New ways of visualizing and querying RDF data

Jiří Dokulil

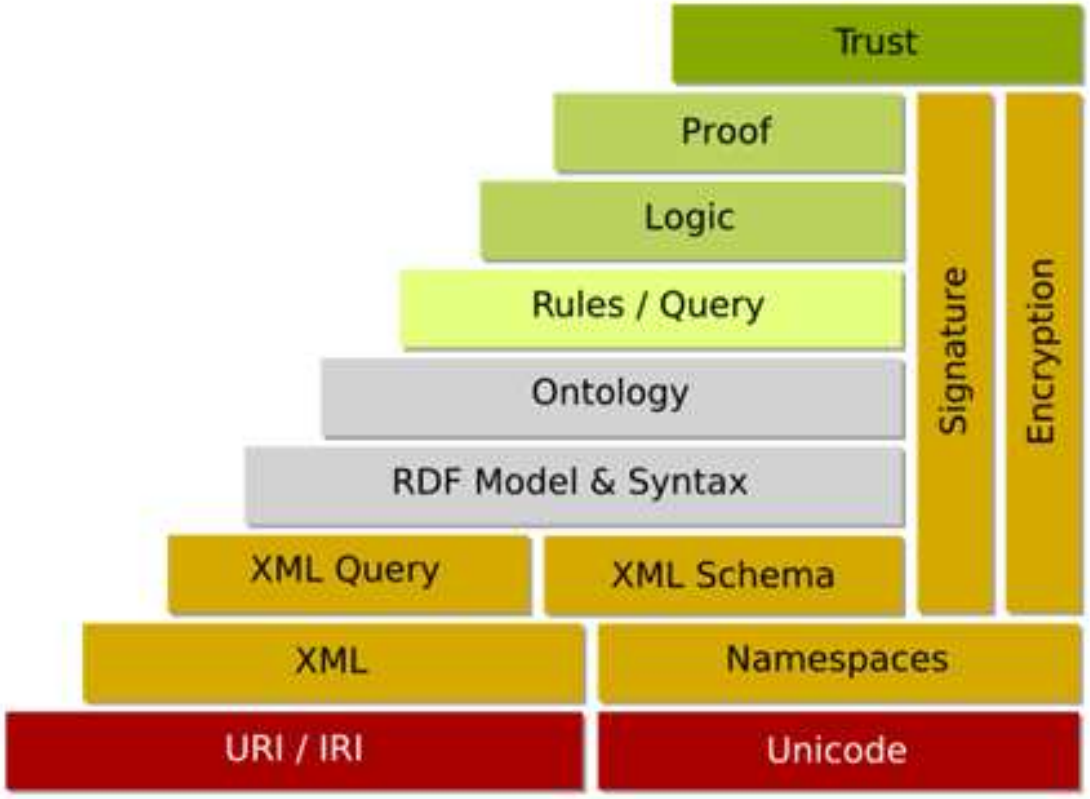
KSI MFF UK

Outline

- RDF
- RDF query life cycle
- RDF visualization
- TriQuery
- Bobox
- Conclusion

RDF

- part of the Semantic Web



RDF

- statements about resources
- triples
  - subject, predicate, object
- labeled, directed multi-graph
- abstract model
  - serialization formats
    - RDF / XML
    - N3
    - Turtle
    - …
- further standards built on top of RDF
  - RDFS, OWL
    - outside scope of this thesis

RDF query life

- idea
  - what does the user want to achieve
  - what do the inputs look like
  - what should the result look like
- query formulation
  - textual representation of the intended transformation within constraints of a query system
- query execution
  - correct and efficient execution

RDF query life – our work

- idea — RDF visualizer
  - what does the user want to achieve
  - what do the inputs look like
  - what should the result look like
- query formulation — TriQuery
  - textual representation of the intended transformation within constraints of a query system
- query execution — Bobox
  - correct and efficient execution

RDF visualization

- support RDF-enabled software development
  - display data in a compact way
  - display “raw” RDF data
  - handle large data sets
- visualization algorithm
  - triangle layout
  - edge routing
  - node merging
- supplementary techniques
  - navigation
  - animation
  - …

RDF visualization – example

RDF visualization – remarks

- node merging could be adopted for most other RDF visualizers and improve their performance
- usability of triangle layout depends on node merging
  - wide but low rectangles would result in a lot of wasted space
- triangle layout requires quadratic area
  - optimal for rooted trees with layered drawing
- many technical details (see the thesis)
  - special views for specific situations (e.g. reifications)
  - limitation on reasonable node and handling of overflows
  - implementation – user interface, architecture
  - …

TriQuery

- XQuery extension
  - why XQuery?
    - powerful, yet relatively simple
    - well standardized
    - we are already developing an XQuery engine
    - RDF is closely related to XML in some aspects
      - XSD data types
      - RDF/XML serialization format
  - new language constructs
    - extension of the XQuery grammar (14 new or changed rules)
    - TriQuery queries are not valid XQuery queries
    - using pure XQuery to handle RDF has been tried, but is hard to use due to long expressions with many function calls

Records in XQuery

- introduction of records
  - the only modification of XQuery grammar and evalueation
- record is a structured value
  - identifier $\Rightarrow$ value
    - identifier – qualified name or number
      - names and numbers cannot be mixed
      - names $\Rightarrow$ named record
      - numbers $\Rightarrow$ anonymous record
    - value – sequence of values
      - may not contain records – no nested records are allowed

Records – language constructs

- constructor

```
[8, "Hello, world!", $x]
[eight := 8, hello := "Hello, world!", var := $x]
```

- dot – field access

```
let $x=[ 1 ]
let $y=[ one := 1 ]
return ($x.1, $y.one)
```

- (natural) extension of existing operators
  - sequence concatenation, equality testing, node comparators, FLWOR

Records – signature

- signature of a record is the set of keys
  - `[one := 1, two := 2]` has signature {1,2}
- signature of a sequence
  - if sequence contains a record, it must contain nothing but records
  - if sequence contains a record, all must have the same signature
  - the signature of a sequence is the signature shared by the records in the sequence, or empty set if no records are present

Records – signature cont.

- signature of an expression
  - the signature of the sequence returned by the expression
  - can be determined statically by examining a query
    - all language construct are defined to allow this
    - improper use can be detected during query compilation

```
$x = [ one := 1 ]
$y = [ uno := 1 ]
return ($x, $y)
```

      - invalid – $x and $y have different signatures, the result of concatenation would violate the restriction of sequences of records

Records – pattern matching

- searches for a specific pattern in a sequence
- **`<expression> match ( <pattern> )`**
- **`expression`** is TriQuery expression
  - must provide sequence of anonymous records
- **`pattern`** is a set of n-tuples
  - n is the size of the signature of the sequnce
  - values may be expressions or a variable
- result is a sequence of named records
  - names are the names of variables used in the pattern
  - for each possible variable mapping that transforms the pattern into a subset of the input, one record (wioth the corresponding variable mapping) is added to the result sequence

Records – pattern matching cont.

- basic example

```
for $r in $x match{ "John" {1+2} ?a . ?a ?b ?c}
return
  <r><a>{$r.a}</a><b>{$r.b}</b><c>{$r.c}</c></r>
```

- RDF example

```
let $c := triq:doc("data1")
return $c with triq:RDFS match
   ( $x ex:first-name $fn . $x ex:last-name $ln )
```

  - uses the optional **`with`** modifier
    - the pattern matching may then perform any transformation defined by the implementation, providing that the signature of the output is preserved

RDF support

- based on records
- no further extension to XQuery
  - only a set of RDF-related functions
- RDF data set represented as a set of anonymous records with three fields
- pattern matching
  - TriQuery pattern matching similar to basic graph patterns in SPARQL or SeRQL

RDF support – examples

- easy conversions from RDF to XML and back

```
let $d := triq:doc("people.rdf")
for $x in $d match ( $x ex:id $id .
  $x ex:name $n . $x ex:mail $m )
return <person id="{$x.id}">
  <name>{$x.name}</name>
  <email>{$x.mail}</email></person>

let $d := fn:doc("in.xml")
for $x in $d//item
return [ $x/id, ex:name, $x/name],
  [ $x/id, ex:price, $x/price]
```

RDF support – examples cont.

- construction of new RDF statements

```
let $d := triq:doc("people.rdf")
for $x in $d match ( $person ex:age $age )
where $x.age>=13 and $x.age<=19
return [$x.person, ex:age-group, ex:teenager]
```

- matching with RDFS entailment
  - like normal match, but the input data is extended with statements that can be inferred using RDFS

```
let $d := triq:doc("people.rdf")
for $x in $d with triq:rdfs match ( $person
  rdf:type ex:person )
return $x.person
```

RDF support – examples cont.

```
fn:count(
  (
    let $r := triq:doc("people.rdf"),
    let $i :=
      (
        for $n in fn:doc("input.xml")//father
        return [ father/id, rdf:type, ex:father ]
      )
    return $r,$i
  )
  with triq:rdfs match ( $person rdf:type ex:man )
)
```

RDF examples (SP^{2}Bench Q1)

```
SELECT ?yr
WHERE {
  ?journal rdf:type bench:Journal .
  ?journal dc:title
  "Journal 1 (1940)"^^xsd:string .
  ?journal dcterms:issued ?yr
}

for $x in triq:doc("dblp") match (
$journal rdf:type bench:Journal .
$journal dc:title "Journal 1 (1940)" .
$journal dcterms:issued $yr )
return $yr
```

RDF examples (SP^{2}Bench Q9)

```
SELECT DISTINCT ?predicate
WHERE {
  {
    ?person rdf:type foaf:Person . ?subject ?predicate ?person
  } UNION {
    ?person rdf:type foaf:Person . ?person ?predicate ?object
  }
}

return fn:distinct(
  let $x := triq:doc("dblp") match (
    $person rdf:type foaf:Person.
    $subject $predicate $person )
  let $y := triq:doc("dblp") match (
    $person rdf:type foaf:Person.
    $person $predicate $object }
  return $x.predicate, $y.predicate
)
```

- notice the use of filed access operator on sequence of records: **`$x.predicate`**

TriQuery vs XSPARQL

- both extensions of XQuery to handle RDF
- XSPARQL
  - DERI, W3C Member Submission, January 2009
  - inclusion of SPARQL, “glue”
  - mixes languages with different paradigms
    - XQuery – closed, full compositionality, expressions
    - SPARQL – `CONSTRUCT`, no compositionality, SQL-like constructs
  - can be implemented by combining XQuery and SPARQL engines
    - query rewriting minimizes changes to XQuery engine
- TriQuery
  - records
  - more general, not only RDF
    - semantics not tailored to exactly fit for RDF
  - requires significant changes to XQuery engine or completely new engine
    - allows optimizations across XML – RDF border

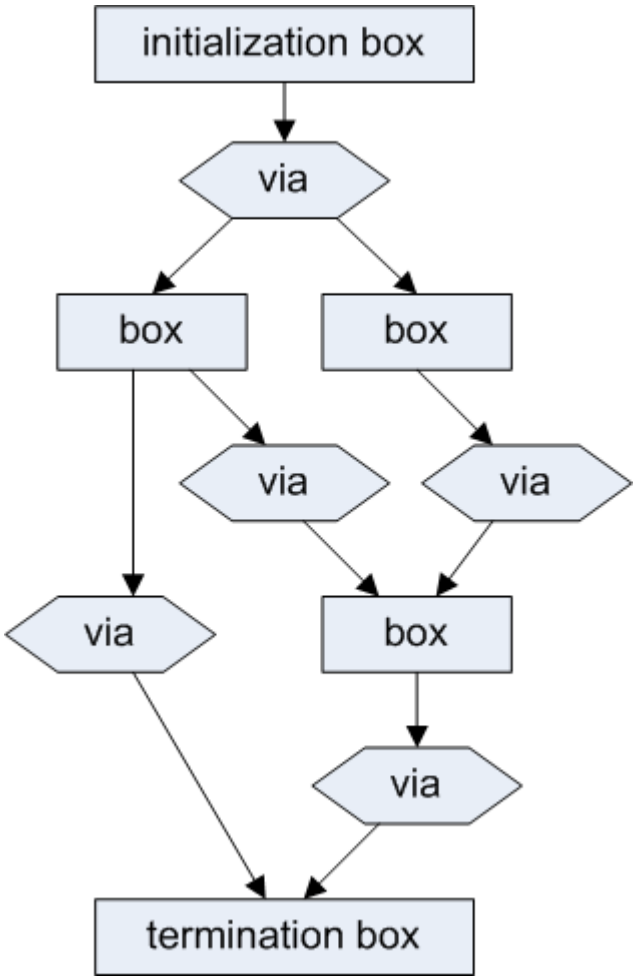
Bobox

- parallel computation framework
- one class of problems
  - many computational components connected to form a non-linear pipeline
  - data-intensive
- independent on query language and data representation
- execution control handled by the framework
  - unlike traditional parallelization libraries
  - user specifies the way the pipeline is connected
  - execution is controlled by the flow of the data

Bobox model

- definition of the structure of the pipeline
- usually an execution plan of a query

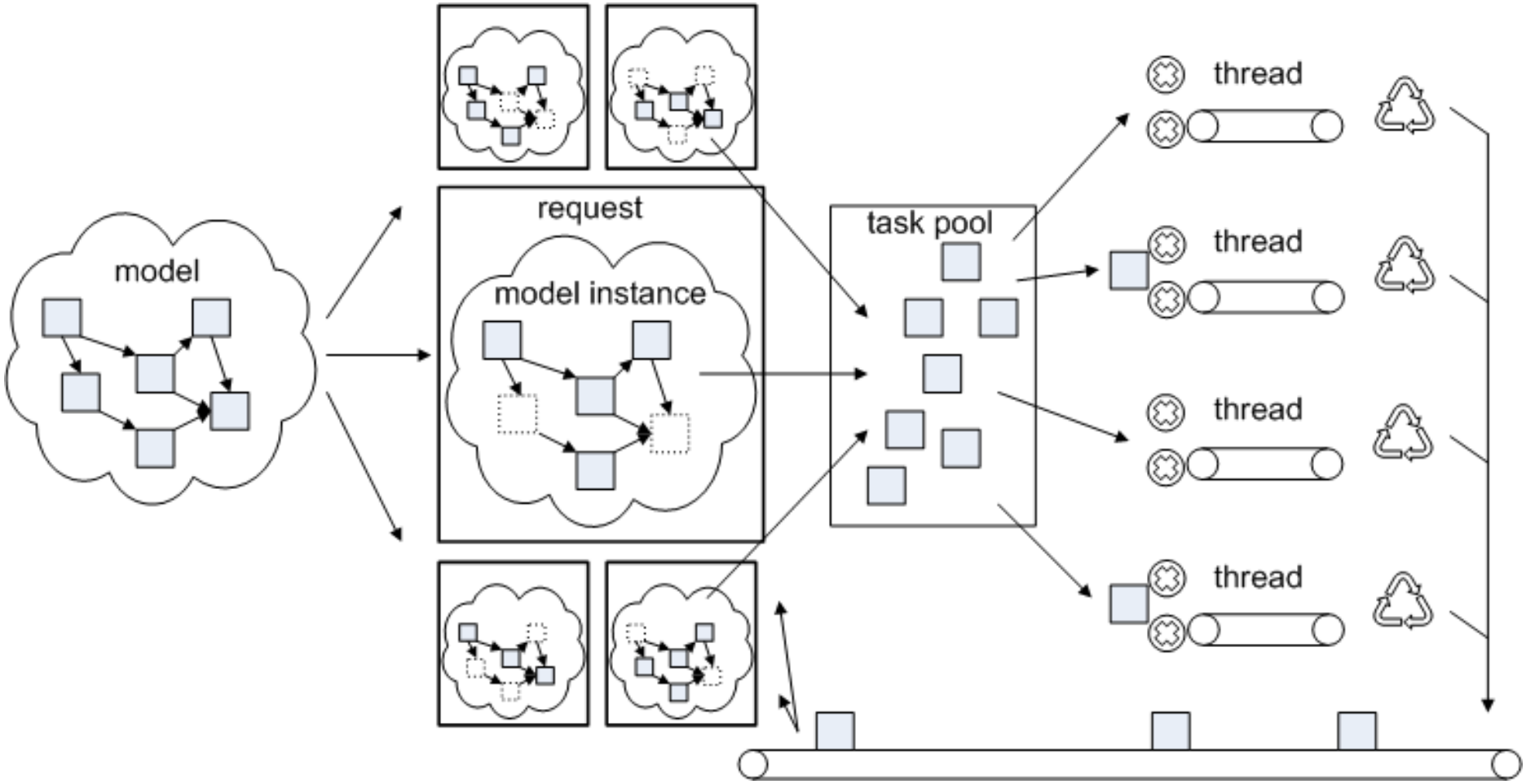
- boxes
  - components that perform the actual computation
- vias
  - links between boxes
  - control data flow



Execution

- model instantiated
  - model instance looks like the model, contains actual code
- task level parallelism
  - task
    - unit of work (data and algorithm) to be executed in parallel
    - box and input data
    - placed into task pool when ready to execute
  - thread pool
    - fixed number of execution threads that execute the tasks from the task pool

Execution – overview



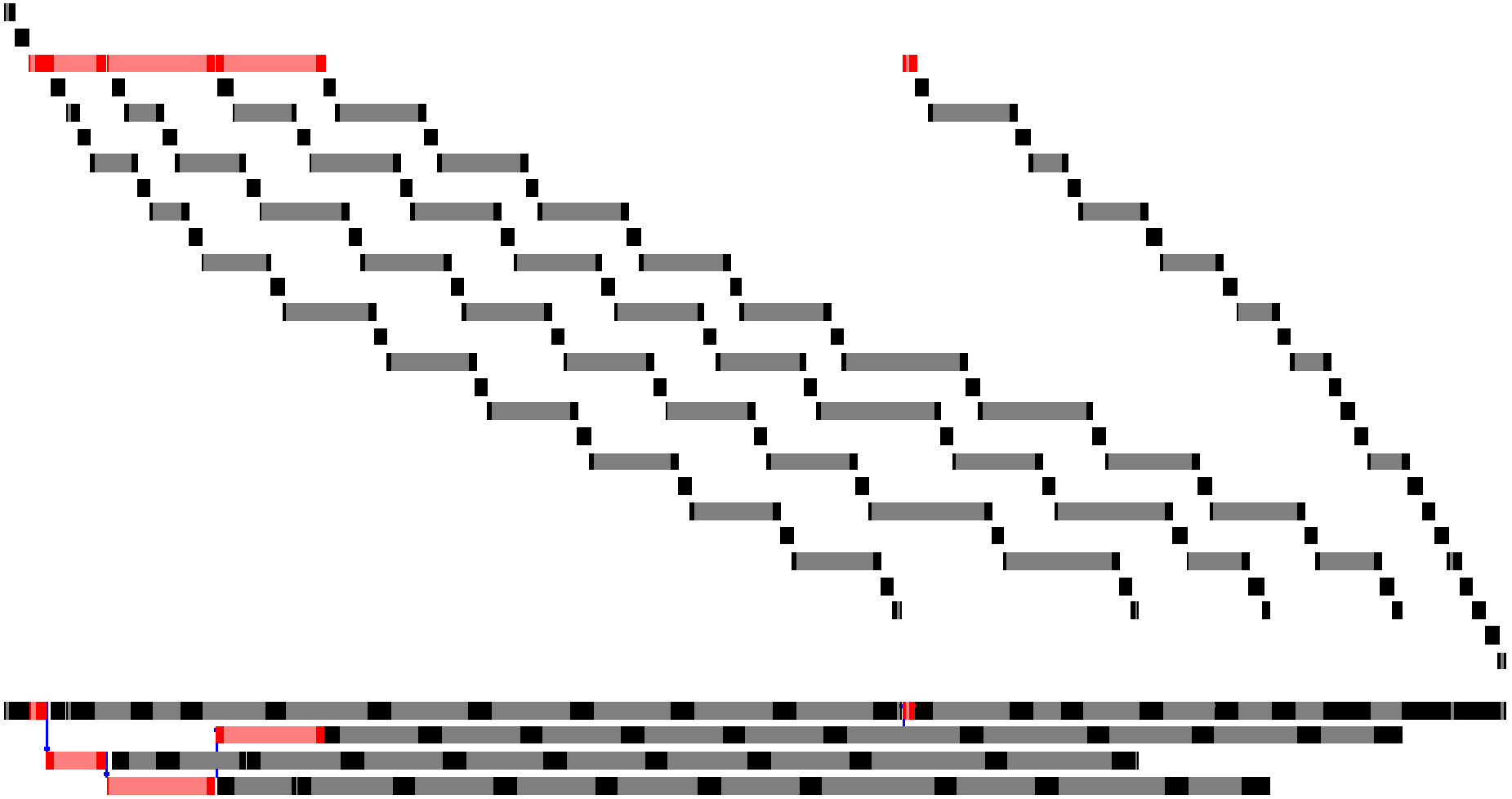
Bobox – the data

- the data is passed along the pipeline
- envelope
  - a unit of data to be passed
  - similar to a table in a column oriented DBMS
    - data for each column is stored separately in a continuous block of memory
    - compared to each row being stored as a record and table being a sequence of such records
    - allows data level parallelism, e.g. SSE instructions
- poisoned pill
  - special kind of envelope
  - when poisoned pill passes through a certain point, it guarantees no further envelopes will pass it

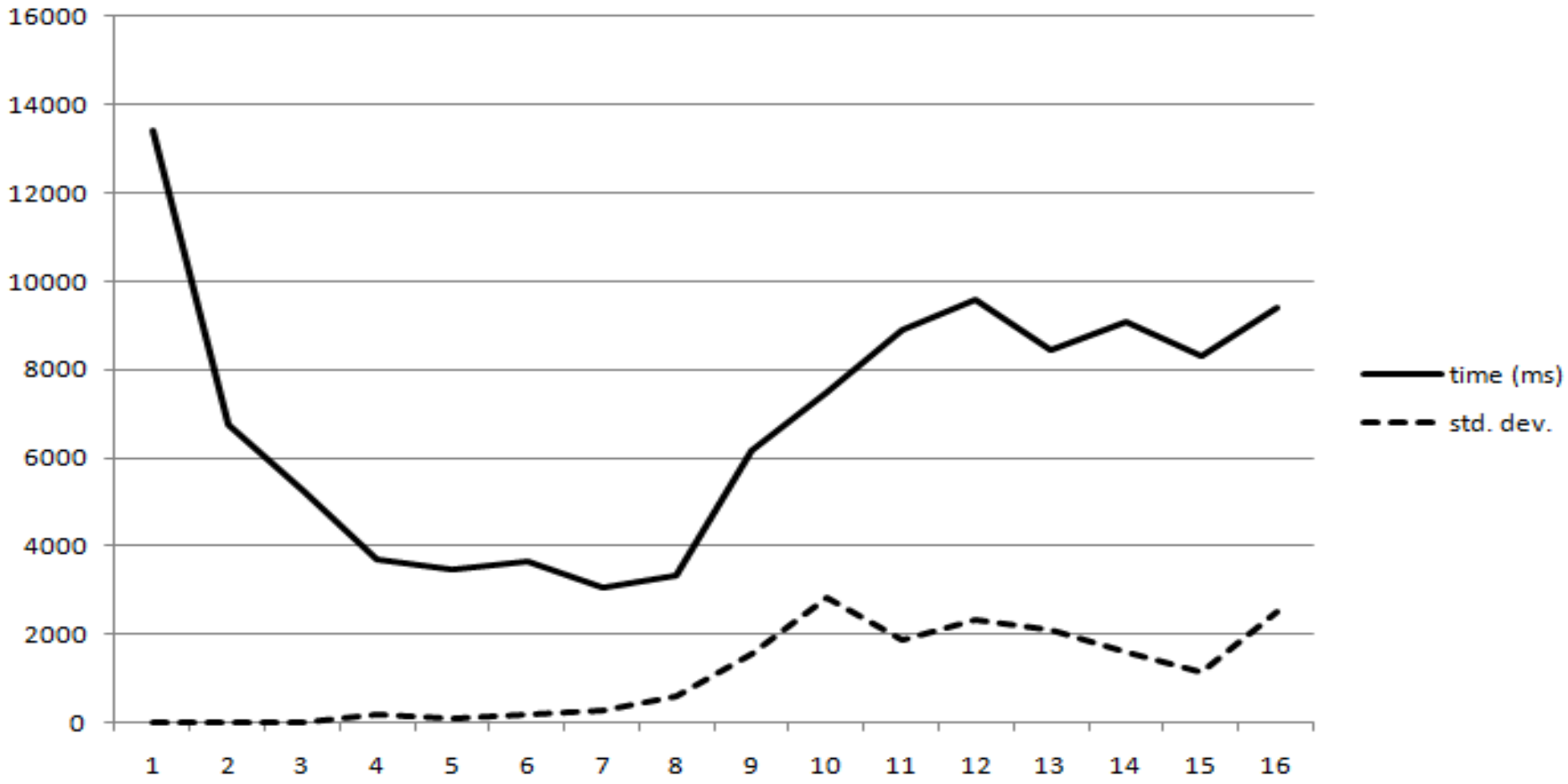
Bobox – experiments

- an experimental implementation of key Bobox components
  - can execute individual queries
  - no physical data store, only in-memory database

Execution of a simple pipeline



Performance on multiple CPU cores

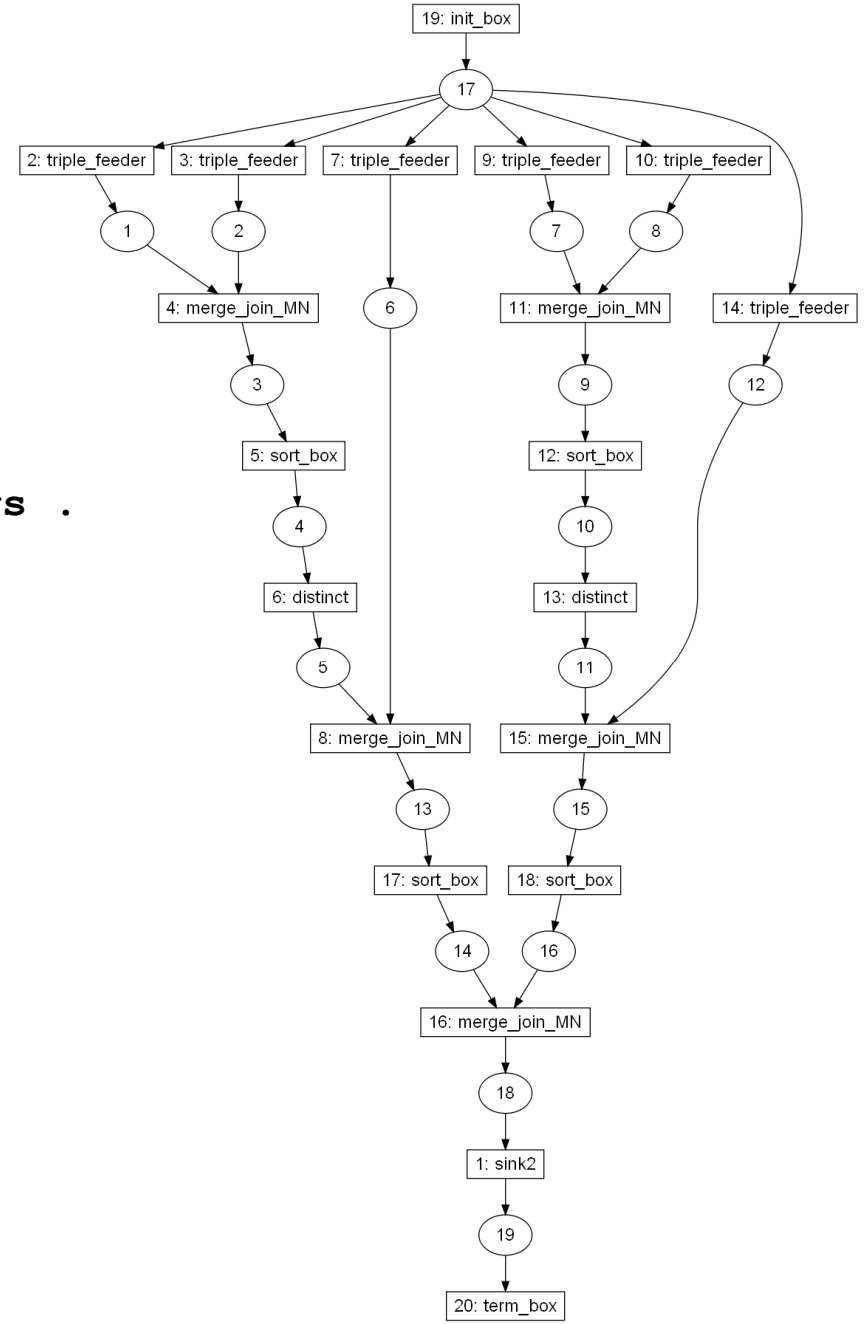


4 physical CPU cores, hyperthreading (8 logical CPU cores), CPU intensive operations with floating point arithmetics

SP2Bench Q5b

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

- Bobox 486ms
  - no compiler (yet)
  - 4 CPU cores
- Sesame 683ms
  - slightly more stable results
  - limited use of parallelism
    - can outperform current Bobox implementation on single core



Conclusion

- contributions to several steps of an RDF query life cycle
- graph drawing algorithms (triangle layout, edge routing)
- extension of XQuery to allow easy inclusion of RDF handling
- Bobox parallel framework
  - task and data level parallel computation
  - XQuery
  - SPARQL
  - other DBMS and data processing

Future work

- new implementation of RDF visualizer
  - currently uses SDL libraries for GUI
    - poor performance for animations in windowed mode
- implementation of TriQuery
  - reference implementation
  - Bobox implementation
    - extension of the XQuery engine
- improvements to Bobox
  - new scheduler and memory allocator
  - run-time modification of model instances
  - distributed execution

The End

Thank you for your attention!