Live Linked Data

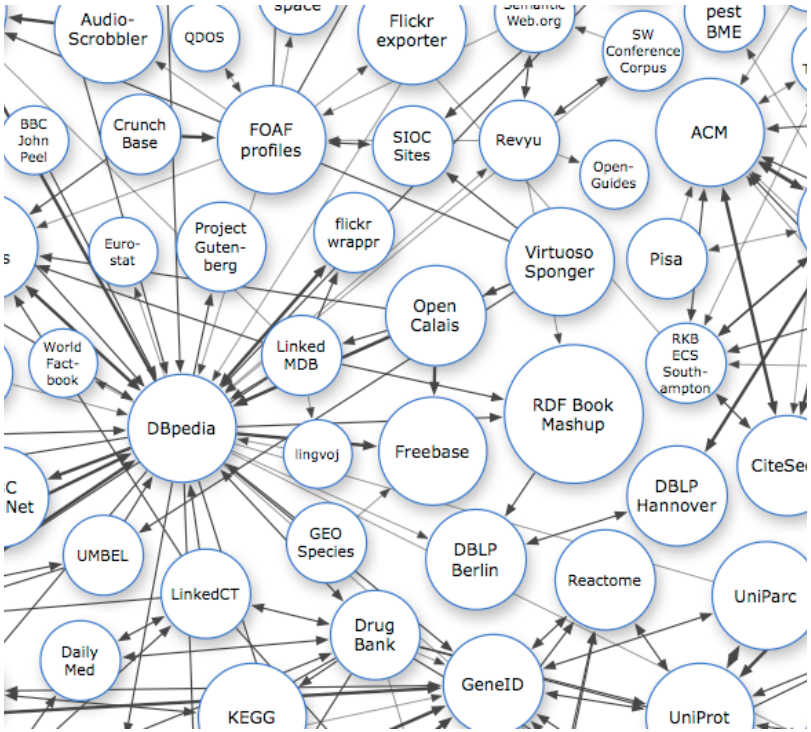
or how to make linked data writable with massive optimistic replication and Conflict-Free Replicated Data Types

Luis-Daniel Ibáñez, Nantes University, GDD team

Luis-Daniel Ibáñez, Hala Skaf-Molli, Pascal Molli, and Olivier Corby. 2012. Synchronizing semantic stores with commutative replicated data types. In Proceedings of the 21st international conference companion on World Wide Web (WWW '12 Companion). ACM, New York, NY, USA, 1091-1096. DOI=10.1145/2187980.2188246 http://doi.acm.org/10.1145/2187980.2188246

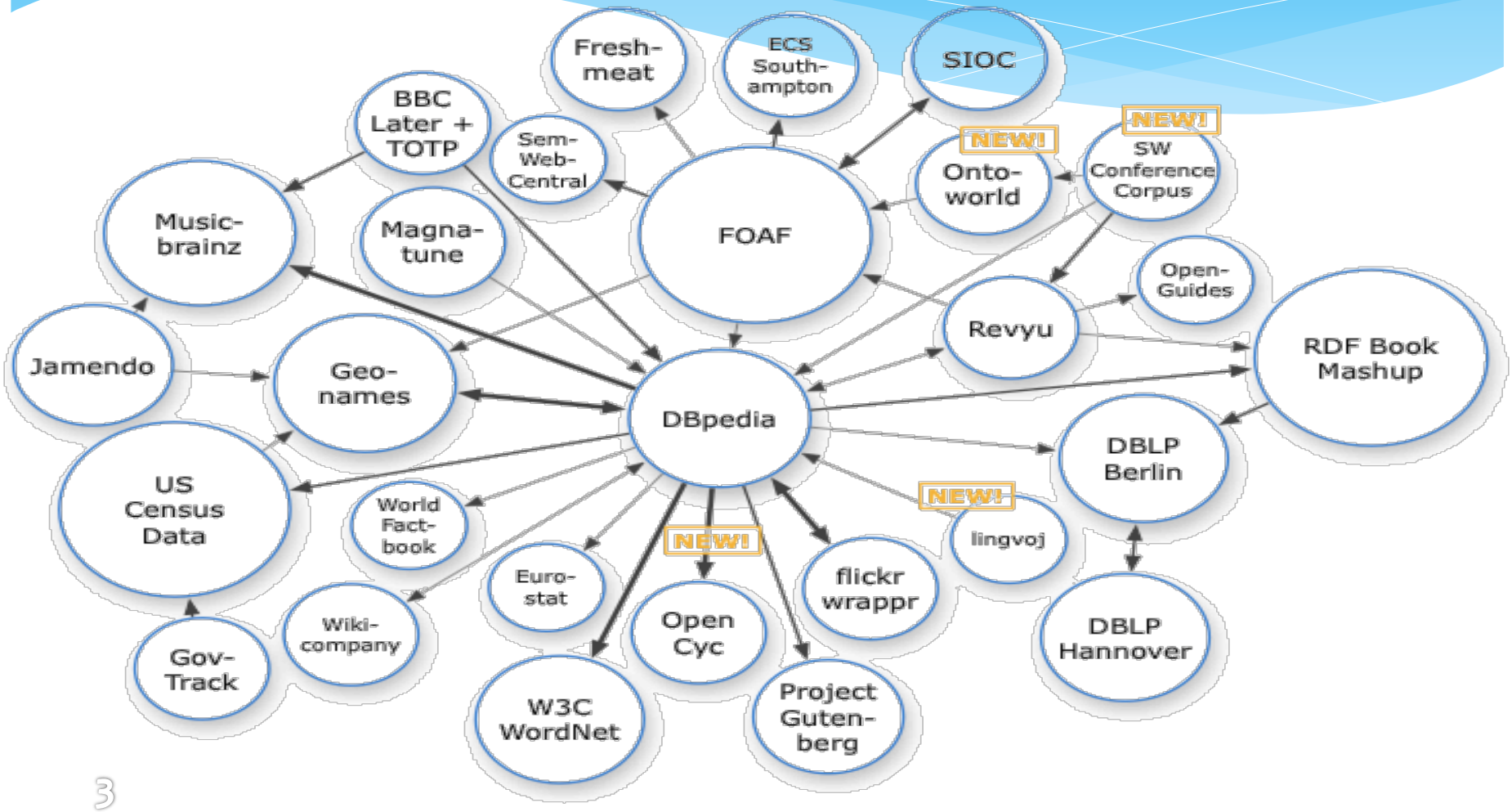
Linked Data

* **Autonomous participants** agree on a set of principles for publishing RDF data, allowing its querying and browsing across distributed servers.
* In 2011, More than 30 billion RDF triples distributed between ~700 **autonomous participants[1].**

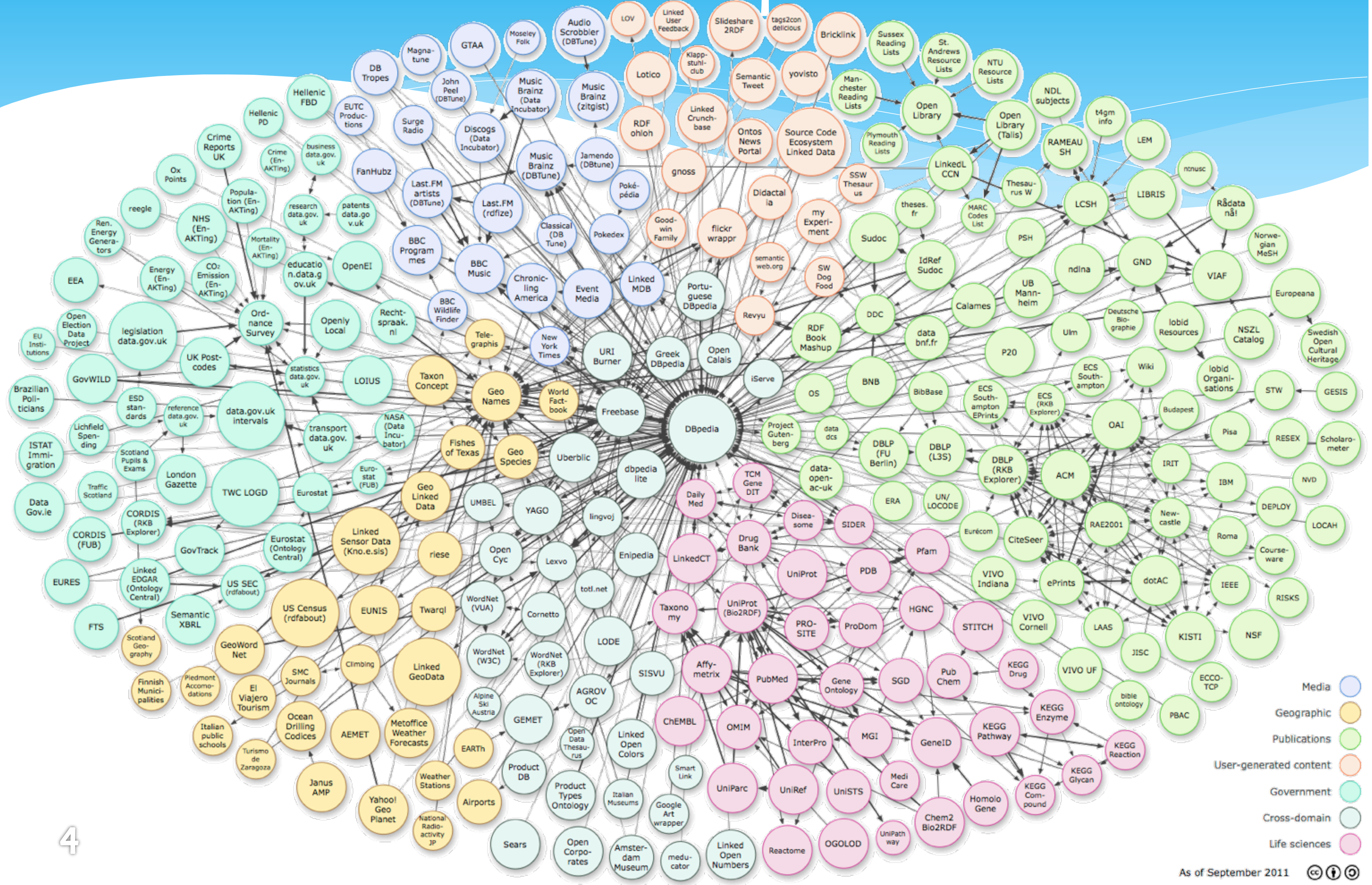


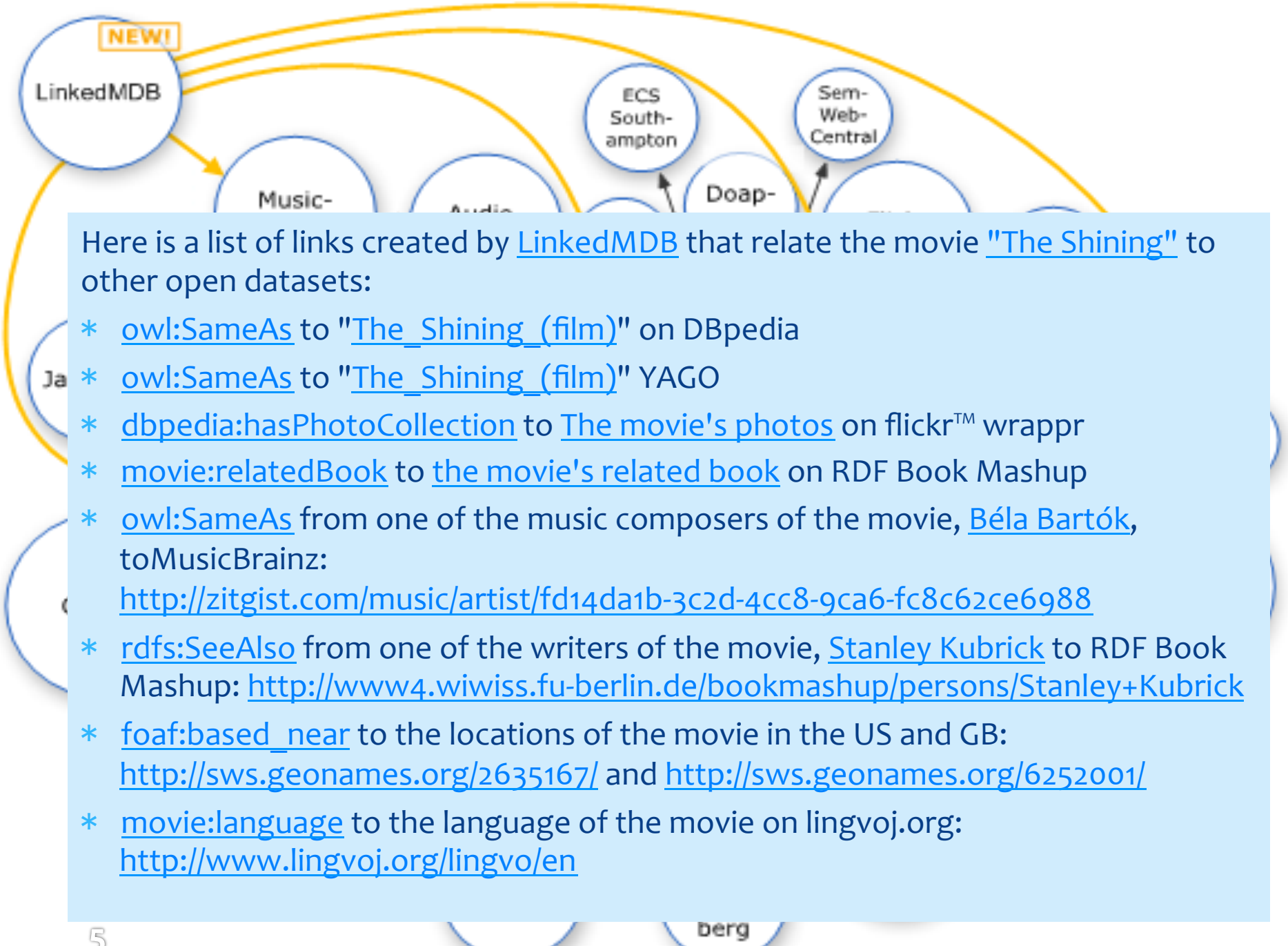
[1]Billion Triple Challenge + Linked Open Data Metadata
T. Käfer et. Al., Towards a Dynamic Linked Data Observatory, LDOW@WWW12

LOD - 2007 October



LOD - 2011 September





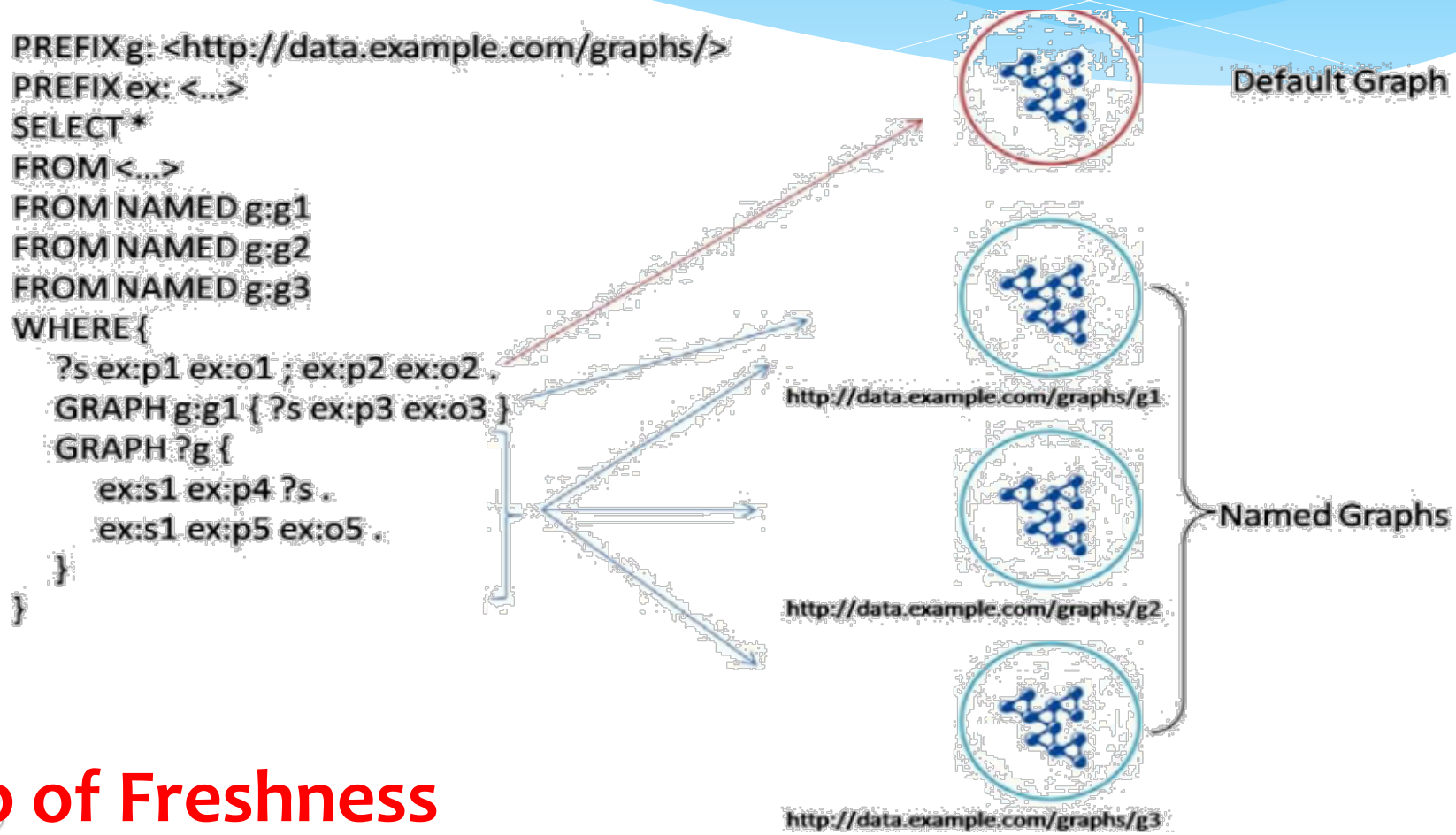
Here is a list of links created by LinkedMDB that relate the movie "The Shining" to other open datasets:

* owl:SameAs to "The_Shining_(film)" on DBpedia
* owl:SameAs to "The_Shining_(film)" YAGO
* dbpedia:hasPhotoCollection to The movie's photos on flickr™ wrappr
* movie:relatedBook to the movie's related book on RDF Book Mashup
* owl:SameAs from one of the music composers of the movie, Béla Bartók, toMusicBrainz: http://zitgist.com/music/artist/fd14da1b-3c2d-4cc8-9ca6-fc8c62ce6988
* rdfs:SeeAlso from one of the writers of the movie, Stanley Kubrick to RDF Book Mashup: http://www4.wiwiss.fu-berlin.de/bookmashup/persons/Stanley+Kubrick
* foaf:based_near to the locations of the movie in the US and GB: http://sws.geonames.org/2635167/ and http://sws.geonames.org/6252001/
* movie:language to the language of the movie on lingvoj.org: http://www.lingvoj.org/lingvo/en

Source : http://wiki.linkedmdb.org/Main/Interlinking

Queries in Linked Data: Local Copy and Query

```
PREFIX g: <http://data.example.com/graphs/>
PREFIX ex: <...>
SELECT *
FROM <...>
FROM NAMED g:g1
FROM NAMED g:g2
FROM NAMED g:g3
WHERE {
    ?s ex:p1 ex:o1 ; ex:p2 ex:o2 .
    GRAPH g:g1 { ?s ex:p3 ex:o3 }
    GRAPH ?g {
        ex:s1 ex:p4 ?s .
        ex:s1 ex:p5 ex:o5 .
    }
}
```



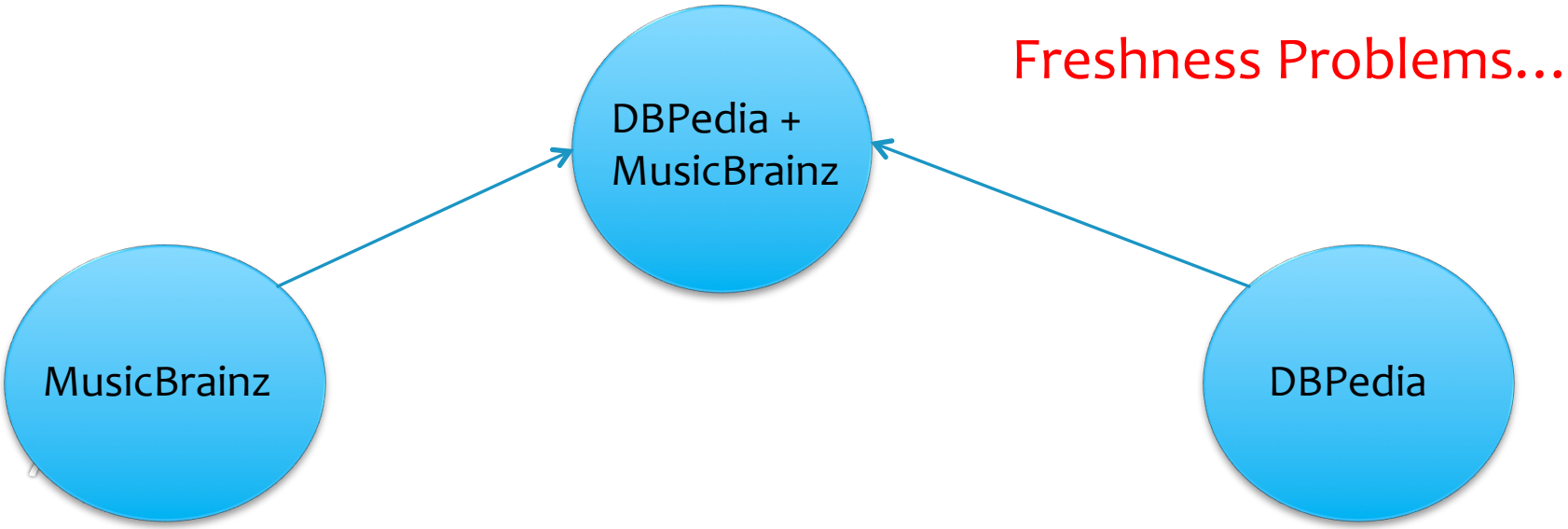
Pb of Freshness

http://www.cambridgesemantics.com/semantic-university/learn-sparql

Queries in Linked Data: Local Copy and Query

What is the largest city of Vicente Fernández' origin country?

Select ?country ?city where
{http://musicbrainz.org/Vicente_Fernandez http://musicbrainz.org/fromCountry ?country.
?country http://dbpedia.org/ontology/largestCity ?city . }



Queries in Linked Data: Distributed Queries

Find the birth dates of all of the actors in Star Trek: The Motion Picture

```
PREFIX movie: <http://data.linkedmdb.org/resource/movie/>
PREFIX dbpedia: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?actor_name ?birth_date
FROM <http://www.w3.org/People/Berners-Lee/card> # placeholder graph
WHERE {
   SERVICE <http://data.linkedmdb.org/sparql> {
     <http://data.linkedmdb.org/resource/film/675> movie:actor ?actor .
     ?actor movie:actor_name ?actor_name
   }
   SERVICE <http://dbpedia.org/sparql> {
     ?actor2 a dbpedia:Actor ; foaf:name ?actor_name_en ; dbpedia:birthDate ?birth_date .
     FILTER(STR(?actor_name_en) = ?actor_name)
   }
}
```

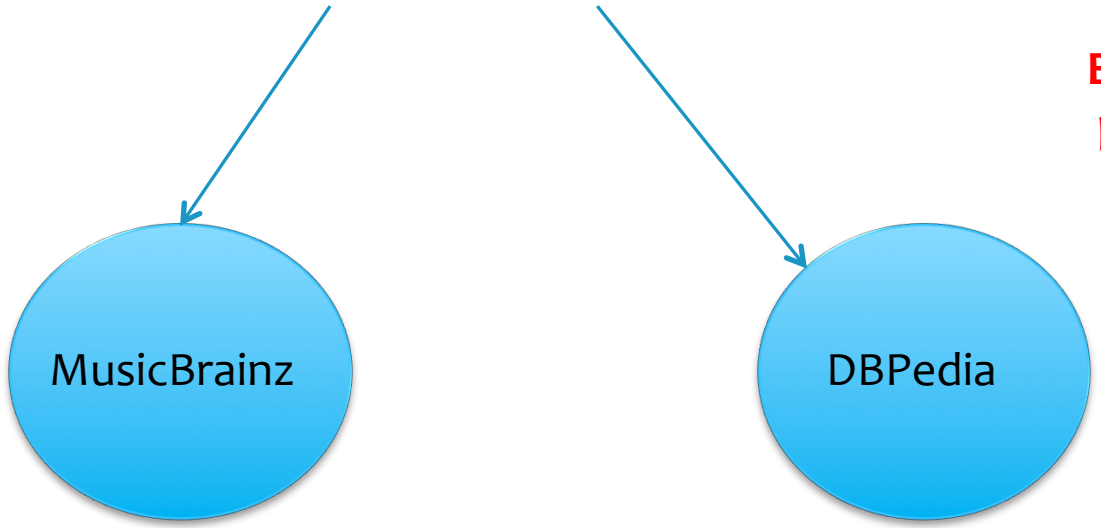
Pb of endpoint reliability and performances (and discovery?)

http://www.cambridgesemantics.com/semantic-university/learn-sparql

Queries in Linked Data: Distributed Queries

What is the largest city of Vicente Fernández' origin country?

```
SELECT ?country ?city WHERE {
SERVICE <http://musicbrainz.org/> {Vicente_Fernandez fromCountry ?country. }
SERVICE http://dbpedia.org/ {?country largestCity ?city . } }
```

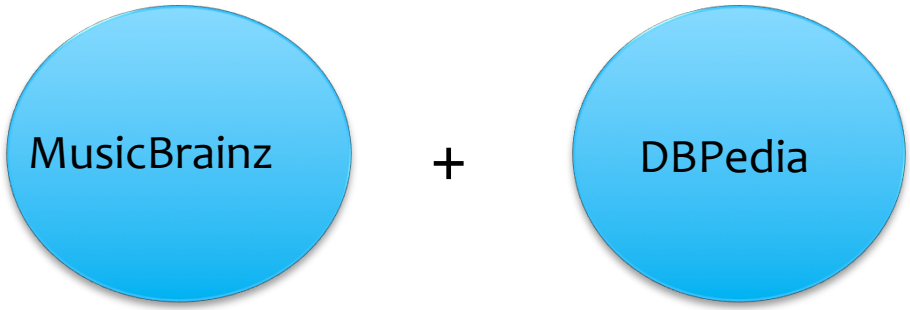


Endpoint reliability and performance Problems

Querying LOD: Sometimes OK

What is the largest city of Vicente Fernández' origin country?

Select ?country ?city where
{http://musicbrainz.org/Vicente_Fernandez http://musicbrainz.org/fromCountry ?country.
?country http://dbpedia.org/ontology/largestCity ?city . }

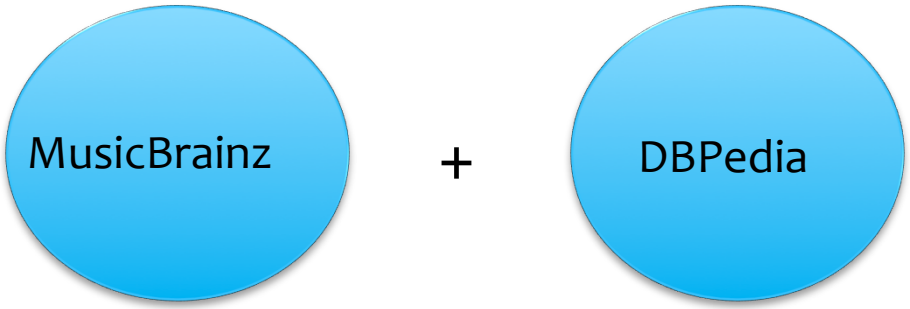


?country = Mexico
?city = Mexico City

Querying LOD: Sometimes KO

What is the largest city of Serge Gainsbourg's origin country?

Select ?country ?city where
{http://musicbrainz.org/Serge_Gainsbourg http://musicbrainz.org/fromCountry ?country.
?country http://dbpedia.org/ontology/largestCity ?city . }



?country = France
?city = Prefectures In France

?????

Issues

* Quality of data is poor, if I find an error, where to change and how can I change?
  * If accessing remote datasets: they are read-only (autonomous participants)
  * If modifying local copy: how to synchronize with original? How to publish changes?
* **How to make Linked Data Writable ? How to move from Linked Data 1.0 to Linked Data 2.0?**

Live Linked Data Approach

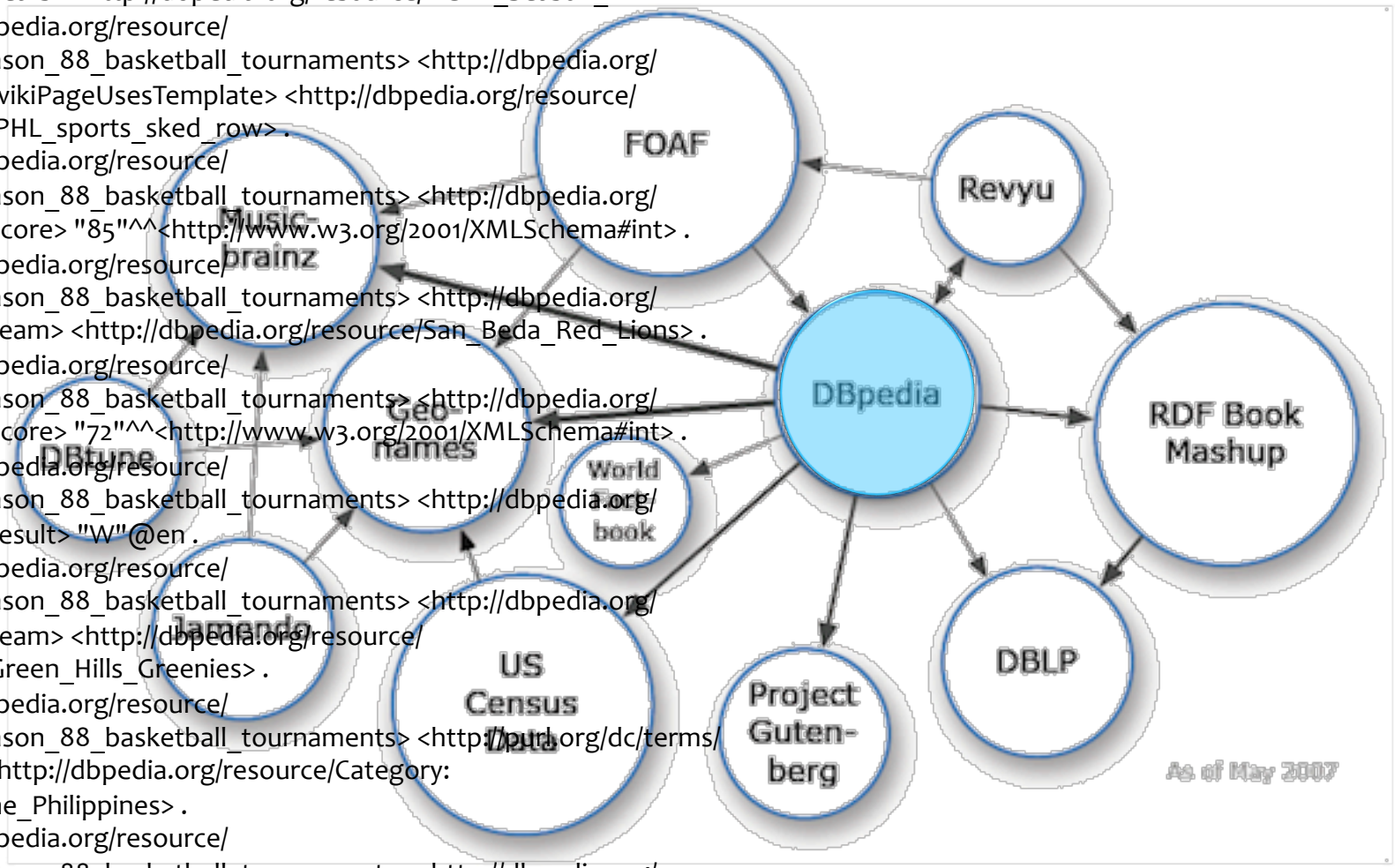
* Enable Massive Optimistic Replication in Linked Data:
  * SPARQL Update 1.1 allows to update RDF data locally.
  * Update feeds to provide streams of updates (e.g. DBPedia Live) -> The writing is "indirect", only if the other participant decides to consume I can change his dataset.
  * Conflict-Free Replicated Datatypes[1] (CRDT) to ensure data consistency when concurrent updates occur.
  * Consistency = System Correctness = Convergence + Intention preservation

[1] M. Shapiro, N. Preguiça, C. Baquero, M. Zawirski. Conflict-free Replicated Data Types. SSS 2011.

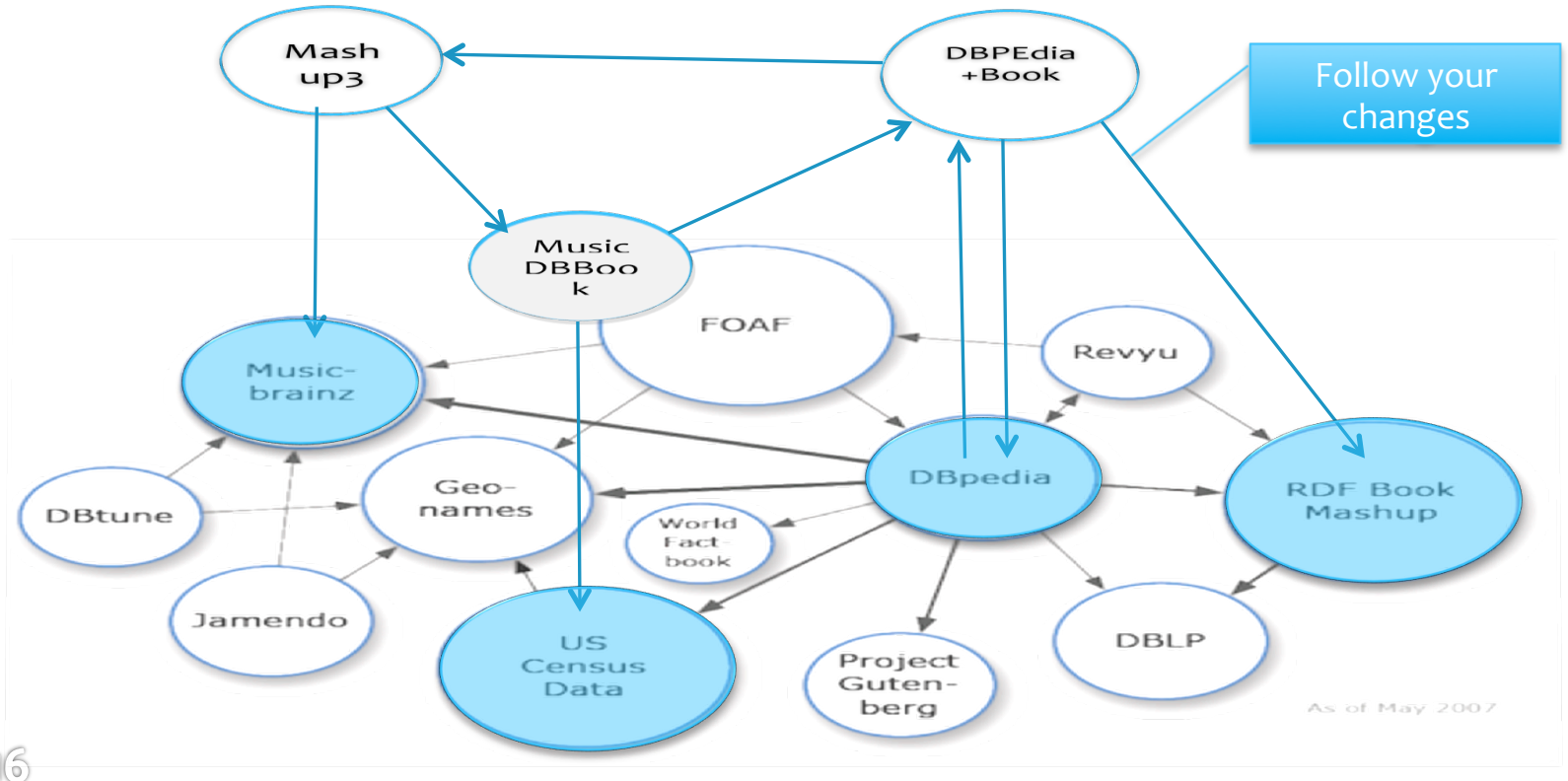
Related Works

* **Database: Multi-Master Replication and Eventual Consistency**
  * Duplicated Database (Johnson 75), base of Usenet, Multi-Master Database Replication, Eventual Consistency in NoSQL stores
  * P. Johnson and R. Thomas. Rfc677: The maintenance of duplicate databases, 1976.
* **Distributed System: Optimistic Replication and Eventual Consistency**
  * Yasushi Saito and Marc Shapiro. Optimistic replication. ACM Comput. Surv., 37(1):42–81, 2005.
  * Stéphane Weiss, Pascal Urso, and Pascal Molli. Logoot-undo: Distributed collaborative editing system on p2p networks. IEEE Transactions on Parallel Distributed Systems, 21(8):1162–1174, 2010
* **CSCW: Multi-Synchronous Collaboration Model and Eventual Consistency**
  * Chengzheng Sun, Xiaohua Jia, Yanchun Zhang, Yun Yang, and David Chen. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. ACM Transactions on Computer-Human Interaction, 5(1):63–108, 1998.
  * Gérald Oster, Pascal Urso, Pascal Molli, and Abdessamad Imine. 2006. Data consistency for P2P collaborative editing. In Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work (CSCW '06). ACM, New York, NY, USA, 259-268. DOI=10.1145/1180875.1180916 http://doi.acm.org/10.1145/1180875.1180916
* **Semantic Web: Synchronization of RDF stores**
  * Tim Berners-Lee and Dan Connolly. Delta: an ontology for the distribution of differences between rdf graphs. http://www.w3.org/DesignIssues/ Diff, 2004
  * Giovanni Tummarello, Christian Morbidoni, Reto Bachmann-Gmür, and Orri Erling. Rdfsync: Efficient remote synchronization of rdf models. In 6th International and 2nd Asian Semantic Web Conference (ISWC + ASWC), pages 537–551, 2007.
  * SD Share Group W3C, http://www.sdshare.org

<http://dbpedia.org/resource/
NCAA_Season_88_basketball_tournaments> <http://dbpedia.org/
property/years> <http://dbpedia.org/resource/NCAA_Season_88> .
<http://dbpedia.org/resource/
NCAA_Season_88_basketball_tournaments> <http://dbpedia.org/
property/wikiPageUsesTemplate> <http://dbpedia.org/resource/
Template:PHL_sports_sked_row> .
<http://dbpedia.org/resource/
NCAA_Season_88_basketball_tournaments> <http://dbpedia.org/
property/score> "85"^^<http://www.w3.org/2001/XMLSchema#int> .
<http://dbpedia.org/resource/
NCAA_Season_88_basketball_tournaments> <http://dbpedia.org/
property/team> <http://dbpedia.org/resource/San_Beda_Red_Lions> .
<http://dbpedia.org/resource/
NCAA_Season_88_basketball_tournaments> <http://dbpedia.org/
property/score> "72"^^<http://www.w3.org/2001/XMLSchema#int> .
<http://dbpedia.org/resource/
NCAA_Season_88_basketball_tournaments> <http://dbpedia.org/
property/result> "W"@en .
<http://dbpedia.org/resource/
NCAA_Season_88_basketball_tournaments> <http://dbpedia.org/
property/team> <http://dbpedia.org/resource/
La_Salle_Green_Hills_Greenies> .
<http://dbpedia.org/resource/
NCAA_Season_88_basketball_tournaments> <http://purl.org/dc/terms/
subject> <http://dbpedia.org/resource/Category:
2012_in_the_Philippines> .
<http://dbpedia.org/resource/
NCAA_Season_88_basketball_tournaments> <http://dbpedia.org/
resource/Template:PHL_sports_sked_row> "result16"@en .
<http://dbpedia.org/resource/



Live Linked Data Overview



Live Linked Data Overview

* A social network of linked data participants based on a « follow your change » relation.
* Makes Linked Data a « read/write » space **: from linked data 1.0 to linked data 2.0**
* Creates assemblies of datasets and enable « **synchronize and search** » paradigm: between warehousing approach and distributed queries approach.

Enabling LLD with CRDTs

* Construct a CRDT for RDF Graph Stores with SPARQL UPDATE 1.1 Operations with minimal overhead under Live Linked Data conditions:
  * Unknown but steadily growing number of autonomous participants.
    * No central controls (coordination, primary replicas,small core of datacenters)
    * Followers pull from followed.
  * Dynamicity parameters:
    * Degree of change / Change frequency → varies between producers.
    * Growth rate → Positive, knowledge grows.

RDF Datasets

Definition 1.1 (RDF Terms, Triples, and Variables) *Assume there are pairwise disjoint infinite sets* I, B, *and* L *(IRIs, Blank nodes, and literals). A tuple* $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$ *is called an* RDF triple. *In this tuple,* s *is the* subject, p *the* predicate *and* o *the* object. *We denote the union* $I \cup B \cup L$ *by* T *(RDF terms). Assume additionally the existence of an infinite set* V *of variables disjoint from the above sets.*

Definition 1.2 (RDF Graph) *An* RDF graph *is a set of RDF triples. If* G *is an RDF graph,* $\text{term}(G)$ *is the set of elements of* T *appearing in the triples of* G, *and* $\text{blank}(G)$ *is the set of blank nodes appearing in* G, *i.e.* $\text{blank}(G) = \text{term}(G) \cap B$.

Definition 1.3 (RDF Dataset) *An* RDF dataset *[2] is a set*

$$\mathcal{D} = \{G_0, \langle u_1, G_1 \rangle, \ldots, \langle u_n, G_n \rangle\}$$

http://users.dcc.uchile.cl/~cgutierr/ftp/sparql_semantics.pdf

RDF Datasets

* Assume there are pairwise disjoint infinite sets I , B, and L (IRIs, Blank nodes, and Literals, respectively). A triple $(s,p,o) \in (I \cup B) \times I \times (I \cup B \cup L)$ is called an **RDF triple**. In this tuple, *s* is the subject, *p* the predicate, and *o* the object[1].
* An **RDF Graph** is a set of RDF triples[1].
* An **RDF dataset** is a set: $\{ G, (<u_1>, G_1), (<u_2>, G_2), \ldots (<u_n>, G_n) \}$ where G and each G_i are graphs, and each $<u_i>$ is an IRI. Each $<u_i>$ is distinct. G, is called the "default graph" and the pairs $(<u_i>, G_i)$ are called "named graphs"[2].

[1] J. Pérez, M. Arenas and C. Gutiérrez, Semantics and Complexity of SPARQL, ACM Transactions on DB Systems, 2009

[2] http://www.w3.org/TR/sparql11-query/#sparqlDataset

Datasets examples

```
# Default graph
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ns: <http://example.org/ns#> .
<http://example/book2> ns:price 42 .
<http://example/book2> dc:title "David Copperfield" .
<http://example/book2> dc:creator "Edmund Wells" .

 # with blank nodes
 ex:John   foaf:knows      _:p1
 _:p1     foaf:birthDate   04-21
```

SPARQL Update Queries

Syntax	Formal Operation
INSERT DATA *QuadData*	Dataset-UNION(GS, Dataset(*QuadData*,{},GS,GS))
DELETE DATA *QuadData*	Dataset-DIFF(GS, Dataset(*QuadData*,{},GS,GS))
DELETE $QuadPattern_{DEL}$ INSERT $QuadPattern_{INS}$ WHERE *GroupGraphPattern*	Dataset-UNION(Dataset-DIFF(GS, Dataset($QuadPattern_{DEL}$,*GroupGraphPattern*,DS,GS)), Dataset($QuadPattern_{INS}$, *GroupGraphPattern*,DS,GS)
DELETE $QuadPattern_{DEL}$ WHERE *GroupGraphPattern*	Dataset-UNION(Dataset-DIFF(GS, Dataset($QuadPattern_{DEL}$,*GroupGraphPattern*,DS,GS)), Dataset({}, *GroupGraphPattern*,DS,GS)
INSERT $QuadPattern_{INS}$ *UsingClause** WHERE *GroupGraphPattern*	Dataset-UNION(Dataset-DIFF(GS, Dataset({},*GroupGraphPattern*,DS,GS)), Dataset($QuadPattern_{INS}$, *GroupGraphPattern*,DS,GS)
LOAD (SILENT)? *IRIref*	Dataset-UNION(GS, { graph(*documentIRI*) })
CLEAR (SILENT)? DEFAULT	{{}} union {(iri_i, G_i) | 1 ≤ i ≤ n}
CREATE (SILENT)? *IRIref*	GS union {(iri, {})} if iri not in graphNames(GS); otherwise, OpCreate(GS, iri) = GS
DROP (SILENT)? IRIref	GS if iri not in graphNames(GS); otherwise, OpDrop(GS, iri_j) = {DG} union {(iri_i, G_i) | i ≠ j and 1 ≤ i ≤ n}

http://www.w3.org/TR/sparql11-update/#formalModel

Insert Ground Triples

PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
INSERT DATA
{ GRAPH <http://example/bookStore>
{ <http://example/book1> ns:price 42 .
<http://example/book1> dc:creator "A.N.Other" . } }

Data before:

Graph: http://example/bookStore
@prefix dc: <http://purl.org/dc/elements/1.1/> .
<http://example/book1> dc:title "Fundamentals of Compiler Design" .

Data after:

Graph: http://example/bookStore
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ns: <http://example.org/ns#> .
<http://example/book1> dc:title "Fundamentals of Compiler Design" .
<http://example/book1> ns:price 42 .
<http://example/book1> dc:creator "*A.N.Other*" 42 .

http://www.w3.org/TR/sparql11-update/

Insert Data with Blank Node

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
INSERT DATA {
   _:book1 dc:title   "A new book" ; dc:creator "A.N.Other" .
}
```

Data before:

```
# Default Graph
@prefix dc: <http://purl.org/dc/elements/1.1/> .
<http://example/book1> dc:title "Fundamentals of Compiler Design" .
```

Data after:

```
# Default Graph
@prefix dc: <http://purl.org/dc/elements/1.1/> .
<http://example/book1> dc:title "Fundamentals of Compiler Design" .
_:b1 dc:title   "A new book"
_:b1 dc:creator "A.N.Other"
```

Delete Ground Triples

```
PREFIX dc: http://purl.org/dc/elements/1.1/
DELETE DATA {
  <http://example/book2> dc:title "David Copperfield" ;  dc:creator "Edmund Wells" .
}
```

Data before:

```
# Default graph
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ns: <http://example.org/ns#> .
<http://example/book2> ns:price 42 .
<http://example/book2> dc:title "David Copperfield" .
<http://example/book2> dc:creator "Edmund Wells" .
```

Data after:

```
# Default graph
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ns: <http://example.org/ns#> .
<http://example/book2> ns:price 42 .
```


http://www.w3.org/TR/sparql11-update/

Delete-Insert

```
PREFIX foaf:  http://xmlns.com/foaf/0.1/
WITH <http://example/addresses>
DELETE { ?person foaf:givenName 'Bill' }
  INSERT { ?person foaf:givenName 'William' }
  WHERE { ?person foaf:givenName 'Bill'  }
```

Data before:

```
# Graph: http://example/addresses
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .
<http://example/president25> foaf:givenName "Bill" .
<http://example/president25> foaf:familyName "McKinley" .
<http://example/president27> foaf:givenName "Bill" .
<http://example/president27> foaf:familyName "Taft" .
<http://example/president42> foaf:givenName "Bill" .
<http://example/president42> foaf:familyName "Clinton" .
```

Data after:

```
# Graph: http://example/addresses
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .
<http://example/president25> foaf:givenName "William" .
<http://example/president25> foaf:familyName "McKinley" .
<http://example/president27> foaf:givenName "William" .
<http://example/president27> foaf:familyName "Taft" .
<http://example/president42> foaf:givenName "William" .
<http://example/president42> foaf:familyName "Clinton" .
```

Enabling Convergence in LLD

* Operations can be received on different sites in different orders.
* If the application of all operations over any state commutes, the object is Conflict-Free
  * $S \circ op_i \circ op_j = S \circ op_j \circ op_i$ => Convergence
* Basic SPARQL update application does not commute

  [INSERT DATA(x); DELETE DATA(x);INSERT DATA(x)] = {x}

  [INSERT DATA(x); INSERT DATA(x);DELETE DATA(x) = { }

RDF Data-Stores are not CRDTs

Enabling Intentions Preservation in LLD

* Observed effect of SPARQL update queries at generation time should be preserved at re-execution time…
  * If an Sparql query inserts a triple at generation time, this triple will be added at all sites whatever the state of the store at re-execution time…
* Intentions can be impossible to preserve (non-deterministic operations) or impossible without more order constraints (re-execution time evaluation)

```
PREFIX foaf:  http://xmlns.com/foaf/0.1/
WITH <http://example/addresses>
DELETE { ?person foaf:givenName 'Bill' }
INSERT { ?person foaf:givenName 'William' }
WHERE { ?person foaf:givenName 'Bill'  }
```

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
INSERT DATA {
  _:book1 dc:title   "A new book" ;
          dc:creator "A.N.Other" .
}
```

A CRDT for SPARQL update

* Rewrite SPARQL UPDATE Operations to another type that does commute, while preserving the original semantics.
* Graph Update operations work over RDF-Graphs, and can be expressed in terms of the basic INSERT and DELETE of ground triples.
  * RDF-Graphs are Sets of RDF-Triples, and CRDTs for Sets with Insert and Delete already exists…

SU-Set: A Sparql-Update CRDT

```
payload set S
    initial ∅
query lookup (triple t) : boolean b
    let b = (∃u | : (t, u) ∈ S)
update insert (Set<triple> T)
    prepare(T)
        let T' = ∅
        foreach t in T:
            if (!lookup(t)):
            let α = unique() // returns unique value
            T' := T' ∪ {(t, α)}
            endif
    effect(T')
        S := S ∪ T'
update delete (Set<triple> T)
    prepare(T)
        let T' = ∅
        foreach t in T:
            if (lookup(t)):
             T' := T' ∪ {u | : (t, u) ∈ S}
            endif
    effect(T')
        let R = {(t, u) | (∃u : u ∈ T')}
      // Causal delivery
      pre ∀(t, u) ∈ R : add(t, u) has been delivered
      S := S \ R
```

Abstract Operation is SPARQL Update

One id per inserted triple
Communication Expensive

Need only to send ids to delete:
Communication Inexpensive

Not good for LLD, as knowledge grows…

SU-Set : A Sparql-Update CRDT

```
payload set S
    initial ∅
query lookup (triple e) : boolean b
    let b = (∃u : (t, u) ∈ S)
update insert (set<triple> T)
    prepare(T)
        let T' = ∅
        foreach t in T:
            if (!lookup(t)) then:
            T' := T' ∪ {(t, α)}
            endif
        let α = unique()
    effect(R, α)
        foreach t in R:
          S := S ∪ {(t, α)}
update delete (set<triple> T)
    prepare(T)
        let R = ∅
        foreach t in T:
            let Q = {(t, u) | (∃u : | (t, u) ∈ S)}
            R := R ∪ Q
    effect(R)
        //Causal Delivery
        pre ∀(t, u) ∈ R : add(t, u) has been delivered
        S := S \ R
```

Same id for all triples inserted together, unicity is preserved
Less expensive in Communication

Need to send (triple,id) when deleting
More expensive in communication

Better for LLD as knowledge grows…

SU-Set

```
update delete − insert(whrPat, delPat, insPat)
    // match(m, pattern): triples that match
    //     pattern within mapping m.
    prepare(whrPat, delPat, insPat)
        let S' = {t | (∃u | : (t, u) ∈ S)}
        // M is a Multiset of mappings
        let M = eval(Select *
                     from S' where whrPat)
        D' = ∅
        foreach m in M:
            let D' = D' ∪ match(m, delPat)
        let D = {(t, u) : t ∈ D' ∧ (t, u) ∈ S}
        foreach m in M:
            let I' = I' ∪ match(m, insPat)
        let α = unique()
    effect(D, I, α)
        // Causal Reception
        pre All add(f,u) ∈ D have been delivered
        S := (S \ D)
        foreach t in I:
            S := S ∪ {(t, α)}
```

SU-Set on Live Linked Data

Insert Data
{France in Asia}

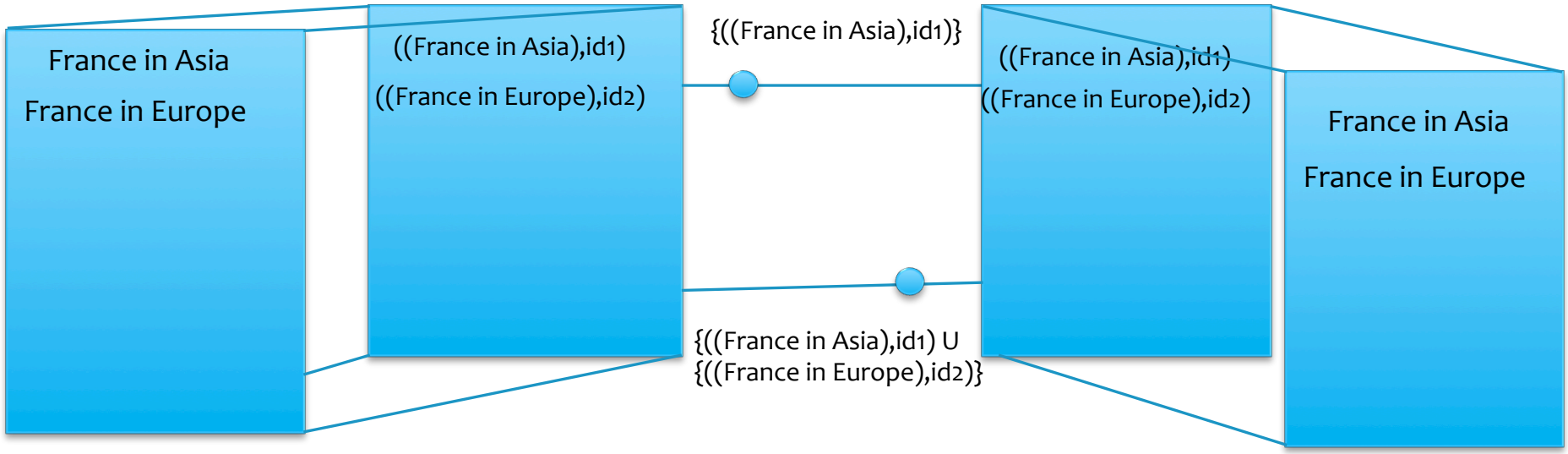
Insert({(France in Asia)})

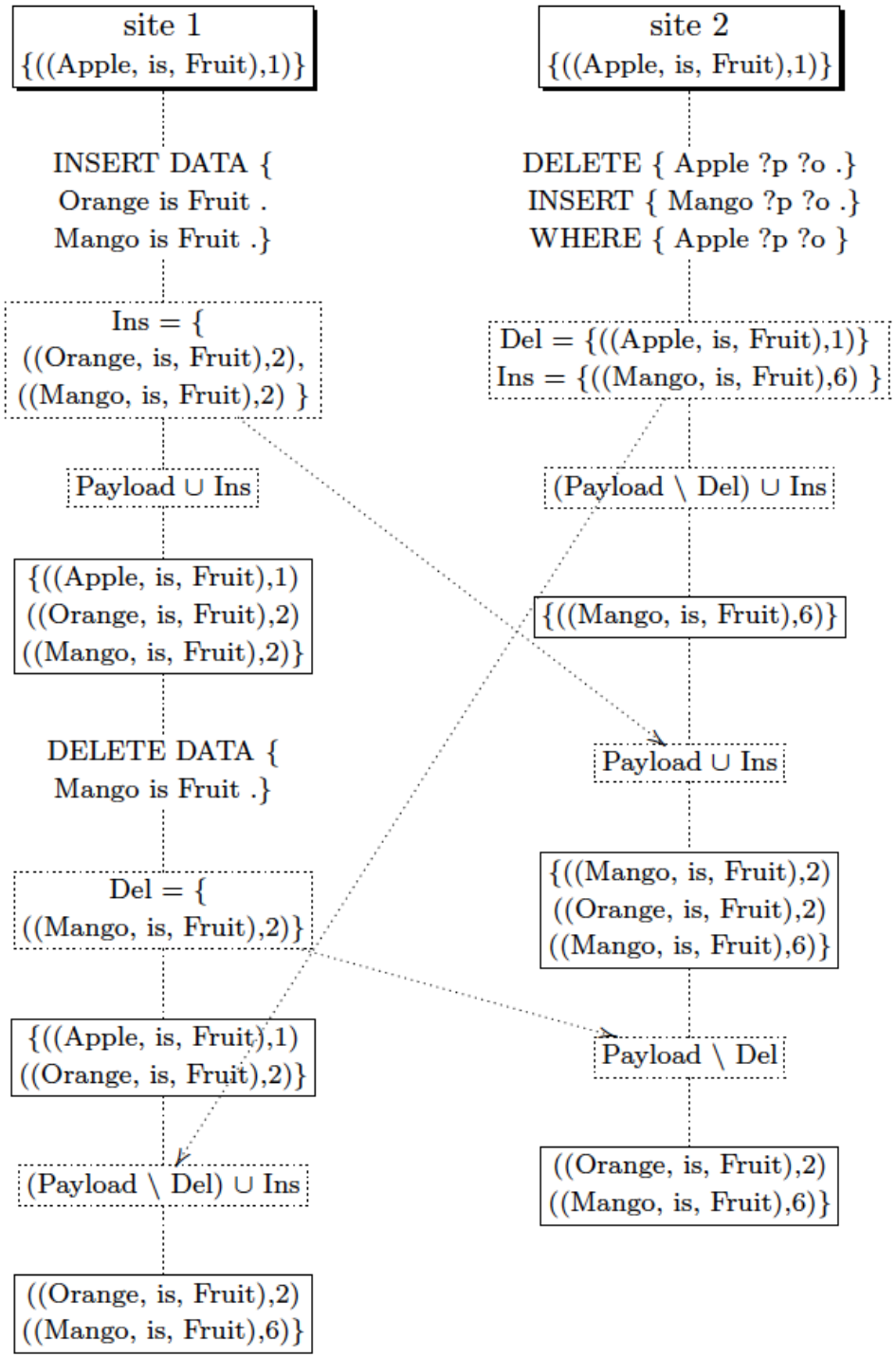
Payload U
{((France in Asia),id1)}

Delete(
{(France in Asia),
(France in Europe)})

Payload \
{((France in Asia),id1)
{((France in Europe),id2)}

DELETE {?c in Asia}
INSERT {?c in Europe}
WHERE {?c in Asia}




site 1
{((Apple, is, Fruit),1)}
INSERT DATA {
Orange is Fruit .
Mango is Fruit .}
Ins = {
((Orange, is, Fruit),2),
((Mango, is, Fruit),2) }
Payload ∪ Ins
{((Apple, is, Fruit),1)
((Orange, is, Fruit),2)
((Mango, is, Fruit),2)}
DELETE DATA {
Mango is Fruit .}
Del = {
((Mango, is, Fruit),2)}
{((Apple, is, Fruit),1)
((Orange, is, Fruit),2)}
(Payload \ Del) ∪ Ins
((Orange, is, Fruit),2)
((Mango, is, Fruit),6)}
site 2
{((Apple, is, Fruit),1)}
DELETE { Apple ?p ?o .}
INSERT { Mango ?p ?o .}
WHERE { Apple ?p ?o }
Del = {((Apple, is, Fruit),1)}
Ins = {((Mango, is, Fruit),6) }
(Payload \ Del) ∪ Ins
{((Mango, is, Fruit),6)}
Payload ∪ Ins
{((Mango, is, Fruit),2)
((Orange, is, Fruit),2)
((Mango, is, Fruit),6)}
Payload \ Del
((Orange, is, Fruit),2)
((Mango, is, Fruit),6)}

SU-SET – Drawbacks

* Concurrent insertions generate pairs with the same triple and different ids.
  * But we can ease this at the cost of some communication…

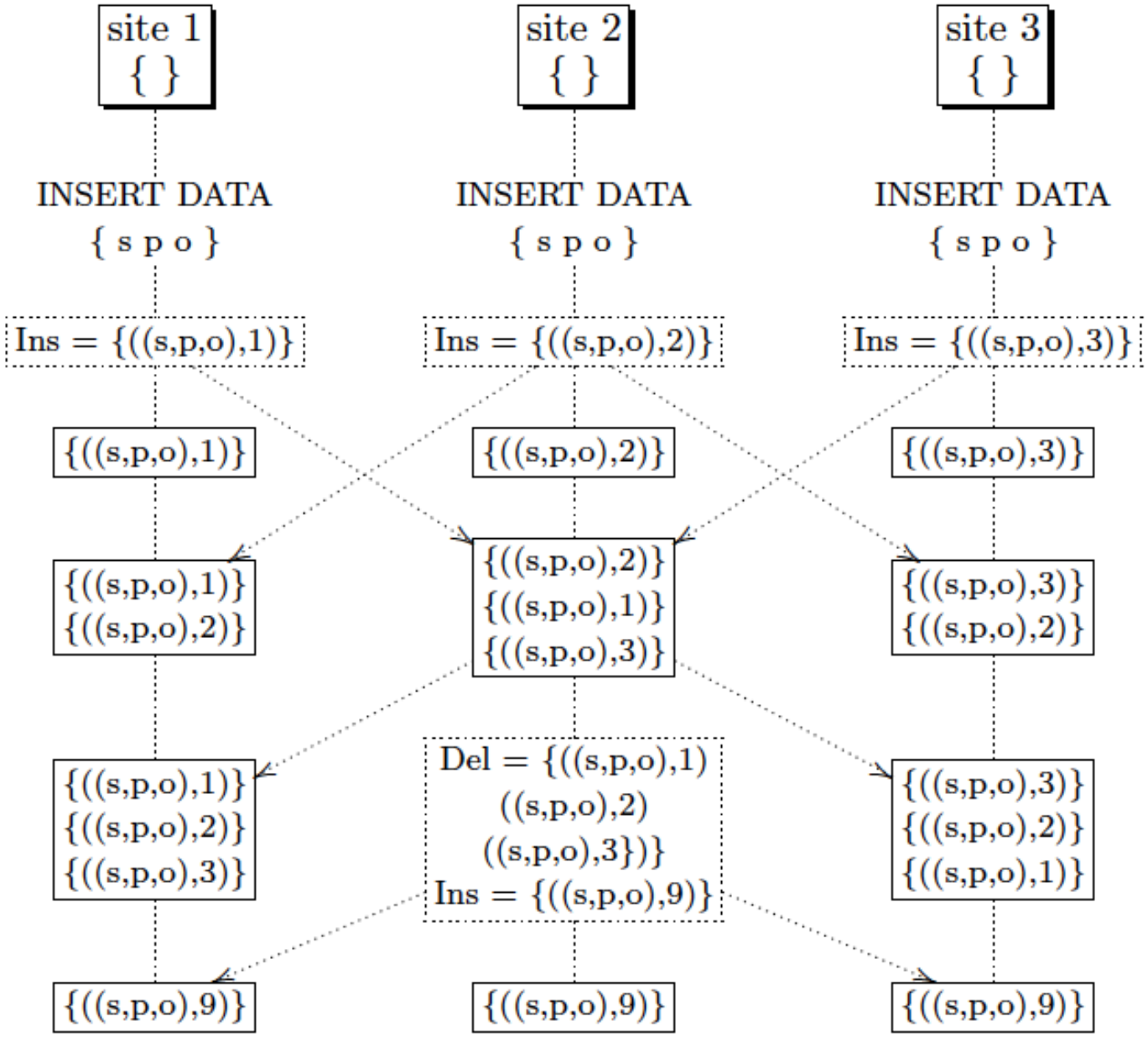
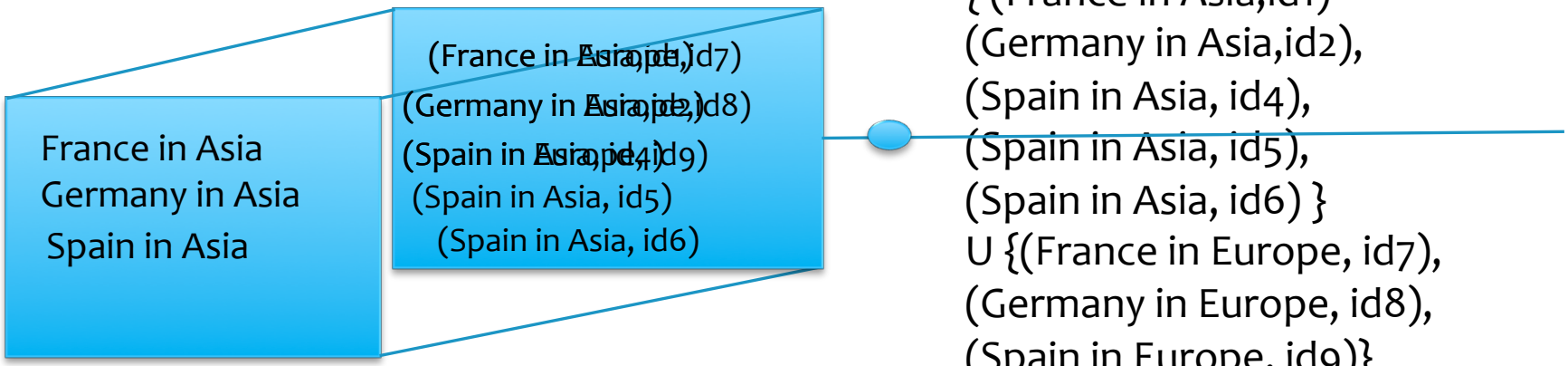


Figure 7 Generation and garbage collection of pairs referring to the same triple

SPARQL-Update Pattern Queries



Delete {?c in Asia}
Insert {?c in Europe}
Where {?c in Asia}

Payload \ { (France in Asia,id1)
(Germany in Asia,id2),
(Spain in Asia, id4),
(Spain in Asia, id5),
(Spain in Asia, id6) }
U {(France in Europe, id7),
(Germany in Europe, id8),
(Spain in Europe, id9)}

Expensive!

Implementation

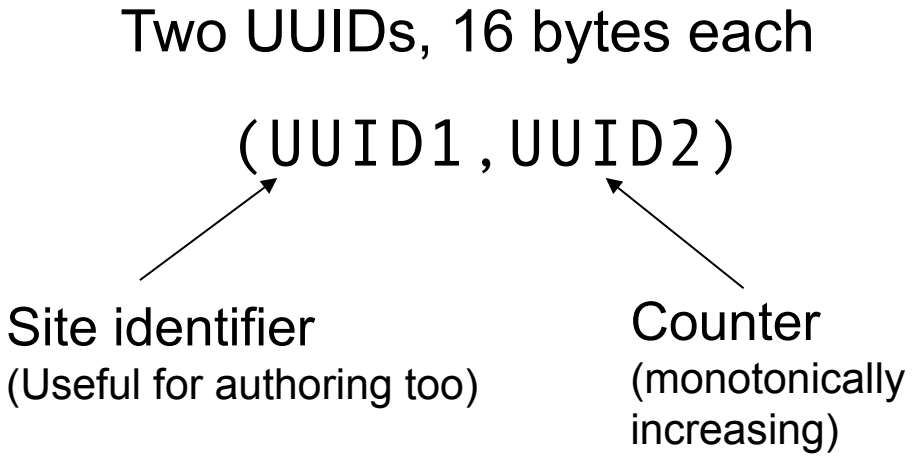
* SU-Set embedded into Corese, a reference implementation of RDF-Stores and SPARQL Update by Wimmics team.
  * Rewrite Sparql Update into SUSet, execute and log.
  * “Follow your change” implemented with anti-entropy over logs and version vectors.

How much we need to pay to have eventual consistency ?

* Time Overhead :
  * Adding an id to each element is linear.
  * Selection and lookup is not affected by many pairs with the same triple.
* Round and # of messages Overhead :
  * Convergence after one round, one message per operation → Optimal

Validation – Space Overhead

* 32 bytes per 1 billion triples = 32 GB → 1 Ipod
* Semantic Stores already use an internal id → Reuse it
* Extra pairs produced by concurrent insertions could cause problems…
* A version vector for each site: Max size= number of participants (~300-800).



Dynamicity: DBPedia Live

* Framework to register changes in DBPedia in near real-time.
* Generates one file with inserted triples and one with deleted triples approximately each 10 seconds.
  * No pattern operations logged ➔ No Overhead here.
* **Many more insertions than deletions**
  * Good for SU-Set.
* Many triples inserted per operation
  * Even better for SU-Set

SU-Set Communication overhead on DBPedia Live

7 days of streaming
No concurrent insertions
IdSize[1]: 0,096 Kb
Avg triple size[1]: 0,155 Kb

Size (MB)

Operation	# of Triples	Without ids	1 id per triple	1 id per operation
21957 Inserts	21762190	3294,08	5334,29	3296,6
21957 Deletes	1755888	265,78	164,61	430,4
Overhead			54,47%	4,68%

[1]Measured with 'wc –c' over DBPedia Live log files

Conclusion

* Live Linked Data makes linked data “writable” and allows a new query paradigm
* SU-Set is a CRDT for RDF-Graphs updated with SPARQL-Update 1.1 that ensure eventual consistency and intentions on Live Linked Data.
* Future work:
  * Finish and release LLD-Corese.
  * Write the composed CRDT for RDF-Datasets
  * Benchmark Live Linked Data.

Communication Complexity

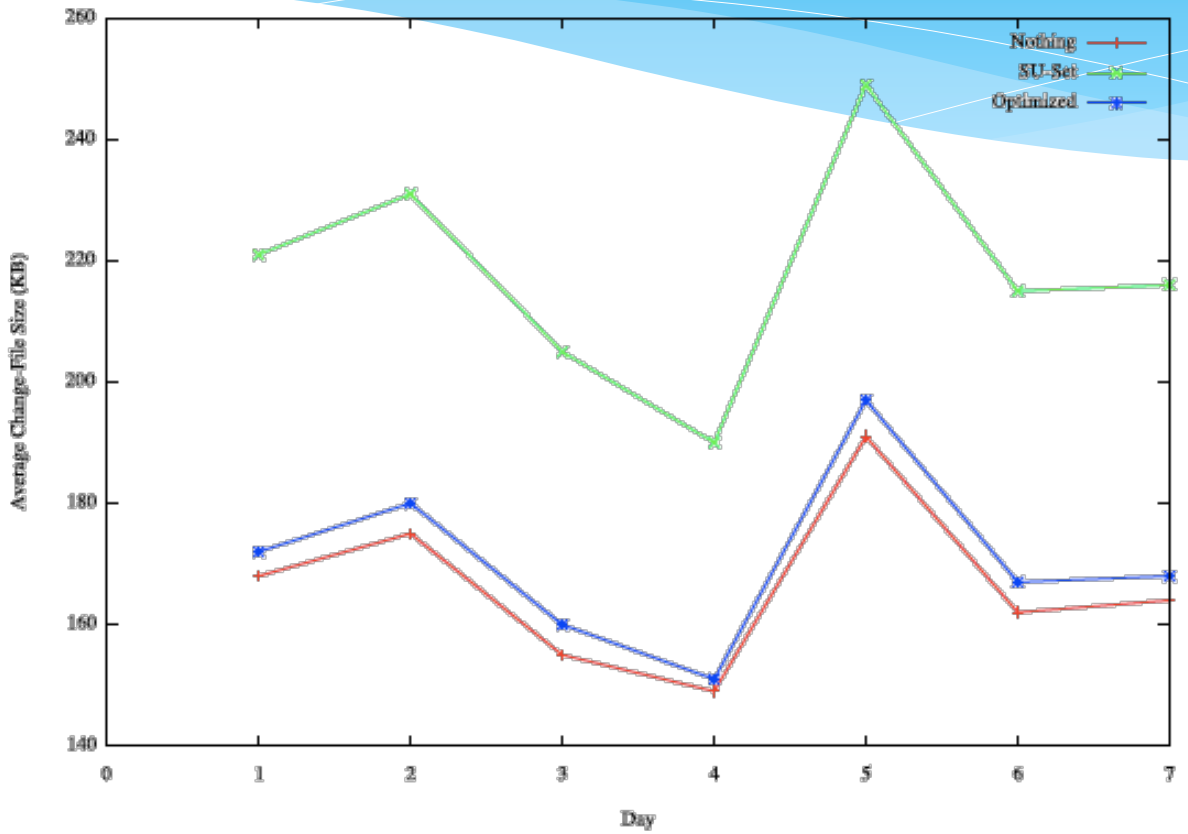


Figure 9 Communication complexity for SU-Set and its optimized version in DBPedia Live case.

But what about causality?

* Causal delivery has a hidden cost…
  * Tombstones Maintenance
    * Impractical with DBPedia Live dynamicity parameters (1755888 triples deleted)
  * Vector clocks
    * Requires piggybacking each message with the full vector, an 800 entries vector attached to each message is too expensive.
  * Anti-Entropy
    * State-based is expensive when datasets are big…

Future Work

* OptOR-Set to do the ant

Log-based Anti-Entropy on LLD

France in Asia
China in Asia
Japan in Asia

1: Ins((France in Asia, site1-1)
2: Ins({(China in Asia),(Japan in Asia},site1-2)
3: Del((France in Asia, site1-1))
4: Ins((France in Europe, site1-3))

Future work

* Divergence – How to evaluate divergence between node in live linked data
* Provenance – How to collect it efficiently in LLD
* Resource discovery in LLD
* Queries with weakly consistent replicated datasets
* More efficient pattern operations are possible ?

Luis Daniel Ibáñez, Hala Skaf-Molli, Pascal Molli, and Olivier Corby. 2012. Synchronizing semantic stores with commutative replicated data types. In Proceedings of the 21st international conference companion on World Wide Web (WWW '12 Companion). ACM, New York, NY, USA, 1091-1096. DOI=10.1145/2187980.2188246 http://doi.acm.org/10.1145/2187980.2188246

LOD Stats

* Billion Triple Challenge to gather linked data (2011 – Crawling)
  * 2,145 billion quadruples extracted from 7,411 million RDF/XML Document
  * Extracted from 791 pay-level domain (PLD)
* CKAN (lod cloud, constraints on selections)
  * 297 datasets (133 PLD)
  * 28,4 billions of triples

http://events.linkeddata.org/ldow2012/papers/ldow2012-paper-14.pdf

http://dbpedia.org/page/France

dbpedia-owl:anthem	▪ dbpedia:La_Marseillaise
dbpedia-owl:areaTotal	▪ 674842122052.927490 (xsd:double) ▪ 674843000000.000000 (xsd:double)
dbpedia-owl:capital	▪ dbpedia:Paris
dbpedia-owl:currency	▪ dbpedia:CFP_franc
dbpedia-owl:demonym	▪ French
dbpedia-owl:governmentType	▪ dbpedia:Unitary_state
dbpedia-owl:language	▪ dbpedia:French_language
dbpedia-owl:largestCity	▪ dbpedia:Prefectures_in_France
dbpedia-owl:leaderName	▪ dbpedia:Jean-Marc_Ayrault ▪ dbpedia:François_Hollande
dbpedia-owl:leaderTitle	▪ President ▪ Prime Minister
dbpedia-owl:longName	▪ French Republic
dbpedia-owl:motto	▪ (Liberty, Equality, Fraternity)
dbpedia-owl:officialLanguage	▪ dbpedia:French_language
dbpedia-owl:populationDensity	▪ 116.000000 (xsd:double) ▪ 116.216750 (xsd:double)

Linked Data Problems

Category	Problem
Incompleteness	Dereferencability issues
	No structured data available
	Misreported content types
	RDF/XML Syntax Errors
Incoherence	Atypical use of collections, containers and reification
	Use of undefined classes and properties
	Misplaced classes/properties
	Misuse of *owl:DatatypeProperty (ObjectProperty)*
	Members of deprecated classes/properties
	Malformed datatype literals
	Literals incompatible with datatype range
	Ontology hijacking
Hijacking	Bogus *owl:InverseFunctionalProperty* values
	Ontology hijacking
Inconsistencies	Literals incompatible with datatype range
	OWL inconsistencies

http://factforge.net/

$S := S \cup R$

update *delete* (set<triple> T)

atSource(T)

let $R = \emptyset$

foreach t **in** T:

let $Q = \{(t,u) \mid (\exists u : \mid (t,u) \in S)$

$R := R \cup Q$

downstream(R)

// *Causal Reception*

pre All add(t,u) delivered

$S := S \setminus R$

SU-Set and Causality problem

* in fact Ins(t), del(t) does not commute, to avoid the problem
  * Require causal order, if op1 precedes op2 on one site, then it precedes on all sites… [Lamport79]
    * Vector clocks [Mattern89]
    * Anti-entropy [Demers87]
  * Death certificates, tomstones [Demers87]

Site1 arrival: {id-1-1 -> (

[<http://example/president25>, foaf:givenName ,"William"]

[<http://example/president25> ,foaf:familyName, "McKinley"])}

Site2 arrival: {id-2-1 -> (

[<http://example/president25>, foaf:givenName ,"Will"]

[<http://example/president25> ,foaf:familyName, "McKinley"])}

Data Before:

[id-3-10, <http://example/president25>, foaf:givenName ,"William"]

[id-3-40, <http://example/president25> ,foaf:familyName, "McKinley"]

Data After:

[id-3-10, <http://example/president25>, foaf:givenName ,"William"]

[id-2-1, <http://example/president25>, foaf:givenName ,"Will"]

[id-1-1, <http://example/president25>, foaf:givenName ,"William"]

[id-3-40, <http://example/president25> ,foaf:familyName, "McKinley"]

[id-1-1, <http://example/president25> ,foaf:familyName, "McKinley"]

[id-2-1, <http://example/president25> ,foaf:familyName, "McKinley »]