Big Data Analytics

Map-Reduce and Spark

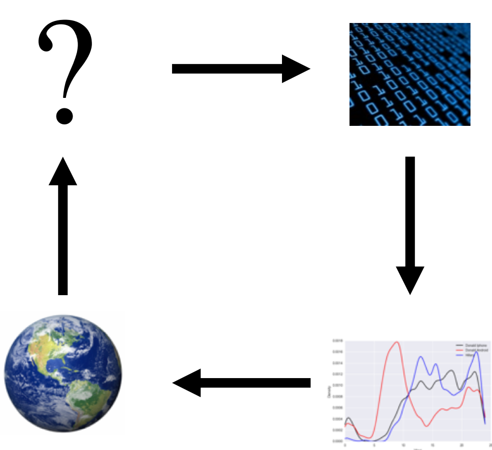
Slides by:

Joseph E. Gonzalez

jegonzal@cs.berkeley.edu

Guest Lecturer:

Vikram Sreekanti



From SQL to Big Data (with SQL)

- A few weeks ago…
  - 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

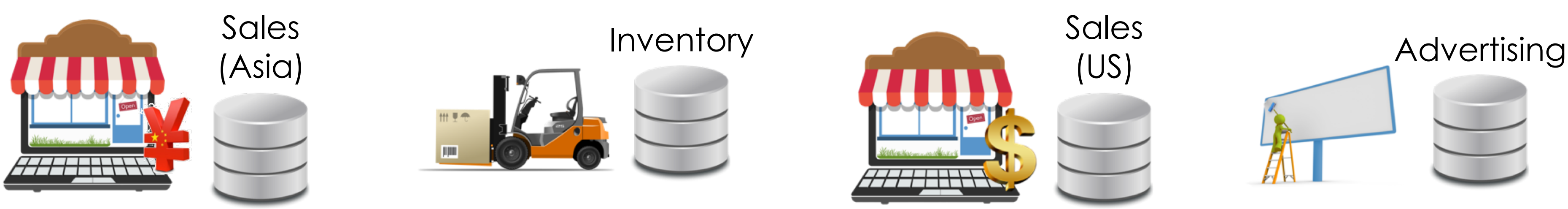
Data in the Organization

A little bit of buzzword bingo!

Inventory

How we like to think of data in the organization

The reality…

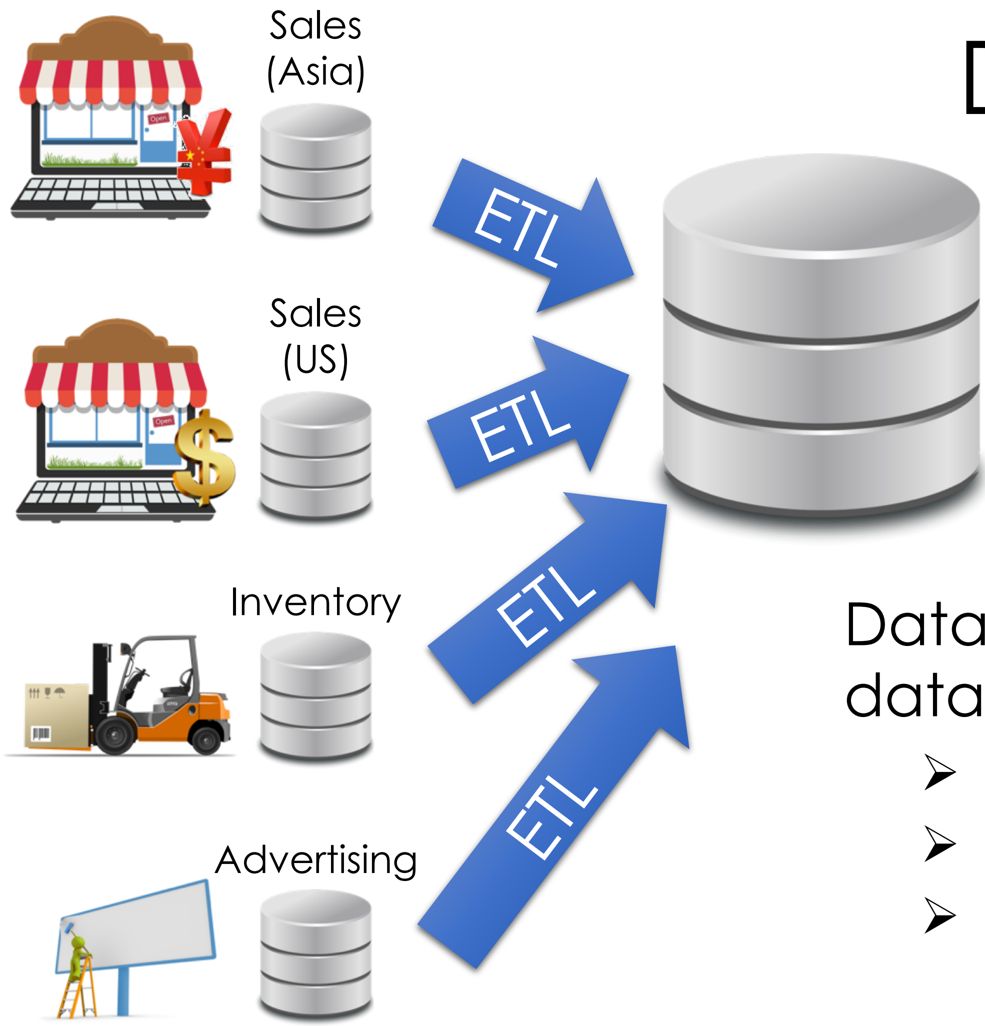


Operational Data Stores

- 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

Extract → Transform → Load (ETL)

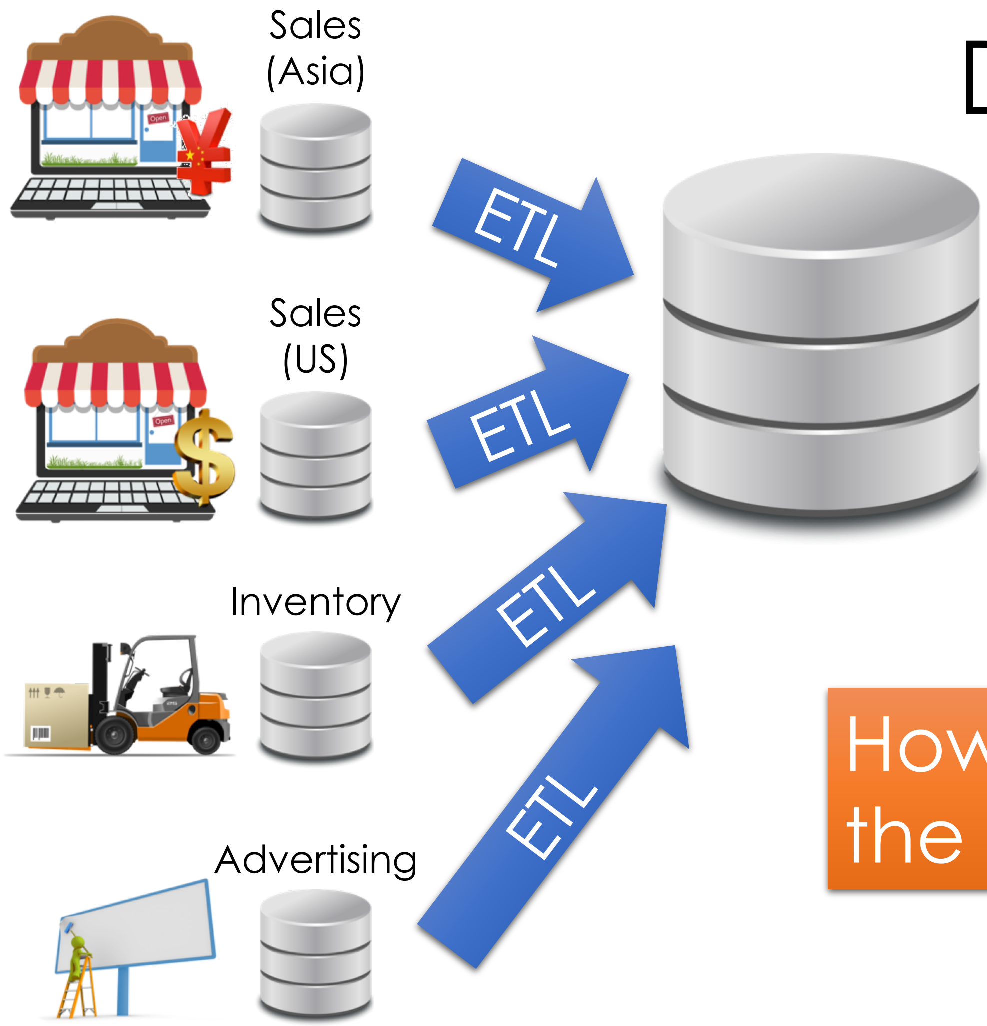
Extract & Load: provides a snapshot of operational data

- 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	[illegible]	[illegible]	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

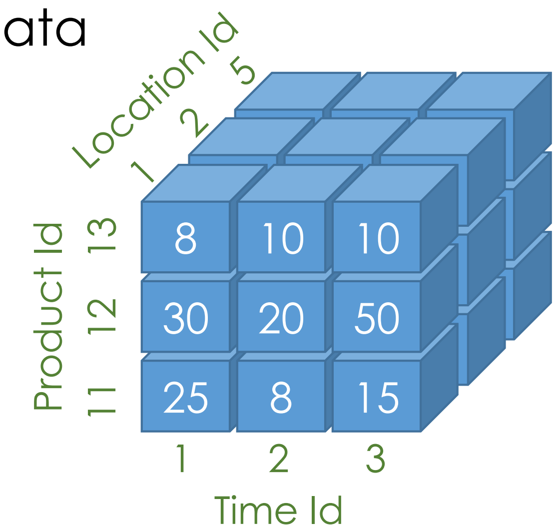
Products

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



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

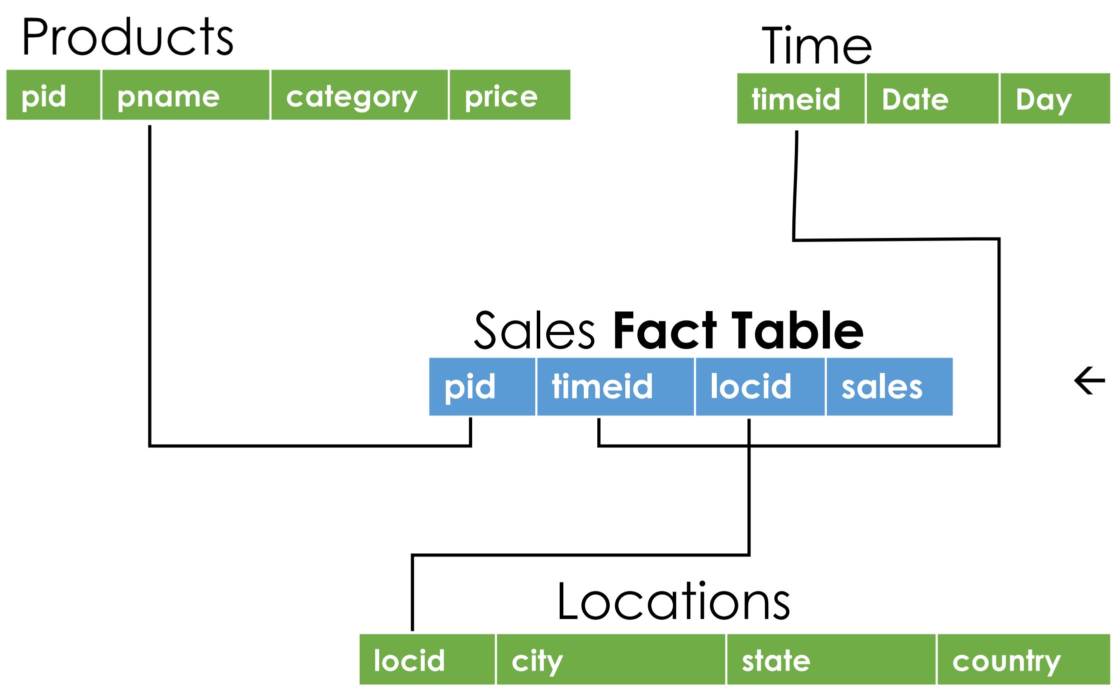
pid	pname	category	price
11	Corn	Food	25
12	Galaxy 1	Phones	18
13	Peanuts	Food	2

Time

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

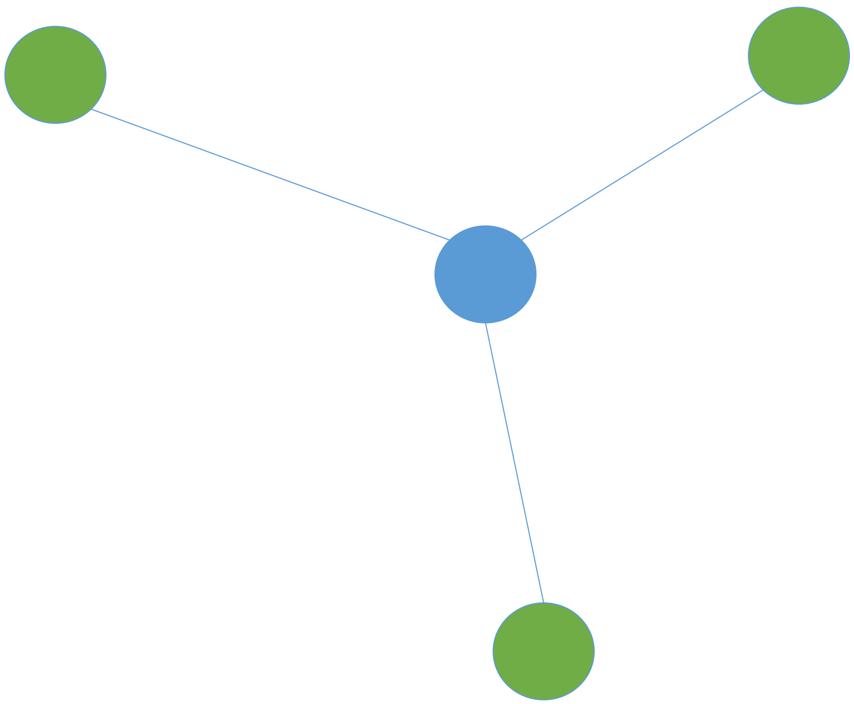
- Fact Table
  - Minimizes redundant info
  - Reduces data errors
- Dimensions
  - Easy to manage and summarize
  - Rename: Galaxy1 → Phablet
- Normalized Representation
- How do we do analysis?
  - **Joins!**

The Star Schema



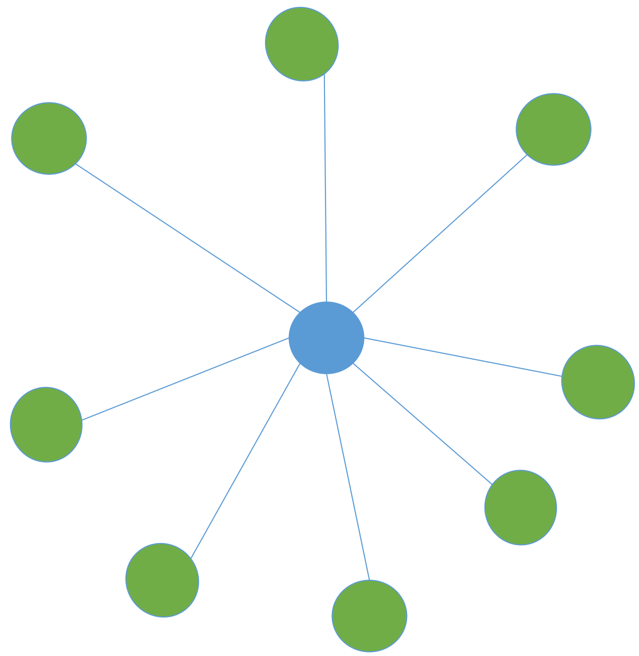
← This looks like a star …

The Star Schema



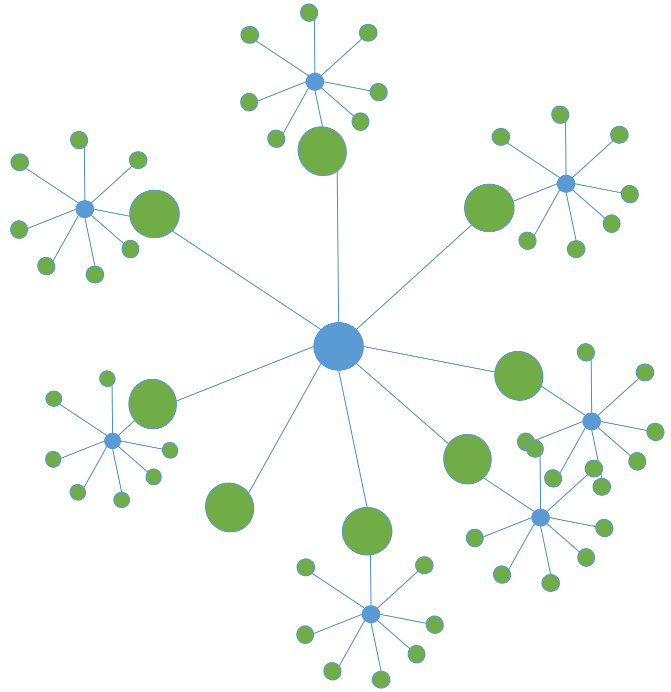
← This looks like a star …

The Star Schema



← This looks like a star …

The Snowflake Schema

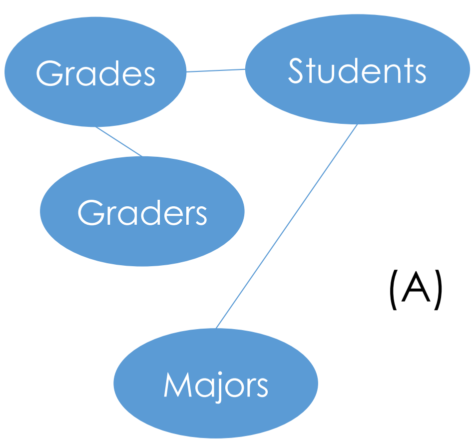


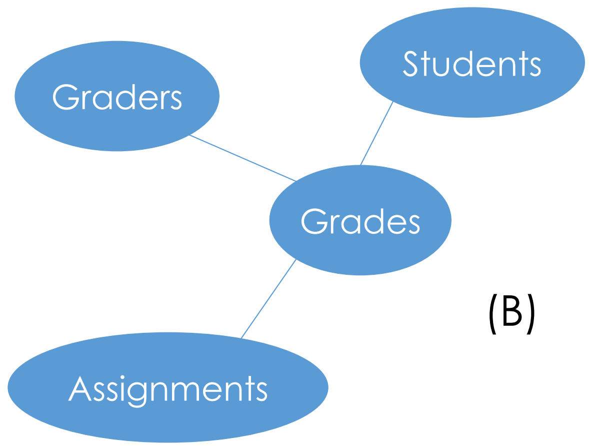
← This looks like a snowflake …?

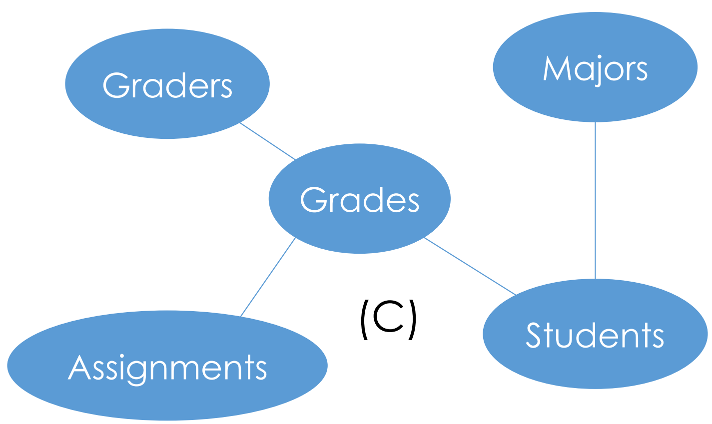
Which schema illustration would best organize this data?

http://bit.ly/ds100-sp18-star

```
Data(calid, student_name, year,
     major, major_grade_req,
     asg1_name, asg1_pts, asg1_score, asg1_grader_name
     asg2_name, asg2_pts, asg2_score, asg2_grader_name
     avg_grade)
```

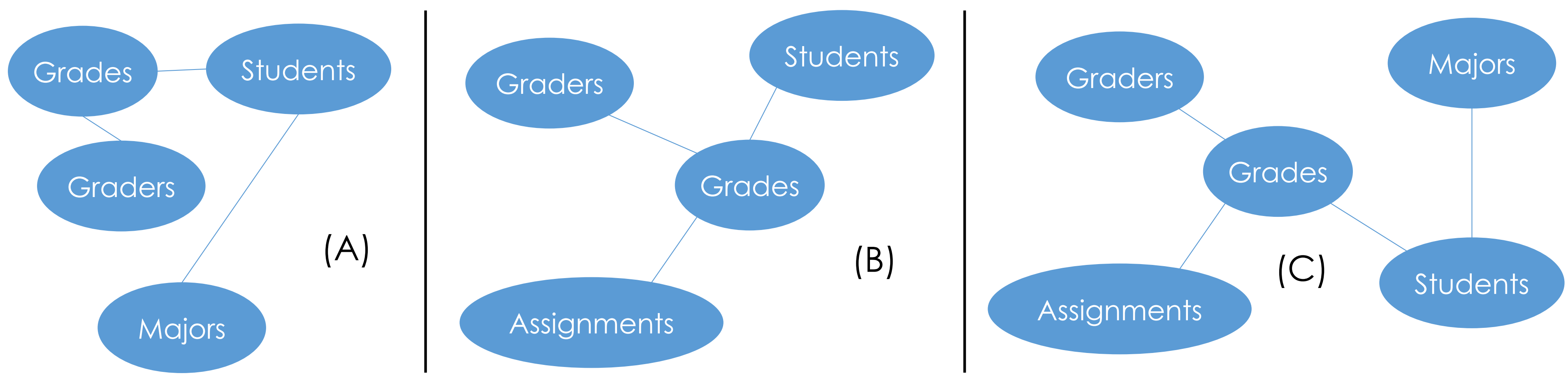






```
Data(calid, student_name, year,
     major, major_grade_req,
     asg1_name, asg1_pts, asg1_score, asg1_grader_name
     asg2_name, asg2_pts, asg2_score, asg2_grader_name
     avg_grade)
```

```
Grades(calid, asg_name, grader_id, score)
Graders(grader_id, grader_name)
Student(calid, name, year, major_name, avg_grade)
Majors(major_name, grade_req)
Assignments(asg_name, asg_pts)
```



Online Analytics Processing (**OLAP**)

Users interact with multidimensional data:

- Constructing ad-hoc and often complex SQL queries
- Using graphical tools that to construct queries
- Sharing views that summarize data across important dimensions

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?

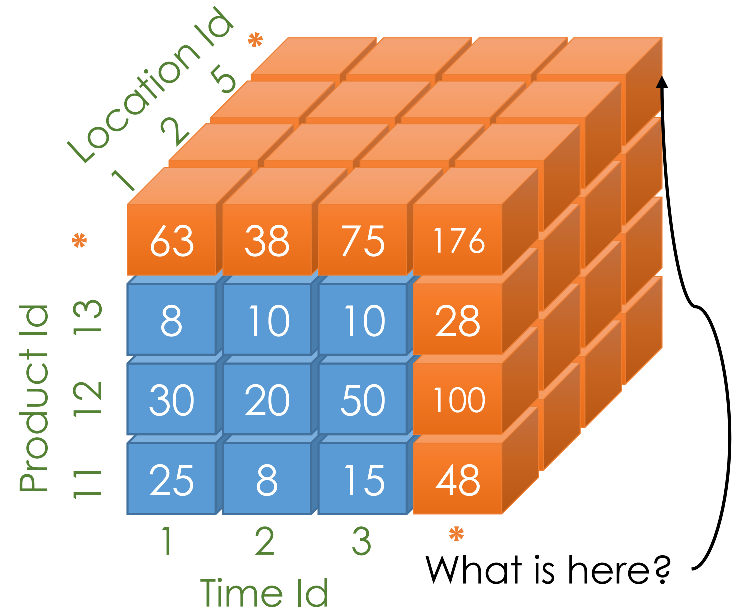
Cube Operator

- ➢ Generalizes cross-tabulation to higher dimensions.

➢In SQL:

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

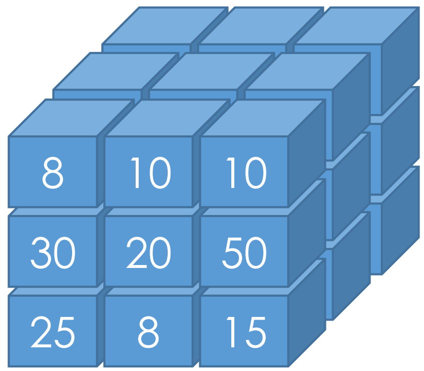
Item	Color	Quantity
Desk	Blue	2
Desk	Red	3
Sofa	Blue	4
Sofa	Red	5

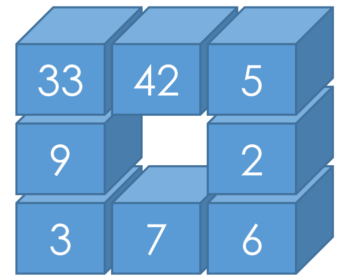


Item	Color	QtySum
Desk	Blue	2
Desk	Red	3
Desk	*	5
Sofa	Blue	4
Sofa	Red	5
Sofa	*	9
*	*	14
*	Blue	6
*	Red	8

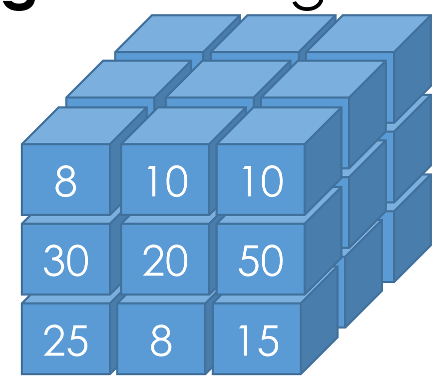
OLAP Queries

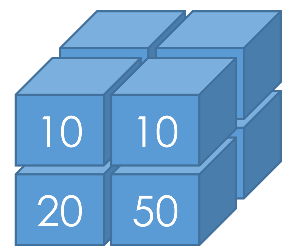
- **Slicing:** *selecting a value for a dimension*





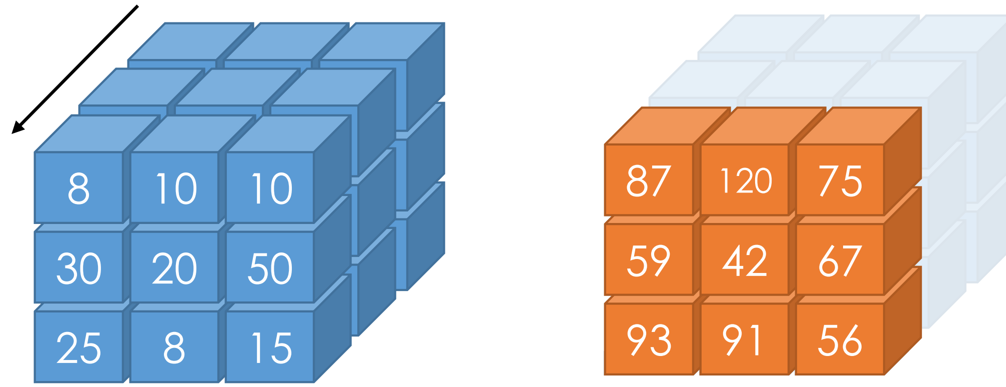
- **Dicing:** *selecting a range of values in multiple dimension*



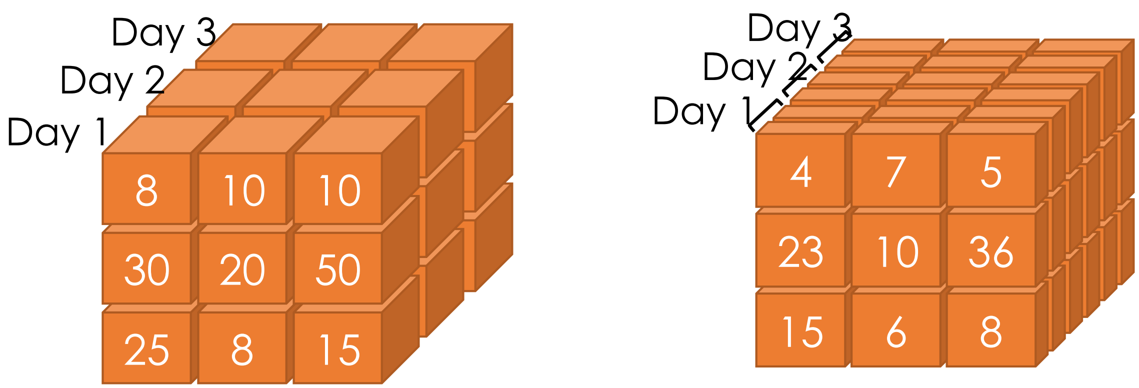


OLAP Queries

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

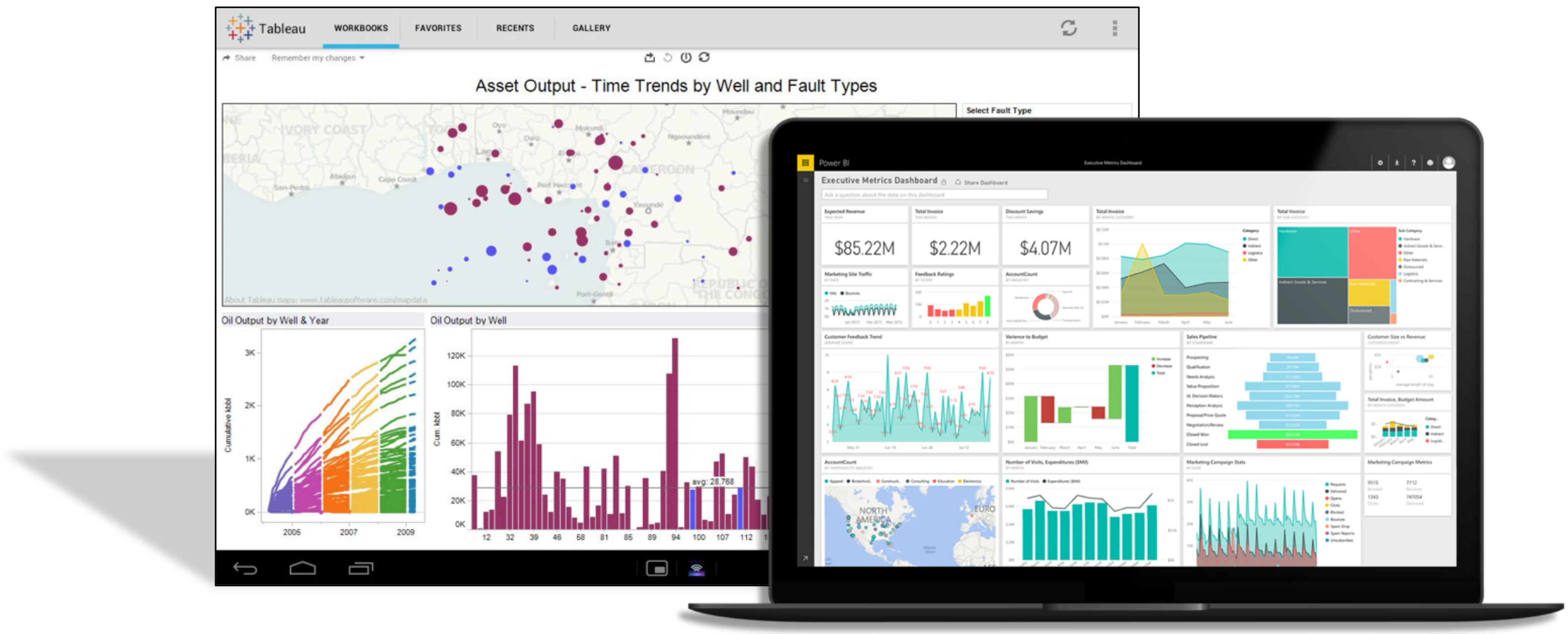


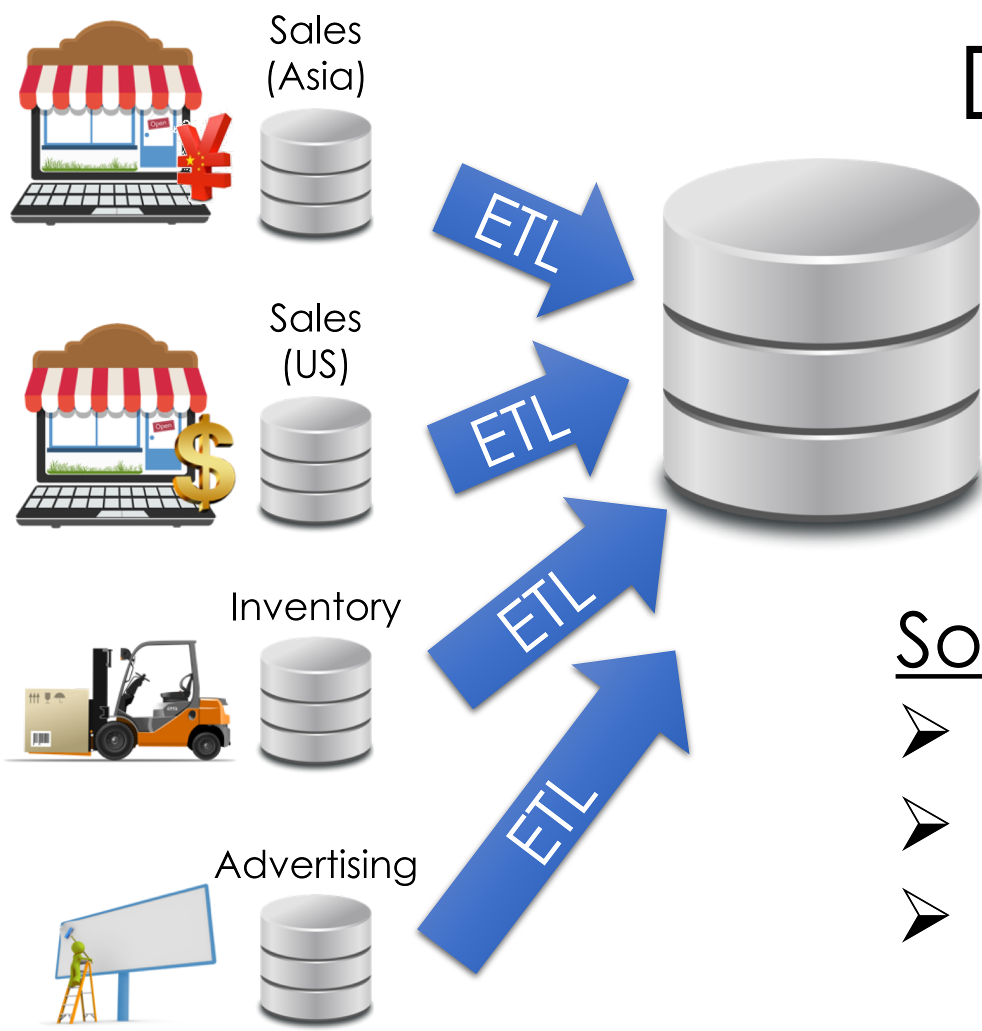
- **Drill-Down:** *de-aggregating along a dimension*



Reporting and Business Intelligence (BI)

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





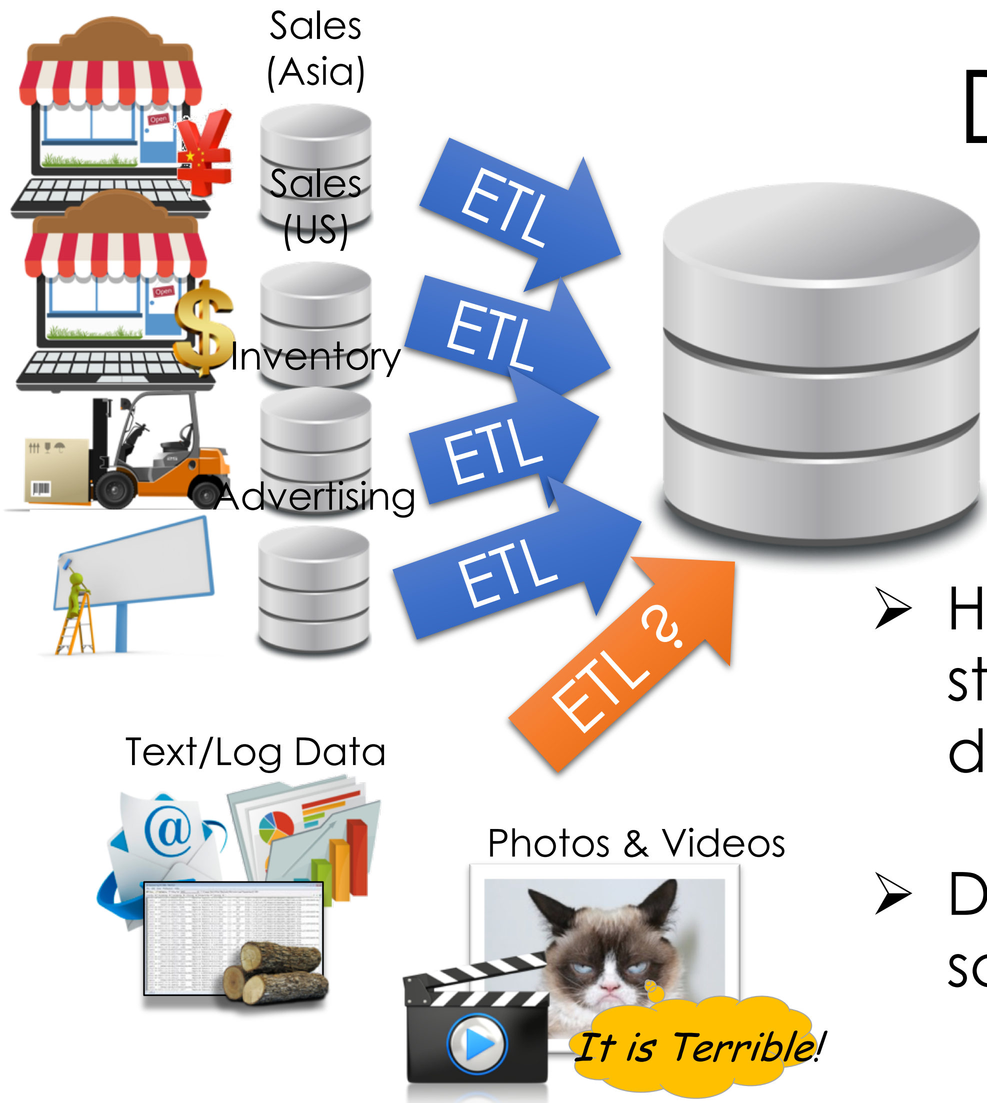
Data Warehouse

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

Collects and organizes historical data from multiple sources

How do we **clean** and **organize** this data?

Depends on use …

Text/Log Data

Photos & Videos

How do we **load** and **process** this data in a relational system?

Depends on use…
Can be difficult...
Requires thought...

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 & catalog managers
- Organizations are evolving
  - Tracking data usage and file permissions
  - New job title: data engineers

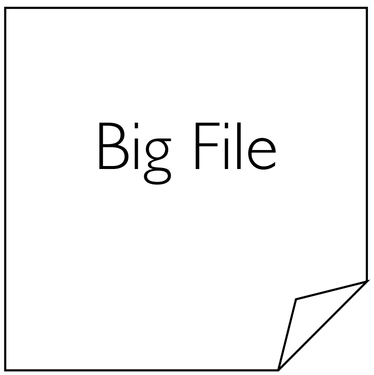
How do we **store** and **compute** on large unstructured datasets

- 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

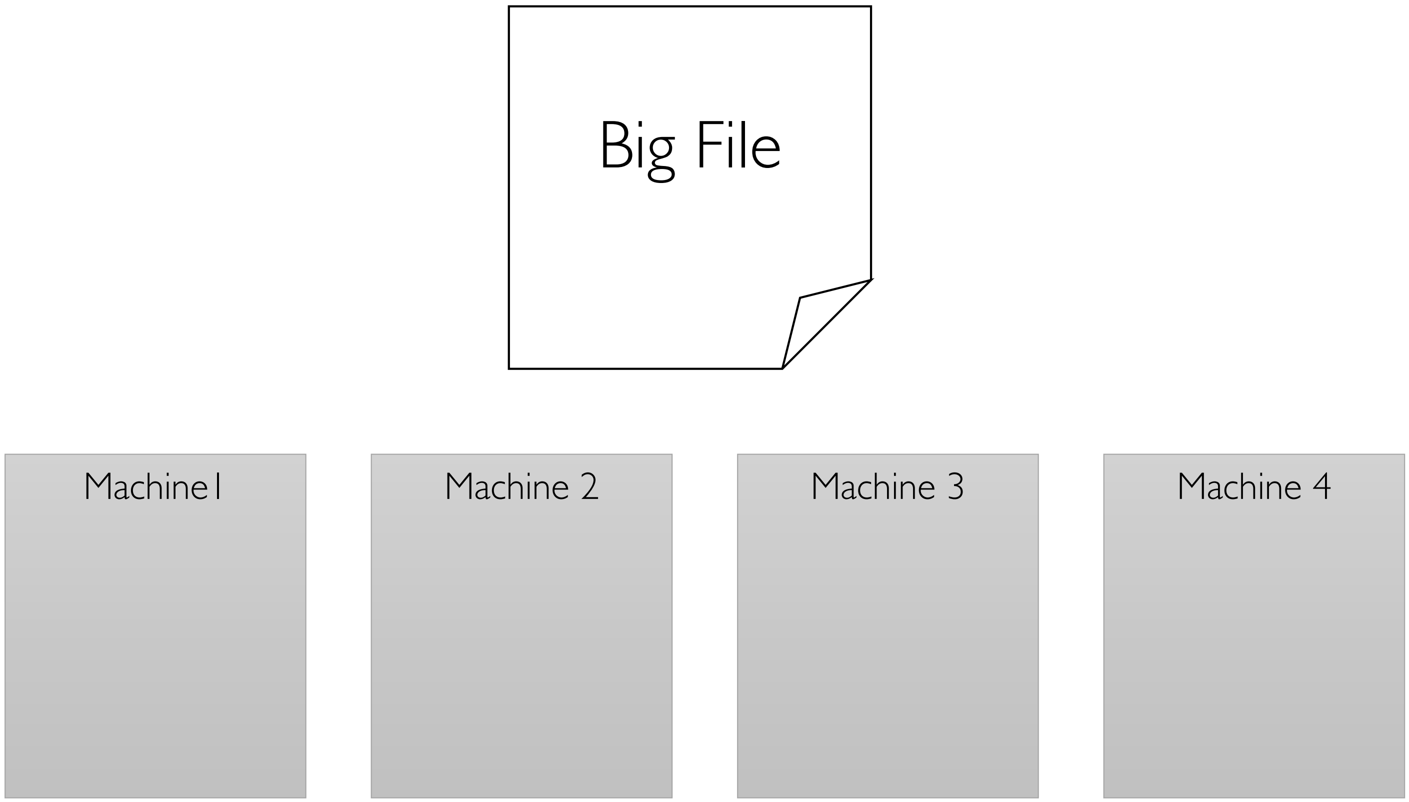
Fault Tolerant Distributed File Systems



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

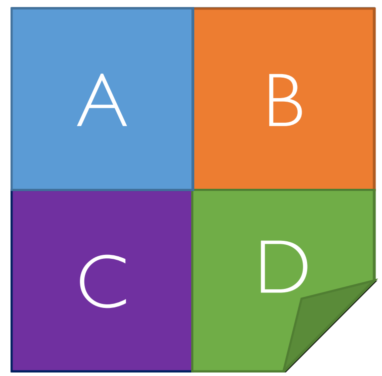
[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems

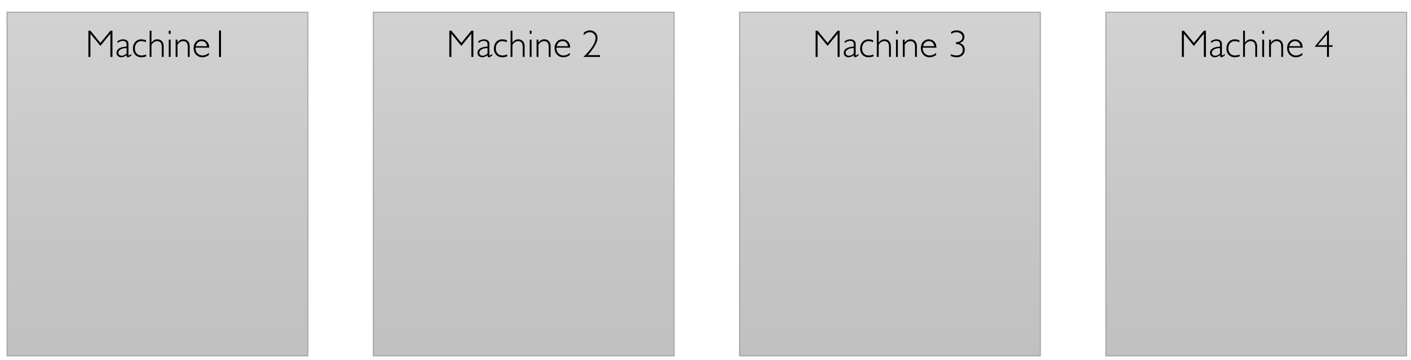


[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems

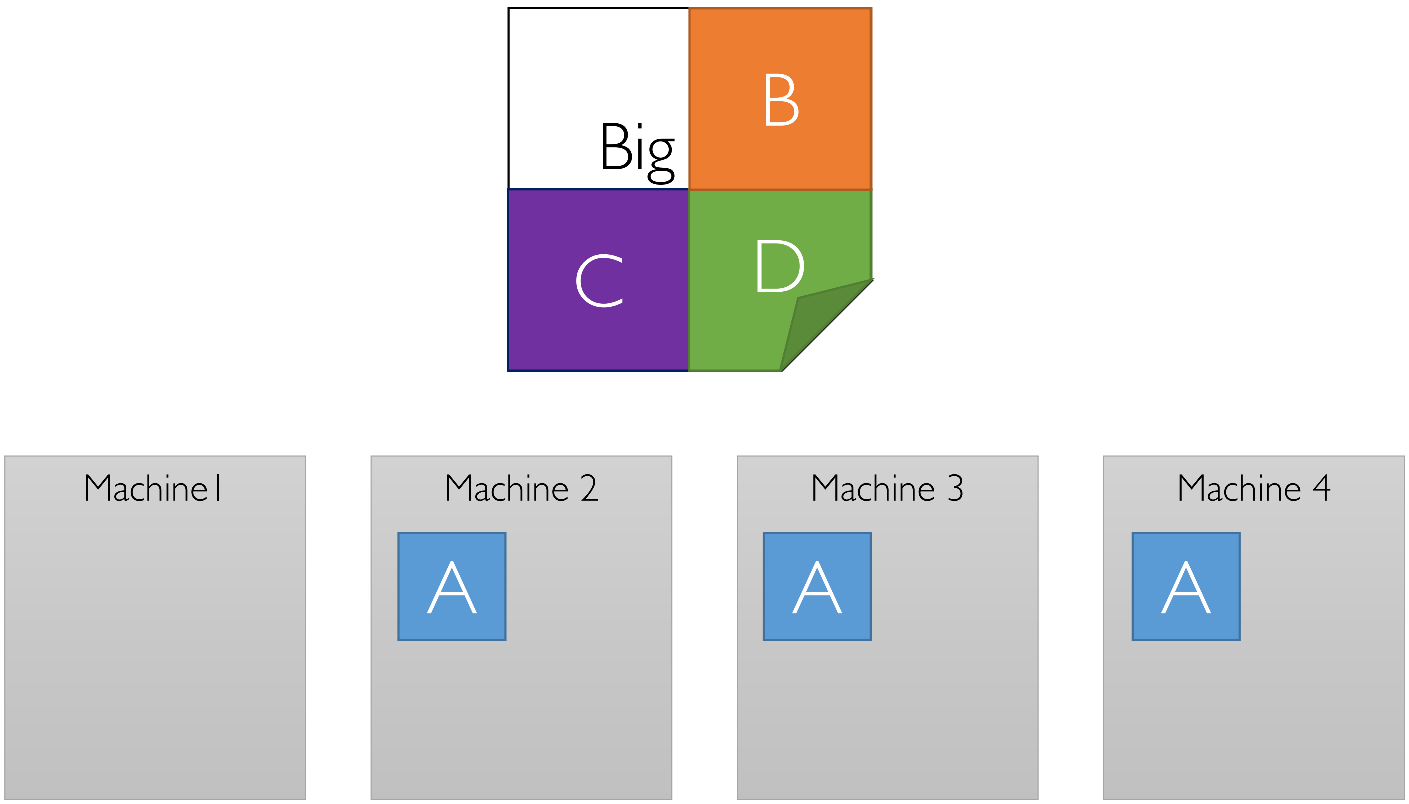


- Split the file into smaller parts.
- How?
  - Ideally at record boundaries
  - What if records are big?



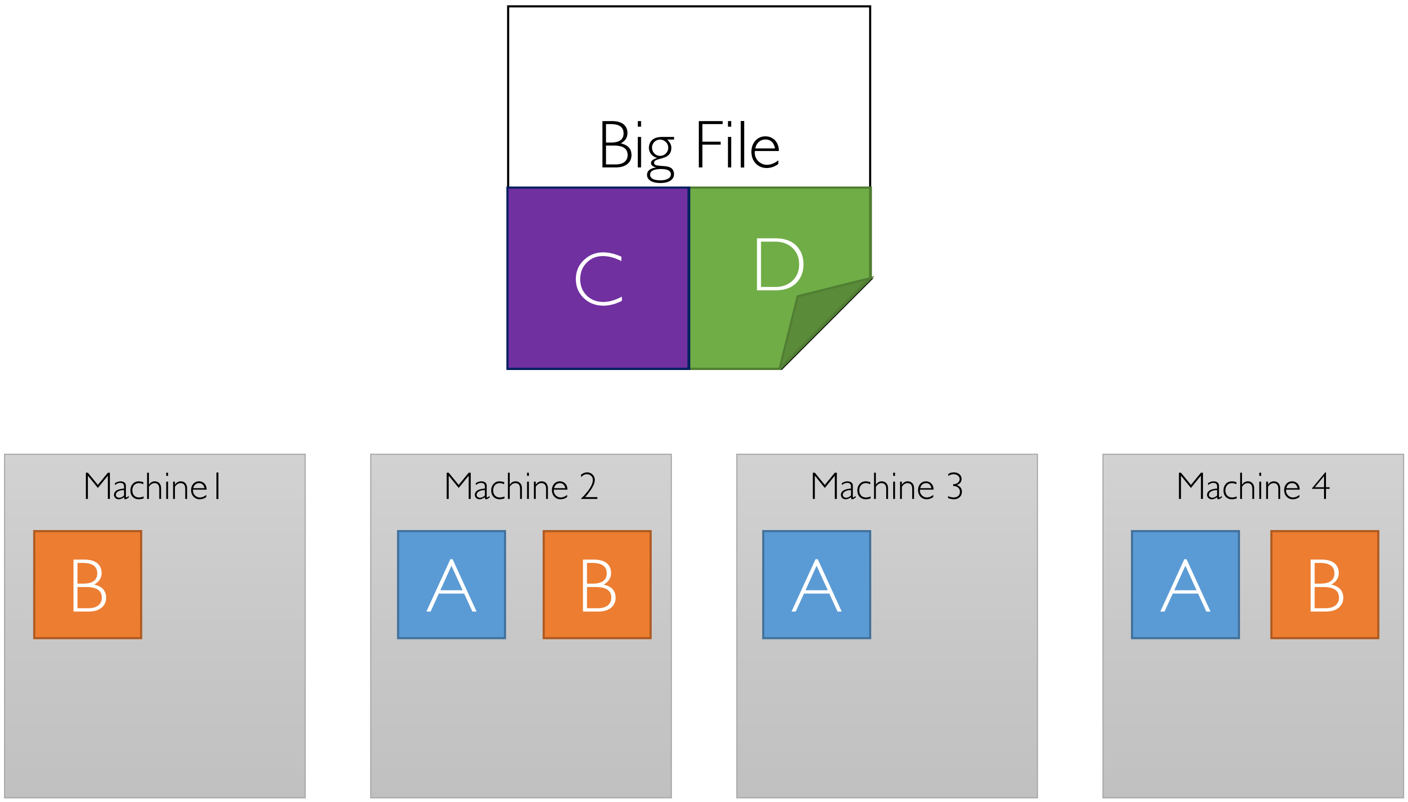
[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems



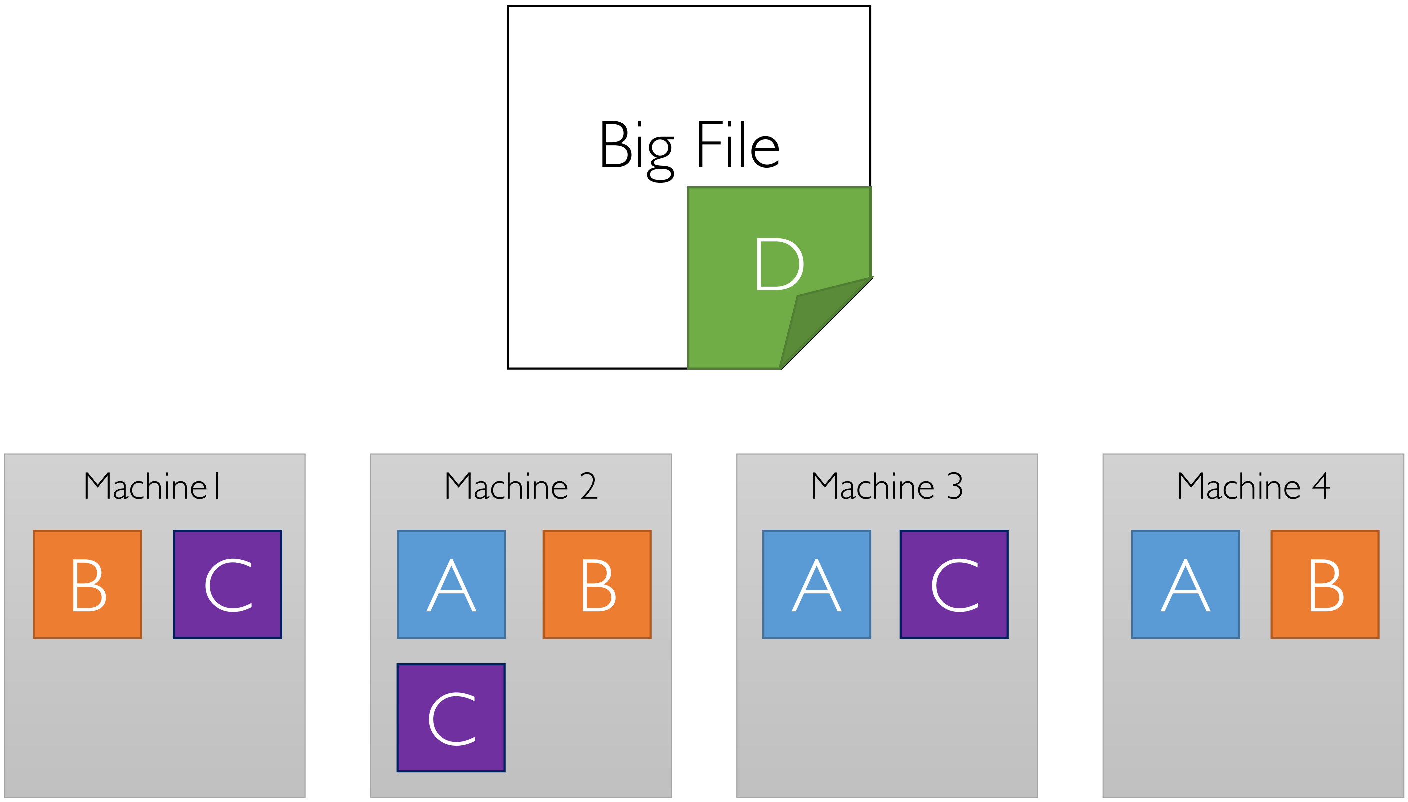
[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems



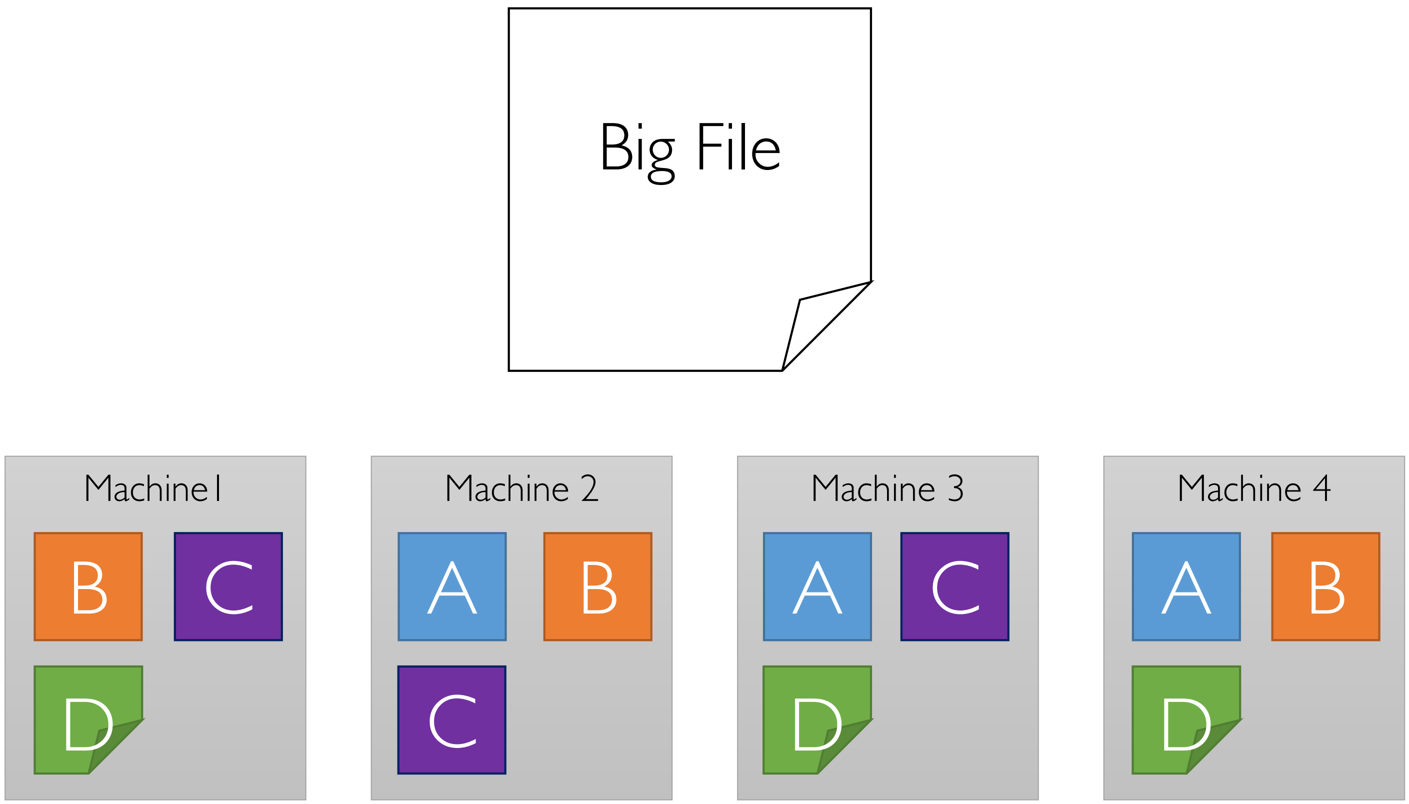
[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems



[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems

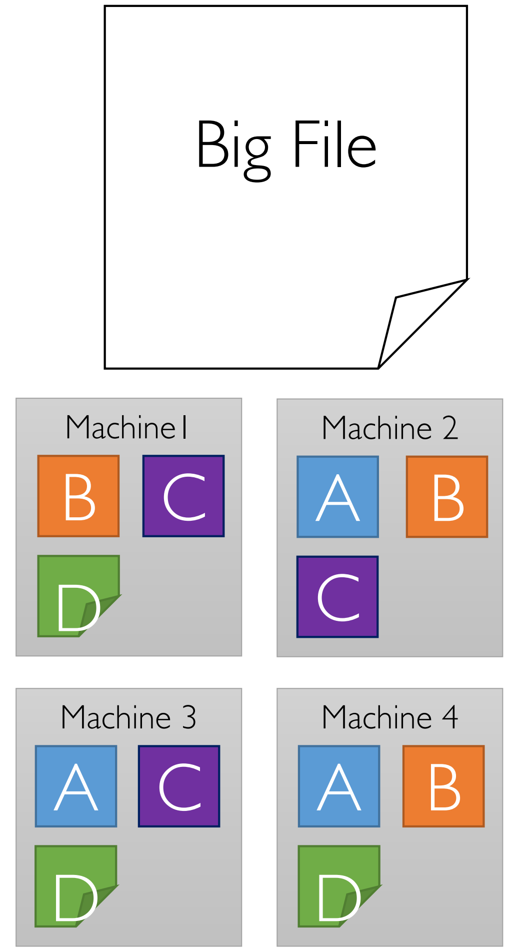


[Ghemawat et al., SOSP'03]

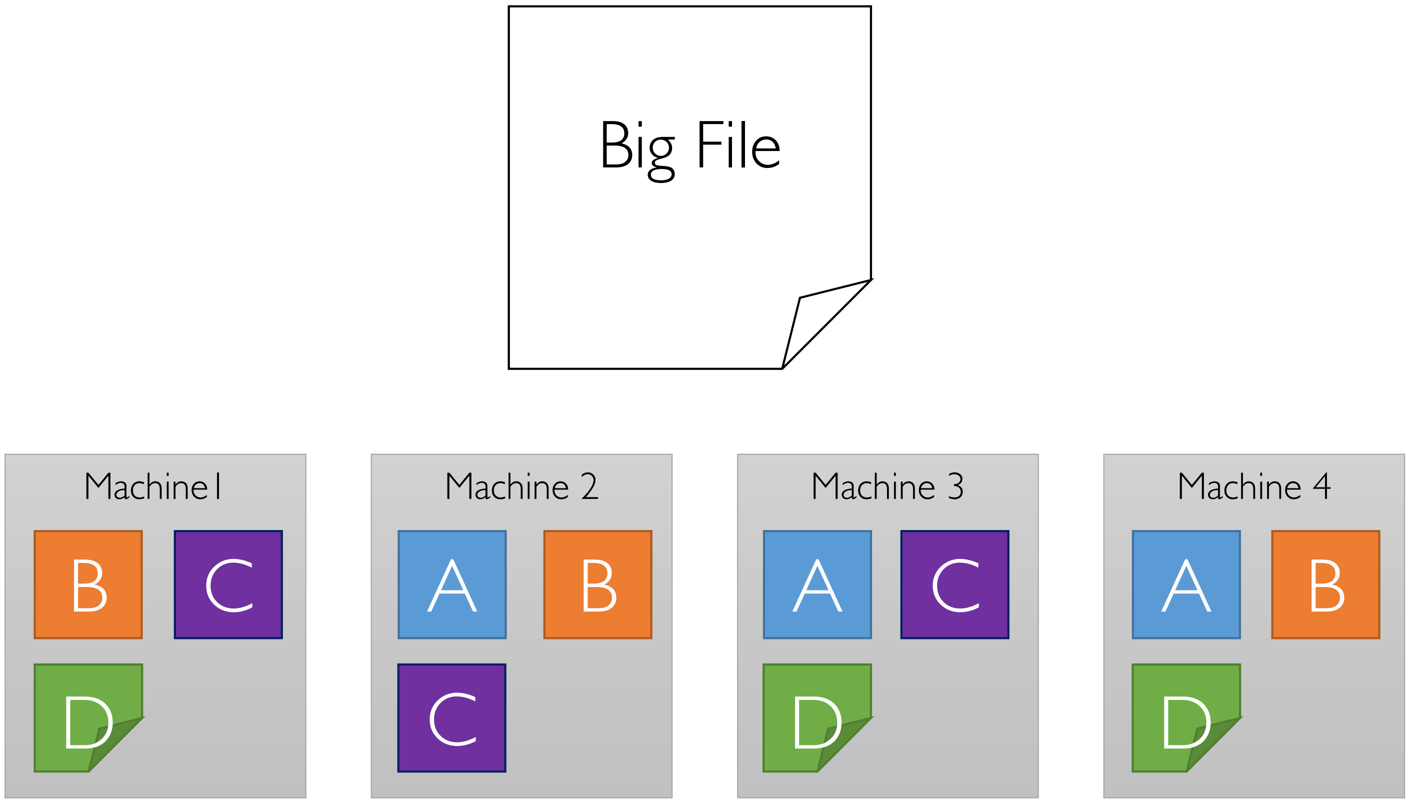
Fault Tolerant Distributed File Systems

- 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!

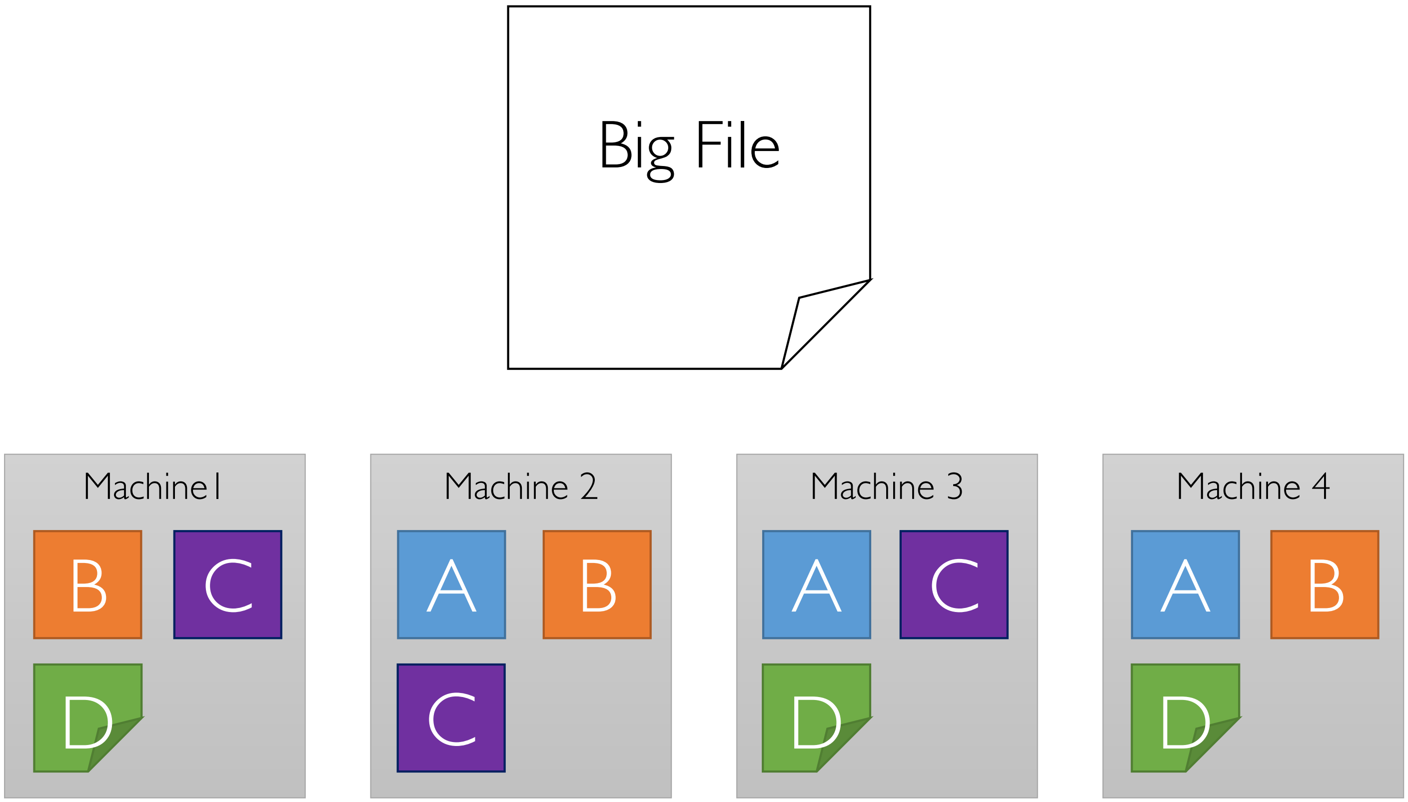


Fault Tolerant Distributed File Systems
Failure Event



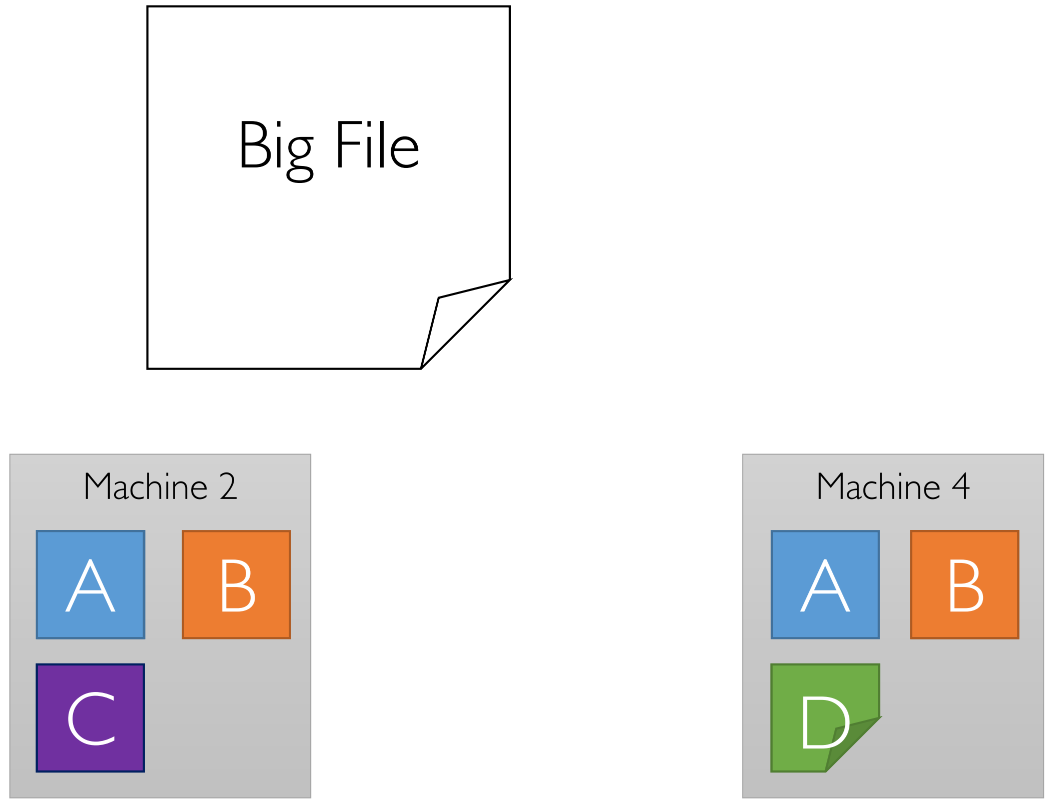
[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems
Failure Event



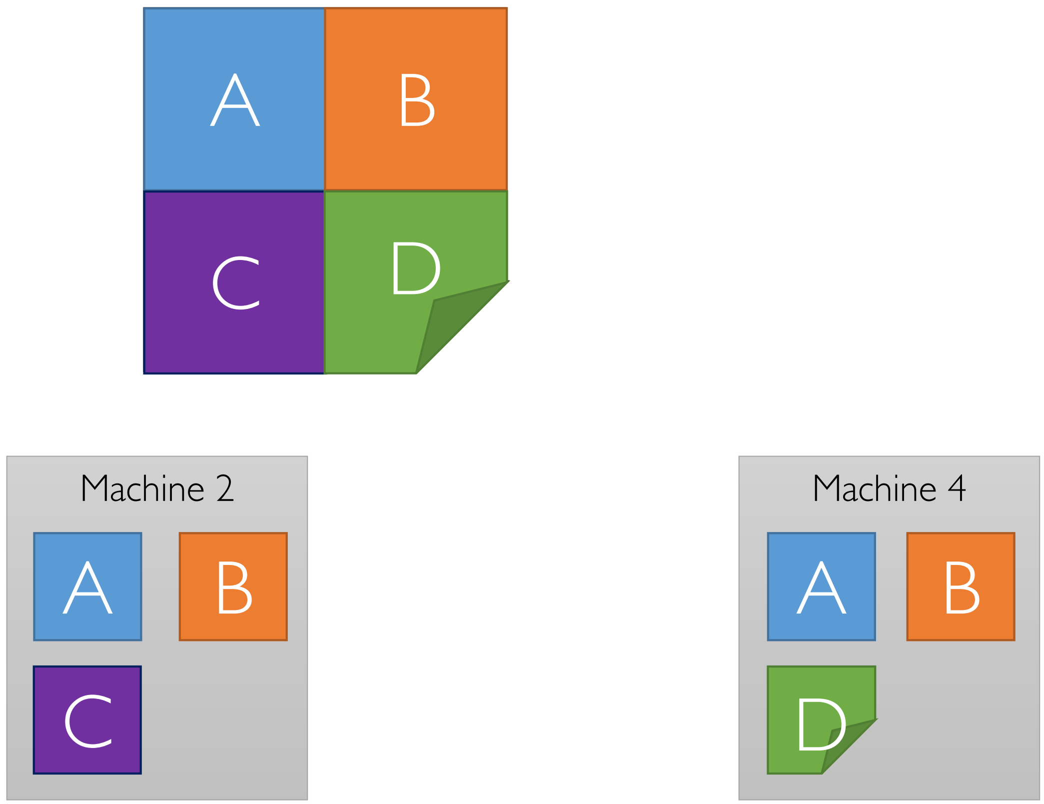
[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems
Failure Event



[Ghemawat et al., SOSP'03]

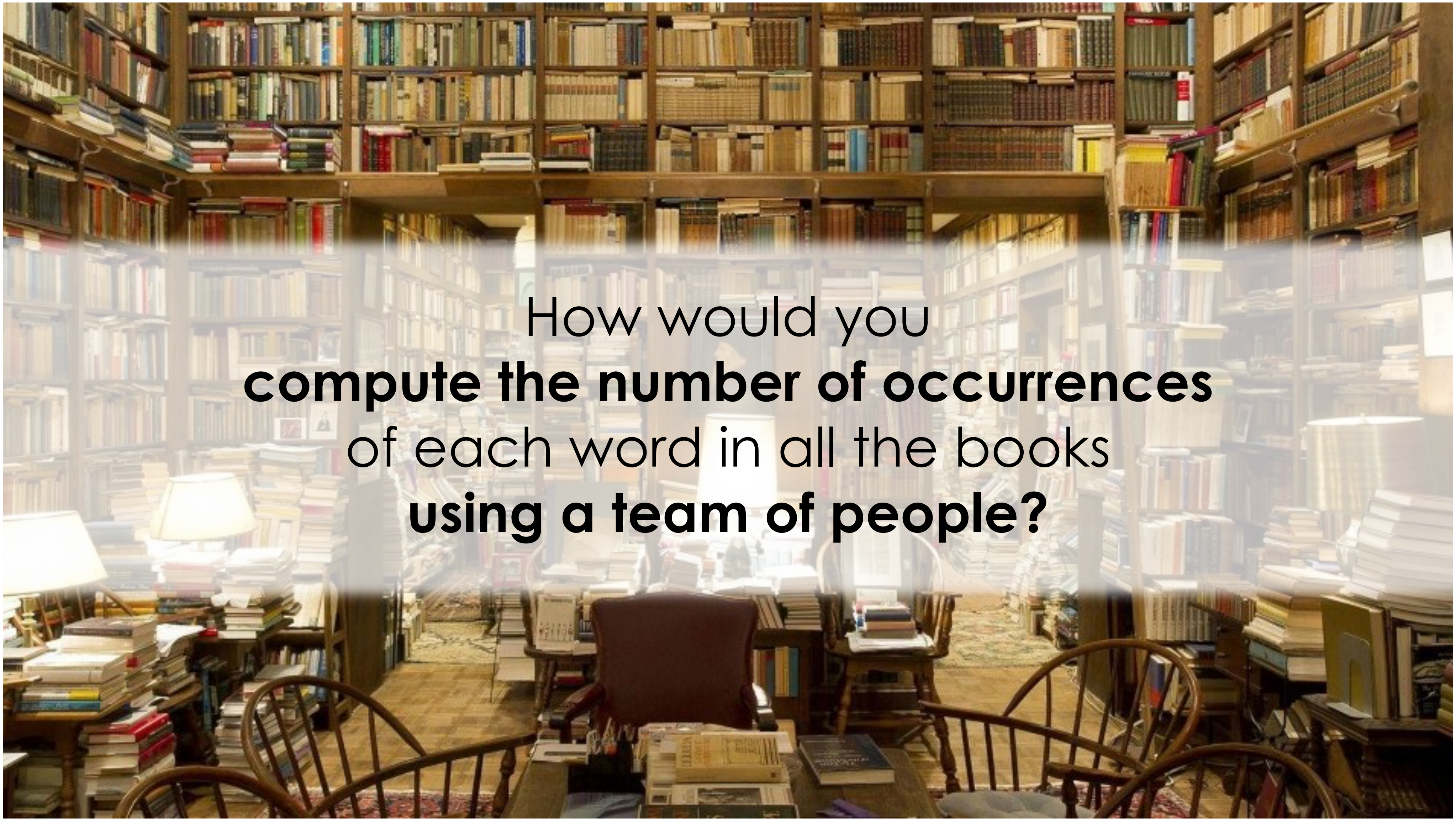
Fault Tolerant Distributed File Systems
Failure Event



[Ghemawat et al., SOSP'03]

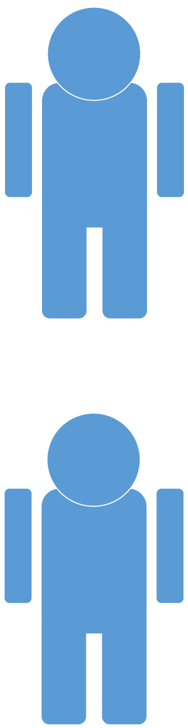
Map-Reduce
Distributed Aggregation

Computing are very large files


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

Simple Solution

Simple Solution

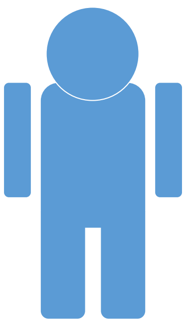


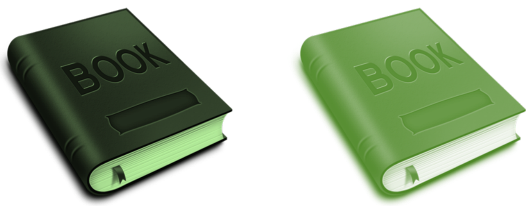
1) Divide Books Across Individuals

Simple Solution

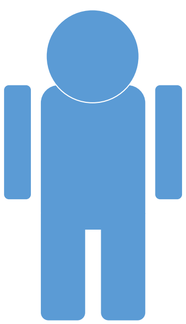
1) Divide Books Across Individuals

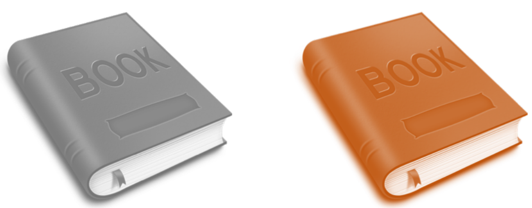
2) Compute Counts Locally





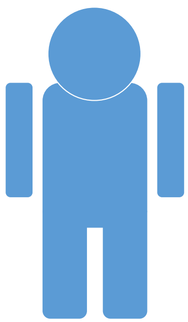
Word	Count
Apple	2
Bird	7
…	



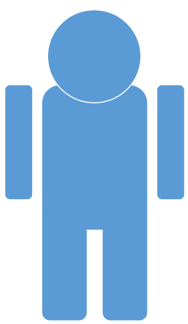


Word	Count
Apple	0
Bird	1
…	

Simple Solution



1) Divide Books Across Individuals



2) Compute Counts Locally

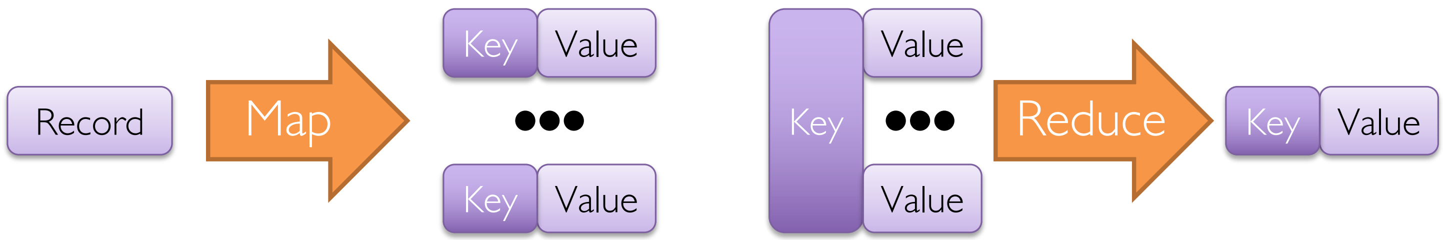
Word	Count
Apple	2
Bird	7
…	

Word	Count
Apple	0
Bird	1
…	

3) Aggregate Tables

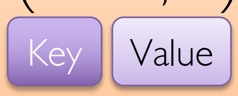
Word	Count
Apple	2
Bird	8
…	

The Map Reduce Abstraction



Example: *Word-Count*

```
Map(docRecord) {
  for (word in docRecord) {
    emit (word, 1)
  }
}
```




```
Reduce(word, counts) {
  emit (word, SUM(counts))
}
```

Map: Deterministic

Reduce: Commutative and Associative

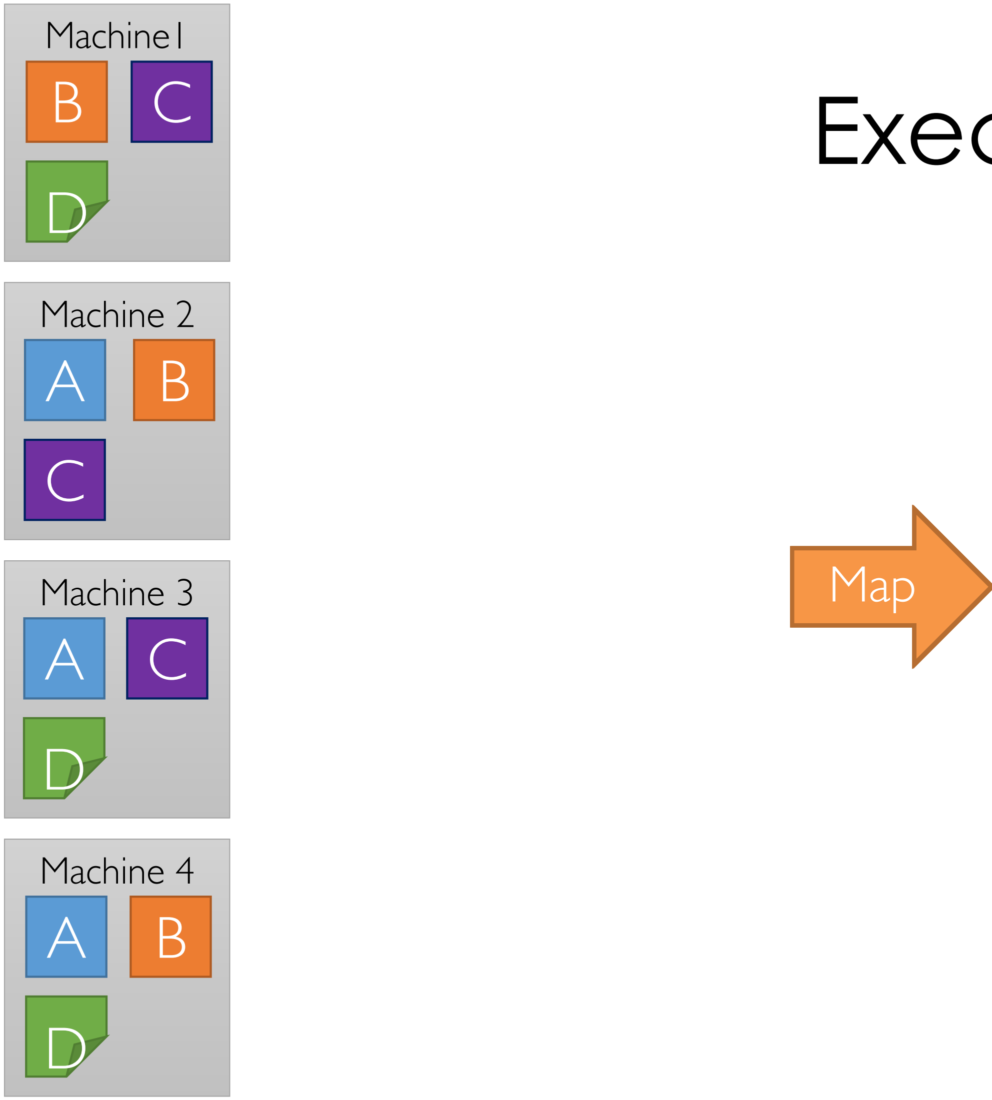
[Dean & Ghemawat, OSDI'04]

Key properties of Map And Reduce

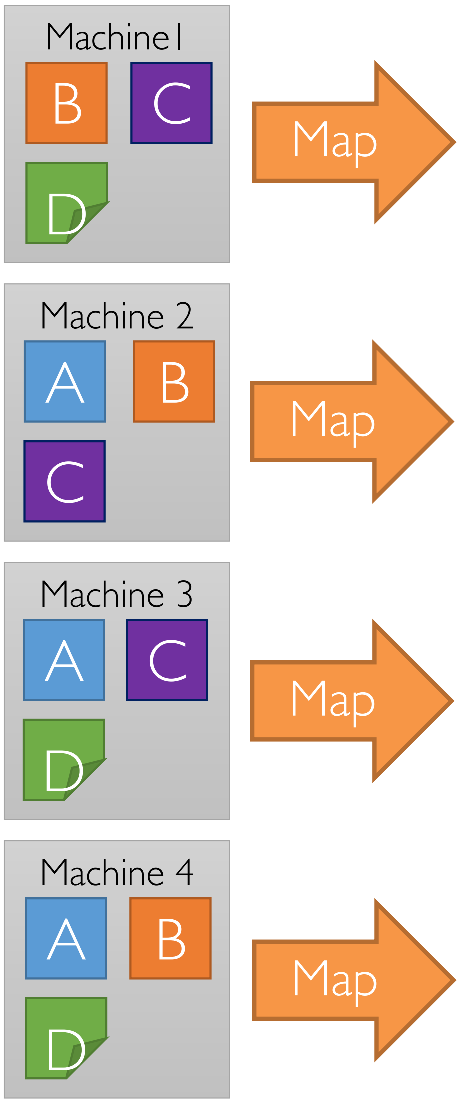
- **Deterministic <u>Map</u>:** allows for re-execution on failure
  - If some computation is lost we can always re-compute
  - Issues with samples?
- **Commutative <u>Reduce</u>:** *allows for re-order of operations*
  - Reduce(A,B) = Reduce(B,A)
  - Example (addition): A + B = B + A
  - Is floating point math commutative?
- **Associative <u>Reduce</u>:** *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))

Executing Map Reduce

Executing Map Reduce



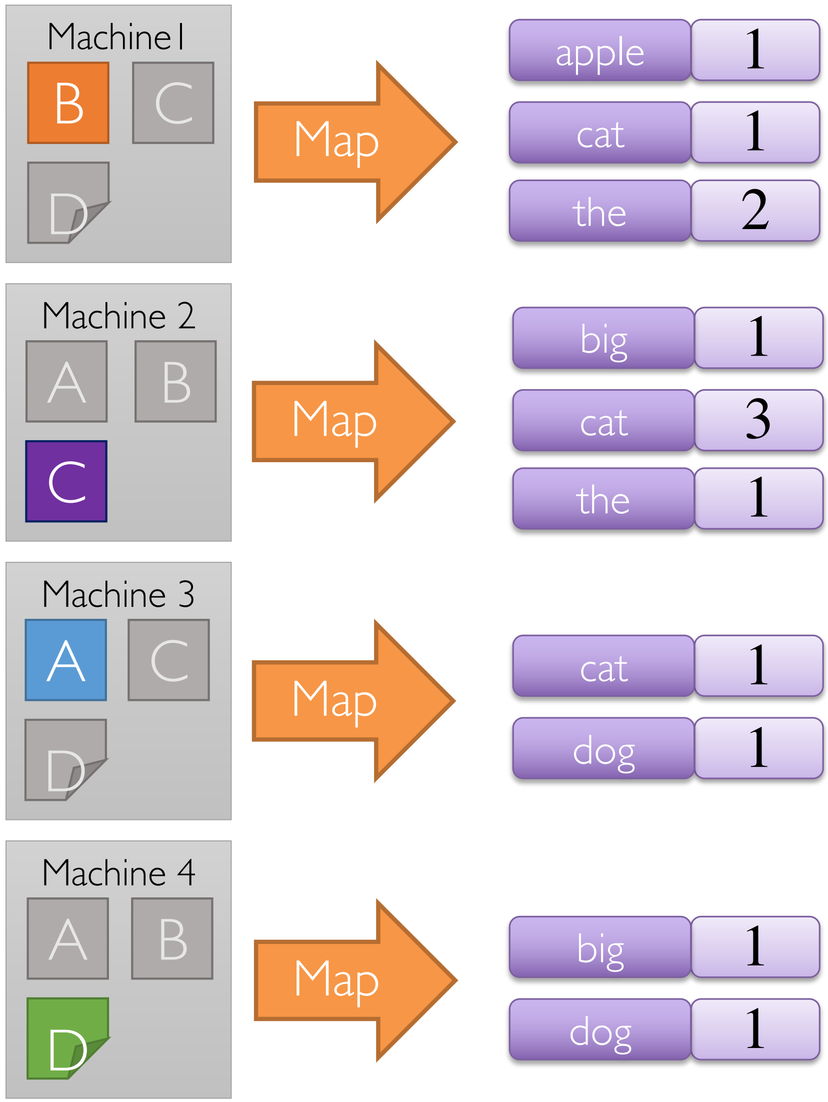
Distributing the Map Function



Executing Map Reduce

Distributing the Map Function

Executing Map Reduce

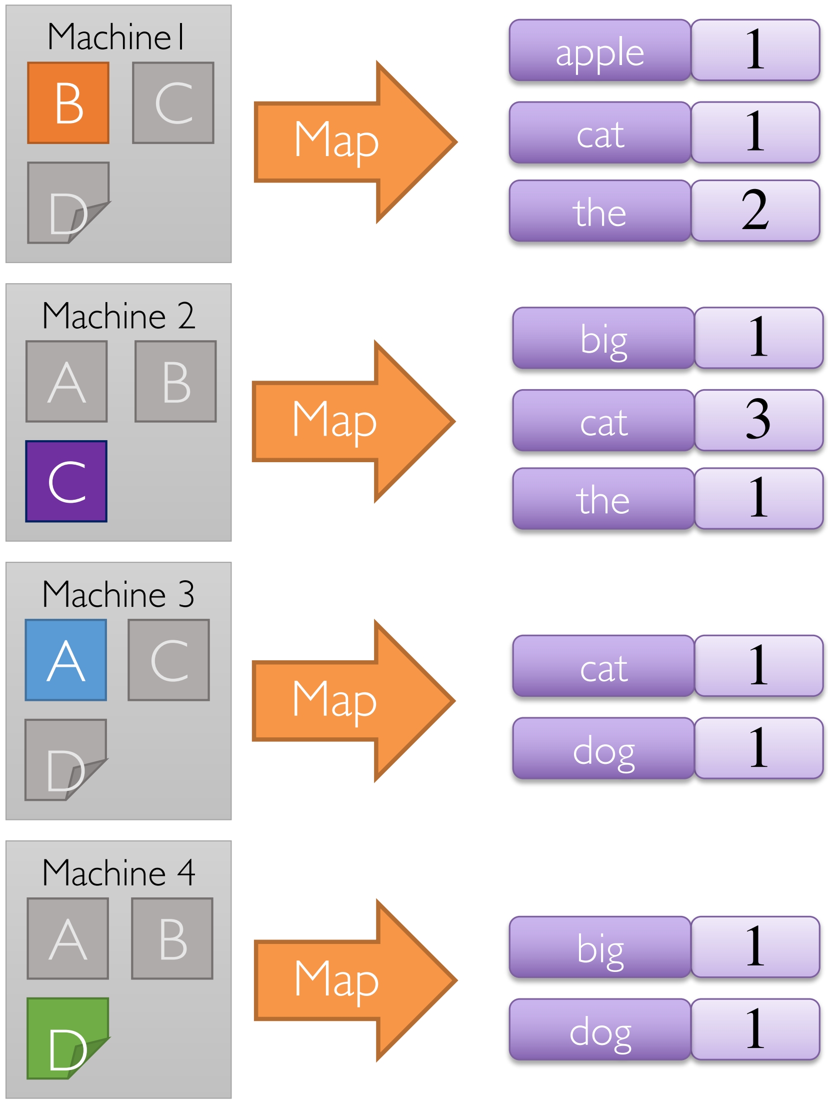


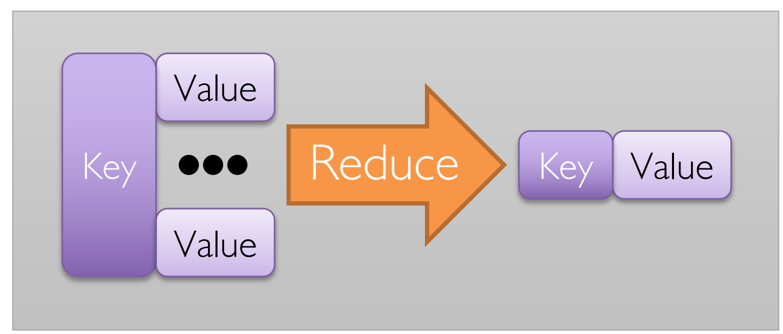
The map function applied to a local part of the big file.

Run in Parallel.

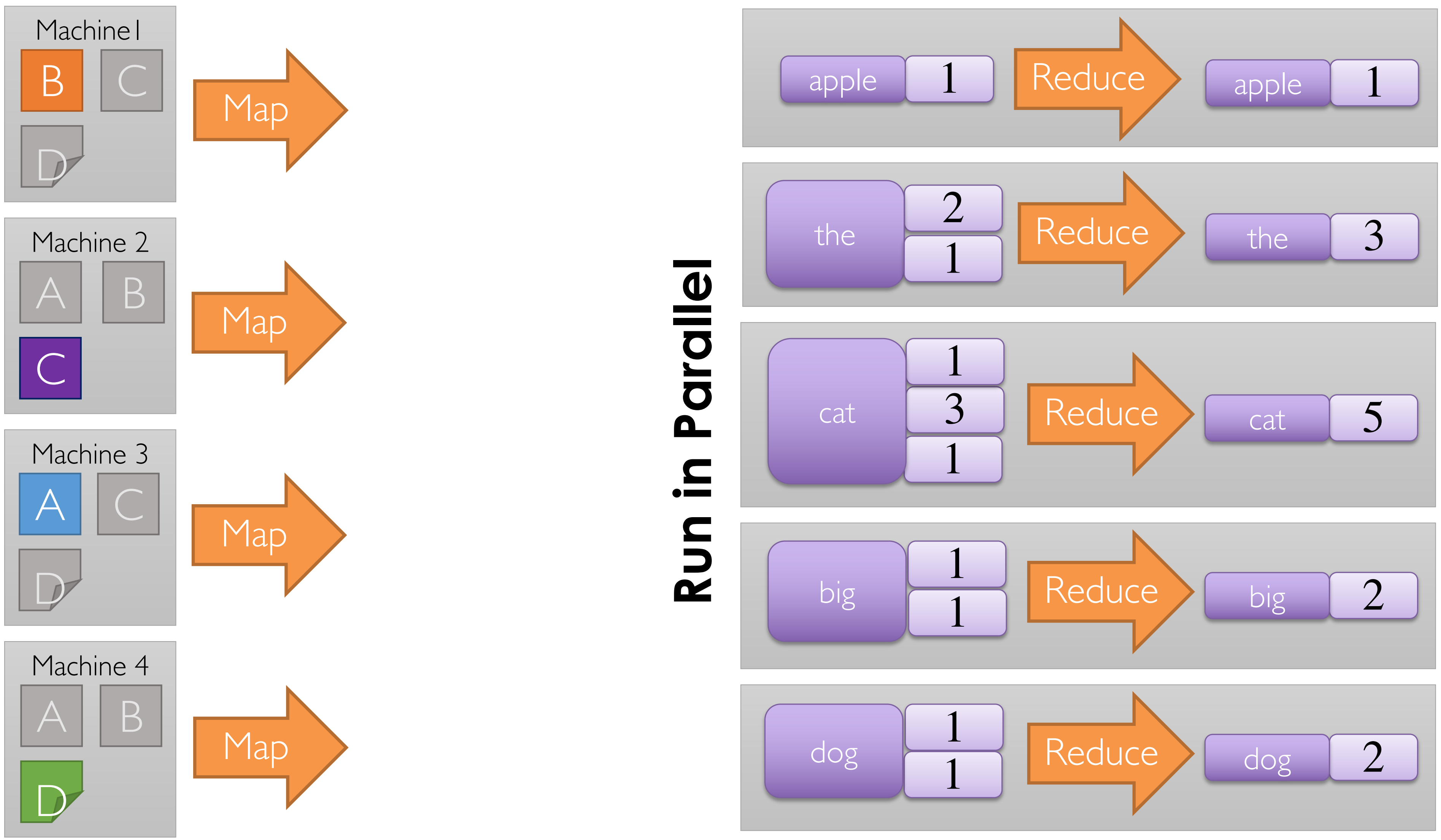
Output is cached for fast recovery on node failure

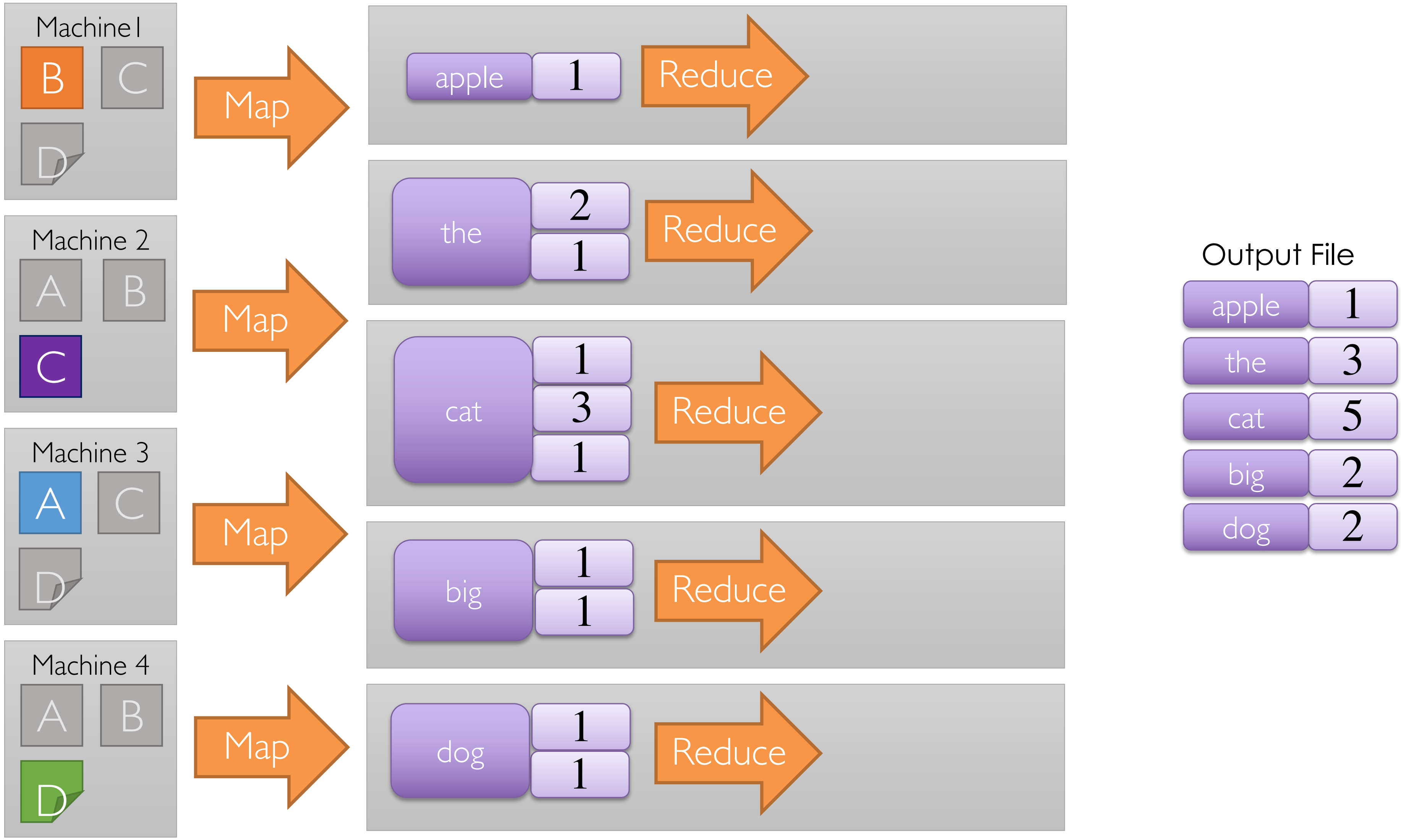
Executing Map Reduce



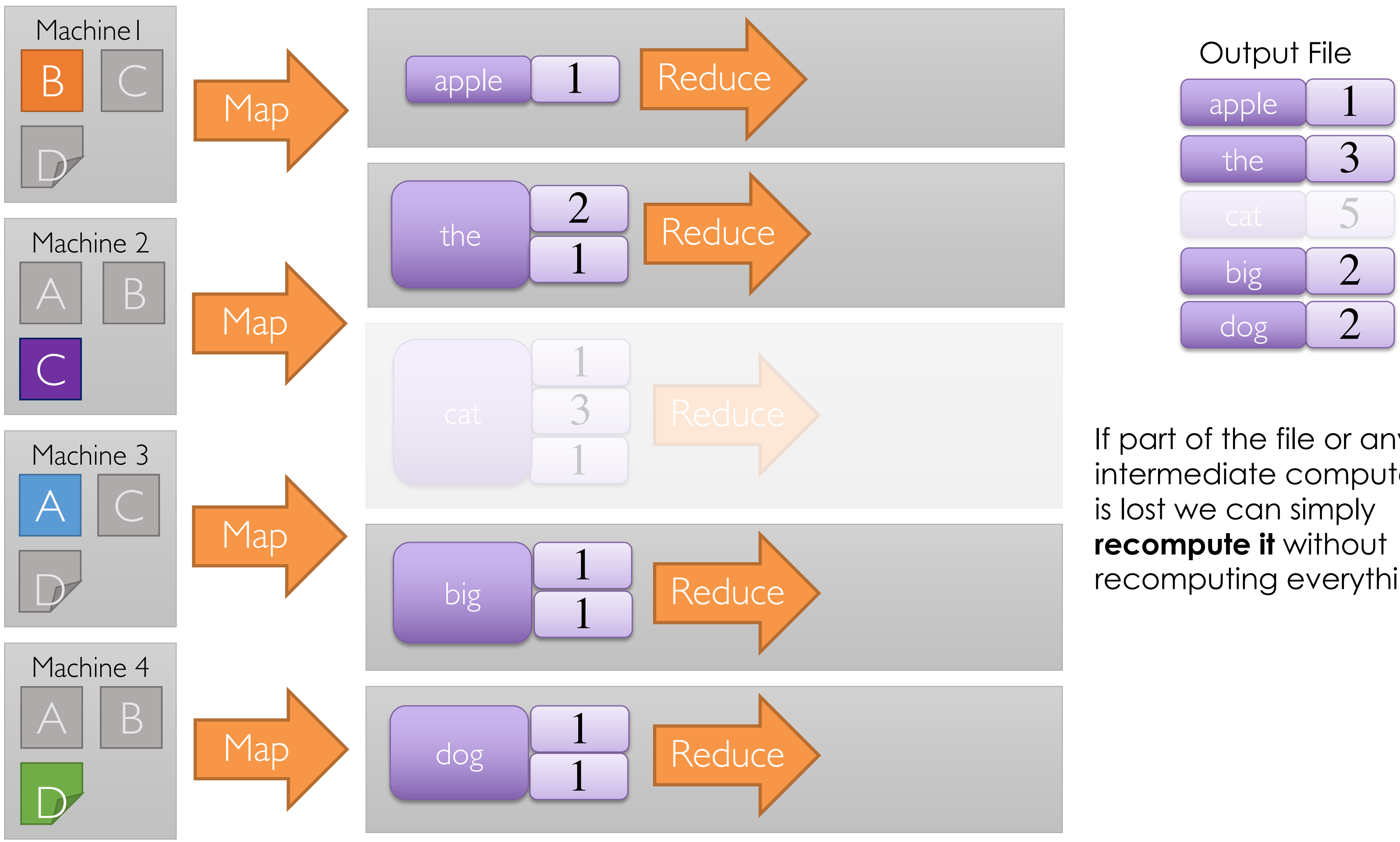


Reduce function can be run on many machines …


Machine I
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 I
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



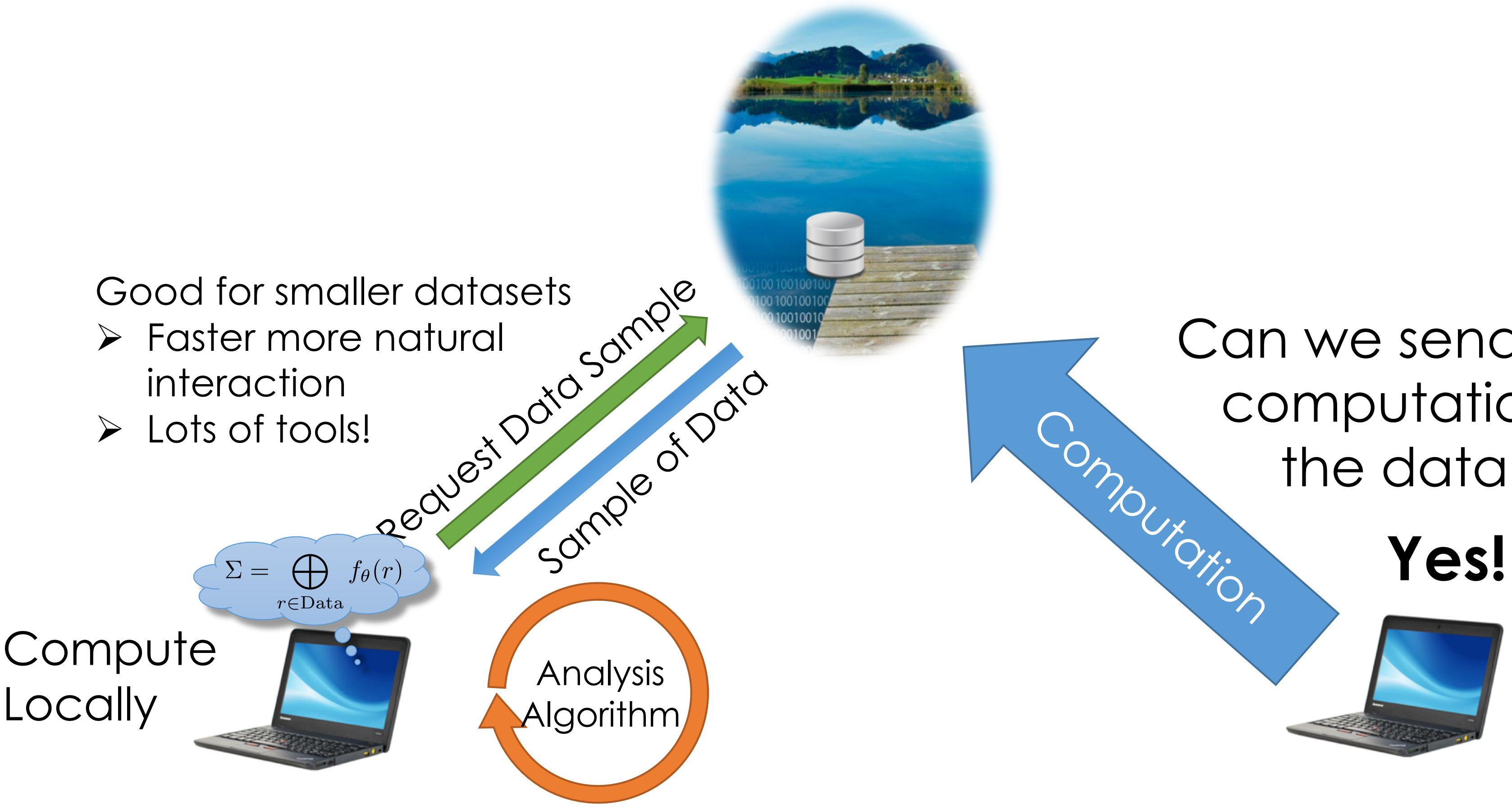
If part of the file or any intermediate computation is lost we can simply **recompute it** without recomputing everything.

Interacting with Data @ Scale

Map-Reduce

Interacting With the Data

Good for smaller datasets
- Faster more natural interaction
- Lots of tools!

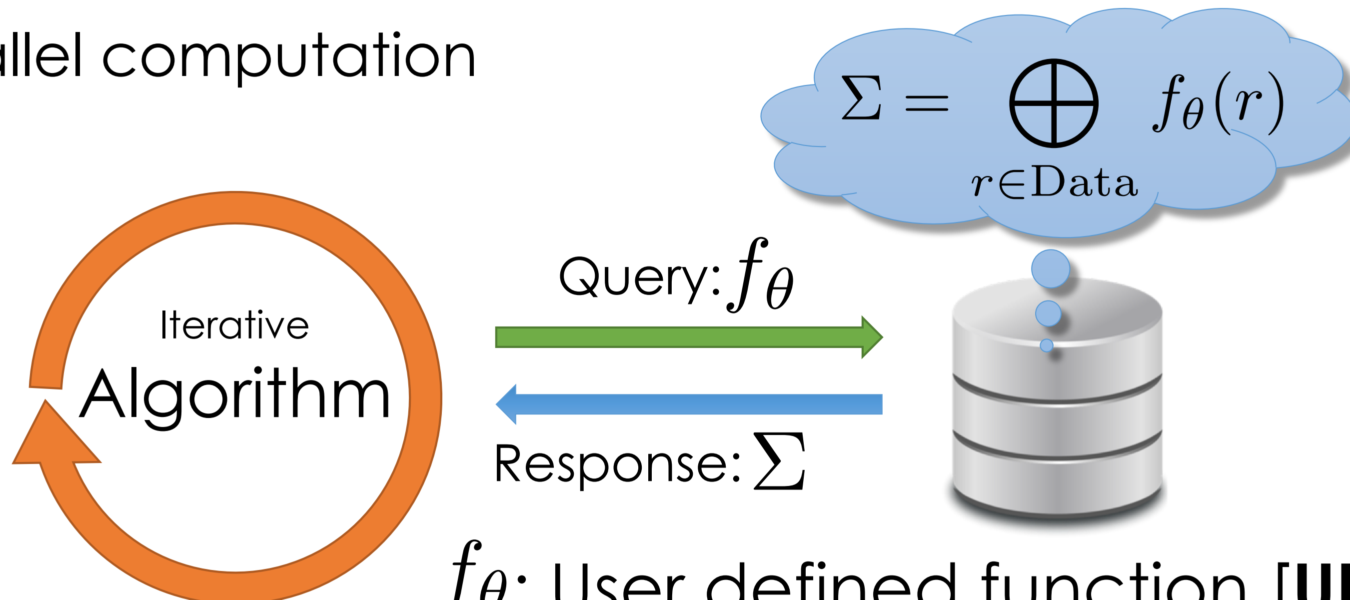


Can we send the computation to the data?

Yes!

Statistical Query Pattern
Common Machine Learning Pattern

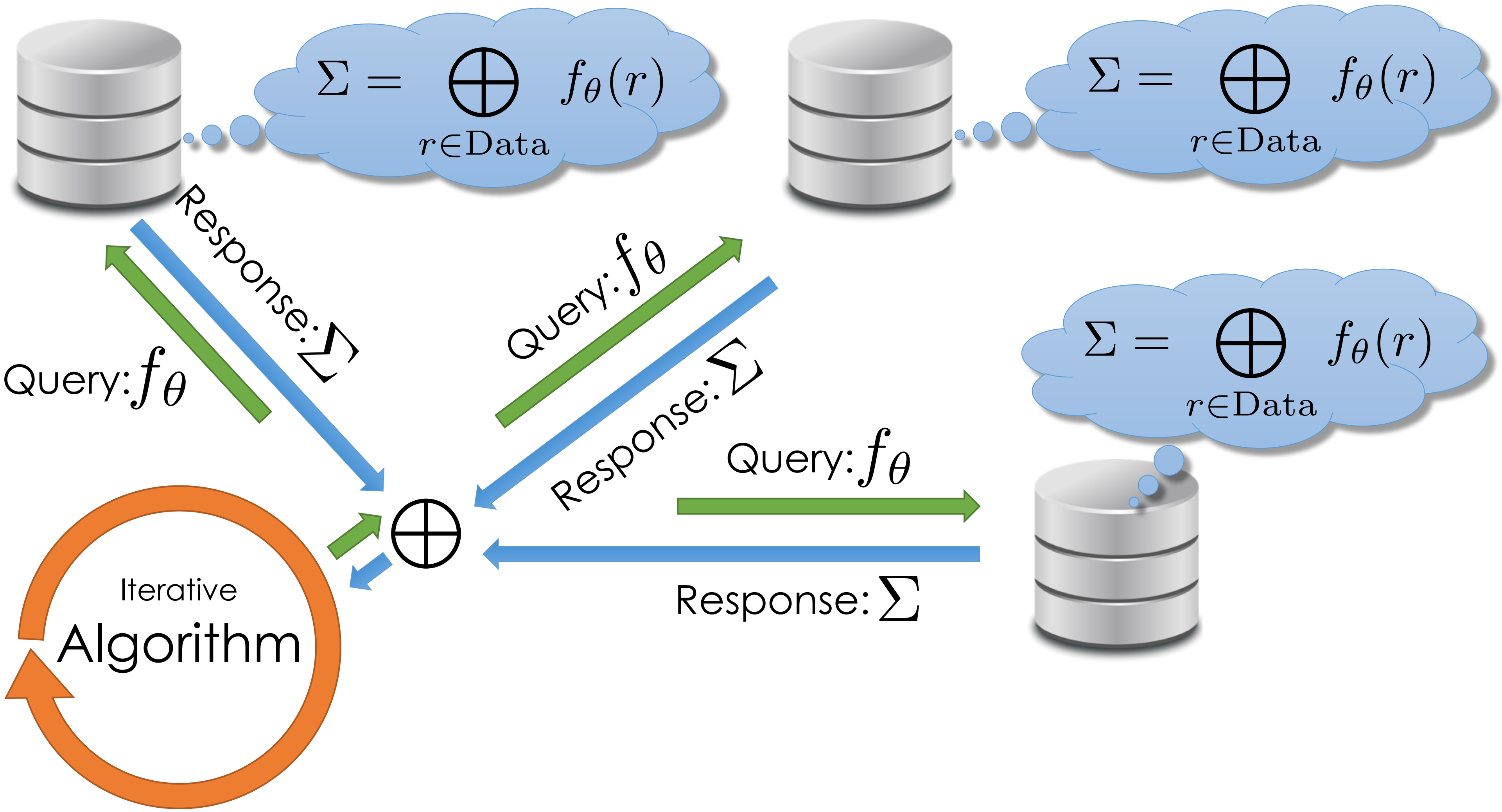
- Computing aggregates of user defined functions
- Data-Parallel computation



f_θ: User defined function [**UDF**]

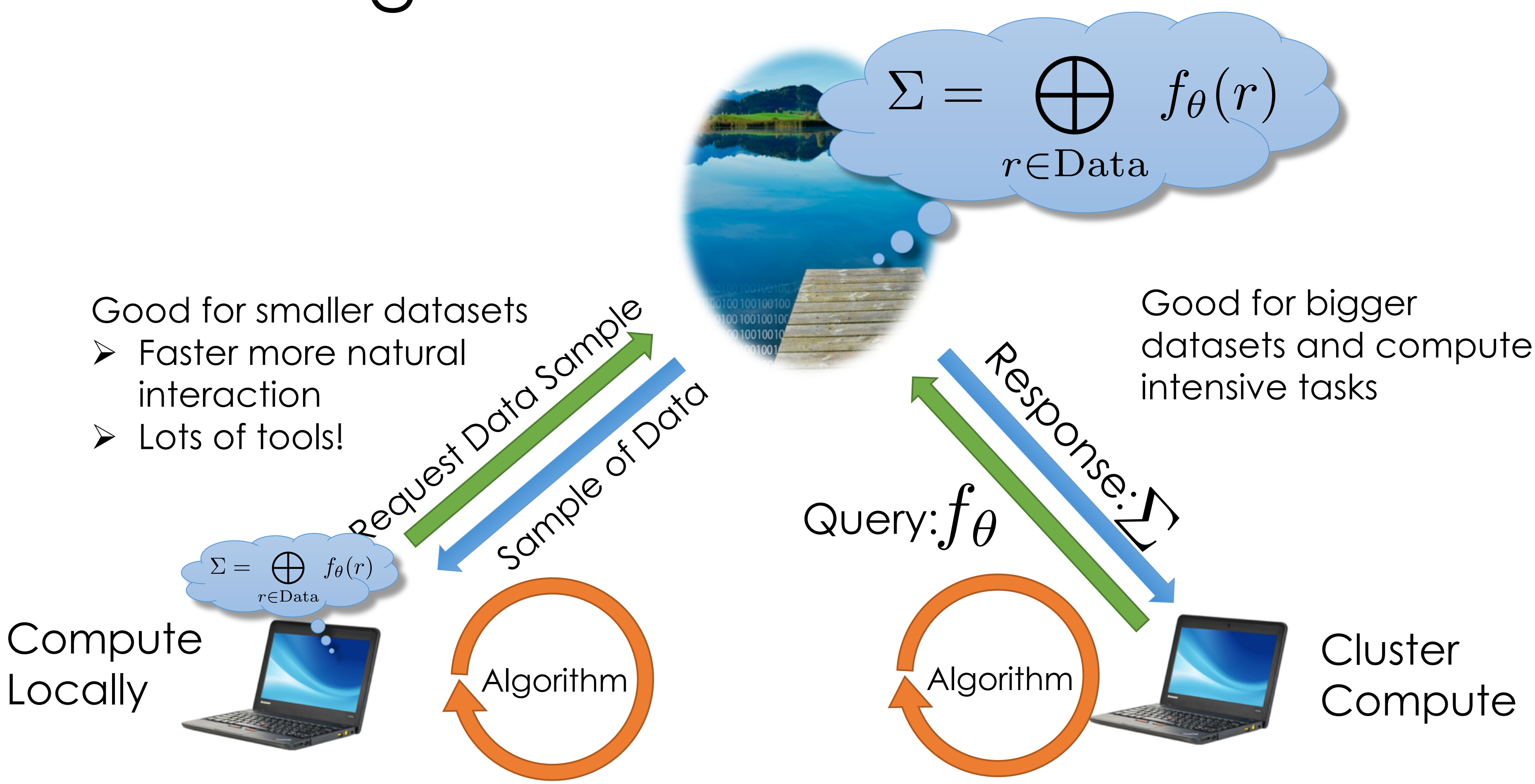
$\bigoplus$: User defined aggregate [**UDA**]

D. Caragea et al., *A Framework for Learning from Distributed Data Using Sufficient Statistics and Its Application to Learning Decision Trees.* Int. J. Hybrid Intell. Syst. 2004



$\Sigma = \bigoplus_{r \in \mathrm{Data}} f_\theta(r)$
$\Sigma = \bigoplus_{r \in \mathrm{Data}} f_\theta(r)$
$\Sigma = \bigoplus_{r \in \mathrm{Data}} f_\theta(r)$
Response: Σ
Query: f_θ
Query: f_θ
Response: Σ
Query: f_θ
Response: Σ
$\bigoplus$
Iterative
Algorithm

Interacting With the Data



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
  - …

Spark

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

Slide provided by M. Zaharia

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

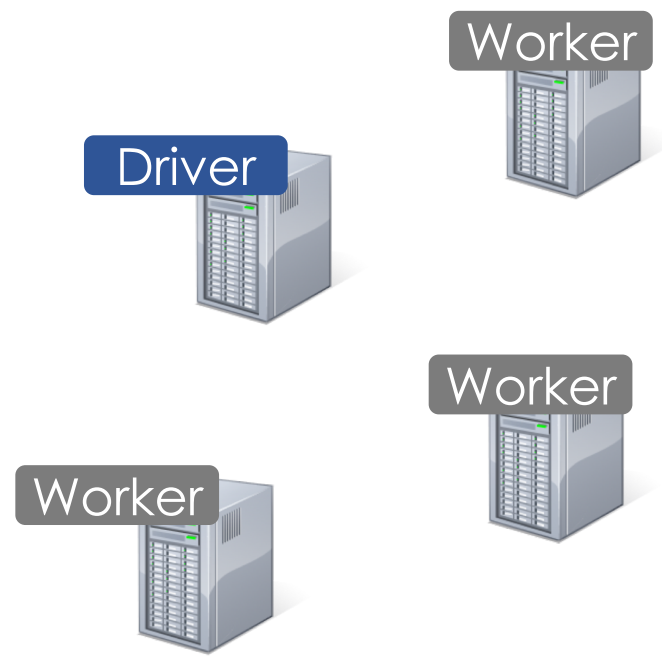
- Transformations f(RDD) => RDD
  - Lazy (not computed immediately)
  - E.g., “map”, “filter”, “groupBy”
- Actions:
  - Triggers computation
  - E.g. “count”, “collect”, “saveAsTextFile”

Example: Log Mining

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

Example: Log Mining

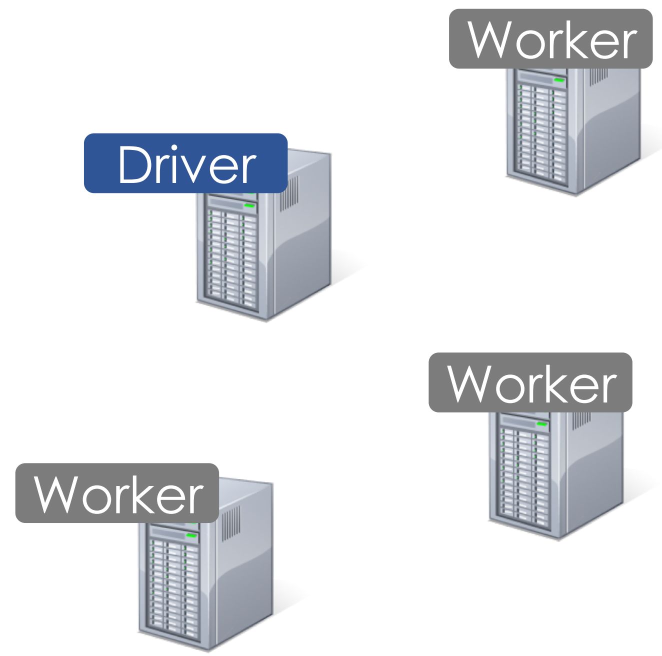
Load error messages from a log into memory, then interactively search for various patterns



Example: Log Mining

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

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

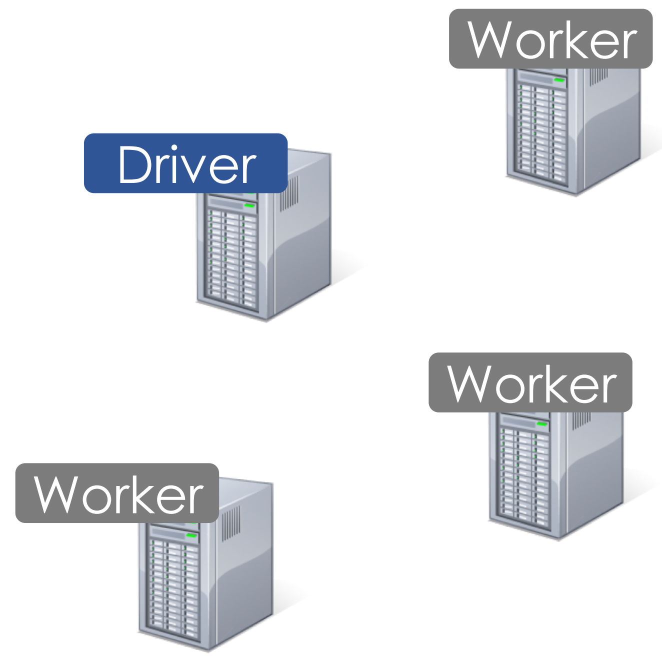


Example: Log Mining

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

Base RDD

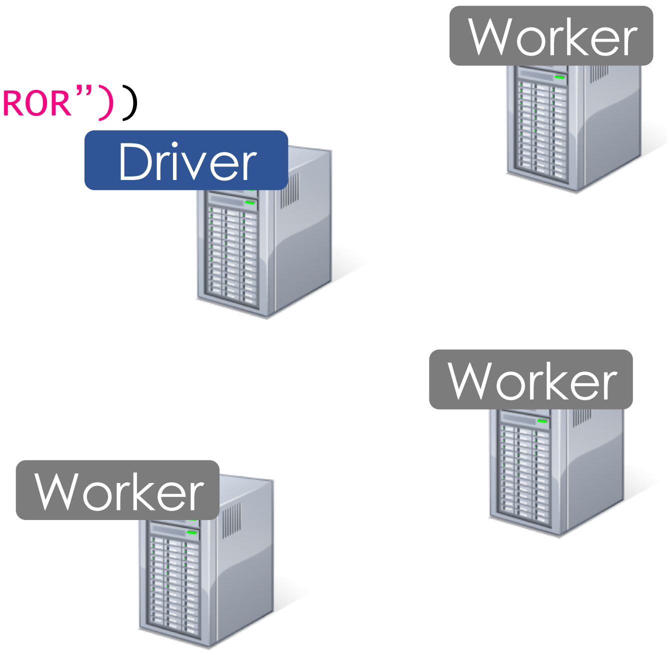
```
lines = spark.textFile("hdfs://file.txt")
```



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"))
```

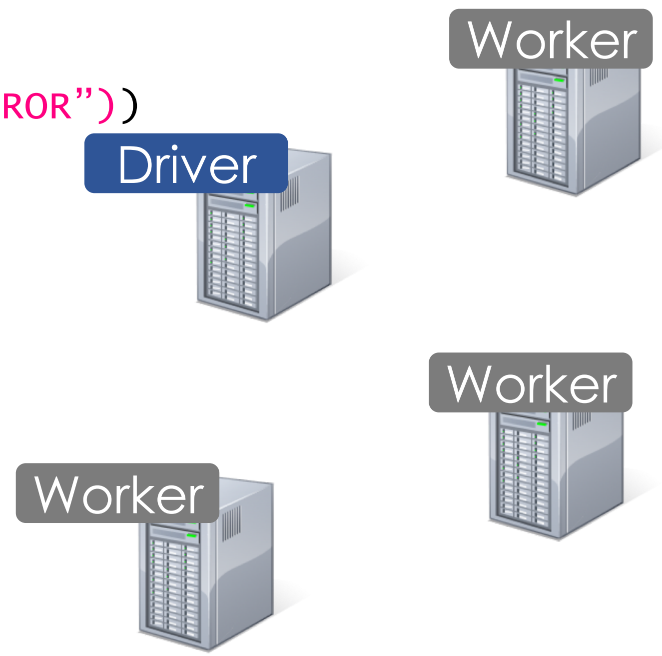


Example: Log Mining

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"))
```

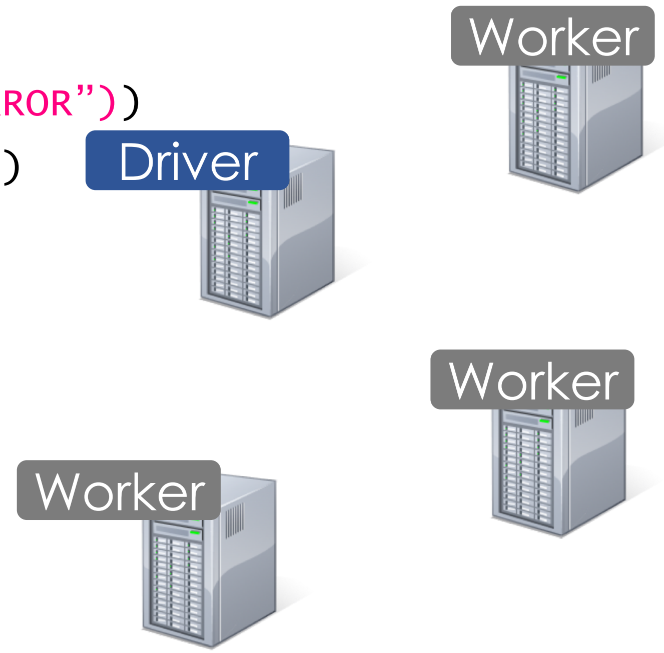


Example: Log Mining

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()
```

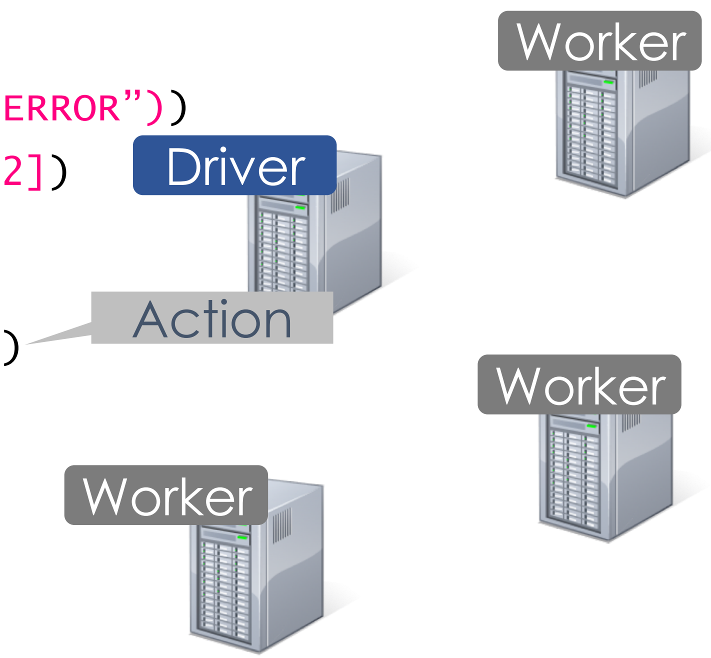


Example: Log Mining

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()
```

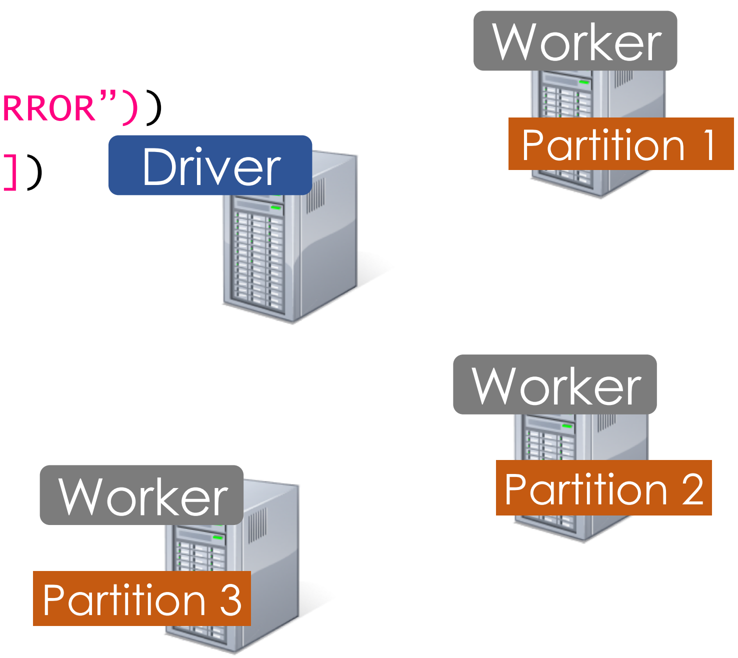


Example: Log Mining

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()
```

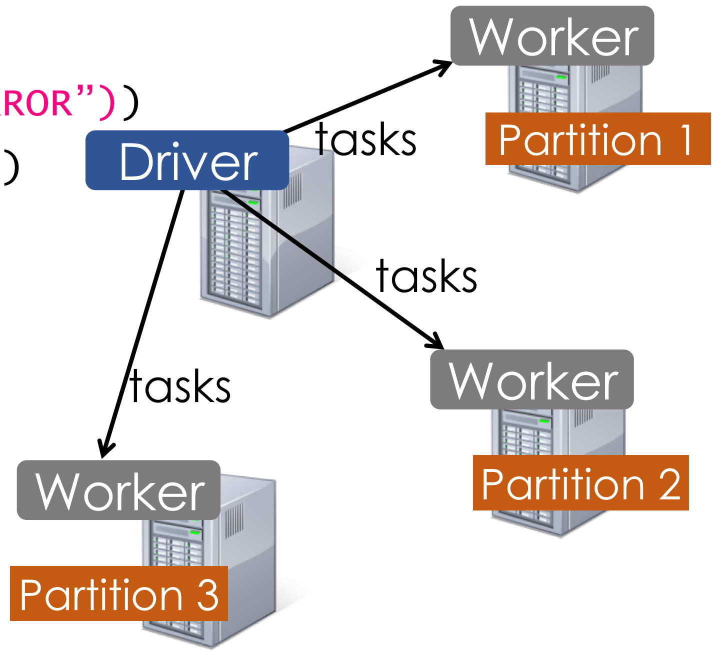


Example: Log Mining

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()
```

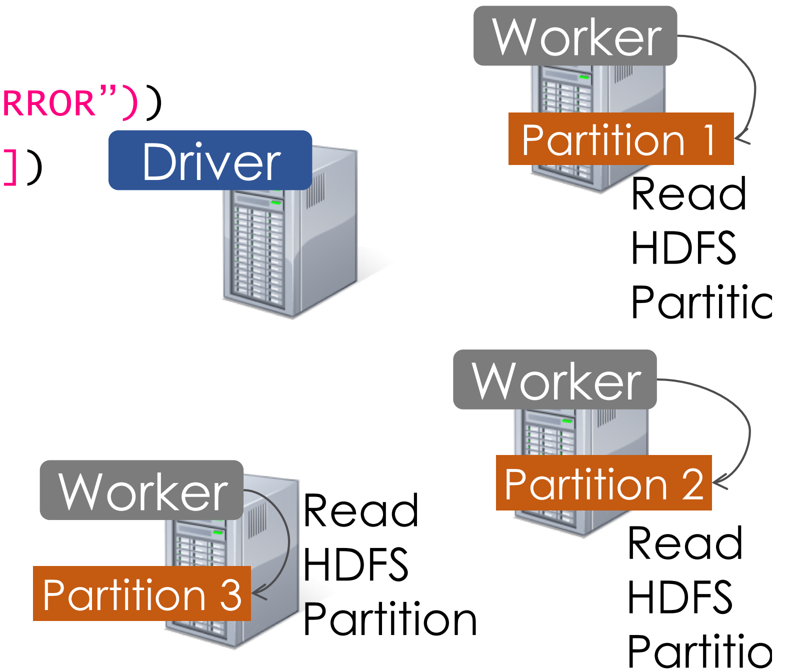


Example: Log Mining

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()
```

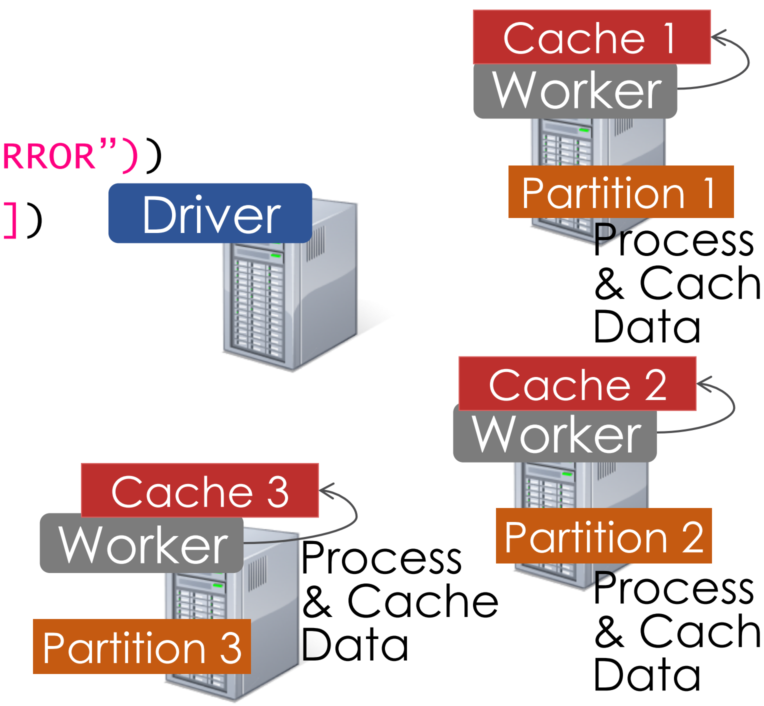


Example: Log Mining

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()
```

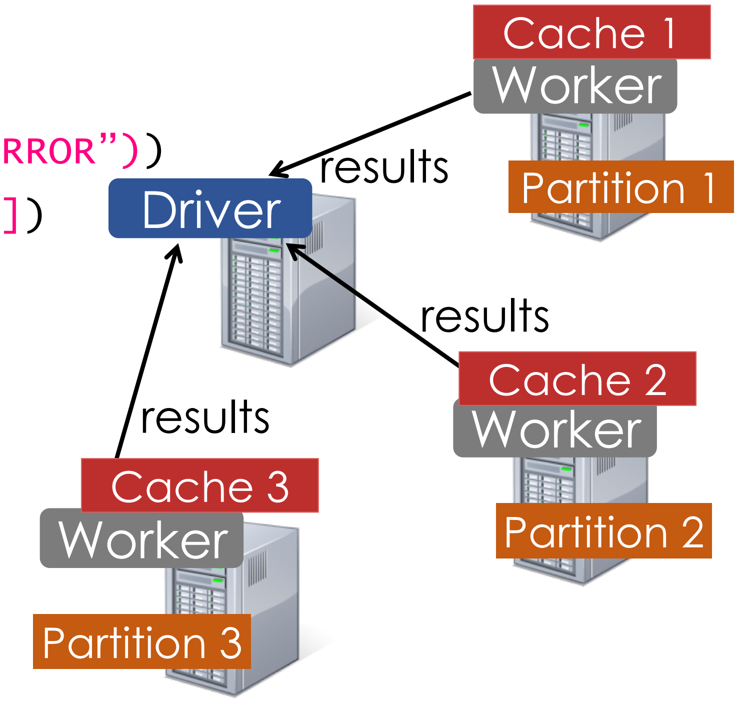


Example: Log Mining

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()
```

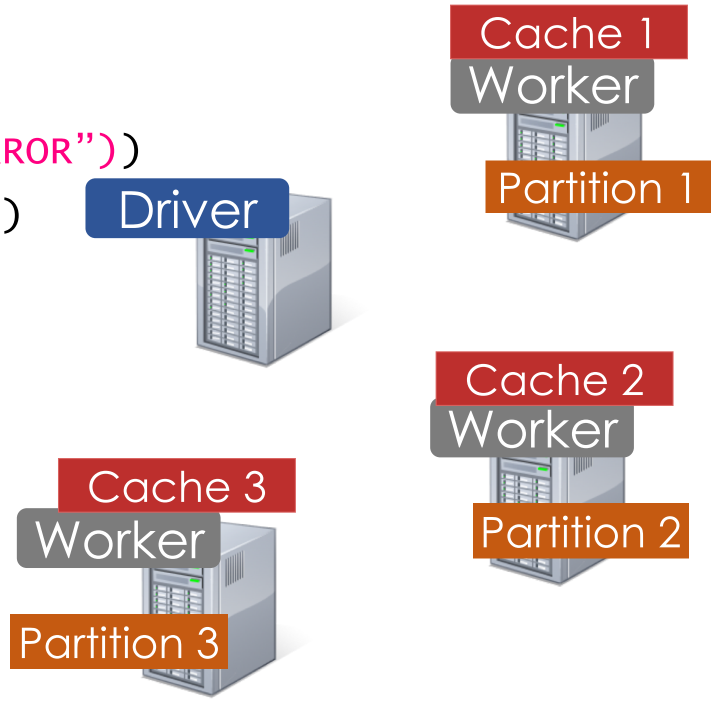


Example: Log Mining

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()
```

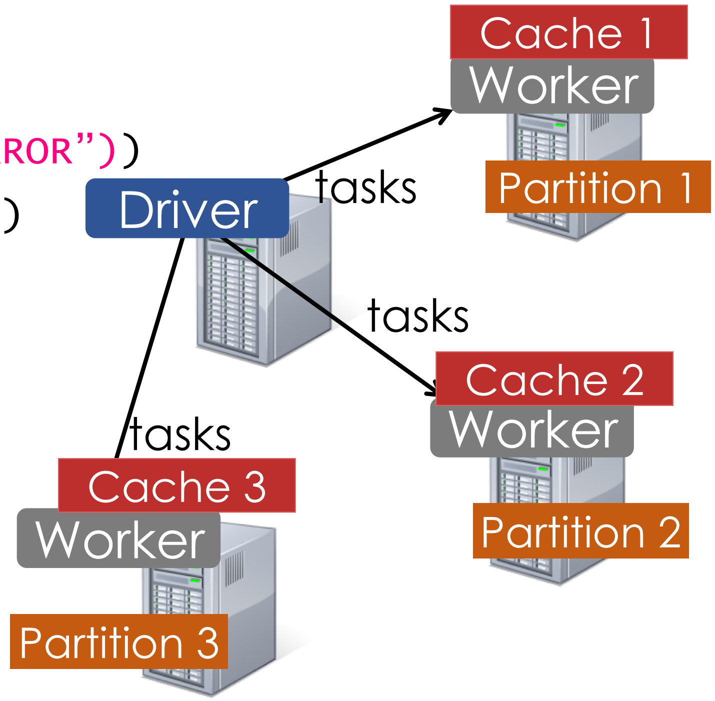


Example: Log Mining

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()
```

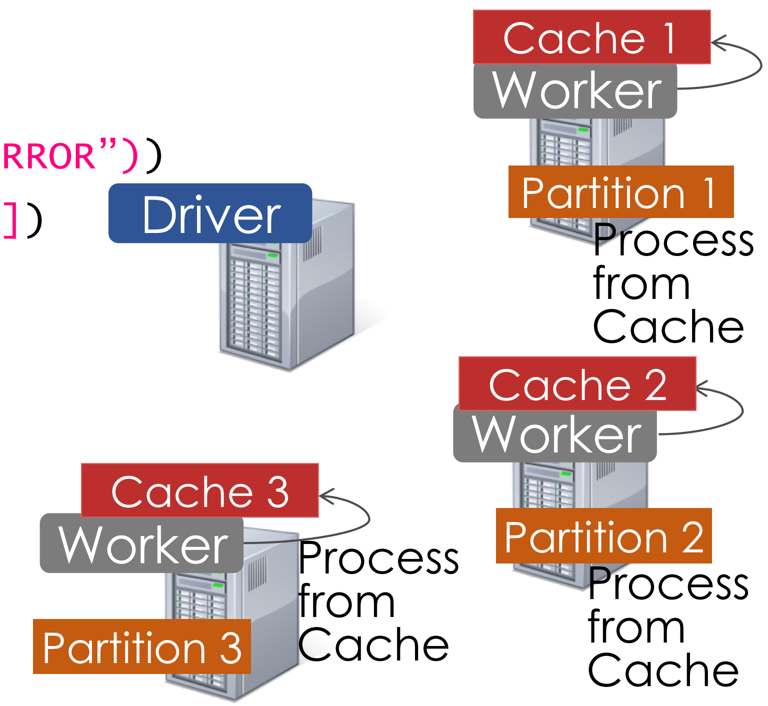


Example: Log Mining

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()
```

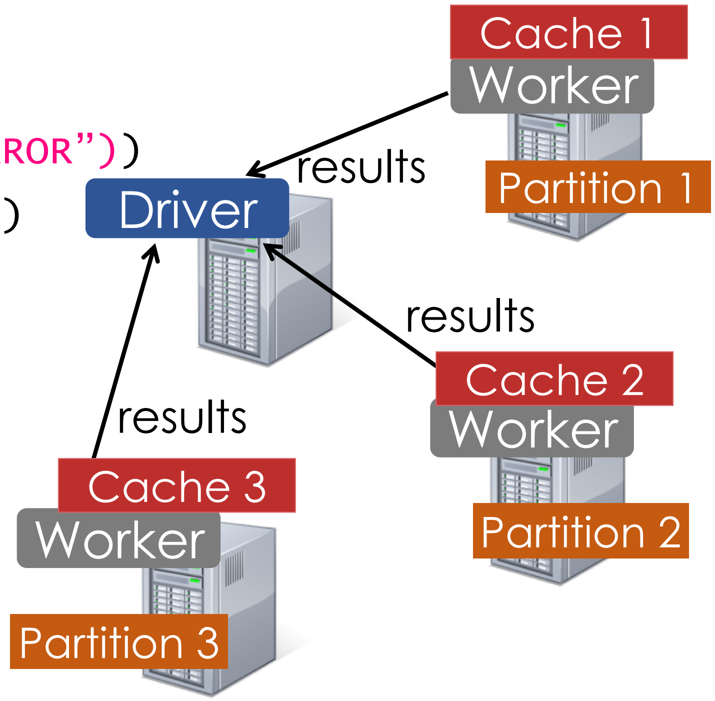


Example: Log Mining

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()
```

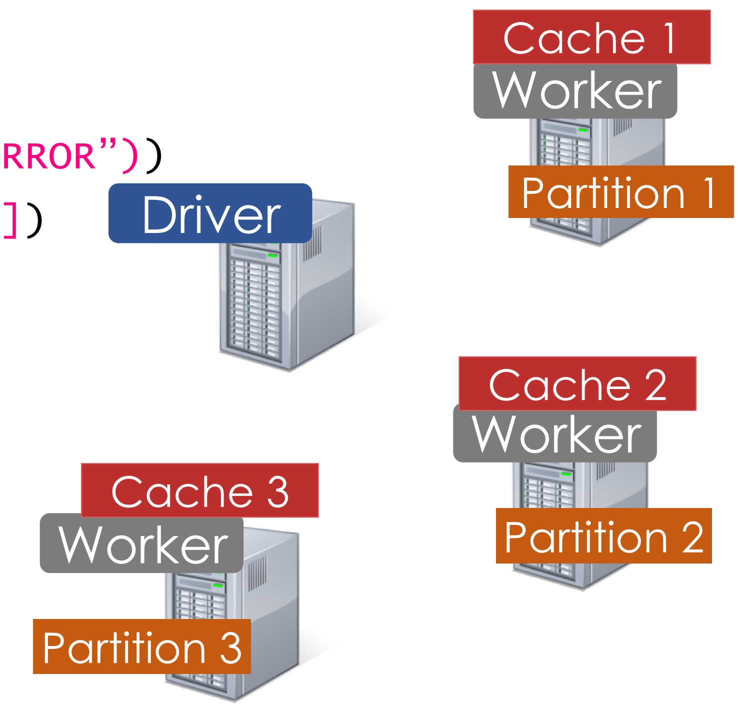


Example: Log Mining

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

Abstraction: *Dataflow Operators*

```
map
filter
groupBy
sort
union
join
leftOuterJoin
rightOuterJoin
```

```
reduce
count
fold
reduceByKey
groupByKey
cogroup
cross
zip
```

```
sample
take
first
partitionBy
mapWith
pipe
save
...
```

Abstraction: *Dataflow Operators*

`map`
`filter`
`groupBy`
`sort`
`union`
`join`
`leftOuterJoin`
`rightOuterJoin`

`reduce`
`count`
`fold`
`reduceByKey`
`groupByKey`
`cogroup`
`cross`
`zip`

`sample`
`take`
`first`
`partitionBy`
`mapWith`
`pipe`
`save`
`...`

Language Support

Python

```
lines = sc.textFile(...)
lines.filter(lambda s: "ERROR" in s).count()
```

Scala

```
val lines = sc.textFile(...)
lines.filter(x => x.contains("ERROR")).count()
```

Java

```
JavaRDD<String> lines = sc.textFile(...);
lines.filter(new Function<String, Boolean>() {
  Boolean call(String s) {
    return s.contains("error");
  }
}).count();
```

Standalone Programs

Python, Scala, & Java

Interactive Shells

Python & Scala

Performance

Java & Scala are faster due to static typing