International World Wide Web Conference
Seoul, Korea
April 7 - 11, 2014
WWW 2014

# Entity Resolution in the Web of Data

# Part II

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, 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 |
| 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 (= 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:

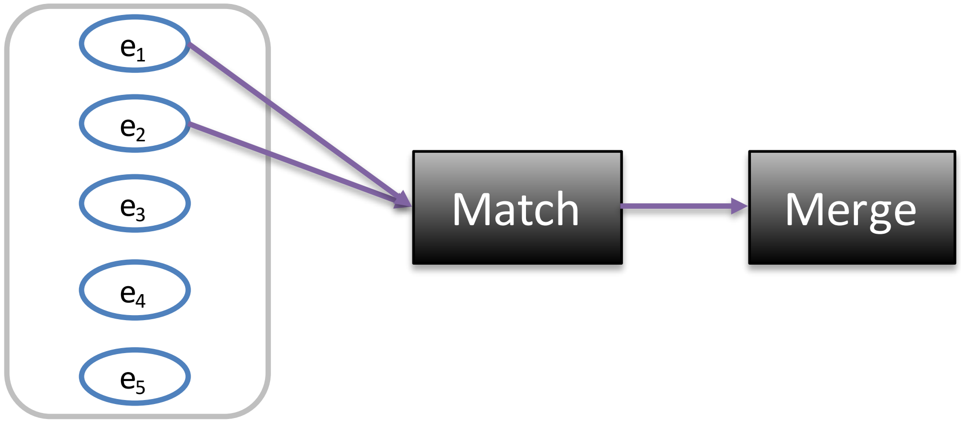
- A match function M
- A merge function μ

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:

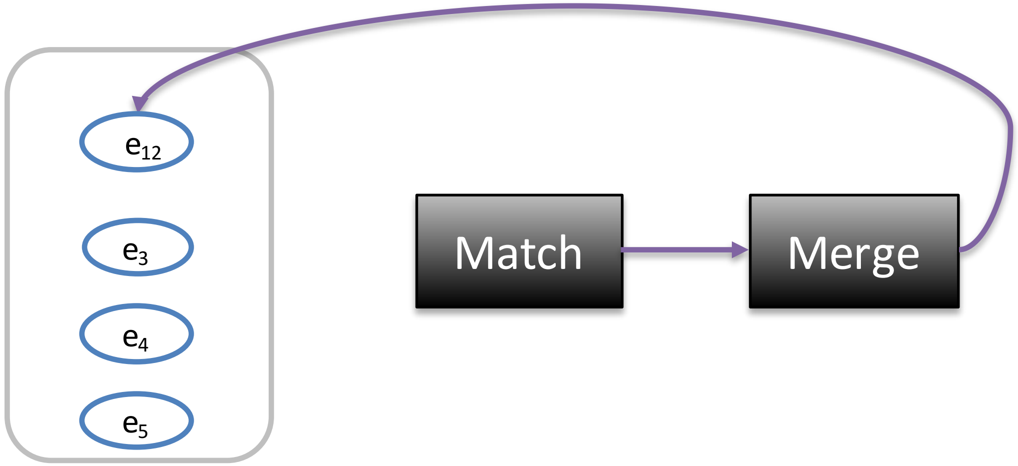
- A <u>match</u> function M
- A <u>merge</u> function μ

<u>The goal</u>:

*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:

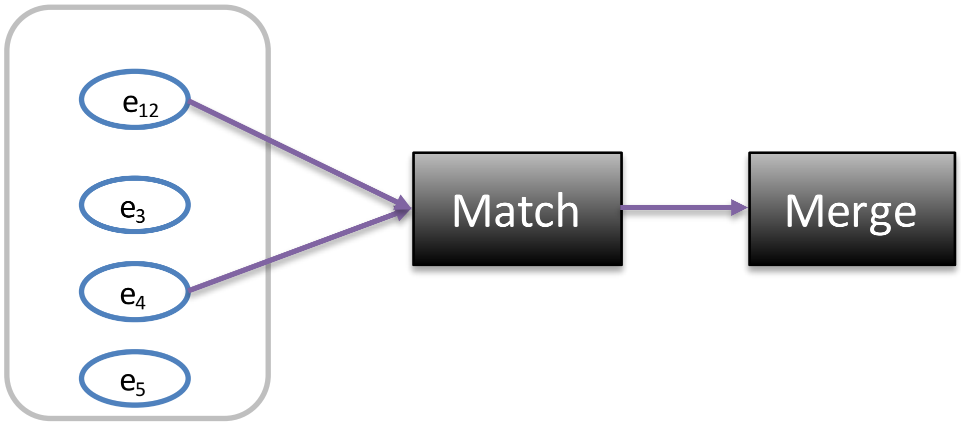
$M(e_1, e_2) = M(e_2, e_1)$ and $\mu(e_1, e_2) = \mu(e_2, e_1)$

- Associativity:

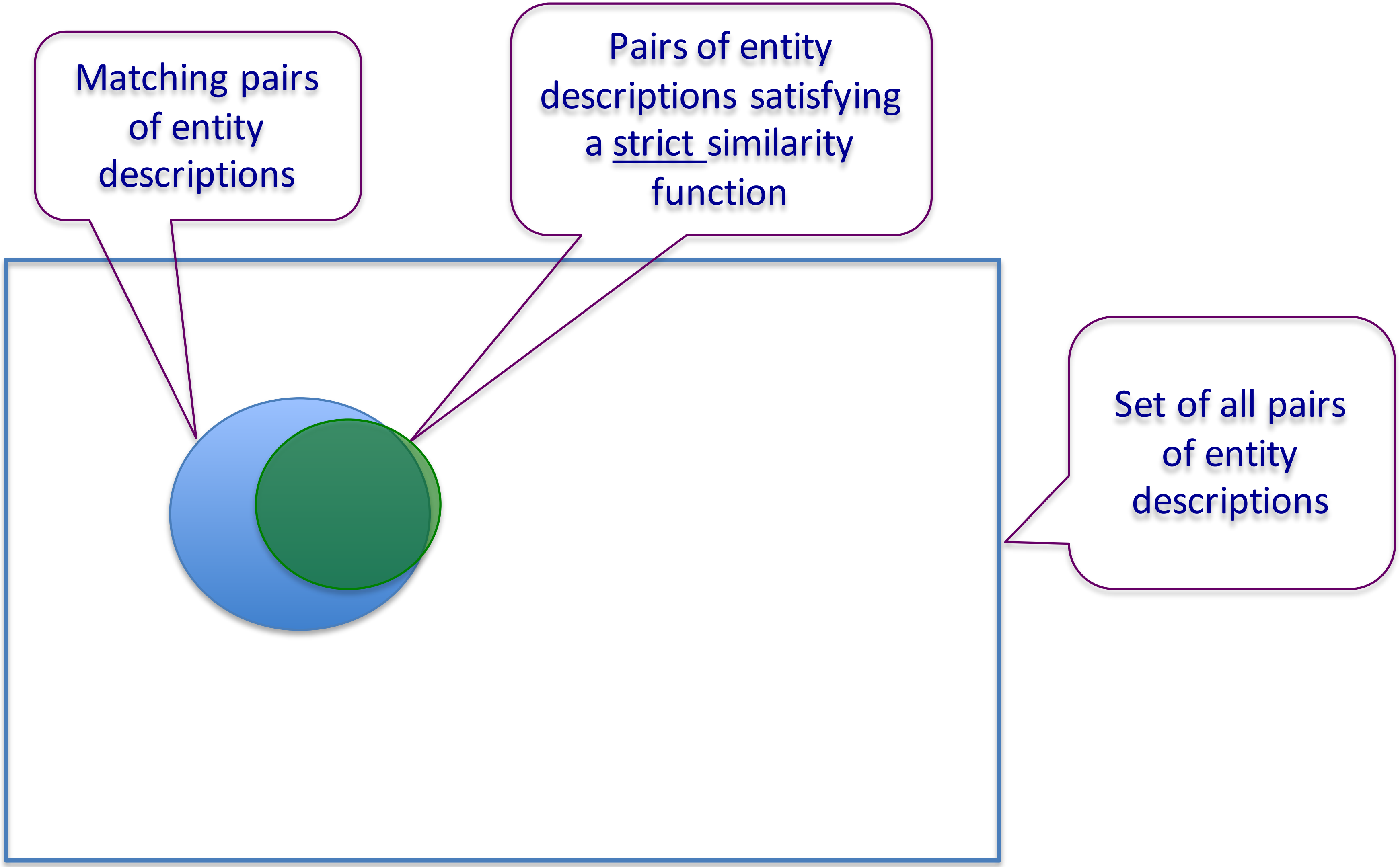
$\mu(e_1, \mu(e_2, e_3)) = \mu(\mu(e_1, e_2), e_3)$

- Representativity:

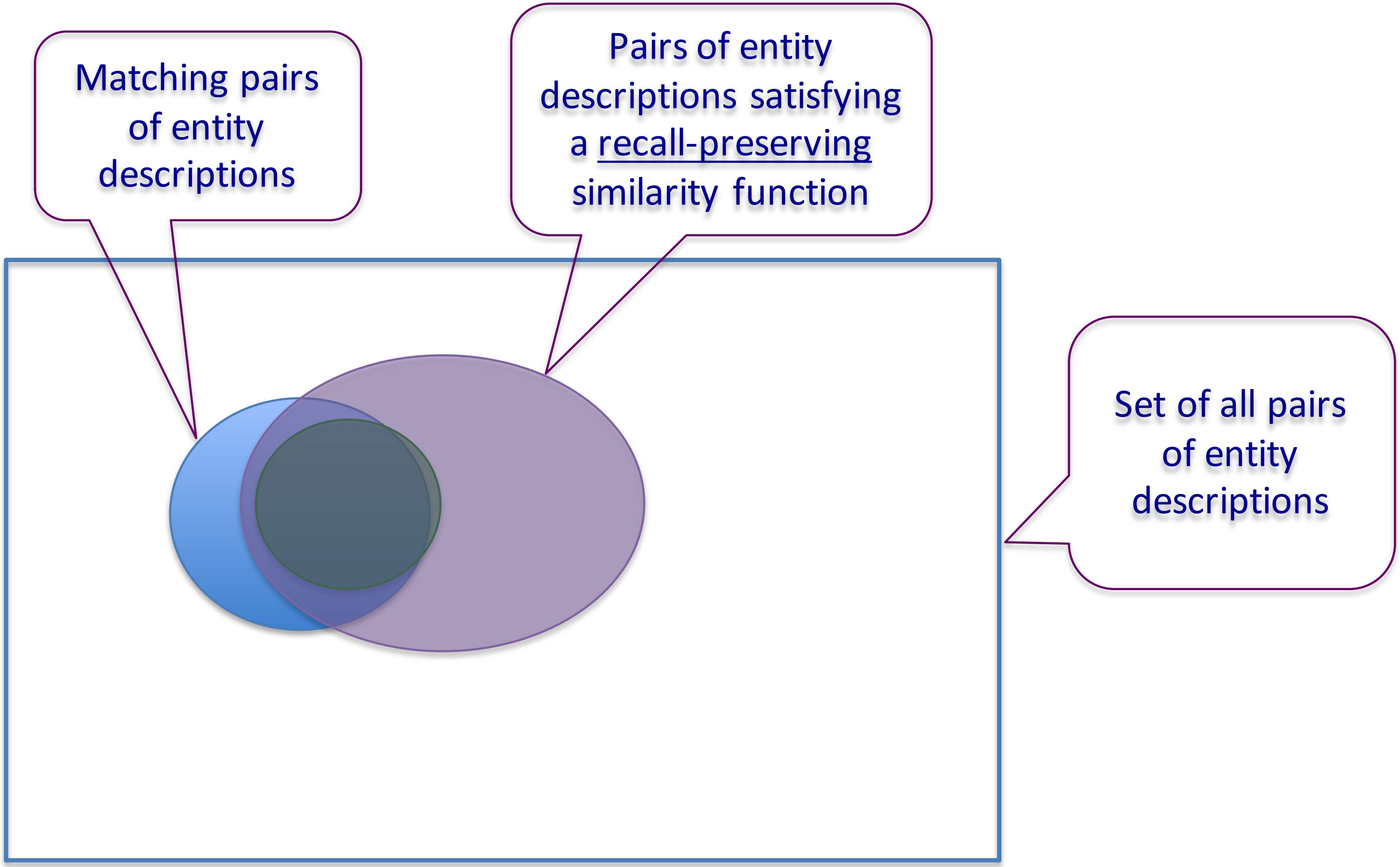
if $\mu(e_1, e_2) = e_3$ and $M(e_1, e_4) = true$, then $M(e_3, e_4) = true$



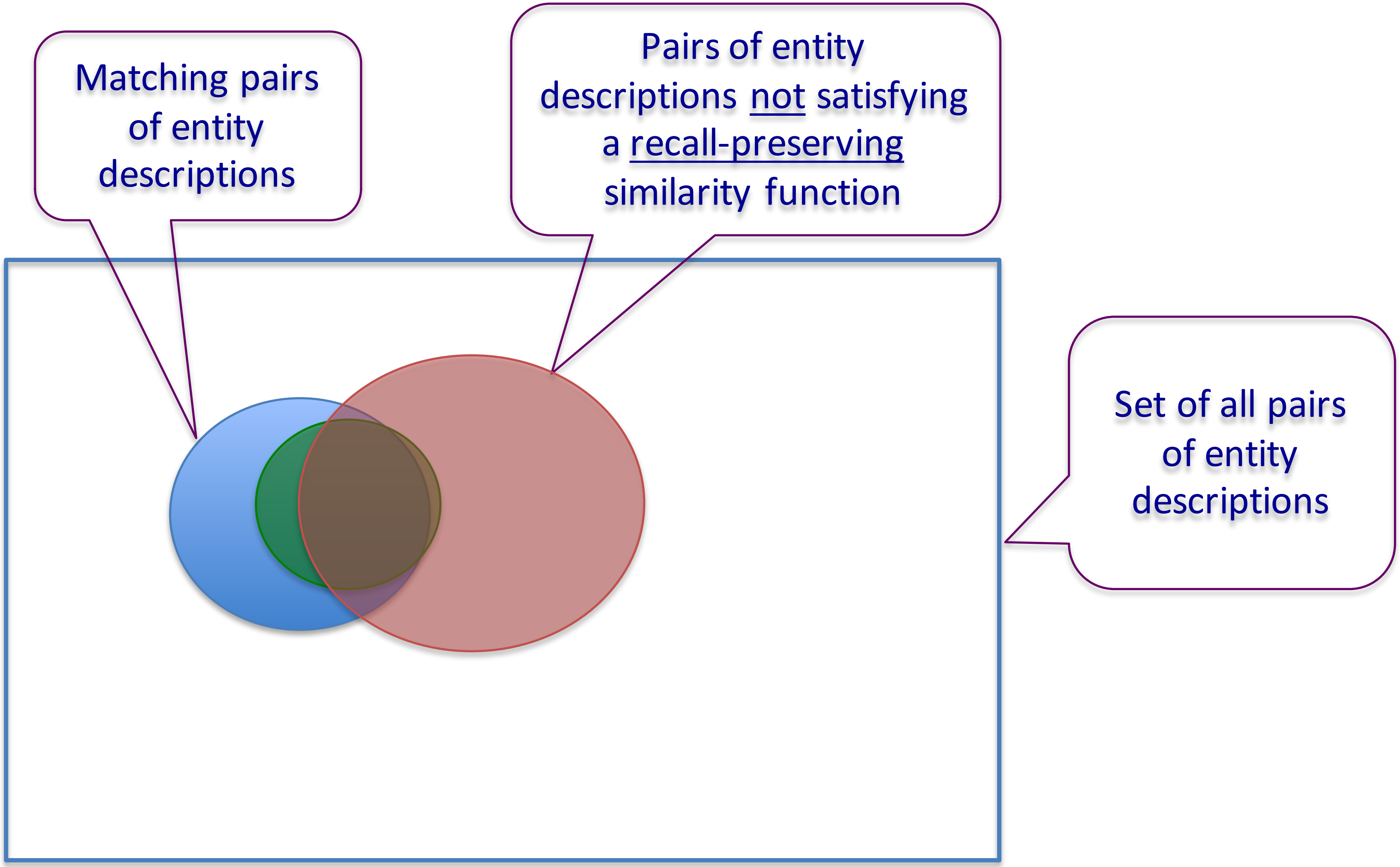
# Recall-Maintaining Similarity Functions



# Recall-Maintaining Similarity Functions



# Recall-Maintaining Similarity Functions



# Iterative Entity Resolution – Tree Data

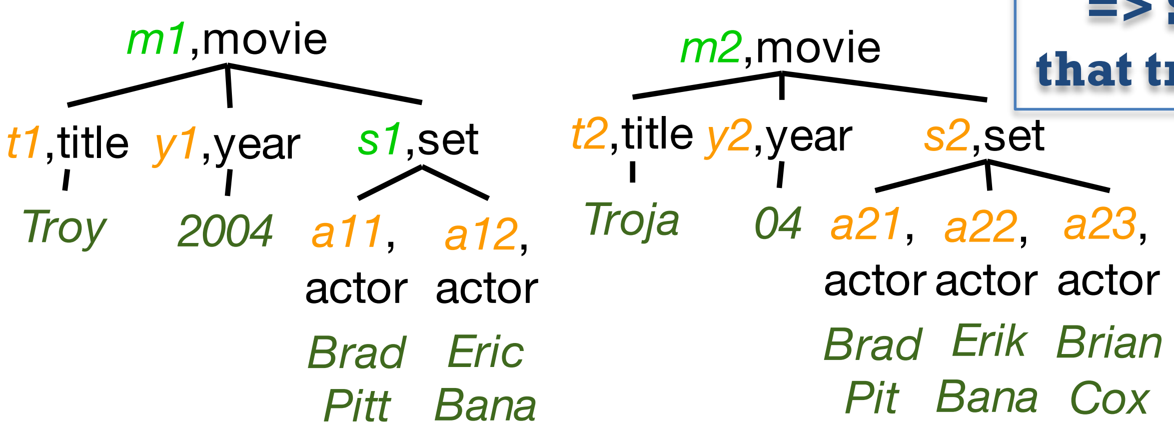
- Examples of hierarchically organized data
  - Relational star / snowflake schema [Ananthakrishna et al. 2002]

| 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 → 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:
    *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 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)* = {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

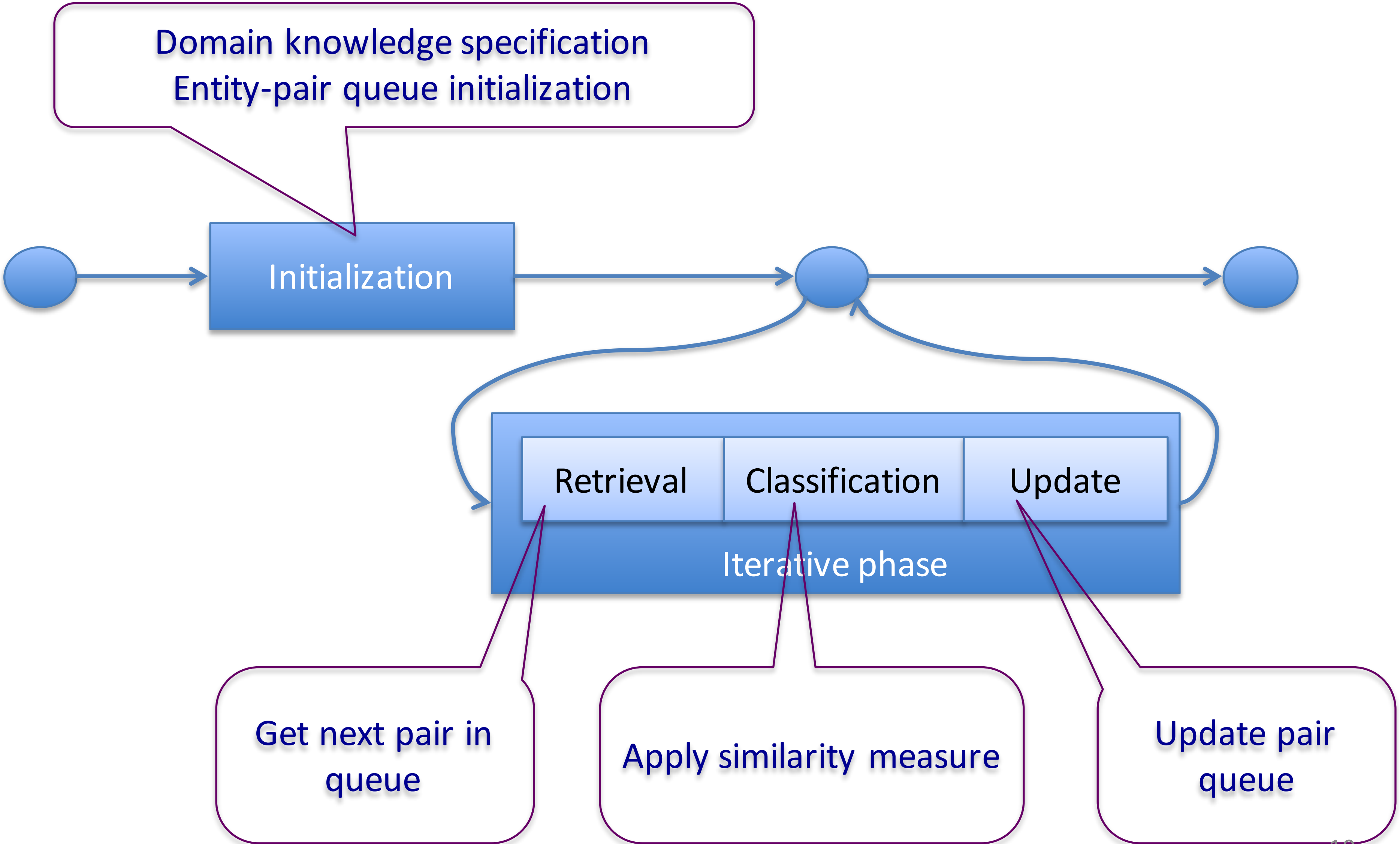
**5. Children co-occurrence**
*fkcm(F1,F2)* = 1, *fkcm(F2,F1)* = ½

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 match

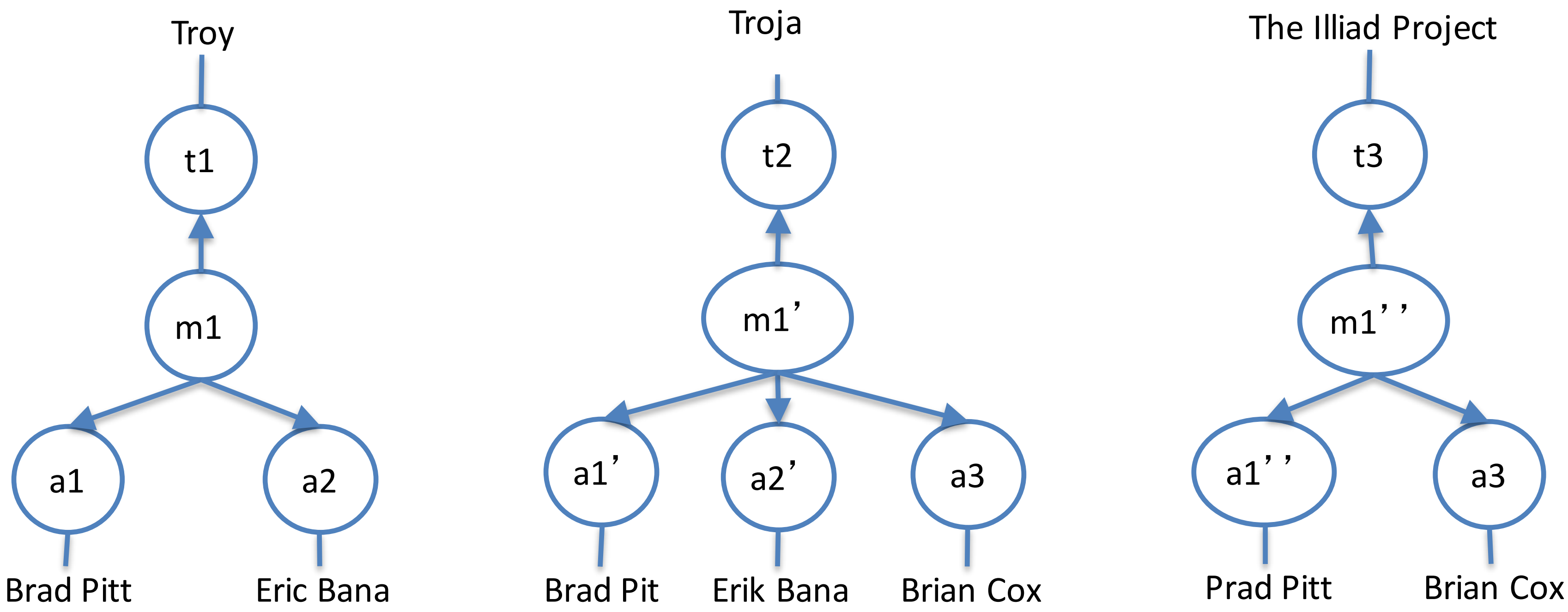
# Iterative Entity Resolution - Graph Data

- 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

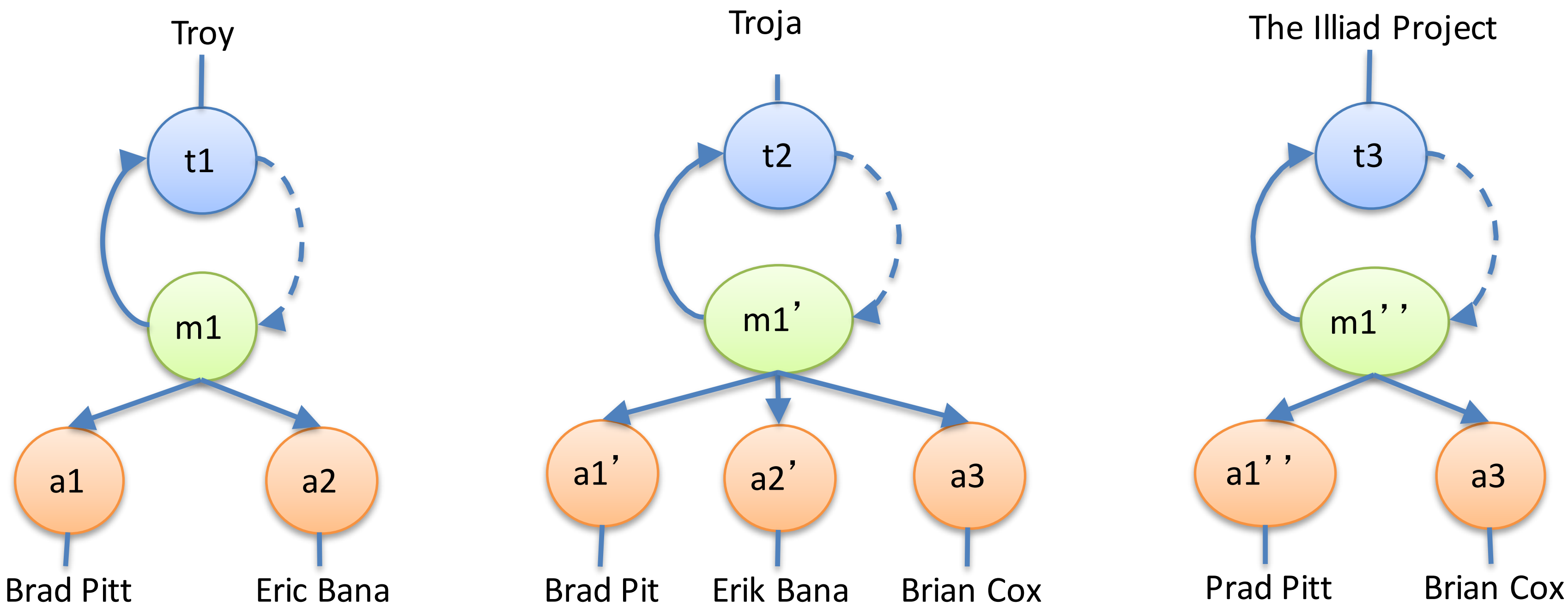


# Domain Expert Knowledge Specification



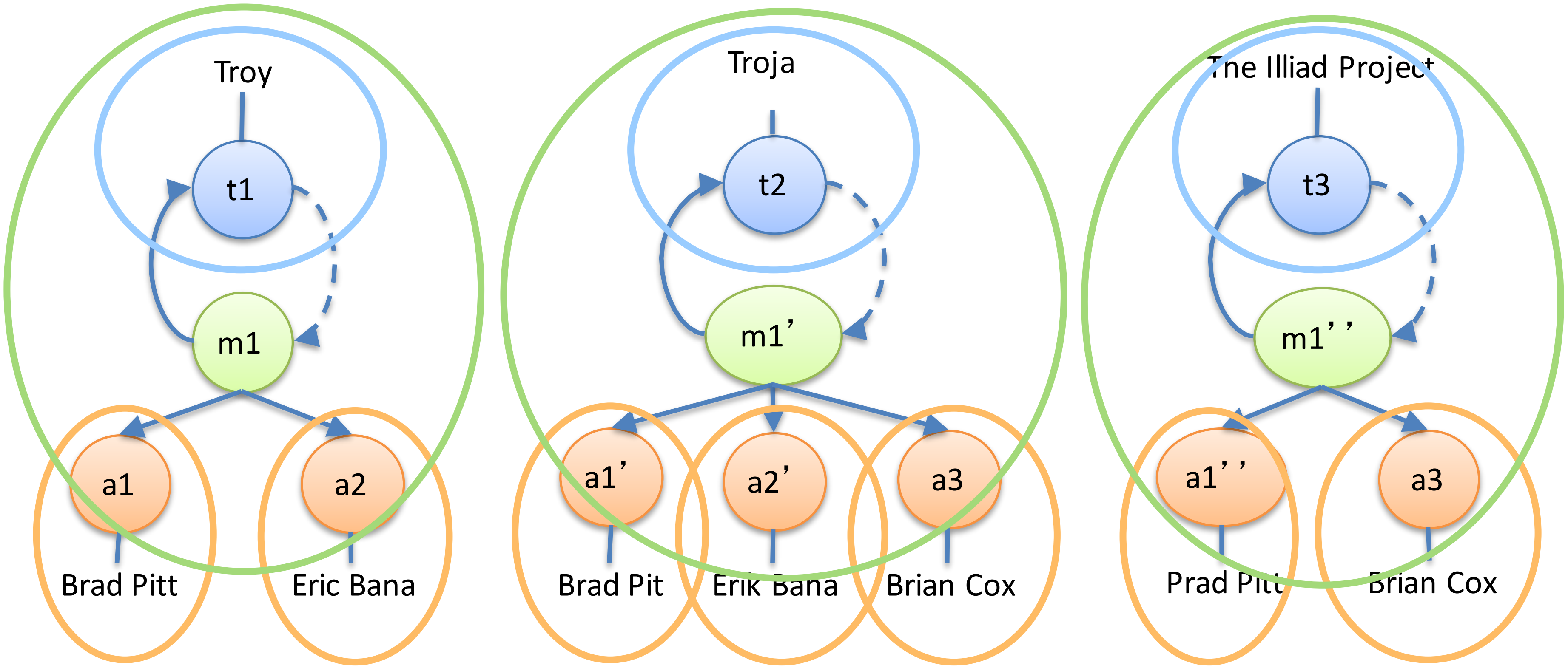

- Domain expert specifies
  - Duplicate candidate entities (e.g., movie, actor, title)
  - (Additional) relationships between candidates (e.g., title → movie)

# Domain Expert Knowledge Specification



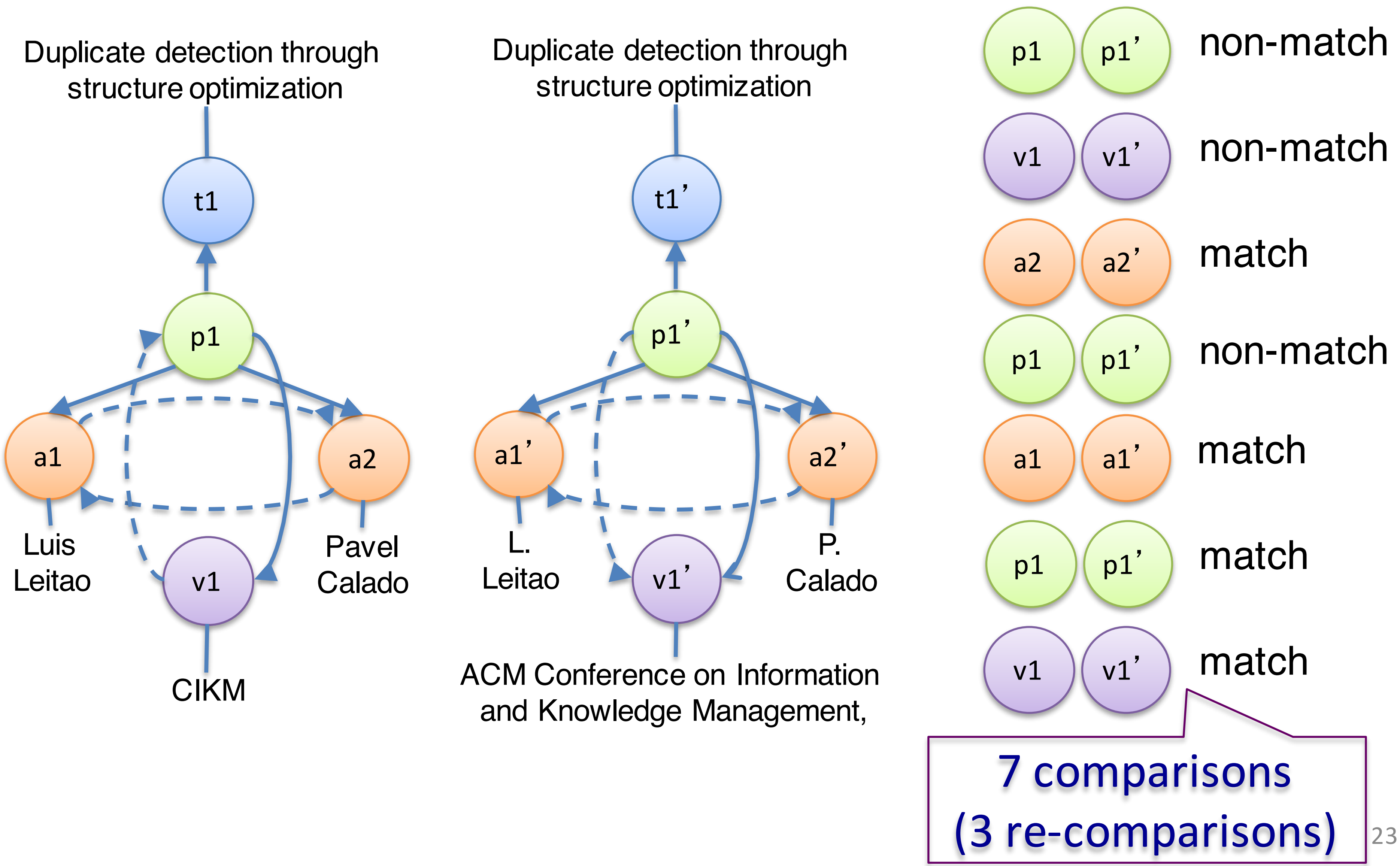

- Domain expert specifies
  - Duplicate candidate entities (e.g., movie, actor, title)
  - (Additional) relationships between candidates (e.g., title → movie)

# Domain Expert Knowledge Specification

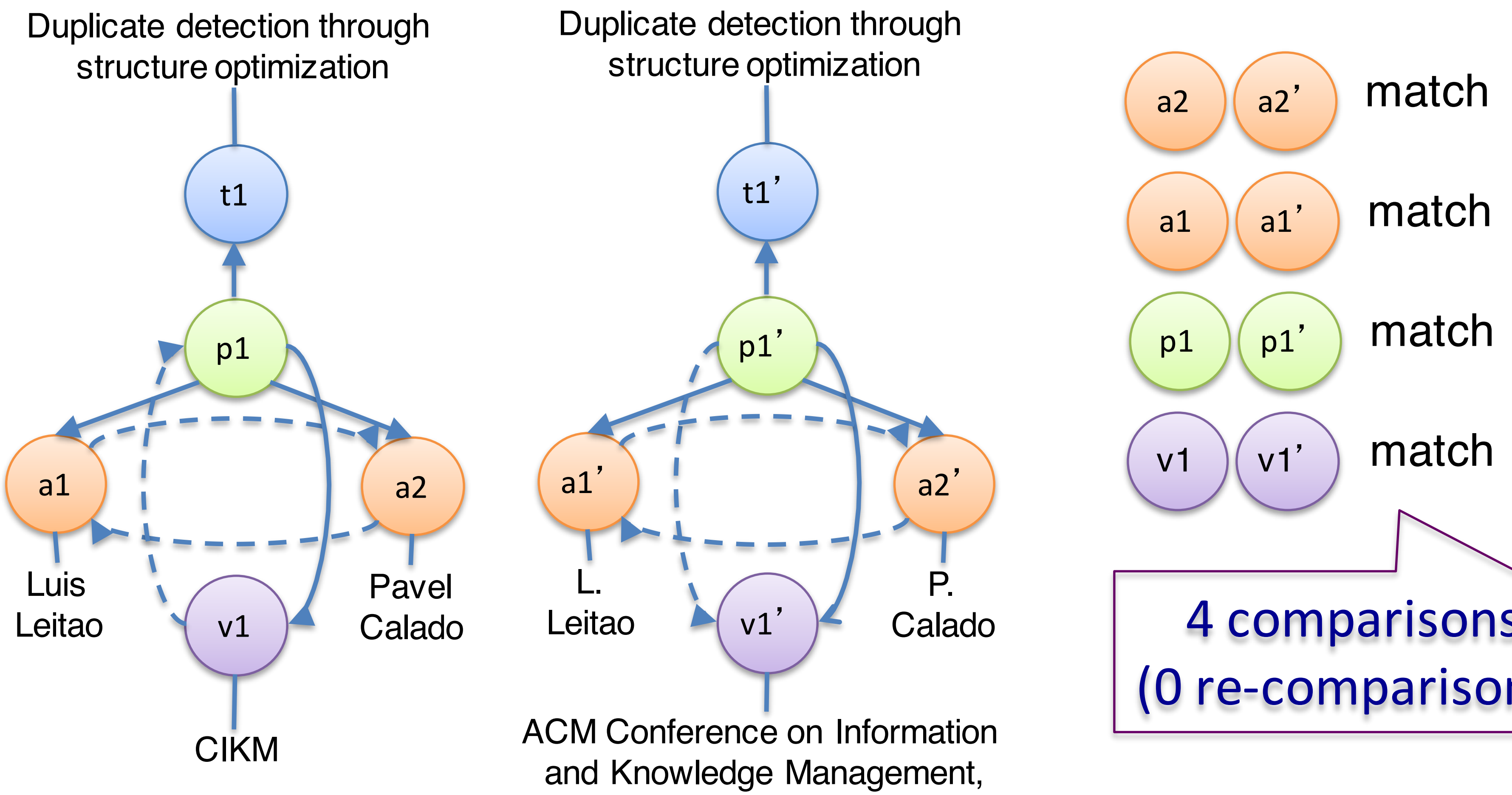



- For pairwise similarity computation, domain expert also selects what information is relevant for comparisons
  - Entity description (attribute values)
  - Influencing neighbor candidates

# Entity Pair Queue



# Entity Pair Queue




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 $m = merge\ (e1,e2)$
    - 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

# 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 | Year | Architects | Location |
|---|---|---|---|---|
| $e_1$ | Eiffel Tower | 1889 | Sauvestre | Paris |
| $e_2$ | Statue of Liberty | 1886 | Bartholdi, Eiffel | NY |
| $e_3$ | Lady Liberty | 1885 | Eiffel | Liberty Island, NY |
| $e_4$ | Eiffel Tower | 1889 | | Paris |
| $e_5$ | Miss Liberty | 1886 | Gustave Eiffel | Liberty 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 | Paris |
| $e_2$ | Statue of Liberty | 1886 | Bartholdi, Eiffel | NY |
| $e_3$ | Lady Liberty | 1885 | Eiffel | Liberty Island, NY |
| $e_4$ | Eiffel Tower | 1889 | | Paris |
| $e_5$ | Miss Liberty | 1886 | Gustave Eiffel | Liberty 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 | Year | Architects | Location |
|---|---|---|---|---|
| $e_1$ | Eiffel Tower | 1889 | Sauvestre | Paris |
| $e_2$ | Statue of Liberty | 1886 | Bartholdi, Eiffel | NY |
| $e_3$ | Lady Liberty | 1885 | Eiffel | Liberty Island, NY |
| $e_4$ | Eiffel Tower | 1889 | | Paris |
| $e_5$ | Miss Liberty | 1886 | Gustave Eiffel | Liberty 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 | Year | Architects | Location |
|---|---|---|---|---|
| $e_1$ | Eiffel Tower | 1889 | Sauvestre | Paris |
| $e_2$ | Statue of Liberty | 1886 | Bartholdi, Eiffel | NY |
| $e_3$ | Lady Liberty | 1885 | Eiffel | Liberty Island, NY |
| $e_4$ | Eiffel Tower | 1889 | | Paris |
| $e_5$ | Miss Liberty | 1886 | Gustave Eiffel | Liberty 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 | Year | Architects | Location |
|---|---|---|---|---|
| $e_1$ | Eiffel Tower | 1889 | Sauvestre | Paris |
| $e_2$ | Statue of Liberty | 1886 | Bartholdi, Eiffel | NY |
| $e_3$ | Lady Liberty | 1885 | Eiffel | Liberty Island, NY |
| $e_4$ | Eiffel Tower | 1889 | | Paris |
| $e_5$ | Miss Liberty | 1886 | Gustave Eiffel | Liberty 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 | Year | Architects | Location |
|---|---|---|---|---|
| $e_1$ | Eiffel Tower | 1889 | Sauvestre | Paris |
| $e_2$ | Statue of Liberty | 1886 | Bartholdi, Eiffel | NY |
| $e_3$ | Lady Liberty | 1885 | Eiffel | Liberty Island, NY |
| $e_4$ | Eiffel Tower | 1889 | | Paris |
| $e_5$ | Miss Liberty | 1886 | Gustave Eiffel | Liberty 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 | Year | Architects | Location |
|---|---|---|---|---|
| $e_1$ | Eiffel Tower | 1889 | Sauvestre | Paris |
| $e_2$ | Statue of Liberty | 1886 | Bartholdi, Eiffel | NY |
| $e_3$ | Lady Liberty | 1885 | Eiffel | Liberty Island, NY |
| $e_4$ | Eiffel Tower | 1889 | | Paris |
| $e_5$ | Miss Liberty | 1886 | Gustave Eiffel | Liberty 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 | Year | Architects | Location |
|---|---|---|---|---|
| $e_1$ | Eiffel Tower | 1889 | Sauvestre | Paris |
| $e_2$ | Statue of Liberty | 1886 | Bartholdi, Eiffel | NY |
| $e_3$ | Lady Liberty | 1885 | Eiffel | Liberty Island, NY |
| $e_4$ | Eiffel Tower | 1889 | | Paris |
| $e_5$ | Miss Liberty | 1886 | Gustave Eiffel | Liberty 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 | Year | Architects | Location |
|---|---|---|---|---|
| $e_1$ | Eiffel Tower | 1889 | Sauvestre | Paris |
| $e_2$ | Statue of Liberty | 1886 | Bartholdi, Eiffel | NY |
| $e_3$ | Lady Liberty | 1885 | Eiffel | Liberty Island, NY |
| $e_4$ | Eiffel Tower | 1889 | | Paris |
| $e_5$ | Miss Liberty | 1886 | Gustave Eiffel | Liberty 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 |
|---|---|---|
| $e_{14}$ | $e_{25}$ | $e_3$, $e_{25}$ |

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

# Iterative Blocking - Example

| | Name | Year | Architects | Location |
|---|---|---|---|---|
| $e_1$ | Eiffel Tower | 1889 | Sauvestre | Paris |
| $e_2$ | Statue of Liberty | 1886 | Bartholdi, Eiffel | NY |
| $e_3$ | Lady Liberty | 1885 | Eiffel | Liberty Island, NY |
| $e_4$ | Eiffel Tower | 1889 | | Paris |
| $e_5$ | Miss Liberty | 1886 | Gustave Eiffel | Liberty 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 | Year | Architects | Location |
|---|---|---|---|---|
| $e_1$ | Eiffel Tower | 1889 | Sauvestre | Paris |
| $e_2$ | Statue of Liberty | 1886 | Bartholdi, Eiffel | NY |
| $e_3$ | Lady Liberty | 1885 | Eiffel | Liberty Island, NY |
| $e_4$ | Eiffel Tower | 1889 | | Paris |
| $e_5$ | Miss Liberty | 1886 | Gustave Eiffel | Liberty 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 = {the, statue, of, liberty}

e2 = {lady, liberty, statue}

e3 = {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
- minHash(ei): the first token of ei in the permuted order of tokens

minHash(e1) = 'of'

minHash(e2) = 'lady'

minhash(e3) = '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[minHash(ei) = minHash(ej)] = Jaccard(ei, ej)

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

P[minHash(ei) = minHash(ej)] =
= #A / (#A + #B) =
= Jaccard(ei, ej)

| 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 |

Type C row

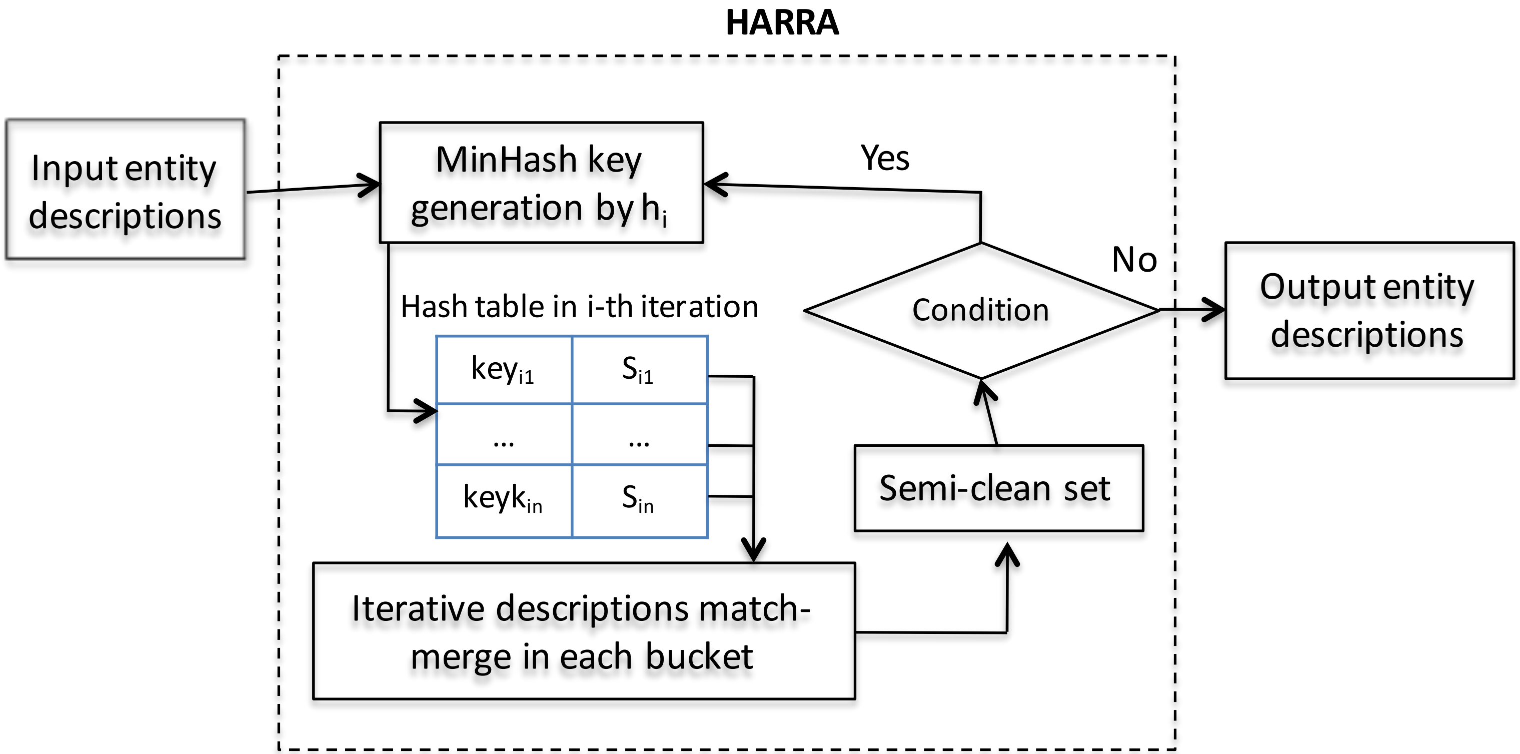
Type B row

Type A row

- Create several random permutations (h1, …, hn) of the tokens, n << #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)

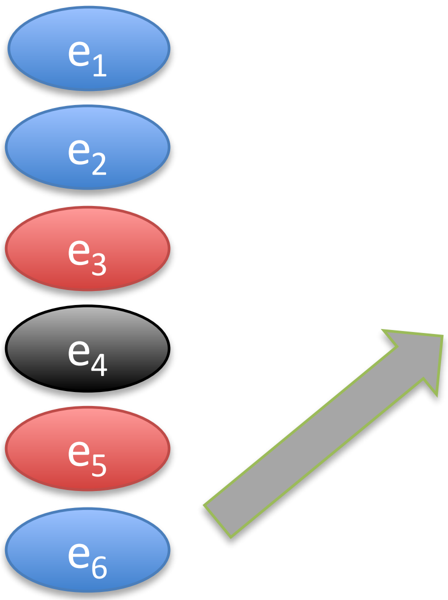



Scalability: A single hash table is used

- 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:

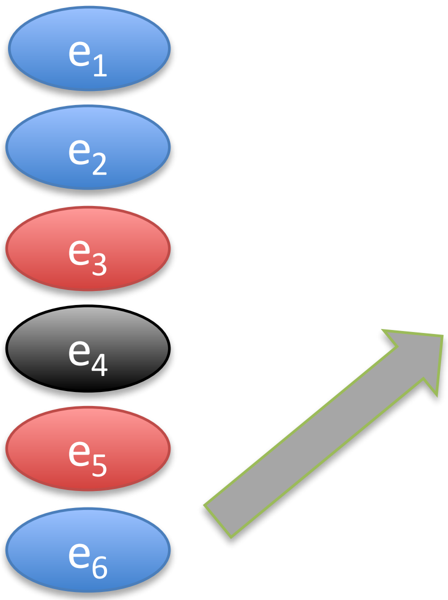
| Keys | Values |
|---|---|
| Blue | $e_1$ $e_2$ |
| Red | $e_3$ $e_5$ |
| Black | $e_4$ |

# 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 | $e_1$ $e_2$ |
| Red | $e_3$ $e_5$ |
| Black | $e_4$ |

# 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

$e_1$

$e_2$

$e_3$

$e_4$

$e_5$

$e_6$

Hash Table:

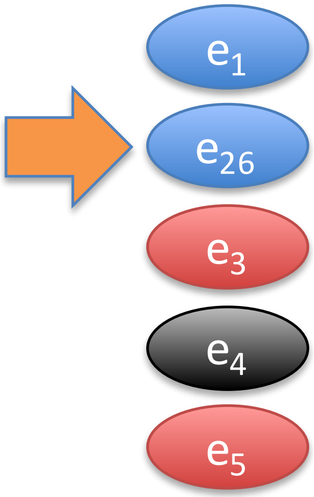
| Keys | Values |
|---|---|
| Blue | $e_1$ $e_{26}$ |
| Red | $e_3$ $e_5$ |
| Black | $e_4$ |

# HARRA - Example

Continue until:

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

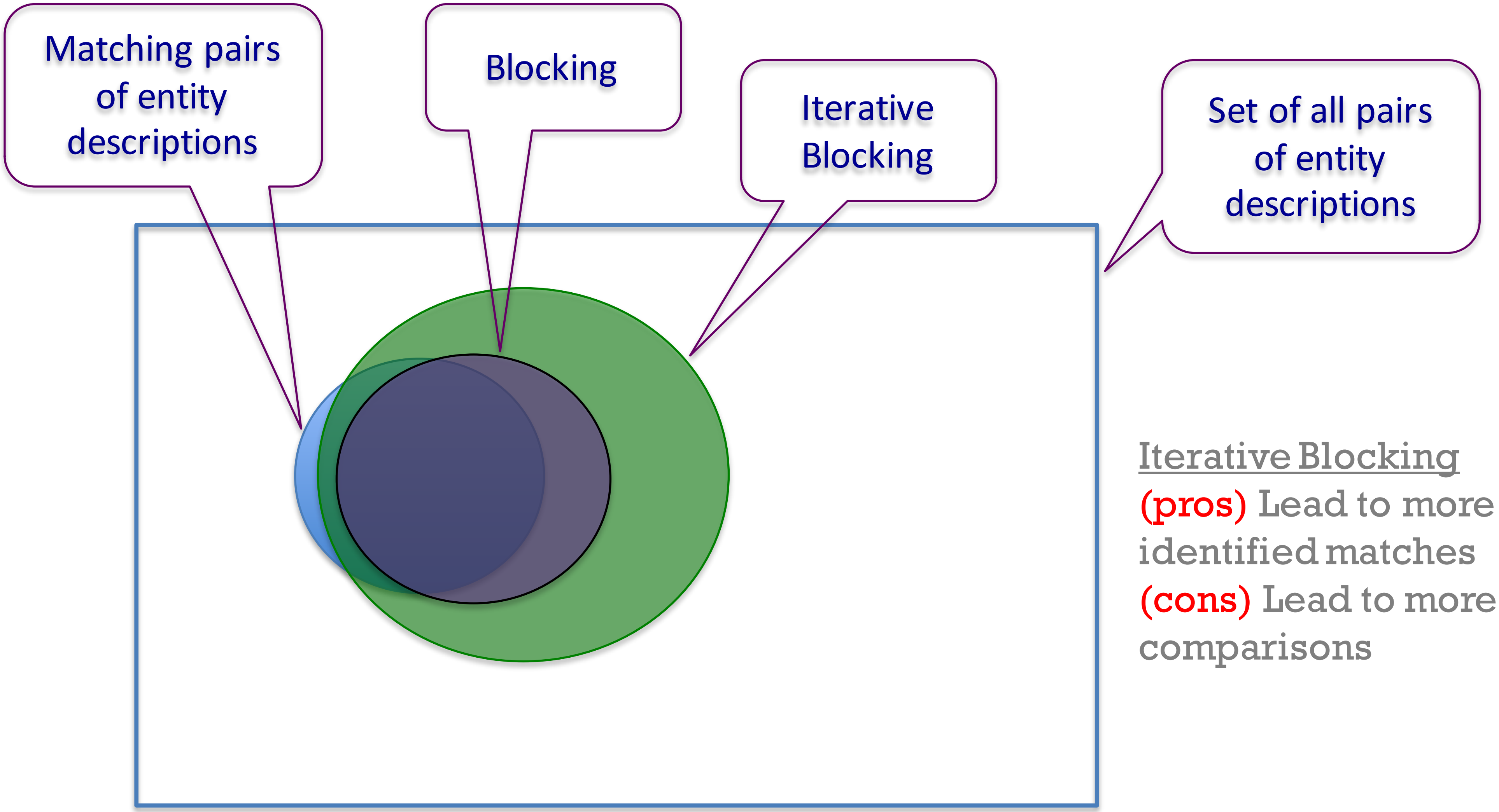
Re-initialize the input:




Hash Table:

| Keys | Values |
|---|---|
| Blue | |
| Red | |
| Black | |

# Blocking vs Iterative Blocking




<u>Iterative Blocking</u>

(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!*

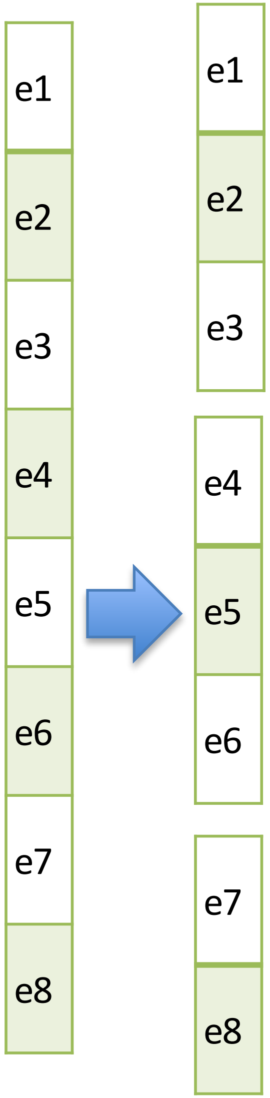
The map phase reflects blocking (re-distribute descriptions)

The reduce phase reflects entity resolution (check for matches)

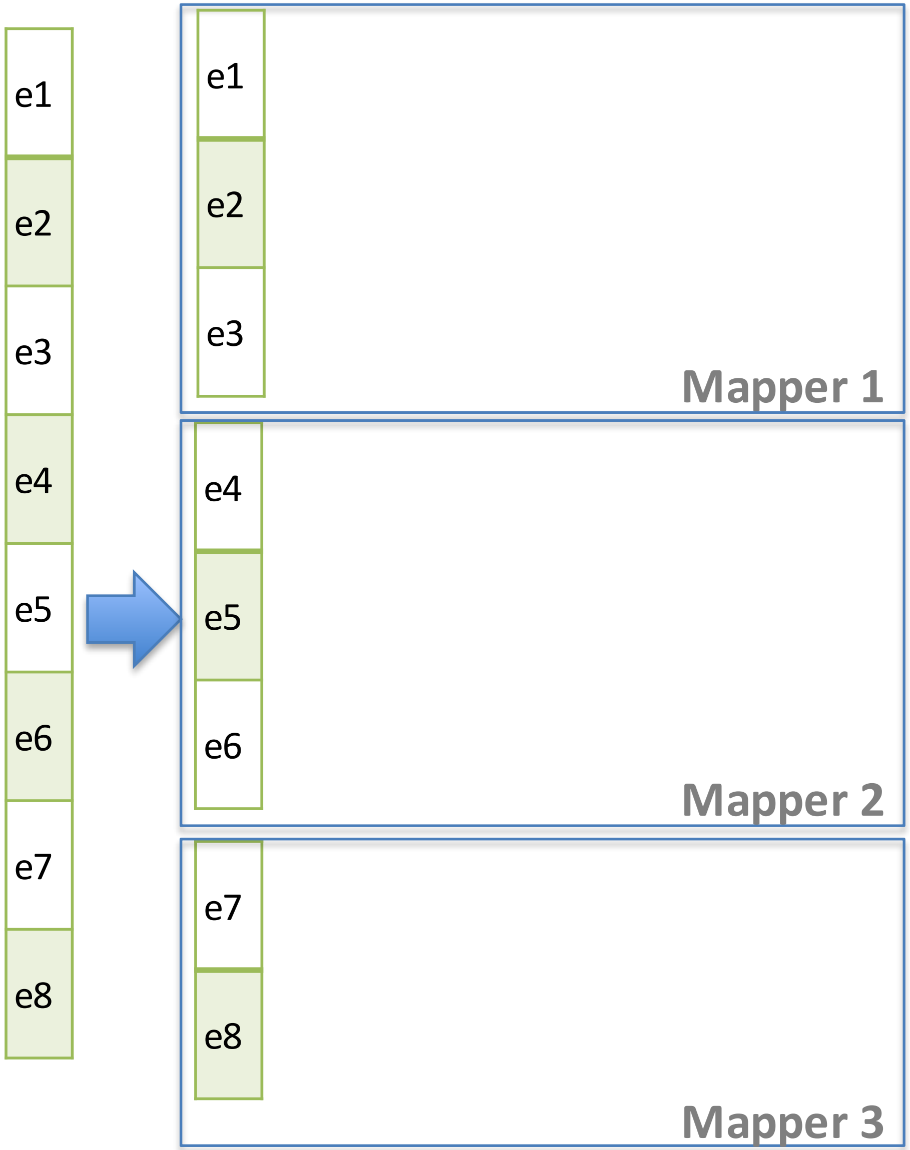
# MapReduce – Input Data

| |
|---|
| e1 |
| e2 |
| e3 |
| e4 |
| e5 |
| e6 |
| e7 |
| e8 |

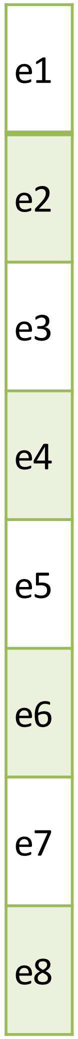
# MapReduce – Input Data Partitioning

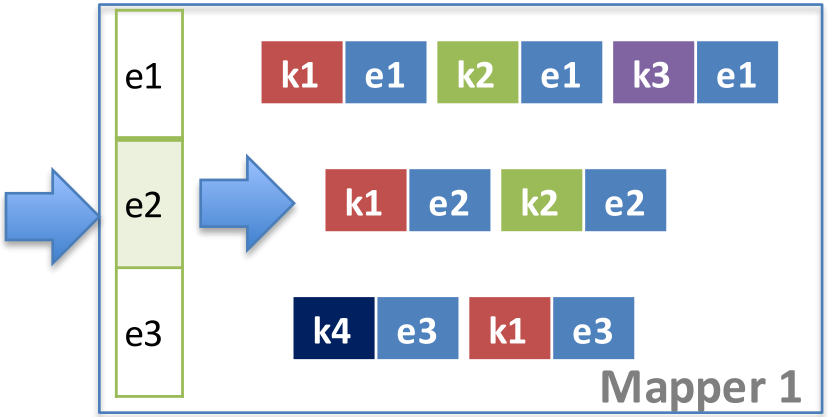


# MapReduce – Mapper Input



# MapReduce – Mapper Example



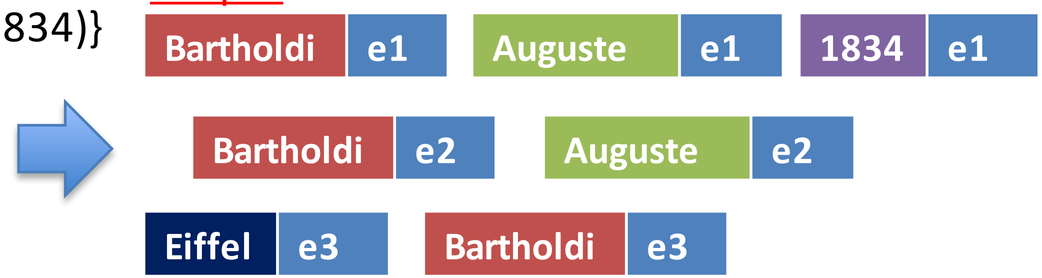




Input :
e1={(name, Auguste Bartholdi),(year,1834)}

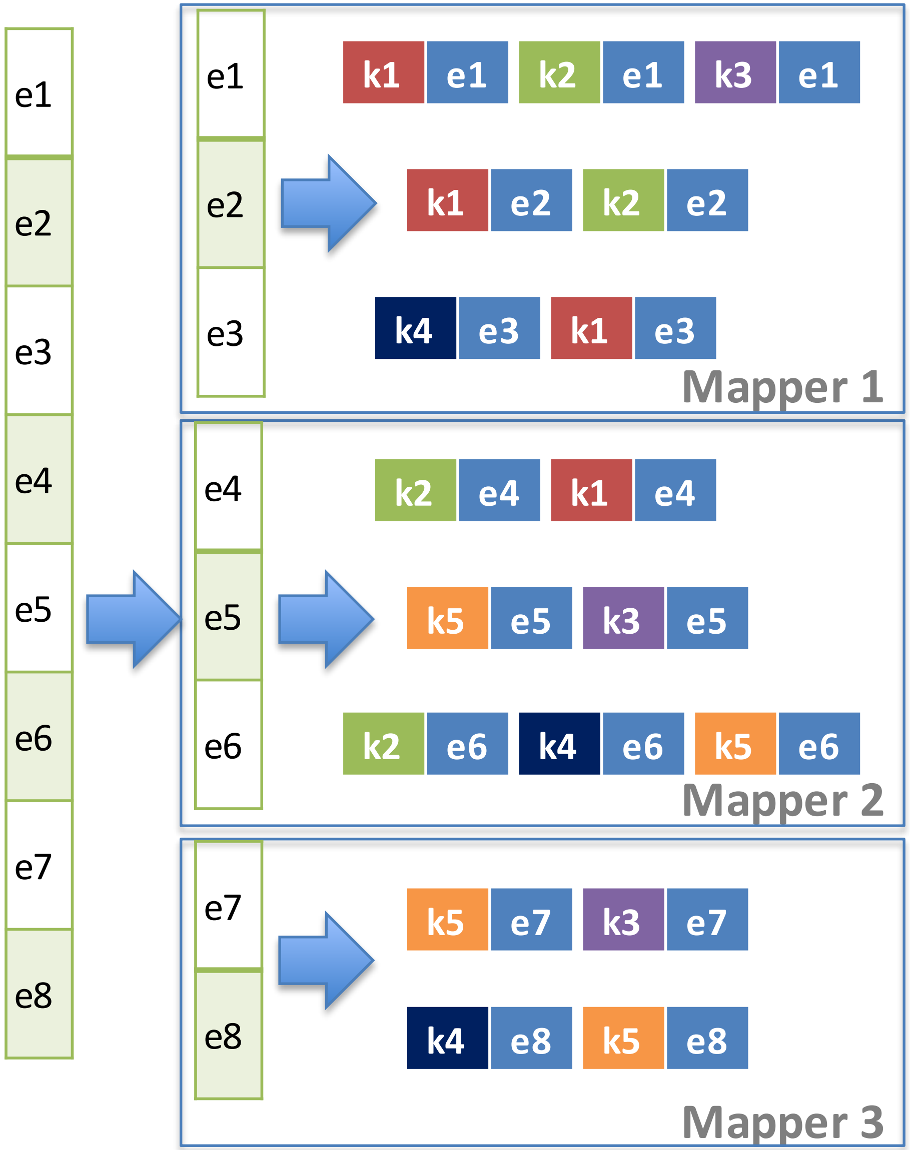
e2={(about, Auguste Bartholdi)}

e3={(architects, Bartholdi Eiffel)

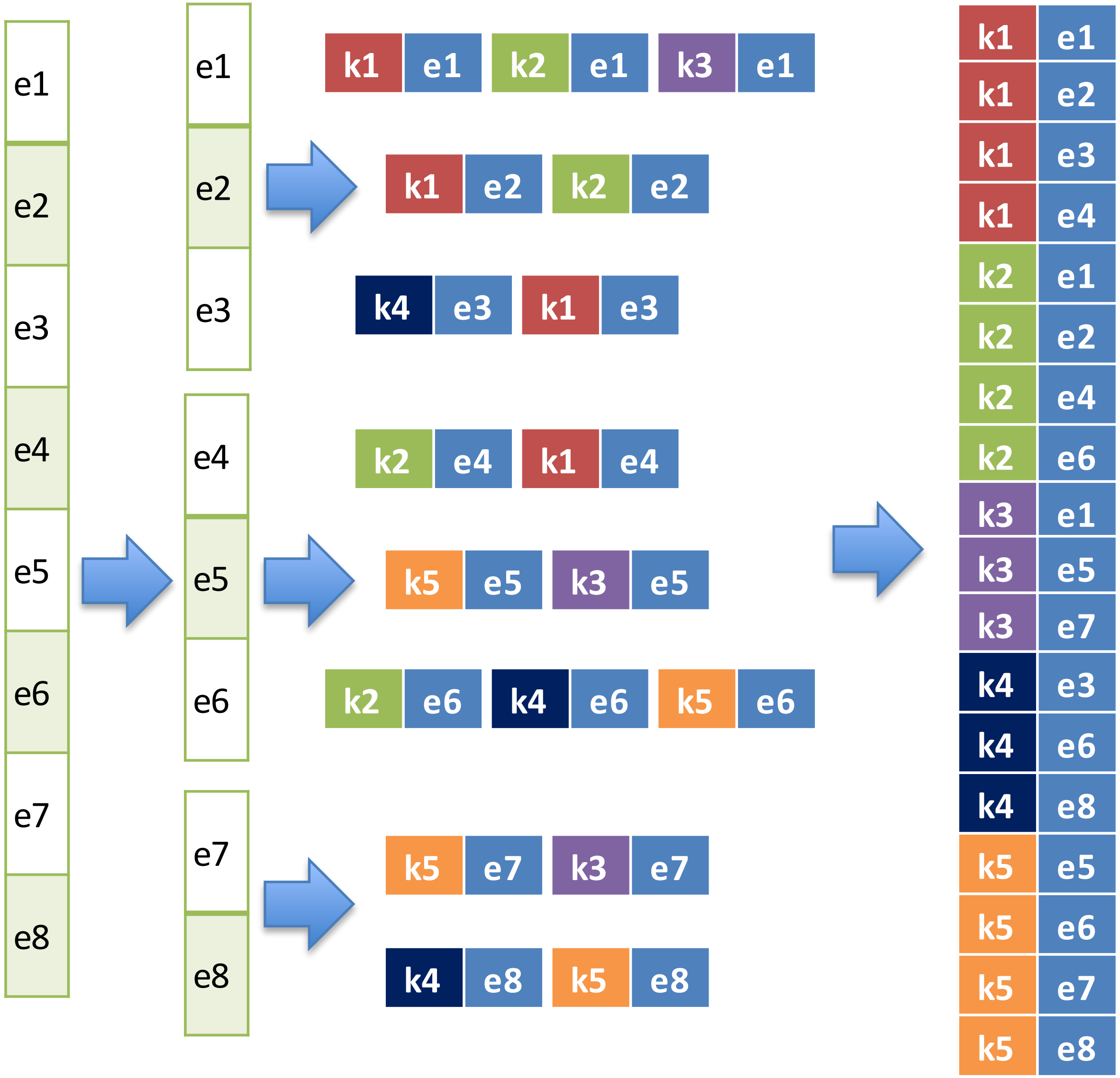
Output :



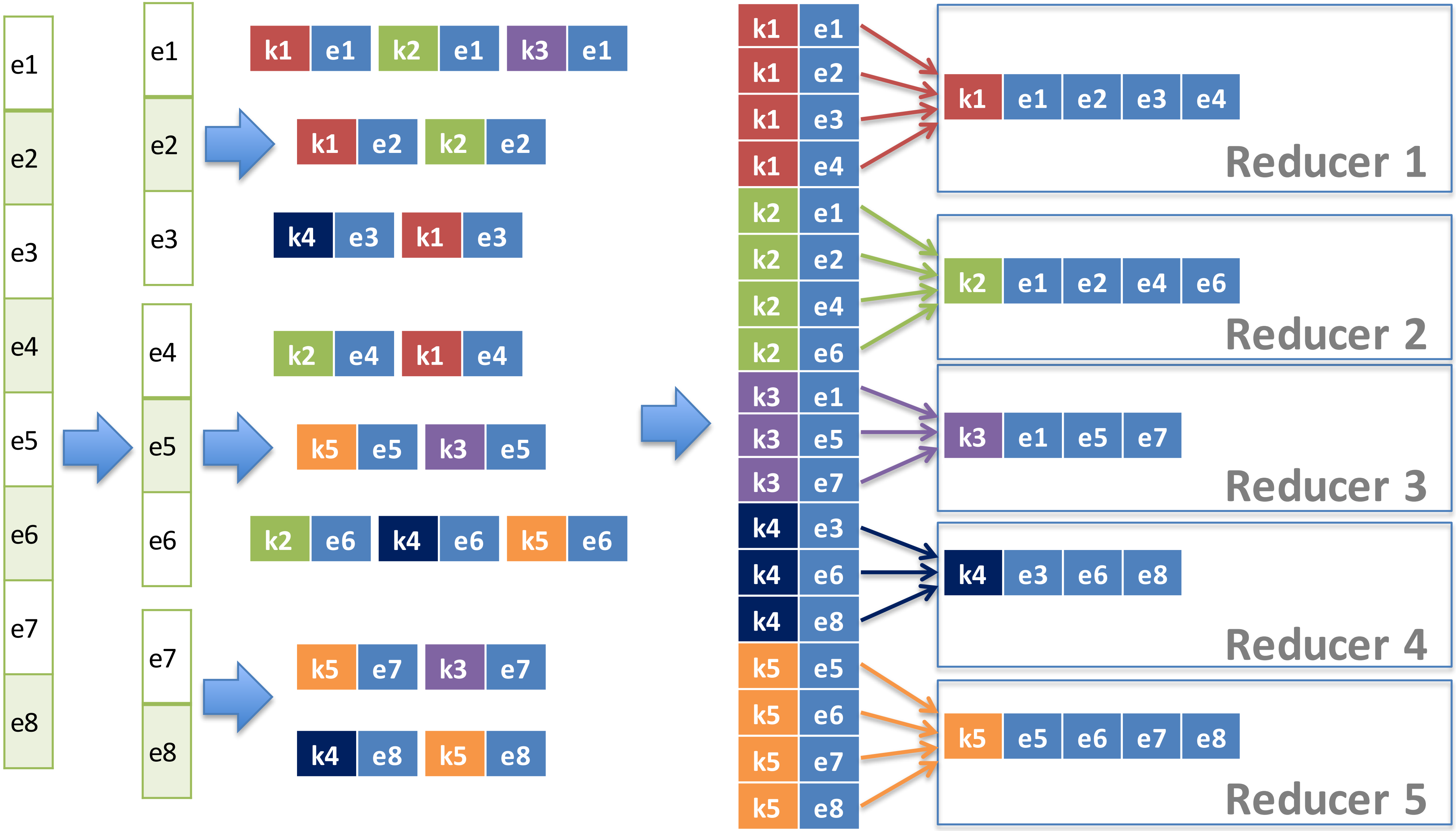
# MapReduce – Mapper Output



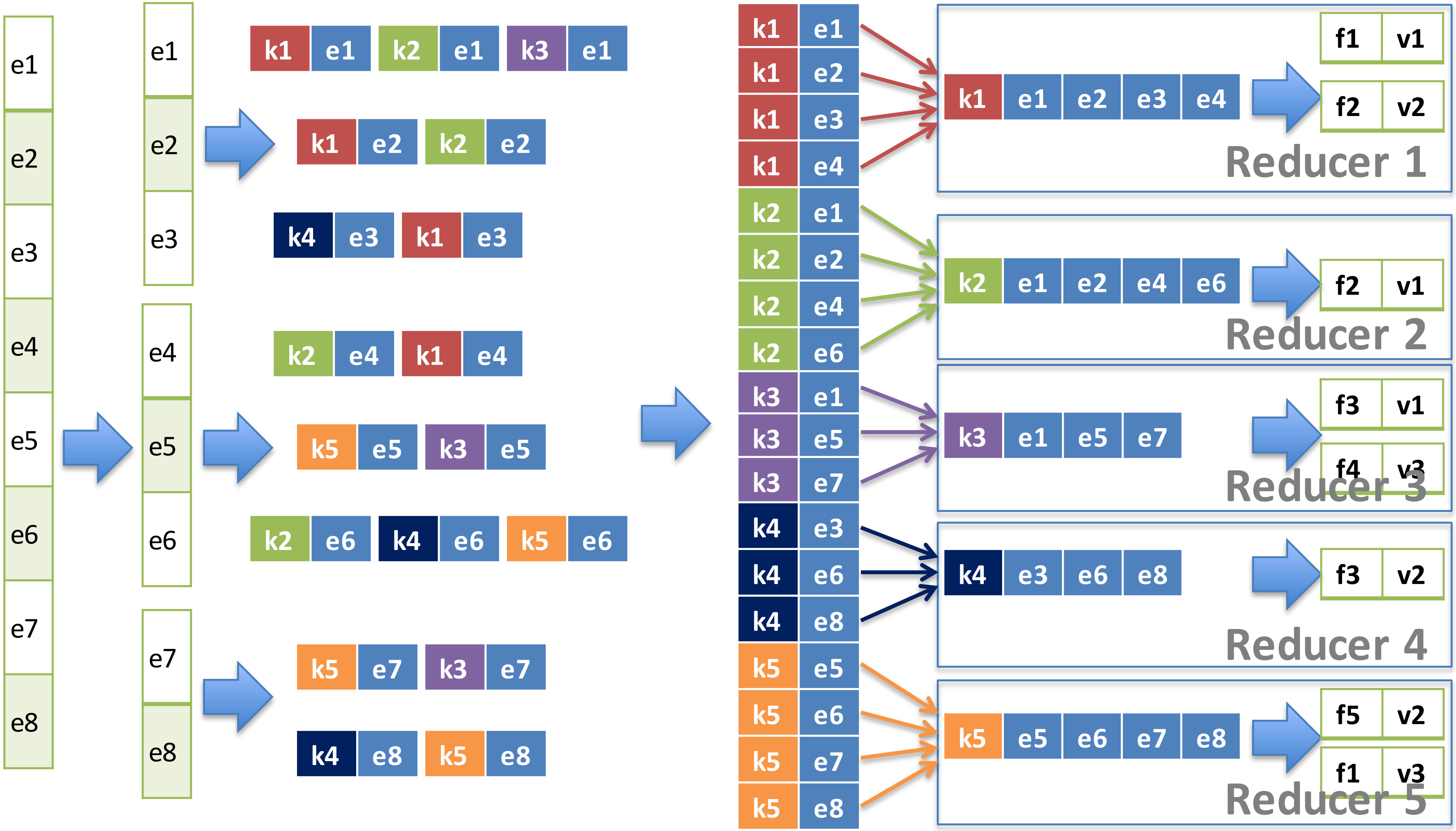
# MapReduce – Shuffling & Sorting



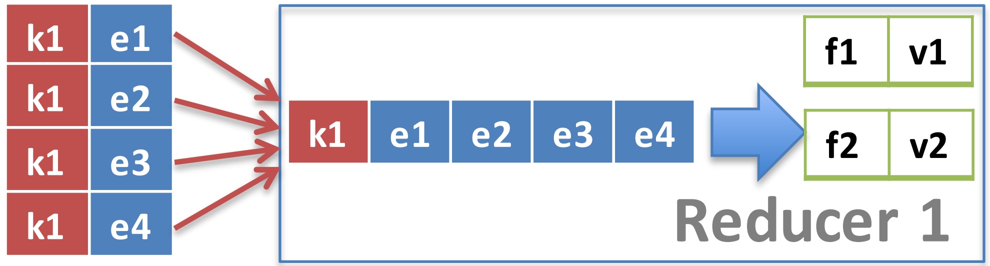
# MapReduce – Merging

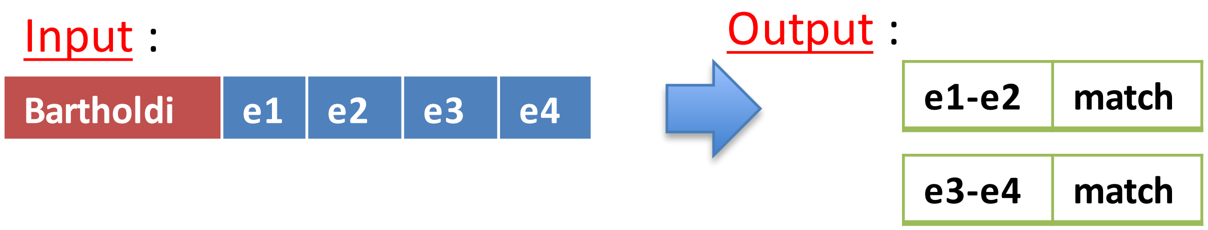


# MapReduce – Reducer



# MapReduce – Reducer Example



# Dedoop – Standard Blocking [Kolb et al. 2012]

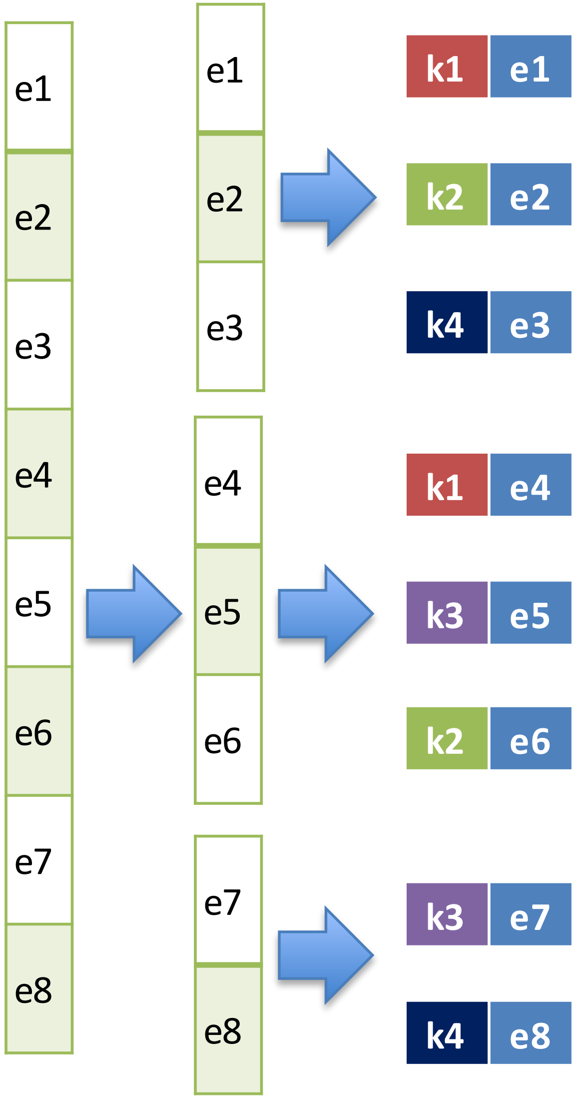
*Dedoop performs standard blocking using MapReduce*

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

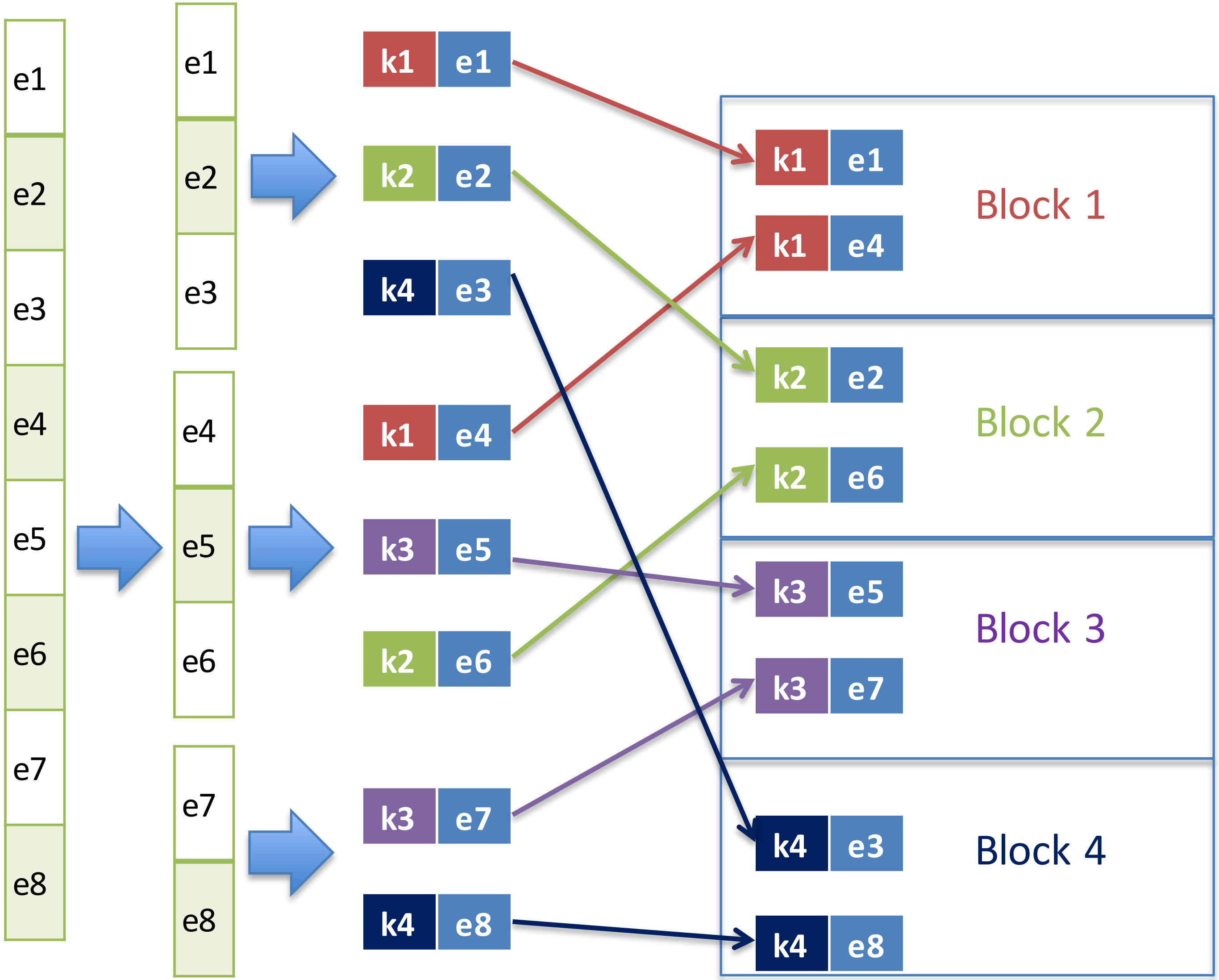
The partitioning operates on the BKVs and distributes (key, value) pairs among reduce tasks
- All entities sharing the same BKV are assigned to the same reduce task

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

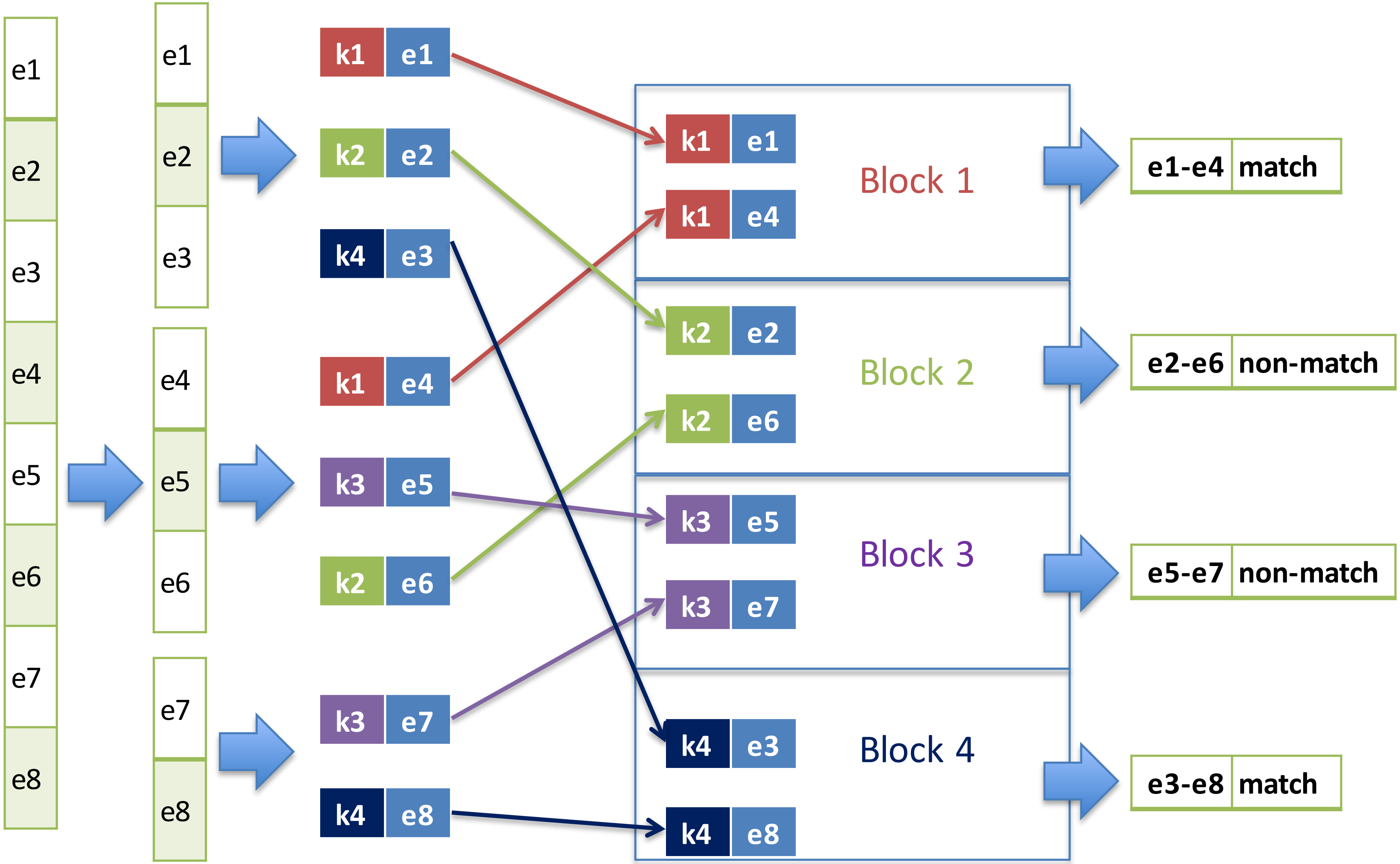
# Dedoop – Mapper: BKVs as intermediate keys



# Dedoop – Mappers: Build Blocks

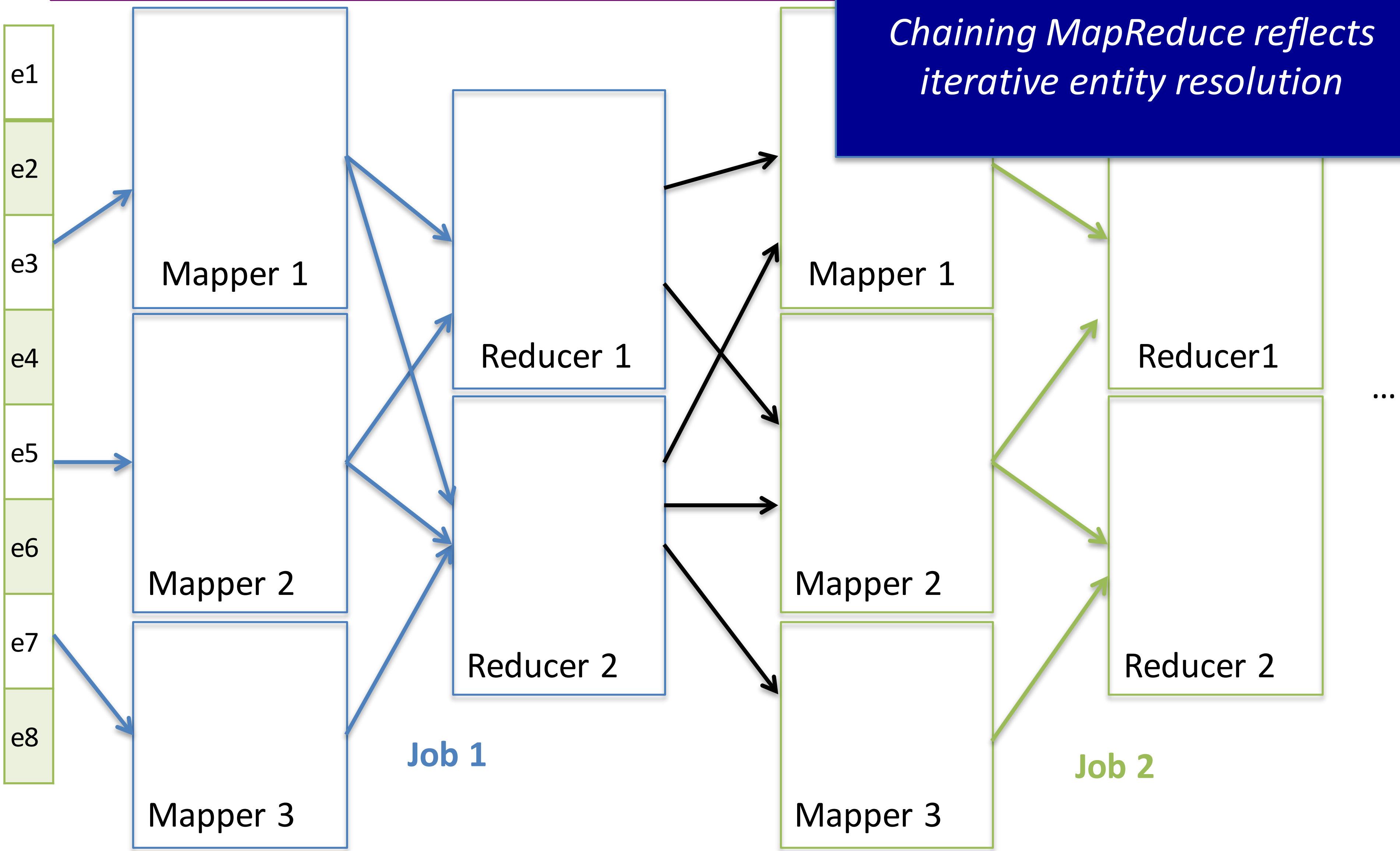


# Dedoop – Reducers: Compare Block Contents



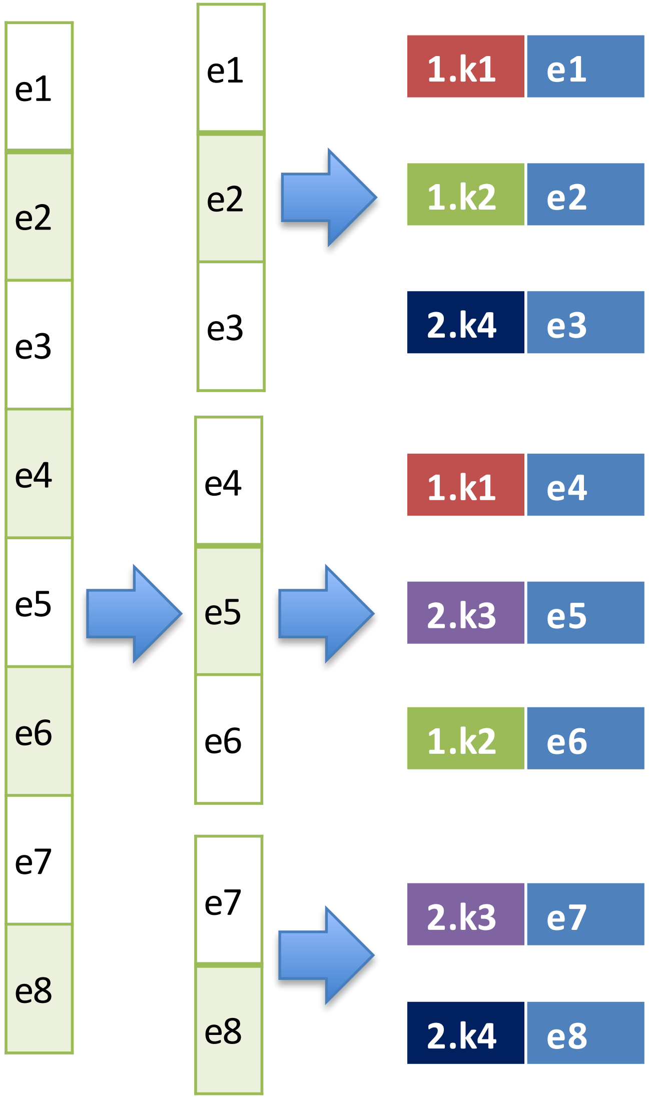
# Chaining MapReduce Jobs

*Chaining MapReduce reflects iterative entity resolution*




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

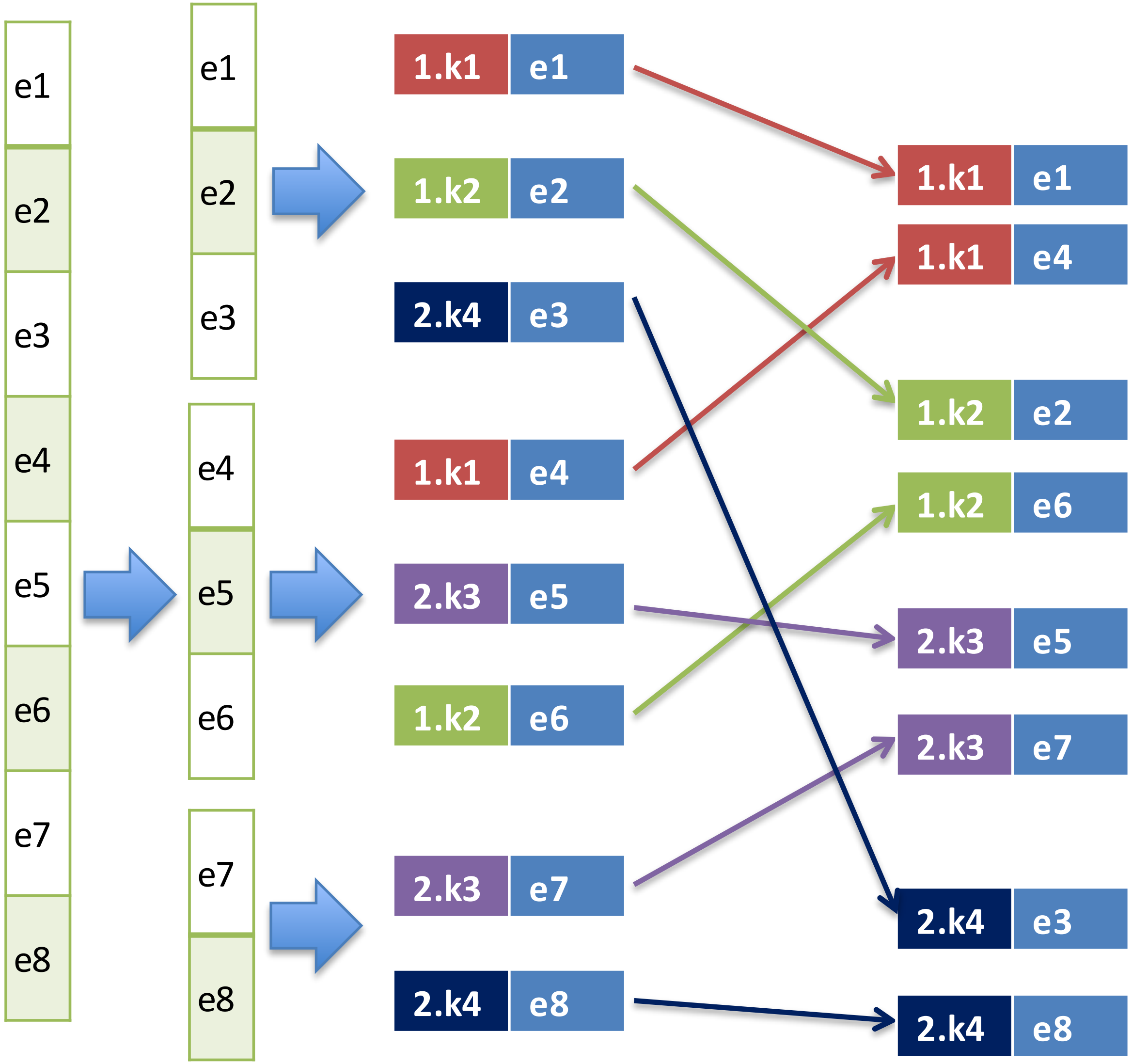
# Dedoop – Sorted Neighborhood [Kolb et al. 2011]



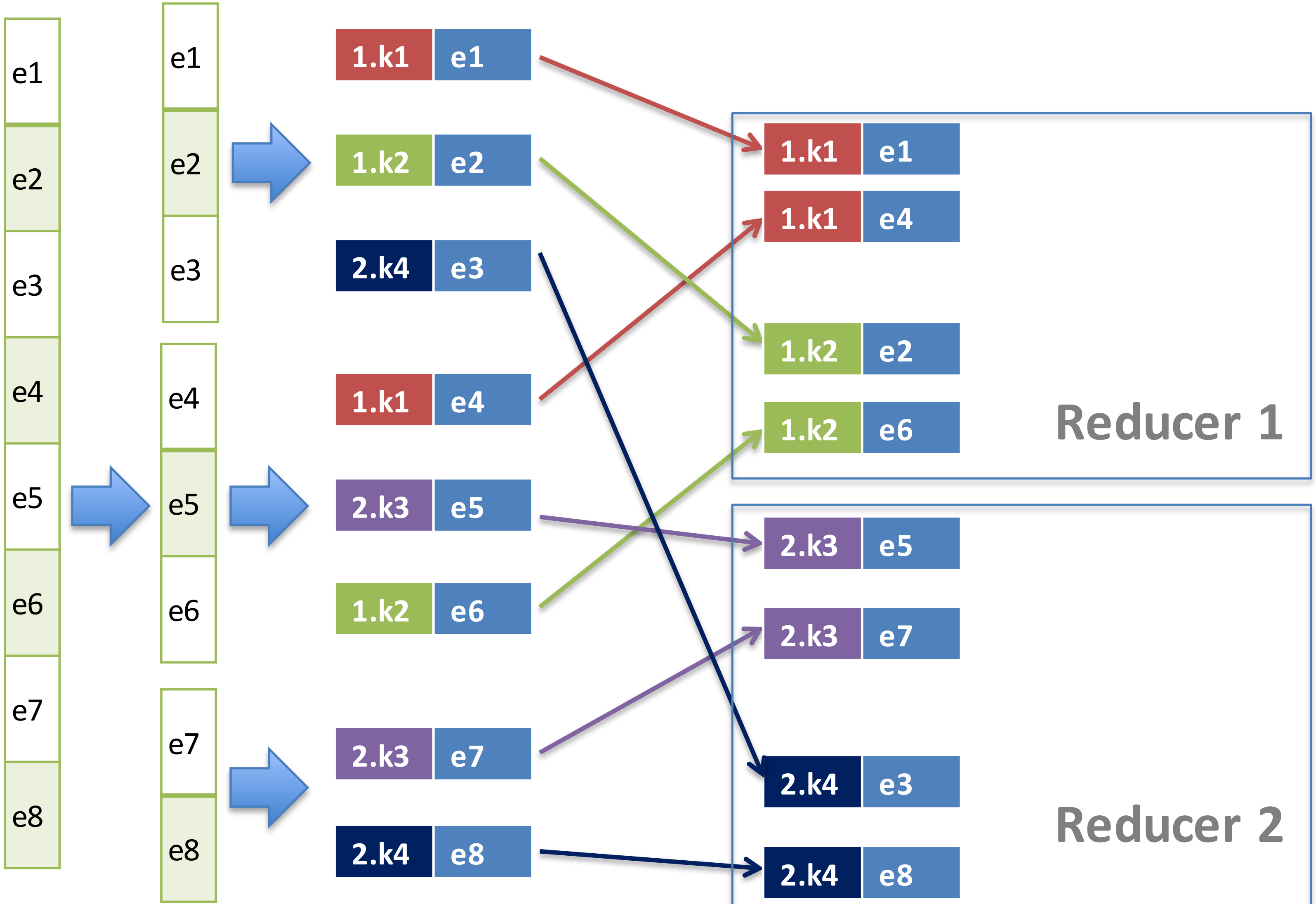

composite key = (partitionID, BKV)
partitionID(BKV) = 1, if BKV < "k3"
partitionID(BKV) = 2, else

(we know that we have two reducers available)

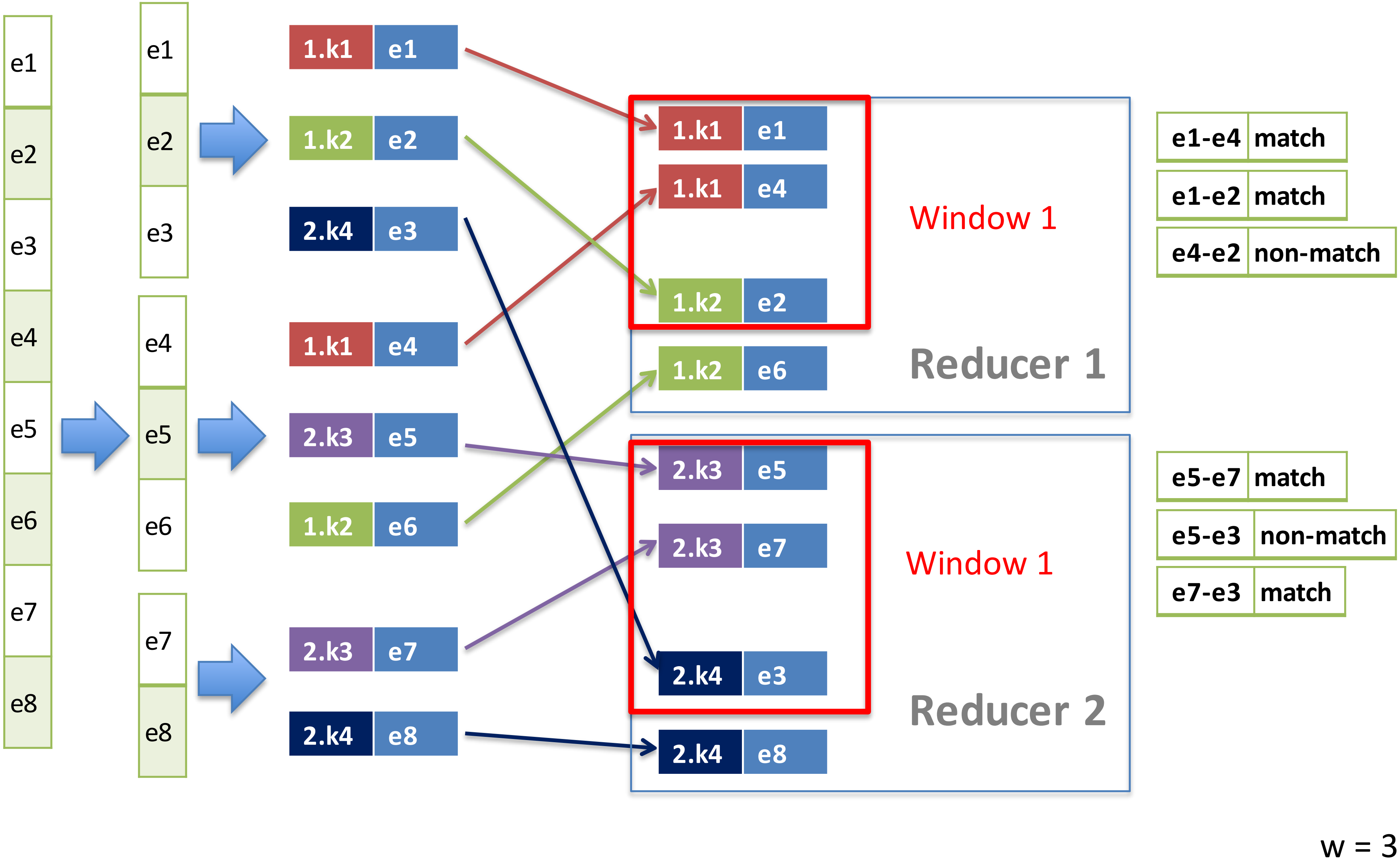
# Dedoop SN: Sorting the Keys



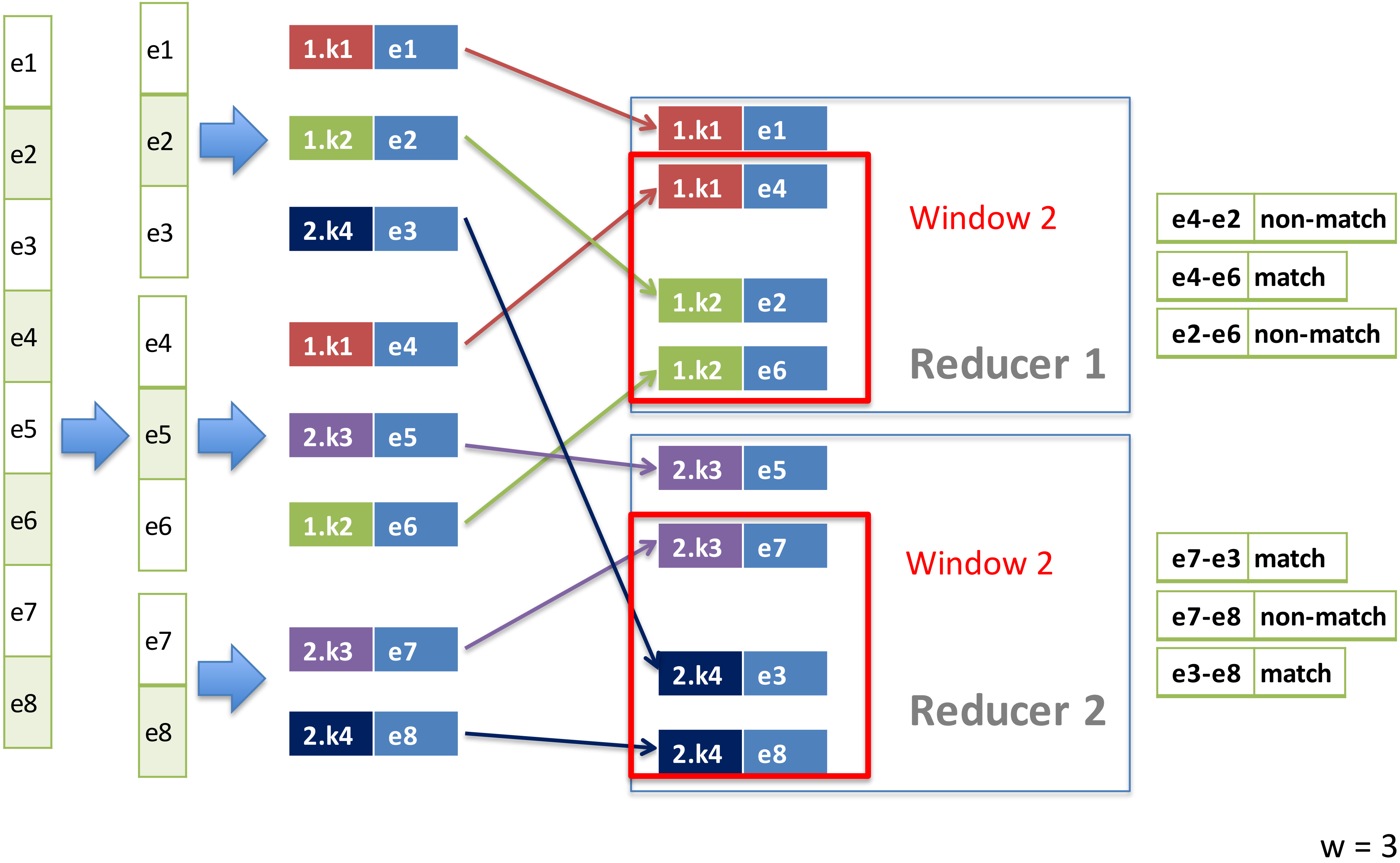
# Dedoop SN: Reducers Apply the Sliding Window



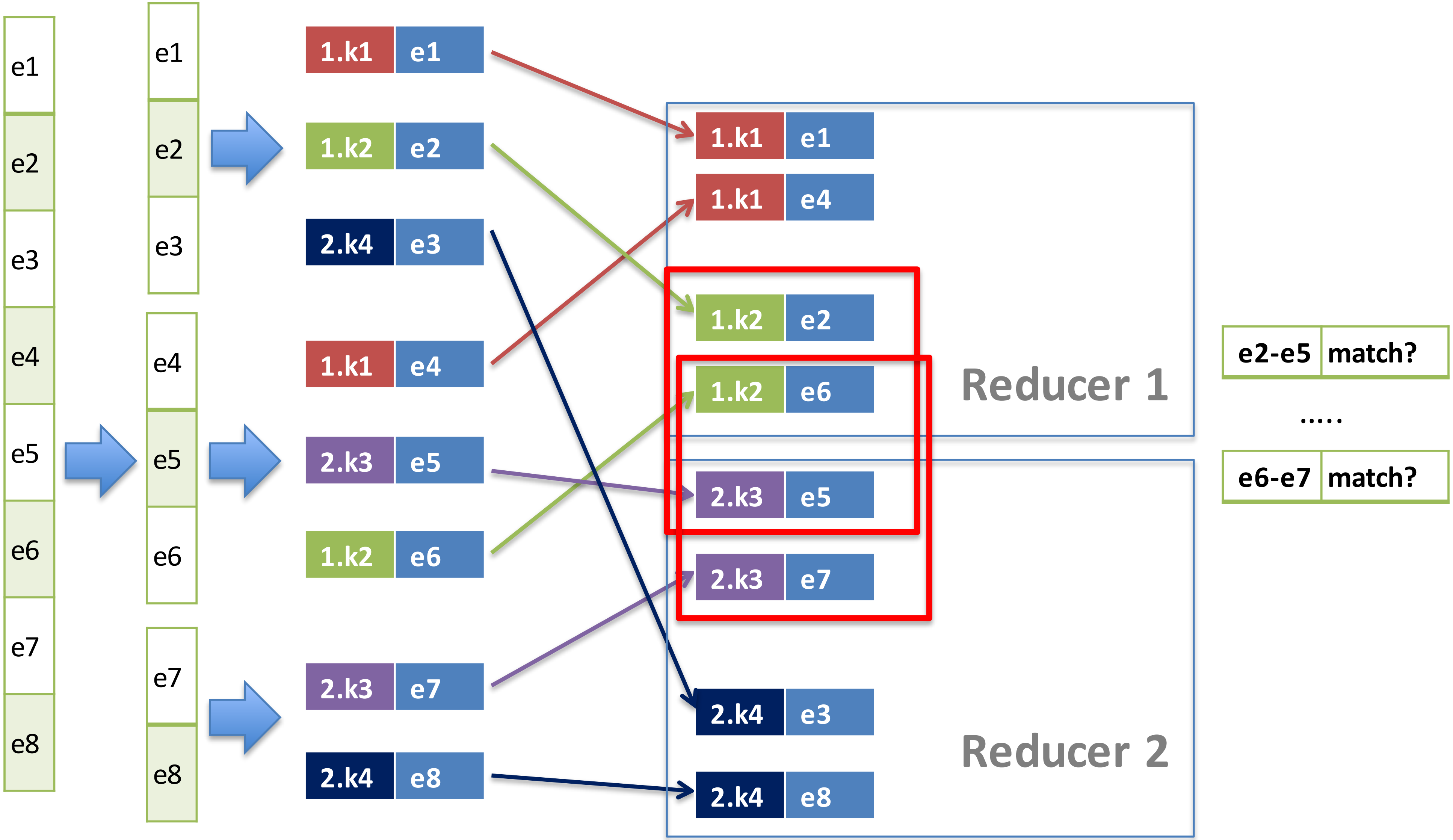
# Dedoop SN: Reducers Apply the Sliding Window



# Dedoop SN: Reducers Apply the Sliding Window



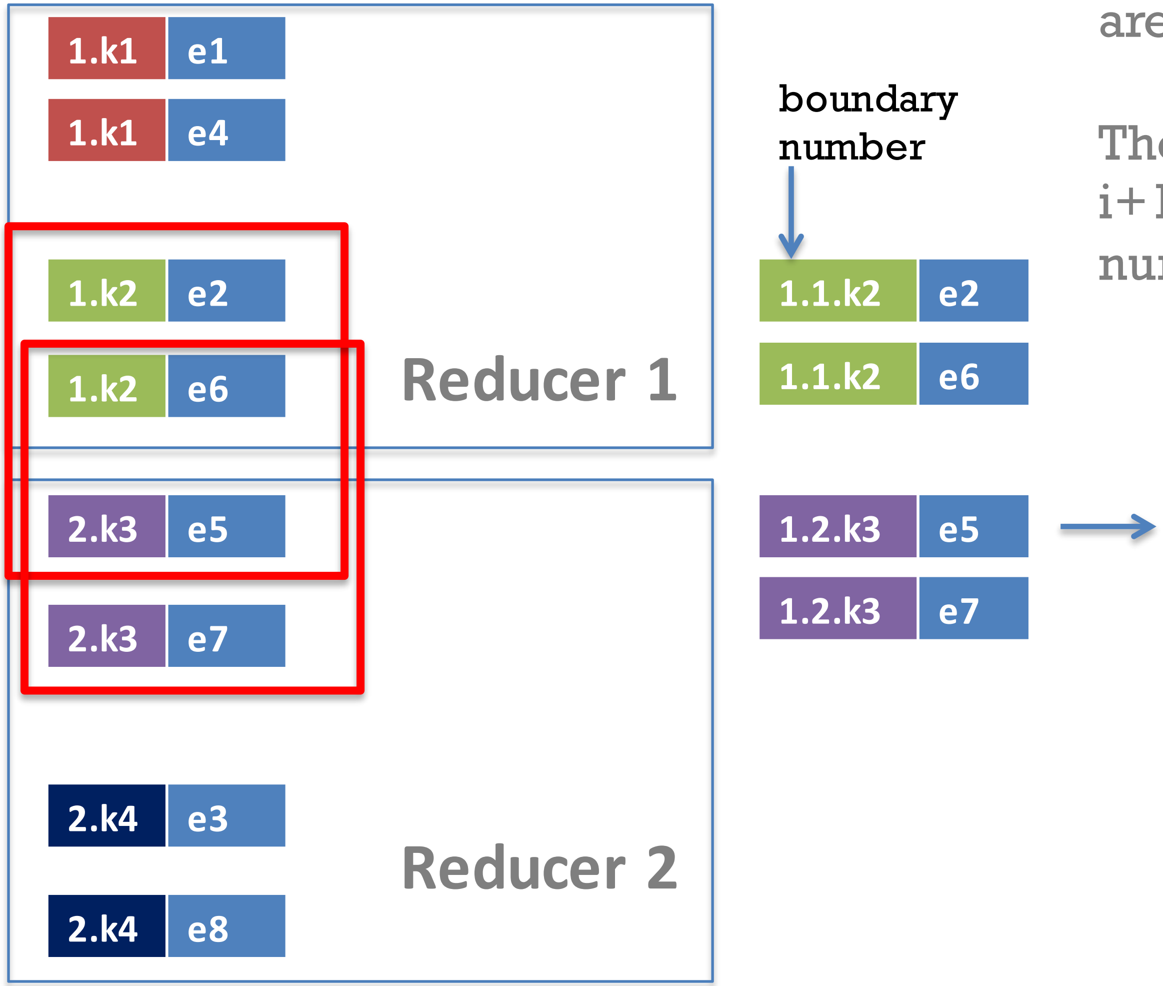
# Dedoop SN: We Also Need To Compare The Boundary Entities




w = 3

# Dedoop SN: Reducers Also Output the Boundary Descriptions

Add a boundary number prefix to the output composite keys




**Boundary number:**

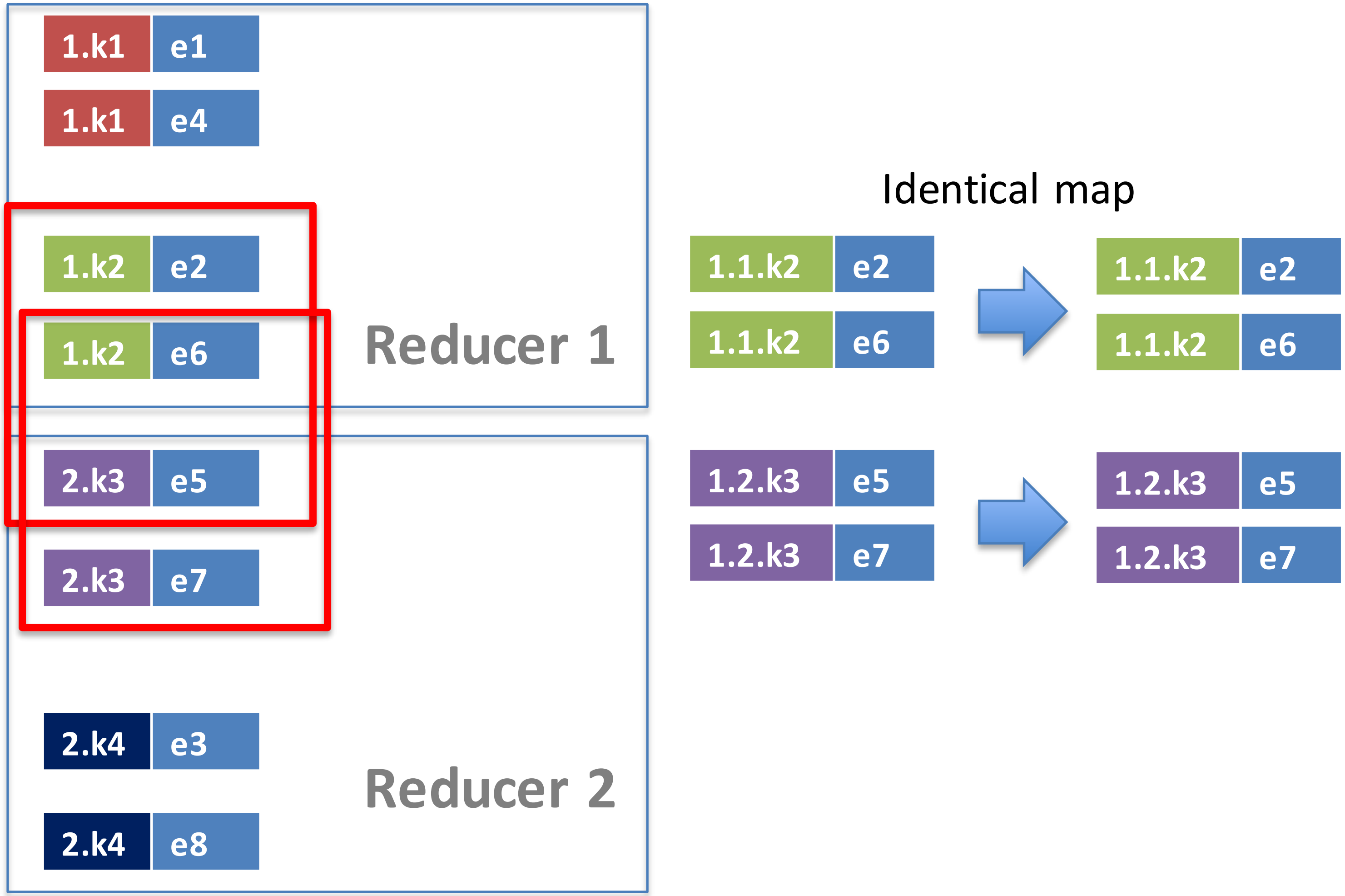
The last w-1 descriptions of reducer i are assigned the boundary number i

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

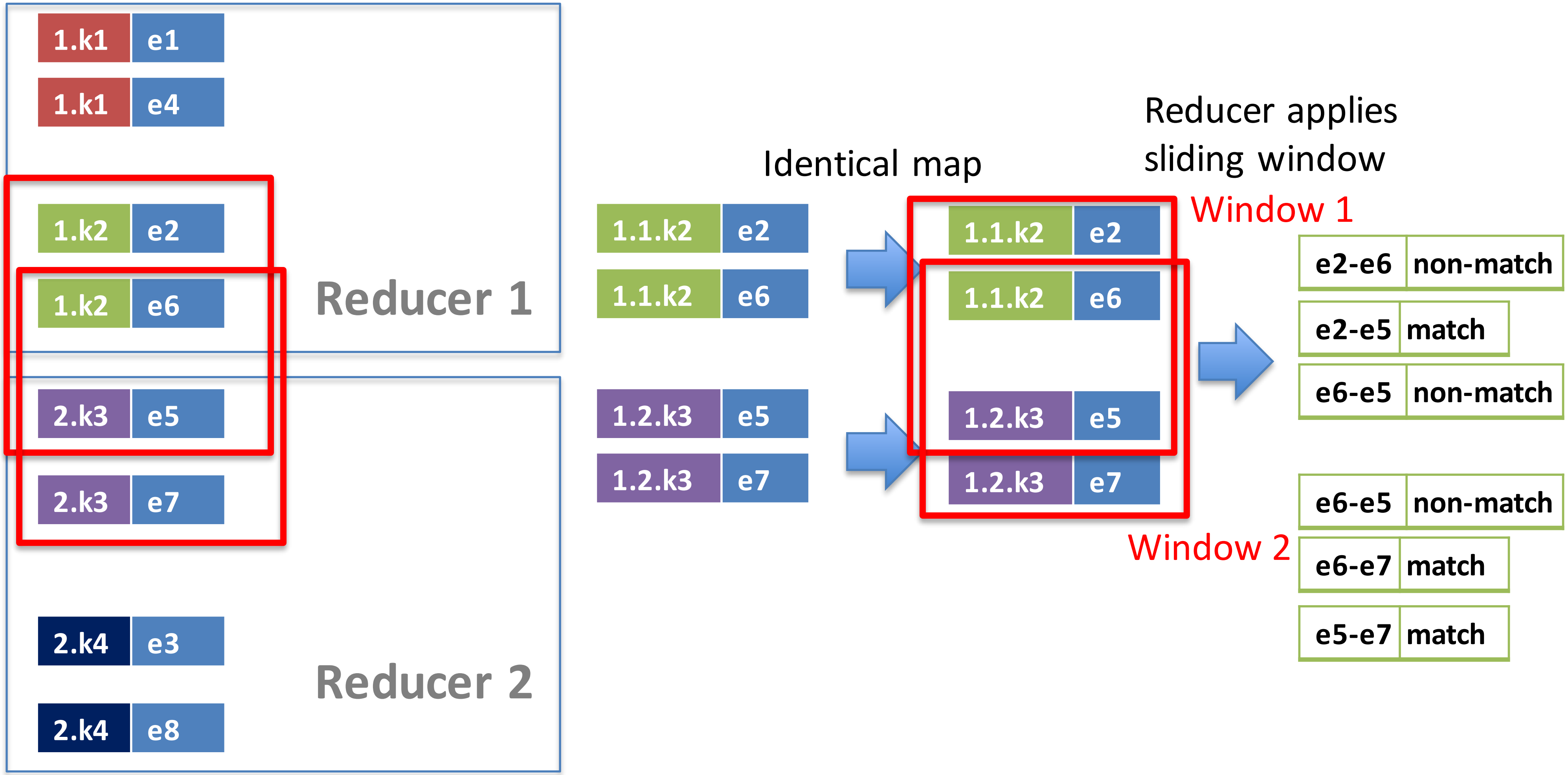
The actual blocking key of e5 is k3, it was assigned to reducer 2 and it is associated with boundary number 1

w = 3

# Dedoop SN: New MapReduce Job for the Boundary Pairs

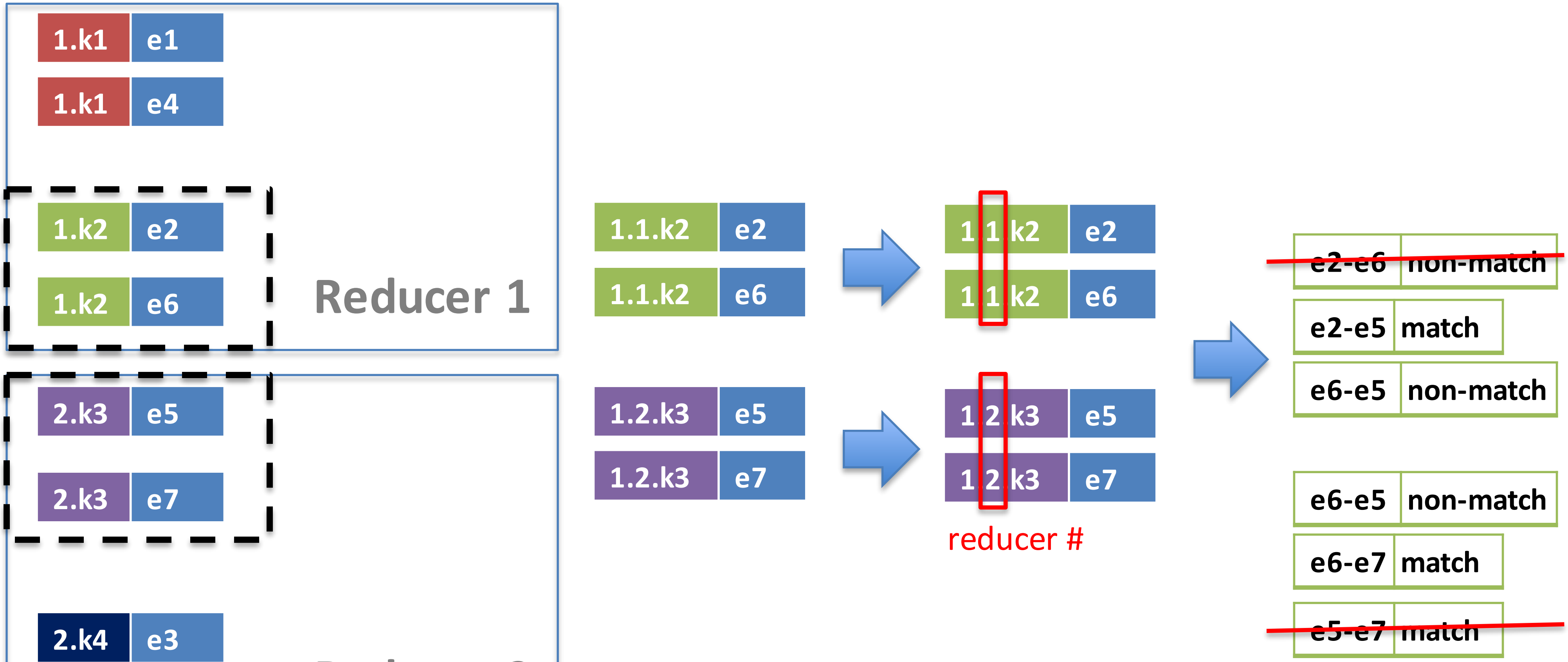


# Dedoop SN: Partition by Boundary Number



*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)

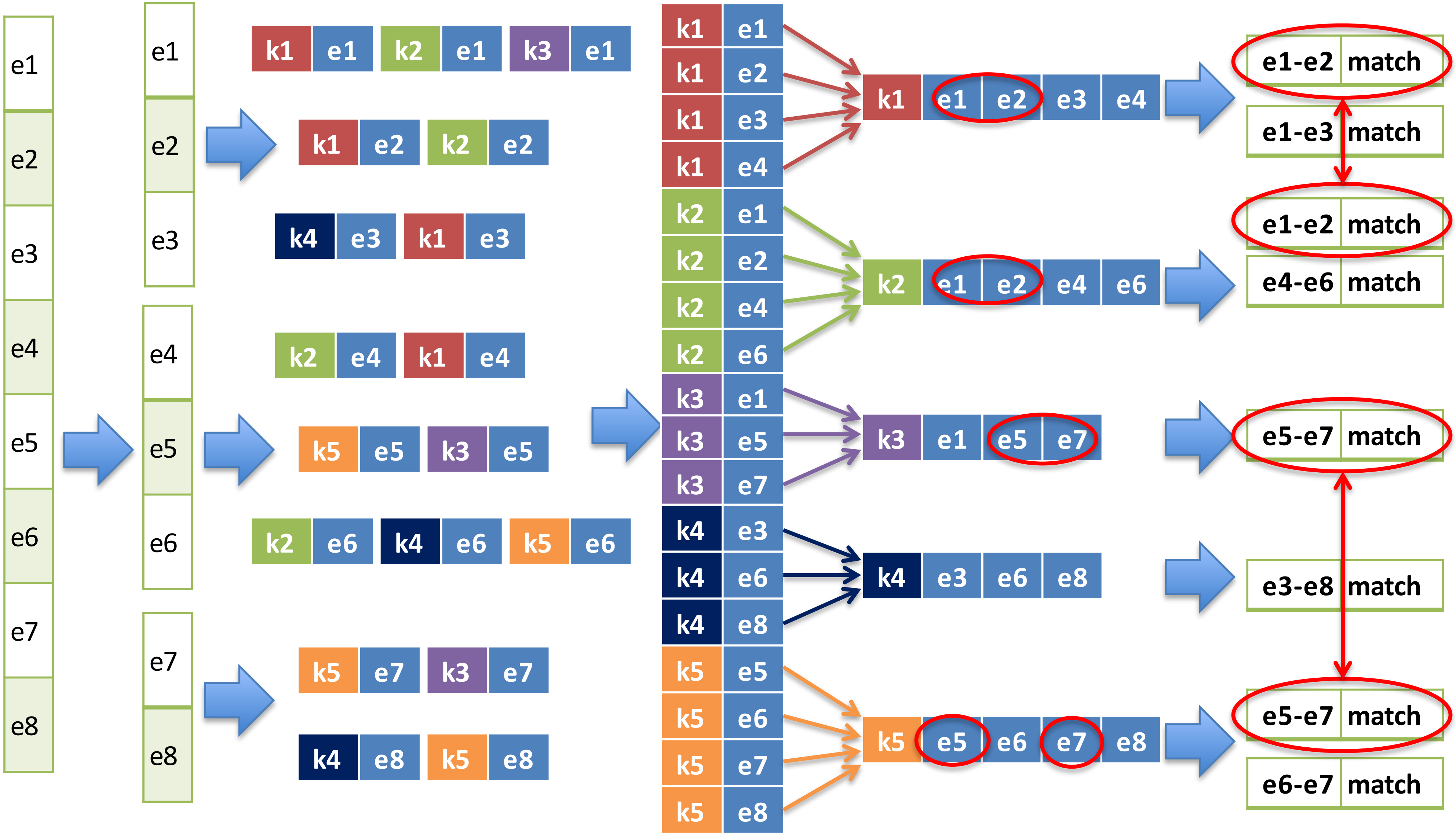
# Don’t match twice [Kolb et al. 2013]

Overlapping blocks lead to repeated comparisons

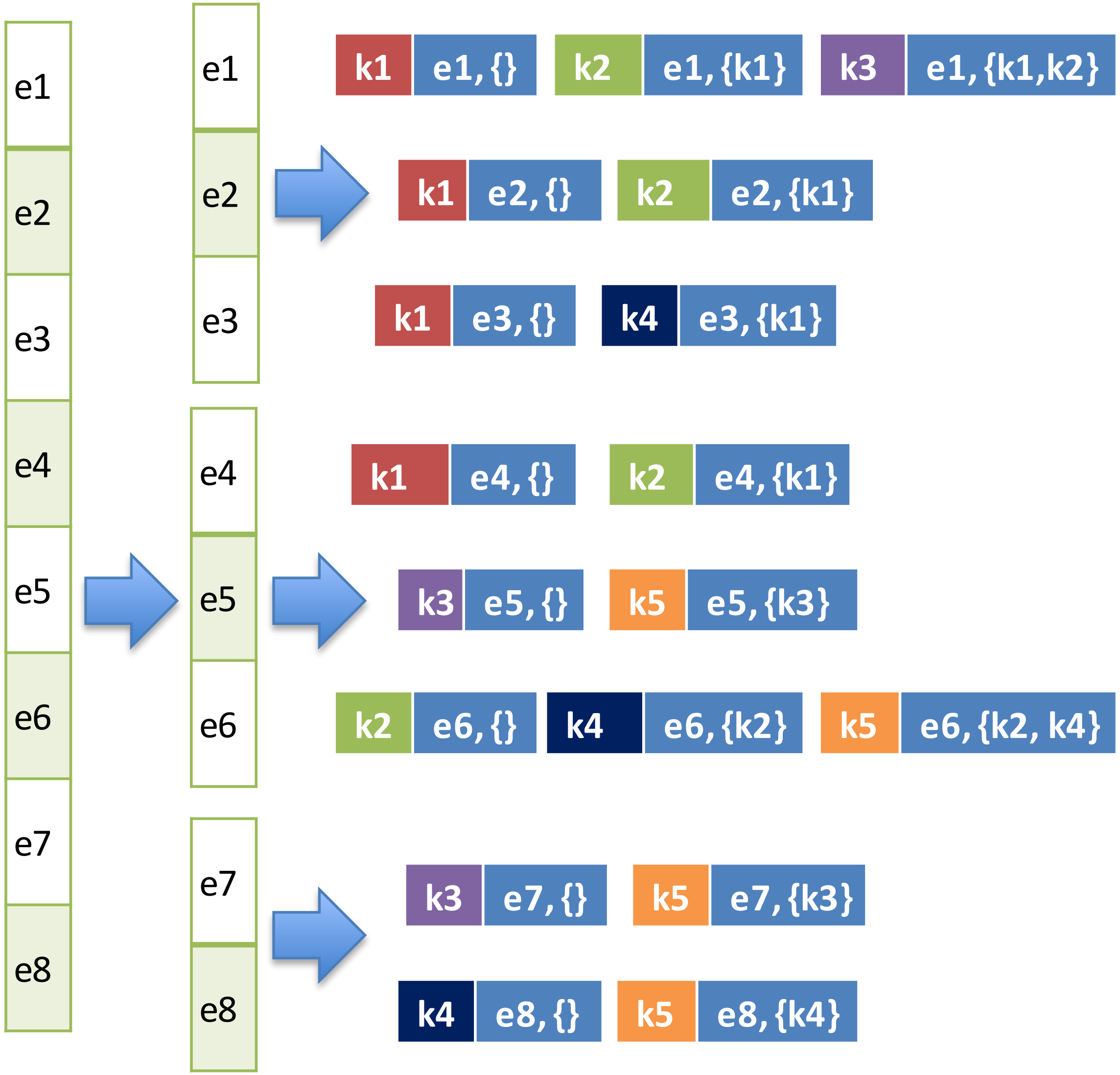
Adopt Comparison Propagation [Papadakis et al. 2012] to MapReduce:

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

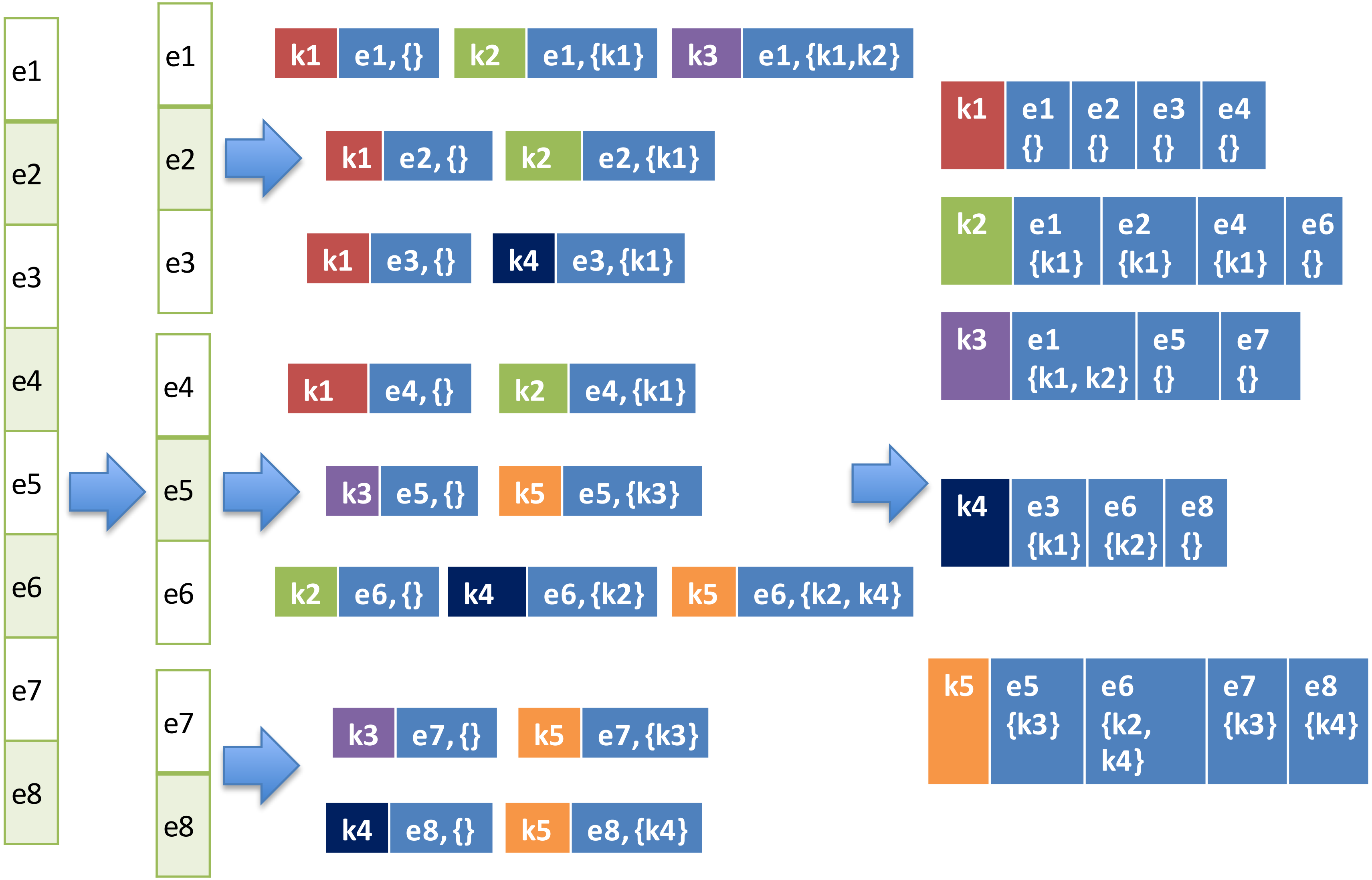
# Overlapping Blocks Lead to Repeated Comparisons



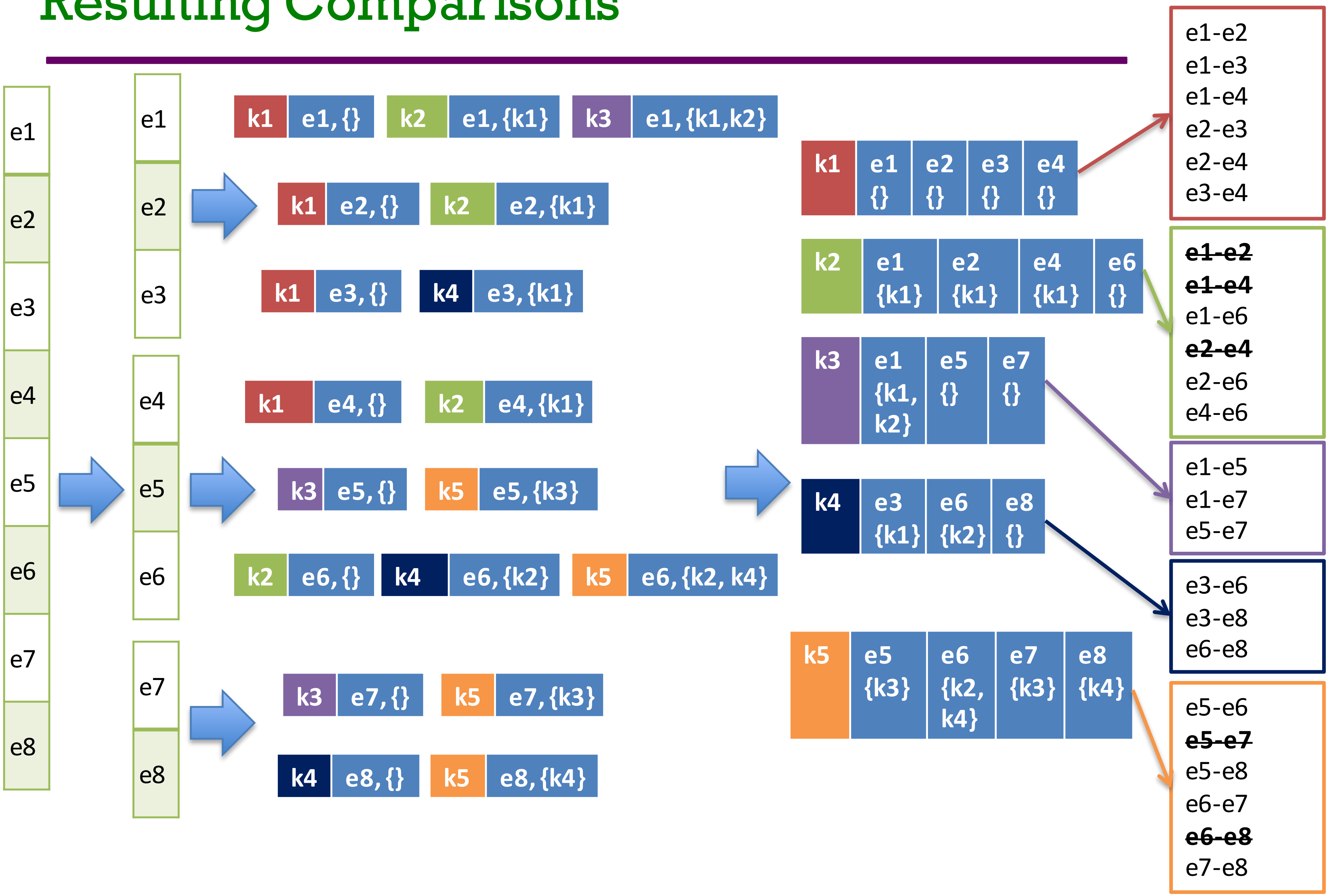
# Map: Append the Subset of Smaller Keys for the Same Description



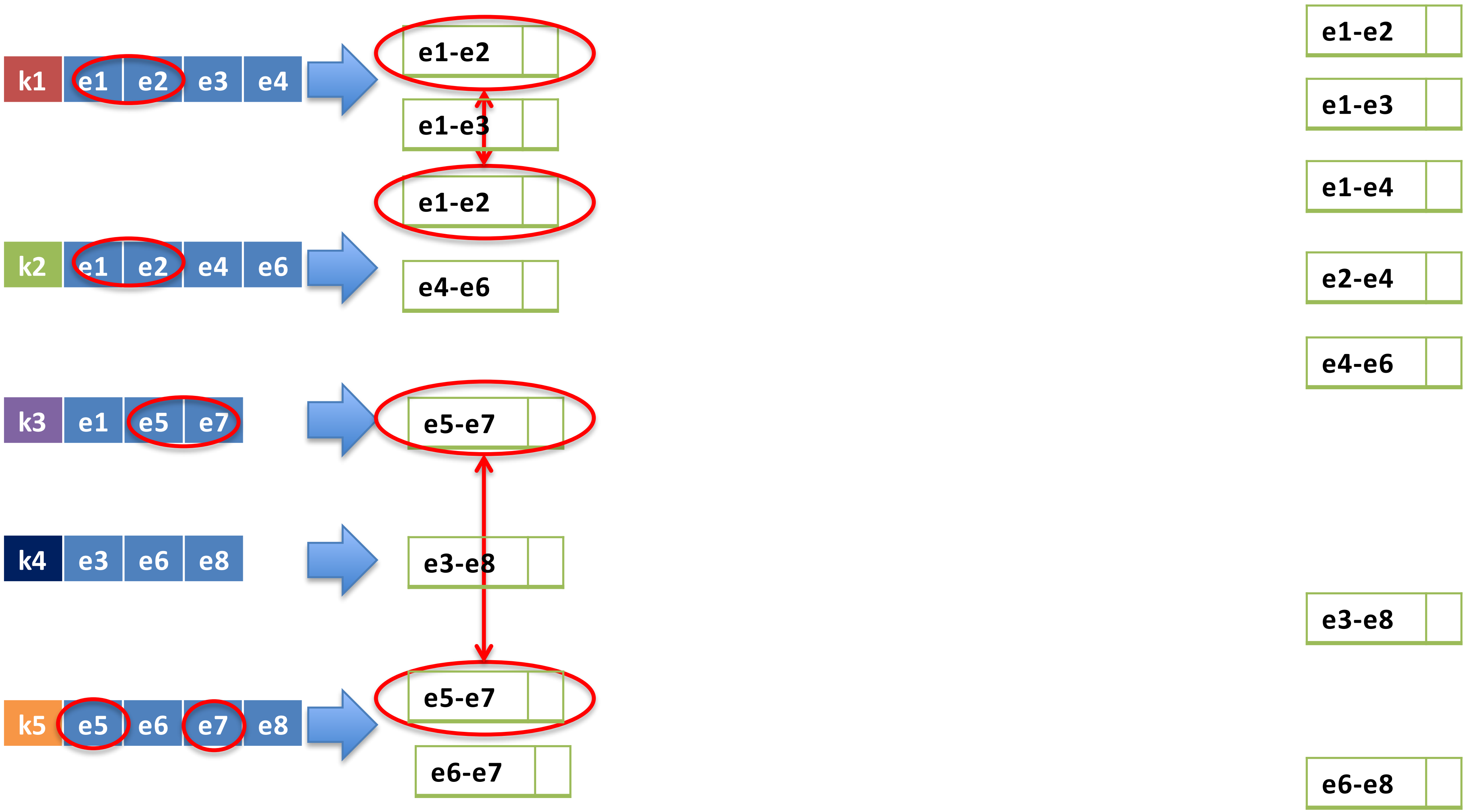
# Map: Append the Subset of Smaller Keys for the Same Description



# Resulting Comparisons

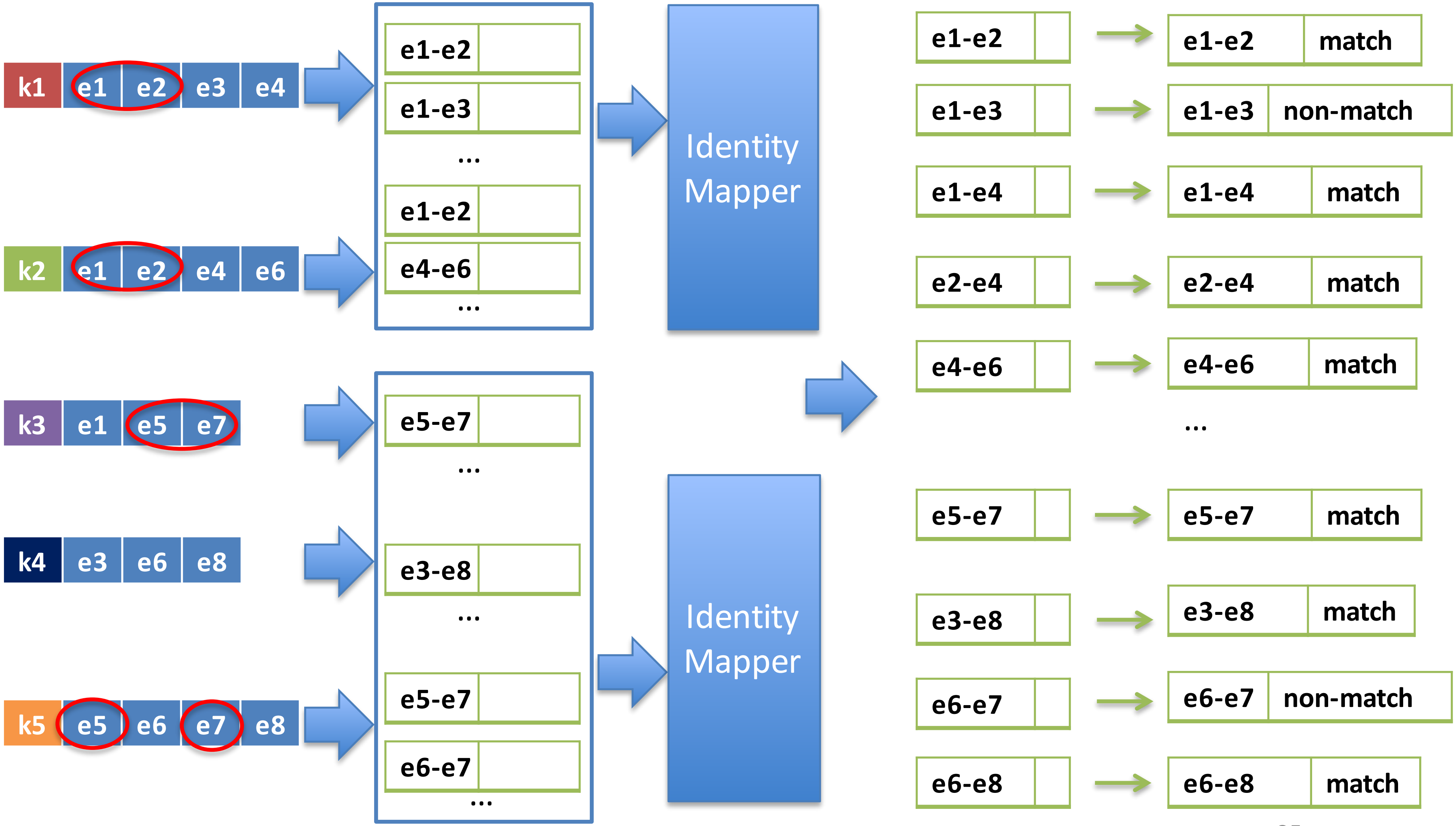


# A More Efficient Approach [Vernica et al. 2010]




Reducer's output...

# A More Efficient Approach [Vernica et al. 2010]



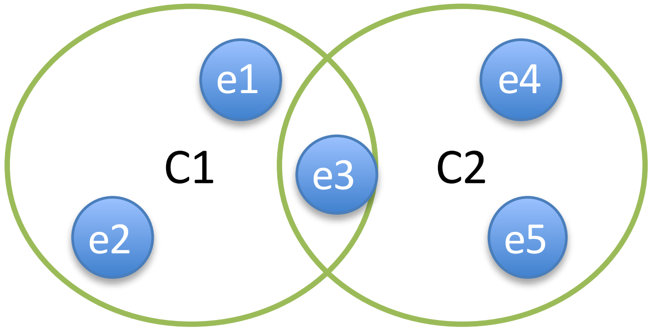

...becomes a new job's input

# Large-Scale Collective Entity Matching
[Rastogi et al. 2011]

Assume that there is a rule *R: Match(e1, e2) => Match(e4, e5)*

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

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




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

reduce: bring all the new evidence for each Ci 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 <u>entity graph</u> constructed from RDF triples having URIs as subject, predicate and object
  - Literals are stored for each entity e as L(e)

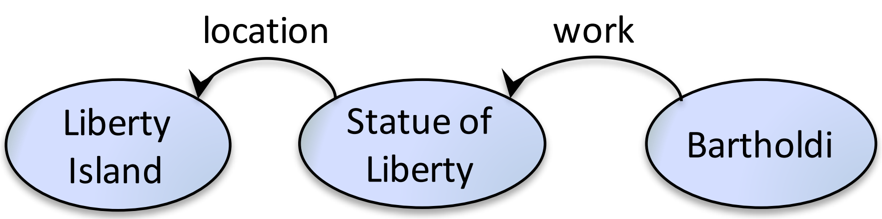
- Matches are identified using <u>two kinds of similarities</u>:
  - String similarity (token-based) of their literal values L(e)
    - Checked once
  - Contextual similarity (based on neighbors in the entity graph)
    - Checked iteratively

# 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(Statue of liberty) =
{(location, Liberty Island, w1),
(*is* work *of*, Bartholdi, w2)}

# Contextual Similarity

The contextual similarity of nodes n and m is:

context_sim(n,m) =

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

where

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

$sim(p_i, p_j)$ 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:

$$sim(n, m) + \beta \cdot context\_sim(C(n), C(m)) - \theta$$

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 β = 1 to perform well*

# LINDA Algorithm

Two square matrices (|E|x|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

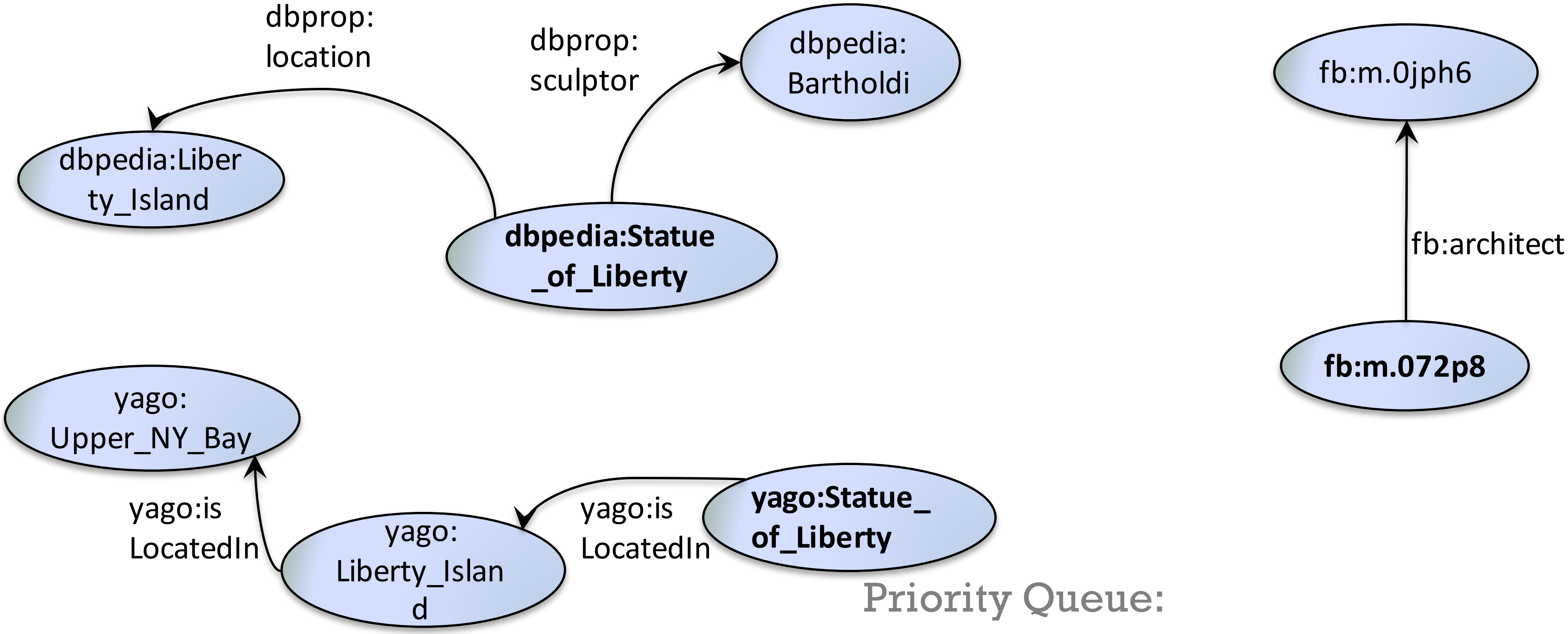
- A node i holds a portion $Q_i$ of the priority queue and the respective part $G_i$ of the graph

<u>Map phase</u>

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

<u>Reduce phase</u>

- 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) |

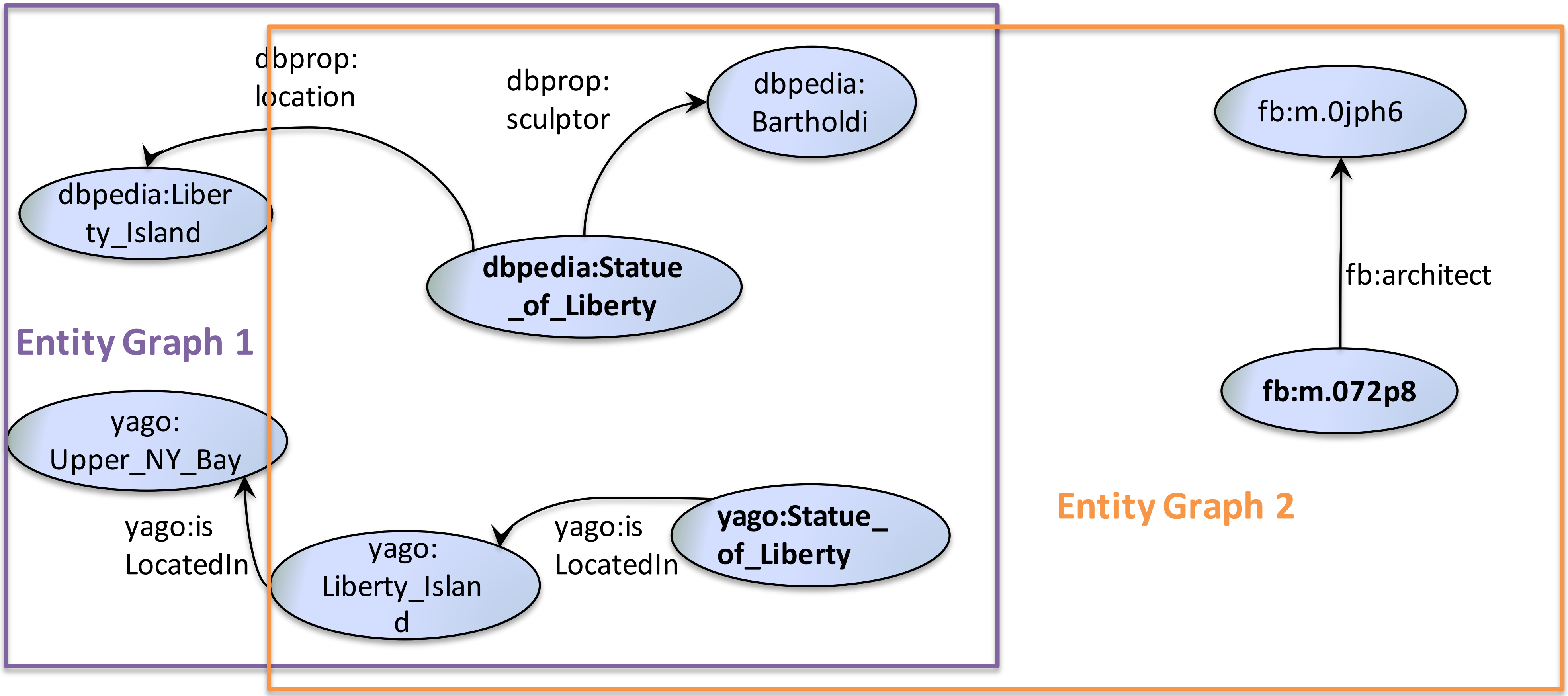
Priority Queue 1 (machine 1):

| |
|---|
| (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



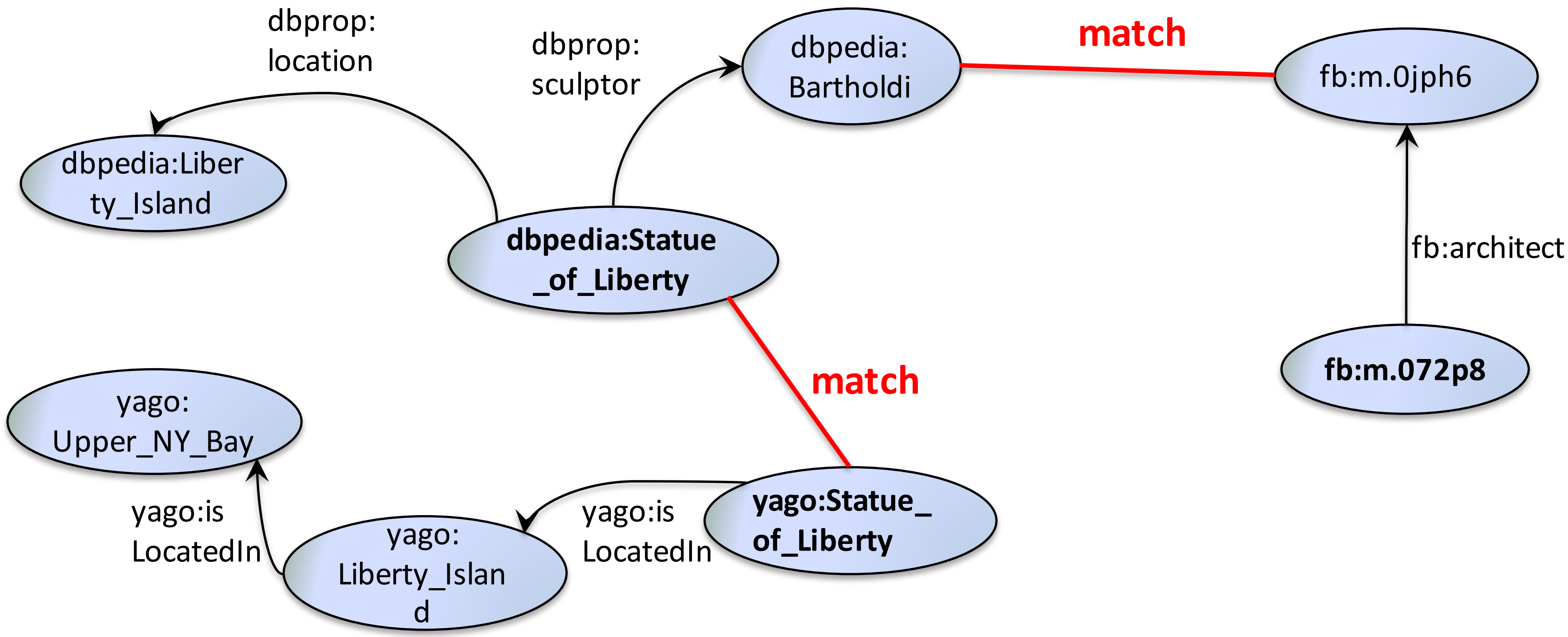

Priority Queue 1 (machine 1):

| |
|---|
| (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:

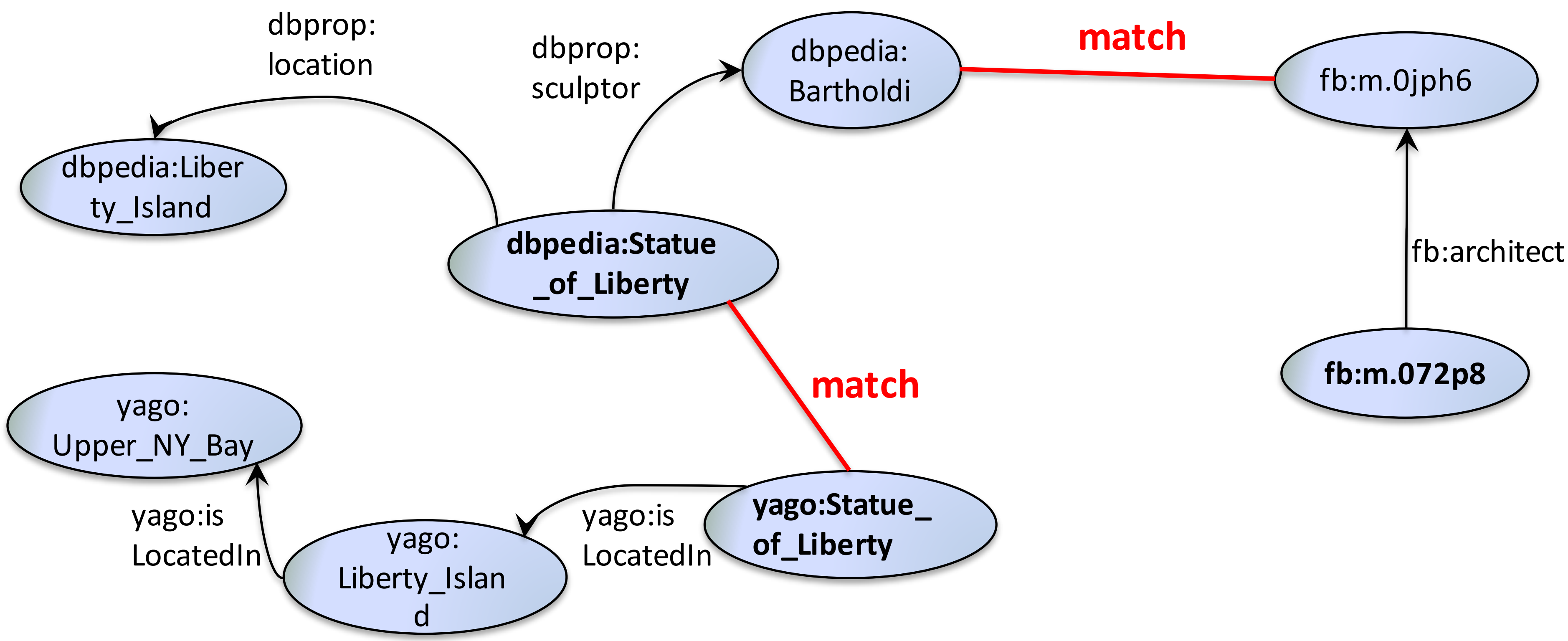
| |
|---|
| (dbpedia:Statue_of_Liberty, yago:Statue_of_Liberty) |
| (dbpedia:Statue_of_Liberty, yago:Liberty_Island) |
| (dbpedia:Liberty_Island,yago:Upper_NY_Bay) |
| (dbpedia:Liberty_Island, yago:Liberty_Island) |
| (dbpedia:Liberty_Island, yago:Statue_of_Liberty) |

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



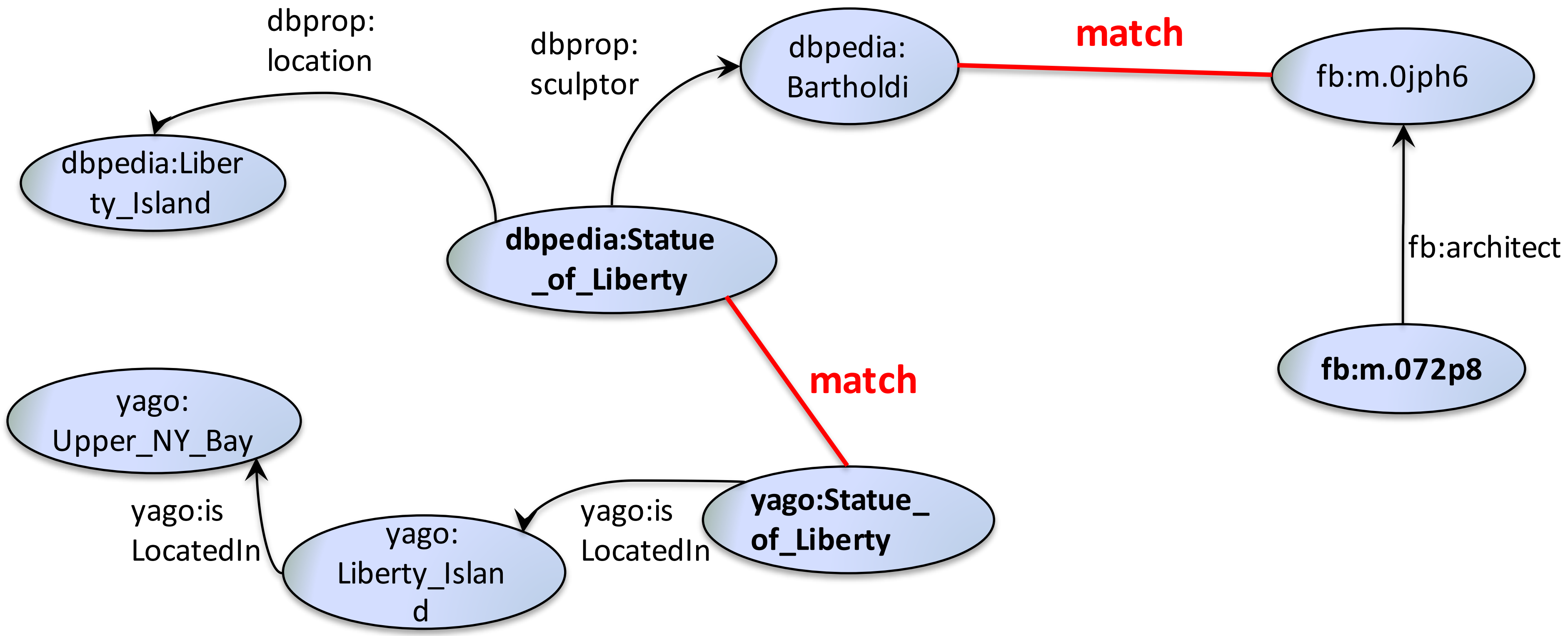

Priority Queue 1:

| |
|---|
| (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



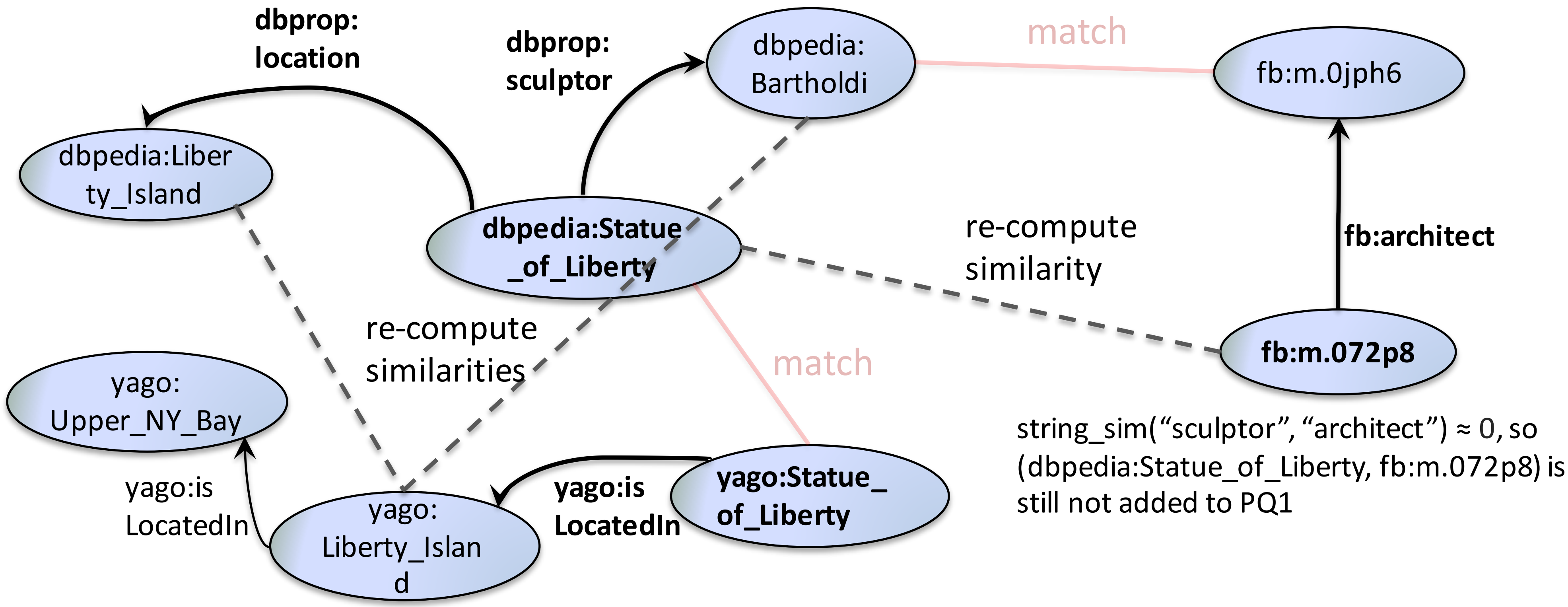

Priority Queue 1:

| |
|---|
| (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:

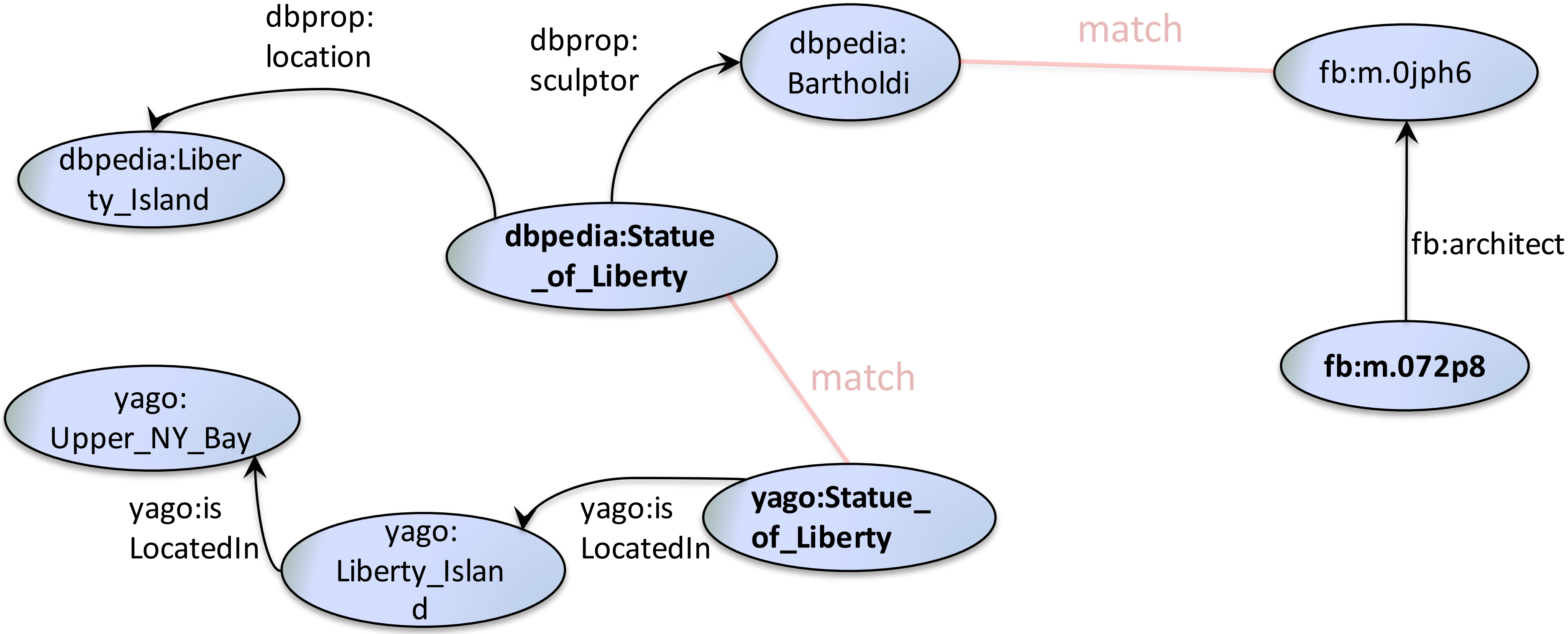
(dbpedia:Liberty_Island,yago:Upper_NY_Bay)

**(dbpedia:Liberty_Island, yago:Liberty_Island)**

Priority Queue 2:

Contextual similarity re-computations

Property names are also taken into account



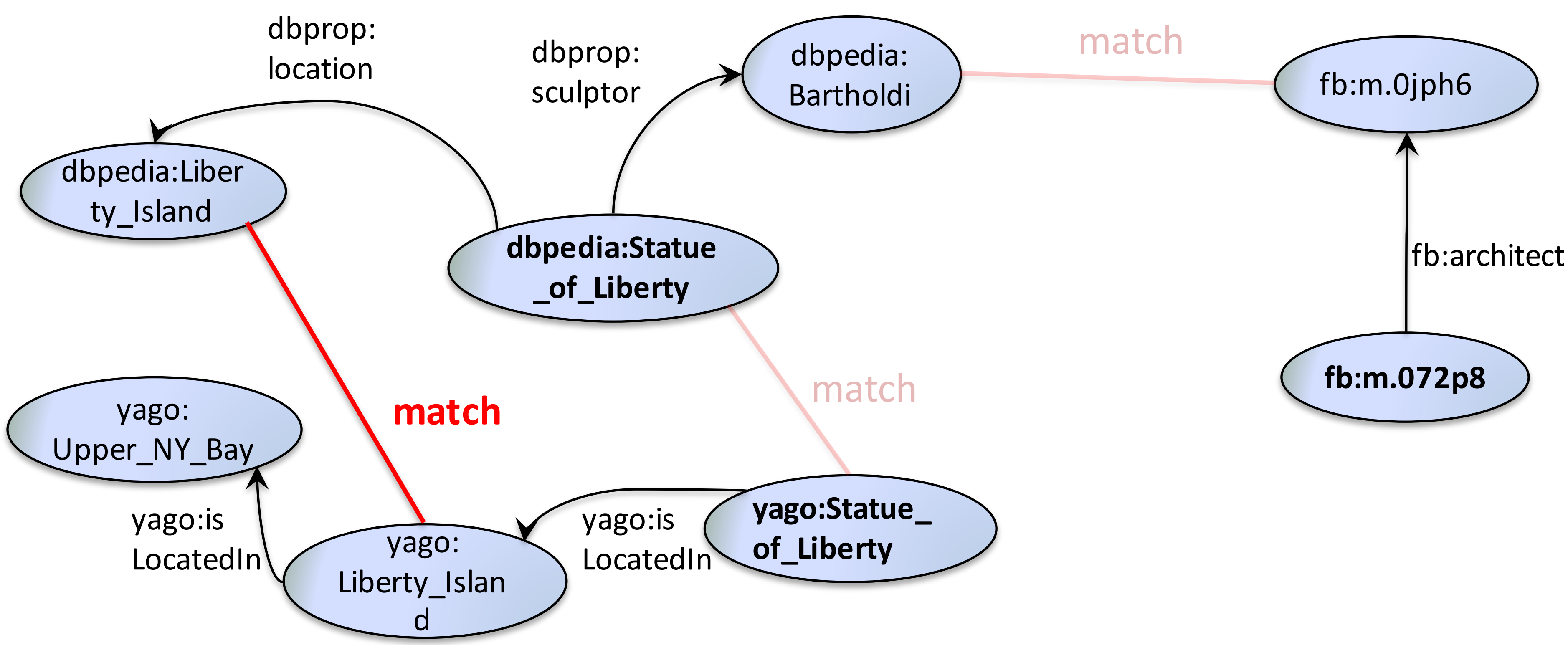

Priority Queue 1:

| |
|---|
| **(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:

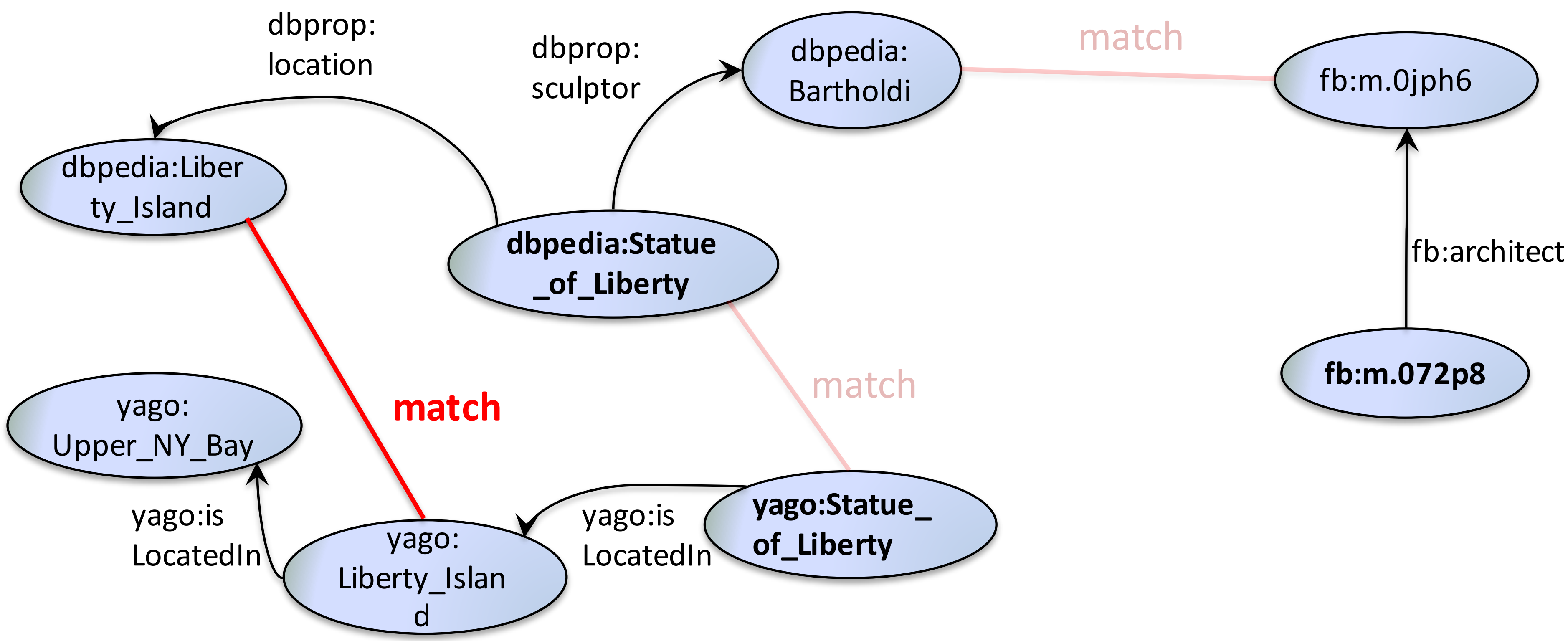
| |
|---|
| **(dbpedia:Liberty_Island, yago:Liberty_Island)** |
| (dbpedia:Liberty_Island,yago:Upper_NY_Bay) |

Priority Queue 2:

| |
|---|
| |

The head of each queue is a match by default

This triggers update messages



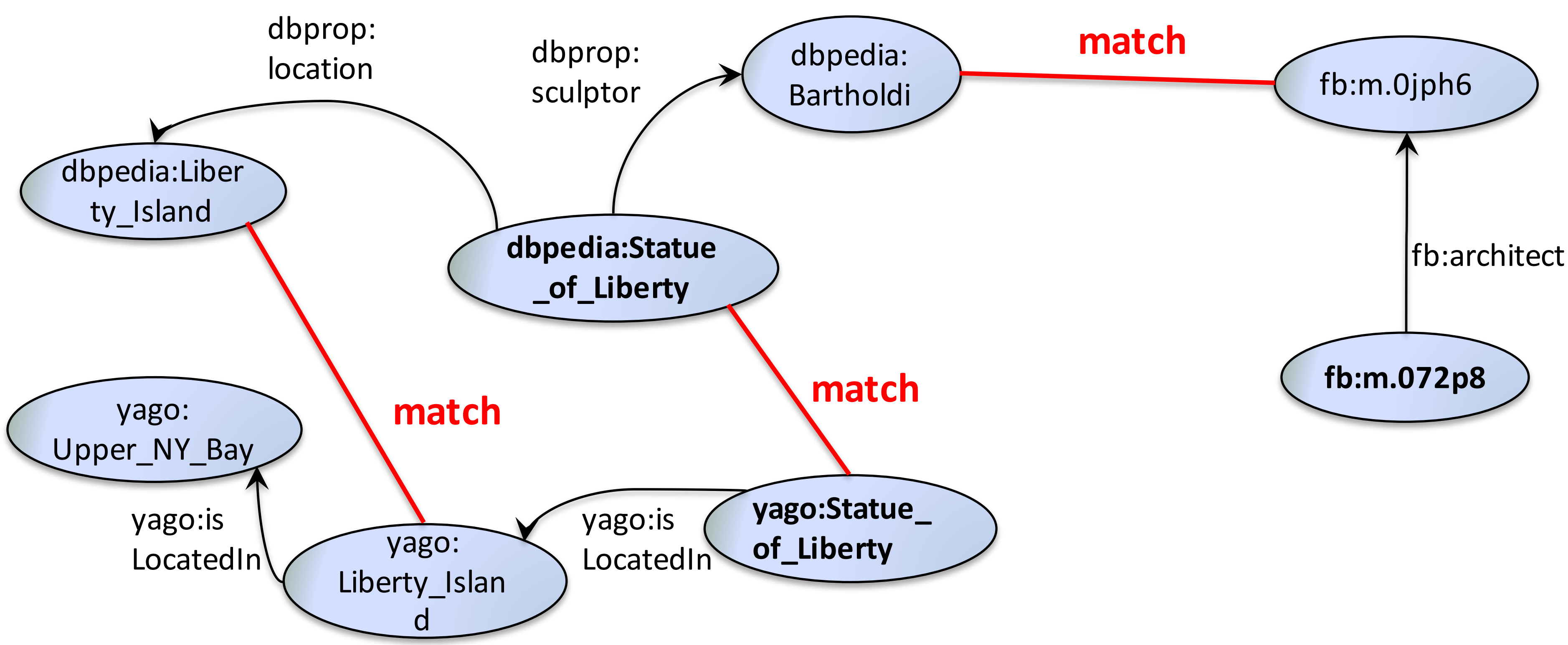

Priority Queue 1:

| |
|---|
| **(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

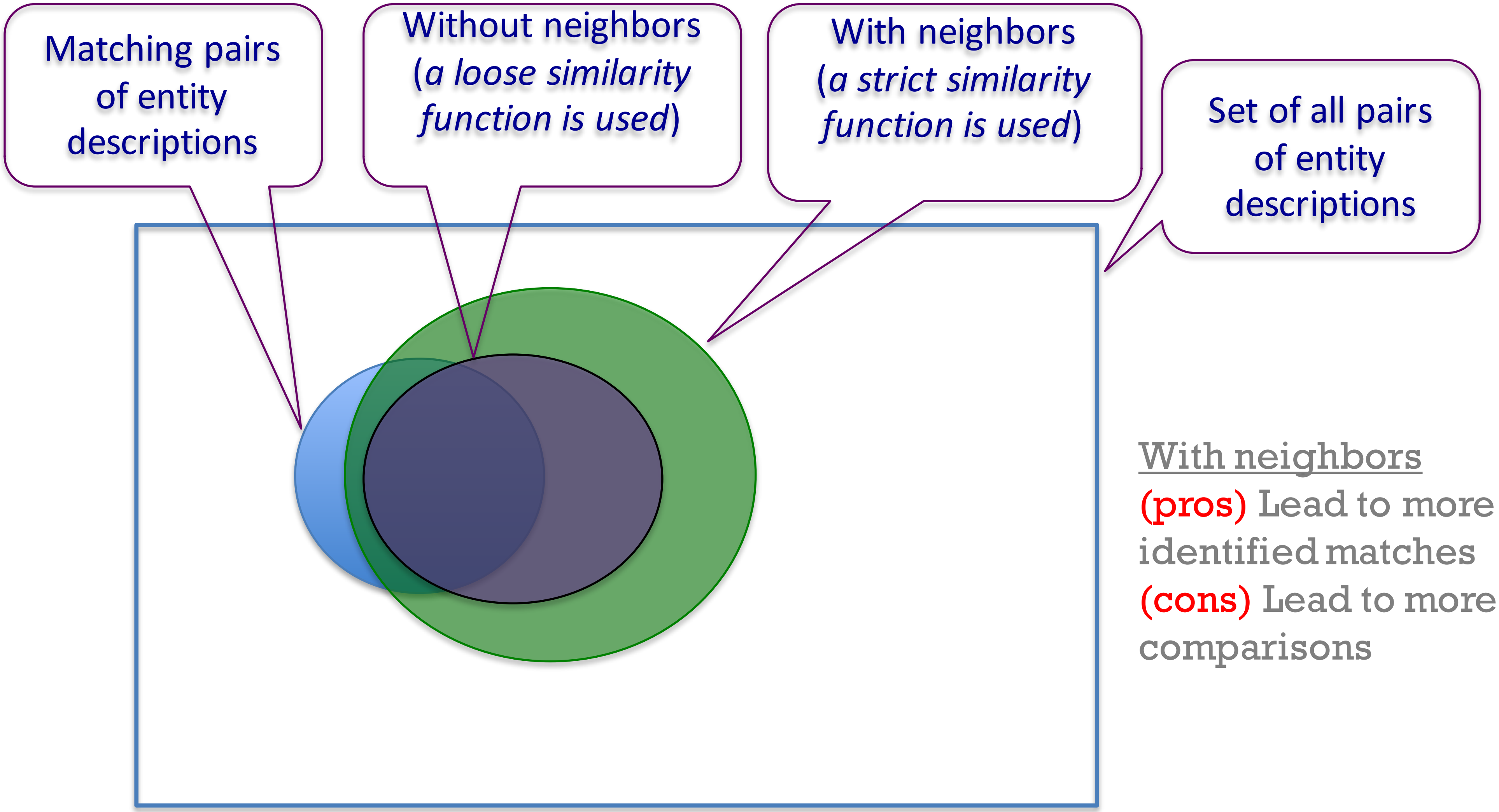



Priority Queue 1:

Priority Queue 2:

Output mappings

# Using Neighbors for Computing Similarities




<u>With neighbors</u>
(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 per

- Can we also use the structural type of th

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)

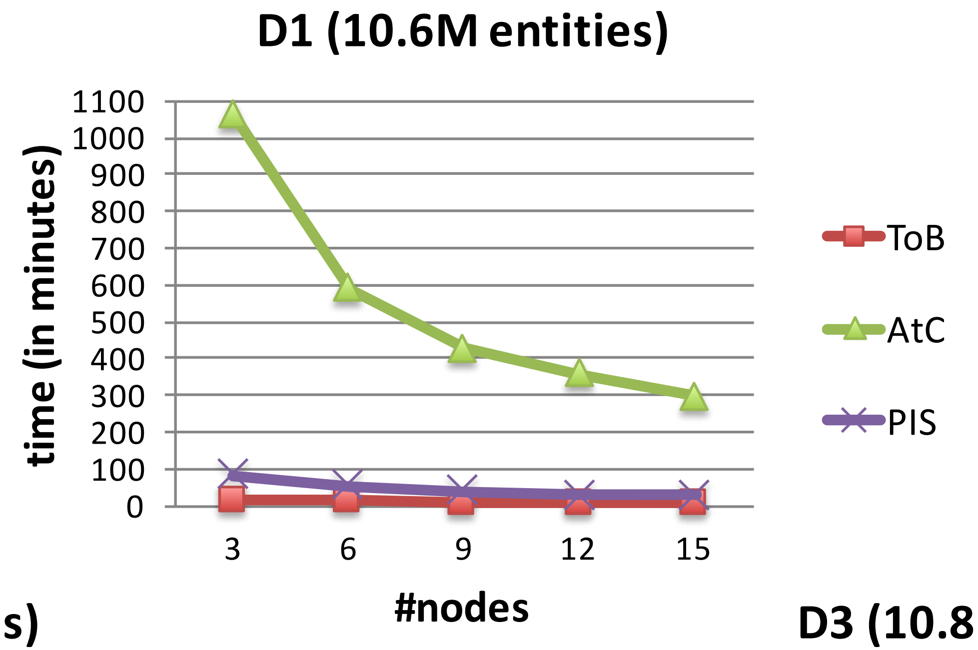
- D1: BTC12 DBpedia (~148M triples) and [DBpedia 3.5] (~32M triples)
- D2: BTC12 DBpedia and BTC12 Rest (~918K triples)
- D3: BTC12 DBpedia and BTC12 Freebase (~29M triples)

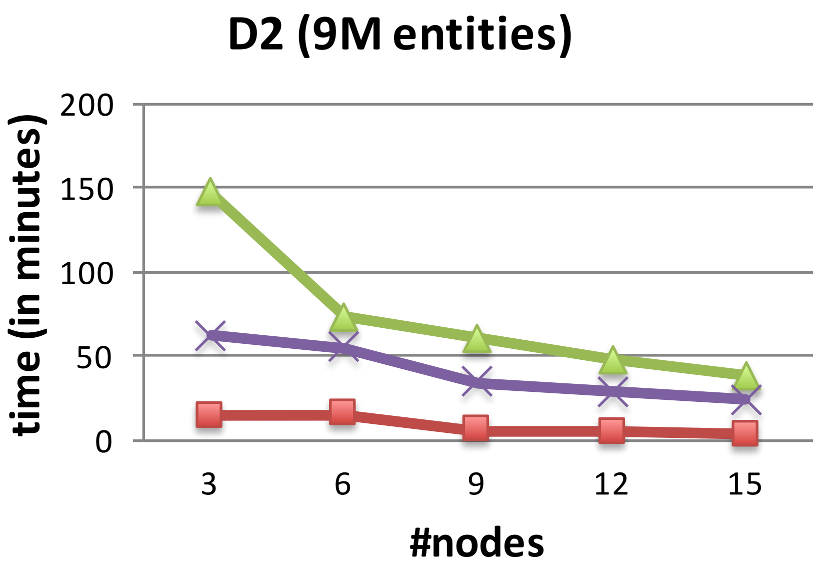
On a cluster of 15 nodes (each with 8 CPUs, 8GB RAM, and 60GB HDD)

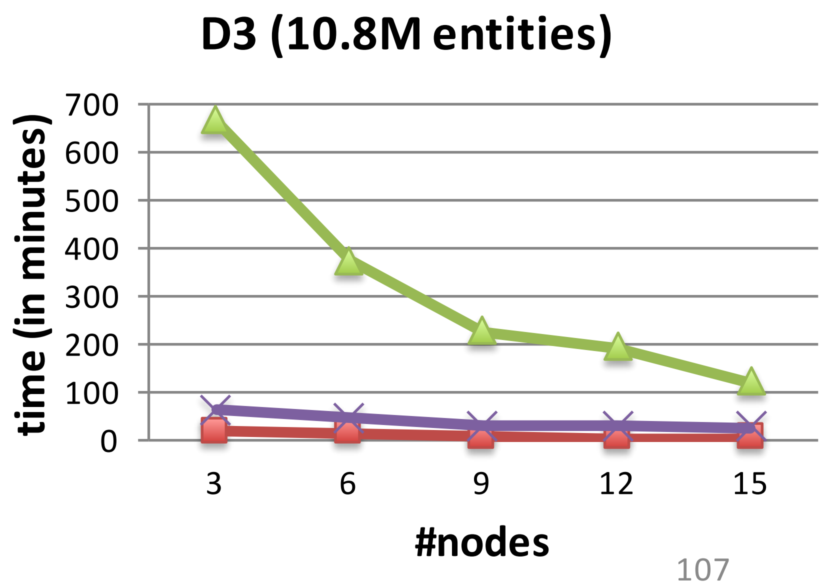
For ground-truth, we used the *owl:sameAs* links provided by DBpedia

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

# Scalability



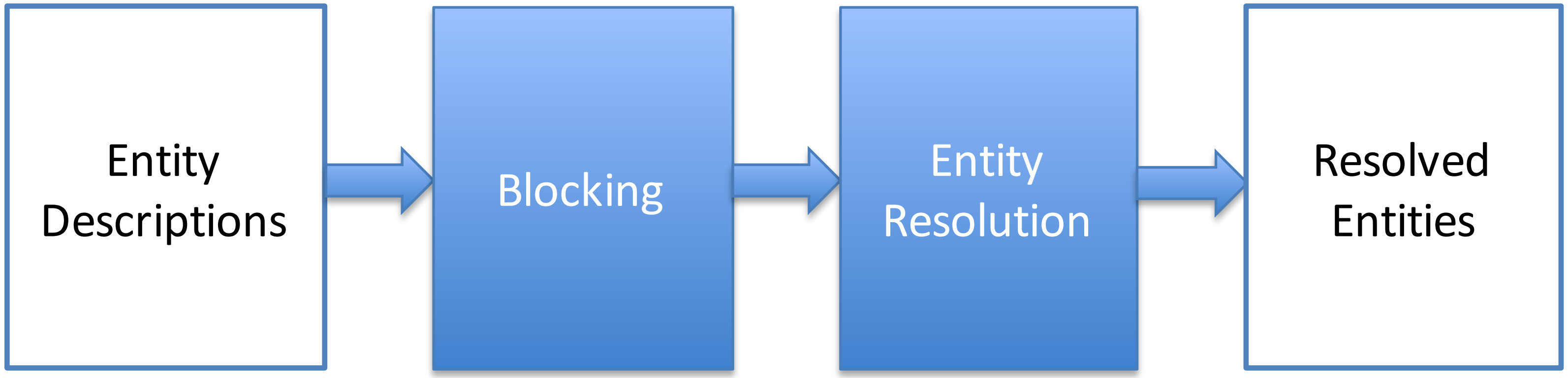
# Quality Results

| | | 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

<u>Ideally</u>:

- 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 Ronal 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. Ronal 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](http://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
- FP7-PEOPLE- 2013-IRSES SemData (Semantic Data Management)

# 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/](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.