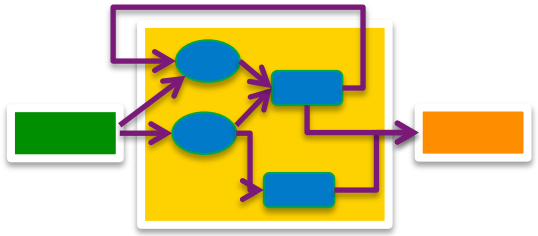
Aalto University
School of Science



Constructing Event Processing Systems of Layered and Heterogeneous Events with SPARQL

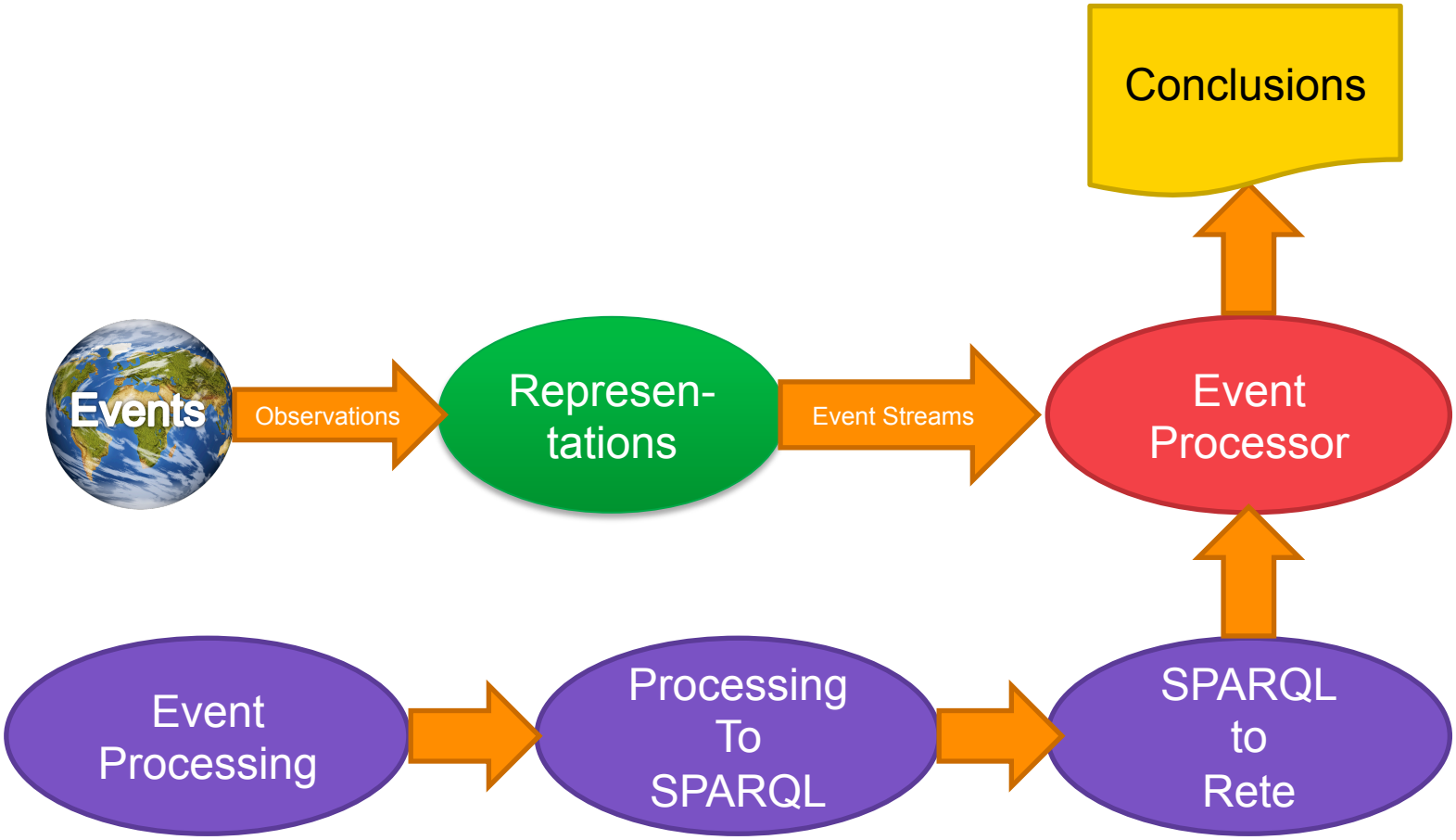
Mikko Rinne, Esko Nuutila

ODBASE 2014, Amantea, Italy
29.10.2014

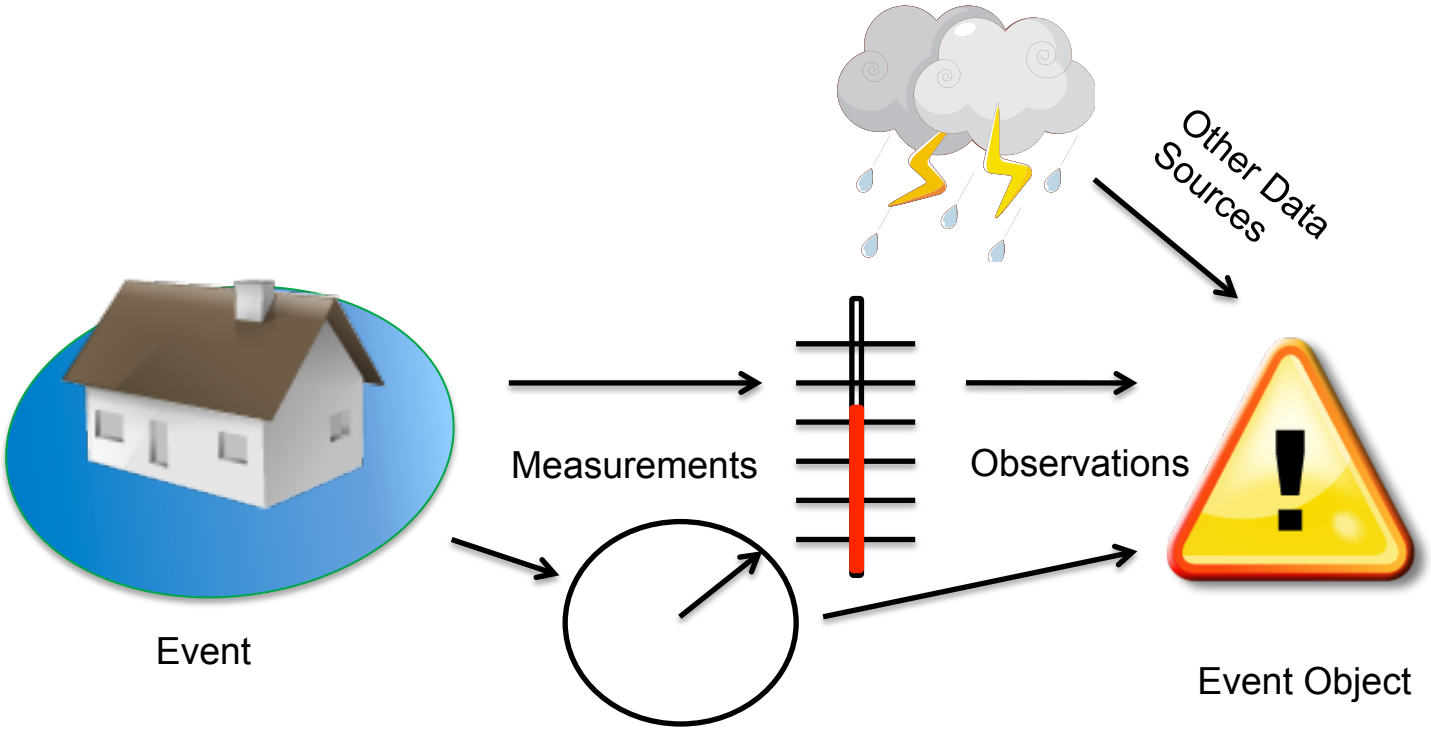
Department of Computer Science and Engineering

Distributed Systems Group

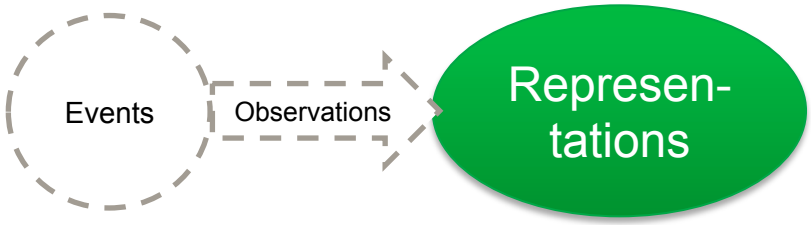
Table of Contents



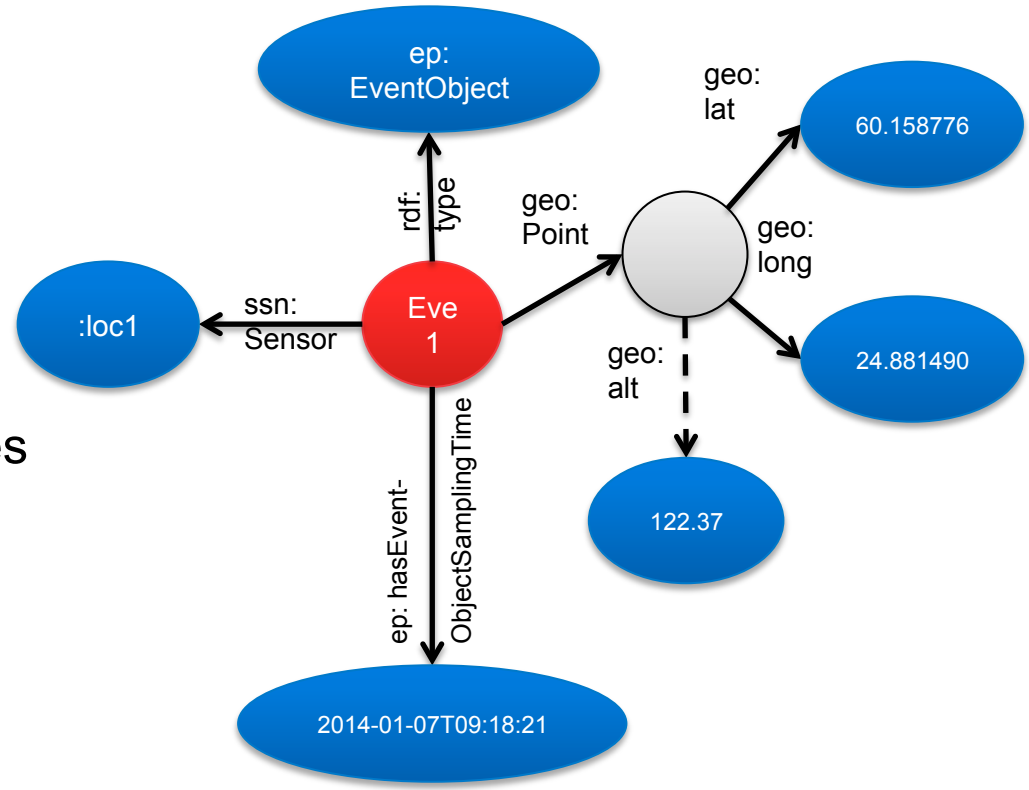
Events, Observations, Event Objects



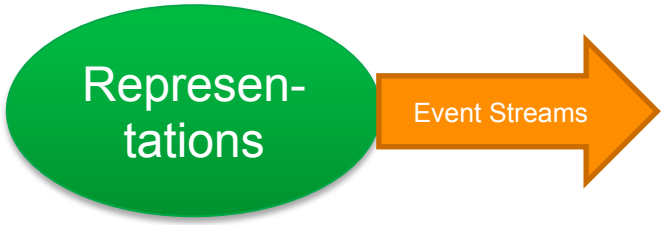
Heterogeneous Event Representations



- Variable event structures in an open environment
  - Different sensors may support different parameters
  - Queries can match the data of interest and disregard the rest
- RDF has flexible support for heterogeneous event structures



Turtle vs. TriG



- Turtle has challenges with event object boundaries*)
  - Grammar rule 6 specifies “triples”, but the order of triples impacts the composition of the block

Block 1

```
_:5 geo:lat 60.158775;
    geo:long 24.88149 .
```

Block 2

```
<Eve1> rdf:type ep:EventObject;
       ssn:Sensor :loc1;
       geo:Point _:5;
       ep:hasEventObjectSamplingTime
         "2014-01-07T09:18:21"^^xsd:dateTime .
```

vs.

Block 1

```
<Eve1> rdf:type ep:EventObject;
       ssn:Sensor :loc1;
       geo:Point _:5;
       ep:hasEventObjectSamplingTime
         "2014-01-07T09:18:21"^^xsd:dateTime .

_:5 geo:lat 60.158775;
    geo:long 24.88149 .
```

- TriG specifies the Turtle format for sending datasets
  - Each event in a separate graph

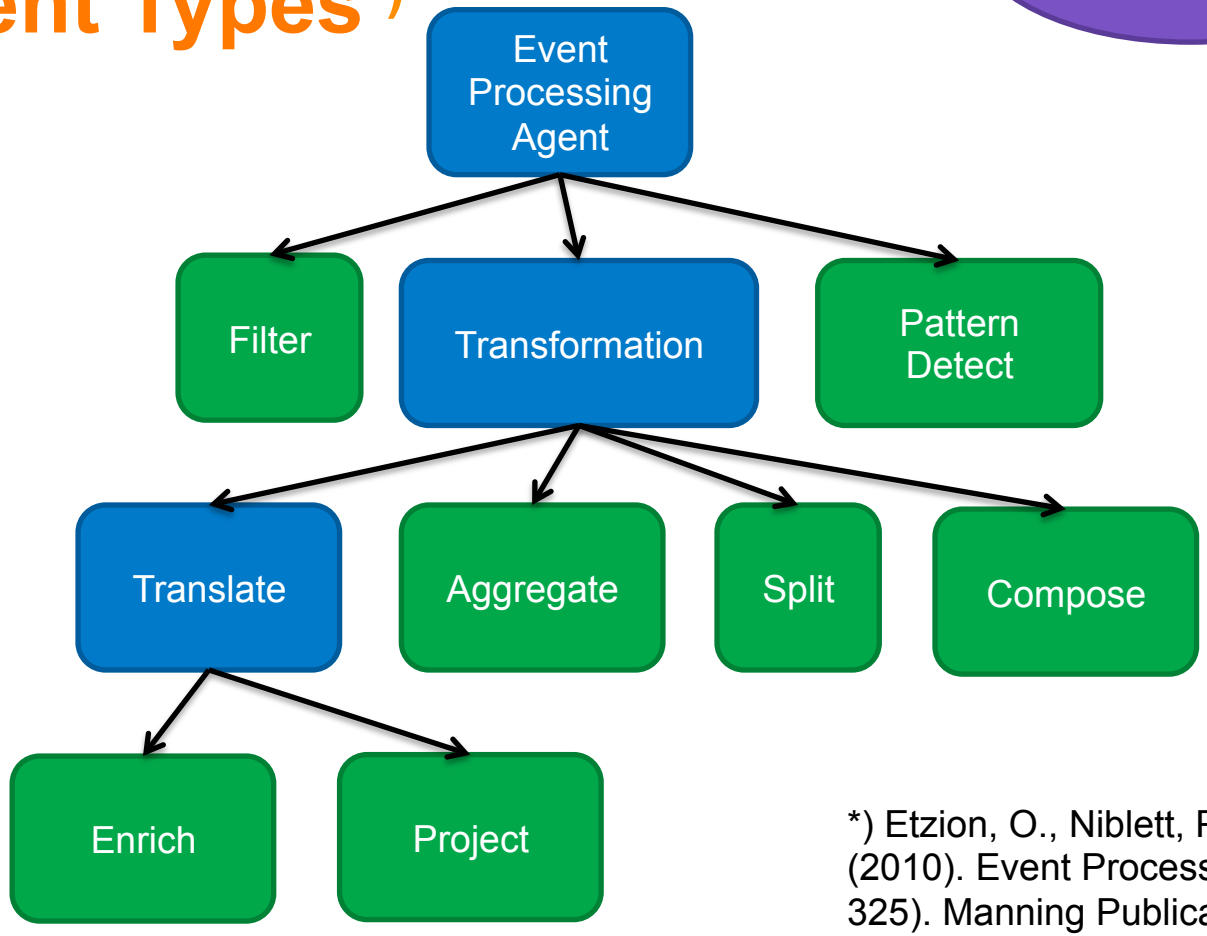
Graph 1

```
<Eve1> { _:5 geo:lat 60.158775;
         geo:long 24.88149 .

       <Eve1> rdf:type ep:EventObject;
              ssn:Sensor :loc1;
              geo:Point _:5;
              ep:hasEventObjectSamplingTime
                "2014-01-07T09:18:21"^^xsd:dateTime . }
```

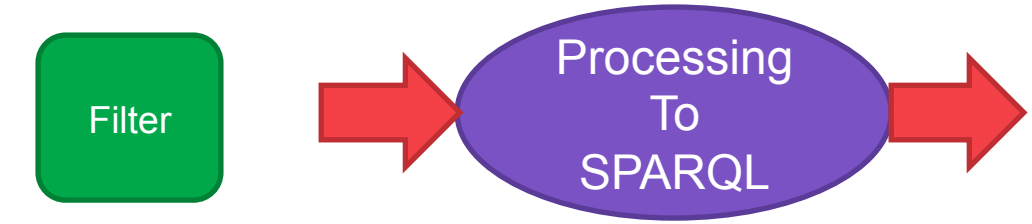

*) Keskisärkkä, R., Blomqvist, E.: Event Object Boundaries in RDF Streams: A Position Paper. In: ISWC 2013 Workshop: Proceedings of OrdRing 2013 - 2nd International Workshop on Ordering and Reasoning. CEUR workshop proceedings, Sydney, Australia (October 2013)

Event Processing Agent Types*)



*) Etzion, O., Niblett, P., & Luckham, D. (2010). Event Processing in Action (p. 325). Manning Publications.

Stateless Filter



- Filters select interesting events or eliminate uninteresting ones.

6. Copy event to output graph

1. Recognize incoming events with graph name

2. Match the whole incoming event

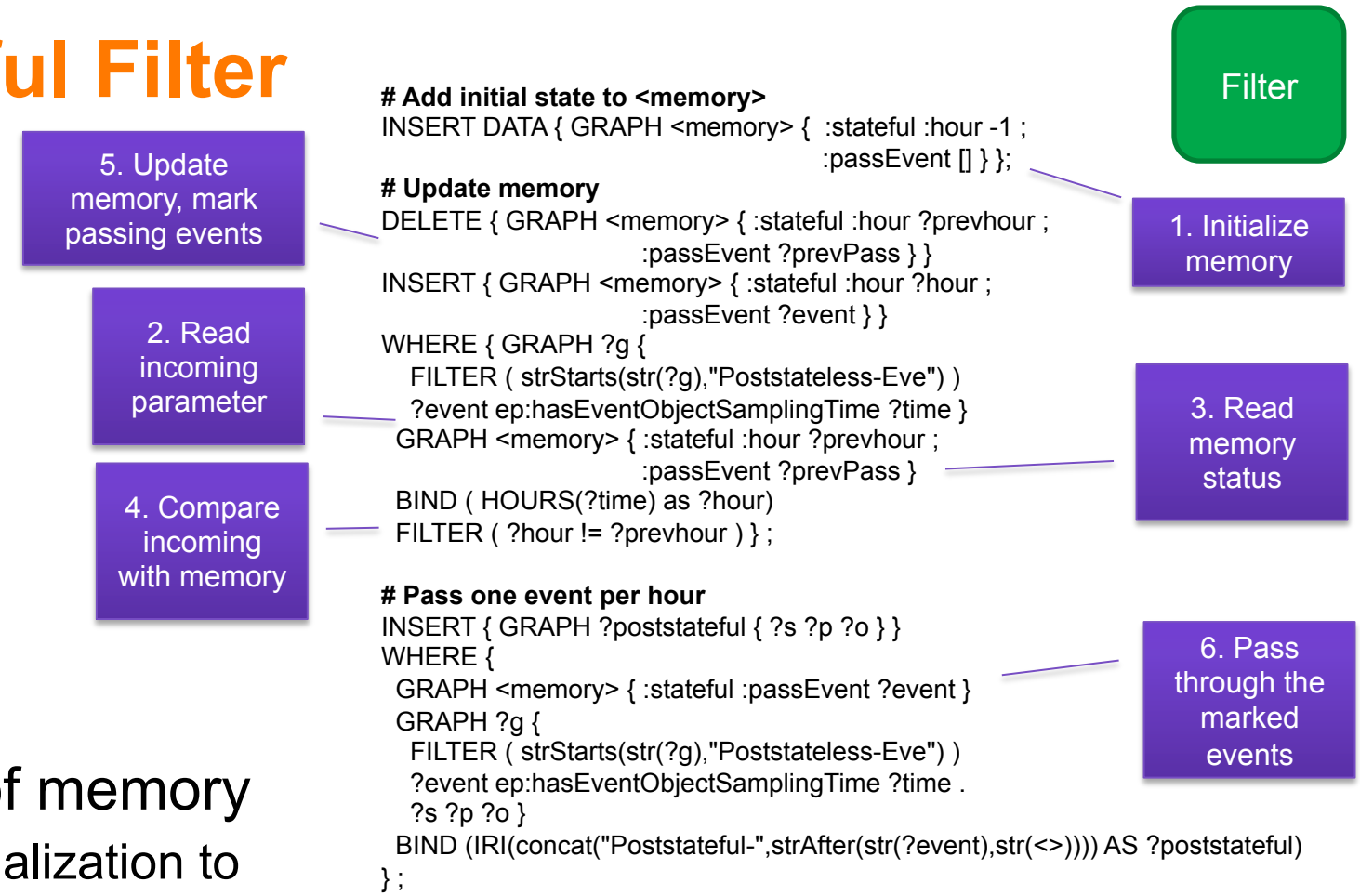
```
INSERT { GRAPH ?poststateless { ?s ?p ?o } }
WHERE { GRAPH ?g { FILTER (strStarts(str(?g),concat(str(<>),"Eve")))
    ?s ?p ?o . ?event ep:hasEventObjectSamplingTime ?time }
  FILTER ( ( HOURS(?time) > 8 ) && ( HOURS(?time) < 17 ) )
  BIND (IRI(concat("Poststateless-",strAfter(str(?event),str(<>))))
                                                AS ?poststateless) }
```

3. Separately match filtering parameter to a variable

4. Select events based on criteria

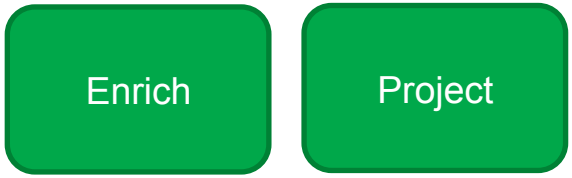
5. Build a name for the output graph

Stateful Filter



- Use of memory
  - Initialization to avoid OPTIONAL clauses

Enrich and Project



- Transformation agents, operate on each event object separately with single-in single-out
- *Enrich* could typically add data from a SPARQL endpoint using a federated SERVICE query
  - Example shown to add location label based on coordinates
- *Project* removes information from the incoming event
  - Match only the desired parts of the event
  - If the complete event structure is not known, unwanted parts can be removed with FILTER

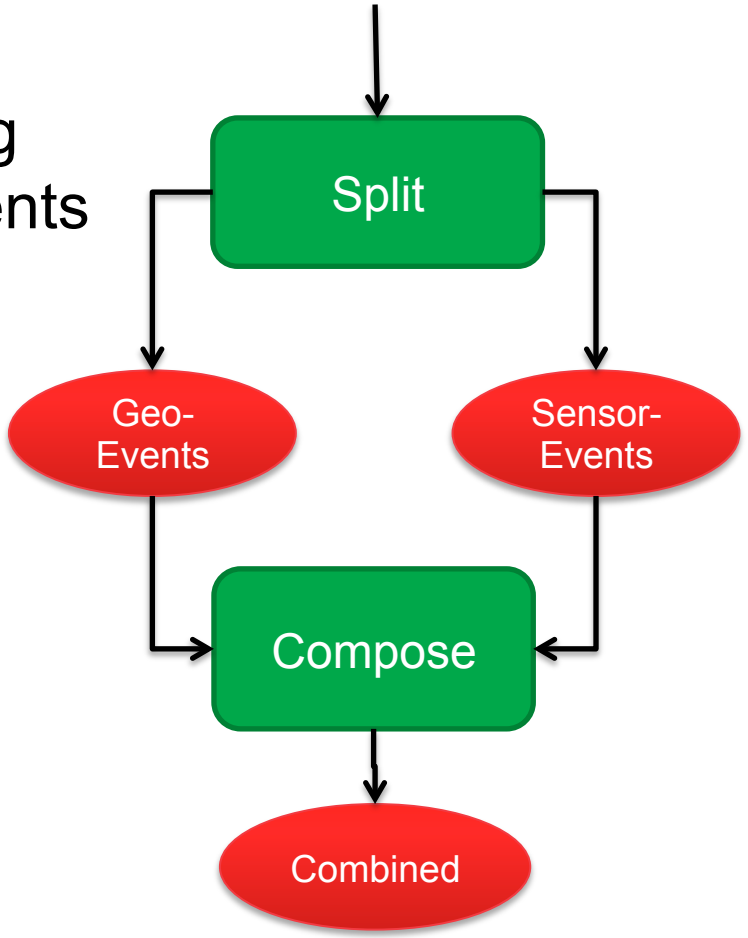
2. Enrich event

```
INSERT { GRAPH ?translated {
   ?event :locationName ?label .
   ?s ?p ?o } }
WHERE { GRAPH ?g {
   FILTER ( strStarts(str(?g),"Poststateful-Eve") )
   ?s ?p ?o . ?event a ep:EventObject ;
      geo:Point [ geo:lat ?lat ; geo:long ?long ; ]
   SERVICE <http://factforge.net/sparql> {
      ?location omgeo:nearby(?lat ?long "1km");
            ff:preferredLabel ?label }
 BIND (IRI(concat("Translated-",
       strAfter(str(?event),str(<>)))) AS ?translated) } } ;
```

1. Find related information

Split and Compose

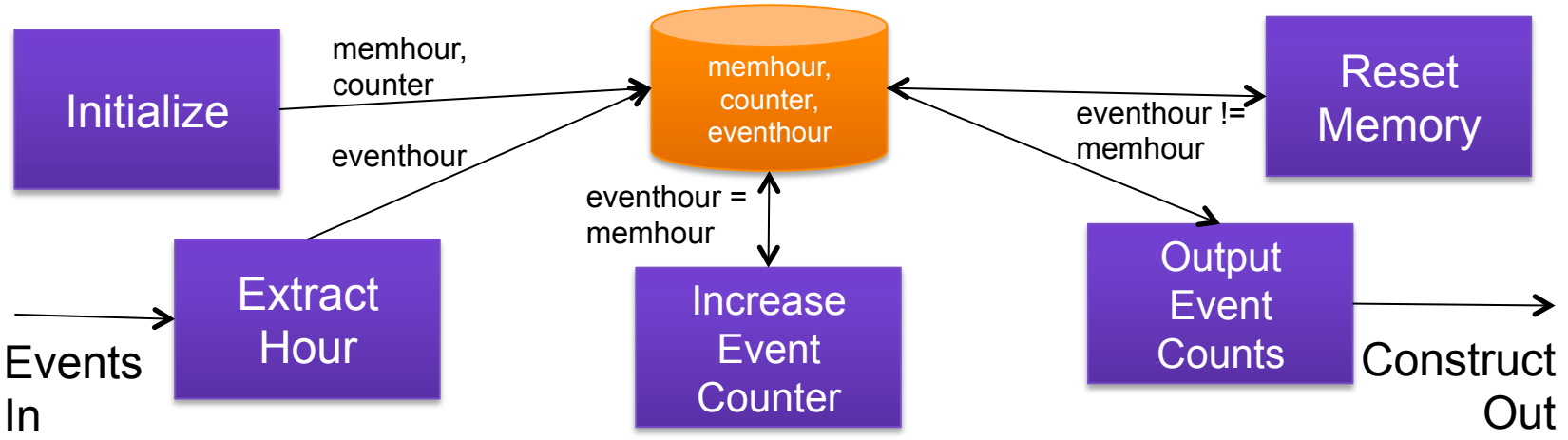
- *Split* partitions a single incoming event into multiple outgoing events
  - Demonstrated by splitting single incoming events into two outgoing events sent into separate event channels
- *Compose* combines multiple incoming streams into a single output streams



Aggregate

Aggregate

- A multiple-in single-out function of incoming events
- Typical examples count, min, max, average, sum
- Example to calculate number of events per hour
  - Uses an auxiliary query to extract data utilized by three other queries



Pattern Detect

Pattern Detect

- Search patterns in incoming events
  - Describe pattern, pass through qualifying patterns or both
- Example to detect patterns of movement
  - Specific to the sensor sending the location

1. Transform points to compass directions

2. Advance pattern index with positive match

3. Reset pattern index when not matching

4. Detect completed pattern, reset index

Runtime Initialization

Pattern Detect

- Other examples have only initialized memory to constants
- In pattern detect memory needs to be initialized to values only known at runtime
  - Previous point for calculation direction
  - Originating sensor of the previous location entry

```
INSERT { GRAPH <memory> {
   [] a :transformPrevPoint ;
      ssn:Sensor ?sensor ;
      geo:Point  [ geo:lat ?lat ; geo:long ?long ; ]  } }
WHERE { GRAPH ?g { FILTER (strStarts(str(?g),concat(str(<>),"Eve")))
   ?event a ep:EventObject ;
         ssn:Sensor ?sensor ;
         geo:Point [ geo:lat ?lat ; geo:long ?long ; ] }
  FILTER NOT EXISTS {
   GRAPH <memory> { ?x a :transformPrevPoint ; ssn:Sensor ?sensor } } } ;
```

Use FILTER NOT EXISTS in initialization rather than OPTIONAL in the actual query

Query Text Repetition

Pattern Detect

- Repetition of completely identical WHERE-clauses
- Needed only because SPARQL does not allow DELETE and CONSTRUCT in the same query

```
# Reset index when pattern is complete
DELETE {  GRAPH <memory> { # Remove last index from memory
    ?mem a :patternIndexEntry ; ssn:Sensor ?sensor ;
        :basedOnEvent ?oldEvent ; :patternIndex ?oldIndex } }
WHERE { GRAPH <pattern> { :pattern :length ?length }
  GRAPH <memory> {
     ?mem a :patternIndexEntry ; ssn:Sensor ?sensor ;
         :basedOnEvent ?oldEvent ; :patternIndex ?oldIndex }
  FILTER ( ?oldIndex = ?length ) } ;

# Output-Result
CONSTRUCT { GRAPH <PatternDetect> {
    [] a ep:EventObject ; :patterndetected "Pattern detected!" ;
      ssn:Sensor ?sensor ; :lastEvent ?oldEvent } }
WHERE { GRAPH <pattern> { :pattern :length ?length }
  GRAPH <memory> {
     ?mem a :patternIndexEntry ; ssn:Sensor ?sensor ;
         :basedOnEvent ?oldEvent ; :patternIndex ?oldIndex }
  FILTER ( ?oldIndex = ?length ) } ;
```

Cleaning Up

- Main graph input can be cleaned by an operational policy after all the related queries have executed
- Events pushed to event channels by INSERT are not automatically cleaned up
  - Currently using separate clean-up queries to DELETE old events from event channels

```
DELETE { GRAPH ?g1 { ?s ?p ?o } }
WHERE { GRAPH ?g1 {
   FILTER ( strStarts(str(?g1),"Poststateless-Eve") )
   ?event1 ep:hasEventObjectSamplingTime ?time1 .
   ?s ?p ?o }
  FILTER EXISTS { GRAPH ?g2 {
   FILTER ( strStarts(str(?g2),"Poststateless-Eve") )
   ?event2 ep:hasEventObjectSamplingTime ?time2 .
    FILTER (?time1 < ?time2)
  } }
} ;
```

2. Pass through and DELETE if a newer event exists

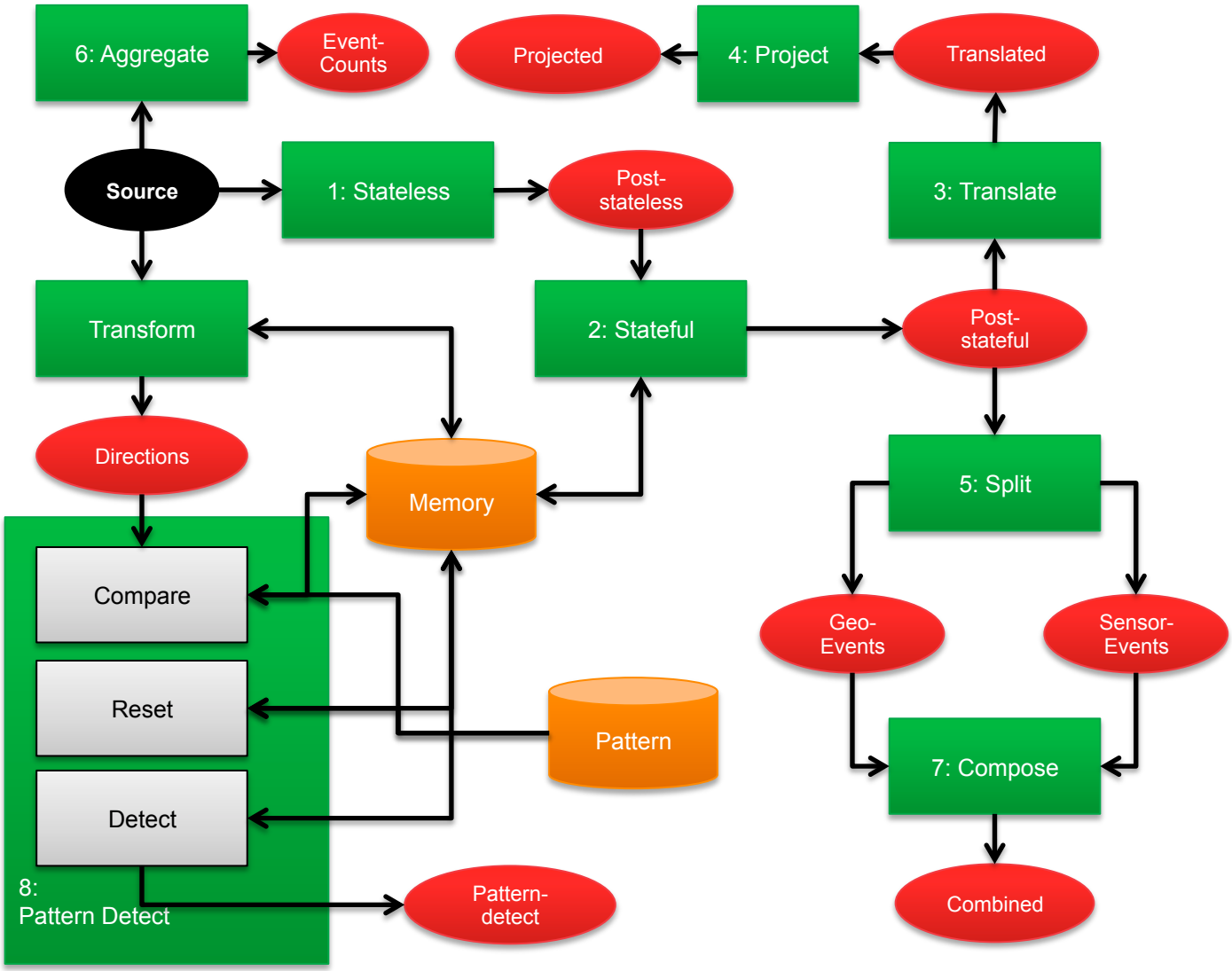
1. Match times for two events

The Issue with OPTIONAL?

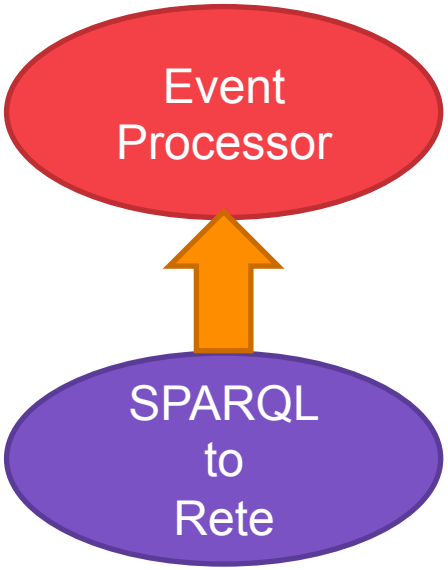
- OPTIONAL is available, but may have unexpected results in a continuously matching SPARQL system
- Unless events are properly handled as blocks (ref. the discussion on Turtle vs. TriG) multiple results per event may be triggered
- Removal of OPTIONAL data will trigger new matches

=> The query network becomes easier to manage when not using OPTIONAL

Putting It All Together



INSTANS



- A platform for translating a network of SPARQL 1.1 queries and update rules into a continuously-executing Rete network
- The complete executable example presented in the paper is available in https://github.com/aaltodsg/instans/tree/master/tests/input/CEP2SPARQL
- The presented query network is fairly complex
  - Demonstrates platform readiness
  - In practice should be modularized so that each Event Processing Agent would run in a separate Rete engine

*) Incremental eNgine for STANding Sparql, http://instans.org/

Conclusions

- All types of event processing agents and input modifiers presented in the referenced literature were successfully implemented with a network of SPARQL queries
  - The query networks should be kept simple by modularizing the event processing agents
- Either an operational policy or an explicit cleanup rule is needed to remove intermediate event objects in a stream processing system
- Event object encapsulation challenges of Turtle can be overcome with TriG
- The need for some straightforward additions to SPARQL was detected
  - Generating dataset output from a query
    - Simple addition to CONSTRUCT using the same format as INSERT
  - Combination of output and update
    - E.g. a combination of DELETE + CONSTRUCT + INSERT is very commonly needed