Part II:
Linked Stream Data Processing - Building a Processing Engine

Danh Le-Phuoc, Josiane X. Parreira, and Manfred Hauswirth
DERI - National University of Ireland, Galway

Reasoning Web Summer School 2012

Outline

- Part I: Basic Concepts & Modeling (Josi)
  - Linked Stream Data
  - Data models
  - Query Languages and Operators
  - Choices/Challenges when designing a Linked Stream Data processor
- Part II: Building a Linked Stream Processing Engine (Danh)
  - Analysis of available Linked Stream Processing Engines
    - Design choices, implementation
    - Performance comparison
    - Open Challenges

“Why should I care?”

- As an application developer
  - What can I do with it?
  - How does it work?
  - How to choose one?
- As a processing engine developer
  - How to build one?
  - How to build/identify a better one?
- As a researcher
  - What have been done? What left?
  - Is there any interesting research problem?
  - How to find room to improvement?

Why a continuous query processing engine is needed?

Separation of concerns

- Focus of application logic
- Let the experts deal with data operations on streams
- Minimize the learning efforts
  - Learn simple APIs using the engine
  - Learn a simple query language

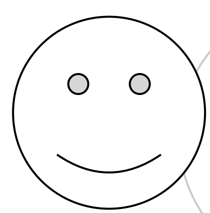
How a stream-based application is built with a stream processing engine?

① Initialize the engine (less than 5 lines of code)

② Write and register the queries to the engine (1 line for 1 query)

③ Write codes for wiring output streams to the application logic (depends on the application logic but the each wiring code snippet ≈5 lines of code)

④ Connect input streams to engine (1-3 lines for each stream)

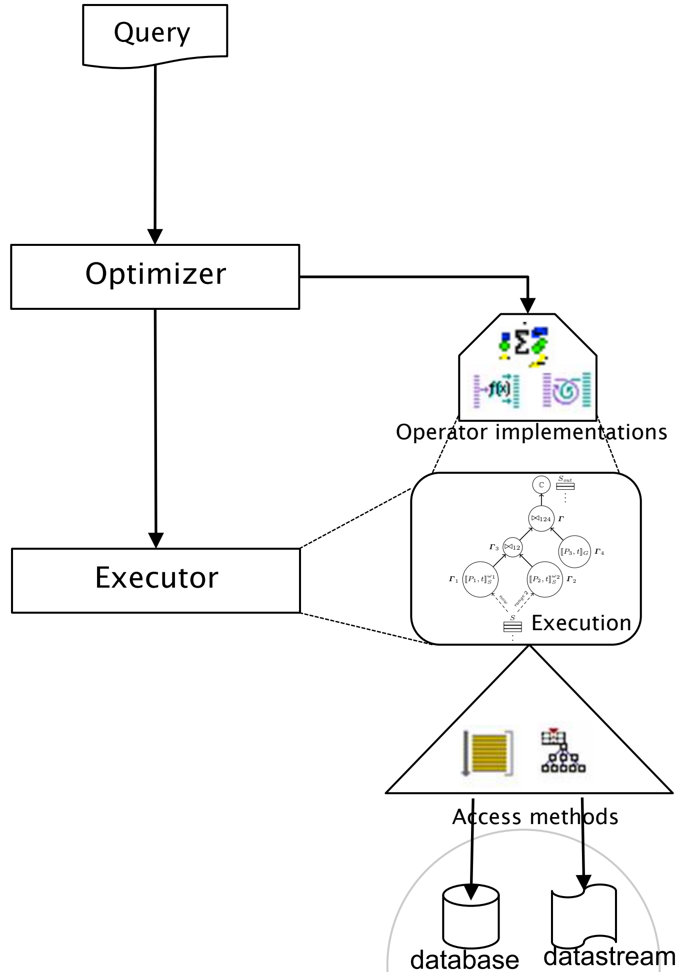


Just 10-20 lines of code!!!

What need to be done to build a processing engine?

- Data model : relational, object-oriented, etc
- Query model :
  - Logical operators : sliding windows, relational algebras
  - Query language: CQL, C-SPARQL,CQELS,etc
- Build a processing engine
  - Handling input streams
  - Implement the execution engine
  - Schedule the executions
  - Optimization

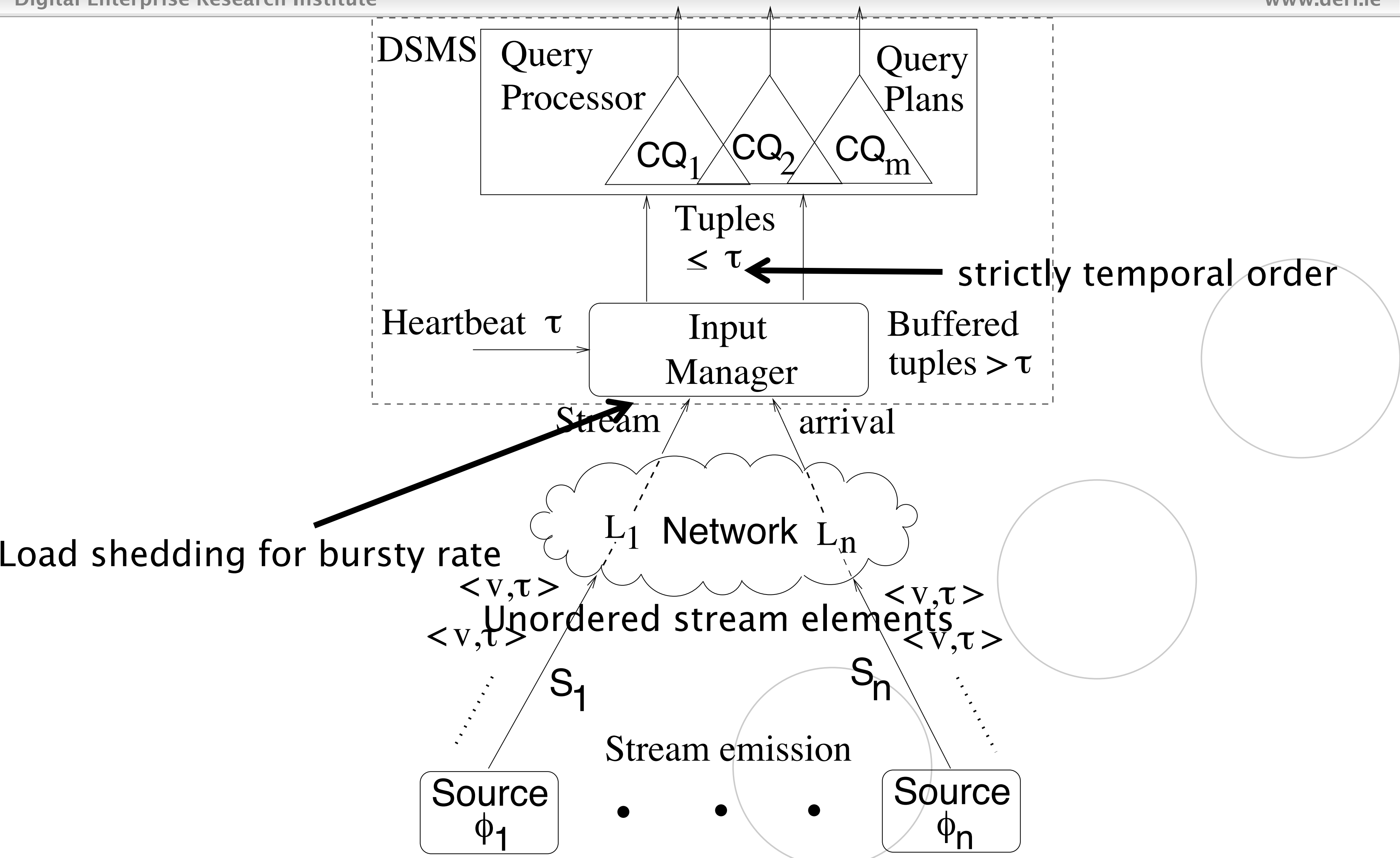
Building blocks of a query processing engine



Algorithms/technologies/solutions for stream processing

- Handling live and push-based data stream sources
  - Time management
  - Load shedding for bursty streams
- Operator implementation for execution engine
  - Data structure and physical storage
  - Handling the new stream elements/expired ones
  - Incremental execution
  - Memory overflow
- Optimization
- Scheduling

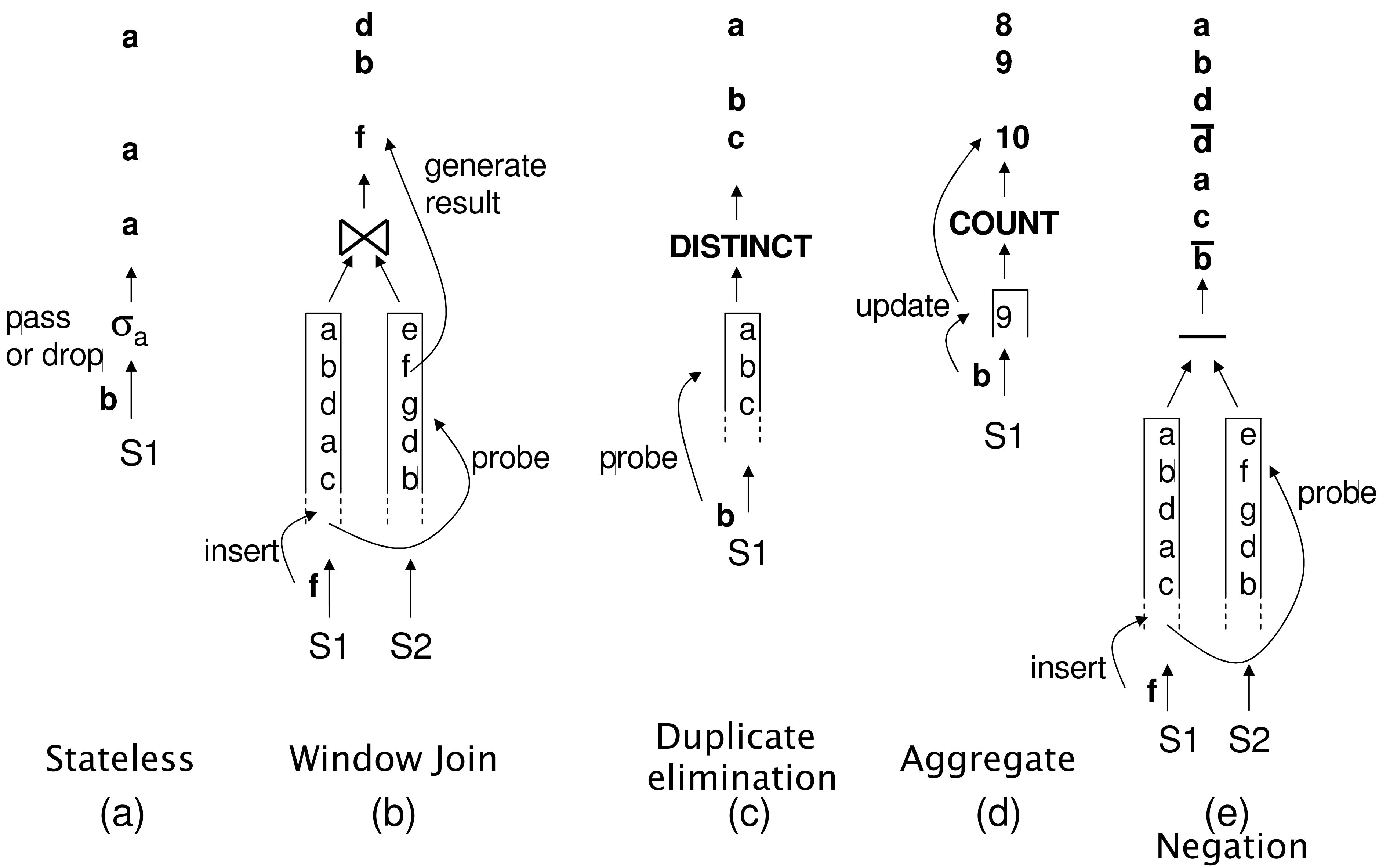
Handling input streams



Execution engine & Operator implementation

- Data structure and physical storages for high-update-rate processing buffers
- Handling the new data stream elements/expired ones
- Operators && Incremental execution
  - Stateless
  - Stateful
    - Duplicate elimination
    - Window Join
    - Negation
    - Aggregation
- Memory overflow
- Dynamic Optimization of the continuous execution
- Schedule execution for fluctuate execution settings

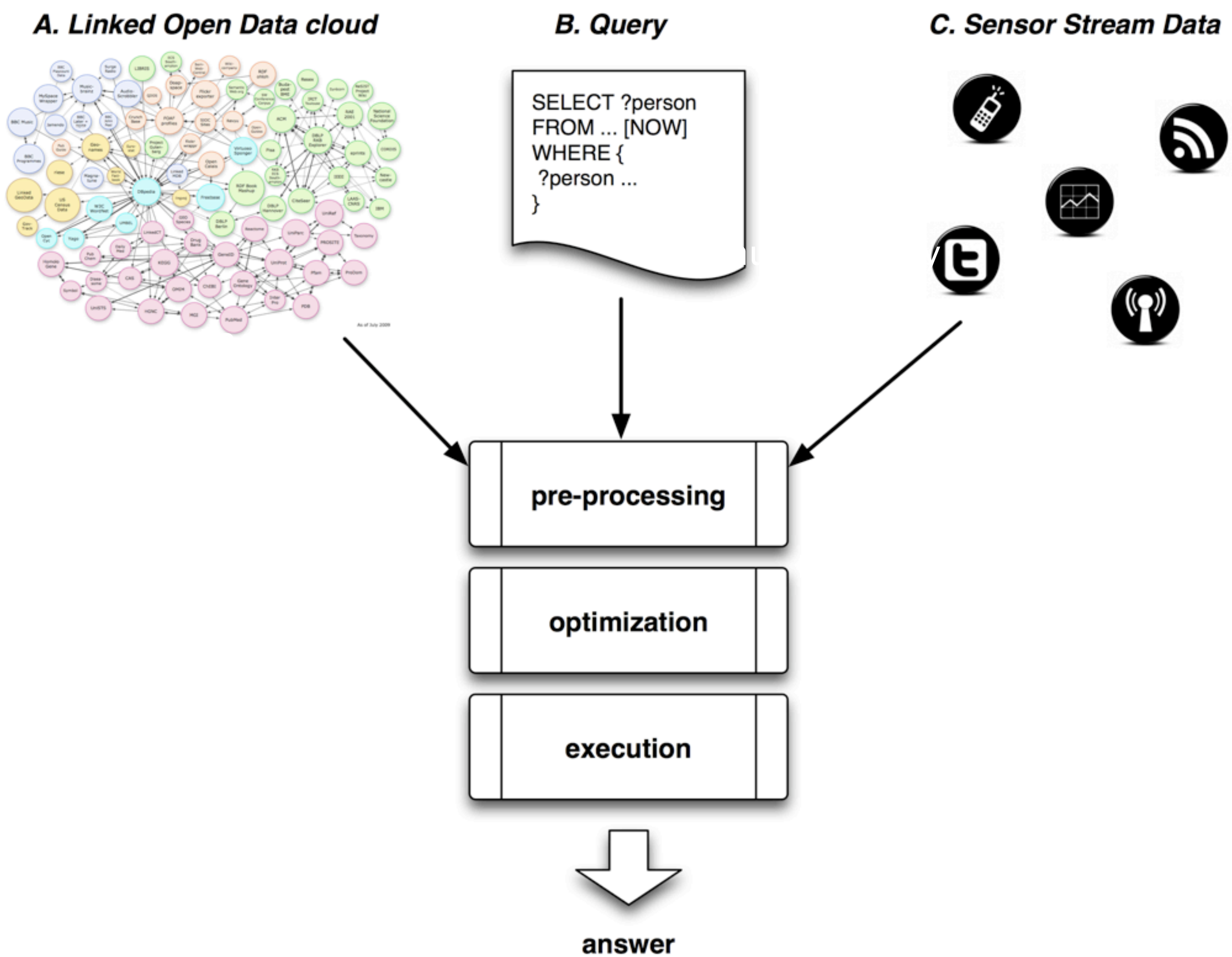
Incremental execution of windowing operators



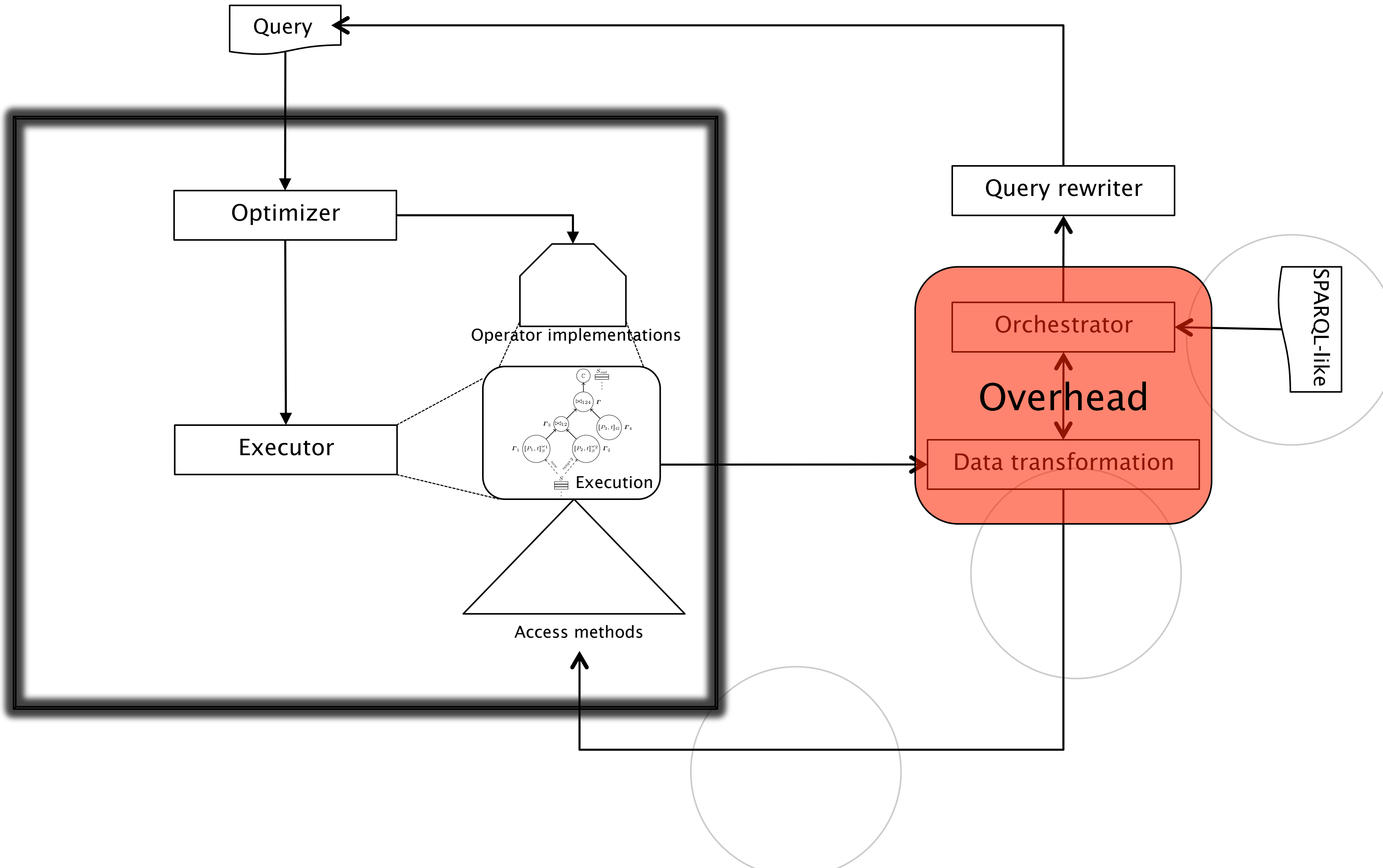
What have “I” learned?

- 12-15 years of techniques/algorithms/solutions for general stream processing (DSMS)
- Only few prototypes and commercial products
  - STREAM ,Borealis/Aurora, etc
  - StreamBase, IBM InfoSphere streams, etc
- Don’t take for granted!!!
- DSMS is not mature as DBMS(>40 years)

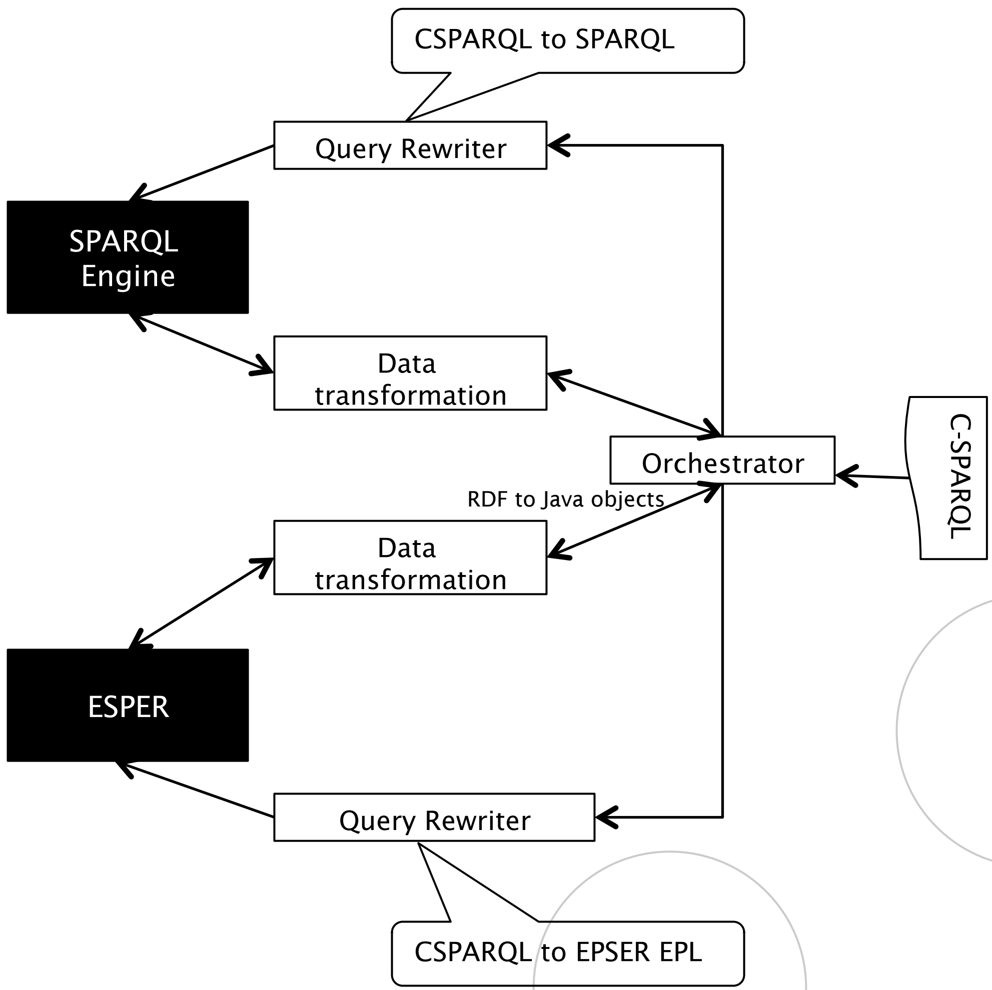
Processing Linked Stream Data In A Nutshell



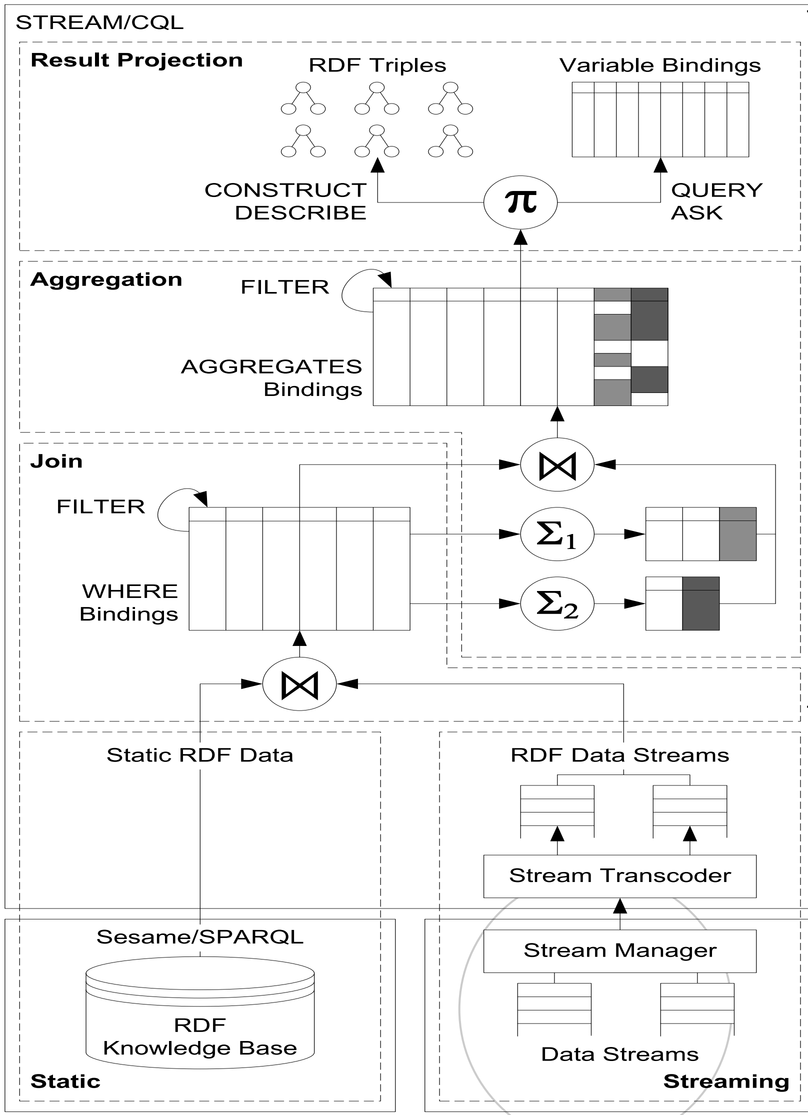
Black-box approach



C-SPARQL


CSPARQL to SPARQL
Query Rewriter
SPARQL Engine
Data transformation
Orchestrator
C-SPARQL
RDF to Java objects
Data transformation
ESPER
Query Rewriter
CSPARQL to EPSER EPL

C-SPARQL execution process

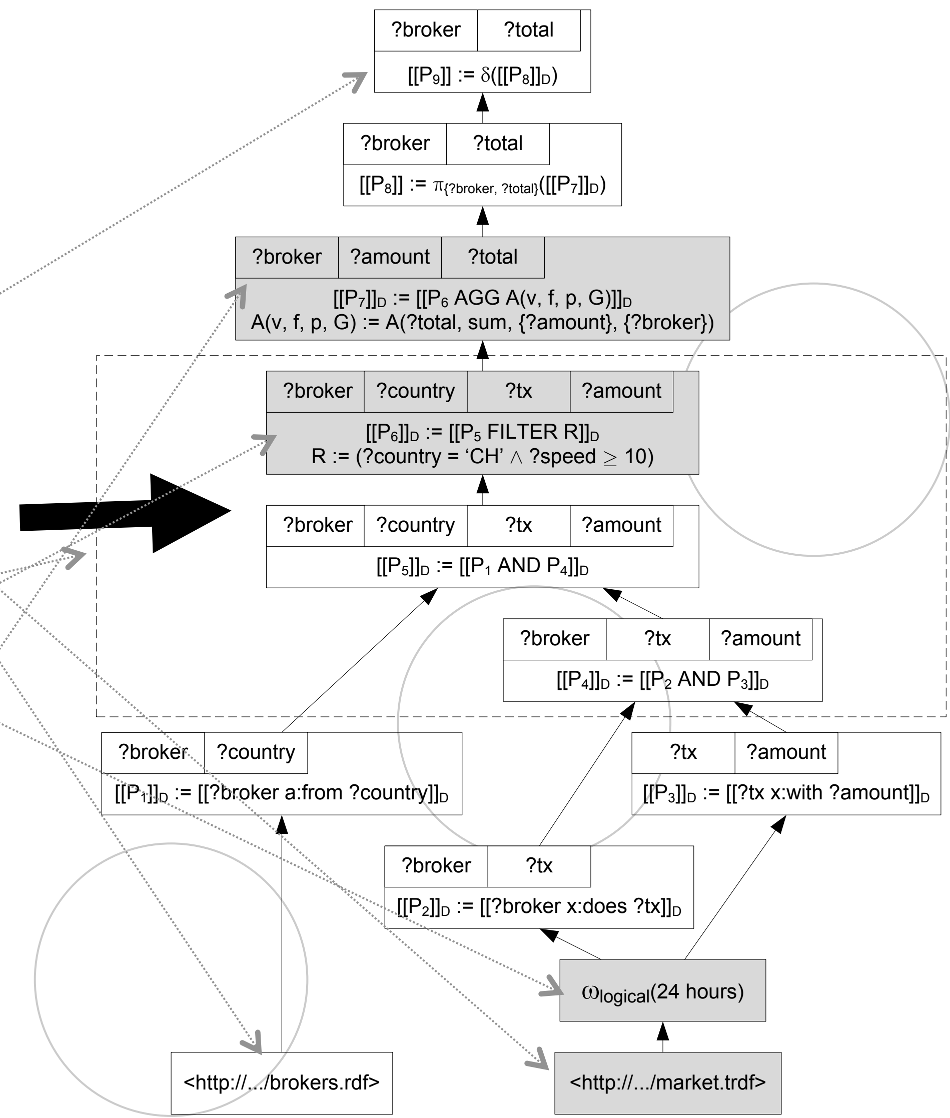


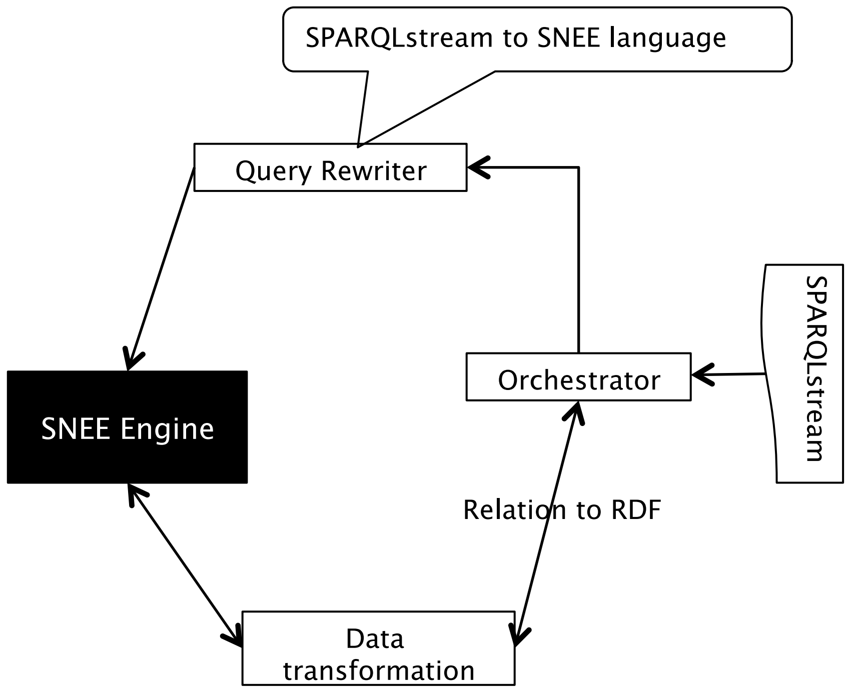
Execution order dictated by rewriting rule

Data transformation

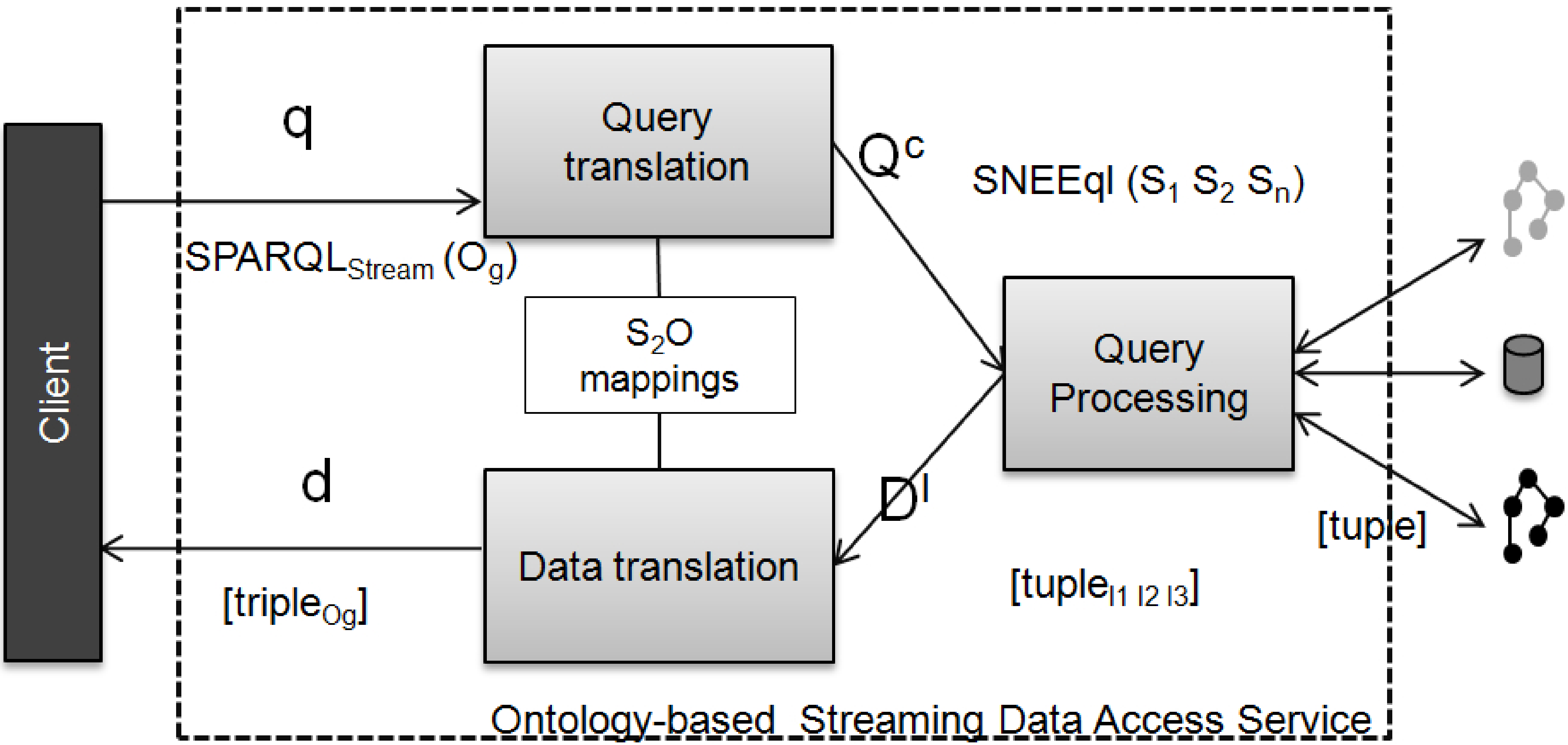
C-SPARQL query rewriting


```
REGISTER QUERY TotalAmountPerDayAndBroker AS

PREFIX b: <http://brokerscentral.org/accounts#>
PREFIX x: <http://stockexchange.org/exchanges#>

SELECT DISTINCT ?broker ?total
FROM <http://brokerscentral.org/brokers.rdf>
FROM STREAM <http://stockexchange.org/market.trdf>
            [RANGE 24h TUMBLING]
WHERE { ?broker b:is_from ?country .
        ?broker x:does ?tx .
        ?tx x:with ?amount .
        FILTER (?country = 'CH' && ?amount >= 10)
      }
AGGREGATE { (?total, SUM(?amount), ?broker) }
```




SPARQLstream to SNEE language
Query Rewriter
SNEE Engine
Orchestrator
SPARQLstream
Relation to RDF
Data
transformation

$SPARQL_{stream}$: Ontology-based mapping



An example query of SPARQL$_{stream}$


```
PREFIX fire: <http://www.semsorgrid4env.eu#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT RSTREAM ?WindSpeedAvg
FROM STREAM <www.semsorgrid4env.eu/SensorReadings.srdf> [FROM NOW - 10
    MINUTES TO NOW STEP 1 MINUTE]
FROM STREAM <www.semsorgrid4env.eu/SensorArchiveReadings.srdf> [FROM NOW - 3
    HOURS TO NOW -2 HOURS STEP 1 MINUTE]
WHERE {
  {
    SELECT AVG(?speed) AS ?WindSpeedAvg
    WHERE
    {
        GRAPH <www.semsorgrid4env.eu/SensorReadings.srdf> {
        ?WindSpeed a fire:WindSpeedMeasurement;
             fire:hasSpeed ?speed; }
    } GROUP BY ?WindSpeed
  }
  {
    SELECT AVG(?archivedSpeed) AS ?WindSpeedHistoryAvg
    WHERE
    {
      GRAPH <www.semsorgrid4env.eu/SensorArchiveReadings.srdf> {
        ?ArchWindSpeed a fire:WindSpeedMeasurement;
        fire:hasSpeed ?archivedSpeed; }
    } GROUP BY ?ArchWindSpeed
  }
  FILTER (?WindSpeedAvg > ?WindSpeedHistoryAvg)
}
```

every minute computes the average wind **speed** measurement for each sensor over the last 10 minutes if it is higher than the average of the last 2 to 3 hours.

An example of mapping rule in S_2O

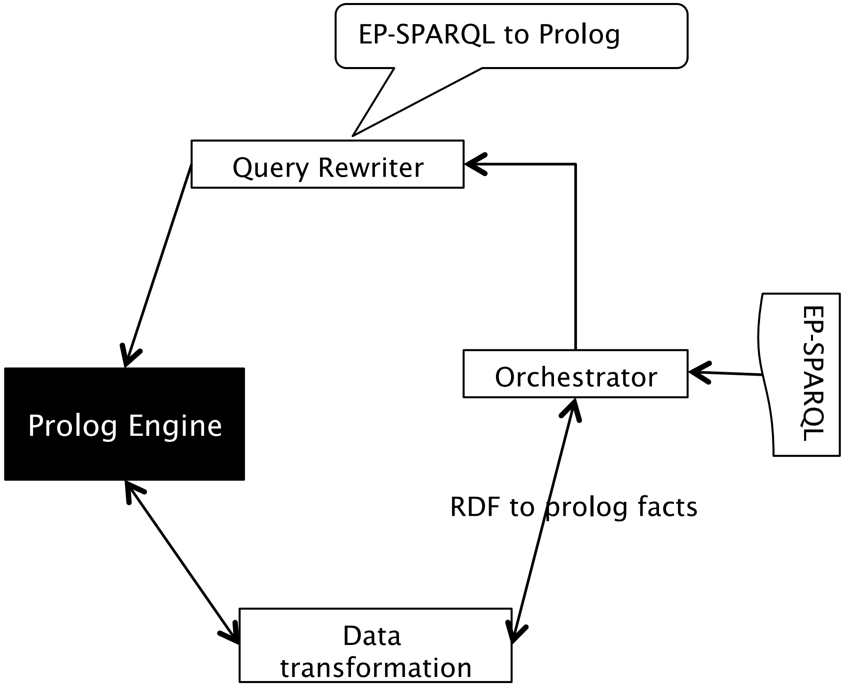

```
streamschema-desc
  name MeteoSensors
  has-stream SensorWind
    streamType pushed
    documentation "Wind measurements"
    keycol-desc measurementId
      columnType integer
    timestamp-desc measureTime
      columnType datetime
    nonkeycol-desc measureSpeed
      columnType float
    nonkeycol-desc measureDirection
      columnType float
...
conceptmap-def Wind
  virtualStream <http://semsorgrid4env.eu/SensorReadings.srdf>
  uri-as
    concat(SensorWind.measurementID)
  applies-if
   <cond-expr>
  described-by
    attributemap-def hasSpeed
      virtualStream http://semsorgrid4env.eu/SensorReadings.srdf>
      operation constant
        has-column SensorWind.measureSpeed
```

Relational stream to be mapped

Map the columns with ontological properties

S_2O declaration of a data stream schema and mapping from a stream schema to an ontology concept.

EP-SPARQL



EP-SPARQL

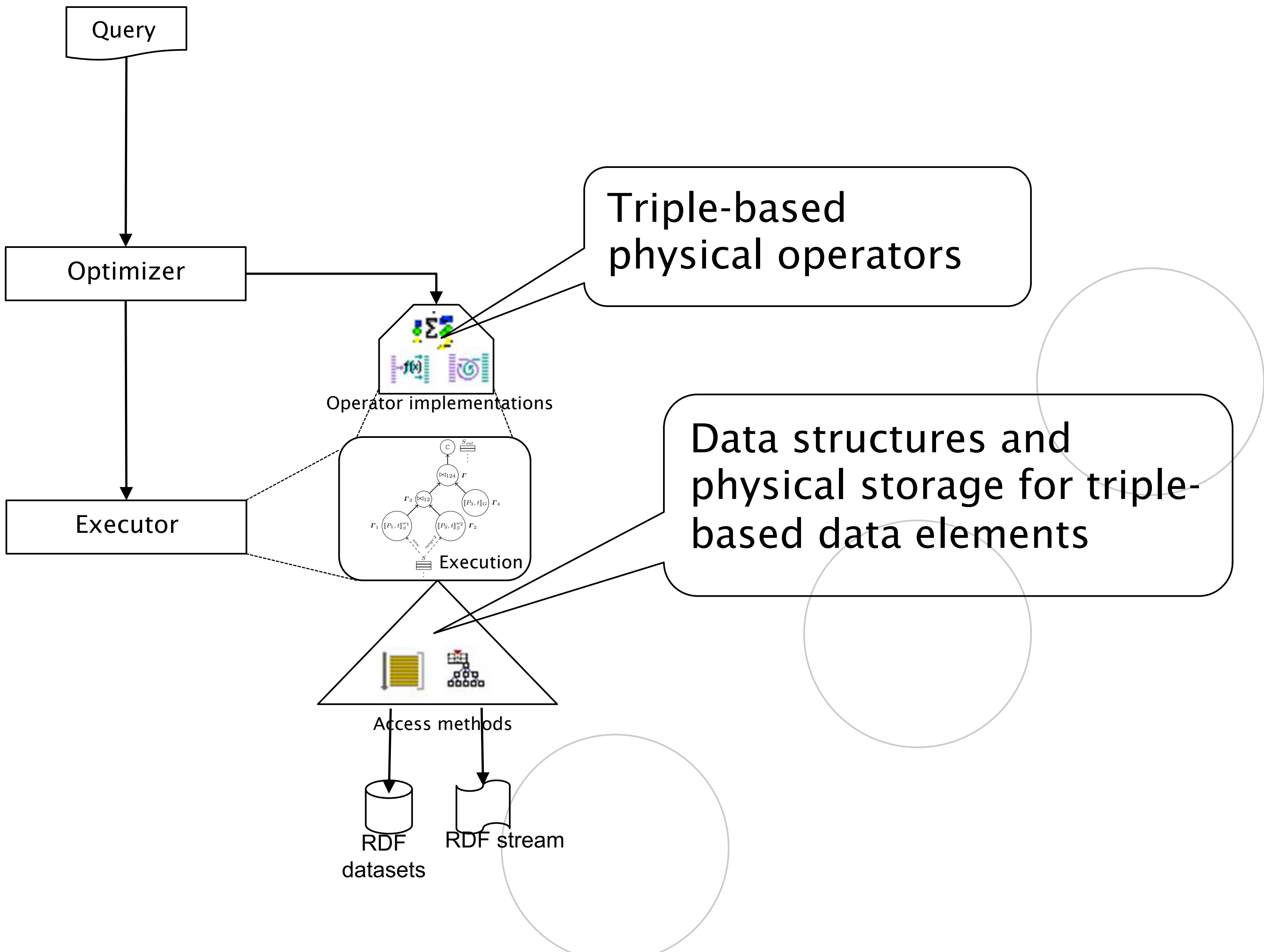
EP-SPARQL

- Execution mechanism : Prolog-based event-driven backward chaining (EDBC) rules

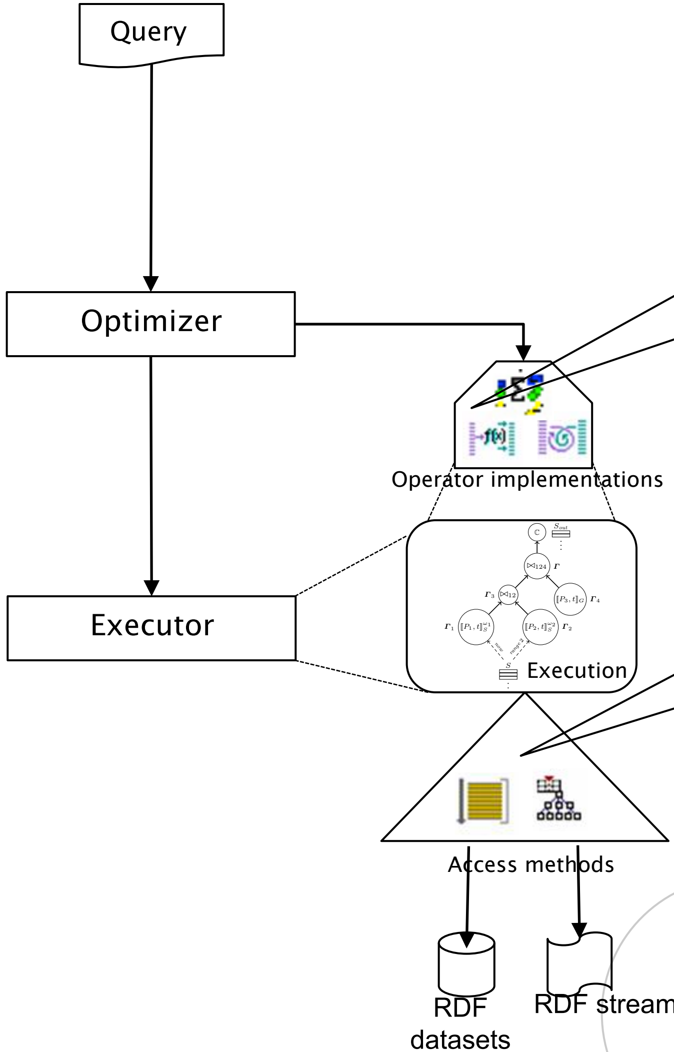
- Representation
  - RDF triple (s,p,o) ➔predicate *triple(s,p,o)*
  - Time-stamped RDF triple (s,p,o,t_1,t_2) ➔ predicate $triple(s,p,o,T_1,T_2)$

- Operators rewriting
  - Operators (SeqJoin, Filters, etc) are rewritten in Prolog rules
  - Two types of EDBC rules
    - Goal-insertion rules : to create intermediate goals of incoming events
    - Checking-rule: check if intermediate goals are triggered

Whitebox approach: Streaming SPARQL and CQELS



Streaming SPARQL

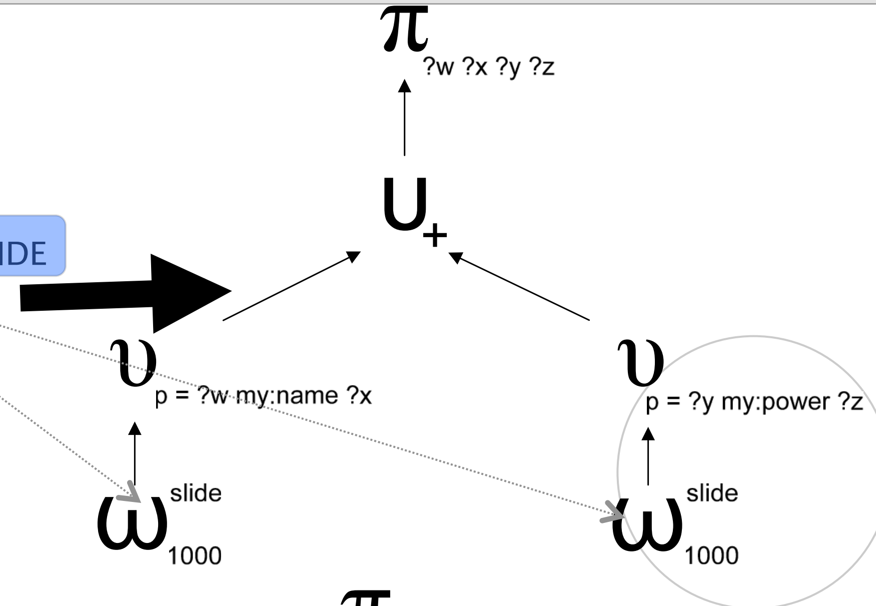


Extension of SPARQL physical operators for windowing graph patterns

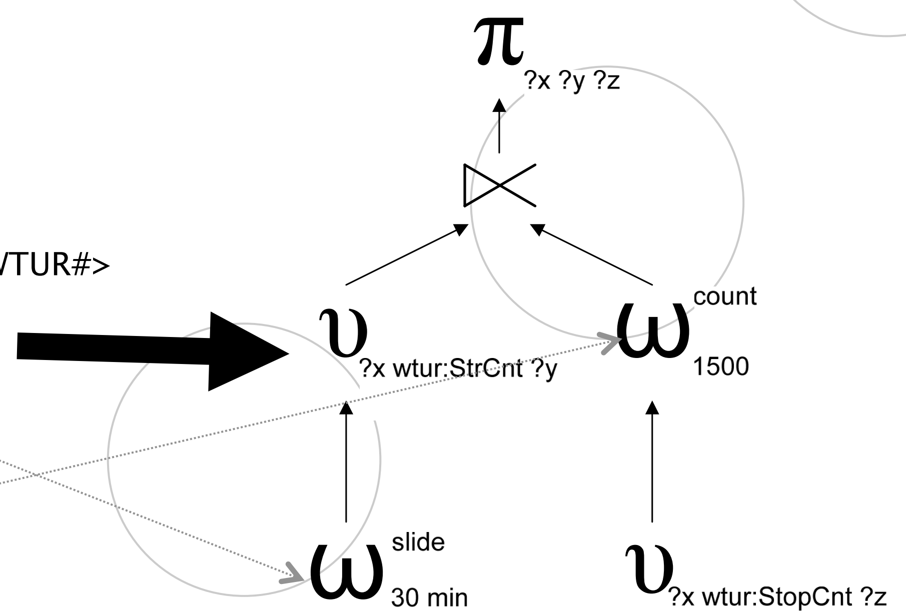
Extends SweepArea for triple-based stream elements

Examples of executing physical operators of Streaming SPARQL engine

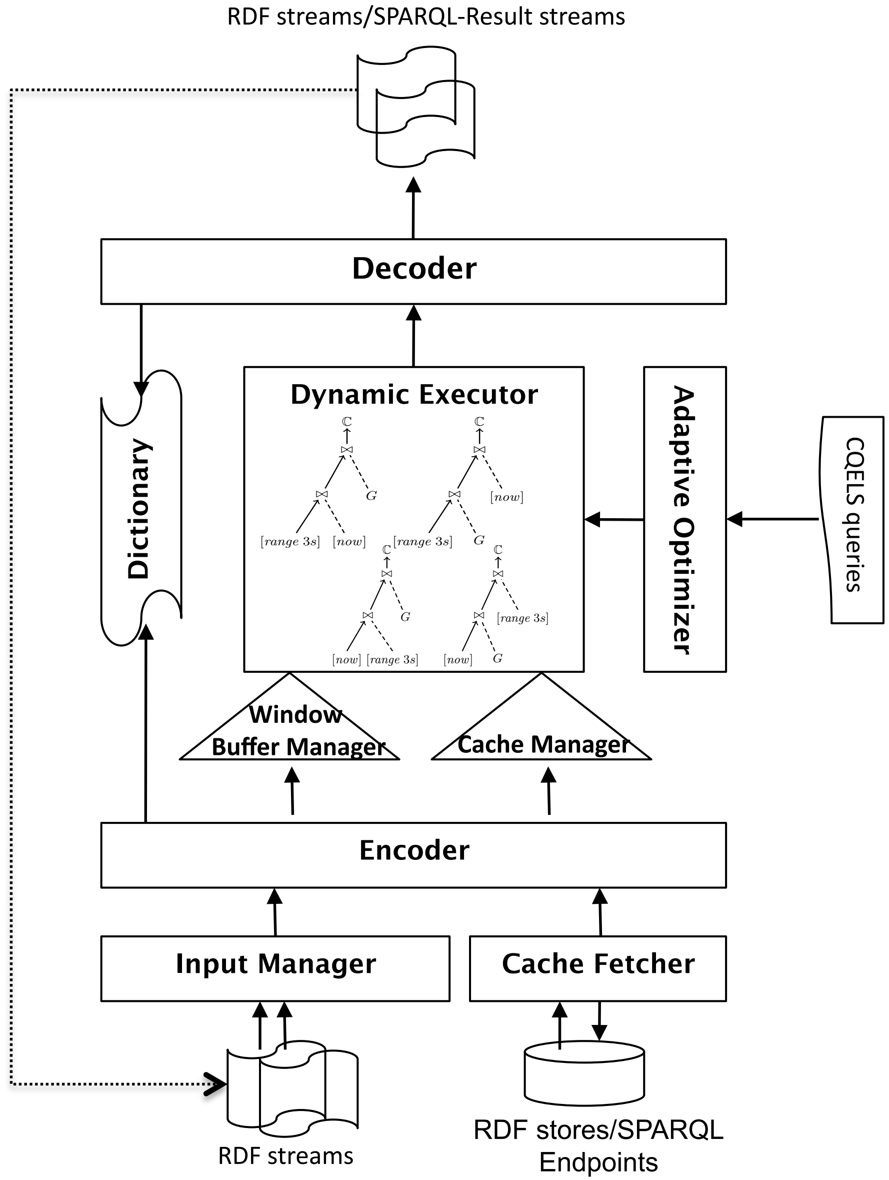
SELECT ?w ?x ?y ?z
FROM STREAM <http:src.net/graph.rdf> WINDOW RANGE 1000 SLIDE
WHERE {?w my:name ?x} UNION {?y my:power ?z}



PREFIX wtur: <http:iec.org/61400-25/root/ln/classes/WTUR#>
SELECT ?x ?y ?z
FROM STREAM <http:/iec.org/61400-25/root/td.Rdf>
WINDOW RANGE 30 MINUTE SLIDE
WHERE {?x wtur:StrCnt ?y .
OPTIONAL {?x wtur:StopCnt ?z .
{WINDOW ELEMS 1500}}

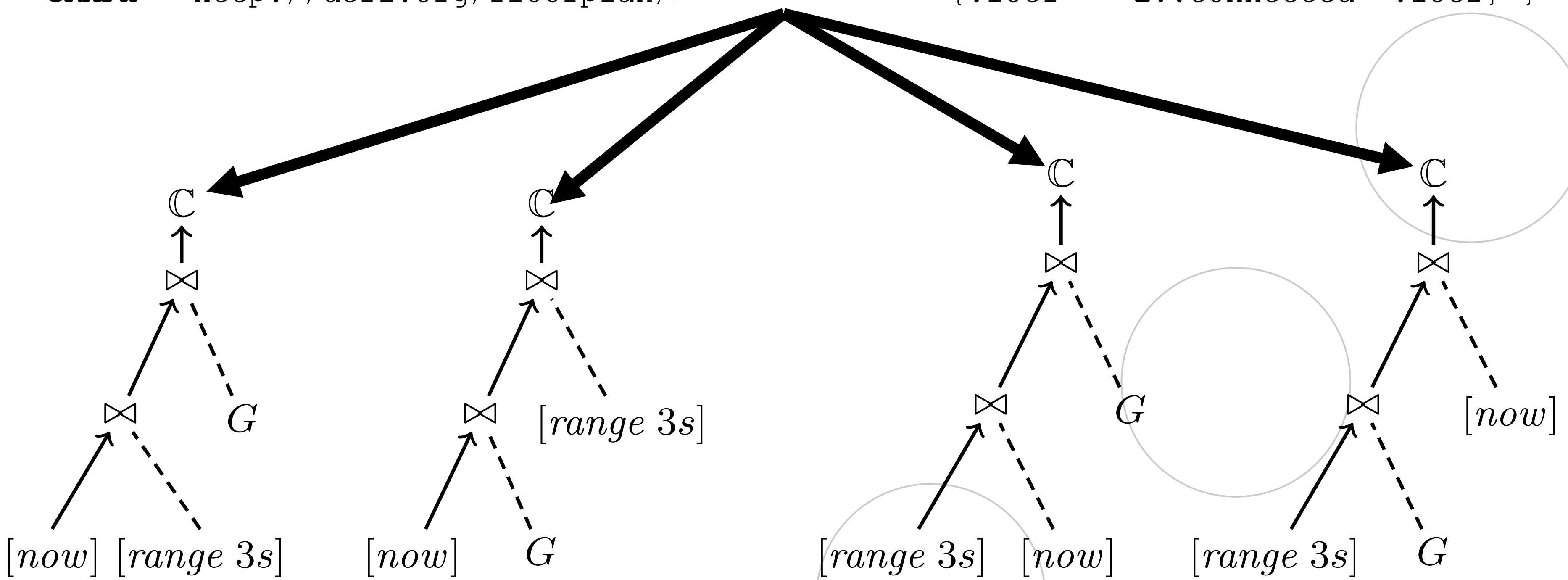


CQELS architecture for adaptive and native processing



Adaptive execution of CQELS

```
CONSTRUCT {?person1 lv:reachable ?person2}
FROM NAMED <http://deri.org/floorplan/>
WHERE {
  STREAM <http://deri.org/streams/rfid> [NOW]       {?person1 lv:detectedAt ?loc1}
  STREAM <http://deri.org/streams/rfid> [RANGE 3s] {?person2 lv:detectedAt ?loc2}
  GRAPH  <http://deri.org/floorplan/>               {?loc1    lv:connected  ?loc2} }
```



(a) From window *now*   (b) From window *range 3s*

System Comparisons

	Input	Query language	Extras
Streaming SPARQL	RDF stream		
C-SPARQL	RDF Stream & RDF	TF	
EP-SPARQL	RDF Stream & RDF	EVENT,TF	Event operators
$SPARQL_{stream}$	Relational stream	NEST	Ontology-based mapping
CQELS	RDF Stream & RDF	VoS,NEST	Disk spilling

TF: built-in time functions **EVENT:** event pattern **NEST:** nested patterns **VoS:** Variables on streams ID

	Architecture	Re-execution	Scheduling	Optimisation
Streaming SPARQL	whitebox	periodical	Logical plan	Algebraic & Static
C-SPARQL	blackbox	periodical	Logical plan	Algebraic & Static
EP-SPARQL	blackbox	eager	Logic program	Externalised
$SPARQL_{stream}$	blackbox	periodical	External call	Externalised
CQELS	whitebox	eager	Adaptive physical plans	Physical & Adaptive

Experiment setup for performance comparisons

- Conference scenario : combine linked stream from RFID tags (physical relationships) with DBLP data (social relationships)
- Setup
  - Systems : CQELS vs ETALIS and C-SPARQL
  - Datasets
    - Replayed RFID data from Open Beacon deployments
    - Simulated DBLP by SP^2Bench
  - Queries : 5 query templates with different complexities
    - Q1: selection,
    - Q2: stream joins, Q3,Q4 : Stream and non-stream joins
    - Q5: aggregation
  - Experiments
    - Single query : generate 10 query instances of each template by varying the constants
    - Vary size of the DBLP (10^4-10^7triples)
    - Multiple queries : register 2^M instances at the same time($0 \leq M \leq 10$)

Performance comparison- Query execution time

- CQELS perform fasters by orders of magnitudes

	Query 1	Query 2	Query 3	Query 4	Query5
CQELS	0.47	3.90	0.51	0.53	21.83
C-SPARQL	332.46	99.84	331.68	395.18	322.64
ETALIS	0.06	27.47	79.95	469.23	160.83

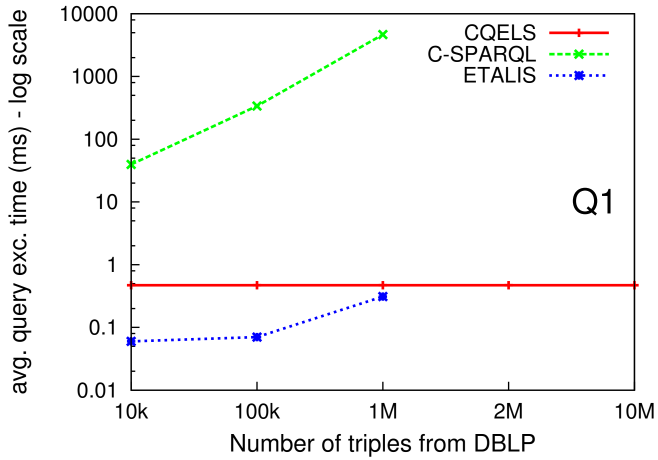
Simple selection : ETALIS perform best

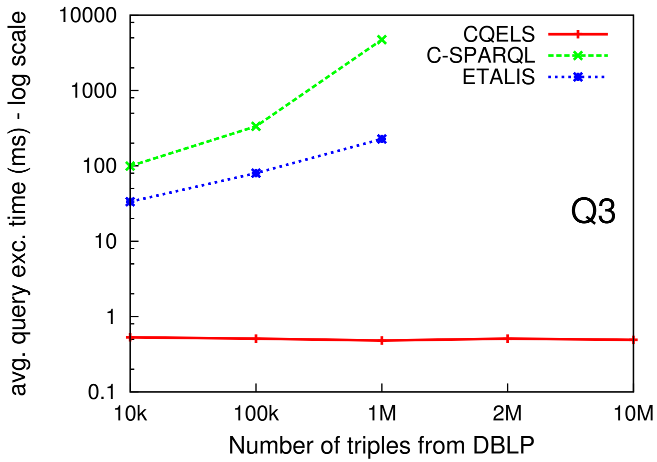
Stream join :
25 times faster than C-SPARQL
8 times faster than ETALIS

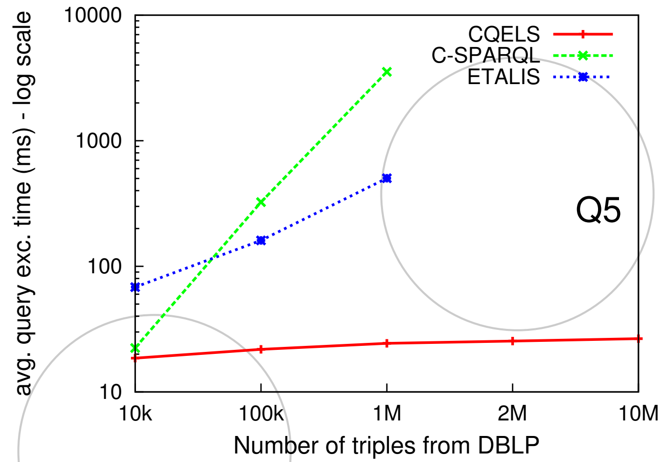
Stream and non-stream joins :
>600 times faster than C-SPARQL
150-850 times faster than ETALIS

Aggregation:
15 times faster than C-SPARQL
8 times faster than ETALIS

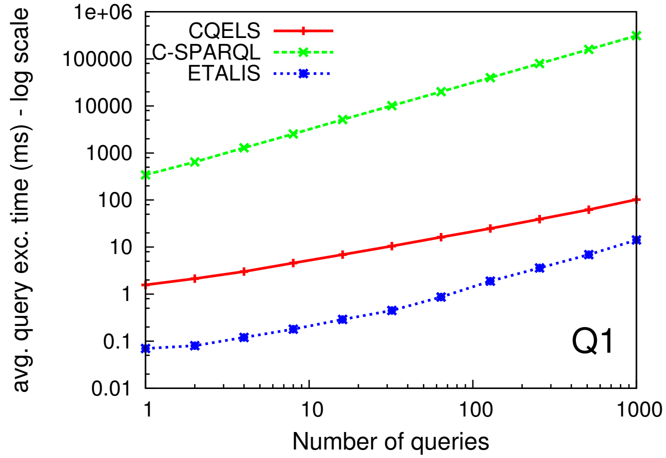
Non-stream data size : logarithmic to size of static intermediate results

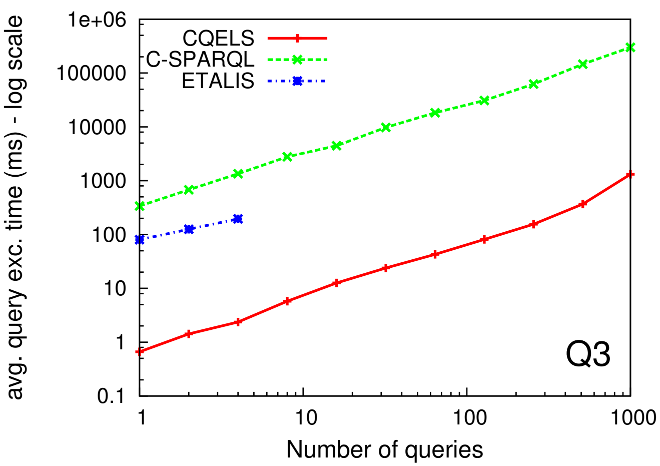


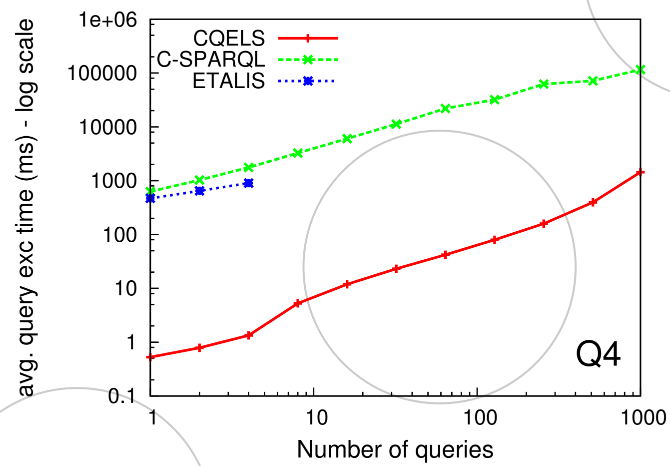




Performance comparison- Scalability (number of queries)

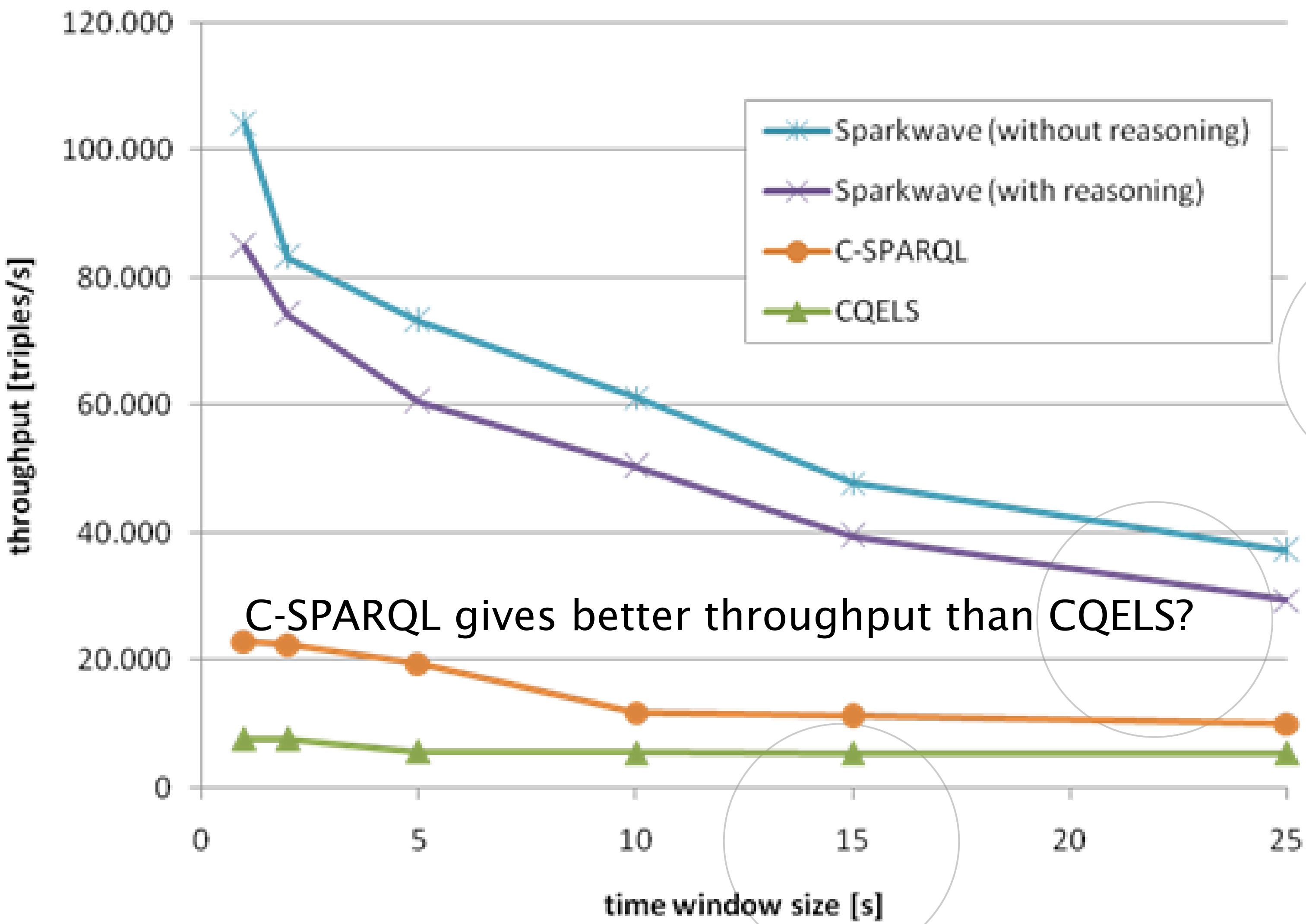






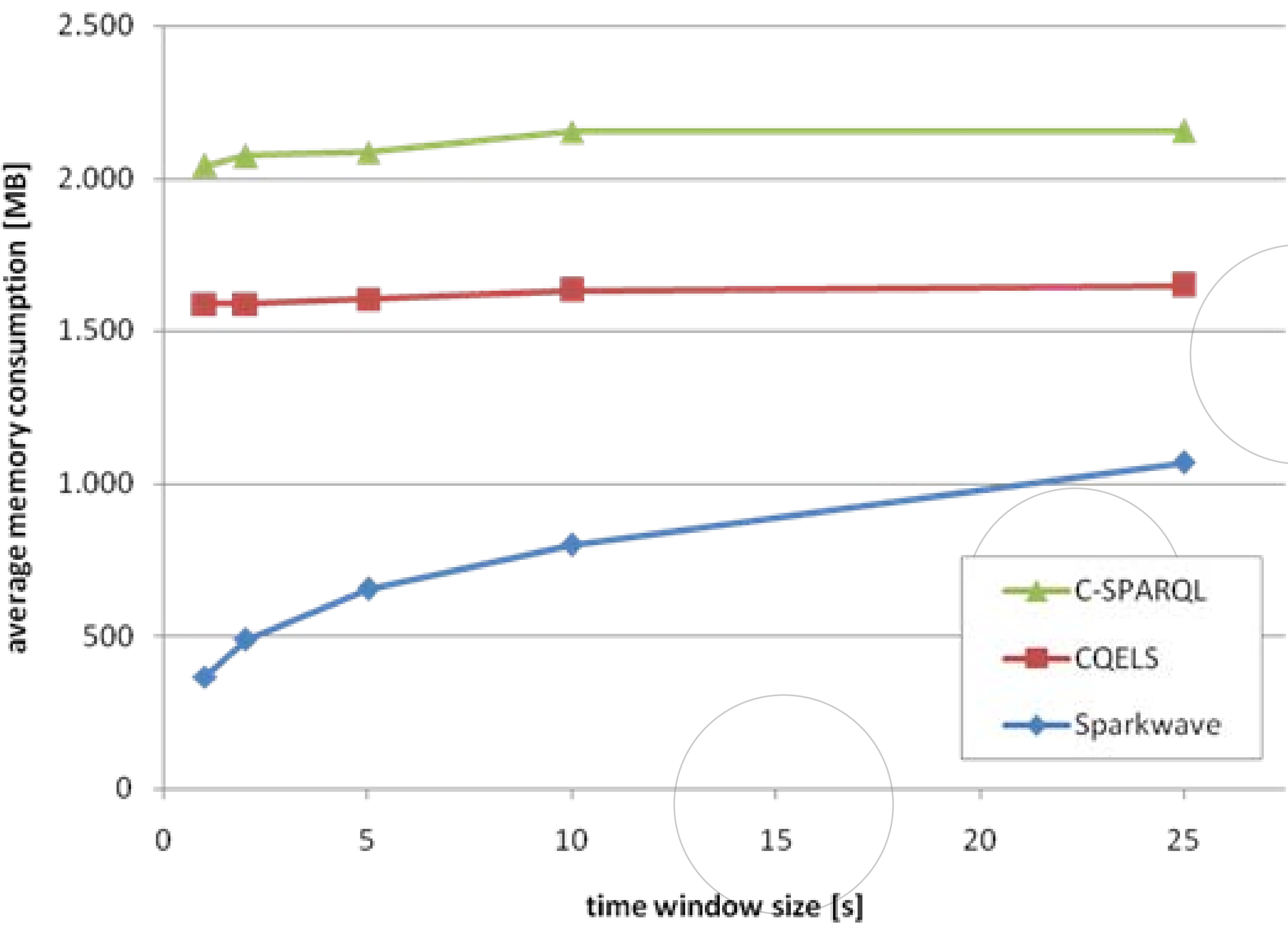
No multiple query optimization

Performance comparison-Maximum input throughput



C-SPARQL gives better throughput than CQELS?

Performance comparison – Memory consumption



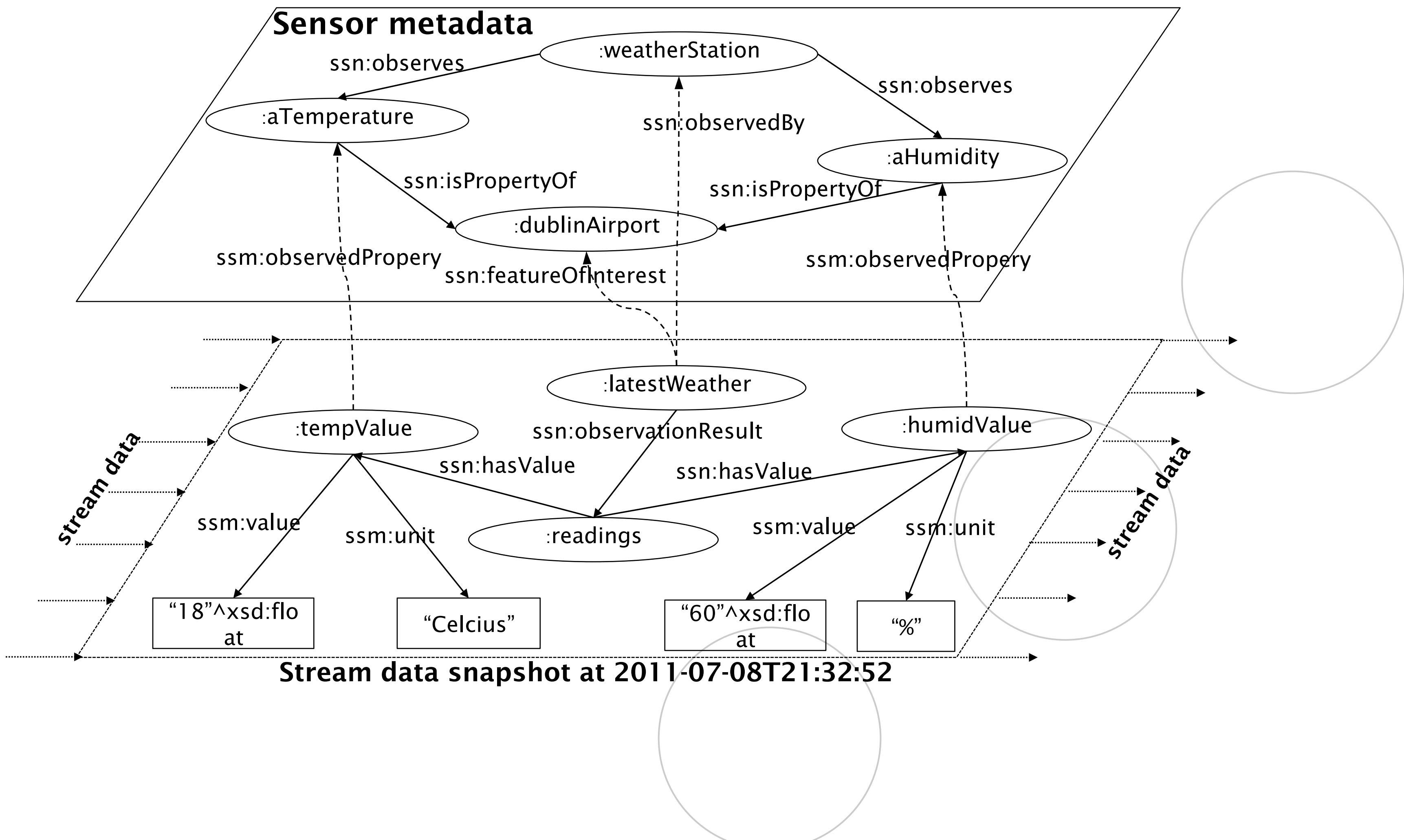
Are they ready for production?

- Not quite !!!(just 4-6 years)
- Why?
  - Functionality
  - Performance
  - Scalability
- But interesting to test/compete/extend/study
  - Vast amount of interesting of heterogeneous data streams and open Linked Data sources
  - Plethora of use cases/applications
  - New interesting research problems

Open challenges

- Serialization of RDF-based stream elements
- Optimization & Scheduling
- How to measure and compare performances
- Early-stage
  - Expressiveness
  - Reasoning
- Lack of functionalities
  - Disk-based stream processing
  - Distributed processing for large-scale data

How to serialize the RDF stream elements- Graph-based stream layout



How to serialize the RDF stream elements

Extension of NTRIPLE: line-based, plain text format

triple	Timestamp
`c:Distr1 t:has-entering-cars "100"`	t_{400}
`c:Distr2 t:has-entering-cars "75"`	t_{400}
`c:Distr1 t:has-entering-cars "130"`	t_{401}
`c:Distr2 t:has-entering-cars "95"`	t_{401}
`c:Distr3 t:has-entering-cars "65"`	t_{401}

- Lack of standard way to serialize RDF Stream
- N-Triple-like representation is inefficient
  - 100-500 bytes to represent 1 integer reading
  - 100k triples/sec➔10-50MB/sec➔ 80-400Mbps bandwidth
- Is it necessary in text-line format? Binary format?

Optimization & Scheduling

- At logical plan level → inefficient and restricted on highly dynamic settings of the stream processing.
- Few at physical level but only with naïve/simplistic algorithms/strategies.
- None support multiple query optimization

☞ Needs more studies on optimization and scheduling graph-based query patterns

How to compare performance

- There are only few stream benchmarking systems
  - Linear Road benchmark (VLDB 2004)
  - LSBench (to appear at ISWC 2012)
  - SRBench (to appear at ISWC 2012)
- How to define the measurement to compare
  - Execution time/Response time?
  - Throughput?
- Too many elements to cause differences in outputs
  - Difference in semantics
  - The execution mechanisms
  - Execution environments and settings

Early-stage work

- Expressiveness
  - SPARQL extensions based on relational algebra is not expressive enough for stream/event processing applications
  - Higher expressive continuous query language?
    - Recursive expression
    - Rule-based expression
    - Support uncertainty in matching pattern
- Stream reasoning based on RDF data model
  - Emerging topic with some early work
    - Barbieri et al. Incremental Reasoning on Streams and Rich Background Knowledge (ESWC'2010).
    - Komazec et al. Sparkwave: continuous schema-enhanced pattern matching over RDF data streams (DEBS'2012)
  - Complexity vs low latency ➔ need quantitative metrics to judge the advantages of each stream reasoner

Lack of functionalities

- Disk-based stream processing
  - Big windows
  - Big linked data sets
- Distributed stream processing
  - Some general distributed stream processing platform/ systems
    - Borealis, StreamBase, IBM Stream Spheres, etc
    - S4, Storm, Kafka, etc
  - Use black-box approach delegate processing ➔ How to deal with the over head and restriction of optimization?
  - Use whitebox approach ➔ which physical processing can be reused from such platform/system? Can it be better?

Summary

- What is Linked Stream Data?
- Data models for Linked Stream Data
- Query operators and query languages
- How to build a Linked Stream Processing Engine
- Comparisons and analysis of State-Of-The-Art systems
- Open challenges
  - Serialization of RDF-based stream elements
  - Optimization & Scheduling
  - How to measure and compare performances
  - Early-stage & Lack of functionalities