Linked Stream Data Processing
Part I: Basic Concepts & Modeling

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

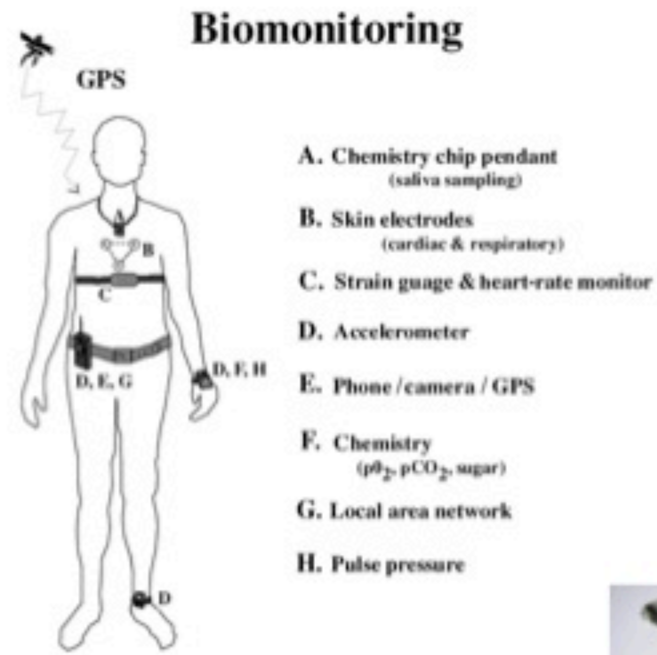
Reasoning Web Summer School 2012

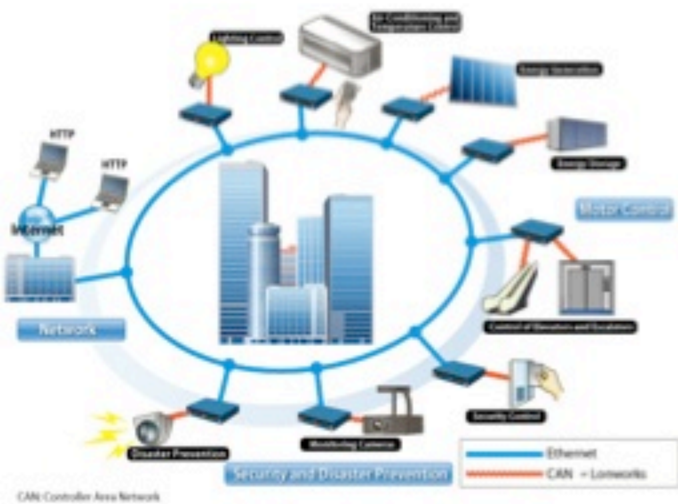
Enabling **networked** knowledge

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

Streams everywhere





Application Domains

Enterprise Environments

Smart Cities

Green IT

Telehealth

Sorry, I can't understand you…

- Heterogeneous data representations
- Lack of semantics
- A priori knowledge of data sources needed
- Disconnected

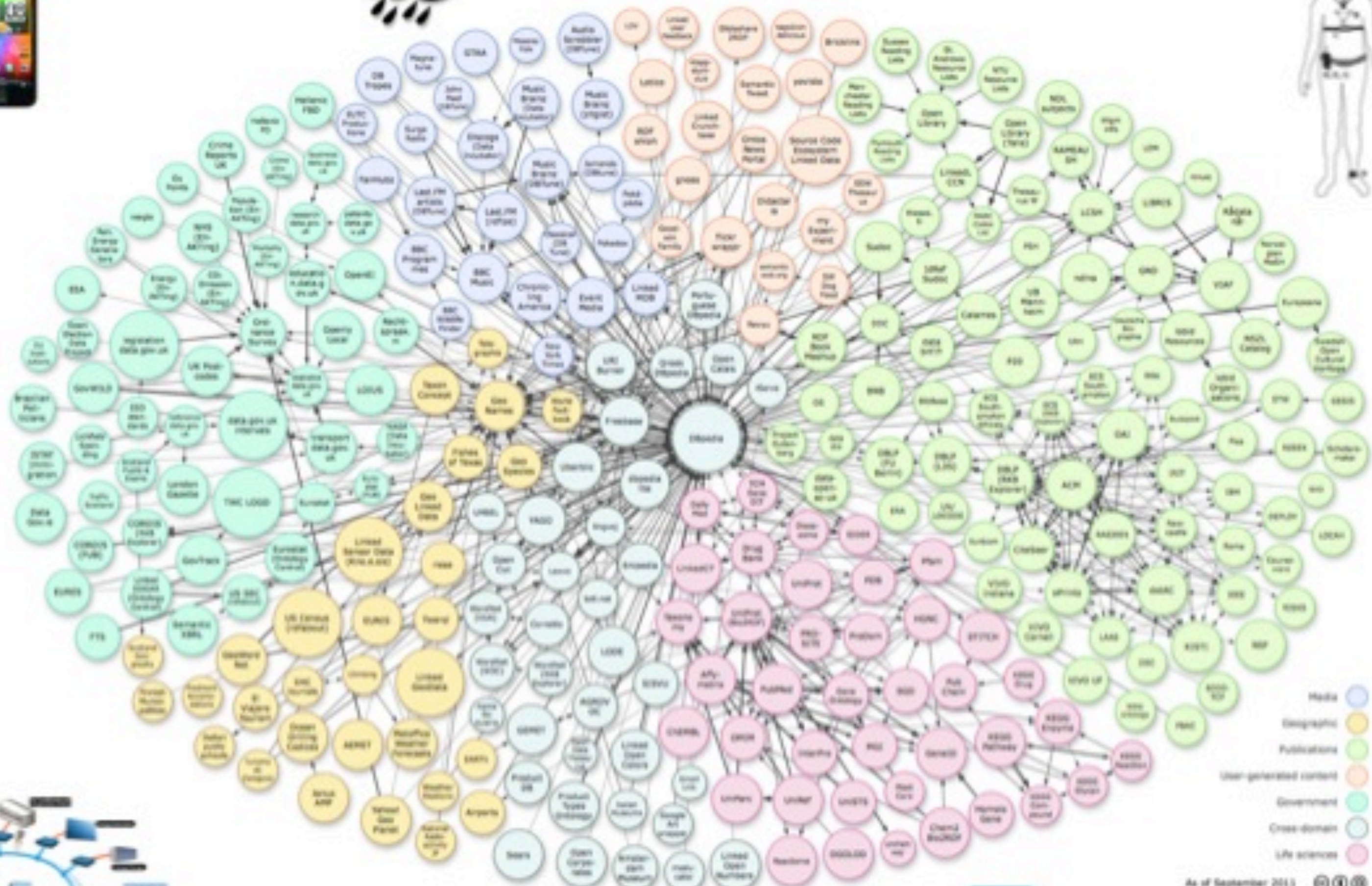
Integration Problem!

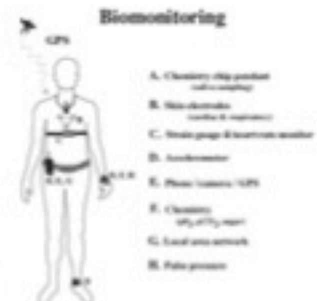
Semantic Web, Linked Data

- Semantic Web
  - Collaborative movement to promote common data formats on the World Wide Web.
  - Inclusion of semantic content in web pages
  - From unstructured and semi-structured documents to a "Web of data"
- Linked Data
  - Best practices to represent, publish, link data on the Semantic Web
  - Linked Data Cloud: collection of datasets that have been published in Linked Data format

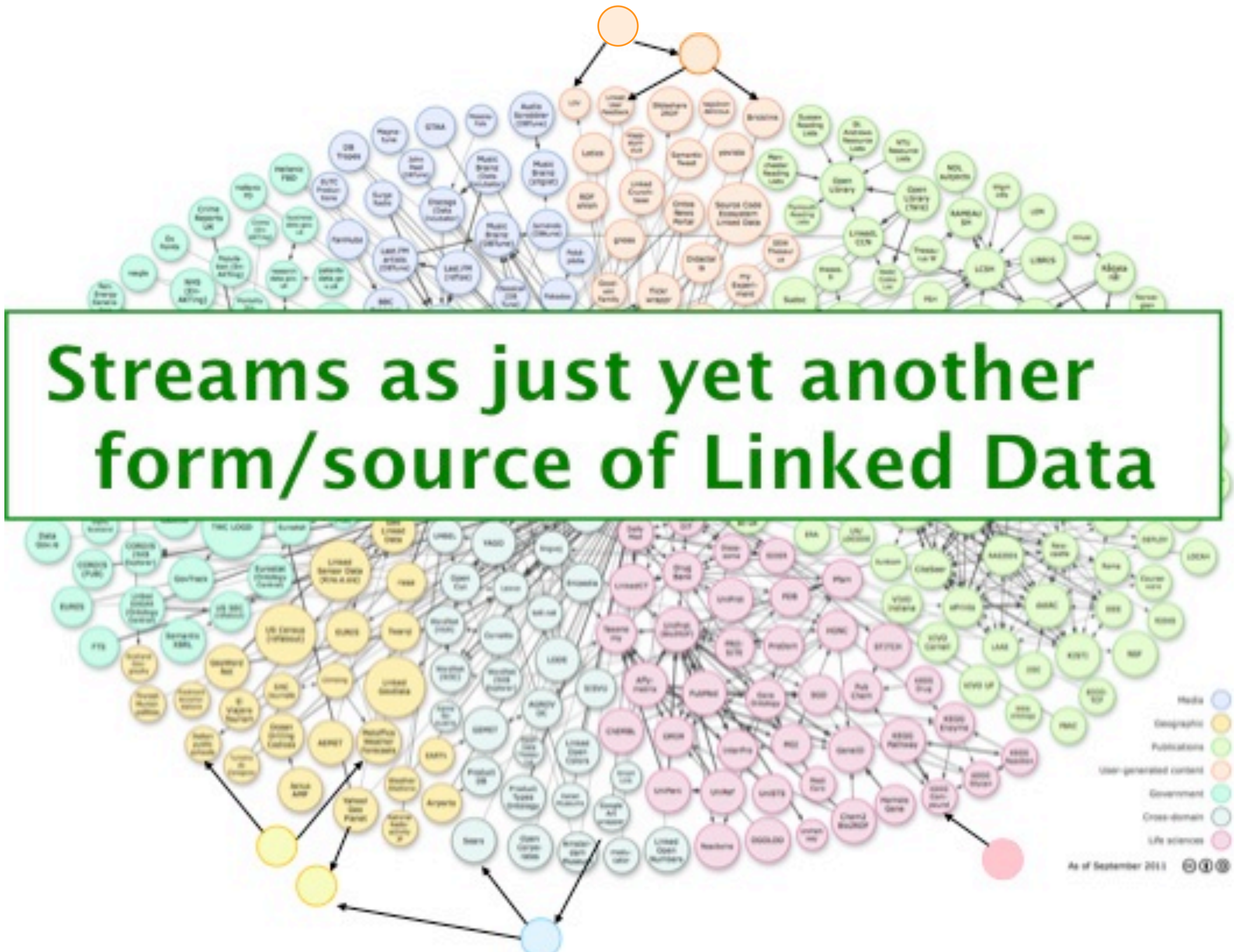
LINKED STREAM DATA

Linked Stream Data





Linked Stream Data



Streams as just yet another form/source of Linked Data

Linked Stream Data

- Semantically enriched stream data
- Linked Stream Data examples
  - W3C Semantic Sensor Network Incubator Group
  - RDF wrappers for Twitter, Facebook, etc
- Data integration, connects dynamic and static data
- Linked Data + DSMS
  - Stream Data representation/processing different from standard RDF/SPARQL
    - Temporal aspect, continuous query processing
  - DSMS use relational storage model
    - Efficient RDF processing requires heavy replication

Running example

Running Example – Conference scenario

- Tracking system (e.g RFID tags) : Stream data
- Attendees information (e.g. DBLP records, FOAF)
- Building information (e.g. layout, connections, room names)
- Different sources (no common schema)
- Linked data used as unified model

Running Example

(Q1) Inform a participant about the name and description of the location he currently is

```
PREFIX lv: http://deri.org/floorplan/
PREFIX foaf: http://xmlns.com/foaf/0.1/

SELECT ?locName ?locDesc
FROM NAMED <http://deri.org/floorplan/>
WHERE {
STREAM <http://deri.org/streams/rfid> [NOW] {?person lv:detectedat ?loc}
 GRAPH <http://deri.org/floorplan/>
                        {?loc lv:name ?locName. ?loc lv:desc ?locDesc}
?person foaf:name ‘‘$Name$’’.
}
```

Linked Stream Data

- Linked Data principles applied to stream data
- Extensions to deal with the temporal aspects
  - Data modeling
  - Query languages
  - Query operators
  - System design and architectures

DATA MODELS, QUERY LANGUAGES AND OPERATORS

Linked Stream Data model

- Extends the definition of RDF nodes and RDF triples
  - RDF node: I, B, and L, which are pair-wise disjoint infinite sets of Information Resource Identifiers (IRIs), blank nodes and literals
  - RDF triple: $(s, p, o) \in IB \times I \times IBL$, where $IL = I \cup L$, $IB = I \cup B$ and $IBL = I \cup B \cup L$
- Stream element: RDF triple with temporal annotations
  - Interval-based (e.g. ⟨:John :at :office,[7,9]⟩) – Streaming SPARQL
  - Point-based (e.g. ⟨John :at :office,7⟩, ⟨:John :at :office,8⟩, ⟨:John :at :office,9⟩) – EP-SPARQL, C-SPARQL, $\text{SPARQL}_{\text{Stream}}$, CQELS
  - Point-based (maybe) redundant, but instantaneous (more practical)

Linked Stream Data model

- RDF Stream: bag of elements $\langle(s,p,o) : [t]\rangle$
  - (s,p,o) : RDF triple
  - t : timestamp
  - stream elements from stream S with timestamp $\leq$ t

    $$S^{\leq t} = \{\langle(s,p,o):[t']\rangle \in S \mid t' \leq t\}$$
- Non-stream data (RDF datasets) also need to follow the Linked Stream Data model to allow integration → Instantaneous RDF dataset: G(t)
- G(t) : set of RDF triples valid at time t, called instantaneous RDF dataset.
- RDF dataset : sequence $G = [G(t)]$, $t \in N$, ordered by t.
  - Static RDF dataset (G^s): $G(t) = G(t+1)$ for all $t \geq 0$

Query Operators

- Pattern matching as basic operator (extended from SPARQL)
  - Mappings which are defined as partial functions

$$\mu : V \longmapsto IBL$$

where V is an infinite set of variables disjoint from IBL, and dom(μ) is the subset of V where μ is defined.

- Compatible mappings

$$\mu_1 \doteq \mu_2 \iff \forall x \in dom(\mu_1) \cap dom(\mu_2) \Rightarrow \mu_1(x) = \mu_2(x)$$

Query Operators

- Join, union, different and left outer-join follow mappings (Ω_1 and Ω_2 are mapping sets)

$$\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1 \wedge \mu_2 \in \Omega_2 \wedge \mu_1 \doteq \mu_2\}$$

$$\Omega_1 \cup \Omega_2 = \{\mu \mid \mu_1 \in \Omega_1 \vee \mu_2 \in \Omega_2\}$$

$$\Omega_1 \setminus \Omega_2 = \{\mu \in \Omega_1 \mid \neg\exists \mu' \in \Omega_2, \mu' \doteq \mu\}$$

$$\Omega_1 ⟕ \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2)$$

Query Operators

- Triple matching operator

$$[\![P, t]\!]_G = \{\mu \mid dom(\mu) = var(P) \wedge \mu(P) \in G(t)\}$$

  - Triple pattern P $\in$(I $\cup$ V)$\times$(I $\cup$ V)$\times$(IL $\cup$ V)
  - μ(P): triple obtained by replacing variables within P according to μ
- Window matching operator

$$[\![P, t]\!]_S^\omega = \{\mu \mid dom(\mu) = var(P) \wedge \langle \mu(P) : [t'] \rangle \in S \wedge t' \in \omega(t)\}$$

  - ω(t): N $\rightarrow$ 2^N : function mapping a timestamp to a (possibly infinite) set of timestamps (N : set of natural numbers)
  - ω(t) will depend on the type of the window (e.g. time-based, tuple-based)

Query Operators

- Sequential Operator

$$[\![P_1 \Rightarrow^t P_2]\!]_S^\omega = \{\mu_1 \cup \mu_2 \mid \mu_1 \in [\![P_1, t]\!]_S^\omega \wedge \mu_2 \in [\![P_2, t]\!]_S^\omega \wedge \mu_1 \doteq \mu_2$$
$$\wedge \langle \mu_1(P_1) : [t'_1] \rangle \in S \wedge \langle \mu_2(P_2) : [t'_2] \rangle \in S \wedge t'_1 \leq t'_2\}$$

- AND, UNION, OPT, FILTER, AGG can be derived from operators introduced so far

Query Languages

- Extensions of SPARQL grammar for continuous queries
  - Few different languages have been proposed
  - Clauses to handle streams and to add window operators
- StreamingSPARQL: DatastreamClause, Window

$$
\begin{array}{rcl}
\mathit{SelectQuery} & ::= & \text{`SELECT' (`DISTINCT' | `REDUCED'')?'}(\mathit{Var} \mid \text{`*'})(\mathit{DatasetClause}* \mid \\
 & & \textbf{DatastreamClause}*)\ \mathit{WhereClause}\ \mathit{SolutionModifier} \\
\mathit{DatastreamClause} & ::= & \text{`FROM'}\ (\mathit{DefaultStreamClause} \mid \mathit{NamedStreamClause}) \\
\mathit{DefaultStreamClause} & ::= & \text{`STREAM'}\ \mathit{SourceSelector}\ \mathit{Window} \\
\mathit{NamedStreamClause} & ::= & \text{`NAMED' `STREAM'}\ \mathit{SourceSelector}\ \mathit{Window} \\
\mathit{GroupGraphPattern} & ::= & \{\ \mathit{TriplesBlock}?\ ((\mathit{GraphPatternNotTriples} \mid \mathit{Filter}\ )\text{`.'}? \\
 & & \mathit{TriplesBlock}?\ )^*(\textbf{Window})?)\} \\
\textbf{Window} & ::= & (\mathit{SlidingDeltaWindow} \mid \mathit{SlidingTupleWindow} \mid \mathit{FixedWindow}) \\
\mathit{skipSlidingDeltaWindow} & := & \text{`WINDOW' `RANGE'}\ \mathit{ValSpec}\ \text{`SLIDE'}\ \mathit{ValSpec}? \\
\mathit{FixedWindow} & := & \text{`WINDOW' `RANGE'}\ \mathit{ValSpec}\ \text{`FIXED'} \\
\mathit{SlidingTupleWindow} & ::= & \text{`WINDOW'ELEMSINTEGER} \\
\mathit{ValSpec} & ::= & \text{INTEGER} \mid \mathit{Timeunit}? \\
\mathit{Timeunit} & := & (\text{`MS'} \mid \text{`S'} \mid \text{`MINUTE'} \mid \text{`HOUR'} \mid \text{`DAY'} \mid \text{`WEEK'})
\end{array}
$$

Query Languages

- C-SPARQL: FromStrClause, Window

$$
\begin{aligned}
\mathit{FromStrClause} &\rightarrow \text{`FROM' [`NAMED'] `STREAM'}\ \mathit{StreamIRI}\ \text{`[RANGE'}\ \mathit{Window}\ \text{`]'} \\
\mathit{Window} &\rightarrow \mathit{LogicalWindow} \mid \mathit{PhysicalWindow} \\
\mathit{LogicalWindow} &\rightarrow \mathit{Number}\ \mathit{TimeUnit}\ \mathit{WindowOverlap} \\
\mathit{TimeUnit} &\rightarrow \text{`d'} \mid \text{`h'} \mid \text{`m'} \mid \text{`s'} \mid \text{`ms'} \\
\mathit{WindowOverlap} &\rightarrow \text{`}\mathit{STEP}' \ \mathit{Number}\ \mathit{TimeUnit} \mid \text{`TUMBLING'}
\end{aligned}
$$

- CQELS: StreamGraphPattern (IRIs for streams)

$$
\begin{aligned}
\mathit{GraphPatternNotTriples} ::= {} & \mathit{GroupOrUnionGraphPattern} \mid \mathit{OptionalGraphPattern} \\
& \mid \mathit{MinusGraphPattern} \mid \mathit{GraphGraphPattern} \\
& \mid \textbf{StreamGraphPattern} \mid \mathit{ServiceGraphPattern} \mid \mathit{Filter} \mid \mathit{Bind}
\end{aligned}
$$

$$
\begin{aligned}
\textbf{StreamGraphPattern} &::= \text{`STREAM' `['}\ \mathit{Window}\ \text{`]'}\ \mathit{VarOrIRIref}\ \text{`\{'}\mathit{TriplesTemplate}\text{`\}'} \\
\mathit{Window} &::= \mathit{Range} \mid \mathit{Triple} \mid \text{`NOW'} \mid \text{`ALL'} \\
\mathit{Range} &::= \text{`RANGE'}\ \mathit{Duration}\ (\text{`SLIDE'}\ \mathit{Duration} \mid \text{`TUMBLING'})? \\
\mathit{Triple} &::= \text{`TRIPLES' INTEGER} \\
\mathit{Duration} &::= (\text{INTEGER `d'} \mid \text{`h'} \mid \text{`m'} \mid \text{`s'} \mid \text{`ms'} \mid \text{`ns'})^{+}
\end{aligned}
$$

Query Example: 1 stream

(Q1) Inform a participant about the name and description of the location he just entered

- C-SPARQL

```
SELECT ?locName ?locDesc
FROM STREAM <http://deri.org/streams/rfid> [NOW]
FROM NAMED <http://deri.org/floorplan/>
WHERE {
?person lv:detectedat ?loc. ?loc lv:name ?locName.
?loc lv:desc ?locDesc
?person foaf:name ‘‘$Name’’.
}
```

- CQELS

```
SELECT ?locName ?locDesc
FROM NAMED <http://deri.org/floorplan/>
WHERE {
STREAM <http://deri.org/streams/rfid> [NOW] {?person lv:detectedat ?loc}
GRAPH <http://deri.org/floorplan/>
{?loc lv:name ?locName. ?loc lv:desc ?locDesc} ?person foaf:name ‘‘$Name
$ ’’.
}
```

Query Example: 2+ windows on streams

(Q2) Notify two people when they can reach each other from two different and directly connected (from now on called nearby) locations.

- Streaming SPARQL and C-SPARQL don't allow multiple windows in one stream in their grammar
  - C-SPARQL solution: create two virtual streams
- 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}
}
```

Query Example: Stream as var

- Different streams can provide the same pattern

Q3: Name of location of people nearby the DERI building

- CQELS (queries all streams that provide “nearby” info)

```
SELECT ?name ?locName
FROM NAMED <http://deri.org/floorplan/>
WHERE {
STREAM ?streamURI [NOW] {?person lv:detectedat ?loc}
GRAPH <http://deri.org/floorplan/>
{
?streamURI lv:nearby :DERI_Building. ?loc lv:name ?locName.
?person foaf:name ?name.
 }
}
```

Query Example: Timestamps

- EP-SPARQL and C-SPARQL allow functions to deal with timestamps
  - Timestamp can be retrieved and bound to a variable
  - Timestamp of a bound stream element can be retrieved

- Q4: Return pairs of people that were detected in a location in consecutive times (in the last 30min)
- EP-SPARQL

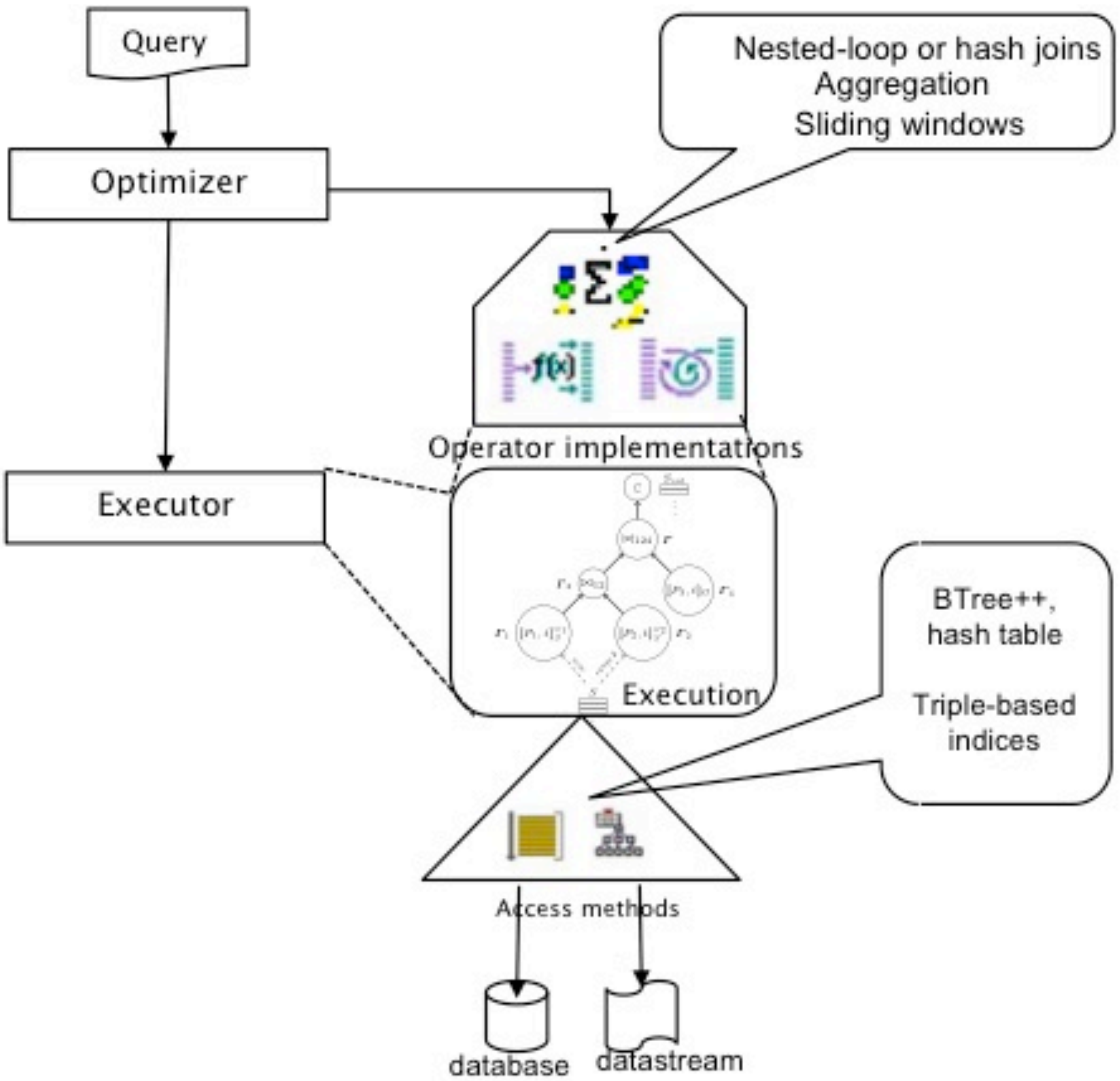
```
CONSTRUCT {?person2 lv:comesAfter ?person1} {
SELECT ?person1 ?person2
WHERE {
{?person1 lv:detectedat ?loc}
SEQ {?person2 lv:detectedat ?loc}
}
FILTER (getDURATION()<"P30m"^^xsd:duration)
```

DESIGN CHOICES & CHALLENGES

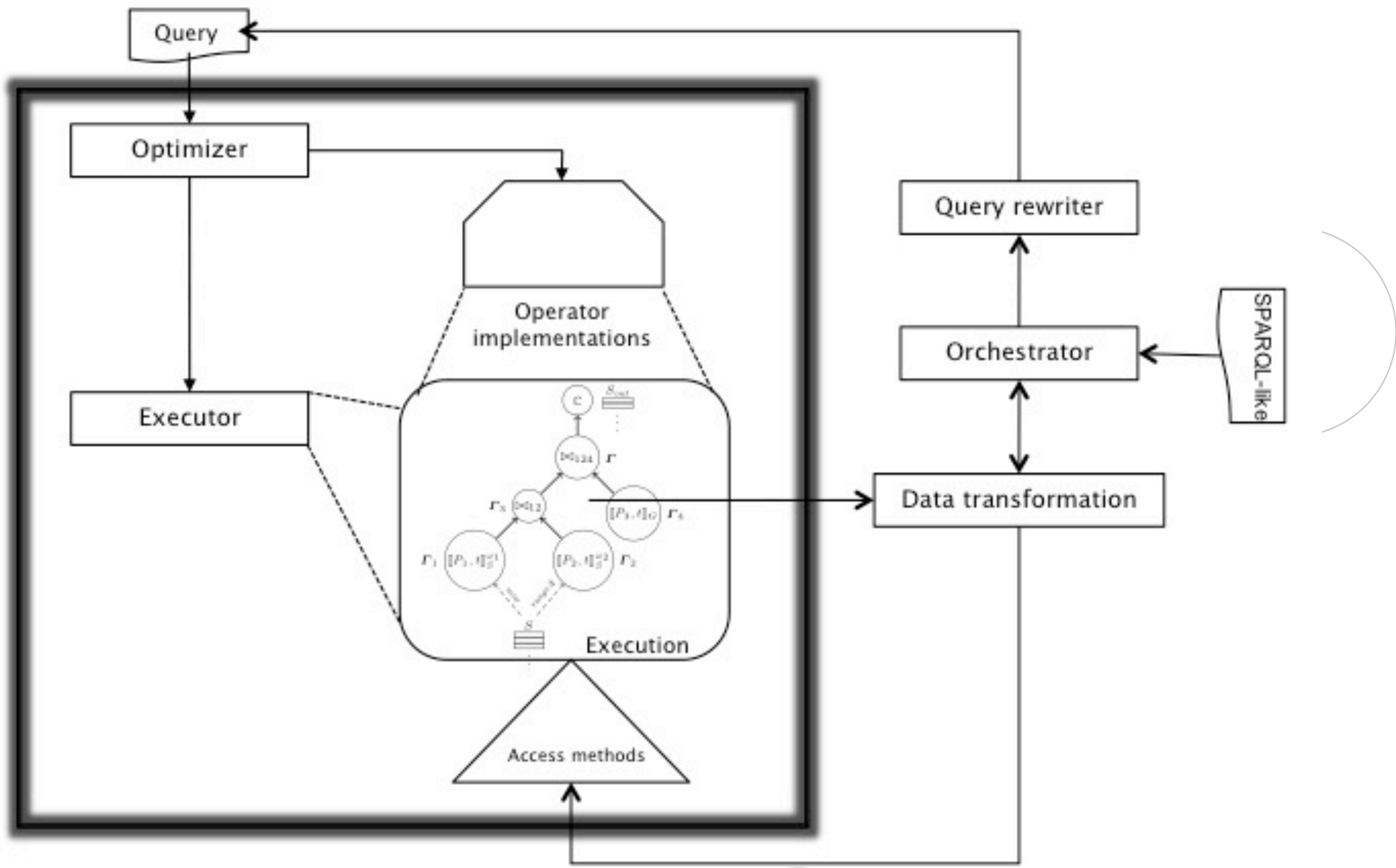
System design & Architectures

- Current available systems can be classified into two categories based on their architecture design
- White box architecture
  - Implements all required components
    - physical operators (e.g. windows, join, triple pattern matching)
    - data structures (e.g. B+-Trees, hashtables)
    - query generator/optimizer/executor
- Black box architecture
  - Uses existing RDF and data stream processing systems as sub-components
  - Query rewriter, data translator and orchestrator among sub-components is needed
- Black box easier to implement, but no full-control and data transformation overhead

White box


Query
Optimizer
Executor
Nested-loop or hash joins
Aggregation
Sliding windows
Operator implementations
Execution
BTree++,
hash table
Triple-based
indices
Access methods
database
datastream

Black box


Query
Optimizer
Executor
Operator implementations
Execution
Access methods
Query rewriter
Orchestrator
SPARQL-like
Data transformation

Continuous Semantics

- Current Linked Stream Data approaches follow/reuse operators from relational DSMS
- Continuous query
  - Q(t): query results up to time t
  - R(t) : unordered bag of tuples (relations) at time instant t
  - Relation R: sequence R = [R(t)], $t \in N$, ordered by t.
- Query algebra
  - Stream-to-stream (Streaming SPARQL): stream-to-stream operator
  - Mixed (C-SPARQL, $\text{SPARQL}_{\text{Stream}}$, CQELS): stream-to-relation, relation-to-relation and relation-to-stream operators

Query Algebra

- Stream-to-stream operator
  - One-time queries in SQL that are continuously executed
- Relation-to-relation operator
  - As in traditional relational DBMS
- Stream-to-relation operator → Windows
  - Time-based (e.g. last 3 secs)

Timestamp (sec)	Person	Loc
1001	1	loc1
1000	4	loc1
999	5	loc2
998	6	loc1
997	7	loc1

Query Algebra

- Stream-to-stream operator
  - One-time queries in SQL that are continuously executed
- Relation-to-relation operator
  - As in traditional relational DBMS
- Stream-to-relation operator ➔ Windows
  - Time-based (e.g. last 3 secs)
  - Tuple-based (e.g. last 4 tuples)

Timestamp (sec)	Person	Loc
1001	1	loc1
1000	4	loc1
999	5	loc2
998	6	loc1
997	7	loc1

Query Algebra

- Stream-to-stream operator
  - One-time queries in SQL that are continuously executed
- Relation-to-relation operator
  - As in traditional relational DBMS
- Stream-to-relation operator ➔ Windows
  - Time-based (e.g. last 3 secs)
  - Tuple-based (e.g. last 4 tuples)
  - Partitioned (e.g. Loc last tuple)

Timestamp (sec)	Person	Loc
1001	1	loc1
1000	4	loc1
999	5	loc2
998	6	loc1
997	7	loc1

Query Algebra

- Relation-to-stream operator: produces a stream from relation R
  - Istream (insert stream): add element <s,t> whenever s is in R(t)—R(t-1)
  - Dstream (delete stream): add element <s,t> whenever s is in R(t-1)—R(t)
  - Rstream (relation stream): add element <s,t> whenever s is in R at time t.

Istream

```
SELECT Istream(*)
FROM RFIDstream [RANGE Unbounded]
WHERE signalstrength>=85
```

Rstream

```
SELECT Rstream(*)
FROM RFIDstream [NOW]
WHERE signalstrength>=85
```

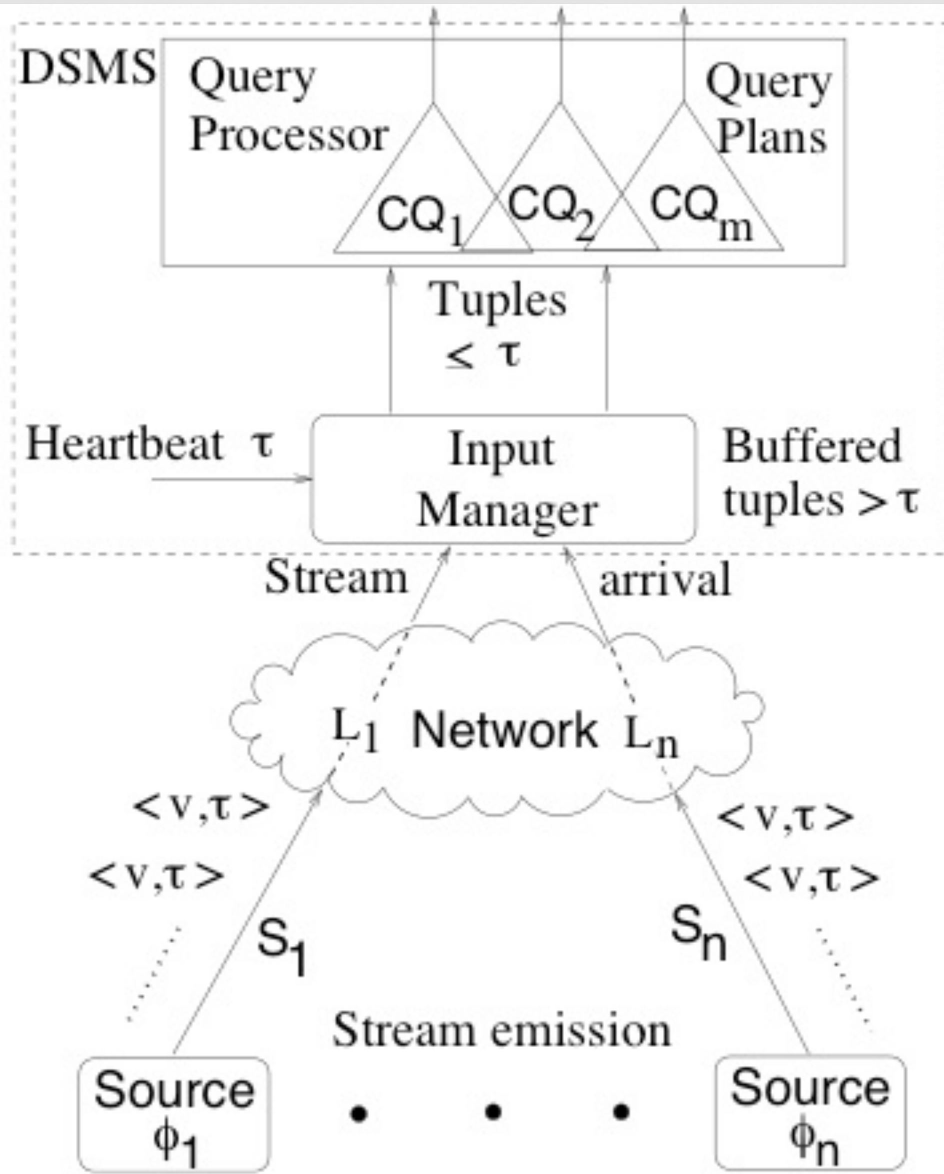
Dstream

```
SELECT Dstream(tagid) FROM RFIDstream [60 seconds]
```

Time Management

- Timestamps are necessary to order stream elements
- Application timestamp (source) vs. system timestamp (DSMS)
- Input manager: buffers to order tuples, ensure they are processed in order
  - Heartbeat (timestamp)
  - Punctuation (pattern)

Time Management

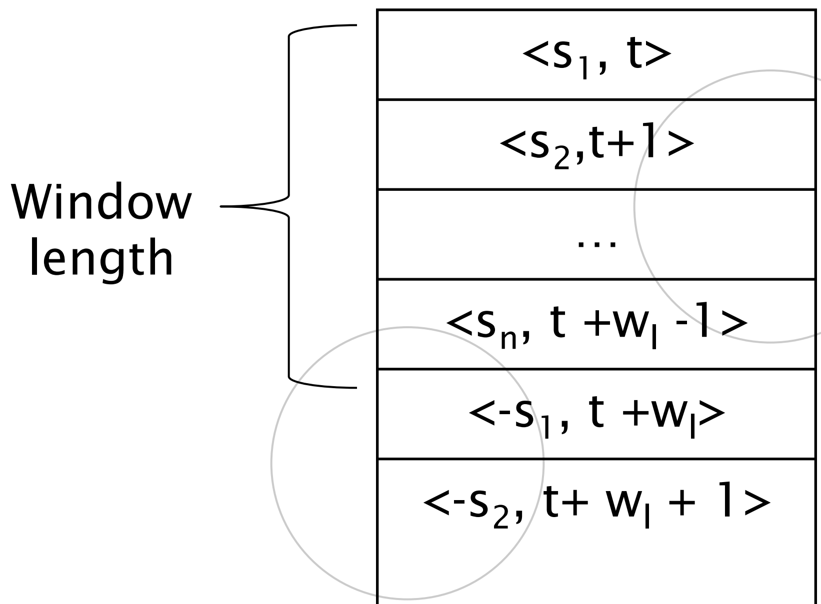


Query Evaluation

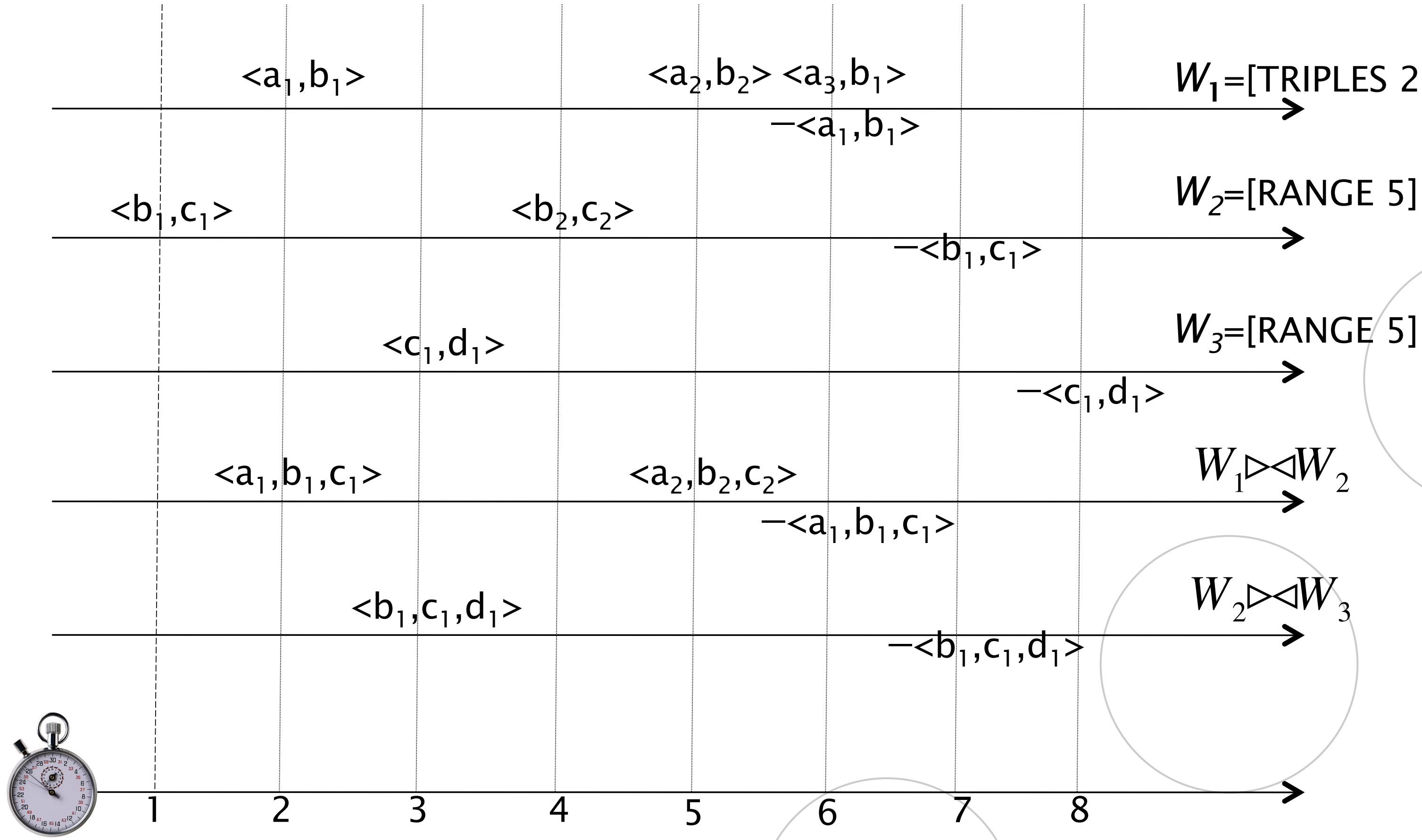
- Eager re-evaluation vs. period re-evaluation
  - Eager: too expensive if update rate is high
  - Periodic: might cause stale results
- Query evaluation needs to handle two types of events
  - Arrival of new stream elements
  - Expiration of old stream elements
  - Action upon events vary across operators, e.g. an arrival might generate a new result (join) or trigger the removal of an existing result (negation)

Query Evaluation

- Arrivals are triggered by stream source
- Expiration needs to be handle by the query processor
  - Timestamp
  - Negative tuple: for a window of length w_l, a tuple inserted at time t will generate a negative tuple at time $t+w_l$

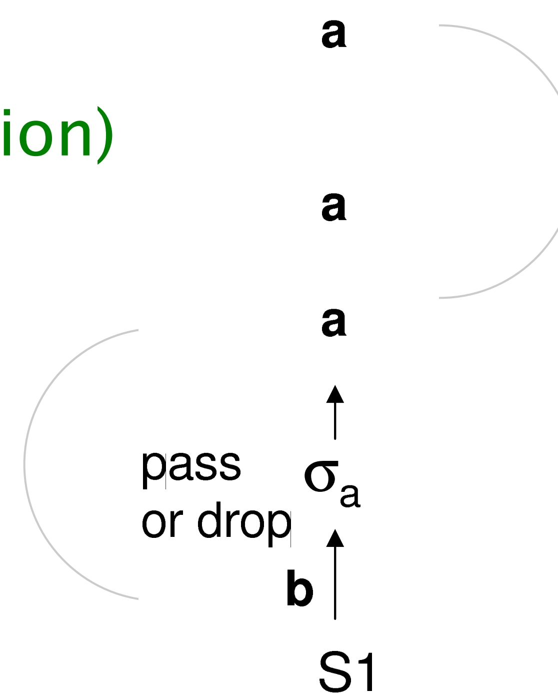


Adding and evicting stream elements



Query Evaluation

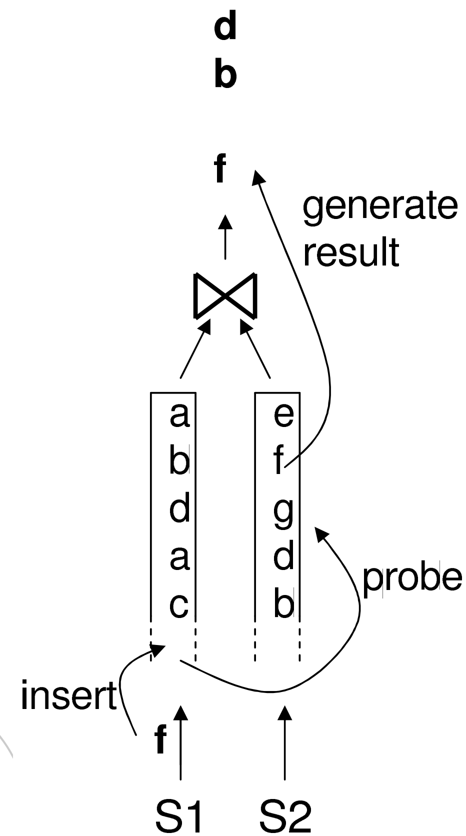
- Stateless operators: processed "on the fly" (directly on stream)
  - E.g. Selection, union.
- Stateful operators: need to maintain processing states (probed at re-evaluation)
  - E.g. window join, aggregation, duplication elimination, non-monotonic operators



Selection

Query Evaluation

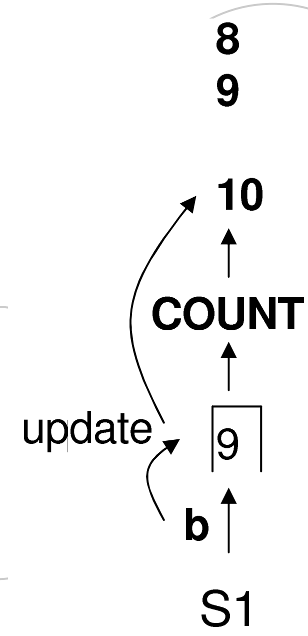
- Window join : new arrival in one input triggers probing on the other input



Window join

Query Evaluation

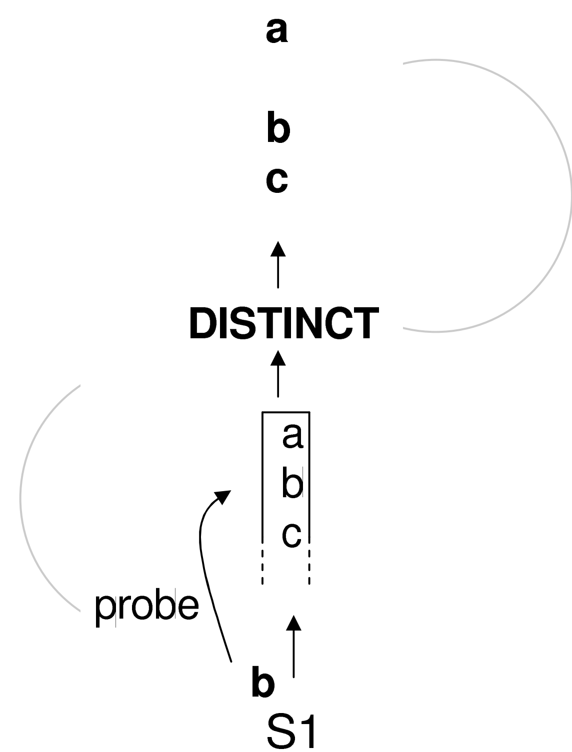
- Aggregation
  - Expirations must be dealt with immediately
  - Time and space requirements depends on the aggregation function
- Distributive aggregates
  - Computed incrementally, constant time/space requirements
  - E.g. COUNT, SUM, MAX, MIN
- Algebraic aggregates
  - Computed using values from distributive aggregates. Constant time/space requirements
  - E.g. AVG (SUM/COUNT)



Aggregation

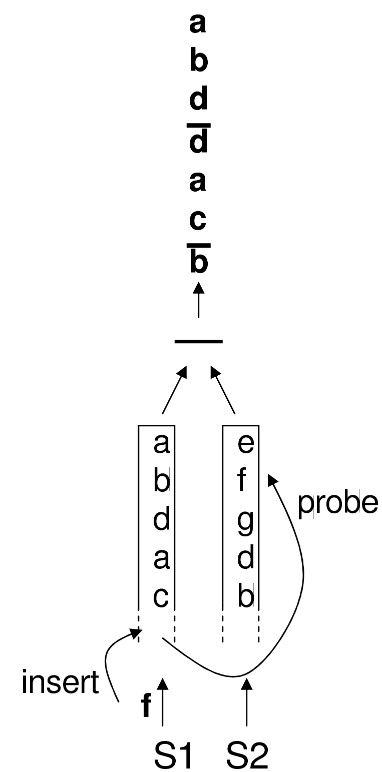
Query Evaluation

- Holistic aggregates: space consumption linear to input sizes
  - E.g. TOP-k, COUNT DISTINCT
- Duplicate elimination
  - Distinct values are kept
  - Expirations are handled eagerly



Duplicate elimination

- **Non-monotonic operators**
  - Previous results removed when they no longer satisfy query
  - E.g. negation
  - Negative tuples can be used



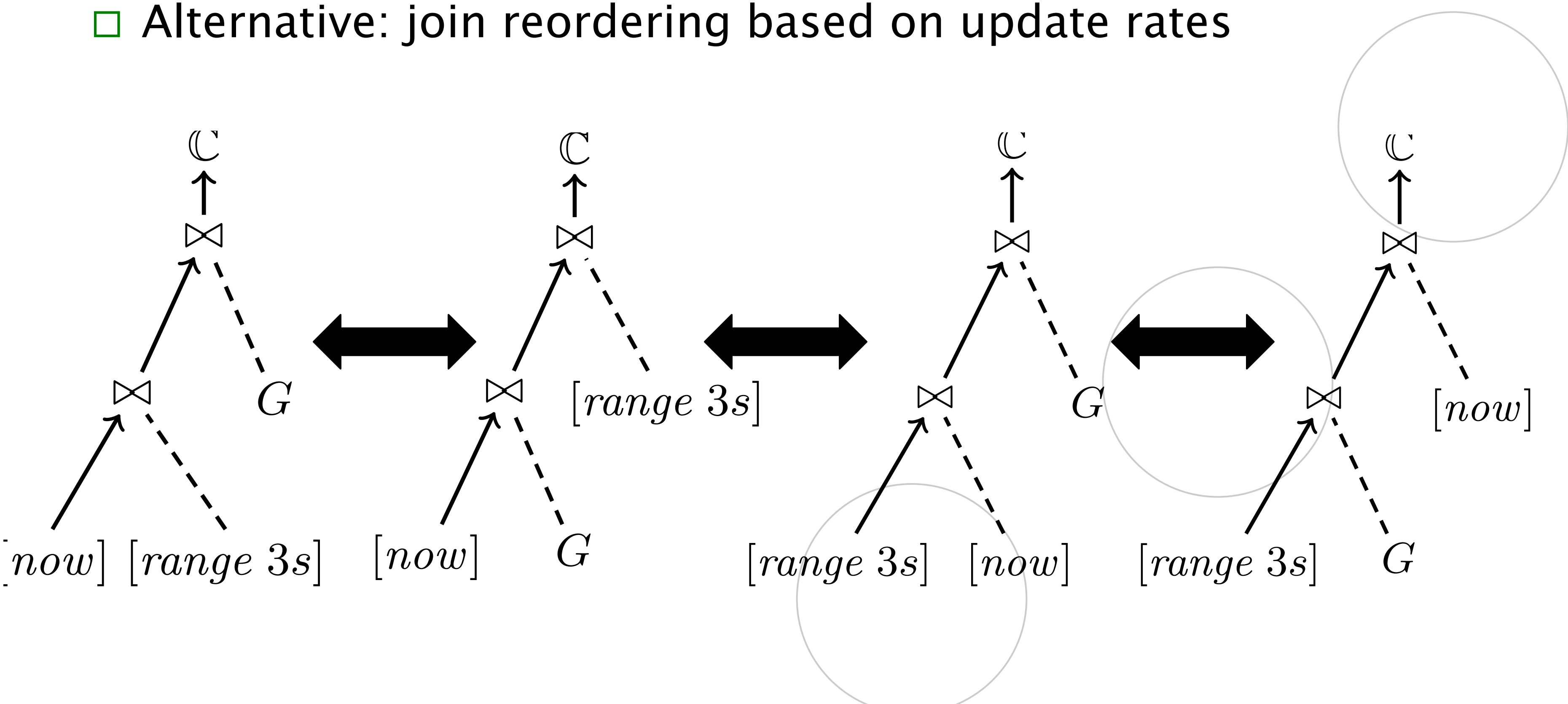
Negation

Memory Overflow

- Some join operators already handles memory overflow by sending input partitions to disk.
- Use of secondary storage requires indexes
  - Expensive under high update rates
- Alternative: Partition the data to make updates "local"
  - Sort tuples chronologically
  - Inserts in newer partition only
  - Deletes in older partition only
  - Problem: search is not efficient. Assumes insertion/ expiration order is the same
    - Sub-indexes
    - Doubly partitioned indexes

Query Optimization

- Re-arrange query operators for more efficient execution
  - Traditional selectivity estimates can't be applied
  - Alternative: join reordering based on update rates



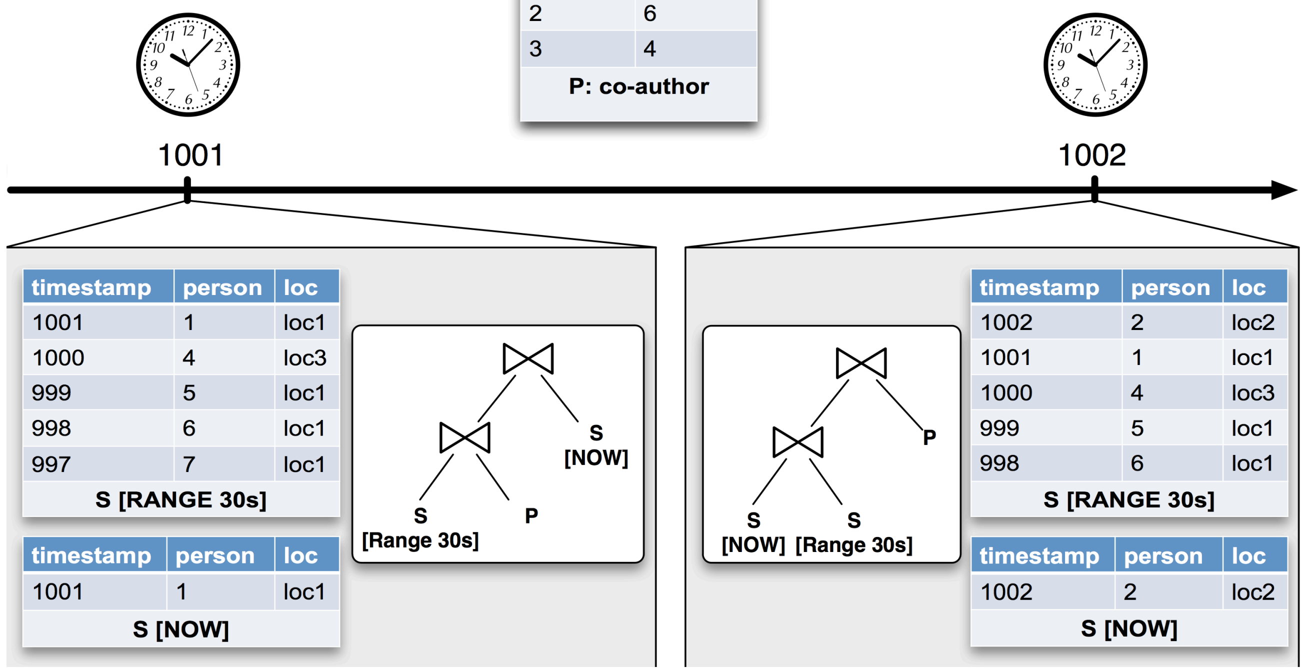
Query Optimization

- Adaptivity is key!
  - Processor must be able to reorder query operators on the fly
  - Changes in:
    - operator costs (processing time),
    - update rate,
    - input selectivity

"Notify two people who are co-authors of a paper if they are in the same location (within the last 30 seconds)"

person1	person2
2	3
2	4
2	5
2	6
3	4

P: co-author



Time 1001:

timestamp	person	loc
1001	1	loc1
1000	4	loc3
999	5	loc1
998	6	loc1
997	7	loc1

S [RANGE 30s]

timestamp	person	loc
1001	1	loc1

S [NOW]

Time 1002:

timestamp	person	loc
1002	2	loc2
1001	1	loc1
1000	4	loc3
999	5	loc1
998	6	loc1

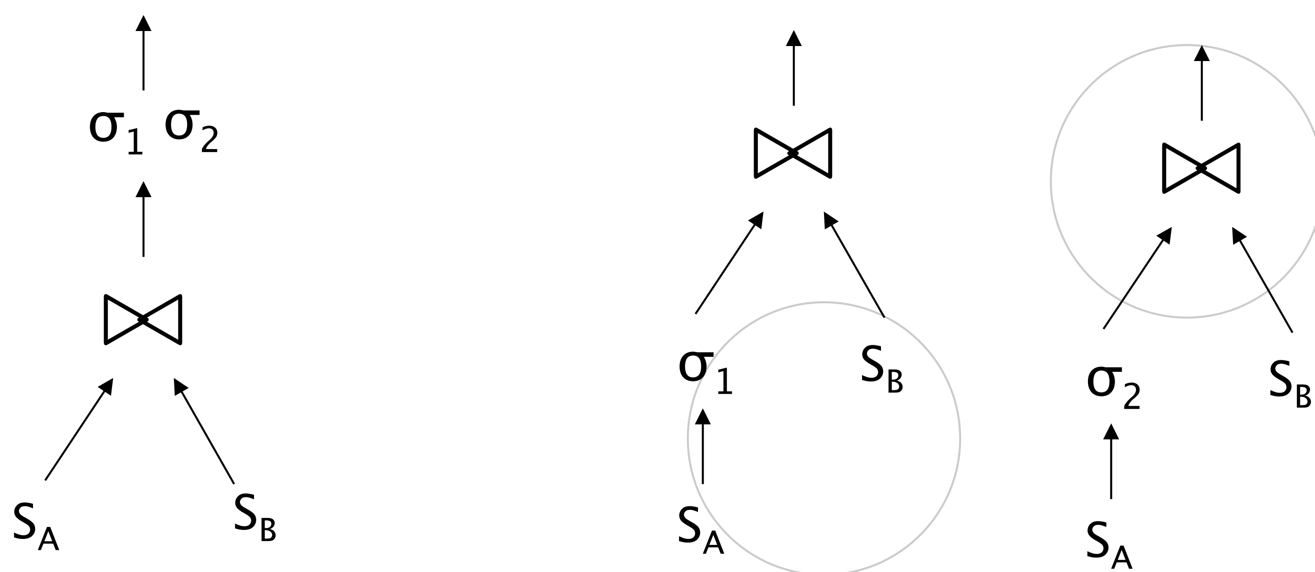
S [RANGE 30s]

timestamp	person	loc
1002	2	loc2

S [NOW]

Query Optimization

- Operators routing (instead of fixed query plan tree)
  - Eddies: estimate which operators are faster/more selective
  - Overhead: migration of internal state of query plan
- Continuous query: multi-query optimization possible
  - Better memory usage
  - Trade-offs exists (e.g. join -> selection vs. selection -> join)



Scheduling

- Data first push into queues, then consumed by operators
- Scheduler determiners which data in which queue to process next
  - Different scheduling strategies (e.g. round robin, arrival time, time slice)
  - Choice depends on factors such as stream arrival patterns, max/avg output latency.