Entity Resolution in the Web of Data

Kostas Stefanidis[1], Vasilis Efthymiou[1,2], Melanie Herschel[3,4], Vassilis Christophides[5]

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

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

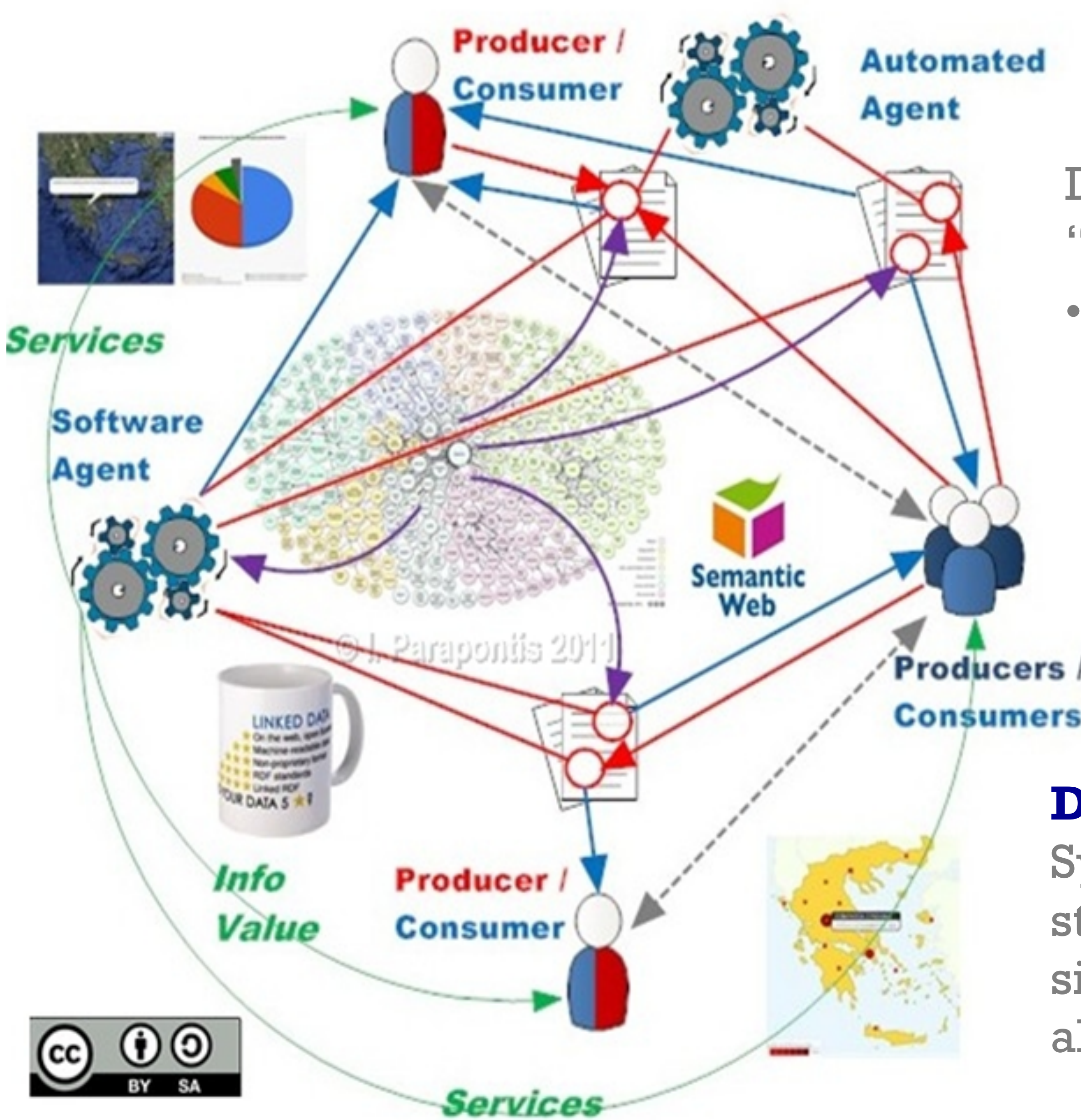
A bit of History: from Web 1.0 & 2.0 …

Many Web sites containing unstructured, textual content

Few large Web sites are specialized on specific content types

- Semi-structured/xml content floating around e-services

A bit of History: …to Web 3.0



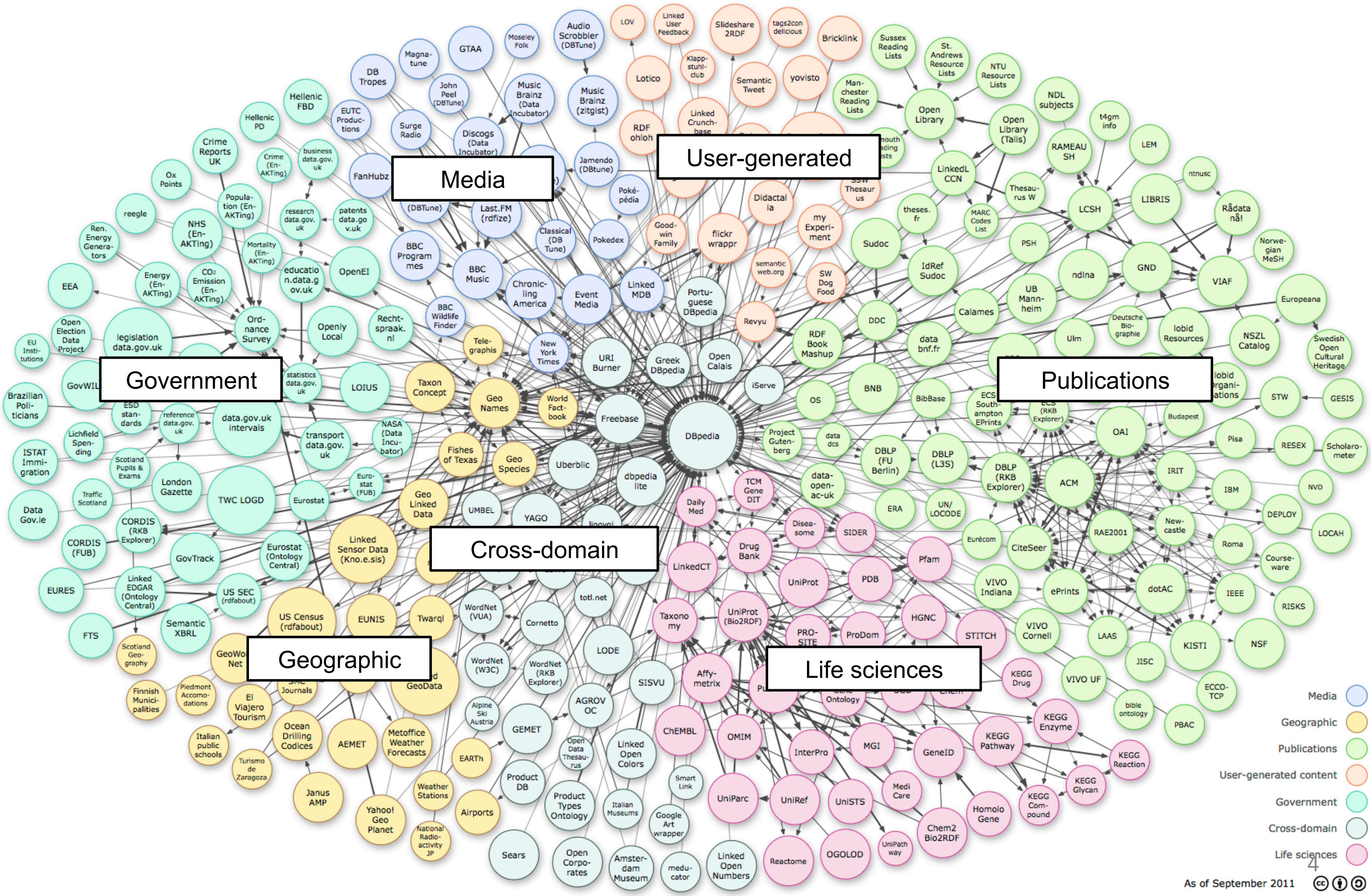
Data themselves become "*infrastructure*"

- A valuable asset, on which science, technology, economy and society can advance

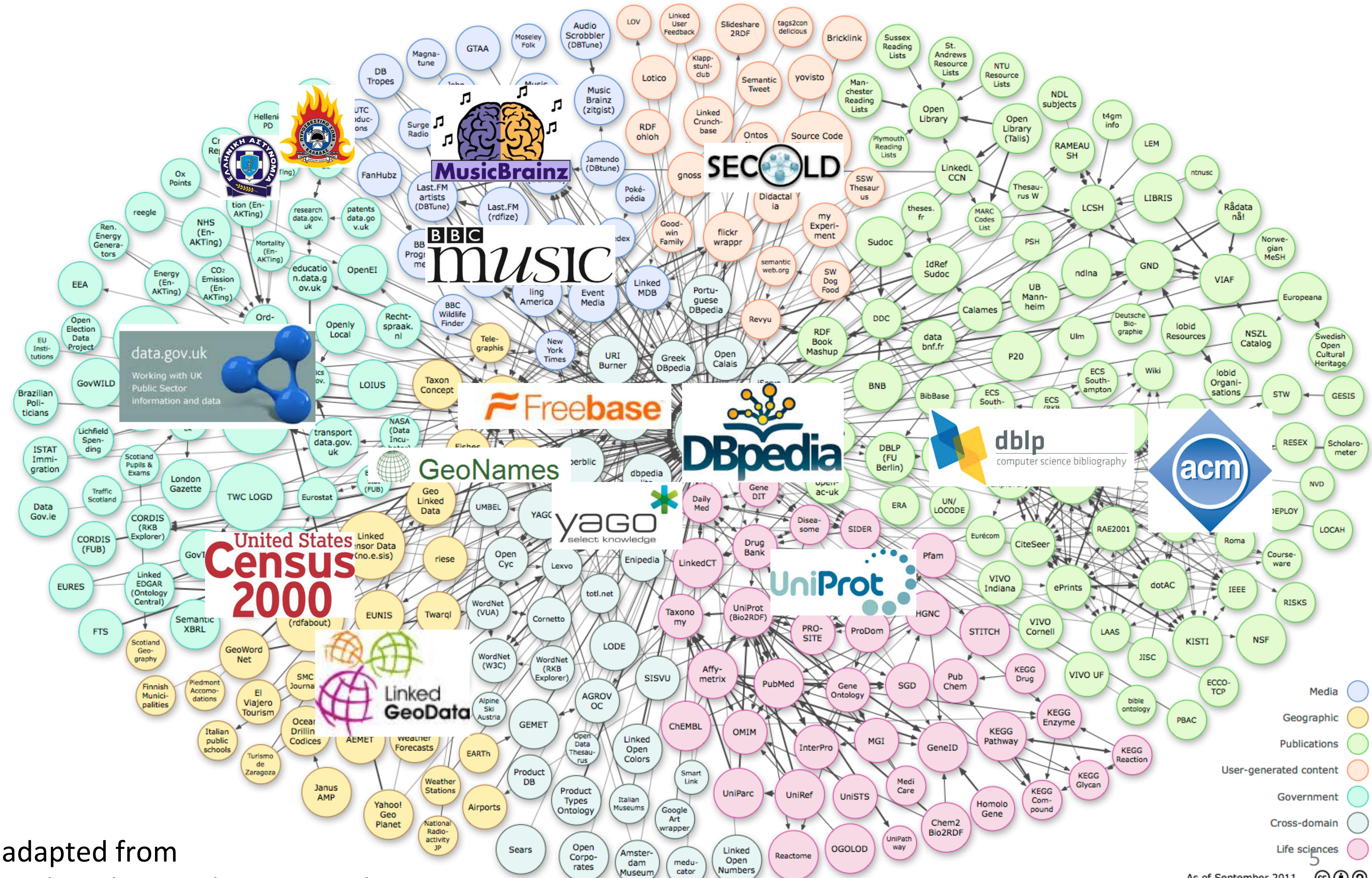
Data as Service (DaaS)

Syndicating arbitrarily semi-structured content across Web sites using higher-level data abstractions (entities)

LOD Cloud – The Structured Subset of the Web of Data

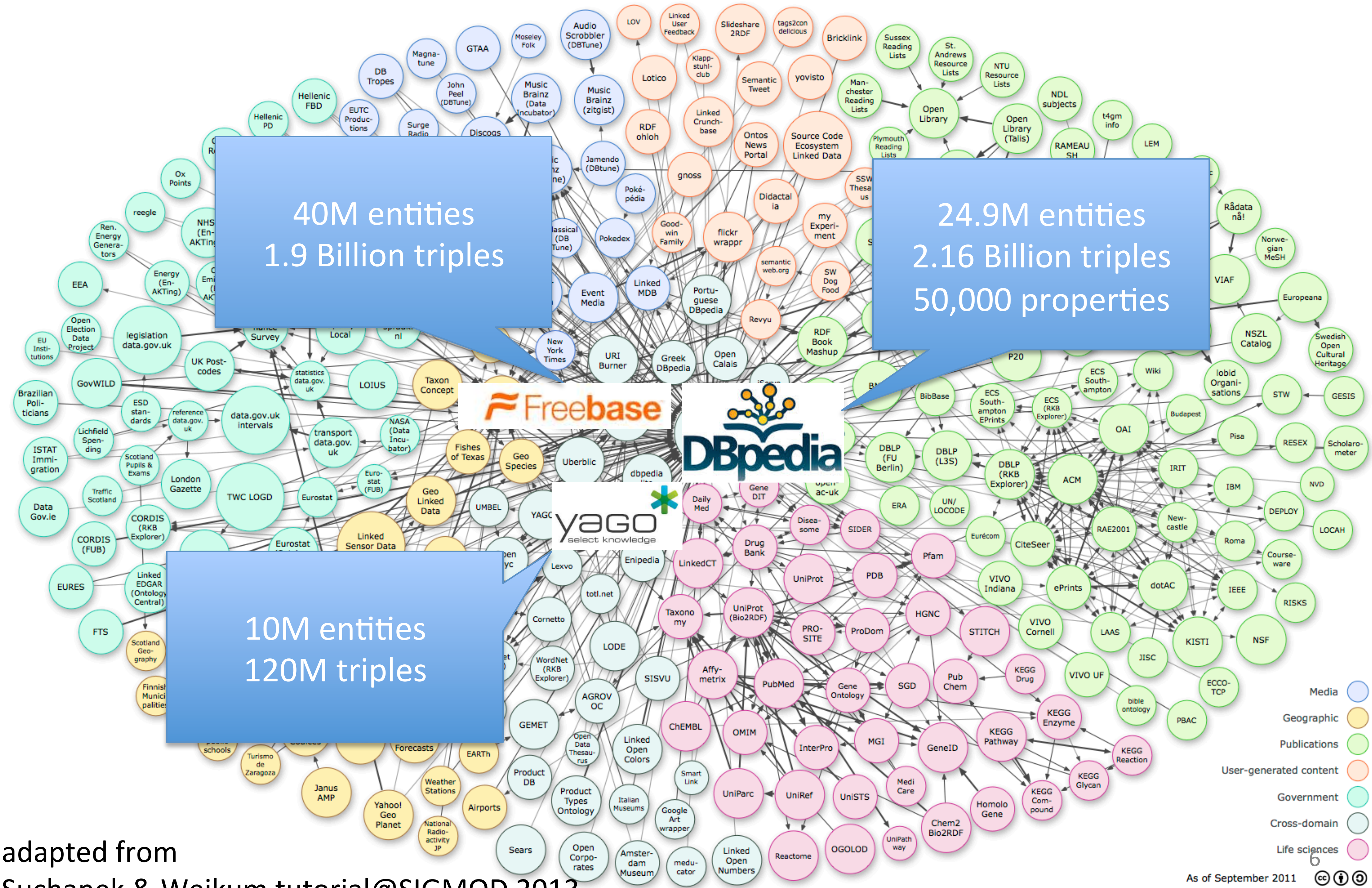


LOD Cloud – The Structured Subset of the Web of Data



adapted from
Suchanek & Weikum tutorial@SIGMOD 2013

LOD Cloud – The Structured Subset of the Web of Data



adapted from
Suchanek & Weikum tutorial@SIGMOD 2013

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

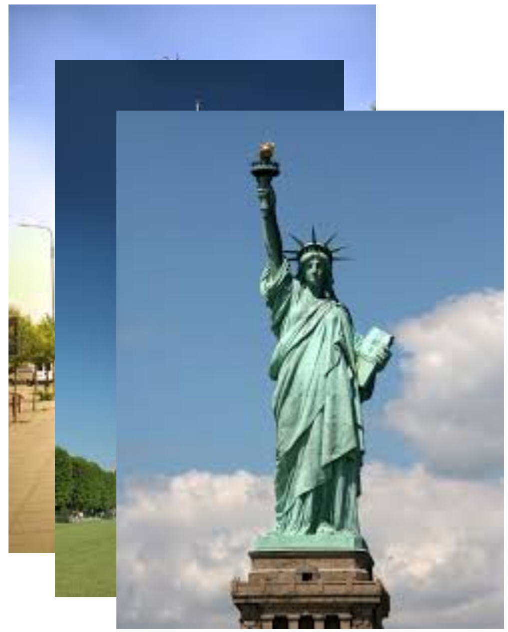
Entities: An Invaluable Asset

Monuments

"Entities" is what a large part of our knowledge is about

Entities: An Invaluable Asset

Locations

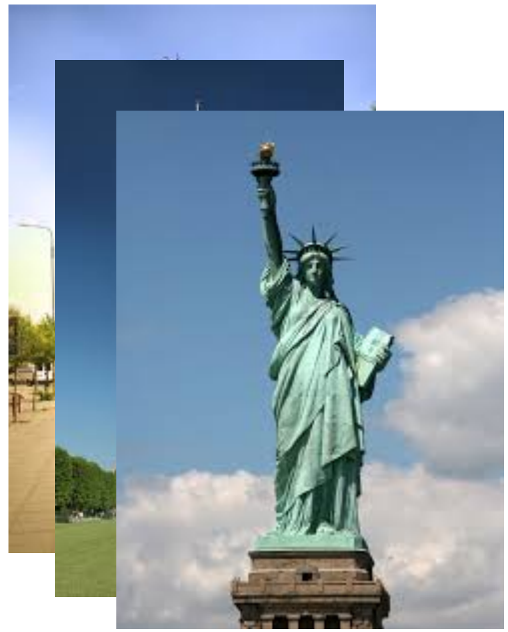


Monuments

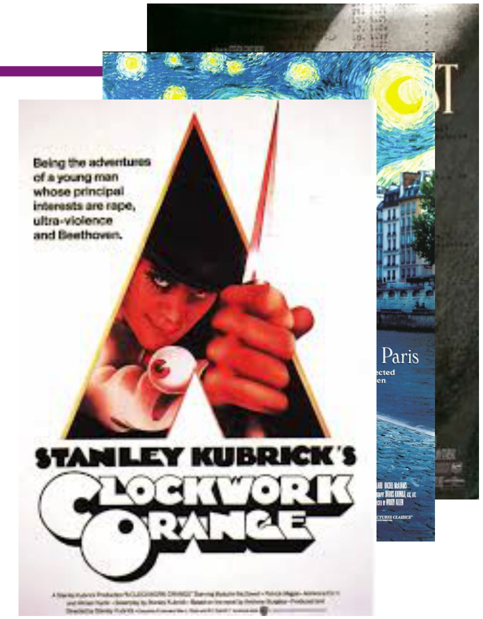
"Entities" is what a large part of our knowledge is about

Entities: An Invaluable Asset

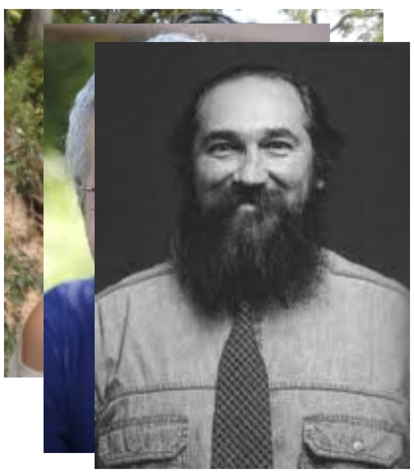
Locations



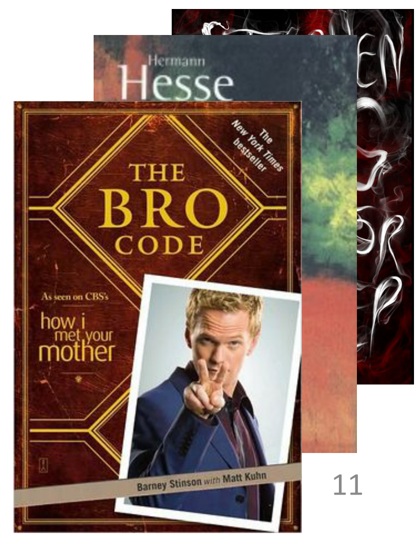
Monuments



Movies

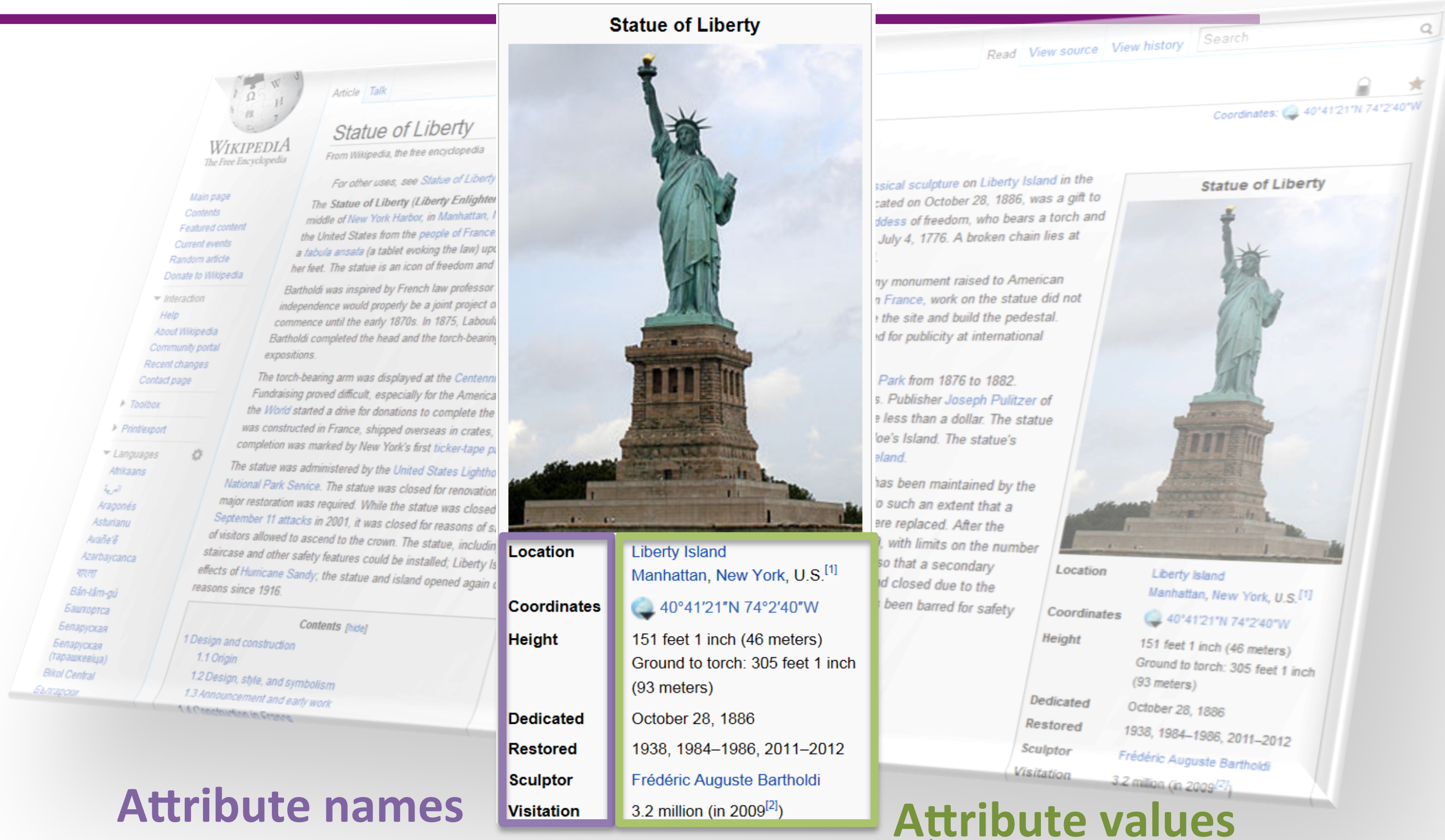


Persons



Books

Example: General Knowledge Bases



Attribute names

Attribute values

yago select knowledge

DBpedia

Freebase

Different Descriptions of the same Entity

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 select knowledge	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	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 select knowledge	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 select knowledge	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 Evolve Over Time

Current version of 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: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

We should somehow link these descriptions

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

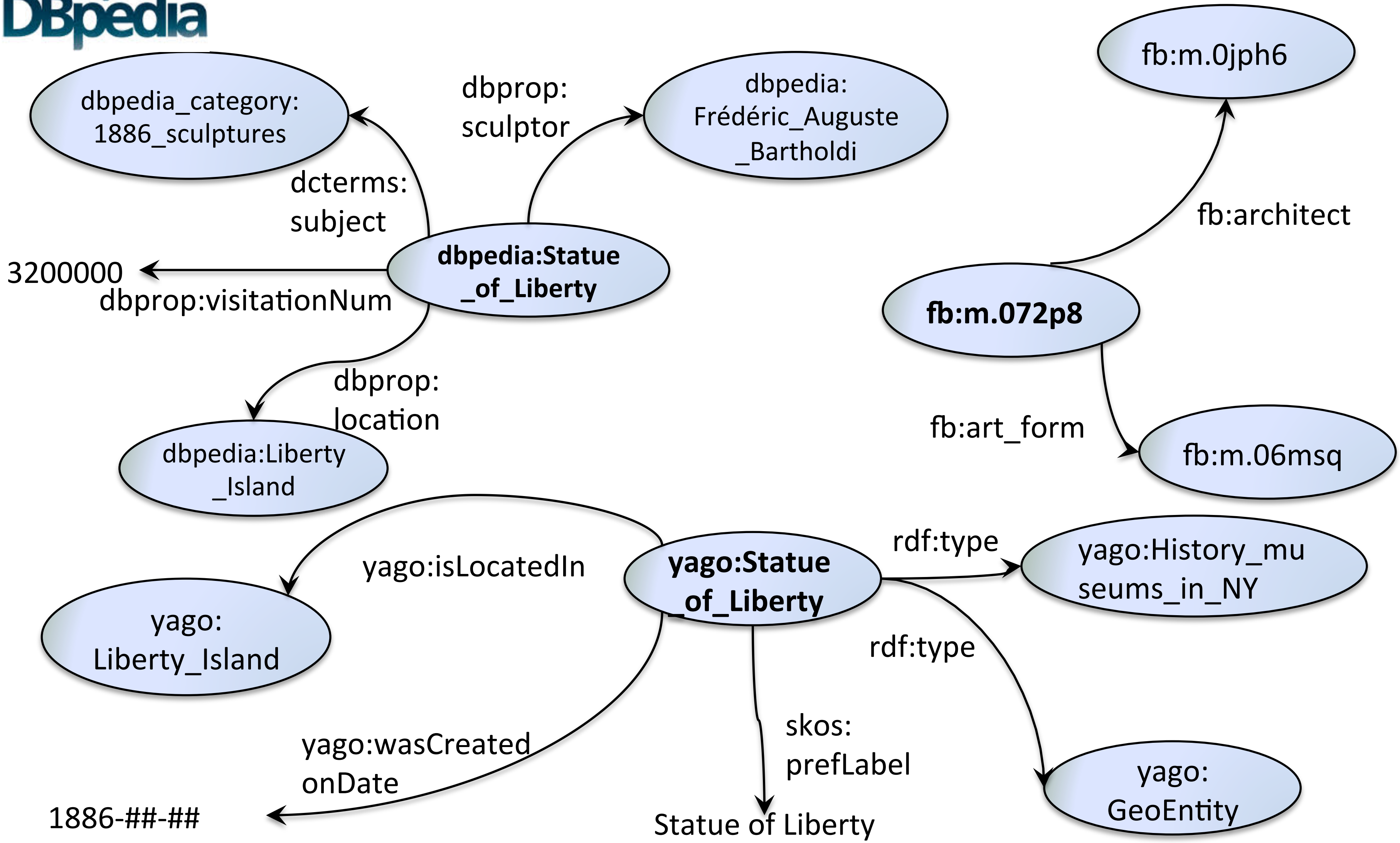
Two kinds of entity collections as input:

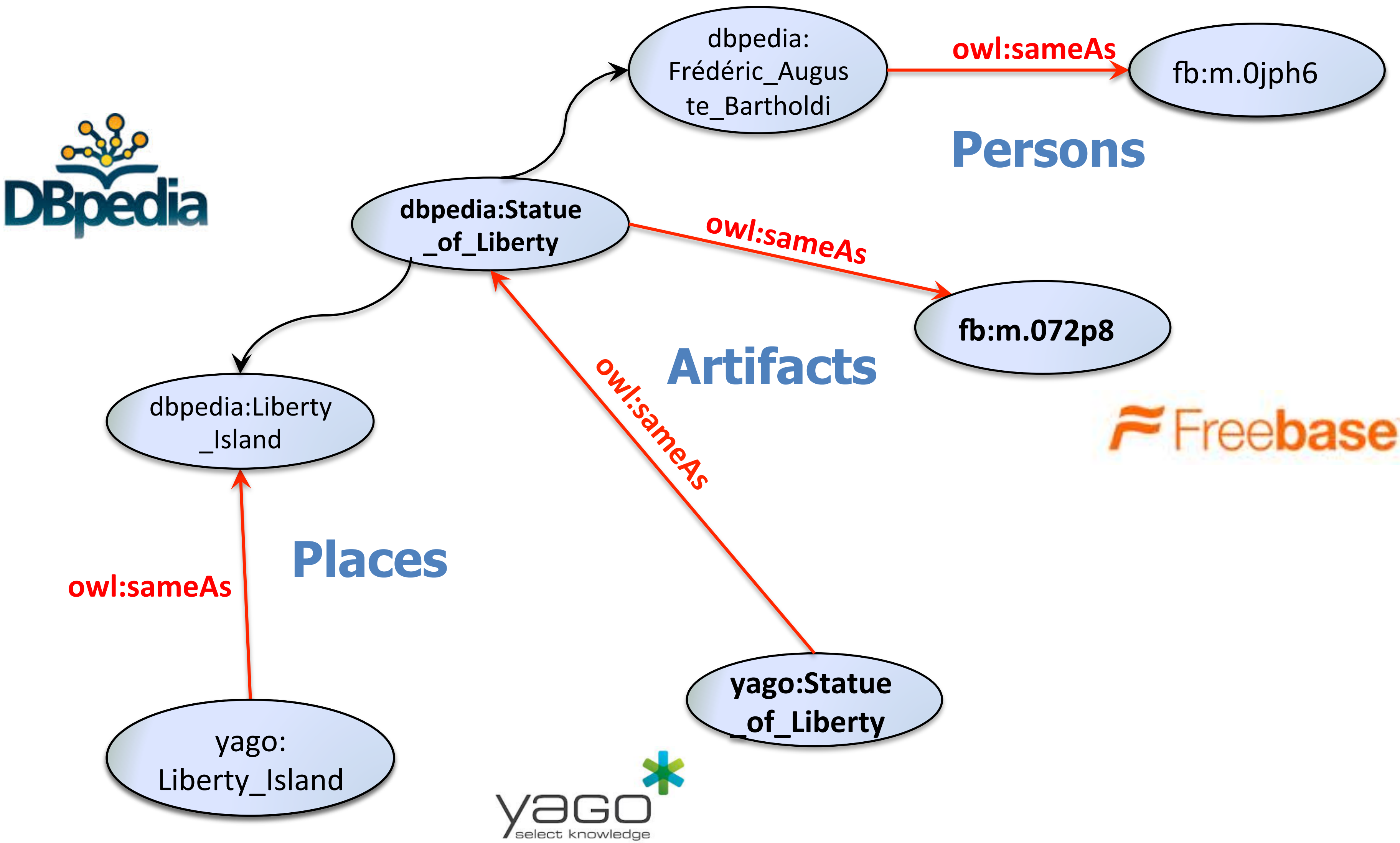
- Clean: duplicate-free
- Dirty: contains duplicate entity descriptions

An entity resolution task with input two entity collections can be:

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

An entity resolution task can also receive only one Dirty entity collection as input (aka the *deduplication* problem in databases)


dbpedia_category:
1886_sculptures
dbprop:
sculptor
dbpedia:
Frédéric_Auguste
_Bartholdi
dcterms:
subject
dbpedia:Statue
_of_Liberty
3200000
dbprop:visitationNum
dbprop:
location
dbpedia:Liberty
_Island
fb:m.0jph6
fb:architect
fb:m.072p8
fb:art_form
fb:m.06msq
yago:isLocatedIn
yago:Statue
_of_Liberty
rdf:type
yago:History_mu
seums_in_NY
yago:
Liberty_Island
rdf:type
skos:
prefLabel
yago:wasCreated
onDate
1886-##-##
Statue of Liberty
yago:
GeoEntity



⇒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 Description Example – Tabular Data

name	**Eiffel Tower**
architect	Sauvestre
year	1889
location	Paris **e1**

Table example

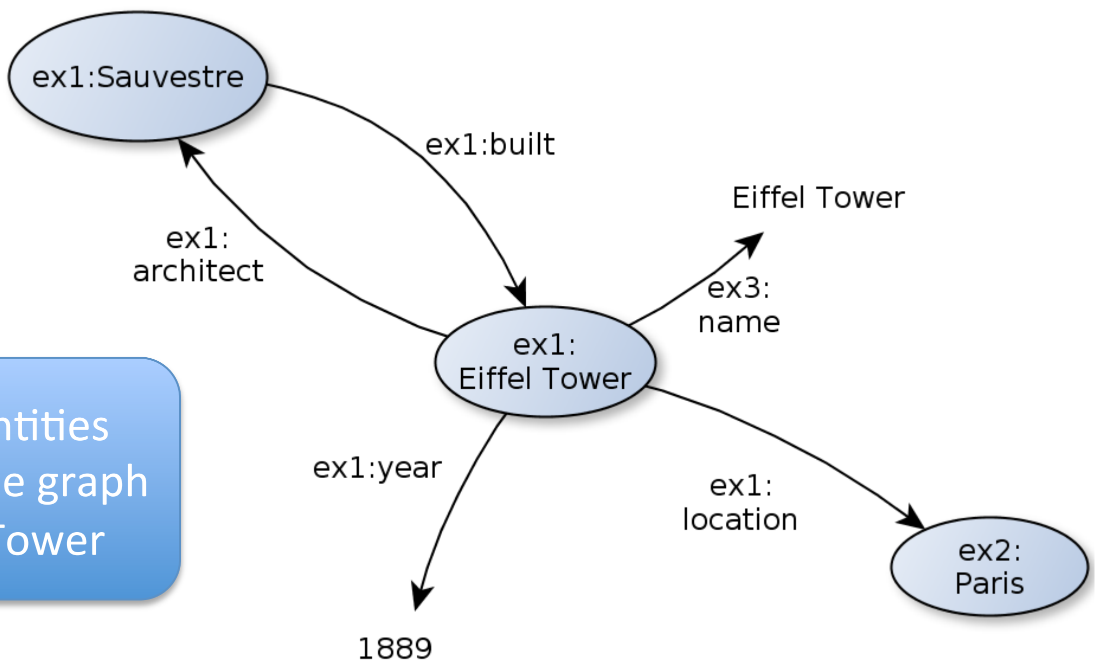
Name	Year	Architects	Location
Eiffel Tower	1889	Sauvestre	Paris

{(name, Eiffel Tower), (architect, Sauvestre), (year, 1889), (location, Paris)}

Entity Description Example – RDF Data

name	**Eiffel Tower**	
architect	Sauvestre	
year	1889	
location	Paris	**e1**

Graph example



one of the entities described in the graph is the Eiffel Tower

{(name, Eiffel Tower), (architect, Sauvestre), (year, 1889), (location, Paris)}

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:

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

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

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

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

all the matching descriptions are in the same partition

each partition contains only matching descriptions

Entity Resolution - Assumption

Our definition for entity resolution makes the assumption that each description represents exactly one entity

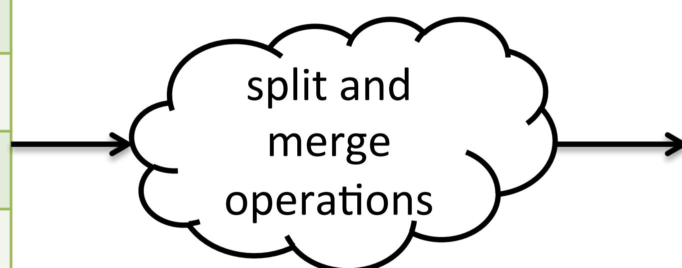
– *That is, a description is not decomposed to multiple descriptions*

Alternatively:

What if descriptions represent multiple entities?

– Employ split and merge operations on the descriptions

name	**Eiffel Tower**
year	1889
location	Paris
architect	Sauvestre
architect-birthday	26-10-1847 **e1**



name	**Eiffel Tower**
year	1889
location	Paris
architect	Sauvestre **e1**

architect	**Sauvestre**
architect-birthday	26-10-1847 **e2**

similar to normalization in relational databases?

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**

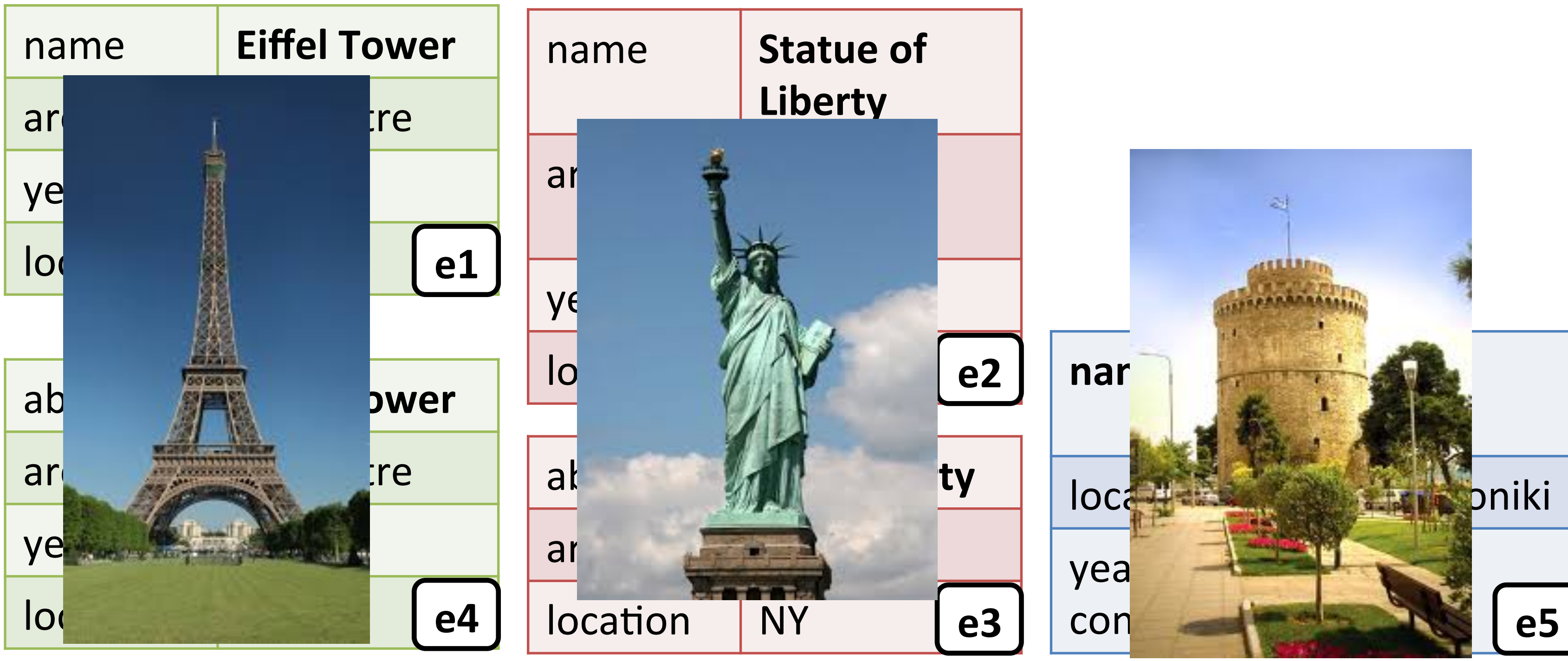
about	**Lady liberty**	
architect	Eiffel	
location	NY	**e3**

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



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 {true, false}

- $M(e_i, e_j)$ = true => e_i, e_j are matching descriptions
- $M(e_i, e_j)$ = false => e_i, e_j are non-matches

Entity Resolution - Similarity

Typically, the <u>match function</u> is expressed wrt. a similarity measure <u>sim</u>

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

Given a similarity threshold t:

– $M(e_i, e_j)$ = true, if $sim(e_i, e_j) \geq t$

– $M(e_i, e_j)$ = false, if $sim(e_i, e_j) < t$

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**

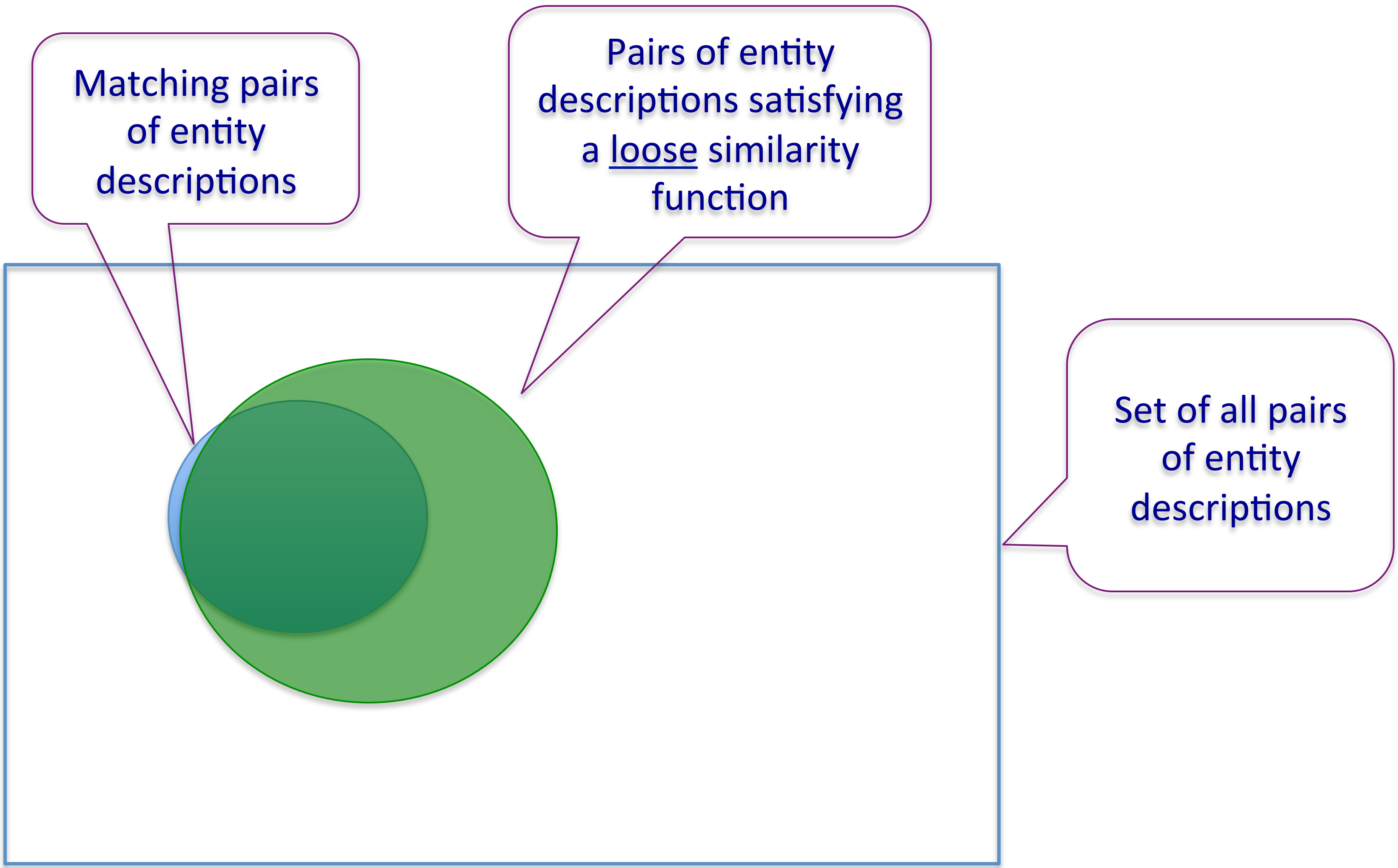
Exact match is rather impractical for entity resolution in the Web of data

- Too strict for a highly heterogeneous information space

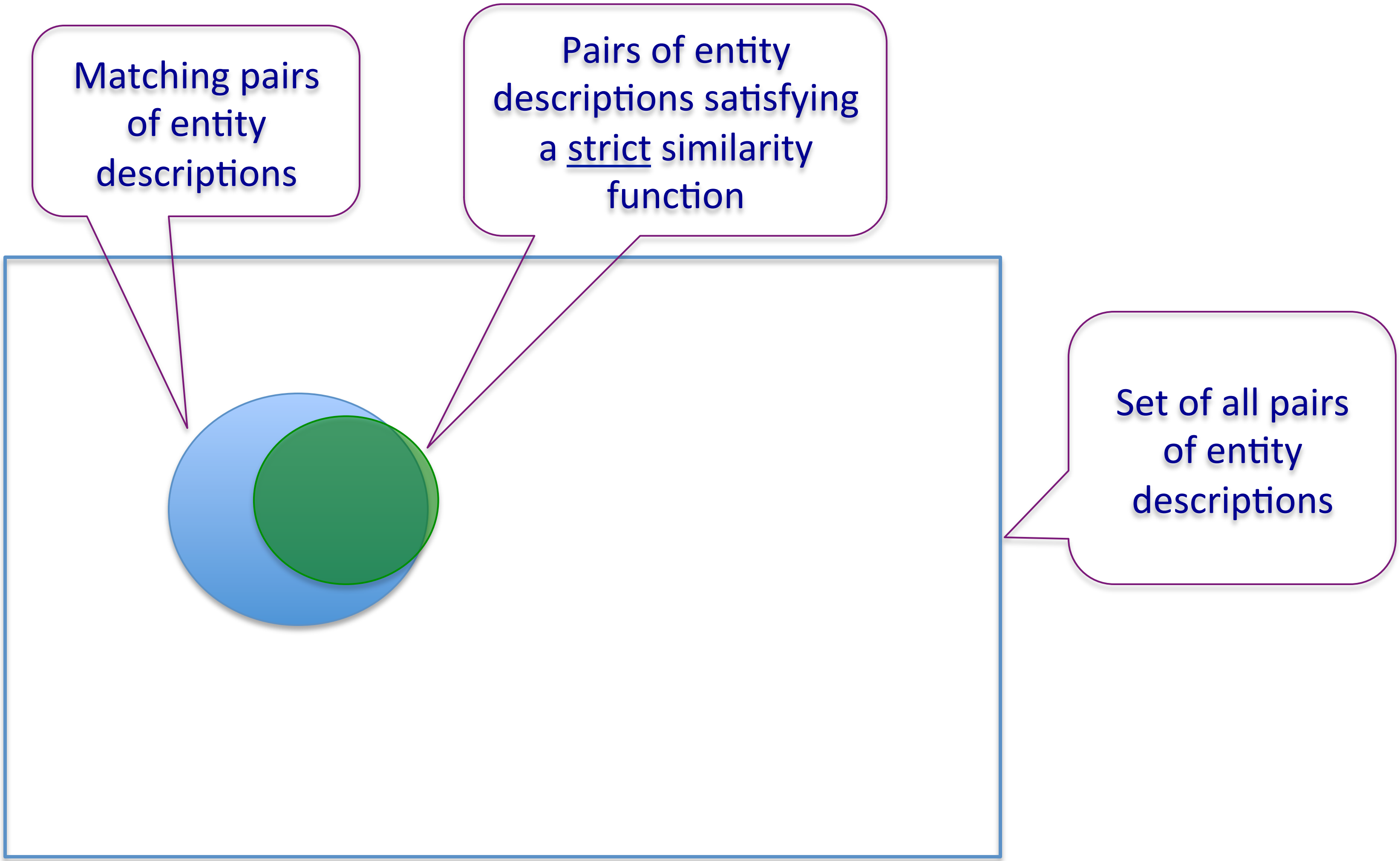
A more loose similarity measure could identify more matches, but…

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

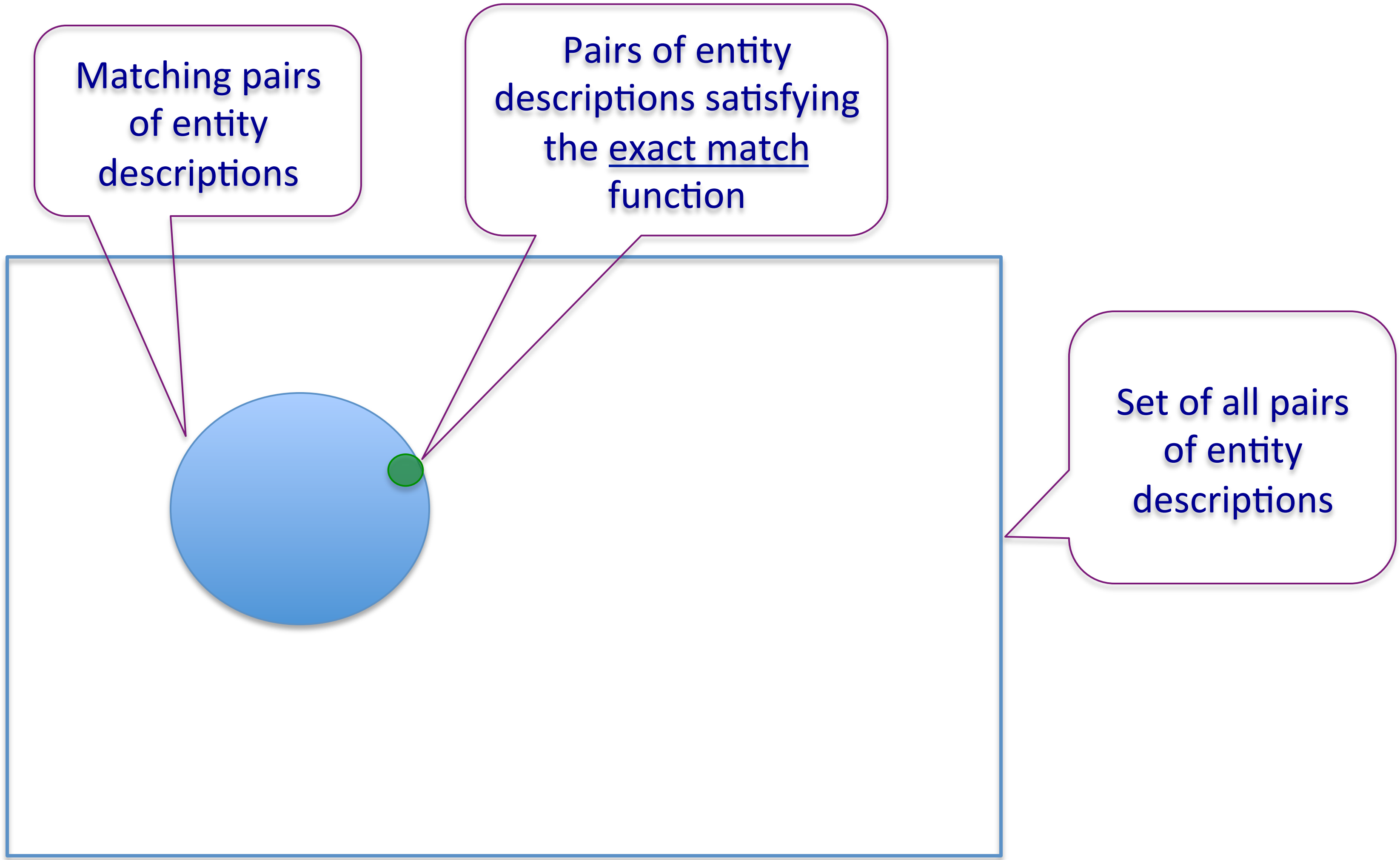
The Role of Similarity Functions – Loose Function



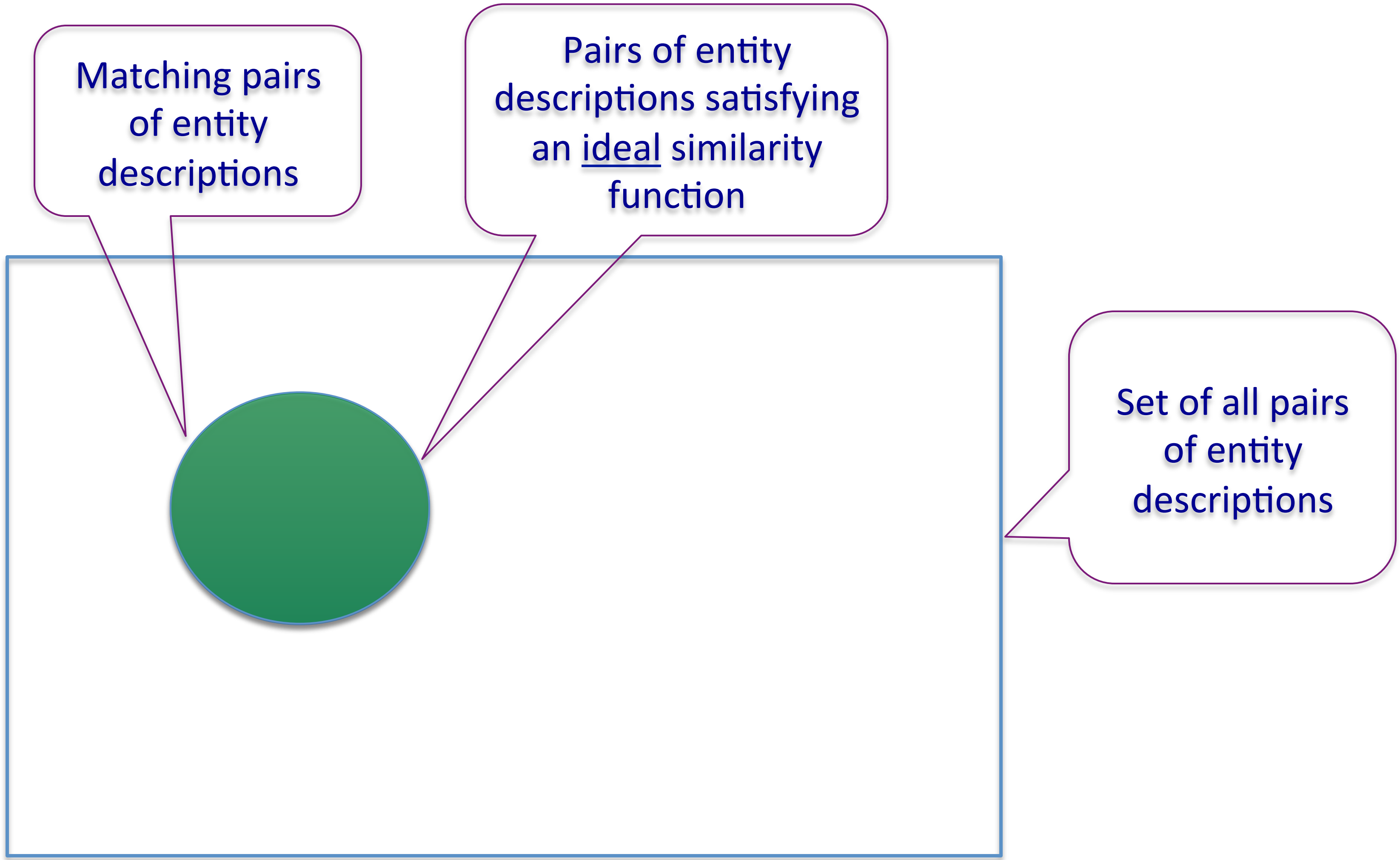
The Role of Similarity Functions – Strict Function



The Role of Similarity Functions – Exact Match



The Role of Similarity Functions – Ideal Case



Do the different forms of data influence the similarity computation complexity?

Data are published on the Web in multiple forms

Different forms of data have different degrees of structuredness, which influence the difficulty of entity resolution methods

Structuredness [Duan et al. 2011]

*Intuitively, the degree of **structuredness** for descriptions of the same semantic type T (e.g. buildings, architects) is determined by how much the descriptions conform to a common structural type for T within an entity collection E*

	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

semantic types

structural type

An entity description can have many semantic types, but only one structural type

Structuredness

name	Statue of Liberty
architect	Bartholdi
location	NY
height	93

e1

name	Eiffel Tower
architect	Sauvestre
location	Paris
height	324

e2

name	Lady liberty
architect	Eiffel
location	Paris
height	46

e3

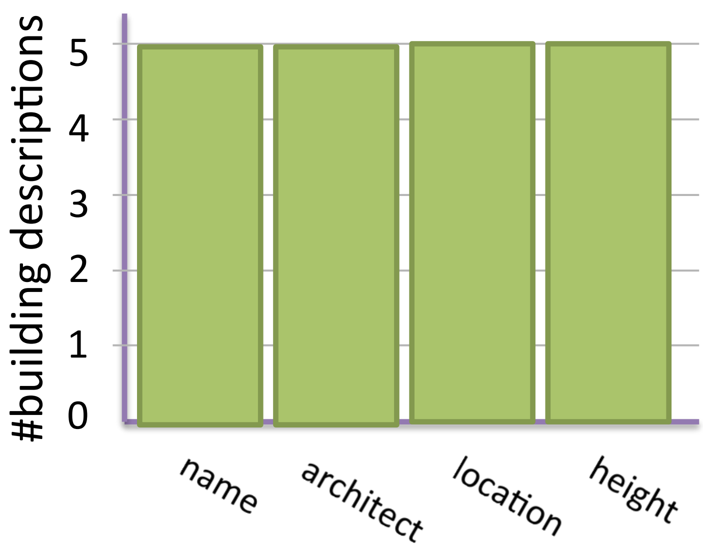
name	Eiffel Tower
architect	Sauvestre
location	Paris
height	324

e4

name	White Tower
architect	N/A
location	Thessaloniki
height	27

e5

A dataset of high structuredness, as in relational datasets



Structuredness

name	Statue of Liberty
architect	Bartholdi
location	NY

e1

name	Eiffel Tower
location	Paris

e2

name	Eiffel Tower
architect	Sauvestre
location	Paris

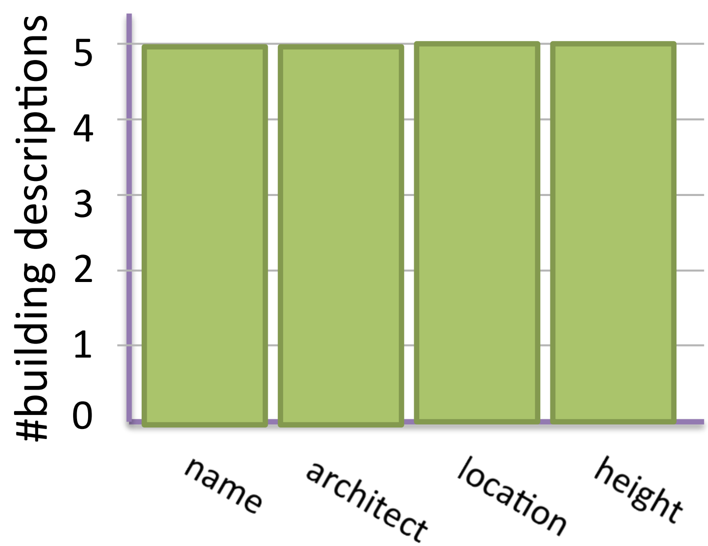
e4

name	Lady liberty
height	46

e3

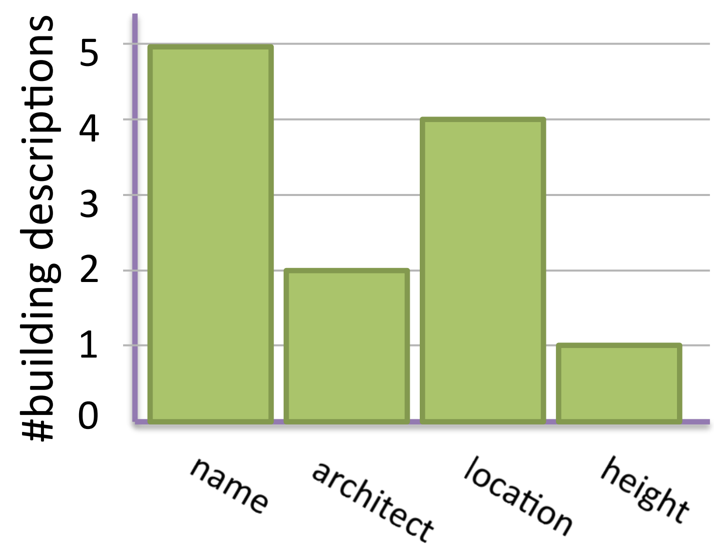
name	White Tower
location	Thessaloniki

e5



vs.

A dataset of lower structuredness, as in RDF datasets



Structuredness

Different forms of data have different degrees of structuredness, which influence the difficulty of entity resolution methods

Typically:

- Tabular data exhibit high structuredness

=> Comparing values of same attributes is enough

- Tree, graph data present varying structuredness

=> The problem becomes harder
e.g. XML sub-elements can be optional, RDF data typically do not follow a structural type

Solution Space

Solution Space

In general, entity resolution has been studied in a variety of contexts, using several approaches

Solution space wrt the type of method:

– Iterative methods: Identify matches that can lead to new matches
  - E.g. use the already merged descriptions

=> More matches

– Blocking methods: Group together descriptions close to each other
  - Rely on criteria for placing descriptions into blocks (blocking keys)

=> Less comparisons

– Learning methods: Use training data, annotated as matches or not
  - Classify descriptions, using statistical inference

Tutorial Overview

What follows in Part I:

- Iterative approaches
- Blocking approaches

Coffee break! ☺

Continuing with blocking in Part II

Iterative Approaches

Iterative Entity Resolution

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

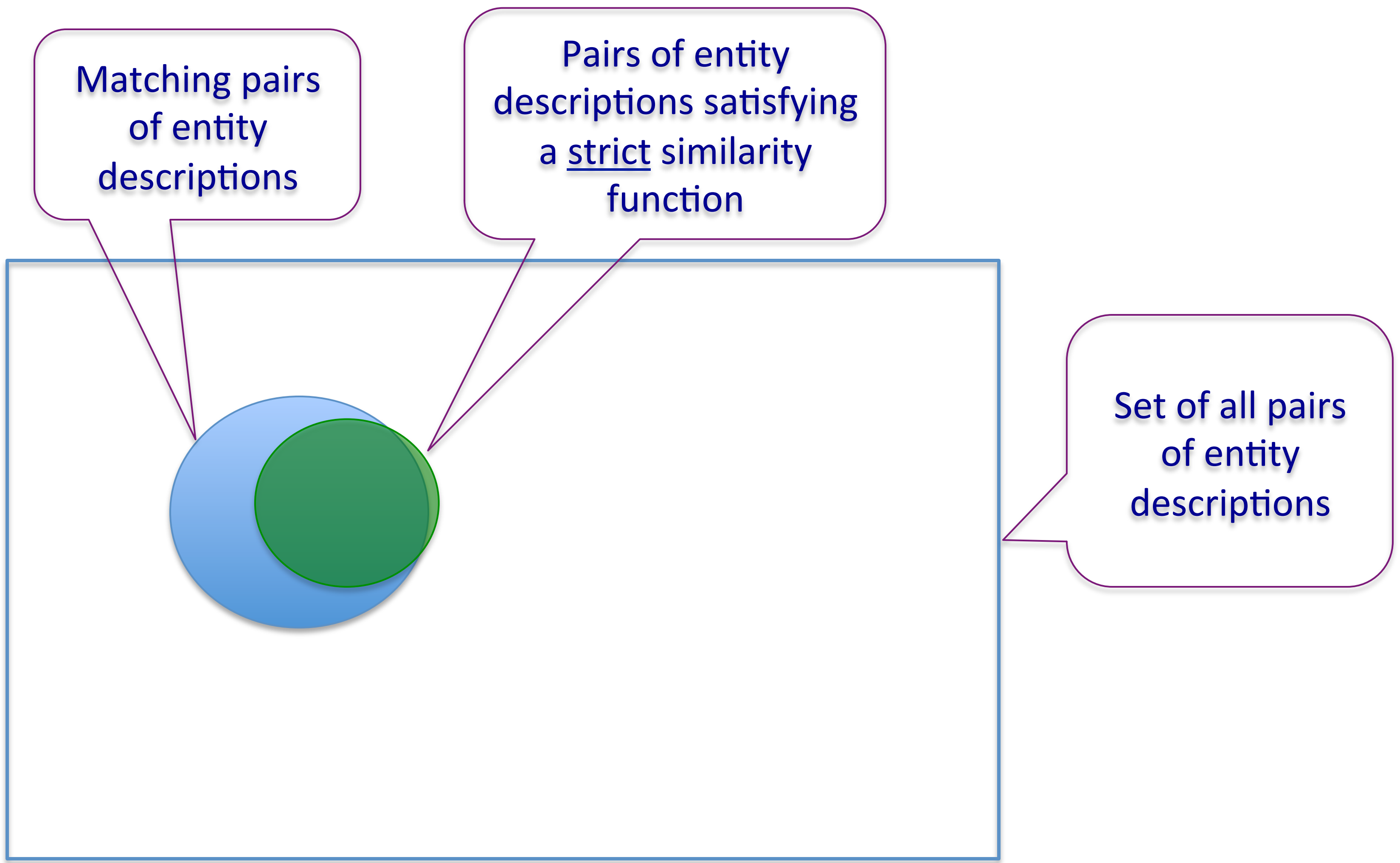
- Compare each entity description $d_i \in S$ with all other entity descriptions in S, i.e., with all $d_j \in S \setminus \{d_i\}$
- For comparison, use a classifier to classify each pair (d_i, d_j) as a duplicate pair
  - Based on similarity measures
  - Based on domain-specific rules
  - Based on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

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

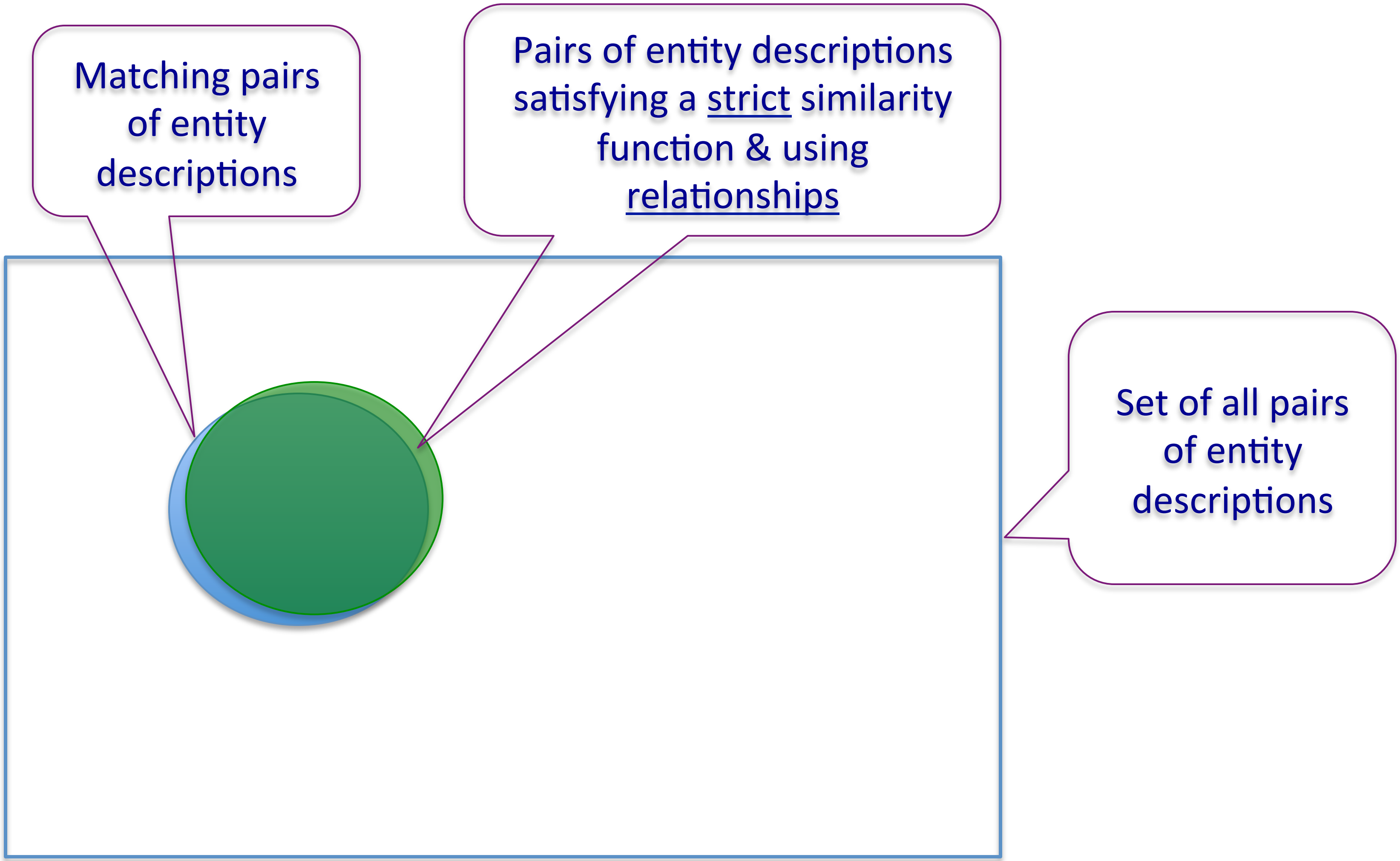
Iterative Entity Resolution

- Using relationships between duplicate classifications
  - Transitivity: If (A,B) are duplicates and (B, C) are duplicates, then (B,C) are also duplicates
  - Duplicate dependency: if entities Author1 and Author2 are duplicates, then related entities Publication1 and Publication2 are more likely to be duplicates than before the duplicate match of Author1 and Author2
  - Merge dependency: Once a duplicate pair has been identified, the unified (or more generally merged) entity representations create a new entity representation that should be compared to the remaining ones
- All these methods improve effectiveness by identifying more duplicate matches

Impact of Using Relationships



Impact of Using Relationships



Iterative Entity Resolution on Complex Data

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

Iterative Entity Resolution – Tabular Data

Table example

Name	Year	Architects	Location
Eiffel Tower	1889	Sauvestre	Paris
↕ typo		↕ missing value	↕ conflicting values
Eifel Tower	1889	NULL	France

- Input:
  - A relation with *N* tuples
  - A similarity measure
- Output:
  - Classes (clusters) of equivalent tuples (= duplicates)
- 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 duplicate

Sorted Neighborhood Method

- Idea
  - Create partitions
  - Perform comparisons only within a partition
- The initial algorithm [Hernández & Stolfo 1995]

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

Sorted Neighborhood Method – Key Generation

- Key: for a given tuple t, its key consists of a sequence of attribute value substrings taken from t
- Quality of entity resolution strongly depends on the choice of the key
- The key is virtual and is not necessarily unique
  - It only serves to sort the tuples

Sorted Neighborhood Method – Sort Phase

- Sort tuples according to the lexicographic order of their generated keys

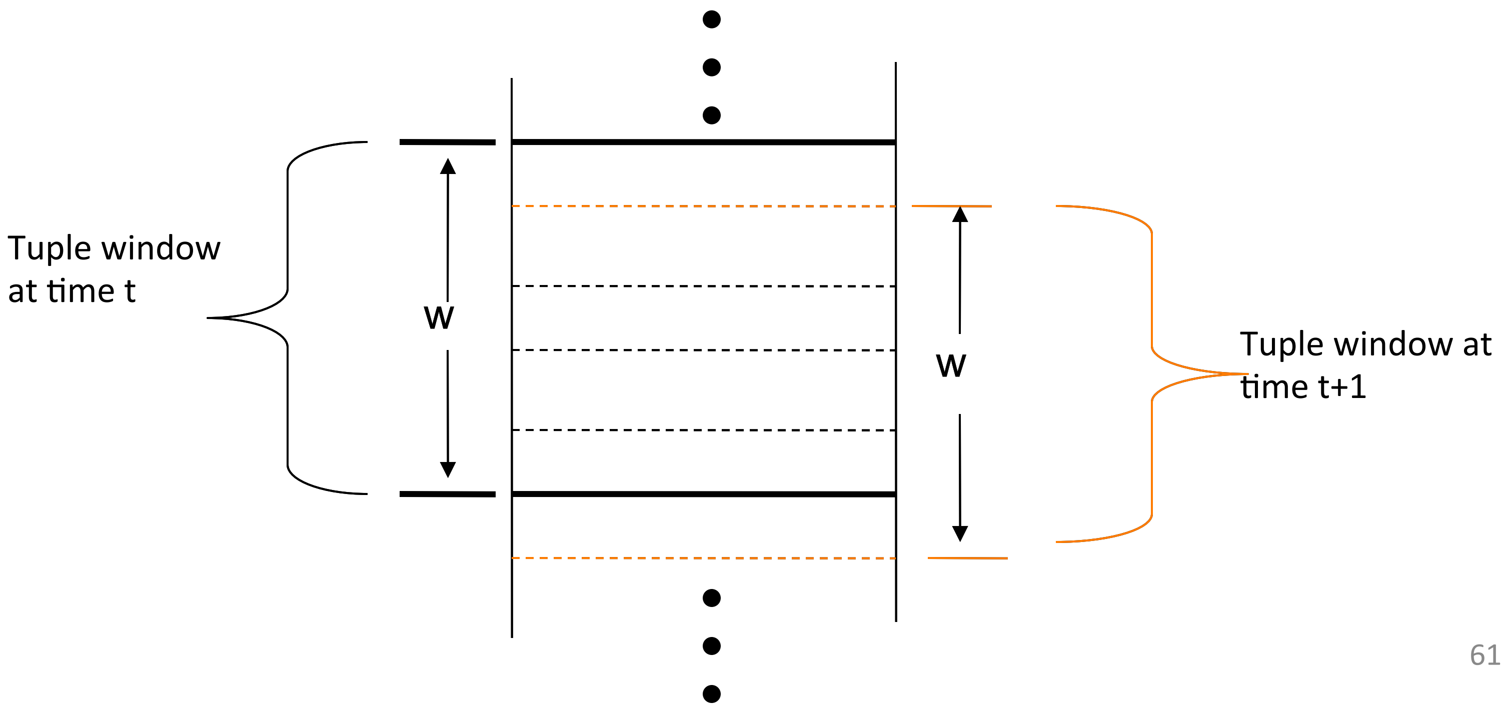
- Goal:
  Equivalent tuples (duplicates) are sorted close to each other

- Different sorting strategies (Quicksort, AlphaSort, etc.)

  – For scalability, use of a DBMS for efficient secondary memory access (two passes over the data)

Sorted Neighborhood Method – Merge Phase

- A window of predefined fixed size w goes over the sorted data.
- $2 \leq w \leq N$
- Only compare tuples that fall in the same window
- [Hernández & Stolfo 1995] proposes a rule-based classifier to detect duplicates, but any similarity measure can be used as well



Sorted Neighborhood Method - Discussion

- Complexity
  - N : number of tuples
  - w: window size
  - In theory:
    - $O(N) + O(N \log N) + O(w N) = O(N \log N)$ when $w < \log N$;
    - $O(wN)$ otherwise
  - In practice:
    - Three scans of the relational data stored on disk

Iterative Entity Resolution – Tabular Data

- Extensions to the Sorted Neighborhood Method
  - Multi-Pass Sorted Neighborhood Method [Hernández & Stolfo 1995, 1998]
  - Sorted-Neighborhood for XML data [Puhlmann et al. 2006]
  - Automatic adjustment of the window size [Yan et al. 2007, Draisbach et al. 2012]
- Identifying additional duplicates:
  - Transitive closure is commonly applied over the set of detected duplicate pairs to obtain clusters of duplicates
  - Some methods [Benjelloun et al. 2009] merge descriptions of duplicates and re-evaluate the similarity of these to other descriptions
- Other means to reduce complexity by saving pairwise comparisons
  - Blocking (partitioning w.r.t. one or more attribute values, see next part of the tutorial)
  - Recall-preserving filter functions (upper / lower bound for distance / similarity measures) [Ananthakrishna et al. 2002, Weis & Naumann 2004]

Swoosh [Benjelloun et al. 2009]

A generic approach for entity resolution in tabular data

Black-boxes:

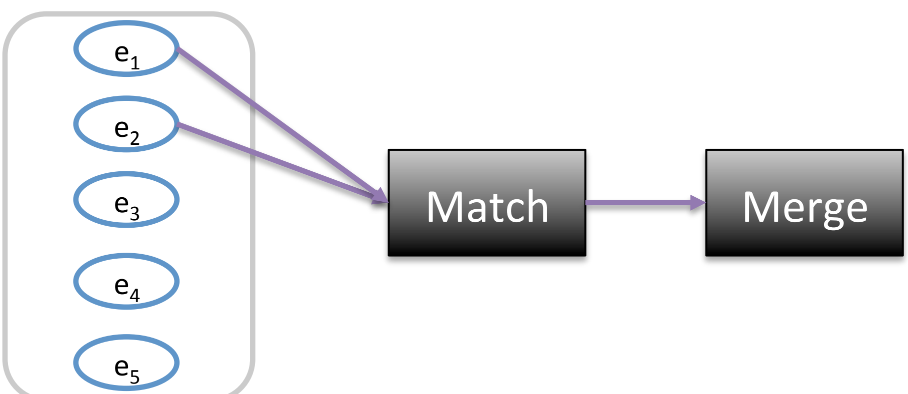
- A match function M
- A merge function μ

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:

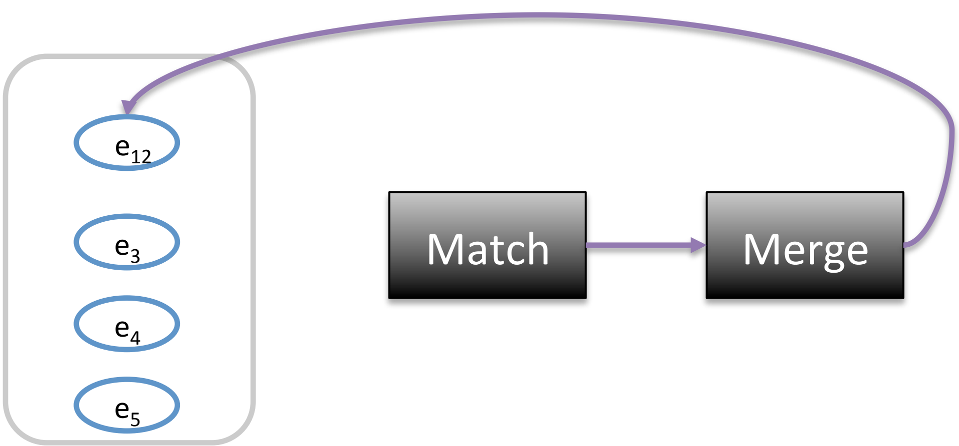
- A match function M
- A merge function μ

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:

$M(e_1, e_1) = true$ and $\mu(e_1, e_1) = e_1$

- Commutativity:

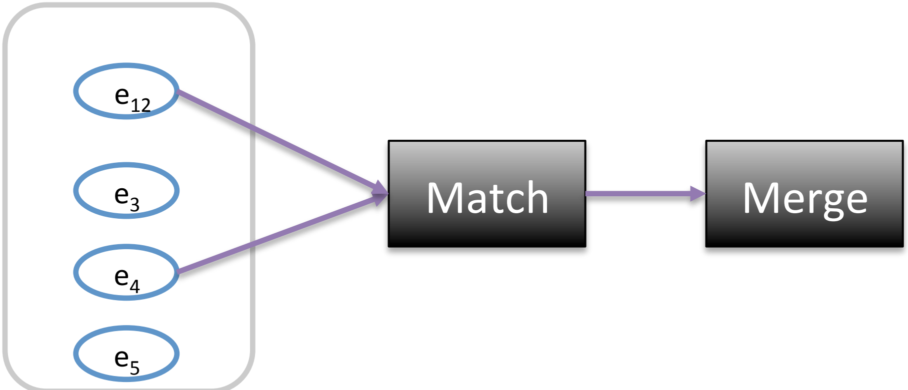
$M(e_1, e_2) = M(e_2, e_1)$ and $\mu(e_1, e_2) = \mu(e_2, e_1)$

- Associativity:

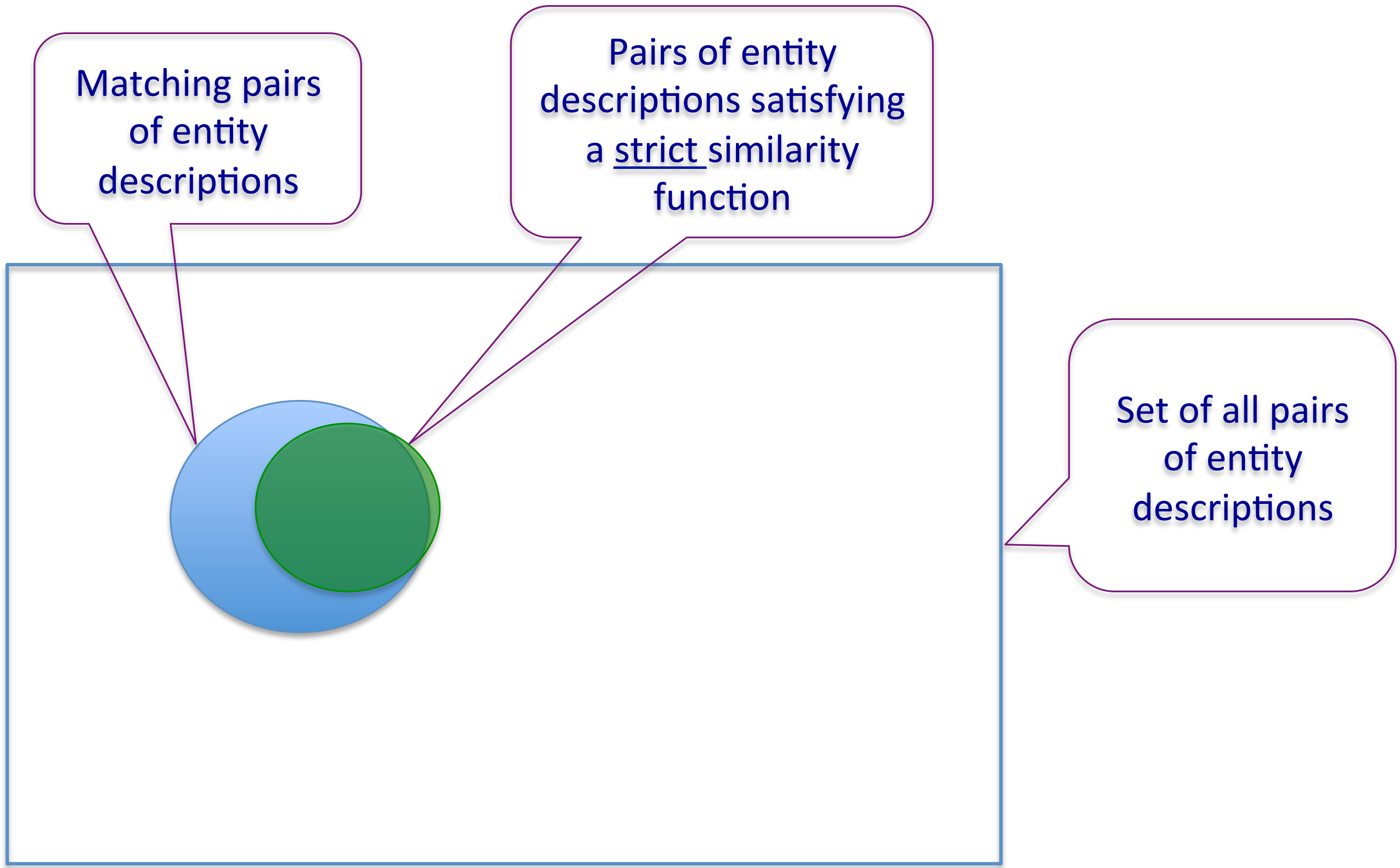
$\mu(e_1, \mu(e_2, e_3)) = \mu(\mu(e_1, e_2), e_3)$

- Representativity:

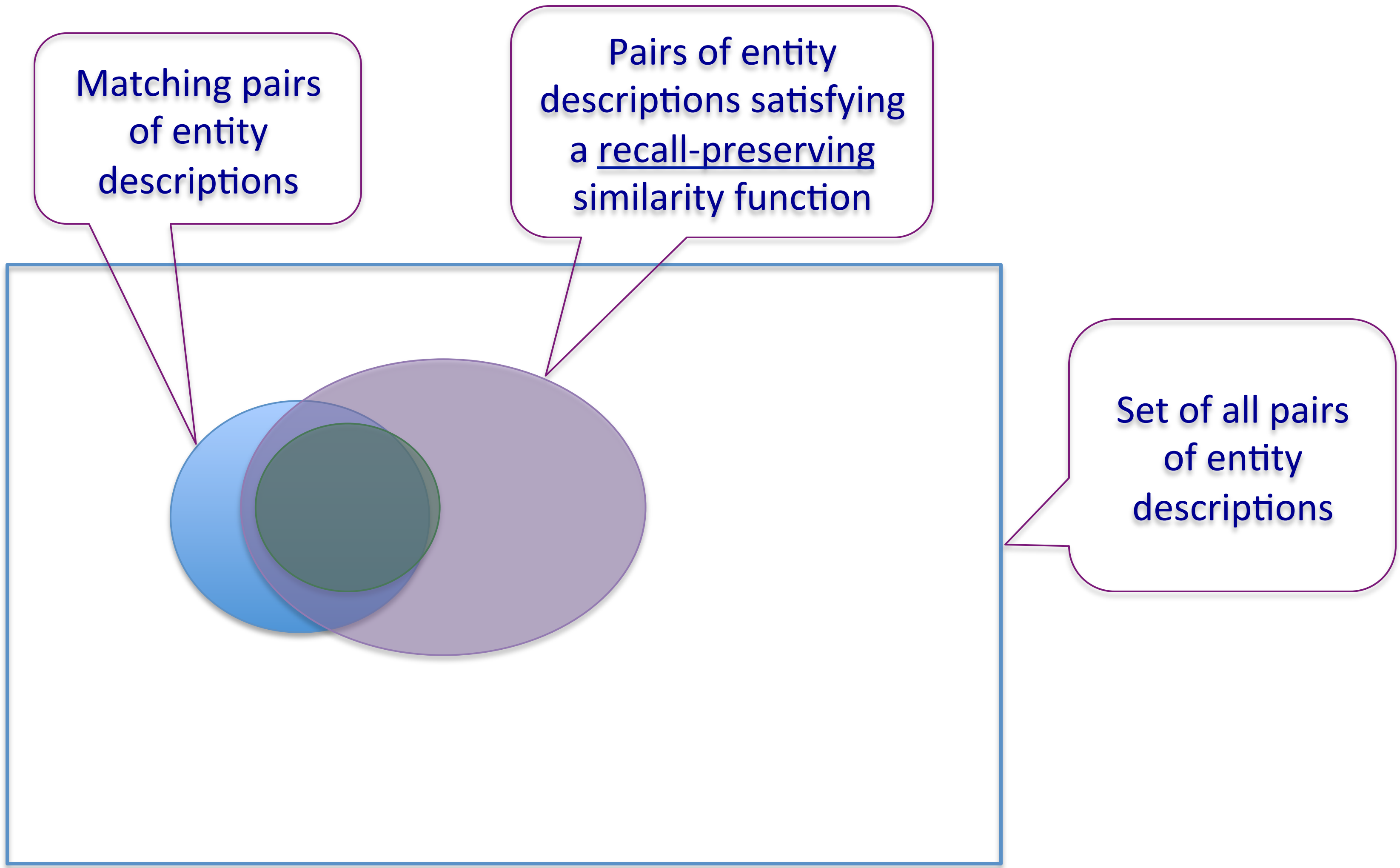
if $\mu(e_1, e_2) = e_3$ and $M(e_1, e_4) = true$, then $M(e_3, e_4) = true$



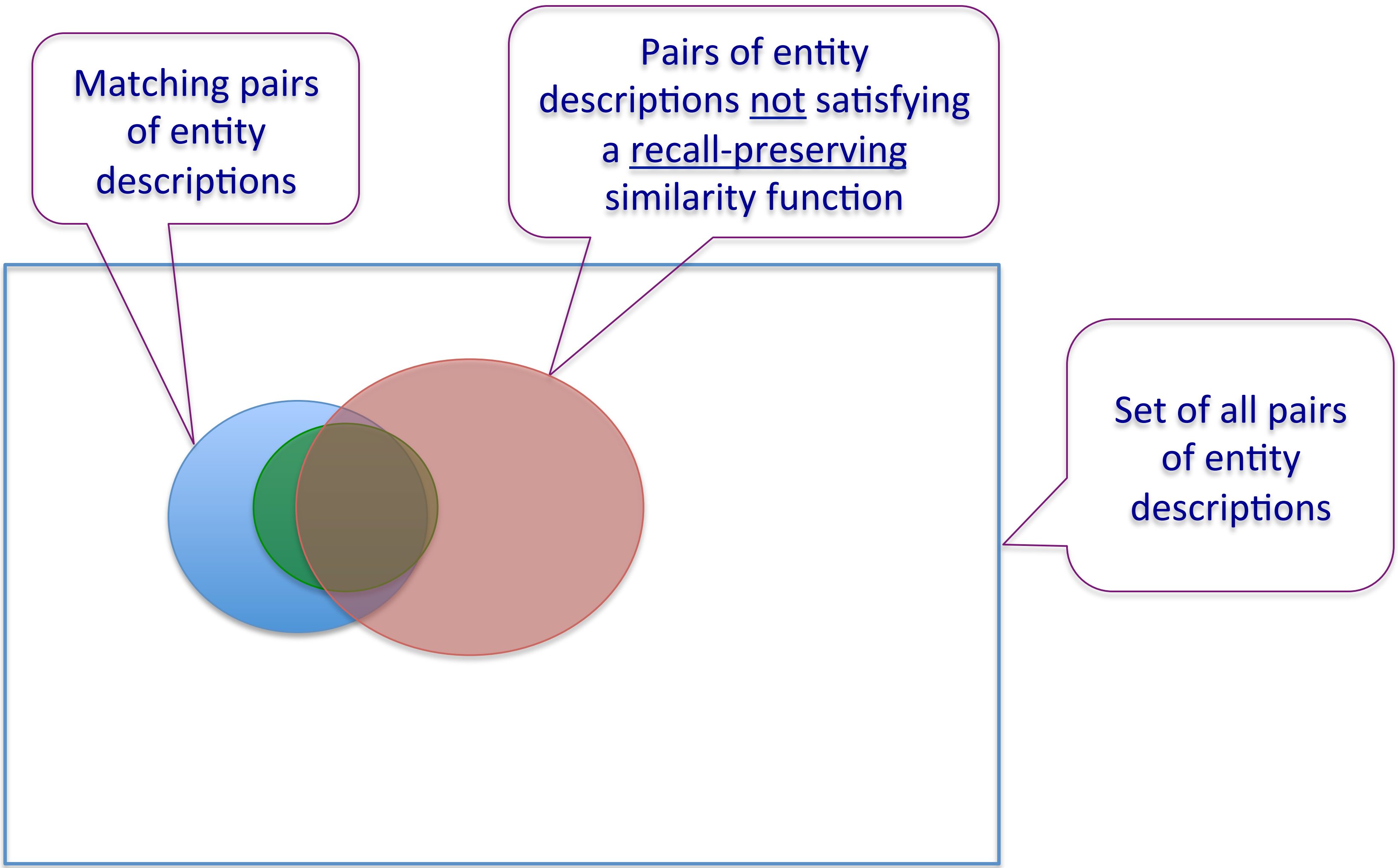
Recall-Maintaining Filter Functions



Recall-Maintaining Filter Functions



Recall-Maintaining Filter Functions



Iterative Entity Resolution – Tree Data

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

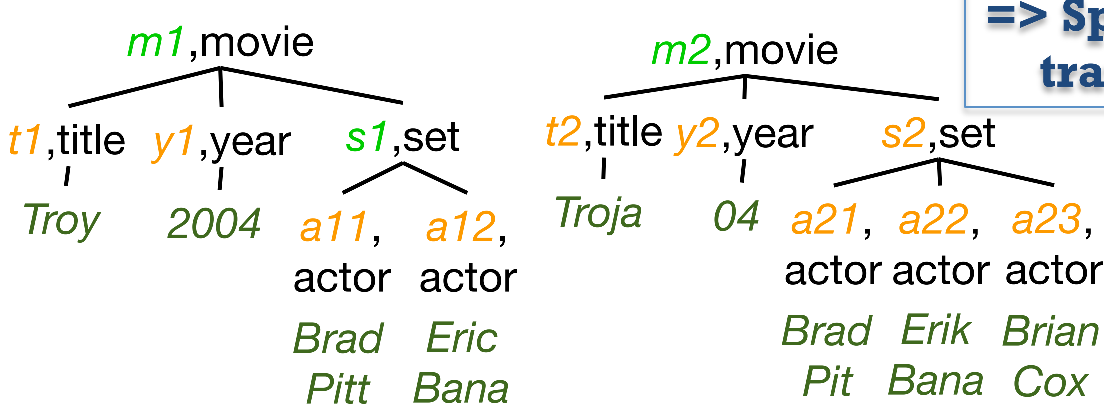
ID	Actor	Film
S1	Al Pacino	F1
S2	Al Pacino	F2
S3	Marlon Brando	F2

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 → token sets *TS*
  - Compute the edit distance between token sets
  - Determine weight of each token using IDF [Baeza-Yates & Ribeiro-Neto1999]
  - 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:
    pos(x) = 1 if x > 0,
    -1 otherwise
  - Threshold for *tcm*: *s1*
  - *Threshold for fkcm: s2*
  - Classification of pairwise comparison between *T* and *T'* using

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

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

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) = {The, Godfather, 1972, 9.2}
TS(F2) = {Gottvatter, The, 72}

2. Attribute similarities
The = The, Godfather = Gottvatter, 1972 = 72.

3. Weights
For simplification, we assume all tokens have equal weight.

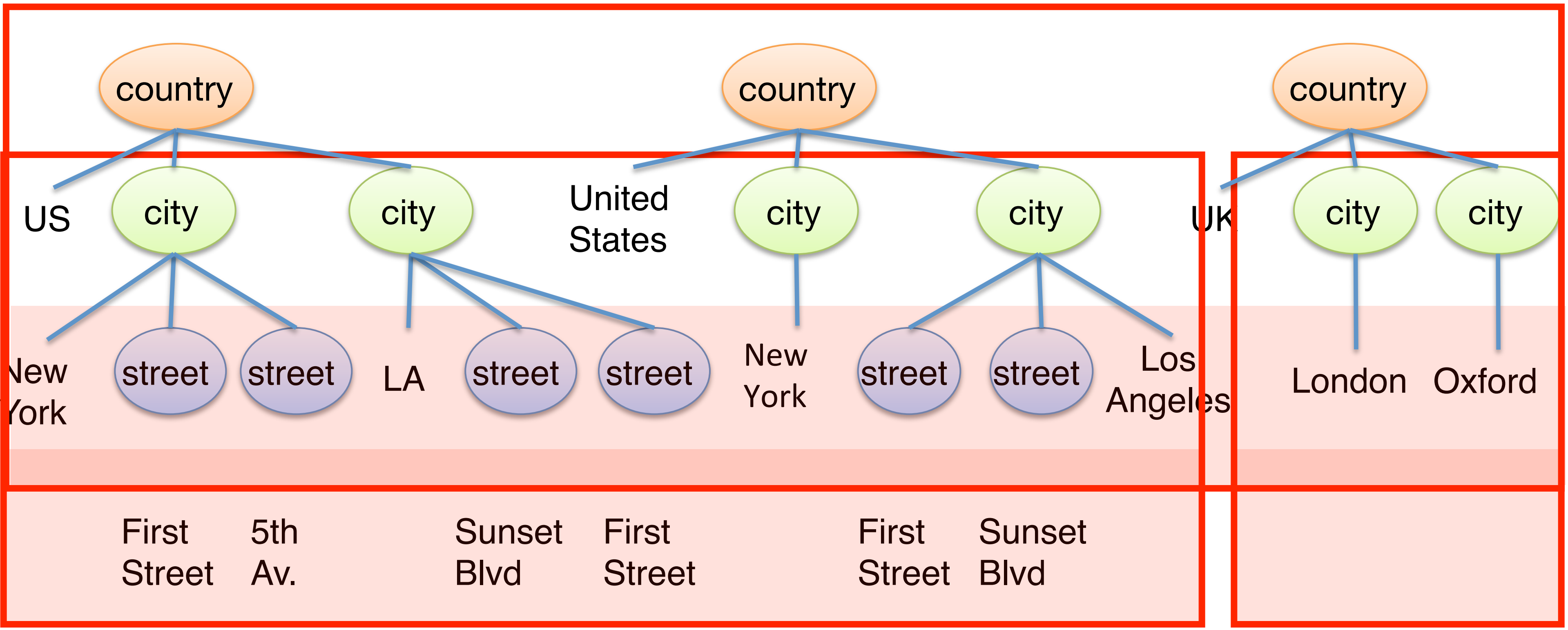
4. Token containment metric
tcm(F1,F2) = ¾ , tcm(F2,F1) = 1

5. Children co-occurrence
fkcm(F1,F2) = 1, *fkcm(F2,F1)* = ½

6. **Combination of both metrics**
(*s1* = *s2* = 0.5, weights = 1)
pos(pos(3/4 - 0.5) + pos(1 - 0.5) = 1
→ F1 and F2 duplicates

Top-Down Algorithms

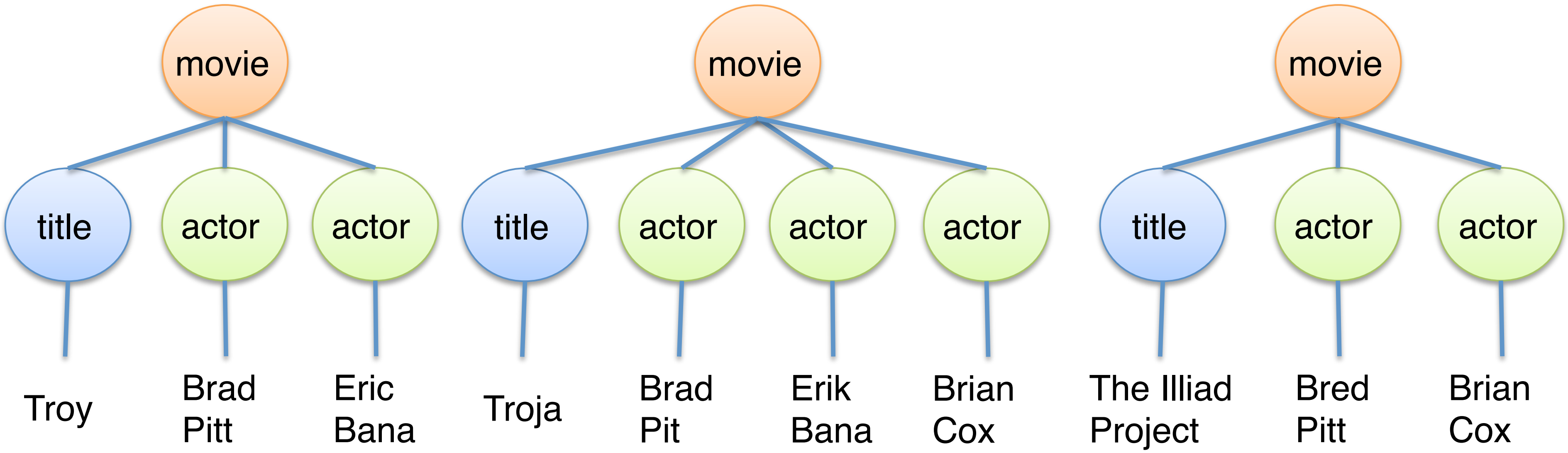
Top-Down Algorithms [Ananthakrishna et al. 2002, Weis & Naumann 2004]



2. Only search for duplicates among children with common ancestor

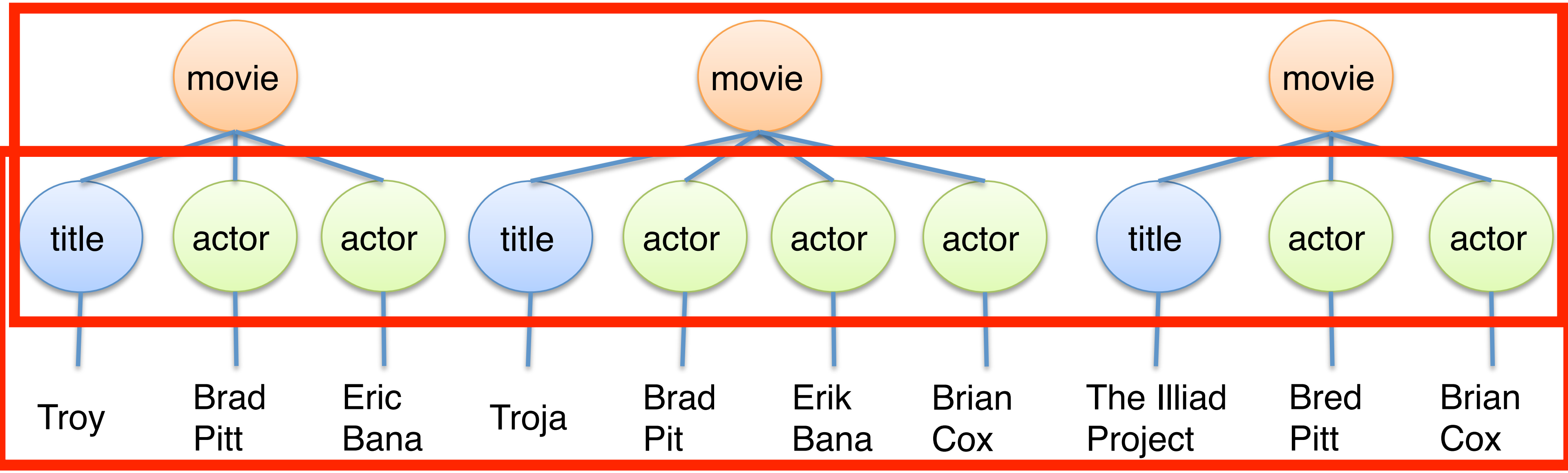
Bottom-Up Algorithms

Bottom-Up Algorithms [Puhlmann et al. 2006, Leitão et al. 2007, Leitão et al. 2013]



Bottom-Up Algorithms

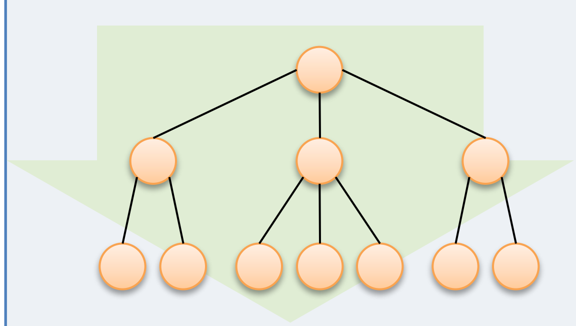
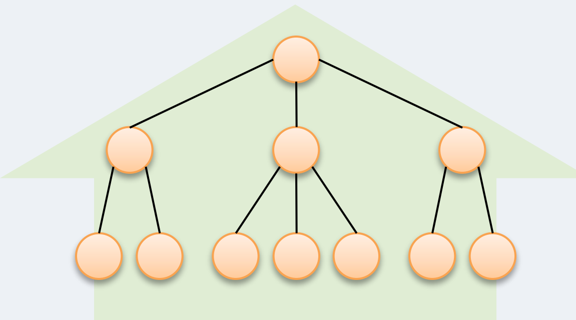
Bottom-Up Algorithms [Puhlmann et al. 2006, Leitão et al. 2007, Leitão et al. 2013]



2. Propagate duplicate decisions to parent level and perform comparisons one level up, taking into account identified child duplicates (e.g., propagate similarities that reflect duplicate probability through a Bayesian Network [Leitão et al. 2007])

Iterative Entity Resolution – Tree Data

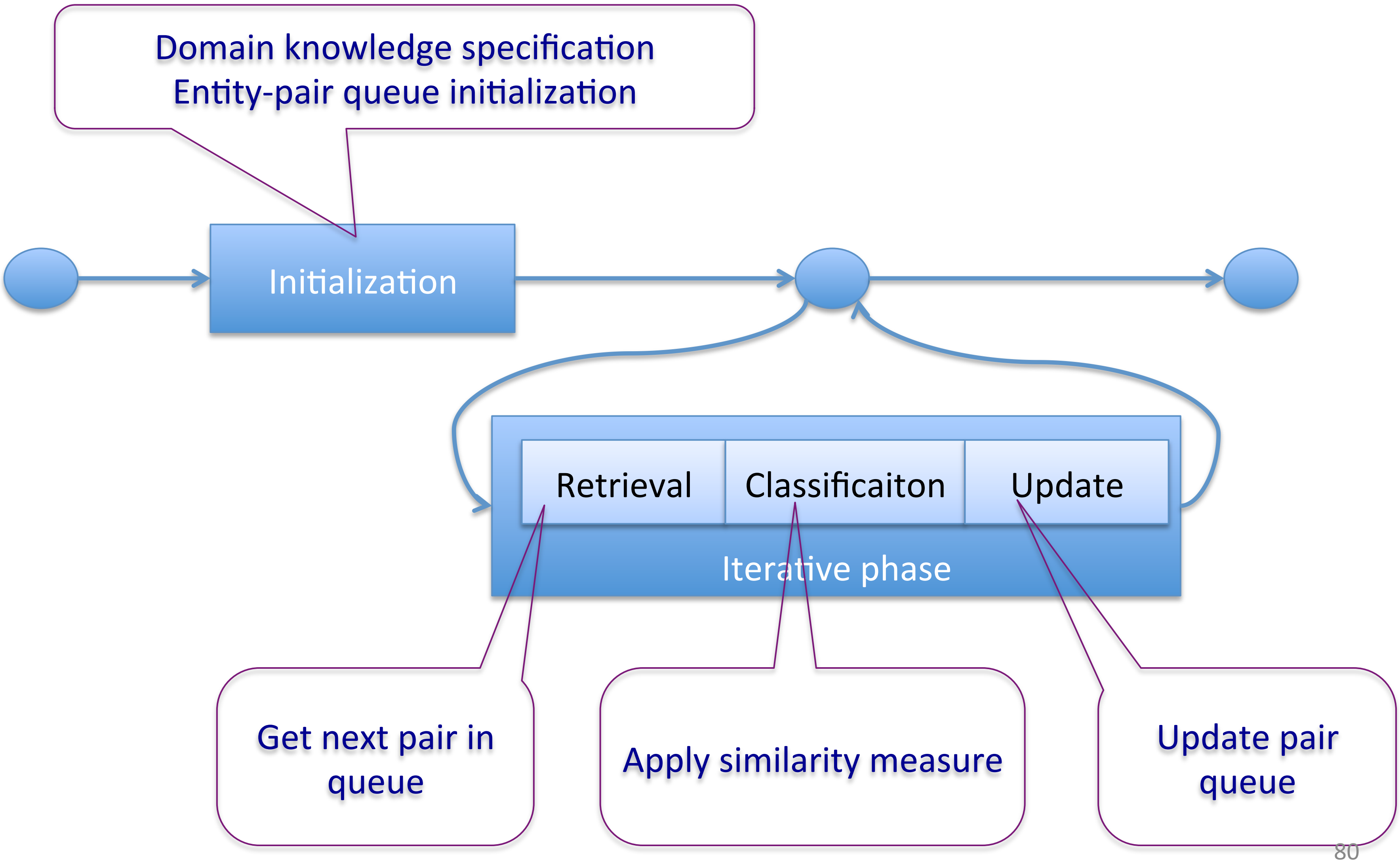
- Similarity measures
- Consider attribute <u>values</u> (text value data)
- Consider similarity of <u>children sets</u> (referencing tuples or child XML elements)

Top-down [WN04]	Bottom-up [PWN06]
	
Hierarchy represents 1:N relationships between candidate entities.	Hierarchy represents candidate entities that can be in M:N relationship.
Prune comparisons of descendants that do not have same or similar ancestors.	Sorted neighborhood method applied to every candidate type.

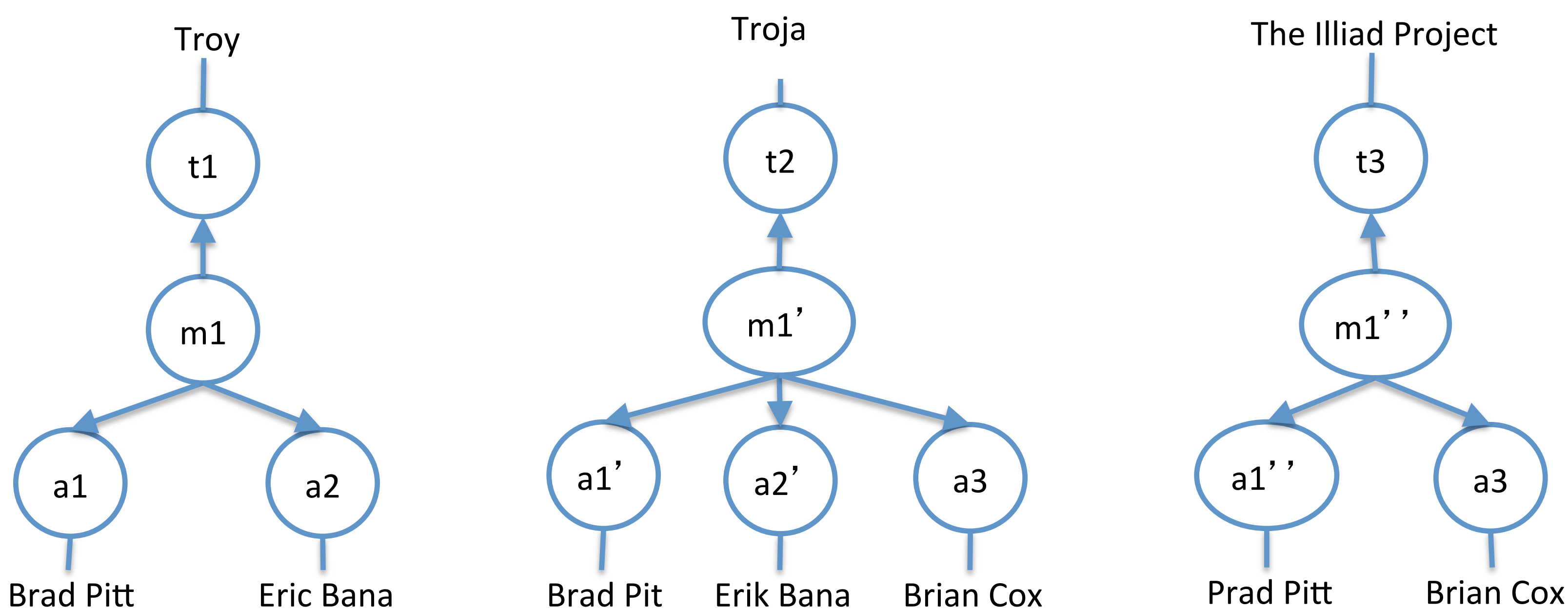
Iterative Entity Resolution - Graph Data

- 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

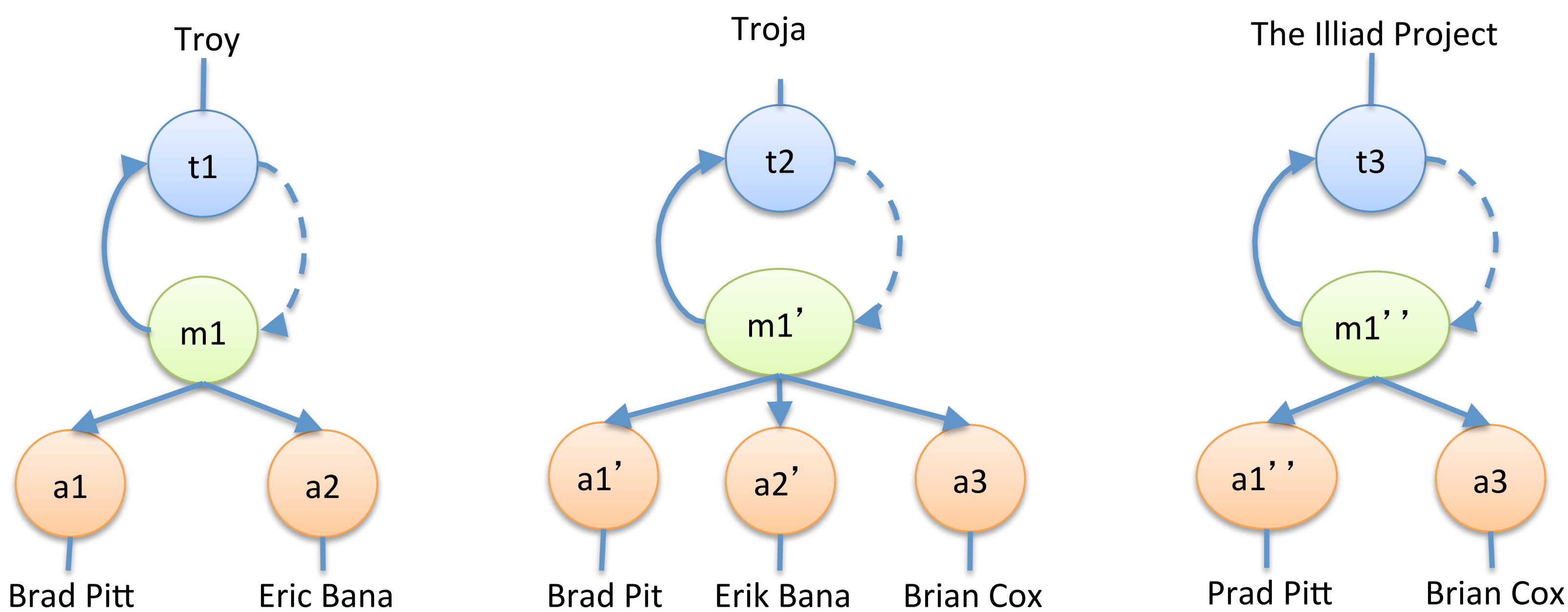


Domain Expert Knowledge Specification



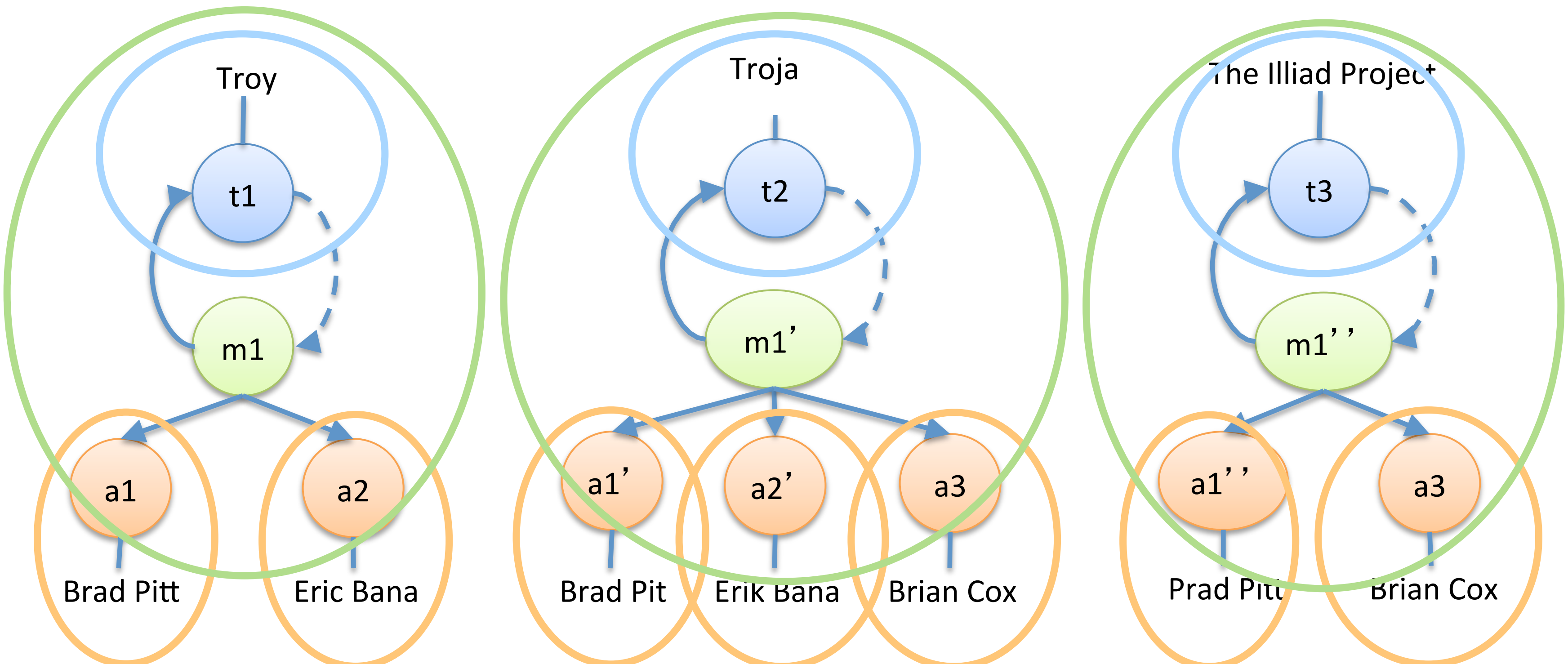
- Domain expert specifies
  - Duplicate candidate entities (e.g., movie, actor, title)
  - (Additional) relationships between candidates (e.g., title → movie)

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

Similarity Measure Template

weight functions
(e.g., IDF)

Set of similar
entity descriptions

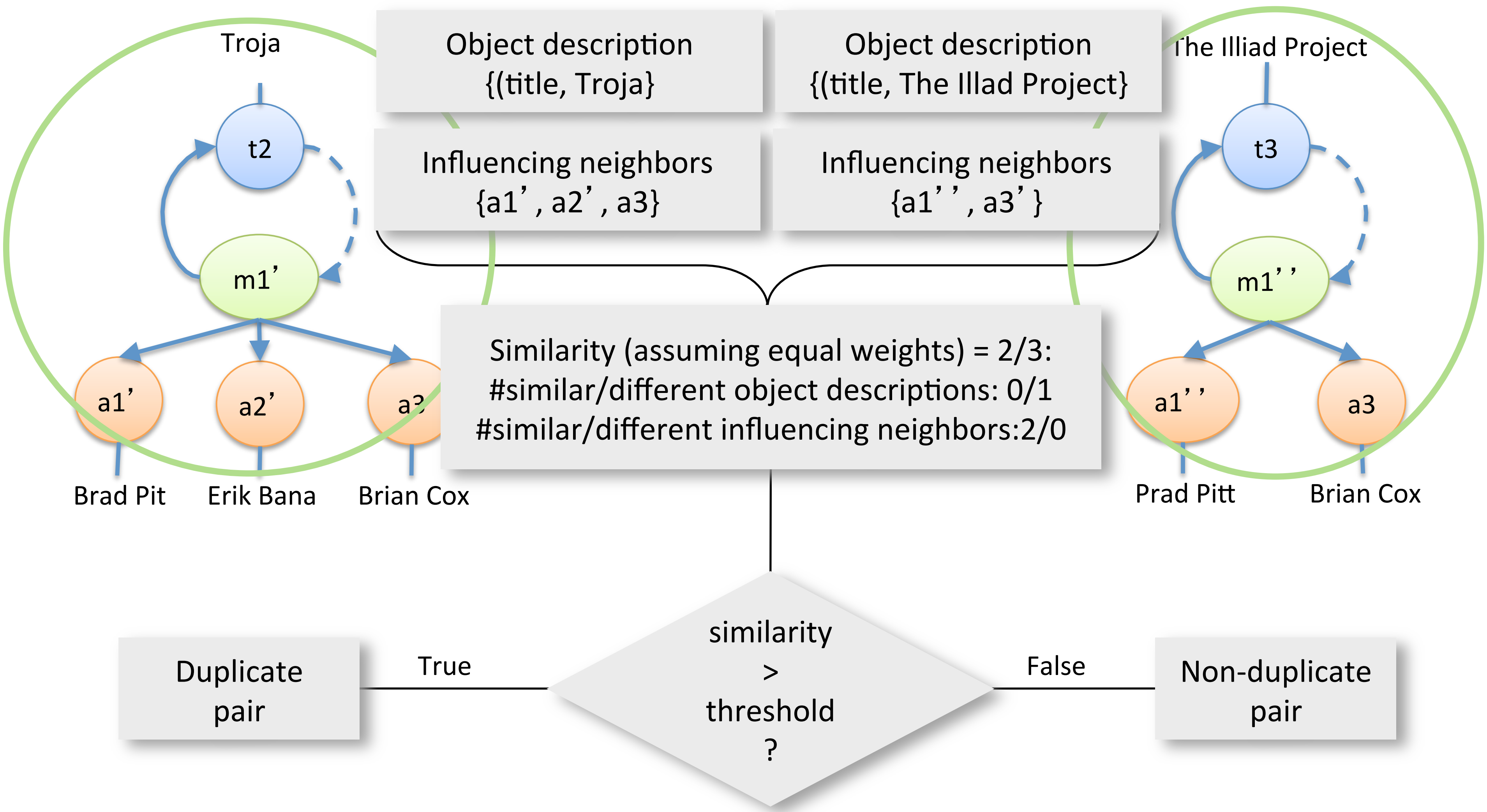
Set of similar
influencing neighbors

$$sim(c, c') = \frac{w_{od}(N_{od}^{\approx}) + w_{ip}(N_{ip}^{\approx})}{w_{od}(N_{od}^{\approx}) + w_{ip}(N_{ip}^{\approx}) + w_{od}(N_{od}^{\neq}) + w_{ip}(N_{ip}^{\neq})}$$

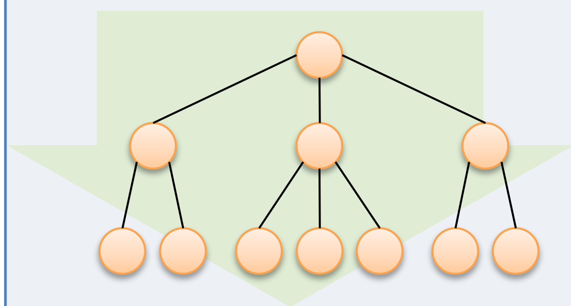
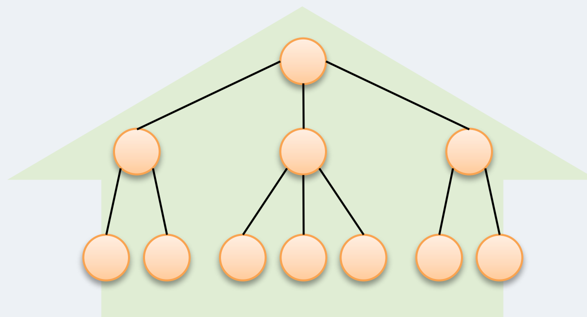
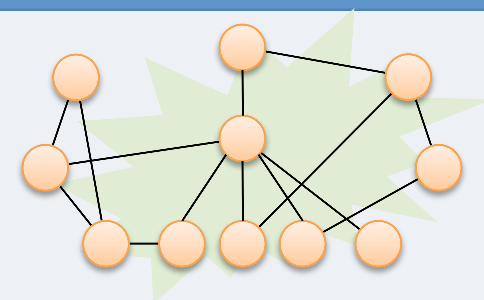
Set of different
entity descriptions

Set of different
influencing neighbors

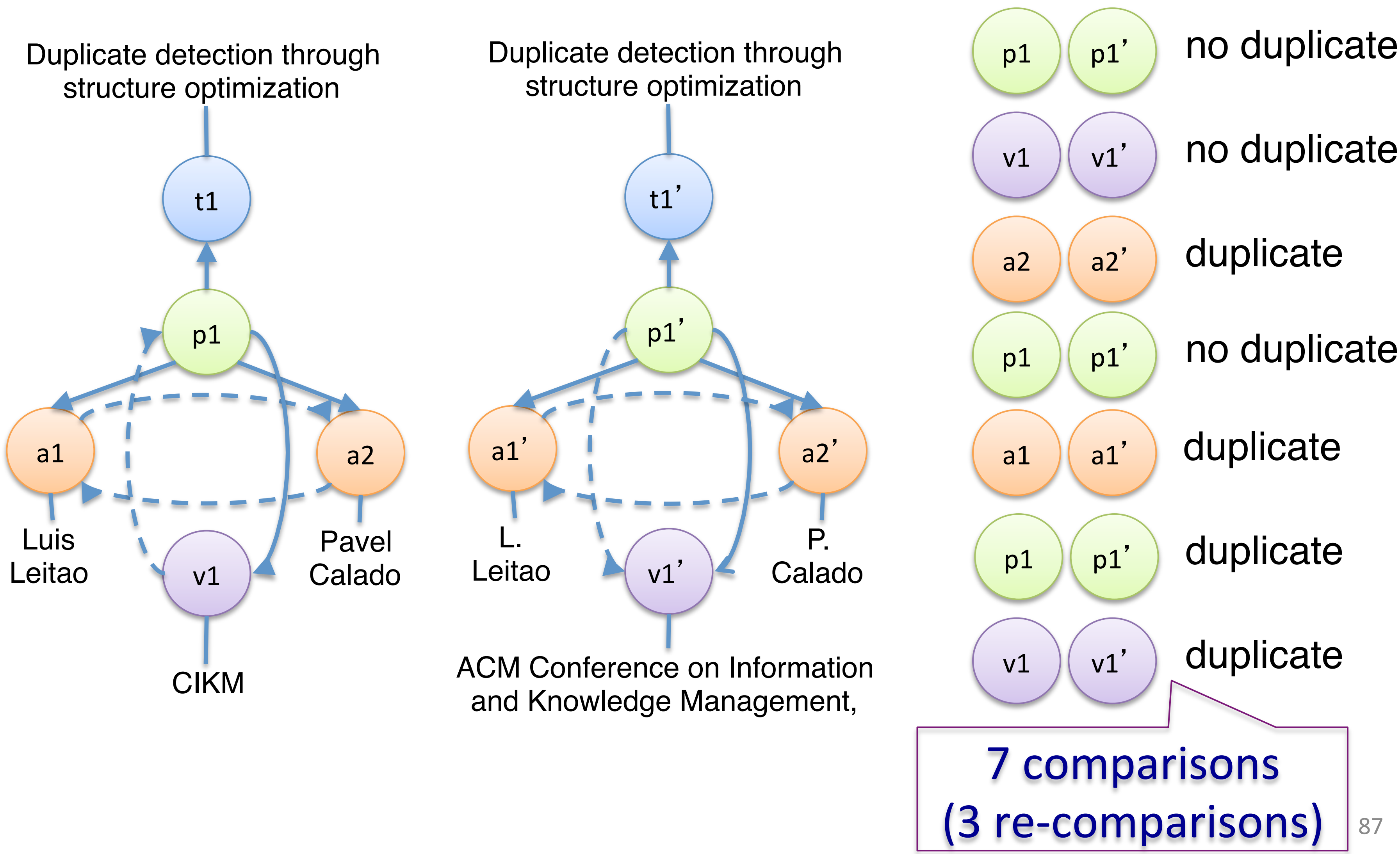
DogmatiX Similarity Measure [Weis & Naumann 2005]



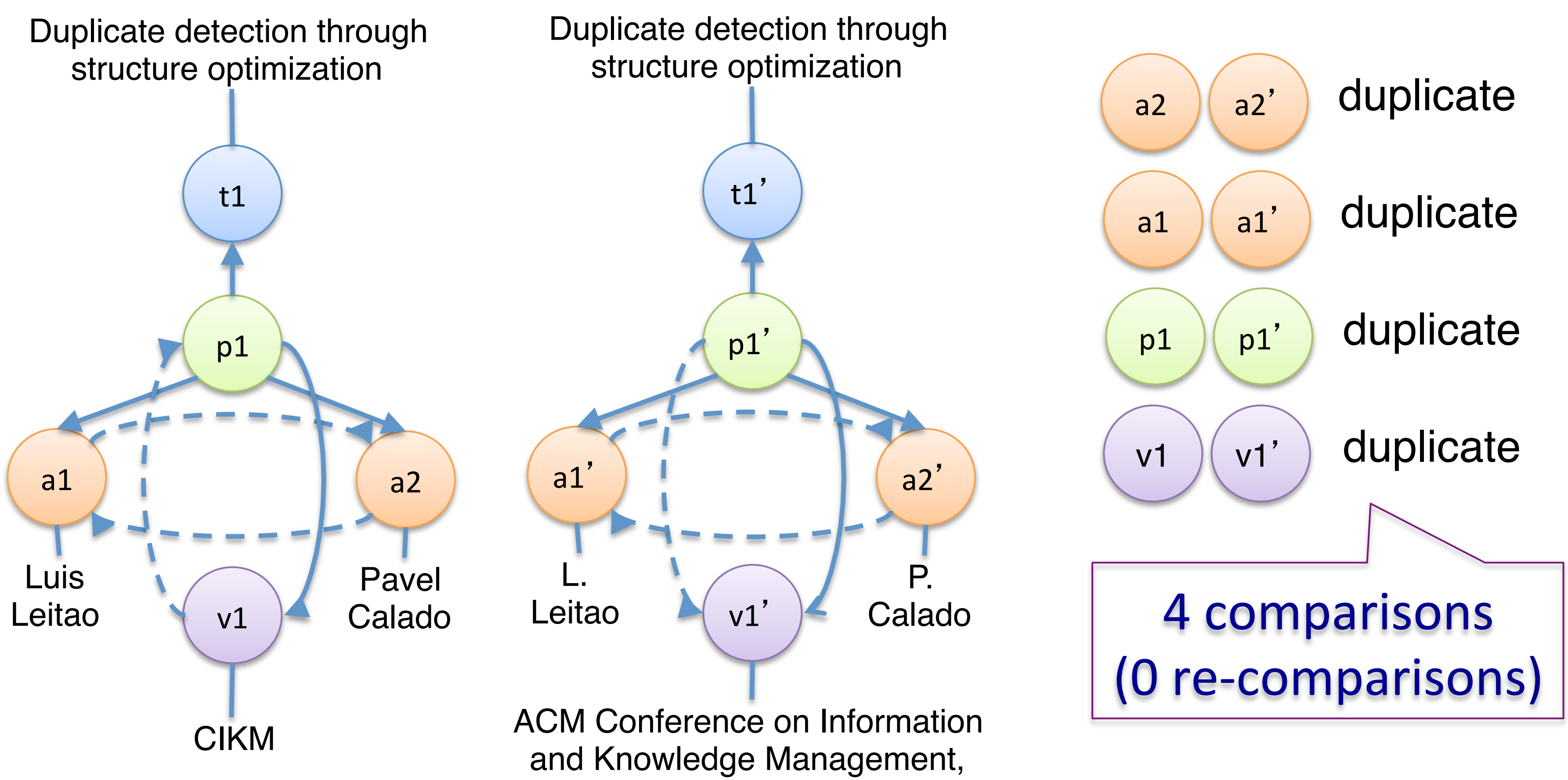
Entity Pair Queue

Top-down [WN04]	Bottom-up [PWN06]	
		
Hierarchy represents 1:N relationships between candidate entities.	Hierarchy represents candidate entities that can be in M:N relationship.	Edges represent all kinds of relationships.
Prune comparisons of descendants that do not have same or similar ancestors.	Sorted neighborhood method applied to every candidate type.	Pairs can be compared more than once →reduce re-comparisons by maintaining an priority queue

Entity Pair Queue



Entity Pair Queue



Entity Pair Queue

- Queue maintenance necessary whenever a duplicate is found
  - Manage order in which pairs are compared to reduce re-comparisons
  - Merge duplicates:
    - Let *m = merge (e1,e2)*
    - Replace all occurrences of e1 and e2 in pair queue by *m*
    - *A*dd 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
- Different strategies of order maintenance:
  - Based on heuristics (degree of nodes in graph) [Weis & Naumann 2006]
  - Based on calculation of (approximate) similarities [Bhattacharya & Getoor 2007]
  - Based on different edge types (FIFO, LIFO) [Dong et al. 2005]
  - Lazy maintenance [Herschel et al. 2012]

Summary of Iterative Entity Resolution

- Various approaches for tabular, tree, and graph data
- Exploiting relationships between duplicate classifications helps in finding more duplicates
  - Transitivity
  - Duplicate dependency
  - Merge dependency
- All approaches assume knowledge of the schema
  - Assuming all compared entities adhere to the same schema
  - Assuming we know the attribute correspondences (mapping) among the different schemas entities conform to.
- Next, we will see methods that lift that assumption

Summary of Iterative Entity Resolution

- We have seen first solutions that also address efficiency
  - Sorted Neighborhood Method (reduction of pairwise comparisons)
  - Recall-preserving filter functions (less complex pairwise comparisons)
  - Pair queue maintenance (less re-comparisons)
- Next: Efficient solutions for entity resolution in graphs

Blocking Approaches

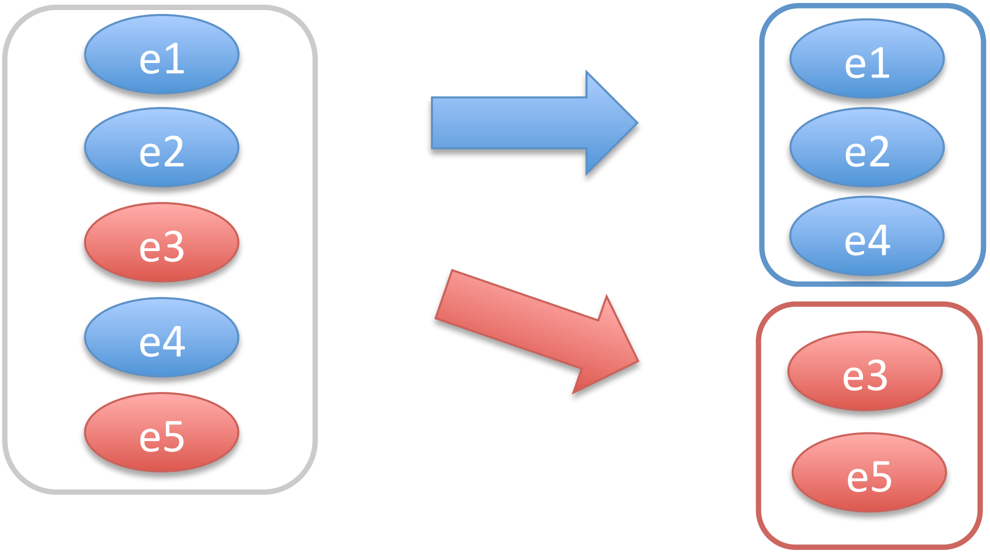
Blocking

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

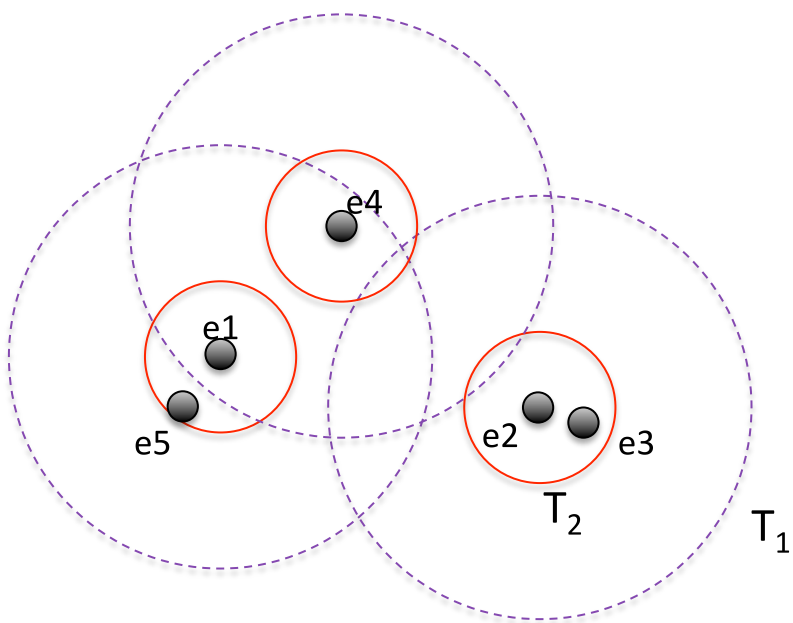
	Name	Year	Architects	Location
e_1	Eiffel Tower	1889	Sauvestre	Paris
e_2	Statue of Liberty	1886	Bartholdi, Eiffel	NY
e_3	Lady Liberty		Eiffel	NY
e_4	Eiffel Tower	1889	Sauvestre	Paris
e_5	White Tower	1450		Thessaloniki

Generated blocks (partition):

Paris	NY	Thessaloniki
e_1, e_4	e_2, e_3	e_5

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_{ei}
   Add to C_{ei} 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 entities 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

e1

name	Statue of Liberty
architect	Bartholdi Eiffel
year	1886
located	NY

e2

about	Lady liberty
architect	Eiffel
location	NY

e3

E_2

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	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	~~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~~

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	Liberty	1889
e_1, e_2, e_3, e_4	e_1, e_4, e_5	e_2, e_3	e_1, e_4

NY	Paris	Sauvestre
e_2, e_3	e_1, e_4	e_1, e_4

The pair (e_1, e_4) 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	Liberty	1889
e_1, e_2, e_3, e_4	e_1, e_4, e_5	e_2, e_3	e_1, e_4

NY	Paris	Sauvestre
e_2, e_3	e_1, e_4	e_1, e_4

Redundant comparisons are performed between (e_1, e_3), (e_2, e_4), (e_1, e_5)

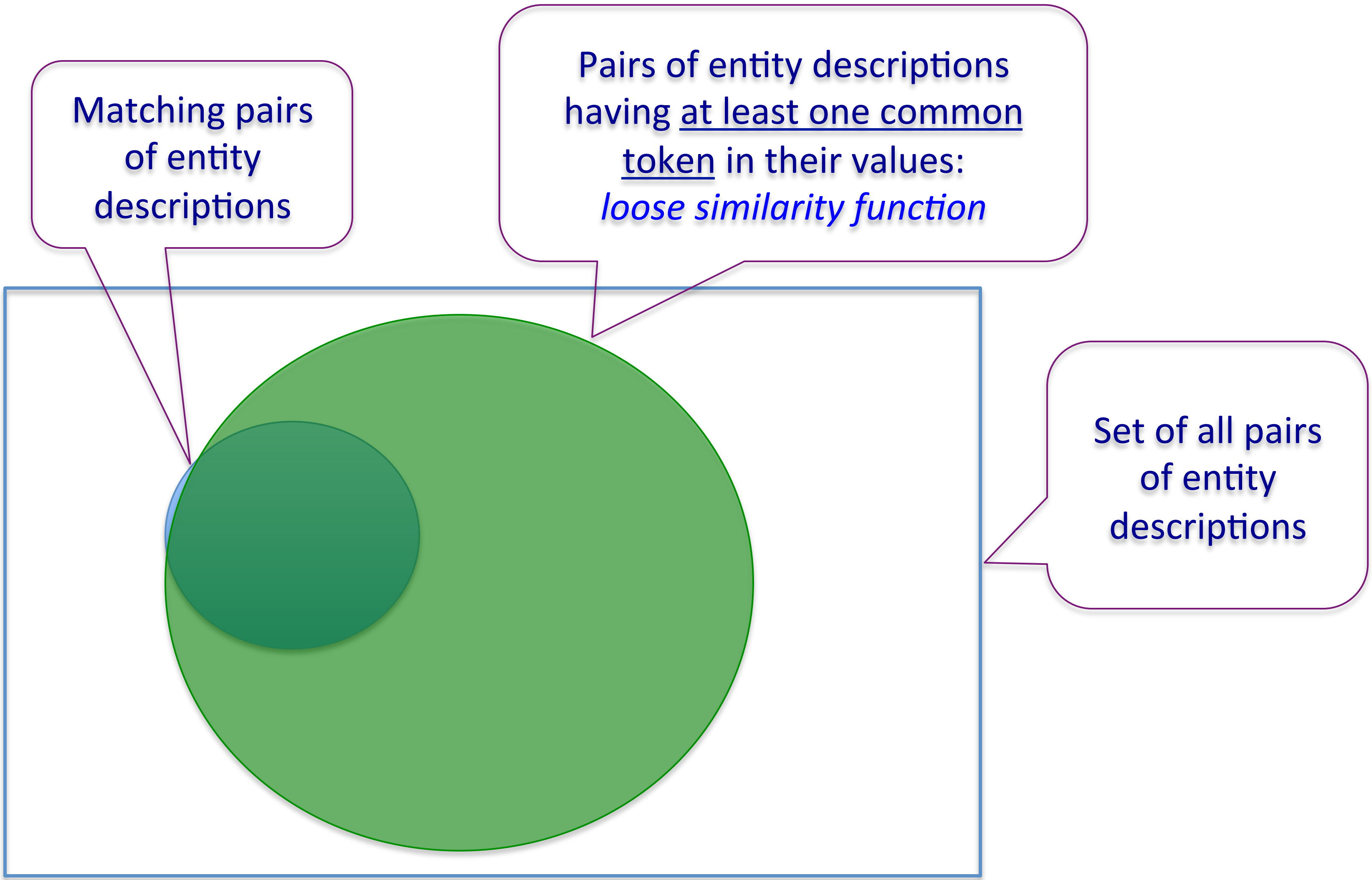
Token blocking achieves:

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

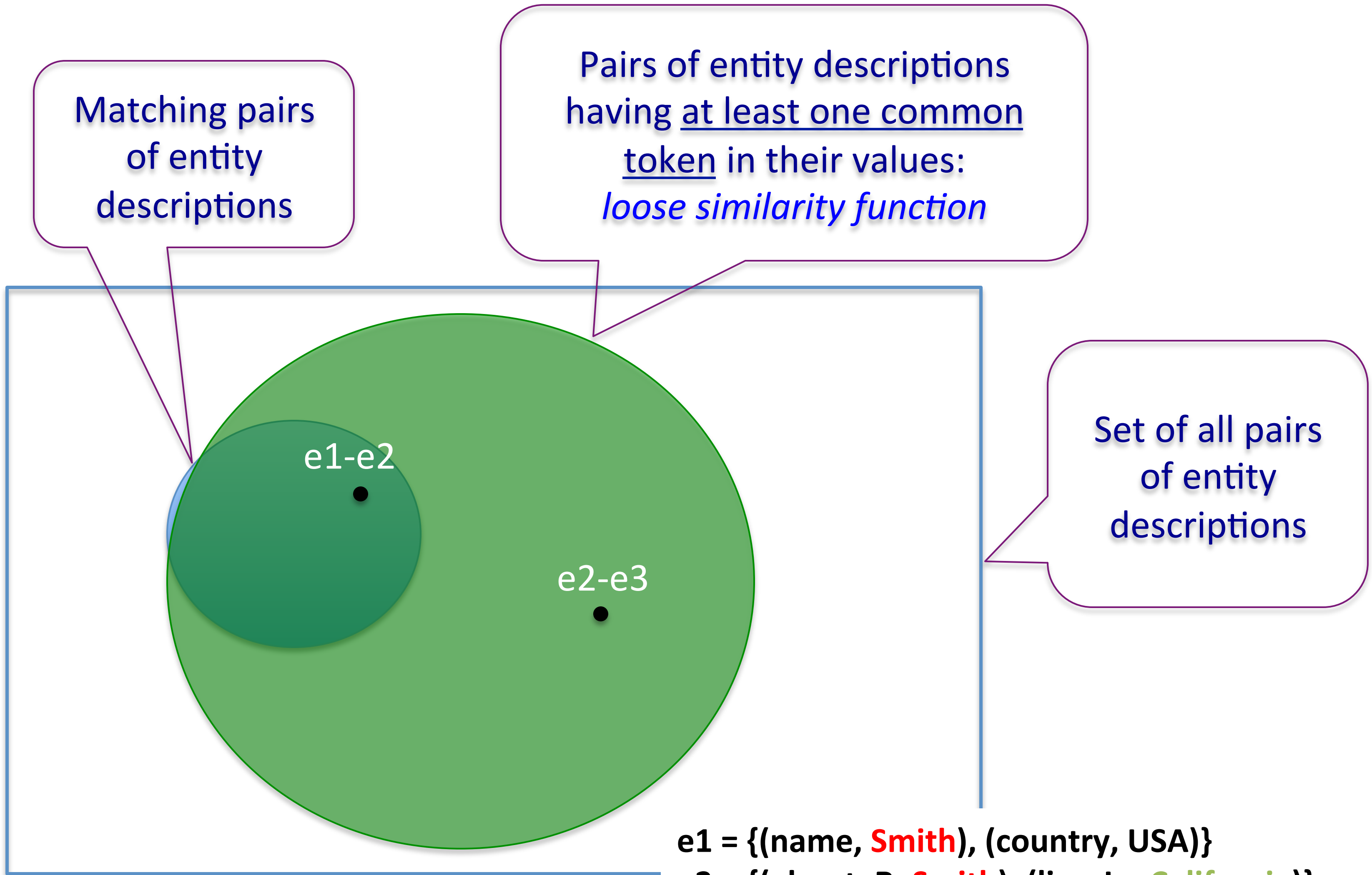
Token blocking totally ignores the valuable information of attribute names

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



e1 = {(name, Smith), (country, USA)}
e2 = {(about, R. Smith), (livesIn, California)}
e3 = {(title, California Dreamin'), (length, 2:34)}

Is this enough?

Token blocking totally ignores the valuable information of attribute names

To improves 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

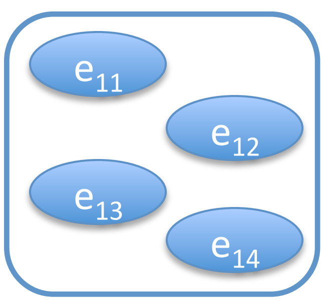
work	Eiffel Tower
year-constructed	1889
location	Paris

e16

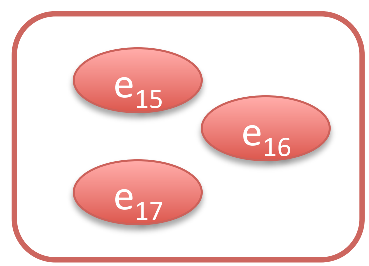
work	Bartholdi Fountain
year-constructed	1876
location	Washington D.C.

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 <u>**work**</u>: {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

work	Lady Liberty
artist	Bartholdi
location	NY

e15

about	Statue of Liberty
architect	Bartholdi Eiffel
year	1886
located	NY

e12

about	Auguste Bartholdi
born	1834

e13

work	Eiffel Tower
year-constructed	1889
location	Paris

e16

about	Joan Tower
born	1938

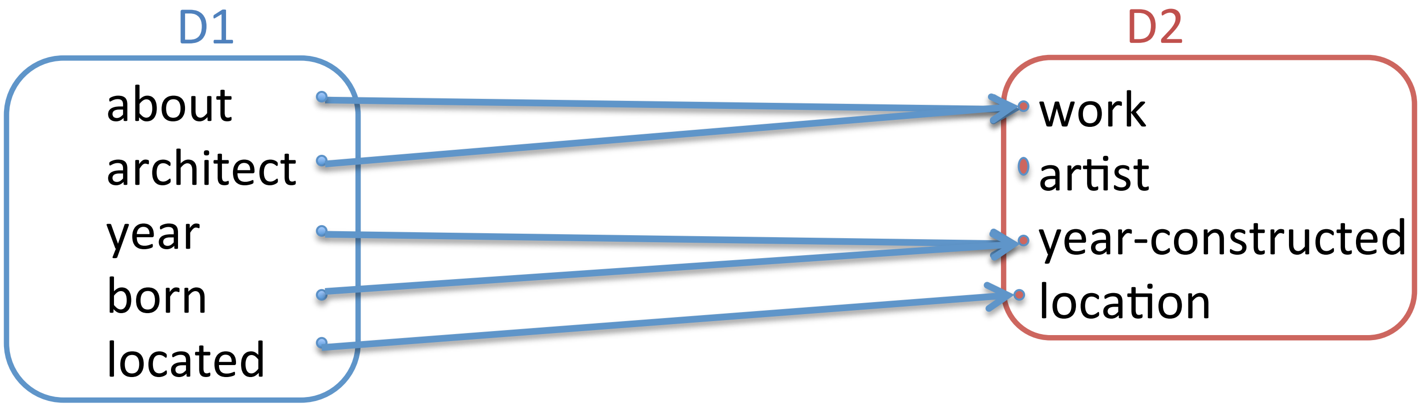
e14

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

e17

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

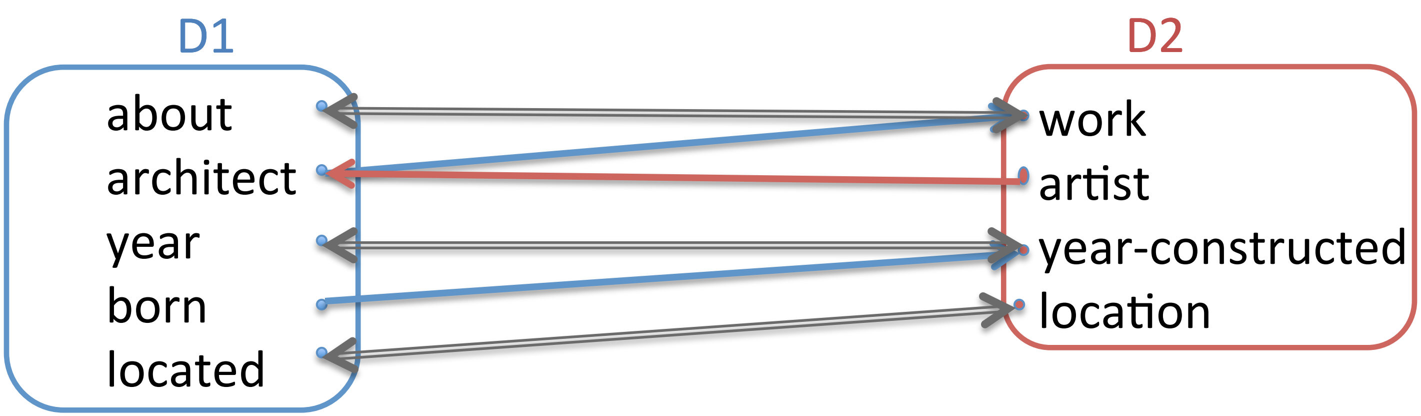
e15

work	Eiffel Tower
year-constructed	1889
location	Paris

e16

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

e17



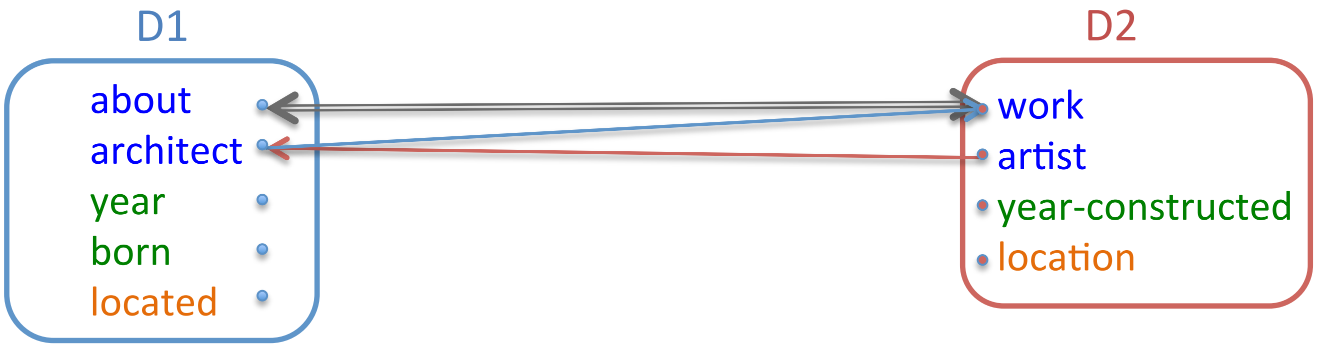
Clustering Attributes: Example

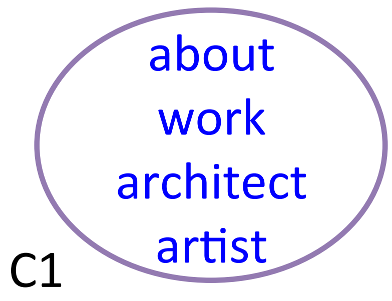
Compute the transitive closure of the generated attribute pairs

- Connected attributes form clusters

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

Transitive closure:





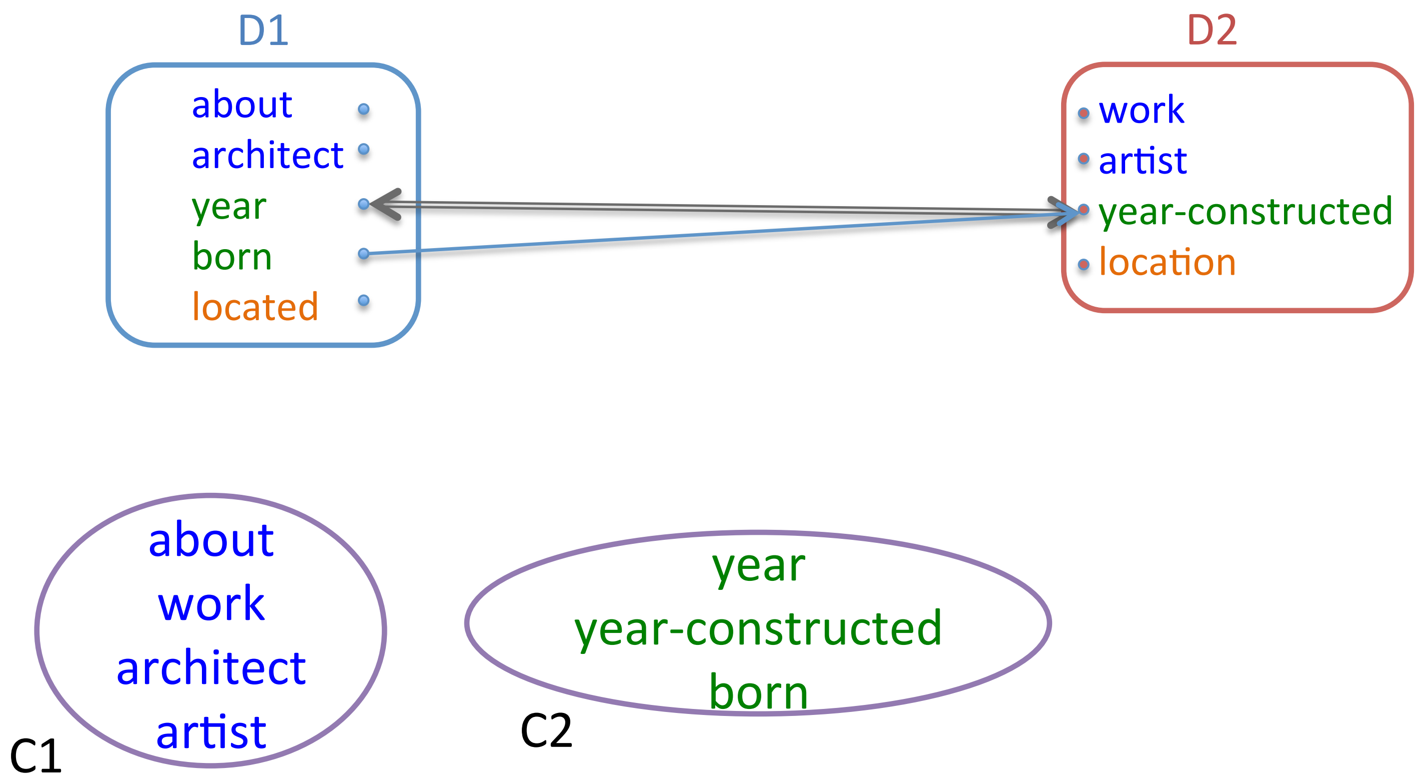
Clustering Attributes: Example

Compute the <u>transitive closure</u> of the generated attribute pairs

- Connected attributes form clusters

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

Transitive closure:



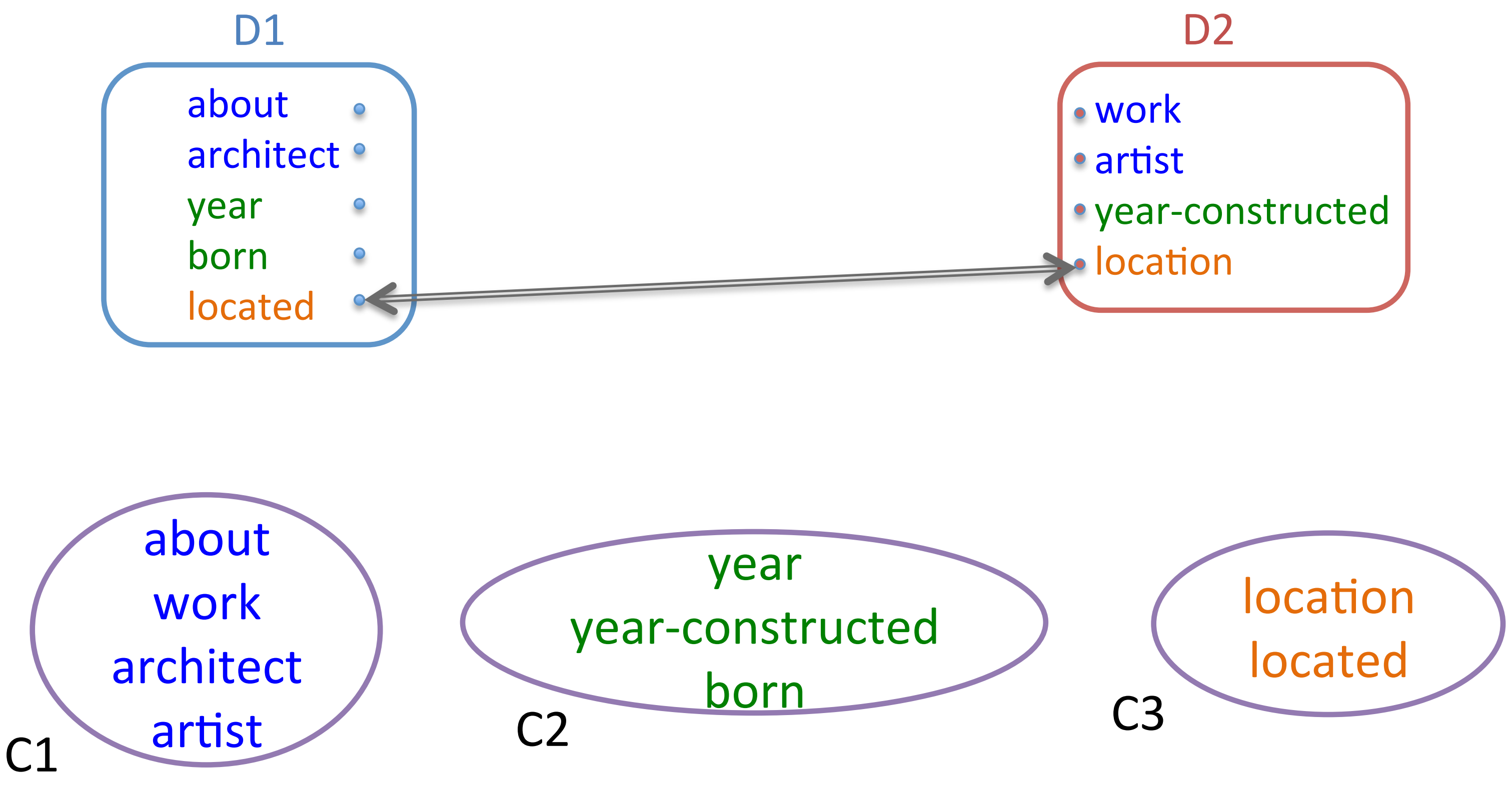
Clustering Attributes: Example

Compute the transitive closure of the generated attribute pairs

- Connected attributes form clusters

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

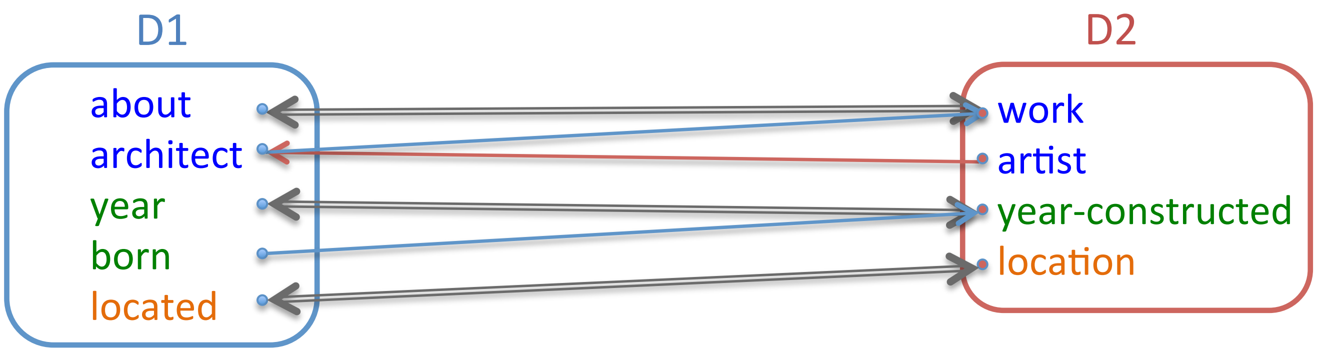
Transitive closure:



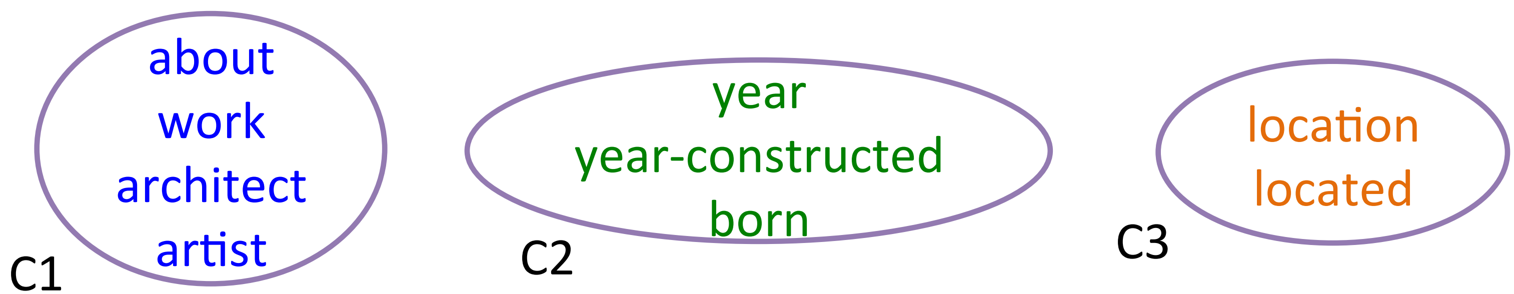
Clustering Attributes: Example

Compute the transitive closure of the generated attribute pairs

- Connected attributes form clusters



Generated attribute clusters:



Token Blocking for Each Cluster

e11

about	Eiffel Tower
architect	Sauvestre
year	1889
located	Paris

e12

about	Statue of Liberty
architect	Bartholdi Eiffel
year	1886
located	NY

e13

about	Auguste Bartholdi
born	1834

e14

about	Joan Tower
born	1938

e15

work	Lady Liberty
artist	Bartholdi
location	NY

e16

work	Eiffel Tower
year-constructed	1889
location	Paris

e17

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

C1: about, work, architect, artist

C2: year, year-constructed, born

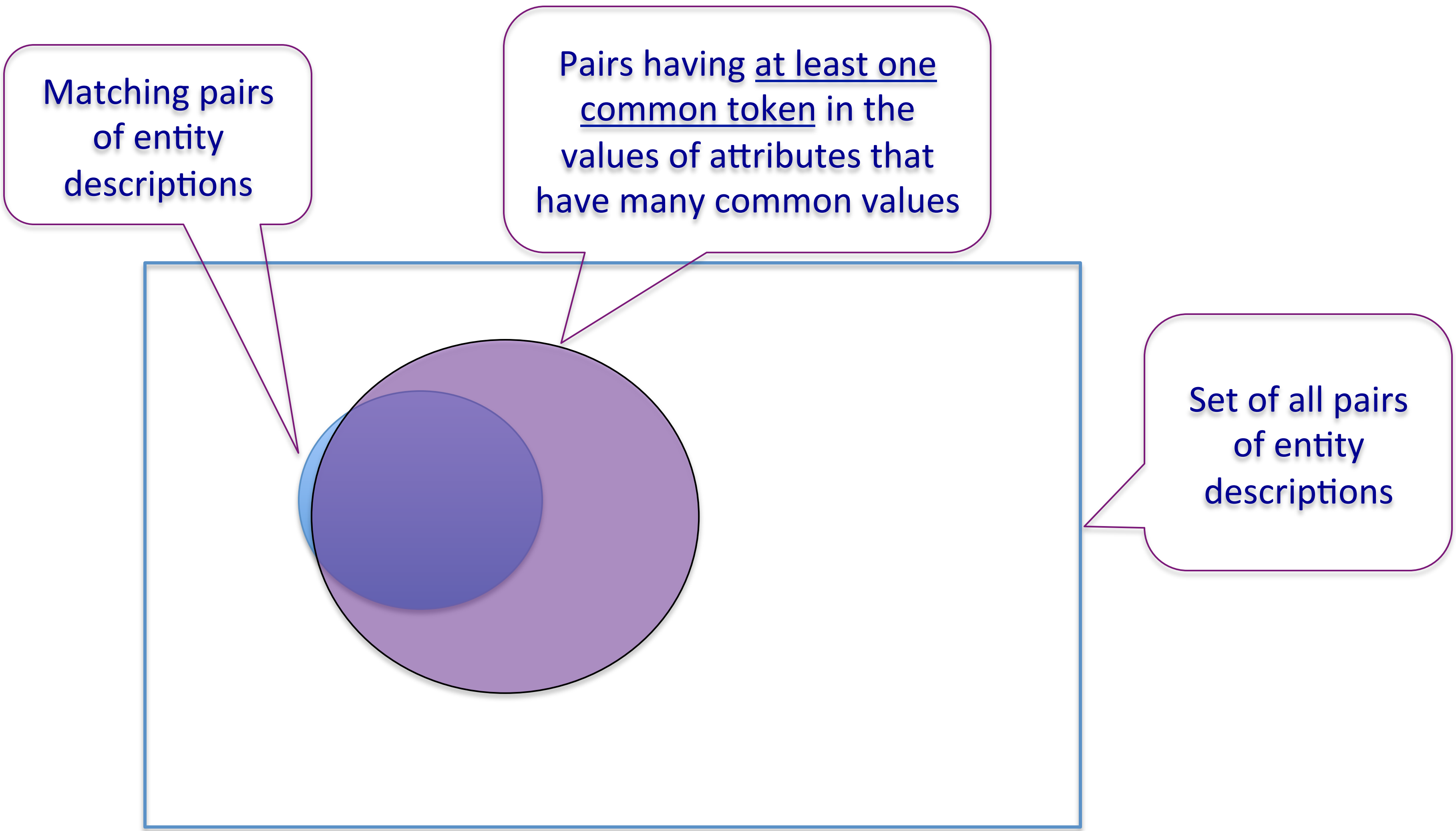
C3: location, located

Some of the generated blocks:

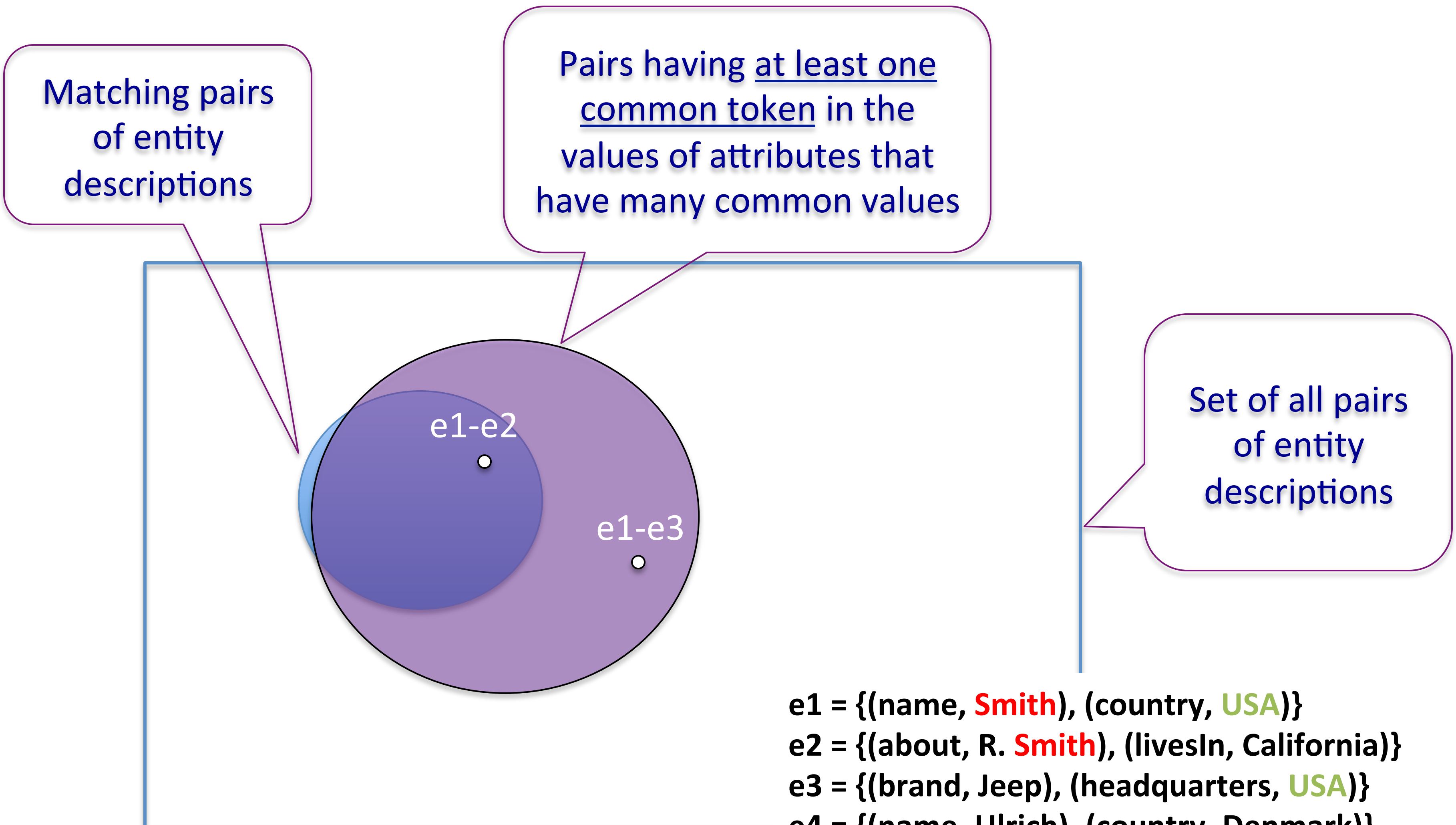
C3.NY	C1.Tower	C1.Bartholdi
e_{12}, e_{15}	e_{11}, e_{14}, e_{16}	e_{12}, e_{13}, e_{15}, e_{17}

→ *compare the Lady Liberty to Auguste Bartholdi*

Attribute Clustering Blocking- Evaluation



Attribute Clustering Blocking- Evaluation



e1 = {(name, Smith), (country, USA)}
e2 = {(about, R. Smith), (livesIn, California)}
e3 = {(brand, Jeep), (headquarters, USA)}
e4 = {(name, Ulrich), (country, Denmark)}
e5 = {(about, D. Brunson), (livesIn, Nevada)}

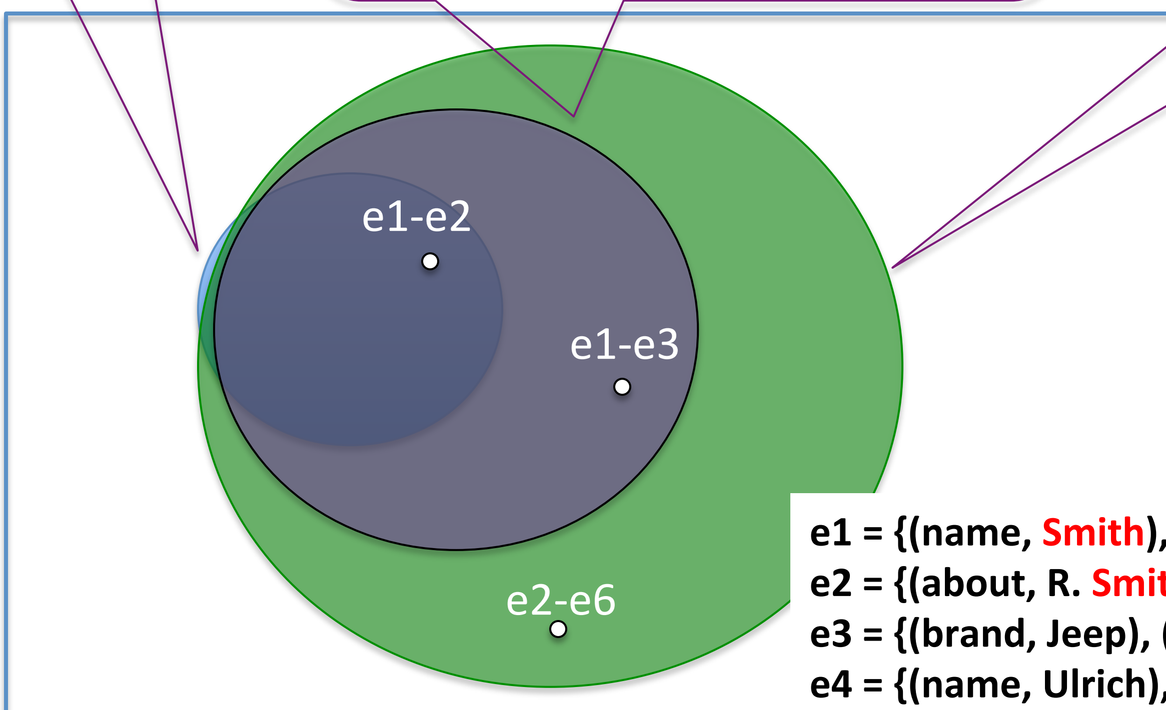
Attribute Clustering Blocking vs Token Blocking

Matching pairs of entity descriptions

Pairs having <u>at least one common token</u> in the values of attributes that have many common values: *a not so loose similarity function*

Pairs having <u>at least one common token</u> in their values: *loose similarity function*

Set of all pairs of entity descriptions



e1 = {(name, Smith), (country, USA)}
e2 = {(about, R. Smith), (livesIn, California)}
e3 = {(brand, Jeep), (headquarters, USA)}
e4 = {(name, Ulrich), (country, Denmark)}
e5 = {(about, D. Brunson), (livesIn, Nevada)}
e6 = {(title, California Dreamin'), (length, 2:34)}

Now, is this enough?

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

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

http://km.aifb.kit.edu/pro jects/btc-2009/

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

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

dbpedia:Statue_of_Liberty

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

fb:m.072p8

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

geonames:5139572

geonames:name	Statue of Liberty	
geonames: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

e1

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

e2

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

e3

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

e4

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

e5

skos:prefLabel	Tina Brown
yago:linksTo	yago:Liberty_Island

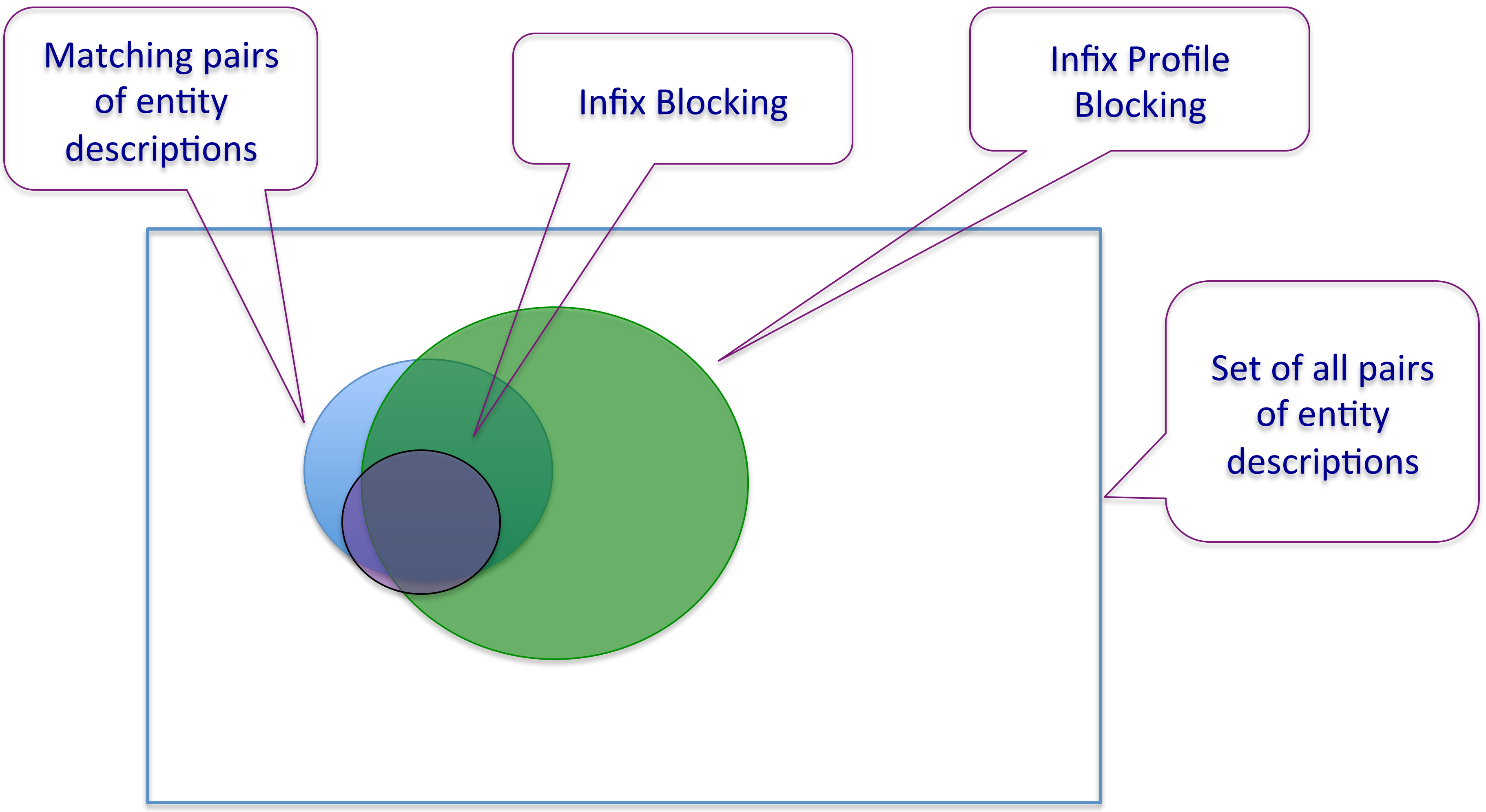
pros: (e1, e3) correctly identified
cons: (e1, e5) mistakenly identified

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



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?

Tutorial Overview

Coffee break!

What follows in Part II:

- Blocking & post-blocking approaches
- Large scale entity resolution using MapReduce
- Conclusions