

# Pushing dynamic and ubiquitous event-based interaction in the Internet of services: a middleware for event clouds

Laurent Pellegrino

Thesis supervised by Françoise BAUDE  
and co-supervised by Fabrice HUET

<i>Reviewers</i>	Ioana MANOLESCU Etienne RIVIÈRE	Inria Saclay - Île-de-France Université de Neuchâtel
<i>Examiners</i>	Johan MONTAGNAT Ester PACITTI Vivien QUÉMA	CNRS Université de Montpellier 2 Grenoble INP



# Outline

- ① Introduction
- ② Distributed RDF storage
- ③ Distributed RDF pub/sub
- ④ Distributed RDF load balancing
- ⑤ Implementation
- ⑥ Conclusion

# Outline

- 1 Introduction
  - Motivation
  - Context
  - Problem definition
- 2 Distributed RDF storage
- 3 Distributed RDF pub/sub
- 4 Distributed RDF load balancing
- 5 Implementation
- 6 Conclusion

# Motivation

- Exponential growing of information produced
  - ▶ Internet of Things
    - Internet scale collection of connected data sources
- Data mining
  - ▶ Prerequisite
    - Integrate data from heterogeneous sources
  - ▶ Discover interesting patterns
    - Filter information of interest and correlate them with others
- Semantic Web
  - ▶ Provides a full technology stack (RDF, SPARQL, RDFS, OWL, etc.) for managing structured data
  - ▶ Mainly in a synchronous and centralized environment



# Context: PLAY and SocEDA projects



# Context: PLAY and SocEDA projects

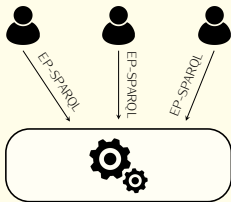


## Platform users

Subscribe to detect interesting patterns and react consequently



# Context: PLAY and SocEDA projects

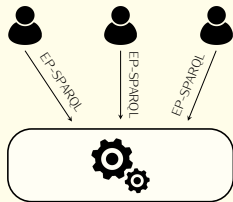


## Platform users

Subscribe to detect interesting patterns and react consequently



# Context: PLAY and SocEDA projects



## Platform users

Subscribe to detect interesting patterns and react consequently

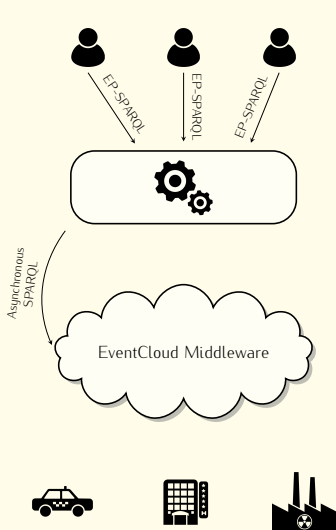
## CEP

Complex Event Processing engine (Etalis/Esper)





# Context: PLAY and SocEDA projects



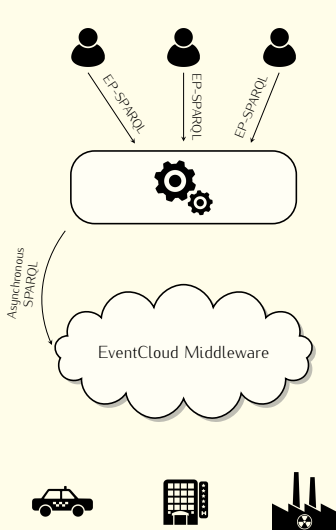
## Platform users

Subscribe to detect interesting patterns and react consequently

## CEP

Complex Event Processing engine (Etalis/Esper)

# Context: PLAY and SocEDA projects



## Platform users

Subscribe to detect interesting patterns and react consequently

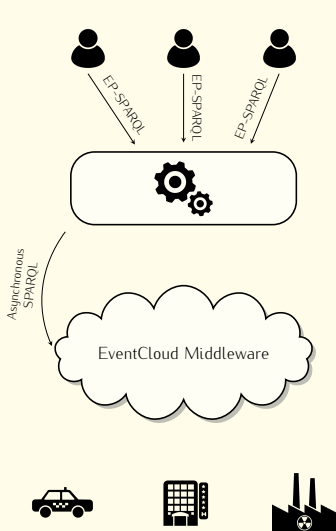
## CEP

Complex Event Processing engine (Etalis/Esper)

## EventCloud

Scalable storage and retrieval of RDF-based data

# Context: PLAY and SocEDA projects



## Platform users

Subscribe to detect interesting patterns and react consequently

## CEP

Complex Event Processing engine (Etalis/Esper)

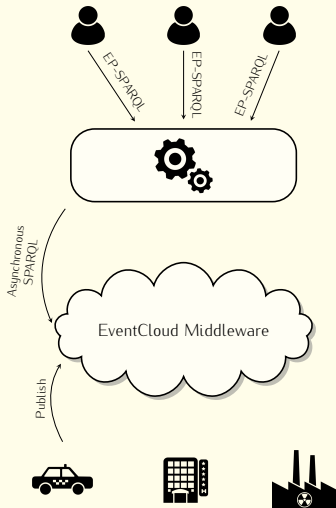
## EventCloud

Scalable storage and retrieval of RDF-based data

## Event sources

Send data synchronously or asynchronously

# Context: PLAY and SocEDA projects



## Platform users

Subscribe to detect interesting patterns and react consequently

## CEP

Complex Event Processing engine (Etalis/Esper)

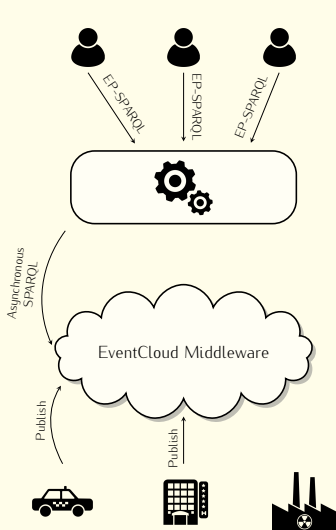
## EventCloud

Scalable storage and retrieval of RDF-based data

## Event sources

Send data synchronously or asynchronously

# Context: PLAY and SocEDA projects



## Platform users

Subscribe to detect interesting patterns and react consequentially

## CEP

Complex Event Processing engine (Etalis/Esper)

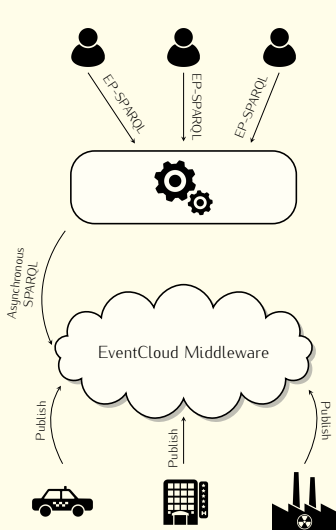
## EventCloud

Scalable storage and retrieval of RDF-based data

## Event sources

Send data synchronously or asynchronously

# Context: PLAY and SocEDA projects



## Platform users

Subscribe to detect interesting patterns and react consequently

## CEP

Complex Event Processing engine (Etlis/Esper)

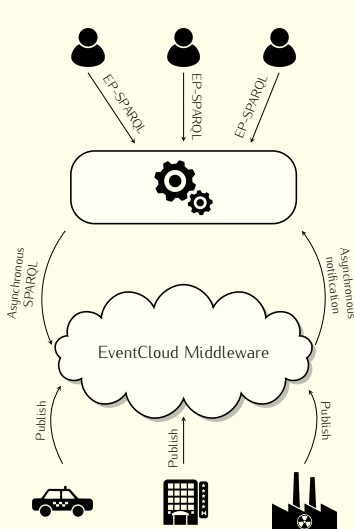
## EventCloud

Scalable storage and retrieval of RDF-based data

## Event sources

Send data synchronously or asynchronously

# Context: PLAY and SocEDA projects



## Platform users

Subscribe to detect interesting patterns and react consequently

## CEP

Complex Event Processing engine (Etalis/Esper)

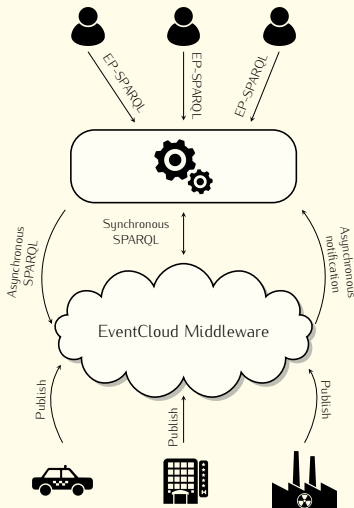
## EventCloud

Scalable storage and retrieval of RDF-based data

## Event sources

Send data synchronously or asynchronously

# Context: PLAY and SocEDA projects



## Platform users

Subscribe to detect interesting patterns and react consequently

## CEP

Complex Event Processing engine (Etalis/Esper)

## EventCloud

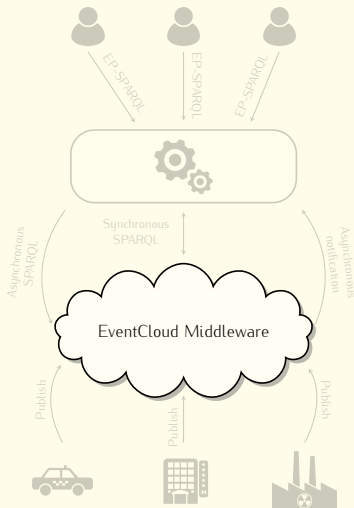
Scalable storage and retrieval of RDF-based data

## Event sources

Send data synchronously or asynchronously



# Context: PLAY and SocEDA projects



## Platform users

Subscribe to detect interesting patterns and react consequently

## CEP

Complex Event Processing engine (Etalis/Esper)

## EventCloud

Scalable storage and retrieval of RDF-based data

## Event sources

Send data synchronously or asynchronously

# Problem definition

- ① *How can we efficiently store and retrieve Semantic Web data in a distributed environment?*
  - ② *How can we pragmatically filter and disseminate Semantic Web events to users with individual preferences?*
- Challenges
    - ▶ Data indexing/retrieval
    - ▶ Near real-time filtering
    - ▶ Load balancing
    - ▶ Scalability

Middleware devoted to storing, retrieving synchronously but also disseminating selectively and asynchronously RDF data

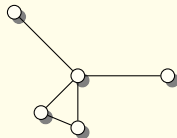
# Outline

- 1 Introduction
- 2 Distributed RDF storage
  - Background
  - P2P Infrastructure for RDF
  - Evaluation
- 3 Distributed RDF pub/sub
- 4 Distributed RDF load balancing
- 5 Implementation
- 6 Conclusion

# Peer-to-Peer (P2P) networks

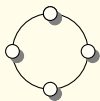
- Unstructured P2P networks

- ▶ links between peers are chosen randomly
- ▶ bad performance for search (flooding)



- Structured P2P networks

- ▶ various topologies (ring, hypertorus, tree)
- ▶ communication overhead for *join/leave*
- ▶ more efficient for lookup



Chord



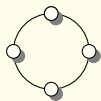
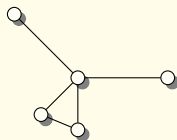
CAN



P-grid

# Peer-to-Peer (P2P) networks

- Unstructured P2P networks
  - ▶ links between peers are chosen randomly
  - ▶ bad performance for search (flooding)
- Structured P2P networks
  - ▶ various topologies (ring, hypertorus, tree)
  - ▶ communication overhead for *join/leave*
  - ▶ more efficient for lookup



Chord



CAN

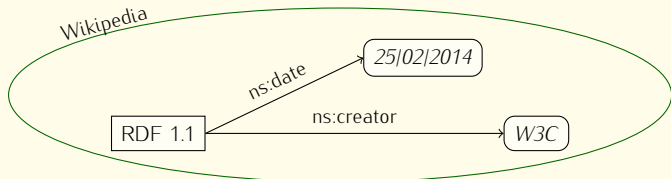


P-grid



# RDF/SPARQL

- Resource Description Framework (RDF)
  - ▶ Model information based on tuples (3-tuple or 4-tuple)
  - ▶ Quadruples depicted as  $q = (g, s, p, o) \mid g, s, p, o \in RDFTerm$



- SPARQL (SPARQL Protocol and Query Language)
  - ▶ Declarative query language for RDF data
  - ▶ Highly expressive query language
    - Several query types (SELECT, DESCRIBE, CONSTRUCT, ASK)
    - Multiple graph patterns (Basic, Union, Constraints)
    - Many solution modifiers (DISTINCT, LIMIT, OFFSET)

## Elementary query types

- **Atomic queries** are quadruple patterns where the graph, the subject, the predicate and/or the object may be a variable
  - ▶  $(?g, ?s, ns:creator, W3C)$
- **Conjunctive queries** are expressed as a list of quadruple patterns (sub-queries).
  - ▶  $(g_1, ?s, p_1, ?o_1) \wedge (g_1, ?s, p_2, ?o_2)$
- **Range queries** involve queries for single or multiple variables whose values fall into a range defined by the query
  - ▶  $(?g, s, p, ?o) \text{ FILTER } v_1 \leq ?o < v_2$

## Related works

### P2P

RDFPeers

RDFCube

LIAROU *et al.*

### NoSQL

CumulusRDF

H<sub>2</sub>RDF

AllegroGraph

BigData

Limited SPARQL support

Monolithic software architecture

Hashing for indexing and storing RDF data



## Related works

### P2P

RDFPeers

RDFCube

LIAROU *et al.*FILALI<sup>1</sup>BONGIOVANNI<sup>2</sup>

### NoSQL

CumulusRDF

H<sub>2</sub>RDF

AllegroGraph

BigData

Limited SPARQL support

Monolithic software architecture

Hashing for indexing and storing RDF data

<sup>1</sup> FILALI, "Improving resource discovery in P2P systems".

<sup>2</sup> BONGIOVANNI, "Design, formalization and implementation of overlay networks; application to RDF data storage".

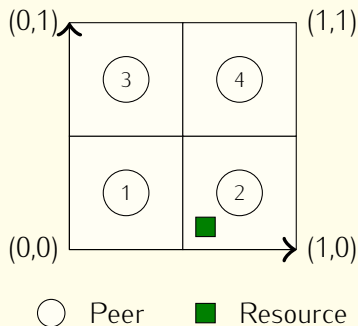
# Hashing vs Lexicographic based indexing

- Hash based indexing
  - ▶ Uniform hash functions
    - Destroy data locality
  - ▶ Locality preserving hash functions
    - Keep relative distance between input values and output values but cause load imbalances
- Lexicographic order based indexing
  - ▶ Preserves data locality
    - Native support for range queries
  - ▶ Distribution on peers depends of data values

Lexicographic based indexing exhibits similar properties to hash based indexing using locality preserving hash functions without the extra complexity of hashing

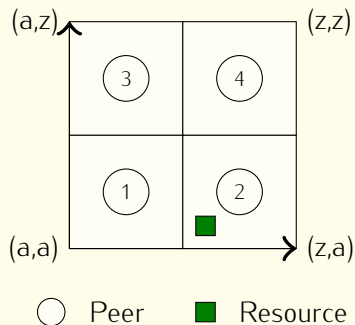
# Our P2P infrastructure

- Built atop Content Addressable Network (CAN)
  - ▶ Resources are distributed in a 4-dimensional space  $\mathbb{D}$ 
    - $\mathbb{D}$  is divided into zones
    - Each zone is managed by a peer
    - Each zone may contain multiple resources
    - A resource is a quadruple  $q = (g, s, p, o)$  indexed as a point in  $\mathbb{D}$



# Our P2P infrastructure

- Built atop Content Addressable Network (CAN)
  - ▶ Resources are distributed in a 4-dimensional space  $\mathbb{D}$ 
    - $\mathbb{D}$  is divided into zones
    - Each zone is managed by a peer
    - Each zone may contain multiple resources
    - A resource is a quadruple  $\mathbf{q} = (\mathbf{g}, \mathbf{s}, \mathbf{p}, \mathbf{o})$  indexed as a point in  $\mathbb{D}$



No hashing but lexicographic order

# Routing algorithms

- Unicast routing
  - ▶ Used to route a message to one specific peer in the CAN space
    - Based on a key that depicts a point to reach
  - ▶ Scheme based on greedy forwarding per dimension
    - Does not use euclidean distance since radix 1114112 is used
- Multicast routing
  - ▶ Used to route messages to a subset of peers on the CAN overlay
    - Based on optimal broadcast proposed by Francesco BONGIOVANNI<sup>3</sup>

---

<sup>3</sup> BONGIOVANNI and HENRIO, "A Mechanized Model for CAN Protocols".

# Routing algorithms

- Unicast routing
  - ▶ Used to route a message to one specific peer in the CAN space
    - Based on a key that depicts a point to reach
  - ▶ Scheme based on greedy forwarding per dimension
    - Does not use euclidean distance since radix 1114112 is used
- Multicast routing
  - ▶ Used to route messages to a subset of peers on the CAN overlay
    - Based on optimal broadcast proposed by Francesco BONGIOVANNI<sup>3</sup>

Unicast routing is used to index a quadruple

---

<sup>3</sup> BONGIOVANNI and HENRIO, "A Mechanized Model for CAN Protocols".

# Routing algorithms

- Unicast routing
  - ▶ Used to route a message to one specific peer in the CAN space
    - Based on a key that depicts a point to reach
  - ▶ Scheme based on greedy forwarding per dimension
    - Does not use euclidean distance since radix 1114112 is used
- Multicast routing
  - ▶ Used to route messages to a subset of peers on the CAN overlay
    - Based on optimal broadcast proposed by Francesco BONGIOVANNI<sup>3</sup>

Unicast routing is used to index a quadruple  
Multicast routing is the substrat for retrieving information

---

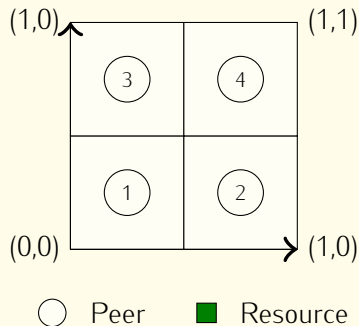
<sup>3</sup> BONGIOVANNI and HENRIO, "A Mechanized Model for CAN Protocols".

# RDF data indexing

```

Data: Quadruple  $q$ 
begin
  if  $q \notin \text{peer's zone}$  then
    Forward the request to the closest
    neighbor
  else
    Store  $q$ ;
    Send back response;
  end
end
end

```



No hashing but lexicographic order

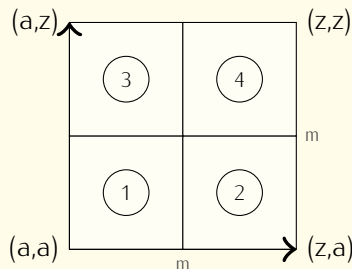


# RDF data indexing

```

Data: Quadruple  $q$ 
begin
  if  $q \notin \text{peer's zone}$  then
    Forward the request to the closest
    neighbor
  else
    Store  $q$ ;
    Send back response;
  end
end

```



○ Peer    ■ Resource

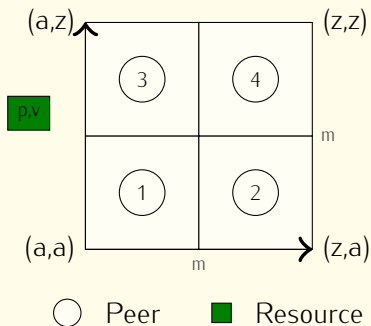
No hashing but lexicographic order

# RDF data indexing

```

Data: Quadruple  $q$ 
begin
  if  $q \notin \text{peer's zone}$  then
    Forward the request to the closest
    neighbor
  else
    Store  $q$ ;
    Send back response;
  end
end

```



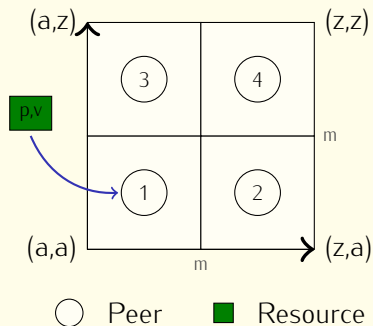
No hashing but lexicographic order

## RDF data indexing

```

Data: Quadruple  $q$ 
begin
  if  $q \notin \text{peer's zone}$  then
    Forward the request to the closest
    neighbor
  else
    Store  $q$ ;
    Send back response;
  end
end
end

```



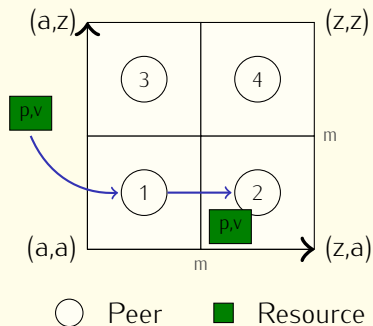
No hashing but lexicographic order

## RDF data indexing

```

Data: Quadruple  $q$ 
begin
  if  $q \notin \text{peer's zone}$  then
    Forward the request to the closest
    neighbor
  else
    Store  $q$ ;
    Send back response;
  end
end
end

```



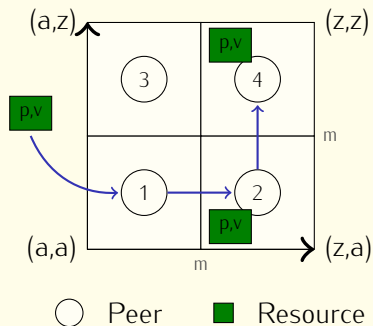
No hashing but lexicographic order

# RDF data indexing

```

Data: Quadruple  $q$ 
begin
  if  $q \notin \text{peer's zone}$  then
    Forward the request to the closest
    neighbor
  else
    Store  $q$ ;
    Send back response;
  end
end
end

```



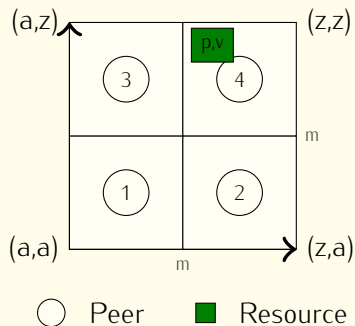
No hashing but lexicographic order

## RDF data indexing

```

Data: Quadruple  $q$ 
begin
  if  $q \notin \text{peer's zone}$  then
    Forward the request to the closest
    neighbor
  else
    Store  $q$ ;
    Send back response;
  end
end
end

```



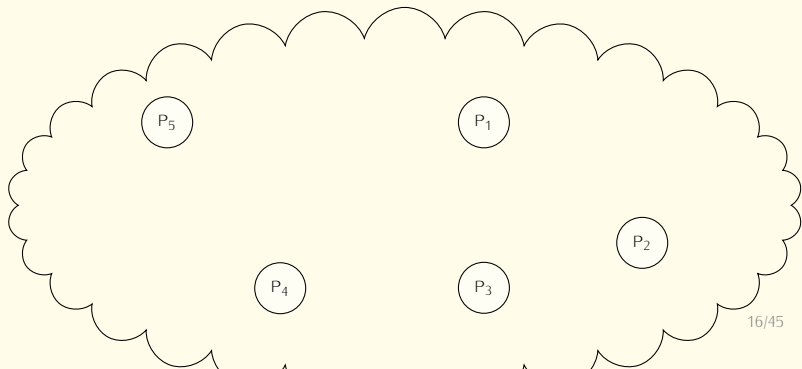
No hashing but lexicographic order

# RDF data retrieval

SPARQL Query  $Q$

```
SELECT ?g ?user WHERE {  
  GRAPH ?g {  
    ?user foaf:name "John Doe" .  
    ?user foaf:age ?age  
  } FILTER (?age ≥ 18 && ?age ≤ 25)  
}
```

SPARQL Decomposer



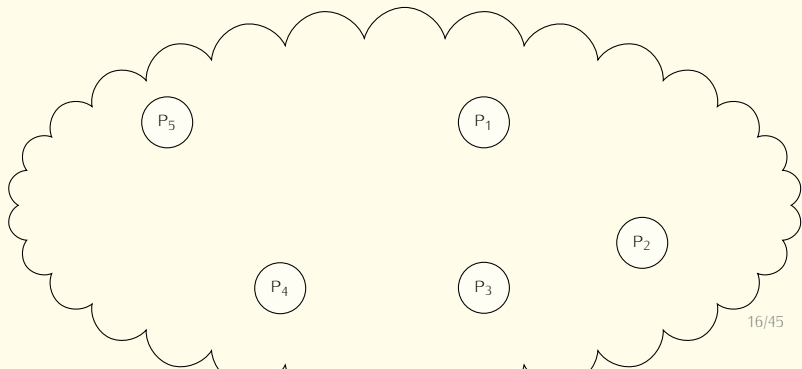
# RDF data retrieval

SPARQL Query  $Q$



```
SELECT ?g ?user WHERE {  
  GRAPH ?g {  
    ?user foaf:name "John Doe" .  
    ?user foaf:age ?age  
  } FILTER (?age ≥ 18 && ?age ≤ 25)  
}
```

SPARQL Decomposer





# RDF data retrieval

SPARQL Query  $Q$



```
SELECT ?g ?user WHERE {  
  GRAPH ?g {  
    ?user foaf:name "John Doe" .  
    ?user foaf:age ?age  
  } FILTER (?age ≥ 18 && ?age ≤ 25)  
}
```

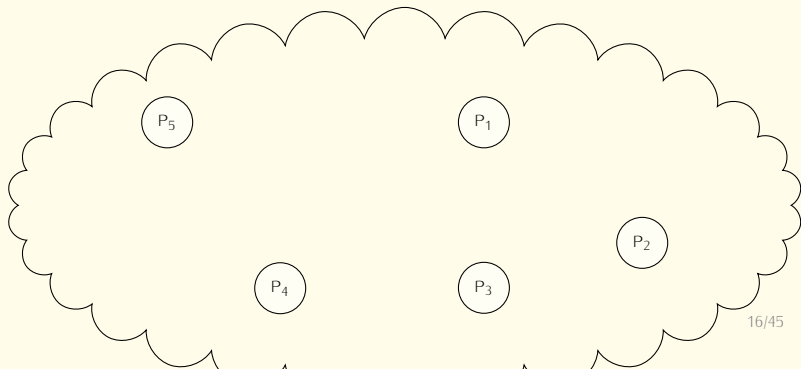
SPARQL Decomposer

$q_1$

```
?user foaf:name "John Doe"
```

$q_2$

```
?user foaf:age ?age  
FILTER ?age ≥ 18 && ?age ≤ 25
```



# RDF data retrieval

SPARQL Query  $Q$



```
SELECT ?g ?user WHERE {  
  GRAPH ?g {  
    ?user foaf:name "John Doe" .  
    ?user foaf:age ?age  
  } FILTER (?age ≥ 18 && ?age ≤ 25)  
}
```

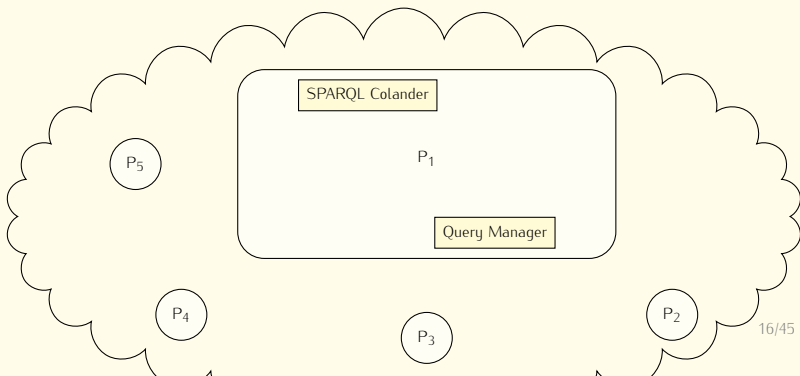
SPARQL Decomposer

$q_1$

$q_2$

```
?user foaf:name "John Doe"
```

```
?user foaf:age ?age  
FILTER ?age ≥ 18 && ?age ≤ 25
```



# RDF data retrieval

SPARQL Query  $Q$



```
SELECT ?g ?user WHERE {  
  GRAPH ?g {  
    ?user foaf:name "John Doe" .  
    ?user foaf:age ?age  
  } FILTER (?age ≥ 18 && ?age ≤ 25)  
}
```

SPARQL Decomposer

$q_1$

$q_2$

?user foaf:name "John Doe"

?user foaf:age ?age  
FILTER ?age ≥ 18 && ?age ≤ 25

SPARQL Colander

$P_1$

Query Manager

$P_5$

$P_4$

$P_3$

$P_2$

# RDF data retrieval

SPARQL Query  $Q$



```
SELECT ?g ?user WHERE {
  GRAPH ?g {
    ?user foaf:name "John Doe" .
    ?user foaf:age ?age
  } FILTER (?age ≥ 18 && ?age ≤ 25)
}
```

SPARQL Decomposer

$q_1$

$q_2$

?user foaf:name "John Doe"

?user foaf:age ?age  
FILTER ?age ≥ 18 && ?age ≤ 25

SPARQL Colander

$P_1$

Query Manager

$q_1$

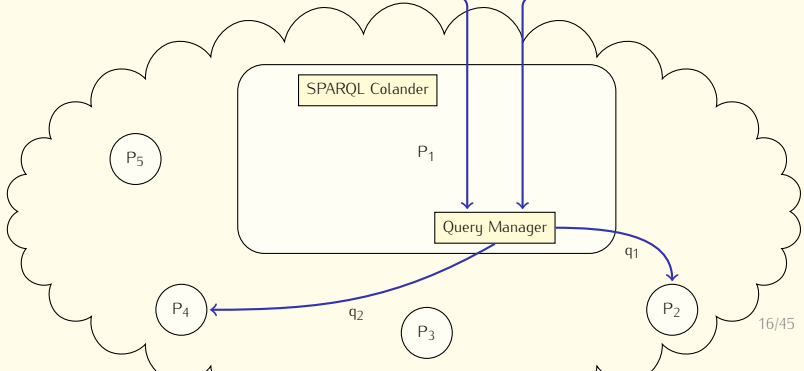
$q_2$

$P_5$

$P_4$

$P_3$

$P_2$



# RDF data retrieval

SPARQL Query  $Q$



```
SELECT ?g ?user WHERE {
  GRAPH ?g {
    ?user foaf:name "John Doe" .
    ?user foaf:age ?age
  } FILTER (?age ≥ 18 && ?age ≤ 25)
}
```

SPARQL Decomposer

$q_1$

$q_2$

?user foaf:name "John Doe"

?user foaf:age ?age  
FILTER ?age ≥ 18 && ?age ≤ 25

SPARQL Colander

$P_1$

Query Manager

$P_5$

$q_2$

$P_4$

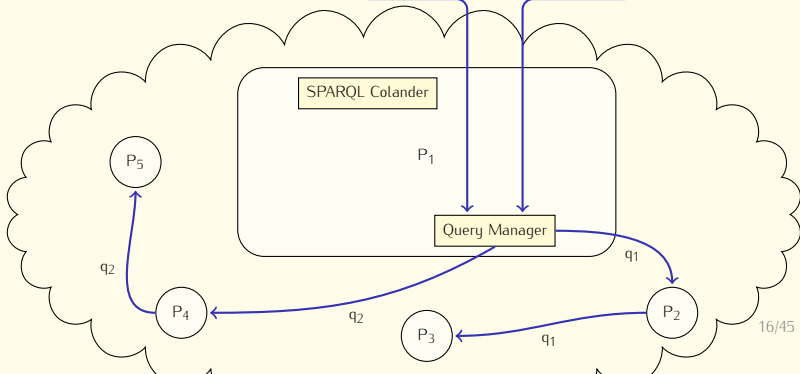
$q_2$

$P_3$

$q_1$

$P_2$

$q_1$



# RDF data retrieval

SPARQL Query  $Q$



```
SELECT ?g ?user WHERE {
  GRAPH ?g {
    ?user foaf:name "John Doe" .
    ?user foaf:age ?age
  } FILTER (?age ≥ 18 && ?age ≤ 25)
}
```

SPARQL Decomposer

$q_1$

$q_2$

?user foaf:name "John Doe"

?user foaf:age ?age  
FILTER ?age ≥ 18 && ?age ≤ 25

SPARQL Colander

$P_1$

Query Manager

$P_5$

$q_2$

$P_4$

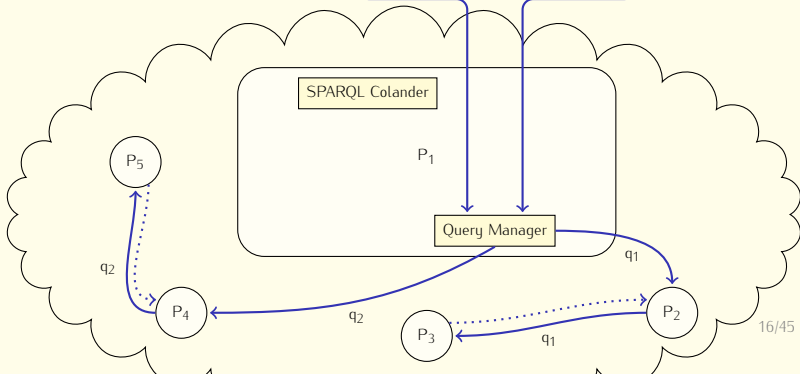
$q_2$

$P_3$

$q_1$

$P_2$

$q_1$



# RDF data retrieval

SPARQL Query  $Q$

```
SELECT ?g ?user WHERE {
  GRAPH ?g {
    ?user foaf:name "John Doe" .
    ?user foaf:age ?age
  } FILTER (?age ≥ 18 && ?age ≤ 25)
}
```

SPARQL Decomposer

$q_1$

$q_2$

?user foaf:name "John Doe"

?user foaf:age ?age  
FILTER ?age ≥ 18 && ?age ≤ 25

SPARQL Colander

$P_1$

Query Manager

$P_5$

$q_2$

$P_4$

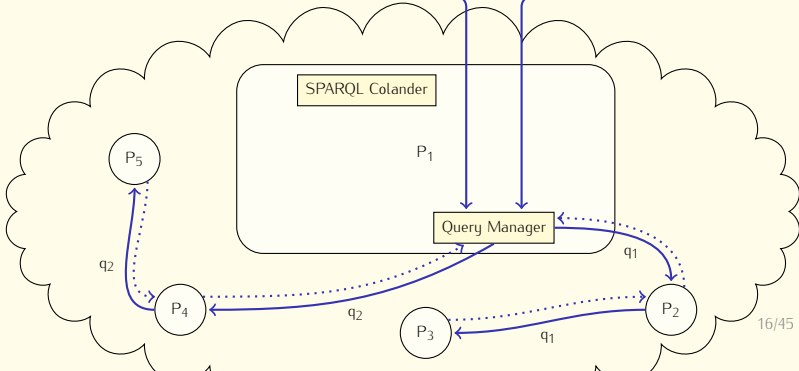
$q_2$

$P_3$

$q_1$

$P_2$

$q_1$



# RDF data retrieval

SPARQL Query  $Q$

```
SELECT ?g ?user WHERE {
  GRAPH ?g {
    ?user foaf:name "John Doe" .
    ?user foaf:age ?age
  } FILTER (?age ≥ 18 && ?age ≤ 25)
}
```

SPARQL Decomposer

$q_1$

$q_2$

?user foaf:name "John Doe"

?user foaf:age ?age  
FILTER ?age ≥ 18 && ?age ≤ 25

SPARQL Colander

$P_1$

Query Manager

$P_5$

$q_2$

$P_4$

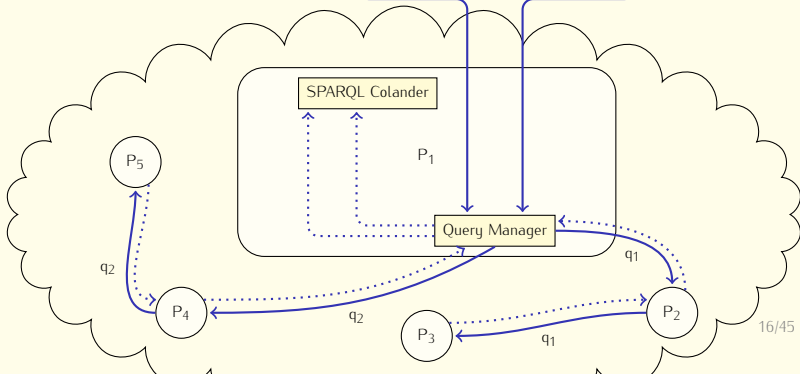
$q_2$

$P_3$

$q_1$

$P_2$

$q_1$





# RDF data retrieval

SPARQL Query  $Q$

```
SELECT ?g ?user WHERE {
  GRAPH ?g {
    ?user foaf:name "John Doe" .
    ?user foaf:age ?age
  } FILTER (?age ≥ 18 && ?age ≤ 25)
}
```

SPARQL Decomposer

$q_1$

$q_2$

?user foaf:name "John Doe"

?user foaf:age ?age  
FILTER ?age ≥ 18 && ?age ≤ 25

SPARQL Colander

$P_1$

Query Manager

$P_5$

$q_2$

$P_4$

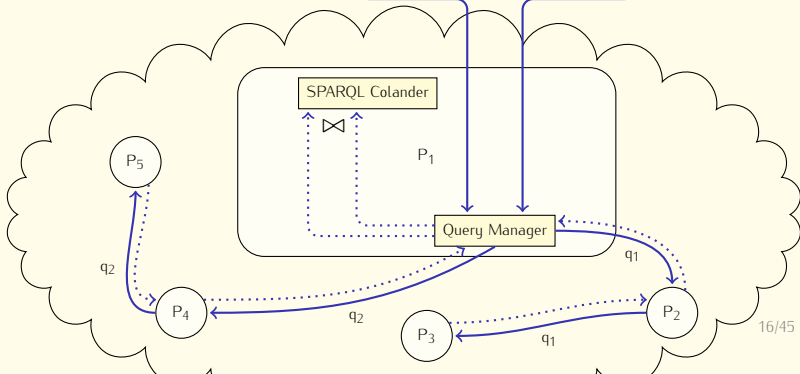
$q_2$

$P_3$

$q_1$

$P_2$

$q_1$



# RDF data retrieval

SPARQL Query  $Q$

```
SELECT ?g ?user WHERE {
  GRAPH ?g {
    ?user foaf:name "John Doe" .
    ?user foaf:age ?age
  } FILTER (?age ≥ 18 && ?age ≤ 25)
}
```

SPARQL Decomposer

$q_1$

$q_2$

?user foaf:name "John Doe"

?user foaf:age ?age  
FILTER ?age ≥ 18 && ?age ≤ 25

SPARQL Colander

$P_1$

Query Manager

$P_5$

$q_2$

$P_4$

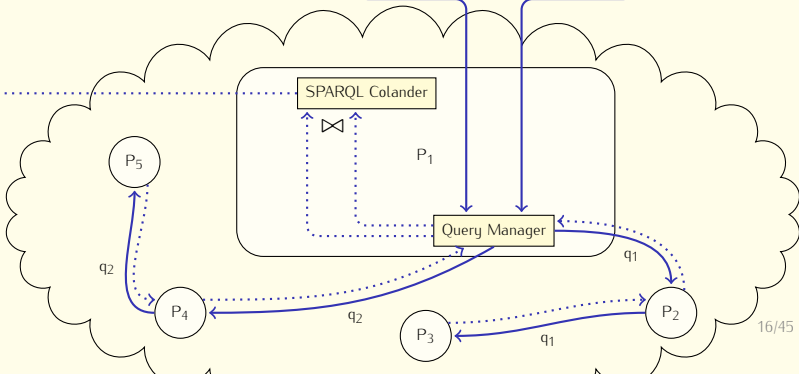
$q_2$

$P_3$

$q_1$

$P_2$

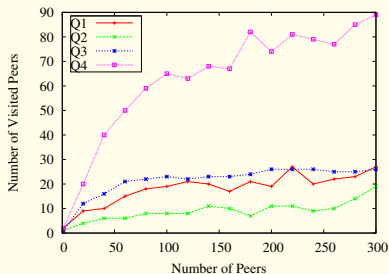
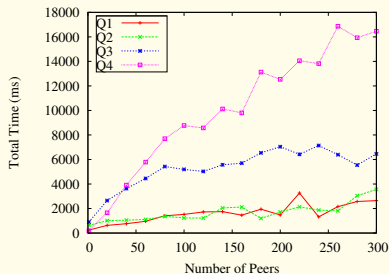
$q_1$



# Experimental setup

- Up to 300 peers with 75 machines on the Grid'5000 testbed
  - ▶ Xeon L5420@2,5GHz
  - ▶ 16 GB RAM
  - ▶ 7200 RPM HDD
- Dataset
  - ▶ BSBM Berlin SPARQL benchmark
    - Built around an e-commerce usecase
- Queries
  - Q1 Conjunctive query with same subject
  - Q2 Atomic query with fixed predicate and object
  - Q3 Atomic query with fixed predicate and free subject and object
  - Q4 Conjunctive query with two variables in each subquery

# Experiments



Performance results strongly depend on the type of queries

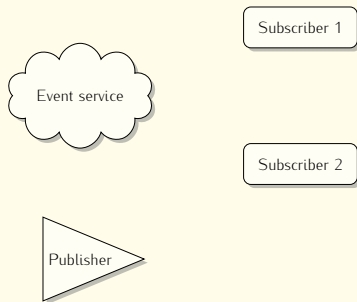
# Outline

- 1 Introduction
- 2 Distributed RDF storage
- 3 Distributed RDF pub/sub
  - Background
  - Publish/Subscribe Infrastructure for RDF
  - Evaluation
- 4 Distributed RDF load balancing
- 5 Implementation
- 6 Conclusion

# Context

- Event Driven Architecture

- ▶ Publish/Subscribe  
*asynchronous*  
communication style
  - Subscribers
  - Publishers
  - Event notification service
- ▶ Entities are decoupled in  
*space, time* and  
*synchronization*



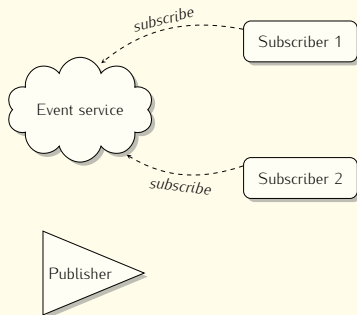
- Publish/subscribe layer for RDF data

- ▶ Reuses some technologies from the Semantic Web stack
- ▶ Filters and disseminates RDF events to interested parties
- ▶ Stores published events for future reuse

# Context

- Event Driven Architecture

- ▶ Publish/Subscribe  
*asynchronous*  
communication style
  - Subscribers
  - Publishers
  - Event notification service
- ▶ Entities are decoupled in  
*space, time* and  
*synchronization*



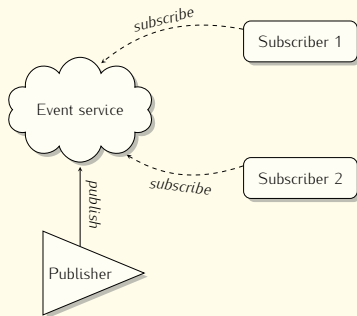
- Publish/subscribe layer for RDF data

- ▶ Reuses some technologies from the Semantic Web stack
- ▶ Filters and disseminates RDF events to interested parties
- ▶ Stores published events for future reuse

# Context

- Event Driven Architecture

- ▶ Publish/Subscribe *asynchronous* communication style
  - Subscribers
  - Publishers
  - Event notification service
- ▶ Entities are decoupled in *space, time* and *synchronization*



- Publish/subscribe layer for RDF data

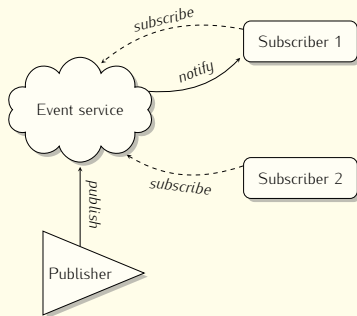
- ▶ Reuses some technologies from the Semantic Web stack
- ▶ Filters and disseminates RDF events to interested parties
- ▶ Stores published events for future reuse



# Context

- Event Driven Architecture

- ▶ Publish/Subscribe  
*asynchronous*  
communication style
  - Subscribers
  - Publishers
  - Event notification service
- ▶ Entities are decoupled in  
*space, time* and  
*synchronization*



- Publish/subscribe layer for RDF data

- ▶ Reuses some technologies from the Semantic Web stack
- ▶ Filters and disseminates RDF events to interested parties
- ▶ Stores published events for future reuse

## Related works: pub/sub systems

### Topic-based

Tibco<sup>4</sup>

Pubsubhubbub<sup>5</sup>

### Content-based

Siena<sup>6</sup>

Hermes<sup>7</sup>

BlueDove<sup>8</sup>

Do not focus on the specific characteristics of RDF  
*Attribute/value pairs vs Tuples*  
*Constraints defined at startup (event size, type, domain)*

---

<sup>4</sup> TIBCO, "TIB/Rendezvous White Paper".

<sup>5</sup> FITZPATRICK, SLATKIN, and ATKINS, *PubSubHubbub protocol*.

<sup>6</sup> CARZANIGA, ROSENBLUM, and WOLF, "Design and evaluation of a wide-area event notification service".

<sup>7</sup> PIETZUCH and BACON, "Hermes: A distributed event-based middleware architecture".

<sup>8</sup> LI, YE, KIM, CHEN, and LEI, "A scalable and elastic publish/subscribe service".

## Related works: RDF pub/sub systems (1)

- RDFPeers<sup>9</sup>
  - ▶ Built atop MAAN
  - ▶ Rendezvous nodes are created for publications and subscriptions
  - ▶ Each publication is indexed three times
  - ▶ Not possible to subscribe for all information
  - ▶ Some join constraints are not supported
  - ▶ Popular RDF terms are ignored (e.g., *rdf:type*)
- Ranger *et al.*<sup>10</sup>
  - ▶ Information sharing platform using multicast trees
  - ▶ The peers participating to the propagation are responsible for removing duplicate results within the limit of their buffer

---

<sup>9</sup> CAI, FRANK, YAN, and MACGREGOR, "A subscribable peer-to-peer RDF repository for distributed metadata management".

<sup>10</sup> RANGER and CLOUTIER, "Scalable peer-to-peer RDF query algorithm".

## Related works: RDF pub/sub systems (2)

- CSBV<sup>11</sup>
  - ▶ Distributed Hash Table (DHT) agnostic
  - ▶ Each publication is indexed seven times
- Shvartzshnaider *et al.*<sup>12</sup>
  - ▶ Support arbitrary tuples
  - ▶ Combine AI and P2P research by applying Rete algorithms
  - ▶ Ad-hoc scripting language for subscribing
  - ▶ No explanation about how duplicates are avoided when in-memory buffers overflow
  - ▶ No experiment is provided

---

<sup>11</sup> LIAROU, IDREOS, and KOUBARAKIS, "Continuous RDF query processing over DHTs".

<sup>12</sup> SHVARTZSHNAIDER, OTT, and LEVY, "Publish/subscribe on top of DHT using RETE algorithm".

# Proposed data and subscription model

- Data Model

- ▶ RDF quadruple  $q = (g, s, p, o) \mid g, s, p, o \in \text{RDFTerm}$
- ▶ Compound Event
  - $CE = (q_1, \dots, q_i, \dots, q_n) \mid q_i = (g, s_i, p_i, o_i)$
  - Ability to put more things in an event

- Subscription Model

- ▶ Filter language based on a subset of SPARQL

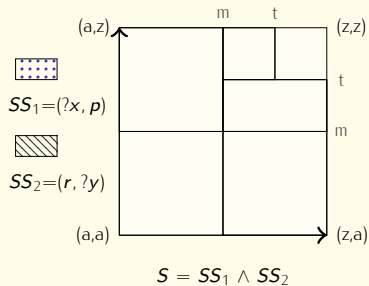
```

1 SELECT ?g ?name WHERE {
2     GRAPH ?g {
3         ?user foaf:name "John Doe" .
4         ?user foaf:age ?age
5     } FILTER (?age >= 18 && ?age <= 25)
6 }
```

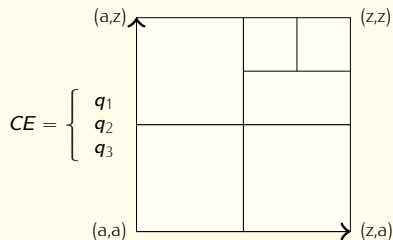
- ▶ Two sub-subscriptions (SSs): one on line 3 and one on line 4–5
- ▶ Subscription matched by a subset of quadruples from a CE

# Indexing subscriptions and events

- Indexing a subscription

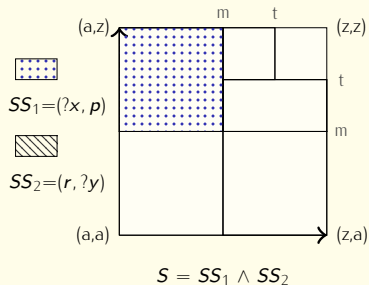


- Indexing a compound event

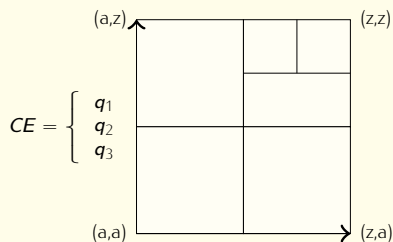


# Indexing subscriptions and events

- Indexing a subscription

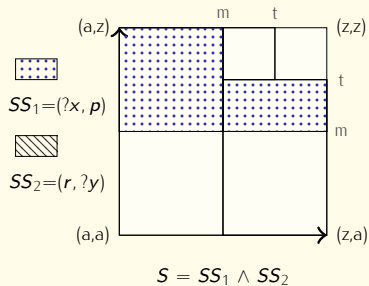


- Indexing a compound event

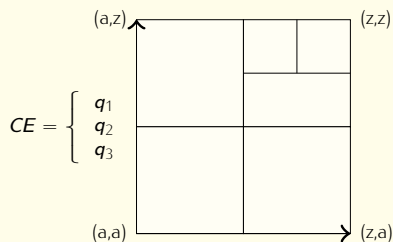


# Indexing subscriptions and events

- Indexing a subscription



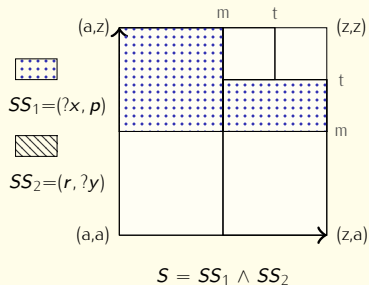
- Indexing a compound event



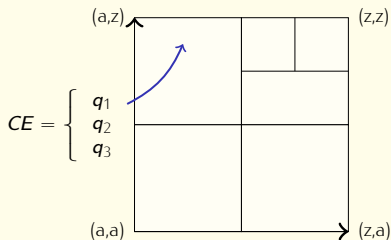


# Indexing subscriptions and events

- Indexing a subscription

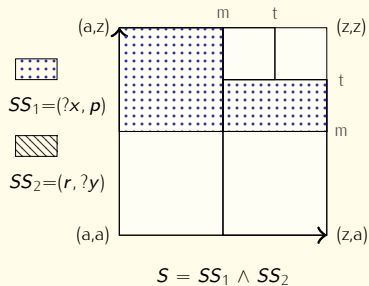


- Indexing a compound event

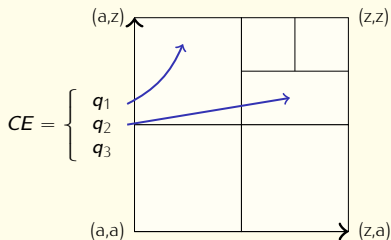


# Indexing subscriptions and events

- Indexing a subscription

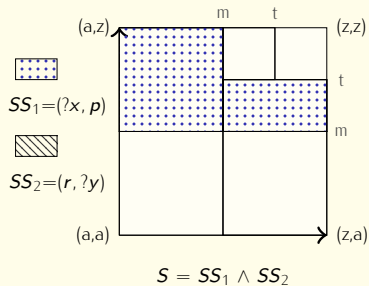


- Indexing a compound event

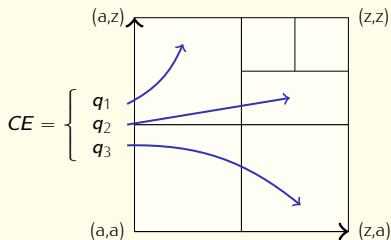


# Indexing subscriptions and events

- Indexing a subscription

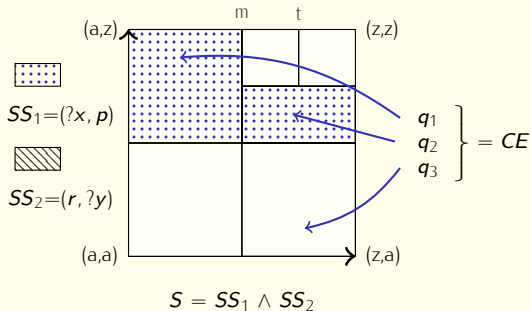


- Indexing a compound event



# Indexing subscriptions and events

- Indexing a subscription
- Indexing a compound event



Perform matching

# Publish/Subscribe Algorithms

## 1 Chained Semantic Matching Algorithm (CSMA)

- ▶ Inspired from the original idea of CSBV<sup>13</sup>
- ▶ Events published in parallel
- ▶ Subscriptions matched sequentially (chain)
- ▶ Subscribers may require a final filtering operation to prevent duplicates

## 2 One-step Semantic Matching Algorithm (OSMA)

- ▶ Subscriptions matched in one step
- ▶ Improves matching time (i.e. delivery time perceived by subscribers)
- ▶ Avoids duplicate notifications that may be received per CE on subscribers with CSMA

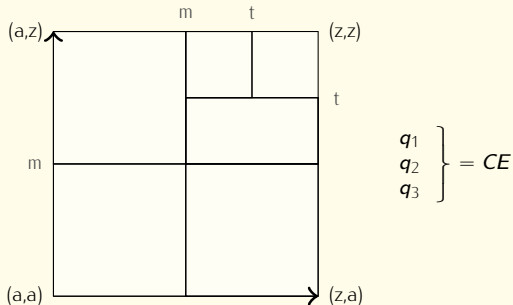
---

<sup>13</sup> LIAROU, IDREOS, and KOUBARAKIS, "Continuous RDF query processing over DHTs".

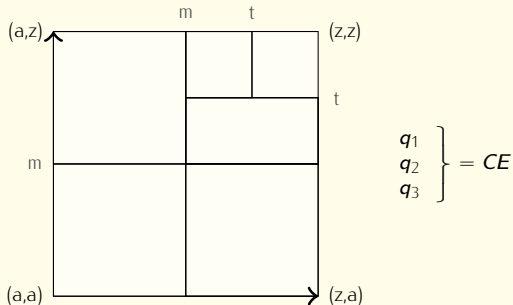
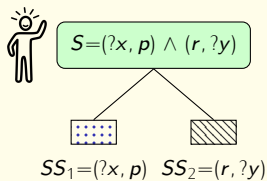
# Chained Semantic Matching Algorithm (CSMA)



$$S = (?x, p) \wedge (r, ?y)$$



# Chained Semantic Matching Algorithm (CSMA)



# Chained Semantic Matching Algorithm (CSMA)

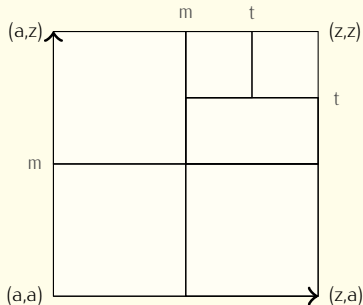


$$S = (?x, p) \wedge (r, ?y)$$



$$SS_1 = (?x, p) \quad SS_2 = (r, ?y)$$

Indexing  $S$  according to  $SS_1$



$$\left. \begin{array}{l} q_1 \\ q_2 \\ q_3 \end{array} \right\} = CE$$



# Chained Semantic Matching Algorithm (CSMA)

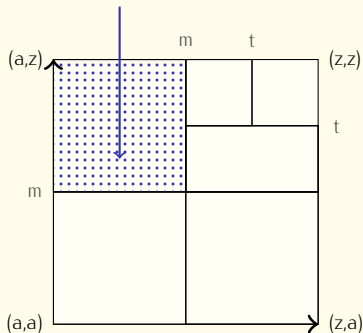


$$S = (?x, p) \wedge (r, ?y)$$



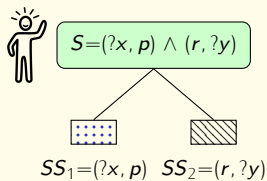
$$SS_1 = (?x, p) \quad SS_2 = (r, ?y)$$

Indexing  $S$  according to  $SS_1$

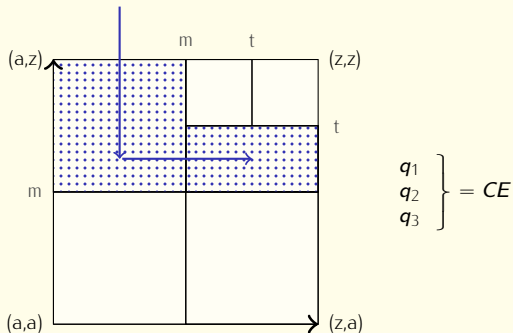


$$\left. \begin{array}{l} q_1 \\ q_2 \\ q_3 \end{array} \right\} = CE$$

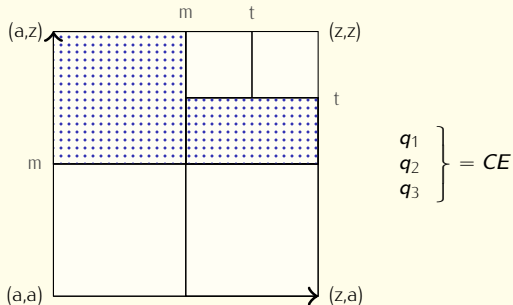
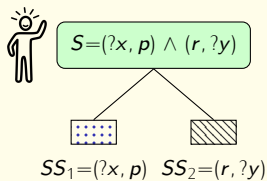
# Chained Semantic Matching Algorithm (CSMA)



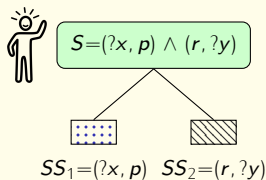
Indexing  $S$  according to  $SS_1$



# Chained Semantic Matching Algorithm (CSMA)



# Chained Semantic Matching Algorithm (CSMA)

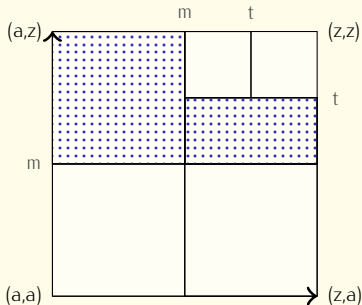


Indexing  $CE$

$$q_1 = (h, p)$$

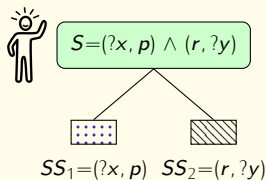
$$q_2 = (x, q)$$

$$q_3 = (r, d)$$



$$\left. \begin{array}{l} q_1 \\ q_2 \\ q_3 \end{array} \right\} = CE$$

# Chained Semantic Matching Algorithm (CSMA)

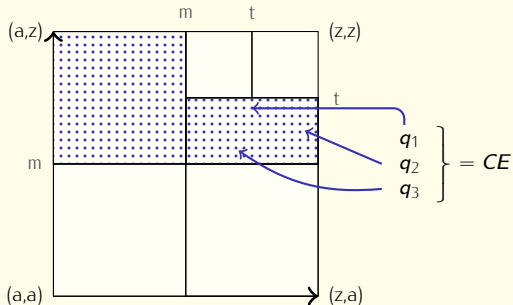


Indexing  $CE$

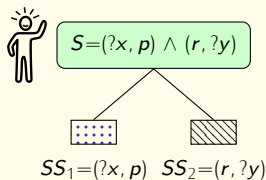
$$q_1 = (h, p)$$

$$q_2 = (x, q)$$

$$q_3 = (r, d)$$



# Chained Semantic Matching Algorithm (CSMA)

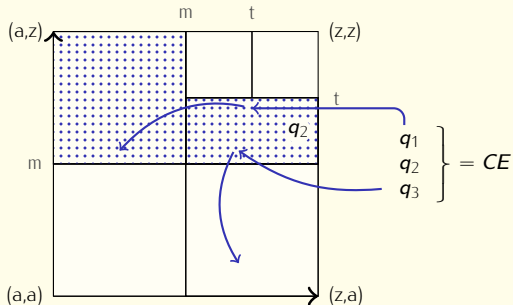


Indexing  $CE$

$$q_1 = (h, p)$$

$$q_2 = (x, q)$$

$$q_3 = (r, d)$$



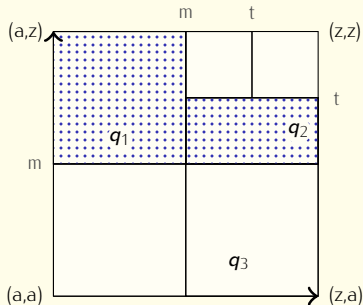
# Chained Semantic Matching Algorithm (CSMA)



$$S = (?x, p) \wedge (r, ?y)$$

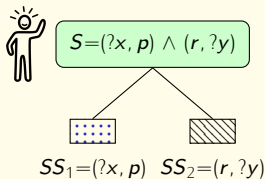


$$SS_1 = (?x, p) \quad SS_2 = (r, ?y)$$



$$\left. \begin{array}{l} q_1 \\ q_2 \\ q_3 \end{array} \right\} = CE$$

# Chained Semantic Matching Algorithm (CSMA)

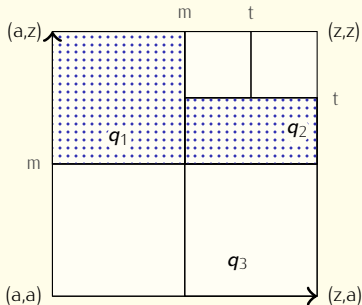


Performing matching between  
subscriptions and events

$q_2$  does not satisfy  $SS_1$

$q_3$  matches no subscription

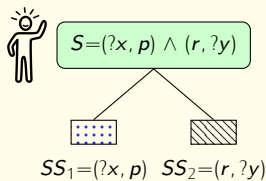
$q_1$  matches  $SS_1$



$\left. \begin{array}{l} q_1 \\ q_2 \\ q_3 \end{array} \right\} = CE$

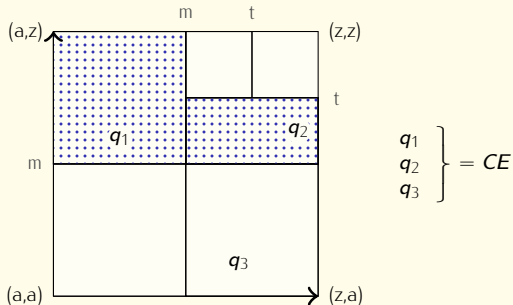


# Chained Semantic Matching Algorithm (CSMA)

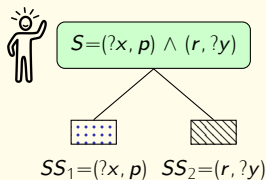


Rewriting  $S$  into  $S'$

$$S' = (r, ?y)$$

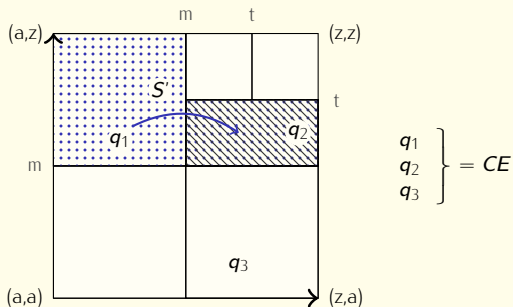


# Chained Semantic Matching Algorithm (CSMA)

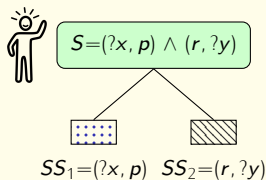


Rewriting  $S$  into  $S'$

$$S' = (r, ?y)$$

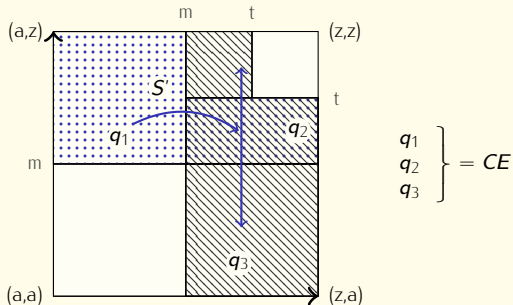


# Chained Semantic Matching Algorithm (CSMA)

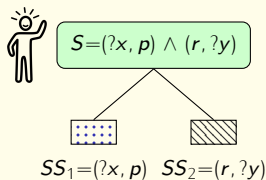


Rewriting  $S$  into  $S'$

$$S' = (r, ?y)$$



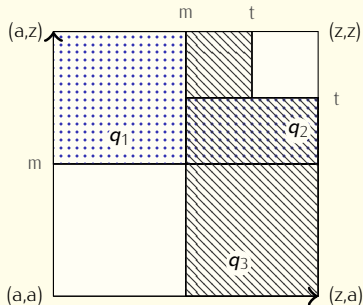
# Chained Semantic Matching Algorithm (CSMA)



Checking subscriptions satisfaction

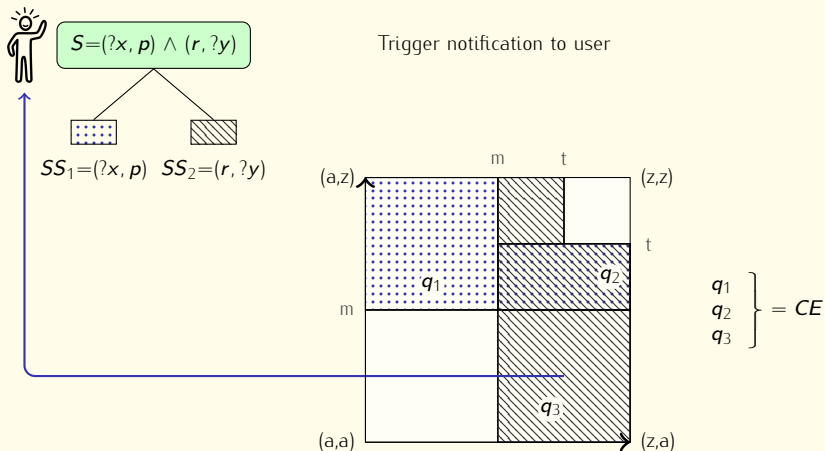
$q_2$  is not satisfying  $SS_1$

$q_3$  is satisfying  $SS_2$



$\left. \begin{array}{l} q_1 \\ q_2 \\ q_3 \end{array} \right\} = CE$

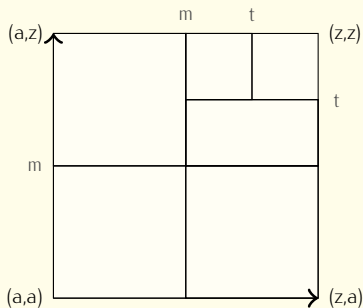
# Chained Semantic Matching Algorithm (CSMA)



# One-step Semantic Matching Algorithm (OSMA)



$$S = (?x, p) \wedge (r, ?y)$$



$$\left. \begin{array}{l} q_1 \\ q_2 \\ q_3 \end{array} \right\} = CE$$

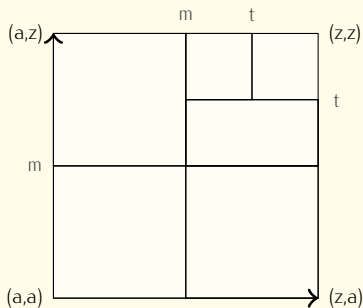
# One-step Semantic Matching Algorithm (OSMA)



$$S = (?x, p) \wedge (r, ?y)$$



$$SS_1 = (?x, p) \quad SS_2 = (r, ?y)$$



$$\left. \begin{matrix} q_1 \\ q_2 \\ q_3 \end{matrix} \right\} = CE$$

# One-step Semantic Matching Algorithm (OSMA)

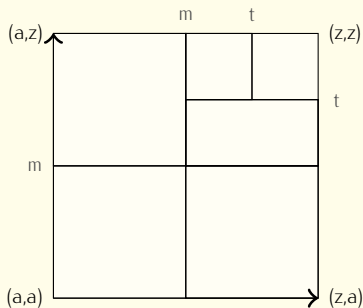


$$S = (?x, p) \wedge (r, ?y)$$



$$SS_1 = (?x, p) \quad SS_2 = (r, ?y)$$

Indexing  $S$  according to  $S_1$



$$\left. \begin{array}{l} q_1 \\ q_2 \\ q_3 \end{array} \right\} = CE$$



# One-step Semantic Matching Algorithm (OSMA)

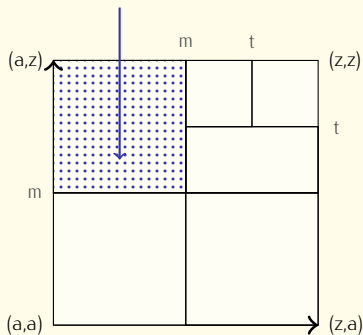


$$S = (?x, p) \wedge (r, ?y)$$



$$SS_1 = (?x, p) \quad SS_2 = (r, ?y)$$

Indexing  $S$  according to  $S_1$



$$\left. \begin{array}{l} q_1 \\ q_2 \\ q_3 \end{array} \right\} = CE$$

# One-step Semantic Matching Algorithm (OSMA)

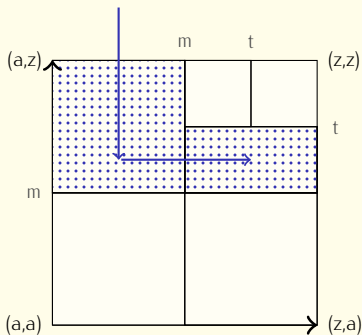


$$S = (?x, p) \wedge (r, ?y)$$



$$SS_1 = (?x, p) \quad SS_2 = (r, ?y)$$

Indexing  $S$  according to  $S_1$



$$\left. \begin{array}{l} q_1 \\ q_2 \\ q_3 \end{array} \right\} = CE$$

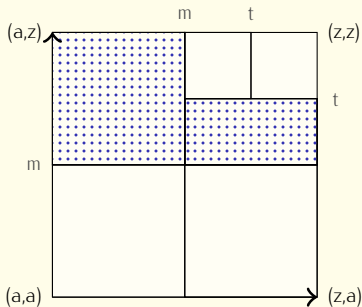
# One-step Semantic Matching Algorithm (OSMA)



$$S = (?x, p) \wedge (r, ?y)$$



$$SS_1 = (?x, p) \quad SS_2 = (r, ?y)$$



$$\left. \begin{array}{l} q_1 \\ q_2 \\ q_3 \end{array} \right\} = CE$$

# One-step Semantic Matching Algorithm (OSMA)



$$S = (?x, p) \wedge (r, ?y)$$



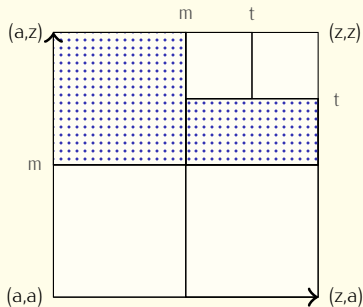
$$SS_1 = (?x, p) \quad SS_2 = (r, ?y)$$

Indexing  $CE$

$$q_1 = (h, p)$$

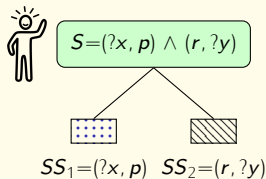
$$q_2 = (x, q)$$

$$q_3 = (r, d)$$



$$\left. \begin{array}{l} q_1 \\ q_2 \\ q_3 \end{array} \right\} = CE$$

# One-step Semantic Matching Algorithm (OSMA)

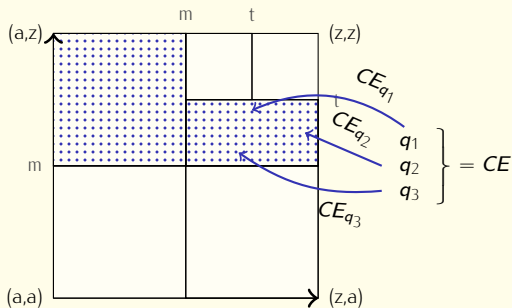


Indexing  $CE$

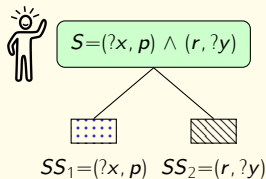
$$q_1 = (h, p)$$

$$q_2 = (x, q)$$

$$q_3 = (r, d)$$



# One-step Semantic Matching Algorithm (OSMA)

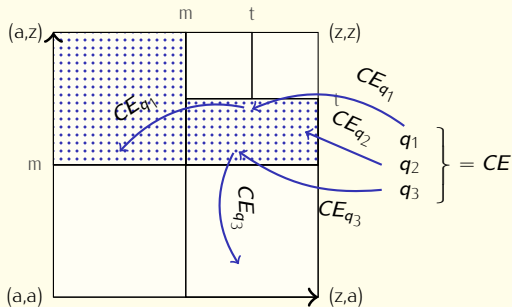


Indexing  $CE$

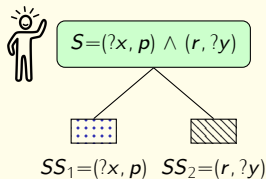
$$q_1 = (h, p)$$

$$q_2 = (x, q)$$

$$q_3 = (r, d)$$



# One-step Semantic Matching Algorithm (OSMA)

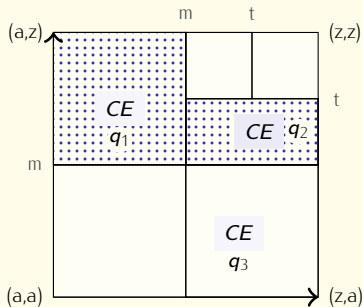


Indexing  $CE$

$$q_1 = (h, p)$$

$$q_2 = (x, q)$$

$$q_3 = (r, d)$$



$$\left. \begin{array}{l} q_1 \\ q_2 \\ q_3 \end{array} \right\} = CE$$

# One-step Semantic Matching Algorithm (OSMA)

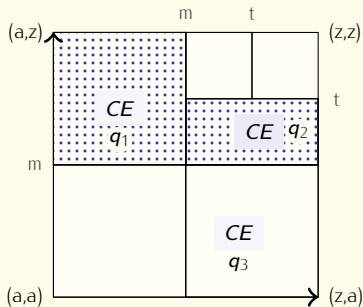


$$S = (?x, p) \wedge (r, ?y)$$



$$SS_1 = (?x, p) \quad SS_2 = (r, ?y)$$

Performing matching between  
subscriptions and events



$$\left. \begin{array}{l} q_1 \\ q_2 \\ q_3 \end{array} \right\} = CE$$



# One-step Semantic Matching Algorithm (OSMA)

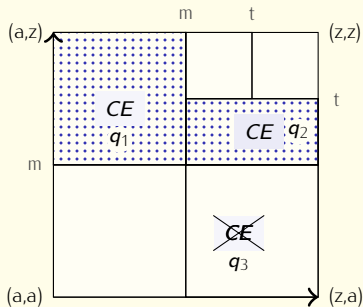


$$S = (?x, p) \wedge (r, ?y)$$



$$SS_1 = (?x, p) \quad SS_2 = (r, ?y)$$

$q_3$  and the *CE* sent along with the quadruple matches no subscription



$$\left. \begin{array}{l} q_1 \\ q_2 \\ q_3 \end{array} \right\} = CE$$

# One-step Semantic Matching Algorithm (OSMA)

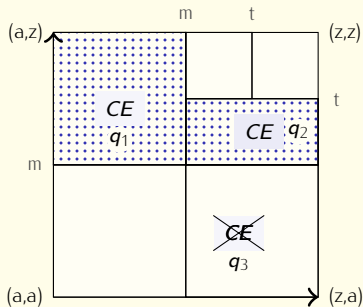


$$S = (?x, p) \wedge (r, ?y)$$



$$SS_1 = (?x, p) \quad SS_2 = (r, ?y)$$

Peers indexing  $q_1$  and  $q_2$   
are each satisfying  $S$



$$\left. \begin{array}{l} q_1 \\ q_2 \\ q_3 \end{array} \right\} = CE$$

# One-step Semantic Matching Algorithm (OSMA)

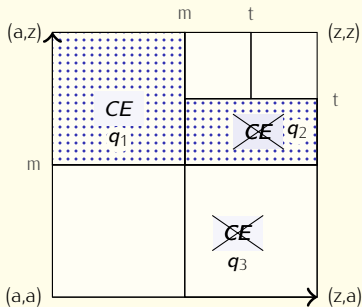


$$S = (?x, p) \wedge (r, ?y)$$



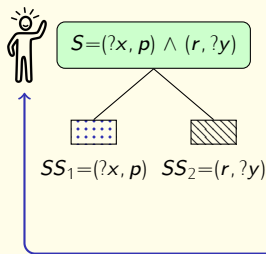
$$SS_1 = (?x, p) \quad SS_2 = (r, ?y)$$

The subscriber is notified by the peer responsible for the first of the matching quadruples contained by the *CE*

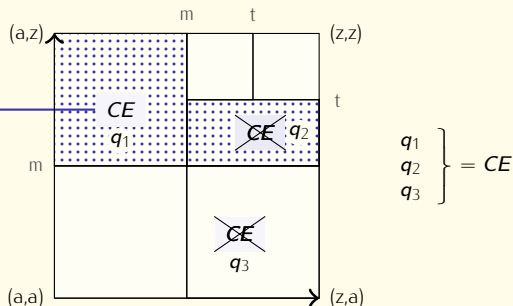


$$\left. \begin{array}{l} q_1 \\ q_2 \\ q_3 \end{array} \right\} = CE$$

# One-step Semantic Matching Algorithm (OSMA)



The subscriber is notified by the peer responsible for the first of the matching quadruples contained by the *CE*



# CSMA vs OSMA

	<b>Routed Element</b>	<b>Matching Steps</b>	<b>Duplicates</b>	<b>Happen-Before</b>
<b>CSMA</b>	Individual quadruples	Multiple, Chain-like and Reconstruction	Yes, filtering required	Enforced
<b>OSMA</b>	Whole <i>Compound Event</i>	Single	No	Requires CSMA

**Table:** Comparison of the two publish/subscribe algorithms.

# CSMA vs OSMA

	Routed Element	Matching Steps	Duplicates	Happen-Before
CSMA	Individual quadruples	Multiple, Chain-like and Reconstruction	Yes, filtering required	Enforced
OSMA	Whole <i>Compound Event</i>	Single	No	Requires CSMA

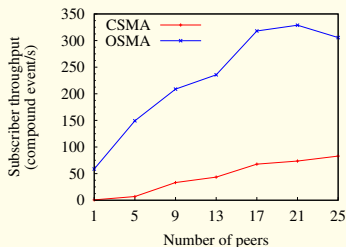
Table: Comparison of the two publish/subscribe algorithms.

Experiments are required to confirm the promising behavior of OSMA

## Experimental setup

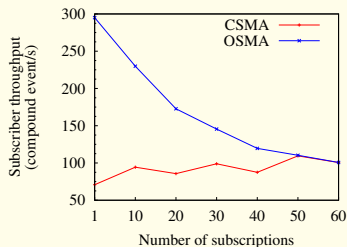
- Up to 25 nodes from the French Grid'5000 testbed
  - ▶ Xeon E5520@2,26GHz
  - ▶ 32 GB RAM
  - ▶ 7200 RPM HDD
- Each result is the average execution time for 6 runs
- Workload made of synthetic events
  - ▶ Each CE embeds 5 quadruples
  - ▶ Size per CE is about 670 Bytes
- Subscriptions are path queries
  - ▶ Each subscription embeds 5 SSs
  - ▶  $(?g, ?s1, p1, ?o1) \wedge (?g, ?o1, p2, ?o2) \wedge \dots \wedge (?g, ?o_{k-1}, p_k, ?o_k)$

# Experiments



(a) Impact of overlay size

*OSMA outperforms CSMA by a factor of 5.43 because it does not require to rewrite subscriptions.*



(b) Impact of subscriptions

*CSMA throughput remains almost stable because rewritten subscriptions involves more peers.*

## Trade-off between CSMA and OSMA

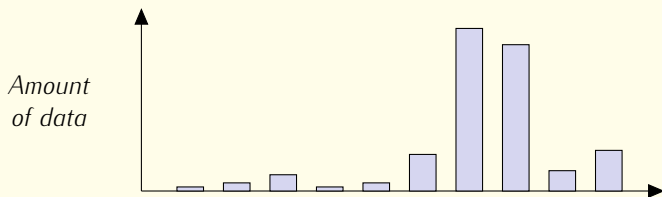


# Outline

- 1 Introduction
- 2 Distributed RDF storage
- 3 Distributed RDF pub/sub
- 4 Distributed RDF load balancing
  - Background
  - Load balancing solution
  - Evaluation
- 5 Implementation
- 6 Conclusion

# Background

- RDF data are highly skewed
  - ▶ Some RDF terms occur more often than others (e.g. *rdf:type*)
    - According to Kotoulas<sup>14</sup> the most popular term appear around 10-20% of all others



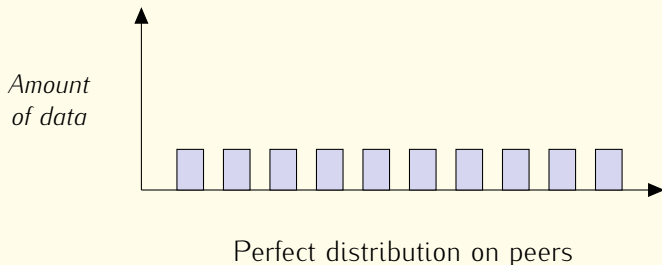
Skewed distribution on peers

---

<sup>14</sup> KOTOULAS, OREN, and VAN HARMELEN, "Mind the data skew: distributed inferencing by speeddating in elastic regions".

# Background

- RDF data are highly skewed
  - ▶ Some RDF terms occur more often than others (e.g. *rdf:type*)
    - According to Kotoulas<sup>14</sup> the most popular term appear around 10-20% of all others



---

<sup>14</sup> KOTOULAS, OREN, and VAN HARMELEN, "Mind the data skew: distributed inferencing by speeddating in elastic regions".

## Related works

### Static

BAYERS *et al.*

BATTRÉ *et al.*

RCAN

Meghdoot

### Dynamic

GODFREY *et al.*

BIENKOWSKY *et al.*

VU *et al.*

Mercury

Load balancing decision during join/leave

Usually one resource considered

Solutions that often differ by minor but subtle changes

## Related works

### Static

BAYERS *et al.*

BATTRÉ *et al.*

RCAN

Meghdoot

### Dynamic

GODFREY *et al.*

BIENKOWSKY *et al.*

VU *et al.*

Mercury

Load balancing decision during join/leave

Usually one resource considered

Solutions that often differ by minor but subtle changes

Our goal is to propose a dynamic solution that is flexible enough to balance RDF data on peers but also other resources

# Proposed solution

## ① Detecting load imbalances

# Proposed solution

- 1 Detecting load imbalances
  - ▶ Measuring load

# Proposed solution

- 1 Detecting load imbalances
  - ▶ Measuring load
    - Define resources/criteria to consider

RDF Data  
Query load  
Subscription load



# Proposed solution

- 1 Detecting load imbalances
  - ▶ Measuring load
    - Define resources/criteria to consider
  - ▶ Deciding about imbalance

# Proposed solution

- 1 Detecting load imbalances
  - ▶ Measuring load
    - Define resources/criteria to consider
  - ▶ Deciding about imbalance
    - Define imbalance states

Overloaded  
Normal  
Underloaded

# Proposed solution

## 1 Detecting load imbalances

- ▶ Measuring load
  - Define resources/criteria to consider
- ▶ Deciding about imbalance
  - Define imbalance states
  - Propose load state estimation algorithm

```
Data:  $C, M, E, K_1, K_2$ 
Result: LoadState
for  $i, m \in M$  do
    //  $i$ , load measurement index
    //  $m$ , load measurement value
    if  $m \geq E[i] \times K_1[i]$  then
        return
        LoadState(Overloaded,  $C[i]$ )
    if  $m < E[i] \times K_2[i]$  then
        return
        LoadState(Underloaded,  $C[i]$ )
return LoadState(Normal)
```

# Proposed solution

## 1 Detecting load imbalances

- ▶ Measuring load
  - Define resources/criteria to consider
- ▶ Deciding about imbalance
  - Define imbalance states
  - Propose load state estimation algorithm
  - Define how threshold is estimated

- **Absolute strategy**
  - ▶ Define threshold based on local information (e.g. Hard disk drive space capacity)
- **Relative strategy**
  - ▶ Aggregate information from other peers to estimate network load and use it as threshold

# Proposed solution

- ① Detecting load imbalances
  - ▶ Measuring load
    - Define resources/criteria to consider
  - ▶ Deciding about imbalance
    - Define imbalance states
    - Propose load state estimation algorithm
    - Define how threshold is estimated
- ② Balancing the load

# Proposed solution

- ① Detecting load imbalances
  - ▶ Measuring load
    - Define resources/criteria to consider
  - ▶ Deciding about imbalance
    - Define imbalance states
    - Propose load state estimation algorithm
    - Define how threshold is estimated
- ② Balancing the load
  - ▶ Selecting imbalance receiver

# Proposed solution

## ① Detecting load imbalances

- ▶ Measuring load
  - Define resources/criteria to consider
- ▶ Deciding about imbalance
  - Define imbalance states
  - Propose load state estimation algorithm
  - Define how threshold is estimated

## ② Balancing the load

- ▶ Selecting imbalance receiver
  - Use neighbor(s), allocate new peer or leverage information exchanged for estimating threshold values

# Proposed solution

## ① Detecting load imbalances

- ▶ Measuring load
  - Define resources/criteria to consider
- ▶ Deciding about imbalance
  - Define imbalance states
  - Propose load state estimation algorithm
  - Define how threshold is estimated

## ② Balancing the load

- ▶ Selecting imbalance receiver
  - Use neighbor(s), allocate new peer or leverage information exchanged for estimating threshold values
- ▶ Executing load balancing (per criteria)



# Proposed solution

## ① Detecting load imbalances

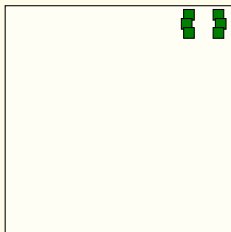
- ▶ Measuring load
  - Define resources/criteria to consider
- ▶ Deciding about imbalance
  - Define imbalance states
  - Propose load state estimation algorithm
  - Define how threshold is estimated

## ② Balancing the load

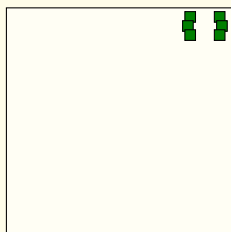
- ▶ Selecting imbalance receiver
  - Use neighbor(s), allocate new peer or leverage information exchanged for estimating threshold values
- ▶ Executing load balancing (per criteria)
  - Taking advantage of join/leave operations through virtual peers

## Executing scale-out load balancing for RDF data

- Partition zones based on centroid not middle
- Record statistical information about distribution
  - ▶ Using background threads to reduce overhead



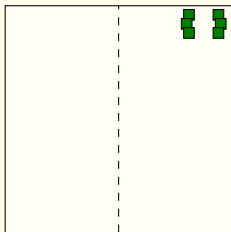
(a) Cutting based on middle



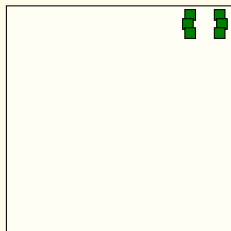
(b) Cutting based on centroid

## Executing scale-out load balancing for RDF data

- Partition zones based on centroid not middle
- Record statistical information about distribution
  - ▶ Using background threads to reduce overhead



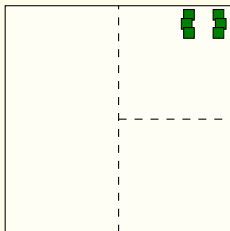
(a) Cutting based on middle



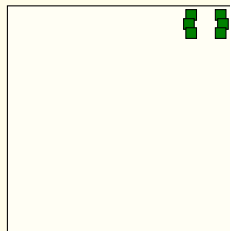
(b) Cutting based on centroid

## Executing scale-out load balancing for RDF data

- Partition zones based on centroid not middle
- Record statistical information about distribution
  - ▶ Using background threads to reduce overhead



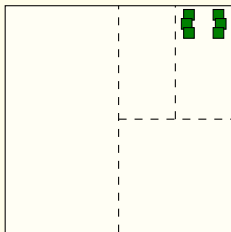
(a) Cutting based on middle



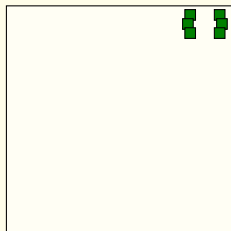
(b) Cutting based on centroid

## Executing scale-out load balancing for RDF data

- Partition zones based on centroid not middle
- Record statistical information about distribution
  - ▶ Using background threads to reduce overhead



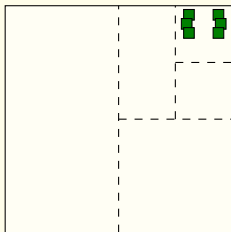
(a) Cutting based on middle



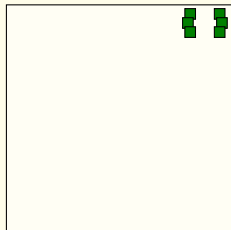
(b) Cutting based on centroid

## Executing scale-out load balancing for RDF data

- Partition zones based on centroid not middle
- Record statistical information about distribution
  - ▶ Using background threads to reduce overhead



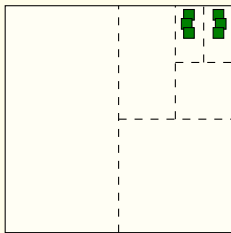
(a) Cutting based on middle



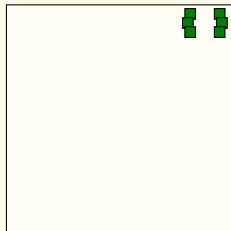
(b) Cutting based on centroid

## Executing scale-out load balancing for RDF data

- Partition zones based on centroid not middle
- Record statistical information about distribution
  - ▶ Using background threads to reduce overhead



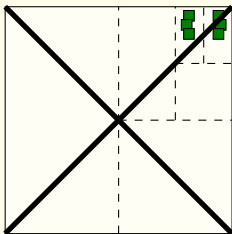
(a) Cutting based on middle



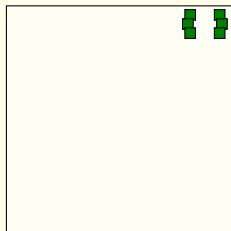
(b) Cutting based on centroid

## Executing scale-out load balancing for RDF data

- Partition zones based on centroid not middle
- Record statistical information about distribution
  - ▶ Using background threads to reduce overhead



(a) Cutting based on middle

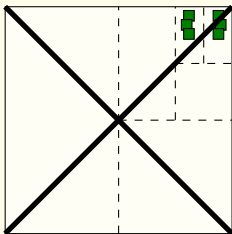


(b) Cutting based on centroid

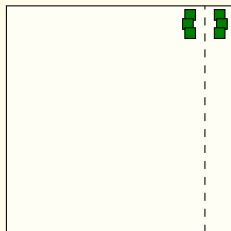


## Executing scale-out load balancing for RDF data

- Partition zones based on centroid not middle
- Record statistical information about distribution
  - ▶ Using background threads to reduce overhead



(a) Cutting based on middle



(b) Cutting based on centroid

## Experiments

- Workload is real data extracted from a Twitter data flow
  - ▶ Converted to our RDF data model with a python adapter
  - ▶ Size is about  $10^5$  quadruples
- Considered RDF data as load balancing criterion
- Virtual peers with peers allocation only (scale-out)
  - ▶ Using 32 peers

Static load balancing		Dynamic load balancing	
<i>Middle</i>	<i>Centroid</i>	<i>Absolute</i>	<i>Relative</i>
559.4%	69.5%	119.75%	96.57%

**Table:** Load balancing strategies comparison using relative stddev.

# Experiments

- Workload is real data extracted from a Twitter data flow
  - ▶ Converted to our RDF data model with a python adapter
  - ▶ Size is about  $10^5$  quadruples
- Considered RDF data as load balancing criterion
- Virtual peers with peers allocation only (scale-out)
  - ▶ Using 32 peers

Static load balancing		Dynamic load balancing	
<i>Middle</i>	<i>Centroid</i>	<i>Absolute</i>	<i>Relative</i>
559.4%	69.5%	119.75%	96.57%

**Table:** Load balancing strategies comparison using relative stddev.

# Experiments

- Workload is real data extracted from a Twitter data flow
  - ▶ Converted to our RDF data model with a python adapter
  - ▶ Size is about  $10^5$  quadruples
- Considered RDF data as load balancing criterion
- Virtual peers with peers allocation only (scale-out)
  - ▶ Using 32 peers

Static load balancing		Dynamic load balancing	
<i>Middle</i>	<i>Centroid</i>	<i>Absolute</i>	<i>Relative</i>
559.4%	69.5%	119.75%	96.57%

**Table:** Load balancing strategies comparison using relative stddev.

# Outline

- 1 Introduction
- 2 Distributed RDF storage
- 3 Distributed RDF pub/sub
- 4 Distributed RDF load balancing
- 5 Implementation
  - Middleware design
  - Performance tuning
- 6 Conclusion

# Middleware design (EventCloud)

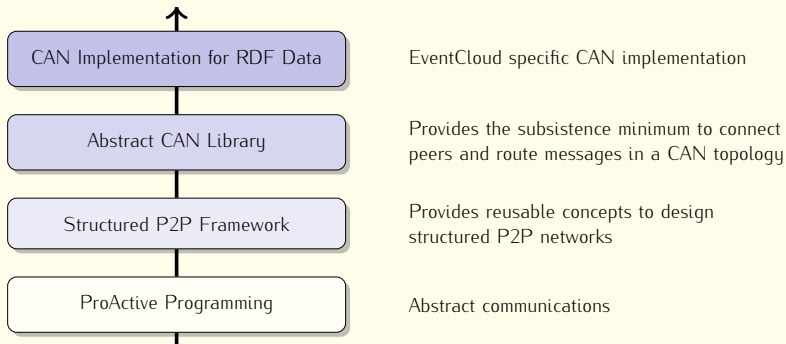


Figure: Stack of main software blocks designed and/or used.

# Implementation-level performance and tuning aspects

- Serialization (*section 6.2.2*)
  - ▶ Prevent message payload marshalling/unmarshalling at each routing step
- Local storage (*section 6.2.3*)
  - ▶ Proposed delayer mechanism to perform bulk processing
  - ▶ Investigated parameters to improve throughput
- Multi-active objects (*section 6.2.1*)
  - ▶ Hard and soft limit definition
    - Required due to datastructures contention and I/O accesses
  - ▶ Support for scheduling requests according to priorities
    - Purpose is to avoid starvation

# Outline

- 1 Introduction
- 2 Distributed RDF storage
- 3 Distributed RDF pub/sub
- 4 Distributed RDF load balancing
- 5 Implementation
- 6 Conclusion



# Conclusion

- Contribution
  - ▶ Middleware devoted to storing, retrieving synchronously but also disseminating selectively and asynchronously RDF data
    - Synchronous RDF data indexing and retrieval
    - Asynchronous RDF data indexing and retrieval
    - RDF data load-balancing
    - Modular architecture with reusable abstractions



# Perspectives

- Increasing reliability and availability
  - ▶ Safe recovery in case of peers failure
    - Replication (e.g. checkpointing or state machines)
- Optimizing query and subscriptions evaluation
  - ▶ Improving query plan execution
  - ▶ Subscriptions summarization
- Reasoning over RDF data

Thank you for your attention

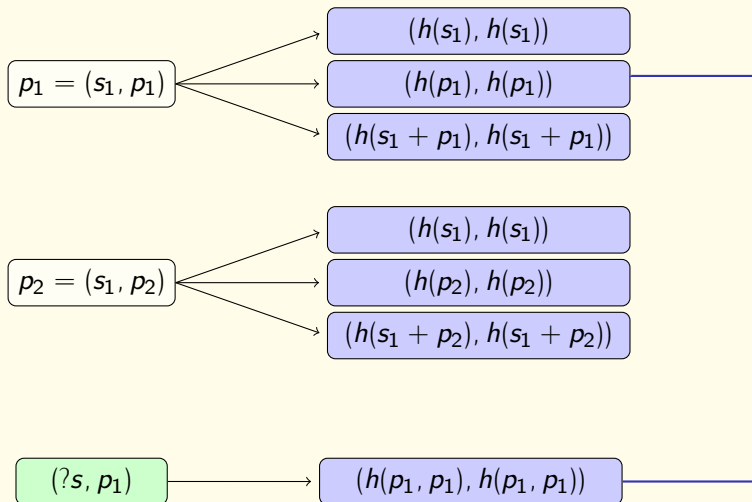


# Publications

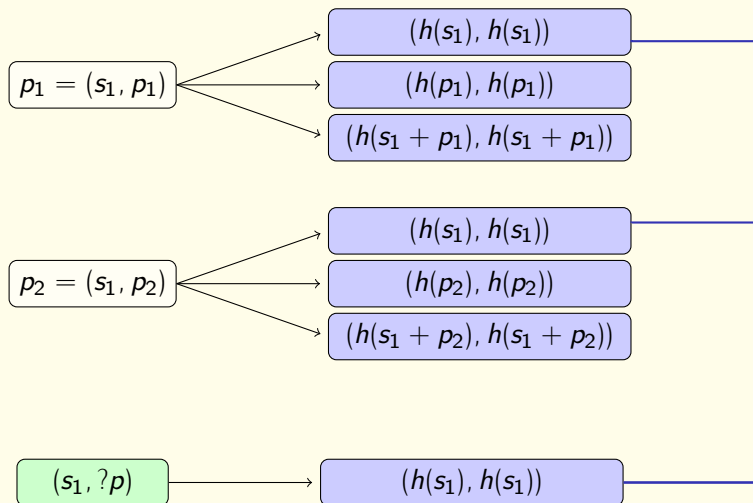
-  Imen FILALI, Laurent PELLEGRINO, Francesco BONGIOVANNI, Fabrice HUET, Françoise BAUDE, et al. "Modular P2P-based approach for RDF data storage and retrieval". In: *Proceedings of the international conference on Advances in P2P Systems*. 2011.
-  Laurent PELLEGRINO, Iyad ALSHABANI, Françoise BAUDE, Roland STUEHMER, and Nenad STOJANOVIC. "An Approach for Efficiently Combining Real-time and Past Events for Ubiquitous Business Processing". In: *International workshop on Semantic Business Process Management (SBPM)*. 2012.
-  Laurent PELLEGRINO, Françoise BAUDE, and Iyad ALSHABANI. "Towards a Scalable Cloud-based RDF Storage Offering a Pub/Sub Query Service". In: *International conference on Cloud Computing, GRIDs, and Virtualization*. 2012.
-  Laurent PELLEGRINO, Fabrice HUET, Françoise BAUDE, and Amjad ALSHABANI. "A Distributed Publish/Subscribe System for RDF Data". In: *Proceedings of the international conference on Data Management in Cloud, Grid and P2P Systems (Globe)*. Springer, 2013.
-  Quirino ZAGARESE, Gerardo CANFORA, Eugenio ZIMEO, Iyad ALSHABANI, Laurent PELLEGRINO, and Françoise BAUDE. "Efficient data-intensive event-driven interaction in SOA". In: *Proceedings of the ACM Symposium on Applied Computing (SAC)*. Accepted special issue SCP journal. ACM. 2013.

Backup slides

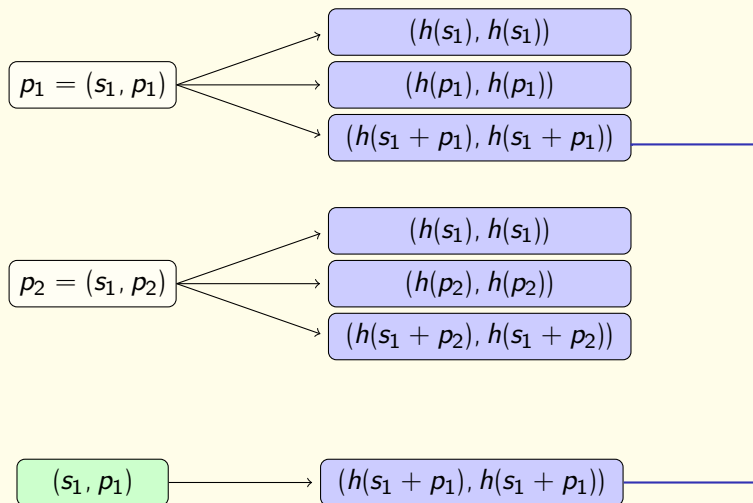
# Hash based indexing using indexes



# Hash based indexing using indexes

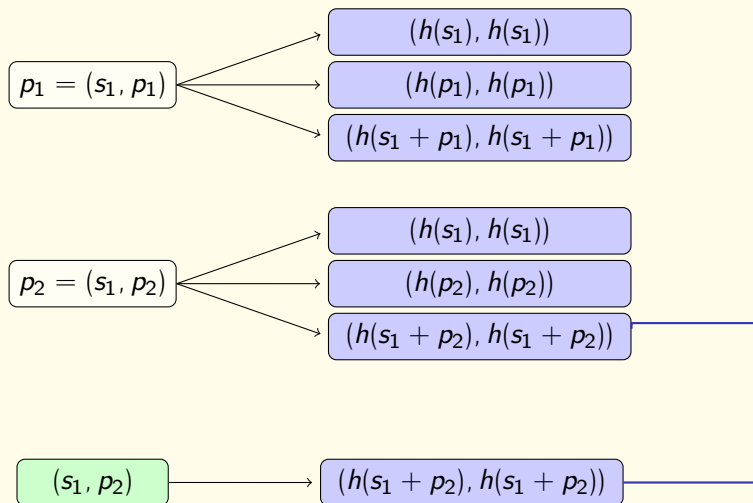


# Hash based indexing using indexes



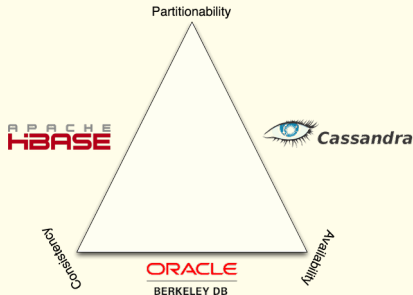


# Hash based indexing using indexes



# CAP theorem

- Any practical database can only provide 2 of the 3 desirable properties: *Consistency*, *Availability*, and *Partitionability*



The choice of a storage backend is a tradeoff guided by the requirements of a particular use case

# SPARQL queries support

- Distributed RDF storage (synchronous)
  - ▶ External point of view
    - Full SPARQL 1.0 support
  - ▶ Internal point of view
    - Only atomic queries and range queries (FILTERs with XPath tests operators)
- Distributed RDF pub/sub (asynchronous)
  - ▶ Limitations
    - Uses the SELECT query form
    - Contains at most one group GRAPH pattern with a graph variable
    - Returns the graph variable declared in the GRAPH pattern
  - ▶ Allowance
    - Multiple triple patterns may be used inside the graph pattern defined in the subscription
    - One or more FILTER clauses are also allowed to restrict solutions
    - Standard logical operators but also filter functions like REGEX, STRSTARTS, etc. are permitted

## Pub/sub layer requirements (1)

- R1 Events and subscriptions are assumed to be submitted to the event notification service by means of proxies
- R2 Clocks are assumed synchronized between machines inside the P2P network only
- R3 Causal ordering between publish and subscribe requests handled asynchronously from a same proxy must be enforced
- R4 Quadruples must *eventually* be stored on peers
- R5 Events that must be notified are notified

## Pub/sub layer requirements (2)

- R6 Data indexation does not rely on hashing on in order to avoid multiple indexations of the same publications
- R7 Subscribers must have the possibility to receive notifications as:
- ▶ a signal
  - ▶ a collection of bindings (values matching the variables contained by their subscription)
  - ▶ the full event that has matched their interests
- R8 No *false positives* nor *duplicates*

# Unicast routing complexity

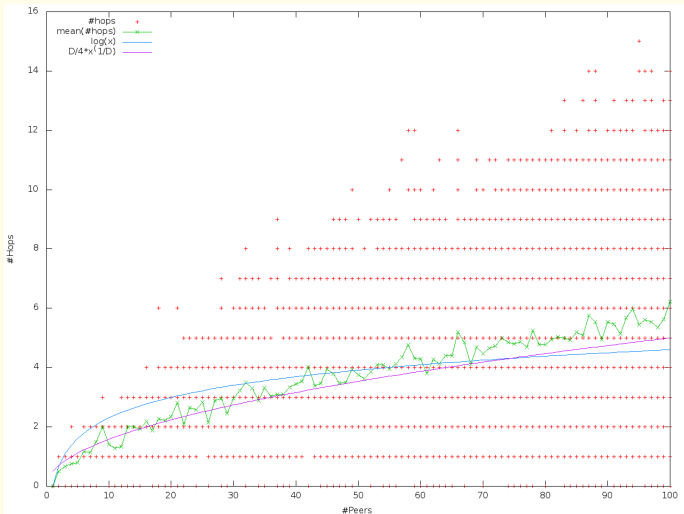
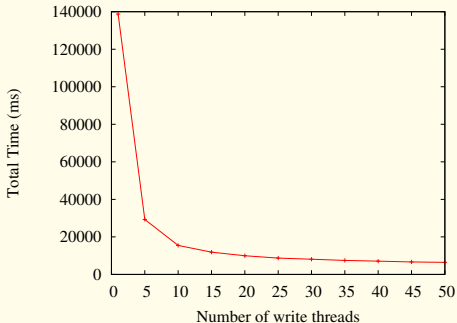


Figure: Unicast routing complexity comparison (made by Lokman RAHMANI)

# Evaluation concurrent insertions



*Evolution of the time for concurrent insertions with 300 peers cooperating.*

## CSMA – Indexing a subscription

- 1 Extract the first atomic<sup>15</sup> or range<sup>16</sup> query ( $SS$ )
- 2 Forward the subscription  $S$  on peers managing  $SS$ ' conditions
- 3 Each peer receiving  $S$ 
  - 3.1 Stores  $S$
  - 3.2 Finds quadruples matching  $SS$
  - 3.3 Rewrites  $S$  and index derived subscription or notifies subscriber

```
SELECT ?g ?name WHERE { GRAPH ?g { ?user foaf:name ?name .  
?user foaf:age ?age } FILTER (?age >= 18 && ?age <= 25) }
```

---

<sup>15</sup> Quadruple where any RDF term may be a variable

<sup>16</sup> Atomic query where some conditions may be attached to variables



## CSMA – Indexing a subscription

- 1 Extract the first atomic<sup>15</sup> or range<sup>16</sup> query ( $SS$ )
- 2 Forward the subscription  $S$  on peers managing  $SS$ ' conditions
- 3 Each peer receiving  $S$ 
  - 3.1 Stores  $S$
  - 3.2 Finds quadruples matching  $SS$
  - 3.3 Rewrites  $S$  and index derived subscription or notifies subscriber

```
SELECT ?g ?name WHERE { GRAPH ?g { ?user foaf:name ?name .  
?user foaf:age ?age } FILTER (?age >= 18 && ?age <= 25) }
```

```
?g ?user foaf:name ?name
```

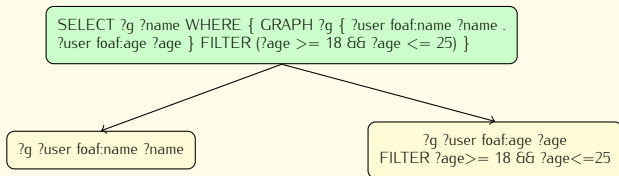
---

<sup>15</sup> Quadruple where any RDF term may be a variable

<sup>16</sup> Atomic query where some conditions may be attached to variables

## CSMA – Indexing a subscription

- 1 Extract the first atomic<sup>15</sup> or range<sup>16</sup> query ( $SS$ )
- 2 Forward the subscription  $S$  on peers managing  $SS$ ' conditions
- 3 Each peer receiving  $S$ 
  - 3.1 Stores  $S$
  - 3.2 Finds quadruples matching  $SS$
  - 3.3 Rewrites  $S$  and index derived subscription or notifies subscriber

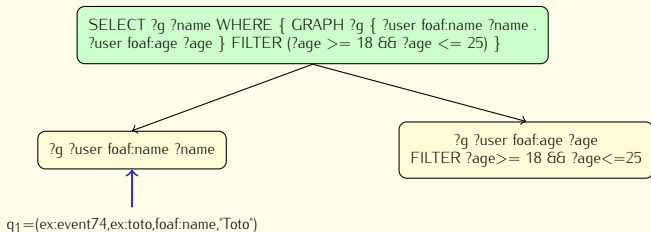


<sup>15</sup> Quadruple where any RDF term may be a variable

<sup>16</sup> Atomic query where some conditions may be attached to variables

## CSMA – Indexing a subscription

- 1 Extract the first atomic<sup>15</sup> or range<sup>16</sup> query ( $SS$ )
- 2 Forward the subscription  $S$  on peers managing  $SS'$  conditions
- 3 Each peer receiving  $S$ 
  - 3.1 Stores  $S$
  - 3.2 Finds quadruples matching  $SS$
  - 3.3 Rewrites  $S$  and index derived subscription or notifies subscriber

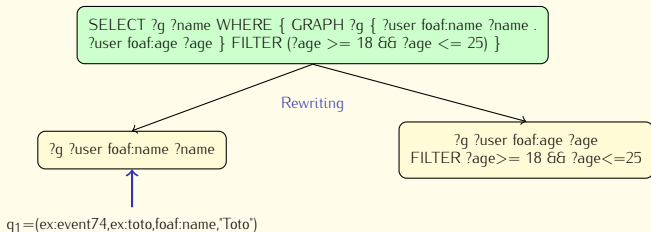


<sup>15</sup> Quadruple where any RDF term may be a variable

<sup>16</sup> Atomic query where some conditions may be attached to variables

## CSMA – Indexing a subscription

- 1 Extract the first atomic<sup>15</sup> or range<sup>16</sup> query ( $SS$ )
- 2 Forward the subscription  $S$  on peers managing  $SS'$  conditions
- 3 Each peer receiving  $S$ 
  - 3.1 Stores  $S$
  - 3.2 Finds quadruples matching  $SS$
  - 3.3 Rewrites  $S$  and index derived subscription or notifies subscriber

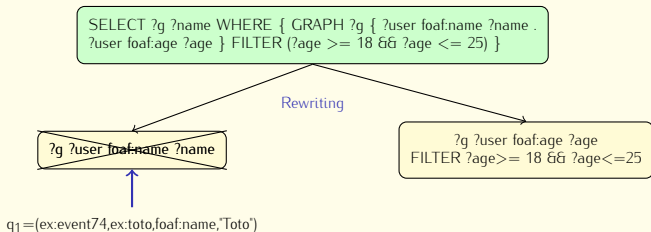


<sup>15</sup> Quadruple where any RDF term may be a variable

<sup>16</sup> Atomic query where some conditions may be attached to variables

## CSMA – Indexing a subscription

- 1 Extract the first atomic<sup>15</sup> or range<sup>16</sup> query ( $SS$ )
- 2 Forward the subscription  $S$  on peers managing  $SS'$  conditions
- 3 Each peer receiving  $S$ 
  - 3.1 Stores  $S$
  - 3.2 Finds quadruples matching  $SS$
  - 3.3 Rewrites  $S$  and index derived subscription or notifies subscriber

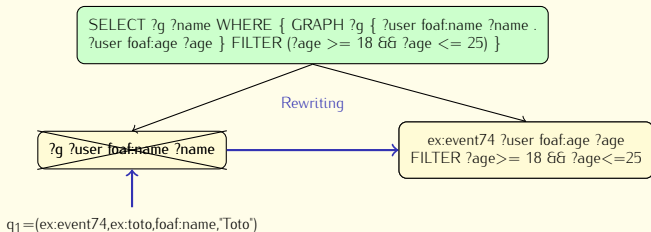


<sup>15</sup> Quadruple where any RDF term may be a variable

<sup>16</sup> Atomic query where some conditions may be attached to variables

## CSMA – Indexing a subscription

- 1 Extract the first atomic<sup>15</sup> or range<sup>16</sup> query ( $SS$ )
- 2 Forward the subscription  $S$  on peers managing  $SS'$  conditions
- 3 Each peer receiving  $S$ 
  - 3.1 Stores  $S$
  - 3.2 Finds quadruples matching  $SS$
  - 3.3 Rewrites  $S$  and index derived subscription or notifies subscriber

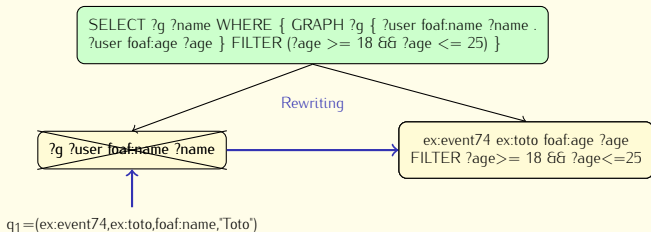


<sup>15</sup> Quadruple where any RDF term may be a variable

<sup>16</sup> Atomic query where some conditions may be attached to variables

## CSMA – Indexing a subscription

- 1 Extract the first atomic<sup>15</sup> or range<sup>16</sup> query ( $SS$ )
- 2 Forward the subscription  $S$  on peers managing  $SS$ ' conditions
- 3 Each peer receiving  $S$ 
  - 3.1 Stores  $S$
  - 3.2 Finds quadruples matching  $SS$
  - 3.3 Rewrites  $S$  and index derived subscription or notifies subscriber

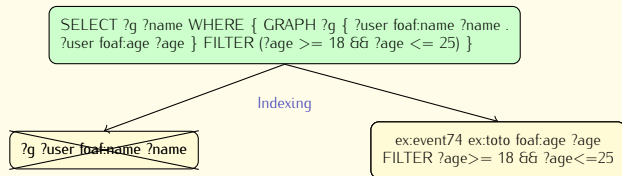


<sup>15</sup> Quadruple where any RDF term may be a variable

<sup>16</sup> Atomic query where some conditions may be attached to variables

## CSMA – Indexing a subscription

- 1 Extract the first atomic<sup>15</sup> or range<sup>16</sup> query ( $SS$ )
- 2 Forward the subscription  $S$  on peers managing  $SS$ ' conditions
- 3 Each peer receiving  $S$ 
  - 3.1 Stores  $S$
  - 3.2 Finds quadruples matching  $SS$
  - 3.3 Rewrites  $S$  and index derived subscription or notifies subscriber



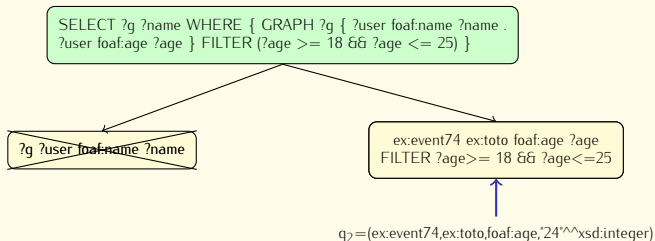
<sup>15</sup> Quadruple where any RDF term may be a variable

<sup>16</sup> Atomic query where some conditions may be attached to variables



## CSMA – Indexing a subscription

- 1 Extract the first atomic<sup>15</sup> or range<sup>16</sup> query ( $SS$ )
- 2 Forward the subscription  $S$  on peers managing  $SS'$  conditions
- 3 Each peer receiving  $S$ 
  - 3.1 Stores  $S$
  - 3.2 Finds quadruples matching  $SS$
  - 3.3 Rewrites  $S$  and index derived subscription or notifies subscriber

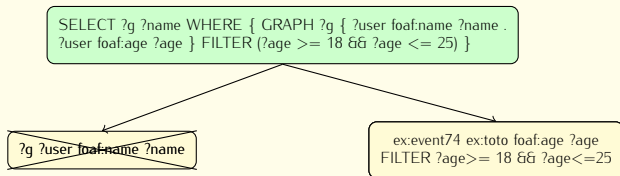


<sup>15</sup> Quadruple where any RDF term may be a variable

<sup>16</sup> Atomic query where some conditions may be attached to variables

## CSMA – Indexing a subscription

- 1 Extract the first atomic<sup>15</sup> or range<sup>16</sup> query ( $SS$ )
- 2 Forward the subscription  $S$  on peers managing  $SS'$  conditions
- 3 Each peer receiving  $S$ 
  - 3.1 Stores  $S$
  - 3.2 Finds quadruples matching  $SS$
  - 3.3 Rewrites  $S$  and index derived subscription or notifies subscriber



Subscription fully satisfied

<sup>15</sup> Quadruple where any RDF term may be a variable

<sup>16</sup> Atomic query where some conditions may be attached to variables

# CSMA

## Indexing a publication

- 1 Publish each  $q$  from a CE independently
- 2 Each peer receiving  $q$ 
  - 2.1 Stores  $q$
  - 2.2 Finds SSs satisfied by  $q$
  - 2.3 Rewrites SSs and indexes derived SSs or notifies subscribers



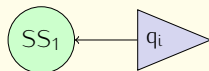
## Notification reception

- Each subscriber receiving a notification  $n$ 
  - ▶ Starts a full event reconstruction process

# CSMA

## Indexing a publication

- 1 Publish each  $q$  from a CE independently
- 2 Each peer receiving  $q$ 
  - 2.1 Stores  $q$
  - 2.2 Finds SSs satisfied by  $q$
  - 2.3 Rewrites SSs and indexes derived SSs or notifies subscribers



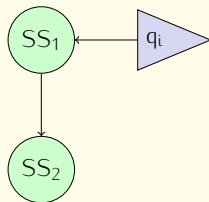
## Notification reception

- Each subscriber receiving a notification  $n$ 
  - ▶ Starts a full event reconstruction process

# CSMA

## Indexing a publication

- 1 Publish each  $q$  from a CE independently
- 2 Each peer receiving  $q$ 
  - 2.1 Stores  $q$
  - 2.2 Finds SSs satisfied by  $q$
  - 2.3 Rewrites SSs and indexes derived SSs or notifies subscribers



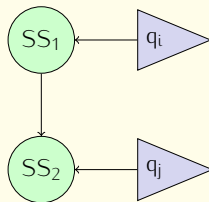
## Notification reception

- Each subscriber receiving a notification  $n$ 
  - ▶ Starts a full event reconstruction process

# CSMA

## Indexing a publication

- 1 Publish each  $q$  from a CE independently
- 2 Each peer receiving  $q$ 
  - 2.1 Stores  $q$
  - 2.2 Finds SSs satisfied by  $q$
  - 2.3 Rewrites SSs and indexes derived SSs or notifies subscribers



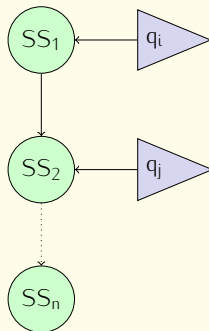
## Notification reception

- Each subscriber receiving a notification  $n$ 
  - ▶ Starts a full event reconstruction process

# CSMA

## Indexing a publication

- 1 Publish each  $q$  from a CE independently
- 2 Each peer receiving  $q$ 
  - 2.1 Stores  $q$
  - 2.2 Finds SSs satisfied by  $q$
  - 2.3 Rewrites SSs and indexes derived SSs or notifies subscribers



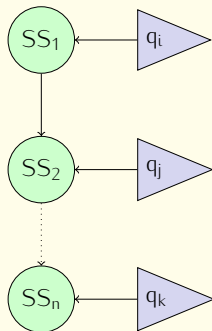
## Notification reception

- Each subscriber receiving a notification  $n$ 
  - ▶ Starts a full event reconstruction process

# CSMA

## Indexing a publication

- 1 Publish each  $q$  from a CE independently
- 2 Each peer receiving  $q$ 
  - 2.1 Stores  $q$
  - 2.2 Finds SSs satisfied by  $q$
  - 2.3 Rewrites SSs and indexes derived SSs or notifies subscribers



## Notification reception

- Each subscriber receiving a notification  $n$ 
  - ▶ Starts a full event reconstruction process



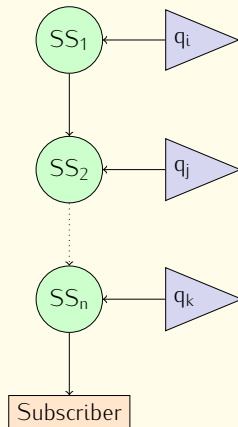
# CSMA

## Indexing a publication

- 1 Publish each  $q$  from a CE independently
- 2 Each peer receiving  $q$ 
  - 2.1 Stores  $q$
  - 2.2 Finds SSs satisfied by  $q$
  - 2.3 Rewrites SSs and indexes derived SSs or notifies subscribers

## Notification reception

- Each subscriber receiving a notification  $n$ 
  - ▶ Starts a full event reconstruction process



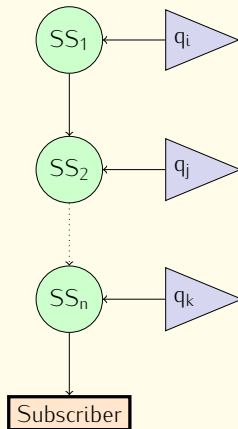
# CSMA

## Indexing a publication

- 1 Publish each  $q$  from a CE independently
- 2 Each peer receiving  $q$ 
  - 2.1 Stores  $q$
  - 2.2 Finds SSs satisfied by  $q$
  - 2.3 Rewrites SSs and indexes derived SSs or notifies subscribers

## Notification reception

- Each subscriber receiving a notification  $n$ 
  - ▶ Starts a full event reconstruction process



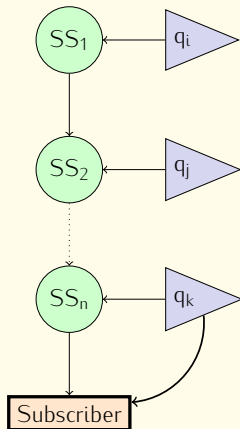
# CSMA

## Indexing a publication

- 1 Publish each  $q$  from a CE independently
- 2 Each peer receiving  $q$ 
  - 2.1 Stores  $q$
  - 2.2 Finds SSs satisfied by  $q$
  - 2.3 Rewrites SSs and indexes derived SSs or notifies subscribers

## Notification reception

- Each subscriber receiving a notification  $n$ 
  - ▶ Starts a full event reconstruction process



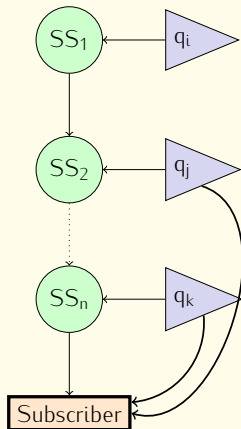
# CSMA

## Indexing a publication

- 1 Publish each  $q$  from a CE independently
- 2 Each peer receiving  $q$ 
  - 2.1 Stores  $q$
  - 2.2 Finds SSs satisfied by  $q$
  - 2.3 Rewrites SSs and indexes derived SSs or notifies subscribers

## Notification reception

- Each subscriber receiving a notification  $n$ 
  - ▶ Starts a full event reconstruction process



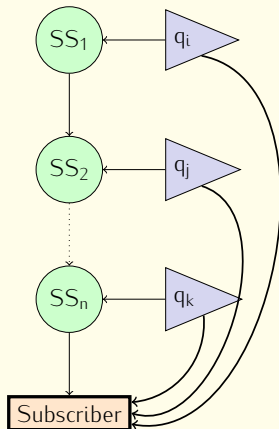
# CSMA

## Indexing a publication

- 1 Publish each  $q$  from a CE independently
- 2 Each peer receiving  $q$ 
  - 2.1 Stores  $q$
  - 2.2 Finds SSs satisfied by  $q$
  - 2.3 Rewrites SSs and indexes derived SSs or notifies subscribers

## Notification reception

- Each subscriber receiving a notification  $n$ 
  - ▶ Starts a full event reconstruction process



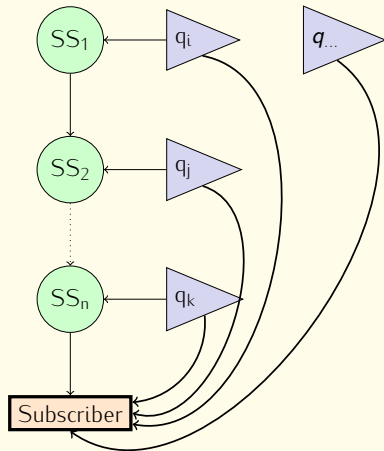
# CSMA

## Indexing a publication

- 1 Publish each  $q$  from a CE independently
- 2 Each peer receiving  $q$ 
  - 2.1 Stores  $q$
  - 2.2 Finds SSs satisfied by  $q$
  - 2.3 Rewrites SSs and indexes derived SSs or notifies subscribers

## Notification reception

- Each subscriber receiving a notification  $n$ 
  - ▶ Starts a full event reconstruction process



# OSMA

## Indexing a subscription

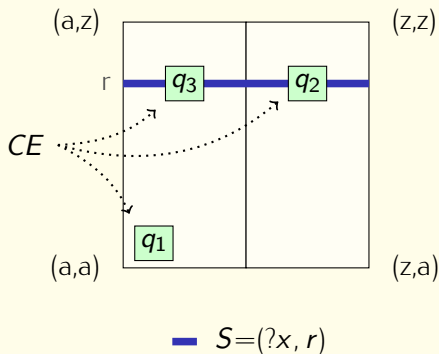
- Same as CSMA

## Indexing a publication

- 1 Send the whole CE on peers using each  $q$  as routing key
- 2 Each peer receiving the Compound Event by using  $q$  as key
  - 2.1 Stores  $q$  but not the full CE
  - 2.2 Finds subscriptions (not  $SS(s)$ ) satisfied by the CE in one step
  - 2.3 Notifies the subscriber about the CE (under one condition)
    - No reconstruction required
    - No duplicate notifications

## Avoiding duplicates with OSMA

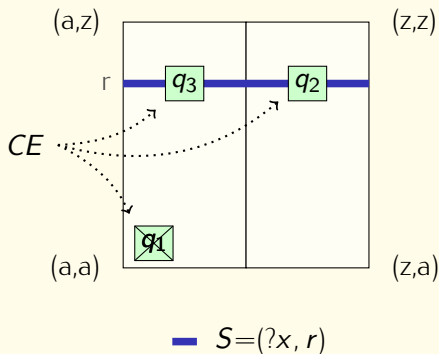
- A peer notifies a match if and only if it is responsible for the first of the matching quadruples contained by the CE





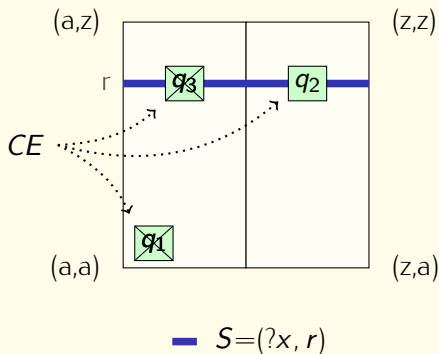
## Avoiding duplicates with OSMA

- A peer notifies a match if and only if it is responsible for the first of the matching quadruples contained by the CE



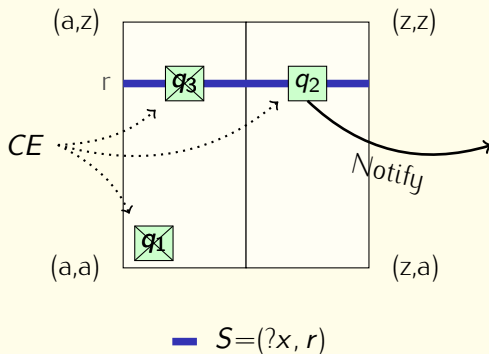
## Avoiding duplicates with OSMA

- A peer notifies a match if and only if it is responsible for the first of the matching quadruples contained by the CE

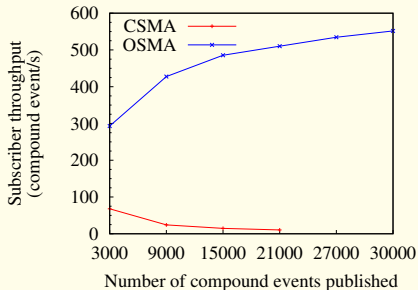


## Avoiding duplicates with OSMA

- A peer notifies a match if and only if it is responsible for the first of the matching quadruples contained by the CE

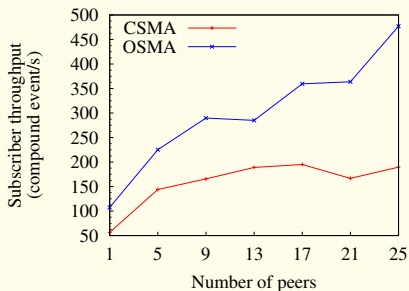


## Impact of the number of publications



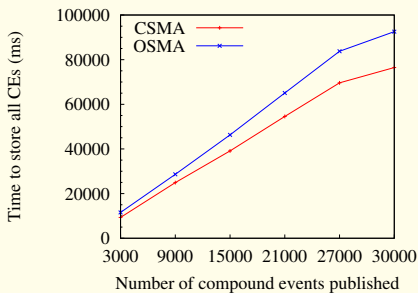
*CSMA throughput decreases quickly with the number of publications due to required reconstructions.*

## Scalability with one accept-all subscription



*In contrary to CSMA, OSMA does not generate any duplicate. Consequently, OSMA performs much better than CSMA whose the throughput stagnates.*

## Time to store publications on peers



*OSMA requires more bandwidth than CSMA since the full CE is sent along with each quadruple.*

# Background

- Load balancing goals
  - ▶ achieve maximum resources utilization
  - ▶ avoid overload
  - ▶ maximize throughput/minimize response time
  - ▶ prevent crashing
- Load balancing strategies
  - ▶ Data replication
    - Improve data access by balancing queries load but not data itself
  - ▶ Data relocation through indirections
    - Mechanisms are expensive to maintain up to date pointers
  - ▶ Multiple hash functions for indexing data
    - Increase lookup complexity
  - ▶ Virtual peers (peers relocation or allocation)
    - Maintenance overhead

# Background

- RDF data are highly skewed
  - ▶ Some RDF terms occur more often than others (e.g. *rdf:type*)
    - According to Kotoulas<sup>17</sup> the most popular term appear around 10-20% of all others
  - ▶ Many RDF term prefixes (namespaces) are the same
  - ▶ RDF terms belong to groups of characters, where members of each group surround a particular and common character, thus shaping RDF terms clusters.
- Skewness affects data distribution on peers (hot-spots)
  - ▶ Prevents system scalability

---

<sup>17</sup> KOTOULAS, OREN, and VAN HARMELEN, "Mind the data skew: distributed inferencing by speeddating in elastic regions".



# Background

- RDF data are highly skewed
  - ▶ Some RDF terms occur more often than others (e.g. *rdf:type*)
    - According to Kotoulas<sup>17</sup> the most popular term appear around 10-20% of all others
  - ▶ Many RDF term prefixes (namespaces) are the same
  - ▶ RDF terms belong to groups of characters, where members of each group surround a particular and common character, thus shaping RDF terms clusters.
- Skewness affects data distribution on peers (hot-spots)
  - ▶ Prevents system scalability

Achieve maximum resources utilization to maximize throughput  
Hashing is not the solution  
when the issue is the popularity

---

<sup>17</sup> KOTOULAS, OREN, and VAN HARMELEN, "Mind the data skew: distributed inferencing by speeddating in elastic regions".

# Background

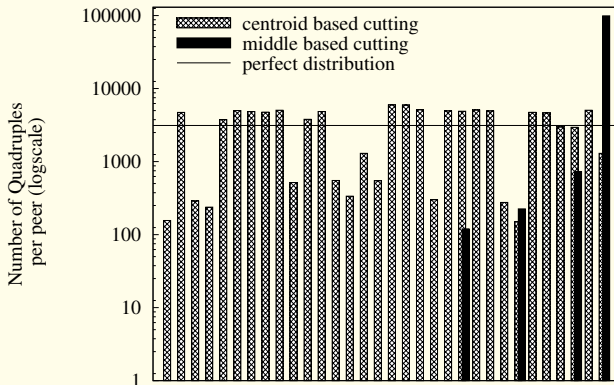
- Static load balancing
  - ▶ System load is assumed stable (no continuous insertions or deletions)
  - ▶ Load balancing decision taken during the join of a peer
- Dynamic load balancing
  - ▶ Decisions and adaptations at runtime
  - ▶ Support for endless data insertions

# Proposed solution

## Balancing load

- Selecting imbalance receiver (*setting parameter  $E$* )
  - ▶ Absolute strategy
    - Purely local decision based on parameters defined at startup
  - ▶ Relative strategy
    - Make use of a gossip protocol to estimate network load
- Virtual peers
  - ▶ Peers allocation and/or relocation
  - ▶ Reuse existing join and leave operations
  - ▶ Good compromise between overhead, implementation complexity and achievable results

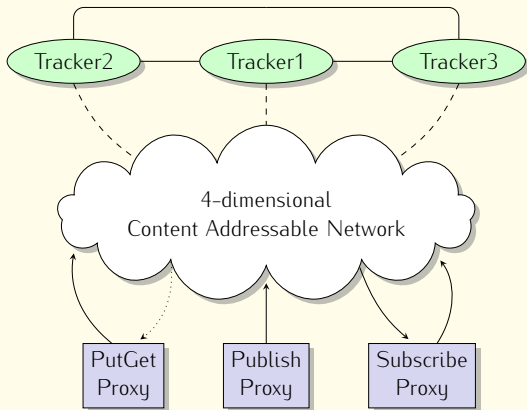
# Experiments



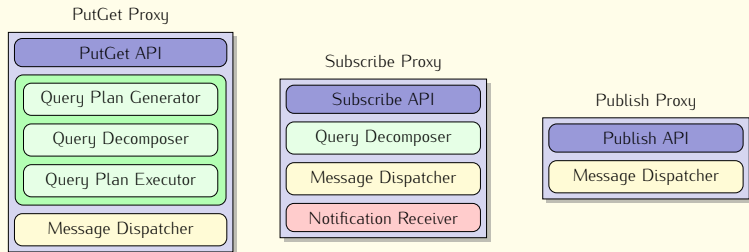
Distribution with 32 peers

Figure: Static load balancing using middle vs centroid partitioning.

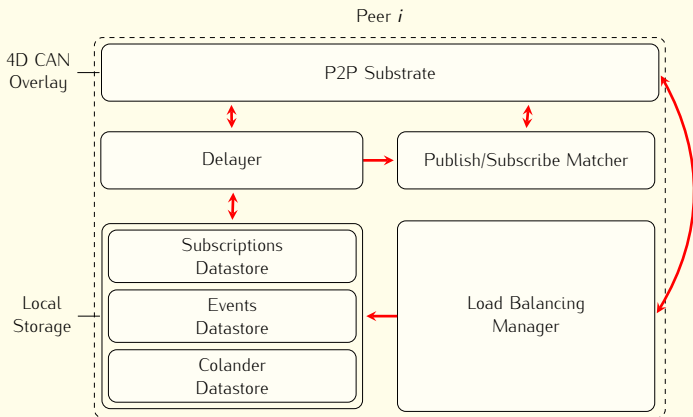
# High-level view of the EventCloud architecture



# Internal proxies architecture



# Internal peer architecture



# Priorities effect on reconstructions with CSMA

