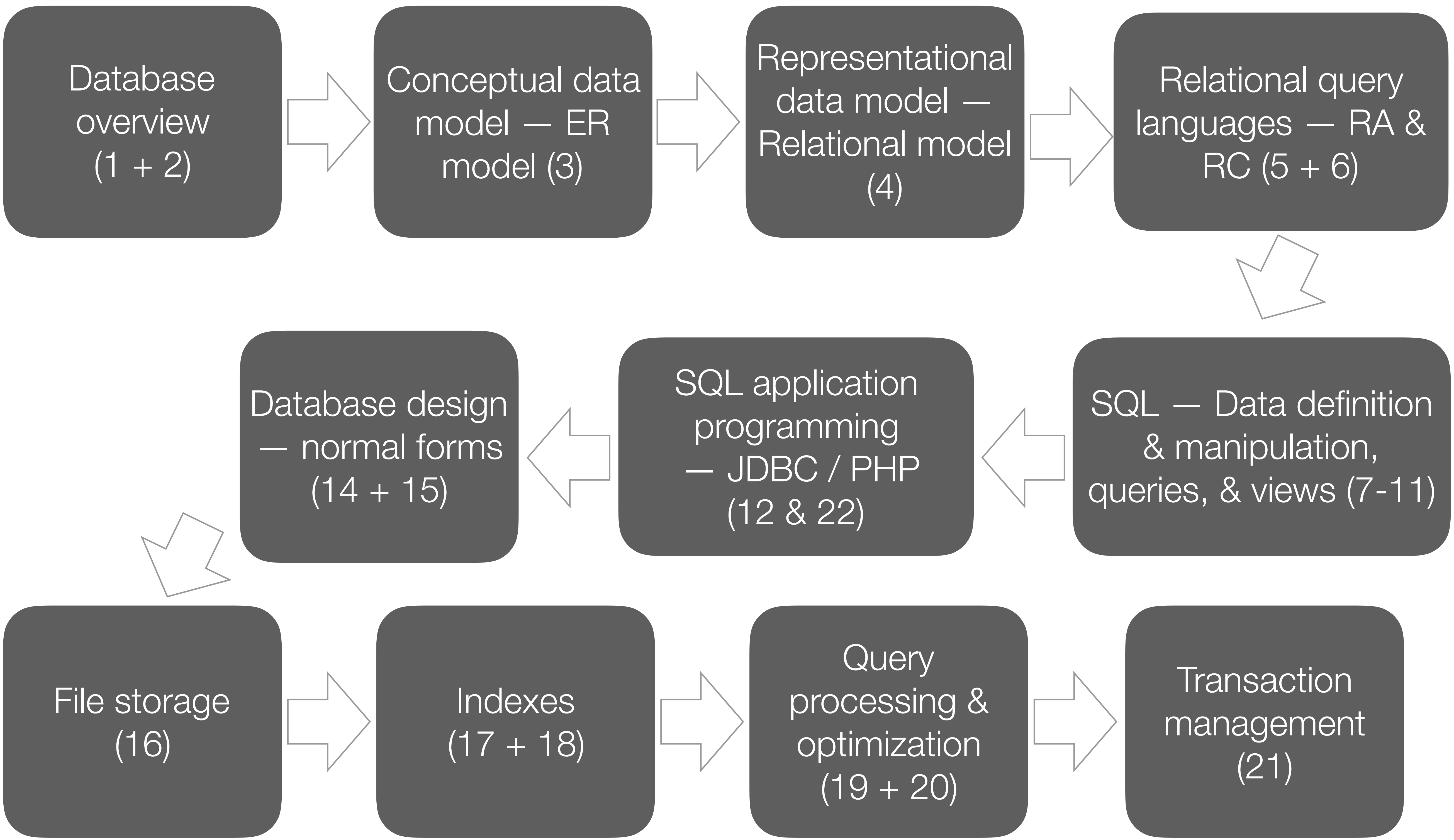
Big Data Systems

CS 377: Database Systems

Recap: What Has Been Covered



Recap: What Has Been Covered

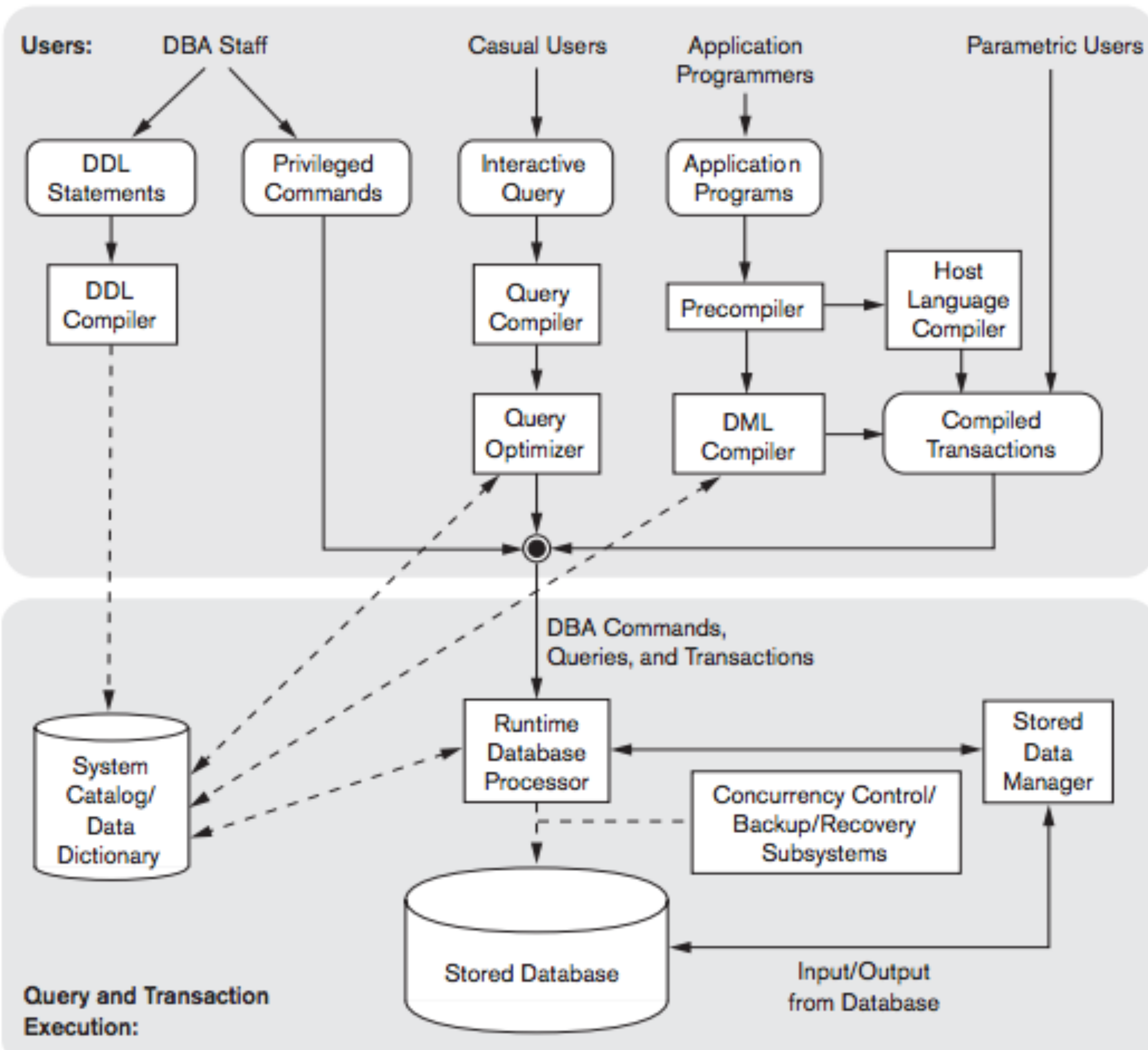
What I hope you've learned…

- Design a database
  Requirements -> ER diagram -> Relational model -> Database normalization

- Query a database (even with concurrent users and crashes)
  Relational algebra, calculus, SQL queries, transactions

- Optimizing the performance of your database (at a high level)

Recap: DBMS Architecture



- Most aspects of traditional database system is “understood” (high-level)
- Learned enough to be “dangerous”
- Additional details can be picked up
  - Courses
  - On your own

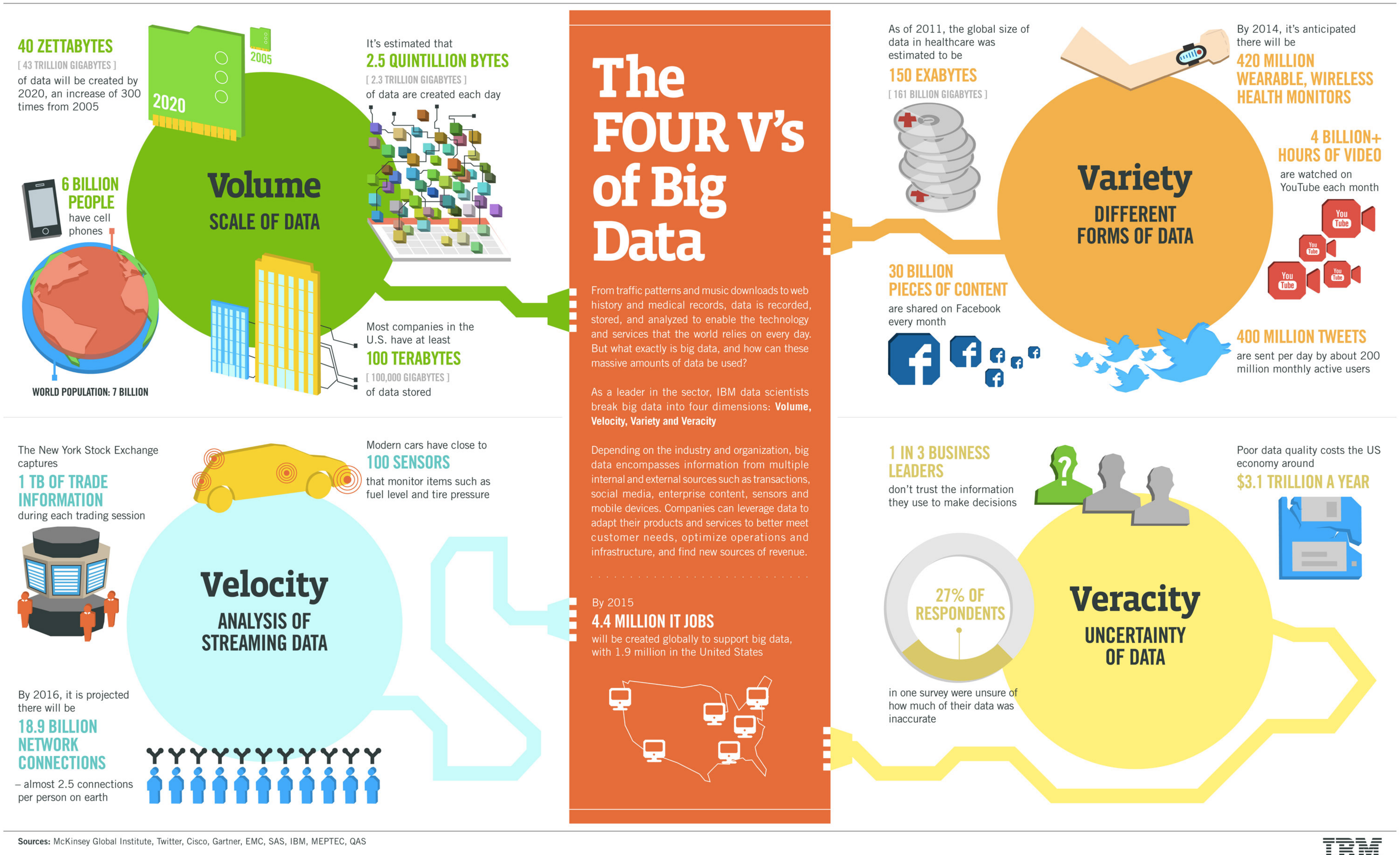
Goal of Today's Lecture

- High-level overview of dealing with "big data"
  - What is big data?
  - What are different technologies I can use?
- Not meant to be detailed examination of all aspects of systems covered

Data Never Sleeps

https://www.domo.com/blog/2015/08/data-never-sleeps-3-0/

4 V's of Big Data



http://www.ibmbigdatahub.com/infographic/four-vs-big-data

Motivation for Parallel / Distributed DBMS

- Single, monolithic DBMS is impractical and expensive
  - Constantly need to move data to deal with storage
- Improve performance
- Increased availability & reliability
- Potentially lower cost of ownership
- Easier, more economical system expansion

Parallel/Distributed DBMS

- Data partitioned across multiple disks
  - Allows parallel I/O for better speed-up
- Individual relational operations (e.g., sort, join, aggregation) can be executed in parallel
  - Each processor can work independently on its own partition
- Queries can be run in parallel with each other
  - Concurrency control takes care of conflicts

Parallel vs Distributed

- Parallel DBMS:
  - Nodes are physically close to each other
  - Nodes connected via high-speed LAN
  - Communication cost is small
- Distributed DBMS
  - Nodes can be far away
  - Nodes connected via public network
  - Communication cost and problems shouldn’t be ignored

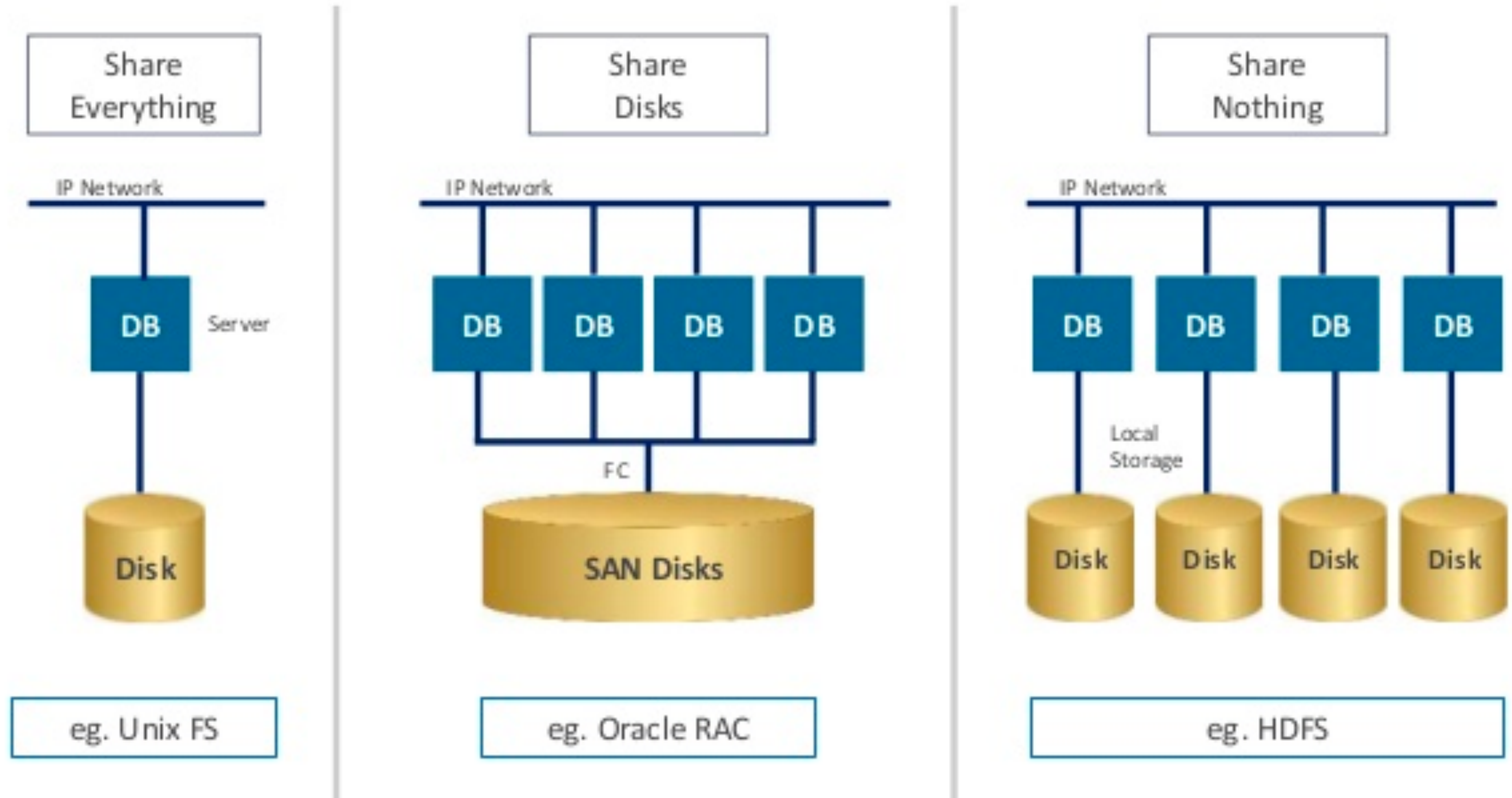
Parallel/Distributed DBMS Issues

- How to distribute the data
- How to optimize the cost of queries
  - Data transmission + local processing
- How to perform concurrency control
- How to make system resilient to failures and achieve atomicity & durability

Scale-Up vs Scale-Out

- Terminology to measure performance
- Speed-up: using more processors, how much faster will the task run (assuming same problem size)?
- Scale-up: using more processors, does performance remain the same as we increase problem size?
- Scale-out: using a larger number of servers, does performance improve?

Parallel Architectures

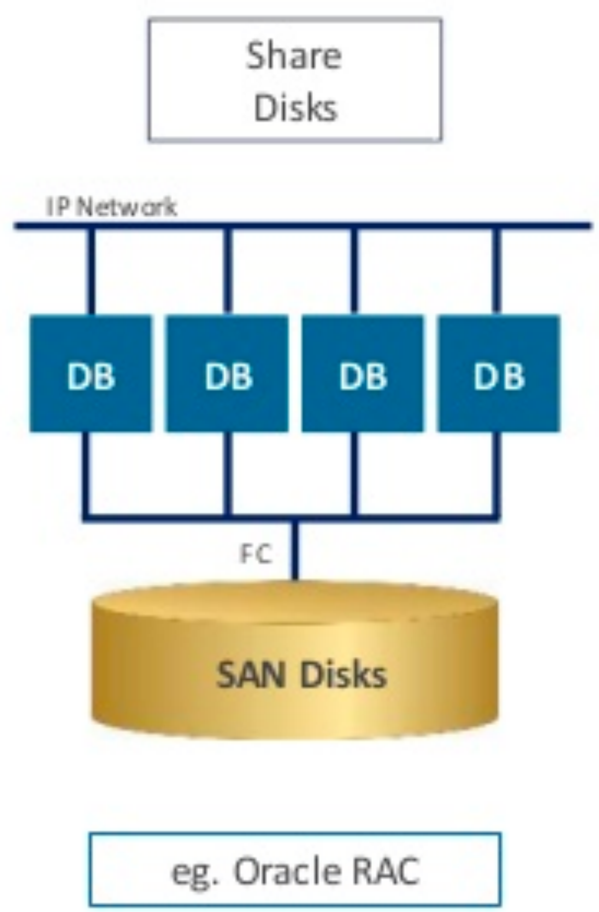


http://image.slidesharecdn.com/hadooparchitecture-091019175427-phpapp01/95/big-data-analytics-with-hadoop-18-638.jpg?cb=1411533606

Shared Everything

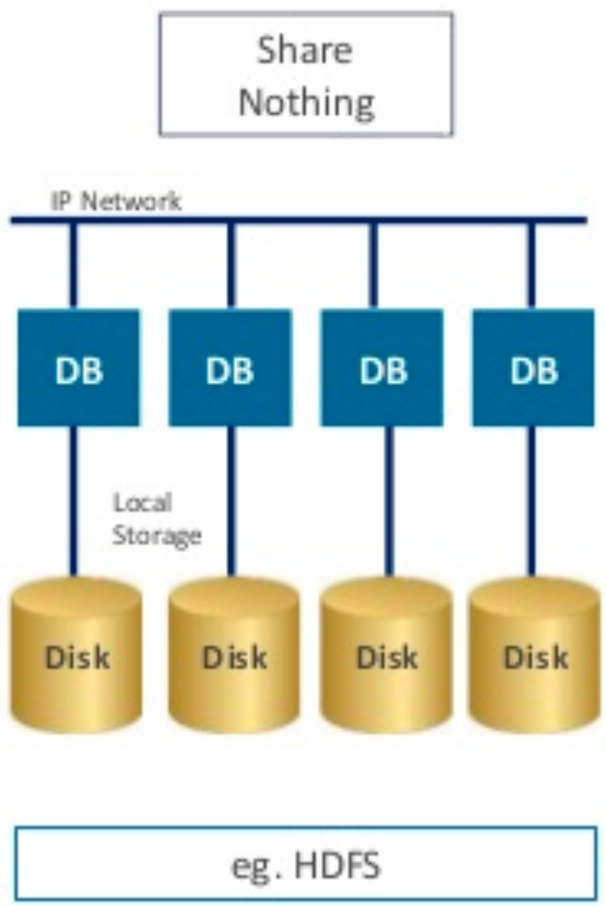
- Nodes share RAM + disk
- Dozes to hundreds of processors
- Easy to use and program
- Expensive to scale — last remaining cash cow in the hardware industry
- Example: SQL server running on single machine but leverage many threads to get a query to run faster

Shared Disk



- Nodes share same disk
- Easy fault tolerance + easy consistency (single copy of DB)
- Hard to scale past a certain point — existing deployments typically have fewer than 10 machines
- Example: Oracle servers use this paradigm quite a bit

Shared Nothing



- Each instance has its own CPU, memory, and disk
- Connected via fast network
- Easy to increase capacity
- Hard to ensure consistency
- Most scalable architecture
- Most difficult to administer and tune!

How to Distribute the Data?

- Replication: system maintains multiple copies of data, stored in different sites for faster retrieval and fault tolerance
  - (PRO) Improves availability, parallelism, and reduced data transfer
  - (CON) Increased cost of updates, complexity of concurrency control
- Fragmentation: relation is partitioned into several fragments stored at distinct sites
- Combination of both replication & fragmentation

Example: Replication & Fragmentation

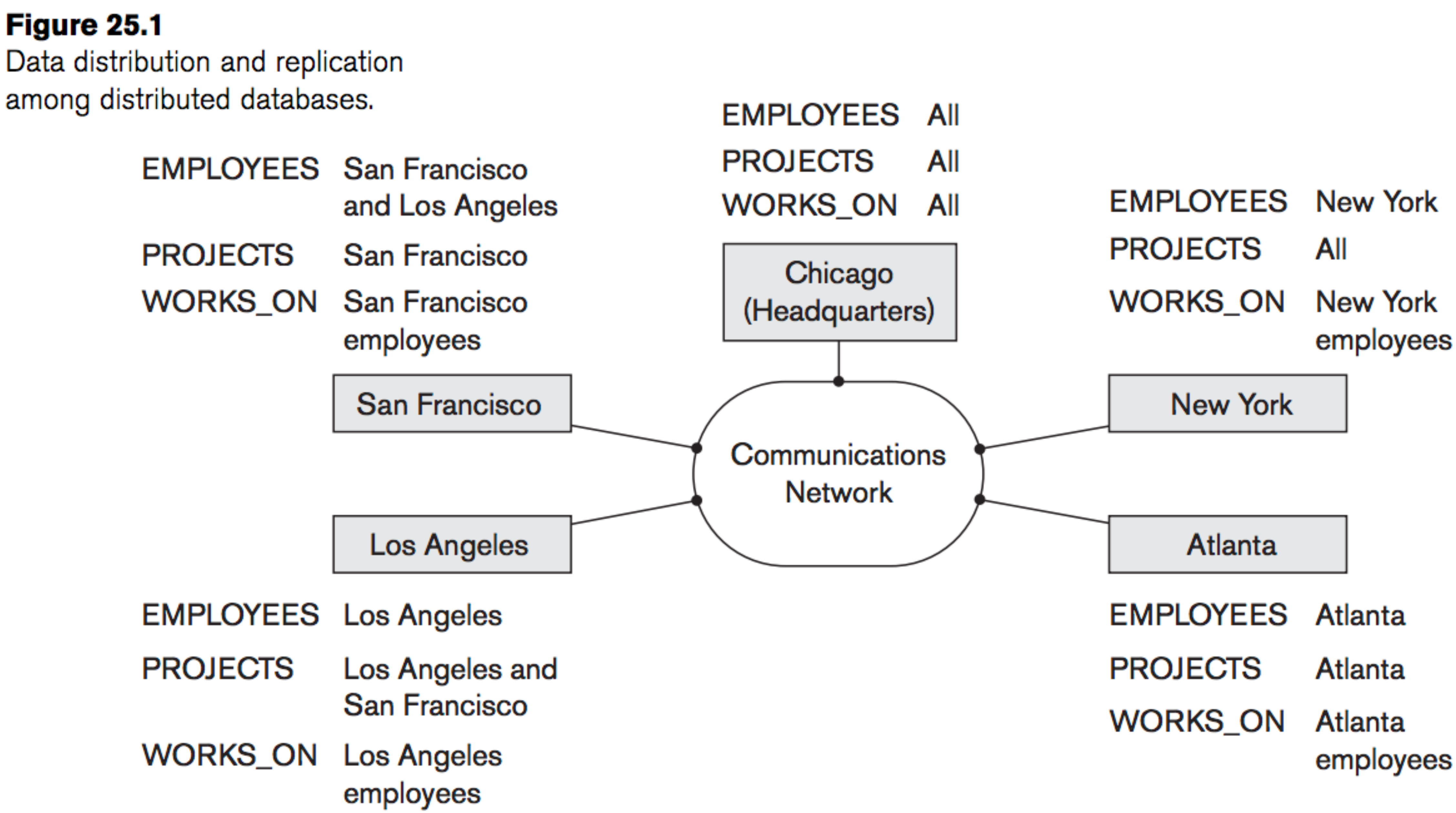


Figure 25.1 from FoDS book

Fragmentation Strategies

- Horizontal partition: each tuple is assigned to one or more fragments
  - Round robin
  - Hash partitioning
  - Range partitioning
- Vertical partition: relation is split into smaller schemas each with a common candidate key to ensure lossless join

Example: Horizontal Partition

branch_name	*account_number*	*balance*
Hillside	A-305	500
Hillside	A-226	336
Hillside	A-155	62

$$account_1 = \sigma_{branch\_name=\text{"Hillside"}}(account)$$

branch_name	*account_number*	*balance*
Valleyview	A-177	205
Valleyview	A-402	10000
Valleyview	A-408	1123
Valleyview	A-639	750

$$account_2 = \sigma_{branch\_name=\text{"Valleyview"}}(account)$$

http://www.db-book.com/

Example: Vertical Partition

branch_name	*customer_name*	*tuple_id*
Hillside	Lowman	1
Hillside	Camp	2
Valleyview	Camp	3
Valleyview	Kahn	4
Hillside	Kahn	5
Valleyview	Kahn	6
Valleyview	Green	7

$deposit_1 = \Pi_{branch\_name,\ customer\_name,\ tuple\_id}(employee\_info)$

account_number	*balance*	*tuple_id*
A-305	500	1
A-226	336	2
A-177	205	3
A-402	10000	4
A-155	62	5
A-408	1123	6
A-639	750	7

$deposit_2 = \Pi_{account\_number,\ balance,\ tuple\_id}(employee\_info)$

http://www.db-book.com/

Query Processing in Distributed DBMS

- Single, centralized system — primary criterion for cost is just number of disk accesses
- Distributed system
  - Cost of data transmission over network
  - Potential gain in performance from having several sites process parts of the query

Review: Query Processing Single Machine

Given two relations R(A, B) and S(B,C) with no indexes, how do we compute the following?

- Selection: $\sigma_{A=123}(R)$
  Ans: Scan file R, select records with A = 123

- Group by: ${}_{A}\mathcal{F}_{\mathrm{SUM}(B)}(R)$
  Ans: Use either sorting or hashing to aggregate on A then apply sum to each group

- Join: $R * S$
  Ans: Nested block join, create hash index on B for smaller relation and doing hash join, or sort on B and do sort-merge join

Query Processing: Parallel/Distributed DBMS

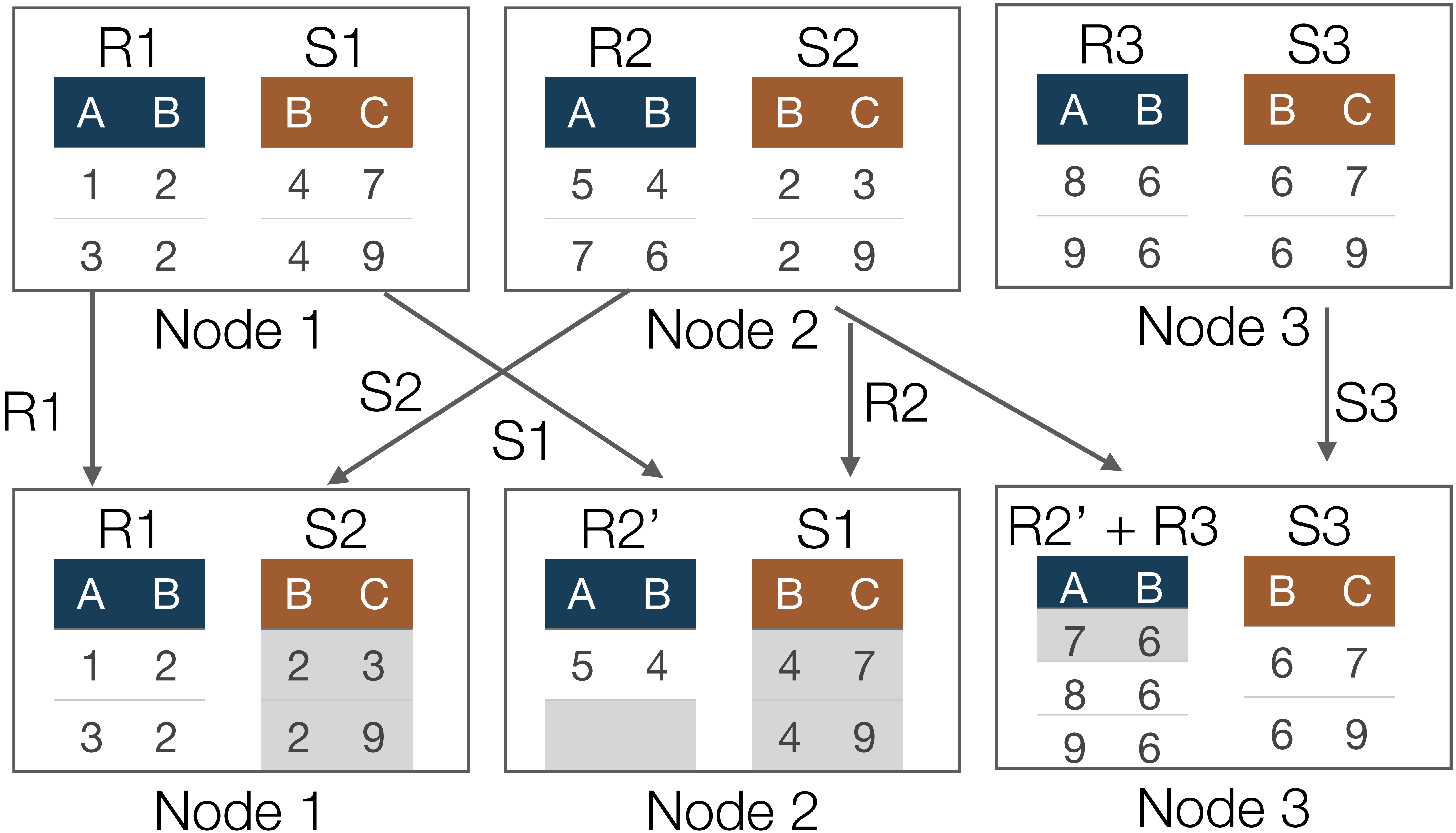
Given two relations R(A, B) and S(B,C) with horizontal partitioning and no indexes, how do we compute the following?

- Selection: $\sigma_{A=123}(R)$
  Ans: Relatively easy, not that different from single
- Group by: ${}_A\mathcal{F}_{\mathrm{SUM}(B)}(R)$
  Ans: If already partitioned based on A to each system (hash or range), relatively easy. Otherwise, if it is round robin, need to pass data to the nodes to aggregate the same values together

Query Processing: Parallel/Distributed DBMS (2)

- Join: $R * S$
  - Strategy 1: Transfer both R and S into one central location and join (very expensive from sending)
  - Strategy 2: Perform local join by just sending the joining column of one relation, S, to where the other one is located, R (minimizes data transmission)

Example: Distributed Join



Example: Distributed Join (2)

A	B	C
1	2	3
1	2	9
3	2	3
3	2	9

Node 1

A	B	C
5	4	7
5	4	9

Node 2

A	B	C
7	6	7
7	6	9
8	6	7
…	…	…

Node 3

combine tuples for final output

Distributed Transactions & Recovery

Problems arising only in distributed/parallel setting

- Dealing with multiple copies of data items — how to maintain consistency amongst the copies
- Failure of individual sites — what to do when one site fails and then rejoins the system later
- Failure of communication issues
- Distributed commit — what to do if some nodes fail during commit process?
- Distributed deadlock

Parallel / Distributed Database Properties

- Advantages
  - Data sharing
  - Reliability and availability
  - Improved query processing speed
- Disadvantages
  - May increase processing overhead
  - Harder to ensure ACID guarantees
  - More database design issues

MapReduce

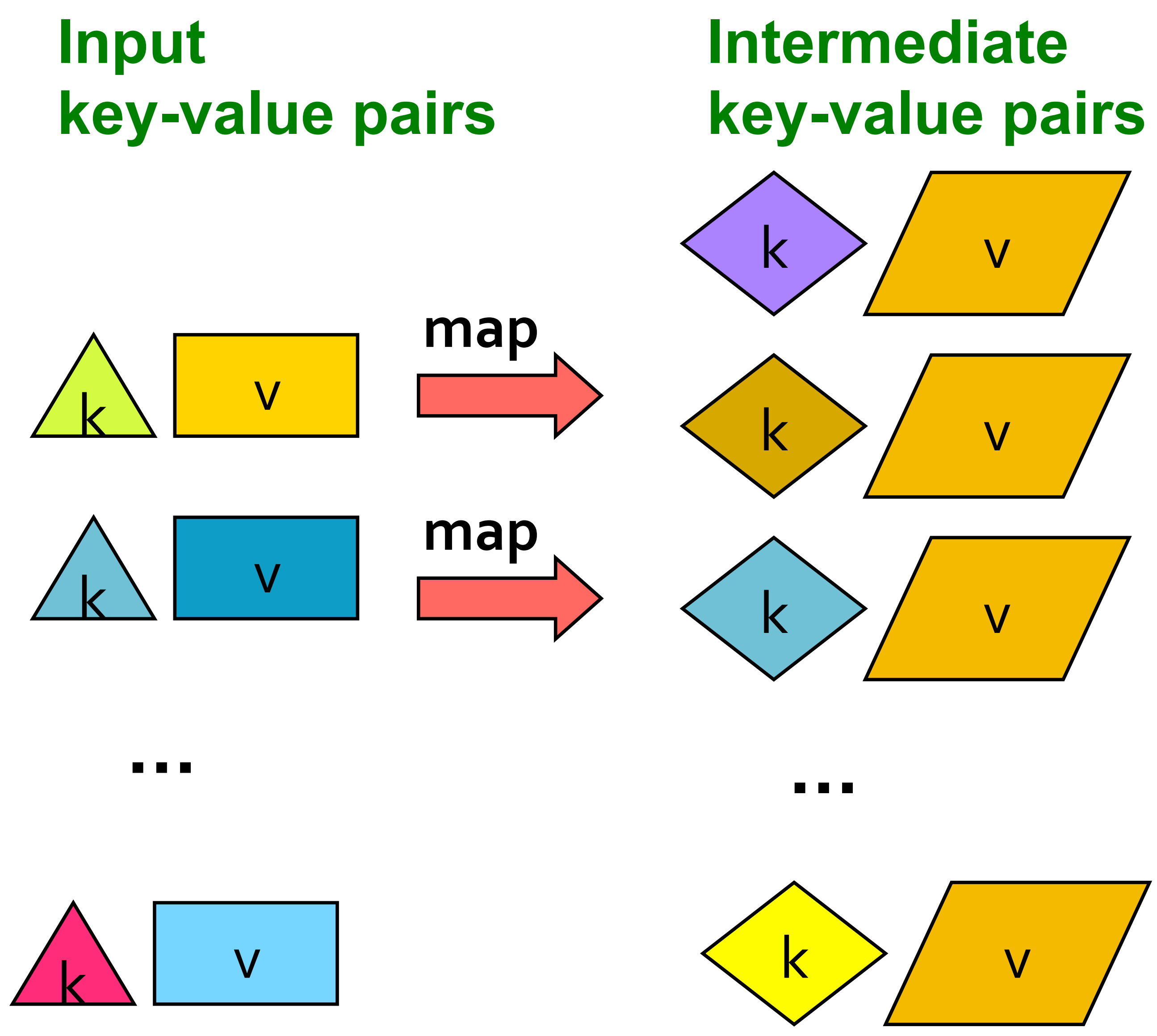
- Initially developed by Jeffrey Dean & Sanjay Ghemawat at Google [2004]
- Open source implementation: Apache Hadoop
- High-level programming model and implementation for large-scale parallel data processing
- Designed to simplify the task of writing parallel programs

MapReduce: Overview

- Read partitioned data
- Map: extract something you care about from each record
- Group by key: sort and shuffle (done by the system)
- Reduce: aggregate, summarize, filter, or transform
- Write the result

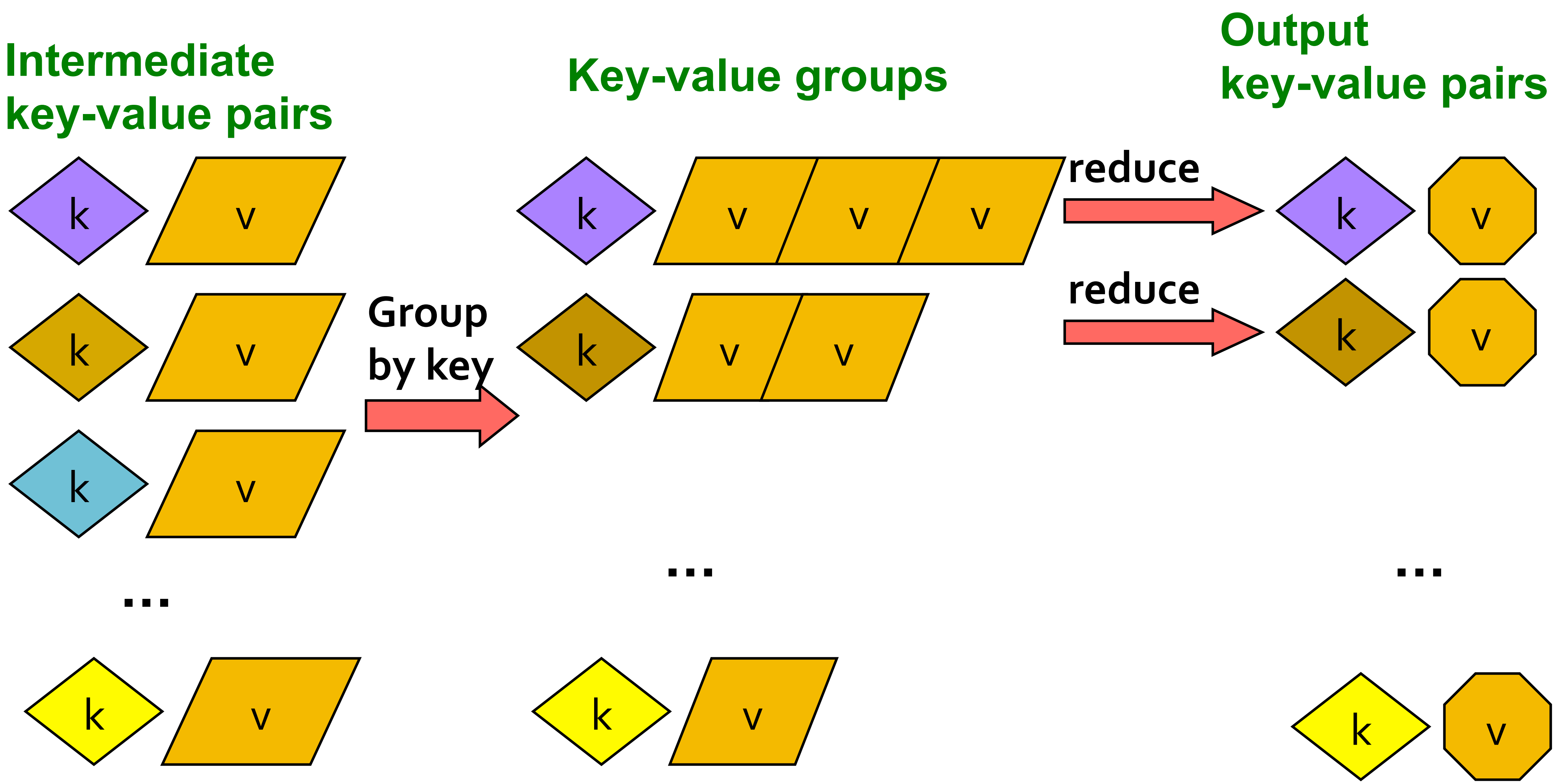
Outline stays the same, map and reduce should be tailored to the problem

MapReduce: Map Step



http://www.mmds.org/#book

MapReduce: Reduce Step

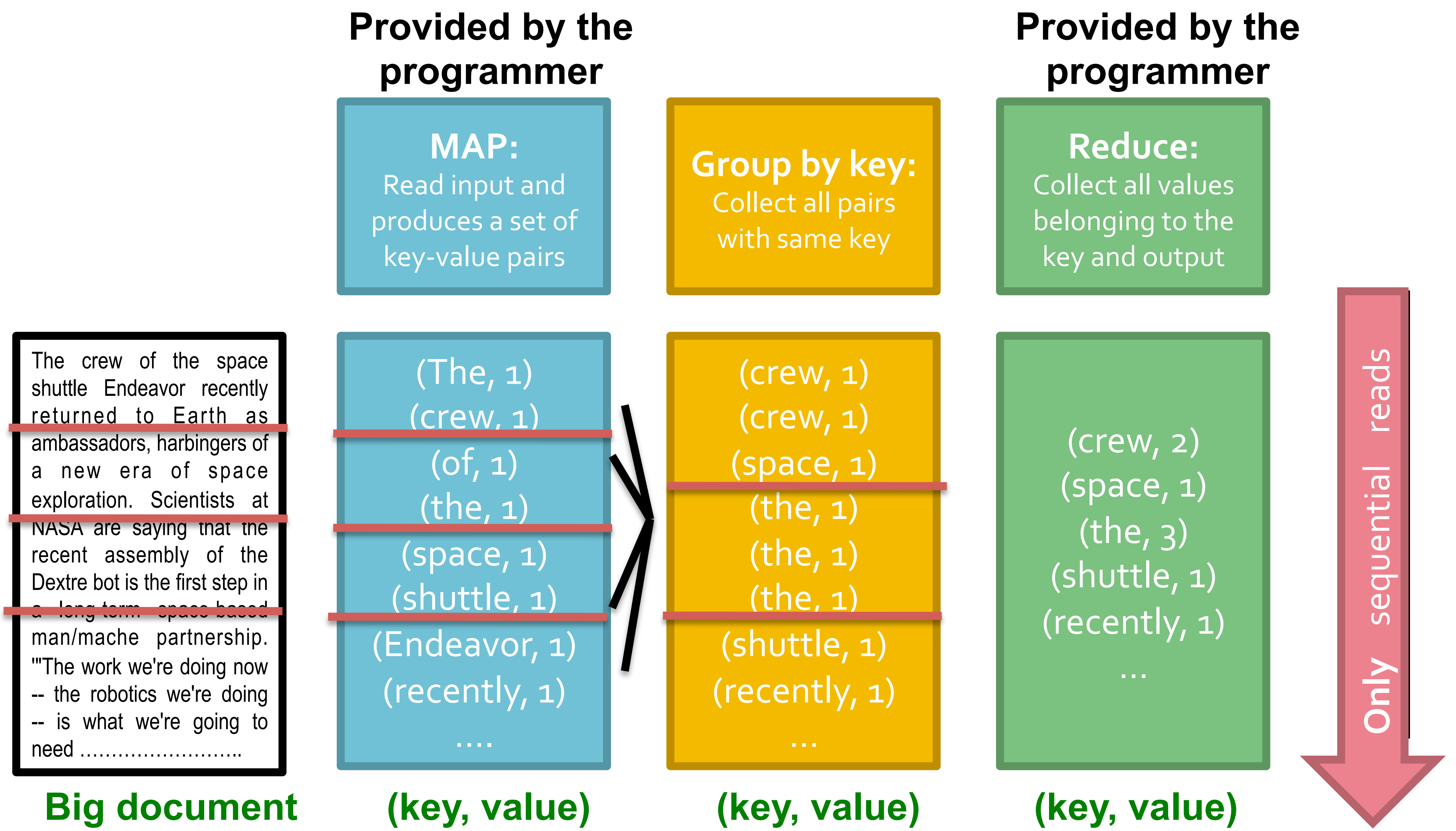


http://www.mmds.org/#book

Example: Word Counting

- We have a huge text document (~ 1 million words)
- Task: Count the number of times each distinct word appears in the file
- Traditional DBMS
  - Load document words into a table
  - SQL query:
    SELECT count(*)
    FROM document
    GROUP BY word

Example: Word Counting (MapReduce)



http://www.mmds.org/#book

MapReduce Ecosystem

Many extensions to address limitations

- Capabilities to write directed acyclic graphs of MapReduce jobs (e.g., PIG by Yahoo!)
- Declarative languages (e.g., Hive by Facebook or SQL/Tenzing by Google)
- Increased integration of DBMS with MapReduce

Parallel DBMS vs MapReduce

Parallel DBMS

- Relational data model and schema
- Declarative query language (SQL)
- Easily combine operators into complex queries
- Query optimization, indexing, and physical tuning
- Streams data from one operator to next without blocking

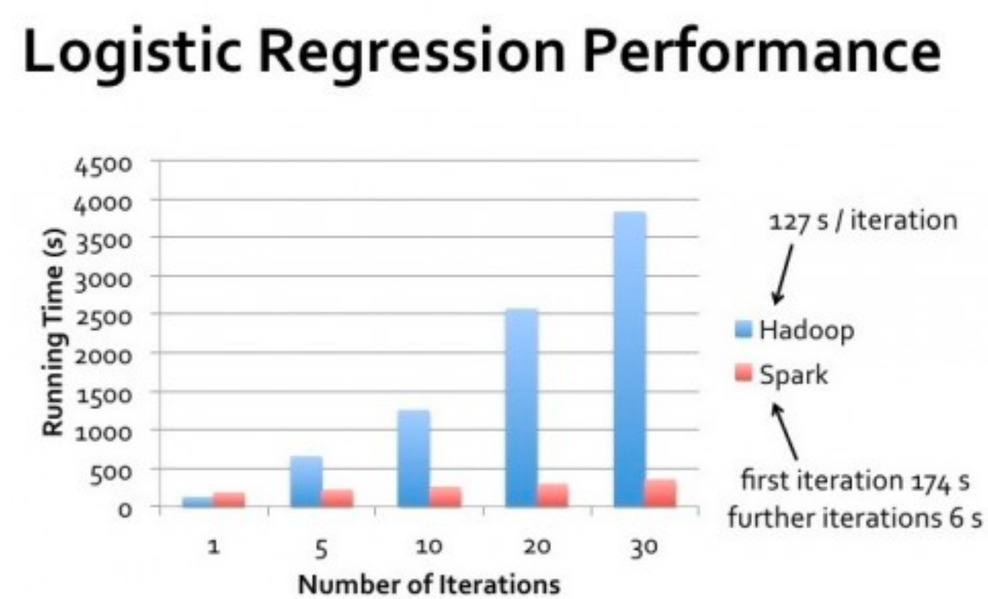
Parallel DBMS vs MapReduce (2)

MapReduce

- Data model is file with key-value pairs
- Pre-loading data is not necessary before processing
- Easy to write user-defined operators
- Easily add nodes to the cluster
- Arguably more scalable, but also needs more nodes

Spark: MapReduce Replacement

- Tagline: Lightning-fast cluster computing
- Run programs up to 100x faster than MapReduce in memory or 10x faster on disk
- Easy to use with support for Java, Scala, Python, and R



http://d287f0h5fel5hu.cloudfront.net/blog/wp-content/uploads/2015/12/4-481x300.jpg

Big Data Systems: Recap

- Big Data (4 V's)
- Parallel/Distributed DBMS
  - Different architectures
  - Data distribution
  - Query processing
- MapReduce