

Big Data Design

Big Data Management



Knowledge objectives

1. Define the impedance mismatch
2. Identify applications handling different kinds of data
3. Name four different kinds of NOSQL systems
4. Explain three consequences of schema variability
5. Explain the consequences of physical independence
6. Explain the two dimensions to classify NOSQL systems according to how they manage schema
7. Explain the three elements of the RUM conjecture
8. Justify the need of polyglot persistence

Understanding objectives

1. Decide whether two NOSQL designs have a more or less explicit/fix schema

Application objectives

1. Given a relatively small UML conceptual diagram, translate it into a logical representation of data considering flexible schema representation

Motivation

From SQL to NOSQL



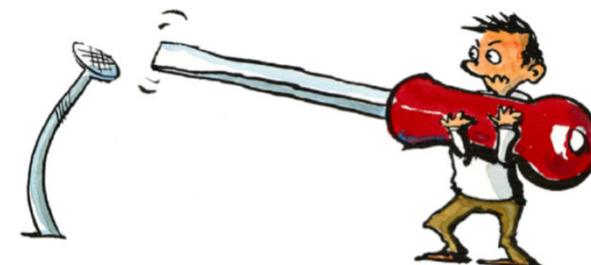
UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Law of the instrument

"Over-reliance on a familiar tool."

Wikipedia

- *Golden hammer* anti-pattern: “A familiar technology or concept applied obsessively to many software problems.”



If the only tool you have is a hammer,
everything looks like a nail.

Law of the Relational Database

Object-relational impedance mismatch is "... one in which a program written using an object-oriented language uses a relational database for storage."

Ireland et al.

- Since we only know relational databases, every time we want to model a new domain we'll automatically think on how to represent it as columns and rows



If the only tool you have is a relational database,
everything looks like a table.

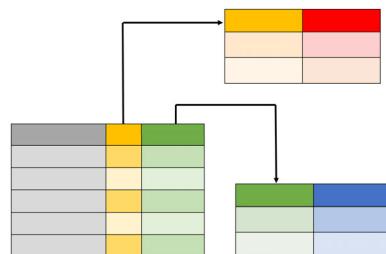
One size does not fit all (Michael Stonebraker)

Not Only SQL (different problems entail different solutions)

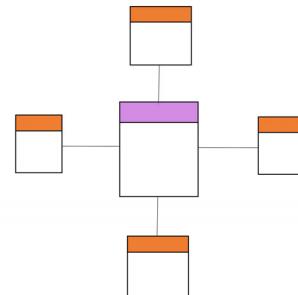
- OLTP
 - VoltDB, HANA, Hekaton
- Data warehousing and OLAP
 - Vertica, Red Shift, Sybase IQ
- Scientific data
 - R, Matlab, SciDB
- Semantic Web and Open Data
 - Virtuoso, GraphDB
- Text
 - Google, Yahoo
- Documents (XML, JSON)
 - MongoDB, CouchDB
- Stream processing
 - Storm, Spark Streaming, Flink

Different data models

Relational (OLTP)



Multidimensional (OLAP)



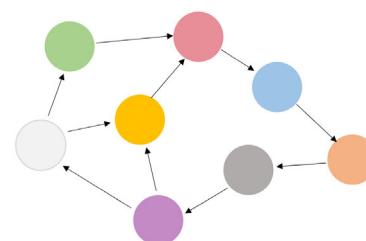
Key-Value

KEY	VALUE

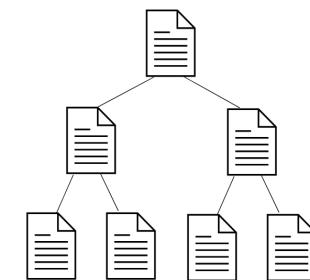
Wide-Column

	Family1	Family2	Family3	Family4
Key				

Graph

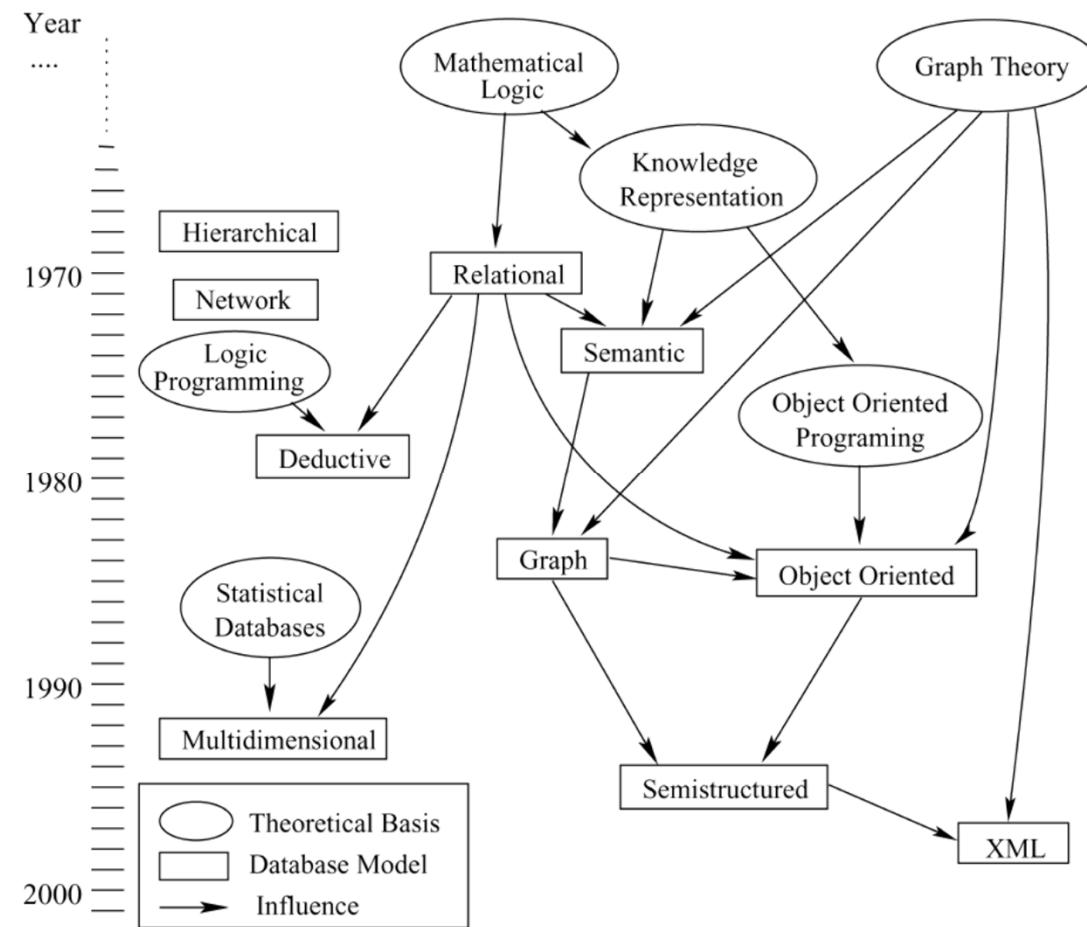


Document



By Aina Montalban, inspired by Daniel G. McCreary and Ann M. Kelly

Evolution of different data models



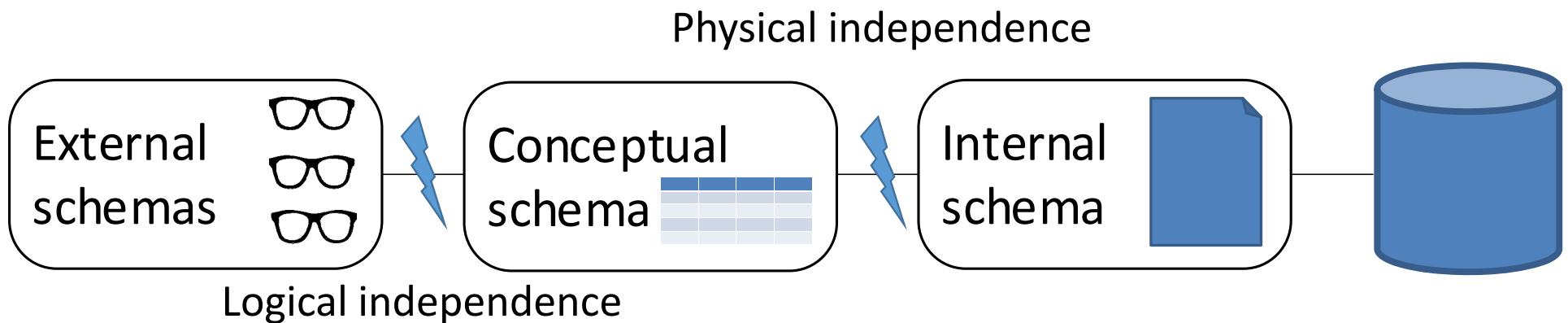
R. Angles and C. Gutierrez

Schema definition

Schema variability

- CREATE TABLE Students(id int, name varchar(50), surname varchar(50), enrolment date);
- INSERT INTO Students (1, 'Sergi', 'Nadal', '01/01/2012', true, 'Igualada'); **WRONG**
- INSERT INTO Students (1, 'Sergi', 'Nadal', NULL); **OK**
- INSERT INTO Students (1, 'Sergi', 'Nadal', '01/01/2012'); **OK**
- Schemaless → `INSERT INTO Students (1, {'Sergi', 'Nadal', '01/01/2012', true});`
- Consequences
 - Gain flexibility
 - Lose semantics (also consistency)
 - The data independence principle is lost (!)
 - The ANSI / SPARC architecture is not followed → Implicit schema
 - Applications can access and manipulate the database internal structures

ANSI/SPARC



ANSI/SPARC



Database models

RELATIONAL

- Based on the relational model
 - Tables, rows and columns
 - Sets, instances and attributes
 - Constraints are allowed
 - PK, FK, Check, ...

When creating the tables you **MUST specify their schema** (i.e., columns and constraints)

Data is restructured when brought into memory (**impedance mismatch**)

NOSQL

- No single reference model
 - Graph data model
 - Document-oriented databases
 - Key-value (~ hash tables)
 - Streams (~ vectors and matrixes)

Ideally, schema specified at insertion, not at definition (**schemaless databases**)

The closer the data model in use looks to the way data is stored internally the better (**read/write through**)

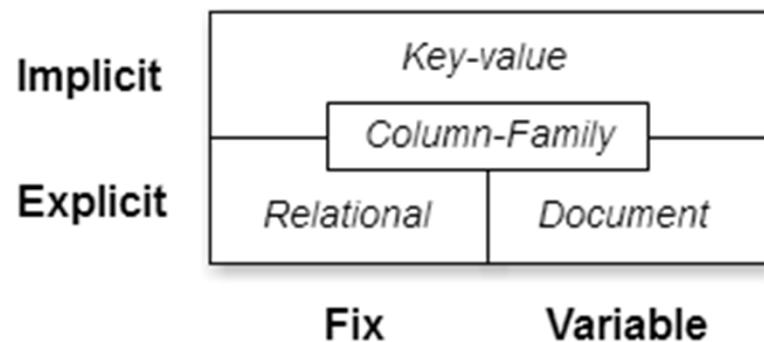
Considered database models

- Relational
city(name, population, region) VALUES ('BCN', '2,000,000', 'CAT')
- Key-Value
['BCN', '2,000,000;CAT']
- Document
{id:'BCN', population:'2,000,000', region:'CAT'}
- Wide-Column
['BCN', population:{value:'2,000,000'}, region:{value:'CAT'}]
['BCN', all:{value:'2,000,000;CAT'}]
['BCN', all:{population:'2,000,000';region:'CAT'}]

Relevant schema dimensions

Some *new* models lack of an explicit schema (declared by the user)

- An implicit schema (hidden in the application code) always remains
 - May reduce the impedance mismatch



Just Another Point of View

SQL



NOSQL



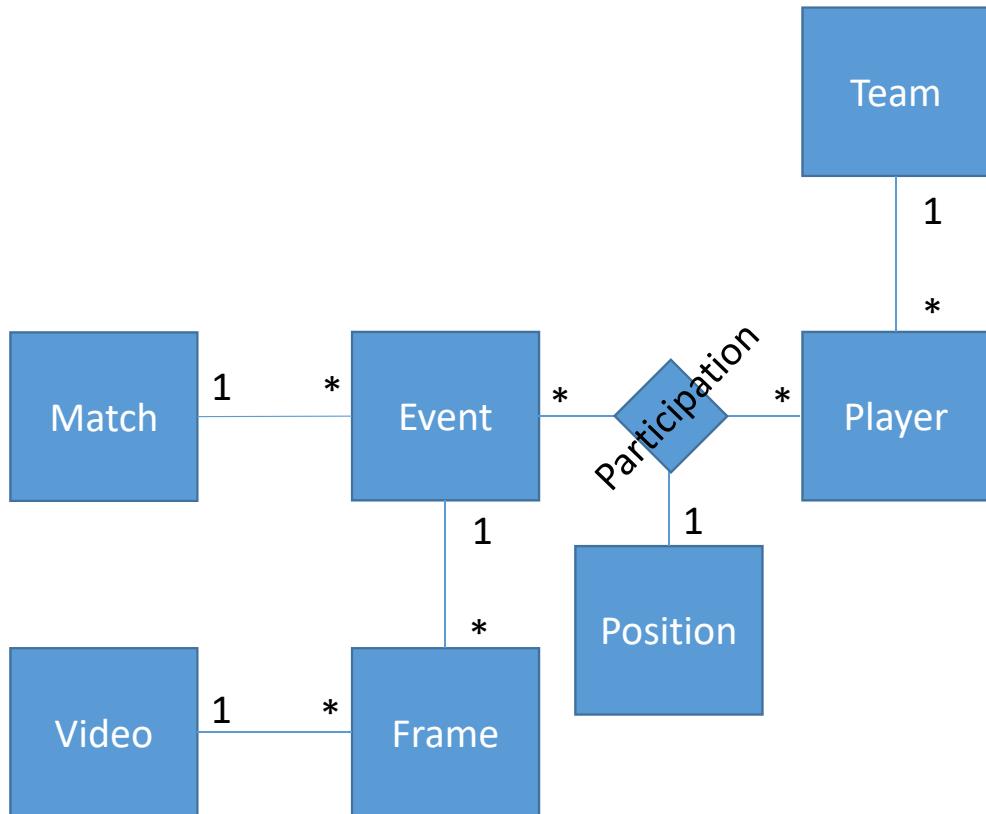
E. Meijer and G. Bierman

Just Another Point of View



E. Meijer and G. Bierman

Events example



Match(mID, name, team, ...)

Video(vID, filename)

Event(eID, kind, duration, ..., mID)

Frame(fID, eID, vID)

Team(tID, name)

Player(pID, name, ..., tID)

Participation(eID, pID, x, y)

First Normal Form (1NF)

Match			
mID	Name	Team	...
1	FCB-SFC	First	...

Video	
vID	CustKey
15	file:///c:/...

Frame		
fID	eID	vID
8	8	15

Event				
eID	Type	duration	...	mID
8	Pass	1.2	...	1

Player			
pID	Name	...	tID
10	Pique	...	1

Team	
tID	Name
1	FC Barcelona

Participation				
eID	pID	xPos	yPos	...
8	10	8	10	...

Non-First Normal Form (NF^2)

```
//in match
{
  "mID" : 1,
  "name" : "FCB-SFC"
...
}
```

```
//in event
{
  "eID" : 8,
  "type" : "Pass",
  "duration" : 1.2,
  "origin_pos" : {
    "type" : "Point",
    "coordinates" : [
      0.430116,
      0.420086
    ]
  },
  "destination_pos" : {
    "type" : "Point",
    "coordinates" : [
      0.350085,
      0.250115
    ]
  }
}
```

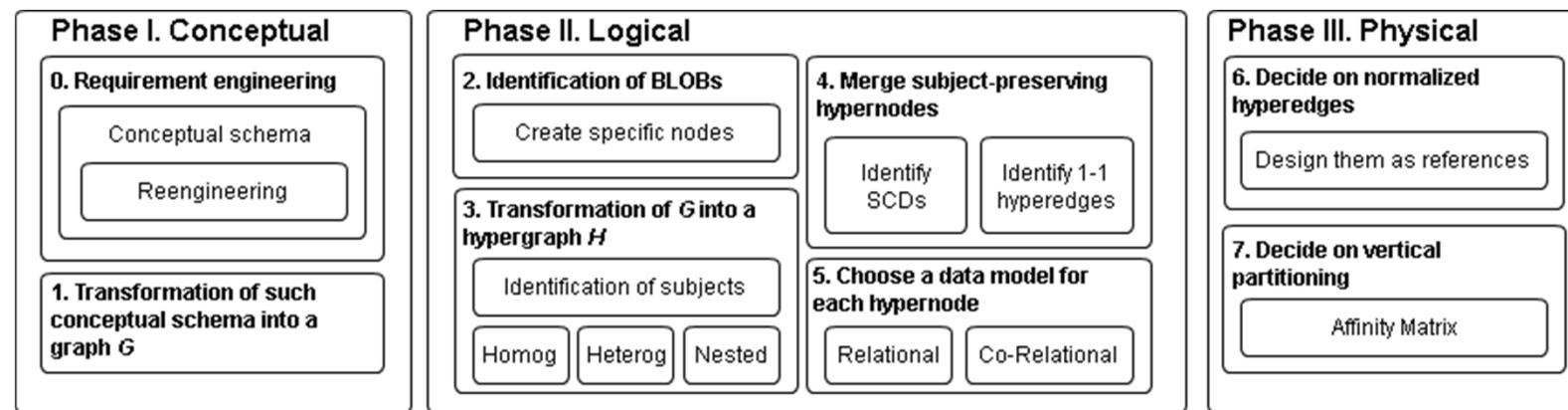
```
//in participation
{
  "eID" : 8,
  "stat_pos_players" : [
    {
      "norm_pos_x" : 0.3060192,
      "norm_pos_y" : 0.492700235294118,
      "origin_player_name" : "PIQUE",
    },
    {
      "norm_pos_x" : 0.463992685714286,
      "norm_pos_y" : 0.0835062352941176,
      "origin_player_name" : "ALBA",
    },
    {
      "norm_pos_x" : 0.429419657142857,
      "norm_pos_y" : 0.420086117647059,
      "origin_player_name" : "BUSQUETS",
    },
    {
      "norm_pos_x" : 0.535141714285714,
      "norm_pos_y" : 0.494179411764706,
      "origin_player_name" : "MESSI",
    },
    {
      "norm_pos_x" : 0.116172342857143,
      "norm_pos_y" : 0.488128235294118,
      "origin_player_name" : "TER STEGEN",
    }, ...
  ]
}
```



Design choices

- Denormalization
- Partitioning/Fragmenting
 - Horizontal
 - Vertical
- Data placement
 - Distribution
 - Clustering

Method's steps



V. Herrero et al. *NOSQL Design for Analytical Workloads: Variability Matters.* ER, 2016

Potential deployments

- Commercial (e.g., Oracle)
 - Relational -> Tables
 - Co-Relational -> XMLType
 - Vertical fragments -> Nested tables
- Open Source (e.g., Hbase+Hive)
 - Relational -> Columns+HCatalog
 - Co-Relational -> Columns
 - Vertical fragments -> Families

Alternative storage structures

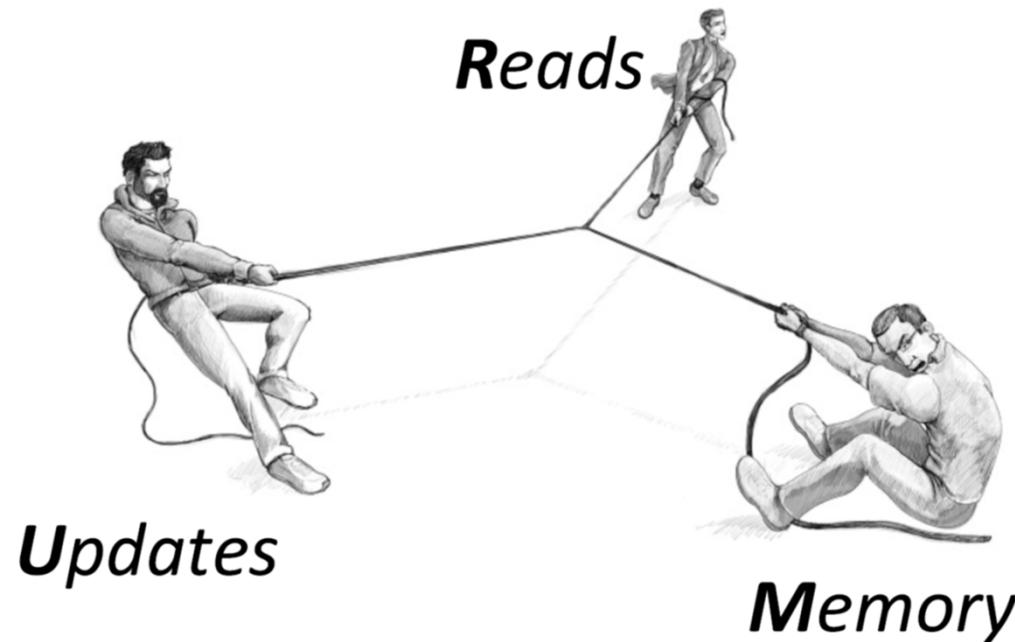
The problem is not SQL

- Relational systems are too generic...
 - OLTP: stored procedures and simple queries
 - OLAP: ad-hoc complex queries
 - Documents: large objects
 - Streams: time windows with volatile data
 - Scientific: uncertainty and heterogeneity
- ...but the overhead of RDBMS has nothing to do with SQL
 - Low-level, record-at-a-time interface is not the solution

Michael Stonebraker
SQL Databases vs. NoSQL Databases
Communications of the ACM, 53(4), 2010

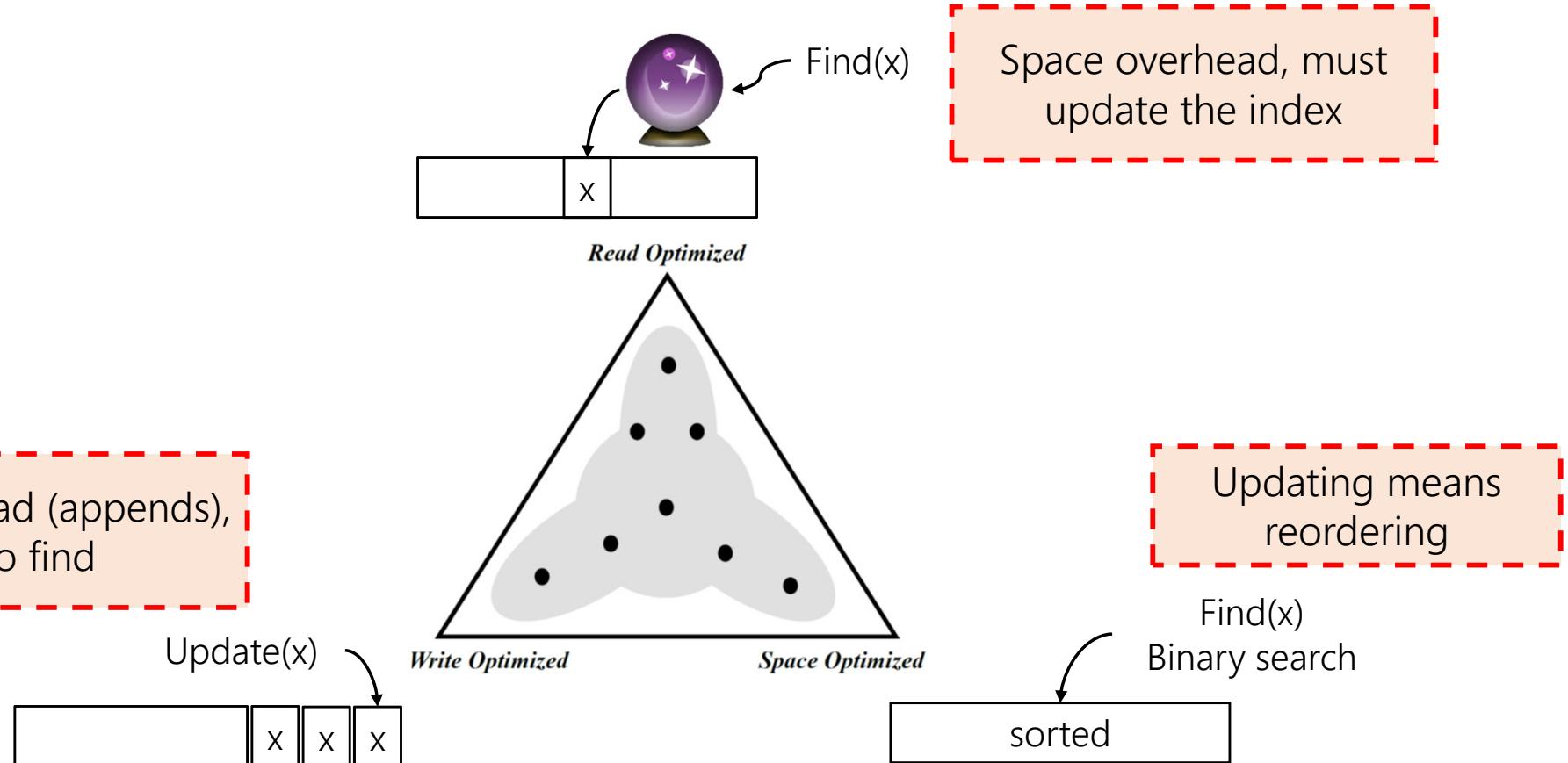
The RUM conjecture

"Designing access methods that set an upper bound for two of the RUM overheads, leads to a hard lower bound for the third overhead which cannot be further reduced."



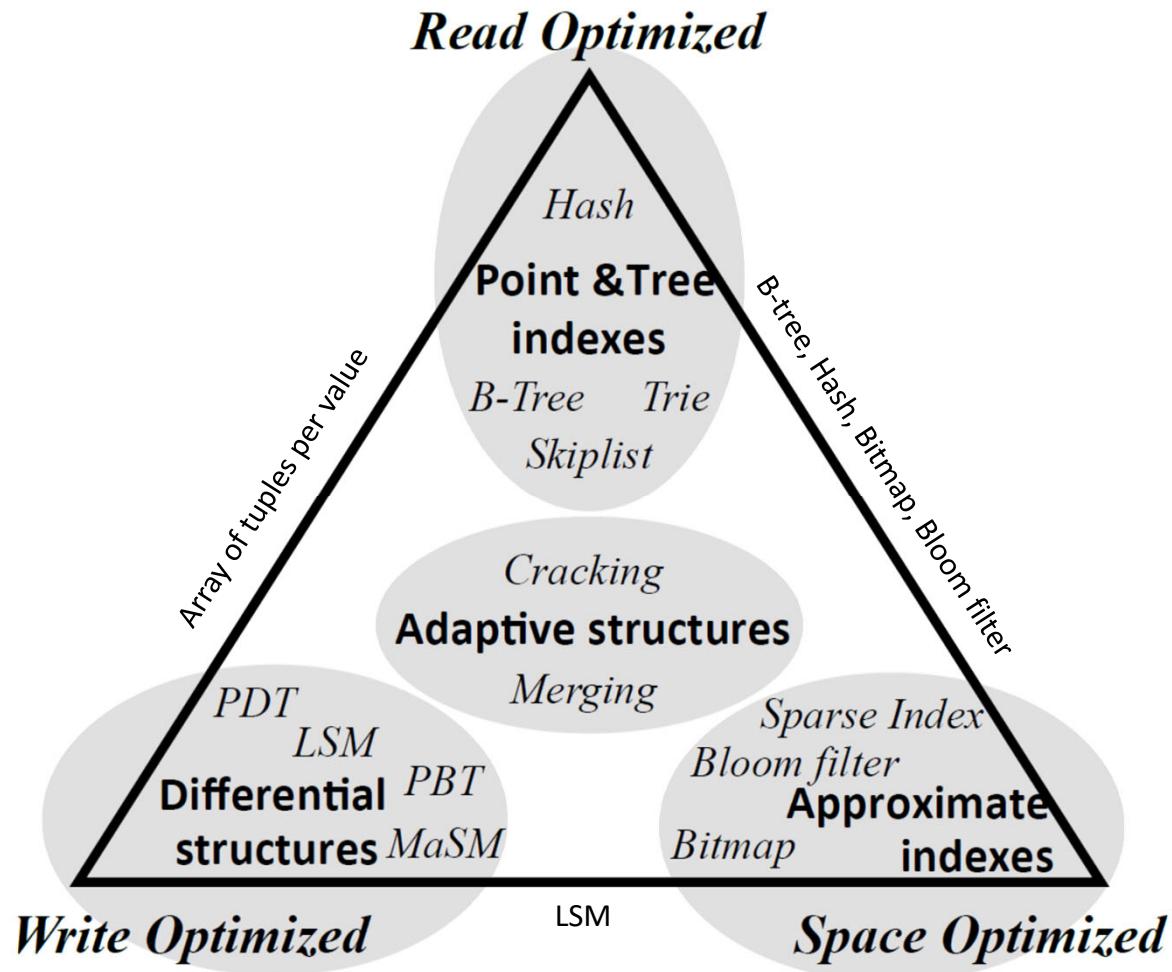
M. Athanassoulis et al.

Example of RUM conjecture



M. Athanassoulis et al.

RUM classification space



M. Athanassoulis et al.

Data Storage

RELATIONAL

Generic architecture that can be tuned according to the needs:

- Mainly-write OLTP Systems
 - Normalization
 - Indexes: B+, Hash
 - Joins: BNL, RNL, Hash-Join, Merge Join
- Read-only DW Systems
 - Denormalized data
 - Indexes: Bitmaps
 - Joins: Star-join
 - Materialized Views

NOSQL

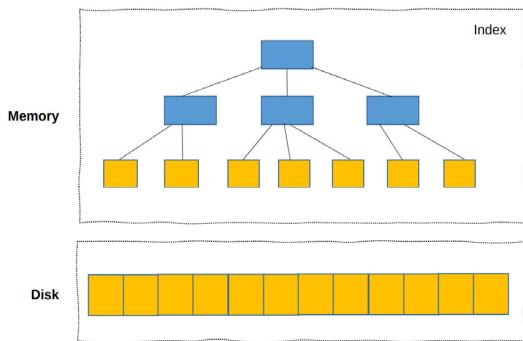
Specific architectures/techniques for a specific need:

- Primary indexes
- Sequential reads
- Vertical partitioning
- Compression
- Fixed-size values
- In-memory processing

Very **specific** and good (but very good) in solving a particular problem

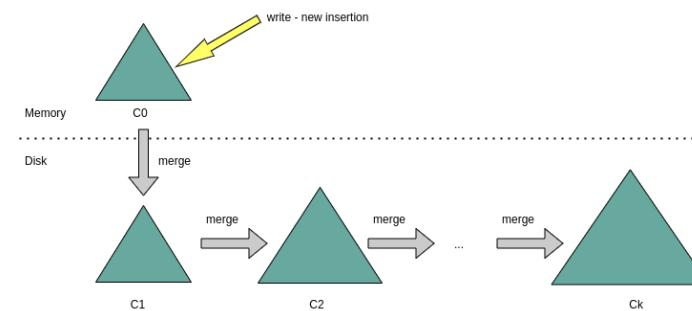
Different internal structures

B-tree



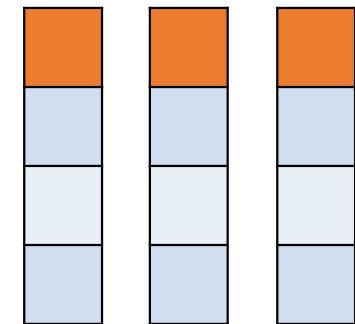
MongoDB, Riak

LSM



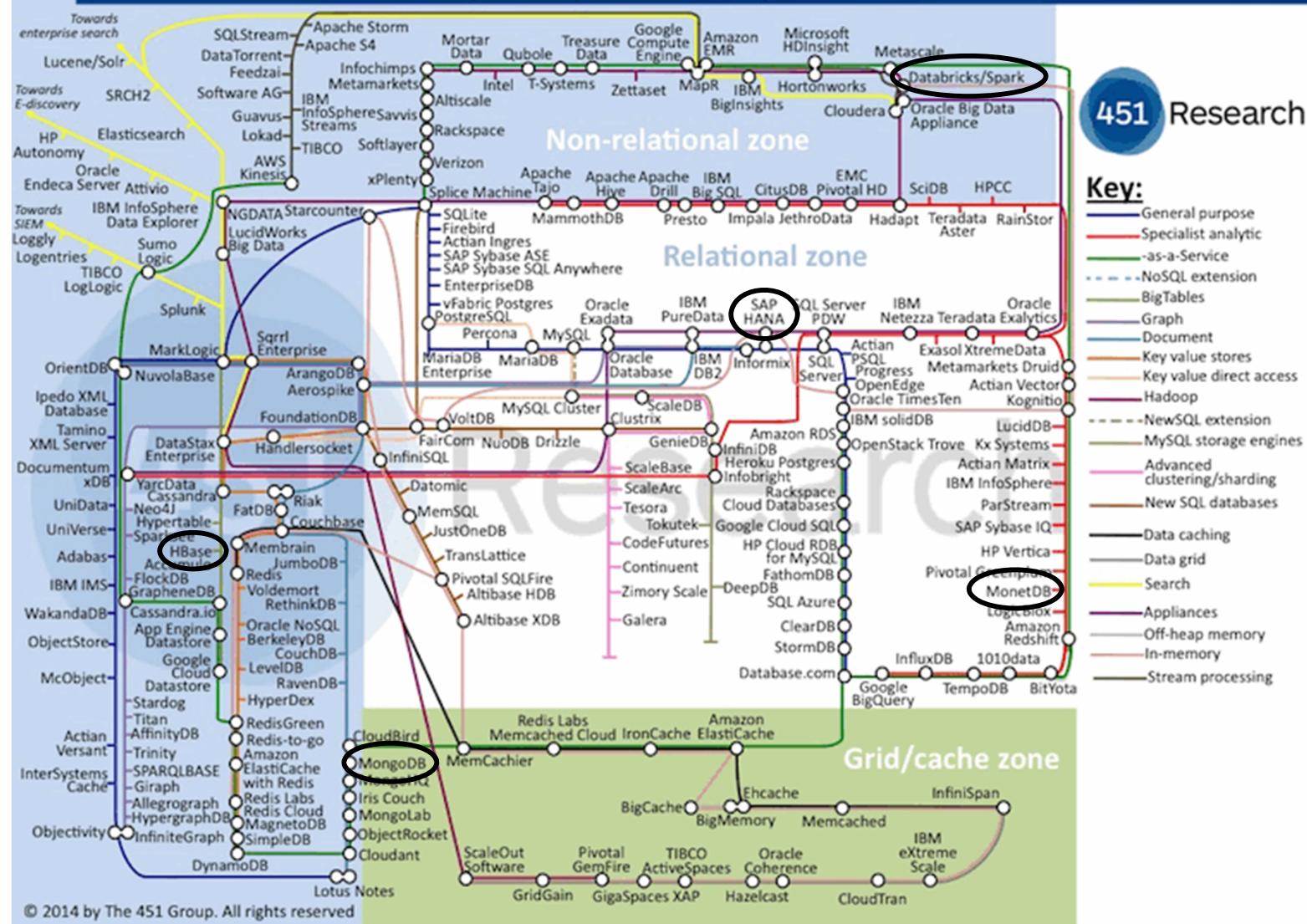
HBase, Cassandra, RocksDB

Vertical Partitioning



Aina Montalban

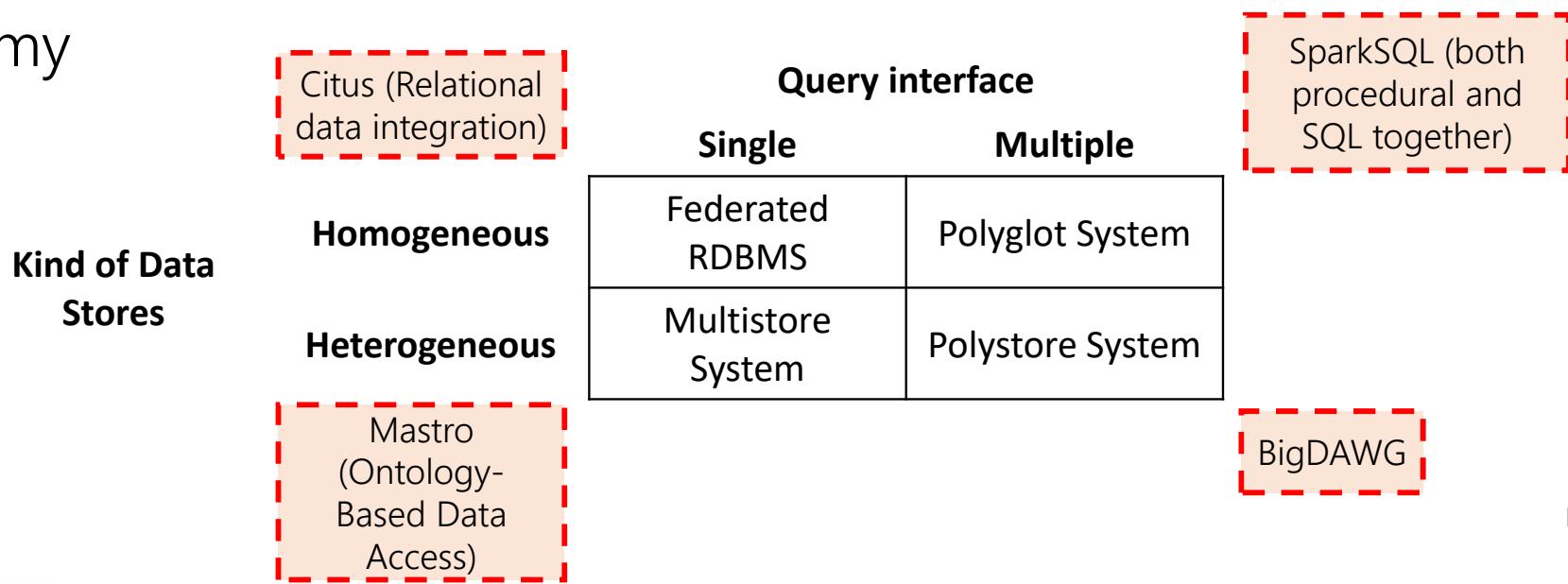
Data Platforms Landscape Map – February 2014



451 Research

Polystores

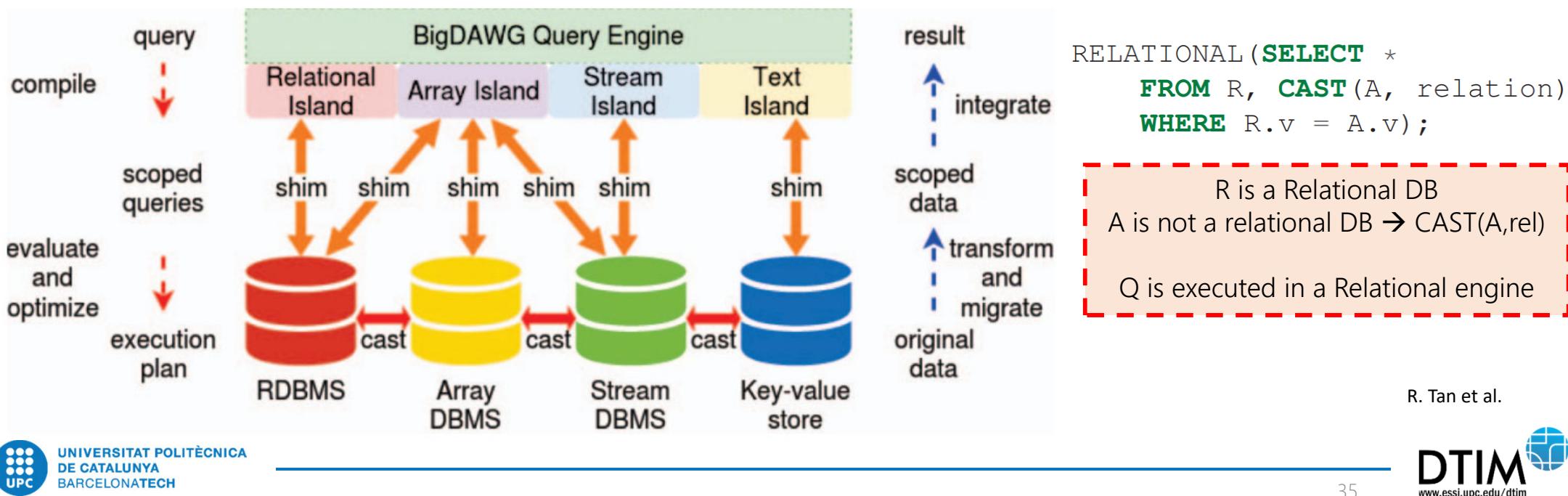
- Federating specialized data stores and enabling query processing across heterogeneous data models
 - ETL disparate data into a single data model degrades performance
 - Curating the pipeline is labor intensive
- Taxonomy



R. Tan et al.

BigDAWG: a polystore system

- A Polystore is a collection of *islands of information*
 - Each island provides a single data model
 - *Shims* transform data from the underlying stores into island's data model
 - Data can be moved across islands



Closing



Summary

- NOSQL systems
- Schemaless databases
- Impedance mismatch
- Polystores

References

- W. J. Brown et al. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. Wiley, 1998
- C. Ireland et al. *A classification of object-relational impedance mismatch* . DBKDA 2009
- M. Stonebraker et al. *The End of an Architectural Era (It's Time for a Complete Rewrite)*. VLDB, 2007
- L. Liu, M.T. Özsu (Eds.). *Encyclopedia of Database Systems*. Springer, 2009
- R. Cattell. *Scalable SQL and NoSQL Data Stores*. SIGMOD Record 39(4), 2010
- M. Stonebraker. *SQL Databases vs. NoSQL Databases*. Communications of the ACM, 53(4), 2010
- E. Meijer and G. Bierman. *A Co-Relational model of data for large shared data banks*. Communications of the ACM 54(4), 2011
- P. Sadagale and M. Fowler. *NoSQL distilled*. Addison-Wesley, 2013
- V. Herrero et al. *NOSQL Design for Analytical Workloads: Variability Matters*. ER, 2016
- M. Athanassoulis et al. *Designing Access Methods: The RUM Conjecture*. EDBT, 2016
- R. Tan et al. *Enabling query processing across heterogeneous data models: A survey*. BigData 2017