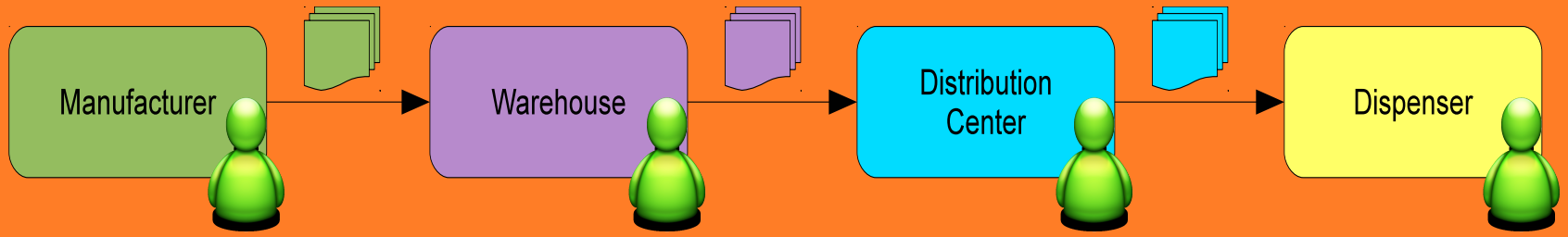
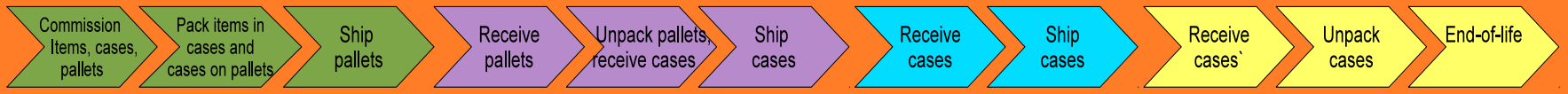
Aalto University
School of Science





RFID-Based Logistics Monitoring with Semantics-Driven Event Processing

Mikko Rinne[1], Monika Solanki[2] and Esko Nuutila[1]

23rd of June 2016

DEBS 2016

[1] *Aalto University*
[2] *University of Oxford*

Scenario: Pharmaceutical Manufacturing

- **Manufacturer process**
  1. Commissioning
     - *An Electronic Product Code (EPC) is assigned to each item*
  2. Packing
     - *Items are packed in cases*
     - *An aggregation identifier (AGE) is assigned to each case*
  3. Shipping
     - *Cases are loaded onto pallets*
     - *Pallets are shipped*

- **A code (EPC or AGE) is scanned at each step by a reader, e.g. RFID**
- **Monitored issues**
  - Lost after commissioning
    - *A commissioned EPC is not found in packing*
  - Counterfeit in packing
    - *An uncommissioned EPC is found in packing*
  - Lost after packing
    - *A packed EPC is not found in shipping*

Event Processing with Semantic Web Methods

- **INSTANS[1] platform based on semantic web technologies**
  - Events encoded in RDF (TriG, an event object = an RDF graph)
  - Queries in SPARQL
- **Same task formulated for Esper[2]**
  - Events encoded in XML (an event object = an XML document)
  - Queries in Esper proprietary *Event Processing Language* (EPL)
- **Quantitative comparison of performance**
- **Qualitative observations of differences**

[1] http://instans.org
[2] http://www.espertech.com

Requirements

- **R1 / Predictability and consistency:** The same expected result shall be consistently produced by all platforms on any number of repeated executions of the same experiment on any suitable execution hardware.
- **R2 / Stream time synchronisation:** Any time filtering operations shall be synchronised to a time reference in the event stream, not the internal clock of the host computer, so that the result is independent of the speed of execution. A consequence of **R1**.
- **R3 / Stability:** No memory residue shall be accumulated from processed events.
- **R4 / Memory efficiency:** Intermediate storage shall be done efficiently so that a maximum number of events can be buffered in a given computing environment.
- **R5 / Performance:** Meet or exceed an average rate of 2.83 EPC/s (based on interviews of people in the pharmaceutical industry quoting manufacturer production figures up to 102,000 EPCs during a 10-hour day).

Differences between INSTANS and Esper

Property	INSTANS	Esper
Programmed using	Common Lisp	Java
Execution environment	Shell or Lisp	Java library (wrapped in our Scala application “XEvePro”)
Query language	SPARQL 1.1 Query & Update	EPL
Operation principle	Rule network	Event-driven
Input formats	RDF as turtle, ntriples, TriG and nquads	POJO, Map, Object-array, XML DOM and Ext
Output formats	turtle, ntriples, TriG, nquads, SRX and CSV	EventBean objects, XML, JSON, CSV

Example Events (Commissioning)

```
context:event0 {
  eve:eve0 a eem:ObjectEvent ;
    eem:hasEventID "eve0" ;
    eem:eventOccurredAt "2015-04-16T09:14:59.395+01:00"
                                        ^^xsd:dateTime ;
  rdr2:reader102 a eem:Reader ;
    eem:logicalID "reader102" .
  eve:eve0 eem:recordedByReader rdr2:reader102 ;
    eem:action eem:ADD ;
    eem:hasBusinessStepType cbv:commissioning ;
    eem:hasDisposition cbv:active ;
    eem:eventRecordedAt "2015-04-16T09:14:59.395+01:00"
                                        ^^xsd:dateTime ;
    eem:associatedWithEPCList _:node19j0kn68ix1 .
  _:node19j0kn68ix1 a eem:SetOfEPCs ;
    co:element epc:030001.0012345.10000001001 , ... ,
      epc:030001.0012345.100000010020 . }
```

Turtle / TriG

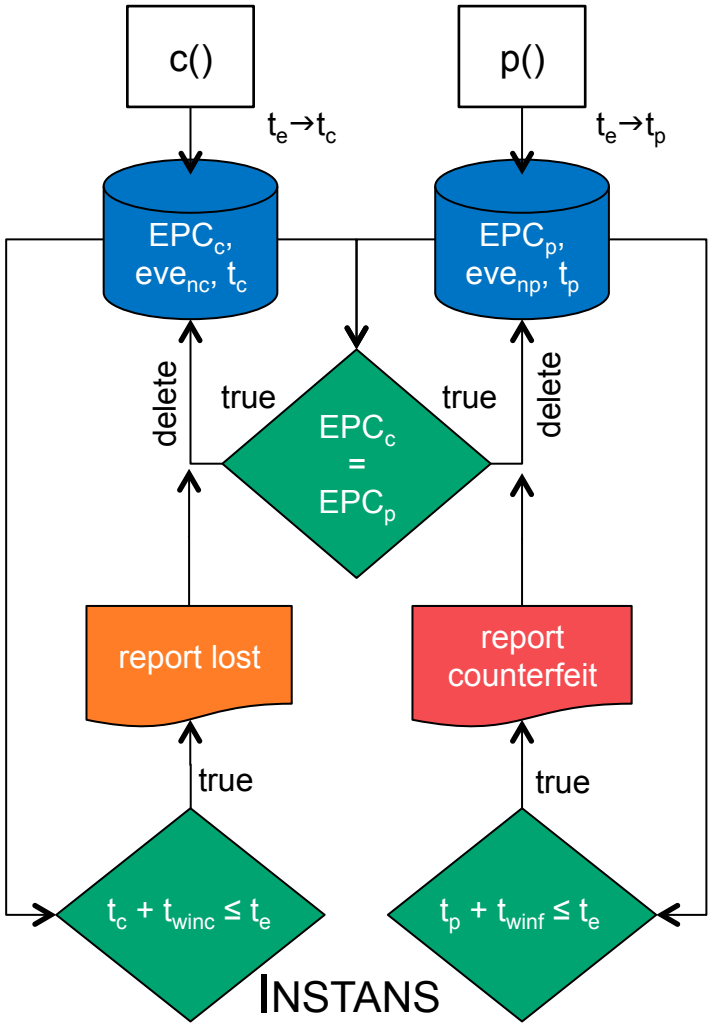
```
<eem:ObjectEvent xmlns:eem="http://purl.org/FIspace/eem#"
                 xmlns:cbv="http://purl.org/FIspace/cbv#">
  <hasEventID>0</hasEventID>
  <eventOccurredAt>2015-04-16T09:14:59.395+01:00</eventOccurredAt>
  <recordedByReader>
    <logicalID>102</logicalID>
  </recordedByReader>
  <action resource="eem:ADD" />
  <hasBusinessStepType resource="cbv:commissioning" />
  <hasDisposition resource="cbv:active" />
  <eventRecordedAt>2015-04-16T09:14:59.395+01:00</eventRecordedAt>
  <associatedWithEPCList xmlns:epc="http://fispace.aston.ac.uk/
                                    pharmaCo1/data/epc/id/sgtin/">
    <element resource="epc:030001.0012345.10000001001" /> ...
    <element resource="epc:030001.0012345.100000010020" />
  </associatedWithEPCList>
</eem:ObjectEvent>
```

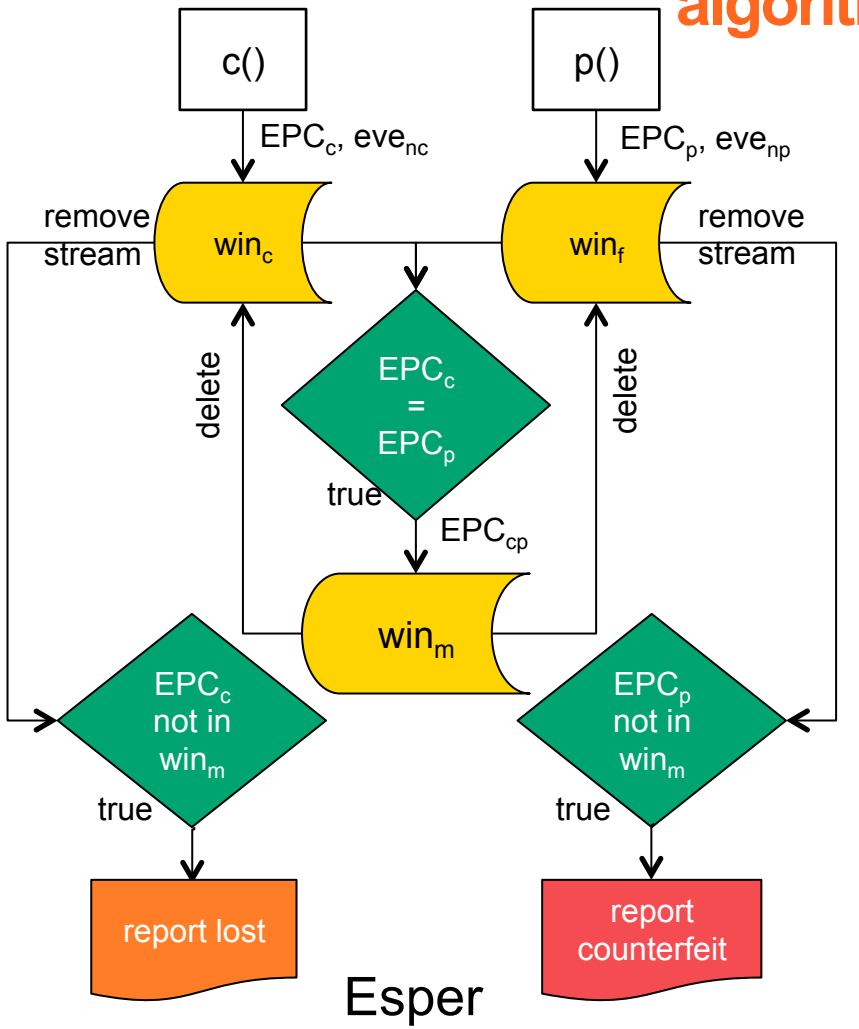
XML

- Automatic conversion from TriG to RDF/XML breaks the event to three entities
  - Manually generated XML schema
- Our TriG files are 33% smaller (depends on choices in label lengths)

Aalto University
School of Science

Solution algorithms





Aalto University
School of Science

Example Queries to Extract EPCs from Commissioning Events

```
DELETE { GRAPH ?g { ?elist co:element ?epc } }
INSERT { ?epc :commissionedEPC ?e ;
              :commissionedInSeconds ?current_sec }
WHERE { GRAPH ?g {
    ?e a eem:ObjectEvent ;
       eem:hasBusinessStepType cbv:commissioning ;
       eem:action eem:ADD ;
       eem:hasDisposition cbv:active ;
       eem:associatedWithEPCList ?elist .
    ?elist co:element ?epc }
  :latest :eventInSeconds ?current_sec } ;
```

SPARQL Update

```
insert into CommissioningEPCs
select hasEventID, resource as EPCNumber
  from ObjectEvent(
    hasBusinessStepType.resource="cbv:commissioning",
     action.resource="eem:ADD",
     hasDisposition.resource="cbv:active")
   [select hasEventID, resource
    from associatedWithEPCList.element];
```

EPL

Stream Time Synchronisation (R2)

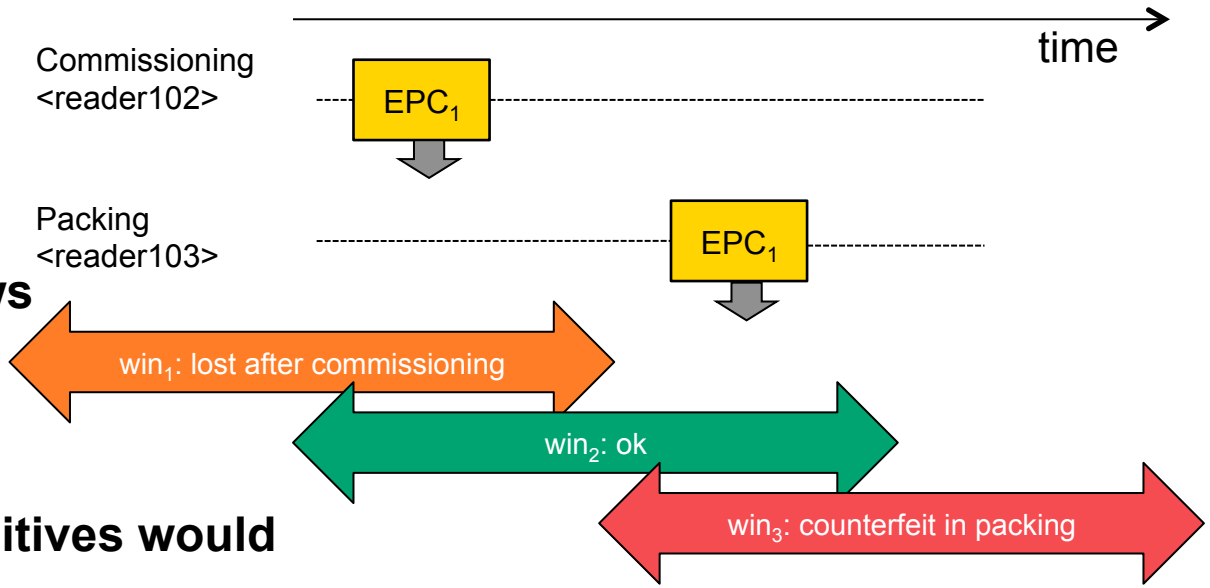
- **Clocks synchronised to the recorded event stream**
  - Use value of *eventOccurredAt*
  - Time has to progress after end-of-file to trigger detection
    - *Otherwise last anomalies will not be detected*
- **Esper**
  - Internal timer disabled
  - *CurrentTimeEvents* sent based on incoming stream
  - End-of-file detected by file reader, time moved forward
- **INSTANS**
  - A custom current time triple updated with rules in the main graph
  - All other time-dependent queries match said triple
  - Another input file with an end-marker sent after recorded stream

Why INSTANS and Esper?

1. **Events with nested structures**
   - Available RDF-based implementations at time of testing worked on timestamped triples
     - *RDF stream processing community group working on a common specification for timestamped graphs*
   - Distributed stream computing platforms typically run on tuples
2. **Query-based processing**
3. **Free license permitting result publication**
   - Most query-based non-RDF complex event processing platforms are commercial offerings
     - *Esper is available with a community license, which permits publication of results*
4. **Event patterns**
   - Most RDF stream processing platforms are based on the *data stream management system* (DSMS) paradigm with stream-level windowing

Stream Windows vs. Event Patterns

- **Non-overlapping (tumbling) windows would produce random results**
- **Overlapping windows catch both correct results and false positives**
- **Cancelling false positives would require**
  - Post-processing with inter-window memory
    - *Auxiliary streams*
    - *Persistent states specified in TEF-SPARQL[1)] (no implementation available)*
  - Restrictions for the time position of a match within the window



[1)] Gao, S., Scharrenbach, T., & Bernstein, A. Running out of Bindings? Integrating Facts and Events in Linked Data Stream Processing. CEUR Vol-1488.

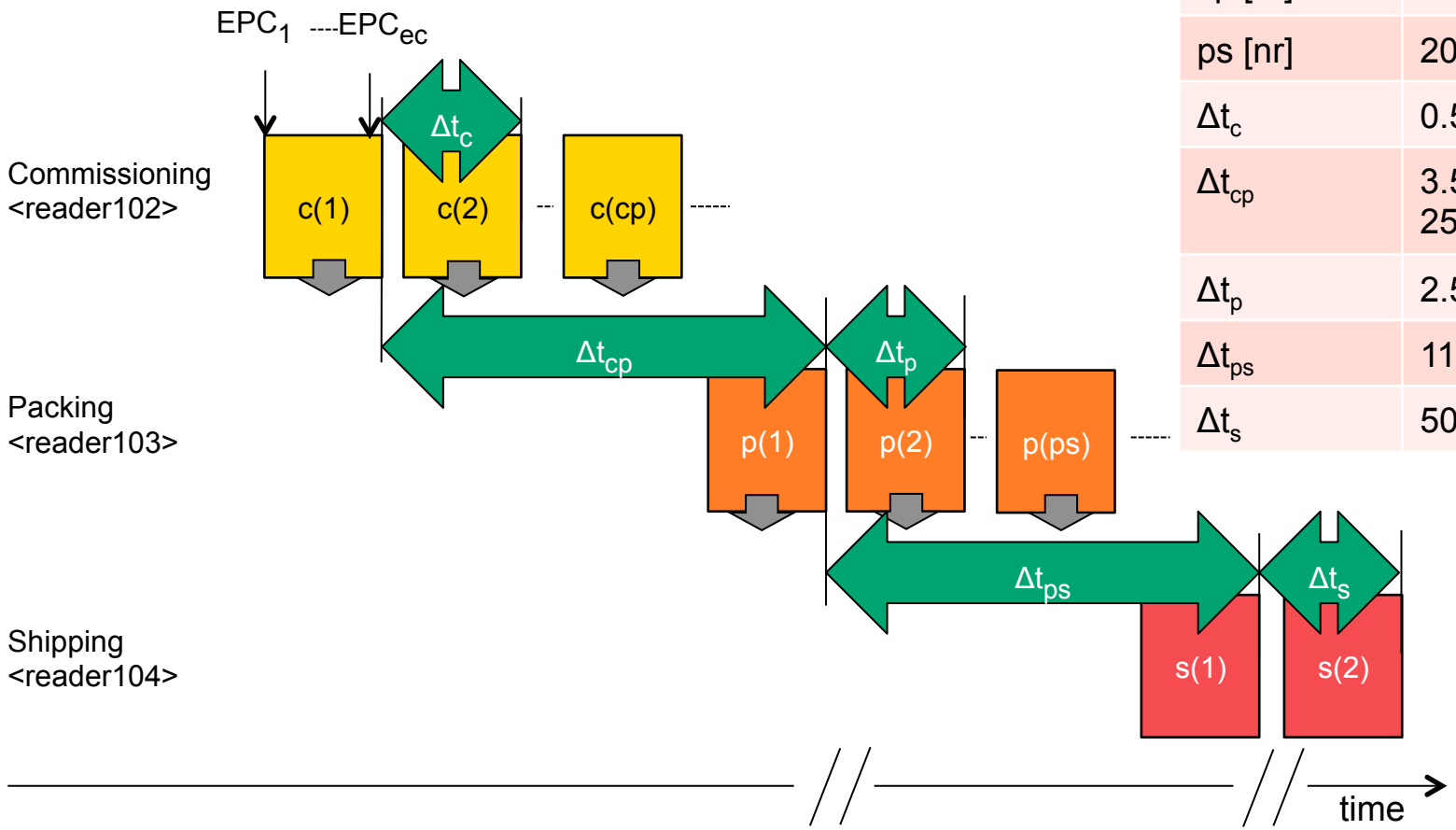
Experimental Setup

- **TriG and XML input files generated separately (no run-time conversion)**
- **Identical output confirmed by *diff* on sorted CSV output files**
- **Execution time measured as the median time of three last runs in a batch of four**
  - Output to /dev/null to minimize I/O impact
- **Memory consumption monitored with *top* using 1 second sample intervals**
- **Complete datasets, queries, results, execution scripts and instructions for repeating the experiment available[1)]**

[1)] https://bitbucket.org/aaltodsg/manufacturer-logistics-monitoring

Event Timing Diagram

Parameter	Values
ec [nr]	20
cp [nr]	5
ps [nr]	20
Δt_c	0.5 s
Δt_{cp}	3.5 – 25,000 s
Δt_p	2.5, 3 s
Δt_{ps}	110 s
Δt_s	50 s

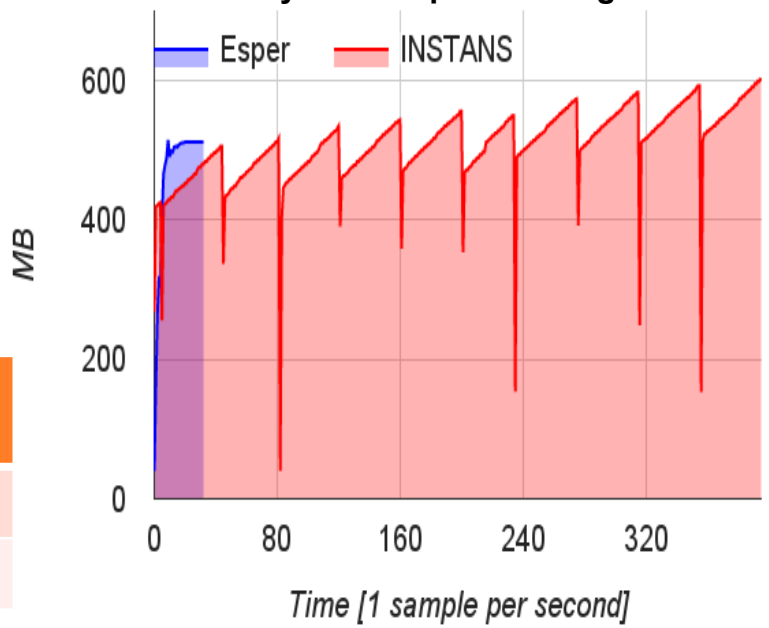


E1: Speed and Memory Stability

- **Commissioning to packing only (E1-4)**
- **Buffering kept low**
- **Esper up to ~15x faster**
- **INSTANS memory consumption shows a slight upward trend**

Parameter	Values
Δt_{cp}	3.5 s
t_{winc}	5 s
Counterfeit EPC	8%

Memory consumption during 1M EPC



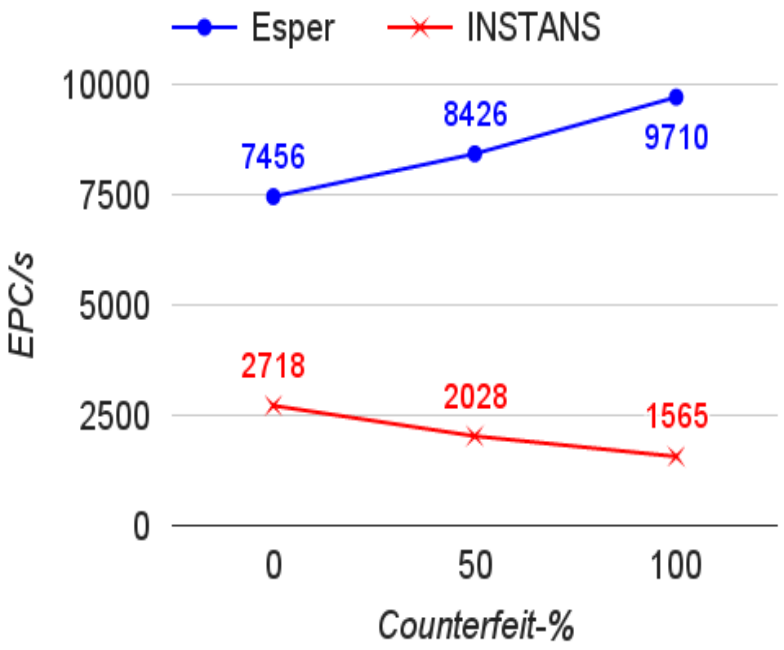
EPC-codes per second

Nr. Of EPC	100	1,000	10,000	100,000	1,000,000
Esper	734	2,867	8,146	21,790	39,243
INSTANS	964	1,952	2,566	2,686	2,514

E2: Performance Impact of Counterfeit Detection

Parameter	Values
Δt_{cp}	3.5 s
t_{winc}	5 s
Counterfeit EPC	0%, 50%, 100%

- **Everything else stable, alter the share of counterfeits**
- **INSTANS slowed down by more than 40% from 0 to 100% error rate due to an increased amount of output to process**
- **Esper speed increased over 30%**
  - Difference in algorithms: as there are fewer entries going to the win_m matching window in the higher counterfeit cases (nothing from commissioning matches packing in the 100% case), there are also fewer comparisons with win_m contents.

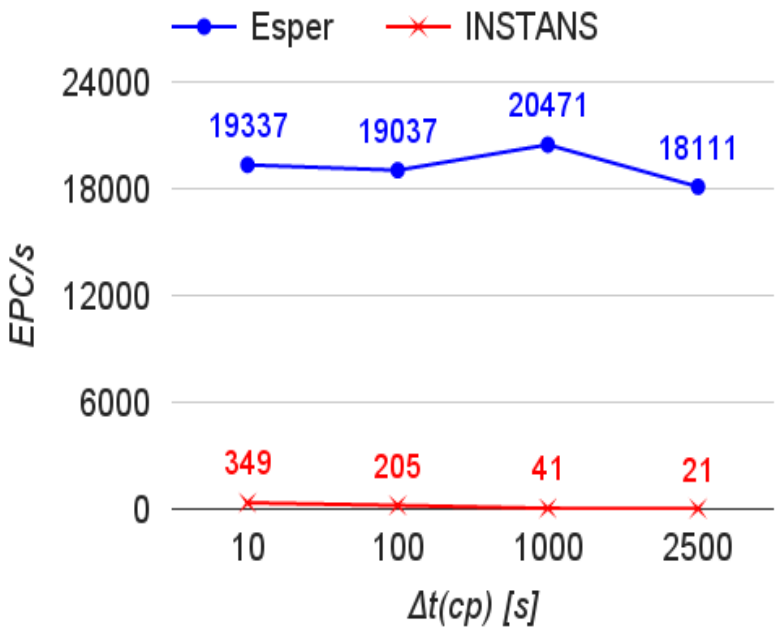


Aalto University
School of Science

E3: Intermediate Buffering Impact

- **All test batches equally long (100k EPC)**
- **Time between commissioning and packing varies, long commissioning-window**
- **Esper**
  - No clear performance impact
  - 20% memory consumption increase
- **INSTANS**
  - Addition of new EPC codes to the graph slows down when the number of buffered codes increases due to the increased number of bindings to be verified for each addition (130x slowdown)
  - 121% memory consumption increase

Parameter	Values
Δt_{cp}	10; 100; 1,000; 2,500 s
t_{winc}	2,600 s
Counterfeit EPC	8%



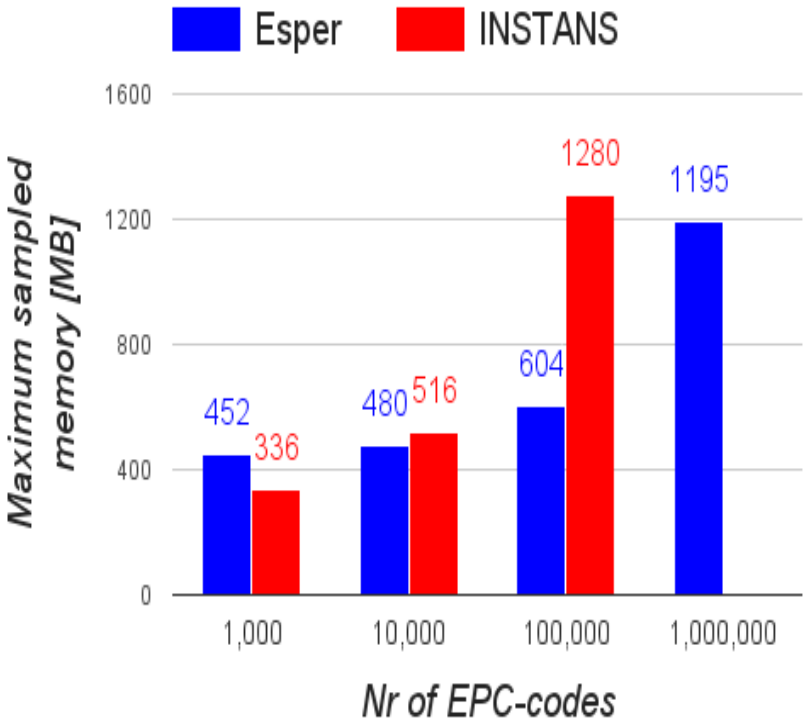
E4: Buffer Everything

- **Variable test batch length**
- **Everything buffered between commissioning and packing**
- **Esper**
  - Performance only slows down with 1M EPCs
- **INSTANS**
  - 100k EPCs at 20 EPC/s, 1M EPC did not complete in six days (vs. Esper 91 secs)
    - *Preliminary result with latest optimisations (after paper submission) 86x improvements achieved for 100k, 1736 EPC/s and for 1M EPC 1117 EPC/s*

Parameter	Values
Δt_{cp}	25; 250; 2,500; 25,000 s
t_{winc}	30,000 s

Nr. of EPC	1,000	10,000	100,000	1,000,000
Esper	3,008	7,941	20,360	11,028
INSTANS	1,083	228	20	N/A

EPC-codes per second



Aalto University
School of Science

E5: Complete Processing Chain

- **As experiment 1, but include shipping**
- **Esper**
  - 8.5 times slower than itself in exp1 at 1M EPCs
  - 1,600 times requirement **R5** (1M EPC)
- **INSTANS**
  - 1M EPCs did not complete due to a token collision error
  - 950 times requirement **R5** (100k EPC)

Parameter	Values
ps [nr]	20
Δt_c	0.5 s
Δt_{cp}	3.5 s
Δt_p	2.5 s
Δt_{ps}	110 s
Δt_s	50 s

Nr. Of EPC	100	1,000	10,000	100,000	1,000,000
Esper	724	3,012	3,161	4,526	4,639
INSTANS	888	1,811	2,573	2,714	N/A

EPC-codes per second

Results vs. Requirements

Requirement	Esper + XEvePro	INSTANS
Predictability and consistency (R1)	Ok[1)]	Ok[1)]
Stream time synchronisation (R2)	Ok[1)]	Ok[1)]
Stability (R3)	Ok	Slow memory residue buildup observed. Exp5 1M did not complete due to a runtime error.
Memory efficiency (R4)[2)]	166 EPC/MiB at 100k EPC, 837 EPC/MiB at 1M EPC (Exp4).	78.1 EPC/MiB at 100k EPC, 1M EPC did not complete (Exp4).
Performance (R5)	Stable performance, target exceeded in all tests. Exp5 1M EPC exceeds target 1,600 times.	Performance impacted by output (Exp2) and buffering (Exp3, Exp4). Exp5 100k EPC exceeds target 950 times.

[1)] identical results verified
[2)] sampled with *top*

Qualitative Comparison (1/2)

	Esper + XEvePro	INSTANS
Configuration parameters	EPL and command line	RDF (Turtle) and command line
Programming	EPL and Scala	SPARQL
Synchronisation to timestamps in events	EPL and Scala providing an external sync to the Esper system timer	SPARQL
Buffering of EPC codes	Built-in window mechanism	In-memory graph storage
Output formatting	Scala listeners attached to EPL queries	SPARQL SELECT queries
Use of globally accessible ontologies	XML namespaces but not RDF/OWL ontologies (ontology names preserved in the XML format)	Yes
Semantic web tool compatibility	No	Through input data (TriG), output (TriG, turtle, ntriples, nquads available) and federated query support
Reasoning and entailments	No	INSTANS supports entailment rules, RDF is compatible with other reasoning tools

Qualitative Comparison (2/2)

- **Considering the next step in the logistics chain**
  - The manufacturer ships the pallets and a shipping manifest (a “pedigree”) to a warehouse
- **Semantic Web / INSTANS**
  - The pedigree can be a graph or dataset and exposed to the warehouse as an IRI
  - The warehouse can send federated queries to compare the contents of a shipment against the pedigree.
  - INSTANS could
    1. *construct a graph or dataset of the successfully shipped items in the sending site*
    2. *perform the federated queries against the observed item codes in the receiving site.*
- **Non-semantic web / Esper**
  - SQL database connectivity supported
  - A similar outcome could be achieved by saving the pedigree to a database and exposing the database to the warehouse
  - Extra coding effort required

Conclusions

- **The example industrial logistics monitoring task was successfully implemented on both RDF/SPARQL and XML/EPL frameworks**
  - Identical results confirmed
- **Esper demonstrated clearly better performance and higher maturity of the platform**
  - Platform of choice for best performance
- **INSTANS also met or exceeded real-world derived requirements, when excessive buffering was not required**
  - Work in progress, but with benefits in integration to the semantic web framework