Big Data Analytics & Spark

Slides by:
Joseph E. Gonzalez
jegonzal@cs.berkeley.edu

With revisions by:
Josh Hug & John DeNero

Operational Data Store

Data Warehouse

ETL (Extract, Transform, Load)

Snowflake Schema

Schema on Read

Data in the Organization

A little bit of buzzword bingo!

Star Schema

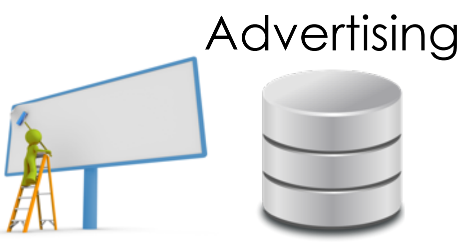
Data Lake

OLAP (Online Analytics Processing)

Inventory

How we like to think of data in the organization

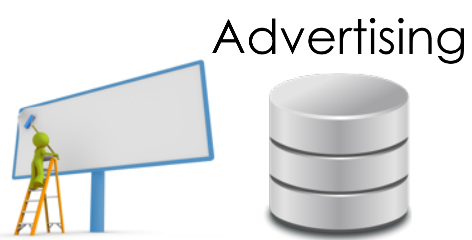
The reality…



Sales (Asia)

Sales (US)

Inventory



Advertising

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)

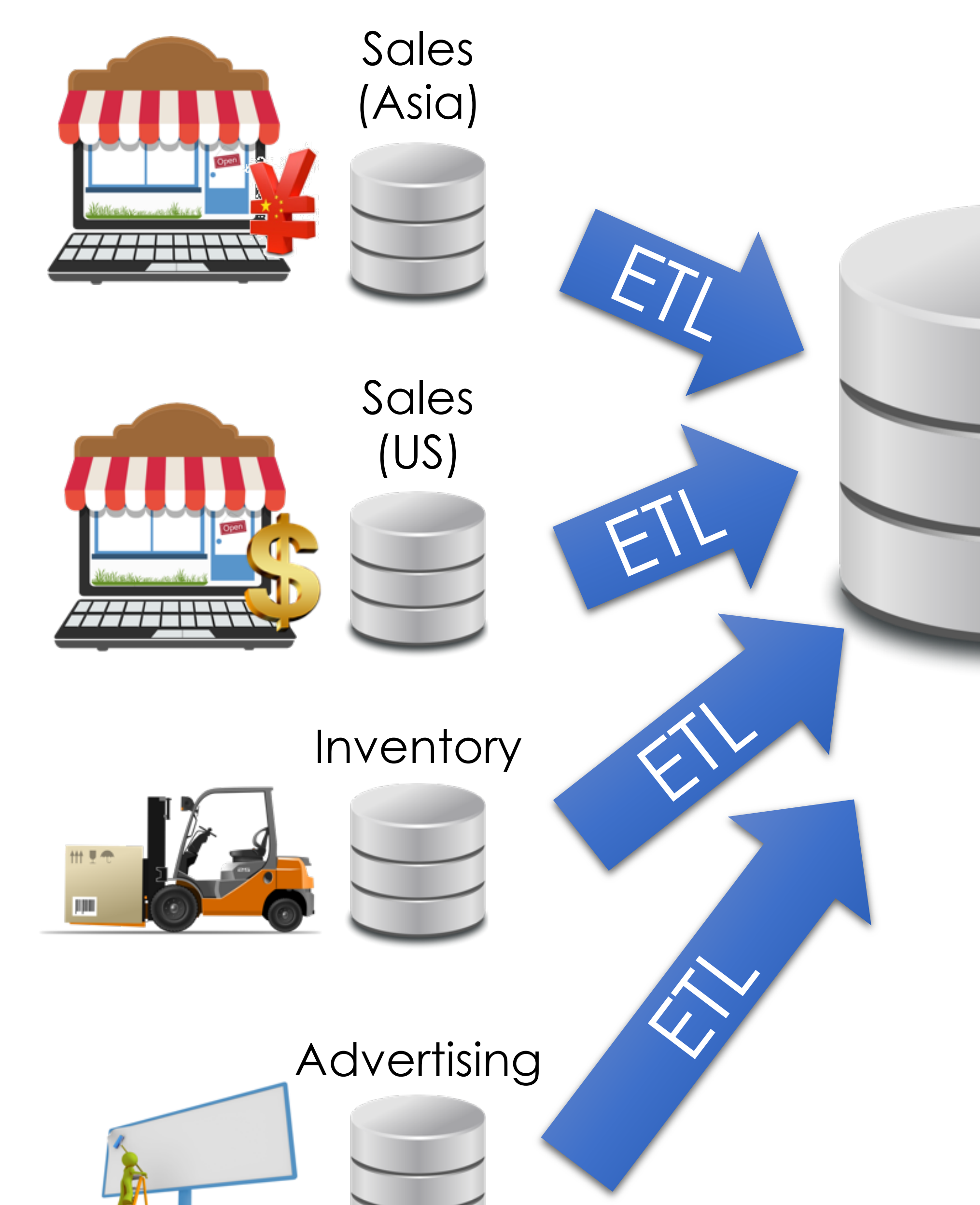
Extract & Load: provides a snapshot of operational data

- 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	8	[illegible]0/16	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

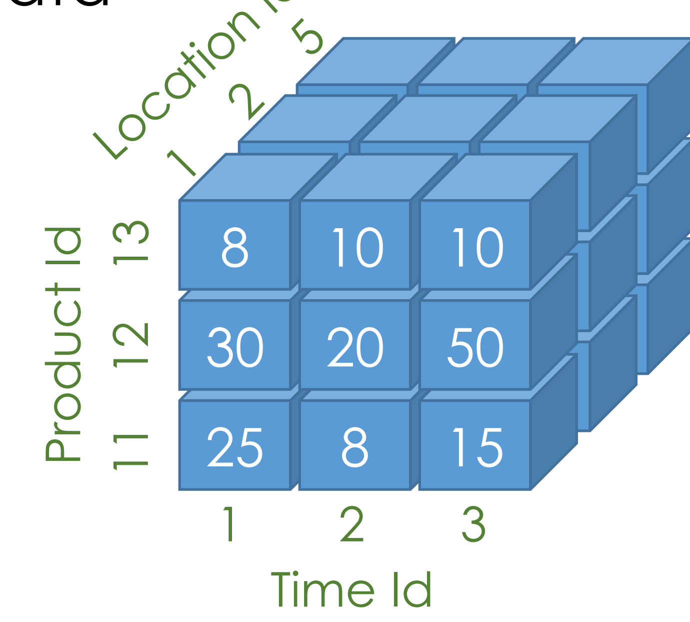
Products

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



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

Dimension Tables

Locations

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

Products

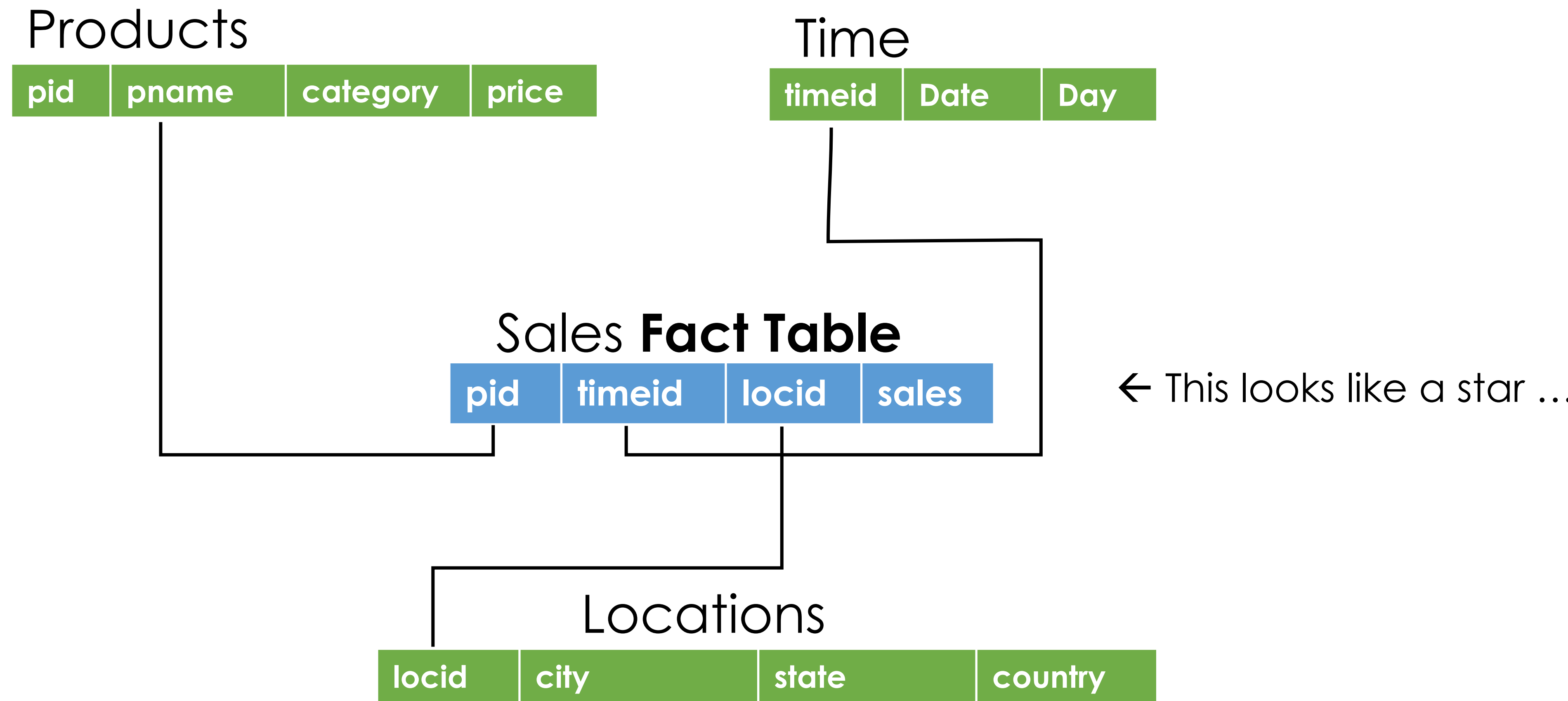
pid	pname	category	price
11	Corn	Food	25
12	Galaxy 1	Phones	18
13	Peanuts	Food	2

Time

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

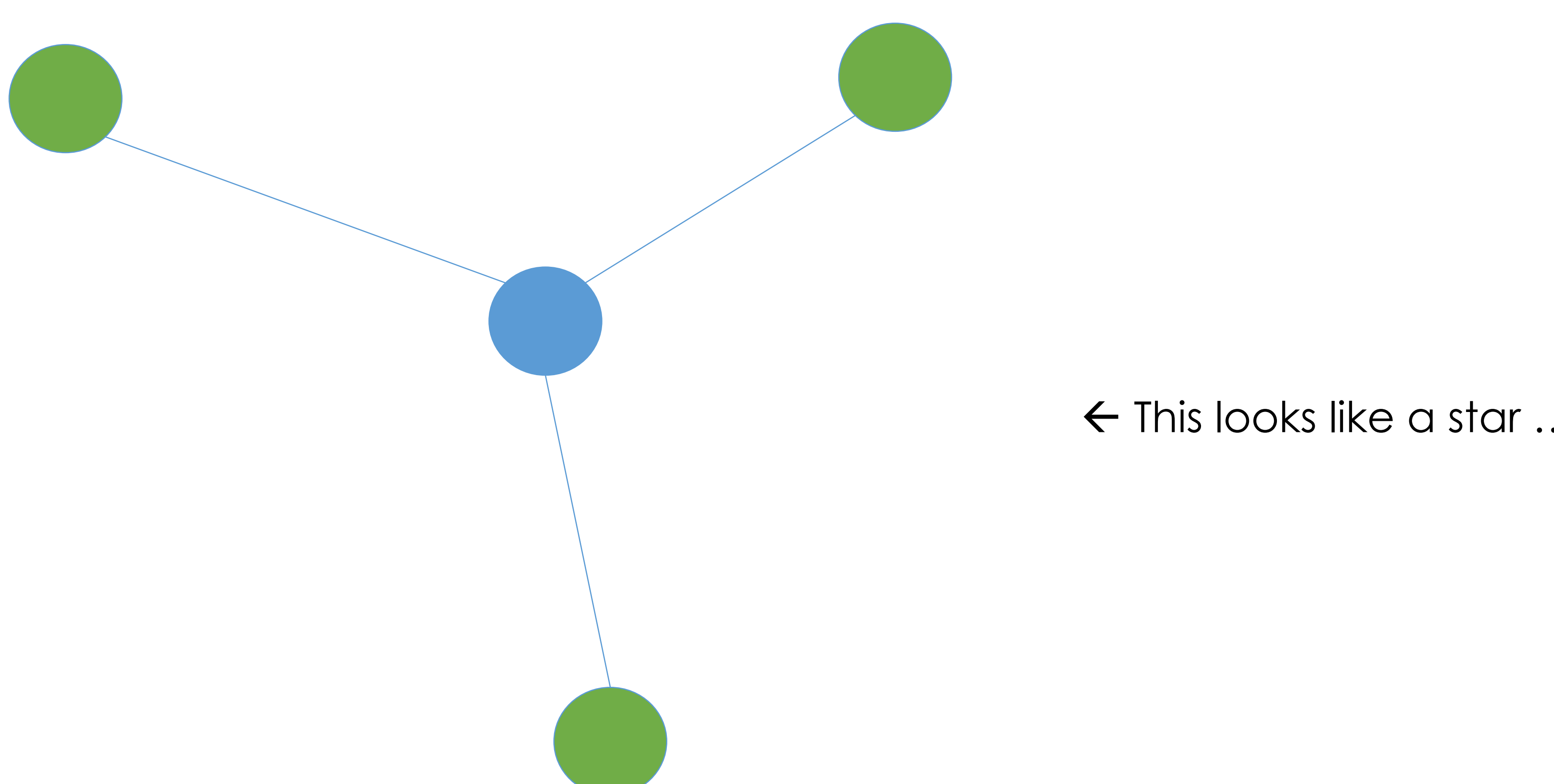
- Fact Table
  - Minimizes redundant info
  - Reduces data errors
- Dimensions
  - Easy to manage and summarize
  - Rename: Galaxy1 → Phablet
- Normalized Representation
- How do we do analysis?
  - **Joins!**

The Star Schema



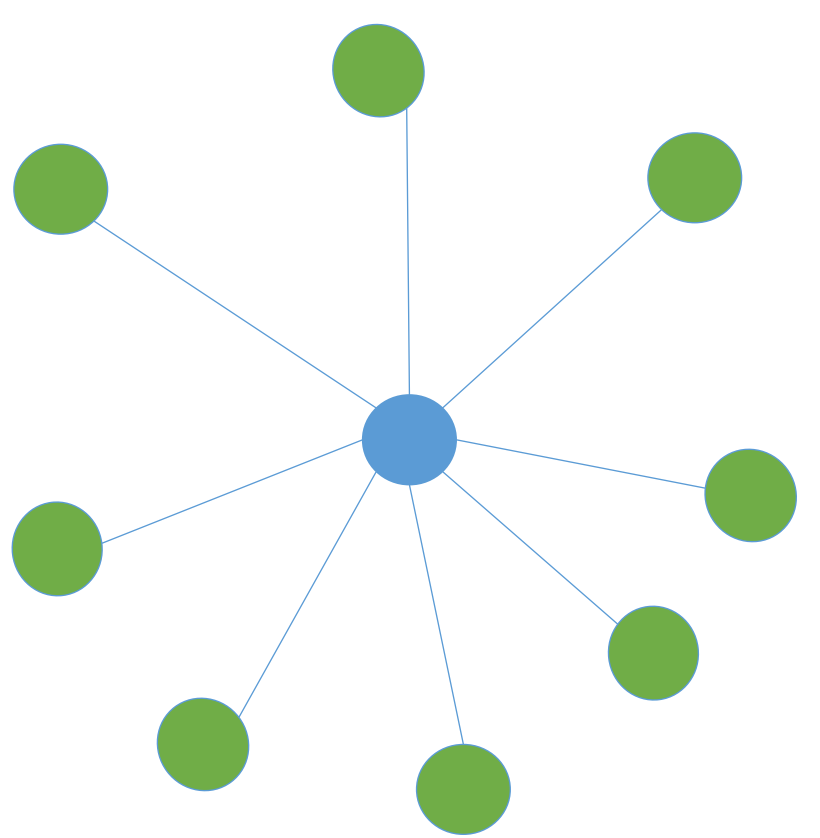
← This looks like a star …

The Star Schema



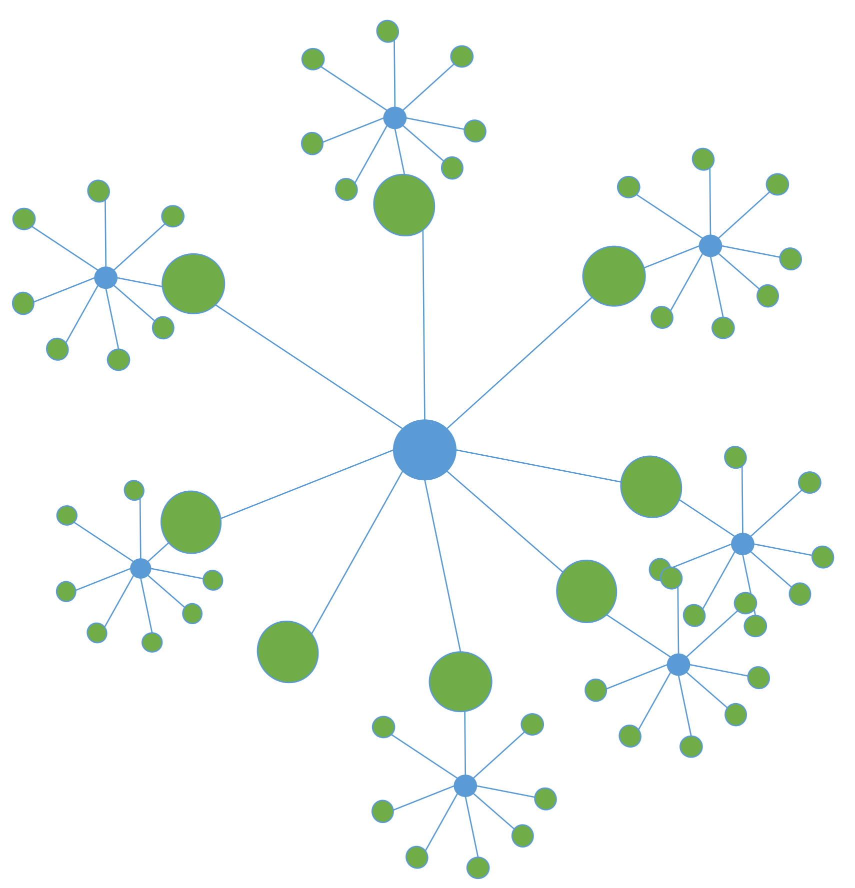
← This looks like a star …

The Star Schema



← This looks like a star …

The Snowflake Schema



← This looks like a snowflake …

See CS 186 for more!

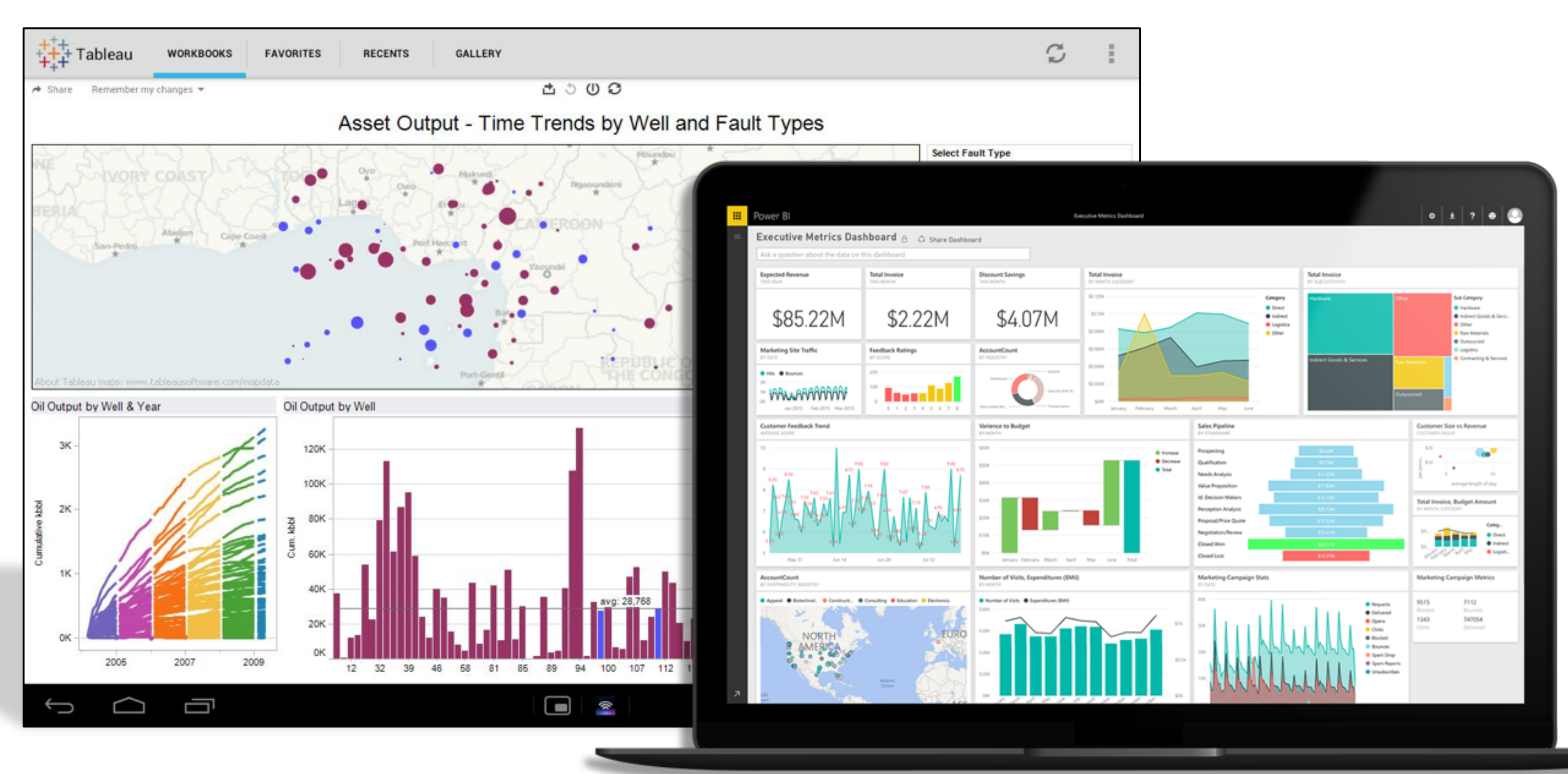
Online Analytics Processing (**OLAP**)

Users interact with multidimensional data:

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

Reporting and Business Intelligence (BI)

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



Sales (US)

ETL

ETL

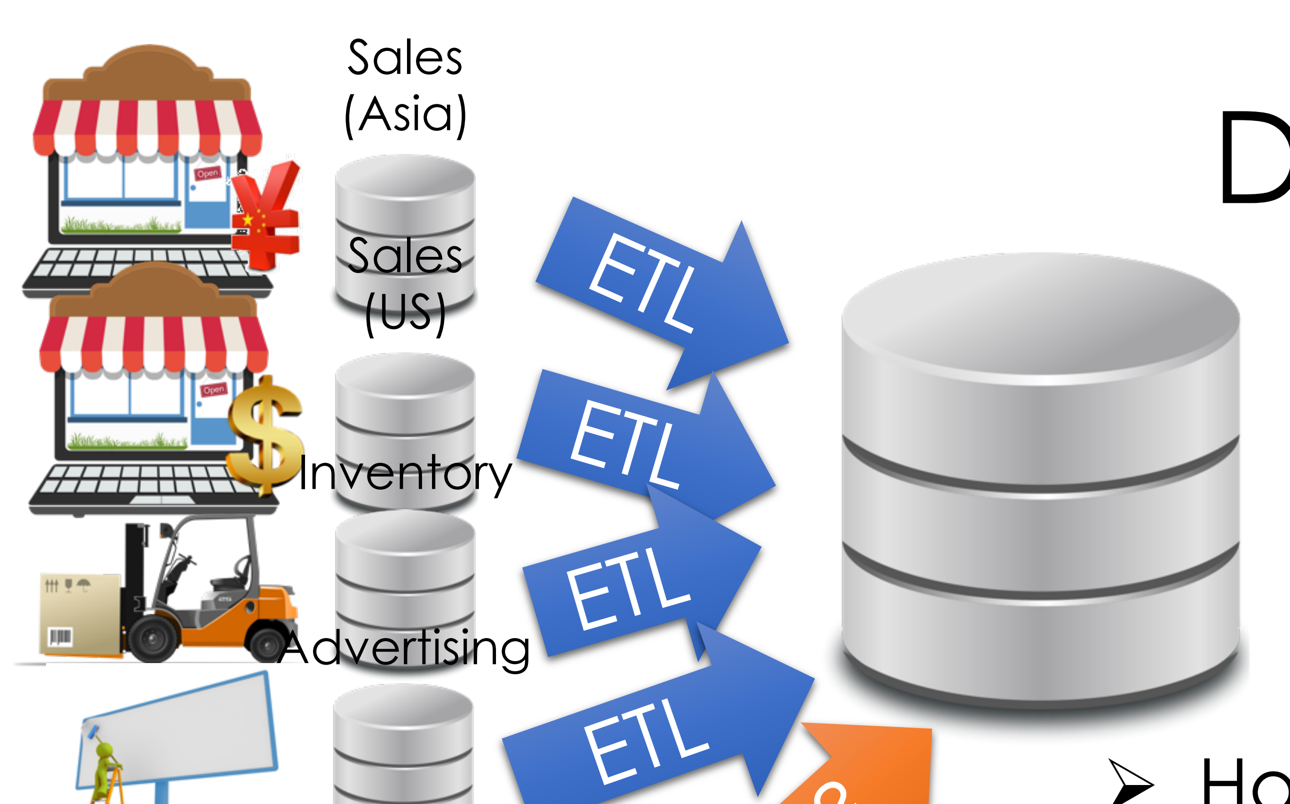
Advertising

Collects and organizes historical data from multiple sources

So far …

- Star Schemas
- Data cubes
- OLAP

Data Warehouse



Text/Log Data

Photos & Videos

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
45123 1333	01-22-2016	1	1	
47234 2122	06-17-2017	0	1	
57182 7231	03-15-2009	0	0	
23847 2733	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

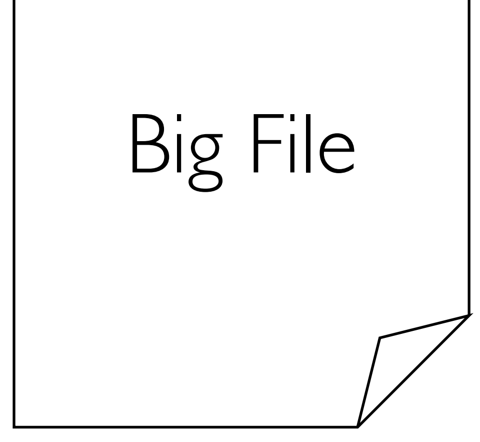
How do we **store** and **compute** on large unstructured datasets

- 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

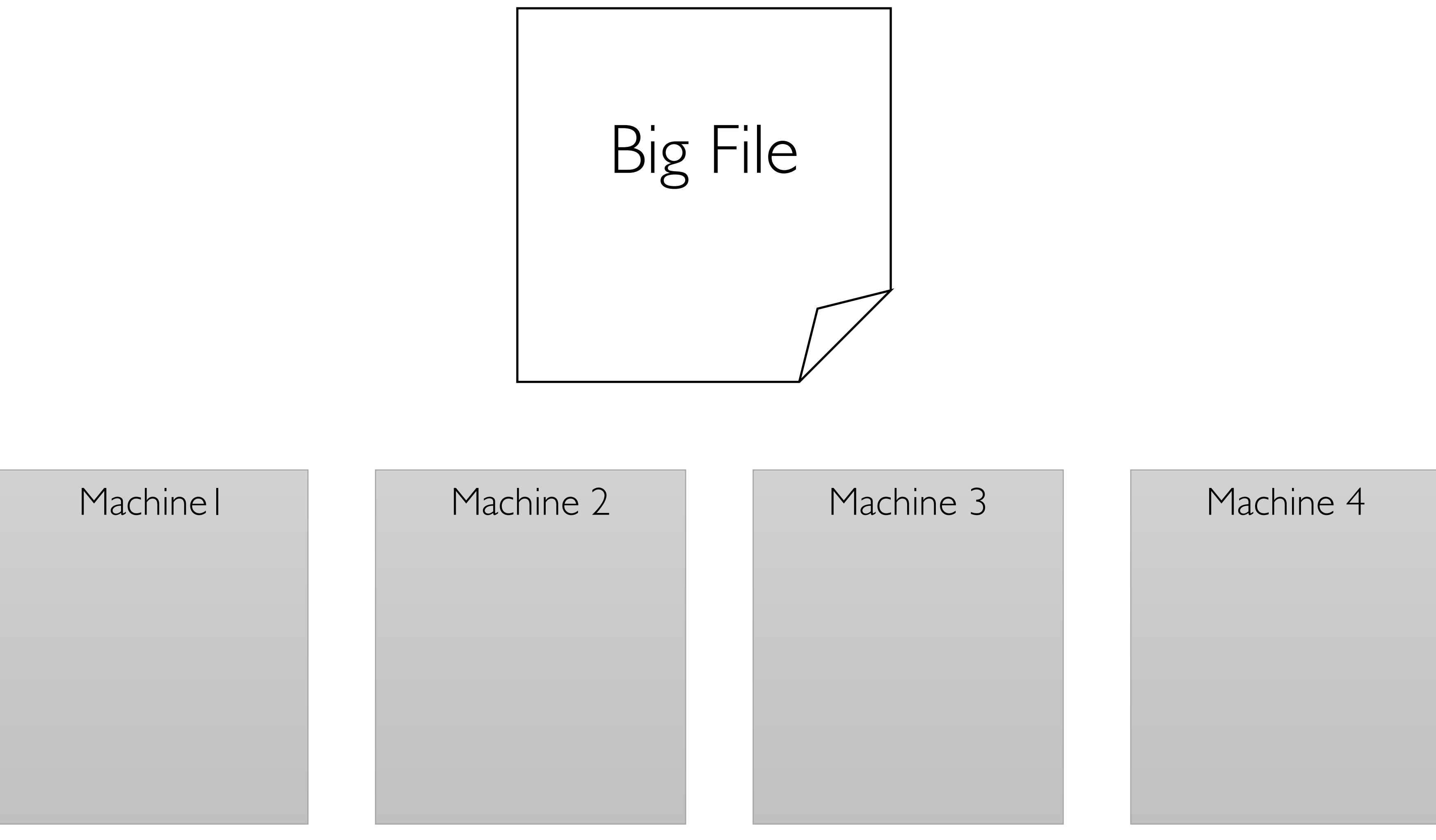
Fault Tolerant Distributed File Systems



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

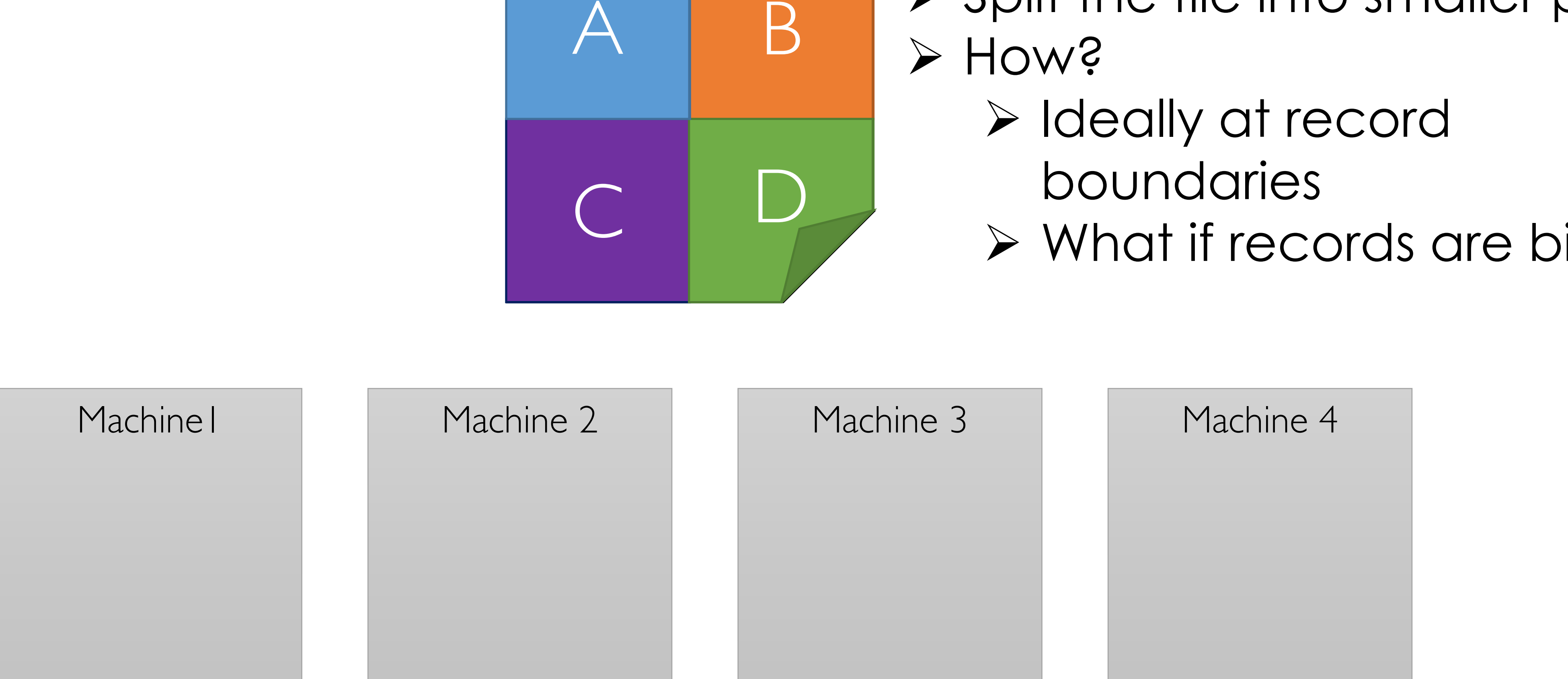
[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems



[Ghemawat et al., SOSP'03]

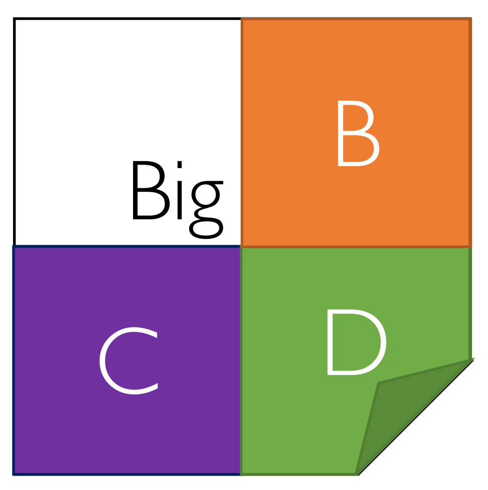
Fault Tolerant Distributed File Systems

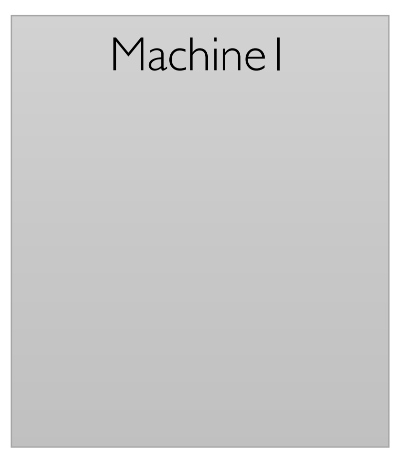


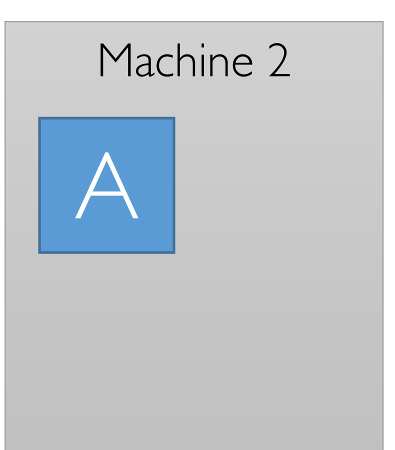
- Split the file into smaller parts.
- How?
  - Ideally at record boundaries
  - What if records are big?

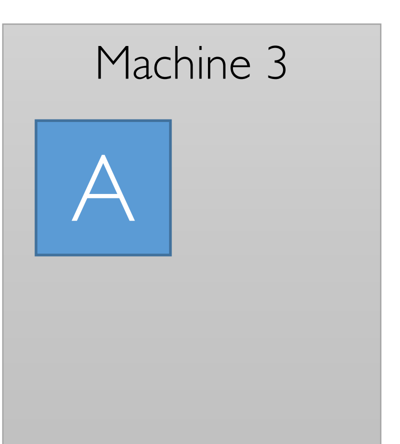
[Ghemawat et al., SOSP'03]

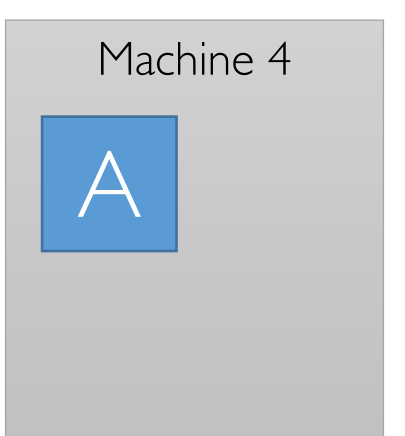
Fault Tolerant Distributed File Systems





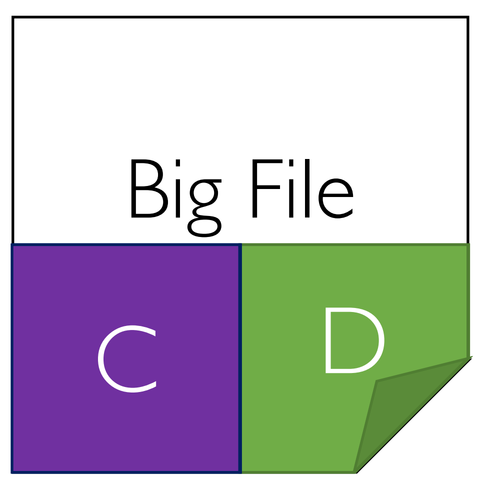


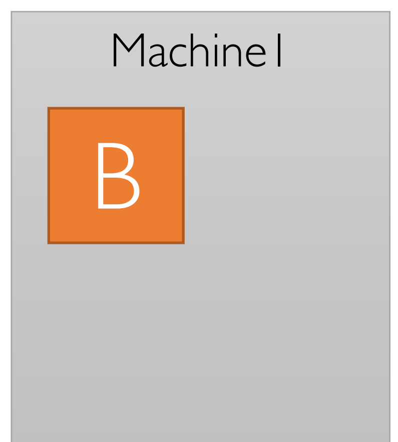


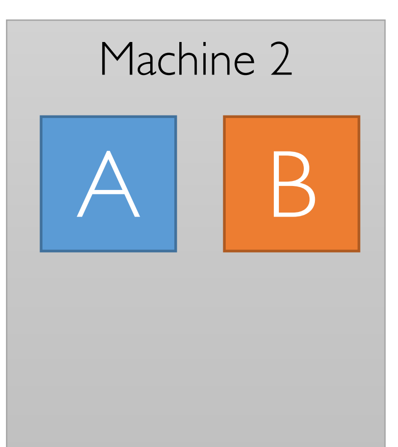


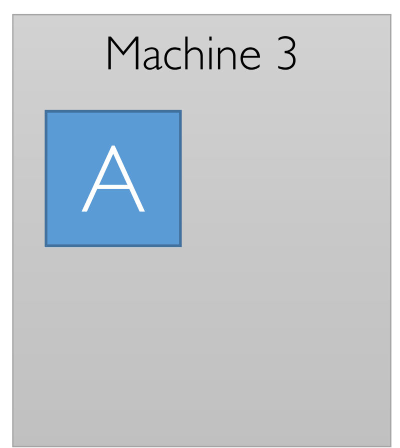
[Ghemawat et al., SOSP'03]

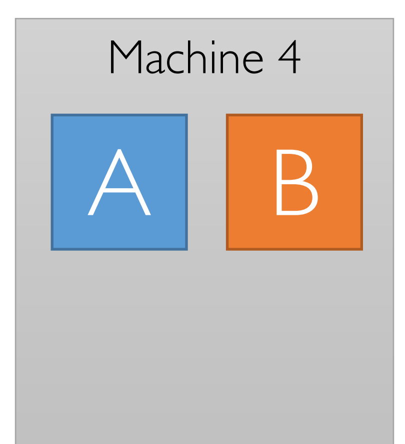
Fault Tolerant Distributed File Systems





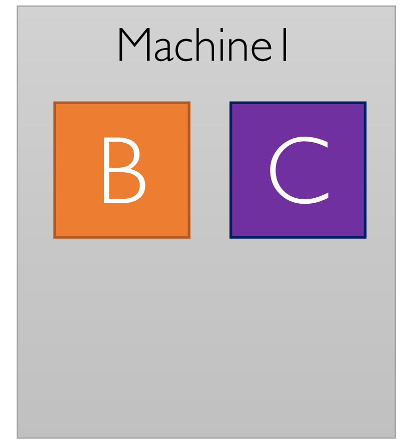


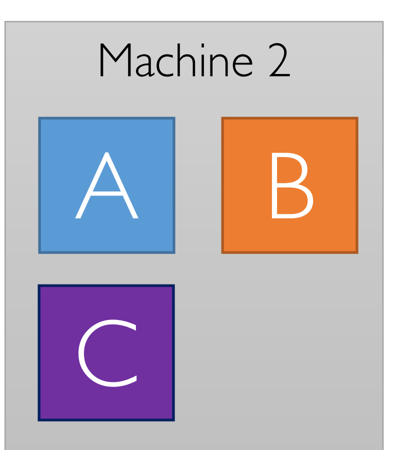


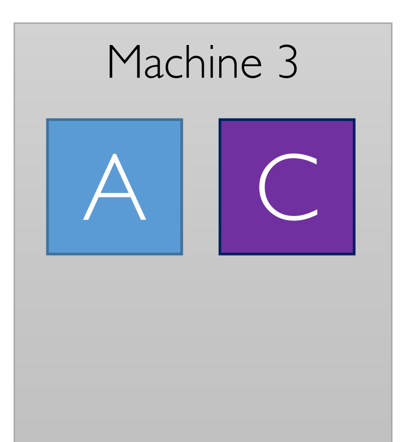


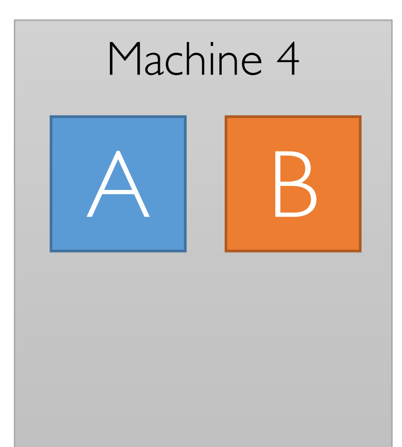
[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems



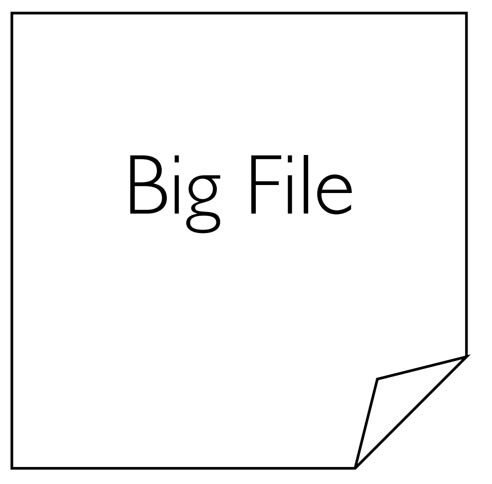


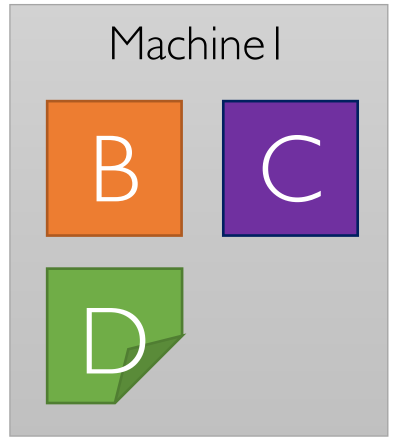


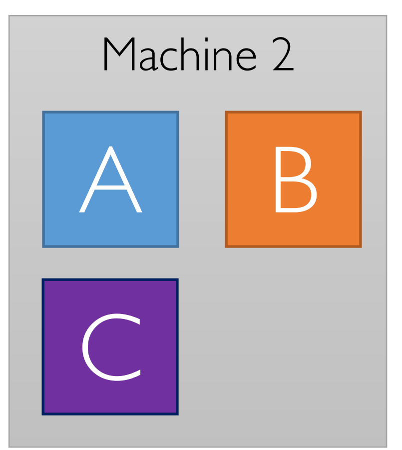


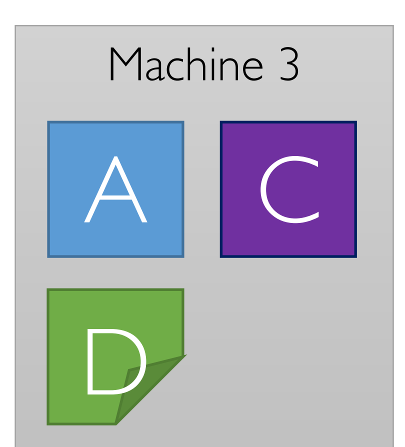
[Ghemawat et al., SOSP'03]

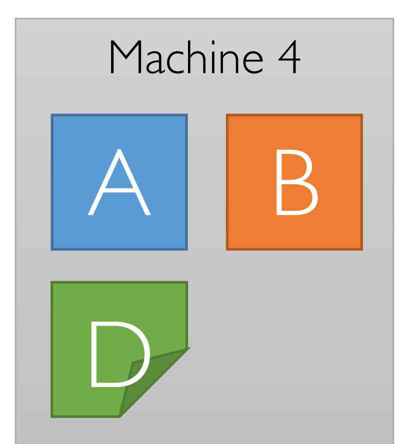
Fault Tolerant Distributed File Systems









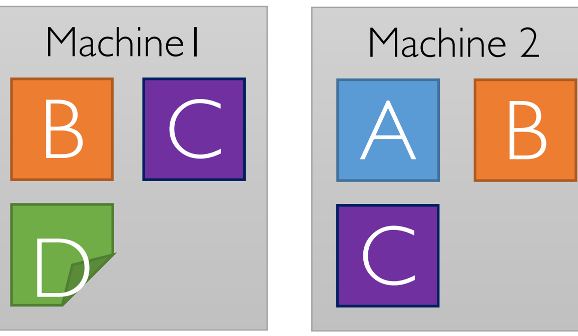


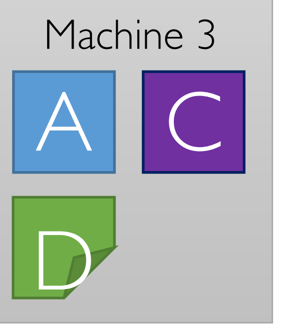
[Ghemawat et al., SOSP'03]

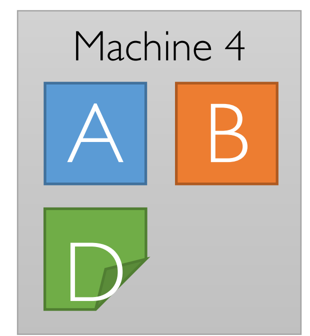
Fault Tolerant Distributed File Systems

- 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

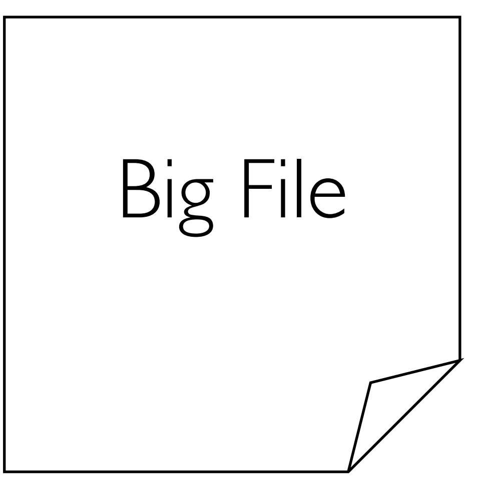
Cheap commodity storage devices will fail!

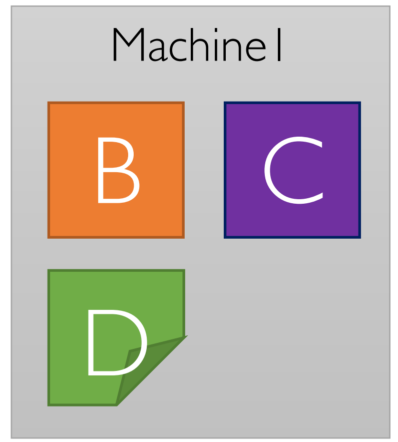


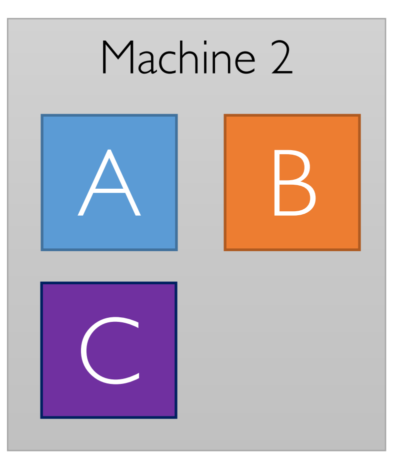


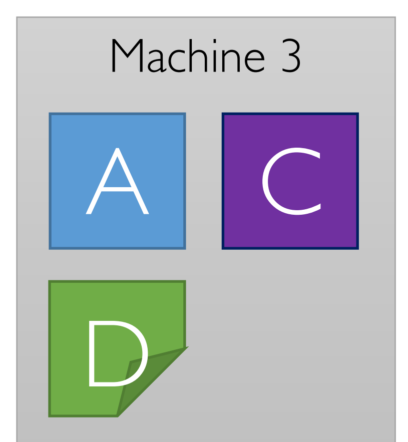


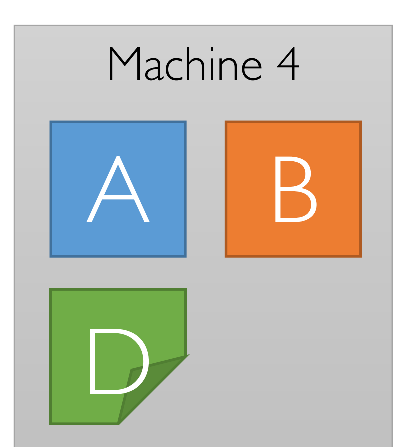
Fault Tolerant Distributed File Systems
Failure Event





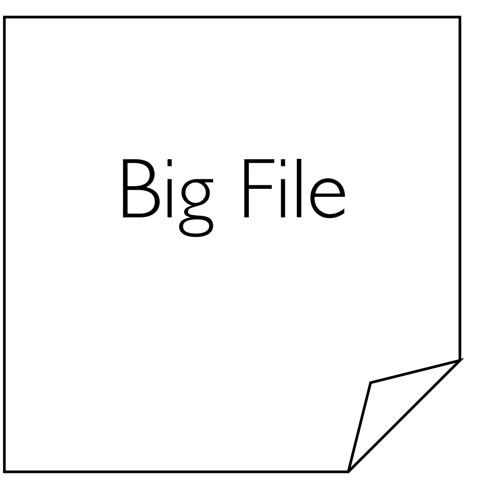


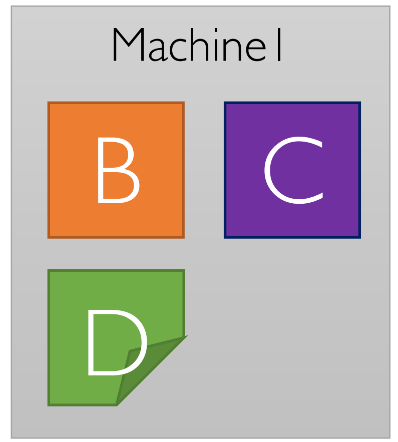


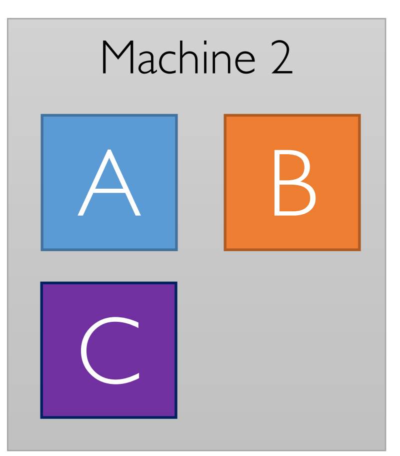


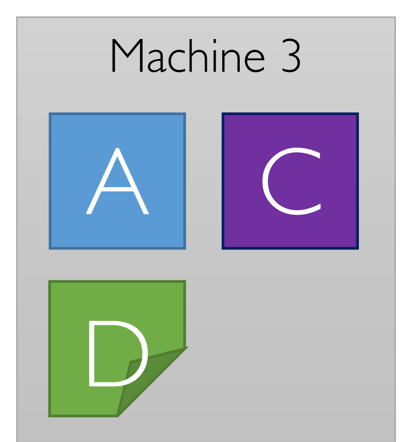
[Ghemawat et al., SOSP'03]

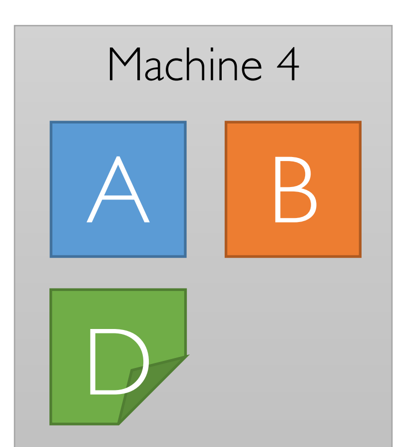
Fault Tolerant Distributed File Systems
Failure Event





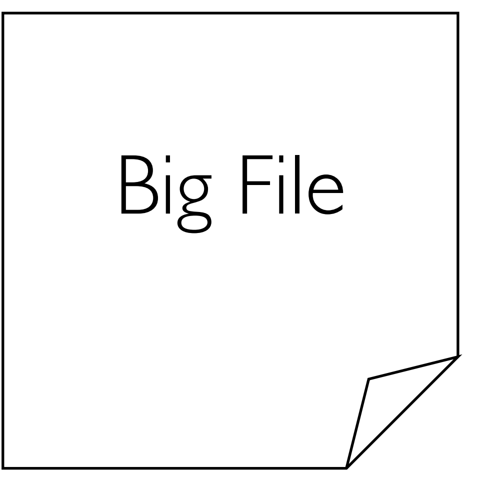


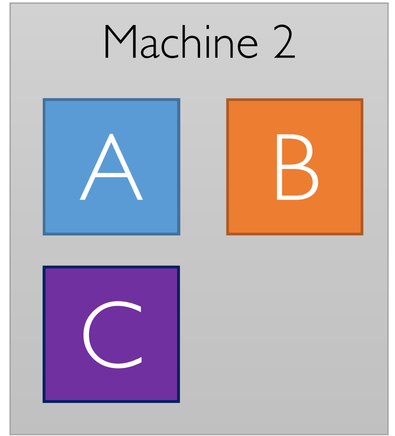


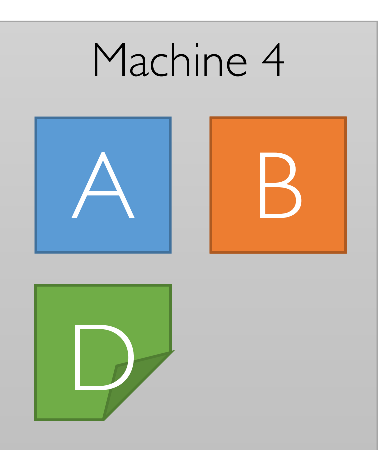


[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems
Failure Event

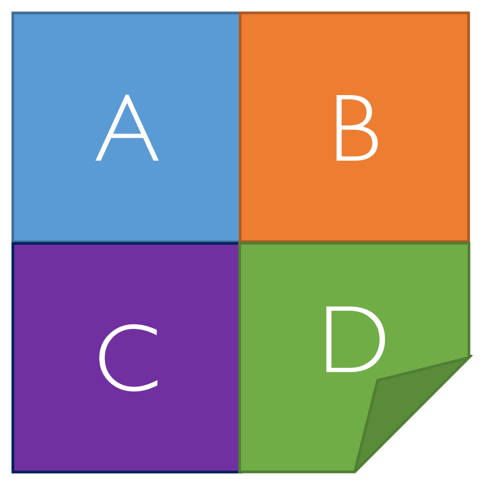


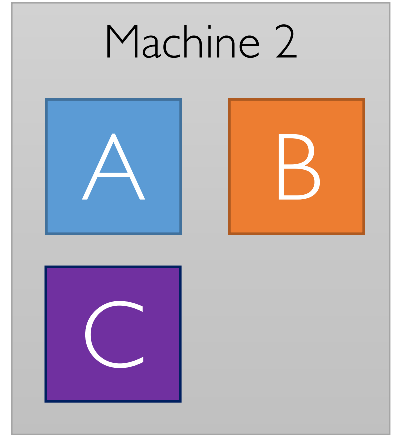


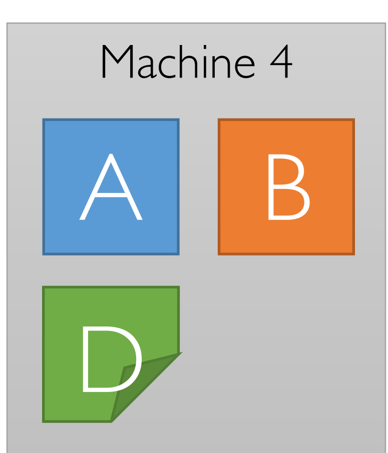


[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems
Failure Event







[Ghemawat et al., SOSP'03]

Distributed Computing

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

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 device failure

Slide provided by M. Zaharia

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

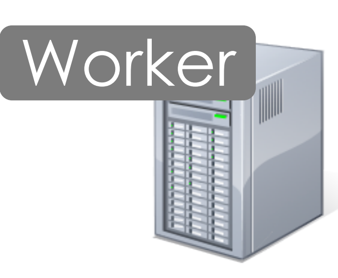
```
lines = spark.textFile("hdfs://file.txt")
```

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

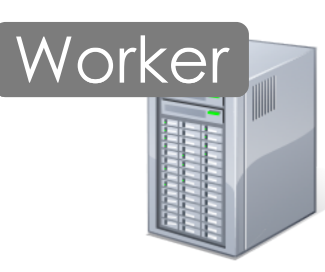
```
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
```

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()
```

Driver

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()
```

Driver

Action

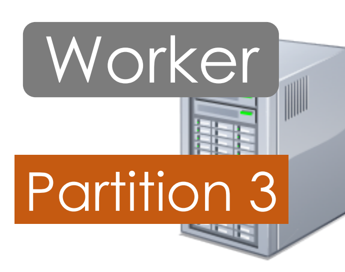
Example: Log Mining

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()
```

Driver



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()
```

Driver

tasks

Worker

Partition 1

tasks

Worker

Partition 2

tasks

Worker

Partition 3

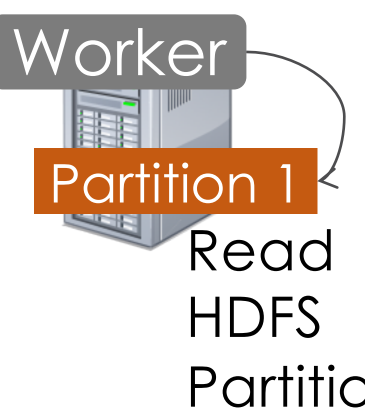
Example: Log Mining

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()
```

Driver



Worker
Partition 2
Read
HDFS

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()
```

Driver

Cache 1
Worker
Partition 1
Process & Cache Data

Cache 2
Worker
Partition 2
Process & Cache Data

Cache 3
Worker
Partition 3
Process & Cache Data

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()
```

Cache 1
Worker
Partition 1
results
Driver
results
Cache 2
Worker
Partition 2
results
Cache 3
Worker
Partition 3

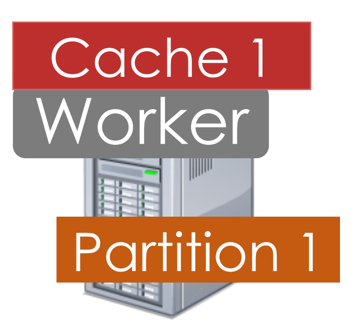
Example: Log Mining

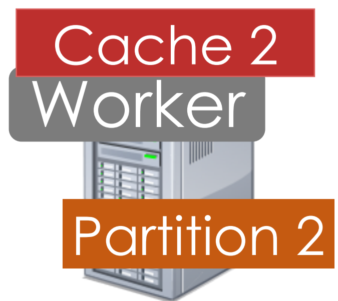
Load error messages from a log into memory, then interactively search for various patterns

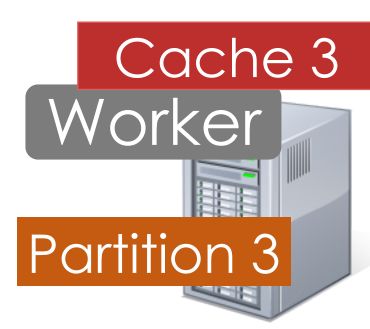
```
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

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

Driver







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()
```

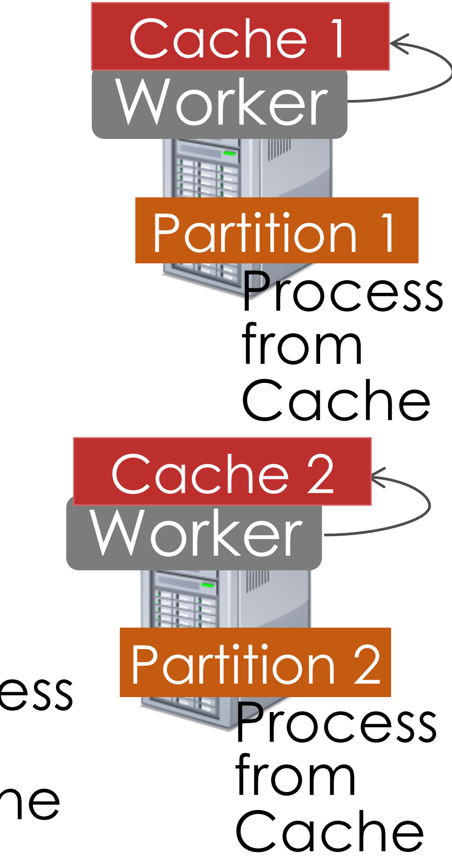
Driver

Cache 3

Worker

Partition 3

Process from Cache



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 1
Worker
Partition 1
results
Driver
results
Cache 2
Worker
Partition 2
results
Cache 3
Worker
Partition 3

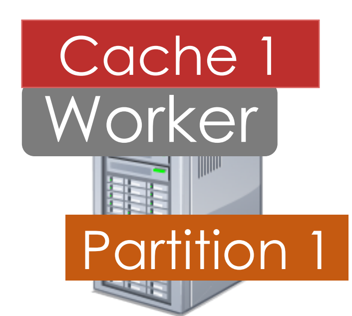
Example: Log Mining

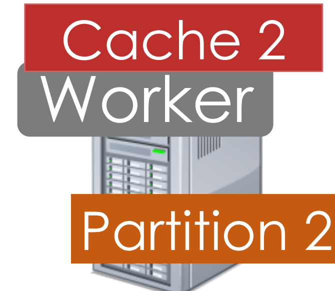
Load error messages from a log into memory, then interactively search for various patterns

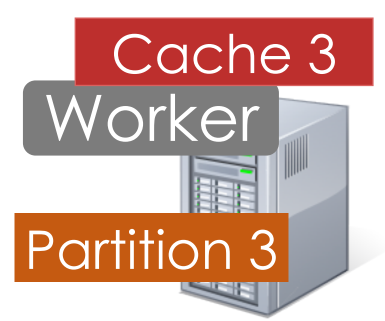
```
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

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

Driver







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

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.
- 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.