Big Linked Data Storage and Query Processing

Prof. Sherif Sakr

ACM and IEEE Distinguished Speaker

The 3rd KEYSTONE Training School on Keyword search in Big Linked Data
Vienna, Austria
August 21, 2017

http://www.cse.unsw.edu.au/~ssakr/
ssakr@cse.unsw.edu.au

Motivation: Tutorial Goal

- **Overall Goal:** Comprehensive review of systems and techniques that tackle data storage and querying challenges of big RDF databases
  - Categorize Existing Systems
  - Survey State-of-the-Art Techniques
- **Intended Takeaways**
  - Awareness of existing systems and techniques
  - Survey of effective storage and query optimization techniques of RDF databases
  - Overview of open research problems
- **What this Tutorial is Not?**
  - Introduction to Big Data
  - Introduction to Semantic Web and RDF
  - Introduction to SPARQL

Today's Agenda

- **Overview of RDF and SPARQL**
- **Taxonomy of RDF Processing Systems**
  - Centralized RDF Processing Systems
  - Distributed RDF Processing Systems
- **Open Challenges in Big RDF Processing Systems**
- **Conclusions**

Part I

Overview of RDF and SPARQL

RDF

- RDF, the Resource Description Framework, is a data model that provides the means to describe resources in a semi-structured manner.
- RDF is gaining widespread momentum and usage in different domains such as Semantic Web, Linked Data, Open Data, social networks, digital libraries, bioinformatics, or business intelligence.
- A number of ontologies and knowledge bases storing millions to billions of facts such as *DBpedia*[1], *Probase*[2] and *Wikidata*[3] that are now publicly available.
- key search engines like Google and Bing are providing better support for RDF.

[1] `http://wiki.dbpedia.org/`
[2] `https://www.microsoft.com/en-us/research/project/probase/`
[3] `https://www.wikidata.org/`

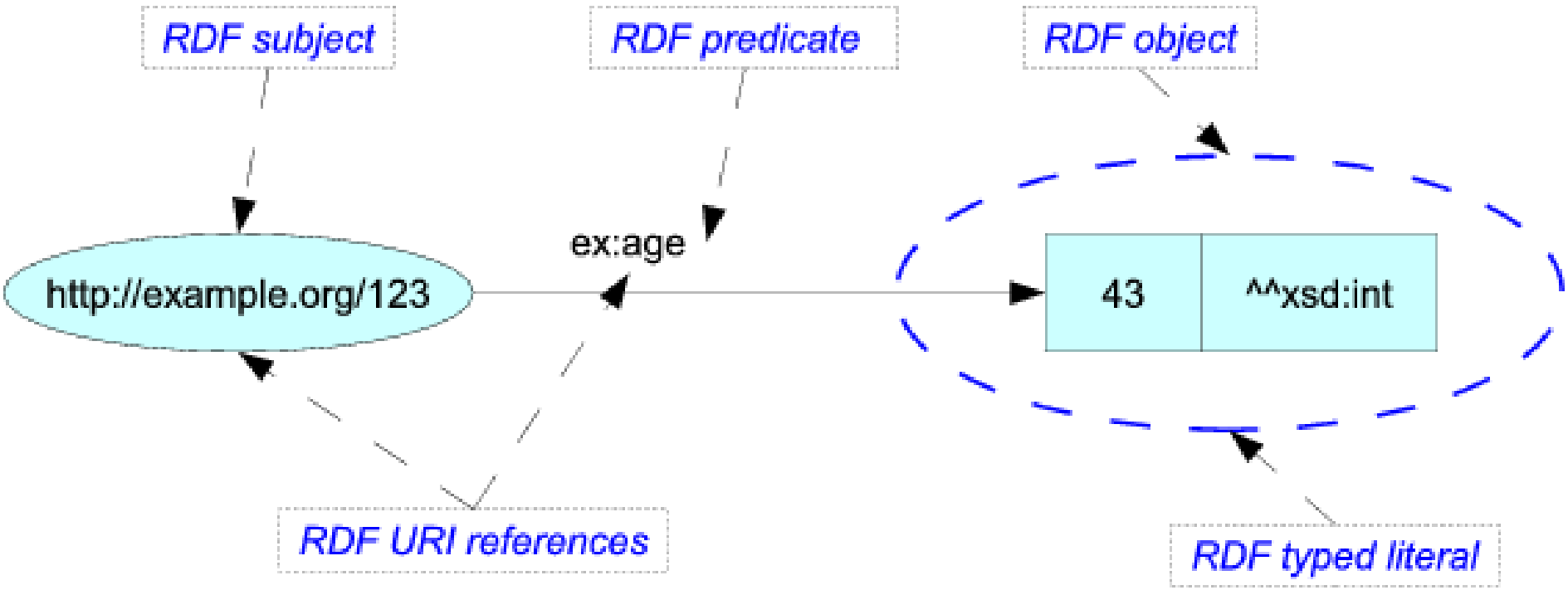
RDF

- RDF is designed to flexibly model schema-free information which represents data objects as triples, each of the form **(S, P, O)**, where S represents a *subject*, P represents a *predicate* and O represents an *object*.

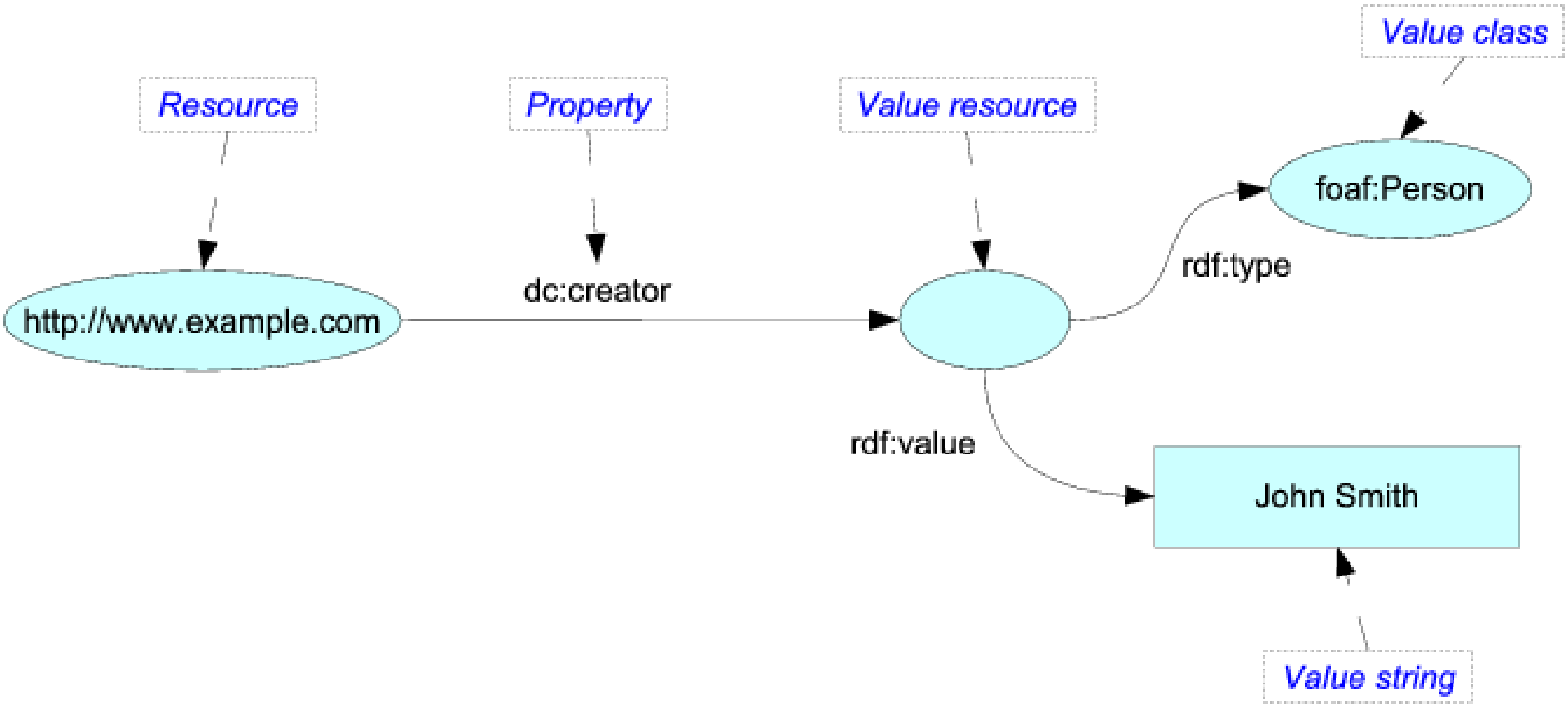
- A triple indicates a relationship between S and O captured by P. Consequently, a collection of triples can be represented as a directed graph where the graph vertices denote subjects and objects while graph edges are used to denote predicates.

- The same resource can be used in multiple triples playing the same or different roles, e.g., it can be used as the subject in one triple and as the object in another. This ability enables to define multiple connections between the triples, hence creating a connected graph of data.

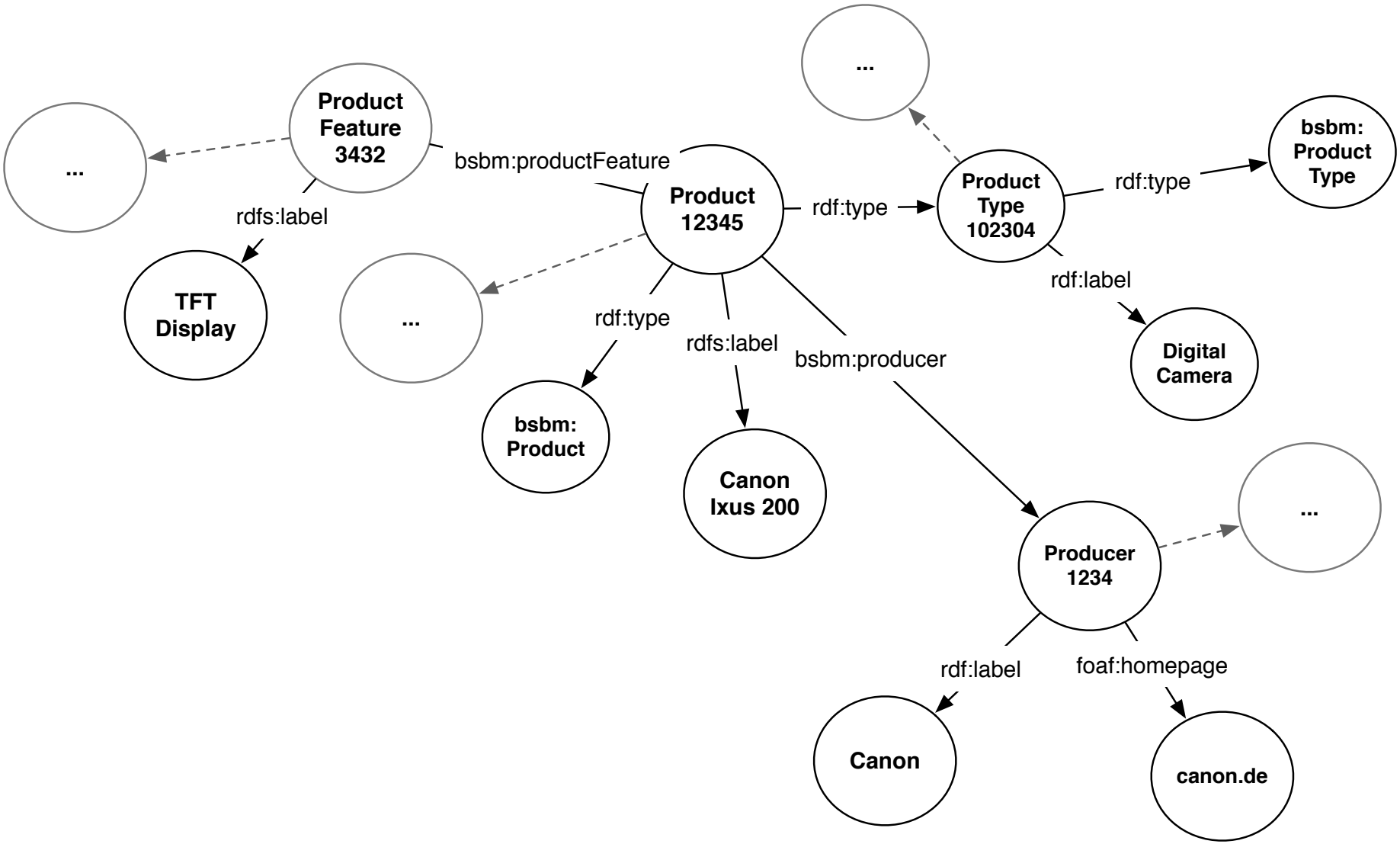
RDF


RDF subject
RDF predicate
RDF object
http://example.org/123
ex:age
43
^^xsd:int
RDF URI references
RDF typed literal

RDF


Resource
Property
Value resource
Value class
foaf:Person
rdf:type
http://www.example.com
dc:creator
rdf:value
John Smith
Value string

RDF

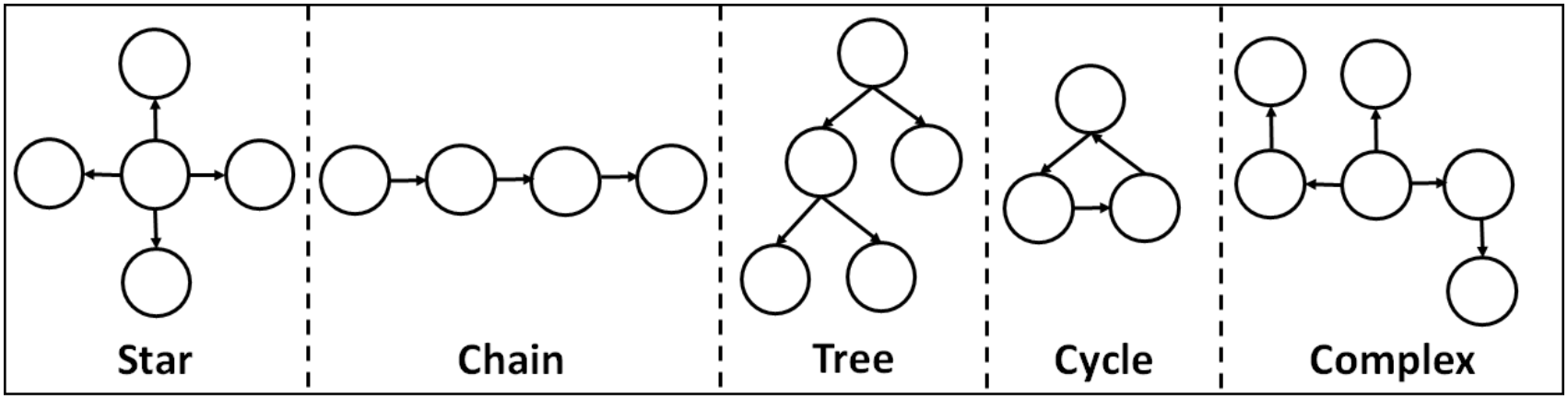

...
Product Feature 3432
bsbm:productFeature
Product 12345
rdf:type
Product Type 102304
rdf:type
bsbm: Product Type
rdfs:label
TFT Display
...
rdf:type
rdfs:label
bsbm:producer
rdf:label
Digital Camera
bsbm: Product
Canon Ixus 200
...
Producer 1234
...
rdf:label
foaf:homepage
Canon
canon.de

SPARQL

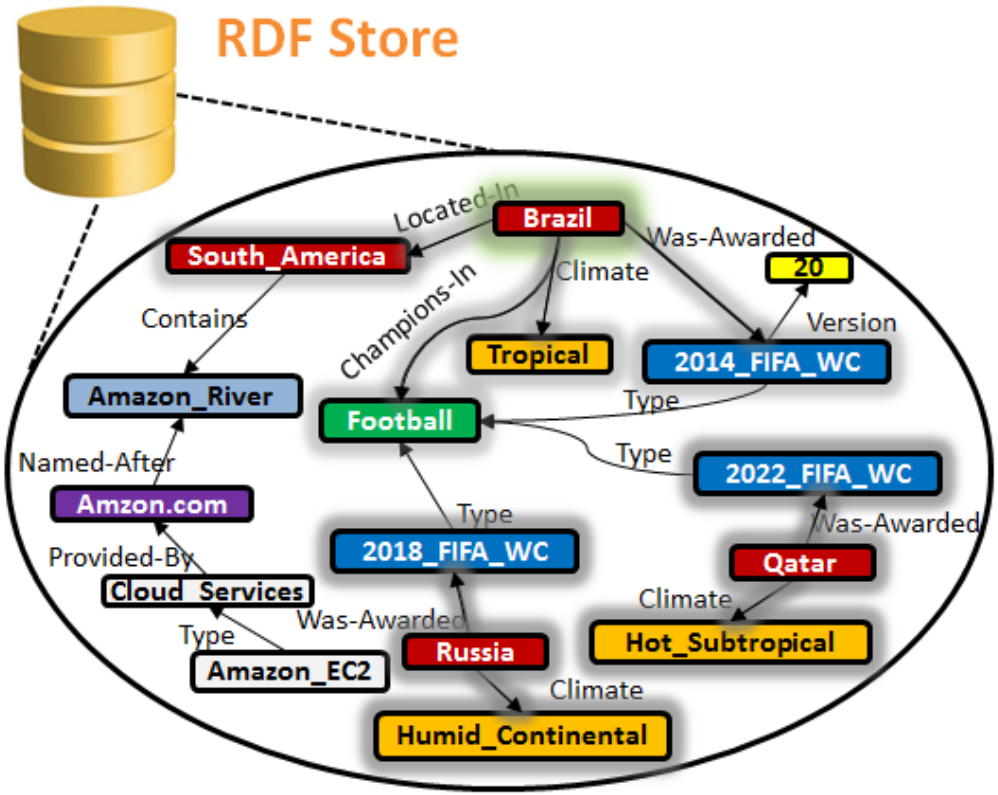
- The SPARQL query language has been recommended by the W3C as the standard language for querying RDF data.
- A SPARQL query Q specifies a graph pattern P which is matched against an RDF graph G.
- The query matching process is performed via matching the variables in P with elements of G such that the returned graph is contained in G (**graph pattern matching**).
- A triple pattern is much like a triple, except that S, P and/or O can be replaced by variables.
- Similar to triples, triple patterns can be modeled as directed graphs. A set of triple patterns is called a *basic graph pattern (BGP)* and SPARQL expressions that only contain such type of patterns are called **BGP queries**.

Shapes of SPARQL BGP Queries

- **Star query**: only consists of subject-subject joins where each join variable is the subject of all the triple patterns involved in the query.
- **Chain query**: consist of subject-object joins where the triple patterns are consecutively connected like a chain.
- **Tree query**: consists of subject-subject joins and subject-object joins.
- **Cycle query**: contains subject-subject joins, subject-object joins and object-object join.
- **Complex query**: combination of different shapes.



SPARQL



A SPARQL QUERY Q :

```
SELECT ?country
WHERE {
?country Located-In South_America
?country Champions-In Football }
```

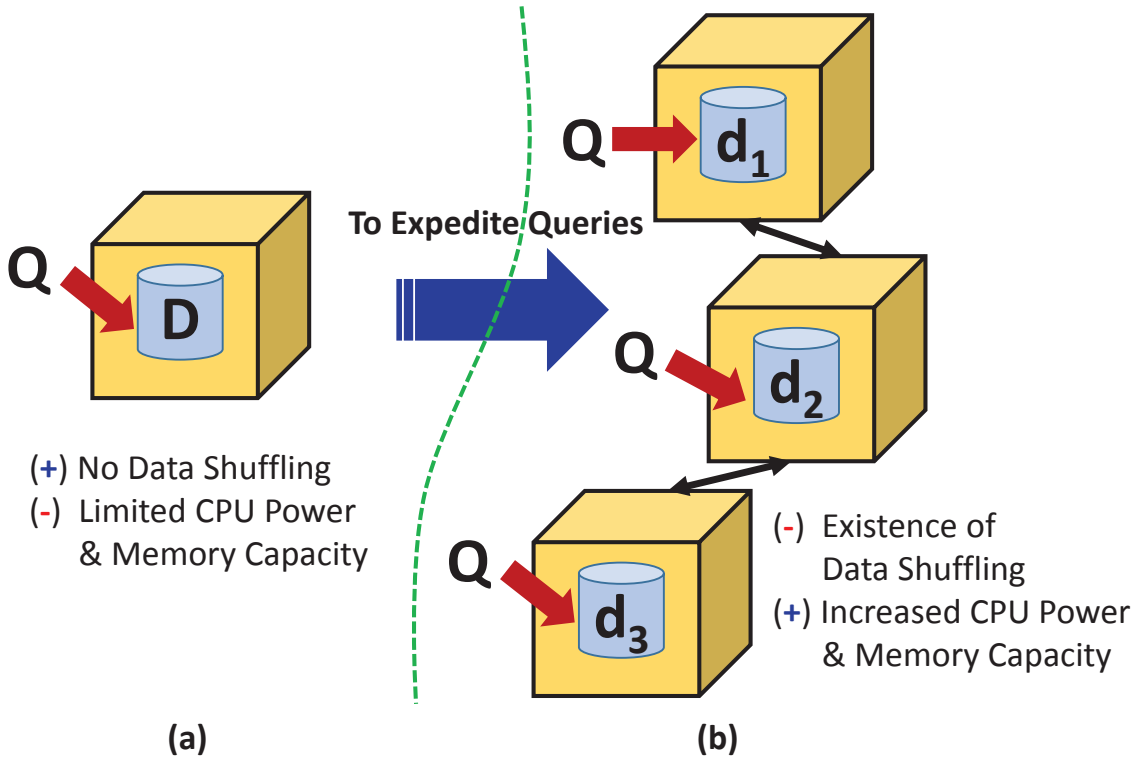
FINAL RESULT SET :

?country
Brazil

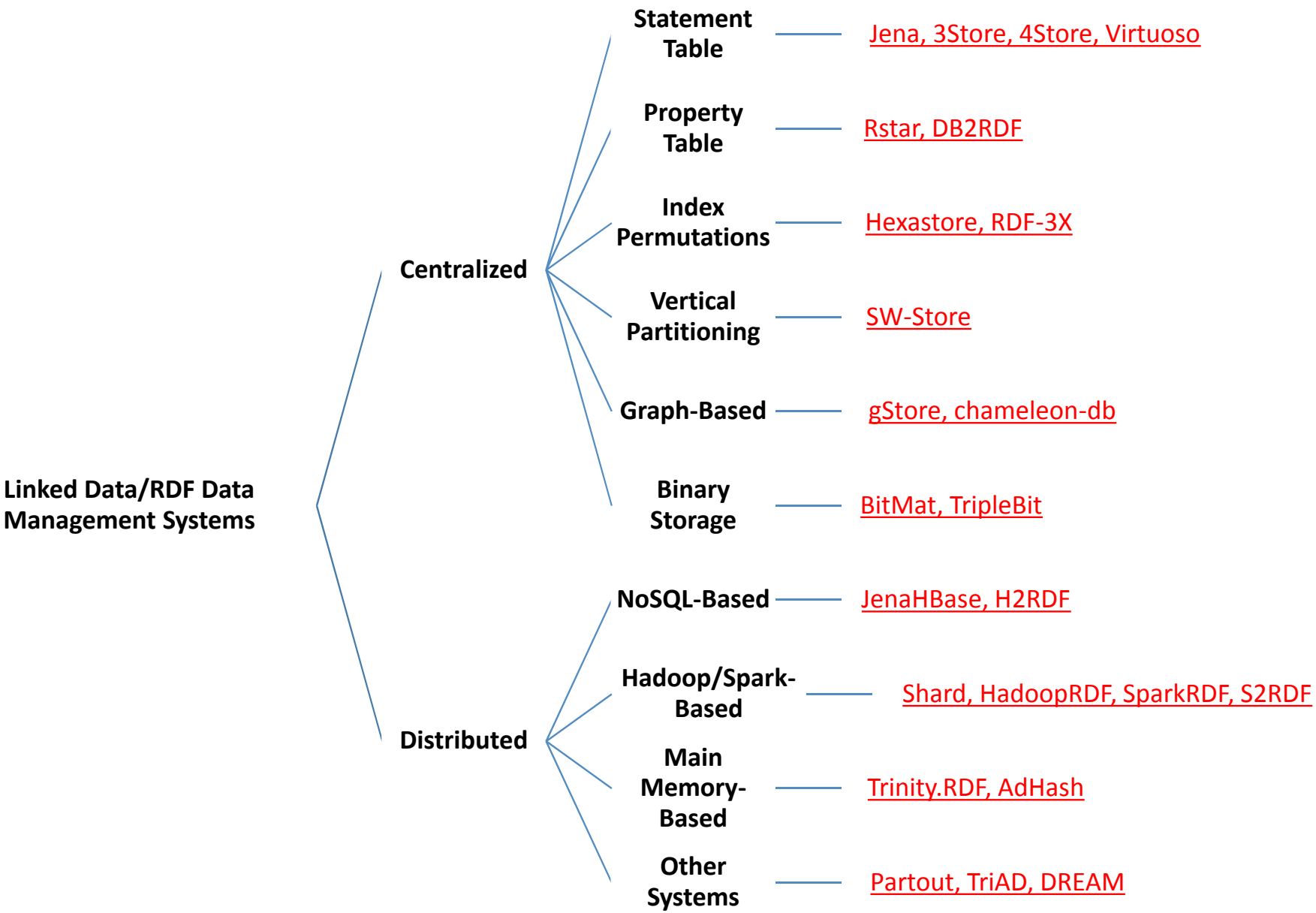
Centralized Systems Vs Distributed Systems

The wide adoption of the RDF data model has called for efficient and scalable RDF querying schemes.

- **Centralized systems**: where the storage and query processing of RDF data is managed on a *single* node.
- **Distributed systems**: where the storage and query processing of RDF data is managed on *multiple* nodes.



Taxonomy of RDF Processing Systems



Part II

Centralized RDF Processing Systems

Statement Tables

- A straightforward way to persist RDF triples is to store triple statements directly in a table-like structure as a linearized list of triples (ternary tuples).

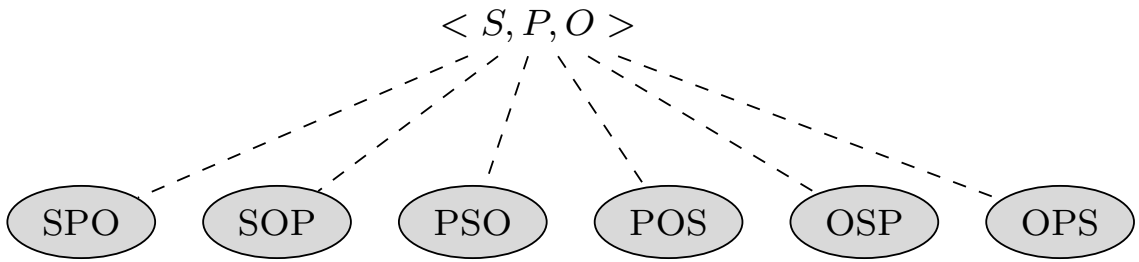
Subject	Predicate	Object
Product12345	rdf:type	bsbm:Product
Product12345	rdfs:label	Canon Ixus 2010
Product12345	bsbm:producer	bsbm-inst:Producer1234
...	...	...
Producer1234	rdf:label	Canon
Producer1234	foaf:homepage	http://www.canon.com
...	...	...

- A common approach is to encode URIs and Strings as IDs and two separate dictionaries are maintained for literals and resources/URIs.
- Example systems include Jena[4], 3Store[5] , 4Store[6] and Virtuoso[7]

[4] https://jena.apache.org/
[5] https://sourceforge.net/projects/threestore/
[6] https://github.com/4store/4store
[7] https://virtuoso.openlinksw.com/

Indexing Permutations

- This approach exploits and optimizes traditional indexing techniques for storing RDF data by applying exhaustive indexing over the RDF triples.
- All possible combinations the three components is indexed and materialized.



- The foundation for this approach is that any query can be answered using the available indices so that it allows fast access to all parts of the triples by sorted lists and fast merge-joins.
- Example systems include Hexastore[8] and RDF-3x[9]

[8] Weiss, Cathrin, Panagiotis Karras, and Abraham Bernstein. *Hexastore: sextuple indexing for semantic web data management*. PVLDB 2008

[9] Neumann, Thomas, and Gerhard Weikum. *RDF-3X: a RISC-style engine for RDF*. PVLDB 2008

Property Tables

- RDF does not describe any specific schema for the graph.
- There is no definite notion of schema stability, meaning that at any time the data schema might change.
- There is no easy way to determine a set of partitioning or clustering criteria to derive a set of tables to store the information.
- Storing RDF triples in a *single large statement* table presents a number of disadvantages when it comes to query evaluation. In most cases, for each set of triple patterns which is evaluated in the query, a set of self-joins is necessary to evaluate the graph traversal.
- Since the single statement table can become very large, this can have a negative effect on query execution.

Property Tables

- The main goal of **clustered property tables** is to cluster commonly accessed nodes in the graph together in a single table to avoid the expensive cost of many self-join operations on the large statement table encoding the RDF data.
- The property tables approach attempts to improve the performance of evaluating RDF queries by decreasing the cost of the join operation via reducing the number of required

Product Property Table

Subject	Type	Label	NumericProperty1	aaa
Product12345	bsbm:Product	Canon Ixus 2010	NULL	...
...	...	...	...	...

Left-Over Triples

Subject	Predicate	Object
Producer1234	foaf:homepage	http://www.canon.com
...	...	...

- Example systems include DB2RDF[10], Jena2

[10] Bornea, Mihaela A., et al. *Building an efficient RDF store over a relational database*. SIGMOD, 2013

Vertical Partitioning

- This approach applies a fully decomposed storage model.
- The approach rewrites the triple table into *m* tables where *m* is the number of unique properties in the dataset. Each of the *m* table consists of two columns: subject and the object value. The subjects which are not described by a particular property are simply omitted from the table for that property. For the case of a multi-valued attribute, each distinct value is listed in a successive row in the table for that property.
- Each of the *m* tables is indexed by subject so that particular subjects can be retrieved quickly. Fast merge join operations are exploited to reconstruct information about multiple properties for subsets of subjects.

<rdf:type>

Subject	Object
Product12345	bsbm:Product

<rdfs:label>

Subject	Object
Product12345	Canon Ixus 2010
Producer1234	Canon

<aaa>

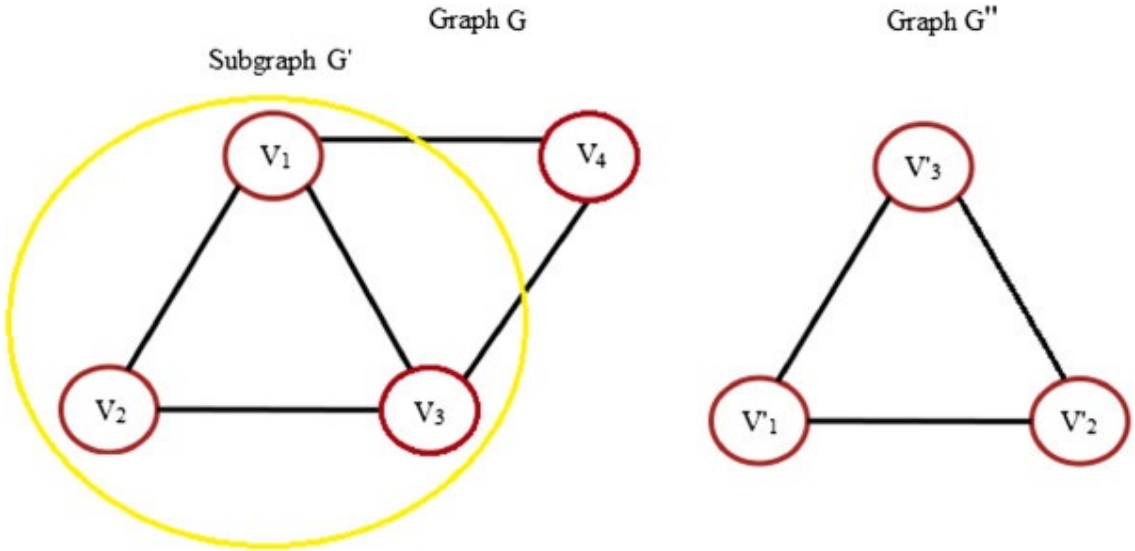
Subject	Object
uuu	xxx
...	...

- Example systems include SW-Store[11]

[11] Abadi, Daniel J., et al. *Scalable semantic web data management using vertical partitioning*. VLDB, 2007

Graph-Based Storage

- RDF naturally forms graph structures, hence one way to store and process it is through graph-driven data structures and algorithms.
- Some approaches have applied ideas from the graph querying world to efficiently handle RDF data.
- SPARQL queries are treated as sub-graph matching problem.



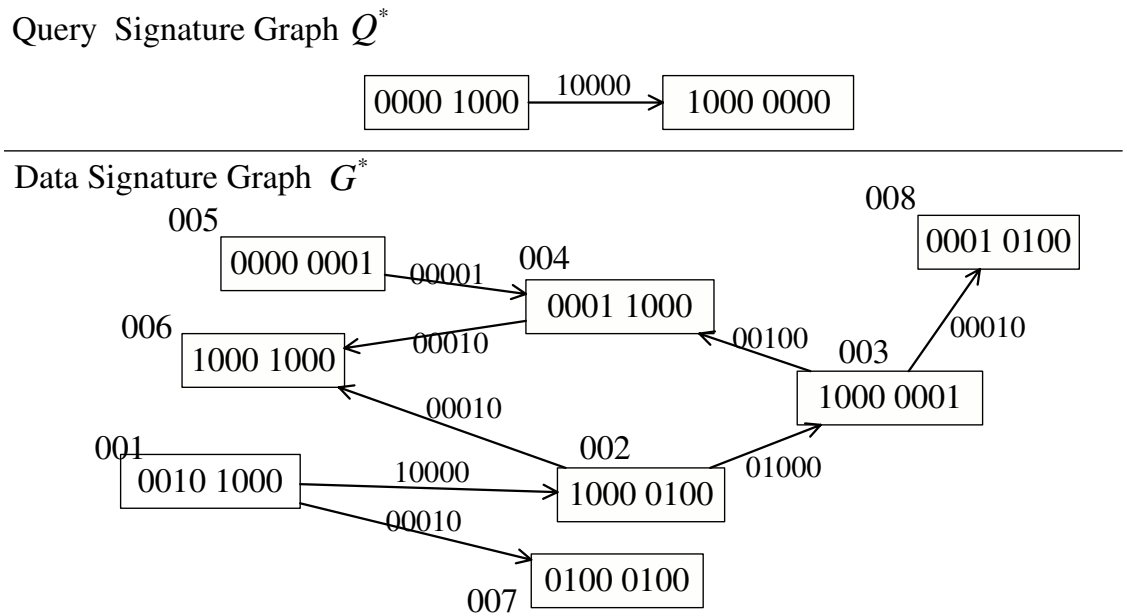
- Example systems include gStore, chameleon-db, Turbo$_{HOM++}$[12], AMbER[13]

[12] Kim et al. *Taming subgraph isomorphism for RDF query processing*. PVLDB, 2015

[13] Ingalalli et al. *Querying RDF Data Using A Multigraph-based Approach*. EDBT, 2016.

Graph-Based Storage: gStore[14]

- RDF graph is stored as a disk-based adjacency list table.
- For each class vertex in the RDF graph, gStore assigns a bit string as its vertex signature.
- During query processing, the vertices of the SPARQL query are encoded into vertex signatures and then the query is encoded into its corresponding query signature graph.
- Answering the SPARQL query is done by matching the vertex signature of the query graph over vertex signature of the RDF graph.



[14] Zou, Lei, et al. *gStore: a graph-based SPARQL query engine*. VLDB J., 2014

Graph-Based Storage: chameleon-db[15]

- A workload-aware RDF data management system that automatically adjusts its layout of the RDF database with the aim of optimizing the query execution time and auto-tuning its performance.
- In contrast to gStore which evaluates the queries over the entire RDF graph, chameleon-db partitions the RDF graph and prunes out the irrelevant partitions during query evaluation by using partition indexes.
- The main goal of the partitioning strategy is to carefully identify the graph partitions that truly contribute to the final results in order to minimize the number of dormant triples which is required to be processed during query evaluation and hence improve the system performance for that workload.
- To prune the irrelevant partitions, it uses an incremental indexing technique that uses a decision tree to keep track of which segments are relevant to which queries.

[15] Aluc et al. *chameleon-db: a workload-aware robust RDF data management system*. University of Waterloo, Tech. Rep. CS-2013-10, 2013.

Binary Storage: BitMat[16]

- A 3-dimensional (subject, predicate, object) bit matrix which is flattened in 2-dimensions for representing RDF triples.
- Each element of the matrix is a bit encoding the absence or presence of that triple. Therefore, very large RDF triple-sets can be represented compactly in memory as BitMats.
- The data is compressed on each row, using RLE, and Bitwise AND/OR operators are used to process join queries expressed as conjunctive patterns.
- During query processing, the BitMat representation allows fast identification of candidate result triples in addition to providing a compact representation of the intermediate results for multi-joins.

[16] Atre et al. *Bitmat: A main-memory bit matrix of rdf triples for conjunctive triple pattern queries*. ISWC, 2008.

Binary Storage: BitMat

Subject	Predicate	Object
:the_matrix	:released_in	"1999"
:the_thirteenth_floor	:released_in	"1999"
:the_thirteenth_floor	:similar_plot_as	:the_matrix
:the_matrix	:is_a	:movie
:the_thirteenth_floor	:is_a	:movie

Distinct subjects: [:the_matrix, :the_thirteenth_floor]
Distinct predicates: [:released_in, :similar_plot_as, :is_a]
Distinct objects: [:the_matrix, "1999", :movie]

	:released_in	:similar_plot_as	:is_a
:the_matrix	0 1 0	0 0 0	0 0 1
:the_thirteenth_floor	0 1 0	1 0 0	0 0 1

Note: Each bit sequence represents sequence of objects (:the_matrix, "1999", :movie)

Binary Storage: TripleBit[17]

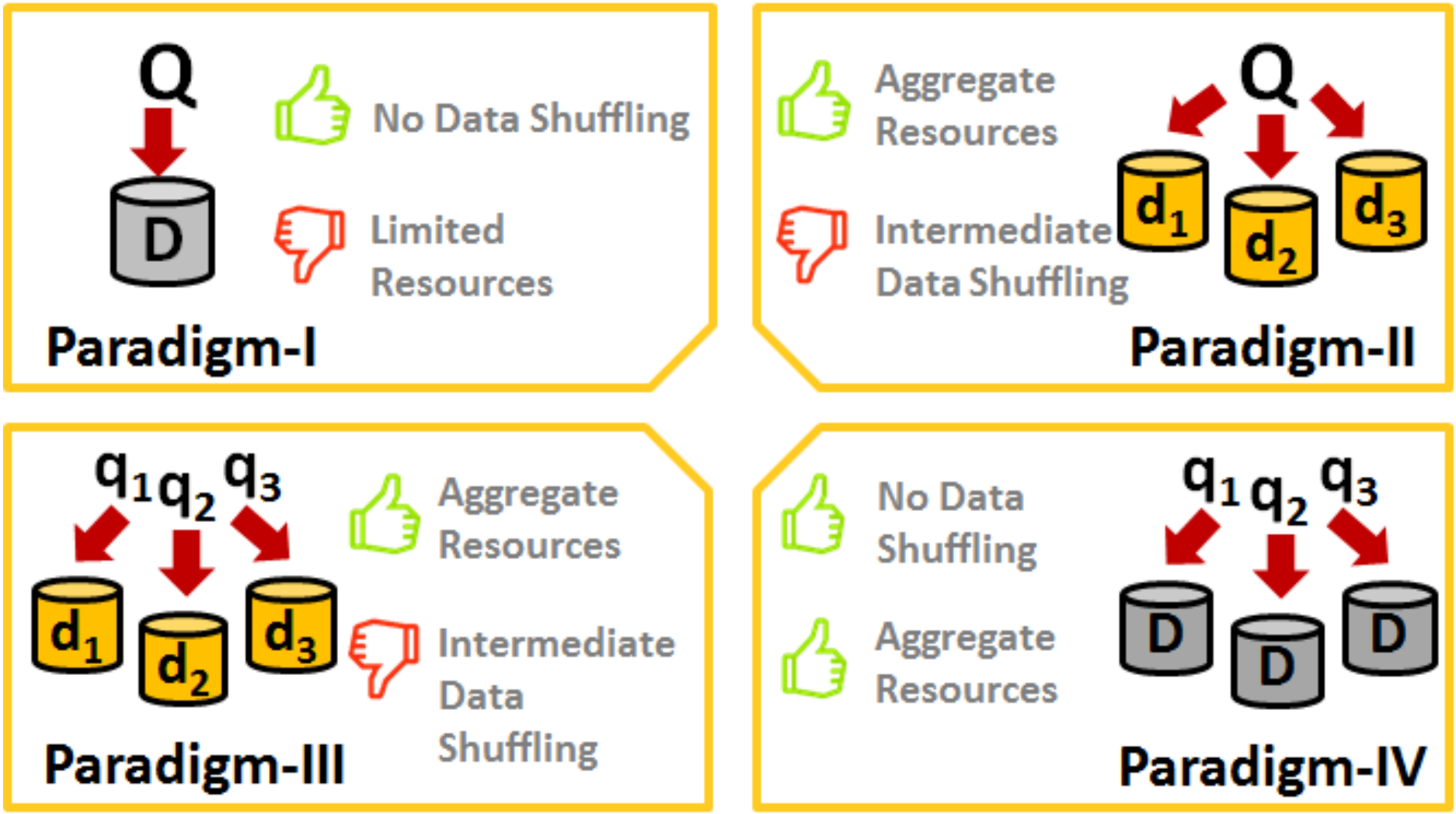
- It is designed as a storage structure that can directly and efficiently query the compressed data.
- TripleBit sorts the columns by predicates in lexicographic order and vertically partition the matrix into multiple disjoint buckets, one per predicate.
- TripleBit uses two auxiliary indexing structures:
  - *ID-Chunk bit matrix*: supports a fast search of the relevant chunks matching to a given subject or object.
  - *ID-Predicate bit matrix*: provides a mapping of a subject (S) or an object (O) to the list of predicates to which it relates.
- These indexing structures are effectively used to improve the speedup for scan and merge-join performance.

[17] Yuan et al., *TripleBit: a fast and compact system for large scale RDF data*. PVLDB, 2013

Part III

Distributed RDF Processing Systems

Processing Models of RDF Systems



NoSQL Databases

- NoSQL database systems represent a new generation of low-cost, high-performance database software which is increasingly gaining more and more popularity.
- These systems promise to simplify administration, be fault-tolerant and able to scale on commodity hardware (Scale out).
- The original intention has been **modern web-scale databases**. The movement began early 2009 and is growing rapidly.
- **Design Features of NoSQL Database Systems**
  - The ability to horizontally scale out throughput over many servers.
  - A simple call level interface or protocol (in contrast to a SQL binding).
  - Efficient use of distributed indexes and RAM for data storage.
  - The ability to dynamically define new attributes or data schema.

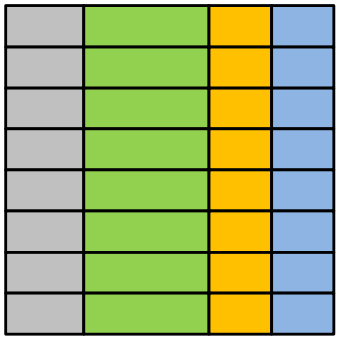
Main Categories of NoSQL Database Systems[18]

- **Key-value stores**: A collection of objects where each object has a unique key and a set of attribute/ value pairs.
- **Extensible record stores**: Variable-width tables (Column Families) that can be partitioned vertically and horizontally across multiple servers.
- **Document stores**: Consists of objects with a variable number of attributes with a possibility of having nested objects.
- **Graph stores**: A database that uses graph structures with nodes, edges, and properties to represent and store data. objects.

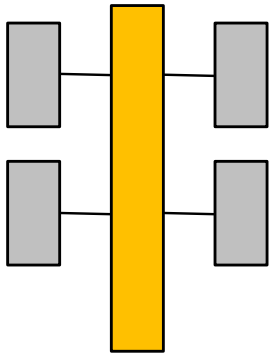
[18] Sakr et al. "*A Survey of Large Scale Data Management Approaches in Cloud Environments*".IEEE Communications Surveys and Tutorials (IEEE COMST) 13(3), 2011

Main Categories of NoSQL Database Systems

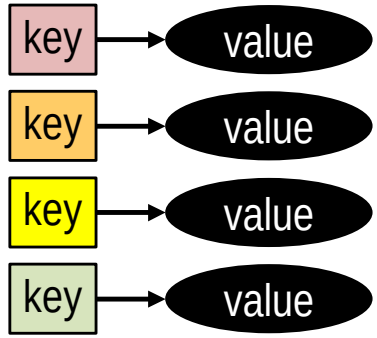
Relational



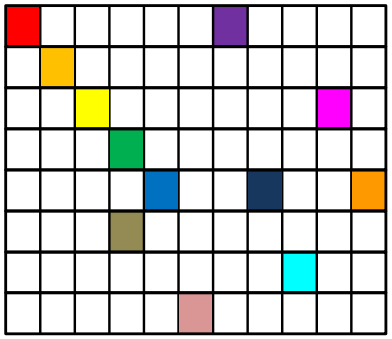
Analytical (OLAP)



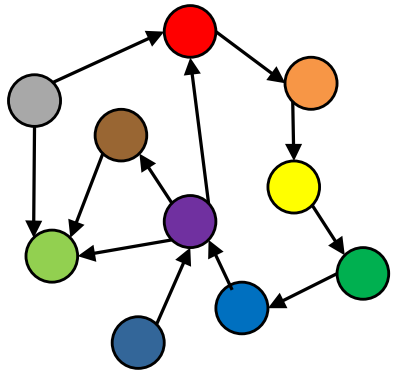
Key-Value



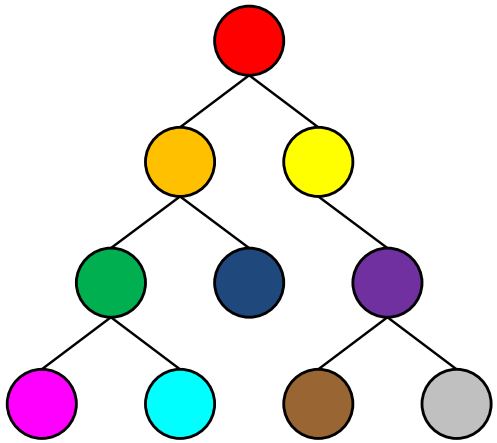
Column-Family



Graph



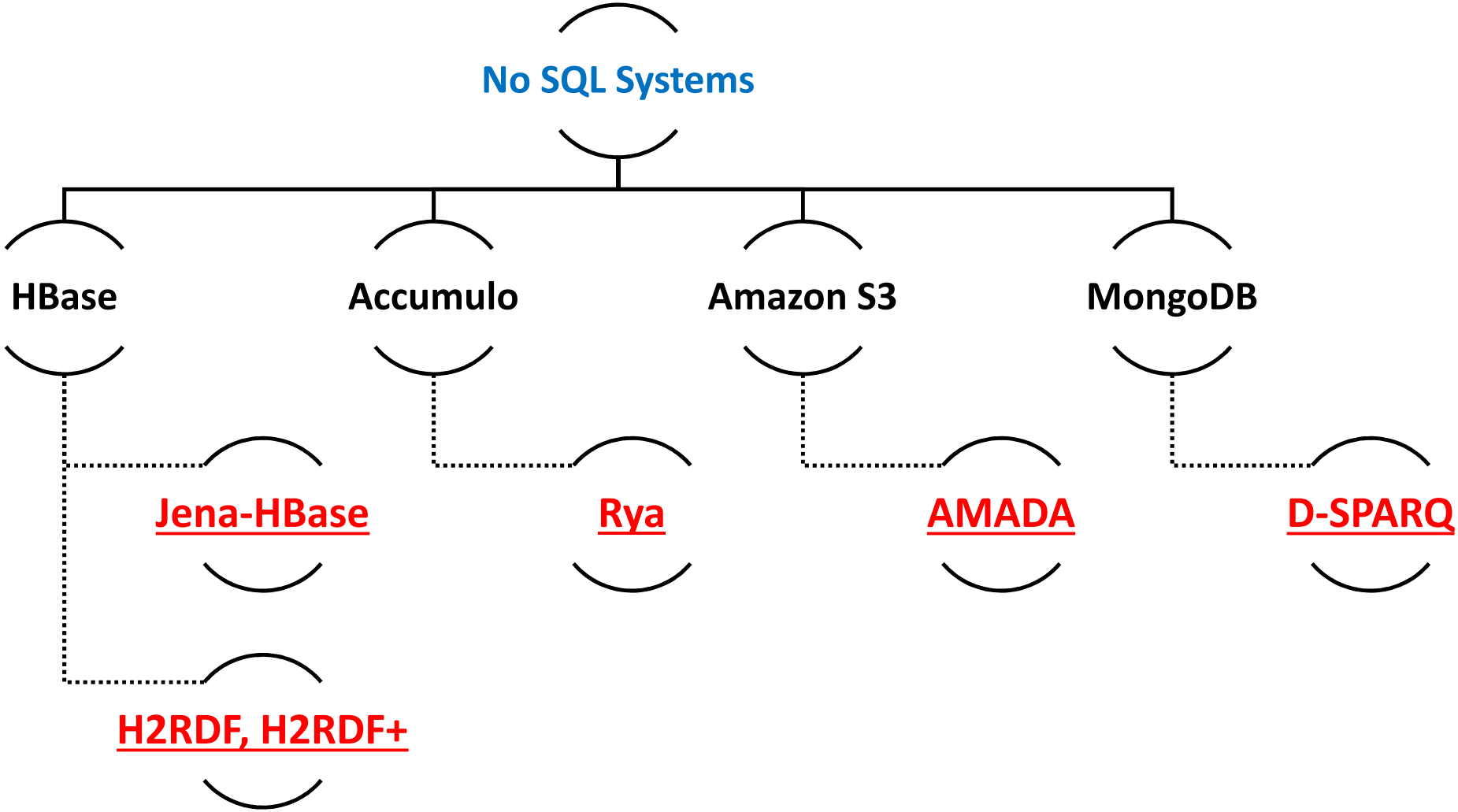
Document



NoSQL Database Systems

http://nosql-database.org/

NoSQL-Based RDF Systems

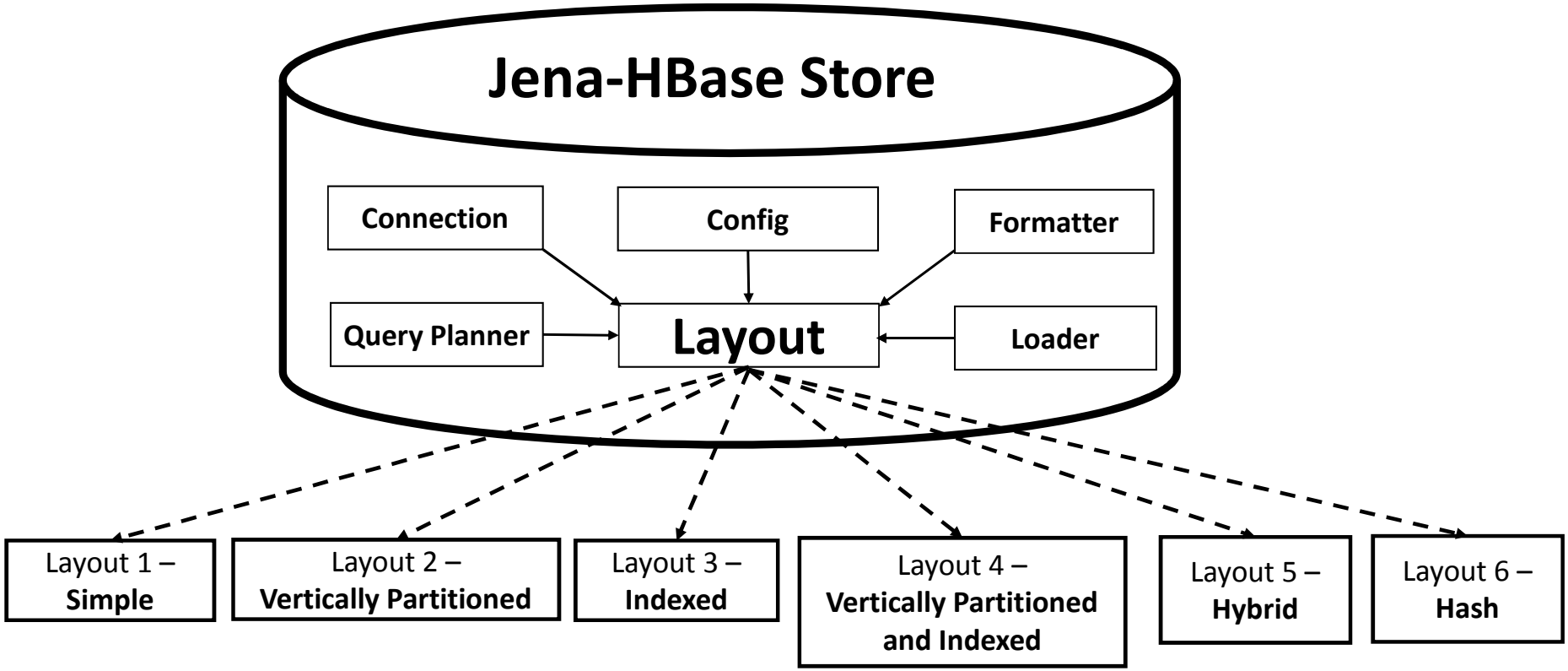


NoSQL-Based RDF Systems: JenaHBase[19]

- It uses HBase, a NoSQL column family store, to provide various custom-built RDF data storage layouts which cover various tradeoffs in terms of query performance and physical storage
- It designs several HBase tables with different schemas to store RDF triples.
  - The *simple* layout uses three tables each indexed by subjects, predicates and objects.
  - For every unique predicate, the *vertically partitioned* layout creates two tables where each of them is indexed by subjects and objects.
  - The *indexed* layout uses six tables representing the six possible combinations of indexing RDF triples.
  - The *hybrid* layout combines both the simple and vertical partitioning.
  - The *hash* layout combines the hybrid layout with hash values for nodes and a separate table maintaining hash-to-node encodings.
- For each of these layouts, JenaHBase processes all operations (e.g., loading triples, deleting triples, querying) on a RDF graph by implicitly converting them into operations on the underlying storage layout.

[19] Khadilkar et al., *Jena-HBase: A distributed, scalable and efficient RDF triple store*. ISWC, 2012.

NoSQL-Based RDF Systems: JenaHBase

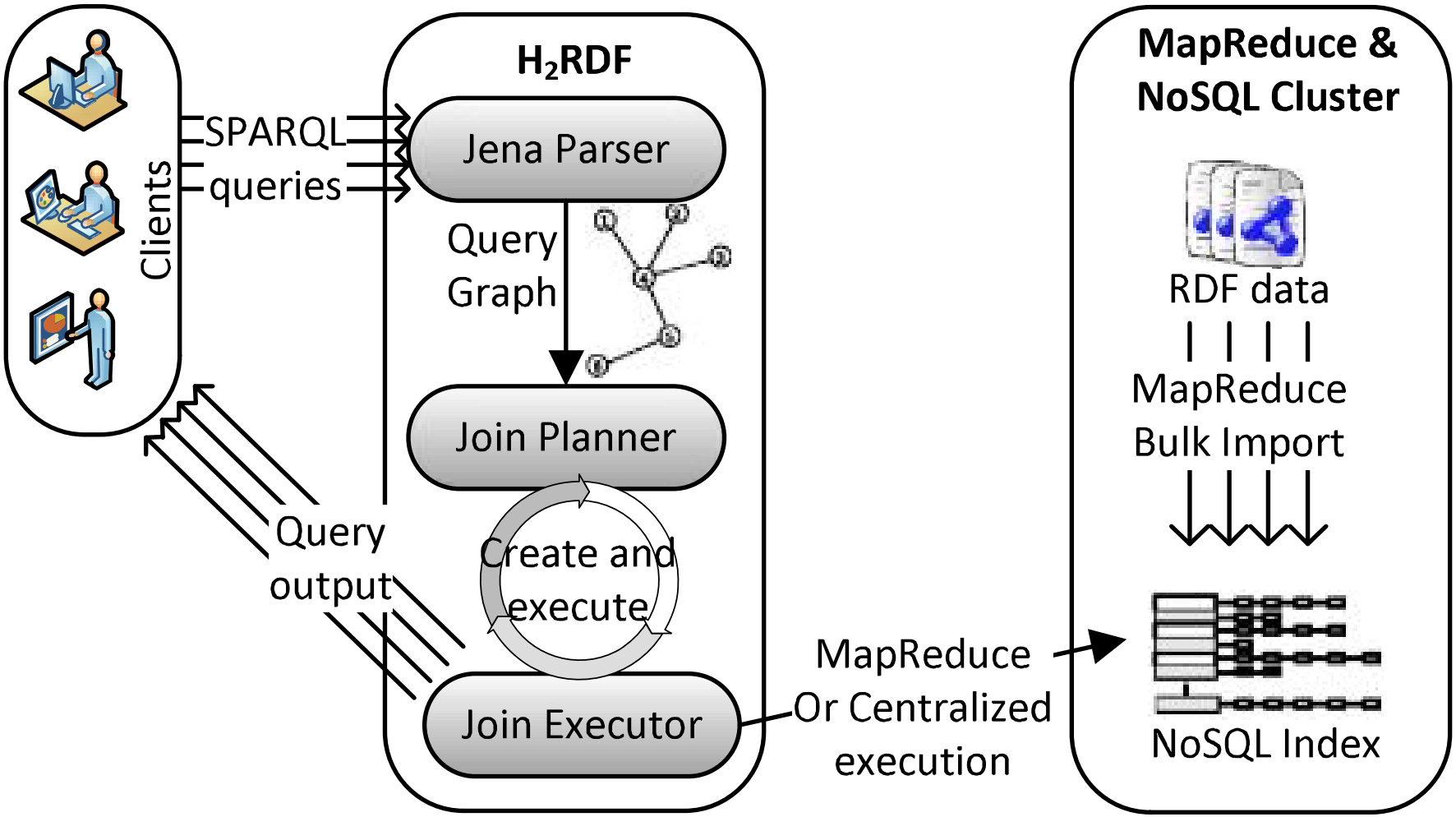


NoSQL-Based RDF Systems: H2RDF+[20]

- A distributed RDF storage system that combines a multiple-indexing scheme over HBase and the Hadoop framework.
- H2RDF+ creates three RDF indices (*spo*, *pos* and *osp*) over the HBase store.
- During the data loading, H2RDF collects all the statistical information which is utilized by the join planner algorithm during query processing.
- During query processing, the Join Planner navigates through the query graph and greedily selects the joins that need to be executed based on the selectivity information and the execution cost of all alternative join operations.
- H2RDF+ uses a join executor module which, for any join operation, chooses the most advantageous join scenario by selecting among centralized and fully distributed execution, via the Hadoop platform.

[20] Papailiou et al. *H2RDF+: an efficient data management system for big RDF graphs*. SIGMOD, 2014.

NoSQL-Based RDF Systems: H2RDF+



NoSQL-Based RDF Systems: CumulusRDF[21]

- An RDF store which provides triple pattern lookups, a linked data server and proxy capabilities, bulk loading, and querying via SPARQL.
- The storage back-end of CumulusRDF is Apache Cassandra.
- The index schema of Cumulus-RDF consists of four indices (SPO, PSO, OSP, CSPO) to support a complete index on triples and lookups on named graphs (contexts).
- The indices are stored in a *flat layout* utilizing the standard key-value model of Cassandra.
- Each index provides a hash based lookup of the row key, a sorted lookup on column keys and values, thus enabling prefix lookups.
- CumulusRDF translates SPARQL queries to index lookups on the distributed Cassandra indices and processes joins and filter operations on a dedicated query node.

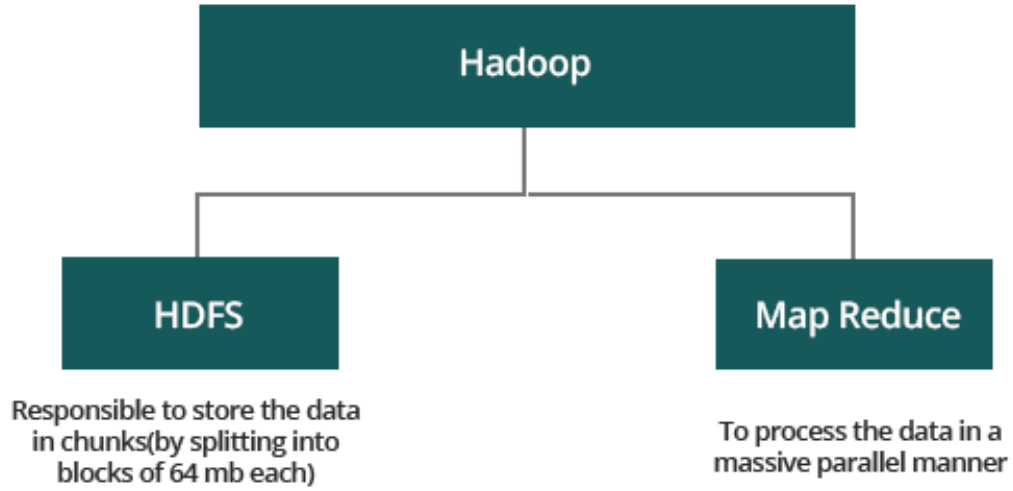
[21] Ladwig and Harth. *CumulusRDF: linked data management on nested key-value stores*. SSWS, 2011

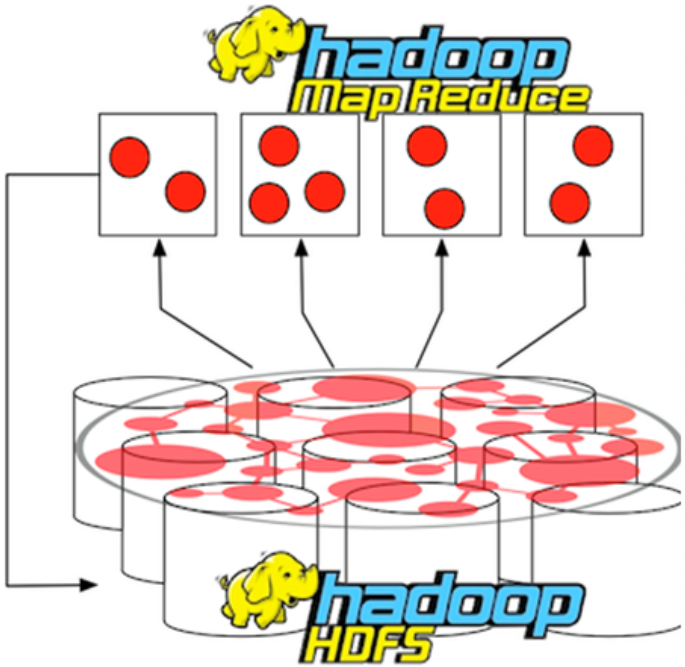
NoSQL-Based RDF Systems: D-SPARQ[22]

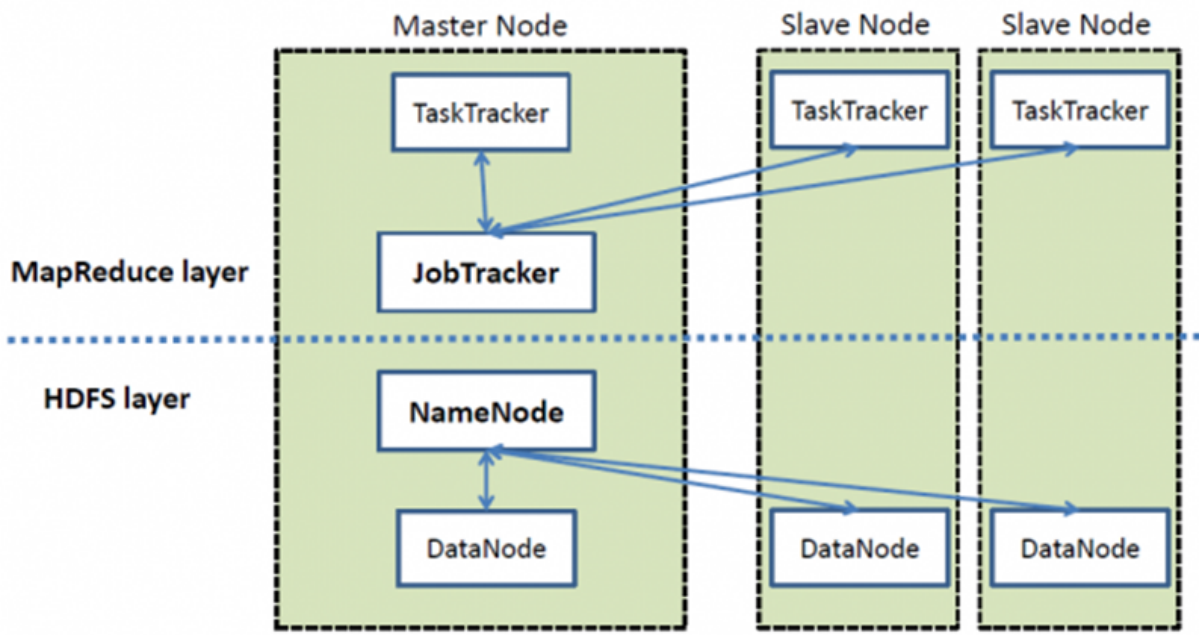
- A distributed RDF query engine on top of MongoDB, a NoSQL document database
- D-SPARQ constructs a graph from the input RDF triples, which is then partitioned using hash partitioning across the machines in the cluster.
- After partitioning, all the triples whose subject matches a vertex are placed in the same partition as the vertex (hash partitioning based on subject).
- A partial data replication is applied where some of the triples are replicated across different partitions to enable the parallelization of query execution.
- Grouping the triples with the same subject enables D-SPARQ to efficiently retrieve triples which satisfy subject-based star patterns in one read call for a single document.
- D-SPARQ also uses indexes involving subject-predicate and predicate-object.
- The selectivity of each triple pattern is used to reduce the query runtime during query execution by reordering the individual triple patterns within a star pattern.

[22] Mutharaju et al. *D-SPARQ: distributed, scalable and efficient RDF query engine*. ISWC, 2013.

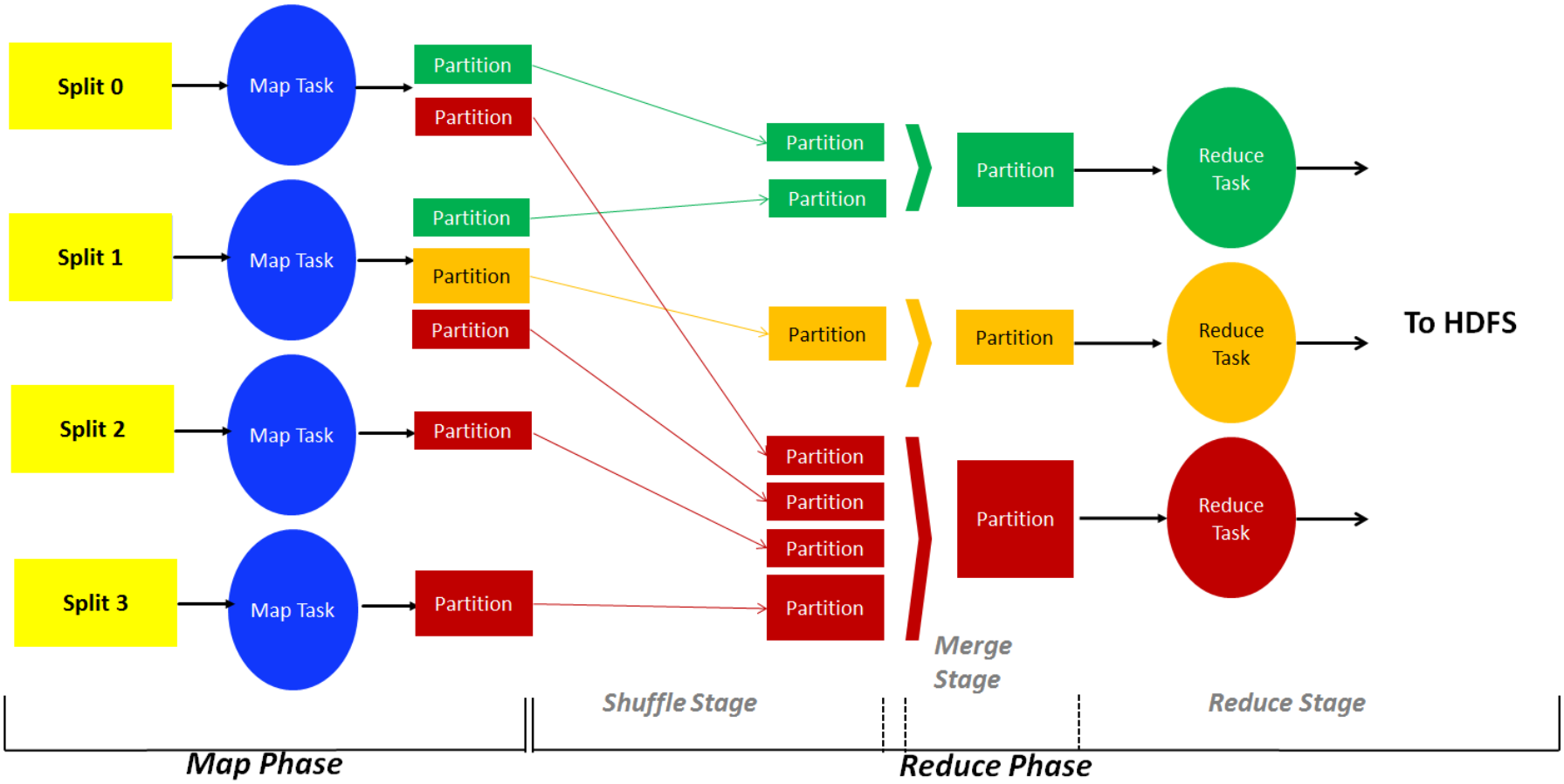
Hadoop


Hadoop
HDFS
Map Reduce
Responsible to store the data in chunks(by splitting into blocks of 64 mb each)
To process the data in a massive parallel manner


hadoop MapReduce
hadoop HDFS


Master Node
Slave Node
Slave Node
TaskTracker
TaskTracker
TaskTracker
MapReduce layer
JobTracker
HDFS layer
NameNode
DataNode
DataNode
DataNode

Hadoop's Execution Architecture

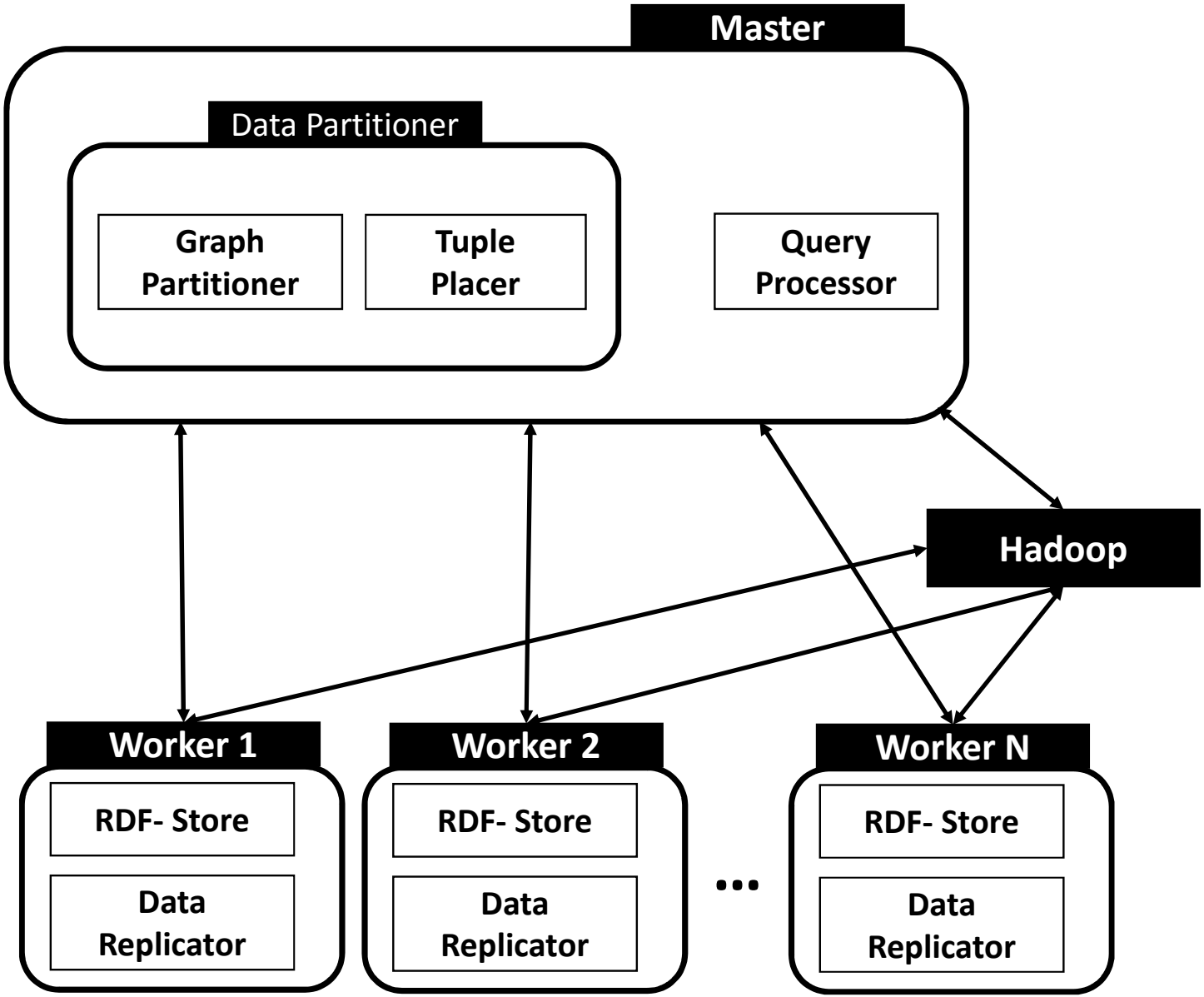


Hadoop-Based RDF Systems: HadoopRDF[23]

- A scale-out architecture which combines the distributed Hadoop framework with a centralized RDF store, RDF-3X for querying RDF databases.
- The data partitioner of HadoopRDF executes a disjoint partitioning of the input RDF graph by vertex using a graph partitioning algorithm that allows triples which are close to each other in the RDF graph to be allocated on the same node.
- HadoopRDF replicates some triples on multiple machines based on specified n-hop guarantees.
- HadoopRDF automatically decomposes the input query into chunks which can be evaluated independently with zero communication across partitions and uses the Hadoop framework to combine the resulting distributed chunks

[23] Huang et al. *Scalable SPARQL querying of large RDF graphs*. PVLDB, 2011

Hadoop-Based RDF Systems: HadoopRDF

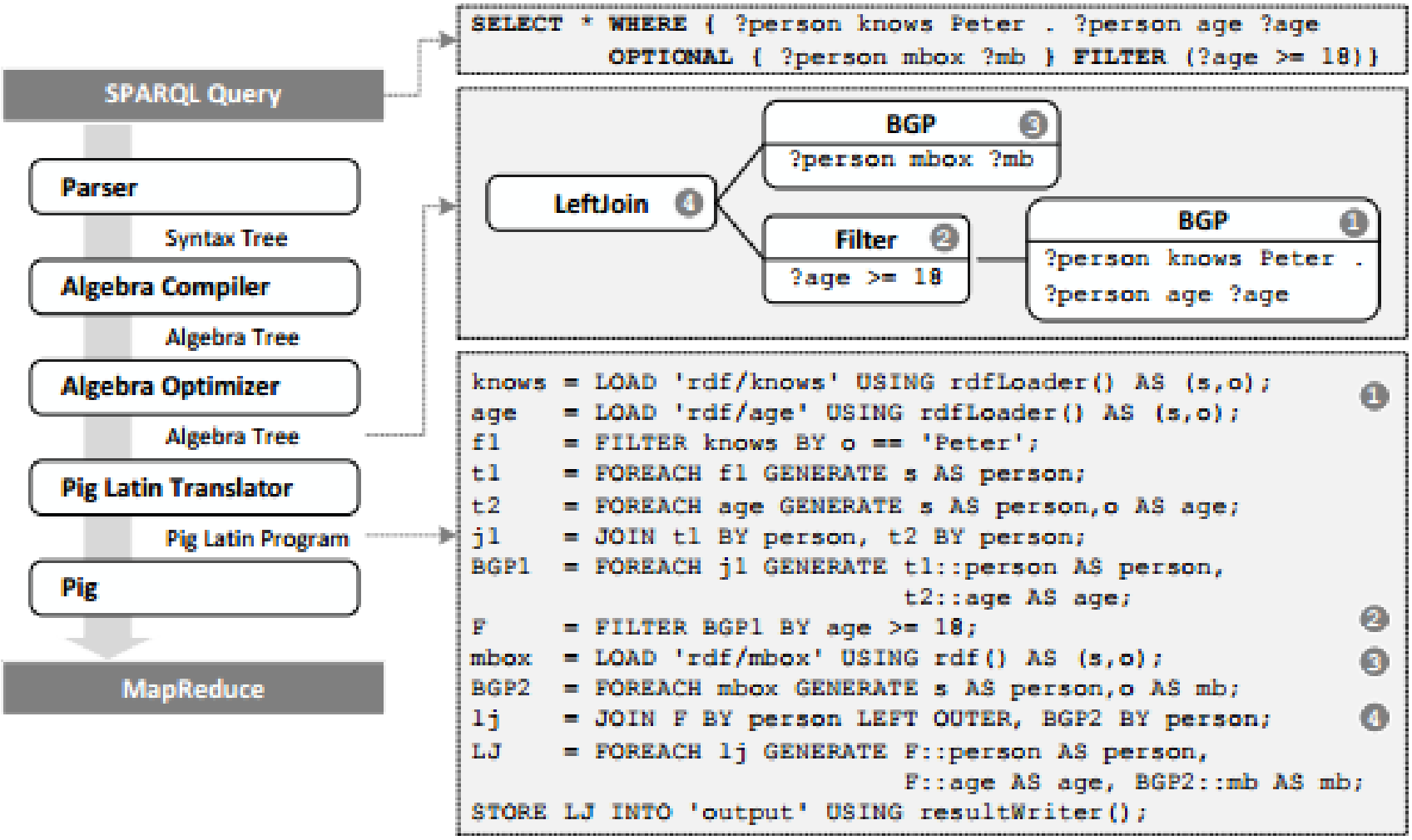


Hadoop-Based RDF Systems: PigSPARQL[24]

- PigSPARQL compiles SPARQL queries into the Pig query language, a data analysis platform over the Hadoop framework.
- Pig uses a fully nested data model and provides relational style operators (e.g., filters and joins).
- A SPARQL query is parsed to generate an abstract syntax tree which is subsequently compiled into a SPARQL algebra tree.
- Using this tree, PigSPARQL applies various optimizations on the algebra level such as the early evaluation of filters and using the selectivity information for reordering the triple patterns.
- PigSPARQL traverses the optimized algebra tree bottom up and generates an equivalent sequence of Pig Latin expressions for every SPARQL algebra operator.
- For query execution, Pig automatically maps the resulting Pig Latin script onto a sequence of Hadoop jobs.

[24] Schatzle et al. *PigSPARQL: a SPARQL query processing baseline for big data*. ISWC, 2013

Hadoop-Based RDF Systems: PigSPARQL



Hadoop-Based RDF Systems: SHAPE[25]

- The SHAPE system is implemented on top of the Hadoop framework with the master server as the coordinator and the set of slave servers as the workers.
- The SHAPE system uses RDF-3X on each slave server and used Hadoop to join the intermediate results generated by subqueries.
- The SHAPE system uses a semantic hash partitioning approach that combines locality-optimized RDF graph partitioning with cost-aware query partitioning for processing queries over big RDF graphs. It maximizes the intra-partition processing capability and minimizes the inter-partition communication cost.
- The SHAPE system classifies the query processing into two types: *intra-partition* processing and *inter-partition* processing.
- The *intra-partition* processing is used for the queries that can be fully executed in parallel on each server by locally searching the subgraphs matching the triple patterns of the query without any inter-partition coordination.
- The *inter-partition* processing is used for the queries that cannot be executed on any partition server, and it needs to be decomposed into a set of subqueries such that each subquery can be evaluated by intra-partition processing.

[25] Lee and Liu. *Scaling queries over big RDF graphs with semantic hash partitioning*. PVLDB, 2013

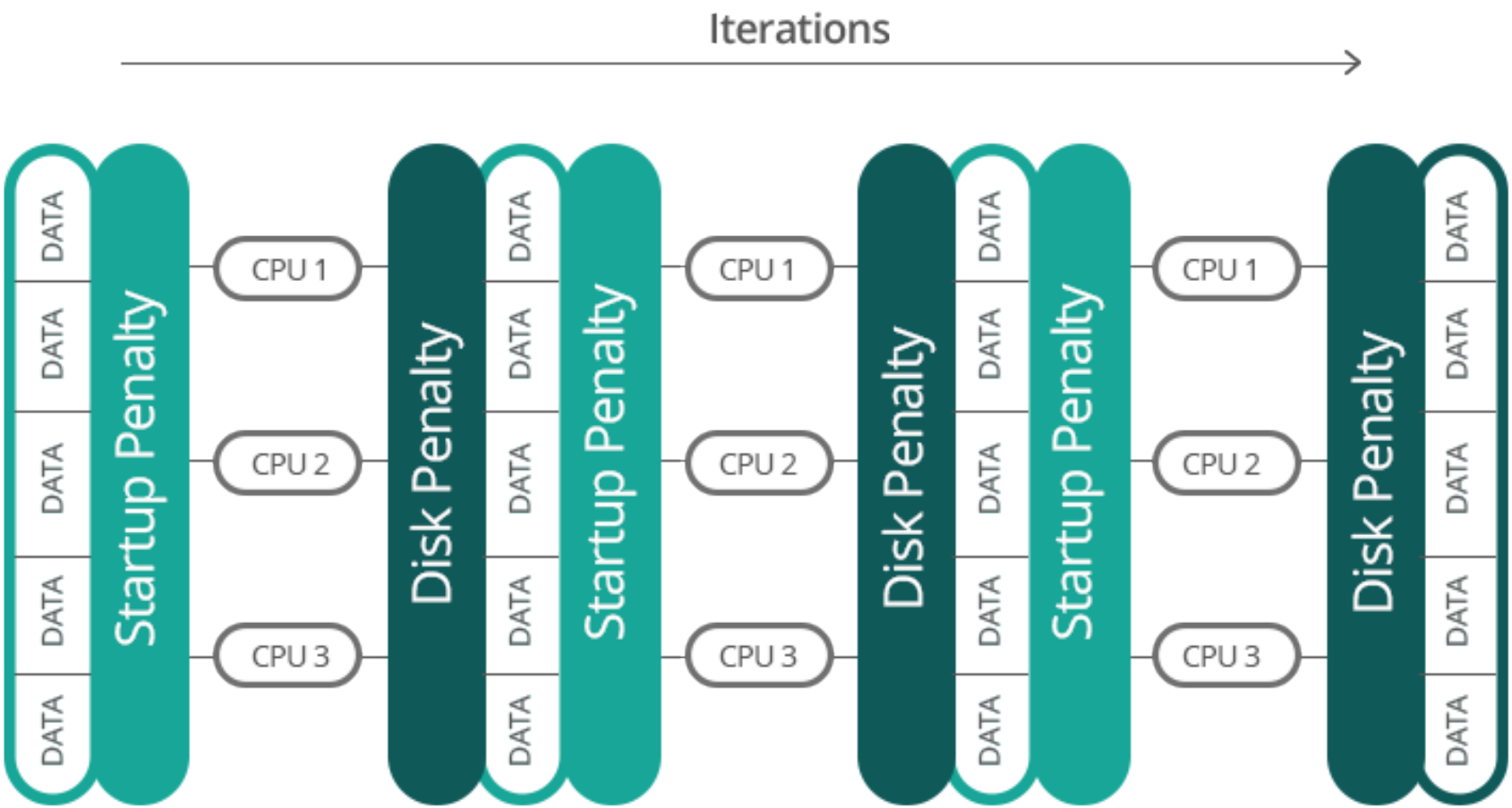
Hadoop-Based RDF Systems: CliqueSquare[26]

- CliqueSquare exploits the built-in data replication mechanism of HDFS, three replicas by default, to partition the RDF dataset in different ways.
- For the *first* replica, it partitions triples based on their subject, property, and object values. For the *second* replica, it stores all subject, property, and object partitions of the same value within the same node. Finally, for the *third* replica, it groups all the subject partitions within a node by the value of the property in their triples.
- For query processing, CliqueSquare relies on a *clique-based algorithm* to produces query plans that minimize the number of MapReduce stages.
- The algorithm is based on the variable graph of a query and its decomposition into clique subgraphs. The algorithm works in an iterative way to identify cliques and to collapse them by evaluating the joins on the common variables of each clique.

[26] Goasdoue et al., *Cliquesquare: Flat plans for massively parallel RDF queries*. ICDE, 2015.

MapReduce for Iterative Operations

MapReduce is **not optimized** for iterative operations



Spark[29]

- Apache Spark is a fast, general engine for large scale data processing on a computing cluster (new engine for Hadoop)[27]
- Developed initially at UC Berkeley, in 2009, in Scala, and is currently supported by Databricks[28]
- One of the most active and fastest growing Apache projects
- Committers from Cloudera, Yahoo, Databricks, UC Berkeley, Intel, Groupon and others.

[27] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. *Spark: Cluster Computing with Working Sets*. HotCloud, 2010.

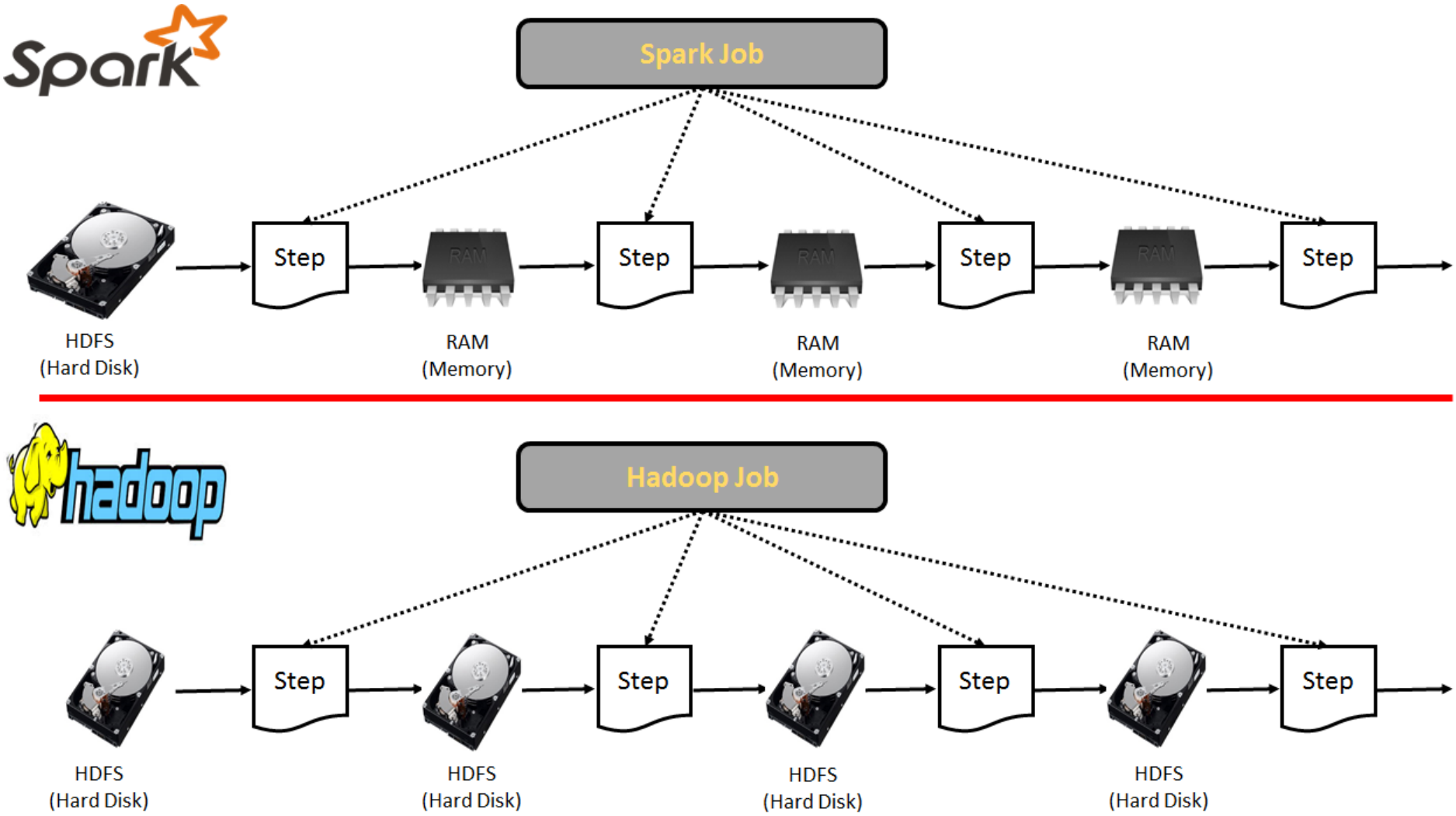
[28] `https://databricks.com/`

[29] `http://spark.apache.org/`

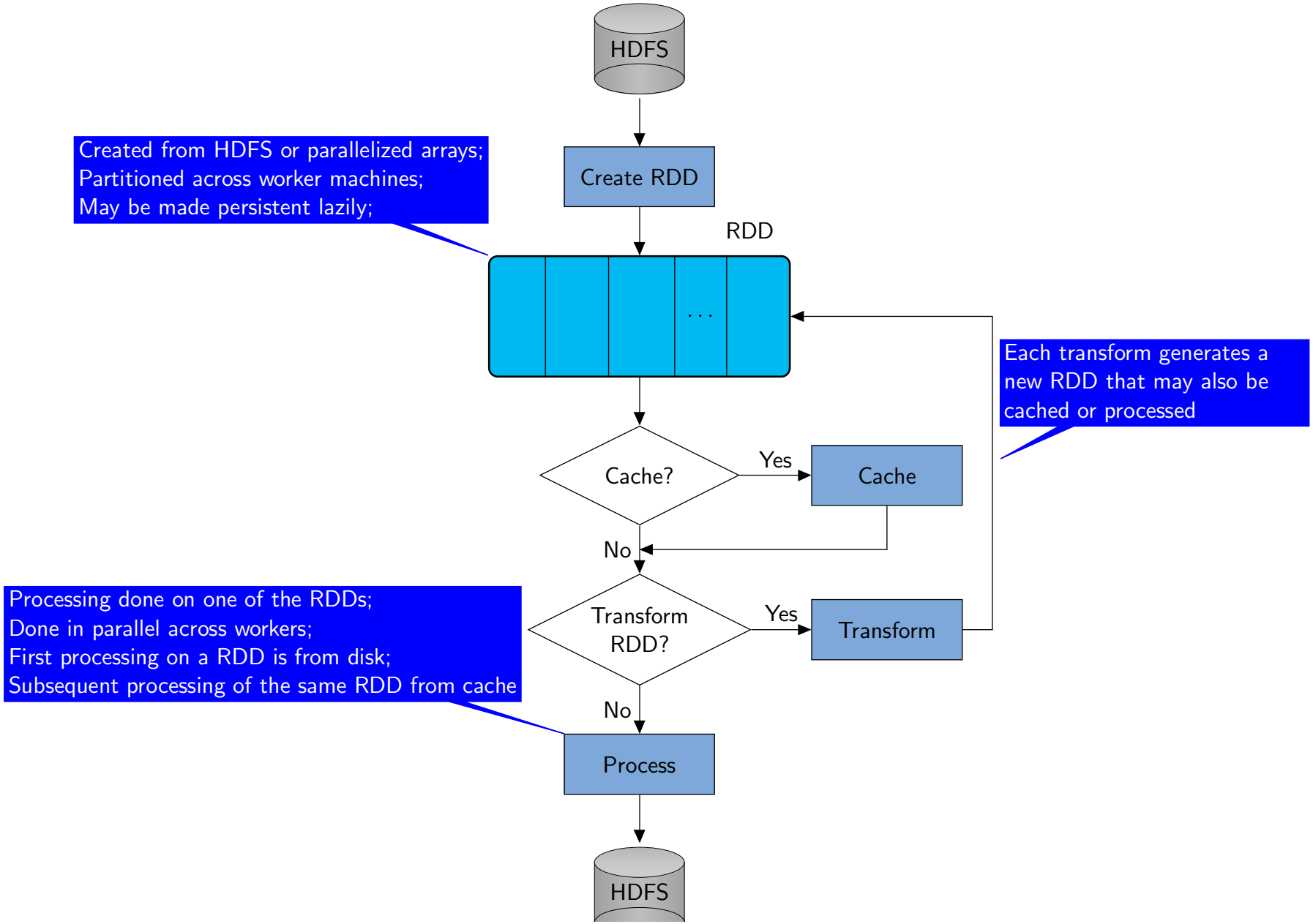
Spark

- **RDD (Resilient Distributed Dataset)**, an in-memory data abstraction, is the fundamental unit of data in Spark
- **Resilient**: if data in memory is lost, it can be recreated
- **Distributed**: stored in memory across the cluster
- **Dataset**: data can come from a file or be created programmatically
- Spark programming consists of performing operations (e.g., Map, Filter) on RDDs

Spark VS Hadoop

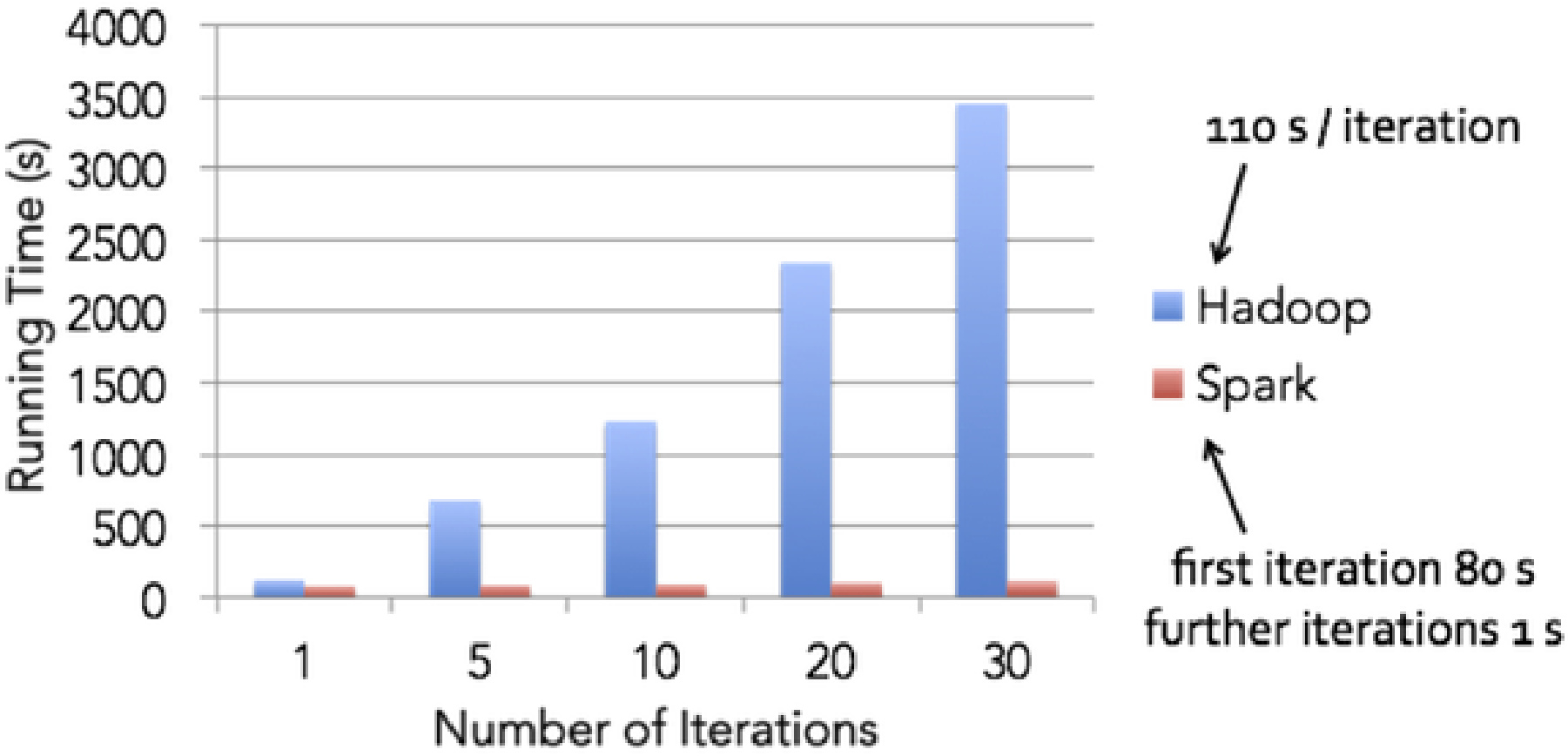


Spark Programming Model



Spark VS Hadoop

- **Spark takes the concepts and performance of MapReduce to the next level**



Spark-Based RDF Systems: S2RDF (SPARQL on Spark for RDF)[30]

- Applies a relational partitioning schema for encoding RDF data called *ExtVP* (the Extended Vertical Partitioning) and uses a semi-join based preprocessing to efficiently minimize query input size by taking into account the possible join correlations between underlying encoding tables of the RDF data (join indices).
- ExtVP precomputes the possible join relations between partitions (i.e. tables).
- S2RDF determines the subsets of a *VP* table VP_{p1} that are guaranteed to find at least one match when joined with another *VP* table VP_{p2} where $p1$ and $p2$ are query predicates.
- The query evaluation of S2RDF is based on SparkSQL, the relational interface of Spark.

[30] Schatzle et al., *S2RDF: RDF querying with SPARQL on spark*. PVLDB, 2016

Spark-Based RDF Systems: S2X (SPARQL on Spark with GraphX)[32]

- RDF engine has been implemented on top of **GraphX**, an abstraction for graph-parallel computation that has been augmented to Spark
- It combines graph-parallel abstractions of GraphX to implement the graph pattern matching constructs of SPARQL.
- **Other Similar approaches**
  - RDF engine on top the **GraphLab** framework, another graph-parallel computation platform
  - **TripleRush**[31] which is based on the graph processing framework **Signal/Collect**, a parallel graph processing system written in Scala.

[31] Stutz et al. *Triplerush: A fast and scalable triple store*. ICSSWK, 2013.

[32] Schatzle al., *S2X: graph-parallel querying of RDF with GraphX*. VLDB Workshop, 2015.

Main Memory-Based RDF Systems: Trinity.RDF[33]

- Trinity.RDF is built on top of Trinity, a distributed main memory-based key/value storage system and a custom communication protocol using the Message Passing Interface (MPI) standard.
- It provides a graph interface on top of the key/value store by partitioning the RDF dataset across the machines using hashing on the graph nodes where each machine maintains a disjoint part of the graph.
- For any SPARQL query, a user submits his query to a proxy. Trinity.RDF performs parallel search on each machine by decomposing the input query into a set of triple patterns and conducting a sequence of graph traversal to produce bindings for each of the triple pattern.
- The proxy generates a query plan and submits the plan to all the machines that maintain the RDF dataset where each machine evaluates its part of the query plan under the coordination of the proxy node.

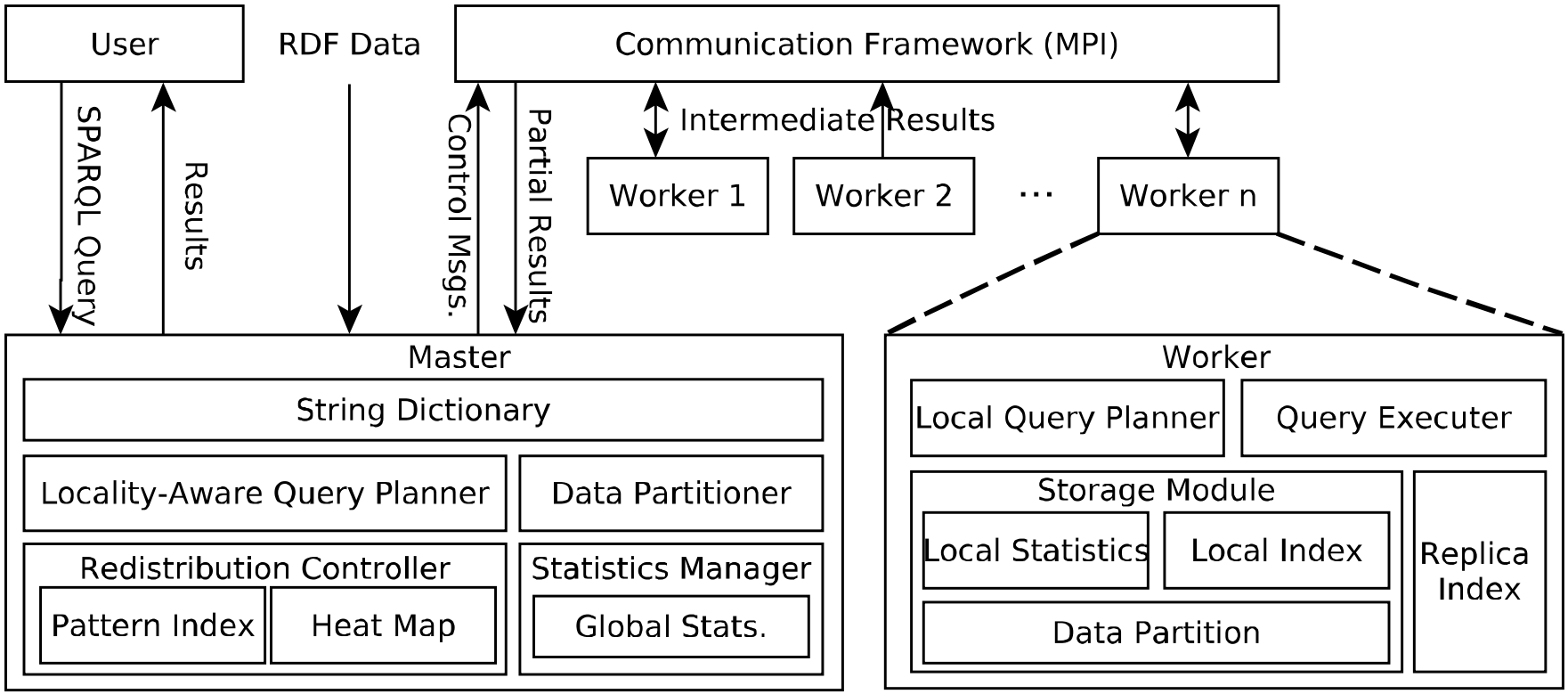
[33] Zeng et al., *A distributed graph engine for web scale RDF data*. PVLDB, 2013.

Main Memory-Based RDF Systems: AdHash [34]

- AdHash initially applies lightweight hash partitioning that distributes triples of the RDF triples by hashing on their subjects.
- It attempts to improve the query execution times by increasing the number of join operations that can be executed in parallel without data communication through utilizing hash-based locality.
- AdHash *continuously monitors* the data access patterns of the executed workload and dynamically adapts to the query workload by incrementally redistributing and replicating the frequently partitions of the graphs.
- The main goal for the adaptive dynamic strategy of AdHash is to effectively minimize or eliminate the data communication cost for future queries.
- *Hot patterns* are redistributed and potentially replicated to allow future workloads which contain them to be evaluated in parallel by all worker nodes without any data transfer
- To efficiently manage the replication process, AdHash specifies a budget constraint and uses an eviction policy for the redistributed patterns.

[34] Harbi et al. *Accelerating SPARQL queries by exploiting hash-based locality and adaptive partitioning*. VLDB J., 2016

Main Memory-Based RDF Systems: AdHash



Other Distributed RDF Systems: Partout[35]

- The Partout engine relies on a *workload-aware* partitioning strategy that allows queries to be executed over a minimum number of machines.
- Partout exploits a representative query workload to collect information about frequently co-ocurring subqueries and for achieving optimized data partitioning and allocation the data to multiple nodes.
- The architecture of Partout consists of a *coordinator* node and a cluster of *hosts* that store the actual data. The coordinator node is responsible for distributing the RDF data among the host nodes, designing an efficient distributed query plan for a SPARQL query, and initiating query evaluation. Each of the host nodes runs a triple store, **RDF-3X**.
- Partout's global query optimization algorithm avoids the need for a two-step approach by starting with a plan optimized with respect to the selectivities of the query predicates and then applying heuristics to obtain an efficient plan for the distributed setup. Each host relies on the RDF-3X optimizer for optimizing its local query plan.

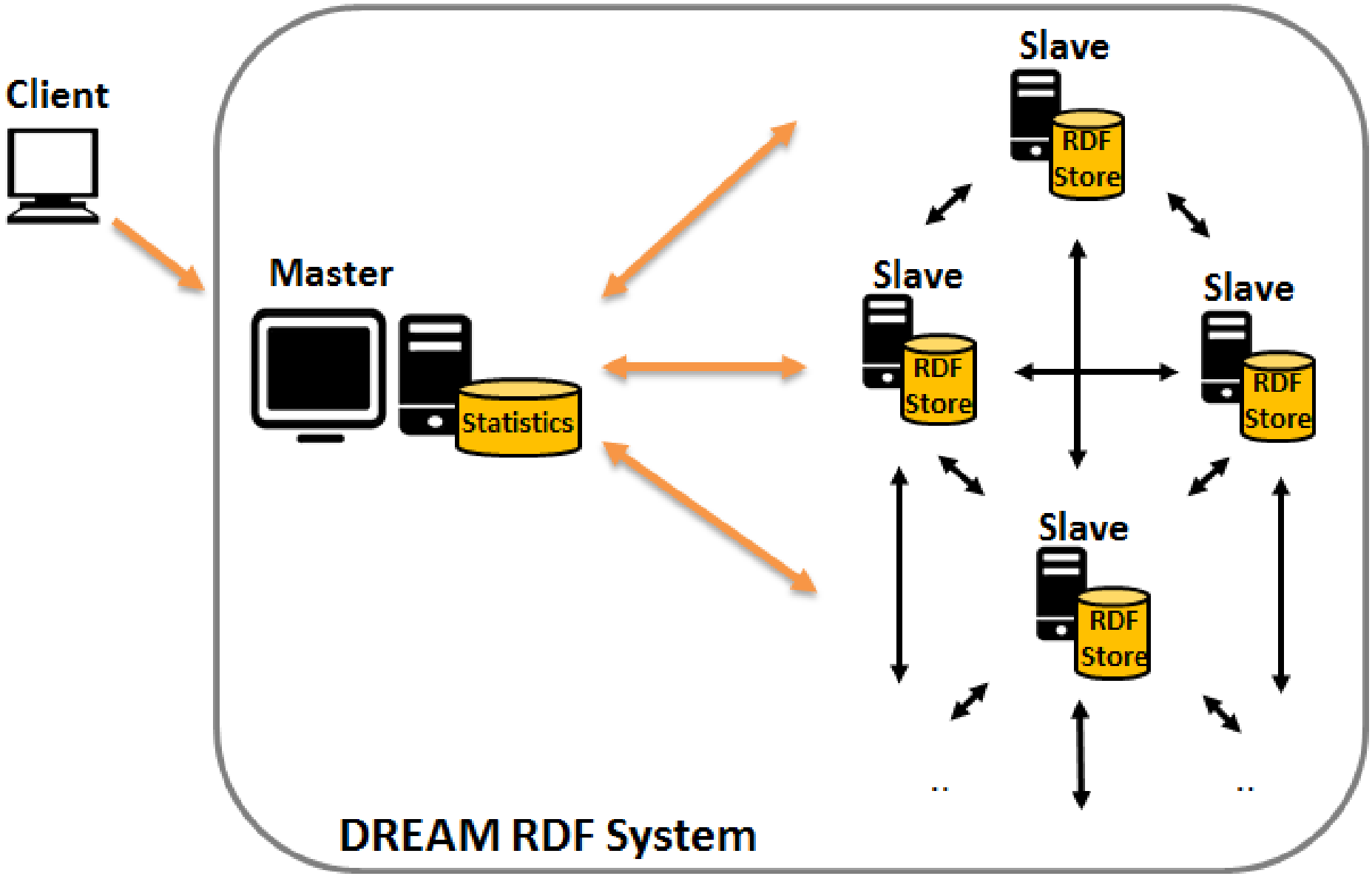
[35] Galarraga et al., *Partout: a distributed engine for efficient RDF processing*. WWW, 2014.

Other Distributed RDF Systems: DREAM[36]

- The DREAM system has been designed as a *D*istributed *R*DF *E*ngine with *A*daptive Query Planner and *M*inimal Communication.
- It is designed with the aim of avoiding partitioning RDF graphs. DREAM stores a dataset intact at each cluster machine and it partitions SPARQL queries rather than partitioning RDF datasets.
- The query planner of DREAM transforms Q into a graph, G, decomposes G into sets of sub-graphs, each with a basic two-level tree structure, and maps each set to a separate machine. Afterwards, all machines process their sub-queries in parallel and coordinate with each other to return the final result.
- Each of the host nodes uses **RDF-3X** to evaluate the sub-queries.
- No intermediate data is shuffled. Only minimal control messages and meta-data are exchanged.
- DREAM is able to select different numbers of machines for different query types, hence, rendering it adaptive.

[36] Hammoud et al., *DREAM: distributed RDF engine with adaptive query planner and minimal communication*. PVLDB, 2015.

Main Memory-Based RDF Systems: DREAM

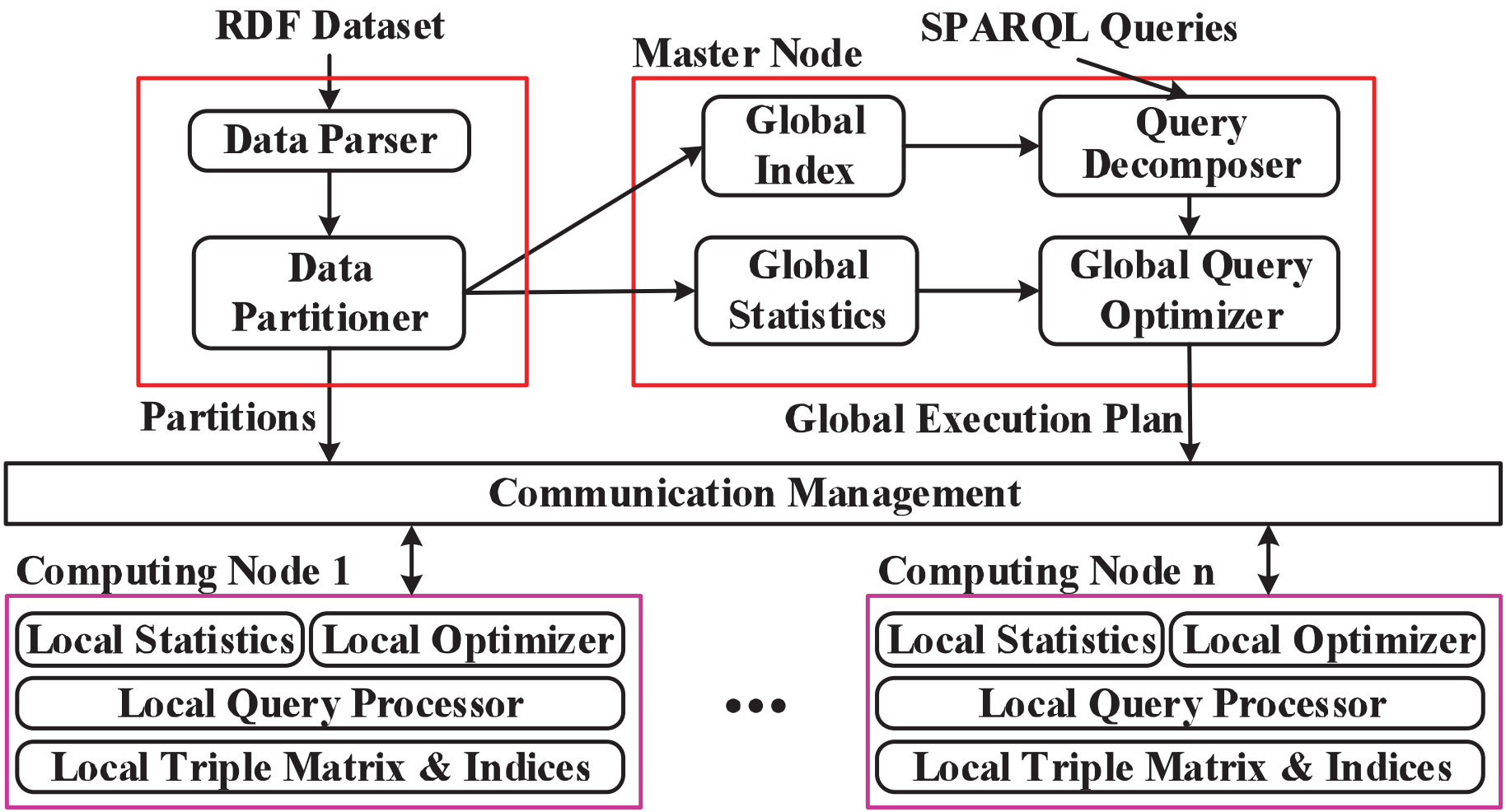


Other Distributed RDF Systems: Semstore[37]

- SemStore system adopts a partitioning mechanism, Rooted Sub-Graph (RSG), that is designed to effectively localize the processing of RDF queries.
- After partitioning the RDF graph, the data partitioner assigns each partition to one of the underlying computing nodes.
- The SemStore partitioner uses a k-means partitioning algorithm for assigning the highly correlated RSGs into the same node.
- Each computing node builds local data indices and statistics for its assigned subgraph and utilizes this information during local join processing and optimizations.
- The data partitioner builds a global bitmap index over the vertices of the RDF graph and collects the global statistics.
- Each computing node uses a centralized RDF processor, **TripleBit**, for local query evaluation.

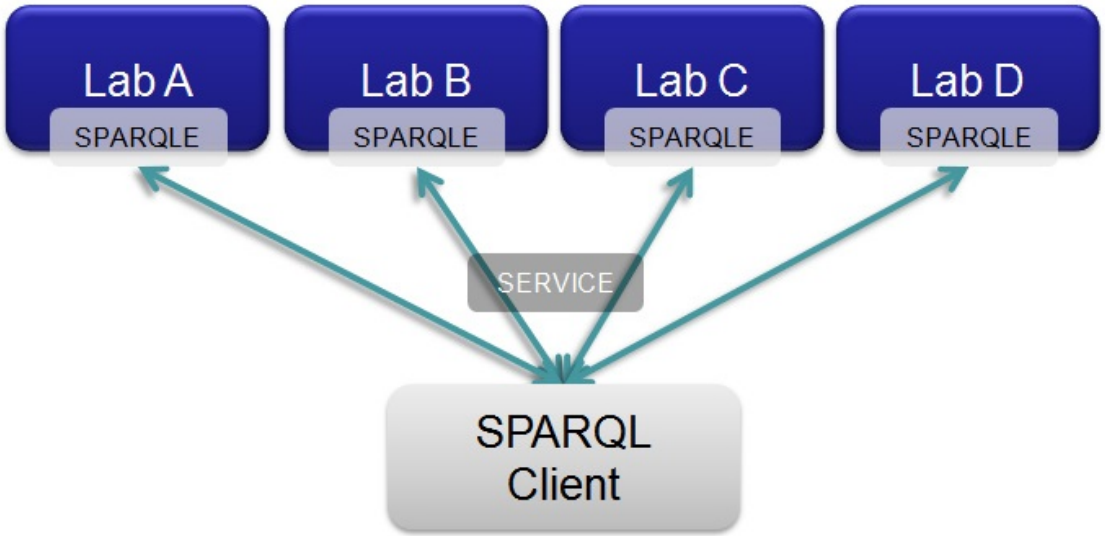
[37] Wu et al. *Semstore: A semantic-preserving distributed rdf triple store*. CIKM, 2014.

Main Memory-Based RDF Systems: Semstore



Federated RDF Query Processing

- The proliferation of RDF datasets created a significant need for answering RDF queries over multiple SPARQL endpoints. Such queries, referred to as **RDF federated queries**.
- Answering such type of queries requires performing on-the-fly data integration and complex graph operation over heterogeneous distributed RDF datasets.
- Factors like the number of sources selected, total number of SPARQL ASK requests used, and source selection time have significant impact on the query execution time.
- To minimize the number of sub-queries most of the systems group the tripe patterns that can be entirely executed on one endpoint.

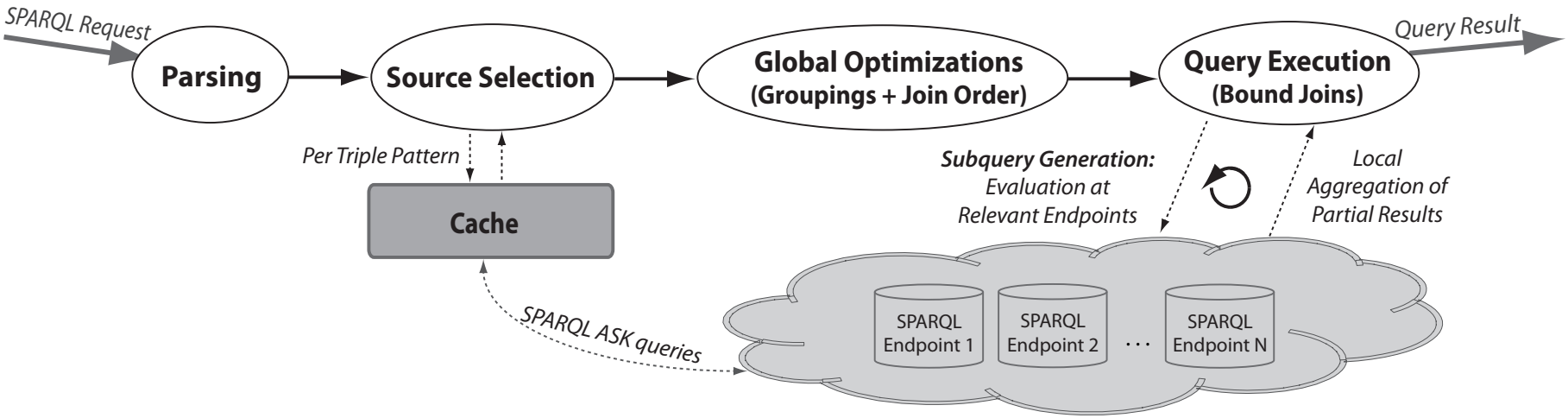


Federated RDF Query Processing: FedX[38]

- To select relevant source for a triple pattern FedX sends a SPARQL ASK query to all known endpoints.
- The join order optimization is based on the variable counting technique which estimates the cost of execution by counting free variables, that ate not bound trough previous joins.
- FedX groups triples that have the same set of sources on which they can be executed. This allows to sent them to the endpoints as a conjunctive query and minimize the cost of local joins as well as the network traffic.
- The system implements joins in a block nested fashion. The advantage of block nested loop join is that the number of remote requests can be reduced by the factor determined the the size of a block.

[38] Schwarte et al. *Fedx: Optimization techniques for federated query processing on linked data*. ISWC, 2011.

Federated RDF Query Processing: FedX

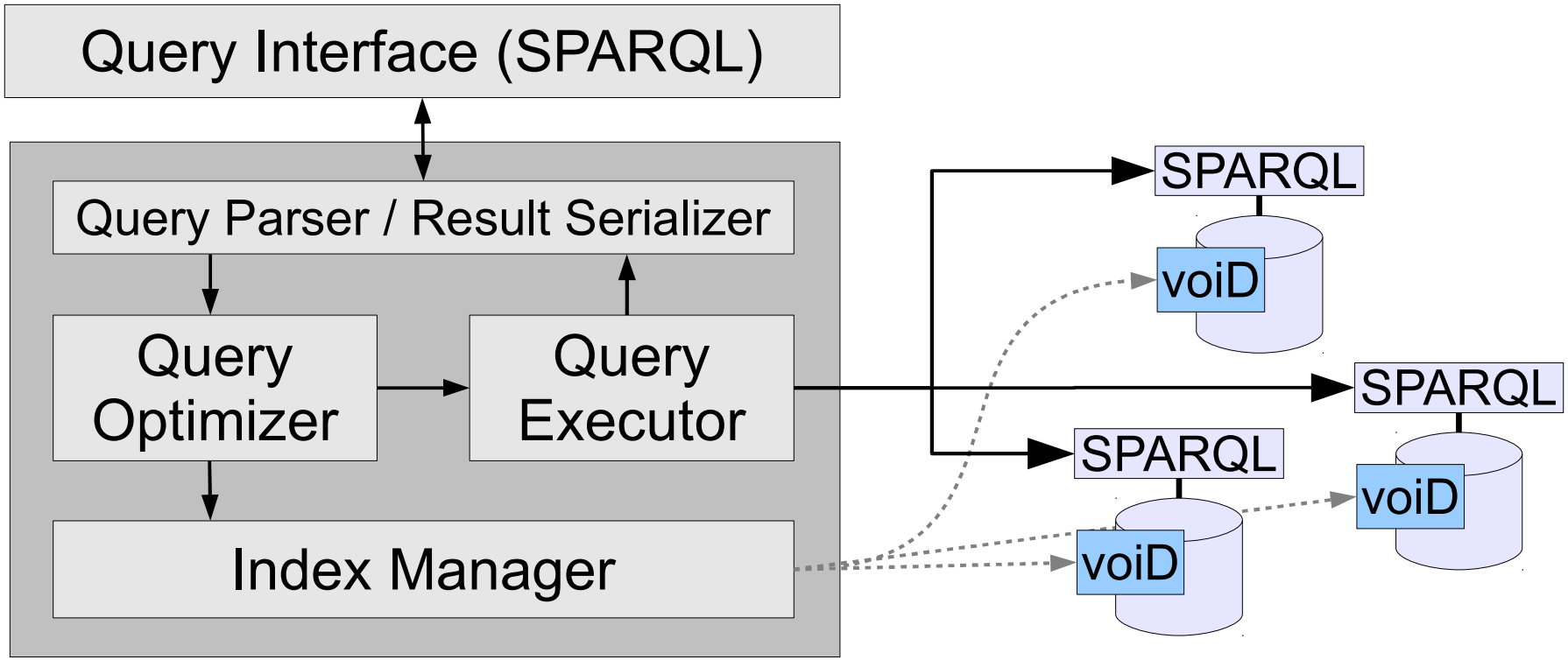


Federated RDF Query Processing: SPLENDID[39]

- SPLENDID uses statistics which is obtained from VOID (Vocabulary of Interlinked Datasets) descriptions to optimize the execution of federated queries.
- The Index Manager maintains the local copy of collected and aggregated statistics from remote SPARQL endpoints.
- The Query Optimizer transforms the query into a syntax tree, select a data source to federate the execution and optimize the order of joins.
- To select a data source for a triple pastern SPLENDID uses two inverted indexes for bound predicates and types, with priority for types.
- To join the sub-results SPLENDID implements two strategies:
  - For small result sets, the tuples are requested in parallel and a hash join is performed locally.
  - For large result sets and high selectivity of a join variable. one sub-query is executed and the the join variable in the second one is repeatedly replaced with the results of the first one.

[39] Gorlitz and Staab. *Splendid: Sparql endpoint federation exploiting void descriptions*. 2nd International Conference on Consuming Linked Data, 2011.

Federated RDF Query Processing: SPLENDID



Part IV

Open Challenges

Benchmarking

- The Semantic Web community has developed several frameworks to evaluate the performance and scalability of RDF Systems.
  - The Lehigh University Benchmark (LUBM)[40]
  - The SP2B benchmark[41]
  - The Berlin SPARQL Benchmark (BSBM)[42]
  - The DBpedia SPARQL Benchmark (DBPSB)
  - The Semantic Publishing Benchmark v2.0 (SPB)[43]
  - The Social Network Intelligence BenchMark[44]
  - The WatDiv Benchmark[45]

---

[40] http://swat.cse.lehigh.edu/projects/lubm/

[41] http://dbis.informatik.uni-freiburg.de/index.php?project=SP2B

[42] http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/

[43] http://ldbcouncil.org/developer/spb

[44] http://ldbcouncil.org/developer/snb

[45] http://dsg.uwaterloo.ca/watdiv/

Benchmarking

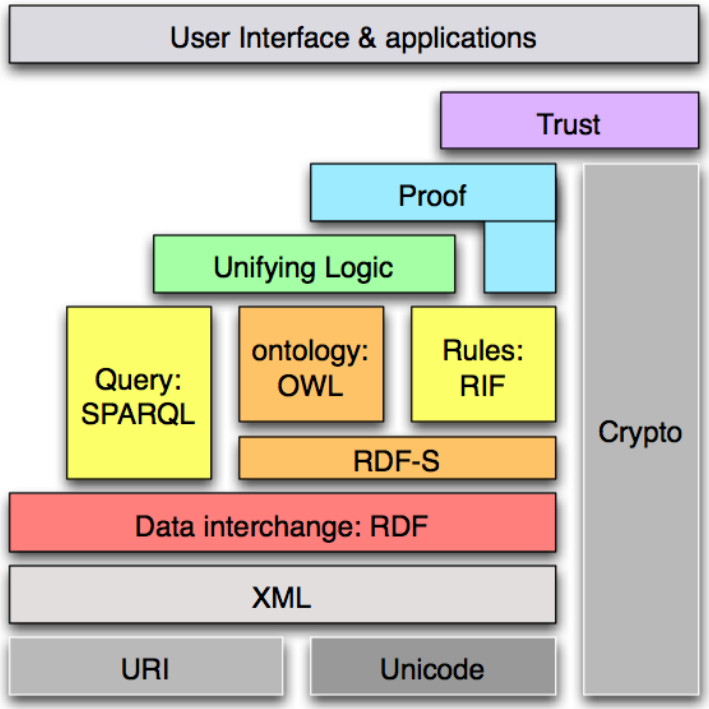
- Many RDF management systems present their own evaluation and comparison with related systems; however, such evaluations are inherently biased and difficult to generalize.
- Several benchmarking studies have been conducted to provide an evaluation of a subset of the existing RDF data management systems. The biggest data collection was used in the report published by within BSBM benchmark project (10M-150B triples). The largest tests on NoSQL systems were performed was on up to 16 Amazon EC2 units.
- In general, the set of selected systems benchmarked in each study has been quite limited in comparison to the available spectrum of systems.
- The set of selected systems and the benchmarking setup of the various studies varied significantly, such that they do not allow to build neither a comparable nor a comprehensive picture of the state of the art in this domain.
- More comprehensive benchmarking efforts are singularly required in order to allow users to clearly understand the strengths and weaknesses of the various systems and design decisions.

Efficient and Scalable Processing of Complex SPARQL Features

- SPARQL is an expressive language that supports different RDF querying features such as such as the OPTIONAL operation (i.e, a triple pattern can be optionally matched), filter expressions and string functions with regular expressions.
- The majority of scalable SPARQL querying techniques have been designed for the evaluation of conjunctive BGP queries on RDF databases.
- Designing scalable and efficient queering techniques/systems for the complex features of SPARQL 1.1 requires more attention from the research community.
- Support for other SPARQL 1.1. features
  - SPARQL 1.1 Update
  - SPARQL 1.1 Graph Store Protocol
  - SPARQL 1.1 Service Description
  - SPARQL 1.1 Federated Query
  - SPARQL 1.1 Query Results JSON, XML, CSV and TSV Format
  - ...

Distributed and Scalable RDF Reasoning

- Increasing amounts of RDF data is getting generated and consumed. This type of massive structured RDF data along with its model and provenance information is often referred to as a **knowledge graph**.
- An important operation that can be performed over RDF triples is **reasoning**.
- Existing reasoners cannot handle such large knowledge bases.
- There is a crucial need to exploit modern big data processing systems on building efficient and scalable RDF reasoning solution.



Part V

Conclusions

Conclusions

- Scalable querying and reasoning of big RDF datasets involve various unique challenges.
- In the last few years, several distributed RDF data processing systems have been introduced with various design decisions.
- Open challenges include benchmarking, support complex SPARQL features and building salable reasoners.

Conclusions

Our Book on Linked Data: Storage, Querying and Reasoning

Sherif Sakr, Marcin Wylot, Raghava Mutharaju, Danh Le Phuoc and Irini Fundulaki. "*Linked Data: Storage, Querying and Reasoning*", Springer, 2018

The End


Thank You
Mahalo
Kiitos
Tack
Toda
Grazie
Thanks
Obrigado
Takk
Merci
Gracias