NoSQL Overview

Relational (SQL) Databases

- All about relations (tables)
  - Schema is fixed for each table
- … and transactions
  - Consistency across table updates
- Use SQL for querying
- Still the most popular on the current DB market:
  - The top 4 DBMS: Oracle, MySQL, Microsoft SQL Server, PostgreSQL
  - 7 of the top 10 DBMS are relational

DB Engines Ranking

364 systems in ranking, March 2021

Rank Mar 2021	Rank Feb 2021	Rank Mar 2020	DBMS	Database Model	Score Mar 2021	Score Feb 2021	Score Mar 2020
1.	1.	1.	Oracle	Relational, Multi-model	1321.73	+5.06	-18.91
2.	2.	2.	MySQL	Relational, Multi-model	1254.83	+11.46	-4.90
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1015.30	-7.63	-82.55
4.	4.	4.	PostgreSQL	Relational, Multi-model	549.29	-1.67	+35.37
5.	5.	5.	MongoDB	Document, Multi-model	462.39	+3.44	+24.78
6.	6.	6.	IBM Db2	Relational, Multi-model	156.01	-1.60	-6.55
7.	7.	↑ 8.	Redis	Key-value, Multi-model	154.15	+1.58	+6.57
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model	152.34	+1.34	+3.17
9.	9.	↑ 10.	SQLite	Relational	122.64	-0.53	+0.69
10.	↑ 11.	↓ 9.	Microsoft Access	Relational	118.14	+3.97	-7.00
11.	↓ 10.	11.	Cassandra	Wide column	113.63	-0.99	-7.32
12.	12.	↑ 13.	MariaDB	Relational, Multi-model	94.45	+0.56	+6.10
13.	13.	↓ 12.	Splunk	Search engine	86.93	-1.61	-1.59
14.	14.	14.	Hive	Relational	76.04	+3.72	-9.34
15.	↑ 16.	15.	Teradata	Relational, Multi-model	71.43	+0.53	-6.41

[Source: db-engines.com]

Popular Open-Source RDBMS

- SQLite
  - A full-featured embedded database, can be stored in a single file
  - For example, Skype uses SQLite to store chat history
- MySQL
  - The most popular and common used open-source RDBMS, particularly in web-based solutions
- PostgreSQL
  - The most advanced, SQL-compliant and open-source RDBMS with a well-known GeoSpatial extenstion, PostGIS (Carto)

ACID Properties of RDBMS

- **A**tomic – All of the work in a transaction completes (commit) or none of it completes
- **C**onsistent – A transaction transforms the database from one consistent state to another consistent state, i.e. data must always meet all validation rules regardless of any failure.
- **I**solated – The results of any changes made during a transaction are not visible until the transaction has committed.
- **D**urable – The results of a committed transaction survive failures

RDBMS and Big Data

- It is expensive to maintain ACID properties
- Parallelism often relies on replication
  - Query response time can be scale-out effectively
  - But capacity is limited by the ability to scale-up
- Scale-out RDBMS requires s*harding* (horizontal partitioning)
  - Similar to how we partition/shuffle data with MapReduce/Spark

Original Table

CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
1	TAEKO	OHNUKI	BLUE
2	O.V.	WRIGHT	GREEN
3	SELDA	BAĞCAN	PURPLE
4	JIM	PEPPER	AUBERGINE

Vertical Partitions

VP1

CUSTOMER ID	FIRST NAME	LAST NAME
1	TAEKO	OHNUKI
2	O.V.	WRIGHT
3	SELDA	BAĞCAN
4	JIM	PEPPER

VP2

CUSTOMER ID	FAVORITE COLOR
1	BLUE
2	GREEN
3	PURPLE
4	AUBERGINE

Horizontal Partitions

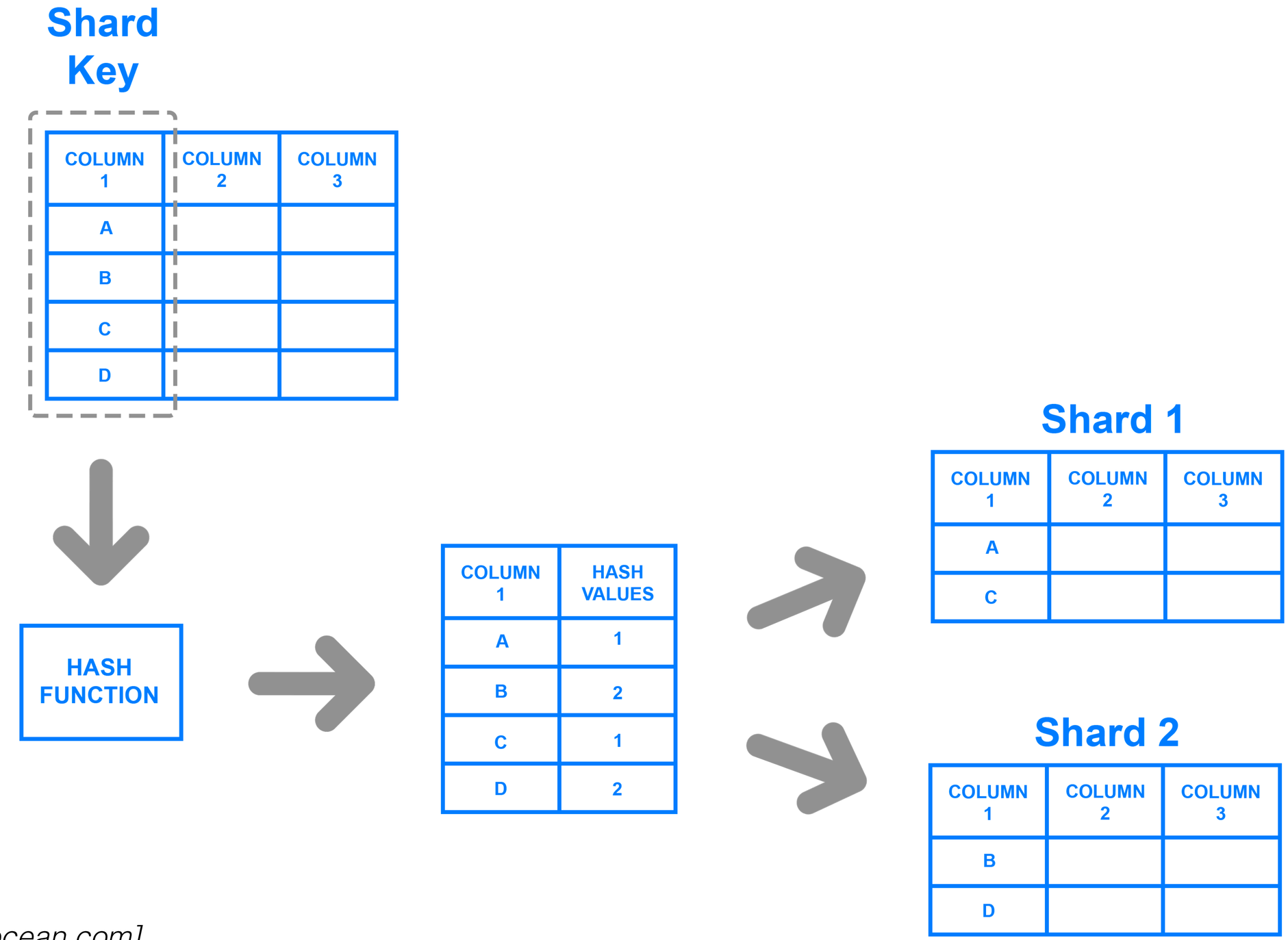
HP1

CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
1	TAEKO	OHNUKI	BLUE
2	O.V.	WRIGHT	GREEN

HP2

CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
3	SELDA	BAĞCAN	PURPLE
4	JIM	PEPPER	AUBERGINE

[Source: digitalocean.com]



[Source: digitalocean.com]

Hash Partitioner!

Shard Key

COLUMN 1	COLUMN 2	COLUMN 3
A		
B		
C		
D		

HASH FUNCTION

COLUMN 1	HASH VALUES
A	1
B	2
C	1
D	2

Shard 1

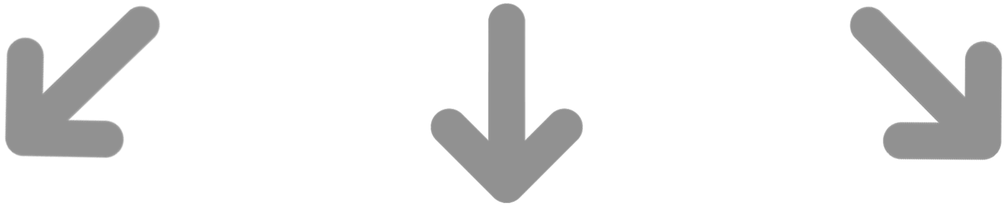
COLUMN 1	COLUMN 2	COLUMN 3
A		
C		

Shard 2

COLUMN 1	COLUMN 2	COLUMN 3
B		
D		

[Source: digitalocean.com]

PRODUCT	PRICE
WIDGET	$118
GIZMO	$88
TRINKET	$37
THINGAMAJIG	$18
DOODAD	$60
TCHOTCHKE	$999



($0-$49.99)

PRODUCT	PRICE
TRINKET	$37
THINGAMAJIG	$18

($50-$99.99)

PRODUCT	PRICE
GIZMO	$88
DOODAD	$60

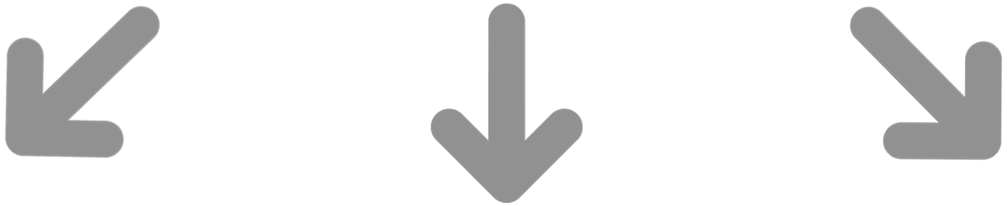
($100+)

PRODUCT	PRICE
WIDGET	$118
TCHOTCHKE	$999

[Source: digitalocean.com]

Range Partitioner!

PRODUCT	PRICE
WIDGET	$118
GIZMO	$88
TRINKET	$37
THINGAMAJIG	$18
DOODAD	$60
TCHOTCHKE	$999



($0-$49.99)

PRODUCT	PRICE
TRINKET	$37
THINGAMAJIG	$18

($50-$99.99)

PRODUCT	PRICE
GIZMO	$88
DOODAD	$60

($100+)

PRODUCT	PRICE
WIDGET	$118
TCHOTCHKE	$999

[Source: digitalocean.com]

Pre-Sharded Table

Shard Key

DELIVERY ZONE	FIRST NAME	LAST NAME
3	DARCY	CLAY
1	DENISE	LASALLE
2	HIROSHI	YOSHIMURA
4	KIRSTY	MACCOLL

Shard Key

DELIVERY ZONE	SHARD ID
1	S1
2	S2
3	S3
4	S4

S1

1	DENISE	LASALLE

S2

2	HIROSHI	YOSHIMURA

S3

3	DARCY	CLAY

S4

4	KIRSTY	MACCOLL

[Source: digitalocean.com]

Pre-Sharded Table

Directory Partitioner!

Shard Key

DELIVERY ZONE	FIRST NAME	LAST NAME
3	DARCY	CLAY
1	DENISE	LASALLE
2	HIROSHI	YOSHIMURA
4	KIRSTY	MACCOLL

Shard Key

DELIVERY ZONE	SHARD ID
1	S1
2	S2
3	S3
4	S4

S1

1	DENISE	LASALLE

S2

2	HIROSHI	YOSHIMURA

S3

3	DARCY	CLAY

S4

4	KIRSTY	MACCOLL

[Source: digitalocean.com]

Drawbacks of Sharding in RDBMS

- Non-trivial to configure and manage
  - Not supported by default in many RDBMS
  - Users have to manually partition their tables
- May lead to data imbalance
- Partitioning is static, data are tied to machines/server
  - Difficult to repartition

Drawbacks of Sharding in RDBMS

- Non-trivial to configure and manage
  - Not supported by default in many RDBMS
  - Users have to manually partition their tables
- May lead to data imbalance
- Partitioning is static, data are tied to machines/server
  - Difficult to repartition

Many of these issues are addressed by Big Data DBMS

What is NoSQL?

- **N**ot **o**nly **SQL**, or **No**t **SQL**
- Core data model is non-relational
  - Usually do not require a fixed table schema, or the schema can be loosely defined
- Has to relax one of the ACID properties for performance and scalability
  - Aim for big data platforms and distributed architecture

CAP Theorem

- “it is impossible for a distributed computer system to simultaneously provide more than two out of three of the following guarantees” (distributed >> data replication)
  - *Consistency*: Every read receives the most recent write or an error.
  - *Availability:* Every request receives a (non-error) response – without guarantee that it contains the most recent write.
  - Partition tolerance: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes
- To scale out, we must have partition tolerance (the last property)
  - We must choose between Consistency and Availability

Consistency Model

Consistency Model

- A consistency model determines rules for visibility, and to favor availability or consistency

Consistency Model

- A consistency model determines rules for visibility, and to favor availability or consistency
- Consider this example:
  - Row X is replicated on nodes M and N
  - Client A writes row X to node N
  - Some period of time t elapses
  - Client B reads row X from node M

Consistency Model

- A consistency model determines rules for visibility, and to favor availability or consistency
- Consider this example:
  - Row X is replicated on nodes M and N
  - Client A writes row X to node N
  - Some period of time t elapses
  - Client B reads row X from node M
- Does client B see the write from client A?

Consistency Model

- A consistency model determines rules for visibility, and to favor availability or consistency
- Consider this example:
  - Row X is replicated on nodes M and N
  - Client A writes row X to node N
  - Some period of time t elapses
  - Client B reads row X from node M
- Does client B see the write from client A?
- For ACID model-based system including most RDBMS: yes (favor consistency)

Consistency Model

- A consistency model determines rules for visibility, and to favor availability or consistency
- Consider this example:
  - Row X is replicated on nodes M and N
  - Client A writes row X to node N
  - Some period of time t elapses
  - Client B reads row X from node M
- Does client B see the write from client A?
- For ACID model-based system including most RDBMS: yes (favor consistency)
- For NoSQL, the answer would be: maybe (favor availability)

BASE Properties and Eventual Consistency

- When no updates occur for a long period of time, eventually all updates will propagate through the system and all the nodes will be consistent
  - For a given accepted update and a given node, eventually the update reaches the node or the node is removed from service
  - How long is “eventually”
- Known as BASE (**B**asically **A**vailable, **S**oft state, **E**ventual consistency), as opposed to ACID

NoSQL DBMS

- **Partition tolerance, availability, and eventual consistency**
- Usually do not required data schemas at loading time
- Does not have a uniform way to query the data (aka. no guaranteed language like SQL)
- Each system provides its own querying system, thus, optimized for specific data types
- Easier to scale out compared to RDBMS

Major NoSQL Classifications

- Relational (Hive, Impala)
- Wide-column Stores (HBase, Accumulo, Cassandra, etc.)
- Key-Value Stores (Memcached, Redis, Dynamo, etc.)
- Document Stores (MongoDB, Firebase, etc.)
- Graph DBMS (Neo4J, OrientDB, Titan, etc.)
- Search Engine (Solr, Splunk, Elasticsearch)

Major NoSQL Classifications

- Relational (Hive, Impala)
- Wide-column Stores (HBase, Accumulo, Cassandra, etc.)
- Key-Value Stores (Memcached, Redis, Dynamo, etc.)
- Document Stores (MongoDB, Firebase, etc.)
- Graph DBMS (Neo4J, OrientDB, Titan, etc.)
- Search Engine (Solr, Splunk, Elasticsearch)

Big Data Relational DBMS

- Often part of the Hadoop ecosystem
  - Can be deployed alongside and scale with Hadoop
  - Use Hadoop as a datastore: HDFS or ZooKeeper
  - Use Hadoop components to manage the system
- Most provide SQL-like query capability through API or command line
- Partitioning is still managed by users but can be done dynamically

Hive

- Data warehousing on Hadoop
  - provides web, server and shell interface
- Query language is HiveQL (HQL), variant of SQL
  - partial implementation of SQL-92 (~MySQL)
  - Queries are turned into MapReduce jobs
- All data stored in tables (special encodings on HDFS)
- Schema is managed by Hive (can be applied to existing data on HDFS) — internal tables
  - external tables can “lazily” load schema

Hive Example

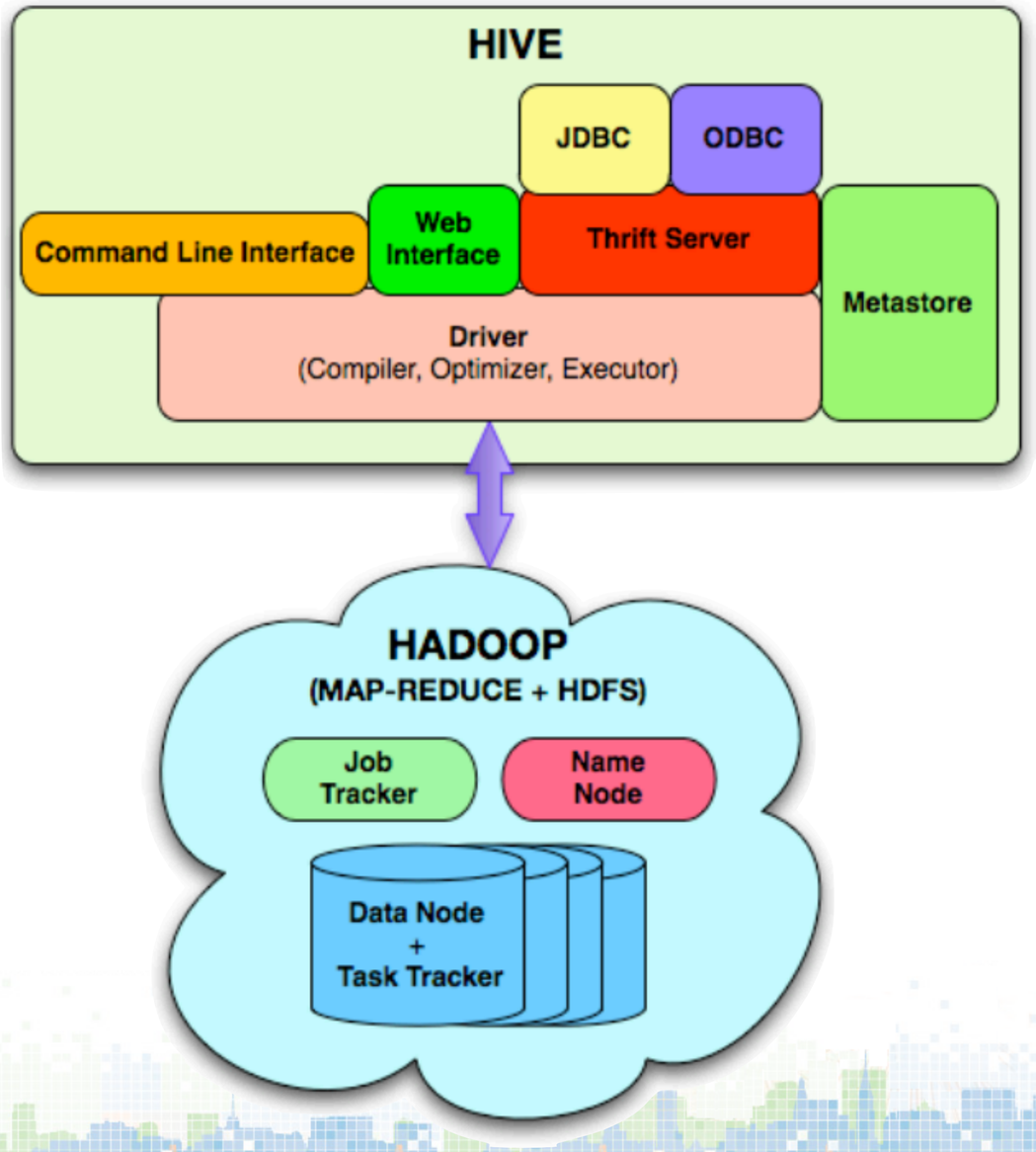
	Example Hive Code
1	`CREATE TABLE records (year STRING, temperature INT, quality INT) ROW FORMAT DELIMITED FIELDS TERMINATED by '\t' ;`
2	`LOAD DATA LOCAL 'data/samples.txt' OVERWRITE INTO TABLE records ;`
3	`SELECT year, MAX(temperature) FROM records WHERE temperature != 9999 AND (quality == 0 OR quality == 1 OR quality == 4 OR quality == 5 OR quality == 9)  GROUP BY year;`

Hive vs. SQL

Hive	Database
“Schema on Read”	“Schema on Write”
Incomplete SQL-92 (never a design goal)	Full SQL-92
No updates, transactions. Indexes available in v0. 7	Updates, transactions and indexes.

Hive Architecture

- Metastore holds metadata
  - Databases, tables
  - Schemas(field names, field types, etc.)
  - Permission information (roles and users)
- Hive data stored in HDFS
  - Tables in directories
  - Partitions of tables in sub-directories
  - Actual data in files



Hive Partitioning

- Hive scans all data for each query unless there is partitioning
- Hive use HDFS folder structures as partitions
  - Data of the same partitions belongs to the same folder
- Partitioning helps speeding up Hive's query but must be specified by users at writing
  - Static partitioning is done when creating tables
  - Dynamic partitioning is done when inserting records
- Partitioning in Hive is Directory-Based Partitioner
- Bucketing in Hive is Hash Partitioner

Major NoSQL Classifications

- Relational (Hive, Impala)
- Wide-column Stores (HBase, Accumulo, Cassandra, etc.)
- Key-Value Stores (Memcached, Redis, Dynamo, etc.)
- Document Stores (MongoDB, Firebase, etc.)
- Graph DBMS (Neo4J, OrientDB, Titan, etc.)
- Search Engine (Solr, Splunk, Elasticsearch)

Wide-column Stores

- Holds millions of columns and billions of rows
  - Data are stores in family of columns, and blocks of values inside a column
    - Employ both vertical and horizontal sharding
- More efficient than row-based DBMS when:
  - Records are inserted in bulk such that column blocks can be updated in blocks
  - Retrievals often touch only a few column

Wide-column Stores

- Pros:
  - Schema free
  - Can hold a large amount of data
- Cons:
  - Huge tables should be sparse
  - Not truly transactional
  - Can only index and sort on keys (not any fields)

Wide-column Stores

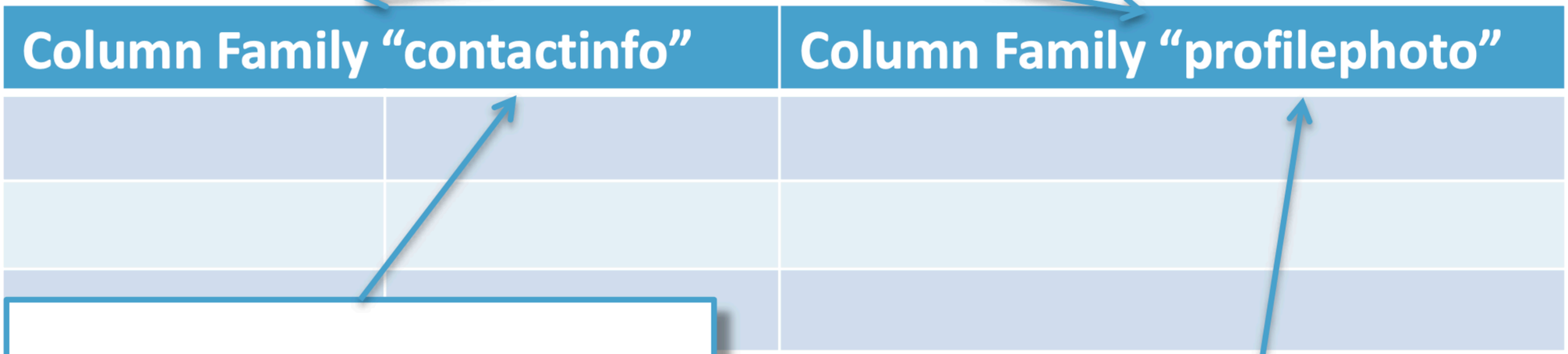
- Cassandra — has highest throughput among other DBMS in class but also with a high latency in reading data (eventual consistency) — ScylaDB is a C++ alternative
  - Apple uses 75,000 Cassandra nodes with 10PB data
  - NetFlix uses 2,500 nodes with 420TB and 1 trillion requests per day
- HBase — better read latency and tightly coupled with Hadoop.
  - Airbnb uses HBase for real-time computation network
  - Facebook uses HBase for its messaging system
- Accumulo — also tightly coupled with Hadoop but with added cell-level access control
- Data can be queried through a SQL like language including the use of 3rd party package such as Apache Drill (or Spark).

— the Hadoop Database

- “Column oriented” database built over HDFS supporting MapReduce and point queries
  - Data indexed by (row key, column key, timestamp)
  - Designed for random, real-time read/write access to big data
  - Designed for sparse tables
- Depends on Zookeeper for consistency and Hadoop for distributed data
- The Site server component provides several interfaces to Web clients

Tables and Column Families

Tables are broken into groupings called Column Families.



Group data frequently accessed together and compress it

Group photos with different settings

[Source: Cloudera Academic Resources]

Rows and Columns

No storage penalty for unused columns

Row key	Column Family "contactinfo"		Column Family "profilephoto"
adupont	fname: Andre	lname: Dupont	
jsmith	fname: John	lname: Smith	image: <smith.jpg>
mrossi	fname: Mario	lname: Rossi	image: <mario.jpg>

Row keys identify a row

Each Column Family can have many columns

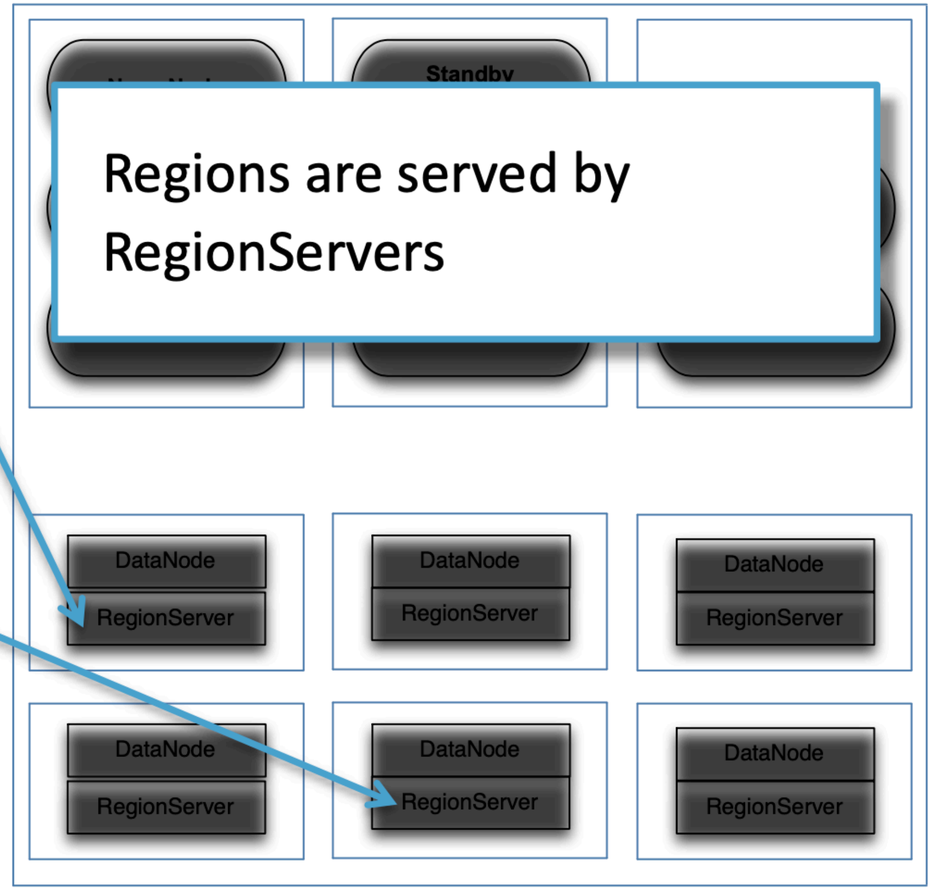
[Source: Cloudera Academic Resources]

Regions — *Horizontal Sharding*

Row key	Column Family "contactinfo"	
adupont	fname: Andre	lname: Dupont
jsmith	fname: John	lname: Smith

Row key	Column Family "contactinfo"	
mrossi	fname: Mario	lname: Rossi
zstevens	fname: Zack	lname: Stevens

A table is broken into regions



[Source: Cloudera Academic Resources]

Access Data in HBase

- HBase does not provide SQL access
- Must interact with data through **`get`**, **`put`**, and **`scan`**
  - **`get`**: retrieve data based on a row key, and column key
  - **`put`**: insert a new record into a table with a row key, and column key
  - **`scan`**: find all matching data in a table
- row-key needs to be carefully designed, and often contain compounds values (joining multiple fields)

Sample get and put

```
1 Get g = new Get(ROW_KEY_BYTES);
2 Result r= table.get(g);
3 byte[] byteArray =
  r.getValue(COLFAM_BYTS,COLDESC_BYTS);
4 String columnValue =
  Bytes.toString(byteArray);
```

```
1 Put p = new
  Put(Bytes.toBytes(ROW_KEY_BYTES);
2 p.add(COLFAM_BYTES, COLDESC_BYTES,
  Bytes.toBytes("value"));
3
4 table.put(p);
```

[Source: Cloudera Academic Resources]

HBase vs. Traditional DBMS

	RDBMS	HBase
Data layout	Row-oriented	Column-oriented
Transactions	Yes	Single row only
Query language	SQL	get/put/scan (or use Hive or Impala)
Security	Authentication/Authorization	Kerberos
Indexes	Any column	Row-key only
Max data size	TBs	PB+
Read/write throughput (queries per second)	Thousands	Millions

[Source: Cloudera Academic Resources]

HBase Use Case

- Use plain HDFS if:
  - Only need to append to dataset (no random write)
  - Usually just read the whole dataset (no random read)
- Use HBase if:
  - Need random write and/or read
  - Do thousands of operations per second on TB+ of data
  - Your data is large but sparse
- Use an RDBMS if:
  - Your data fits on one big node
  - You need full transaction support

Major NoSQL Classifications

- Relational (Hive, Impala)
- Wide-column Stores (HBase, Accumulo, Cassandra, etc.)
- Key-Value Stores (Memcached, Redis, Dynamo, etc.)
- Document Stores (MongoDB, Firebase, etc.)
- Graph DBMS (Neo4J, OrientDB, Titan, etc.)
- Search Engine (Solr, Splunk, Elasticsearch)

Key-Value Stores

- The simplest form of DBMS: a big hash table (or a Python's dictionary)
- Usually for quickly storing lookup information
- Pros:
  - Very fast and efficient
  - Easy to scale out (partition by keys)
  - Simple model
- Cons:
  - Many data specifications cannot be modeled in key-value pairs
    - Opt for the more general document store
  - No query language or relational algebra (aka. just a data store)

Key-Value Stores

- Memcached — in-memory stores to speedup database queries
  - MemcacheDB is a persistent version of Memcached
- Redis — in-memory stores with the ability to persist data to disks
  - Also support Geospatial data through Geohash
- Riak — based on Amazon's DynamoDB, which is a distributed key-value store used in S3
- Project Voldermort — a distributed key-value store used by LinkedIn with a built-in in-memory caching mechanism — *phasing out*

Redis Key/Value Store

```
import redis

r = redis.Redis(

    host='hostname',

    port=port,

    password=‘password')

r.set('foo', 'bar')

value = r.get('foo')

print(value)
```

Redis slides

- [from Arnab Mitra]

Major NoSQL Classifications

- Relational (Hive, Impala)
- Wide-column Stores (HBase, Accumulo, Cassandra, etc.)
- Key-Value Stores (Memcached, Redis, Dynamo, etc.)
- Document Stores (MongoDB, Firebase, etc.)
- Graph DBMS (Neo4J, OrientDB, Titan, etc.)
- Search Engine (Solr, Splunk, Elasticsearch)

Document Stores

- Schema Free (usually JSON like interchange model)
  - No uniform record structure
  - Columns can have different types across records and/or include a list of values
  - Records can be nested
- Query Model: JavaScript or custom
- Aggregations: Map/Reduce
- Indexes are done via B-Trees (self-balance tree)
- Examples:
  - CouchDB, Firebase, and MongoDB

Document Stores

- CouchDB: employ document-level ACID with multi-master replication for high efficiency
  - BBC uses CouchDB for its dynamic contents platform
- MongoDB: document store, most popular with flexible API
  - MetLife and Expedia are using it

```
{
  "_id": "guid goes here",
  "_rev": "314159",

  "type": "abstract",

  "author": "Keith W. Hare"

  "title": "SQL Standard and NoSQL Databases",

  "body": "NoSQL databases (either no-SQL or Not Only SQL)
           are currently a hot topic in some parts of
           computing.",
  "creation_timestamp": "2011/05/10 13:30:00 +0004"
}
```

Document Stores

- Pros:
  - Flexible Schema
  - Can handle dynamic data
- Cons:
  - Joins are expensive, or not supported
  - Not fully transactional
  - Validation has to be designed separately

MongoDB slides

- From Damon LaCaille and Thomas Boyd, MongoDB

Major NoSQL Classifications

- Relational (Hive, Impala)
- Wide-column Stores (HBase, Accumulo, Cassandra, etc.)
- Key-Value Stores (Memcached, Redis, Dynamo, etc.)
- Document Stores (MongoDB, Firebase, etc.)
- Graph DBMS (Neo4J, OrientDB, Titan, etc.)
- Search Engine (Solr, Splunk, Elasticsearch)

Graph Databases

- Store vertices and links, along with data attributes
- Scale vertically, going horizontally
- You can use graph algorithms easily
  - MetLife and Expedia are using Neo4J
- Graph Library for Cluster
  - Spark's GraphX, Giraph, GraphLab
  - More like toolbox, and not DBMS

Graph Database

- Pros:
  - Handle connected data efficiently
  - Easy to query relationships among connected data
- Cons:
  - No true index, or keys
    - Sort on an attribute is expensive
  - Needs additional DBMS for “non-connected” data

Neo4j Slides

- From Michael Hunger, Neo4j

NoSQL Common Advantages

- Cheap, easy to implement (open source)
- Data are replicated to multiple nodes (therefore identical and fault-tolerant) and can be partitioned
  - Down nodes easily replaced
  - No single point of failure
- Easy to distribute
- Don't require a schema
- Can scale up and scale out

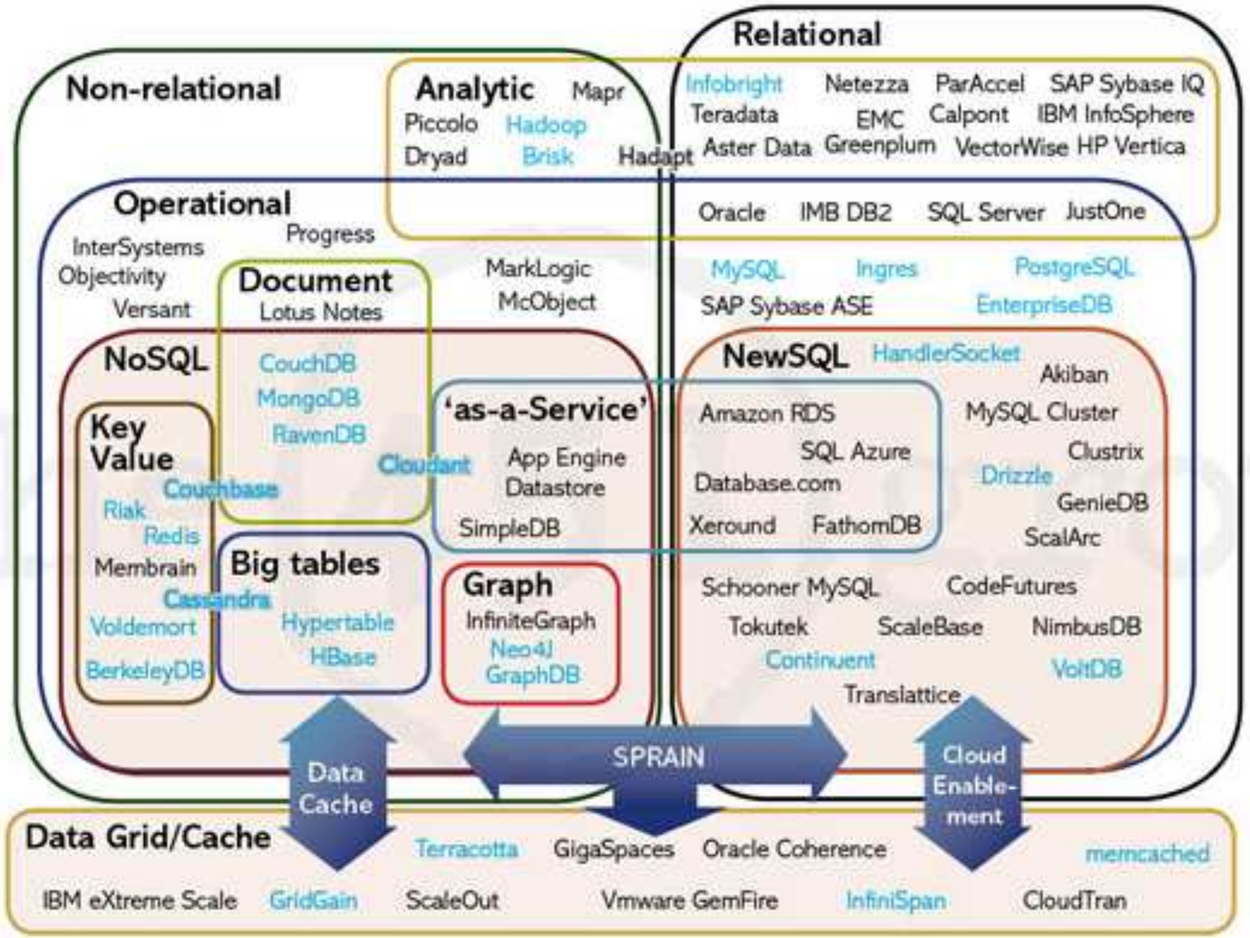
Potential Drawbacks of NoSQL

- ACID transactions
- joins, group by, or order by are expensive
  - due to the relaxation of the key/partition requirements
- SQL, and its API

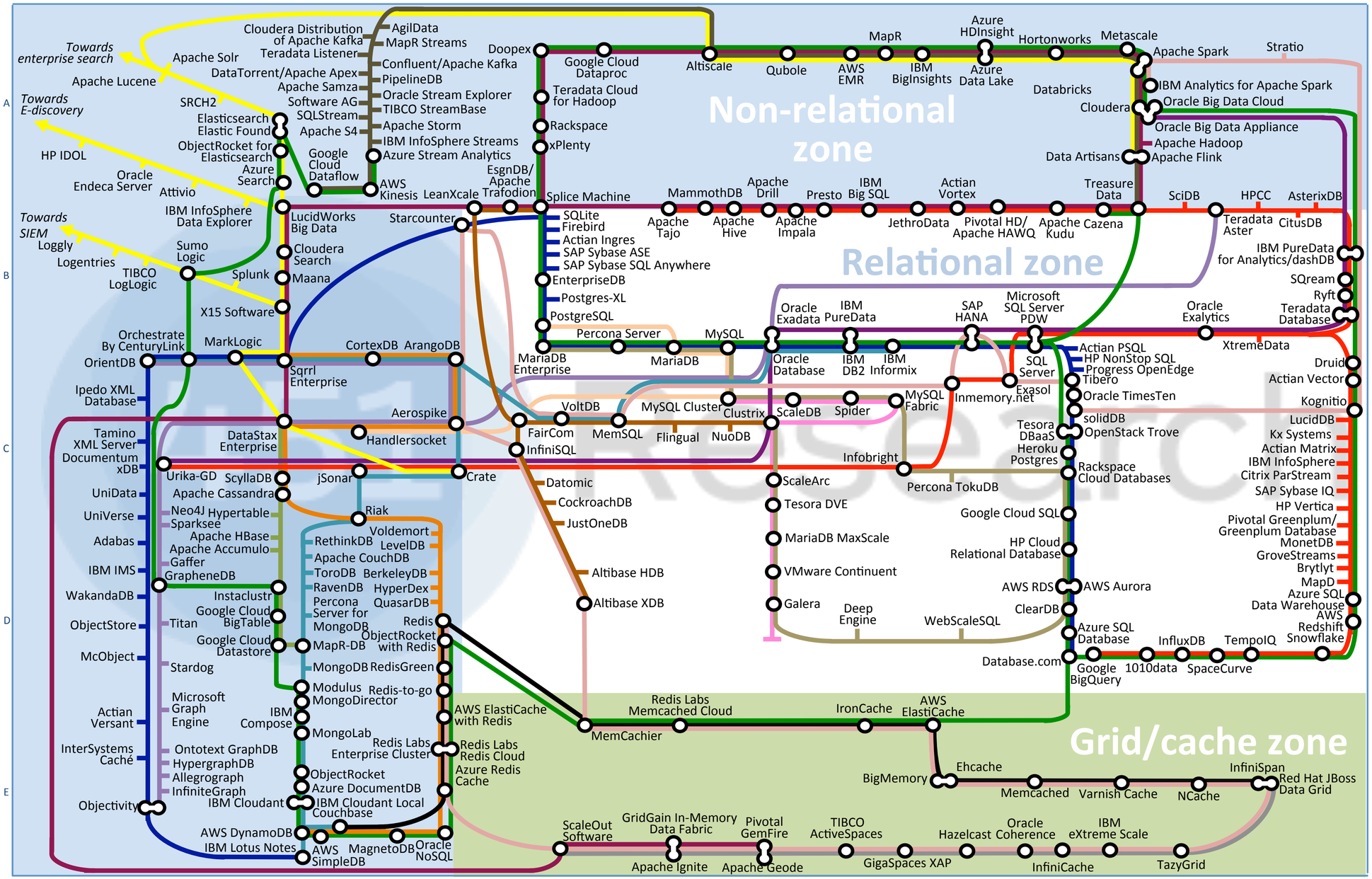
Potential use of NoSQL

- Unstructured Data Analysis, eg. log files
- Social Networking Feeds (many firms hooked in through Facebook or Twitter)
- Data that is not easily analyzed in a RDBMS such as time-series data
- Large data feeds that need to be massaged before entry into an RDBMS

Database Map — 2012



Database Map — 2016



Data Platforms Map

January 2016

Key:

- General purpose
- Specialist analytic
- -as-a-Service
- BigTables
- Graph
- Document
- Key value stores
- Key value direct access
- Hadoop
- MySQL ecosystem
- Advanced clustering/sharding
- New SQL databases
- Data caching
- Data grid
- Search
- Appliances
- In-memory
- Stream processing

https://451research.com/state-of-the-database-landscape

NoSQL Distinguished Characteristics

- Large data volumes
- Scalable replication and distribution
  - Potentially thousands of machine
  - Potentially distributed around the world
- Queries need to return answers quickly
- Mostly query, few updates
- Asynchronous Inserts & Updates
- Schema-less
- ACID transaction properties are not needed – BASE

More Readings on NoSQL

- https://hostingdata.co.uk/nosql-database/
- https://db-engines.com/en/