

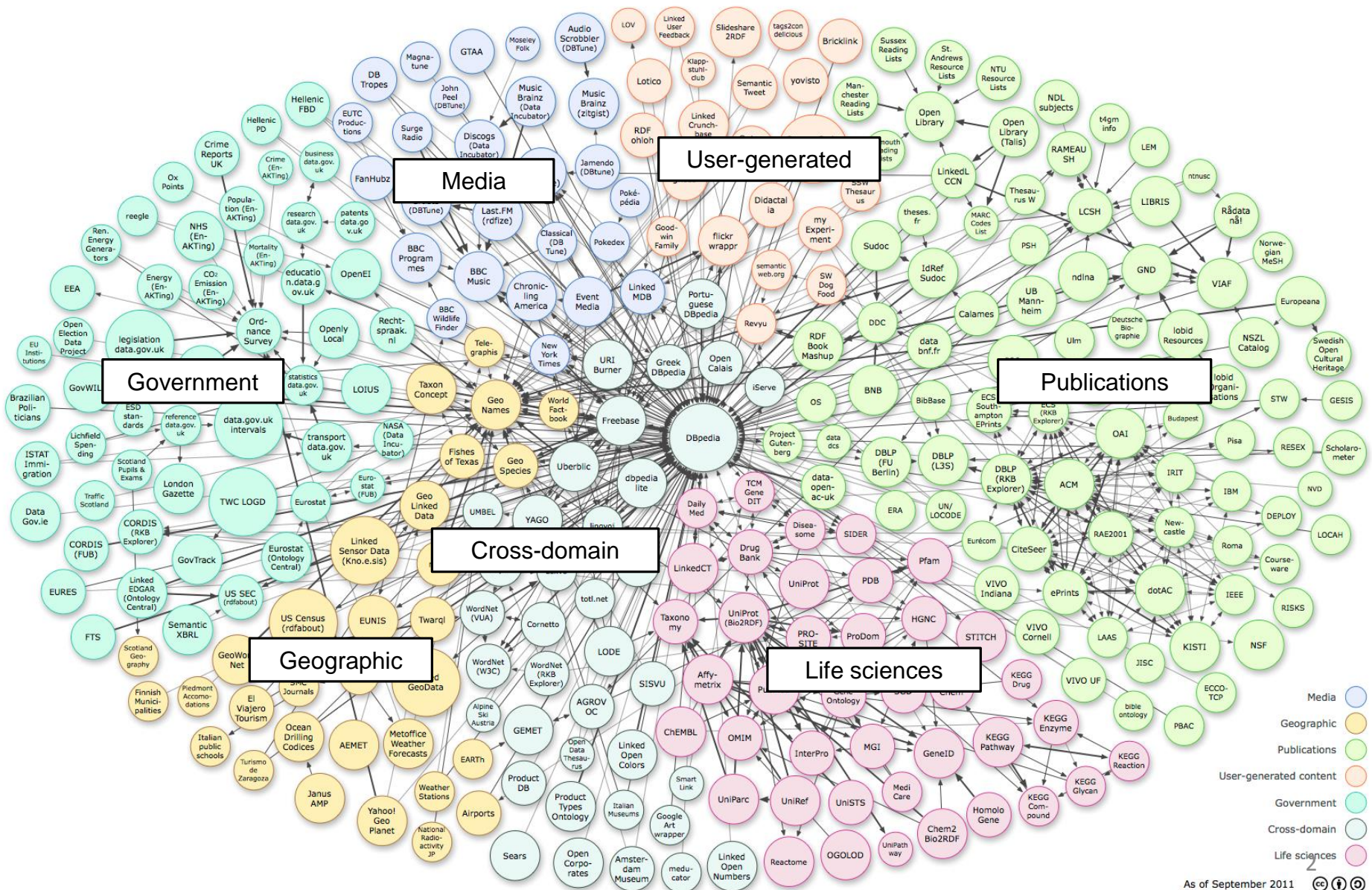
Entity Resolution in the Web of Data

Kostas Stefanidis¹, Vasilis Efthymiou^{1,2},
Melanie Herschel^{3,4}, Vassilis Christophides⁵

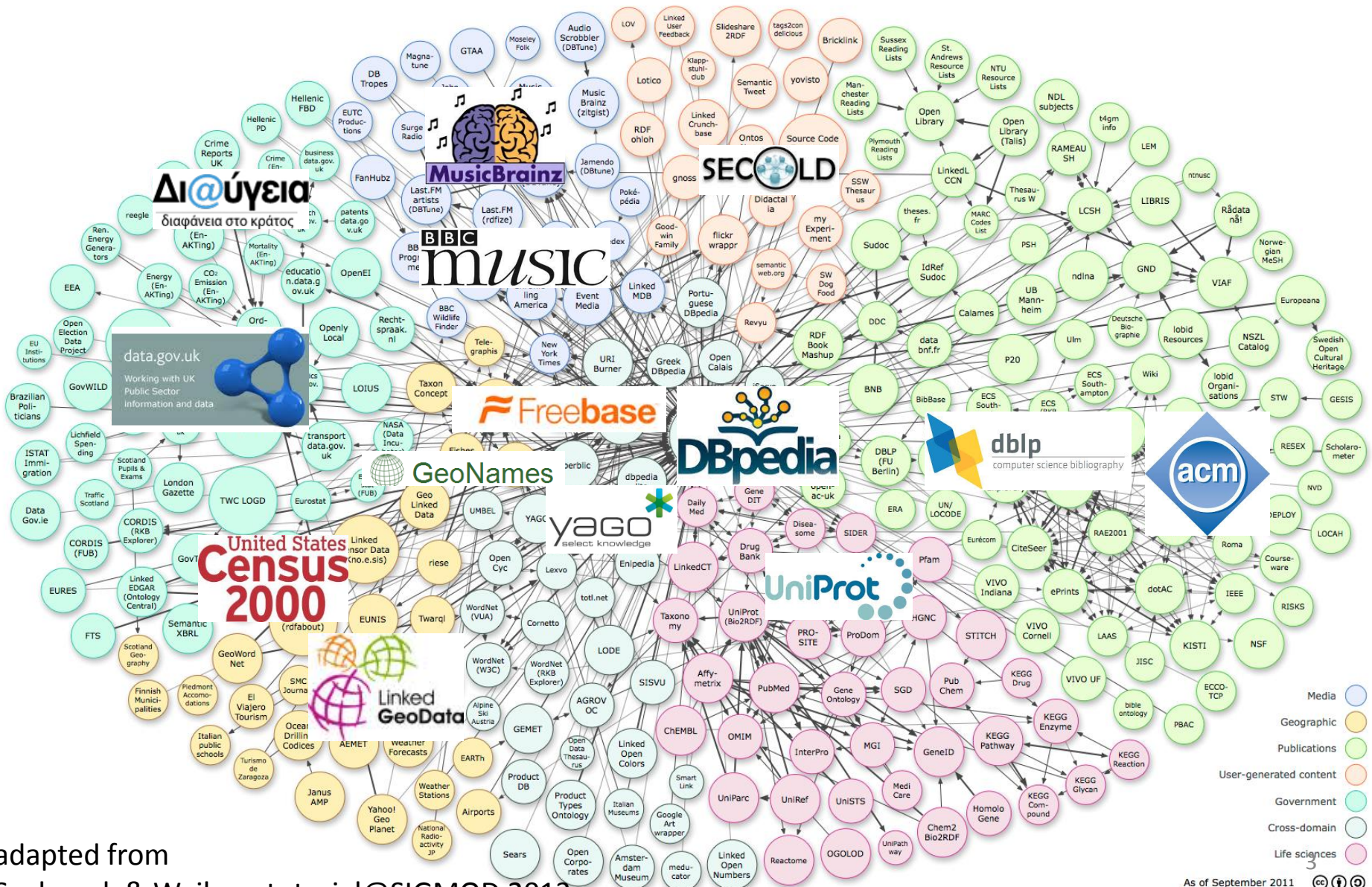
kstef@ics.forth.gr, vefthym@ics.forth.gr, melanie.herschel@lri.fr
vassilis.christophides@technicolor.com


¹FORTH, ²University of Crete, ³Université Paris Sud, ⁴Inria Saclay,
⁵Paris R&I Center, Technicolor

LOD Cloud and the Web of Data

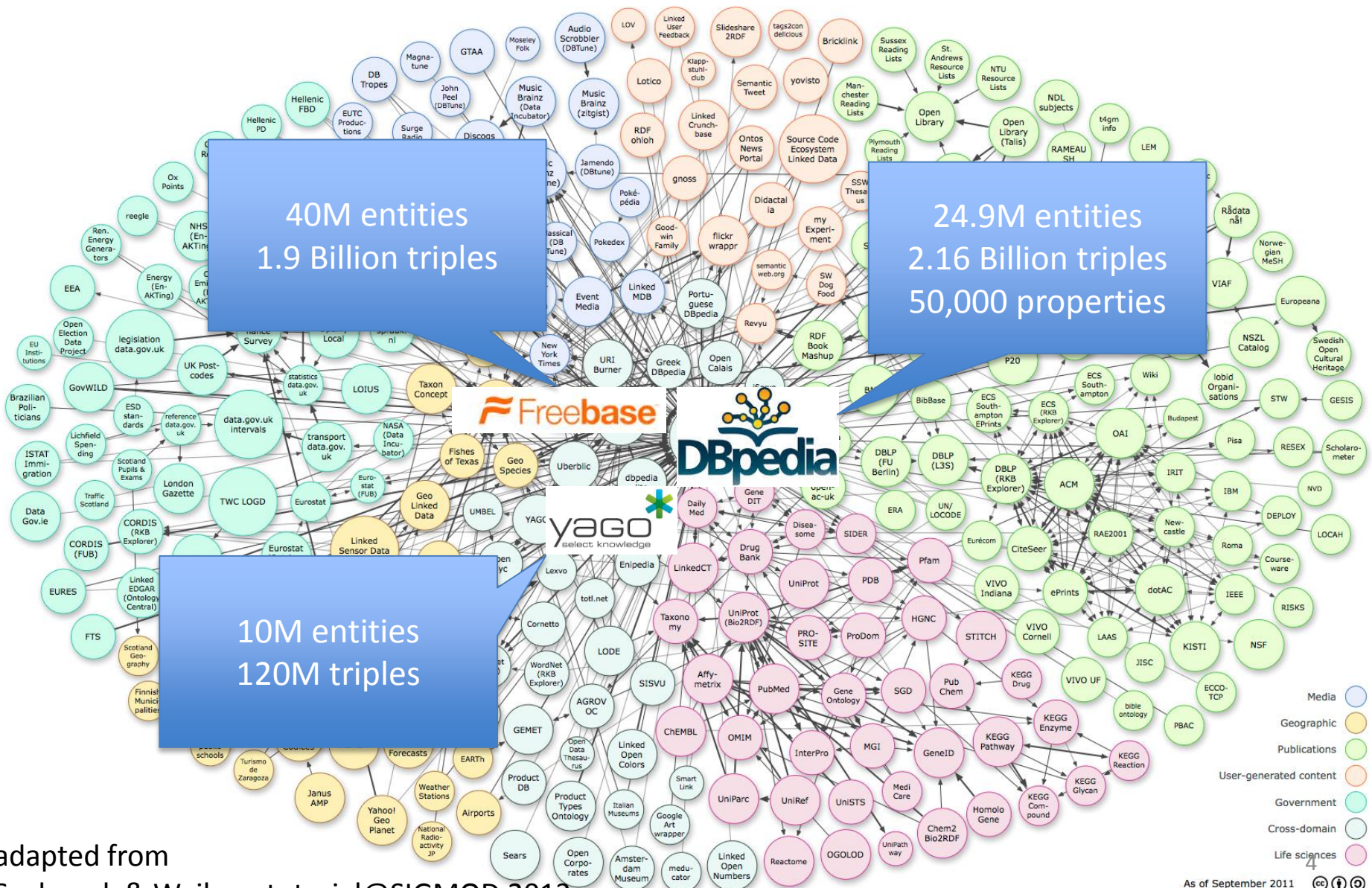


LOD Cloud and the Web of Data



As of September 2011 

LOD Cloud and the Web of Data



As of September 2011

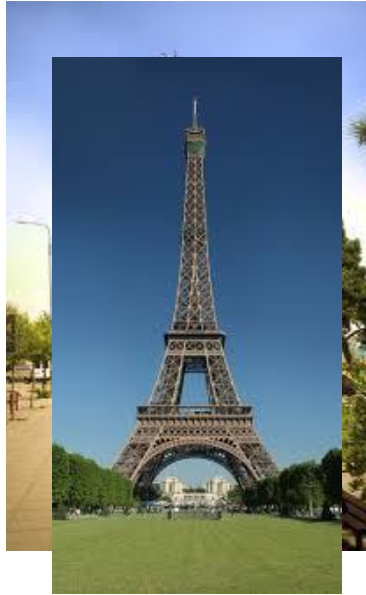
Entities: An Invaluable Asset



Monuments

“Entities” is what a large part of our knowledge is about

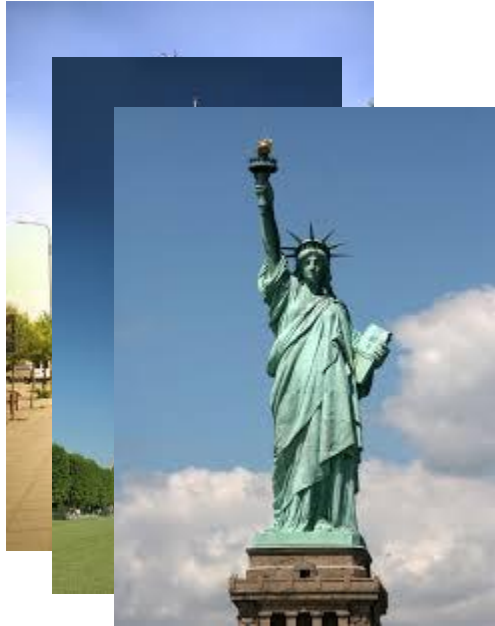
Entities: An Invaluable Asset



Monuments

“Entities” is what a large part of our knowledge is about

Entities: An Invaluable Asset



Monuments

“Entities” is what a large part of our knowledge is about

Entities: An Invaluable Asset



Locations



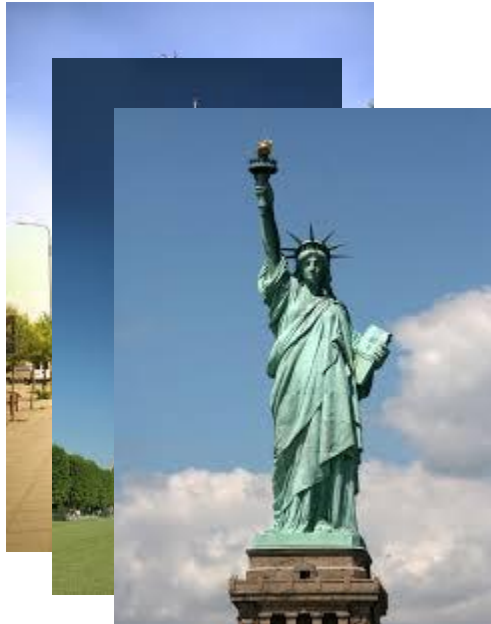
Monuments

“Entities” is what a large part of our knowledge is about

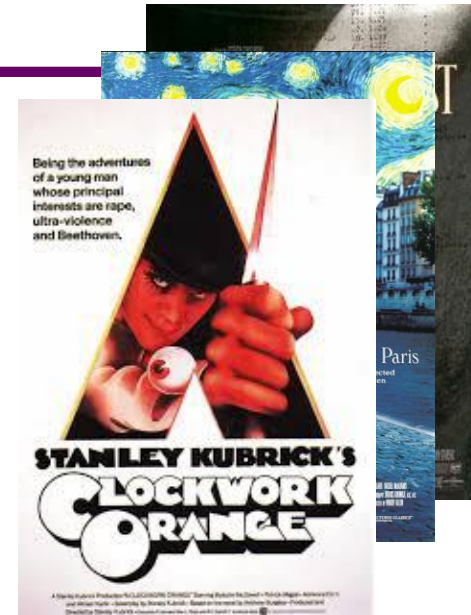
Entities: An Invaluable Asset



Locations



Monuments

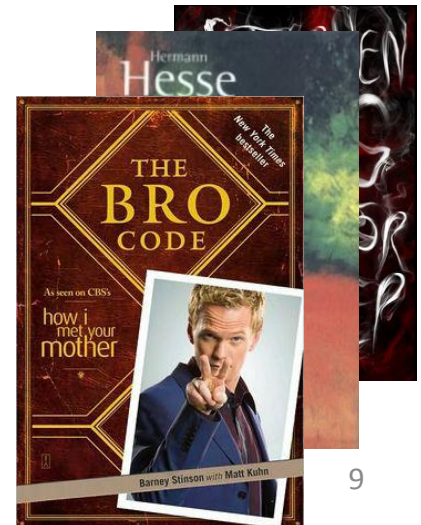


Movies

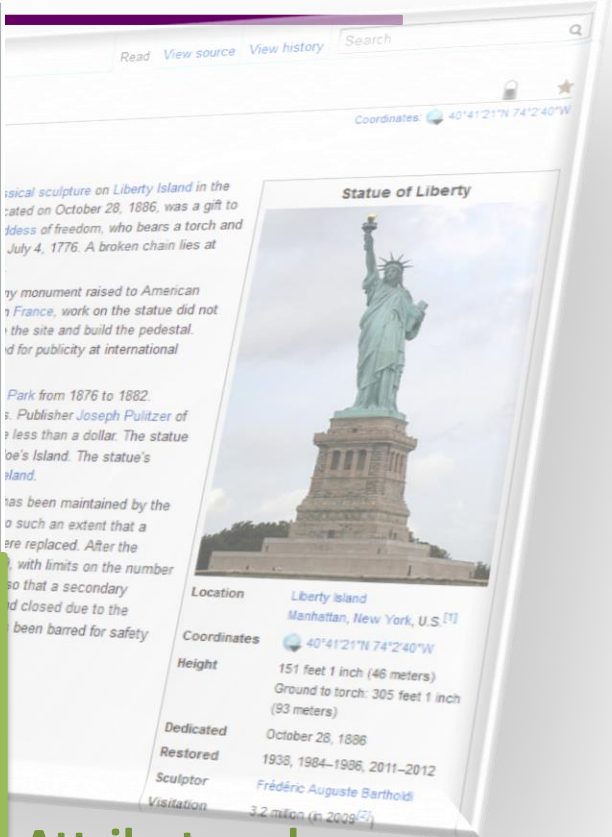



Persons

Books





Example: General Knowledge Bases




Attribute names	Attribute values
Location	Liberty Island Manhattan, New York, U.S. ^[1]
Coordinates	 40°41′21″N 74°2′40″W
Height	151 feet 1 inch (46 meters) Ground to torch: 305 feet 1 inch (93 meters)
Dedicated	October 28, 1886
Restored	1938, 1984–1986, 2011–2012
Sculptor	Frédéric Auguste Bartholdi
Visitation	3.2 million (in 2009) ^[2]


Different Descriptions of the same Entity


	dbpedia:Statue_of_Liberty
rdfs:label	Statue of Liberty, Freiheitsstatue, ...
dbpprop:location	New York City, New York, U.S., dbpedia:Liberty_Island
dbpprop:sculptor	dbpedia:Frédéric_Auguste_Bartholdi
dcterms:subject	dbpedia_category:1886_sculptures , ...
foaf:isPrimaryTopicOf	http://en.wikipedia.org/wiki/Statue_of_Liberty
dbpprop:beginningDate	1886-10-28 (xsd:date)
dbpprop:restored	19381984 (xsd:integer)
dbpprop:visitationNum	3200000 (xsd:integer)
dbpprop:visitationYear	2009 (xsd:integer)
http://www.w3.org/ns/prov#wasDerivedFrom	http://en.wikipedia.org/wiki/Statue_of_Liberty?oldid=494328330


	fb:m.072p8
fb:art_form	fb:m.06msq (Sculpture)
fb:media	fb:m.025rsfk (Copper)
fb:architect	fb:m.0jph6 (F. Bartholdi), fb:m.036qb (G. Eiffel), fb:m.02wj4z (R. Hunt)
fb:height_meters	93
fb:opened	1886-10-28

	yago:Statue_of_Liberty
skos:prefLabel	Statue of Liberty
rdf:type	yago:History_museums_in_NY , yago:GeoEntity
yago:hasHeight	46.0248
yago:wasCreatedOnDate	1886-##-##
yago:isLocatedIn	yago:Manhattan , yago:Liberty_Island ,
yago:hasWikipediaUrl	http://en.wikipedia.org/wiki/Statue_of_Liberty

Linked Datasets Depend on Vocabularies


	dbpedia:Statue_of_Liberty
rdfs:label	Statue of Liberty, Freiheitsstatue, ...
dbpprop:location	New York City, New York, U.S., dbpedia:Liberty_Island
dbpprop:sculptor	dbpedia:Frédéric_Auguste_Bartholdi
dcterms:subject	dbpedia_category:1886_sculptures , ...
foaf:isPrimaryTopicOf	http://en.wikipedia.org/wiki/Statue_of_Liberty
dbpprop:beginningDate	1886-10-28 (xsd:date)
dbpprop:restored	19381984 (xsd:integer)
dbpprop:visitationNum	3200000 (xsd:integer)
dbpprop:visitationYear	2009 (xsd:integer)
http://www.w3.org/ns/prov#wasDerivedFrom	http://en.wikipedia.org/wiki/Statue_of_Liberty?oldid=494328330


	fb:m.072p8
fb:art_form	fb:m.06msq (Sculpture)
fb:media	fb:m.025rsfk (Copper)
fb:architect	fb:m.0jph6 (F. Bartholdi), fb:m.036qb (G. Eiffel), fb:m.02wj4z (R. Hunt)
fb:height_meters	93
fb:opened	1886-10-28

	yago:Statue_of_Liberty
skos:prefLabel	Statue of Liberty
rdf:type	yago:History_museums_in_NY , yago:GeoEntity
yago:hasHeight	46.0248
yago:wasCreatedOnDate	1886-##-##
yago:isLocatedIn	yago:Manhattan , yago:Liberty_Island ,
yago:hasWikipediaUrl	http://en.wikipedia.org/wiki/Statue_of_Liberty

Linked Datasets Have Varying Quality

 DBpedia	dbpedia:Statue_of_Liberty
rdfs:label	Statue of Liberty, Freiheitsstatue, ...
dbpprop:location	New York City, New York, U.S., dbpedia:Liberty_Island
dbpprop:sculptor	dbpedia:Frédéric_Auguste_Bartholdi
dcterms:subject	dbpedia_category:1886_sculptures , ...
foaf:isPrimaryTopicOf	http://en.wikipedia.org/wiki/Statue_of_Liberty
dbpprop:beginningDate	1886-10-28 (xsd:date)
dbpprop:restored	19381984 (xsd:integer)
dbpprop:visitationNum	3200000 (xsd:integer)
dbpprop:visitationYear	2009 (xsd:integer)
http://www.w3.org/ns/prov#wasDerivedFrom	http://en.wikipedia.org/wiki/Statue_of_Liberty?oldid=494328330

 Freebase	fb:m.072p8
fb:art_form	fb:m.06msq (Sculpture)
fb:media	fb:m.025rsfk (Copper)
fb:architect	fb:m.0jph6 (F. Bartholdi), fb:m.036qb (G. Eiffel), fb:m.02wj4z (R. Hunt)
fb:height_meters	93
fb:opened	1886-10-28

 YAGO	yago:Statue_of_Liberty
skos:prefLabel	Statue of Liberty
rdf:type	yago:History_museums_in_NY , yago:GeoEntity
yago:hasHeight	46.0248
yago:wasCreatedOnDate	1886-##-##
yago:isLocatedIn	yago:Manhattan , yago:Liberty_Island ,
yago:hasWikipediaUrl	http://en.wikipedia.org/wiki/Statue_of_Liberty

The Problem Entity Resolution

We need to identify that all descriptions refer to the same real-world object

Entity resolution is the problem of identifying descriptions of the same entity within one or across multiple data sources

A prerequisite to several applications:

- Enable semantic search in terms of entities & relations (*on top of the web of text*)
- Interlink entity descriptions in autonomous sources (*strengthen the web of data*)
- Support deep reasoning using related ontologies (*create the web of knowledge*)

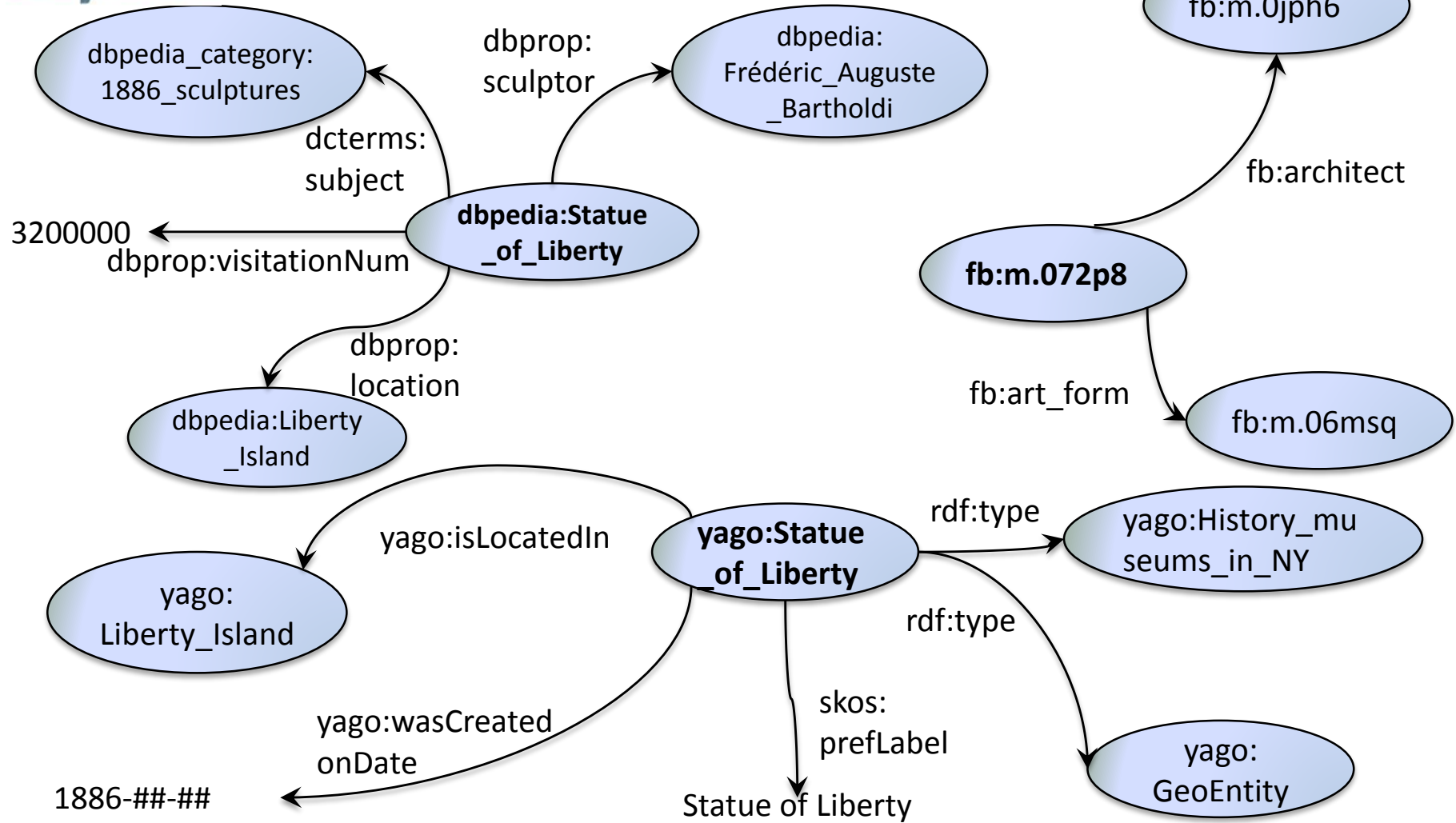
Entity Collections and Entity Resolution Types

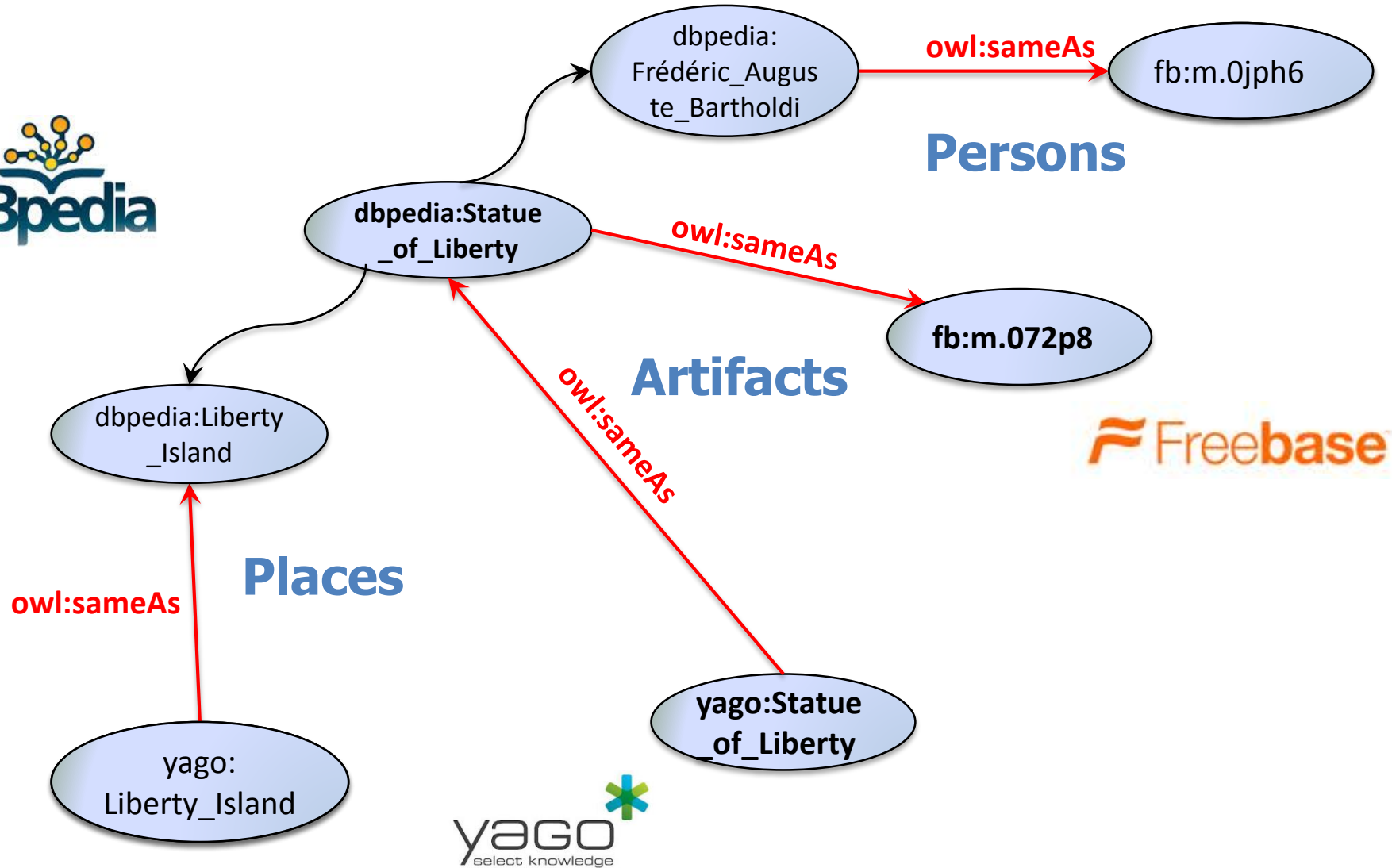
Two kinds of entity collections as input:

- Clean: duplicate-free
- Dirty: contains duplicate entity descriptions

An entity resolution task can be:

- Clean-Clean Entity Resolution: Given two clean, but overlapping entity collections, identify the common entity descriptions
 - a.k.a. *record linkage* in databases
- Dirty-Clean Entity Resolution
- Dirty Entity Resolution: Identify unique entity descriptions contained in one dirty entity collection
 - a.k.a. *deduplication* in databases





⇒ Need to infer also other kind of relationships than “equivalence”

What Makes Entity Resolution Difficult for the Web of Data

Linked Data are inherently semi-structured

- Several semantic types could be employed (see `rdf:type` properties in Yago), resulting to quite different structures even for entity descriptions of the same type (persons, places, ...)

=> Deal with loosely structured entities

Linked Data heavily rely on various vocabularies

- 366 distinct vocabulary spaces in the LOD cloud (<http://lov.okfn.org/dataset/lov/>)
- DBPedia 3.4: 50,000 attribute names

=> Need for cross-domain techniques

Linked Data are Big (semi-structured) Data

- LOD cloud: 60 billion RDF triples
- DBPedia 3.9: 2.46 billion triples, 24.9 million entity descriptions
- Freebase: 1.9 billion triples, 40 million entity descriptions
- Yago: >10 million entities, >120 million triples

=> Call for efficient parallel techniques

Problem Statement

Entity Description

Each description is expressed as a set of attribute-value pairs

An entity description $e_i \in E$ is defined as: $e_i = \{(a_{ij}, v_{ij}) \mid a_{ij} \in N, v_{ij} \in V\}$

N : a set of attribute names

V : a set of values

E : a set of entity descriptions

We use a generic definition for entity descriptions to cover different data models

Structural type of e_i : the set of attributes along with their domains in e_i

- In the Web of data, the descriptions even of the same entities do not always conform to the same structural type

Entity Description Examples

name	Eiffel Tower
architect	Sauvestre
year	1889
location	Paris

e1

about	Eiffel Tower
architect	Sauvestre
year	1889
located	Paris

e4

name	Statue of Liberty
architect	Bartholdi Eiffel
year	1886
located	NY

e2

about	Lady liberty
architect	Eiffel
location	NY

e3

name	White Tower
location	Thessaloniki
year-constructed	1450

e5

Entity Resolution – Formal Definition

Entity resolution: The problem of identifying descriptions of the same entity within one or across multiple data sources wrt. a match function

Formally:

$E = \{e_1, \dots, e_m\}$ is a set of entity descriptions

$M : E \times E \rightarrow \{\text{true}, \text{false}\}$ is a match function

An entity resolution of E is a partition $P = \{p_1, \dots, p_n\}$ of E , such that:

1. $\forall e_i, e_j \in E : M(e_i, e_j) = \text{true}, \exists p_k \in P : e_i, e_j \in p_k$
2. $\forall p_k \in P, \forall e_i, e_j \in p_k, M(e_i, e_j) = \text{true}$

each partition contains only matching descriptions

all the matching descriptions are in the same partition

Entity Resolution - Example

name	Eiffel Tower
architect	Sauvestre
year	1889
location	Paris

e1

about	Eiffel Tower
architect	Sauvestre
year	1889
located	Paris

e4

name	Statue of Liberty
architect	Bartholdi Eiffel
year	1886
located	NY

e2

about	Lady liberty
architect	Eiffel
location	NY

e3

name	White Tower
location	Thessaloniki
year-constructed	1450


e5

Assume as input of entity resolution, the set $E = \{e_1, e_2, e_3, e_4, e_5\}$


- A possible output $P = \{\{e_1, e_4\}, \{e_2, e_3\}, \{e_5\}\}$ indicates that:

Entity Resolution - Example


name	Eiffel Tower	
are	re	
ye		
loc		e1
ab	ower	
are	re	
ye		
loc		e4



name	Statue of Liberty	
are		
ye		
loc		e2
ab	ty	
are		
location	NY	e3



name		
location	oniki	
year		
con		e5



Assume as input of entity resolution, the set $E = \{e_1, e_2, e_3, e_4, e_5\}$

- A possible output $P = \{\{e_1, e_4\}, \{e_2, e_3\}, \{e_5\}\}$ indicates that:
 - e_1, e_4 refer to the same real-world object, the Eiffel Tower
 - e_2, e_3 represent a different object, the Statue of Liberty
 - e_5 represents a third object, the White Tower

Entity Resolution - Match

Matches: Sets of entity descriptions that refer to the same real-world entity

Intuitively:

- Matching entity descriptions are placed in the same subset of P
- All the descriptions of the same subset of P match

A match function maps each pair of entity descriptions (e_i, e_j) to $\{\text{true}, \text{false}\}$

- $M(e_i, e_j) = \text{true} \Rightarrow e_i, e_j$ are matching descriptions
- $M(e_i, e_j) = \text{false} \Rightarrow e_i, e_j$ are non-matches

Entity Resolution - Similarity

Typically, the match function is expressed wrt. a similarity measure sim

- *sim* counts how close two entity descriptions are to each other

Given a similarity threshold t :

- $M(e_i, e_j) = \text{true}$, if $\text{sim}(e_i, e_j) \geq t$
- $M(e_i, e_j) = \text{false}$, if $\text{sim}(e_i, e_j) < t$

Similarity of Entity Descriptions

How can we identify that two entity descriptions refer to the same entity?

Similarity of Entity Descriptions

How can we identify that two entity descriptions refer to the same entity?

- If they are identical, then we assume they match (exact match function)

E.g.

name	Eiffel Tower
architect	Sauvestre
year	1889
location	Paris

e1

name	Eiffel Tower
architect	Sauvestre
year	1889
location	Paris

e2

Similarity of Entity Descriptions

How can we identify that two entity descriptions refer to the same entity?

- If they are identical, then we assume they match (exact match function)
 - Even this assumption could be false!

E.g.

first	John
last	Doe
born	1980
location	UK

e1

first	John
last	Doe
born	1980
location	UK

e2

... could describe namesakes, born in the same country and year

Similarity of Entity Descriptions

How can we identify that two entity descriptions refer to the same entity?

- What if they are not identical, but it looks like they match?

– e.g.

about	Gustave Eiffel	e1	name	G. Eiffel	e2
-------	----------------	----	------	-----------	----

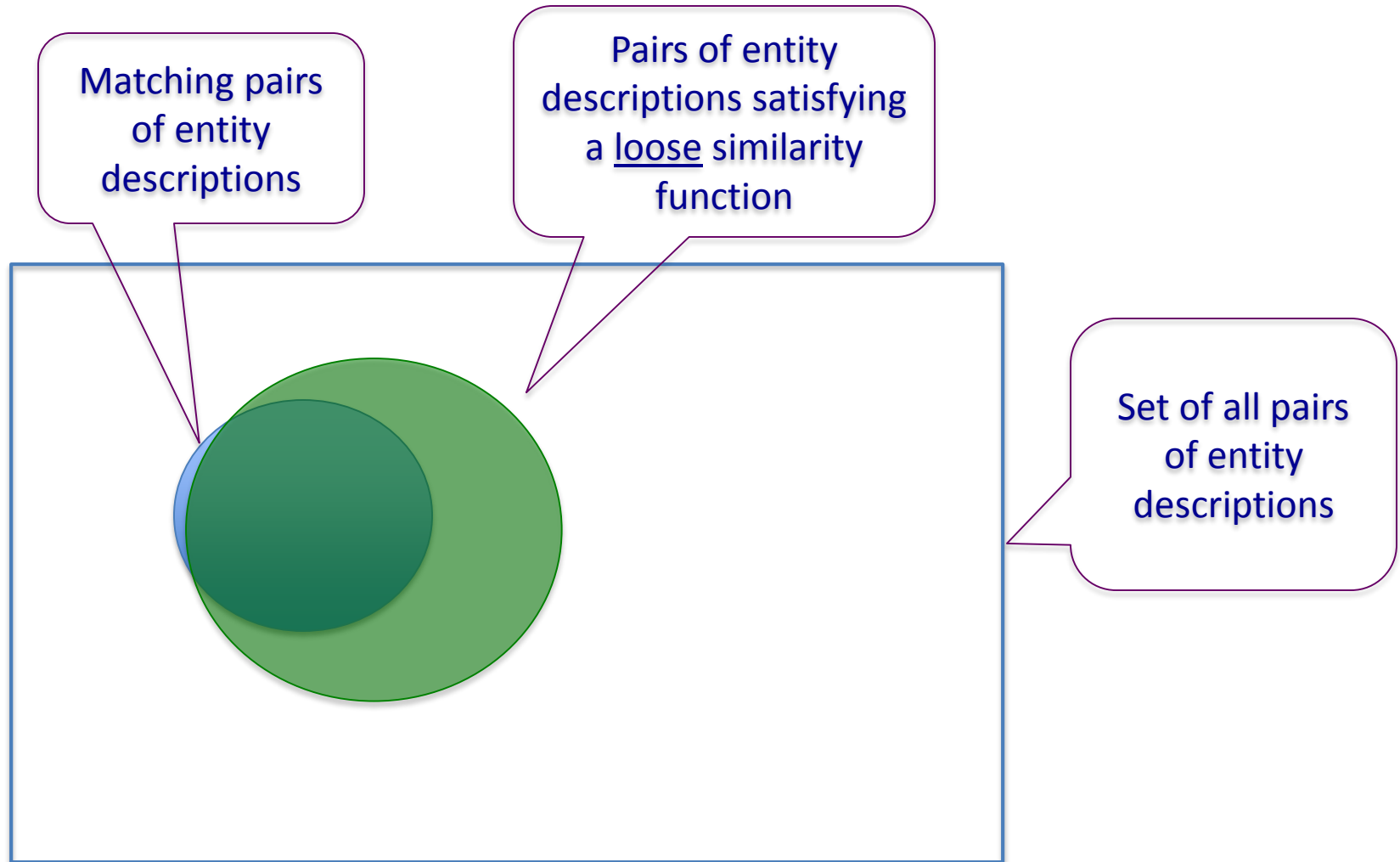
Exact match is rather impractical for entity resolution in the Web of data

- Too strict for a highly heterogeneous information space

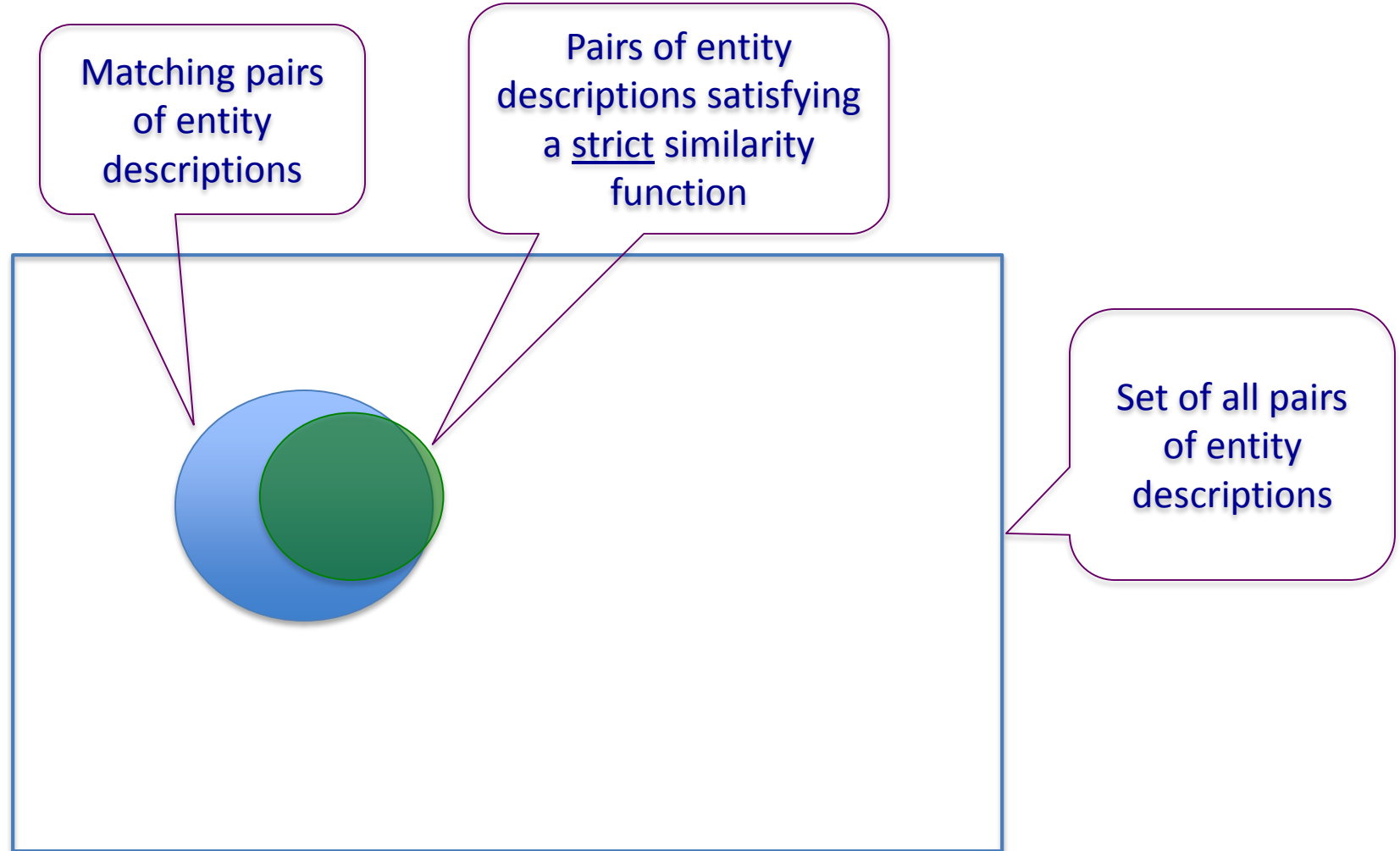
A more loose similarity measure could identify more matches, but...

- Which similarity measure is that?
- What should it compare? Values/Structure/Neighbors?
- It might be too loose and return many false matches too!

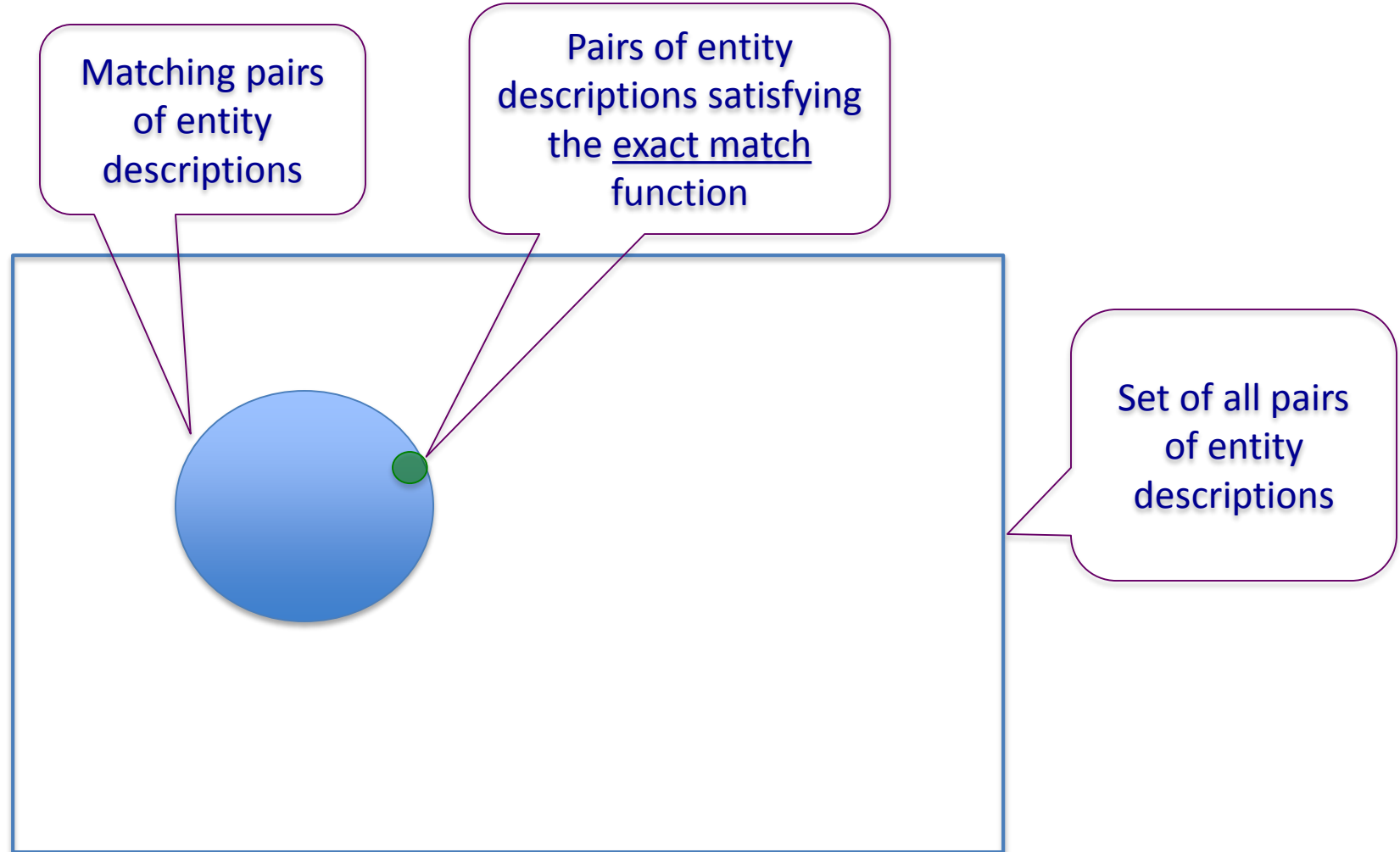
The Role of Similarity Functions – Loose Function



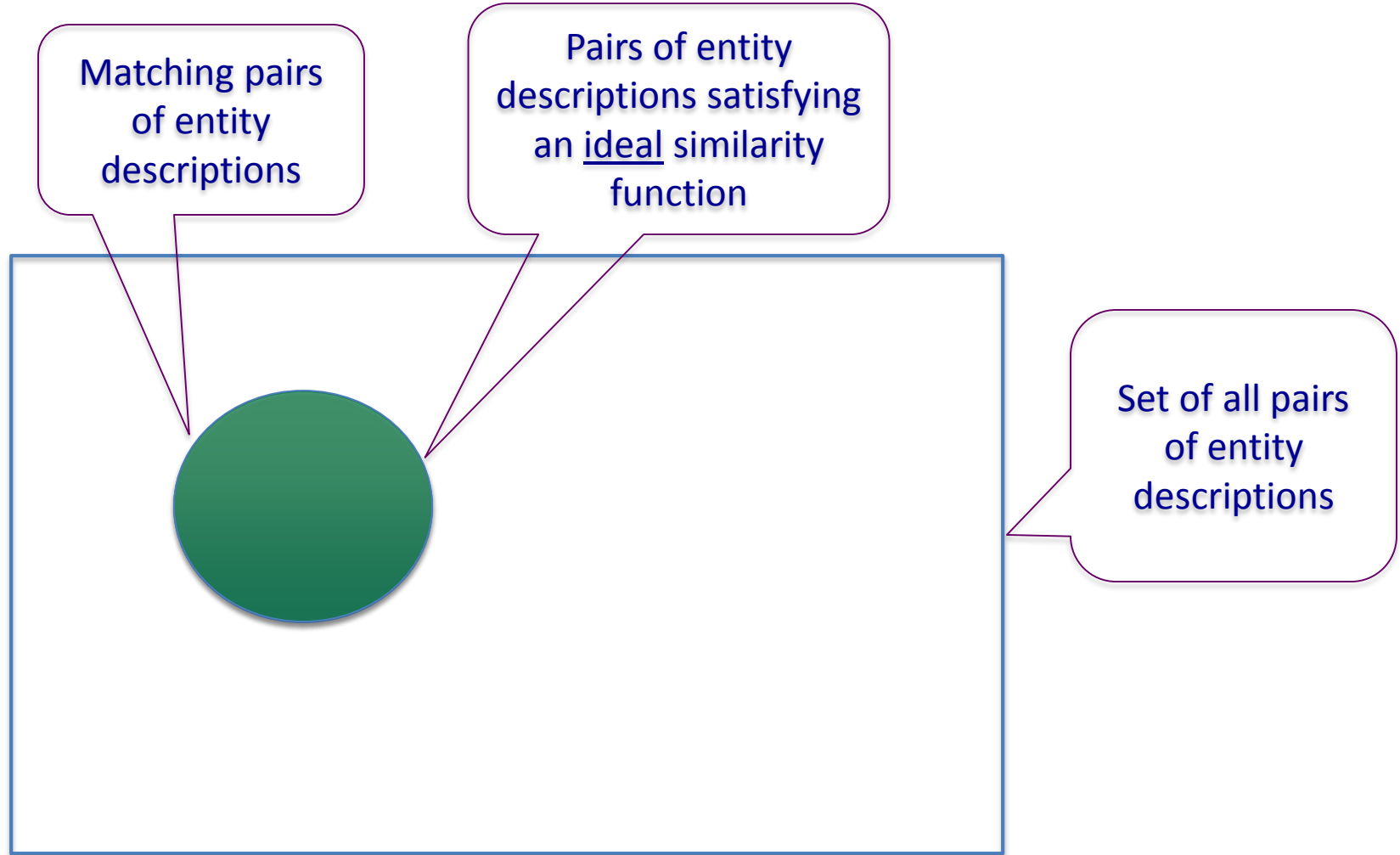
The Role of Similarity Functions – Strict Function



The Role of Similarity Functions – Exact Match



The Role of Similarity Functions – Ideal Case

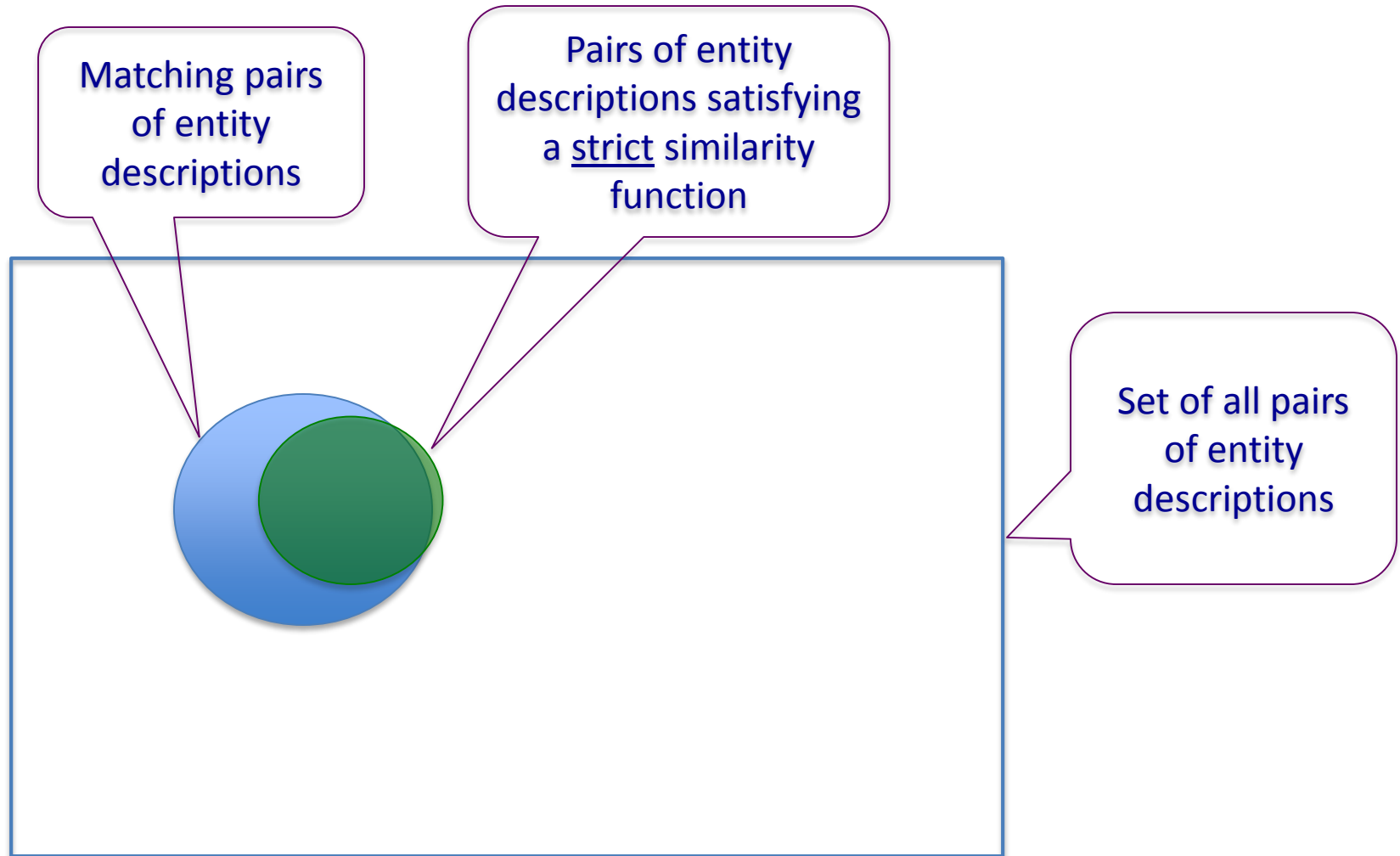


Using Relationships

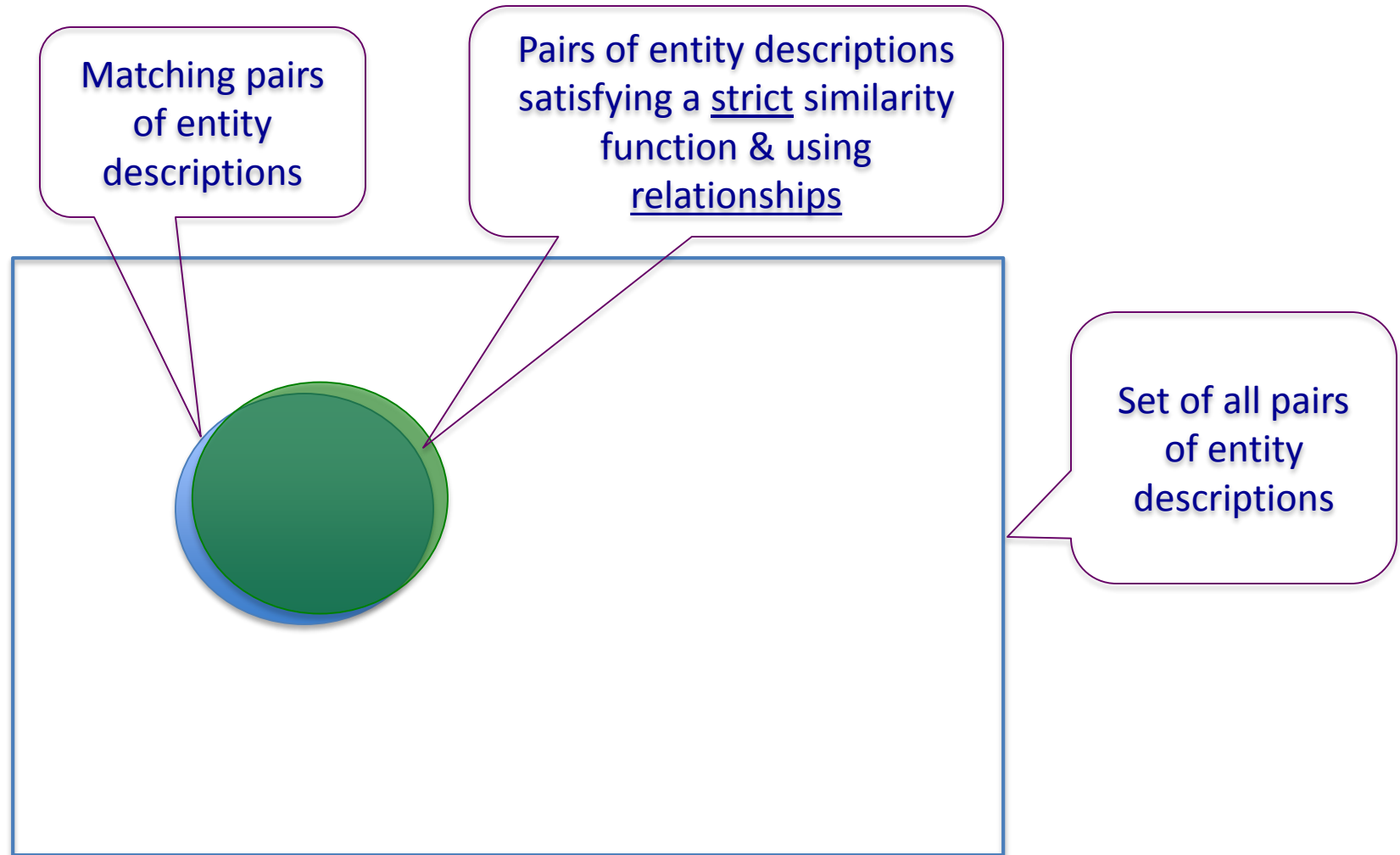
- Transitivity: If (A,B) are matches and (B, C) are matches, then (B,C) are also matches
- Duplicate dependency: If entities Author1 and Author2 are matches, then related entities Publication1 and Publication2 are more likely to be matches than before the matching of Author1 and Author2
- Merge dependency: Once a matching pair has been identified, the merged entity descriptions create a new description that should be compared to the remaining ones

Using these relationships lead to identifying more matches

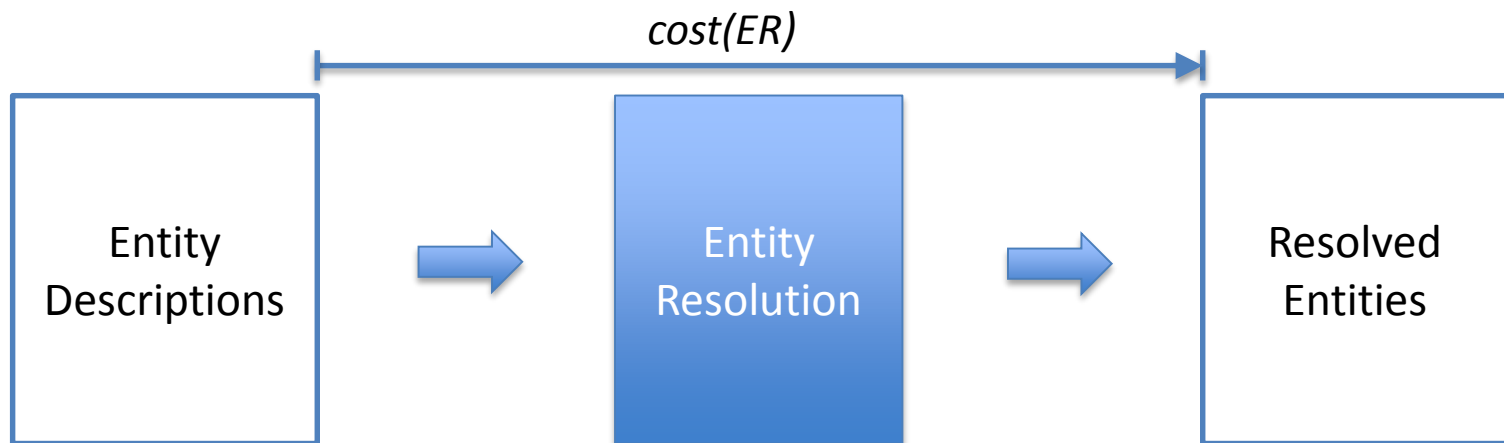
Impact of Using Relationships



Impact of Using Relationships



Entity Resolution Workflow

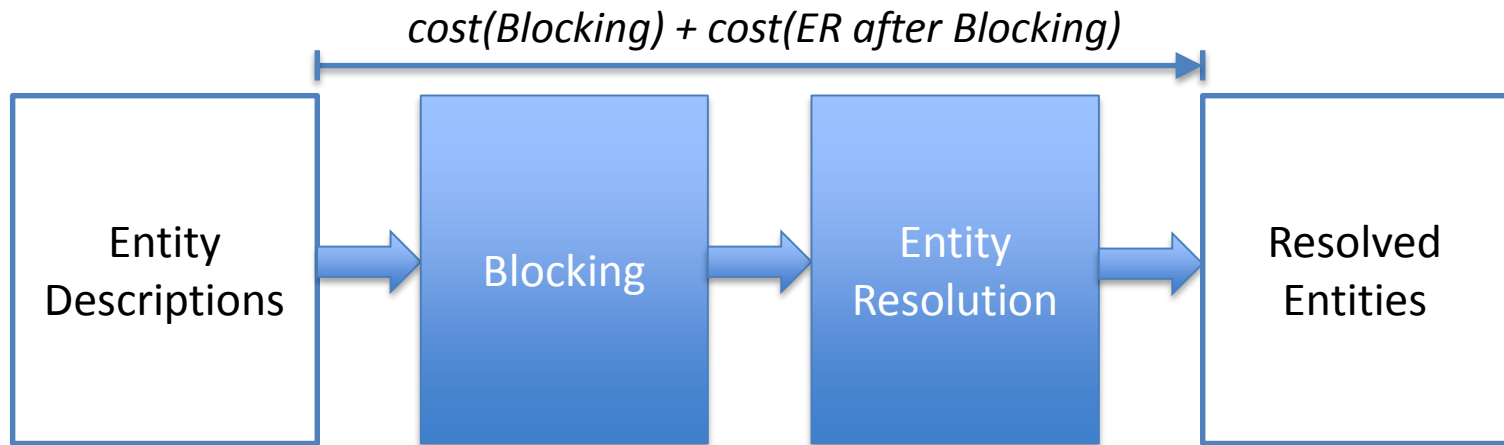


This is a global optimization problem!

Good balance between:

- Number of identified matching descriptions
- Number of generated comparisons

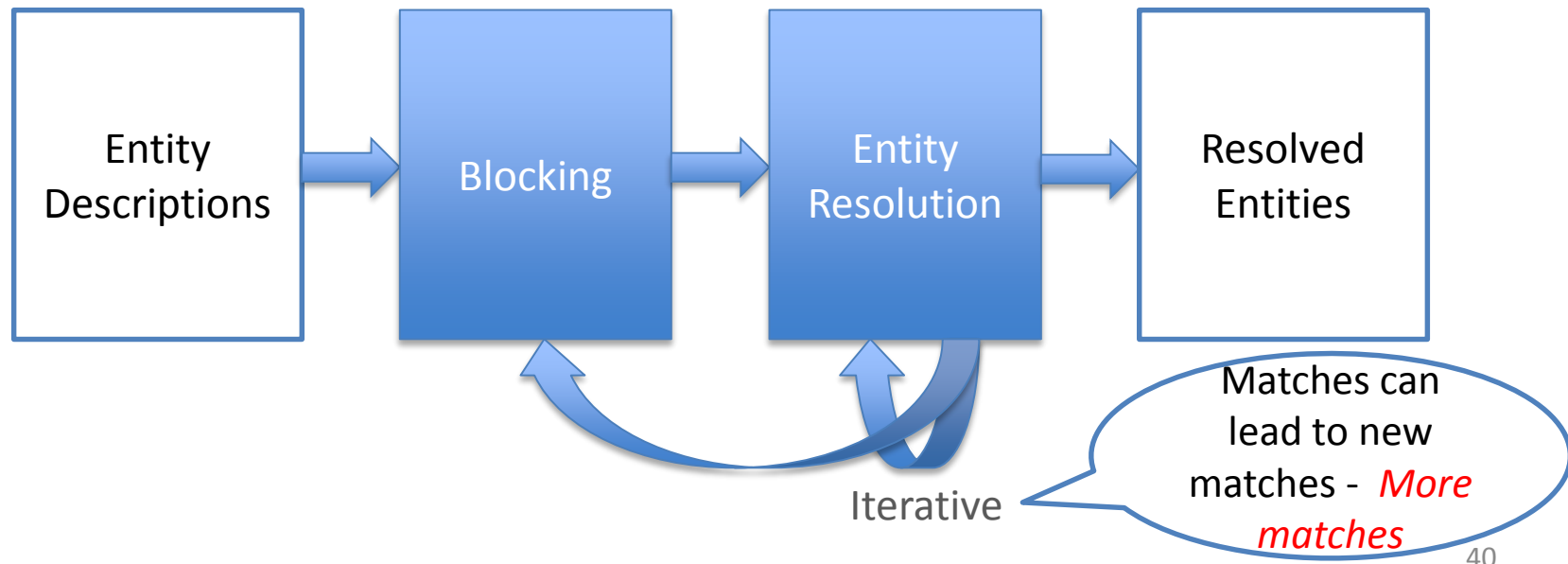
Entity Resolution Workflow



A preprocessing step to group together descriptions close to each other - *Fewer comparisons*

- $cost(ER \text{ after } Blocking) < cost(ER)$
 $benefit(Blocking) = cost(ER) - cost(ER \text{ after } Blocking)$
- $cost(Blocking) + cost(ER \text{ after } Blocking) < cost(ER)$
 $cost(Blocking) < benefit(Blocking) ???$

Entity Resolution Workflow



Blocking Approaches

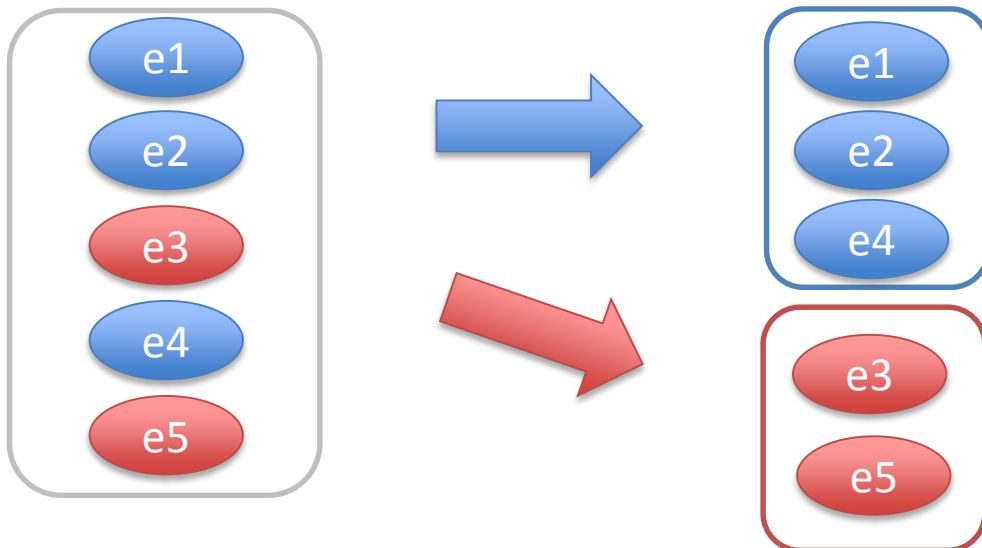
Blocking

To reduce the number of comparisons:

- Split entity descriptions into blocks
- Compare each description to the descriptions within the same block

Desiderata

- Similar entity descriptions in the same block
- Dissimilar entity descriptions in different blocks



Blocking Methodology

Blocking approaches rely on blocking keys

- Criteria on attributes, based on which the descriptions are placed into blocks

Given a blocking key:

The block in which a description will end up is determined by a similarity function on the value of the description for the blocking key

- Blocking key value (BKV)

Using several blocking keys, places each description in many blocks

- Overlapping

Standard Blocking [Fellegi & Sunter 1969]

Entity descriptions with the same BKV end up in the same block

E.g. buildings located at the same place are put in the same block

	Name	Year	Architects	Location
e ₁	Eiffel Tower	1889	Sauvestre	Paris
e ₂	Statue of Liberty	1886	Bartholdi, Eiffel	NY
e ₃	Lady Liberty		Eiffel	NY
e ₄	Eiffel Tower	1889	Sauvestre	Paris
e ₅	White Tower	1450		Thessaloniki

Standard Blocking [Fellegi & Sunter 1969]

Entity descriptions with the same BKV end up in the same block

E.g. buildings located at the same place are put in the same block

	Name	Year	Architects	Location
e ₁	Eiffel Tower	1889	Sauvestre	Paris
e ₂	Statue of Liberty	1886	Bartholdi, Eiffel	NY
e ₃	Lady Liberty		Eiffel	NY
e ₄	Eiffel Tower	1889	Sauvestre	Paris
e ₅	White Tower	1450		Thessaloniki

Generated blocks (partition):

Paris	NY	Thessaloniki
e ₁ , e ₄	e ₂ , e ₃	e ₅

Sorted Neighborhood Method [Hernandez & Stolfo 1995]

The idea

1. Create key
 - Creates a key value based on relevant attribute values
2. Sort
 - Sort tuples in lexicographical order of their generated keys
3. Merge
 - Slide a window (of fixed size w) over the sorted data
 - Limit to comparisons of tuple pairs falling in the same window

Sorted Neighborhood Method

ID	Title	Year	Genre
17	Mask of Zorro	1998	Adventure
18	Addams Family	1991	Comedy
25	Rush Hour	1998	Comedy
31	Matrix	1999	Sci-Fi
52	Return of Dschafar	1994	Children
113	Adams Family	1991	Comedie
207	Return of Djaffar	1995	Children

(1) create key

ID	Key
17	MSKAD98
18	DDMCO91
25	RSHCO98
31	MTRSC99
52	RTRCH94
113	DMSCO91
207	RTRCH95

(2)
sort

ID	Key
18	DDMCO91
113	DMSCO91
17	MSKAD98
31	MTRSC99
25	RSHCO98
52	RTRCH94
207	RTRCH95

(3) merge

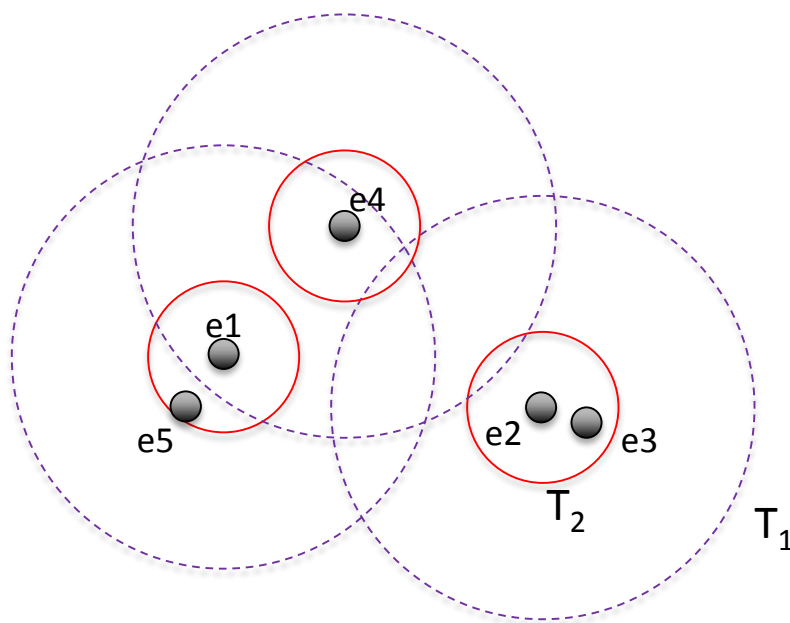
ID	Key
18	DDMCO91
113	DMSCO91
17	MSKAD98
31	MTRSC99
25	RSHCO98
52	RTRCH94
207	RTRCH95

compare(18,113) → duplicates

compare(52,207) → duplicates

Canopy Clustering [McCallum et al. 2000]

1. Pick a random entity description e_i from E
2. Create, for e_i , a new canopy C_{e_i}
Add to C_{e_i} the descriptions e_j , s.t. $d(e_i, e_j) < T_1$
3. Remove all descriptions e_j from E , s.t. $d(e_i, e_j) < T_2$
4. Return to Step 1, if E is not empty



Generated Blocks:

e1	e4	e2
e_1, e_4, e_5	e_1, e_4	e_2, e_3

What is the intuition behind thresholds T_1, T_2 ?

Token Blocking [Papadakis et al. 2011]

Assume two clean sets E_1, E_2 of entity descriptions – *Clean-Clean Entity Resolution*

- Each distinct token t_i of each value of each description in $E_1 \cup E_2$ corresponds to a block
 - Each block contains all entity descriptions with the corresponding token
 - Pairs originating from the same (clean) set are not compared

Redundancy!

- The same pair of descriptions is contained in many blocks
- Many dissimilar pairs are put in the same block

Token Blocking - Example

 E_1

name	Eiffel Tower
architect	Sauvestre
year	1889
location	Paris

name	Statue of Liberty
architect	Bartholdi Eiffel
year	1886
located	NY

about	Lady liberty
architect	Eiffel
location	NY

 E_2

about	Eiffel Tower
architect	Sauvestre
year	1889
located	Paris

name	White Tower
location	Thessaloniki
year-constructed	1450

Generated Blocks

Eiffel	Tower	Statue	Liberty	White	1889	Bartholdi
e_1, e_2, e_3, e_4	e_1, e_4, e_5	e_2	e_2, e_3	e_5	e_1, e_4	e_2
NY	Paris	1886	1450	Lady	Sauvestre	Thessaloniki
e_2, e_3	e_1, e_4	e_2	e_5	e_3	e_1, e_4	e_5

Token Blocking - Example

name	Eiffel Tower
architect	Sauvestre
year	1889
location	Paris

e1

name	Statue of Liberty
architect	Bartholdi Eiffel
year	1886
located	NY

e2

about	Lady liberty
architect	Eiffel
location	NY

e3

about	Eiffel Tower
architect	Sauvestre
year	1889
located	Paris

e4

name	White Tower
location	Thessaloniki
year-constructed	1450

e5

Generated Blocks

Eiffel	Tower	Statue	Liberty	White	1889	Bartholdi
e ₁ , e ₂ , e ₃ , e ₄	e ₁ , e ₄ , e ₅	e₂	e ₂ , e ₃	e₅	e ₁ , e ₄	e₂
NY	Paris	1886	1450	Lady	Sauvestre	Thessaloniki
e ₂ , e ₃	e ₁ , e ₄	e₂	e₃	e₃	e ₁ , e ₄	e₅

Blocks containing descriptions from only one collection are discarded

Token Blocking - Example

name	Eiffel Tower
architect	Sauvestre
year	1889
location	Paris

e1

name	Statue of Liberty
architect	Bartholdi Eiffel
year	1886
located	NY

e2

about	Lady liberty
architect	Eiffel
location	NY

e3

about	Eiffel Tower
architect	Sauvestre
year	1889
located	Paris

e4

name	White Tower
location	Thessaloniki
year-constructed	1450

e5

Generated Blocks

Eiffel	Tower
e ₁ , e ₂ , e ₃ , e ₄	e ₁ , e ₄ , e ₅
NY	Paris
e ₂ , e ₃	e ₁ , e ₄

Liberty
e ₂ , e ₃

1889
e ₁ , e ₄

Sauvestre
e ₁ , e ₄

The pair (e₁, e₄) is contained in 5 different blocks!

Token Blocking - Example

name	Eiffel Tower
architect	Sauvestre
year	1889
location	Paris

e1

name	Statue of Liberty
architect	Bartholdi Eiffel
year	1886
located	NY

e2

about	Lady liberty
architect	Eiffel
location	NY

e3

about	Eiffel Tower
architect	Sauvestre
year	1889
located	Paris

e4

name	White Tower
location	Thessaloniki
year-constructed	1450

e5

Generated Blocks

Eiffel	Tower
e ₁ , e ₂ , e ₃ , e ₄	e ₁ , e ₄ , e ₅
NY	Paris
e ₂ , e ₃	e ₁ , e ₄

Liberty
e ₂ , e ₃

1889
e ₁ , e ₄

Sauvestre
e ₁ , e ₄

Redundant comparisons are performed between (e₁, e₃), (e₂, e₄), (e₁, e₅)

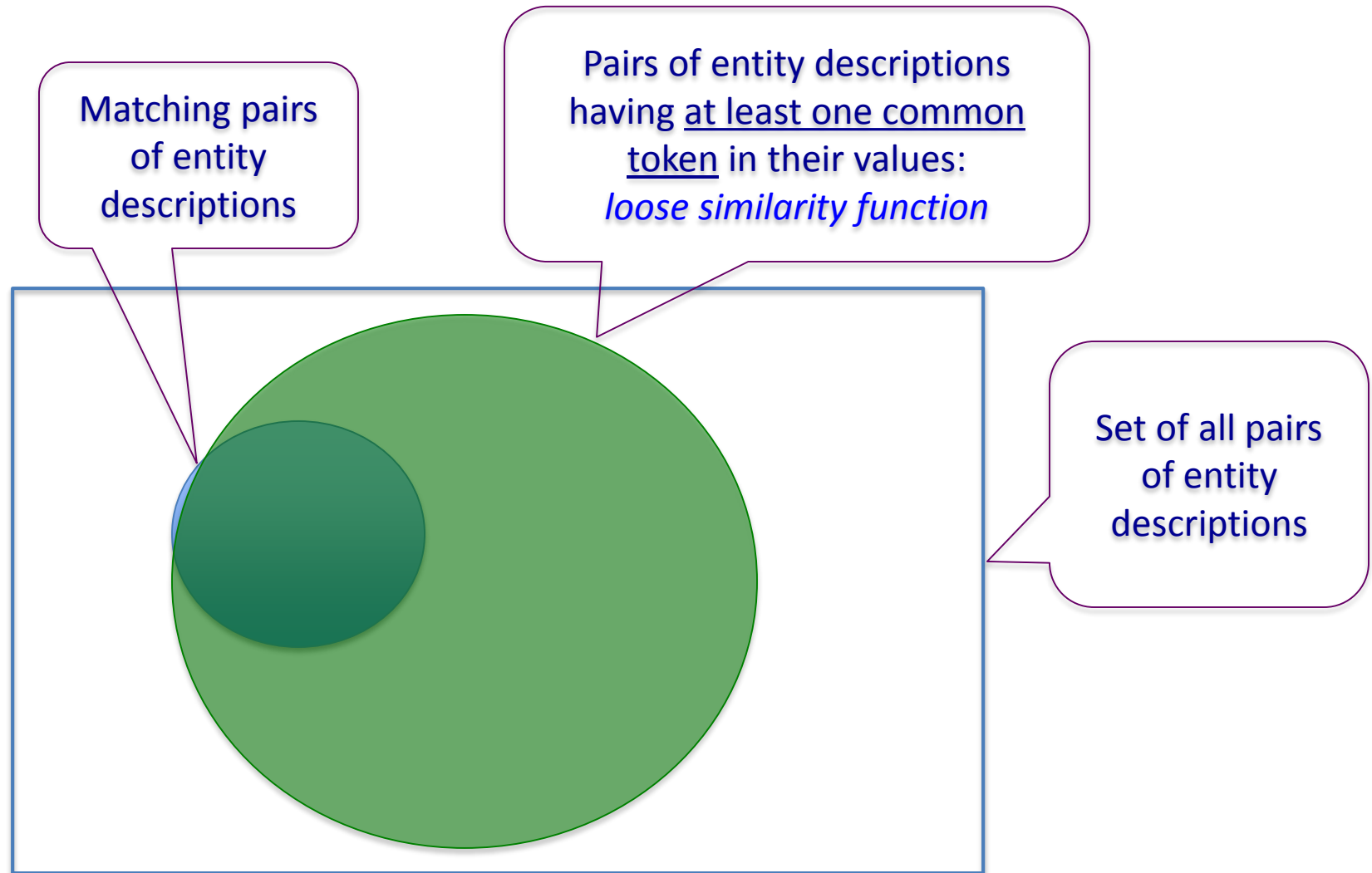
Token blocking achieves:

High recall at the cost of low precision and low efficiency:

- Most true matches are placed in the same block
- Many non-matches are also placed in the same block
- The same pair of descriptions is contained in many blocks

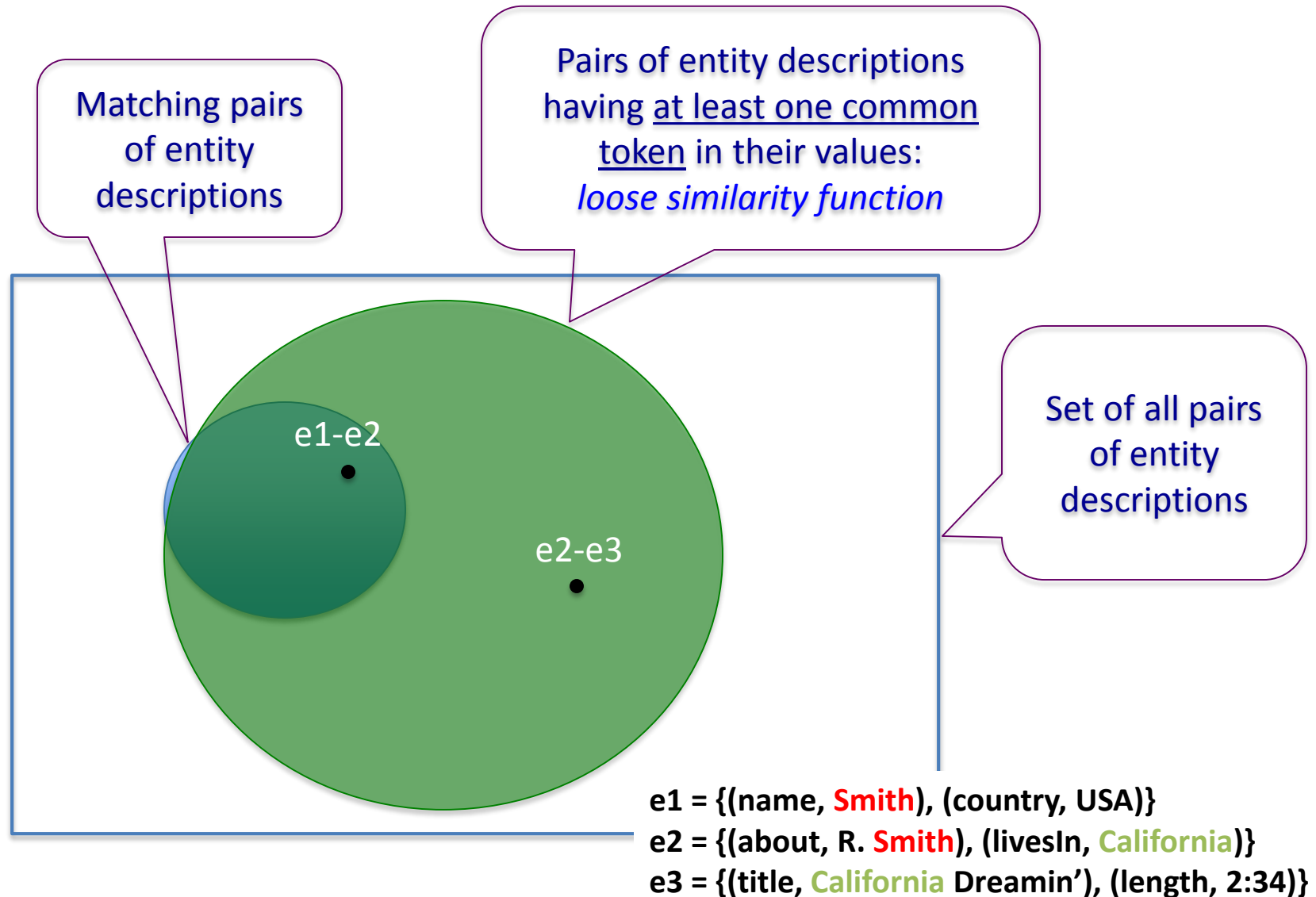
Token blocking totally ignores the valuable information of attribute names

Token Blocking - Evaluation



A single common token in the set of values is enough to place two descriptions in the same block

Token Blocking - Evaluation



Is this enough?

Token blocking totally ignores the valuable information of attribute names

To improve this, attribute clustering considers patterns in the values

[Papadakis et al. 2013 (a)]

Attribute Clustering Blocking [Papadakis et al. 2013 (a)]

The goal again is to identify matches between two datasets, D_1 and D_2 , each containing no duplicates – Clean-Clean Entity Resolution

Two main steps:

1. Similar attributes are placed together in non-overlapping clusters
2. Token blocking is performed on the descriptions of each cluster

Creating Clusters of Attributes

1. For each attribute of dataset D_1 :
 - Find the most similar attribute of dataset D_2
2. For each attribute of dataset D_2 :
 - Find the most similar attribute of dataset D_1
3. Compute the transitive closure of the generated pairs of attributes
4. Connected attributes form clusters
5. All single-member clusters are merged into a common cluster

Similarities between attributes are computed wrt. the string similarities of the values appearing in these attributes

Creating Clusters of Attributes

about	Eiffel Tower
architect	Sauvestre
year	1889
located	Paris

e11

about	Statue of Liberty
architect	Bartholdi Eiffel
year	1886
located	NY

e12

about	Auguste Bartholdi
born	1834

e13

about	Joan Tower
born	1938

e14

work	Lady Liberty
artist	Bartholdi
location	NY

e15

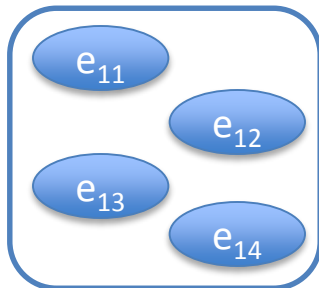
work	Eiffel Tower
year-constructed	1889
location	Paris

e16

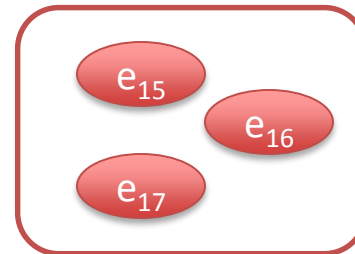
work	Bartholdi Fountain
year-constructed	1876
location	Washington D.C.

e17

D1



D2



Clustering Attributes: Example

about	Eiffel Tower
architect	Sauvestre
year	1889
located	Paris

e11

about	Statue of Liberty
architect	Bartholdi Eiffel
year	1886
located	NY

e12

about	Auguste Bartholdi
born	1834

e13

about	Joan Tower
born	1938

e14

work	Lady Liberty
artist	Bartholdi
location	NY

e15

work	Eiffel Tower
year-constructed	1889
location	Paris

e16

work	Bartholdi Fountain
year-constructed	1876
location	Washington D.C.

e17

Finding the attribute of **D2** that is the most similar to the attribute “about” of **D1**:

values of about: {Eiffel, Tower, Statue, Liberty, Auguste, Bartholdi, Joan}

compared to (with Jaccard similarity) :

values of **work**: {Lady, Liberty, Eiffel, Tower, Bartholdi, Fountain} → **Jaccard = 4/9**

values of artist: {Bartholdi} → Jaccard = 1/8

values of location: {NY, Paris, Washington, D.C.} → Jaccard = 0

values of year-constructed: {1889, 1876} → Jaccard = 0

Clustering Attributes: Example

about	Eiffel Tower
architect	Sauvestre
year	1889
located	Paris

e11

about	Statue of Liberty
architect	Bartholdi Eiffel
year	1886
located	NY

e12

about	Auguste Bartholdi
born	1834

e13

about	Joan Tower
born	1938

e14

work	Lady Liberty
artist	Bartholdi
location	NY

e15

work	Eiffel Tower
year-constructed	1889
location	Paris

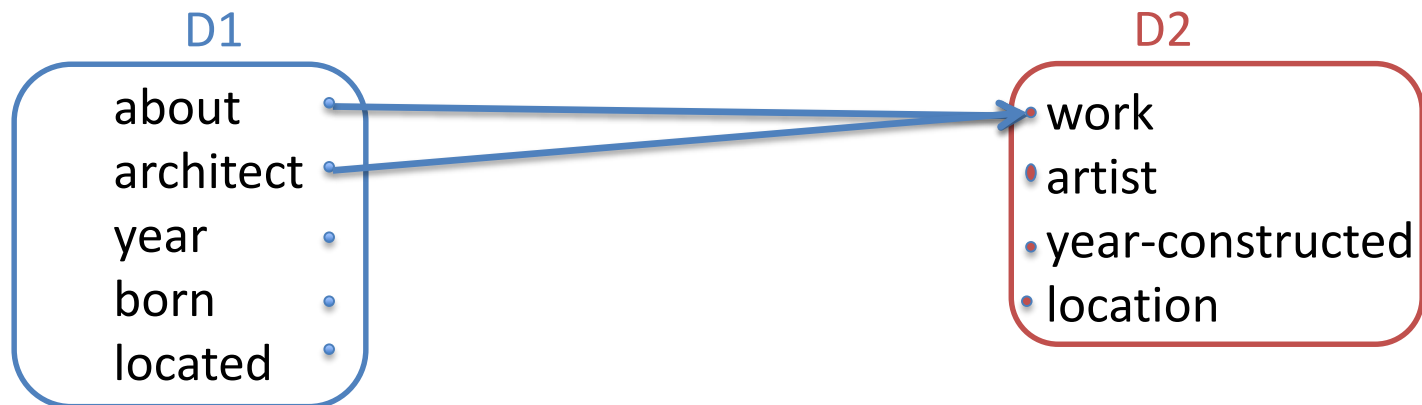
e16

work	Bartholdi Fountain
year-constructed	1876
location	Washington D.C.

e17

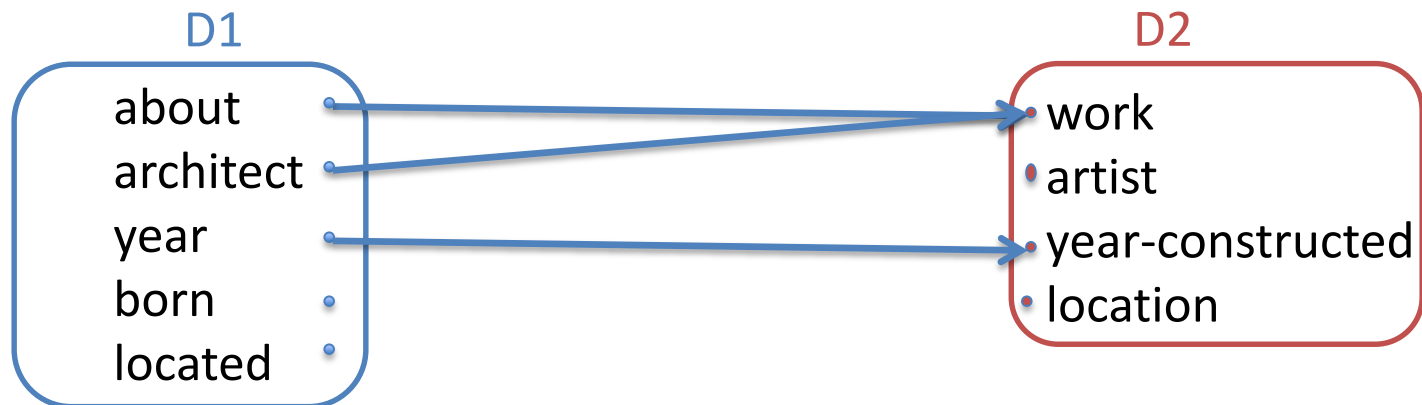
Clustering Attributes: Example

Similarly for the rest of the attributes...



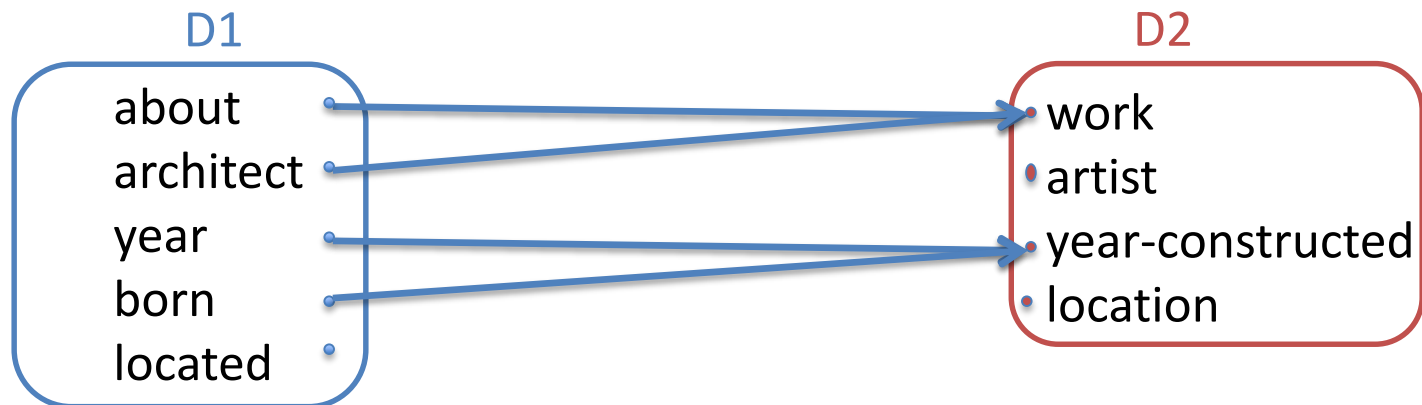
Clustering Attributes: Example

Similarly for the rest of the attributes...



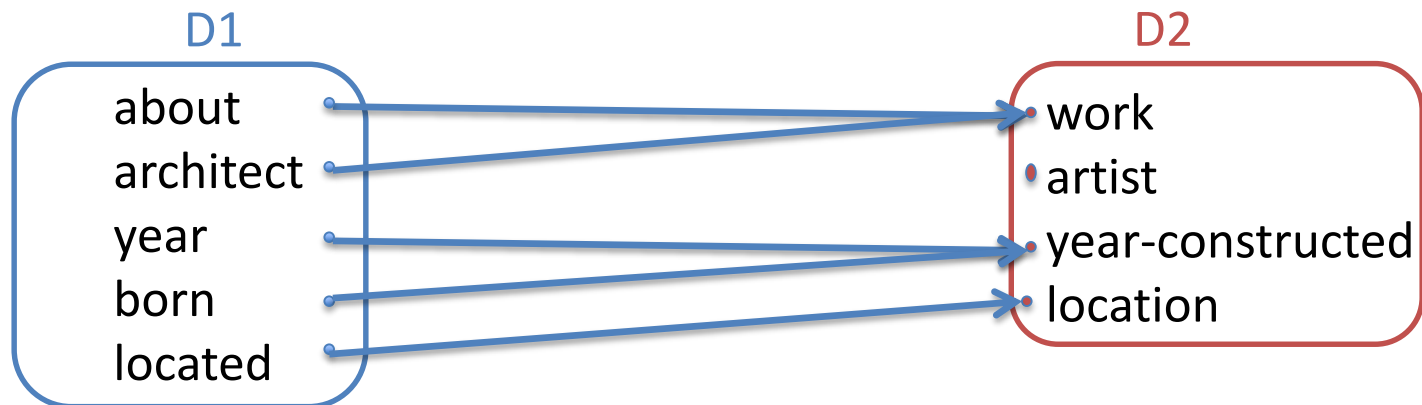
Clustering Attributes: Example

Similarly for the rest of the attributes...



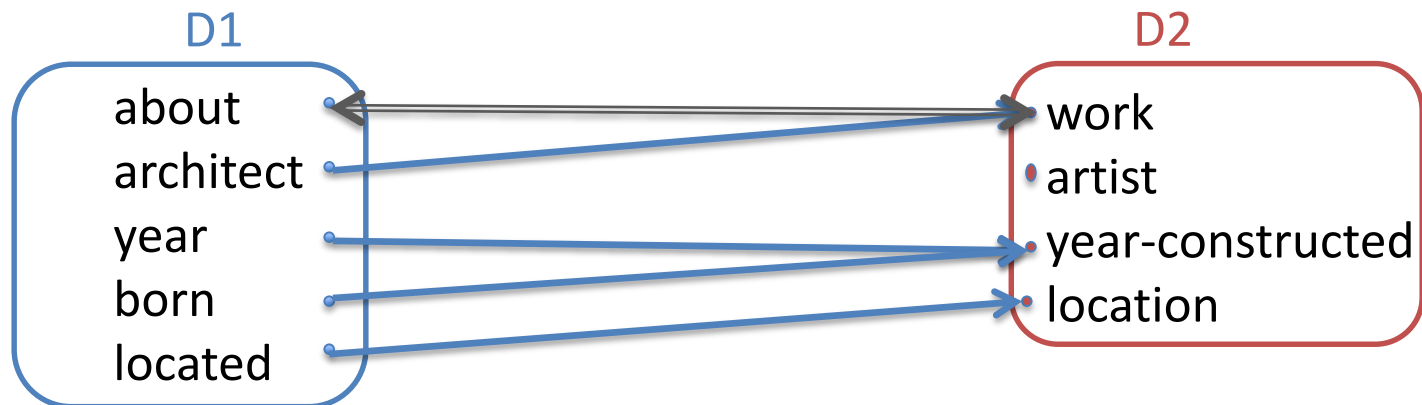
Clustering Attributes: Example

Similarly for the rest of the attributes...



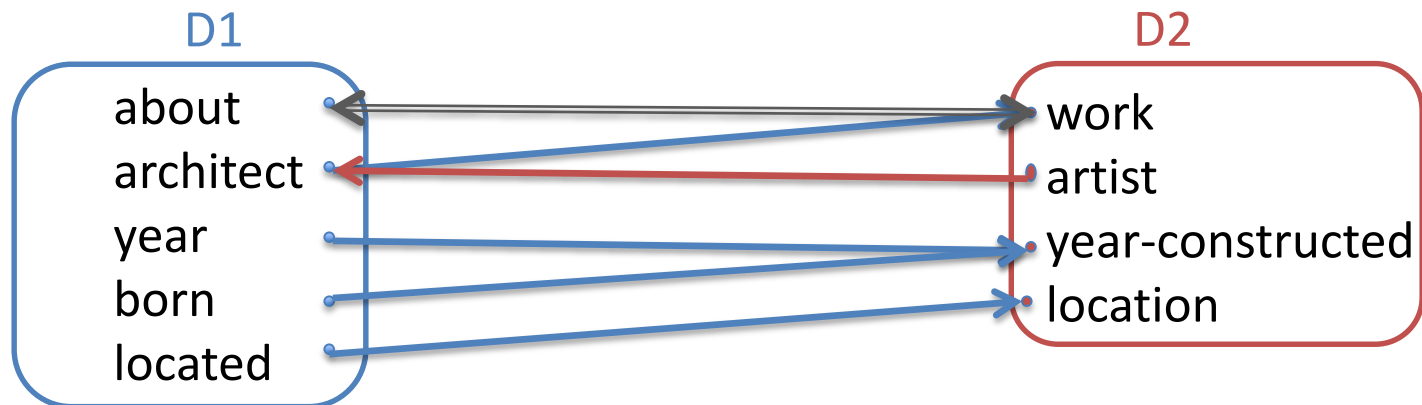
Clustering Attributes: Example

Similarly for the rest of the attributes...



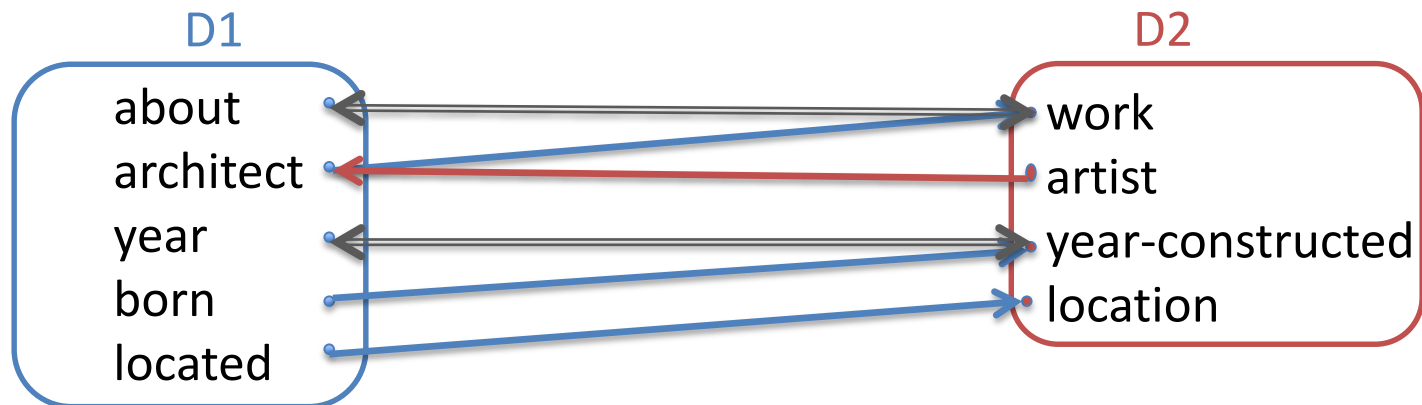
Clustering Attributes: Example

Similarly for the rest of the attributes...



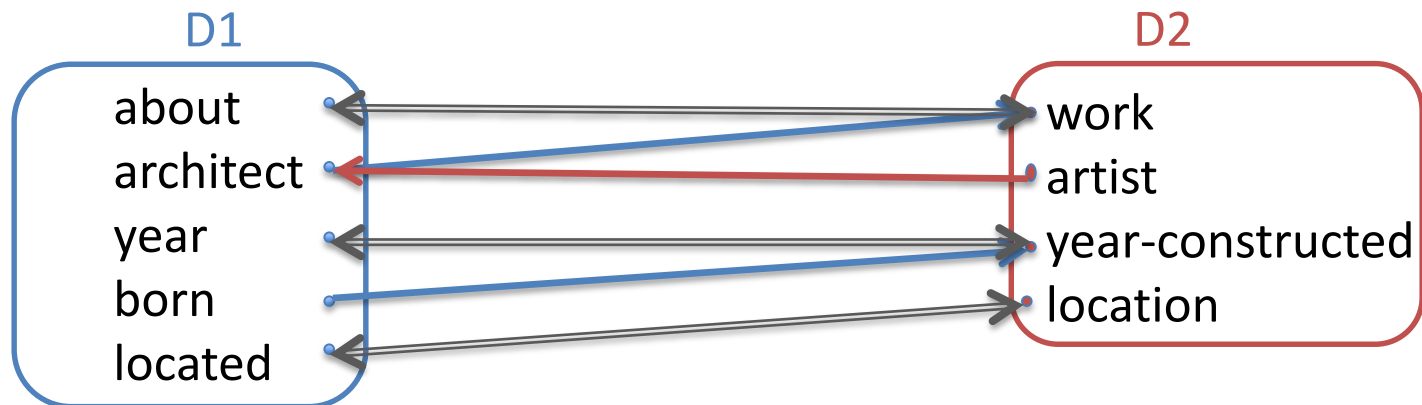
Clustering Attributes: Example

Similarly for the rest of the attributes...



Clustering Attributes: Example

Similarly for the rest of the attributes...



Clustering Attributes: Example

about	Eiffel Tower
architect	Sauvestre
year	1889
located	Paris

e11

about	Statue of Liberty
architect	Bartholdi Eiffel
year	1886
located	NY

e12

about	Auguste Bartholdi
born	1834

e13

about	Joan Tower
born	1938

e14

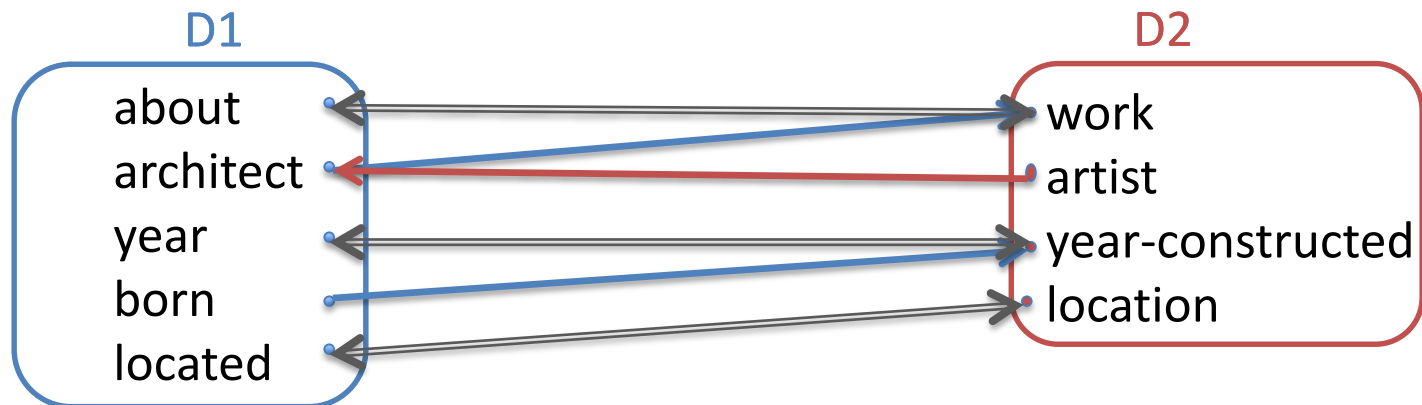
work	Lady Liberty
artist	Bartholdi
location	NY

e15

work	Eiffel Tower
year-constructed	1889
location	Paris

e16

work	Bartholdi Fountain
year-constructed	1876
location	Washington D.C.

e17

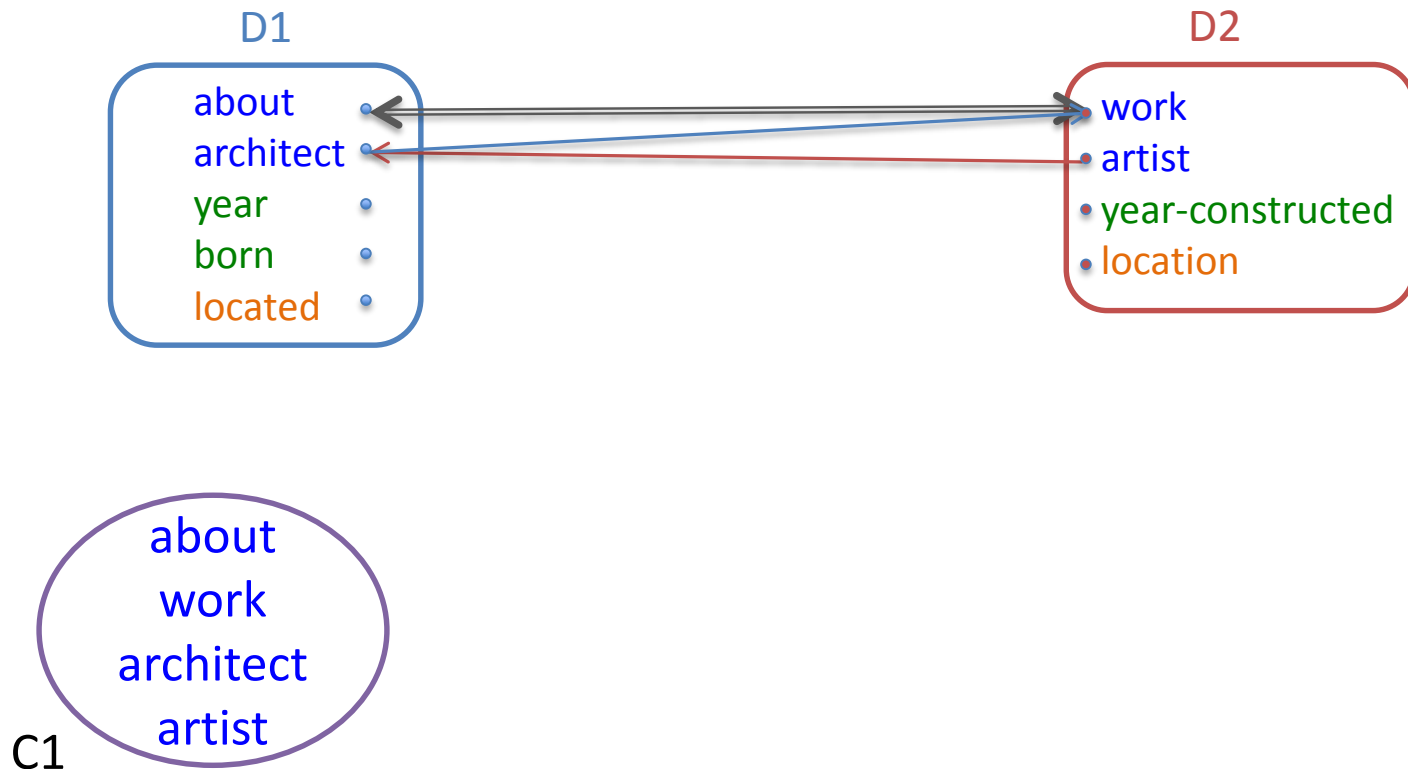
Clustering Attributes: Example

Compute the transitive closure of the generated attribute pairs

- Connected attributes form clusters

Pairs: (about, work), (work, about), (artist, architect), (architect, work)

Transitive closure:



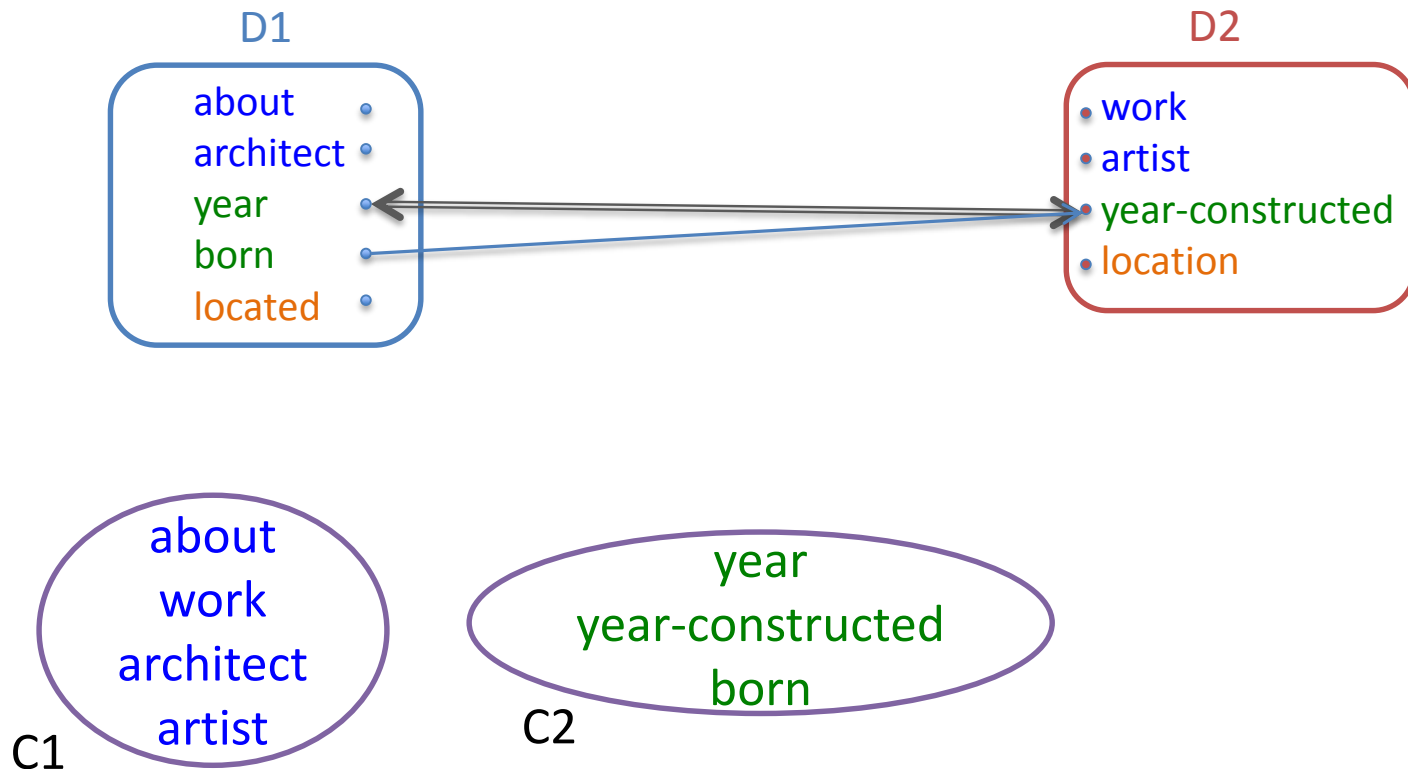
Clustering Attributes: Example

Compute the transitive closure of the generated attribute pairs

- Connected attributes form clusters

Pairs: (year, year-constructed), (year-constructed, year), (year-constructed, born)

Transitive closure:



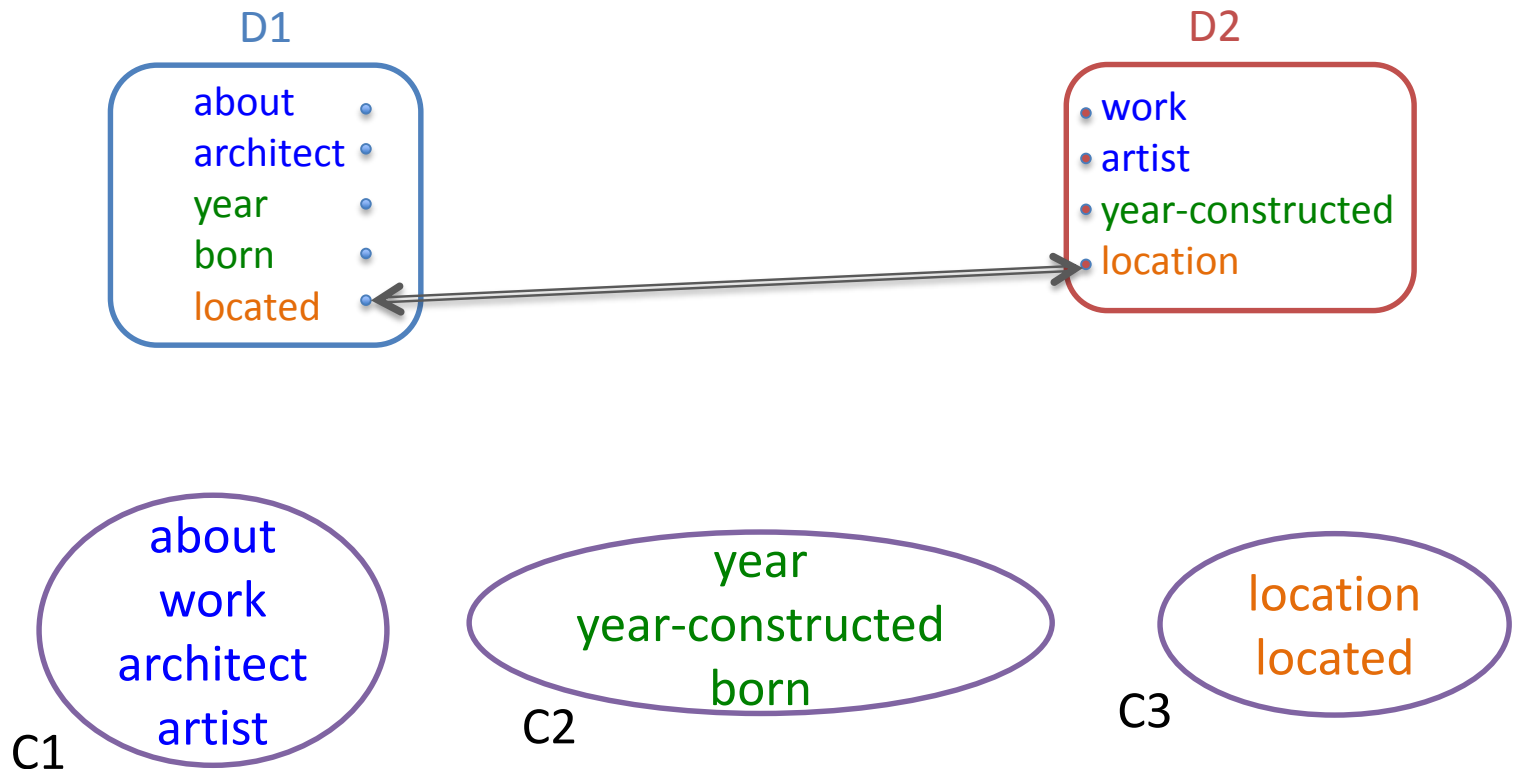
Clustering Attributes: Example

Compute the transitive closure of the generated attribute pairs

- Connected attributes form clusters

Pairs: (located, location), (location, located)

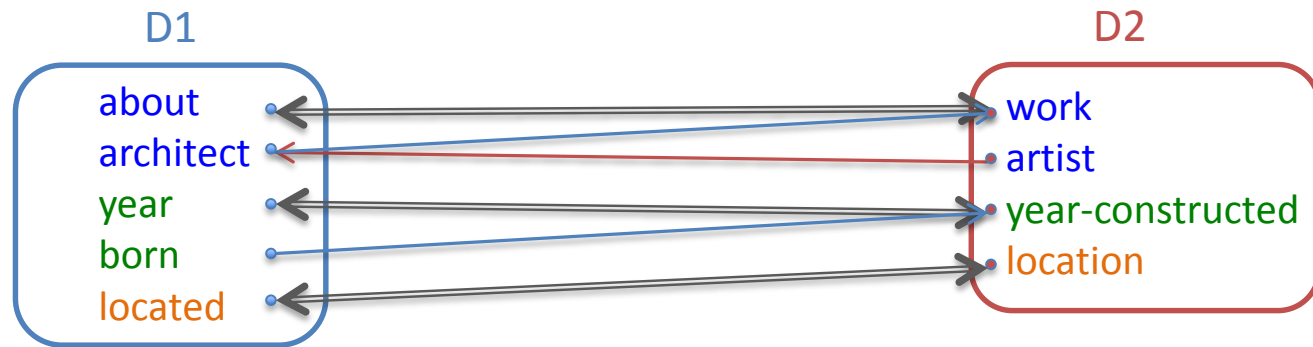
Transitive closure:



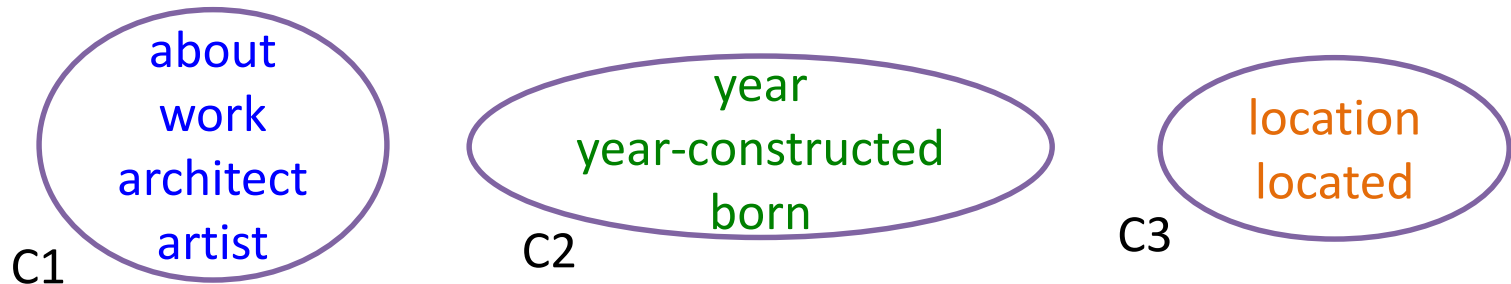
Clustering Attributes: Example

Compute the transitive closure of the generated attribute pairs

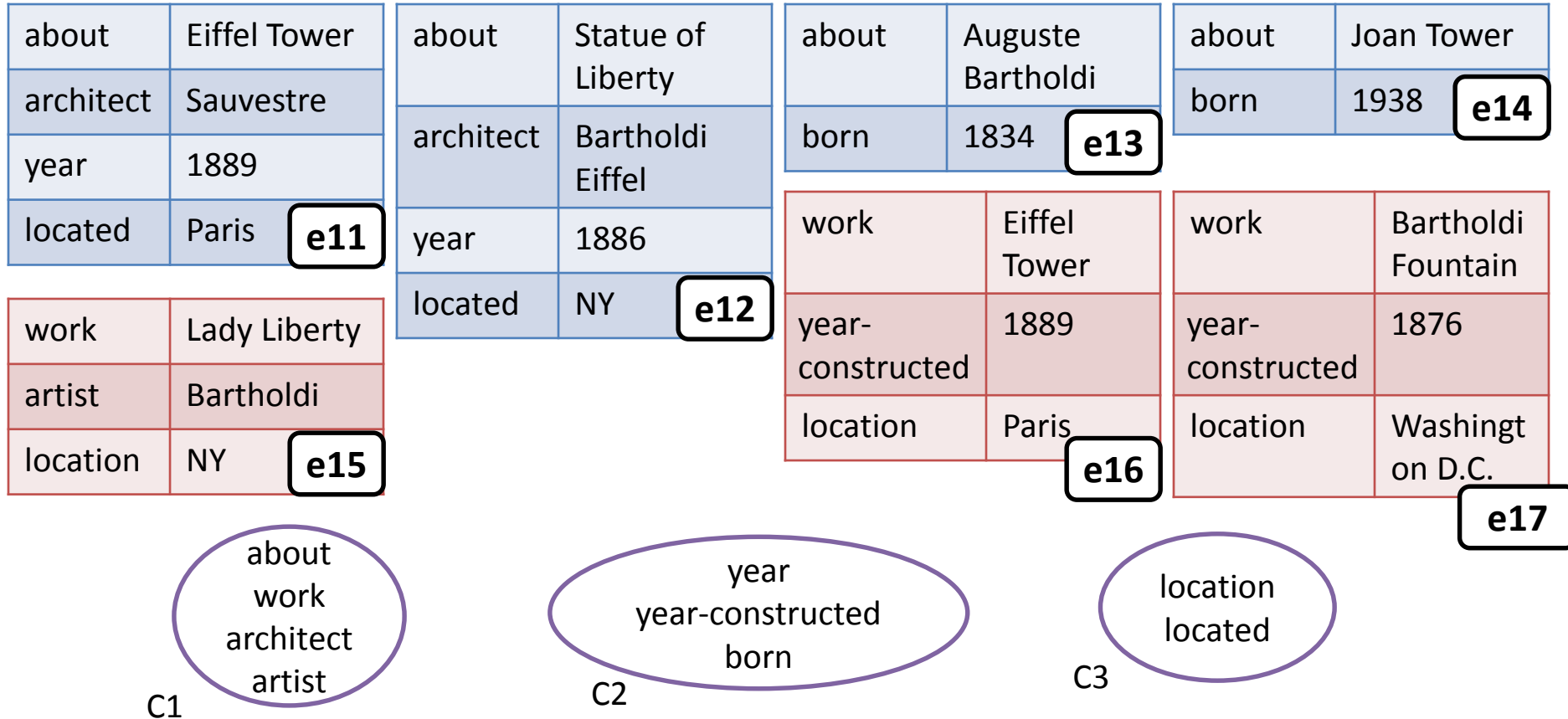
- Connected attributes form clusters



Generated attribute clusters:



Token Blocking for Each Cluster

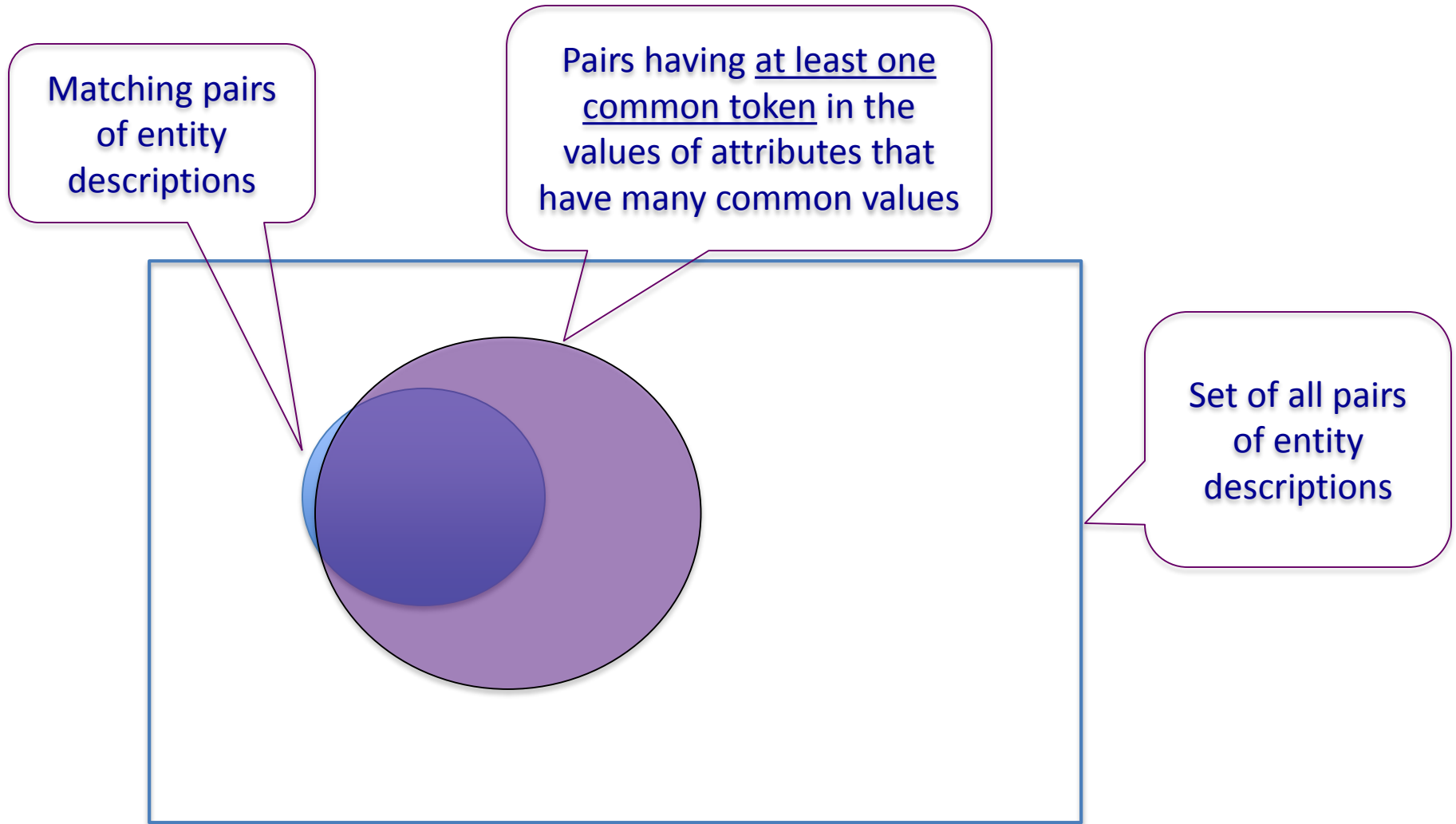


Some of the generated blocks:

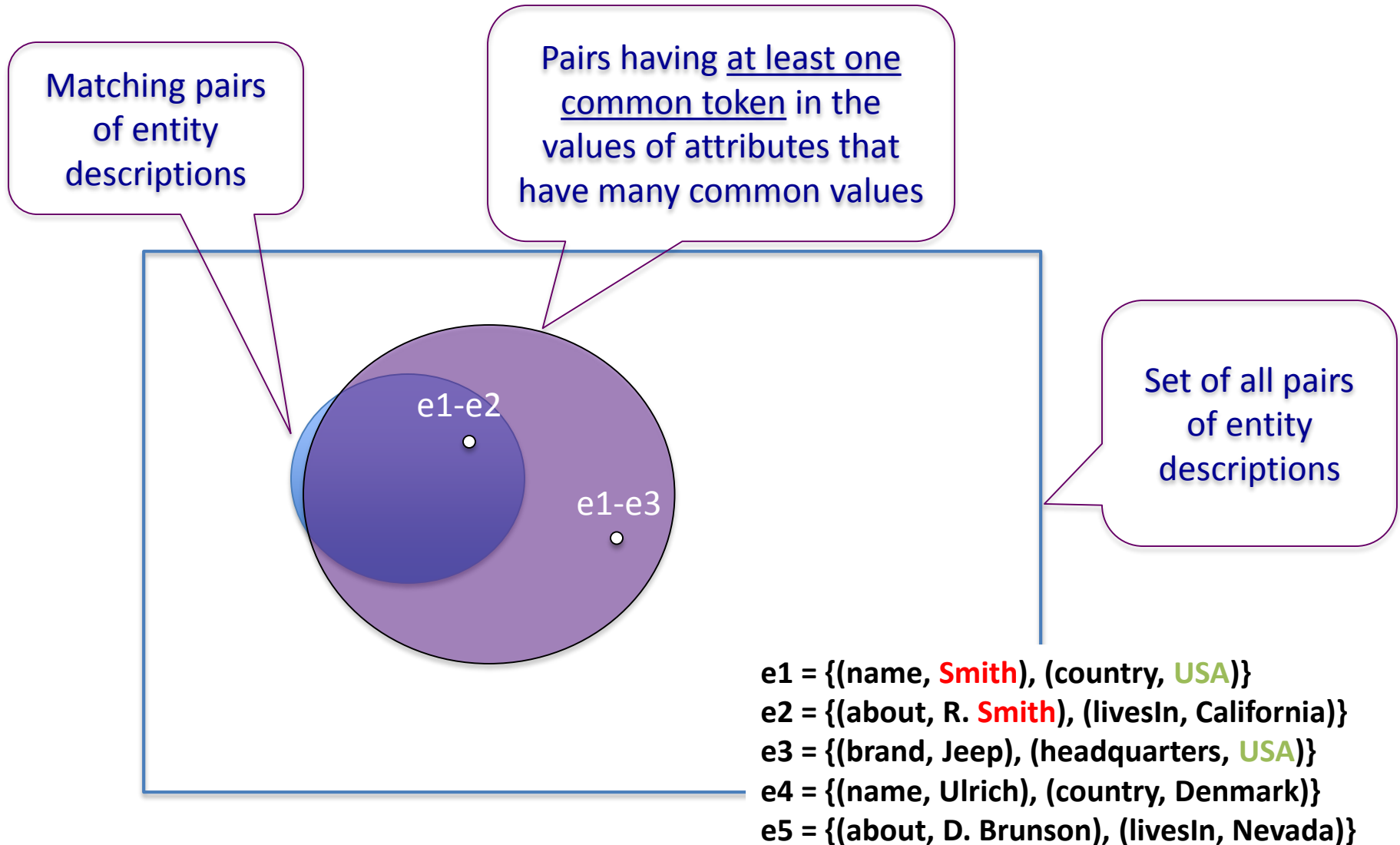
C3.NY	C1.Tower	C1.Bartholdi
e ₁₂ , e ₁₅	e ₁₁ , e ₁₄ , e ₁₆	e ₁₂ , e ₁₃ , e ₁₅ , e ₁₇

→ compare Lady Liberty to Auguste Bartholdi

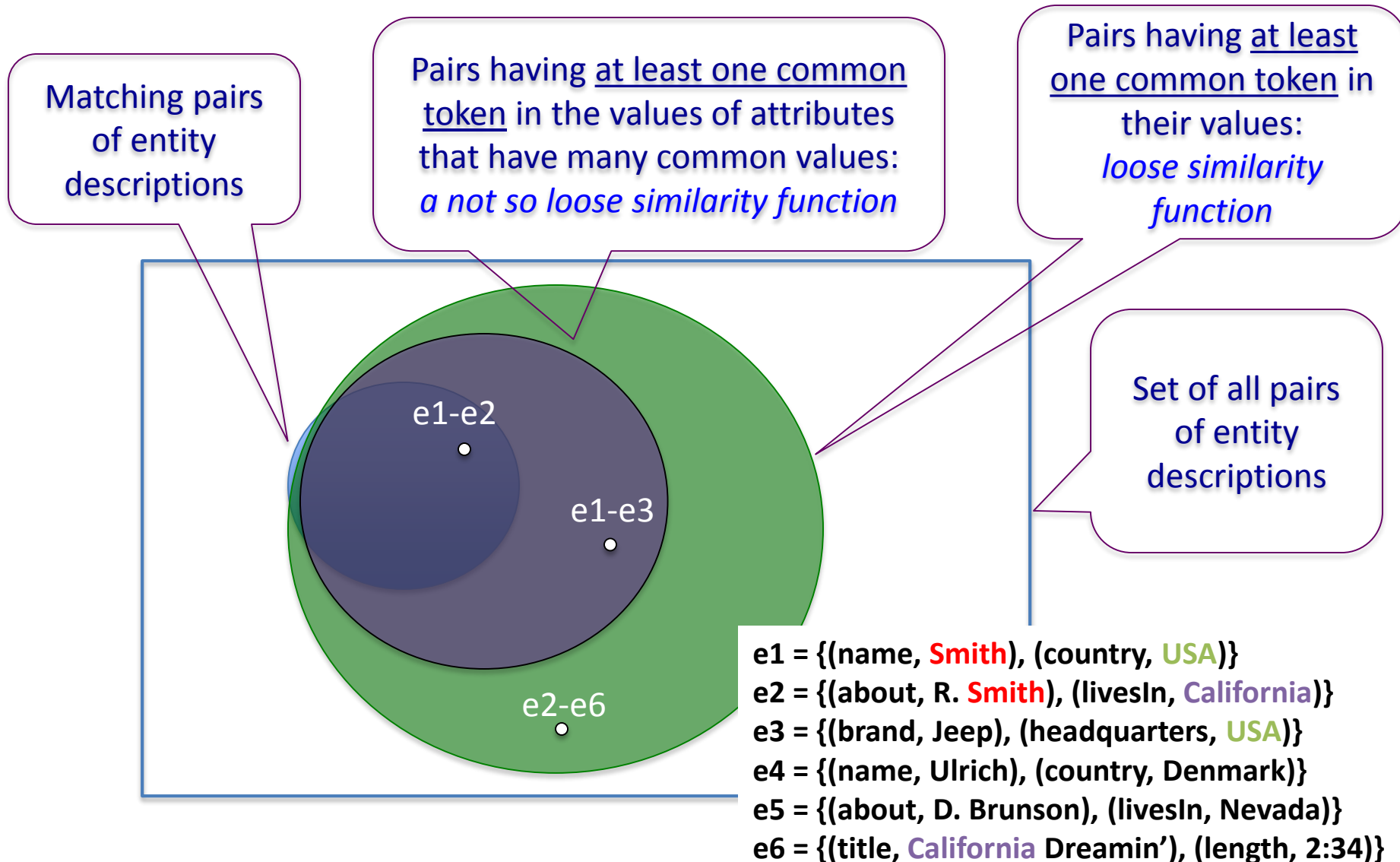
Attribute Clustering Blocking- Evaluation



Attribute Clustering Blocking- Evaluation



Attribute Clustering Blocking vs Token Blocking



Attribute Clustering Blocking vs Token Blocking

In attribute clustering:

- High recall
- Better efficiency compared to token blocking (save many redundant comparisons)
- Low precision

Many non-matches are placed in the same block

The same pair of descriptions is contained in many blocks

Much more expensive to build the blocks, than just performing token blocking

Again, it ignores the valuable semantics that attributes and entity relationships offer

ZenCrowd [Demartini et al. 2013]

A different approach to attribute clustering

Three-stage blocking:

1. Token blocking on the labels of the descriptions
2. Rank description pairs within blocks, based on the Jaccard similarity of the values of matching attribute pairs
 - Attribute matching is based on the number of exact string matches that two attributes have in their values (within block)

3. Ask humans for the low-ranked pairs (crowdsourcing)

Find this Target Entity:
Spoletto (Italy)

☐ Ariulf of Spoleto

☐ Spoletto Festival, Italy

☐ Spoletto

☐ Spoletto Festival (taped in Italy): Sir John Gielgud; Eileen Farrell

☐ Winiges of Spoleto

☐ No Result is the same as the Target Entity

ZenCrowd - Example

name	Statue of Liberty
architect	Bartholdi Eiffel
year	1886
located	NY

e1

about	Lady liberty
architect	Eiffel
location	NY

e2

about	Eiffel Tower
architect	Sauvestre
year	1889
location	Paris

e3

1. token blocking on the labels of the descriptions

Statue	Liberty	Lady	Eiffel	Tower
e ₁	e ₁ , e ₂	e ₂	e ₃	e ₃

=> Pairs: {(e₁, e₂)}

2. attribute matching (only between e₁ and e₂):

- #exact string matches(name, about) = 1 ("Liberty")
- #exact string matches(architect, architect) = 1 ("Eiffel")
- #exact string matches(architect, location) = 0
- #exact string matches(year, architect) = 0
- ...
- #exact string matches(located, location) = 1 ("NY")
 - matching attribute-pairs: (name, about), (architect, architect), (located, location)

$$J(\text{name, about}) = J(\{\text{Statue, Liberty}\}, \{\text{Lady, Liberty}\}) = 1/3$$

$$\text{similarity}(e_1, e_2) = (J(\text{located, location}) + J(\text{architect, architect}) + J(\text{name, about})) / 3 = (1 + 1/2 + 1/3) / 3 = 0.61$$

Blocking in the Web of Data

Technique	Put two descriptions in a common block, when they have...
Token Blocking	a common token in their values
Attribute Clustering Blocking	a common token in the values of attributes that have similar values in overall
ZenCrowd	on average, similar values for attributes that have similar values in overall

*An entity resolution task can also receive only one (**Dirty**) entity collection as input*

Can we exploit the way data are published on the Web?

Many URIs contain semantics

- Use them as indications of matches between descriptions

[Papadakis et al. 2010]

E.g. 66% of the 182 million URIs of BTC09 follow the scheme: Prefix-Infix(-Suffix)

- Prefix describes the source, i.e. domain, of the URI
- Infix is a local identifier
- The optional Suffix contains details about the format, e.g. .rdf and .nt, or a named anchor

Prefix-Infix(-Suffix) [Papadakis et al. 2012]

Token blocking on the Infixes/literals appearing in the values of descriptions

http://en.wikipedia.org/wiki/Linked_data#Principles

- **Prefix**: describes the source (domain)
- **Infix**: local identifier
- **Suffix** (optional): details about the format, or a named anchor

Techniques:

Infix blocking

- The blocking key is the infix of the URI of the entity description

Infix profile blocking

- The blocking keys are the infixes in the values of each entity description

Infix Blocking

The blocking key is the infix of the URI of the entity description

yago:Statue_of_Liberty

dbpedia:Statue_of_Liberty

fb:m.072p8

geonames:5139572

skos:prefLabel	Statue of Liberty
yago:isLocatedIn	yago:Liberty_Island e1

rdfs:label	Statue of Liberty
dbprop:location	dbpedia:Liberty_Island e2

fb:official_name	Statue of Liberty
fb:contained_by	fb:m.026kp2
ex:location	ex:Liberty_Island e3

geoname:name	Statue of Liberty
geoname:nearby	geonames:5124330 e4

yago:Tina_Brown

skos:prefLabel	Tina Brown
yago:linksTo	yago:Liberty_Island e5

Generated blocks:

Statue_of_Liberty	m.072p8	5139572	Tina_Brown
e ₁ , e ₂	e ₃	e ₄	e ₅

Infix Profile Blocking

The blocking keys are the infixes in the values of each entity description

skos:prefLabel	Statue of Liberty	rdfs:label	Statue of Liberty	fb:official_name	Statue of Liberty	geoname:s:name	Statue of Liberty
yago:isLocatedIn	yago:Liberty_Island e1	dbprop:location	dbpedia:Liberty_Island e2	fb:contained_by	fb:m.026kp2	geoname:s:nearby	geonames:5124330 e4
skos:prefLabel	Tina Brown			ex:location	ex:Liberty_Island e3		
yago:linksTo	yago:Liberty_Island e5						

pros: (e1, e3) correctly identified
cons: (e1, e5) mistakenly identified

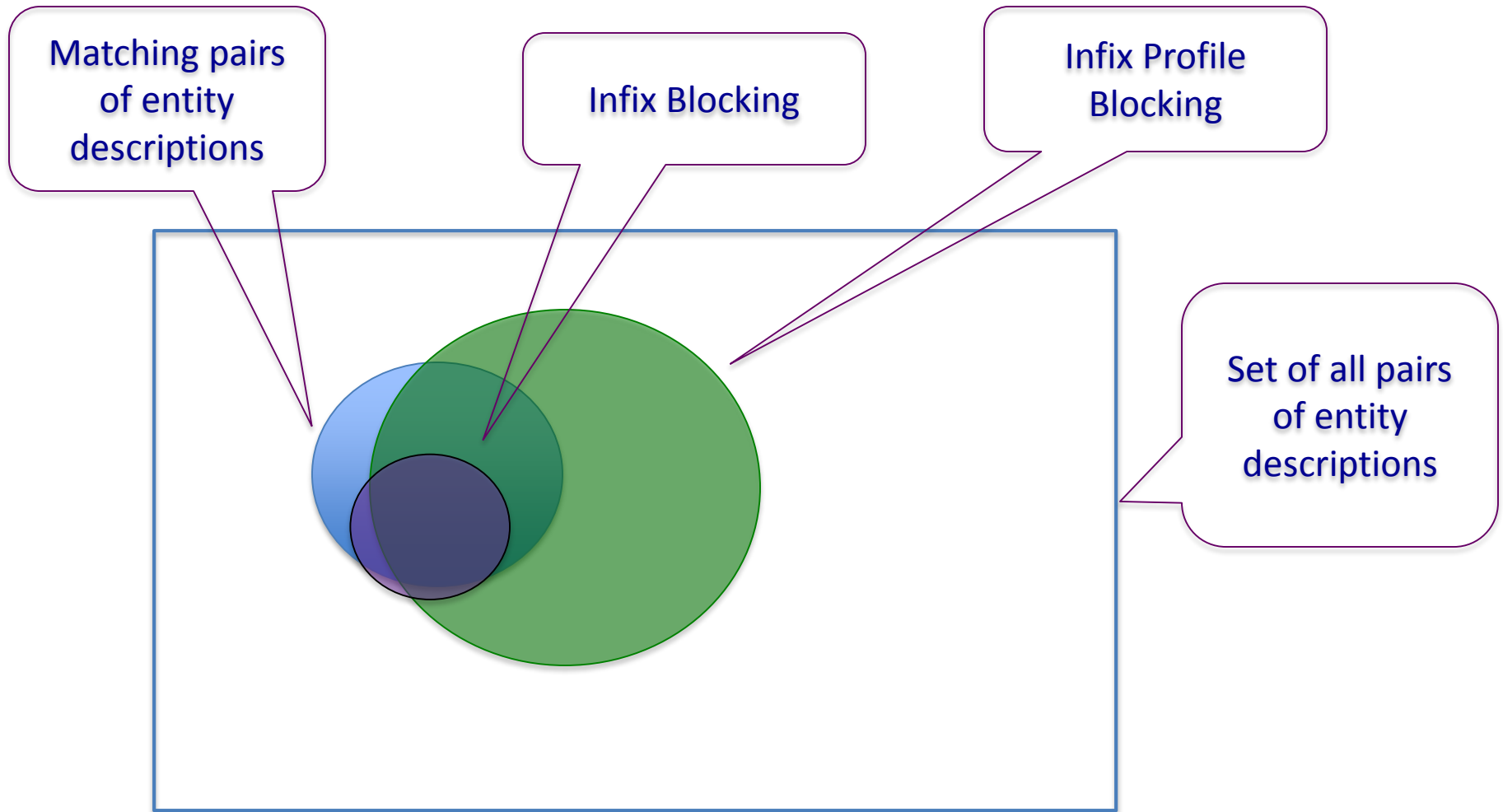
Generated blocks:

Liberty_Island	m.026kp2	5124330
e ₁ , e ₂ , e ₃ , e ₅	e ₃	e ₄

Drawback!

The effectiveness of these approaches relies on the good naming practices of the data

Prefix-Infix(-Suffix) - Evaluation



Blocking in the Web of Data

Technique	Put two descriptions in a common block, when they have...
Token Blocking	a common token in their values
Attribute Clustering Blocking	a common token in the values of attributes that have similar values in overall
ZenCrowd	on average, similar values for attributes that have similar values in overall
Prefix-Infix(-Suffix)	a common token in their literal values, or a common URI

Entity Resolution in the Web of Data

So far...

Rely on the values of the descriptions

- *A good way to handle data heterogeneity and low structuredness*

=> Deal with loosely structured entities

=> Deal with various vocabularies
(side effect)

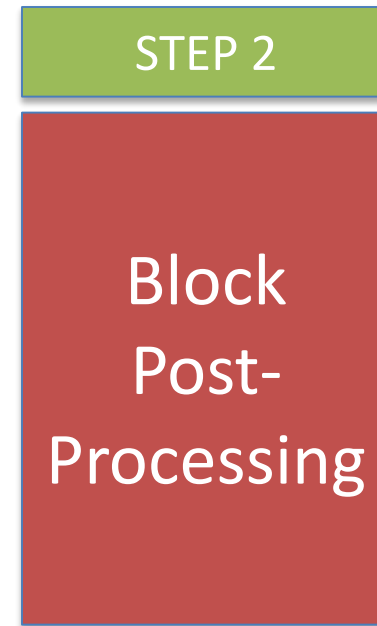
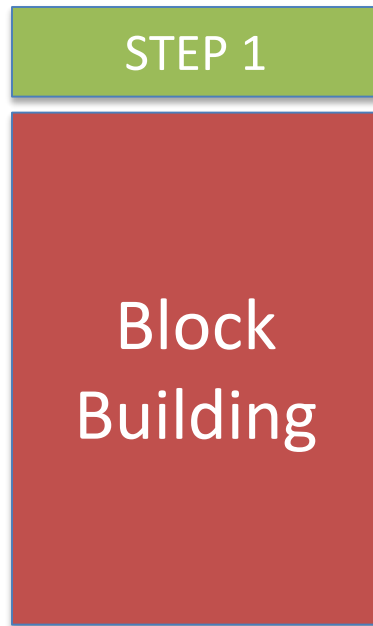
Still, many redundant comparisons are performed!

- Can we also use the structural type of the descriptions?

For further enhancing efficiency of entity resolution

Block Post-Processing

Block Post-Processing

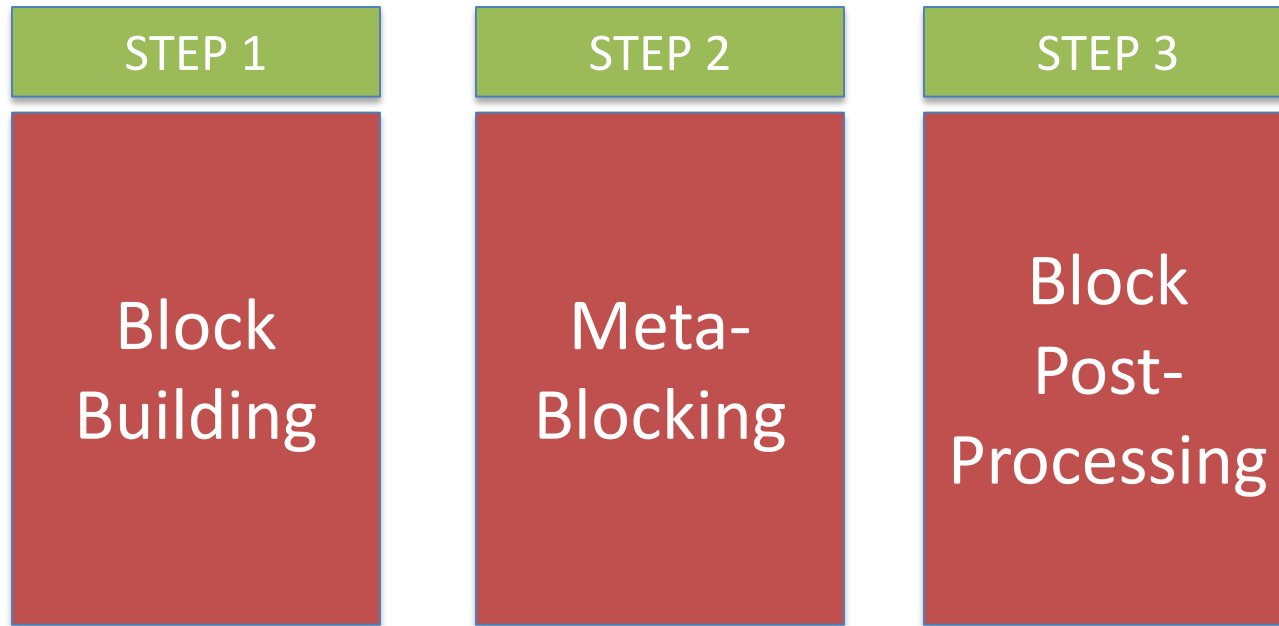


The goal: *Reduce further the number of comparison*

Block Post-Processing

- Remove oversized blocks
 - Threshold on the number of descriptions in a block
- Order blocks
 - Examine first the blocks which are more likely to contain matches
 - Wrt. the number of superfluous comparisons spared in subsequently examined blocks
- Remove low-order blocks
 - We do not gain much by examining them
- Order comparisons
 - Perform first the comparisons that are more likely to result in matches
 - Based on the number of blocks they appear together [Papadakis et al. 2011b]
- Remove low-order comparisons [Whang et al. 2013, Papadakis et al. 2011b]
 - Similar to removing low-order blocks

Meta-Blocking



Meta-blocking [Papadakis et al. 2013 (b)]

A generic procedure for block re-construction

- Create blocks resulting in fewer comparisons
- Preserve effectiveness

Blocking graph: abstract graph representation of the original set of blocks

- Nodes: entity descriptions
- Edges: connect descriptions co-occurring in blocks

Use the blocking graph for discarding redundant comparisons

- i.e. comparisons already performed

Prune edges, not satisfying a criterion, for discarding superfluous comparisons

- i.e. comparisons between non-matches

Meta-blocking - Example

about	Eiffel Tower
architect	Sauvestre
year	1889
located	Paris

e4

name	White Tower
location	Thessaloniki
year-constructed	1450

e5

name	Eiffel Tower
architect	Sauvestre
year	1889
location	Paris

e1

name	Statue of Liberty
architect	Bartholdi Eiffel
year	1886
located	NY

e2

about	Lady liberty
architect	Eiffel
location	NY

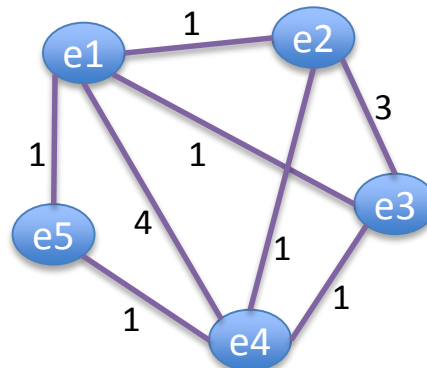
e3

Blocks:
(with token blocking)

Eiffel	Tower	Liberty
e ₁ , e ₂ , e ₃ , e ₄	e ₁ , e ₄ , e ₅	e ₂ , e ₃
NY	Paris	1889
e ₂ , e ₃	e ₁ , e ₄	e ₁ , e ₄

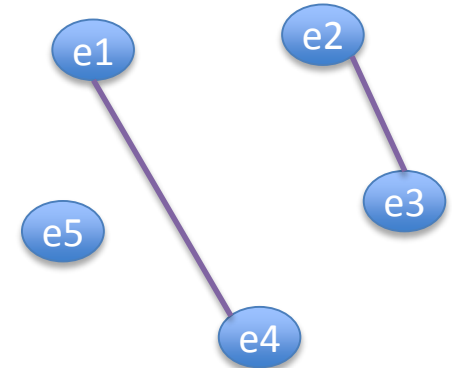
13 comparisons
to identify 2 matches

Blocking graph:



edge weights = #common blocks

Pruned blocking graph:
(remove edges with weight < 2)



2 comparisons
to identify 2 matches

Conclusions of Part I

Partitioning vs. Overlapping Blocks

Blocking approaches can be distinguished between:

- Partitioning: Each description is placed in exactly one block
 - Fewer comparisons
- Overlapping: Each description is placed in more than one block
 - More identified matches

Selecting a good blocking key is more important than the blocking technique
[Christen 2012]

In the Web of Data, selecting a (good) blocking key is not straightforward!

Discussion on Blocking

In overlapping approaches, *the number of common blocks between two descriptions can be an indication of their similarity*

- Overlap-positive: many common blocks → very similar
- Overlap-negative: few common blocks → very similar
- Overlap-neutral: #common blocks is irrelevant

Overlapping approaches return more matches

- Trade-off between the number and the size of the blocks:
 - Few, large blocks vs. many, small blocks
 - More comparisons vs. more missed matches

Overlap-positive: lower misclassification cost

- *Seem more appropriate for the Web of data*

A Classification of Blocking Approaches

Approach	Partitioning	Overlapping		
		positive	negative	neutral
Fellegi & Sunter 1969	•			
Hernandez & Stolfo 1995				•
Yan et al. 2007	•			
Draisbach & Naumann 2009				•
McCallum et al. 2000			•	
Christen 2012			•	
Gravano et al. 2001		•		
Aizawa & Oyama 2005		•		
Jin et al. 2003		•		
Kolb et al. 2011, 2012	•			
Papadakis et al. 2011		+		
Papadakis et al. 2013 (a)		+		
Papadakis et al. 2013 (b)		+		
Papadakis et al. 2012		+		

•: tabular data

+ : graph data

Tutorial Overview

- Iterative entity resolution approaches
 - Coffee break!

What follows in Part II:

- Continue on iterative entity resolution approaches
- Large scale entity resolution using MapReduce
- Conclusions

Iterative Approaches

Iterative Entity Resolution

Basic algorithm for entity resolution in one source E (dirty)

- Compare each entity description $e_i \in S$ with all other entity descriptions in E , i.e., with all $e_j \in E \setminus \{e_i\}$
- For comparison, use a match function to classify each pair (e_i, e_j) as a match/non-match
 - Based on similarity measures
 - Based on domain-specific rules
 - Based on a combination of both
- Complexity: $O(N^2)$, with N being the number of entity descriptions in E

Algorithm easily extends to entity resolution among two sources (clean-clean or dirty-dirty)

Iterative Entity Resolution

Partial results of the entity resolution process can be propagated to generate new results

Iterative approaches can be grouped into:

- Matching-based: Exploit relationships between entity descriptions
 - *If descriptions related to e_i are similar to descriptions related to e_j , this is an evidence that e_i and e_j are also similar*
- Merging-based: Exploit the partial results of merging descriptions

Tutorial Overview

What follows in Part II:

- Continue on iterative entity resolution approaches
- Large scale entity resolution using MapReduce
- Conclusions

Entity Resolution in the Web of Data

Part II

Kostas Stefanidis¹, Vasilis Efthymiou^{1,2},
Melanie Herschel^{3,4}, Vassilis Christophides⁵

kstef@ics.forth.gr, vefthym@ics.forth.gr, melanie.herschel@lri.fr
vassilis.christophides@technicolor.com

¹FORTH, ²University of Crete, ³Université Paris Sud, ⁴Inria Saclay,
⁵Paris R&I Center, Technicolor

Iterative Approaches

Iterative Entity Resolution

Basic algorithm for entity resolution in one source E (dirty)

- Compare each entity description $e_i \in S$ with all other entity descriptions in E , i.e., with all $e_j \in E \setminus \{e_i\}$
- For comparison, use a match function to classify each pair (e_i, e_j) as a match/non-match
 - Based on similarity measures
 - Based on domain-specific rules
 - Based on a combination of both
- Complexity: $O(N^2)$, with N being the number of entity descriptions in E

Algorithm easily extends to entity resolution among two sources (clean-clean or dirty-dirty)

Iterative Entity Resolution

Partial results of the entity resolution process can be propagated to generate new results

Iterative approaches can be grouped into:

- Matching-based: Exploit relationships between entity descriptions
 - *If descriptions related to e_i are similar to descriptions related to e_j , this is an evidence that e_i and e_j are also similar*
- Merging-based: Exploit the partial results of merging descriptions

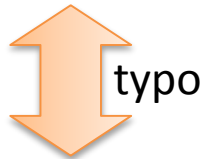
Iterative Entity Resolution on Complex Data

- Tabular data
 - Homogeneous structure
 - Similarity measures focus on variations in the values, not the structure
- Tree data
 - Structure of entity descriptions (of same and different types) varies
 - Similarity measures consider values, structure, and parent-child relationships
- Graph data
 - Structure of entity descriptions varies
 - Similarity functions consider values, structure, and neighbor relationships

Iterative Entity Resolution – Tabular Data

Table example

Name	Year	Architects	Location
Eiffel Tower	1889	Sauvestre	Paris



Eifel Tower	1889	NULL	France
-------------	------	------	--------



- Input:
 - A relation with N tuples
 - A similarity measure
- Output:
 - Classes (clusters) of equivalent tuples (= matches)
- Problem: a large number of tuples
 - Comparing each pair is too costly

=> Effectiveness strongly depends on good choice

=> Avoid comparisons that (most likely) yield no match

Swoosh [Benjelloun et al. 2009]

A generic approach for entity resolution in tabular data

Black-boxes:

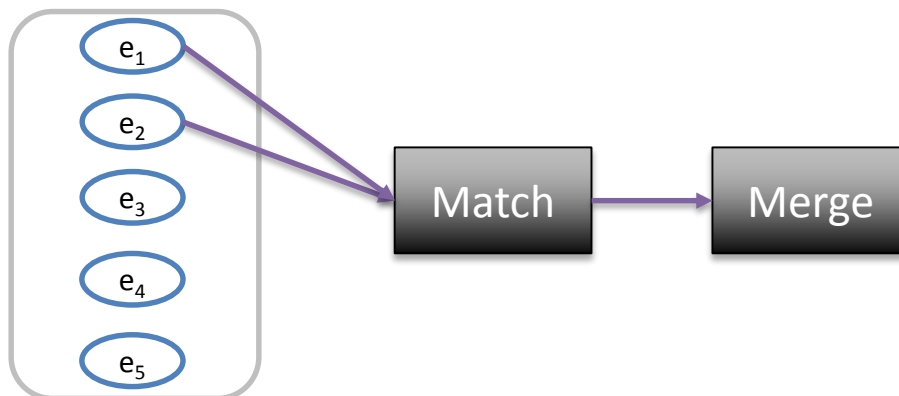
- A match function M
- A merge function μ

The goal:

Minimize the number of invocations to the these expensive black-boxes

Merged entity descriptions are considered as new entity descriptions

- Possible match candidates to other, already examined descriptions



Swoosh

A generic approach for entity resolution in tabular data

Black-boxes:

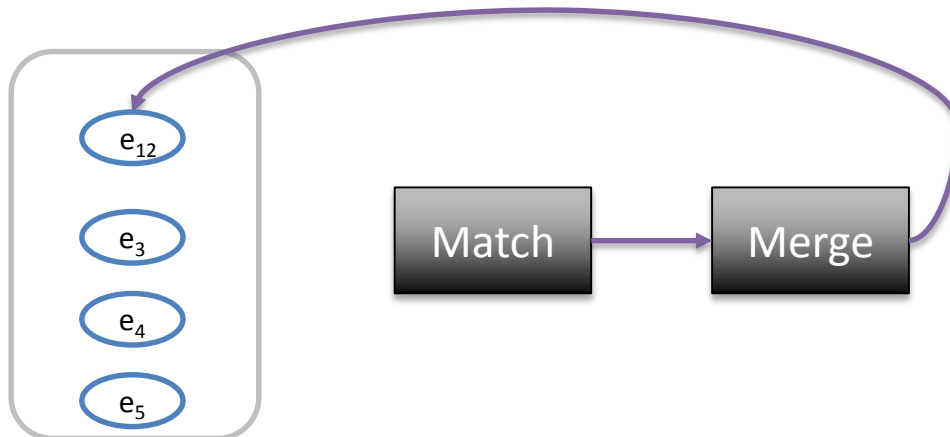
- A match function M
- A merge function μ

The goal:

Minimize the number of invocations to the these expensive black-boxes

Merged entity descriptions are considered as new entity descriptions

- Possible match candidates to other, already examined descriptions



Swoosh

Properties that can be exploited to enhance efficiency

- Idempotence:

$$M(e_1, e_1) = \text{true} \text{ and } \mu(e_1, e_1) = e_1$$

- Commutativity:

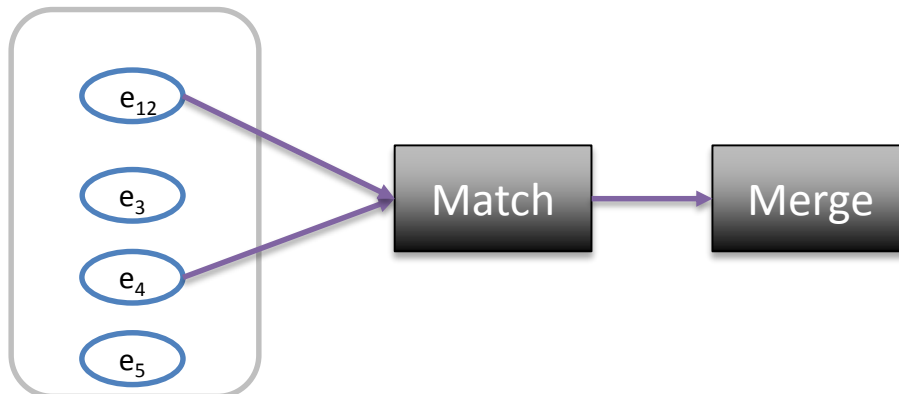
$$M(e_1, e_2) = M(e_2, e_1) \text{ and } \mu(e_1, e_2) = \mu(e_2, e_1)$$

- Associativity:

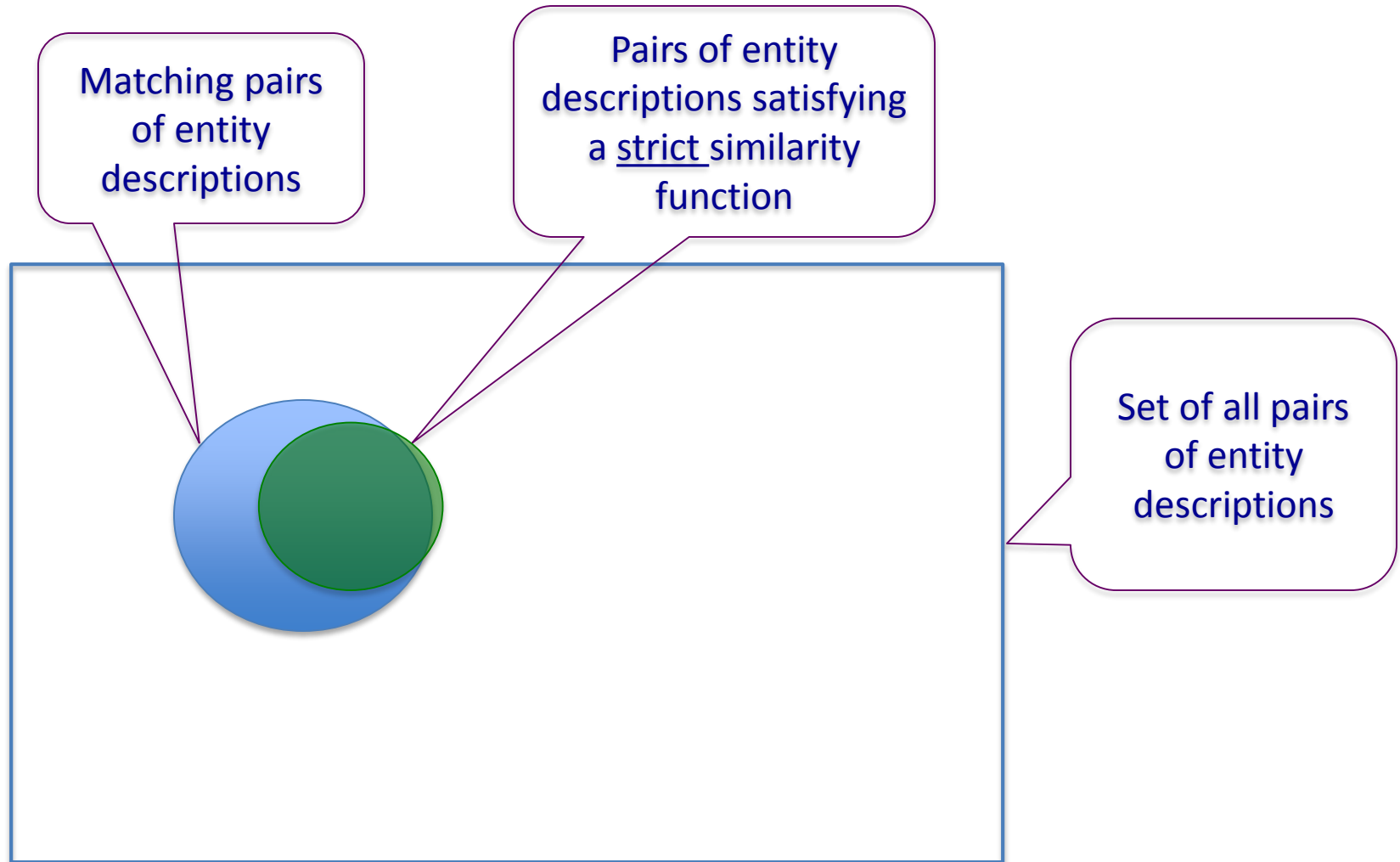
$$\mu(e_1, \mu(e_2, e_3)) = \mu(\mu(e_1, e_2), e_3)$$

- Representativity:

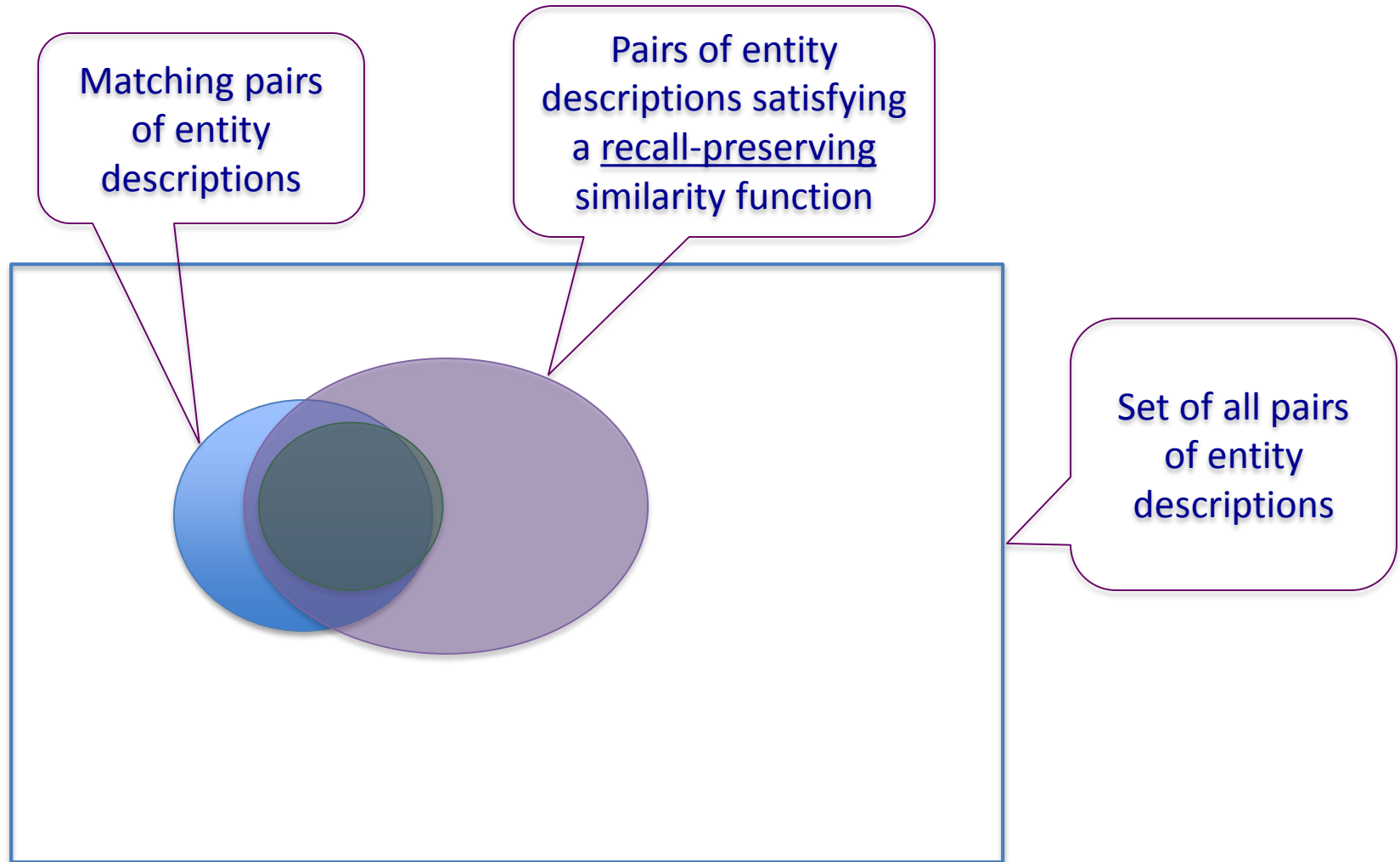
if $\mu(e_1, e_2) = e_3$ and $M(e_1, e_4) = \text{true}$, then $M(e_3, e_4) = \text{true}$



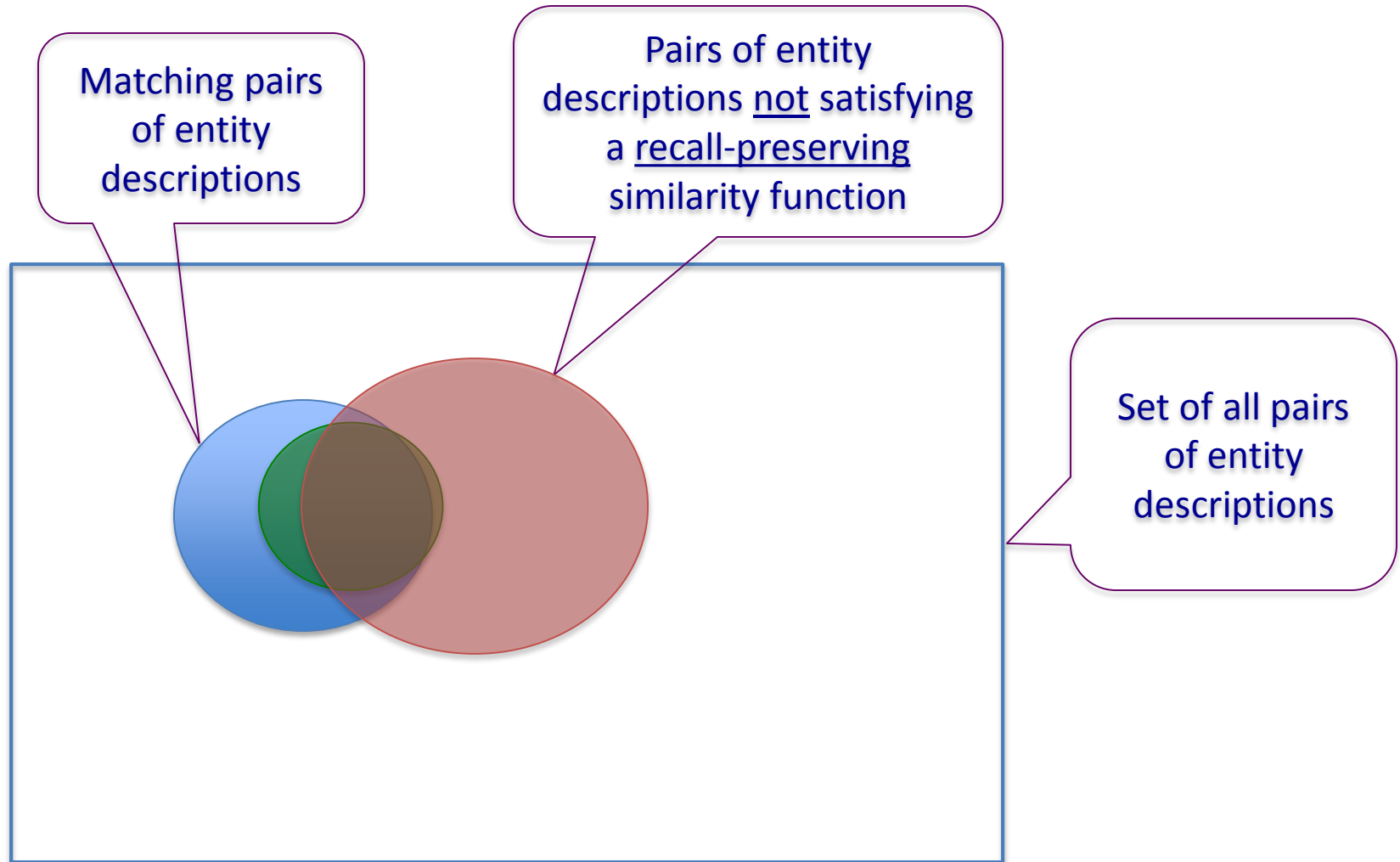
Recall-Maintaining Similarity Functions



Recall-Maintaining Similarity Functions



Recall-Maintaining Similarity Functions



Iterative Entity Resolution – Tree Data

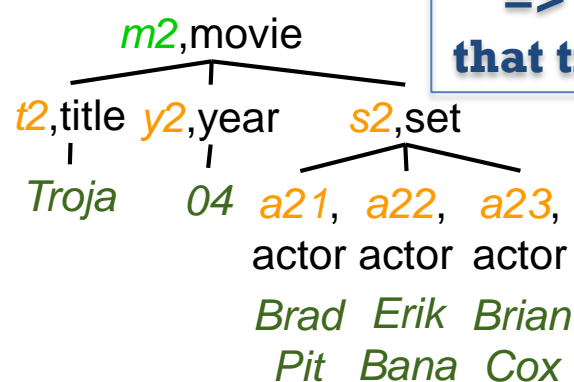
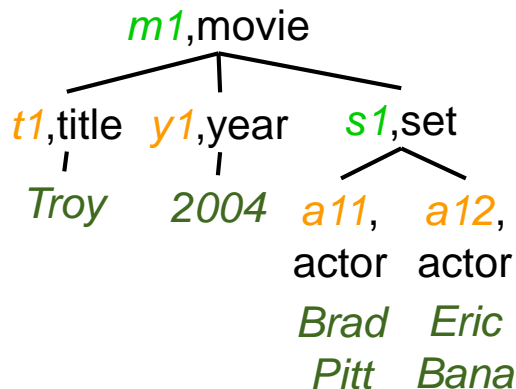
- Examples of hierarchically organized data
 - Relational star / snowflake schema [Ananthakrishna et al. 2002]

ID	Actor	Film
S1	Al Pacino	F1
S2	Al Pacino	F2
S3	Marlon Brando	F2

ID	Name	Year	Rating
F1	The Godfather	1972	9.2
F2	Gottvatter, The	72	

=> Specialized similarity measures

- Hierarchical XML data [CHL10]



=> Specialized algorithms that traverse the tree structure

DELPHI Containment Metric [ACG02]

- Hybrid similarity measure [Ananthakrishna et al. 2002] considering
 - Similarity of attribute values (*tcm*)
 - Similarity of children sets reached by following foreign keys (*fkcm*)
- Similarity of attribute values
 - Divide tuples into tokens \rightarrow token sets *TS*
 - Compute the edit distance between token sets
 - Determine weight of each token using IDF [Baeza-Yates & Ribeiro-Neto 1999]
 - The token similarity metric *tcm* measures which fraction of one tuple *T* is covered by the other tuple *T'*

$$tcm(T, T') = \frac{\sum idf(TS(T) \cap TS(T'))}{\sum idf(TS(T))}$$

DELPHI Containment Metric [ACG02]

- Similarity of children sets
 - The children set of a tuple T includes all tuples referencing T from other relations by means of a foreign key
→ Children sets CS
 - Foreign-key containment metric ($fkcm$) measures at what extent the children set of a tuple T is covered by the children set of a tuple T'

$$fkcm(T, T') = \frac{|CS(T) \cap CS(T')|}{|CS(T)|}$$

Containment Metric

– Combining *tcm* and *fkcm*:

- Both *tcm* and *fkcm* are assigned an IDF weight
- Use of a classification function:

$$\text{pos}(x) = \begin{cases} 1 & \text{if } x > 0, \\ -1 & \text{otherwise} \end{cases}$$

- Threshold for *tcm*: *s1*
- Threshold for *fkcm*: *s2*
- Classification of pairwise comparison between *T* and *T'* using

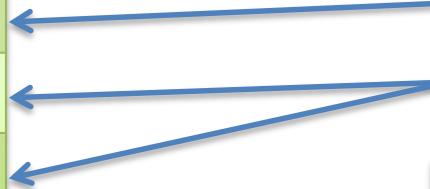
$$\text{pos}(\text{IDF}(TS) * \text{pos}(tcm(T, T') - s1) + \text{IDF}(CS) * \text{pos}(fkcm(T, T') - s2))$$

- If final result equals 1, then match, otherwise non-match

Containment Metric - Example

ID	Actor	Film
S1	Al Pacino	F1
S2	Al Pacino	F2
S3	Marlon Brando	F2

ID	Name	Year	Rating
F1	The Godfather	1972	9.2
F2	Gottvatter, The	72	



1. Token sets:

$TS(F1) = \{\text{The, Godfather, 1972, 9.2}\}$

$TS(F2) = \{\text{Gottvatter, The, 72}\}$

2. Attribute similarities

The = The, Godfather = Gottvatter, 1972 = 72.

3. Weights

For simplification, we assume all tokens have equal weight.

4. Token containment metric

$tcm(F1, F2) = \frac{3}{4}$, $tcm(F2, F1) = 1$

5. Children co-occurrence

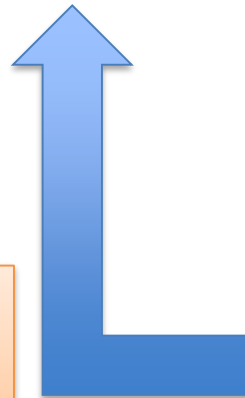
$fkcm(F1, F2) = 1$, $fkcm(F2, F1) = \frac{1}{2}$



6. Combination of both metrics

($s1 = s2 = 0.5$, weights = 1)

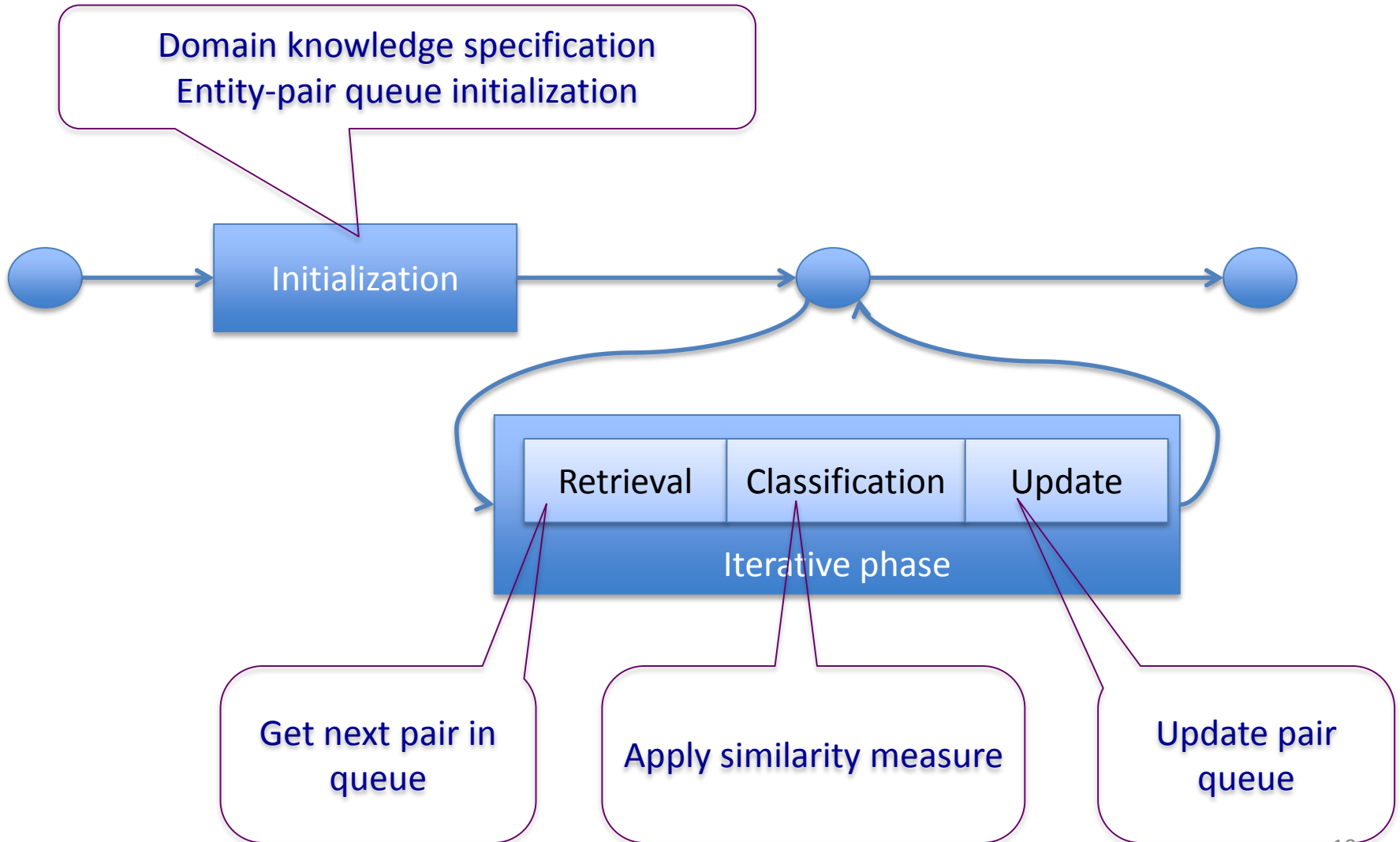
$pos(pos(\frac{3}{4} - 0.5) + pos(1 - 0.5)) = 1$
 \rightarrow F1 and F2 match



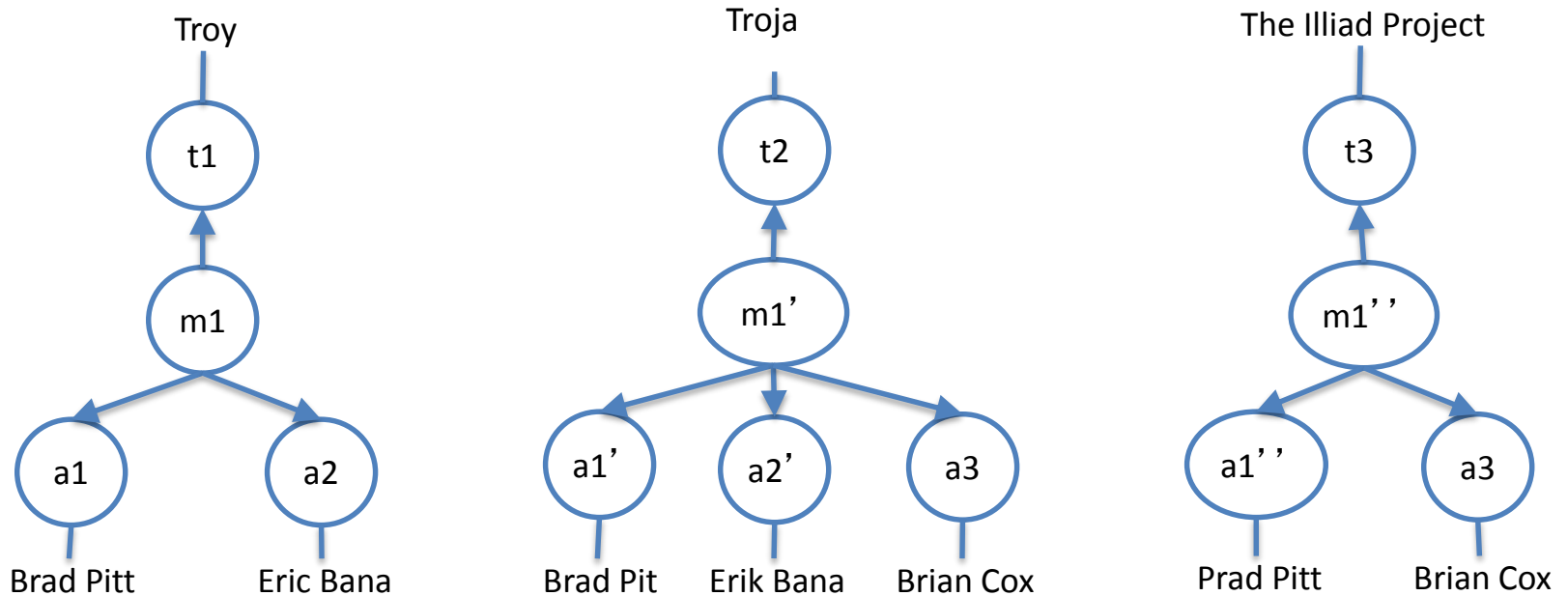
Iterative Entity Resolution - Graph Data

- In the most general case, data not only form a tree, but a graph
 - LOD graph
 - General relational schema
 - Domain-knowledge about entity relationships
 - ...
- In graph data, there is no clear order of comparisons (top down, bottom-up?)
- Several algorithms for entity resolution in graph data have been proposed [Dong et al. 2005, Weis & Naumann 2006, Bhattacharya & Getoor 2007, ...]
 - Based on an entity graph
(1 node = 1 entity, 1 edge = relationship between 2 entities)
 - Based on reference graph
(1 node = 2 entities, 1 edge = relationship to another entity pair)
- Many of them conform to a general framework [Herschel et al. 2012]

Iterative Entity Resolution – Graph Framework

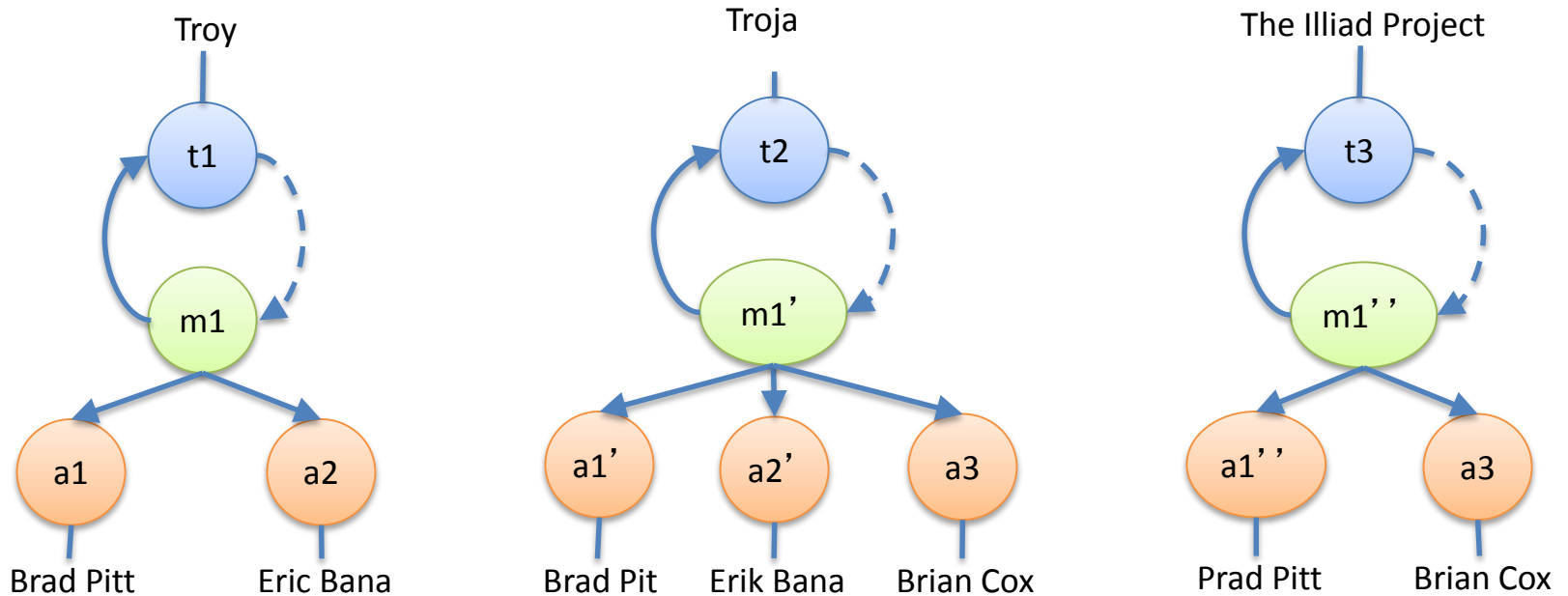


Domain Expert Knowledge Specification



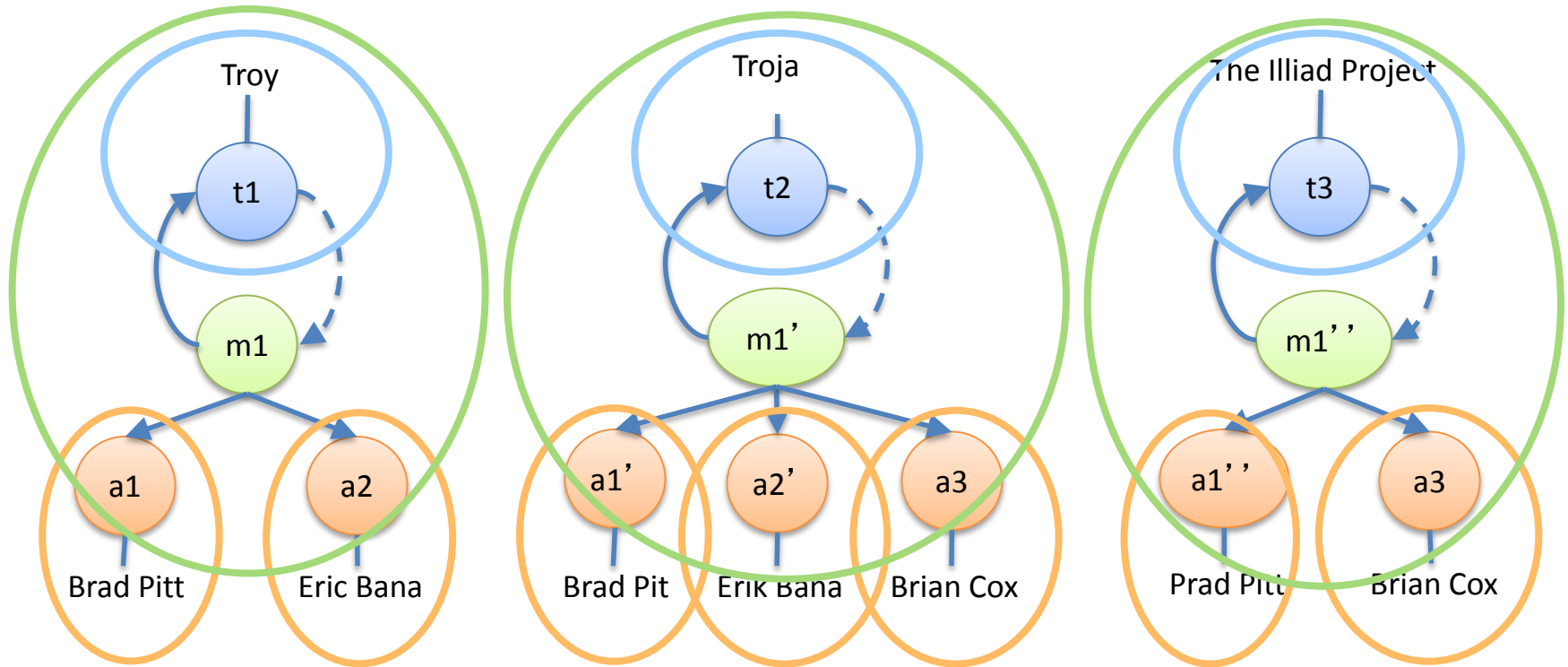
- Domain expert specifies
 - Duplicate candidate entities (e.g., movie, actor, title)
 - (Additional) relationships between candidates (e.g., title \rightarrow movie)

Domain Expert Knowledge Specification



- Domain expert specifies
 - Duplicate candidate entities (e.g., movie, actor, title)
 - (Additional) relationships between candidates (e.g., title → movie)

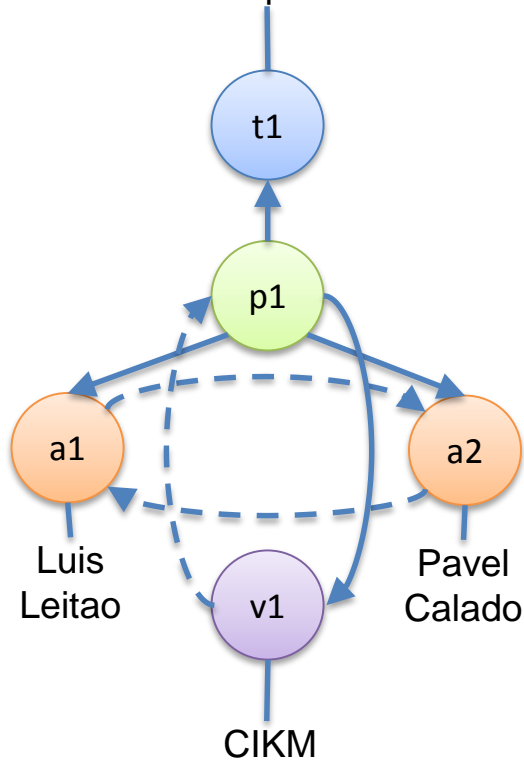
Domain Expert Knowledge Specification



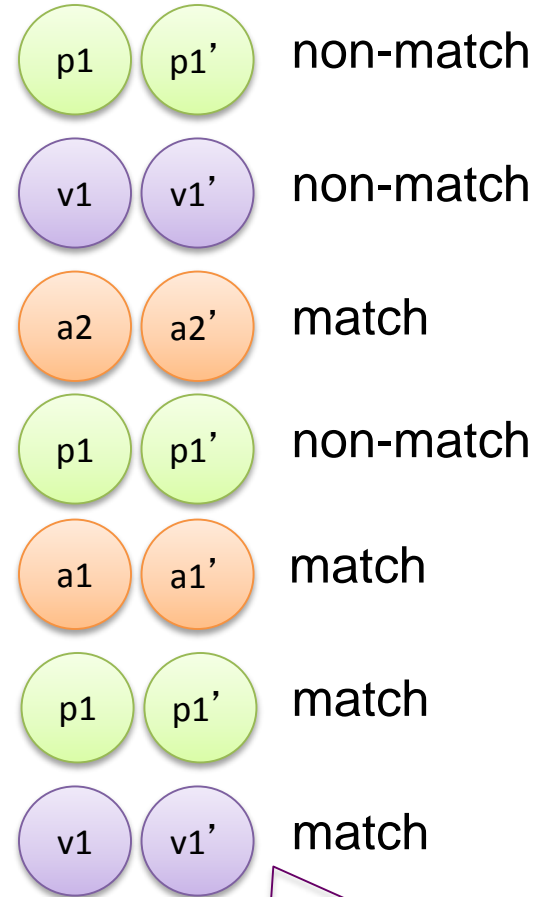
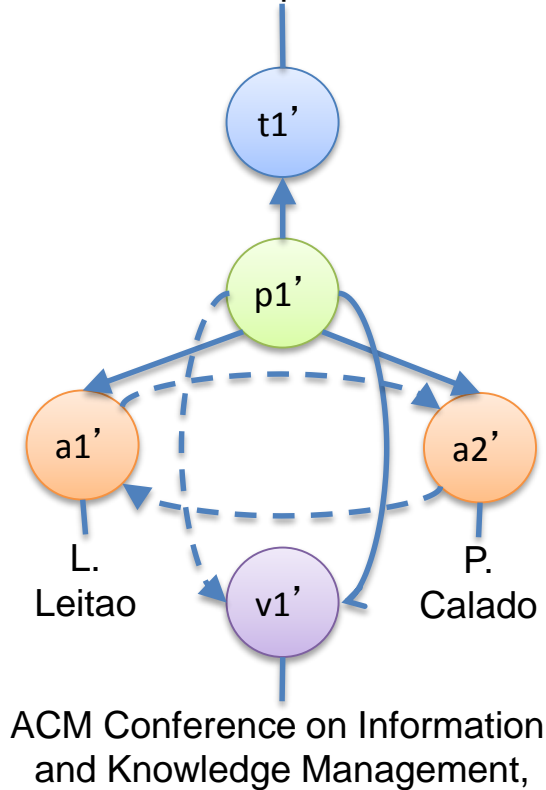
- For pairwise similarity computation, domain expert also selects what information is relevant for comparisons
 - Entity description (attribute values)
 - Influencing neighbor candidates

Entity Pair Queue

Duplicate detection through structure optimization



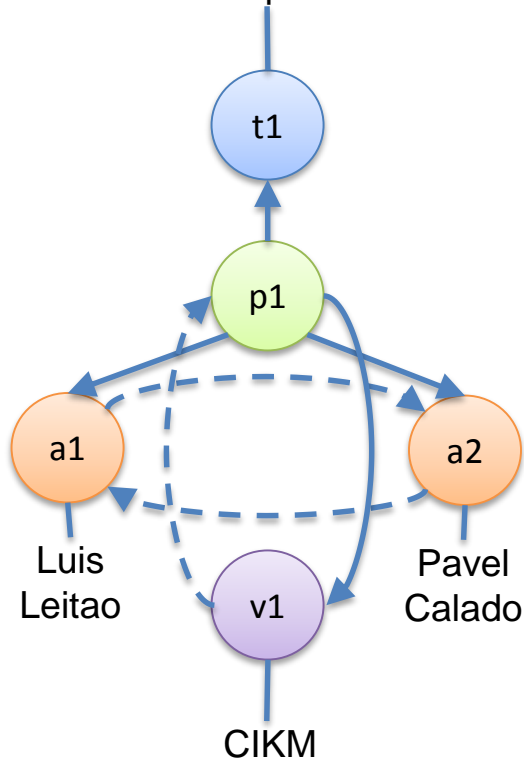
Duplicate detection through structure optimization



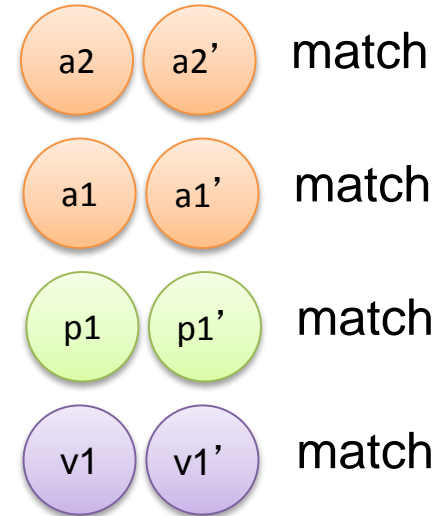
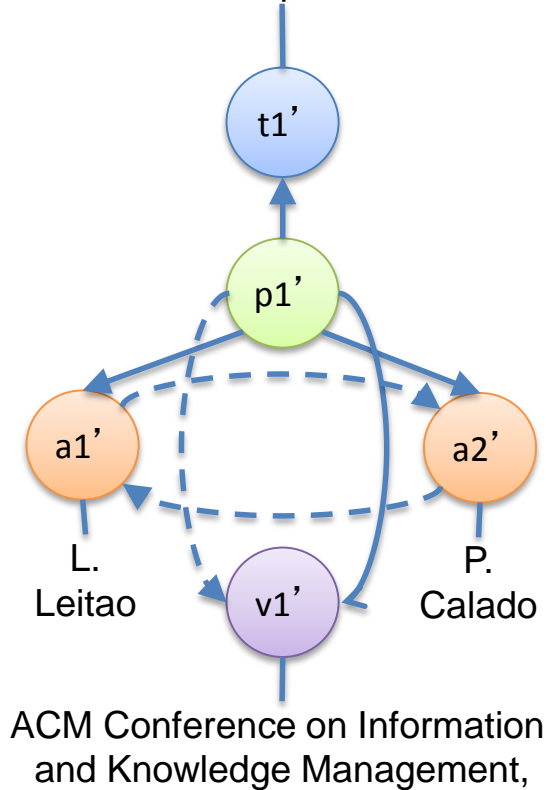
7 comparisons
(3 re-comparisons)

Entity Pair Queue

Duplicate detection through
structure optimization



Duplicate detection through
structure optimization



4 comparisons
(0 re-comparisons)

Entity Pair Queue

- Queue maintenance necessary whenever a match is found
 - Manage order in which pairs are compared to reduce re-comparisons
 - Merge matches:
 - Let $m = \text{merge}(e1, e2)$
 - Replace all occurrences of $e1$ and $e2$ in pair queue by m
 - Add additional pairs to queue that compare m with entities already compared to either $e1$ or $e2$
- In general, goal of maintaining the priority queue is to reduce the number of re-comparisons while maximizing effectiveness

Iterative blocking

Entity resolution interleaved with blocking

Iterative Blocking [Whang et al. 2009]

Blocking is not just a simple preprocessing step of entity resolution

Perform entity resolution on each block, and propagate results to other blocks

Entity resolution results of a processed block, may help identifying more matches in another block

- Newly created entity descriptions, i.e. merges of descriptions found matching, are distributed to other blocks, replacing the found matches

Blocks are processed multiple times, until no new matches are found

Iterative Blocking - Example

	Name	<u>Year</u>	Architects	<u>Location</u>
e_1	Eiffel Tower	1889	Sauvestre	<u>P</u> aris
e_2	Statue of Liberty	1886	Bartholdi, Eiffel	<u>N</u> Y
e_3	Lady Liberty	1885	Eiffel	<u>L</u> iberty Island, NY
e_4	Eiffel Tower	1889		<u>P</u> aris
e_5	Miss Liberty	1886	Gustave Eiffel	<u>L</u> iberty Island

Blocks generated if blocking keys are the year and the 1st letter of the location:

1889	1886	1885
e_1, e_4	e_2, e_5	e_3
P	N	L
e_1, e_4	e_2	e_3, e_5

Iterative Blocking - Example

	Name	Year	Architects	Location
e_1	Eiffel Tower	1889	Sauvestre	<u>P</u> aris
e_2	Statue of Liberty	1886	Bartholdi, Eiffel	<u>N</u> Y
e_3	Lady Liberty	1885	Eiffel	<u>L</u> iberty Island, NY
e_4	Eiffel Tower	1889		<u>P</u> aris
e_5	Miss Liberty	1886	Gustave Eiffel	<u>L</u> iberty Island

Blocks generated if blocking keys are the year and the 1st letter of the location:

1889	1886	1885
e_1, e_4	e_2, e_5	e_3
P	N	L
e_1, e_4	e_2	e_3, e_5

e_1, e_4 match! they are merged as e_{14}

Iterative Blocking - Example

	Name	<u>Year</u>	Architects	<u>Location</u>
e_1	Eiffel Tower	1889	Sauvestre	<u>P</u> aris
e_2	Statue of Liberty	1886	Bartholdi, Eiffel	<u>N</u> Y
e_3	Lady Liberty	1885	Eiffel	<u>L</u> iberty Island, NY
e_4	Eiffel Tower	1889		<u>P</u> aris
e_5	Miss Liberty	1886	Gustave Eiffel	<u>L</u> iberty Island

Blocks generated if blocking keys are the year and the 1st letter of the location:

1889	1886	1885
e_1, e_4, e_{14}	e_2, e_5	e_3
P	N	L
e_1, e_4, e_{14}	e_2	e_3, e_5

Iterative Blocking - Example

	Name	<u>Year</u>	Architects	<u>Location</u>
e_1	Eiffel Tower	1889	Sauvestre	<u>P</u> aris
e_2	Statue of Liberty	1886	Bartholdi, Eiffel	<u>N</u> Y
e_3	Lady Liberty	1885	Eiffel	<u>L</u> iberty Island, NY
e_4	Eiffel Tower	1889		<u>P</u> aris
e_5	Miss Liberty	1886	Gustave Eiffel	<u>L</u> iberty Island

Blocks generated if blocking keys are the year and the 1st letter of the location:

1889	1886	1885
e_1, e_4, e_{14}	e_2, e_5	e_3
P	N	L
e_1, e_4, e_{14}	e_2	e_3, e_5

e_2, e_5 match! they are merged as e_{25}

Iterative Blocking - Example

	Name	<u>Year</u>	Architects	<u>Location</u>
e_1	Eiffel Tower	1889	Sauvestre	<u>P</u> aris
e_2	Statue of Liberty	1886	Bartholdi, Eiffel	<u>N</u> Y
e_3	Lady Liberty	1885	Eiffel	<u>L</u> iberty Island, NY
e_4	Eiffel Tower	1889		<u>P</u> aris
e_5	Miss Liberty	1886	Gustave Eiffel	<u>L</u> iberty Island

Blocks generated if blocking keys are the year and the 1st letter of the location:

1889	1886	1885
e_1, e_4, e_{14}	e_2, e_5, e_{25}	e_3
P	N	L
e_1, e_4, e_{14}	e_2, e_{25}	e_3, e_5, e_{25}

e_2, e_5 match! they are merged as e_{25}

Iterative Blocking - Example

	Name	<u>Year</u>	Architects	<u>Location</u>
e_1	Eiffel Tower	1889	Sauvestre	<u>P</u> aris
e_2	Statue of Liberty	1886	Bartholdi, Eiffel	<u>N</u> Y
e_3	Lady Liberty	1885	Eiffel	<u>L</u> iberty Island, NY
e_4	Eiffel Tower	1889		<u>P</u> aris
e_5	Miss Liberty	1886	Gustave Eiffel	<u>L</u> iberty Island

Blocks generated if blocking keys are the year and the 1st letter of the location:

1889	1886	1885
e_1, e_4, e_{14}	e_2, e_5, e_{25}	e_3
P	N	L
e_1, e_4, e_{14}	e_2, e_{25}	e_3, e_5, e_{25}

Iterative Blocking - Example

	Name	<u>Year</u>	Architects	<u>Location</u>
e_1	Eiffel Tower	1889	Sauvestre	<u>P</u> aris
e_2	Statue of Liberty	1886	Bartholdi, Eiffel	<u>N</u> Y
e_3	Lady Liberty	1885	Eiffel	<u>L</u> iberty Island, NY
e_4	Eiffel Tower	1889		<u>P</u> aris
e_5	Miss Liberty	1886	Gustave Eiffel	<u>L</u> iberty Island

Blocks generated if blocking keys are the year and the 1st letter of the location:

1889	1886	1885
e_1, e_4, e_{14}	e_2, e_5, e_{25}	e_3
P	N	L
e_{14}	e_2, e_{25}	e_3, e_5, e_{25}

Iterative Blocking - Example

	Name	<u>Year</u>	Architects	<u>Location</u>
e_1	Eiffel Tower	1889	Sauvestre	<u>P</u> aris
e_2	Statue of Liberty	1886	Bartholdi, Eiffel	<u>N</u> Y
e_3	Lady Liberty	1885	Eiffel	<u>L</u> iberty Island, NY
e_4	Eiffel Tower	1889		<u>P</u> aris
e_5	Miss Liberty	1886	Gustave Eiffel	<u>L</u> iberty Island

Blocks generated if blocking keys are the year and the 1st letter of the location:

1889	1886	1885
e_1, e_4, e_{14}	e_2, e_5, e_{25}	e_3
P	N	L
e_{14}	e_{25}	e_3, e_5, e_{25}

Iterative Blocking - Example

	Name	<u>Year</u>	Architects	<u>Location</u>
e_1	Eiffel Tower	1889	Sauvestre	<u>P</u> aris
e_2	Statue of Liberty	1886	Bartholdi, Eiffel	<u>N</u> Y
e_3	Lady Liberty	1885	Eiffel	<u>L</u> iberty Island, NY
e_4	Eiffel Tower	1889		<u>P</u> aris
e_5	Miss Liberty	1886	Gustave Eiffel	<u>L</u> iberty Island

Blocks generated if blocking keys are the year and the 1st letter of the location:

1889	1886	1885
e_1, e_4, e_{14}	e_2, e_5, e_{25}	e_3

e_3, e_{25} match! they are merged as e_{235}

P	N	L
e_{14}	e_{25}	e_3, e_{25}

Iterative Blocking - Example

	Name	<u>Year</u>	Architects	<u>Location</u>
e_1	Eiffel Tower	1889	Sauvestre	<u>P</u> aris
e_2	Statue of Liberty	1886	Bartholdi, Eiffel	<u>N</u> Y
e_3	Lady Liberty	1885	Eiffel	<u>L</u> iberty Island, NY
e_4	Eiffel Tower	1889		<u>P</u> aris
e_5	Miss Liberty	1886	Gustave Eiffel	<u>L</u> iberty Island

Blocks generated if blocking keys are the year and the 1st letter of the location:

1889	1886	1885
e_1, e_4, e_{14}	$e_2, e_5, e_{25}, e_{235}$	e_3, e_{235}
P	N	L
e_{14}	e_{25}, e_{235}	e_3, e_{25}, e_{235}

e_3, e_{25} match! they are merged as e_{235}

Iterative Blocking - Example

	Name	<u>Year</u>	Architects	<u>Location</u>
e_1	Eiffel Tower	1889	Sauvestre	<u>P</u> aris
e_2	Statue of Liberty	1886	Bartholdi, Eiffel	<u>N</u> Y
e_3	Lady Liberty	1885	Eiffel	<u>L</u> iberty Island, NY
e_4	Eiffel Tower	1889		<u>P</u> aris
e_5	Miss Liberty	1886	Gustave Eiffel	<u>L</u> iberty Island

Blocks generated if blocking keys are the year and the 1st letter of the location:

1889	1886	1885
e_{14}	$e_2, e_5, e_{25}, e_{235}$	e_3, e_{235}
P	N	L
e_{14}	e_{25}, e_{235}	e_3, e_{25}, e_{235}

process continues iteratively, until no new matches are found

Extend iterative blocking by using MinHash

MinHash

Assume an entity description is a set of tokens:

$e1 = \{\text{the, statue, of, liberty}\}$

$e2 = \{\text{lady, liberty, statue}\}$

$e3 = \{\text{the, eiffel, tower}\}$

token\description	e1	e2	e3
the	1	0	1
statue	1	1	0
of	1	0	0
liberty	1	1	0
lady	0	1	0
eiffel	0	0	1
tower	0	0	1

MinHash:

- Pick a permutation of the tokens
- $\text{minHash}(e_i)$: the first token of e_i in the permuted order of tokens

$\text{minHash}(e1) = \text{'of'}$

$\text{minHash}(e2) = \text{'lady'}$

$\text{minhash}(e3) = \text{'tower'}$

token\description	e1	e2	e3
lady	0	1	0
tower	0	0	1
of	1	0	0
eiffel	0	0	1
the	1	0	1
statue	1	1	0
liberty	1	1	0

MinHash as a Jaccard Approximation

$$P[\text{minHash}(e_i) = \text{minHash}(e_j)] = \text{Jaccard}(e_i, e_j)$$

- Type A rows: both descriptions have 1
- Type B rows: one has 1 and the other has 0
- Type C rows: both have 0

$$\begin{aligned} P[\text{minHash}(e_i) = \text{minHash}(e_j)] &= \\ &= \#A / (\#A + \#B) = \\ &= \text{Jaccard}(e_i, e_j) \end{aligned}$$

Type C row

Type B row

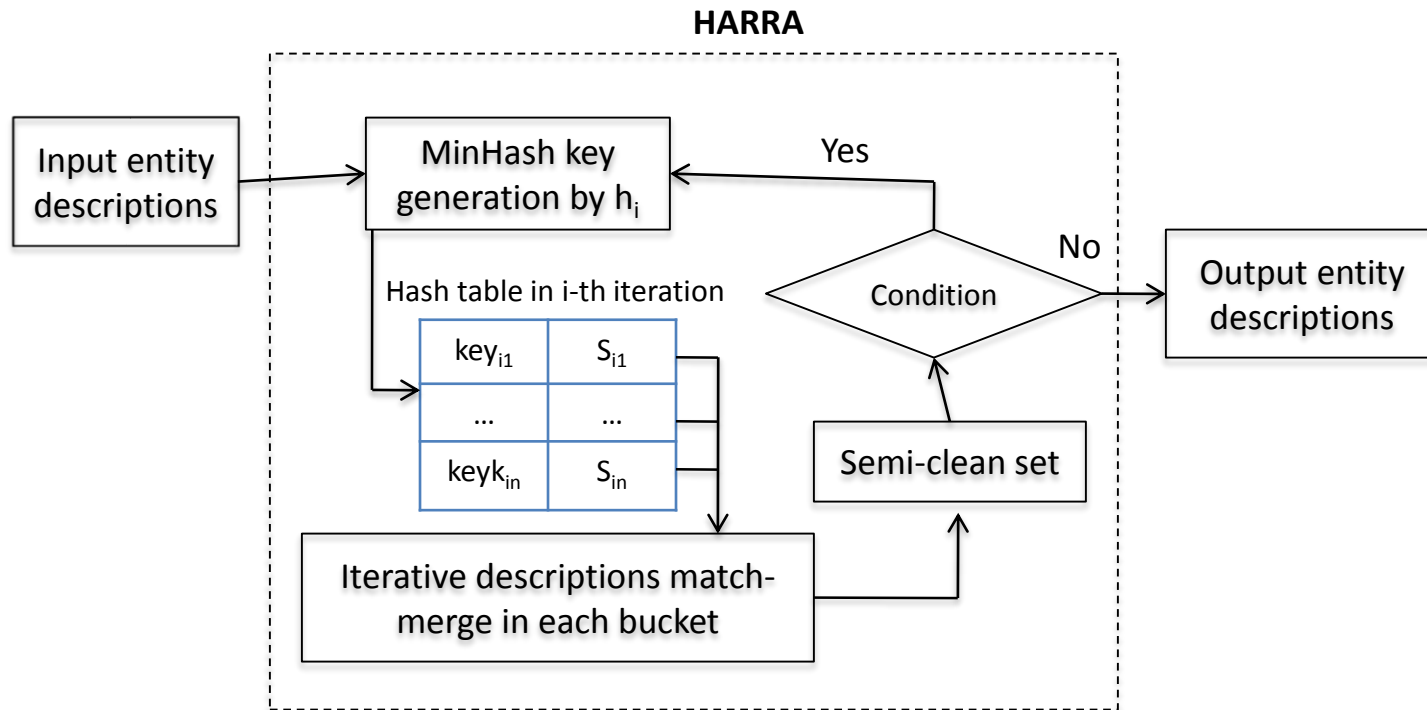
Type A row

token\description	e1	e2	e3
lady	0	1	0
tower	0	0	1
of	1	0	0
eiffel	0	0	1
the	1	0	1
statue	1	1	0
liberty	1	1	0

- Create several random permutations (h_1, \dots, h_n) of the tokens, $n \ll \#tokens$
 - Permutations are expensive; hash functions can simulate this functionality
- For each permutation, find the minHash of each description
 - These, concatenated, constitute the minHash signature of each description
- Compare the minHash signatures of the descriptions
 - Using Locality-Sensitive Hashing (LSH)

HARRA [Kim & Lee 2010]

Extends iterative blocking by employing MinHash (for Jaccard approximation)

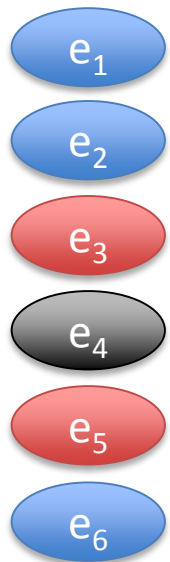


Scalability: A single hash table is used


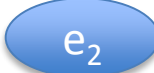



- Before placing a description in a block, the description is compared to the contents of the block

HARRA - Example

e_6 should be placed in the blue bucket



Hash Table:

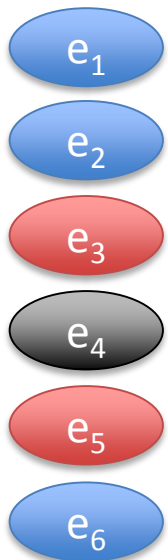
Keys	Values
Blue	 
Red	 
Black	

HARRA - Example

Before placing it there, we check if it matches e_1 or e_2

$e_6 = e_1$? NO

$e_6 = e_2$? YES



Hash Table:

Keys	Values
Blue	<div>e_1 e_2</div>
Red	<div>e_3 e_5</div>
Black	<div>e_4</div>

HARRA - Example

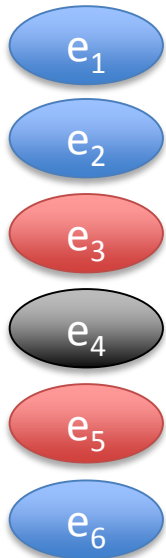
Before placing it there, we check if it matches e_1 or e_2

$e_6 = e_1$? NO

$e_6 = e_2$? YES

e_{26} is the result of merging e_6 and e_2

$e_{26} = e_1$? NO



Hash Table:

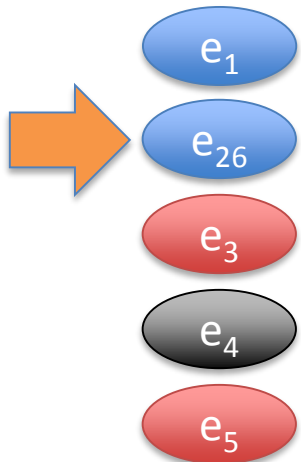
Keys	Values
Blue	<div>e_1 e_{26}</div>
Red	<div>e_3 e_5</div>
Black	<div>e_4</div>

HARRA - Example

Continue until:

- no merge occurs, OR
- saved comparisons $>$ threshold, OR
- # iterations $>$ constant

Re-initialize the input:



Hash Table:

Keys	Values
Blue	<input type="text"/>
Red	<input type="text"/>
Black	<input type="text"/>

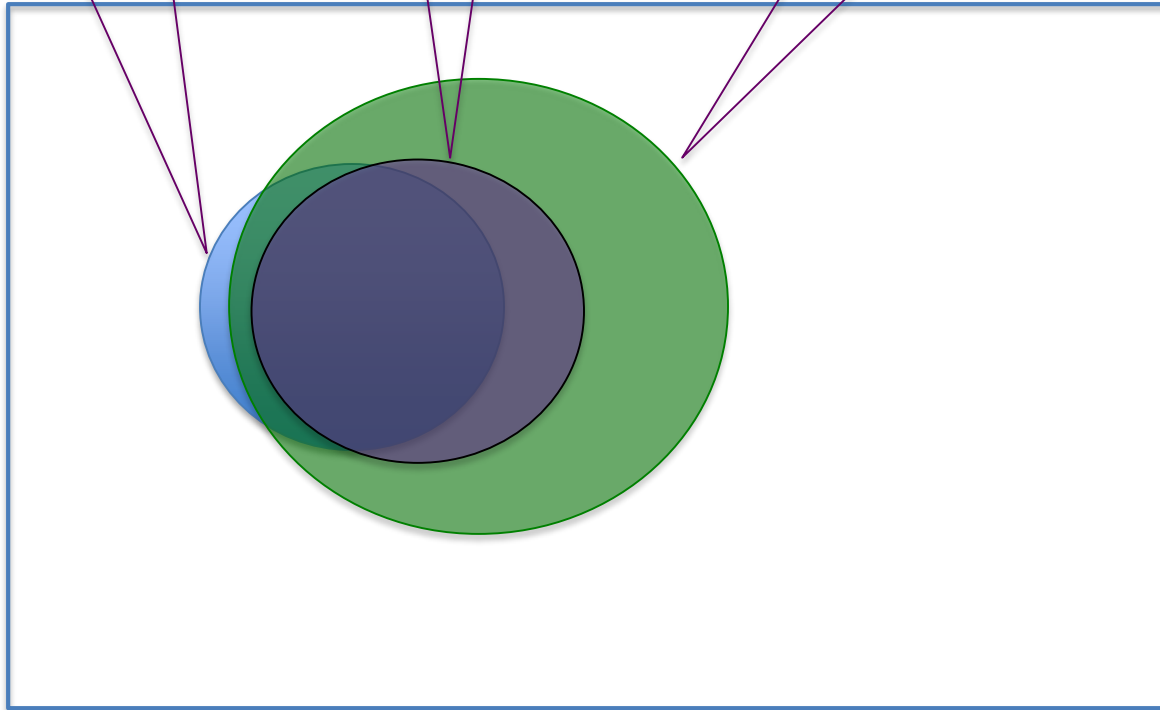
Blocking vs Iterative Blocking

Matching pairs
of entity
descriptions

Blocking

Iterative
Blocking

Set of all pairs
of entity
descriptions



Iterative Blocking
(pros) Lead to more
identified matches
(cons) Lead to more
comparisons

Discussion on Iterative Approaches

Each iteration is based on new knowledge

- Identified matches
- Merged descriptions of identified matches

Hybrid methods, i.e. *iterative blocking*, benefit from:

- The *efficiency* of blocking approaches
- The *effectiveness* of iterative approaches

Iterative approaches seem to fit well to similarity functions using relationships

- Relationships between descriptions are an important part of the available semantics

A Classification of Iterative Approaches

Approach	Matching-based	Merging-based
Bhattacharya & Getoor 2004, 2007	•	
Rastogi et al. 2011	•	
Dong et al. 2005	•	
Herschel et al. 2012	•	
Weis & Naumann 2006	□	
Weis & Naumann 2004	□	
Leitão et al. 2007, 2013	□	
Puhlmann et al. 2006	□	
Böhm et al. 2012	+	
Benjelloun et al. 2009		•
Benjelloun et al. 2007		•
Whang et al. 2009		•
Kim & Lee 2010		•

- : tabular data
- : tree data
- + : graph data

Scalability Limitations

Computations are sequential

- High time requirements
 - E.g., 66 hours, just to create the clusters of attributes for attribute clustering blocking for a dataset of 3M entities

Data reside in the memory of a single machine

- High memory and space requirements
 - I.e., “out of memory” errors, when datasets get bigger and in some cases “no space left on device” errors

Parallel algorithms can be used to overcome these limitations

For handling huge volumes of data

MapReduce

MapReduce

Input data are partitioned

Input data partitions are sent to different nodes (mappers) in the cluster

- **Map phase:** distribute the current partition to multiple nodes (reducers)
 - Emit (key, value) pairs
 - Pairs with the same key are processed by the same reducer
- **Reduce phase:** process the pairs having the same key
 - Emit (key, value) pairs – the output of the program

MapReduce

For handling huge volumes of data:

Proceed entity resolution in partitions!

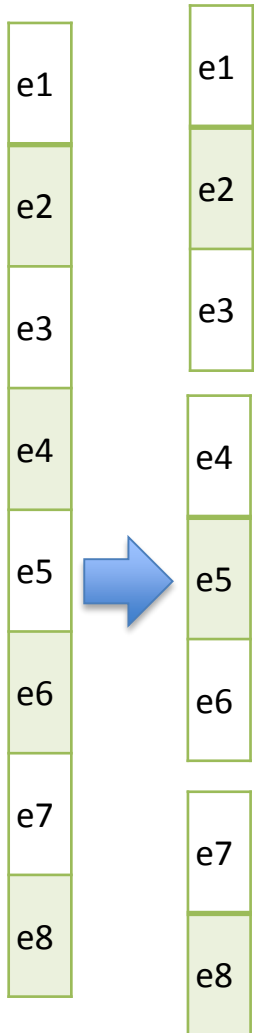
The map phase reflects blocking (re-distribute descriptions)

The reduce phase reflects entity resolution (check for matches)

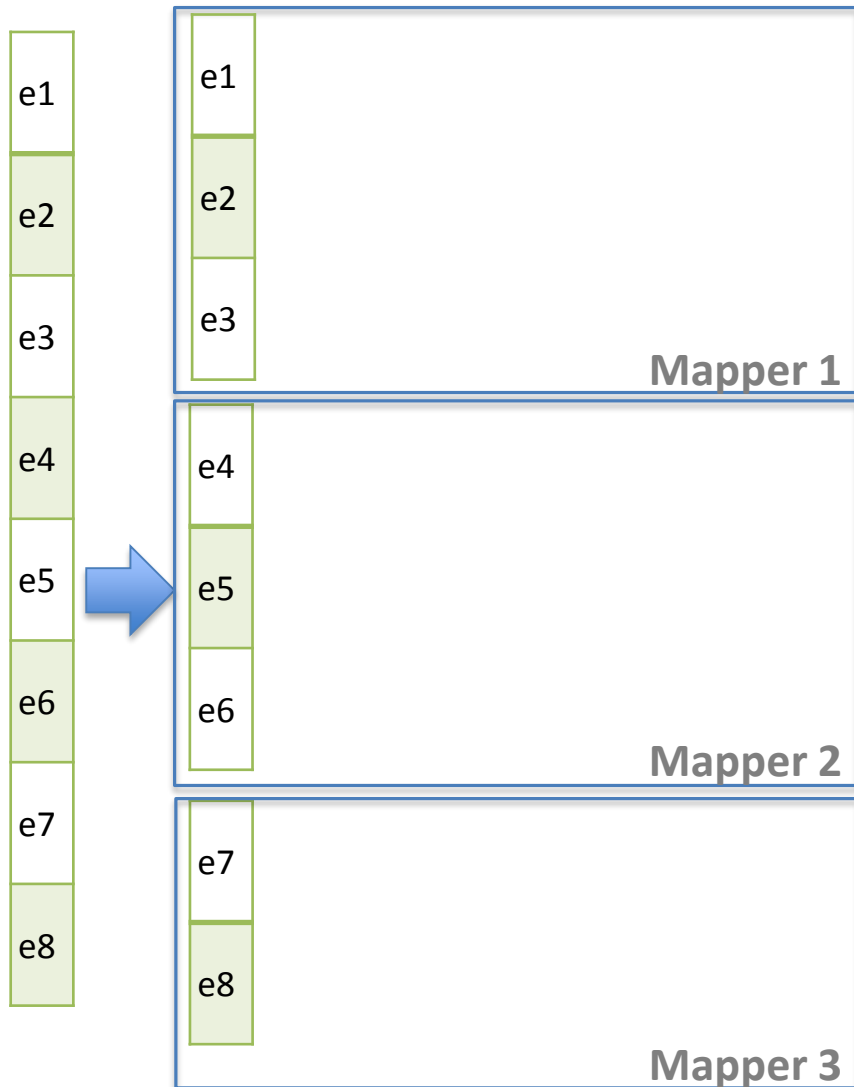
MapReduce – Input Data

e1
e2
e3
e4
e5
e6
e7
e8

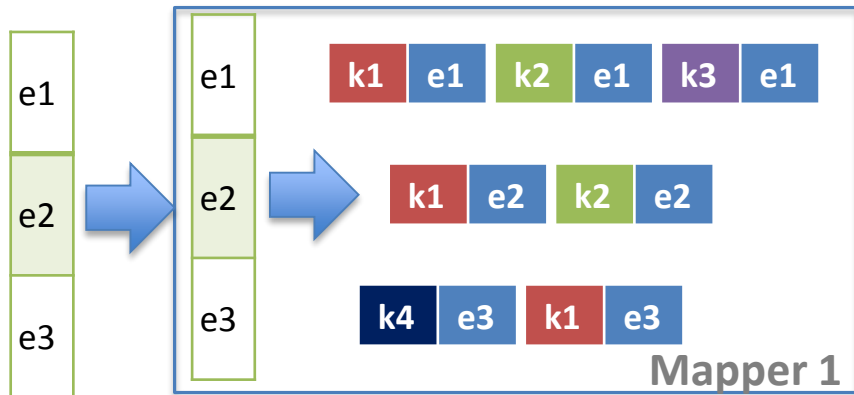
MapReduce – Input Data Partitioning



MapReduce – Mapper Input



MapReduce – Mapper Example



Input :

e1={{(name, Auguste Bartholdi),(year,1834)}}

e2={{(about, Auguste Bartholdi)}}

e3={{(architects, Bartholdi Eiffel)}}



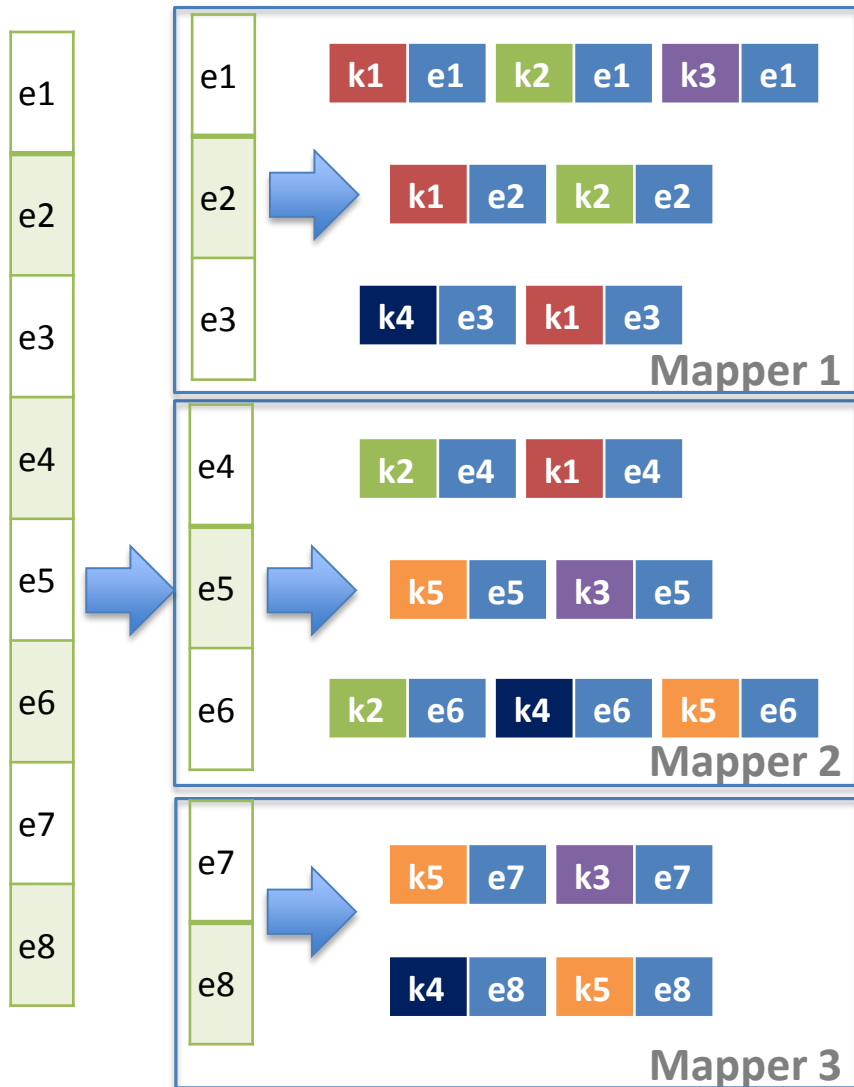
Output :

Bartholdi e1 Auguste e1 1834 e1

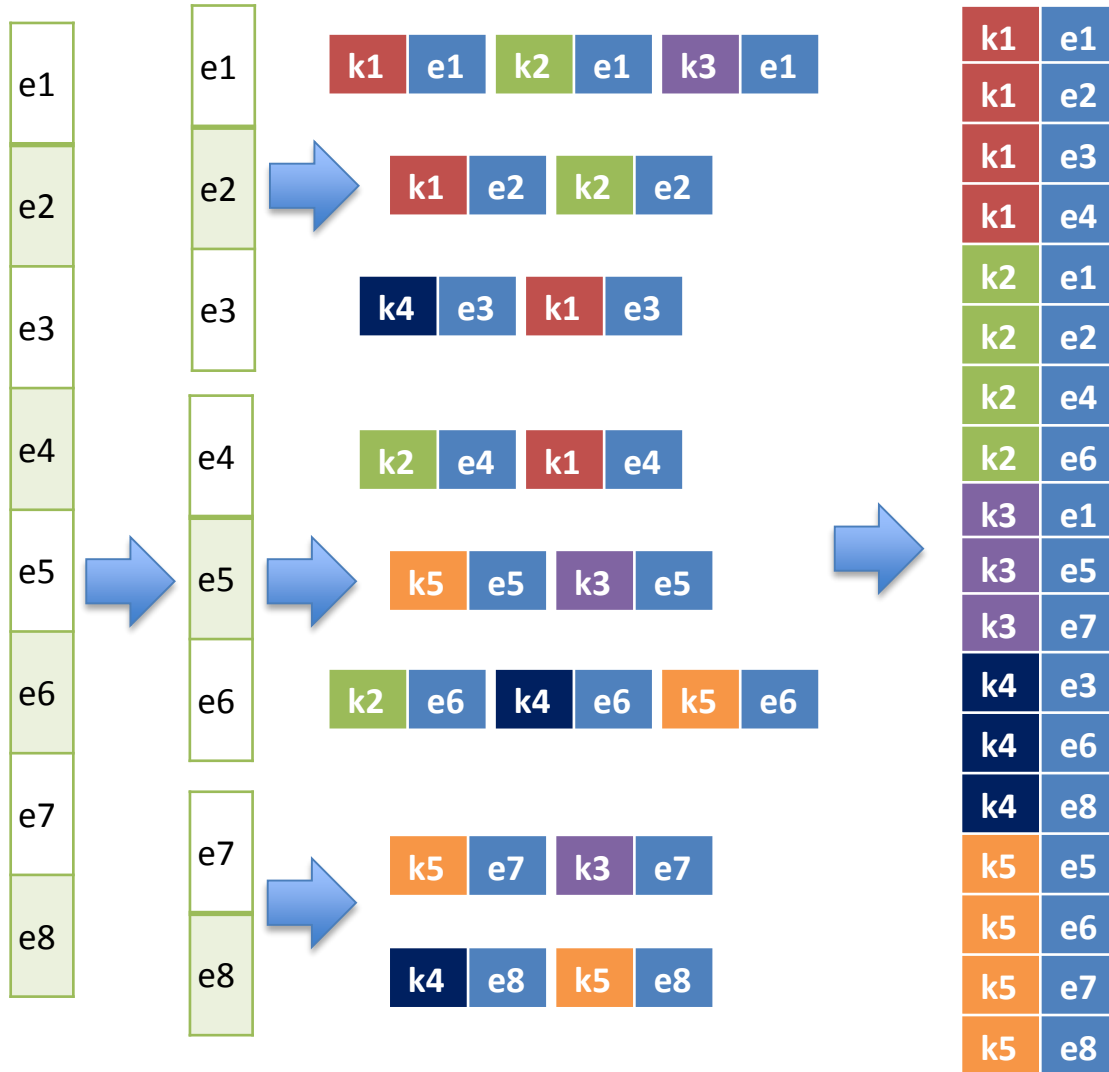
Bartholdi e2 Auguste e2

Eiffel e3 Bartholdi e3

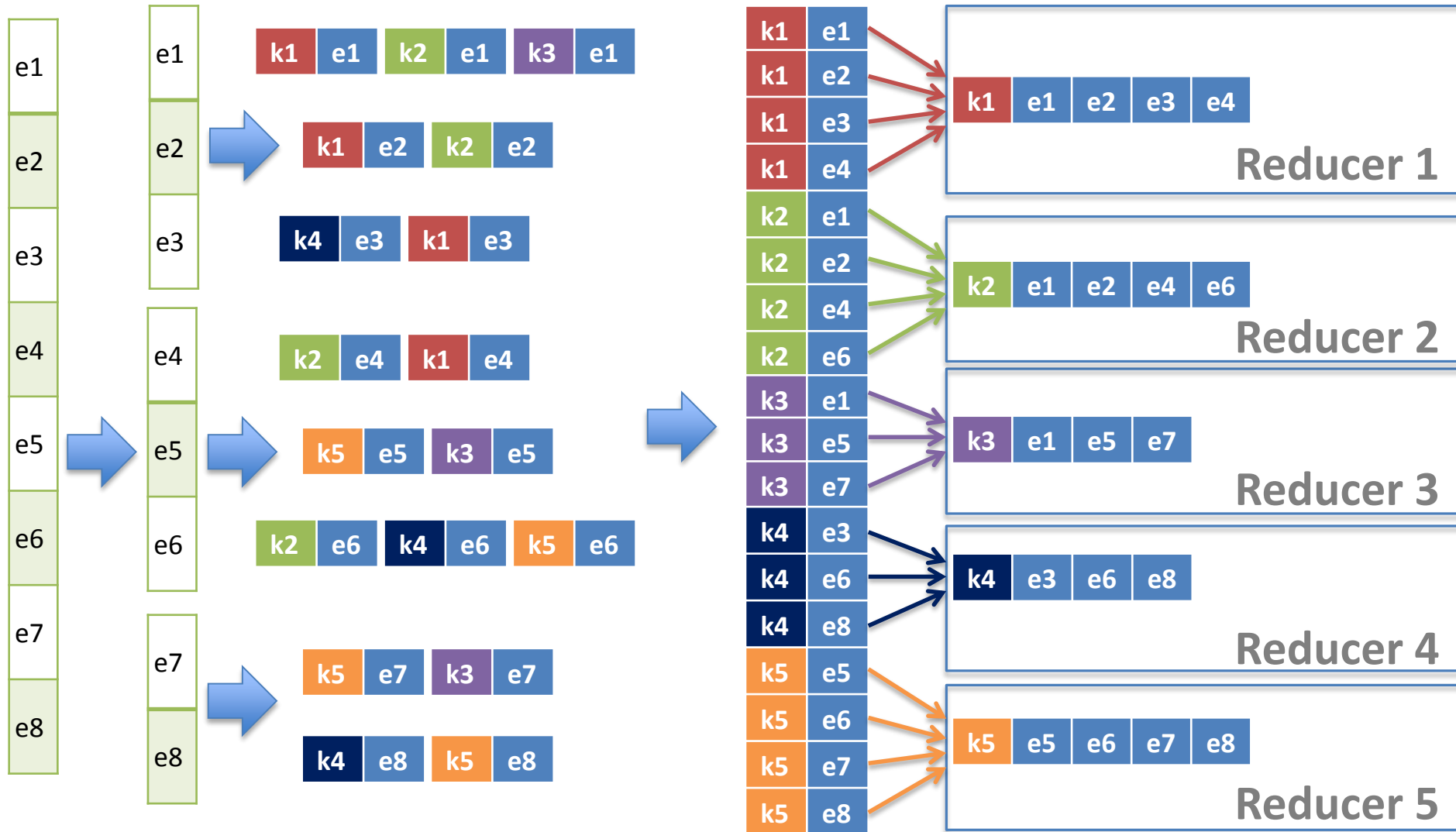
MapReduce – Mapper Output



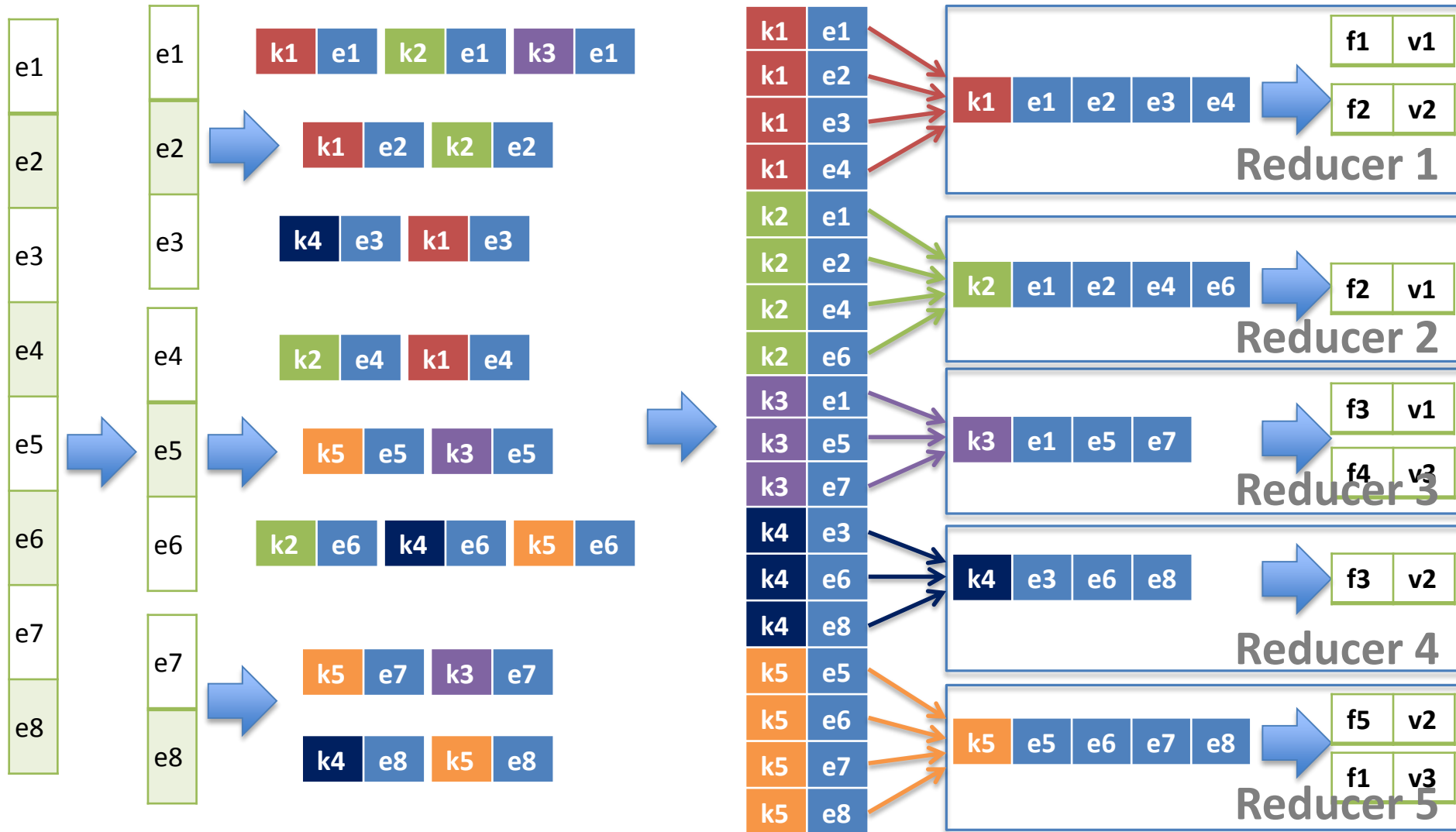
MapReduce – Shuffling & Sorting



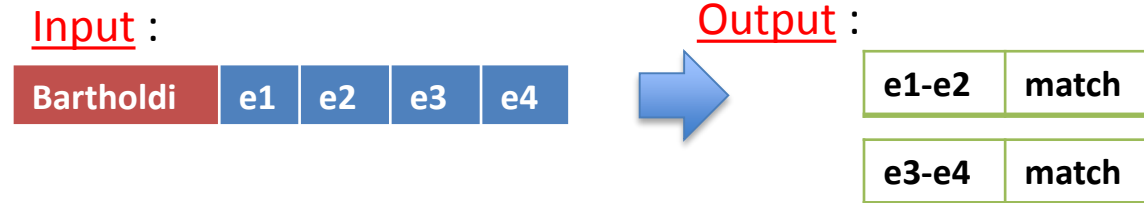
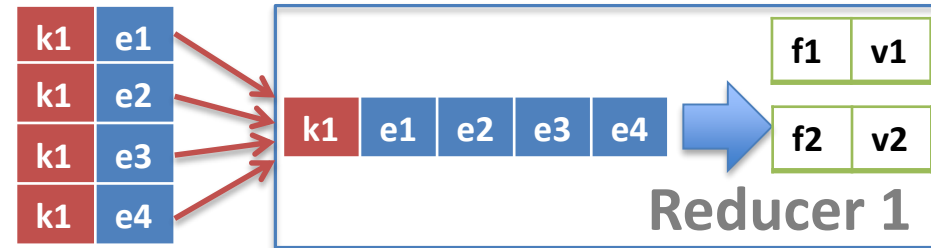
MapReduce – Merging



MapReduce – Reducer



MapReduce – Reducer Example



Dedoop – Standard Blocking [Kolb et al. 2012]

Dedoop performs standard blocking using MapReduce

Map function

- Input: an entity description
- Output: a (key, value) pair
 - key: the BKV of the description
 - value: the description having this BKV

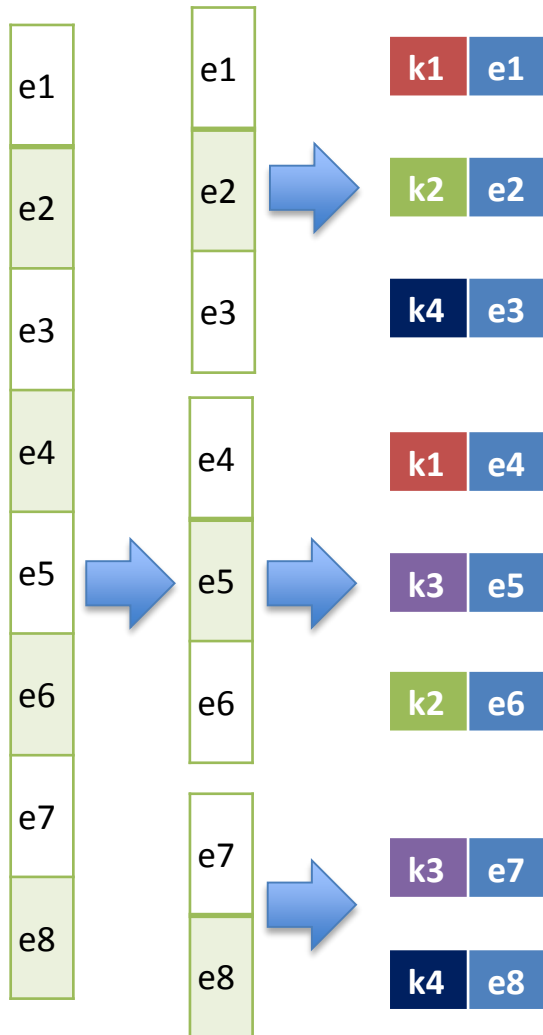
The partitioning operates on the BKVs and distributes (key, value) pairs among reduce tasks

- All entities sharing the same BKV are assigned to the same reduce task

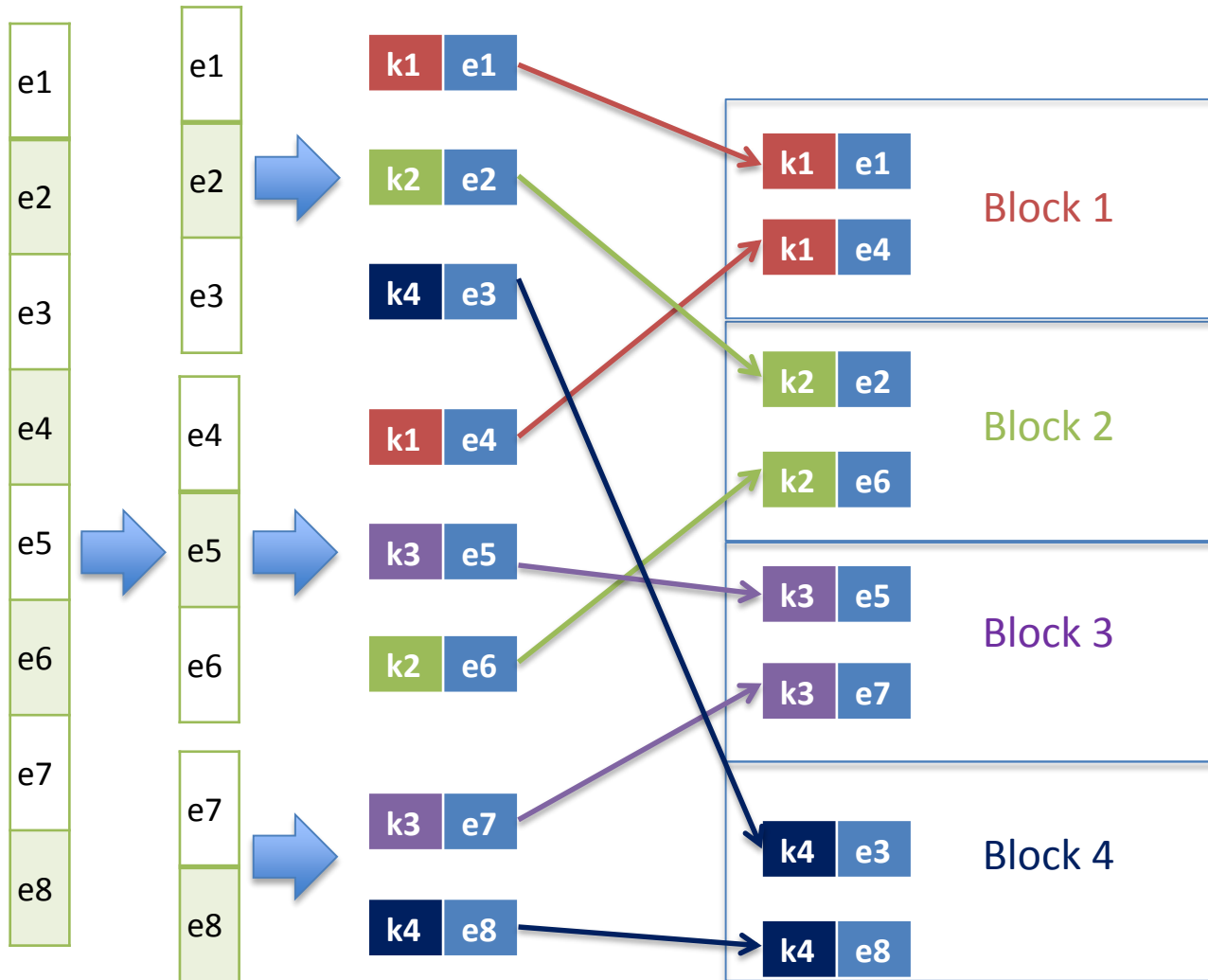
Reduce function: Computes in each block the similarities between all description pairs within the block

- Input: A BKV along with descriptions with this BKV
- Output: (key, value) pairs
 - key: a pair of descriptions
 - value: match/non-match

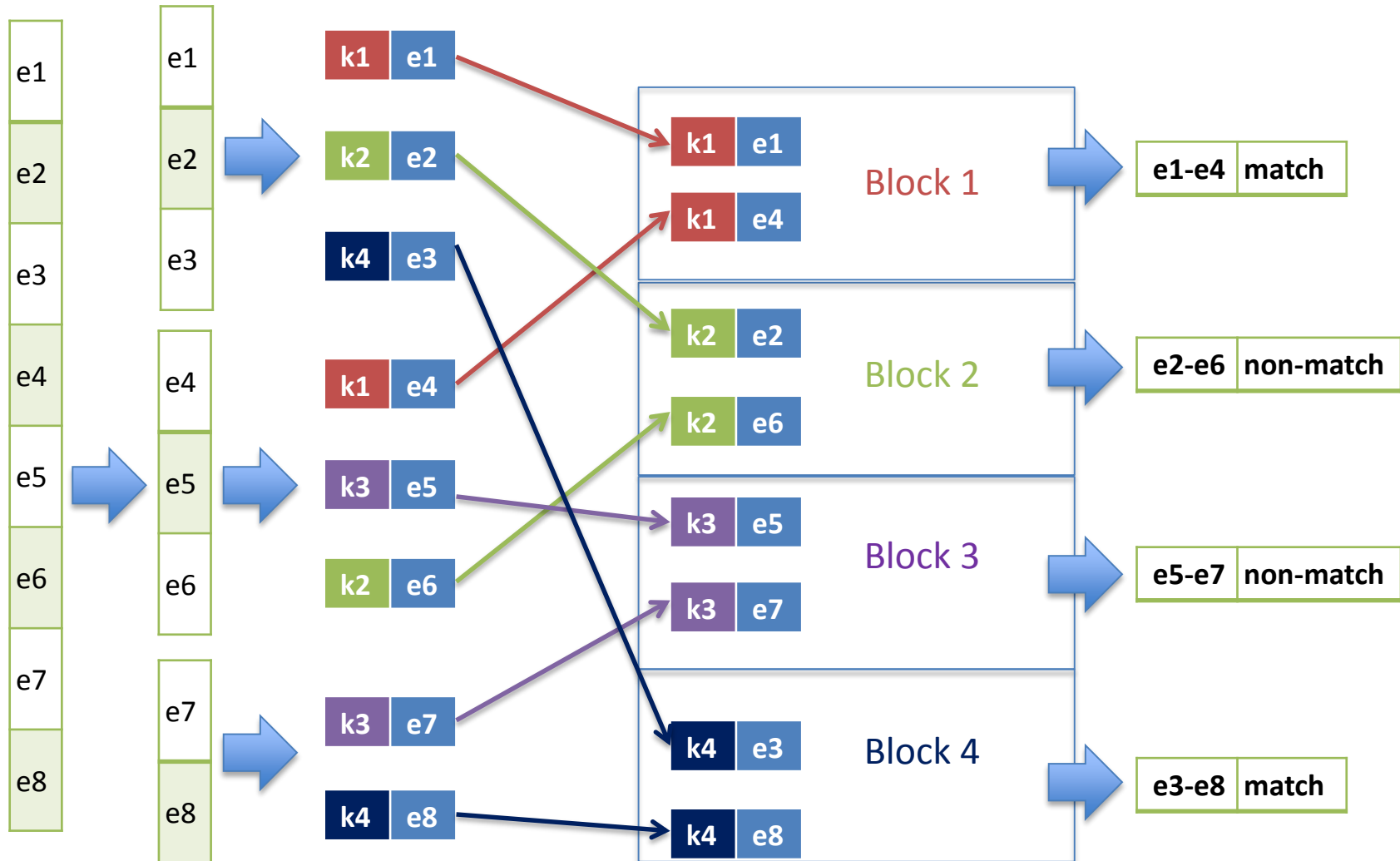
Dedoop – Mapper: BKVs as intermediate keys



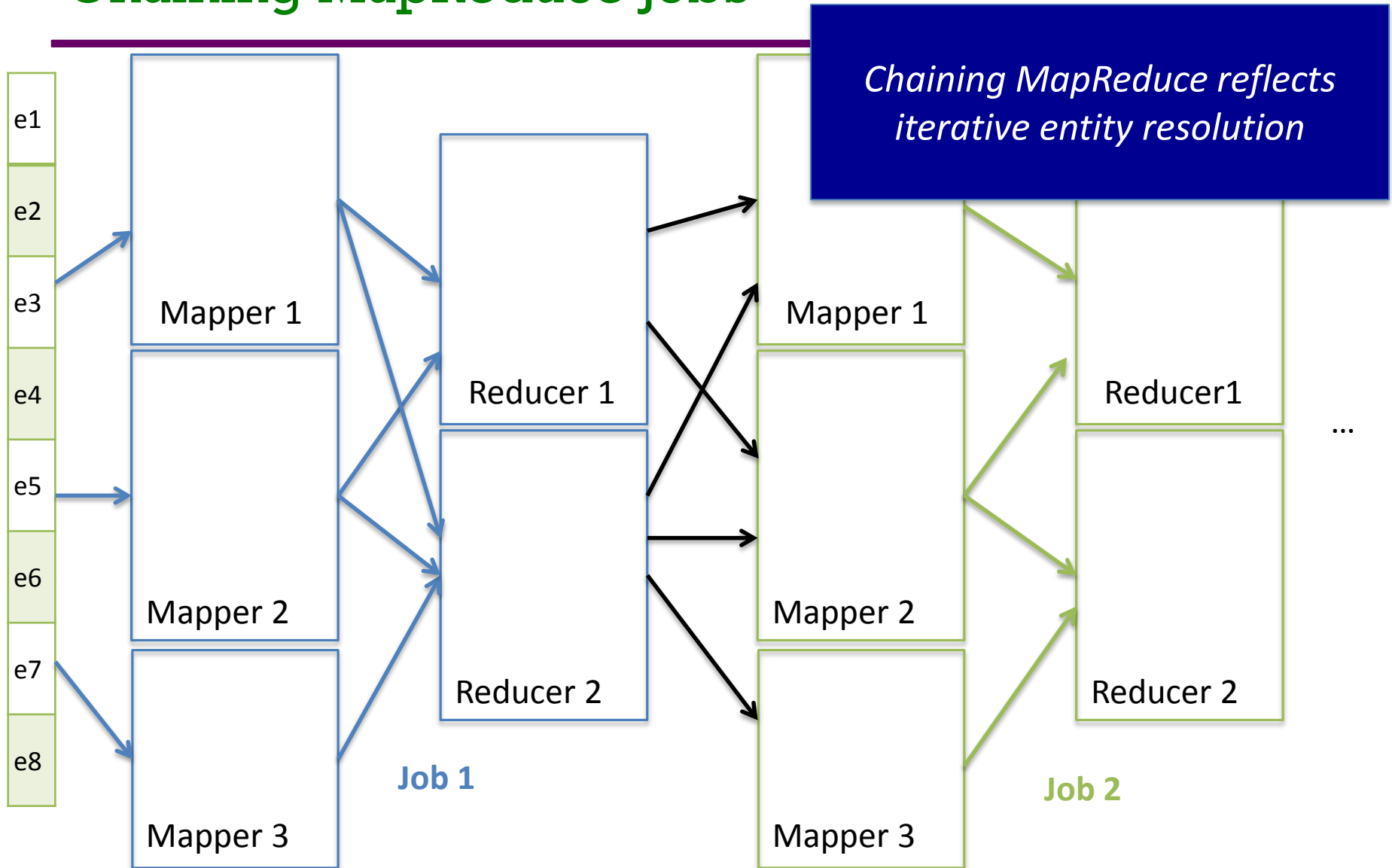
Dedoop – Mappers: Build Blocks



Dedup – Reducers: Compare Block Contents

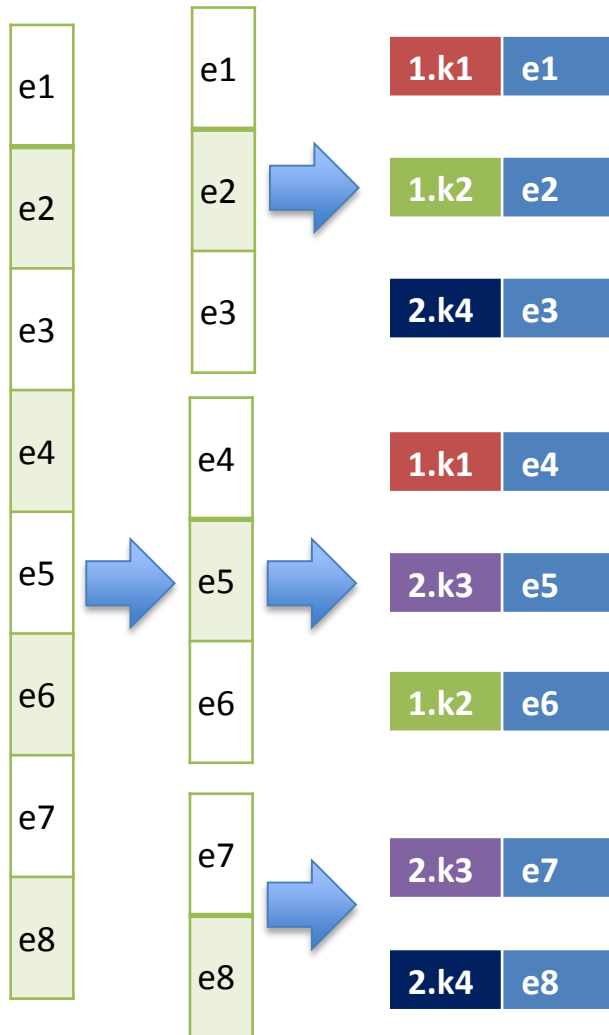


Chaining MapReduce Jobs



The output of a MapReduce Job can be the input of another

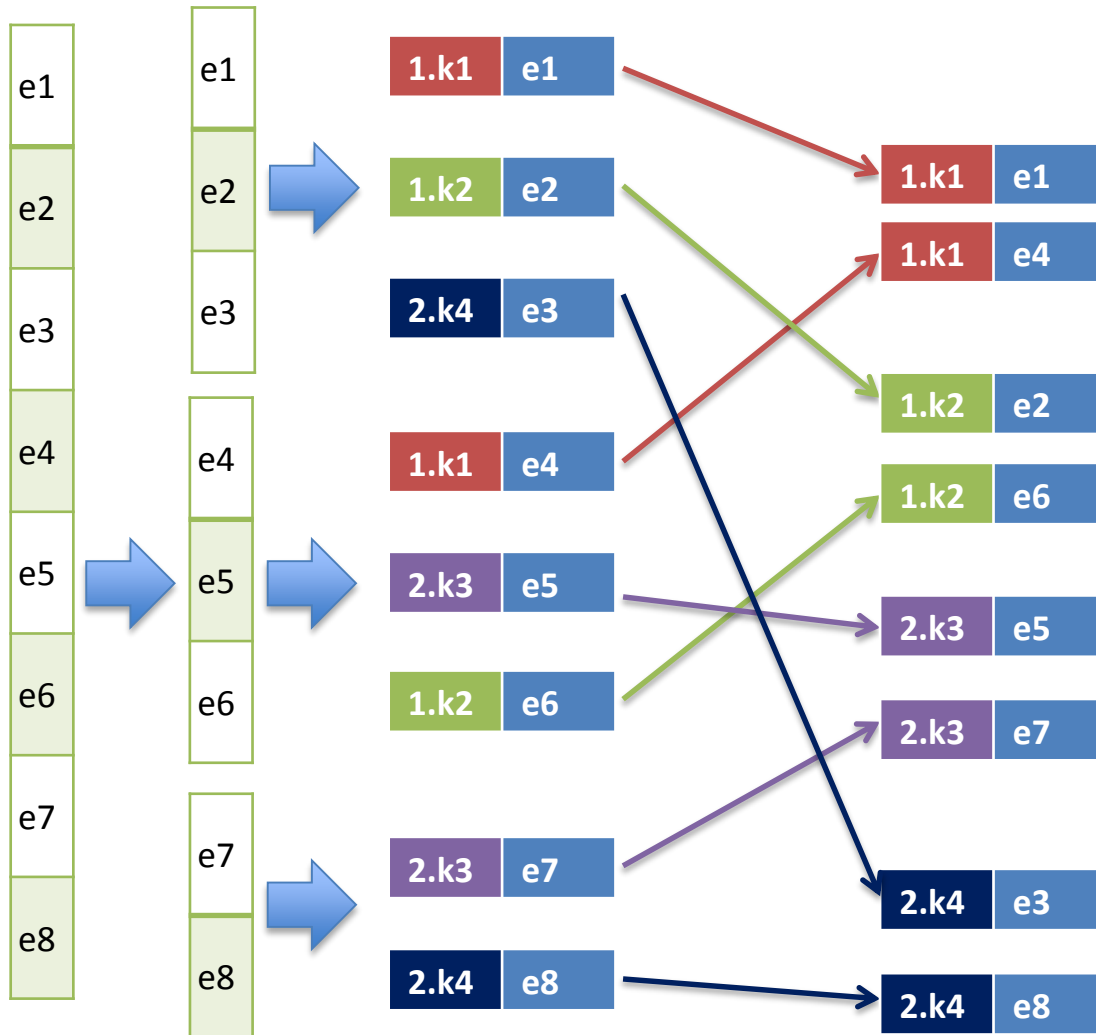
Dedoop – Sorted Neighborhood [Kolb et al. 2011]



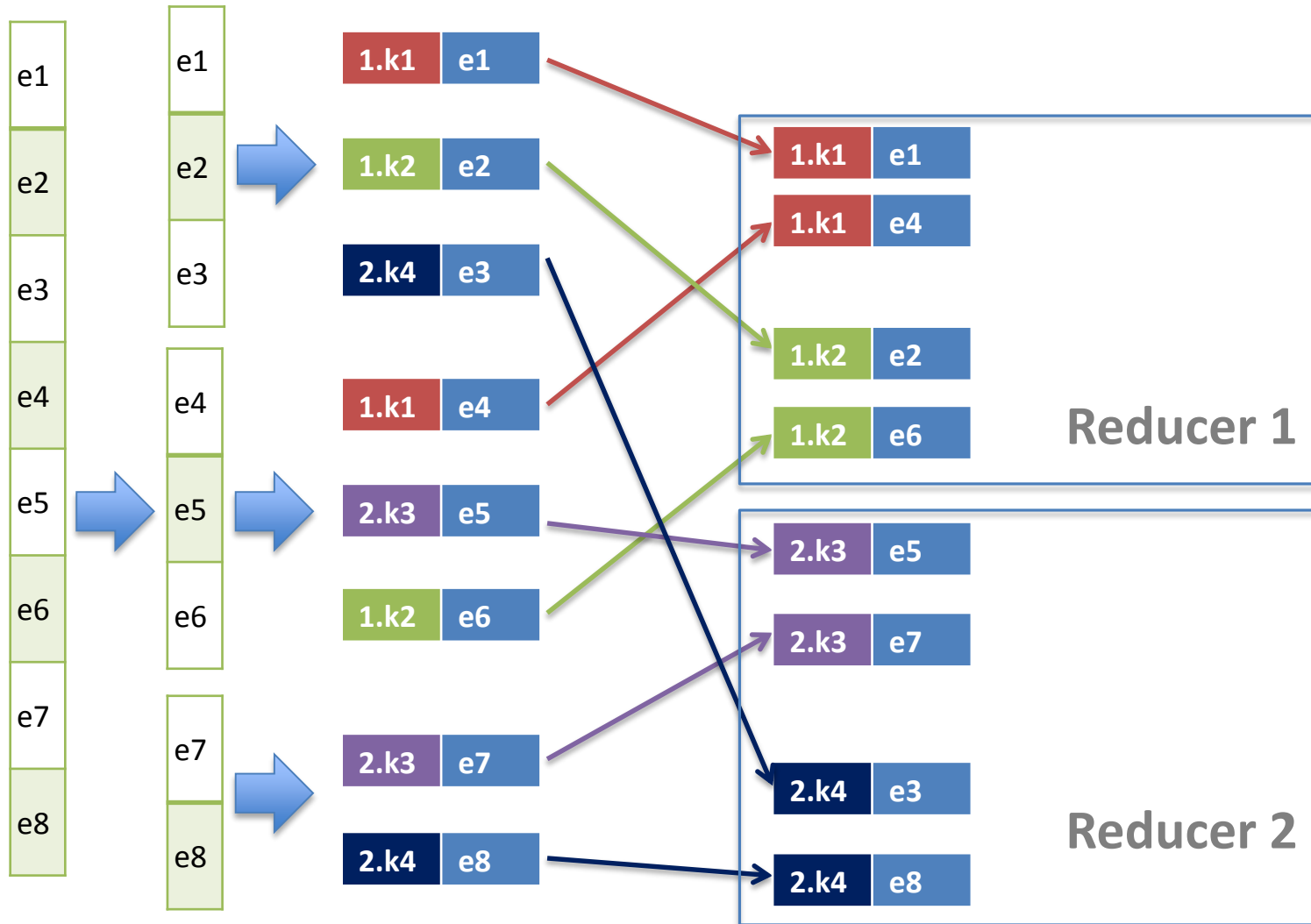
composite key = (partitionID, BKV)
partitionID(BKV) = 1, if BKV < “k3”
partitionID(BKV) = 2, else

(we know that we have two reducers available)

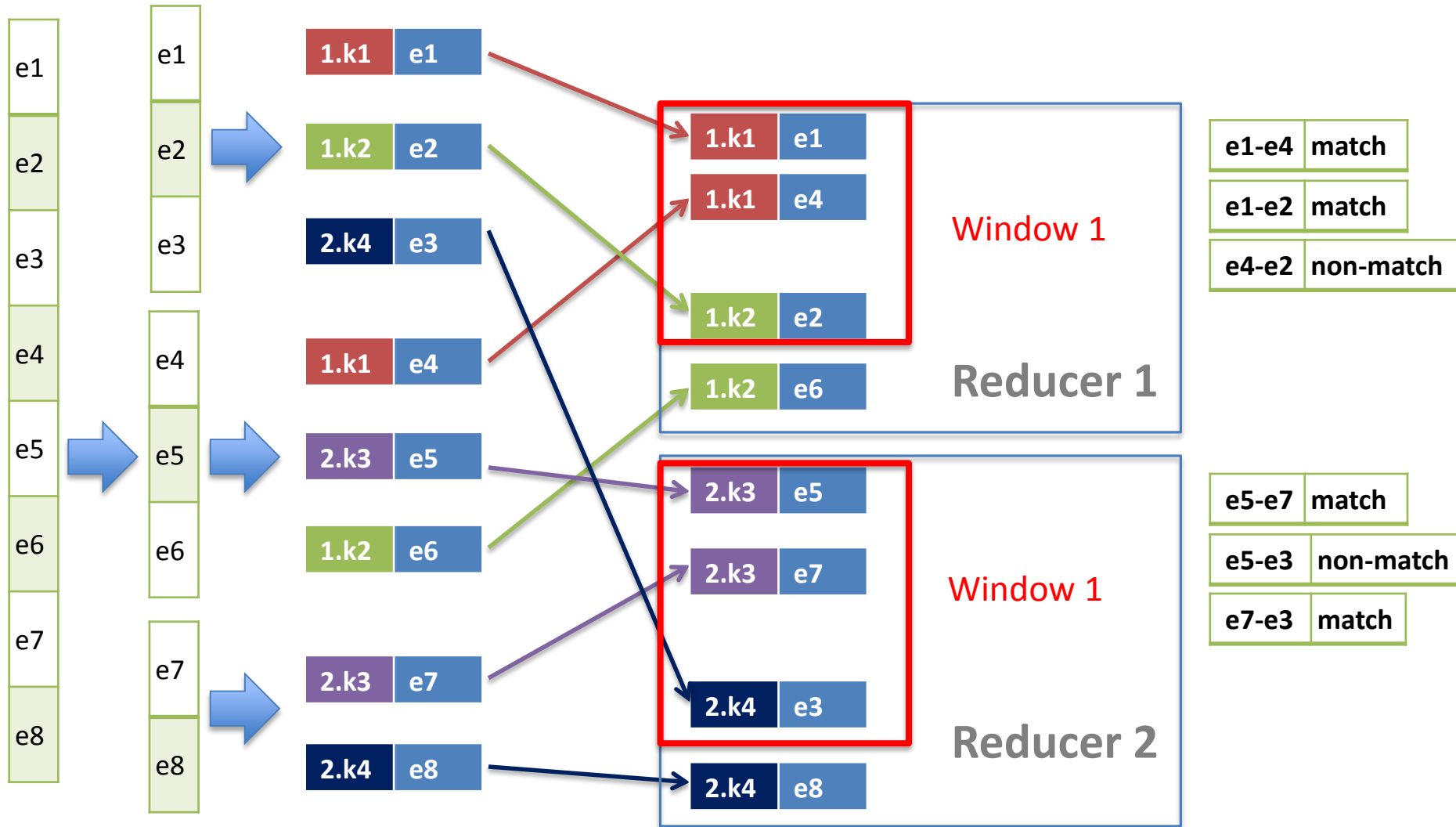
Dedoop SN: Sorting the Keys



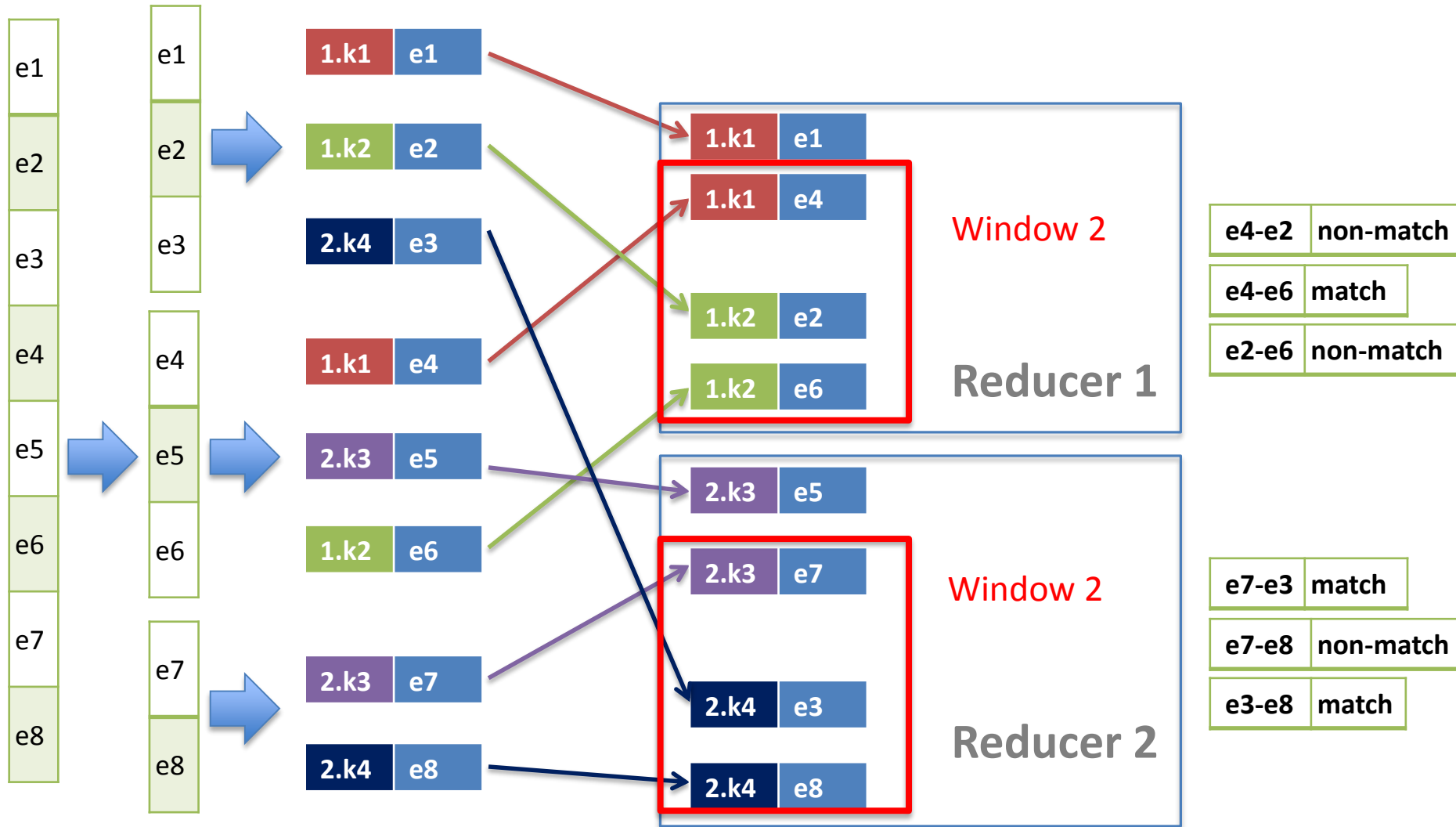
Dedoop SN: Reducers Apply the Sliding Window



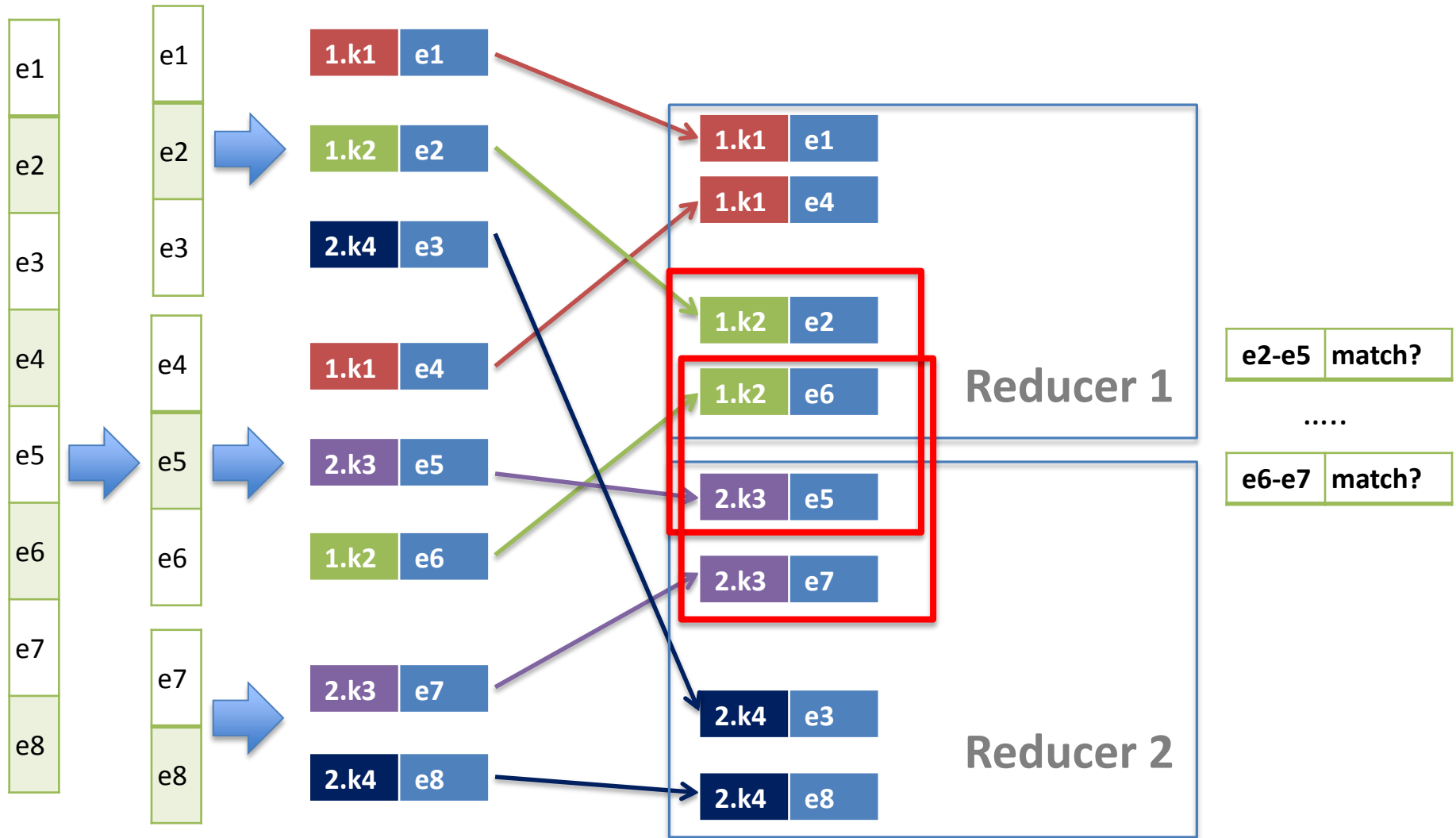
Dedoop SN: Reducers Apply the Sliding Window



Dedoop SN: Reducers Apply the Sliding Window



Dedoop SN: We Also Need To Compare The Boundary Entities



Dedoop SN: Reducers Also Output the Boundary Descriptions

Add a boundary number prefix to the output composite keys

Boundary number:

The last $w-1$ descriptions of reducer i are assigned the boundary number i

The first $w-1$ descriptions of reducer $i+1$ are also assigned the boundary number i

1.k1	e1
1.k1	e4

1.k2	e2
1.k2	e6

2.k3	e5
2.k3	e7

2.k4	e3
2.k4	e8

Reducer 1

Reducer 2

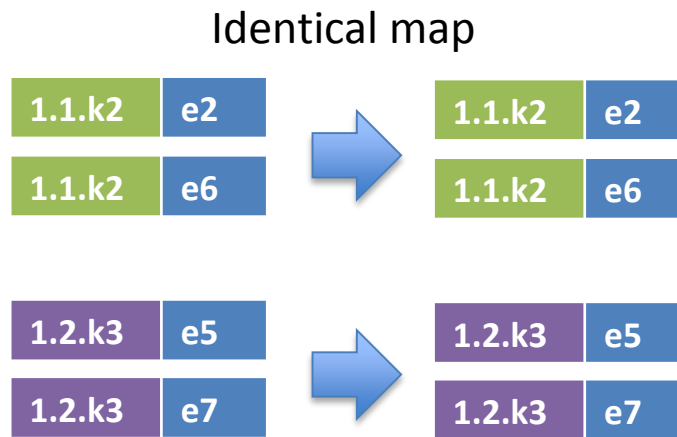
boundary
number

1.1.k2	e2
1.1.k2	e6

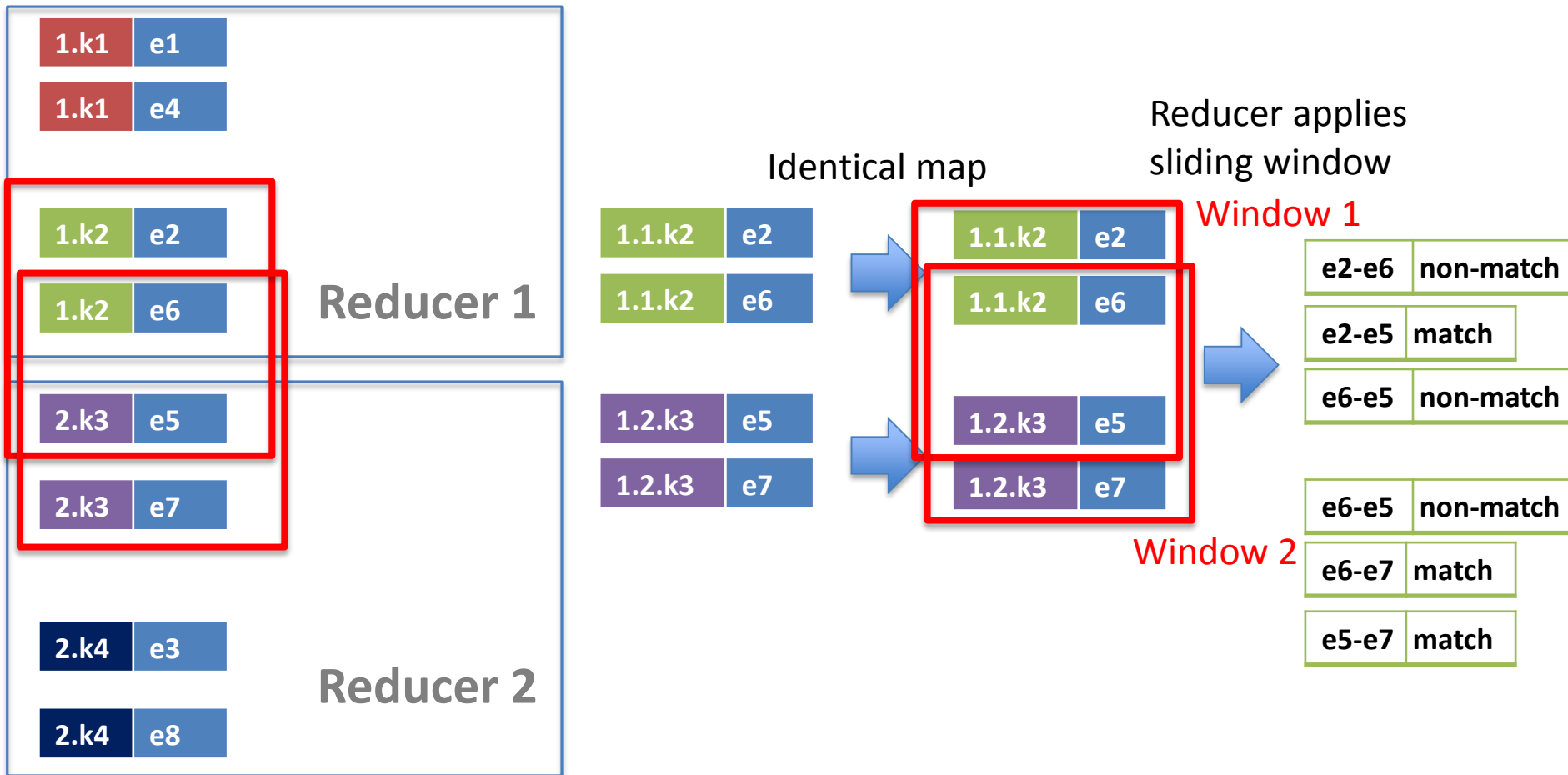
1.2.k3	e5
1.2.k3	e7

→ The actual blocking key of $e5$ is $k3$, it was assigned to reducer 2 and it is associated with boundary number 1

Dedoop SN: New MapReduce Job for the Boundary Pairs

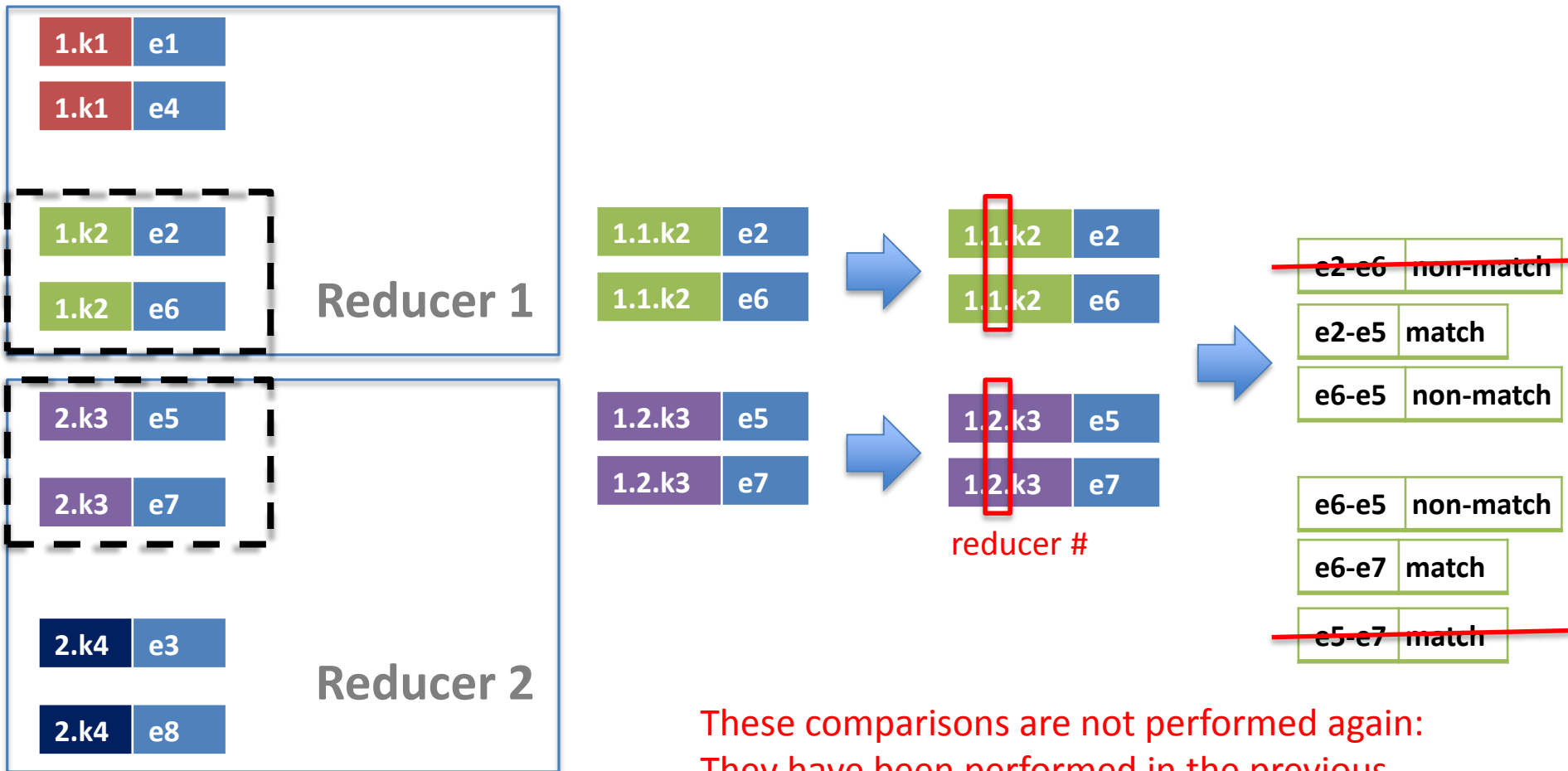


Dedoop SN: Partition by Boundary Number



Still, there are repeated comparisons

Dedoop SN: Skipping Repeated Comparisons



These comparisons are not performed again:
They have been performed in the previous
MapReduce job (they come from the same reducer)

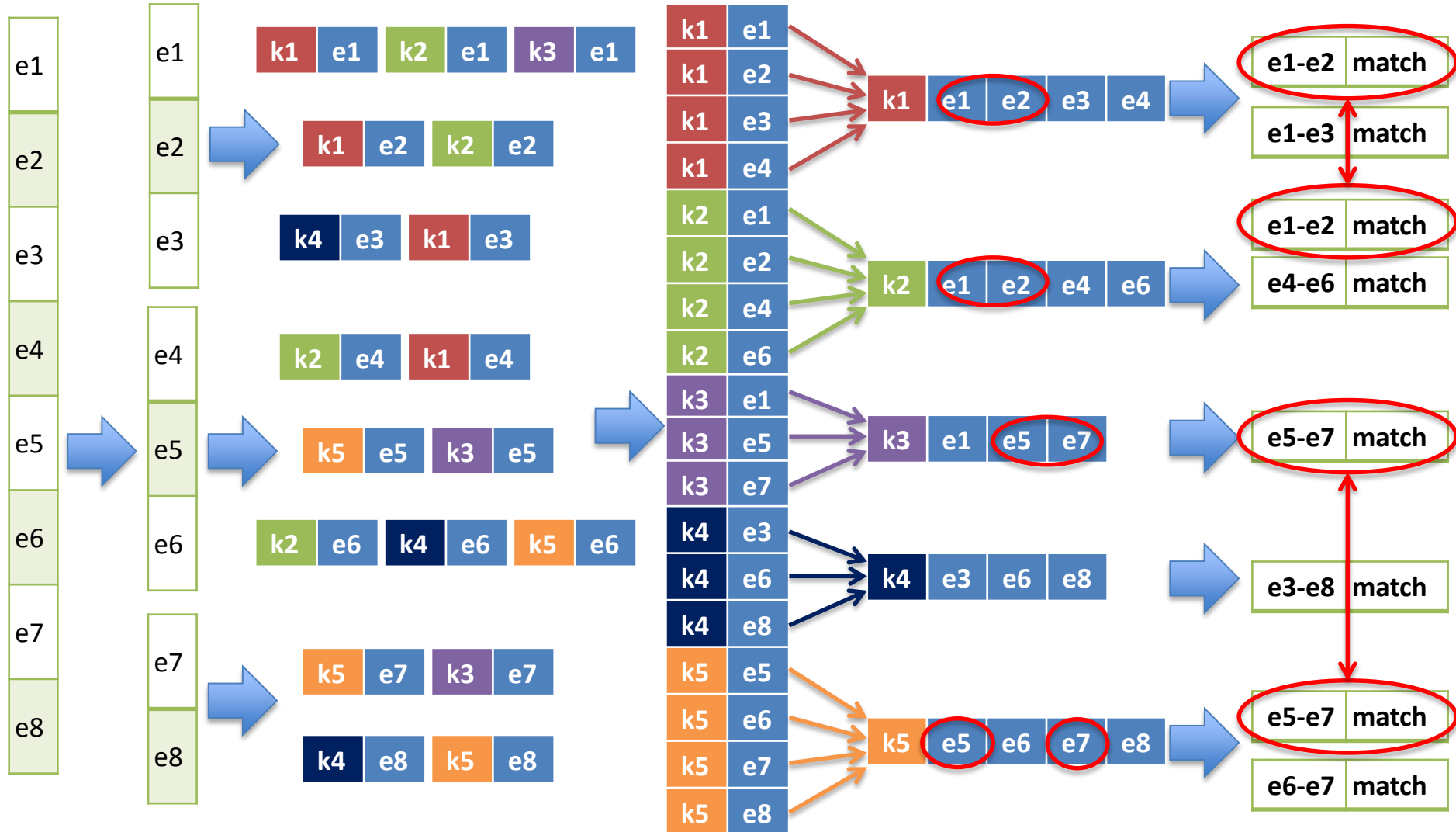
Don't match twice [Kolb et al. 2013]

Overlapping blocks lead to repeated comparisons

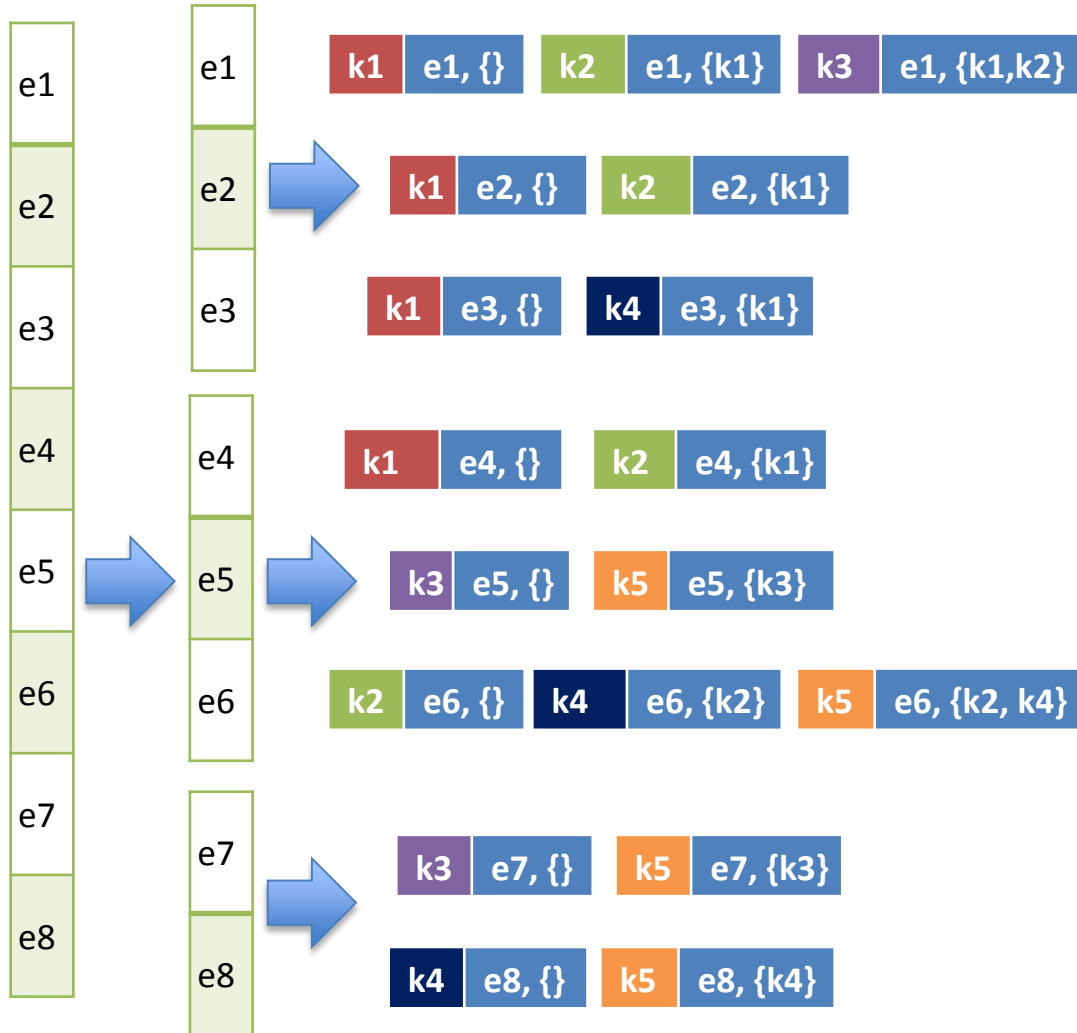
Adopt Comparison Propagation [Papadakis et al. 2012] to MapReduce:

- Descriptions need to be compared only within their least common block

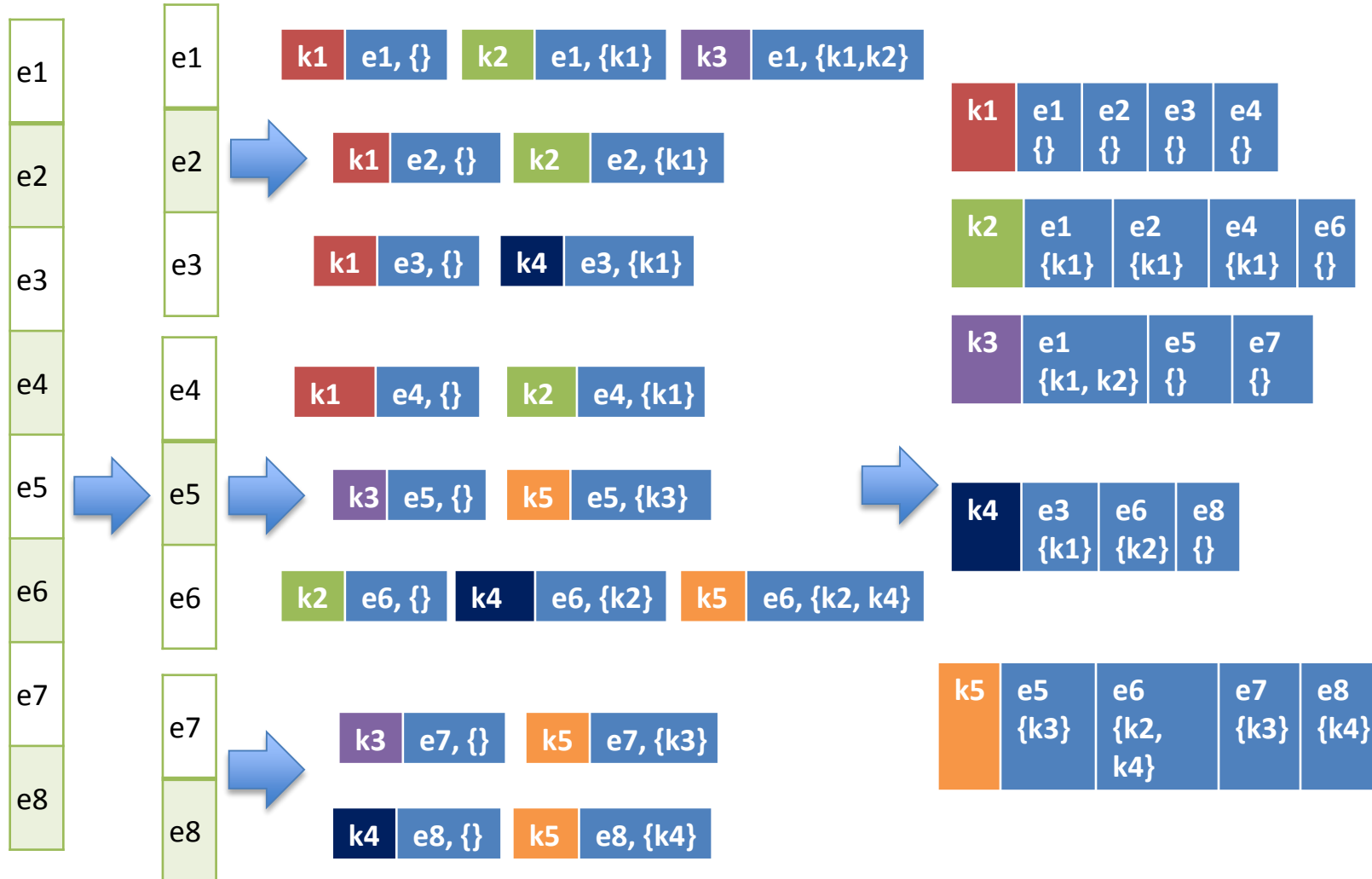
Overlapping Blocks Lead to Repeated Comparisons



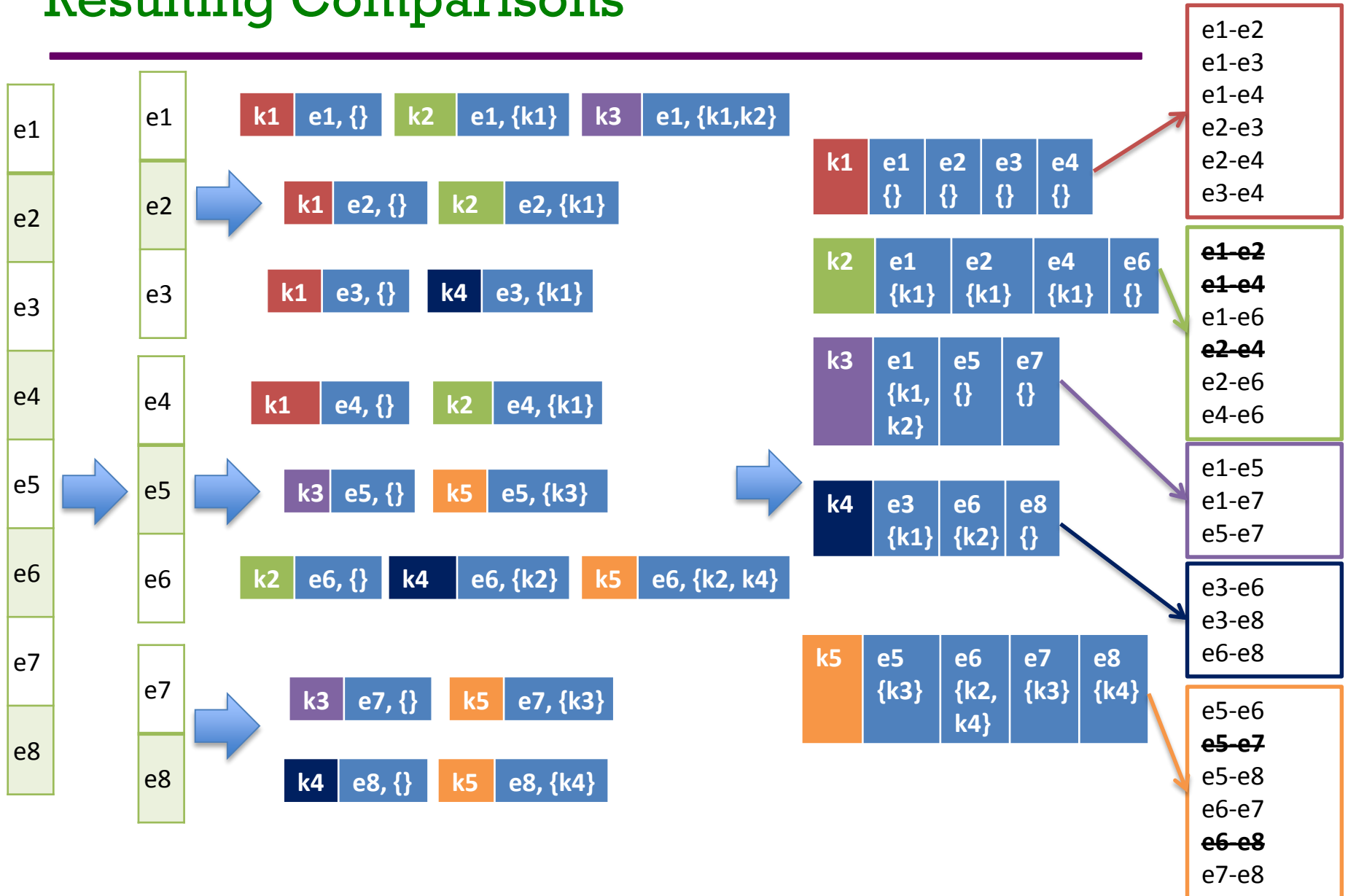
Map: Append the Subset of Smaller Keys for the Same Description



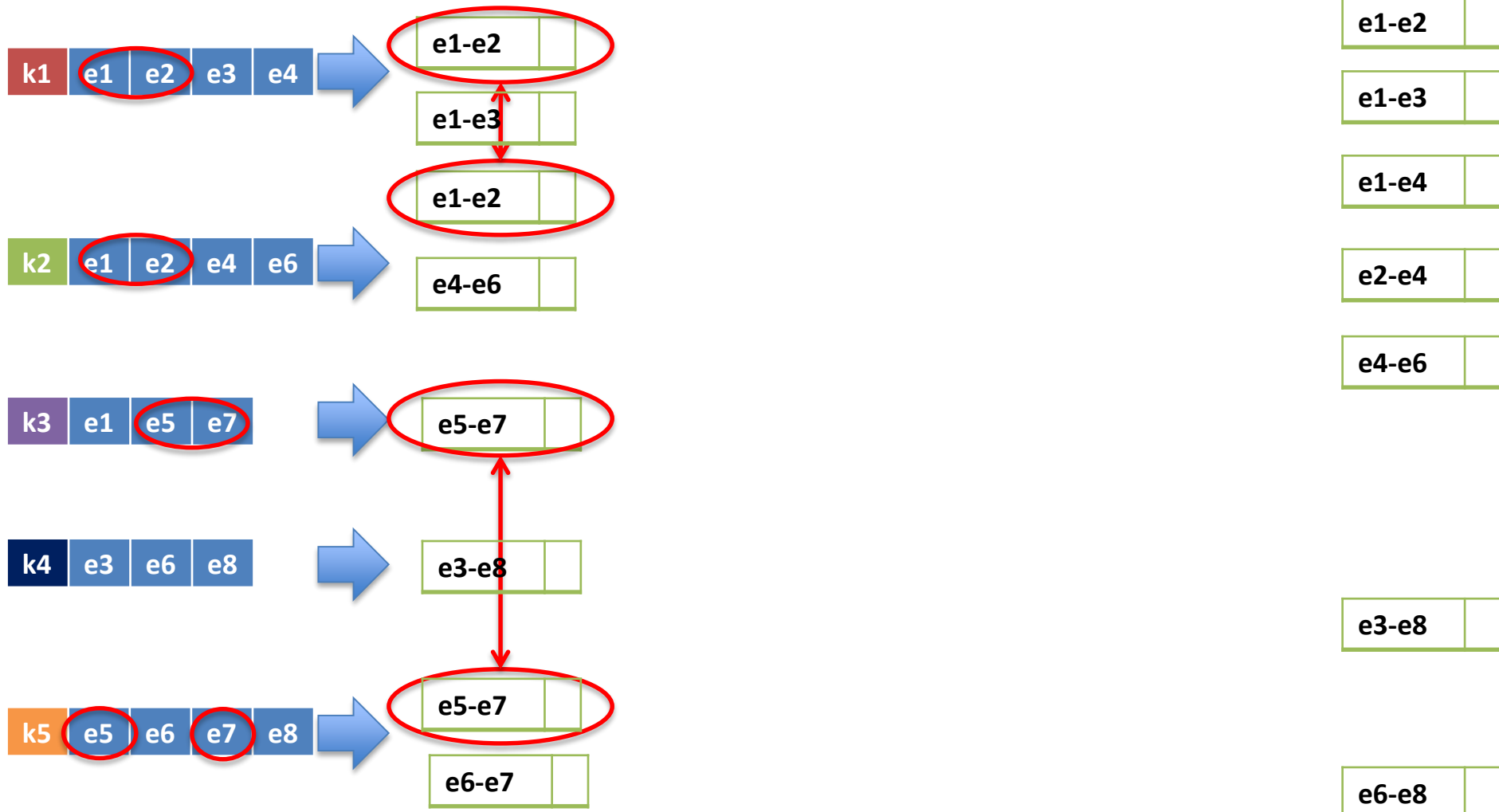
Map: Append the Subset of Smaller Keys for the Same Description



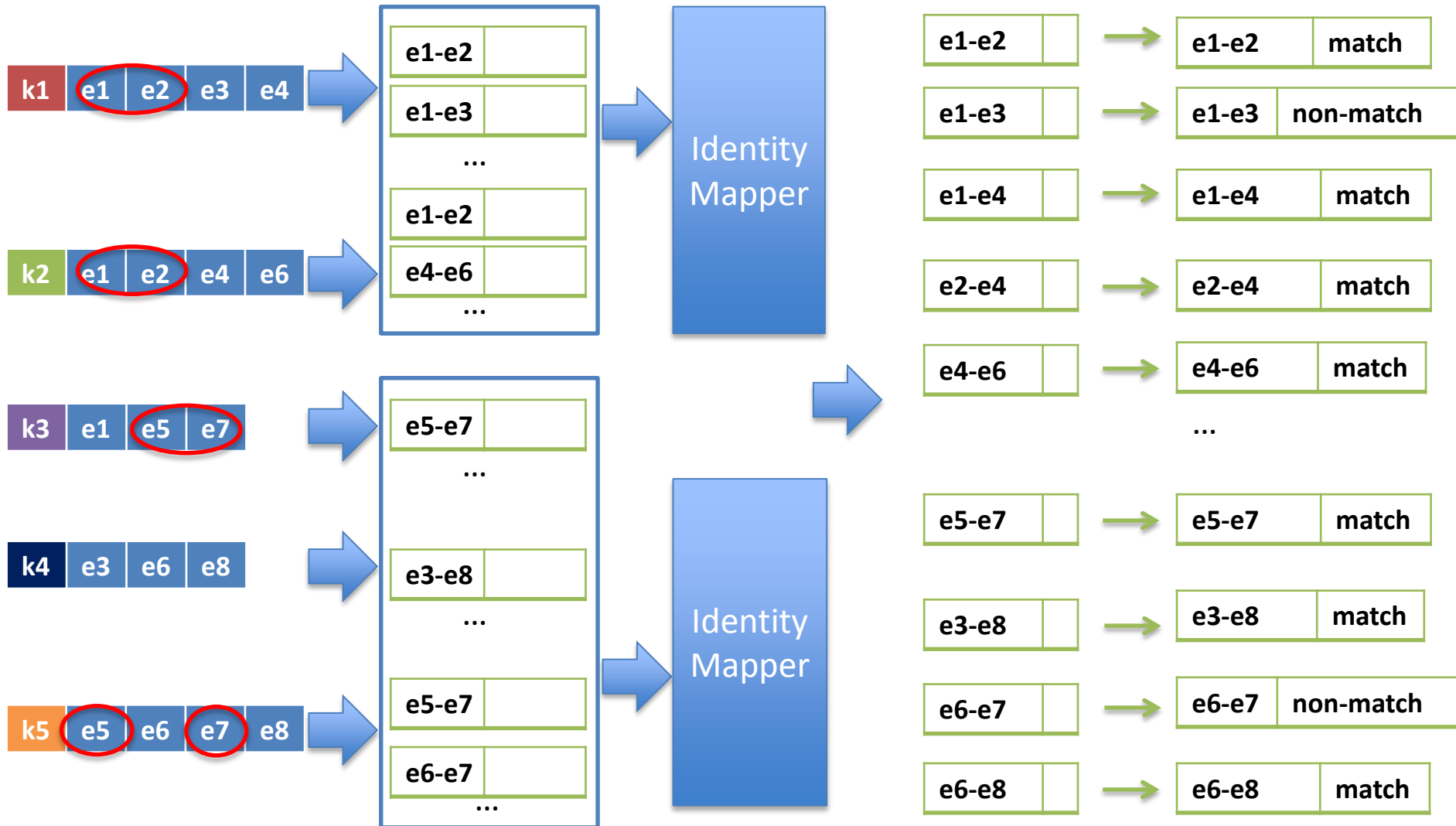
Resulting Comparisons



A More Efficient Approach [Vernica et al. 2010]



A More Efficient Approach [Vernica et al. 2010]



...becomes a new job's input

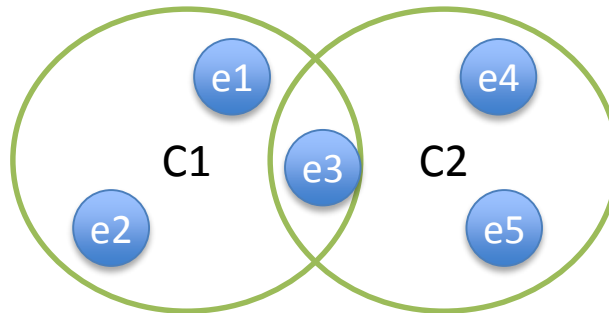
Large-Scale Collective Entity Matching

[Rastogi et al. 2011]

Assume that there is a rule $R: Match(e1, e2) \Rightarrow Match(e4, e5)$

and that we have inferred: $Match(e1, e2)$

In C2, we cannot infer $Match(e4, e5)$



map: assign each C_i to a cluster node and run entity resolution on it

reduce: bring all the new evidence for each C_i together

We should somehow inform C2 that e1 matches e2

- Then we could infer that e4 matches e5, according to rule R

Solution: **message passing**

- After matching in C1 finishes, send a message “ $Match(e1, e2)$ ”
- In the next MapReduce round, entity resolution runs with the new evidence and infers $Match(e4, e5)$

Linda [Böhm et al. 2012]

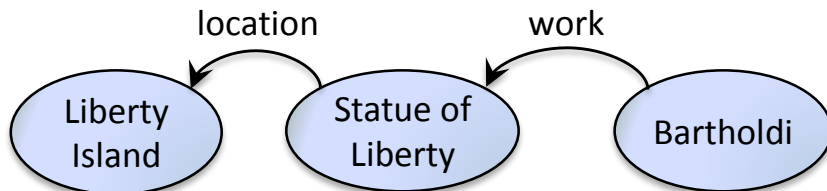
- Works on an entity graph constructed from RDF triples having URIs as subject, predicate and object
 - Literals are stored for each entity e as $L(e)$
- Matches are identified using two kinds of similarities:
 - String similarity (token-based) of their literal values $L(e)$
 - Checked once
 - Contextual similarity (based on neighbors in the entity graph)
 - Checked iteratively

Contextual Similarity

What is context?

- Let node n in an entity graph correspond to an RDF subject or object, identified by a URI
- The context $C(n)$ of n is a set of tuples (p_i, z_i, w_i) , where
 - z_i is a neighboring node of n
 - p_i is the predicate associated with an edge connecting n with z_i
 - w_i is a numeric weight (how discriminative this information is)

That is, the context of n includes objects z_i of triples with n as subject and subjects z_i of triples with n as object



$C(\text{Statue of liberty}) =$
 $\{(\text{location}, \text{Liberty Island}, w_1),$
 $(\text{is work of}, \text{Bartholdi}, w_2)\}$

Contextual Similarity

The contextual similarity of nodes n and m is:

$\text{context_sim}(n, m) =$

- $\sum_{(p_i, z_i, w_i) \in C(n)} \max_{(p_j, z_j, w_j) \in C(m)} w_i \cdot x_{z_i, z_j} \cdot \text{sim}(p_i, p_j), \text{if } |C(n)| \leq |C(m)|$
- $\sum_{(p_j, z_j, w_j) \in C(m)} \max_{(p_i, z_i, w_i) \in C(n)} w_j \cdot x_{z_i, z_j} \cdot \text{sim}(p_i, p_j), \text{else}$

where

$x_{n,m}$ is 1, if n, m are identified as matches, and 0, else

$\text{sim}(p_i, p_j)$ is the string similarity of the predicates of n, m

Intuitively, the contextual similarity finds matching neighbors and sums up their similarity values

Contextual Similarity

Overall similarity: combine `sim` and `context_sim`

The similarity score for descriptions `n` and `m` is:

$$\text{sim}(n, m) + \beta \cdot \text{context_sim}(\mathbf{C}(n), \mathbf{C}(m)) - \theta$$

β controls the contextual influence

θ is used for re-normalization to values around 0

- positive scores reflect likely mappings
- negative scores imply dissimilarities

Experiments have shown $\beta = 1$ to perform well

LINDA Algorithm

Two square matrices ($|E| \times |E|$) are used:

- X captures the identified matches (binary values)
- Y captures the pair-wise similarities (real values)
 - Initialization: common neighbors and string similarity of literals
 - Updates: Use the new identified matches of X

Until a priority queue of pairs (extracted from Y) becomes empty:

- Get the pair (e_i, e_j) with the highest similarity
 - (e_i, e_j) match by default!
 - Update X: matches of e_i are also matches of e_j
- Update the queue wrt. the new matches

LINDA – Distributed Entity Resolution Using MapReduce

Distribute across a cluster the input entity graph

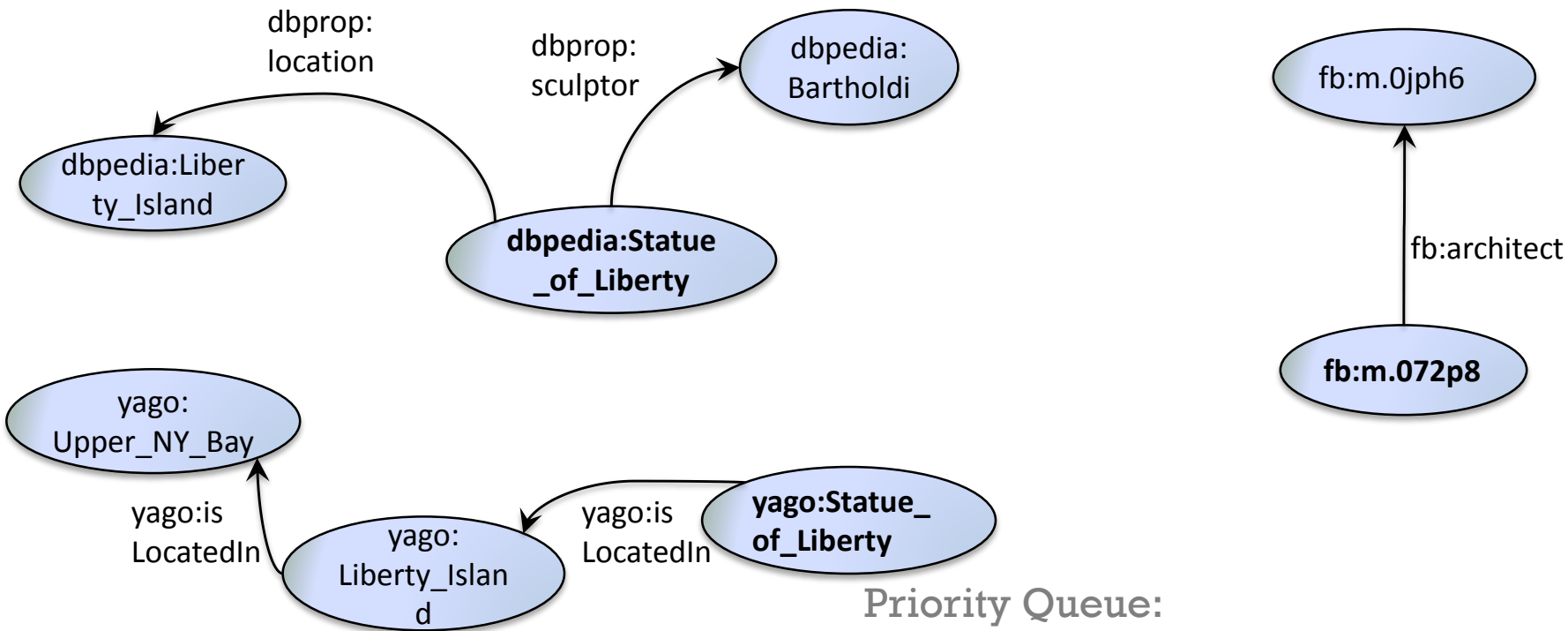
- A node i holds a portion Q_i of the priority queue and the respective part G_i of the graph

Map phase

- Mapper i reads Q_i and forwards messages to reducers for similarities re-computations
 - Matrix X of identified matches is updated

Reduce phase

- Similarities re-computations (Matrix Y)
- Updates on priority queues



Priority Queue:

(dbpedia:Statue_of_Liberty, yago:Statue_of_Liberty)
(dbpedia:Statue_of_Liberty, yago:Liberty_Island)
(dbpedia:Liberty_Island, yago:Upper_NY_Bay)
(dbpedia:Liberty_Island, yago:Liberty_Island)
(dbpedia:Liberty_Island, yago:Statue_of_Liberty)
(dbpedia:Bartholdi, fb:m.0jph6)
(dbpedia:Bartholdi, yago:Statue_of_Liberty)
(dbpedia:Bartholdi, fb:m.072p8)

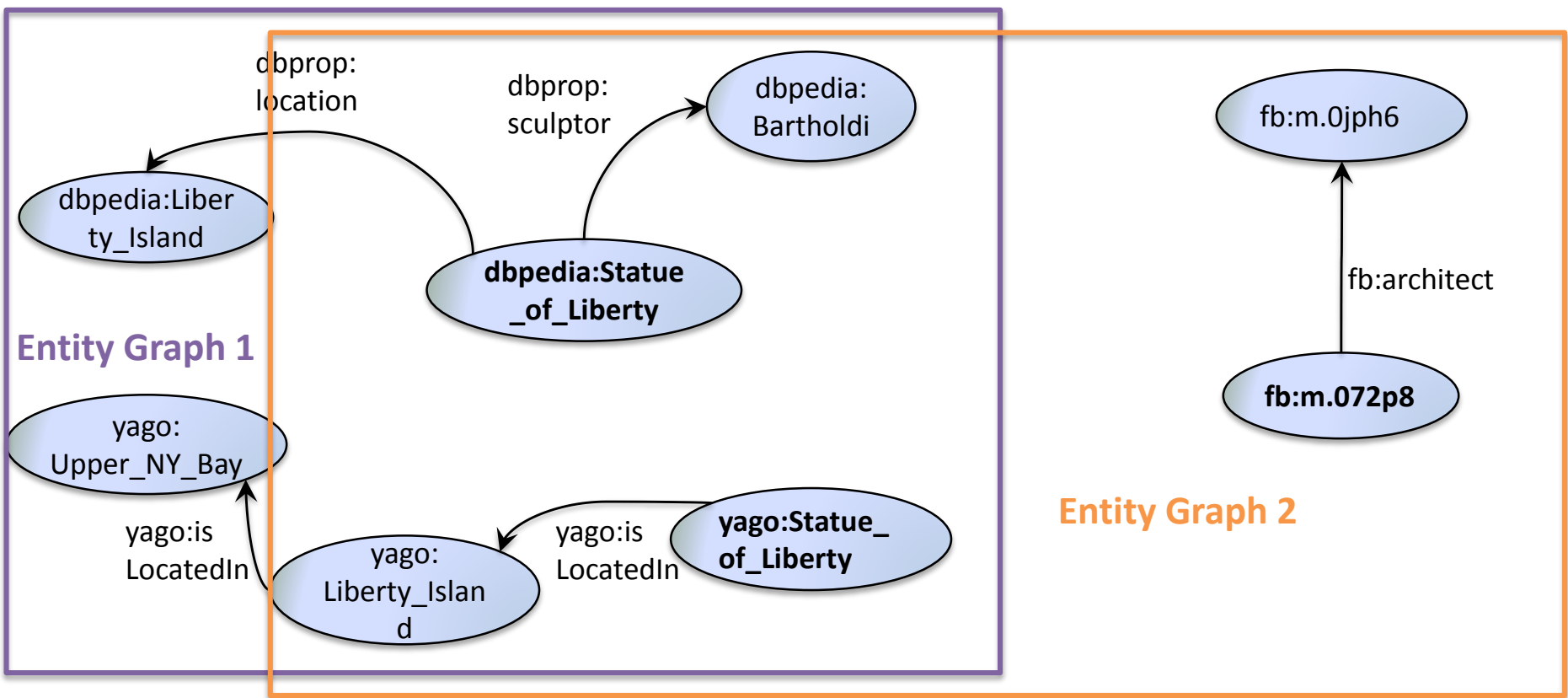
Priority Queue 1 (machine 1):

(dbpedia:Statue_of_Liberty, yago:Statue_of_Liberty)
(dbpedia:Statue_of_Liberty, yago:Liberty_Island)
(dbpedia:Liberty_Island, yago:Upper_NY_Bay)
(dbpedia:Liberty_Island, yago:Liberty_Island)
(dbpedia:Liberty_Island, yago:Statue_of_Liberty)

Priority Queue 2 (machine 2):

(dbpedia:Bartholdi, fb:m.0jph6)
(dbpedia:Bartholdi, yago:Statue_of_Liberty)
(dbpedia:Bartholdi, fb:m.072p8)

The priority queue is partitioned and partitions are sent to the MapReduce nodes



Priority Queue 1 (machine 1):

Priority Queue 2 (machine 2):

(dbpedia:Statue_of_Liberty, yago:Statue_of_Liberty)

(dbpedia:Statue_of_Liberty, yago:Liberty_Island)

(dbpedia:Liberty_Island, yago:Upper_NY_Bay)

(dbpedia:Liberty_Island, yago:Liberty_Island)

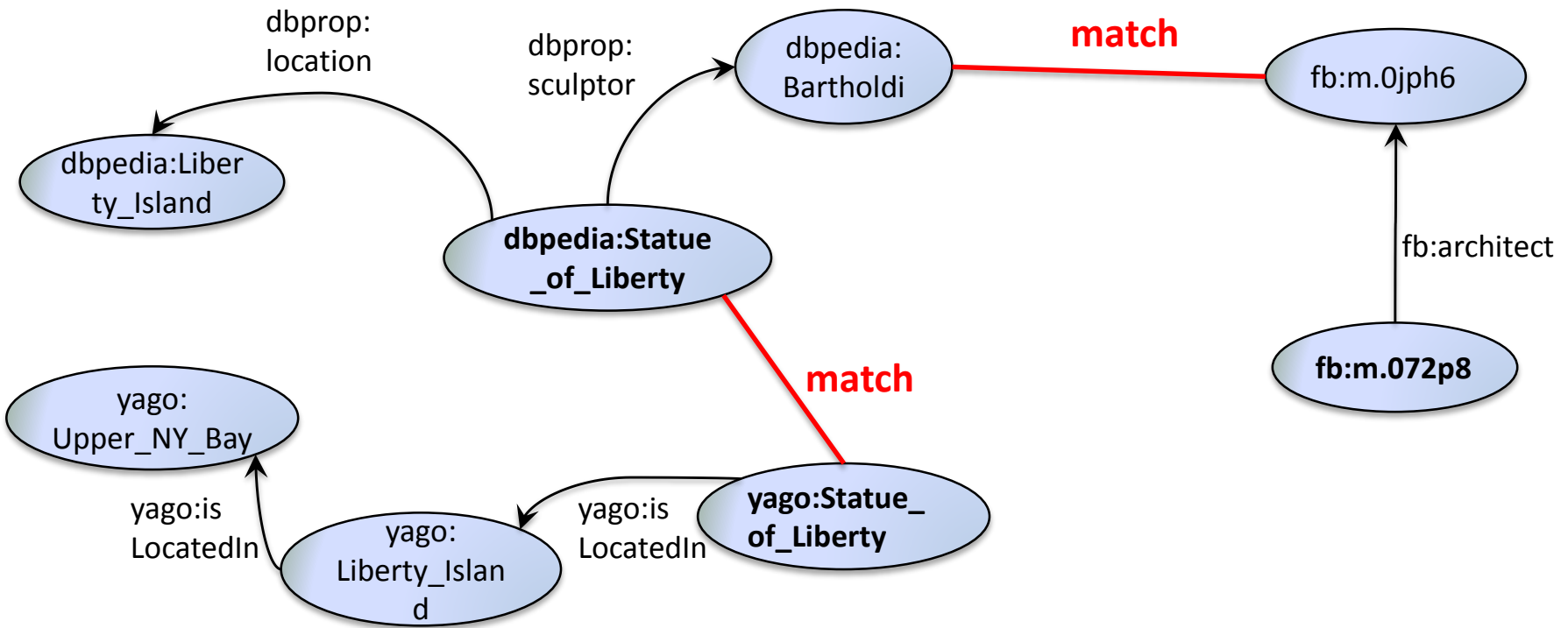
(dbpedia:Liberty_Island, yago:Statue_of_Liberty)

(dbpedia:Bartholdi, fb:m.0jph6)

(dbpedia:Bartholdi, yago:Statue_of_Liberty)

(dbpedia:Bartholdi, fb:m.072p8)

The priority queue is partitioned and partitions are sent to the MapReduce nodes



Priority Queue 1:

(dbpedia:Statue_of_Liberty, yago:Statue_of_Liberty)

(dbpedia:Statue_of_Liberty, yago:Liberty_Island)

(dbpedia:Liberty_Island, yago:Upper_NY_Bay)

(dbpedia:Liberty_Island, yago:Liberty_Island)

(dbpedia:Liberty_Island, yago:Statue_of_Liberty)

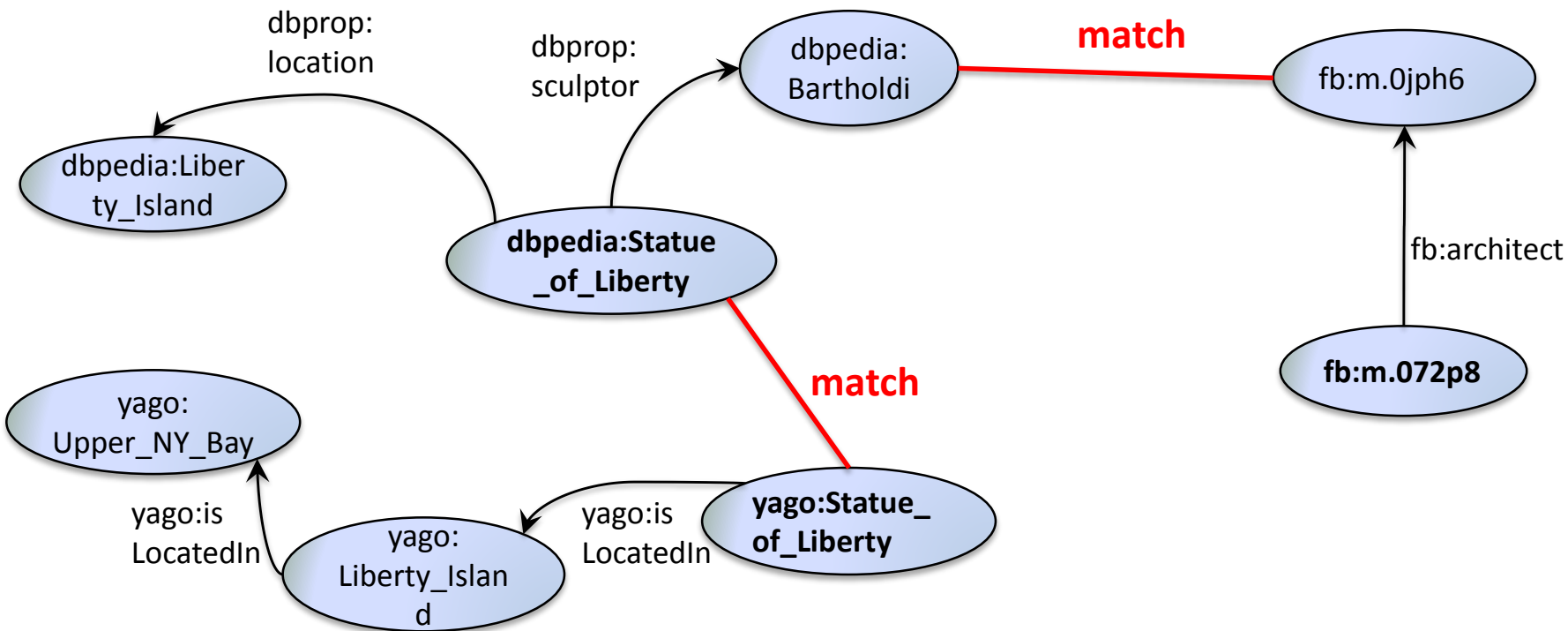
Priority Queue 2:

(dbpedia:Bartholdi, fb:m.0jph6)

(dbpedia:Bartholdi, yago:Statue_of_Liberty)

(dbpedia:Bartholdi, fb:m.072p8)

The head of each queue is a match by default
This triggers update messages



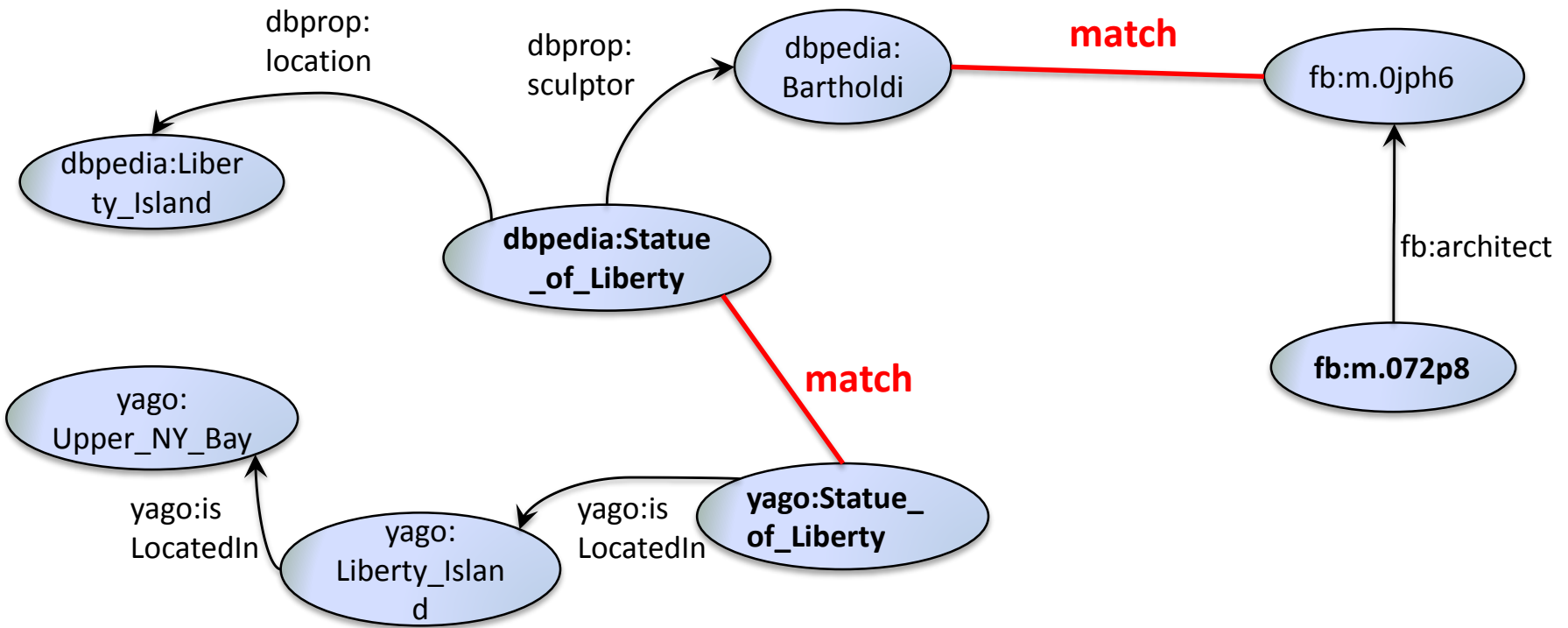
Priority Queue 1:

(dbpedia:Statue_of_Liberty, yago:Statue_of_Liberty)
(dbpedia:Statue_of_Liberty, yago:Liberty_Island)
(dbpedia:Liberty_Island, yago:Upper_NY_Bay)
(dbpedia:Liberty_Island, yago:Liberty_Island)
(dbpedia:Liberty_Island, yago:Statue_of_Liberty)

Priority Queue 2:

(dbpedia:Bartholdi, fb:m.0jph6)
(dbpedia:Bartholdi, yago:Statue_of_Liberty)
(dbpedia:Bartholdi, fb:m.072p8)

Dequeue these pairs, as each entity can be mapped to at most one entity per data source



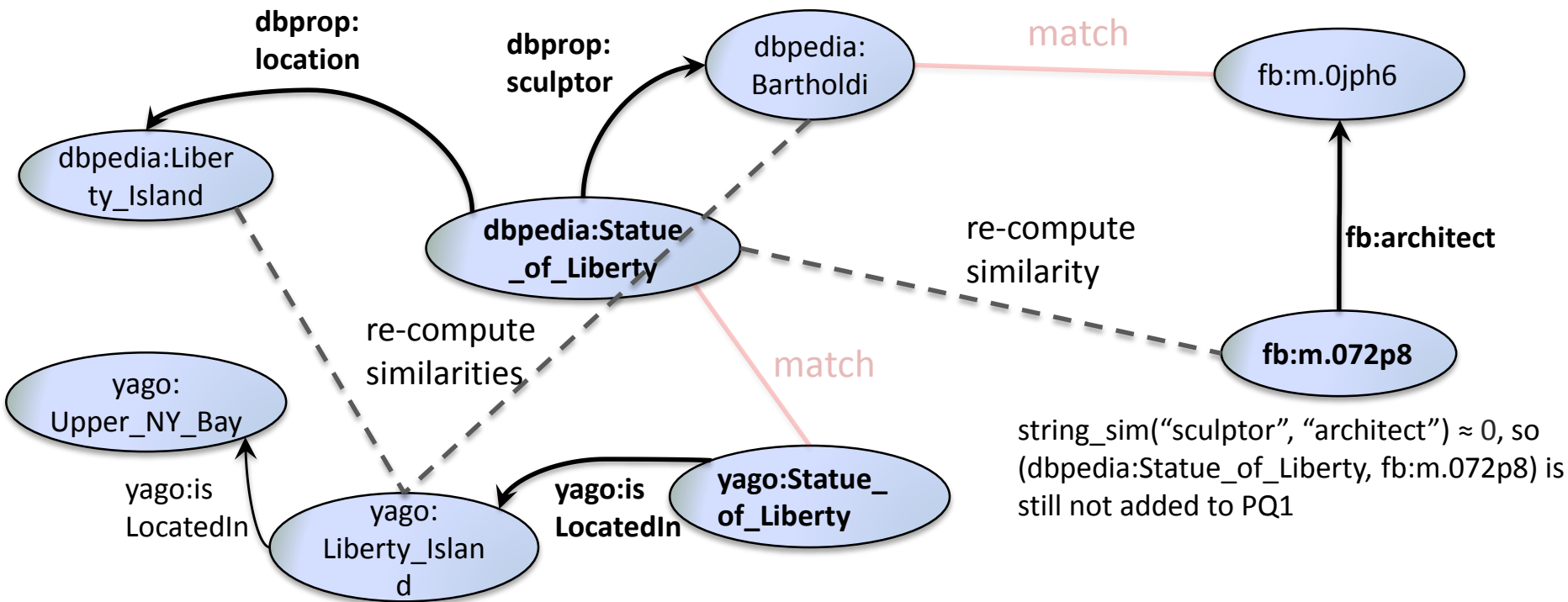
Priority Queue 1:

(dbpedia:Statue_of_Liberty, yago:Statue_of_Liberty)
(dbpedia:Statue_of_Liberty, yago:Liberty_Island)
(dbpedia:Liberty_Island, yago:Upper_NY_Bay)
(dbpedia:Liberty_Island, yago:Liberty_Island)
(dbpedia:Liberty_Island, yago:Statue_of_Liberty)

Priority Queue 2:

(dbpedia:Bartholdi, fb:m.0jph6)
(dbpedia:Bartholdi, yago:Statue_of_Liberty)
(dbpedia:Bartholdi, fb:m.072p8)

Send messages to the other nodes and check this constraint again

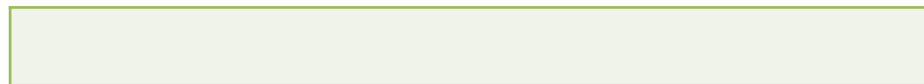


Priority Queue 1:

(dbpedia:Liberty_Island,yago:Upper_NY_Bay)

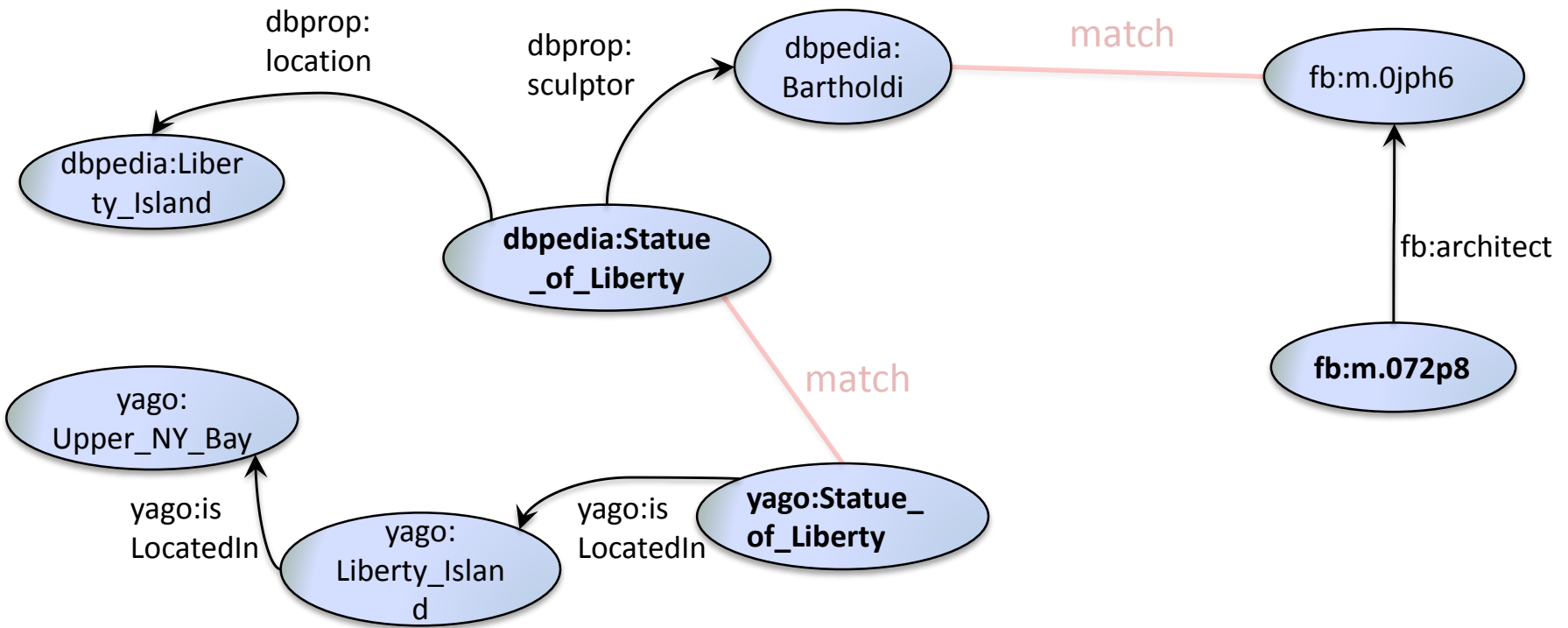
(dbpedia:Liberty_Island, yago:Liberty_Island)

Priority Queue 2:



Contextual similarity re-computations

Property names are also taken into account



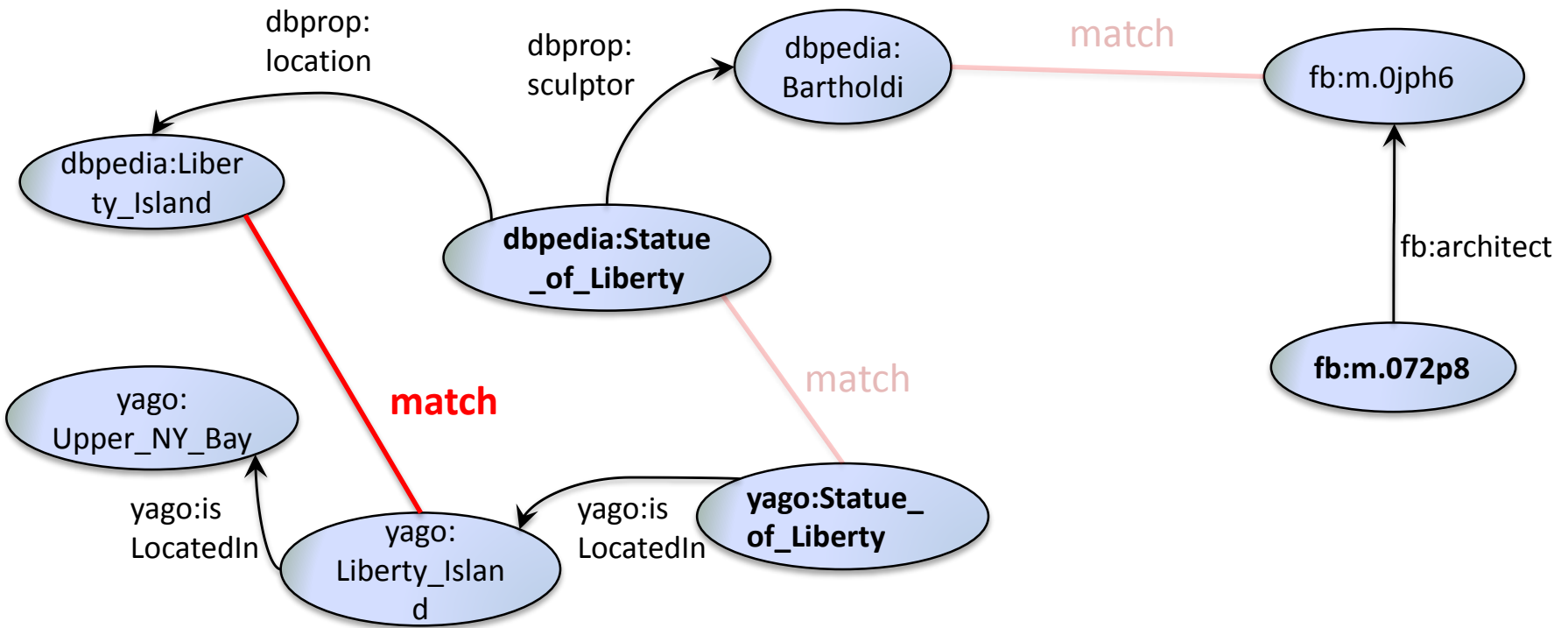
Priority Queue 1:

(dbpedia:Liberty_Island, yago:Liberty_Island)

(dbpedia:Liberty_Island, yago:Upper_NY_Bay)

Priority Queue 2:

Priority queues are updated based on the new similarities



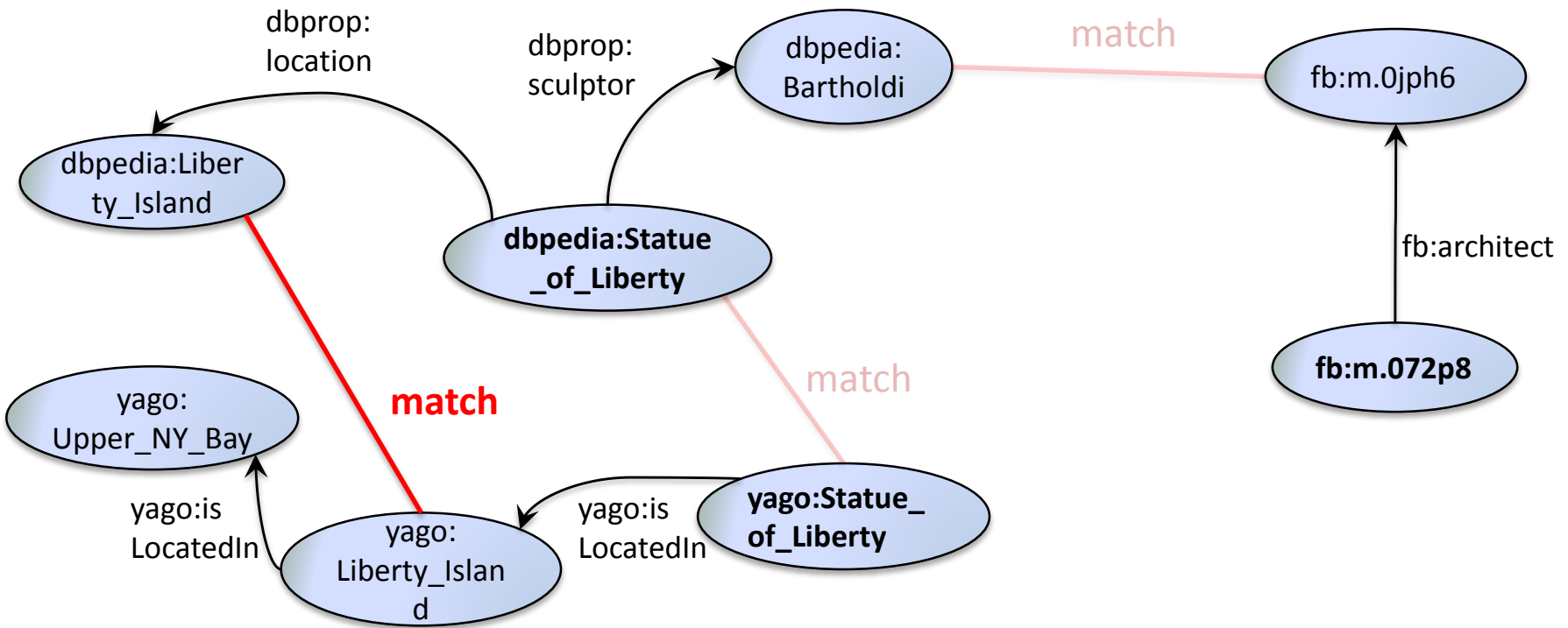
Priority Queue 1:

(dbpedia:Liberty_Island, yago:Liberty_Island)

(dbpedia:Liberty_Island, yago:Upper_NY_Bay)

Priority Queue 2:

The head of each queue is a match by default
This triggers update messages



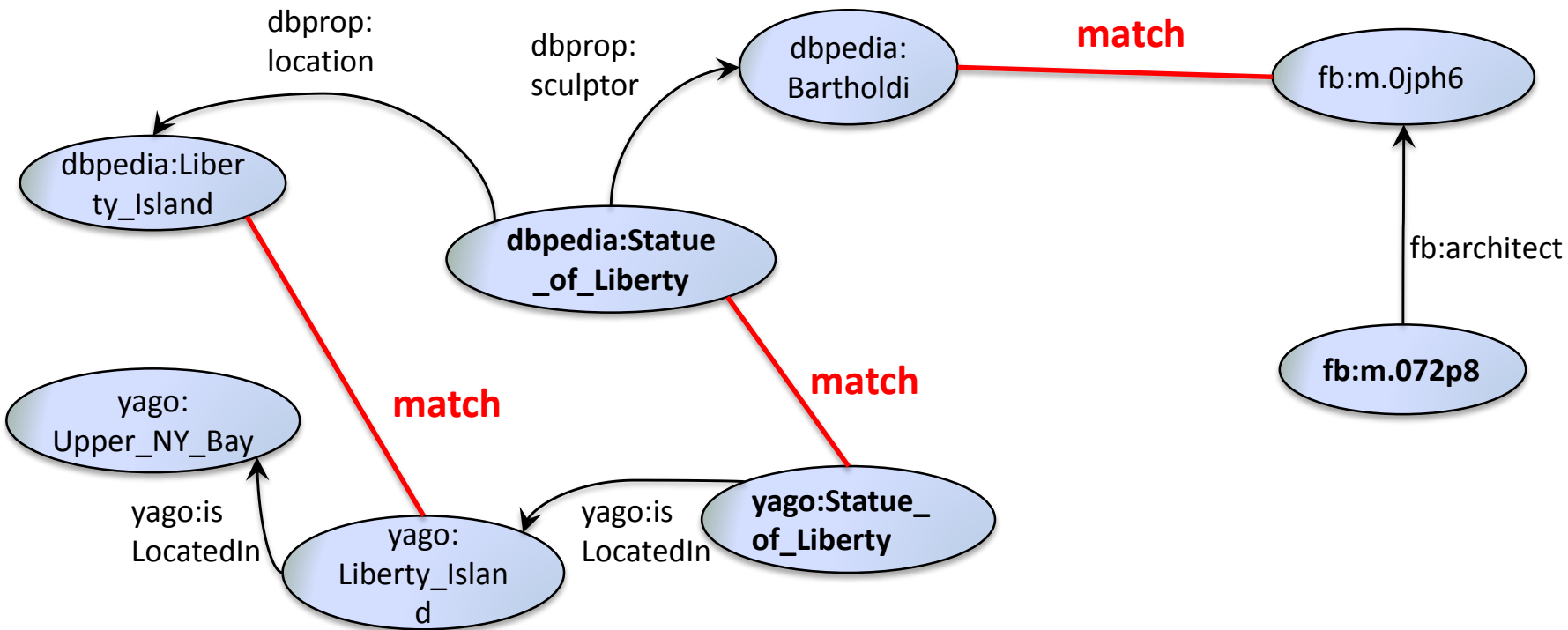
Priority Queue 1:

(dbpedia:Liberty_Island, yago:Liberty_Island)

{dbpedia:Liberty_Island, yago:Upper_NY_Bay}

Priority Queue 2:

Dequeue this pair, as each entity can be mapped to at most one entity per data source



Priority Queue 1:

Priority Queue 2:

Output mappings

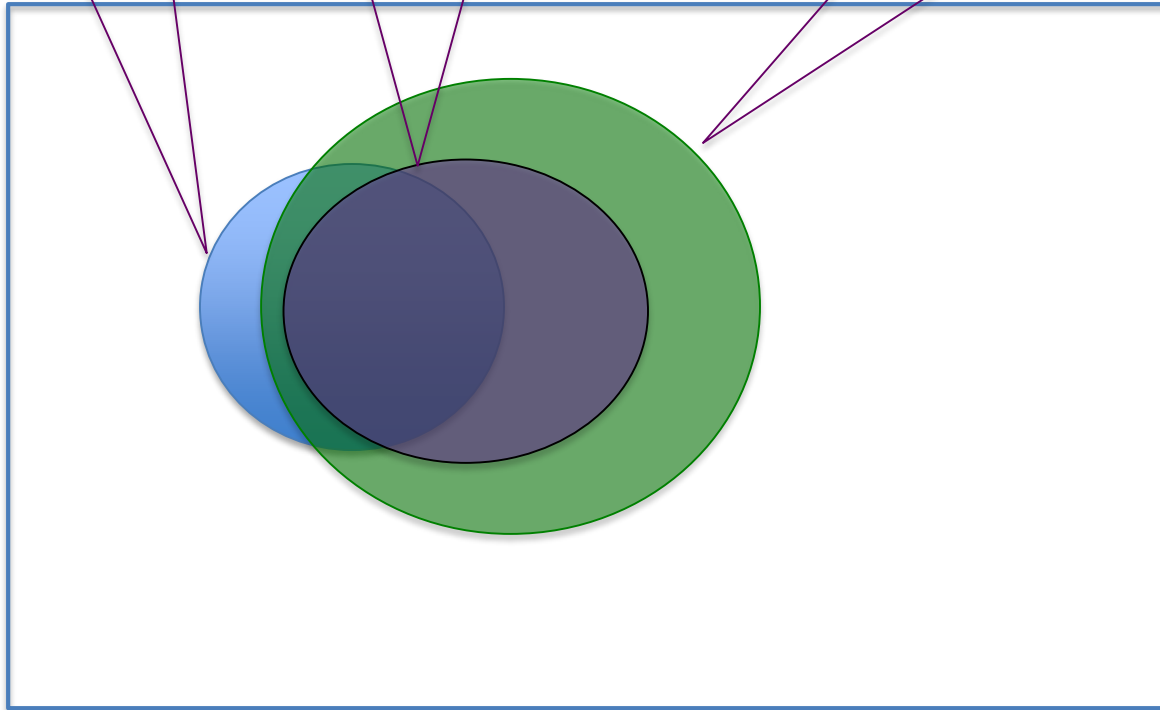
Using Neighbors for Computing Similarities

Matching pairs
of entity
descriptions

Without neighbors
(a loose similarity
function is used)

With neighbors
(a strict similarity
function is used)

Set of all pairs
of entity
descriptions



With neighbors
(pros) Lead to more
identified matches
(cons) Lead to more
comparisons

Entity Resolution in the Web of Data

So far...

Rely on the values of the descriptions

- *A good way to handle data heterogeneity and low structuredness*

=> Deal with loosely structured entities

=> Deal with various vocabularies
(side effect)

=> Deal with large volumes of data

Still, many redundant comparisons are performed

- Can we also use the structural type of the descriptions?

A brief overview of our ongoing work on blocking using MapReduce follows...

Methods and Datasets

MapReduce implementations of:

- Token Blocking (ToB)
- Attribute Clustering Blocking (AtC)
- Prefix-Infix(-Suffix) Blocking (PIS)

Datasets: [Billion Triples Challenge 2012](#) (BTC12)

- D1: BTC12 DBpedia (~148M triples) and [DBpedia 3.5](#) (~32M triples)
- D2: BTC12 DBpedia and BTC12 Rest (~918K triples)
- D3: BTC12 DBpedia and BTC12 Freebase (~29M triples)

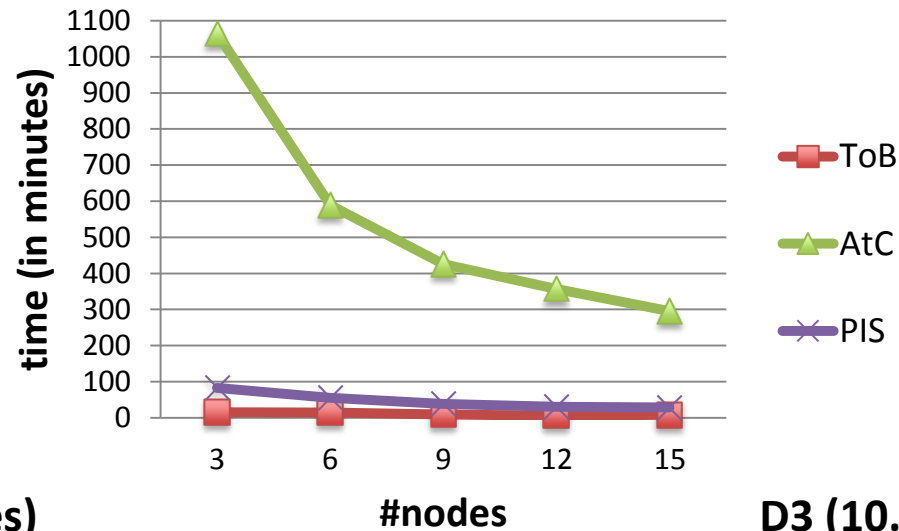
On a cluster of 15 nodes (each with 8 CPUs, 8GB RAM, and 60GB HDD)

For ground-truth, we used the *owl:sameAs* links provided by DBpedia

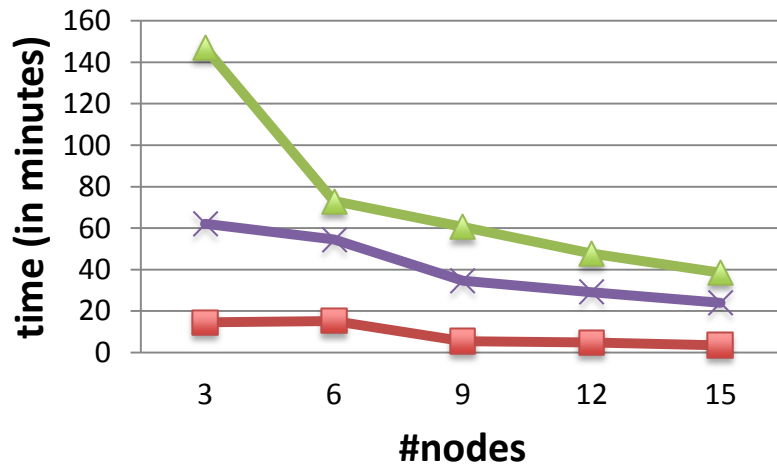
For each dataset, we performed both dirty and clean-clean ER

Scalability

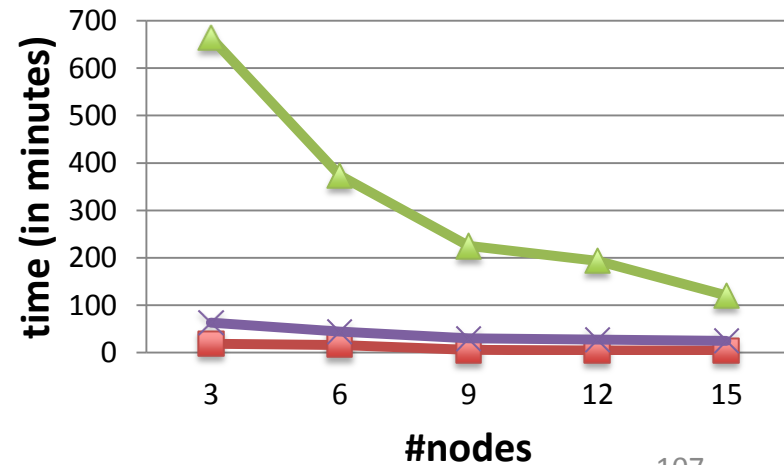
D1 (10.6M entities)



D2 (9M entities)



D3 (10.8M entities)



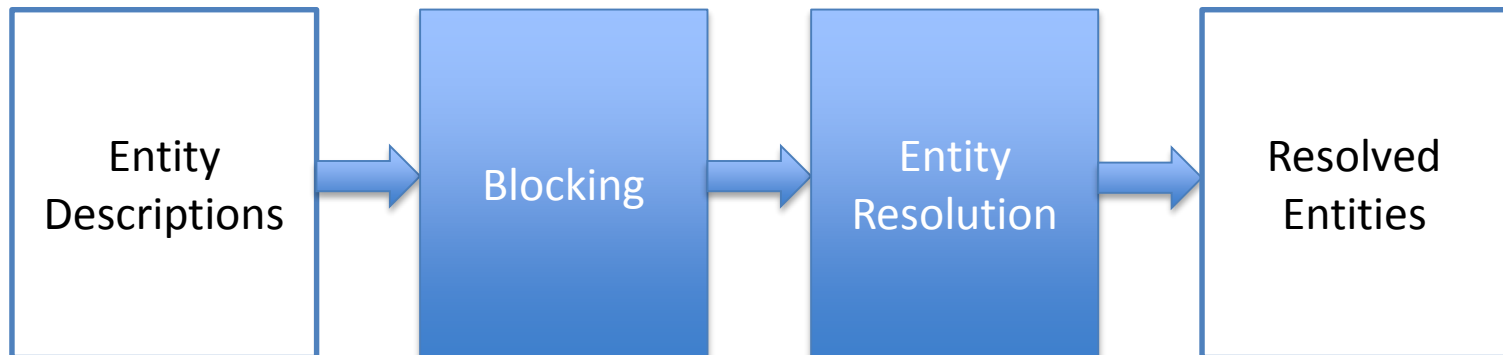
Quality Results

		ToB (Dirty)	AtC (Clean-Clean)	PIS (Dirty)
Precision	D1	$3.598 \cdot 10^{-7}$	$8.064 \cdot 10^{-6}$	$9.955 \cdot 10^{-7}$
	D2	$2.540 \cdot 10^{-8}$	$6.103 \cdot 10^{-6}$	$5.964 \cdot 10^{-8}$
	D3	$4.376 \cdot 10^{-7}$	$9.759 \cdot 10^{-6}$	$3.729 \cdot 10^{-6}$
Recall	D1	90.5%	88%	91.7%
	D2	86.3%	70.6%	87.6%
	D3	86.5%	84.3%	84.4%
RR	D1	90.1%	97.8%	92.5%
	D2	90.9%	98.6%	91.7%
	D3	92.7%	98.9%	90.8%
#comparisons	D1	5,588 B	322 B	4,394 B
	D2	3,681 B	4 B	3,449 B
	D3	4,272 B	184 B	5,339 B

Room for Improvement

Ideally:

- Similar entity descriptions in the same block
- **Dissimilar entity descriptions in different blocks?**



Current Trends

Temporal Entity Resolution

Entity resolution should account for changes over time

- Entities evolve over time
- Entities have a lifetime

Linked Datasets Evolve Over Time

Current version of DBpedia

 DBpedia	dbpedia:Statue_of_Liberty
rdfs:label	Statue of Liberty, Freiheitsstatue, ...
dbpprop:location	New York City, New York, U.S., dbpedia:Liberty_Island
dbpprop:sculptor	dbpedia:Frédéric_Auguste_Bartholdi
dcterms:subject	dbpedia_category:1886_sculptures , ...
foaf:isPrimaryTopicOf	http://en.wikipedia.org/wiki/Statue_of_Liberty
dbpprop:beginningDate	1886-10-28 (xsd:date)
dbpprop:restored	19381984 (xsd:integer)
dbpprop:visitationNum	3200000 (xsd:integer)
dbpprop:visitationYear	2009 (xsd:integer)
http://www.w3.org/ns/prov#wasDerivedFrom	http://en.wikipedia.org/wiki/Statue_of_Liberty?oldid=494328330

Previous version of DBpedia

 DBpedia	dbpedia:Statue_of_Liberty
rdfs:label	Statue of Liberty, Freiheitsstatue, ...
dbpprop:location	New York City, New York, U.S., dbpedia:Liberty_Island
dbpprop:sculptor	dbpedia:Frédéric_Auguste_Bartholdi
dcterms:subject	dbpedia_category:1886_sculptures , ...
foaf:isPrimaryTopicOf	http://en.wikipedia.org/wiki/Statue_of_Liberty
dbpprop:built	1886-10-28 (xsd:date)
dbpprop:restored	19381984 (xsd:integer)
dbpprop:hasHeight	151 (xsd:integer)
http://www.w3.org/ns/prov#wasDerivedFrom	http://en.wikipedia.org/wiki/Statue_of_Liberty?oldid=494328330

Entities Have a Lifetime

Example: Matching a description of Ronald Reagan, stating that he **was** the US President to a description of Barack Obama, stating that he **is** the US President

Yago2 [Hoffart et al. 2012]: A temporal knowledge base, built with data from Wikipedia, GeoNames and Wordnet

- Entities are assigned a time span to denote their existence in time
 - e.g. Ronald Reagan is associated with 1911-02-06 (birthdate) and 2004-06-05 (time of death)
- Facts are assigned a time point, or a time span
 - e.g. the fact “Ronald Reagan is US President” is associated with the time span from 1981-01-20 to 1989-01-20

Privacy Protection

Example (PleaseRobMe.com): Exploit Foursquare to detect the current location of Twitter users and identify houses easy to burgle, as the inhabitants are traveling at the time

Measure one's ability to link two matching descriptions and then, try to minimize it

Privacy breach can be measured in terms of how much of the complete information about a person is available to an adversary [Whang et al. 2011, 2013]

- Use *disinformation* to make entity resolution for an adversary difficult:
 - Link a description to irrelevant descriptions
 - Add “incorrect but believable” information to a description

Crowdsourcing

Instead of machine-only approaches, use hybrid human-machine approaches

- Pass the critical (expensive/difficult) decisions to the users

Machines do a first, coarse pass over all the data (e.g. blocking)

- When algorithms fail to reach a match decision, ask humans [Demartini et al. 2013]
- Humans are used to verify only the most likely matches [Wang et al. 2012]

Humans can make mistakes too! (spam, low attention, etc.)

- Given a set of descriptions, find which questions between pairs of descriptions will reveal the most about the underlying entities [Verroios et al. 2014]

Thank You!

Other points for future work?

Questions?

References

References

- Menestrina, D., Whang, S., Garcia-Molina, H.: Evaluating entity resolution results. PVLDB 3(1), 208–219 (2010)
- Papadakis, G., Ioannou, E., Niederee, C., Palpanas, T., Nejdl, W.: Beyond 100 million entities: large-scale blocking-based resolution for heterogeneous data. In: WSDM, pp. 53–62 (2012)
- Fellegi, I.P., Sunter, A.B.: A theory for record linkage. Journal of the American Statistical Association 64(328), 1183–1210 (1969)
- Hernandez, M.A., Stolfo, S.J.: The merge/purge problem for large databases. In: SIGMOD (1995)
- Yan, S., Lee, D., Kan, M.Y., Giles, C.L.: Adaptive sorted neighborhood methods for efficient record linkage. In: JCDL, pp. 185–194 (2007)
- Draisbach, U., Naumann, F.: A comparison and generalization of blocking and windowing algorithms for duplicate detection. In: QDB, pp. 51–56 (2009)
- McCallum, A., Nigam, K., Ungar, L.H.: Efficient clustering of high-dimensional data sets with application to reference matching. In: KDD, pp. 169–178 (2000)
- Christen, P.: A survey of indexing techniques for scalable record linkage and deduplication. IEEE Trans. Knowl. Data Eng. 24(9), 1537–1555 (2012)
- Gravano, L., Ipeirotis, P.G., Jagadish, H.V., Koudas, N., Muthukrishnan, S., Srivastava, D.: Approximate string joins in a database (almost) for free. In: VLDB, pp. 491–500 (2001)
- Aizawa, A.N., Oyama, K.: A fast linkage detection scheme for multi-source information integration. In: WIRI, pp. 30–39 (2005)
- Jin, L., Li, C., Mehrotra, S.: Efficient record linkage in large data sets. In: DASFAA, pp. 137–146 (2003)
- Draisbach, U., Naumann, F., Szott, S., Wonneberg, O.: Adaptive Windows for Duplicate Detection. ICDE 2012: 1073–1083

References

- Kolb, L., Thor, A., Rahm, E.: Block-based Load Balancing for Entity Resolution with MapReduce. In: CIKM, pp. 2397–2400 (2011)
- Kolb, L., Thor, A., Rahm, E.: Dedoop: Efficient Deduplication with Hadoop. PVLDB 5(12), 1878–1881 (2012)
- Benjelloun, O., Garcia-Molina, H., Menestrina, D., Su, Q., Whang, S.E., Widom, J.: Swoosh: a generic approach to entity resolution. VLDB J. 18(1), 255–276 (2009)
- Benjelloun, O., Garcia-Molina, H., Gong, H., Kawai, H., Larson, T.E., Menestrina, D., Thavisomboon, S.: D-swoosh: A family of algorithms for generic, distributed entity resolution. In: ICDCS, p. 37 (2007)
- Whang, S.E., Menestrina, D., Koutrika, G., Theobald, M., Garcia-Molina, H.: Entity resolution with iterative blocking. In: SIGMOD, pp. 219–232 (2009)
- Kim, H., Lee, D.: Harra: fast iterative hashed record linkage for large-scale data collections. In: EDBT, pp. 525–536 (2010)
- Herschel, M., Naumann, F., Szott, S., Taubert, M.: Scalable iterative graph duplicate detection. IEEE Trans. Knowl. Data Eng. 24(11), 2094–2108 (2012)
- Dong, X., Halevy, A.Y., Madhavan, J.: Reference reconciliation in complex information spaces. In: SIGMOD, pp. 85–96 (2005)
- Bhattacharya, I., Getoor, L.: Iterative record linkage for cleaning and integration. In: DMKD (2004)
- Bhattacharya, I., Getoor, L.: Collective entity resolution in relational data. TKDD 1(1) (2007)
- Cochinwala, M., Kurien, V., Lalk, G., Shasha, D.: Efficient data reconciliation. Inf. Sci. 137(1-4), 1–15 (2001)
- Bilenko, M., Mooney, R.J.: Adaptive duplicate detection using learnable string similarity measures. In: KDD, pp. 39–48 (2003)

References

- Christen, P.: Automatic record linkage using seeded nearest neighbour and support vector machine classification. In: KDD, pp. 151–159 (2008)
- Chen, Z., Kalashnikov, D. V., Mehrotra, S.: Exploiting context analysis for combining multiple entity resolution systems. In: SIGMOD, pp. 207–218 (2009)
- Ravikumar, P. D., Cohen, W. W.: A hierarchical graphical model for record linkage. In: UAI (2004)
- Bhattacharya, I., Getoor, L.: A latent dirichlet model for unsupervised entity resolution. In: SDM (2006)
- Sarawagi, S., Bhamidipaty, A.: Interactive deduplication using active learning. In: KDD (2002)
- Tejada, S., Knoblock, C. A., Minton, S.: Learning domain-independent string transformation weights for high accuracy object identification. In: KDD, pp. 350–359 (2002)
- Tejada, S., Knoblock, C. A., Minton, S.: Learning object identification rules for information integration. *Inf. Syst.* 26(8), 607–633 (2001)
- Wang, J., Kraska, T., Franklin, M. J., Feng, J.: Crowder: Crowdsourcing entity resolution. *PVLDB* 5(11), 1483–1494 (2012)
- Verykios, V. S., Elmagarmid, A. K., Houstis, E. N.: Automating the approximate record-matching process. *Inf. Sci.* 126(1-4), 83–98 (2000)
- Weis, M., Naumann, F.: Detecting duplicates in complex xml data. In: ICDE, p. 109 (2006)
- Weis, M., Naumann, F.: Detecting duplicate objects in xml documents. In: IQIS, pp. 10–19 (2004)
- Leitao, L., Calado, P., Weis, M.: Structure-based inference of xml similarity for fuzzy duplicate detection. In: CIKM, pp. 293–302 (2007)
- Leitao, L., Calado, P., Herschel, M.: Efficient and effective duplicate detection in hierarchical data. *IEEE Trans. Knowl. Data Eng.* 25(5), 1028–1041 (2013)
- Herschel, M., Berti, L.: Application de mesures de distance pour la détection de problèmes de qualité de données. Book Chapter in *La qualité et la gouvernance de données au service de la performance des entreprises*, Ed. Hermes (2012)

References

- Puhlmann, S., Weis, M., Naumann, F.: Xml duplicate detection using sorted neighborhoods. In: EDBT (2006)
- Bohm, C., de Melo, G., Naumann, F., Weikum, G.: Linda: distributed web-of-data-scale entity matching. In: CIKM (2012)
- Papadakis, G., Ioannou, E., Niederee, C., Fankhauser, P.: Efficient entity resolution for large heterogeneous information spaces. In: WSDM (2011)
- Papadakis, G., Ioannou, E., Palpanas, T., Niederee, C., Nejdl, W.: A Blocking Framework for Entity Resolution in Highly Heterogeneous Information Spaces. *IEEE Trans. Knowl. Data Eng.* 25(12) 2665–2682 (2013) (a)
- Papadakis, G., Koutrika, G., Palpanas, T., Nejdl, W.: Meta-blocking: Taking entity resolution to the next level. *IEEE Trans. Knowl. Data Eng.* (2013) (b). To appear
- Hoffart, J., Seufert, S., Nguyen, D.B., Theobald, M., Weikum, G.: Kore: keyphrase overlap relatedness for entity disambiguation. In: CIKM (2012)
- Ananthakrishna, R., Chaudhuri, S., Ganti, V.: Eliminating fuzzy duplicates in data warehouses. In: VLDB (2002)
- Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.* 19(1), 1–16 (2007)
- Getoor, L., Machanavajjhala, A.: Entity resolution: Theory, practice & open challenges. *PVLDB* 5(12), 2018–2019 (2012)
- Papadakis, G., Demartini, G., Fankhauser, P., Karger, P.: The missing links: discovering hidden same-as links among a billion of triples. In: iiWAS, pp. 453–460 (2010)
- Hernández, M.A., Stolfo, S.J.: Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. *Data Min. Knowl. Discov.* 2(1): 9–37 (1998)

References

- Duan, S., Kementsietsidis, A., Srinivas, K., Udrea, O.: Apples and oranges: a comparison of rdf benchmarks and real rdf datasets. In: SIGMOD, pp. 145–156 (2011)
- Neumann, T., Moerkotte, G.: Characteristic sets: Accu-rate cardinality estimation for rdf queries with multiplejoins. In: ICDE, pp. 984–994 (2011)
- Rastogi, V., Dalvi, N.N., Garofalakis, M. N. : Large-Scale Collective Entity Matching. PVLDB 4(4): 208-218 (2011)
- Suchanek, F.M., Weikum, G.: Knowledge harvesting in the big-data era. In: SIGMOD, pp. 933-938 (2013)
- Hoffart, J., Suchanek F.M., Berberich, K., Weikum, G.: : YAGO2: A Spatially and Temporally Enhanced Knowledge Base from Wikipedia: Extended Abstract. In: IJCAI (2013)
- Whang, S.E., Marmaros, D., Garcia-Molina, H.: Pay-As-You-Go Entity Resolution. IEEE Trans. Knowl. Data Eng. 25(5): 1111-1124 (2013)
- Kolb, L., Thor, A., Rahm, E.: Don't match twice: redundancy-free similarity computation with MapReduce. In: Data Analytics in the Cloud (2013)
- Vernica, R., Carey, M.J., Li, C.: Efficient parallel set-similarity joins using MapReduce. In SIGMOD, pp. 495-506 (2010)
- Bilenko, M., Mooney, R. J. , Cohen, W. W., Ravikumar, P. D., Fienberg, S. E.: Adaptive NameMatching in Information Integration. IEEE Intelligent Systems 18(5): 16-23 (2003)
- Baeza-Yates, R. A., Ribeiro-Neto, B. A.: Modern Information Retrieval. ACM Press / Addison-Wesley 1999, ISBN 0-201-39829-X
- Calado, P., Herschel, M., Leitão, L.: An Overview of XML Duplicate Detection Algorithms. Soft Computing in XML Data Management 2010: 193-224
- Christen, P.: Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection. Data-centric systems and applications, Springer 2012, ISBN 978-3-642-31163-5, pp. I-XIX, 1-270
- Jaro, M.A.. Advances in record linking methodology as applied to matching the 1985 census of Tampa Florida. J. Am. Stat. Assoc., vol. 84 (406), p. 414 - 420, 1989

References

- Kolb, L., Thor, A., Rahm, E.: Load Balancing for MapReduce-based Entity Resolution. In: ICDE (2012)
- Leser, U., Naumann, F.: Informationsintegration: Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen. dpunkt 2006, ISBN 3-89864-400-6
- McClellan, M.A.: Duplicate Medical Records: A Survey of Twin Cities Healthcare Organizations. AMIA Annual Symposium (2009)
- Naumann, F., Herschel, M.: An Introduction to Duplicate Detection. Synthesis Lectures on Data Management, Morgan & Claypool Publishers (2010)
- Weis, M., Naumann, F.: DogmatiX Tracks down Duplicates in XML. In: SIGMOD, pp. 431-442 (2005)
- Weis, M., Naumann, F., Jehle, U., Lufter, J., Schuster, H.: Industry-scale duplicate detection. PVLDB 1(2): 1253-1264 (2008)
- Whang S.E., Garcia-Molina, H.: Managing Information Leakage. In: CIDR (2011)
- Whang, S.E., Garcia-Molina, H.: Disinformation techniques for entity resolution. In: CIKM (2013)
- Demartini, G., Difallah, D.E., Cudré-Mauroux, P.: Large-scale linked data integration using probabilistic reasoning and crowdsourcing. VLDB J. 22(5): 665-687 (2013)

Acknowledgements

We are thankful to the support provided by the following projects:

- FP7 ICT IdeaGarden STREP <http://idea-garden.org/>
- GSRT ARISTEIA (LODGOV) Data Governance in the era of the Web of Data

License

These slides are made available under a Creative Commons Attribution-ShareAlike license (CC BY-SA 3.0):

<http://creativecommons.org/licenses/by-sa/3.0/>



You can share and remix this work, provided that you keep the attribution to the original authors intact, and that, if you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.