
# Triples in the clouds

**Zoi Kaoudi and Ioana Manolescu**
**Inria Saclay, France**

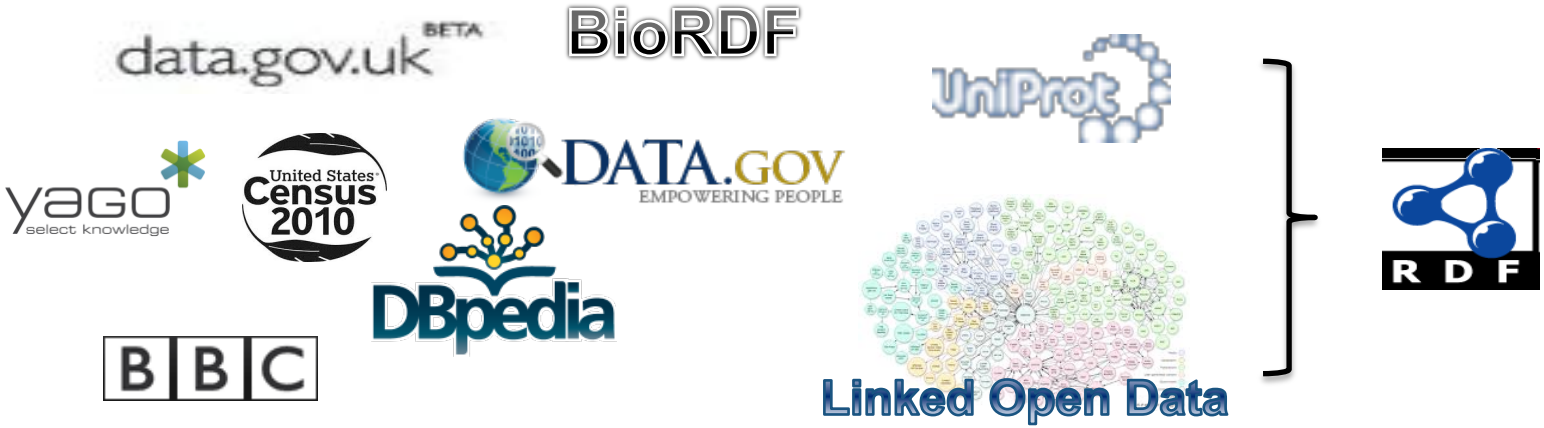
**ICDE seminar 2013**
http://pages.saclay.inria.fr/zoi.kaoudi/rdfcloud_icde13.pdf

# Where is RDF found today?

- Semantic Web applications
- Commercial database systems (Oracle, IBM, etc.)
- Open Data applications
- Linked Data applications

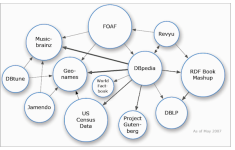
} Linked Open Data

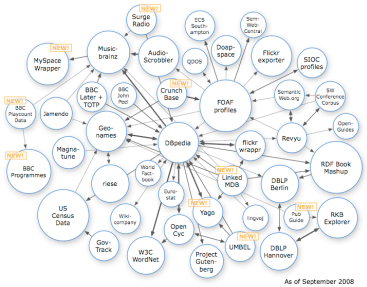
- Many available data sources in RDF

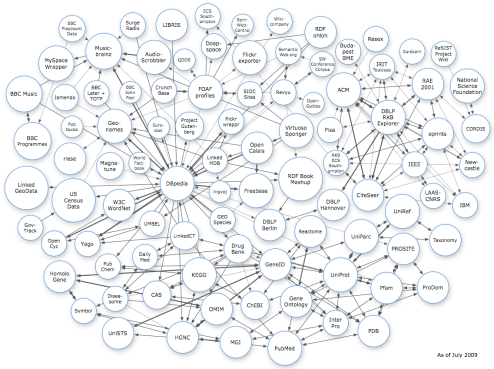


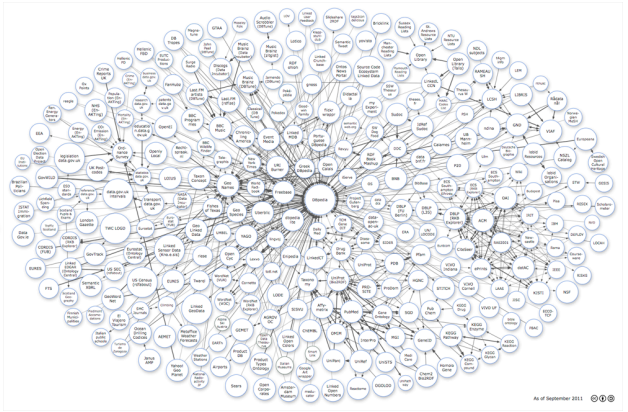
# RDF is growing

## LOD cloud









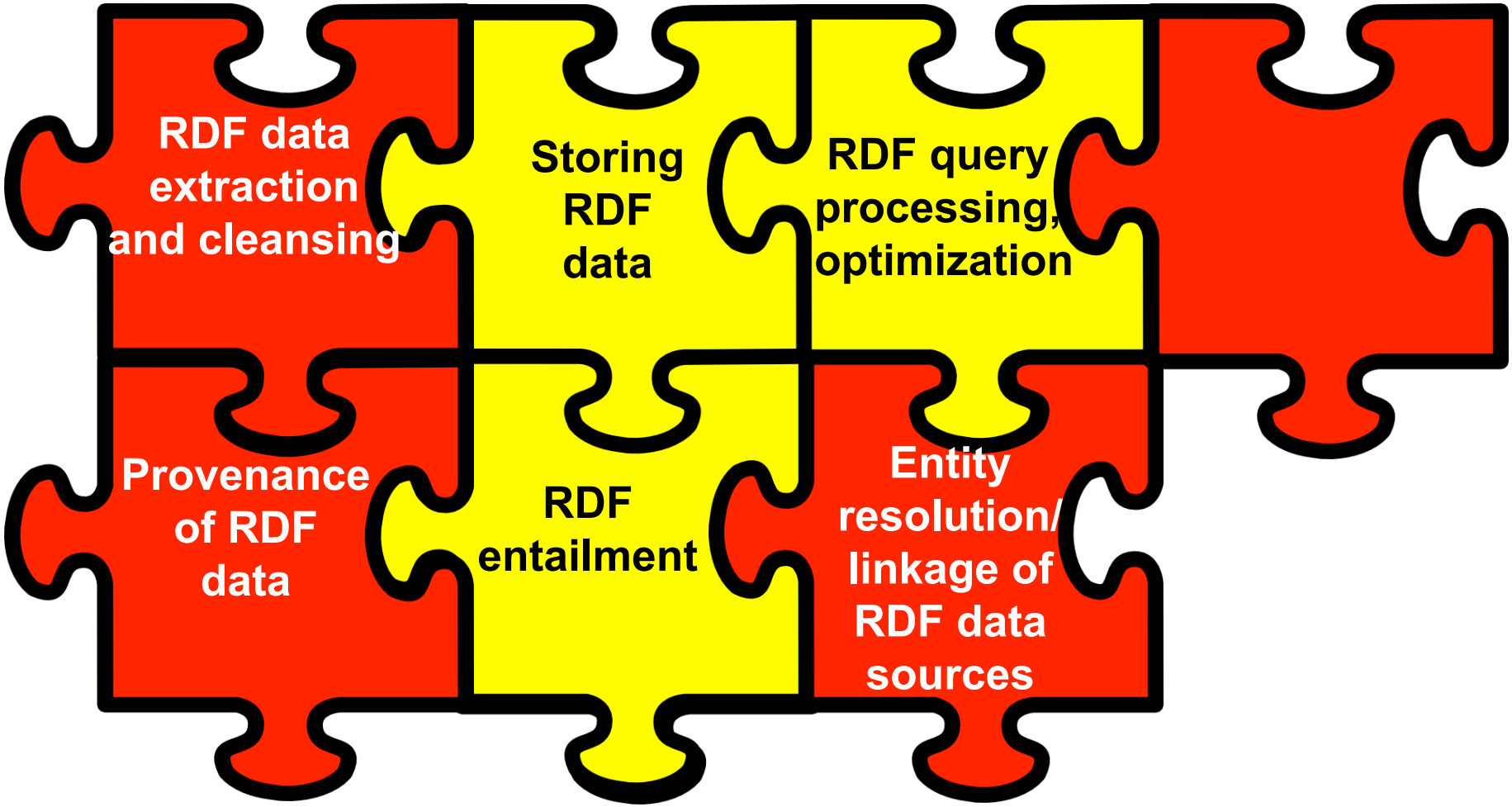
2007 2008 2009 2011

# RDF data sources in numbers

| | | |
|---|---|---|
| DBpedia | RDF-encoded Wikipedia | 1.89 billion triples |
| BioRDF | RDF-encoded biological data | 2.7 billion triples |
| DATA.GOV EMPOWERING PEOPLE | US government data in RDF | 5 billion triples |
| A new application award Semantic Web Challenge http://challenge.semanticweb.org | crawled Web data | 2 billion triples |
| United States Census 2010 | US population statistics | 1 billion triples |
| yago select knowledge | facts from Wikipedia, Wordnet, Geonames | 0.12 billion triples |
| LOD cloud | **Linked Open Data cloud** | **31 billion** triples, Sept. 2011 |

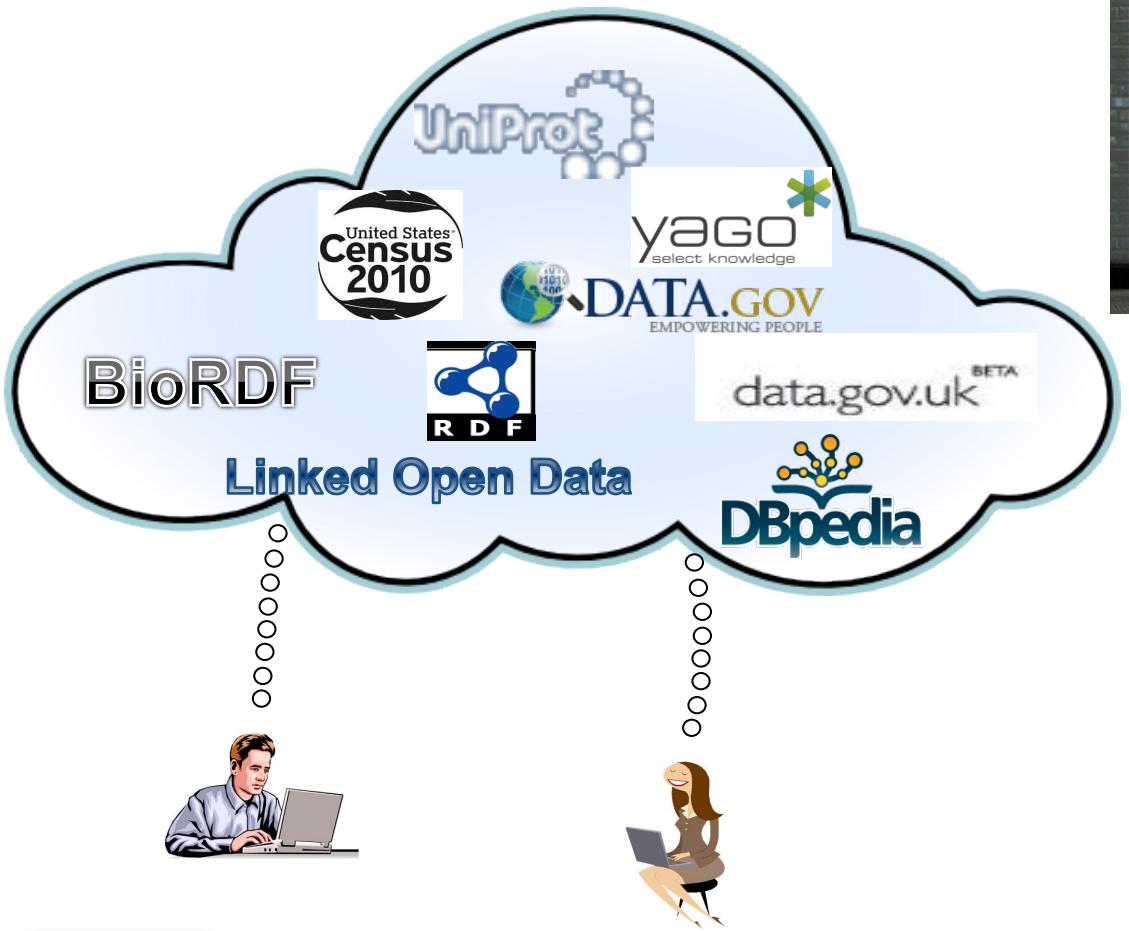
… and many more … http://www.w3.org/wiki/DataSetRDFDumps
http://datahub.io/

# RDF data management tasks



# Managing large volumes of RDF data

## Cloud computing

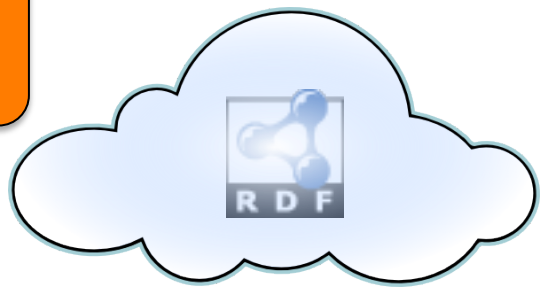




- Scalability
- Elasticity
- Fault-tolerance

# What this seminar is about

## Triples in the clouds




- Storage RDF data in the cloud
- Evaluation of RDF queries in the cloud
- Inference of RDF data in the cloud

# Outline

1. RDF basics
2. Cloud basics
3. Analysis of RDF systems in the cloud
   - Storage
   - Querying
   - Inference
4. Open issues

# 1

# RDF BASICS

# Resource Description Framework (RDF)

- RDF: representation of **resources** on the Web identified by Universal Resource Identifiers (**URIs**) http://live.dbpedia.org/page/Pablo_Picasso
- RDF data: facts = **triples** (s, p, o)


- **Literals**: strings, numbers, dates, etc.
  - Only in the object position
- RDF **schema** (RDFS): specifies concepts, properties and relationships between them http://live.dbpedia.org/ontology/Artist

```
(dbpedia:Pablo_Picasso, rdf:type, dbpedia-owl:Artist)
(dbpedia-owl:Artist, rdfs:subClassOf, dbpedia-owl:Person)
```

Notation: `(:Pablo_Picasso, type, :Artist)`
`(:Artist, sc, :Person)`

# Running example: RDF database (triple syntax)

N-TRIPLE syntax

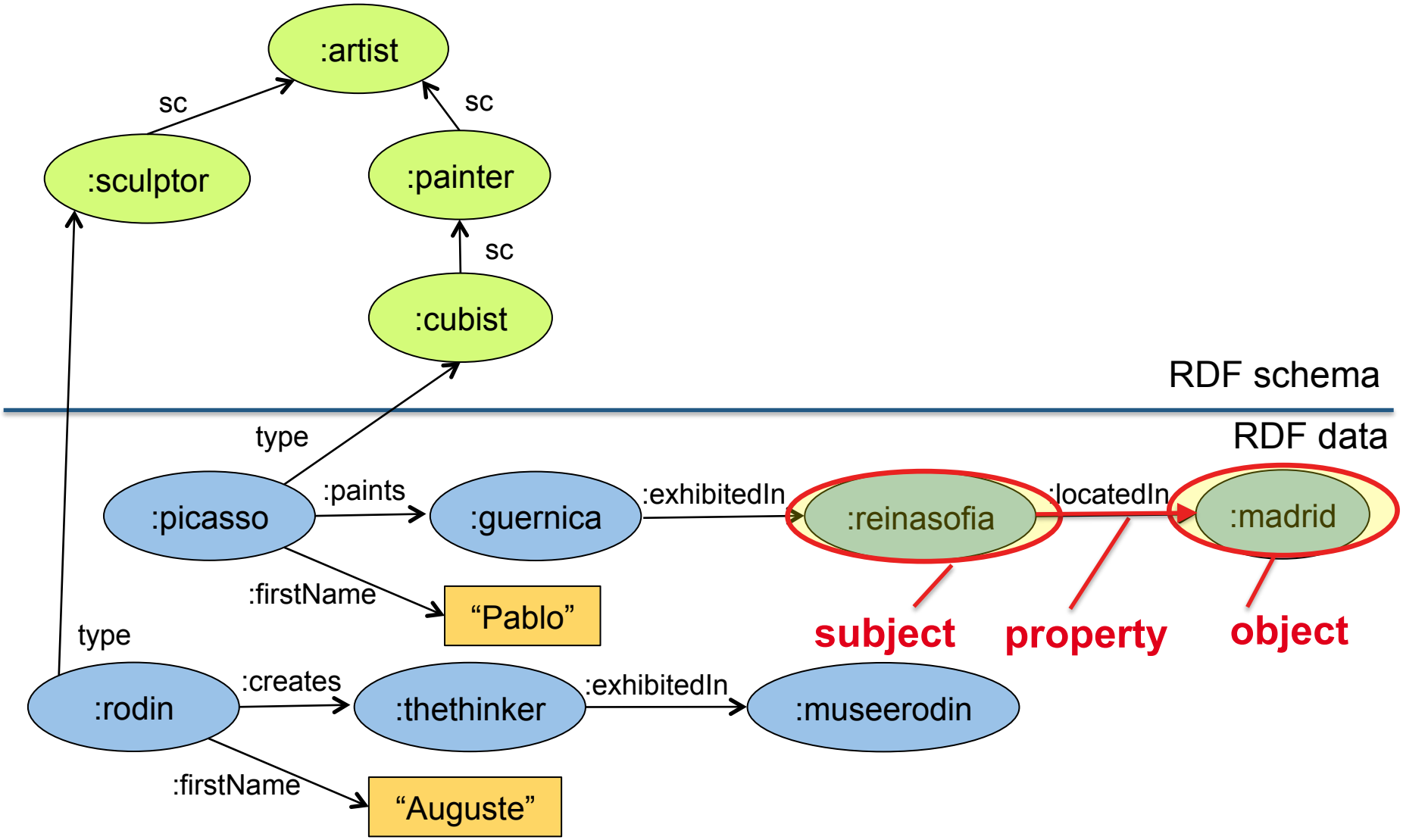
```
:picasso type :cubist .
:picasso :firstName "Pablo" .
:picasso :paints :guernica .
:guernica :exhibitedIn :reinasofia .
:reinasofia :locatedIn :madrid .
:rodin type :sculptor .
:rodin :firstName "Auguste" .
:rodin :creates :thethinker .
:thethinker :exhibitedIn :museerodin .
```

RDF data

```
:sculptor sc :artist .
:painter sc :artist .
:cubist sc :painter.
```

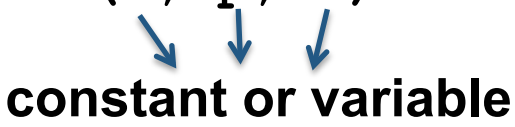
RDF Schema

# Running example: RDF database



# SPARQL Query Language

- Standard language to query RDF data
- SQL-based syntax
- Basic building block: triple pattern `(s, p, o)`



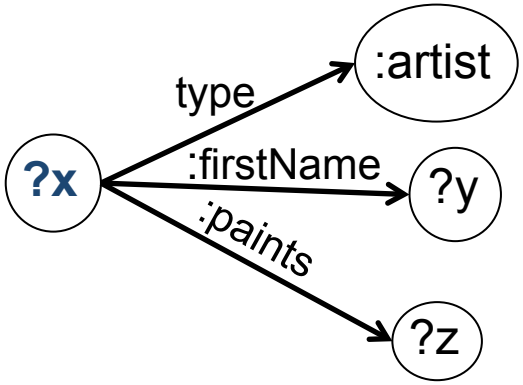

- Basic graph pattern (BGP) queries
  - conjunctive triple pattern queries
  - $?x_1, ?x_2, \ldots, ?x_n : (s_1, p_1, o_1) \wedge \ldots \wedge (s_k, p_k, o_k)$


- SPARQL 1.1: W3C recommendation, March 2013

# SPARQL examples

## Star query

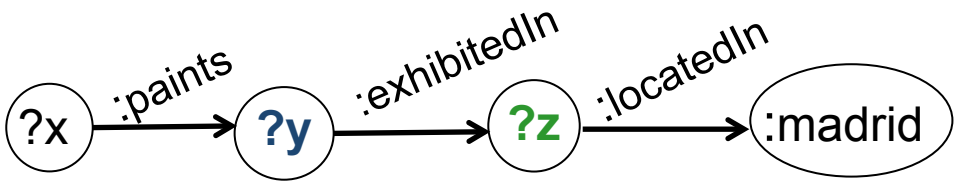
```
SELECT ?y ?z
WHERE {?x type :artist .
       ?x :firstName ?y .
       ?x :paints ?z .}
```




?y ?z: (?x type :artist ∧
(?x :firstName ?y) ∧
(?x :paints ?z)

## Path query

```
SELECT ?x ?y ?z
WHERE {?x :paints ?y .
       ?y :exhibitedIn ?z .
       ?z :locatedIn :madrid.}
```




?x ?y ?z: (?x :paints ?y) ∧
(?y :exhibitedIn ?z) ∧
(?z :locatedIn :madrid)

# Centralized RDF stores

- How do we usually store RDF in centralized environment?
- Approaches relying on RDBMs
  - Big **triple tables**

Triples

| subject | property | object |
|---|---|---|
| :picasso | type | :cubist |
| :picasso | :firstName | “Pablo” |
| picasso | :paints | :guernica |
| :guernica | :exhibitedIn | :reinasofia |
| :reinasofia | :locatedIn | :madrid |
| :rodin | type | :sculptor |
| :rodin | :creates | :thethinker |
| … | … | … |

# Centralized RDF stores

- How do we usually store RDF in centralized environment?
- Approaches relying on RDBMs
  - Big triple tables
  - **Property tables** [Alexaki01, Wilkison06]

Artists

| subject | type | firstName | paints | creates |
|---|---|---|---|---|
| :picasso | :cubist | “Pablo” | :guernica | null |
| :rodin | :sculptor | “Auguste” | null | :thethinker |

Artifacts

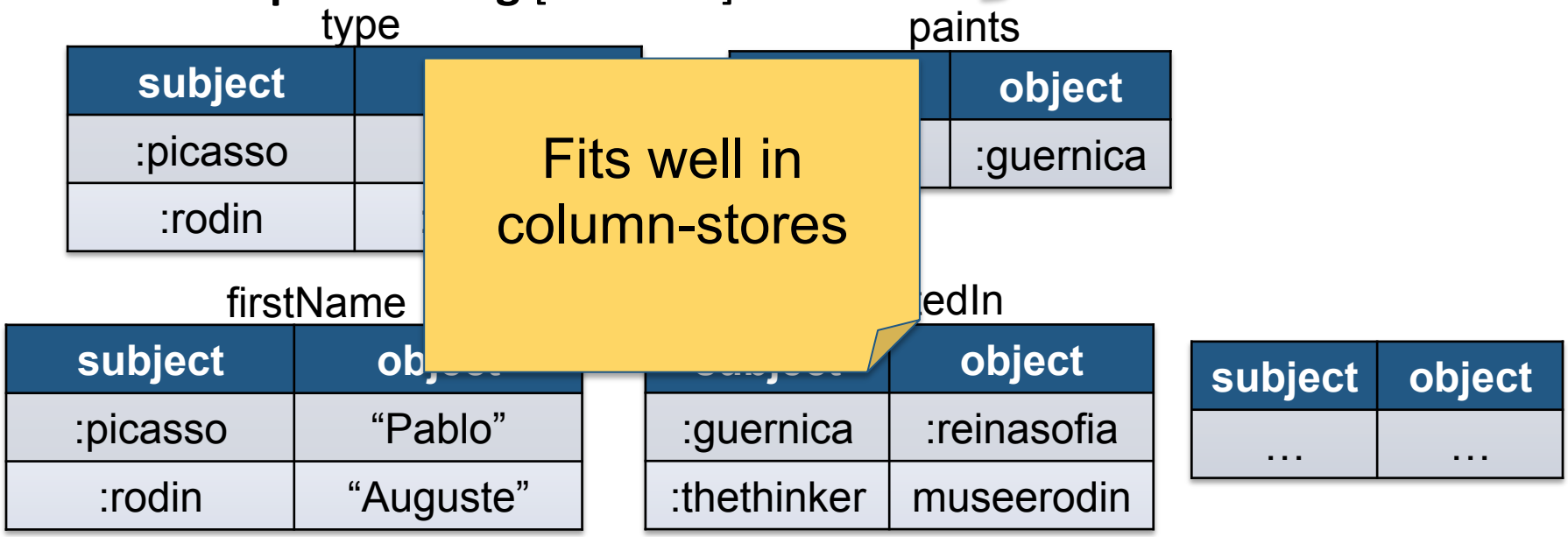
| subject | exhibitedIn |
|---|---|
| :guernica | :reinasofia |
| :thethinker | :museerodin |

| subject | … |
|---|---|
| … | … |
| … | … |

# Centralized RDF stores

- How do we usually store RDF in centralized environment?
- Approaches relying on RDBMs
  - Big triple tables
  - Property tables [Alexaki01, Wilkison06]
  - **Vertical partitioning** [Abadi07]

Benchmarking [Theoharis05]



# Centralized RDF stores

- How do we usually store RDF in centralized environment?
- Approaches relying on RDBMs
  - Big triple tables
  - Property tables [Alexaki01, Wilkison06]
  - Vertical partitioning [Abadi07]
- Native RDF stores
  - RDF-3X [Neumann10], Hexastore [Weiss08]
  - 6 indexes (1 for each permutation of s p o)
  - compressed indices
  - aggregated indices

# Dictionary encoding

- RDF contains long strings of URIs

  Size of triples

| min | max | avg |
|---|---|---|
| 80 bytes | 2100 bytes | 240 bytes |

Source http://gromgull.net/blog/category/semantic-web/billion-triple-challenge/

Dictionary

| id | resource |
|---|---|
| 1 | :picasso |
| 2 | :rodin |
| 3 | type |
| 4 | :cubist |
| 5 | :firstName |
| 6 | :guernica |
| … | … |

Triples

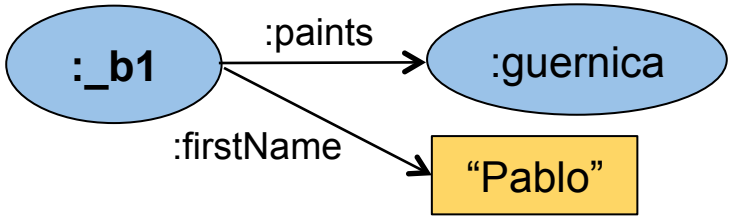
| pro | |
|---|---|
| t | |
| :firs | |
| :p | |
| :exh | |
| :loc | |
| type | :sculptor |
| :creates | :thethinker |
| … | … |

Triples

| subject | property | object |
|---|---|---|
| 1 | 3 | 4 |
| 1 | 5 | 10 |
| 1 | 7 | 6 |
| 6 | 12 | 9 |
| … | … | … |

# RDF is not just a 3-attribute table

- Blank nodes
  - unknown/anonymous resources
  - existential variables




  - for more on blank nodes: [Mallea11]
- Schema and data can be queried together
- Properties can also be described
  - `(:paints, domain, :painter)`
  - joins between properties and subjects/objects
- Inference
- and more … [Hayes04]

# Inference in RDF

- RDF schema statements and set of entailment rules lead to implicit (entailed) RDF data

Sample rules

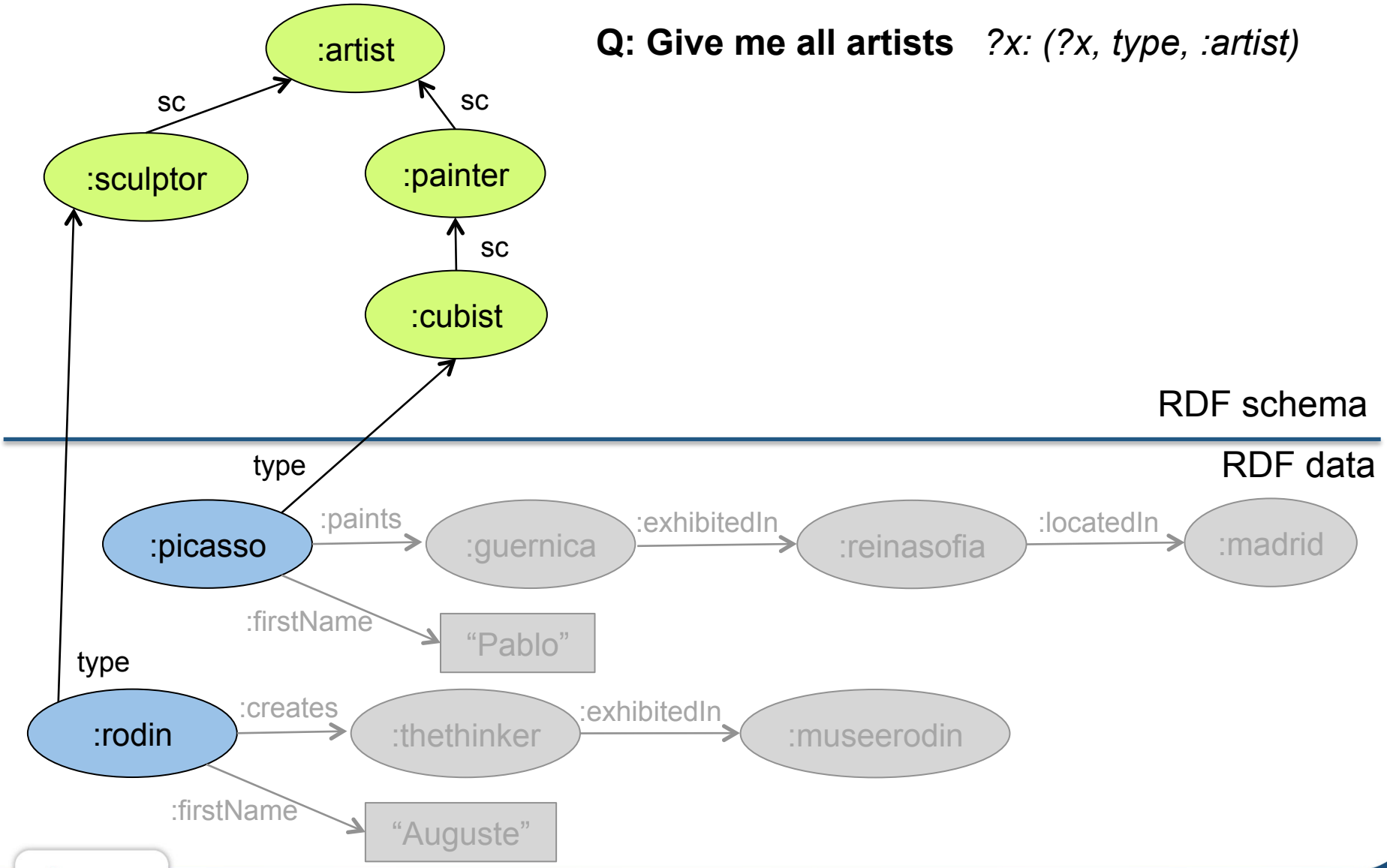
$(X, type, A) \wedge (A, sc, B) \rightarrow (X, type, B)$
$(X, P, O) \wedge (P, domain, A) \rightarrow (X, type, A)$

***(:picasso, :paints, :guernica) ∧ (:paints, domain, :painter) → (:picasso, type, :painter)***

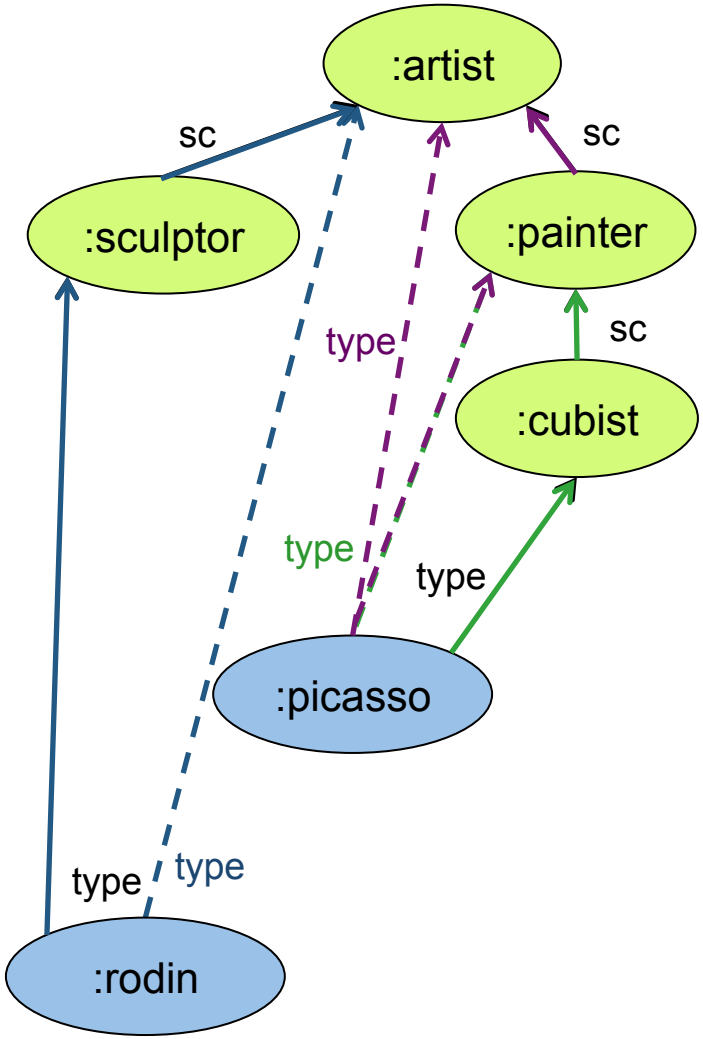
- RDFS entailment : *Given an RDF(S) database DB and a triple t, is t entailed from DB (or does t logically follows from DB)?*

# Entailed triples and RDF querying

**Q: Give me all artists** *?x: (?x, type, :artist)*



# Entailed triples and RDF querying




**Q: Give me all artists** *?x: (?x, type, :artist)*

**A:**

| ?x |
|---|
| - |

no entailment

Rule:

*(X, type, A) ∧ (A, sc, B) → (X, type, B)*

**Implicit / entailed triples**

***(:rodin, type, :artist)***

***(:picasso, type, :painter)***

***(:picasso, type, :artist)***

**A:**

| ?x |
|---|
| :picasso |
| :rodin |

with entailment

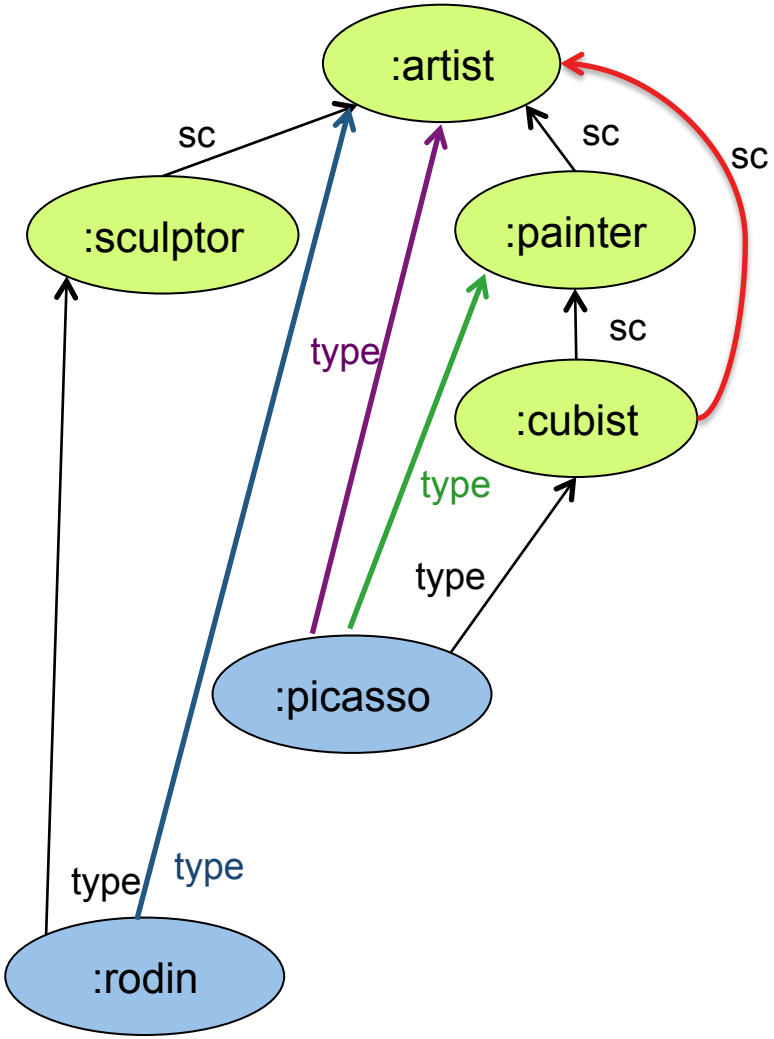
# RDFS entailment techniques

- RDFS closure
  - compute all implicit triples and store them in the database
- Query reformulation
  - transform the query so that you get all the answers (even the ones that come from implicit triples)
- Hybrid approaches
  - compute the schema closure, and then reformulate the query
  - magic sets: based on the queries, precompute and store only the implicit triples that are required for the answer
- For a comparison on these techniques:
  - centralized [Goasdoué13]
  - distributed [Kaoudi13]

# RDFS entailment – RDFS closure

- Before querying

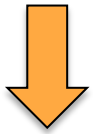
| subject | property | object |
|---|---|---|
| :picasso | type | :cubist |
| :rodin | type | :sculptor |
| :sculptor | sc | :artist |
| :cubist | sc | :painter |
| :painter | sc | :artist |
| **:cubist** | **sc** | **:painter** |
| **:rodin** | **type** | **:artist** |
| **:picasso** | **type** | **:painter** |
| **:picasso** | **type** | **:artist** |



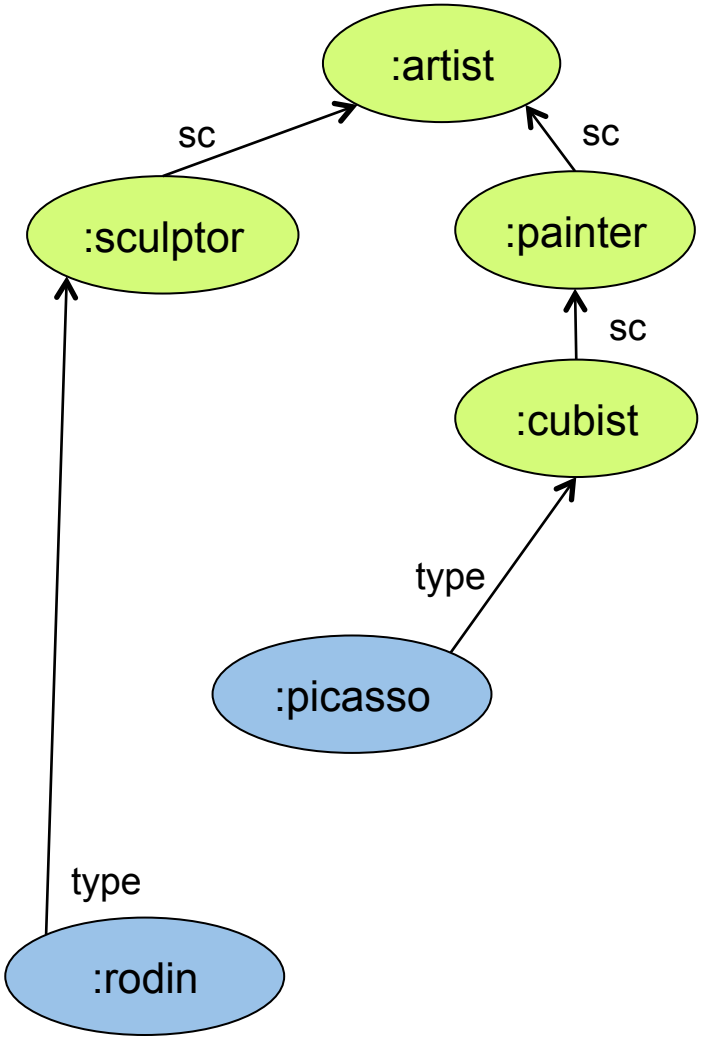
# RDFS entailment – query reformulation

- At query time

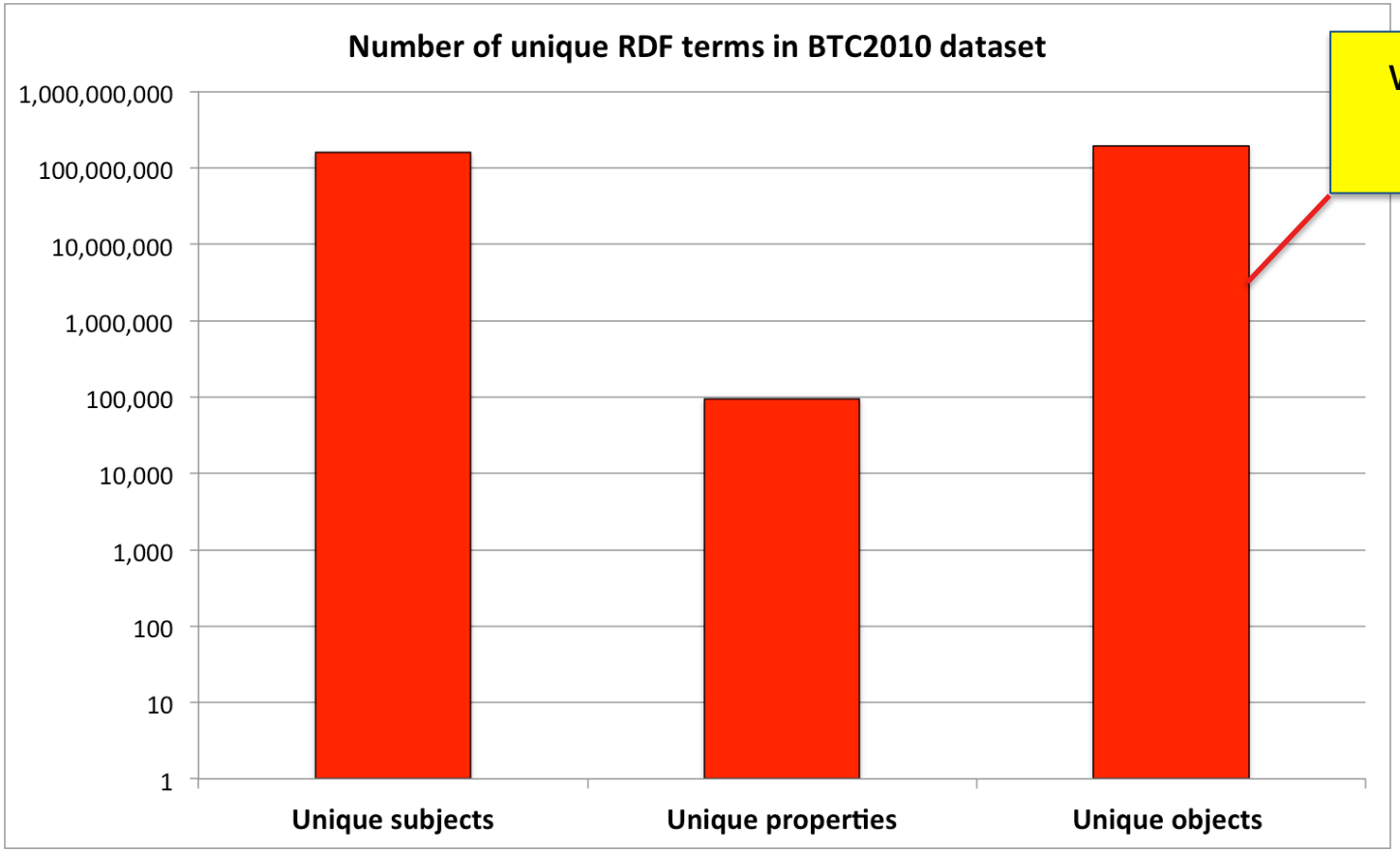
?x: (?x, type, :artist)



?x: (?x, type, :artist) ∨
(?x, type, :sculptor) ∨
(?y, type, :painter) ∨
(?x, type, :cubist)



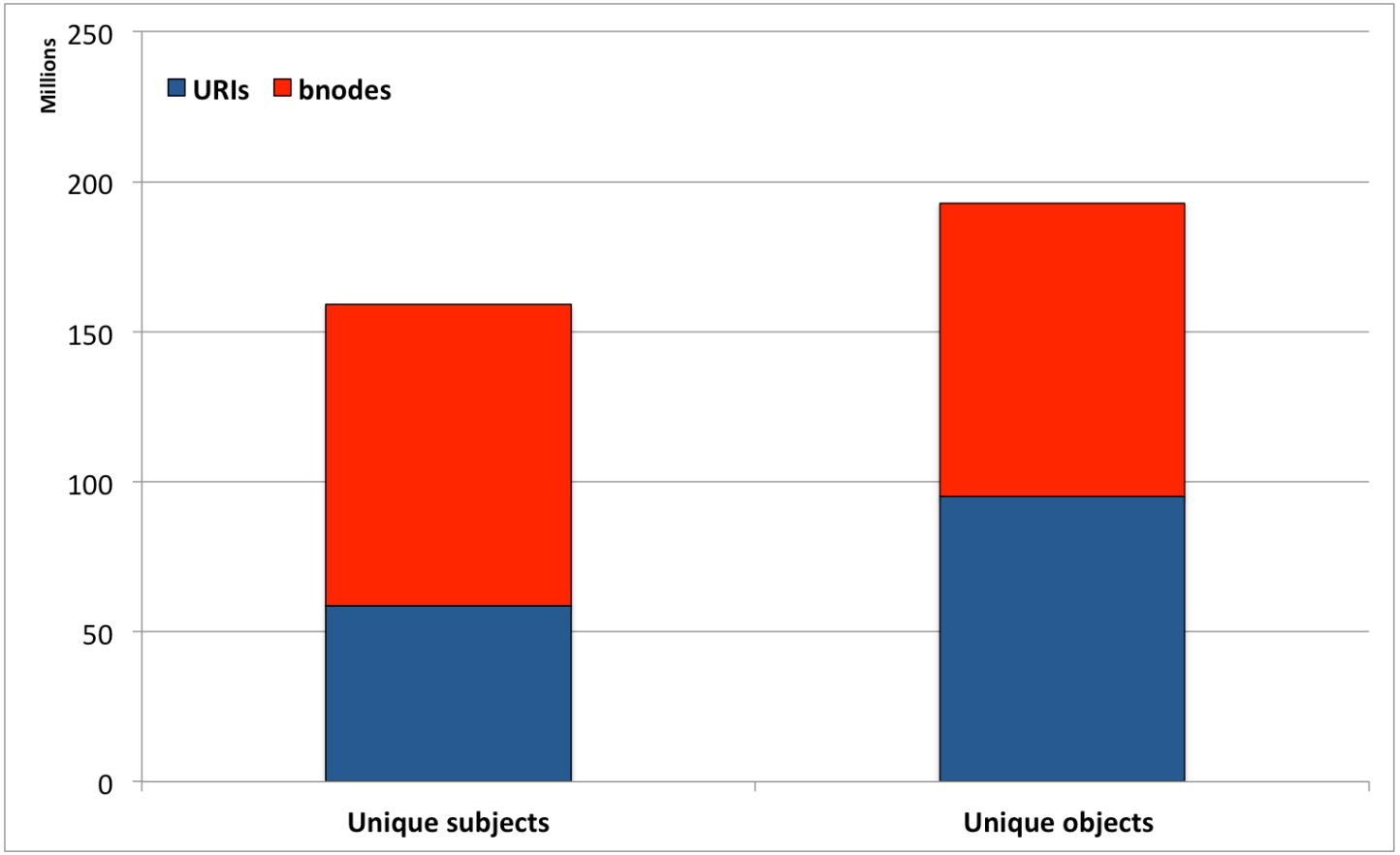
# How does real RDF look?




Data taken from http://gromgull.net/blog/2010/09/btc2010-basic-stats/

# How does real RDF look?

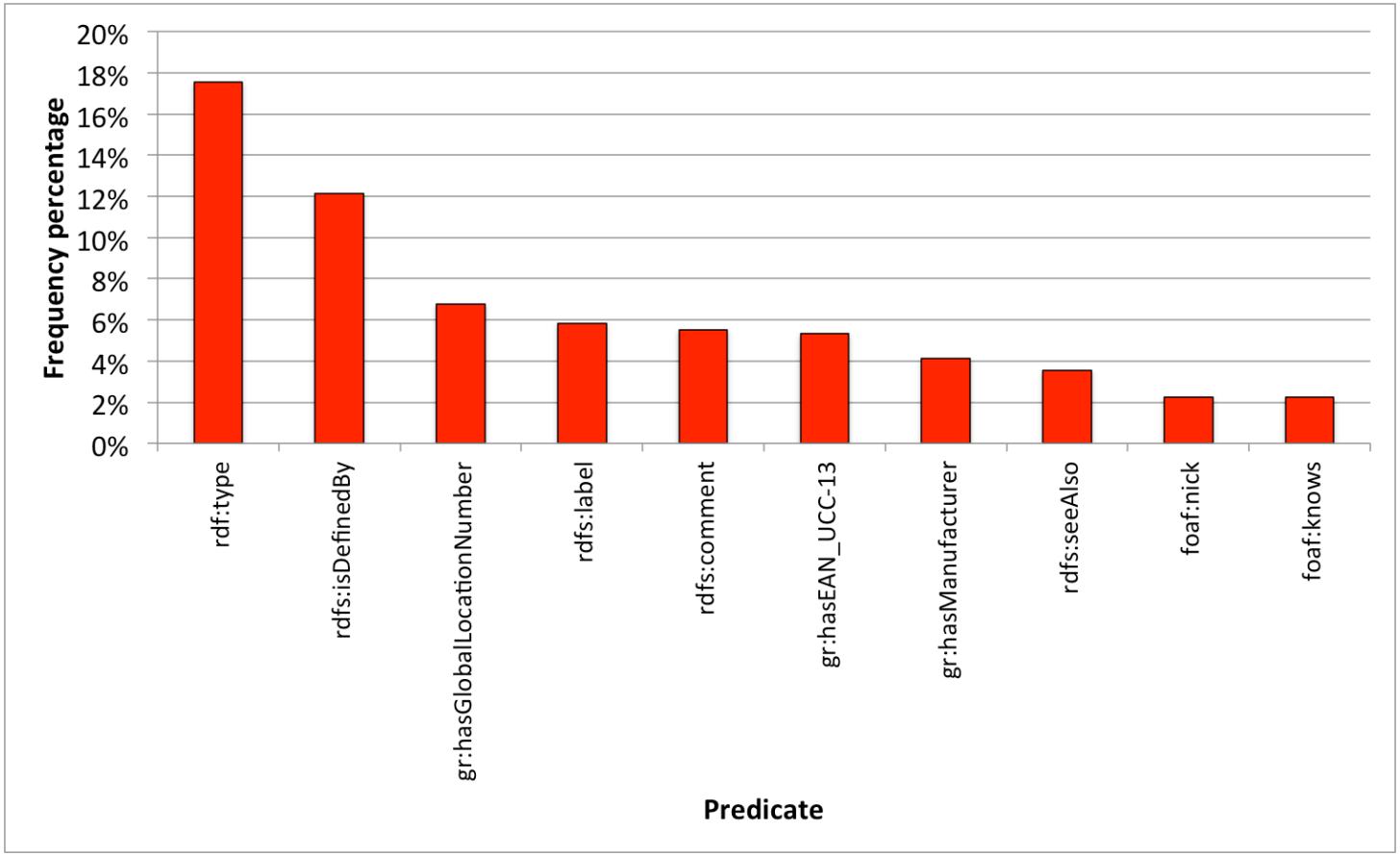
BTC2010 dataset




Data taken from http://gromgull.net/blog/2010/09/btc2010-basic-stats/

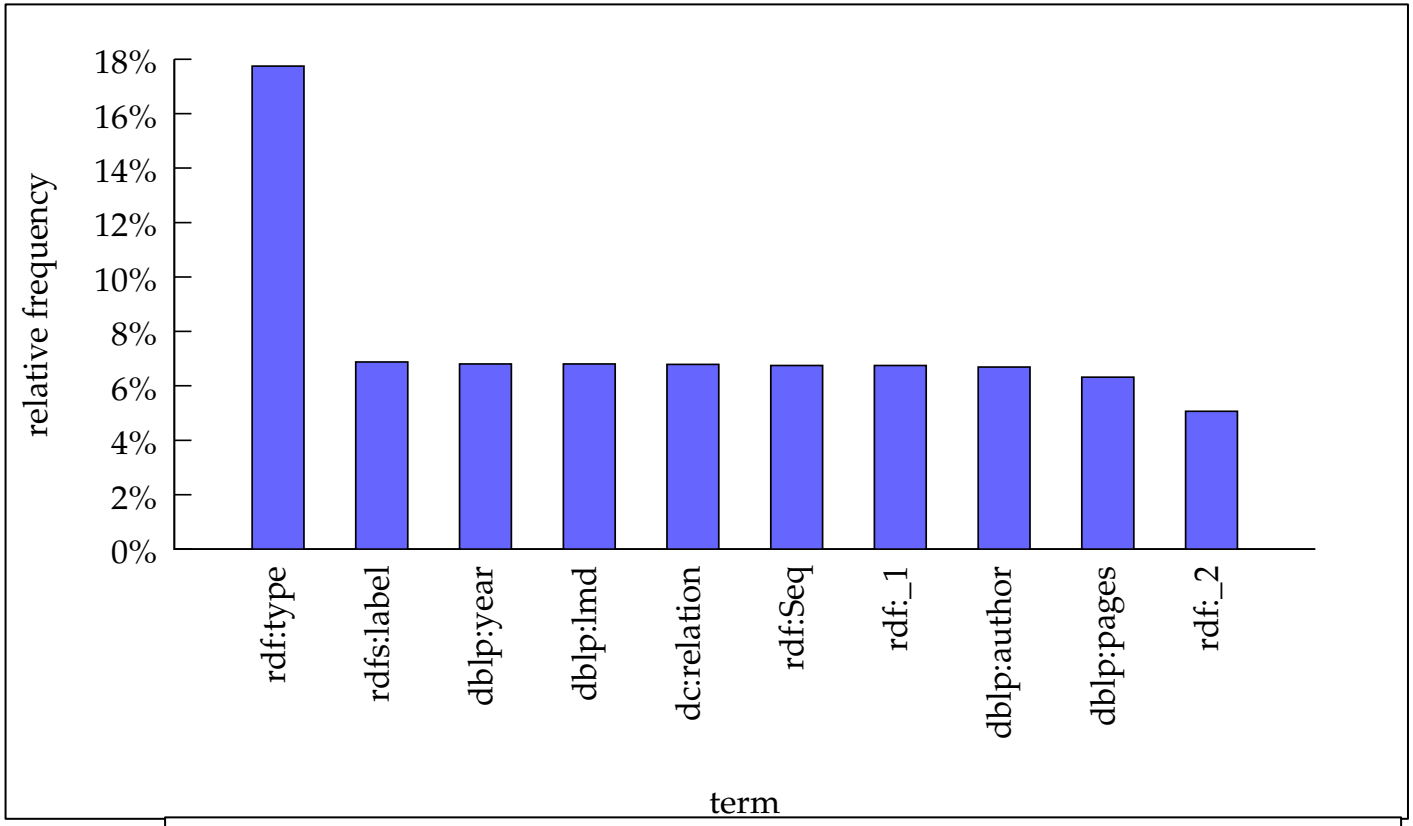
# How does real RDF look?

BTC2010 dataset




Data taken from http://gromgull.net/blog/2010/09/btc2010-basic-stats/

# How does real RDF look?




Top-10 terms of SwetoDBLP [Kotoulas10]

# 2

# CLOUD BASICS

# Cloud computing

*The origin of the term cloud computing was derived from […] drawings of stylized clouds to denote networks [...]*
*The cloud symbol was used to represent the Internet as early as 1994.*
[Wikipedia: http://en.wikipedia.org/wiki/Cloud_computing]

- Use of computing resources as a service over the network
- Scalability
- Elasticity
- Reliability

# Cloud computing service models

Google Apps

**Software-as-a-Service (SaaS)**

Applications

Windows Azure

web sites

**Platform-as-a-Service (PaaS)**

Platforms

amazon web services™

Elastic Compute Cloud (EC2)

**Infrastructure-as-a-Service (IaaS)**

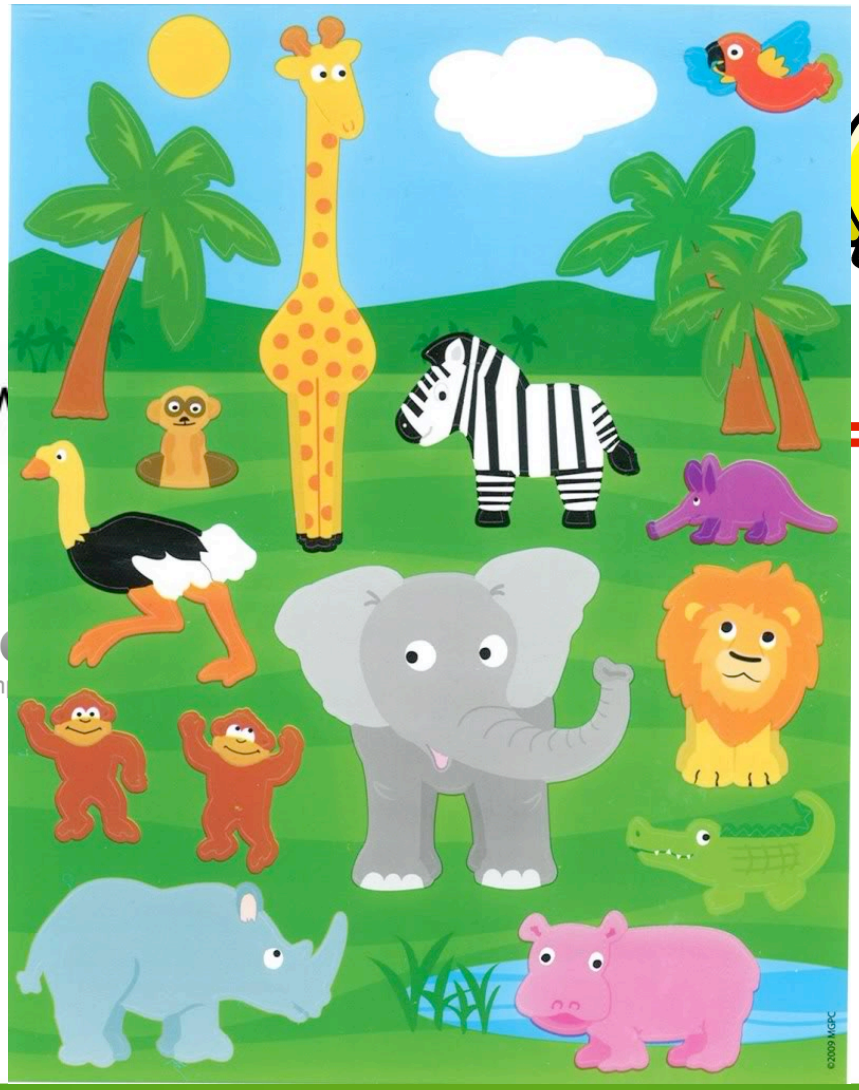
Hardware

**pay-per-use**

# Cloud computing and data management

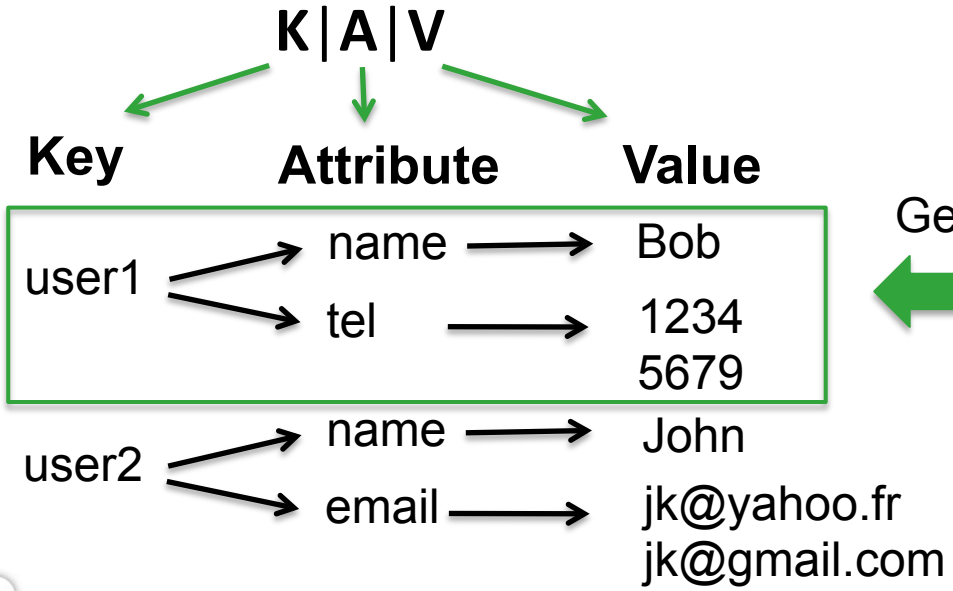
- Relational databases in the cloud (DBaaS)
  - IBM DB2 on SmartCloud
  - Amazon RDS (MySQL, Oracle or SQL Server)
- NoSQL systems [Catell11]:
  - Key-value stores (memcached, DynamoDB, SimpleDB)
  - Extensible record stores (BigTable, Cassandra, HBase, Accumulo)
  - Document stores (MongoDB, CouchDB)
  - Graph databases (Pregel, Apache Giraph, Neo4J)
- Parallel data processing paradigms:
  - MapReduce (Hadoop, Amazon EMR), PACTs/Stratosphere

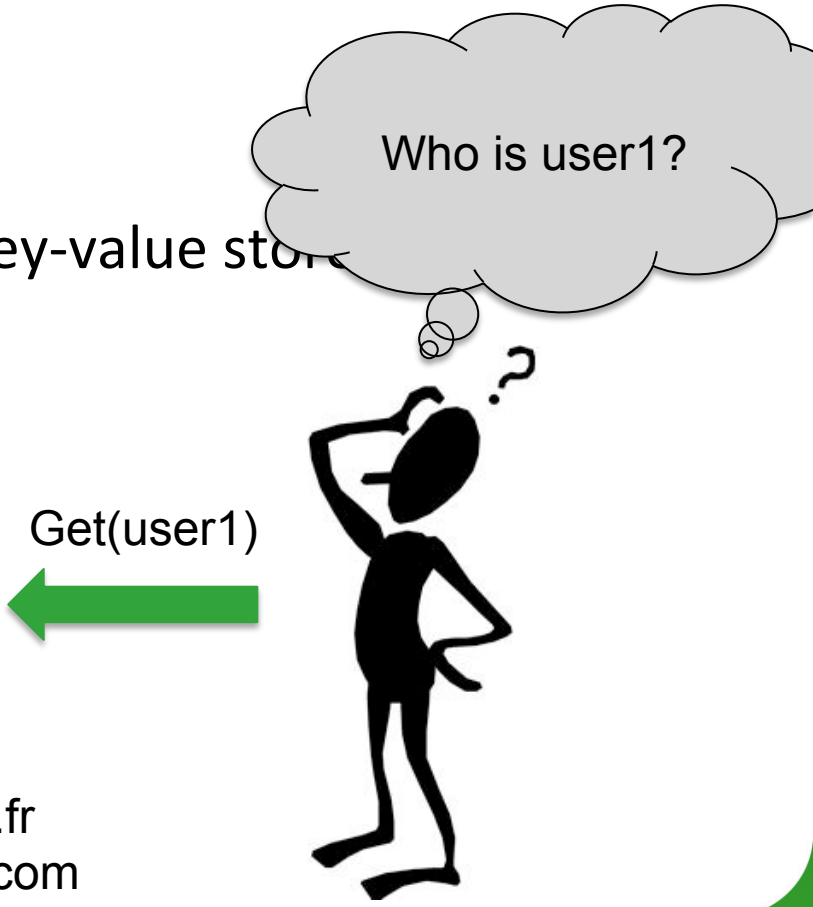
# Cloud ecosystem



# Distributed key-value stores

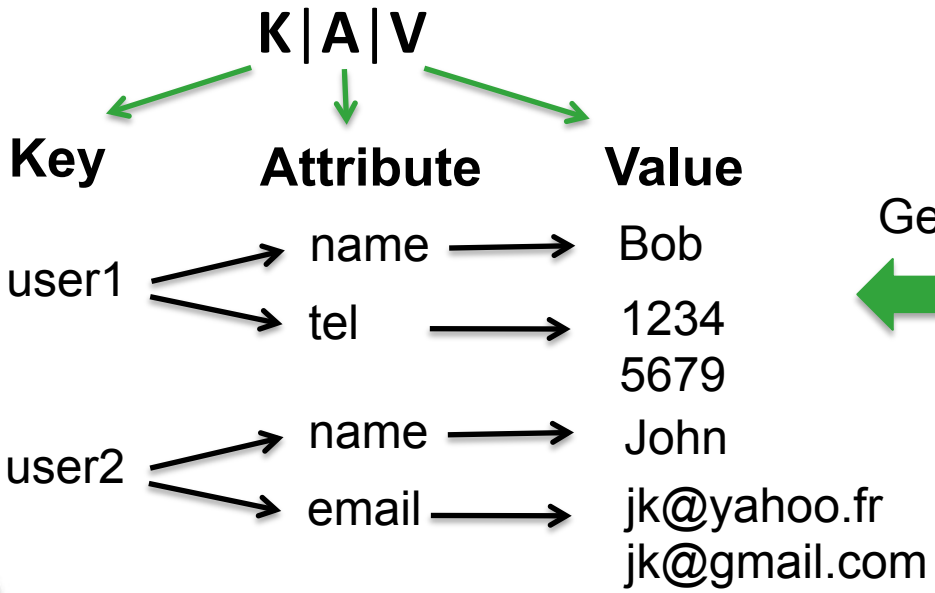
- Index: Hash-based or lexicographically sorted on the key
- Basic APIs: Put(k, v), Get(k), Delete(k)
- No schema, no joins
- Multi-value attributes
- Notation we will use for a table in a key-value sto

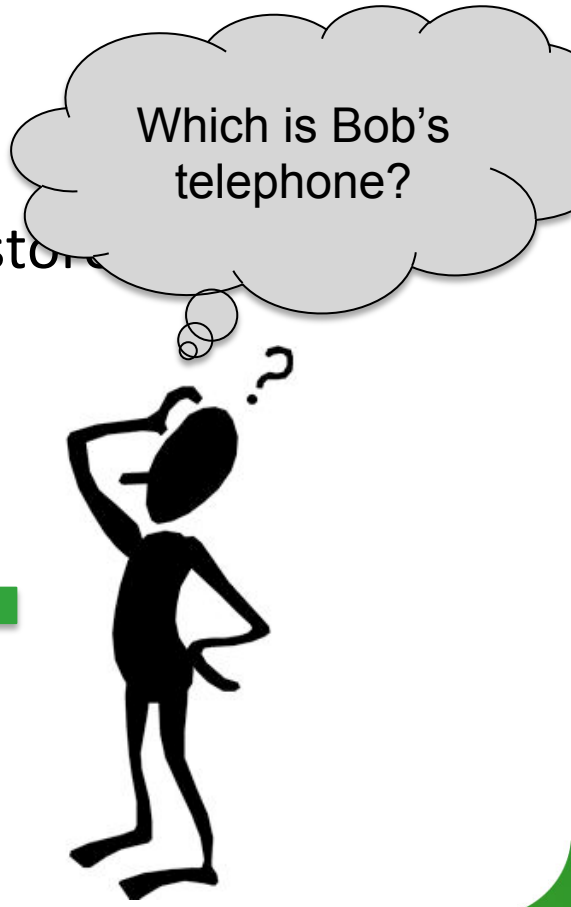


# Distributed key-value stores

- Index: Hash-based or lexicographically sorted on the key
- Basic APIs: Put(k, v), Get(k), Delete(k)
- No schema, no joins
- Multi-value attributes
- Notation we will use for a table in a key-value sto

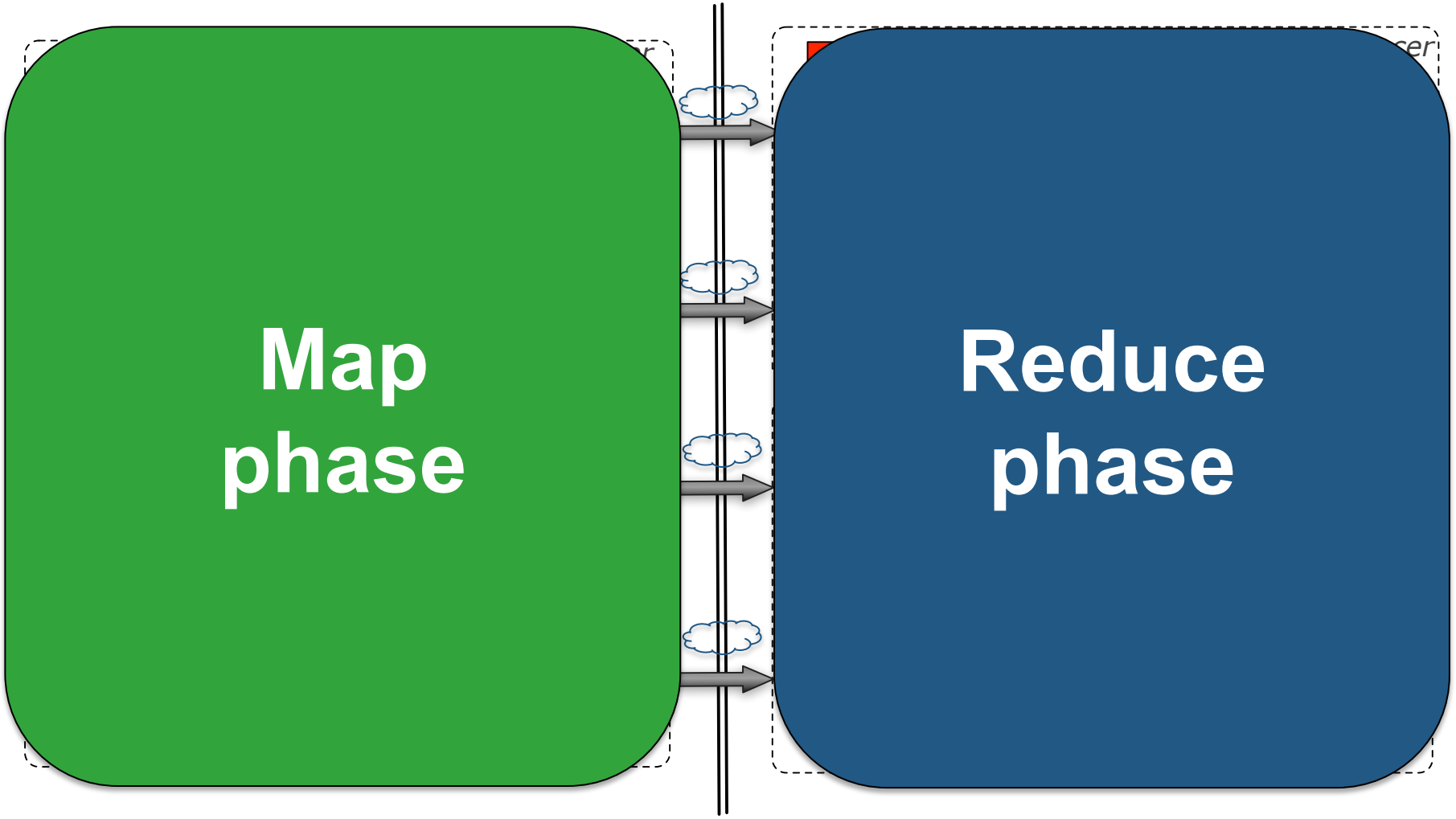


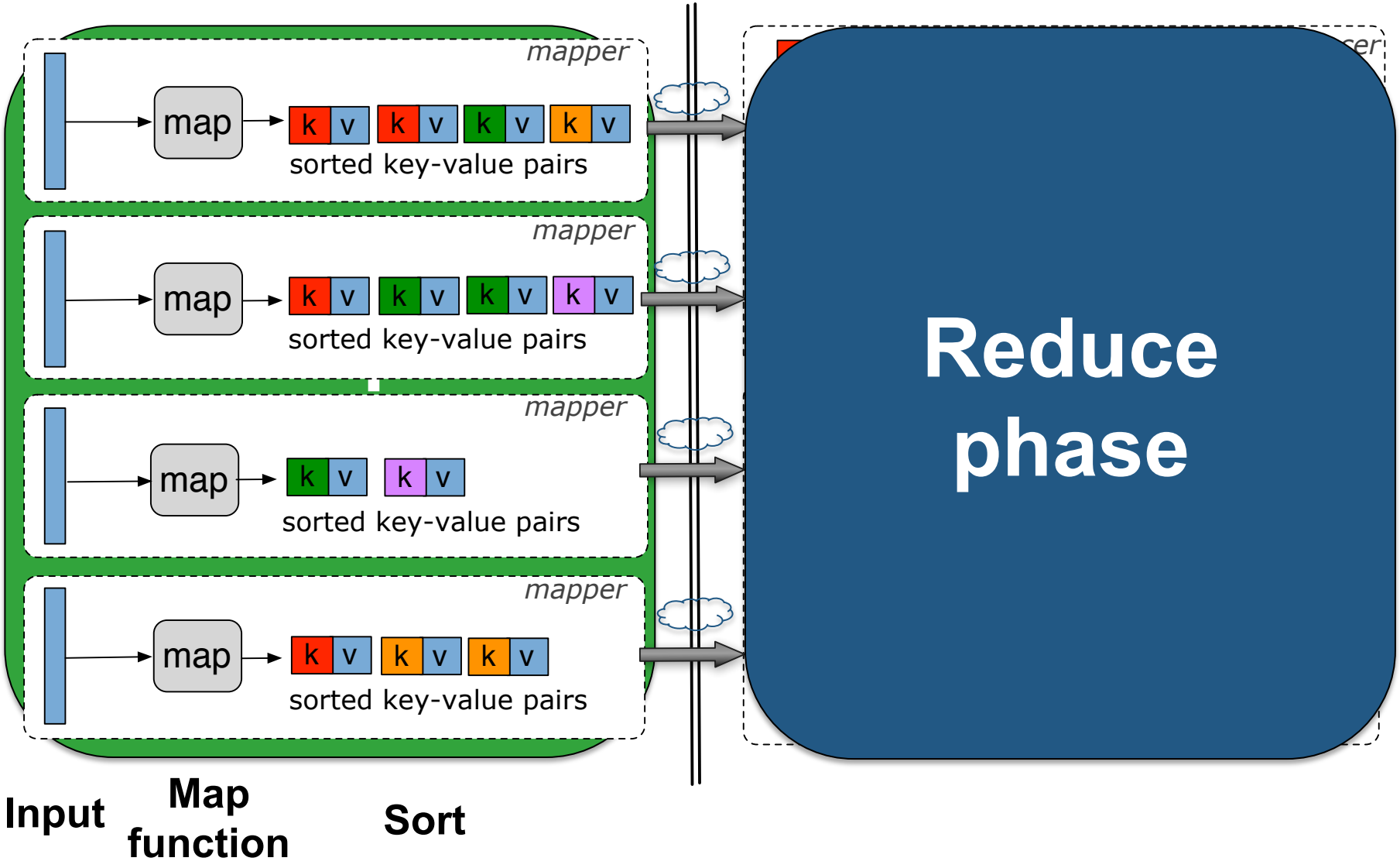
# More “advanced” distributed key-value stores

- Google BigTable, Apache Cassandra
- Hash or sorted index on the key
- Sorted index on the attribute name
- Supercolumns
  - K|{SC}A|V
  - Hash or sorted index
- Secondary index
  - maps values to keys

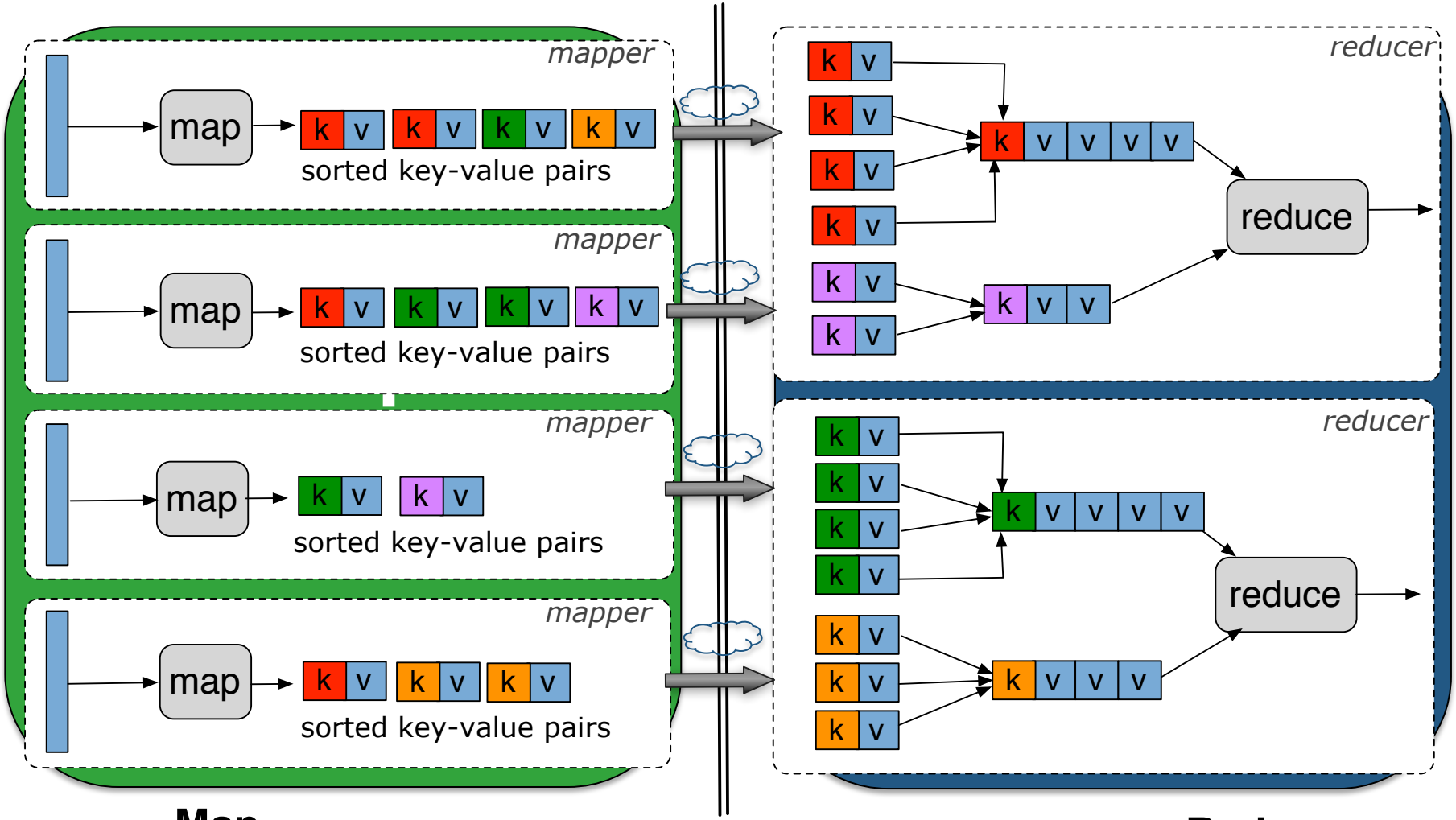
# MapReduce



# MapReduce


mapper
map
k v k v k v k v
sorted key-value pairs
mapper
map
k v k v k v k v
sorted key-value pairs
mapper
map
k v k v
sorted key-value pairs
mapper
map
k v k v k v
sorted key-value pairs
Reduce phase
Input
Map function
Sort

# MapReduce


mapper
map
k v k v k v k v
sorted key-value pairs
mapper
map
k v k v k v k v
sorted key-value pairs
mapper
map
k v k v
sorted key-value pairs
mapper
map
k v k v k v
sorted key-value pairs
reducer
k v
k v
k v
k v
k v
k v v v v
k v v
reduce
reducer
k v
k v
k v
k v
k v
k v
k v v v v
k v v v
reduce
Input
Map function
Sort
Shuffle
Merge
Reduce function
Output

# MapReduce

- High parallelization
- Scalable
- Fault-tolerant

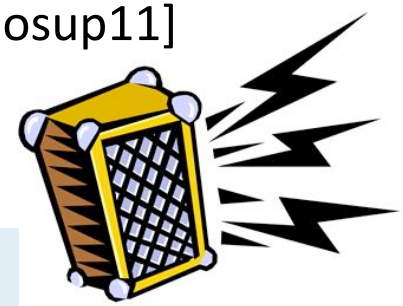
- Operates on 1 input
- No indexes
- Blocking operations

# Joins in MapReduce

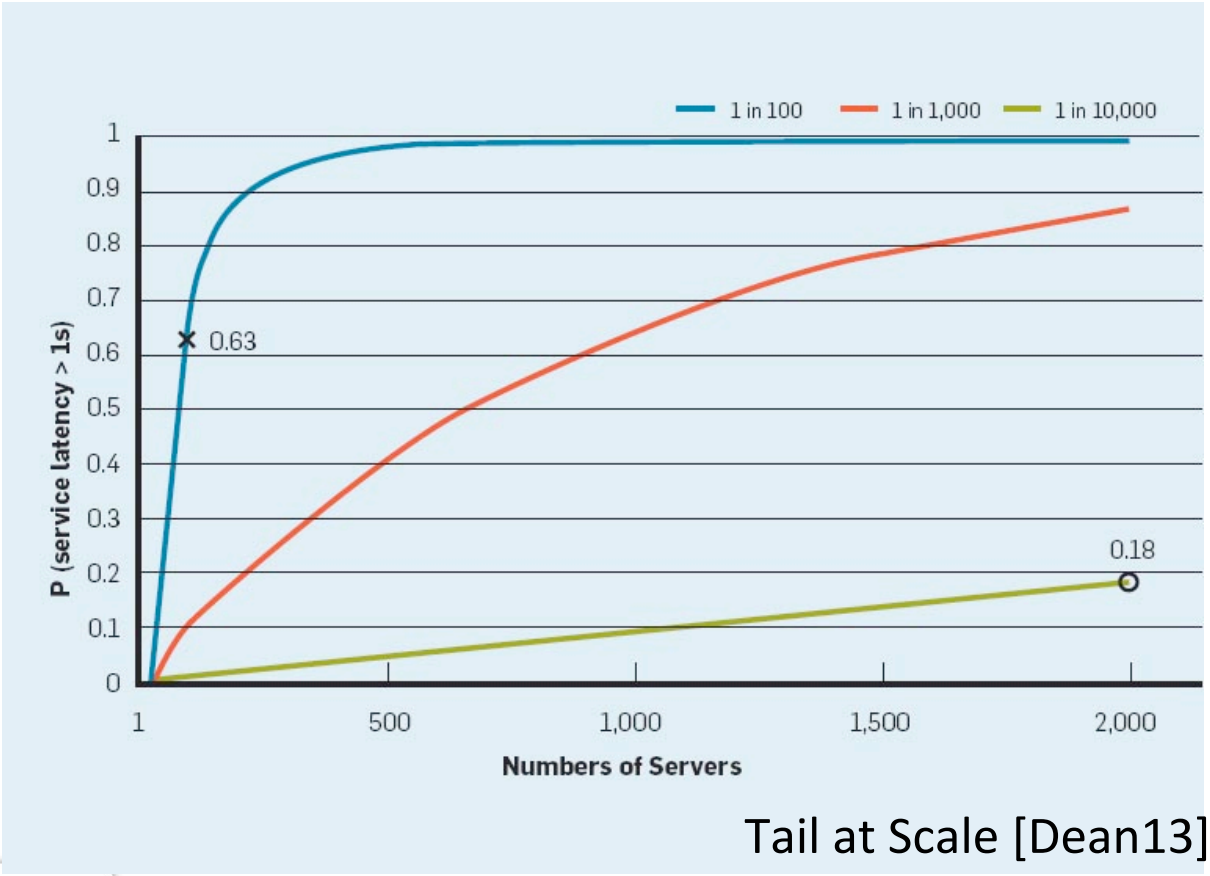
- Two main strategies:
  - symmetric hash join
  - replicated join
- Representative works:
  - [Blanas10] on different join methods in MapReduce
  - [Afrati11] on optimizing multiway joins in MapReduce
  - [Wu11] on query optimization in MapReduce

# Variability in the clouds

Cloud = performance variability [Schad10], [Iosup11]



**Variability is amplified by scale**




Tail at Scale [Dean13]

# 3

# ANALYSIS OF RDF SYSTEMS IN THE CLOUD

# RDF stores in the cloud

- **AMADA** [Bugiotti11, Arandar12]
- CumulusRDF [Ladwig11]
- **Graph partitioning** [Huang11]
- H2RDF [Papailiou12]
- **HadoopRDF** [Husain11]
- MAPSIN [Schätzle12]
- PigSPARQL [Schätzle11]
- RAPID+ [Ravindra11]
- **Rya** [Punnoose12]
- Stratustore [Ladwig11]
- **SHARD** [Rohloff10]
- Trinity.RDF [Zeng13]
- **WebPie** [Urbani09]
- QueryPie [Urbani11]
- …

# Analysis dimensions

**Data storage**

**Query processing**

**RDFS entailment**

# Analysis dimensions

**Data storage**

**Query processing**

**RDFS entailment**

# Categorization based on storage and query processing

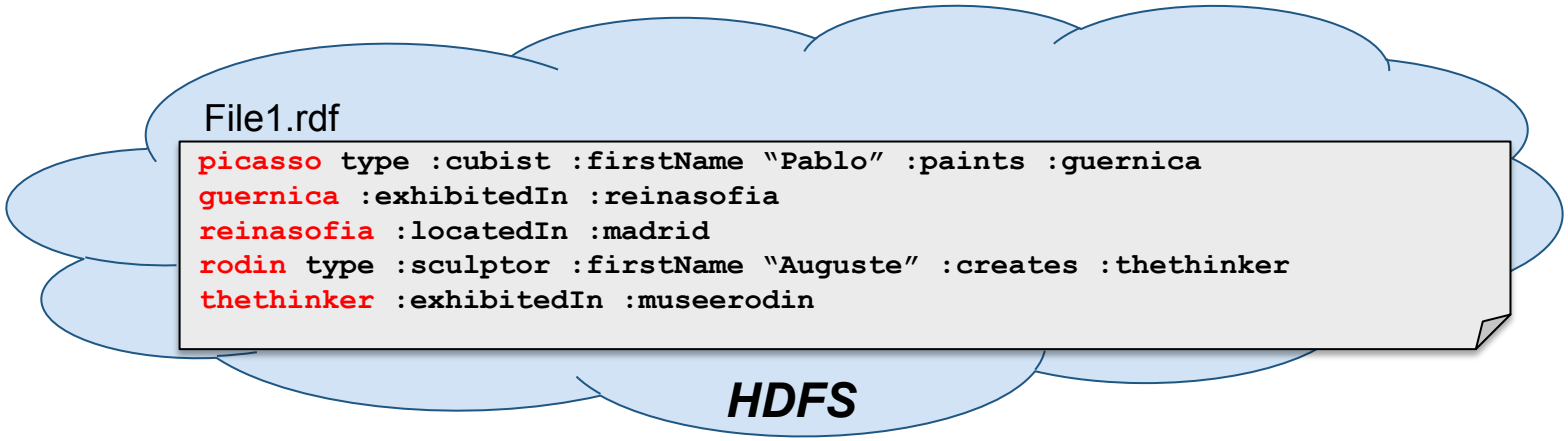
1. MapReduce-based approaches
2. Approaches based on key-value stores
3. Graph-based approaches
4. Systems based on commercial cloud services

# 1. MapReduce-based approaches

- Store RDF triples in DFS (usually HDFS)
- Use MapReduce to answer the queries (usually Hadoop)
- Representative works:
  - SHARD [Rohloff10]
  - HadoopRDF [Husain11]
  - RAPID+ [Ravindra11]
  - PigSPARQL [Schätzle11]

# SHARD triple-store [Rohloff10]

**Storing**: each line holds all triples about a given subject

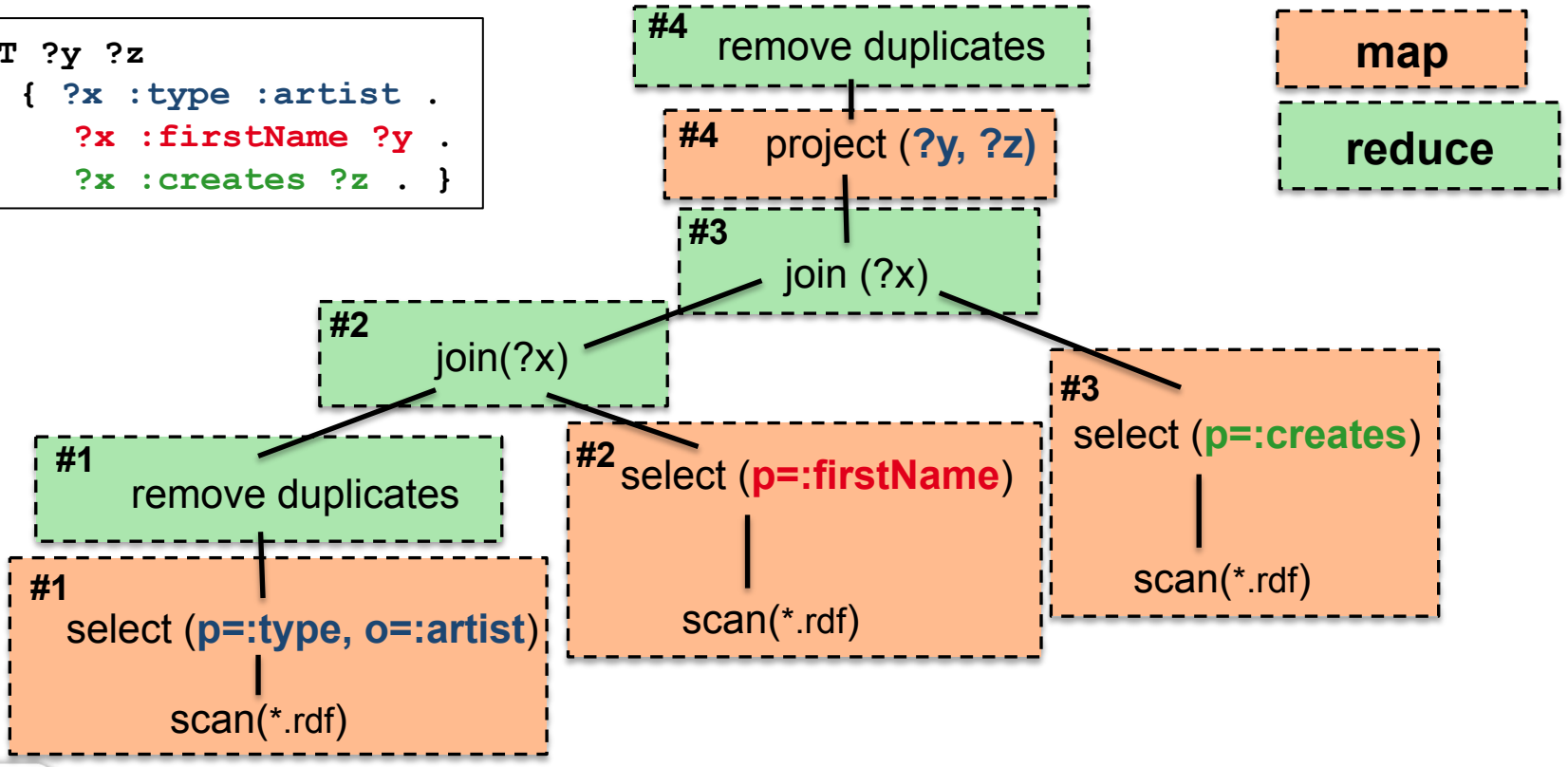


# SHARD triple-store [Rohloff10]

**Query processing**: Iterative MapReduce jobs

- 1 MapReduce job for each triple pattern + 1 for the projection at the end
- Map phase: match triple pattern by scanning all triples
- Reduce phase: join triple pattern with intermediate results from previous ones

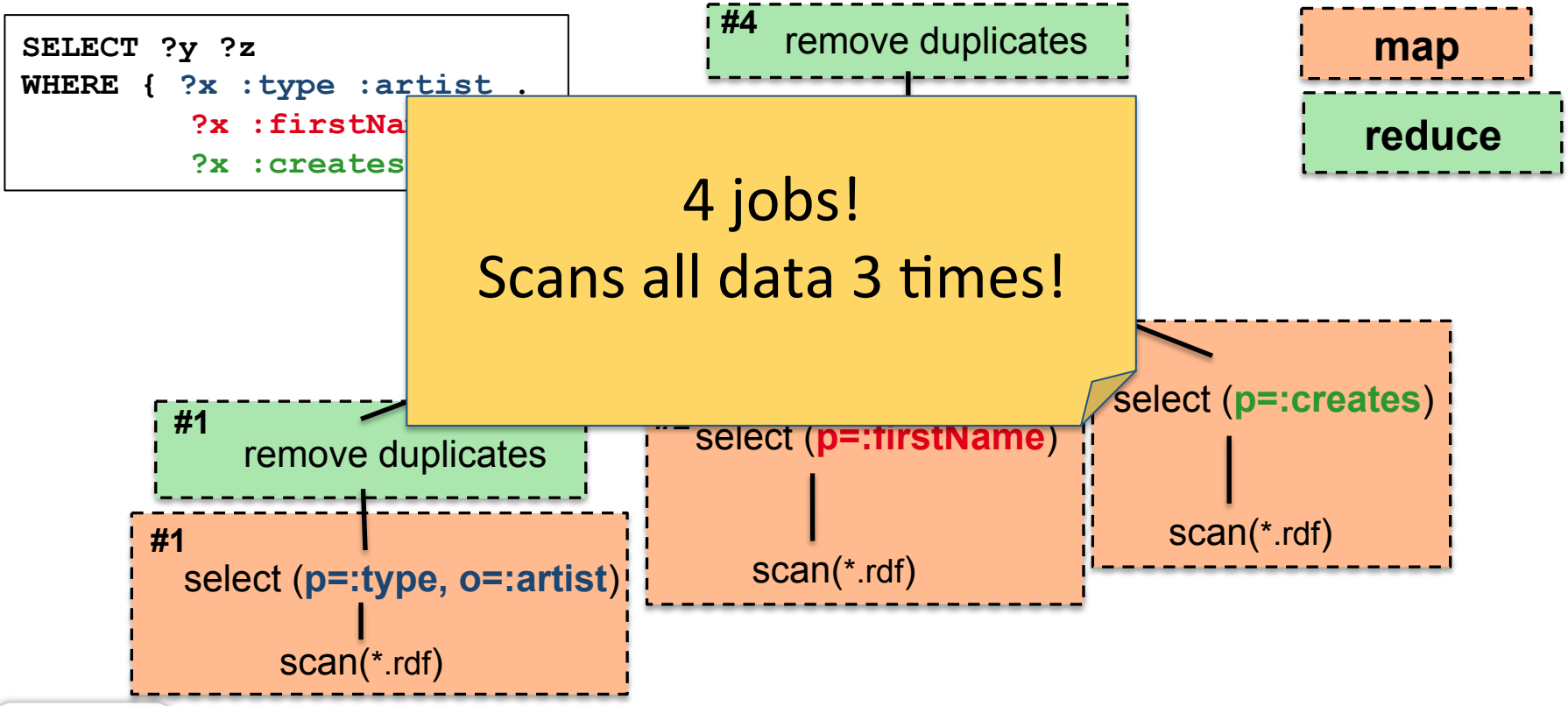
```
SELECT ?y ?z
WHERE { ?x :type :artist .
        ?x :firstName ?y .
        ?x :creates ?z . }
```



# SHARD triple-store [Rohloff10]

**Query processing**: Iterative MapReduce jobs

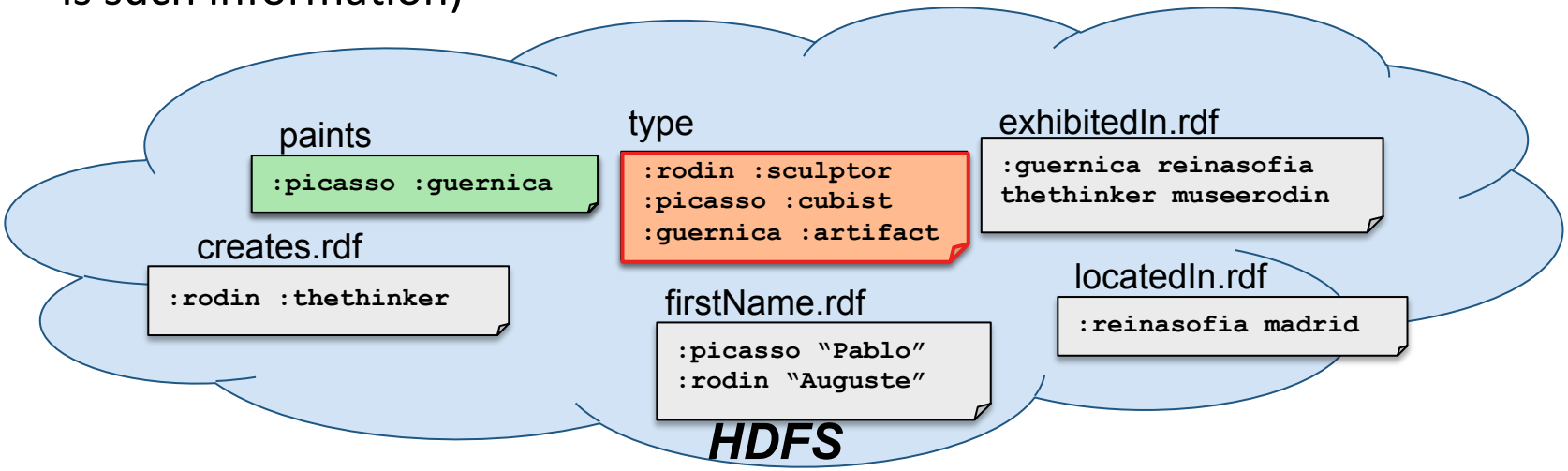
- 1 MapReduce job for each triple pattern + 1 for the projection at the end
- Map phase: match triple pattern by scanning all triples
- Reduce phase: join triple pattern with intermediate results from previous ones



# HadoopRDF [Husain11]

**Storing:**

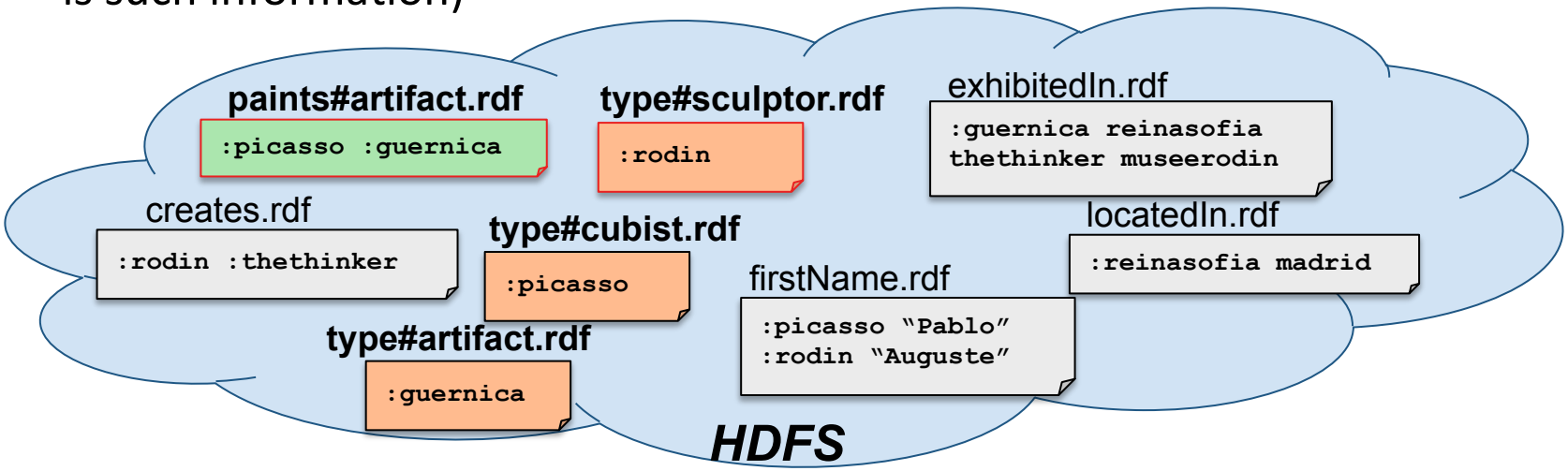
- RDF triples are grouped by the property names in files (like in vertical partitioning of centralized RDF stores)
- The file for the property type is split based on the object
- Triples with the same property are then grouped by their object type (if there is such information)



# HadoopRDF [Husain11]

**Storing:**

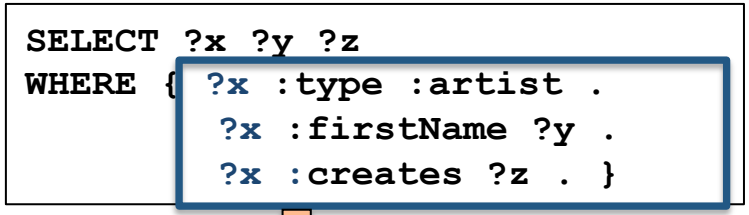
- RDF triples are grouped by the property names in files (like in vertical partitioning of centralized RDF stores)
- The file for the property type is split based on the object
- Triples with the same property are then grouped by their object type (if there is such information)

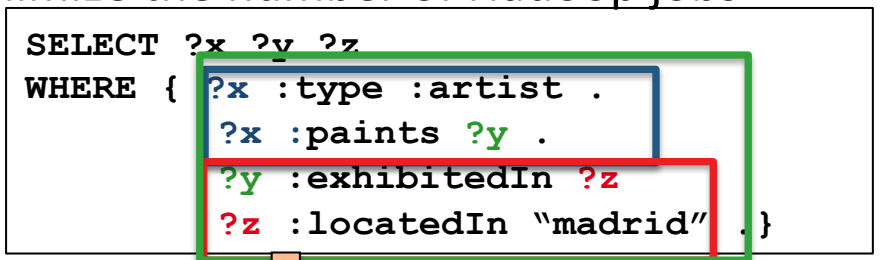


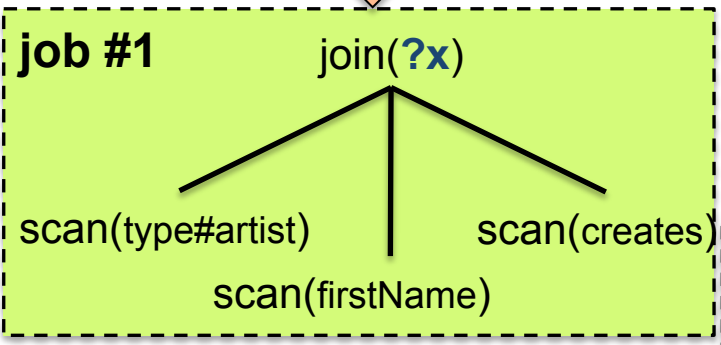
# HadoopRDF [Husain11]

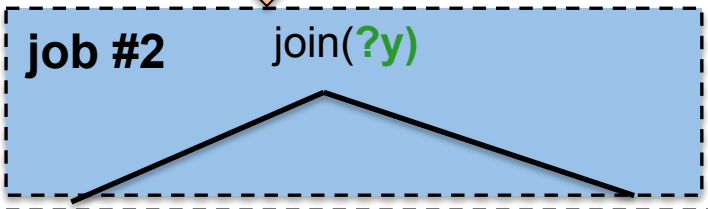
**Query processing**:

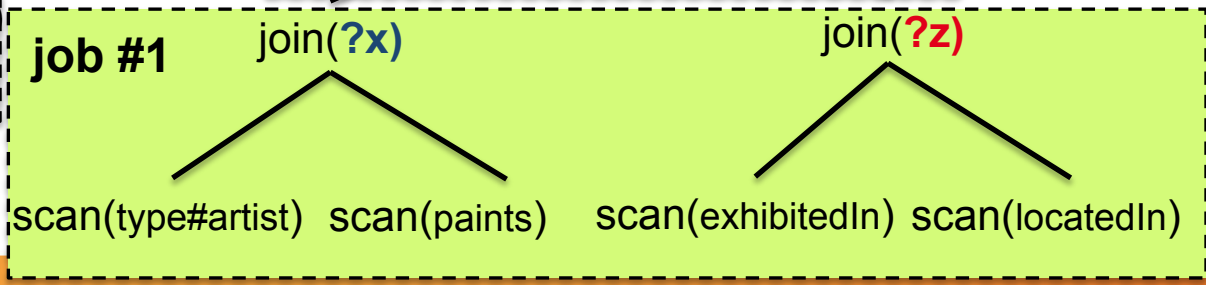
- File selection based on the property (and object) of the triple
- Joining two or more triple patterns on the same single variable
- Several joins may be executed in parallel if they are on different variables
- Heuristic: Query decomposition to minimize the number of Hadoop jobs



# 1. MapReduce-based approaches

| Pros | Cons |
| --- | --- |
| Scalability | No indexes – scan all data |
| Fault-tolerance | Shuffling of data |
| Fast data uploading | Semi-support for joins |
| Fast and easy to implement | Big initialization overhead |
| | High query response times |

# 2. Approaches based on key-value stores

- Store RDF triples in a key-value store
  - Indexes on RDF triples
- Lookups to the key-value store for matching each triple pattern
- How do I perform the joins?
  - Collect the results somewhere locally
  - Use MapReduce
- Representative works
  - CumulusRDF [Ladwig11]
  - Stratustore [Stein10]
  - Rya [Punnoose12]
  - AMADA [Bugiotti12]
  - H2RDF [Papailiou12]
  - MAPSIN [Schätzle12]

Local query processing

MapReduce-based query processing

# Storing RDF in key-value stores

- Rya uses Apache Accumulo — **Sorted Index**
  - SPO|-|- POS|-|- OSP|-|-
- H2RDF uses Apache HBase — **Sorted Index**
  - SP|O|- PO|S|- OS|P|-
- AMADA uses Amazon DynamoDB — **Hash Index**
  - S|P|O P|O|S O|S|P
- MAPSIN uses Apache HBase — **Sorted Index**
  - S|P|O|- O|P|S|-
- Stratustore uses Amazon SimpleDB — **Hash Index**
  - S|P|O
- CumulusRDF uses Apache Cassandra — **Hash Index** **Sorted Index**
  - hierarchical layout: S|{P}O|- P|{O}S|- O|{S}P|-
  - flat layout: S|PO|- PO|S|- PO|'P'| P - O|SP|-

# Storing RDF in key-value stores

- Rya uses Apache Accumulo — **Sorted Index**
  - SPO|-|- POS|-|- OSP|-|-
- H2RDF uses Apache HBase — **Sorted Index**
  - SP|O|- PO|S|- OS|P|-
- AMADA uses Am… — …dex
  - S|P|O P|…
- MAPSIN uses Apa… — …ndex
  - S|P|O|- O|…
- Stratustore uses Amazon SimpleDB — **Hash Index**
  - S|P|O
- CumulusRDF uses Apache Cassandra — **Hash Index** **Sorted Index**
  - hierarchical layout: S|{P}O|- P|{O}S|- O|{S}P|-
  - flat layout: S|PO|- PO|S|- PO|'P'| P - O|SP|-

3 indexes:
SPO, POS, OSP

# Rya storage - example

SPO

| Key | (attribute, value) |
|---|---|
| :picasso,:firstName,"Pablo" | - |
| :picasso,:p | |
| :picasso,ty | |
| :guernica,: | |
| … | |

POS

| Key | (attribute, value) |
|---|---|
| :exhibitedIn,:reinasofia,:guernica | - |
| :firstName,"Pablo",:picasso, | - |
| :paints,:gu | |
| type,:cubis | |
| … | |

OSP

| Key | (attribute, value) |
|---|---|
| :cubist,:picasso,type, | - |
| :guernica,:picasso,:paints | - |
| "Pablo",:picasso,:firstName | - |
| :reinasofia,:guernica,:exhibitedIn | - |
| … | - |

# Strong point of key-value stores: RDF lookups

## Different access patterns for triple pattern matching

| Triple pattern | Rya (Accumulo) | H2RDF (HBase) | AMADA (DynamoDB) | MAPSIN (HBase) |
|---|---|---|---|---|
| (s, p, o) | any lookup | any lookup + select | any lookup + select | any lookup + sselect |
| (s, p, ?o) | SPO range scan | SP\|O lookup | S\|P\|O lookup + select | S\|P\|O lookup + sselect |
| (s, ?p, o) | OSP range scan | OS\|P lookup | O\|S\|P lookup + select | O\|S\|P lookup + sselect |
| (s, ?p, ?o) | SPO range scan | SP\|O range scan | S\|P\|O lookup | S\|P\|O lookup |
| (?s, p, o) | POS range scan | PO\|S lookup | P\|O\|S lookup + select | O\|S\|P lookup + sselect |
| (?s, p, ?o) | POS range scan | PO\|S range scan | P\|O\|S lookup | any scan + sselect |
| (?s, ?p, o) | OSP range scan | OS\|P range scan | O\|S\|P lookup | O\|S\|P lookup |
| (?s, ?p, ?o) | any scan | any scan | any scan | any scan |

select: filter on client side
sselect: filter on server side

# Weak point of key-value stores: joins

Key-value stores do not support joins

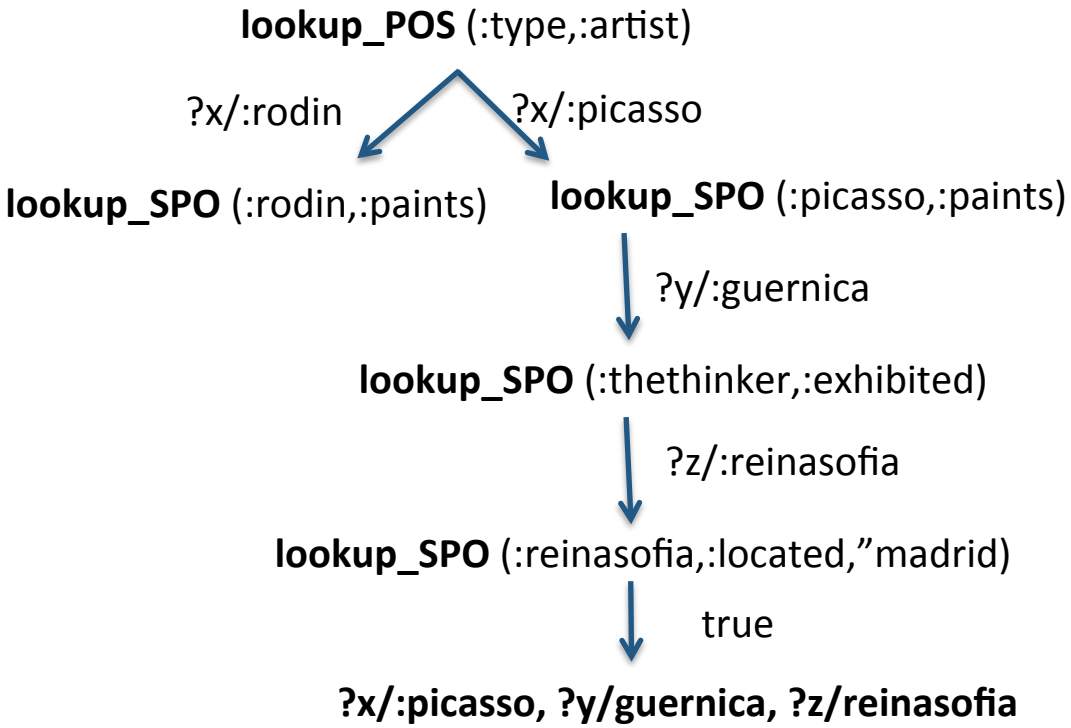
Make all lookups and perform joins out of the store

- **Rya**: index nested loops by query rewriting and lookups
- **H2RDF**: 2 strategies
  - selective joins → centralized – index nested loops
  - non-selective → MapReduce
- **AMADA**: memory hash join
- **MAPSIN**: MapReduce (map phase only)
- **Stratustore**: SimpleDB SELECT query for each star-join and rest of the joins locally
- **CumulusRDF**: Only single triple pattern queries are supported

# Rya joining strategy - example

- Index nested loops
- A lookup for the 1st triple pattern
- For each triple pattern $t_i$, use the $n$ results from triple pattern $t_{i-1}$ to rewrite $t_i$ and then perform $n$ lookups for the rewritten $t_i$

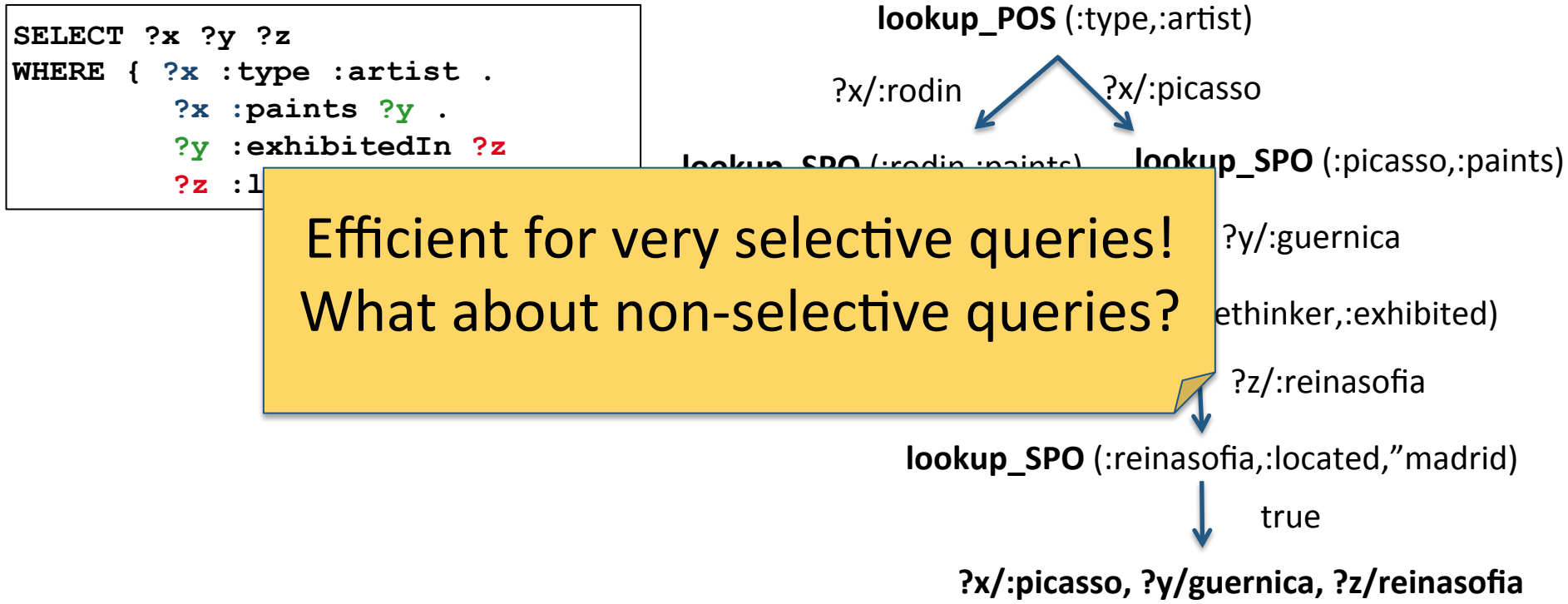
```
SELECT ?x ?y ?z
WHERE { ?x :type :artist .
        ?x :paints ?y .
        ?y :exhibitedIn ?z
        ?z :locatedIn "madrid".}
```



# Rya joining strategy - example

- Index nested loops
- A lookup for the 1st triple pattern
- For each triple pattern $t_i$, use the $n$ results from triple pattern $t_{i-1}$ to rewrite $t_i$ and then perform $n$ lookups for the rewritten $t_i$

```
SELECT ?x ?y ?z
WHERE { ?x :type :artist .
        ?x :paints ?y .
        ?y :exhibitedIn ?z
        ?z :l
```




Efficient for very selective queries!
What about non-selective queries?

# 2. Approaches based on key-value stores

| Pros | Cons |
| --- | --- |
| Scalability | Low schema expressivity |
| Fault-tolerance | Joins should be implemented by the user |
| Very fast lookups | |
| Efficient for selective queries | Joins are performed on the client-side |

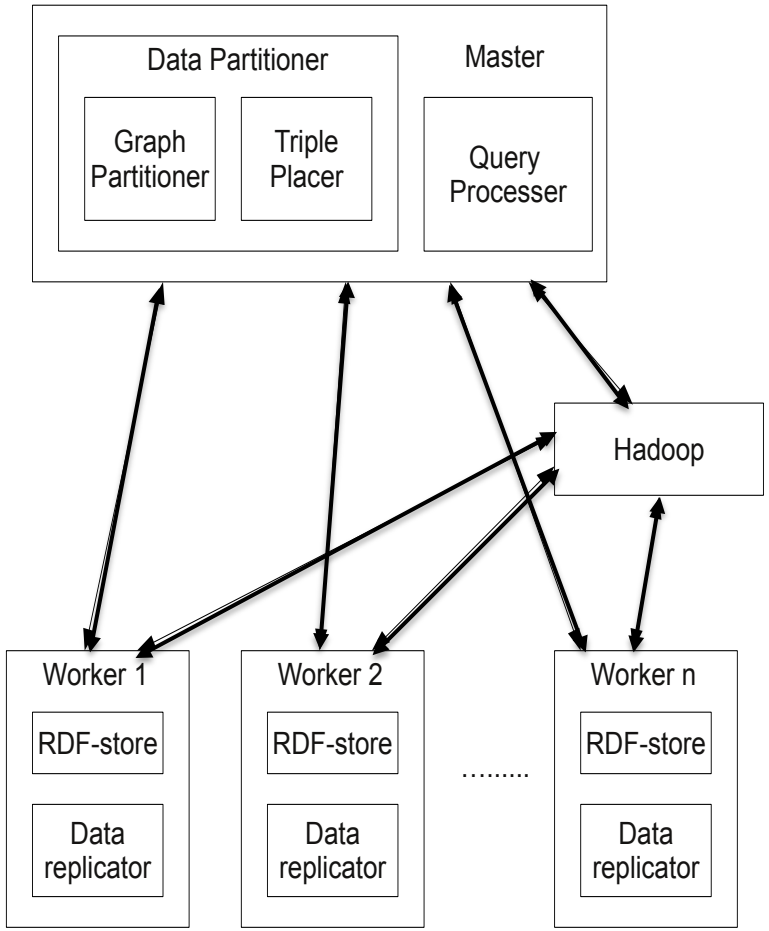
# 3. Graph-based approaches

- RDF data partitioning using known graph tools
  - [Huang11] uses METIS: a graph partitioning tool

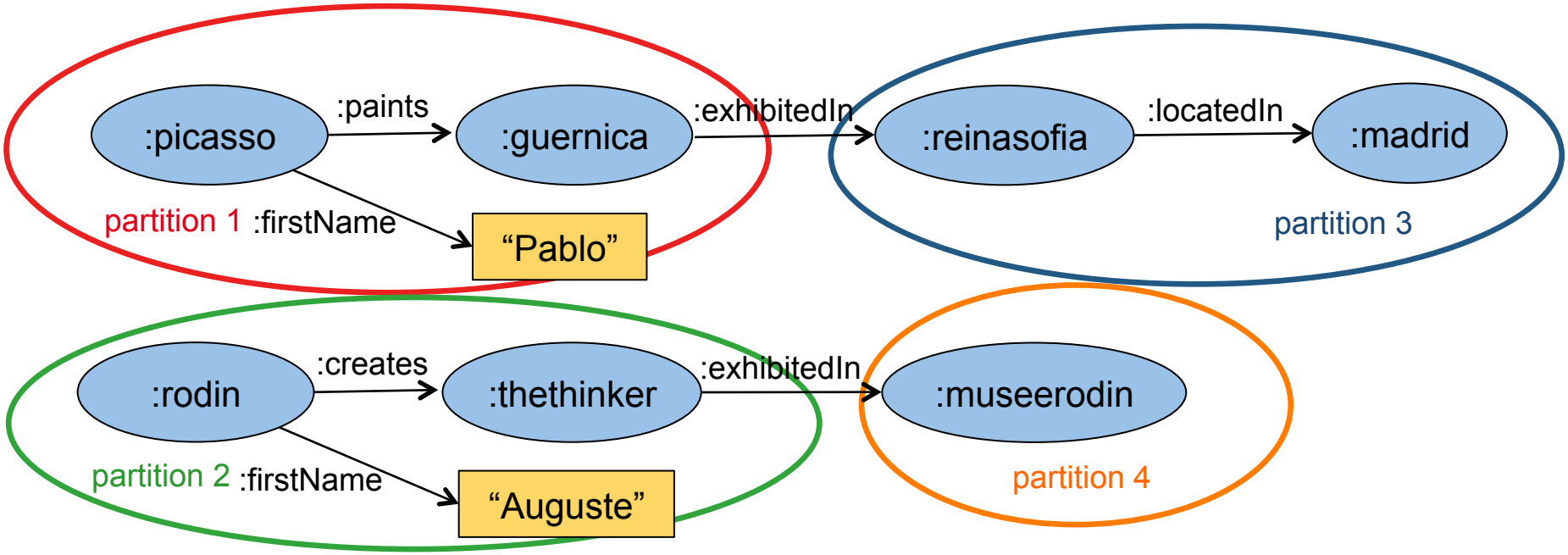
- Trinity.RDF [Zeng13] by Microsoft
  - Uses Trinity: a distributed graph system over a memory cloud

# Graph partitioning [Huang11]

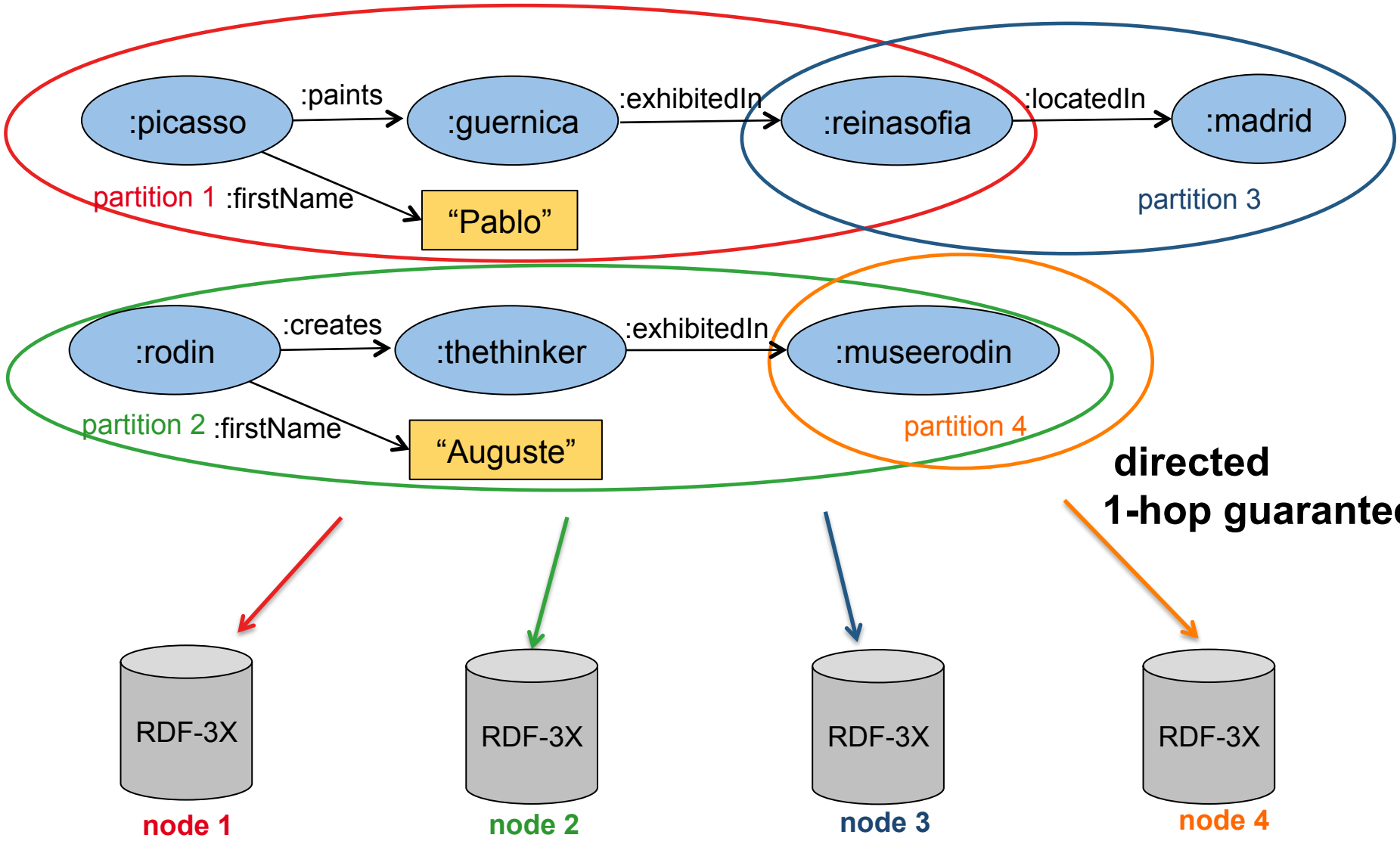
- Data partitioning
  - METIS graph partitioner [METIS]
  - vertex partitioning
  - #partitions = #machines
  - minimum edge cut
  - **`type`** triples are excluded from the partitioning process
- Data placement
  - place the triple in the partition in which the subject belongs to
  - replication across the partitions
  - directed/undirected n-hop guarantee
- Data storage
  - each machine stores triples into RDF-3X



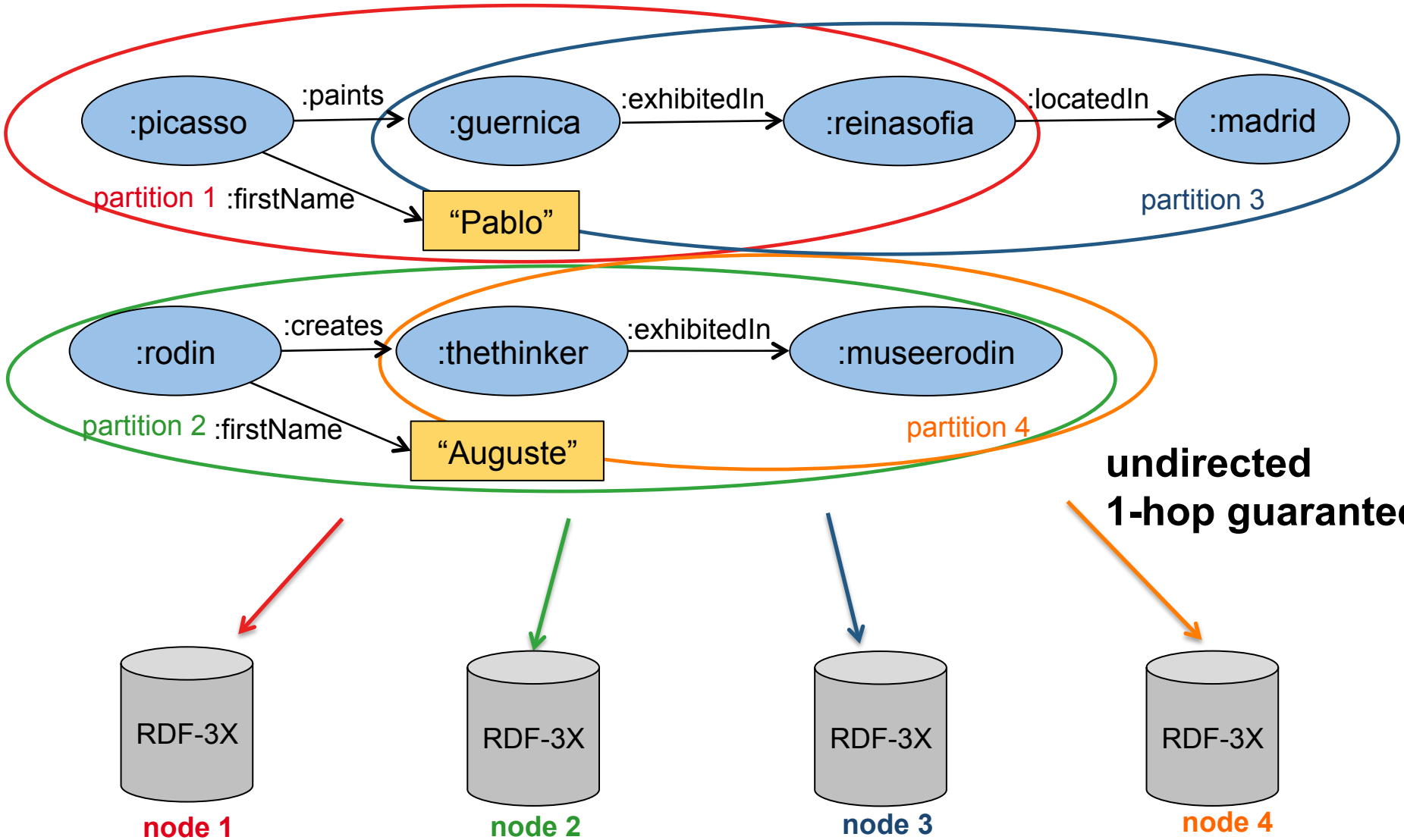
# Graph partitioning [Huang11] - storage



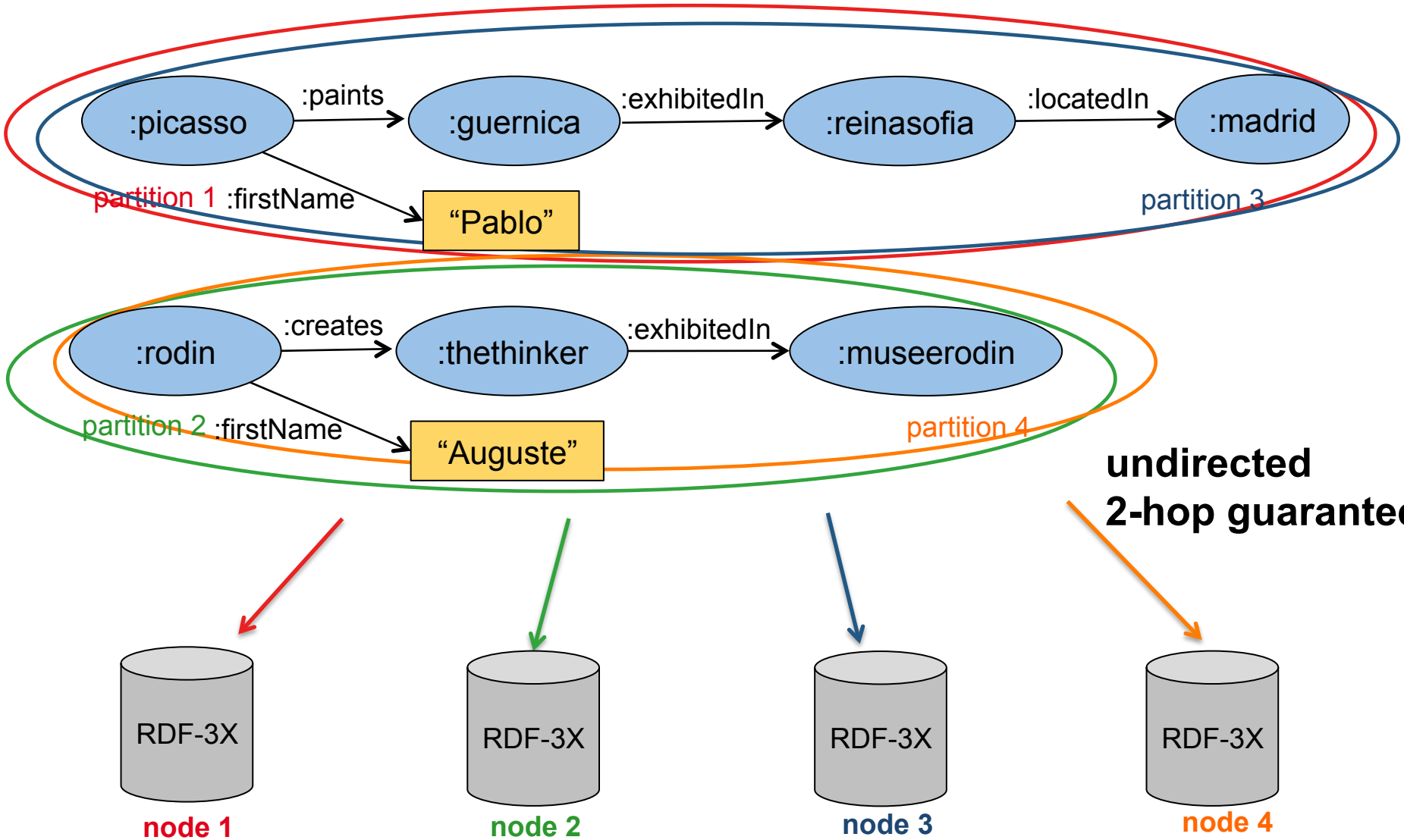

# Graph partitioning [Huang11] - storage



# Graph partitioning [Huang11] - storage

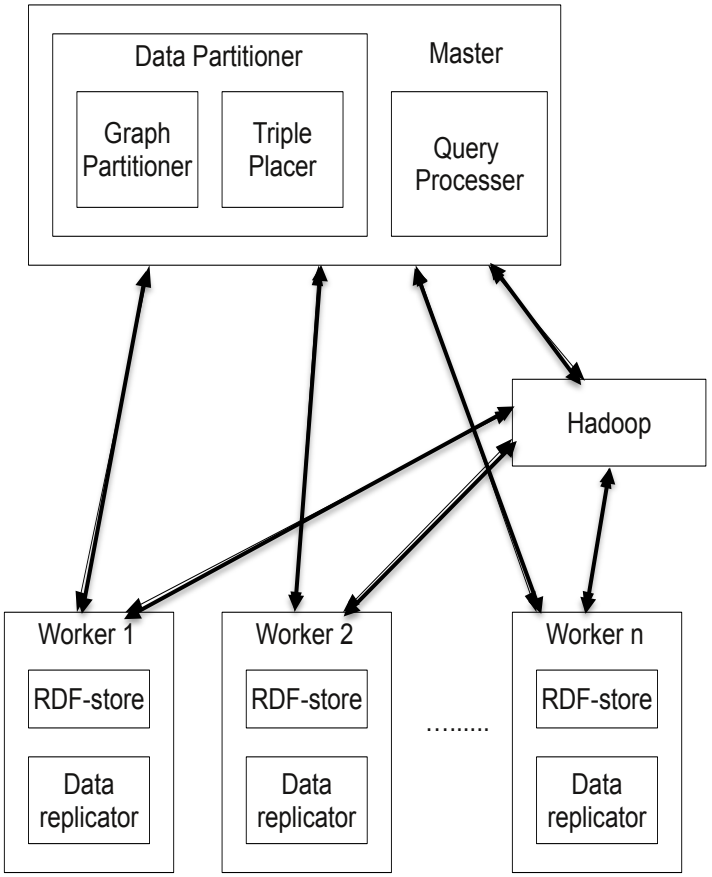


# Graph partitioning [Huang11] - storage



# Graph partitioning [Huang11] - querying

- Query decomposition for parallelization
- Check if q is PWOC (parallelizable without communication)
  - Yes: answer only from RDF-3X and union results
  - No: answer subqueries from RDF-3X and join them with Hadoop
- # jobs = # subqueries
- Heuristic optimization
  - decomposition so as to have minimum number of subqueries → minimal edge partitioning of a graph into subgraphs of bounded diameter

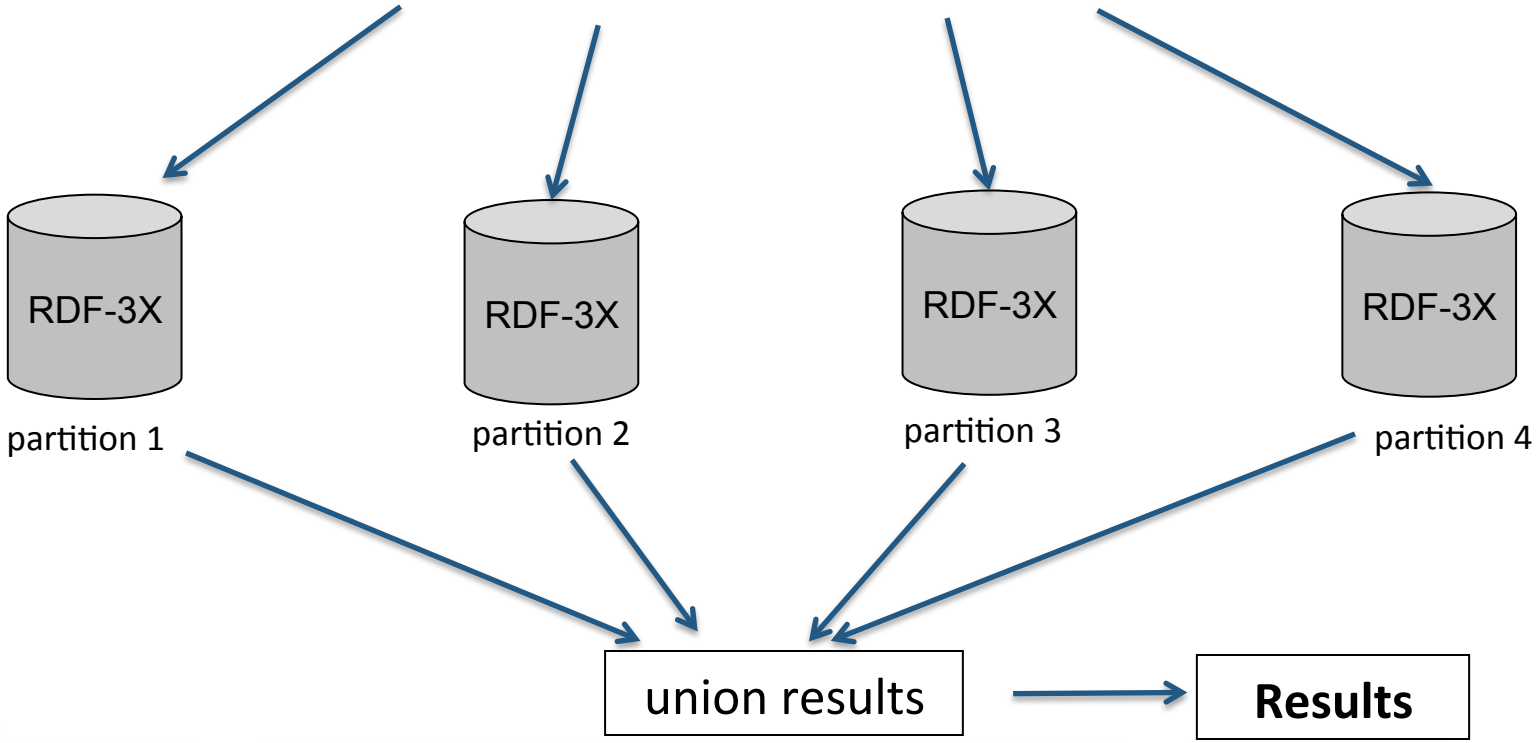


# Graph partitioning [Huang11] - querying

replication with
1-hop guarantee

**PWOC query**

```
SELECT ?x ?y ?z
WHERE {
        ?x type :artist .
        ?x :firstName ?y .
        ?x :creates ?z .}
```



# Graph partitioning [Huang11] - querying

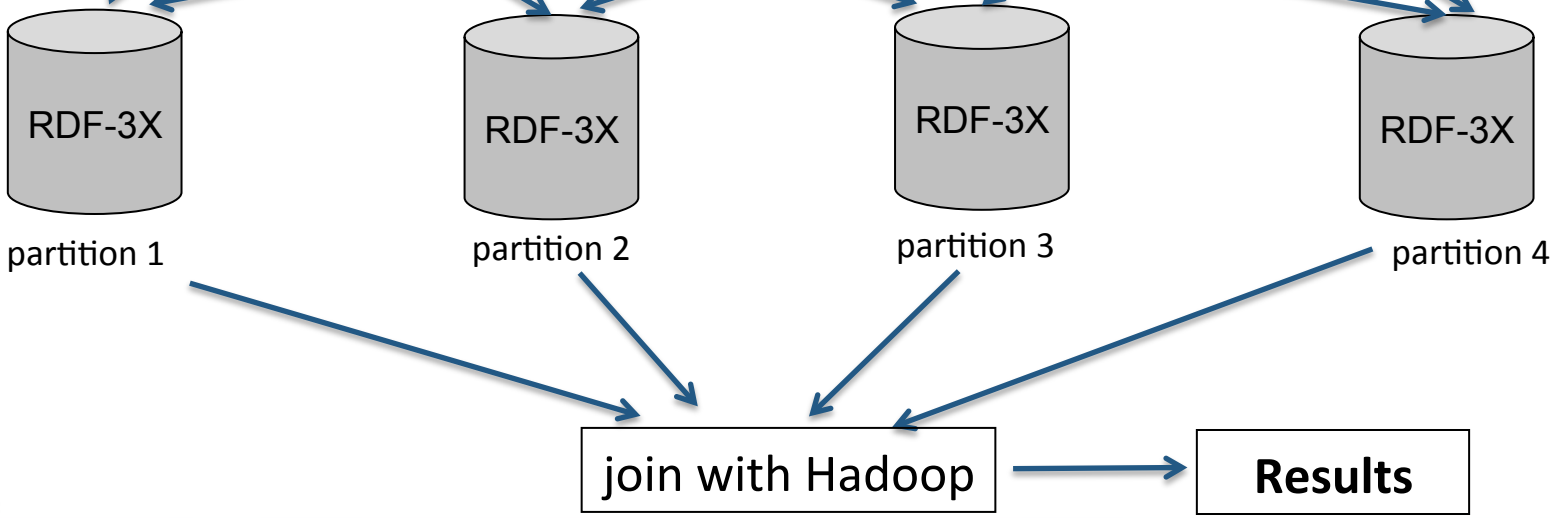
replication with
1-hop guarantee

**non PWOC query**

```
SELECT ?x ?y ?z
WHERE {  ?x :creates ?y .
         ?y :exhibitedIn ?z .
         ?z :locatedIn :paris .}
```

```
SELECT ?x ?y ?z
WHERE {  ?x :creates ?y .
         ?y :exhibitedIn ?z .}
```

```
SELECT ?z
WHERE {  ?z :locatedIn paris .}
```

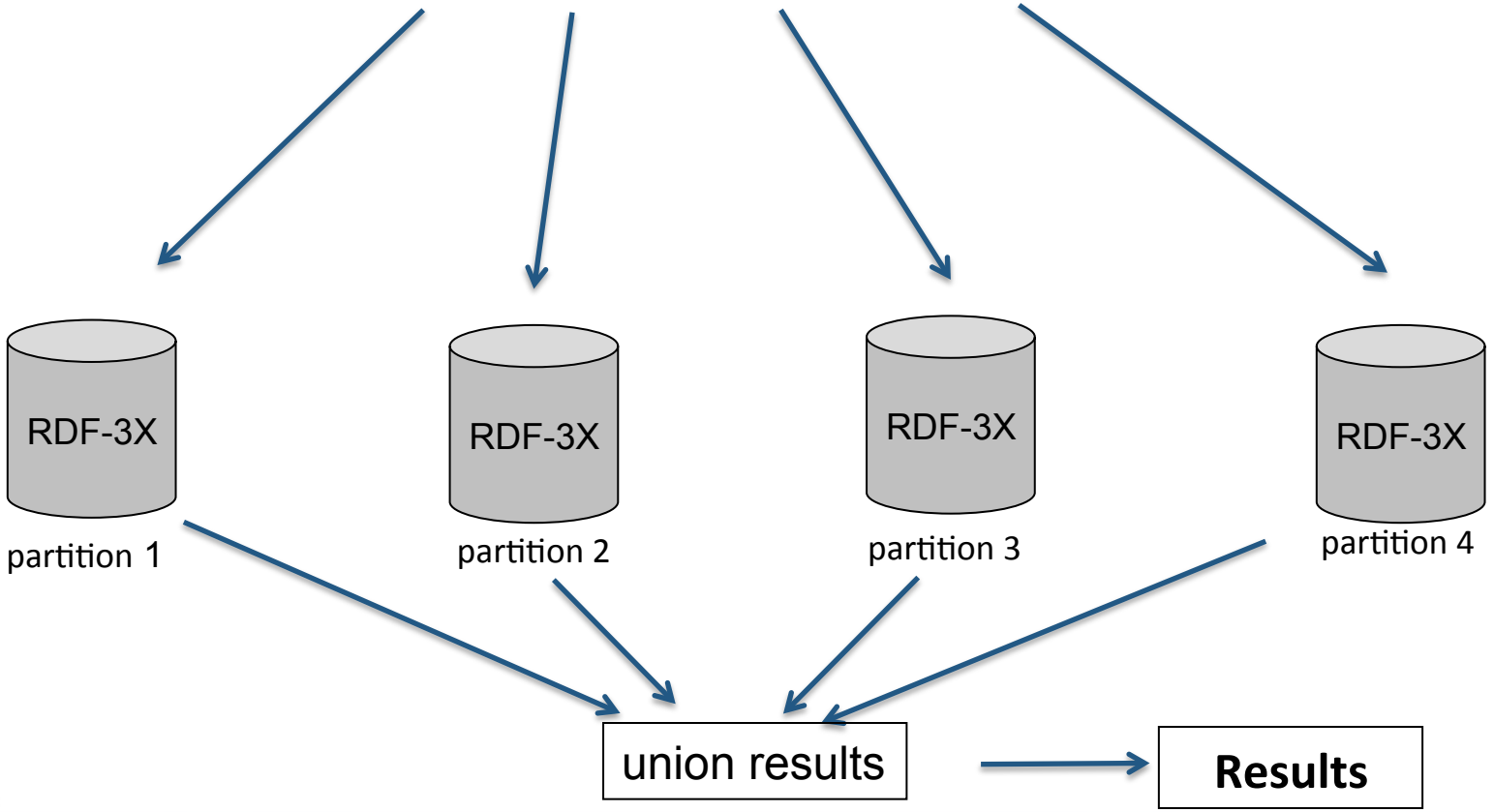


# Graph partitioning [Huang11] - querying

replication with
2-hop guarantee

**PWOC query**

```
SELECT ?x ?y ?z
WHERE {  ?x :creates ?y .
         ?y :exhibitedIn ?z .
         ?z :locatedIn :paris .}
```



# 3. Graph-based approaches

| Pros | Cons |
| --- | --- |
| Benefits from RDF structure | Not scalable for big datasets* |
| Data placement | Subqueries go to all machines |
| Shuffling is avoided | Uses a lot of resources |

# 4. RDF stores based on cloud services

- Statustore [Stein10]
  - AWS SimpleDB
- AMADA [Bugiotti12, Aranda12]
  - Generic cloud-based architecture
  - Built on top of Amazon Web Services as a SaaS
  - Route the queries to a small subset of the data that are likely to have matches
  - Trade-offs between efficiency and monetary costs

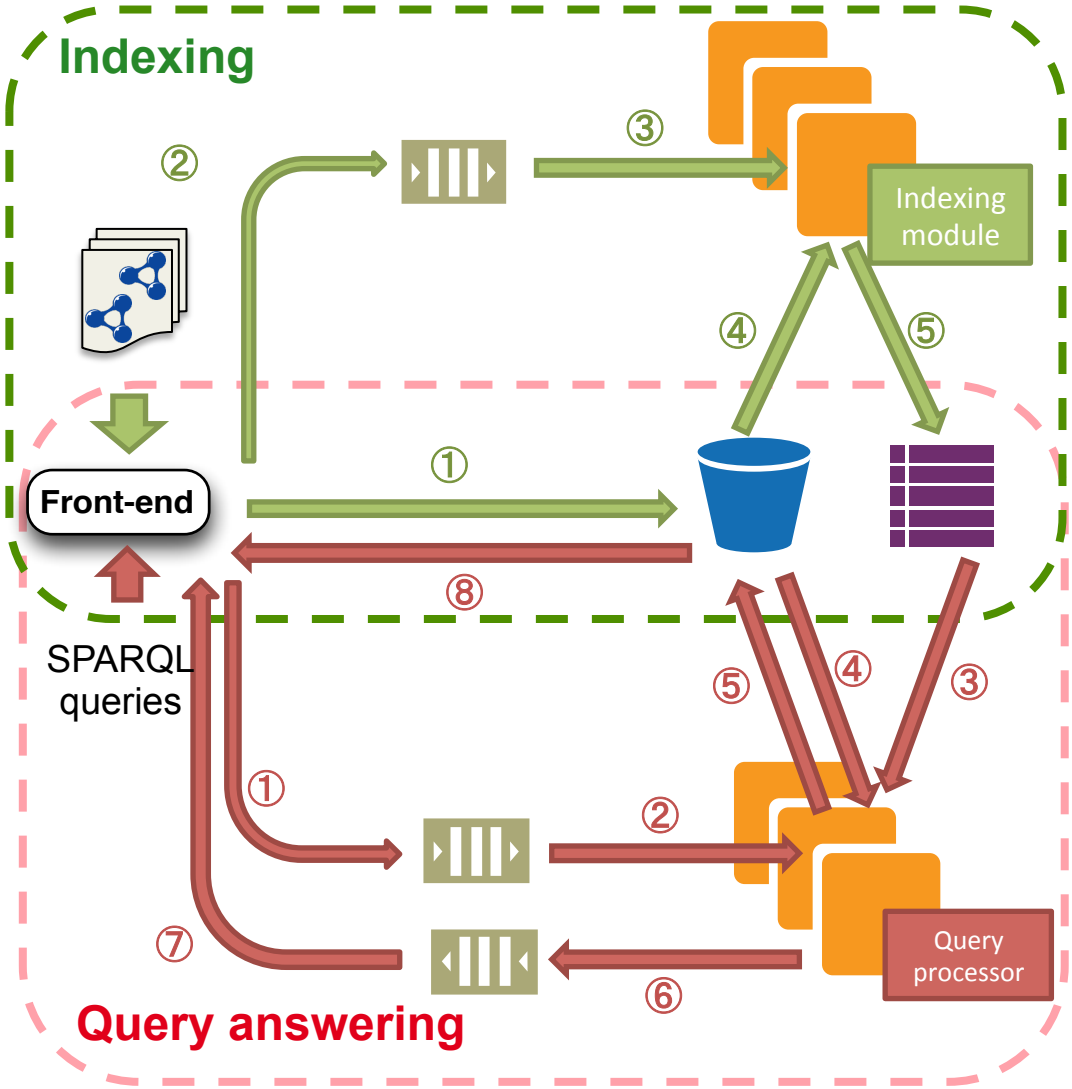
# AMADA architecture

**Key-value store**
Ideal for indexing and querying small, structured data

**Storage for raw data**
Ideal for large files

**Virtual machines**
Resizable computing capacity in the cloud

**Messaging queues**
Queues for communication between distributed components



# AMADA indexing and storage

## S table

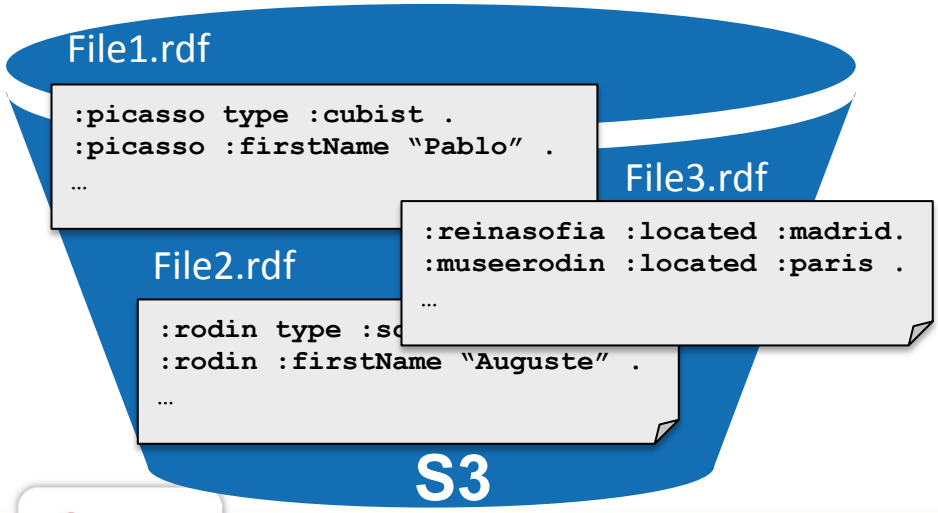
| key | (attribute, value) |
|---|---|
| :picasso | (file1, -) |
| :guernica | (file1, -) |
| :reinasofia | (file1, -) |
| :rodin | (file2, -) |
| :thethinker | (file2, -) |
| :museerodin | (file2, -) |
| … | … |

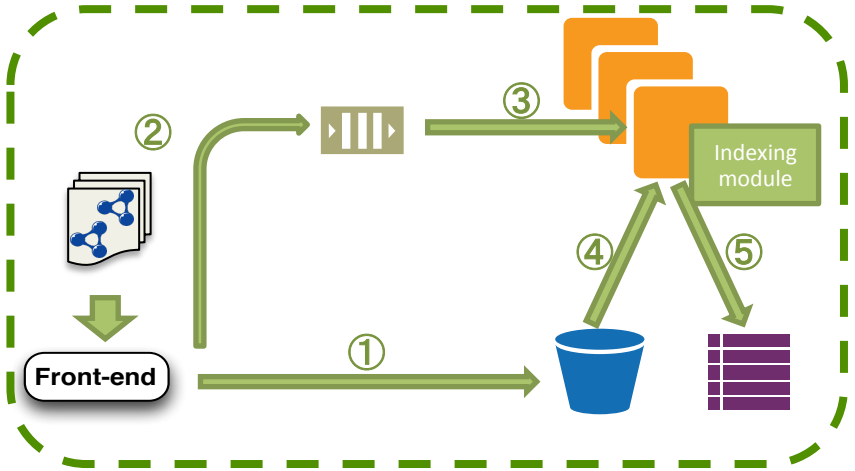
## P table

| key | (attribute, value) |
|---|---|
| type | (file1,-), (file2, -) |
| :firstName | (file1,-), (file2, -) |
| :paints | (file1,-) |
| :exhibitedIn | (file1,-), (file2, -) |
| :locatedIn | (file1,-) |
| :creates | (file2,-) |
| … | … |

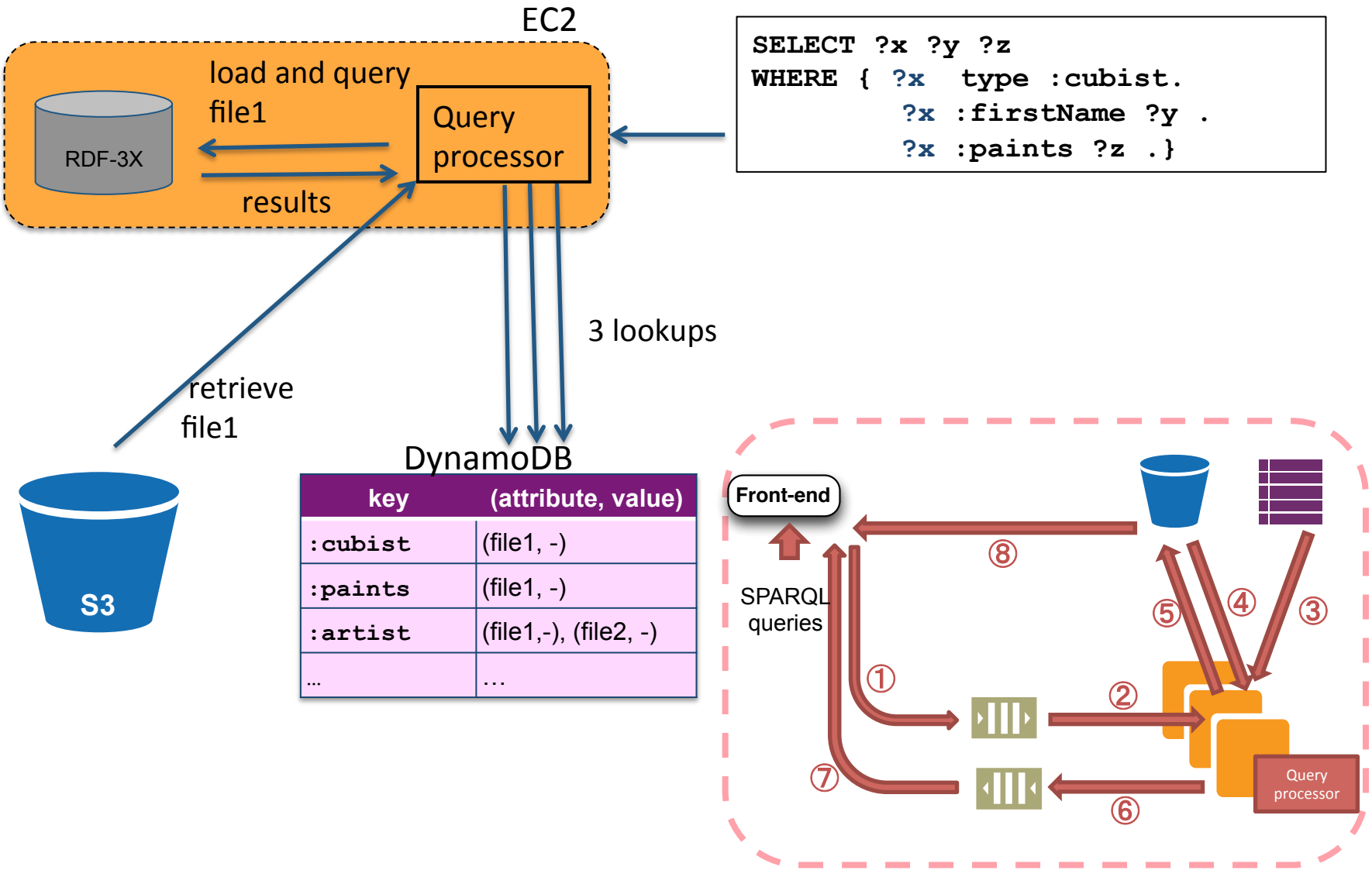
## O table

| key | (attribute, value) |
|---|---|
| :cubist | (file1,-) |
| :guernica | (file1,-) |
| "Pablo" | (file1,-) |
| :sculptor | (file2,-) |
| :thethinker | (file2,-) |
| :museerodin | (file2,-) |
| … | … |



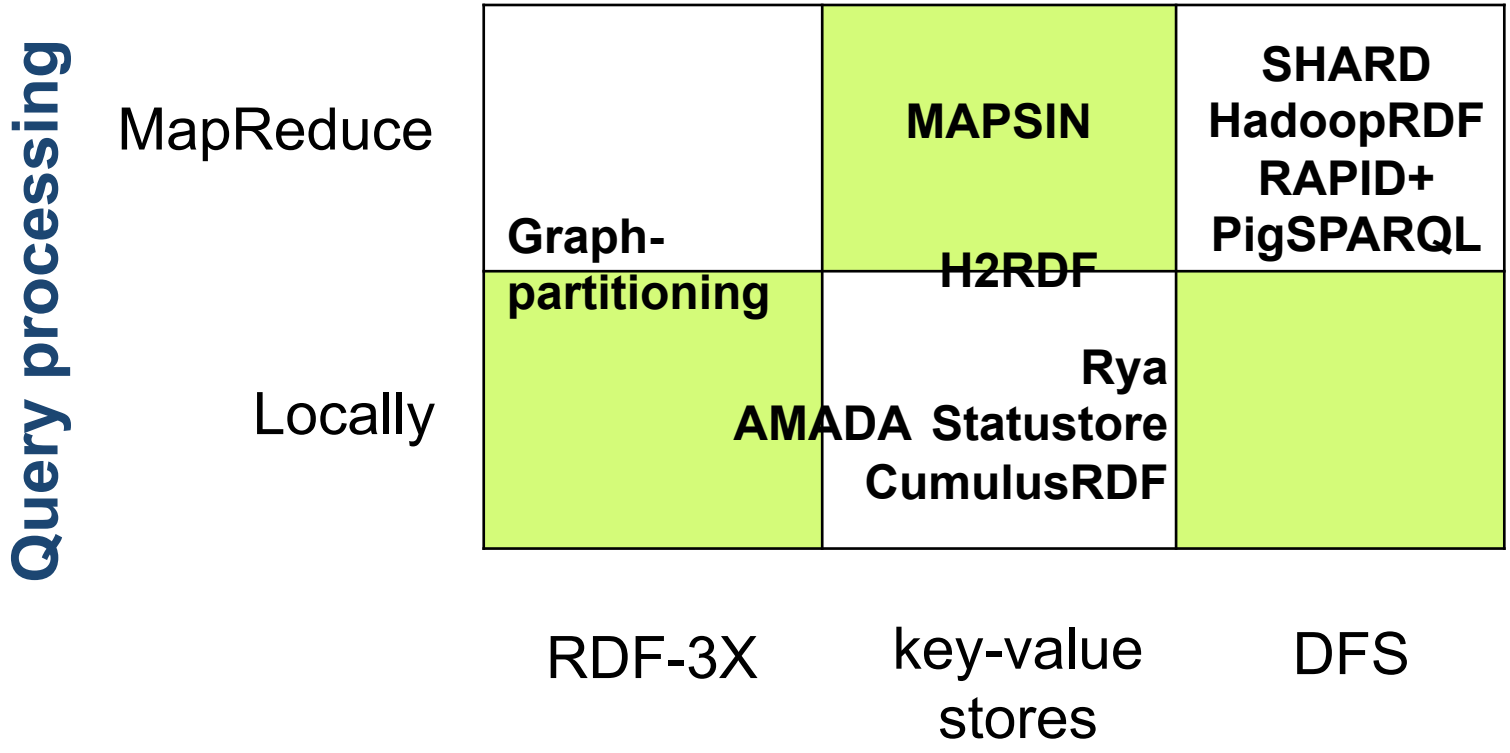
# AMADA querying



# 4. RDF stores based on cloud services

| Pros | Cons |
|---|---|
| Scalability | Depends on data partitioning |
| Fault-tolerance | Separation between storage and processing services |
| Elastic allocation of resources | Low latency for non-selective queries |
| Inter-query parallelism | No control on the services |
| Benefit from higher-level services (PaaS) | |

# Classification of systems



# Analysis dimensions

## Underlying data storage:

- Key-value stores
- DFS (distributed file system)
- Single-site RDF stores or data storage services supplied by cloud providers

## Query processing of conjunctive queries:

- MapReduce-based
- Local processing (joins are usually performed at a single site)

## RDFS entailment

# Analysis dimensions

## Underlying data storage:

- Key-value stores
- DFS (distributed file system)
- Single-site RDF stores or data storage services supplied by cloud providers

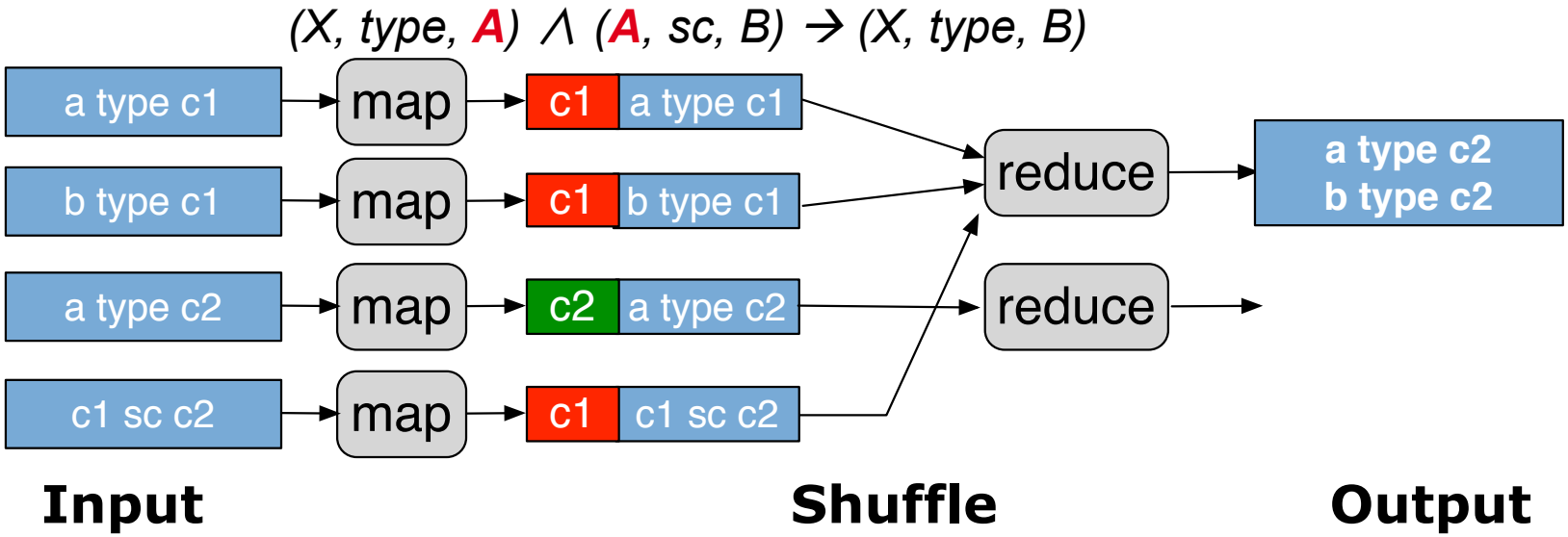
## Query processing of conjunctive queries:

- MapReduce-based
- Local processing (joins are usually performed at a single site)

**RDFS entailment**

# RDFS closure in the cloud

- Forward chaining in MapReduce: **WebPie** [Urbani09]

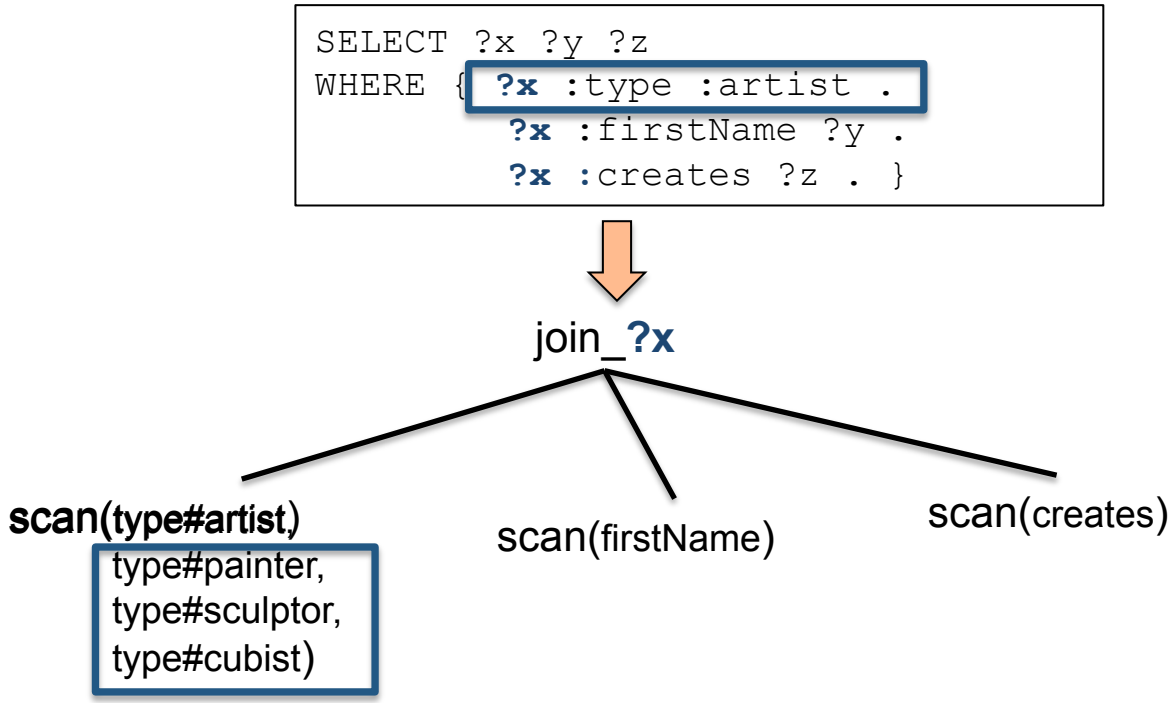



  - RDFS triples are kept in memory in each node
  - RDFS rules are executed in a specific order → the number of iterations is limited
- Full RDFS closure computation in parallel (MPI-based) [Weaver09]
  - embarrassingly parallel algorithm
  - RDFS triples are kept in memory in each process

# Query reformulation in the cloud

**HadoopRDF** [Husain11]

- Query reformulation during the file selection of the query processing




- Transitive closure computation and recursive query processing in MapReduce [Afrati12, Bu12] may apply to RDFS reasoning

# Hybrid approaches

- QueryPie [Urbani11]
  - Precompute implicit triples of the schema
  - Reformulate the query (faster reformulations)
- Rya [Punnoose12]
  - 1 MapReduce job to precompute subclass/subproperty hierarchies
  - Query reformulation at query time

# Analysis dimensions

## Underlying data storage:

- Key-value stores
- DFS (distributed file system)
- Single-site RDF stores or data storage services supplied by cloud providers

## Query processing of conjunctive queries:

- MapReduce-based
- Local processing (joins are usually performed at a single site)

## RDFS entailment:

- Materialization of RDFS closure
- Query reformulation
- Hybrid (only the schema closure is precomputed)

# Do we have a winner?

- No, at least not yet ☺
- Performance depends on query workload
  - Star queries or path queries?
  - Selective or analytics-style queries?
- Benchmarks on how real SPARQL queries look:
  - [Duan11]
  - [Arias11]
  - [Picalausa11]
  - Linked Data benchmark council (ldbc.eu)

# 4

# OPEN ISSUES

# Where we go from here?

- A lot of room for optimization
  - Improve query response time in the first place
  - Can we apply traditional optimization techniques (join ordering, statistics, etc.)?
- RDF data partitioning
  - Do the existing graph partitioning algorithms fit?
- RDFS entailment
  - Query reformulation
- More SPARQL features
  - Blank nodes, optional, negation, property paths, etc.

# Where we go from here?

- RDF views/indexes in the cloud
- SPARQL multi-query optimization in the cloud
- RDF updates in the cloud
- RDF data analytics
- How does cloud variability affect RDF data management?

# Thank you


**Slides can be found at:**
**http://pages.saclay.inria.fr/zoi.kaoudi/rdfcloud_icde13.pdf**

# References


- [Abadi07] D. J. Abadi, A. Marcus, S. Madden, K. J. Hollenbach, “Scalable Semantic Web Data Management Using Vertical Partitioning”, in VLDB 2007.
- [Afrati10] F. N. Afrati and J. D. Ullman, “Optimizing Joins in a Map-Reduce Environment,” in EDBT 2010.
- [Afrati11a] F. N. Afrati and J. D. Ullman, “Optimizing Multiway Joins in a Map-Reduce Environment,” IEEE Trans. Knowl. Data Eng., 2011.
- [Afrati11b] F. N. Afrati, V. R. Borkar, M. J. Carey, N. Polyzotis, and J. D. Ullman, “Map-Reduce Extensions and Recursive Queries,” in EDBT 2011.
- [Afrati12] F. N. Afrati, J. D. Ullman, “Transitive closure and recursive Datalog implemented on clusters” in EDBT 2012.
- [Alexaki01] S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis. “On Storing Voluminous RDF Descriptions: The Case of Web Portal Catalogs”, in WebDB 2001.
- [Aranda12] A. Aranda-Andujar, F. Bugiotti, J. Camacho-Rodriguez, D. Colazzo, F. Goasdoué, Z. Kaoudi, and I. Manolescu, “Amada: Web Data Repositories in the Amazon Cloud (demo)”, in CIKM 2012.
- [Arias11] M. Arias, J. D. Fernandez, M. A. Martínez-Prieto, “An Empirical Study of Real-World SPARQL Queries”, in USEWOD 2011.
- [Battre06] D. Battre, A. Hoing, F. Heine, O. Kao, “On Triple Dissemination, Forward-Chaining, and Load Balancing in DHT based RDF stores”, in DBISP2P 2006.
- [Blanas10] S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, and Y. Tian, “A Comparison of Join Algorithms for Log Processing in MapReduce,” in SIGMOD 2010.

# References


- [Bu12] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "The HaLoop Approach to Large-Scale Iterative Data Analysis," VLDB J, 2012.
- [Bugiotti12] F. Bugiotti, F. Goasdoué, Z. Kaoudi, and I. Manolescu, "RDF Data Management in the Amazon Cloud," in ICDT/EDBT Workshops 2012.
- [Cattell11] R. Cattell, "Scalable SQL and NoSQL data stores," SIGMOD Record, May 2011.
- [Dean13] J. Dean, L. A. Barroso, "Tail at Scale", communication of the ACM, February 2013.
- [Duan11] S. Duan, A. Kementsietsidis, K. Srinivas, O. Udrea, "Apples and oranges: a comparison of RDF benchmarks and real RDF datasets", in SIGMOD 2011.
- [Goasdoué13] F. Goasdoué, I. Manolescu, and A. Roatiș, "Efficient Query Answering against Dynamic RDF Databases", in EDBT 2013.
- [Hayes04] P. Hayes, "RDF Semantics", W3C Recommendation, February 2004, http://www.w3.org/TR/rdf- mt/.
- [Husain11] M. Husain, J. McGlothlin, M. M. Masud, L. Khan, and B. M. Thuraisingham, "Heuristics-Based Query Processing for Large RDF Graphs Using Cloud Computing," IEEE Trans. on Knowl. and Data Eng., 2011.
- [Iosup11] A. Iosup, N. Yigitbasi, D.H. J. Epema, "On the Performance Variability of Production Cloud Services", in CCGRID 2011.
- [Kaoudi13] Z. Kaoudi, M. Koubarakis, "Distributed RDFS Reasoning over Structured Overlay Networks", Journal on Data Semantics, 2013.
- [Kotoulas10] S. Kotoulas, E. Oren, F. Harmelen, "Mind the data skew: distributed inferencing by speeddating in elastic regions", in WWW 2010.

# References


- [Mallea11] A. Mallea, M. Arenas, A. Hogan, A. Polleres, “On Blank Nodes”, in ISWC 2011.
- [METIS] METIS - Serial Graph Partitioning and Fill-reducing Matrix Ordering: http://glaros.dtc.umn.edu/gkhome/metis/metis/overview
- [Neumann10] T. Neumann and G. Weikum, “The RDF-3X Engine for Scalable Management of RDF Data,” VLDBJ 2010.
- [Picalausa11] F. Picalausa, S. Vansummeren, “What are real SPARQL queries like?”, in SWIM 2011.
- [Punnoose12] R. Punnoose, A. Crainiceanu, and D. Rapp, “Rya: A Scalable RDF Triple Store for the Clouds,” in 1st International Workshop on Cloud Intelligence (in conjunction with VLDB), 2012.
- [Ravindra11] P. Ravindra, H. Kim, K. Anyanwu, “An Intermediate Algebra for Optimizing RDF Graph Pattern Matching on MapReduce”, in ESWC 2011.
- [Rohloff10] K. Rohloff and R. E. Schantz, “High-Performance, Massively Scalable Distributed Systems using the MapReduce Software Framework: the SHARD Triple-Store,” in Programming Support Innovations for Emerging Distributed Applications, 2010.
- [Schad10] J. Schad, J. Dittrich, J. Quiané-Ruiz, “Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance”, PVLDB 2010.
- [Schätzle12] A. Schätzle, M. Przyjaciel-Zablocki, C. Dorner, T. Hornung, G. Lausen, “Cascading Map-Side Joins over HBase for Scalable Join Processing”, in SSWS+HPCSW 2012.
- [Schätzle11] A. Schätzle, M. Przyjaciel-Zablocki, G. Lausen, ”PigSPARQL: mapping SPARQL to Pig Latin”, in SWIM 2011.

# References


- [Stein10] R. Stein and V. Zacharias, "RDF On Cloud Number Nine," in 4th Workshop on New Forms of Reasoning for the Semantic Web: Scalable and Dynamic, May 2010.
- [Theoharis05] Y. Theoharis, V. Christophides, and G. Karvounarakis. "Benchmarking Database Representations of RDF/S Stores", in ISWC 2005.
- [Urbani09] J. Urbani, S. Kotoulas, E. Oren, and F. van Harmelen, "Scalable Distributed Reasoning using MapReduce," in ISWC 2009.
- [Urbani11] J. Urbani, F. van Harmelen, S. Schlobach, and H. Bal, "QueryPIE: Backward Reasoning for OWL Horst over Very Large Knowledge Bases," in ISWC 2011.
- [Weaver09] J. Weaver and J. A. Hendler, "Parallel Materialization of the Finite RDFS Closure for Hundreds of Millions of Triples", in ISWC 2009.
- [Weiss08] C. Weiss, P. Karras, A. Bernstein, "Hexastore: sextuple indexing for semantic web data management", PVLDB 2008.
- [Wilkinson06] K. Wilkinson, "Jena Property Table Implementation", in SSWS 2006.
- [Wu11] S. Wu, F. Li, S. Mehrotra, B. C. Ooi, "Query Optimization for Massively Parallel Data Processing", in SOCC 2011.
- [Zeng13] K. Zeng, J. Yang, H. Wang, B. Shao, and Z. Wang, "A Distributed Graph Engine for Web Scale RDF Data", in PVLDB 2013.