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

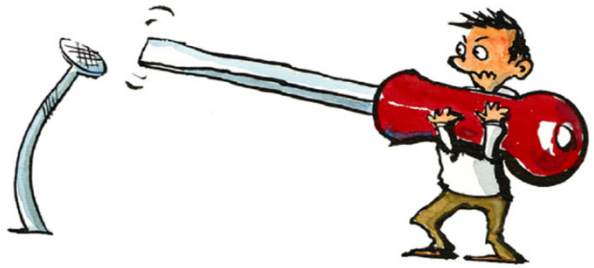
From SQL to NOSQL

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.

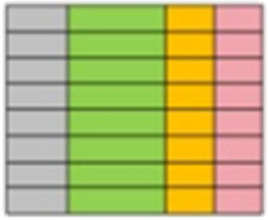
One size does not fit all (Michael Stonebraker)

Not Only SQL (different problems entail different solutions)

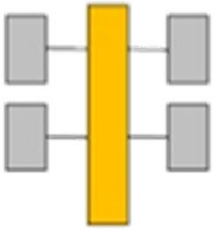
- OLTP
  - Object-Relational
- Data warehousing and OLAP
  - MOLAP
  - Column stores
- Scientific databases and other massive Big Data repositories
  - Key-value stores
  - Column-Family
- Semantic Web and Open Data
  - Graph databases
- Text/documents
  - Document stores (XML, JSON)
- Real-time processing
  - Stream processors

Different data models

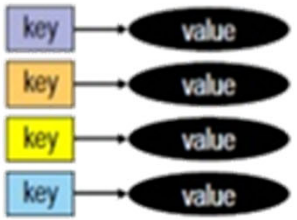
Relational



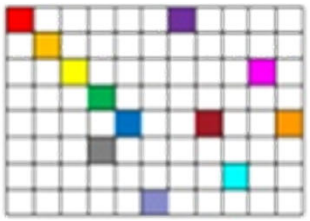
Analytical (OLAP)



Key-Value

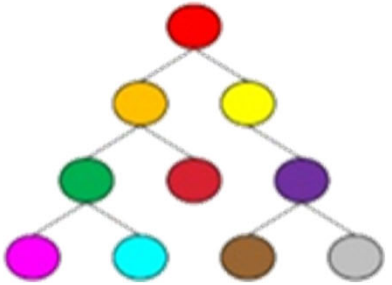


Column-Family



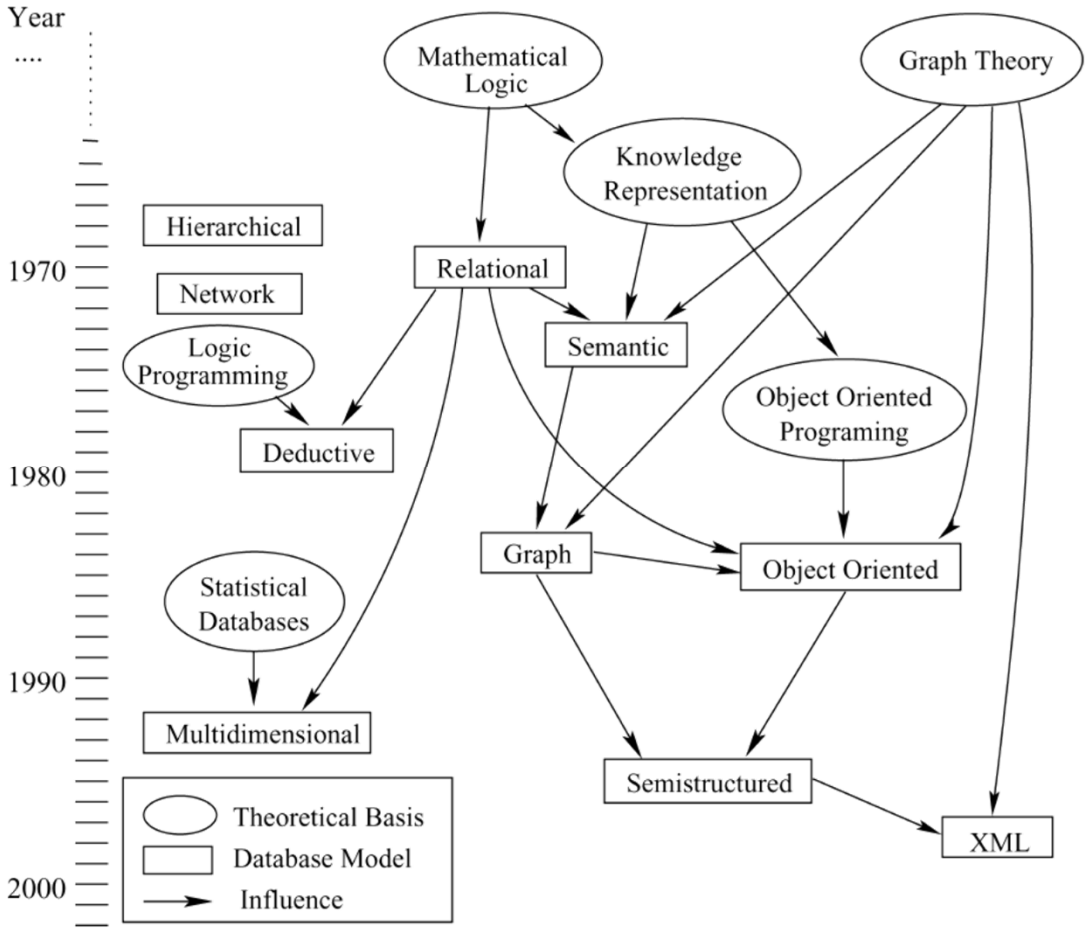
Graph

Document



Daniel G. McCreary and Ann M. Kelly

Evolution of different data models



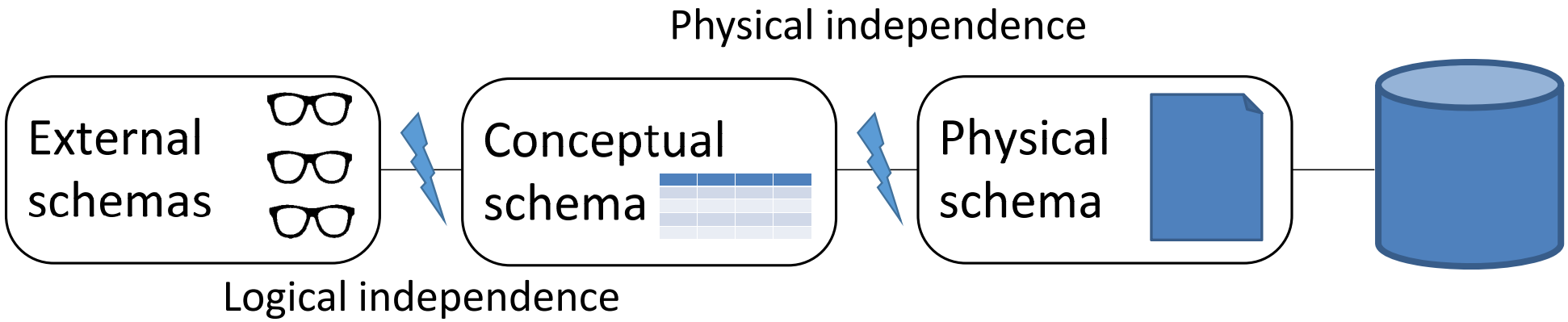
R. Angles and C. Gutierrez

Schema definition

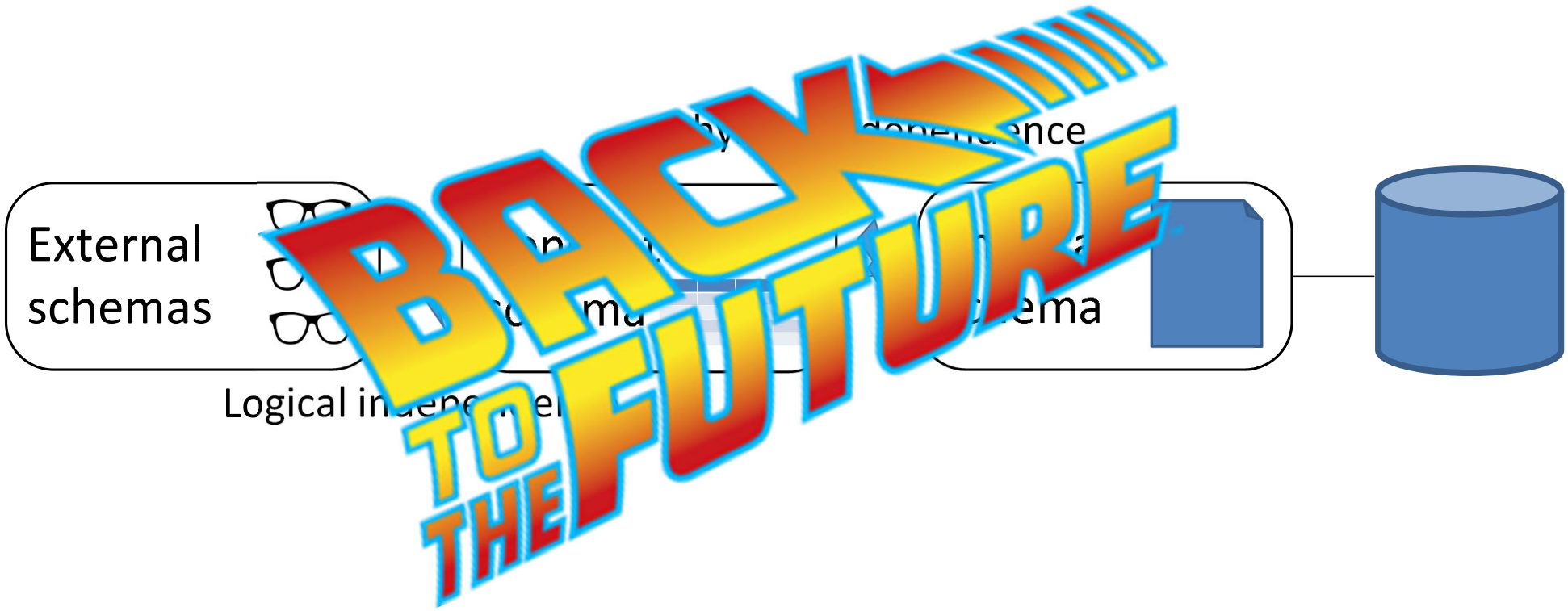
Schema variability

- `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
  - Lose semantics (also consistency)
  - Gain flexibility
  - 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


Physical independence
External schemas
Conceptual schema
Physical schema
Logical independence

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 defined 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'}

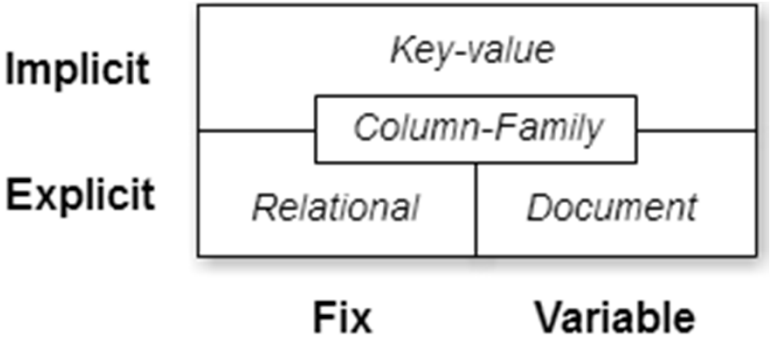
- Column-family

  ['BCN', population:{value:'2,000,000'}, region:{value:'CAT'}]
  ['BCN', all:{value:'2,000,000;CAT'}]
  ['BCN', all:{population:'2,000,000';region:'CAT'}]

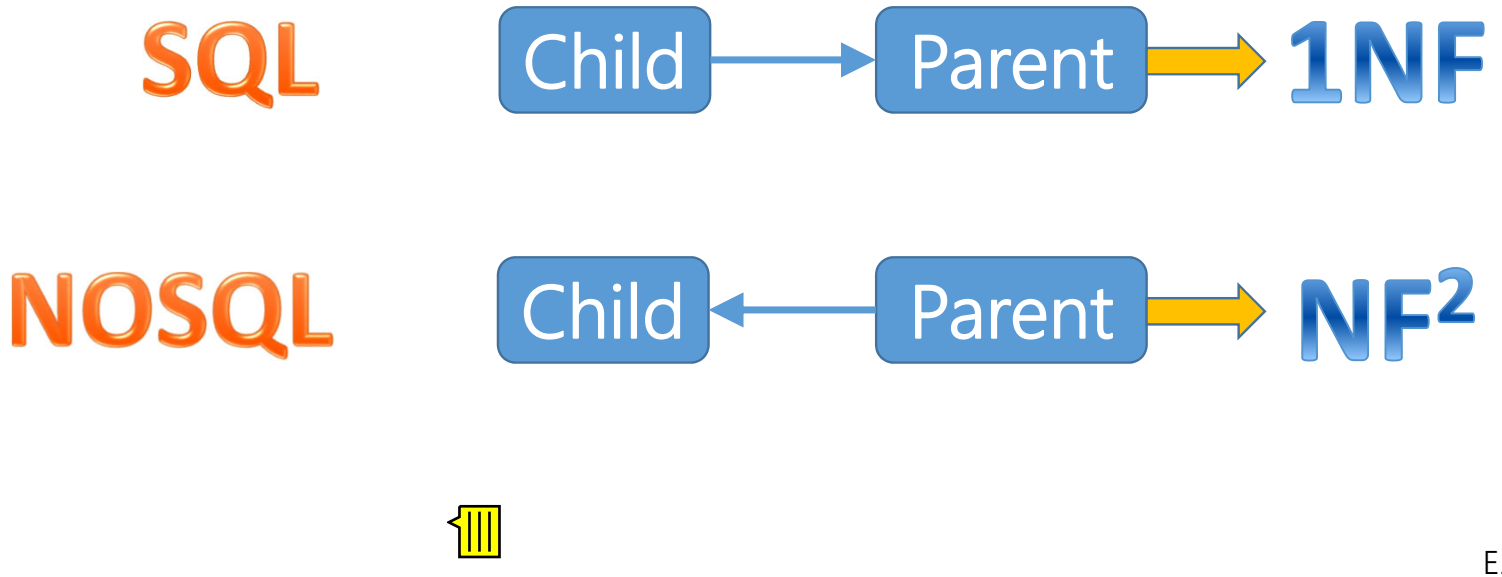
Relevant schema dimensions

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

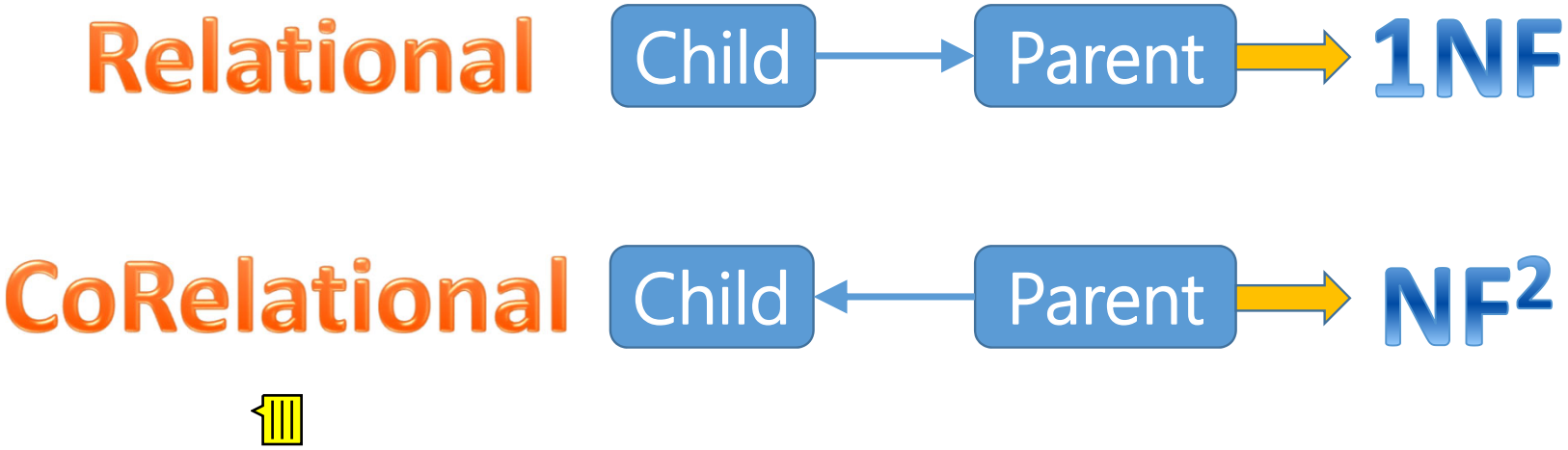


Just Another Point of View



E. Meijer and G. Bierman

Just Another Point of View

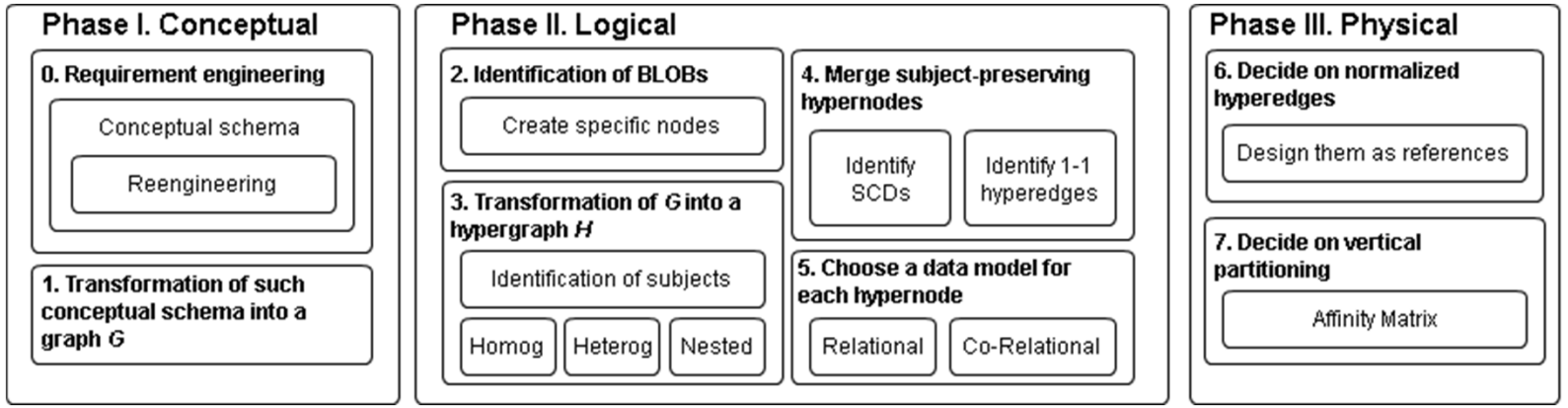


E. Meijer and G. Bierman

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

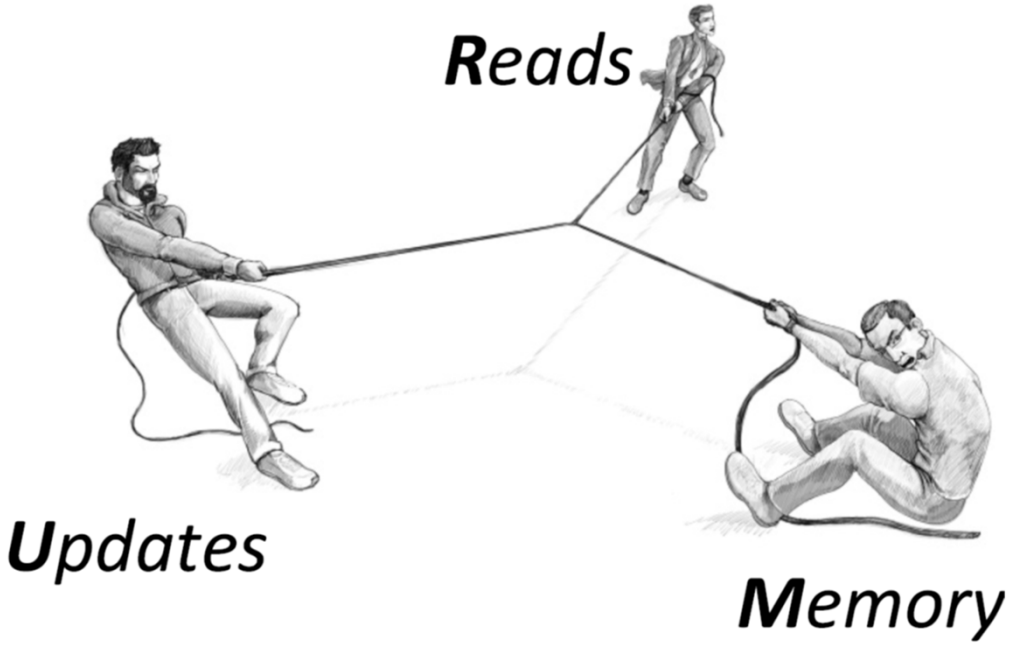
The problem is not SQL

- 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

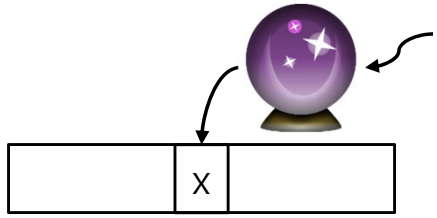
The RUM conjecture

"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."

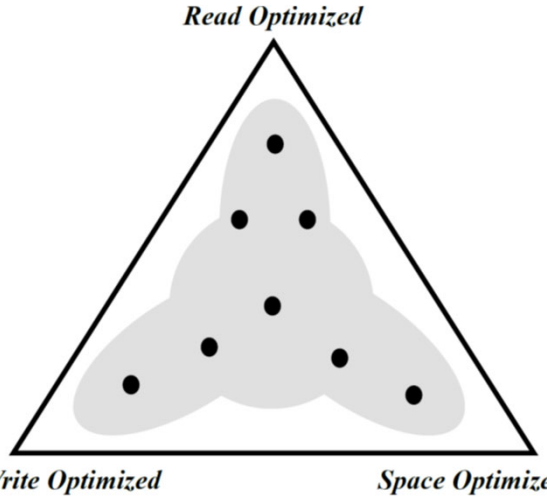


M. Athanassoulis et al.

Example of RUM conjecture

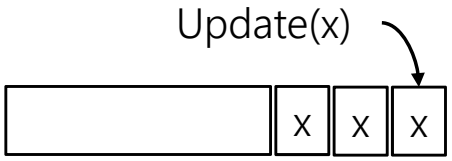


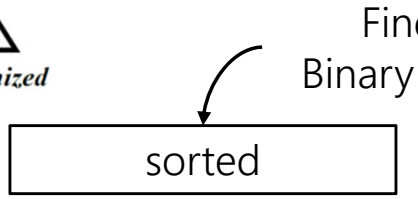
Space overhead, must update the index



Space overhead (appends), hard to find

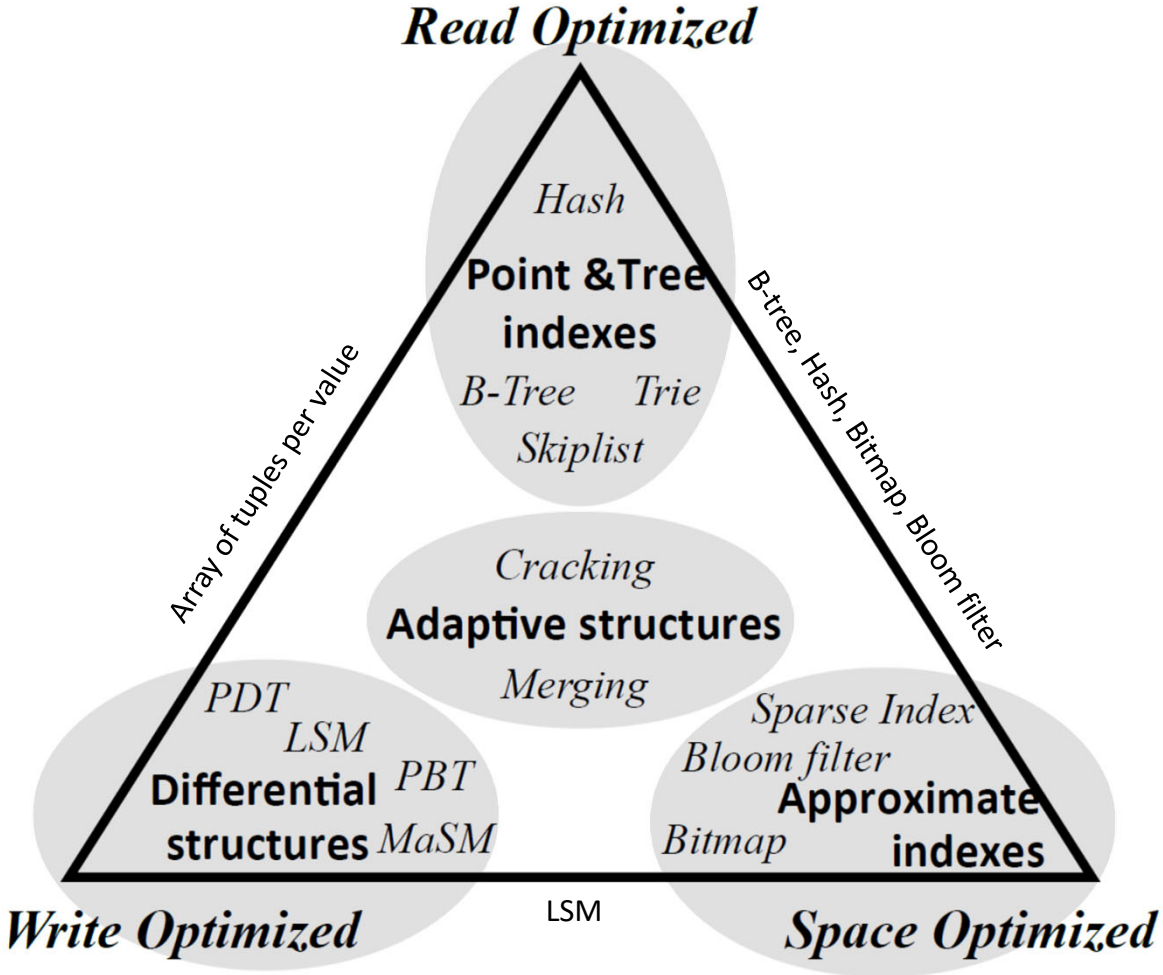
Updating means reordering





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

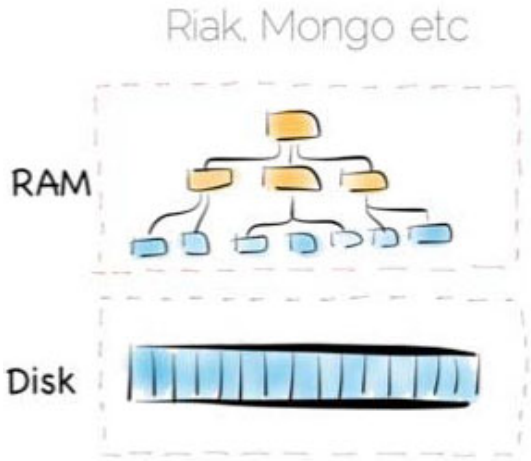
NOSQL

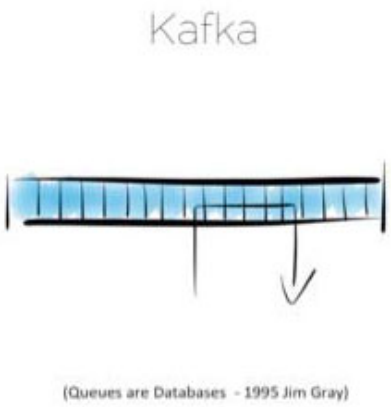
Specific architectures/techniques for a specific need:

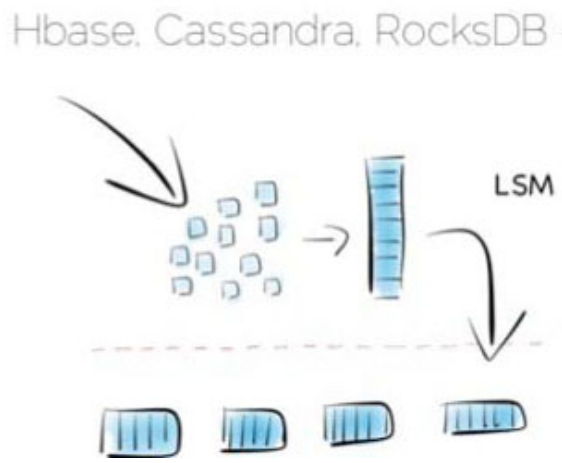
- Primary indexes
- Sequential reads
- Vertical partitioning
- Compression
- Fixed-size values
- In-memory processing

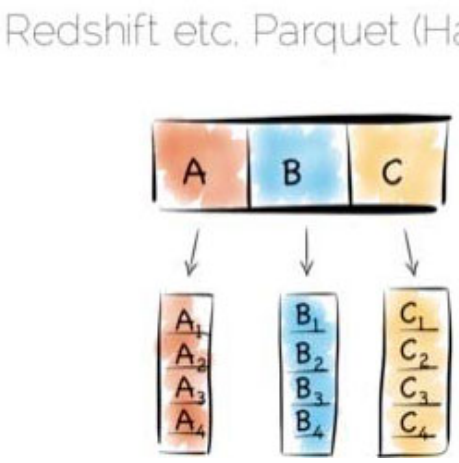
Very specific and good (but very good) in solving a particular problem

Different internal structures

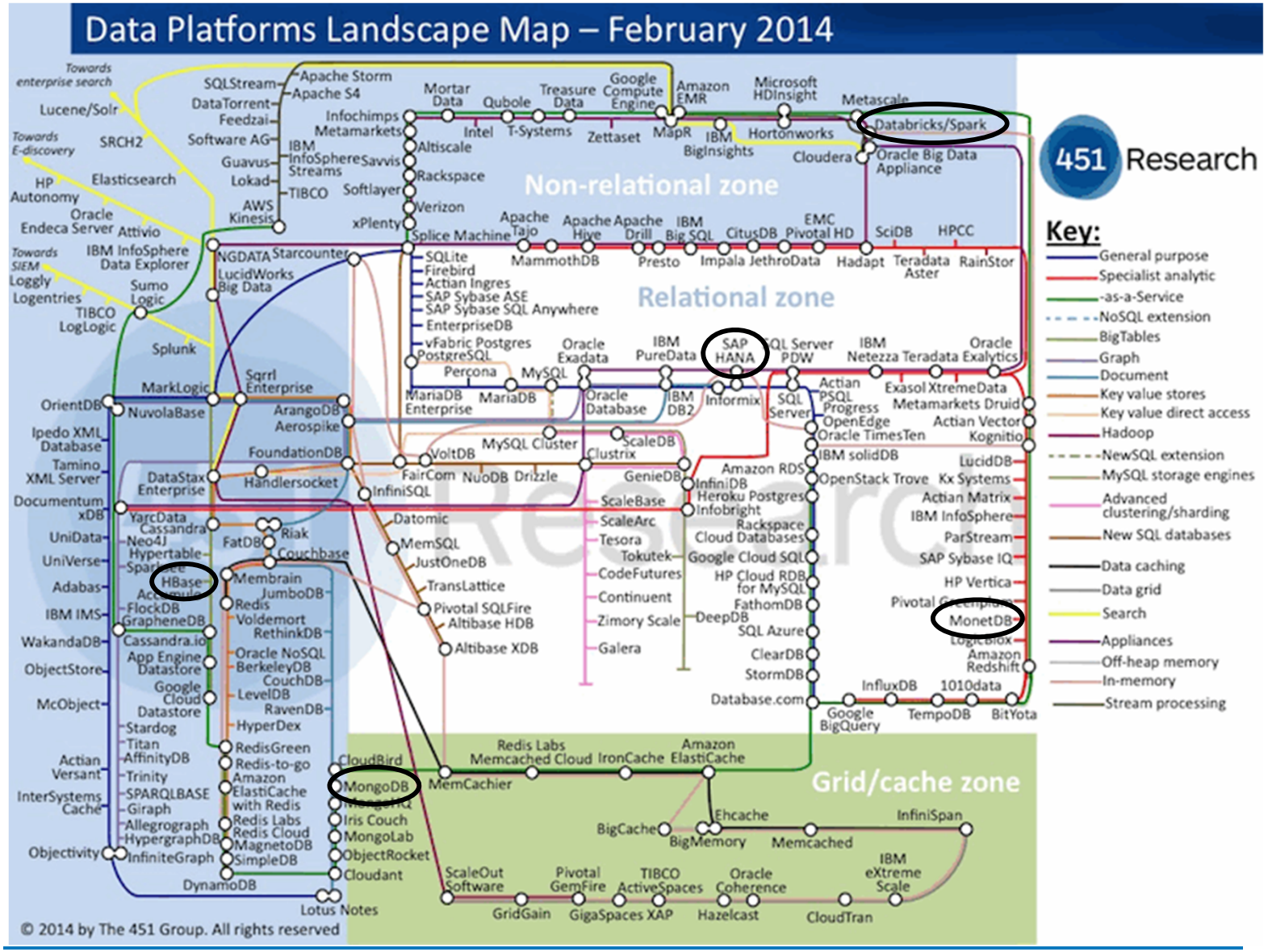






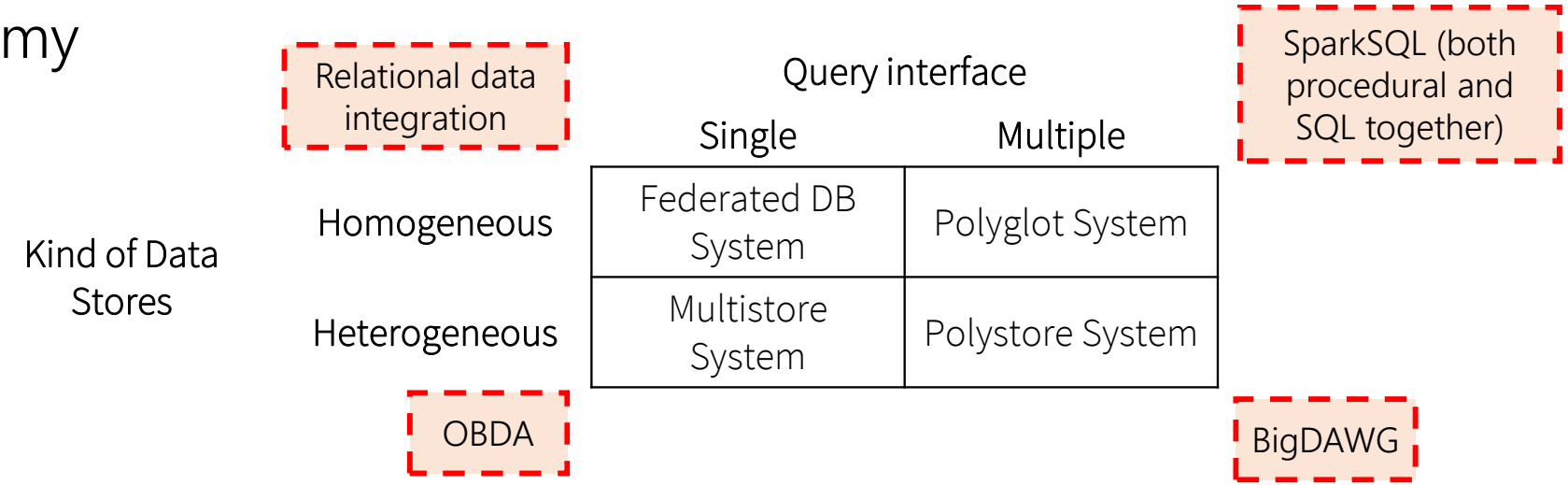


Ben Stopford
Progscon & JAX Finance 2015


Data Platforms Landscape Map – February 2014
451 Research
Non-relational zone
Relational zone
Grid/cache zone
Key:
General purpose
Specialist analytic
-as-a-Service
NoSQL extension
BigTables
Graph
Document
Key value stores
Key value direct access
Hadoop
NewSQL extension
MySQL storage engines
Advanced clustering/sharding
New SQL databases
Data caching
Data grid
Search
Appliances
Off-heap memory
In-memory
Stream processing
Towards enterprise search
Towards E-discovery
Towards SIEM
SQLStream
DataTorrent
Feedzai
Software AG
Guavus
Lokad
AWS Kinesis
Apache Storm
Apache S4
IBM InfoSphere Streams
TIBCO
Splunk
Lucene/Solr
SRCH2
Elasticsearch
HP Autonomy
Oracle Endeca Server
Attivio
IBM InfoSphere Data Explorer
Loggly
Logentries
Sumo Logic
TIBCO LogLogic
Mortar Data
Qubole
Treasure Data
Google Compute Engine
Amazon EMR
Microsoft HDInsight
Metascale
Databricks/Spark
Infochimps
Metamarkets
Intel
T-Systems
Zettaset
MapR
IBM BigInsights
Hortonworks
Cloudera
Oracle Big Data Appliance
Altiscale
Savvis
Rackspace
Softlayer
Verizon
xPlenty
Splice Machine
Apache Tajo
Apache Hive
Apache Drill
IBM Big SQL
CitusDB
EMC Pivotal HD
SciDB
HPCC
Hadapt
Teradata Aster
RainStor
NGDATA
Starcounter
LucidWorks Big Data
SQLite
Firebird
Actian Ingres
SAP Sybase ASE
SAP Sybase SQL Anywhere
EnterpriseDB
vFabric Postgres
PostgreSQL
MammothDB
Presto
Impala
JethroData
Oracle Exadata
IBM PureData
SAP HANA
SQL Server PDW
IBM Netezza
Teradata Aster
Oracle Exalytics
Percona
MySQL
MariaDB Enterprise
MariaDB
Oracle Database
IBM DB2
Informix
SQL Server
Actian PSQL
Progress OpenEdge
Oracle TimesTen
Exasol
XtremeData
Metamarkets Druid
Actian Vector
Kognitio
MarkLogic
Sqrrl Enterprise
OrientDB
NuvolaBase
ArangoDB
Aerospike
Ipedo XML Database
Tamino XML Server
DataStax Enterprise
FoundationDB
Handlersocket
VoltDB
FairCom
NuoDB
Drizzle
MySQL Cluster
Clustrix
ScaleDB
GenieDB
InfiniSQL
IBM solidDB
OpenStack Trove
LucidDB
Kx Systems
Amazon RDS
InfiniDB
Heroku Postgres
Infobright
Actian Matrix
IBM InfoSphere
Rackspace Cloud Databases
ParStream
Google Cloud SQL
SAP Sybase IQ
HP Cloud RDB for MySQL
HP Vertica
FathomDB
Pivotal Greenplum
DeepDB
MonetDB
SQL Azure
ClearDB
StormDB
Amazon Redshift
Database.com
InfluxDB
1010data
Google BigQuery
TempoDB
BitYota
Documentum xDB
UniData
UniVerse
Adabas
IBM IMS
WakandaDB
ObjectStore
McObject
Actian Versant
InterSystems Cache
Objectivity
YarcData
Cassandra
Neo4J
Hypertable
Sparcity
HBase
Accumulo
FlockDB
GrapheneDB
Cassandra.io
App Engine Datastore
Google Cloud Datastore
Stardog
Titan
AffinityDB
Trinity
SPARQLBASE
Giraph
Allegrograph
HypergraphDB
InfiniteGraph
FatDB
Riak
Couchbase
Membrain
JumboDB
Redis
Voldemort
RethinkDB
Oracle NoSQL
BerkeleyDB
CouchDB
LevelDB
RavenDB
HyperDex
RedisGreen
Redis-to-go
Amazon ElastiCache with Redis
Redis Labs
Redis Cloud
MagnetoDB
SimpleDB
DynamoDB
Datomic
MemSQL
JustOneDB
TransLattice
Pivotal SQLFire
Altibase HDB
Altibase XDB
ScaleBase
ScaleArc
Tesora
Tokutek
CodeFutures
Continuent
Zimory Scale
Galera
CloudBird
MongoDB
Iris Couch
MongoLab
ObjectRocket
Cloudant
Lotus Notes
Redis Labs Memcached Cloud
IronCache
Amazon ElastiCache
MemCachier
BigCache
Ehcache
BigMemory
Memcached
InfiniSpan
ScaleOut Software
Pivotal GemFire
TIBCO ActiveSpaces
Oracle Coherence
IBM eXtreme Scale
GridGain
GigaSpaces XAP
Hazelcast
CloudTran
© 2014 by The 451 Group. All rights reserved

Polystores

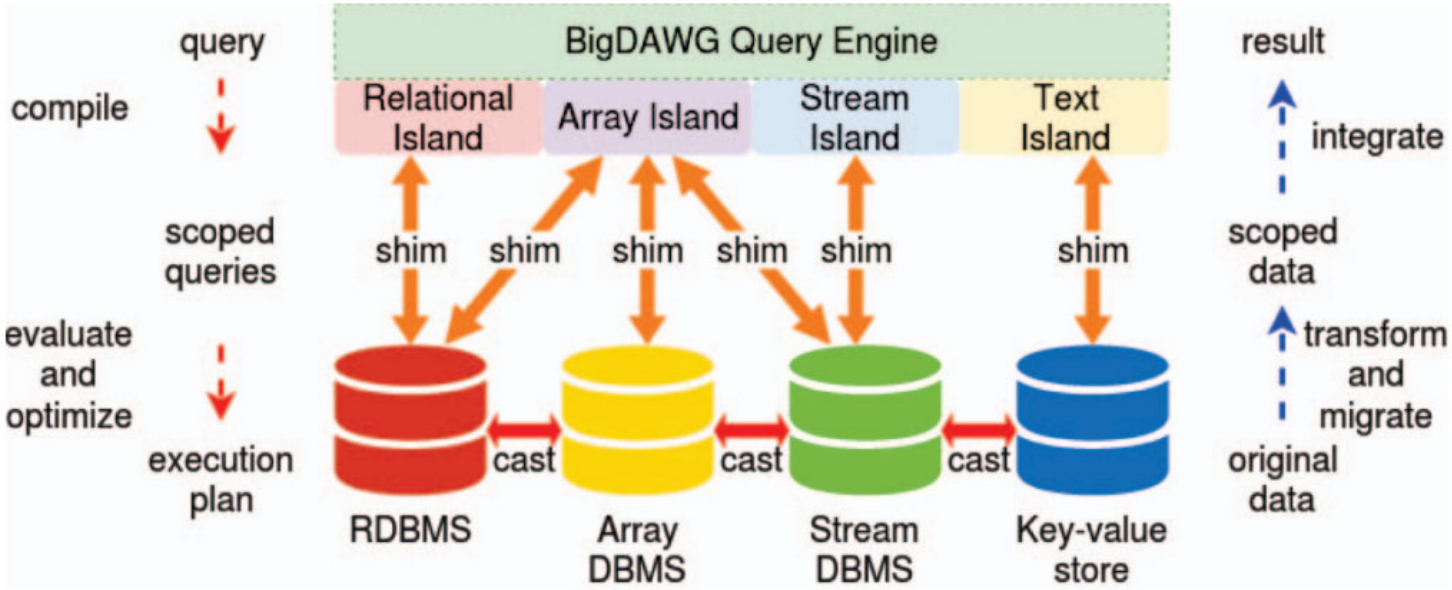
- Federating specialized data stores and enabling query processing across heterogeneous data models
  - ETL disparate data into a single data model degrades performance
  - Curating the pipeline is labor intensive
- Taxonomy



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




```
RELATIONAL(SELECT *
    FROM R, CAST(A, relation)
    WHERE R.v = A.v);
```

R is a relational DB
A is not a relational DB → CAST(A,rel)

Q is executed in a relational engine

R. Tan et al.

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