A Decentralized Architecture for Sharing and Querying Semantic Data

Christian Aebeloe Gabriela Montoya Katja Hose

Aalborg University, Denmark
{caebel,gmontoya,khose}@cs.aau.dk

Linked Data is *often unavailable*

Over *half* the public SPARQL endpoints have *<95%* availability[1]

[1] C. Buil-Aranda et al. **SPARQL Web Querying Infrastructure: Ready for Action?** ISWC 2013

INTRODUCTION

- Interfaces require *considerable resources* to maintain[2]

[2] R. Verborgh et al. **Triple Pattern Fragments: A Low-Cost Knowledge Graph Interface for the Web.** J. Web Semantics 37–38:184–206 2016

[3] T. Grubenmann et al. **Financing the Web of Data with Delayed-Answer Auctions.** WWW 2018

INTRODUCTION

- Interfaces require *considerable resources* to maintain[2]
- *Huge burden* for data providers

[2] R. Verborgh et al. **Triple Pattern Fragments: A Low-Cost Knowledge Graph Interface for the Web.** J. Web Semantics 37–38:184–206 2016

[3] T. Grubenmann et al. **Financing the Web of Data with Delayed-Answer Auctions.** WWW 2018

INTRODUCTION

- Interfaces require *considerable resources* to maintain[2]
- *Huge burden* for data providers

- Recent efforts proposed *monetary incentives*[3]

[2] R. Verborgh et al. **Triple Pattern Fragments: A Low-Cost Knowledge Graph Interface for the Web.** J. Web Semantics 37–38:184–206 2016

[3] T. Grubenmann et al. **Financing the Web of Data with Delayed-Answer Auctions.** WWW 2018

INTRODUCTION

- Interfaces require *considerable resources* to maintain[2]
- *Huge burden* for data providers

- Recent efforts proposed *monetary incentives*[3]
- **We argue, that the burden can be lifted through *decentralization***

[2] R. Verborgh et al. **Triple Pattern Fragments: A Low-Cost Knowledge Graph Interface for the Web.** J. Web Semantics 37–38:184–206 2016

[3] T. Grubenmann et al. **Financing the Web of Data with Delayed-Answer Auctions.** WWW 2018

DECENTRALIZATION TO INCREASE AVAILABILITY

Decentralization has previously been proposed

and evidenced to increase availability of data sources

- *Solid*[4]: Decentralized Personal Online Datastores (PODs)
  - Focuses on data privacy
- *TPF*[5]: Lower computational load on server
  - Single point of entry
- *Fog of Browsers*[6] : Sharing the load in a network of browsers
  - Browsers are relatively unstable, limited resources
- *Ulysses*[7]: Replicated TPFs over multiple servers
  - Fixed set of servers and replicated fragments

---

[4] E. Mansour et al. **A Demonstration of the Solid Platform for Social Web Applications.** WWW 2016

[5] R. Verborgh et al. **Triple Pattern Fragments: A Low-Cost Knowledge Graph Interface for the Web.** J. Web Semantics 37–38:184–206 2016

[6] P. Molli et al. **Semantic Web in the Fog of Browsers.** DeSemWeb 2017

[7] T. Minier et al. **Intelligent Clients for Replicated Triple Pattern Fragments.** ESWC 2018

PEER-TO-PEER SYSTEMS

- *Structured P2P Systems*:
  Organize peers into overlay networks (e.g. Distributed Hash Tables) to decide where to store and find data. Vulnerable to high churn.

PEER-TO-PEER SYSTEMS

- *Structured P2P Systems*:
  Organize peers into overlay networks (e.g. Distributed Hash Tables) to decide where to store and find data. Vulnerable to high churn.
- *Unstructured P2P Systems*:
  Unstructured connections between peers. More reliable during high churn. Allows for more dynamic behavior.

PIQNIC

PIQNIC: a P2p clIent for Query processiNg over semantIC data

- *Unstructured* P2P-based architecture

[8] J. D. Fernández et al. **Binary RDF Representation for Publication and Exchange (HDT).** J. Web Semantics 19:22-41 2013

PIQNIC

PIQNIC: a P2p clIent for Query processiNg over semantIC data

- *Unstructured* P2P-based architecture
- Each client maintains a *local datastore*

[8] J. D. Fernández et al. **Binary RDF Representation for Publication and Exchange (HDT).** J. Web Semantics 19:22-41 2013

PIQNIC

PIQNIC: a P2p clIent for Query processiNg over semantIC data

- *Unstructured* P2P-based architecture
- Each client maintains a *local datastore*
- *HDT*[8] backend

[8] J. D. Fernández et al. **Binary RDF Representation for Publication and Exchange (HDT).** J. Web Semantics 19:22-41 2013

PIQNIC

PIQNIC: a P2p clIent for Query processiNg over semantIC data

- *Unstructured* P2P-based architecture
- Each client maintains a *local datastore*
- *HDT*[8] backend
- Queryable through *any node* in the network

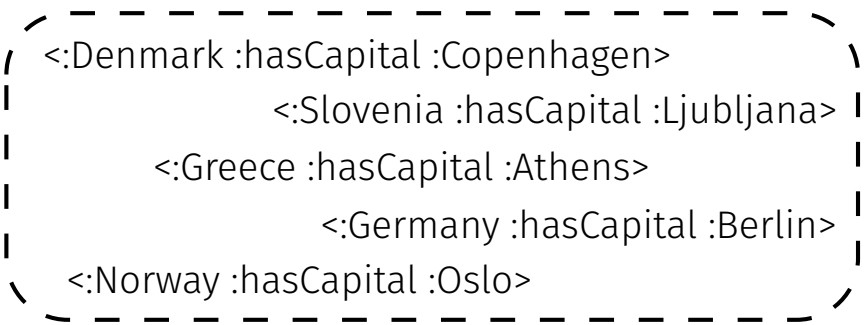
[8] J. D. Fernández et al. **Binary RDF Representation for Publication and Exchange (HDT).** J. Web Semantics 19:22-41 2013

DATA MANAGEMENT

Fragment

Let $\mathcal{G}_N$ be a knowledge graph that includes all RDF triples in a PIQNIC-network. A fragment f is a 4-tuple $f = \langle T, N, u, i \rangle$ where

- T is a finite set of RDF triples, and $T \subseteq \mathcal{G}_N$,
- N is a set of PIQNIC nodes containing the fragment,
- u is a URI/IRI that identifies the fragment, and
- i is an identification function that determines whether the fragment contains triples matching a given triple pattern.

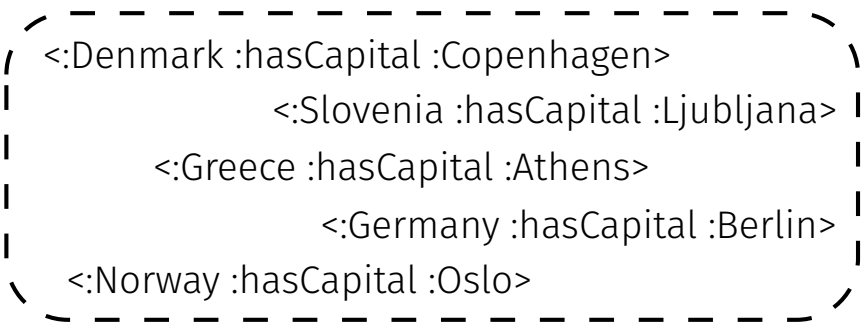


DATA MANAGEMENT

Fragment

Let $\mathcal{G}_N$ be a knowledge graph that includes all RDF triples in a PIQNIC-network. A fragment f is a 4-tuple $f = \langle T, N, u, i \rangle$ where

- T is a finite set of RDF triples, and $T \subseteq \mathcal{G}_N$,
- N is a set of PIQNIC nodes containing the fragment,
- u is a URI/IRI that identifies the fragment, and
- i is an identification function that determines whether the fragment contains triples matching a given triple pattern.



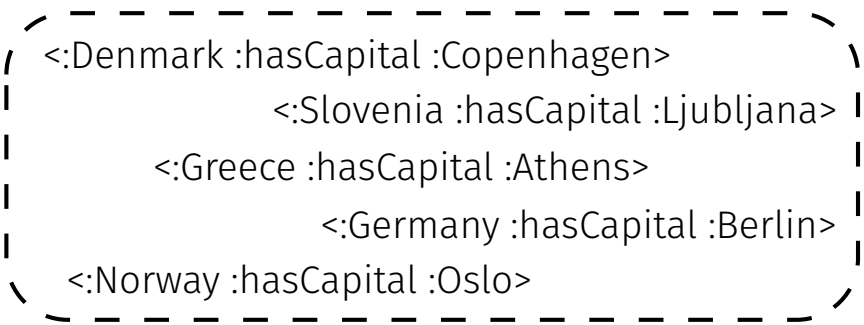
Predicate-based identification function

DATA MANAGEMENT

Fragment

Let $\mathcal{G}_N$ be a knowledge graph that includes all RDF triples in a PIQNIC-network. A fragment f is a 4-tuple $f = \langle T, N, u, i \rangle$ where

- T is a finite set of RDF triples, and $T \subseteq \mathcal{G}_N$,
- N is a set of PIQNIC nodes containing the fragment,
- u is a URI/IRI that identifies the fragment, and
- i is an identification function that determines whether the fragment contains triples matching a given triple pattern.



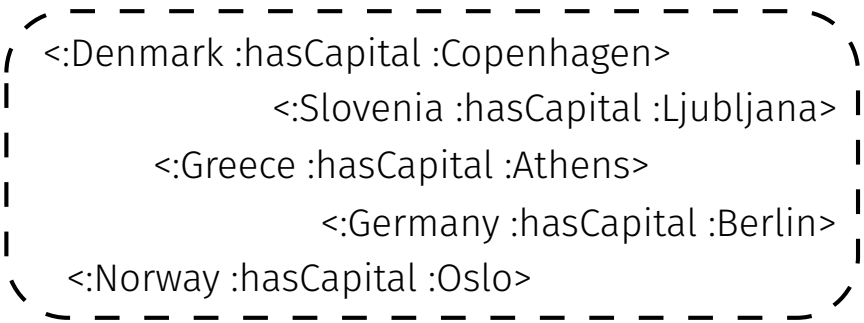
Predicate-based identification function

<?v1 :**hasCapital** :Ljubljana> True

Fragment

Let $\mathcal{G}_N$ be a knowledge graph that includes all RDF triples in a PIQNIC-network. A fragment f is a 4-tuple $f = \langle T, N, u, i \rangle$ where

- T is a finite set of RDF triples, and $T \subseteq \mathcal{G}_N$,
- N is a set of PIQNIC nodes containing the fragment,
- u is a URI/IRI that identifies the fragment, and
- i is an identification function that determines whether the fragment contains triples matching a given triple pattern.



Predicate-based identification function

<?v1 :**hasCapital** :Ljubljana> True

<?v2 :**isLeaderOf** :Slovenia> False

DATA MANAGEMENT

Dataset

A dataset D is a triple $D = \langle F, u, o \rangle$ where

- F is a set of fragments,
- u is a URI/IRI that identifies the dataset, and
- o is an identifier of the "owner" node, i.e., the node that uploads F to the network.

<:Denmark :hasCapital :Copenhagen>
<:Slovenia :hasCapital :Ljubljana>
<:Greece :hasCapital :Athens>
<:Germany :hasCapital :Berlin>
<:Norway :hasCapital :Oslo>

<:Oslo :locatedIn :Norway>
<:Athens :locatedIn :Greece>

<:Frank_Jensen :isMayorOf :Copenhagen>
<:Zoran_Jankovic :isMayorOf :Ljubljana>
<:Michael_Muller :isMayorOf :Berlin>

<:Ljubljana :hosted :ESWC>
<:Athens :hosted :ESWC>

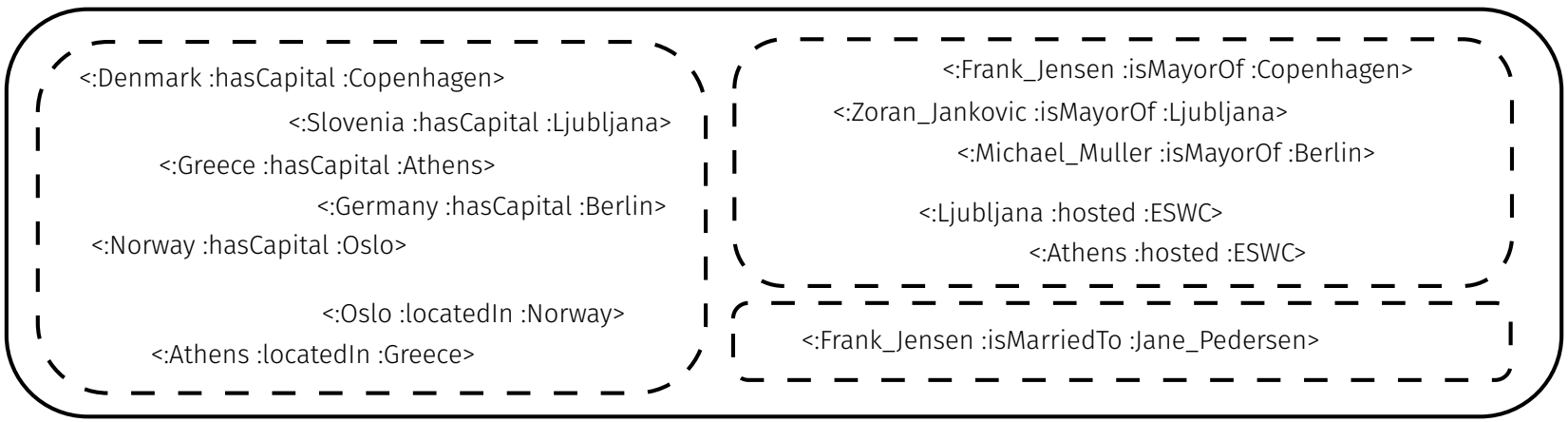
<:Frank_Jensen :isMarriedTo :Jane_Pedersen>

DATA MANAGEMENT

Dataset

A dataset D is a triple $D = \langle F, u, o \rangle$ where

- F is a set of fragments,
- u is a URI/IRI that identifies the dataset, and
- o is an identifier of the "owner" node, i.e., the node that uploads F to the network.



FRAGMENTATION

Given a knowledge graph, we would like to create a dataset

Fragmentation Function

A fragmentation function $\mathcal{F}$ is a function that, when applied to a knowledge graph $\mathcal{G}$, creates a set of fragments $F = \mathcal{F}(\mathcal{G})$, i.e., $\mathcal{F}(\mathcal{G}) : \mathcal{G} \mapsto 2^{\mathcal{G}}$.

Predicate-Based Fragmentation Function

$$\mathcal{F}_P(\mathcal{G}) = \{F_p \mid \exists t \in \mathcal{G} : p_t = p \wedge (\forall t' \in \mathcal{G})[p_{t'} = p] : t' \in F_p\}$$

Knowledge Graph $\mathcal{G}_E$

<:Denmark :hasCapital :Copenhagen>
<:Slovenia :hasCapital :Ljubljana>
<:Greece :hasCapital :Athens>
<:Germany :hasCapital :Berlin>
<:Norway :hasCapital :Oslo>

<:Oslo :locatedIn :Norway>
<:Athens :locatedIn :Greece>

<:Frank_Jensen :isMayorOf :Copenhagen>
<:Zoran_Jankovic :isMayorOf :Ljubljana>
<:Michael_Muller :isMayorOf :Berlin>

<:Ljubljana :hosted :ESWC>
<:Athens :hosted :ESWC>

<:Frank_Jensen :isMarriedTo :Jane_Pedersen>

FRAGMENTATION

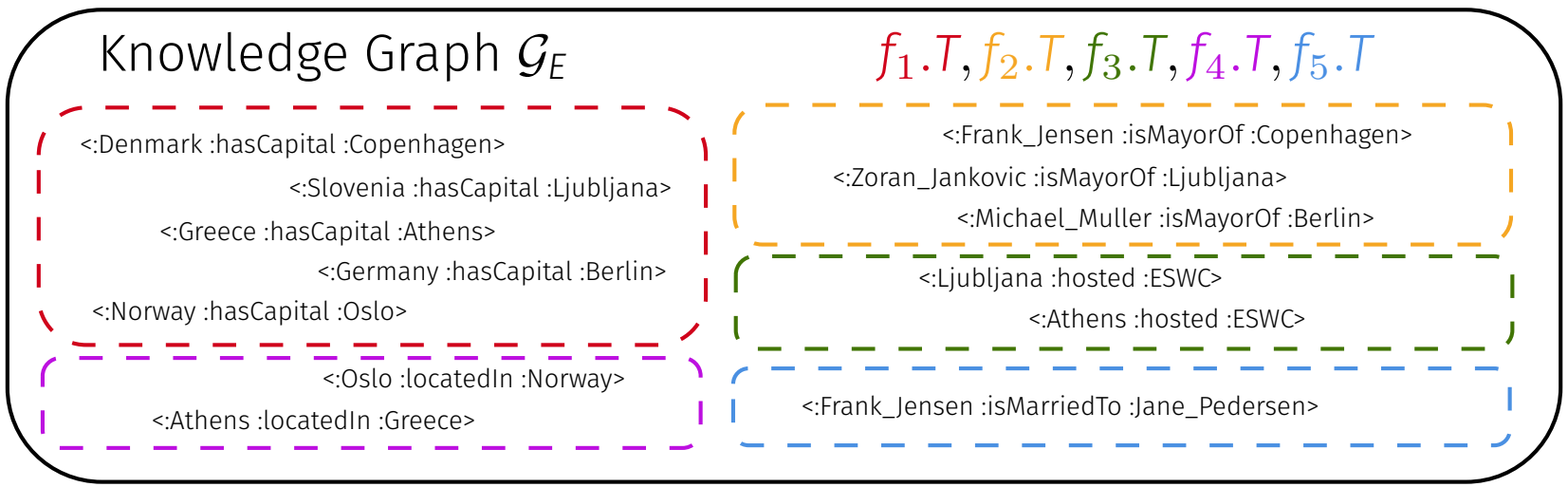
Given a knowledge graph, we would like to create a dataset

Fragmentation Function

A fragmentation function $\mathcal{F}$ is a function that, when applied to a knowledge graph $\mathcal{G}$, creates a set of fragments $F = \mathcal{F}(\mathcal{G})$, i.e., $\mathcal{F}(\mathcal{G}) : \mathcal{G} \mapsto 2^{\mathcal{G}}$.

Predicate-Based Fragmentation Function

$$\mathcal{F}_P(\mathcal{G}) = \{F_p \mid \exists t \in \mathcal{G} : p_t = p \wedge (\forall t' \in \mathcal{G})[p_{t'} = p] : t' \in F_p\}$$



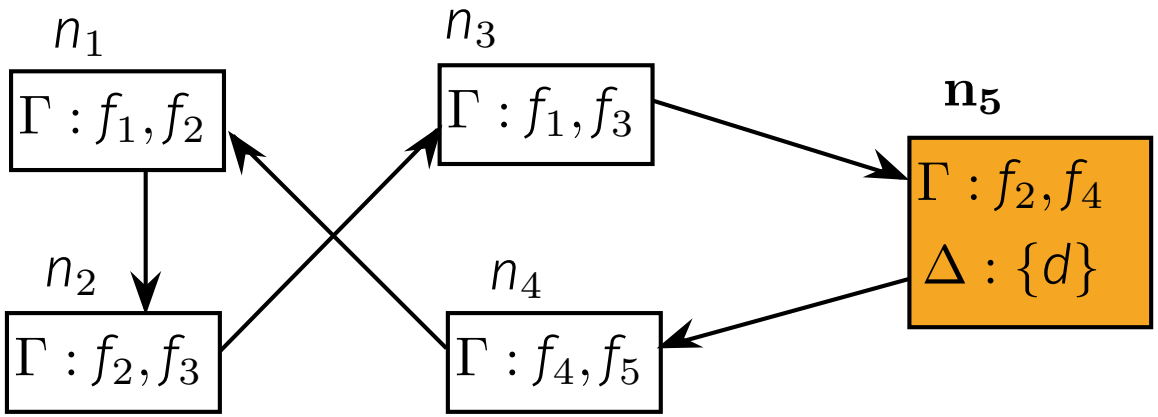
NETWORK ARCHITECTURE

P2P network: clients are also servers

Node

A node n is a triple $n = \langle \Gamma, \Delta, N \rangle$ where

- Γ is the set of fragments located on the node,
- Δ is a set of datasets owned by the node, and
- N is a set of so-called neighbor nodes in the network.



NEIGHBORHOOD MANAGEMENT

We want both *related* and *random* neighbors

obtained through *shuffles*, i.e. we want to shuffle away the *least* related neighbors

Fragment Joinability

Fragments f_1 and f_2 are *joinable*, denoted $f_1 \perp\!\!\!\perp f_2$, iff for at least one triple $t_1 \in f_1$ there exists a triple $t_2 \in f_2$, s.t. $\{s_{t_1}, o_{t_1}\} \cap \{s_{t_2}, o_{t_2}\} \neq \emptyset$.

$$Join(n_1, n_2) = \{f_1 \in n_1.\Gamma \mid \exists f_2 \in n_2.\Gamma : f_1 \perp\!\!\!\perp f_2 \wedge f_1.u \neq f_2.u\}$$

NEIGHBORHOOD MANAGEMENT

We want both *related* and *random* neighbors

obtained through *shuffles*, i.e. we want to shuffle away the *least* related neighbors

Fragment Joinability

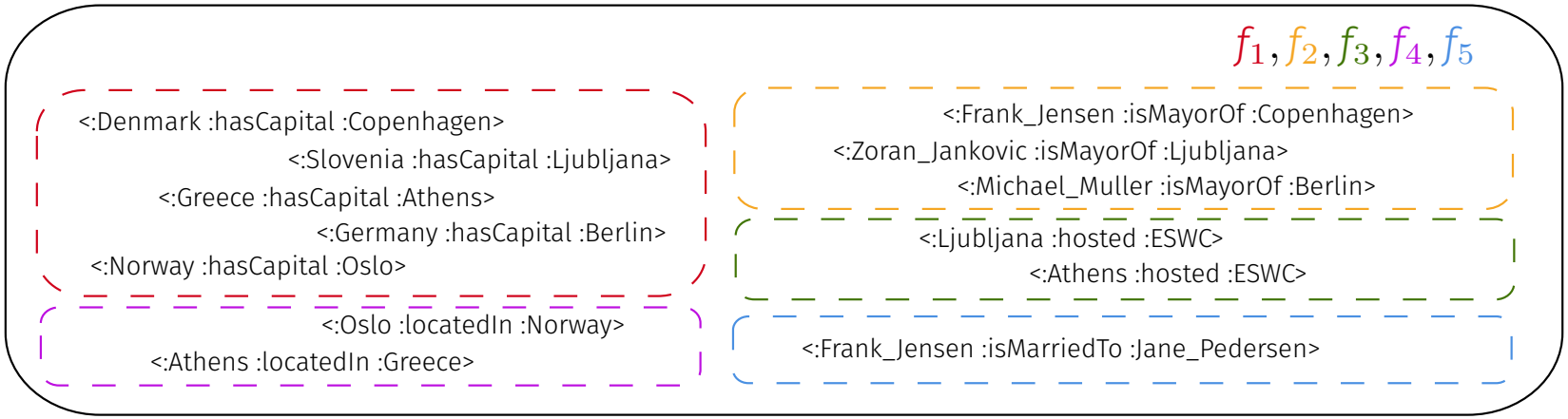
Fragments f_1 and f_2 are *joinable*, denoted $f_1 \perp\!\!\!\perp f_2$, iff for at least one triple $t_1 \in f_1$ there exists a triple $t_2 \in f_2$, s.t. $\{s_{t_1}, o_{t_1}\} \cap \{s_{t_2}, o_{t_2}\} \neq \emptyset$.

$$Join(n_1, n_2) = \{f_1 \in n_1.\Gamma \mid \exists f_2 \in n_2.\Gamma : f_1 \perp\!\!\!\perp f_2 \wedge f_1.u \neq f_2.u\}$$

$$Rel(n) = \underset{R \subseteq n.N}{\arg\min} \sum_{n_i \in n.N} \frac{|Join(n, n_i)|}{|n.\Gamma|} \qquad s.t.\ |R| = k$$

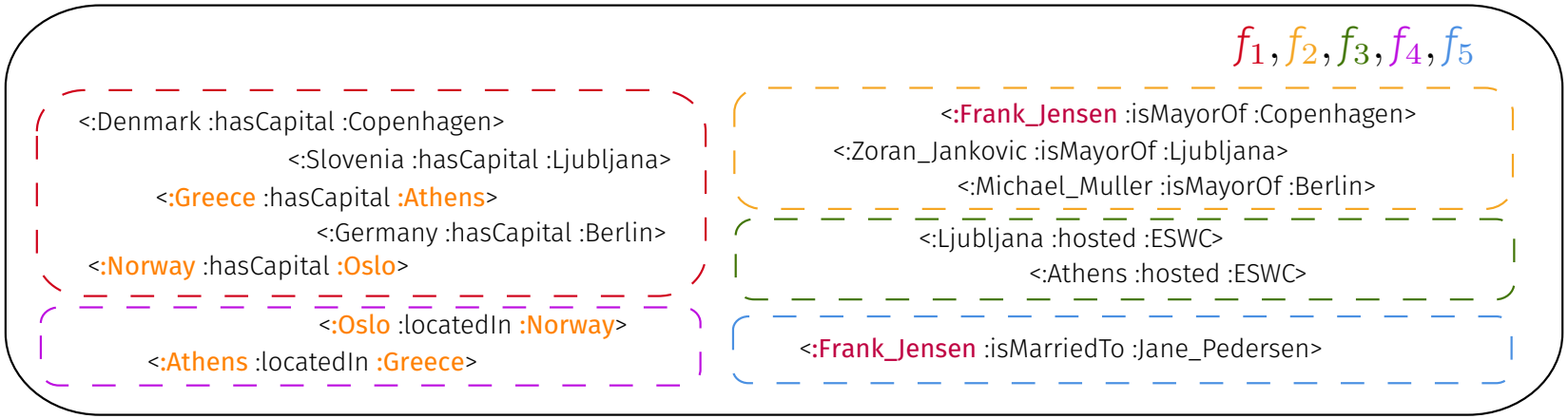
We select the k *least* related neighbors to shuffle

NEIGHBORHOOD MANAGEMENT



$r_i = |Join(n_4, n_i)| \div |n_4.\Gamma|$, n_4 shuffles with n_2, shuffle length = 1

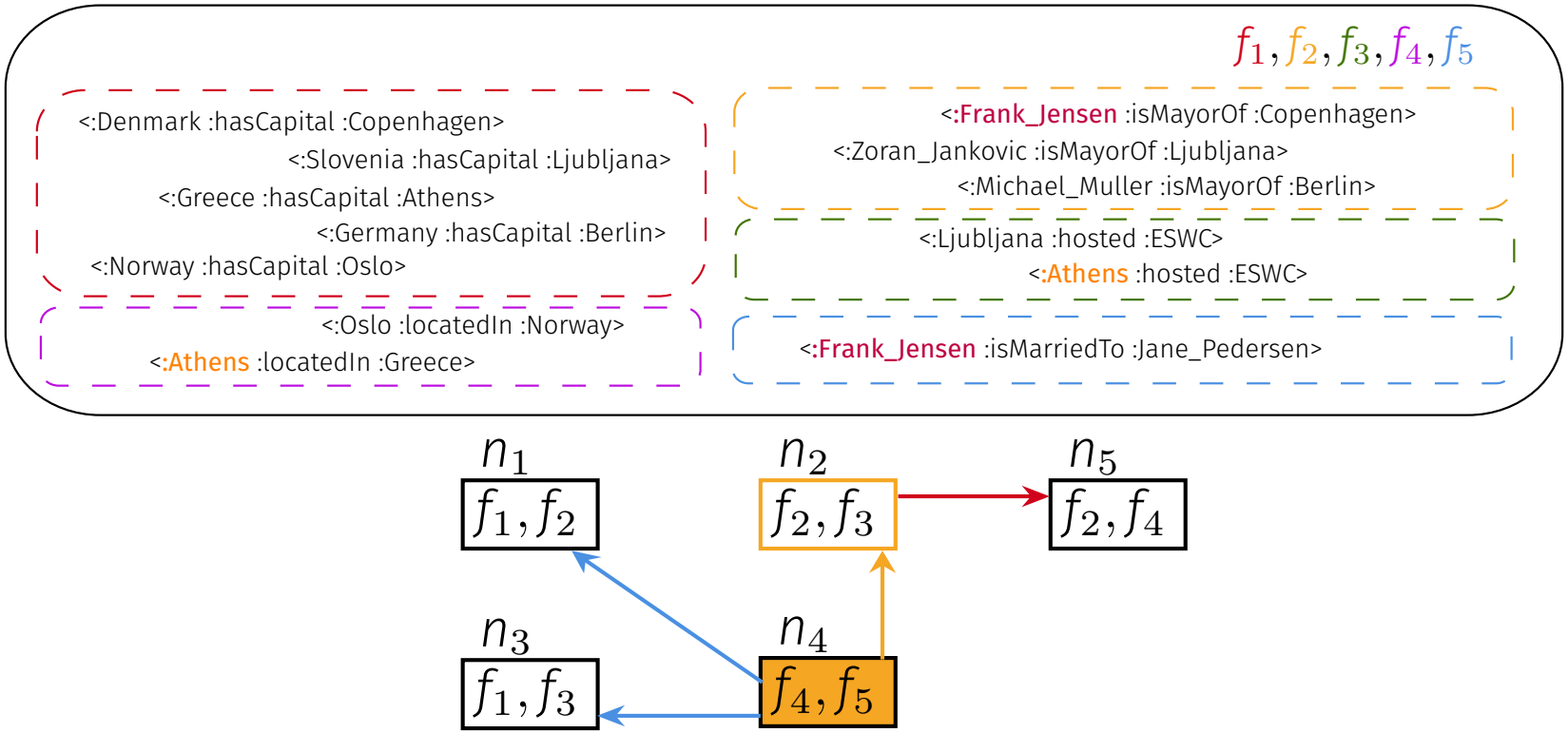
NEIGHBORHOOD MANAGEMENT



$r_i = |Join(n_4, n_i)| \div |n_4.\Gamma|$, n_4 shuffles with n_2, shuffle length = 1

- n_1: $f_4 \perp\!\!\!\perp f_1$ and $f_5 \perp\!\!\!\perp f_2$, then $r_1 = 2/2 = 1$

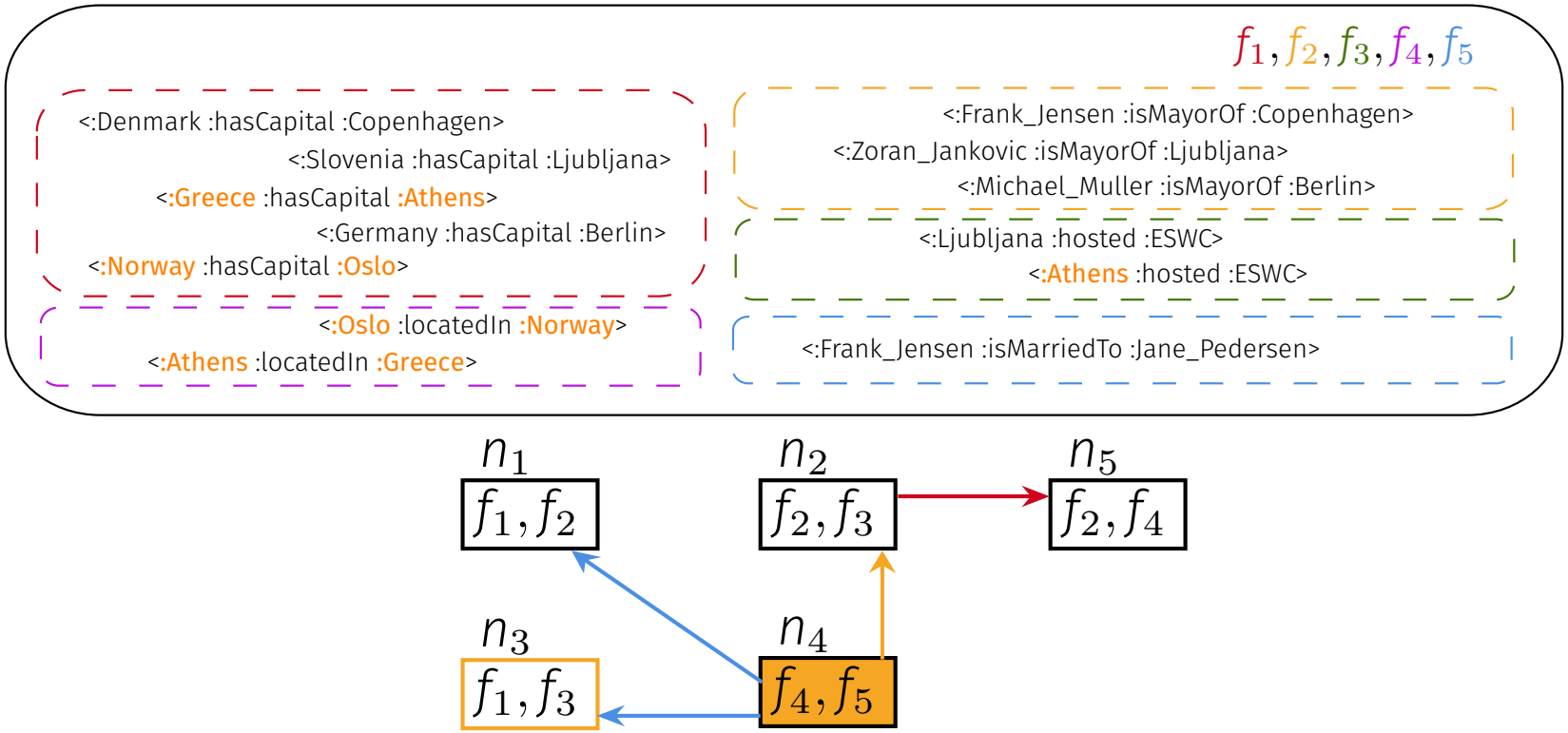
NEIGHBORHOOD MANAGEMENT



$r_i = |Join(n_4, n_i)| \div |n_4.\Gamma|$, n_4 shuffles with n_2, shuffle length = 1

- n_1: $f_4 \perp\!\!\!\perp f_1$ and $f_5 \perp\!\!\!\perp f_2$, then $r_1 = 2/2 = 1$
- n_2: $f_4 \perp\!\!\!\perp f_3$ and $f_5 \perp\!\!\!\perp f_2$, then $r_2 = 2/2 = 1$

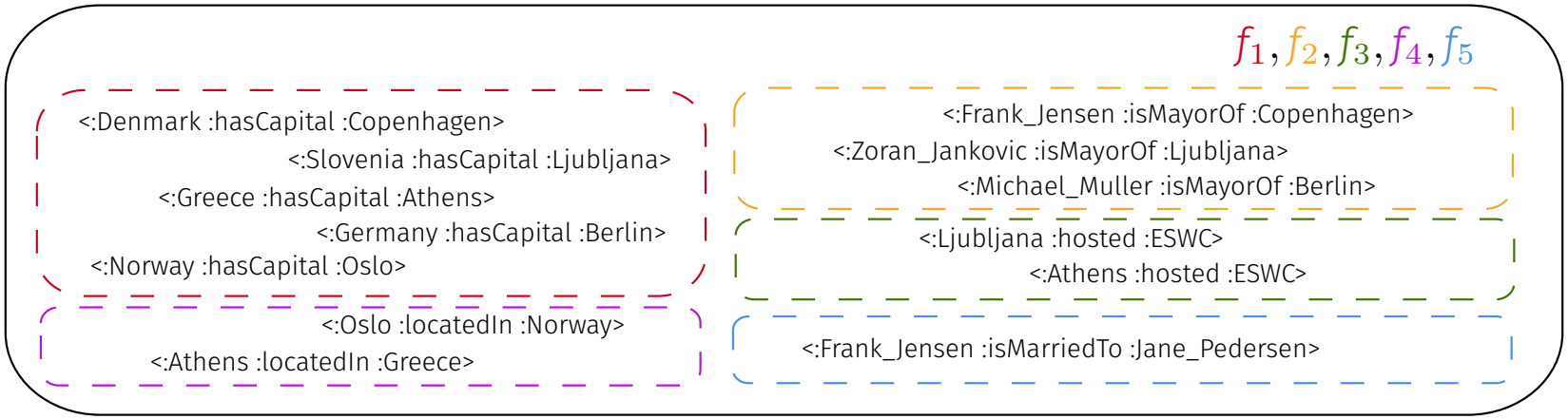
NEIGHBORHOOD MANAGEMENT

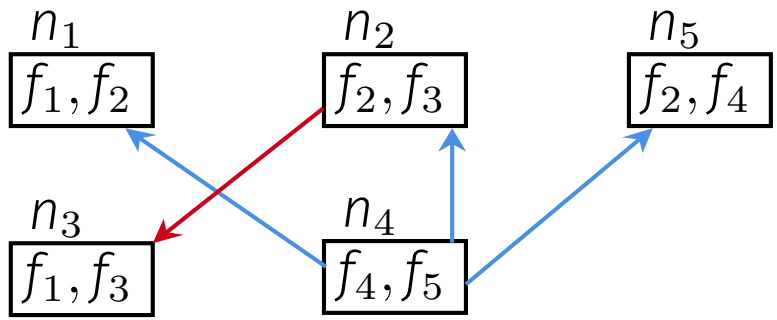


$r_i = |Join(n_4, n_i)| \div |n_4.\Gamma|$, n_4 shuffles with n_2, shuffle length = 1

- n_1: $f_4 \perp\!\!\!\perp f_1$ and $f_5 \perp\!\!\!\perp f_2$, then $r_1 = 2/2 = 1$
- n_2: $f_4 \perp\!\!\!\perp f_3$ and $f_5 \perp\!\!\!\perp f_2$, then $r_2 = 2/2 = 1$
- n_3: $f_4 \perp\!\!\!\perp f_1$, $f_4 \perp\!\!\!\perp f_3$, $f_5 \not\perp\!\!\!\perp f_1$ and $f_5 \not\perp\!\!\!\perp f_3$, then $r_3 = 1/2 = 0.5$

NEIGHBORHOOD MANAGEMENT

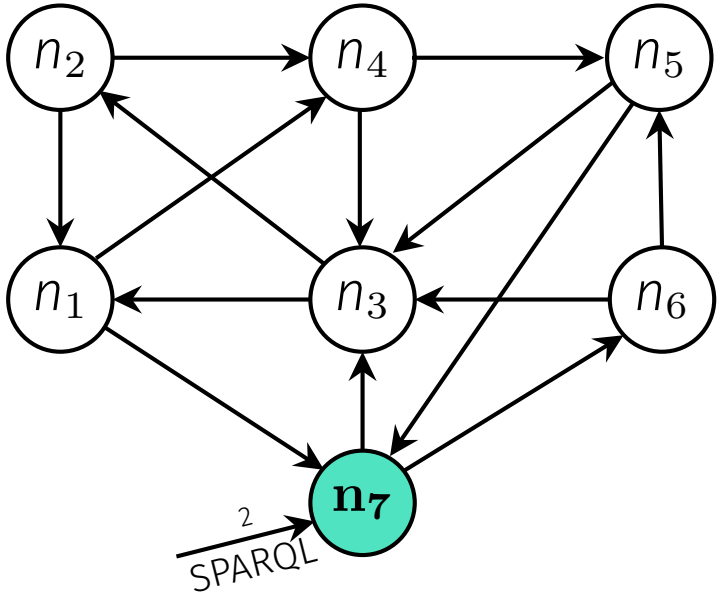




$r_i = |Join(n_4, n_i)| \div |n_4.\Gamma|$, n_4 shuffles with n_2, shuffle length = 1

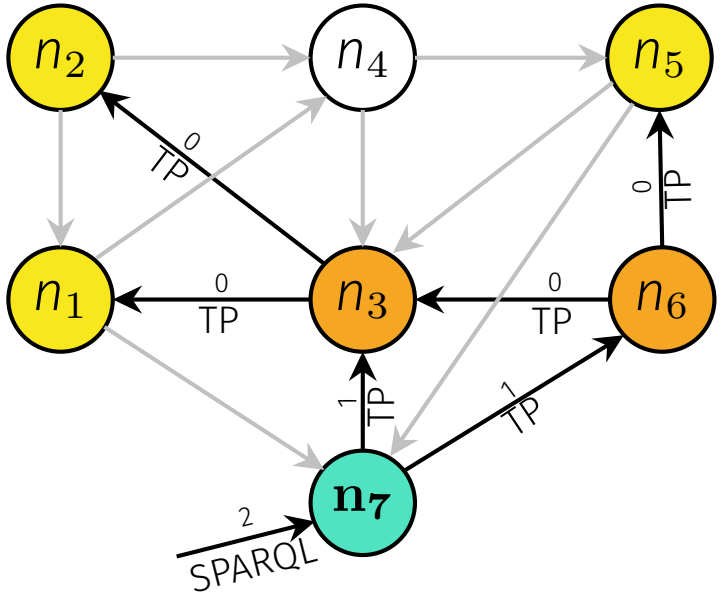
- n_1: $f_4 \perp\!\!\!\perp f_1$ and $f_5 \perp\!\!\!\perp f_2$, then $r_1 = 2/2 = 1$
- n_2: $f_4 \perp\!\!\!\perp f_3$ and $f_5 \perp\!\!\!\perp f_2$, then $r_2 = 2/2 = 1$
- n_3: $f_4 \perp\!\!\!\perp f_1$, $f_4 \perp\!\!\!\perp f_3$, $f_5 \not\perp\!\!\!\perp f_1$ and $f_5 \not\perp\!\!\!\perp f_3$, then $r_3 = 1/2 = 0.5$

QUERY PROCESSING



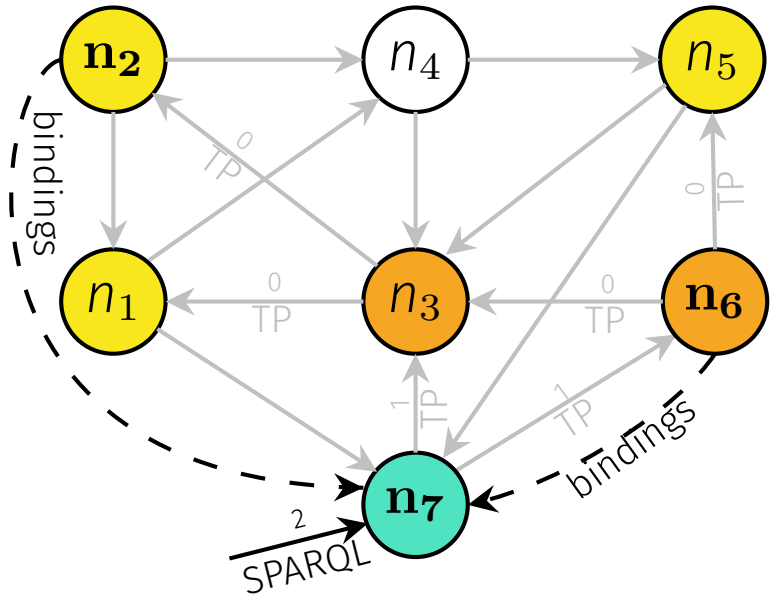
- Query processing follows a flooding approach:
  - i.e. each triple pattern is *flooded* through the network for a specified amount of steps (Time-To-Live)

QUERY PROCESSING



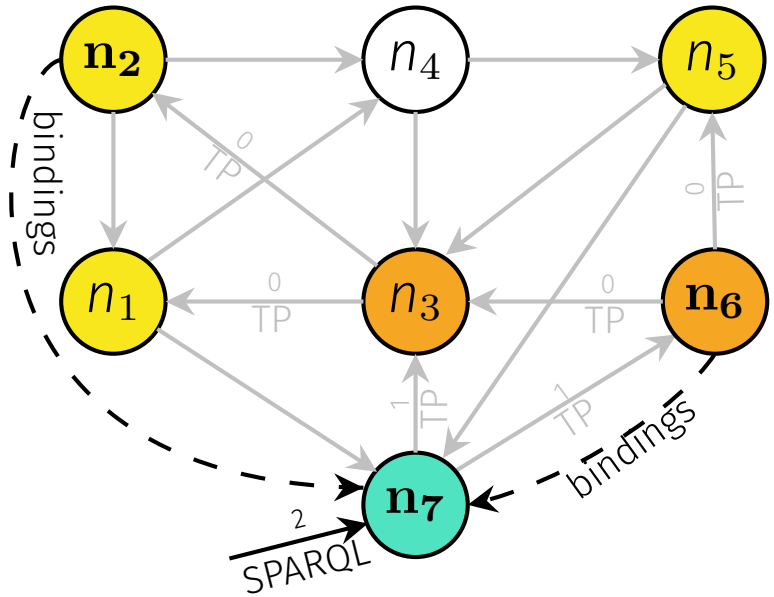
- Query processing follows a flooding approach:
  - i.e. each triple pattern is *flooded* through the network for a specified amount of steps (Time-To-Live)

QUERY PROCESSING



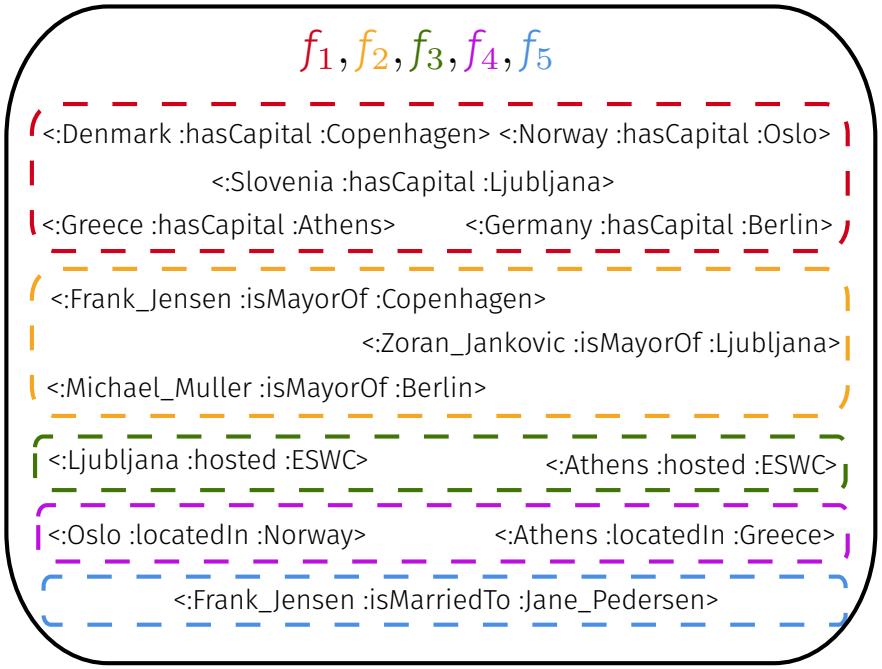
- Query processing follows a flooding approach:
  - i.e. each triple pattern is *flooded* through the network for a specified amount of steps (Time-To-Live)

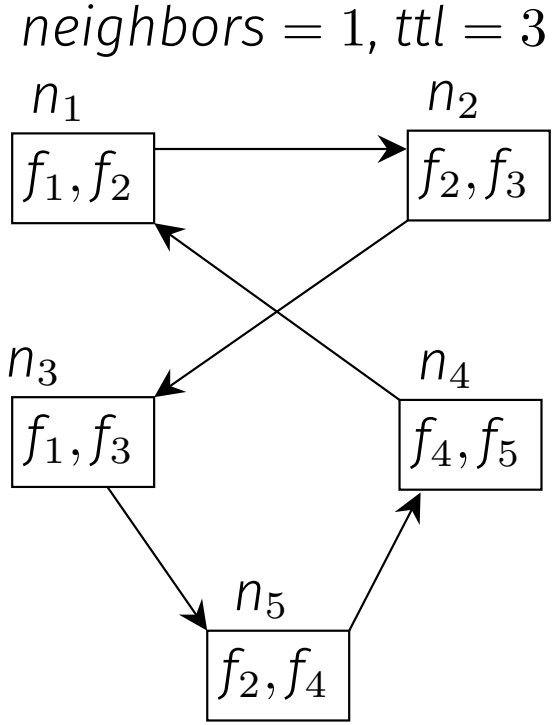
QUERY PROCESSING



- Query processing follows a flooding approach:
  - i.e. each triple pattern is *flooded* through the network for a specified amount of steps (Time-To-Live)
- We implemented three approaches:
  1. *Single*: Previous bindings are flooded one at a time
  2. *Bulk*: Previous bindings are flooded as bulks
  3. *Full*: Only the original triple patterns are flooded

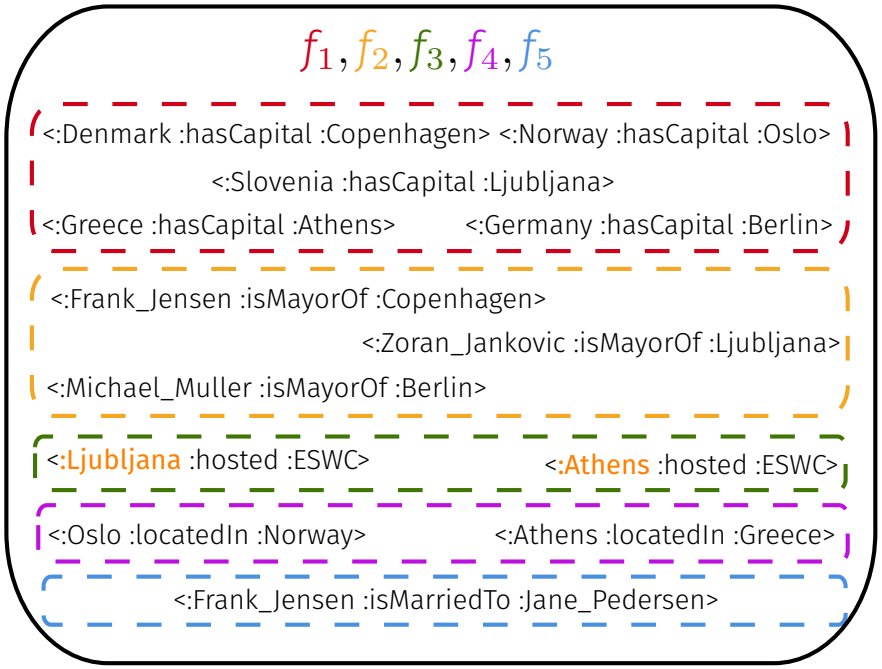
QUERY PROCESSING - SINGLE

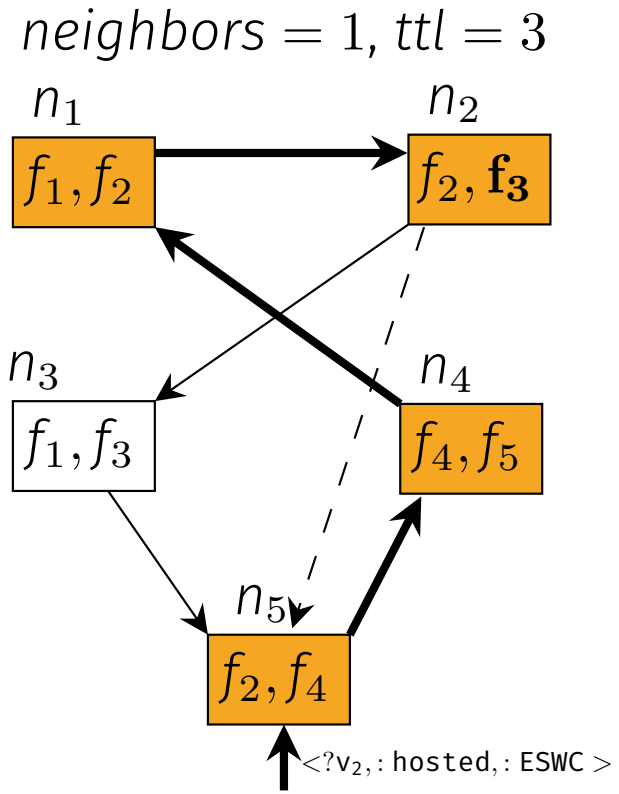





```
SELECT DISTINCT * WHERE {
  ?v1 :hasCapital ?v2 .   (tp1)
  ?v2 :hosted :ESWC       (tp2)
}
```

QUERY PROCESSING - SINGLE



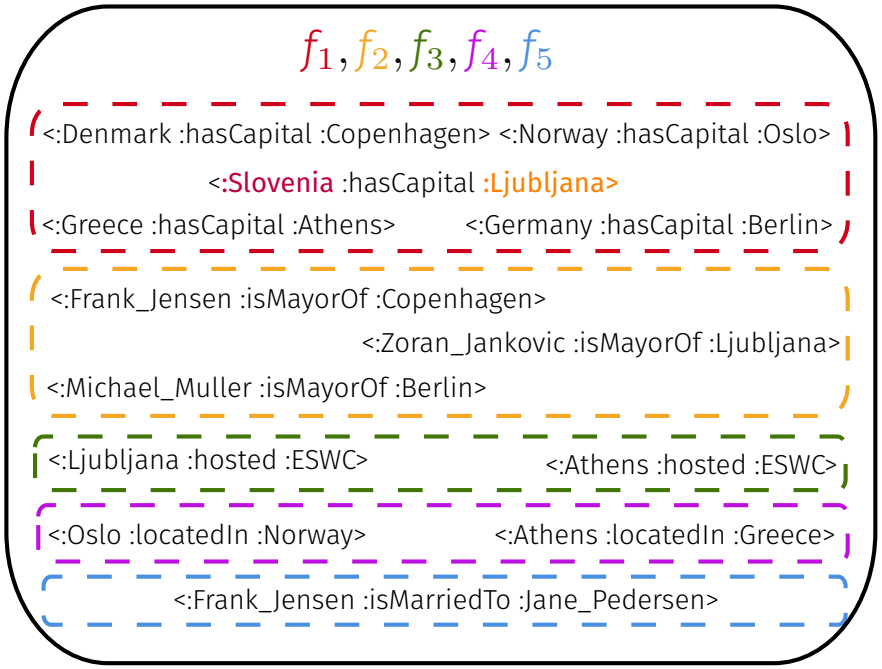


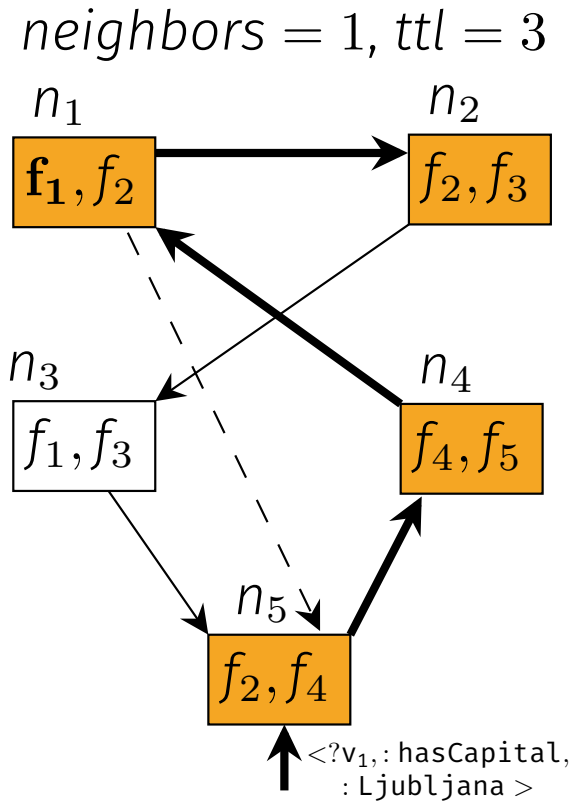

```
SELECT DISTINCT * WHERE {
  ?v1 :hasCapital ?v2 .   (tp1)
  ?v2 :hosted :ESWC       (tp2)
}
```

$\mathbf{tp_2}$:

$\{v_2 =: Ljubljana\}, \{v_2 =: Athens\}$

QUERY PROCESSING - SINGLE






```
SELECT DISTINCT * WHERE {
  ?v1 :hasCapital ?v2 .   (tp1)
  ?v2 :hosted :ESWC       (tp2)
}
```

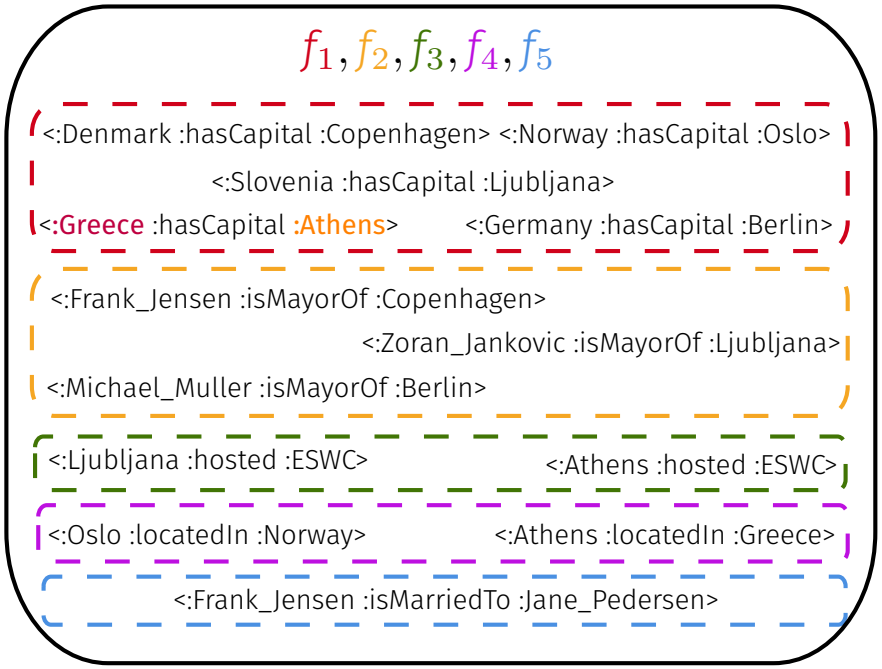
$\mathbf{tp_2}$:

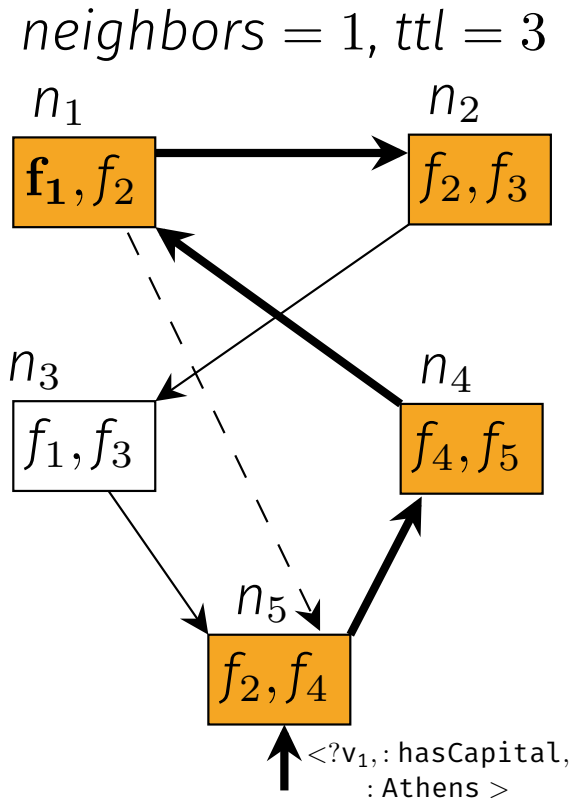
$\{\mathbf{v_2} =: \mathbf{Ljubljana}\}$,

$\{v_2 =: Athens\}$

$\mathbf{tp_2} \bowtie \mathbf{tp_1}$:

QUERY PROCESSING - SINGLE






```
SELECT DISTINCT * WHERE {
  ?v1 :hasCapital ?v2 .   (tp1)
  ?v2 :hosted :ESWC       (tp2)
}
```

$\mathbf{tp_2}$:

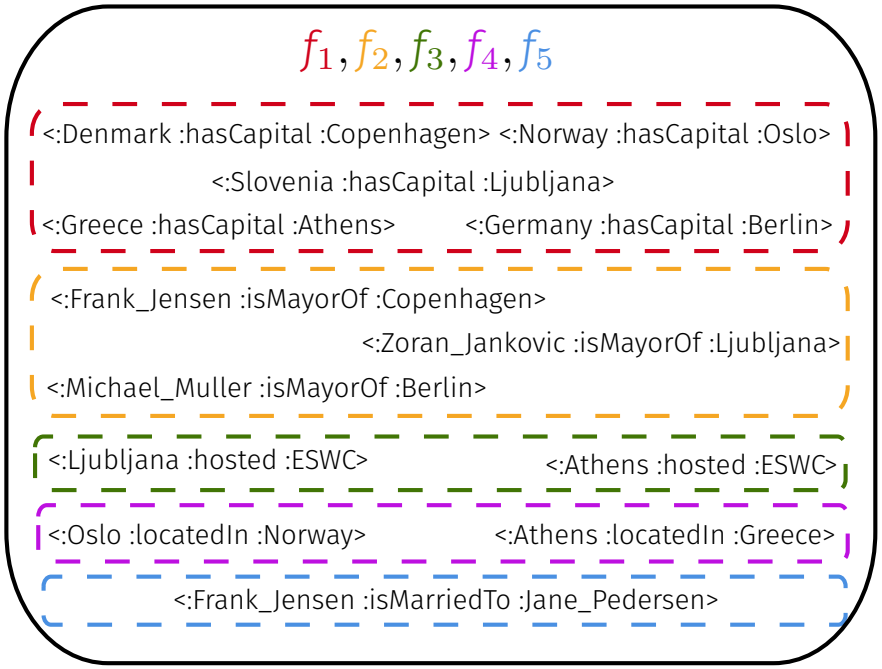
$\{v_2 =: Ljubljana\}$,

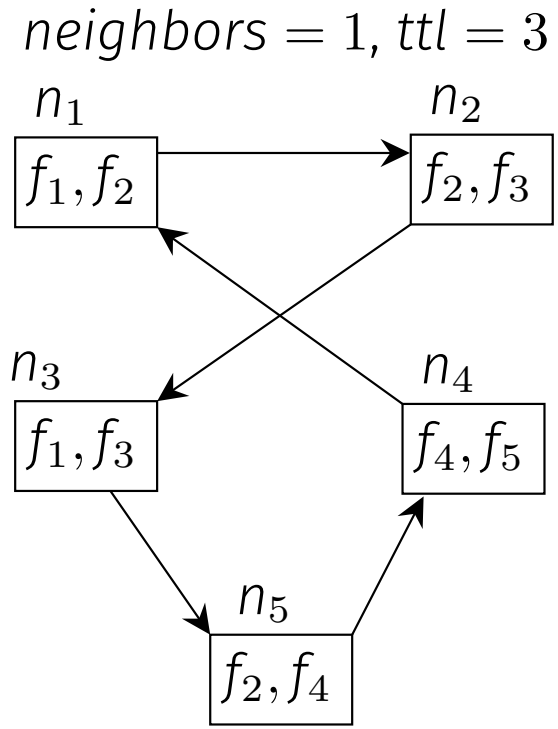
$\{\mathbf{v_2 =: Athens}\}$

$\mathbf{tp_2 \bowtie tp_1}$:

$\{v_2 =: Ljubljana, v_1 =: Slovenia\}$

QUERY PROCESSING - SINGLE






```
SELECT DISTINCT * WHERE {
  ?v1 :hasCapital ?v2 .   (tp1)
  ?v2 :hosted :ESWC       (tp2)
}
```

$\mathbf{tp_2}$:

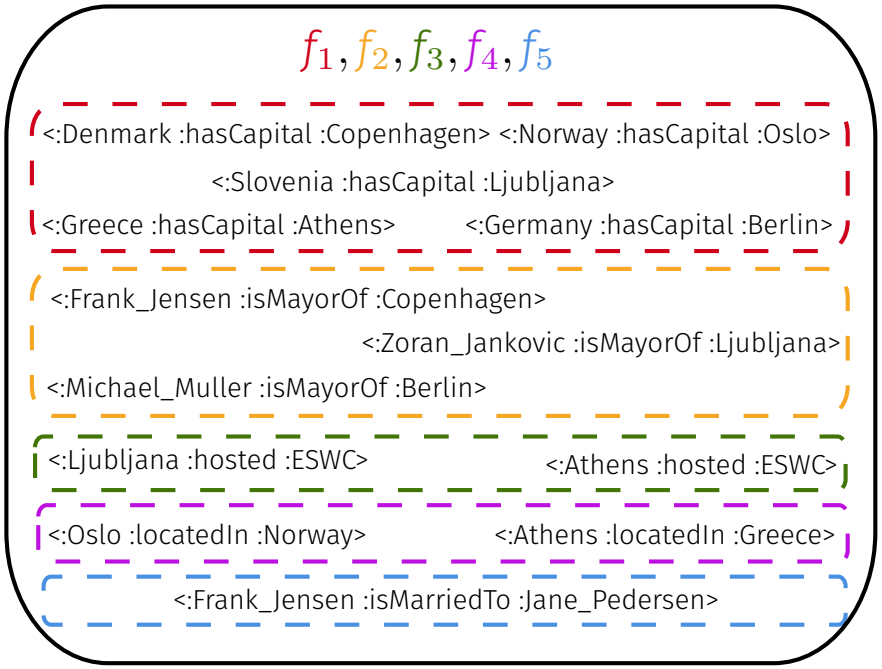
$\{v_2 =: Ljubljana\}, \{v_2 =: Athens\}$

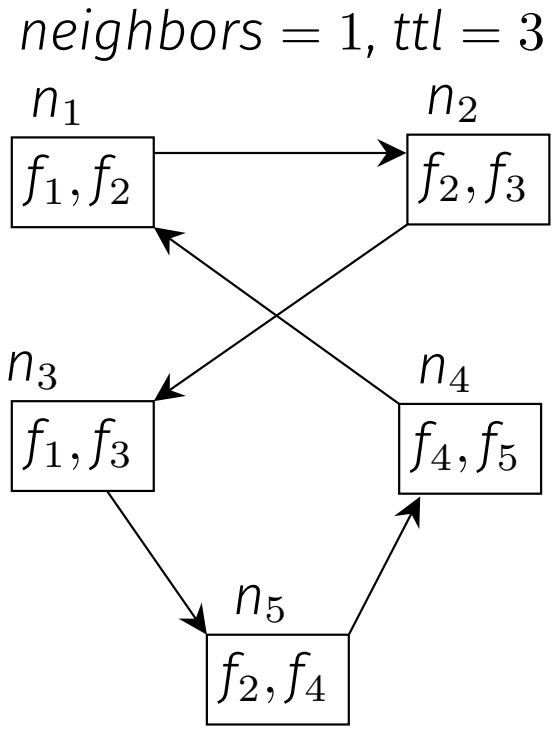
$\mathbf{tp_2} \bowtie \mathbf{tp_1}$:

$\{v_2 =: Ljubljana, v_1 =: Slovenia\}$

$\{v_2 =: Athens, v_1 =: Greece\}$

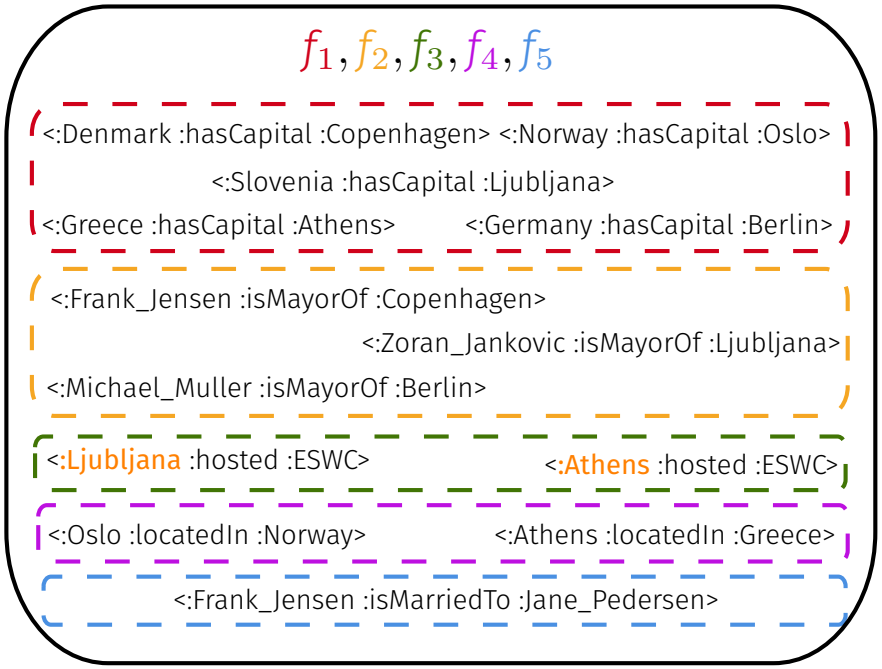
QUERY PROCESSING - BULK

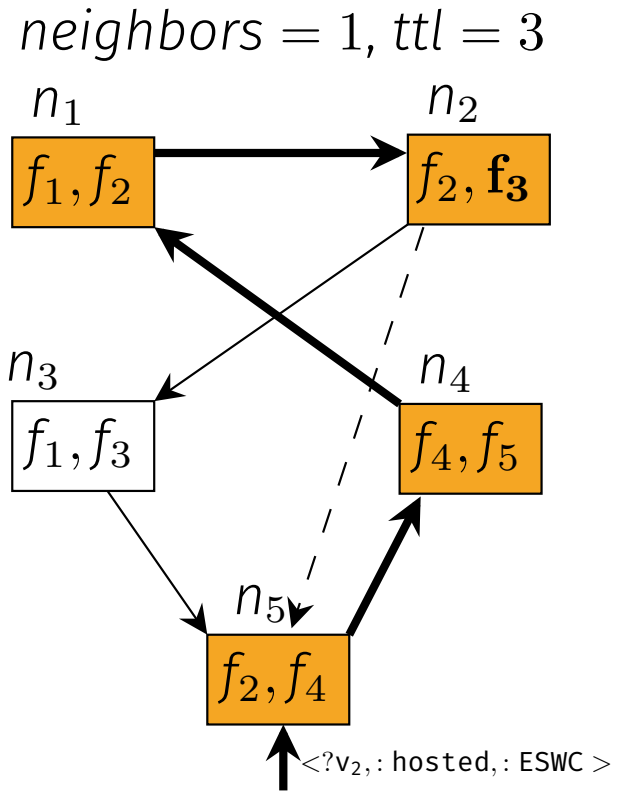





```
SELECT DISTINCT * WHERE {
  ?v1 :hasCapital ?v2 .   (tp1)
  ?v2 :hosted :ESWC       (tp2)
}
```

QUERY PROCESSING - BULK



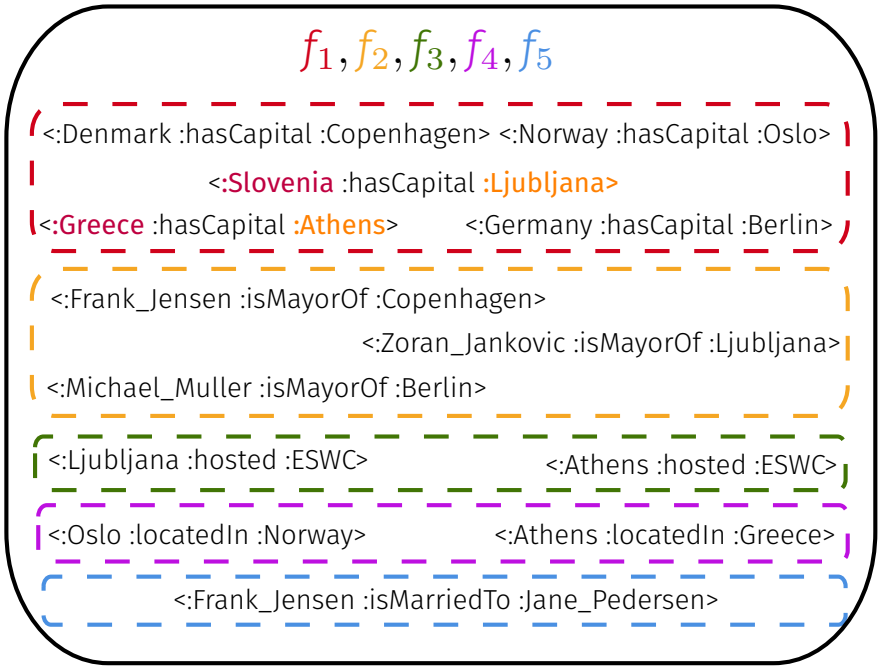


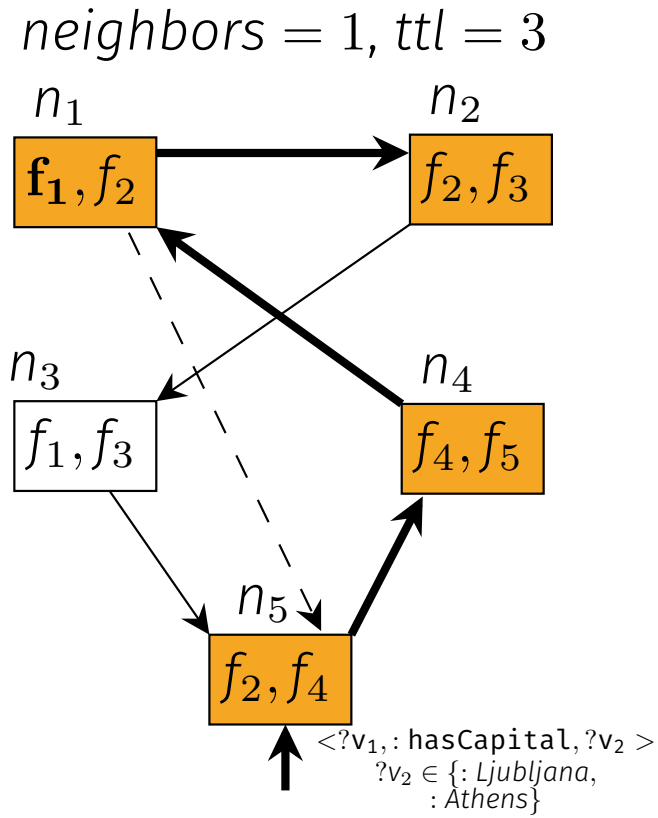

```
SELECT DISTINCT * WHERE {
  ?v1 :hasCapital ?v2 .   (tp1)
  ?v2 :hosted :ESWC       (tp2)
}
```

$\mathbf{tp_2}$:

$\{v_2 =: Ljubljana\}, \{v_2 =: Athens\}$

QUERY PROCESSING - BULK

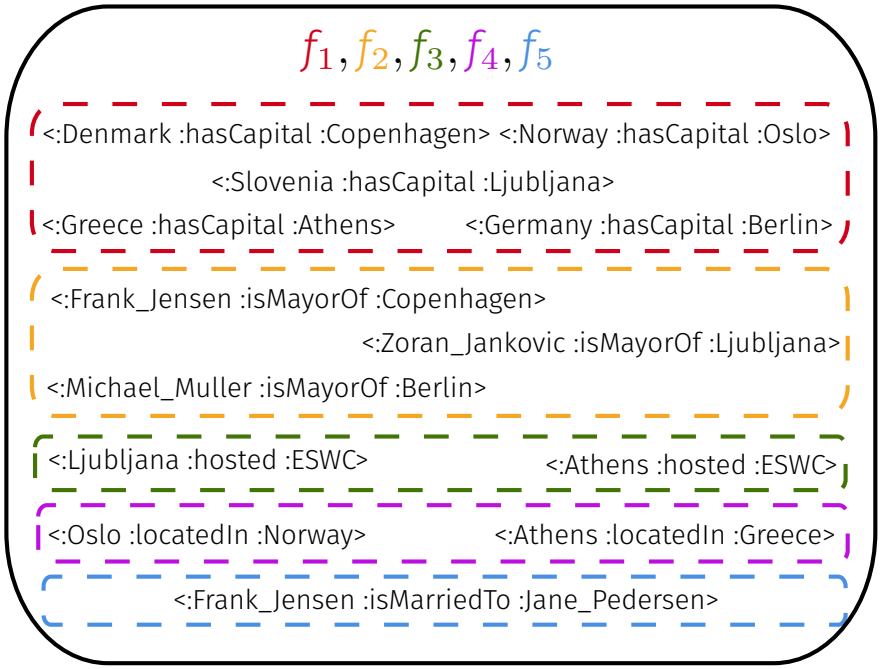


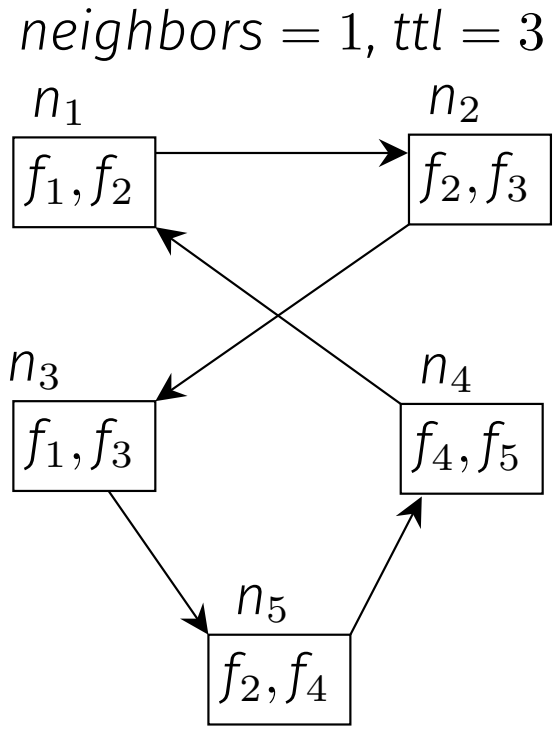



```
SELECT DISTINCT * WHERE {
  ?v1 :hasCapital ?v2 .   (tp1)
  ?v2 :hosted :ESWC       (tp2)
}
```

$\mathbf{tp_2}$:
$\{\mathbf{v_2} =: \mathbf{Ljubljana}\}$,
$\{\mathbf{v_2} =: \mathbf{Athens}\}$

QUERY PROCESSING - BULK






```
SELECT DISTINCT * WHERE {
  ?v1 :hasCapital ?v2 .   (tp1)
  ?v2 :hosted :ESWC       (tp2)
}
```

$\mathbf{tp_2}$:

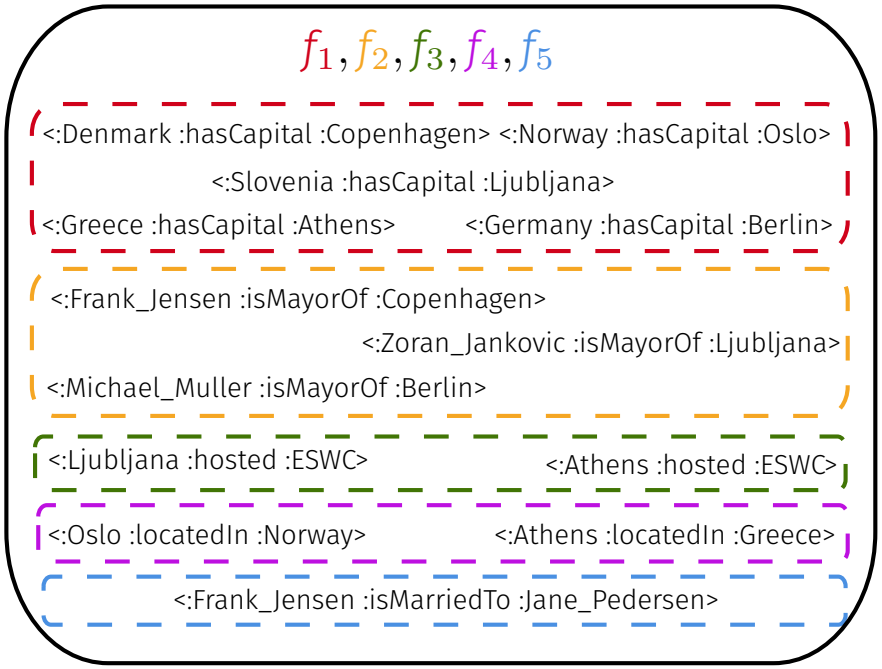
$\{v_2 =: Ljubljana\}, \{v_2 =: Athens\}$

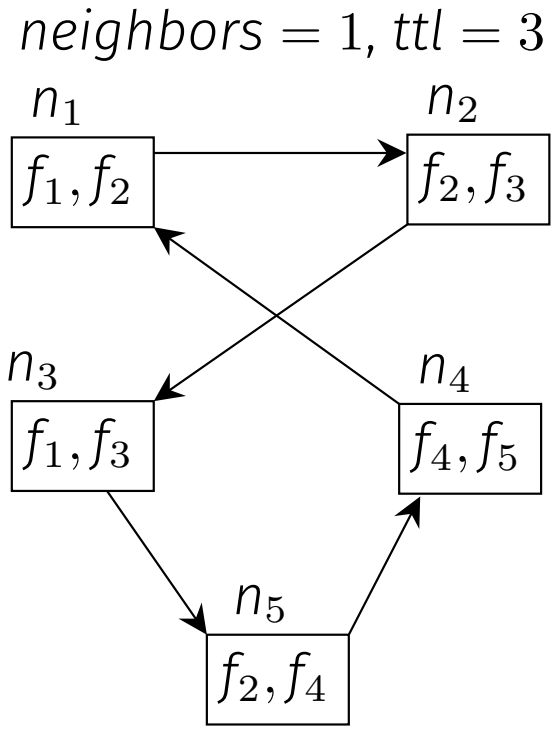
$\mathbf{tp_2} \bowtie \mathbf{tp_1}$:

$\{v_2 =: Ljubljana, v_1 =: Slovenia\}$

$\{v_2 =: Athens, v_1 =: Greece\}$

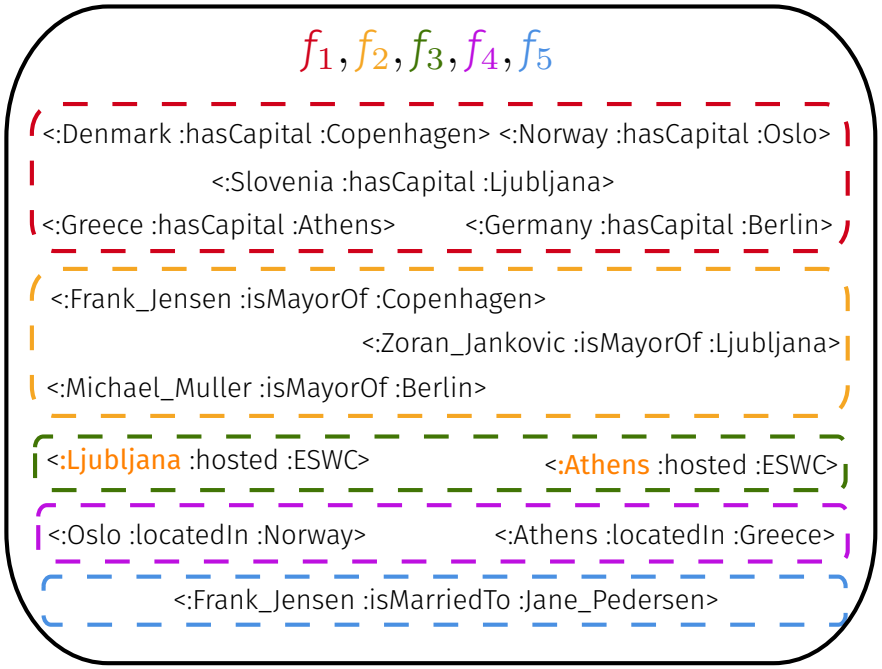
QUERY PROCESSING - FULL

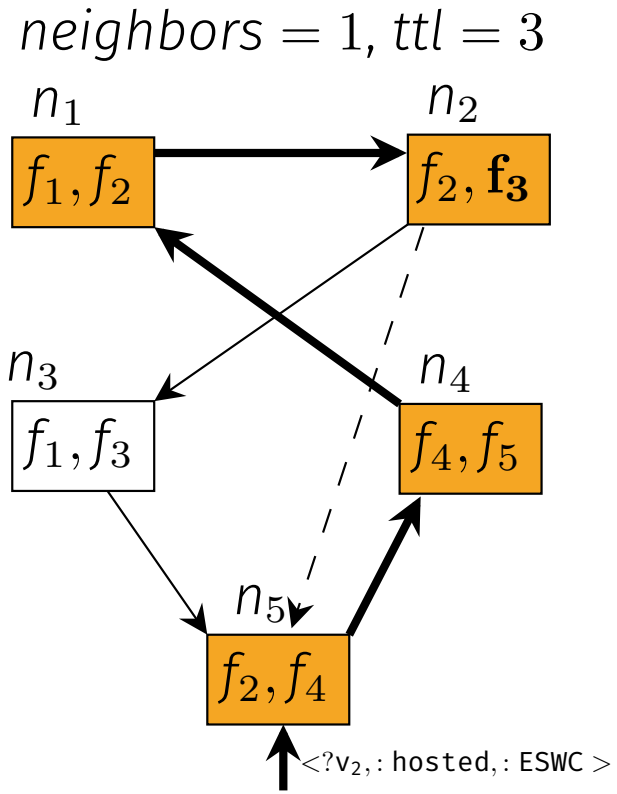





```
SELECT DISTINCT * WHERE {
  ?v1 :hasCapital ?v2 .   (tp1)
  ?v2 :hosted :ESWC       (tp2)
}
```

QUERY PROCESSING - FULL



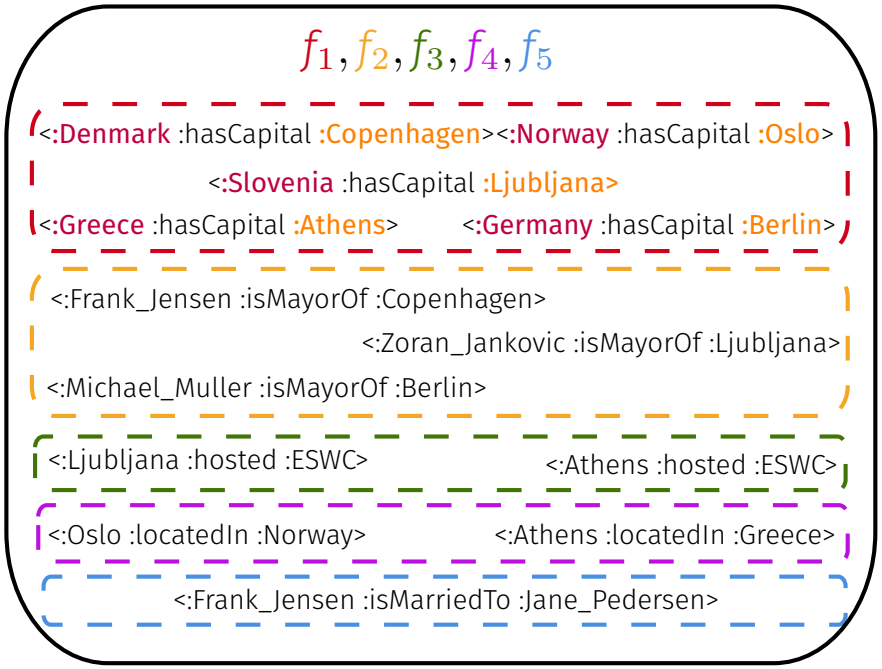


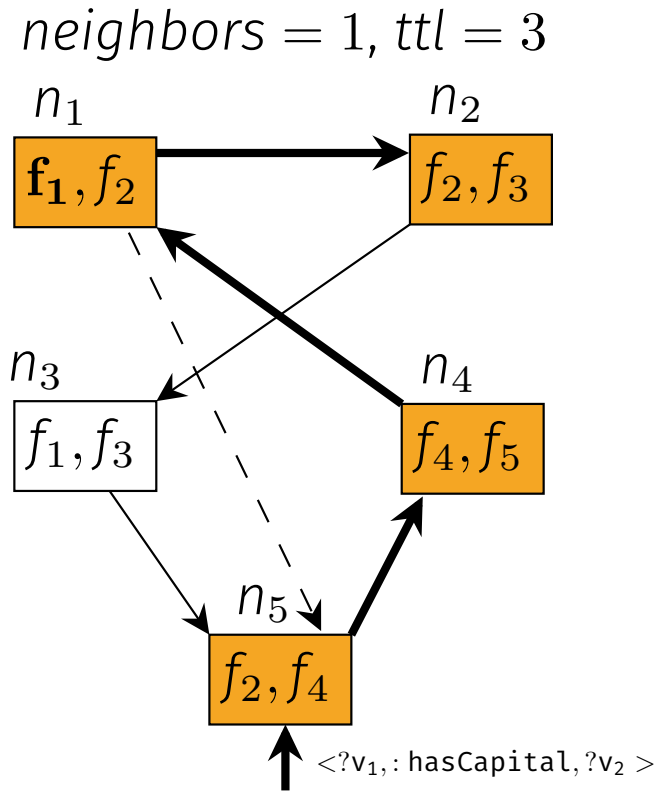

```
SELECT DISTINCT * WHERE {
  ?v1 :hasCapital ?v2 .  (tp1)
  ?v2 :hosted :ESWC      (tp2)
}
```

$\mathbf{tp_2}$:

$\{v_2 =: Ljubljana\}, \{v_2 =: Athens\}$

QUERY PROCESSING - FULL






```
SELECT DISTINCT * WHERE {
  ?v1 :hasCapital ?v2 .   (tp1)
  ?v2 :hosted :ESWC       (tp2)
}
```

$\mathbf{tp_2}$:

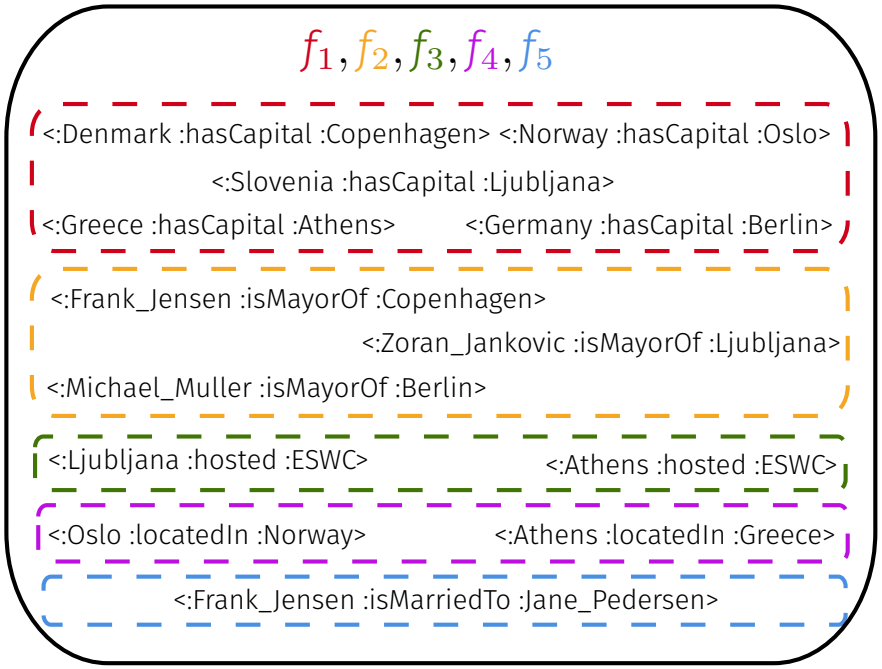
$\{v_2 =: Ljubljana\}, \{v_2 =: Athens\}$

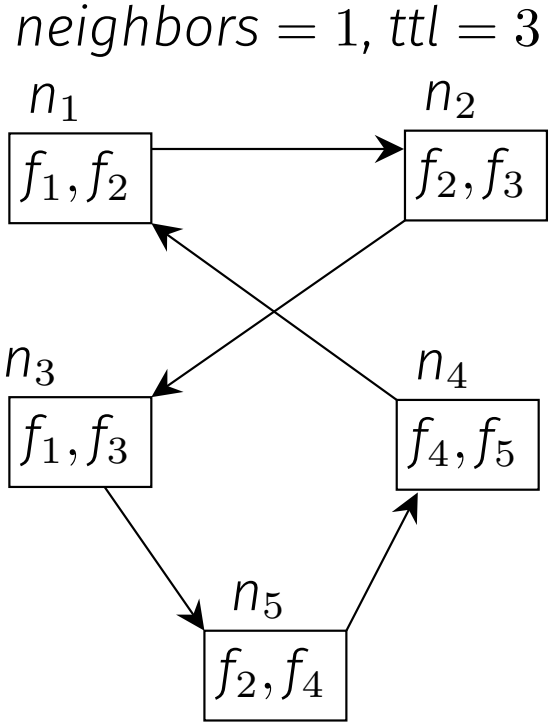
$\mathbf{tp_1}$:

$\{v_2 =: Ljubljana, v_1 =: Slovenia\}$

$\{v_2 =: Athens, v_1 =: Greece\}$

QUERY PROCESSING - FULL






```
SELECT DISTINCT * WHERE {
  ?v1 :hasCapital ?v2 .   (tp1)
  ?v2 :hosted :ESWC       (tp2)
}
```

$\mathbf{tp_2} \bowtie \mathbf{tp_1}$:

$\{v_2 =: Ljubljana, v_1 =: Slovenia\}$

$\{v_2 =: Athens, v_1 =: Greece\}$

EXPERIMENTAL SETUP

- 4xAMD Opteron 6376 16 cores, 2.3GHz each, 516GB RAM
  - *200 nodes* running concurrently

[11] A. Hasnain et al. **Extending LargeRDFBench for Multi-Source Data at Scale for SPARQL Endpoint Federation.** SSWS 2018

EXPERIMENTAL SETUP

- 4xAMD Opteron 6376 16 cores, 2.3GHz each, 516GB RAM
  - *200 nodes* running concurrently
- LargeRDFBench[11]: *Benchmark* suite for federated query engines

[11] A. Hasnain et al. **Extending LargeRDFBench for Multi-Source Data at Scale for SPARQL Endpoint Federation.** SSWS 2018

EXPERIMENTAL SETUP

- 4xAMD Opteron 6376 16 cores, 2.3GHz each, 516GB RAM
  - *200 nodes* running concurrently
- LargeRDFBench[11]: *Benchmark* suite for federated query engines
- 13 datasets with a total of over *1B triples*

[11] A. Hasnain et al. **Extending LargeRDFBench for Multi-Source Data at Scale for SPARQL Endpoint Federation.** SSWS 2018

EXPERIMENTAL SETUP

- 4xAMD Opteron 6376 16 cores, 2.3GHz each, 516GB RAM
  - *200 nodes* running concurrently
- LargeRDFBench[11]: *Benchmark* suite for federated query engines
- 13 datasets with a total of over *1B triples*
- *40* queries in 5 different categories
  - Cross Domain (*CD*): 7 queries
  - Life Sciences (*LS*): 7 queries
  - Complex (*C*): 10 queries
  - Larga-Data (*L*): 8 queries
  - Complex and High Amounts of Data Sources (*CH*): 8 queries

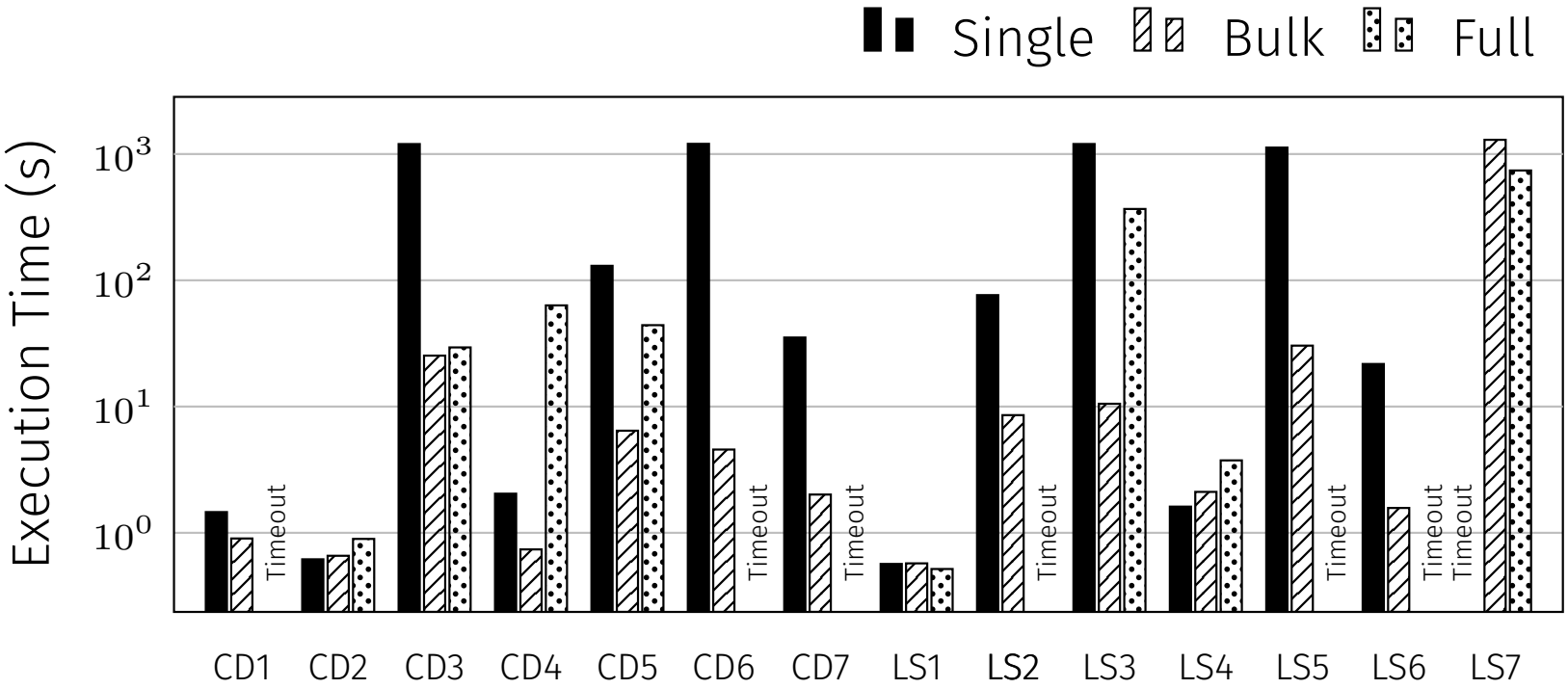
[11] A. Hasnain et al. **Extending LargeRDFBench for Multi-Source Data at Scale for SPARQL Endpoint Federation.** SSWS 2018

EXPERIMENTAL SETUP

- 4xAMD Opteron 6376 16 cores, 2.3GHz each, 516GB RAM
  - *200 nodes* running concurrently
- LargeRDFBench[11]: *Benchmark* suite for federated query engines
- 13 datasets with a total of over *1B triples*
- *40* queries in 5 different categories
  - Cross Domain (*CD*): 7 queries
  - Life Sciences (*LS*): 7 queries
  - Complex (*C*): 10 queries
  - Larga-Data (*L*): 8 queries
  - Complex and High Amounts of Data Sources (*CH*): 8 queries
- $ttl = 5$, $replication = 5\%$, $neighbors = 5$

[11] A. Hasnain et al. **Extending LargeRDFBench for Multi-Source Data at Scale for SPARQL Endpoint Federation.** SSWS 2018

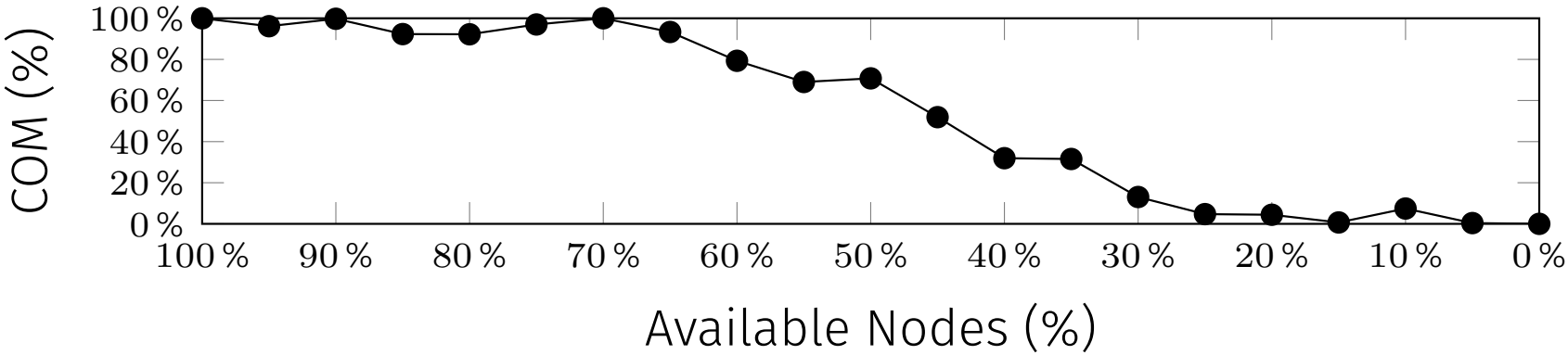
QUERY EXECUTION TIME



Bulk provides the best trade off between *performance*, *traffic* and *data transfer*

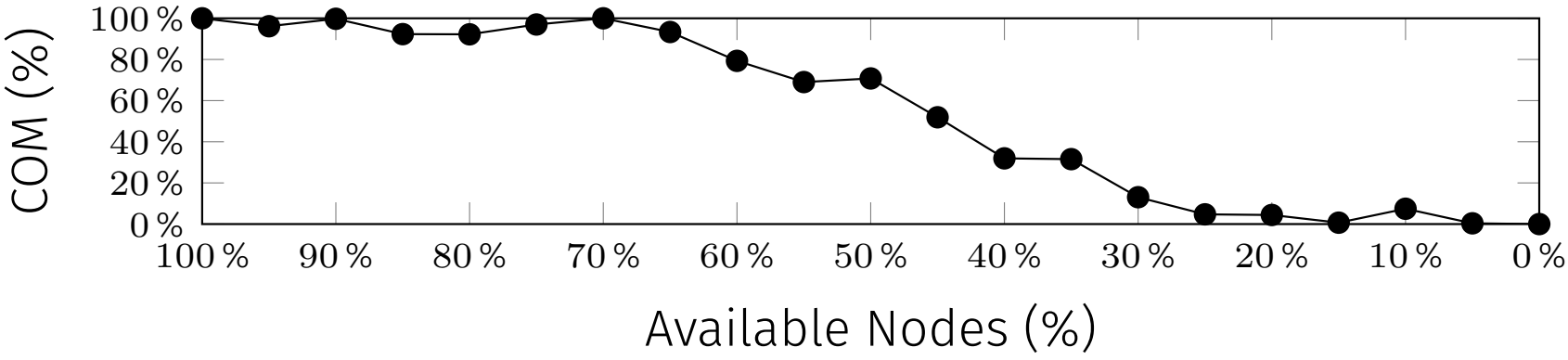
ROBUSTNESS

Without recovery time, completeness *decreases* with the number of available nodes

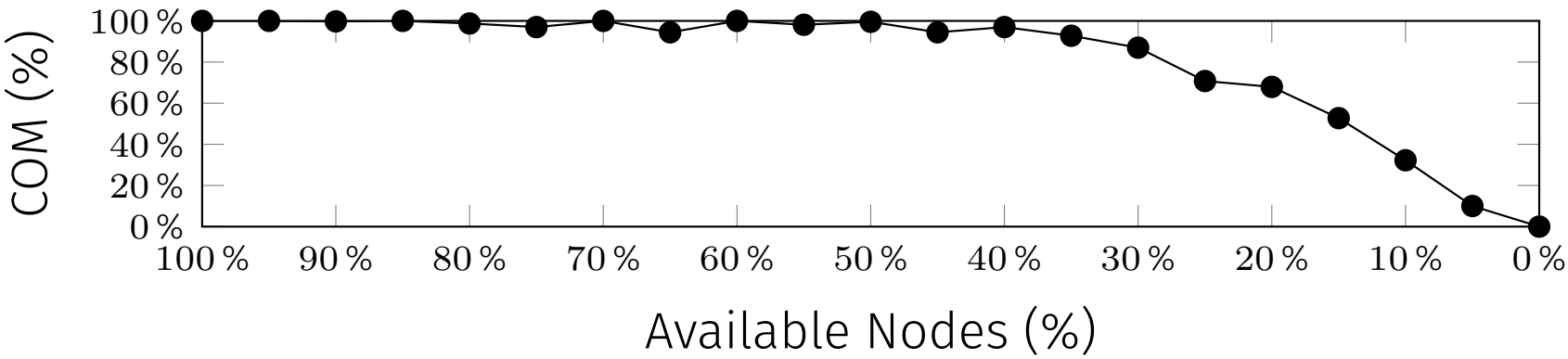


ROBUSTNESS

Without recovery time, completeness *decreases* with the number of available nodes

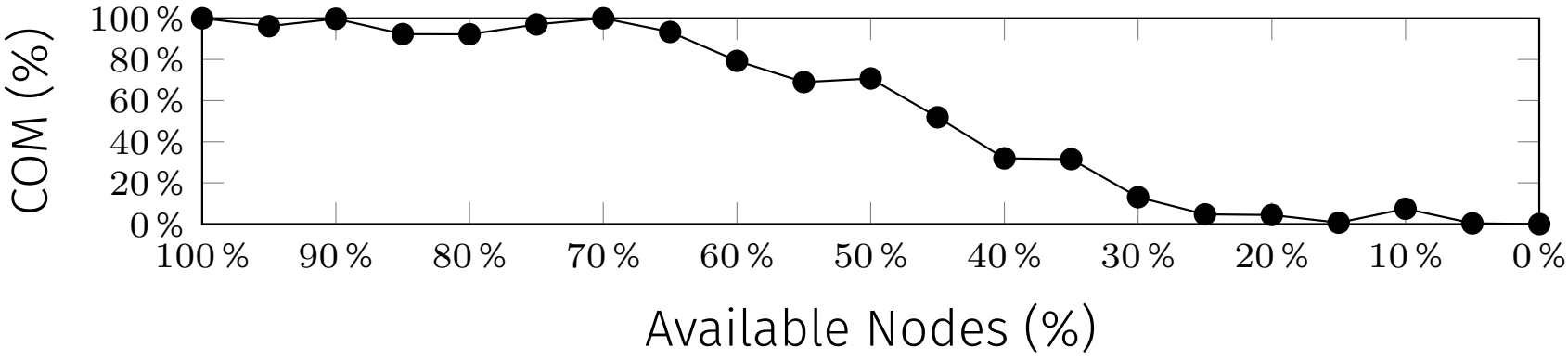


With recovery time, even when up to *50%* of nodes were removed, we achieved at least *94,44%* completeness

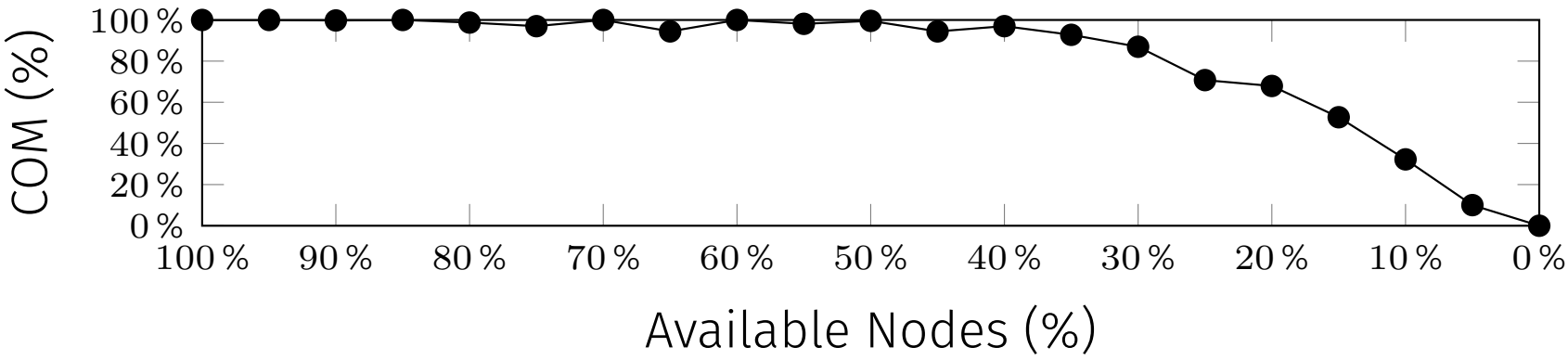


ROBUSTNESS

Without recovery time, completeness *decreases* with the number of available nodes



With recovery time, even when up to *50%* of nodes were removed, we achieved at least *94,44%* completeness



PIQNIC is able to *retain* a high completeness during node failure

CONCLUSIONS & FUTURE WORKS

Summary and Conclusion

- *PIQNIC*: a P2p clIent for Query processiNg over semantIC data

CONCLUSIONS & FUTURE WORKS

Summary and Conclusion

- *PIQNIC*: a P2p clIent for Query processiNg over semantIC data
- *Customizable* approach to data *fragmentation*

CONCLUSIONS & FUTURE WORKS

Summary and Conclusion

- *PIQNIC*: a P2p clIent for Query processiNg over semantIC data
- *Customizable* approach to data *fragmentation*
- Prototype implementation of three query processing approaches: *Single*, *Bulk* and *Full*

CONCLUSIONS & FUTURE WORKS

Summary and Conclusion

- *PIQNIC*: a P2p clIent for Query processiNg over semantIC data
- *Customizable* approach to data *fragmentation*
- Prototype implementation of three query processing approaches: *Single*, *Bulk* and *Full*
- *Bulk is better* overall and PIQNIC is able to *tolerate node failures*

CONCLUSIONS & FUTURE WORKS

Summary and Conclusion

- *PIQNIC*: a P2p clIent for Query processiNg over semantIC data
- *Customizable* approach to data *fragmentation*
- Prototype implementation of three query processing approaches: *Single*, *Bulk* and *Full*
- *Bulk is better* overall and PIQNIC is able to *tolerate node failures*

Future Works

- Process queries with *large intermediate results*

CONCLUSIONS & FUTURE WORKS

Summary and Conclusion

- *PIQNIC*: a P2p clIent for Query processiNg over semantIC data
- *Customizable* approach to data *fragmentation*
- Prototype implementation of three query processing approaches: *Single*, *Bulk* and *Full*
- *Bulk is better* overall and PIQNIC is able to *tolerate node failures*

Future Works

- Process queries with *large intermediate results*
- *Assess the completeness* of query answers during processing

CONCLUSIONS & FUTURE WORKS

Summary and Conclusion

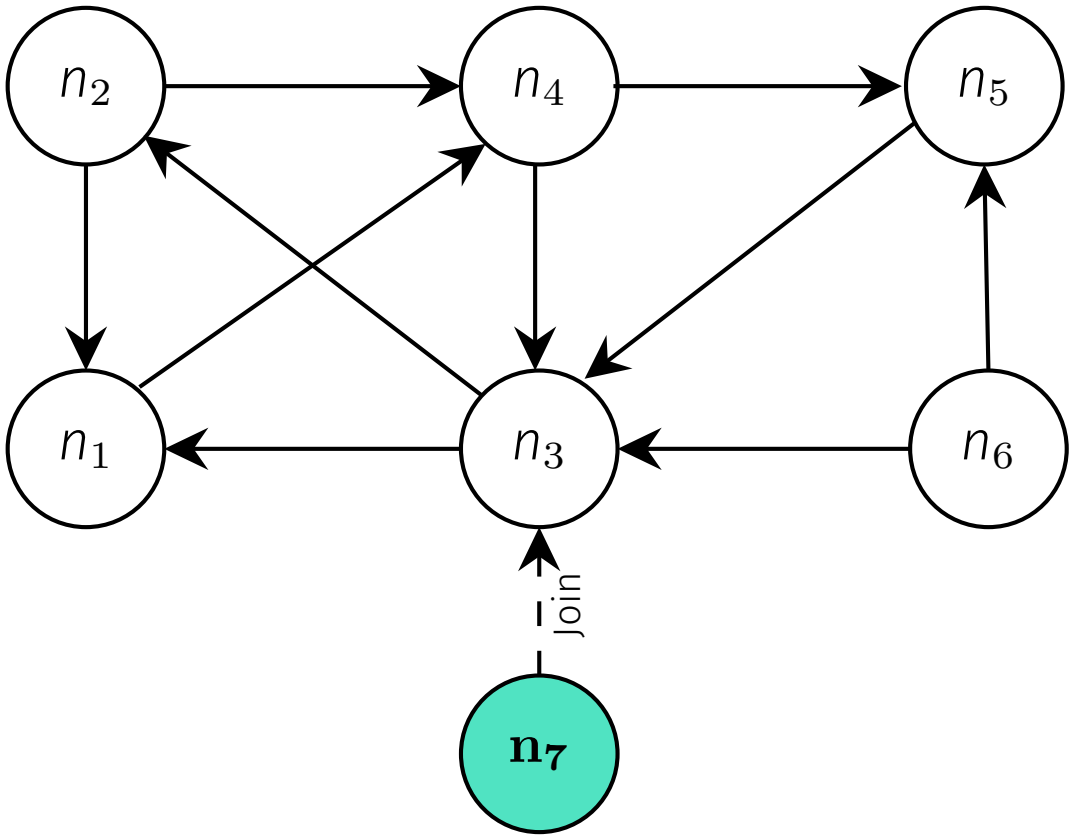
- *PIQNIC*: a P2p clIent for Query processiNg over semantIC data
- *Customizable* approach to data *fragmentation*
- Prototype implementation of three query processing approaches: *Single*, *Bulk* and *Full*
- *Bulk is better* overall and PIQNIC is able to *tolerate node failures*

Future Works

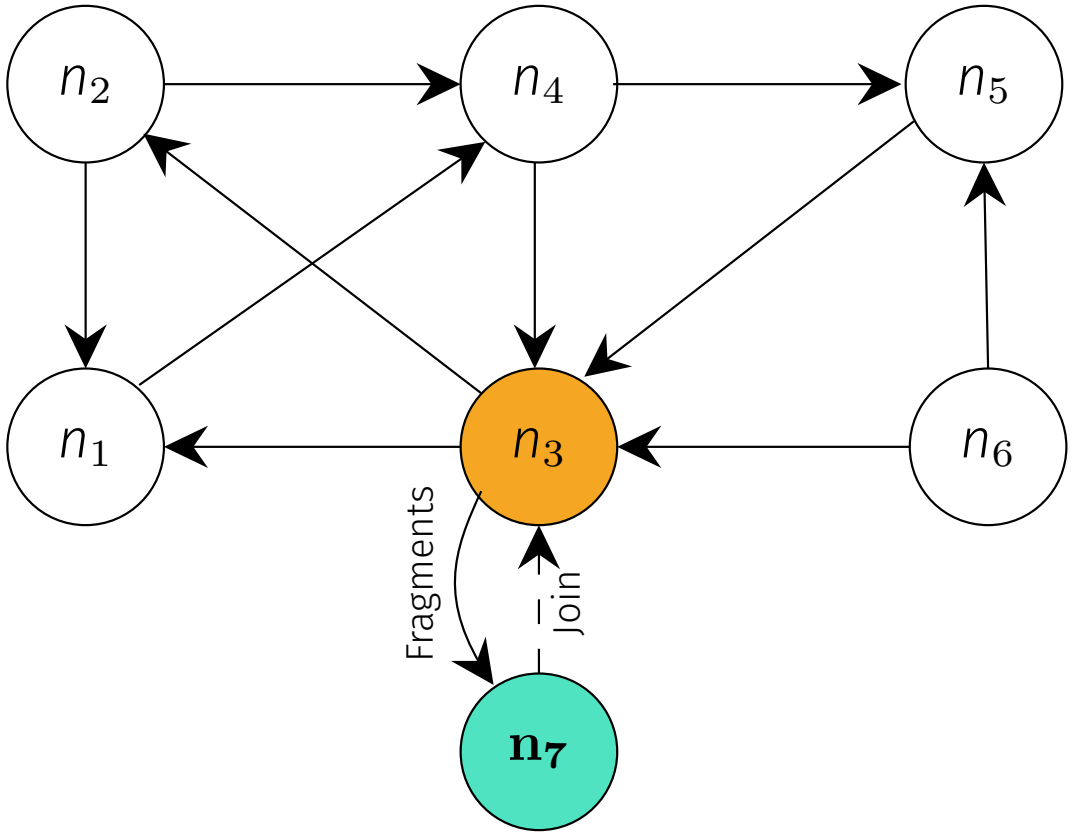
- Process queries with *large intermediate results*
- *Assess the completeness* of query answers during processing
- Alternative *fragmentation* and *relatedness* methods

Questions?

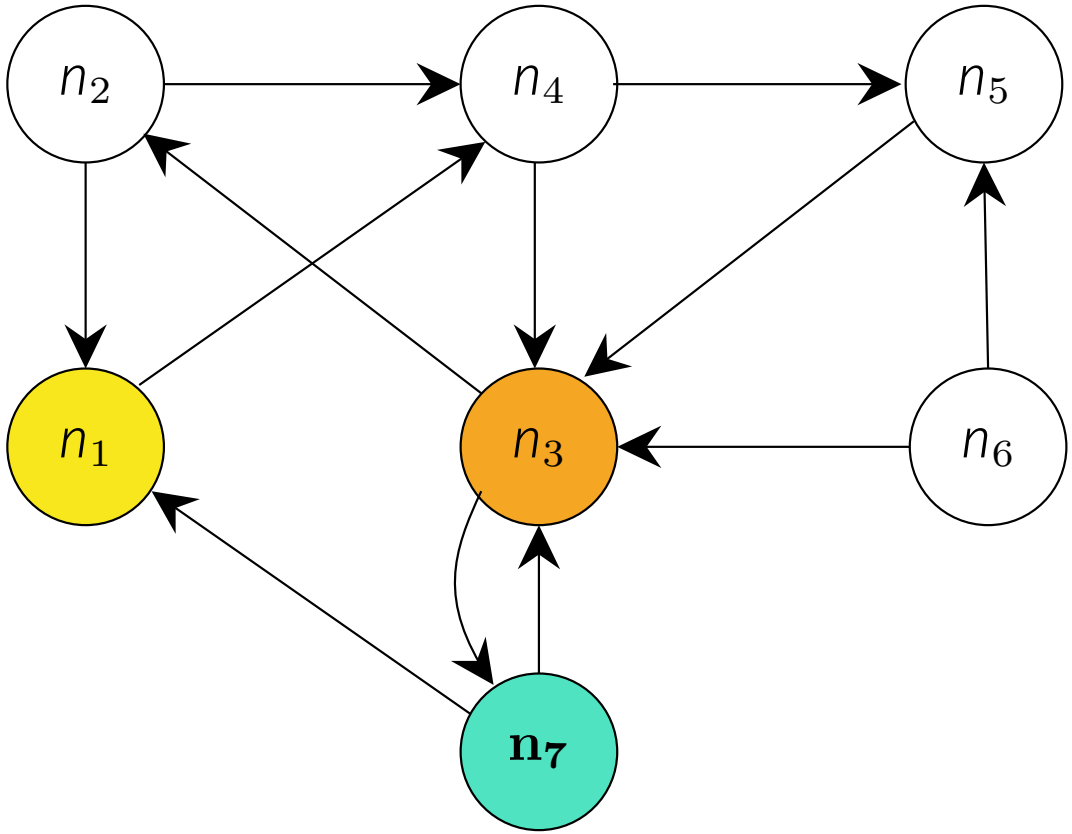
JOINING A NETWORK



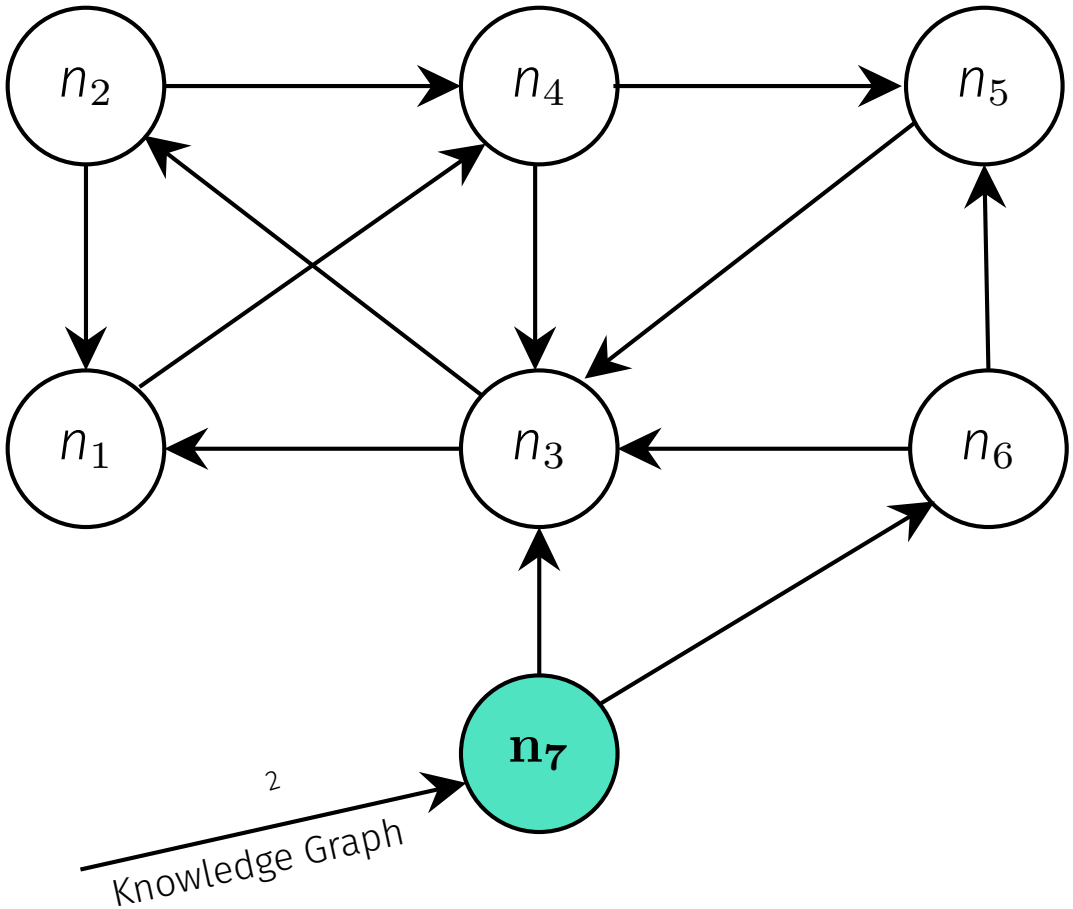
JOINING A NETWORK



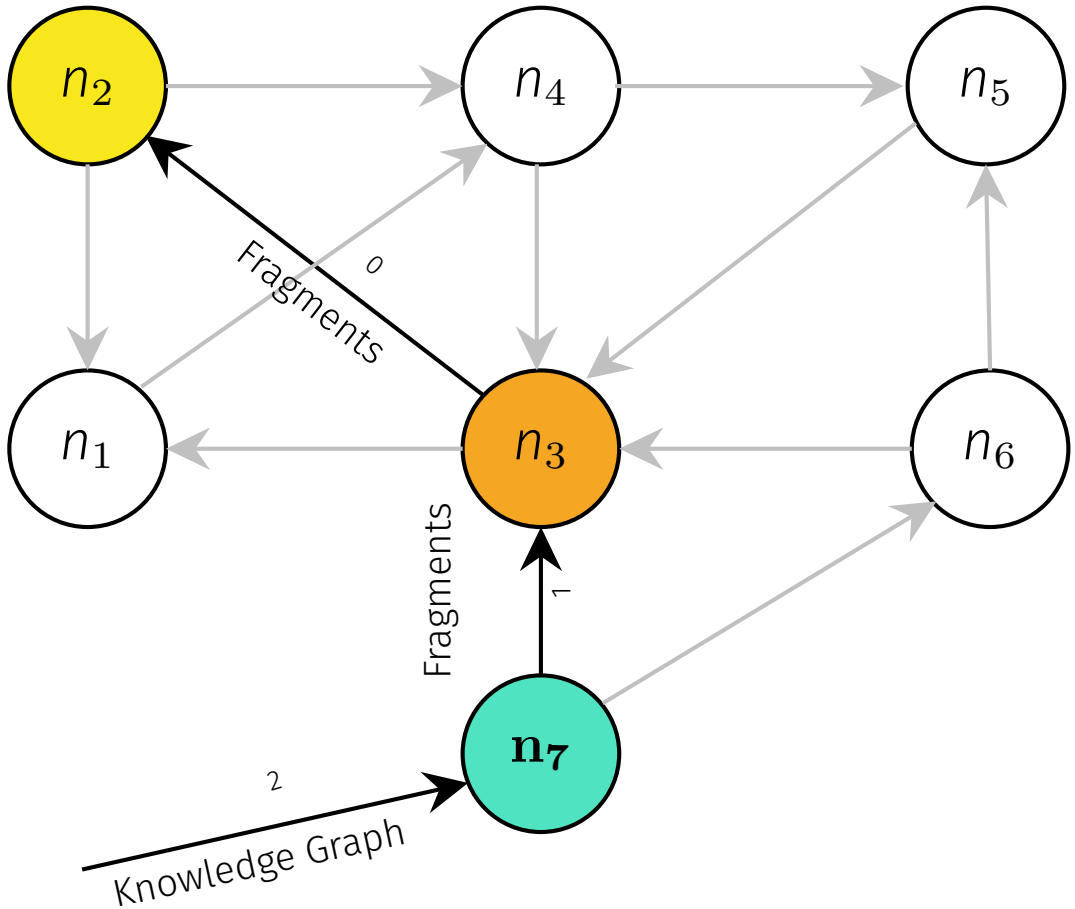
JOINING A NETWORK



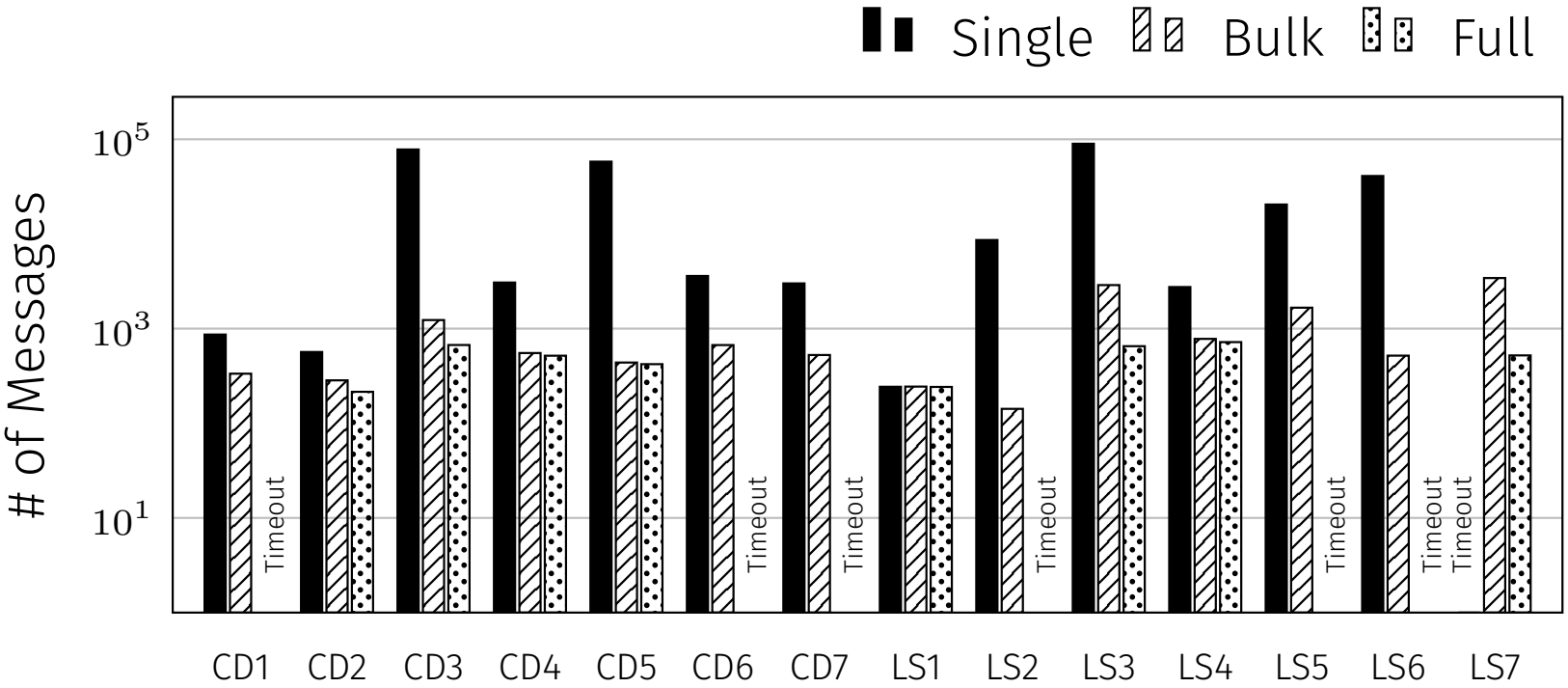
ADDING A DATASET



ADDING A DATASET

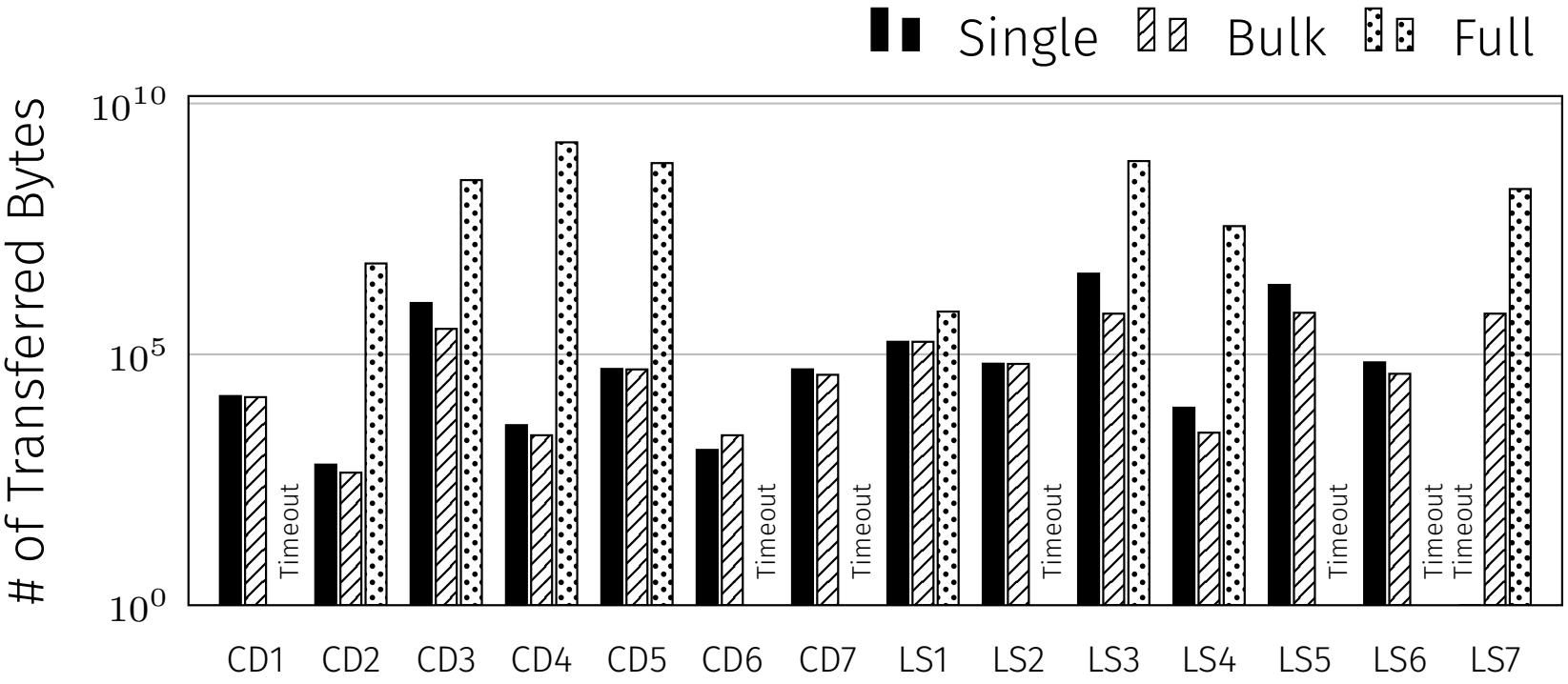


NUMBER OF MESSAGES



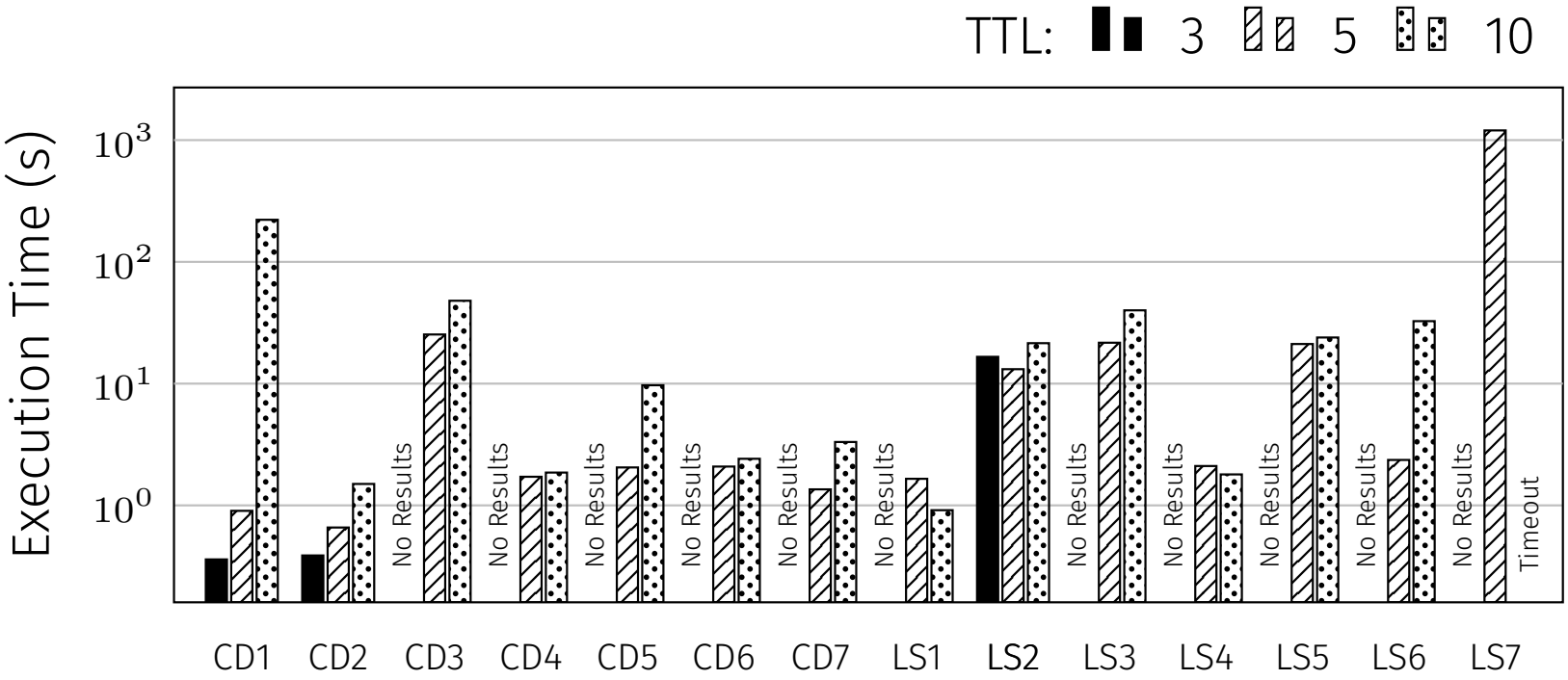
Full sends the fewest messages, *Single* sends significantly more

NUMBER OF TRANSFERRED BYTES



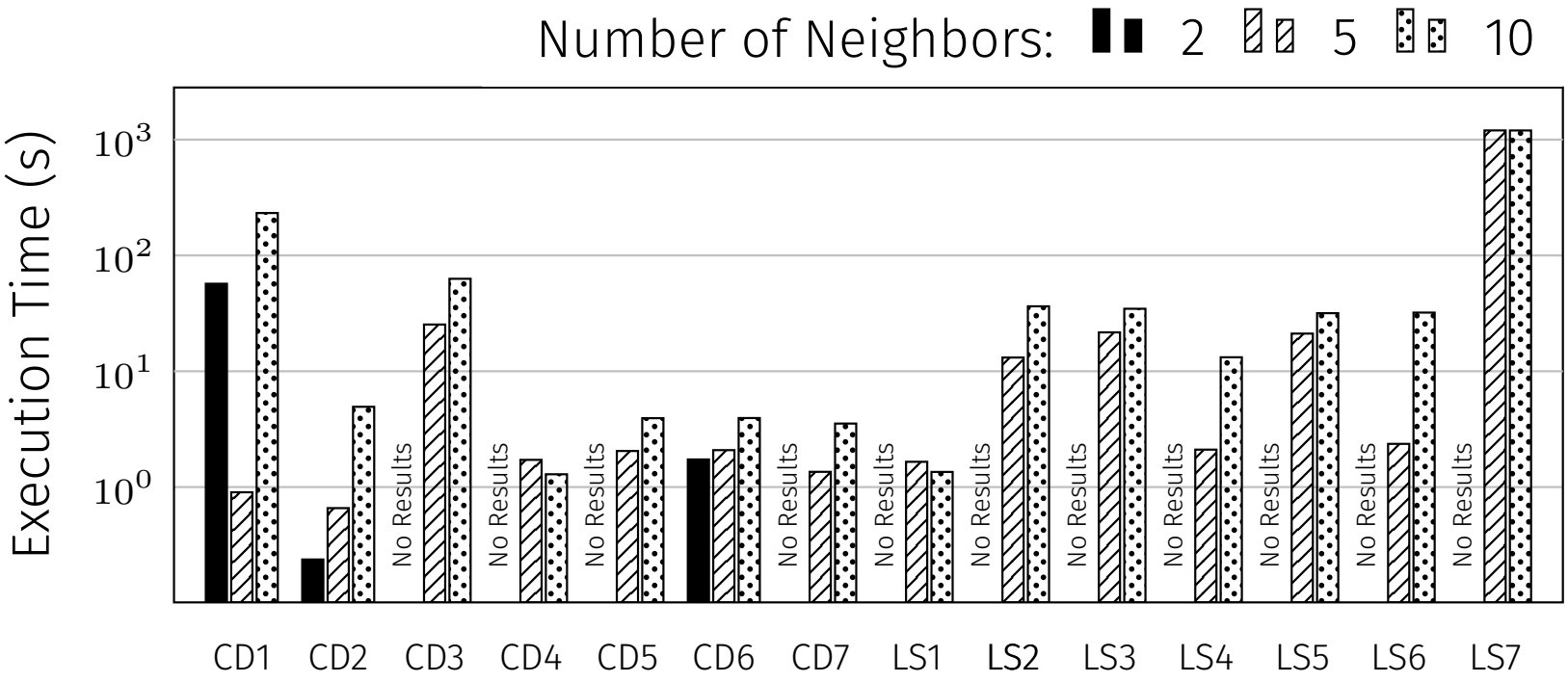
Bulk transfers the least amount of data, *Full* transfers significantly more

VARYING TIME-TO-LIVE VALUE



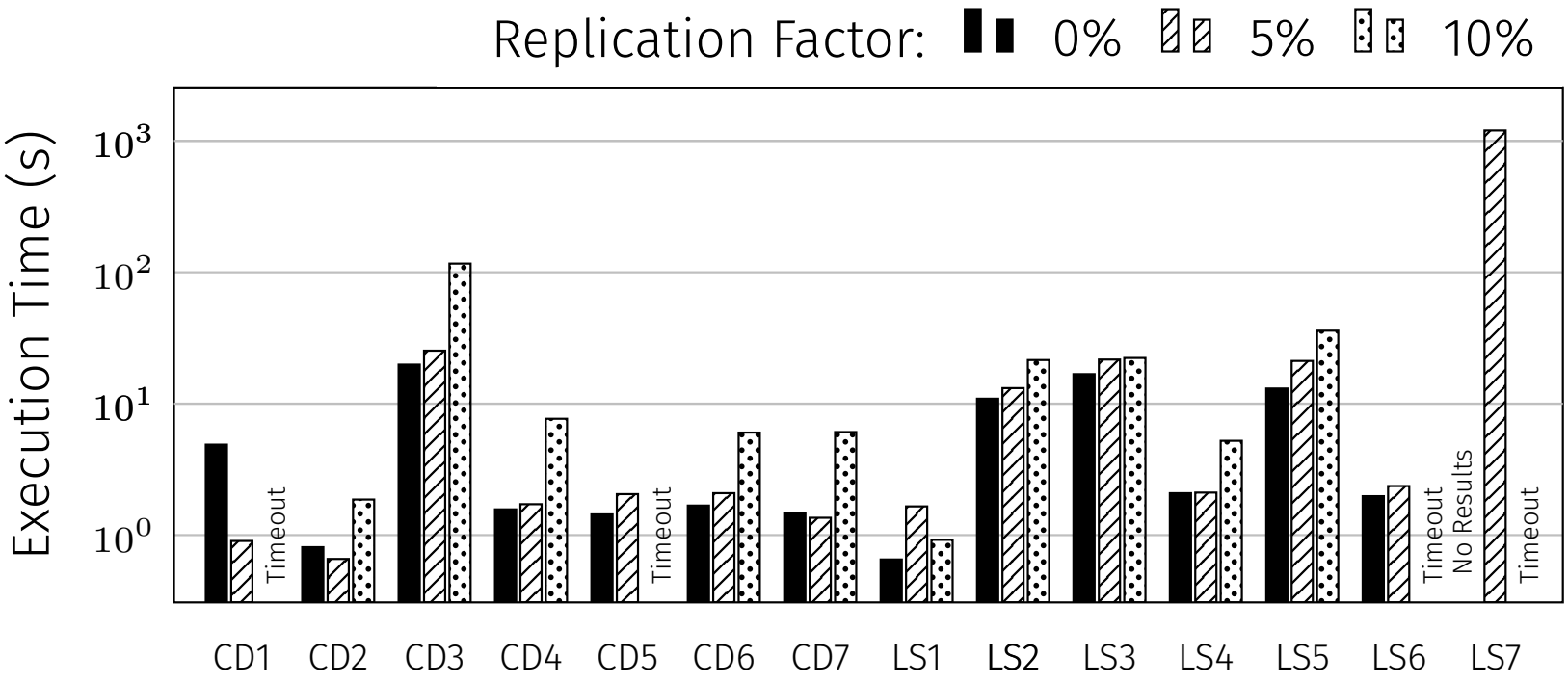
ttl=5 provides the best tradeoff between *performance* and *completeness*

VARYING NUMBER OF NEIGHBORS



nn=5 provides the best tradeoff between *performance* and *completeness*

VARYING REPLICATION FACTOR



rep=5% provides the best tradeoff between *performance* and *completeness*