Eventos

OntoQuad: Native High-Speed RDF DBMS for Semantic Web

Alexander Potocki[1], Anton Polukhin[1], Grigory Drobyazko[2], Daniel Hladky[2], Victor Klintsov[2], and Jörg Unbehauen[3]

[1] Eventos, Moscow, Russia
{alexander.potocki, anton.polukhin}@my-eventos.com

[2] National Research University - Higher School of Economics (NRU HSE), Moscow, Russia
{gdrobyazko, vklintsov, daniel.hladky}@hse.ru

[3] Universität Leipzig, Institut für Informatik, Leipzig, Germany
unbehauen@informatik.uni-leipzig.de

OntoQuad General Information

Eventos

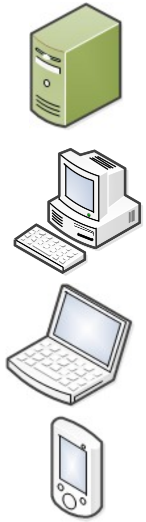
Supported Standards and Platforms

OntoQuad

- Is developed with the latest **C++** Standard (**C++11**) from zero
- is compliant with the latest standards of the W3C (e.g. **RDF**, **SPARQL 1.1**)
- supports Java (**Jena**) **API**
- works in **transactional** mode

OntoQuad is cross-platform and can be deployed on different devices:

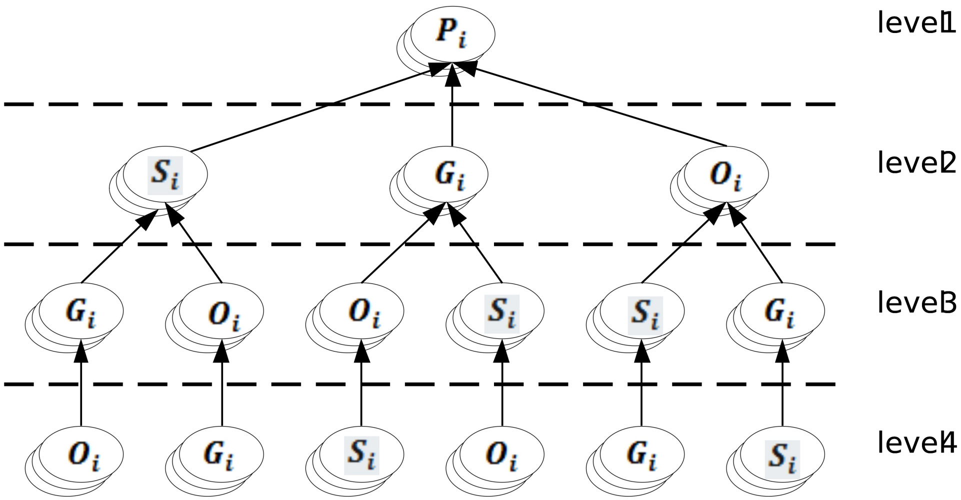
- MS Windows x64 (developed on Windows 7)
- Unix/Linux x64 (tested on Linux CentOS 6.3)
- Mobile Android (Samsung Galaxy Note II, Google Nexus 7 etc.)
- Raspberry Pi Model B rev 2
- iOS & OS X – is coming soon



Information Architecture

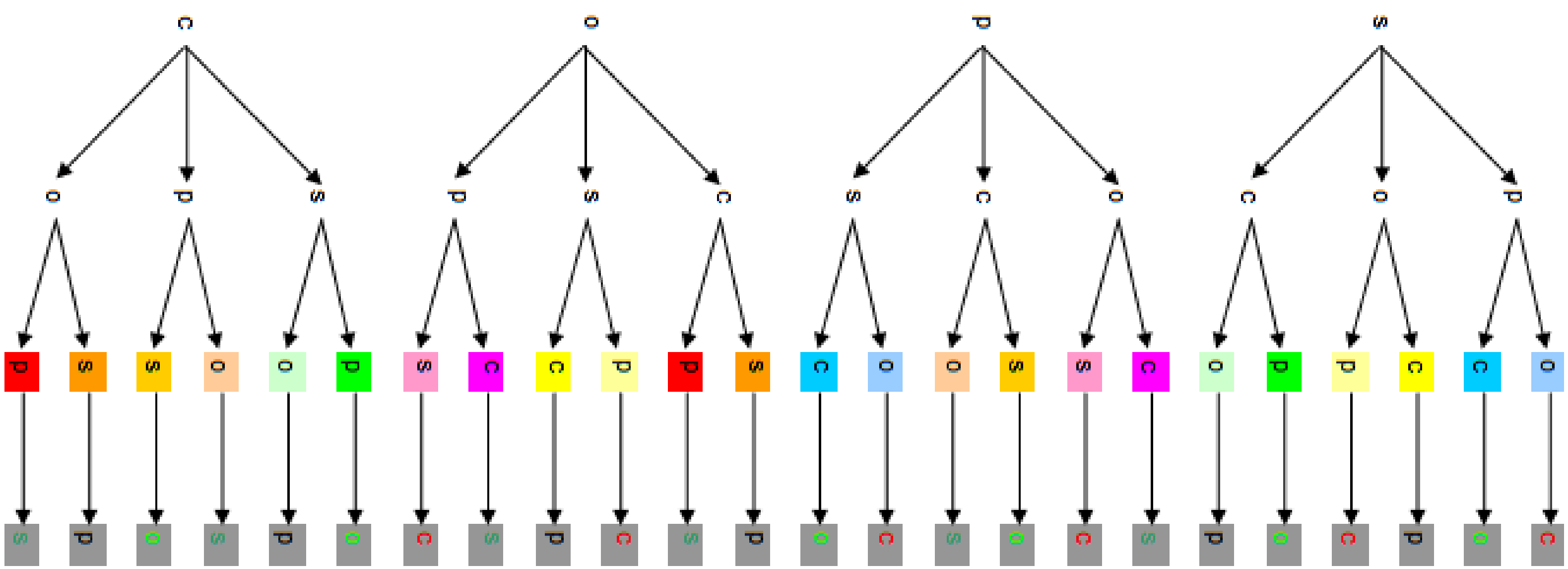
Eventos

Vector Model



HexaStore & the Vector Model

In our work we elaborate on the vector representation of triples proposed for the Hexastore, by expanding it onto quadruple representation



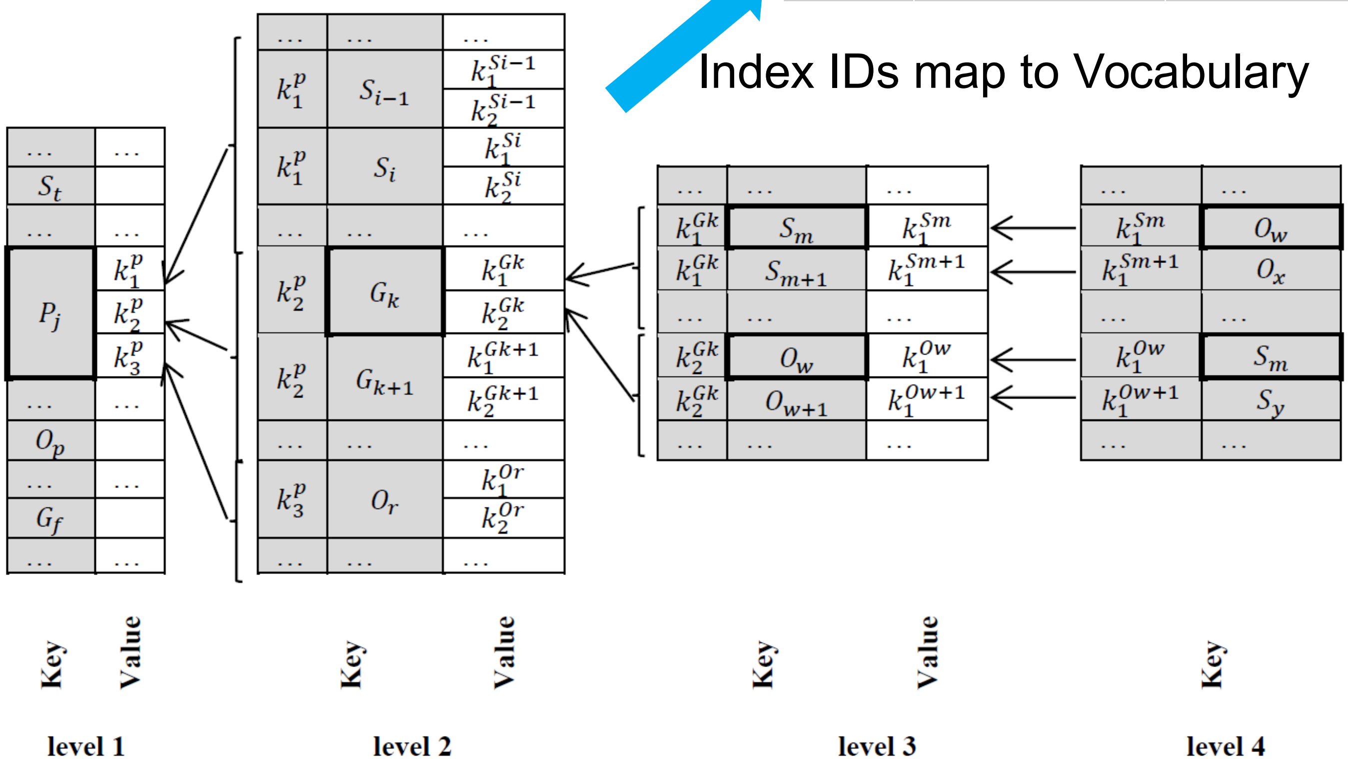
Data Storage Components

Database Structure:

- **Key-value indexes** (Index-24)
- **Vocabulary**

ID	Type	Value
S_t	*IRI*	*<http://purl.org/dc/...>*
...		
P_j	*IRI*	*<http://example.org/...>*
...		
O_p	*xsd:dateTime*	*"2005-02-28T00:00:00Z"*
...		
G_f	*IRI*	*<http://mygraph.com>*

Index IDs map to Vocabulary



Persistence Strategy

•The DBMS creates several files for storing data. The file combines both a structure for storing data and an **Key-Value index** implemented as B-trees (or B*-trees) because it ensures the support of prefix range lookups.

•The DBMS keeps all unique values in a separate **Vocabulary**, and **Key-Value indexes** contain references (fixed-length identifiers) to the **Vocabulary** items.

•**Vocabulary** is a full lexicon of URI’s and literals that are “known” to the base which associates the values of S, P, O and G with their vocabulary ID’s that are unique within a DB instance.

Index-type configuration parameter can take four values:

•**polymorphic2** provides two indexes configuration. Supports PSO, POS indexes.

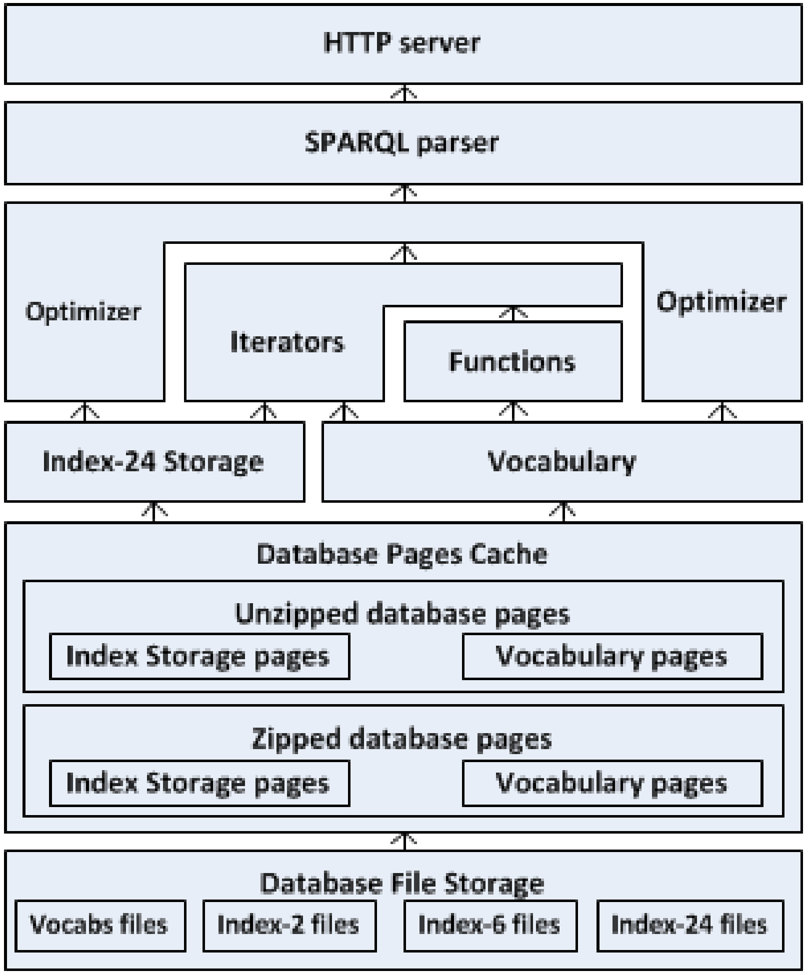
•**polymorphic6 | polymorphic6monolith** provides six indexes configuration. Supports PSOG, PSGO, POSG, POGS, PGOS, PGSO indexes.

•**polymorphic24** provides twenty four indexes configuration. Supports all permutations (24) of four elements SPOG.

Components Architecture

Eventos

Component Scheme



Brief components description is on next page

Main Components

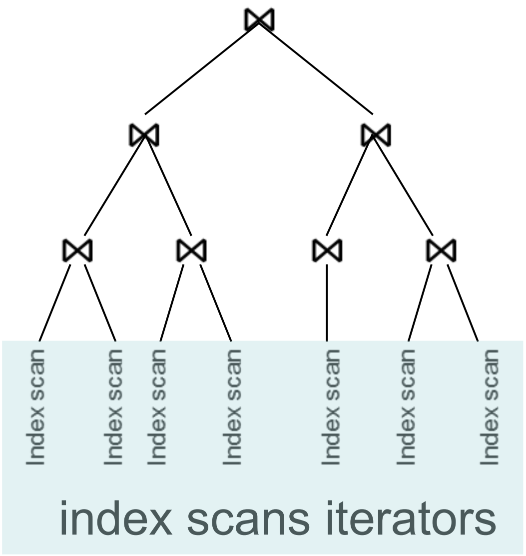
- The built-in **HTTP Server** is a SPARQL 1.1 endpoint;
- The **SPARQL Parser** does syntactic analysis of queries and generates of the initial QEP tree;
- The **Optimizer** transforms the initial QEP into a new equivalent QEP with more optimal performance time and resources;
- The **Iterators** implement SPARQL algebra operators of QEP;
- The **Functions** are either functions of the SPARQL language or custom functions;
- The **Vocabulary** is a comprehensive lexicon of URI's and literals downloaded into the database;
- The **Index-24** implements different PSOG indexes;
- The **Database Page Cache** (zipped and unzipped) keeps last used Index-24 and **Vocabulary** pages from the **Database File Storage**;
- The **Database File Storage** stores the Index-24 and the Vocabulary in the B-tree (B*-tree).

Iterators Algorithms

Iterators

In OntoQuad the **Iterators** are the main building blocks of **Query Execution Plan**

All of the SPARQL algebra operators are implemented by means of the **Iterators**

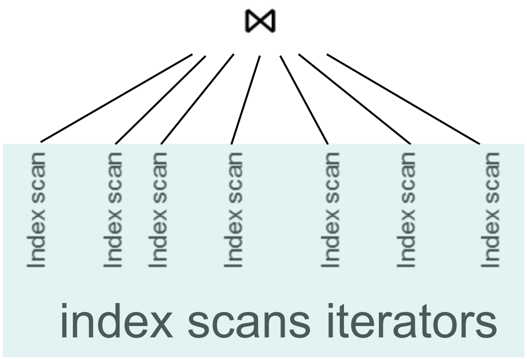


Join Iterator Family: $\bowtie$ (*L*, *R*)

- $\bowtie$ (sorted; sorted),
- $\bowtie$ (unsorted; sorted),
- $\bowtie$ (unsorted; unsorted).

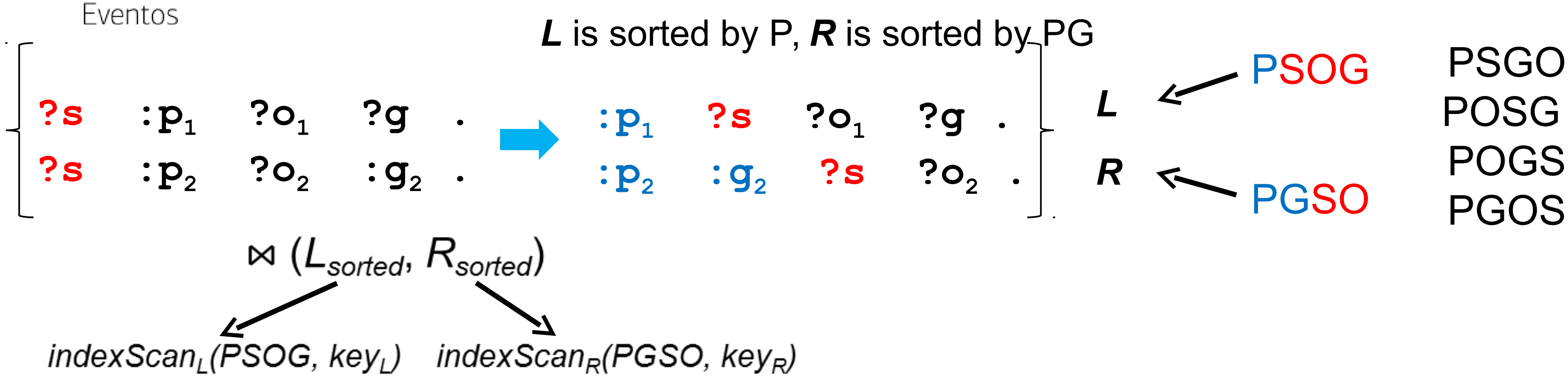
Multiple Join Iterator Family $\bowtie_M(R_1, R_2, \ldots, R_n)$

- $\bowtie_M$(sorted, sorted, …, sorted) and
- $\bowtie_M$ (unsorted, sorted, …, sorted).



All of JOIN iterators for scanning the datasets use the **lowerBound(key_{begin})** method which sets the ***begin*** pointer to the start of the range ***[key_{begin} , MAX_KEY_VAL)***

ZIG-ZAG Join Algorithm for Join Iterator



L=

S	O	G
5	o1	g1
7	o1	g2
…	…	…
10	o12	g1
20	o129	g2
50	o5	g7
…	…	…
55	o8	g126
60	o432	g15
60	o433	g9
81	o8	g43
90	o231	g65
150	o31	g98

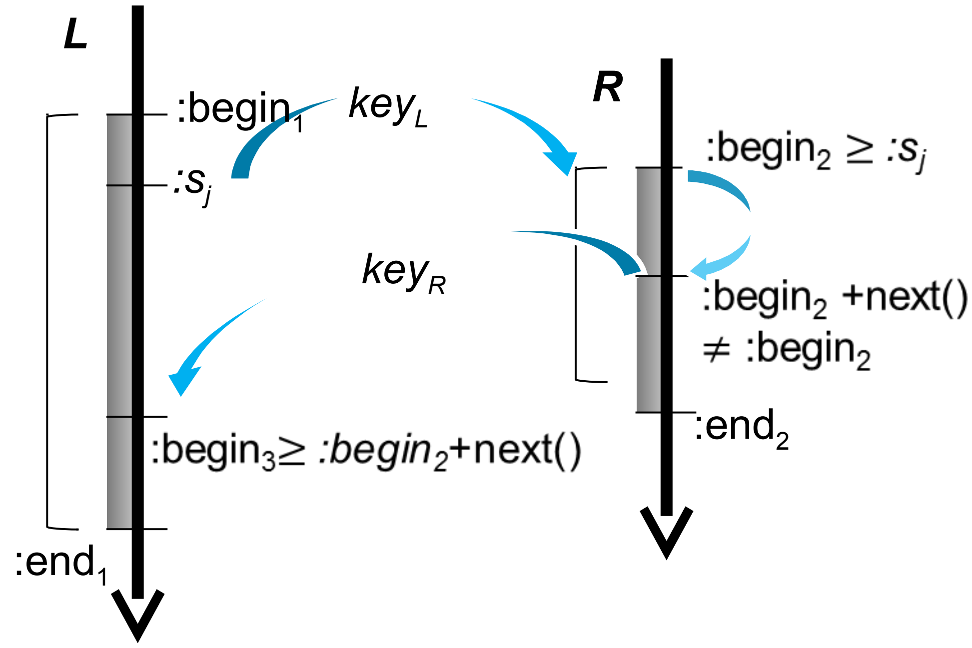
R=

S	O
1	o432
16	o8
17	o432
…	…
19	o129
20	o8
20	o31
60	o55
61	o70
…	…
67	o11

Stop

Join result

20	o129	g2	+	20	o8
20	o129	g2	+	20	o31
60	o432	g15	+	60	o55
60	o433	g9	+	60	o55

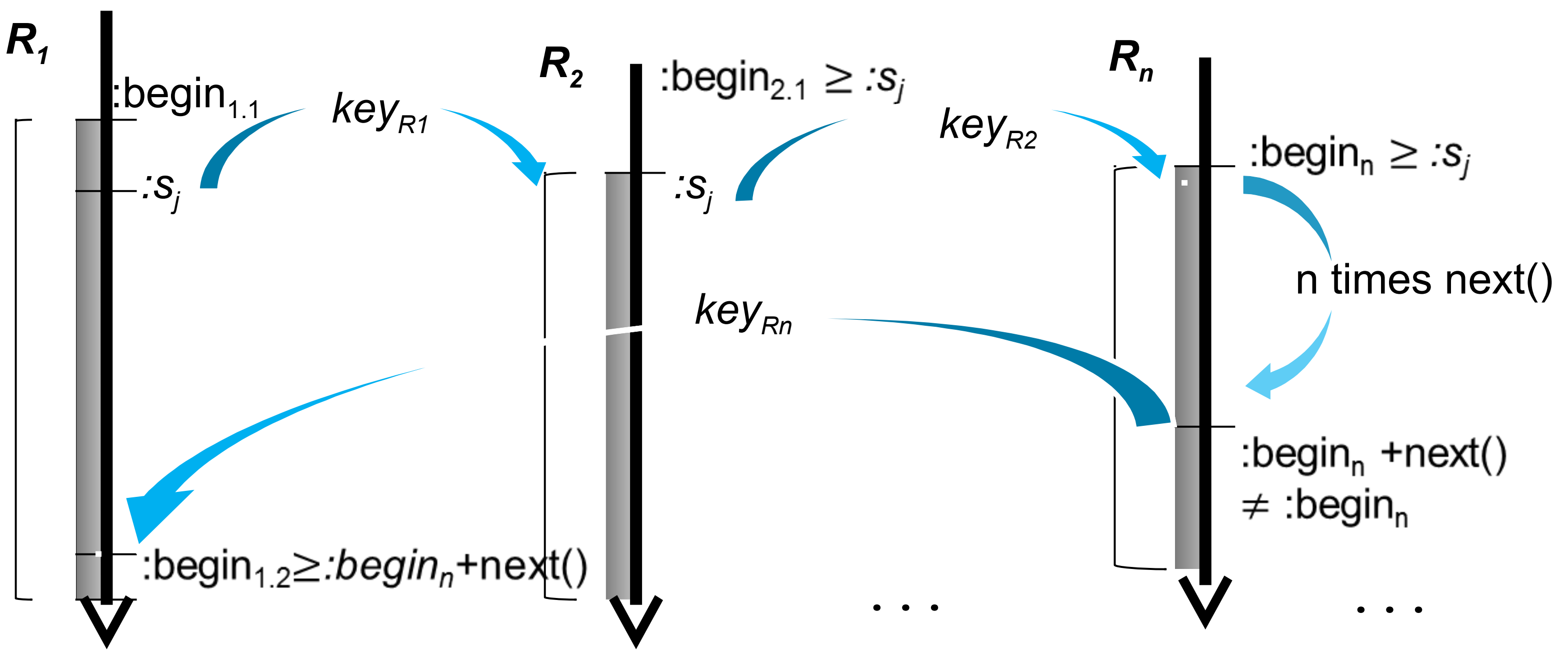


ordered sets of the *L* and *R* keys

Eventos

ZIG-ZAG Join Algorithm for Multiple Join Iterator

lowerBound(key$_{\mathbf{begin}}$) method of JOIN iterator sets ***begin*** pointer to the beginning of the range ***[key$_{begin}$, EOF)***



ordered sets of the R_1, R_2, … and R_n keys

Execution Plan Optimization Based on Heuristics

Eventos

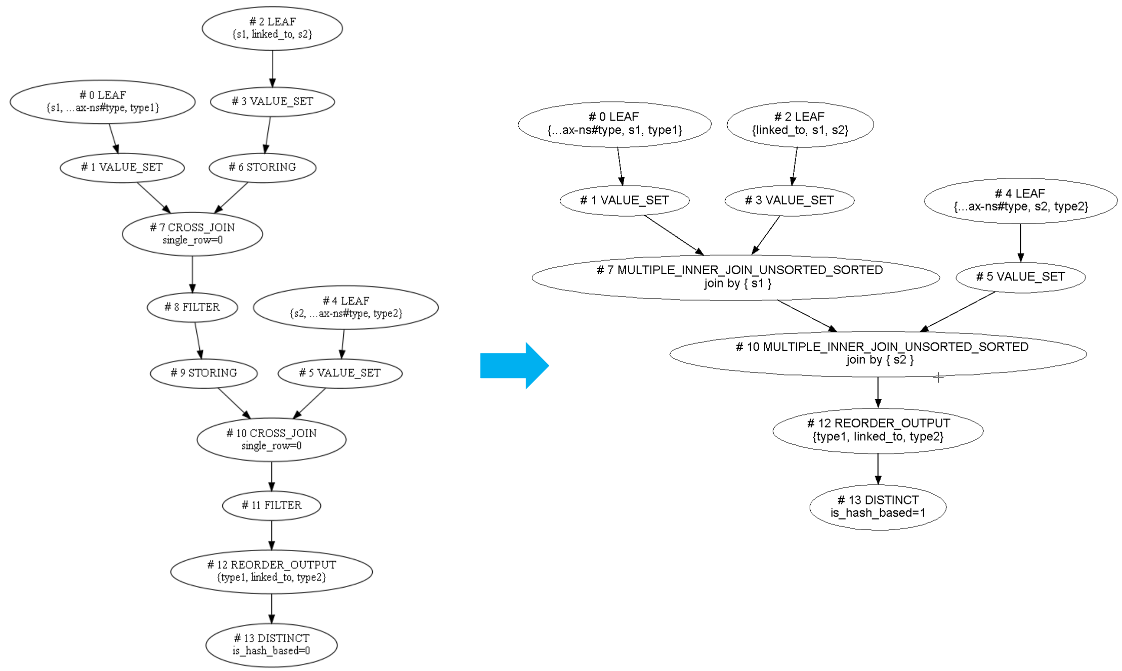
Query Execution Plan Optimization Based on Heuristics

Static Query **Heuristics-based Optimizer** transforms an initial Query Execution Plan ***D*$_0$** into an equivalent plan ***D*$_1$**.

It bases on heuristic transformations of QEP.

Heuristics List Overview:

- Leaf iterator constants shift
- Transform Cartesian product to join
- Reorder joins
- Sort Minus arguments
- Sort outer join arguments
- Remove unrequired reordering
- Execute the simplest union first
- Move Projection closer to leafs
- Move filters closer to leafs
- Merge Distinct with Sorting
- Set sorted set limit
- Chose optimal distinct algorithm
- Merge join with filter
- Replace join with multiple join
- Convert nested multiple joins to one multiple join
- and something *else* …



Eventos

“How it Works” Example

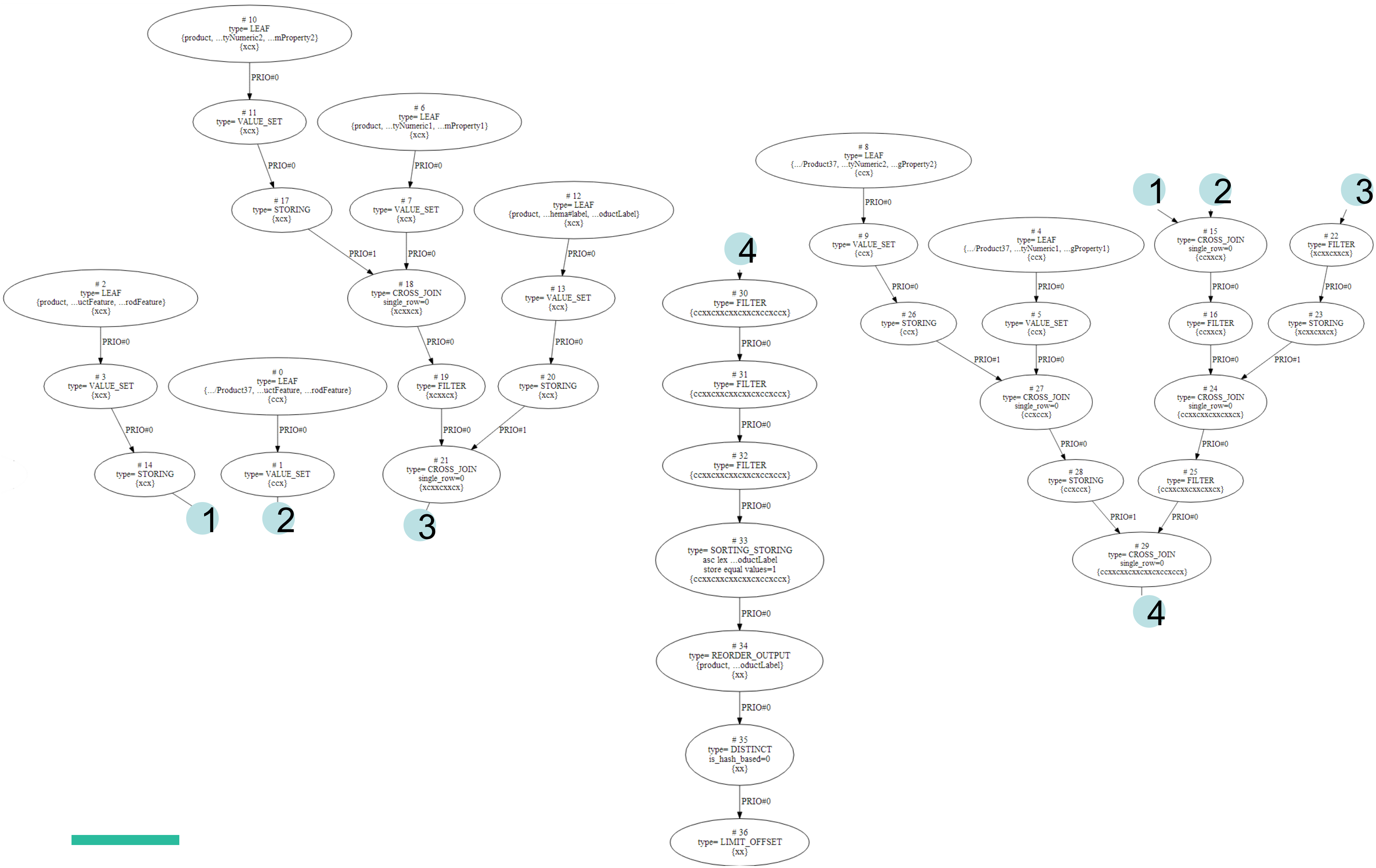
Eventos

Example. Query #5 BSBM

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>

SELECT DISTINCT ?product ?productLabel
WHERE {
   ?product rdfs:label ?productLabel .
   FILTER (<http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer3475/Product175673>
           != ?product)
   <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer3475/Product175673>
            bsbm:productFeature ?prodFeature .
   ?product bsbm:productFeature ?prodFeature .
   <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer3475/Product175673>
            bsbm:productPropertyNumeric1 ?origProperty1 .
   ?product bsbm:productPropertyNumeric1 ?simProperty1 .
   FILTER (?simProperty1 < (?origProperty1 + 120) && ?simProperty1 > (?origProperty1 - 120))
   <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer3475/Product175673>
            bsbm:productPropertyNumeric2 ?origProperty2 .
   ?product bsbm:productPropertyNumeric2 ?simProperty2 .
   FILTER (?simProperty2 < (?origProperty2 + 170) && ?simProperty2 > (?origProperty2 - 170))
}
ORDER BY ?productLabel LIMIT 5
```

Initial QEP Before the Transformations



Eventos

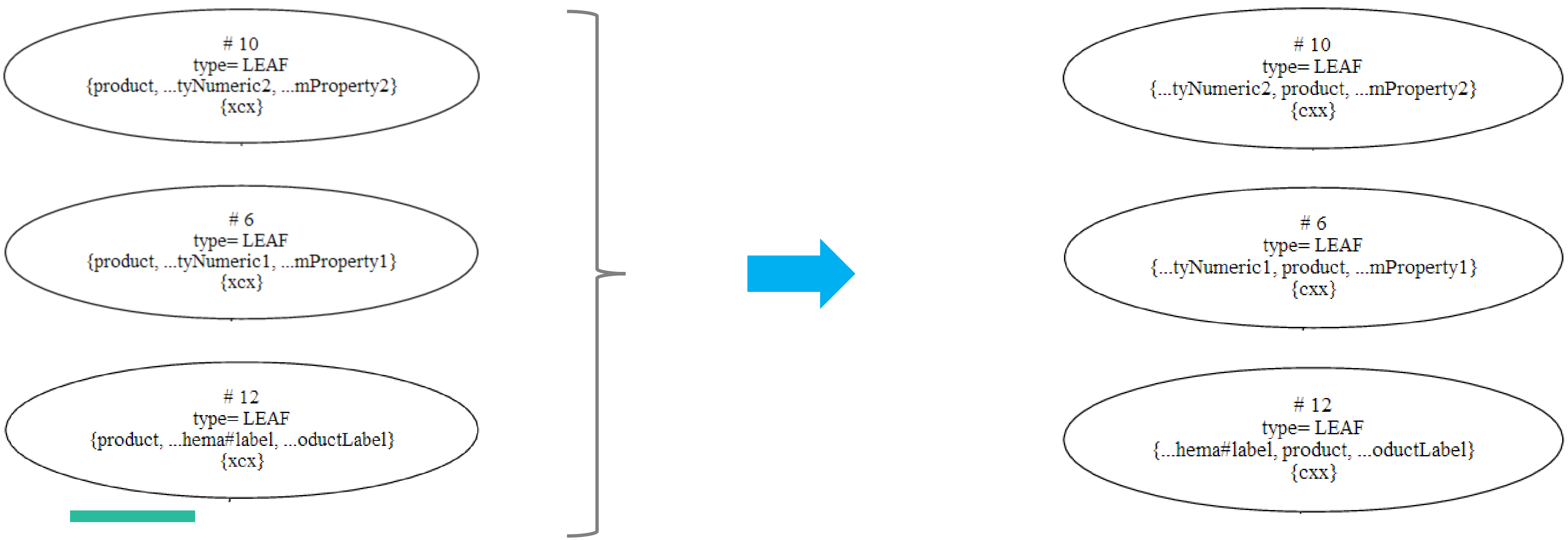
Eventos

“Leaf Iterator Constants Shift” Heuristic

?product bsbm:productFeature ?prodFeature .

bsbm:productFeature ?product ?prodFeature .

“Leaf iterator constants shift” move constants to the beginning

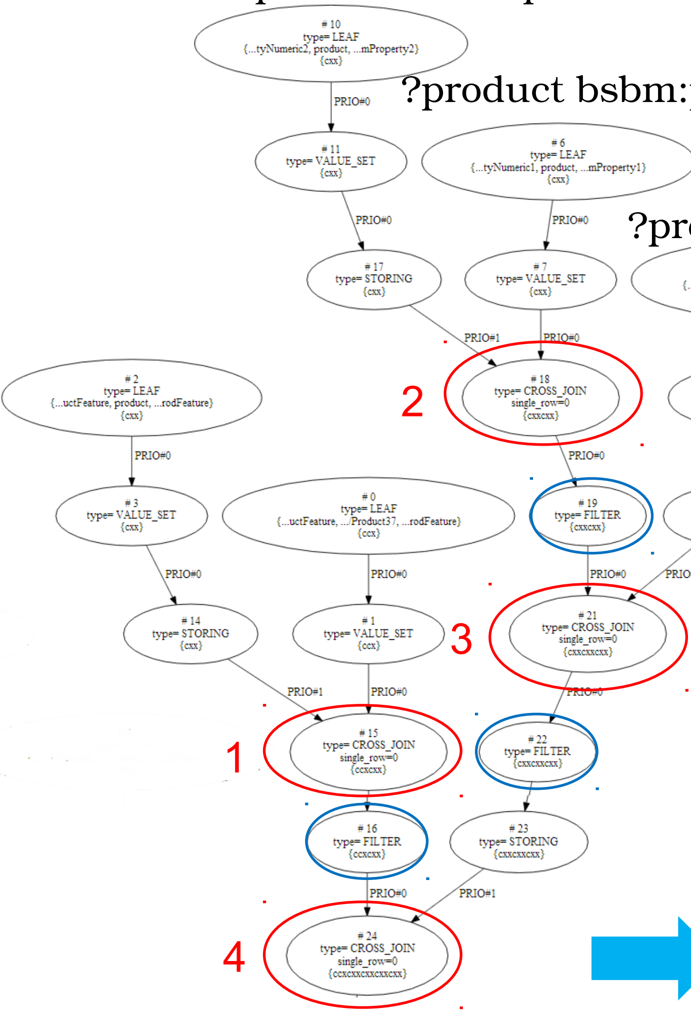


"Transform Cartesian Product to Join" Heuristic

?product bsbm:productPropertyNumeric2 ?simProperty2 .

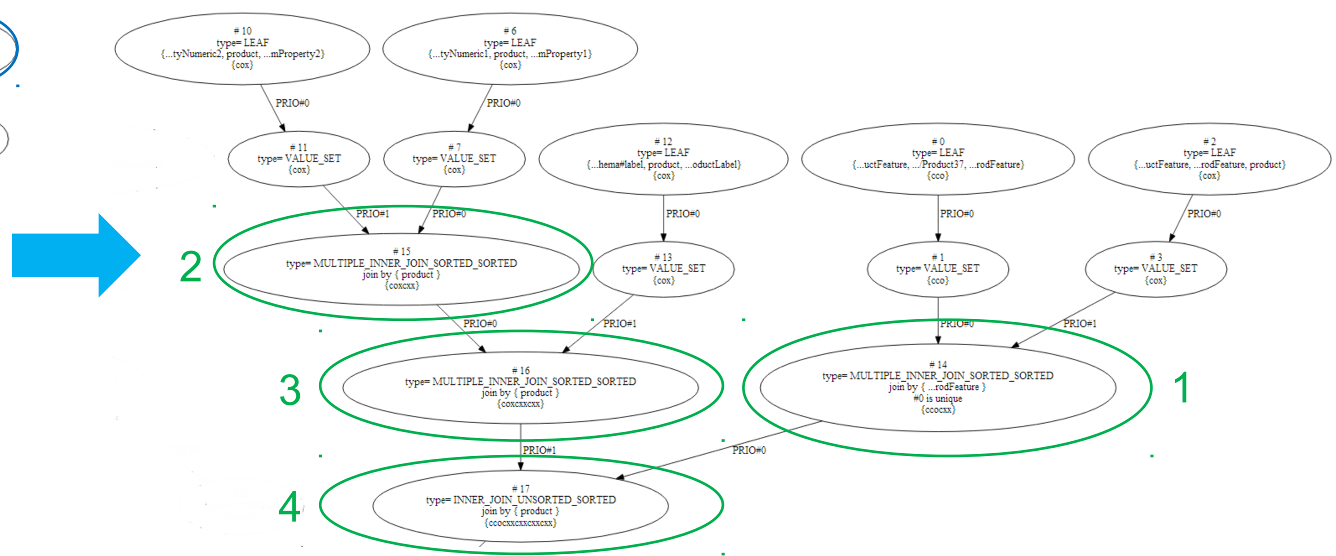
?product bsbm:productPropertyNumeric1 ?simProperty1 .

?product rdfs:label ?productLabel .



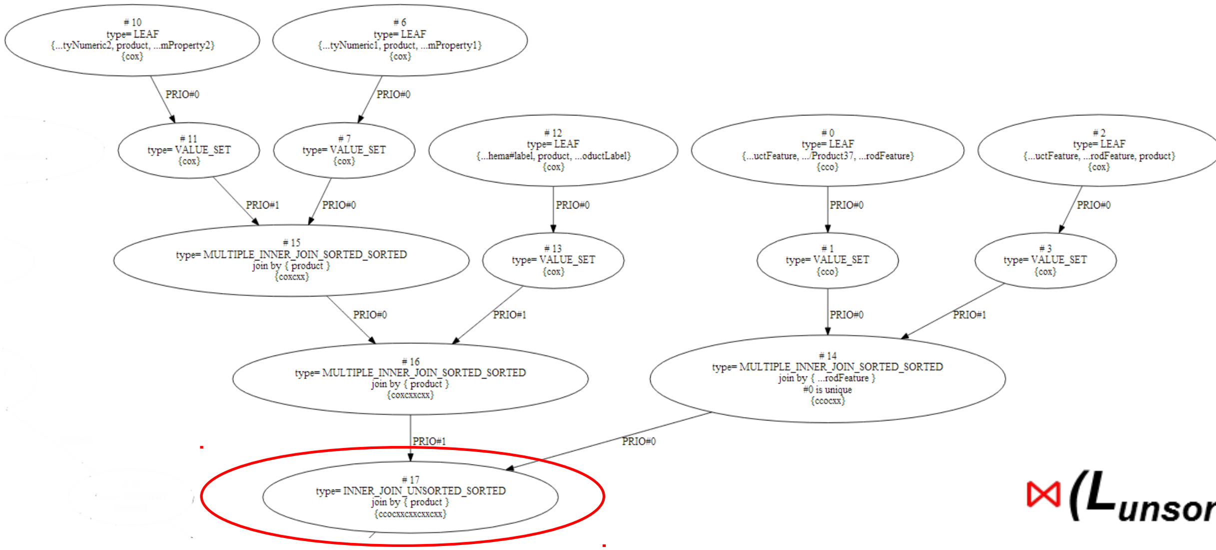
Turn the ***Cross-product*** iterator (×) into an iterator from the ***Join family*** (⋈) enabling the use of indexScan(key) (e.g., zig-zag join).

$$\sigma_F(\times(L_1, R_2)) \Rightarrow \bowtie(L_{1\ sort\_order1}, R_{2\ sort\_order2})$$



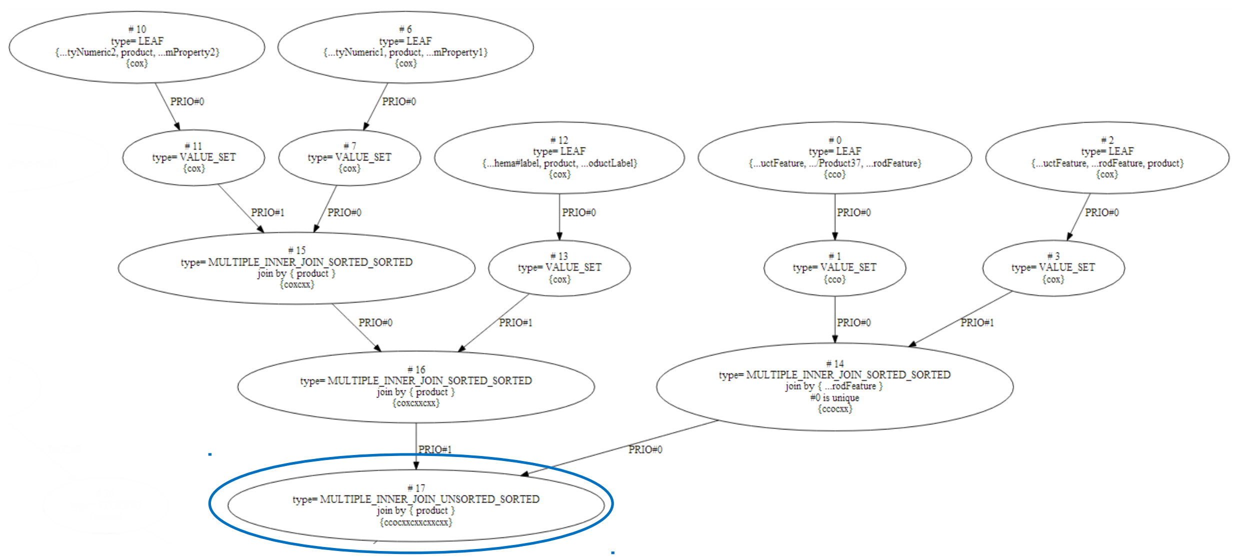
Eventos

"Replace Join with Multiple Join" Heuristic



The ***multiple join*** operator can be used instead of the ***join*** operator even in case of just two input arguments because of it is faster in our implementation.

$$\bowtie(L_{unsort}, R_{sort}) \Rightarrow \bowtie_M(L_{unsort}, R_{sort})$$

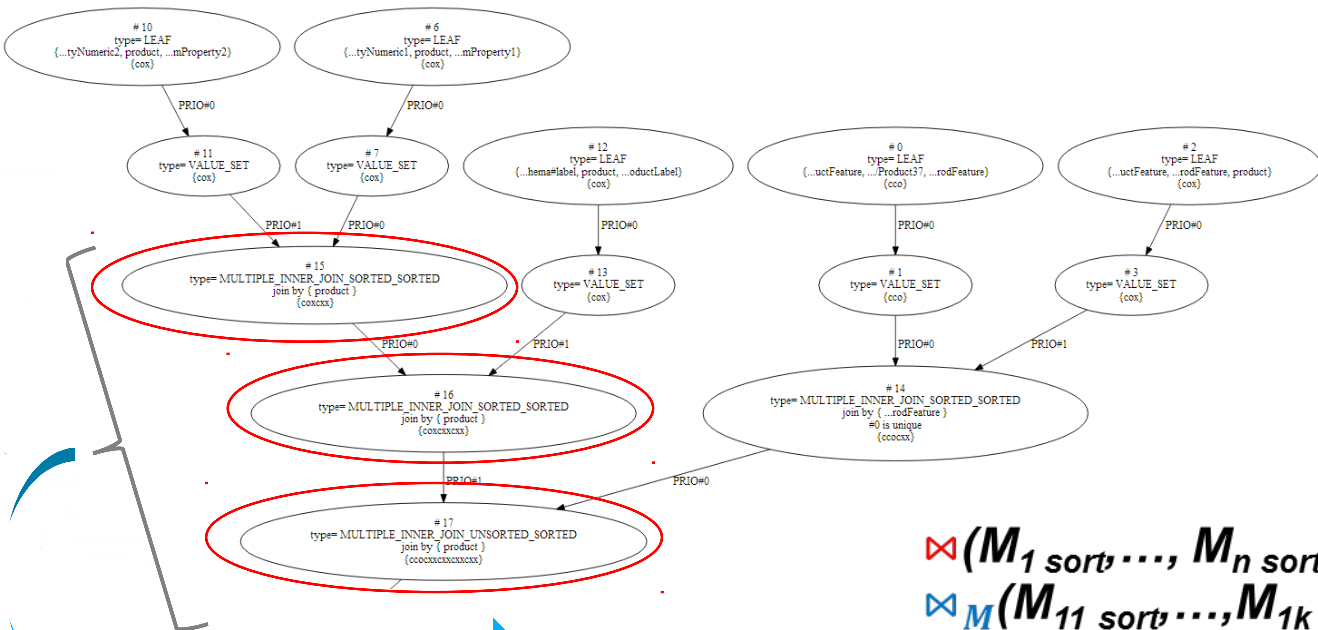


“Convert Nested Multiple Joins to One Multiple Join” Heuristic

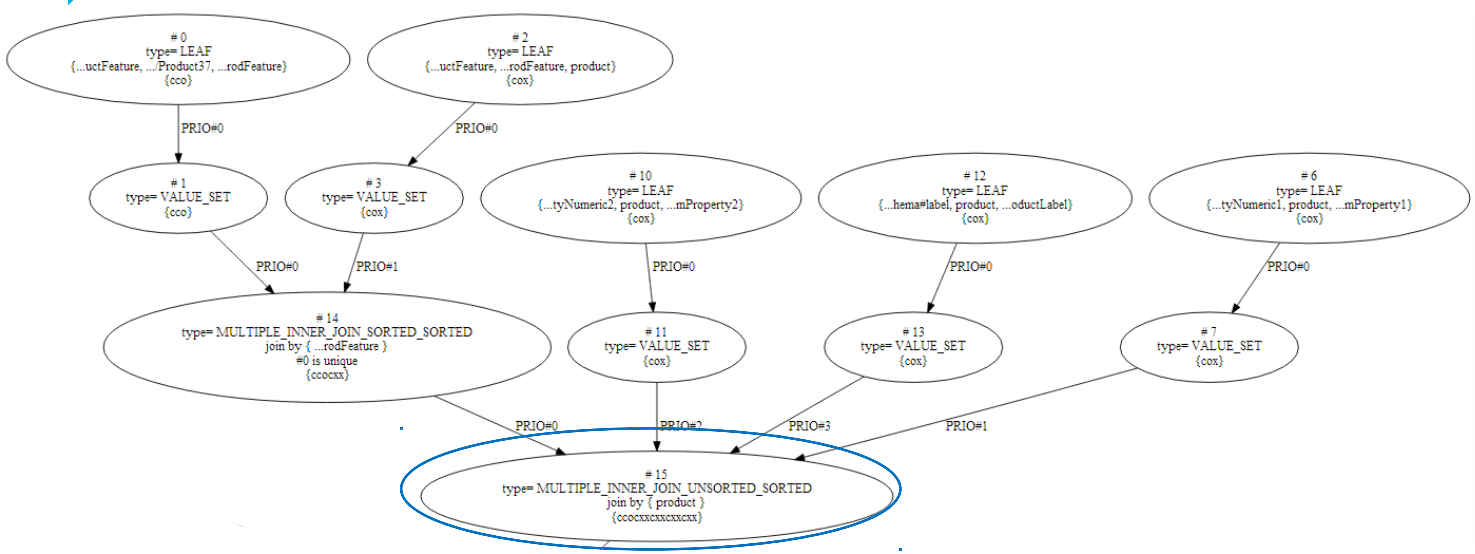
Eventos

The transformation converts several ***nested multiple join*** operators with identically sorted join variables into a ***single multiple join*** operator

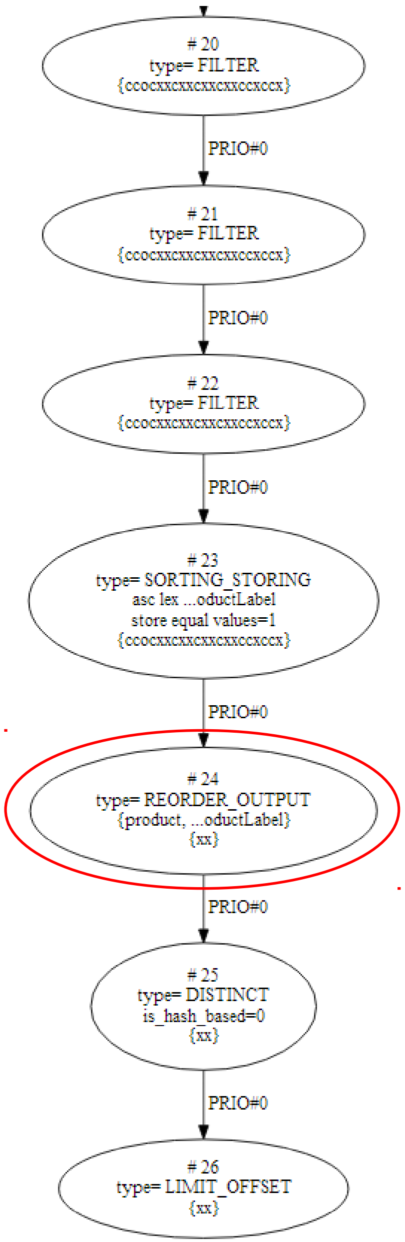
$$\bowtie(M_{1\ sort}, \ldots, M_{n\ sort}) \Rightarrow$$
$$\bowtie_M(M_{11\ sort}, \ldots, M_{1k\ sort}, \ldots, M_{n1\ sort}, \ldots, M_{nm\ sort})$$



Nested Multiple Joins



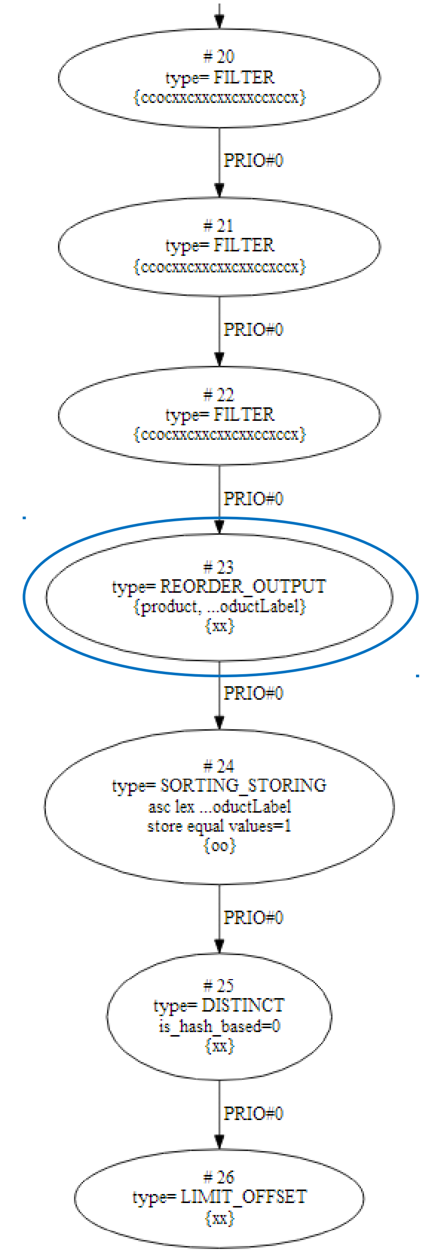
“Move Reordering Closer to the Leafs” Heuristic



ORDER BY ?productLabel

$$\tau_L(op_k(\Omega)) \Rightarrow op_k(\tau_L(\Omega))$$

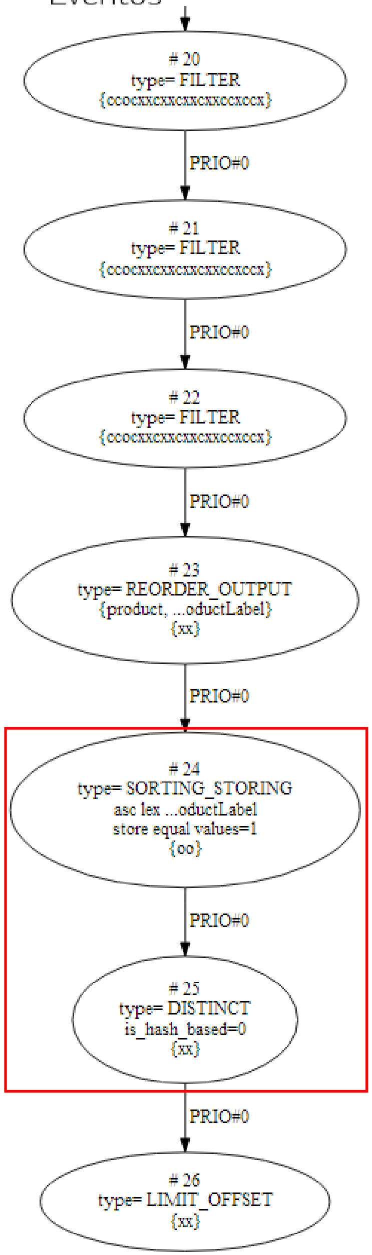
Here τ_L is the ***Order by*** operator



We also use a similar heuristic “Move Projection closer to leafs”

Eventos

“Merge Distinct with Sorting” Heuristic



$$\tau_L(\delta(\Omega)) \Rightarrow \delta\tau_L(\Omega)\,.$$

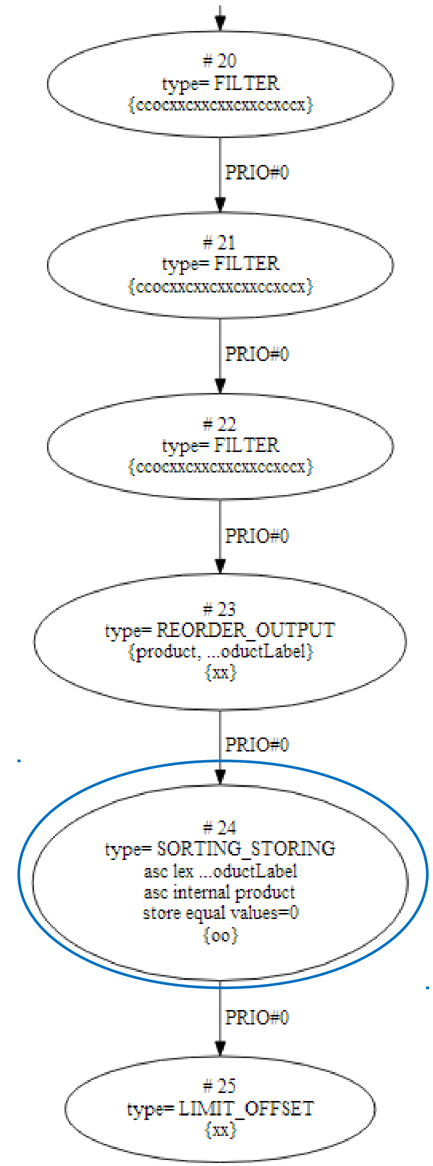
If a ***Select*** clause contains the ***Distinct*** and ***Order by*** solution modifiers, we replace them by a new iterator performing simultaneously the duplicate tuple removal and sorting functions

SELECT DISTINCT ?product ?productLabel

WHERE {

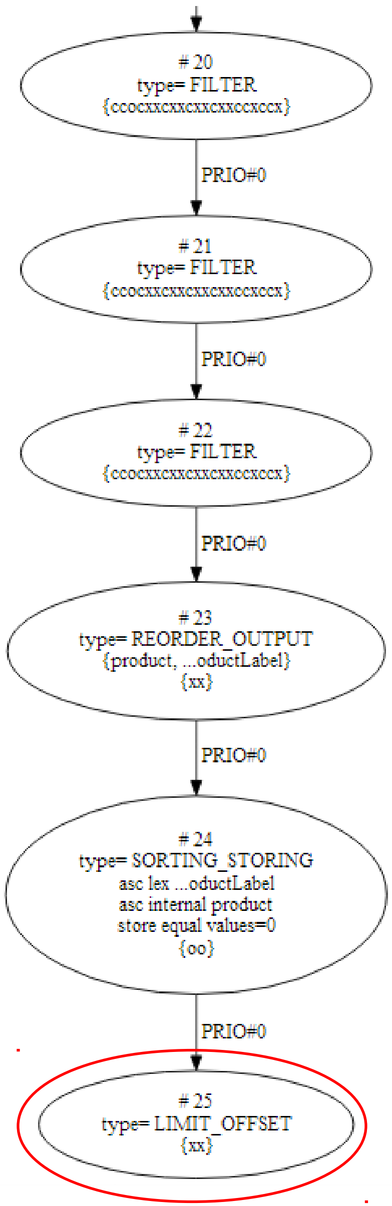
...

} ORDER BY ?productLabel LIMIT 5

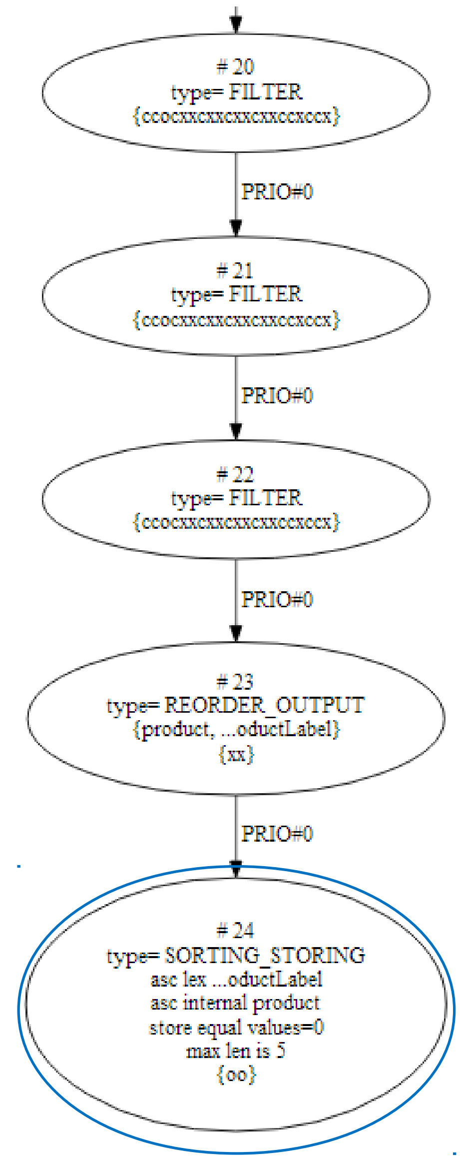


Eventos

“Set Sorted Set Limit” Heuristic



If a **Select** clause contains the **Order by** and **Limit** solution modifiers, then we create a sorted set with a size specified in the **Limit** for storing resulting tuples



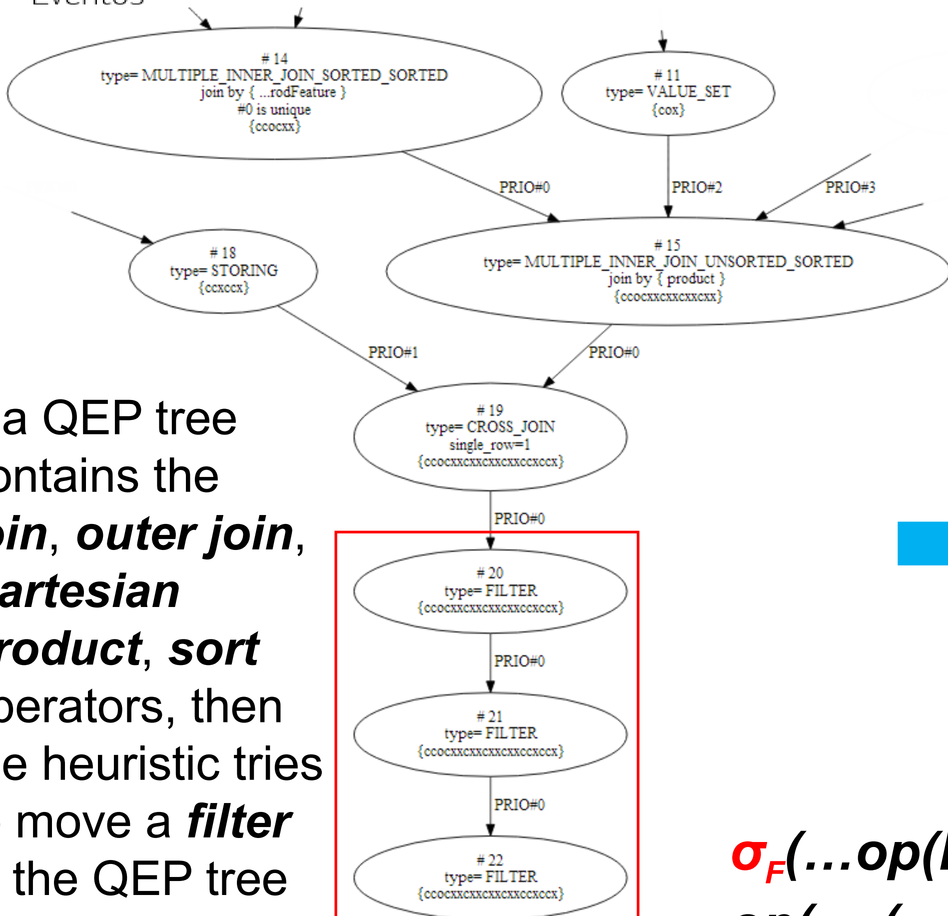
SELECT DISTINCT ?product ?productLabel

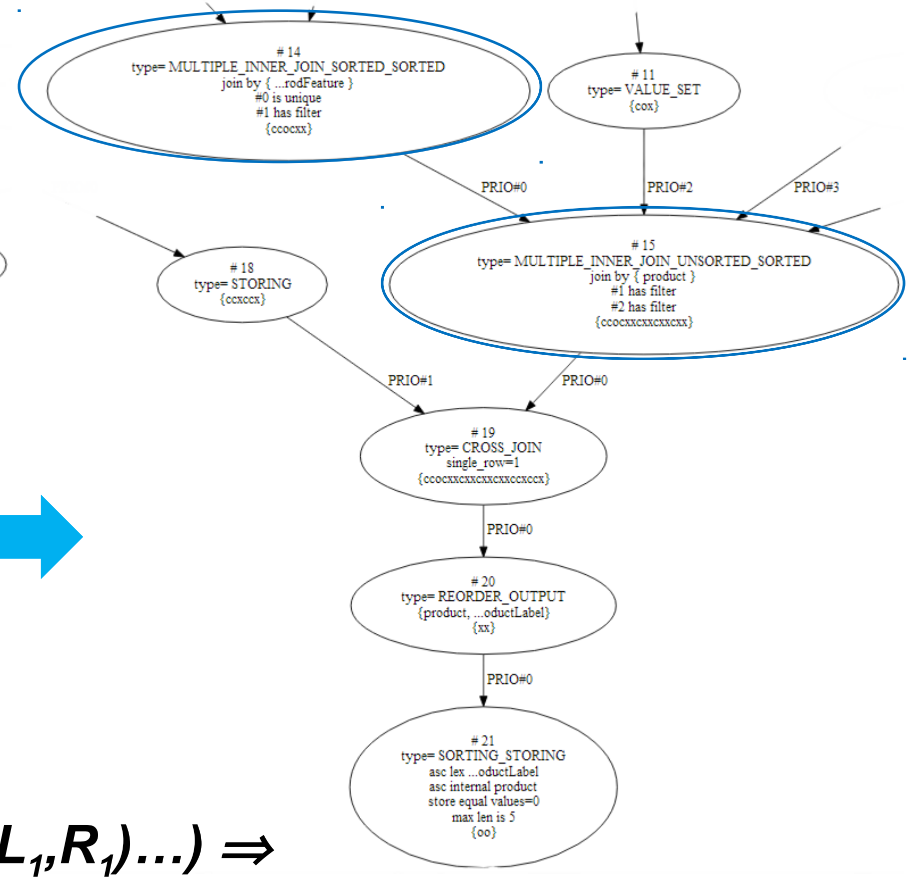
WHERE {

…

} ORDER BY ?productLabel LIMIT 5

“Move Filters Closer to Leafs” Heuristic





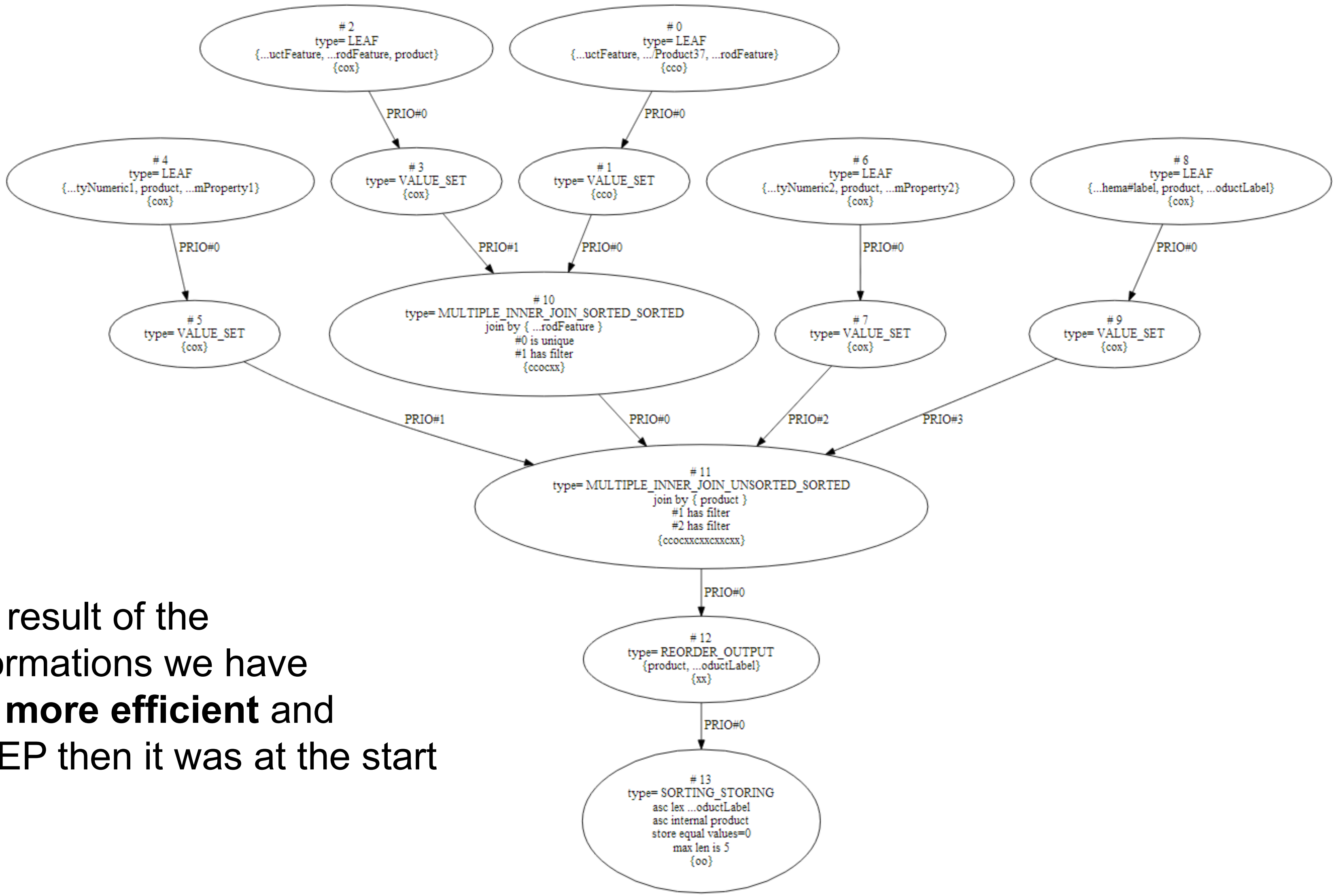
If a QEP tree contains the ***join***, ***outer join***, ***Cartesian product***, ***sort*** operators, then the heuristic tries to move a ***filter*** in the QEP tree closer to the leaf nodes, placing it before these operators

$$\sigma_F(\ldots op(L_1,R_1)\ldots) \Rightarrow op(\ldots(\ldots\sigma_{F1}(\Omega 1)\ldots\sigma_{F2}(\Omega 2)\ldots)\ldots)$$

WHERE {
FILTER (<http://…/Product175673> != ?product)
…
FILTER (?simProperty1 < (?origProperty1 + 120) && ?simProperty1 > (?origProperty1 - 120))
…
FILTER (?simProperty2 < (?origProperty2 + 170) && ?simProperty2 > (?origProperty2 - 170))
} ORDER BY ?productLabel LIMIT 5

Eventos

Resulting QEP



As the result of the transformations we have **short**, **more efficient** and **fast** QEP then it was at the start

Eventos

BSBM Evaluation

Eventos

1-st stage of the Benchmarking: Conditions and Characteristics

The 1-st stage was run in June 2013 in Universität Leipzig, Institut für Informatik, Germany

Benchmark machine

- quad-core Intel i7-3770 CPU with 32 GB of RAM.
- storage is 2x2 TB 7200rpm SATA hard drives, configured as software RAID 1.

Benchmark

Berlin SPARQL Benchmark (BSBM) Specification - V3.1, Explore Use Case

The **database size** varied from **10 million** triples, **100 million** triples and **1 billion** triples, runs done for **1**, **4**, **8**, **16** parallel clients.

All systems were configured to use 22GB of main memory.

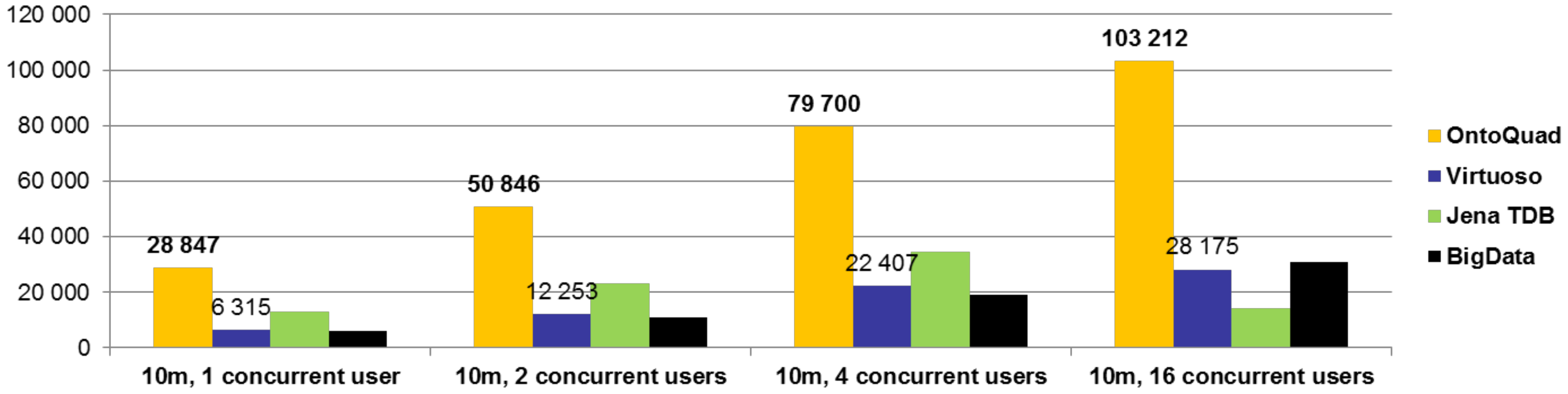
Three RDF DBMS were compared to OntoQuad

- **Virtuoso 6**.1.6,
- **Jena TDB** (Fuseki 0.2.7) and
- **BigData** (Release 1.2.2).

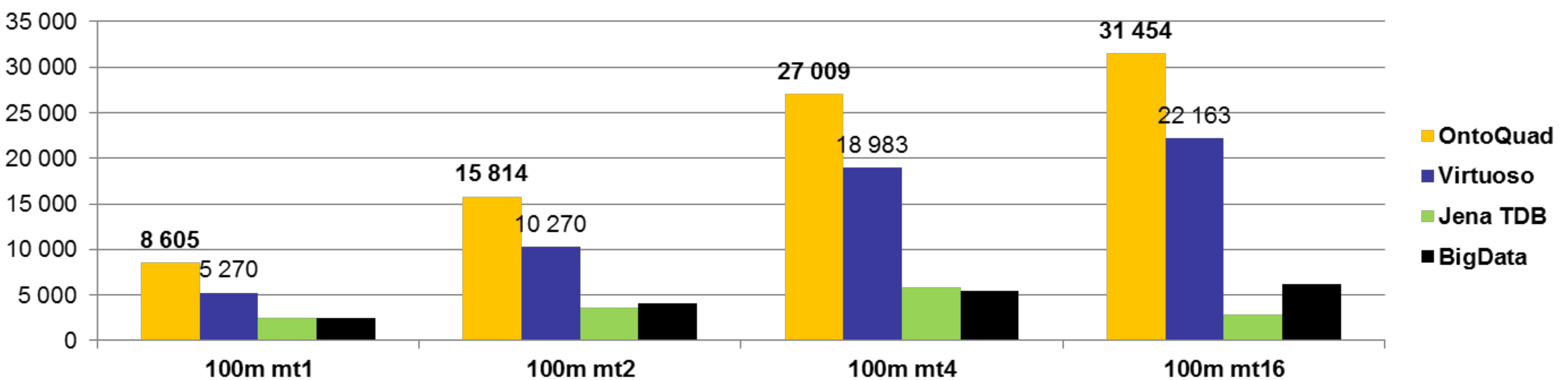
1-st stage of the Benchmarking: BSBM Explore Use Case QMpH for 10 and 100 Millions of Triples

Eventos

Query Mix per Hour for 10 millions of the triples dataset



Query Mix per Hour for 100 millions of the triples dataset



2-nd stage of the Benchmarking: Conditions and Characteristics

Eventos

The 2-nd stage was run in August - September 2013 in National Research University - Higher School of Economics, Semantic Technology Centre, Moscow, Russia. The only RDF DBMS compared to the latest version of OntoQuad is **Open source Virtuoso branch stable/7** – the leader of the BSBM tests

Benchmark machine

- VMware Virtual Platform installed on the machine with 8 processors Intel(R) Xeon(R) (16 hyper threading core) CPU X5550@2.67GHz,
- SCSI storage controller: LSI Logic / Symbios Logic 53c1030 PCI-X Fusion-MPT Dual Ultra320 SCSI, HDD 969 GB.
- 29 GB RAM, 15 GB of swap area

Benchmark

Berlin SPARQL Benchmark (BSBM) Specification - V3.1, Explore Use Case.
The database size varied from **100 million**, **200 million** and **500 million** triples, runs done for **1**, **4**, **8**, **16**, **32**, **64** parallel clients.

We used a reduced set of the query mix. Query #9 (DESCRIBE) has been excluded.

2-nd stage of the Benchmarking: Virtuoso and OntoQuad Performance Tuning

Eventos

Both Virtuoso and OntoQuad were configured to use 24 GB of main memory

Virtuoso 7

Was set up according to RDF Performance Tuning of the Virtuoso Open-Source Wiki.

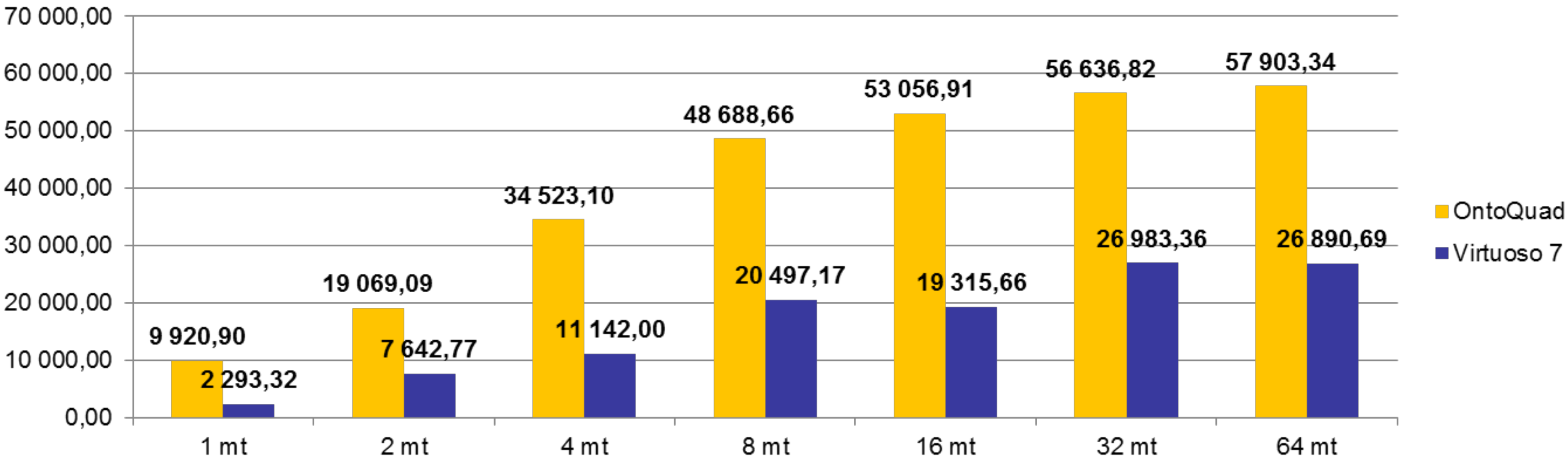
MaxCheckpointRemap	= 200000
NumberOfBuffers	= 2040000
MaxDirtyBuffers	= 1500000
CheckpointInterval	= 600

OntoQuad

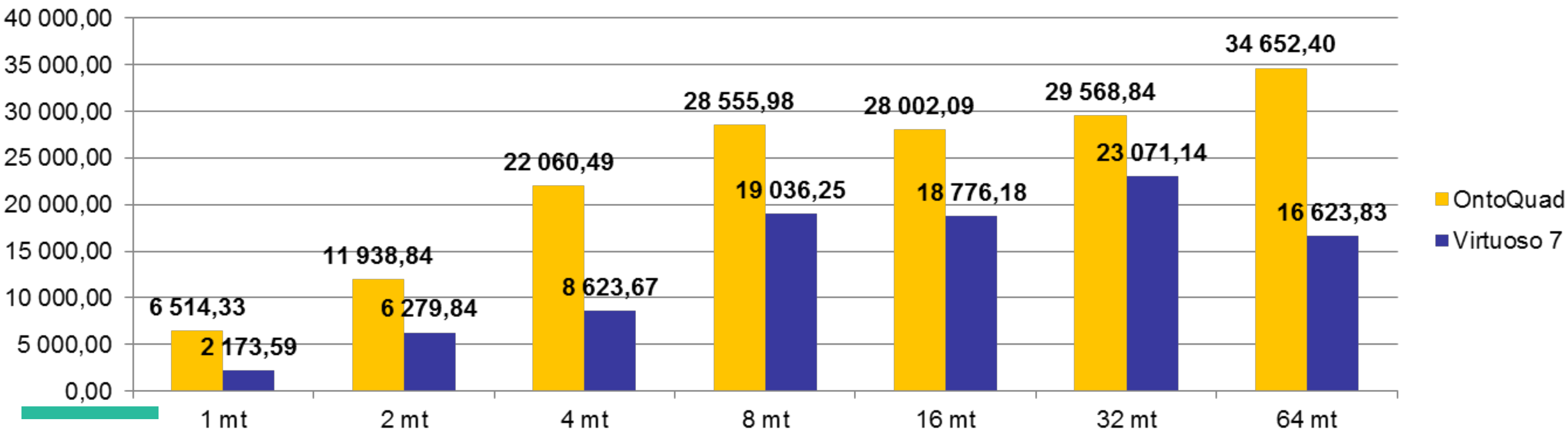
cachesize = 11811160064
compressed-page-cachesize = 13958643712

2-st stage of the Benchmarking: BSBM Explore Use Case QMpH for 100 and 200 Millions of Triples

Query Mix per Hour for 100 millions of the triples dataset



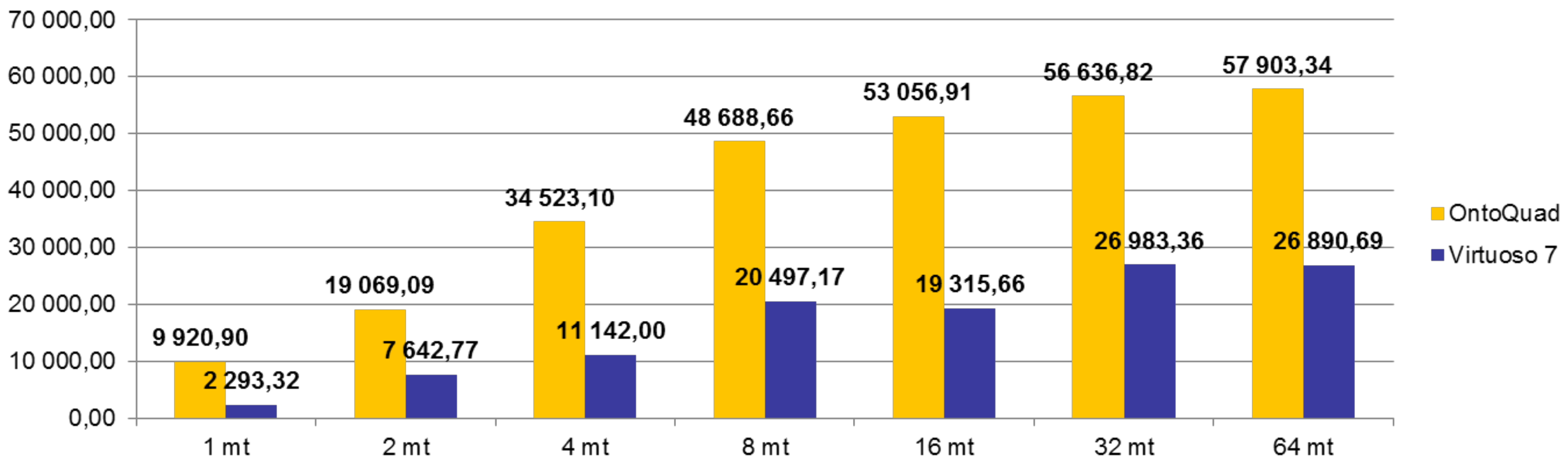
Query Mix per Hour for 200 millions of the triples dataset



2-st stage of the Benchmarking: BSBM Explore Use Case QMpH for 500 Millions of Triples

Eventos

Query Mix per Hour for 500 millions of the triples dataset



Query Mix per Hour for 200 millions of the triples dataset