VIT Chennai, 29th September 2021

Invited Lecture

as part of the series 'Knowledge Representation and Reasoning'

Auditory Introduction to Query Processing in Semantic Web Databases

Professor Dr. rer. nat. habil. Sven Groppe
https://www.ifis.uni-luebeck.de/index.php?id=groppe

Stations of my academic life

Trying to impress (not sure if it's working)...

- $>$ 122 publications
  - 93 co-authors
  - 41 publications (33%) are co-authored by bachelor/master students (being students at time of writing)
- 2 supervised dissertations, 3 current PhD students
- $\approx$ 82 bachelor/master/diploma thesis/student projects
- $\approx$ 60 contributors to LUPOSDATE
- $>$ 35 lectures, in the areas of
  - Semantic Web
  - Cloud- and Web-Technologies
  - Mobile and Distributed Databases
  - Databases
  - Algorithms and Datastructures
  - Compiler

Trying to impress (not sure if it's working)...

- 6 third-party fundings ($\approx$ € 1M)
- Scientific Services
  - Workshop Chairs
    - Semantic Big Data (SBD) @ SIGMOD (2016 - 2020)
    - Big Data in Emergent Distributed Environments (BiDEDE) @ SIGMOD (2021 - 2021)
    - Very Large Internet of Things (VLIoT) @ VLDB (2017 - 2021)
    - Web Data Processing and Reasoning (WDPAR) @ KI 2018
  - General Chair
    - International Semantic Intelligence Conference (ISIC) 2021 - 2022
  - many other scientific services
    - $\approx$ 90 PC memberships
    - reviewer of $\approx$ 30 journals
    - editor of 3 journals

Trying to impress (not sure if it's working)...

Detailed Information available at
https://www.ifis.uni-luebeck.de/~groppe

Agenda: Introduction to Query Processing in Semantic Web Databases

- SPARQL Query Language
  - Transformation to Relational Algebra Expression/Operatorgraph
- Sound of Databases
- RDF3X Indices
- Merge Join
- Sideways Information Passing (SIP) in
  - Merge Joins
  - RDF3X B^+-Tree
- Hash Join
- Inspiring Sounds

Semantic Web (Core) "Standards"

Query:

SPARQL

Ontology:

RDFS

OWL (2)

Rule:

RIF

Data Format: RDF

- Every data model (here Semantic Web) has its own set of languages (data, query, rule, ...)

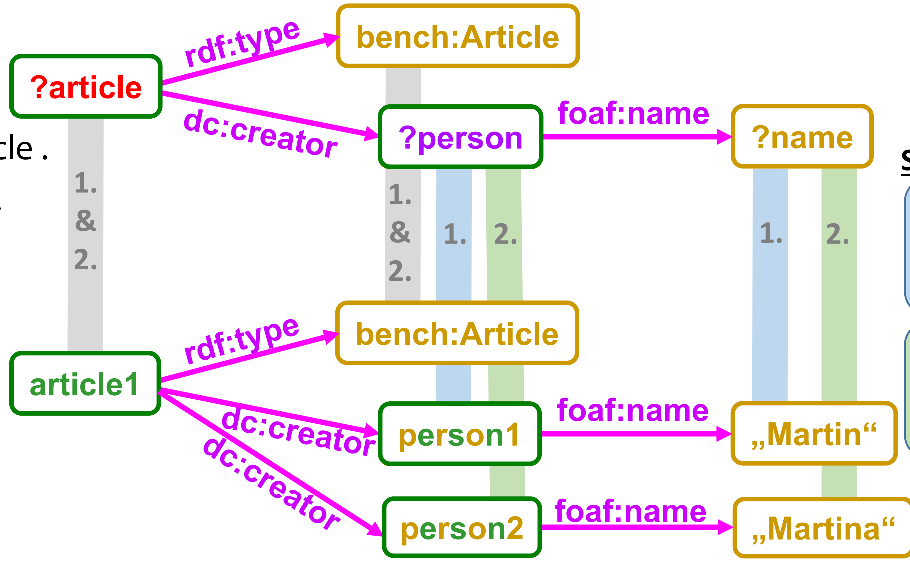
Recap: SPARQL Query and its Result

SPARQL Query:

```
SELECT ?person ?name
WHERE {
  ?article rdf:type bench:Article .
  ?article dc:creator ?person .
  ?person foaf:name ?name .
}
```

Query Graph:

RDF Graph:



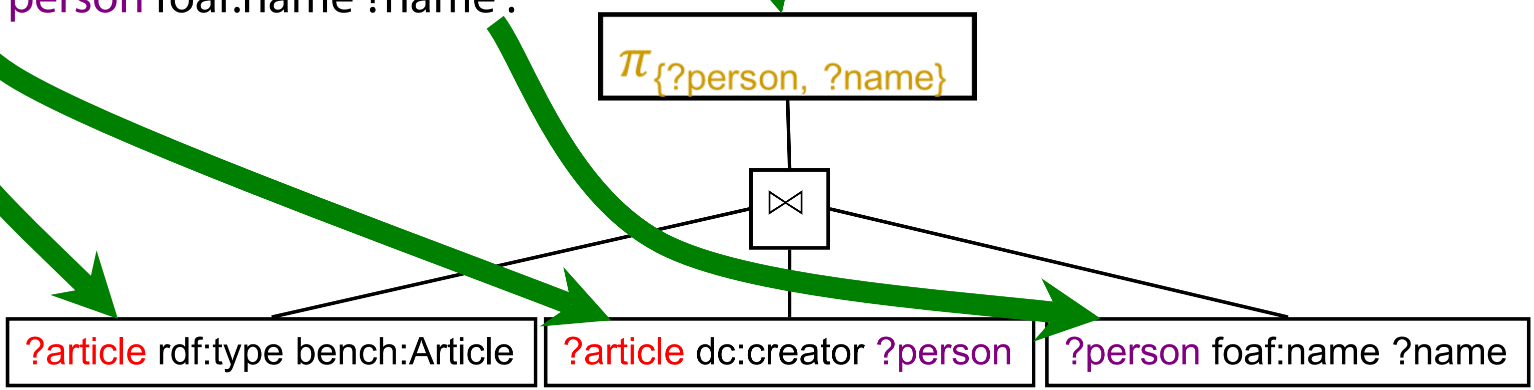
Solutions:

1. ?article=article1
   ?person=person1
   ?name=„Martin“

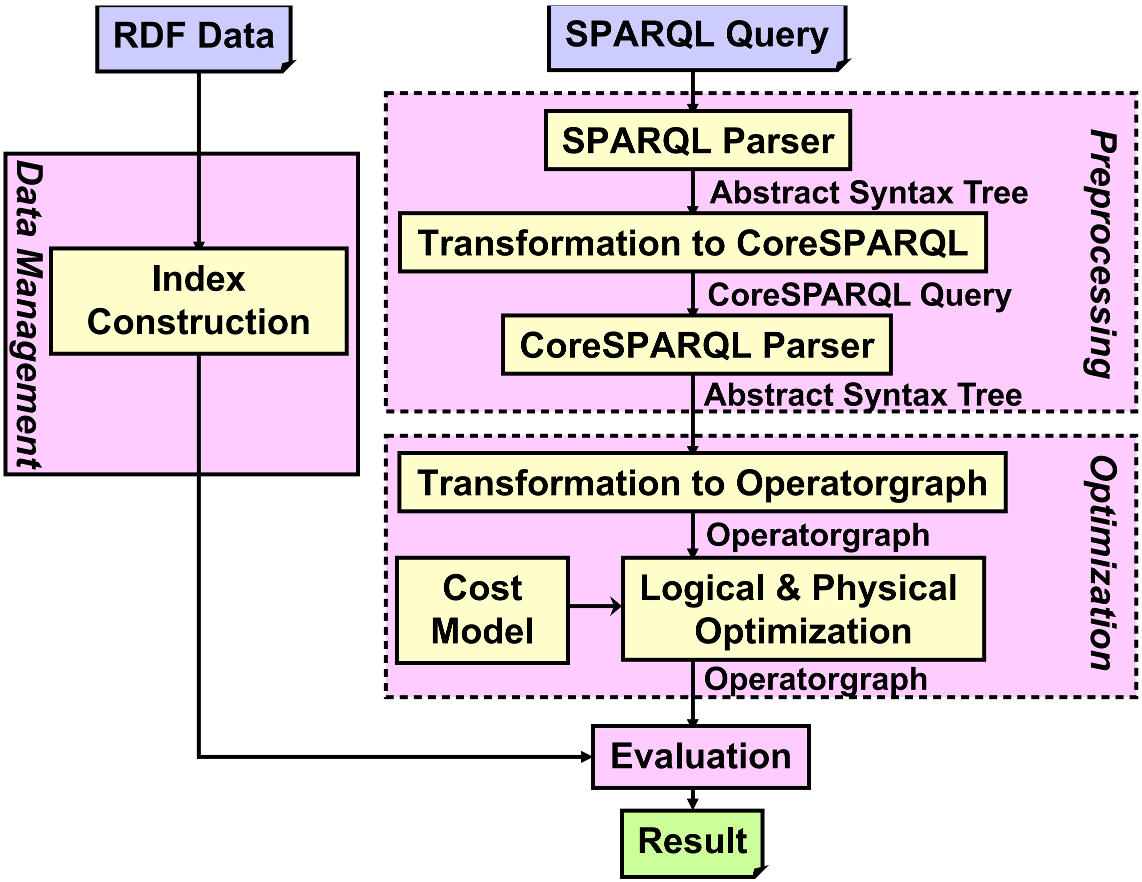
2. ?article=article1
   ?person=person2
   ?name=„Martina“

Transformation to Relational Algebra Expression

```
SELECT ?person ?name
WHERE {
 ?article rdf:type bench:Article .
 ?article dc:creator ?person .
 ?person foaf:name ?name .
}
```



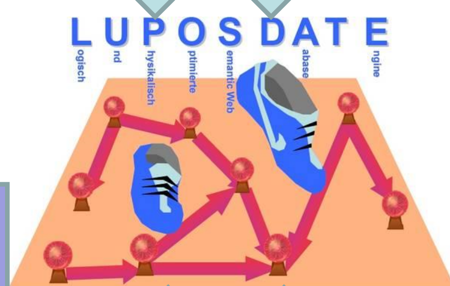
Architecture of a Semantic Web Database Management System



Semantic Web DBMS LUPOSDATE

Support of:
- SPARQL Queries
- RIF Rules
- RDF Schema
- OWL (via OWL2RL in RIF)

Indexing:
- Stream Processing
- Main memory for small datasets
- Disk-based for large datasets
  - RDF3X
- Cloud: HBase
- P2P



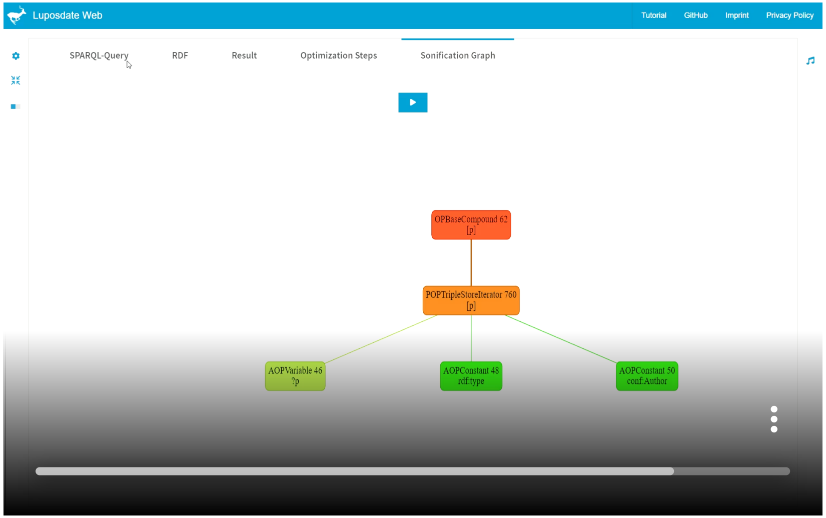
Visualizations:
- Visual Editor
  - Queries (SPARQL)
  - Rules (RIF)
  - Data (RDF) in
    - 2D and
    - 3D
  - Logical Optimization Rules
- Summaries of RDF Data
- Operator graph
- Processing of Queries and Rules
- Optimization Steps

Extra:
- Parallel Processing
- Distributed Processing
- Cloud Computing
- Mobile Computing
- P2P for Internet of Things
- Compression of RDF Data
- Embedding of SW Languages in Programming Languages
- Speeding up by FPGAs

LUPOSDATE3000

- Successor of LUPOSDATE
- Multiplatform Database
  - programming language Kotlin
    - targets: JVM, JS, native binaries (win/linux/macos/iOS/...)
- Focus on:
  - parallel databases
  - distributed databases for Internet-of-Things
  - Semantic Web: RDF, SPARQL
- Planned:
  - RDFS/OWL Inference
  - multi-model support

Sonification of Triple Pattern Evaluation



Start Sonification

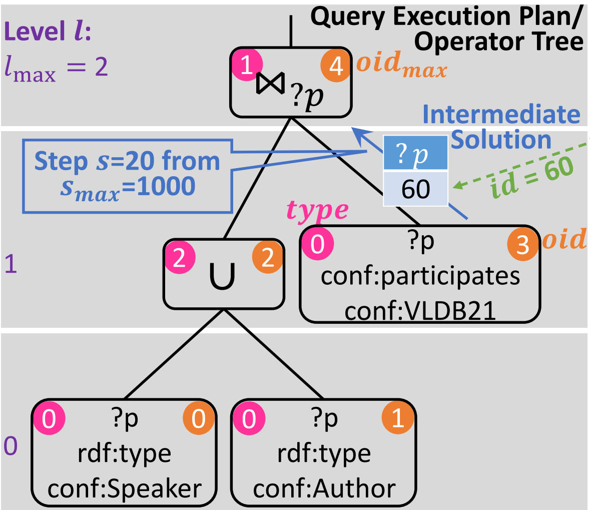
- What do you hear?
  - An ascending scale (in pitch)
- We learn: Semantic Web data often stored in indices
  - Output is pre-sorted
  - Maximize usage of fast merge joins

Sonification

- *use of nonspeech audio to convey information [...] for the purposes of facilitating communication or interpretation*[1]
- Examples
  - [brain acticity]
  - [from deep space to cancer research]
  - [Sound of Sorting]
  - ...
- Motivation
  - reduces barriers for people with poor education
  - presents multidimensional data in an easy-to-understand way
  - additional medium helps with learning new information
  - helps humans with visual impairments

[1] Kramer et al., 1999. The sonification report: Status of the field and research agenda. International Community for Auditory Display (ICAD)

Sound of Databases



Dictionary:

id	value
0	…
…	…
60	conf:Sven
…	…
id_{max} =100	…

Op.Type Mapping:

type	value
Triple pattern	0
⋈ (Join)	1
∪ (Union)	2
…	…
…	$type_{max}$ =45

Possible Mappings m_i of one processing step to $[0;1]$ based on

a) value: $m_0 = \frac{id}{id_{max}} = \frac{60}{100} = 0.6$

b) depth: $m_1 = \frac{l}{l_{max}} = \frac{1}{2} = 0.5$

c) operator id: $m_2 = \frac{oid}{oid_{max}} = \frac{3}{4} = 0.75$

d) operator type: $m_3 = \frac{type}{type_{max}} = \frac{0}{45} = 0$

e) processing step: $m_4 = \frac{s}{s_{max}} = \frac{20}{1000} = 0.02$

f) (weighted) combinations of above mappings:
$m_5 = \sum_{i \in \{0,\ldots 4\}} w_i \cdot m_i$, where $w_i \in [0;1]$
chosen, such that $\sum_{i \in \{0,\ldots 4\}} w_i = 1$
for guaranteeing $m_i \in [0;1]$

?

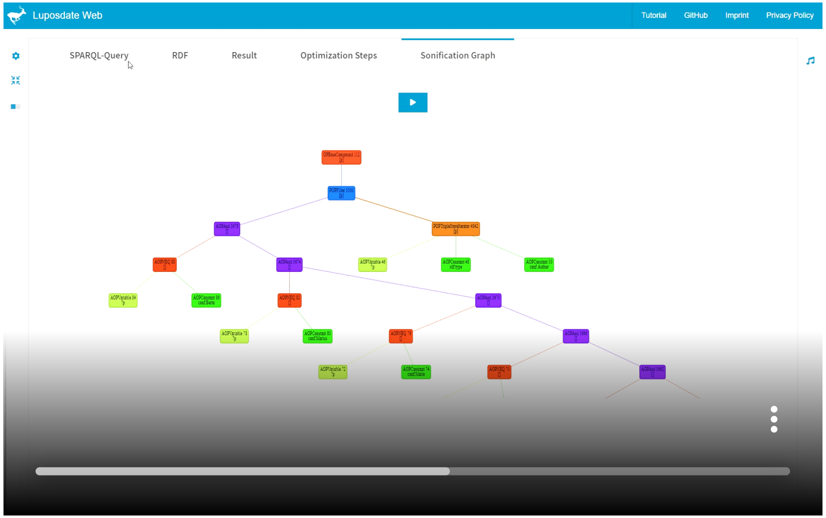
Auditory Dimension
pitch
loudness
instrument
spatia-lization
duration
brightness
timbre
tempo
spectral power
…

⋮

- useful mapping for understanding the operator algorithm (or at least its in- & output):
  - each operator plays another instrument (easy to distinguish from the other)
  - value of intermediate solution maps to pitch
- mapping for melodies: operator depth maps to pitch
- Online: https://www.ifis.uni-luebeck.de/~groppe/soundofdatabases/

Please guess the kind of operator!

- Solution: Selection/Filter



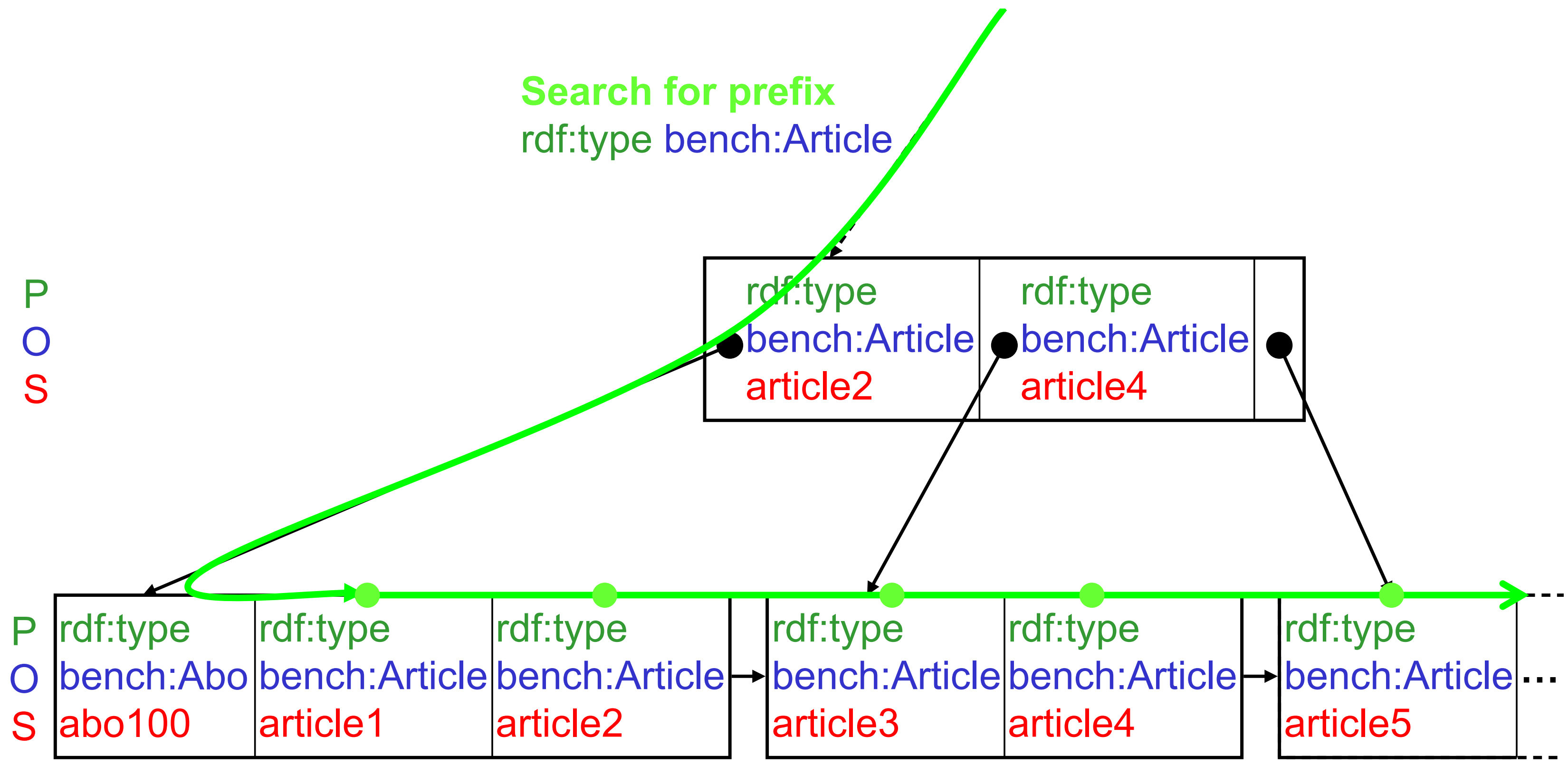
Start Sonification

- not all intermediate results are passing the filter operator

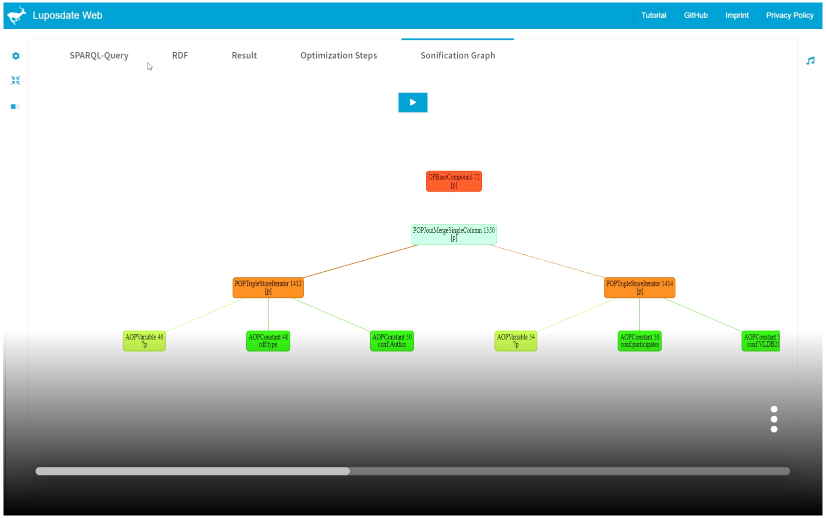
RDF3X Indices for Triple Pattern Evaluation

Triple Pattern	1st Alternative Index	1st Alternative Prefix Key	1st Alternative Sort Criterium	2nd Alternative Index	2nd Alternative Prefix Key	2nd Alternative Sort Criterium
?s f:p f:o	POS	f:p f:o	?s	OPS	f:o f:p	?s
f:s ?p f:o	SOP	f:s f:o	?p	OSP	f:o f:s	?p
f:s f:p ?o	SPO	f:s f:p	?o	PSO	f:p f:s	?o
?s ?p f:o	OSP	f:o	?s ?p	OPS	f:o	?p ?s
?s f:p ?o	PSO	f:p	?s ?o	POS	f:p	?o ?s
f:s ?p ?o	SPO	f:s	?p ?o	SOP	f:s	?o ?p
f:s f:p f:o	SPO	f:s f:p f:o	-	All other indices with corresponding keys		

B+-tree


Search for prefix
rdf:type bench:Article
P
O
S
rdf:type
bench:Article
article2
rdf:type
bench:Article
article4
P
O
S
rdf:type
bench:Abo
abo100
rdf:type
bench:Article
article1
rdf:type
bench:Article
article2
rdf:type
bench:Article
article3
rdf:type
bench:Article
article4
rdf:type
bench:Article
article5
...

Joining the Result of 2 Triple Patterns



Start Sonification

- What to learn?
  - The result is again sorted if merge join is used

$\bowtie_=$ Equi-Join

- Following join algorithms are only suitable for Equi-Joins
- $R \bowtie_C S = \sigma_C(R \times S)$, where C is join condition of the form:
  - $r = s$, where r is attribute of R and s is attribute of S
  - $C_1 \wedge C_2$, where C_1 and C_2 are join conditions

Set Semantics

e.g.: R

A	B
1	2
0	2
3	4
7	8

S

B	C
2	3
2	4
5	6
4	5

$R \bowtie_{R.B=S.B} S$

A	R.B	S.B	C
1	2	2	3
0	2	2	3
1	2	2	4
0	2	2	4
3	4	4	5

Multi-Set Semantics

e.g.: R

A	B
1	2
0	2
7	8
3	4

S

B	C
2	3
4	5
4	5
4	5
5	6

$R \bowtie_{R.B=S.B} S$

A	R.B	S.B	C
1	2	2	3
0	2	2	3
3	4	4	5
3	4	4	5
3	4	4	5

Merge Join

- **Precondition:** Input sorted according to join columns
  - Otherwise a preceding sort phase necessary $\longrightarrow$ "Sort Merge Join"
- **Algorithm:** step-by-step comparison of both sorted (input) relations
  - If it cannot be joined, then it will be read from the relation with the currently smaller value, because this smaller value does not occur any more in the other relation (utilizing the sorting)

Merge Join as Iterator

```
open() {
  R.open(); r = R.next()
  S.open(); s = S.next()
  B = {} × {}; B.open()
}

close() {
  R.close()
  S.close()
}
```

Runtime complexity for
- Input without duplicates in join columns: $O(|R| + |S|)$
- Input with duplicates in join columns: $O(|R| \times |S|)$

```
next() {
  IF(B.hasNext()) RETURN B.next()
  IF(s == null ∨ r == null) RETURN null
  WHILE(π_Join Attributes(s) ≠ π_Join Attributes(r)){
    IF(π_Join Attributes(s) < π_Join Attributes(r)){
      s = S.next()
      IF(s == null) RETURN null
    } ELSE { // r < s (see WHILE- and IF-condition!)
      r = R.next()
      IF(r == null) RETURN null
    }
  }
  cmp = π_Join Attributes(s) // = π_Join Attributes(r) (see WHILE-condition)
  ts = {}; WHILE(π_Join Attributes(s) == cmp) { ts = ts ∪ s; s = S.next() }
  tr = {}; WHILE(π_Join Attributes(r) == cmp) { tr = tr ∪ r; r = R.next() }
  B = ts × tr; B.open()
```

Merge Join 1/11

- Result $R \bowtie_{R.lang=S.lang} S$:
- Buffer:
- Input:

R

first	lang
hallo	de
hello	en
hi	en
hola	sp

S

lang	second
de	Student
en	student
en	collegian
cn	daxuesheng

Precondition: Inputs must be sorted according to join columns

Merge Join 2/11

- Result $R \bowtie_{R.lang=S.lang} S$:

- Buffer:

R

first	lang
hallo	de

S

lang	second
de	Student

Load rows with smallest values in join columns into buffer

- Input:

R

first	lang
hello	en
hi	en
hola	sp

S

lang	second
en	student
en	collegian
cn	daxuesheng

Merge Join 3/11

- Result $R \bowtie_{R.lang=S.lang} S$:

- Buffer:

  $R \bowtie_{R.lang=S.lang} S$

first	R.lang	S.lang	second
hallo	de	de	Student

  Compute join result for current content of buffer

- Input:

R

first	lang
hello	en
hi	en
hola	sp

S

lang	second
en	student
en	collegian
cn	daxuesheng

Merge Join 4/11

- Result $R \bowtie_{R.lang=S.lang} S$:

first	R.lang	S.lang	second
hallo	de	de	Student

Emit join row

- Buffer:
- Input:

R

first	lang
hello	en
hi	en
hola	sp

S

lang	second
en	student
en	collegian
cn	daxuesheng

Merge Join 5/11

- Result $R \bowtie_{R.lang=S.lang} S$:

first	R.lang	S.lang	second
hallo	de	de	Student

- Buffer:

R

first	lang
hello	en
hi	en

S

lang	second
en	student
en	collegian

Load rows with smallest values in join columns into buffer

- Input:

R

first	lang
hola	sp

S

lang	second
cn	daxuesheng

Merge Join 6/11

- Result $R \bowtie_{R.lang=S.lang} S$:

first	R.lang	S.lang	second
hallo	de	de	Student

- Buffer:

first	R.lang	S.lang	second
hello	en	en	student
hello	en	en	collegian
hi	en	en	student
hi	en	en	collegian

Compute join result as cartesian product $\times$ of the buffer

- Input:

R

first	lang
hola	sp

S

lang	second
cn	daxuesheng

Merge Join 7/11

- Result $R \bowtie_{R.lang=S.lang} S$:

first	R.lang	S.lang	second
hallo	de	de	Student
hello	en	en	student
hello	en	en	collegian
hi	en	en	student
hi	en	en	collegian

Emit join rows

- Buffer:
- Input:

R

first	lang
hola	sp

S

lang	second
cn	daxuesheng

Merge Join 8/11

- Result $R \bowtie_{R.lang=S.lang} S$:

first	R.lang	S.lang	second
hallo	de	de	Student
hello	en	en	student
hello	en	en	collegian
hi	en	en	student
hi	en	en	collegian

- Buffer:

S

lang	second
cn	daxuesheng

Load rows with smallest values in join columns into buffer

- Input:

R

first	lang
hola	sp

Merge Join 9/11

- Result $R \bowtie_{R.lang=S.lang} S$:

first	R.lang	S.lang	second
hallo	de	de	Student
hello	en	en	student
hello	en	en	collegian
hi	en	en	student
hi	en	en	collegian

- Buffer: Discard buffer
- Input:

R

first	lang
hola	sp

Merge Join 10/11

- Result $R \bowtie_{R.lang=S.lang} S$:

first	R.lang	S.lang	second
hallo	de	de	Student
hello	en	en	student
hello	en	en	collegian
hi	en	en	student
hi	en	en	collegian

- Buffer:

  R

first	lang
hola	sp

  Load rows with smallest values in join columns into buffer

- Input:

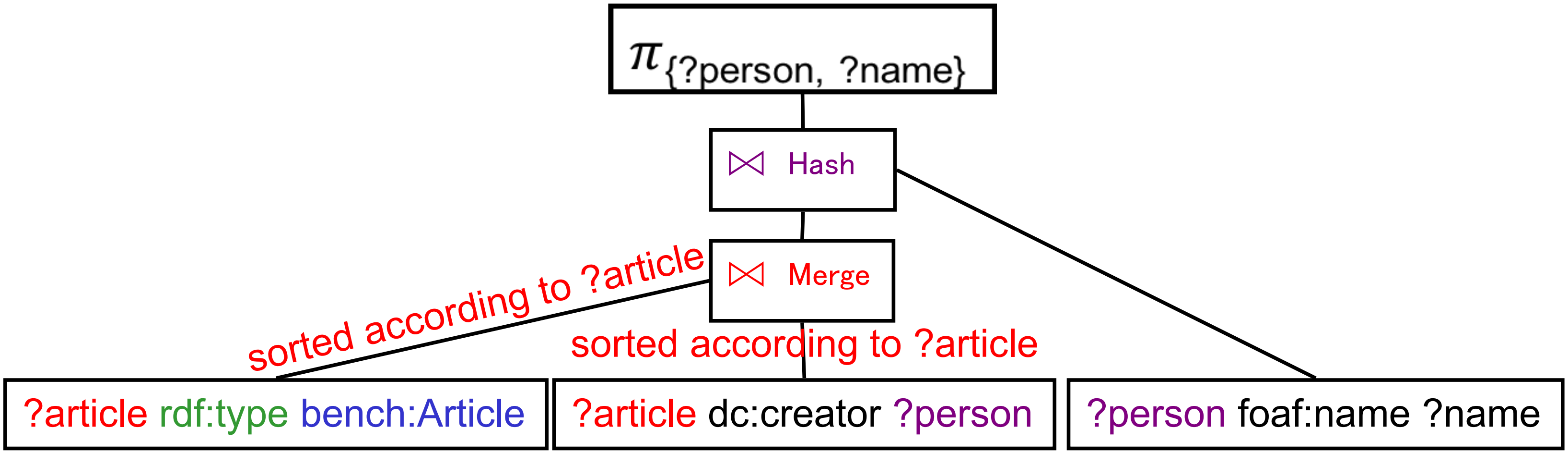
Merge Join 11/11

- Result $R \bowtie_{R.lang=S.lang} S$:

first	R.lang	S.lang	second
hallo	de	de	Student
hello	en	en	student
hello	en	en	collegian
hi	en	en	student
hi	en	en	collegian

- Buffer: Discard buffer
- Input:

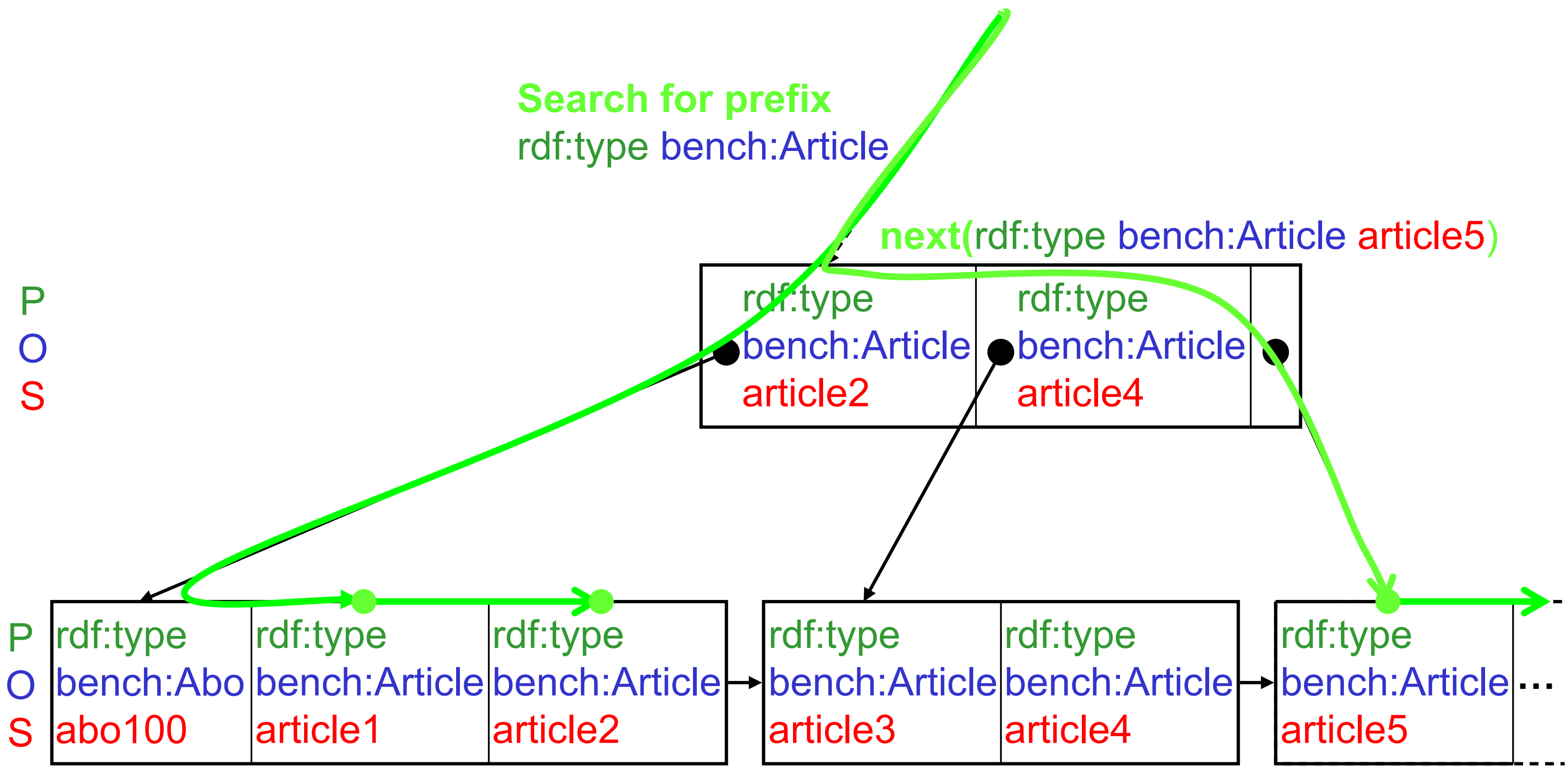
Merge Join



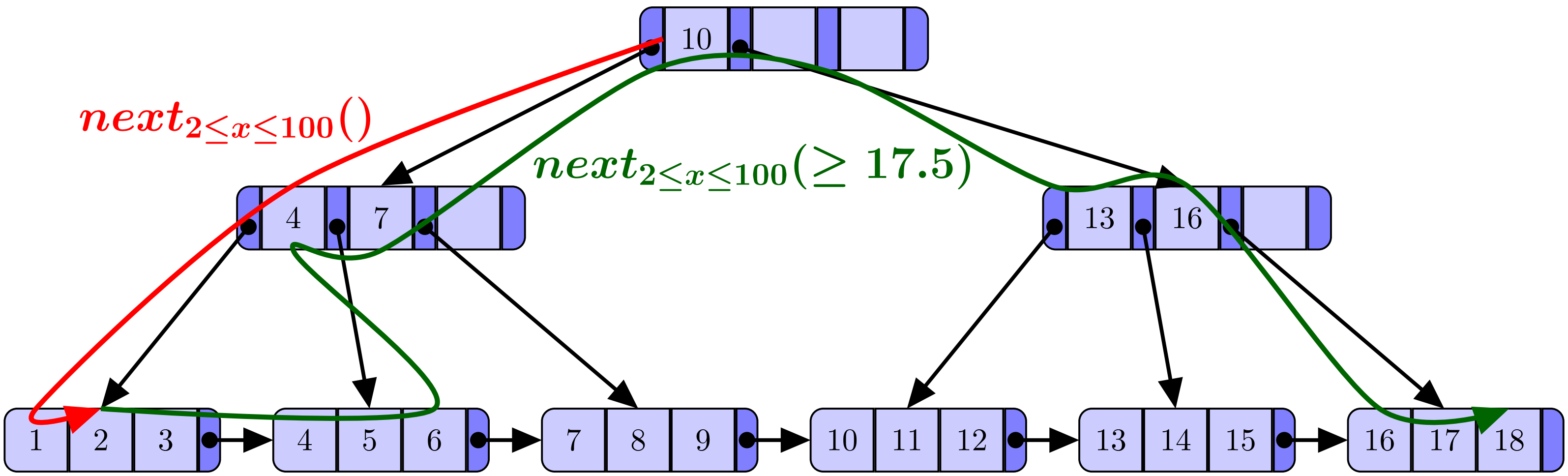
Search for prefix
dc:creator
in index sorted according to P S O:

?article = article1 = ?article = article1 ?person=p1
?article = article2 ?article = article2 ?person=p2
?article = article3
?article = article4
?article = article5 ?article = article5 ?person=p1
… …

Sideways Information Passing (SIP) in RDF3X B^{+}-Tree



SIP in B$^+$-Tree



- Too much overhead if the inner nodes are searched every time and the value to search for is close to the current one
- **Heuristic:** If the SIP value is not in the current and in the next leaf node, then search via the inner nodes (starting from the leaf node to the root via the previous search path)

Sideways Information Passing (SIP)

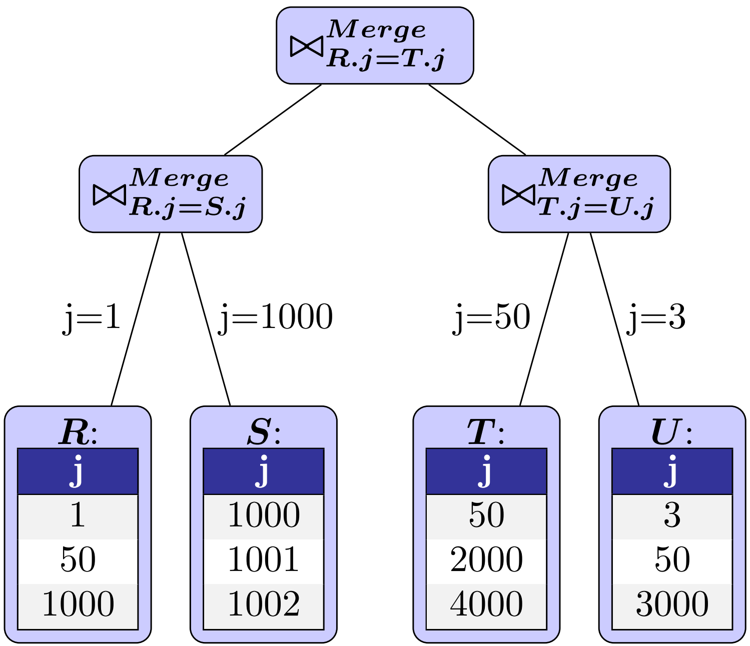
```
open() {                          close() {
  R.open(); r = R.next()            R.close()
  S.open(); s = S.next()            S.close()
  B = {} × {}; B.open()           }
}
```

Runtime Complexity with SIP
In most cases (in practice) $O(|R \bowtie S|)$

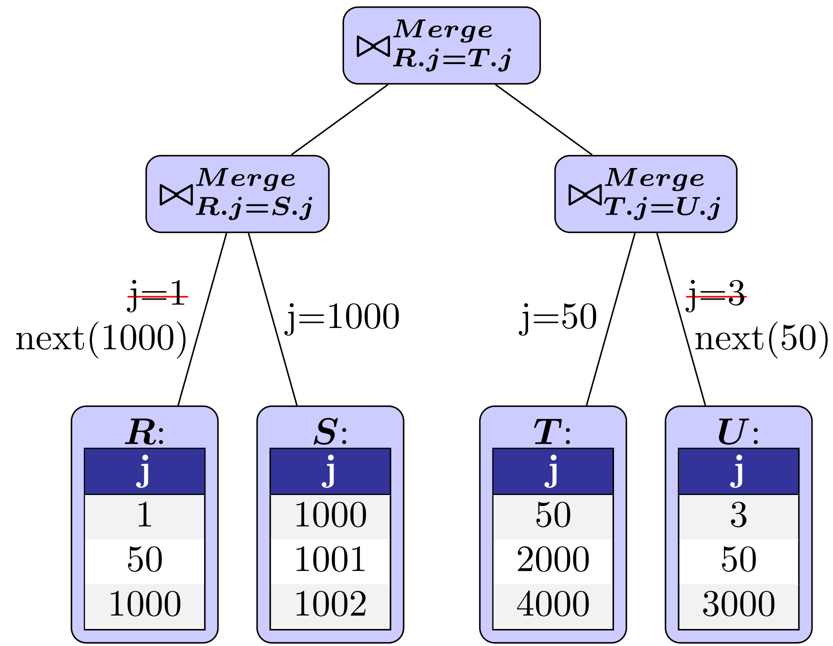
```
next( sip ) { // modified for SIP! Return result ≥ sip (not guaranteed)!
  IF(B.hasNext( sip )) RETURN B.next( sip )                    // SIP!
  IF(s == null ∨ r == null) RETURN null
  WHILE(πJoin Attributes(s) ≠ πJoin Attributes(r) ∨
        πJoin Attributes(r) < sip ∨ πJoin Attributes(s) < sip ){ // SIP!
    IF(πJoin Attributes(s) < πJoin Attributes(r)){
      s = S.next( max(πJoin Attributes(r), sip) )              // SIP!
      IF(s == null) RETURN null
    } ELSE {
      r = R.next( max(πJoin Attributes(s), sip) )              // SIP!
      IF(r == null) RETURN null
    }
  }
  cmp = πJoin Attributes(s) // = πJoin Attributes(r) (see WHILE-condition)
  ts = {}; WHILE(πJoin Attributes(s) == cmp) { ts = ts ∪ s; s = S.next() }
  tr = {}; WHILE(πJoin Attributes(r) == cmp) { tr = tr ∪ r; r = R.next() }
  B = ts × tr; B.open()
  RETURN B.next()
}
```

Sideways Information Passing (SIP) with several Merge Joins 1/2

1.

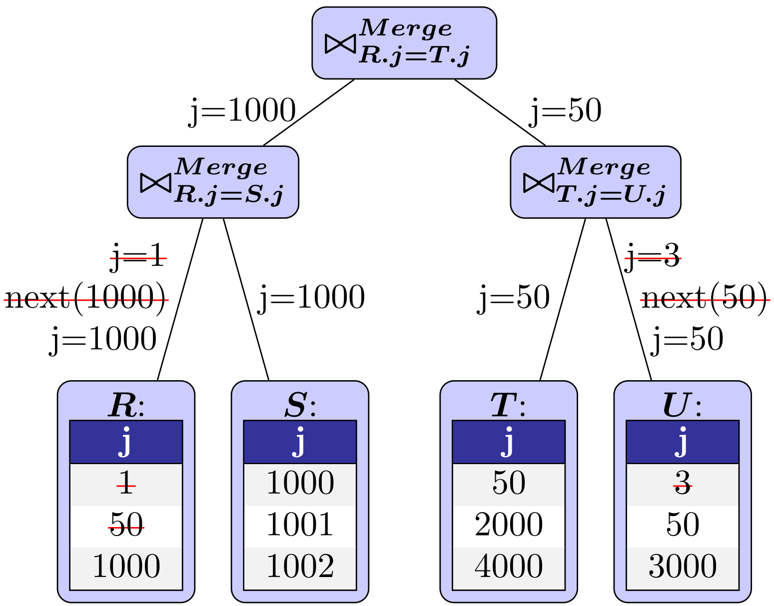


2.

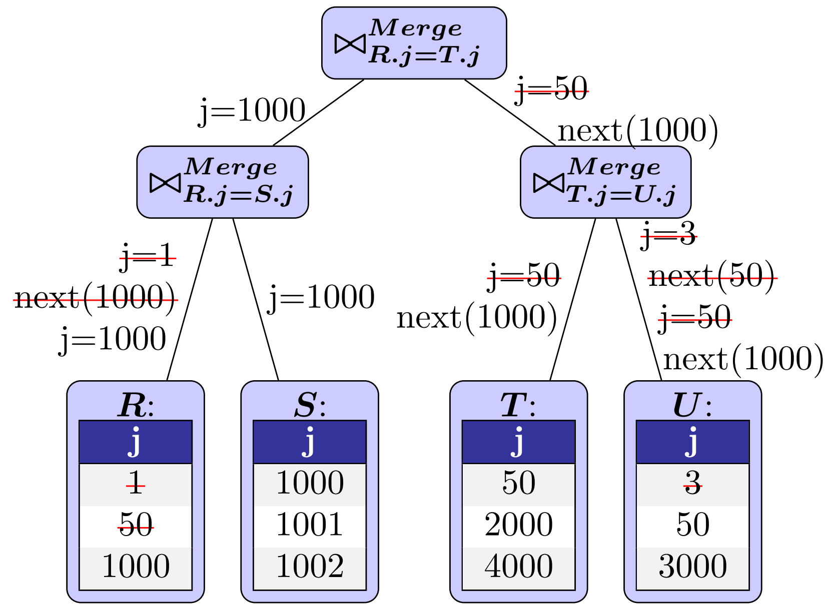


Sideways Information Passing (SIP) with several Merge Joins 2/2

3.



4.



- Information `next(1000)` passes sideways from S to R and also over the root node to T and U

RDF3X - Indexing Scheme for large-scale RDF triple stores

$\bowtie_{Merge(?car)}$

sorted by ?car

R	S
?car rdf:type ont:Car	data:Sven ont:Owns ?car

Maximise usage of **merge joins**
⇒ 3!=6 indices for **all possible sort orders** of triples SPO:
SPO, SOP, PSO, POS, OSP, OPS

	R	S
Prefix-Search in **Index**:	PSO	SPO
with (Prefix-)**Key**:	rdf:type ont:Car	data:Sven ont:Owns

Result of Triple Pattern:

?car	?car
HL 100	**HL 999**
HL 101	
…	
HL 999	
…	

?

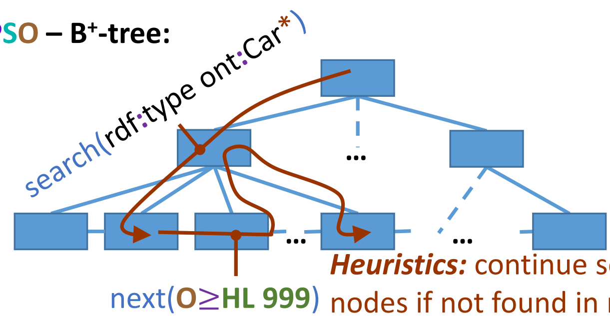
next(?car≥HL 999)

„Sideways **Information** Passing“

Complexity of Merge Join $\bowtie_{Merge}$:

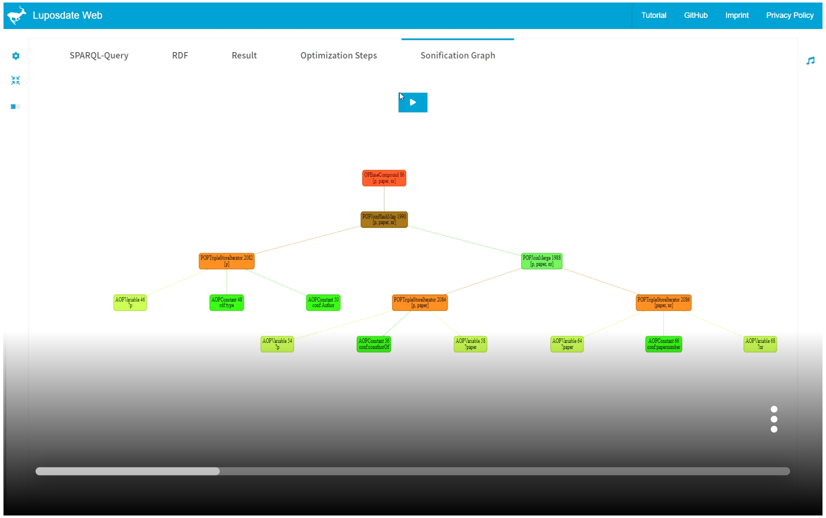
Worst Case (duplicates): $O(|R| \times |S|)$
without duplicates: $O(|R| + |S|)$
with **sideways information passing**: $O(|R \bowtie S|)$
(assuming quasi-constant access in B⁺-tree)

Search in PSO – B⁺-tree:



Heuristics: continue search in inner nodes if not found in next leaf node!

(In-Memory) Hash Join/Index Join with Hash Table

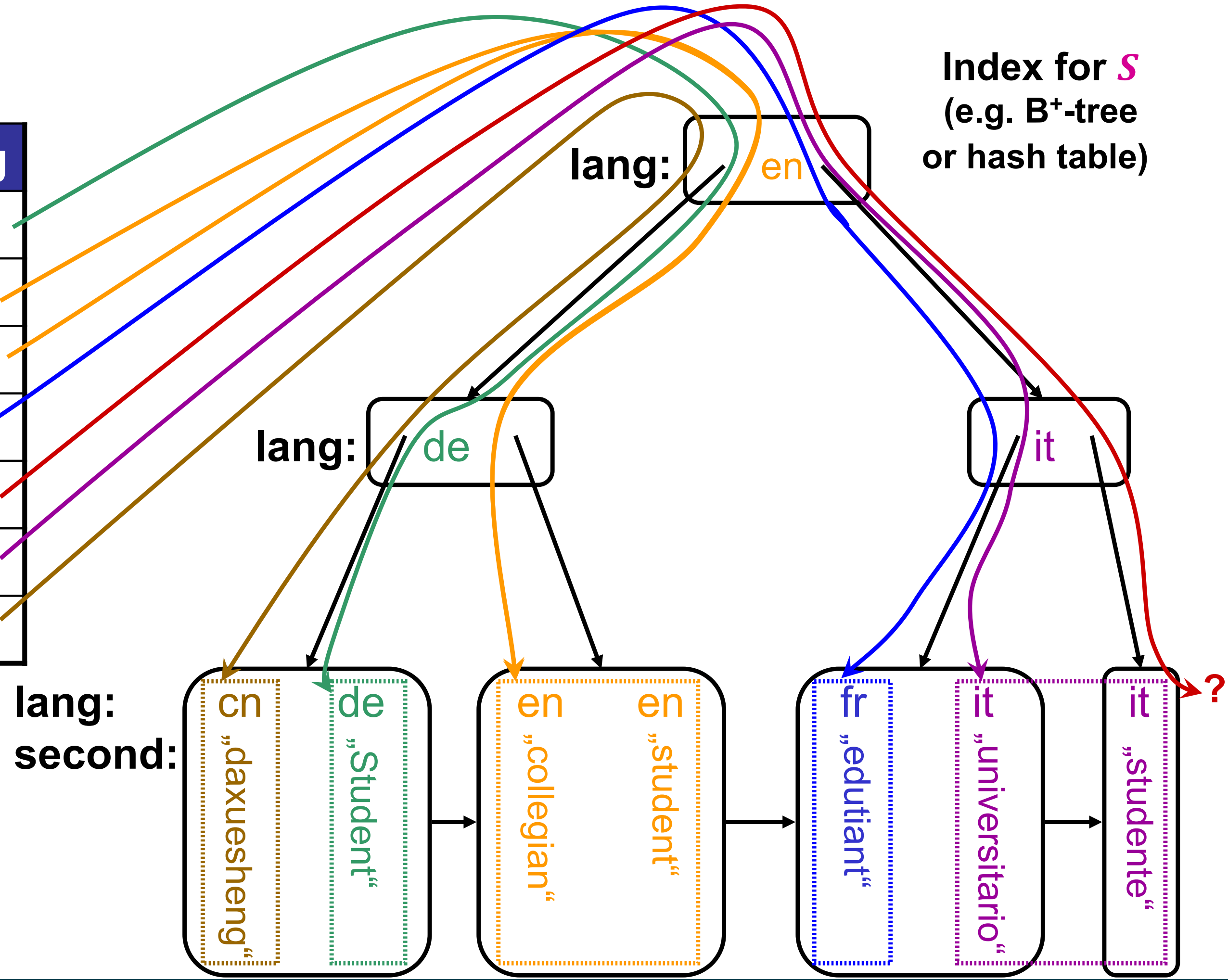


Start Sonification

Index Join 1/2

R

first	lang
„hallo“	de
„hello“	en
„hi“	en
„salut“	fr
„hola“	sp
„ciao“	it
„nihao“	cn



Index Join 2/2

- Runtime $O(Indexing(S) + |R| \times Lookup(Index(S)))$
  - $Indexing(S)$ is omitted in case of existing (primary/secondary) indices

	$O(Indexing(S))$	$O(Lookup(Index(S)))$
Hashtable	$O(\|S\| \times h(max(\|\pi_{key}(s)\| \| s \in S)))$	$O(h(max(\|\pi_{key}(s)\| \| s \in S)))$
B$^+$-Tree	$O(\|S\| \times log(\|S\|) \times max(\|\pi_{key}(s)\| \| s \in S))$	$O(log_k(\|S\|))$ respectively $\approx O(1)$ for large k (Practice!)

With long keys, the time for hashing, comparing, etc. is significant, otherwise $O(1)$

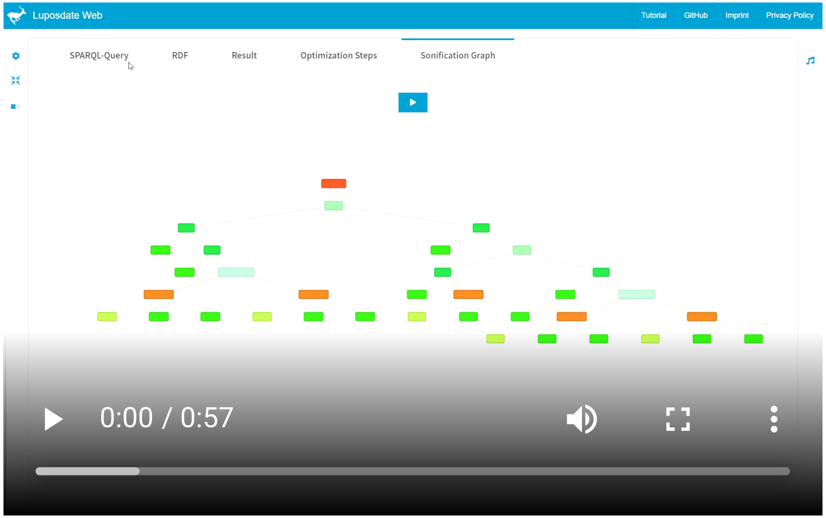
h Runtime for hash function (often linear to key length), k Out degree B$^+$-tree node

B$^+$-tree can be created after sorting by $1\times$ traversing the sorted entries

Summary and Conclusions

- SPARQL Query Language
  - Transformation to Relational Algebra Expression/Operatorgraph
- Sound of Databases
- RDF3X Indices
- Merge Join
- Sideways Information Passing (SIP) in
  - Merge Joins
  - RDF3X B^+-Tree
- Hash Join
- Inspiring Sounds

Whole Pieces of Music from your Database



[Start Sonification]