Data Management Part 2

Yusra AlSayyad
LSST Data Management
Princeton University

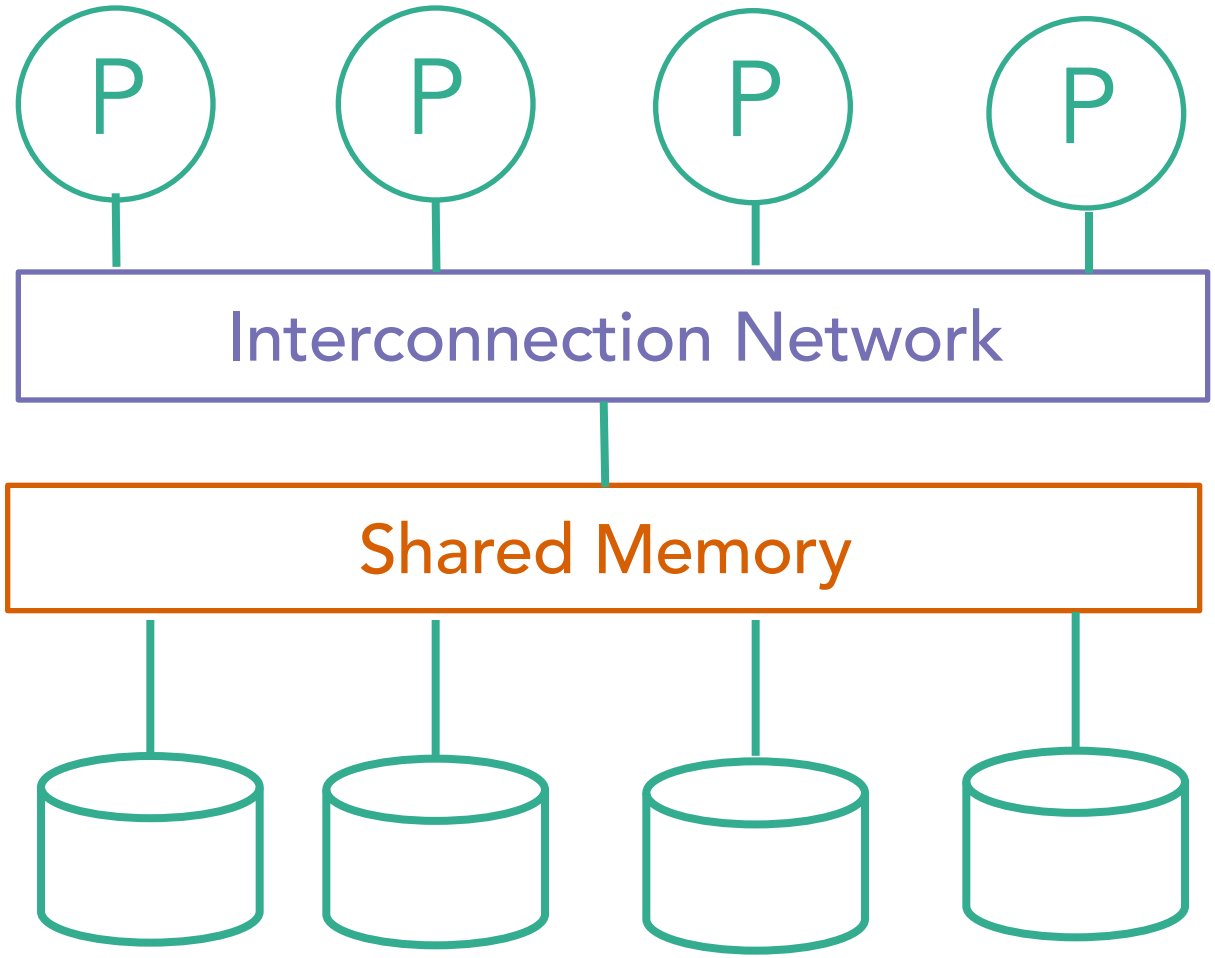
DSFP- Session 3
April 26 2017

Today: Scaling

- How do we parallelize databases?
  - Partitioning and distributing!

- Parallel RDMS
- MapReduce/NoSql

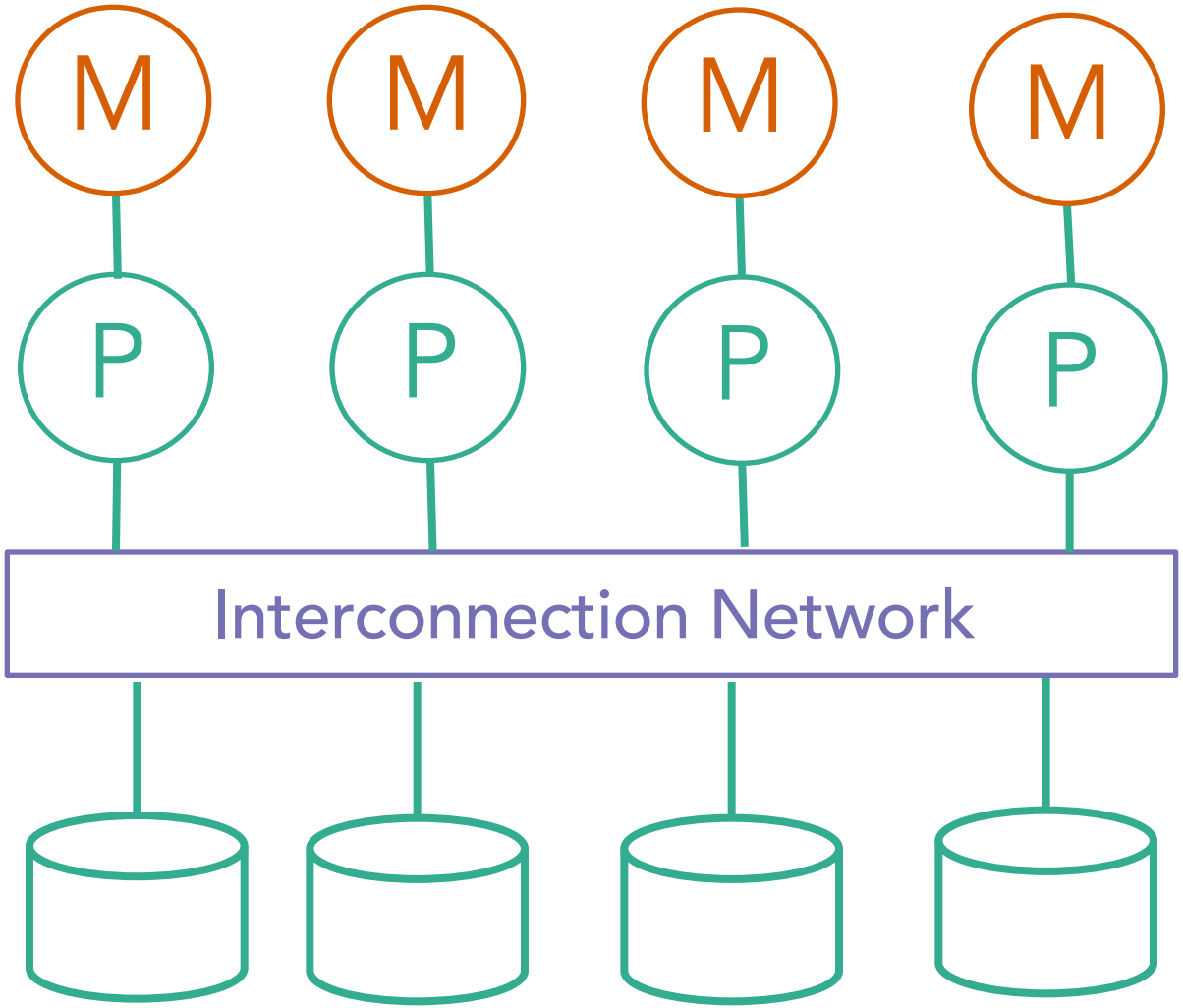
Parallel Architecture: Shared Memory



Fairly common:

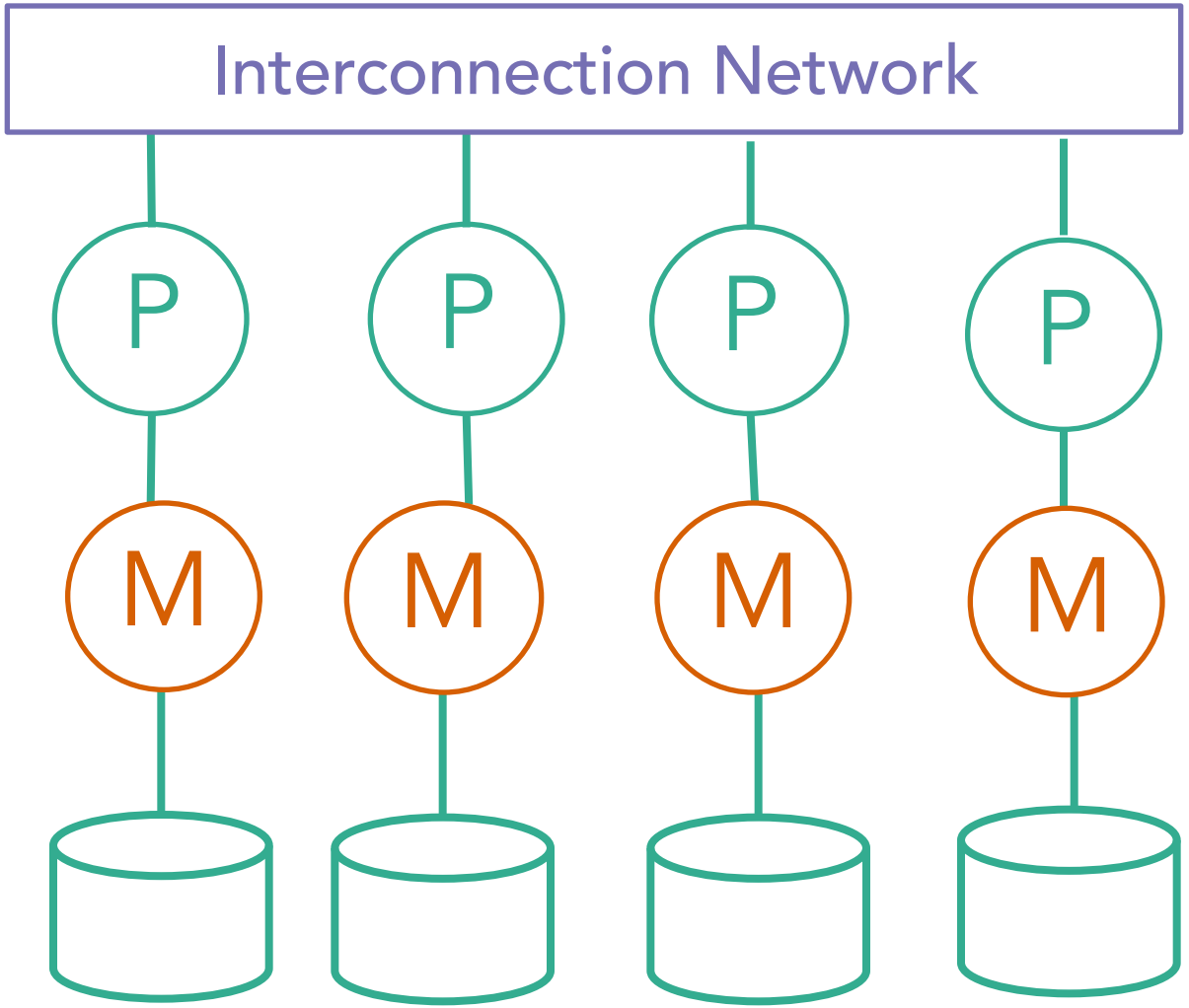
"Scaling up"

Parallel Architecture: Shared Disk



Examples:
Oracle

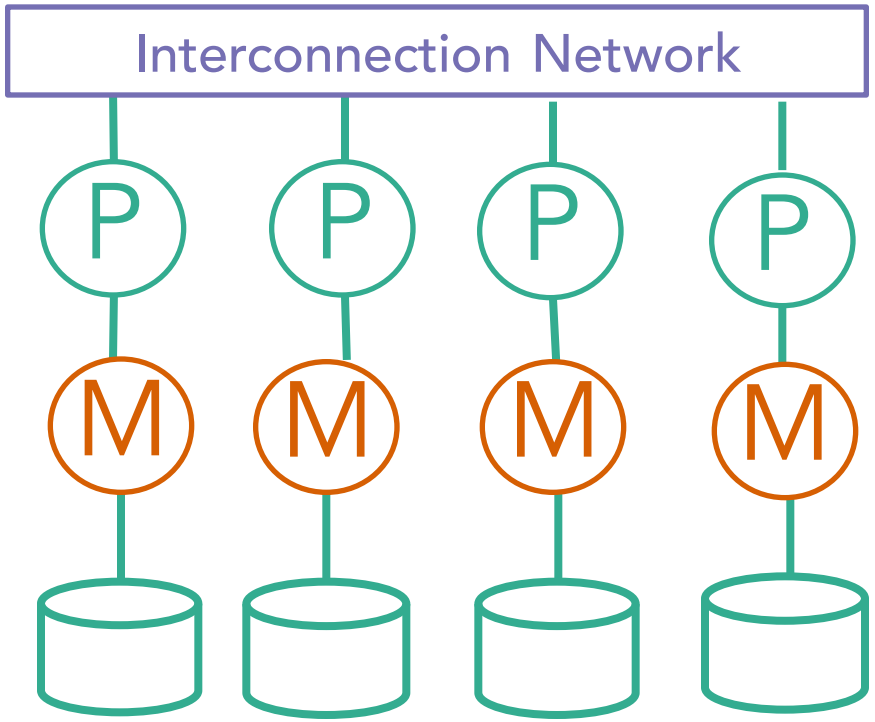
Parallel RDMS Architecture: Shared Nothing

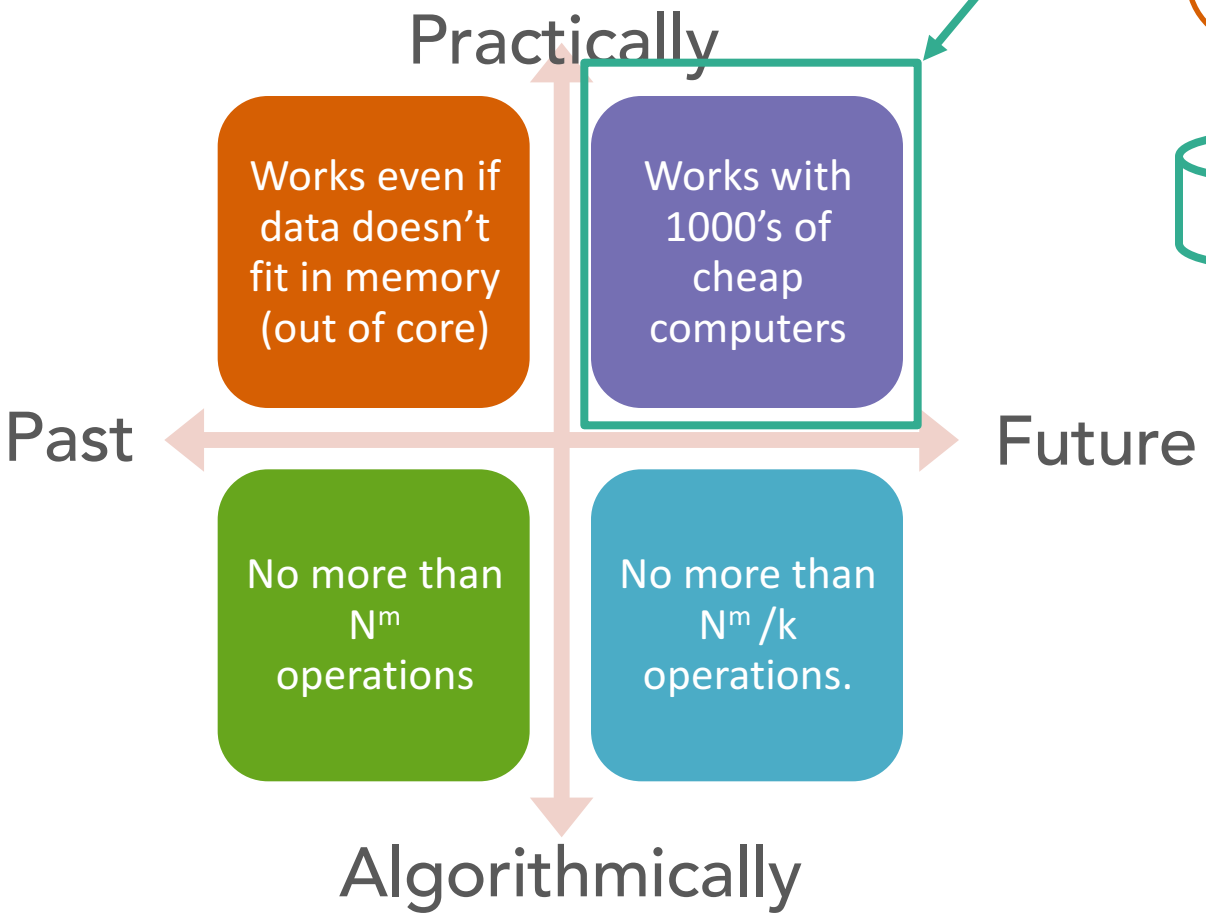


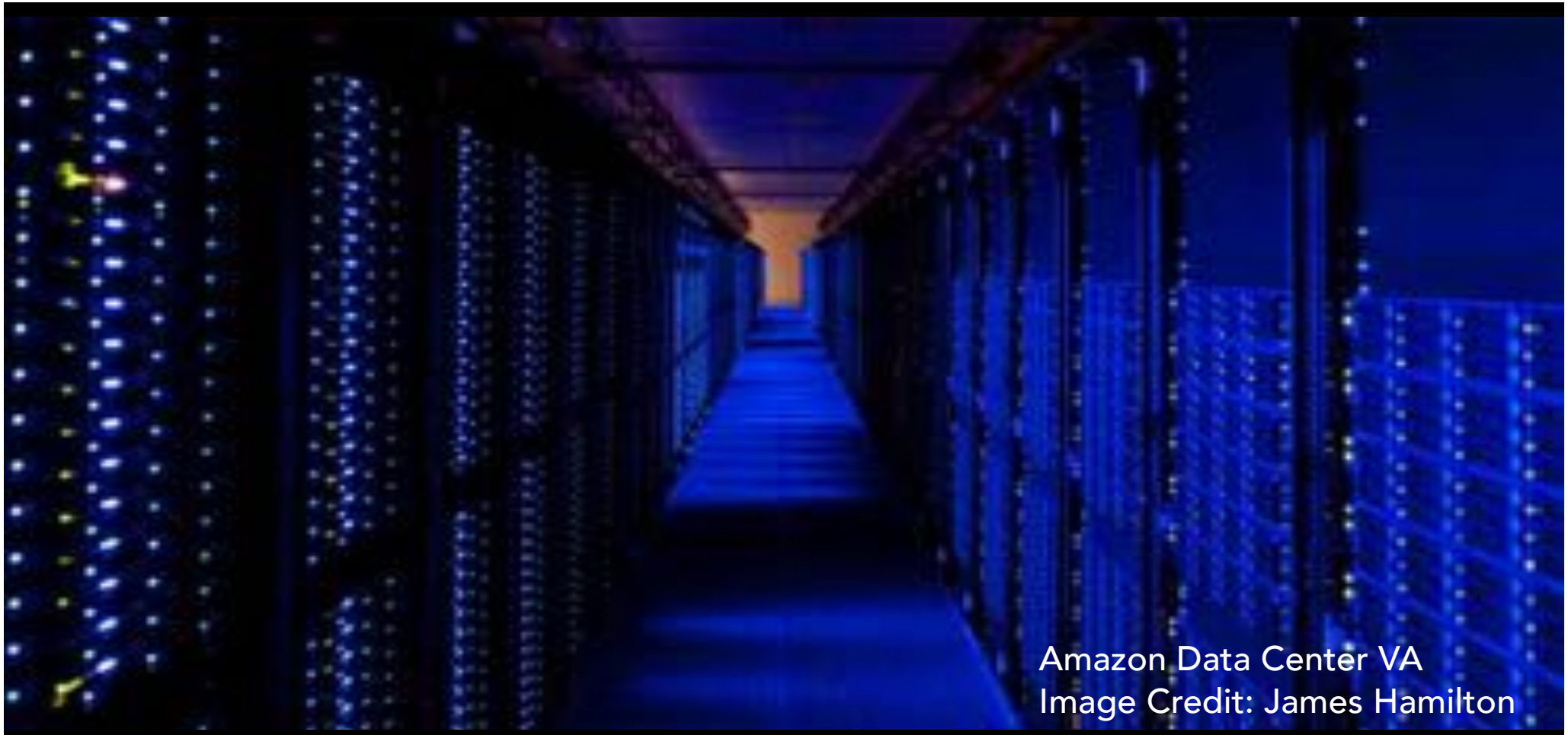
Examples:

LSST's Qserv
Greenplum
Teradata

Scalability







Amazon Data Center VA
Image Credit: James Hamilton

Parallel Databases

Example: LSST's Qserv

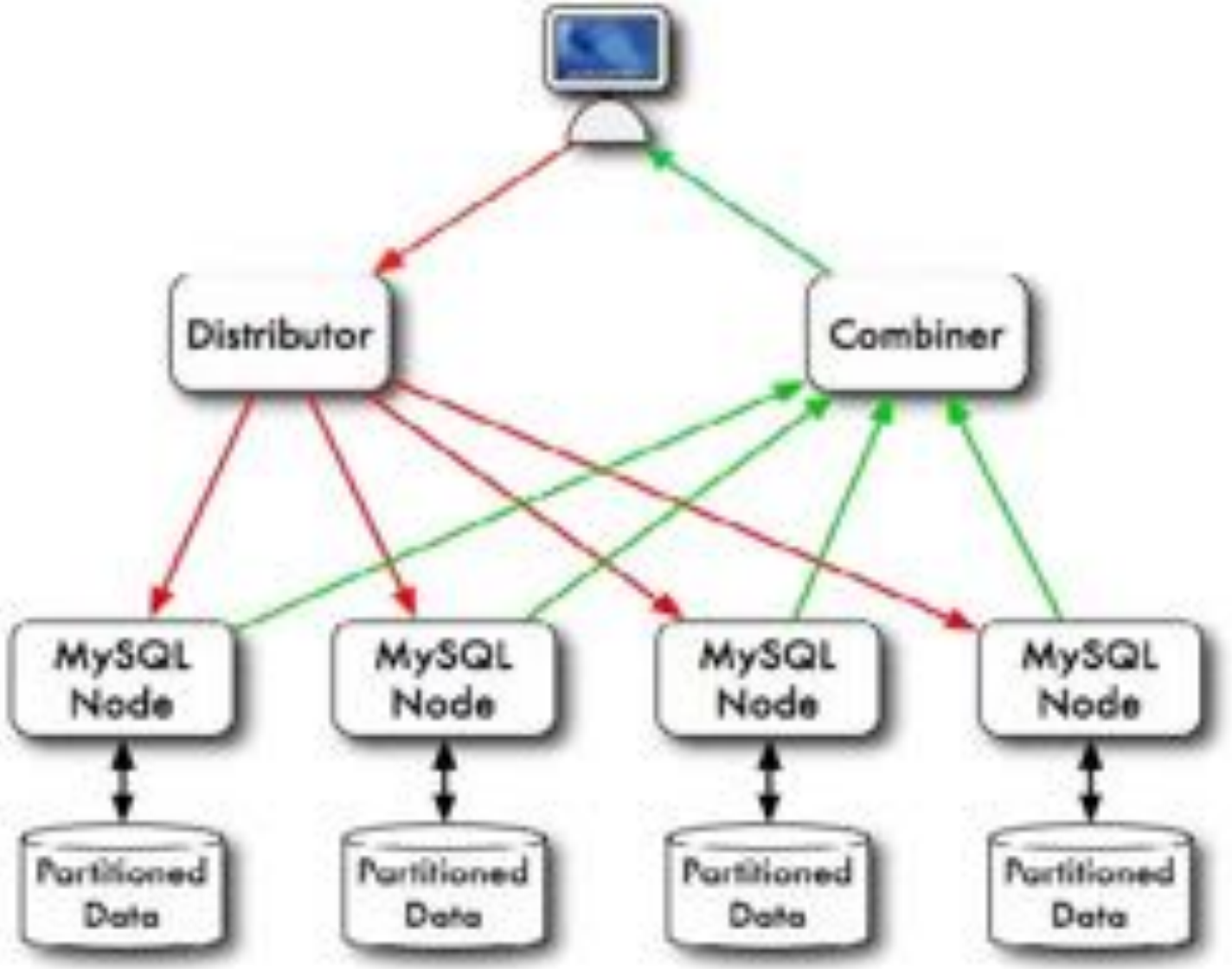
SLAC NATIONAL ACCELERATOR LABORATORY

- LSST's Database Requirements:
  - Incrementally Scalable
  - Reliable and Available
  - **Low-cost** ← No teradata/Greenplum. Must be all open source.
  - Table Scans in (< 1hour)
  - Low latency (< 10 seconds for small area retrieval) ← No MapReduce
  - Spatial self-joins (< 24hr) and cross-matching

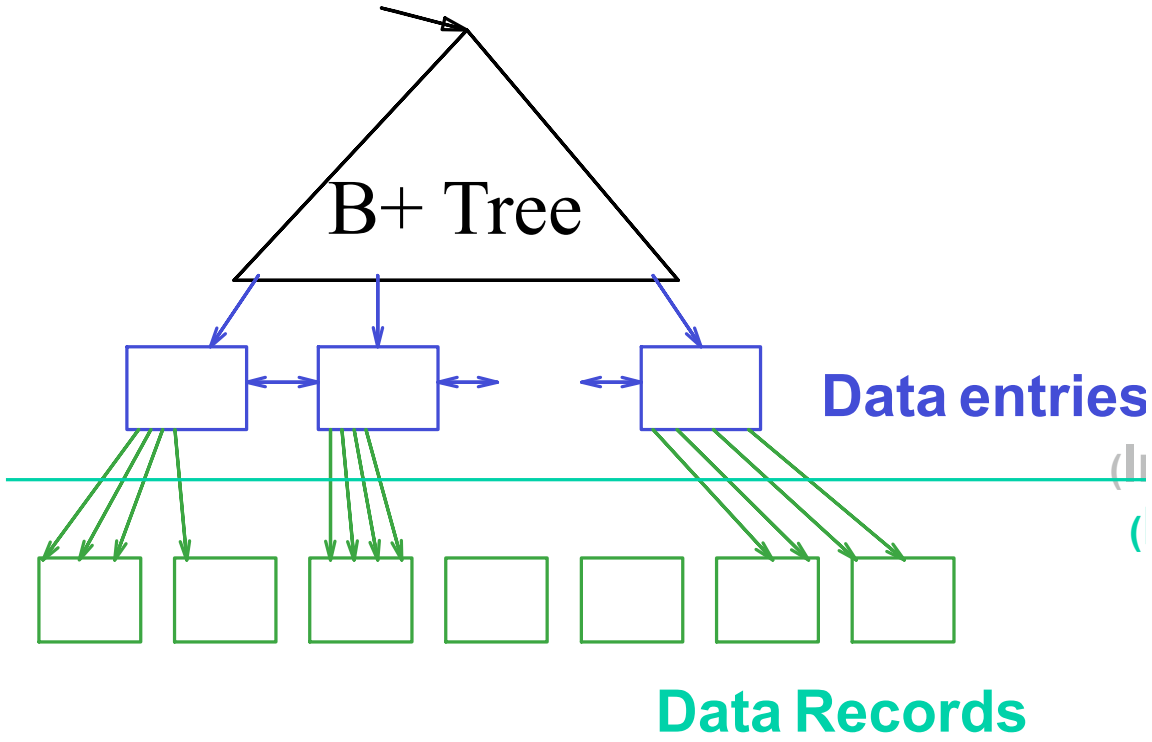
LSST's final data release

Table name	# rows	row size	footprint
Object	38×10^9	1.7kB	64TB
Object_Extra	38×10^9	26kB	1PB
Source (detections)	6.3×10^{12}	0.56kB	3.5PB
ForcedSource (expected det.)	38×10^{12}	40B	1.5PB

Qserv Architecture: Shared nothing



Partitioning
Recall Indexes from yesterday:



CLUSTERED

Image credit: UW CSE

Fun Features of Qserv

- Shared Scanning
  - a.k.a. Convoy scheduling shares the I/O from each scan with multiple queries
- Partitions

NoSql Databases

Problem: So you want to run a job that takes 4 hours on a 10000-node cluster?

- Order of magnitude estimate:
  - A server will fail once per year (8760 hours)
  - => Approximately one node failing every hour.

- Main objective: Fault tolerance

Google developed the Google File System and Map Reduce to solve this

- 2004: Jeff Dean publishes MapReduce paper describing system and framework

- 2008 Open source implementation: hadoop
  - Hadoop 0.17 release (Yahoo → Apache)

  - Runs on Hadoop Distribute File System (HDFS):
    - a distributed file-system
    - Duplicates data 3X for fault tolerance

Map Reduce Framework (Hadoop)

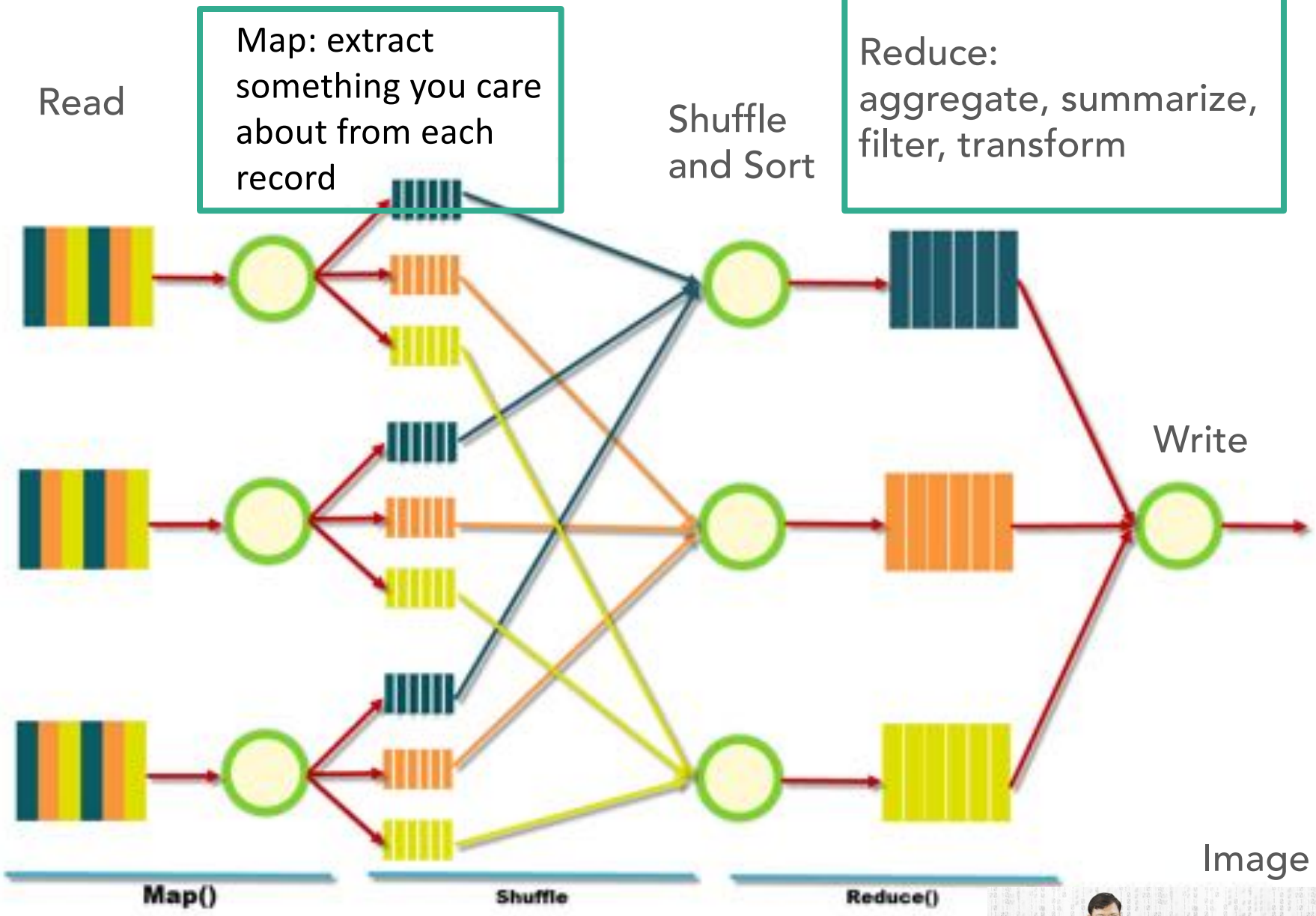


Image Credit:

Data Model

- A file = a bag of **(key, value)** pairs

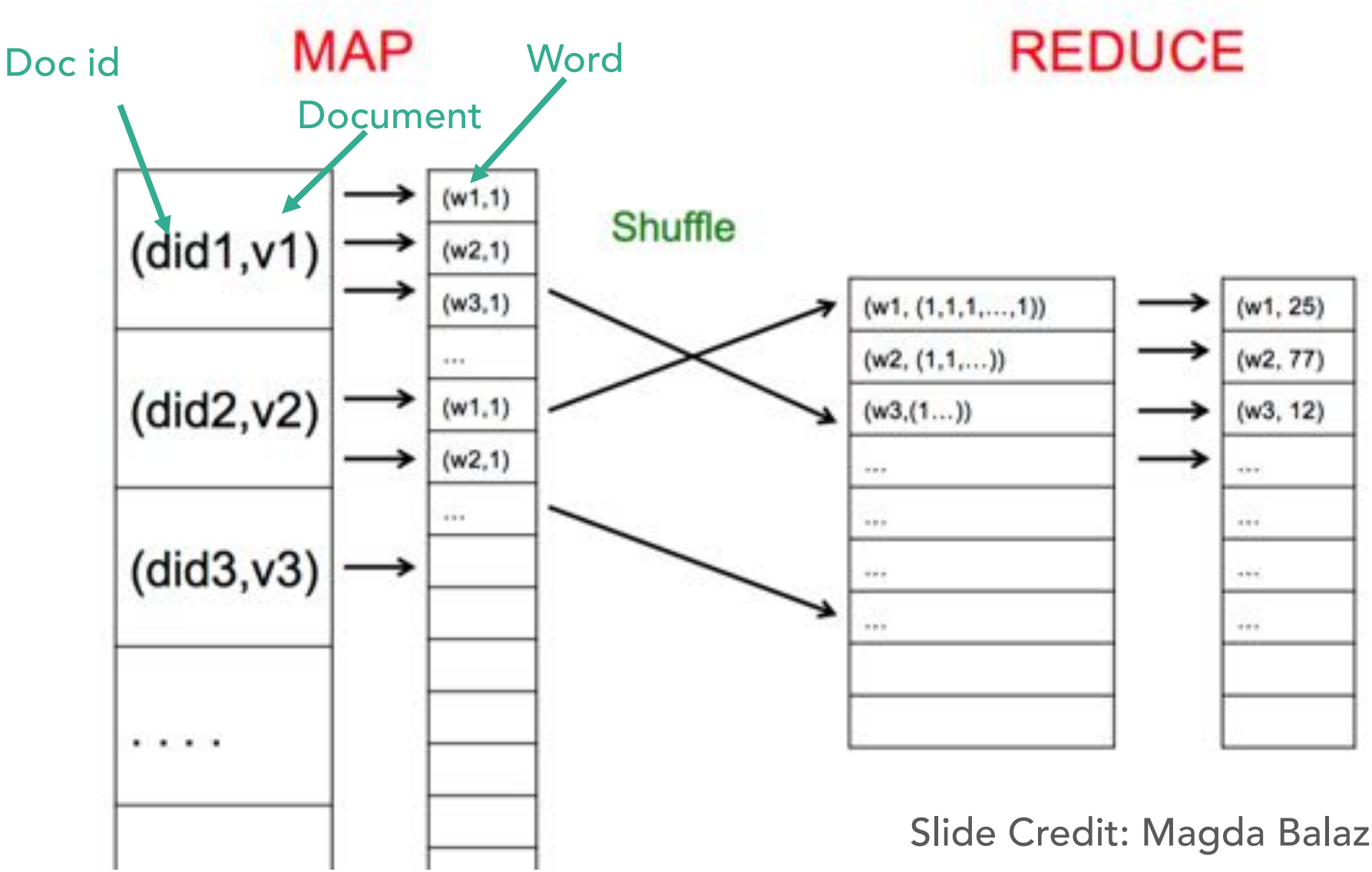
- A MapReduce program:
  - Input: a bag of **(input key, value)** pairs
  - Output: a bag of **(output key, value)** pairs

Map Reduce Example: Word Count

```
def mapper(key, value):
    # key: document identifier
    # value: document contents
    words = value.split()
    for w in words:
        emit_intermediate(w, 1)

def reducer(key, list_of_values):
    # key: word
    # value: list of occurrence counts
    total = 0
    for v in list_of_values:
        total += v
    emit((key, total))
```

Map Reduce Example: Word Count



Slide Credit: Magda Balazinska

Since then…

- Dean et al. 2004 MapReduce
- 2008 Hadoop 0.17 release
- Olston et al. 2008 Pig Relational Algebra on Hadoop
- 2009 Thusso et al. HIVE SQL on Hadoop
- 2009 Hbase: Indexing for Hadoop
- 2010 Dietrich et al. Schemas and Indexing for Hadoop
- 2012 Transactions in Hbase

Adapted from Slide by:
Bill Howe UW eScience

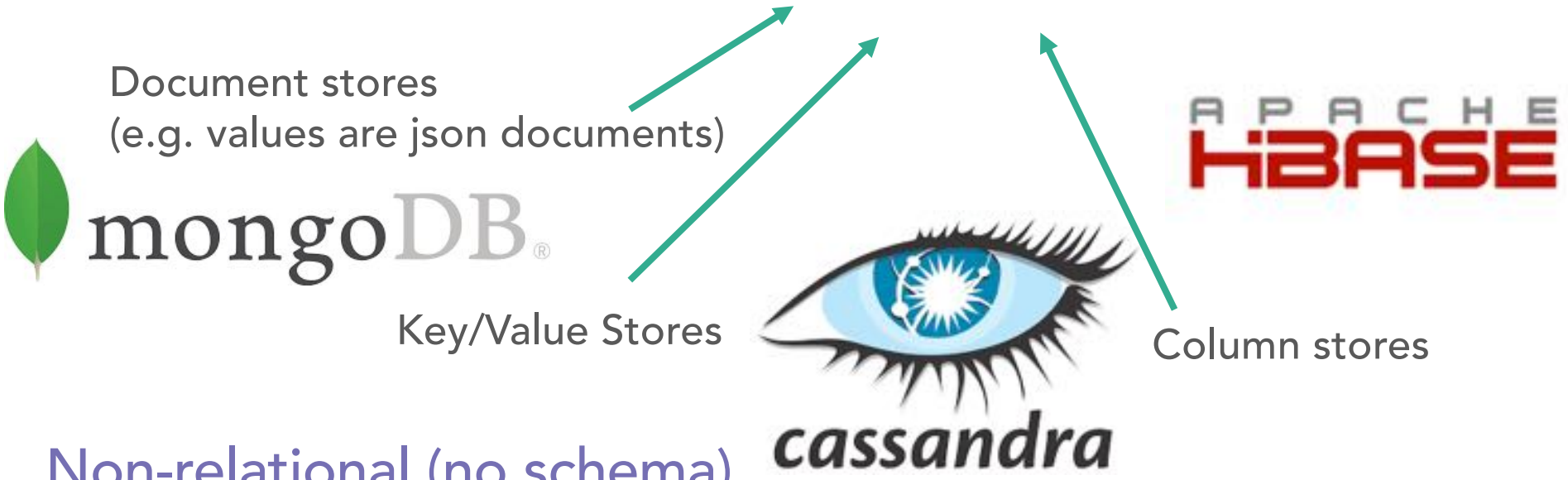
Since then… (in another direction)

- Zaharia et al 2012: Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

We will talk about this on Friday

What do all these NoSql systems have in common?

Data model: aggregate-oriented



Non-relational (no schema)
Mostly open-source
Cluster-friendly/fault tolerant
Evolved from Web needs

amazon DynamoDB

Aggregate-oriented data model more closely matches Object model (from OO-programming)

- Aggregate-orientated data model has tradeoffs:
  - Advantage if you're using the same aggregate, but
  - Disadvantage if you want to slice and dice.

```
Person1:
 publicationList = ["stars"]
 Institution = ["University of Arizona"]

Person 2:
 publicationList = ["stars", "galaxies paper"]
 phoneNumber =["1234567890"]
```

No Atomicity, Concurrency, Isolation, Durability

- "Eventually consistent concurrency" instead.

- "CAP Theorem" says: You can only have 2:
  - Consistency
  - Availability
  - Partitioning

- Features:
  - fault tolerance
  - Schema-on-read ←Trade offs
  - Easier to write user-defined functions

- Why bad for Qserv?
  - HBase no joins
  - No indexing on Hive
  - Overhead on Hadoop (latency)
  - Catalog has a well-defined relational model

Putting it together

- Data Management Considerations:
  - How much data?
  - How many people to share with?
  - How much time/money to invest?
  - Data model? Etc…

amazon web services

XSEDE
Extreme Science and Engineering Discovery Environment