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

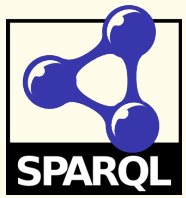
Reviewers	Ioana Manolescu	Inria Saclay - Île-de-France
	Etienne Rivière	Université de Neuchâtel
Examiners	Johan Montagnat	CNRS
	Ester Pacitti	Université de Montpellier 2
	Vivien Quéma	Grenoble INP

Outline

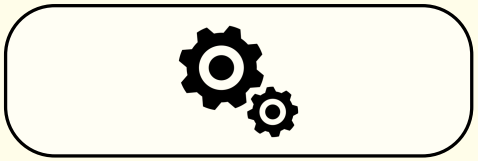
Outline

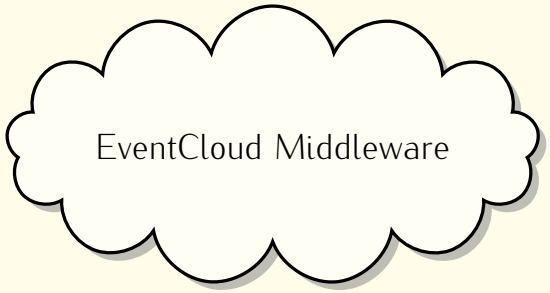
Motivation

- Exponential growing of information produced
  - Internet of Things
    - Internet scale collection of connected data sources
- Data mining
  - Prerequisite
    - Integrate data from heteregeneous 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

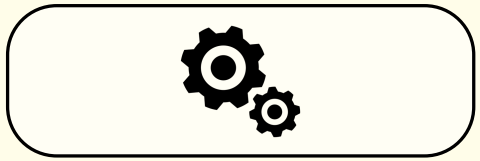


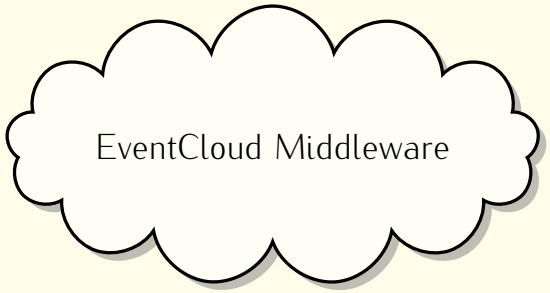


Context: PLAY and SocEDA projects

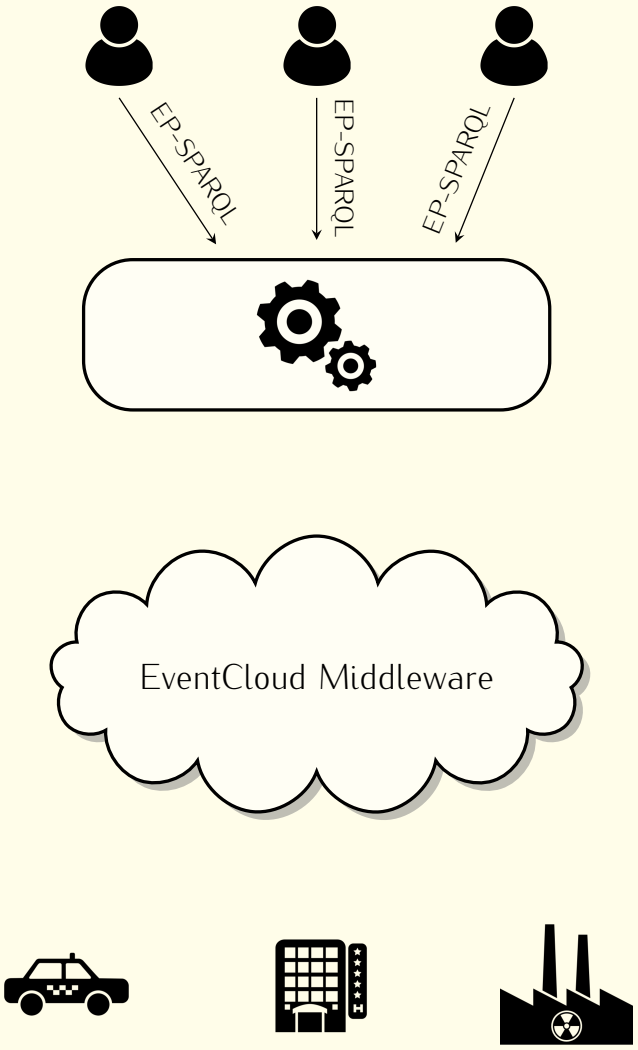
Platform users

Subscribe to detect interesting patterns and react consequently





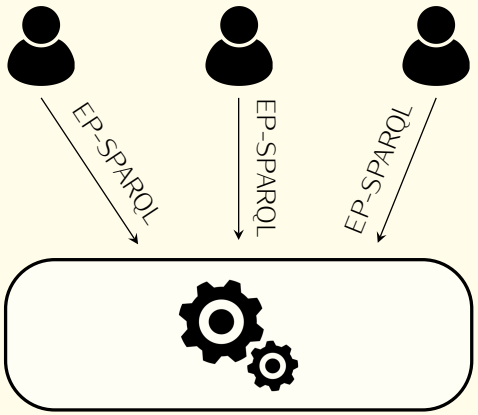
Context: PLAY and SocEDA projects



Platform users

Subscribe to detect interesting patterns and react consequently

Context: PLAY and SocEDA projects

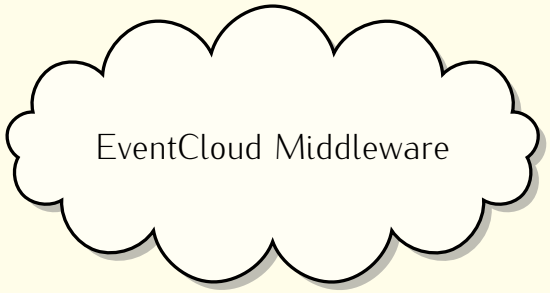


Platform users

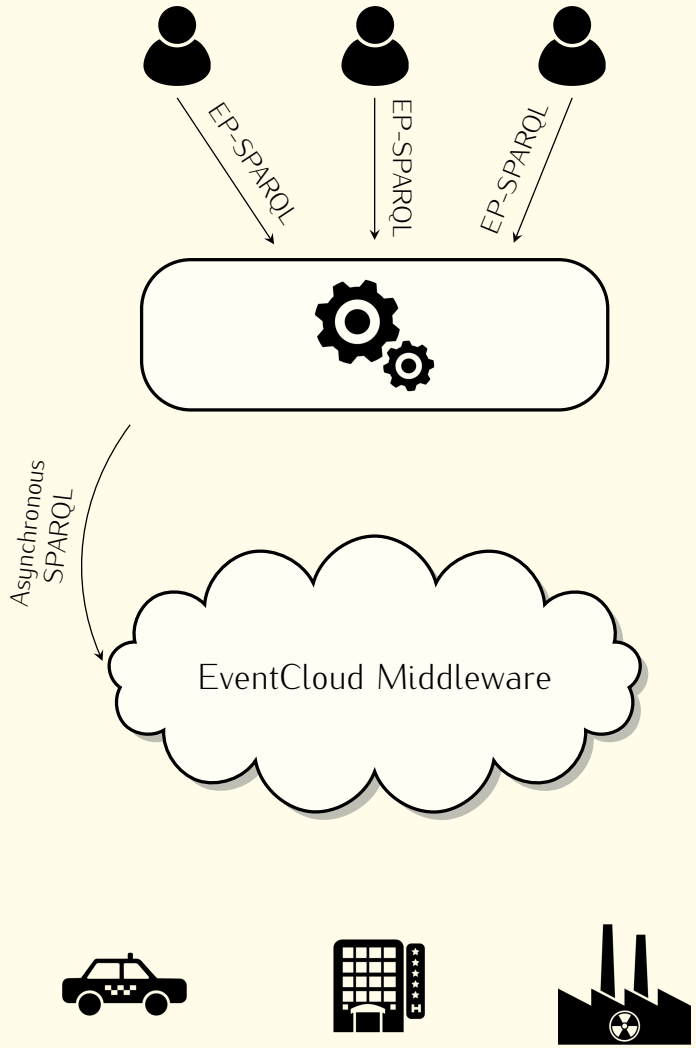
Subscribe to detect interesting patterns and react consequently

CEP

Complex Event Processing engine (Etalis/Esper)



Context: PLAY and SocEDA projects



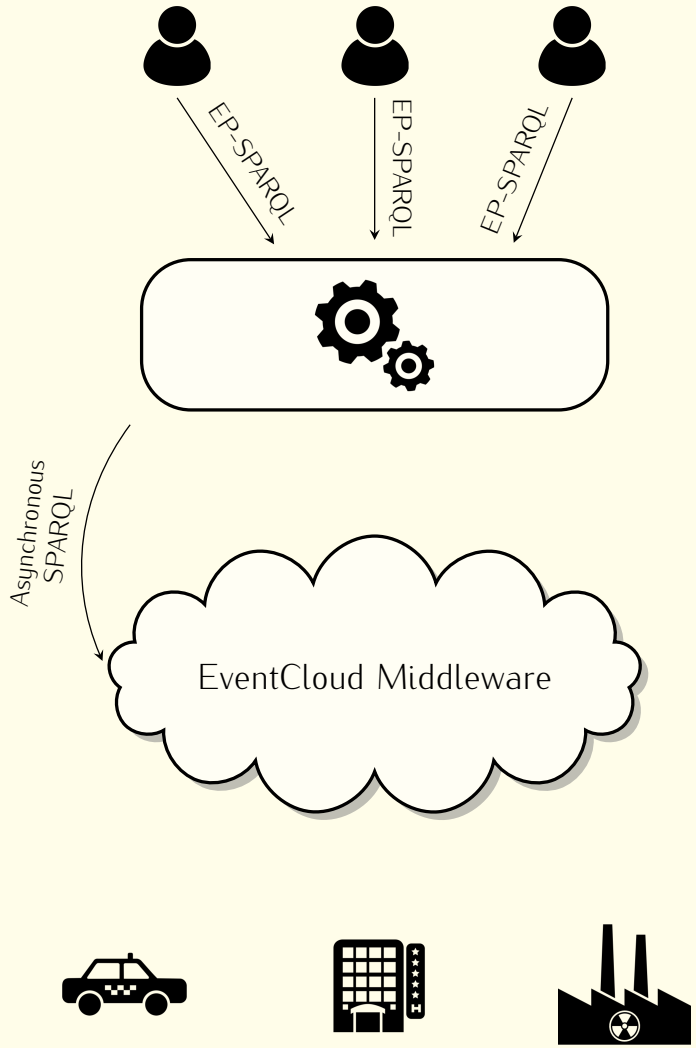
Platform users

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

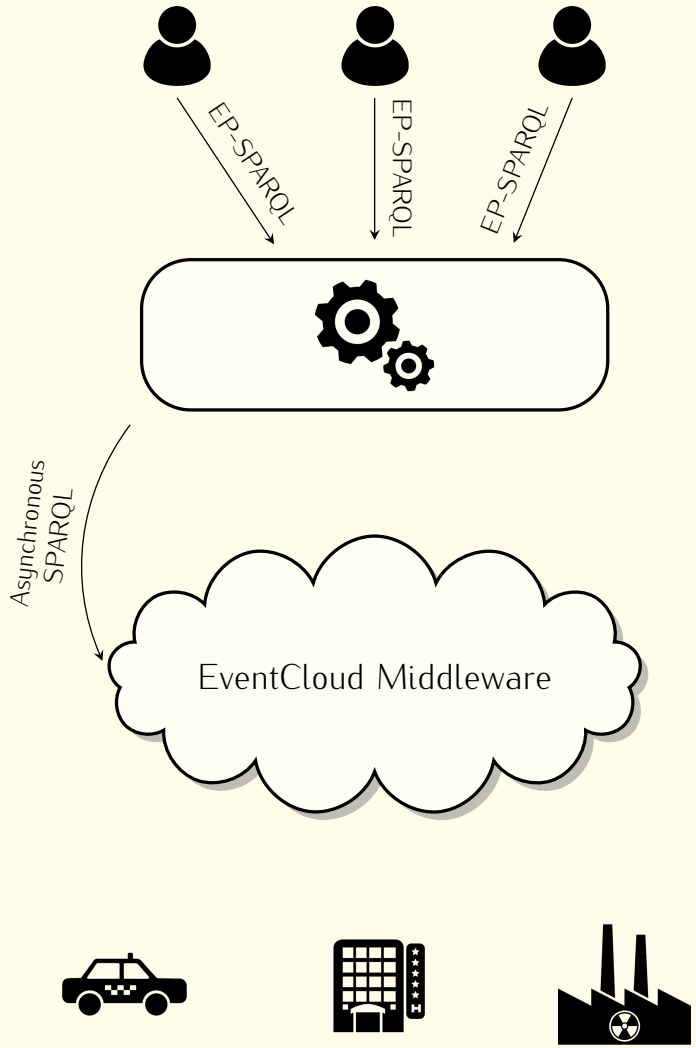
CEP

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)

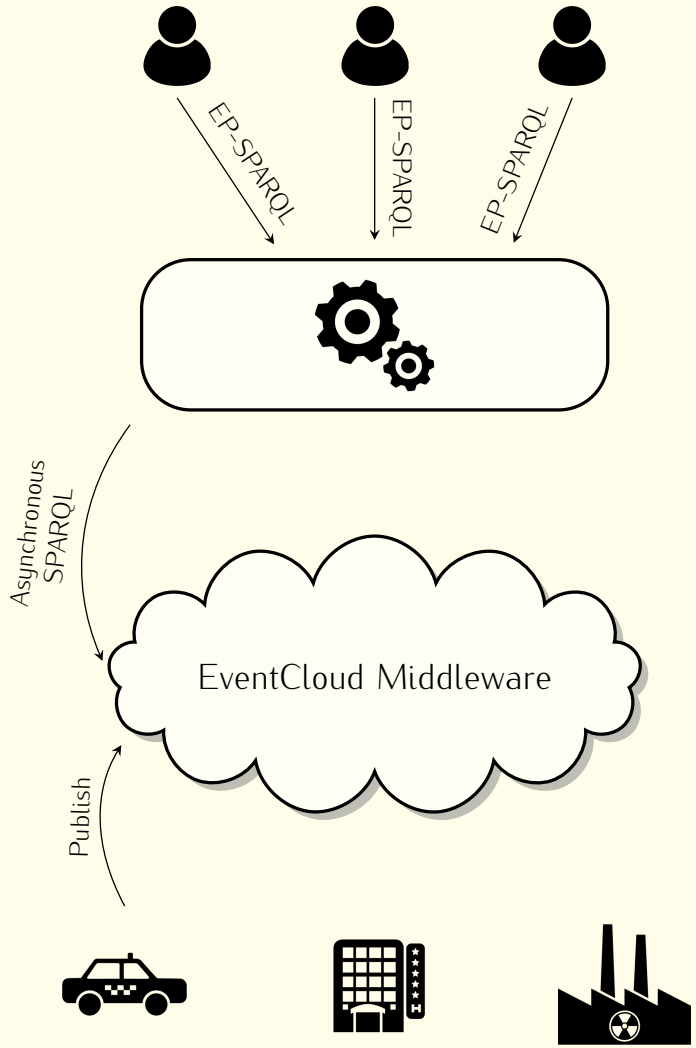
EventCloud

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)

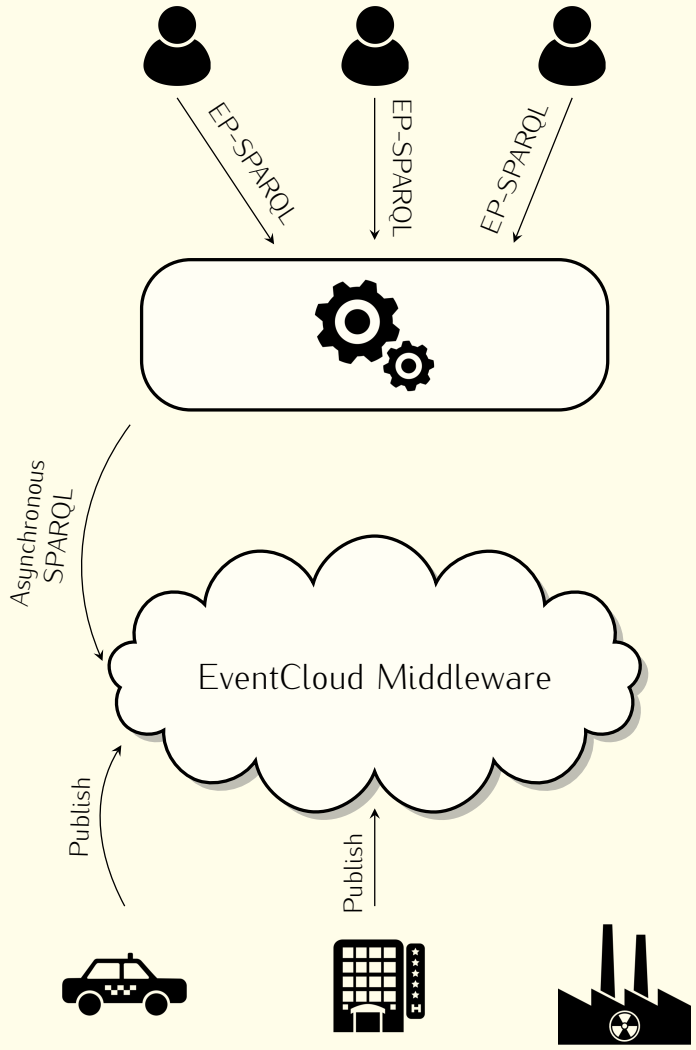
EventCloud

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 consequently

CEP

Complex Event Processing engine (Etalis/Esper)

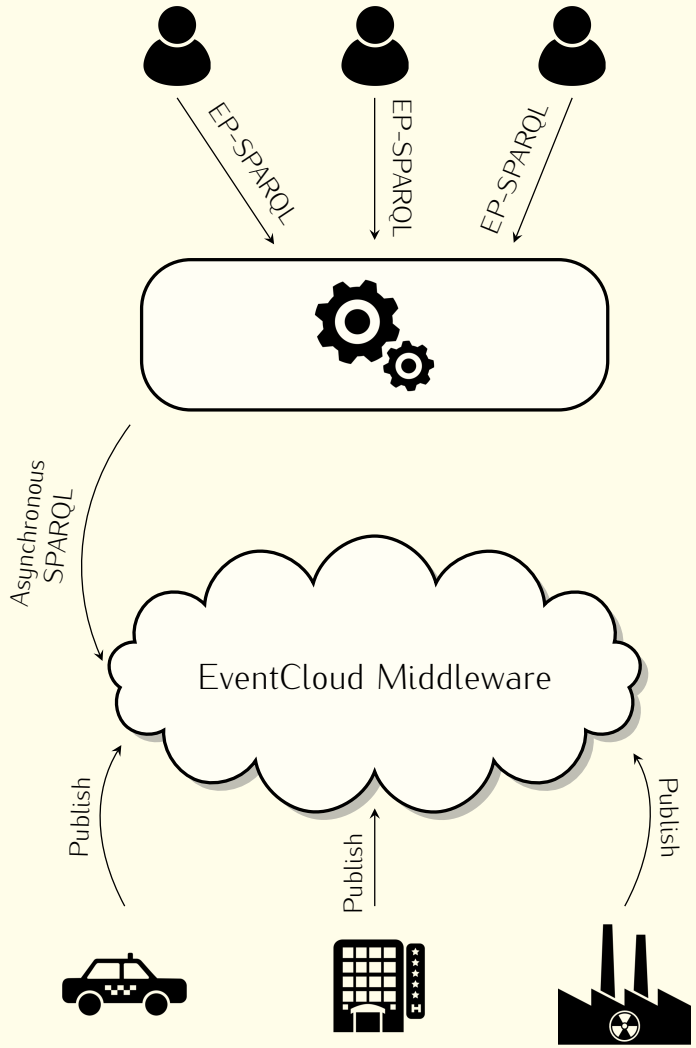
EventCloud

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)

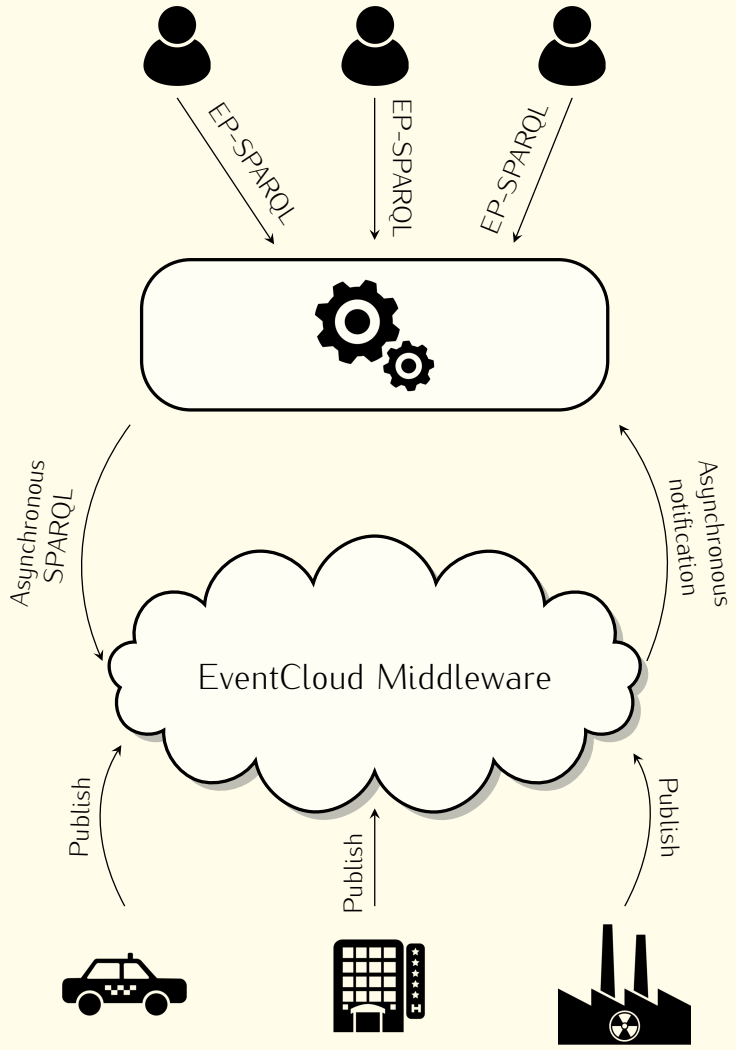
EventCloud

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)

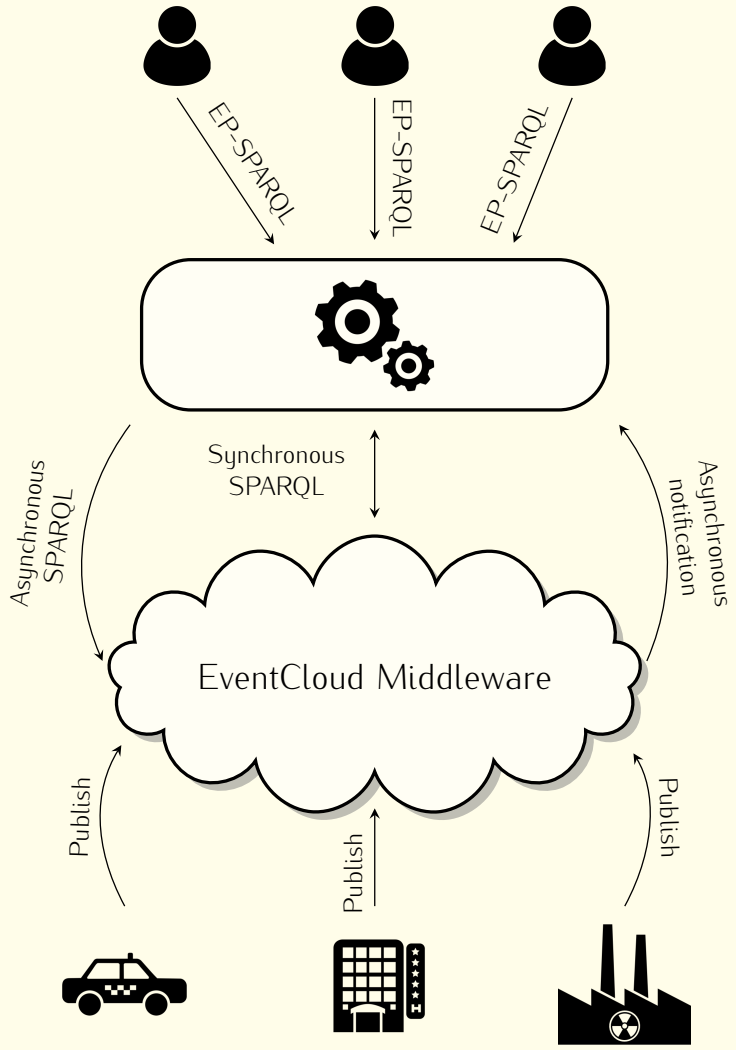
EventCloud

Scalable storage and retrieval of RDF-based data

Event sources

Send data synchronously or asynchronously

Context: PLAY and SocEDA projects



Platform users

Subscribe to detect interesting patterns and react consequently

CEP

Complex Event Processing engine (Etalis/Esper)

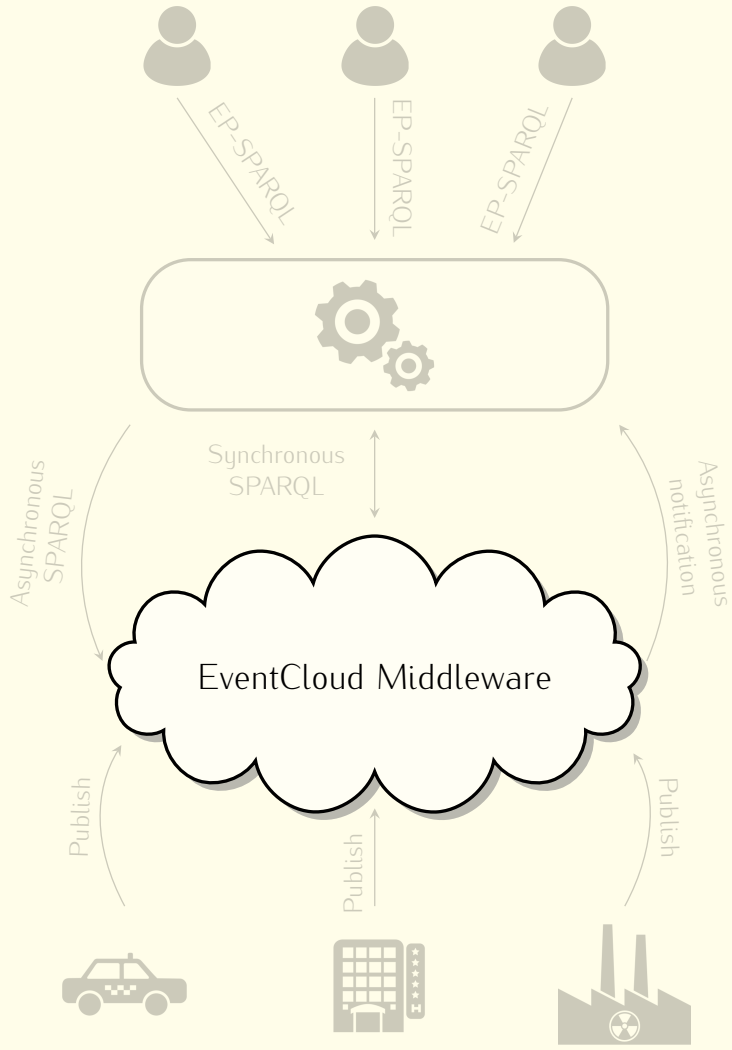
EventCloud

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?*

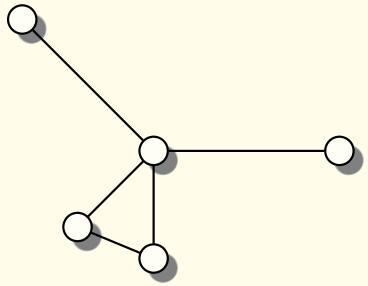
- Challenges
  - Data indexing/retrieval
  - Near real-time filtering
  - Load balancing
  - Scalability

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

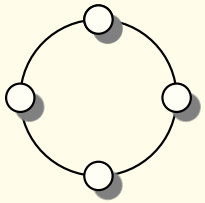
Outline

Peer-to-Peer (P2P) networks

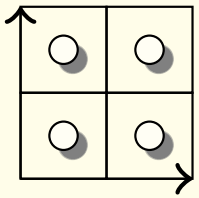
- Unstructured P2P networks
  - links between peers are chosen randomly
  - bad performance for search (flooding)



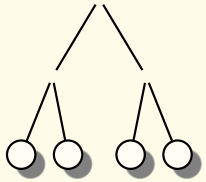
- Structured P2P networks
  - various topologies (ring, hypertorus, tree)
  - communication overhead for *join*/*leave*
  - more efficient for lookup



Chord



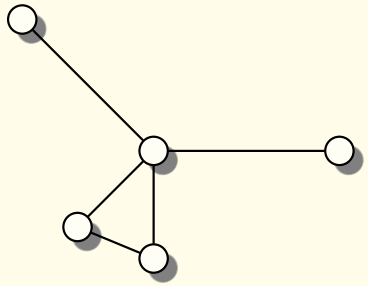
CAN



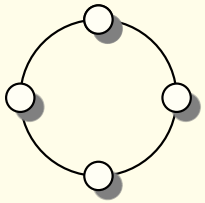
P-grid

Peer-to-Peer (P2P) networks

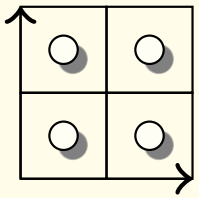
- Unstructured P2P networks
  - links between peers are chosen randomly
  - bad performance for search (flooding)



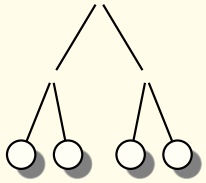
- Structured P2P networks
  - various topologies (ring, hypertorus, tree)
  - communication overhead for *join*/*leave*
  - more efficient for lookup



Chord



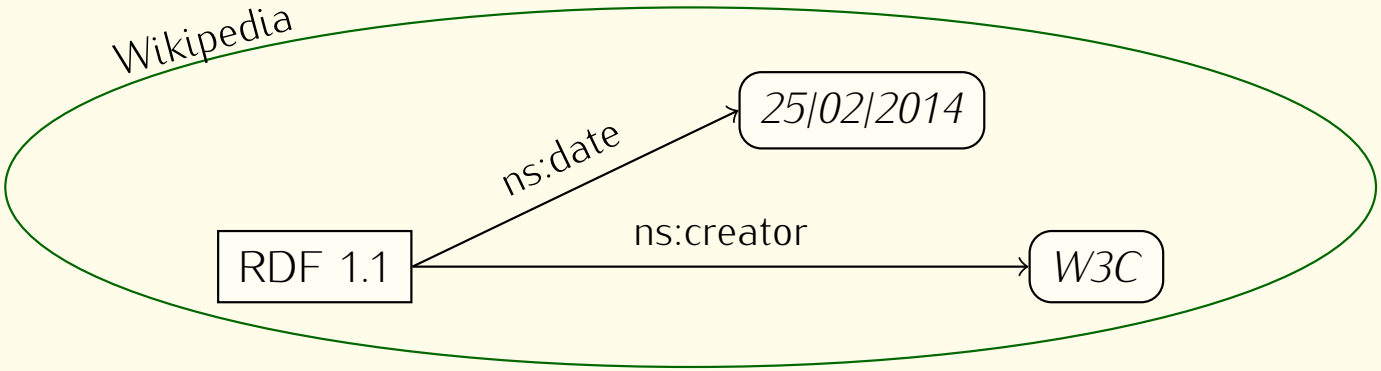
CAN



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
  - $(?g, ?s, \text{ns:creator}, \text{W3C})$
- **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
  - $(?g, s, p, ?o)$ FILTER $v_1 \leq ?o < v_2$

Related works

P2P

- RDFPeers
- RDFCube
- Liarou *et al.*

NoSQL

- CumulusRDF
- H_2RDF
- AllegroGraph
- BigData

- Limited SPARQL support
- Monolithic software architecture
- Hashing for indexing and storing RDF data

Related works

P2P	NoSQL
RDFPeers	CumulusRDF
RDFCube	H_2RDF
Liarou *et al.*	AllegroGraph
	BigData
Filali[1]	
Bongiovanni[2]	

- Limited SPARQL support
- Monolithic software architecture
- Hashing for indexing and storing RDF data

[1] Filali, "Improving resource discovery in P2P systems".

[2] Bongiovanni, "Design, formalization and implementation of overlay networks; application to RDF data storage".

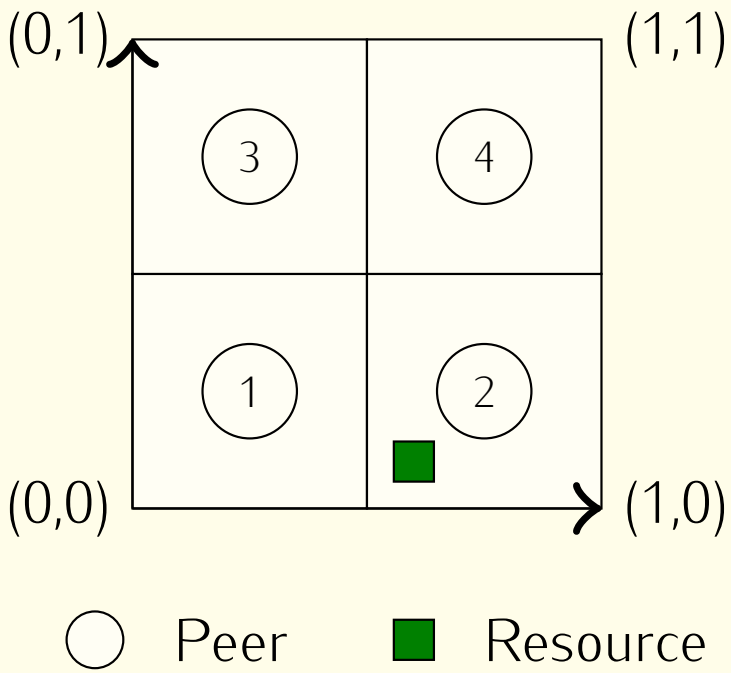
Hashing vs Lexicographic based indexing

- 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

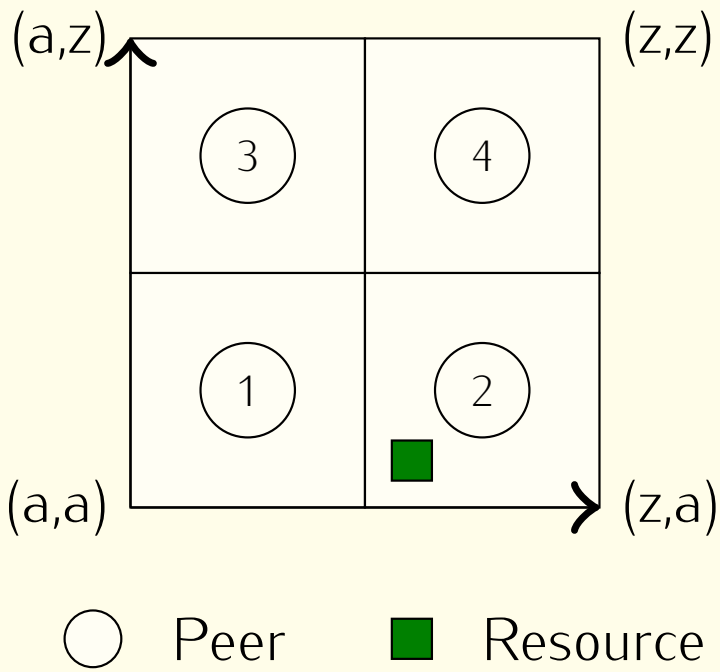
Our P2P infrastructure

- 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 $q = (g, s, p, o)$ 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[3]

[3] 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[3]

Unicast routing is used to index a quadruple

[3] 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[3]

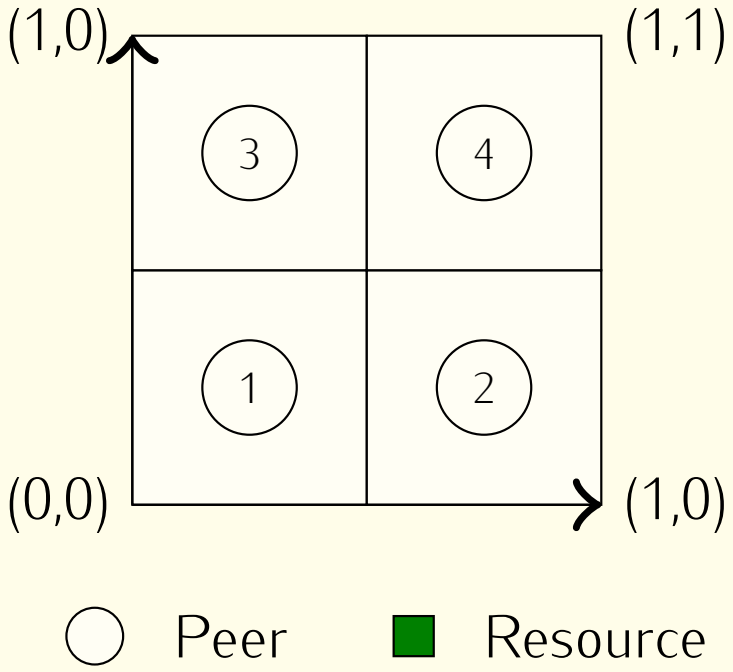
Unicast routing is used to index a quadruple

Multicast routing is the substrat for retrieving information

[3] Bongiovanni and Henrio, "A Mechanized Model for CAN Protocols".

RDF data indexing

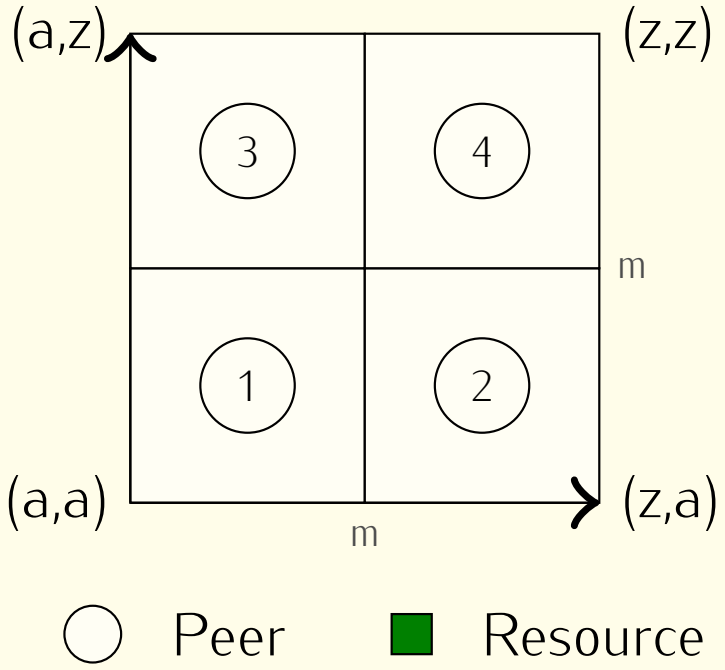
Data: Quadruple q
begin
  if $q \notin$ *peer's zone* **then**
    Forward the request to the closest neighbor
  else
    Store q;
    Send back response;
  end
end



No hashing but lexicographic order

RDF data indexing

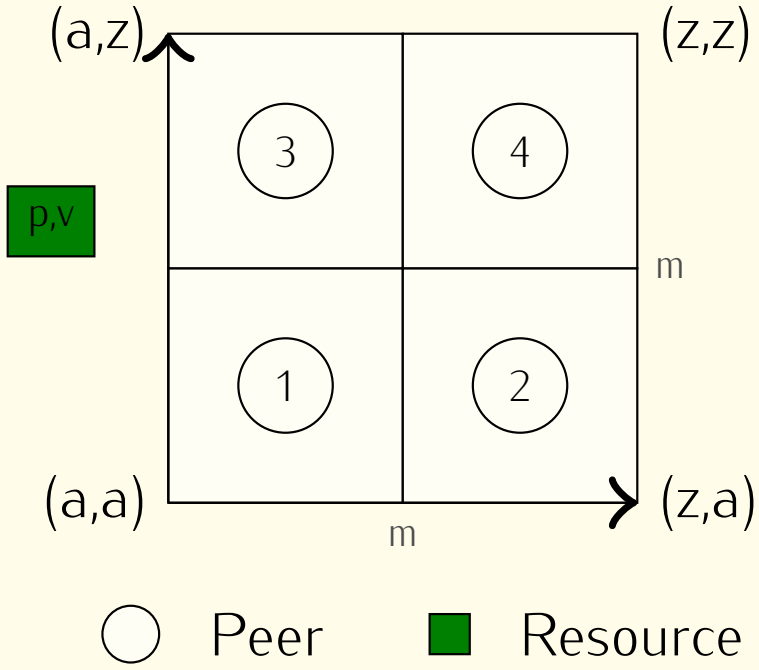
Data: Quadruple q
begin
  if $q \notin$ *peer's zone* **then**
    Forward the request to the closest neighbor
  else
    Store q;
    Send back response;
  end
end



No hashing but lexicographic order

RDF data indexing

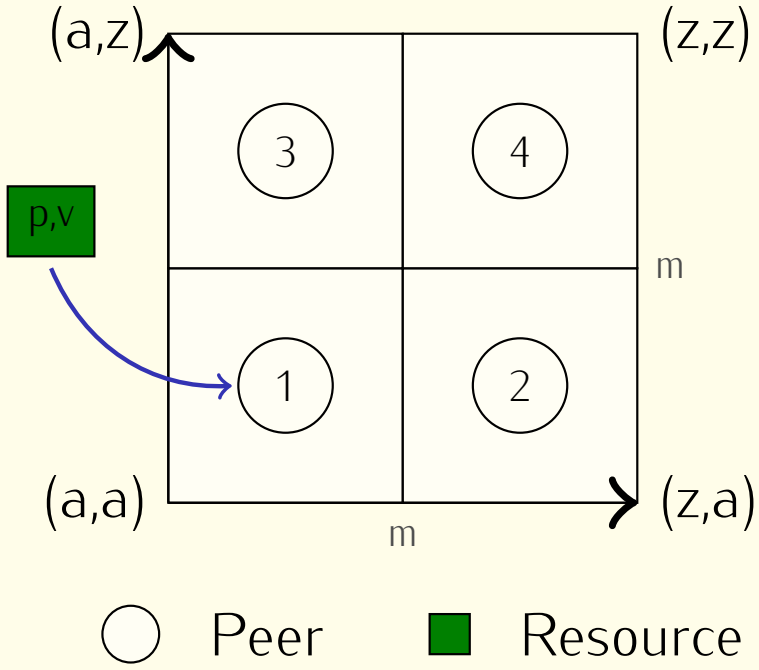
Data: Quadruple q
begin
  if $q \notin$ *peer's zone* **then**
    Forward the request to the closest neighbor
  else
    Store q;
    Send back response;
  end
end



No hashing but lexicographic order

RDF data indexing

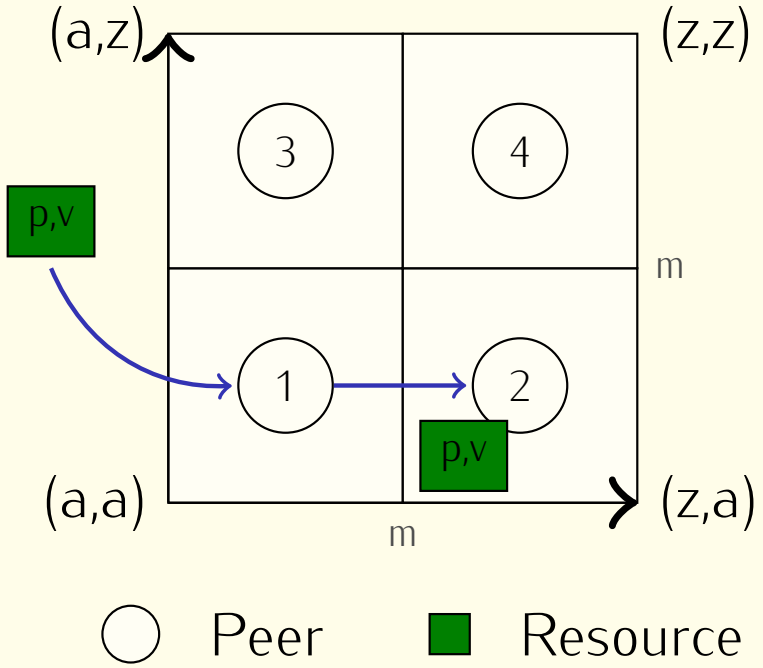
Data: Quadruple q
begin
  if $q \notin$ *peer's zone* **then**
    Forward the request to the closest neighbor
  else
    Store q;
    Send back response;
  end
end



No hashing but lexicographic order

RDF data indexing

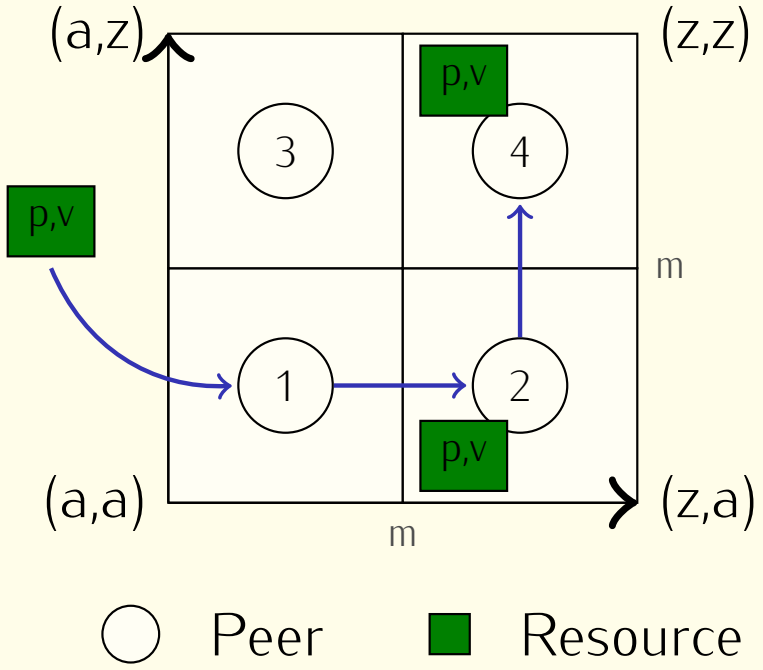
Data: Quadruple q
begin
  if $q \notin$ *peer's zone* **then**
    Forward the request to the closest neighbor
  else
    Store q;
    Send back response;
  end
end



No hashing but lexicographic order

RDF data indexing

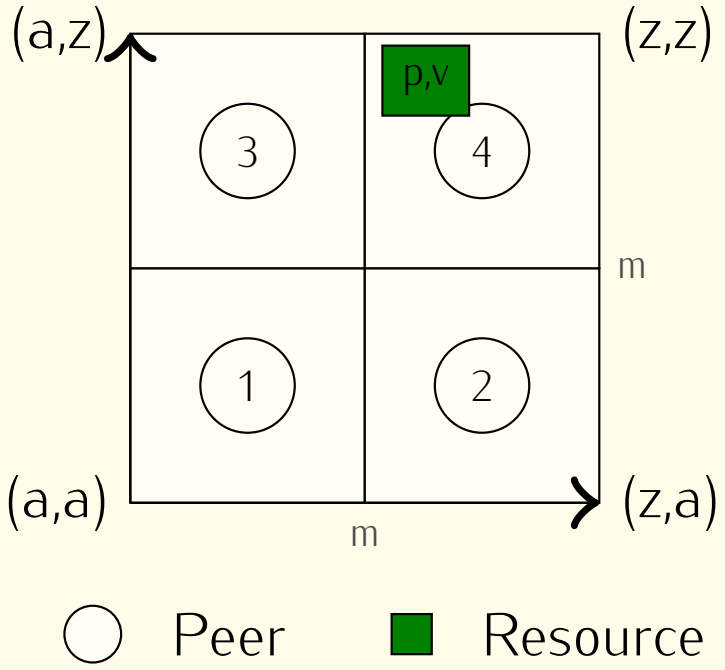
Data: Quadruple q
begin
  if $q \notin$ *peer's zone* **then**
    Forward the request to the closest neighbor
  else
    Store q;
    Send back response;
  end
end



No hashing but lexicographic order

RDF data indexing

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



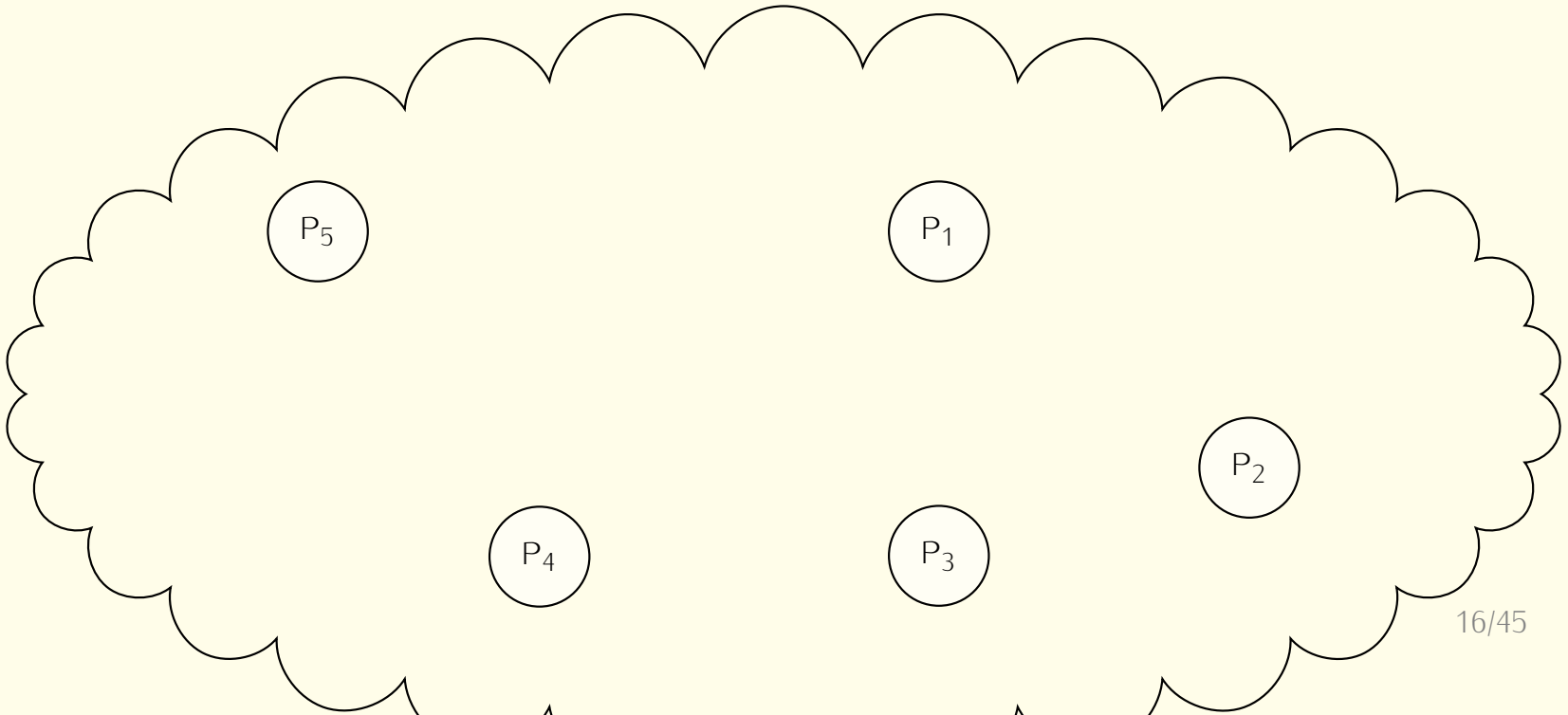
No hashing but lexicographic order

RDF data 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

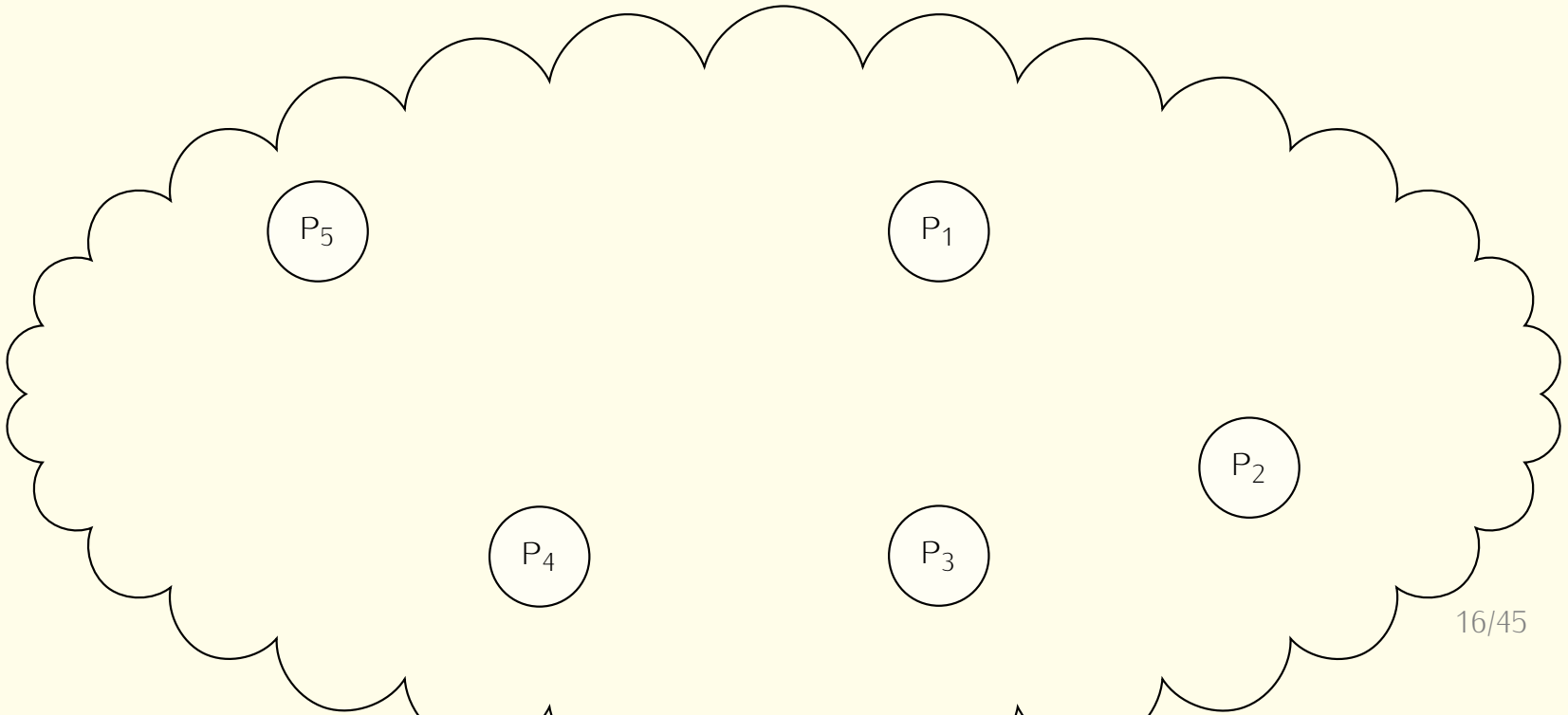


RDF data retrieval

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)
}
```

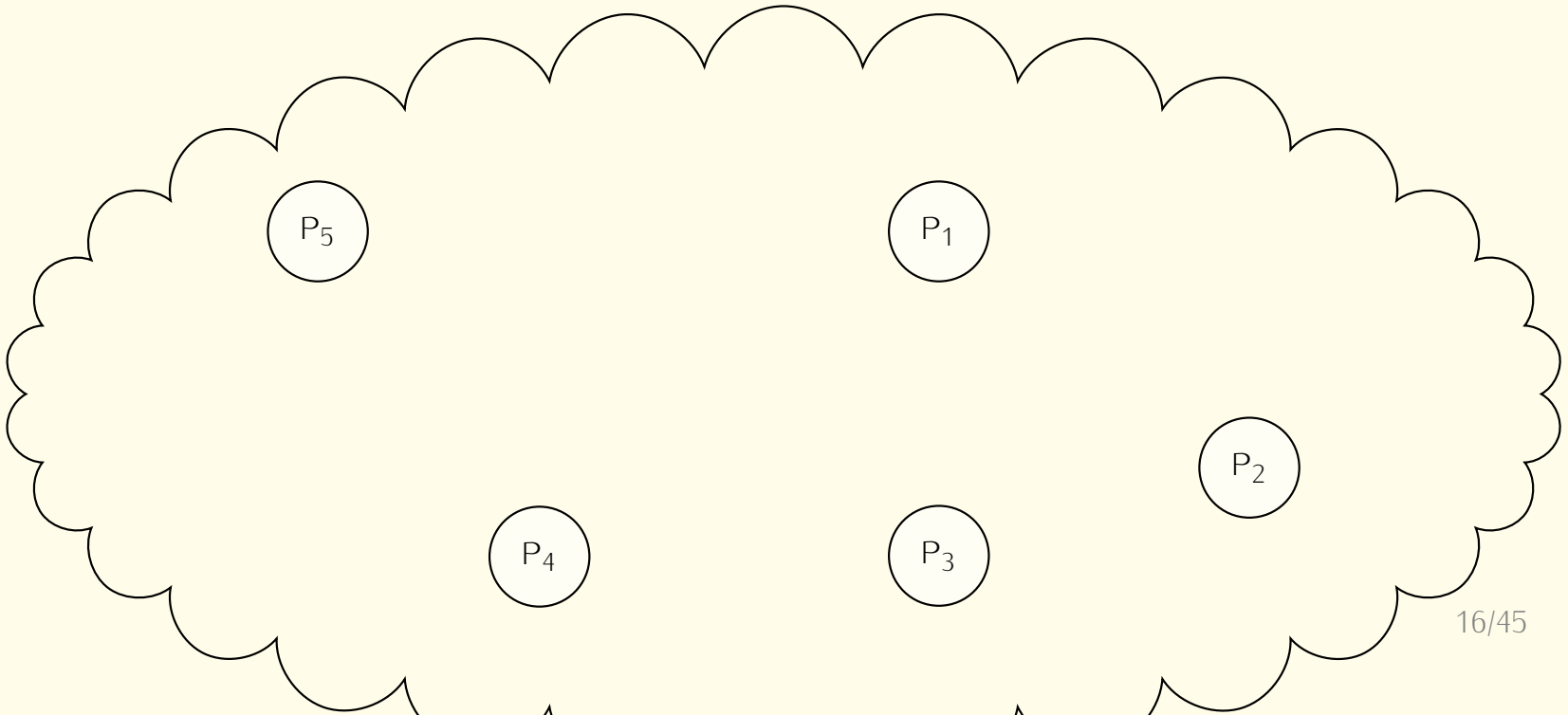
SPARQL Decomposer

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)
}
```

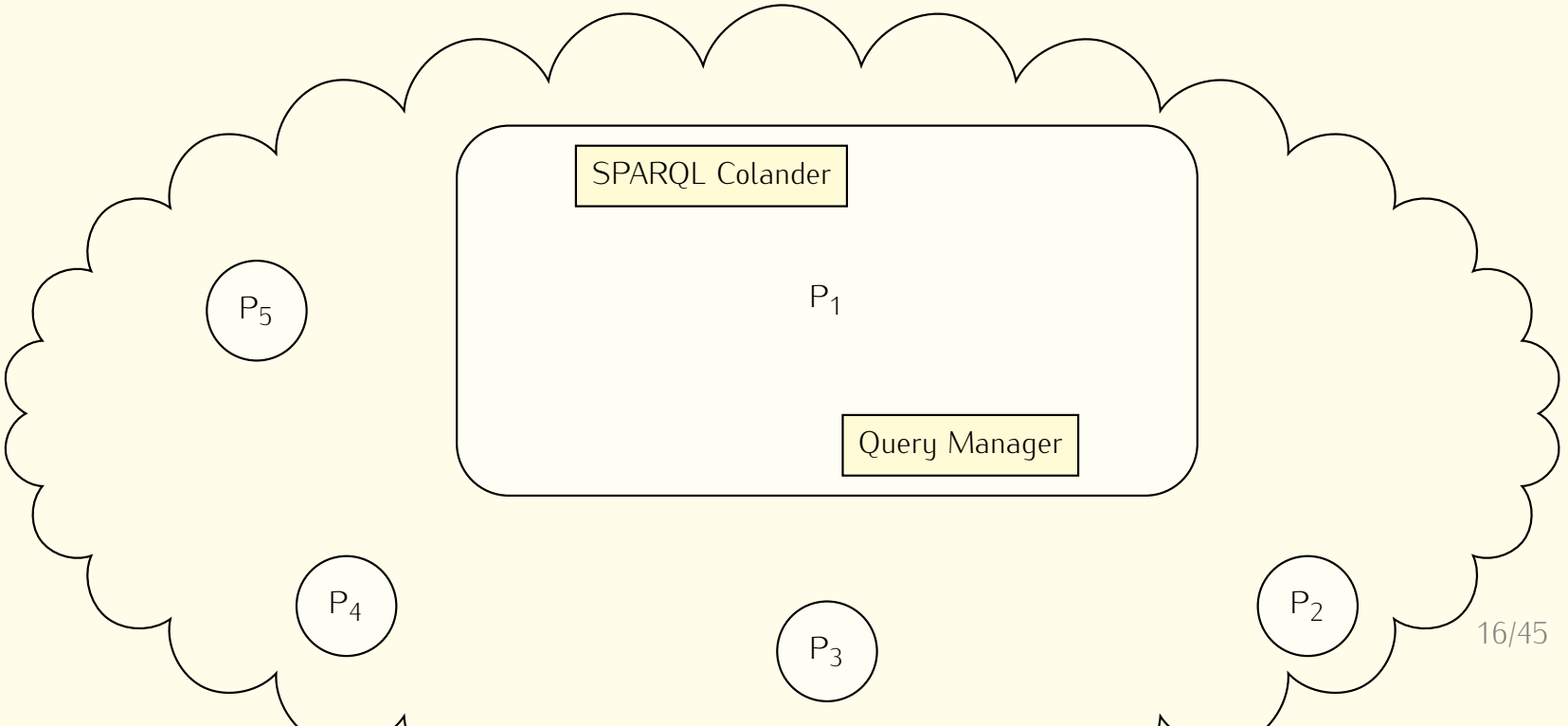
SPARQL Decomposer

q_1

?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

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

P_5

P_4

P_3

P_2

q_1

q_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_2

P_3

P_4

P_5

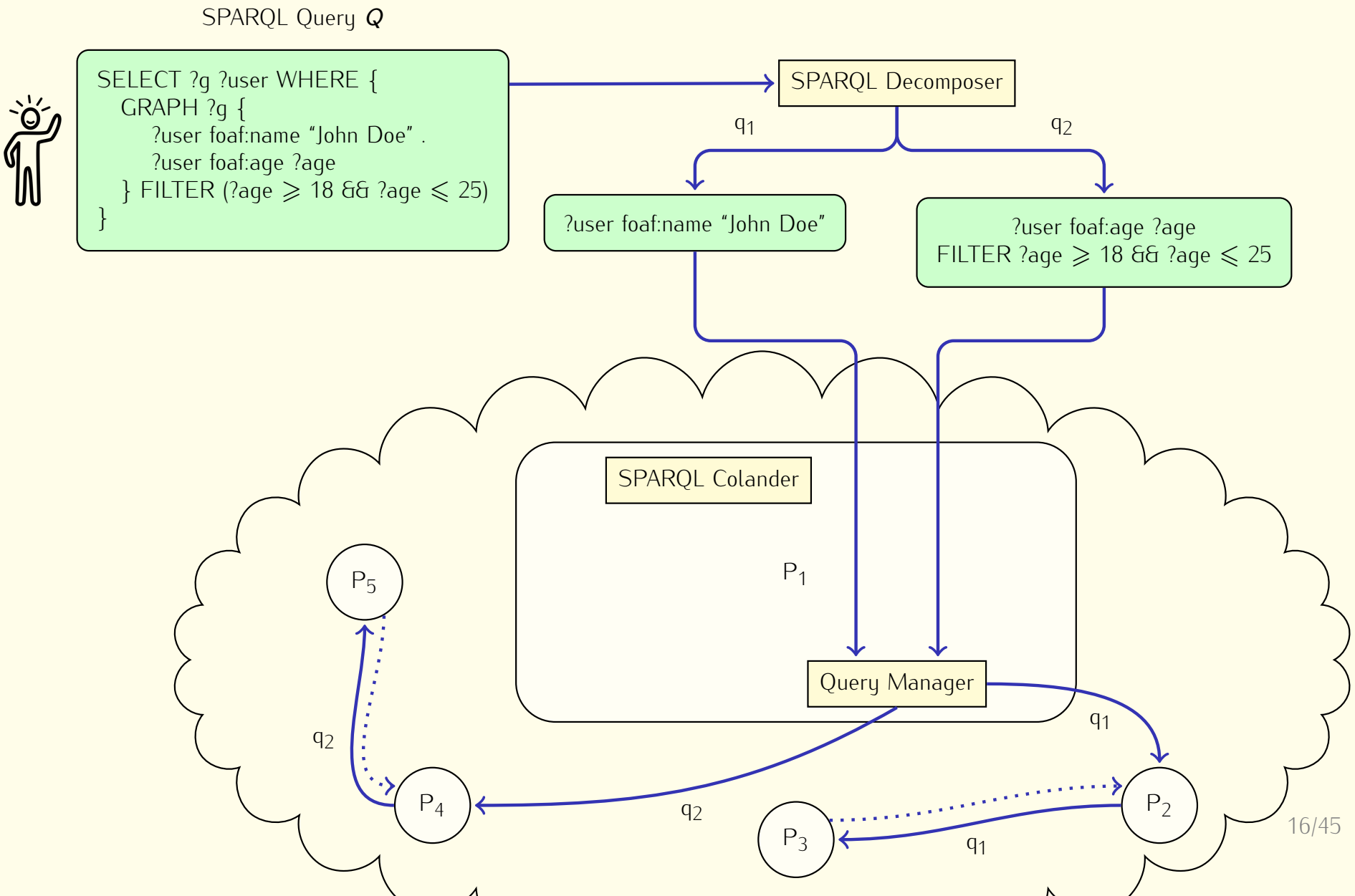
q_1

q_2

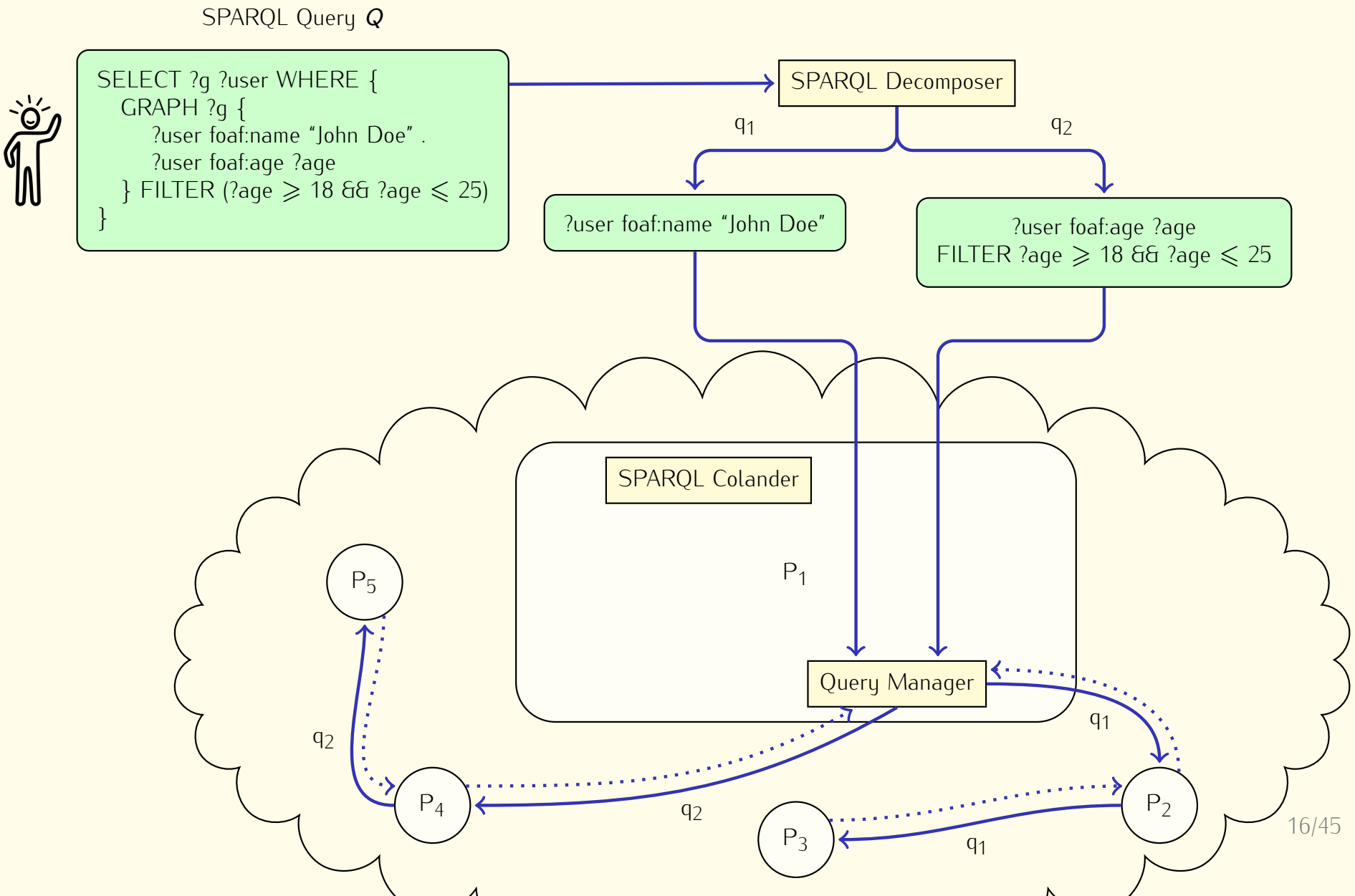
q_1

q_2

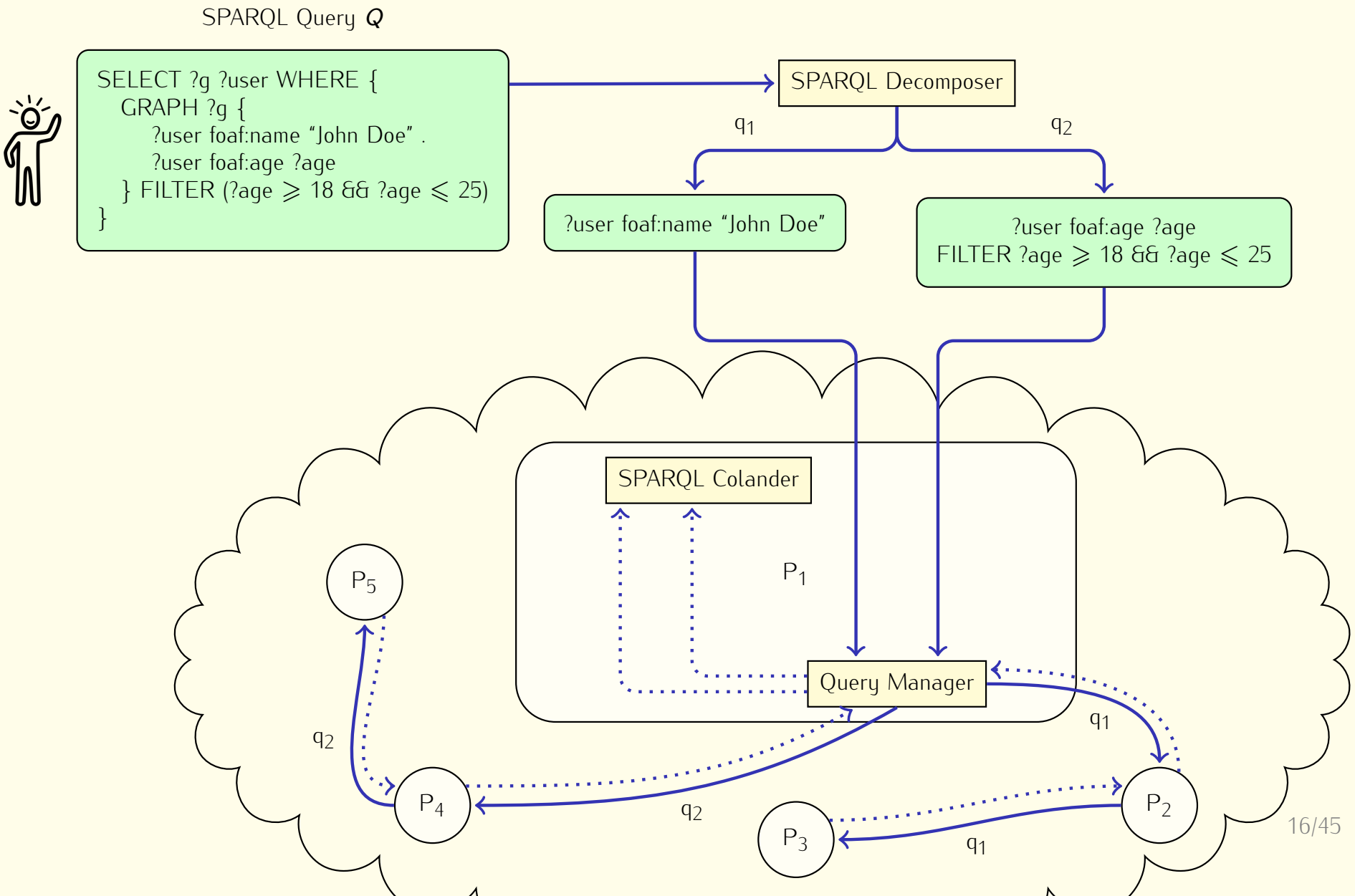
RDF data retrieval



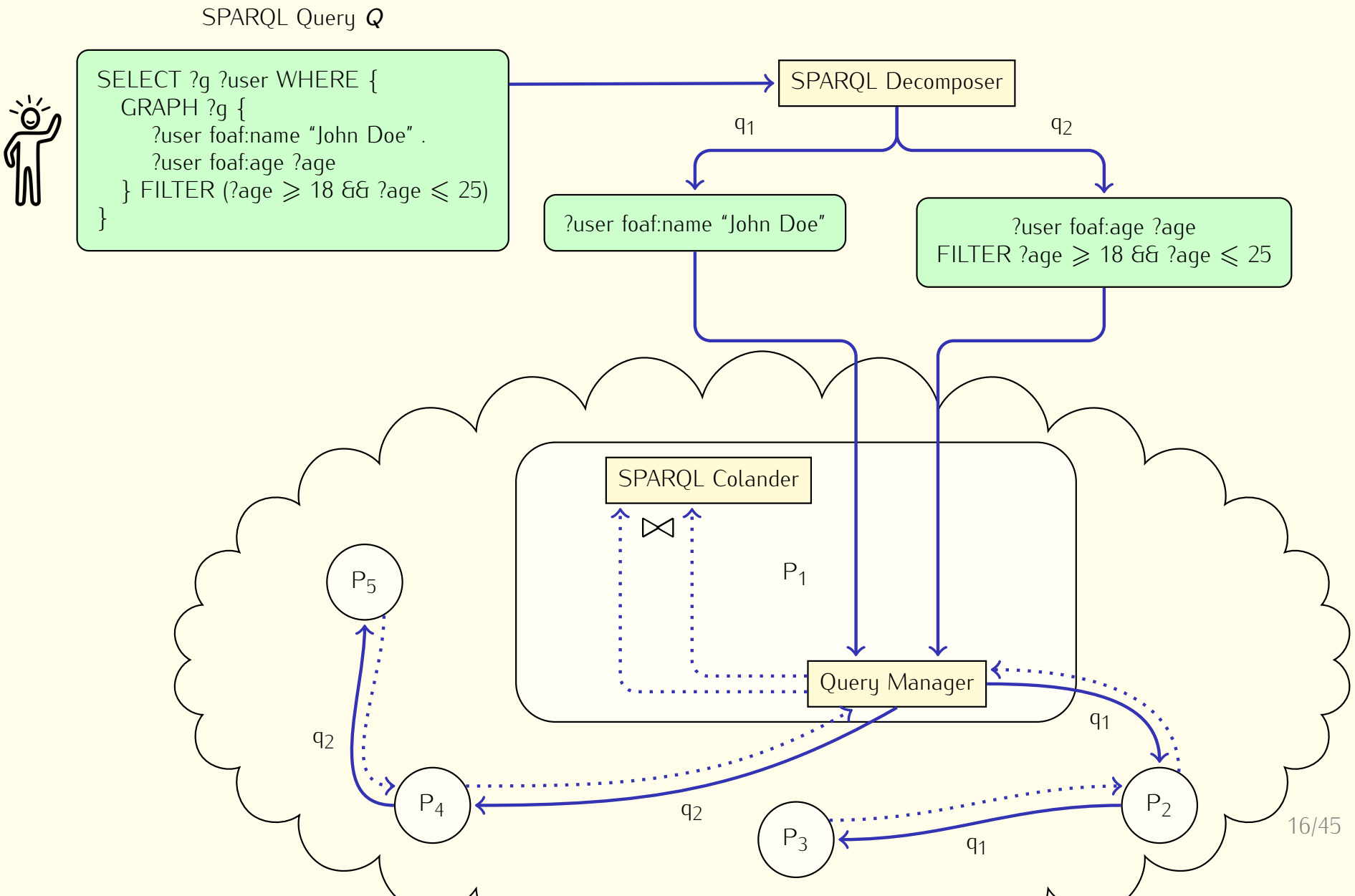
RDF data retrieval



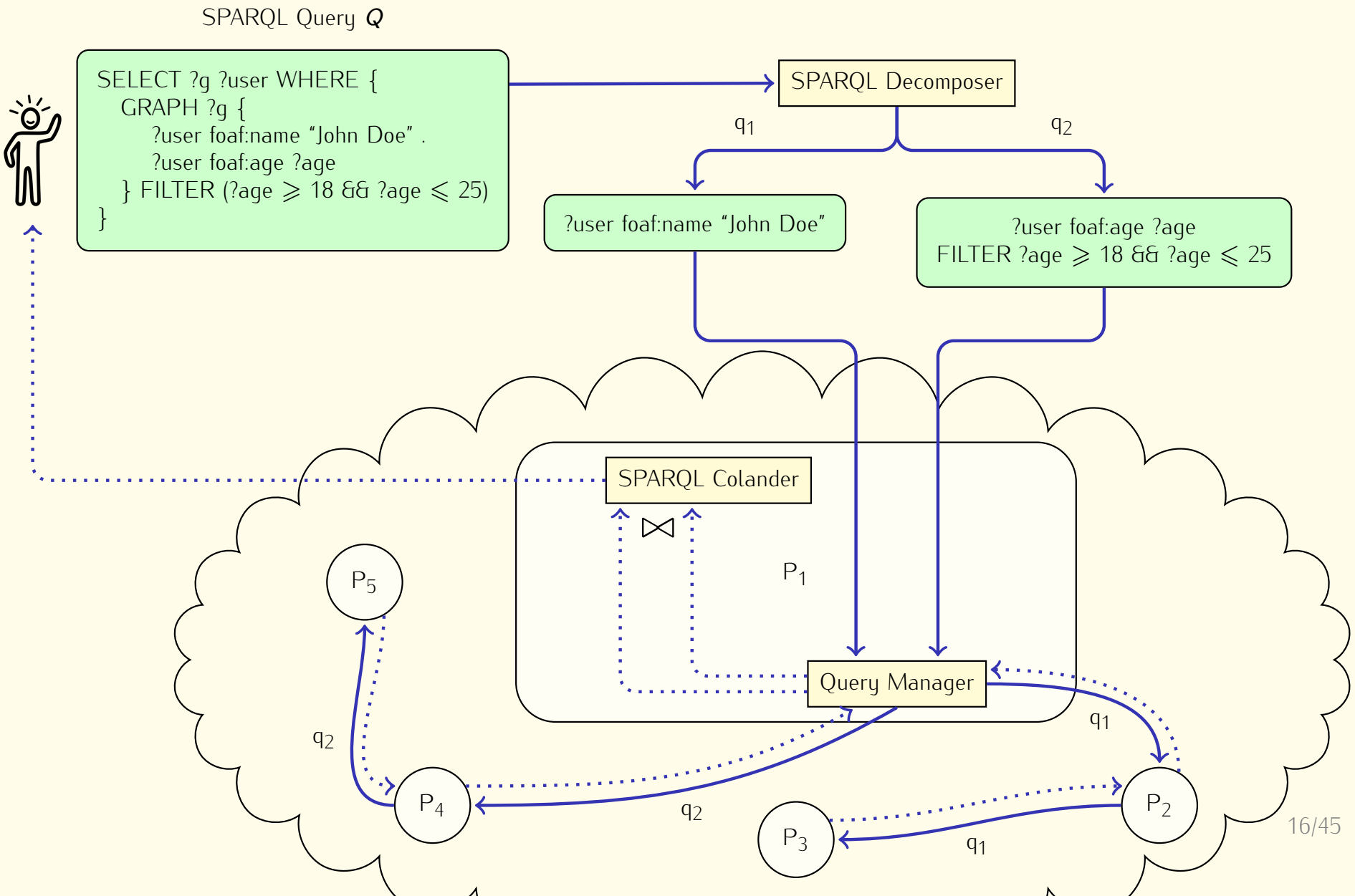
RDF data retrieval



RDF data retrieval



RDF data retrieval

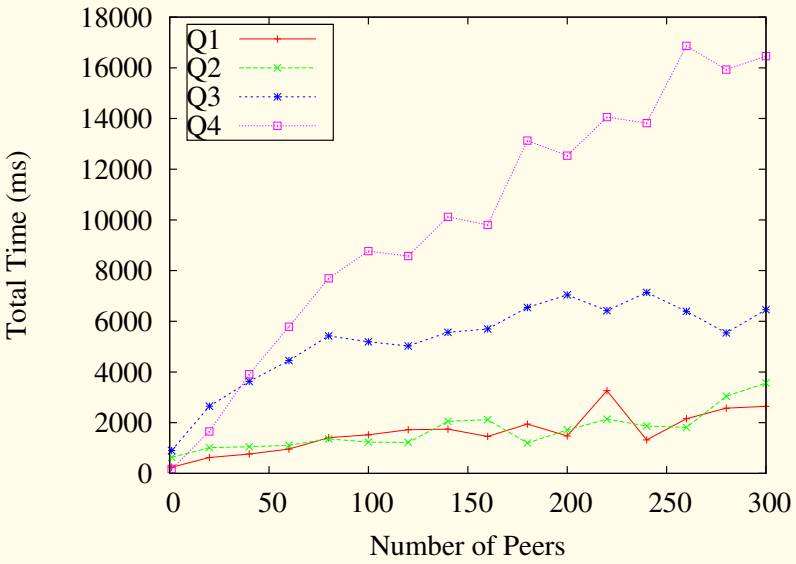


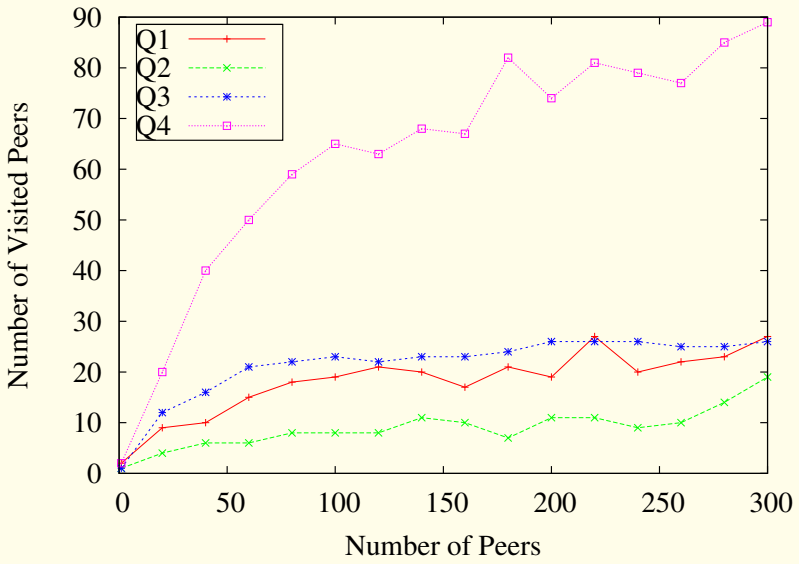
Experimental setup

- 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



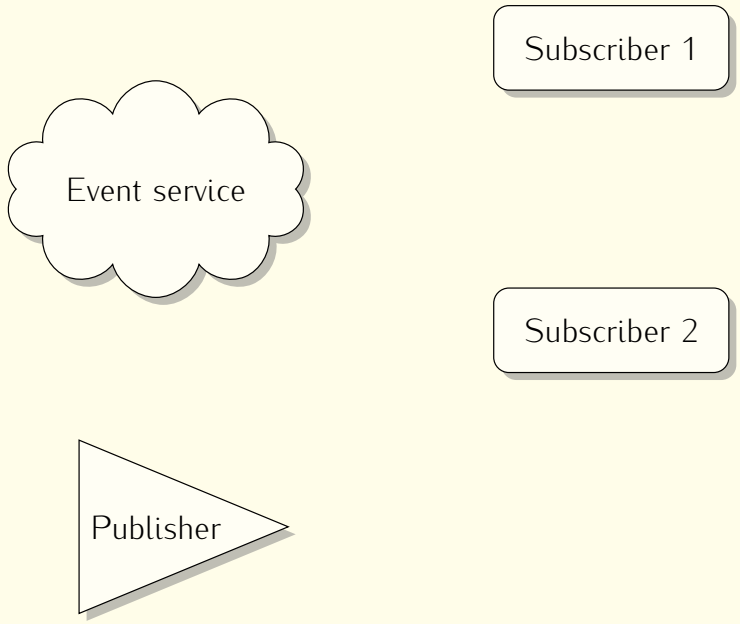


Performance results strongly depend on the type of queries

Outline

Context

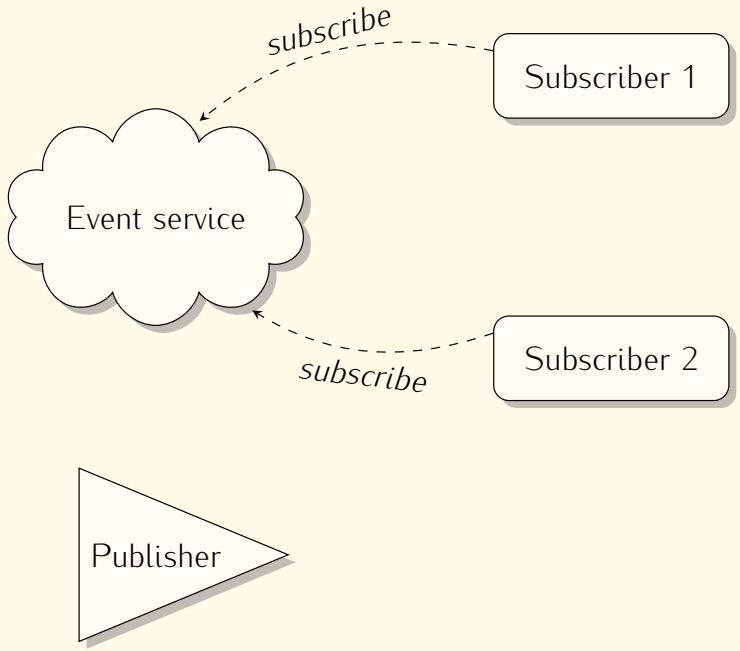
- Event Driven Architecture
  - Publish/Subscribe *asynchronous* communication style
    - Subscribers
    - Publishers
    - Event notification service
  - Entities are decoupled in *space*, *time* and *synchronization*



- 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

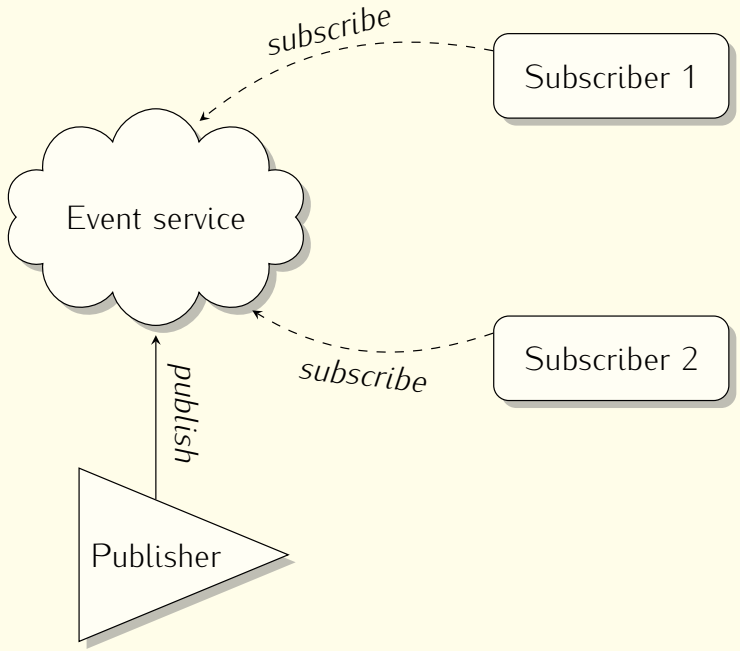
- Event Driven Architecture
  - Publish/Subscribe *asynchronous* communication style
    - Subscribers
    - Publishers
    - Event notification service
  - Entities are decoupled in *space*, *time* and *synchronization*



- 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

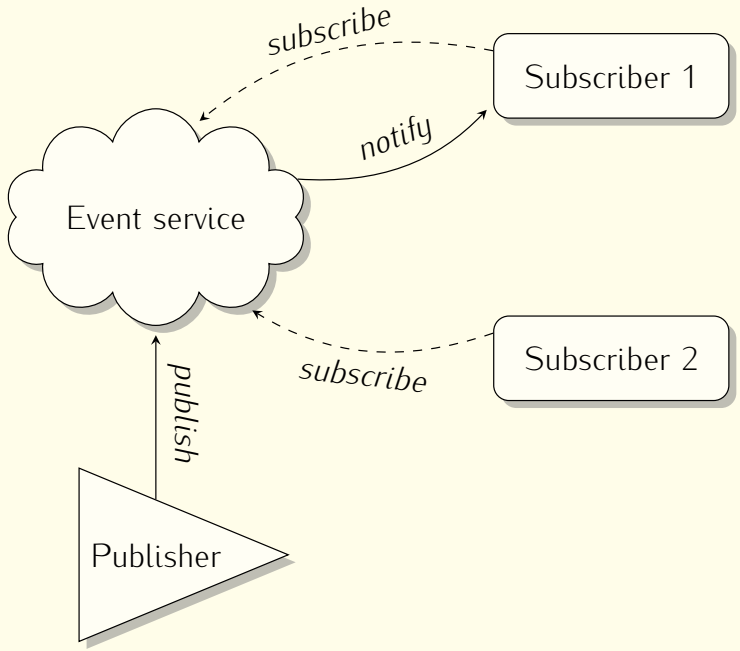
- Event Driven Architecture
  - Publish/Subscribe *asynchronous* communication style
    - Subscribers
    - Publishers
    - Event notification service
  - Entities are decoupled in *space*, *time* and *synchronization*



- 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[4]
- Pubsubhubbub[5]

Content-based

- Siena[6]
- Hermes[7]
- BlueDove[8]

Do not focus on the specific characteristics of RDF

Attribute/value pairs vs Tuples

Constraints defined at startup (event size, type, domain)

---

4 Tibco, "TIB/Rendezvous White Paper".

5 Fitzpatrick, Slatkin, and Atkins, *PubSubHubbub protocol*.

6 Carzaniga, Rosenblum, and Wolf, "Design and evaluation of a wide-area event notification service".

7 Pietzuch and Bacon, "Hermes: A distributed event-based middleware architecture".

8 Li, Ye, Kim, Chen, and Lei, "A scalable and elastic publish/subscribe service".

Related works: RDF pub/sub systems (1)

- RDFPeers[9]
  - 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.*[10]
  - Information sharing platform using multicast trees
  - The peers participating to the propagation are responsible for removing duplicate results within the limit of their buffer

[9] Cai, Frank, Yan, and MacGregor, "A subscribable peer-to-peer RDF repository for distributed metadata management".

[10] Ranger and Cloutier, "Scalable peer-to-peer RDF query algorithm".

Related works: RDF pub/sub systems (2)

- CSBV[11]
  - Distributed Hash Table (DHT) agnostic
  - Each publication is indexed seven times
- Shvartzshnaider *et al.*[12]
  - 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

[11] Liarou, Idreos, and Koubarakis, "Continuous RDF query processing over DHTs".

[12] Shvartzshnaider, Ott, and Levy, "Publish/subscribe on top of DHT using RETE algorithm".

Proposed data and subscription model

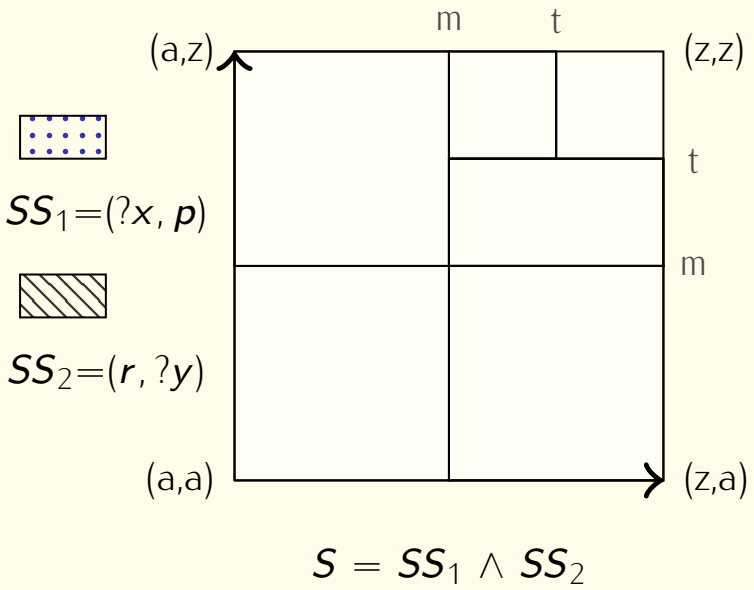
- Data Model
  - RDF quadruple $q = (g, s, p, o) \mid g, s, p, o \in RDFTerm$
  - Compound Event
    - $CE = (q_1, \ldots, q_i, \ldots, q_n) \mid q_i = (g, s_i, p_i, o_i)$
    - Ability to put more things in an event
- Subscription Model
  - Filter language based on a subset of SPARQL

```
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

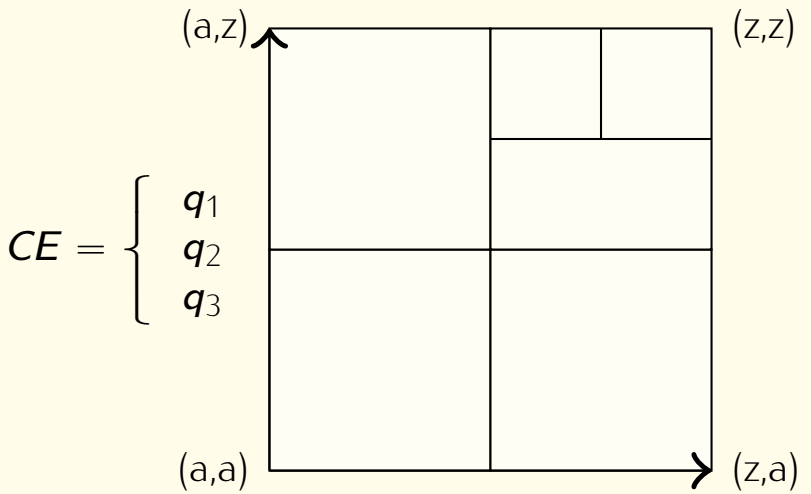
Indexing subscriptions and events

- Indexing a subscription



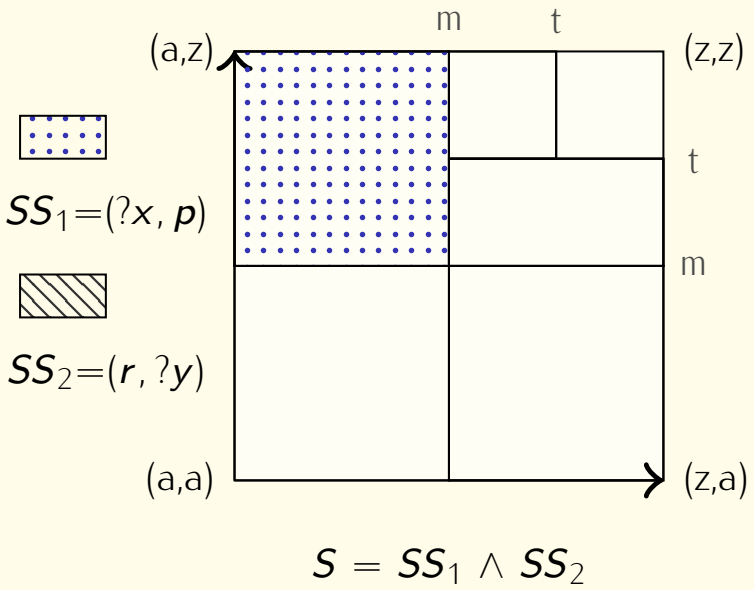
$$S = SS_1 \wedge SS_2$$

- Indexing a compound event



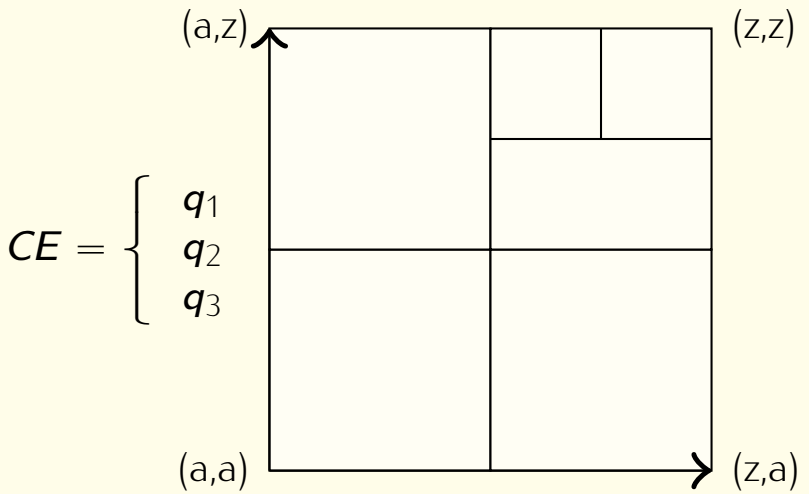
Indexing subscriptions and events

- Indexing a subscription



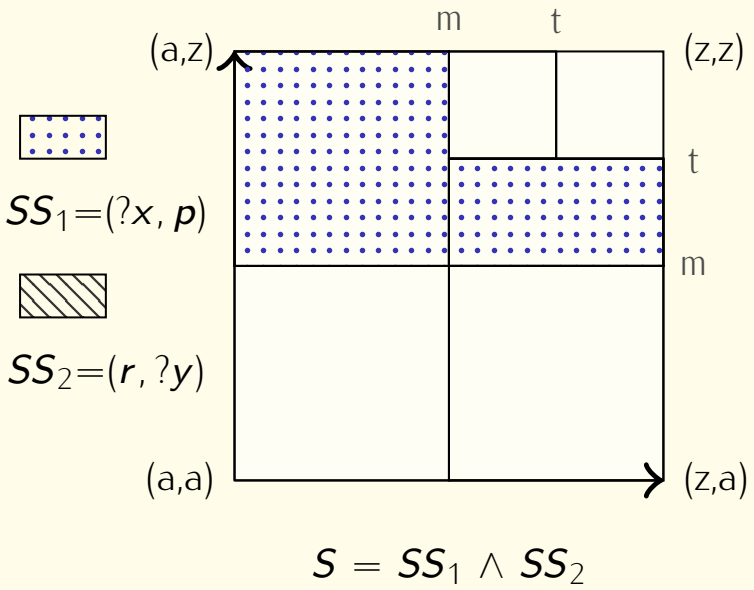
$$S = SS_1 \wedge SS_2$$

- Indexing a compound event



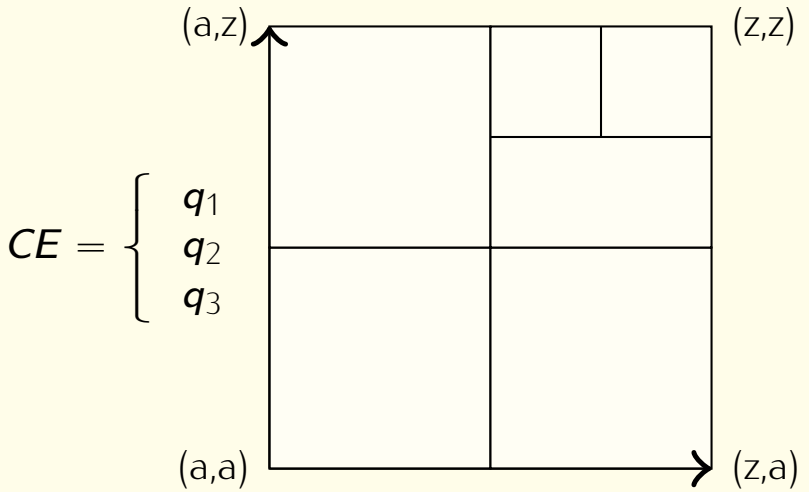
Indexing subscriptions and events

- Indexing a subscription



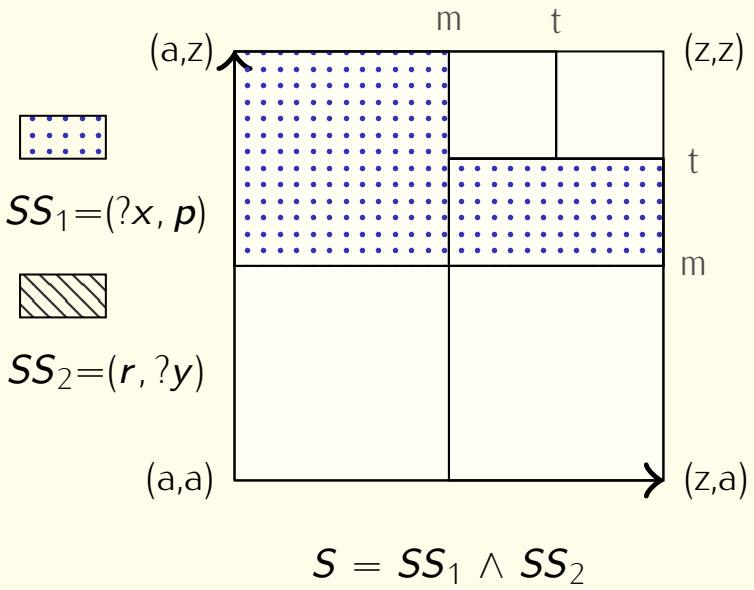
$$S = SS_1 \wedge SS_2$$

- Indexing a compound event



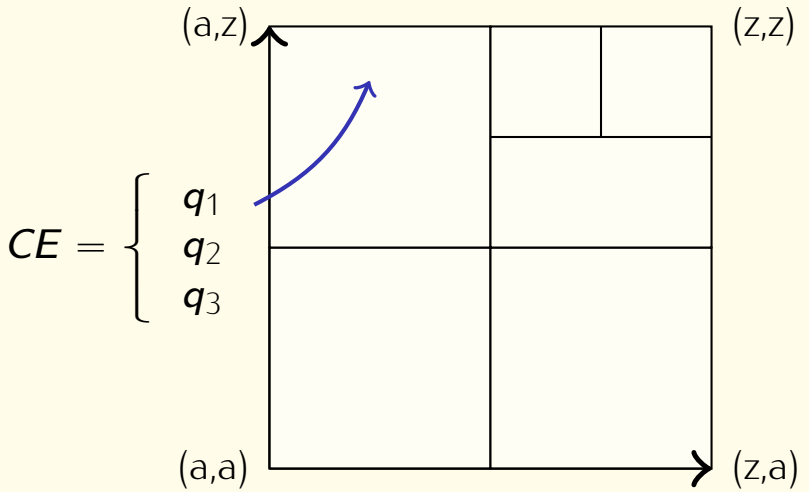
Indexing subscriptions and events

- Indexing a subscription



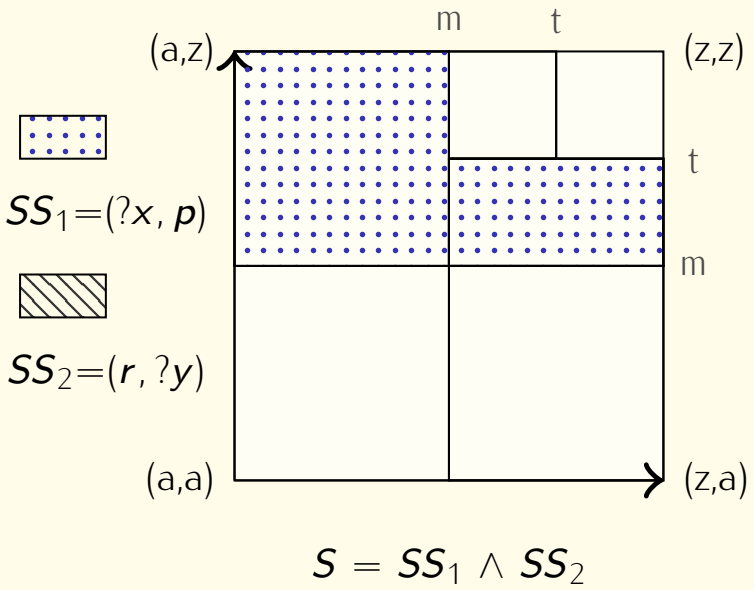
$$S = SS_1 \wedge SS_2$$

- Indexing a compound event



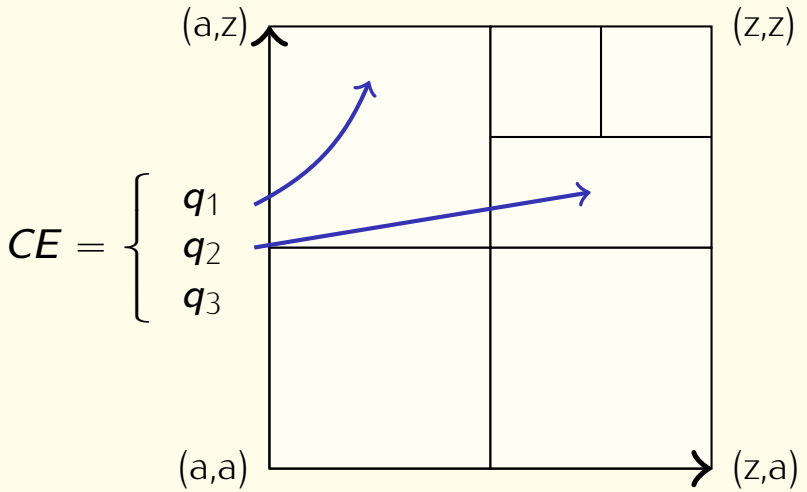
Indexing subscriptions and events

- Indexing a subscription



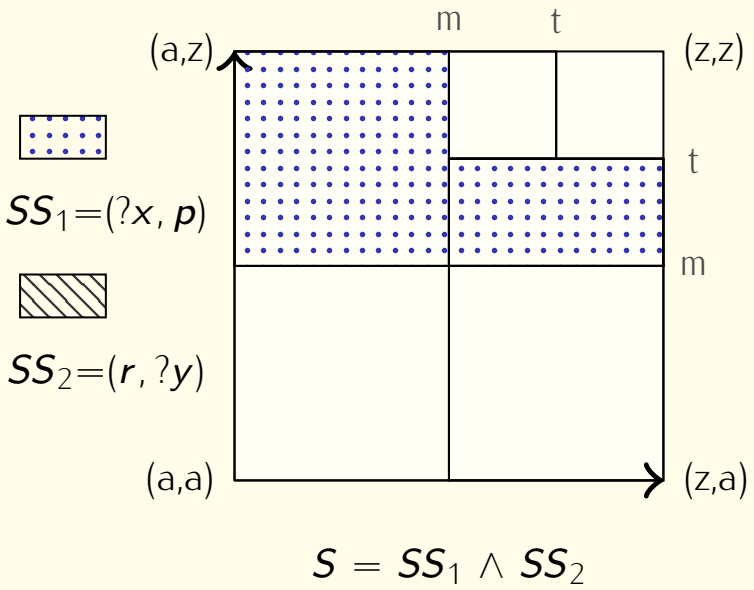
$S = SS_1 \wedge SS_2$

- Indexing a compound event



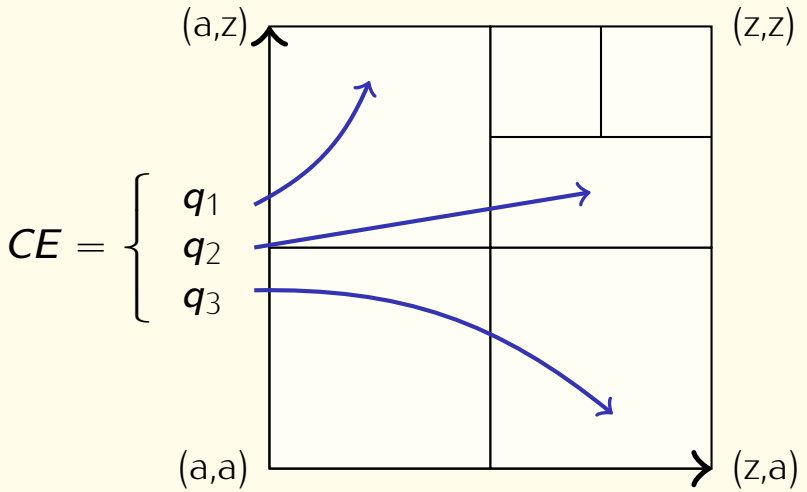
Indexing subscriptions and events

- Indexing a subscription



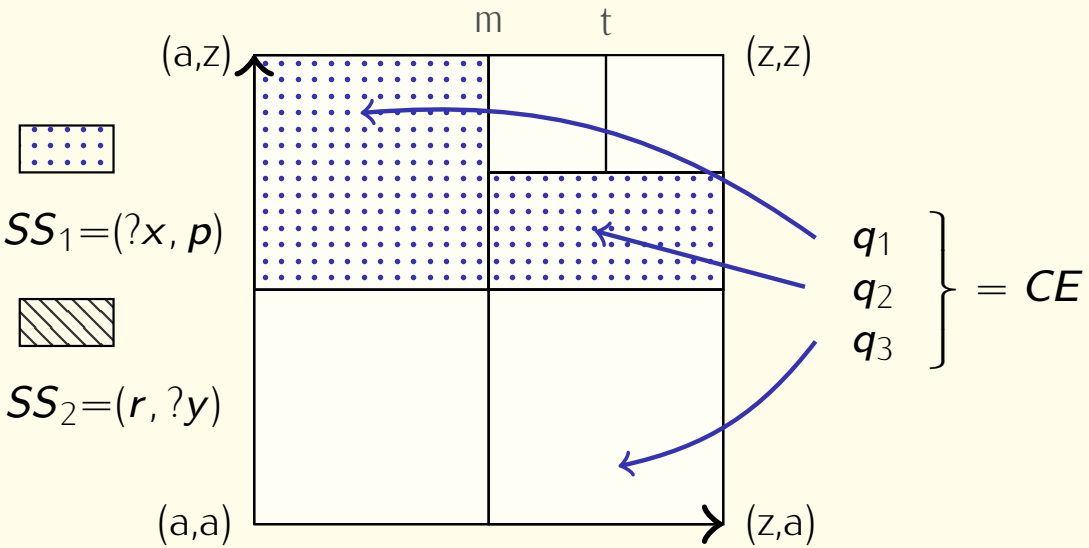
$$S = SS_1 \wedge SS_2$$

- Indexing a compound event



Indexing subscriptions and events

- Indexing a subscription
- Indexing a compound event



$$S = SS_1 \wedge SS_2$$

Perform matching

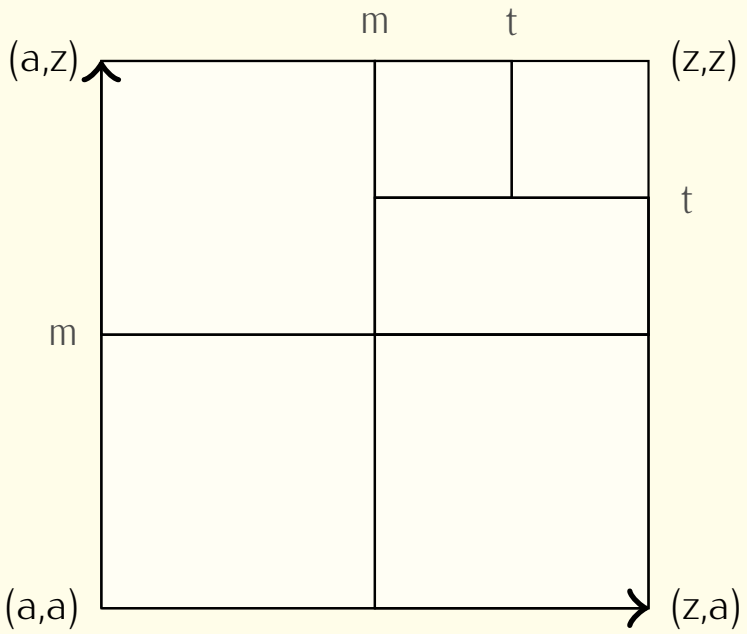
Publish/Subscribe Algorithms

1. Chained Semantic Matching Algorithm (CSMA)
   - Inspired from the original idea of CSBV[13]
   - 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

[13] Liarou, Idreos, and Koubarakis, "Continuous RDF query processing over DHTs".

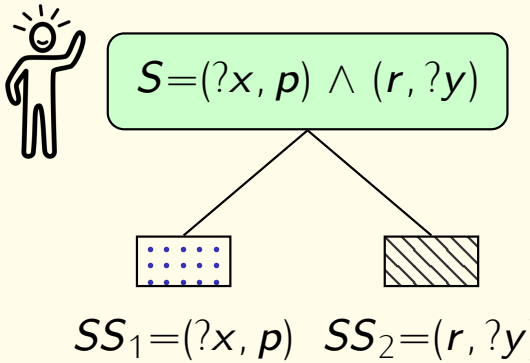
Chained Semantic Matching Algorithm (CSMA)

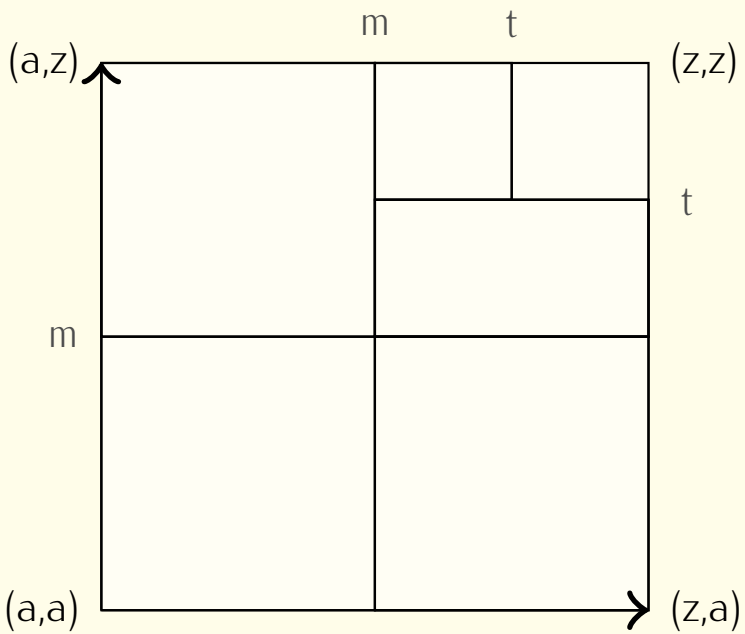
$S = (?x, p) \wedge (r, ?y)$



q_1
q_2
q_3
$= CE$

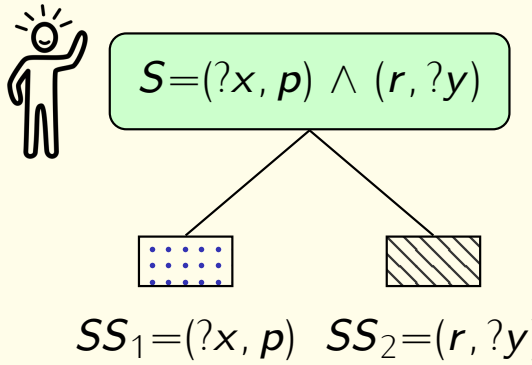
Chained Semantic Matching Algorithm (CSMA)



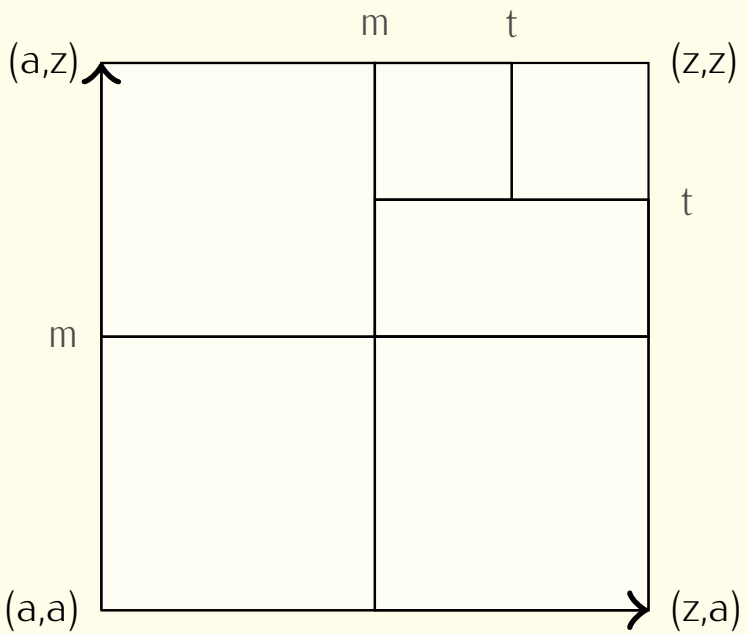


q_1
q_2
q_3
$= CE$

Chained Semantic Matching Algorithm (CSMA)

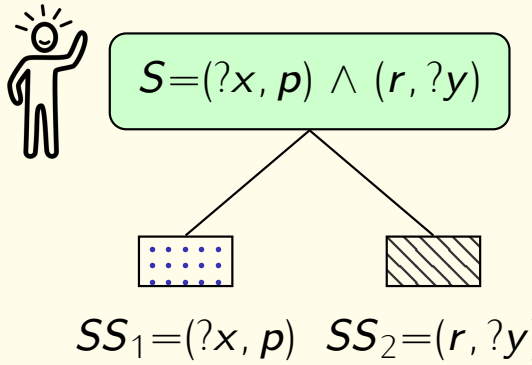


Indexing S according to SS_1

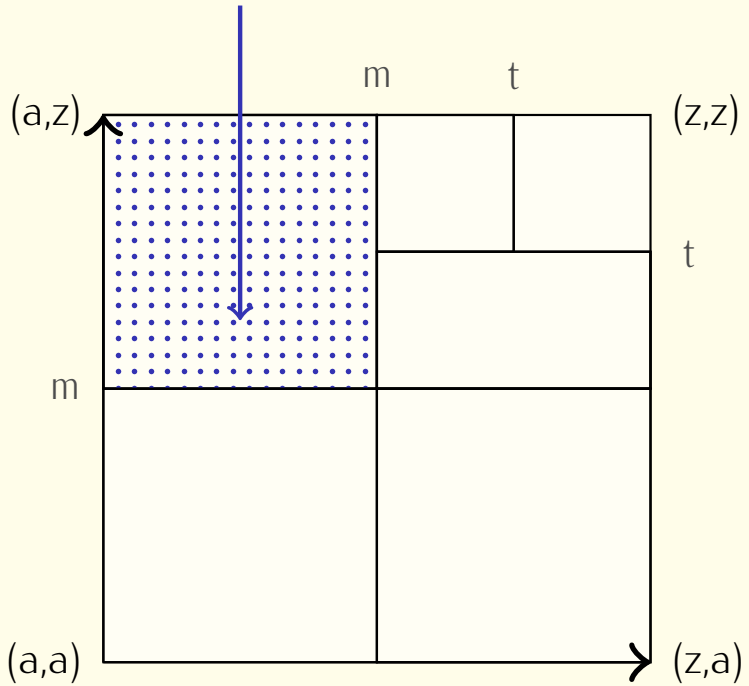


q_1
q_2
q_3
$\} = CE$

Chained Semantic Matching Algorithm (CSMA)

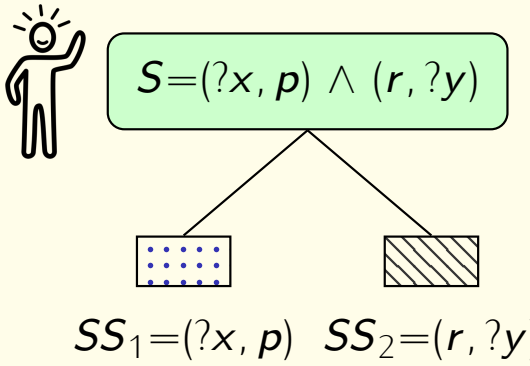


Indexing S according to SS_1

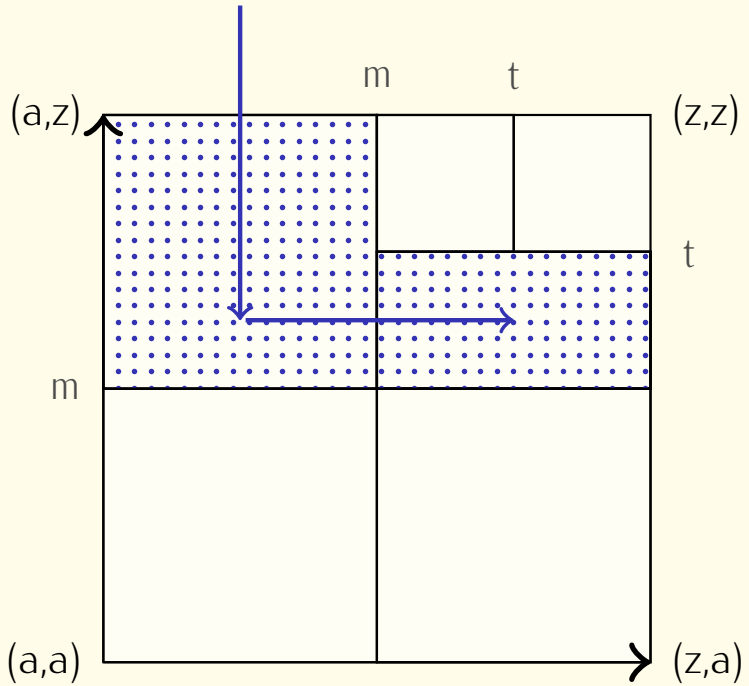


q_1
q_2
q_3
$\} = CE$

Chained Semantic Matching Algorithm (CSMA)

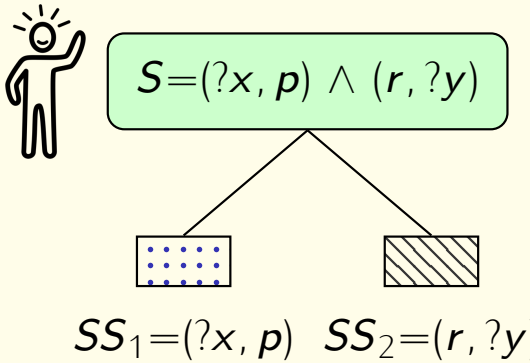


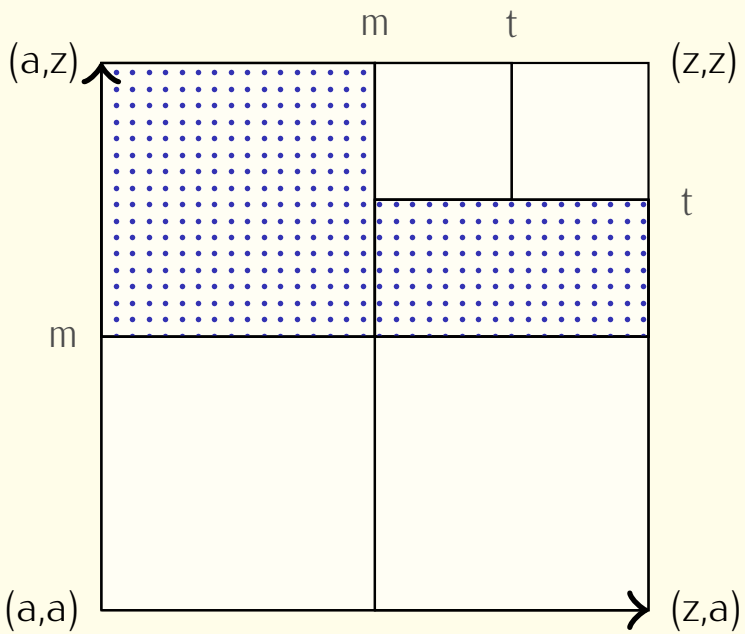
Indexing S according to SS_1



q_1
q_2
q_3
$\}= CE$

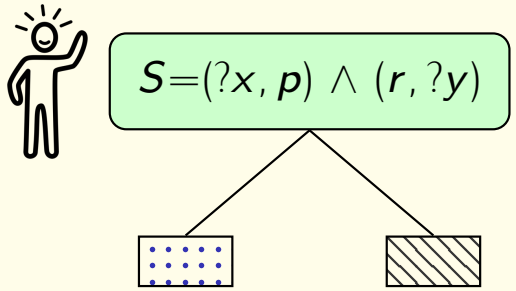
Chained Semantic Matching Algorithm (CSMA)





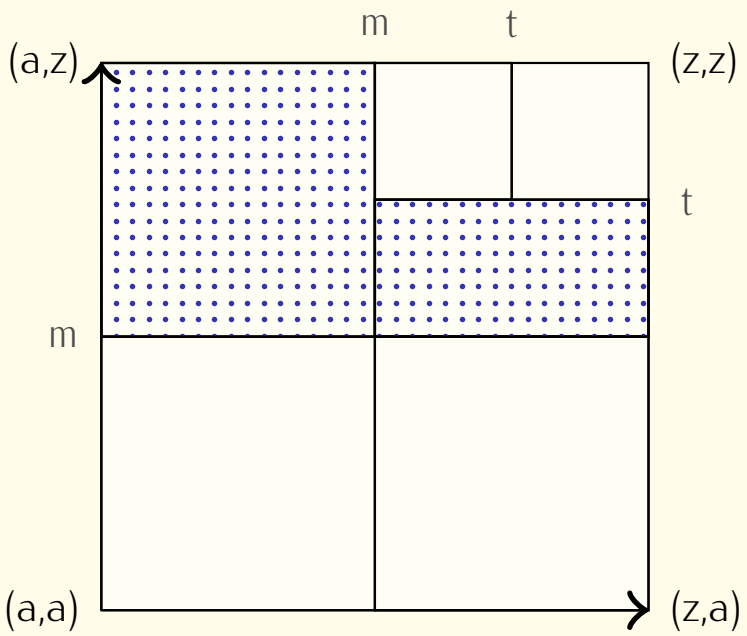
q_1
q_2
q_3
$= CE$

Chained Semantic Matching Algorithm (CSMA)



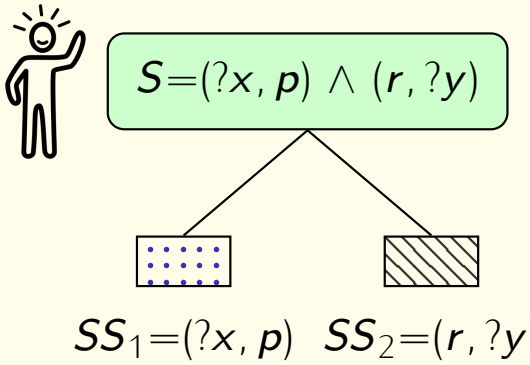
Indexing CE

$q_1=(h, p)$
$q_2=(x, q)$
$q_3=(r, d)$



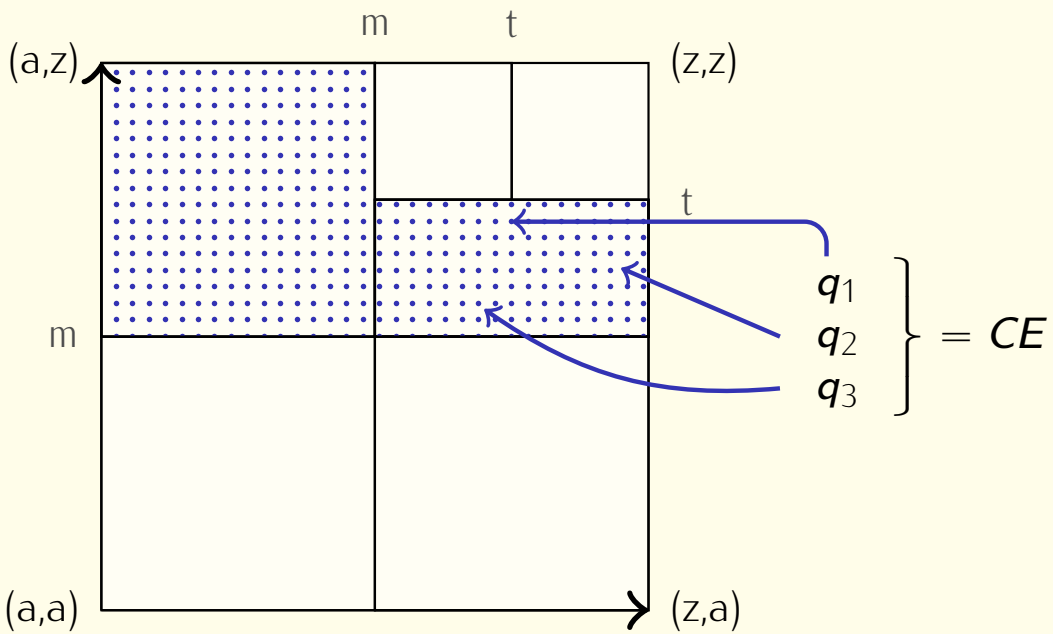
q_1
q_2
q_3
$= CE$

Chained Semantic Matching Algorithm (CSMA)

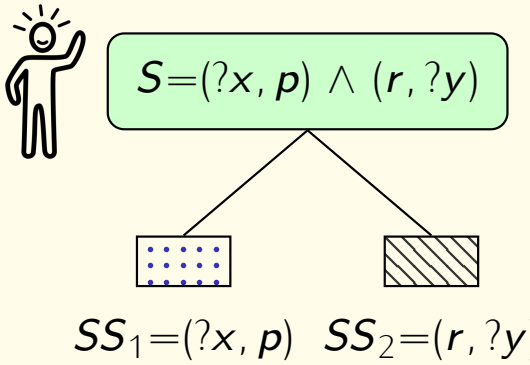


Indexing CE

$q_1=(h, p)$
$q_2=(x, q)$
$q_3=(r, d)$

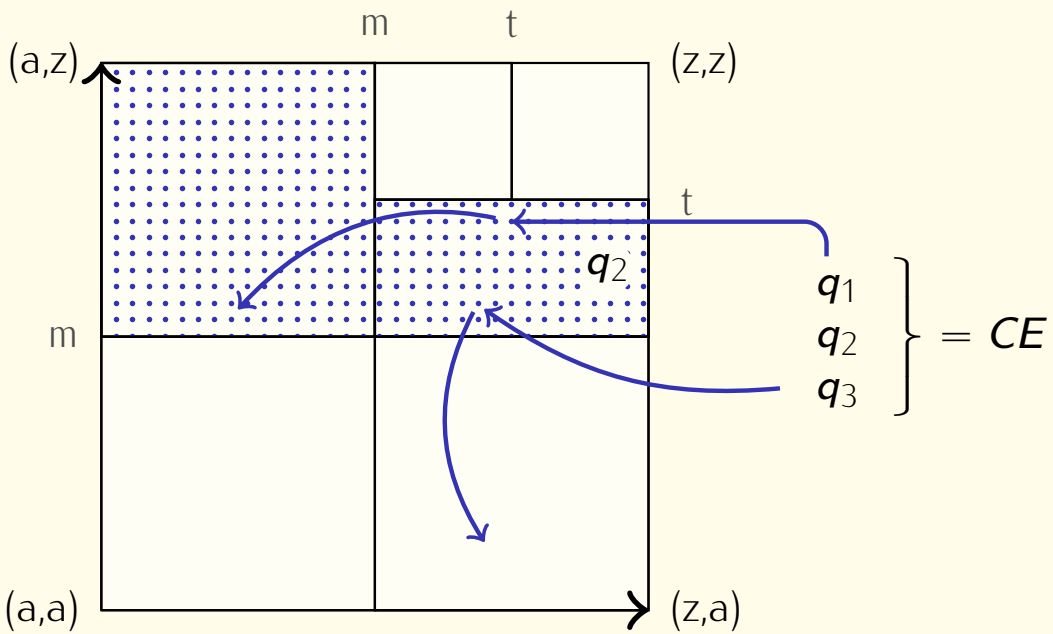


Chained Semantic Matching Algorithm (CSMA)

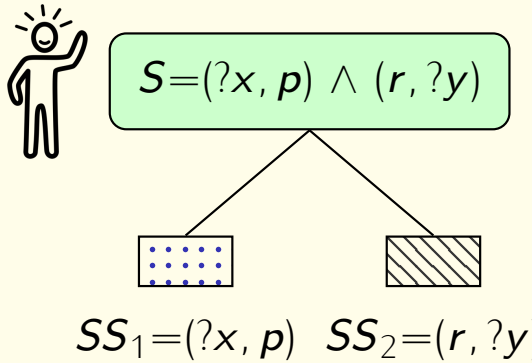


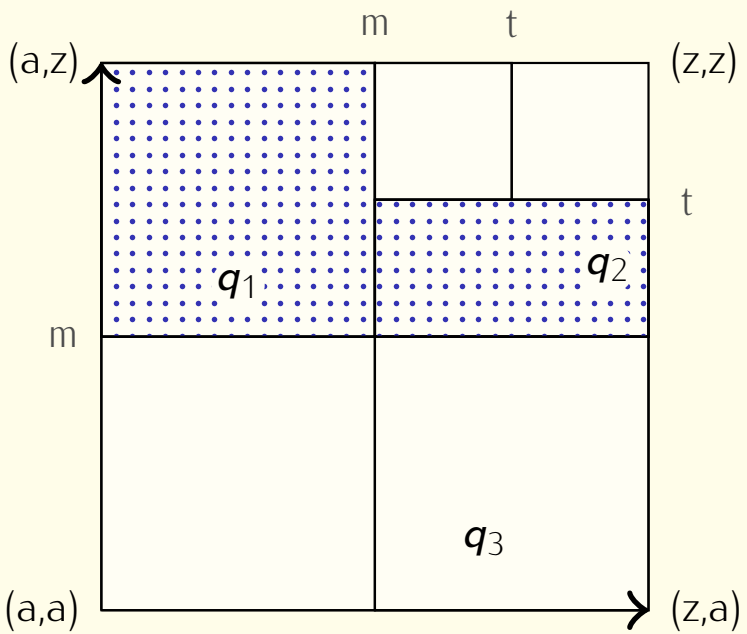
Indexing CE

$q_1=(h, p)$
$q_2=(x, q)$
$q_3=(r, d)$



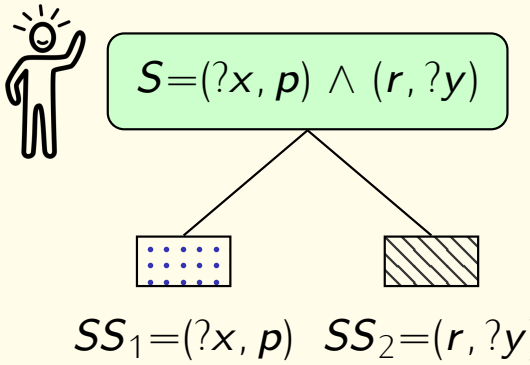
Chained Semantic Matching Algorithm (CSMA)





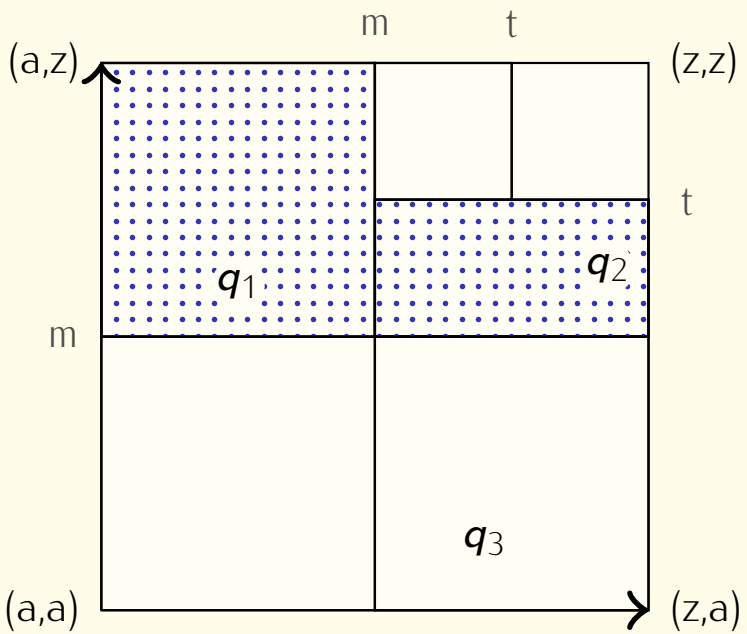
$$\left.\begin{matrix} q_1 \\ q_2 \\ q_3 \end{matrix}\right\} = CE$$

Chained Semantic Matching Algorithm (CSMA)



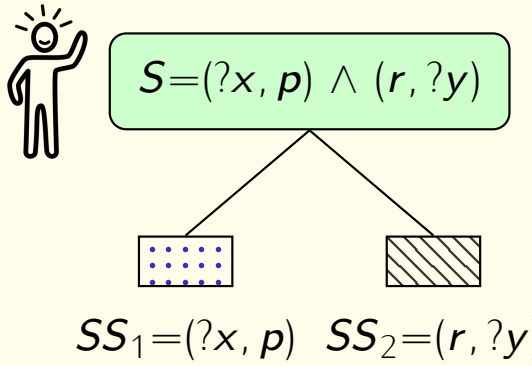
Performing matching between subscriptions and events

q_2 does not satisfy SS_1
q_3 matches no subscription
q_1 matches SS_1



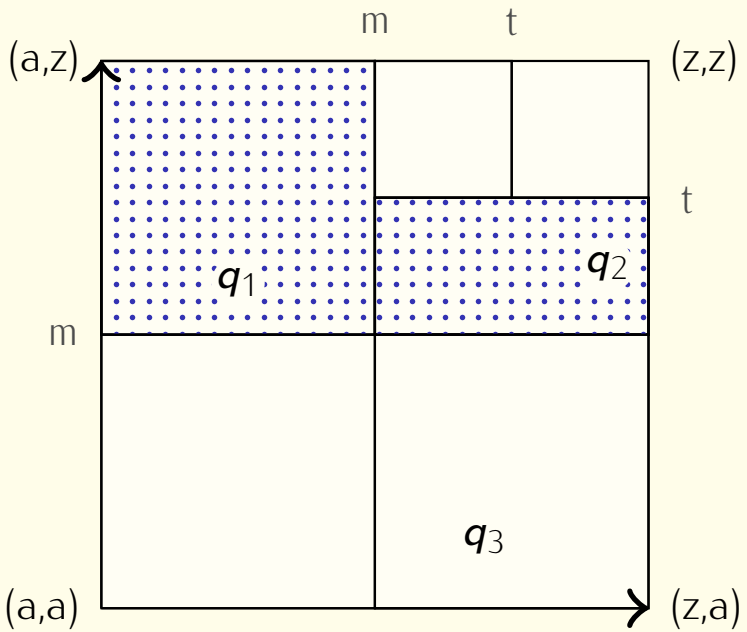
q_1
q_2
q_3
$\}$ $= CE$

Chained Semantic Matching Algorithm (CSMA)



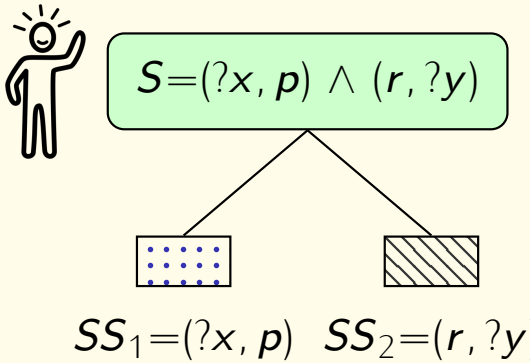
Rewriting S into S'

$S'=(r, ?y)$



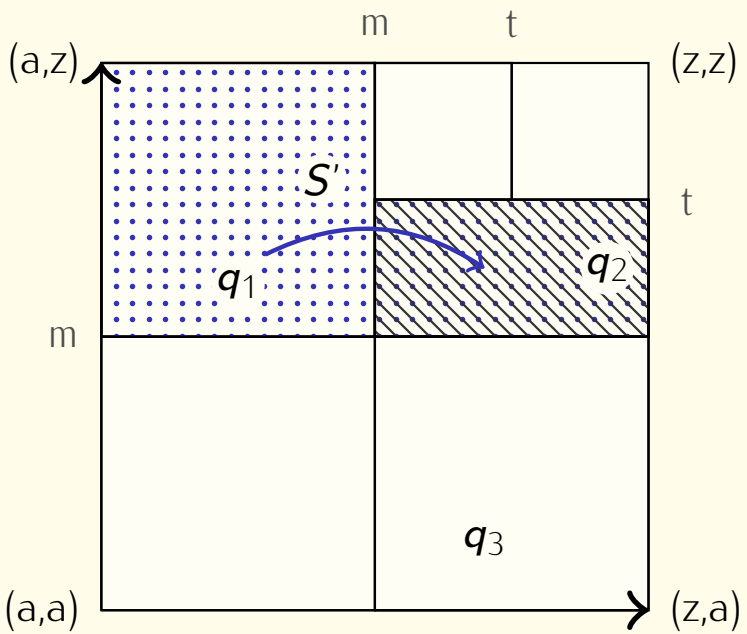
$$\left.\begin{matrix} q_1 \\ q_2 \\ q_3 \end{matrix}\right\} = CE$$

Chained Semantic Matching Algorithm (CSMA)



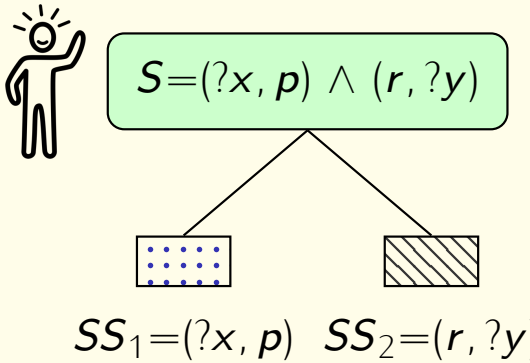
Rewriting S into S'

$S'=(r, ?y)$



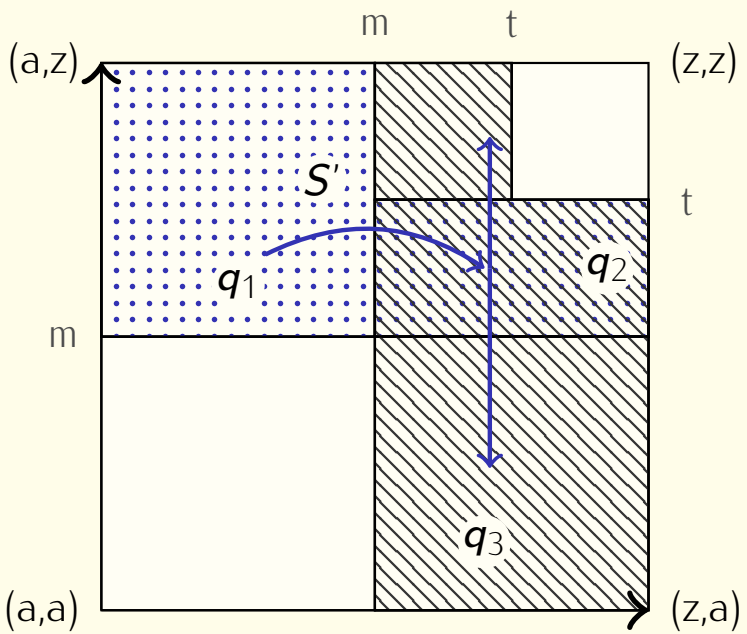
$$\left.\begin{matrix} q_1 \\ q_2 \\ q_3 \end{matrix}\right\} = CE$$

Chained Semantic Matching Algorithm (CSMA)



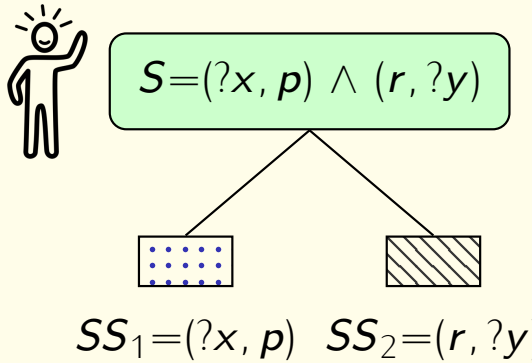
Rewriting S into S'

$S'=(r, ?y)$



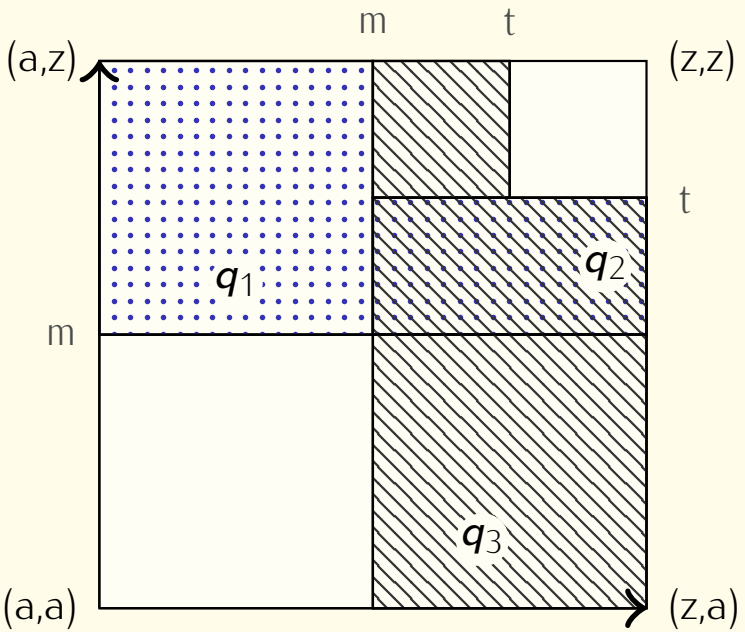
q_1, q_2, q_3 $\}$ $= CE$

Chained Semantic Matching Algorithm (CSMA)



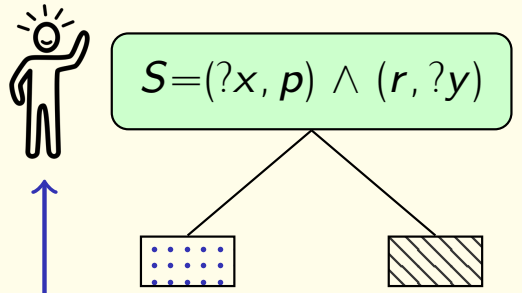
Checking subscriptions satisfaction

q_2 is not satisfying SS_1
q_3 is satisfying SS_2

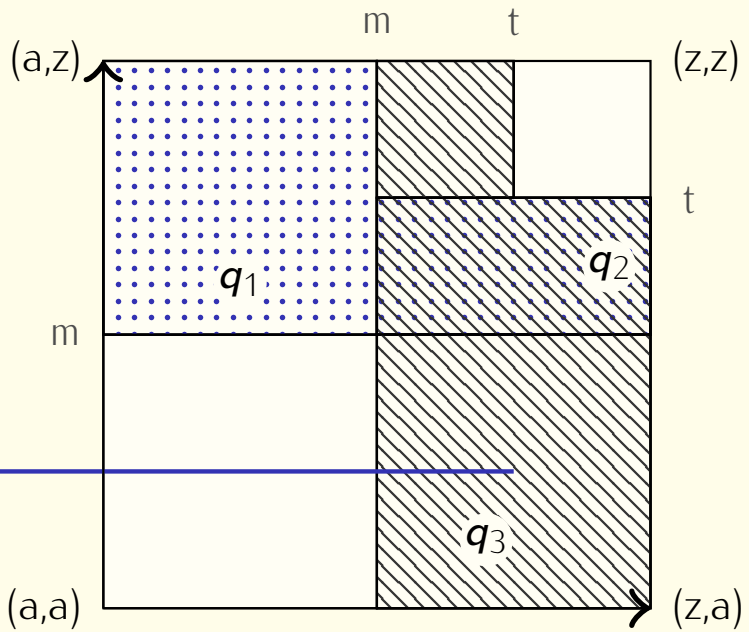


q_1
q_2
q_3
$= CE$

Chained Semantic Matching Algorithm (CSMA)



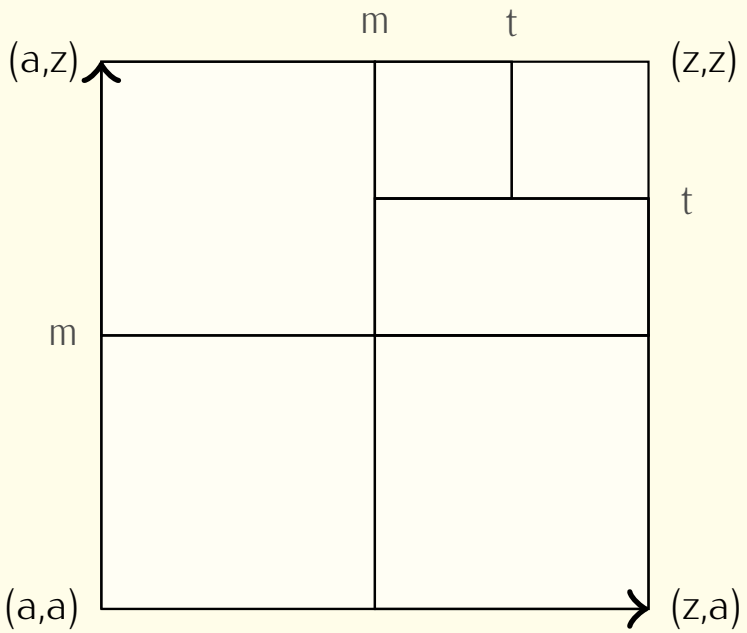
Trigger notification to user



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

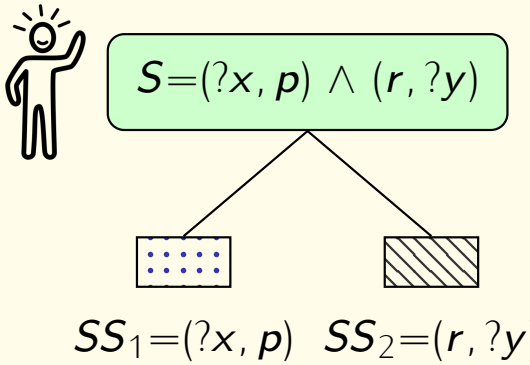
One-step Semantic Matching Algorithm (OSMA)

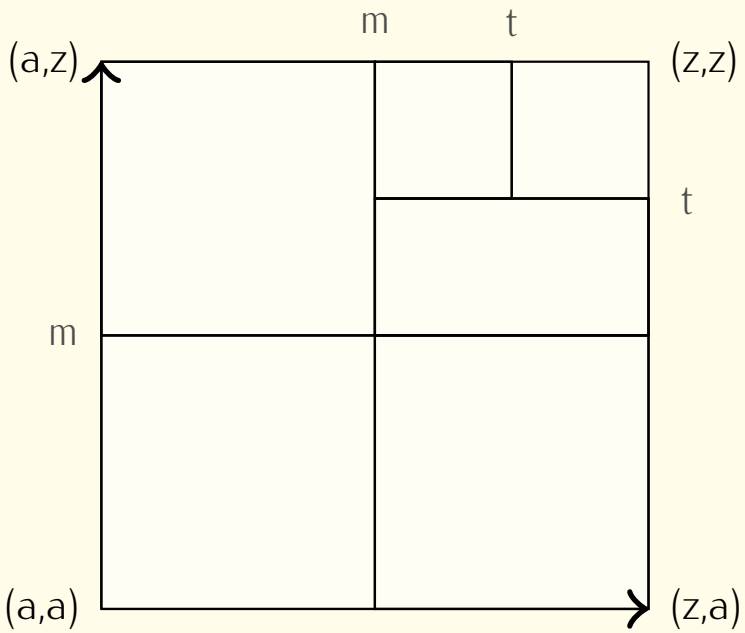
$S = (?x, p) \wedge (r, ?y)$



q_1
q_2
q_3
$= CE$

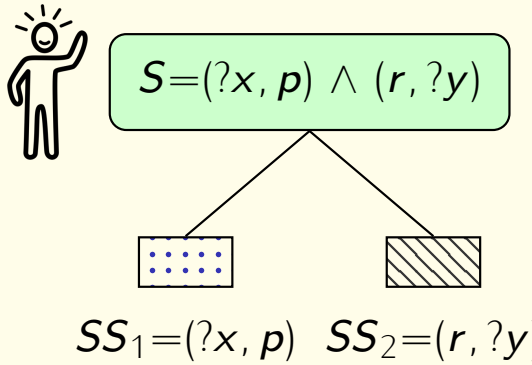
One-step Semantic Matching Algorithm (OSMA)



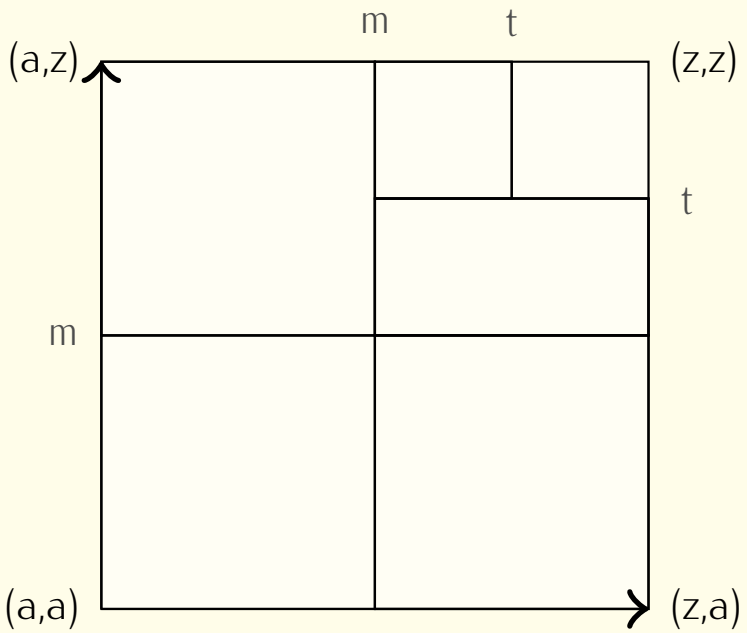


q_1
q_2
q_3
$= CE$

One-step Semantic Matching Algorithm (OSMA)

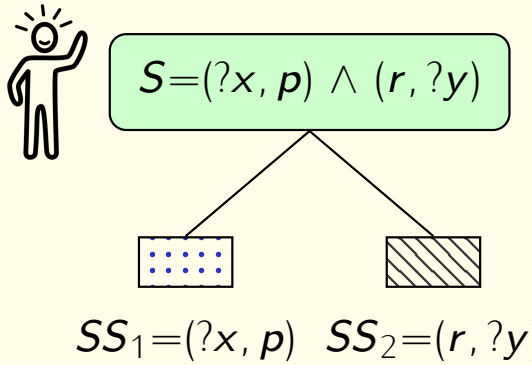


Indexing S according to S_1

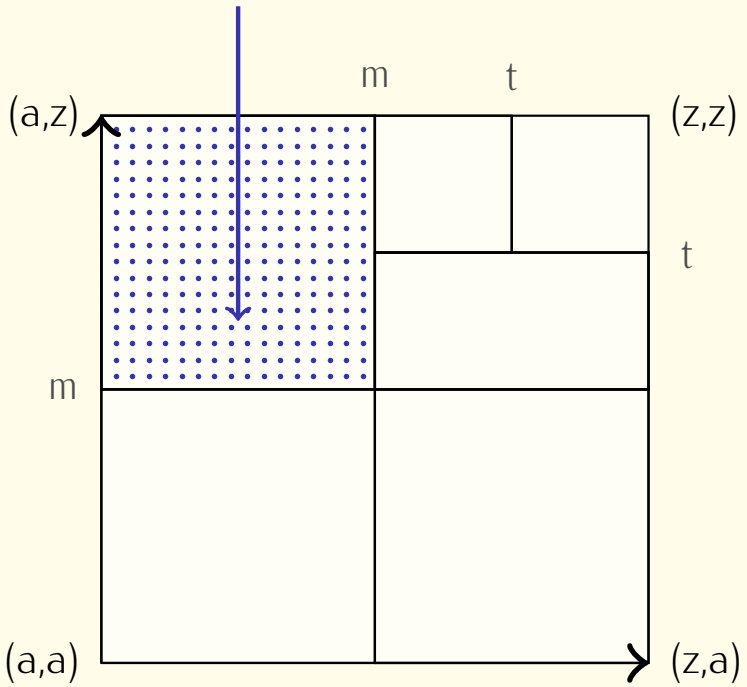


q_1
q_2
q_3
$= CE$

One-step Semantic Matching Algorithm (OSMA)

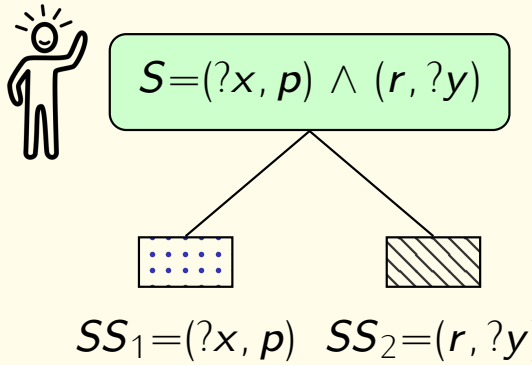


Indexing S according to S_1

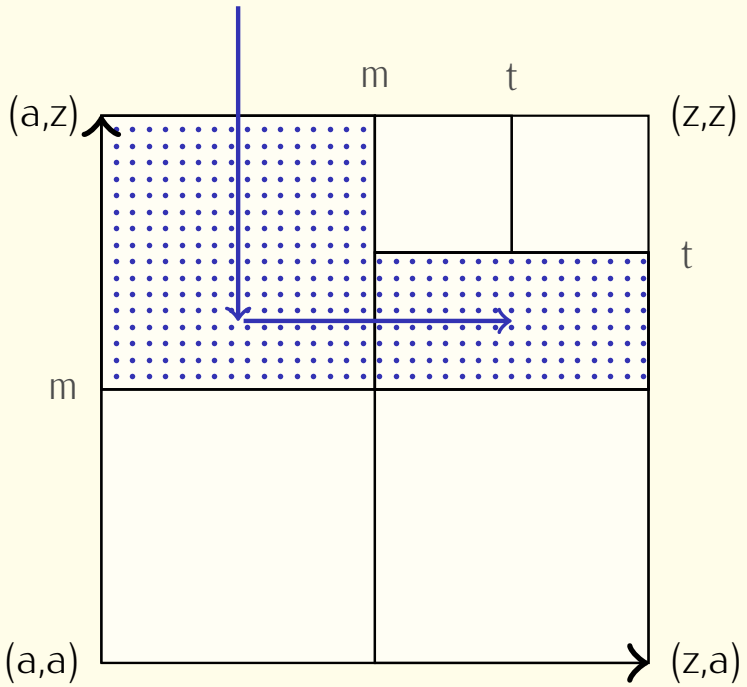


q_1
q_2
q_3
$= CE$

One-step Semantic Matching Algorithm (OSMA)

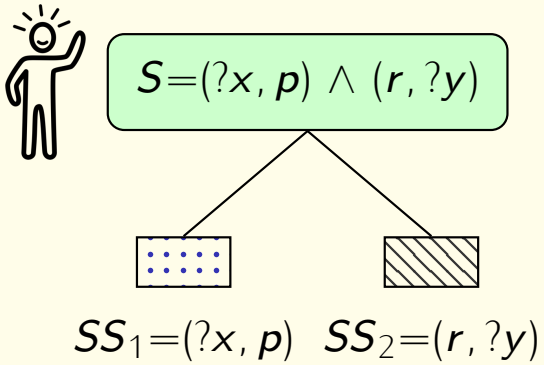


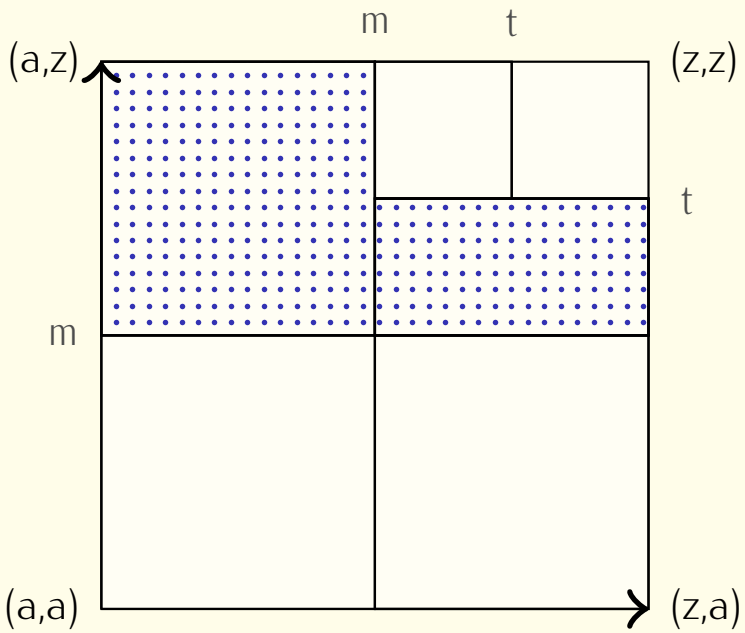
Indexing S according to S_1



q_1
q_2
q_3
$\} = CE$

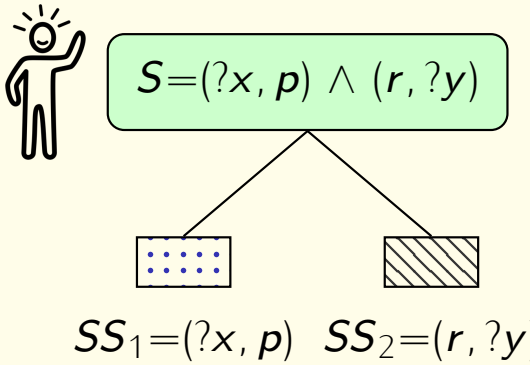
One-step Semantic Matching Algorithm (OSMA)





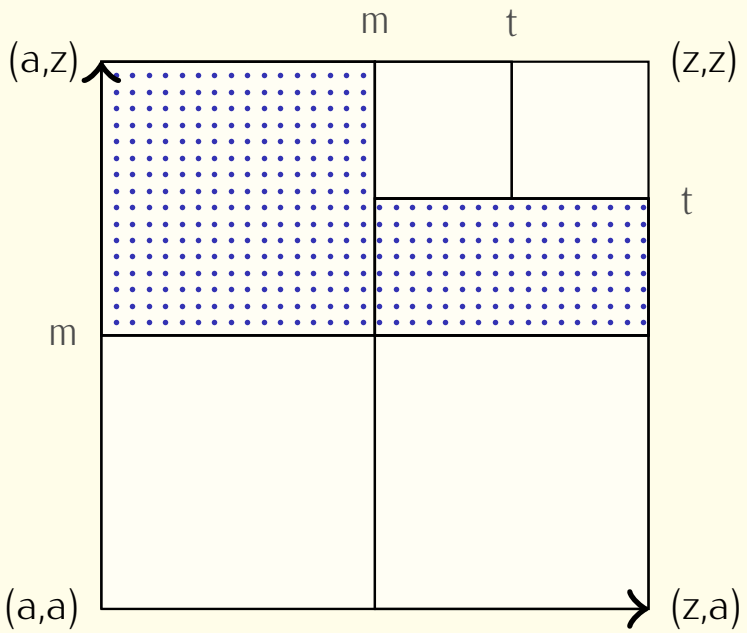
q_1
q_2
q_3
$= CE$

One-step Semantic Matching Algorithm (OSMA)



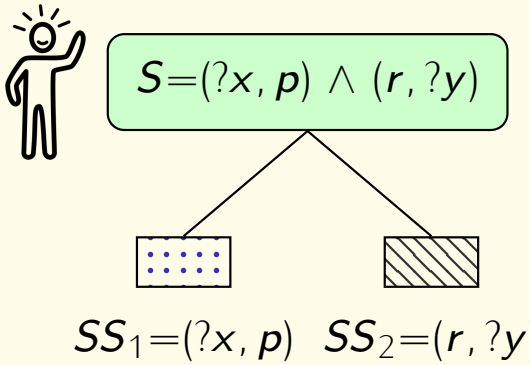
Indexing CE

$q_1=(h, p)$
$q_2=(x, q)$
$q_3=(r, d)$



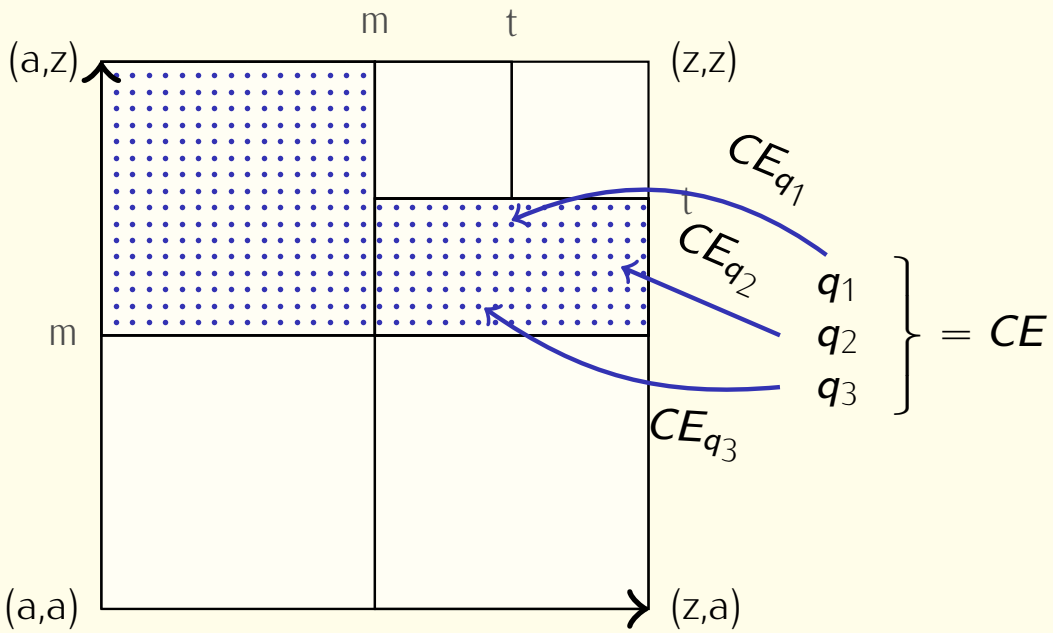
q_1
q_2
q_3 $\Big\}$ $= CE$

One-step Semantic Matching Algorithm (OSMA)

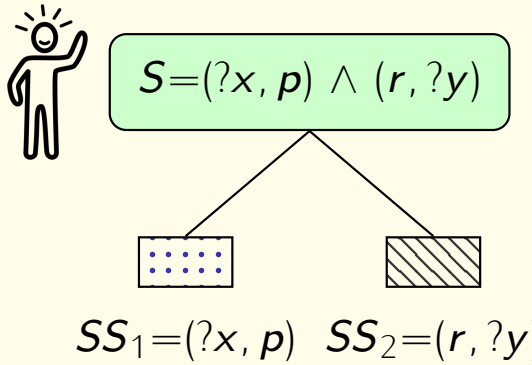


Indexing CE

$q_1=(h, p)$
$q_2=(x, q)$
$q_3=(r, d)$

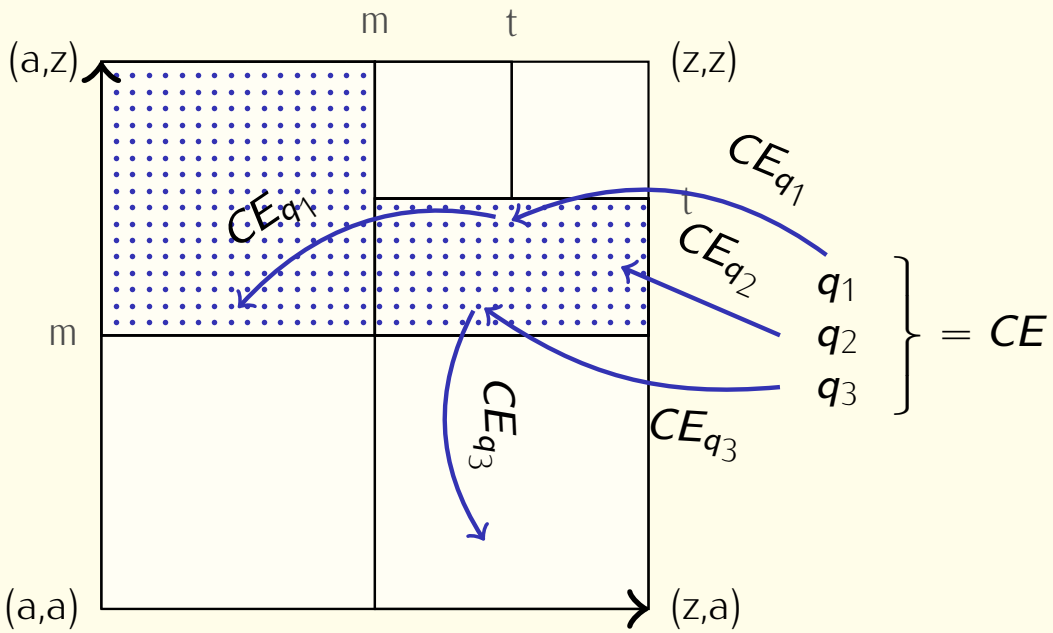


One-step Semantic Matching Algorithm (OSMA)

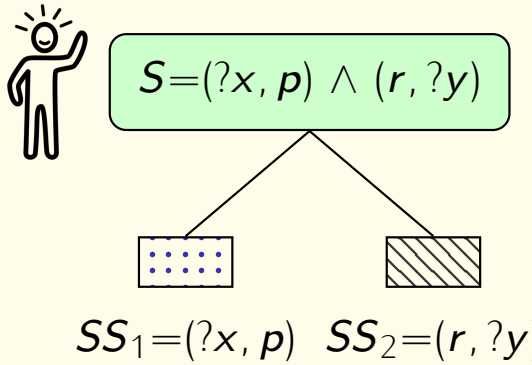


Indexing CE

$q_1=(h, p)$
$q_2=(x, q)$
$q_3=(r, d)$

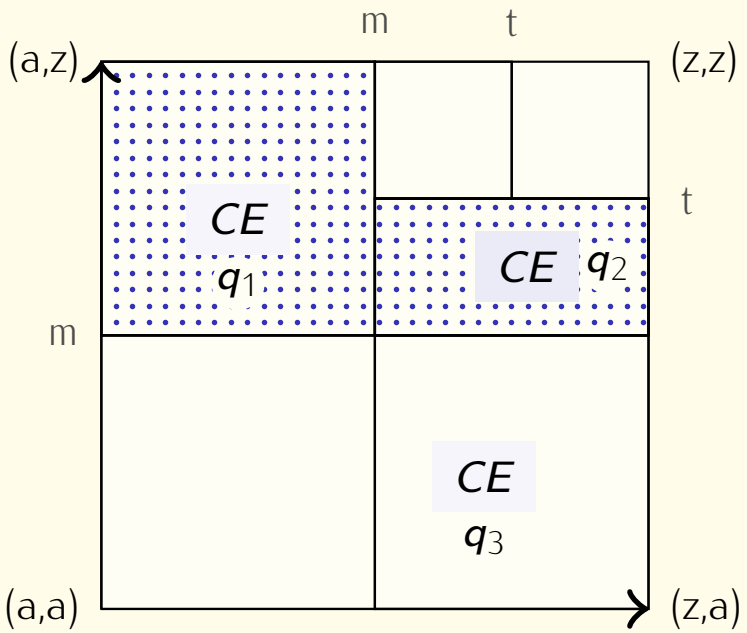


One-step Semantic Matching Algorithm (OSMA)



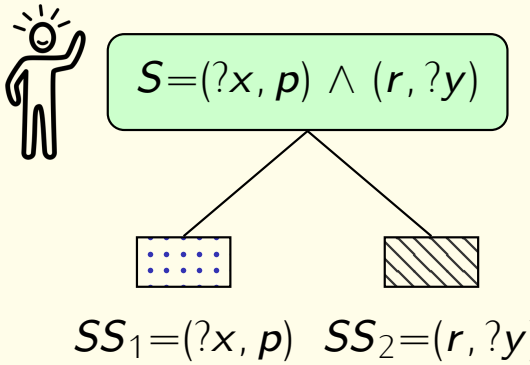
Indexing CE

$q_1=(h, p)$
$q_2=(x, q)$
$q_3=(r, d)$

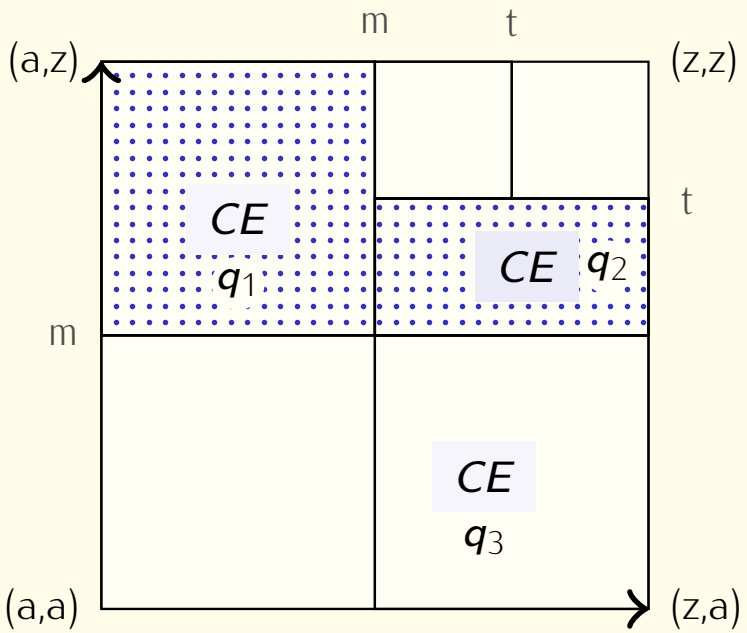


q_1
q_2
q_3
$= CE$

One-step Semantic Matching Algorithm (OSMA)

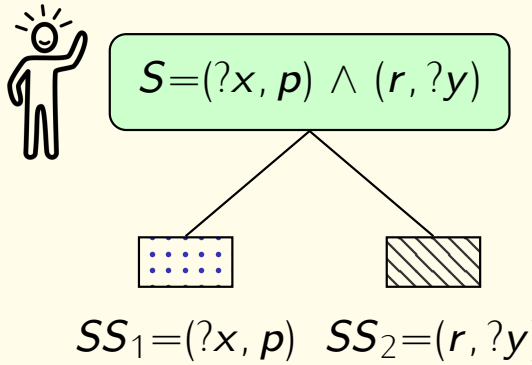


Performing matching between subscriptions and events

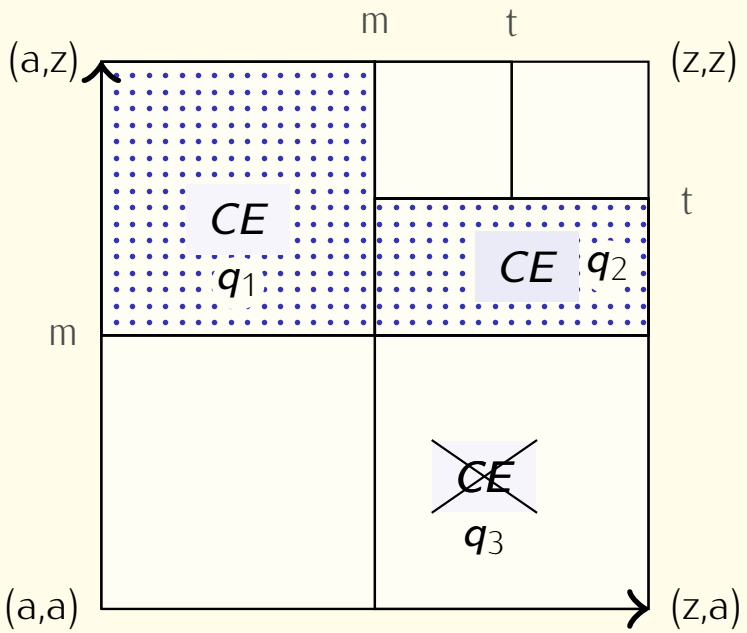


q_1
q_2
q_3
$\}$ $= CE$

One-step Semantic Matching Algorithm (OSMA)

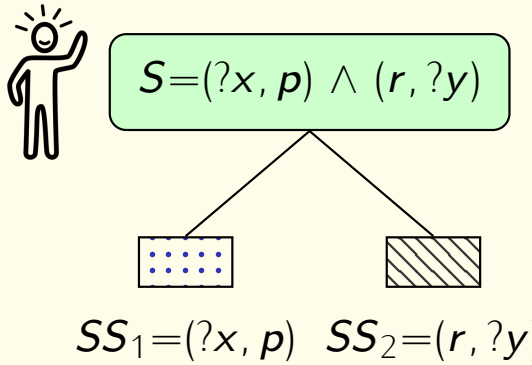


q_3 and the CE sent along with the quadruple matches no subscription

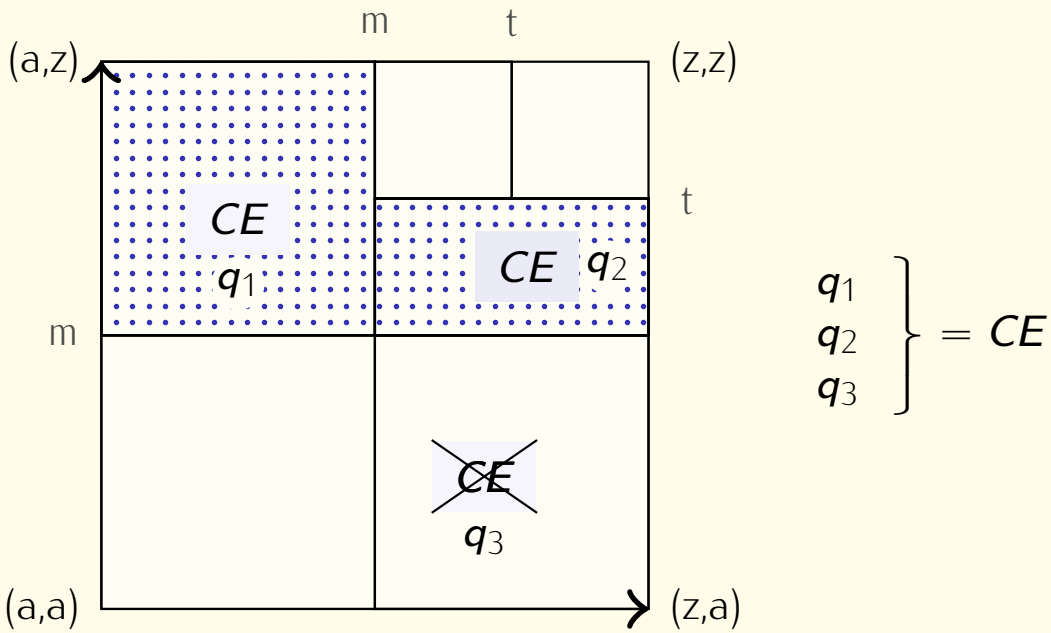


q_1, q_2, q_3 } $= CE$

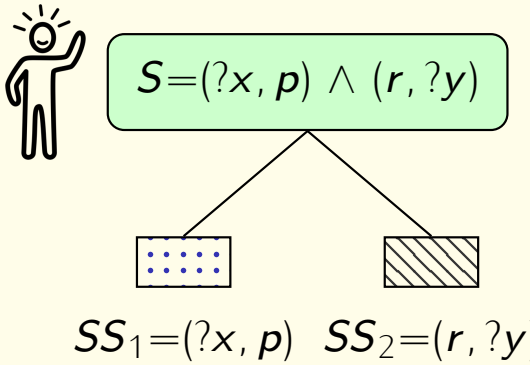
One-step Semantic Matching Algorithm (OSMA)



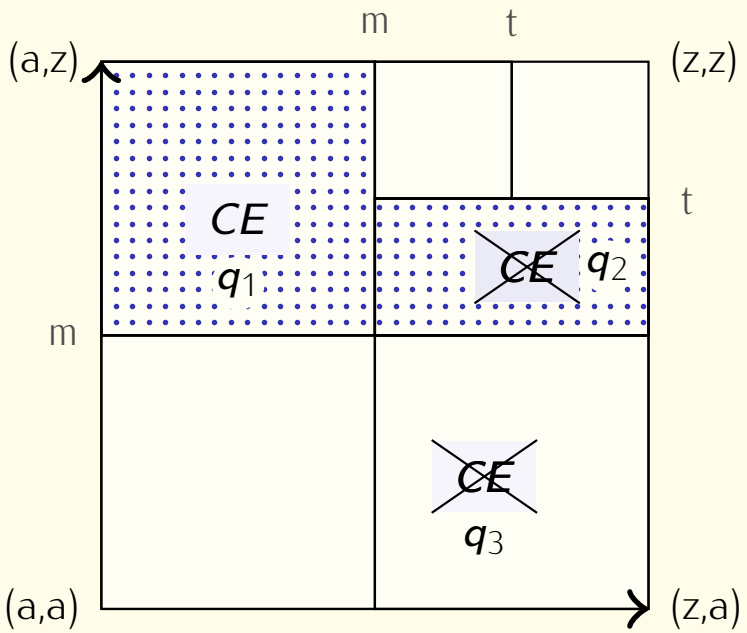
Peers indexing q_1 and q_2 are each satisfying S



One-step Semantic Matching Algorithm (OSMA)

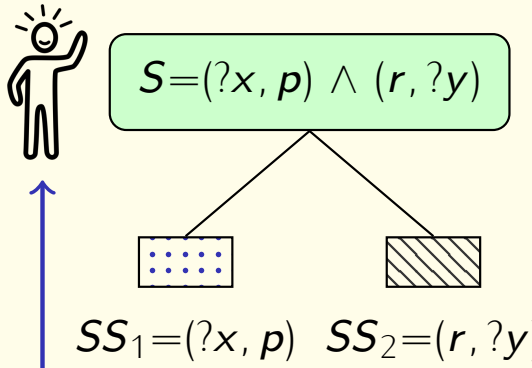


The subscriber is notified by the peer responsible for the first of the matching quadruples contained by the *CE*

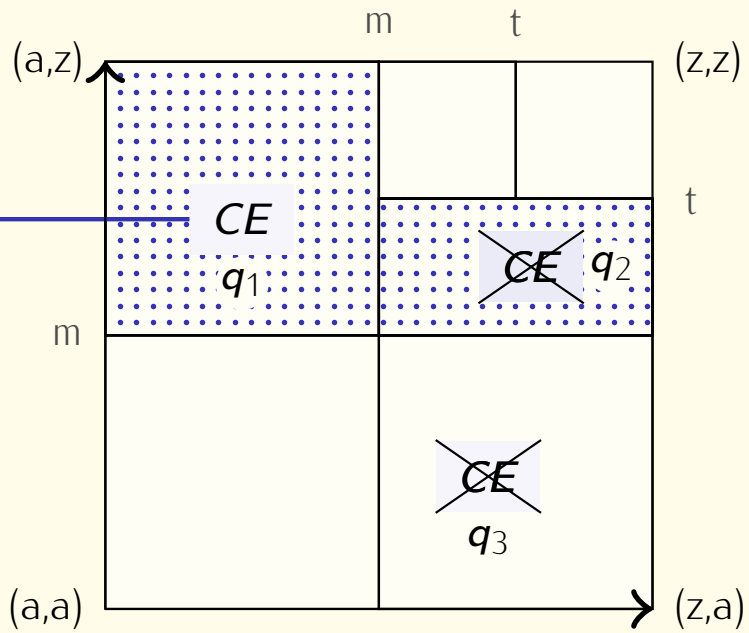


q_1, q_2, q_3 } = *CE*

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*



q_1
q_2
q_3 } = *CE*

CSMA vs OSMA

	Routed Element	Matching Steps	Duplicates	Happen-Before
CSMA	Individual quadruples	Multiple, Chain-like and Recon-struction	Yes, filtering required	Enforced
OSMA	Whole *Compound Event*	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 *Compound Event*	Single	No	Requires CSMA

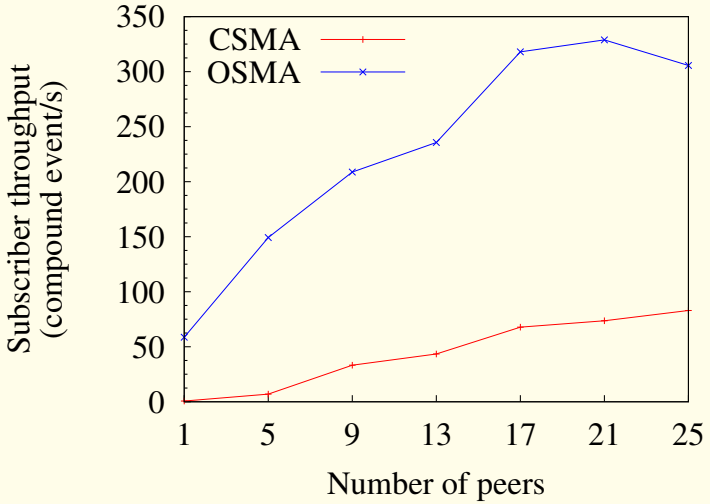
Table: Comparison of the two publish/subscribe algorithms.

Experiments are required to confirm the promising behavior of OSMA

Experimental setup

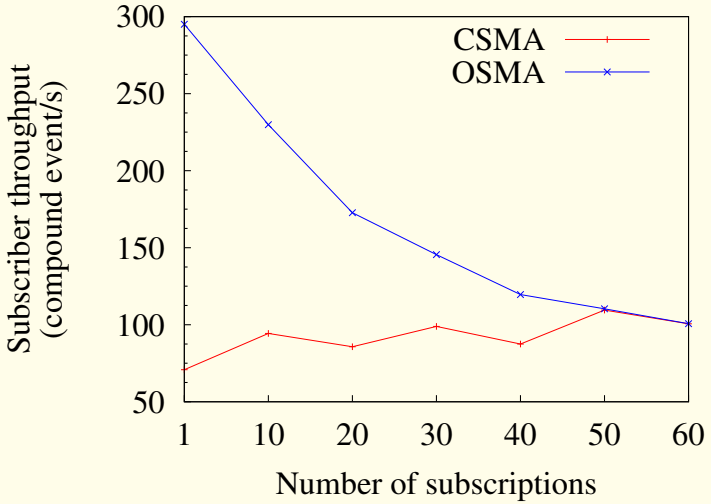
- Up to 25 nodes from the French Grid'5000 testbed
  - Xeon E5520@2,26GHz
  - 32 GB RAM
  - 7200 RPM HDD
- Each result is the average execution time for 6 runs
- Workload made of synthetic events
  - Each CE embeds 5 quadruples
  - Size per CE is about 670 Bytes
- Subscriptions are path queries
  - Each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

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

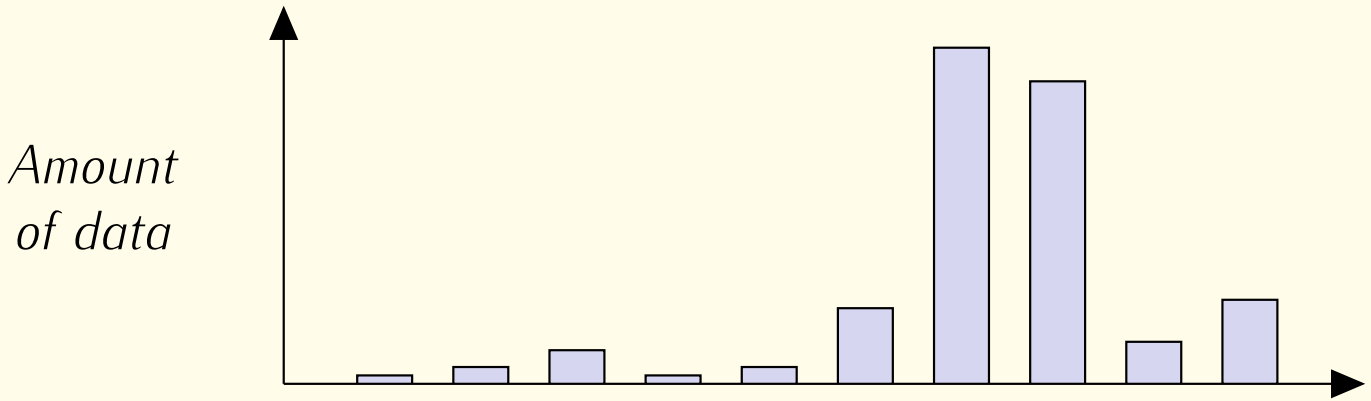
CSMA throughput remains almost stable because rewritten subscriptions involves more peers.

Trade-off between CSMA and OSMA

Outline

Background

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

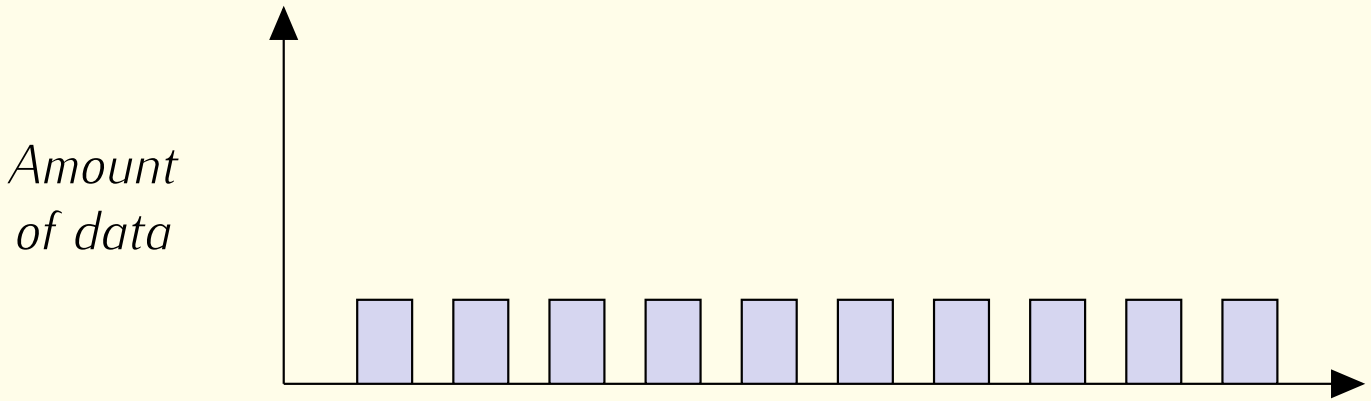


Skewed distribution on peers

[14] 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[14] the most popular term appear around 10–20% of all others



Perfect distribution on peers

[14] Kotoulas, Oren, and Van Harmelen, "Mind the data skew: distributed inferencing by speeddating in elastic regions".

Related works

Static	Dynamic
Bayers *et al.*	Godfrey *et al.*
Battré *et al.*	Bienkowsky *et al.*
RCAN	Vu *et al.*
Meghdoot	Mercury

- Load balancing decision during join/leave
- Usually one resource considered
- Solutions that often differ by minor but subtle changes

Related works

Static	Dynamic
Bayers *et al.*	Godfrey *et al.*
Battré *et al.*	Bienkowsky *et al.*
RCAN	Vu *et al.*
Meghdoot	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 $m \geqslant E[i] \times K_1[i]$ **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

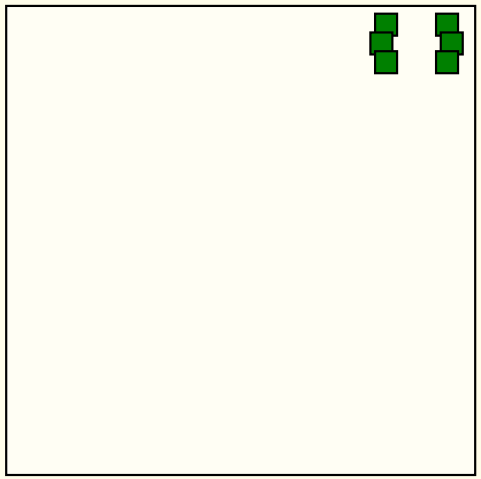
1. Detecting load imbalances
   - Measuring load
     - Define resources/criteria to consider
   - Deciding about imbalance
     - Define imbalance states
     - Propose load state estimation algorithm
     - Define how threshold is estimated
2. Balancing the load
   - Selecting imbalance receiver
     - Use neighbor(s), allocate new peer or leverage information exchanged for estimating threshold values
   - Executing load balancing (per criteria)

Proposed solution

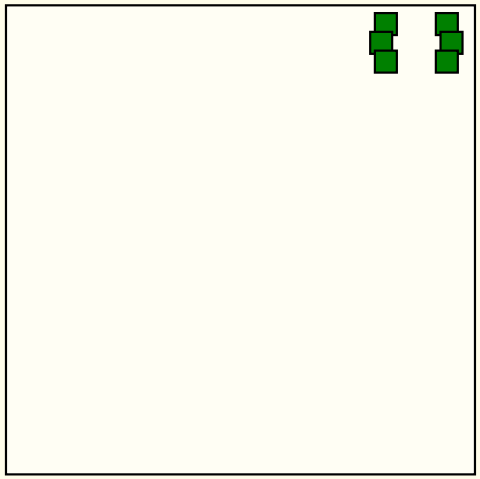
1. Detecting load imbalances
   - Measuring load
     - Define resources/criteria to consider
   - Deciding about imbalance
     - Define imbalance states
     - Propose load state estimation algorithm
     - Define how threshold is estimated
2. Balancing the load
   - Selecting imbalance receiver
 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



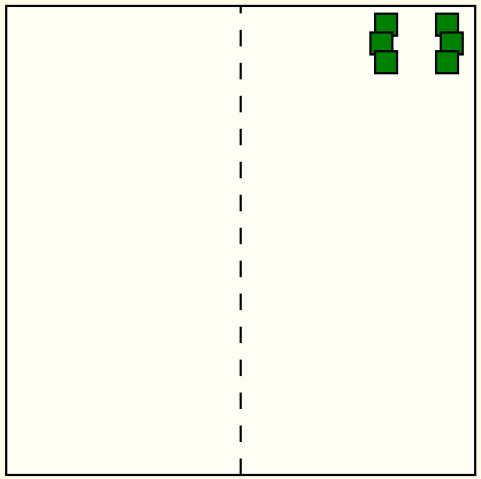
(a) Cutting based on middle



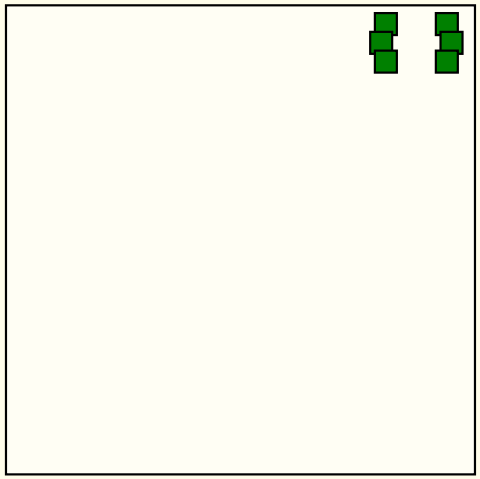
(b) Cutting based on centroid

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



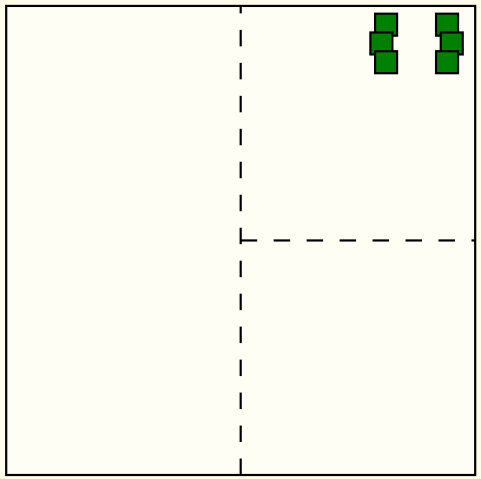
(a) Cutting based on middle



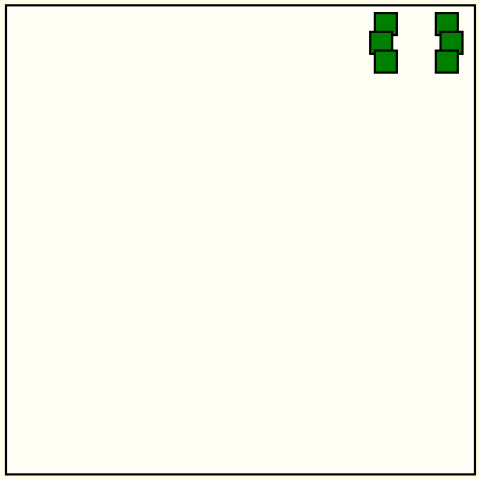
(b) Cutting based on centroid

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



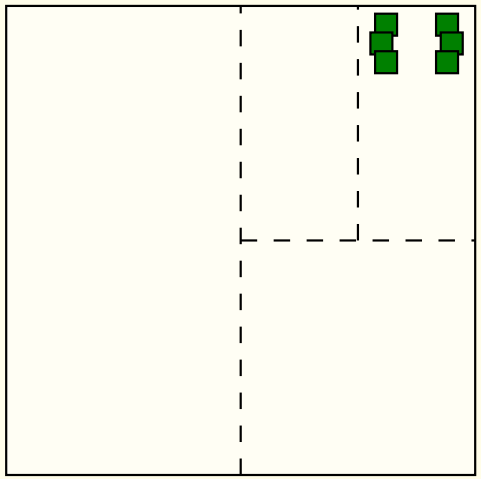
(a) Cutting based on middle



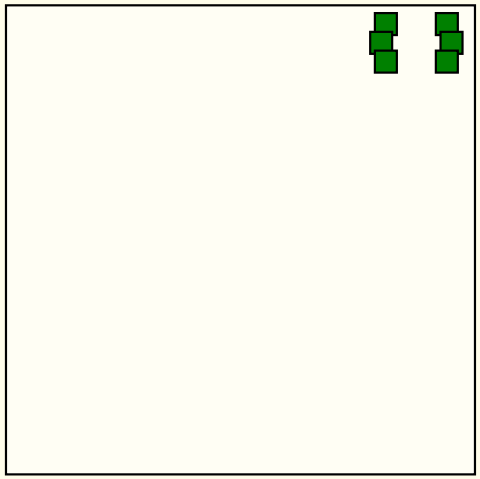
(b) Cutting based on centroid

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



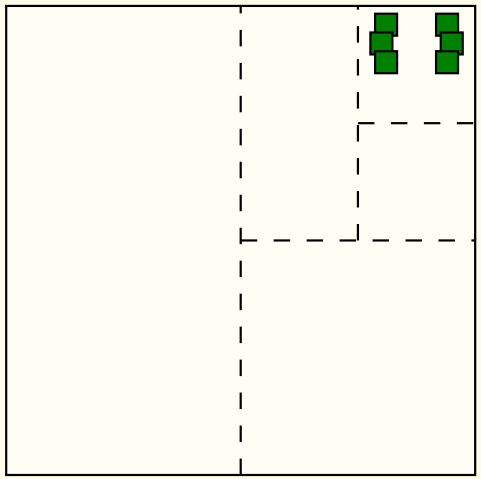
(a) Cutting based on middle



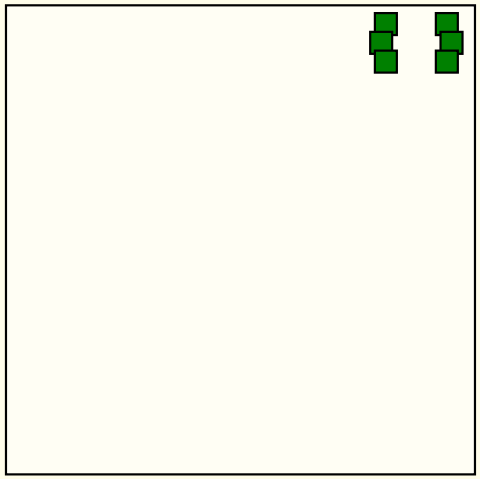
(b) Cutting based on centroid

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



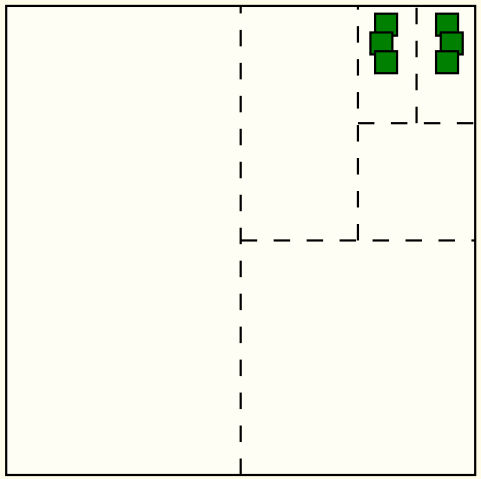
(a) Cutting based on middle



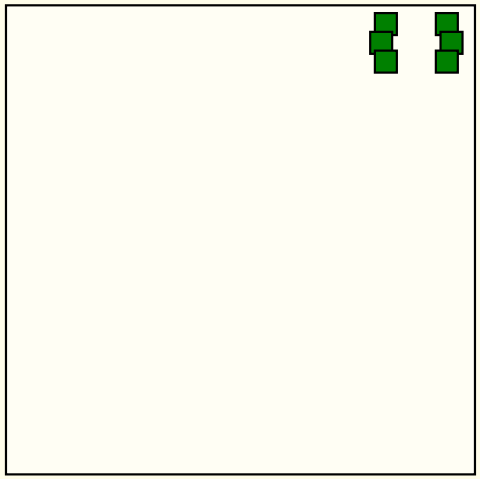
(b) Cutting based on centroid

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



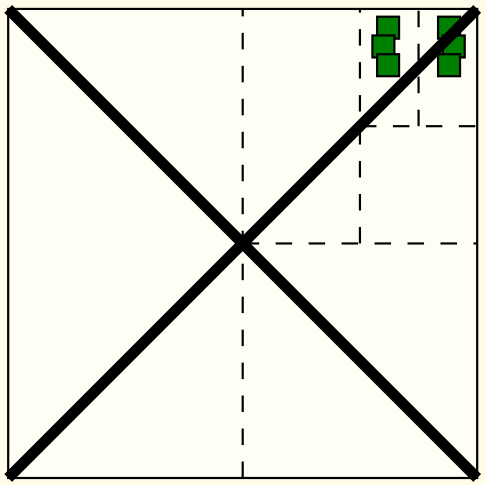
(a) Cutting based on middle



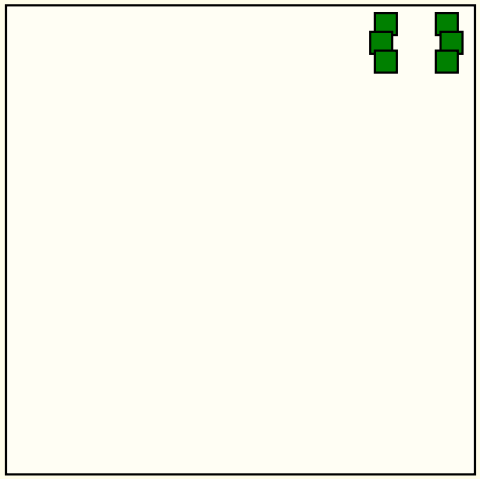
(b) Cutting based on centroid

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



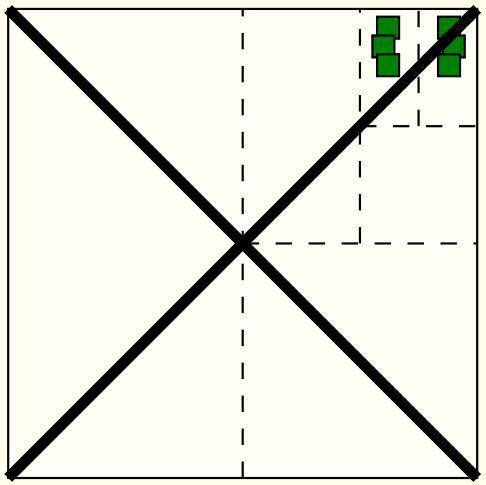
(a) Cutting based on middle



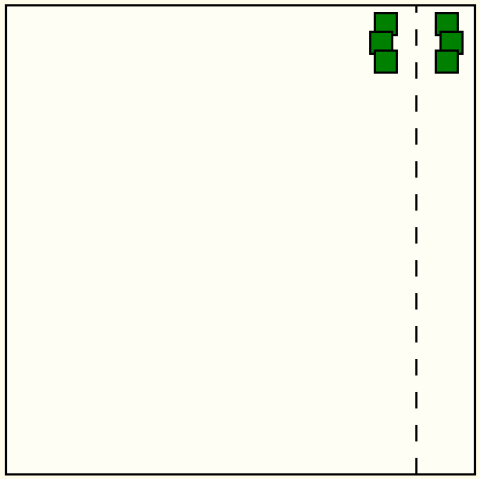
(b) Cutting based on centroid

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**	
Middle	*Centroid*	*Absolute*	*Relative*
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**	
Middle	*Centroid*	*Absolute*	*Relative*
559.4%	69.5%	119.75%	96.57%

Table: Load balancing strategies comparison using relative stddev.

Experiments

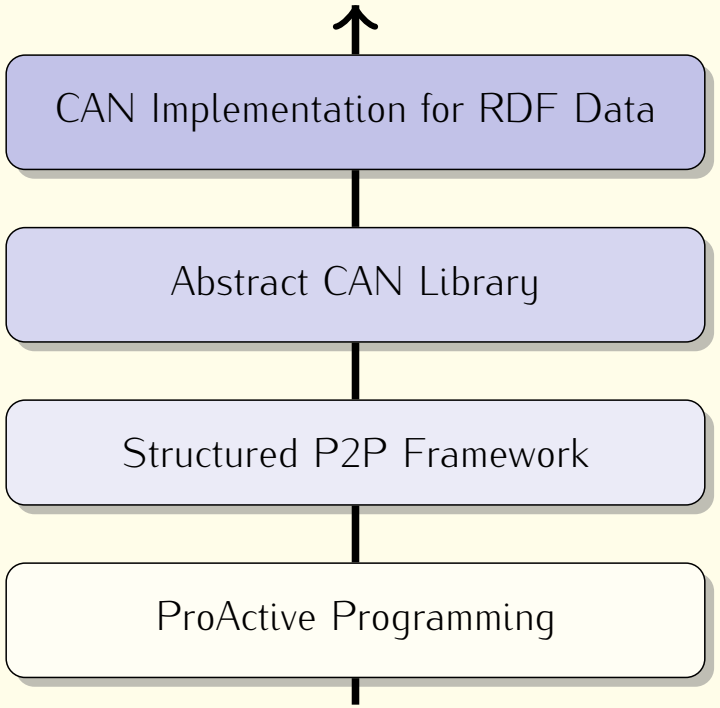
- Workload is real data extracted from a Twitter data flow
  - Converted to our RDF data model with a python adapter
  - Size is about 10^5 quadruples
- Considered RDF data as load balancing criterion
- Virtual peers with peers allocation only (scale-out)
  - Using 32 peers

Static load balancing		**Dynamic load balancing**	
Middle	*Centroid*	*Absolute*	*Relative*
559.4%	69.5%	119.75%	96.57%

Table: Load balancing strategies comparison using relative stddev.

Outline

Middleware design (EventCloud)



EventCloud specific CAN implementation

Provides the subsistence minimum to connect peers and route messages in a CAN topology

Provides reusable concepts to design structured P2P networks

Abstract communications

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

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

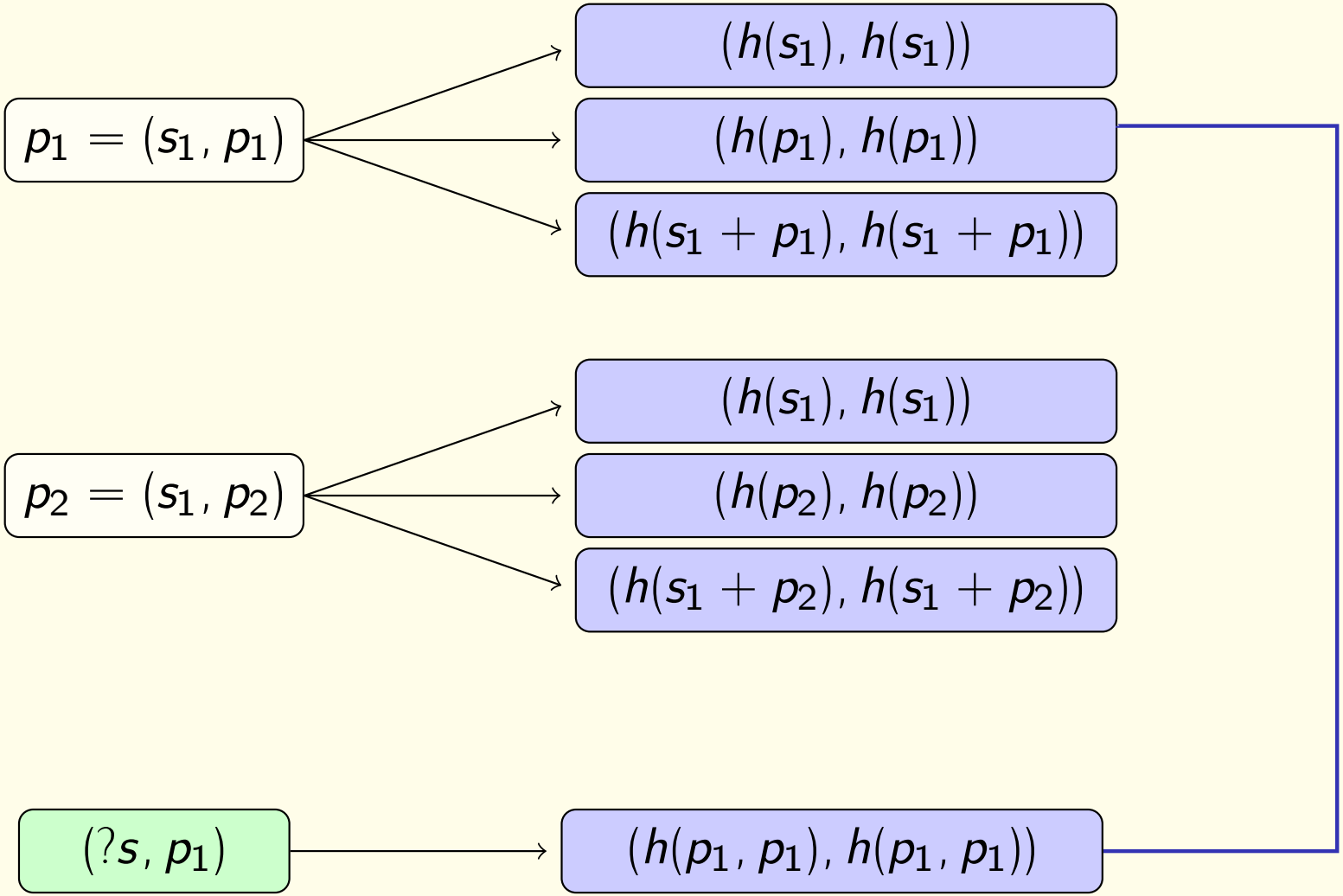
Thank you for your attention

Publications

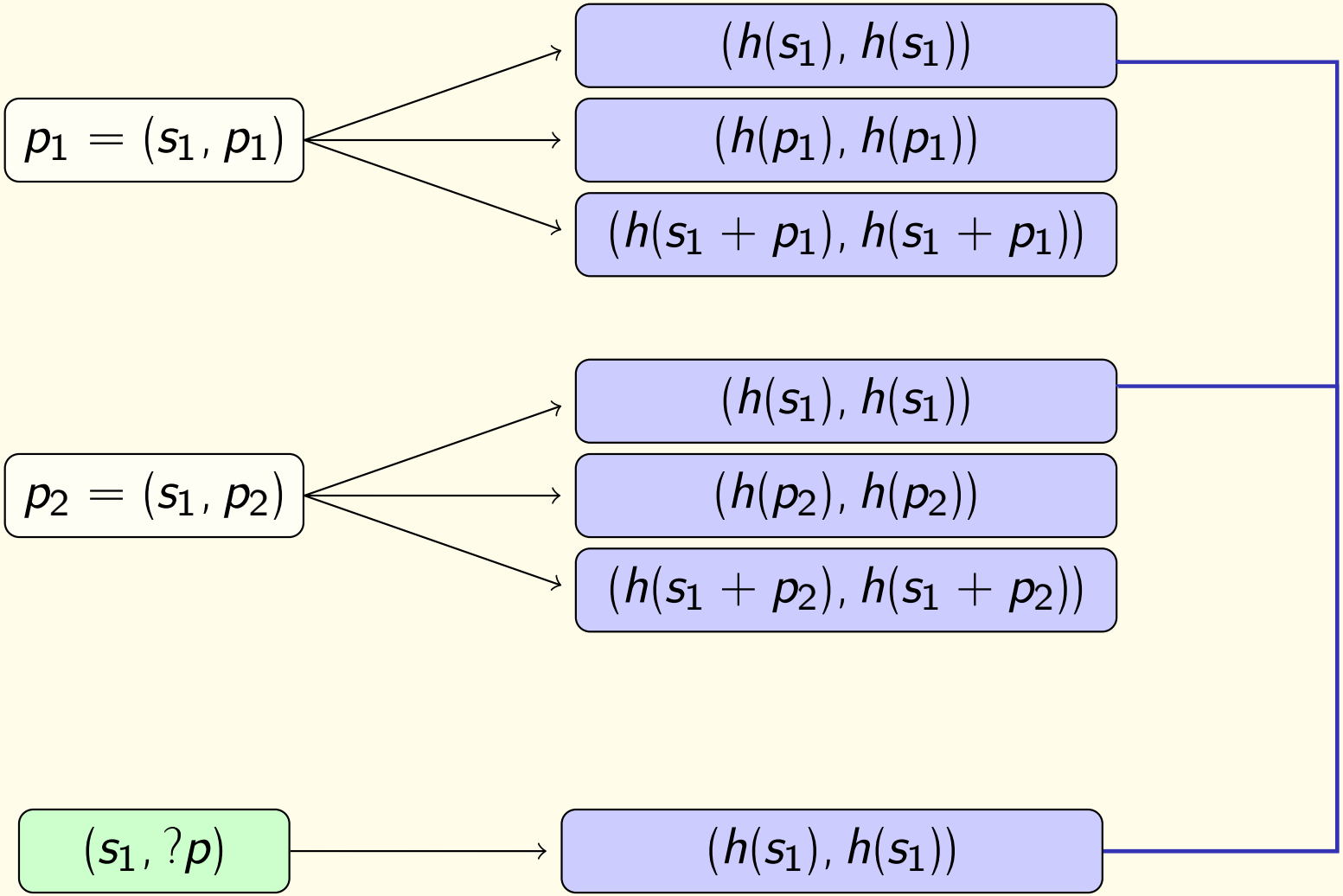
- Imen Filali, Laurent Pellegrino, Francesco Bongiovanni, Fabrice Huet, Françoise Baude, et al. "Modular P2P-based approach for RDF data storage and retrieval". In: *Proceedings of the international conference on Advances in P2P Systems*. 2011.
- Laurent Pellegrino, Iyad Alshabani, Françoise Baude, Roland 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

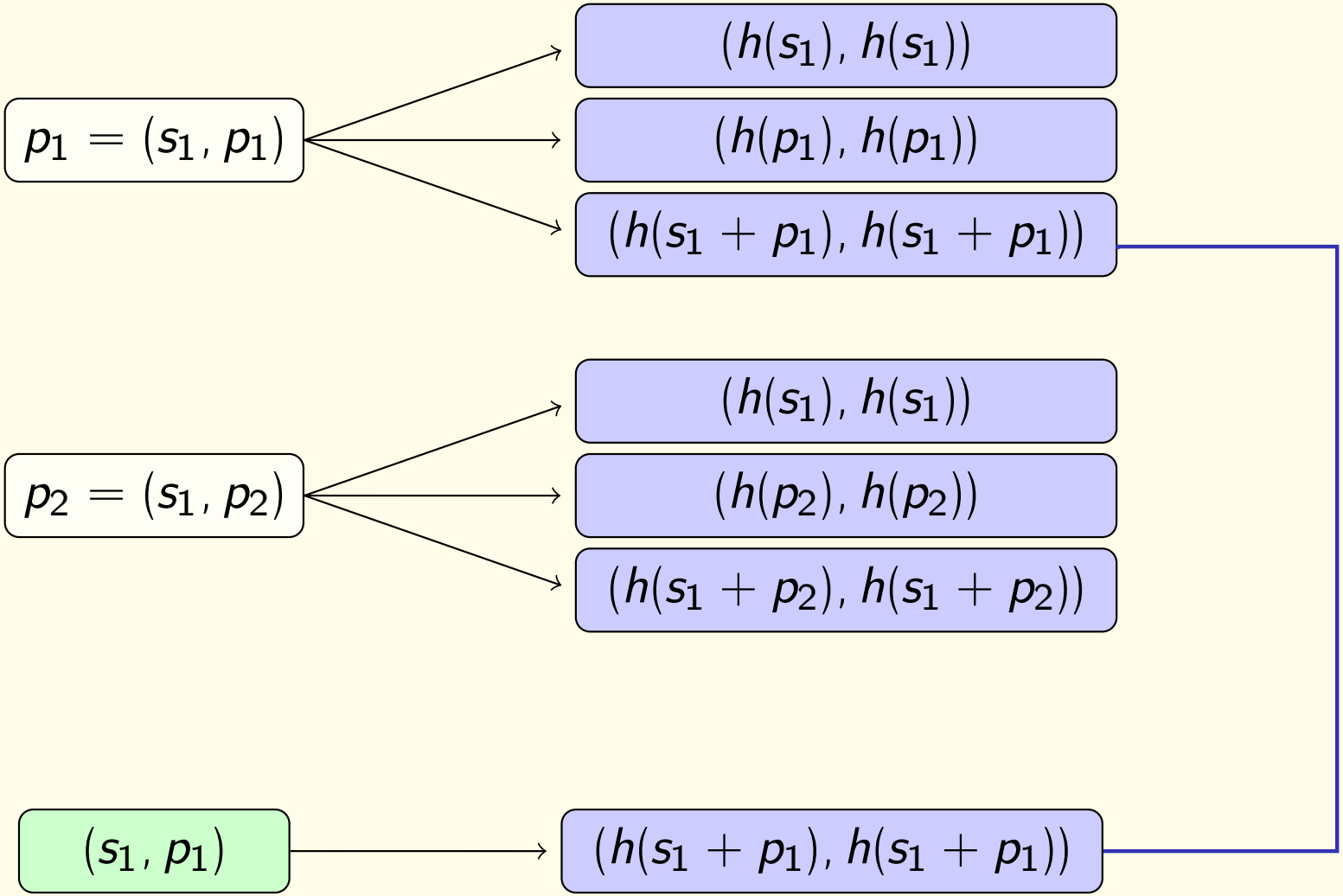
Hash based indexing using indexes



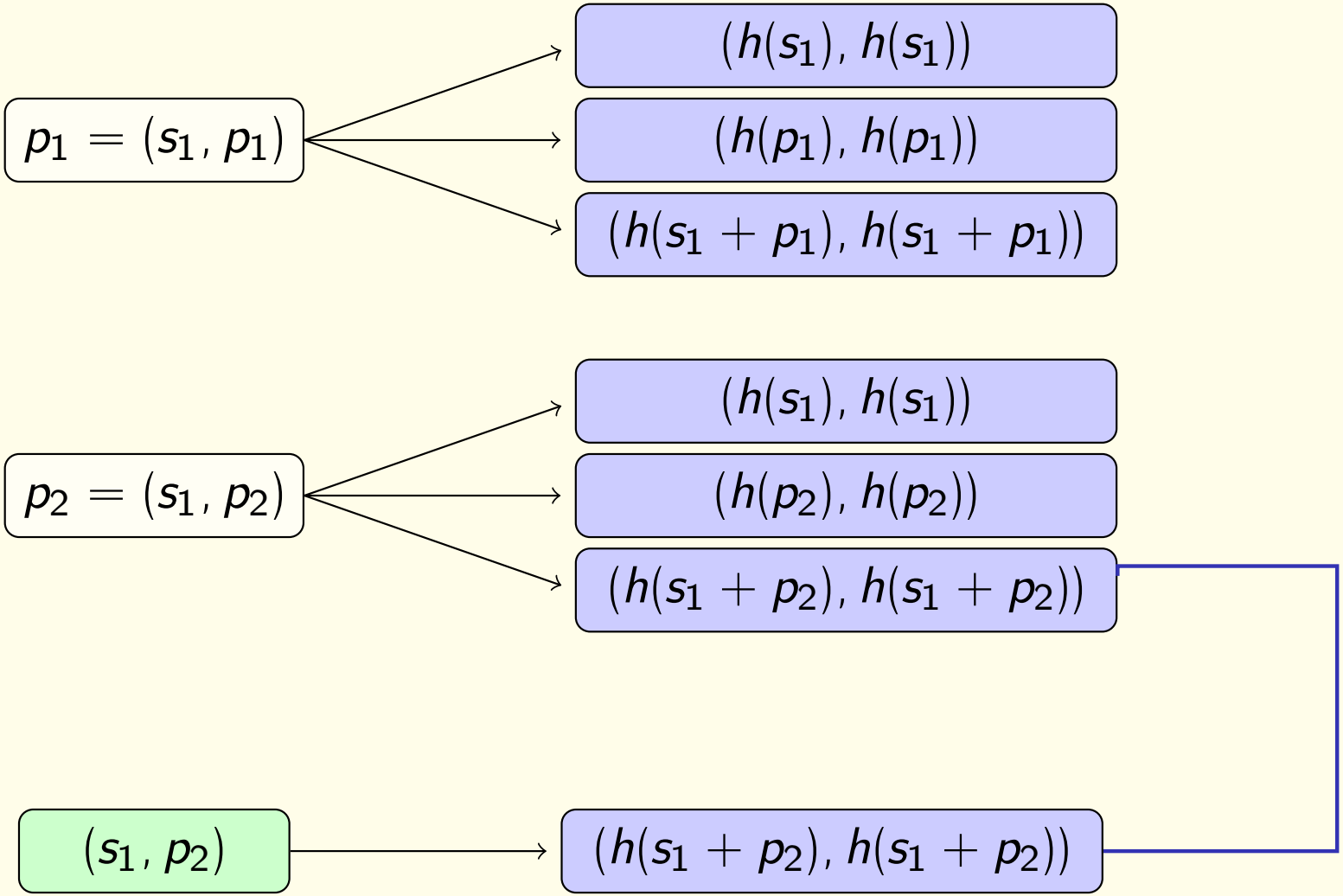
Hash based indexing using indexes



Hash based indexing using indexes

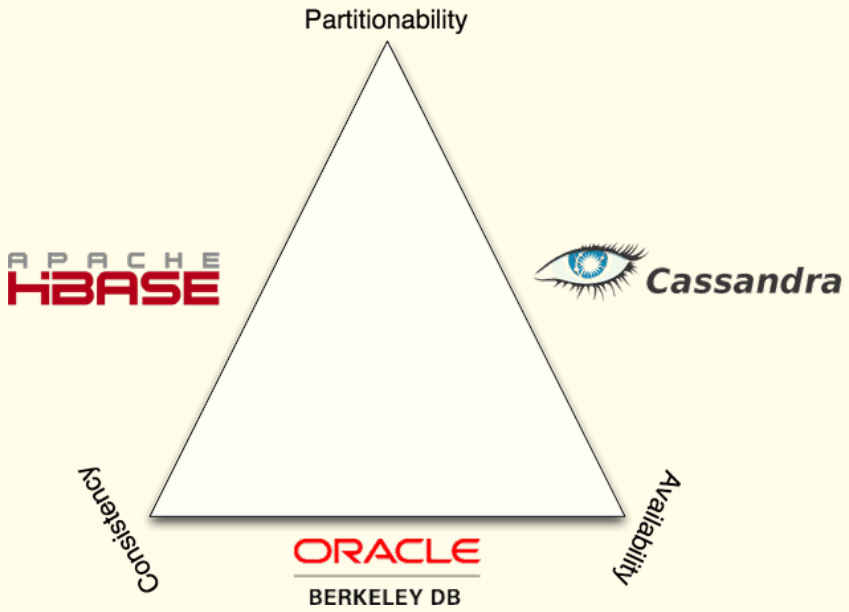


Hash based indexing using indexes



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

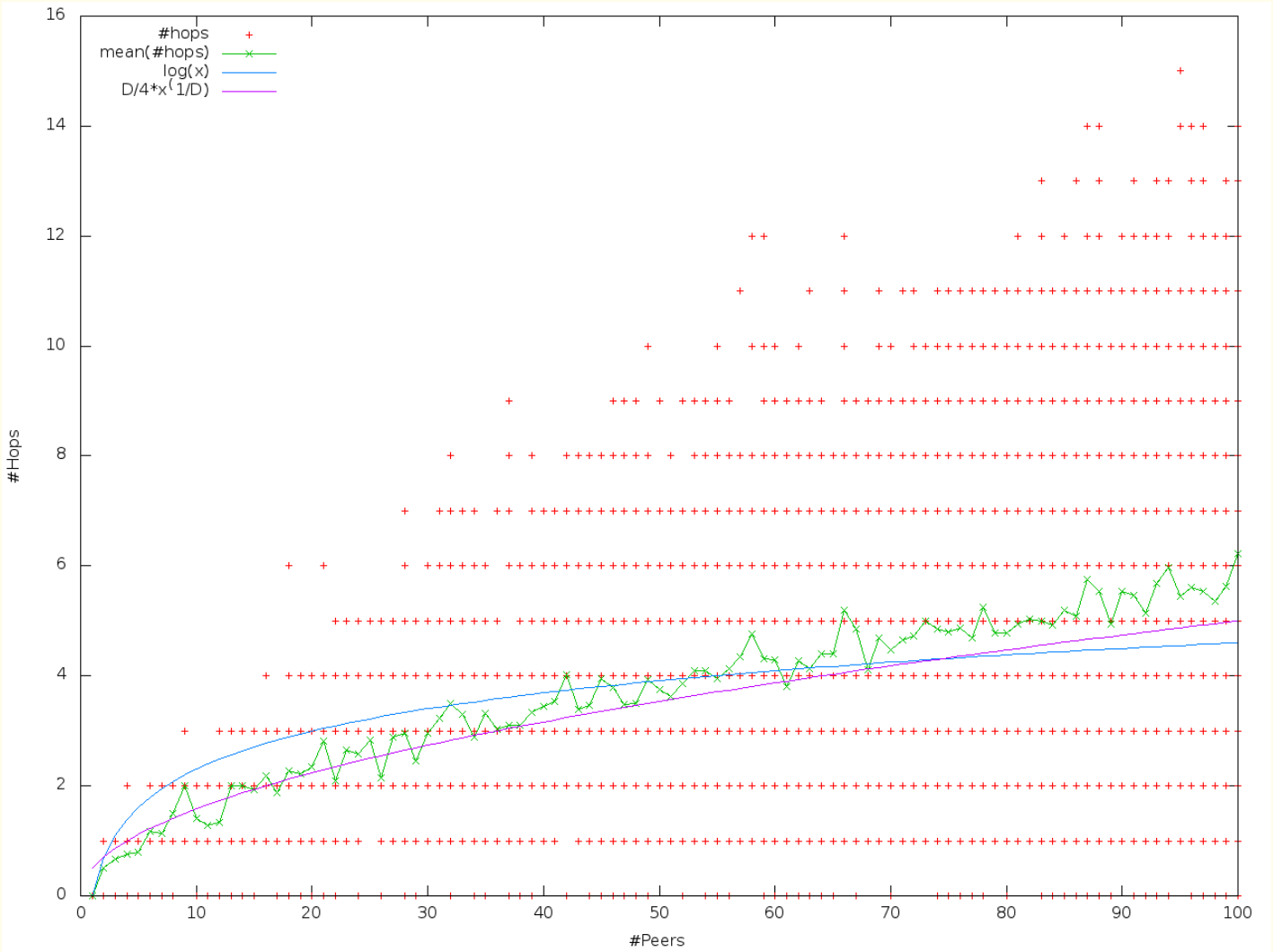
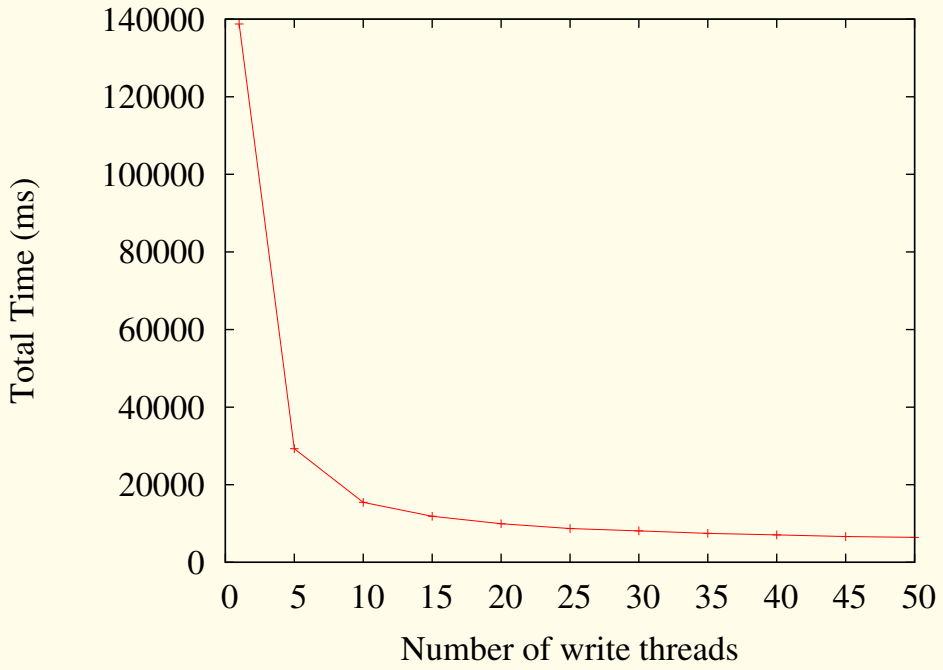


Figure: Unicast routing complexity comparison (made by Lokman Rahmani)

Evaluation concurrent insertions



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

CSMA – Indexing a subscription

1. Extract the first atomic[15] or range[16] 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

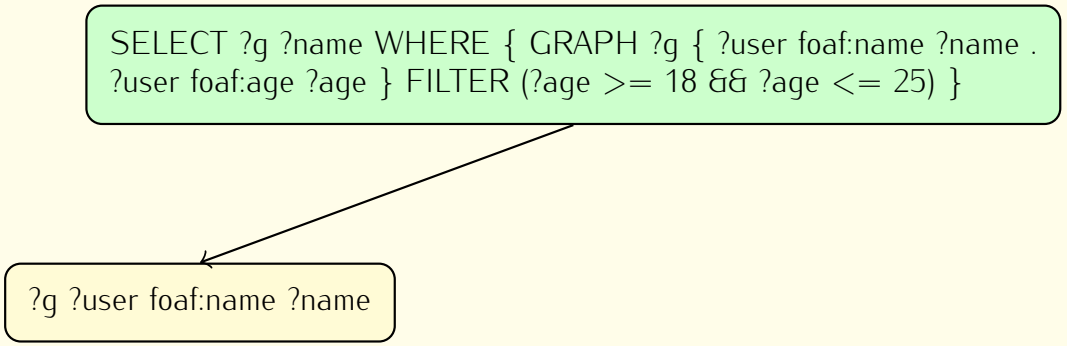
```
SELECT ?g ?name WHERE { GRAPH ?g { ?user foaf:name ?name .
?user foaf:age ?age } FILTER (?age >= 18 && ?age <= 25) }
```

[15] Quadruple where any RDF term may be a variable

[16] Atomic query where some conditions may be attached to variables

CSMA – Indexing a subscription

1. Extract the first atomic[15] or range[16] 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

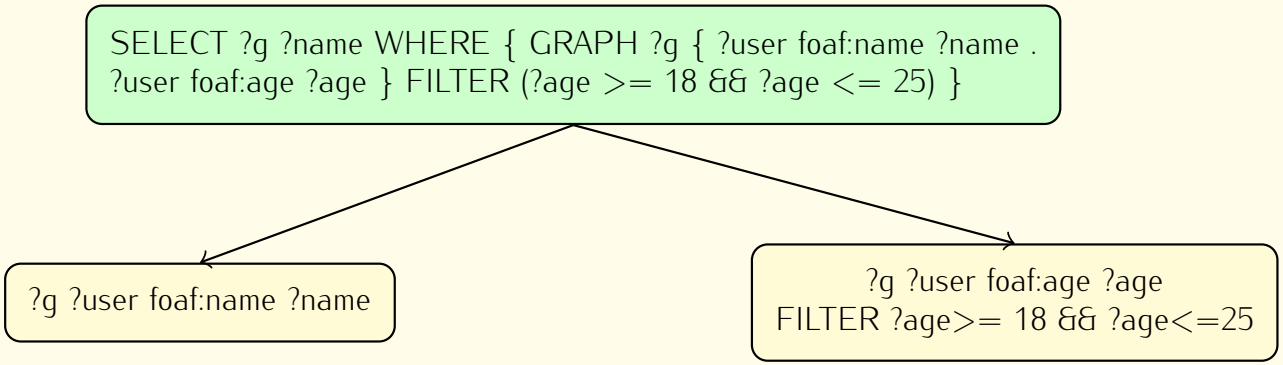


[15] Quadruple where any RDF term may be a variable

[16] Atomic query where some conditions may be attached to variables

CSMA – Indexing a subscription

1. Extract the first atomic[15] or range[16] 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

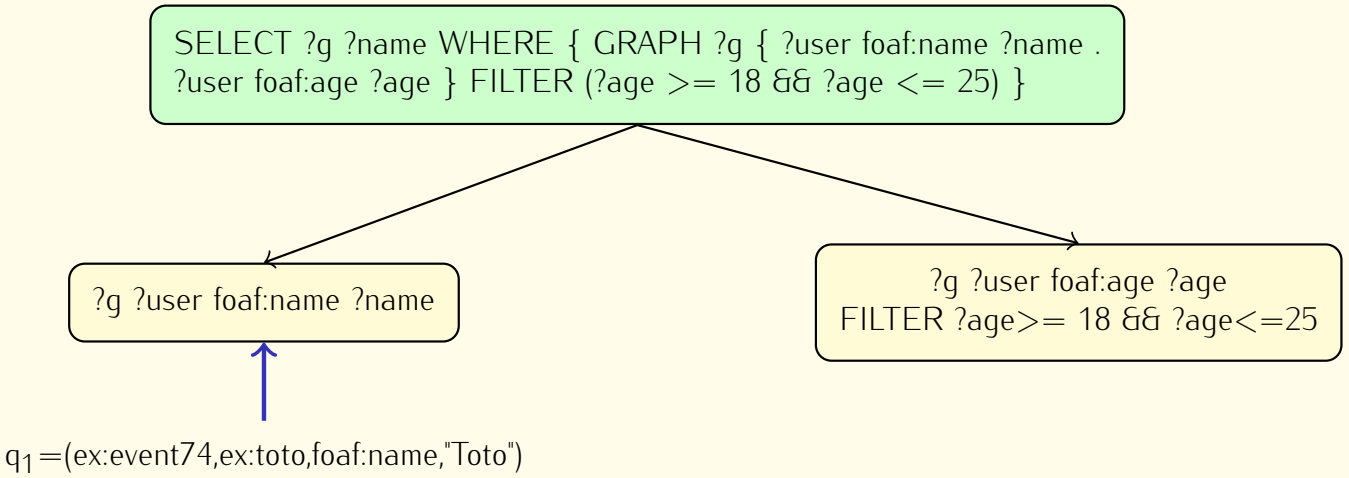


[15] Quadruple where any RDF term may be a variable

[16] Atomic query where some conditions may be attached to variables

CSMA – Indexing a subscription

1. Extract the first atomic[15] or range[16] 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

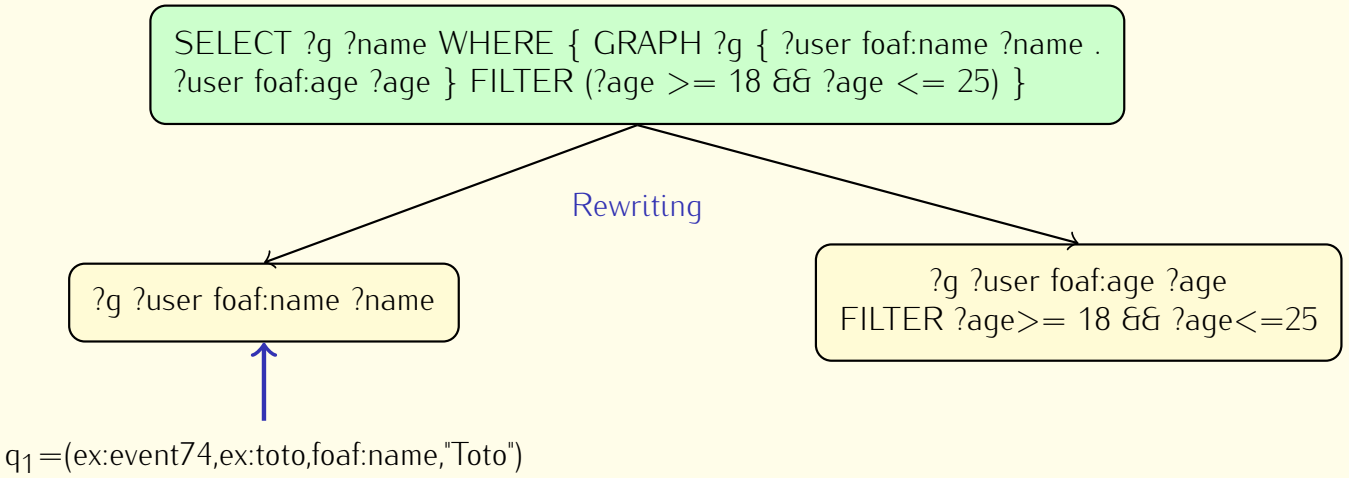


[15] Quadruple where any RDF term may be a variable

[16] Atomic query where some conditions may be attached to variables

CSMA – Indexing a subscription

1. Extract the first atomic[15] or range[16] 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

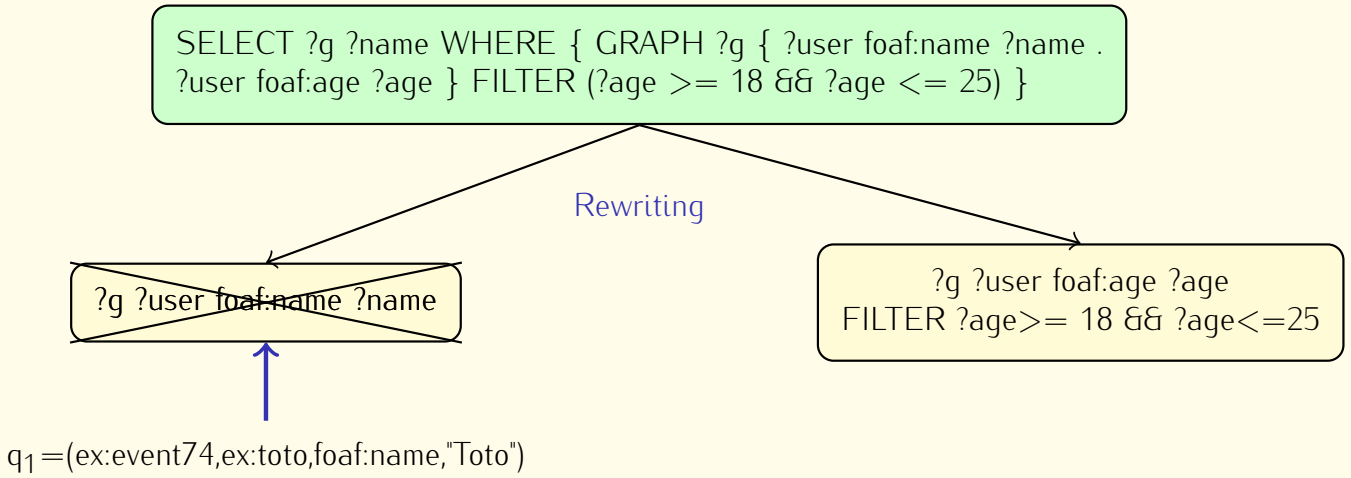


[15] Quadruple where any RDF term may be a variable

[16] Atomic query where some conditions may be attached to variables

CSMA – Indexing a subscription

1. Extract the first atomic[15] or range[16] 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

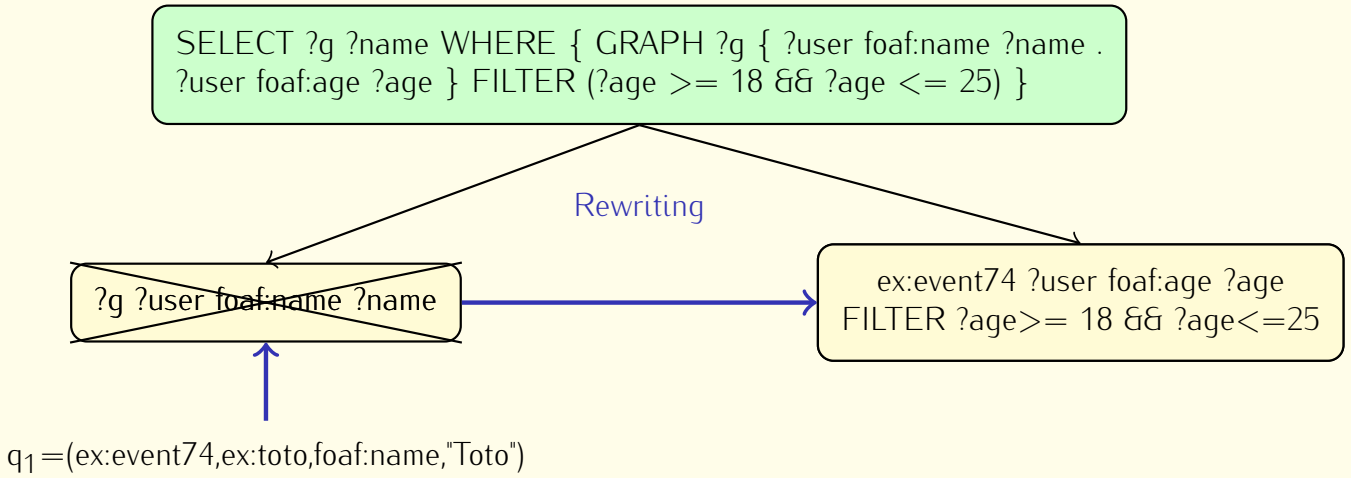


[15] Quadruple where any RDF term may be a variable

[16] Atomic query where some conditions may be attached to variables

CSMA – Indexing a subscription

1. Extract the first atomic[15] or range[16] 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

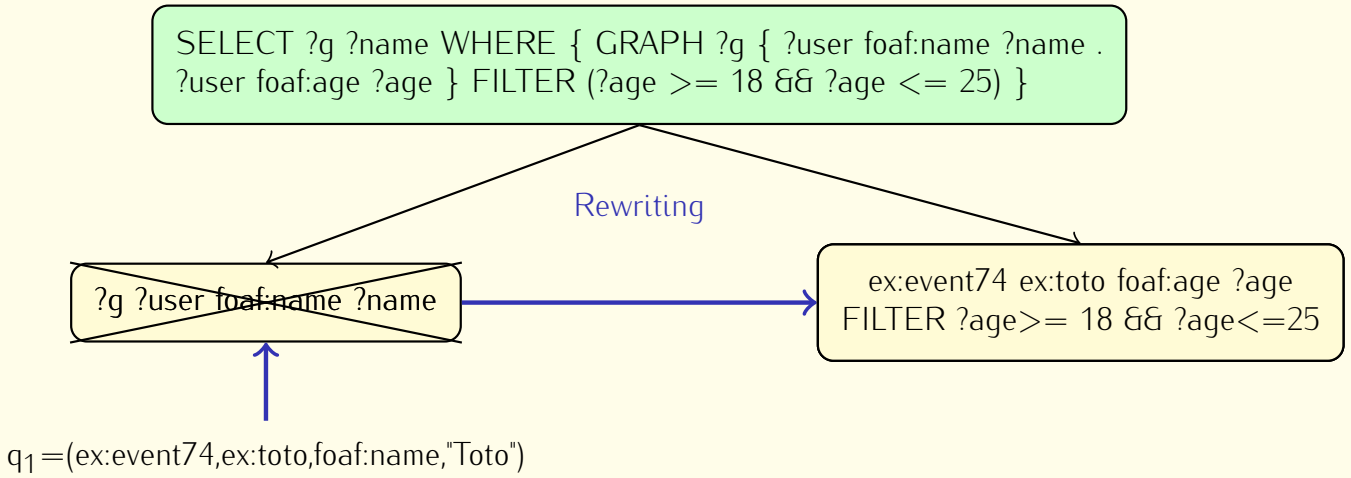


[15] Quadruple where any RDF term may be a variable

[16] Atomic query where some conditions may be attached to variables

CSMA – Indexing a subscription

1. Extract the first atomic[15] or range[16] 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

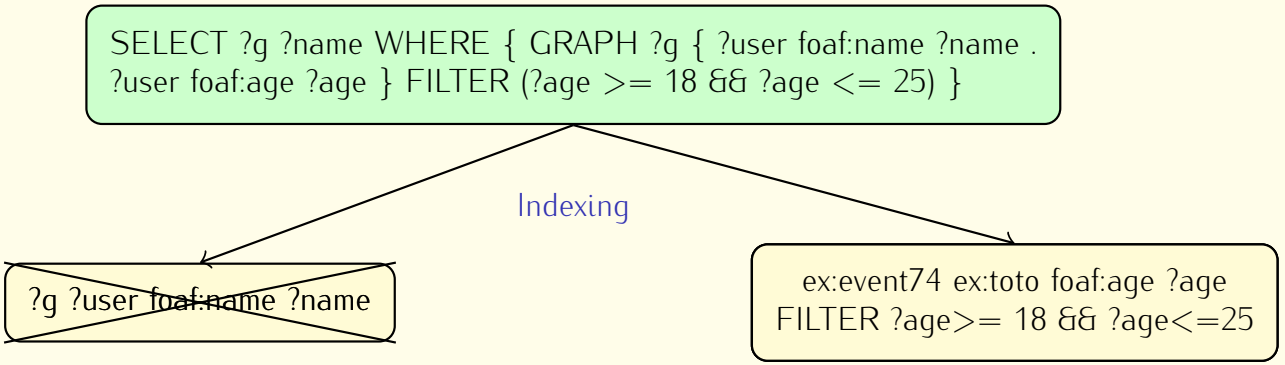


[15] Quadruple where any RDF term may be a variable

[16] Atomic query where some conditions may be attached to variables

CSMA – Indexing a subscription

1. Extract the first atomic[15] or range[16] 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

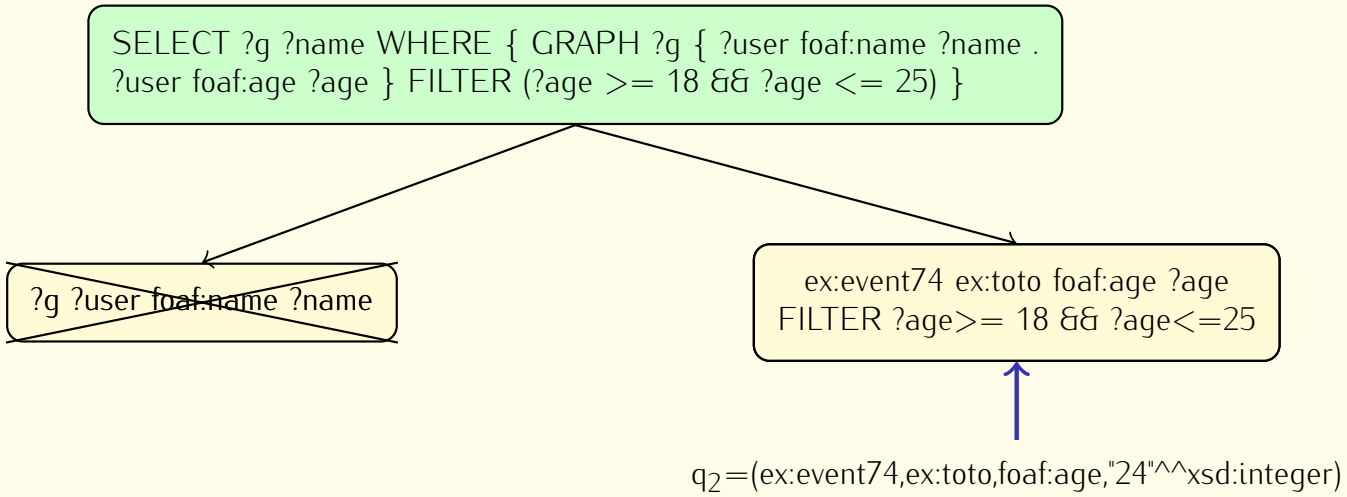


[15] Quadruple where any RDF term may be a variable

[16] Atomic query where some conditions may be attached to variables

CSMA – Indexing a subscription

1. Extract the first atomic[15] or range[16] 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

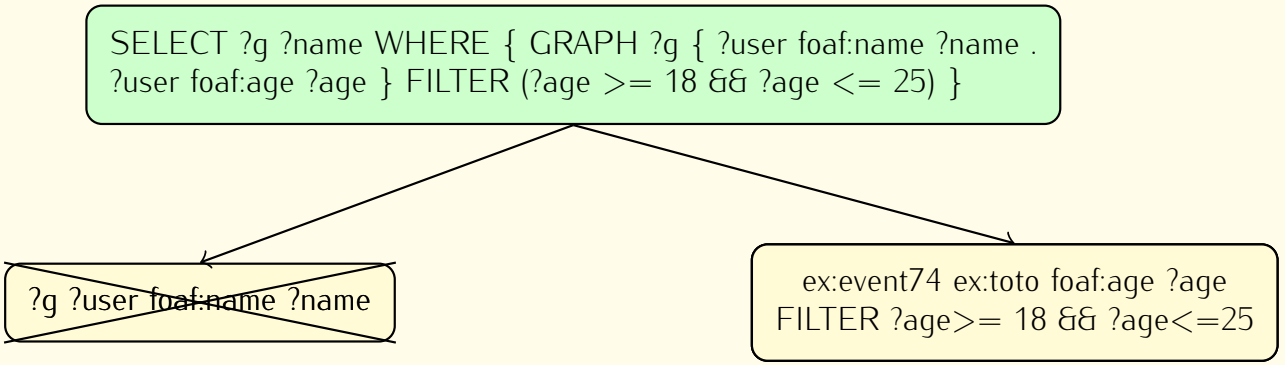


[15] Quadruple where any RDF term may be a variable

[16] Atomic query where some conditions may be attached to variables

CSMA – Indexing a subscription

1. Extract the first atomic[15] or range[16] 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

[15] Quadruple where any RDF term may be a variable

[16] 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

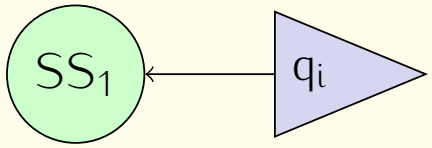
Notification reception

- 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



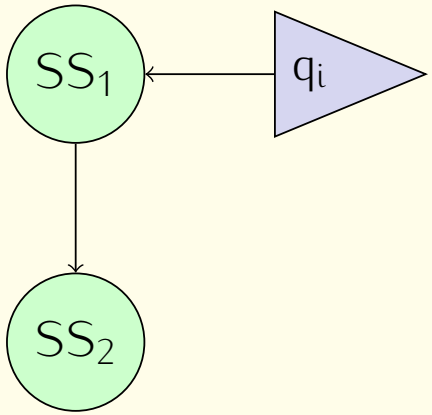
Notification reception

- 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



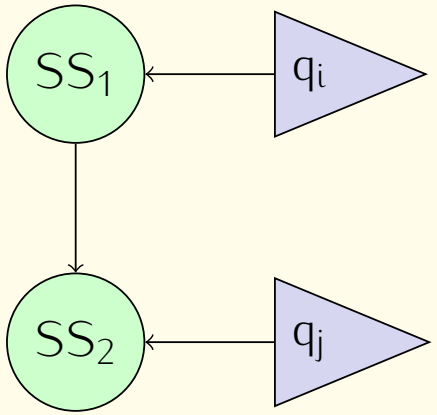
Notification reception

- 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



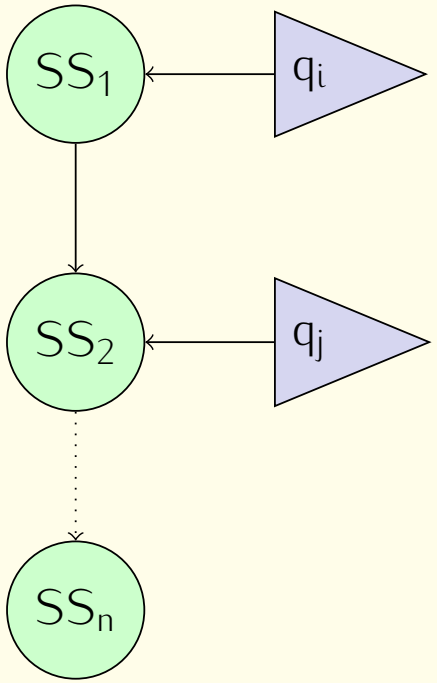
Notification reception

- 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



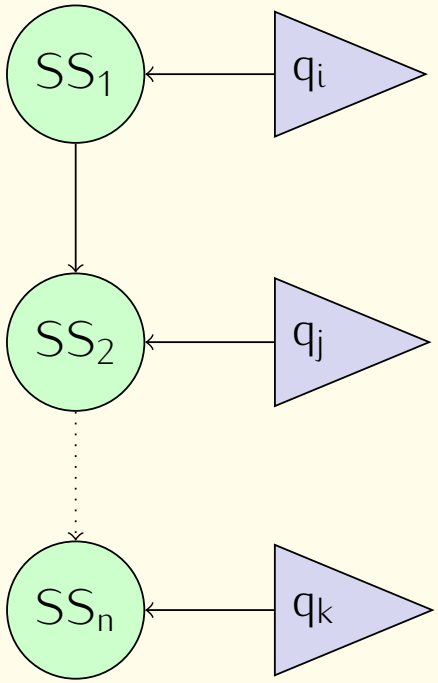
Notification reception

- 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



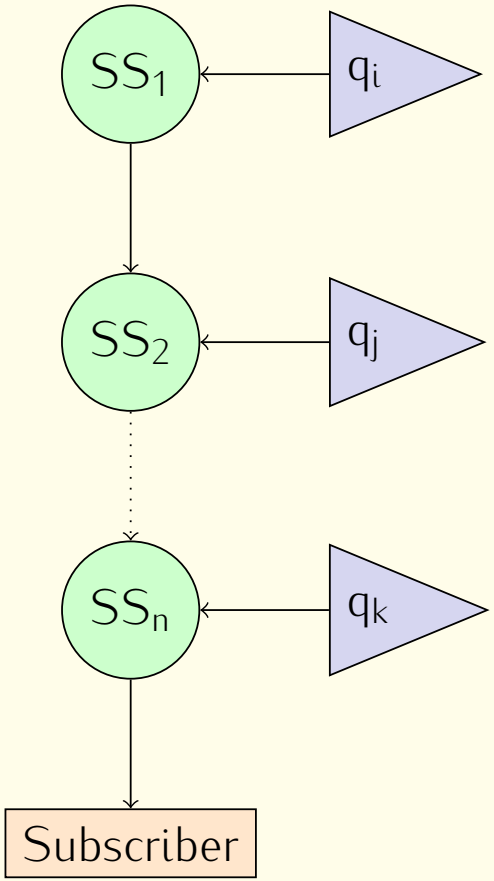
Notification reception

- 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



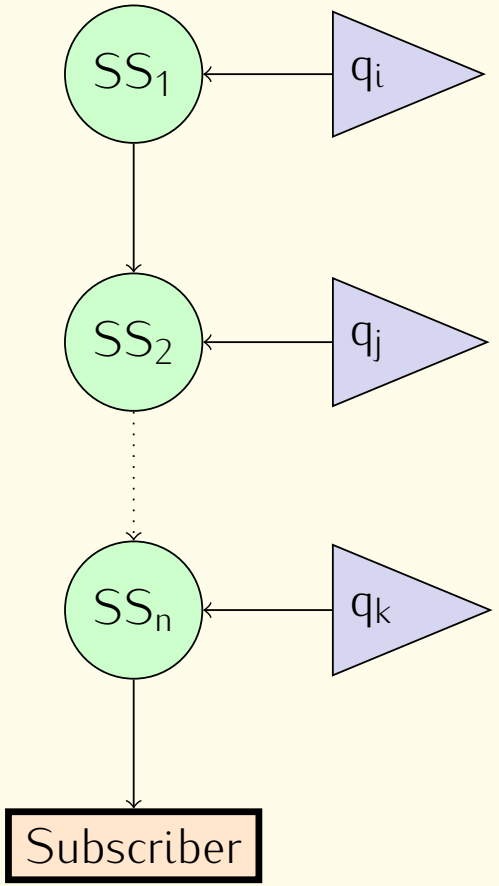
Notification reception

- 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

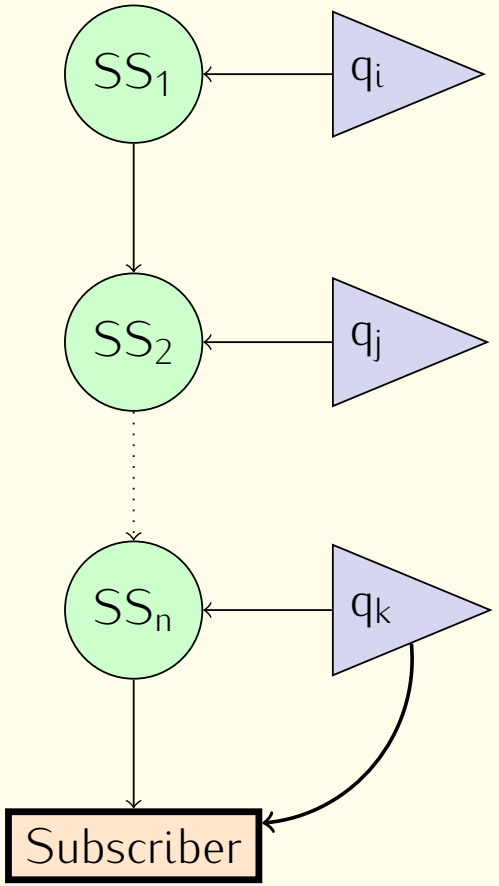
CSMA

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



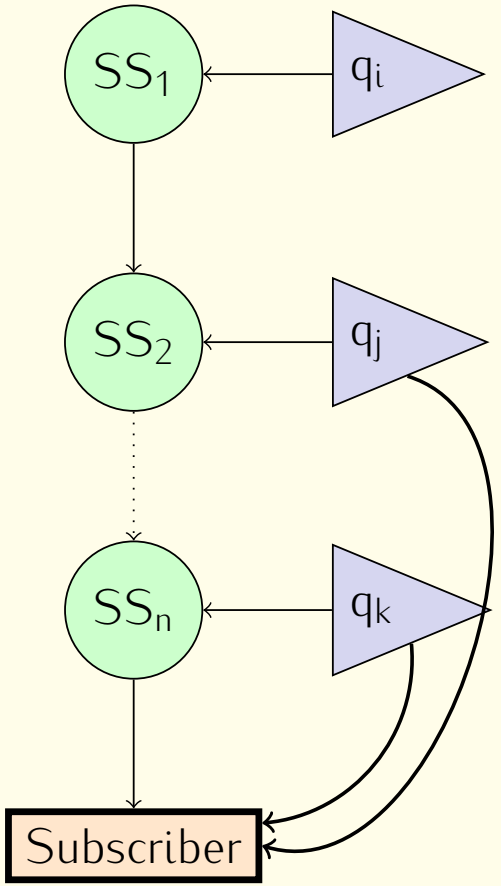
CSMA

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



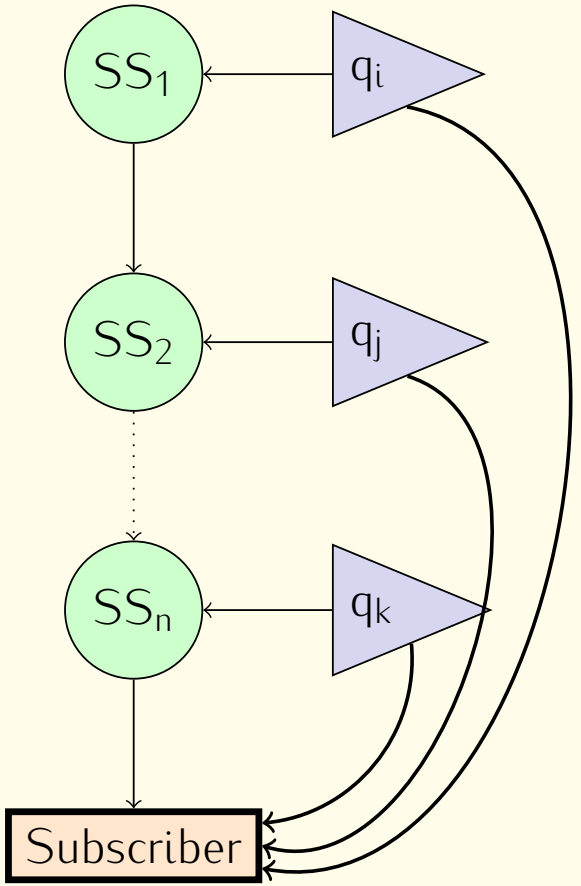
CSMA

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



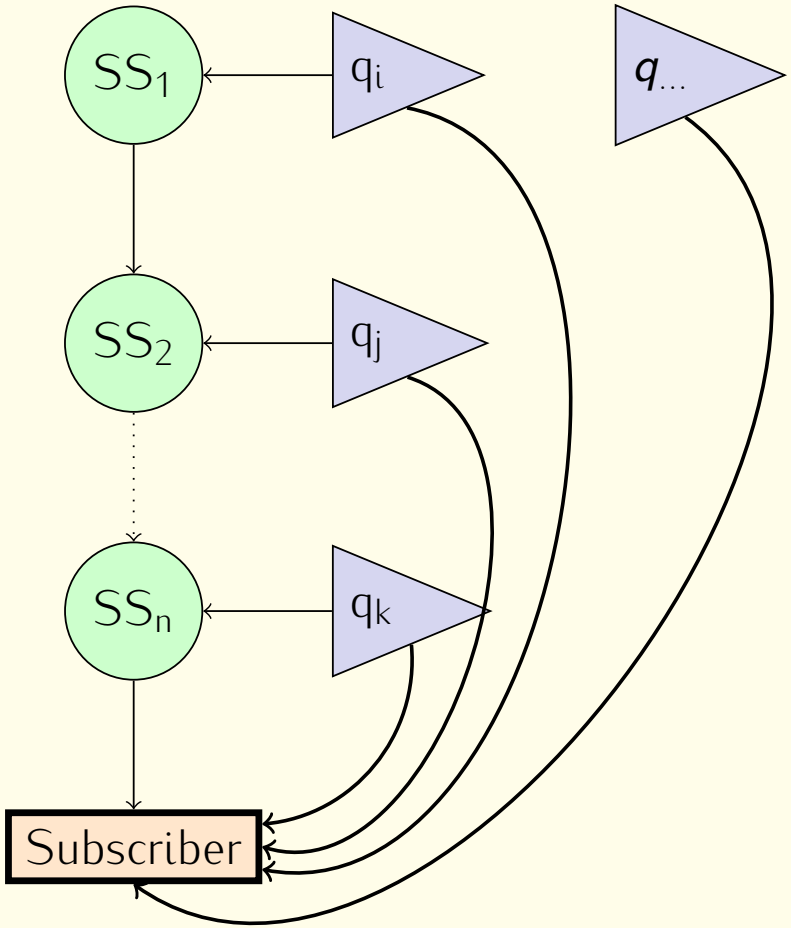
CSMA

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

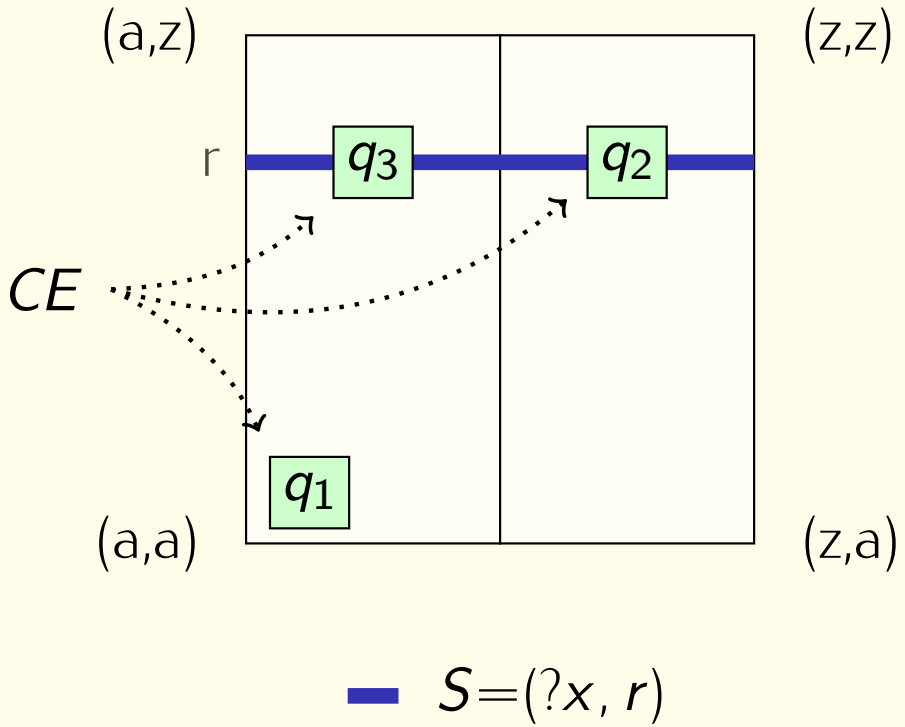
- Same as CSMA

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

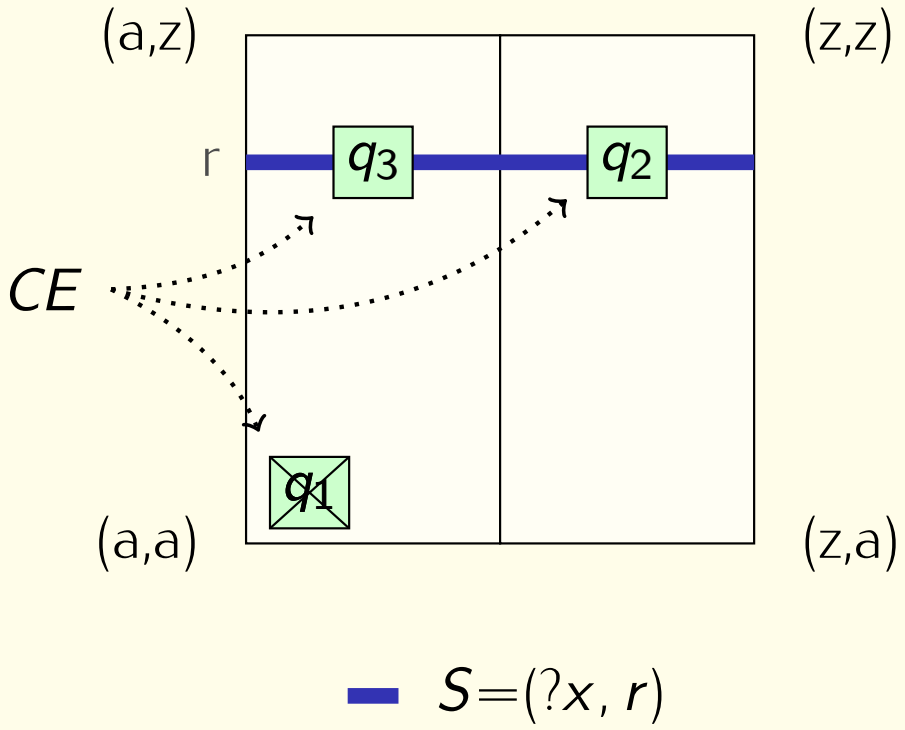
Avoiding duplicates with OSMA

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



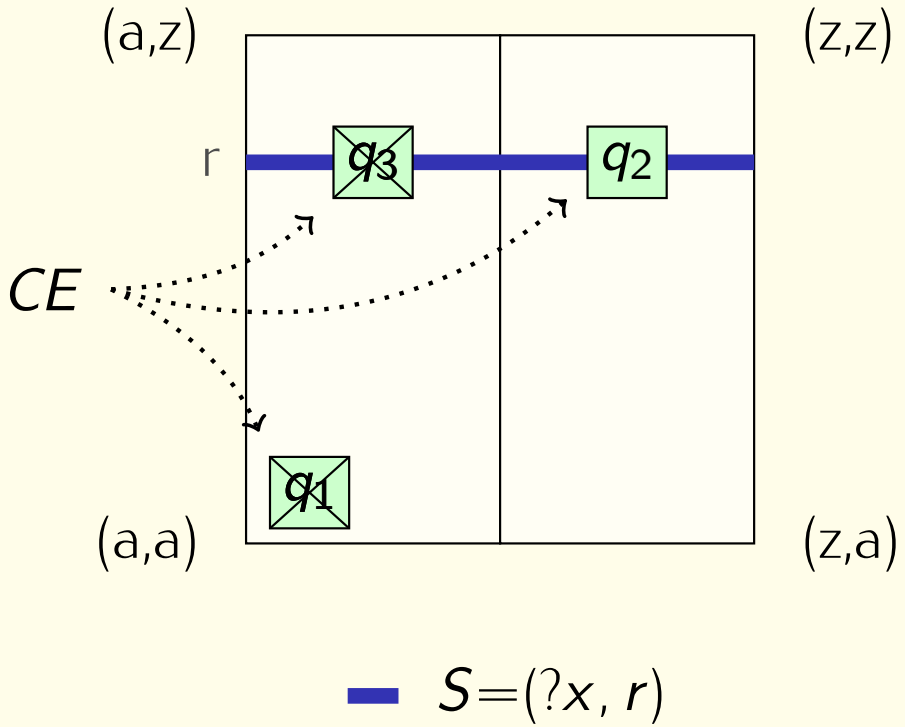
Avoiding duplicates with OSMA

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



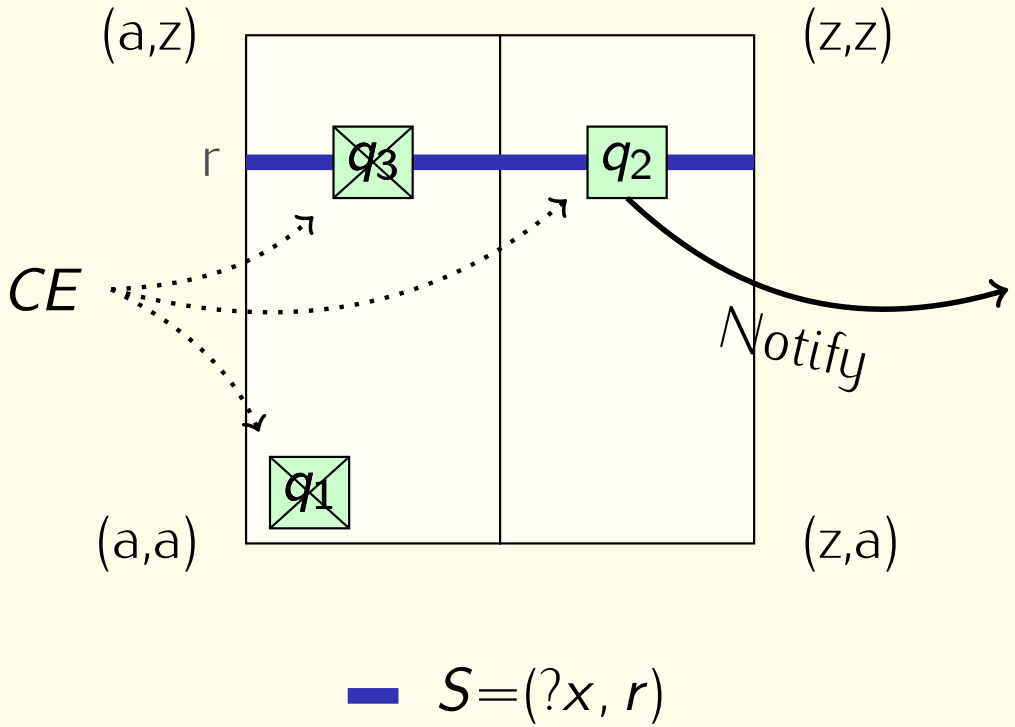
Avoiding duplicates with OSMA

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

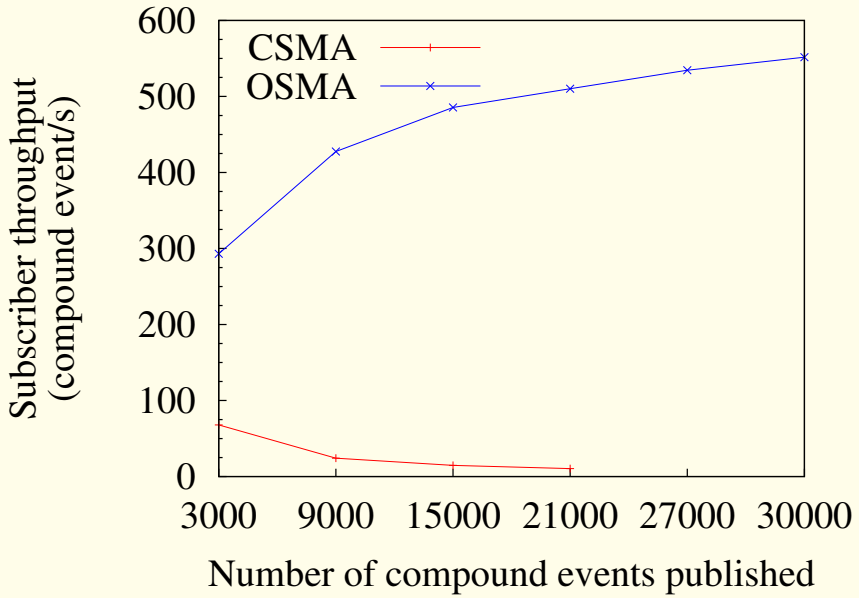


Avoiding duplicates with OSMA

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

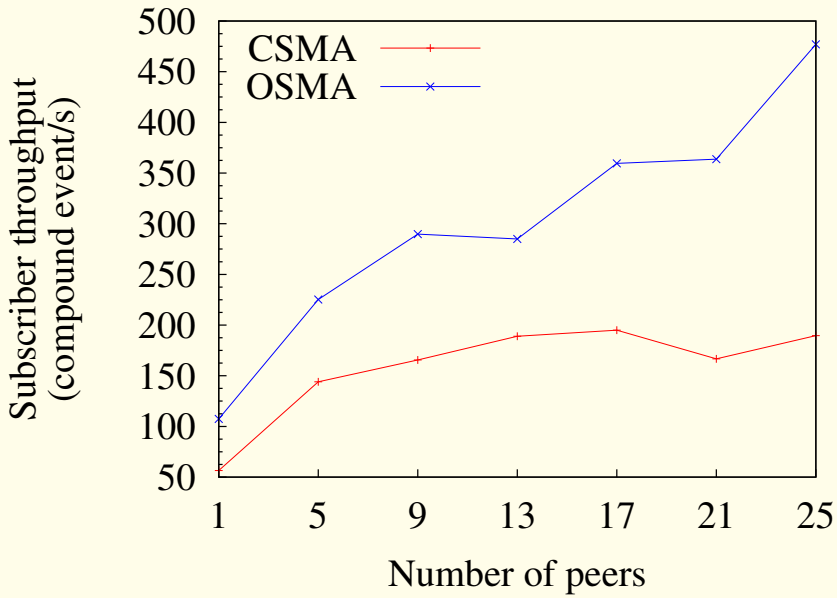


Impact of the number of publications



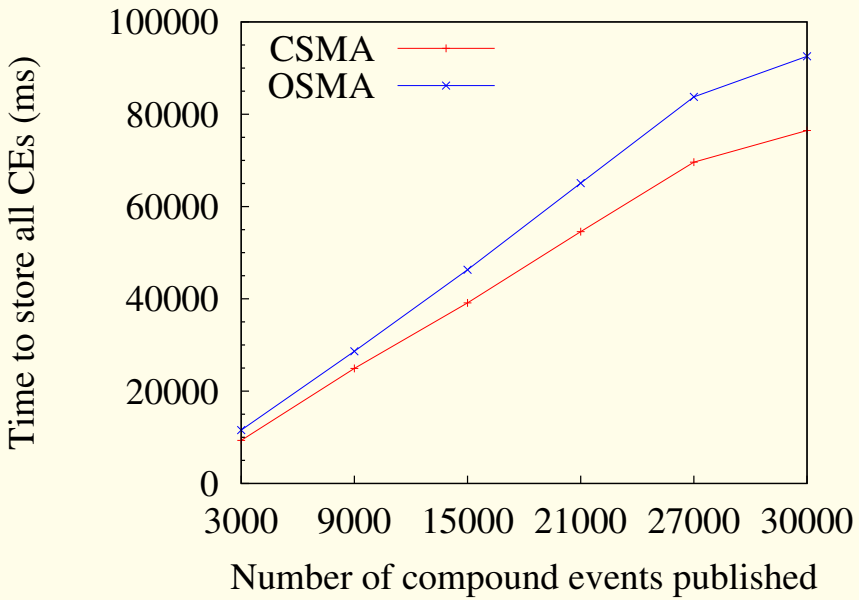
CSMA throughput decreases quickly with the number of publications due to required reconstructions.

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[17] 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

[17] 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[17] 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

[17] 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

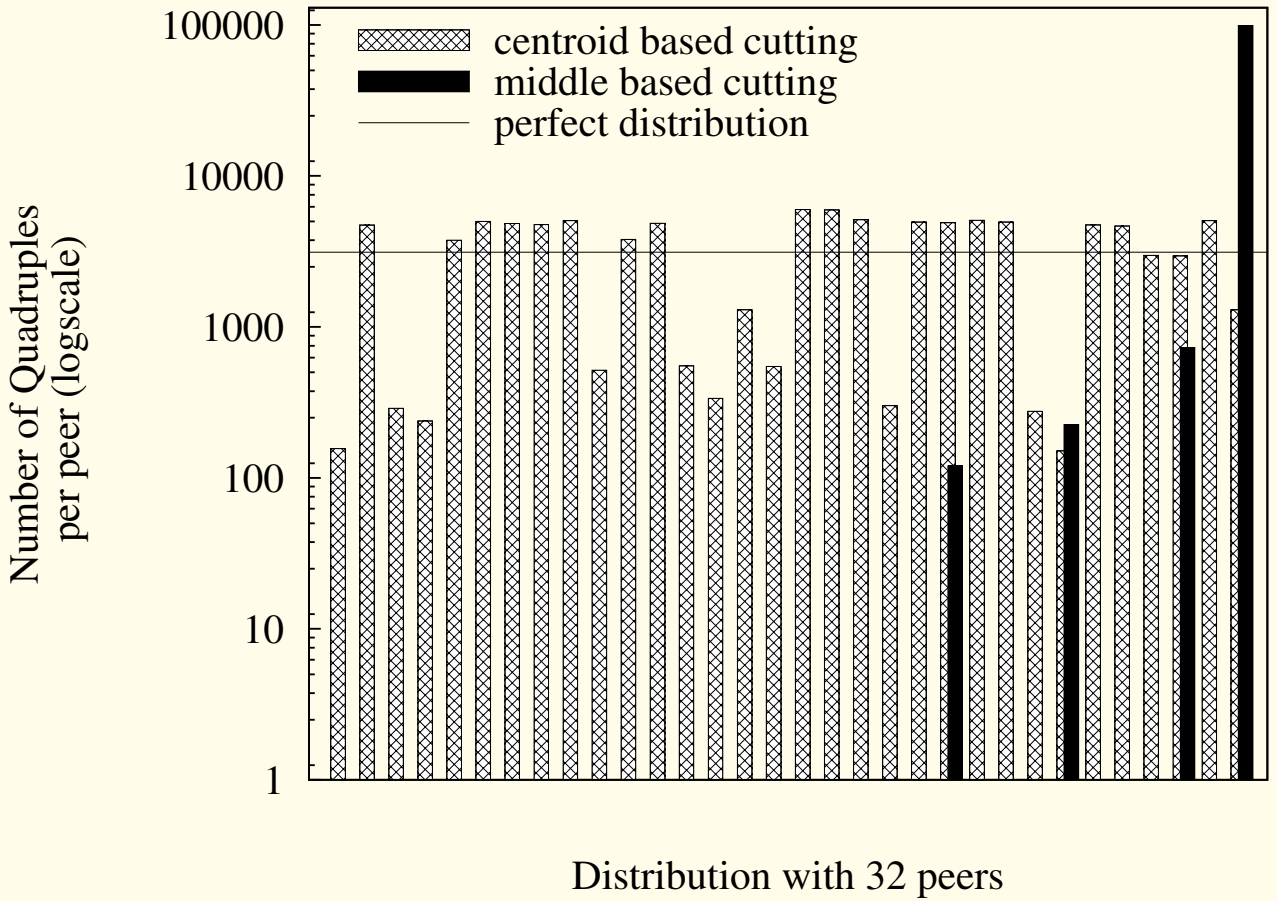
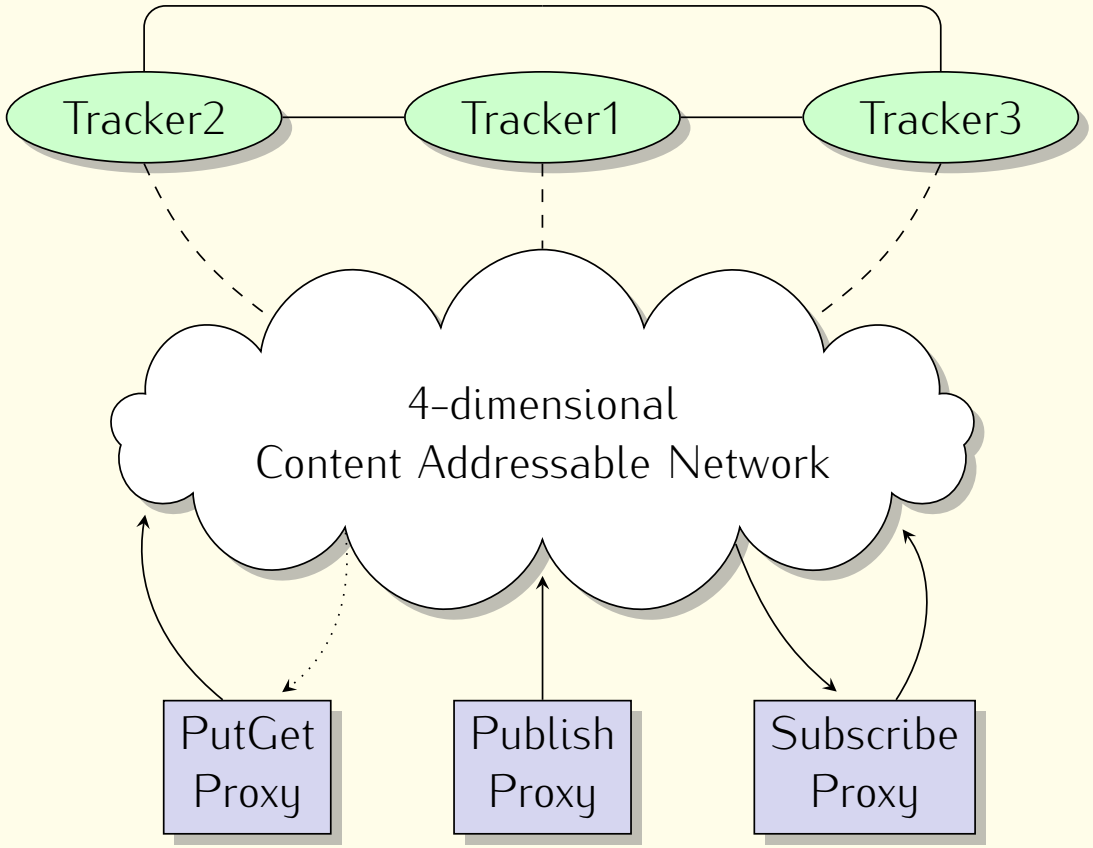
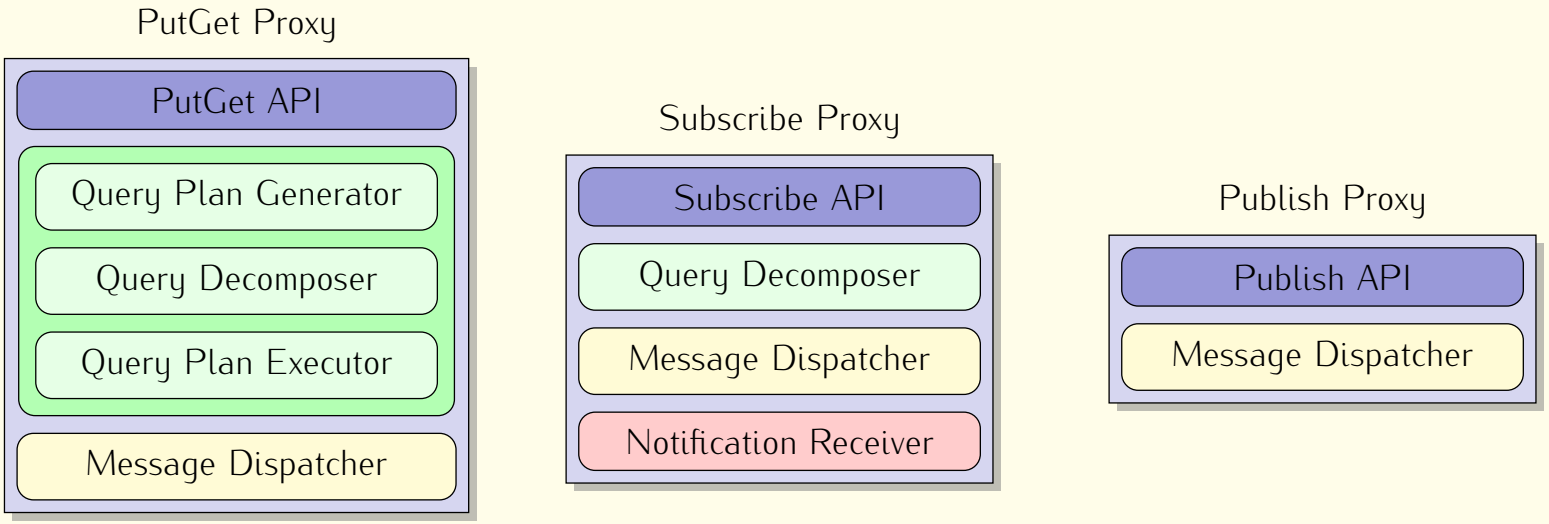


Figure: Static load balancing using middle vs centroid partitioning.

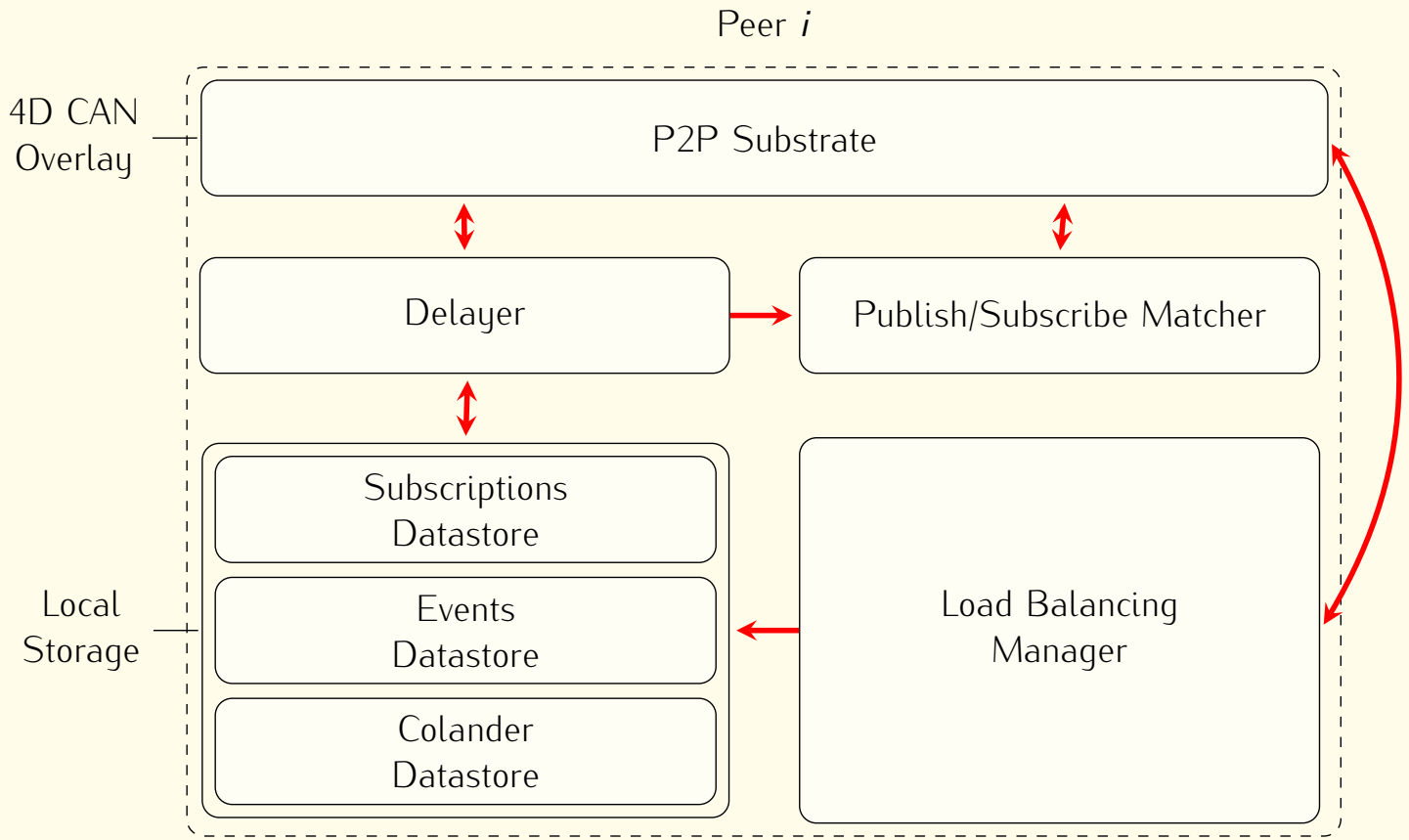
High-level view of the EventCloud architecture



Internal proxies architecture



Internal peer architecture



Priorities effect on reconstructions with CSMA