** without recomputing everything.

Map Reduce Technologies

Hadoop

- First open-source map-reduce software
  - Managed by Apache foundation
- Based on Google's
  - Google File System
  - MapReduce
- Companies formed around Hadoop:
  - Cloudera
  - Hortonworks
  - MapR

Hadoop

- Very active open source ecosystem
- Several key technologies
  - **HDFS:** Hadoop File System
  - **MapReduce:** map-reduce compute framework
  - **YARN:** Yet another resource negotiator
  - **Hive:** SQL queries over MapReduce
  - …
- Downside: Tedious to use!
  - Joey: Word count example from before is 100s of lines of Java code.

In-Memory Dataflow System

Developed at the UC Berkeley AMP Lab

M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. *Spark: cluster computing with working sets.* HotCloud'10

M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, I. Stoica. *Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing*, NSDI 2012

What Is Spark

- Parallel execution engine for big data processing
- **General**: efficient support for multiple workloads
- **Easy** to use: 2-5x less code than Hadoop MR
  - High level API's in Python, Java, and Scala
- **Fast**: up to 100x faster than Hadoop MR
  - Can exploit in-memory when available
  - Low overhead scheduling, optimized engine

Spark Programming Abstraction

- *Write programs in terms of transformations on distributed datasets*
- Resilient Distributed Datasets (RDDs)
  - Distributed collections of objects that can stored in memory or on disk
  - Built via parallel transformations (map, filter, …)
  - Automatically rebuilt on failure

RDD: Resilient Distributed Datasets

- Collections of objects partitioned & distributed across a cluster
  - Stored in RAM or on Disk
  - Resilient to failures
- Operations
  - Transformations
  - Actions

Operations on RDDs

- Transformations f(RDD) => RDD
  - Lazy (not computed immediately)
  - E.g., “map”, “filter”, “groupBy”

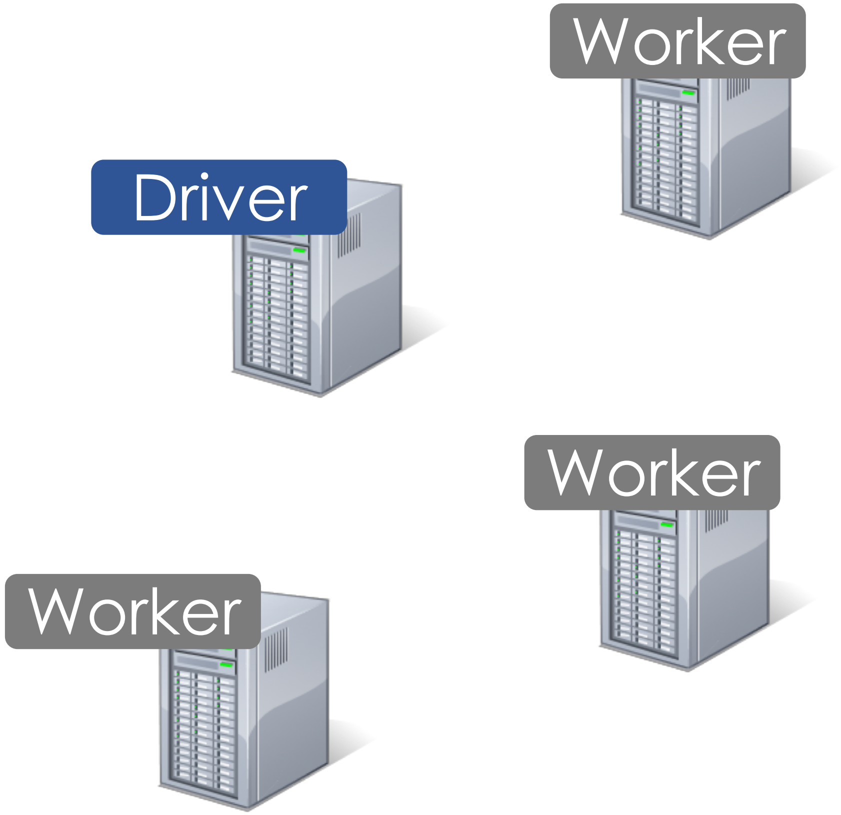
- Actions:
  - Triggers computation
  - E.g. “count”, “collect”, “saveAsTextFile”

Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

Example: Log Mining

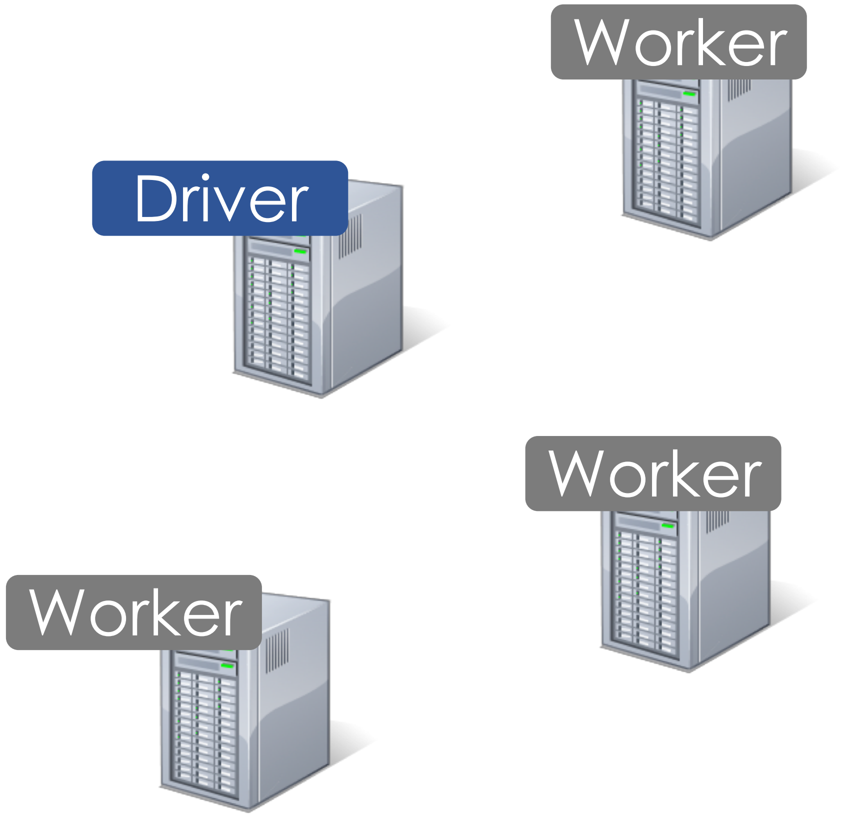
Load error messages from a log into memory, then interactively search for various patterns



Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://file.txt")
```

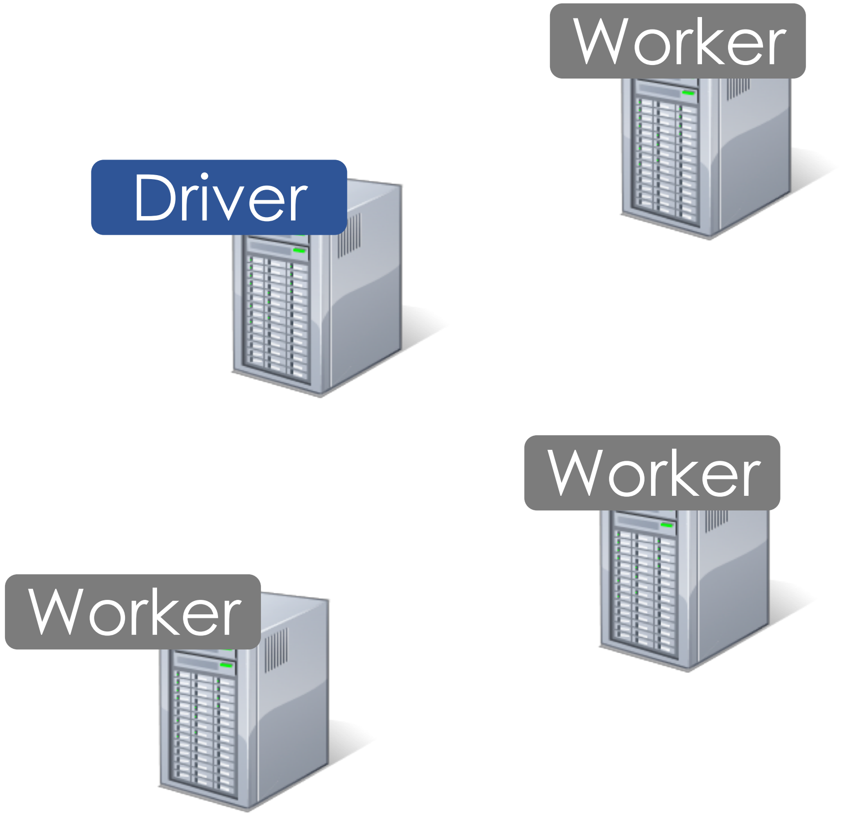


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

Base RDD

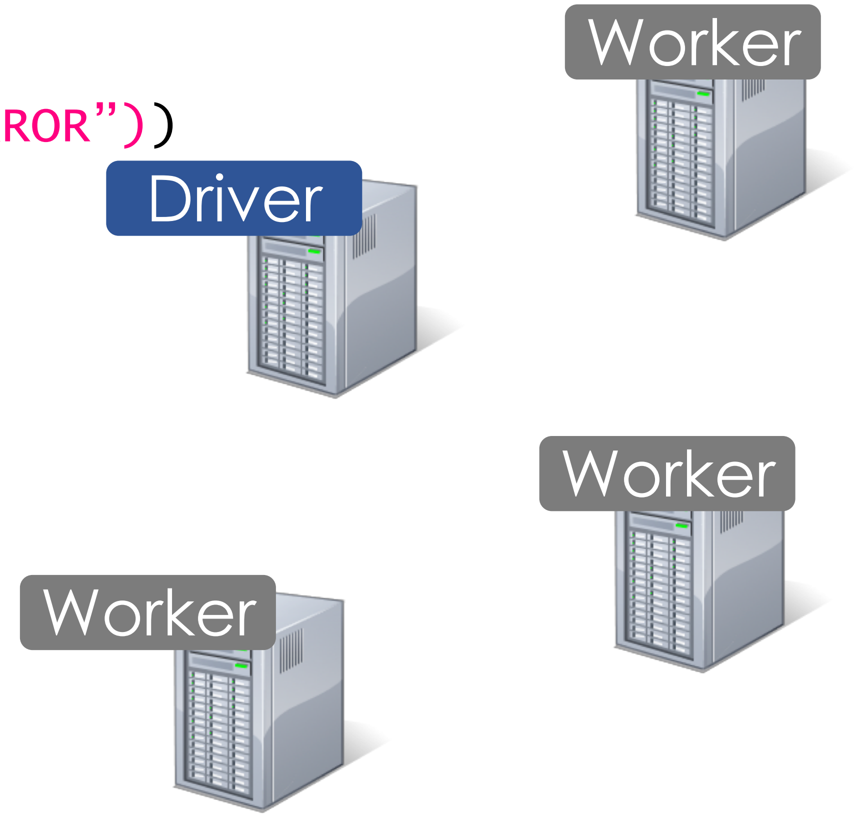
```
lines = spark.textFile("hdfs://file.txt")
```



Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
```

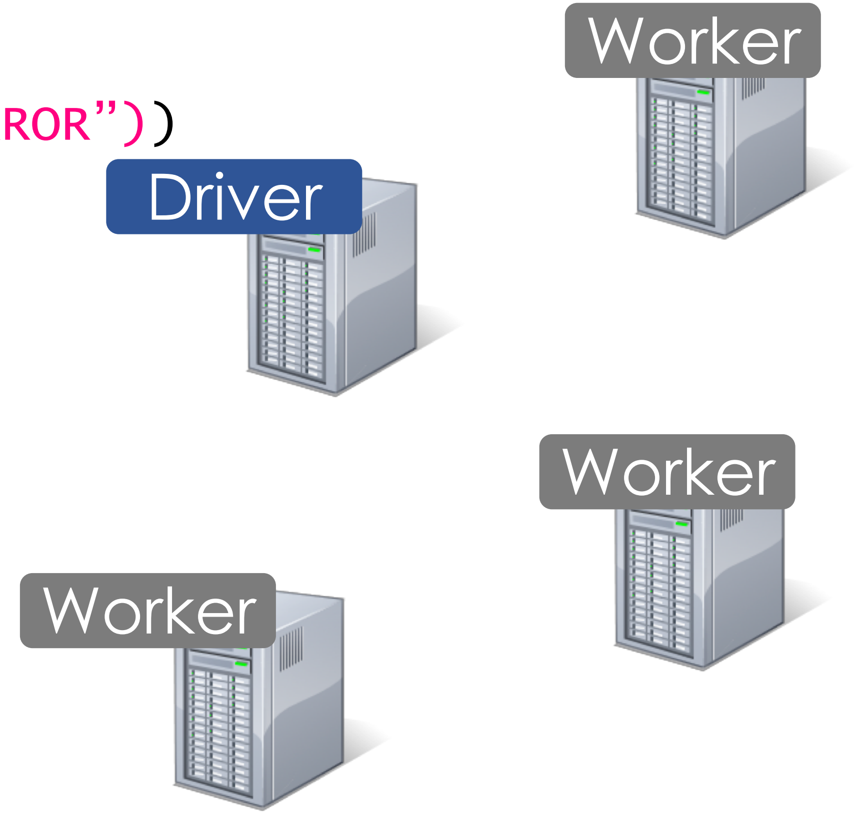


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

Transformed RDD

```
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
```

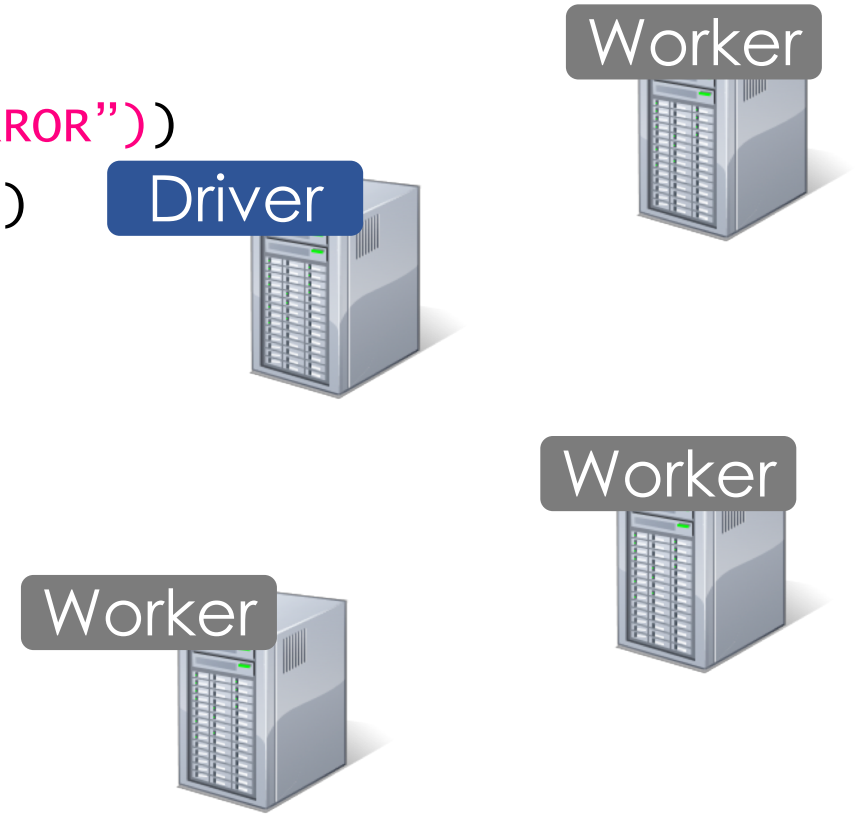


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://file.txt”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
messages = errors.map(lambda s: s.split(“\t”)[2])
messages.cache()

messages.filter(lambda s: “mysql” in s).count()
```

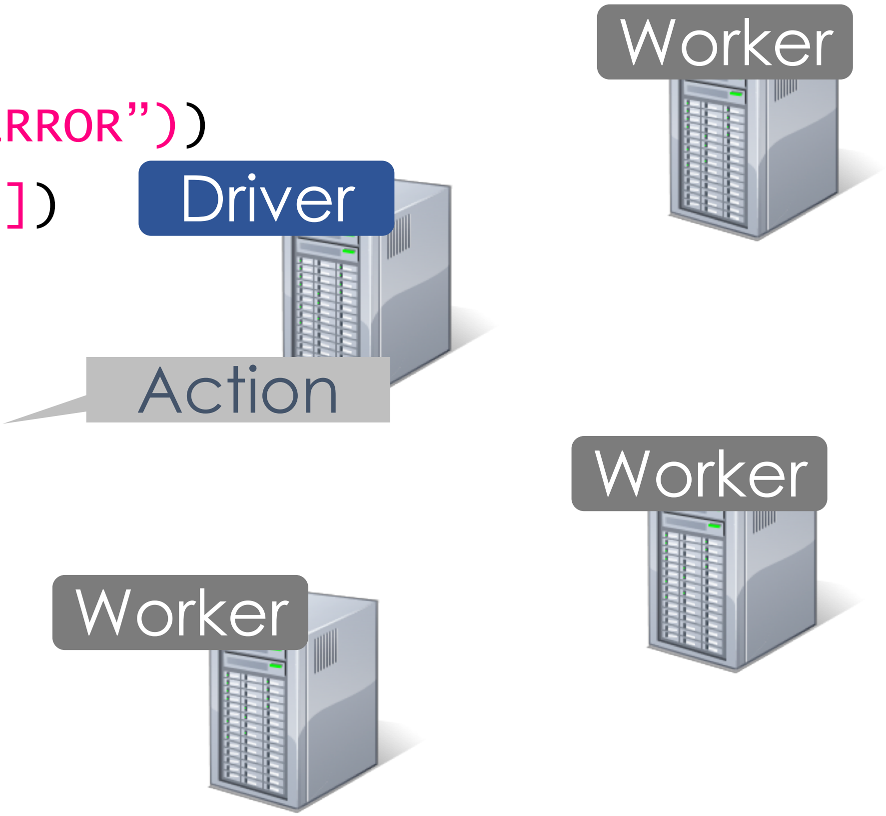


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://file.txt”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
messages = errors.map(lambda s: s.split(“\t”)[2])
messages.cache()

messages.filter(lambda s: “mysql” in s).count()
```

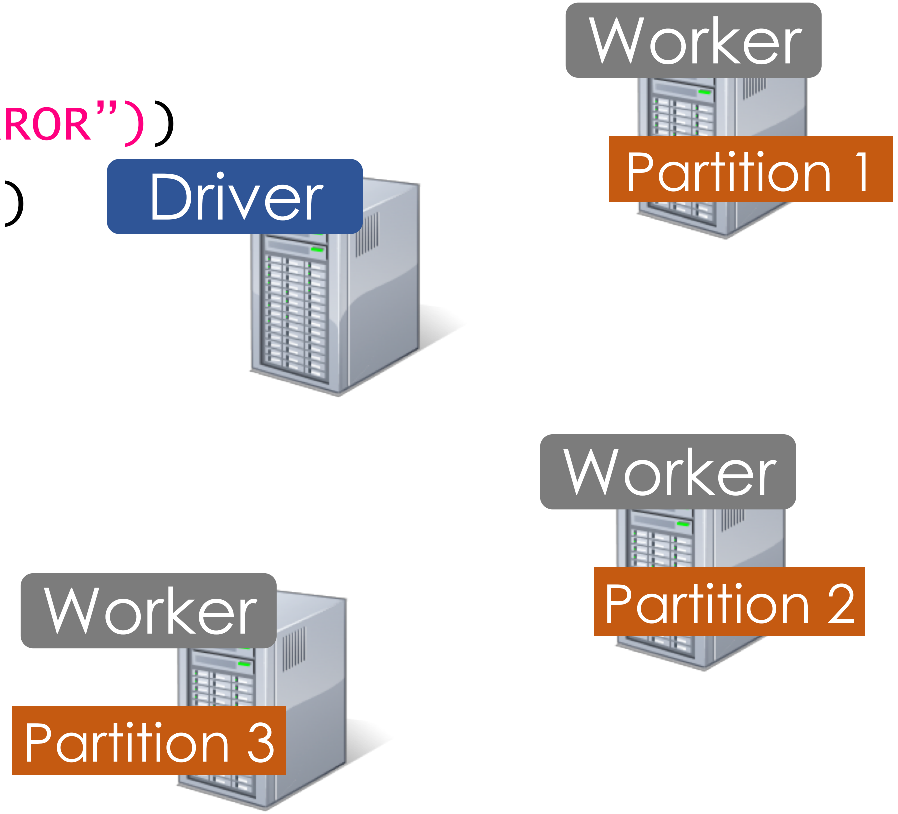


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
```

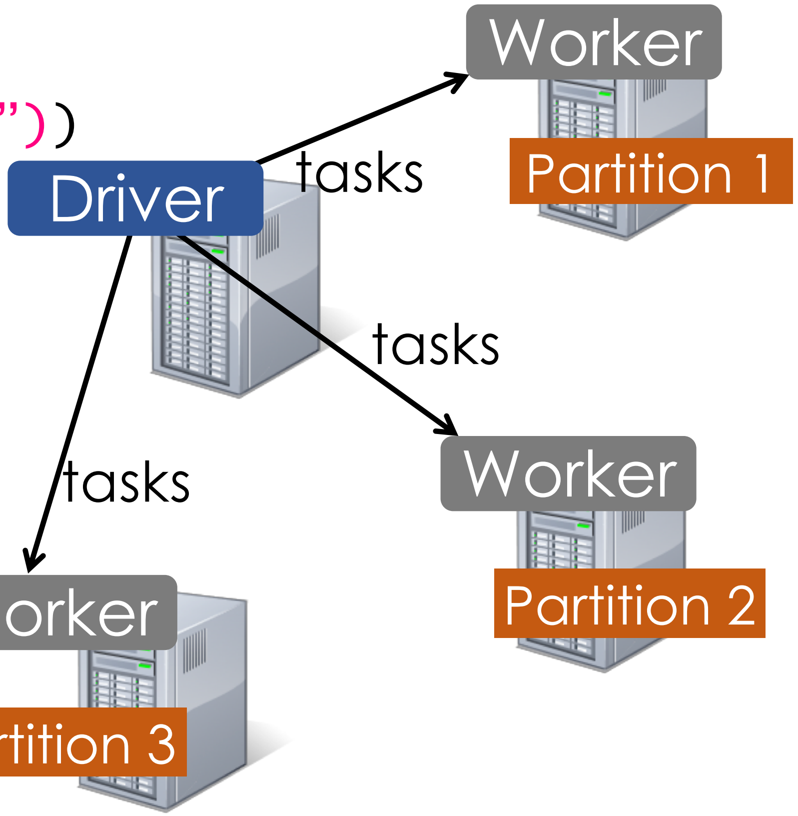


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
```

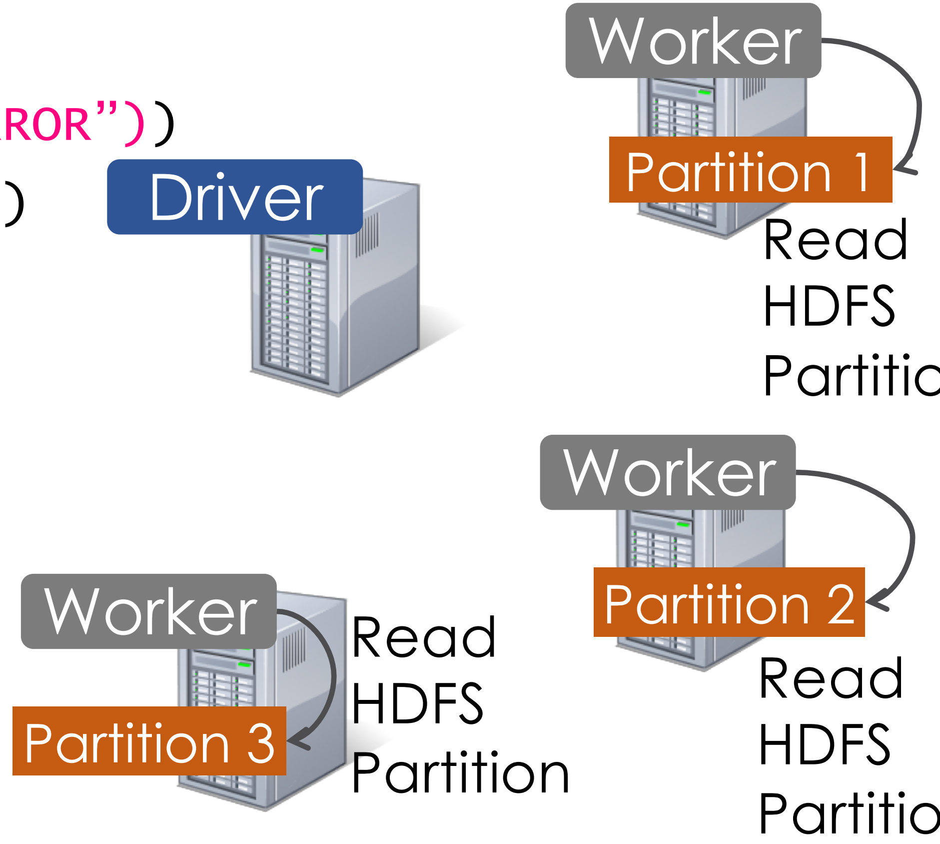


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
```

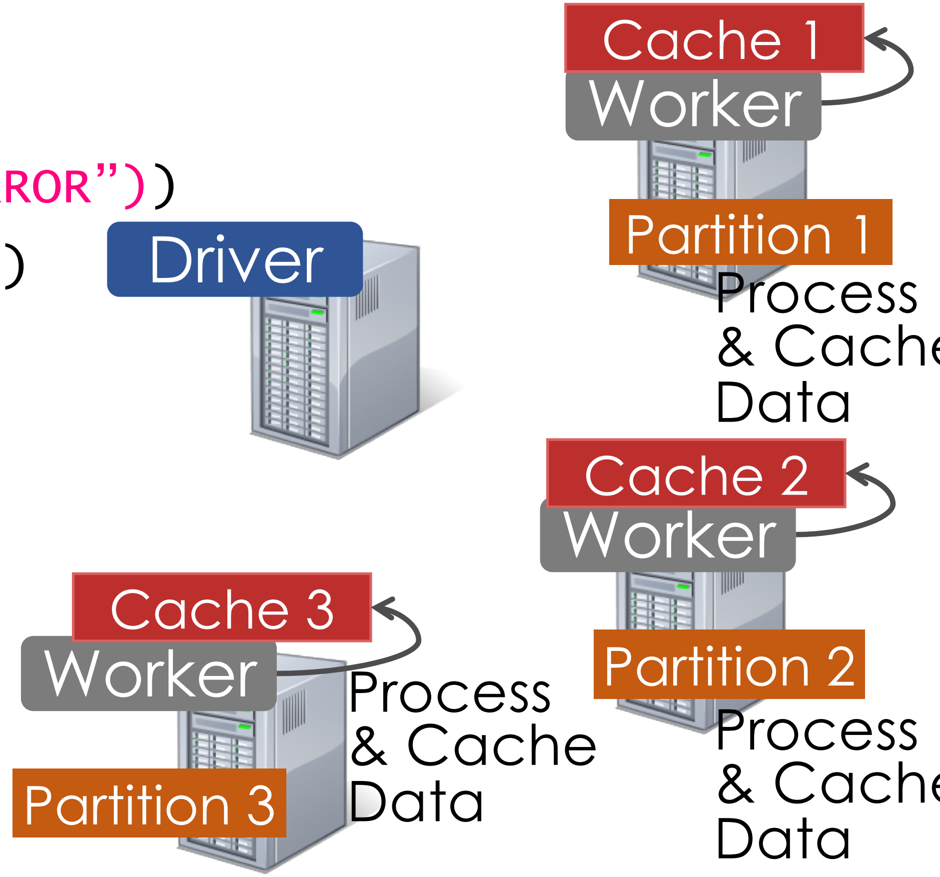


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://file.txt”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
messages = errors.map(lambda s: s.split(“\t”)[2])
messages.cache()

messages.filter(lambda s: “mysql” in s).count()
```

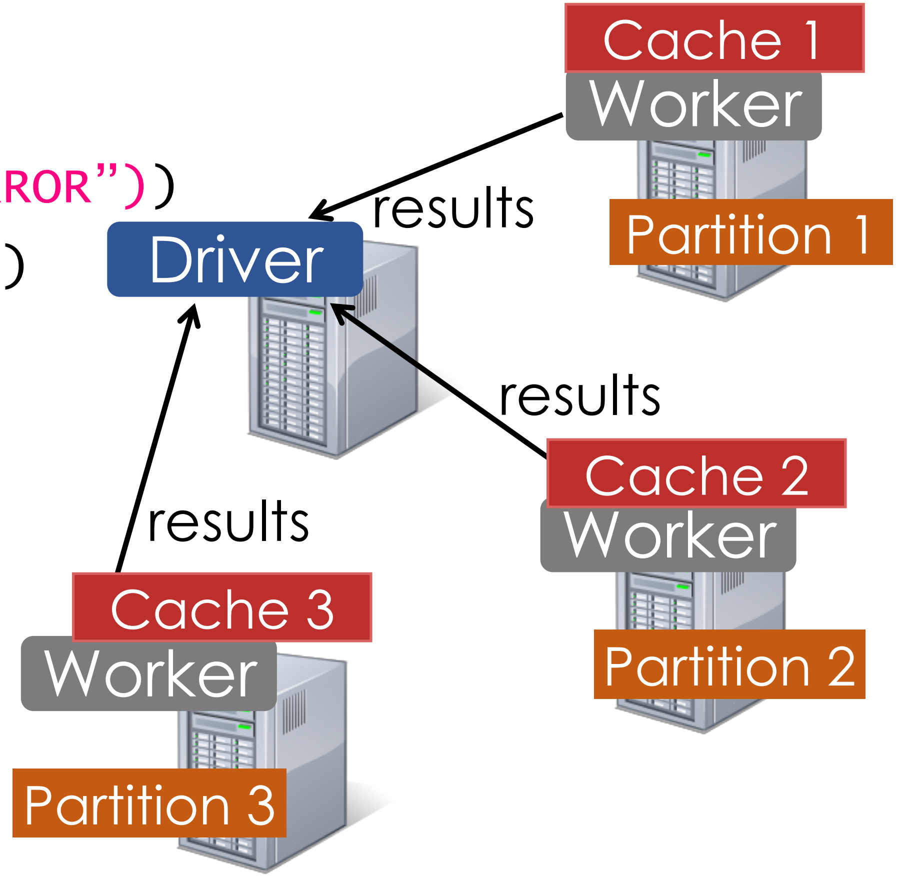


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
```

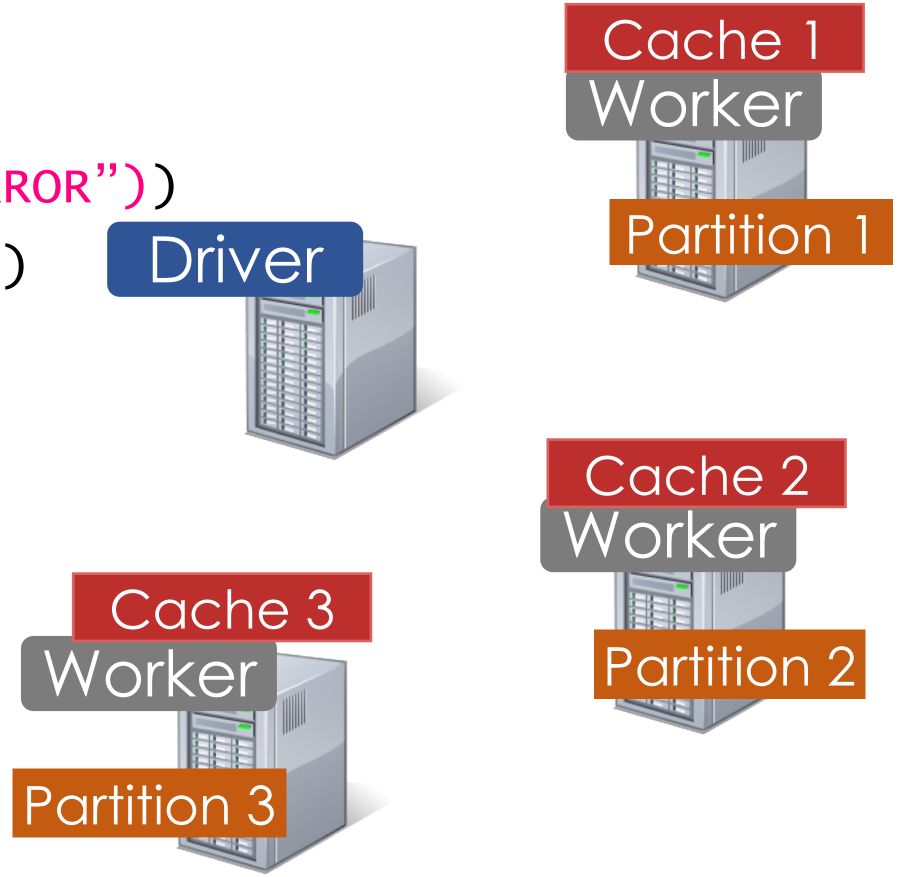


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```

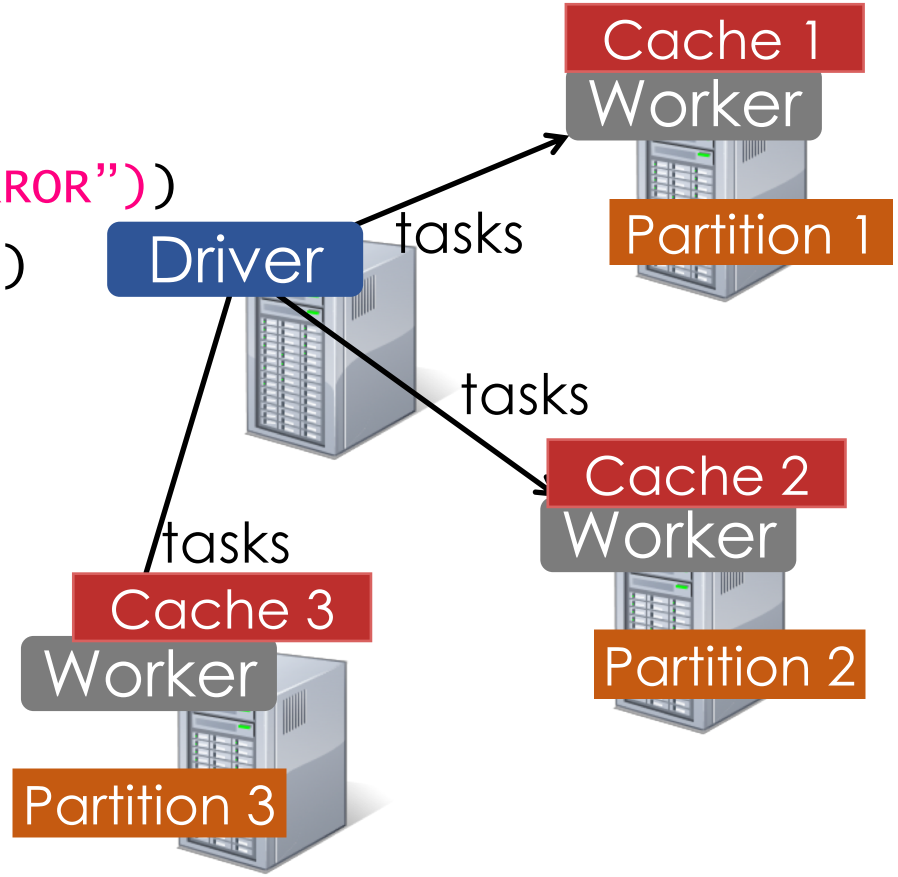


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```

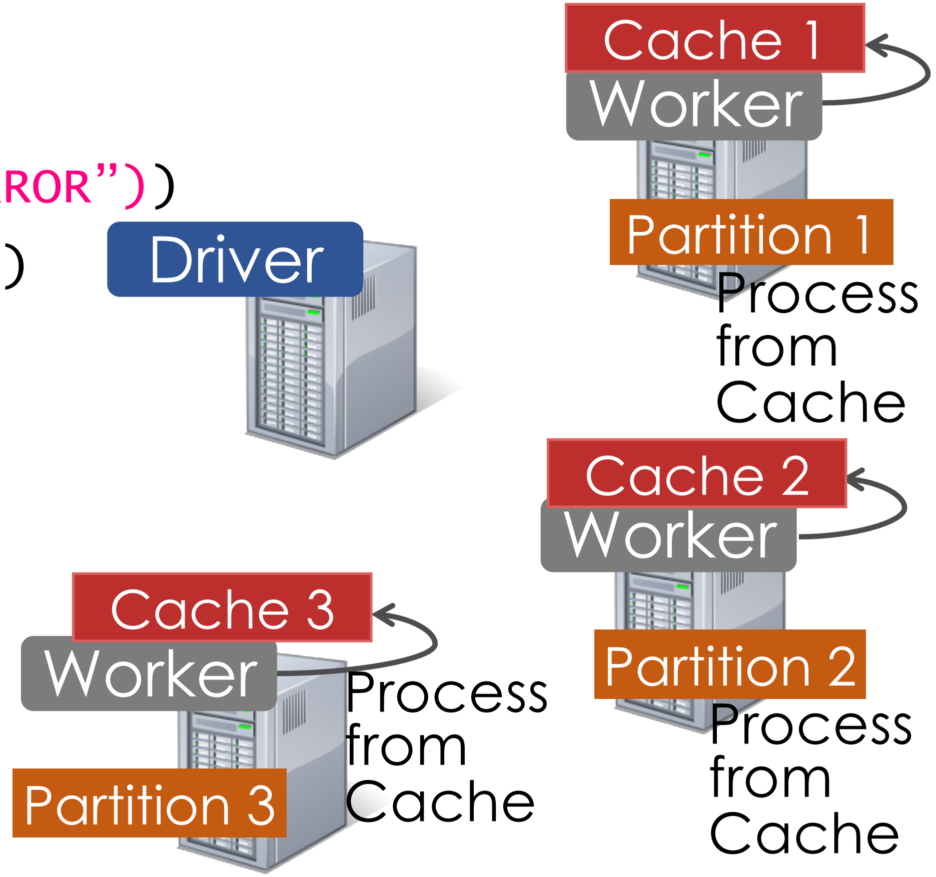


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```

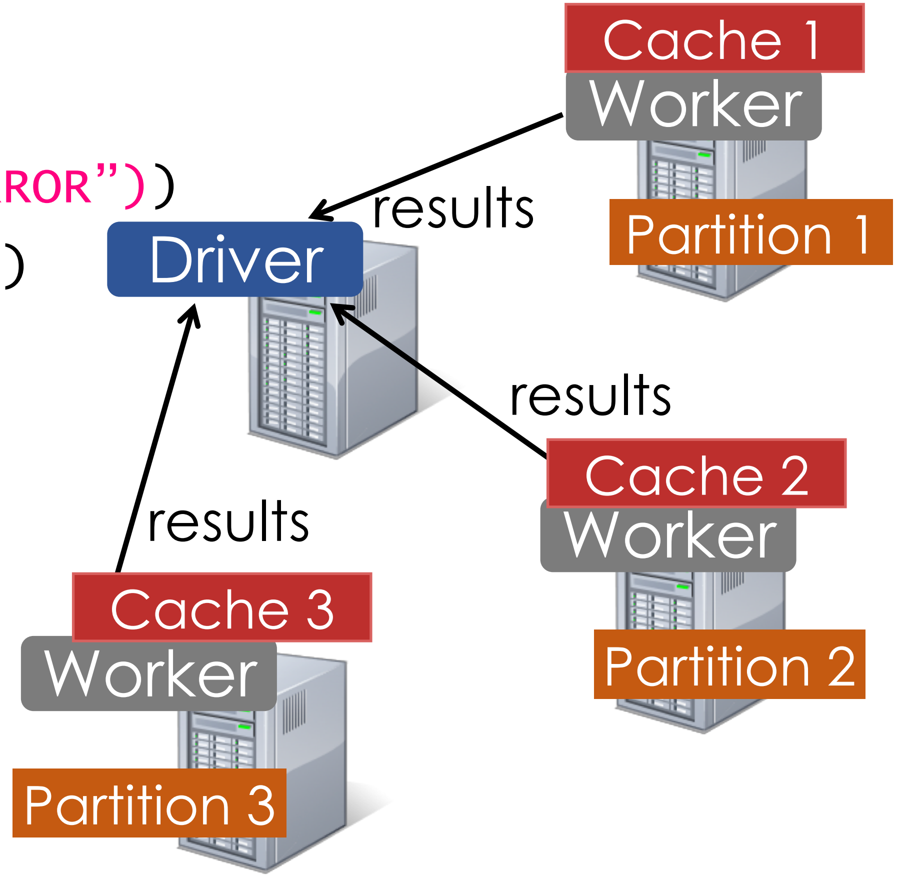


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```

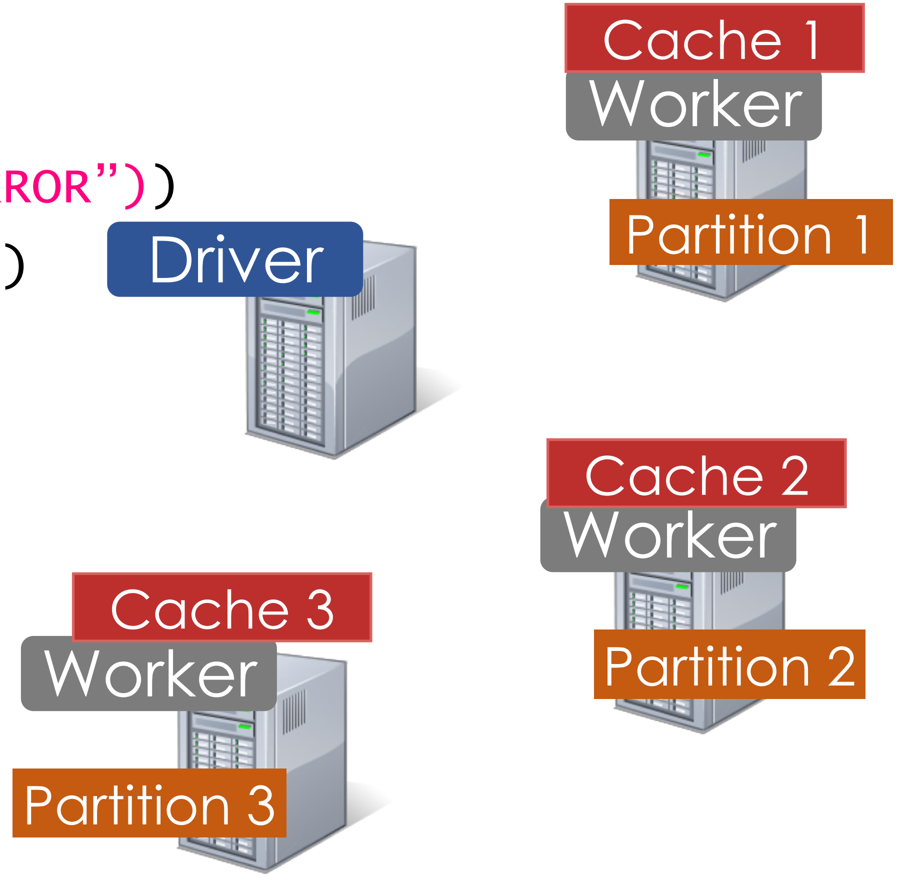


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```



Cache your data → Faster Results

Full-text search of Wikipedia

- 60GB on 20 EC2 machines
- 0.5 sec from mem vs. 20s for on-disk

Example: Counting Words

```
moby_dick = spark.textFile("hdfs://books/mobydick.txt")
lines = moby_dick.flatMap(lambda line: line.split(" "))
counts = lines.map(lambda word: (word, 1))
              .reduceByKey(lambda word: (word, 1))

counts.toDF().toPandas()
```

Much simpler than Hadoop map reduce code!

- The `flatMap` and `map` calls produce transformed RDDs.
  - Computation is lazy! Nothing happens until we get to our action.
- The `reduceByKey` calls kicks off the actual computing.

Abstraction: *Dataflow Operators*

`map`	`reduce`	`sample`
`filter`	`count`	`take`
`groupBy`	`fold`	`first`
`sort`	`reduceByKey`	`partitionBy`
`union`	`groupByKey`	`mapWith`
`join`	`cogroup`	`pipe`
`leftOuterJoin`	`cross`	`save`
`rightOuterJoin`	`zip`	`...`

Abstraction: *Dataflow Operators*

`map`	`reduce`	`sample`
`filter`	`count`	`take`
`groupBy`	`fold`	`first`
`sort`	`reduceByKey`	`partitionBy`
`union`	`groupByKey`	`mapWith`
`join`	`cogroup`	`pipe`
`leftOuterJoin`	`cross`	`save`
`rightOuterJoin`	`zip`	`...`

Spark Demo

Summary (1/2)

- ETL is used to bring data from operational data stores into a data warehouse.
  - Many ways to organize tabular data warehouse, e.g. star and snowflake schemas.
- Online Analytics Processing (OLAP) techniques let us analyze data in data warehouse.
  - Examples: Pivot table, CUBE, slice, dice, rollup, drill down.
- Unstructured data is hard to store in a tabular format in a way that is amenable to standard techniques, e.g. finding pictures of cats.
  - Resulting new paradigm: The Data Lake.

Summary (2/2)

- Data Lake is enabled by two key ideas:
  - Distributed file storage.
  - Distributed computation.
- Distributed file storage involves replication of data.
  - Better speed and reliability, but more costly.
- Distributed computation made easier by map reduce.
  - Hadoop: Open-source implementation of distributed file storage and computation.
  - Spark: Typically faster and easier to use than Hadoop.