CyCLaDEs: A Decentralized Cache for Triple Pattern Fragments

Pauline Folz, Hala Skaf-Molli & Pascal Molli

ESWC 2016 - May 2016 - Heraklion

UNIVERSITÉ DE NANTES

lina

Nantes Métropole

Context

LDF clients execute SPARQL queries over LDF servers [1]

- **Minimize server** side **processing** to simple **triple pattern fragment queries** (TPF)
- **Joins** executed on **clients** side

Linked Data Fragments

[1] R. Verborgh and al. Querying Datasets on the Web with High Availability. In 1*3th International Semantic Web Conference* (ISWC 2014), pages 180-196, 2014.

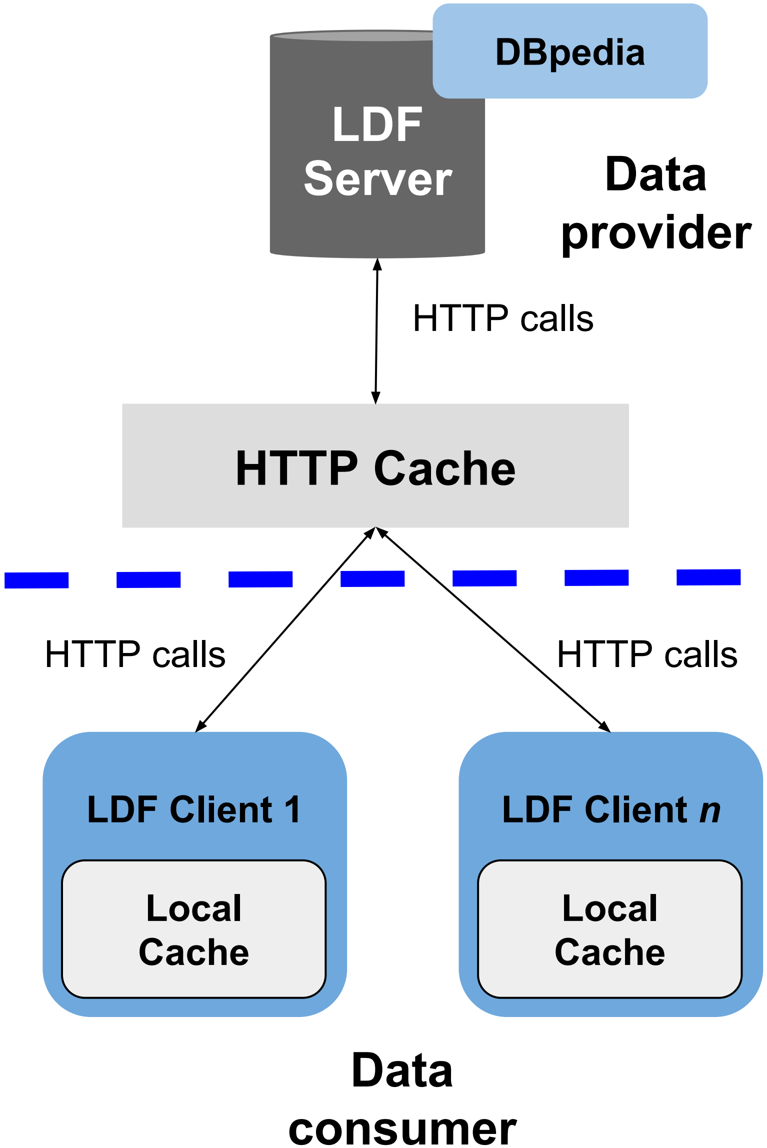
Context

Caches play an important role in the **performances** of LDF server:

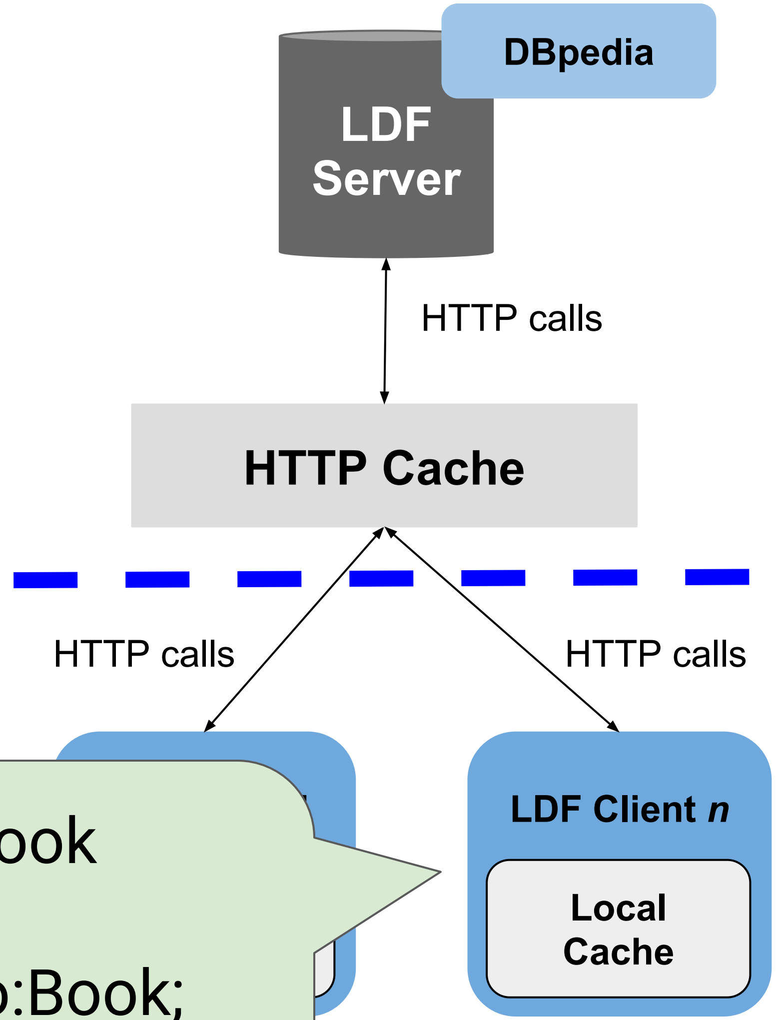
- Caches contain TPF, TPF are more likely to be reused locally and across clients

[1] R. Verborgh and al. Querying Datasets on the Web with High Availability. In 1*3th International Semantic Web Conference* (ISWC 2014), pages 180-196, 2014.

Triple Pattern Fragment & Caches



Triple Pattern Fragment & Caches



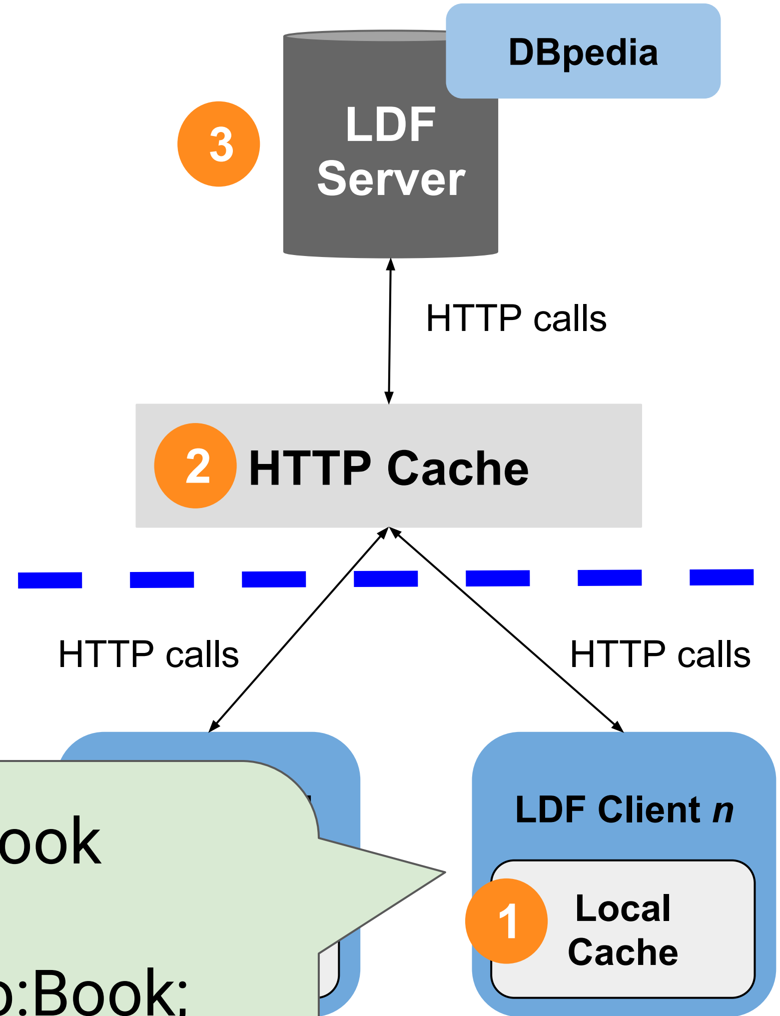
SELECT ?author ?book
WHERE {
?book rdf:label dbo:Book;
dbo:author ?author }
LIMIT 100

Triple Pattern Fragment & Caches

152 HTTP calls

0	?book **http://www.w3.org/1999/02/22-rdf-syntaxns#type** http:.../ontology/Book
1	?book **http:.../ontology/author** ?author
2	http:.../resource/%22...And Ladies of the Club%22 **http:.../ontology/author** ?author
3	http:.../resource/%22 **http:.../ontology/autl** ?author
4	http:.../resource/%22 Burglar **http:.../ontology/auth**

SELECT ?author ?book
WHERE {
?book rdf:label dbo:Book;
dbo:author ?author }
LIMIT 100



What happens if clients collaborate?

United Federation of Data Consumers

What if clients collaborate?

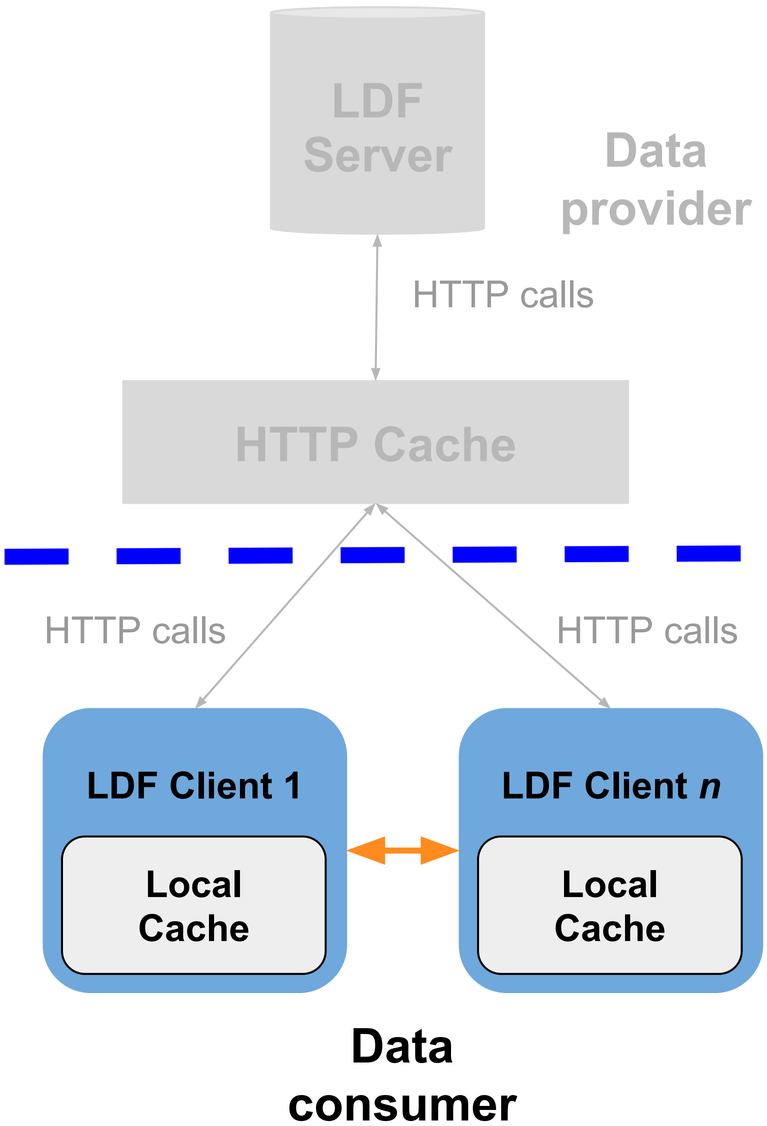
A local cache of a client can be **shared** with other clients

- Reduce the load on the server

Challenges:

- Network with 1 million of clients

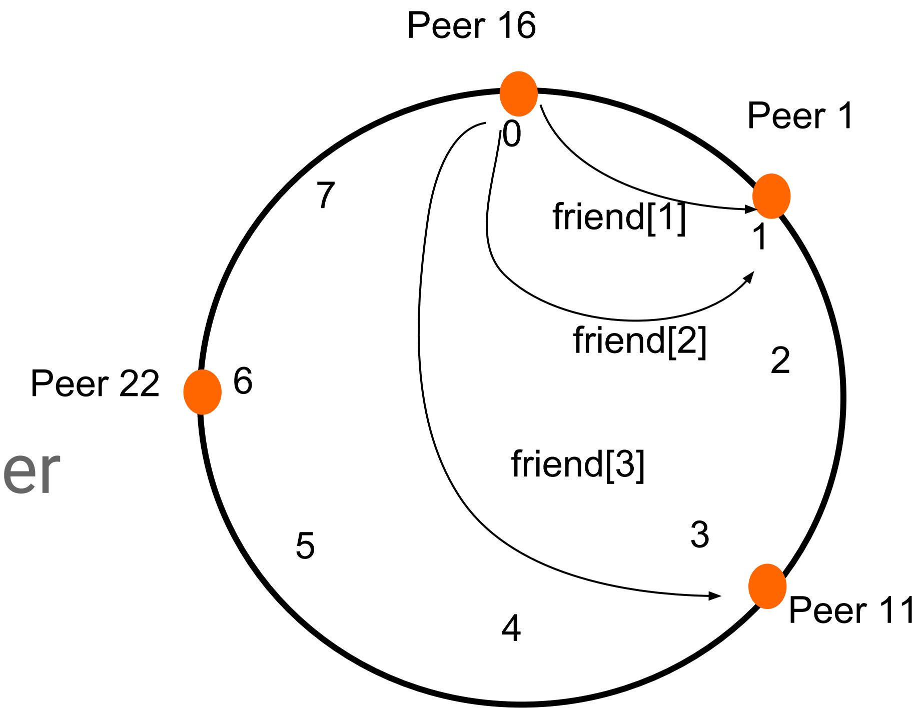
How a client can find a TPF quickly in another client cache?



Related Works

DHT: distributes the cache among participants [2]

- **+** Lookup(TPF): finds TPF if it exists!
- **-** 1 query → 20,000 calls → 20,000 *log(n)* hops, n: number of peers



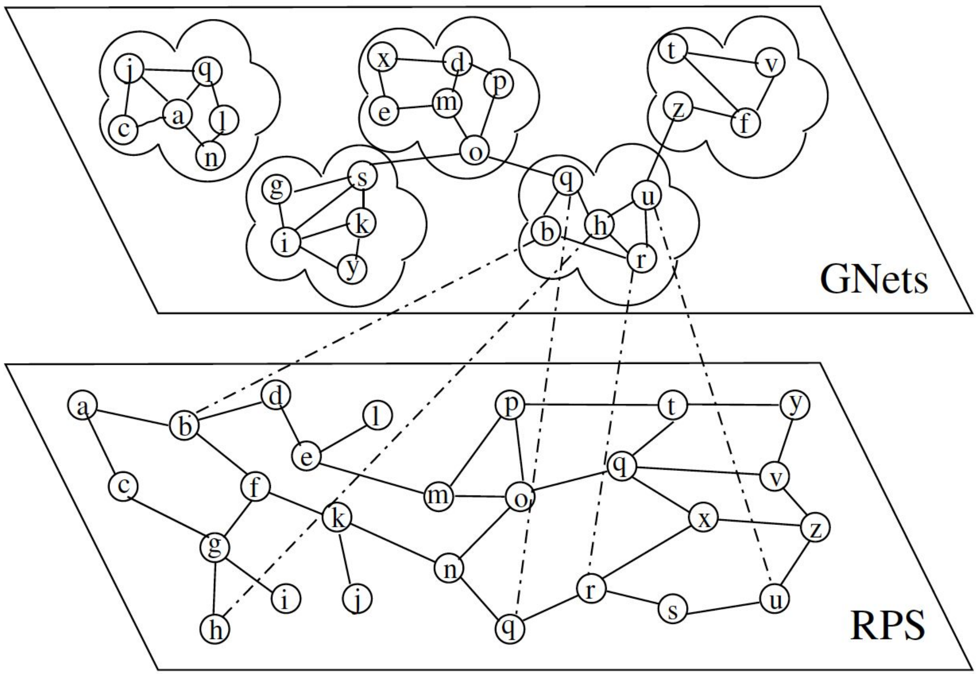
[2] M. El Dick and al. Flower-cdn: A hybrid p2p overlay for efficient query processing in cdn. *In Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, 2009. ACM.

Related Works

Behavioral Cache: connects a fixed size of similar nodes [3]

- + Zero-hop lookup latency
- - Not sure to find a TPF
- ~ Experimented with browsing histories

What if we experiment behavioral cache with queries?



[3] D. Frey and al. Behave: Behavioral cache for web content. In Distributed Applications and Interoperable System, 2014.

CyCLaDEs Approach

Assumption: Clients which performed **similar queries** in the **past** will likely perform **similar queries** in the **future**

CyCLaDEs Approach

Assumption: Clients which performed **similar queries** in the **past** will likely perform **similar queries** in the **future**

Approach:

- Each client builds a **fixed-size** of **best** similar **neighbors** with **zero-hop latency**

CyCLaDEs Approach

Assumption: Clients which performed **similar queries** in the **past** will likely perform **similar queries** in the **future**

Approach:

- Each client builds a **fixed-size** of **best** similar **neighbors** with **zero-hop latency**
- For each call 1) check **local** cache 2) check **neighbors'** cache in parallel 3) go to the server

CyCLaDEs Approach

Assumption: Clients which performed **similar queries** in the **past** will likely perform **similar queries** in the **future**

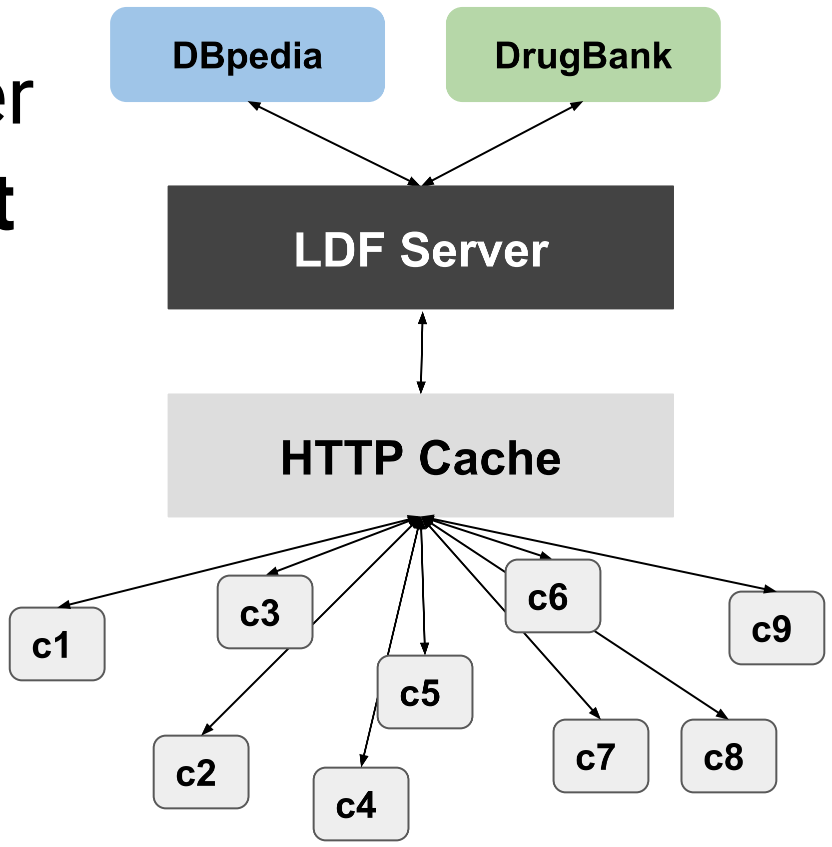
Approach:

- Each client builds a **fixed-size** of **best** similar **neighbors** with **zero-hop latency**
- For each call 1) check **local** cache 2) check **neighbors'** cache in parallel 3) go to the server

How many neighbors cache hits can we get?

LDF: Approach Overview

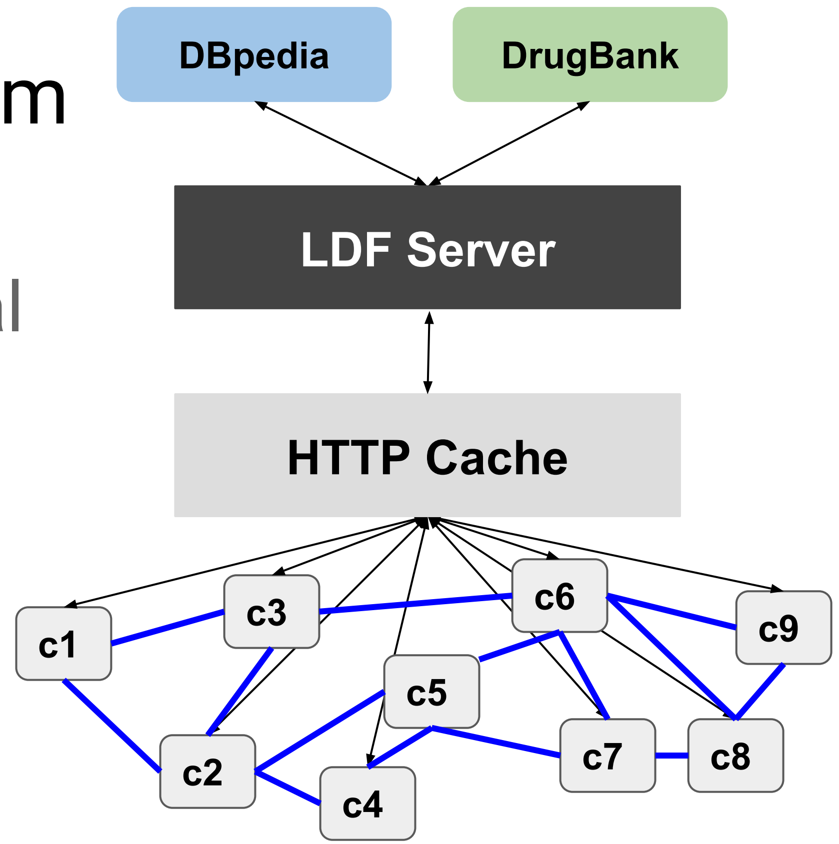
Many clients access LDF server concurrently, but **clients do not collaborate**.



CyCLaDEs: Approach Overview

Connect nodes through Random Peer Sampling (RPS):

- Each node maintains a partial view on the entire network
- The view contains a random subset of network nodes

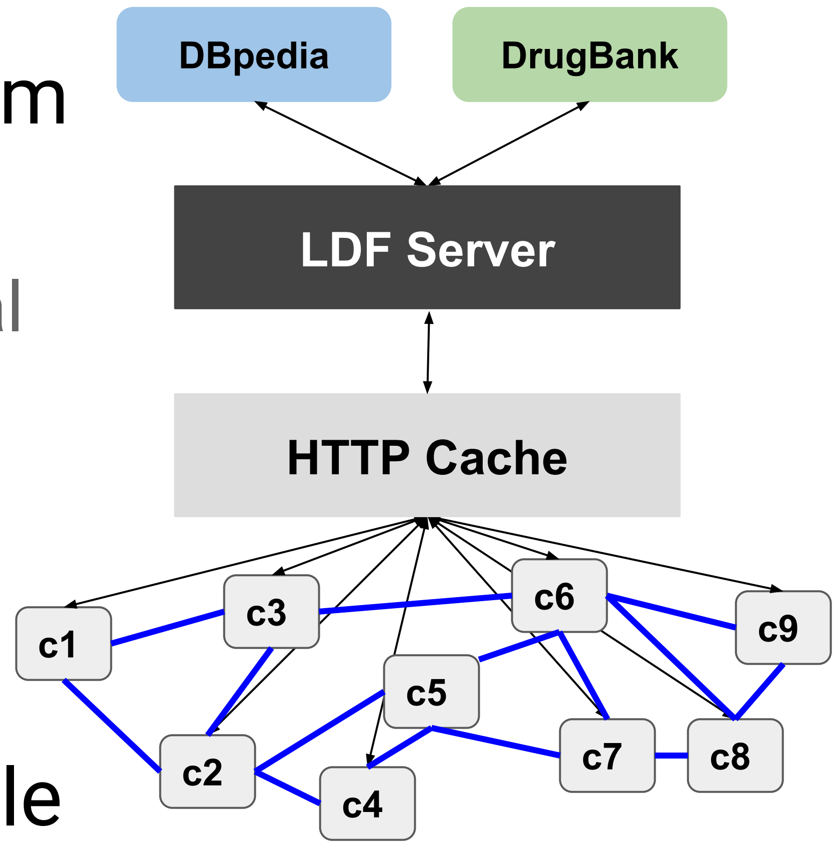


CyCLaDEs: Approach Overview

Connect nodes through Random Peer Sampling (RPS):

- Each node maintains a partial view on the entire network
- The view contains a random subset of network nodes

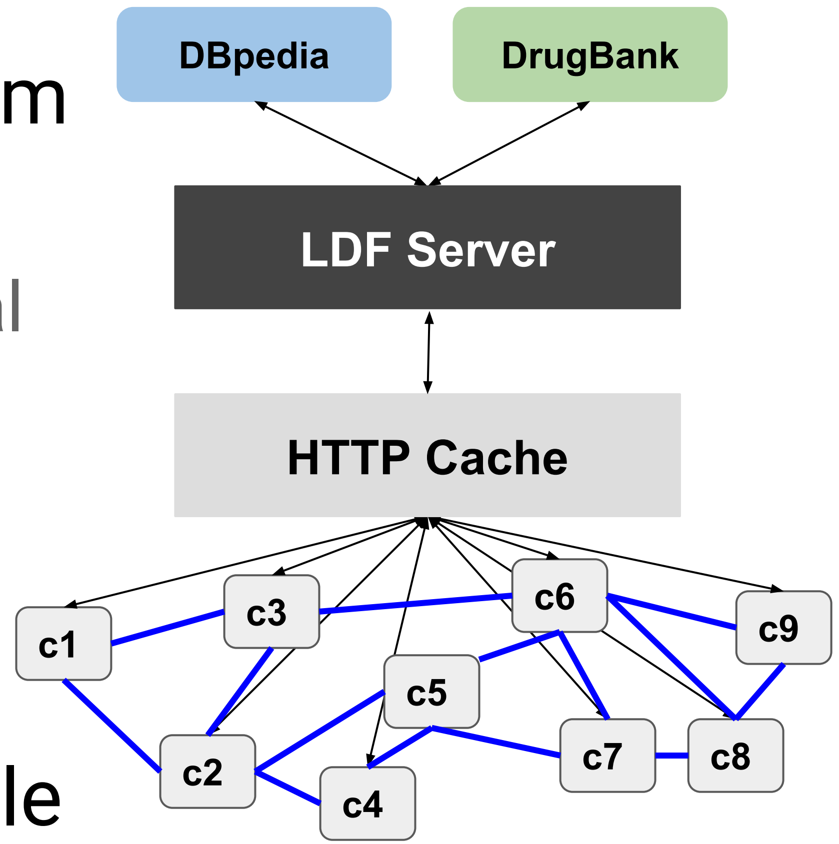
Shuffle phases: **renew periodically** neighbors to handle churn and avoid network partition.



CyCLaDEs: Approach Overview

Connect nodes through Random Peer Sampling (RPS):

- Each node maintains a partial view on the entire network
- The view contains a random subset of network nodes



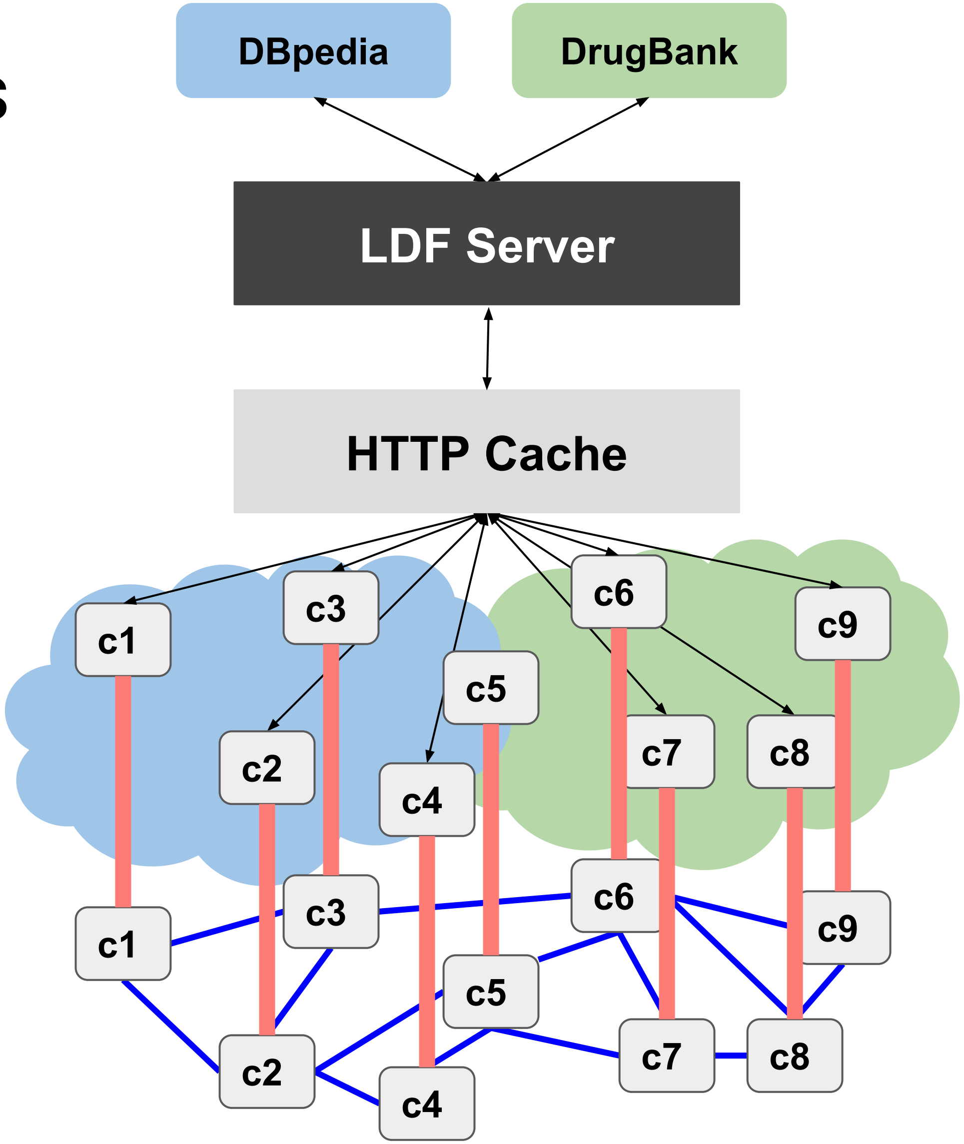
Shuffle phases: **renew periodically** neighbors to handle churn and avoid network partition.

→ We use Cyclon [4] for RPS.

[4] S. Voulgaris and al. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 2005.

CyCLaDEs: Approach Overview

RPS overlay network ensures connectivity among **all** clients.

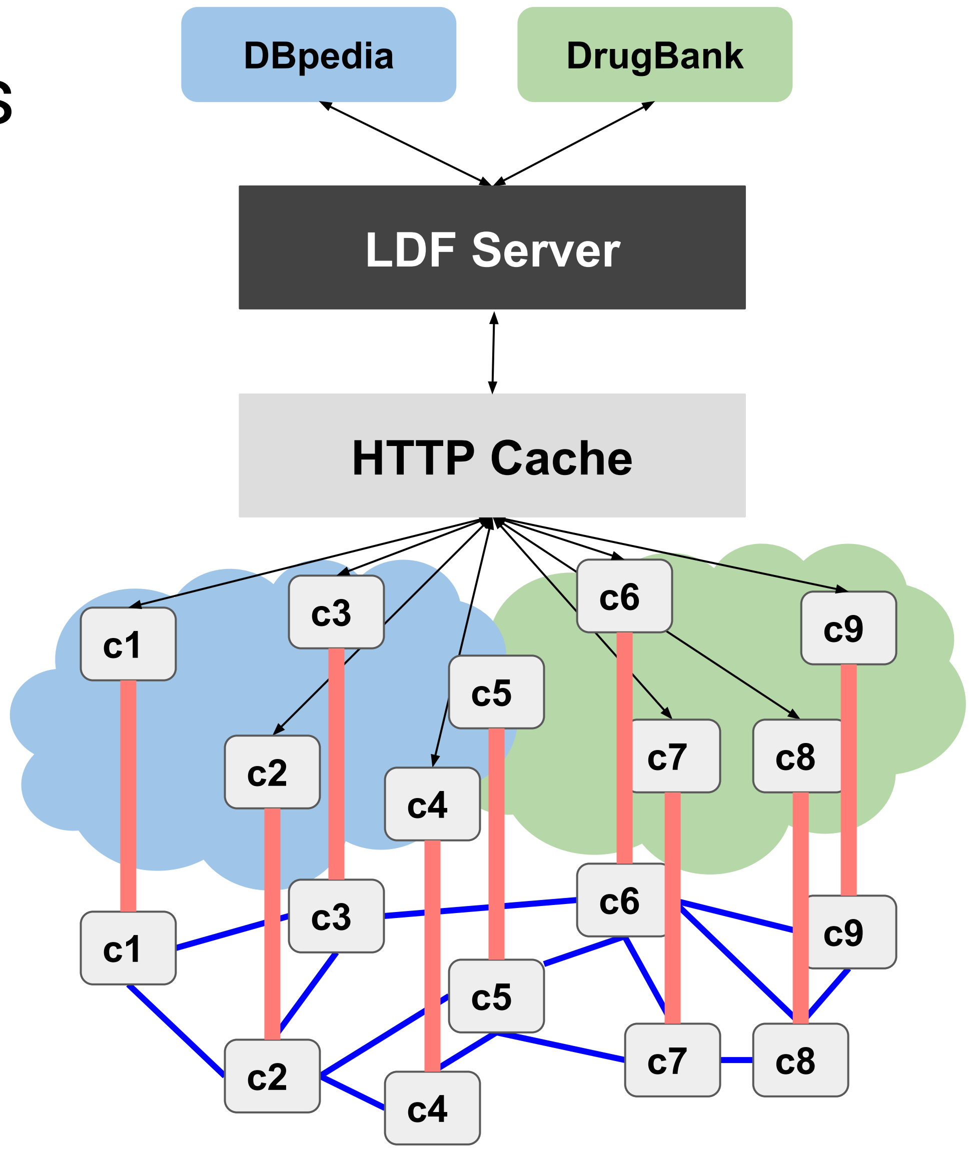


CyCLaDEs: Approach Overview

RPS overlay network ensures connectivity among **all** clients.

C6 is connected to C3:

- C6 → DBpedia
- C3 → DrugBank
- C6 is not **similar** to C3



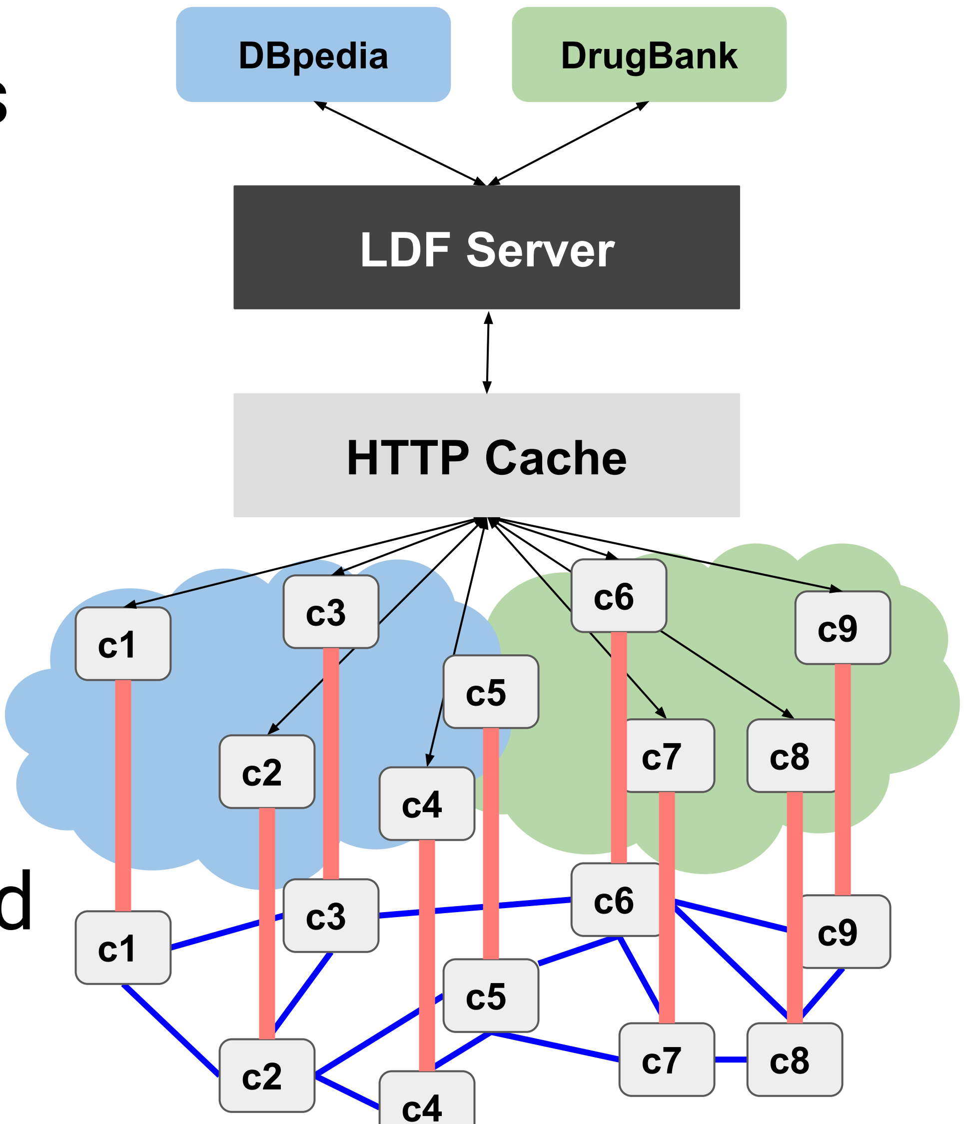
CyCLaDEs: Approach Overview

RPS overlay network ensures connectivity among **all** clients.

C6 is connected to C3:

- C6 → DBpedia
- C3 → DrugBank
- C6 is not **similar** to C3

Need a second overlay to handle **similarity** as proposed in Gossple [5].

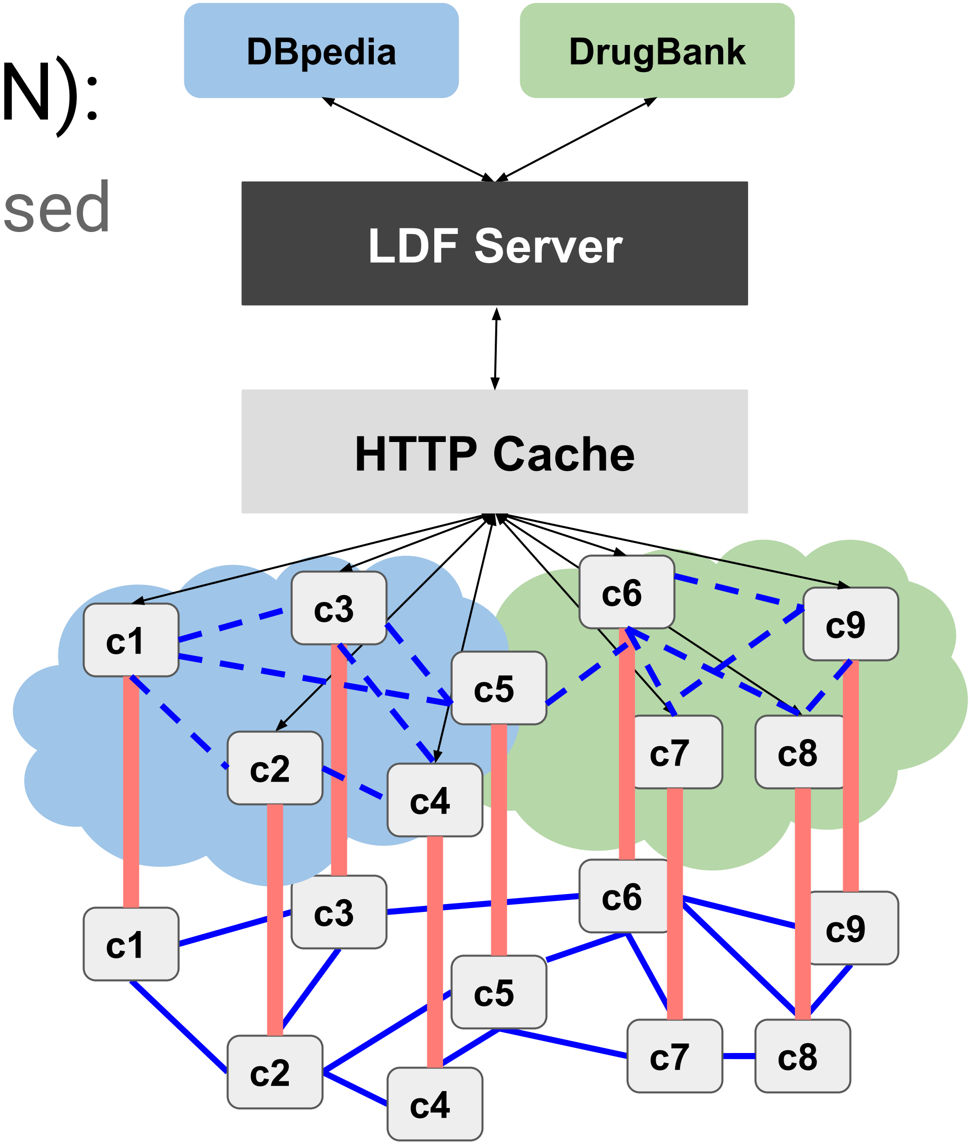


[5] M. Bertier and al. The gossple anonymous social network. In *11th International Middleware Conference Middleware 2010* - ACM/IFIP/USENIX, volume 6452 of LNCS, pages 191, 2010.

CyCLaDEs: Approach Overview

Cluster Overlay Network (CON):

- Each node has a **profile** based on the **history** of executed queries



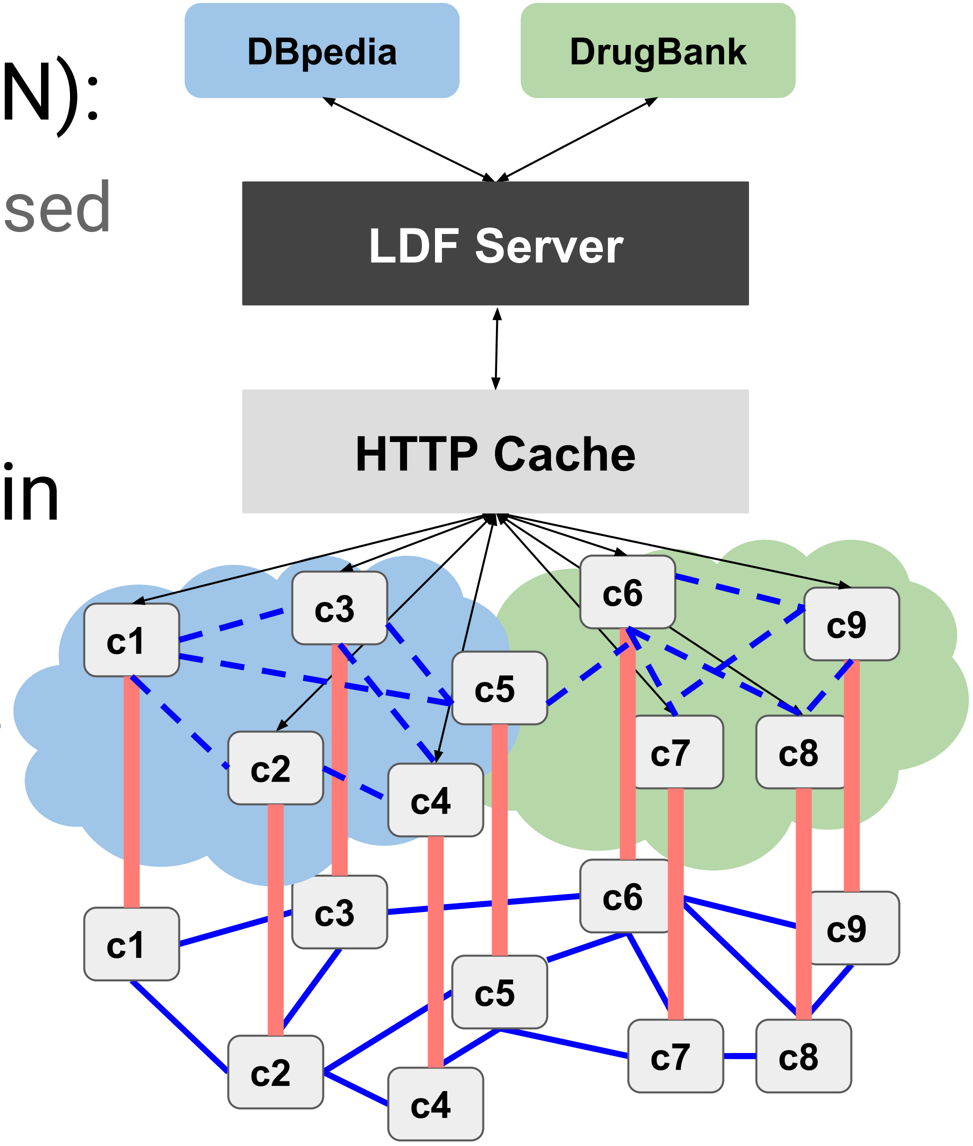
CyCLaDEs: Approach Overview

Cluster Overlay Network (CON):

- Each node has a **profile** based on the **history** of executed queries

Shuffle phases allow to obtain better neighbors.

Each node ranks and selects **best neighbors** based on similarity of profiles.



How to profile nodes?

Executing queries produces a **stream of TPF**, cache is a **window** on this **stream**.

Profile = summary of the recent past = frequency of the *k* last recently used predicates

Node profile - Algorithm

Algorithm 1 ComputeProfile(s,w,t)

Require: : w: Window size, s: Stream of triples, t: timestamp

Ensure: : Pr : set of (predicate, frequency, timestamp) of size w

Pr $\rightarrow \varnothing$

while data stream continues **do**

  Receive the next streaming triple $tp = (s\ p\ o)$

  if $(tp.p, f_p, \_) \in$ Pr **then**

    $Pr.update(tp.p,\ f_p + 1, t)$

  else

    $Pr \cup (tp.p, 1, t)$

  if $|Pr| >$ w **then**

    $Pr \setminus (p_1, f_{p_1}, t_1) : (p_1, f_{p_1}, t_1) \in Pr \wedge \nexists\ (p_2, f_{p_2}, t_2) \in Pr : t_2 < t_1$

Node profile - Algorithm

Algorithm 1 ComputeProfile(s,w,t)

Require: : w: Window size, s: Stream of triples, t: timestamp

Ensure: : Pr : set of (predicate, frequency, timestamp) of size w

Size of the profile

$\mathrm{Pr} \rightarrow \varnothing$
while data stream continue
  Receive the next streaming triple $tp = (s\ p\ o)$
  if $(tp.p, f_p, \_) \in \mathrm{Pr}$ **then**
    $Pr.update(tp.p,\ f_p + 1, t)$
  else
    $Pr \cup (tp.p, 1, t)$
  if $|Pr| > \mathrm{w}$ **then**
    $Pr \setminus (p_1, f_{p_1}, t_1) : (p_1, f_{p_1}, t_1) \in Pr \wedge \nexists\ (p_2, f_{p_2}, t_2) \in Pr : t_2 < t_1$

Node profile - Algorithm

Algorithm 1 ComputeProfile(s,w,t)

Require: : w: Window size, s: Stream of triples, t: timestamp
Ensure: : Pr : set of (predicate, frequency, timestamp) of size w

Stream of triples

$\mathrm{Pr} \rightarrow \varnothing$
while data stream continue
  Receive the next streaming triple $tp = (s\ p\ o)$
  if $(tp.p, f_p, \_) \in \mathrm{Pr}$ **then**
    $Pr.update(tp.p,\ f_p + 1, t)$
  else
    $Pr \cup (tp.p, 1, t)$
  if $|Pr| >$ w **then**
    $Pr \setminus (p_1, f_{p_1}, t_1) : (p_1, f_{p_1}, t_1) \in Pr \wedge \nexists\ (p_2, f_{p_2}, t_2) \in Pr : t_2 < t_1$

Node profile - Algorithm

Algorithm 1 ComputeProfile(s,w,t)

Require: : w: Window size, s: Stream of triples, t: timestamp
Ensure: : Pr : set of (predicate, frequency, timestamp) of size w

```
Pr → ∅
while data stream continue
  Receive the next streaming triple tp = (s p o)
  if (tp.p, f_p, _) ∈ Pr then
    Pr.update(tp.p, f_p + 1, t)
  else
    Pr ∪ (tp.p, 1, t)
  if |Pr| > w then
    Pr \ (p1, f_p1, t1) : (p1, f_p1, t1) ∈ Pr ∧ ∄ (p2, f_p2, t2) ∈ Pr : t2 < t1
```

Timestamp of node

Node profile - Algorithm

Algorithm 1 ComputeProfile(s,w,t)

Require: : w: Window size, s: Stream of triples, t: timestamp
Ensure: : Pr : set of (predicate, frequency, timestamp) of size w

```
Pr → ∅
while data stream continues do
  Receive the next streaming triple tp = (s
  if (tp.p, f_p, _) ∈ Pr then
    Pr.update(tp.p, f_p + 1, t)
  else
    Pr ∪ (tp.p, 1, t)
  if |Pr| > w then
    Pr \ (p_1, f_p1, t_1) : (p_1, f_p1, t_1) ∈ Pr ∧ ∄ (p_2, f_p2, t_2) ∈ Pr : t_2 < t_1
```

For each predicate in the stream

Node profile - Algorithm

Algorithm 1 ComputeProfile(s,w,t)

Require: : w: Window size, s: Stream of triples, t: timestamp

Ensure: : Pr : set of (predicate, frequency, timestamp) of size w

Pr $\rightarrow \varnothing$

while data stream continues **do**

  Receive the next streaming triple $tp = (s\ p\ o)$

  if $(tp.p, f_p, \_) \in$ Pr **then**

    $Pr.update(tp.p,\ f_p + 1, t)$

  else

    $Pr \cup (tp.p, 1, t)$

  if $|Pr| >$ w **then**

    $Pr \setminus (p_1, f_{p_1}, t_1) : (p_1, f_{p_1}, t_1) \in Pr \wedge \nexists\ (p_2, f_{p_2}, t_2) \in Pr : t_2 < t_1$

If predicate exist:

1) Increment frequency by 1
2) Update timestamp

Node profile - Algorithm

Algorithm 1 ComputeProfile(s,w,t)

Require: : w: Window size, s: Stream of triples, t: timestamp
Ensure: : Pr : set of (predicate, frequency, timestamp) of size w

```
Pr → ∅
while data stream continues do
  Receive the next streaming triple tp = (s p o)
  if (tp.p, f_p, _) ∈ Pr then
    Pr.update(tp.p, f_p + 1, t)
  else
    Pr ∪ (tp.p, 1, t)
  if |Pr| > w then
    Pr \ (p_1, f_{p_1}, t_1) : (p_1, f_{p_1}, t_1) ∈ Pr ∧ ∄ (p_2, f_{p_2}, t_2) ∈ Pr : t_2 < t_1
```

If predicate does not exist:

1) Insert predicate in profile
   a) With frequency = 1
   b) Current timestamp

Node profile - Algorithm

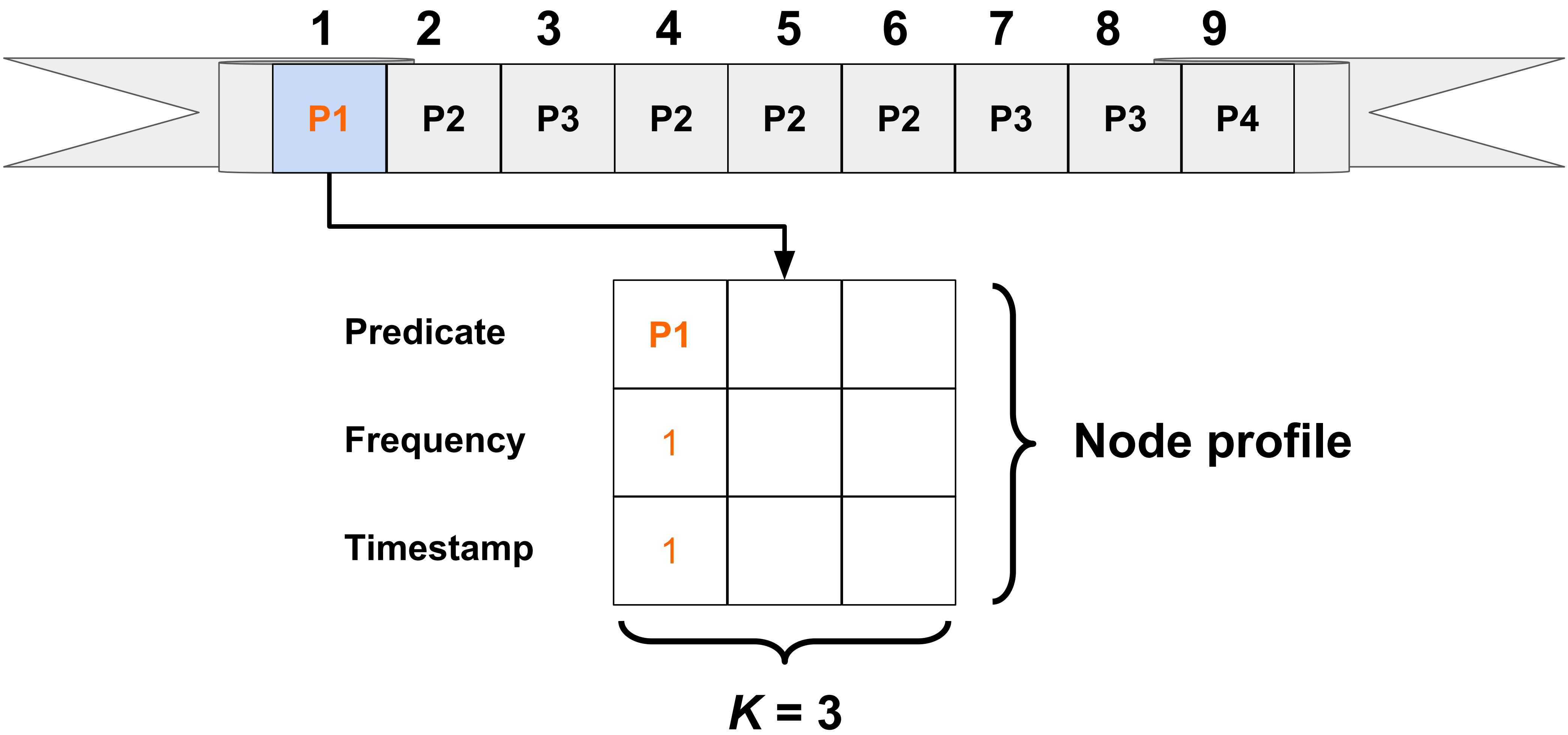
Algorithm 1 ComputeProfile(s,w,t)

Require: : w: Window size, s: Stream of triples, t: timestamp
Ensure: : Pr : set of (predicate, frequency, timestamp) of size w

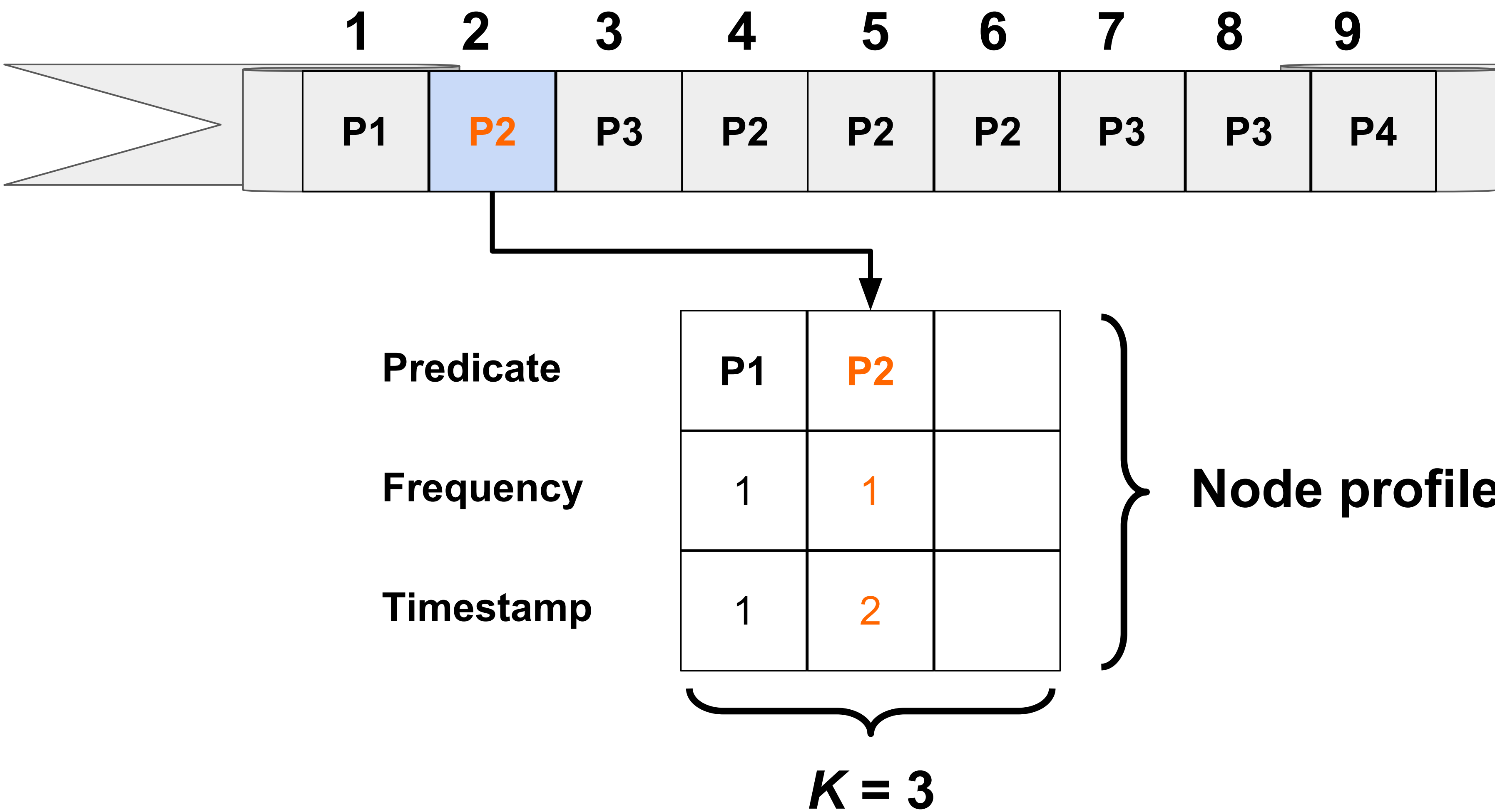
$\mathrm{Pr} \rightarrow \varnothing$
while data stream continues **do**
  Receive the next streaming triple $tp =$
  if $(tp.p, f_p, \_) \in \mathrm{Pr}$ **then**
    $Pr.update(tp.p,\ f_p + 1, t)$
  else
    $Pr \cup (tp.p, 1, t)$
  if $|Pr| >$ w **then**
    $Pr \setminus (p_1, f_{p_1}, t_1) : (p_1, f_{p_1}, t_1) \in Pr \wedge \nexists\ (p_2, f_{p_2}, t_2) \in Pr : t_2 < t_1$

If Pr > w:
Remove the oldest entry

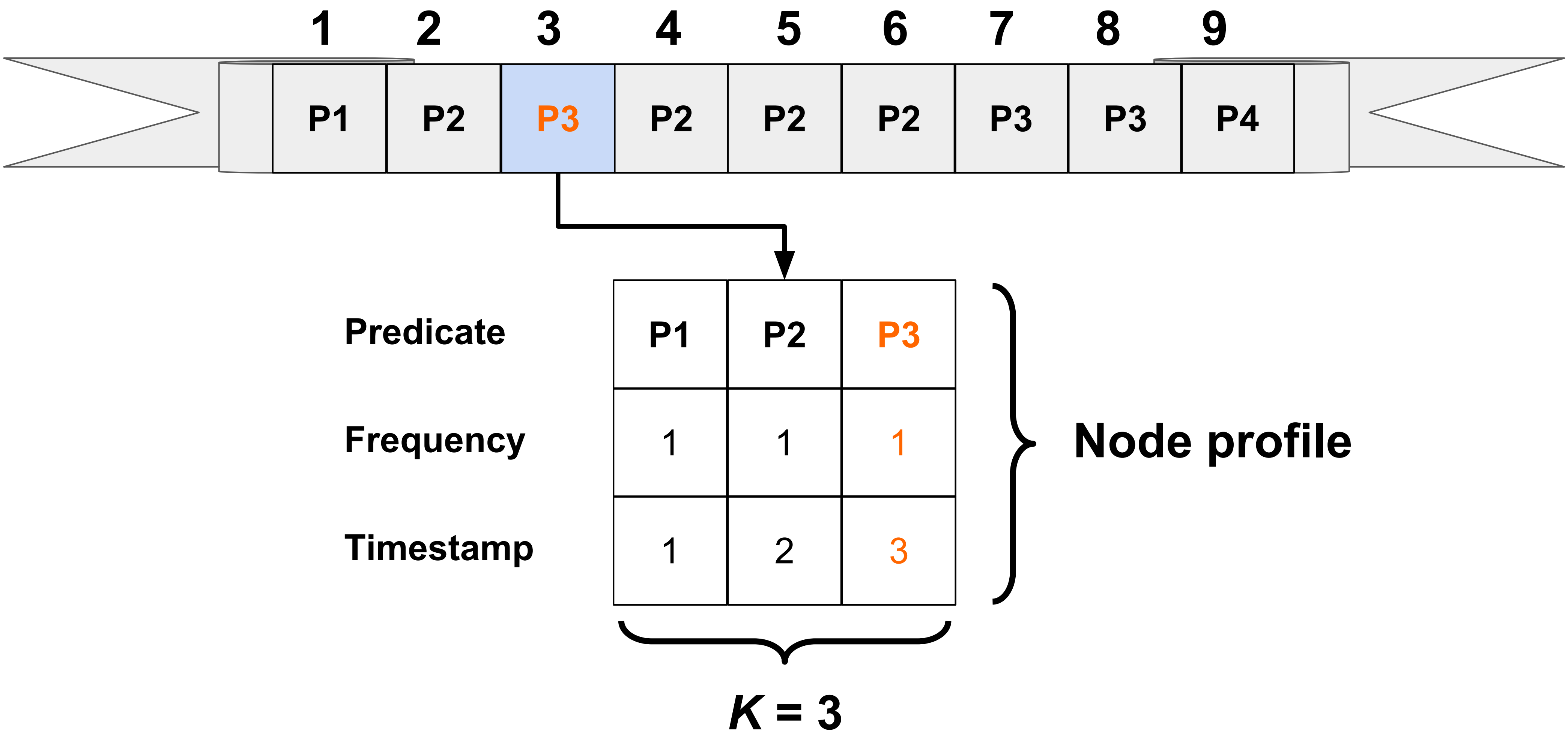
Computing Profile



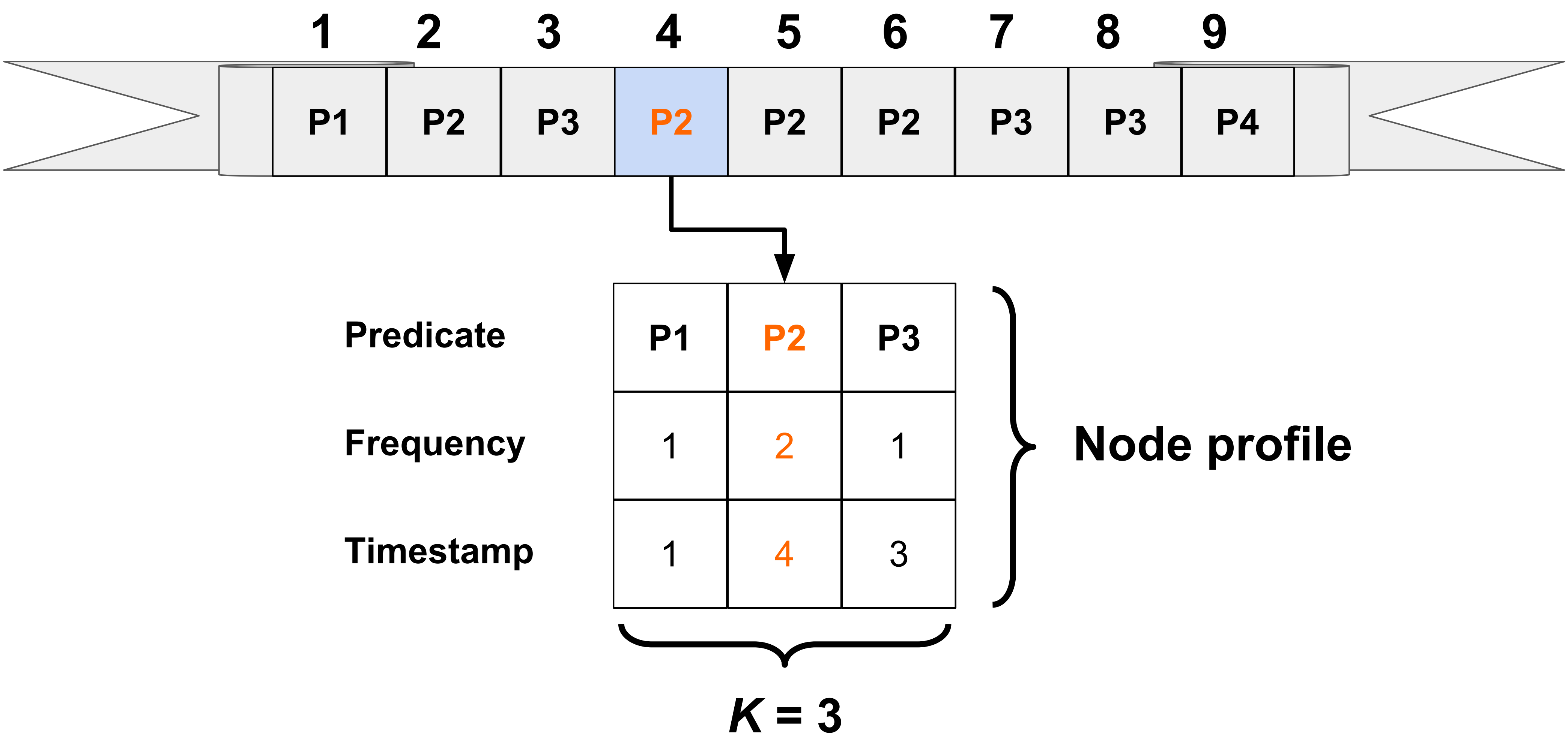
Computing Profile



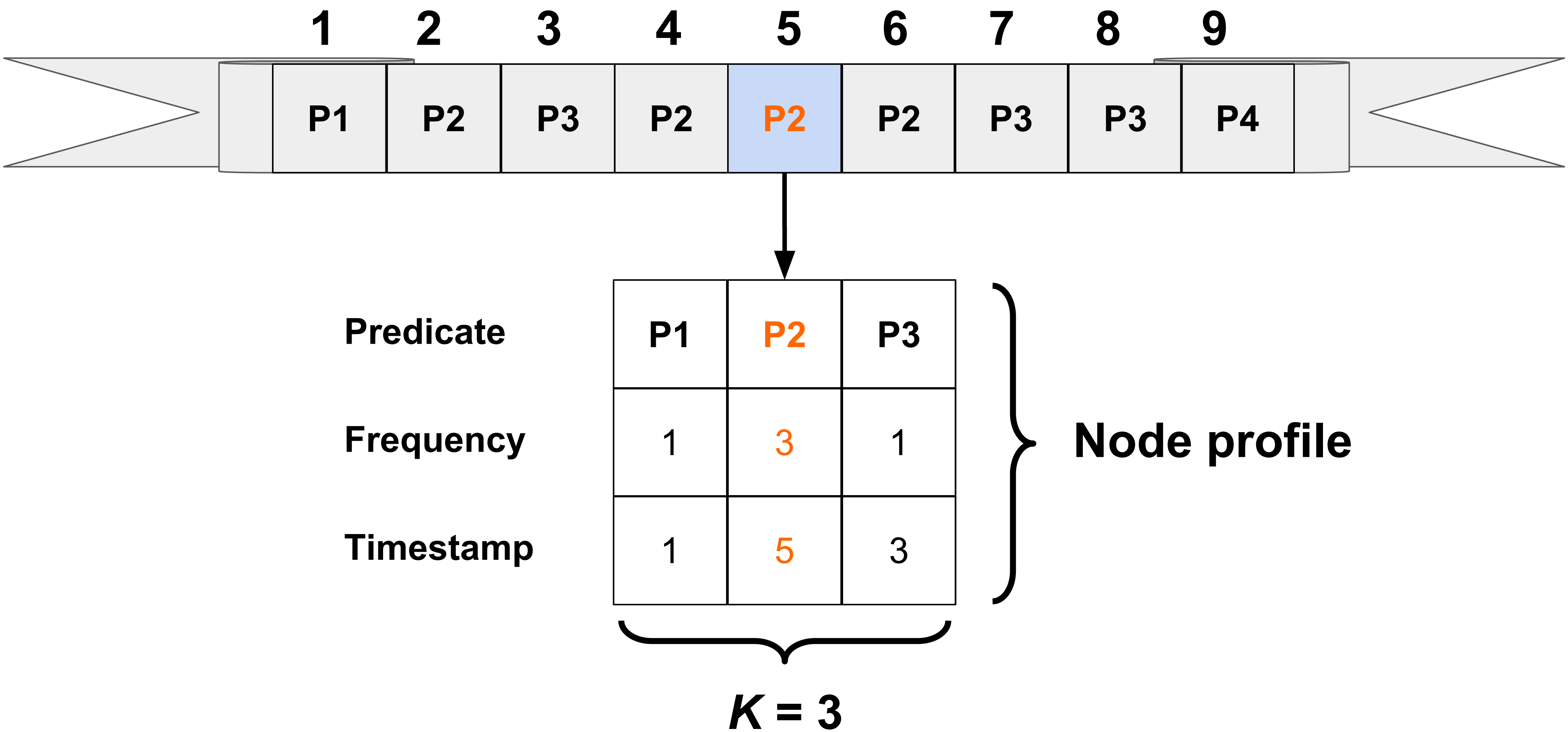
Computing Profile



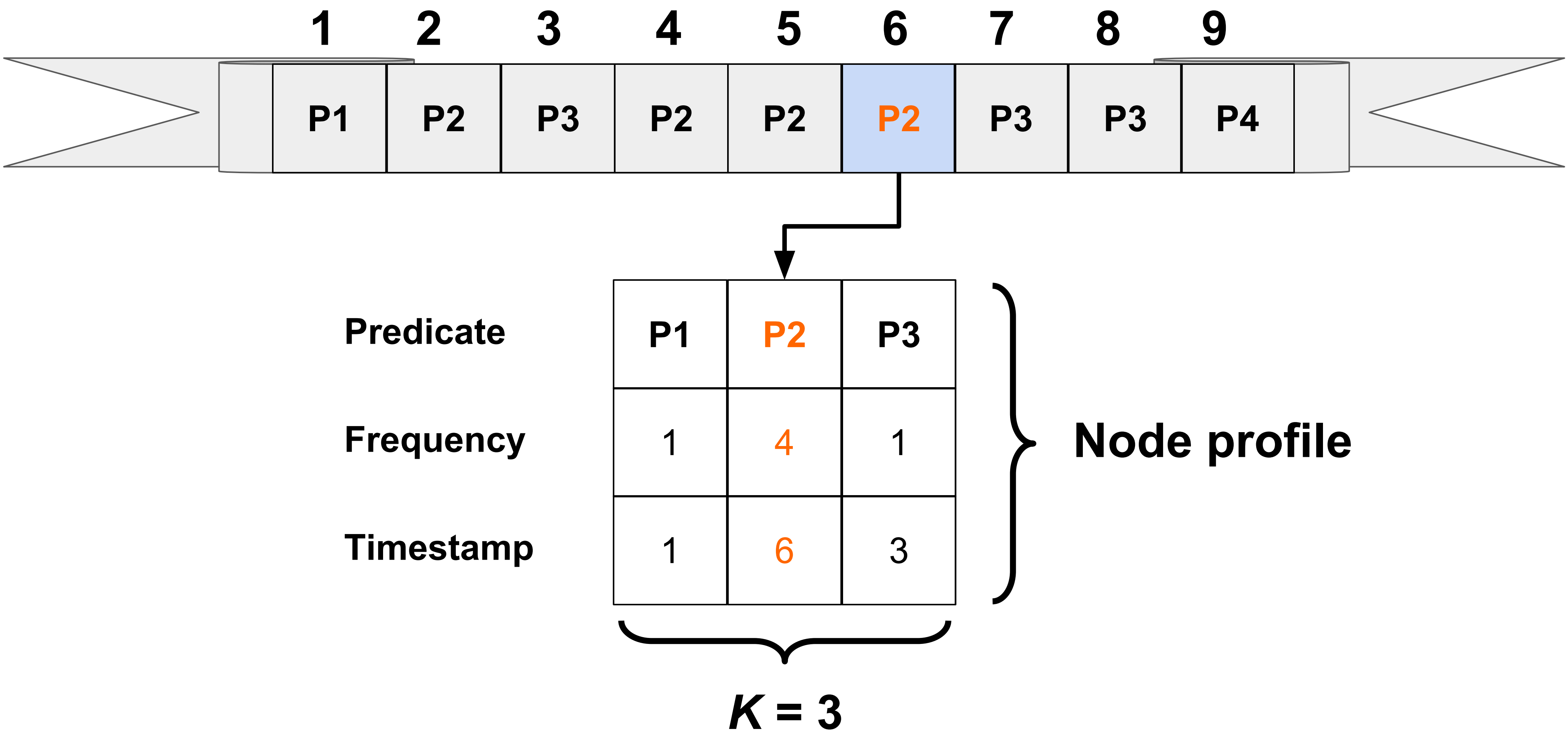
Computing Profile



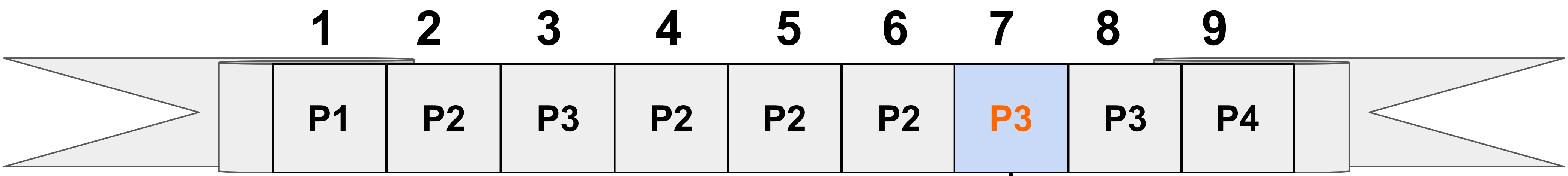
Computing Profile



Computing Profile



Computing Profile

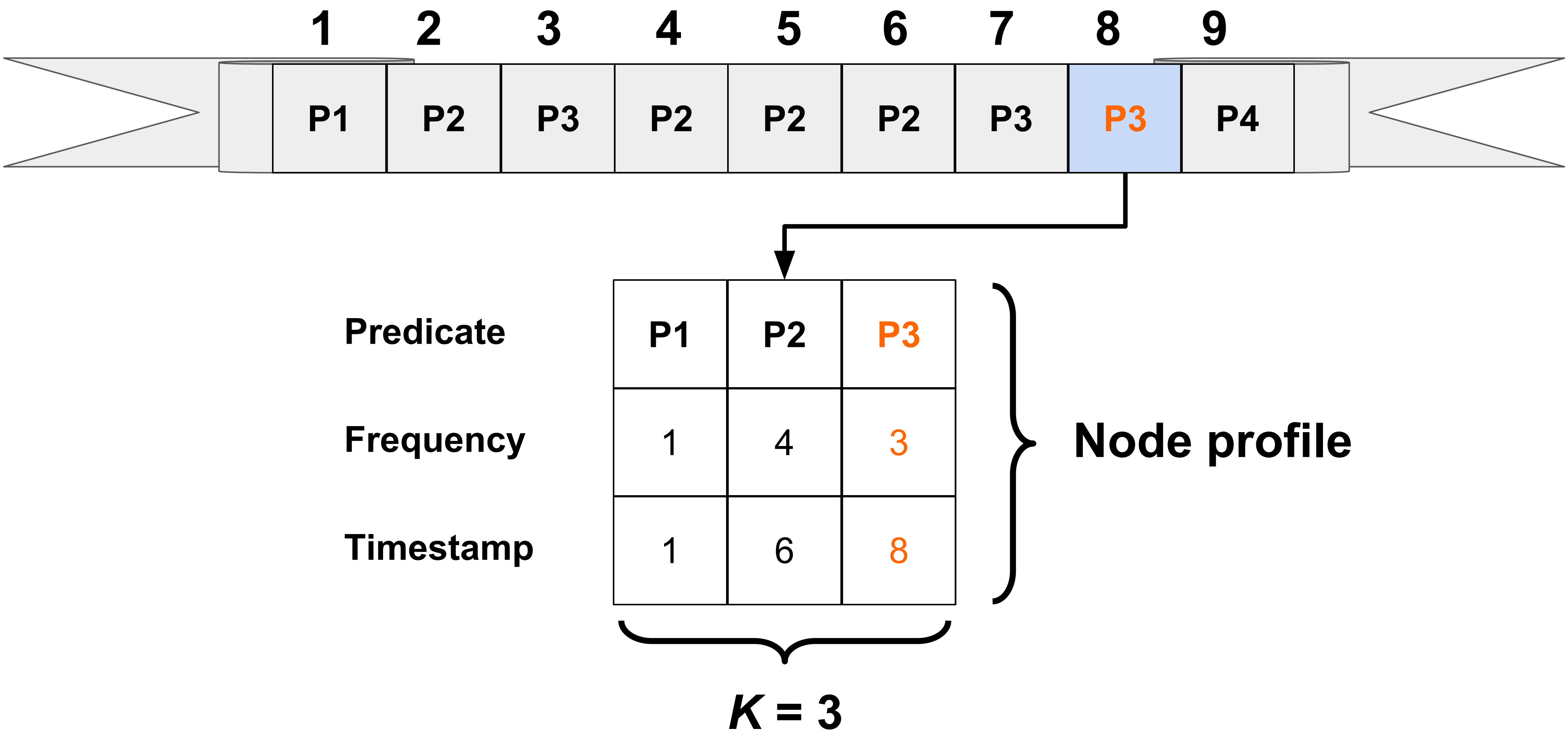


Predicate	**P1**	**P2**	**P3**
Frequency	1	4	2
Timestamp	1	6	7

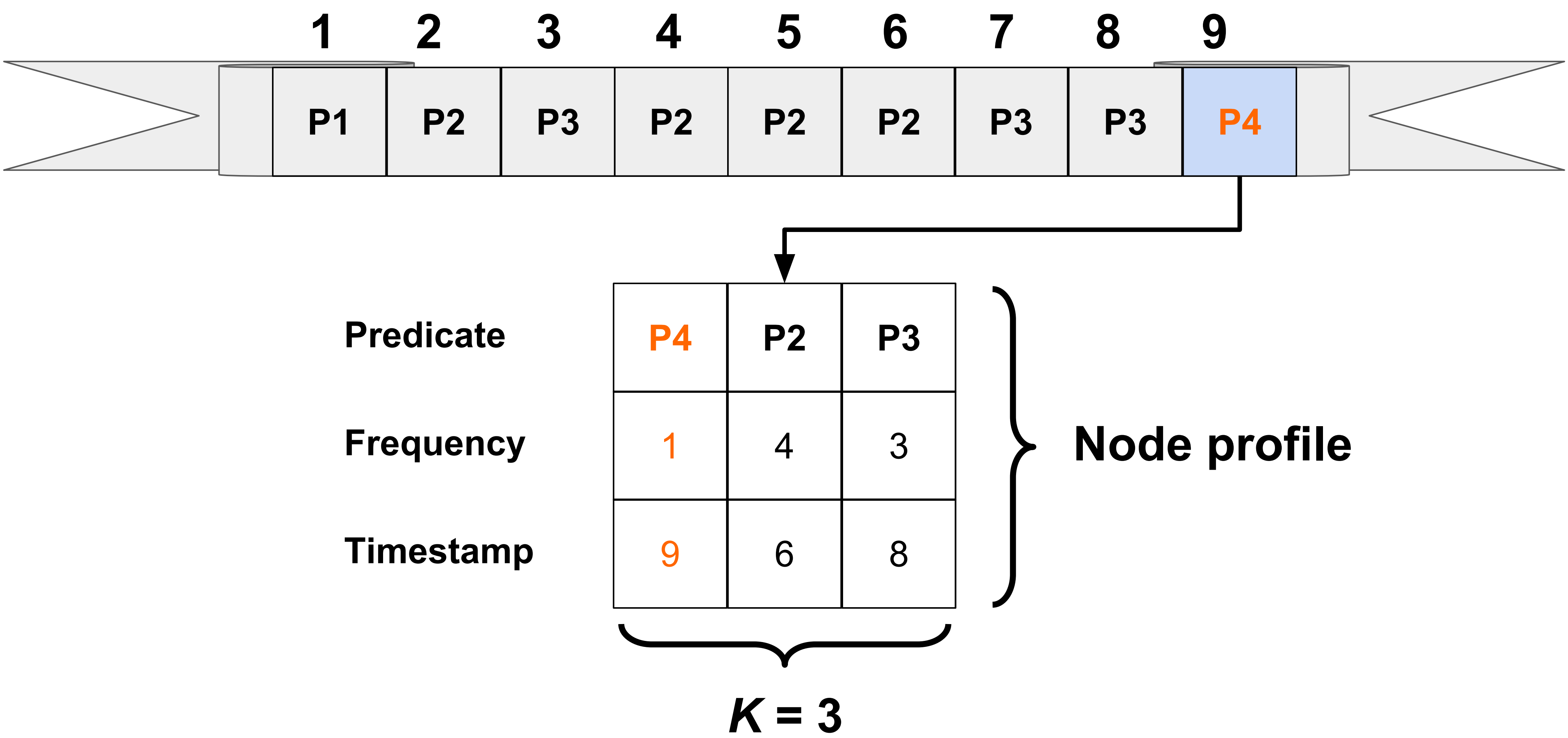
Node profile

$K = 3$

Computing Profile



Computing Profile



How to compare neighbors?

During ranking **nodes** are **compared** thank to their **profile**.

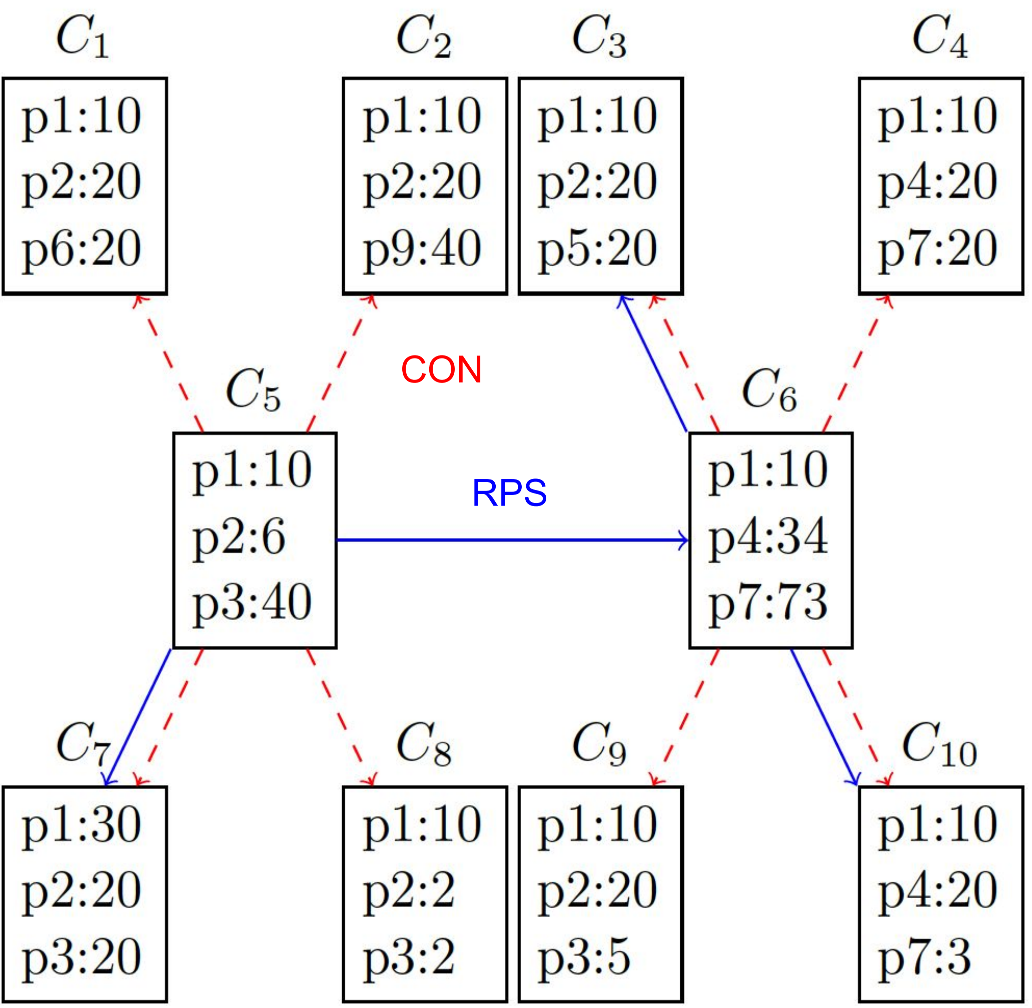
Nodes are compared with the generalized **Jaccard similarity coefficient**.

$$J(x, y) = \frac{\sum_i min(x_i, y_i)}{\sum_i max(x_i, y_i)}$$

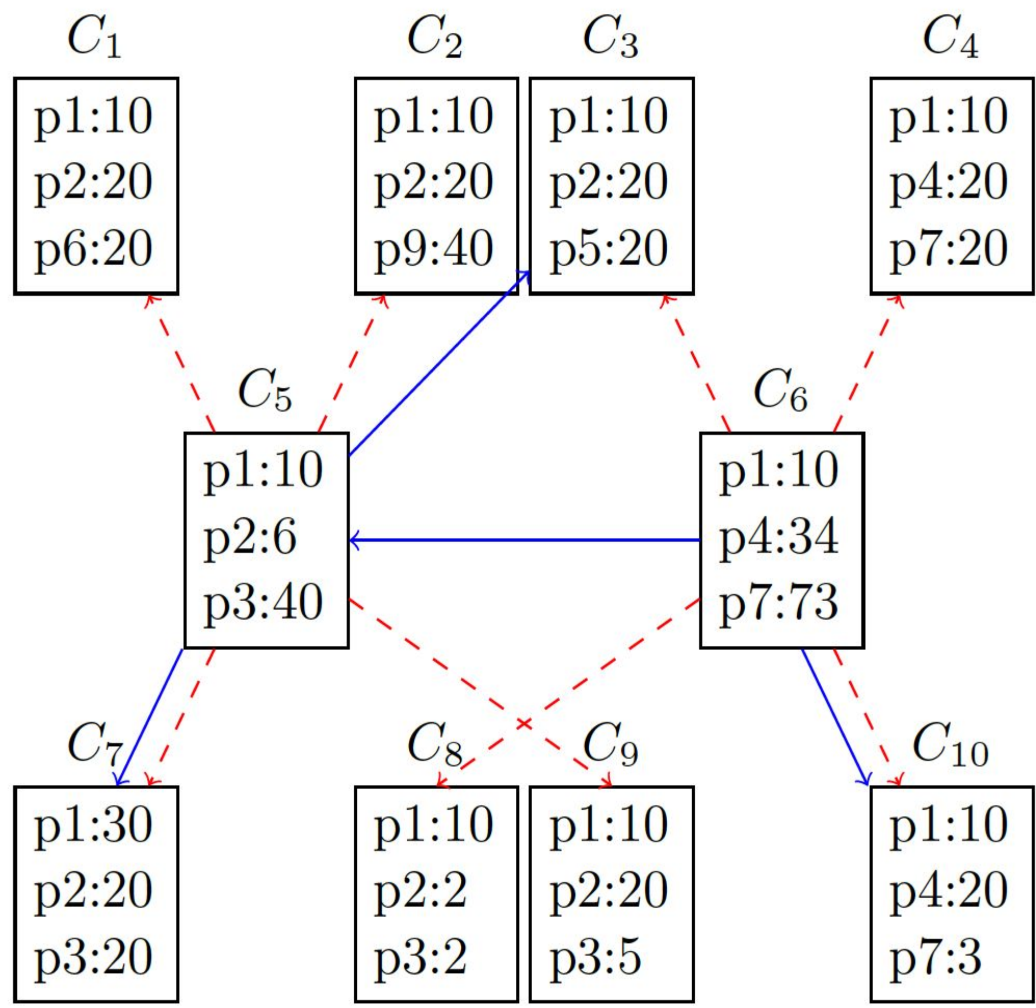
	P1	P2	P3
C5	10	6	40
C9	10	20	5
C8	10	2	2

- J(C5,C9) = (10+6+5)/(10+20+40) = **0.3**
- J(C5,C8) = 14/56 = **0.25**

C5 shuffling with C6 : #RPS=2, #CON=4



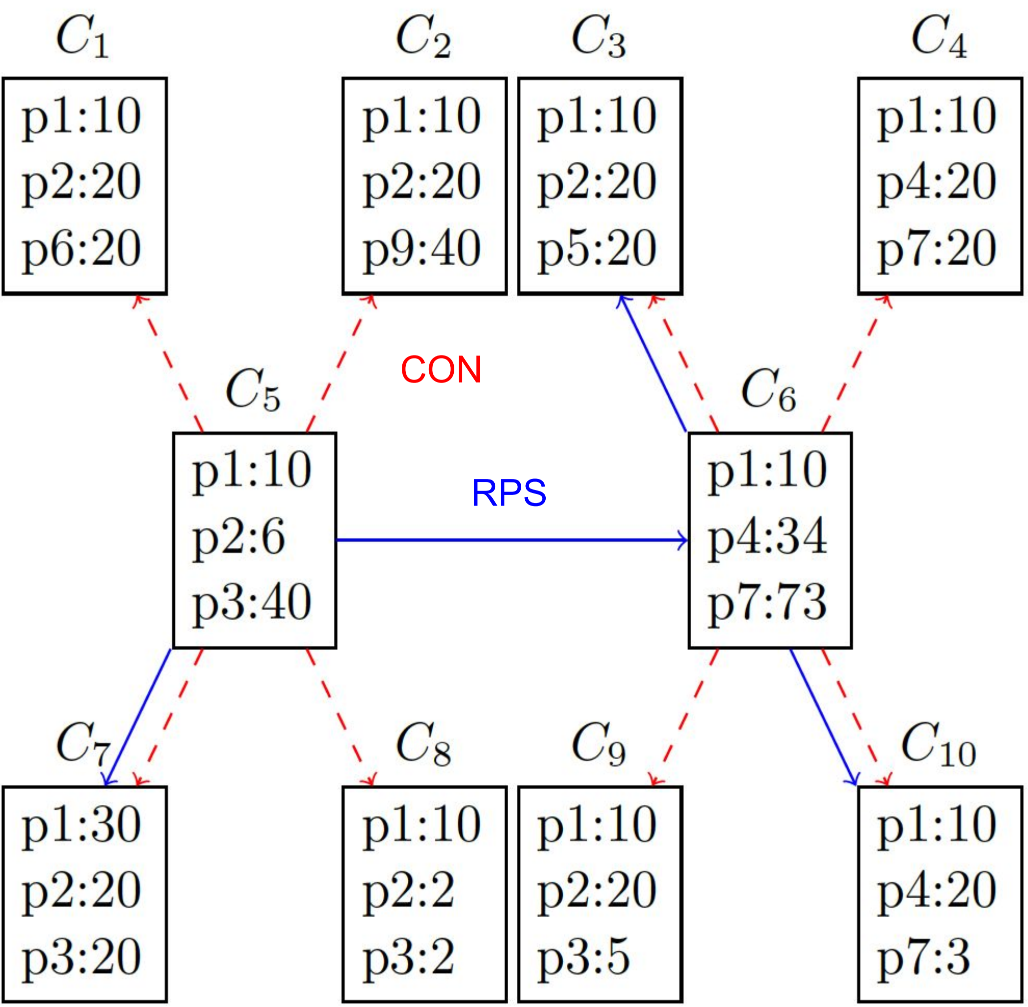
After Shuffling: RPS and CON updated



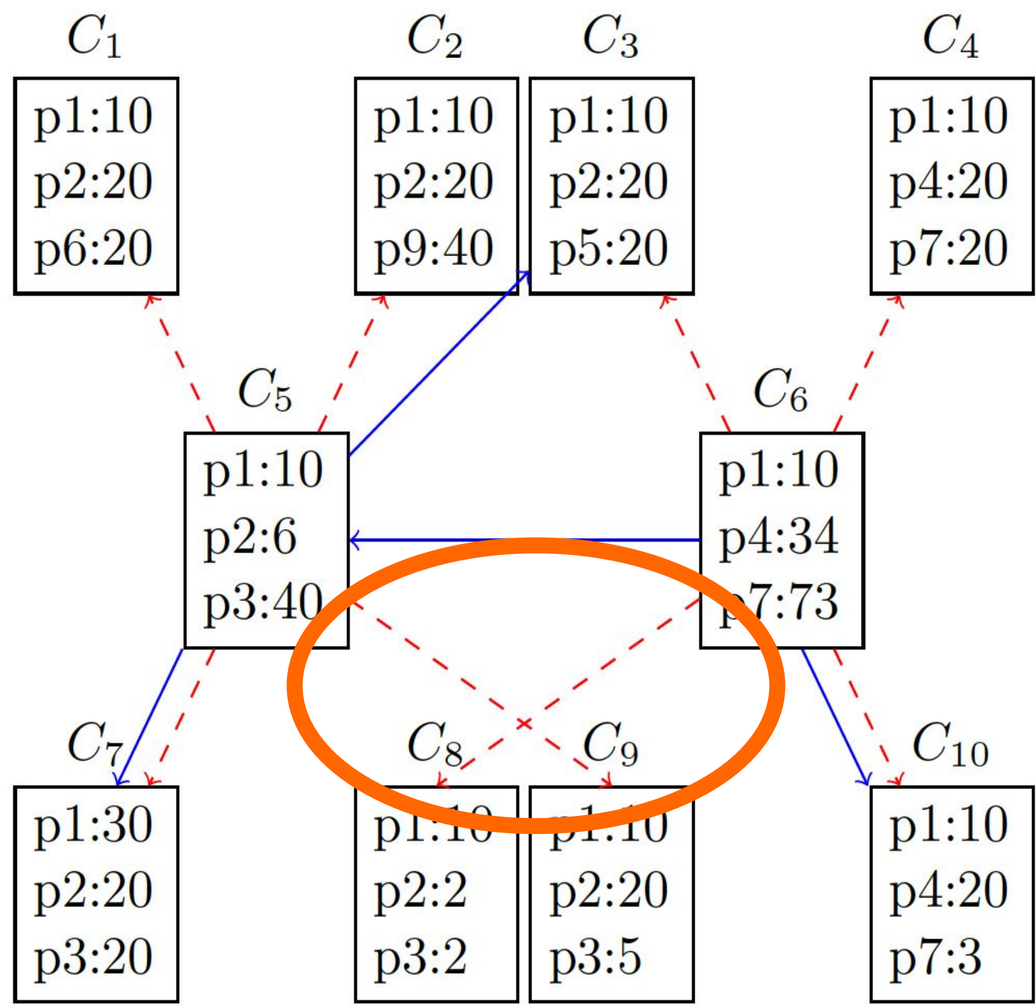
For C5: C9 is more similar than C8

For C6: C8 is more similar than C9

C5 shuffling with C6 : #RPS=2, #CON=4



After Shuffling: RPS and CON updated



For C5: C9 is more similar than C8

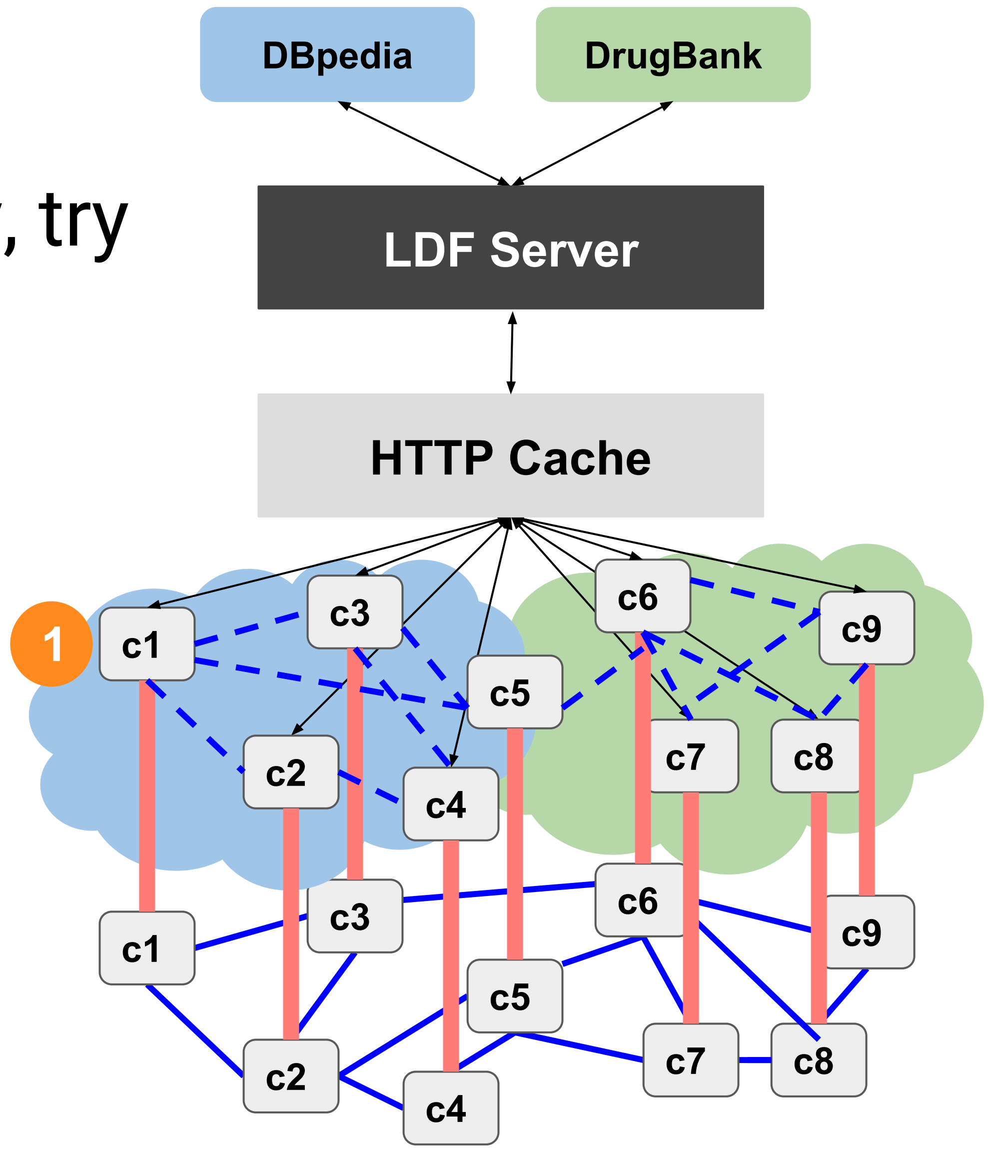
For C6: C8 is more similar than C9

Queries with CyCLaDEs

C1 executes query Q1.

For each triple pattern query, try to resolve pattern in:

1) Local cache
2) Neighborhood cache
3) HTTP cache
4) LDF Server

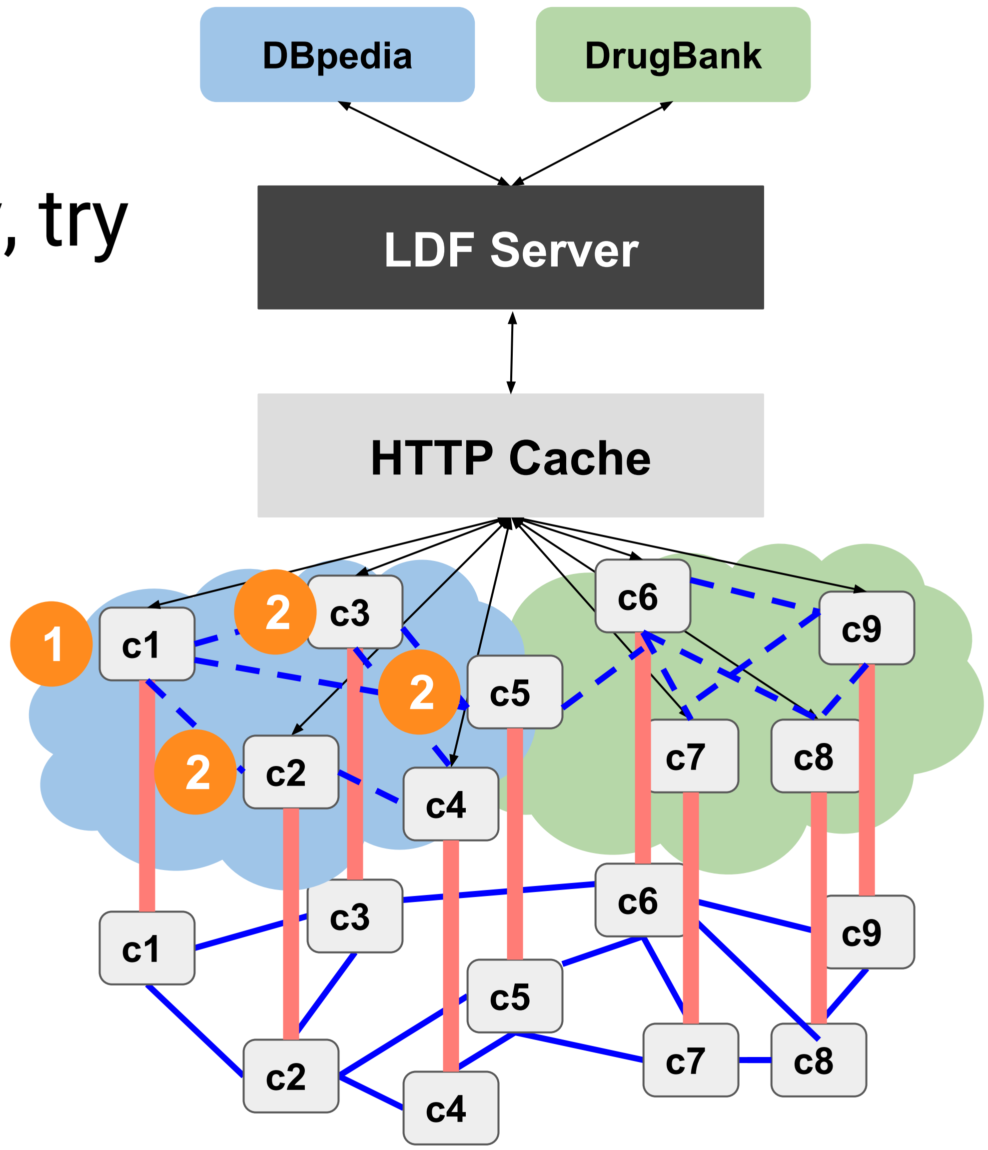


Queries with CyCLaDEs

C1 executes query Q1.

For each triple pattern query, try to resolve pattern in:

1) Local cache
2) Neighborhood cache
3) HTTP cache
4) LDF Server

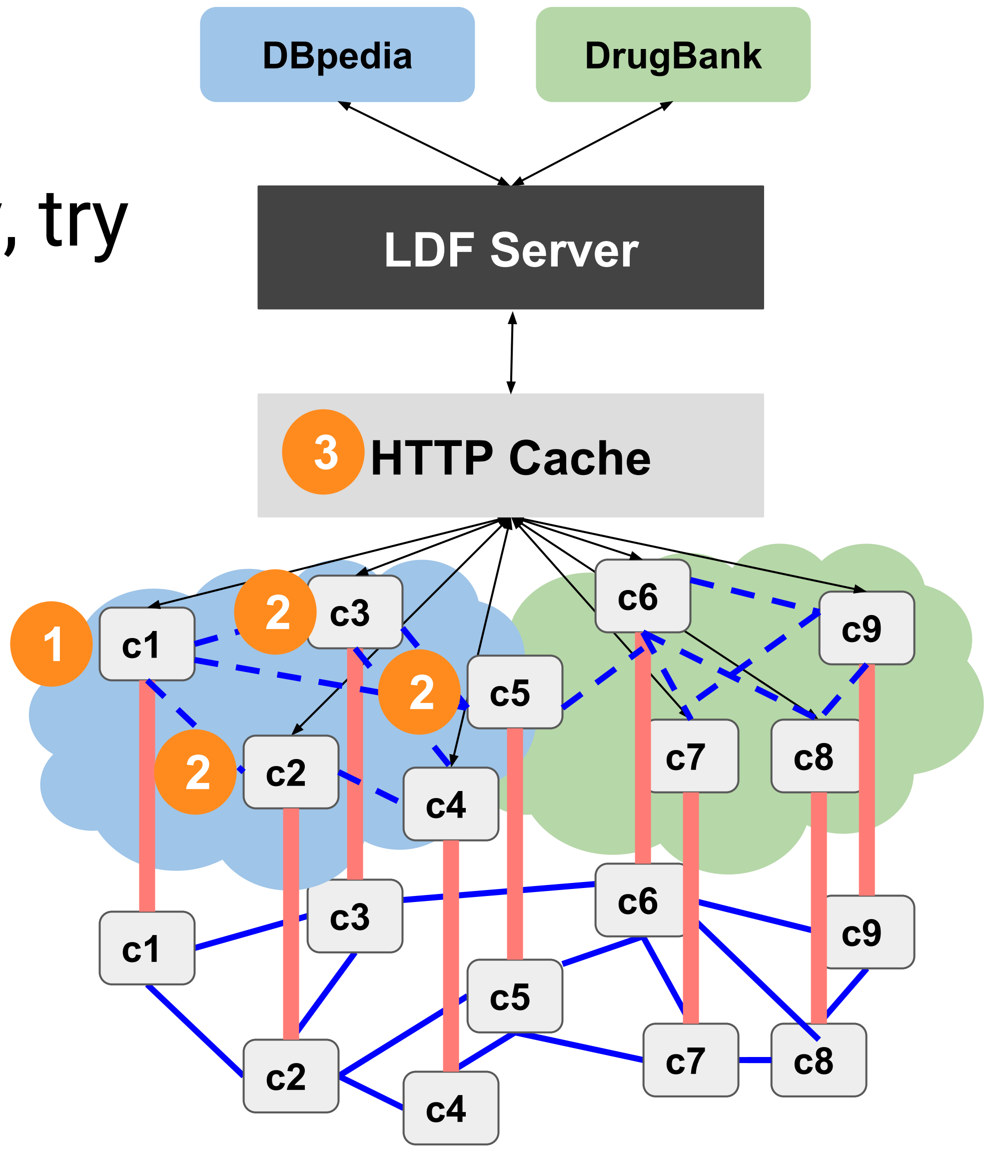


Queries with CyCLaDEs

C1 executes query Q1.

For each triple pattern query, try to resolve pattern in:

1) Local cache
2) Neighborhood cache
3) HTTP cache
4) LDF Server

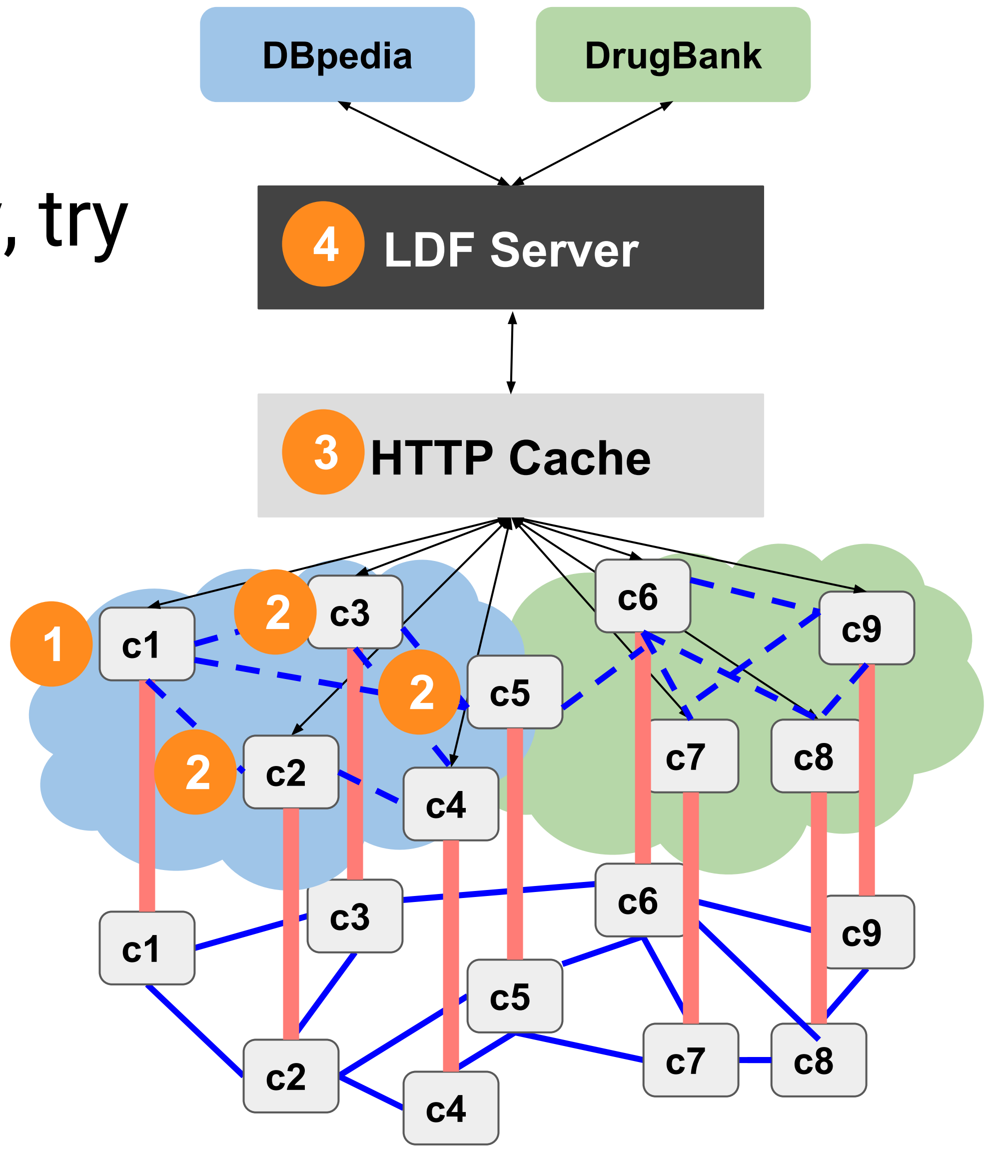


Queries with CyCLaDEs

C1 executes query Q1.

For each triple pattern query, try to resolve pattern in:

1) Local cache
2) Neighborhood cache
3) HTTP cache
4) LDF Server

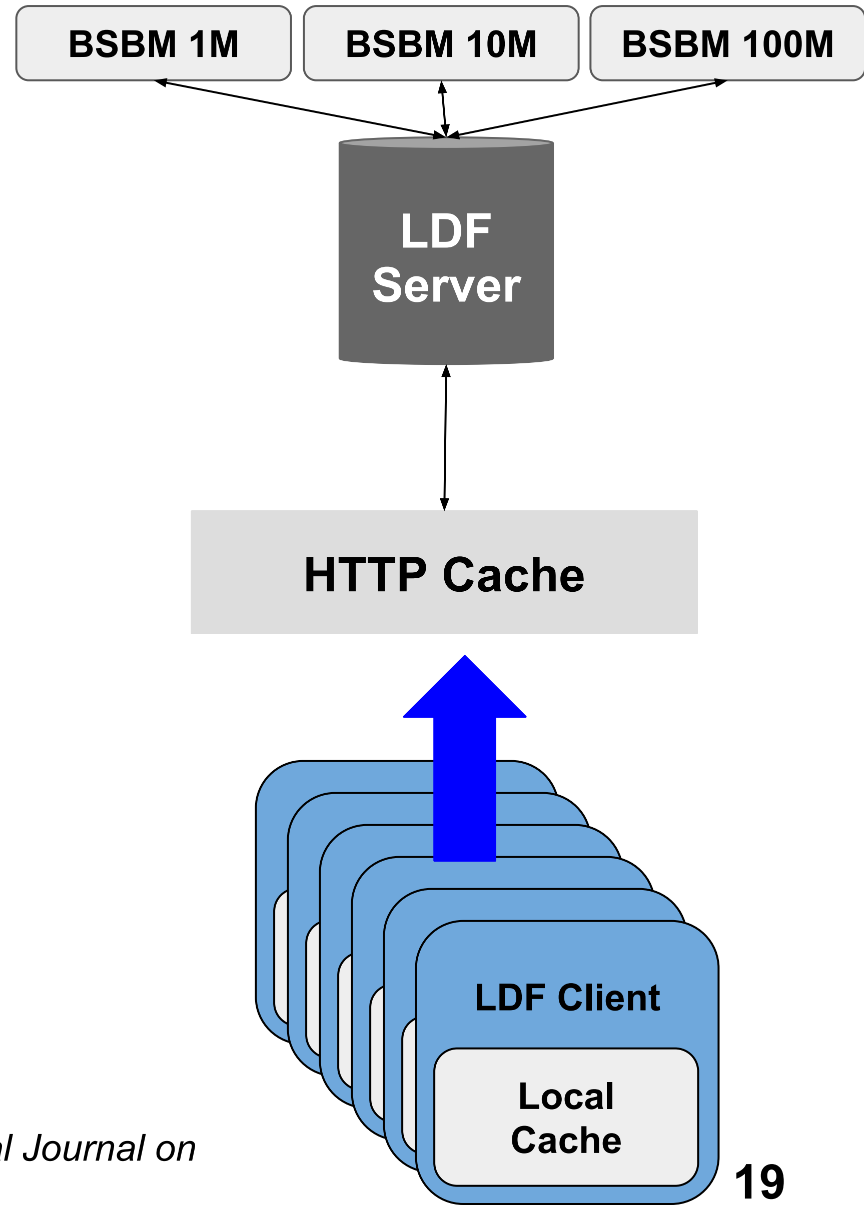


Experiments

Berlin SPARQL Benchmark (BSBM) [7] chosen for simulating **Web applications**.

One LDF Server with one classic HTTP cache hosting BSBM datasets.

A network of extended LDF clients* running a query mix of BSBM queries.



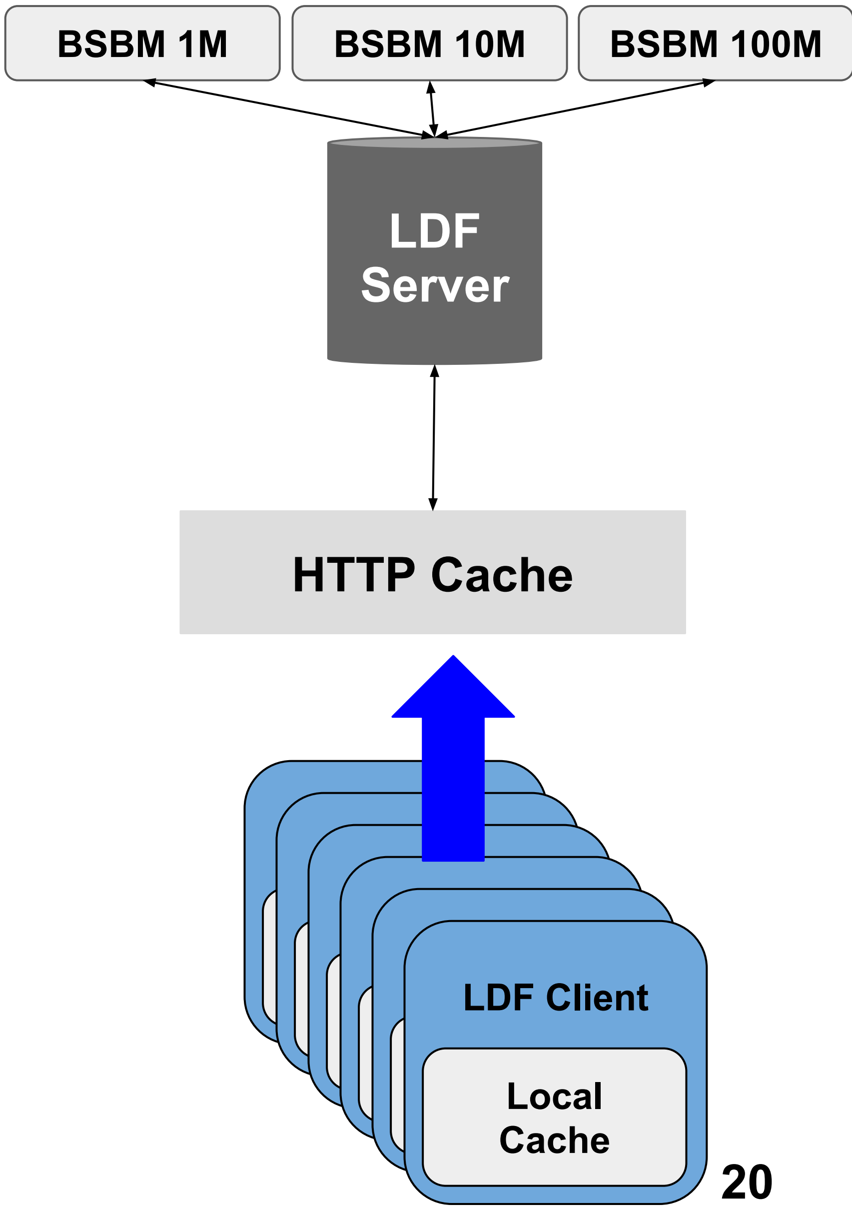
* https://github.com/pfolz/cyclades

[7] C. Bizer and A. Schultz. The berlin sparql benchmark. *International Journal on Semantic Web and Information Systems*, 2009.

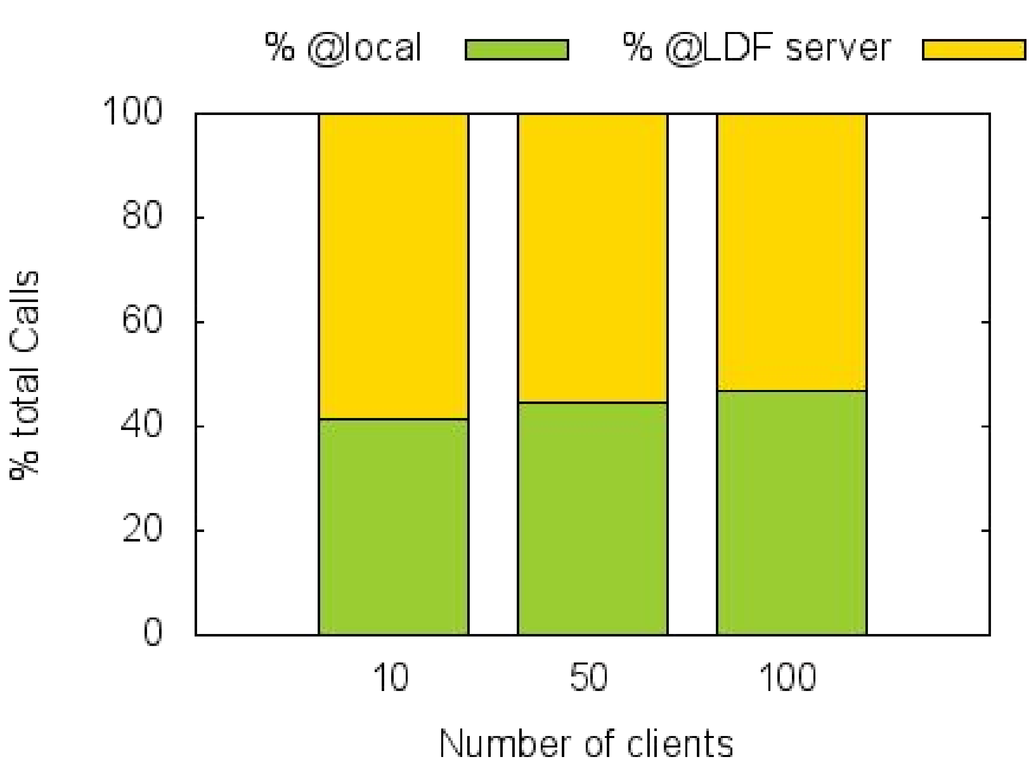
Experiments - Parameters

Each **client** has a **query mix** of **25 queries** generated from **12 templates.**

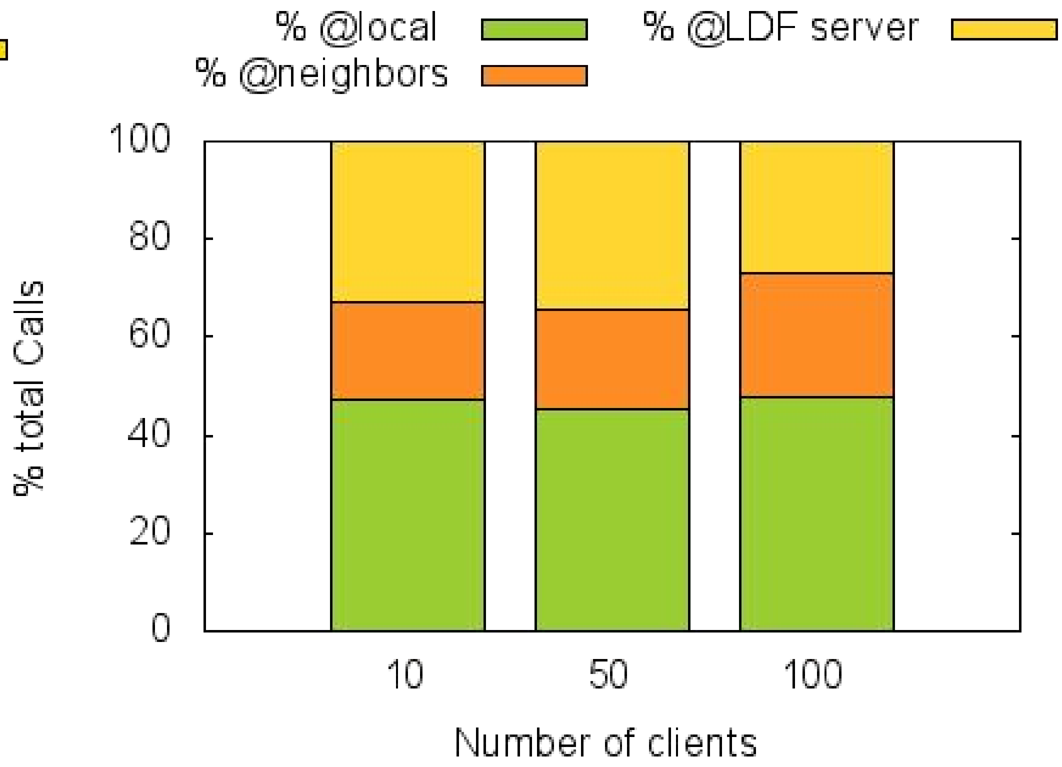
- Shuffling phases occur every 10 seconds
- One **warmup round** followed by one **real round**



BSBM 1M, cache = 1000, profile size = 10
10 clients: RPS = 4, CON = 9
50 clients: RPS = 6, CON = 15
100 clients: RPS = 7, CON = 20



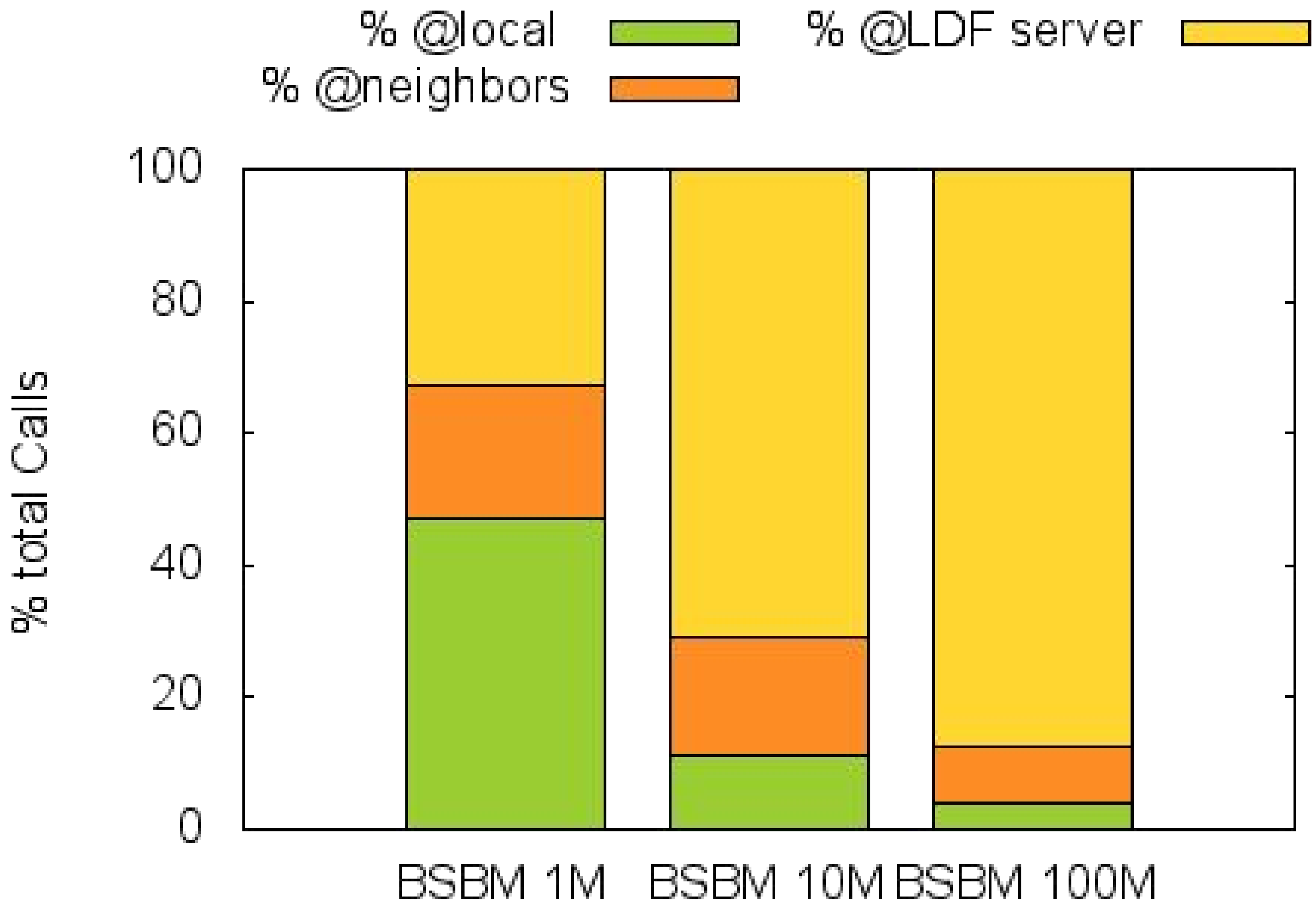
Without CyCLaDEs



With CyCLaDEs

~ 20% neighbors cache hit-rate, whatever the number of clients

10 clients, RPS = 4, CON = 9, cache = 1000

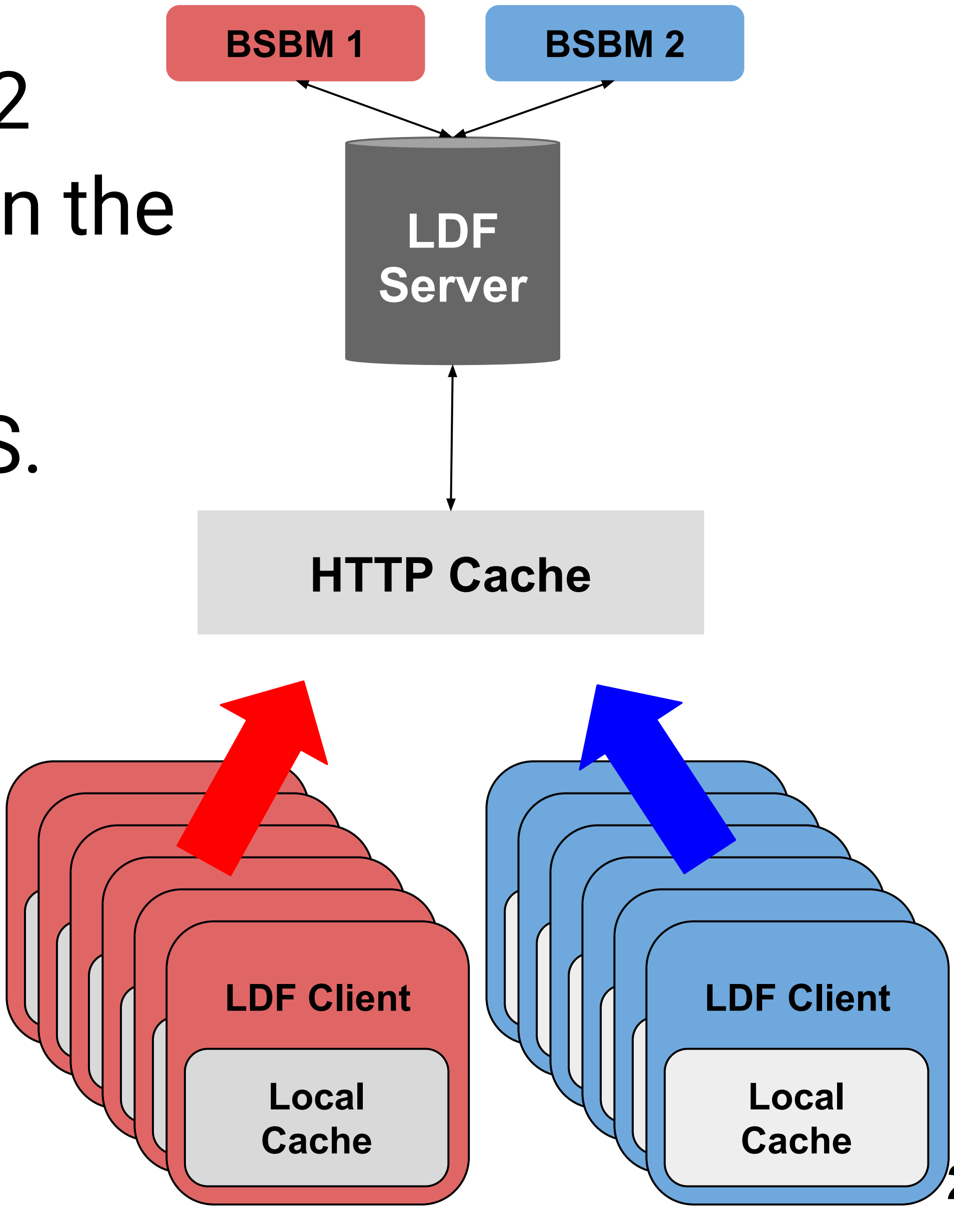


Behavioral cache better resists than local cache

CyCLaDEs with 2 Communities

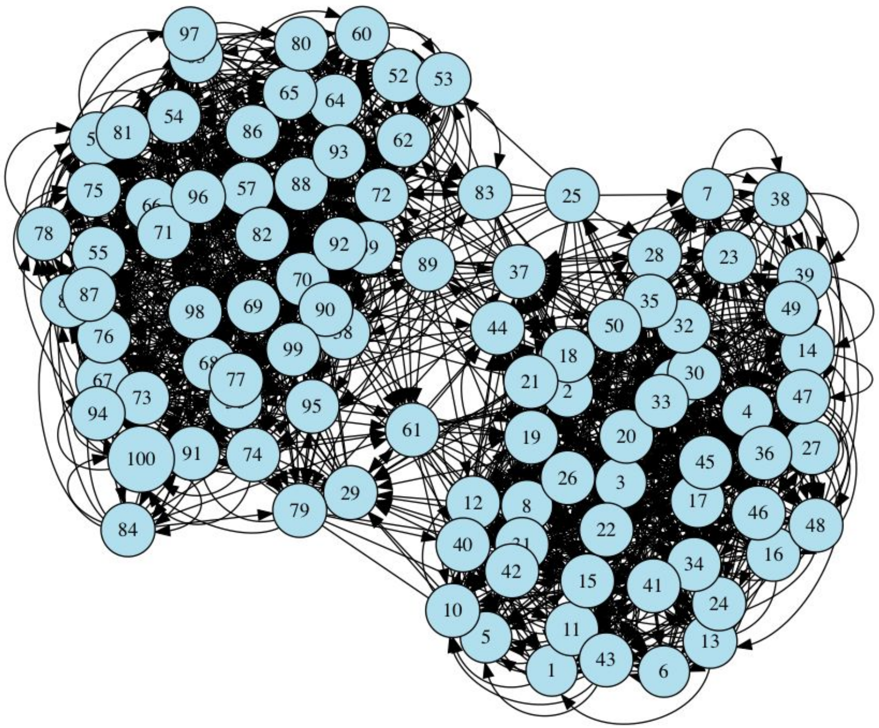
Two communities access 2 **different** BSBM datasets on the same server.

Nodes share the same RPS.

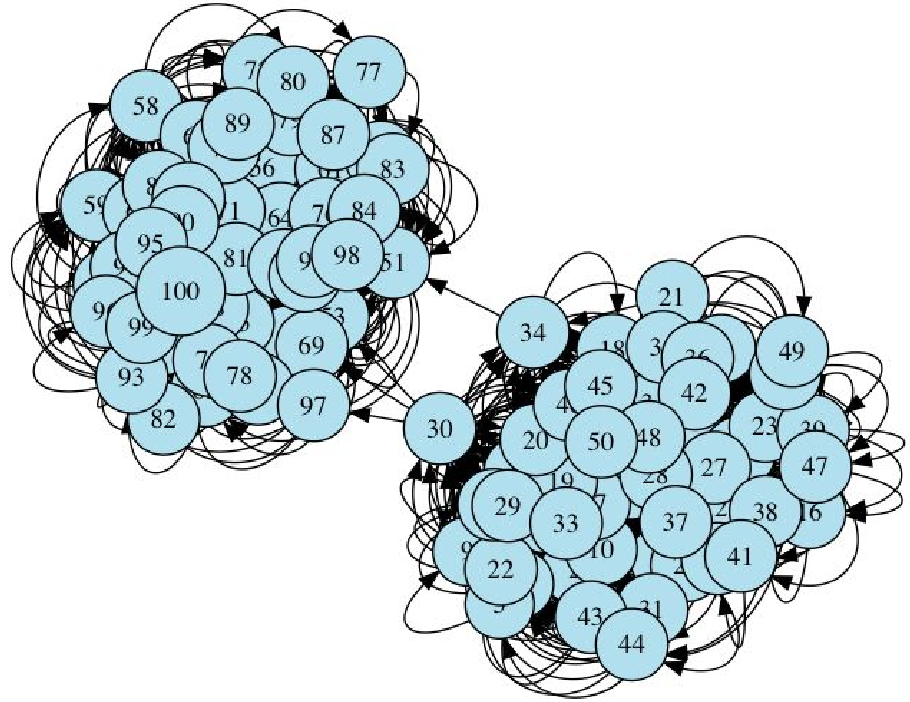


Will CyCLaDEs profiles detect communities?

2 BSBM 1M datasets, 50 clients per data set, cache = 1000



Profile = 5



Profile = 30

In CON overlay, CyCLaDEs builds two distinct communities BSBM1 and BSBM2

Conclusion

CyCLaDEs builds a **behavioral decentralized cache** for LDF clients.

CyCLaDEs **reduces calls** to the **server** in the context of **Web applications**.

Towards a Federation of Data Consumers

Future Works

Measure the impact on **execution time**.

CyCLaDEs **brings** the **data** to the **queries**.

Bring queries to the **data**.

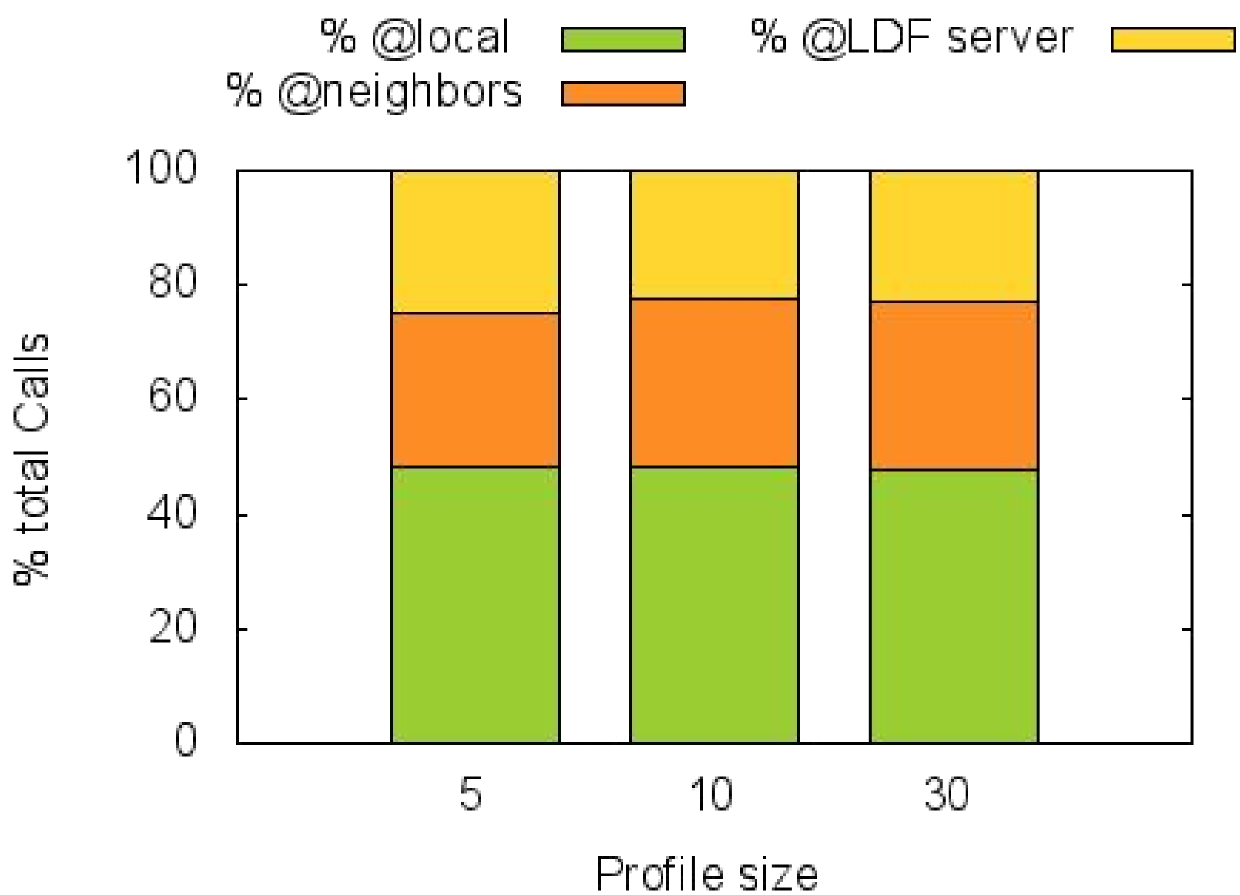
United Federation of Data Consumers

CyCLaDEs: A Decentralized Cache for Triple Pattern Fragments

Pauline Folz, Hala Skaf-Molli & Pascal Molli

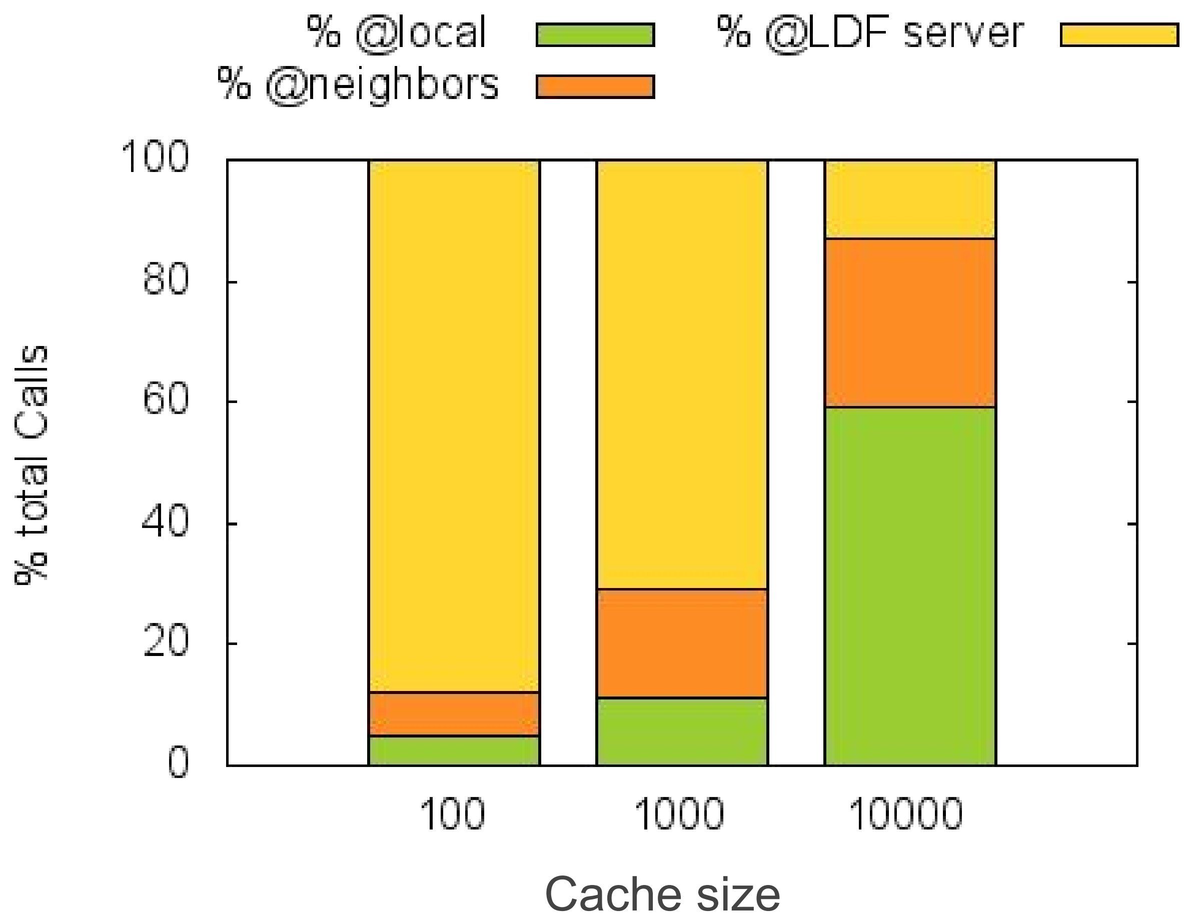
ESWC 2016 - May 2016 - Heraklion

2 BSBM 1M dataset, 50 clients per data set, cache = 1000



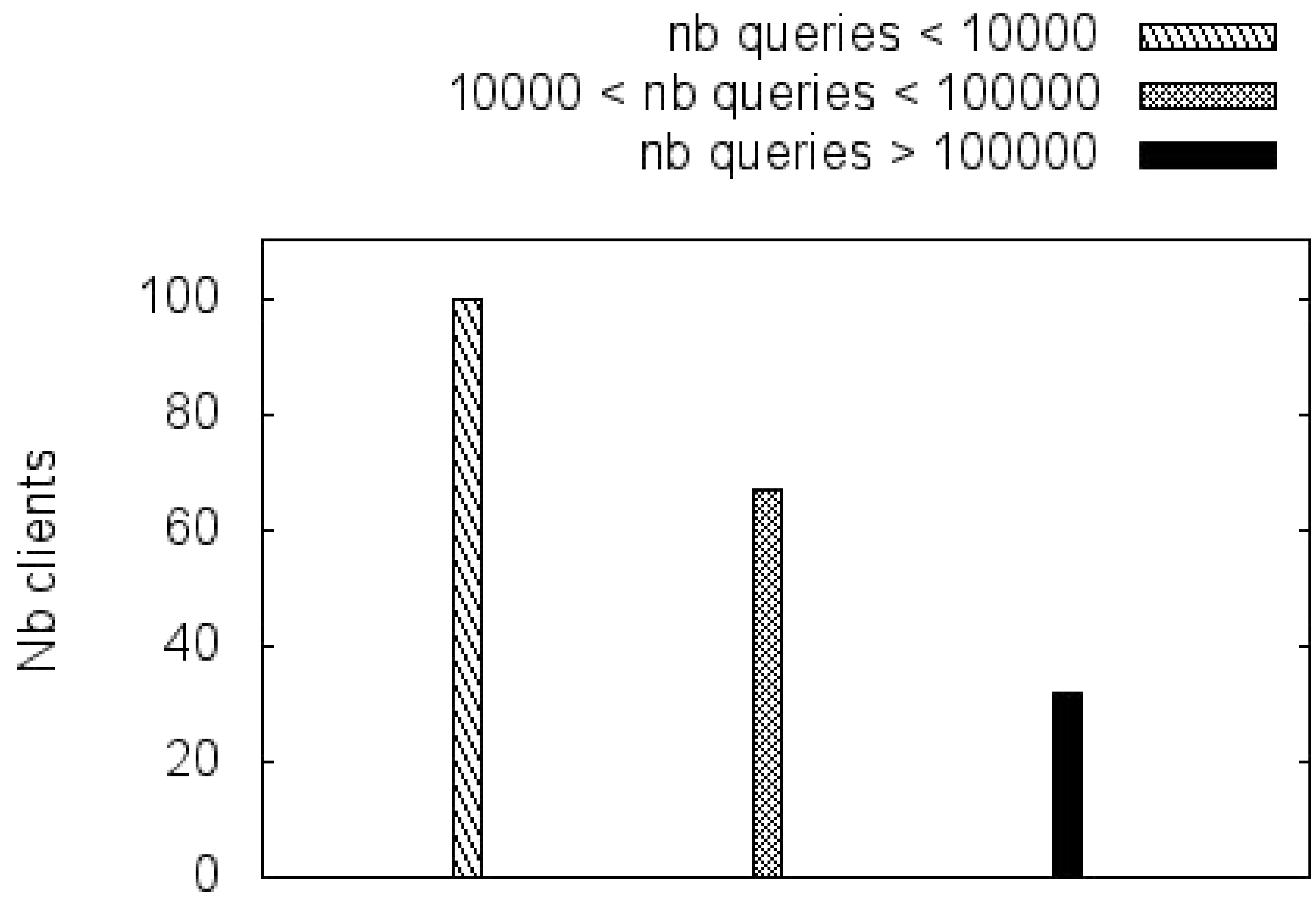
Size of the profile does not impact the number of calls handled by neighborhood, queries use at most 16 differents predicates at once

BSBM 10M, 10 clients, RPS = 4, CON = 9, profile size = 10



Percentage of calls handled by the neighborhood is related to cache size. With a larger cache size we have a bigger decentralized cache

Query load - Is there a hotspot?



Most of the clients handled around 10,000 cache queries and a few handled more than 100,000 cache queries

Query Distribution

- Each client has its own query mix
- Each query mix:
  - is generated randomly
  - has 25 queries based on 12 templates
- Query mix:
  - is the same for both rounds
  - is executed in the same order for both rounds

Shuffle

- Shuffle phases are executed each 10 seconds
- In theory shuffle phases do not impact query processing
  - In implementation, query processing can be interrupted because NodeJs is mono-thread
- Shuffle phase for BSBM 1M:
  - 10 clients ≈ 14 shuffles / client
  - 50 clients ≈ 80 shuffles / client
  - 100 clients ≈ 85 shuffles / client

Experiment

- 2 rounds:
  - **Warmup** → Bootstrap RPS, CON and HTTP cache
  - **Real** → measures are done
- All clients execute the same query mix for both rounds
- All queries are executed in the same order

Overlay size

- Random Peer Sampling:
  - RPS view size is **Log(N)**, where **N** is the **number of peers** in the network
- Cluster Overlay Network:
  - CON view size has been chosen following the guidelines of [5]

[5] M. Bertier and al. The gossple anonymous social network. In *11th International Middleware Conference Middleware 2010* - ACM/IFIP/USENIX, volume 6452 of LNCS, pages 191{211. Springer, 2010.