Design and Implementation of the CloudMdsQL Multistore System

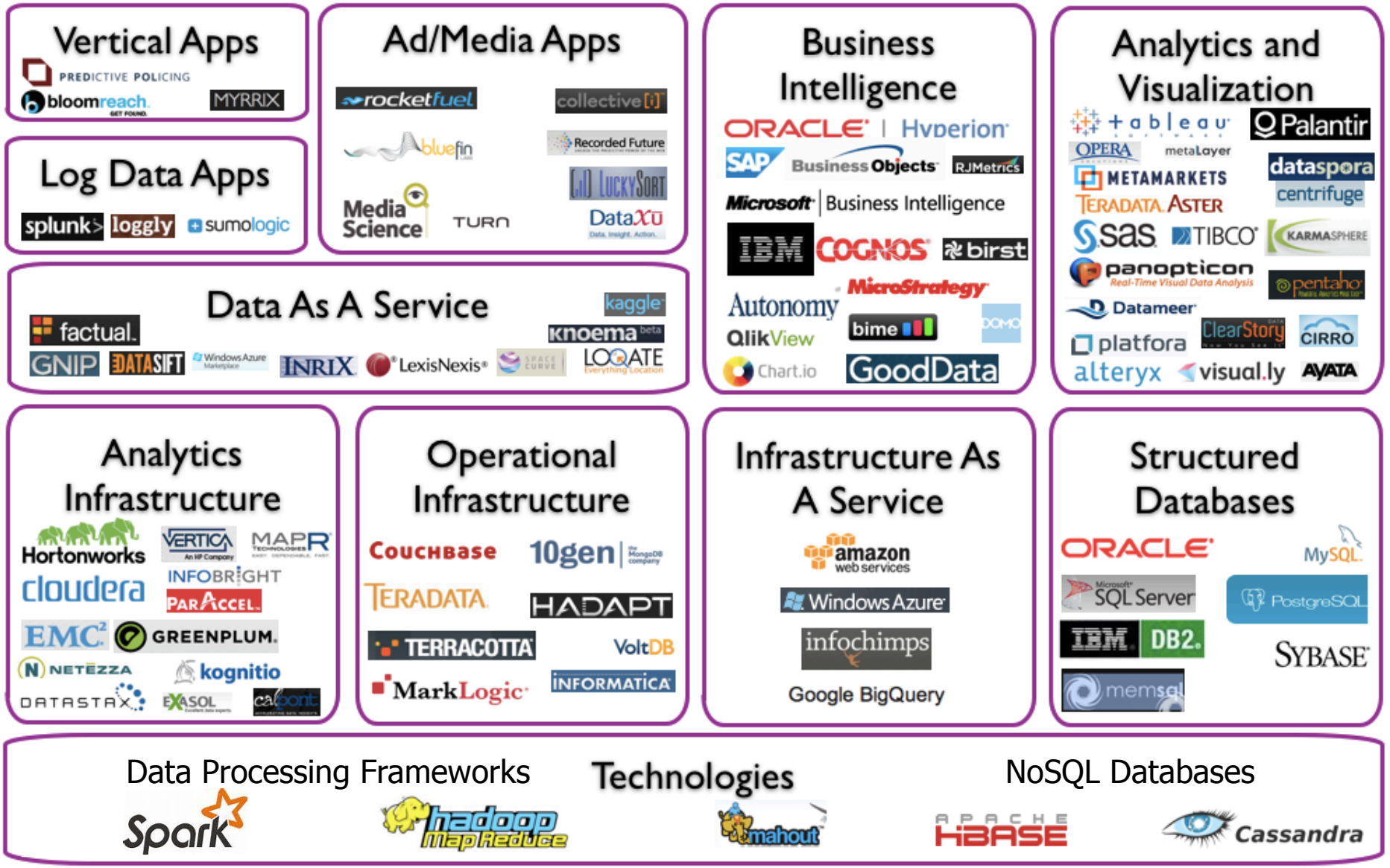
Patrick Valduriez

Inria, Montpellier, France

CoherentPaaS

With Boyan Kolev, Carlyna Bondiombouy, Oleksandra Levchenko, Ricardo Jimenez, Raquel Pau and Jose Pereira

Cloud & Big Data Landscape



dave@vcdave.com

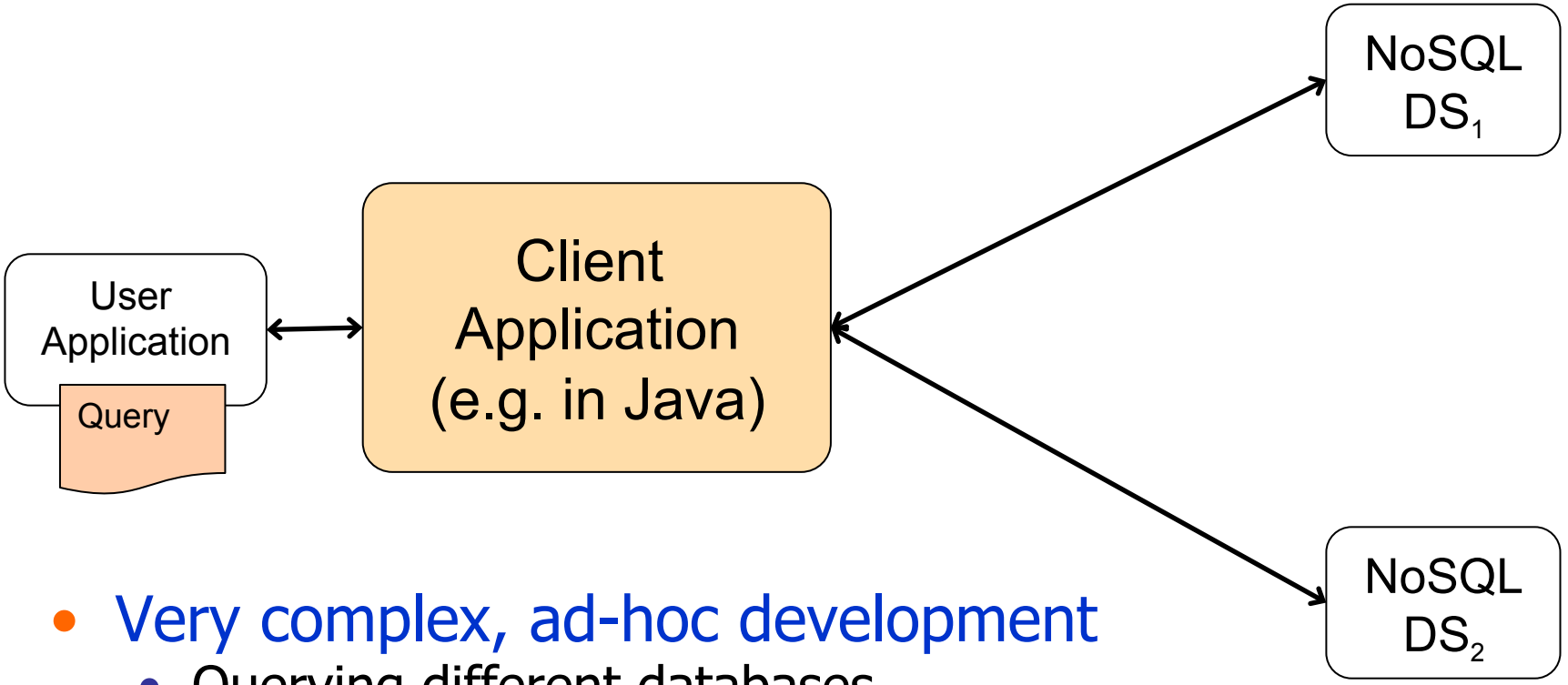
blogs.forbes.com/davefeinleib

Cloud & Big Data Landscape

dave@vcdave.com

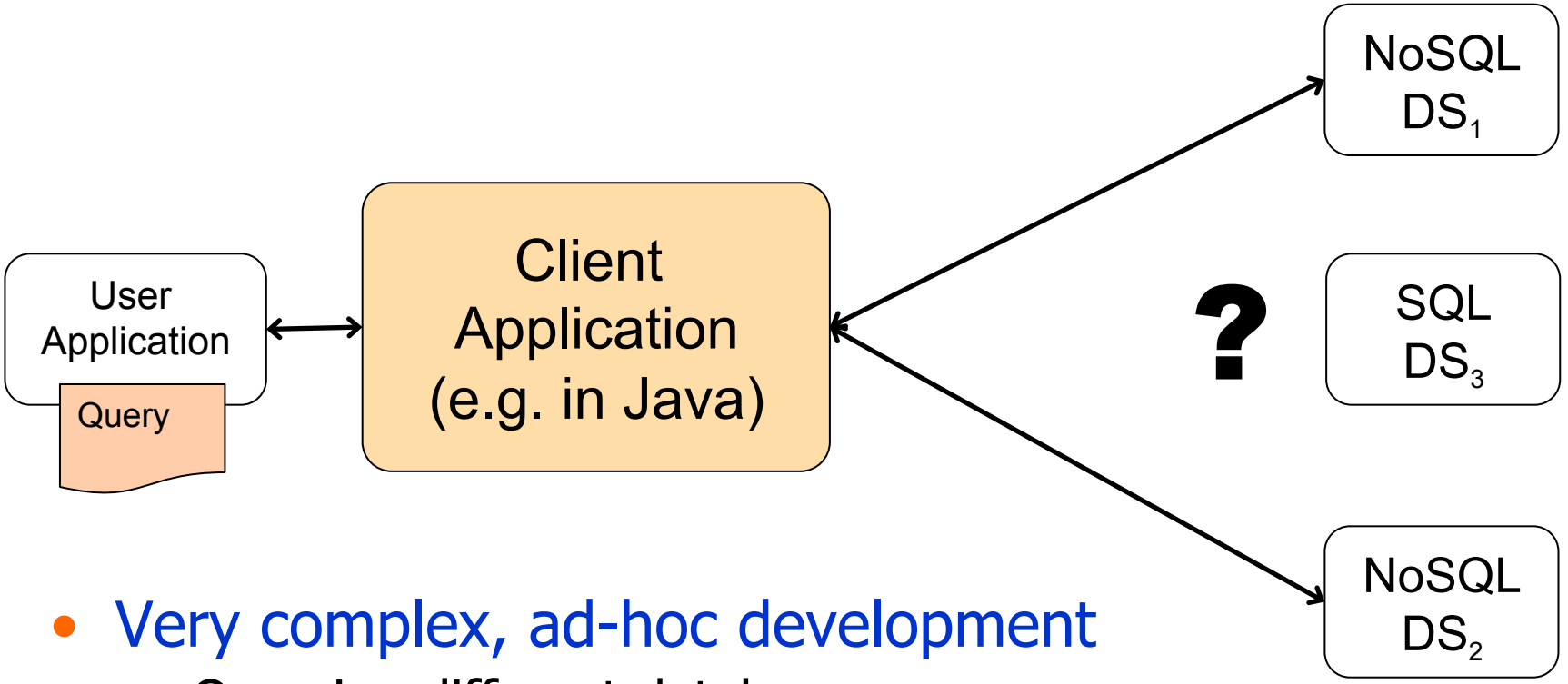
blogs.forbes.com/davefeinleib

General Problem We Address



- Very complex, ad-hoc development
  - Querying different databases
  - Managing intermediate results
  - Delivering (e.g. sorting) the final results

General Problem We Address



- Very complex, ad-hoc development
  - Querying different databases
  - Managing intermediate results
  - Delivering (e.g. sorting) the final results
- Hard to extend
  - What if a new SQL DB appears?

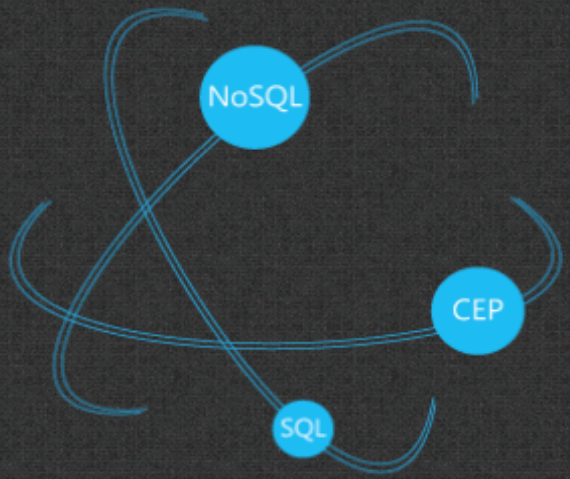
Outline

- The CoherentPaaS IP project
- Related work and background
- CloudMdsQL objectives
- Query language
- Query rewriting
- Use case example
- Experimental validation

FP7 IP project
(2013-2016, 6 M€)

CoherentPaaS

Home | About CoherentPaaS | News | Partners | Case studies | Visitors Corner | Research | Contact



CoherentPaaS

- full ACID coherent
- scalable
- NoSQL
- SQL
- CEP

Coherence

Transactional semantics across cloud data stores

Scalability

Ultra-scalable preserving ACID properties

Simplicity

Programming with a single query language

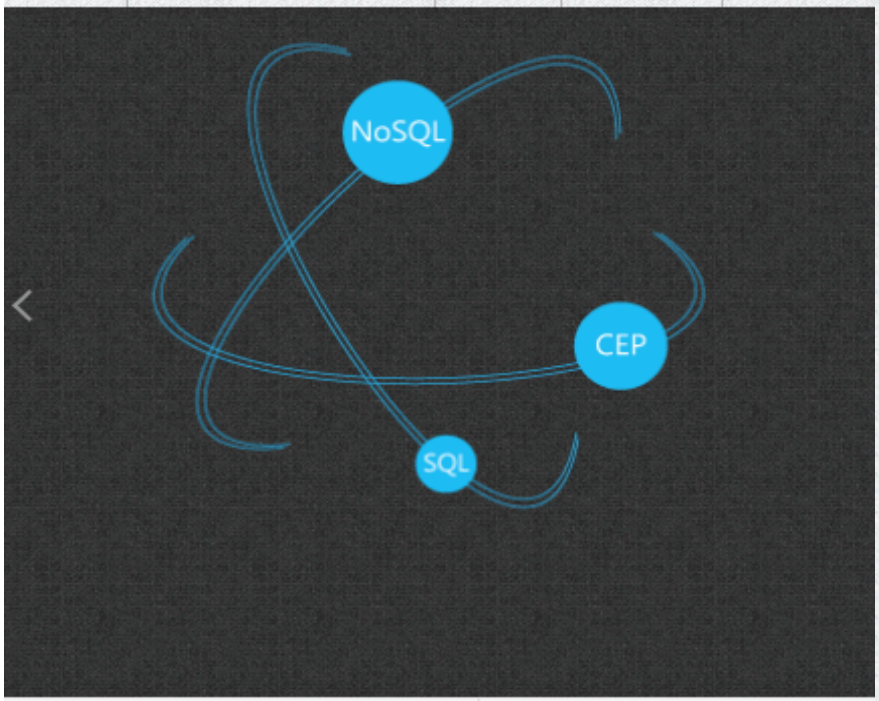
Efficiency

Avoiding ETLs (copying TBs of data across data stores)

FP7 IP project
(2013-2016, 6 M€)

Coh

Home | About CoherentPaaS | News | Partners | Case studi



Coherence

Transactional semantics across cloud data stores

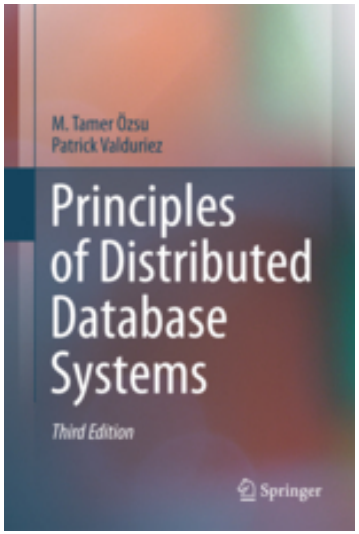
Scalability

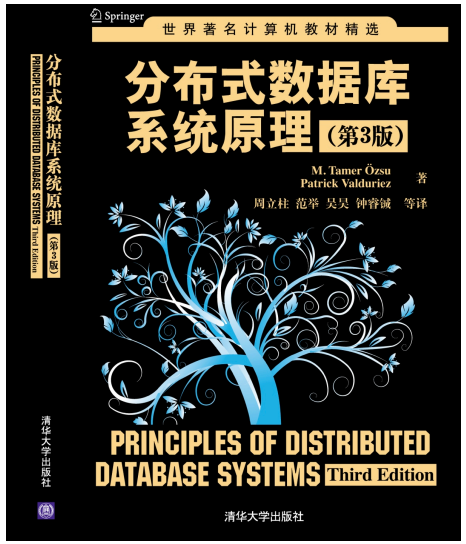
Ultra-scalable preserving ACID properties

Universidad Politecnica de Madrid (Coordinator)	UPM	Spain
Neurocom SA	Neurocom	Greece
INRIA	INRIA	France
Foundation for Research and Technology – Hellas	FORTH	Greece
Institute of Engineering Systems and Computers	INESC	Portugal
Sparsity	Sparsity	Spain
MonetDB	MonetDB	Netherlands
QuartetFS	QuartetFS	United Kingdom
Institute of Communication and Computer Systems	ICCS	Greece
Portugal Telecom Innovaçao	PTIN	Portugal

Related Work

- Multidatabase systems (or federated database systems)
  - A few databases (e.g. less than 10)
    - Corporate DBs
  - Powerful queries (with updates and transactions)
- Web data integration systems
  - Many data sources (e.g. 1000's)
    - DBs or files behind a web server
  - Simple queries (read-only)
- Mediator/wrapper architecture

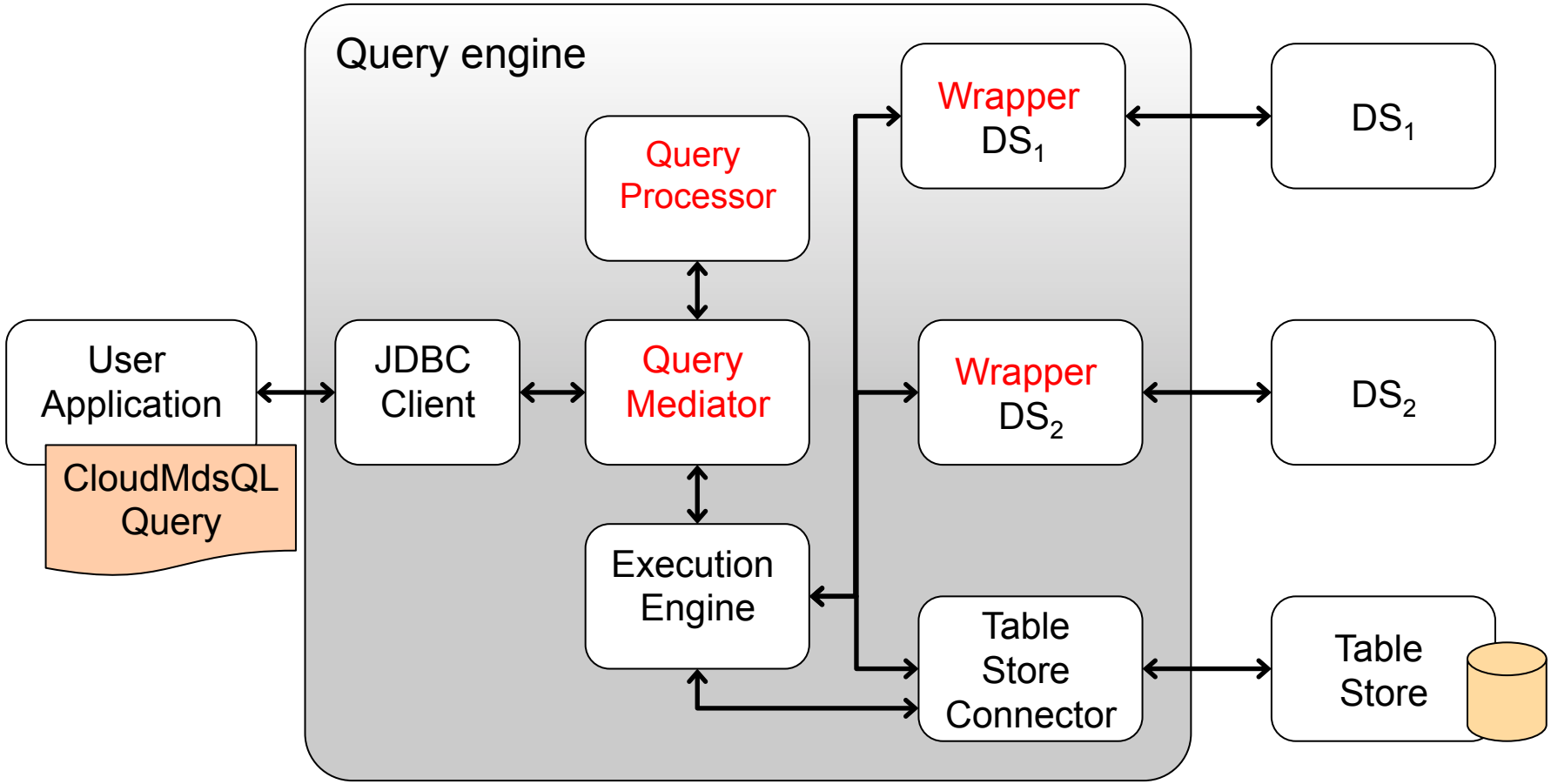




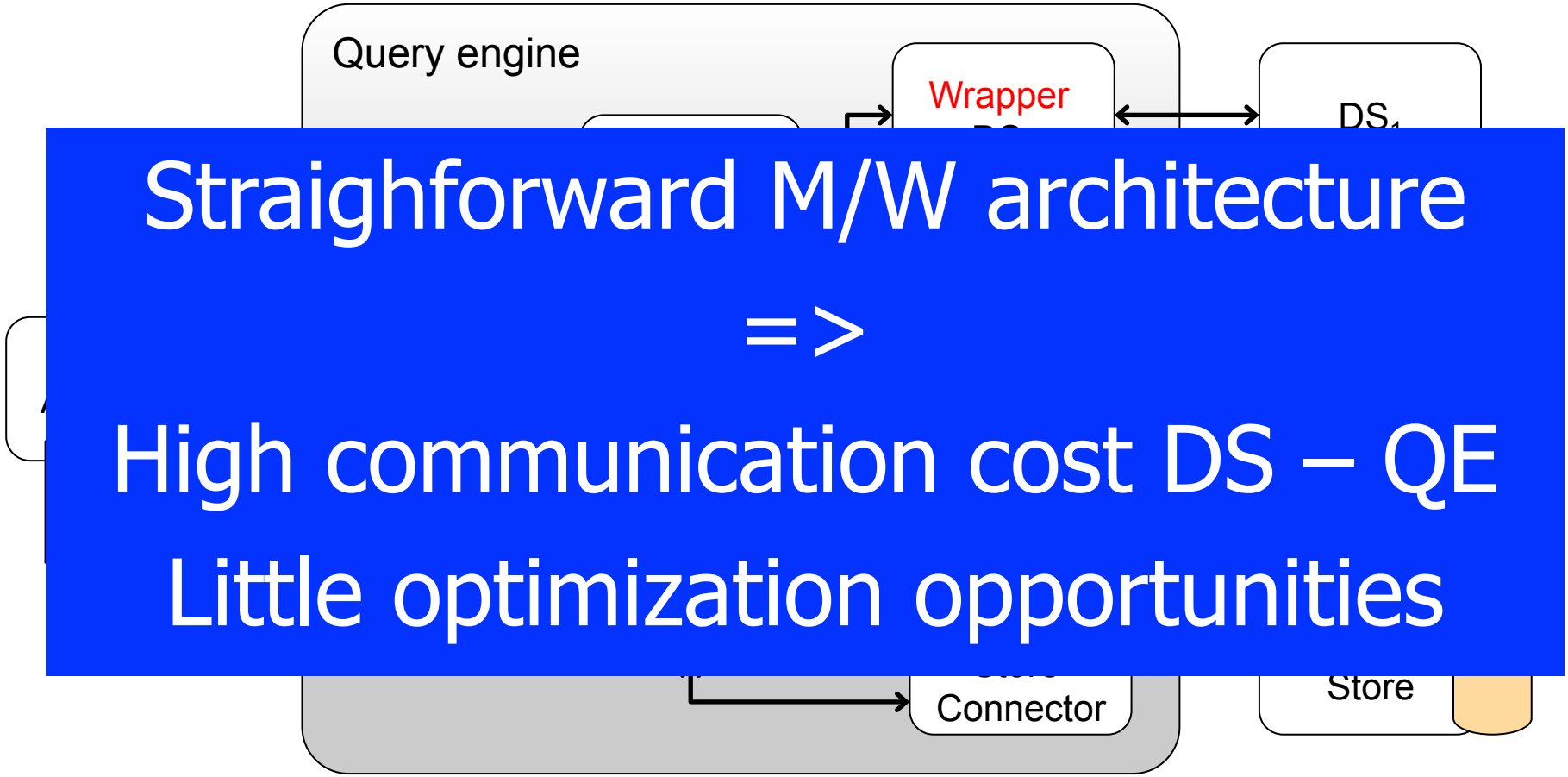
Related Work (cont.)

- Multistore systems
  - Called *Polystores* by M. Stonebraker [The Case for Polystores. Stonebraker's blog. July 2015]
  - Provide integrated access to multiple, heterogeneous cloud data stores such as NoSQL, HDFS and RDBMS
    - E.g. BigDAWG, BigIntegrator, Estocada, Forward, HadoopDB, Odyssey, Polybase, QoX, Spark SQL, etc.
  - Great for integrating structured (relational) data and big data
  - But typically trade data store autonomy for performance or work only for certain categories of data stores (e.g. RDBMS and HDFS)

Centralized Query Engine



Centralized Query Engine



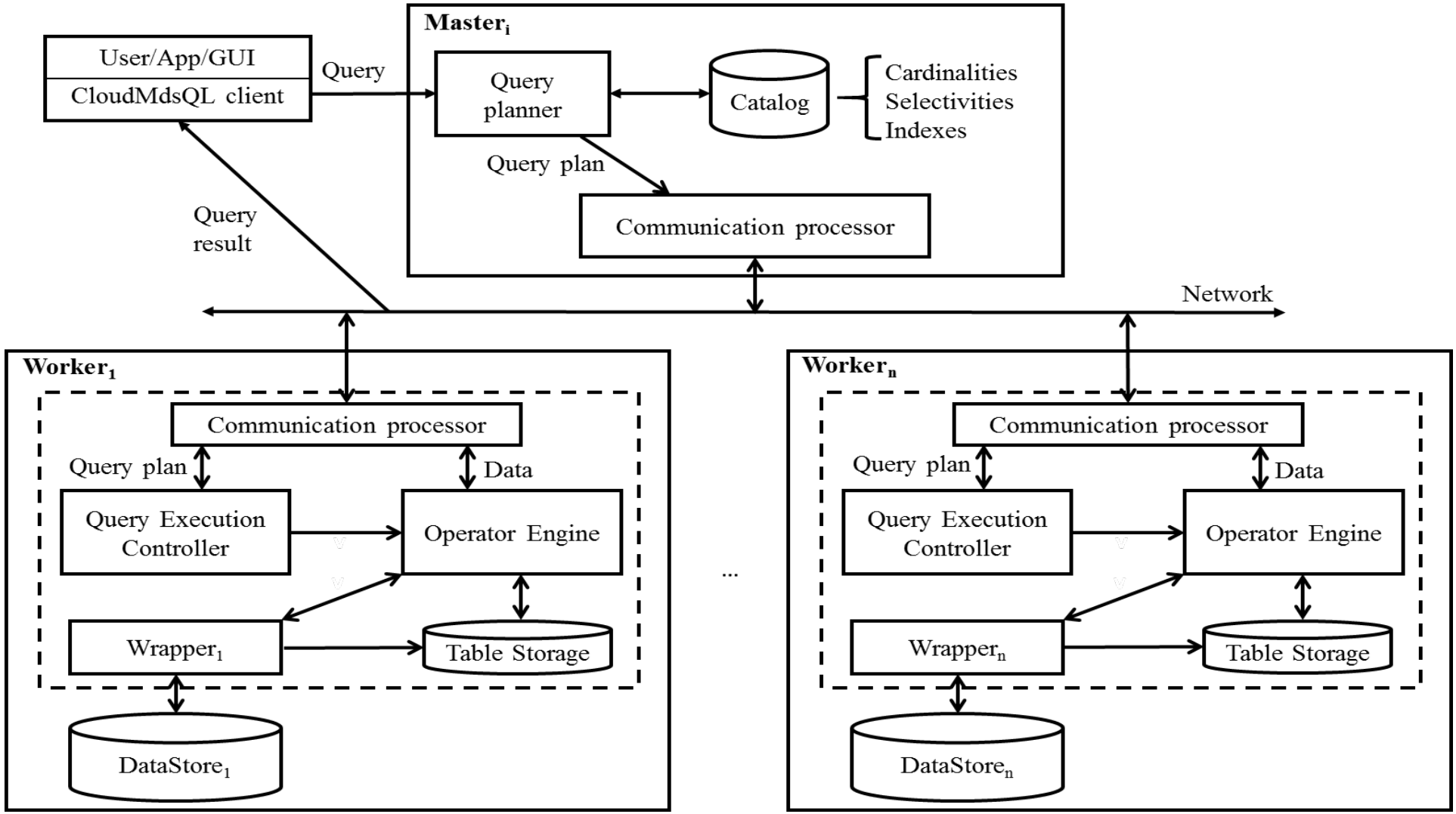
Straighforward M/W architecture

=>

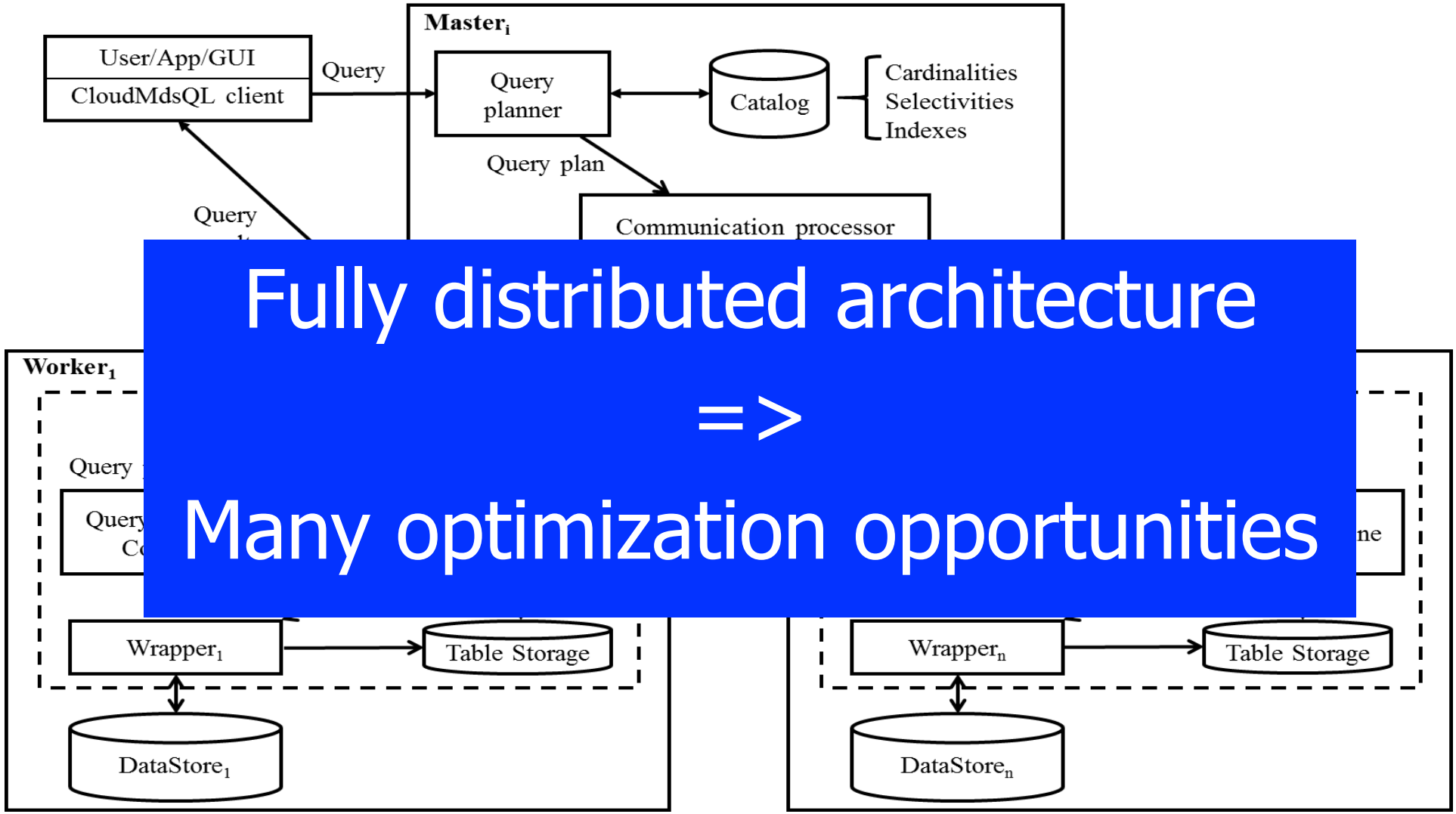
High communication cost DS – QE

Little optimization opportunities

Distributed Query Engine



Distributed Query Engine



Fully distributed architecture

=>

Many optimization opportunities

CloudMdsQL Objectives

- Design an SQL-like query language to query multiple databases (SQL, NoSQL) in a cloud
  - While preserving the autonomy of the data stores
    - This is different from most multistore systems (no autonomy)
- Design a query engine for that language
  - Query processor
    - To produce an efficient execution plan
  - Execution engine
    - To run the query, by calling the data stores and integrating the results
- Validate with a prototype
  - With multiple data stores: Derby, Sparksee, MongoDB,, Hbase, MonetDB, Spark/HDFS, etc.

Issues

- No standard in NoSQL
  - Many different systems
    - Key-value store, big table store, document DBs, graph DBs
- Designing a new language is hard and takes time
  - We should not reinvent the wheel
  - Start simple and useful
- We need to set precise requirements
  - In increasing order of functionality
    - Start simple and useful (and efficient)
  - Guided by the CoherentPaaS project uses cases
    - E.g. bibliography search

Our Design Choices

- Data model: schemaless, table-based
  - With rich data types
    - To allow computing on typed values
  - No global schema and schema mappings to define
- Query language: functional-style SQL[1,2]
  - SQL widely accepted
  - Can represent all query building blocks as functions
    - A function can be expressed in one of the DB languages
  - Function results can be used as input to subsequent functions
  - Functions can transform types and do data-metadata conversion

[1] P. Valduriez, S. Danforth. Functional SQL, an SQL Upward Compatible Database Programming Language. Information Sciences, 1992.

[2] C. Binnig et al. FunSQL: it is time to make SQL functional. EDBT/ICDT, 2012.

CloudMdsQL Data Model

- A kind of nested relational model
  - With JSON flavor
- Data types
  - Basic types: int, float, string, id, idref, timestamp, url, xml, etc. with associated functions (+, concat, etc.)
  - Type constructors
    - Row (called *object* in JSON): an unordered collection of (attribute : value) pairs, denoted by { }
    - Array: a sequence of values, denoted by [ ]
- Set-oriented
  - *A table* is a named collection of rows, denoted by Table-name ()

Data Model – examples*

**Any resemblance to living persons is coincidental*

- Key-value

```
Scientists ({key:"Ricardo", value:"UPM, Spain"},
           {key:"Martin", value:"CWI, Netherlands"})
```

- Relational

```
Scientists ({name:"Ricardo", affiliation:"UPM", country:"Spain"},
           {name:"Martin", affiliation:"CWI", country:"Netherlands"})
Pubs ({id:1, title:"Snapshot isolation", Author:"Ricardo", Year:2005})
```

- Document

```
Reviews ({PID: “1”, reviewer: “Martin”, date: “2012-11-18”,
 tags : ["implementation", "performance"],
 comments :
 [ { when : Date("2012-09-19"), comment : "I like it." },
   {when : Date("2012-09-20"), comment : "I agree with you." } ] })
```

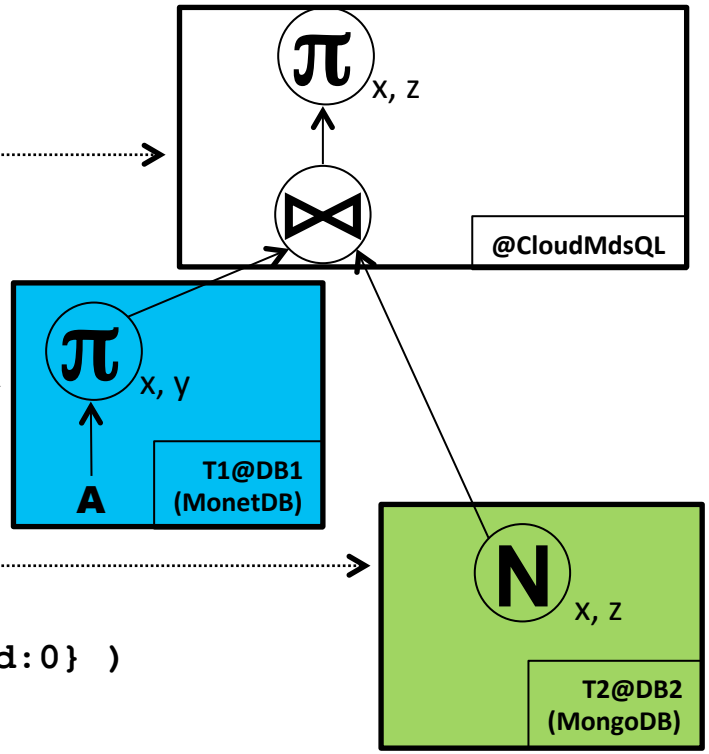
Table Expressions

- Named table expression
  - Expression that returns a table representing a nested query [against a data store]
  - Name and Signature (names and types of attributes)
  - Query is executed in the context of an ad-hoc schema
- 3 kinds of table expressions
  - Native named tables
    - Using a data store's native query mechanism
  - SQL named tables
    - Regular SELECT statements, for SQL-friendly data stores
  - Python named tables
    - Embedded blocks of Python statements that produce tables

CloudMdsQL Example

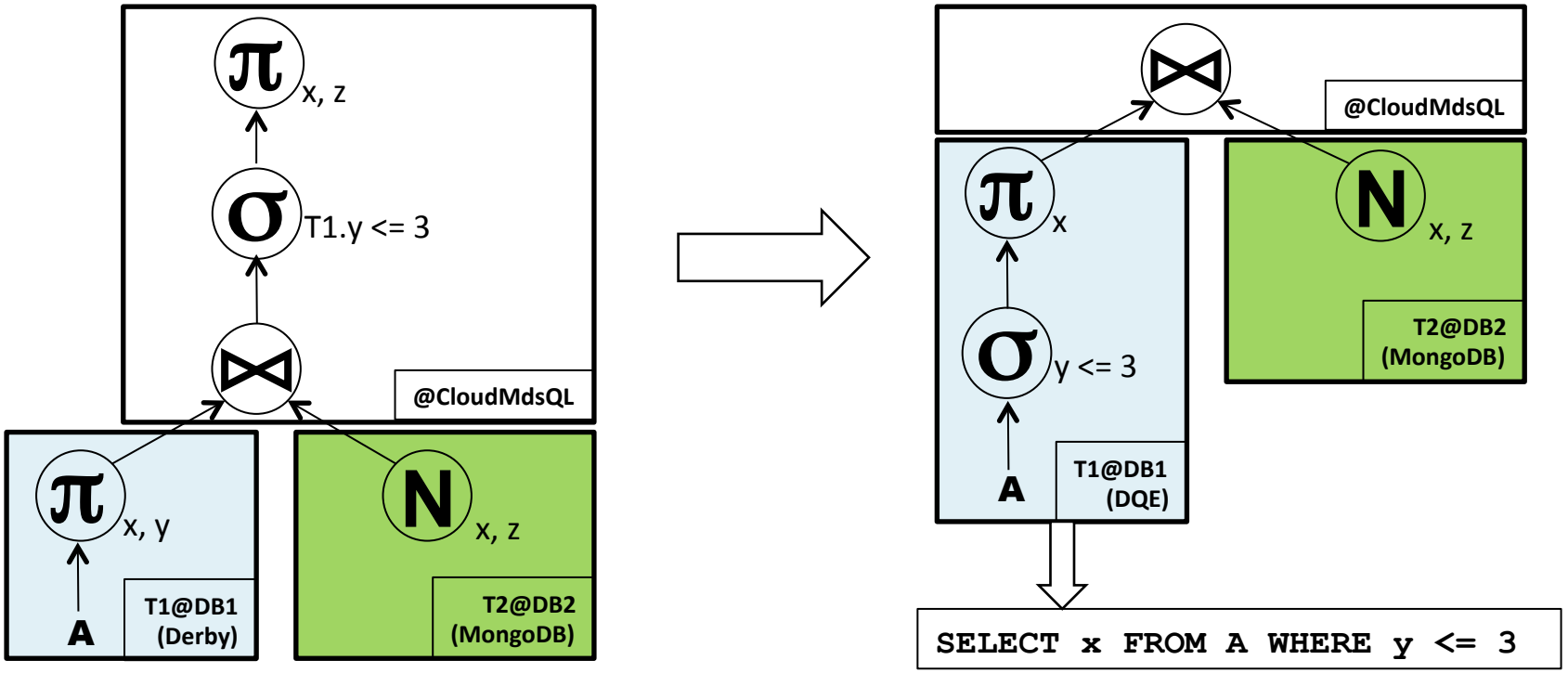
- A query that integrates data from:
  - DB1 – relational (MonetDB)
  - DB2 – document (MongoDB)

```
/* Integration query */
SELECT T1.x, T2.z
FROM T1 JOIN T2
  ON T1.x = T2.x

/* SQL sub-query */
T1(x int, y int)@DB1 =
( SELECT x, y FROM A )

/* Native sub-query */
T2(x int, z string)@DB2 = {*
  db.B.find( {$lt: {x, 10}}, {x:1, z:1, _id:0} )
*}
```

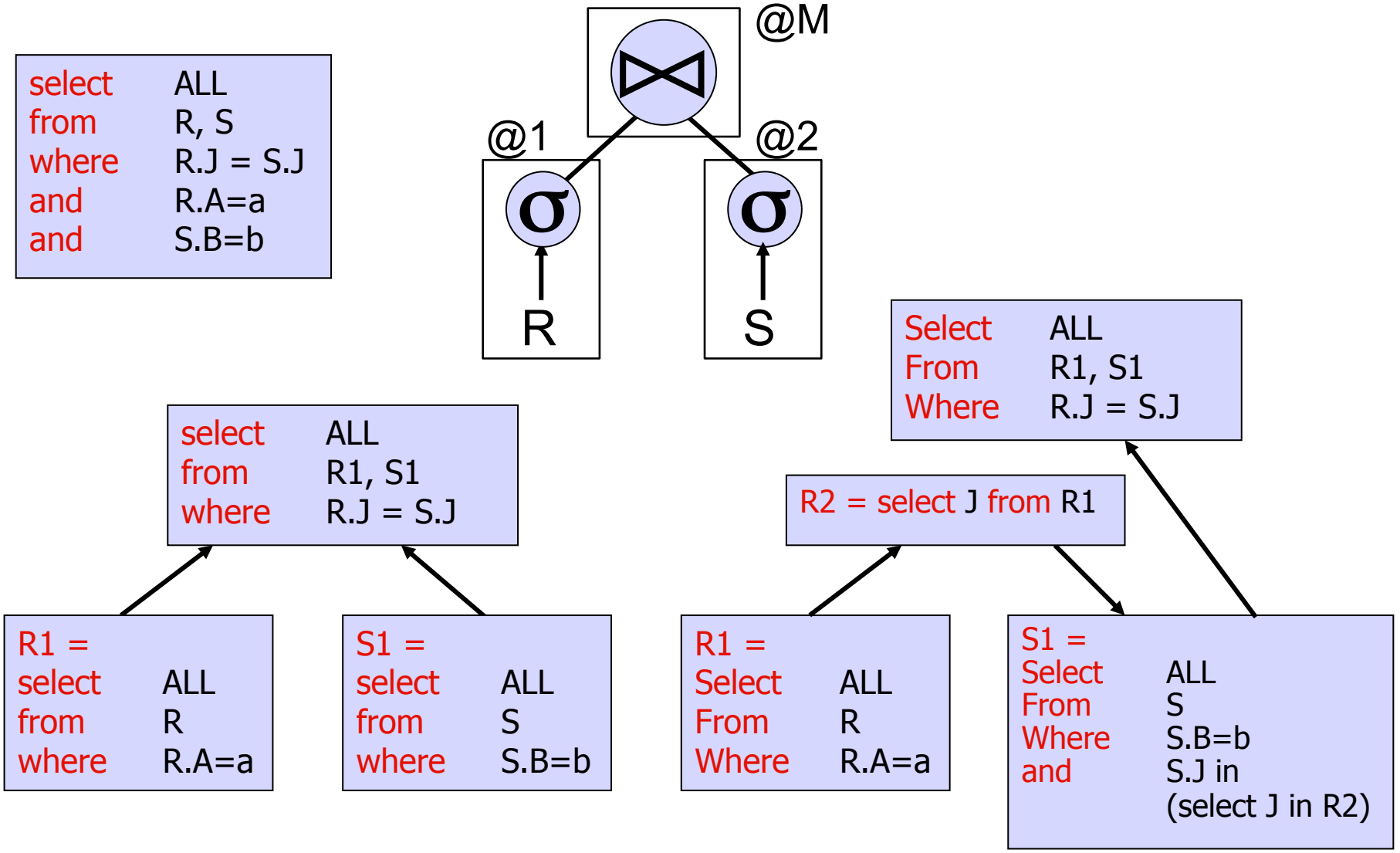


Sub-query Rewriting: selection pushdown

```
T1(x int, y int)@DB1 = ( SELECT x, y FROM A )

T2(x int, z string)@DB2 = {*
  db.B.find( {$lt: {x, 10}}, {x:1, z:1, _id:0} )
*}

SELECT T1.x, T2.z
FROM T1, T2
WHERE T1.x = T2.x AND T1.y <= 3
```

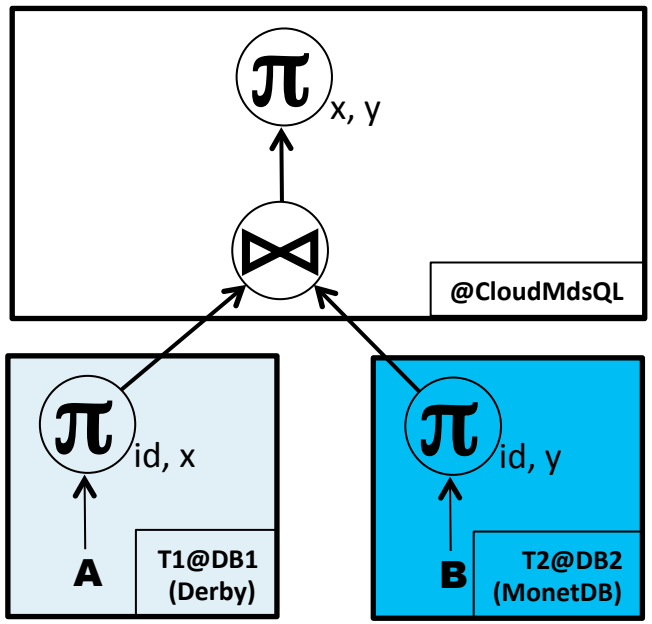


Optimization with Bindjoin

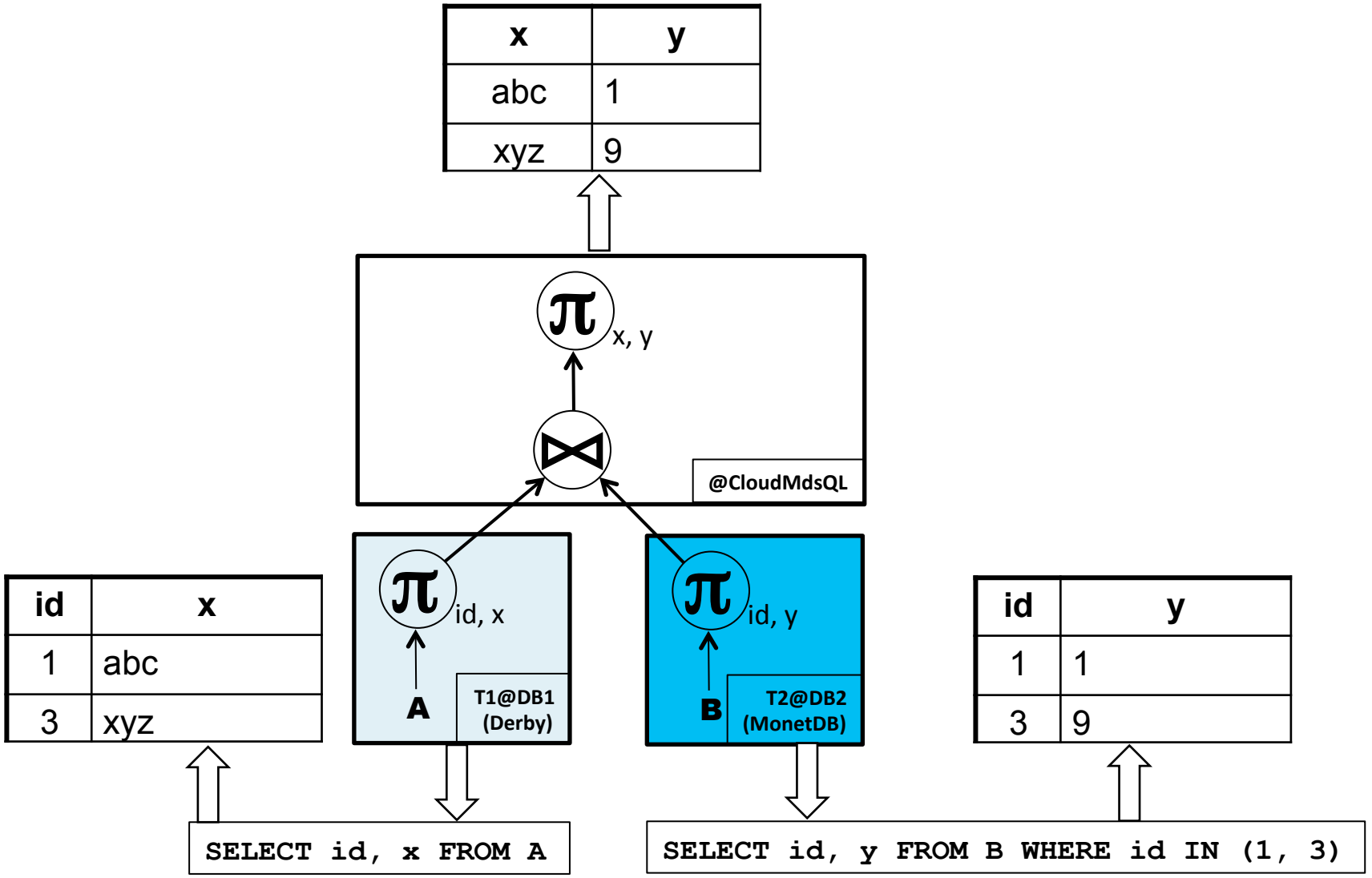


Sub-query Rewriting: bindjoin

```
T1(id int, x string)@DB1 = (SELECT id, x)

T2(id int, y int)@DB2 = (SELECT id, y FROM R2 )

SELECT T1.x, T2.y
FROM T1 BIND JOIN T2 ON T1.id = T2.id
```



Sub-query Rewriting: bindjoin



Use Case Bibliographic App. Example

- 3 data stores
  - Relational
  - Document
  - Graph
- A query that involves integrating data from the three data stores

Example DBs

DB1: a relational DB

Table Scientists (Name char(20), Affiliation char(10), Country char(30)

Scientists

Name	Affiliation	Country
Ricardo	UPM	Spain
Martin	CWI	Netherlands
Patrick	INRIA	France
Boyan	INRIA	France
Larri	UPC	Spain
Rui	INESC	Portugal

Example DBs (cont.)

DB2: a document DB (MongoDB with SQL interface)

Document collection: publications

```
{id: 1, title: 'Snapshot Isolation', author: 'Ricardo', date: '2012-11-10'},
{id: 5, title: 'Principles of DDBS', author: 'Patrick', date: '2011-02-18'},
{id: 8, title: 'Fuzzy DBs', author: 'Boyan', date: '2012-06-29'},
{id: 9, title: 'Graph DBs', author: 'Larri', date: '2013-01-06'}
```

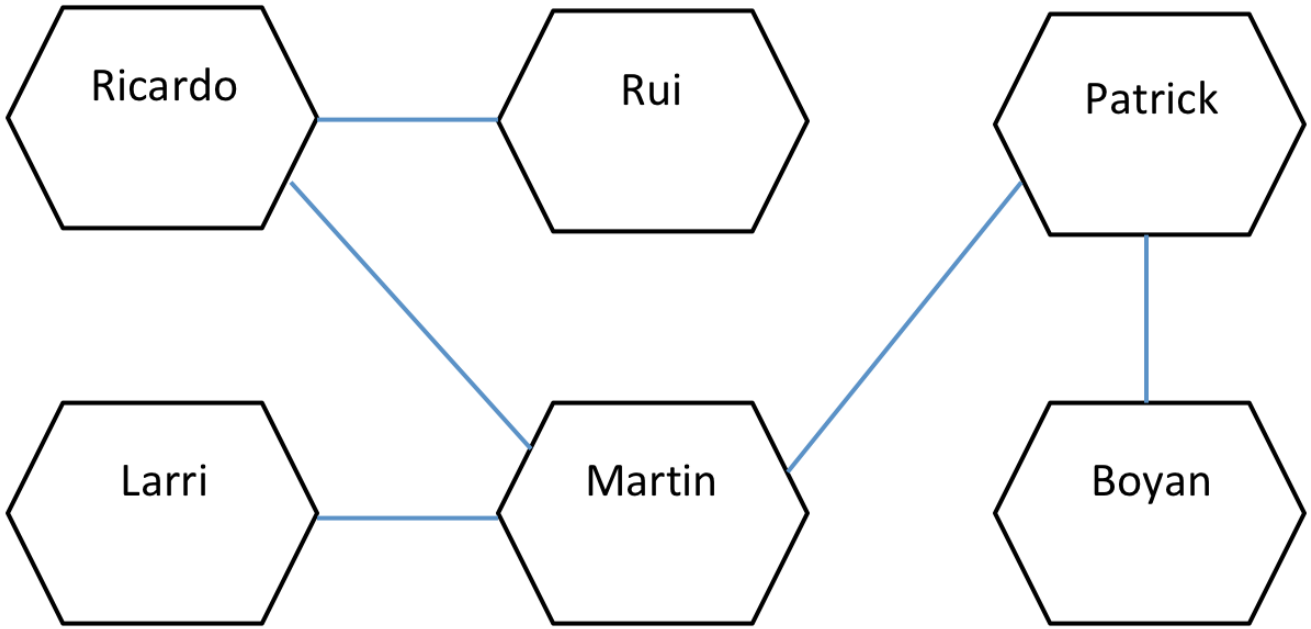
Document collection: reviews

```
{pub_id: “1”, reviewer: “Martin”, date: “2012.11.18”, review: “… text …”},
{pub_id: “5”, reviewer: “Rui”, date: “2013.02.28”, review: “… text …”},
{pub_id: “5”, reviewer: “Ricardo”, date: “2013.02.24”, review: “…text…”},
{pub_id: “8”, reviewer: “Rui”, date: “2012.12.02”, review: “… text …”},
{pub_id: “9”, reviewer: “Patrick”, date: “2013.01.19”, review: “… text …”}
```

Example DBs (cont.)

DB3: a graph DB

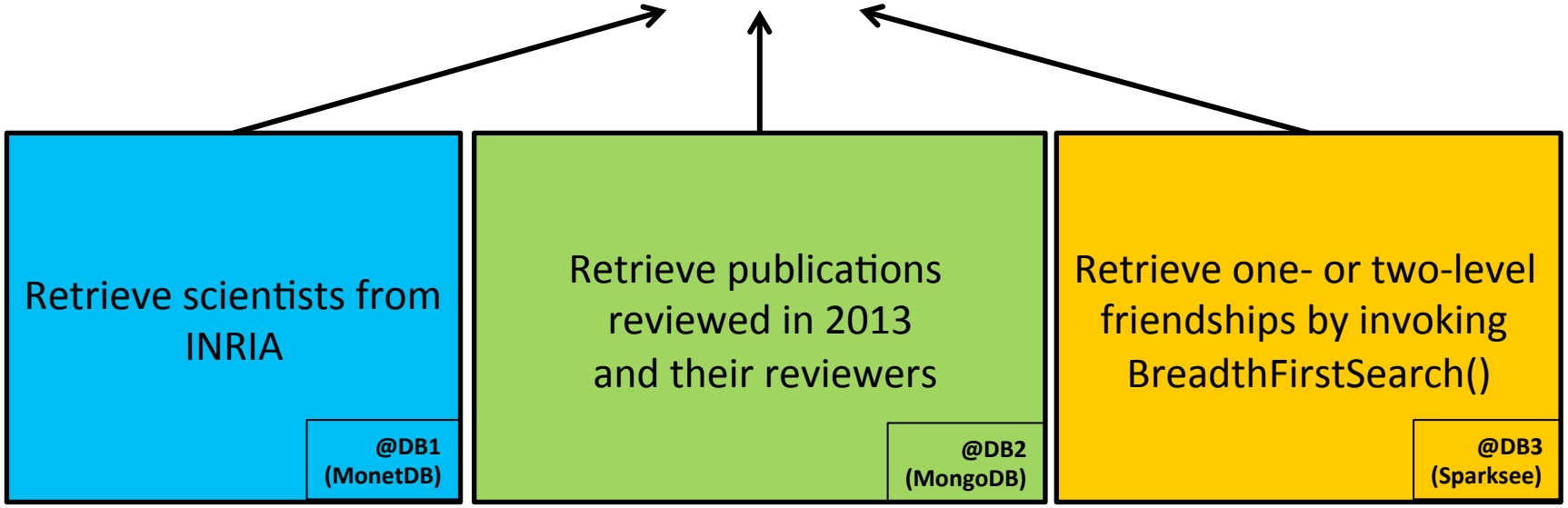
Person (name string, …) is_friend_of Person (name string, …)



CloudMdsQL Query: goal

Find conflicts of interest for papers from INRIA reviewed in 2013

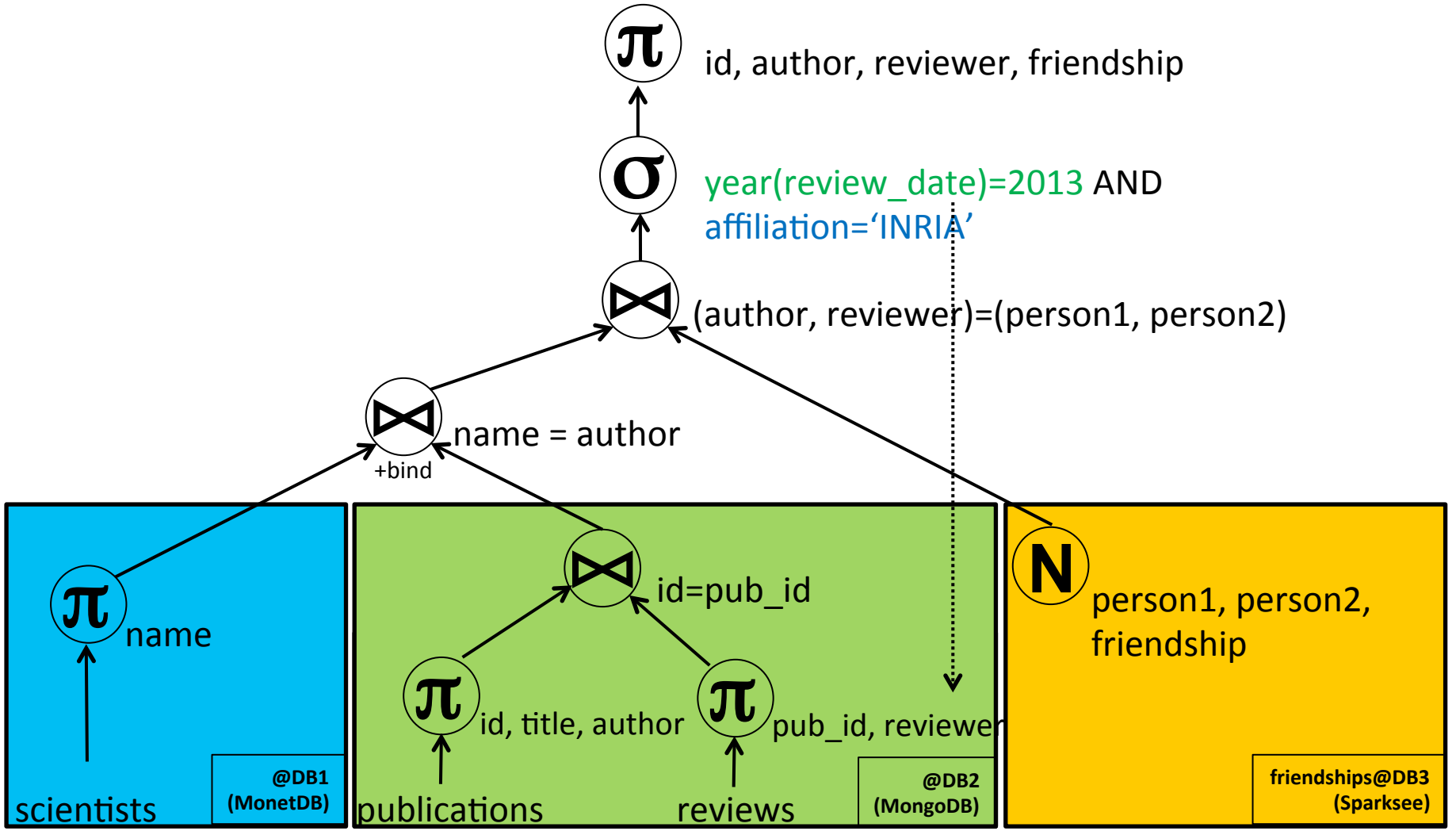
Retrieve papers by scientists from INRIA
that are reviewed in 2013
where the reviewer is a friend or friend-of-friend of the author



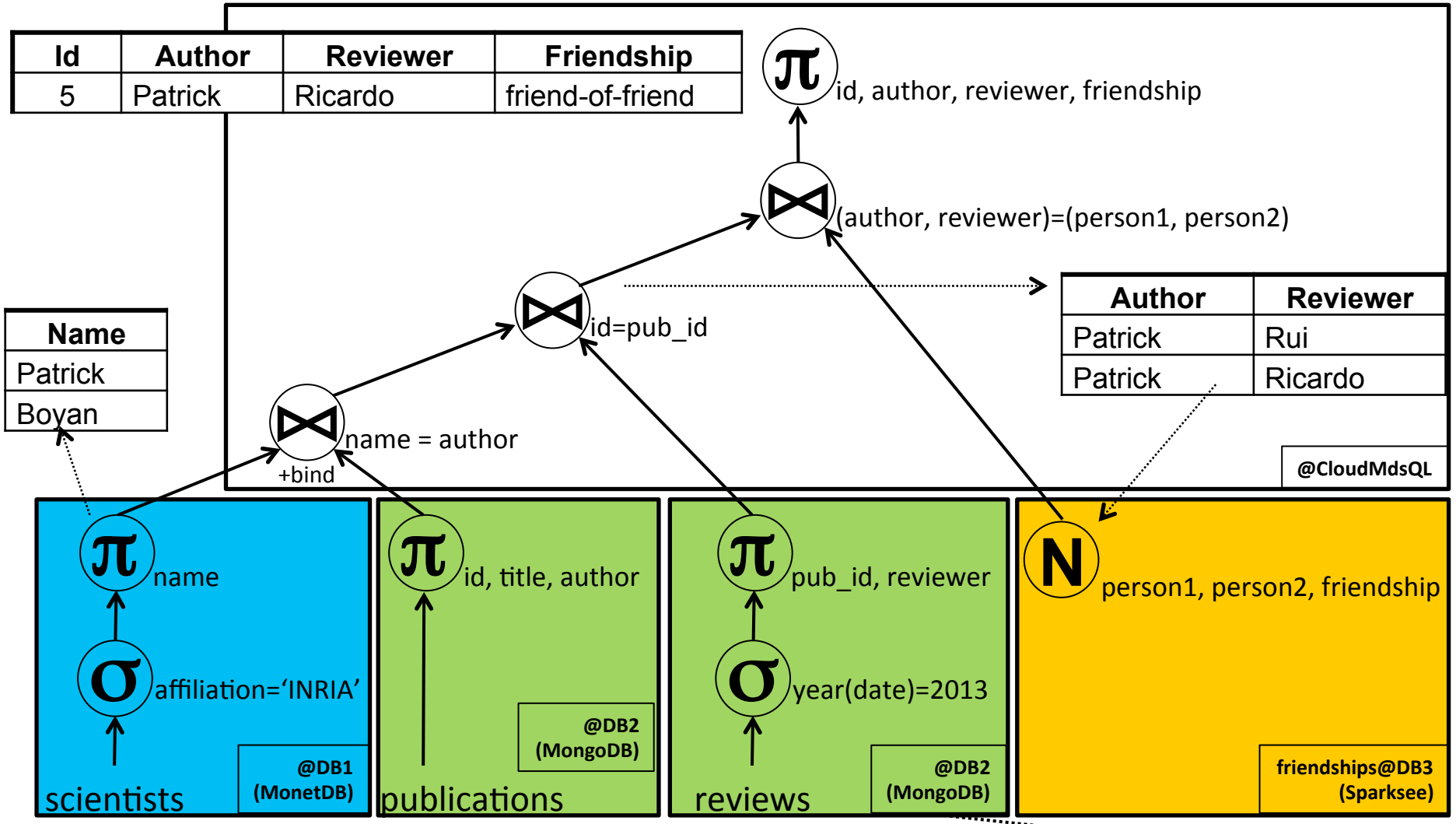
CloudMdsQL Query: expression

```
scientists( name string, aff string )@DB1 = (
  SELECT name, affiliation FROM scientists
)

pubs_revs( p_id, title, author, reviewer, review_date )@DB2 = (
  SELECT p.id, p.title, p.author, r.reviewer, r.date
  FROM publications p, reviews r
  WHERE p.id = r.pub_id
)

friendships( person1 string, person2 string, friendship string
             JOINED ON person1, person2 )@DB3 =
{*
  for (p1, p2) in CloudMdsQL.Outer:
    sp = graph.FindShortestPathByName( p1, p2, max_hops=2)
    if sp.exists():
      yield (p1, p2, 'friend' + '-of-friend' * sp.get_cost())
*}

SELECT pr.id, pr.author, pr.reviewer, f.friendship
FROM scientists s
  BIND JOIN pubs_revs pr ON s.name = pr.author
  JOIN friendships f ON pr.author = f.person1 AND pr.reviewer = f.person2
WHERE pr.review_date BETWEEN '2013-01-01' AND '2013-12-31' AND s.aff = 'INRIA';
```

Initial Query Plan



Rewritten Query Plan



Experimental Validation

- Goal: show the ability of the query engine to optimize CloudMdsQL queries
- Prototype
  - Compiler/optimizer implemented in C++ (using the Boost.Spirit framework)
  - Operator engine (C++) based on the query operators of the Derby query engine
  - Query processor (Java) interacts with the above two components through the Java Native Interface (JNI)
  - The wrappers are Java classes implementing a common interface used by the query processor to interact with them
  - Deployment on a GRID5000 cluster
- Variations of the Bibliographic use case with 3 data stores
  - Relational: Derby
  - Document: MongoDB
  - Graph: Sparksee

Experiments

- Variations of the Bibliographic use case with 3 data stores
  - Relational: Derby
  - Document: MongoDB
  - Graph: Sparksee
- Catalog
  - Information collected through the Derby and MongoDB wrappers
    - Cardinalities, selectivities, indexes
- 5 queries in increasing level of complexity
  - 3 QEPs per query

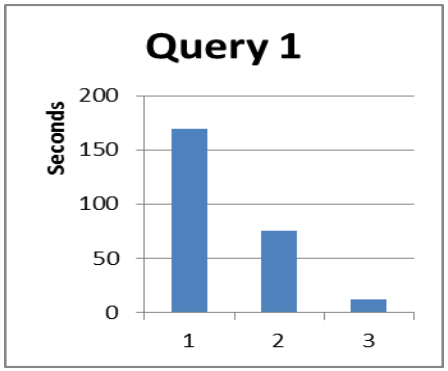
Experimental Results

Query 1

QEP_{11}: $\sigma_{@QE}(R) \bowtie_{@3} P$

QEP_{12}: $\sigma(R) \bowtie_{@3} P$

QEP_{13}: $\sigma(R) \bowtie_{@3} P$

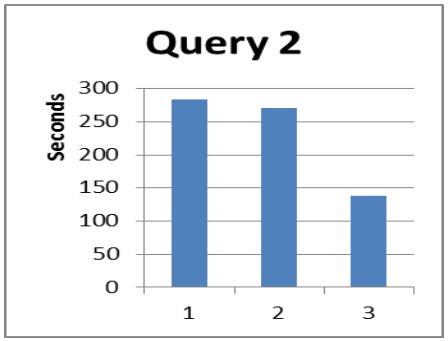


Query 2

QEP_{21}: $(\sigma(S) \bowtie_{@1} P) \bowtie_{@1} \sigma(R)$

QEP_{22}: $(\sigma(S) \bowtie_{@2} P) \bowtie_{@2} \sigma(R)$

QEP_{23}: $(\sigma(S) \bowtie_{@2} P) \bowtie_{@3} \sigma(R)$

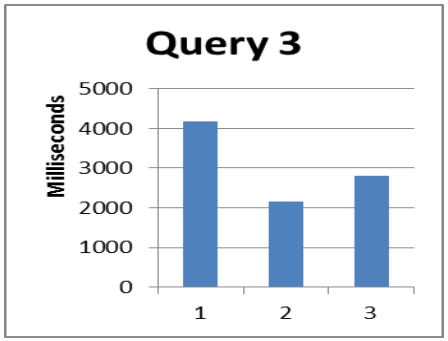


Query 3

QEP_{31}: $((\sigma(Sr) \bowtie_{@3} R) \bowtie_{@3} P) \bowtie_{@3} \sigma(Sa)$

QEP_{32}: $((\sigma(Sa) \bowtie_{@2} P) \bowtie_{@3} R) \bowtie_{@3} \sigma(Sr)$

QEP_{33}: $(\sigma(Sa) \bowtie_{@2} P) \bowtie_{@3} (\sigma(Sr) \bowtie_{@3} R)$



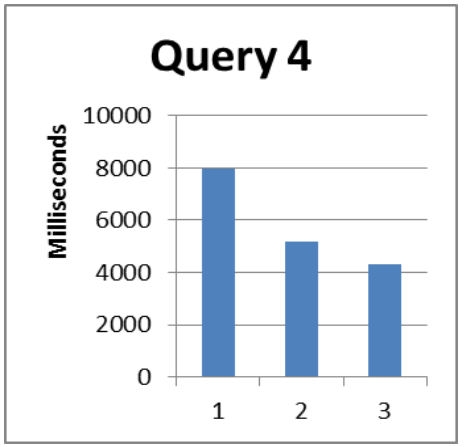
Experiment Results (cont.)

Query 4

QEP_{41}: $(((\sigma(Sr) \bowtie_{@3} R) \bowtie_{@3} P) \bowtie_{@3} F) \bowtie_{@3} \sigma(Sa)$

QEP_{42}: $(((\sigma(Sa) \bowtie_{@2} P) \bowtie_{@3} R) \bowtie_{@3} F) \bowtie_{@3} \sigma(Sr)$

QEP_{43}: $((\sigma(Sa) \bowtie_{@2} P) \bowtie_{@3} (\sigma(Sr) \bowtie_{@3} R)) \bowtie_{@3} F$

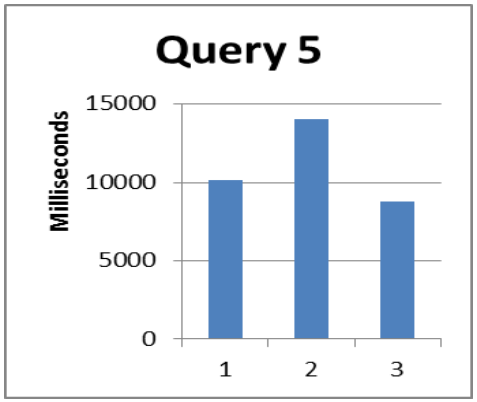


Query 5

QEP_{51}: $(((\sigma(Sr) \bowtie_{@3} R) \bowtie_{@3} P) \bowtie_{@3} F) \bowtie_{@3} \sigma(Sa)$

QEP_{52}: $(((\sigma(Sa) \bowtie_{@2} P) \bowtie_{@3} R) \bowtie_{@3} F) \bowtie_{@3} \sigma(Sr)$

QEP_{53}: $((\sigma(Sa) \bowtie_{@2} P) \bowtie_{@3} (\sigma(Sr) \bowtie_{@3} R)) \bowtie_{@3} F$



CloudMdsQL Contributions

- Advantage
  - Relieves users from building complex client/server applications in order to access multiple data stores
- Innovation
  - Adds value by allowing arbitrary code/native query to be embedded
    - To preserve the expressivity of each data store's query mechanism
  - Provision for traditional distributed query optimization with SQL and NoSQL data stores

References

1. Carlyna Bondiombouy, Boyan Kolev, Oleksandra Levchenko, Patrick Valduriez. Integrating Big Data and Relational Data with a Functional SQL-like Query Language. *DEXA 2015* (extended version to appear in Springer *TLDKS* journal).

2. Carlyna Bondiombouy, Patrick Valduriez. Query Processing in Cloud Multistore Systems: an overview. *Int. Journal of Cloud Computing*, 38 pages, to appear, 2016.

3. Boyan Kolev, Patrick Valduriez, Carlyna Bondiombouy, Ricardo Jiménez-Peris, Raquel Pau, José Pereira. CloudMdsQL: Querying Heterogeneous Cloud Data Stores with a Common Language. *Distributed and Parallel Databases*, online, 43 pages, 2015.

4. Boyan Kolev, Carlyna Bondiombouy, Oleksandra Levchenko, Patrick Valduriez, Ricardo Jiménez-Peris, Raquel Pau, José Pereira. Design and Implementation of the CloudMdsQL Multistore System. *CLOSER 2016.*

5. Boyan Kolev, Carlyna Bondiombouy, Patrick Valduriez, Ricardo Jiménez-Peris, Raquel Pau, José Pereira. The CloudMdsQL Multistore System. *SIGMOD 2016.*