L09: Big Data, Intro to NoSQL Databases

DSAN 6300/PPOL 6810: Relational and Semi-Structured Databases and SQL Programming

Irina Vayndiner

October 26 and 30, 2023

GEORGETOWN UNIVERSITY

Logistics

- Update: starting today one SQL file per students' request
- HW3 is due Tue, Nov 7
- Q03 published on Thu 10/26, due Thu,11/9
- HW04 (mini-project) will be published: Thu 11/9
  - 130 points
  - Due Wed, 12/6 (no extensions!)
    - If you submit by Mon 12/4, you will get 3 bonus points ☺
    - Plan accordingly

- Guest Speaker on **Thu, 11/2 and Mon,11/6**
  - MongoDB lecture and Lab

- Reminder: Test: **Tue, 12/5 at 10:30am**
  - All my sections on zoom (Open books)
    - No early or late dates!

Agenda for today

- Lecture: Big Data and Intro to NoSQL databases
- Exercise in groups

Lecture Outline

- What is Big Data
- Technological innovations in databases
  - Shared-nothing Architectures
  - Hardware and Hardware Acceleration
  - Columnar
  - Distribution Key
- Big Database Vendors
- What is NoSQL

Big Data Technical Overview

Motivation

- Databases we considered so far are traditional transactional databases; they are optimized for running production systems--everything from websites to banks to retail stores.
- They excel at
  - reading and writing *individual rows* of data very quickly
  - while maintaining data integrity.
- Recently analytical, rather than operational, tasks are becoming more popular and important, like
  - Loading and preparation of large data sets for analysis (ETL tasks)
  - Creating reports and training predictive models on large volumes of data
- This *Big Data* revolution required changed how data is stored and retrieved
- However, SQL remains the tool of choice for data manipulation
- Today we will examine what these Big Data technologies are and how SQL is used there

What is "Big Data"?

- O'Reilly:
  - "Big data is when the size of the data itself becomes part of the problem"

- EMC/IDC:
  - *"Big data technologies describe a new generation of technologies and architectures, designed to economically extract value from very large volumes of a wide variety of data, by enabling high-velocity capture, discovery, and/or analysis."*

- IBM: (The famous 3-V's definition)
  - Volume (Gigabytes -> Exabytes)
  - Velocity (Batch -> Streaming Data)
  - Variety (Structured, Semi-structured, & Unstructured)

Credit: Big Data Now, Current Perspectives from O'Reilly Radar (O'Reilly definition); Extracting Value from Chaos, Gantz et al. (IDC definition); Understanding Big Data, Eaton et al. (IBM definition)

Data Size Terminology

Data inflation

2

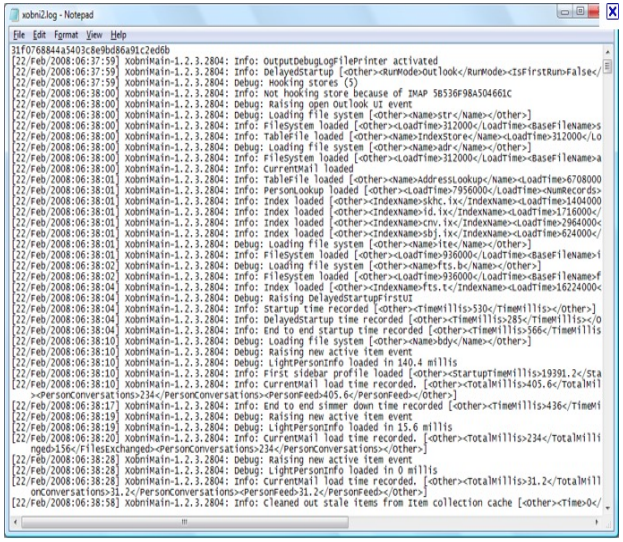
Unit	Size	What it means
Bit (b)	1 or 0	Short for "binary digit", after the binary code (1 or 0) computers use to store and process data
Byte (B)	8 bits	Enough information to create an English letter or number in computer code. It is the basic unit of computing
Kilobyte (KB)	1,000, or 2^{10}, bytes	From "thousand" in Greek. One page of typed text is 2KB
Megabyte (MB)	1,000KB; 2^{20} bytes	From "large" in Greek. The complete works of Shakespeare total 5MB. A typical pop song is about 4MB
Gigabyte (GB)	1,000MB; 2^{30} bytes	From "giant" in Greek. A two-hour film can be compressed into 1-2GB
Terabyte (TB)	1,000GB; 2^{40} bytes	From "monster" in Greek. All the catalogued books in America's Library of Congress total 15TB
Petabyte (PB)	1,000TB; 2^{50} bytes	All letters delivered by America's postal service this year will amount to around 5PB. Google processes around 1PB every hour
Exabyte (EB)	1,000PB; 2^{60} bytes	Equivalent to 10 billion copies of *The Economist*
Zettabyte (ZB)	1,000EB; 2^{70} bytes	The total amount of information in existence this year is forecast to be around 1.2ZB
Yottabyte (YB)	1,000ZB; 2^{80} bytes	Currently too big to imagine

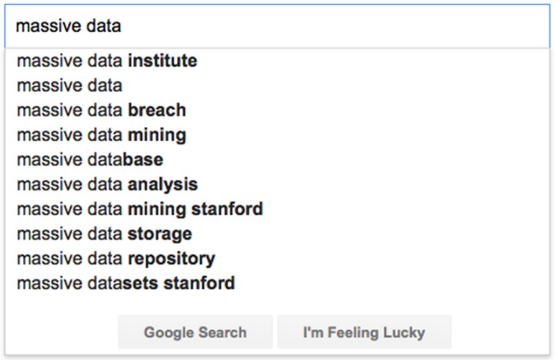
Source: *The Economist*

The prefixes are set by an intergovernmental group, the International Bureau of Weights and Measures. Yotta and Zetta were added in 1991; terms for larger amounts have yet to be established.

Why Data is Getting Bigger?

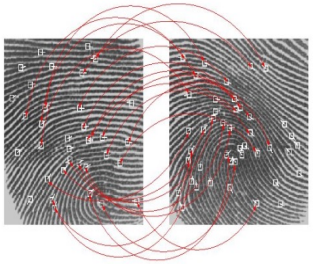
- 1) Internet-Scale Datasets
  - Activity logfiles (e.g., clickstreams, network logs)
  - Internet indices
  - Relationship data / social networks
  - By 2020 an average of nearly 2 Megabytes of data was generated every second for every person on the planet. (source: DataNami)





Why Data is Getting Bigger?

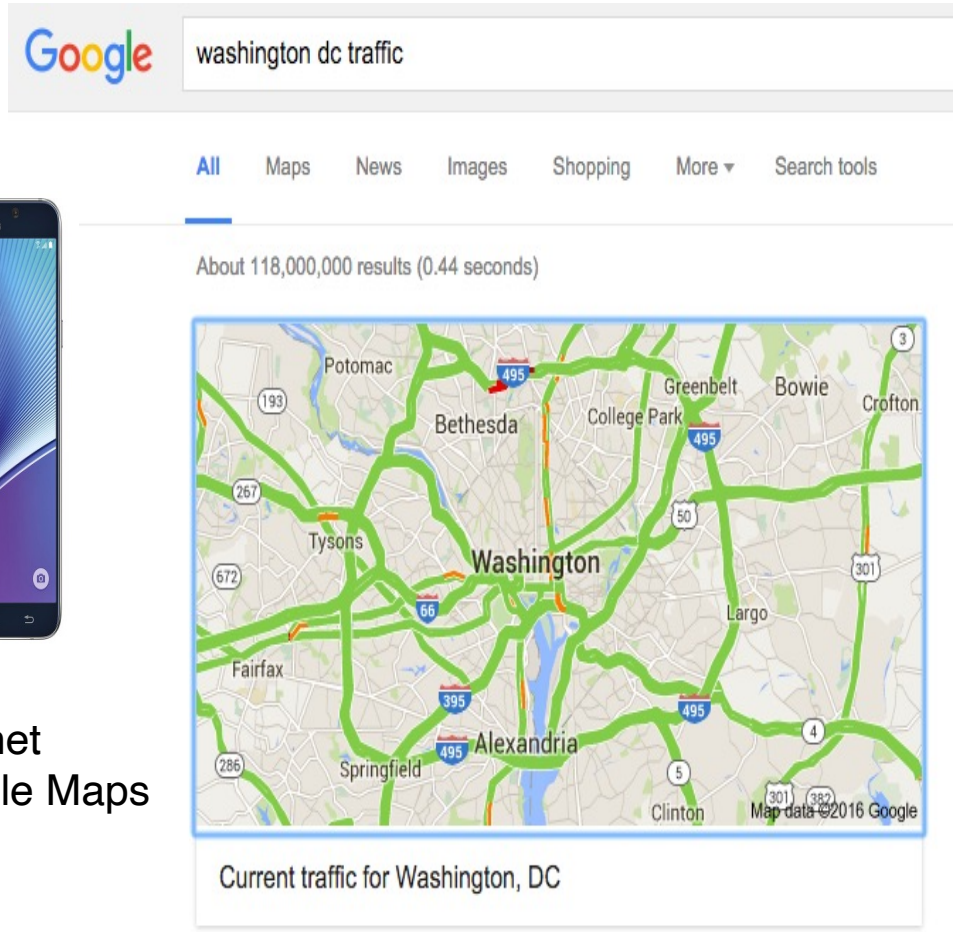
- 2) Sensor Proliferation
  - Radar; medical & scientific instruments; cameras; GPS feeds ...
  - Large Hadron Collider generates up to 30PB/year
  - High Def UAVs that collect 1.4PB/mission
  - Increasing number of various sensor feeds

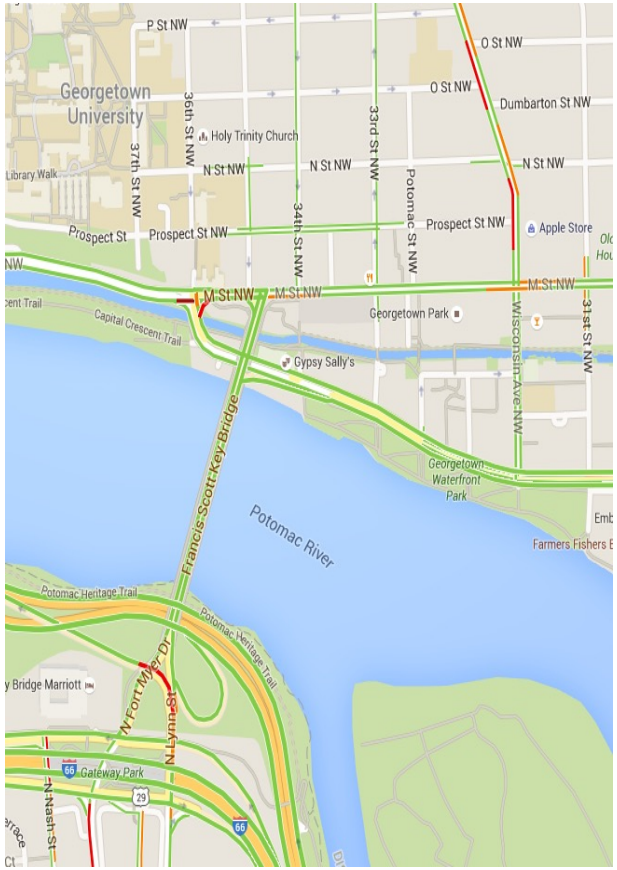


“Massive data” Over Time (continued)

- Today’s “massive data” — Using millions of cell phones as mobile traffic sensors

GPS
Internet
Google Maps





Street Level Detail

A Simple Data Structure Taxonomy

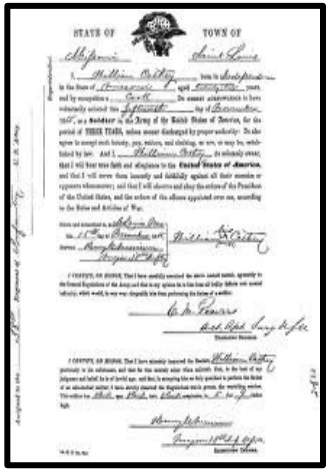
- Structured data
  - Data adheres to a strict template/schema
  - Spreadsheets, relational databases, …

	A	B	C	D	E
1	Month	Product	Store	Quantity	Amount
2	January	Skates	New York	4	$396
3	April	Skis Short	Toronto	4	$836
4	February	Skis Short	Toronto	4	$836
5	March	Skis Short	Montreal	4	$836
6	January	Skis Long	San Francisco	4	$980
7	April	Snow Board	Montreal	4	$1,232
8	February	Snow Board	New York	4	$1,232
9	March	Snow Board	Toronto	4	$1,232
10	March	Bikes	Montreal	4	$1,500
11	February	Skates	Montreal	21	$2,079
12	February	Skates	San Francisco	21	$2,079
13	January	Skates	Montreal	21	$2,079
14	April	Skates	San Francisco	25	$2,475
	April	Skis Short	Montreal	21	$4,389
	February	Skis Short	Montreal	21	$4,389
	January	Skis Short	Toronto	21	$4,389
	January	Skis Short	New York	21	$4,389
	March	Skis Short	San Francisco	21	$4,389
	April	Bikes	Montreal	12	$4,500
	January	Bikes	Toronto	12	$4,500
	April	Skis Long	New York	21	$5,145

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD
  "http://www.w3.org/TR/html4/stri
<html>
  <head>
    <title>Example</title>
    <link rel="stylesheet" href="s
  </head>
  <body>
    <div id="header">
      <h1><a href="." title="Back
    </div>
    <div id="toolbar">
      <span class="left">Today <sp
      <span class="right">
        <span id="time"> </sp
        <select id="timezone">
          <option value="-12">(GMT
          <option value="-11">(GMT
```

- Semi-structured data
  - Data adheres to a flexible (grammar-based) format
    - Optional fields, repeating fields
  - Web pages / forms, documents, XML, JSON, …

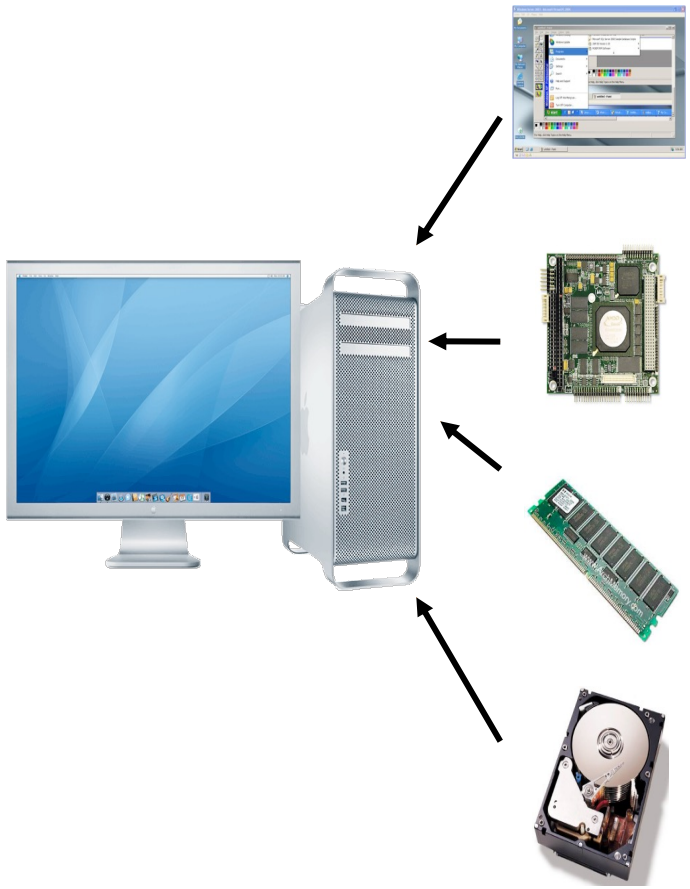
- Unstructured data
  - Data adheres to an unknown format
    - No schema or grammar; you discover what each byte is and means by examining the data
  - Unparsed text, raw disks, raw video & images, …



- "Variety": constantly coping with structure variations; multiple types; changing types

Approaches to Managing Big Data

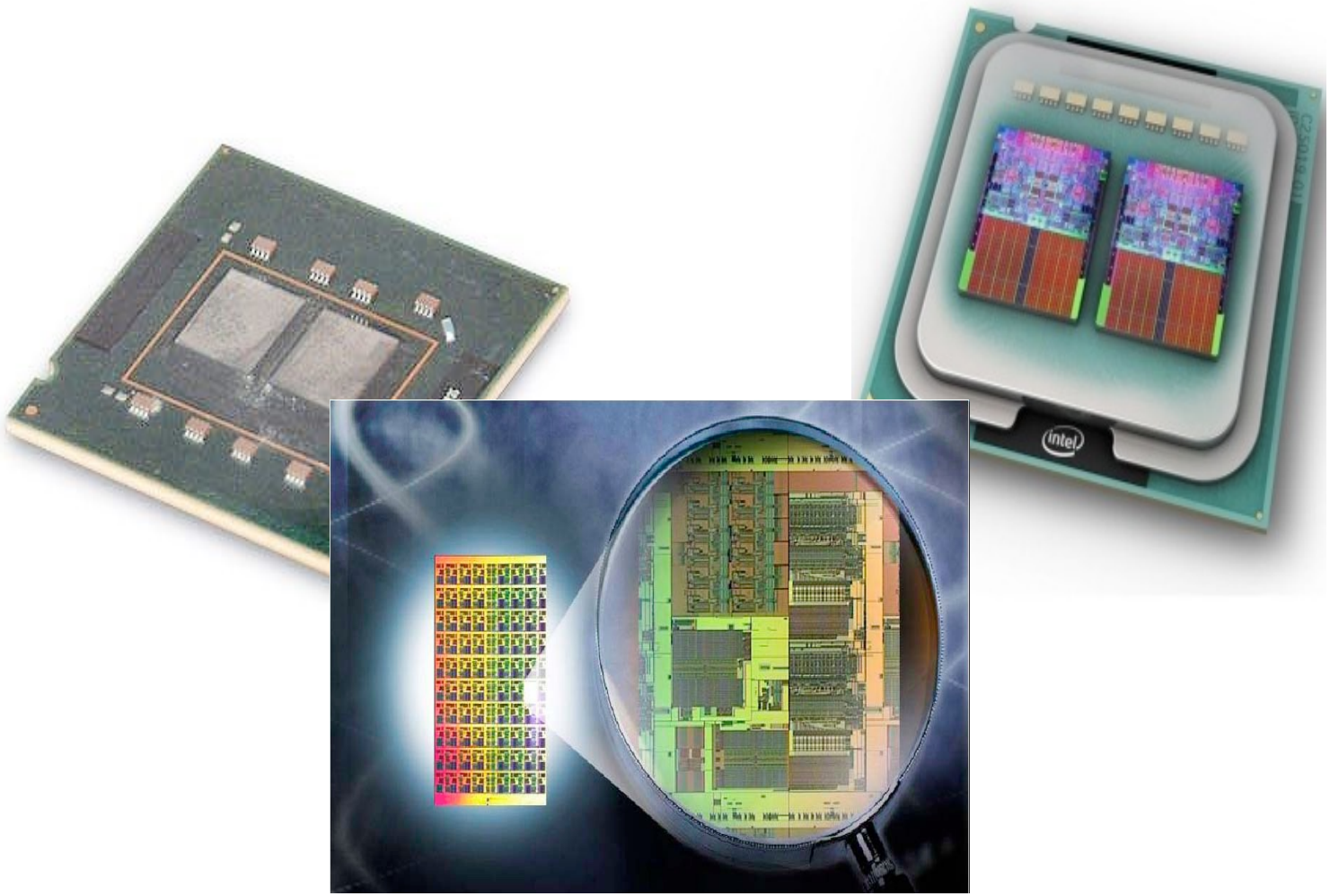
Scaling “Up”: Improve The Components of One System



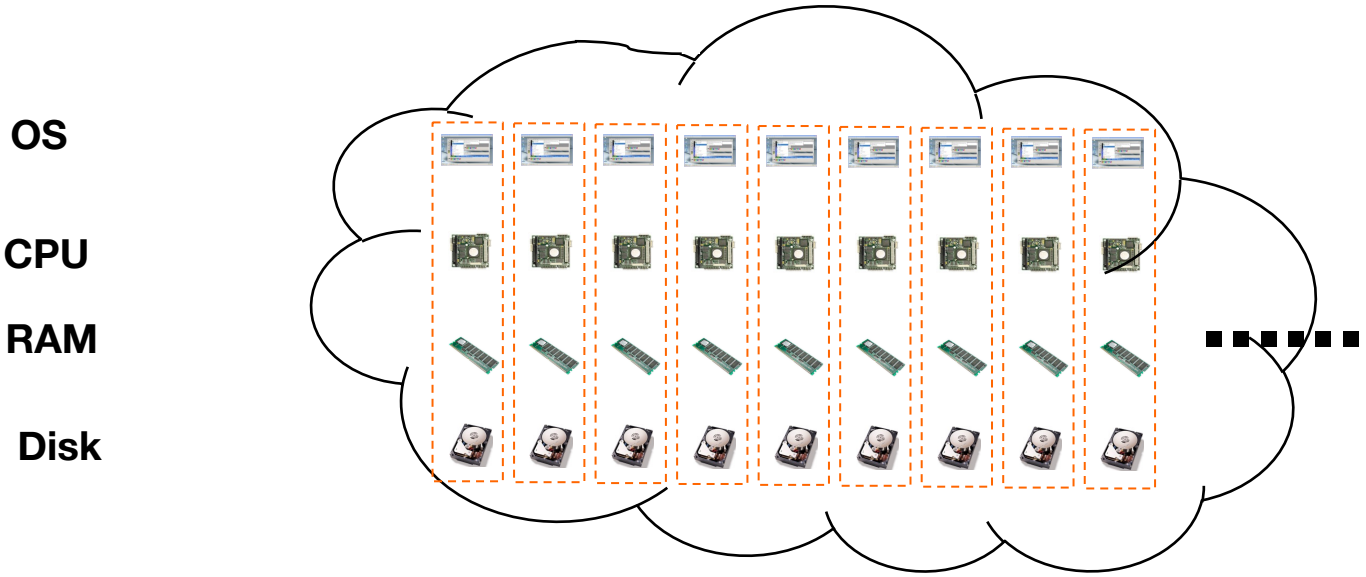
- OS: multiple threads / VMs
- CPU: increase clock speed, bus speed, cache size
- RAM: increase capacity
- Disk: Increase capacity, decrease seek, add platters

Scaling "Out": From Component Speedup to *Distributed Computing*

- Multicore cores on a chip (2, 4, 6, 8, ...)

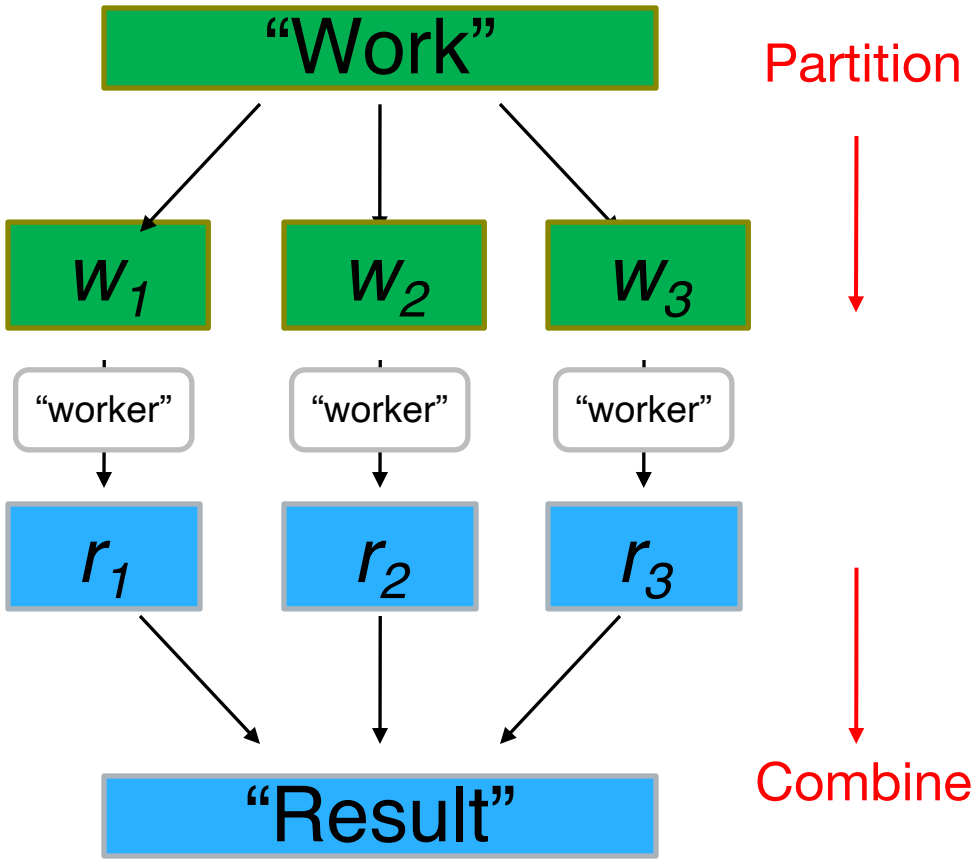


The Resulting "Computer" and its Applications



- This massively parallel architecture can be treated as a single computer
- Applications for this "computer":
  - Can exploit computational parallelism (near linear speedup)
  - Can have a vastly larger effective address space

Scales Using The Power of Parallelism: Divide & Conquer



Source: a slide by Jimmy Lin, cc-licensed

Technological Innovations

Software

Shared-nothing architectures (MPP)

Compressed storage

Columnar and hybrid storage

Integration of SQL and NoSQL

In-memory databases

Hardware

Servers with multi-TB of SSDs

Multi-core CPUs per commodity server

100's-GBs of RAM

Cloud Databases

Hardware acceleration: e.g. FPGAs

1-Gig to 10-Gig to Infiniband Ethernet

"Organizations seek to deploy analytics against "big data" sources that are high in volume and variety" TDWI

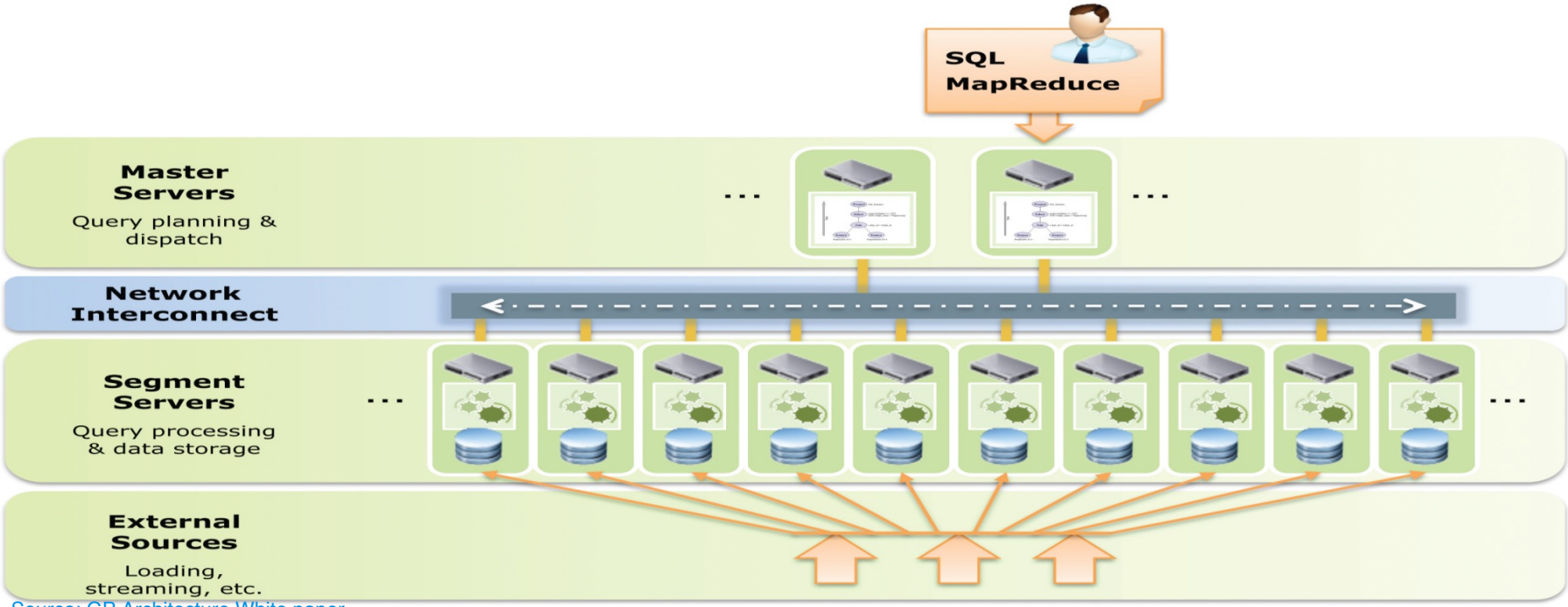
"Shared-nothing" Architecture

Massively Parallel Processing (MPP)

- Divide-and-conquer approach
- Massive refers to the concept of dozens or hundreds of nodes tied together achieving a single process
- Typical MPP architecture implements a shared-nothing (SN) paradigm where each parallel node operates self-sufficiently ("autonomously") and controls its own memory and disk
  - A collection of nodes, each with local disk and local main memory, connected together by a high-speed network interconnect
  - Processor-disk pairs operating in parallel divide the workload to execute queries over large sets of data.
  - Adding nodes increases performance as well as capacity, scales linearly as new nodes are added into the system, to 100s of TBs and into PBs

Massively Parallel Relational Databases

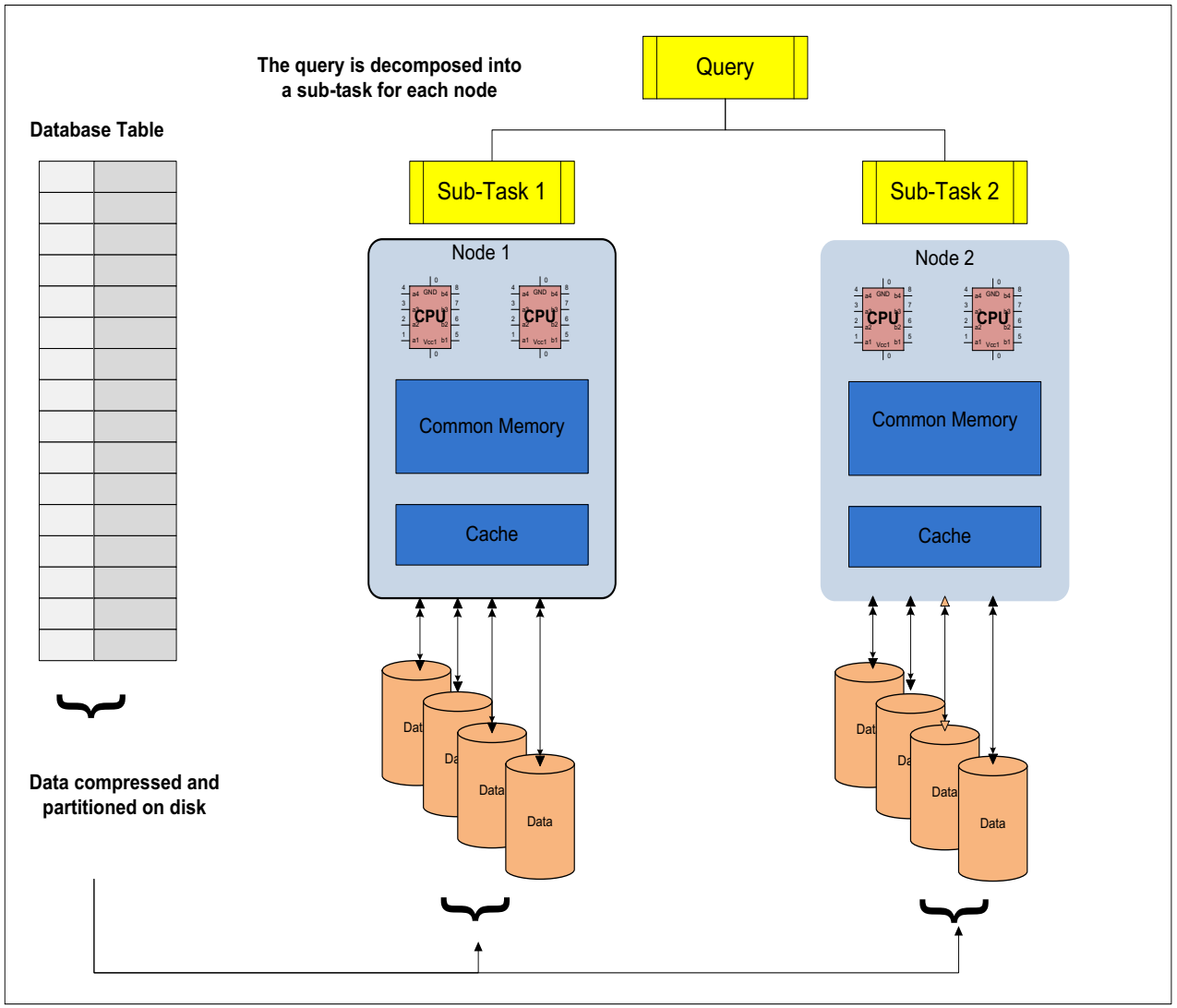
- Nodes are connected by a high-speed Network Interconnect
- Master node acts as a “brain” that ties all the compute nodes together
  - Determines the optimal query plan
  - Dispatches that query to all the compute nodes
  - Gathers results from compute nodes and sends them to the user. All nodes



Source: GP Architecture White paper

MPP Scales Out by Adding Nodes

- MPP parallelizes the query execution of a RDBMS, and splits queries and allocates them to multiple DBMS nodes
- All nodes can process in parallel
- Linear scalability by adding nodes
  - Each adds storage, query performance and loading performance



Some MPP RDBMS Characteristics

- Horizontal partitioning of relational tables (sharding)
  - Data is distributed by a user-defined column(s), a “distribution key”
  - *Example1:* ZIP code, range:
    - Customers with ZIP codes less than 50000 are stored in CustomersEast
    - Customers with ZIP codes greater than or equal to 50000 are stored in CustomersWest
  - *Example 2*: In an MPP Database, the following statement creates the table ORDERS using O_ORDERKEY as a hash key, i.e., the rows of the table will be assigned to segments based on the hash value of their respective order key:
    - CREATE TABLE ORDERS (O_ORDERKEY INT, ...) DISTRIBUTED BY (O_ORDERKEY)
  - Multi-level table partitioning
    - Partitions within partitions
    - Up to 15 partitions per table (for Teradata)
    - Partition “pruning” for unnecessary partitions (e.g. Teradata Aster)
  - The choice of distribution key could affect the performance of query workloads significantly => DBA’s task

Distribution Key Discussion

- What's the Distribution Key
- Choice of the Distribution Key: a crucial driver of MPP performance
- Best Practices for Distribution Key process
  - Planning & Analysis
  - Implementation
  - Recommendations

What's a Distribution Key?

- The Distribution Key defines the distribution policy for the data in database tables, which has great bearing on system performance
- The goals for the distribution policy are to:
  - Distribute the volume of data and query execution work evenly among all the segments
    - Enable segments to accomplish the most expensive query processing steps locally.
    - MPP environment's overall response time for a query is measured by the performance time for all segment instances ("system is as fast as its weakest link")
- Defining an effective distribution policy requires an understanding of:
  - Data's characteristics
  - Kinds of queries that will be run once the data is loaded into the database
  - Which distribution strategies best utilize the parallel execution capacity of the segments.

Planning the Distribution Key: review the workload

- Analyze the workload
- Workload significance for distribution key choices
  - In example 1, data is distributed evenly between the nodes, but if in one week you only need to run queries about CW and in another week only about CE, then each time only one node is working at a time which is not what you want.
  - Conclusion: bad distribution key was chosen
- Selecting the most effective distribution key is not always an obvious choice => DBA's task

Planning of the Distribution Key: obtaining local joins

- Use the Same Distribution Key for Commonly Joined Tables
  - When a local join is performed, a segment operates independently of the other segments without network traffic or communication between segment instances
  - Distribute on the same key used in the join (WHERE clause) to obtain local joins
  - Perform a join operation between the two tables locally (=within the segment) without network activity.
  - "If you distribute a pair of tables on the joining keys, the leader node collocates the rows on the slices according to the values in the joining columns so that matching values from the common columns are physically stored together".

    https://docs.aws.amazon.com/redshift/latest/dg/c_choosing_dist_sort.html
- Issues to watch for while planning for the Distribution Key
  - Distribution keys must be the same data type and length to obtain a local join
  - Distribution Key has to have high cardinality
    - Low cardinality leads to distribution to only few segments (e.g M/F/U)
  - Unique Keys are good candidates, for example, a primary key

Planning of the Distribution Key (continued)

- Planning of distribution key need to be based on actual values stored
  - Presence of the default values (e.g. customer_id = 0, whenever it is not known) can significantly skew the data distribution across the segments

```
column    | type | encoding | distkey
userid    | integer | none     | t |
username | character(8) | none | f |
firstname | character varying(30) | text32k | f |
```

Source: https://docs.aws.amazon.com/redshift/latest/dg/c_Distribution_examples.html

Distribution Key: Implementation

- Use the DISTRIBUTED clause of the CREATE TABLE statement to define the distribution policy for a table.
- Ideally, each segment will store an equal volume of data and perform an equal share of work when processing queries.
- There are two kinds of distribution policies:
  - DISTRIBUTED BY [[KEY]] (column, ...) defines a distribution key from one or more columns.
    - A hash function applied to the distribution key determines which segment stores the row. Rows that have the same distribution key are stored on the same segment. If the distribution keys are unique, the hash function ensures the data is distributed evenly. The default distribution policy is a hash on the primary key of the table, or the first column if no primary key is specified.
  - DISTRIBUTED RANDOMLY distributes rows in round-robin fashion among the segments.
    - When different tables are joined on the same columns that comprise the distribution key, you want the join to be accomplished at the same segments, which is much faster than joining rows across segments that may be the case with random distribution.
    - It is the best practice to define a non-random distribution key to optimize joins
    - For Amazon Redshift this options is represented by “DISTRUBUTED **EVEN**”

Distribution Key Recommendations

- Customer Example:
  - By changing the distribution key from random to the primary key, customer was able to achieve 100x times performance increase
    - This improvement was achieved for the selected queries

- Things to consider:
  - Workload evaluation takes time and effort
  - Requirements not only as of now but also going forward
  - If a chosen distribution key is a bad distribution key and needs to be updated, BDA database system will need to be stopped for a while
    - Distribution Key cannot be updated, and therefore complete delete, table re-create, and reload of all data is required
    - Updating the distribution key usually is not a viable plan while system is in production
    - Walk-around: build another system next to the first one (price!)

Distribution Key Exercise

- In Groups
- Each group reports the result

30

Columnar Storage

ID	First Name	Last Name	Salary
1	Marge	Abbot	$50,000
2	Colleen	Calder	$88,000
3	Ruth	Vanderbilt	$91,000
4	John	Malone	$105,000

ID	First Name	Last Name	Salary
1	Marge	Abbot	$50,000
2	Coleen	Calder	$88,000
3	Ruth	Vanderbilt	$91,000
4	John	Malone	$105,000

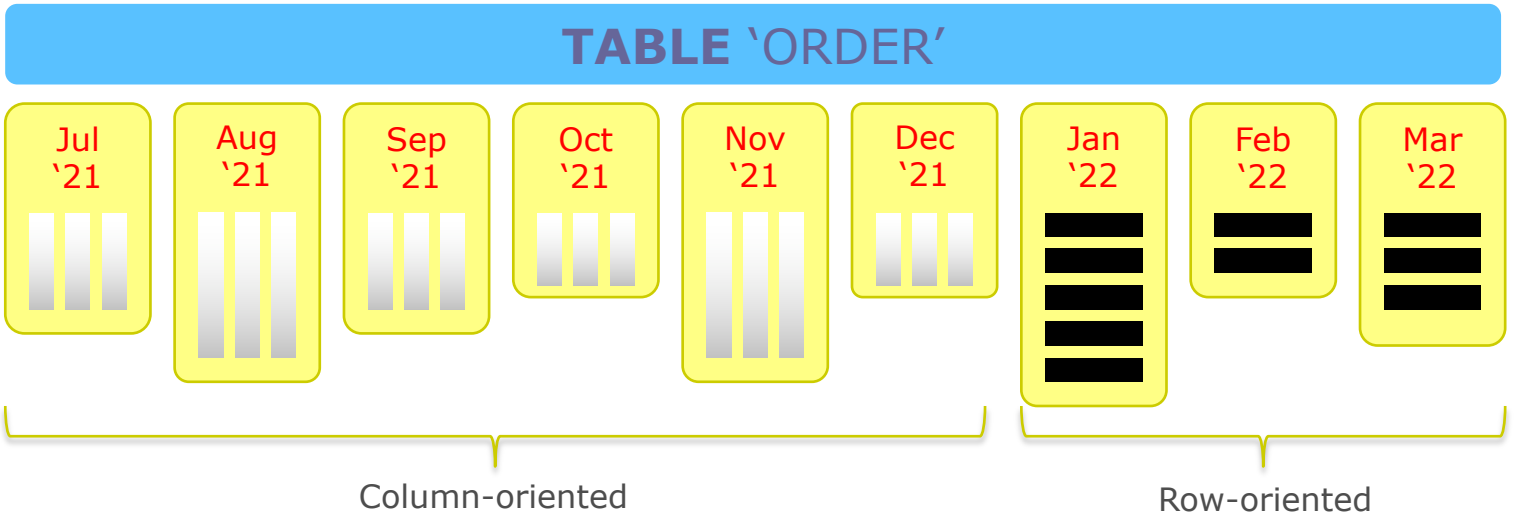
A columnar (column-oriented DBMS) is a database management system which stores its content by column rather than by row

Columnar Storage (continued)

- Columnar are better for
  - Queries that look for an aggregate over many wide rows but only for a small subset of all columns of data (helps I/O)
    - Example: avg age?
  - If a search is being done for items matching a particular value in a column of data, only the storage objects corresponding to that data column within the table need to be accessed
  - Updates on a column that are supplied for all rows at once
  - Great compression potential (based on the similarity of adjacent data)
    - Reduce Data storage space
    - Reduce Transmission (I/O) Capacity
- Row-based are better for
  - Queries where many columns of a row are requested at the same time
  - Row-size is relatively small, as the entire row can be retrieved with a single disk seek
- Hybrids
  - “Best of both worlds”

Hybrid Example

- Greenplum's (Polymorphic) Data Storage
  - The ability to support row- and column-oriented tables in the same database, with unrestricted ability to join these tables efficiently
    - by specifying 'WITH (orientation=column)'
  - A single partitioned table could have older data stored as 'column-oriented' for deep compression and more recent data as 'row-oriented'.



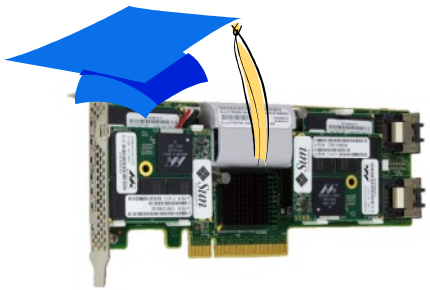
Commodity Servers, High-speed Interconnects

- Modern commodity hardware in a single (inexpensive) server:
  - Two or more quad-core processors
  - 16 to 64 GB RAM
  - Multi-terabyte storage, often with SSDs

- Interconnects
  - From 1-Gig (1 Gigabit per sec) to 10-Gig Ethernet
  - Infiniband
    - According to HPC White Paper:
      - 10GigE has 5-6 times the latency of InfiniBand
      - InfiniBand has 3.7x the throughput of 10GigE
      - Performance difference of MPP system grows rapidly as the number of nodes increases

Solid State Drives (SSDs)/Flash Cache

- Solid-state drive (SSD) are flash memory based and do not employ any moving mechanical components, which distinguishes them from traditional hard disk drives (HDDs) containing spinning disks and movable read/write heads.
- These days most SSDs use NAND-based flash memory, which retains data without power.
- For applications requiring fast access, but not necessarily data persistence after power loss, SSDs may be constructed from random-access memory (RAM).



Solid State Drive (SSD) vs. Hard Disk Drive (HDD)

SSD is more than 20 times faster for concurrent query workloads.

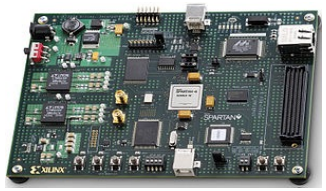
SSD technology has been developing rapidly.

	Enterprise SSD	Enterprise HDD
IOPs	10^5	10^2
Sequential Read (MB/s)	Up to 600	Up to 150
Random Access Time	0.1ms	2.9ms
Cost per GB	$0.59	$0.05

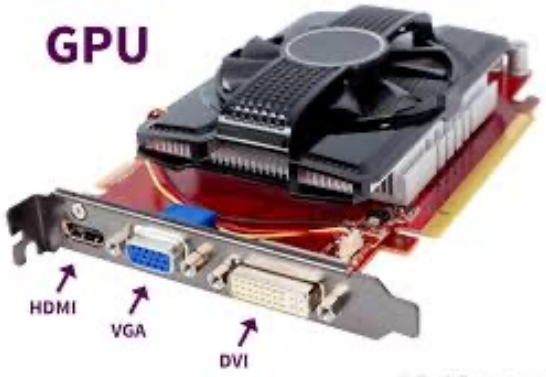
Hybrid drives or solid-state hybrid drives (SSHDs) combine the features of SSDs and HDDs in the same unit, containing a large hard disk drive and an SSD cache to improve performance of frequently accessed data

Databases and Hardware Acceleration

XilinxSpartan 6 FPGA



- Hardware acceleration
  - Use computer hardware to perform some function faster than is possible in software running on the general-purpose CPU.
  - Many hardware accelerators are built on top of field-programmable gate array (FPGA) chips or graphics processing unit (GPU).
  - Some operations performed on FPGA or GPU instead of CPU.



Databases and Hardware Acceleration

FPGA Examples

- **IBM** PureData for Analytics uses FPGA
  - FPGA implementation performs complex data filtering so that only relevant portions of data sets are passed along to the processor to run the SQL query.
  - FPGA cycles are also used to perform data compression and decompression, reducing the load on the processor.
- **Swarm64** used to accelerate PostgreSQL
- **Centaur**: Relational engine in hybrid CPU-FPGA architectures.
  - More info: https://ieeexplore.ieee.org/document/7966689

GPU Examples

- **Kinetica** DB (in-memory DB, company is located in Arlington, VA)
  - "Can perform standard SQL queries on billions of rows in microseconds while concurrently visualizing results, executing machine-learning models and ingesting large amounts of streaming data."
- **SQream** DB (relational, column-oriented)
  - Aimed at the budget multi-terabyte analytics market, due to its modest hardware requirements and use of compression.
  - Partnered with Alibaba group's Alibaba Cloud to deliver a GPU Database solution on Alibaba Cloud
- Microsoft: virtualized CPU-GPU servers on Azure cloud

In-Memory Relational Databases

- Pro: Performance
  - Orders of magnitude than reading from/writing to disk
  - Faster than SSDs
- Con:
  - Stores data entirely in-memory (price)
  - Resource management in multi-user environments
- Example of Vendors:
  - SAP HANA - leader (column-store, also available as appliance)
  - IBM DB2 with Blue Acceleration – InfoSphere DataWorks
    - Almost 100 times performance acceleration
    - www.ibmBLUhub.com
  - Microsoft SQL Server with in-memory OLTP
  - Oracle DB with in-Memory Option
  - VoltDB, MemSQL, many others
- More examples:
  - https://en.wikipedia.org/wiki/List_of_in-memory_databases

Multi-Temperature Data Concept and Examples

5 Temperatures of Data

Data Temperature Classification	Usage Definition	Retail Campaign Example
WHITE HOT	Continuous, often unexpected spikes of repeated data access, usually reflecting the need for rapid response times and intensive usage. Tends to be short-term, but nonetheless can have a significant impact across the overall data warehouse performance. Can indicate a transient business or technical issue.	In the first day or two of the campaign (and often the immediate period before the campaign goes live), the data is repeatedly accessed and queried in a manner not reflective of its normal use. This data is characterized as white hot because hot alone does not reflect the extreme nature of the access frequency.
HOT	Frequently accessed data, reflecting tactically driven behavior and typically short-term need for high-priority, business-driven decision support activity. Can affect data warehouse users, response times and overall performance.	Initially, the data is accessed frequently as various users analyze the sales figures to understand the in-traday effectiveness of the campaign and reports are compiled for management. At this stage, the data is hot—frequently accessed, perhaps by multiple users.
WARM	Data accessed less frequently and usually with less urgency, often reflecting normal decision support systems requirements. Response times are less critical than those reflected by hot data. Normally has less impact to the data warehouse and end-user performance than hot or white-hot data.	In a month or two, the campaign is closed or changed, and the data is analyzed to review the veracity of the offers, the profitability of the exercise and the effect on competitive market share. Now that new campaigns are being launched, the data regarding the earlier campaign is less frequently accessed and probably by fewer users. The data is now warm.
COLD (OR COOL)	Deep historical information usually seen in data mining and analysis activities. Can also be legacy data or long-term business-cycle data for infre-quent or occasional retrospective comparison.	Once the campaign is over and the reports com-pleted, the data is reviewed very infrequently, perhaps as part of an end-of-year review, with little additional value being created from the information itself. With this level of inactivity, the data is now cold—barely used and probably by very few users.
DORMANT	Data that has not been accessed for a consid-erable period of time or not at all. Typically archived data kept for regulatory, reference or legal reasons. When used, performance require-ments are usually not an issue.	The campaign is long finished, as are any analy-sis and end-of-year reviews. The data will remain untouched indefinitely. It is now dormant.

Source: Teradata

SQL and Hadoop

- Opens Big Data to a much larger audience
- Use of SQL tools: BI, ETL, Visualization
- Examples of SQL-on-Hadoop:
  - Apache Hive (the first one)
  - Cloudera Impala
  - Most large RDBMS vendors added this functionality
    - HP Vertica (MapR)
    - Teradata Hadapt
    - MS SQL PolyBase
  - Apache HAWK (aka “Apache Hadoop Native SQL”, former Pivotal HAWK)
  - Apache Spark (Spark SQL)
    - Cannot do what databases can do well: individual updates/inserts

SQL and Hadoop (continued)

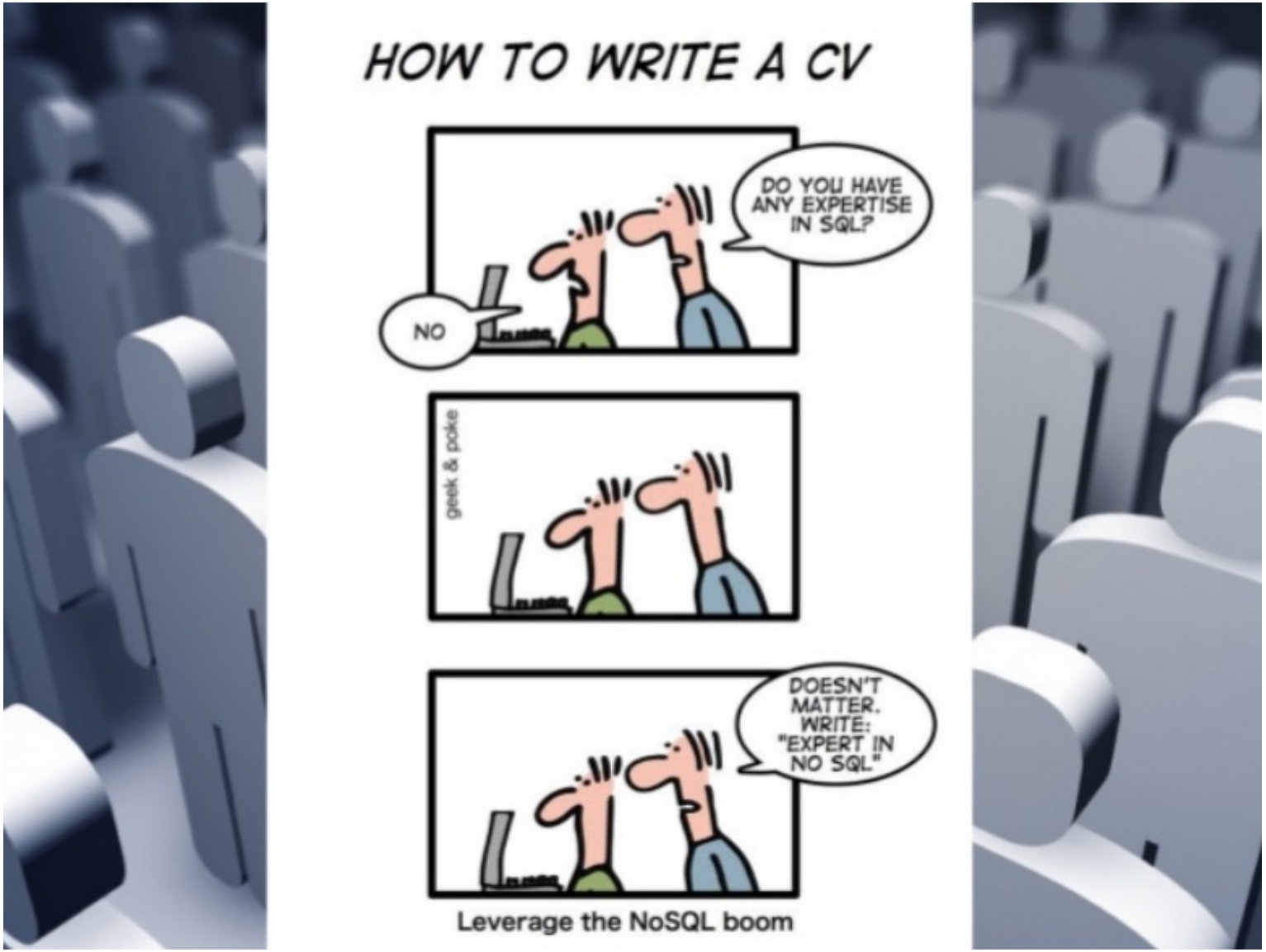
- Federated DB Approach
  - Research: “reinventing” federated DB approach: Polystores* (M. Stonebraker, ACM SIGMOD)
  - Apache Presto, developed by Facebook
    - https://prestodb.github.io/
    - A single Presto query can combine data from multiple sources
      - E.g. Hadoop, AWS S3, MySQL, Kafka, and MongoDB
    - Does not write intermediate results to disk
    - Facebook uses Presto for interactive queries against several internal data stores, including their 300PB data warehouse.
    - Over 1,000 Facebook employees use Presto daily to run more than 30,000 queries that in total scan over a petabyte each per day.
    - Netflix used it for 10PB of storage in Amazon S3
      - More info : https://medium.com/netflix-techblog/using-presto-in-our-big-data-platform-on-aws-938035909fd4
    - More info:
      - https://prestodb.io/ (latest update March 2022)
      - https://trino.io/

*Source:http://wp.sigmod.org/?p=1629

Examples of Open-Source Big Data RDBMSs and Hybrids

- RDBMSs
  - MySQL
  - Greenplum
  - MariaDB (used by Wikipedia, Facebook, Google)
  - PostgreSQL
- Hybrid
  - Apache AsterixDB
    - BDMS (Big Data Management System)
    - Supports: NoSQL, HDFS, Indices
    - Goal: Open source “Postgres-quality” flexible schema DBMS
    - For more info: https://asterixdb.apache.org/

NoSQL Databases



Adapted from https://www.slideshare.net/crisgomal8/nosql-databases-32933737

NoSQL Databases vs Relational databases

RDBMS	NoSQL
Stores data in tables, and uses SQL interface	Different storage models, e.g. stores data in a collection of (JSON) "documents"
Data are normalized.	Data are *denormalized*.
Structured Data	Semi-structured or unstructured data
Designed for transactions	Designed for speed and *ad hoc* changes

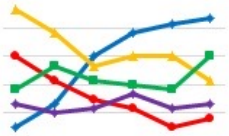
- "NOSQL" stands for “Not-Only-SQL”: many "NoSQL" databases can now be queried with SQL.

MySQL Ranking

DB-Engines Ranking

The DB-Engines Ranking ranks database management systems according to their popularity. The ranking is updated monthly.

Read more about the method of calculating the scores.



trend chart

351 systems in ranking, August 2019

Rank			DBMS	Database Model	Score		
Aug 2019	Jul 2019	Aug 2018			Aug 2019	Jul 2019	Aug 2018
1.	1.	1.	Oracle	Relational, Multi-model	1339.48	+18.22	+27.45
2.	2.	2.	MySQL	Relational, Multi-model	1253.68	+24.16	+46.87
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1093.18	+2.35	+20.53
4.	4.	4.	PostgreSQL	Relational, Multi-model	481.33	-1.94	+63.83
5.	5.	5.	MongoDB	Document	404.57	-5.36	+53.59
6.	6.	6.	IBM Db2	Relational, Multi-model	172.95	-1.19	-8.89
7.	7.	↑ 8.	Elasticsearch	Search engine, Multi-model	149.08	+0.27	+10.97
8.	8.	↓ 7.	Redis	Key-value, Multi-model	144.08	-0.18	+5.51
9.	9.	9.	Microsoft Access	Relational	135.33	-1.98	+6.24
10.	10.	10.	Cassandra	Wide column	125.21	-1.80	+5.63
11.	11.	11.	SQLite	Relational	122.72	-1.91	+8.99
12.	12.	↑ 13.	Splunk	Search engine	85.88	+0.39	+15.39
13.	13.	↑ 14.	MariaDB	Relational, Multi-model	84.95	+0.52	+16.66
14.	14.	↑ 18.	Hive	Relational	81.80	+0.93	+23.86
15.	15.	↓ 12.	Teradata	Relational, Multi-model	76.64	-1.18	-0.77
16.	16.	↓ 15.	Solr	Search engine	59.12	-0.52	-2.78
17.	17.	↑ 19.	FileMaker	Relational	58.02	+0.12	+1.96
18.	↑ 20.	↑ 21.	Amazon DynamoDB	Multi-model	56.57	+0.15	+4.91
19.	↓ 18.	↓ 17.	HBase	Wide column	56.54	-1.00	-2.27
20.	↓ 19.	↓ 16.	SAP Adaptive Server	Relational	55.86	-0.79	-4.57

Some NoSQL Definitions

1. Wiki: A **NoSQL** (originally referring to "non SQL" or "non relational") database provides a mechanism for storage and retrieval of data that is modeled in means *other than* the tabular relations used in relational databases.

2. **NoSQL**, which stand for "not only SQL," is an alternative to traditional relational databases in which data is placed in tables and data schema is carefully designed before the database is built. Many "NoSQL" databases can now be queried with SQL; or or they sit alongside SQL databases

3. Forrester defines Big Data **NoSQL** as:

A nonrelational database management system that provides storage, processing, and accessing of any type of data and which supports a horizontal, scale-out architecture based on a schemaless and flexible data model.

NoSQL: Schema-less and Fast

- Schema: “We do not need to know the schema exactly from the start”
  - “Schema-less” for flexibility
  - “Dynamic schema”: not every record has the same properties
  - Has strong benefits in some cases
    - E.g. nested, sparse, semi-structured (e.g. text) data
    - Data that has no relation to other data (e.g. each JSON document is a stand-alone object)

      But in many cases leads to problem of implicit schemas
    - Not defined by you but by application behavior
- Increased Velocity with Reduced Functionality
  - Often no transactions
  - Often no constraints
  - Somewhat limited query functionality
    - Joins are very limited or non-existent
  - Need to choose between availability or consistency
    - Eventual (or tunable) consistency

For a distributed data store you cannot choose more than 2!

CAP Theorem

- **C**onsistency
  - All the servers in the system will have the same data so anyone using the system will get the same copy regardless of which server answers their request.
- **A**vailability
  - The system will always respond to a request (even if it's not the latest data or consistent across the system or just a message saying the system isn't working)
- **P**artition Tolerance
  - The system continues to operate as a whole even if individual servers fail or can't be reached..

Adapted from www.slideshare.net/ranjidausa/what-is-no-sql-and-cap-theorem

CAP Theorem and Eventual Consistency

- **Consistency**: Every read receives the most recent write or an error
  - **Eventual consistency** informally guarantees that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value

- **Availability**: Every request receives a (non-error) response.

Multiple copies of the shopping cart are allowed to exist and, if one of the replicas becomes unresponsive, the data can be served by one of the other replicas. However, because of network latencies, the copies may occasionally get out of sync, and the customer may occasionally encounter a stale version of the shopping cart. Once again, this can be handled appropriately by the application tier; the node that falls behind can catch up eventually, or inconsistencies can be detected and resolved at an opportune time, such as at checkout. This technique is called "eventual consistency." *AWS DynamoDB*

Google's Big Table

- First mentioned in an article in November 2006
- Is used internally by Google for web indexing, Google Earth, Google Finance etc.
- A sparse, column-oriented distributed storage system for managing structured data that is designed to scale to very large size
  - Petabytes across thousands of commodity computers
  - Supports *billions* of rows and *millions* of columns
  - Used GFS to store data and can be used with MapReduce
- Does not support SQL queries
- Data is indexed by a row id, column id (family) and timestamp
  - Handles missing values (sparse data) very well
- Column-oriented storage
  - Most queries only involve a few columns, so greatly reduced I/O

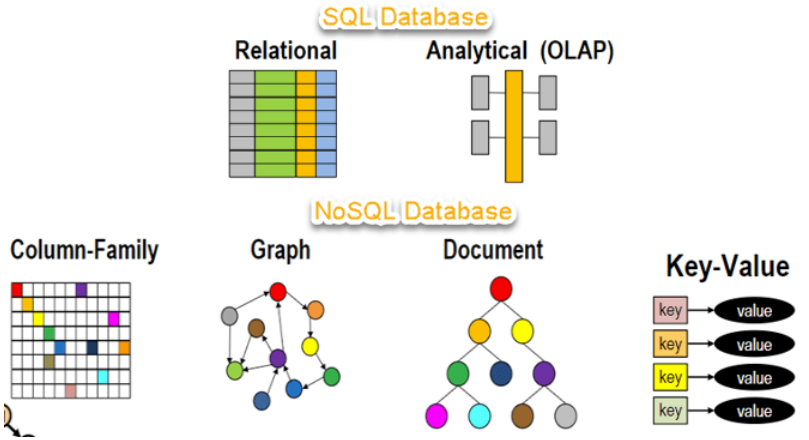
- "New customers get $300 in free credits to spend on Google Cloud during the first 90 days."

- Google Cloud BigTable NoSQL: https://www.youtube.com/watch?v=6YFaF45R3nI

Types of NoSQL databases and some examples

- **Wide Column-based**
  - Stores data in tables, rows, and columns
  - Names and format of the columns can vary from row to row in the same table.
  - Can store a very large number of columns (billions!)
- **Document-based**
  - Stores a collection of documents, e.g. JSON documents
  - Some attributes can be indexed.
  - Query language returns documents or part of documents.
- **Key-value based**
  - Stores pairs of keys and values
  - Retrieve values when a key is known.
- **Graph-based**
  - Good for establishing data relationships

- **Multi-models = c**ombine multiple capabilities
  - Examples: AWS DynamoDB, Azure Cosmos DB, Couchbase, Redis



NoSQL: Key-Value Stores

- Simplest form of database management
- Very fast
- Can add columns “on the fly”
- Eventually consistent (in some cases, tunable)
- Query by key
  - Given a key, return the value
  - Some are able to sort by keys => Range queries

Most suitable for

- Session information
- User profiles & preferences
- Event logging
- Blog comments
- Product recommendations
- Massive, rapidly arriving, highly non-homogenous datasets

Used by Twitter, Quora, etc.

Examples: AWS DynamoDB, Couchbase, Oracle NoSQL

NoSQL: Wide-Column Stores

- Stores data in tables, rows, and columns
  - Names and format of the columns can vary from row to row in the same table.
  - Can store a very large number of columns (billions!)
  - Can group related columns into column families:
    - A typical column family contains a row key as the first column, which uniquely identifies that row within the column family.
- Fast to load and query

Key	Product	Line	Size	Color	Color 2	Genera	Screen Size	Picture	Backlight	OS	Price	Qty	Cost
1	Shoes	Shoes	12								$50	1	$50
2	Jacket	Clothes	L	Blue	Gray						$69	1	$69
3	Apple Watch 3	Electronics	Large	Gold		3					$699	1	$699
4	Samsung SmartTV	Electronics				8	65"	4K	LED	Apple watchOS 4.0	$797	1	$797

Most Suitable for:

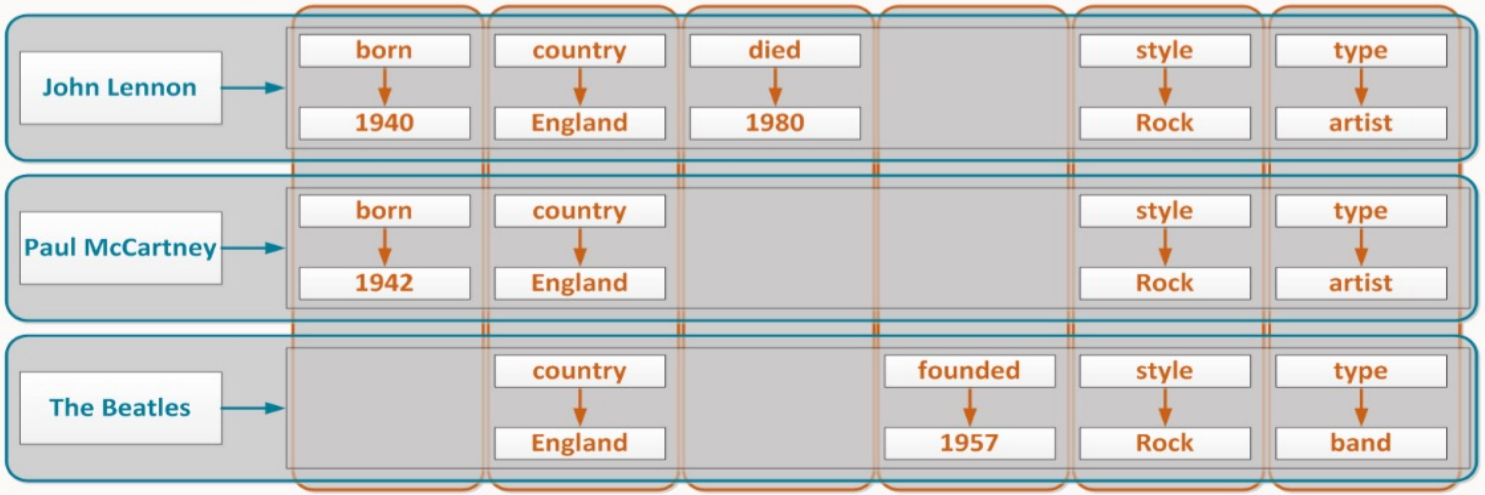
- Ad tech
- Fintech
- IoT

Examples: Accumulo, Cassandra, Hbase, Cloud BigTable (Google)

NoSQL: Wide-Column Stores (example)

- Table with single-row partitions

- Column family view



Real World Example

Product catalog for retailer selling in 20 countries

```
{
    _id: 375,
    en_US: { name: ..., description: ..., <etc...> },
    en_GB: { name: ..., description: ..., <etc...> },
    fr_FR: { name: ..., description: ..., <etc...> },
    fr_CA: { name: ..., description: ..., <etc...> },
    de_DE: ...,
    <... and so on for other locales ...>
}
```

mongoDB

https://www.slideshare.net/knowbigdata/no-sql-cap-theorem-part-of-mongodb-course

NoSQL: Document Stores

- Variable document schemas
- XML, JavaScript Object Notation(JSON) or BSON (Binary JSON) documents
- Tunable consistency (often at the expense of performance)
- Transactions at the document level; some offer multi-document transactions (e.g. MongoDB v > 4.x)
- Query-able by document content

Most suitable for

- Event logging
- Gaming
- Web Applications

Note: Couchbase feature: N1QL (pronounced: nickel) combines the power of SQL with the flexibility of JSON. For example,

```
SELECT country FROM `travel-sample`
WHERE name = "Excel Airways";
```

Note that for all identifiers (bucket names) that contain a hyphen character, you need to enclose the name with backtick (`) characters.

Examples: Couchbase, MongoDB, AWS DynamoDB, MarkLogic

Note: MySQL 8.0 has the Document Store.

NoSQL Graph Databases

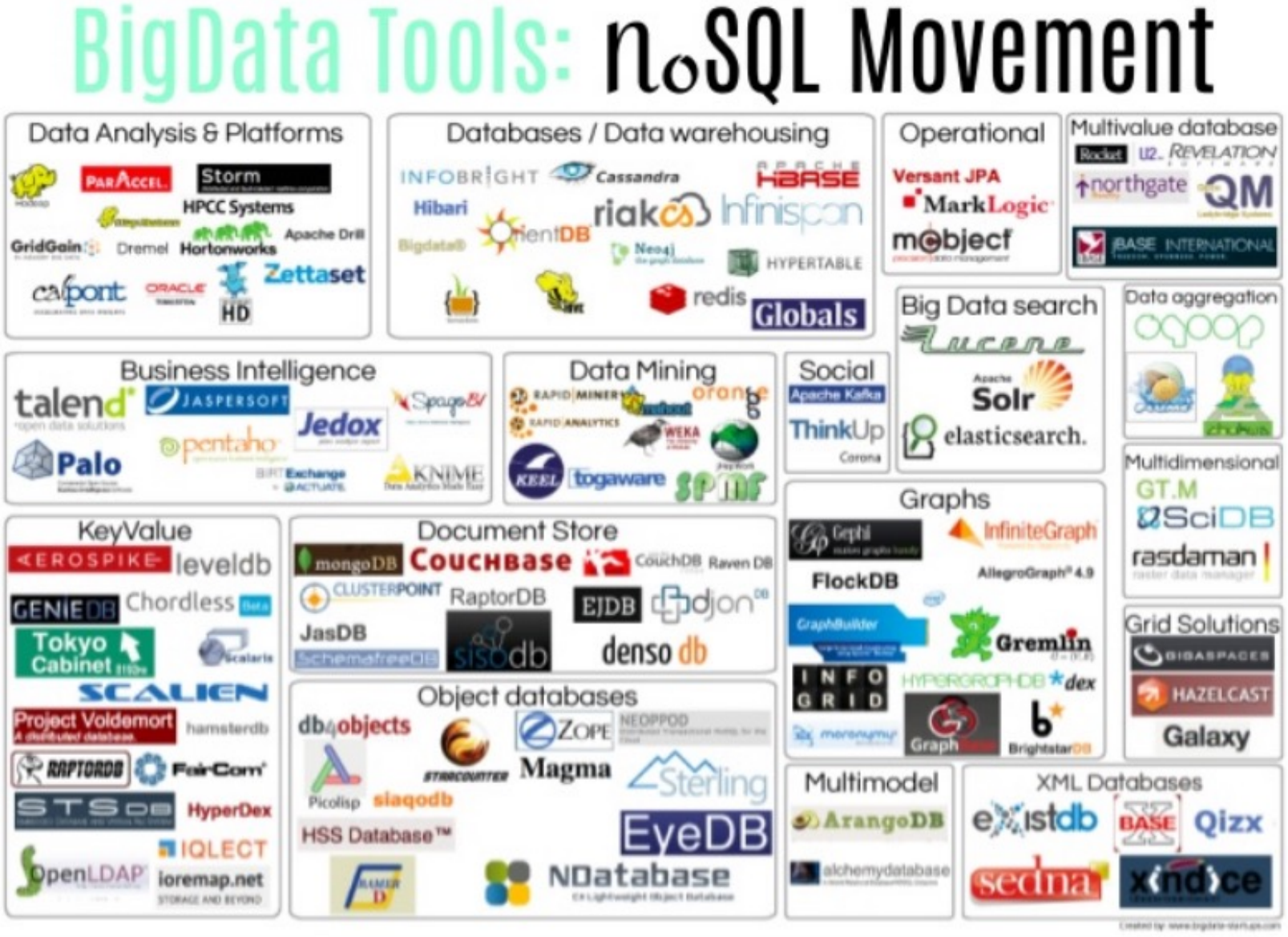
- Store entities (nodes) and relationships between them (edges)
- Node can represent for example, a user
- Optimized for fast traversal of relationships
- Eventual consistency across cluster

Most suitable for

- Event logging
- Content management systems (e.g. blogging platforms)
- Fast performance, especially when data grows deeper
- Applications requiring flexible schemas and data models

Examples: Neo4J, AWS Neptune

Too Many?



Adapted from https://www.slideshare.net/crisgomal8/nosql-databases-32933737. See also : http://nosql-database.org

Amazon DynamoDB

"Customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados. Therefore, the service responsible for managing shopping carts requires that it can always write to and read from its data store, and that its data needs to be available across multiple data centers." Amazon

Amazon's DynamoDB

- Cloud-hosted NoSQL database service
  - Dynamo: Originally built by Amazon in 2007 for shopping cart
  - DynamoDB announced in 2012 (based on Dynamo)
  - Multi-mode: Provides both **Key-Value** and **Document** model
  - Auto-Scale option
    - When enabled, the database will scale automatically

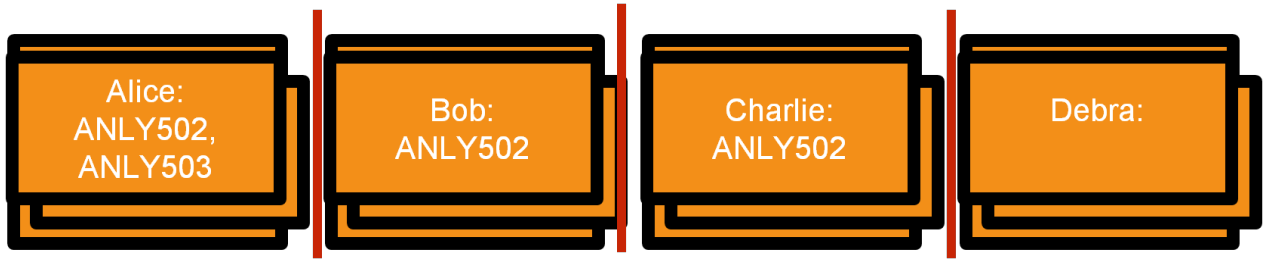
- Predictable Performance
  - You specify the number of Read & Write units you require
    - DynamoDB will spread the data and traffic over a number of servers using SSDs
  - Pay for throughput rather than storage (different from other AWS Services)
  - Performance Metrics that can be tracked using AWS Console, CLI or CloudWatch

  - AWS claims that “DynamoDB can handle more than 10 trillion requests per day and can support peaks of more than 20 million requests per second”

Example of Customers: Lyft, Airbnb, Redfin, Samsung, Toyota, and Capital One

Amazon's DynamoDB (continued)

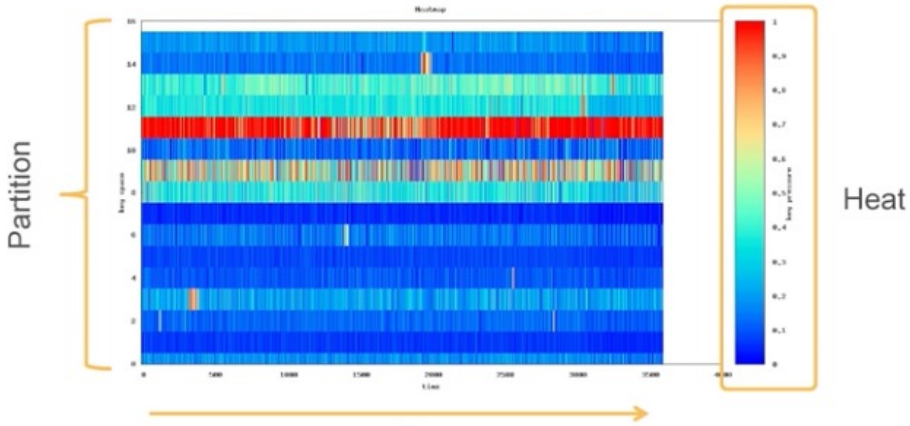
- Each document must have a "partition key"
- Database scales by adding nodes and rebalancing along partitions.
  - "Name" could be the partition key



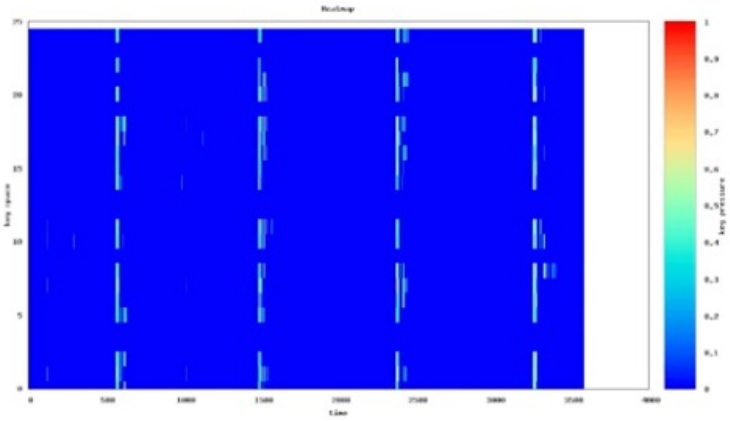
- Limitations:
  - Documents limited to 400K

AmazonDB: Choosing a Partition & Sort Key

- Partition keys
  - Used to divide work between different partitions (systems)
  - Goal — even distribution.
- Some Recommendations
  - Use high-cardinality attributes.
  - Use composite attributes. Try to combine more than one attribute to form a unique key, if that meets your access pattern.
  - Cache the popular items when there is a high volume of read traffic



Poorly partitioned



Well partitioned

Choosing a partition key for AmazonDB

Partition key value	Uniformity
User ID, where the application has many users.	Good
Status code, where there are only a few possible status codes.	Bad
Item creation date, rounded to the nearest time period (e.g. day, hour, minute) *Use Sort Key for this.*	Bad
Device ID, where each device accesses data at relatively similar intervals	Good
Device ID, where even if there are a lot of devices being tracked, one is by far more popular than all the others.	Bad

When to use NoSQL

Do you really need distributed NoSQL clusters?

Probably YES

- Unstable & High Load
- No crucial requirements for data consistency
- High Uptime
- Simple data scheme without a large number of dependencies
- No need to build reports on the fly out of the whole amount of data
- Latency is crucial and customers are widely distributed across multiple geos

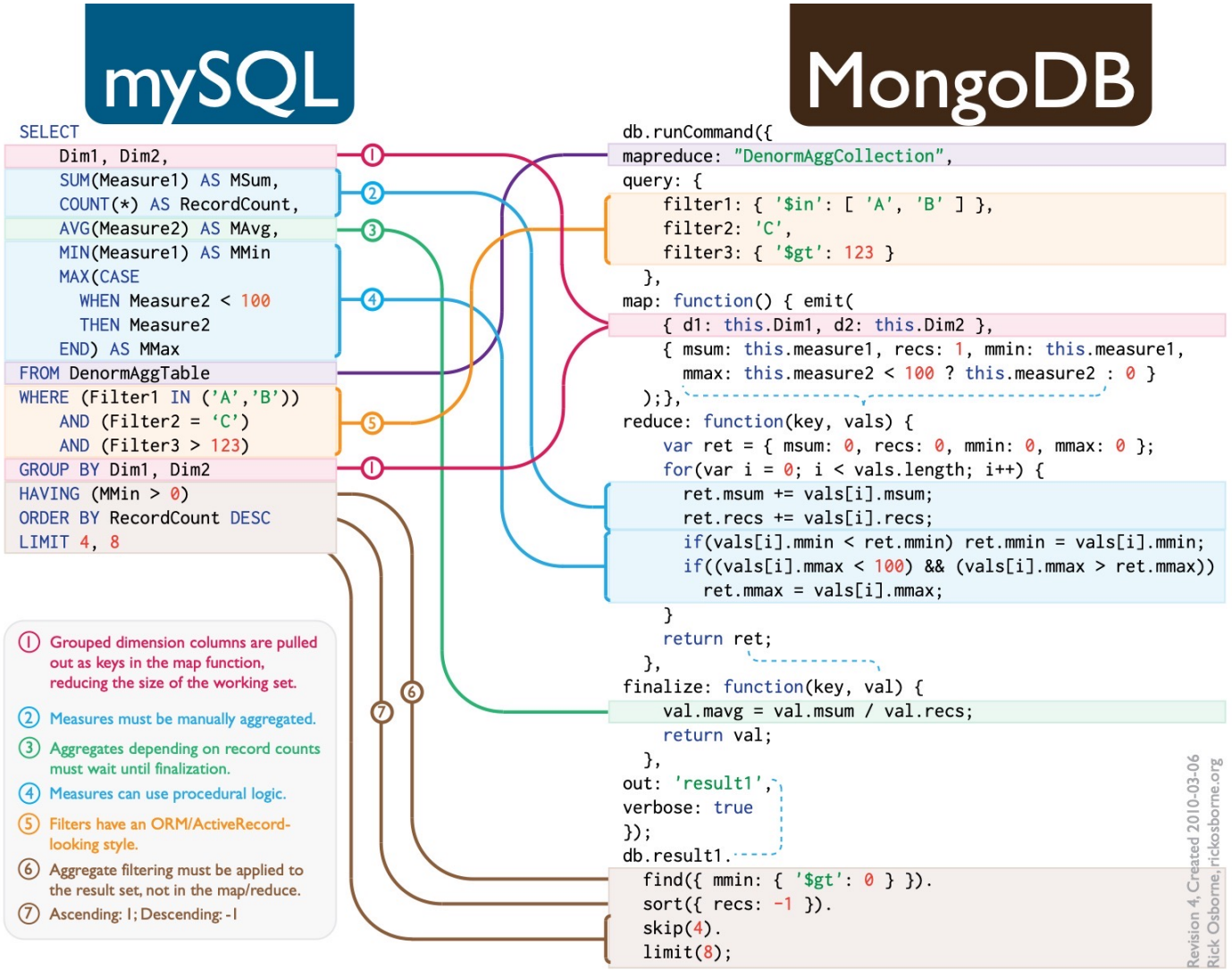
Definitely NOT

- Close to 100% data consistency is absolute must
- Stable & predicable load without significant changes for short period of time
- Well-defined and stable data schema
- Applications & clients can tolerate higher latency & some deviations there
- There is a need to build reports & queries on particular data model using full functionality of SQL

aws re:Invent

aws

SQL or NoSQL?



https://rickosborne.org/download/SQL-to-MongoDB.pdf