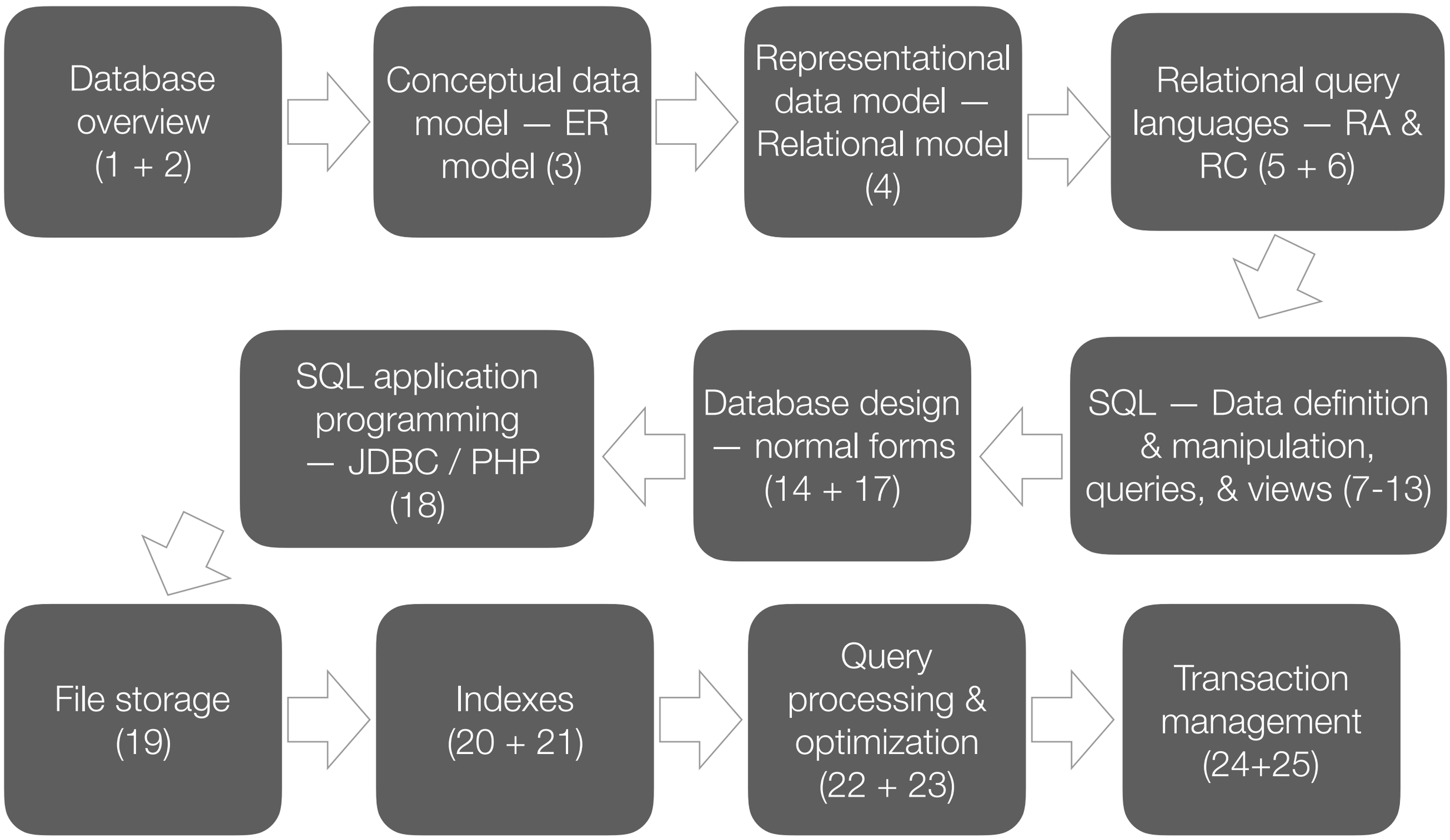
Big Data Systems

CS 377: Database Systems

Review: Course Material to Date



Review: What Has Been Covered

What I hope you've learned…

- Design a database
  (Requirements -> ER diagram -> Relational model -> Database normalization)
- Querying a database
  (Relational algebra, calculus, SQL queries)
- Writing applications to use databases
  (JDBC & SQL)

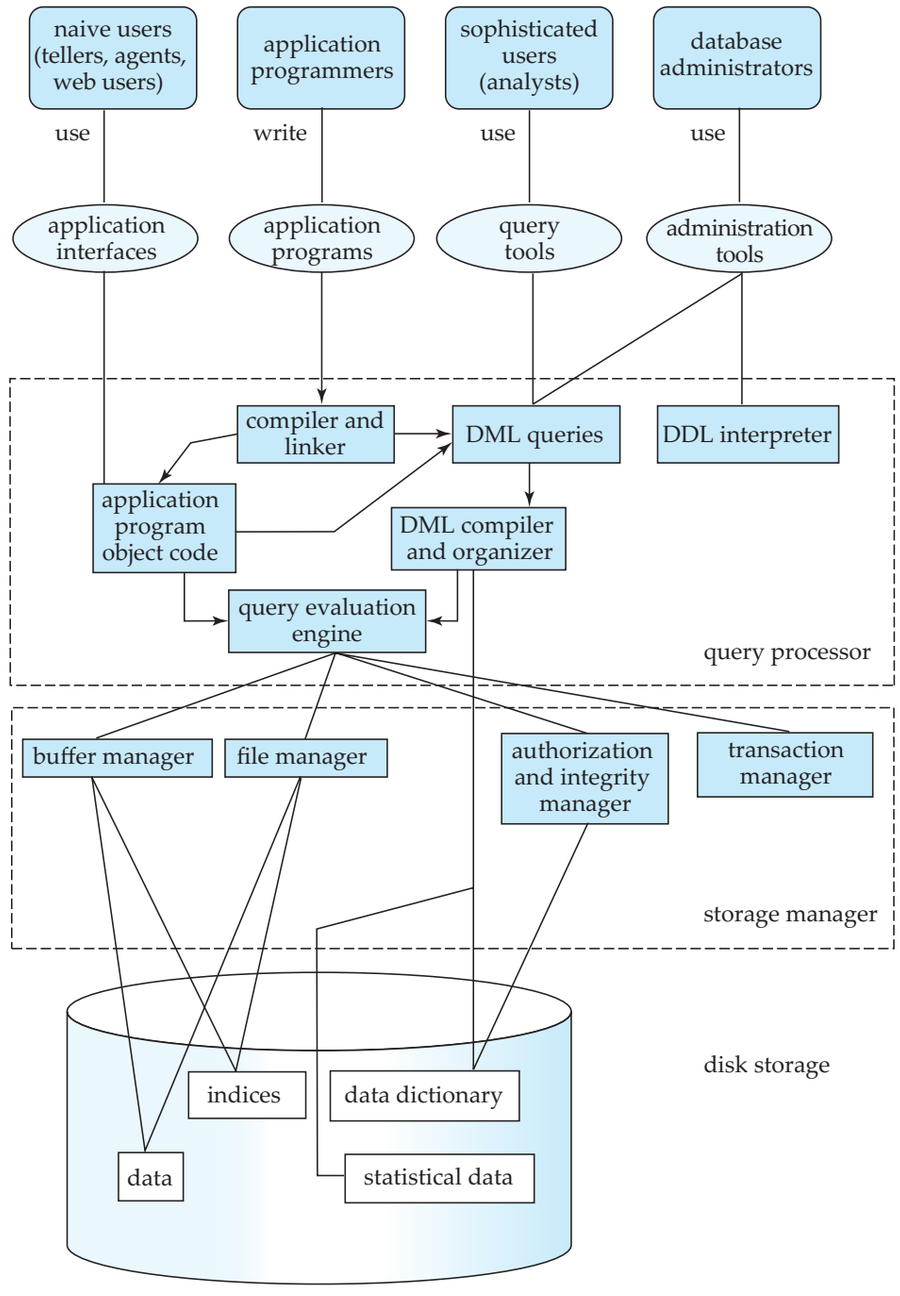
Review: What Has Been Covered

What I hope you've learned (at a high level)…

- Why some queries run faster on some systems compared to others
- How to think about optimizing your performance (Indexes, SQL Processing & optimization)
- How to achieve ACID (Logs & Locks)

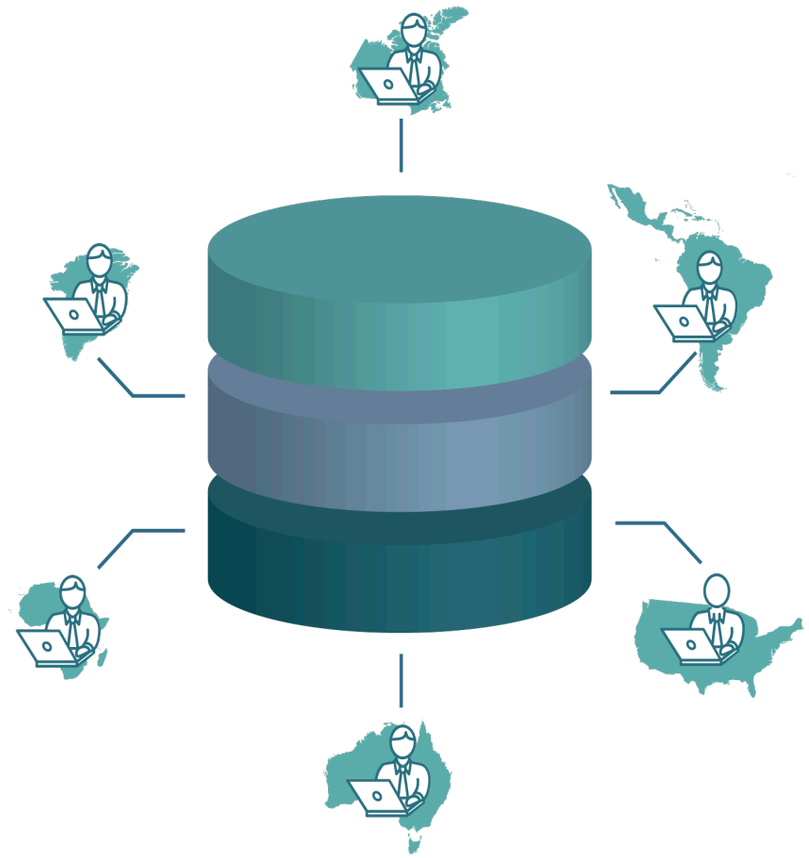
Review: “Peeking” Under the Hood

- Most aspects of traditional RDBMS is understood
- Learned enough to be “dangerous”
- Additional details can be picked up in courses or on your own



Review: Centralized Database

- Data is located in one place (one server)
- All functions performed by the server
  - Query processing
  - Transaction management, concurrency control
  - …



What if I have 100 TB of data?

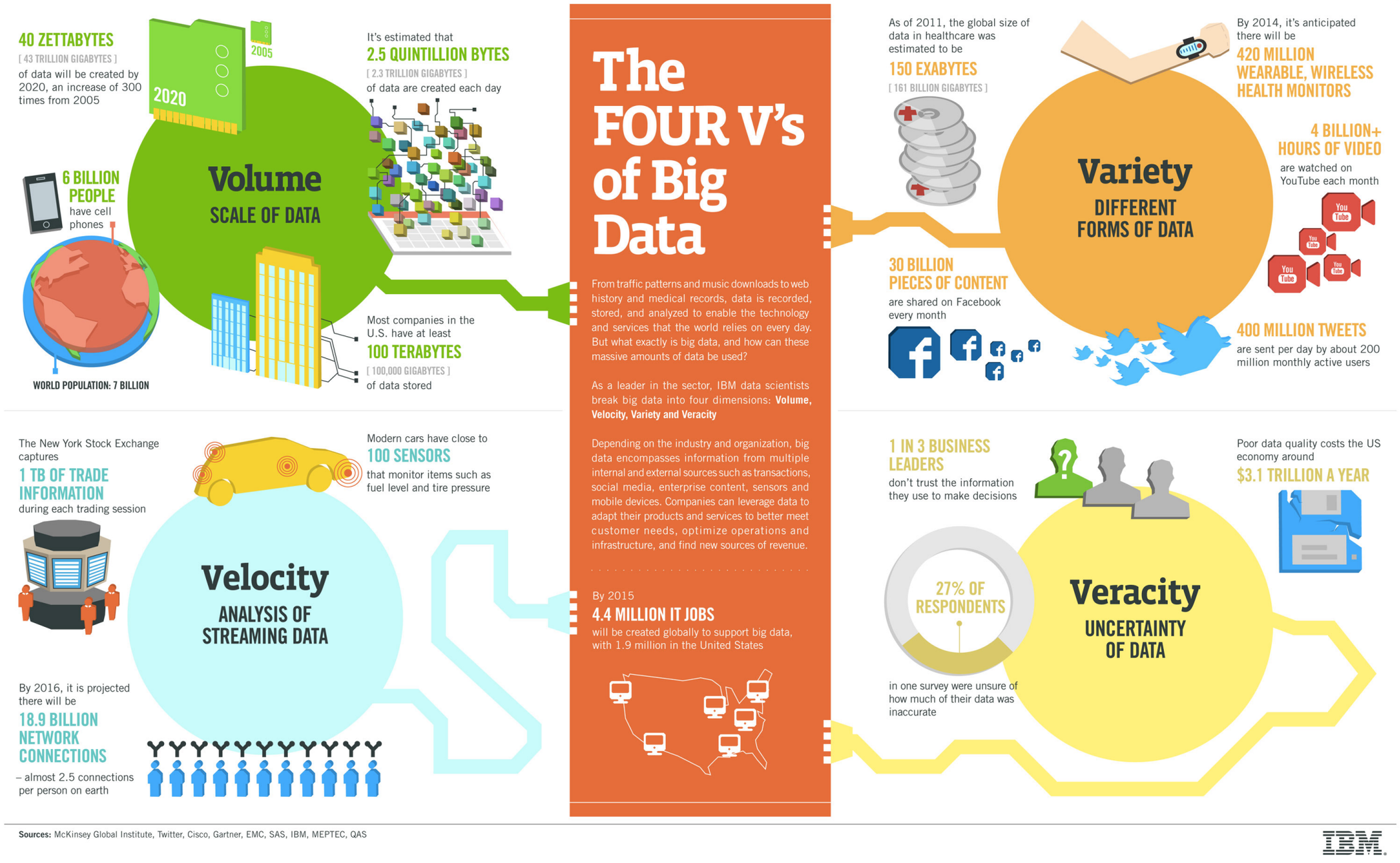
Data Never Sleeps

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

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

4 V's of Big Data



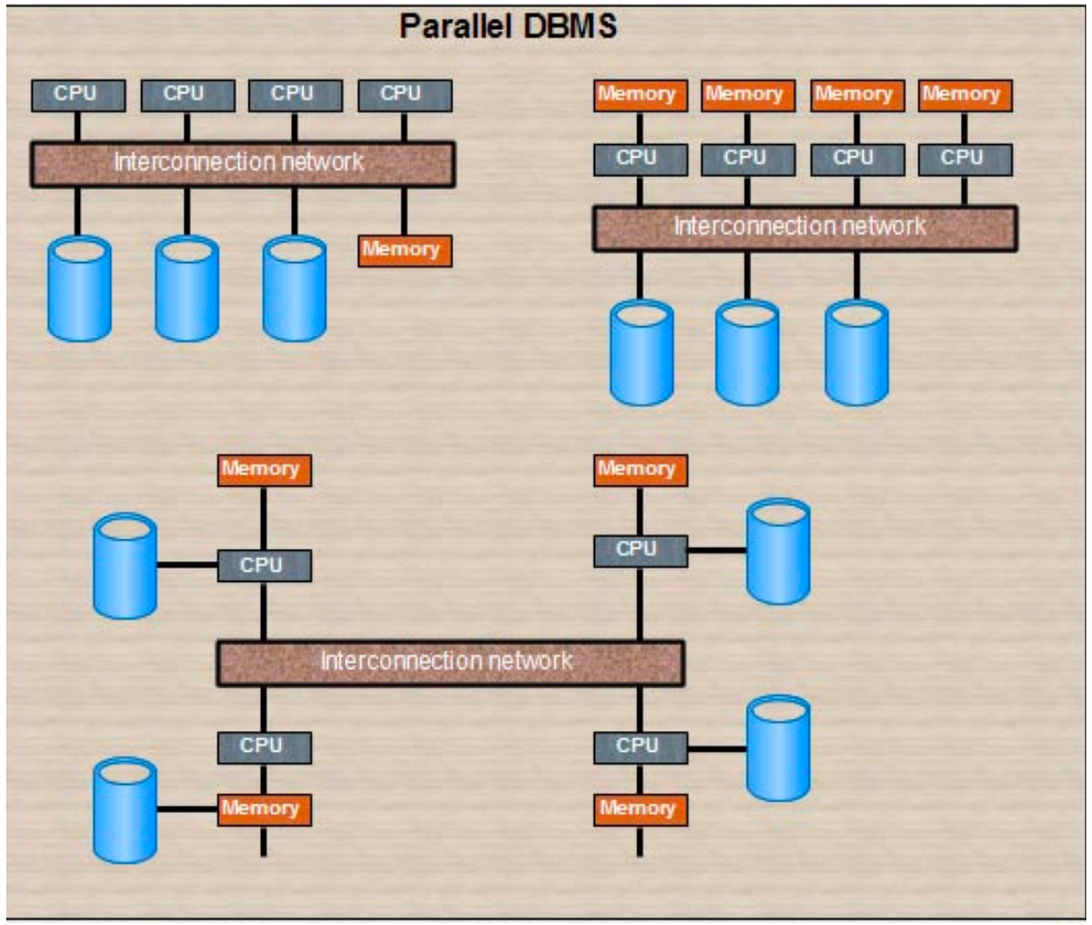
http://www.ibmbigdatahub.com/infographic/four-vs-big-data

Parallel & Distributed DBs: Motivation

- Single, monolithic DBMS is impractical and expensive
- Improve performance
- Increased availability & reliability
- Potentially lower cost of ownership
- Easier, more economical system expansion

Parallel & Distributed DBs: Overview

- Data partitioned across multiple disks
- Allows parallel I/O for better speed-up
- Queries can be run in parallel with each other

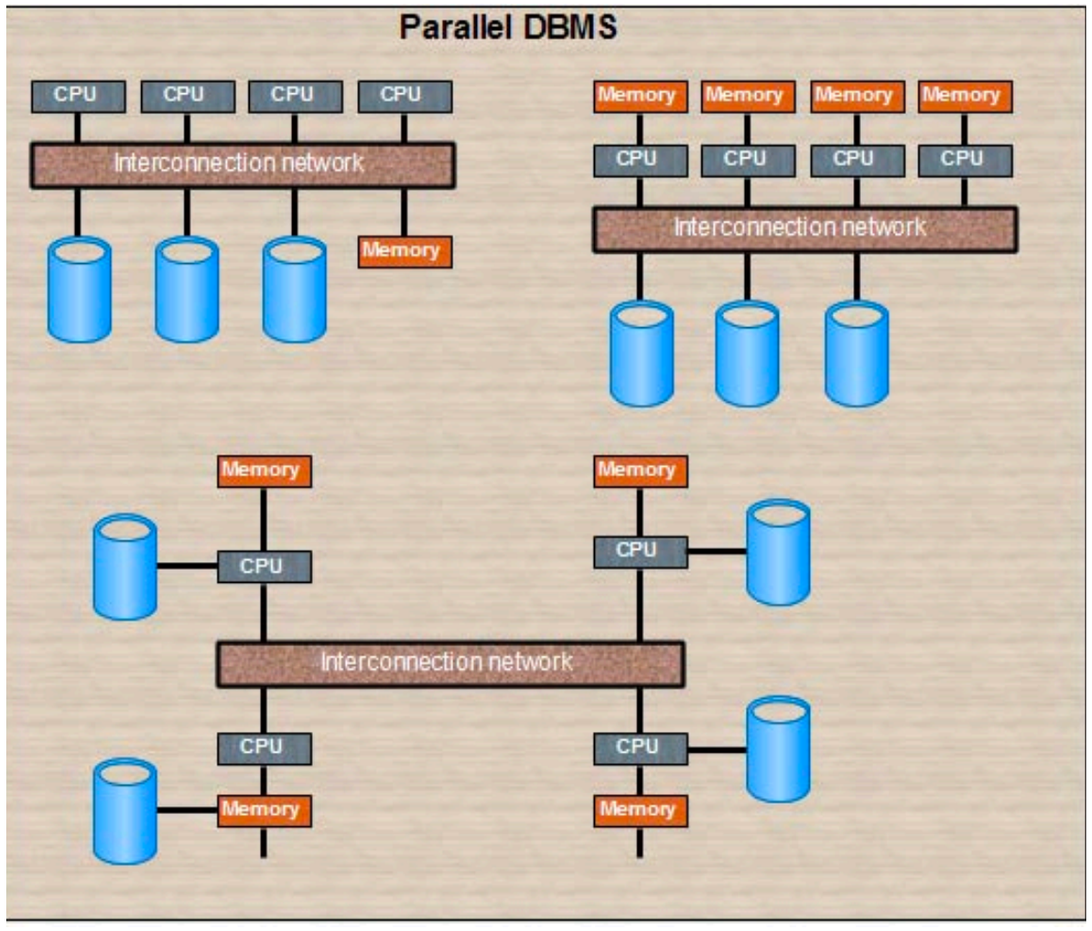


Parallel & Distributed DBs: Overview

- Each processor can work independently on its own partition

- Individual relational operations (e.g., sort, join, aggregation) can be executed in parallel

- Concurrency control takes care of conflicts

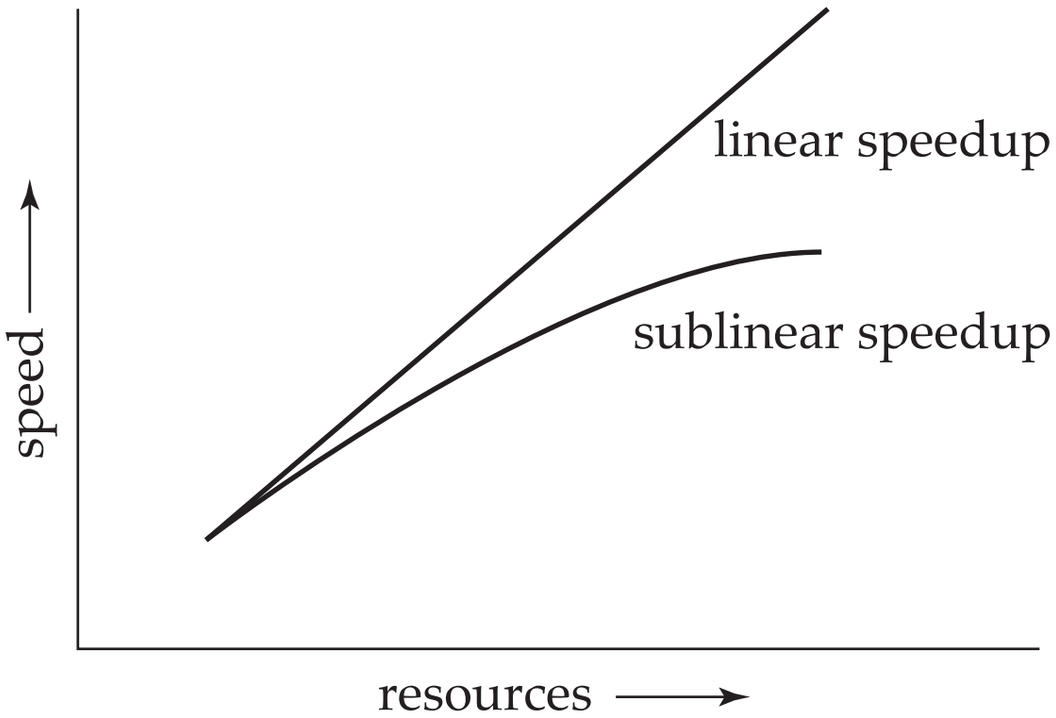


Scale-Up vs Scale-Out

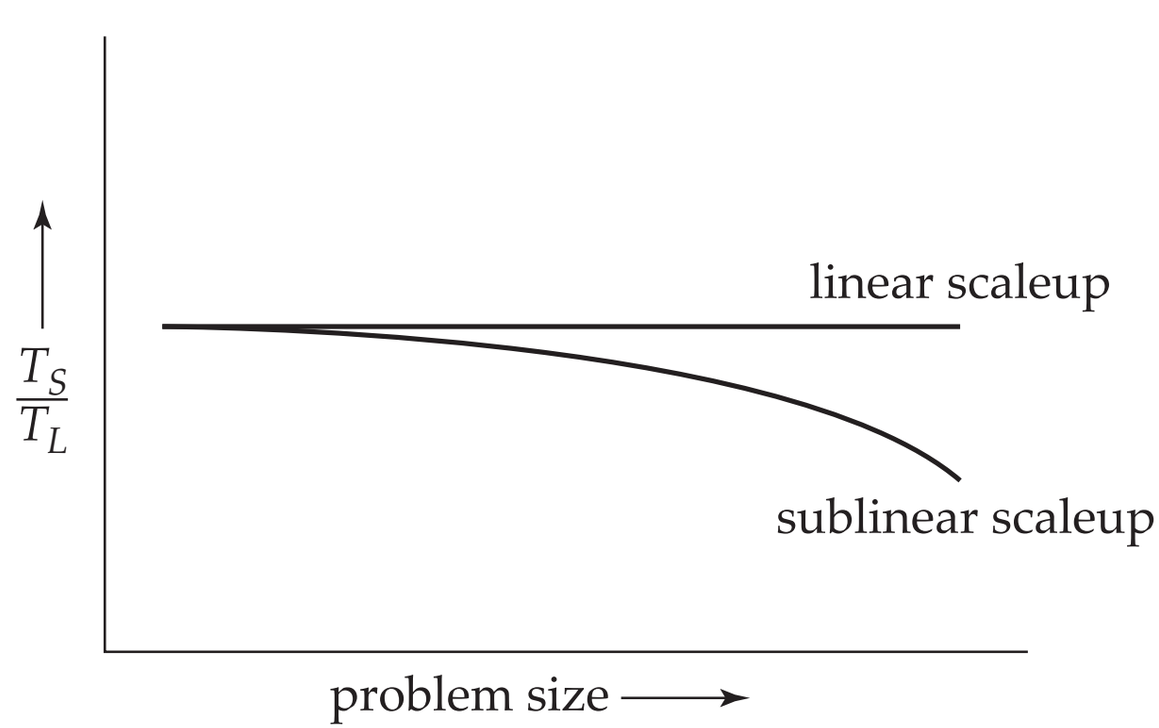
- Terminology to measure performance
- Speed-up: using more processors, how much faster will the task run (assuming same problem size)?
- Scale-up: using more processors, does performance remain the same as we increase problem size?
- Scale-out: using a larger number of servers, does performance improve?

Ideal Scenarios

Speed-up



Scale-up



Parallel & Distributed DBs: 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

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 Architectures

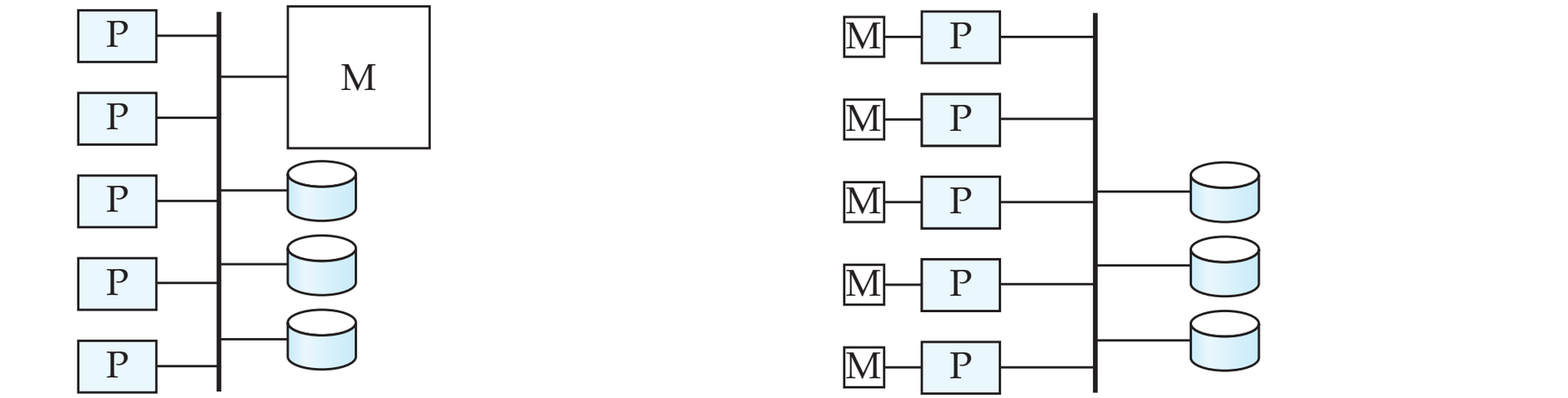
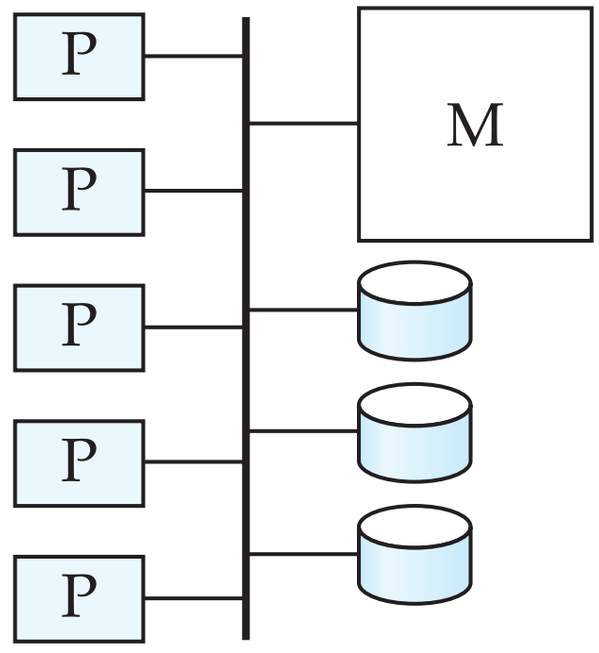


Figure 17.8 (Database System Concepts)

Shared Memory

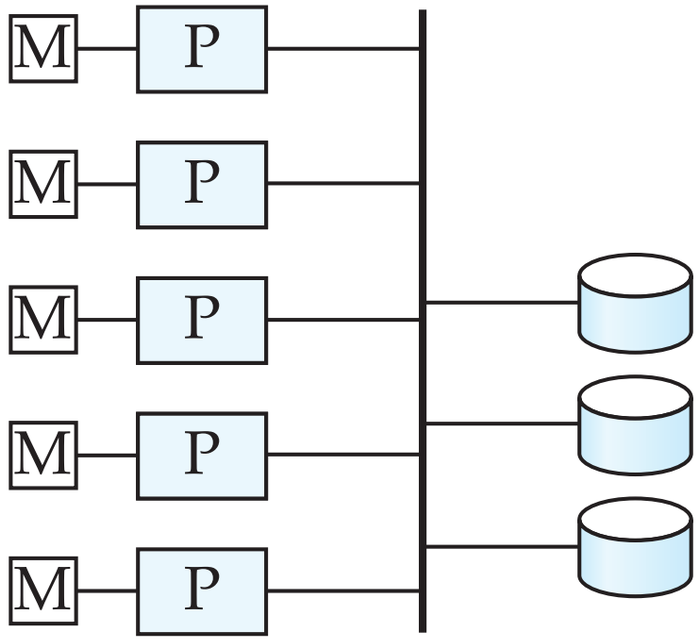
- Nodes share RAM + disk
- 10-100+ processors
- Easy to use and program
- Expensive to scale — last remaining cash cow in the hardware industry



(a) shared memory

Shared Disk

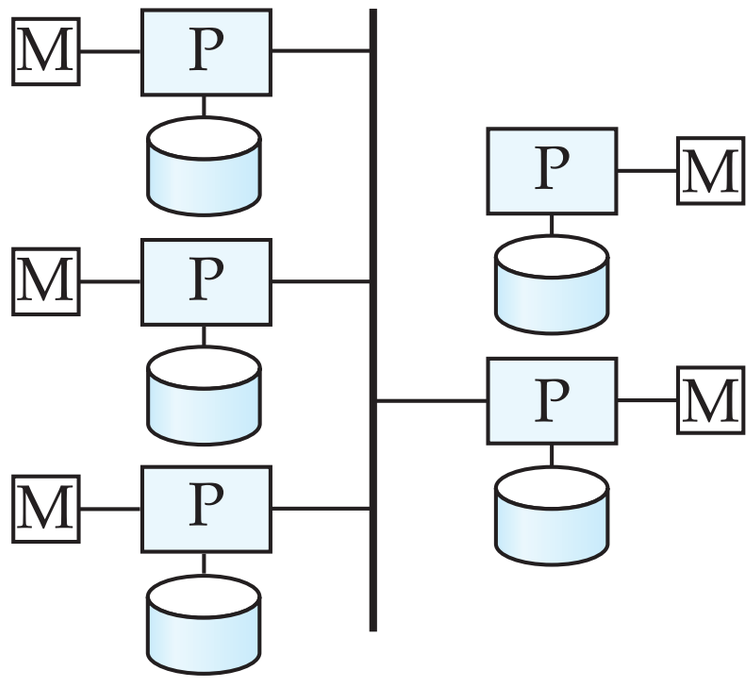
- Nodes share same disk
- Easy fault tolerance & consistency
- Hard to scale past a certain point — existing deployments typically have fewer than 10 machines
- Example: Oracle servers use this paradigm quite a bit



(b) shared disk

Shared Nothing

- Each instance has its own CPU, memory, and disk
- Easy to increase capacity
- Hard to ensure consistency
- Most scalable architecture but difficult to administer & tune



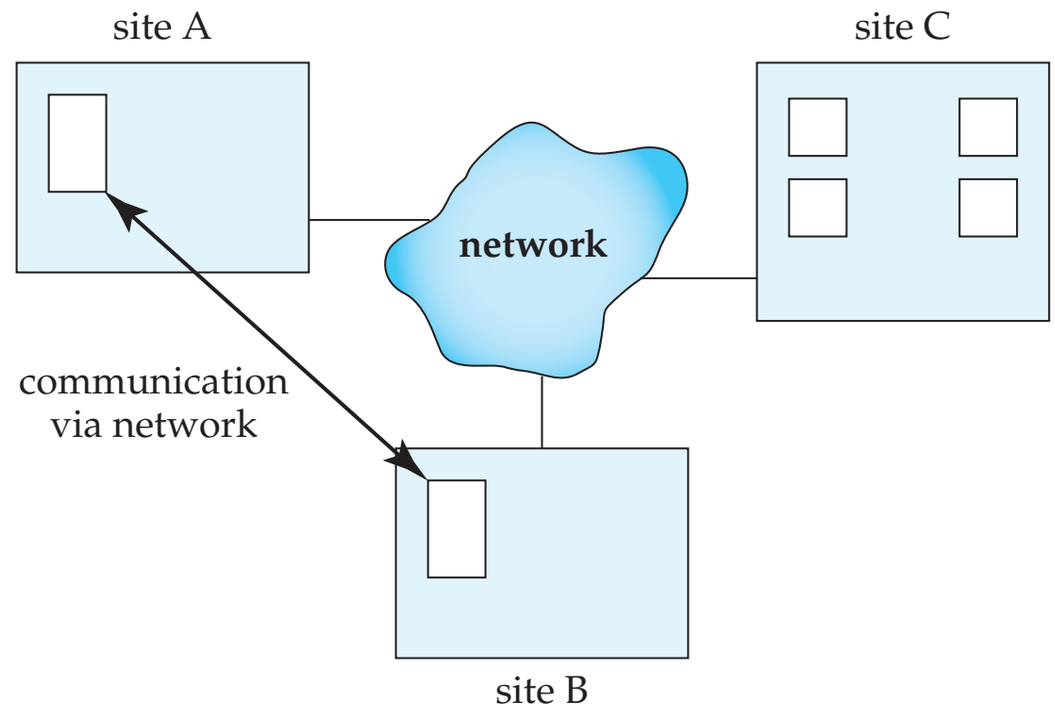
(c) shared nothing

Distributed Databases

- Data spread over multiple machine

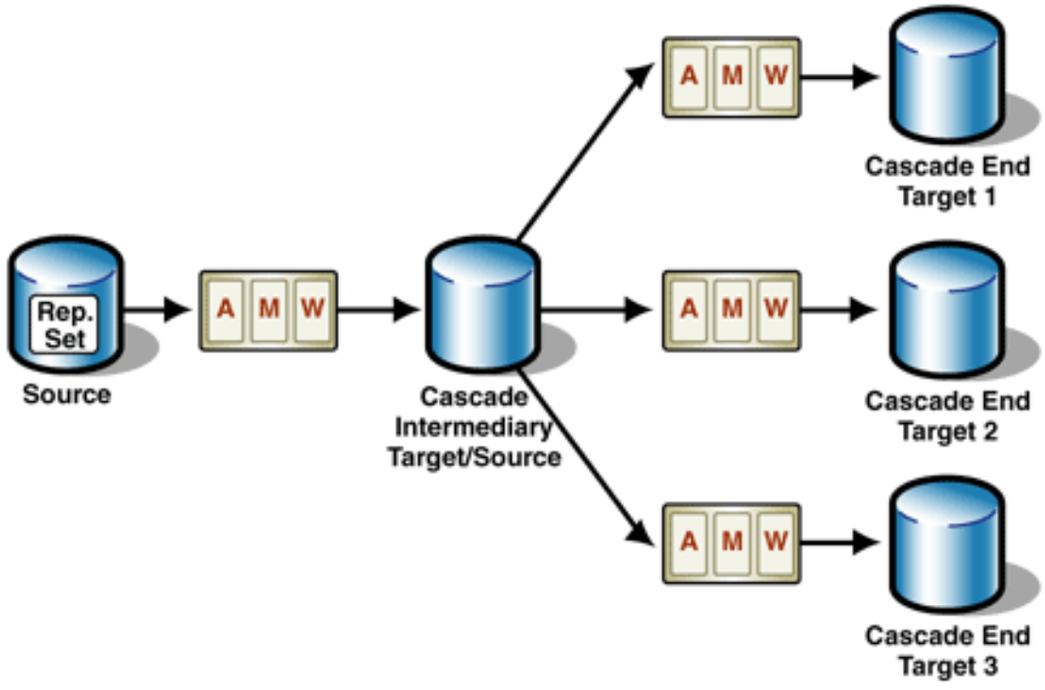
- Network interconnects the machines

- Similar to shared nothing architecture but larger communication cost



How to Distribute the Data?

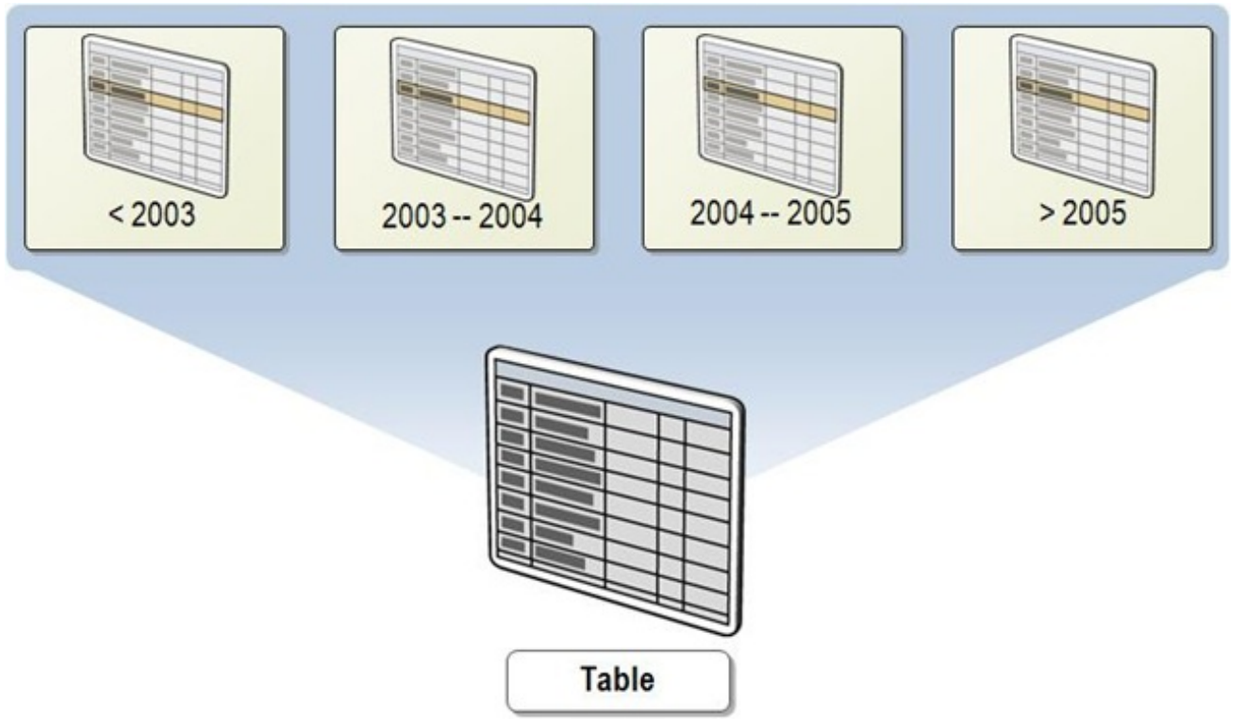
- Replication: system maintains multiple copies of data
  - (PRO) Improves availability, parallelism, and reduced data transfer
  - (CON) Increased cost of updates, complexity of concurrency control



How to Distribute the Data?

- Fragmentation: relation is partitioned into several fragments stored at distinct sites

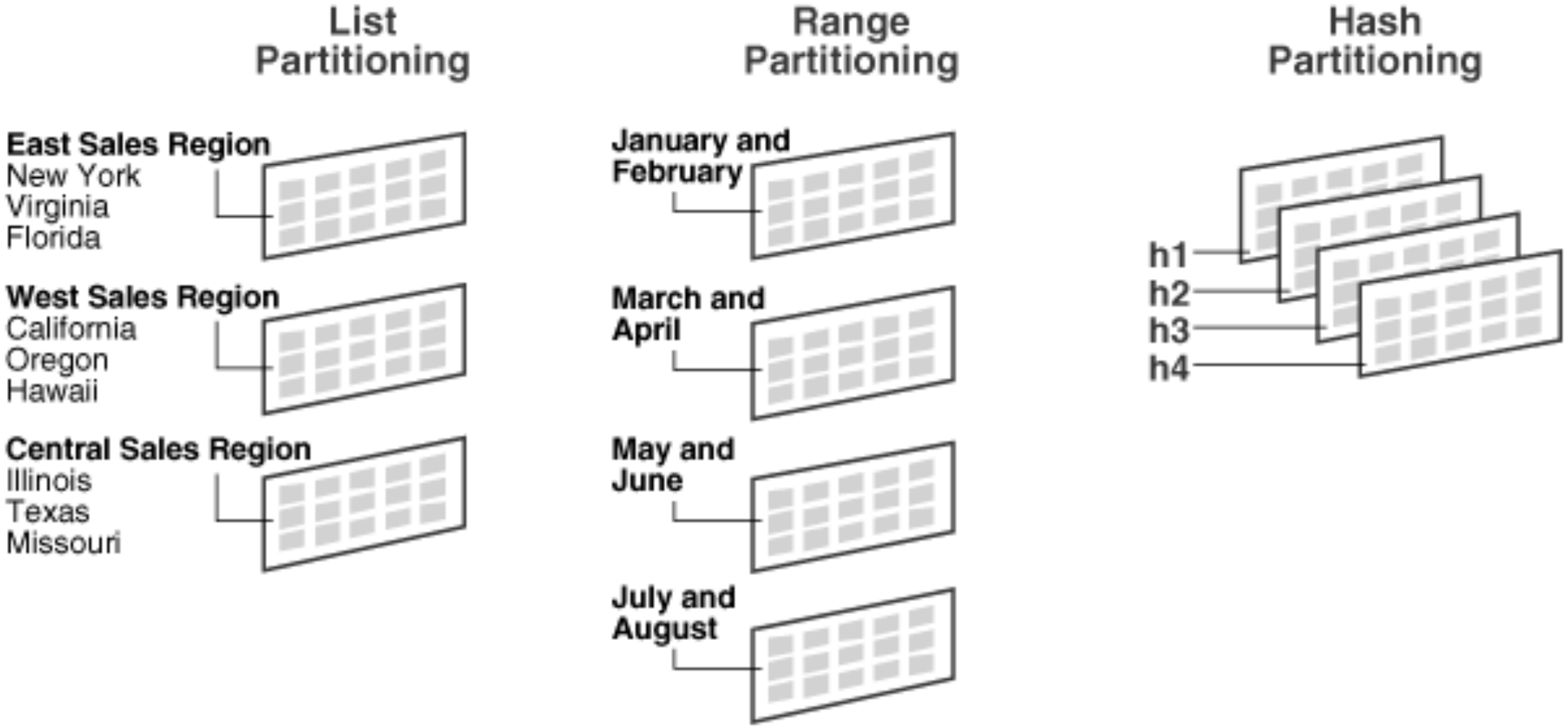
- Combination of both replication & fragmentation



Fragmentation Strategies

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

Horizontal Partition



https://docs.oracle.com/cd/B28359_01/server.111/b32024/partition.htm

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/

Example: Replication & Fragmentation

Figure 25.1
Data distribution and replication among distributed databases.

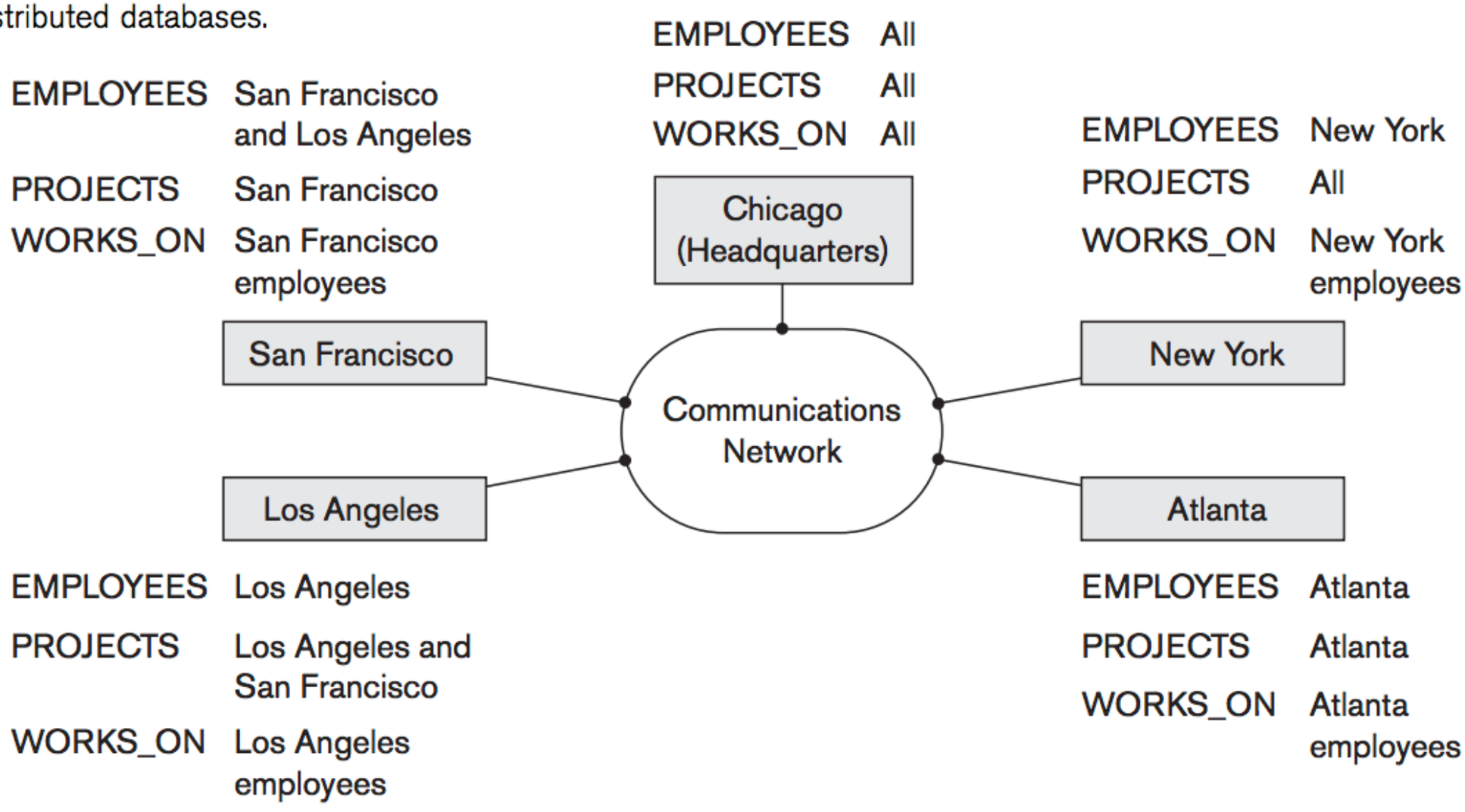


Figure 25.1 from FoDS book

Query Processing

- 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: Centralized DB Query

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

- Selection: $\sigma_{A=123}(R)$

Linear search: scan file R and search for records A=123

Sort/hash for aggregation and apply sum

Review: Centralized DB Query

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

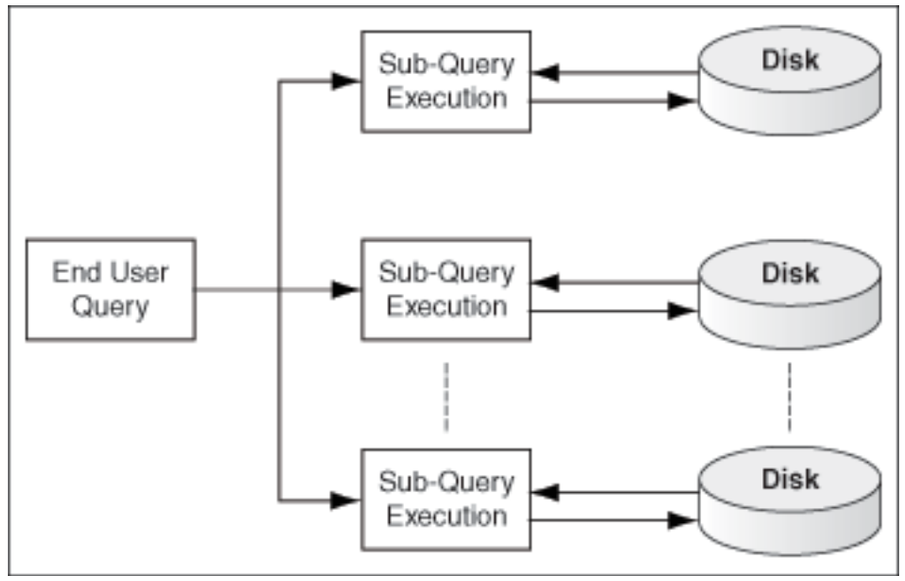
- Join: $R * S$
  - Nested block join
  - Hash join by creating hash index on B for smaller relation
  - Sort-merge join: sort on B for both relations

Parallel & Distributed DBs: Query

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

- Selection: $\sigma_{A=123}(R)$

Relatively straightforward — each machine scans its own partition and applies the condition



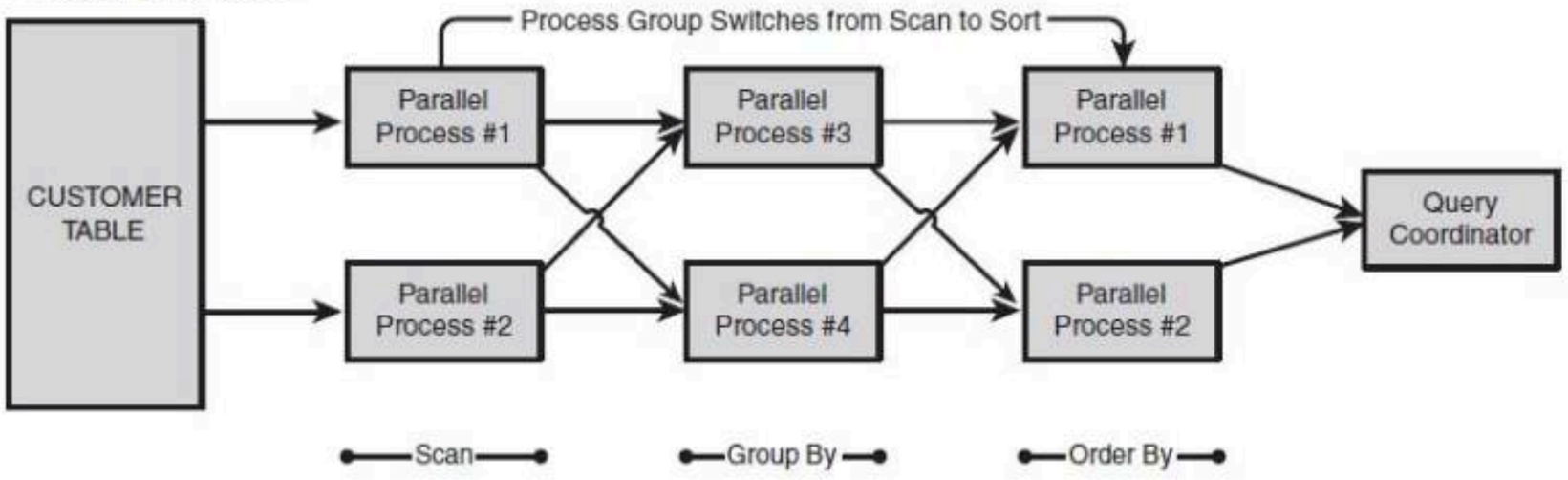
Parallel & Distributed DBs: Query

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

- Selection: $_A\mathcal{F}_{\mathrm{SUM}(B)}(R)$

Fo
com

```
SELECT /*+ parallel(c,2) */ cust_last_name,count(*)
  FROM customers c
 GROUP BY cust_last_name
 ORDER BY 2 desc
```



)

Parallel & Distributed DBs: Query

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

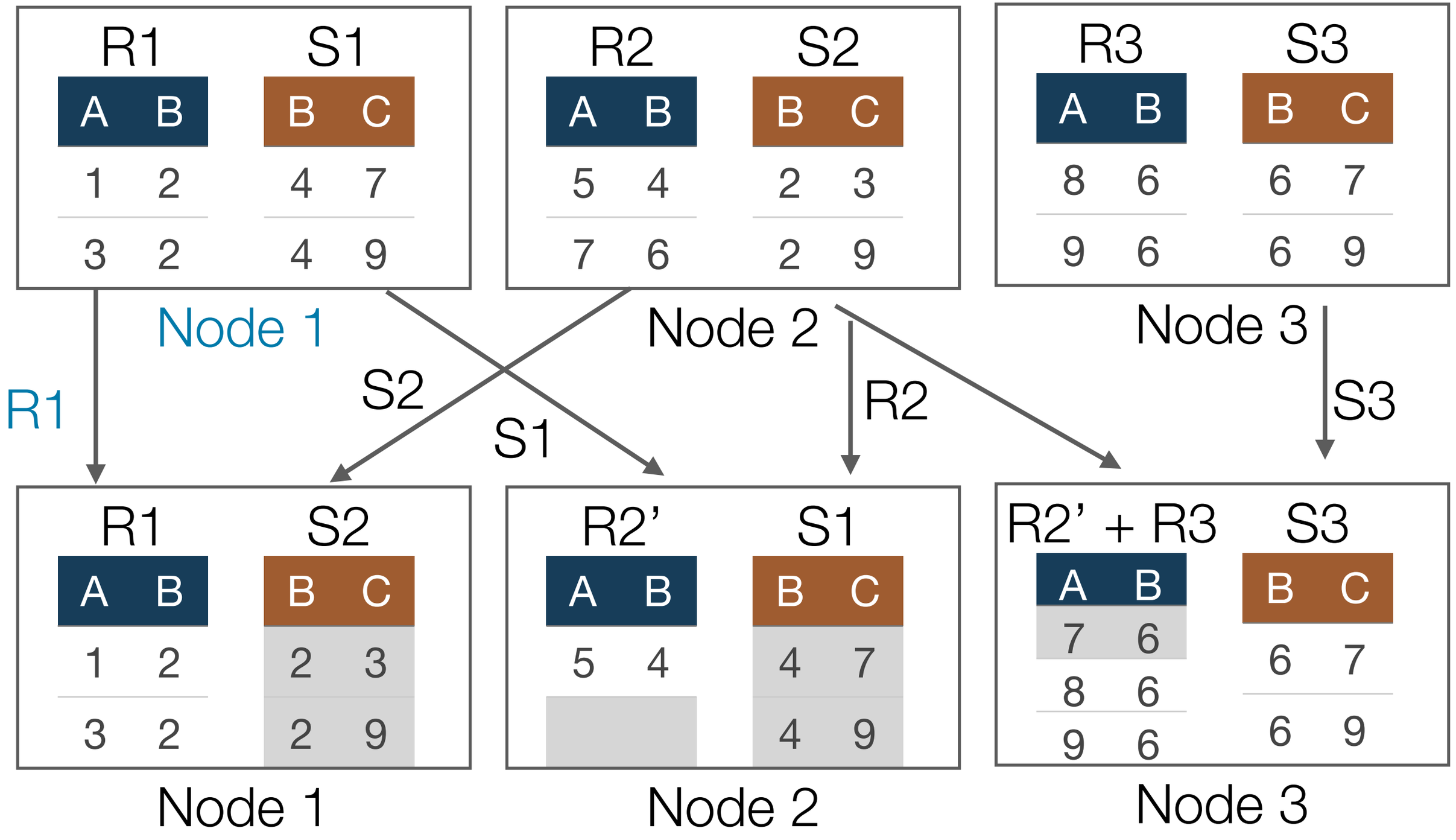
- Selection: $_A\mathcal{F}_{\mathrm{SUM}(B)}(R)$

For hash and range partitions, relatively easy — complication occurs for round robin which needs to aggregate same values together

Parallel & Distributed DBs: Query

- 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

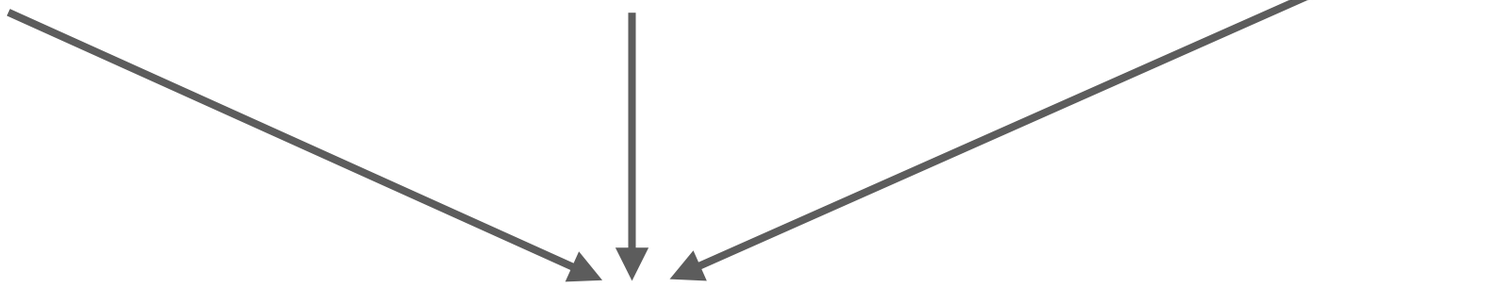
Node 1

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

- 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 DBs: 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

What if I'm looking to manipulate diverse data? For example, what if I want to extract links from webpages and aggregate them by target document? Should I do this all in SQL?

MapReduce

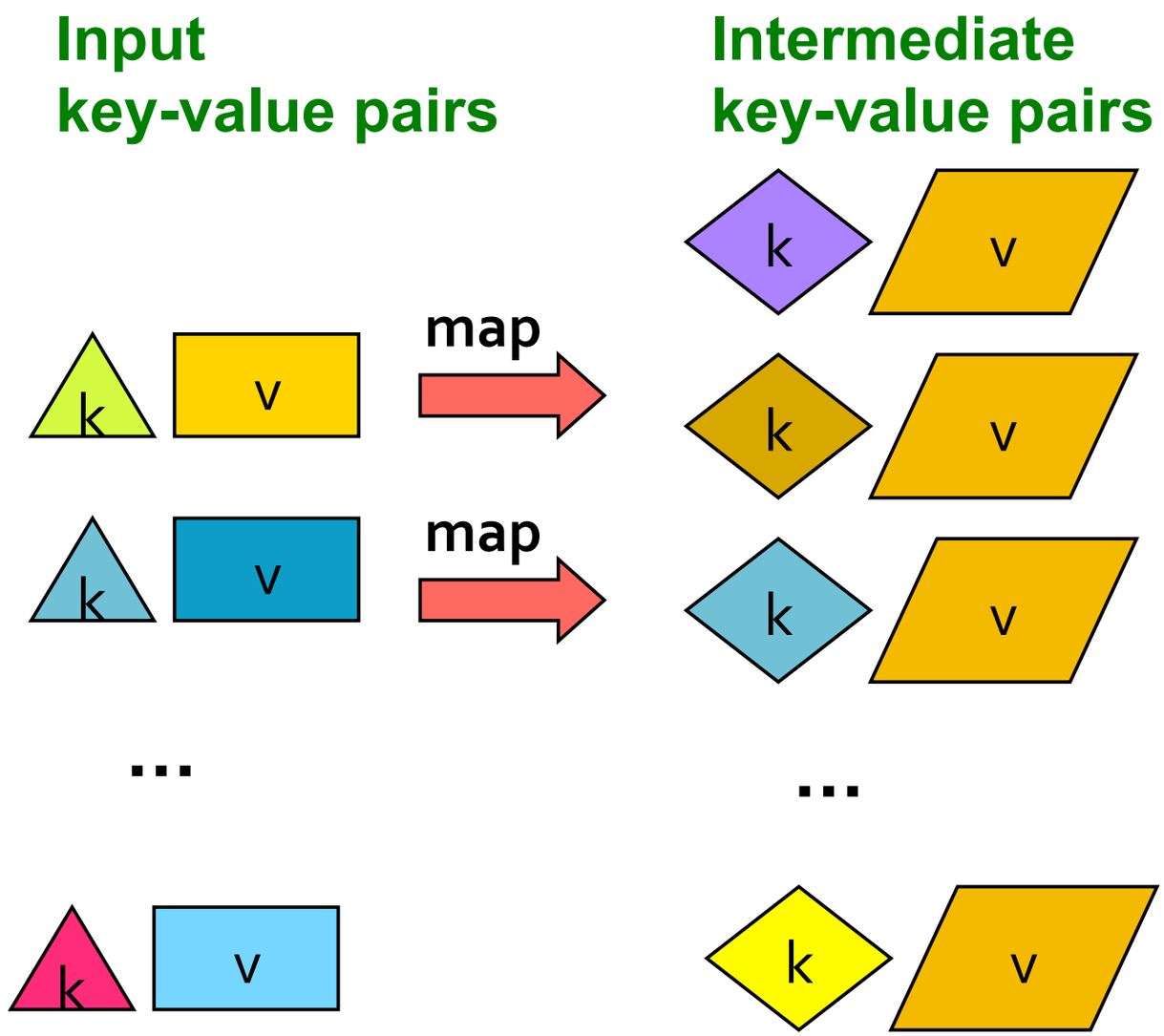
- Initially developed by Jeffrey Dean & Sanjay Ghemawat at Google [2004]
- Open source implementation: Apache Hadoop
- High-level programming model and implementation for large-scale parallel data processing
- Designed to simplify the task of writing parallel programs

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

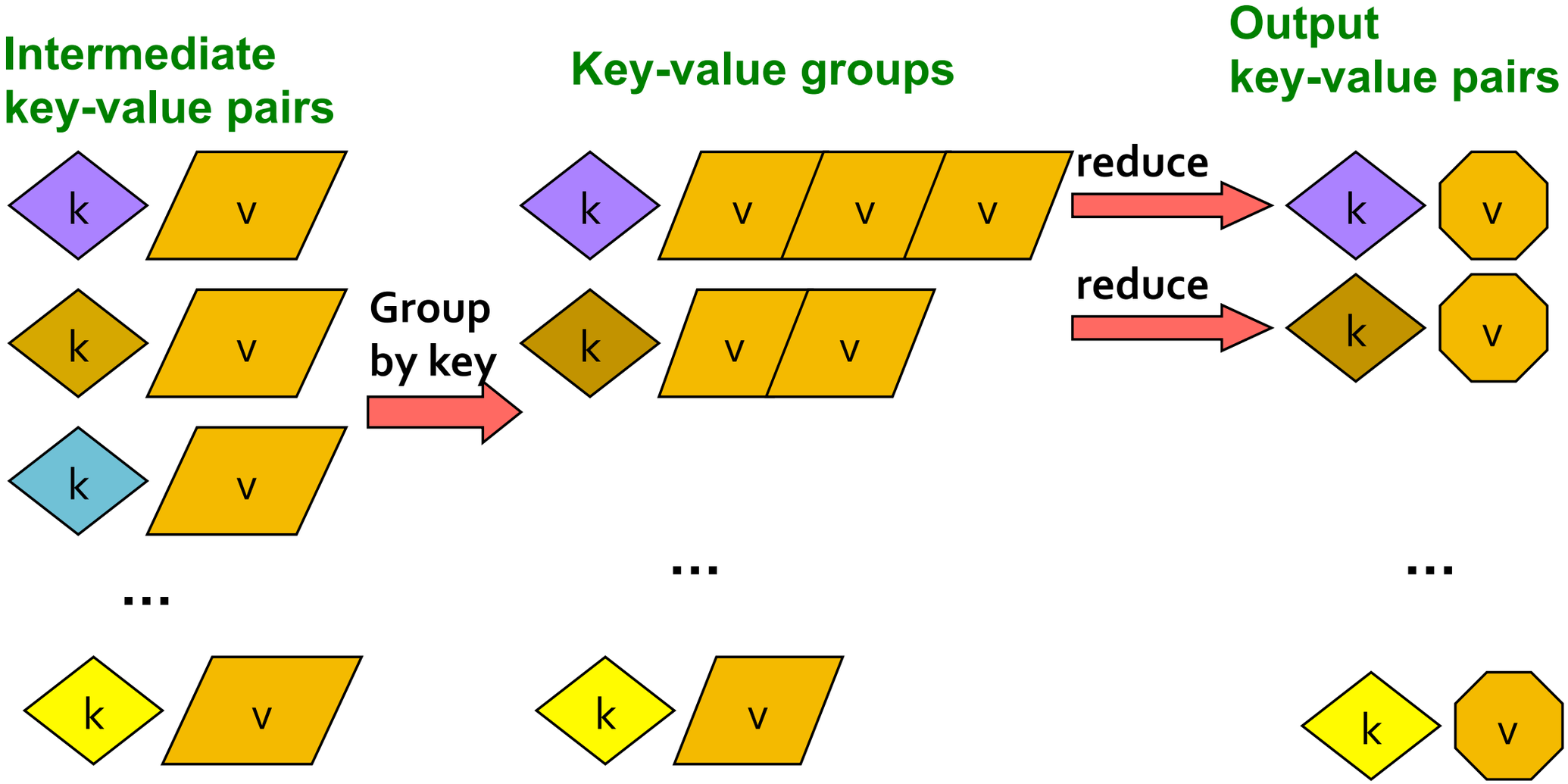
Outline stays the same, map and reduce should be tailored to the problem

MapReduce: Map Step



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

MapReduce: Reduce Step



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

Example: Word Counting

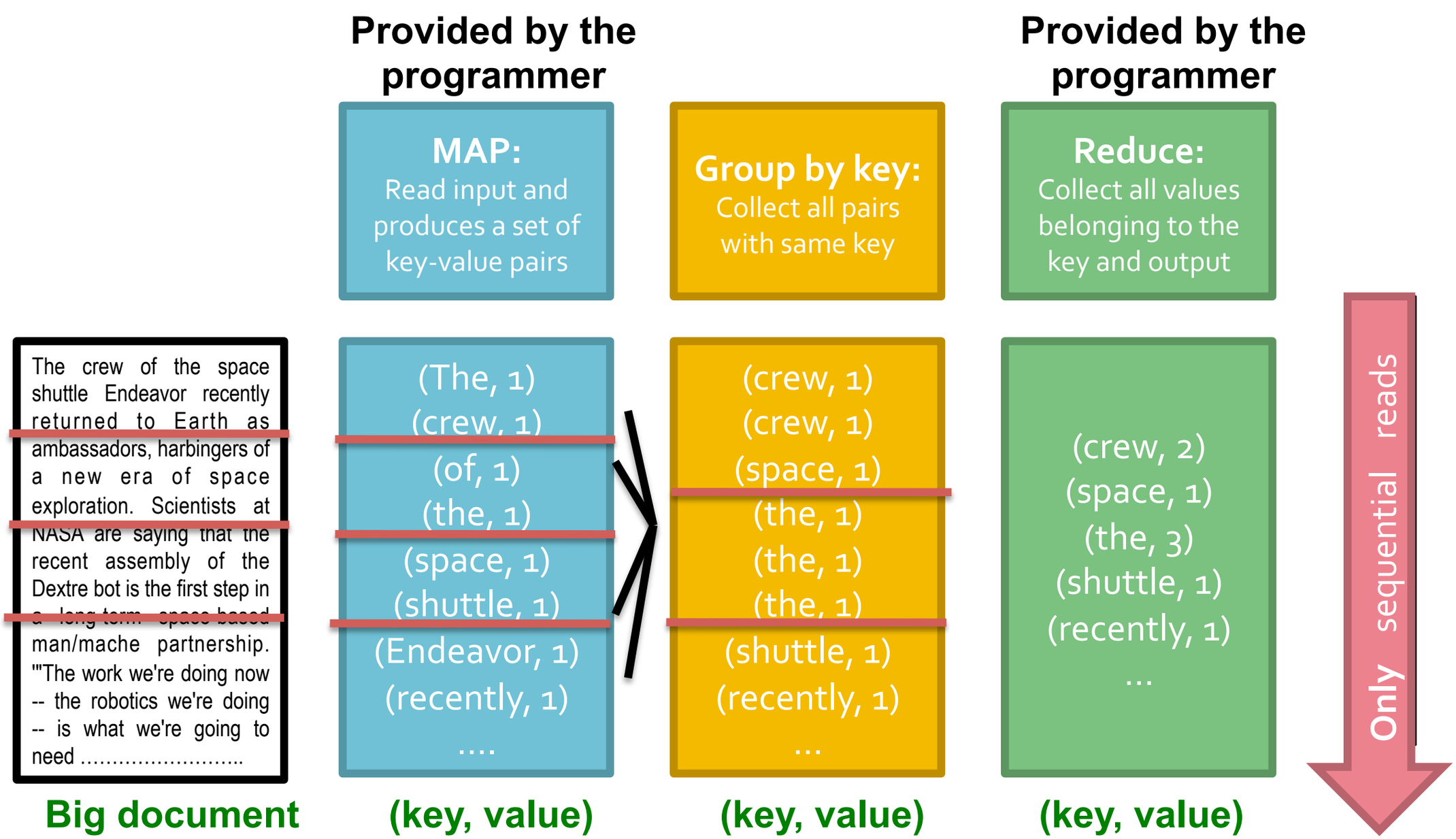
- We have a huge text document (~ 1 million words)
- Task: Count the number of times each distinct word appears in the file

Call me Ishmael. Some years ago--never mind how long precisely--having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen and regulating the circulation. Whenever I find myself growing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenever I find myself involuntarily pausing before coffin warehouses, and bringing up the rear of every funeral I meet; and especially whenever my hypos get such an upper hand of me, that it requires a strong moral principle to prevent me from deliberately stepping into the street, and methodically knocking people's hats off--then, I account it high time to get to sea as soon as I can. This is my substitute for pistol and ball. With a philosophical flourish Cato throws himself upon his sword; I quietly take to the ship. There is nothing surprising in this. If they but knew it, almost all men in their degree, some time or other, cherish very nearly the same feelings towards the ocean with me.

Example: Word Counting

- 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: DB Standpoint

MapReduce: A major step backwards

By David DeWitt on January 17, 2008 4:20 PM | Permalink | Comments (44) | TrackBacks (1)
[Note: Although the system attributes this post to a single author, it was written by David J. DeWitt and Michael Stonebraker]

On January 8, a Database Column reader asked for our views on new distributed database research efforts, and we'll begin here with our views on MapReduce. This is a good time to discuss it, since the recent trade press has been filled with news of the revolution of so-called "cloud computing." This paradigm entails harnessing large numbers of (low-end) processors working in parallel to solve a computing problem. In effect, this suggests constructing a data center by lining up a large number of "jelly beans" rather than utilizing a much smaller number of high-end servers.

For example, IBM and Google have announced plans to make a 1,000 processor cluster available to a few select universities to teach students how to program such clusters using a software tool called MapReduce [1]. Berkeley has gone so far as to plan on teaching their freshman how to program using the MapReduce framework.

As both educators and researchers, we are amazed at the hype that the MapReduce proponents have spread about how it represents a paradigm shift in the development of scalable, data-intensive applications. MapReduce may be a good idea for writing certain types of general-purpose computations, but to the database community, it is:

1. A giant step backward in the programming paradigm for large-scale data intensive applications
2. A sub-optimal implementation, in that it uses brute force instead of indexing
3. Not novel at all -- it represents a specific implementation of well known techniques developed nearly 25 years ago
4. Missing most of the features that are routinely included in current DBMS
5. Incompatible with all of the tools DBMS users have come to depend on

https://homes.cs.washington.edu/~billhowe/mapreduce_a_major_step_backwards.html

MapReduce: Fighting Back

COMMUNICATIONS OF THE ACM

CONTRIBUTED ARTICLES

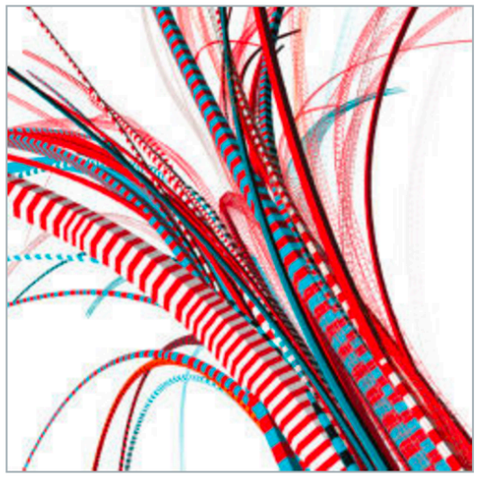
MapReduce: A Flexible Data Processing Tool

By Jeffrey Dean, Sanjay Ghemawat

Communications of the ACM, Vol. 53 No. 1, Pages 72-77
10.1145/1629175.1629198

Comments (3)

VIEW AS: SHARE:



Mapreduce is a programming model for processing and generating large data sets.[4] Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs and a reduce function that merges all intermediate values associated with the same intermediate key. We built a system around this programming model in 2003 to simplify construction of the inverted index for handling searches at Google.com. Since then, more than 10,000 distinct programs have been implemented using MapReduce at Google, including algorithms for large-scale graph processing, text processing, machine learning, and statistical machine translation. the Hadoop open source implementation of MapReduce has been used extensively outside of Google by a number of organizations.[10,11]

https://cacm.acm.org/magazines/2010/1/55744-mapreduce-a-flexible-data-processing-tool/fulltext

Friends or Foes?

CONTRIBUTED ARTICLES

MapReduce and Parallel DBMSs: Friends or Foes?

By Michael Stonebraker, Daniel Abadi, David J. DeWitt, Sam Madden, Erik Paulson, Andrew Pavlo, Alexander Rasin

Communications of the ACM, Vol. 53 No. 1, Pages 64-71
10.1145/1629175.1629197

Comments (4)

The MapReduce[7] (MR) paradigm has been hailed as a revolutionary new platform for large-scale, massively parallel data access.[16] Some proponents claim the extreme scalability of MR will relegate relational database management systems (DBMS) to the status of legacy technology. At least one enterprise, Facebook, has implemented a large data warehouse system using MR technology rather than a DBMS.[14]

Here, we argue that using MR systems to perform tasks that are best suited for DBMSs yields less than satisfactory results,[17] concluding that MR is more like an extract-transform-load (ETL) system than a DBMS, as it quickly loads and processes large amounts of data in an ad hoc manner. As such, it complements DBMS technology rather than competes with it. We also discuss the differences in the architectural decisions of MR systems and database systems and provide insight into how the systems should complement one another.

ARTICLE CONTENTS:

https://cacm.acm.org/magazines/2010/1/55743-mapreduce-and-parallel-dbmss-friends-or-foes/fulltext

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

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

Similarities

- DBMS can do whatever MapReduce can
  - User-defined functions provides equivalent functionality of a Map operation
- SQL aggregates can be used with user-defined functions to achieve Reduce functionality
- GROUP BY operation in SQL is equivalent to Reshuffle in MapReduce

Application Classes for MR

- Extract-transform-load (ETL) task and “read once” data set
- Complex analytics
- Semi-structured data
- Quick-and-dirty analyses
- Limited budget operations

DBMS for the Win

- Repetitive record parsing
- Less compression
- Less pipeline
- Weak scheduling
- No column-oriented storage

MapReduce Ecosystem

Many extensions to address limitations

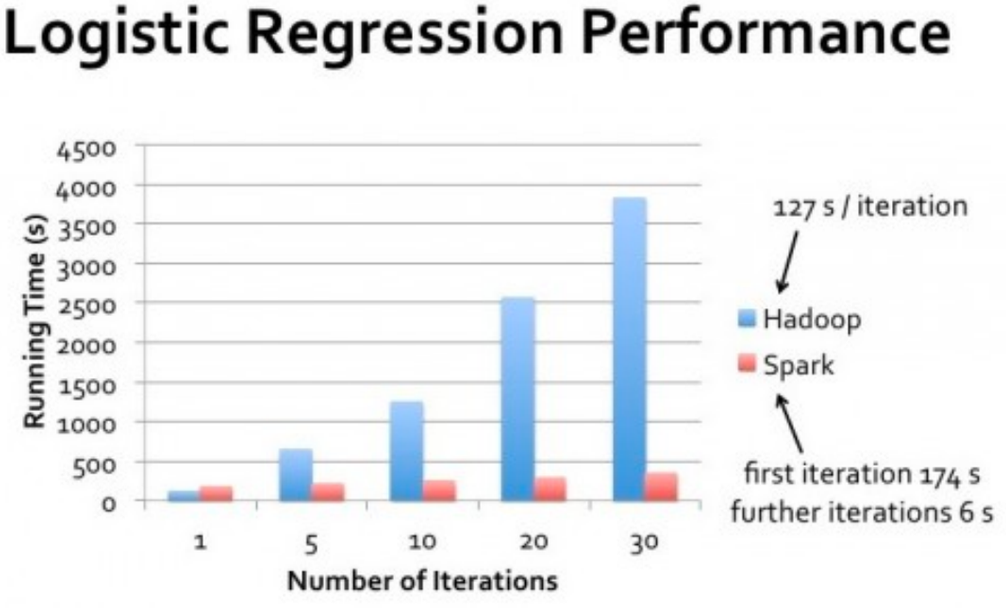
- Capabilities to write directed acyclic graphs of MapReduce jobs (e.g., PIG by Yahoo!)
- Declarative languages (e.g., Hive by Facebook or SQL/Tenzing by Google)
- Increased integration of DBMS with MapReduce

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