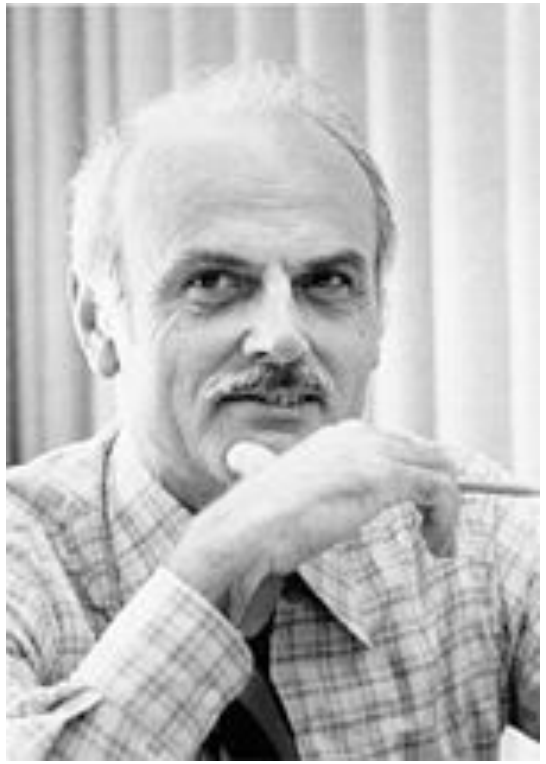
DS4300 - Large-Scale Storage and Retrieval

NoSQL Overview

John Rachlin, PhD
Northeastern University

Codd, 1970. The origins of the relational model



Edgar Frank Codd
1923-2003

Information Retrieval

A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

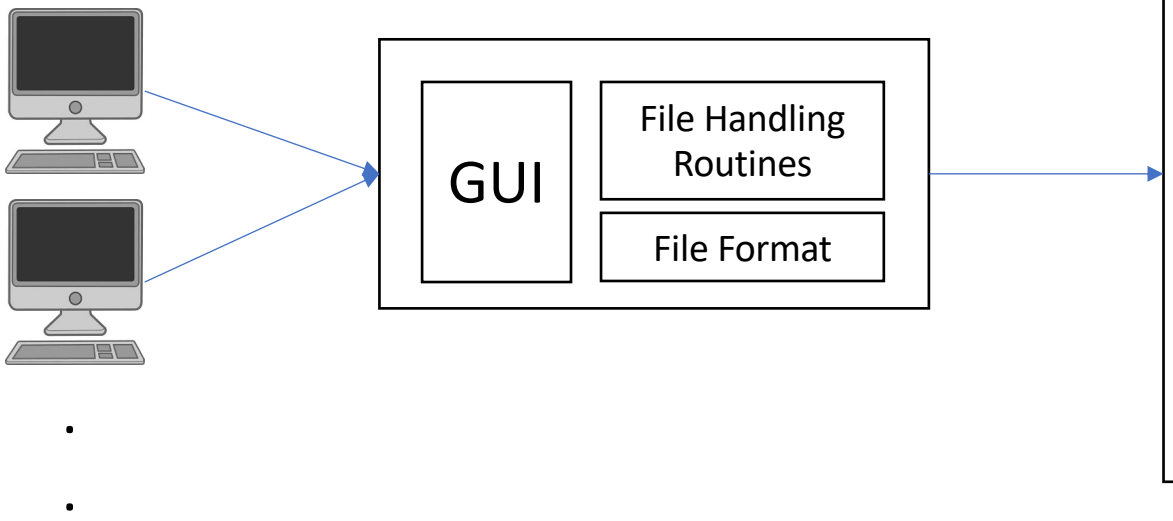
Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Before Databases: File-Based Systems

Problem: You need to build a human-resources (HR) application to manage information about your employees.

User requirements:

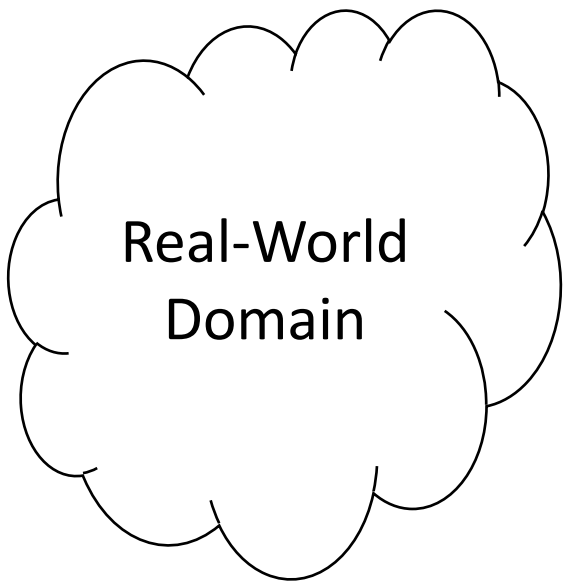
- Search ("Show me information about Joe Smith")
- Add / Delete ("Enter data about Anne Jenkins, our new hire. Delete Saniya Lee, she retired")
- Update ("James Obatola was promoted, update his title.")

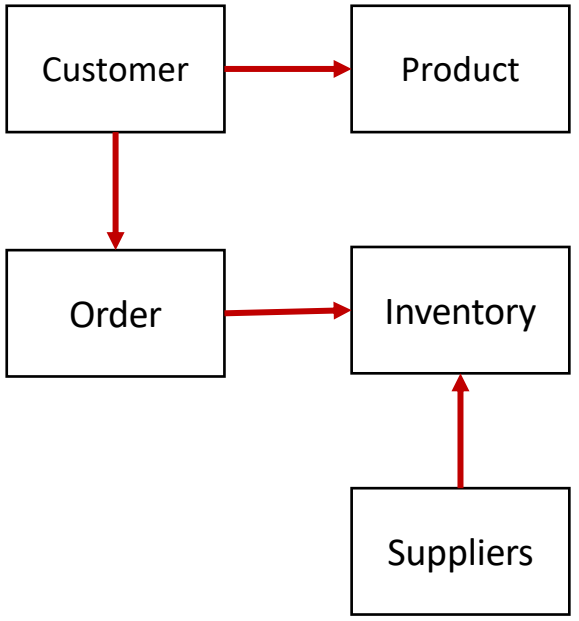


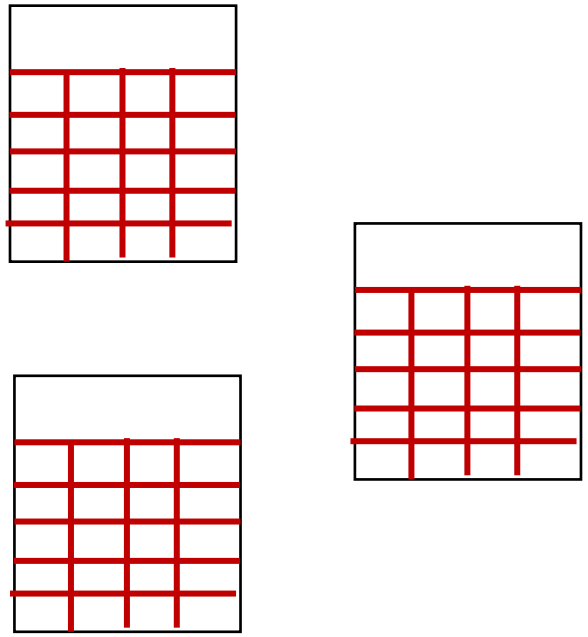

```
LAST,FIRST,DOB,TITLE
Smith,Joe,4/5/1992,Manager
Jenkins,Anne,10/08/2003,Programmer
Obtola,James,12/32/1982,HR
Mike,Crawford,2/29/2002,H.R.
Smith  ,Joe, 7/23/1966,CEO
.
.
```

What is a data model?

A **data model** is a description of the world, a domain, or an enterprise. The model determines how we interact with the data in a database both in terms of the operations we can perform, and the questions we can ask.







- **Entities** (things, events, concepts)
- **Attributes** (annotations on entities)
- **Relationships** (connections or associations between entities)

Entity-Relationship Diagram
(A *conceptual* data model)

Relational Data Model
(A *logical* data model)

Relational Data Model

Entities

Branch

branchNo	street	city	postCode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

A column or ***attribute*** or **field** providing more details about the particular entity.

- Name
- Datatype
- Other attributes

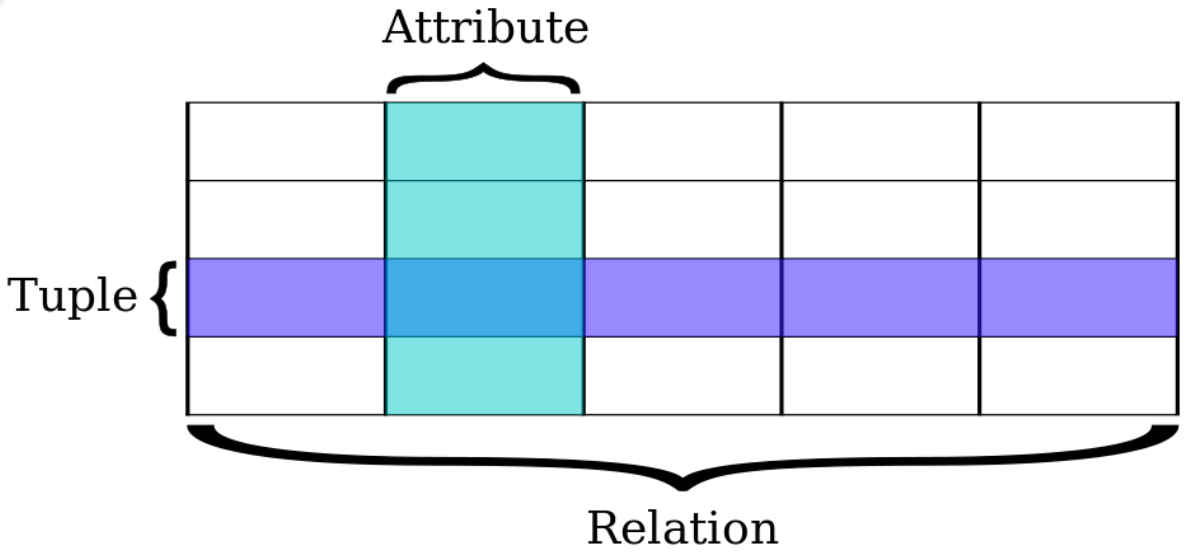
A row or ***record*** describing one instance of the entity

Three pillars of Relational Databases

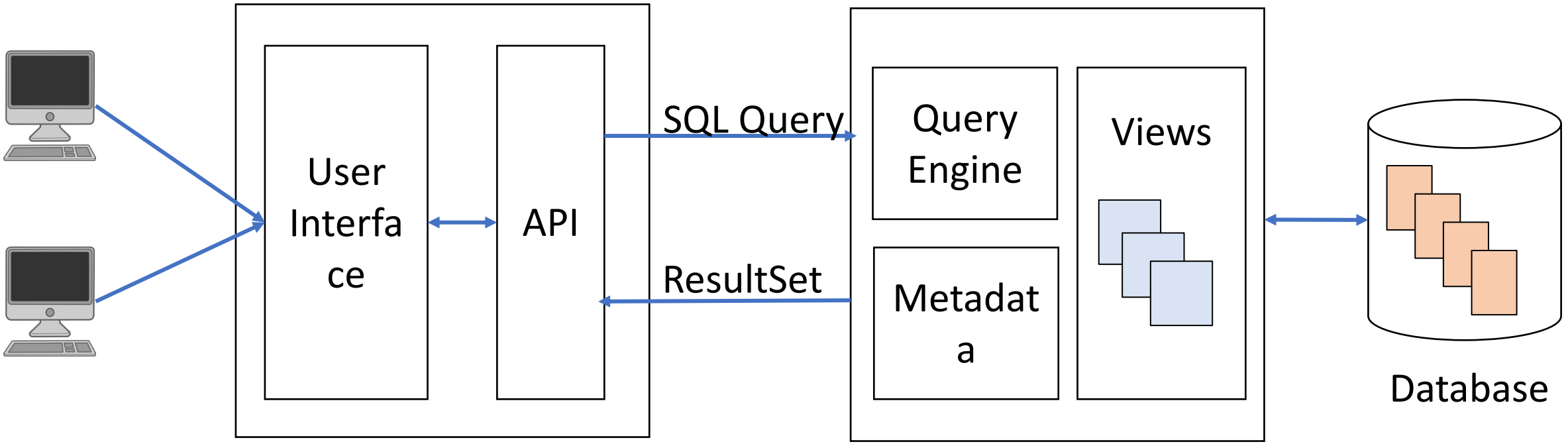
relational model

SQL

ACID



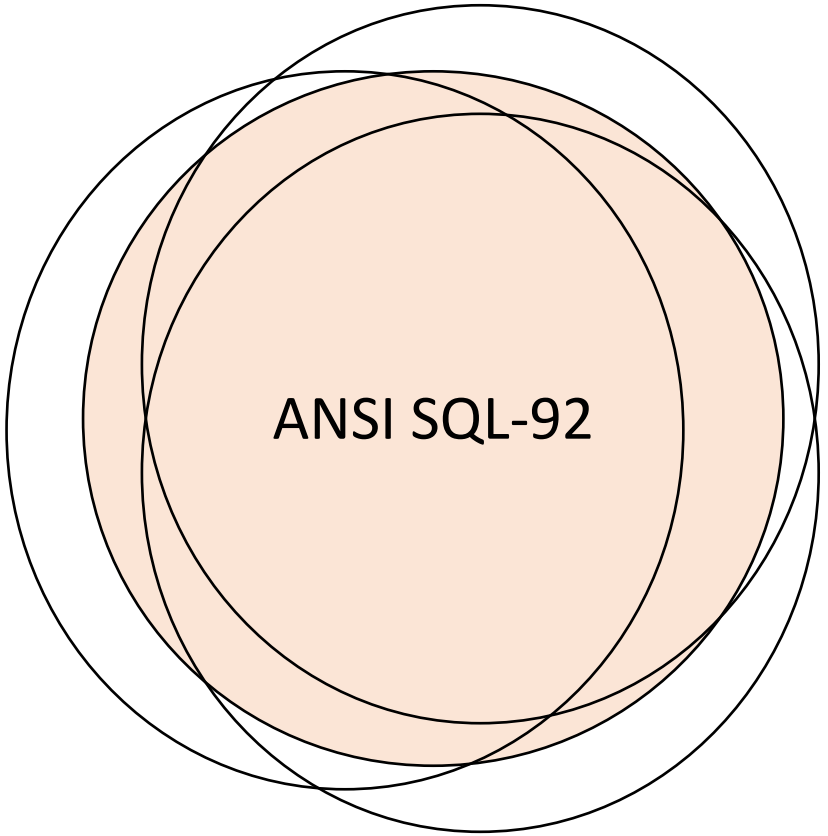
Database Application



(Usually running on a separate server)

SQL: The language of databases

- SQL = Structured Query Language
- Pronounced *either* S.Q.L. or *See*-quel (both are perfectly acceptable.)
- The language was *standardized* over time (SQL-86, SQL-89, SQL-92) but most vendors do not implement the standard completely or they introduce vendor-specific features / dialects.



One standard, many dialects

The dominance of the relational model today

- Overcoming the limitations of file-based systems was a major technical achievement!

- Key features:
  - ✓ Data persistence
  - ✓ Concurrency support: multiple users simultaneously accessing data
  - ✓ Transaction support
  - ✓ Application integration

- A standard query language around which engines could be developed and optimized (though there are many dialects).

- Relational databases transformed how data is modeled, accessed, and transformed and have been **the dominant database model** since their development in the 1970's

Why learn about databases?

Answer: Because databases are *ubiquitous*!

u·biq·ui·tous

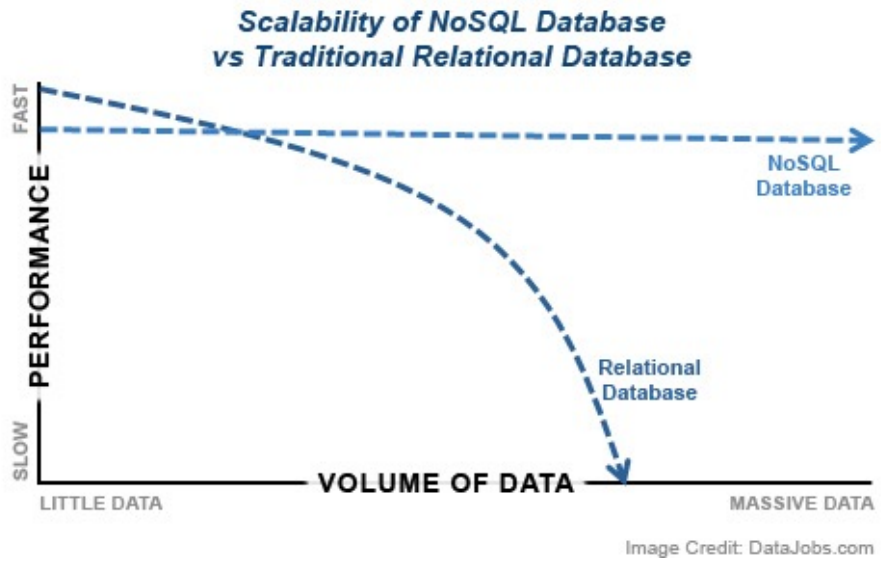
/yo͞o'bikwədəs/

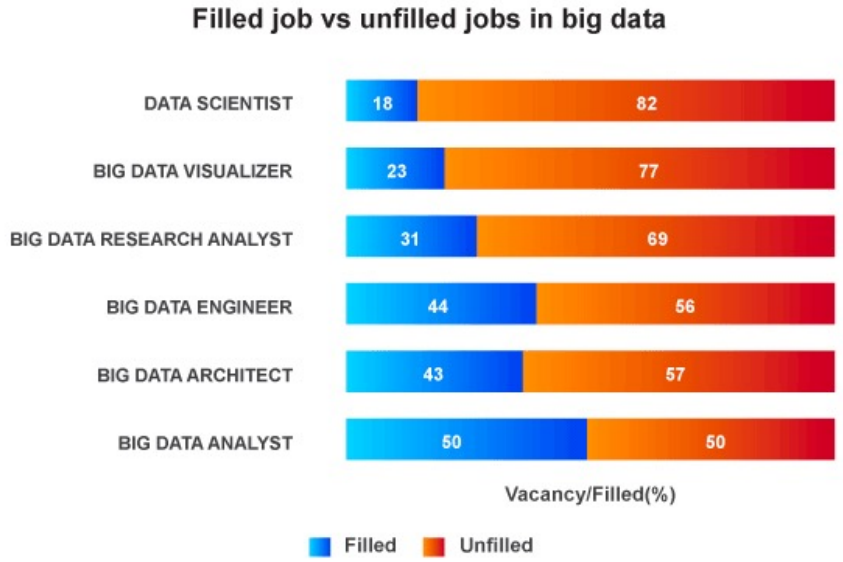
adjective
adjective: **ubiquitous**

present, appearing, or found everywhere.
"his ubiquitous influence was felt by all the family"
synonyms: omnipresent, ever-present, everywhere, all over the place, pervasive, universal, worldwide, global; More
antonyms: rare

Why learn about NoSQL / Non-relational models?

- Traditional relational databases don't always scale well
- Stringent relational schema not ideal for unstructured data
- Expanding the breadth of your knowledge in data management
- $$$

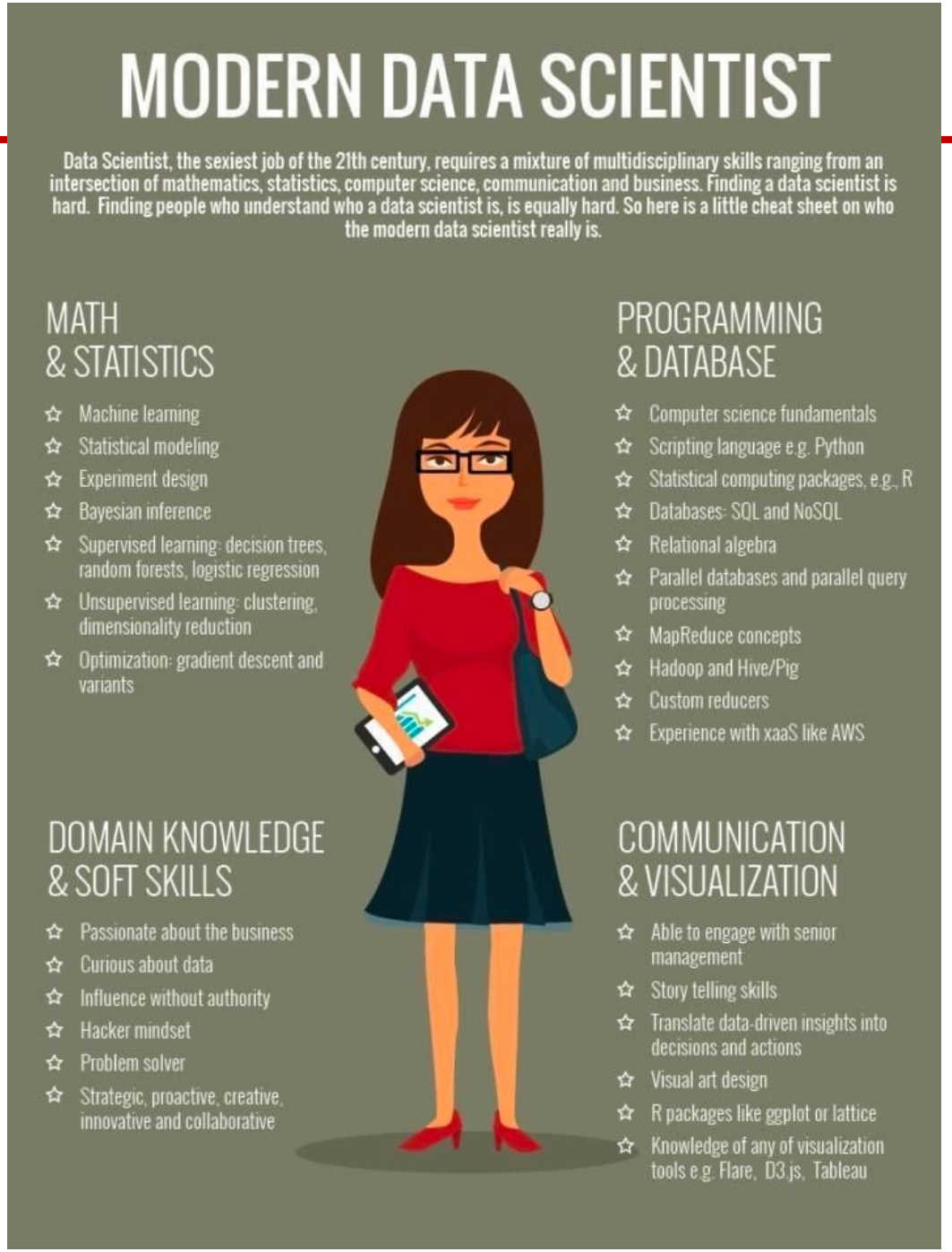




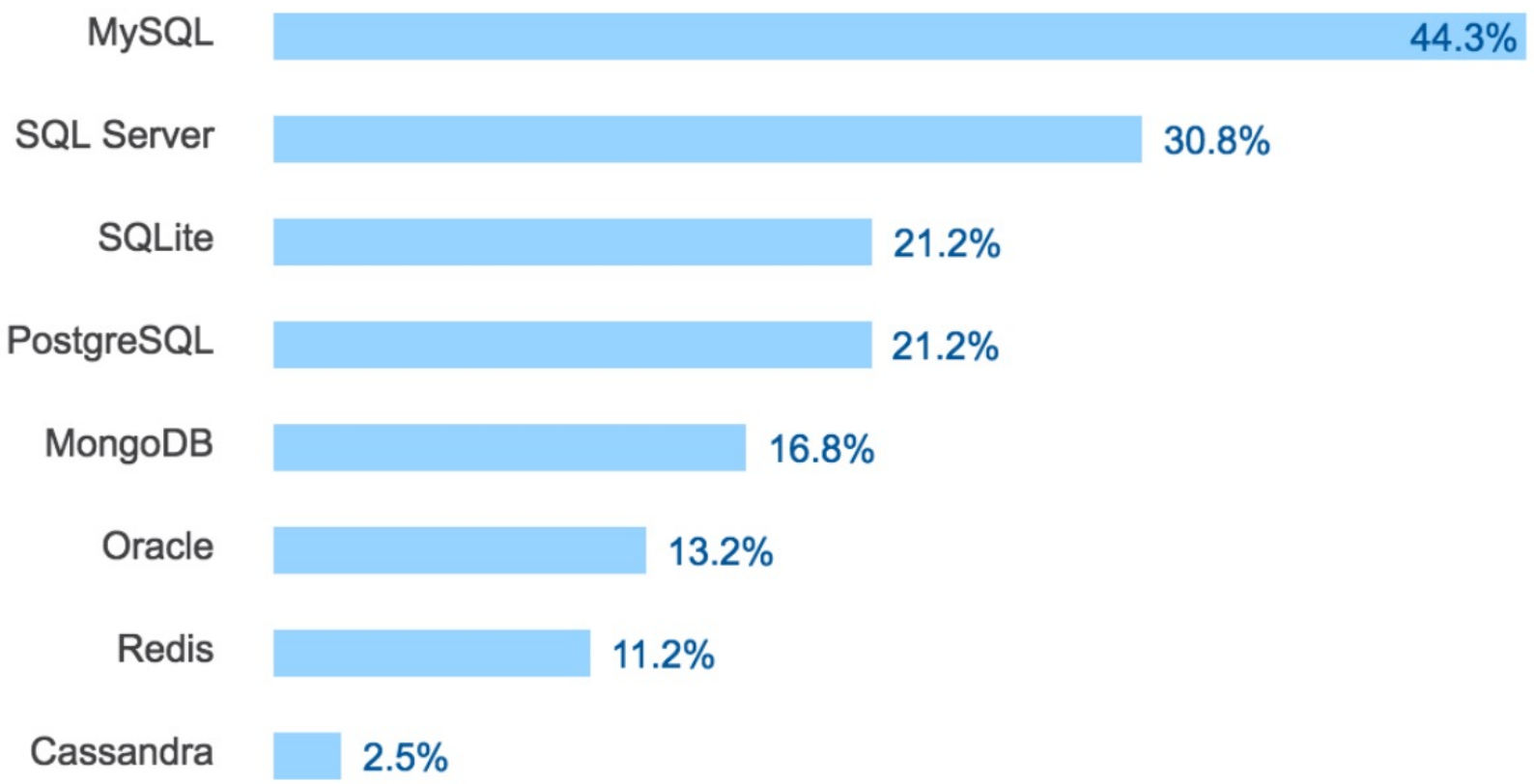
Data Science Careers

Among the various industries that employ NoSQL specialists, the Healthcare Services industry pays the best, with a whopping average salary of $135,948, followed by the Software Development industry, offering an average salary of $120,043.

Source: https://www.simplilearn.com/why-nosql-skills-are-crucial-for-big-data-career-article (Dec, 2016)



StackOverflow survey of most popular databases

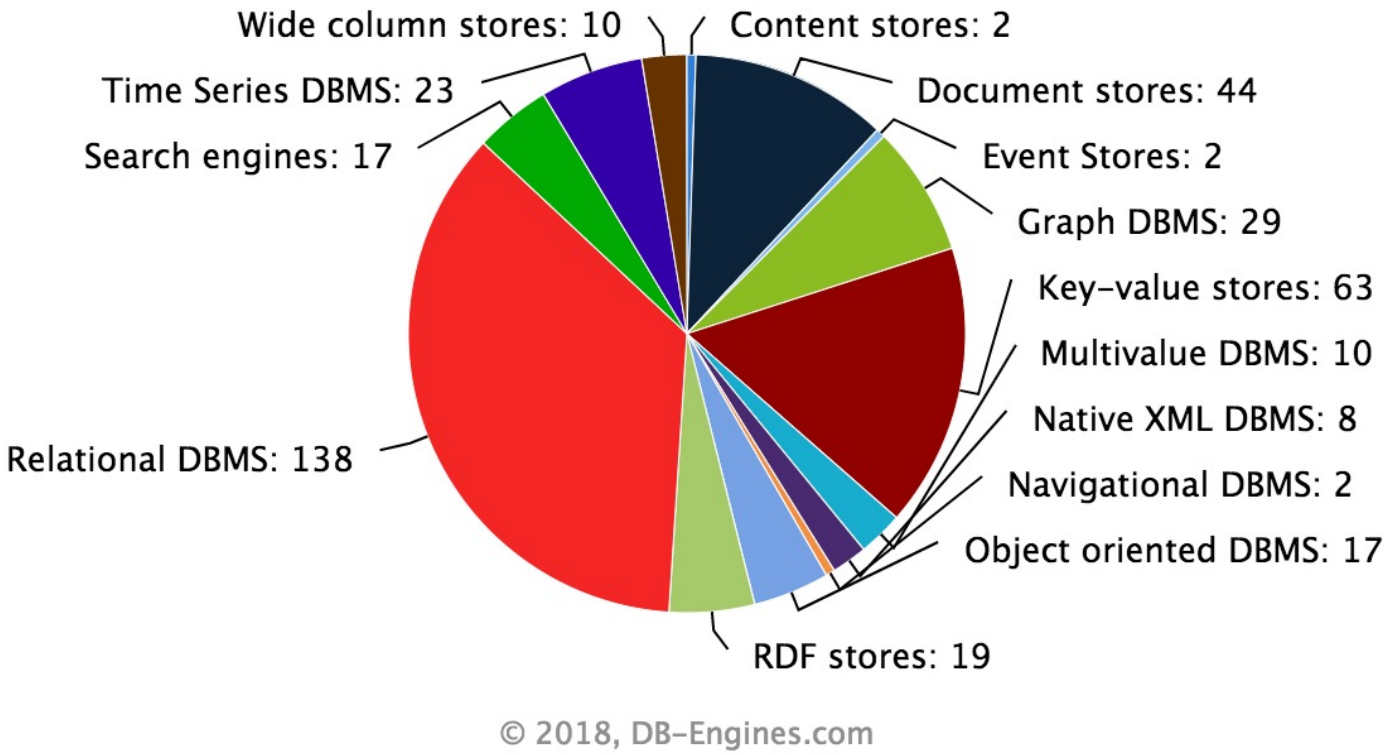


N=36,935
Respondents

https://www.eversql.com/most-popular-databases-in-2017 according-to-stackoverflow-survey/

DBMS Popularity

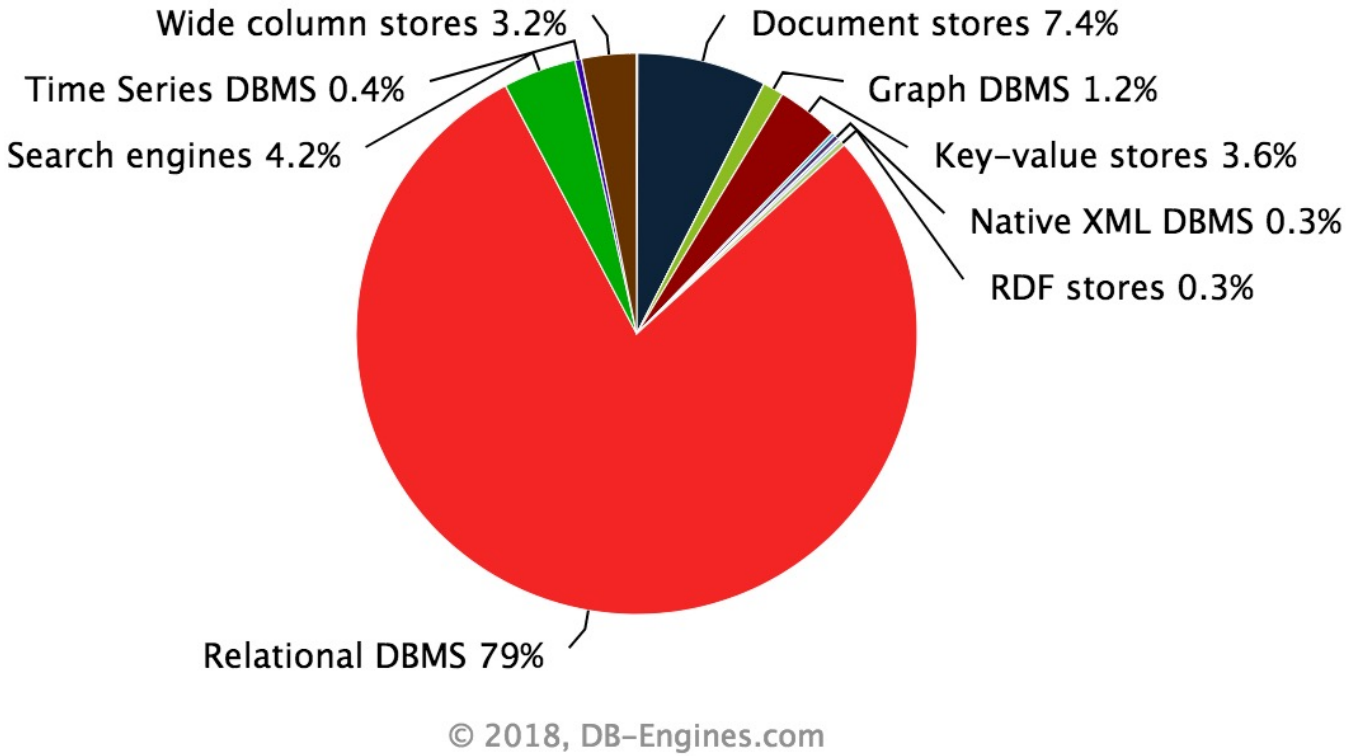
Number of systems per category, January 2018



DB-Engines lists 341 different database management systems, which are classified according to their database model (e.g. relational DBMS, key-value stores etc.).
This pie-chart shows the number of systems in each category. Some of the systems belong to more than one category.

DBMS Popularity

Ranking scores per category in percent, January 2018



This chart shows the popularity of each category. It is calculated with the popularity (i.e. the ranking scores) of all individual systems per category. The sum of all ranking scores is 100%.

DB-Engines Ranking

337 systems in ranking, November 2017

Rank Nov 2017	Rank Oct 2017	Rank Nov 2016	DBMS	Database Model	Score Nov 2017	Score Oct 2017	Score Nov 2016
1.	1.	1.	Oracle	Relational DBMS	1360.05	+11.25	-52.96
2.	2.	2.	MySQL	Relational DBMS	1322.03	+23.20	-51.53
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1215.08	+4.76	+1.27
4.	4.	4.	PostgreSQL	Relational DBMS	379.92	+6.64	+54.10
5.	5.	5.	MongoDB	Document store	330.47	+1.07	+5.00
6.	6.	6.	DB2	Relational DBMS	194.06	-0.53	+12.61
7.	7.	↑ 8.	Microsoft Access	Relational DBMS	133.31	+3.86	+7.34
8.	8.	↓ 7.	Cassandra	Wide column store	124.21	-0.58	-9.76
9.	9.	9.	Redis	Key-value store	121.18	-0.87	+5.64
10.	10.	↑ 11.	Elasticsearch	Search engine	119.41	-0.82	+16.84

https://db-engines.com/en/ranking

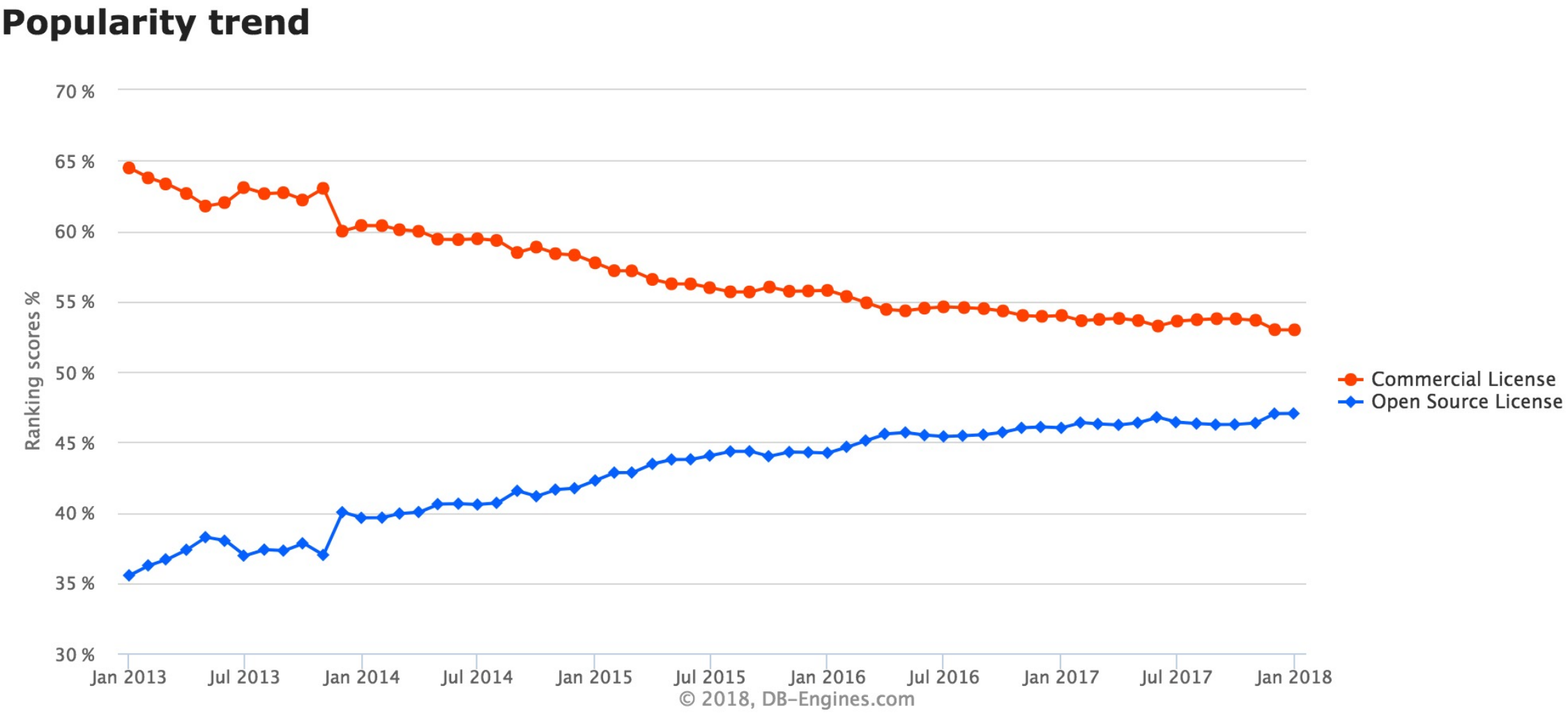
DB-Engines Ranking

371 systems in ranking, May 2021

Rank May 2021	Rank Apr 2021	Rank May 2020	DBMS	Database Model	Score May 2021	Score Apr 2021	Score May 2020
1.	1.	1.	Oracle	Relational, Multi-model	1269.94	-4.98	-75.50
2.	2.	2.	MySQL	Relational, Multi-model	1236.38	+15.69	-46.26
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	992.66	-15.30	-85.64
4.	4.	4.	PostgreSQL	Relational, Multi-model	559.25	+5.73	+44.45
5.	5.	5.	MongoDB	Document, Multi-model	481.01	+11.04	+42.02
6.	6.	6.	IBM Db2	Relational, Multi-model	166.66	+8.88	+4.02
7.	7.	↑ 8.	Redis	Key-value, Multi-model	162.17	+6.28	+18.69
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model	155.35	+3.18	+6.23
9.	9.	9.	SQLite	Relational	126.69	+1.64	+3.66
10.	10.	10.	Microsoft Access	Relational	115.40	-1.33	-4.50

https://db-engines.com/en/ranking

The growing importance of open-source databases



The above chart shows the historical trend of the popularity of open source and commercial database management systems.

Most popular databases, January 2018

Commercial

Rank	System	Score	Overall Rank
1.	Oracle	1342	1.
2.	Microsoft SQL Server	1148	3.
3.	DB2	190	6.
4.	Microsoft Access	127	7.
5.	Teradata	73	12.

Open-Source

Rank	System	Score	Overall Rank
1.	MySQL	1300	2.
2.	PostgreSQL	386	4.
3.	MongoDB	331	5.
4.	Cassandra	124	8.
5.	Redis	123	9.

The NoSQL Landscape: Maturity vs. Features

Figure 1. Magic Quadrant for Operational Database Management Systems

Source: Gartner (October 2014)

What is Big Data?

- **Big Data** is *generally* concerned with the acquisition, storage, and analysis of extremely large volumes of data, typically many Terabytes or more.
- What is consider **big data** today may not be tomorrow. For example, in the 1980's, 10-100 GB might have been considered "big data" with disk drives typically in the 20-100 MB range.
- Circa 2018: 100-1000 TB or more would likely be considered big data by most companies (but not all!)

Prefixes for multiples of bits (bit) or bytes (B)

Decimal			Binary			
Value	SI		Value	IEC		JEDEC
1000	k	kilo	1024	Ki	kibi	K kilo
1000^2	M	mega	1024^2	Mi	mebi	M mega
1000^3	G	giga	1024^3	Gi	gibi	G giga
1000^4	T	tera	1024^4	Ti	tebi	–
1000^5	P	peta	1024^5	Pi	pebi	–
1000^6	E	exa	1024^6	Ei	exbi	–
1000^7	Z	zetta	1024^7	Zi	zebi	–
1000^8	Y	yotta	1024^8	Yi	yobi	–

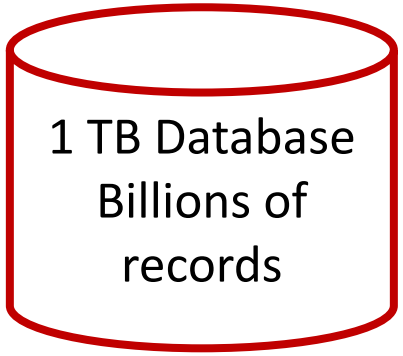
V • T • E

Time is important too!

	N	10^N Sec	Min	Hrs	Days	Years	Centuries
	-	1	0.0	0.0	0.0	0.0	0.0
	1	10	0.2	0.0	0.0	0.0	0.0
	2	100	1.7	0.0	0.0	0.0	0.0
Thousand / Kilo	3	1,000	16.7	0.3	0.0	0.0	0.0
	4	10,000	166.7	2.8	0.1	0.0	0.0
	5	100,000	1,667	27.8	1.2	0.0	0.0
Million / Mega	6	1,000,000	16,667	278	11.6	0.0	0.0
	7	10,000,000	166,667	2,778	116	0.3	0.0
	8	100,000,000	1,666,667	27,778	1,157	3.2	0.0
Billion / Giga	9	1,000,000,000	16,666,667	277,778	11,574	32	0.3
	10	10,000,000,000	166,666,667	2,777,778	115,741	317	3.2
	11	100,000,000,000	1,666,666,667	27,777,778	1,157,407	3,169	31.7
Trillion / Tera	12	1,000,000,000,000	16,666,666,667	277,777,778	11,574,074	31,688	317
	13	1.E+13	2.E+11	3.E+09	1.E+08	3.E+05	3.E+03
	14	1.E+14	2.E+12	3.E+10	1.E+09	3.E+06	3.E+04
Quadrillion / Peta	15	1.E+15	2.E+13	3.E+11	1.E+10	3.E+07	3.E+05
	16	1.E+16	2.E+14	3.E+12	1.E+11	3.E+08	3.E+06
	17	1.E+17	2.E+15	3.E+13	1.E+12	3.E+09	3.E+07
Quintillian / Exa	18	1.E+18	2.E+16	3.E+14	1.E+13	3.E+10	3.E+08
	19	1.E+19	2.E+17	3.E+15	1.E+14	3.E+11	3.E+09
	20	1.E+20	2.E+18	3.E+16	1.E+15	3.E+12	3.E+10

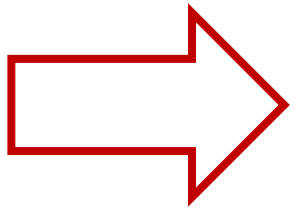
Volume, Velocity, Variety

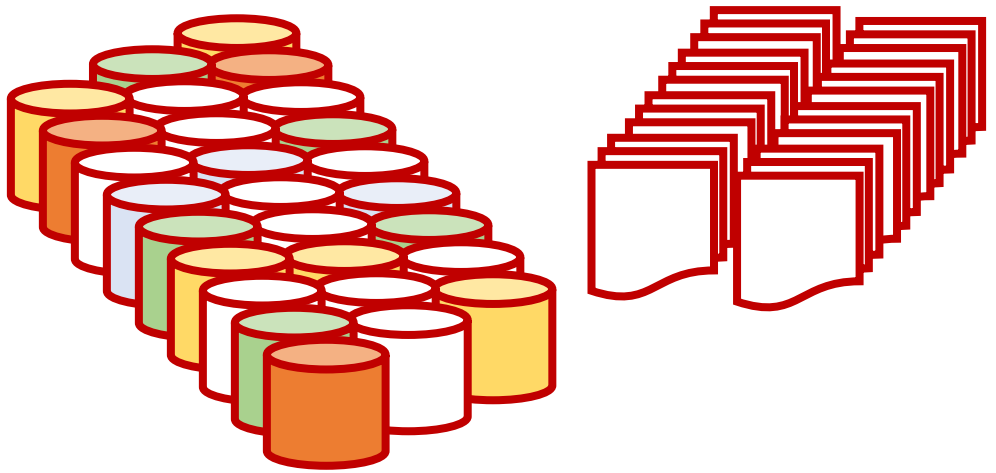
- **Volume**: How much data do you have?
- **Velocity**: How quickly is your data growing?
- **Variety**: How diverse or complex is your data?
- **Veracity**: How trustworthy is your data?



- Manageable volumes
- Limited growth
- Fixed schema, highly structured

Traditional file systems and relational databases





- Massive volumes
- Rapid growth with *streaming* data
- Both structured and un-structured or semi-structured datasets (tables, documents log files, images, videos)

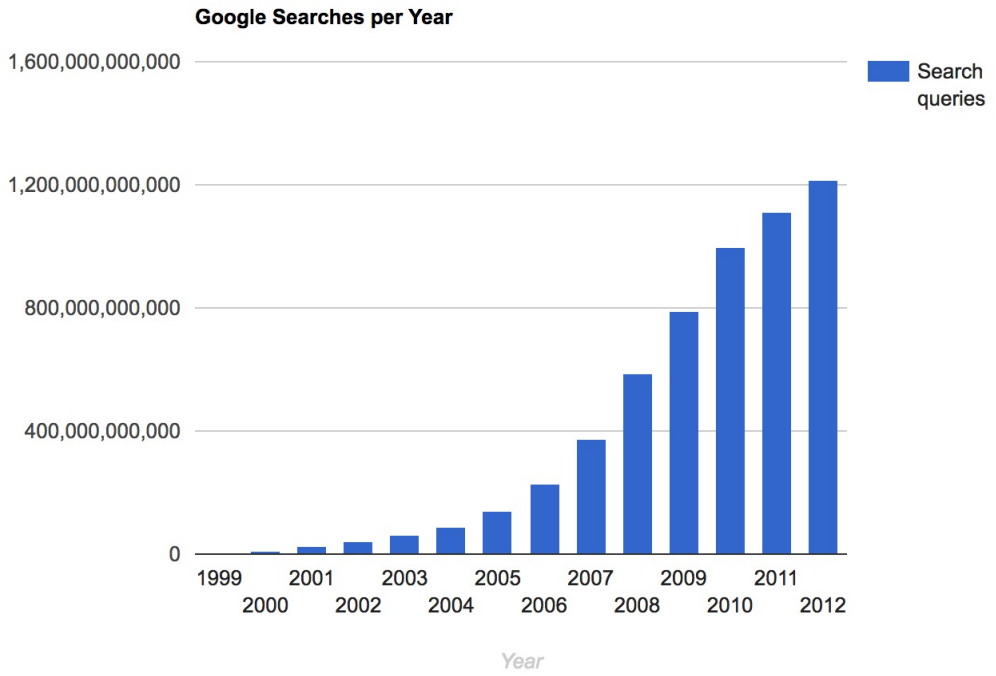
Big Data Solutions

Sources of Big Data

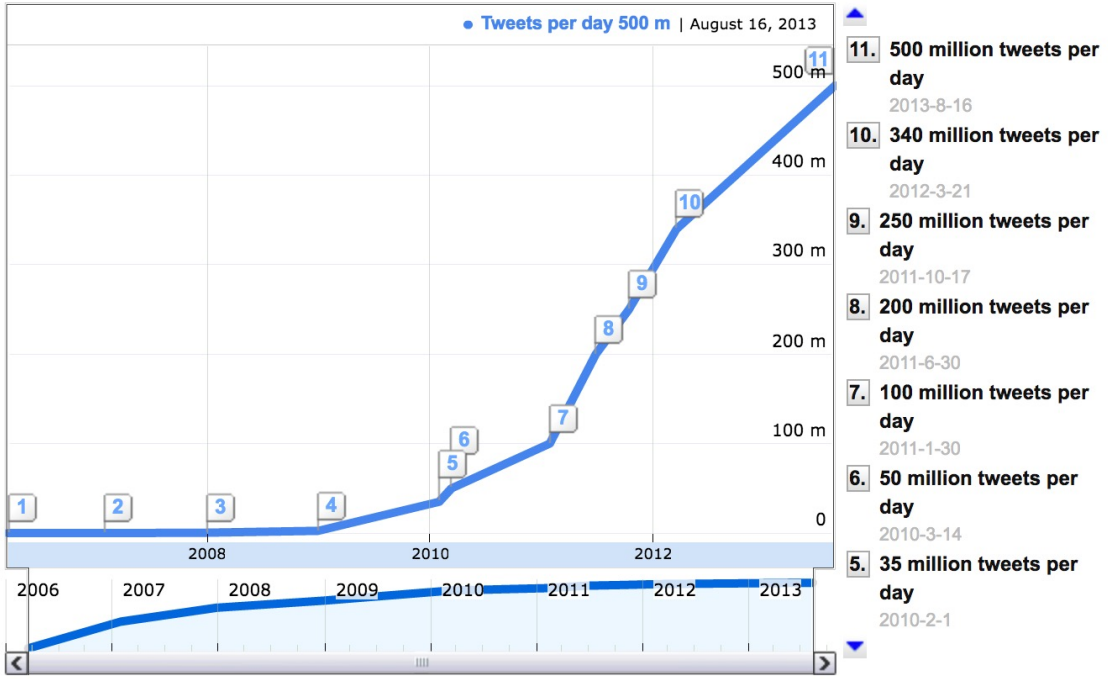
- Activity data (online transactions, credit card data, stock trading)
- Sensor networks
- On-line gaming systems supporting millions of users
- Biological databases, including genomic data
- Scientific instruments (LHC, LSST)
- Social Networking (twitter, Facebook, LinkedIn, text messaging)

Searches and Tweets

Google now processes over 40,000 search queries every second on average (visualize them here), which translates to over **3.5 billion searches per day** and **1.2 trillion searches per year** worldwide. The chart below shows the number of searches per year throughout Google's history:



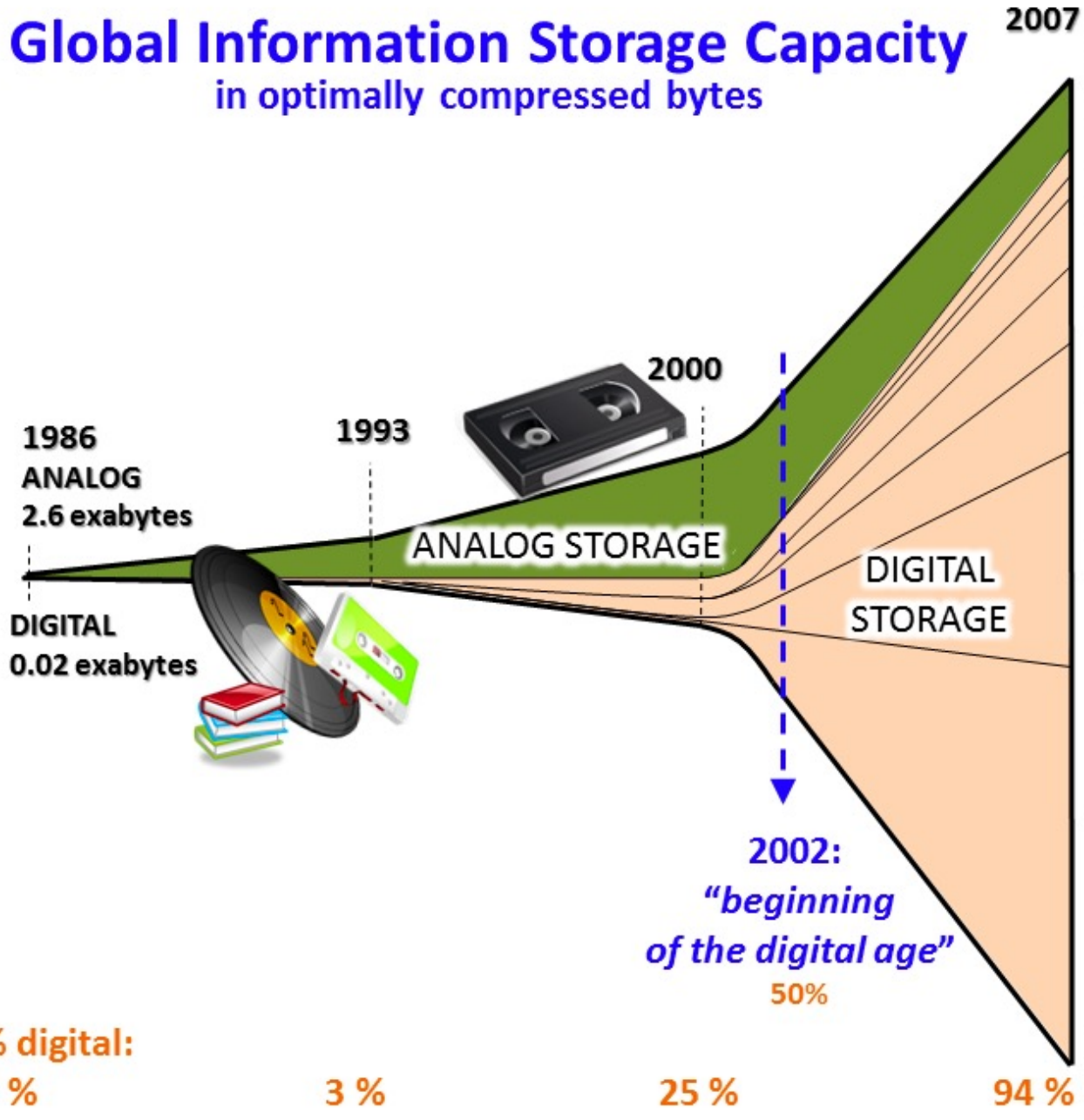
Every second, on average, around 6,000 tweets are tweeted on Twitter (visualize them here), which corresponds to over 350,000 tweets sent per minute, **500 million tweets per day** and around 200 billion tweets per year. The chart below shows the number of tweets per day throughout Twitter's history:



Source: http://www.internetlivestats.com/

Global Information Storage Capacity

in optimally compressed bytes



2007 **ANALOG**
19 exabytes

- Paper, film, audiotape and vinyl: 6 %
- Analog videotapes (VHS, etc): 94 % **ANALOG**

DIGITAL

- Portable media, flash drives: 2 %
- Portable hard disks: 2.4 %
- CDs and minidisks: 6.8 %
- Computer servers and mainframes: 8.9 %
- Digital tape: 11.8 %
- DVD/Blu-ray: 22.8 %

- PC hard disks: 44.5 %
  123 billion gigabytes

- Others: < 1 % (incl. chip cards, memory cards, floppy disks, mobile phones, PDAs, cameras/camcorders, video games)

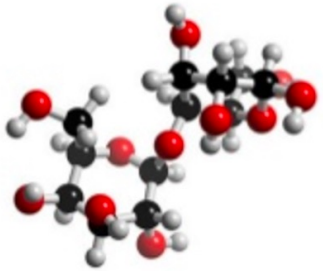
DIGITAL
280 exabytes

Source: Hilbert, M., & López, P. (2011). The World's Technological Capacity to Store, Communicate, and Compute Information. *Science*, 332(6025), 60 –65. http://www.martinhilbert.net/WorldInfoCapacity.html

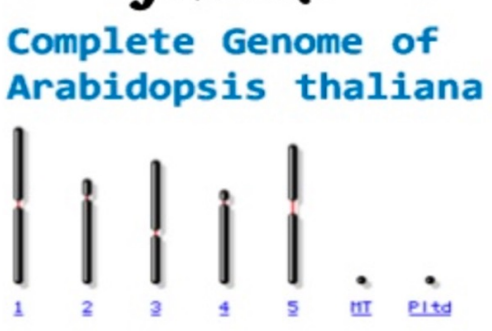
Bioinformatics Databases

Sequences

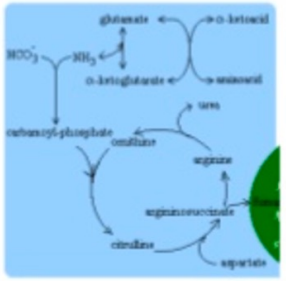
Structure



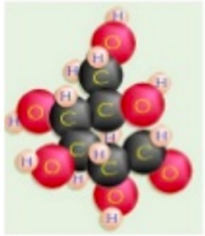
Genome



Pathways



Lipids, Carbohydrates



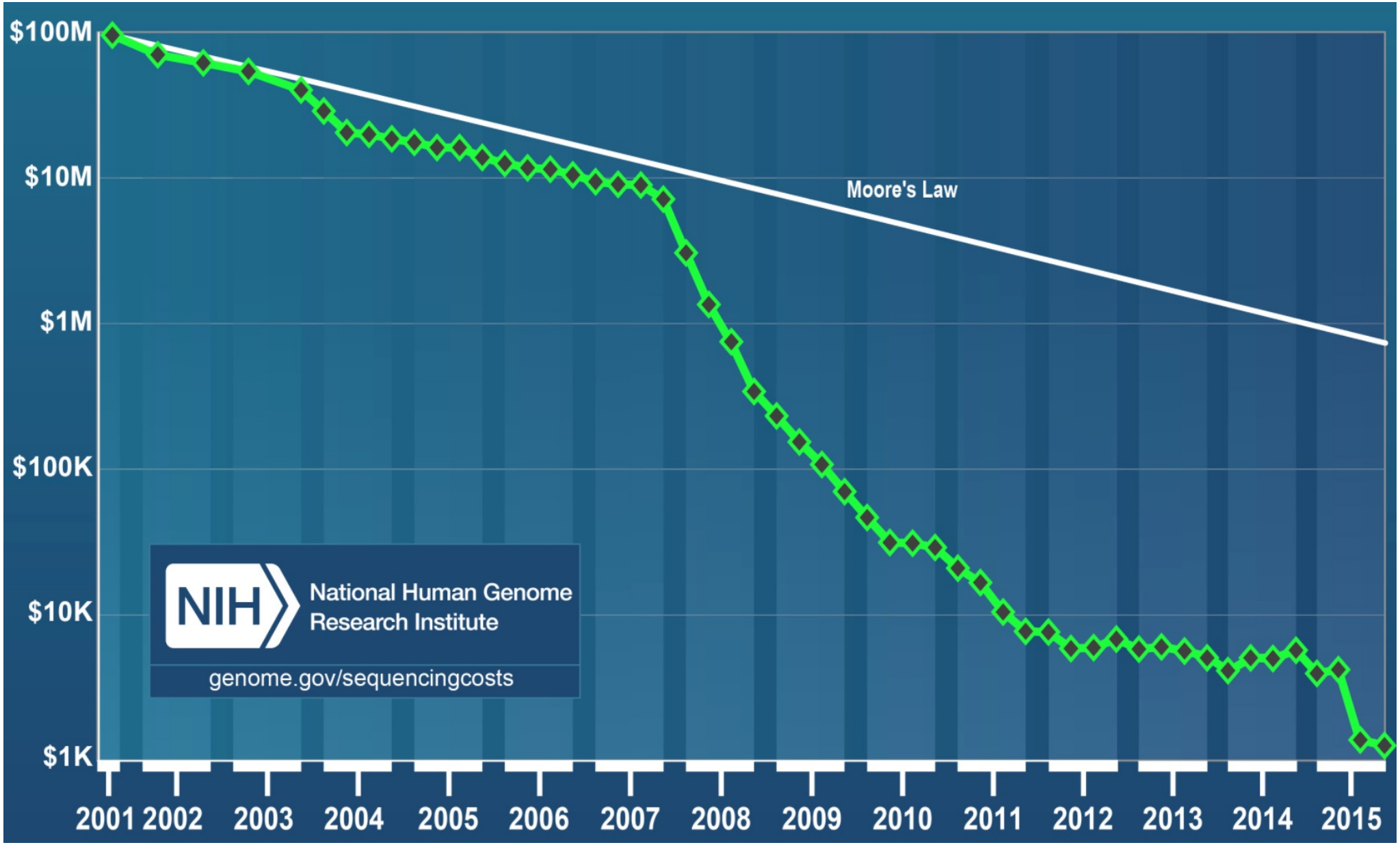
Literature

Primary Database: Direct experimental results

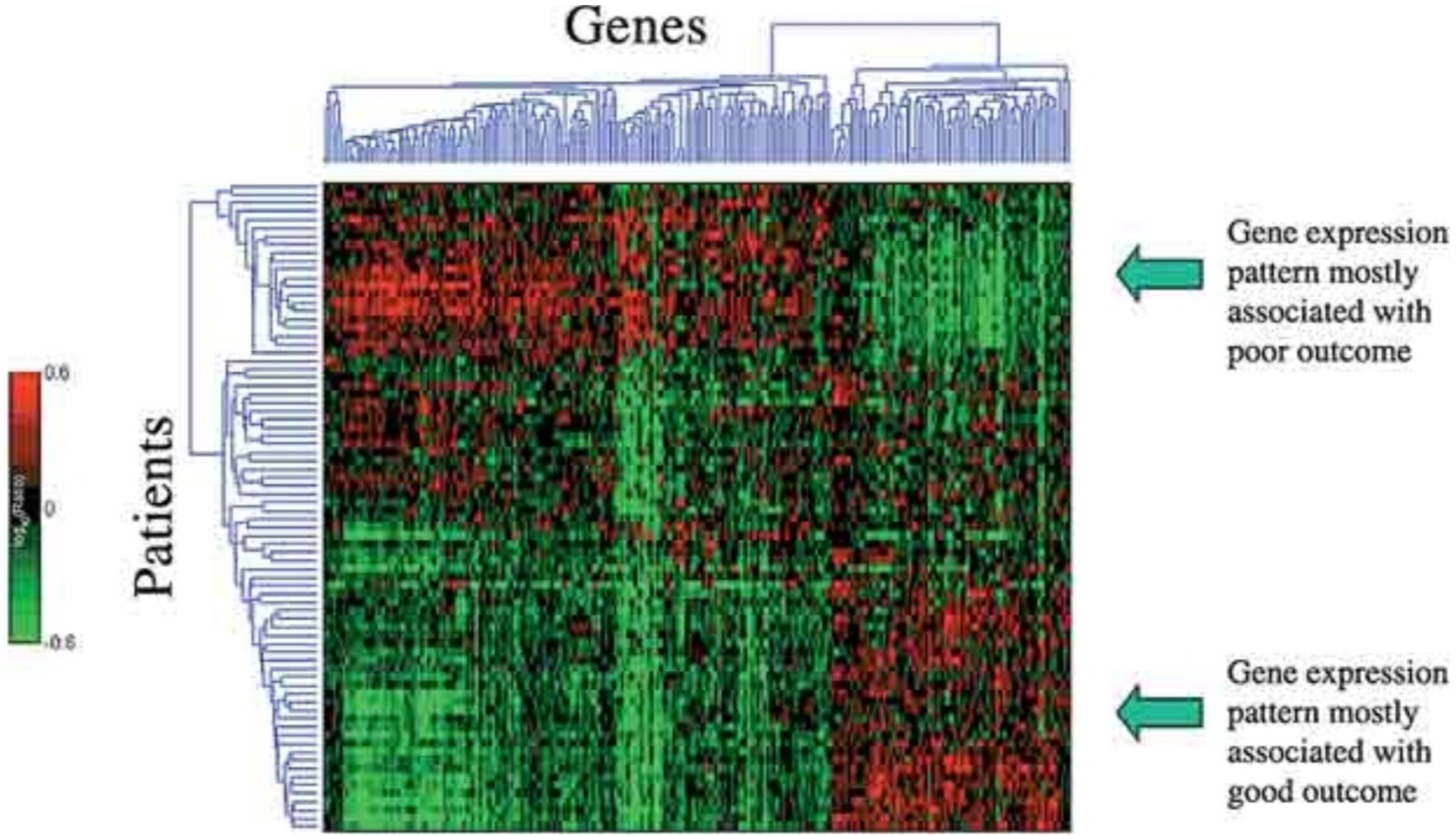
Secondary Database: Derived databases from transformation & analysis

Source: https://www.slideshare.net/shwetakagliwal/biological-databases-11267007

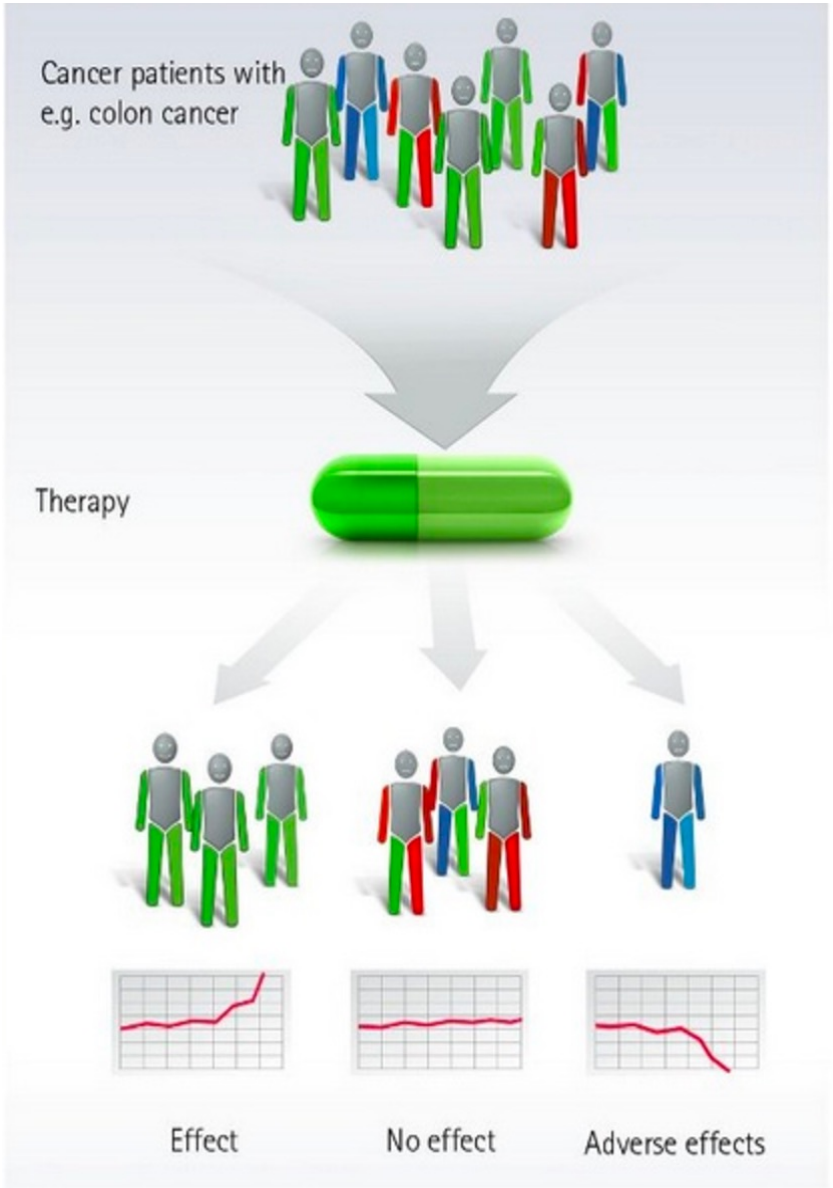
Cost to sequence a single human genome.

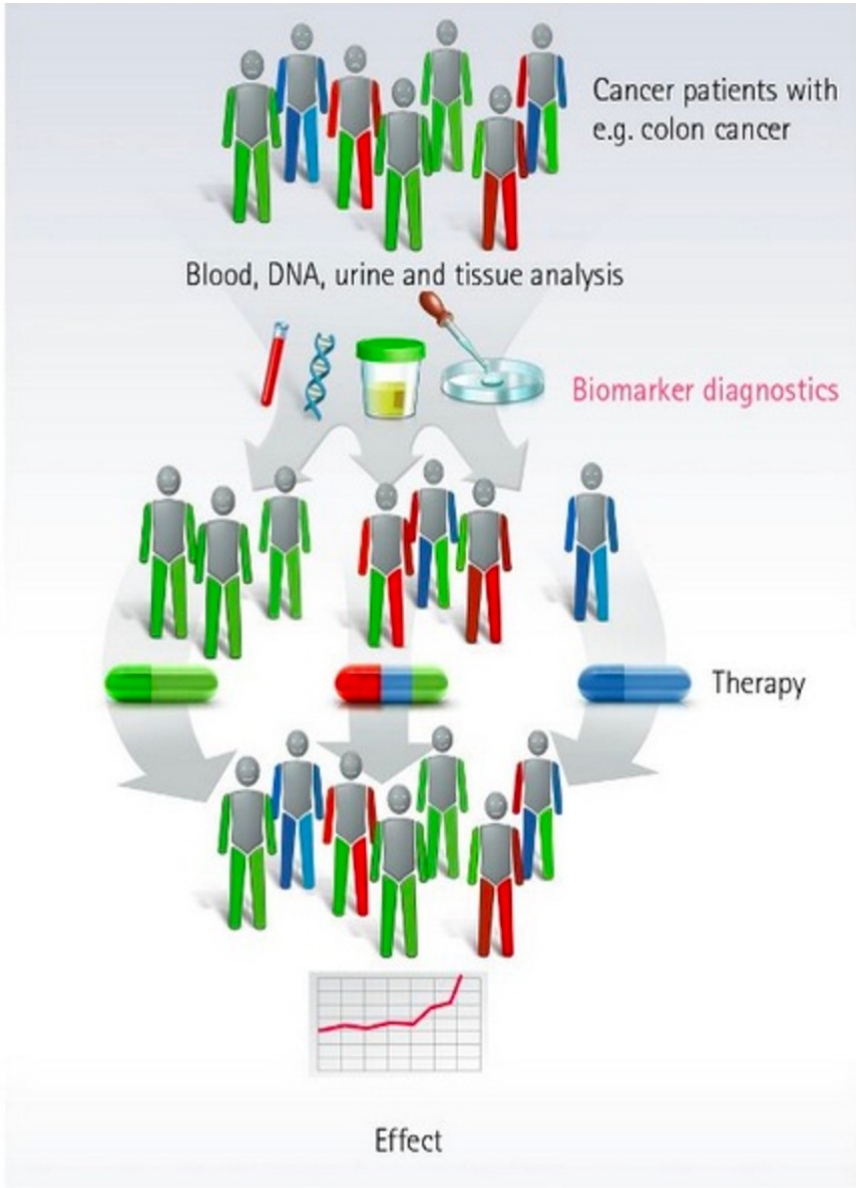


DNA Microarrays



The coming age of personalized medicine





The promise of personalized medicine:

- Improved efficacy
- Reduced adverse side-effects
- Reduced “trial-and-error” delays in the treatment of time-critical diseases such as cancer.

EMR: Electronic Medical Records

- Patient Demographics
- Doctors
- Insurance
- Medical history
- Allergies
- Procedures / Measurements (BP, Temp, O_2)
- Order Lab Tests ➡️ Lab Results
- Dr. Notes

Next time you get a physical, notice how much time your doctor spends at a computer terminal!

Optum Analytics: Aggregator of 90+ million patient records

Staging
(raw data)

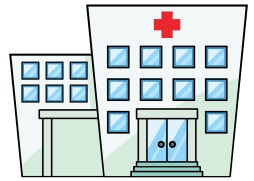
Common
Data
Repository

magic

Extract

Transform

Load



Healthcare
Providers

Pharmaceutical
Companies

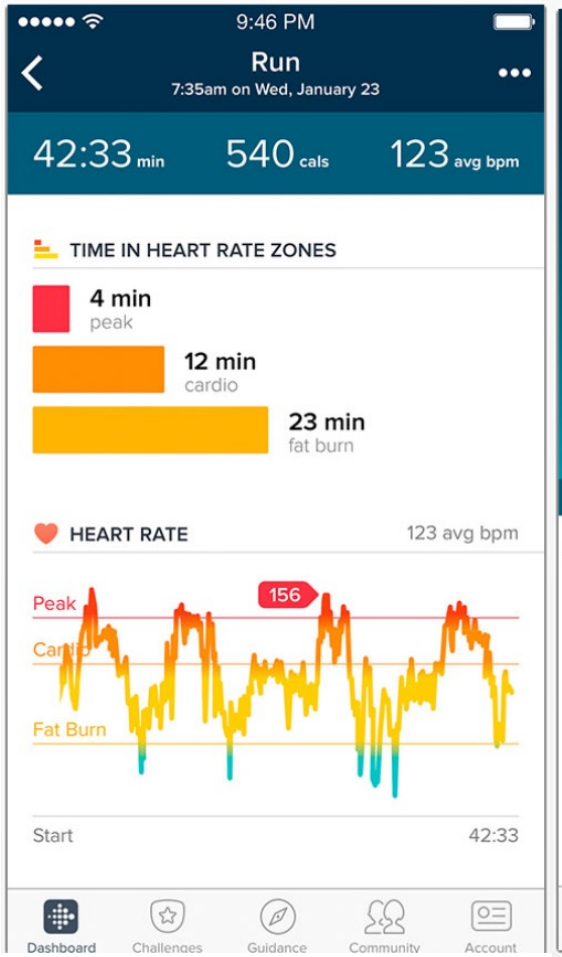
.

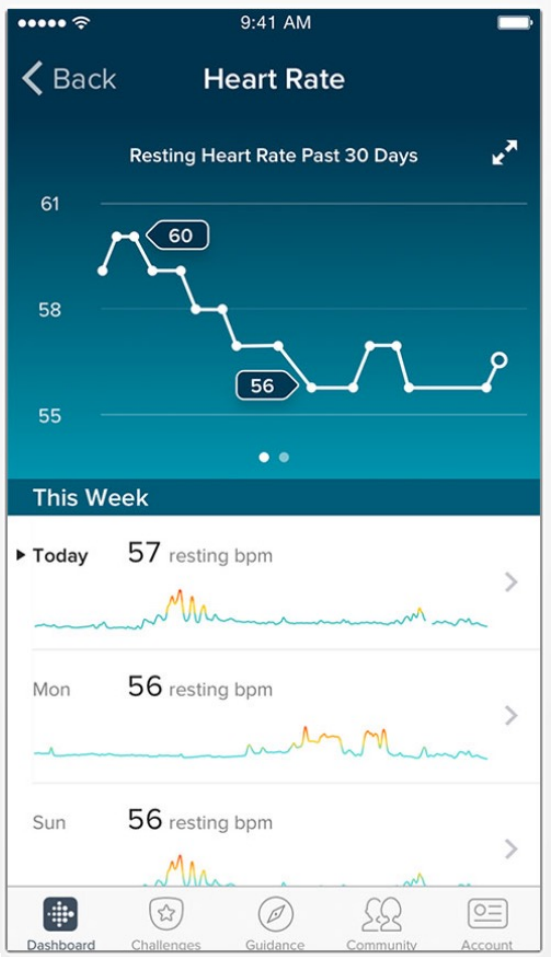
.

.

What questions would you ask if you had the medical records for 90+ million patients?

Personalized Health / Wearable Sensors





https://finance.yahoo.com/news/exclusive-fitbits-6-billion-nights-sleep-data-reveals-us-110058417.html

The dawn of big data presented new challenges

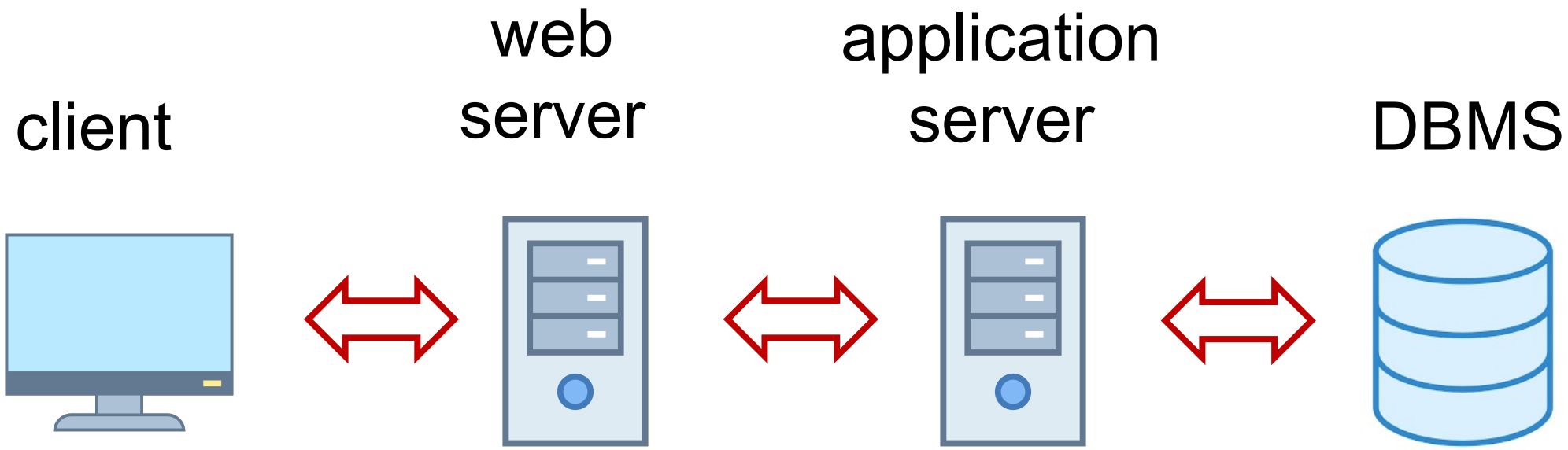
Data assets are [a] major component of the balance sheet, replacing traditional physical assets of the 20th century.

-- Dr. Ralph Kimball: Rethinking EDW in the era of expansive data management

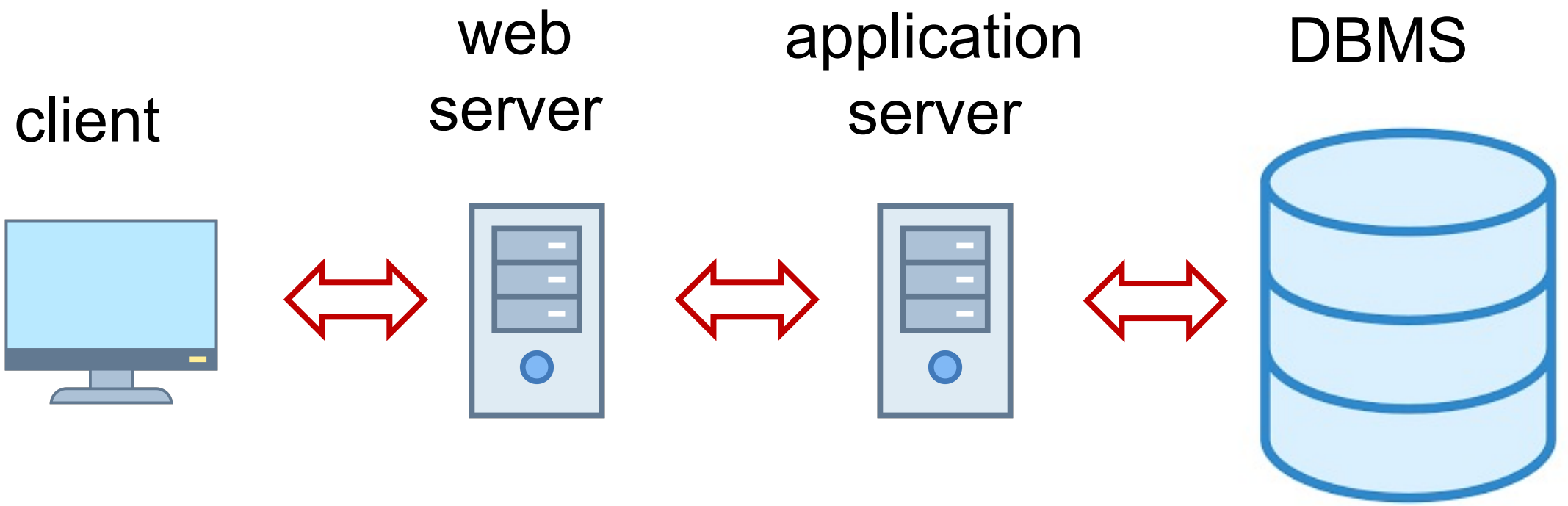
- In the Internet age, we expect *instantaneous* answers to *any* question: medical, financial, scientific, products & services, and turkey recipes!

- It is no longer ok to prune data, say, for the last N days. (We can build better machine learning models, recommendation engines, targeted ads, and predictions if we have *all* of the historical data.)

Early stages of a company. N-tier client-server is adequate

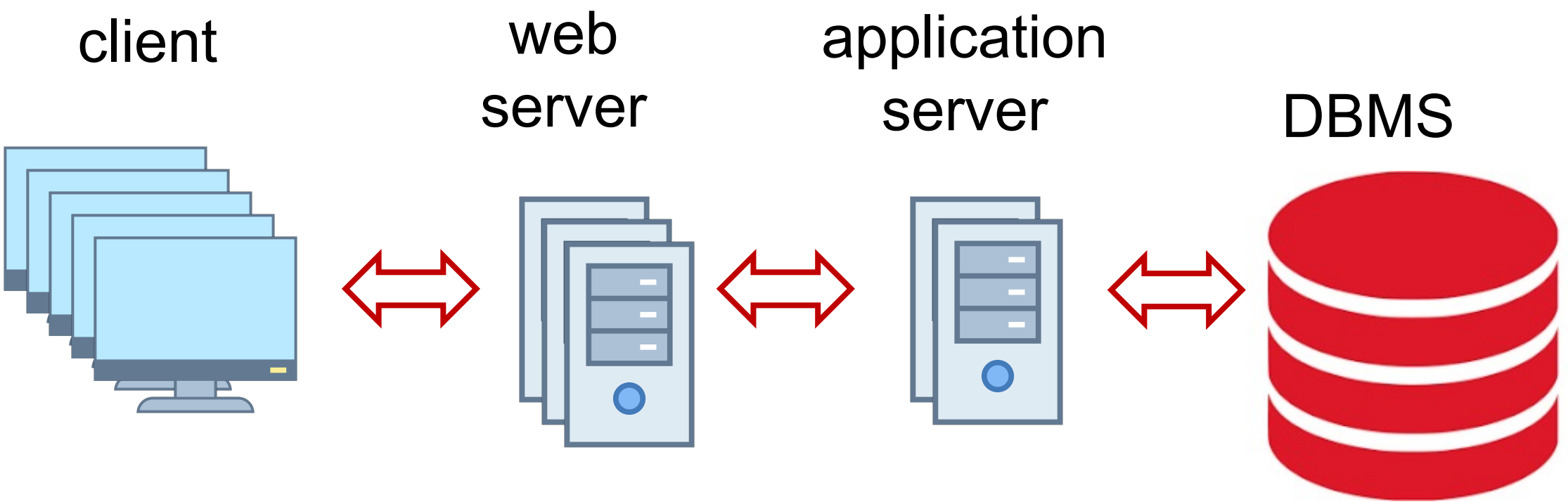


Early stage growth. Vertical scaling

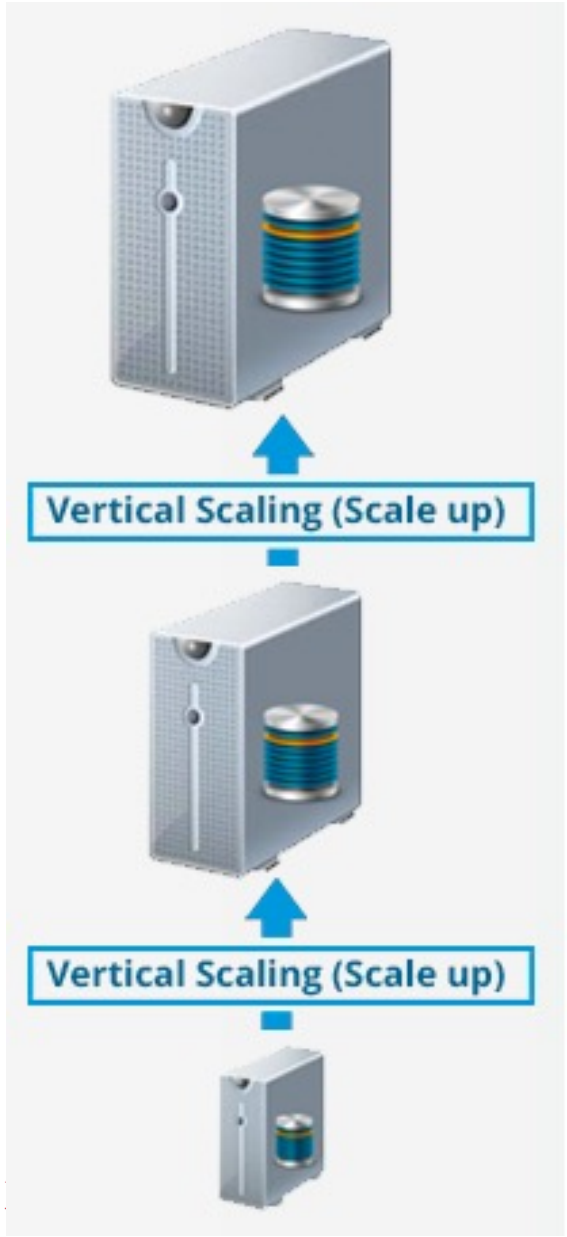


Beef up your DBMS: More RAM, more cores, faster disks, ***more $$$***

Growing user base supported via load balancing



Scaling Up: Advantages and Disadvantages



ADVANTAGES

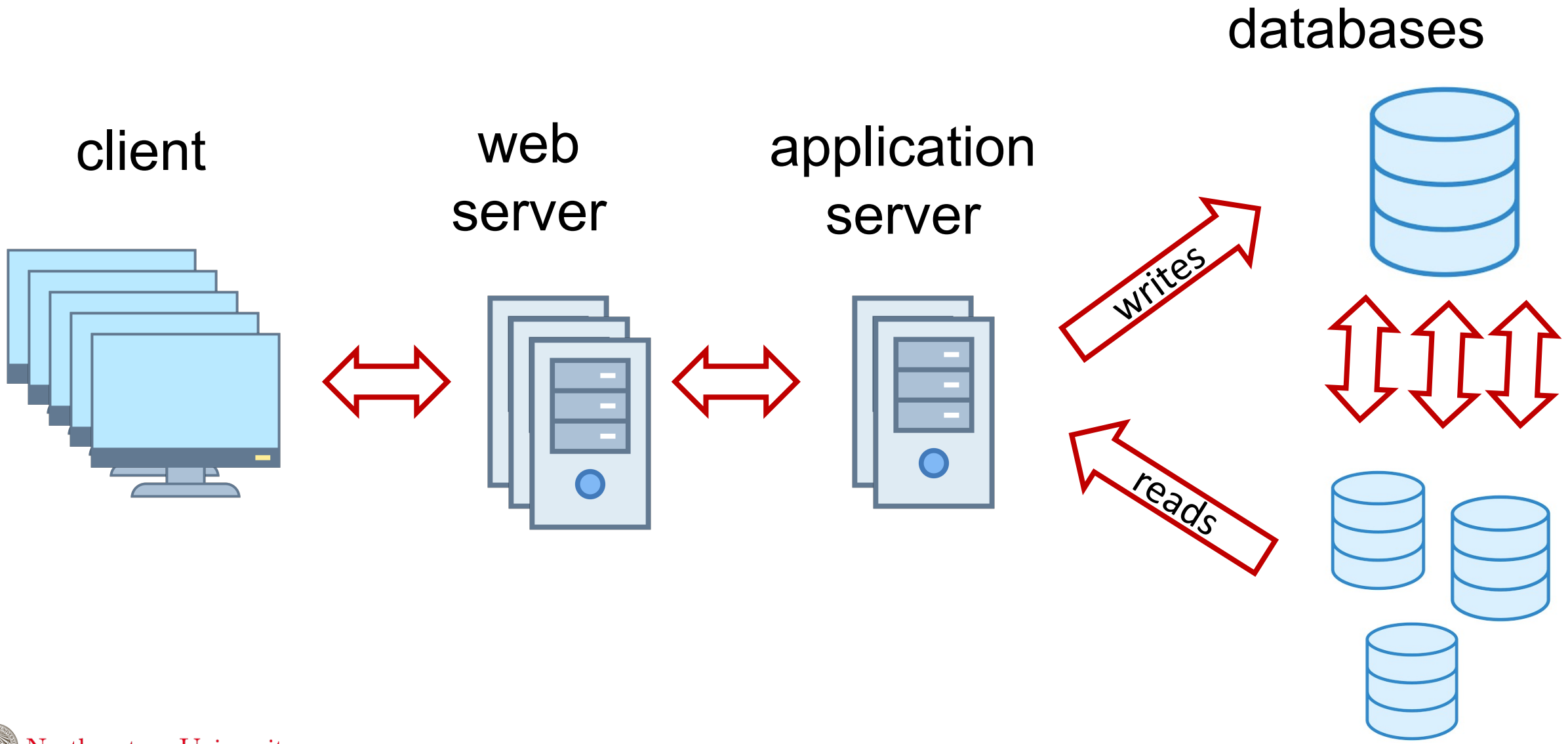
- Leveraging a platform you already know
- Software migration is easier
- Administration is easier
- Relatively easy to manage: add more RAM, CPU or buy a new server if necessary
- Cooling overhead is lower
- Networking hardware overhead is lower

DISADVANTAGES

- Exponential server cost and licensing costs (~10's of millions $)
- Vendor lock-in
- Single point of failure
- Limits on available hardware expansion
- Reliance on specialty human resources

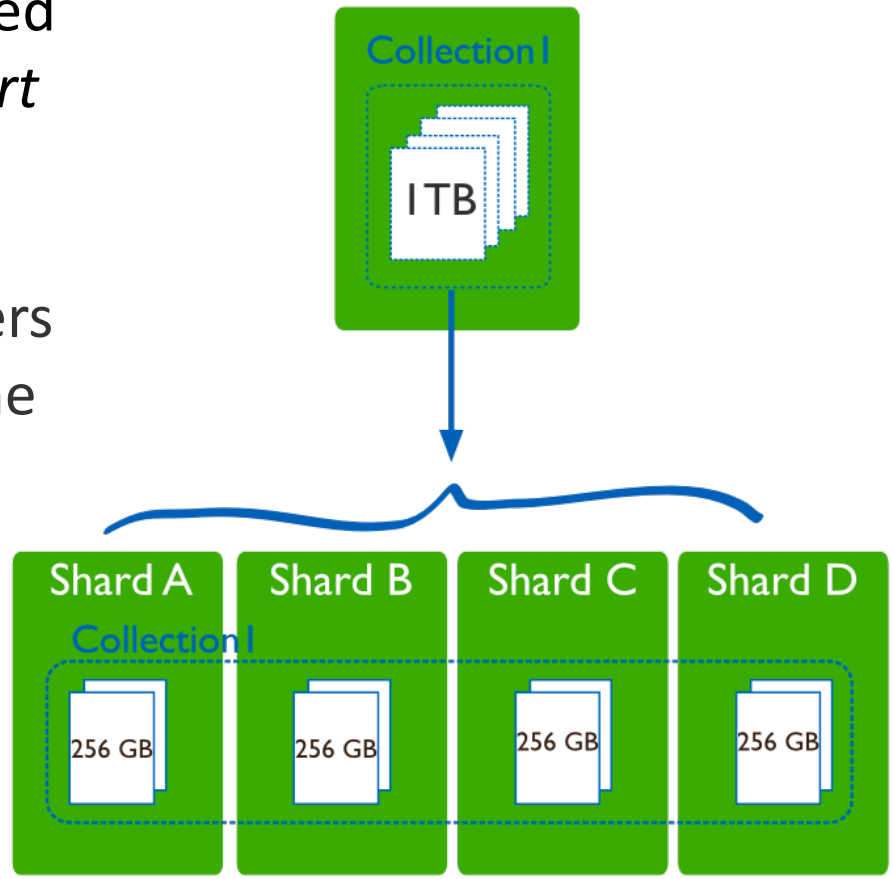
Once you are in the realm of "big data" you need another solution.

Replication: Adding secondary DB servers to keep up with demand



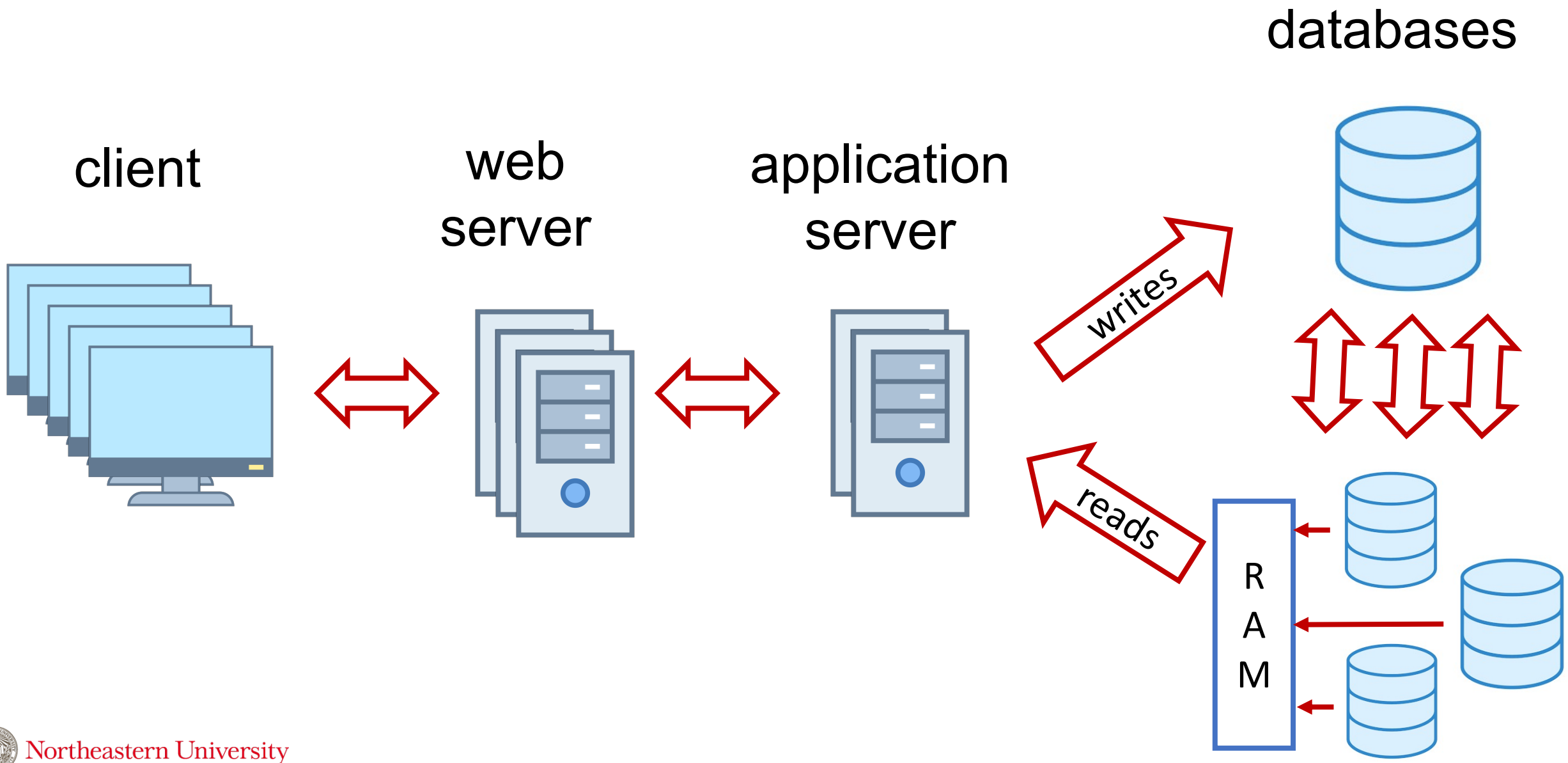
Sharding: Trading application simplicity for speed

Sharding is a type of database partitioning that separates very large databases the into smaller, faster, more easily managed parts called data **shards**. The word shard means *a small part of a whole*.



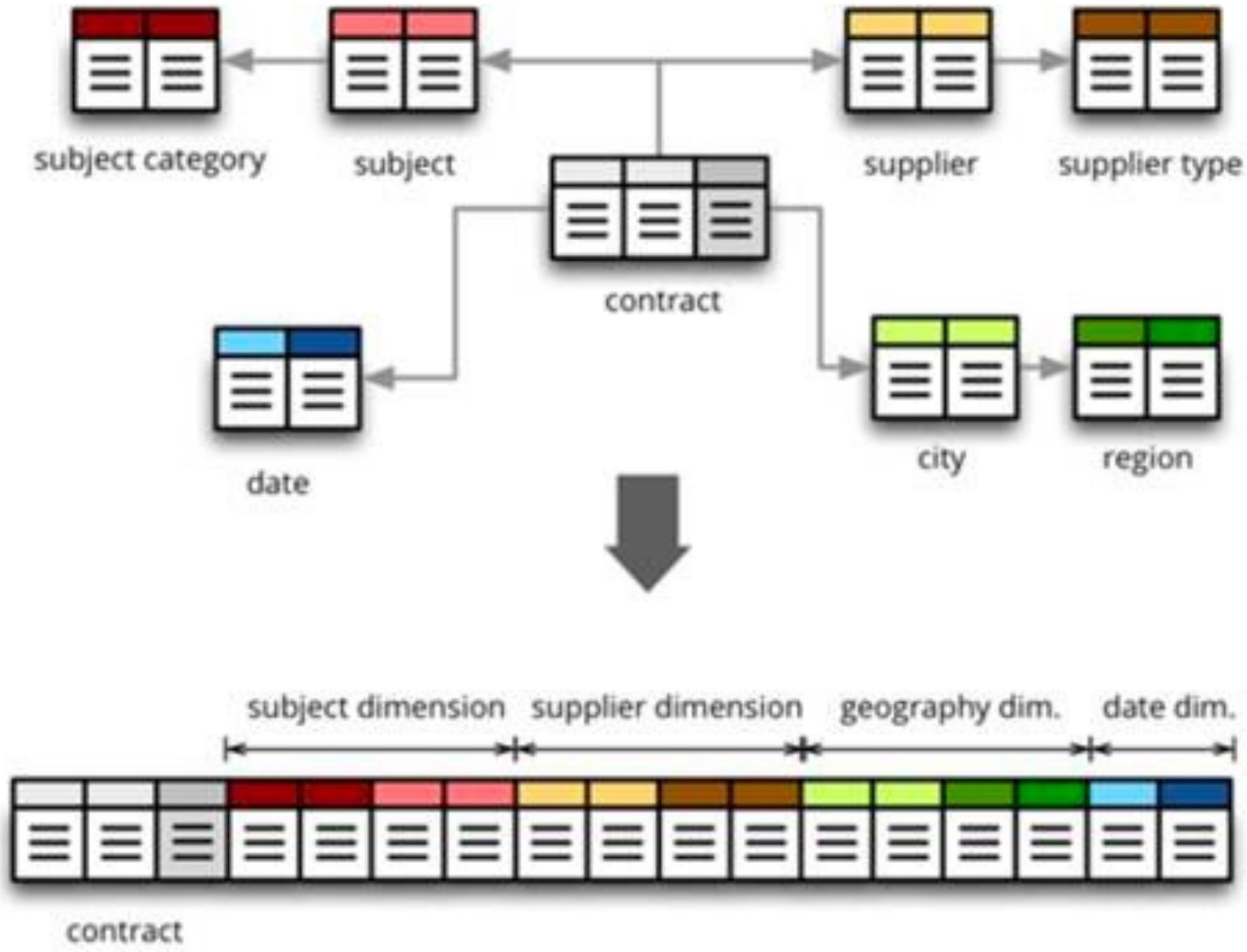
- Relational databases could also be run as separate servers for different shards (sets of data), effectively sharding the database.
- While this separates the load, all the sharding has to be controlled by the application which has to keep track of which database server to talk to for each bit of data.
- *Also, we lose any querying, referential integrity, transactions, or consistency controls that cross shards.*

Memcached: trading consistency for speed.



Denormalization: Trading write performance for speed

- We also incur storage overhead
- Complex JOINs are simplified or eliminated making queries much faster



More desperate measures

- Eliminating stored procedures in favor of off-line processing
- Eliminating secondary indexes which slow down inserts and updates, limiting access to data via primary keys only
- **Materialized views**: To support efficient querying, a common solution is to generate, in advance, a view that materializes the data in a format suited to the required results set.

The Benefits of Relational Databases

- Data persistence via a well-defined schema (without having to know the details of how the data is organized on disk)
- SQL – The query standard (with many dialects). It is both expressive and flexible.
- Data integration: Data can be combined via JOINS for deeper insights
- Reporting / Aggregate analysis
- ACID Transactions

Branch

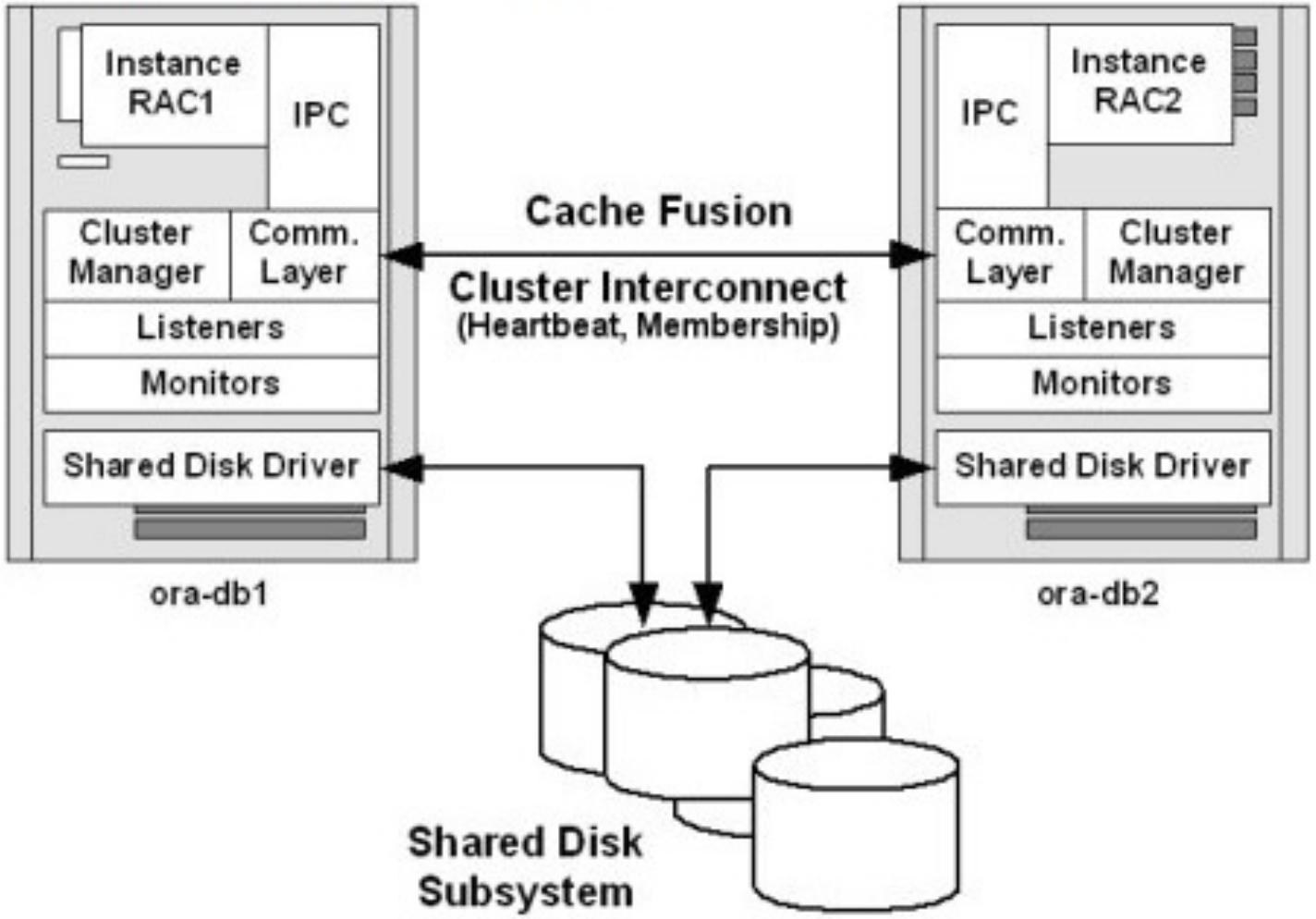
branchNo	street	city	postCode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

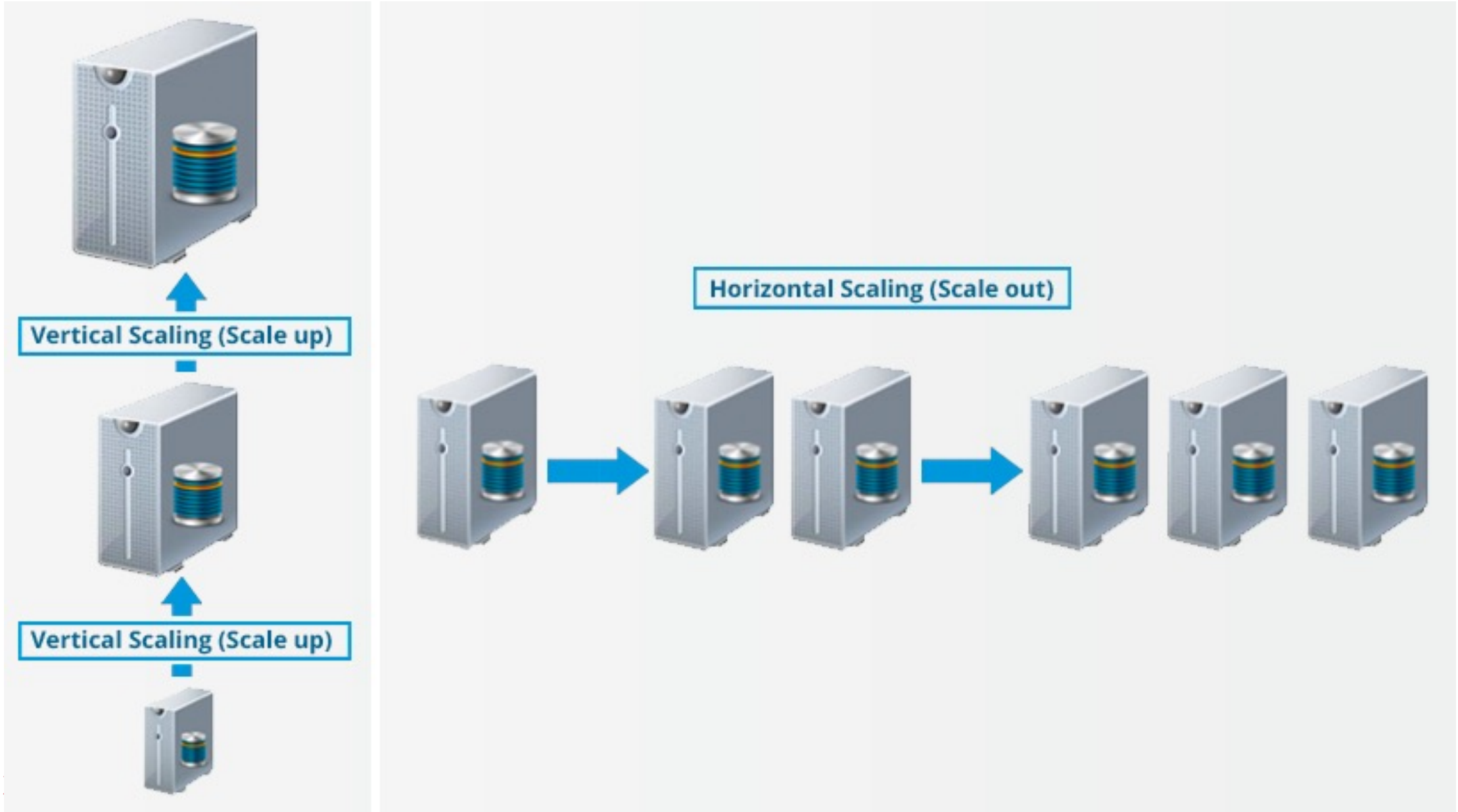
Scale up: Oracle Real Application Cluster (RAC)

The following diagram, adapted from *Oracle Real Application Clusters* by Murali Vallath [*Elsevier Digital Press*, 2004] illustrates some of the basic components of an Oracle RAC cluster under Oracle 9*i*.



Oracle Real Application Cluster (RAC) is a single database that is shared by multiple Oracle instances. It provides opportunities for improved database availability, performance and scalability. Oracle RAC lowers the TCO of a high-end database and IT assets, since a complex Oracle database can be set-up on low-cost, standard, modular parts. **Oracle RAC has matured into a true enterprise, scalable solution, but the adoption curve and skill-set required to maintain RAC is relatively slow, as most organizations who have installed RAC only use it as a two-node cluster.**

Scaling Up vs. Scaling Out



Advantages and disadvantages of *Scaling out*

Advantages

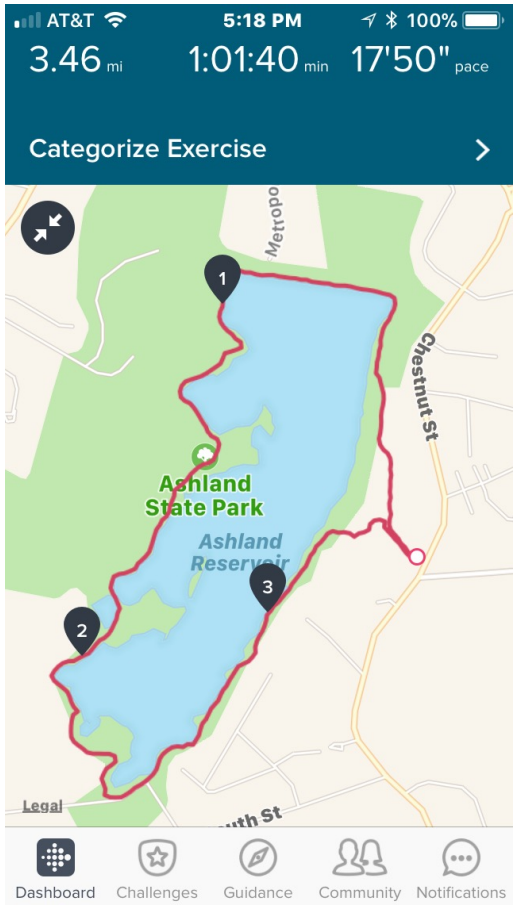
- Horizontal scaling using commodity hardware
- Greater reliability

Disadvantages

- Incompatible with existing relational storage models
- Relational DBs that introduce cluster-based mechanisms have high licensing costs
- The computing models are more complicated
- Security issues may be more complicated or simply less mature
- The technologies are more cutting-edge and less mature making it more challenging to find answers to problems you may encounter.

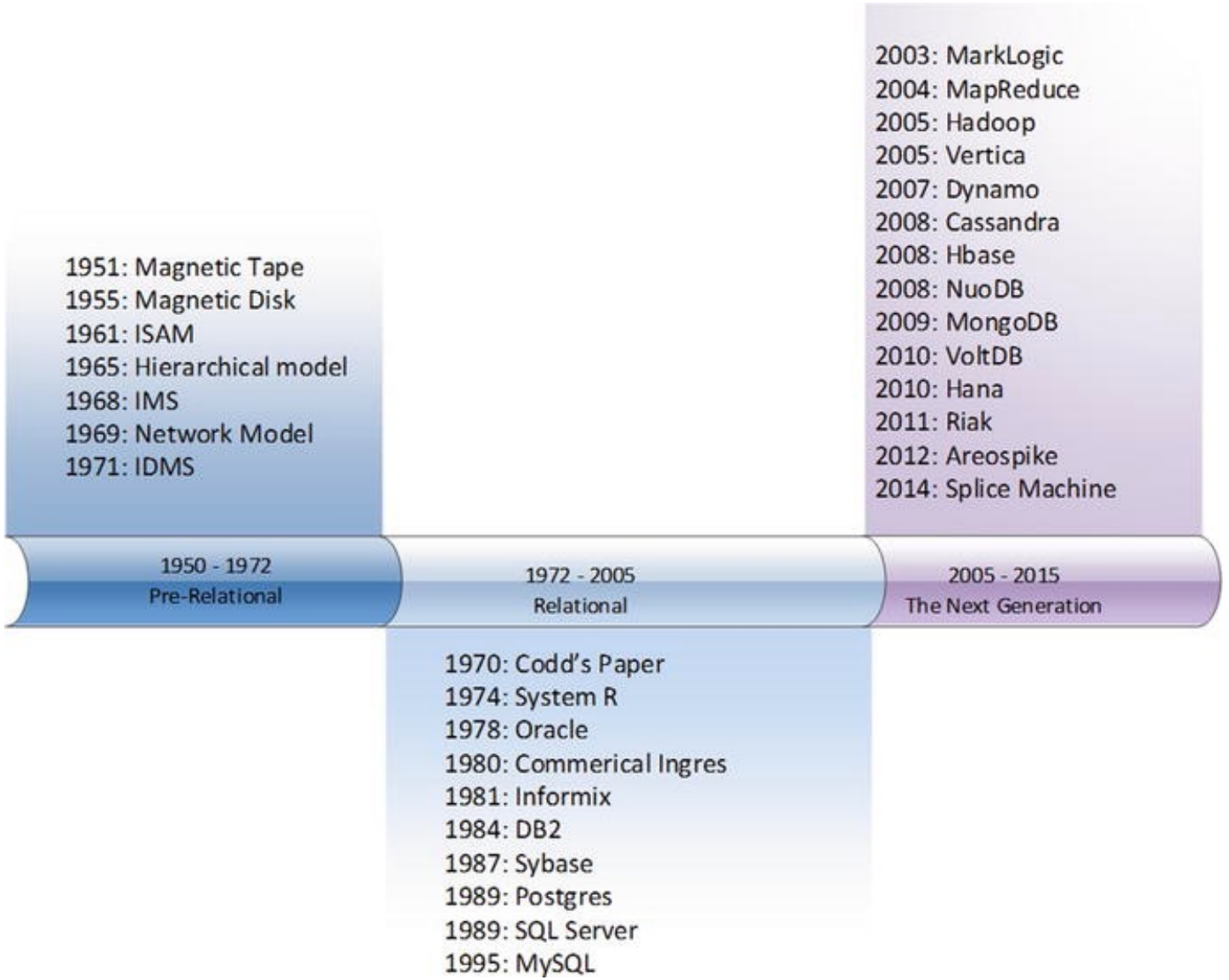
Finding GPS Anomalies

How would you parallelize this?

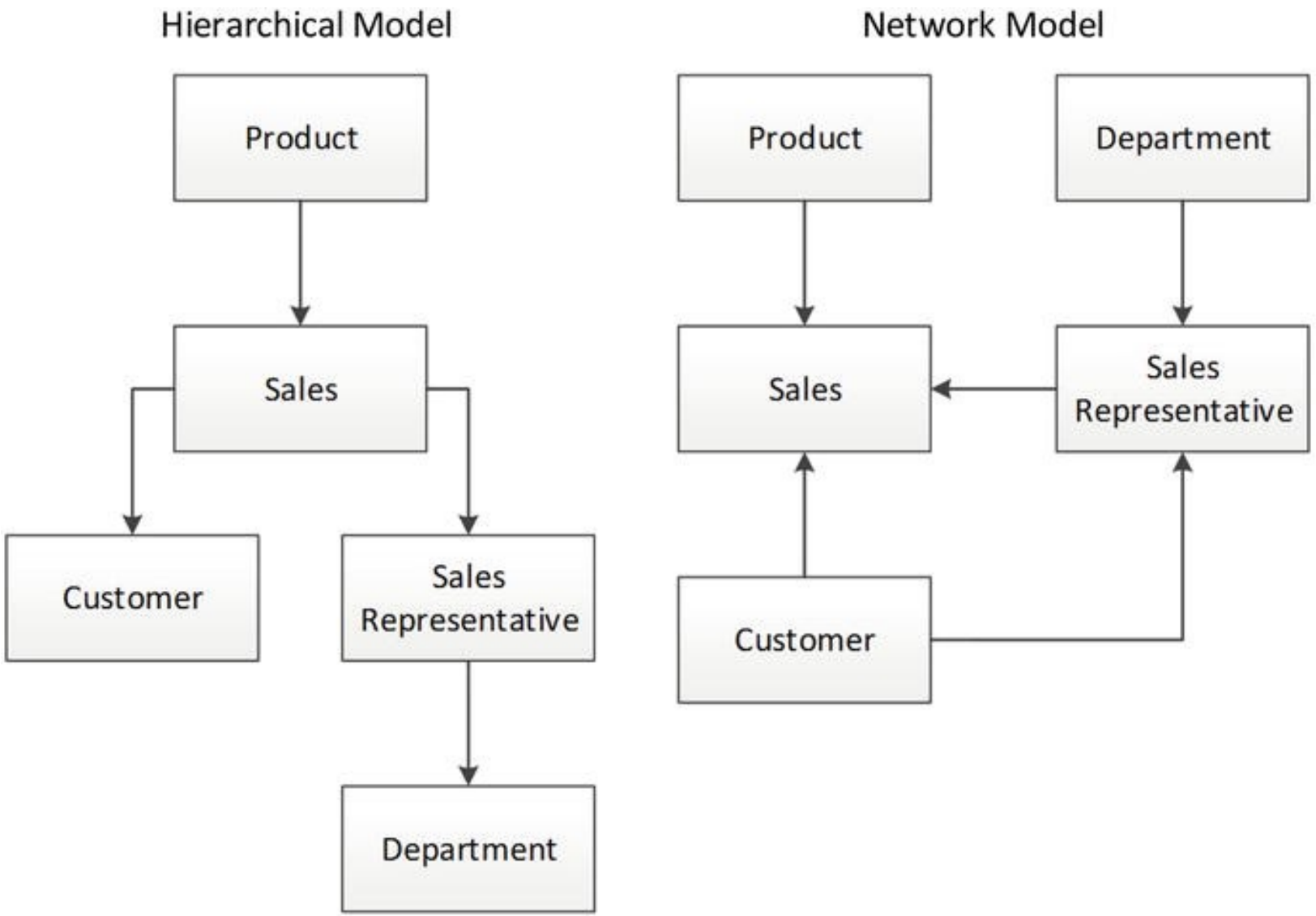



```
anomalies = 0
for each user
   for each exercise
      for each gps measurement
         distance = gps.current – gps.last
         if (distance > 10m) anomalies++

return anomalies
```

The three database revolutions



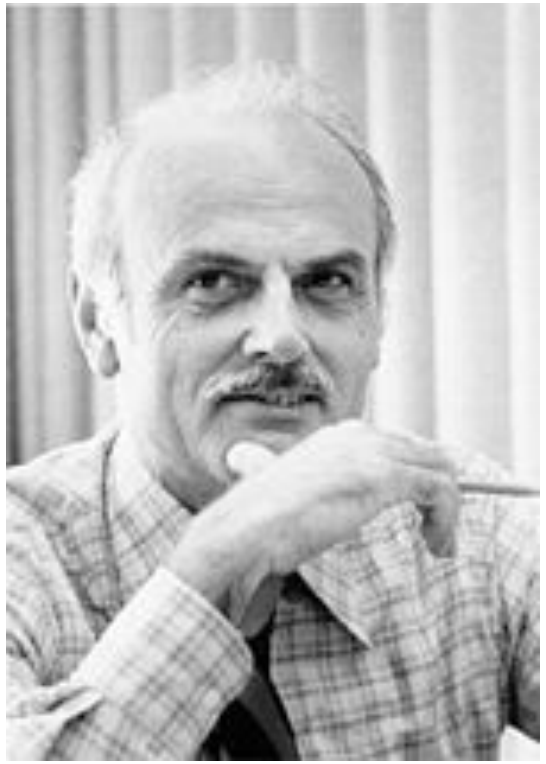
Pre-Relational



Problems with pre-relational databases

- **Hard to use**: required programming expertise
- **Arbitrary representations** – no logical foundation
- **Logical/Physical Dependence** – the representation of the data is closely tied to how it is stored on disk.

Codd, 1970. The origins of the relational model



Edgar Frank Codd
1923-2003

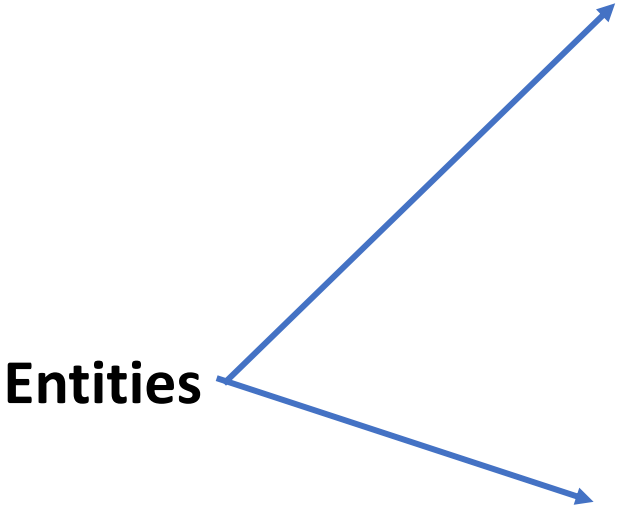
Information Retrieval

A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Relational Data Model



Branch

branchNo	street	city	postCode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

A column or ***attribute*** or **field** providing more details about the particular entity.

- Name
- Datatype
- Other attributes

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

A row or ***record*** describing one instance of the entity

Key concepts of the relational model

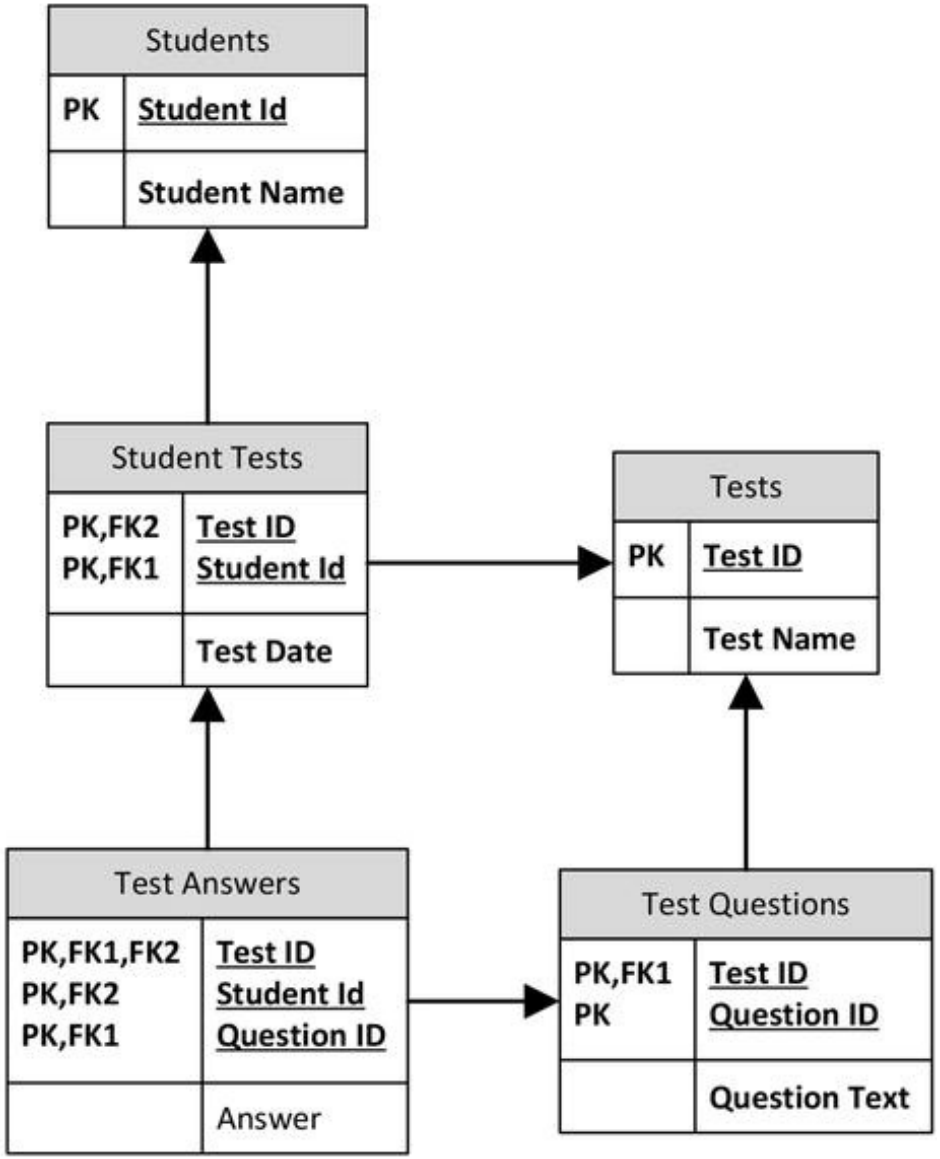
- **Tuples**, an unordered set of **attribute** values. In an actual database system, a tuple corresponds to a row, and an attribute to a column value.
- A **relation**, which is a collection of distinct tuples and corresponds to a table in relational database implementations.
- **Constraints**, which enforce consistency of the database. Key constraints are used to identify tuples and relationships between tuples.
- **Operations** on relations such as joins, projections, unions, and so on. These operations always return relations. In practice, this means that a query on a table returns data in a tabular format.

Normalization

Un-normalized data

Test scores
Student Name Test Name Test Date Answer 1 Answer 2 Answer 3 Answer 4 Answer 5 Answer 6 Answer N

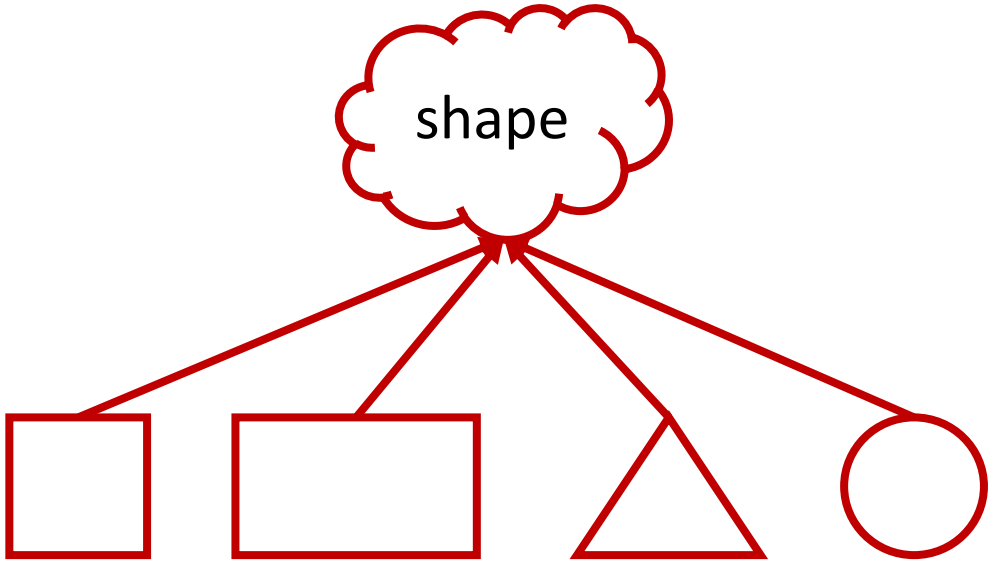
Normalized data



"All non-key attributes depend on the key, the whole key, and nothing but the key -- so help me Codd!"

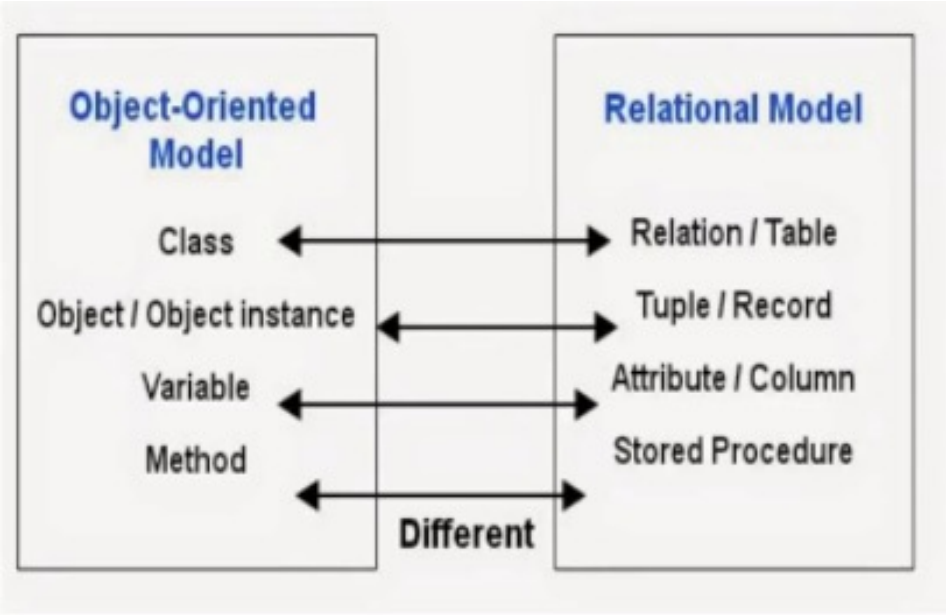
Object-Oriented Design

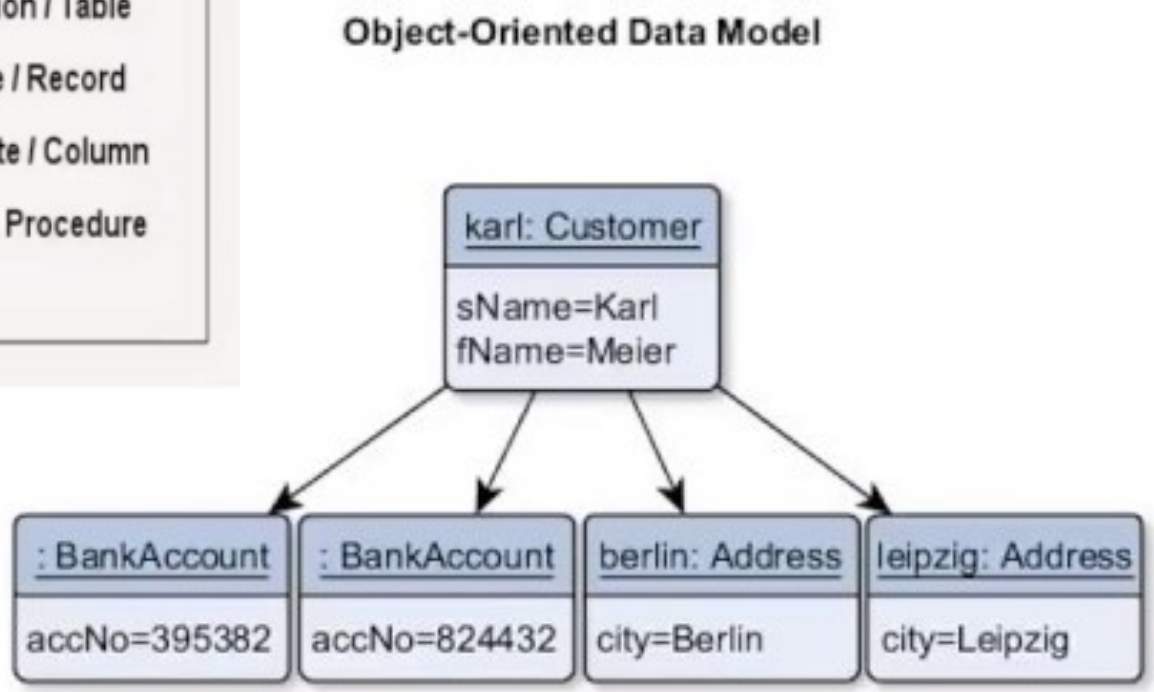
- **Encapsulation**: A *class* encapsulates both data and methods.
- **Inheritance**: Classes can inherent properties & methods of parent classes.




```
Class Circle extends Shape
{
    private double radius;

    double area() { return pi * r * r ; }
}
```

Relational vs. Object-Oriented Databases (1990's)





Relational Data Model

Customers

ID	SName	FName
1	Karl	Meier

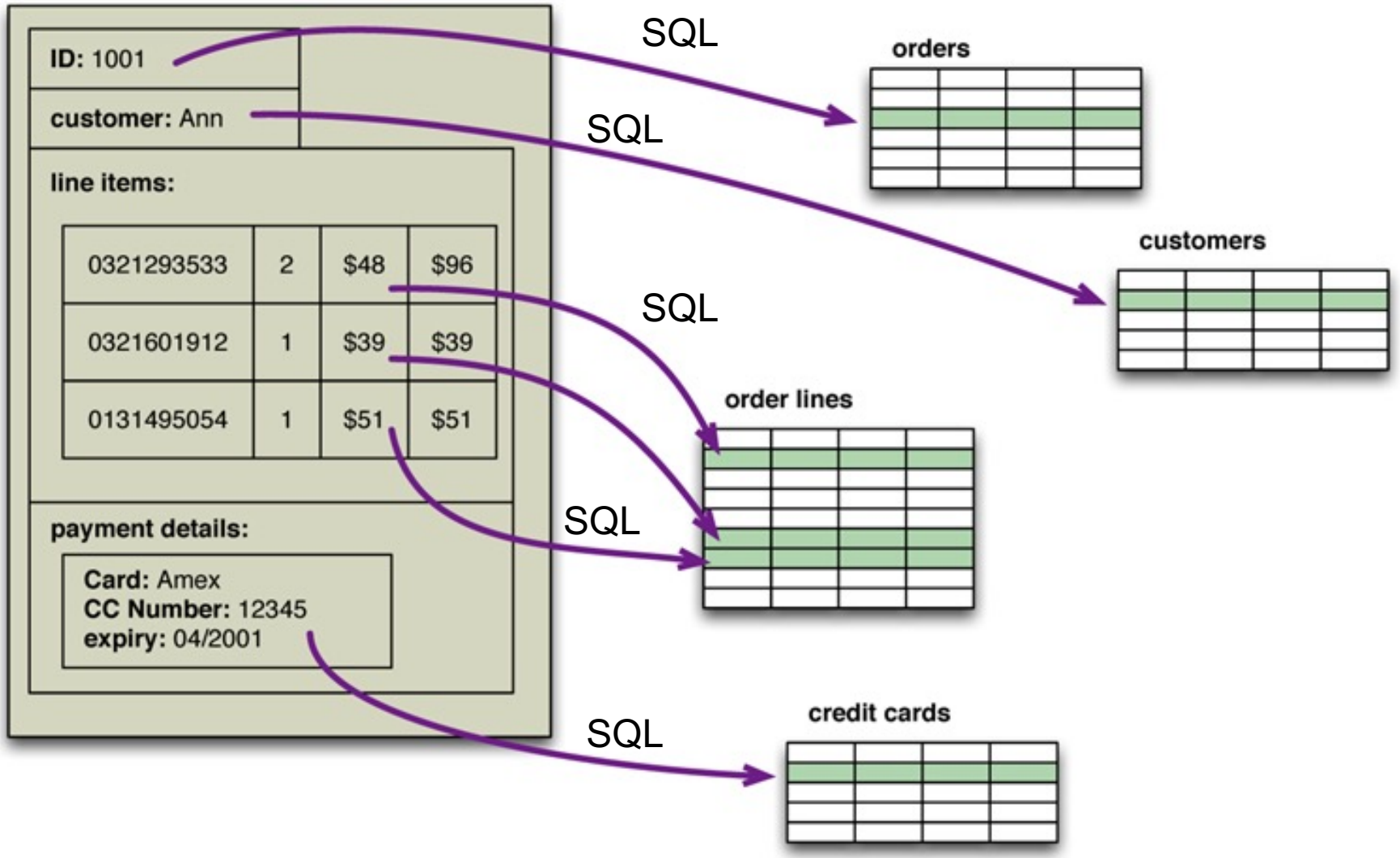
Addresses

ID	City	CID
1	Berlin	1
2	Leipzig	1

BankAccounts

ID	AccNo	CID
1	395382	1
2	824432	1

Why NoSQL?: Impedance mismatch problem



Beyond Relational Databases

A relational database is like a garage that forces you to take your car apart and store the pieces in little drawers.

-- Object-oriented data community, mid-1990s

An object database is like a closet which requires that you hang up your suit, with tie, underwear, belt, socks and shoes all attached.

-- David Ensor, same period.

What happened to Object-Oriented Databases?

- Object-oriented databases were an attempt in the 1980's and 1990's to store objects in a DB and thus avoid the impedance mismatch problem.

- *The idea never took off*, although object-oriented programming is now universal. Why?

  a. **SQL became an industry standard** as a foundation for supporting data integration efforts

  b. The problem was abated to some extent by the emergence of object-relational mapping (ORM) frameworks such as **Hibernate**

  c. **Professional divide** between application developers and database administrators

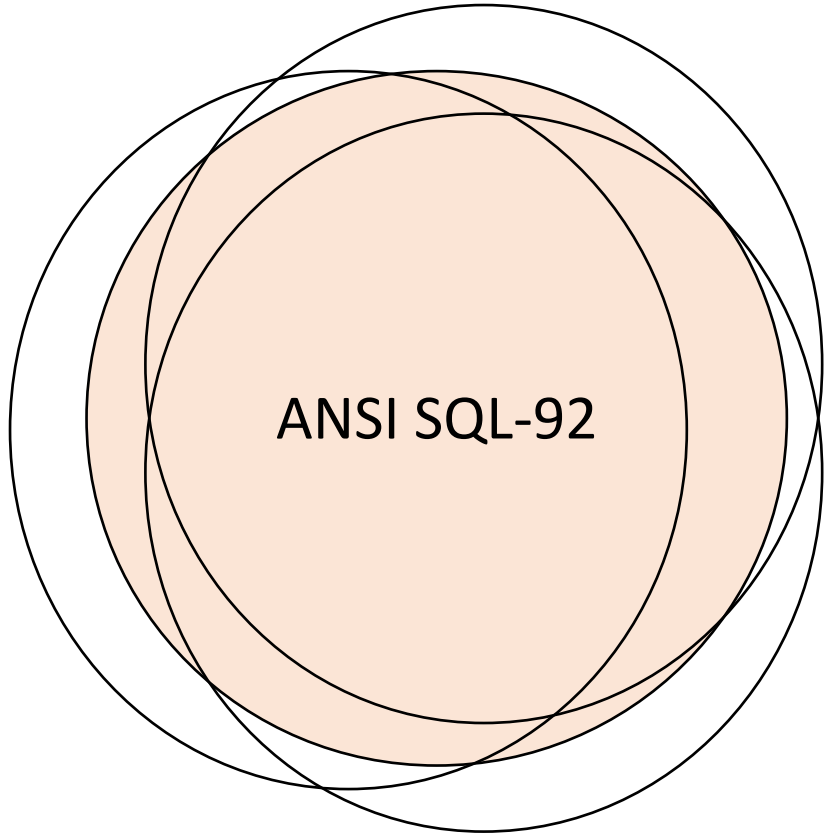
  d. Big data wasn't a technological driver (yet!)

Transaction models: ACID

- **Atomic**: The transaction is indivisible—either all the statements in the transaction are applied to the database or none are.
- **Consistent**: The database remains in a consistent state before and after transaction execution.
- **Isolated**: While multiple transactions can be executed by one or more users simultaneously, one transaction should not see the effects of other in-progress transactions.
- **Durable**: Once a transaction is saved to the database (in SQL databases via the COMMIT command), its changes are expected to persist even if there is a failure of operating system or hardware.

“But everyone knows SQL!”

Year	Name	Alias	Comments
1986	SQL-86	SQL-87	First formalized by ANSI.
1989	SQL-89	FIPS 127-1	Minor revision that added integrity constraints, adopted as FIPS 127-1.
1992	SQL-92	SQL2, FIPS 127-2	Major revision (ISO 9075), *Entry Level* SQL-92 adopted as FIPS 127-2.
1999	SQL:1999	SQL3	Added regular expression matching, recursive queries (e.g. transitive closure), triggers, support for procedural and control-of-flow statements, non-scalar types (arrays), and some object-oriented features (e.g. structured types). Support for embedding SQL in Java (SQL/OLB) and vice versa (SQL/JRT).
2003	SQL:2003		Introduced XML-related features (SQL/XML), *window functions*, standardized sequences, and columns with auto-generated values (including identity-columns).
2006	SQL:2006		ISO/IEC 9075-14:2006 defines ways that SQL can be used with XML. It defines ways of importing and storing XML data in an SQL database, manipulating it within the database, and publishing both XML and conventional SQL-data in XML form. In addition, it lets applications integrate queries into their SQL code with XQuery, the XML Query Language published by the World Wide Web Consortium (W3C), to concurrently access ordinary SQL-data and XML documents.[34]
2008	SQL:2008		Legalizes ORDER BY outside cursor definitions. Adds INSTEAD OF triggers, TRUNCATE statement,[35] FETCH clause.
2011	SQL:2011		Adds temporal data (PERIOD FOR)[36] (more information at: Temporal database#History). Enhancements for *window functions* and FETCH clause.[37]
2016	SQL:2016		Adds row pattern matching, polymorphic table functions, JSON.



One standard, many dialects

The beginning of NoSQL in the 2000's

Google (2006). BigTable: A Distributed Storage System for Structured Data
Proceedings of the 7th USENIX Symposium on Operating Systems Design and implementation: pp 205-218

> Abstract Excerpt: Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance.

https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf

The beginning of NoSQL in the 2000's

Amazon (2007). Dynamo: Amazon's Highly Available Key-value Store
21st ACM Symposium on Operating Systems Principles: pp 205-218

Abstract Excerpt:

This paper presents the design and implementation of Dynamo, a highly available key-value storage system that some of Amazon's core services use to provide an "always-on" experience…To achieve this level of availability, Dynamo sacrifices consistency under certain failure scenarios. It makes extensive use of object versioning and application-assisted conflict resolution in a manner that provides a novel interface for developers to use.

http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf

The origins of “NoSQL” – Johan Oskarsson’s SF Meetup

Braindump

Follow johanoskarsson tumblr.

NOSQL meetup

05/12/09

Hadoop summit
I’m going to attend the Hadoop summit in San Francisco in June, had a great time last year, learned a bunch of stuff and got to meet a lot of people I previously only knew by name.

NOSQL
To make the most of the flight money I’m putting together a free meetup about “open source, distributed, non relational databases” or NOSQL for short.

It’s taking place on the 11th of June, the day after the Hadoop summit in San Francisco. CBS interactive have been kind enough to provide us with both a venue and free lunch!

If you wish to attend, please register.

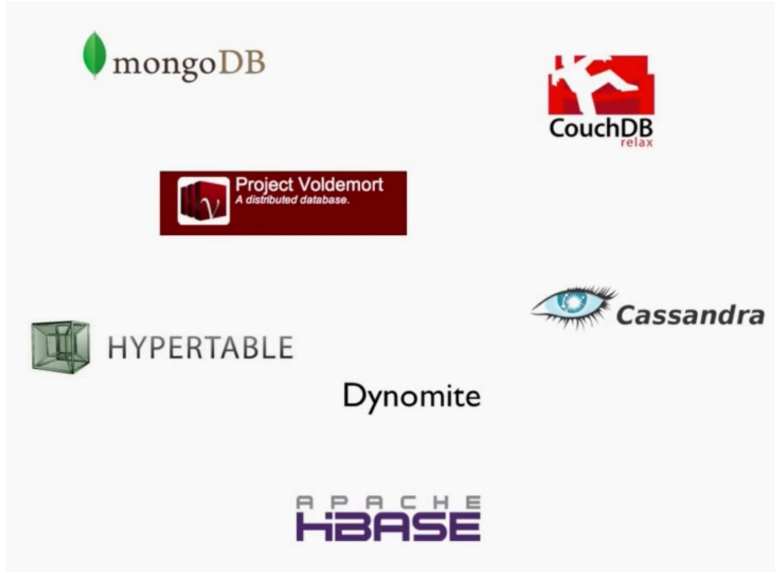
Preliminary schedule
09.45: Doors open
10.00: Intro session (Todd Lipcon, Cloudera)
10.40: Voldemort (Jay Kreps, Linkedin)
11.20: Short break
11.30: Cassandra (Avinash Lakshman, Facebook)
12.10: Free lunch (sponsored by CBSi)
13.10: Dynomite (Cliff Moon, Powerset)
13.50: HBase (Ryan Rawson, Stumbleupon)
14.30: Short break
14.40: Hypertable (Doug Judd, Zvents)
15.20: Panel discussion
16.00: End of meetup, relocate to a pub called Kate O’Brien’s nearby

Things @skr

Search

view archive

“open source, distributed, non-relational databases”



Characteristics of NoSQL (but there are always exceptions)

- Non-relational
- Open-source (but many have commercial versions with add-ons and support)
- Designed for distributed storage / cluster environments
- Schema-less: Fields can be freely added on the fly.
- Non-SQL-like query languages
- Not ACID-compliant. Different options for consistency and distribution of data.
- Databases designed for 21st century web estates with “big data” needs.

NoSQL might mean “No SQL” or “Not Only SQL”

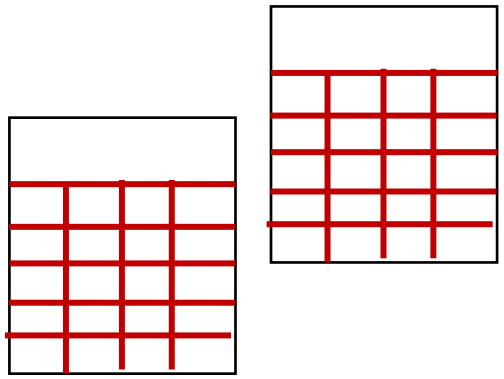
NoSQL supports a trend towards **polyglot persistence**: using different data stores in different circumstances as part of a complete data-engineering architecture.

Schema-less

- NoSQL databases operate without a schema: fields added freely w/o redefinition
- Ideal for non-uniform data and custom fields

	Relational	NoSQL
Data model(s)	Tables	Key-Value, Documents, Graphs, Columns
Computing model(s)	Centralized, rate limiting	Distributed, scalable
Schema	**Pre-declared**	**Flexible**
Datatypes on attributes	Fixed	Not fixed
Hierarchical data types	No	Yes
Cost	Licensing costs become prohibitive as you scale vertically.	Open-source software. Hardware costs scale linearly

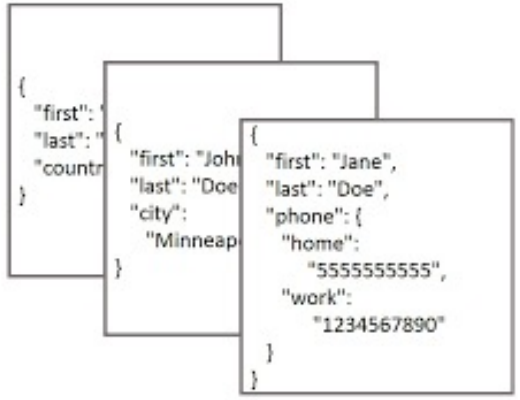
Types of models



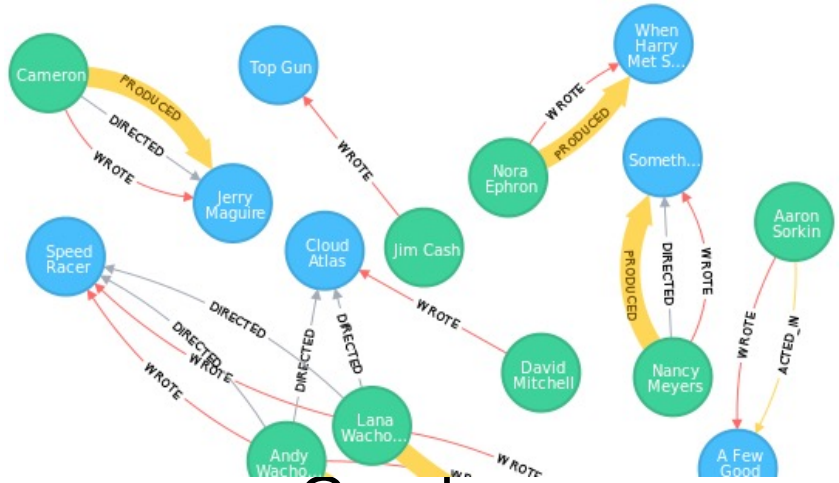
Relational

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

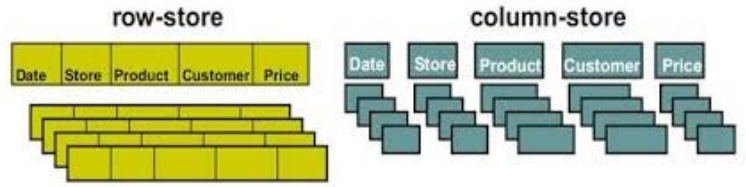
Key-Value

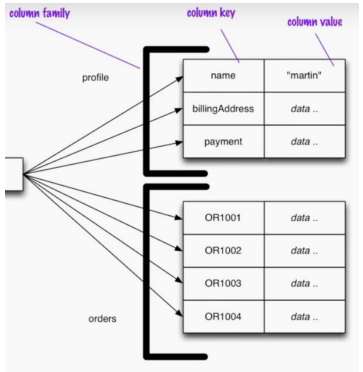


Document



Graphs

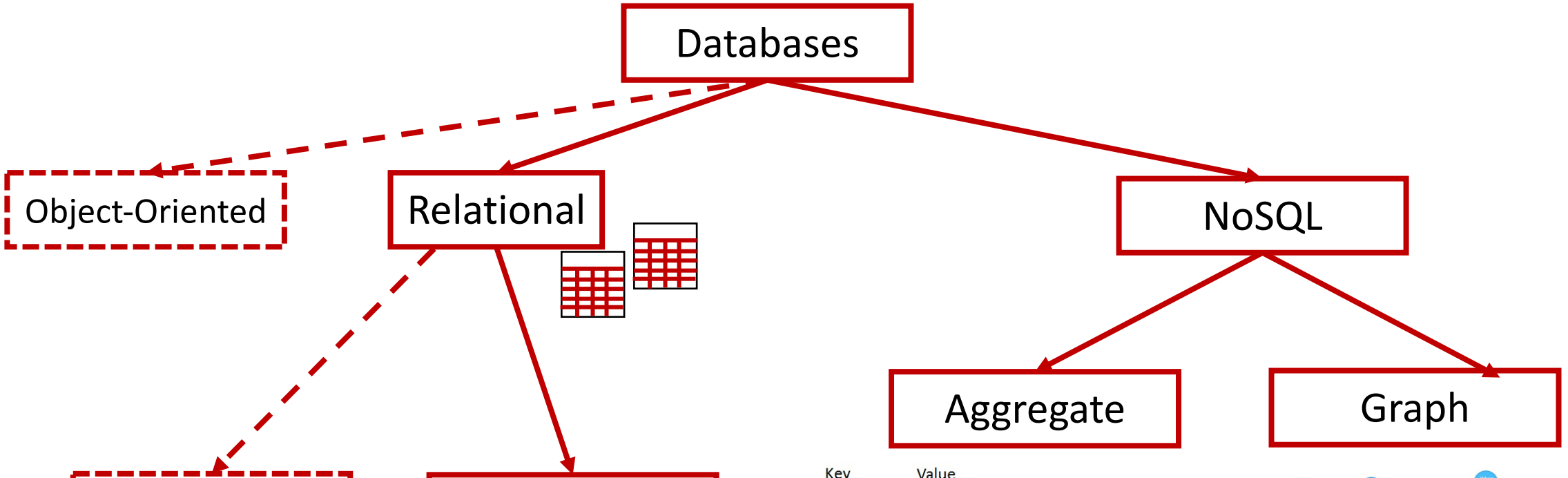




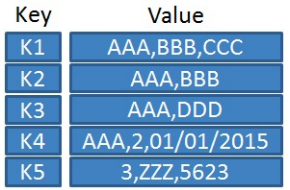
Column-oriented storage
Column-family storage

The non-relational explosion

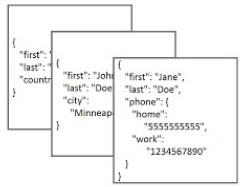
The hierarchy of NoSQL



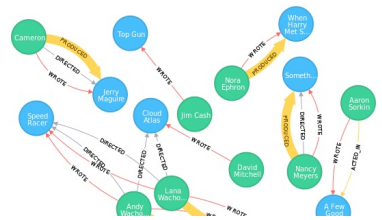
H-Store



Key-Value

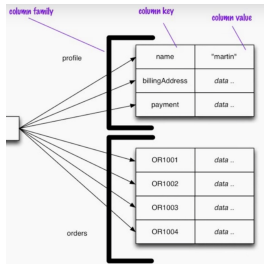


Document

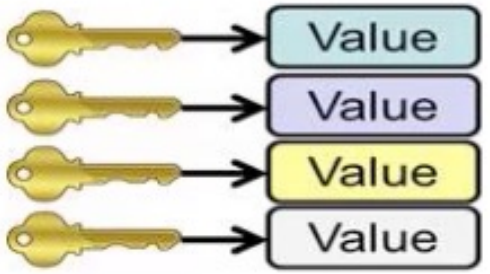


Graphs

Column-family



Key Value



- Your access to data is only via a key, but the value could be anything: a simple value, a list, a document, even a database
- Often used when you need very fast but simple operations
- Common use case: managing online shopping carts.

Document Databases

- A **document database** is a non-relational database that stores data as structured documents, usually XML or JSON formats.

- Simple, flexible, scalable

```
// in customers
{
"id":1,
"name":"Martin",
"billingAddress":[{"city":"Chicago"}]
}

// in orders
{
"id":99,
"customerId":1,
"orderItems":[
  {
  "productId":27,
  "price": 32.45,
  "productName": "NoSQL Distilled"
    }
  ],
"shippingAddress":[{"city":"Chicago"}]
"orderPayment":[
  {
    "ccinfo":"1000-1000-1000-1000",
    "txnId":"abelif879rft",
    "billingAddress": {"city": "Chicago"}
  }
  ],
}
```

XML (eXtensible Markup Language): A document format

- Used as an information storage standard
- XML + CSS (Cascading Style Sheets) enables website design that distinguishes data and format.

- The foundation for web-service specifications (SOAP)

- A rich ecosystem of tools and standards that includes:

  Xpath – Element retrieval
  Xquery –XML document querying
  XML schema – meta-specification for validation
- **XSLT** –XML transformations, e.g., HTML
  DOM – A document navigation API

```
<customer_record>
<customer_id>187693</customer_id>
 <name>"Kiera Brown"</name>
 <address>
    <street>"1232 Sandy Blvd."</street>
    <city>"Vancouver"</city>
    <state>"Washington"</state>
    <zip>"99121"</zip>
 </address>
 <first_order>"01/15/2013"</first_order>
 <last_order>"06/27/2014"</last_order>
</customer_record>
```

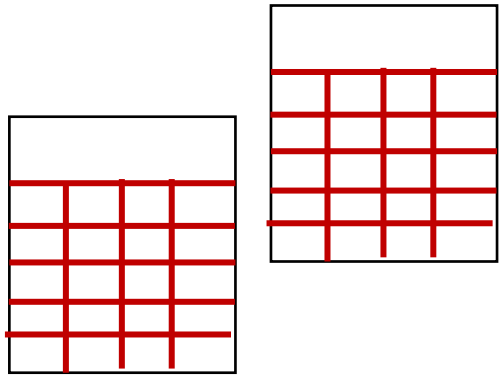
Relational to NoSQL: Concept Mapping

RDBMS		MongoDB
Database		Database
Table, View		Collection
Row		Document (BSON)
Column		Field
Index		Index
Join		Embedded Document
Foreign Key		Reference
Partition		Shard

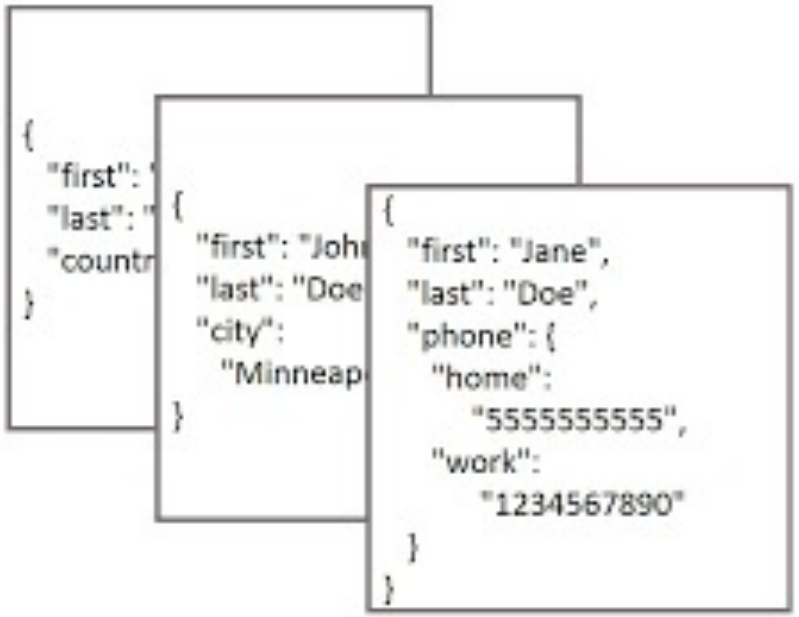
- Collection is not strict about what it stores
- Schema-less
- Hierarchy is evident in the design
- Embedded document are used to represent relationships explicitly.

Document databases (MongoDB): a happy medium

Relational Databases
Pre-defined / rigid schema
SQL: Structured Query Language



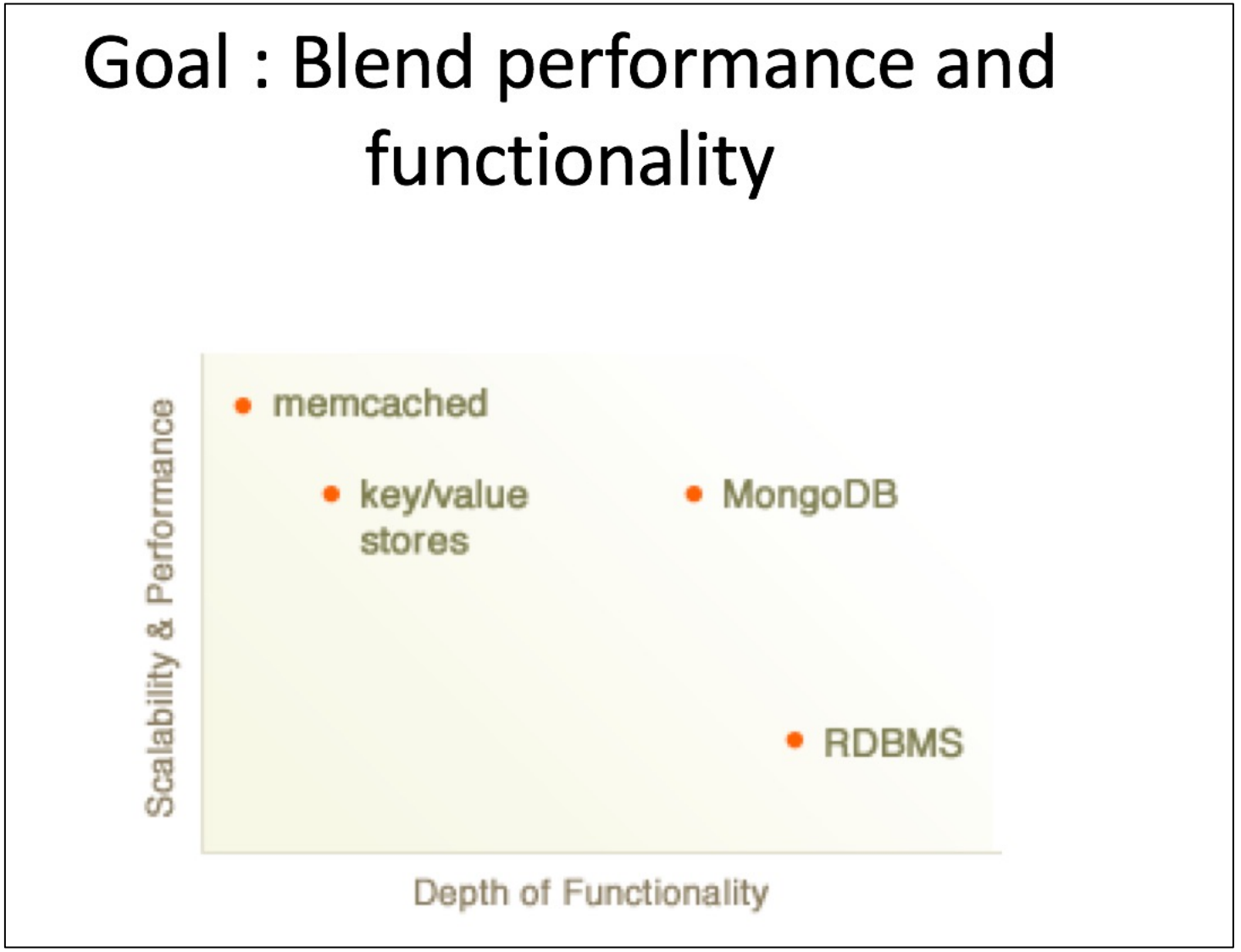
Document Databases
Data is self-describing and flexible
Query by content



Key-Value Stores
Data is simple hash tables
Query by key only (with exceptions)

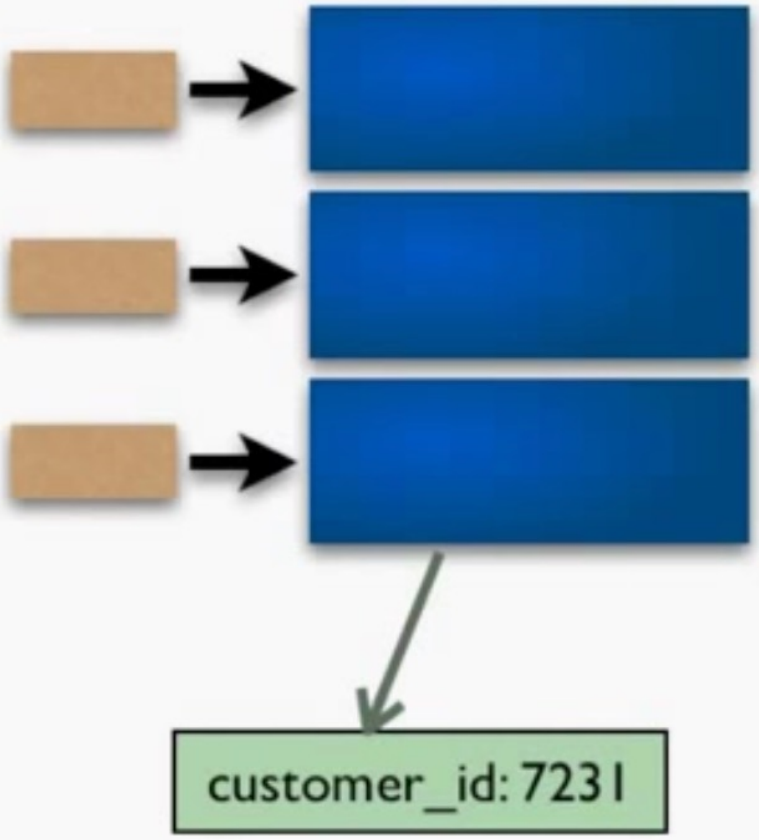
Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

Slide from the original 2009 SF Meetup on NoSQL



The boundary between key-value and document is blurry

Key-Value

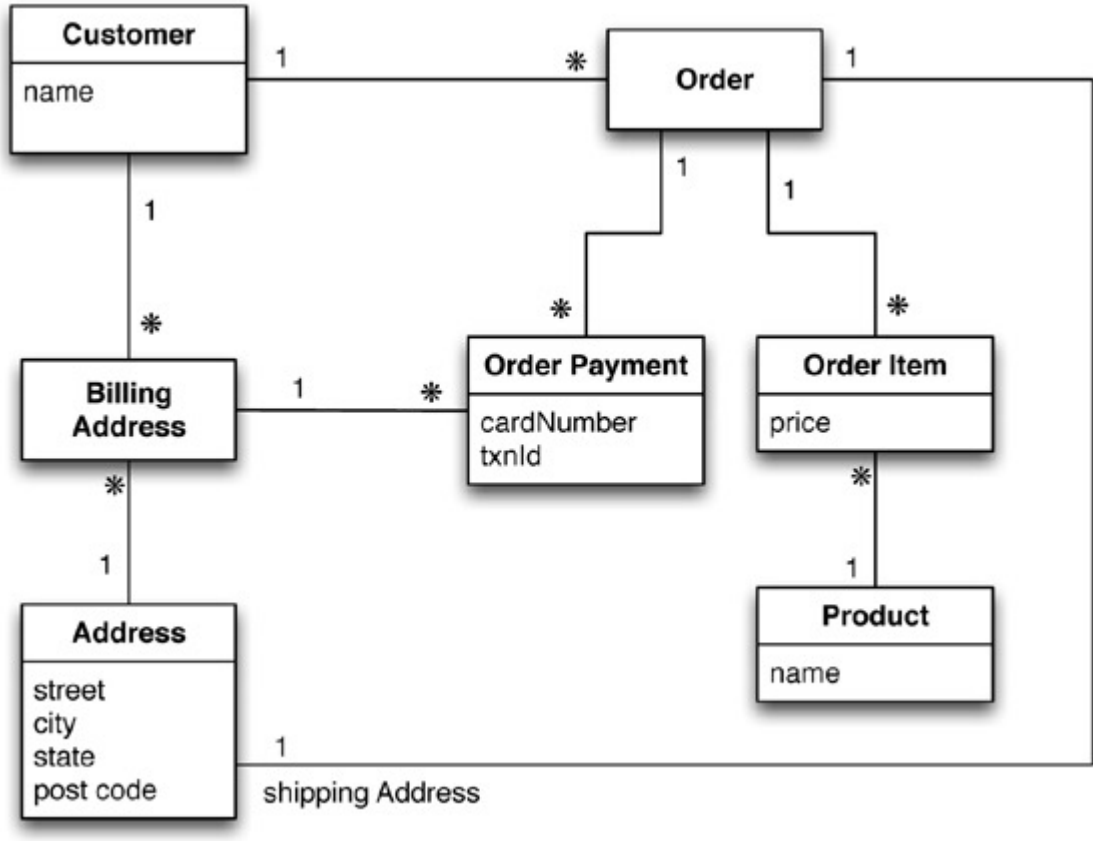


metadata

Document

```
{"id": 1001,
  {"id": 1002,
     "customer_id": 7231,
     "line-itmes": [
        {"product_id": 4555, "quantity": 8},
        {"product_id": 7655, "quantity": 4},
}       {"product_id": 8755, "quantity": 3}]
  }
```

Aggregate vs. Relational



Customer

Id	Name
1	Martin

Orders

Id	CustomerId	ShippingAddressId
99	1	77

Product

Id	Name
27	NoSQL Distilled

BillingAddress

Id	CustomerId	AddressId
55	1	77

OrderItem

Id	OrderId	ProductId	Price
100	99	27	32.45

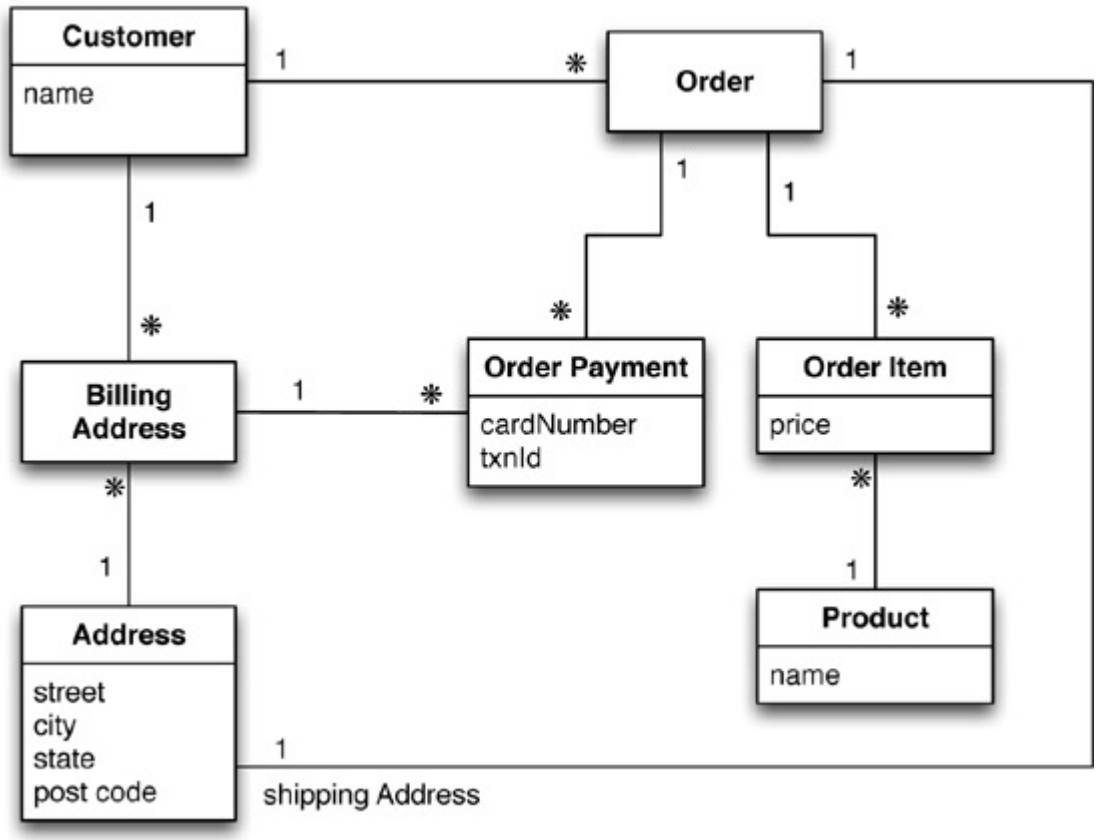
Address

Id	City
77	Chicago

OrderPayment

Id	OrderId	CardNumber	BillingAddressId	txnId
33	99	1000-1000	55	abelif879rft

Aggregate vs. Relational



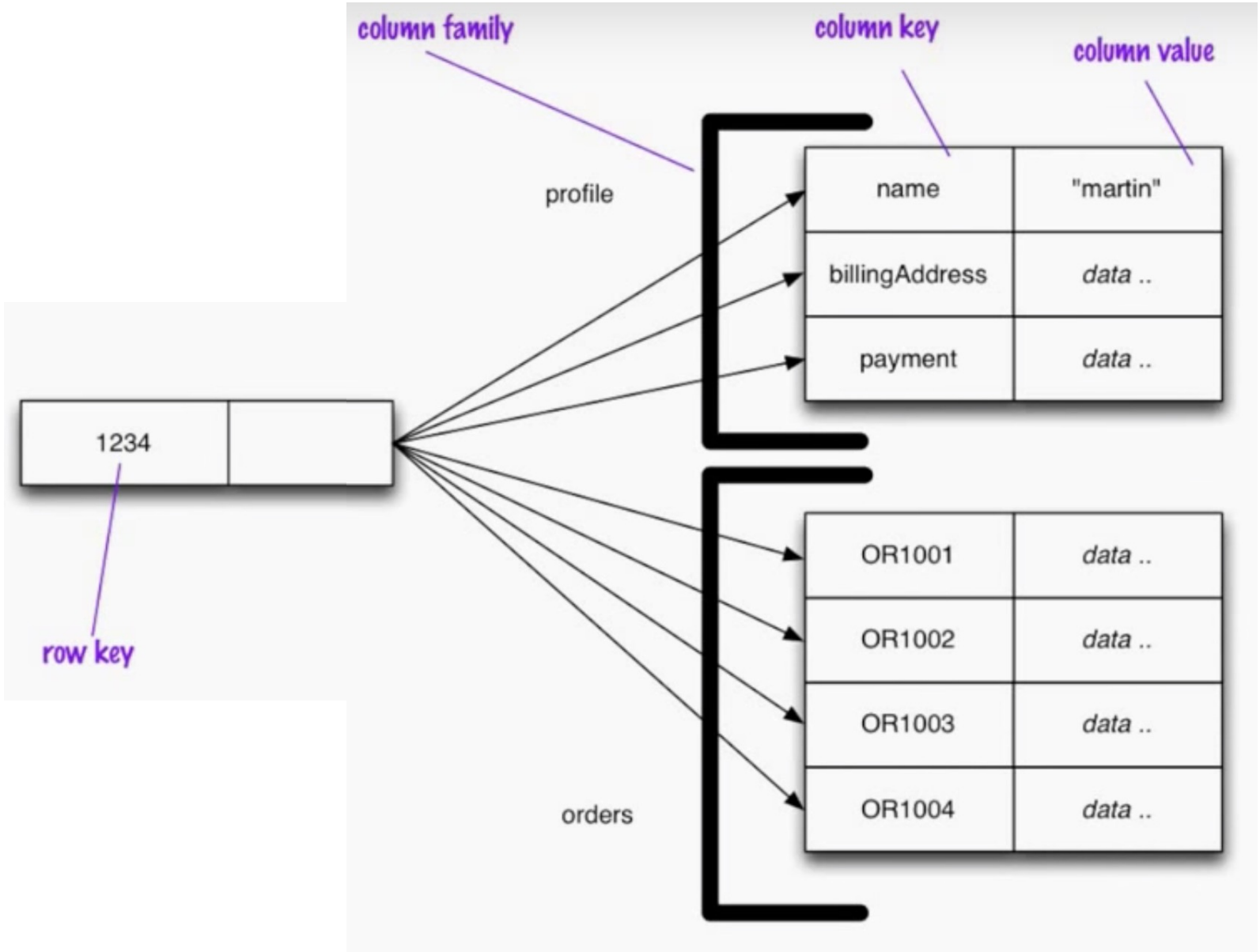

```
// in customers
{
"id":1,
"name":"Martin",
"billingAddress":[{"city":"Chicago"}]
}

// in orders
{
"id":99,
"customerId":1,
"orderItems":[
  {
  "productId":27,
  "price": 32.45,
  "productName": "NoSQL Distilled"
    }
  ],
"shippingAddress":[{"city":"Chicago"}]
"orderPayment":[
  {
    "ccinfo":"1000-1000-1000-1000",
    "txnId":"abelif879rft",
    "billingAddress": {"city": "Chicago"}
  }
  ],
}
```

Why have document databases flourished since 2008?

- They address the *impedance mismatch* problem associated with object-oriented developers needing to persist their data to relational databases. (Object-Relational Mapping)
- Self-describing formats support *ad hoc* query by value absent in pure key-value stores.
- Aligned with web-service based programming models using JSON / XML as a transport layer.
- Suitable for *horizontal scaling* in the world of big data

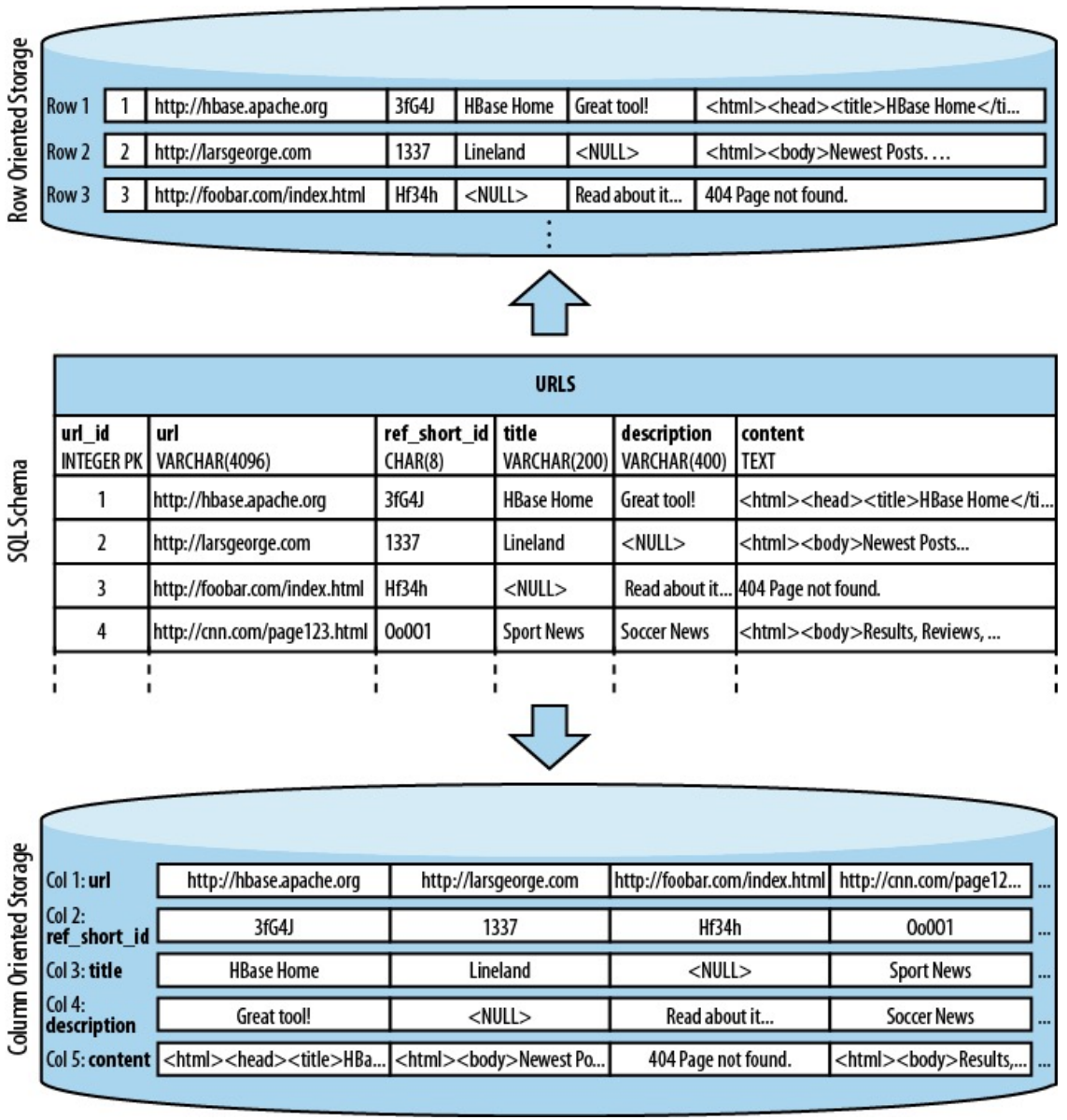
Column Family Databases

Also “aggregate”

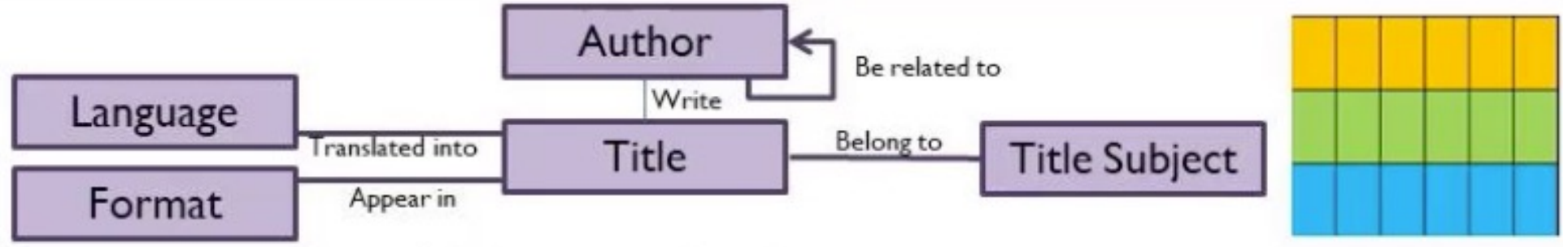


Column-Oriented Databases

- Column databases store data as columns instead rows
- More efficient for certain types of queries / analytics.
- Reduced I/O
- Better compression
- Poor performance on row-level operations (Why?)



Each column attribute is its own table.

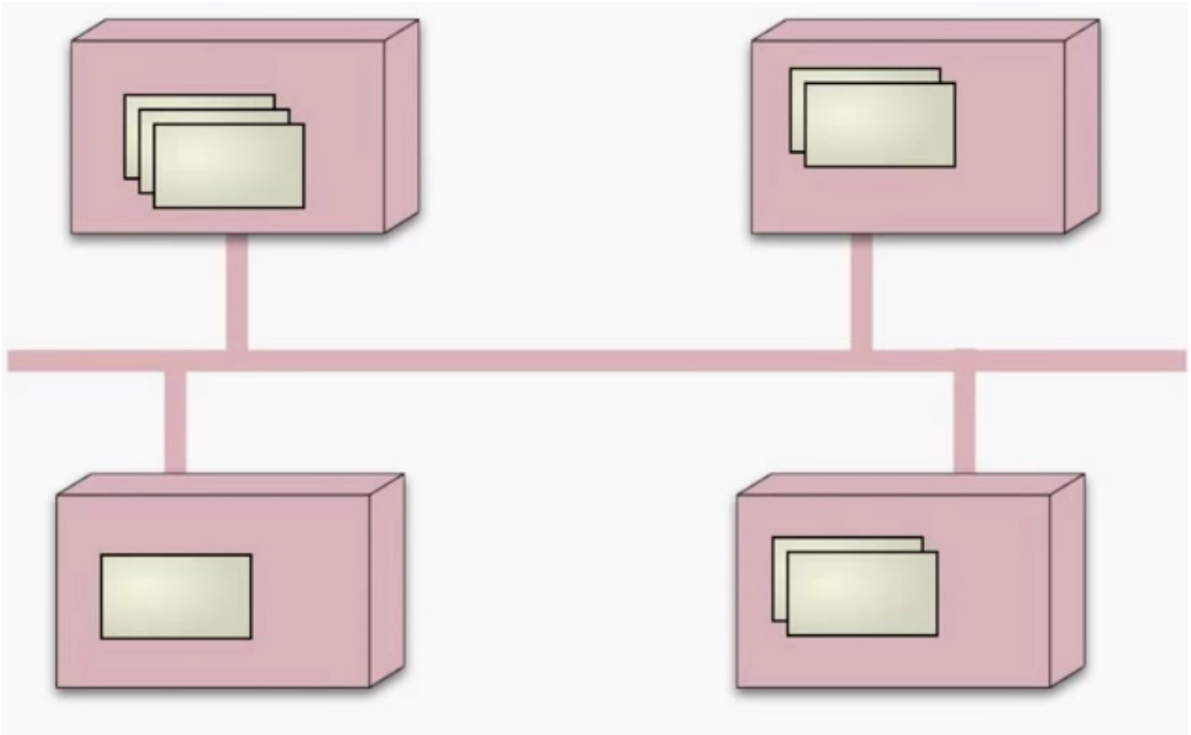


Column-oriented

9780977140060:001, 9781935504535:002, 9781935504511:003,
Data Modeling Made Simple:001, Extreme Scoping:002, UML Database Modeling Workbook:003,
Hoberman:001, Moss: 002, Blaha:003,
Steve: 001, Larissa, 002, Michael:003,
English:001,002,003,
Print Version:001,002,003,
Database Design:001,003, Data Warehousing:001,002, Data Architecture, 001,002,

Scalability of Aggregate-Oriented DBs

- A document lives on one node where all the data can be retrieved in one operation.
- Avoid cross-node joins
- So aggregate-oriented DBs work well when you want to store your data across a cluster and your operations center around particular aggregates!

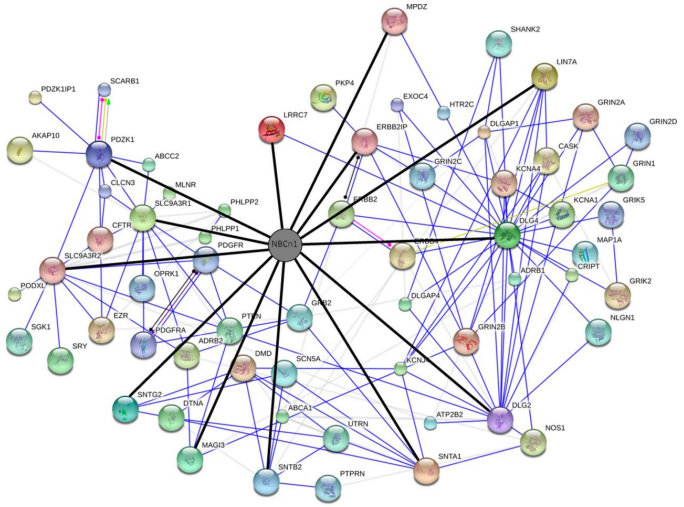


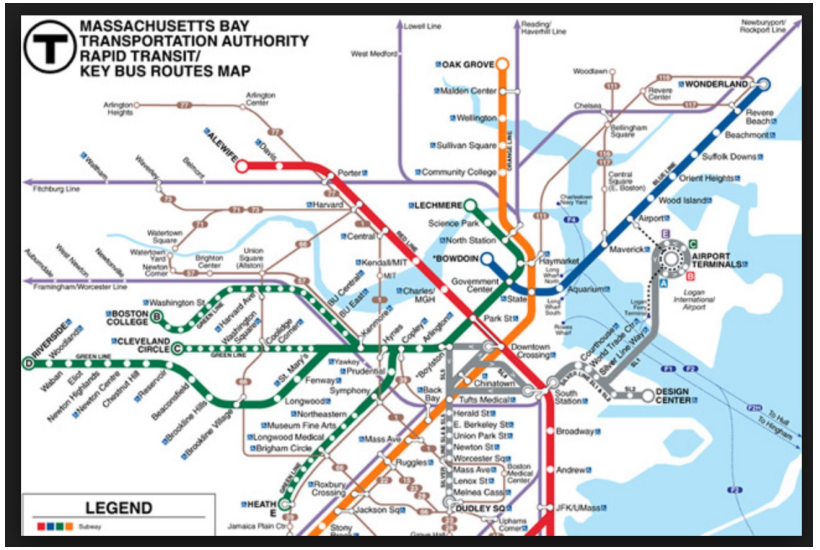
Disadvantage of aggregates

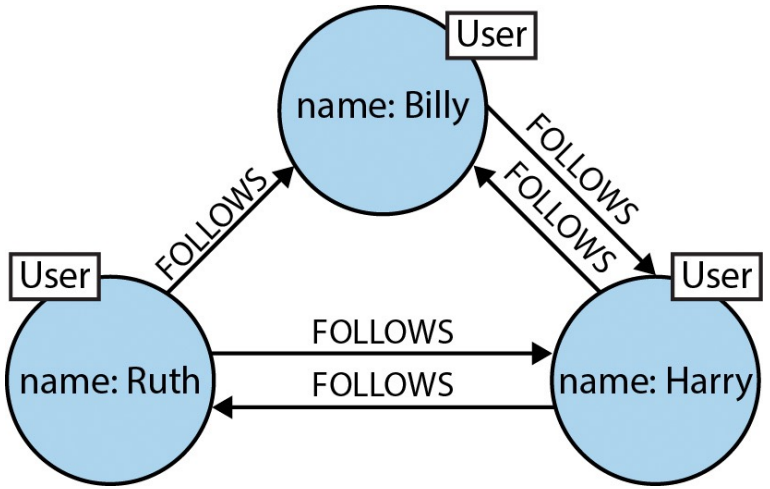
- If you want to slice and dice your data differently, say by product rather than by order, you have to re-aggregate your data so that product is the root of the aggregate.
- This can be a complex and time-consuming operation!
- With relational, it's simple! Just alter your query.

Product	revenue	prior revenue
321293533	3083	7043
321601912	5032	4782
131495054	2198	3187
...	...	...
...	...	...
...	...	...
...	...	...
...	...	...
...	...	...

Graph Databases

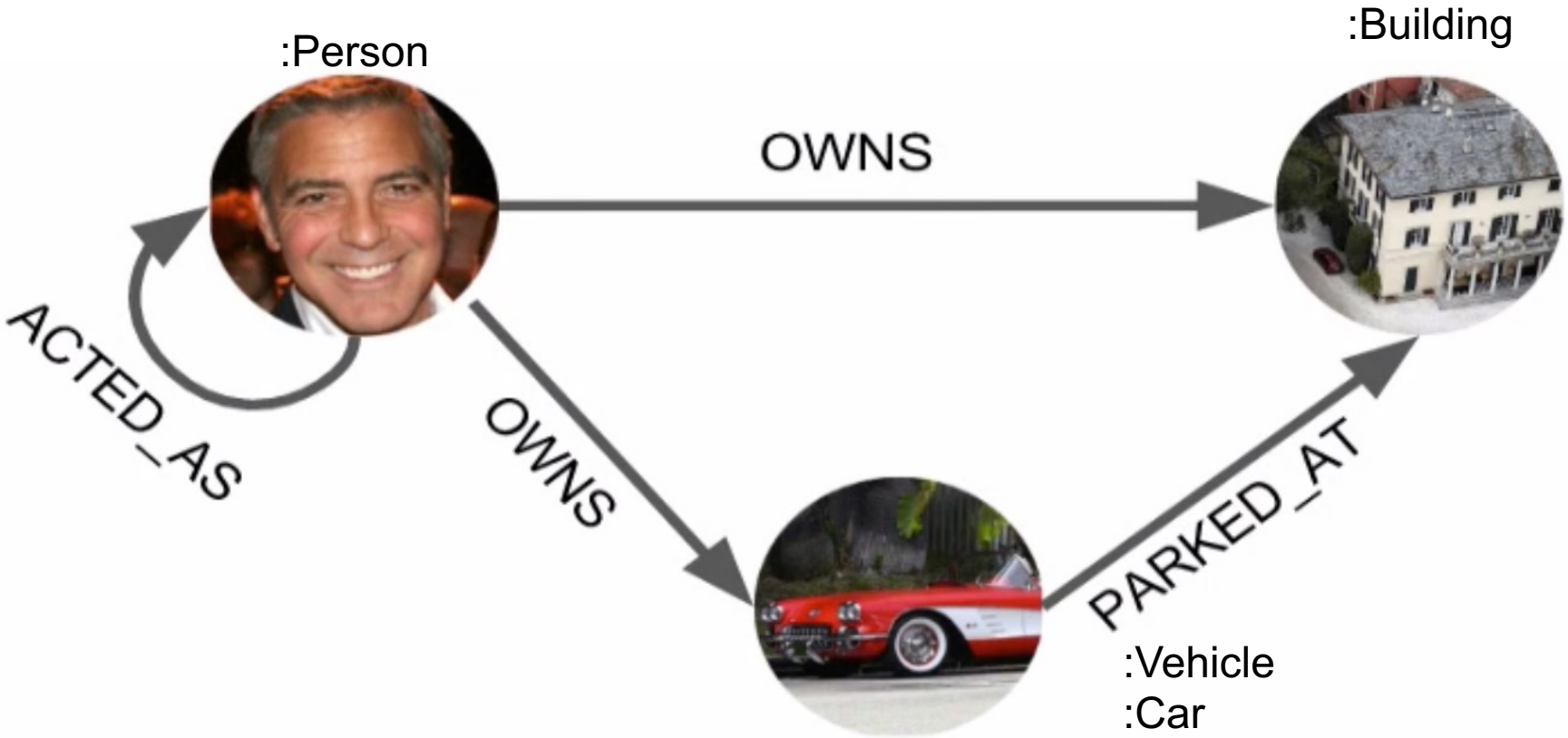






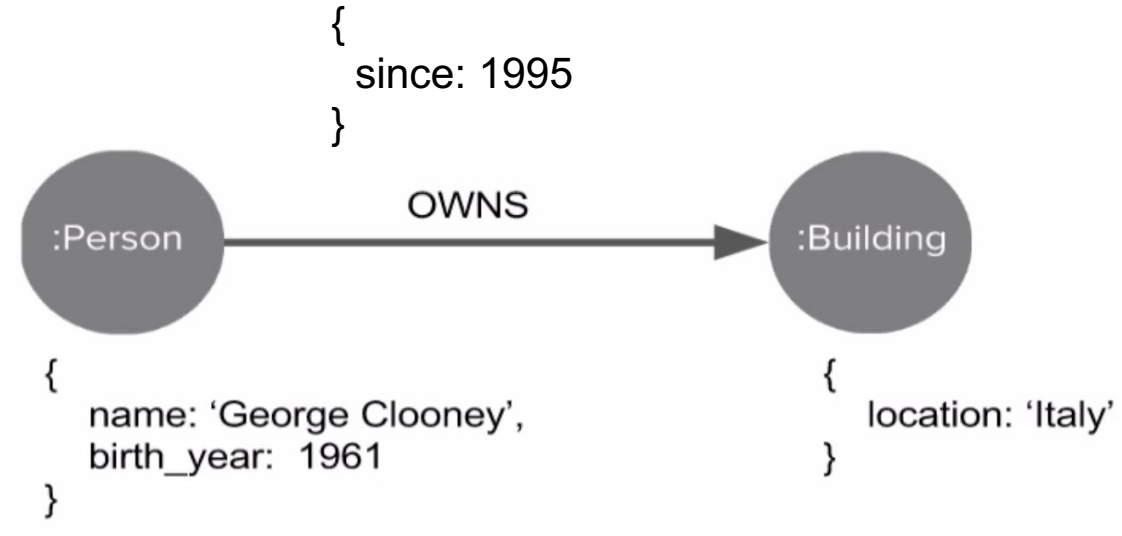
What is a Graph?

- A graph is a collection of *nodes* and *relationships.*
- Nodes have one or more categorical labels and represent things
- Relationships have a single type and are a connected arc between two nodes



Graph properties

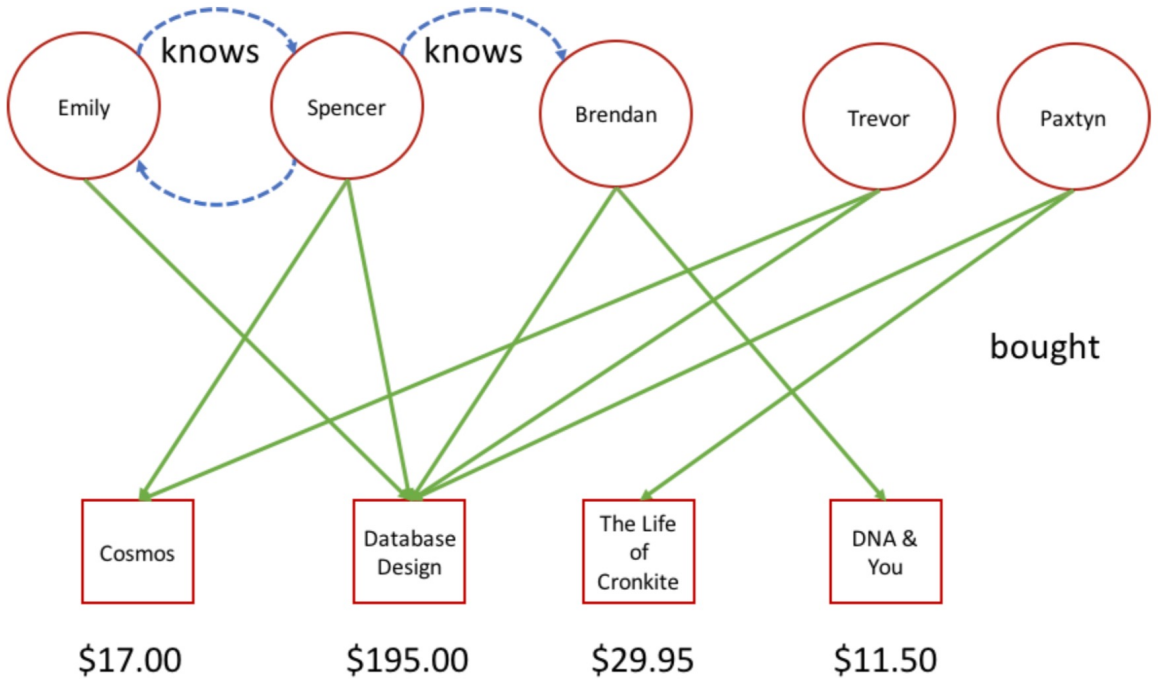
- Properties are key value pairs that are assigned to a node or a relationship
- Nodes have one or more categorical labels and represent things
- Relationships have a single type and are a connected arc between two nodes



Value Types

Type	Example
Boolean	true, false
Text	"AKA string"
Numbers	123, 56.70
Lists	['must', 'be', 'same', 'type']

Modeling graphs in a relational database is ugly!



SQL:

```
-- What books did spencer buy?  Give title and price.
select  npout.string_value title, npout2.num_value price
from edge e
    join node nin on (e.in_node = nin.node_id)
    join node_props npin on (nin.node_id = npin.node_id)
    join node nout on (e.out_node = nout.node_id)
    join node_props npout on (nout.node_id = npout.node_id)
    join node_props npout2 on (nout.node_id = npout2.node_id)
where e.type = 'bought'
and npin.propkey = 'name'
and npin.string_value = 'Spencer'
and npout.propkey = 'title'
and npout2.propkey = 'price'
order by price desc;
```

Cypher (Neo4J):

```
match (e:Person {name:'Spencer'})-[:Buys]->(x:Book) return x.title, x.price
```

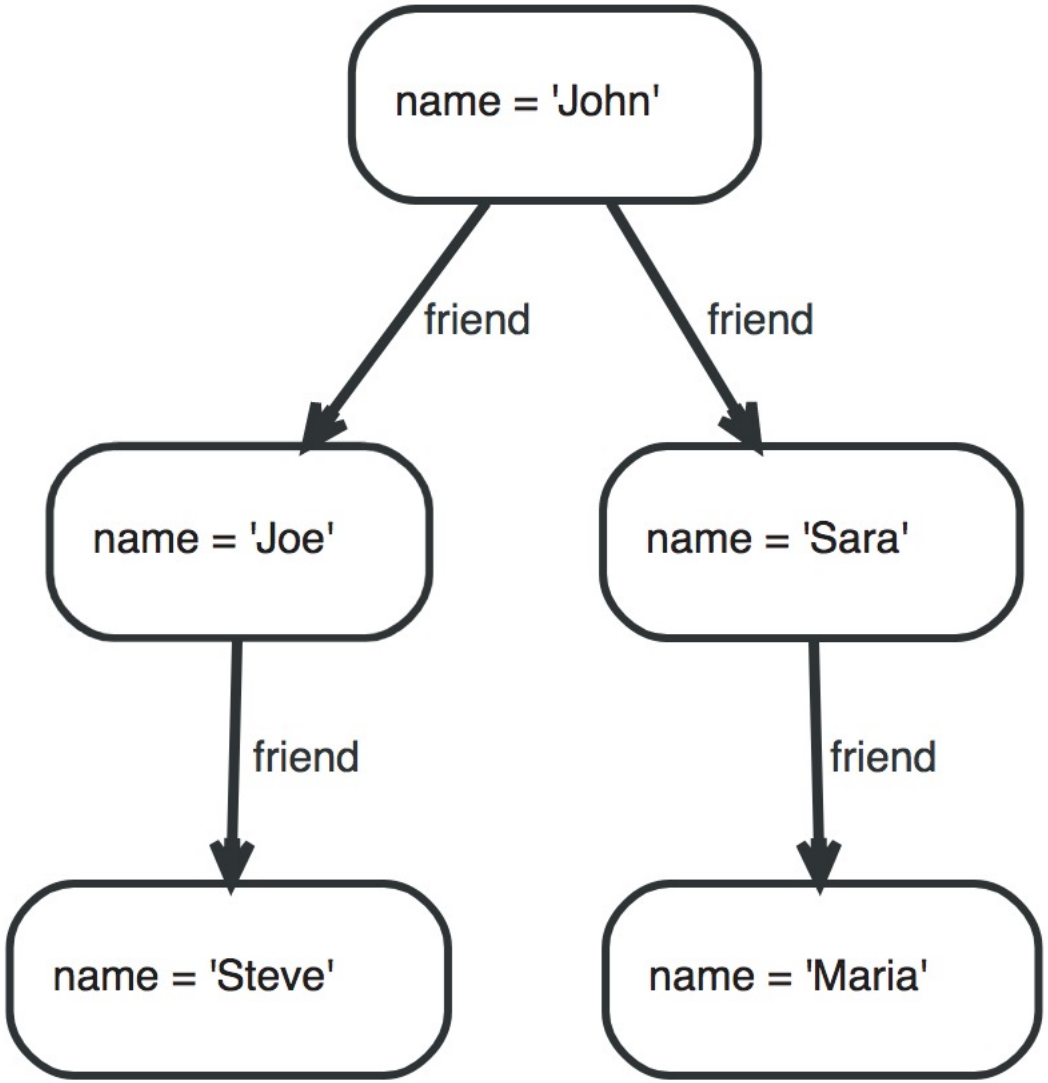
Reasoning about connected knowledge

For example, here is a query which finds a user called **'John'** and **'John's'** friends (though not his direct friends) before returning both **'John'** and any friends-of-friends that are found.

```
MATCH (john {name: 'John'})-[:friend]->()-[:friend]->(fof)
RETURN john.name, fof.name
```

Resulting in:

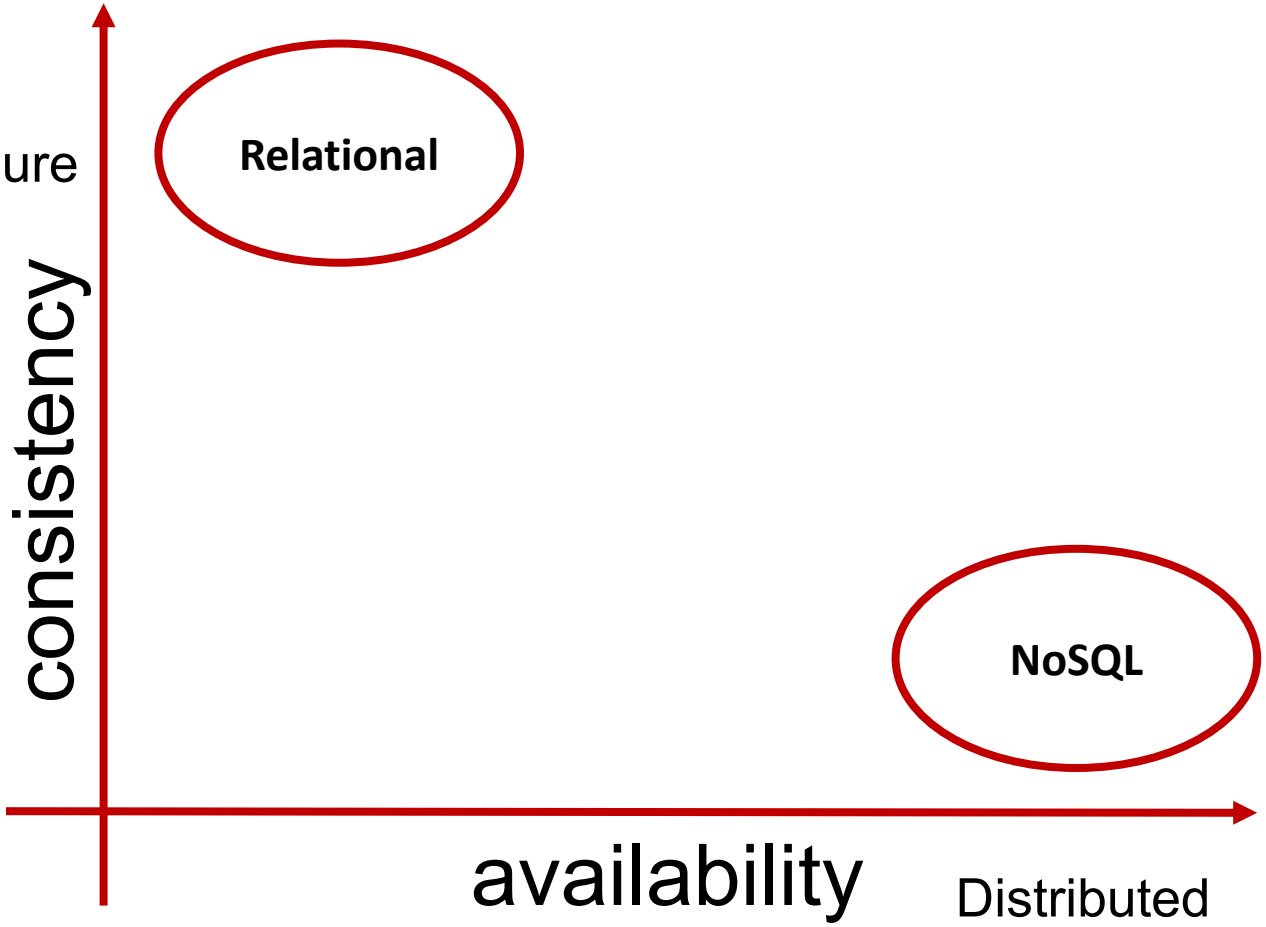
```
+--------------------+
| john.name | fof.name |
+--------------------+
| "John"    | "Maria"  |
| "John"    | "Steve"  |
+--------------------+
2 rows
```



Downside: Difficult to scale

Relational vs. NoSQL

Centralized
Strict consistency
Transactional
Single Point of Failure



Distributed
Highly available
Highly resilient
Weak or Eventual Consistency

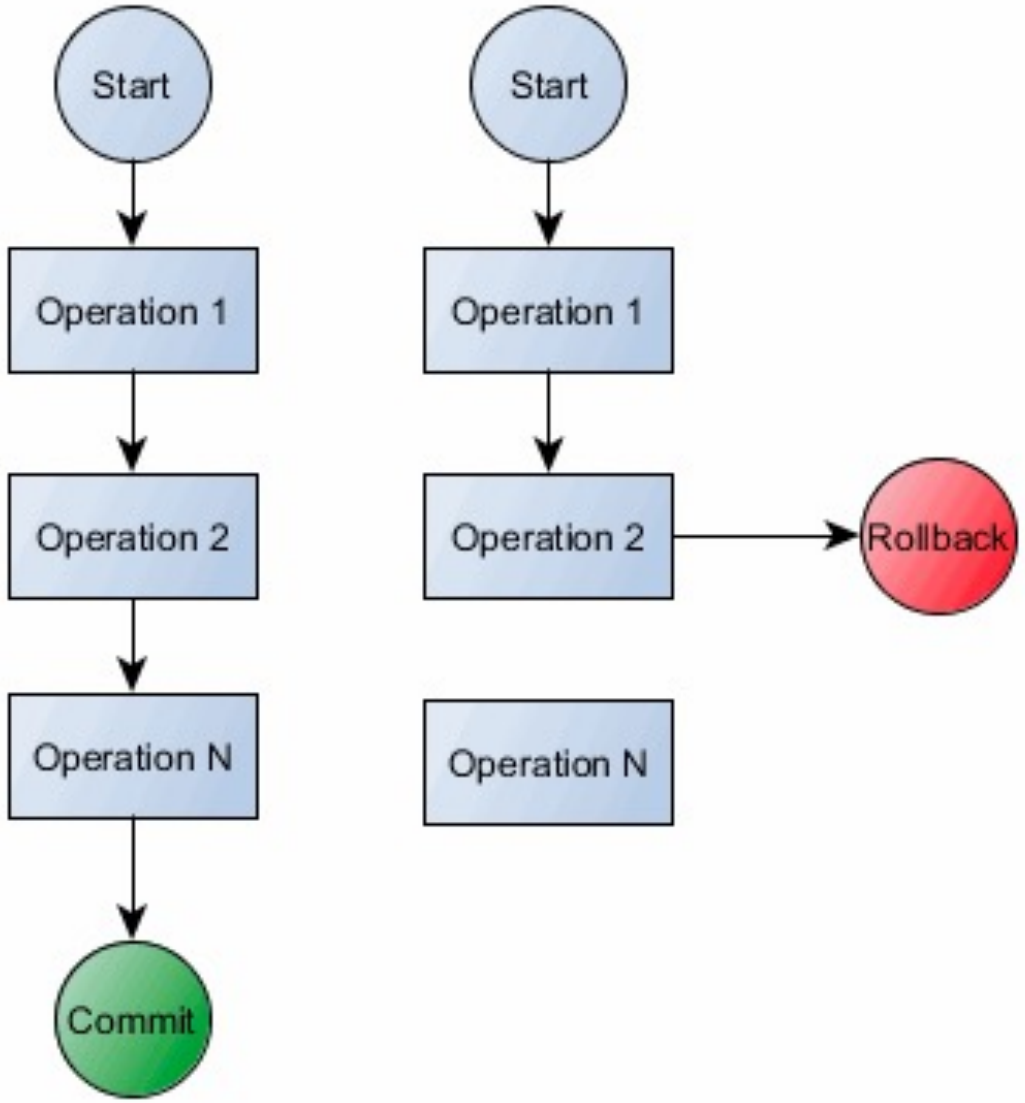
ACID Properties

Atomicity: Transactions are *all or nothing*: They either succeeds or fails as a whole.

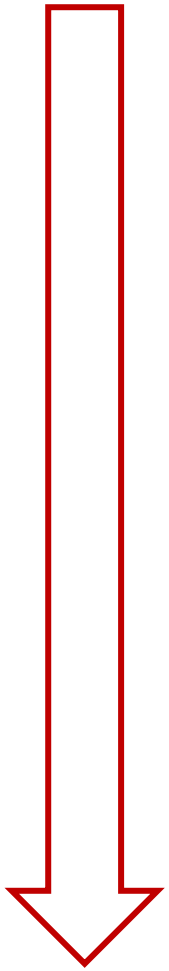
Consistency: Transactions on a database must transform the database from one consistent state to another.

Isolation: Partial effects of incomplete transactions should not be visible to other transactions.

Durability: Effects of a committed transaction are permanent and must not be lost because of later failure.



Transfer $50 from account A to account B



Read (A)

A = A – 50

Write (A)

Read (B)

B = B + 50

Write (B)

Atomicity: Don’t take money from A without giving to B

Consistency: No money should be lost or gained

Isolation: Other queries shouldn’t see A or B change until the entire transaction is completed and both values are updated.

Durability: The money does not go back to A

BASE (An admittedly contrived acronym!)

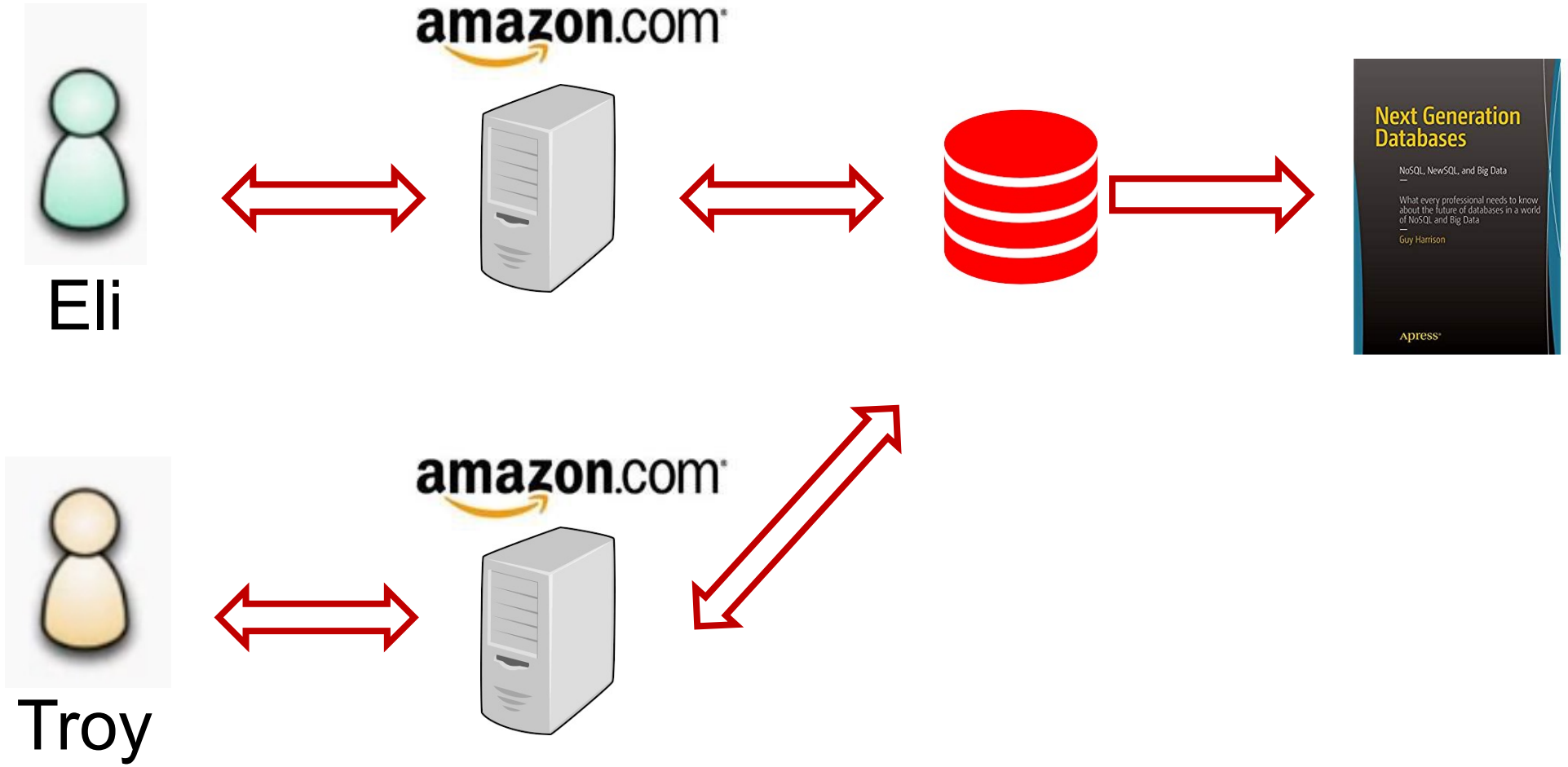
RDBMS → "ACID"
NoSQL → "BASE"

- **B**asically **A**vailable: A partial failure in a distributed system (e.g., one node) doesn't kill the whole system. If each bit of data is replicated to multiple nodes, system can continue unabated.

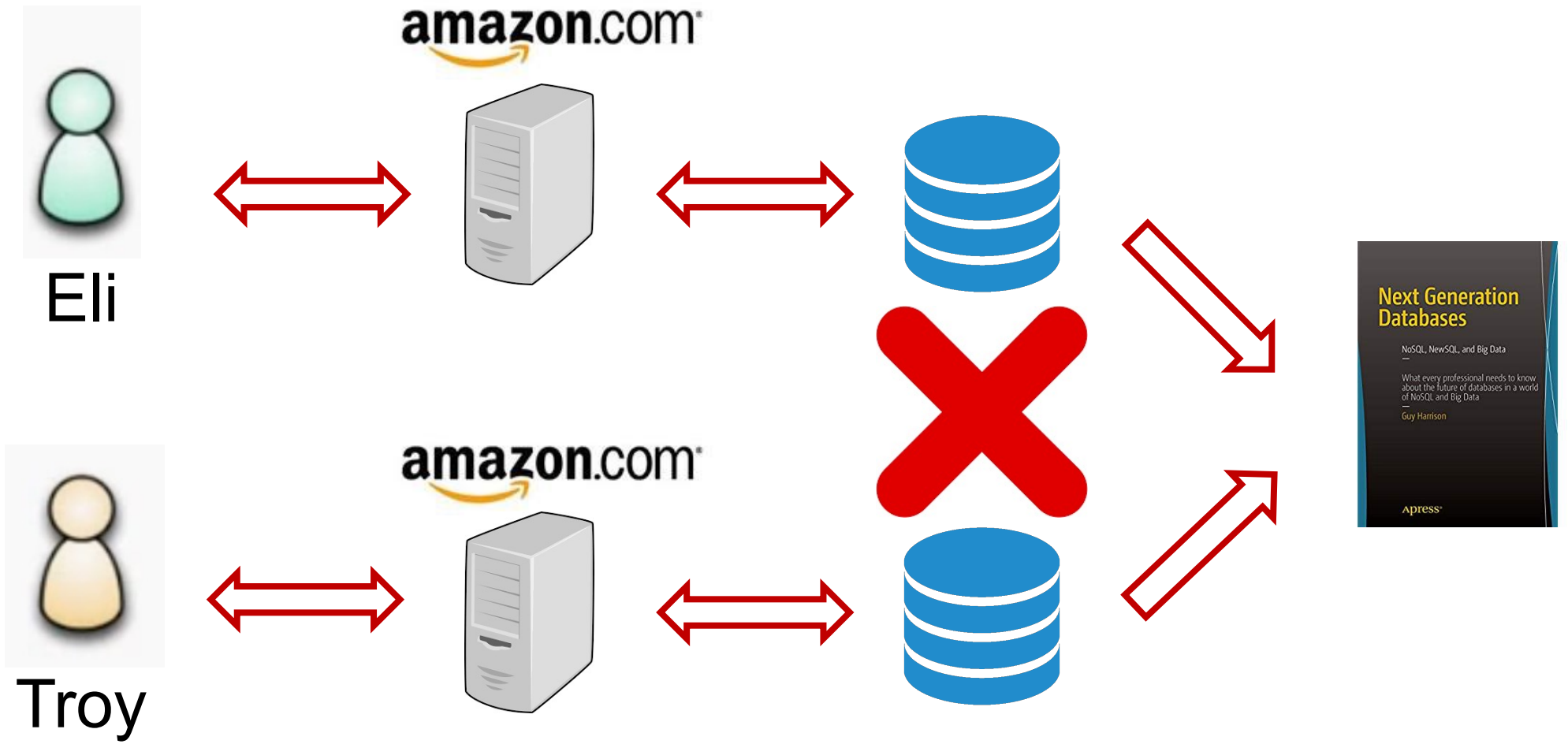
- **S**oft state: Data may be over-written with more recent data

- **E**ventually consistent. There may be times when the database is in an inconsistent state, particularly when data is replicated to multiple nodes for better availability.

Consistency



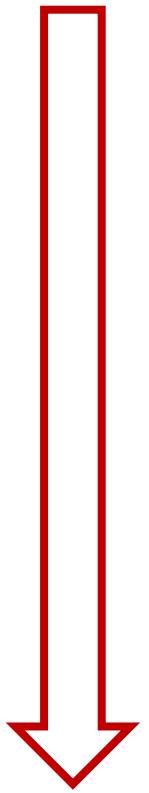
Consistency vs. Availability



Levels of Consistency

Database consistency is concerned with guaranteeing that a database always appears truthful to the user. Every operation must take the database from one consistent state to another.

- **Strict**: Changes to data are atomic and appear instantaneous
- **Sequential**: Clients see changes in the same order they were applied
- **Causal**: All causally-related changes are observed in the same order by all clients
- **Eventual**: All updates will eventually propagate through the system and all replicas will be consistent
- **Weak**: No propagation guarantees. Changes may appear out of order for some clients



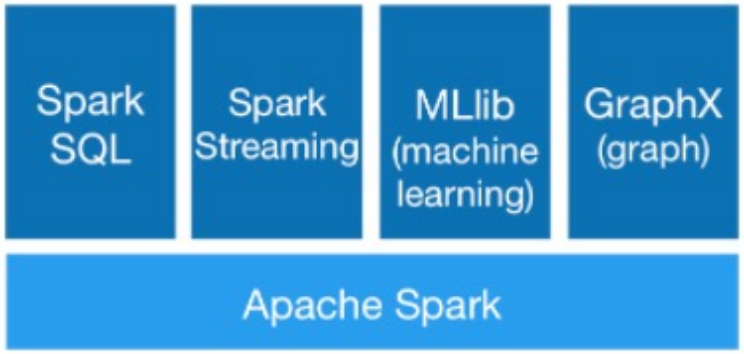
Drivers for the adoption of NoSQL

- The rise of Big Data and Cluster Computing coupled with the inherent limits of the relational model
- Object-Relational mapping challenges (The *Impedance Mismatch* problem)
- Simplified storage and analytics of semi-structured data

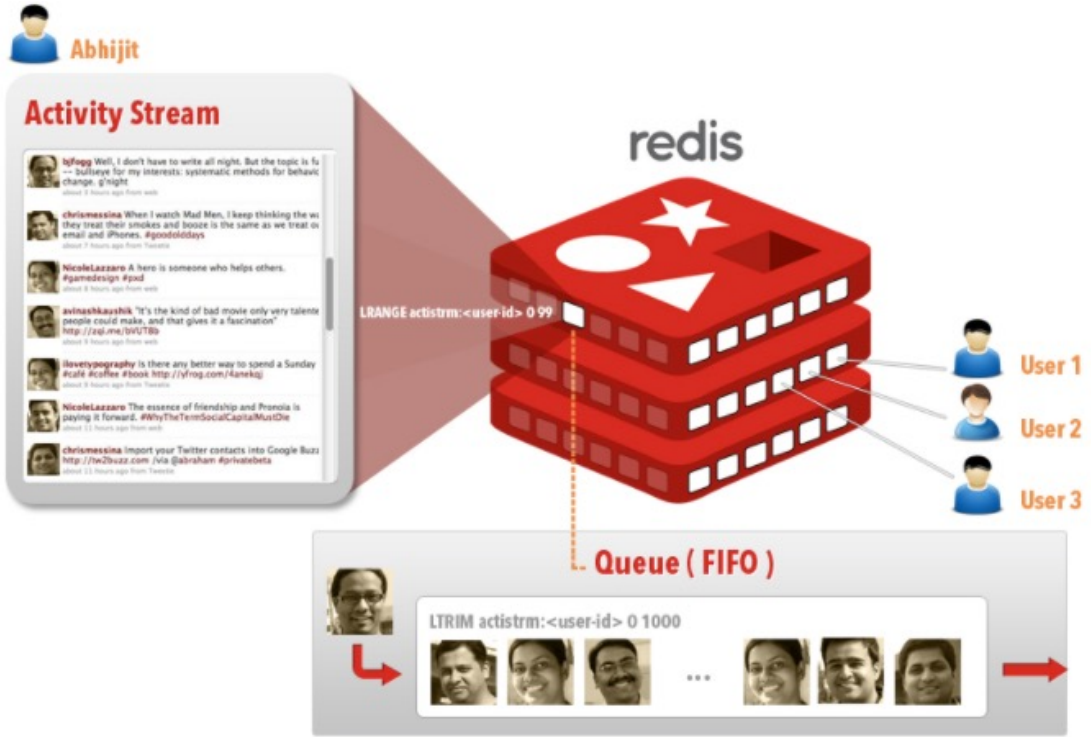
(But relational isn't going away!)

Overlapping capabilities: Some examples

Apache Spark: A batch-processing engine with libraries for SQL-like operations, streaming, ML, and graph processing

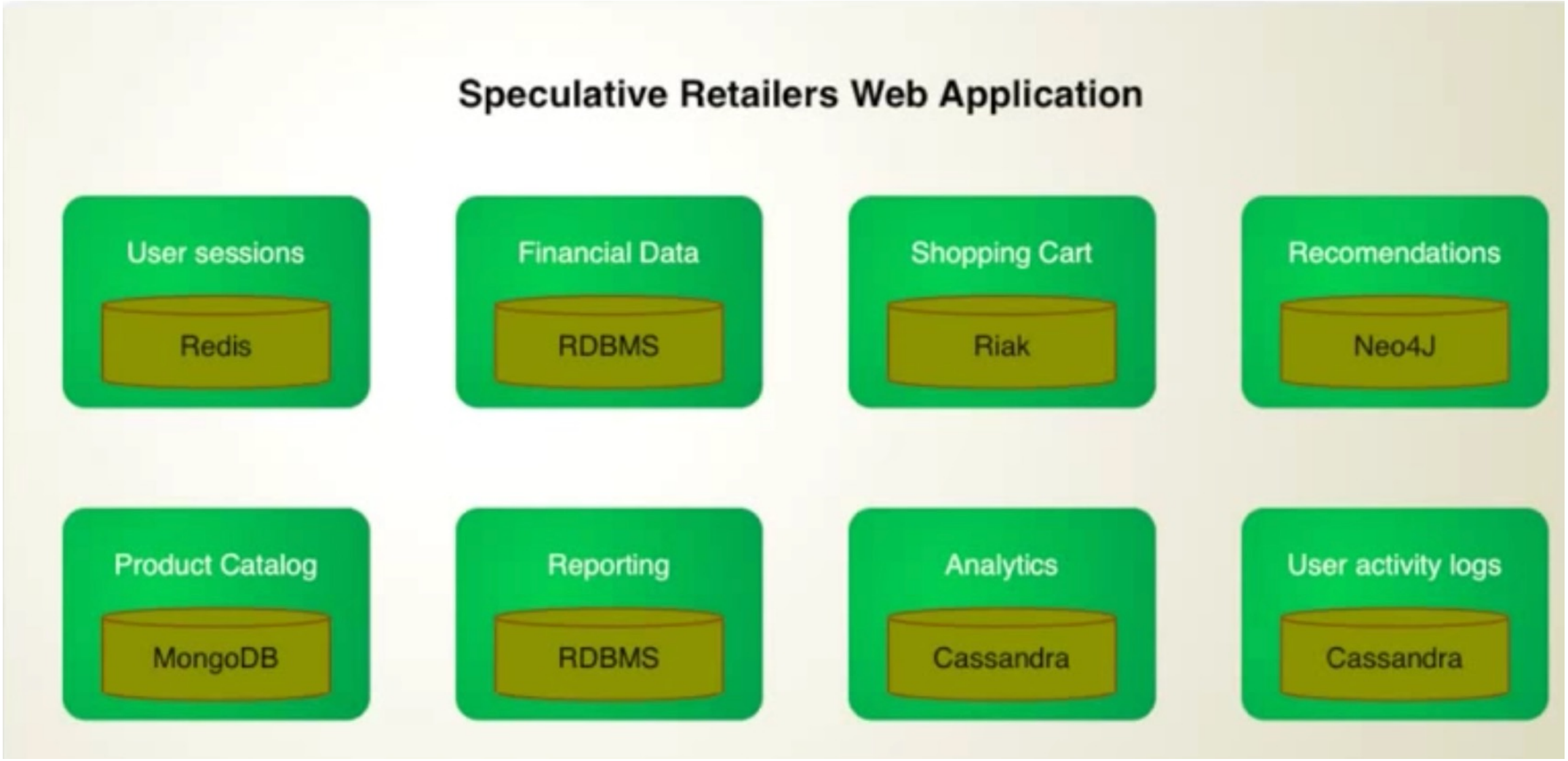


Redis: A key-value store but which can be used for stream-processing / event-processing and message-passing.



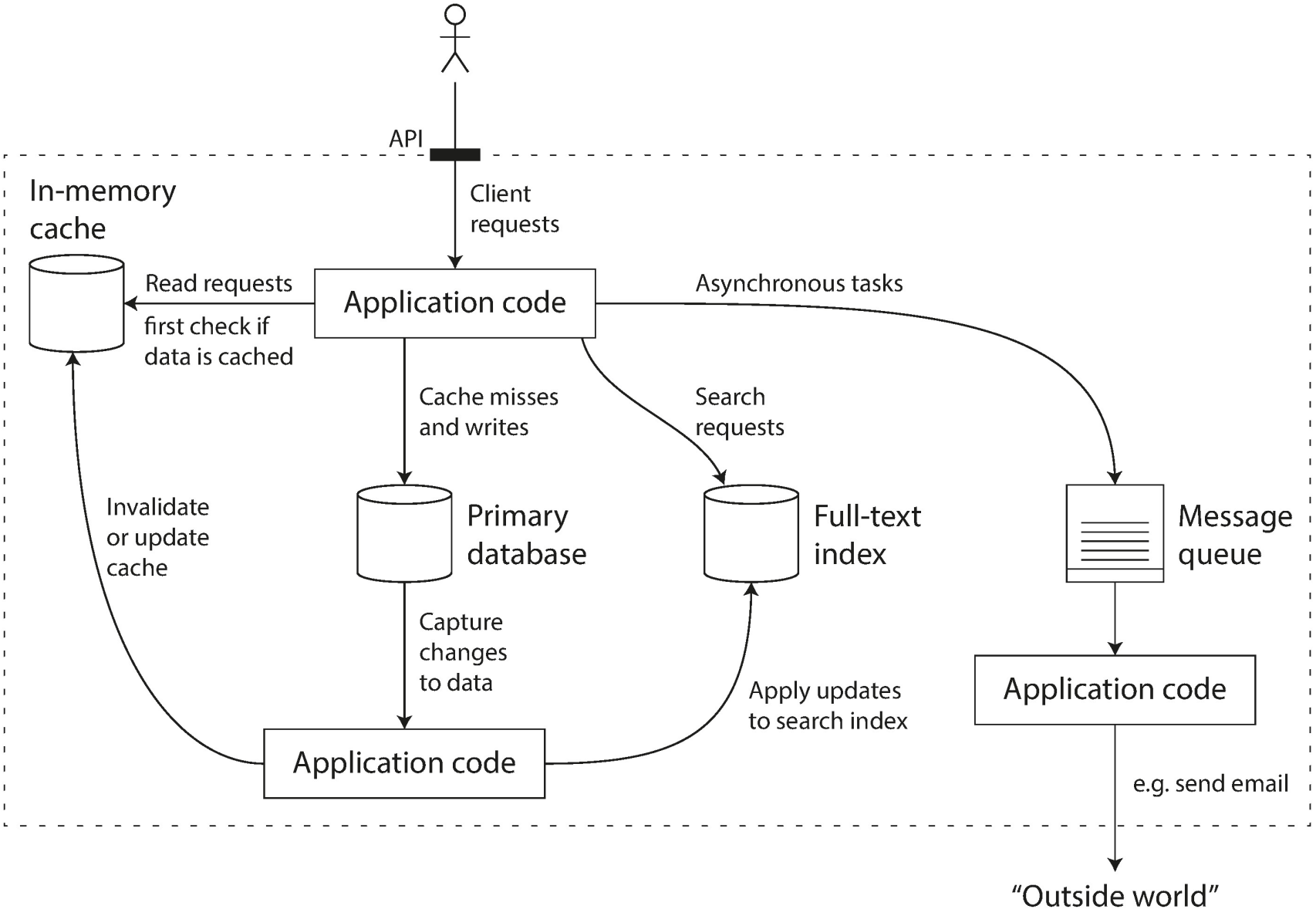
Oracle NoSQL: Document storage (JSON-format) but ith SQL-like querying and analytics.

Polyglot Persistence



Polyglot Persistence

- Many tools optimized for different use-cases. No single tool accommodates all situations. *"One size does not fit all."*
- Different components are stitched together by application software.



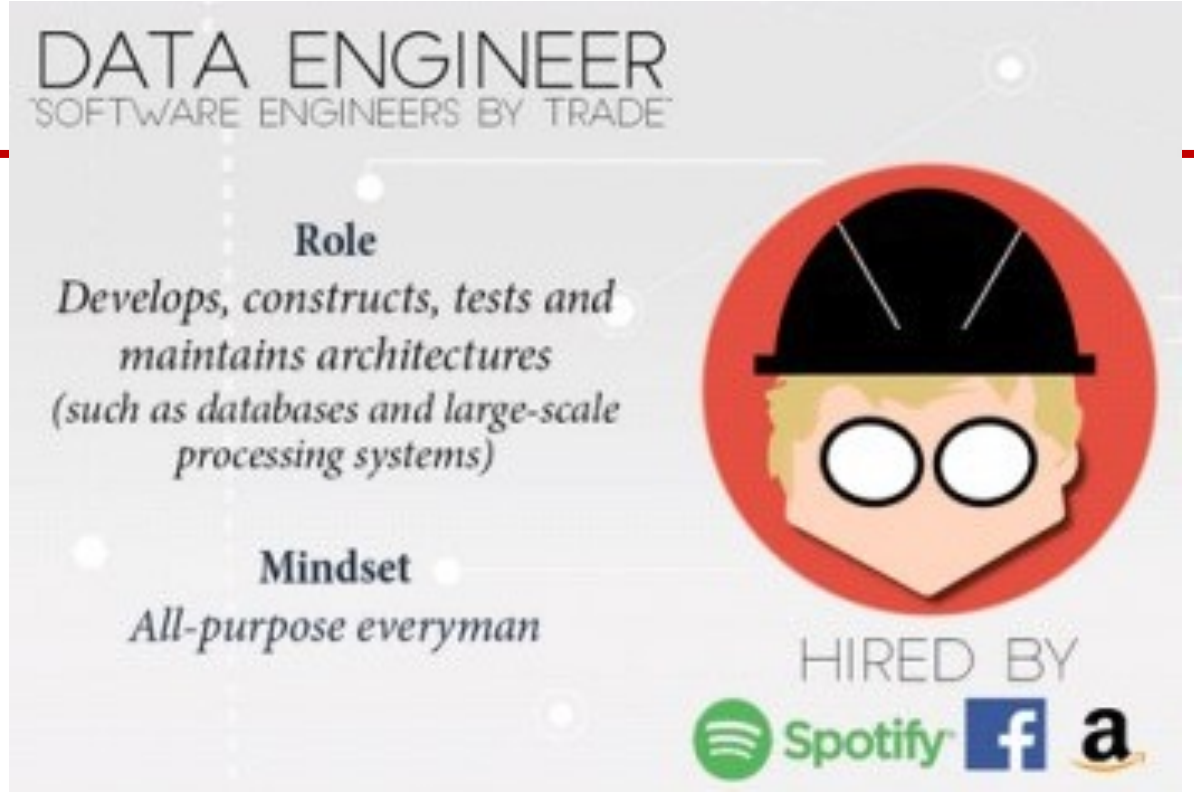
The database landscape

Resembles a Tower of Babel with many systems (large and small) interoperating together.

- Relational DBMS
- NoSQL for special use-cases
- Caching with Key-Value stores
- Messaging brokers for asynchronous communication & coordination / Streaming
- Batch processing with Hadoop or Spark

The rise of data engineering

- Many applications are *data-intensive* rather (merely) than *compute-intensive.*
- CPUs are not the only limiting factor, but also the size (volume), flow rate (velocity), and complexity (variety) of the data.
- This has given rise to the field of *data engineering* (and demand for data-engineers rather than just software engineers.)



and fitbit too!

fitbit

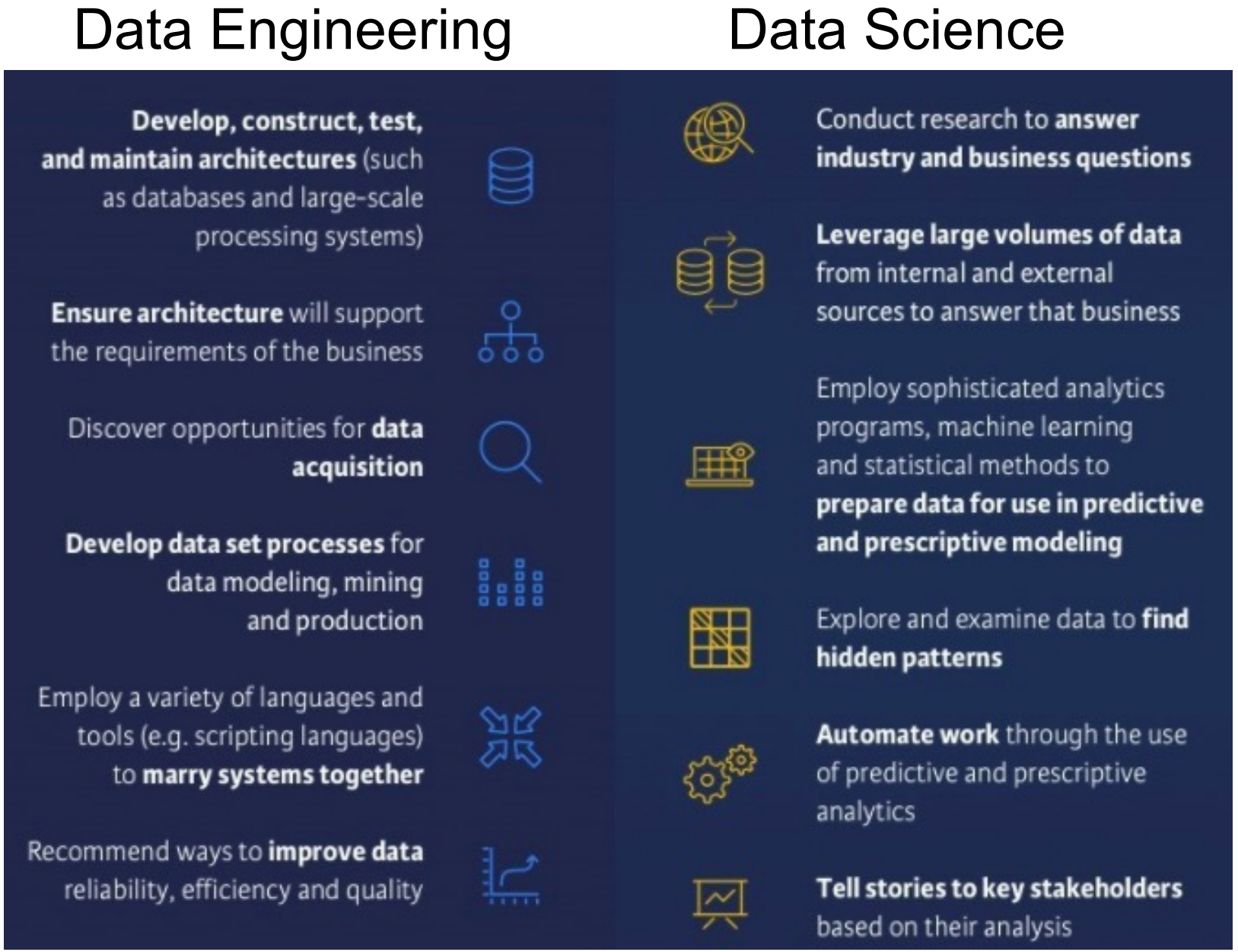
Elements of a data-centric application

- **Databases:** data storage and retrieval (Relational and NoSQL)
- **Caching**: Remembering the result of an expensive operation to speed up reads
- **Indexing**: Allow users to search data by keyword or filter it in various ways
- **Continuous (Stream) Processing**: Asynchronous data flow programming for real-time data
- **Batch**: Periodically crunching large amounts of accumulated data

Data Science vs. Data Engineering

Data Scientist: Performs descriptive statistics to develop insights, build models, solve a business need.

Data Engineer: Architect large scale data processing systems typically using relational, NoSQL, messaging, and distributed computing platforms.



Diverse technologies with blurred boundaries