updated: 23 April 2018

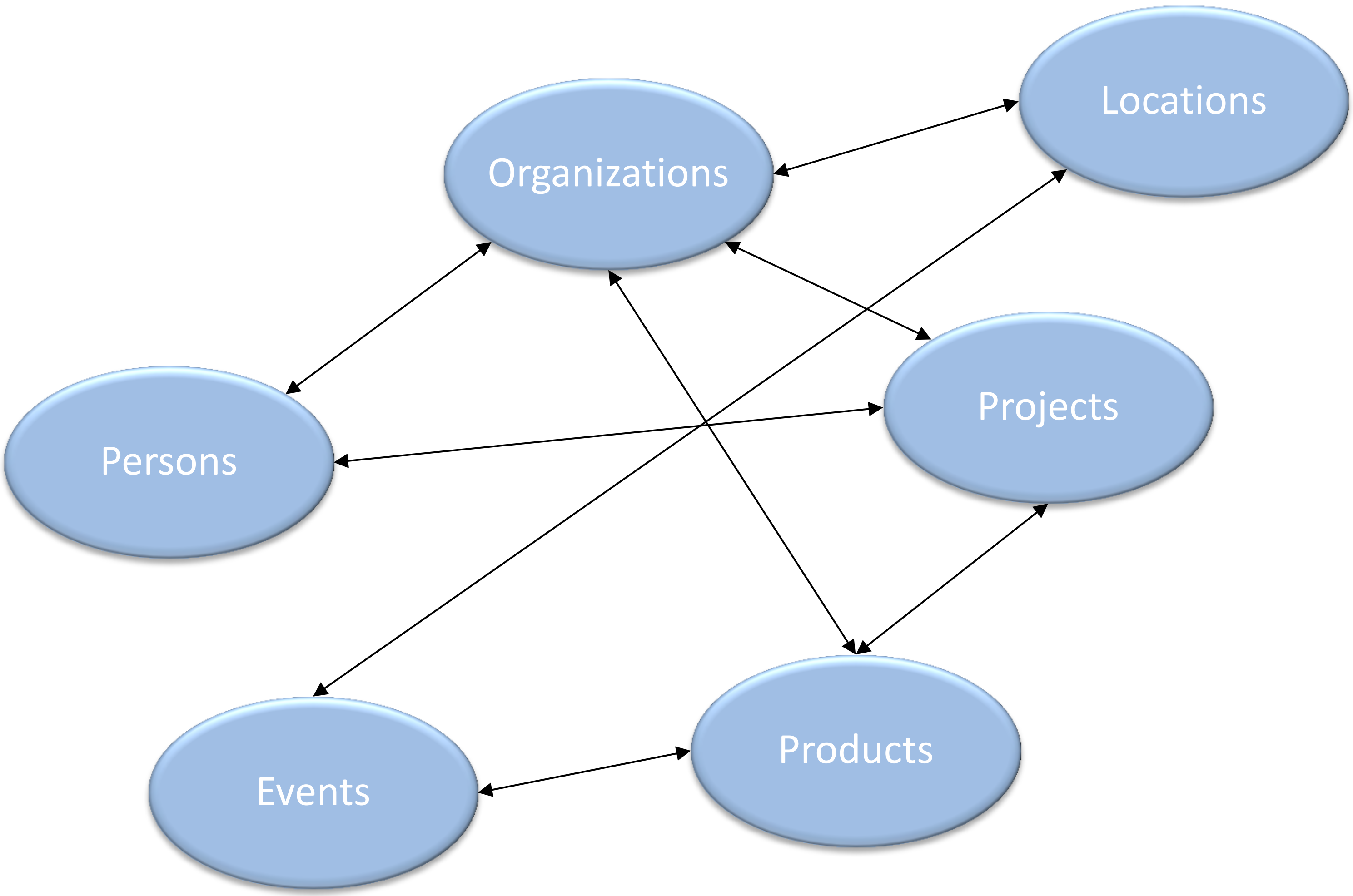
Web-scale, Schema-Agnostic, End-to-End Entity Resolution

George Papadakis
University of Athens
gpapadis@di.uoa.gr

Themis Palpanas
Paris Descartes University
French University Institute
themis@mi.parisdescartes.fr

Entities: an invaluable asset

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



However …

How many names, descriptions or IDs (URIs) are used for the same real-world "entity"?

However …

How many names, descriptions or IDs (URIs) are used for the same real-world “entity”?

London 런던 ལོན་ཊོན། लंडन लंदन લંડન ለንደን ロンドン লন্ডন ลอนดอน இலண்டன் ლონდონი Llundain Londain Londe Londen Londen Londen Londinium London Londona Londonas Londoni Londono Londra Londres Londrez Londyn Lontoo Loundres Luân Đôn Lunden Lundúnir Lunnainn Lunnon لندن لندن لندن لوندون לאנדאן לונדון Λονδίνο Лёндан Лондан Лондон Лондон Лондон Լոնդոն 伦敦 …

However …

How many names, descriptions or IDs (URIs) are used for the same real-world "entity"?

London 런던 ލަންޑަން लंडन लंदन લંડન ለንደን ロンドン লন্ডন ลอนดอน இலண்டன் ლონდონი Llundain Londain Londe Londen Londen Londen Londinium London Londona Londonas Londoni Londono Londra Londres Londrez Londyn Lontoo Loundres Luân Đôn Lunden Lundúnir Lunnainn Lunnon لوندون لندن لندن لندن לונדון לאנדאן Λονδίνο Лёндан Лондан Лондон Лондон Лондон Լոնդոն 伦敦 …

capital of UK, host city of the IV Olympic Games, host city of the XIV Olympic Games, future host of the XXX Olympic Games, city of the Westminster Abbey, city of the London Eye, the city described by Charles Dickens in his novels, …

However …

How many names, descriptions or IDs (URIs) are used for the same real-world "entity"?

London 런던 ලන්ඩන් लंडन लंदन લંડન ለንደን ロンドン লন্ডন ลอนดอน இலண்டன் ლონდონი Llundain Londain Londe Londen Londen Londen Londinium London Londona Londonas Londoni Londono Londra Londres Londrez Londyn Lontoo Loundres Luân Đôn Lunden Lundúnir Lunnainn Lunnon لندن لندن لندن لوندون לאנדאן לונדון Λονδίνο Лёндан Лондан Лондон Лондон Лондон Լոնդոն 伦敦 …

capital of UK, host city of the IV Olympic Games, host city of the XIV Olympic Games, future host of the XXX Olympic Games, city of the Westminster Abbey, city of the London Eye, the city described by Charles Dickens in his novels, …

http://sws.geonames.org/2643743/
http://en.wikipedia.org/wiki/London
http://dbpedia.org/resource/Category:London
…

… or …

How many "entities" have the same name?

- London, KY
- London, Laurel, KY
- London, OH
- London, Madison, OH
- London, AR
- London, Pope, AR
- London, TX
- London, Kimble, TX
- London, MO
- London, MO
- London, London, MI
- London, London, Monroe, MI
- London, Uninc Conecuh County, AL
- London, Uninc Conecuh County, Conecuh, AL
- London, Uninc Shelby County, IN
- London, Uninc Shelby County, Shelby, IN
- London, Deerfield, WI
- London, Deerfield, Dane, WI
- London, Uninc Freeborn County, MN
- ...

… or …

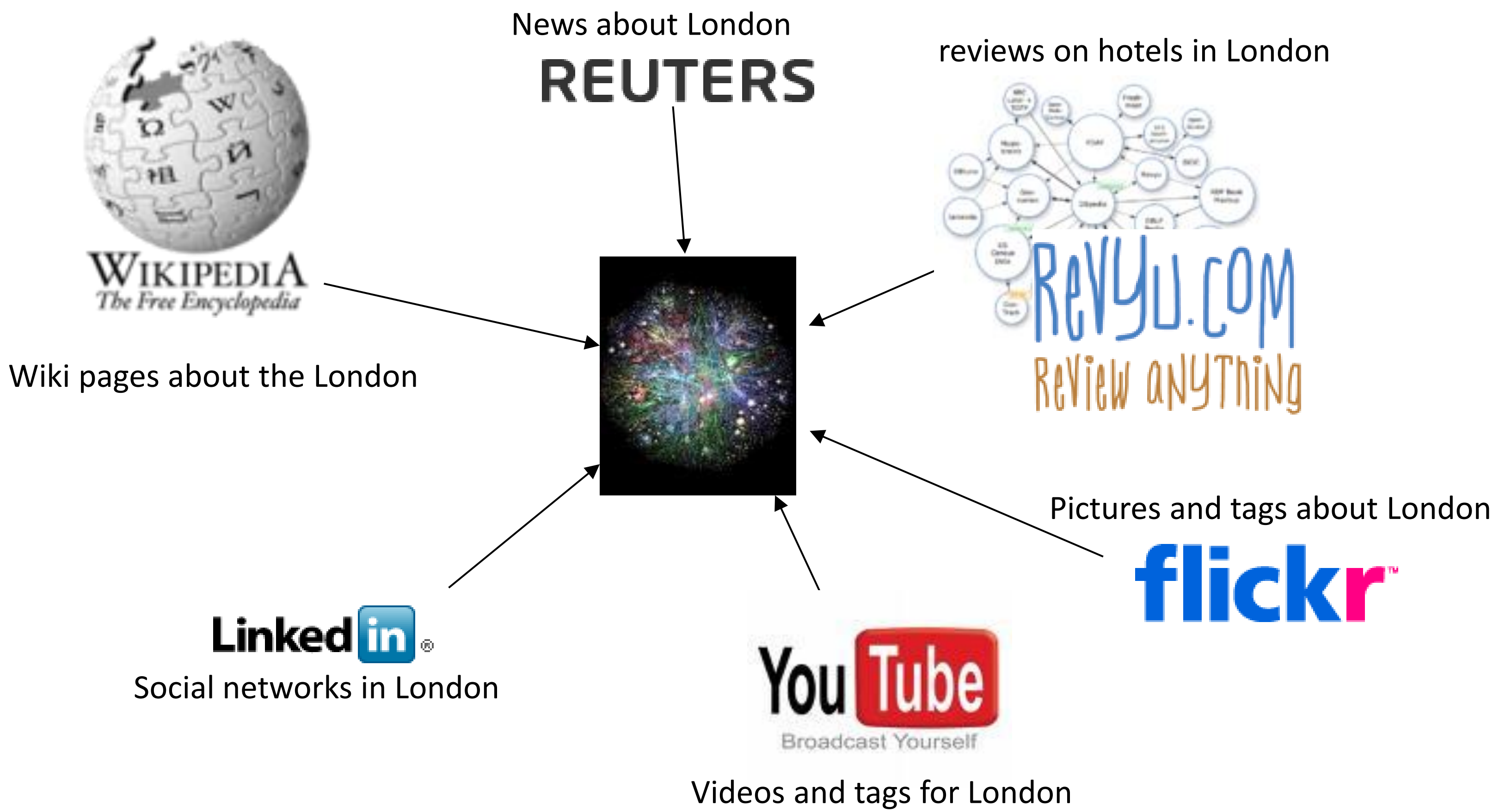
How many "entities" have the same name?

- London, KY
- London, Laurel, KY
- London, OH
- London, Madison, OH
- London, AR
- London, Pope, AR
- London, TX
- London, Kimble, TX
- London, MO
- London, MO
- London, London, MI
- London, London, Monroe, MI
- London, Uninc Conecuh County, AL
- London, Uninc Conecuh County, Conecuh, AL
- London, Uninc Shelby County, IN
- London, Uninc Shelby County, Shelby, IN
- London, Deerfield, WI
- London, Deerfield, Dane, WI
- London, Uninc Freeborn County, MN
- ...

- London, Jack
  2612 Almes Dr
  Montgomery, AL
  (334) 272-7005
- London, Jack R
  2511 Winchester Rd
  Montgomery, AL 36106-3327
  (334) 272-7005
- London, Jack
  1222 Whitetail Trl
  Van Buren, AR 72956-7368
  (479) 474-4136
- London, Jack
  7400 Vista Del Mar Ave
  La Jolla, CA 92037-4954
  (858) 456-1850
- ...

Content Providers

How many content types / applications provide valuable information about each of these "entities"?



Preliminaries on Entity Resolution

Entity Resolution [Dong et al., Book 2015] [Elmagarmid et al., TKDE 2007] :

identifies and aggregates the different entity profiles/records that actually describe the same real-world object.

Useful because:

- improves data quality and integrity
- fosters re-use of existing data sources

Application areas:

Linked Data, Social Networks, census data, price comparison portals

Types of Entity Resolution

The input of ER consists of entity collections that can be of two types [Christen, TKDE 2011]:

- clean, which are duplicate-free

  e.g., DBLP, ACM Digital Library, Wikipedia, Freebase

- dirty, which contain duplicate entity profiles in themselves

  e.g., Google Scholar, Citeseer^X

Types of Entity Resolution

The input of ER consists of entity collections that can be of two types [Christen, TKDE 2011]:

- clean, which are duplicate-free

  e.g., DBLP, ACM Digital Library, Wikipedia, Freebase
- dirty, which contain duplicate entity profiles in themselves

  e.g., Google Scholar, Citeseer^X

Based on the quality of input, we distinguish ER into 3 sub-tasks:

- **Clean-Clean ER** (a.k.a. ***Record Linkage*** in databases)
- Dirty-Clean ER
- Dirty-Dirty ER

Equivalent to **Dirty ER** (a.k.a. ***Deduplication*** in databases)

Challenges for ER over Web Data

- Volume
  - Millions of entities
  - Billions of name-value pairs describing them
  - LOD Cloud*: >$5{,}5 \cdot 10^7$ entities, ~$1{,}5 \cdot 10^{11}$ triples
- Variety
  - Semi-structured data → unprecedented levels of heterogeneity
  - Numerous entity types & vocabularies
  - LOD Cloud*: ~50,000 predicates, ~12,000 vocabularies
- Velocity
  - New DBPedia version every ~6 months

*http://stats.lod2.eu:

Computational cost

ER is an inherently quadratic problem (i.e., $O(n^2)$):
every entity has to be compared with all others

ER does not scale well to large entity collections (e.g., Web Data).

Computational cost

ER is an inherently quadratic problem (i.e., $O(n^2)$):
every entity has to be compared with all others

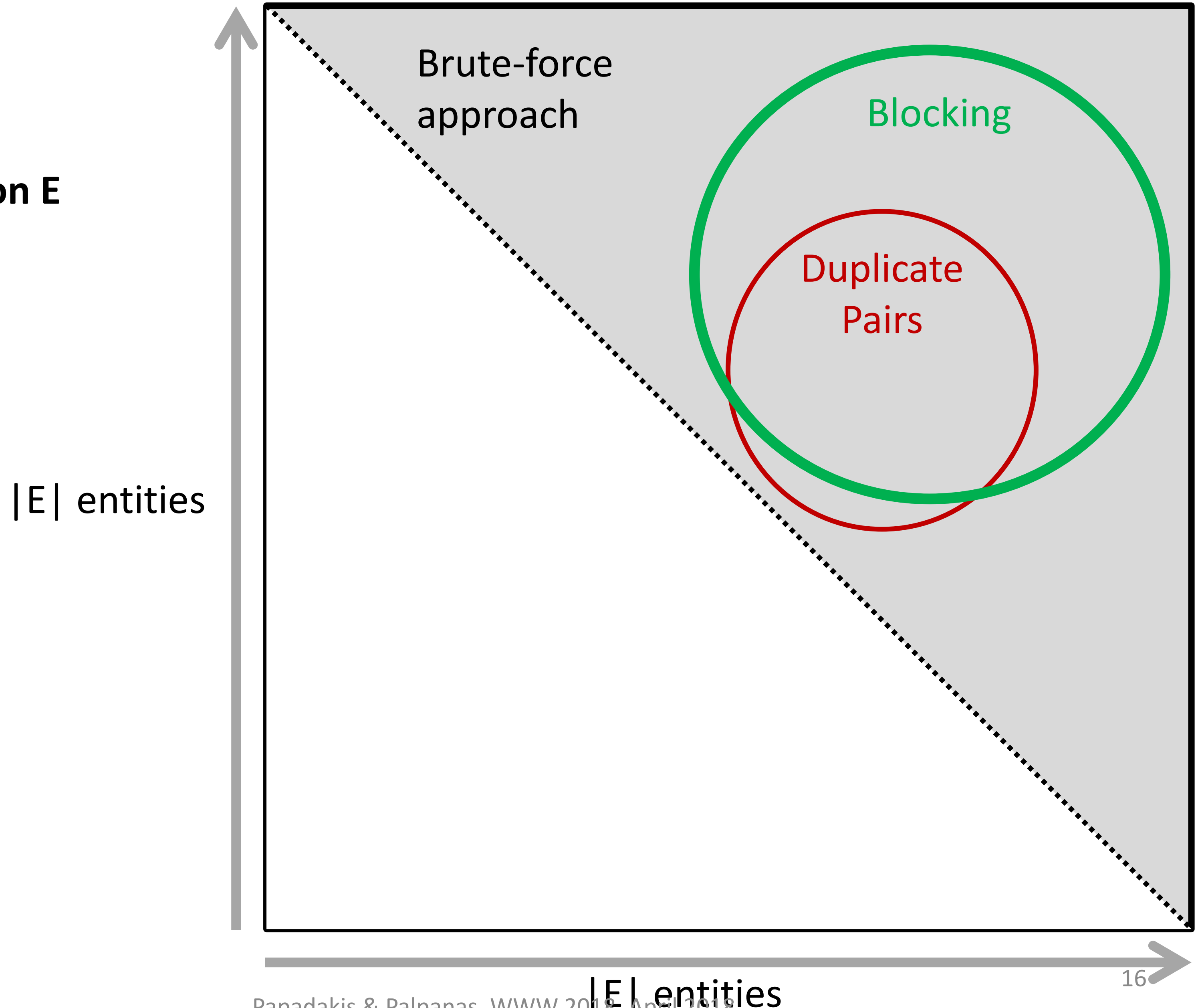
ER does not scale well to large entity collections (e.g., Web Data)

Solution: Blocking

- group similar entities into blocks
- execute comparisons only inside each block
  - complexity is now quadratic to the size of the block (much smaller than dataset size!)

Computational cost

Input:
Entity Collection E



Example of Computational cost

DBPedia 3.0rc ↔ DBPedia 3.4

1.2 million entities ↔ 2.2 million entities

Entity matching: Jaccard similarity of all tokens
Cost per comparison: 0.045 milliseconds (average of 0.1 billion comparisons)

Brute-force approach

Comparisons: $2.58 \cdot 10^{12}$
Recall: 100%
Running time: 1,344 days → **3.7 years**

Optimized Token Blocking Workflow

Overhead time: 4 hours
Comparisons: $8.95 \cdot 10^{6}$
Recall: 99%
Total Running time: **10 hours**

Example of Computational cost

DBPedia 3.0rc ↔ DBPedia 3.4

1.2 million entities ↔ 2.2 million entities

Entity matching: Jaccard similarity of all tokens
Cost per comparison: 0.045 milliseconds (average of 0.1 billion comparisons)

Brute-force approach

Comparisons: $2.58 \cdot 10^{12}$
Recall: 100%
Running time: 1,344 days → **3.7 years**

Optimized Token Blocking Workflow

Overhead time: 4 hours
Comparisons: $8.95 \cdot 10^{6}$
Recall: 99%
Total Running time: **10 hours**

Scalable End-to-end ER workflow

Step 1	Step 2	Step 3	Step 4
Block Building →	**Block Processing** →	**Entity Matching** →	**Entity Clustering**
Cluster together similar entities	Refine blocks to increase precision at no significant cost in recall	Compare the candidate matches	Partition the compared profiles into real-world entities

Outline

1. Introduction to Blocking
2. Blocking Methods for Relational Data
3. Blocking Methods for Web Data
4. Block Processing Techniques
5. Meta-blocking
6. Entity Matching
7. Entity Clustering
8. Massive Parallelization Methods
9. Progressive Entity Resolution
10. Challenges
11. JedAI Toolkit
12. Conclusions

Part 1:

Introduction to Blocking

Fundamental Assumptions

1. Every entity profile consists of a *uniquely identified* set of name-value pairs.

2. Every entity profile corresponds to a single real-world object.

3. Two matching profiles are *detected* as long as they co-occur in at least one block → **entity matching** is an orthogonal problem.

4. Focus on string values.

General Principles

1. Represent each entity by *one or more* blocking keys.
2. Place into blocks all entities having the *same* or *similar* blocking key.

Measures for assessing block quality [Christen, TKDE 2011]:

– Pairs Completeness: $PC = \frac{detected\ matches}{existing\ matches}$ (optimistic recall)

– Pairs Quality: $PQ = \frac{detected\ matches}{executed\ comparisons}$ (pessimistic precision)

Trade-off!

Problem Definition

Given one dirty (Dirty ER), or two clean (Clean-Clean ER) entity collections, cluster their profiles into blocks and process them so that both *Pairs Completeness* (**PC**) and *Pairs Quality* (**PQ**) are maximized.

caution:

- Emphasis on Pairs Completeness (PC).
  - if two entities are matching then they should coincide at some block

Blocking Techniques Taxonomy

1. Performance-wise
   - Exact methods
   - Approximate methods
2. Functionality-wise
   - Supervised methods
   - Unsupervised methods
3. Blocks-wise
   - Disjoint blocks
   - Overlapping blocks
     – Redundancy-neutral
     – Redundancy-positive
     – Redundancy-negative
4. Signature-wise
   - Schema-based
   - Schema-agnostic

Performance-wise Categorization

1. **Exact Blocking Methods**
   – Maximize PQ for PC = 100%
   – **Closed**-world assumption
   – E.g., for bibliographical records , **s ≡ t** if:
     JaccardSimilarity(s.title, t.title) > 0.80 AND
     EditDistance(s.venue, t.venue) < 3
   – Existing methods:
     - **Silk** → filtering technique for edit distance
     - **LIMES** → triangle inequality for similarity metrics
2. **Approximate Blocking Methods**
   – PC < 100% → high PQ
   – **Open**-world assumption

Performance-wise Categorization

1. **Exact Blocking Methods**
   – Maximize PQ for PC = 100%
   – **Closed**-world assumption
   – E.g., for bibliographical records , **s ≡ t** if:

     JaccardSimilarity(s.title, t.title) > 0.80 AND

     EditDistance(s.venue, t.venue) < 3
   – Existing methods:
     - **Silk** → filtering technique for edit distance
     - **LIMES** → triangle inequality for similarity metrics
2. **Approximate Blocking Methods**
   – PC < 100% → high PQ
   – **Open**-world assumption

our focus

Functionality-wise Categorization

1. **Supervised** Methods

- Goal: learn the best blocking keys from a training set
- Approach: identify best combination of attribute names and transformations
- E.g., CBLOCK [Sarma et. al, CIKM 2012],
  [Bilenko et. al., ICDM 2006], [Michelson et. al., AAAI 2006]
- Drawbacks:
  - labelled data
  - domain-dependent

2. **Unsupervised** Methods

- Generic, popular methods

Functionality-wise Categorization

1. **Supervised** Methods

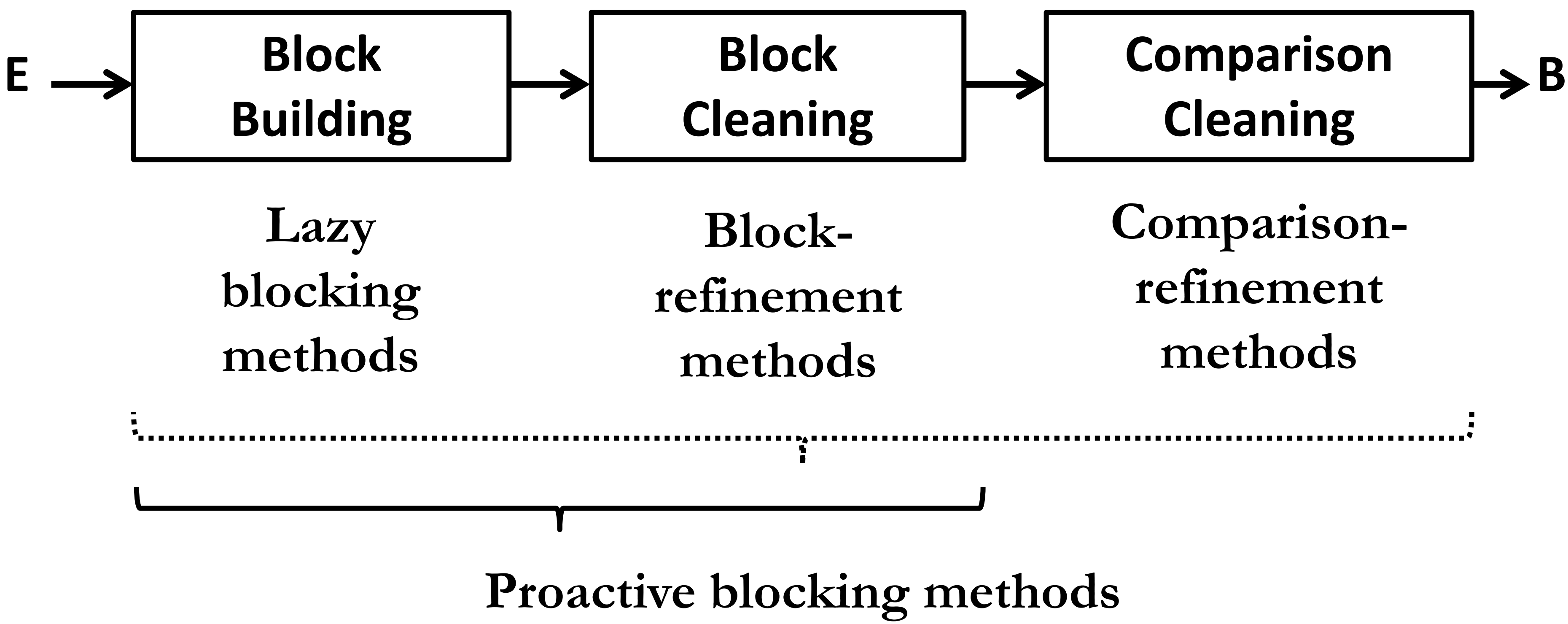
- Goal: learn the best blocking keys from a training set
- Approach: identify best combination of attribute names and transformations
- E.g., CBLOCK [Sarma et. al, CIKM 2012], [Bilenko et. al., ICDM 2006], [Michelson et. al., AAAI 2006]
- Drawbacks:
  - labelled data
  - domain-dependent

2. **Unsupervised** Methods

- Generic, popular methods

our focus

Blocking Workflow [Papadakis et. al., VLDB 2016]



Blocks- and Signature-wise Categorization of Block Building Methods

	Disjoint Blocks	Overlapping Blocks		
		Redundancy-negative	**Redundancy-neutral**	**Redundancy-positive**
Schema-based	Standard Blocking	(Extended) Canopy Clustering	1. (Extended) Sorted Neighborhood 2. MFIBlocks	1. (Extended) Q-grams Blocking 2. (Extended) Suffix Arrays
Schema-agnostic	-	-	-	1. Token Blocking 2. Agnostic Clustering 3. TYPiMatch 4. URI Semantics Blocking

Block Processing Methods

[Papadakis et. al., VLDB 2016]

Mostly for redundancy-positive block building methods.

Block Cleaning

- Block-level
  - constraints on block characteristics
- Entity-level
  - constraints on entity characteristics

Comparison Cleaning

- Redundant comparisons
  - repeated across different blocks
- Superfluous comparisons
  - Involve non-matching entities

Part 2:

Block Building for Relational Data

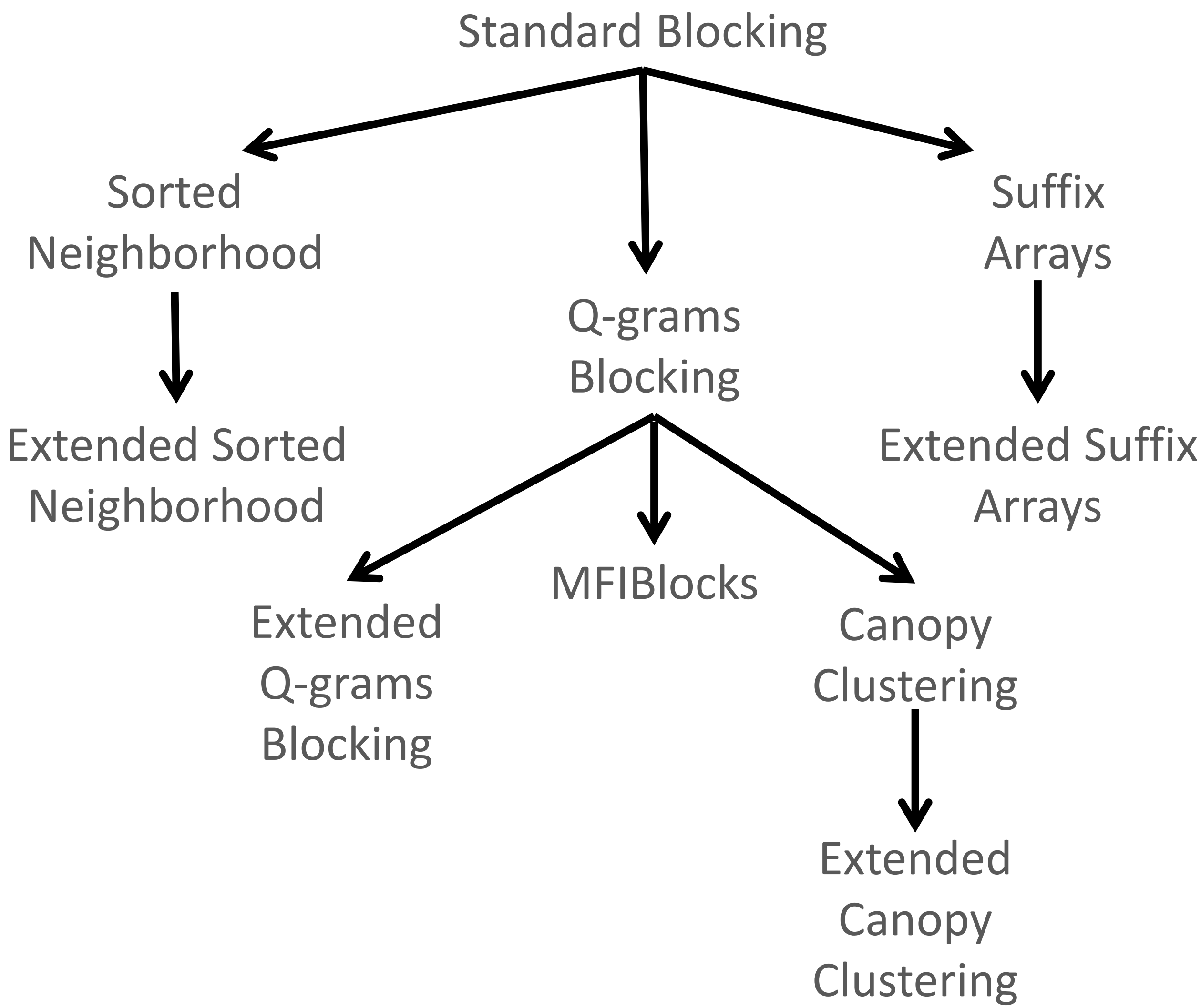
General Principles

Mostly schema-based techniques.

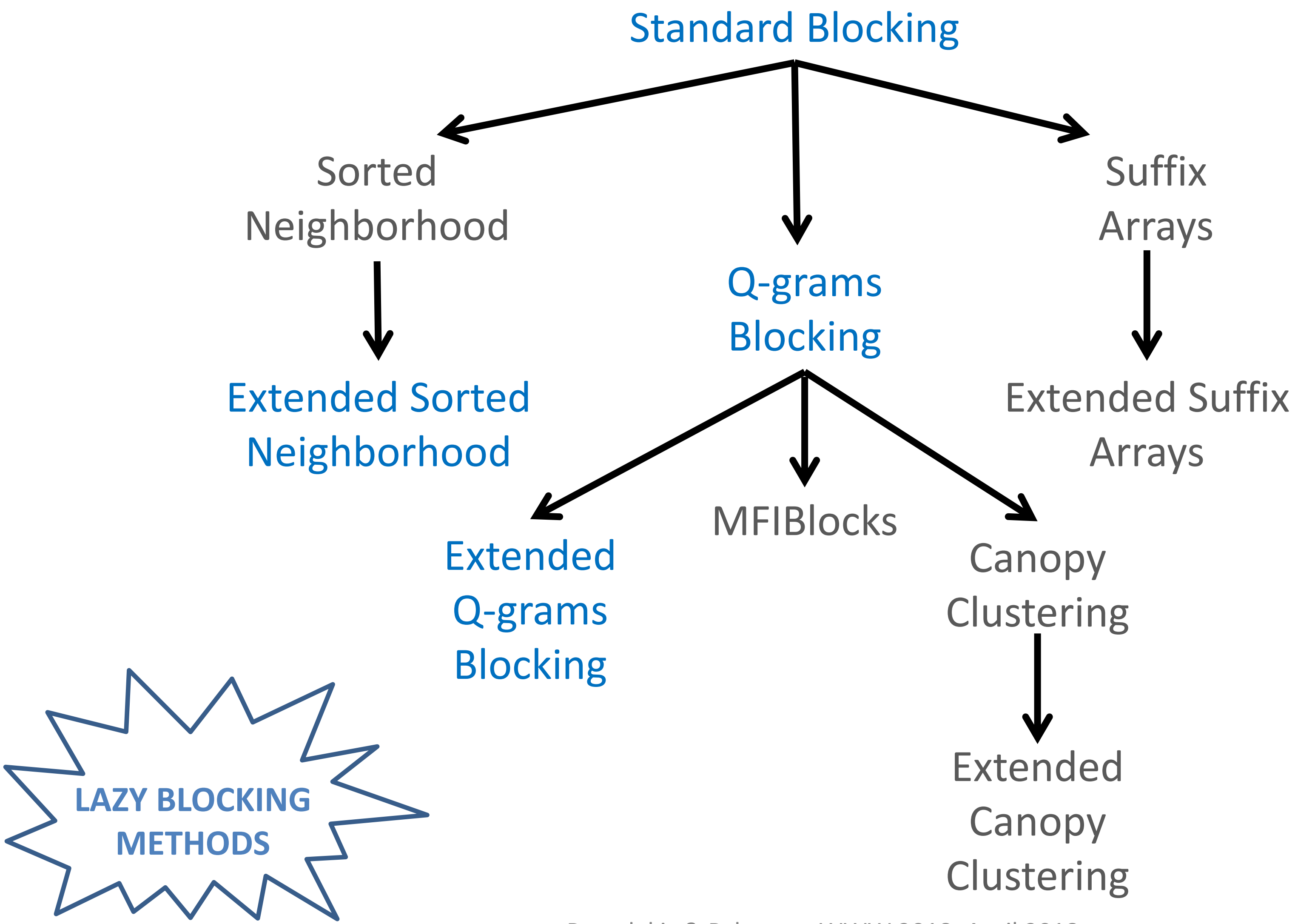
Rely on two assumptions:

1. A-priori known schema → no noise in attribute names.
2. For each attribute name we know some metadata:
   - level of noise (e.g., spelling mistakes, false or missing values)
   - distinctiveness of values

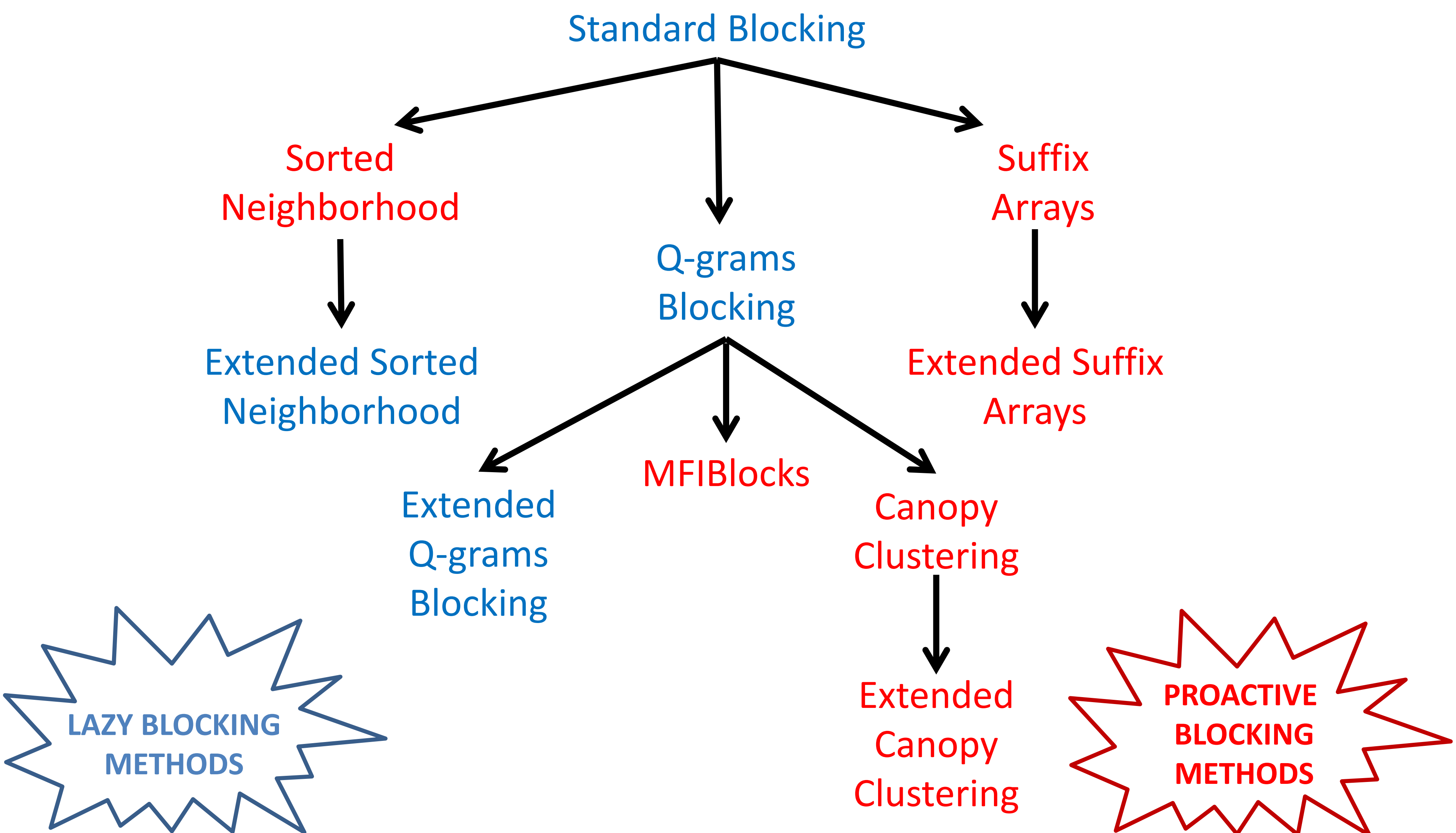
Overview of Schema-based Methods



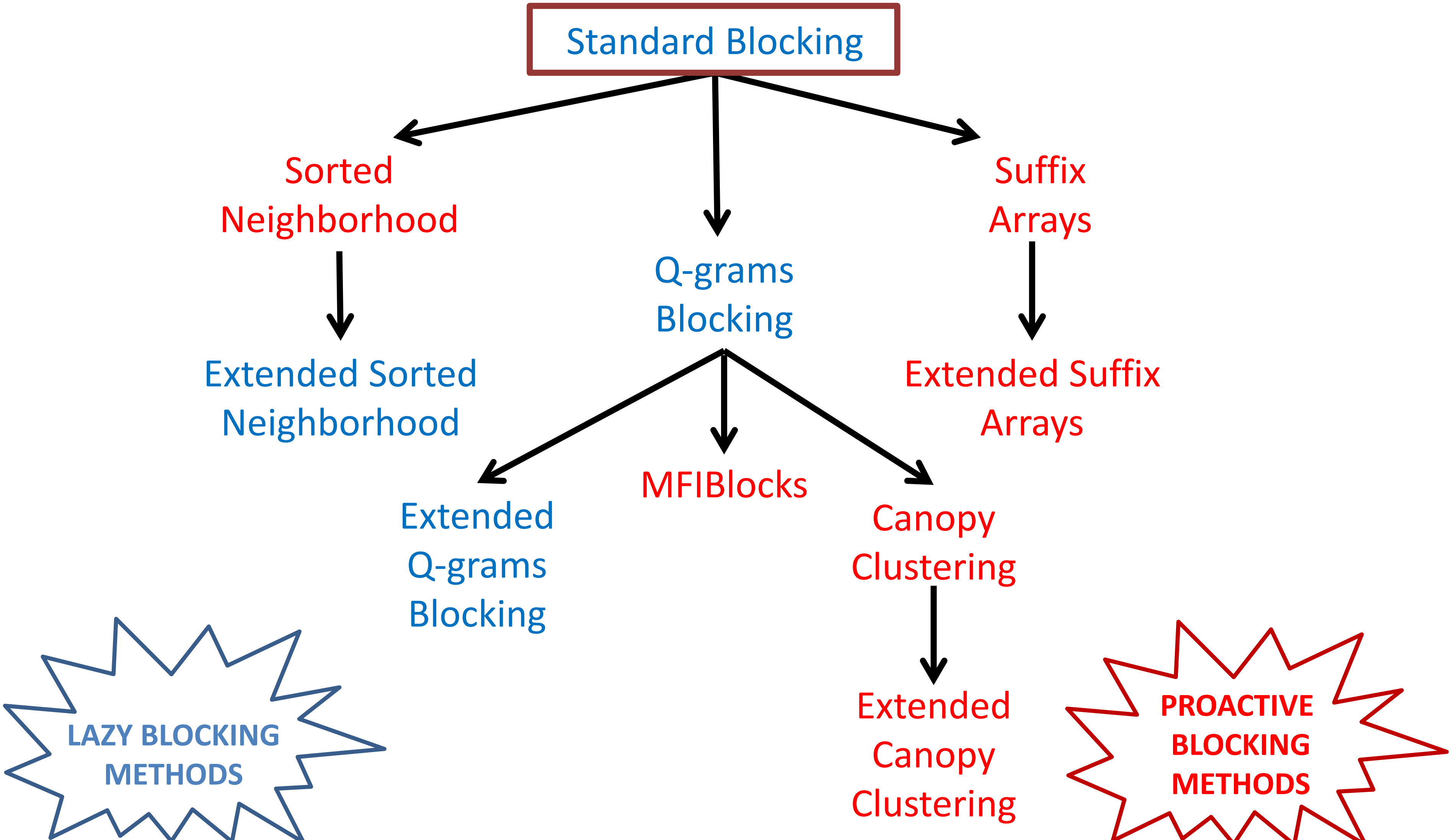
Overview of Schema-based Methods



Overview of Schema-based Methods



Overview of Schema-based Methods



Standard Blocking [Fellegi et. al., JASS 1969]

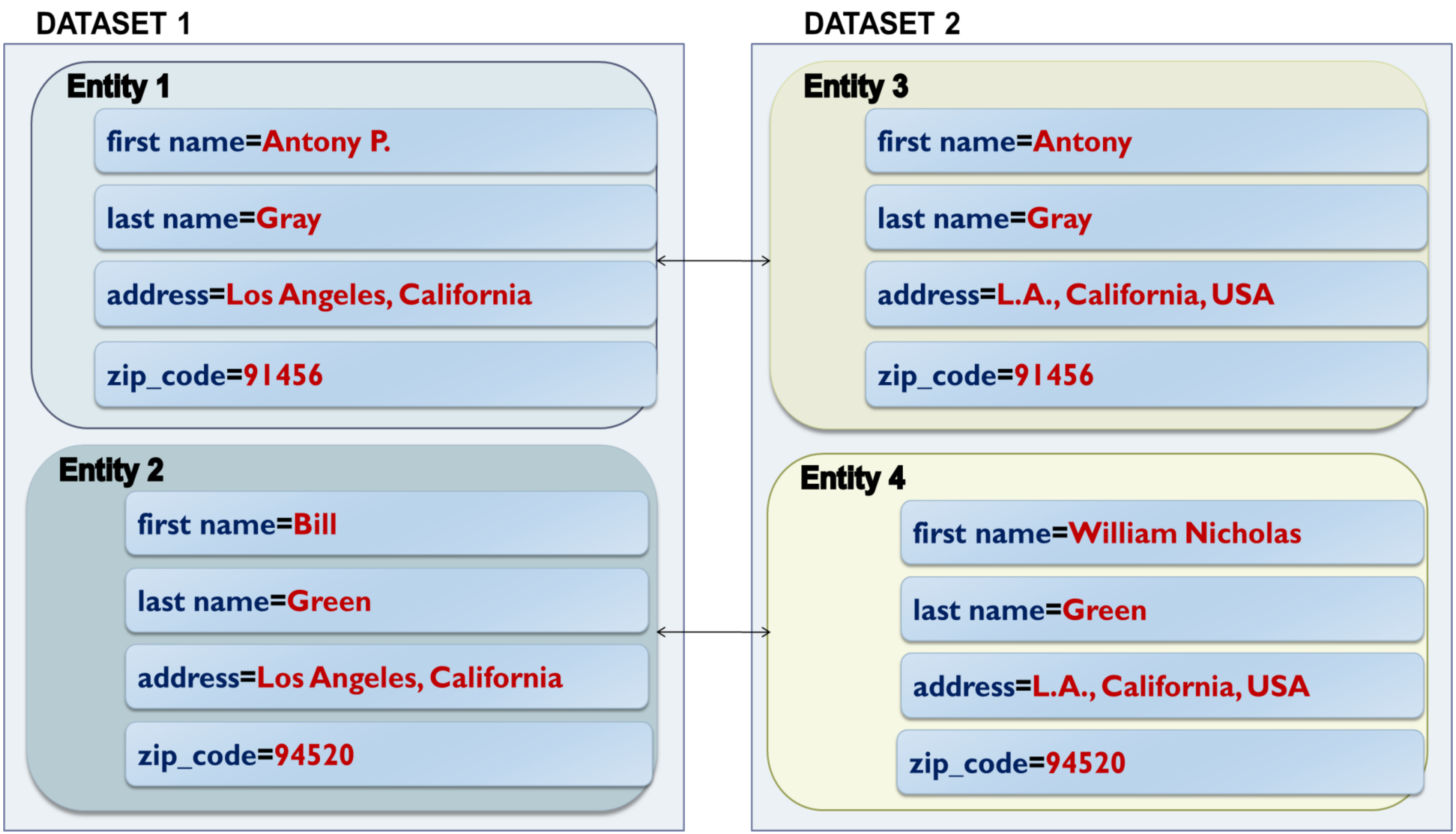
Earliest, simplest form of blocking.

Algorithm:

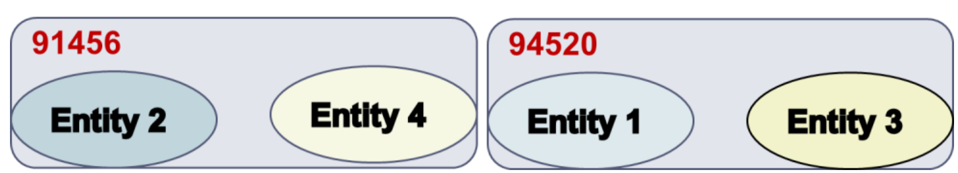
1. Select the most appropriate attribute name(s) w.r.t. noise and distinctiveness.
2. Transform the corresponding value(s) into a Blocking Key (BK)
3. For each BK, create one block that contains all entities having this BK in their transformation.

Works as a hash function! → Blocks on the **equality** of BKs

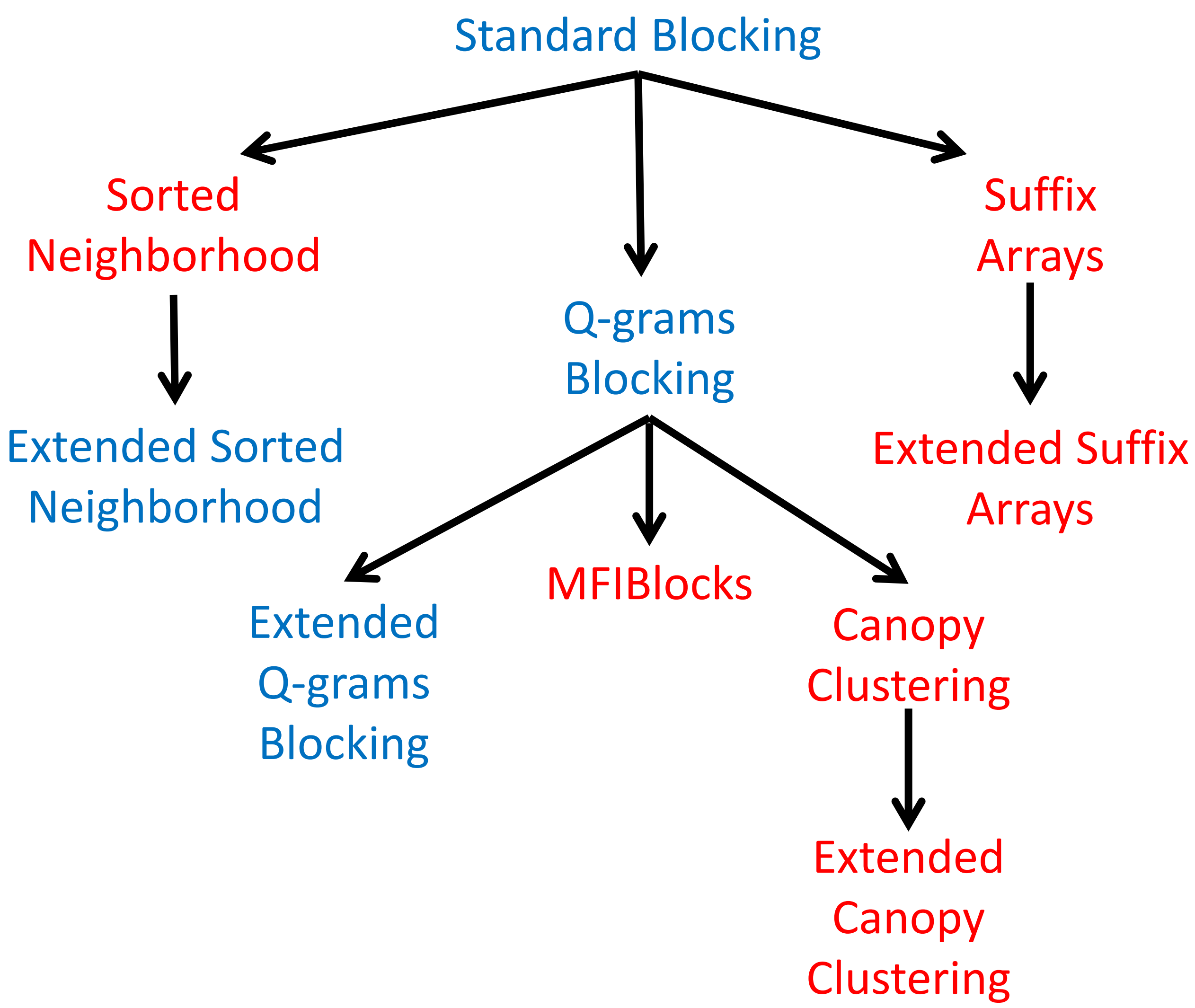
Example of Standard Blocking



Blocks on zip_code:

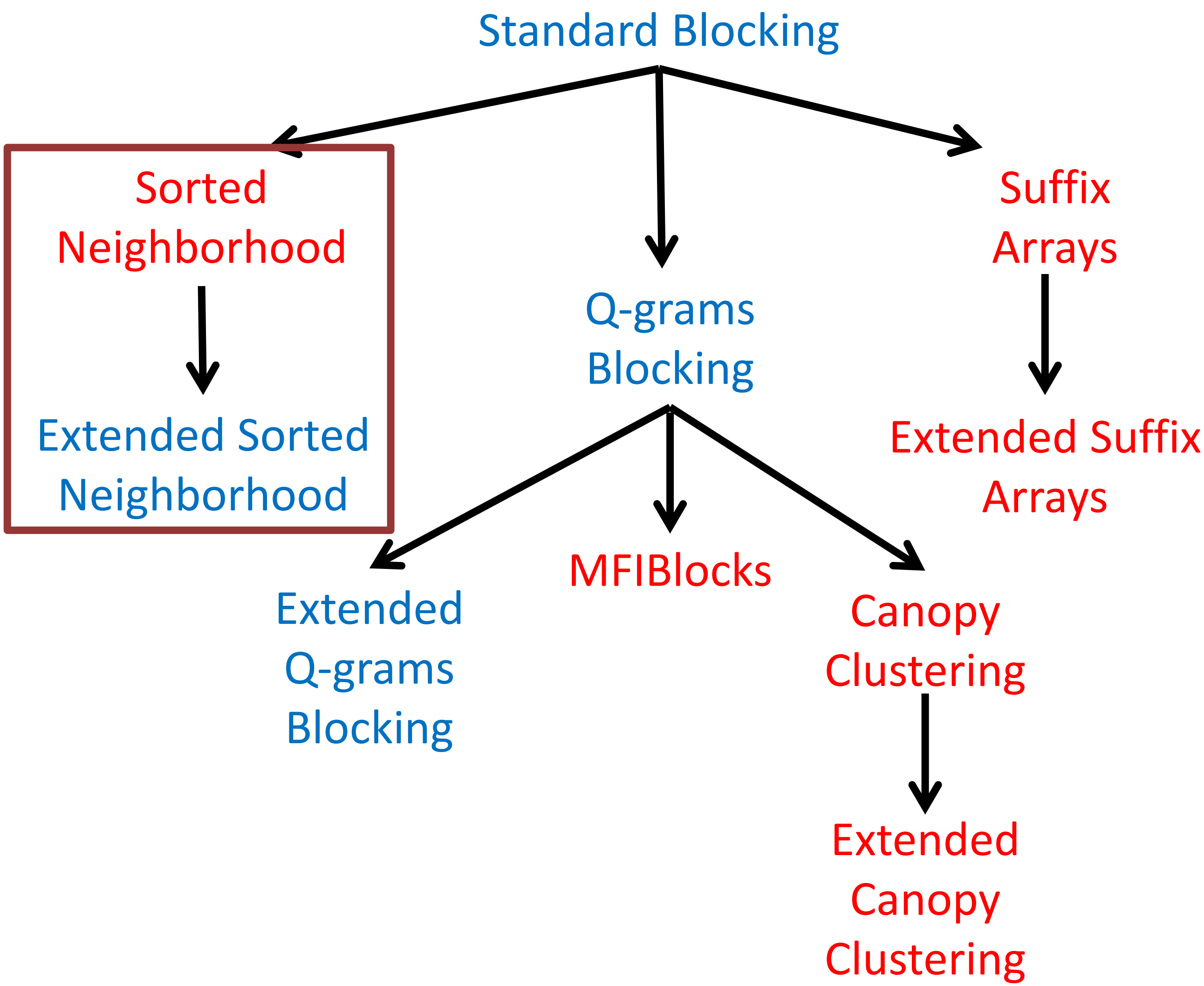


Overview of Schema-based Methods



Overview of Schema-based Methods

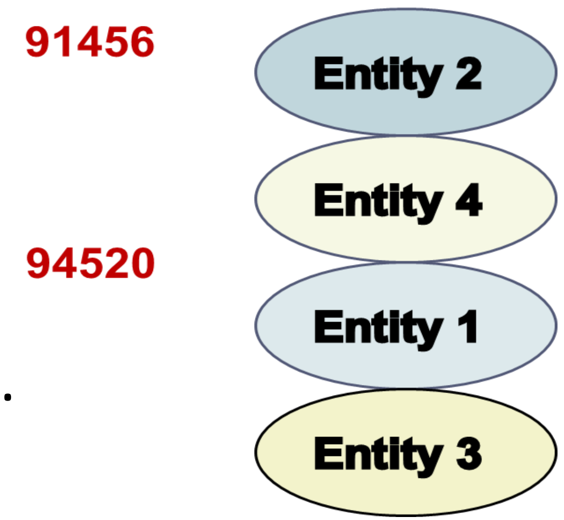
blocks contain entities with **similar** blocking keys



Sorted Neighborhood [Hernandez et. al., SIGMOD 1995]

Blocks on the similarity of BKs.

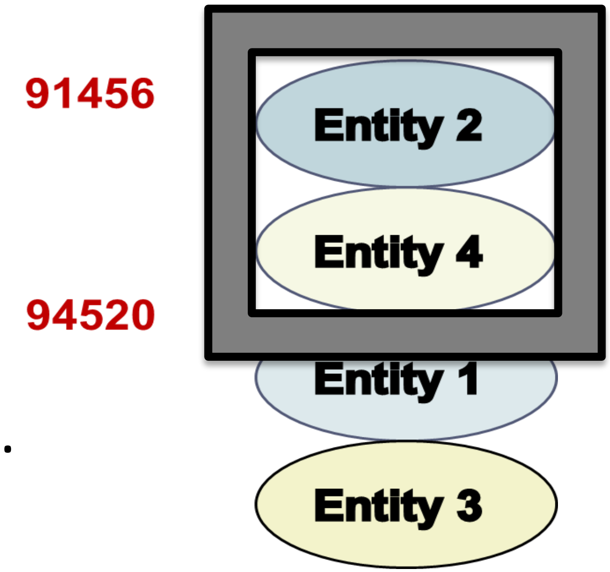
1. Entities are sorted in alphabetic order of BKs.
2. A window of fixed size slides over the sorted list of entities.
3. At each iteration, it compares the entities that co-occur within the window.



Sorted Neighborhood [Hernandez et. al., SIGMOD 1995]

Blocks on the similarity of BKs.

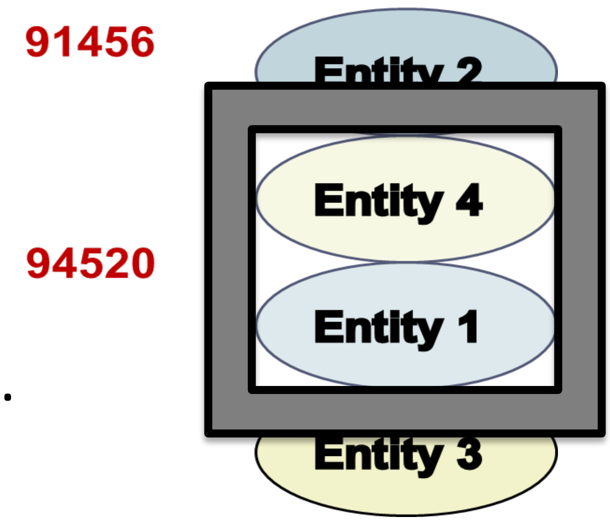
1. Entities are sorted in alphabetic order of BKs.
2. A window of fixed size slides over the sorted list of entities.
3. At each iteration, it compares the entities that co-occur within the window.



Sorted Neighborhood [Hernandez et. al., SIGMOD 1995]

Blocks on the similarity of BKs.

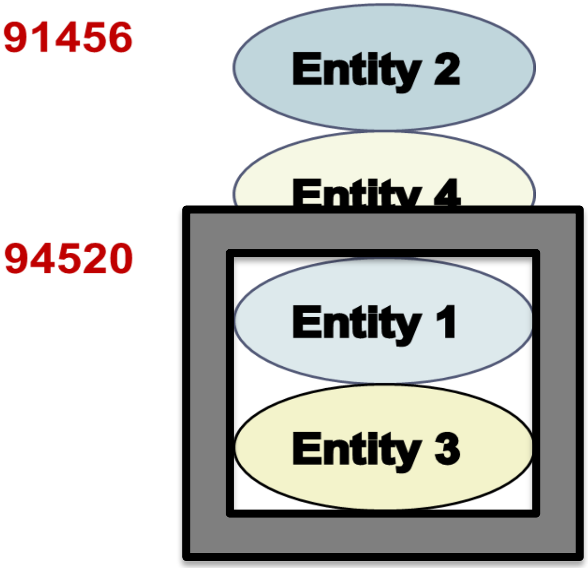
1. Entities are sorted in alphabetic order of BKs.
2. A window of fixed size slides over the sorted list of entities.
3. At each iteration, it compares the entities that co-occur within the window.



Sorted Neighborhood [Hernandez et. al., SIGMOD 1995]

Blocks on the similarity of BKs.

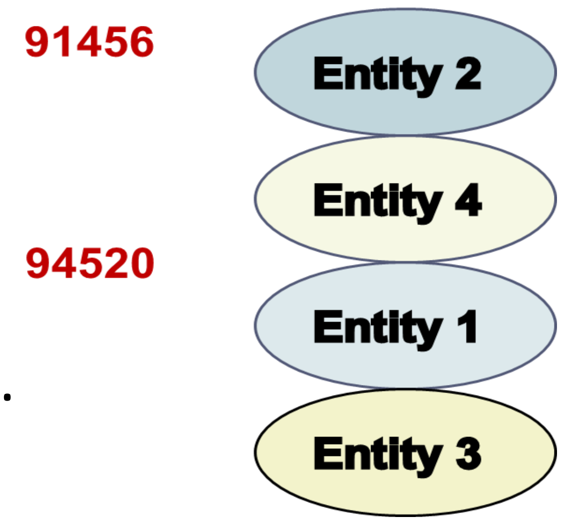
1. Entities are sorted in alphabetic order of BKs.
2. A window of fixed size slides over the sorted list of entities.
3. At each iteration, it compares the entities that co-occur within the window.



Sorted Neighborhood [Hernandez et. al., SIGMOD 1995]

Blocks on the similarity of BKs.

1. Entities are sorted in alphabetic order of BKs.
2. A window of fixed size slides over the sorted list of entities.
3. At each iteration, it compares the entities that co-occur within the window.



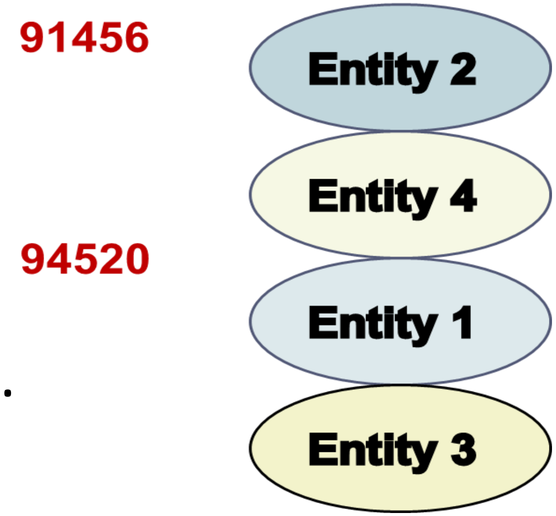
Extended Sorted Neighborhood [Christen, TKDE 2011]

2'. A window of fixed size slides over the sorted list of **BKs**.

Sorted Neighborhood [Hernandez et. al., SIGMOD 1995]

Blocks on the similarity of BKs.

1. Entities are sorted in alphabetic order of BKs.
2. A window of fixed size slides over the sorted list of **entities**.
3. At each iteration, it compares the entities that co-occur within the window.



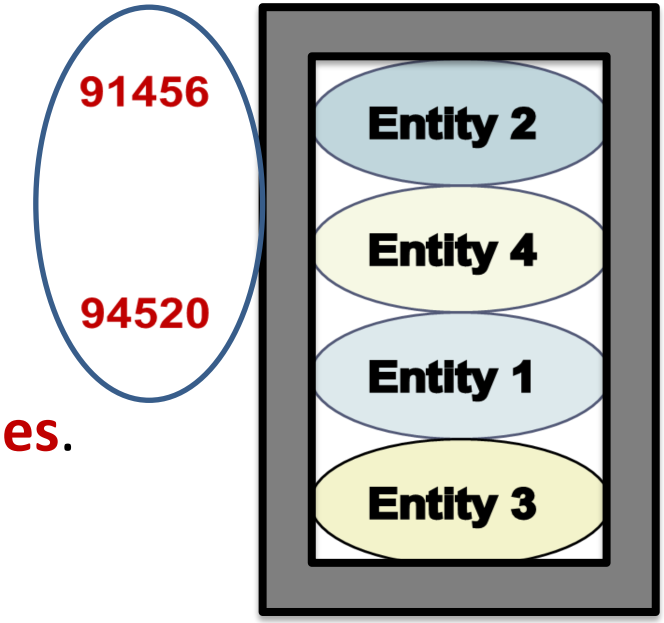
Extended Sorted Neighborhood [Christen, TKDE 2011]

2’. A window of fixed size slides over the sorted list of **BKs**.

Sorted Neighborhood [Hernandez et. al., SIGMOD 1995]

Blocks on the similarity of BKs.

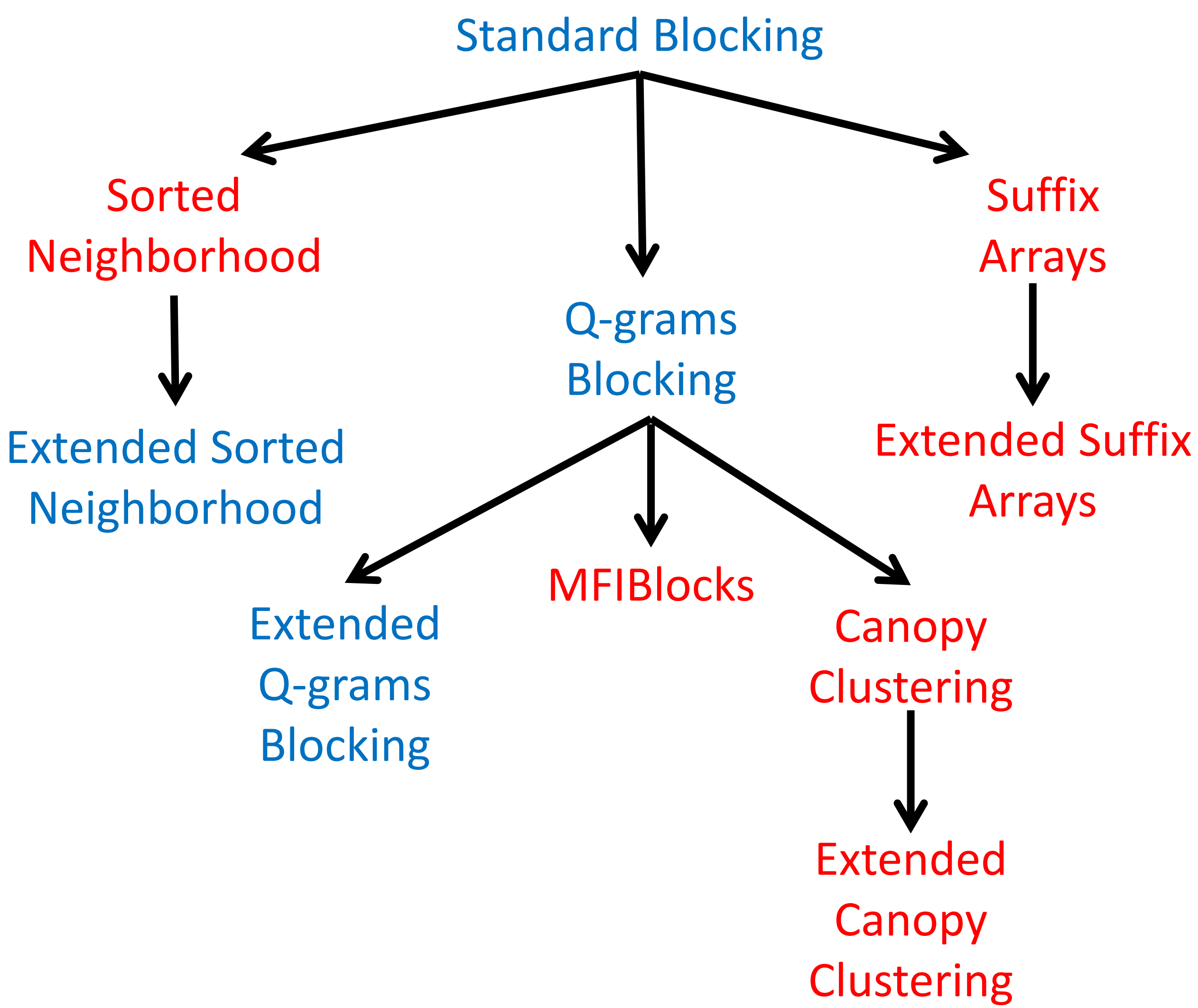
1. Entities are sorted in alphabetic order of BKs.
2. A window of fixed size slides over the sorted list of **entities**.
3. At each iteration, it compares the entities that co-occur within the window.



Extended Sorted Neighborhood [Christen, TKDE 2011]

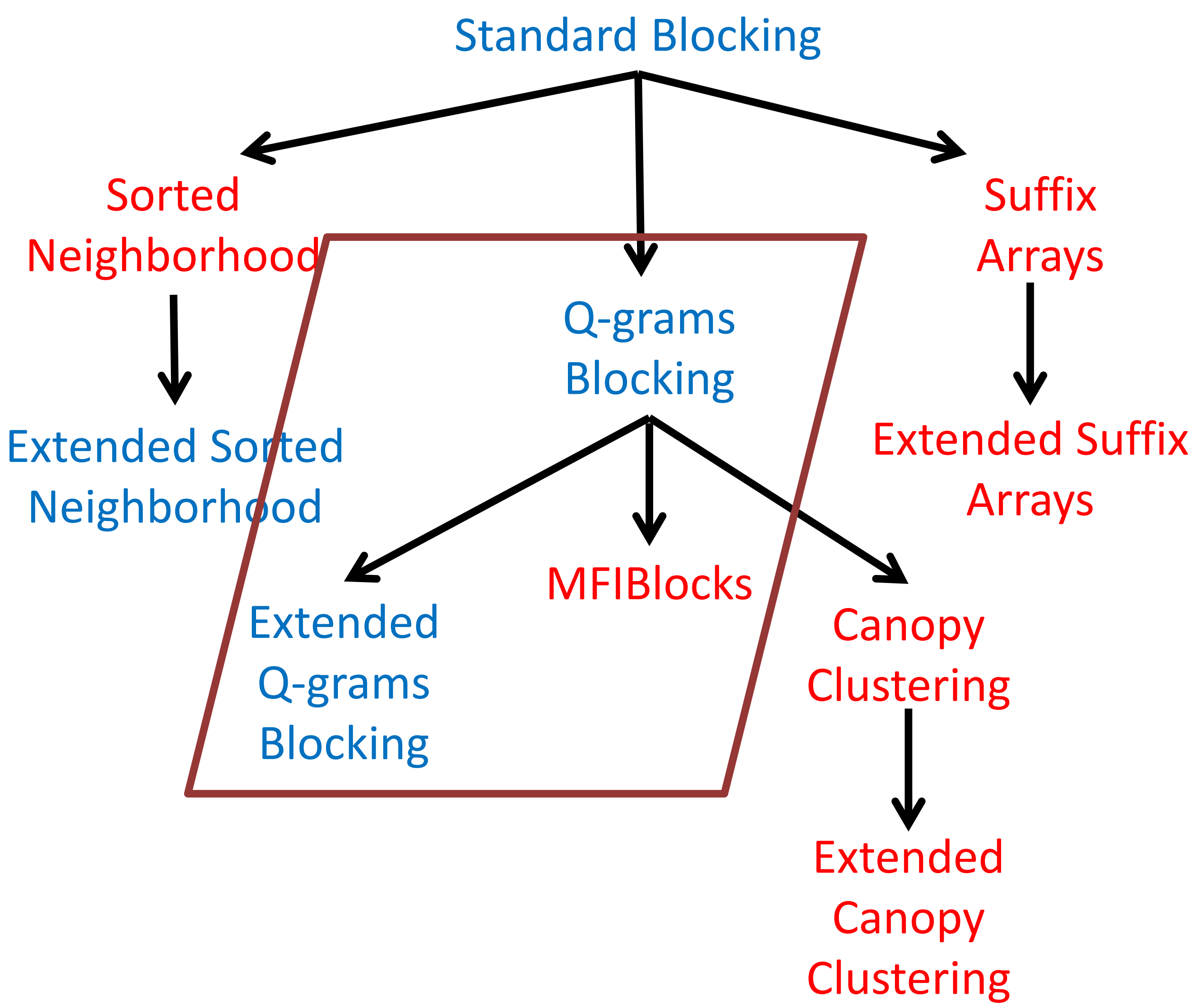
2’. A window of fixed size slides over the sorted list of **BKs**.

Overview of Schema-based Methods



Overview of Schema-based Methods

blocks contain entities with **same, or similar** blocking keys

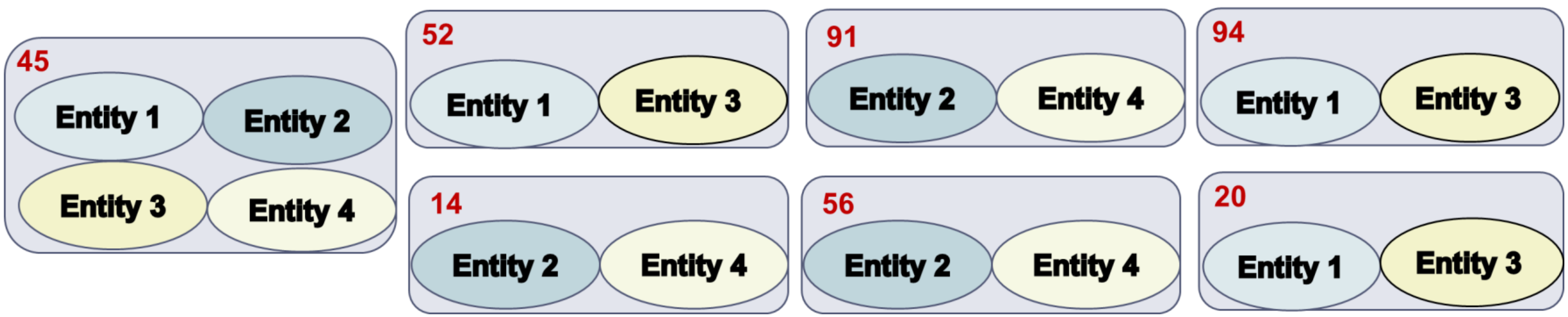


Q-grams Blocking [Gravano et. al., VLDB 2001]

Blocks on equality of BKs.

Converts every BK into the list of its *q*-grams.

For *q*=2, the BKs *91456* and *94520* yield the following blocks:



- Advantage:

  robust to noisy BKVs

- Drawback:

  larger blocks → higher computational cost

Extended Q-grams Blocking [Baxter et. al., KDD 2003]

BKs of higher discriminativeness:

instead of individual *q*-grams, BKs from combinations of *q*-grams.

Additional parameter:

threshold $t \in (0,1)$ specifies the minimum number of *q*-grams per BK as follows: $\boldsymbol{l_{min} = max(1, \lfloor k \cdot t \rfloor)}$, where k is the number of *q*-grams from the original BK

Example:

for BK= 91456, q=2 and t=0.9,

we have l_{min}=3 and the following valid BKs:

91_14_45_56

91_14_45

91_14_56

91_45_56

14_45_56

MFIBlocks [Kenig et. al., IS 2013]

Based on mining Maximum Frequent Itemsets.

Algorithm:

- Place all entities in a pool
- while (minimum_support > 2)
  - For each itemset that satisfies minimum_support
    - Create a block **b**
    - If **b** satisfies certain constraints (Block Cleaning)
      - remove its entities from the pool
      - retain the best comparisons (Comparison Cleaning)
  - decrease minimum_support

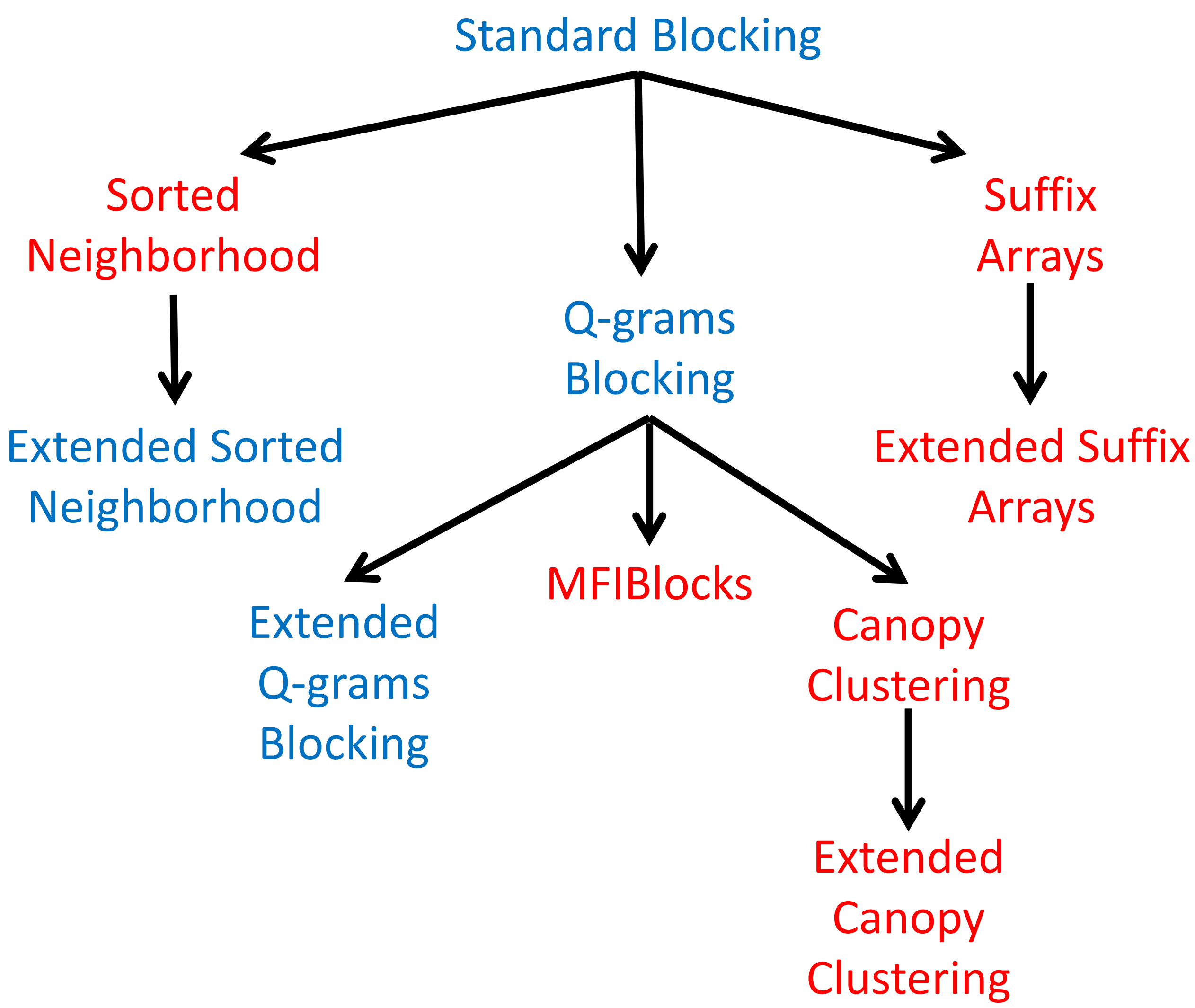
Pros:

- Usually the most effective blocking method for relational data → maximizes PQ (precision)

Cons:

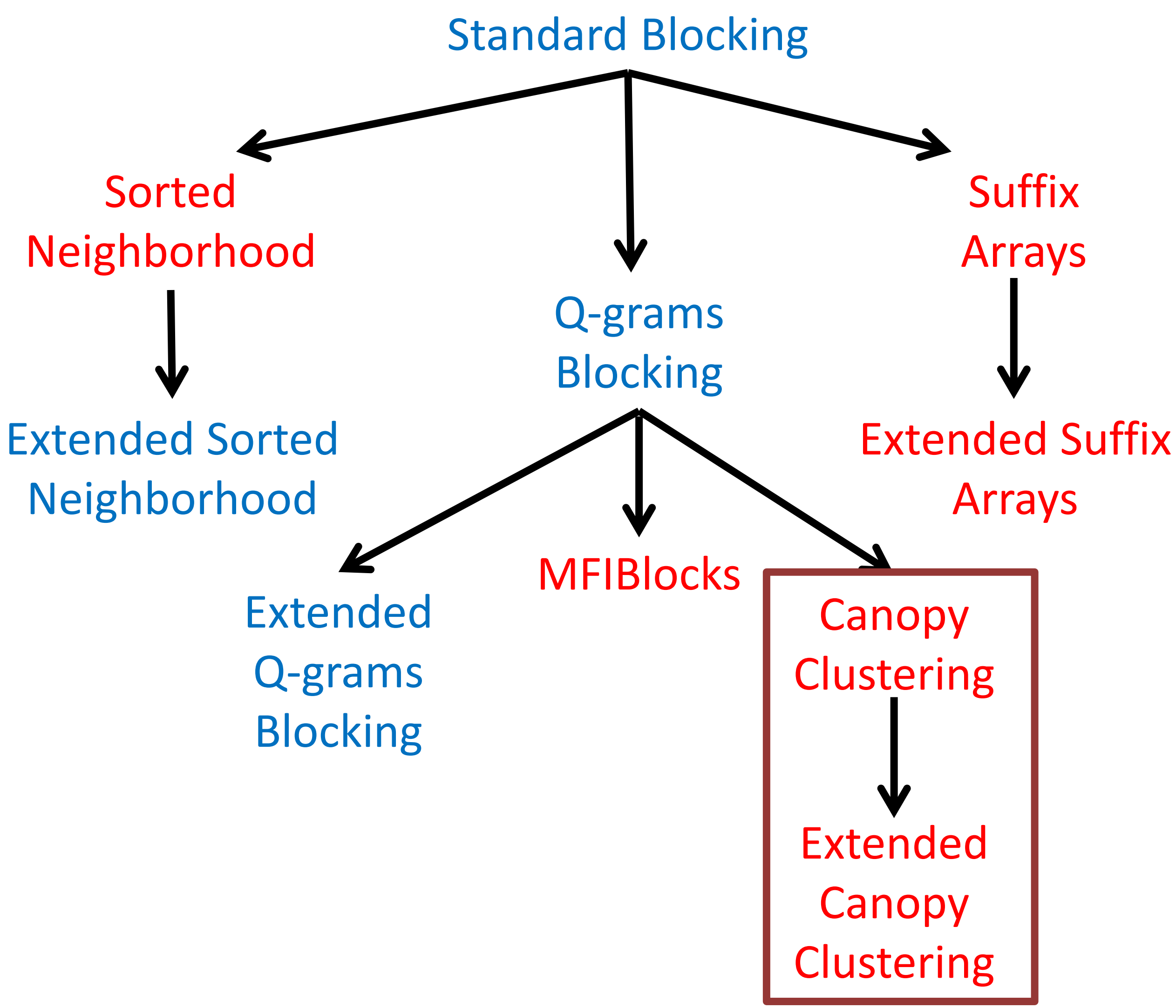
- Difficult to configure
- Time consuming

Overview of Schema-based Methods



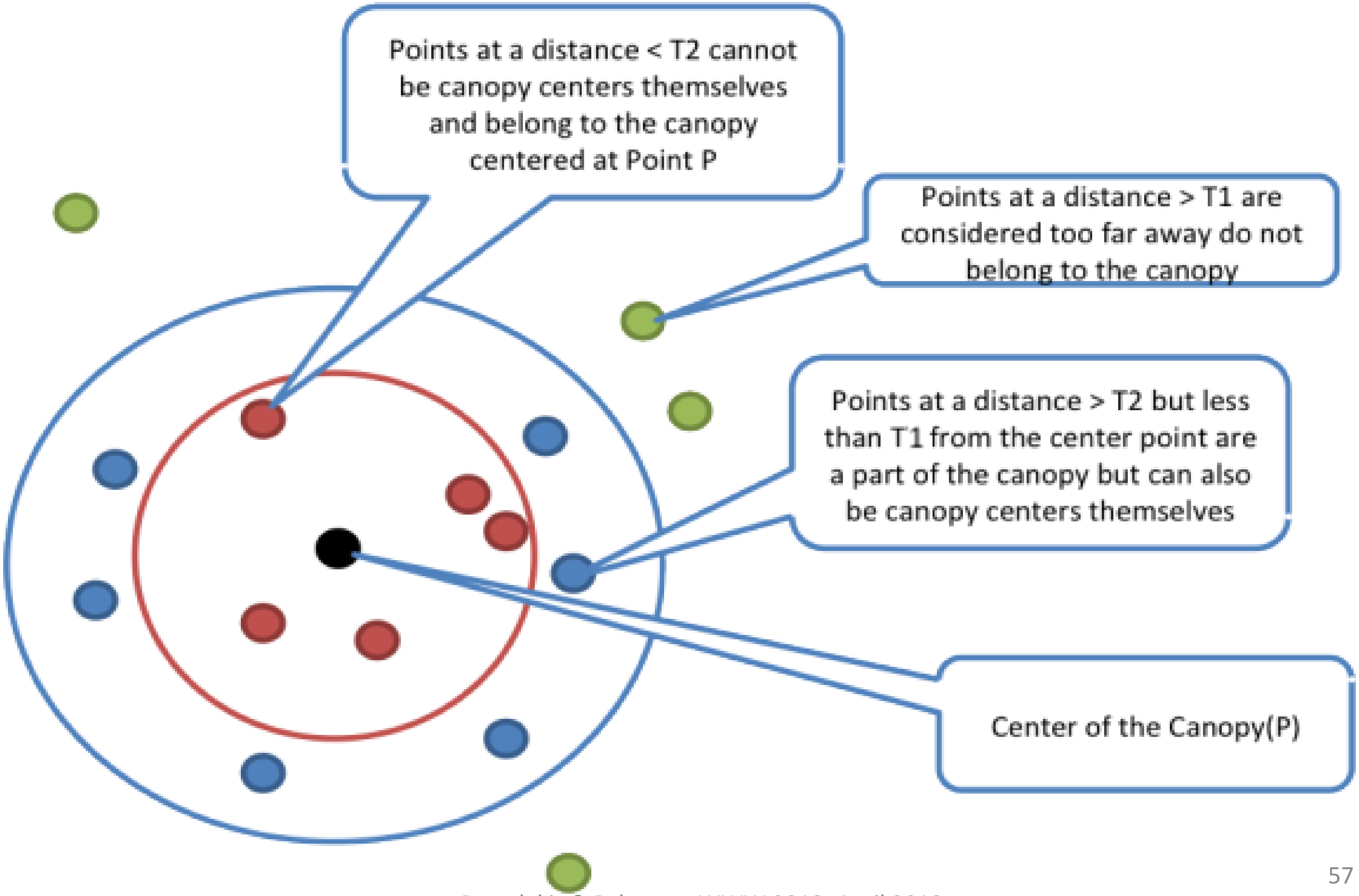
Overview of Schema-based Methods

blocks contain entities with **similar** blocking keys



Canopy Clustering [McCallum et. al., KDD 2000]

Blocks on similarity of BKs.



Extended Canopy Clustering [Christen, TKDE 2011]

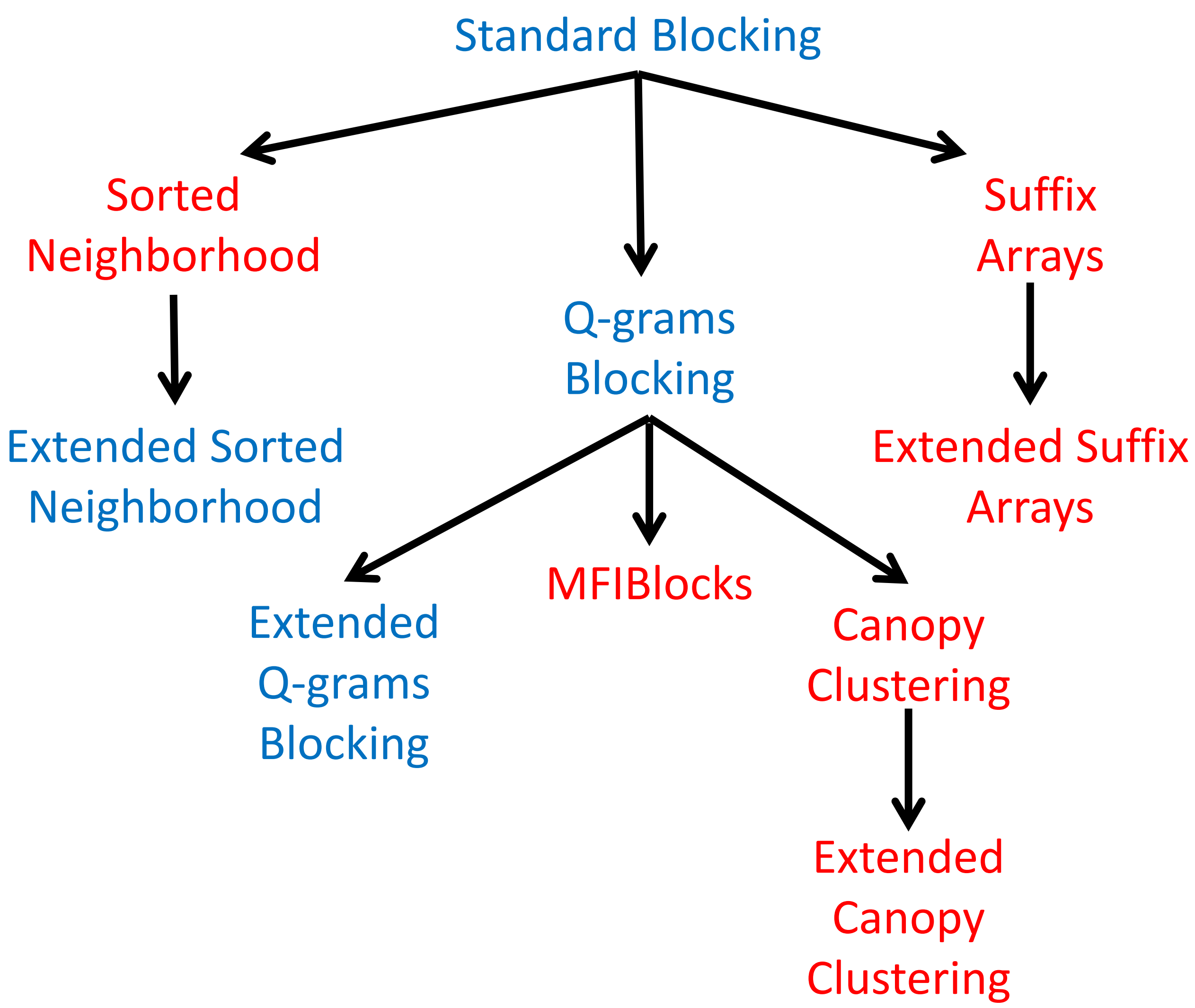
Canopy Clustering is too sensitive w.r.t. its weight thresholds:

- high values may leave many entities out of blocks.

Solution: **Extended Canopy Clustering** [Christen, TKDE 2011]

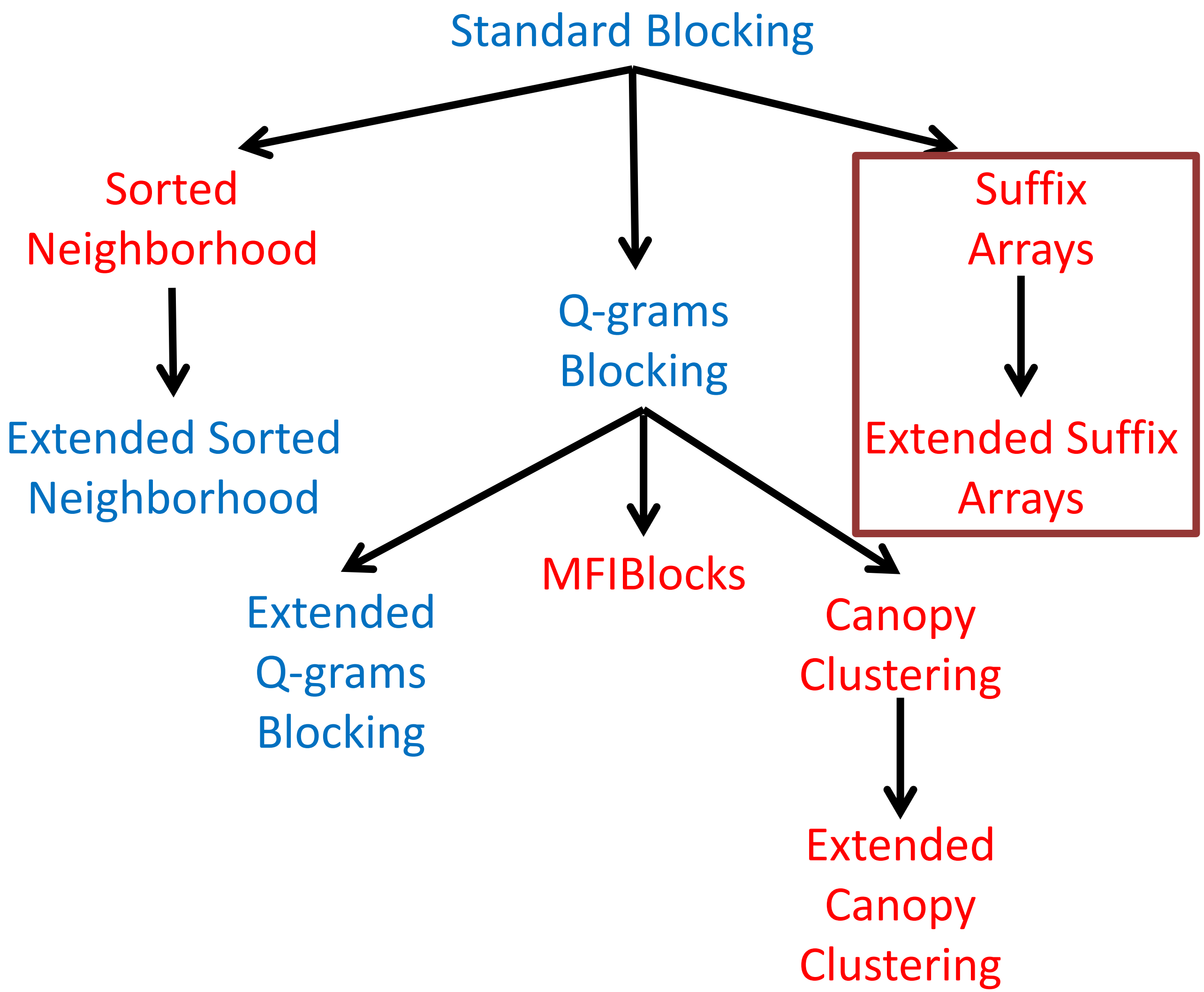
- cardinality thresholds instead of weight thresholds
- for each center of a canopy:
  - the $\boldsymbol{n_1}$ nearest entities are placed in its block
  - the $\boldsymbol{n_2}$ ($\leq n_1$) nearest entities are removed from the pool

Overview of Schema-based Methods



Overview of Schema-based Methods

blocks contain entities with **same** blocking keys

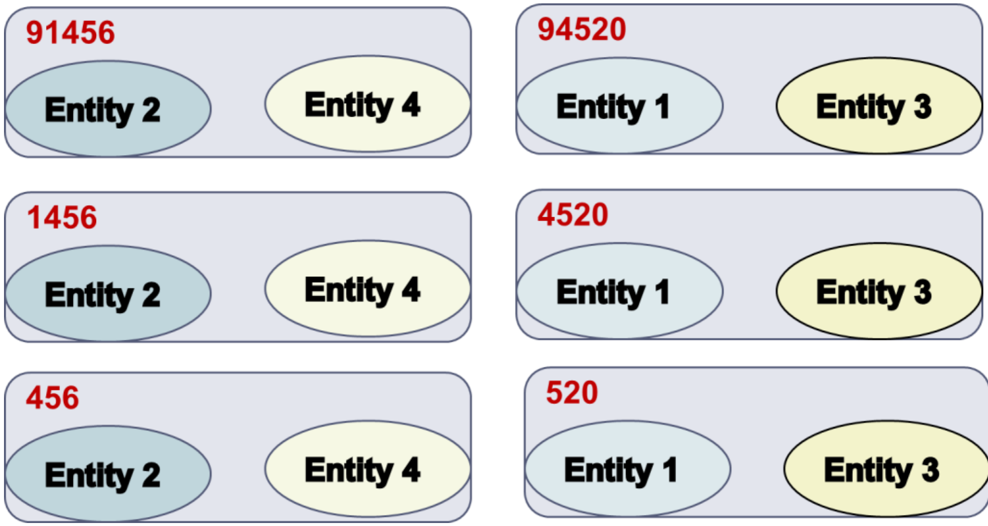


Suffix Arrays Blocking [Aizawa et. al., WIRI 2005]

Blocks on the equality of BKs.

Converts every BK to the list of its suffixes that are longer than a predetermined minimum length l_{min}.

For $l_{min}=3$, the keys *91456* and *94520* yield the blocks:



Frequent suffixes are discarded with the help of the parameter b_M:

- specifies the maximum number of entities per block

Extended Suffix Arrays Blocking [Christen, TKDE 2011]

Goal:

support errors at the end of BKs

Solution:

consider *all substrings* (not only suffixes) with more than l_{min} characters.

For l_{min}=3, the keys *91456* and *94520* are converted to the BKs:

91456,	94520
9145,	9452
1456,	4520
914,	945
145,	452
456	520

Summary of Blocking for Databases [Christen, TKDE2011]

1. They typically employ **redundancy** to ensure higher recall in the context of noise at the cost of lower precision (more comparisons). Still, recall remains low for many datasets.

2. Several parameters to be configured

   E.g., Canopy Clustering has the following parameters:

   I. String matching method

   II. Threshold t_1

   III. Threshold t_2

3. Schema-dependent → manual definition of BKs

Improving Blocking for Databases [Papadakis et. al., VLDB 2015]

Schema-agnostic blocking keys

- Use every token as a key
- Applies to all schema-based blocking methods
- Simplifies configuration, unsupervised approach

Performance evaluation

- For lazy blocking methods →
  very high, robust recall at the cost of more comparisons
- For proactive blocking methods →
  relative recall gets higher with more comparisons,
  absolute recall depends on block constraints

Part 3:

Block Building for Web Data

Characteristics of Web Data

Voluminous, (semi-)structured datasets.

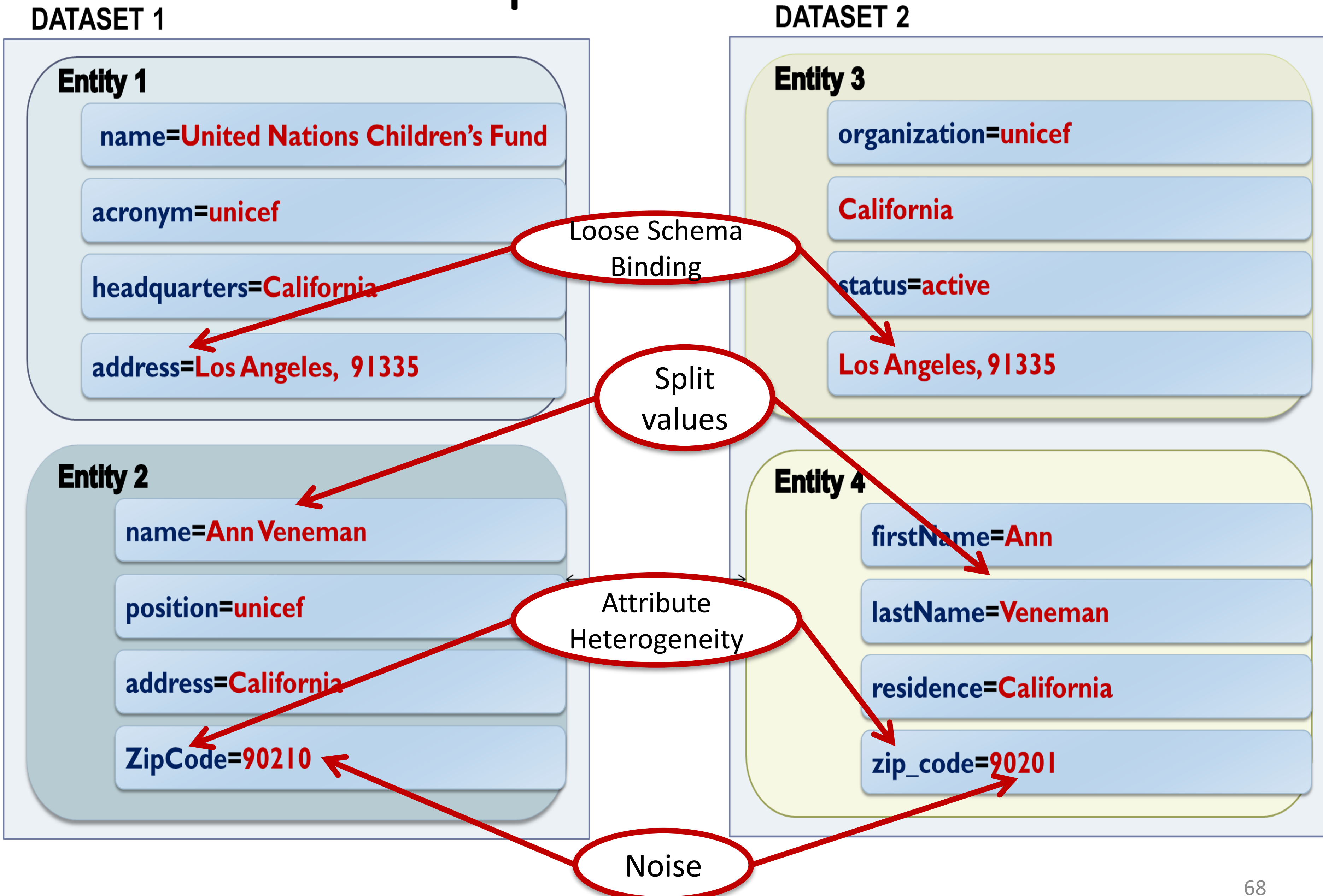
- DBPedia 2014: 3 billion triples and 38 million entities
- BTC09: 1.15 billion triples, 182 million entities.

Users are free to add attribute values and/or attribute names
→ unprecedented levels of schema heterogeneity.

- DBPedia 3.4: 50,000 attribute names
- Google Base: 100,000 schemata for 10,000 entity types
- BTC09: 136,000 attribute names

Several datasets produced by automatic information extraction techniques
→ noise, tag-style values.

Example of Web Data



Token Blocking [Papadakis et al., WSDM2011]

Functionality:

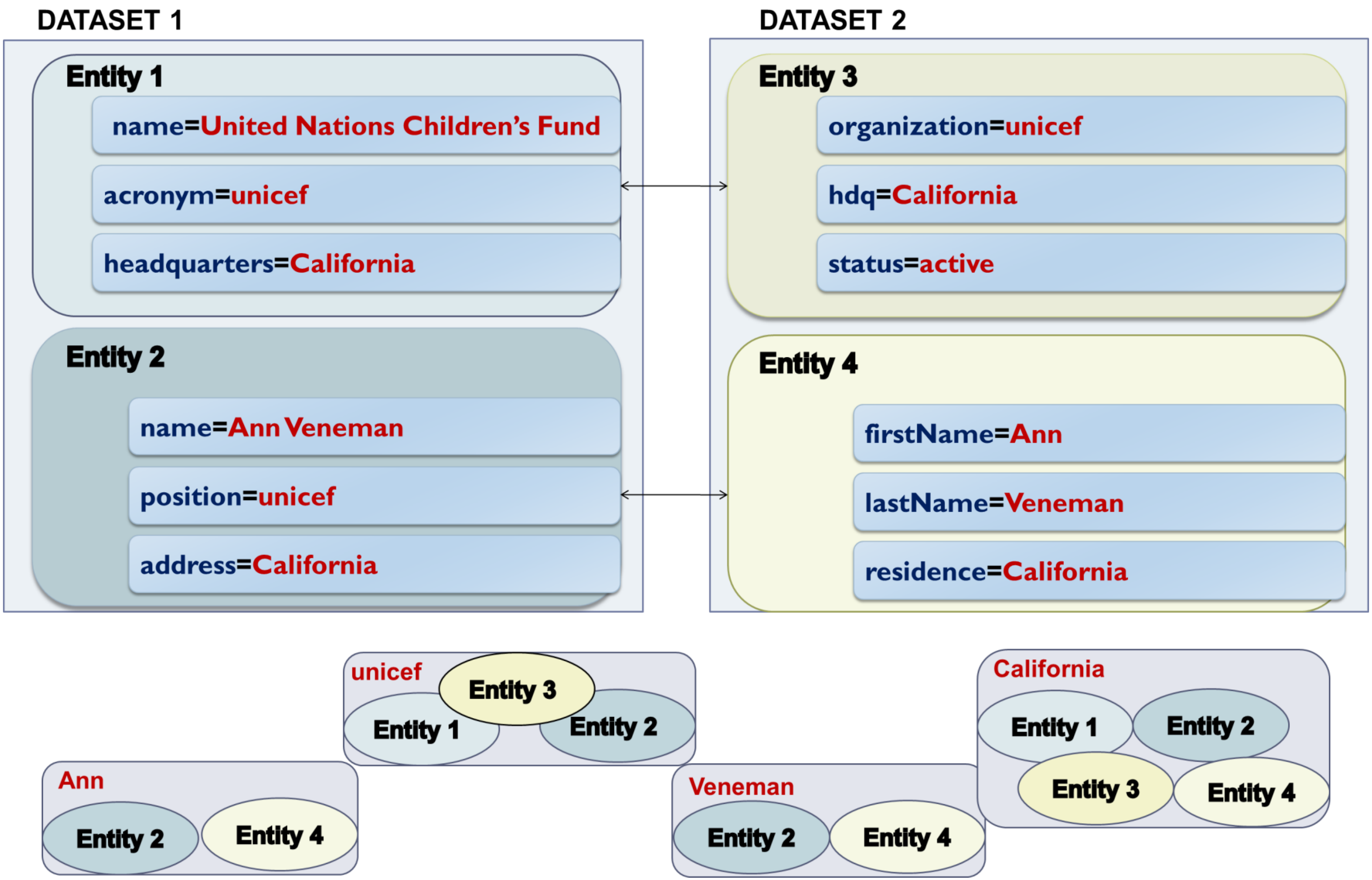
1. given an entity profile, extract all tokens that are contained in its attribute values.
2. create one block for every distinct token → each block contains all entities with the corresponding token*****.

Attribute-agnostic functionality:

- completely ignores all attribute names, but considers all attribute values
- efficient implementation with the help of inverted indices
- ***parameter-free***!

**Each block should contain at least two entities.*

Token Blocking Example

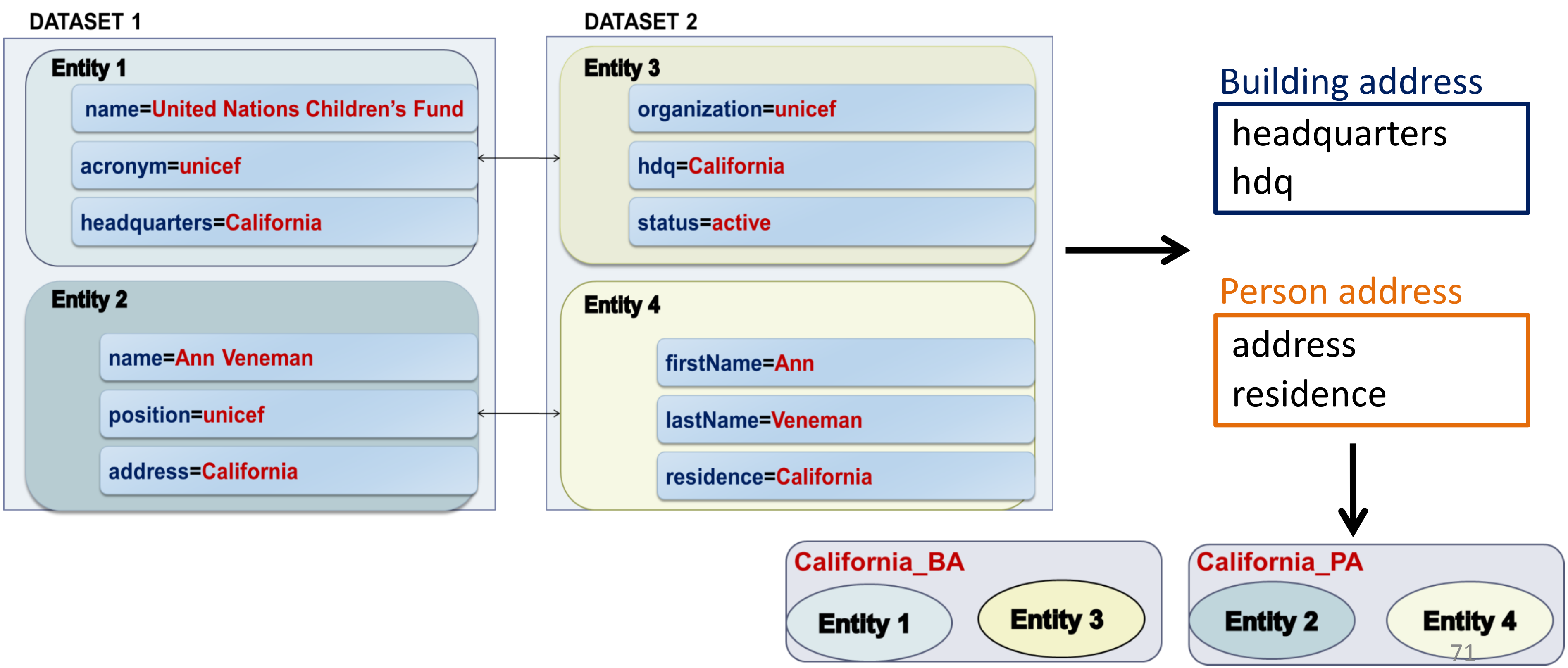


Attribute-Clustering Blocking

[Papadakis et. al., TKDE 2013]

Goal:

group attribute names into clusters s.t. we can apply Token Blocking independently inside each cluster, without affecting effectiveness
→ smaller blocks, higher efficiency.



Attribute-Clustering Blocking

Algorithm

- Create a graph, where every node represents an attribute name and its attribute values
- For each attribute name/node n_i
  - Find the most similar node n_j
  - If $sim(n_i,n_j) > 0$, add an edge $<n_i,n_j>$
- Extract connected components
- Put all singleton nodes in a “glue” cluster

Parameters

1. Representation model
   - Character n-grams, Character n-gram graphs, Tokens
2. Similarity Metric
   - Jaccard, Graph Value Similarity, TF-IDF

Attribute-Clustering vs Schema Matching

Similar to Schema Matching, …but fundamentally different:

1. Associated attribute names do not have to be semantically equivalent. They only have to produce good blocks

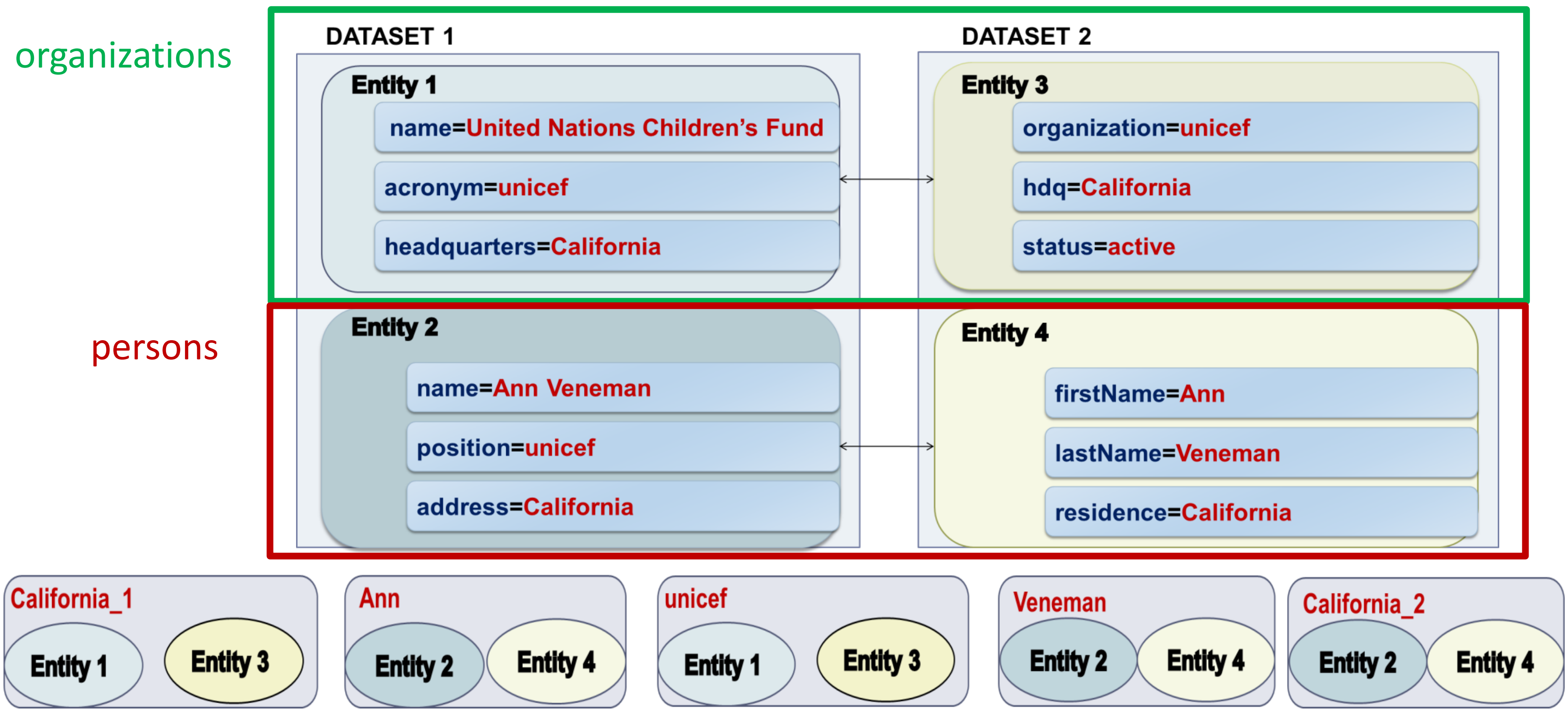
2. All singleton attribute names are associated with each other

3. Unlike Schema Matching, it scales to the very high levels of heterogeneity of Web Data
   – because of the above simplifying assumptions

TYPiMatch [Ma et. al., WSDM 2013]

Goal:

cluster entities into *overlapping types* and apply Token Blocking to the values of the best attribute for each type.



TYPiMatch

Algorithm:

1. Create a directed graph ***G***, where nodes correspond to tokens, and edges connect those co-occurring in the same entity profile, weighted according to conditional co-occurrence probability.
2. Convert **G** to undirected graph **G'** and get maximal cliques (parameter ϑ).
3. Create an undirected graph **G''**, where nodes correspond to cliques and edges connect the frequently co-occurring cliques (parameter ε).
4. Get connected components to form entity types.
5. Get best attribute name for each type using an entropy-based criterion.

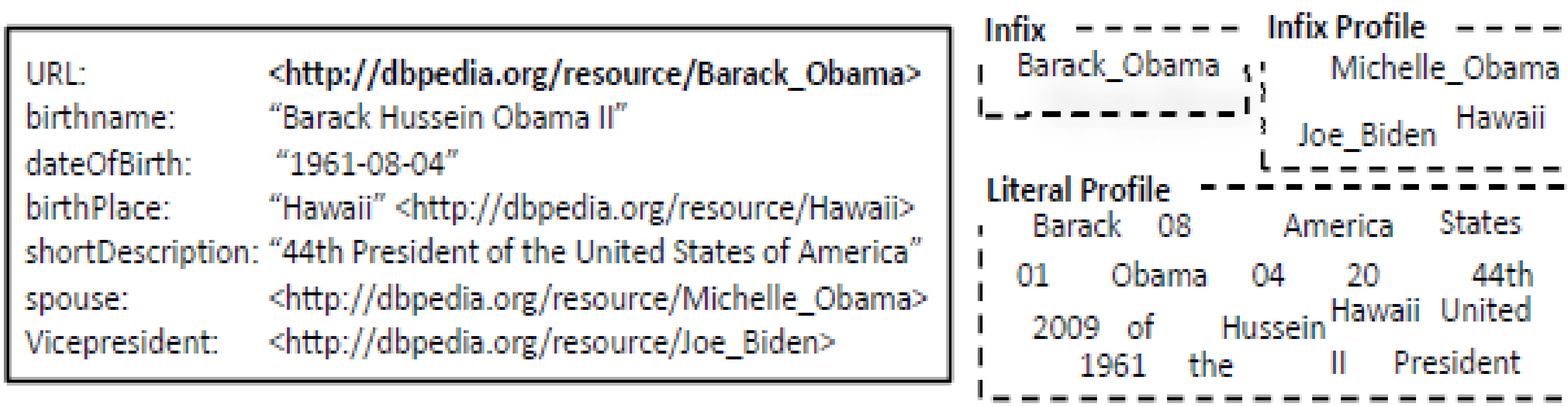
Evidence for Semantic Web Blocking

For Semantic Web data, three sources of evidence create blocks of lower redundancy than Token Blocking:

1. Infix

Prefix	Infix	Suffix
http://dblp.l3s.de/d2r/resource/publications/books/sp/wooldridgeV99	/ThalmannN99	
http://bibsonomy.org/uri/bibtexkey/books/sp/wooldridgeV99	/ThalmannN99	/dblp

2. Infix Profile
3. Literal Profile



Algorithm for URI decomposition in PI(S)-form in [Papadakis et al., iiWAS 2010].

URI Semantics Blocking [Papadakis et al., WSDM2012]

The above sources of evidence lead to 3 parameter-free blocking methods:

1. **Infix Blocking**
   every block contains all entities whose URI has a specific Infix
2. **Infix Profile Blocking**
   every block corresponds to a specific Infix (of an attribute value) and contains all entities having it in their Infix Profile
3. **Literal Profile Blocking**
   every block corresponds to a specific token and contains all entities having it in their Literal Profile

Individually, these atomic methods have limited coverage and, thus, low effectiveness (e.g., Infix Blocking does not cover blank nodes).

However, they are complementary and can be combined into composite blocking methods with high robustness and effectiveness!

Summary of Blocking for Web Data

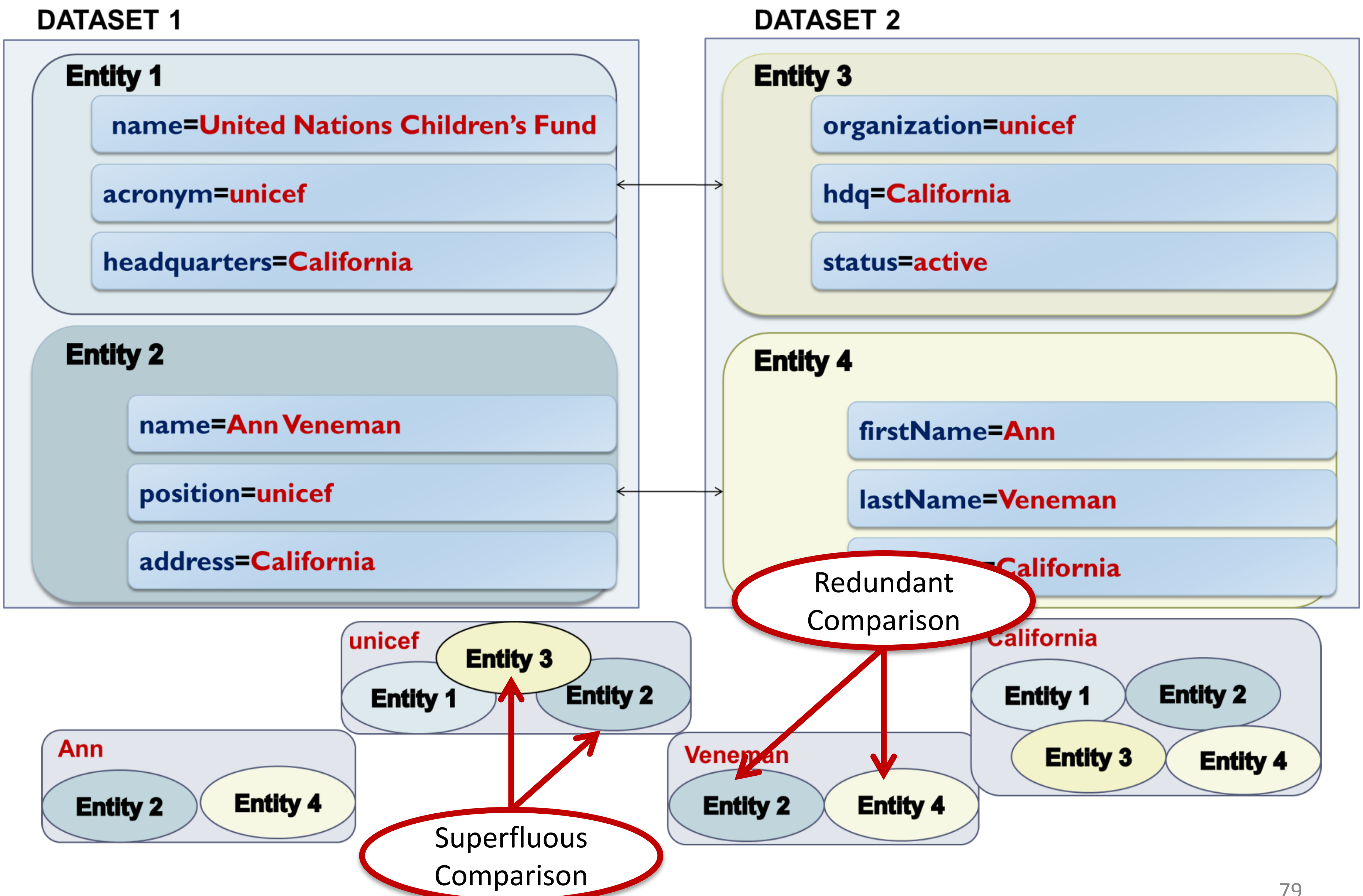
High Recall in the context of noisy entity profiles and extreme schema heterogeneity thanks to:

1. redundancy that reduces the likelihood of missed matches.
2. attribute-agnostic functionality that requires no schema semantics.

Low Precision because:

- the blocks are overlapping → redundant comparisons
- high number of comparisons between irrelevant entities → superfluous comparisons

Token Blocking Example



Part 4:

Block Processing Techniques

Outline

1. Introduction to Blocking
2. Blocking Methods for Relational Data
3. Blocking Methods for Web Data
4. Block Processing Techniques
   - Block Purging
   - Block Filtering
   - Block Clustering
   - Comparison Propagation
   - Iterative Blocking
5. Meta-blocking
6. Entity Matching
7. Entity Clustering
8. Massive Parallelization Methods
9. Progressive Entity Resolution
10. Challenges
11. JedAI Toolkit
12. Conclusions

General Principles

Goals:

1. eliminate *all* redundant comparisons
2. avoid *most* superfluous comparisons

without affecting matching comparisons (i.e., PC).

Depending on the granularity of their functionality, they are distinguished into:

1. Block-refinement
2. Comparison-refinement
    - Iterative Methods

Block Purging

Exploits power-law distribution of block sizes.

Targets **oversized blocks** (i.e., many comparisons, no duplicates)

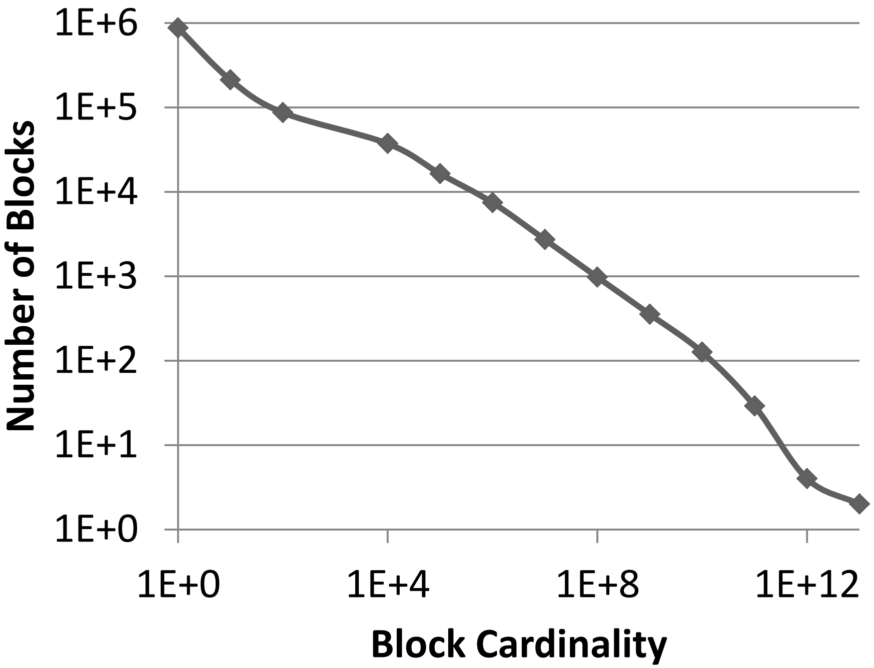
Discards them by setting an upper limit on:

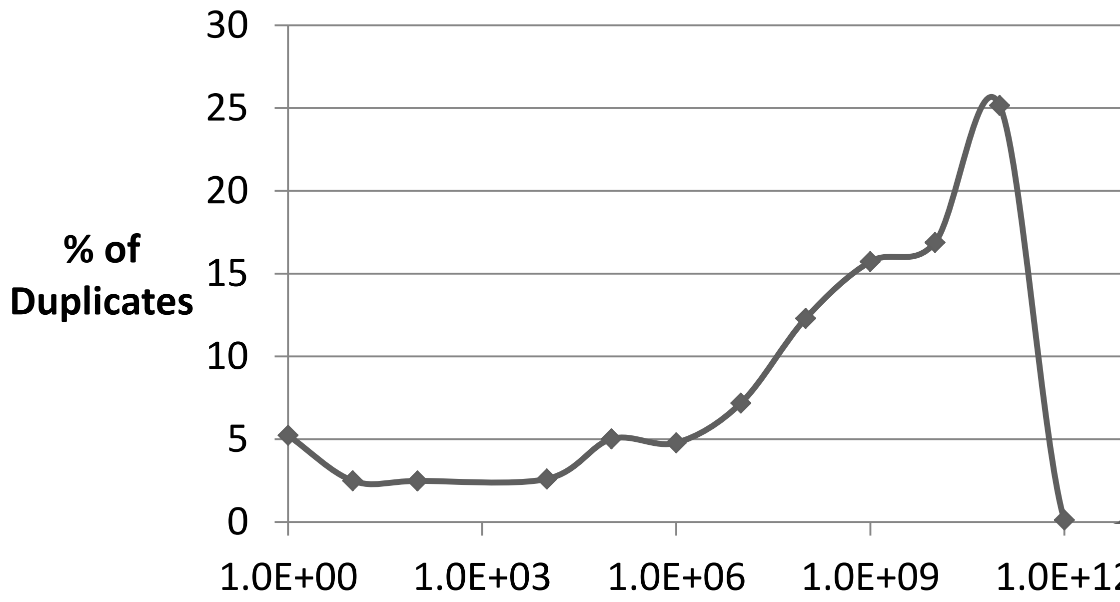
- the size of each block [Papadakis et al., WSDM 2011],
- the cardinality of each block [Papadakis et al., WSDM 2012]

Core method:

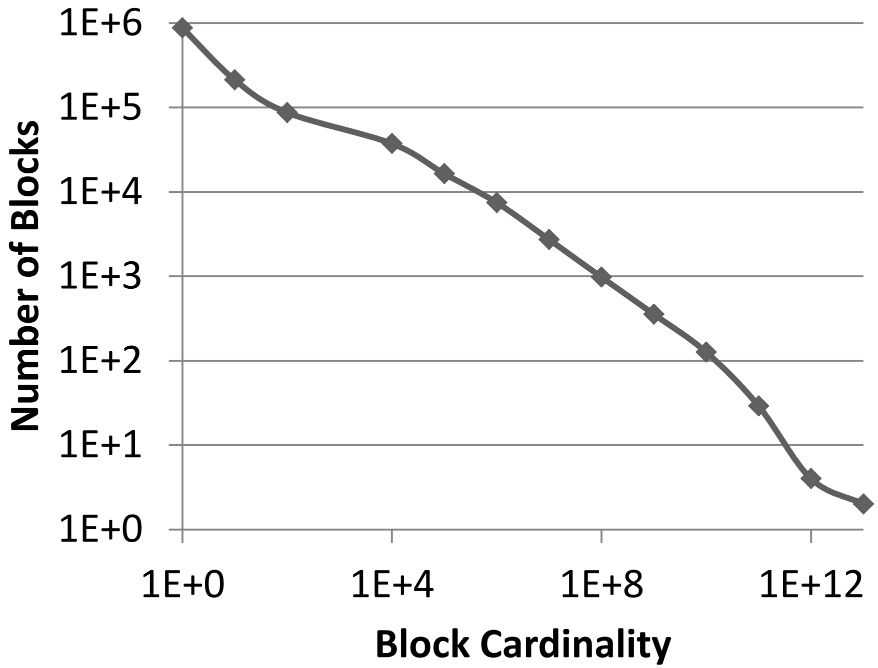
- Low computational cost.
- Low impact on effectiveness.
- Boosts efficiency to a large extent.

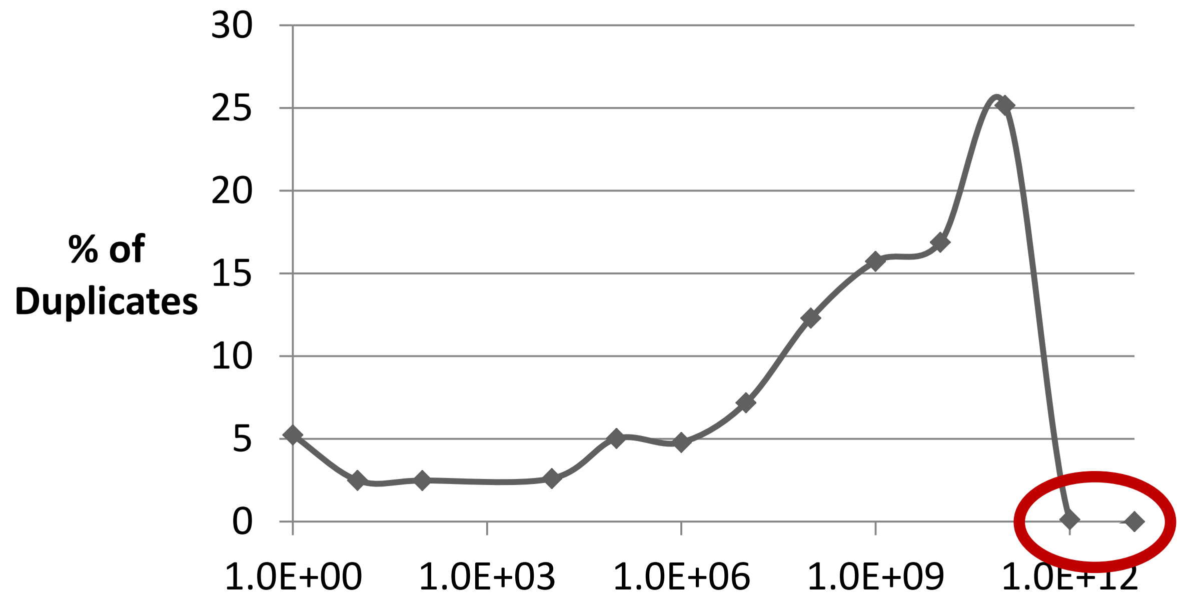
Distributions of Block Sizes and Duplicates



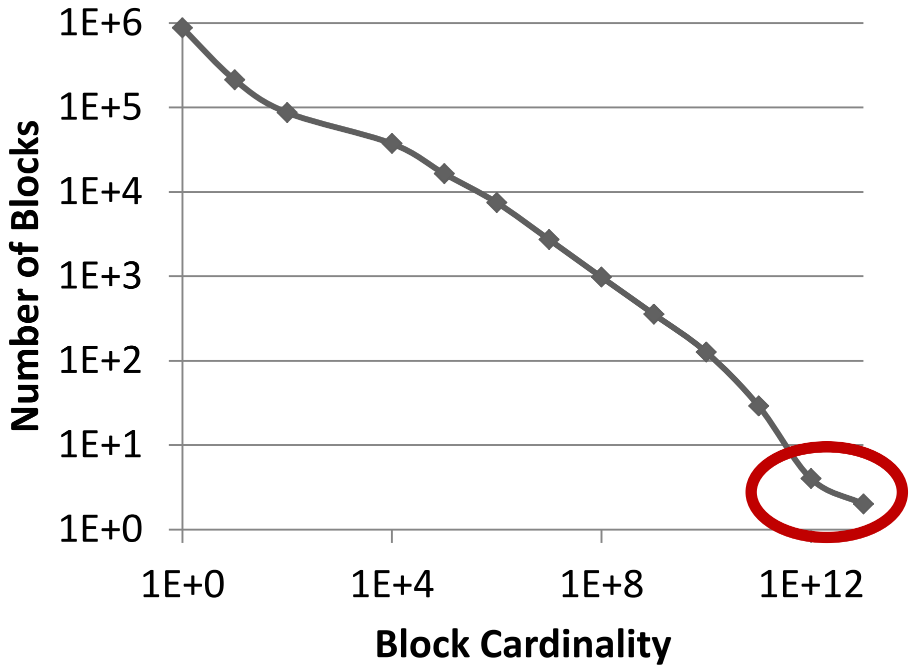


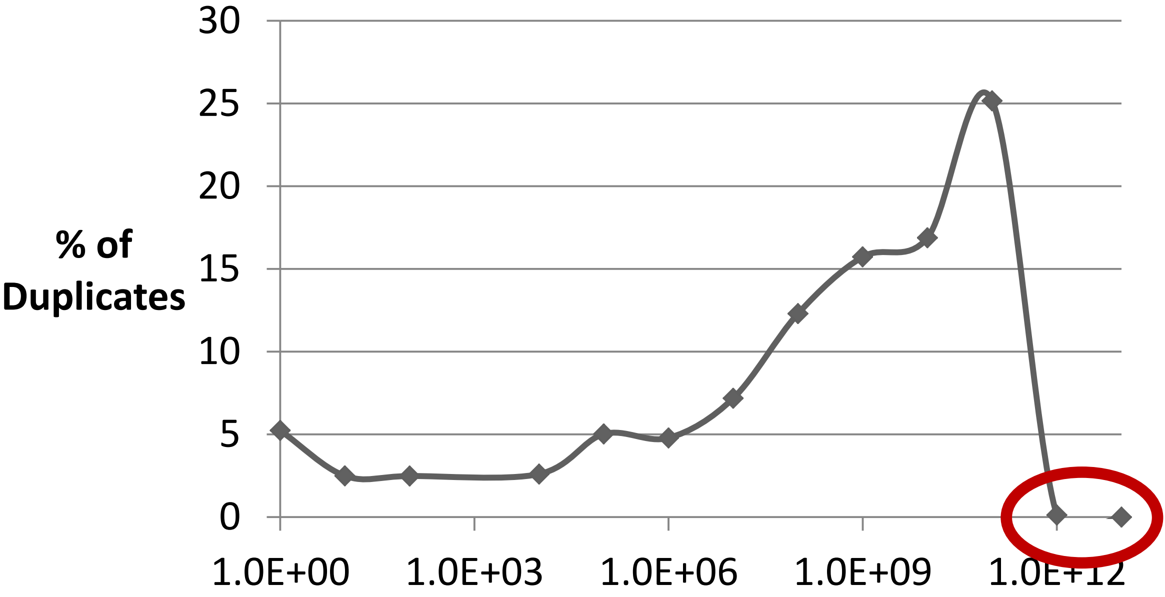
Distributions of Block Sizes and Duplicates





Distributions of Block Sizes and Duplicates





Block Filtering [Papadakis et. al, EDBT 2016]

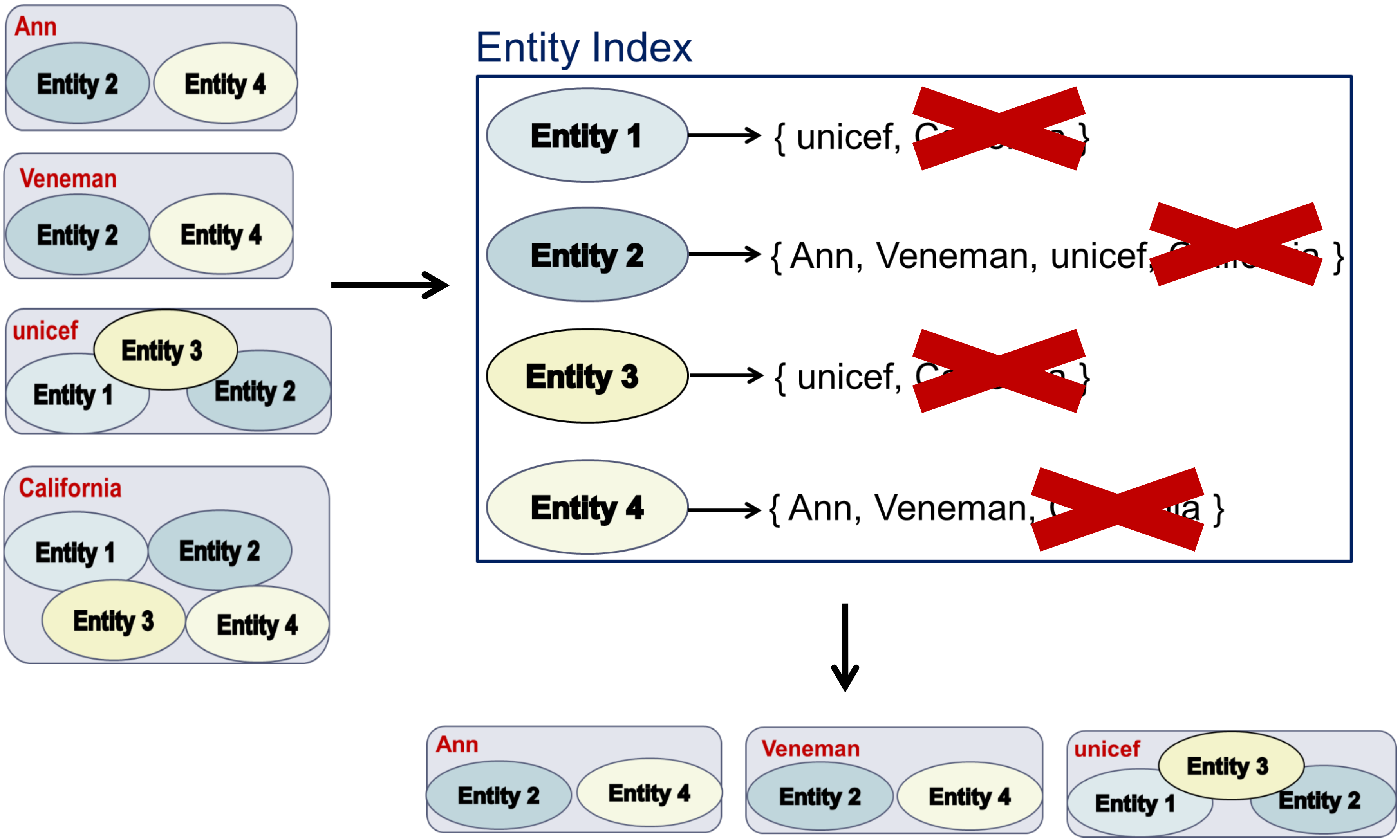
Main ideas:

- each block has a different importance for every entity it contains.
- Larger blocks are less likely to contain unique duplicates and, thus, are less important.

Algorithm

- sort blocks in ascending cardinality
- build Entity Index
- retain every entity in **r%** of its smallest blocks
- reconstruct blocks

Block Filtering Example



Block Clustering [Fisher et. al., KDD 2015]

Main idea:

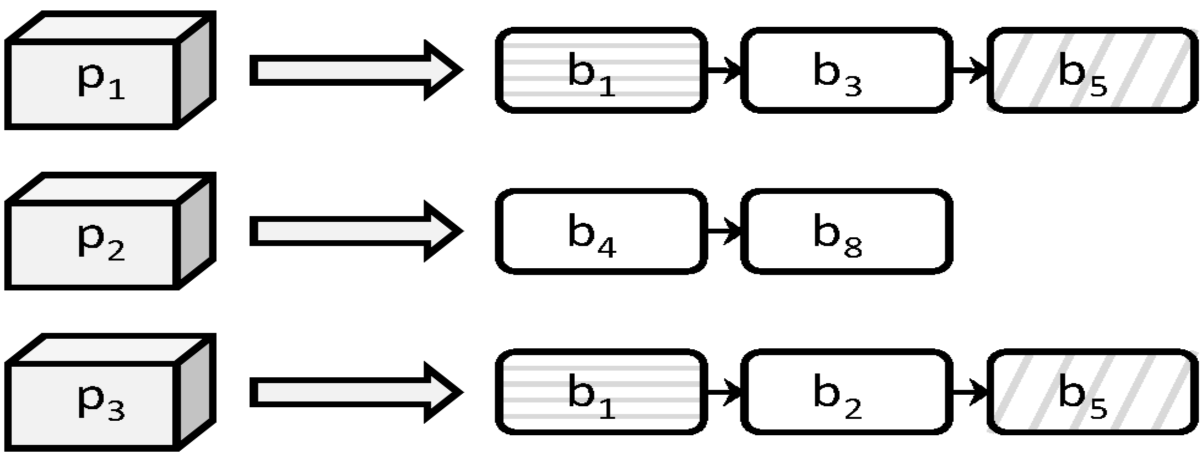
- restrict the size of every block into **[b_{min}, b_{max}]**
  - necessary in applications like privacy-preserving ER
  - operates so that ||B|| increases linearly with |E|

Algorithm

- recursive agglomerative clustering
  - merge similar blocks with size lower than b_{min}
  - split blocks with size larger than b_{max}
- until all blocks have the desired size

Comparison Propagation [Papadakis et al., JCDL 2011]

- Eliminate all redundant comparisons at no cost in recall.
- Naïve approach does not scale.
- Functionality:
  1. Build Entity Index
  2. Least Common Block Index condition.



Iterative Blocking [Whang et. Al, SIGMOD 2009]

Main idea:

integrate block processing with entity matching and reflect outcomes to subsequently processed blocks, until no new matches are detected.

Algorithm

- Put all blocks in a queue Q
- While Q is not empty
  - Get first block
  - Get matches with an ER algorithm (e.g., R-Swoosh)
    - For each new pair of duplicates $p_i \equiv p_j$
      - Merge their profiles $p'_i = p'_j =< p_i, p_j >$ and update them in all associated blocks
      - Place in Q all associated blocks that are not already in it

Part 5:

Meta-blocking

Motivation

DBPedia 3.0rc ↔ DBPedia 3.4

1.2 million entities ↔ 2.2 million entities

Motivation

DBPedia 3.0rc ↔ DBPedia 3.4

1.2 million entities ↔ 2.2 million entities

<u>Brute-force approach</u>

Comparisons: $2.58 \cdot 10^{12}$

Recall: 100%

Running time: 1,344 days → **3.7 years**

Motivation

E → Block Building → Block Cleaning → Comparison Cleaning → B

DBPedia 3.0rc ↔ DBPedia 3.4

1.2 million entities ↔ 2.2 million entities

<u>Brute-force approach</u>

Comparisons: $2.58 \cdot 10^{12}$

Recall: 100%

Running time: 1,344 days → **3.7 years**

<u>Token Blocking + Block Filtering + Comparison Propagation</u>

Overhead time: <30 mins

Comparisons: $3.5 \cdot 10^{10}$

Recall: 99%

Total Running time: **19 days**

Motivation

DBPedia 3.0rc ↔ DBPedia 3.4

1.2 million entities ↔ 2.2 million entities

Brute-force approach

Comparisons: $2.58 \cdot 10^{12}$

Recall: 100%

Running time: 1,344 days → **3.7 years**

Token Blocking + Block Filtering + Comparison Propagation

Overhead time: <30 mins

Comparisons: $3.5 \cdot 10^{10}$

Recall: 99%

Total Running time: **19 days**

Motivation

E → **Block Building** → **Block Cleaning** → **Comparison Cleaning** → **B**

DBPedia 3.0rc ↔ DBPedia 3.4

1.2 million entities ↔ 2.2 million entities

<u>Brute-force approach</u>

Comparisons: $2.58 \cdot 10^{12}$

Recall: 100%

Running time: 1,344 days → **3.7 years**

<u>Token Blocking + Block Filtering + Comparison Propagation</u>

Overhead time: <30 mins

Comparisons: $3.5 \cdot 10^{10}$

Recall: 99%

Total Running time: **19 days**

<u>Token Blocking + Block Filtering + **??**</u>

Meta-blocking [Papadakis et. al., TKDE 2014]

Goal:

restructure a **redundancy-positive** block collection into a new one that contains substantially lower number of redundant and superfluous comparisons, while maintaining the original number of matching ones ($\Delta PC \approx 0$, $\Delta PQ >> 1$) $\rightarrow$

Meta-blocking [Papadakis et. al., TKDE 2014]

Goal:

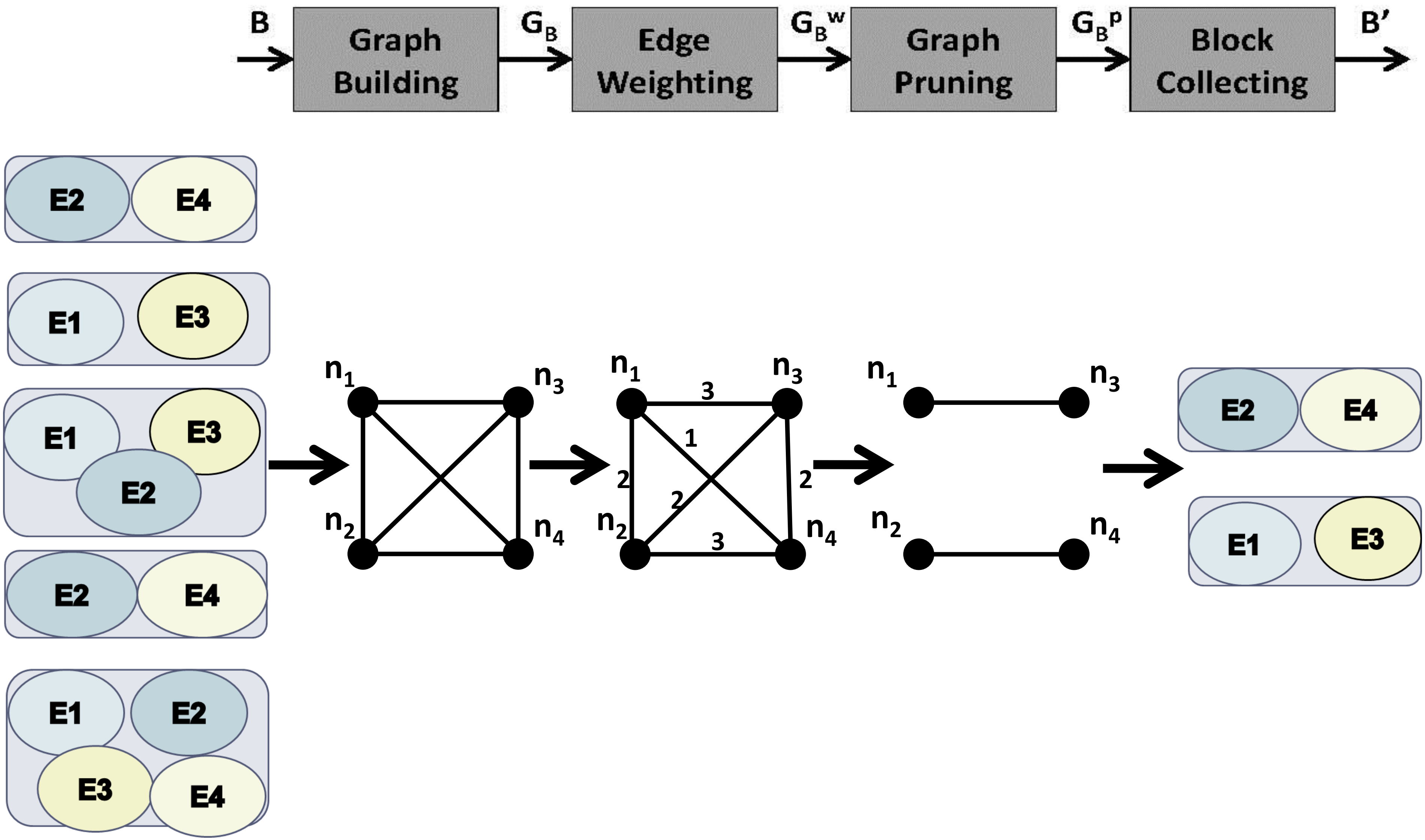
restructure a **redundancy-positive** block collection into a new one that contains substantially lower number of redundant and superfluous comparisons, while maintaining the original number of matching ones ($\Delta PC \approx 0$, $\Delta PQ >> 1$) →

Main idea:

common blocks provide valuable evidence for the similarity of entities

→ the more blocks two entities share, the more similar and the more likely they are to be matching

Outline of Meta-blocking



Graph Building

For every block:

- for every entity → add a node
- for every pair of co-occurring entities → add an undirected edge

Blocking graph:

- It eliminates all redundant comparisons →
  no parallel edges.
- Low materialization cost →
  implicit materialization through inverted indices
- Different from similarity graph!

Edge Weighting

Five generic, attribute-agnostic weighting schemes that rely on the following evidence:

- the number of blocks shared by two entities
- the size of the common blocks
- the number of blocks or comparisons involving each entity.

Computational Cost:

- In theory, equal to executing all pair-wise comparisons in the given block collection.
- In practice, significantly lower because it does not employ string similarity metrics.

Weighting Schemes

1. Aggregate Reciprocal Comparisons Scheme (ARCS)

$$w_{ij} = \sum_{b_k \in B_{ij}} \frac{1}{||b_k||}$$

2. Common Blocks Scheme (CBS)

$$w_{ij} = |B_{ij}|$$

3. Enhanced Common Blocks Scheme (ECBS)

$$w_{ij} = |B_{ij}| \cdot \log \frac{|B|}{|B_i|} \cdot \log \frac{|B|}{|B_j|}$$

4. Jaccard Scheme (JS)

$$w_{ij} = \frac{|B_{ij}|}{|B_i| + |B_j| - |B_{ij}|}$$

5. Enhanced Jaccard Scheme (EJS)

$$w_{ij} = \frac{|B_{ij}|}{|B_i| + |B_j| - |B_{ij}|} \cdot \log \frac{|V_G|}{|v_i|} \cdot \log \frac{|V_G|}{|v_j|}$$

Graph Pruning

Pruning algorithms

1. Edge-centric
2. Node-centric

they produce directed blocking graphs

Pruning criteria

Scope:

1. Global
2. Local

Functionality:

1. Weight thresholds
2. Cardinality thresholds

Edge-centric

scope	functionality: weight	functionality: cardinality
global	WEP	CEP
local	x	x

(a)

Node-centric

scope	functionality: weight	functionality: cardinality
global	x	CNP
local	WNP	CNP

(b)

Thresholds for Graph Pruning

Experiments show robust behavior of the following configurations:

1. **Weighted Edge Pruning** (WEP)
   threshold: average weight across all edges
2. **Cardinality Edge Pruning** (CEP)
   threshold: $K = BPE \cdot |E|/2$
3. **Weighted Node Pruning** (WNP)
   threshold: for each node, the average weight of the adjacent edges
4. **Cardinality Node Pruning** (CNP)
   threshold: for each node, $k=BPE-1$

Back to Motivation

DBPedia 3.0rc ↔ DBPedia 3.4

Brute-force approach

Comparisons: $2.58 \cdot 10^{12}$

Recall: 100%

Running time: 1,344 days → **3.7 years**

Token Blocking + Block Filtering + Comparison Propagation

Overhead time: <30 mins

Comparisons: $3.5 \cdot 10^{10}$

Recall: 99%

Total Running time: **19 days**

Token Blocking + Block Filtering + **Meta-blocking**

Back to Motivation

E → **Block Building** → **Block Cleaning** → **Comparison Cleaning** → **B**

DBPedia 3.0rc ↔ DBPedia 3.4

<u>Brute-force approach</u>

Comparisons: $2.58 \cdot 10^{12}$
Recall: 100%
Running time: 1,344 days → **3.7 years**

<u>Token Blocking + Block Filtering + Comparison Propagation</u>

Overhead time: <30 mins
Comparisons: $3.5 \cdot 10^{10}$
Recall: 99%
Total Running time: **19 days**

<u>Token Blocking + Block Filtering + **Meta-blocking**</u>

Overhead time: 4 hours
Comparisons: $8.95 \cdot 10^{6}$
Recall: 99%
Total Running time: **10 hours**

Back to Motivation

E → **Block Building** → **Block Cleaning** → **Comparison Cleaning** → **B**

DBPedia 3.0rc ↔ DBPedia 3.4

Brute-force approach

Comparisons: $2.58 \cdot 10^{12}$

Recall: 100%

Running time: 1,344 days → **3.7 years**

Token Blocking + Block Filtering + Comparison Propagation

Overhead time: <30 mins

Comparisons: $3.5 \cdot 10^{10}$

Recall: 99%

Total Running time: **19 days**

Token Blocking + Block Filtering + **Meta-blocking**

Overhead time: 4 hours

Comparisons: $8.95 \cdot 10^{6}$

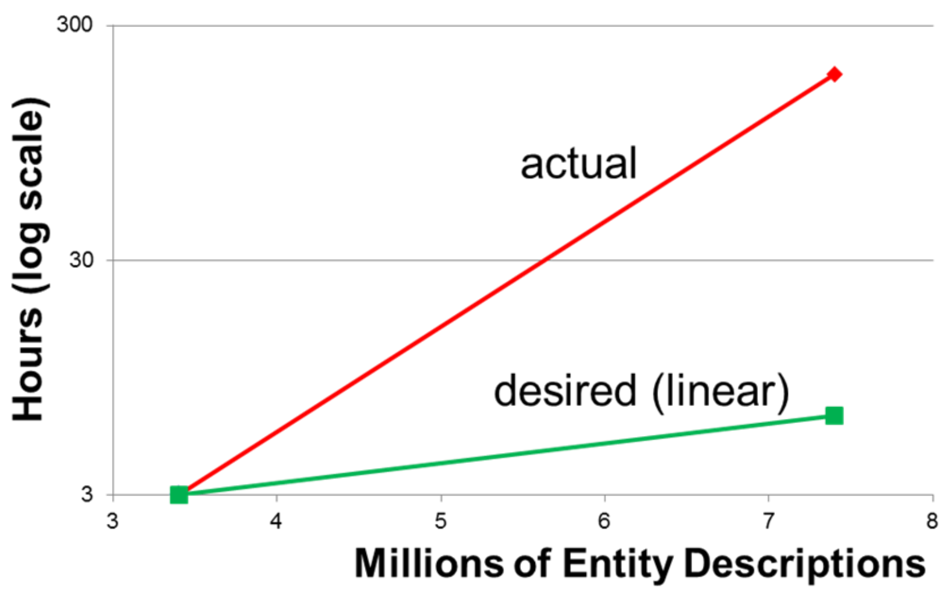
Recall: 99%

Total Running time: **10 hours**

Meta-blocking Challenges: Time Efficiency

Bottleneck: edge weighting

- Depends on $||B||$ & BPE
  - $|E| = 3.4 \times 10^6$, $||B|| = 4 \times 10^{10}$, BPE=15 → 3 hours
  - $|E| = 7.4 \times 10^6$, $||B|| = 2 \times 10^{11}$, BPE=40 → 186 hours



Enhancing Meta-blocking Time Efficiency

1. Block Filtering
   - r = 0.8 → 4 times faster processing, on average
   - reduces both ||B|| and BPE
2. Optimized Edge Weighting [Papadakis et. al., EDBT 2016]
   - Entity-based instead of Block-based implementation
   - An order of magnitude faster processing, in combination with Block Filtering
3. Multi-core Meta-blocking
   - Commodity hardware
4. Parallel Meta-blocking
   - Hadoop Cluster

Multi-core Meta-blocking [Papadakis et. al, Semantics 2017]

Two types of methods:

- Block-based
- Entity-based

Fork-join approach:

- computational cost split into set of chunks* placed in an array, with an index indicating next chunk to be processed
- Every thread retrieves current value of index and assigned to process corresponding chunk

chunk = individual items or a non-overlapping set of items

*item = an individual block or an individual entity

Parallelization Strategies

Depending on the definition of chunks, we defined the following parallelization strategies:

1. Random parallelization → individual items in arbitrary order
2. Naïve Parallelization → individual items sorted by cost (#comparisons)
3. Partition Parallelization → an arbitrary number of non-overlapping groups of items with the same computational cost
4. Segment Parallelization → #cores non-overlapping groups of items with the same computational cost

Segment Parallelization

Input:

- **I = { 3, 5, 10, 6, 1, 9, 2, 4, 7, 8 }**
- **number of segments (4)**

I_{sorted} = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 }

Segment Parallelization

Input:

- **I = { 3, 5, 10, 6, 1, 9, 2, 4, 7, 8 }**
- **number of segments (4)**

I_{sorted} = { $\boxed{10,}$ 9, 8, 7, 6, 5, 4, 3, 2, 1 }

Iterations	It1	Total Cost
Segment1	**10**	10

Segment Parallelization

Input:

- **I = { 3, 5, 10, 6, 1, 9, 2, 4, 7, 8 }**
- **number of segments (4)**

$\mathbf{I_{sorted}} = \{\ \mathbf{10}, \boxed{\mathbf{9},}\ \mathbf{8, 7, 6, 5, 4, 3, 2, 1}\ \}$

Iterations	It1	It2	Total Cost
Segment1	**10**	10	10
Segment2		**9**	9

Segment Parallelization

Input:

- **I = { 3, 5, 10, 6, 1, 9, 2, 4, 7, 8 }**
- **number of segments (4)**

I_{sorted} **= { 10, 9$\boxed{\text{, 8,}}$7, 6, 5, 4, 3, 2, 1 }**

Iterations	It1	It2	It3	Total Cost
Segment1	**10**	10	10	10
Segment2		**9**	9	9
Segment3			**8**	8

Segment Parallelization

Input:

- **I = { 3, 5, 10, 6, 1, 9, 2, 4, 7, 8 }**
- **number of segments (4)**

$\mathbf{I_{sorted} = \{ 10, 9, 8\boxed{, 7,}6, 5, 4, 3, 2, 1 \}}$

Iterations	It1	It2	It3	It4	Total Cost
Segment1	**10**	10	10	10	10
Segment2		**9**	9	9	9
Segment3			**8**	8	8
Segment4				**7**	7

Segment Parallelization

Input:

- **I = { 3, 5, 10, 6, 1, 9, 2, 4, 7, 8 }**
- **number of segments (4)**

I_{sorted} = { 10, 9, 8, 7, $\boxed{6,}$ 5, 4, 3, 2, 1 }

Iterations	It1	It2	It3	It4	It5	Total Cost
Segment1	**10**	10	10	10	10	10
Segment2		**9**	9	9	9	9
Segment3			**8**	8	**8**	8
Segment4				**7**	6 | 7	13

Segment Parallelization

Input:

- **I = { 3, 5, 10, 6, 1, 9, 2, 4, 7, 8 }**
- **number of segments (4)**

$\mathbf{I_{sorted} = \{ 10, 9, 8, 7, 6, \boxed{5,} 4, 3, 2, 1 \}}$

Iterations	It1	It2	It3	It4	It5	It6	Total Cost
Segment1	**10**	10	10	10	10	10	10
Segment2		**9**	9	9	9	**9**	9
Segment3			**8**	8	**8**	**5** | 8	13
Segment4				**7**	6 | 7	6 | 7	13

Segment Parallelization

Input:

- **I = { 3, 5, 10, 6, 1, 9, 2, 4, 7, 8 }**
- **number of segments (4)**

$$I_{sorted} = \{ 10, 9, 8, 7, 6, 5, \boxed{4,}\ 3, 2, 1 \}$$

Iterations	It1	It2	It3	It4	It5	It6	It7	Total Cost
Segment1	**10**	10	10	10	10	10	**10**	10
Segment2		**9**	9	9	9	**9**	**4** | 9	13
Segment3			**8**	8	**8**	5 | 8	5 | 8	13
Segment4				**7**	6 | 7	6 | 7	6 | 7	13

Segment Parallelization

Input:

- **I = { 3, 5, 10, 6, 1, 9, 2, 4, 7, 8 }**
- **number of segments (4)**

$\mathbf{I_{sorted}}$ **= { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 }**

Iterations	It1	It2	It3	It4	It5	It6	It7	It8	Total Cost
Segment1	**10**	10	10	10	10	10	**10**	**3** 10	13
Segment2		**9**	9	9	9	**9**	4 9	4 9	13
Segment3			**8**	8	**8**	5 8	5 8	5 8	13
Segment4				**7**	6 7	6 7	6 7	**6** **7**	13

Segment Parallelization

Input:

- **I = { 3, 5, 10, 6, 1, 9, 2, 4, 7, 8 }**
- **number of segments (4)**

I_{sorted} **= { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 }**

Iterations	It1	It2	It3	It4	It5	It6	It7	It8	It9	Total Cost
Segment1	**10**	10	10	10	10	10	**10**	3 | 10	3 | 10	13
Segment2		**9**	9	9	9	**9**	4 | 9	4 | 9	4 | 9	13
Segment3			**8**	8	**8**	5 | 8	5 | 8	5 | 8	**5** | **8**	13
Segment4				**7**	6 | 7	6 | 7	6 | 7	**6** | **7**	6 | **2** | 7	15

Segment Parallelization

Input:

- **I = { 3, 5, 10, 6, 1, 9, 2, 4, 7, 8 }**
- **number of segments (4)**

$\mathbf{I_{sorted} = \{ 10, 9, 8, 7, 6, 5, 4, 3, 2, \boxed{1} \}}$

Iterations	It1	It2	It3	It4	It5	It6	It7	It8	It9	It10	Total Cost
Segment1	**10**	10	10	10	10	10	**10**	3 | 10	3 | 10	3 | 10	13
Segment2		**9**	9	9	9	**9**	4 | 9	4 | 9	4 | 9	**4** | **9**	13
Segment3			**8**	8	**8**	5 | 8	5 | 8	5 | 8	**5** | **8**	5 | **1** | 8	14
Segment4				**7**	6 | 7	6 | 7	6 | 7	**6** | **7**	6 | 2 | 7	6 | 2 | 7	13

Time complexity: **O(n log n)**

Execution Plan

Stage 1	MWEP	MWNP	MCEP	MCNP
Initialization	Initialize chunk array and **N** threads.	Initialize chunk array and **N** threads.	Initialize chunk array and **N** threads. Estimate ***K***.	Initialize chunk array and **N** threads. Estimate ***k***.
th_1 th_2 th_3 … th_N	Each thread computes local aggregate edge weight and #comparisons	Each thread stores the total weight and #comparisons per entity in two arrays	Each thread stores the ***K*** top-weighted edges in the processed chunks in a priority queue	Every thread stores the ***k*** top-weighted edges for every processed entity in a priority queue
Merge	Estimate average edge weight	Merge the 2 arrays to compute the average edge weight of each node	**Output** the overall ***K*** top-weighted comparisons	**Output** the comparisons that are among the ***k*** top-weighted ones for any of the adjacent entities
Stage 2				
th_1 th_2 th_3 … th_N	Check and keep valid comparisons above the weight threshold	Check and keep valid comparisons above the weight threshold of any adjacent node		
Total valid comparisons	**Output** total valid comparisons	**Output** total valid comparisons		

Experimental Evaluation - Datasets

Original Datasets	DBPedia 3.0rc	DBPedia 3.4
Entities	1,190,733	2,164,040
Duplicates	892,579	
Triples	$1.69 \cdot 10^7$	$3.50 \cdot 10^7$
Predicates	30,757	52,554
Brute-force	**$2.58 \cdot 10^{12}$**	

Blocks	Input	Output (CNP)
Blocks	1,239,066	1,190,733
Comparisons	**$1.30 \cdot 10^{10}$**	**$3.30 \cdot 10^7$**
Detected Matches	890,817	859,554
Recall	0.998	0.963
Precision	$6.86 \cdot 10^{-5}$	$2.61 \cdot 10^{-2}$

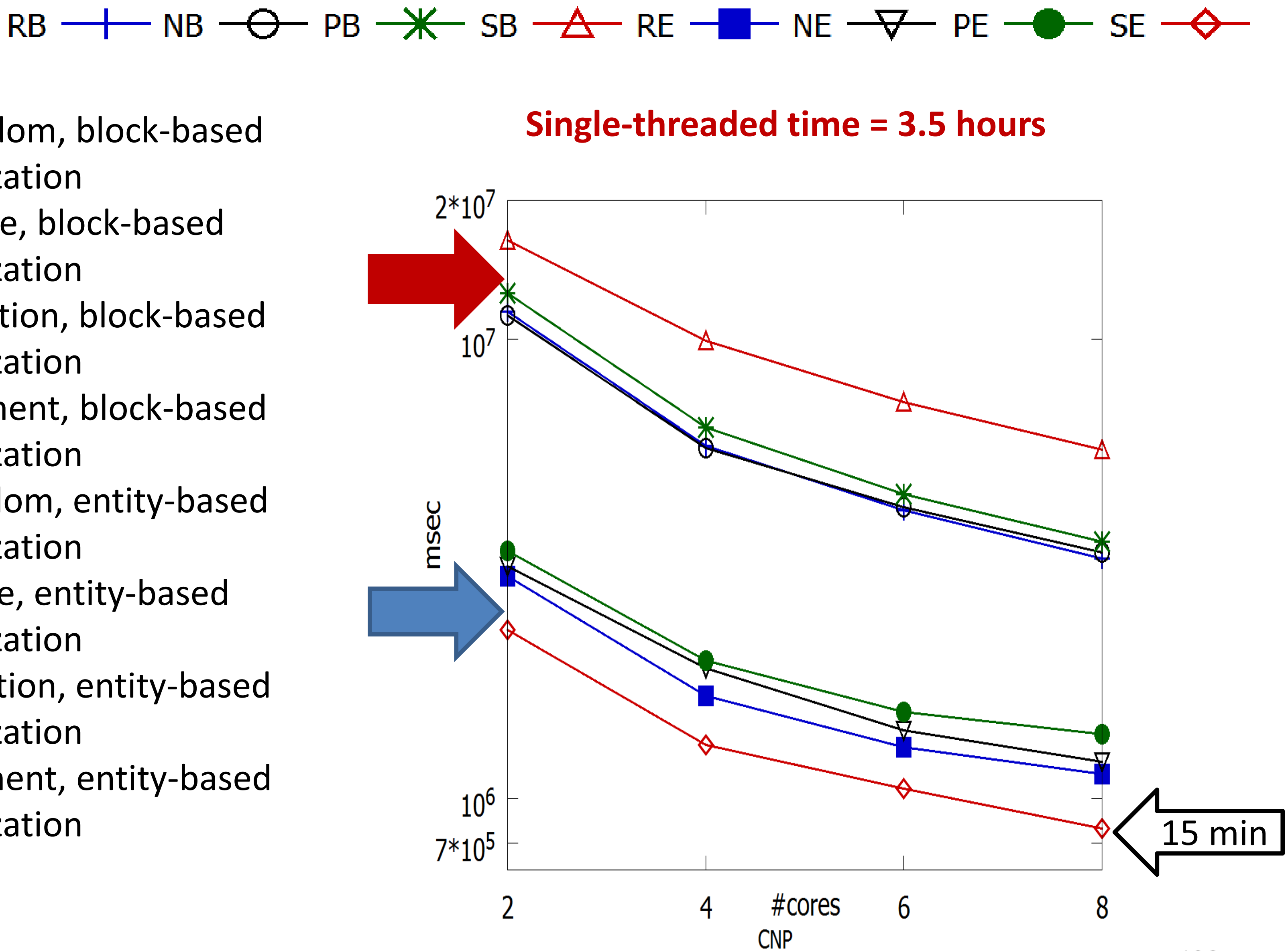
token blocking + block purging + block filtering

token blocking + block purging + block filtering + meta-blocking CNP

System: Server running Ubuntu 12.04, 32GB RAM and 2 Intel Xeon E5620 processors, each having 4 physical cores and 8 logical cores at 2.40GHz.

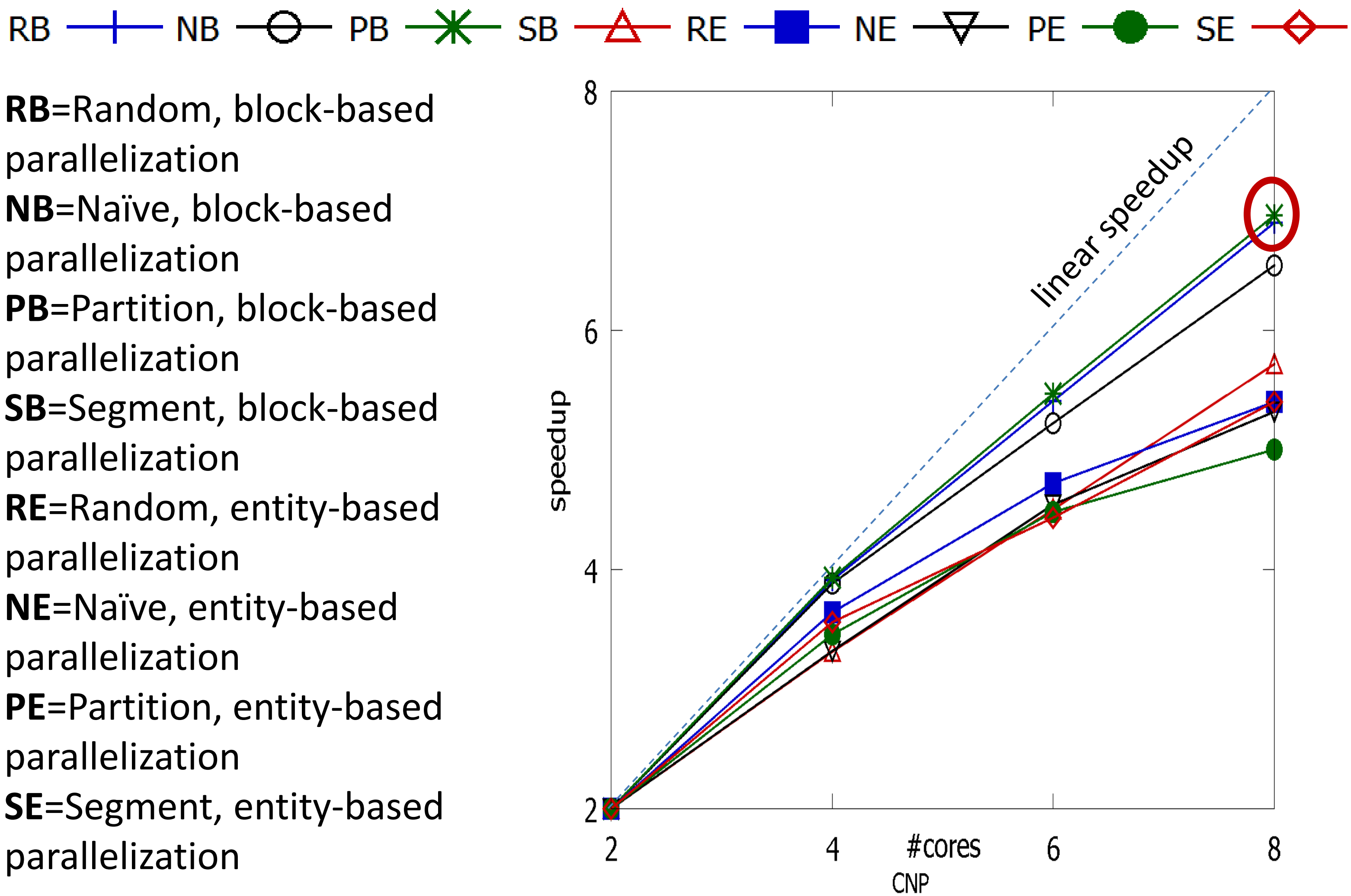
Experimental Evaluation – **CNP** Wall Clock Time

RB=Random, block-based parallelization
NB=Naïve, block-based parallelization
PB=Partition, block-based parallelization
SB=Segment, block-based parallelization
RE=Random, entity-based parallelization
NE=Naïve, entity-based parallelization
PE=Partition, entity-based parallelization
SE=Segment, entity-based parallelization



Experimental Evaluation – CNP Speedup

RB=Random, block-based parallelization
NB=Naïve, block-based parallelization
PB=Partition, block-based parallelization
SB=Segment, block-based parallelization
RE=Random, entity-based parallelization
NE=Naïve, entity-based parallelization
PE=Partition, entity-based parallelization
SE=Segment, entity-based parallelization



Meta-blocking Challenges: Effectiveness

Problem:

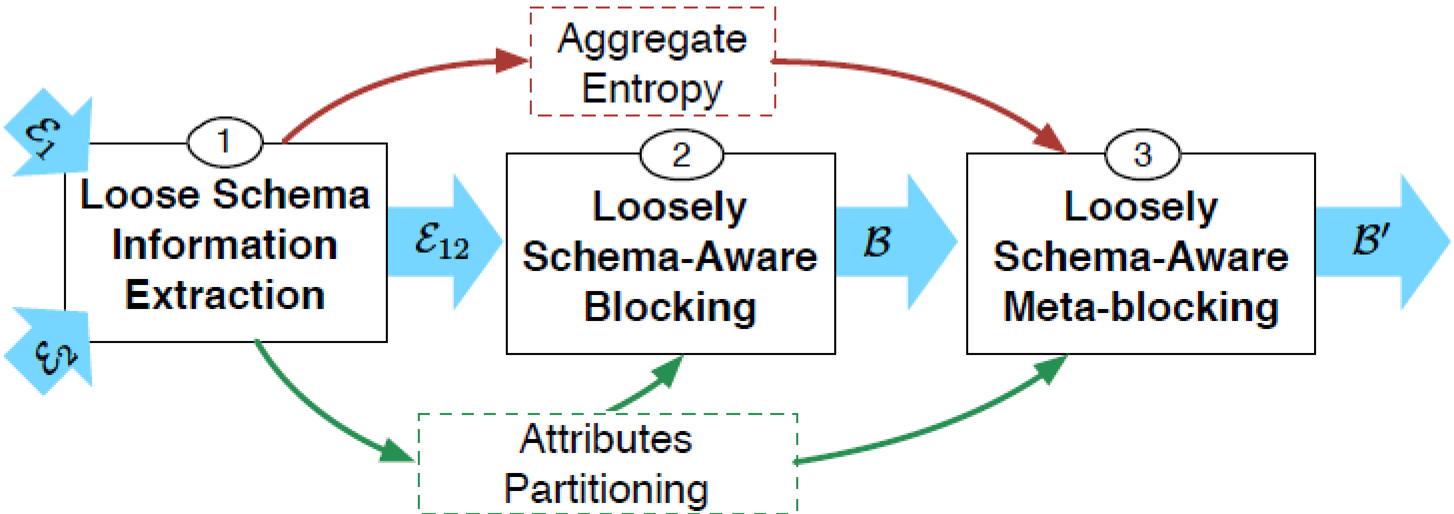
- Simple pruning rules

Solutions:

- Unsupervised methods
  - BLAST
    - Integrates schema information
- Supervised methods
  - Supervised Meta-blocking (SMB)
    - utilizes feature-based classification of blocking-graph edges
  - BLOSS
    - minimizes the size of the required training set

BLAST [Simonini et. al., VLDB 2017]

- Goal:
  improve the edge weighting and pruning in unsupervised WNP with loose schema information
- Solution:



It works for Dirty ER, as well.

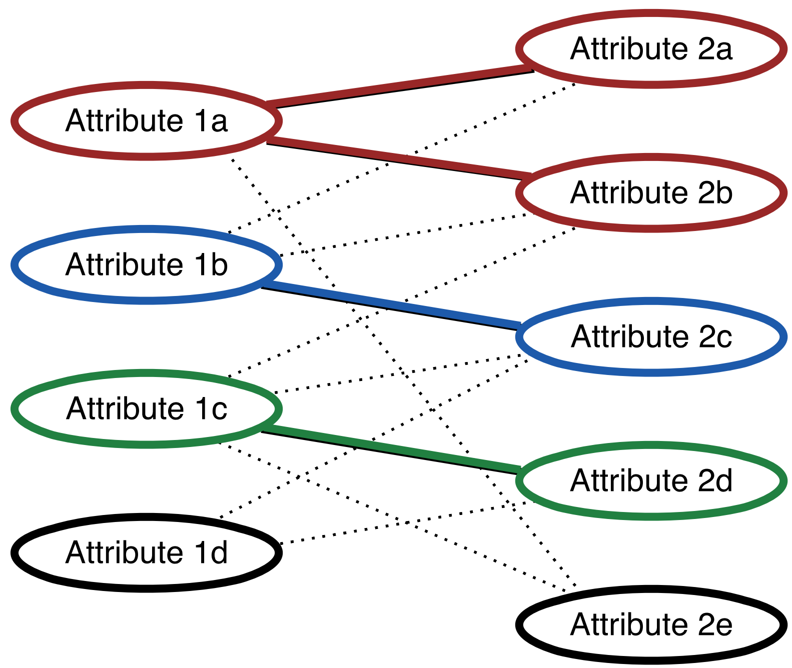
Courtesy of Giovanni Simonini

Loose Schema Information Extraction

Similar to Attribute Clustering:

1. Each attribute is represented as the set of its possible values
2. Builds a (bipartite*) graph with one node for every attribute
3. There is an edge for every pair of attributes with similarity > 0
4. Each connected component is an attribute cluster

* In the case of Clean-Clean ER.



The original Attribute Clustering does not scale to **thousands of attributes** → **very inefficient**

BLAST employs **LSH** to reduce the time complexity (for JaccardSim)

- **Scales well to hundred of thousands attributes**
- **Simultaneously estimates aggregate entropy per cluster**

Courtesy of Giovanni Simonini

Loosely Schema-aware Meta-blocking

1. BLAST improves **edge weighting** as follows:
   every edge → several blocking keys (tokens) → multiple attribute names → $w(e_{ij})$=aggregate entropy·Pearson's χ^2

$$w_{uv} = \chi^2_{uv} \cdot h(\mathcal{B}_{uv}) = \sum_{i \in \{1,2\}} \sum_{j \in \{1,2\}} \frac{o_{ij} - e_{ij}}{e_{ij}} \cdot h(\mathcal{B}_{uv})$$

2. BLAST improves **edge pruning** in two ways:
   - Local weight threshold independent of the size of each node neighborhood (i.e., number of edges):

$$\theta_i = \frac{argmax_i \, w(e_{ij})}{2}$$

   - An edge e_{ij} is retained if $w(e_{ij}) \geq \frac{\theta_i + \theta_j}{2}$.

Courtesy of Giovanni Simonini

Example – Original Meta-blocking

Entity Collection

p1
Name: John Abram Jr
profession: car seller
year: 1985
Addr.: Main street

p3
name: Jon Abram Jr
birth year: 85
job: car retail
Loc: Main st.

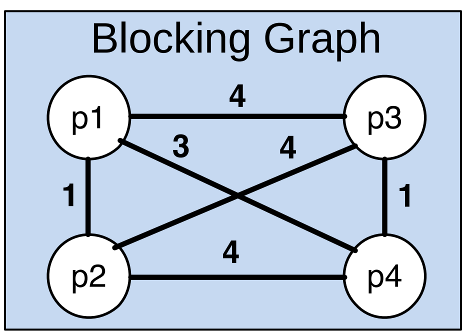
p2
FirstName: Ellen
SecondName: Smith
year: 85
occupation: retail
mail: Abram st. 30 NY

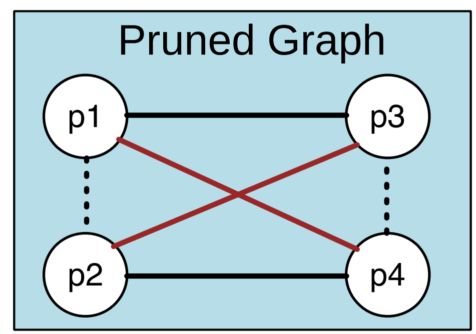
p4
full name: Ellen Smith
b. date: May 10 1985
work info: retailer
loc: Abram street NY

Token Blocking

Ellen	**NY**	**Main**	**1985**	**car**	**Smith**
p2 p4	p2 p4	p1 p3	p1 p4	p1 p3	p2 p4

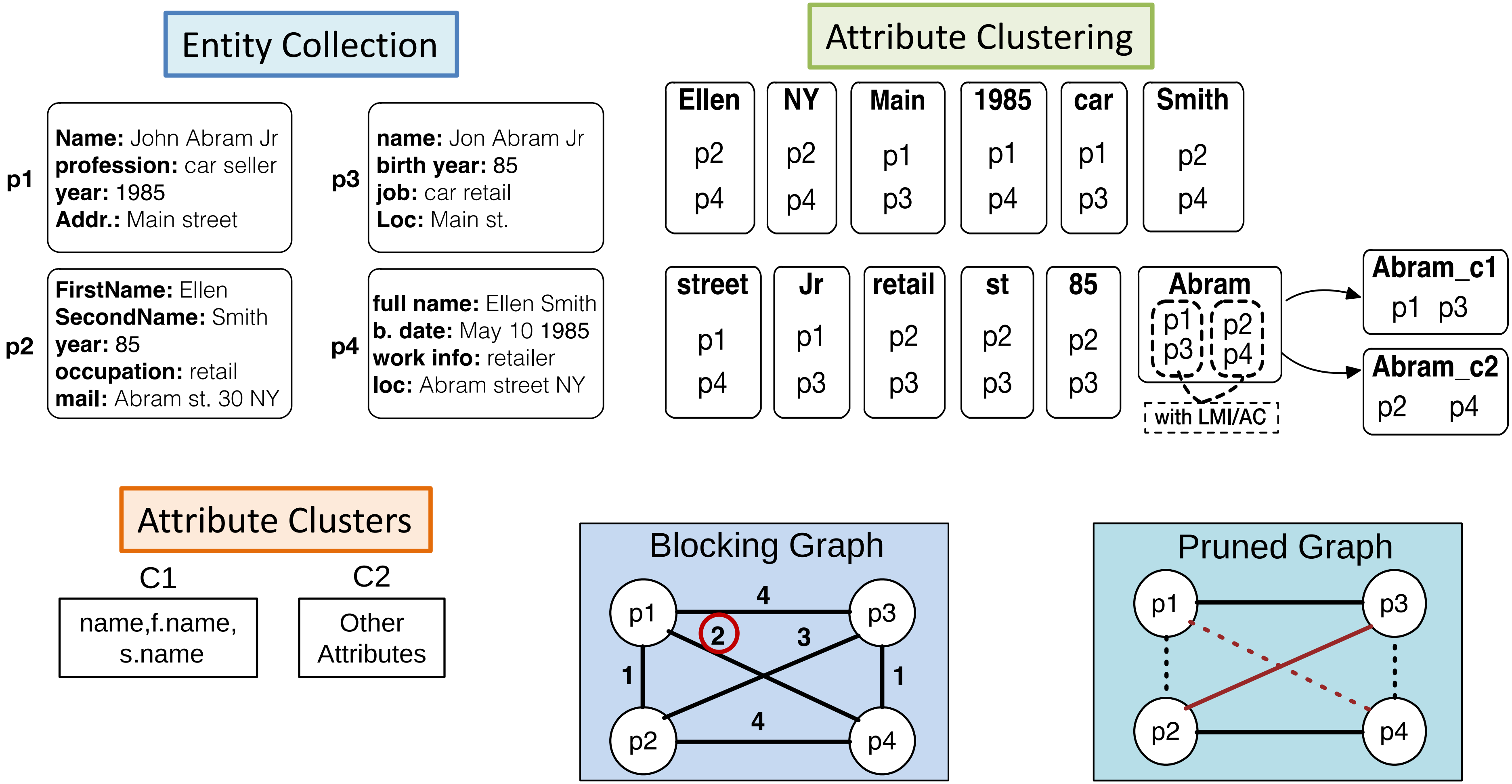
street	**Jr**	**retail**	**st**	**85**	**Abram**
p1 p4	p1 p3	p2 p3	p2 p3	p2 p3	p1 p2 p3 p4





Courtesy of Giovanni Simonini

Example – Meta-blocking over Attribute Clustering



Courtesy of Giovanni Simonini

Example - BLAST

Entity Collection

p1
Name: John Abram Jr
profession: car seller
year: 1985
Addr.: Main street

p3
name: Jon Abram Jr
birth year: 85
job: car retail
Loc: Main st.

p2
FirstName: Ellen
SecondName: Smith
year: 85
occupation: retail
mail: Abram st. 30 NY

p4
full name: Ellen Smith
b. date: May 10 1985
work info: retailer
loc: Abram street NY

Attribute Clustering

Ellen	**NY**	**Main**	**1985**	**car**	**Smith**
p2 p4	p2 p4	p1 p3	p1 p4	p1 p3	p2 p4

street	**Jr**	**retail**	**st**	**85**	**Abram**
p1 p4	p1 p3	p2 p3	p2 p3	p2 p3	p1 p3 / p2 p4

with LMI/AC

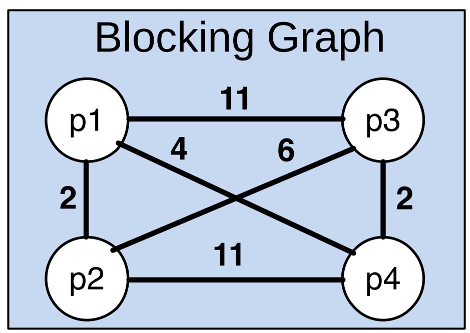
Abram_c1	**Abram_c2**
p1 p3	p2 p4

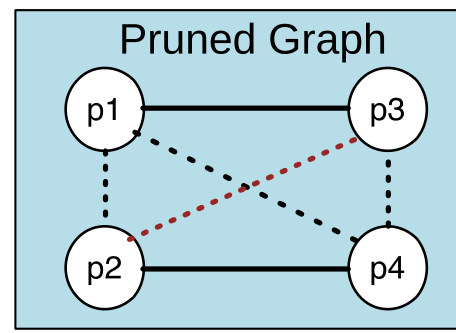
Attribute Clusters

C1	C2
name,f.name, s.name	Other Attributes

Loose Schema Info

Entropy cluster1 (name) = **3.5**
Entropy cluster2 (other atr.)= 2.0





Courtesy of Giovanni Simonini

Supervised Meta-blocking [Papadakis et. al., VLDB 2014]

Goal:

more accurate and comprehensive methodology for pruning the edges of the blocking graph

Solution:

- model edge pruning as a classification task per edge
- two classes: “likely match”, “unlikely match”

Open issues:

- Classification Features
- Training Set
- Classification Algorithms & Configuration

Classification Features

Requirements:

1. Generic   2. Effective
3. Efficient   4. Minimal

- **schema-awareness**
  - schema-based
  - schema-agnostic
  - hybrid
- **source of evidence**
  - block-based
  - graph-based
    - iterative
  - hybrid
- **target**
  - edge-specific
  - node-specific
  - hybrid
- **complexity**
  - raw
  - derived
- **scope**
  - local
  - global
  - hybrid

Feature Engineering

	source of evidence			target			complexity		scope		
	block-based	graph-based	iterative	edge-specific	node-specific	hybrid	raw	derived	local	global	hybrid
CF_IBF	✓					✓		✓			✓
Jaccard_Sim	✓			✓				✓	✓		
RACCB	✓			✓				✓	✓		
Node_Degree		✓			✓		✓		✓		
Iterative_Degree			✓		✓			✓		✓	
Transitive_Degree		✓			✓			✓		✓	

CF-IBF = # of Common Blocks × Inverse Block Frequency per entity
RACCB = Sum of Inverse Block Sizes

We examined all 63 possible combinations to find the minimal set of features, which comprises the first four features.

Training Set

Challenge:

binary classification with heavily imbalanced classes

Solutions:

1. Oversampling
2. Cost-sensitive learning
3. Ensemble learning
4. **Undersampling**
   - Sample size equal to 5% of the minority class.

Classification Algorithms

Weighted Edge Pruning (**WEP**)

- compatible with any classifier
- we selected 4 state-of-the-art:
    1. Naïve Bayes
    2. Bayesian Networks
    3. C4.5 Decision Trees
    4. Support Vector Machines

Configuration

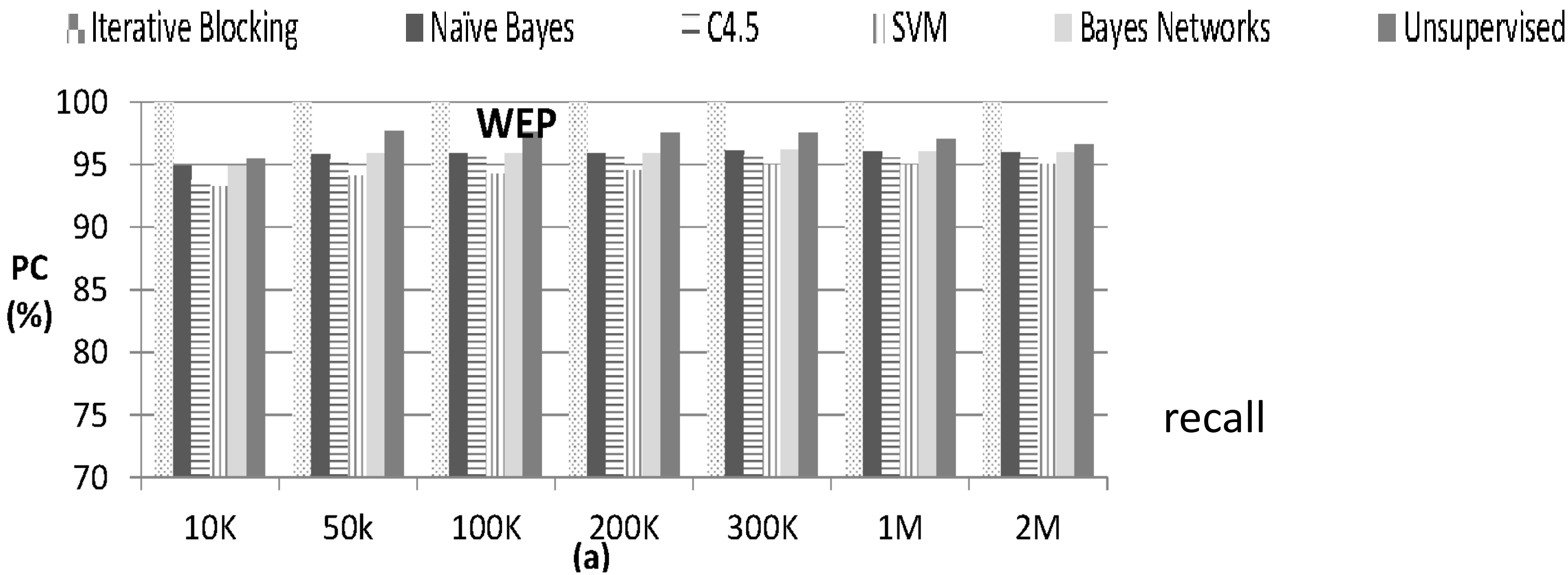
For selected features and sample size, classifiers are robust with respect to their internal parameters

Cardinality Edge Pruning (**CEP**) &

Cardinality Node Pruning (**CNP**)

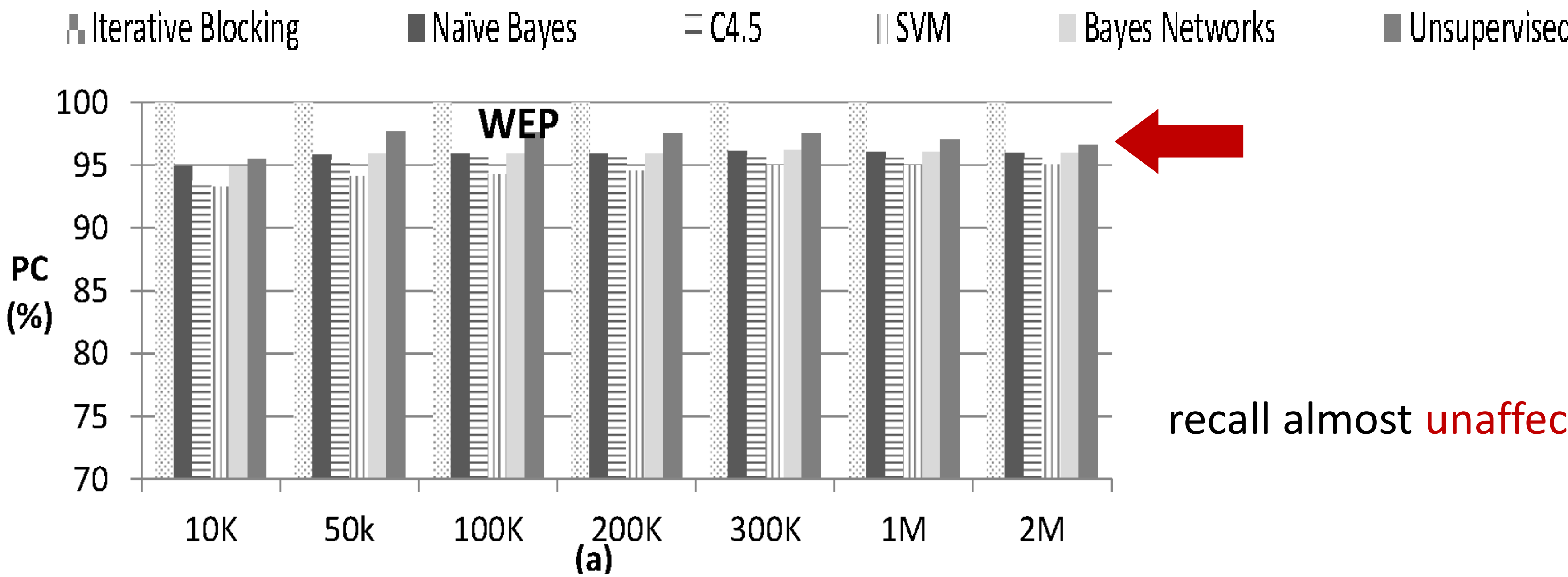
- compatible with probabilistic classifiers
- we selected Naïve Bayes, Bayesian Networks

Experiments



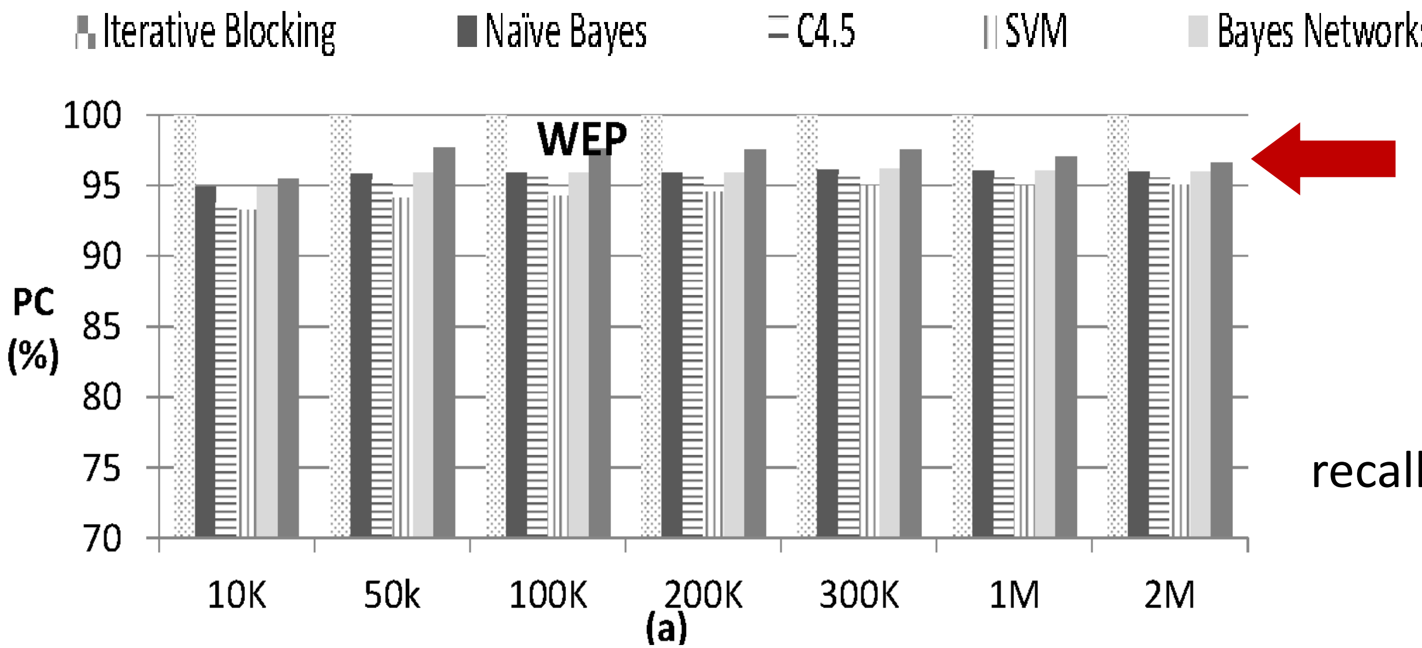
recall

Experiments

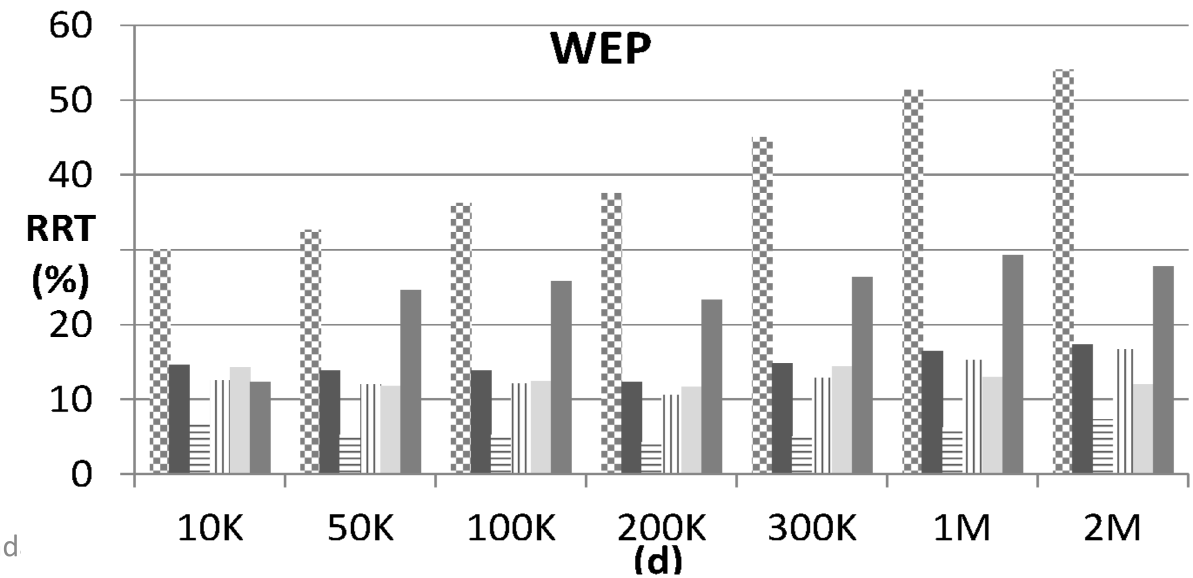


recall almost unaffected

Experiments

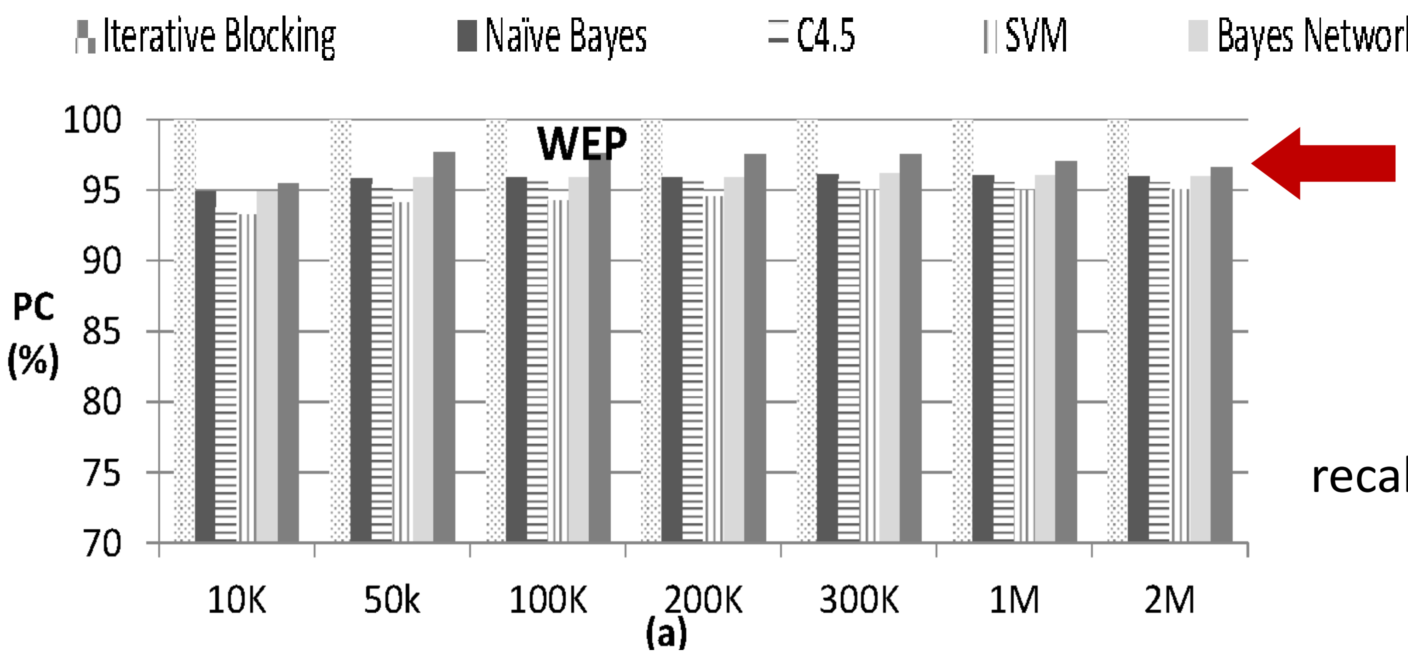


recall almost unaffected

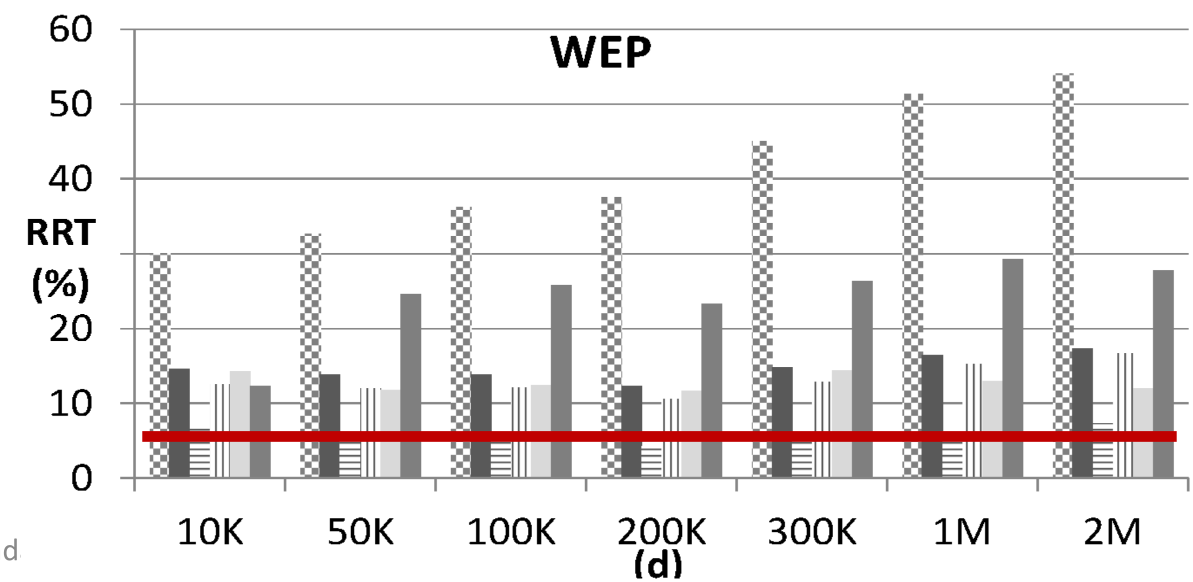


running time

Experiments



recall almost unaffected



running time up to 6x faster
(than original meta-blocking)

BLOSS [Dal Bianco et al., Information Systems, 2018]

Goal:

minimize the labelling effort
for training Supervised Meta-blocking

Solution:

meta-**BLO**cking **S**ampling **S**election

- involves a novel sampling methodology

- combines it with active learning

Key feature:

- removes outliers to improve recall

Courtesy of Guilherme Dal Bianco

BLOSS Outline

It comprises three steps:

1. Definition of Similarity Levels and Random Sampling
   – pre-selects candidate pairs using a metric that assesses the potential of a pair being a match
   – level-based sampling ensures diversity

2. Selection of Pairs for Labeling
   – applies active learning applied to the pre-selected pairs

3. Pruning Non-Matching Outliers
   – filters out noisy non-matching pairs that have been labeled

Courtesy of Guilherme Dal Bianco

BLOSS First Stage

Relies on **CF_IBF**, which is proportional to matching likelihood.

Group candidate pairs into **L** layers

Random selection **N** pairs inside each level

CF_IBF	Entity 1	Entity 2
30	**A** B C D	**A** E F G
30	C T **U** Z	E W **U** V
80	**A** I V **X**	**A** E P **X**
80	L **M** Q **X**	S **M** P **X**
160	**D E F G**	**D E F G**
160	**A L W X**	**A L W X**

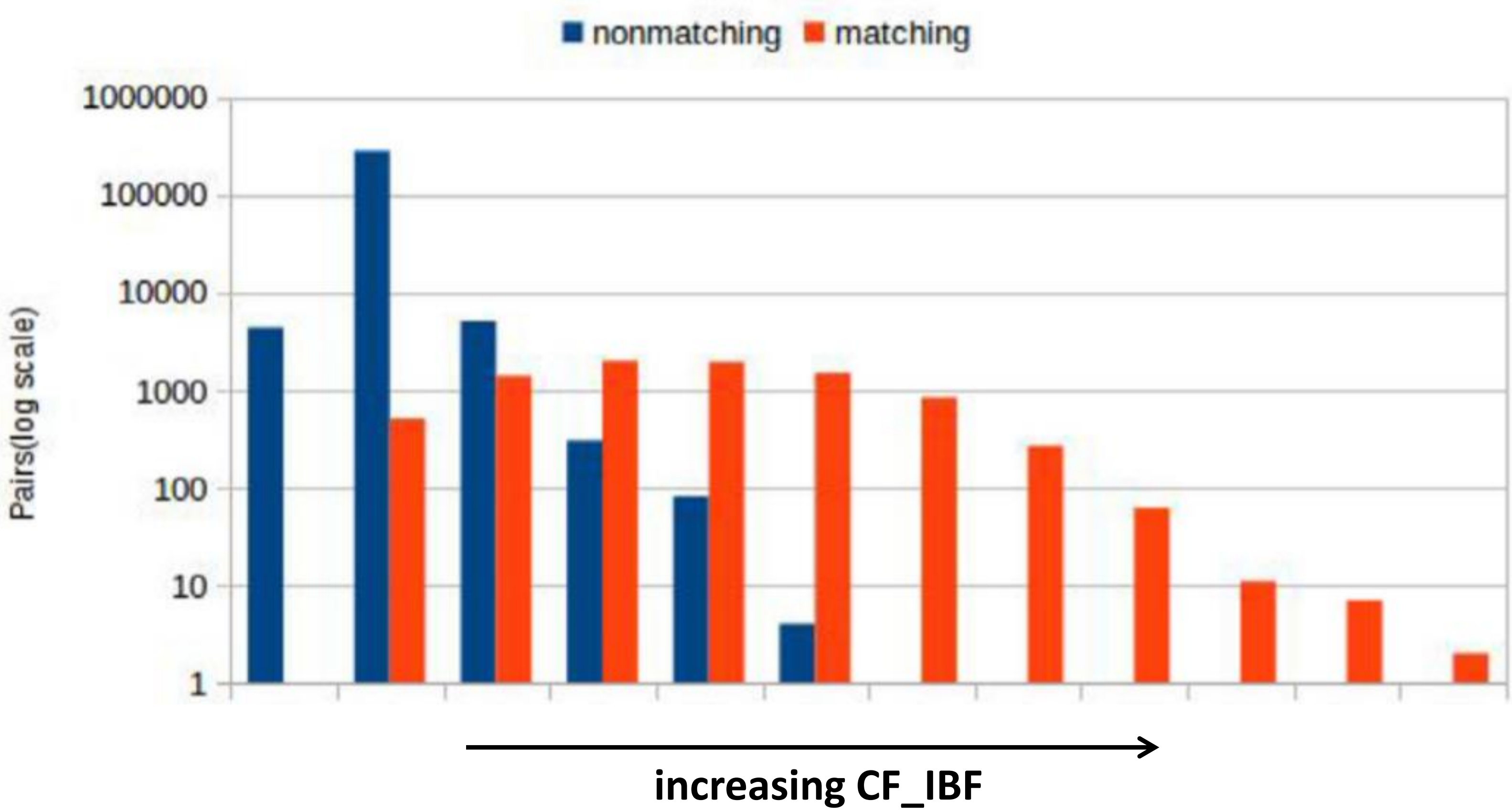
Level 0

Level 1

Level L

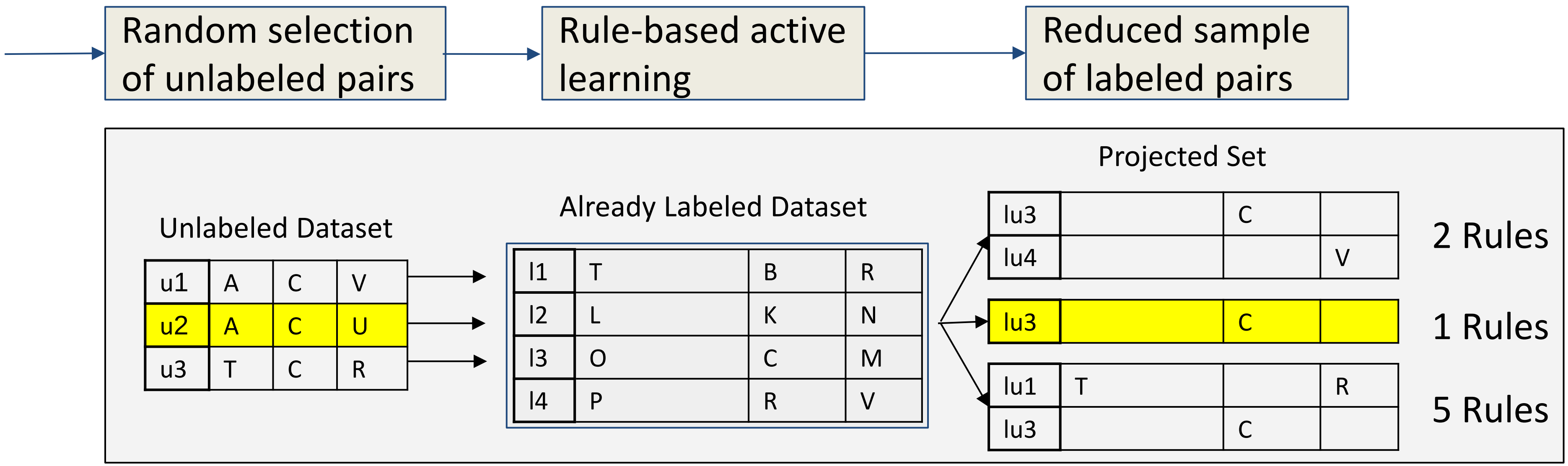
CF_IBF	Entity 1	Entity 2
30	C T **U** Z	E W **U** V
80	**A** I V **X**	**A** E P **X**
160	**A L W X**	**A L W X**

BLOSS First Stage – Layers distribution



- first levels probably include more non-matching pairs
- last levels group more matching pairs

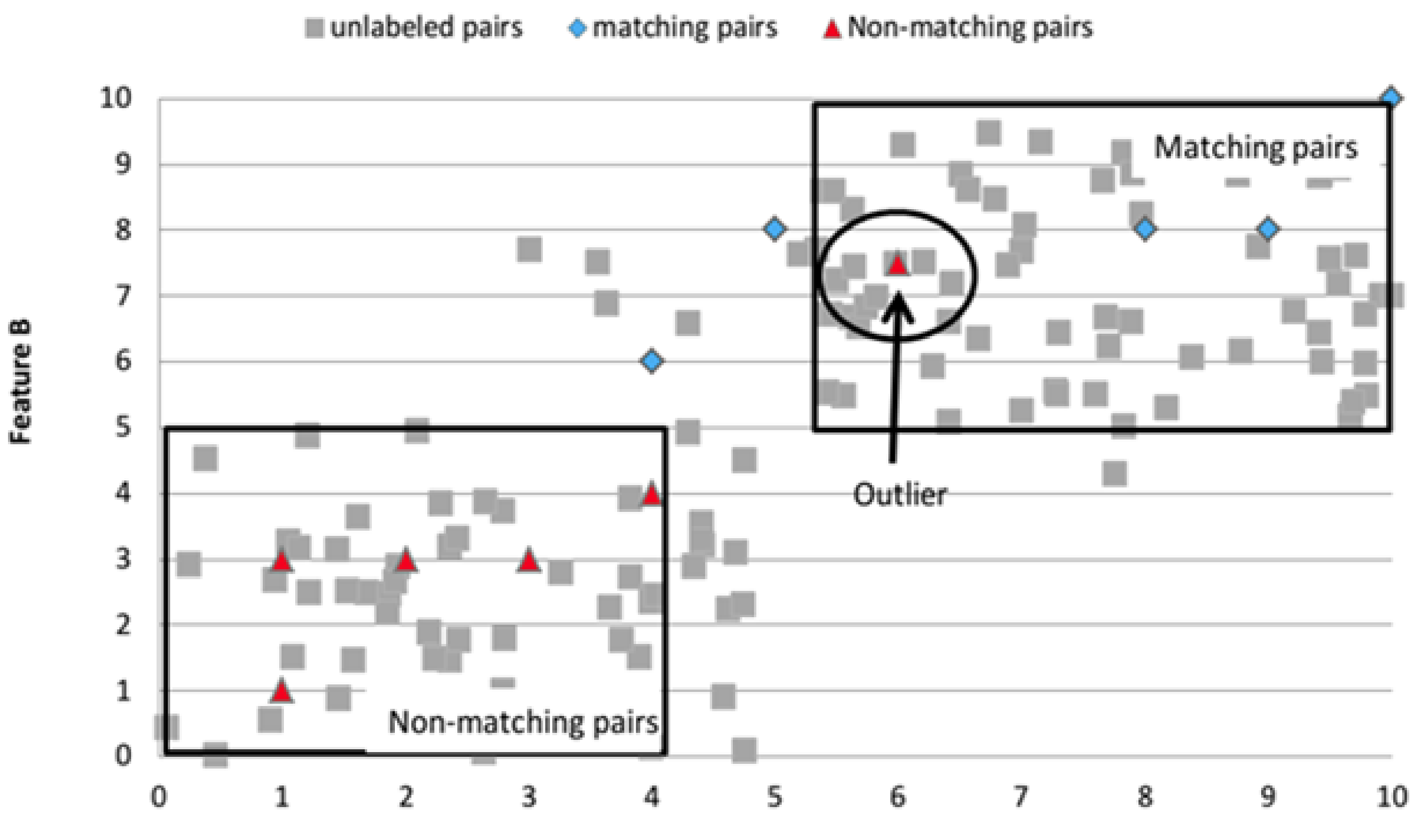
BLOSS Second Stage: Pairs Selection



Pair u2 is the most dissimilar instance compared to the actual training set.

BLOSS Third Stage: Outliers Pruning

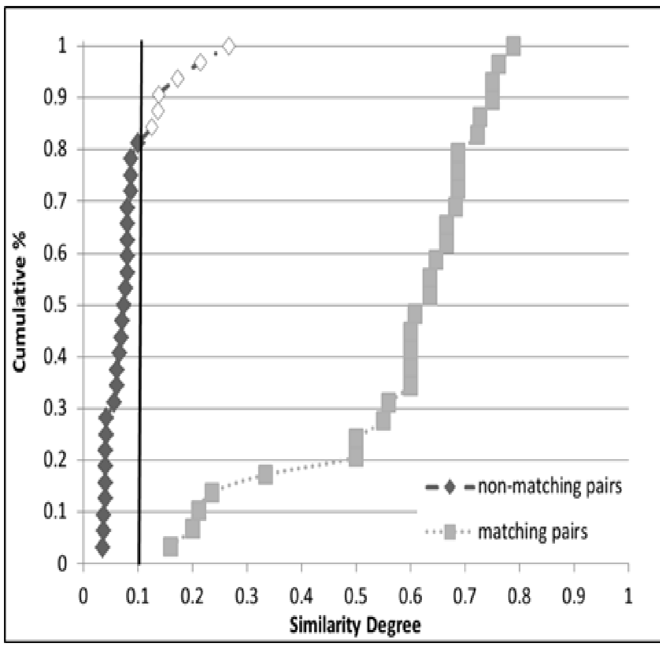
Goal: remove noise, while maximizing recall.



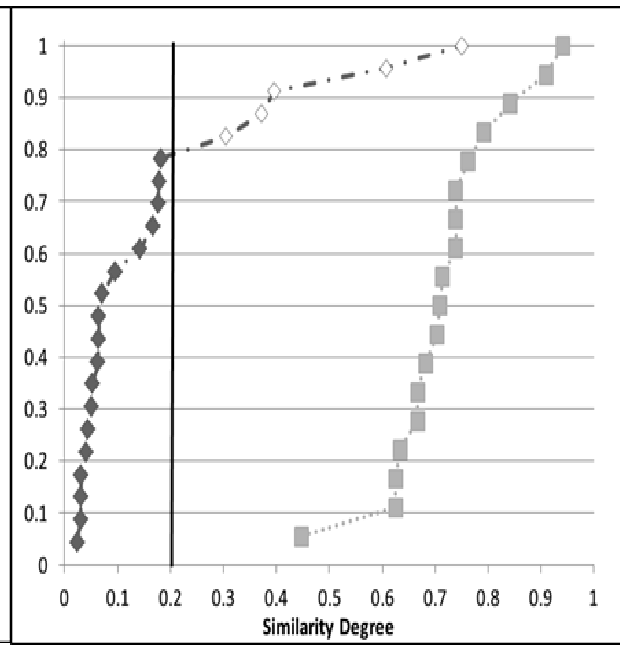
Courtesy of Guilherme Dal Bianco

BLOSS Third Stage – Part B

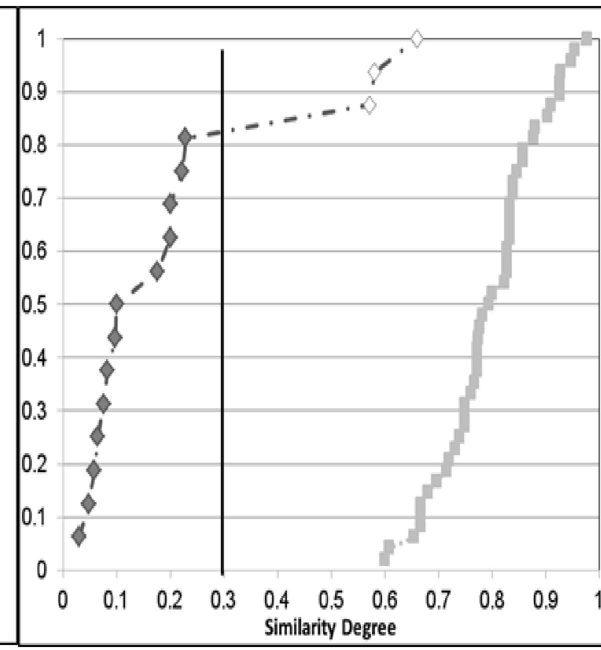
Pruning Threshold:
average **Jaccard** similarity of non-matching labelled instances.



Synthetic Dataset



DBLP-Scholar



DBLP-ACM

Courtesy of Guilherme Dal Bianco

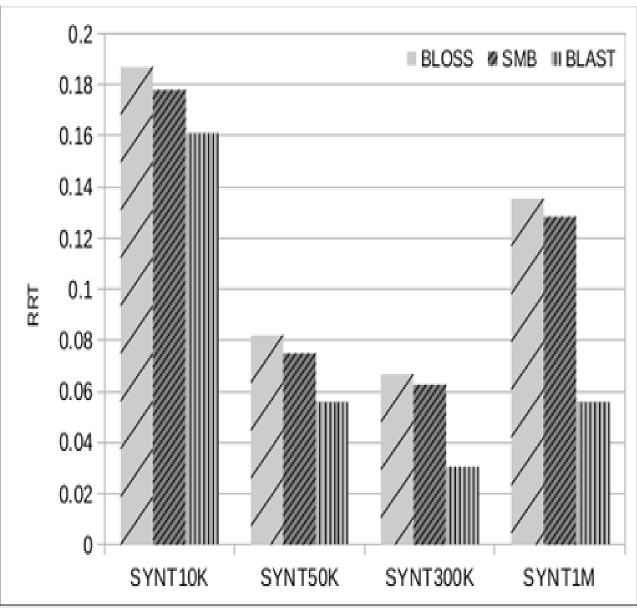
BLOSS Effectiveness & Time Efficiency

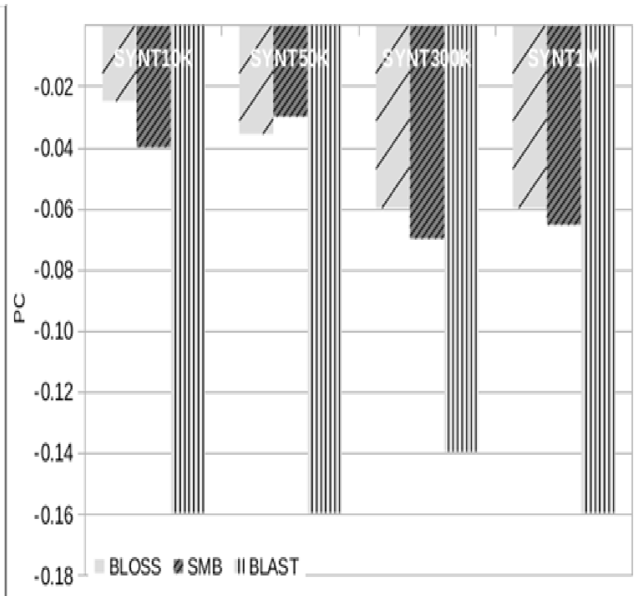
Measures:

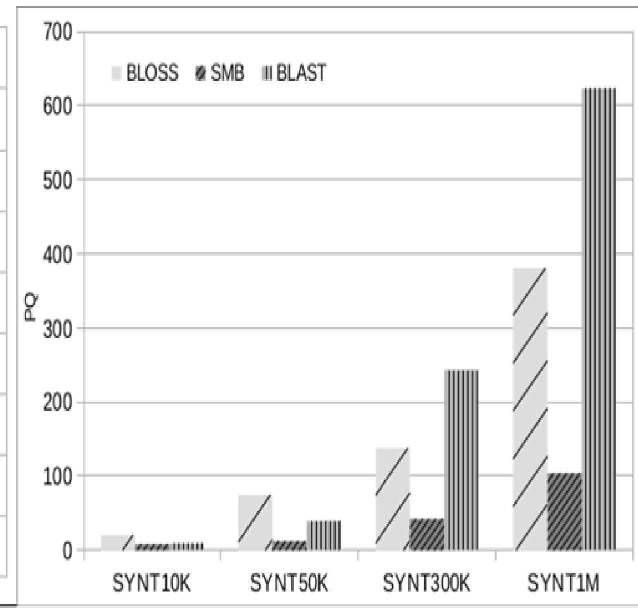
RRT = Relative **running time** wrt to original blocking (without Meta-blocking)

ΔPC = Reduction in **recall** wrt to original blocking

ΔPQ = Increase in **precision** wrt to original blocking





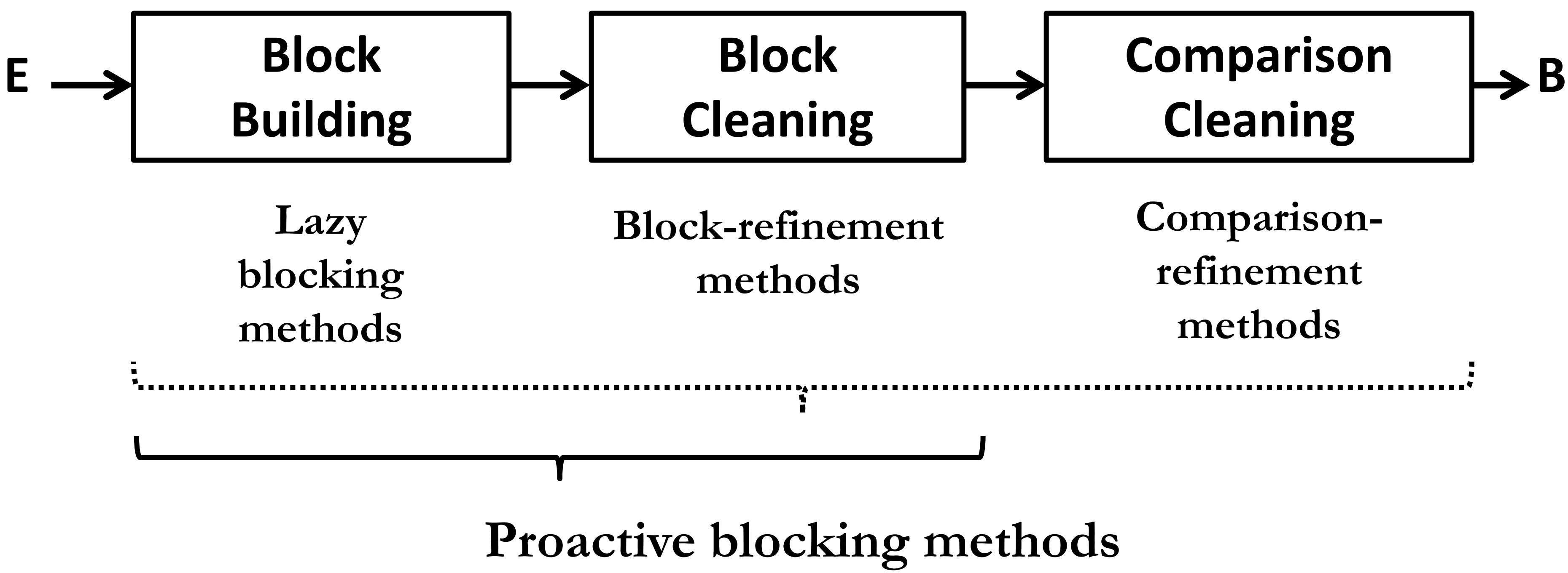


BLOSS achieves a reduction in the training set size of around 40 times.

Courtesy of Guilherme Dal Bianco

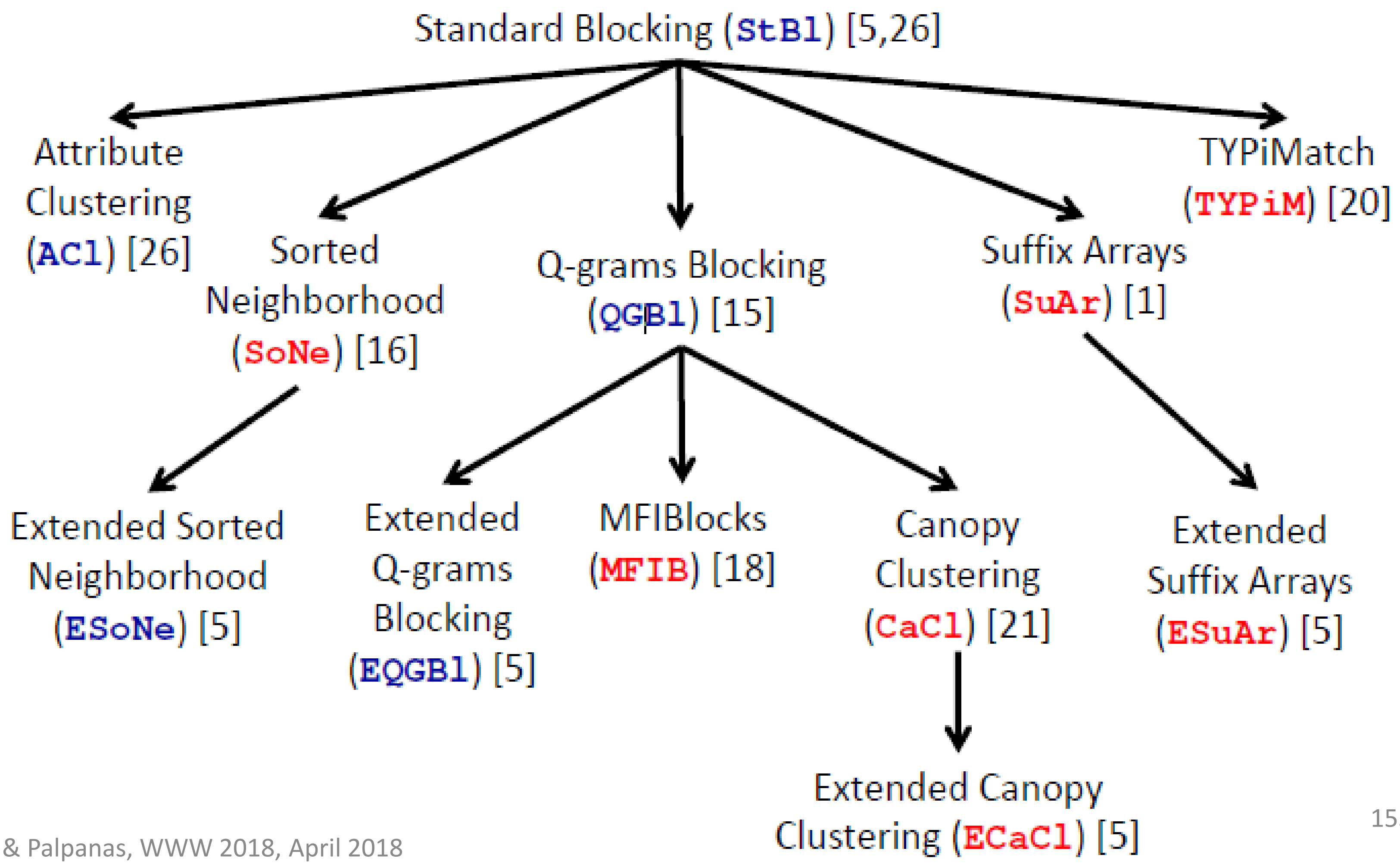
Comparative Analysis of Approximate Blocking Techniques [Papadakis et. al., VLDB 2016]

- employed 3 sub-tasks of blocking



Comparative Analysis of Approximate Blocking Techniques [Papadakis et. al., VLDB 2016]

- considered 5 lazy and 7 proactive blocking methods



Experimental Analysis Setup

- Block Cleaning methods:
  1. Block Purging
  2. Block Filtering

- Comparison Cleaning methods:
  1. Comparison Propagation
  2. Iterative Blocking
  3. Meta-blocking

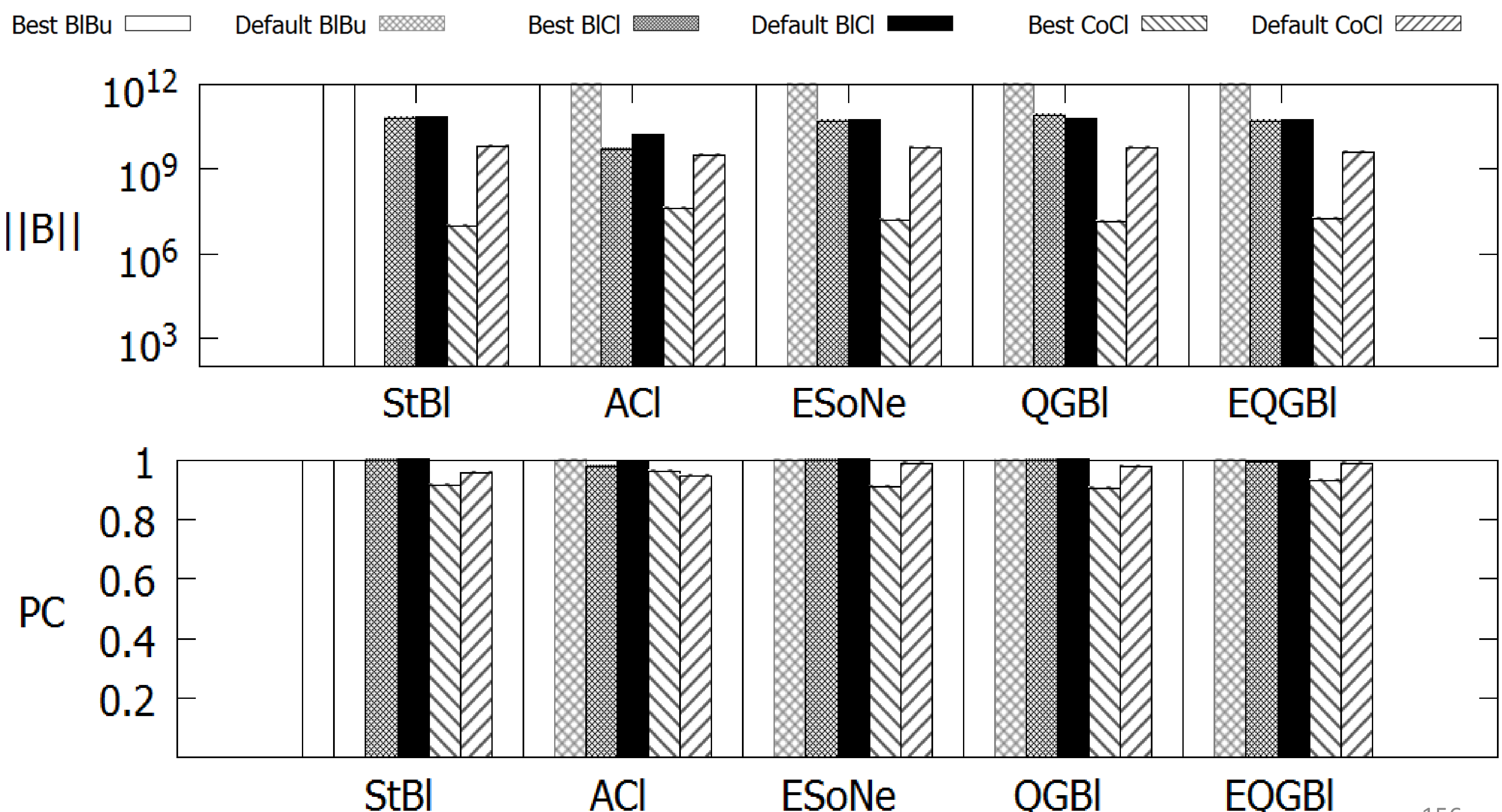
Experimental Analysis Setup

- Exhaustive parameter tuning to identify two configurations for each method:
  1. Best configuration per dataset → maximizes
     $$\boldsymbol{a(B,E) = RR(B,E) \cdot PC(B,E)}$$
  2. Default configuration → highest average $\boldsymbol{a}$ across all datasets

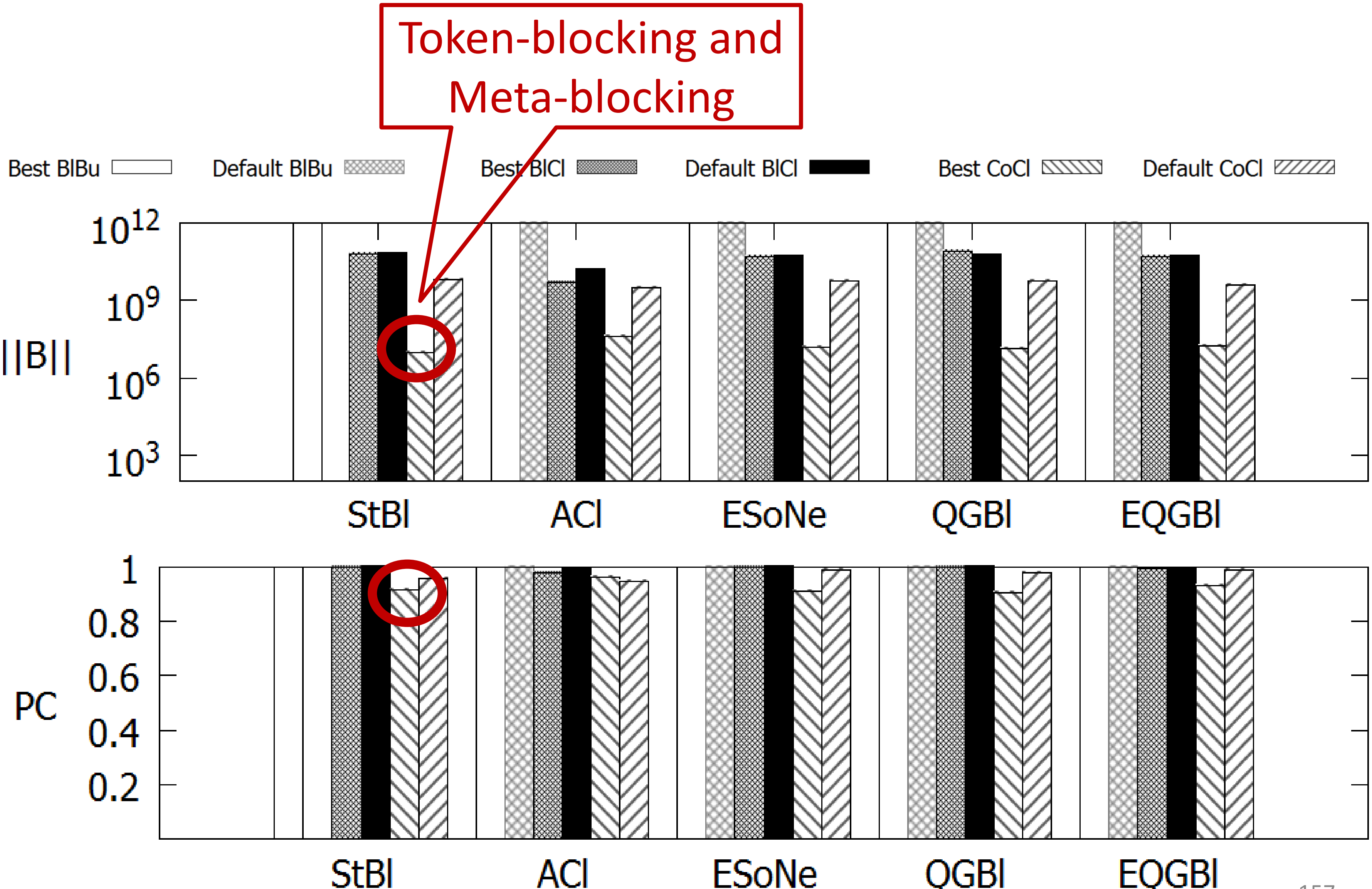
- Extensive experiments measuring effectiveness and time efficiency over **5 real** datasets (up to 3.3M entities).

- Scalability analysis over **7 synthetic** datasets (up to 2M entities).

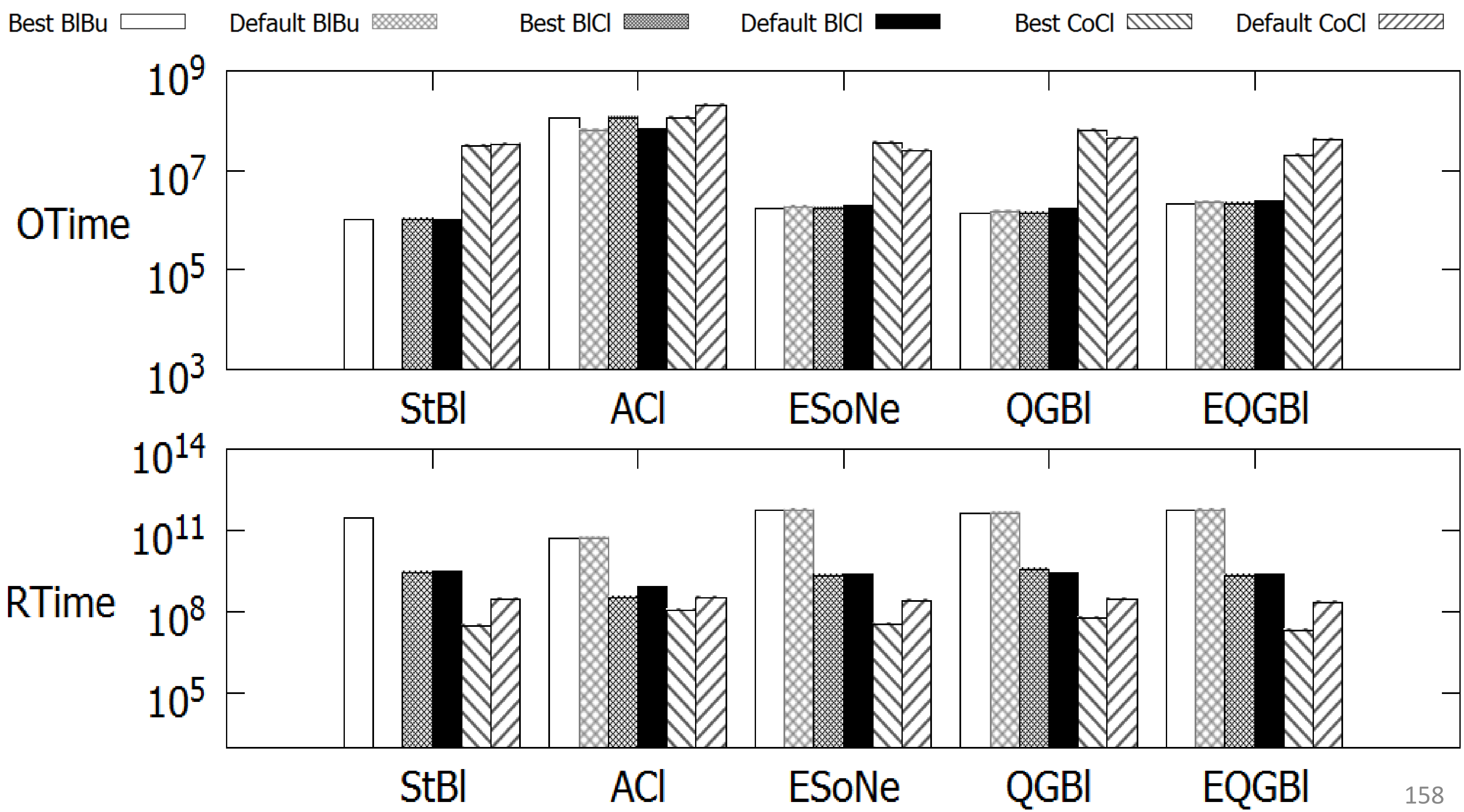
Effectiveness of Lazy Methods on DBPedia



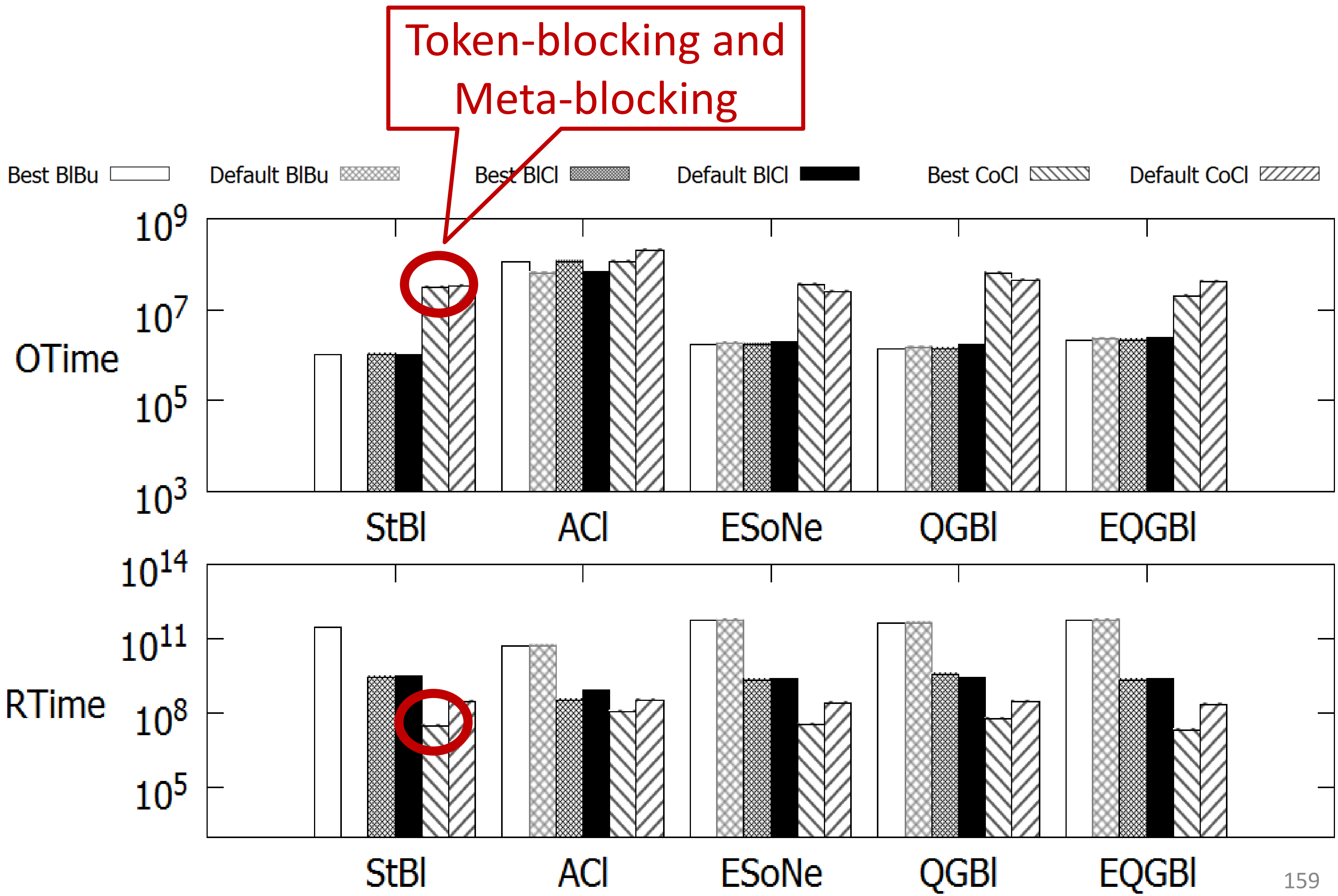
Effectiveness of Lazy Methods on DBPedia



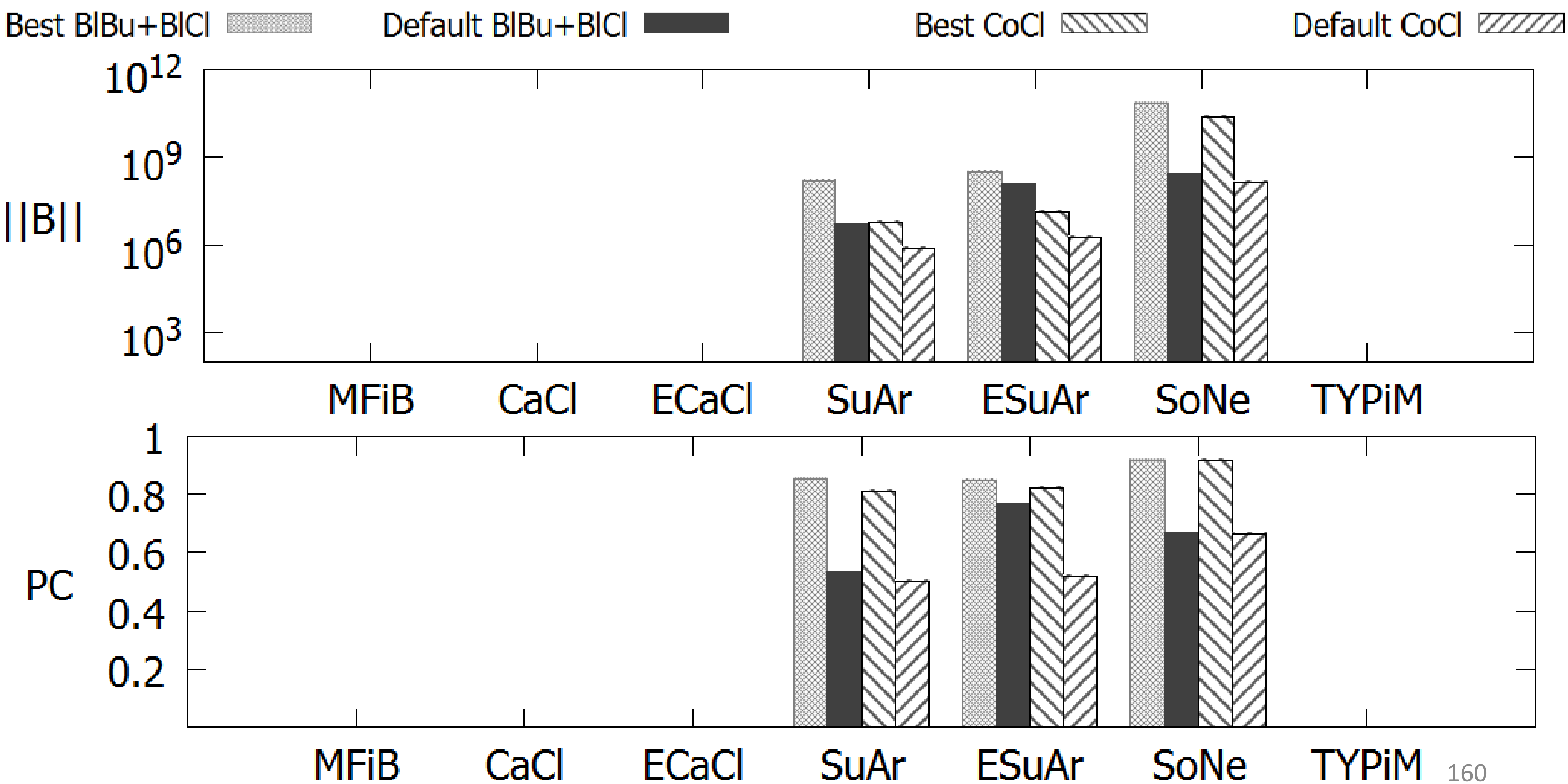
Time Efficiency of Lazy Methods on DBPedia



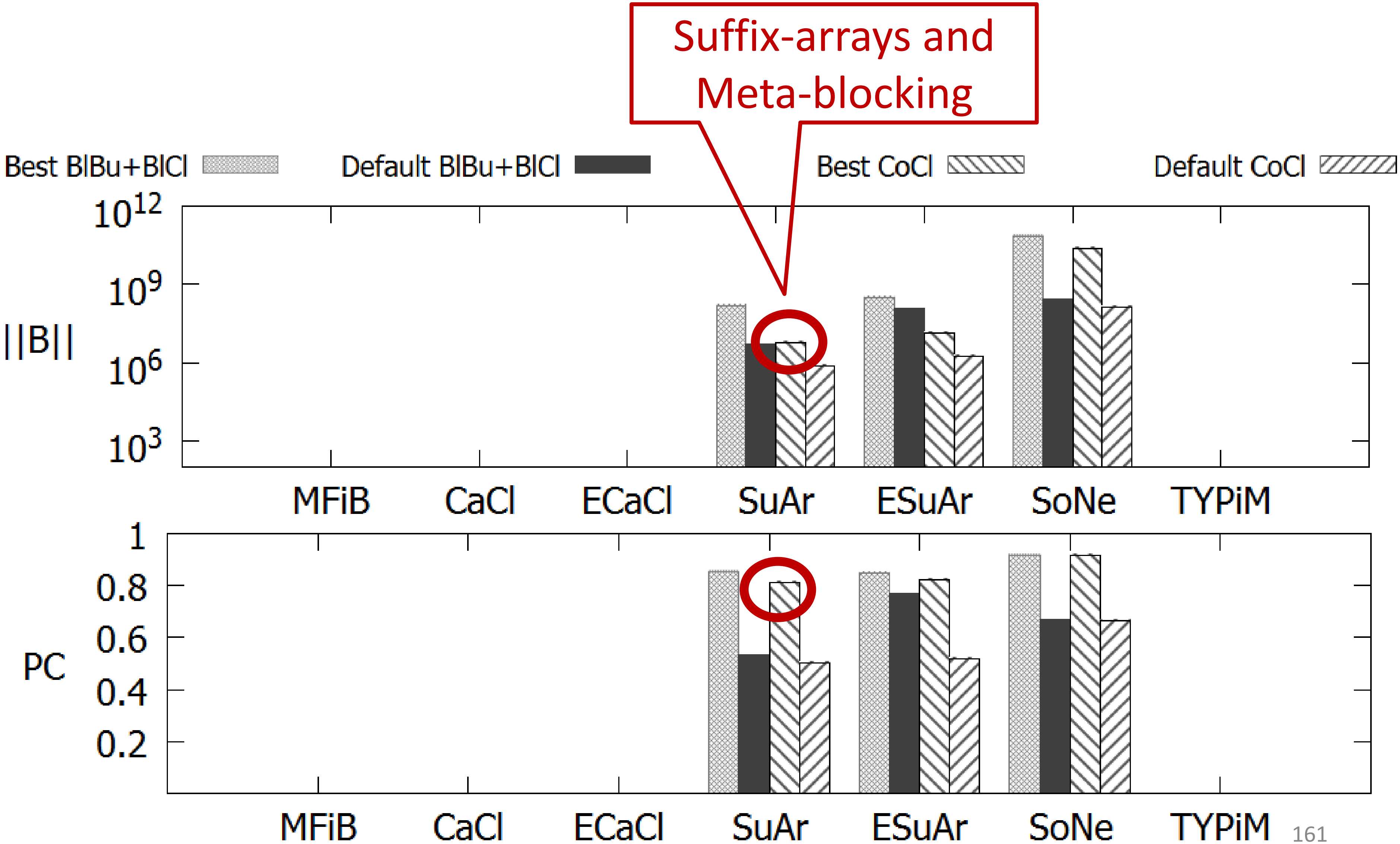
Time Efficiency of Lazy Methods on DBPedia



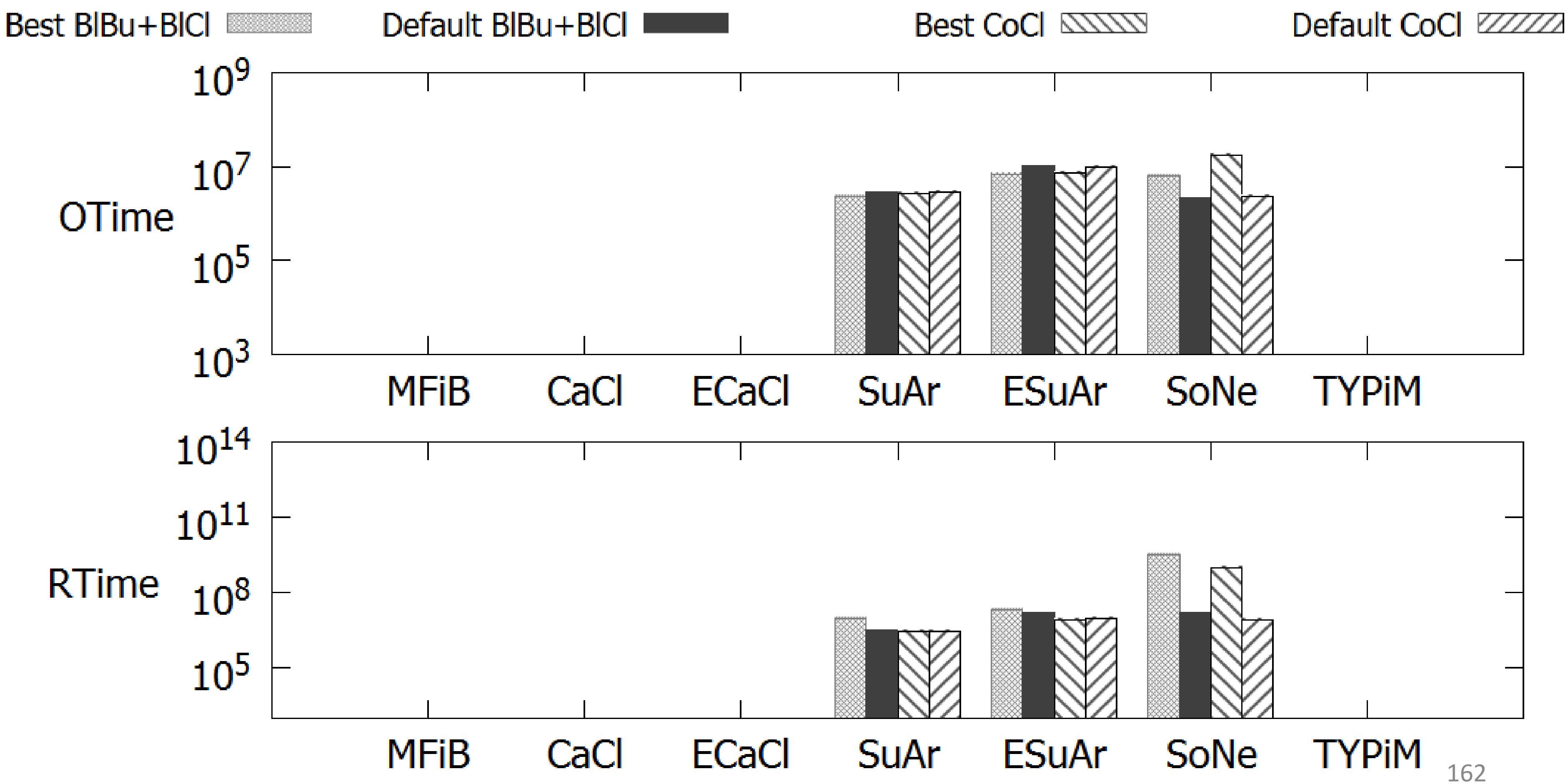
Effectiveness of Proactive methods on DBPedia



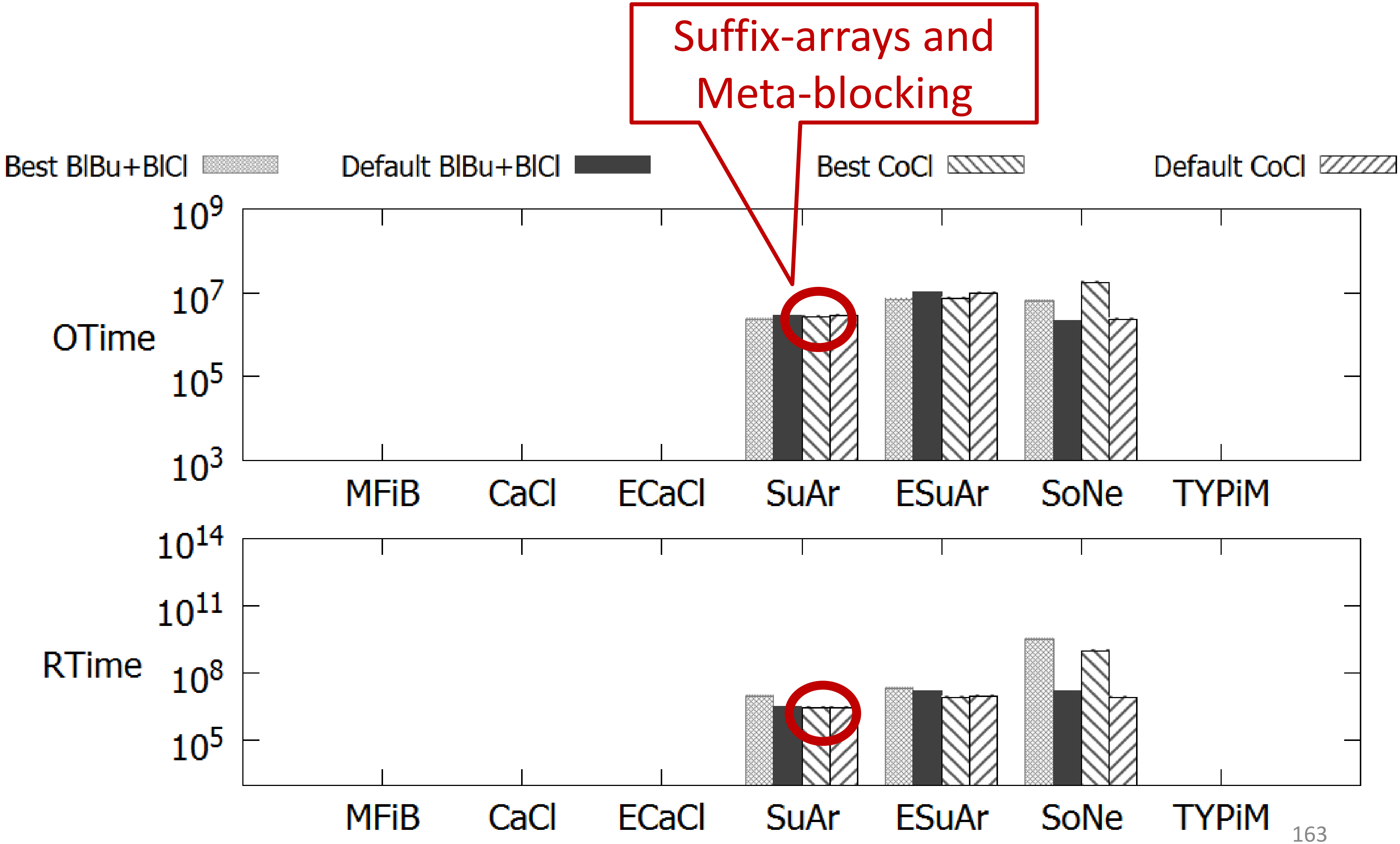
Effectiveness of Proactive methods on DBPedia



Time Efficiency of Proactive Methods on DBPedia



Time Efficiency of Proactive Methods on DBPedia



Part 6:

Entity Matching

Preliminaries

- Estimates the similarity of candidate matches.
- Input
  - Pruned Blocking Graph
    - Nodes → entities
    - Edges → candidate matches
  - **Or**, a set of blocks
    - Every comparison in any block is a candidate match
- Output
  - Similarity Graph
    - Nodes → entities
    - Edges → candidate matches
    - Edge weights → similarity of entity profiles (+neighbors)

Naïve Approach

- For each pair of entities, e_1-e_2
  - Estimate aggregate similarity based on:
    - attribute values
    - neighbors
    - external knowledge
    - combination of above
  - If similarity > threshold → match! (unsupervised)
  - If classifierDecision(e_1,e_2) = true → match! (supervised)

low recall!

Group Linkage [On et al., ICDE 2007]

- Often, "entity" is represented as a uniquely identified **group** of information
- In structured data:
  - An author with a **group** of publication records
  - A household in a census survey with a **group** of family members
- In semi-structured data:
  - Every entity is a group of name-value pairs.

Group Linkage Problem: to determine if two entities represented as groups are approximately the same or not

Courtesy of Dongwon Lee

Group Linkage: Popular Group Similarity

Jaccard

- Intuitive, cheap to run
- Error-prone

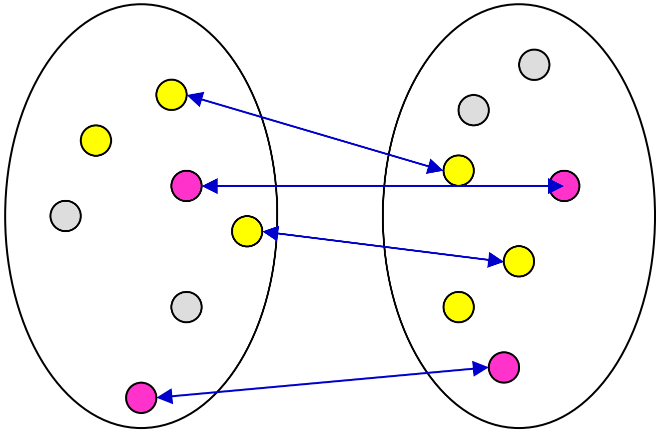
$$sim(g_1, g_2) = \left| \frac{g_1 \cap g_2}{g_1 \cup g_2} \right|$$

Q: Can we combine Jaccard and Bipartite Matching for Group Linkage?

Bipartite Matching

- Cardinality
- Weighted

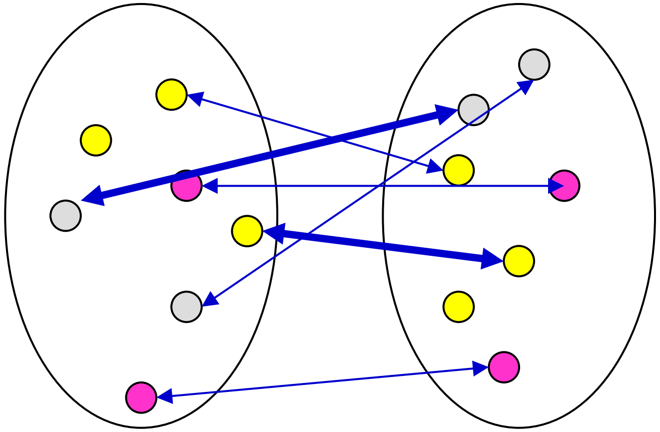
- Rich
- Expensive to run



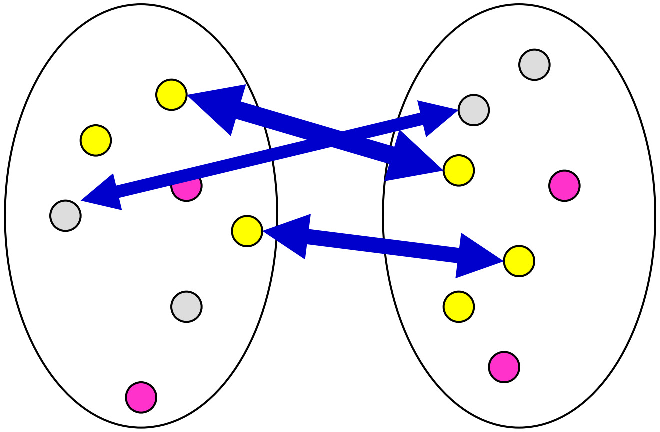
Courtesy of Dongwon Lee

Group Linkage: Intuition for Better Similarity

- Two groups are similar if:
  - A large fraction of elements in the two groups form matching element pairs

  - There is high enough similarity between matching pairs of individual elements that constitute the two groups

Group Linkage: Group Similarity

- Two groups of elements: $$sim(g_1, g_2) = \left| \frac{g_1 \cap g_2}{g_1 \cup g_2} \right|$$
  - $g_1 = \{r_{11}, r_{12}, \ldots, r_{1m1}\}$, $g_2 = \{r_{21}, r_{22}, \ldots, r_{2m2}\}$
  - The group measure ***BM*** is the normalized weight of the maximum bipartite matching *M* in the bipartite graph ($N = g_1 \, U \, g_2$, $E = g_1 \, X \, g_2$)

$$\boxed{BM_{sim,\rho}(g_1, g_2) = \frac{\sum_{(r_{1i}, r_{2j}) \in M} (sim(r_{1i}, r_{2j}))}{m_1 + m_2 - |M|}}$$

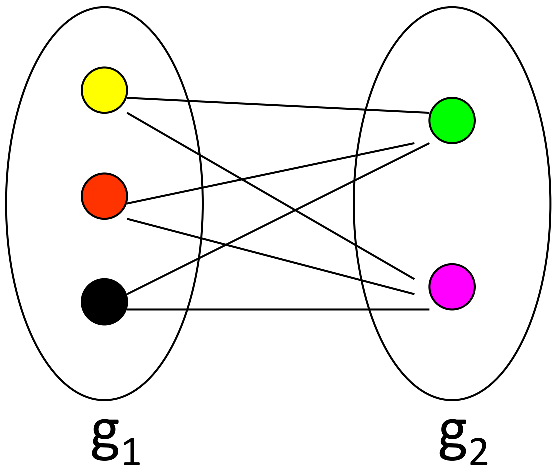
such that $sim(r_{1i}, r_{2j}) \geq \rho$

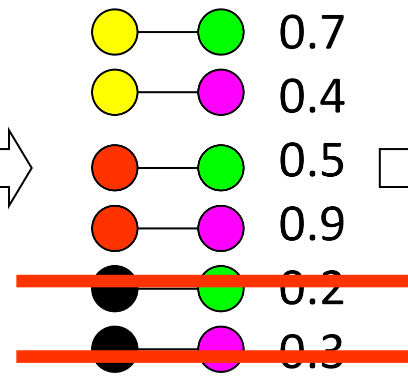
  - $BM(g_1, g_2) \geq \theta$

user-defined parameters

Courtesy of Dongwon Lee

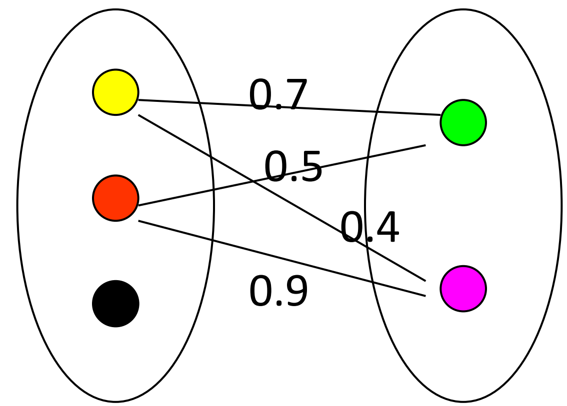
Group Linkage: Example ($\rho = 0.3, \Theta = 0.9$)

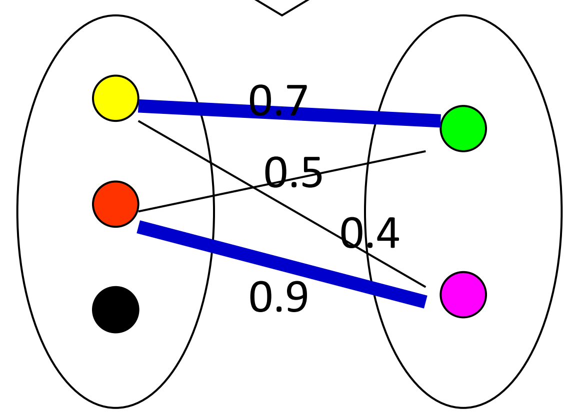




$\rho = 0.3$

sparse bipartite graph





M: max-weight bipartite matching

$$BM_{sim,\rho}(g_1, g_2) = \frac{\sum_{(r_{1i}, r_{2j}) \in M} (sim(r_{1i}, r_{2j}))}{m_1 + m_2 - |M|}$$

$$= \frac{0.9 + 0.7}{3 + 2 - 2} = \frac{1.6}{3} = 0.53 < \Theta$$

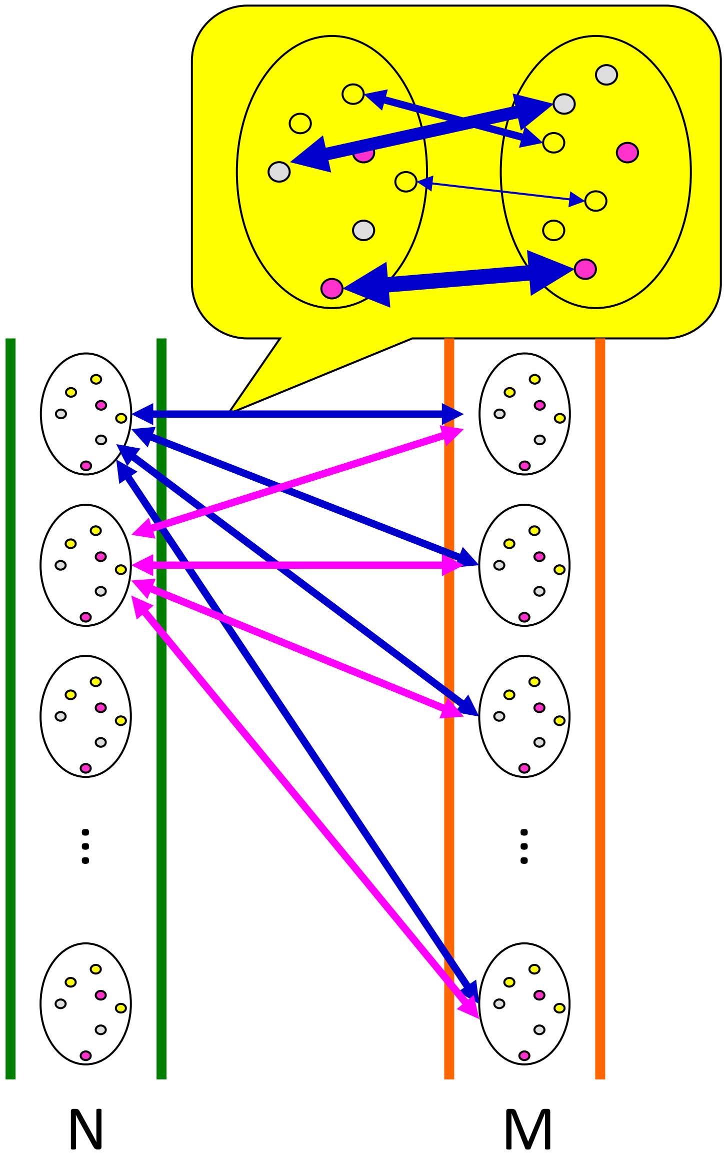
Therefore, g_1 <> g_2 !

Courtesy of Dongwon Lee

Group Linkage: Challenge

- *Each BM* group measure uses the maximum weight bipartite matching
  - Bellman-Ford: $O(V^2E)$
  - Hungarian: $O(V^3)$

- Large number of groups to match
  - $O(NM)$



Courtesy of Dongwon Lee

Group Linkage: Solution: Greedy matching

- Bipartite matching computation is expensive because of the requirement
  - No node in the bipartite graph can have more than one edge incident on it

- Let's relax this constraint:
  - For each element e_i in g_1, find an element e_j in g_2 with the highest element-level similarity ⇔ $\mathbf{S_1}$
  - For each element e_j in g_2, find an element e_i in g_1 with the highest element-level similarity ⇔ $\mathbf{S_2}$

Courtesy of Dongwon Lee

Upper/Lower Bounds

$$BM_{sim,\rho}(g_1, g_2) = \frac{\sum_{(r_{1i}, r_{2j}) \in M} (sim(r_{1i}, r_{2j}))}{m_1 + m_2 - |M|}$$

$$UB_{sim,\rho}(g_1, g_2) = \frac{\sum_{(r_{1i}, r_{2j}) \in S1 \cup S2} (sim(r_{1i}, r_{2j}))}{m_1 + m_2 - |S_1 \cup S_2|}$$

$$LB_{sim,\rho}(g_1, g_2) = \frac{\sum_{(r_{1i}, r_{2j}) \in S1 \cap S2} (sim(r_{1i}, r_{2j}))}{m_1 + m_2 - |S_1 \cap S_2|}$$

Courtesy of Dongwon Lee

Theorem & Algorithm

$$\boxed{BM_{sim,\rho}(g_1, g_2) \leq UB_{sim,\rho}(g_1, g_2)}$$ Theorem 1

- IF $UB(g_1,g_2) < \theta \rightarrow BM(g_1,g_2) < \theta \rightarrow g_1 \neq g_2$

$$\boxed{LB_{sim,\rho}(g_1, g_2) \leq BM_{sim,\rho}(g_1, g_2)}$$ Theorem 2

- **ELSE IF** $LB(g_1,g_2) \geq \theta \rightarrow BM(g_1,g_2) \geq \theta \rightarrow g_1 \approx g_2$
- **ELSE**, compute $BM(g_1,g_2)$

Goal: $BM(g_1,g_2) \geq \theta$

Courtesy of Dongwon Lee

Iterative Approaches [Stefanidis et al., WWW 2014]

- Increase recall by updating related entities upon detection of a new match

- Core principles:
  - Transitivity: if match(e_1,e_2)=true & match(e_2,e_3)=true → match(e_1,e_3)=true
  - Duplicate dependency: if entities of one type (e.g., authors) are matches, related entities of another type (e.g., publications) are more likely to be matches, too.
  - Merge dependency: if match(e_1,e_2)=true, replace e_1 & e_2 with e_{12} and compare again with all other similar entities.

Swoosh [Benjelloun et al., VLDBJ 2009]

- Iterative approach crafted for relational data.
- Relies on two functions: `match` (m) and `merge` (μ)
- Algorithm outline:

while the input list ***I*** is not empty

- $e_1 \leftarrow$ ***I***.removeFirstRecord()
- matchFound = `false`
- for each record in the output list , $e_2 \in \boldsymbol{O}$
  - if **m** (e_1, e_2) == `true` then
    - ***O.***remove (e_2)
    - ***I .***add(**μ** (e_1 , e_2))
    - matchFound = true
    - break
- if matchFound == `false`
  - ***O.***add(e_1)

Swoosh Efficiency [Benjelloun et al., VLDBJ 2009]

- Higher efficiency (fewer calls to `match` & `merge`) when specific properties hold:
  - Idempotence:

    $\mathtt{m}(e_i, e_i) = \mathtt{true}$, $\mu(e_i, e_i) = e_i$
  - Commutativity:

    $\mathtt{m}(e_i, e_j) = \mathtt{M}(e_j, e_i)$, $\mu(e_i, e_j) = \mu(e_j, e_i)$
  - Associativity:

    $\mu(e_i, \mu(e_j, e_k)) = \mu(\mu(e_i,e_j), e_k)$
  - Representativity:

    if $\mu(e_i, e_j) = e_k$ & $\mathtt{m}(e_i, e_l) = \mathtt{true} \rightarrow \mathtt{m}(e_k, e_l) = \mathtt{true}$

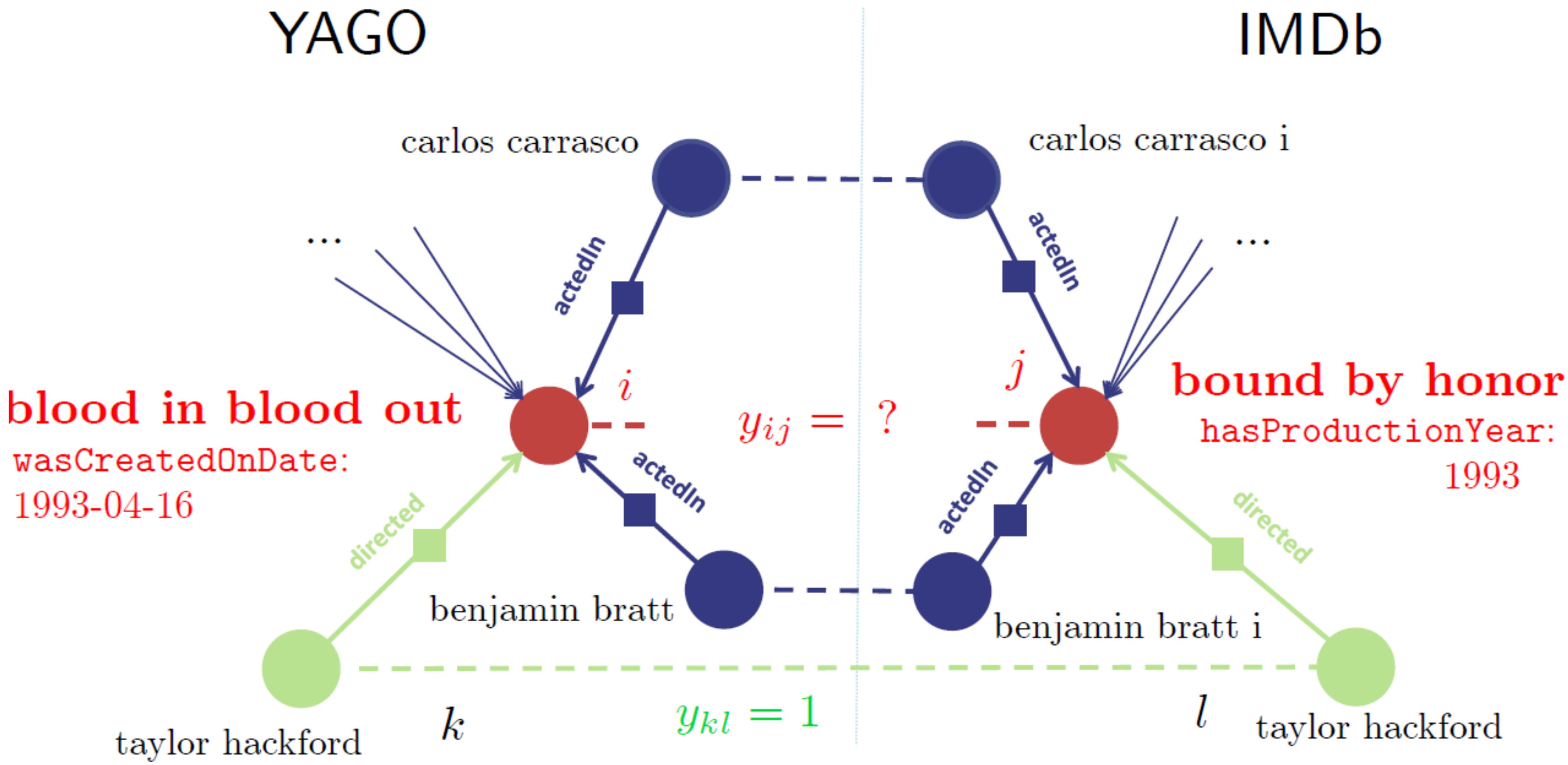
Simple **G**reedy **Ma**tching (SiGMa)

[Lacoste-Julien et al., KDD 2013]

- relies on the 1-1 assumption of Clean-Clean ER
  - once a match is identified, it never needs to be compared to other entities
- exploits relationship graph to score decisions and to propose candidates
- can easily use tailored similarity measures
- iterative algorithm
  - provides natural tradeoff between precision & recall as well as between computation and recall
- simplicity & greediness → high time efficiency
- exhibits high effectiveness, as well

Courtesy of Simon Lacoste-Julien

SiGMa intuition



SiGMa uses neighbors for: 1) scoring candidates

2) suggest candidates (iterative blocking)

Courtesy of Simon Lacoste-Julien

Quadratic Assignment objective

normalizing weight

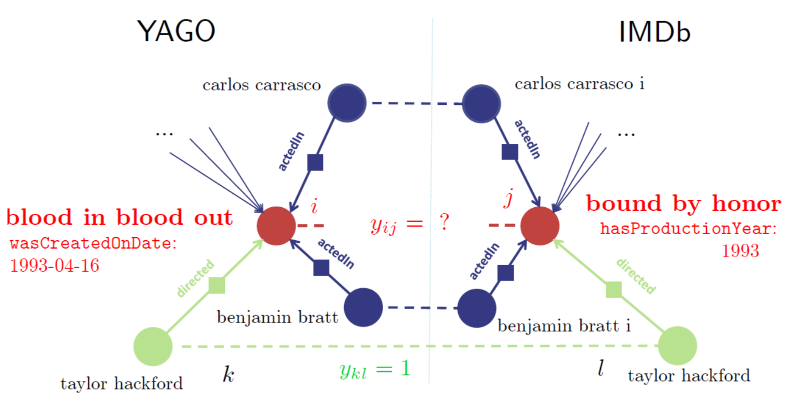
$$\max_{y \in \mathcal{M}} \sum_{(i,j)} y_{ij} \left[s_{ij} + \underbrace{\sum_{(k,l) \in \mathcal{N}_{ij}} y_{kl} w_{ij,kl}}_{} \right] \qquad y_{ij} \in \{0,1\}$$

pairwise similarity score

between i and j

graph compatibility score:

counts the number of valid neighbors which are currently matched (**context**)



Courtesy of Simon Lacoste-Julien

SiGMa similarity scores

- Increase in objective when matching pair (i,j):

$$\texttt{score}(i,j;y) = (1-\alpha)s_{ij} + \alpha \sum_{(k,l)\in\mathcal{N}_{ij}} y_{kl}(w_{ij,kl} + w_{kl,ij})$$

- Pairwise similarity score (could use others):

$$s_{ij} = (1-\beta)\texttt{string}(i,j) + \beta\texttt{prop}(i,j)$$

- Similarity on string representation of entities:
  - Jaccard measure on words in common + smoothing + weights (TF-IDF weights)

$$\texttt{prop}(i,j) = \frac{\sum_{(a,b)\in M_{12}} (w^1_{p_a,v_a} + w^2_{q_b,l_b})\,\mathrm{Sim}_{p_a,q_b}(v_a,l_b)}{2 + \sum_{a=1}^{n_1} w^1_{p_a,v_a} + \sum_{b=1}^{n_2} w^2_{q_b,l_b}}$$

  - Property similarity measure: also smoothed weighted Jaccard similarity measure between sets of properties, with additional similarity on literals:

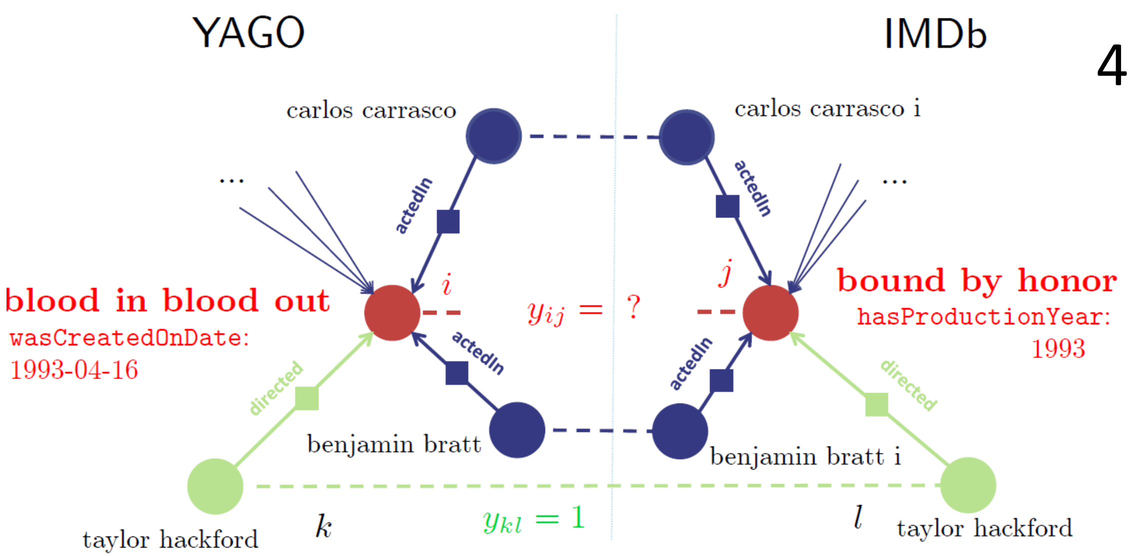
$$\texttt{string}(i,j) = \frac{\sum_{v\in(\mathcal{W}_i\cap\mathcal{W}_j)} (w^1_v + w^2_v)}{\texttt{smoothing} + \sum_{v\in\mathcal{W}_i} w^1_v + \sum_{v'\in\mathcal{W}_j} w^2_{v'}},$$

Courtesy of Simon Lacoste-Julien

SiGMa Algorithm

$$\sum_{(i,j)} y_{ij} \left[s_{ij} + \sum_{(k,l) \in \mathcal{N}_{ij}} y_{kl}\, w_{ij,kl} \right]$$

1. Start with seed match
2. Put neighbors in S
3. At each iteration:
   a) pick new pair in S which max. increase
   b) add new neighbors in S
4. Stop when variation below threshold



Courtesy of Simon Lacoste-Julien

PARIS [Suchanek et al., PVLDB 2011]

- Probabilistic, iterative, parameter-free method
- Collective approach for holistically aligning entities, relations and classes
  - applicable to Clean-Clean ER (for knowledge graphs)
- Algorithm outline:
  1. Fix equalities for literals (numbers, or strings)
  2. Set equalities for relations to a small initial value
  3. Iterate the estimations for relations and entities until convergence (*)
  4. Compute the estimations for classes

(*) There is no proof for convergence, but it seems to happen

Courtesy of Fabian Suchanek

PARIS – Part B

- Equality of Literals
  - $Pr(x \equiv y) := (x = y)\ ?\ 1 : 0$
- Equality of Entities
  - Based on the local inverse functionality of a relation r

    `1/#the number of entities with a given argument for r`
  - The probability of a relation being inverse functional is the harmonic mean of the local inverse functionalities
  - Two entities are matching if they share at least one argument for a highly inverse functional relation
- Equality of Classes
  - Based on the subsumption probability: if all entities of one class are entities of the other, then the former subsumes the latter
- Equality of Relations
  - Based on the probability that one relation is sub-property of the other

Synthesizing Entity Matching Rules

Machine Learning System

Best F-measure
Not interpretable

Rule-Based Approach

Lower F-measure
Interpretable

Courtesy of Paolo Papotti

Synthesizing Entity Matching Rules Using Examples [Singh et al., PVLDB 2017]

name	address	email	nation	gender
Catherine Zeta-Jones	9601 Wilshire Blvd., Beverly Hills, CA 90210-5213	c.jones@gmail.com	Wales	F
C. Zeta-Jones	3rd Floor, Beverly Hills, CA 90210	c.jones@gmail.com	US	F
Michael Jordan	676 North Michigan Avenue, Suite 293, Chicago		US	M
Bob Dylan	1230 Avenue of the Americas, NY 10020		US	M

name	apt	email	country	sex
Catherine Zeta-Jones	9601 Wilshire, 3rd Floor, Beverly Hills, CA 90210	c.jones@gmail.com	Wales	F
B. Dylan	1230 Avenue of the Americas, NY 10020	bob.dylan@gmail.com	US	M
Michael Jordan	427 Evans Hall #3860, Berkeley, CA 94720	jordan@cs.berkeley.edu	US	M

Program Synthesis

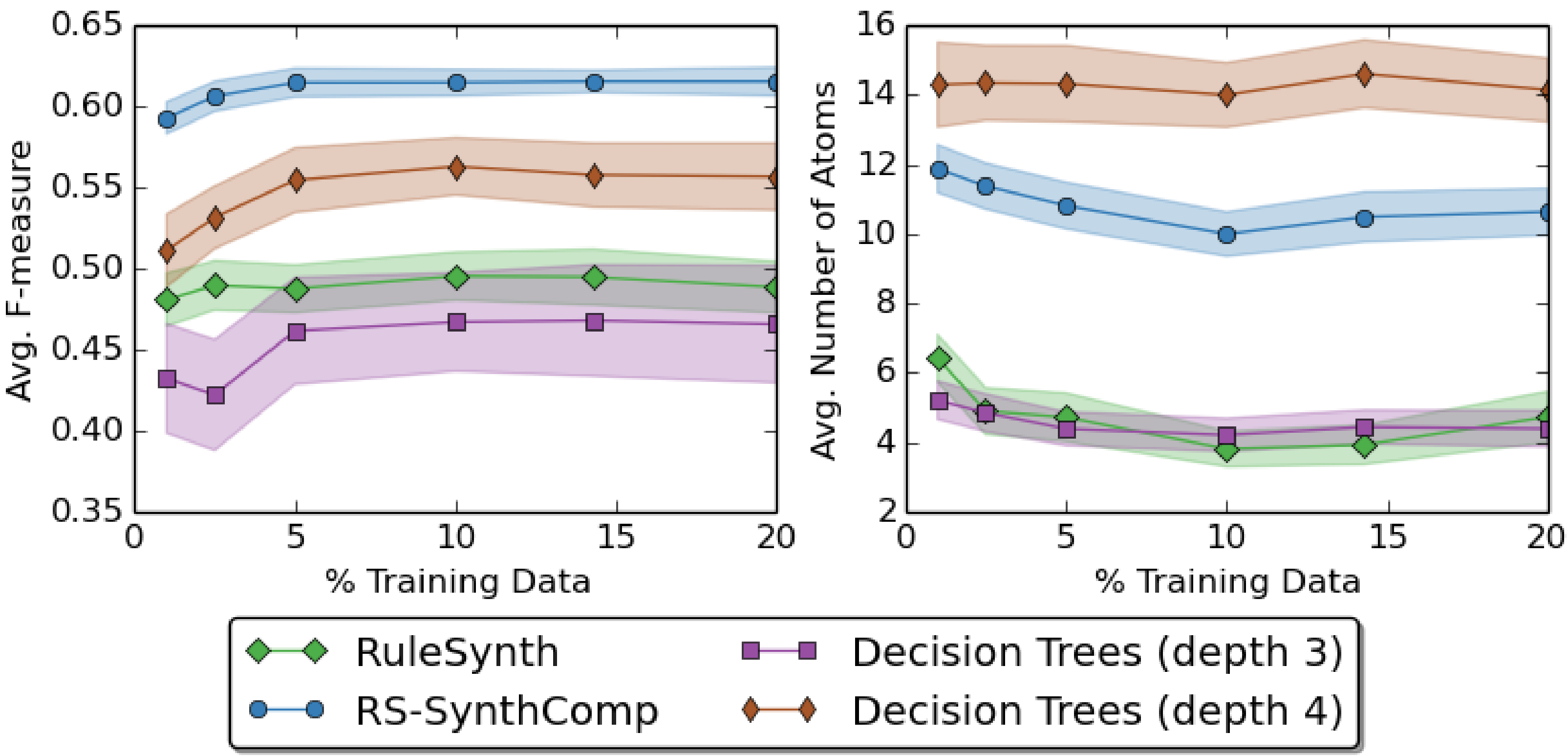
Tuneable trade off between F1 and complexity

$$\begin{array}{ll} \textbf{if} & (r[\text{email}] \neq \text{Null} \bigwedge s[\text{email}] \neq \text{Null}) \\ \textbf{then} & r[\text{name}] \approx_1 s[\text{name}] \bigwedge r[\text{email}] = s[\text{email}] \\ \textbf{else} & r[\text{name}] \approx_3 s[\text{name}] \bigwedge r[\text{address}] \approx_2 s[\text{apt}] \bigwedge \\ & r[\text{nation}] = s[\text{country}] \bigwedge r[\text{gender}] = s[\text{sex}] \end{array}$$

General Boolean Formula (GBF): can include arbitrary attribute matching predicates combined by conjunctions, disjunctions, and negations

Courtesy of Paolo Papotti

Synthesizing Entity Matching Rules Using Examples [Singh et al., PVLDB 2017]



F-measure comparable to DTs depth 10 and SVM

Part 7:

Entity Clustering

Preliminaries

- Partitions the matched pairs into equivalence clusters → groups of entity profiles describing the same real-world object

- Input
  - Similarity Graph:
    - Nodes → entities
    - Edges → candidate matches
    - Edge weights → likelihood of matching entities

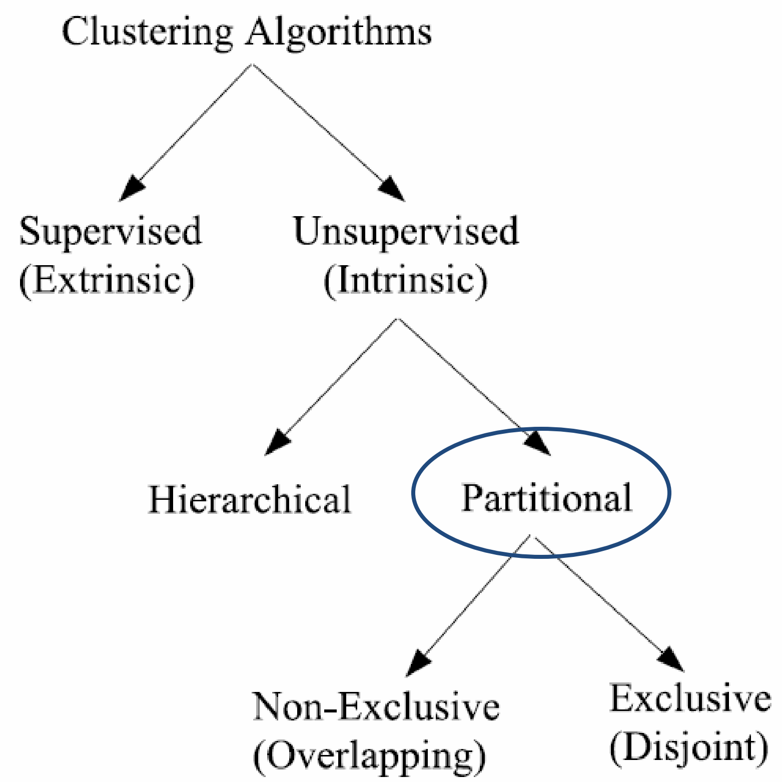
- Output
  - Equivalence Clusters

Clustering Algorithms for Clean-Clean ER

- Unique Mapping Clustering [Lacoste-Julien et al., KDD 2013] [Suchanek et al., PVLDB 2011]
  - Relies on 1-1 constraint
    - 1 entity from first dataset matches to 1 entity from second
  - Sorts all edges in decreasing weight
  - Starting from the top, each edge corresponds to a pair of duplicates **if**:
    - None of the adjacent entities has already been matched
    - predefined threshold < edge weight

Clustering Algorithms for Dirty ER

- A wealth of literature on clustering algorithms
- Requirements:
  - Partitional and disjoint Algorithms
    - Sometimes overlapping may be desirable
  - Goal: Sets of clusters that
    - maximize the intra-cluster weights
    - minimize the inter-cluster edge weights



Classification of clustering algorithms [Jain&Dubes88]

Courtesy of Oktie Hassanzadeh

Clustering Algorithms Characteristics

[Hassanzadeh et al., VLDB 2009]

- Most important feature

  “***Unconstrained algorithms***”

- I.e. , algorithms that do not require as input:
  - The number of clusters
  - The diameter of the clusters
  - Any other domain specific parameters

- Algorithms need to be able to *predict* the correct number of clusters

Courtesy of Oktie Hassanzadeh

Clustering Algorithms Characteristics

[Hassanzadeh et al., VLDB 2009]

- Need to scale well
  - Time complexity < $O(n^2)$
- Need to be robust with respect to characteristics of the data
  - E.g., distribution of the duplicates
- Need to be capable of finding ‘singleton’ clusters
  - Different from many clustering algorithms
    - E.g., algorithms proposed for image segmentation

Courtesy of Oktie Hassanzadeh

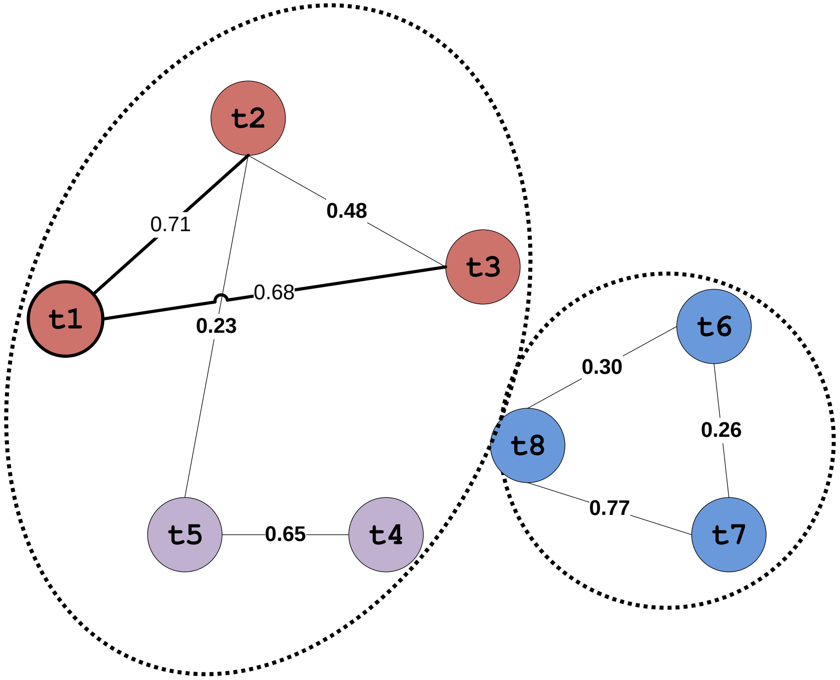
Single-pass Algorithms [Hassanzadeh et al., VLDB 2009]

- Perform clustering by a single scan of the output of the similarity join (the edges of the graph)
  - Partitioning
    - TRANSITIVE CLOSURE
  - CENTER [HGI-WebDB'00]
  - MERGE-CENTER [HM-VLDBJ09]

Courtesy of Oktie Hassanzadeh

Single-pass Algorithms [Hassanzadeh et al., VLDB 2009]

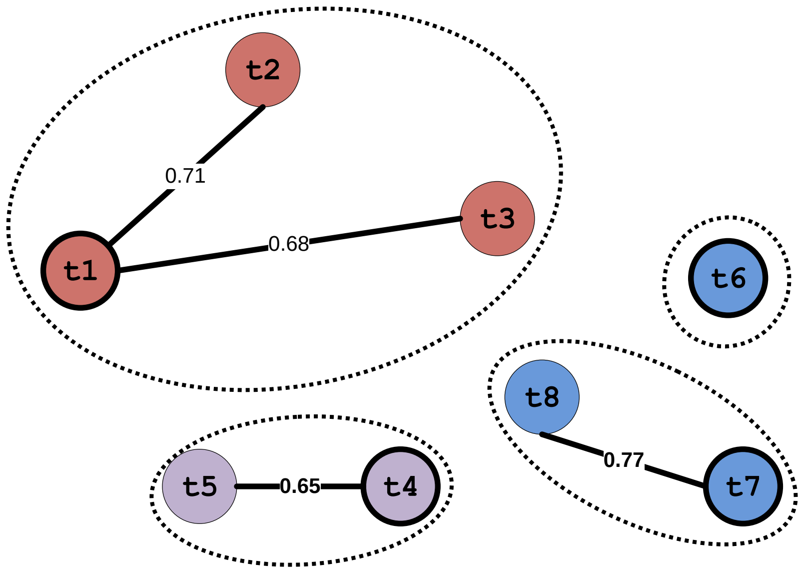
- Perform clustering by a single scan of the output of the similarity join (the edges of the graph)
  - Partitioning
    - **TRANSITIVE CLOSURE**
  - CENTER [HGI-WebDB'00]
  - MERGE-CENTER [HM-VLDBJ09]



Courtesy of Oktie Hassanzadeh

Single-pass Algorithms [Hassanzadeh et al., VLDB 2009]

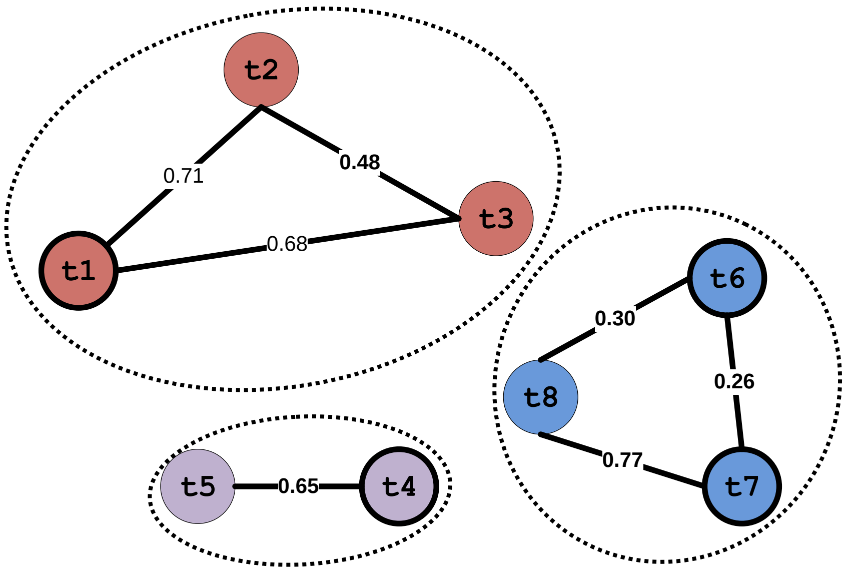
- Perform clustering by a single scan of the output of the similarity join (the edges of the graph)
  - Partitioning
    - TRANSITIVE CLOSURE
  - **CENTER** [HGI-WebDB'00]
  - MERGE-CENTER [HM-VLDBJ09]



Courtesy of Oktie Hassanzadeh

Single-pass Algorithms [Hassanzadeh et al., VLDB 2009]

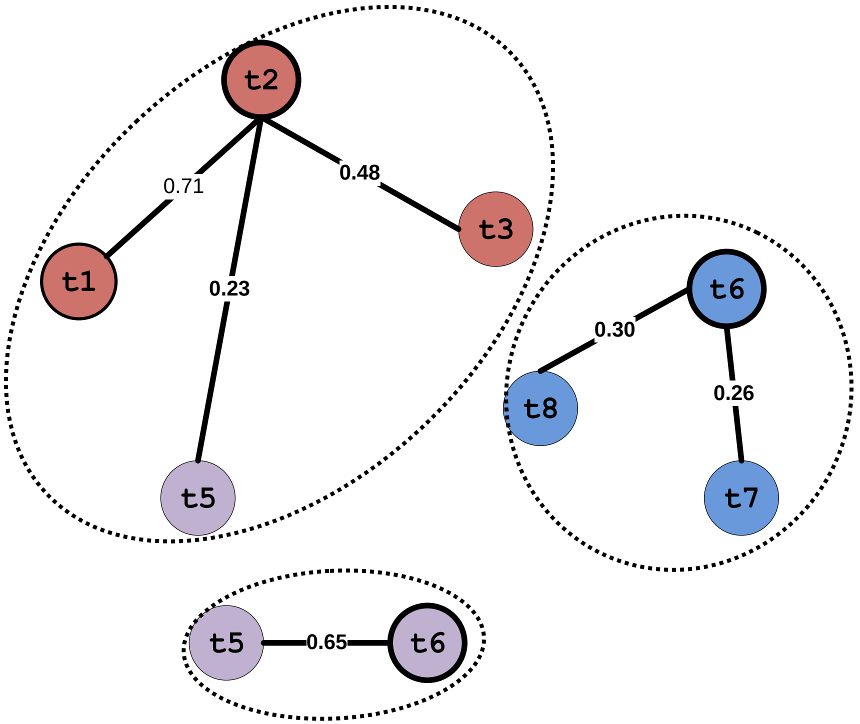
- Perform clustering by a single scan of the output of the similarity join (the edges of the graph)
  - Partitioning
    - TRANSITIVE CLOSURE
  - CENTER [HGI-WebDB'00]
  - **MERGE-CENTER** [HM-VLDBJ09]



Courtesy of Oktie Hassanzadeh

Star Algorithm [APR-JGraph04]

- Creates star-shaped clusters
  - heuristic to approximate problem of finding minimal clique cover of graph
- Similar to CENTER but
  - Allows *overlapping* clusters
  - First sorts nodes in descending order of their degrees



Courtesy of Oktie Hassanzadeh

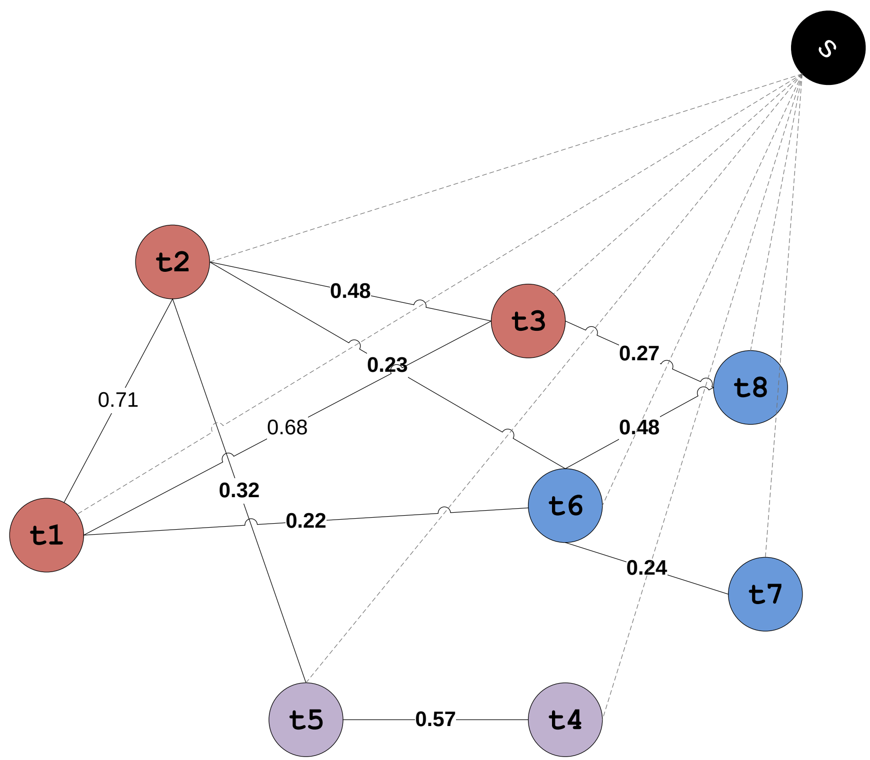
Ricochet Algorithms [WB-DASFAA'09]

- Ricochet family of algorithms
  - Based on a strategy that resembles the rippling of stones thrown in a pond
  - Combine ideas from the classic K-means algorithm and the Star algorithm
    - First selecting seeds (star centers) for the clusters and then refining the clusters iteratively
  - Four unconstrained clustering algorithms, originally proposed for document clustering
    - SR, BSR, CR and OCR
  - SR and BSR perform a sequential selection of the cluster seeds; CR and OCR perform a concurrent selection of the seeds

Courtesy of Oktie Hassanzadeh

Min-Cut Clustering [Hassanzadeh et al., VLDB 2009]

- Based on the Cut-Clustering Algorithm [FTT-IM04]
  - Finding minimum cuts of edges in the similarity graph after inserting an artificial sink into similarity graph G

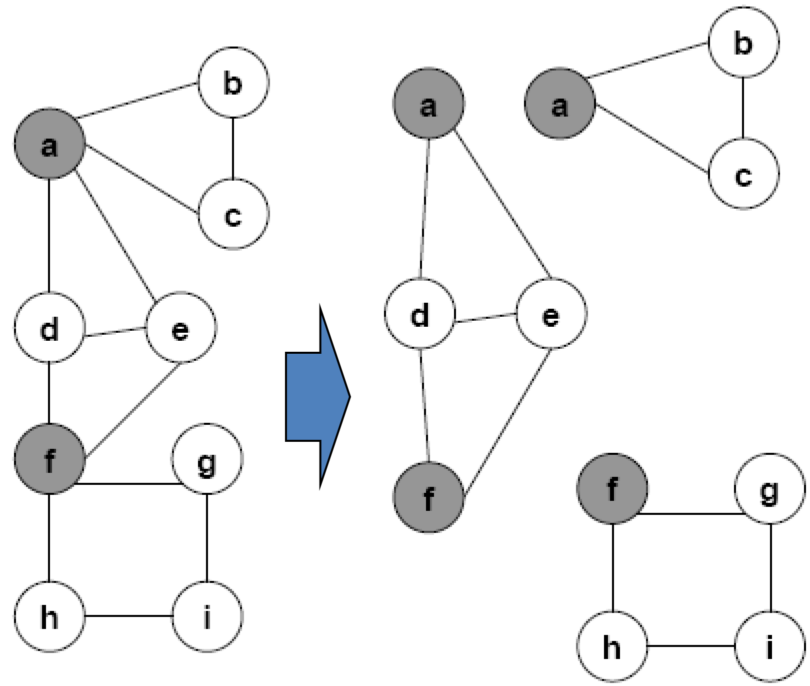


Courtesy of Oktie Hassanzadeh

Articulation Point Clustering

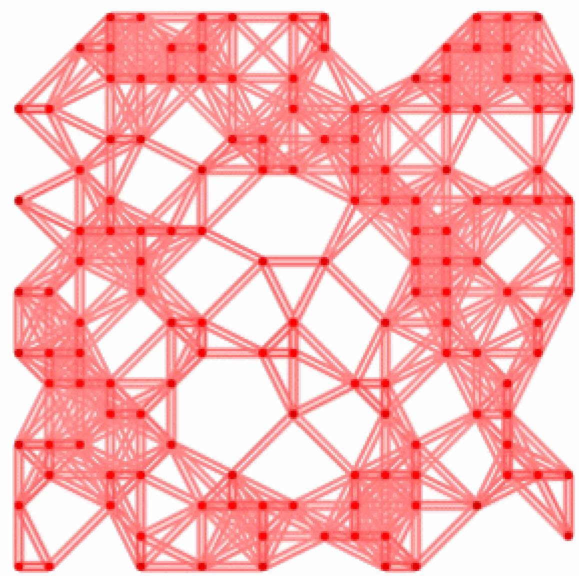
[Hassanzadeh et al., VLDB 2009]

- A scalable graph partitioning algorithm
- Based on finding articulation points
  - Articulation point: a vertex whose removal makes the graph disconnected
- Efficient implementations proposed for identifying chatter in the blogosphere [BCKT-VLDB07]



Markov Clustering (MCL) [Dongen-Thesis00]

- Based on simulation of stochastic flow in graphs
  - Graph is mapped to Markov matrix
  - Transition probabilities recomputed through alternate application of two algebraic operations on matrices
    - *Expansion* and *Inflation*



- Clusterings with different scales of granularity by varying the inflation parameter of the algorithm

- Optimized implementation that makes the algorithm scalable

Courtesy of Oktie Hassanzadeh

Correlation Clustering [BBC-ML04]

- Original problem: a graph clustering given edges labelled with ‘+’ or ‘−’
  - ‘+’ indicates correlation between the nodes
  - ‘−’ indicates uncorrelated nodes
- The goal is to find a clustering that agrees as much as possible with the edge labels
  - NP-Hard: approximations needed
- The labels can be assigned to edges based on the similarity scores of the records (edge weights) and a threshold value
- Several approximations exist
  - We use algorithm Cautious from [BBC-ML04] in our paper

Courtesy of Oktie Hassanzadeh

Summary of Experimental Results

[Hassanzadeh et al., VLDB 2009]

	Scalability (Current Implementations)	Ability to find the correct number of clusters	Robustness Against		
			Choice of threshold	Amount of Errors	Distribution of errors
Partitioning	High	Low	Low	Low	High
CENTER	High	High	Low	Low	High
MERGE CENTER	High	High	Low	Low	High
Star	Medium	High	Low	Low	High
SR	Low	Medium	High	High	Low
BSR	Low	Low	High	High	Low
CR	Low	High	Medium	High	High
OCR	Low	High	Medium	High	Low
Correlation Clustering	Low	High	Low	Low	High
Markov Clustering	High	High	Medium	Medium	High
Cut Clustering	Low	Low	Low	Low	High
Articulation Point	High	Medium	Low	Low	High

Courtesy of Oktie Hassanzadeh

Main Conclusions [Hassanzadeh et al., VLDB 2009]

- None of the clustering algorithms produces perfect clustering
- Transitive closure:
  - highly scalable, but results in poor quality of duplicate groups
  - Poor quality even wrt other single-pass algorithms

- Most algorithms are robust to distribution of duplicates, except Ricochet algorithms:
  - high performance over uniformly distributed duplicates
  - poor performance otherwise

- Cut clustering and Correlation clustering:
  - sophisticated & popular algorithms
  - achieve lower accuracy than some single-pass algorithms

- Markov clustering:
  - very efficient
  - one the most accurate algorithms

Courtesy of Oktie Hassanzadeh

Part 8:

Massive Parallelization Methods

Massive Parallelization Outline

- Based on the Map-Reduce paradigm
  - Data partitioned across the nodes of a cluster
  - Map Phase: transforms a data partition into (key, value) pairs
  - Reduce Phase: processes pairs with the same key
- Parallelization of Blocking
  - Standard Blocking (Dedoop)
- Parallelization of Block Processing
  - Block Filtering
  - Meta-blocking
- Parallelization of Entity Matching
  - LINDA

Parallel Standard Blocking [Kolb et al., PVLDB 2012]

MAP function pseudo-code
1: **Input** Key: id of entity e_i, i Value: entity profile of e_i 2: **Output** Key: blocking key value, bkv Value: entity profile of e_i 3: $bkv =$ extractBlockingKey(e_i) 4: emit(bkv , $k.\|\|b_k\|\|$);

REDUCE function pseudo-code
1: **Input** Key: blocking key value, bkv Value: list of entity profiles $V = \{e_i, e_j, \dots ,\}$ 2: **Output** Key: pair of concatenated entity ids, $i.j$ Value: *true* (match) or *false* (non-match) 3: for each pair of entities e_i-e_j in V 4: decision = compareProfiles(e_i , e_j) 5: emit($i.j$, decision); 6: end loop

Parallel Block Filtering [Efthymiou et. al., BigData 2015]

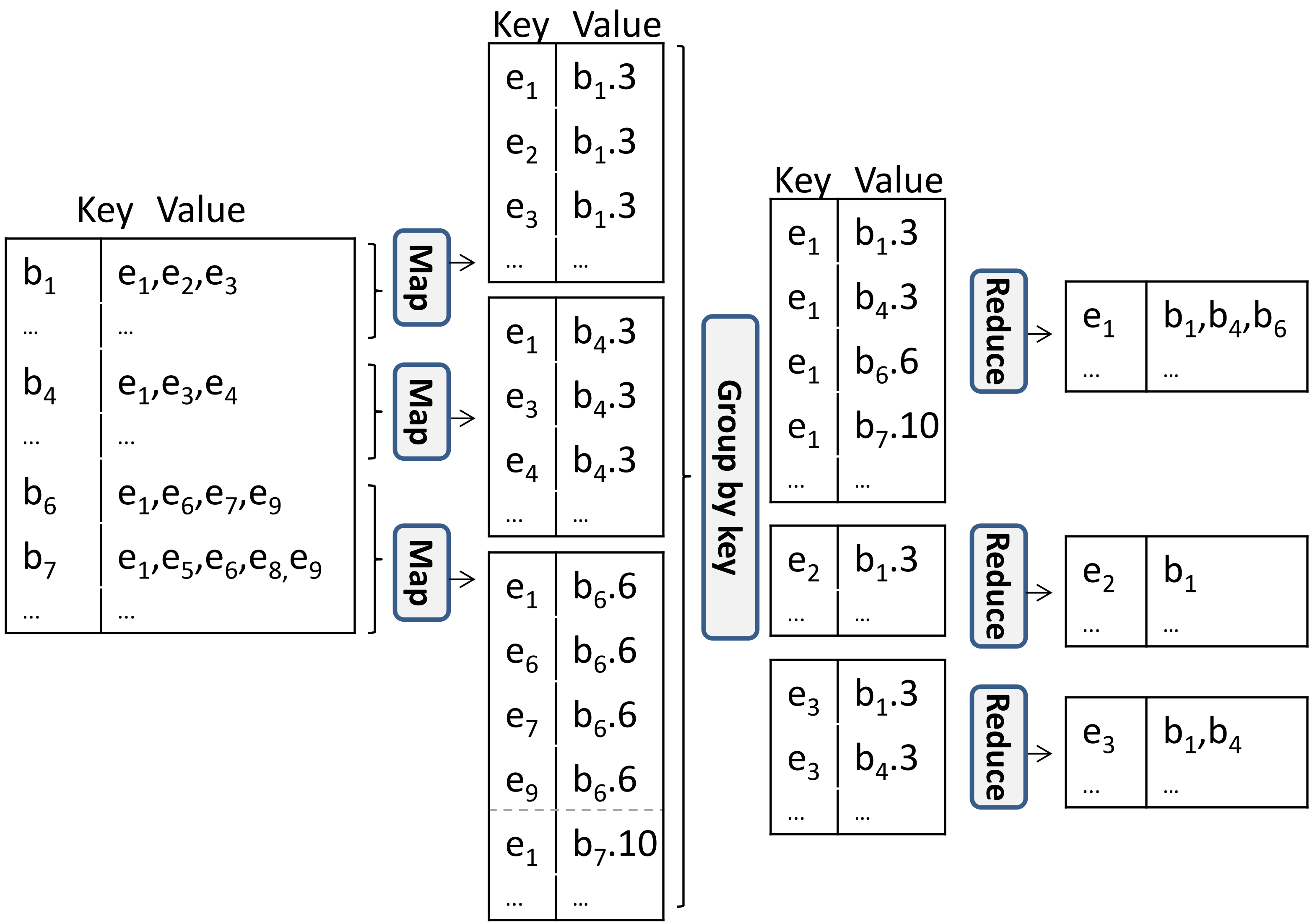
MAP function pseudo-code

```
Input
    Key: id of block b_k, k
    Value: list of entity ids, b_k = {i, j, ..., m}
Output
    Key: id of entity e_i, i
    Value: block id and cardinality, k.||b_k||
compute comparisons in block, ||b_k||
for each i ∈ b_k loop
        emit( i , k.||b_k|| );
end loop
```

REDUCE function pseudo-code

```
Input
    Key: id of entity e_i, i
    Value: list of pairs < k.||b_k|| >, V
Output
    Key: id of entity e_i, i
    Value: list of top-N blocks in B_i, B'_i
order V in ascending block cardinality
B'_i = getTopNBlockIds(V)
emit( i , B'_i );
```

Example Parallel Block Filtering



Parallel Meta-blocking [Efthymiou et al., IS 2017]

- Three strategies:
  1. **Edge-based:** explicitly creates the blocking graph
     - needs pre-processing to perform all weight computations
     - stores all edges of the blocking graph on disk
     - at least 2 MapReduce jobs per pruning algorithm with high I/O
  2. **Comparison-based:** implicitly creates the blocking graph
     - defers weight computations to avoid creating any edges
     - pre-processing enriches the input blocks with the block list of every entity, which is necessary for weight estimation → ideal for CEP/WEP
  3. **Entity-based:** uses the blocking graph only as a conceptual model
     - no pre-processing, all computations in the reducer
     - gathers entire blocks for each entity → ideal for CNP/WNP

Comparison-based Parallel Meta-blocking

Pre-processing

MAP function pseudo-code

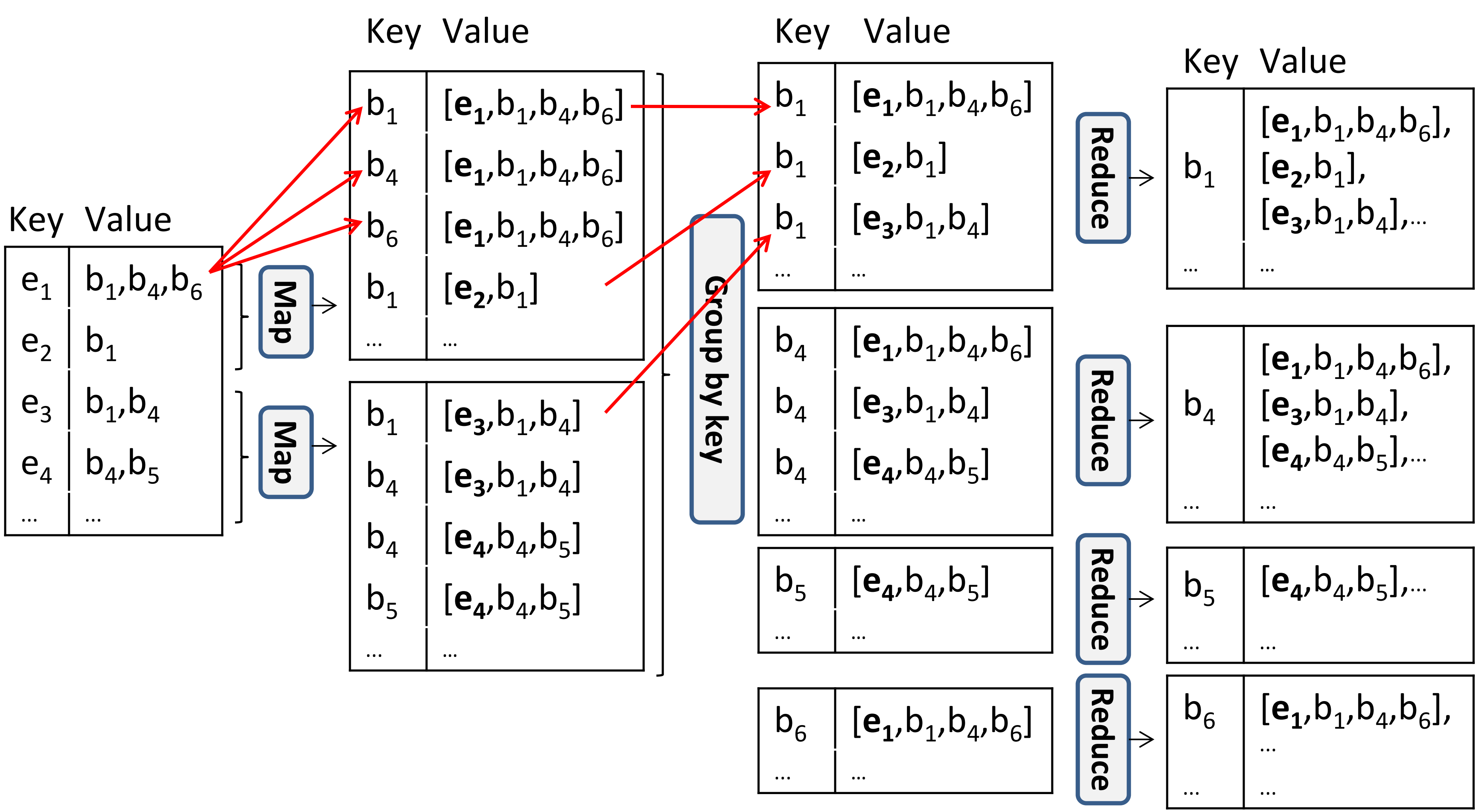
Input
Key: id of entity e_i, i
Value: list of associated block ids, B_i
Output
Key: id of block $b_{k,}$ k
Value: id of entity e_i and associated block ids, $i.B_i$
sort B_i in ascending order of block ids
for each $k \in B_i$ loop
emit(k , $i.B_i$);
end loop

REDUCE function pseudo-code

Input
Key: id of block $b_{k,}$ k
Value: list of pairs $< i.B_i >, V$
Output
Key: input key
Value: input value
if ($2 \leq |V|$)
emit(k , V);

Comparison-based Parallel Meta-blocking

Pre-processing Example



Comparison-based Parallel Meta-blocking
Weighted Edge Pruning (WEP) & Jaccard Scheme (JS)

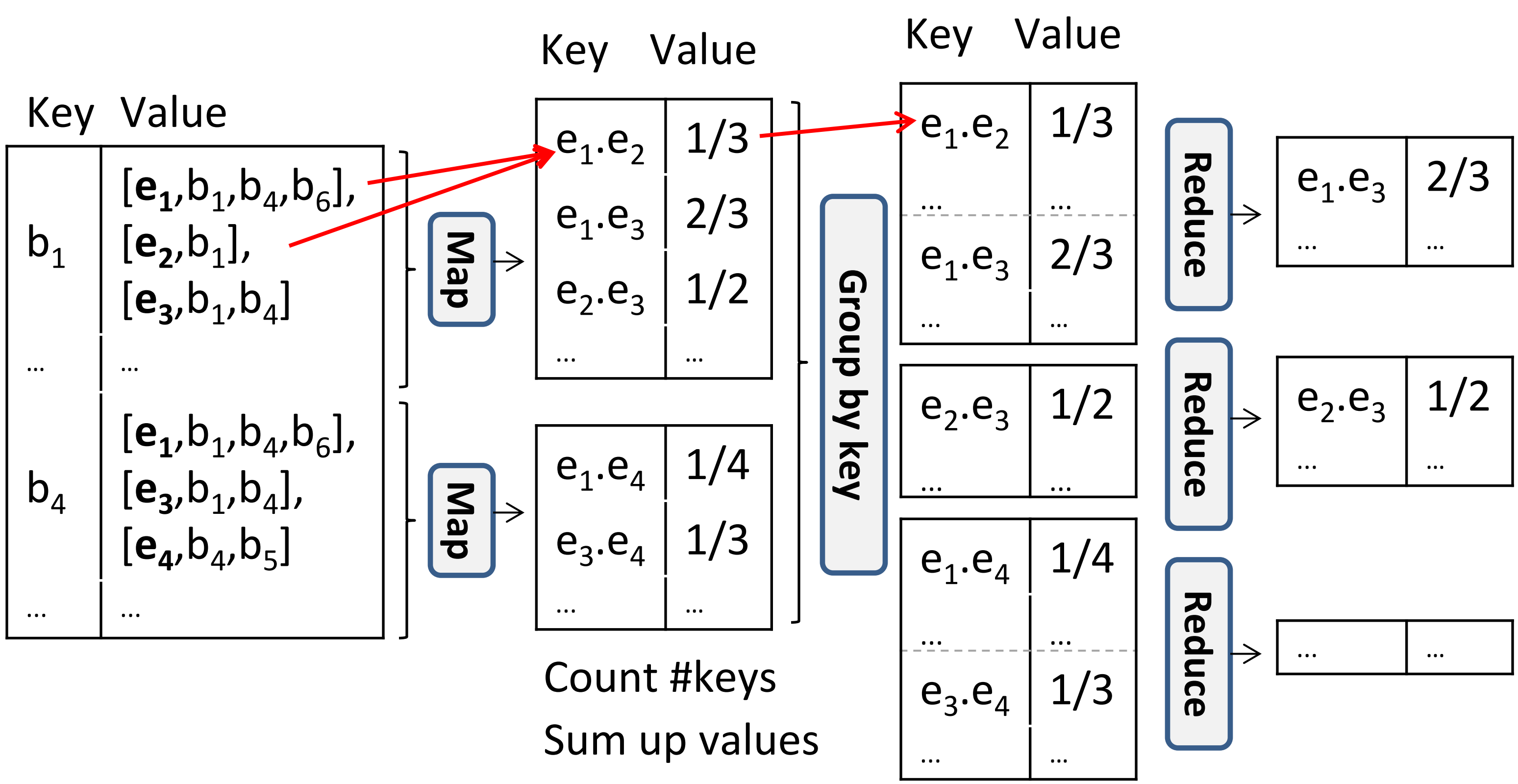
MAP function pseudo-code

Input
Key: id of block b_k, k
Value: list of entity ids, associated blocks and local information, $V = \{i.B_i.X_i, j.B_j.X_j, \ldots\}$
Output
Key: entity ids defining edge <n_i,n_j>, $i.j$
Value: total weight of <n_i,n_j>, w_{ij}
for each $c_{ij} \in b_k.comparisons()$ loop
    if ($isNonRedundant(c_{ij})$ = true)
        compute w_{ij} from $B_i.X_i$, $B_j.X_j$;
        emit($i.j$, w_{ij});
        $|E_G|$++;
        $tw \mathrel{+}= w_{ij}$;
end loop

REDUCE function pseudo-code

Input
Key: entity ids defining edge <n_i,n_j>, $i.j$
Value: total weight of <n_i,n_j>, w_{ij}
Output
Key: entity ids of retained edge <n_i,n_j>, $i.j$
Value: total weight of <n_i,n_j>, w_{ij}
if ($w_{ij} > tw/|E_G|$)
    emit($i.j$, w_{ij});

Comparison-based Parallel Meta-blocking
Weighted Edge Pruning (WEP) & Jaccard Scheme (JS)



Entity-based Parallel Meta-blocking

Cardinality Node Pruning (CNP)

MAP function pseudo-code

Input
    Key: id of block b_k, k
    Value: list of entity ids, $b_k = \{i, j, \dots, m\}$
Output
    Key: id of entity e_i, i
    Value: input value
for each $j \in b_k$ loop
        emit (j , b_k);
end loop

REDUCE function pseudo-code

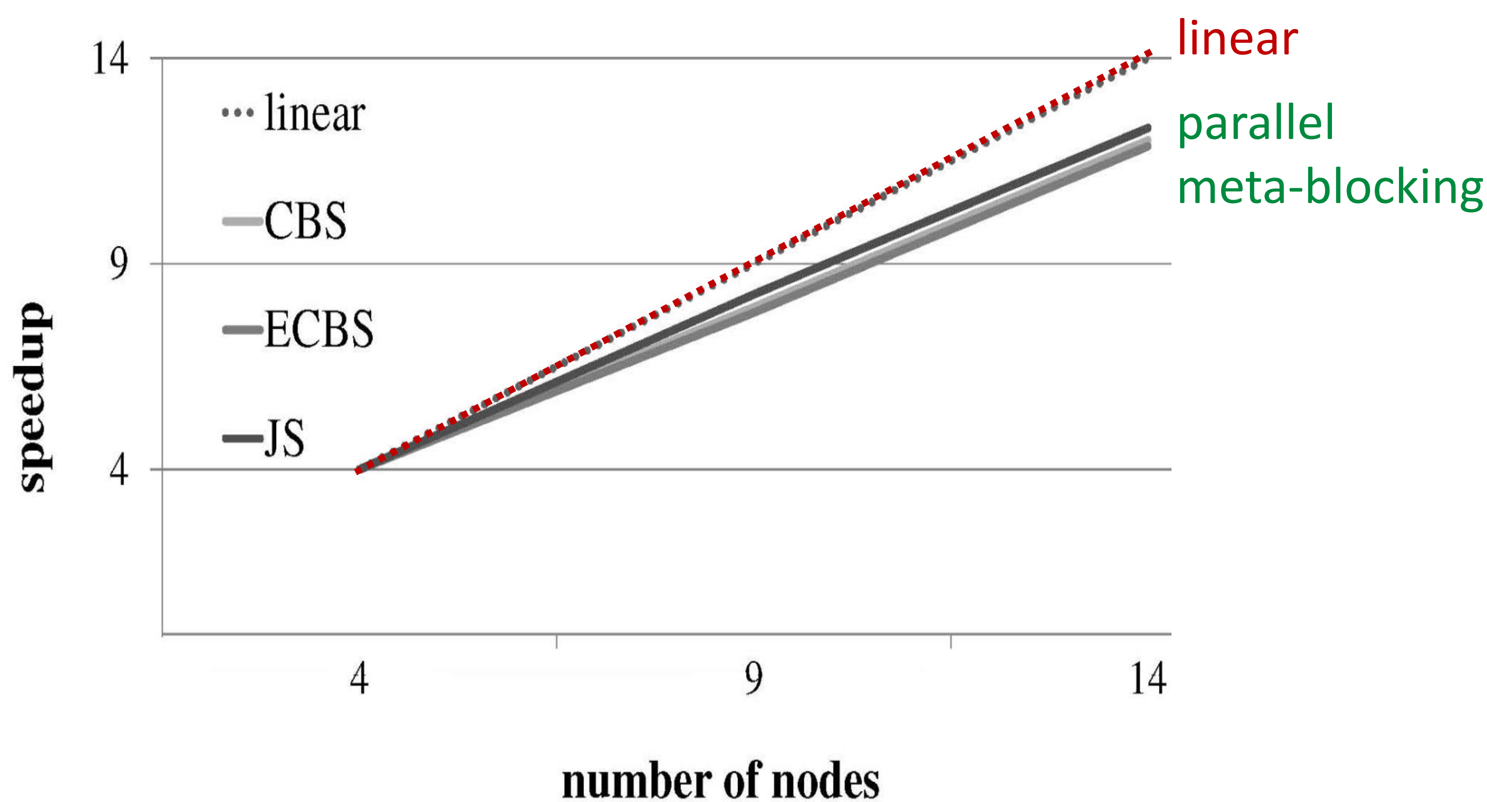
Input
    Key: id of entity e_i, i
    Value: co-occurrence bag, β_i
Output
    Key: entity ids of retained edge <n_i,n_j>, $i.j$
    Value: total weight of <n_i,n_j>, w_{ij}
frequencies[] ← {}; setOfNeighbors ← {};
for each $j \in V$ loop
        frequencies[j]++;
        setOfNeighbors .add(j);
end loop
topEdges ← {};
for each $j \in$ setOfNeighbors loop
        w_{ij} = getWeight (i , j , frequencies[j]);
        topEdges.add(j , w_{ij});
        if (topEdges.size() < k)
                topEdges.pop();
end loop
for each (j , w_{ij}) ∈ topEdges loop
        emit ($i.j$, w_{ij});
end loop

Load Balancing: MaxBlock Algorithm

Parallel Meta-blocking Performance

		DB_C		
Block Filtering	Basic	2		
	Advanced	2		
		Edge Based	Comp. Based	Entity Based
CEP	ARCS	252	89	184
	CBS	222	55	145
	ECBS	240	78	210
	JS	223	60	190
	EJS	1,996	116	>6,000
CNP	ARCS	554	370	73
	CBS	491	301	74
	ECBS	555	383	76
	JS	534	363	83
	EJS	2,645	430	142
WEP	ARCS	203	65	389
	CBS	220	50	123
	ECBS	219	54	123
	JS	219	54	132
	EJS	1,993	81	204
WNP	ARCS	562	363	63
	CBS	498	304	65
	ECBS	568	389	73
	JS	553	373	74
	EJS	2,626	411	142

Comparison-based Parallel Meta-blocking
Weighted Edge Pruning (WEP) & Jaccard Scheme (JS)



Parallel Meta-blocking achieves (almost) linear scale-up

LINDA: Parallel Entity Matching as an Optimization Problem [Böhm et al., CIKM 2012]

- Input:
  - Entity Graph ***G=(V,E)*** where vertices represent distinct URIs
- Output:
  - Assignment Matrix ***X*** where $x_{a,b}=1$ if ***a*** and ***b*** refer to same entity
- Constraints:
  - reflexivity, symmetry, transitivity, and unique mapping per source (entity from source 1 matches with at most one entity from source 2)
- Objective:
  - Given ***sim(a, b, G, X)***, find *X* that maximizes

$$\sum_{a,b \in V : S(a) \neq S(b)} x_{a,b} \ \mathrm{sim}(a, b, G, \boldsymbol{X})$$

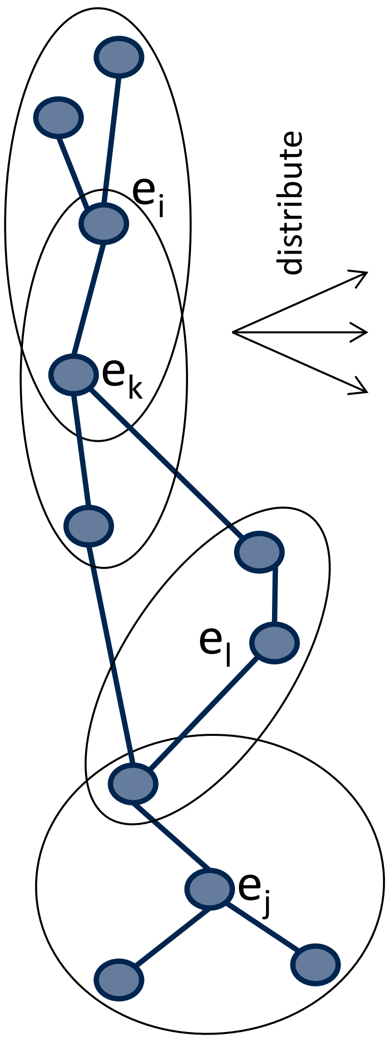
Courtesy of Gerhard Weikum

Multi-Core Assignment Algorithm

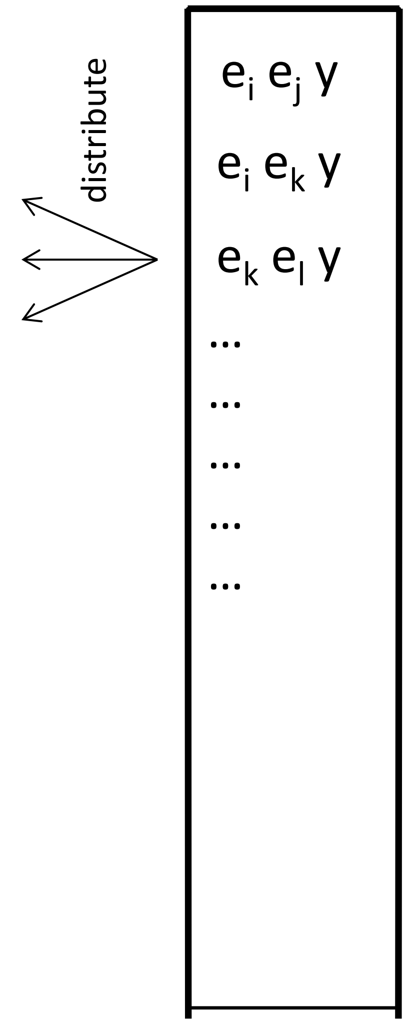
- Initialize matrix X as identity matrix
- Initialize priority queue Q with initial similarities
- While Q not empty
  - Retrieve pair (a,b) of entities with highest similarity value
  - Accept pairs (a',b') with a' and b' in equivalence class of a and b (uses current X to determine equivalents, updates X)
  - For all pairs (c,d) of entities where similarity could have changed
    - Compute new similarities **in parallel**
    - If similarity has changed for (c,d)
      - Update queue with new similarity for (c,d)
- Return X

Map/Reduce Assignment Algorithm

Entity Graph

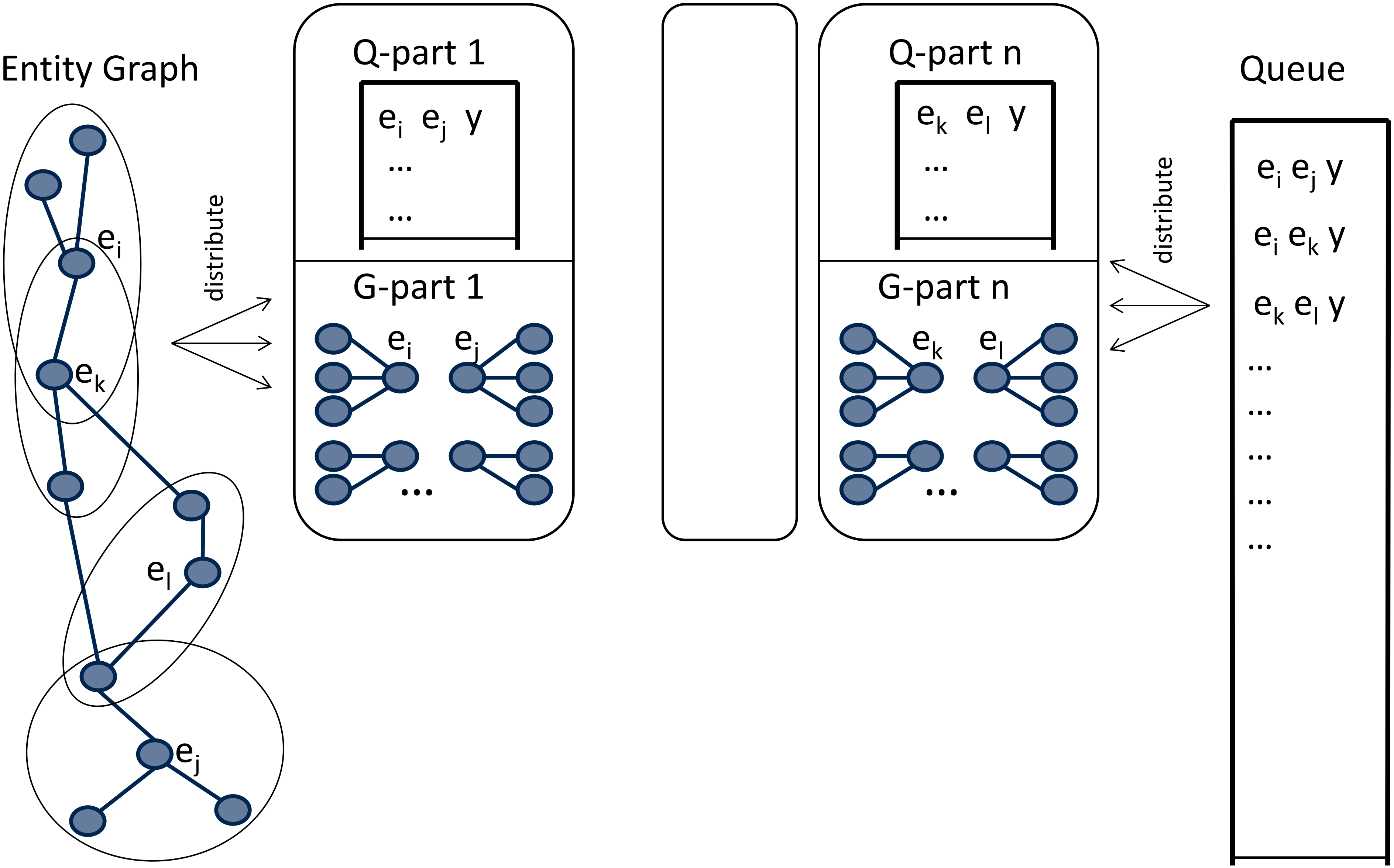


Queue



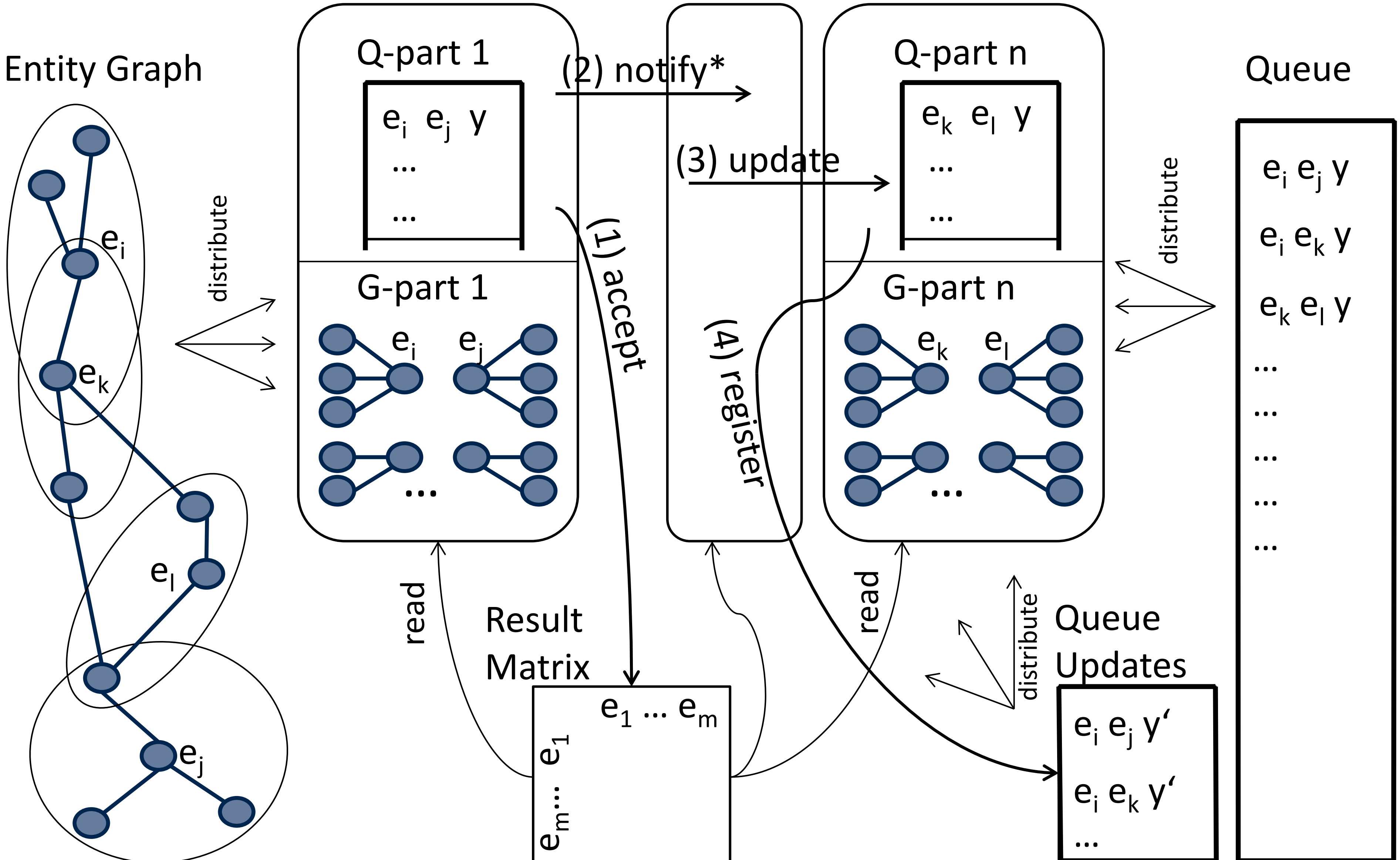
Courtesy of Gerhard Weikum

Map/Reduce Assignment Algorithm



Courtesy of Gerhard Weikum

Map/Reduce Assignment Algorithm



Courtesy of Gerhard Weikum

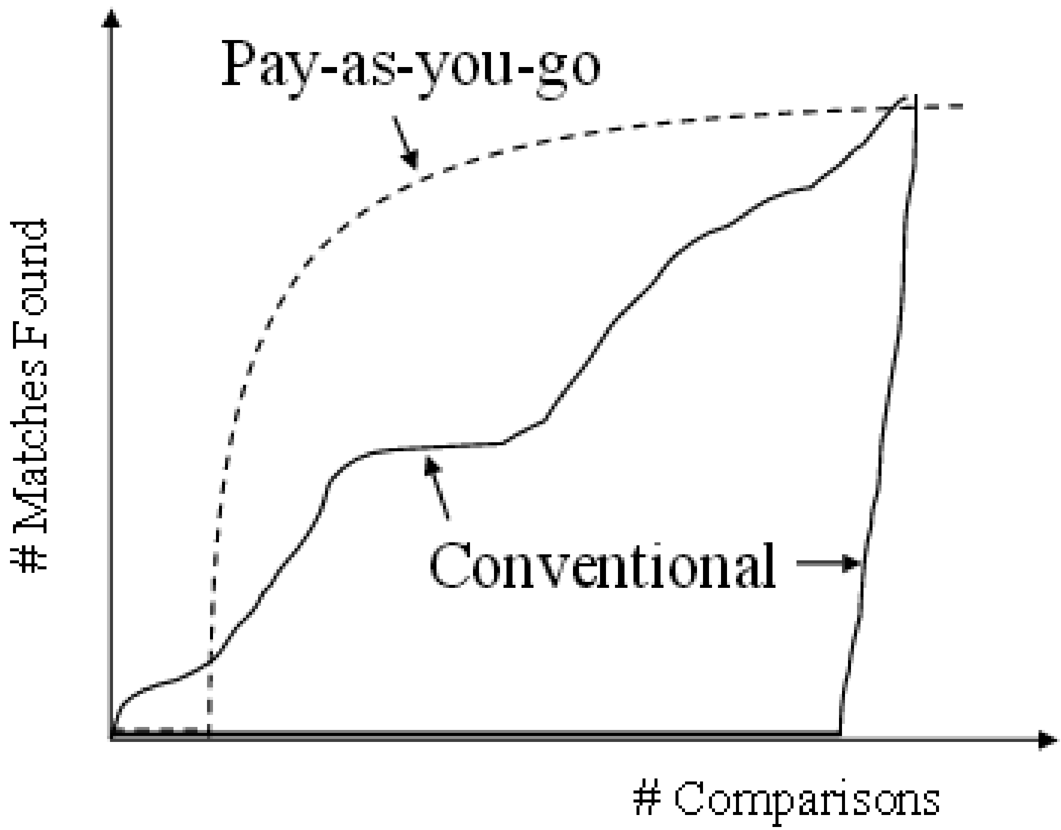
Part 9:

Progressive Entity Resolution

Preliminaries

Facts:

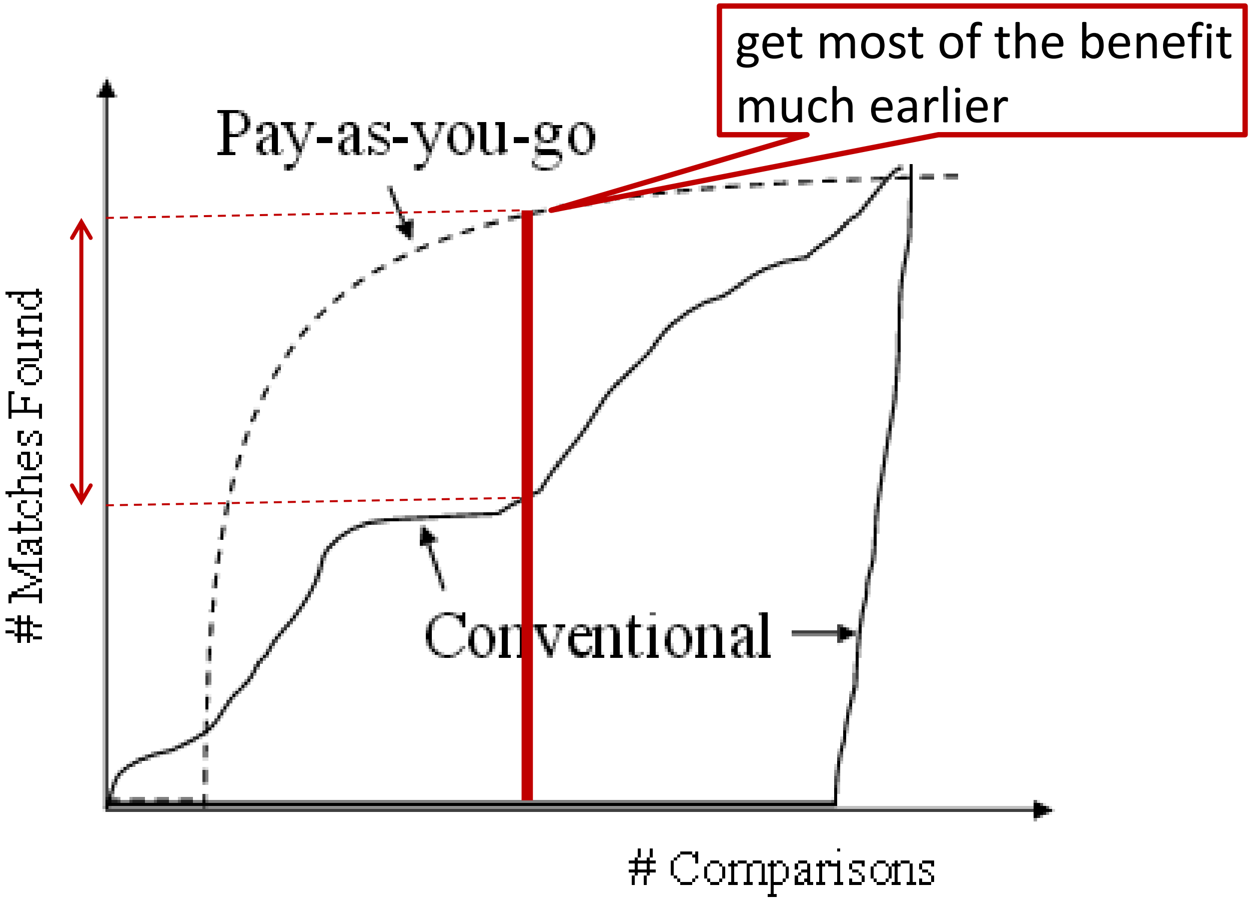
- Progressive, or Pay-as-you-go ER comes is useful



Preliminaries

Facts:

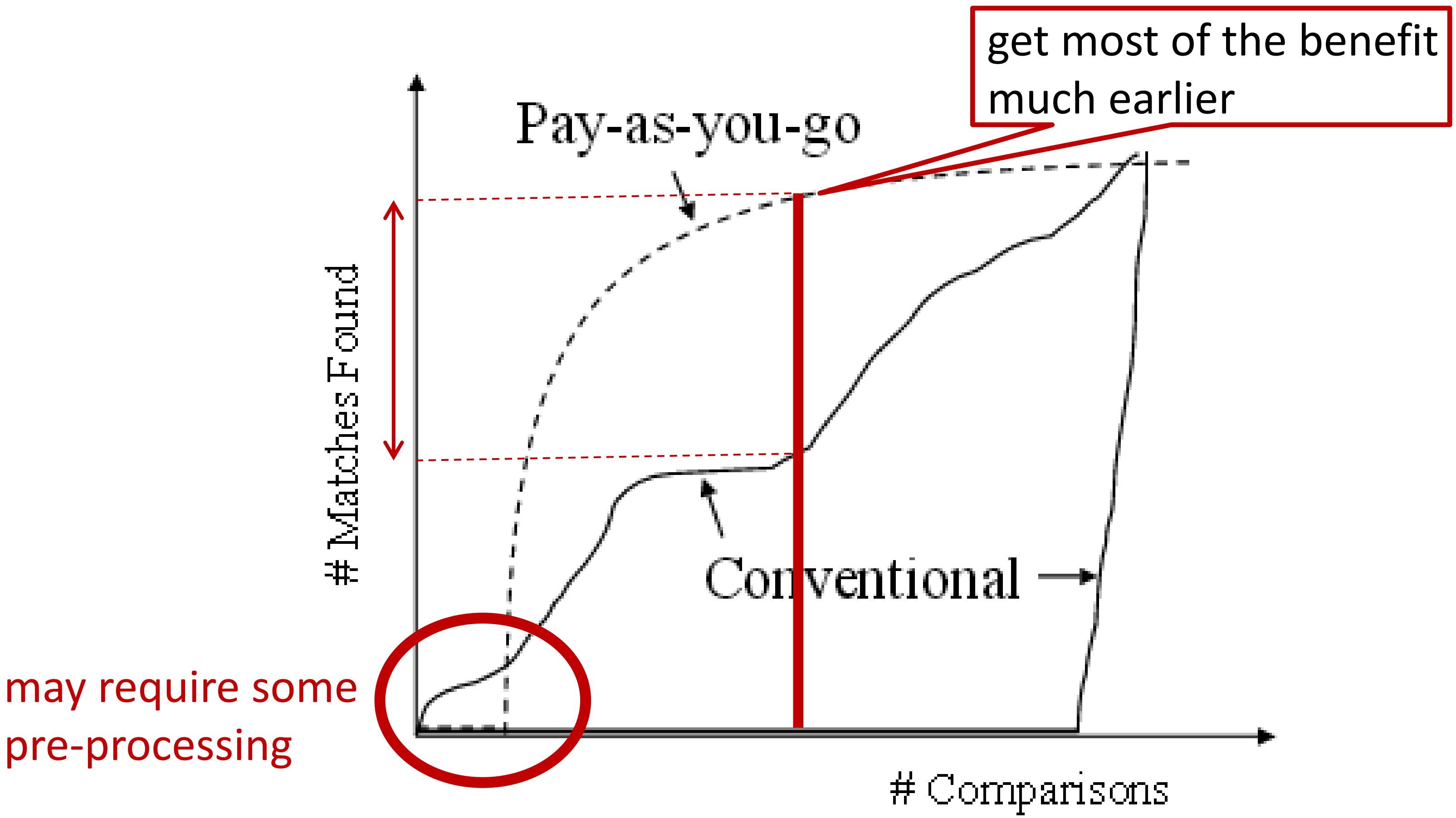
- Progressive, or Pay-as-you-go ER comes is useful



Preliminaries

Facts:

- Progressive, or Pay-as-you-go ER comes is useful



Progressive Entity Resolution

- requires:
  - Improved Early Quality
  - Same Eventual Quality

- defines **optimal processing order** for a set of entities

- Use cases:
  - Limited, unknown time for ER (online ER)
  - Exploratory ER

- Empirical by nature, based on heuristics

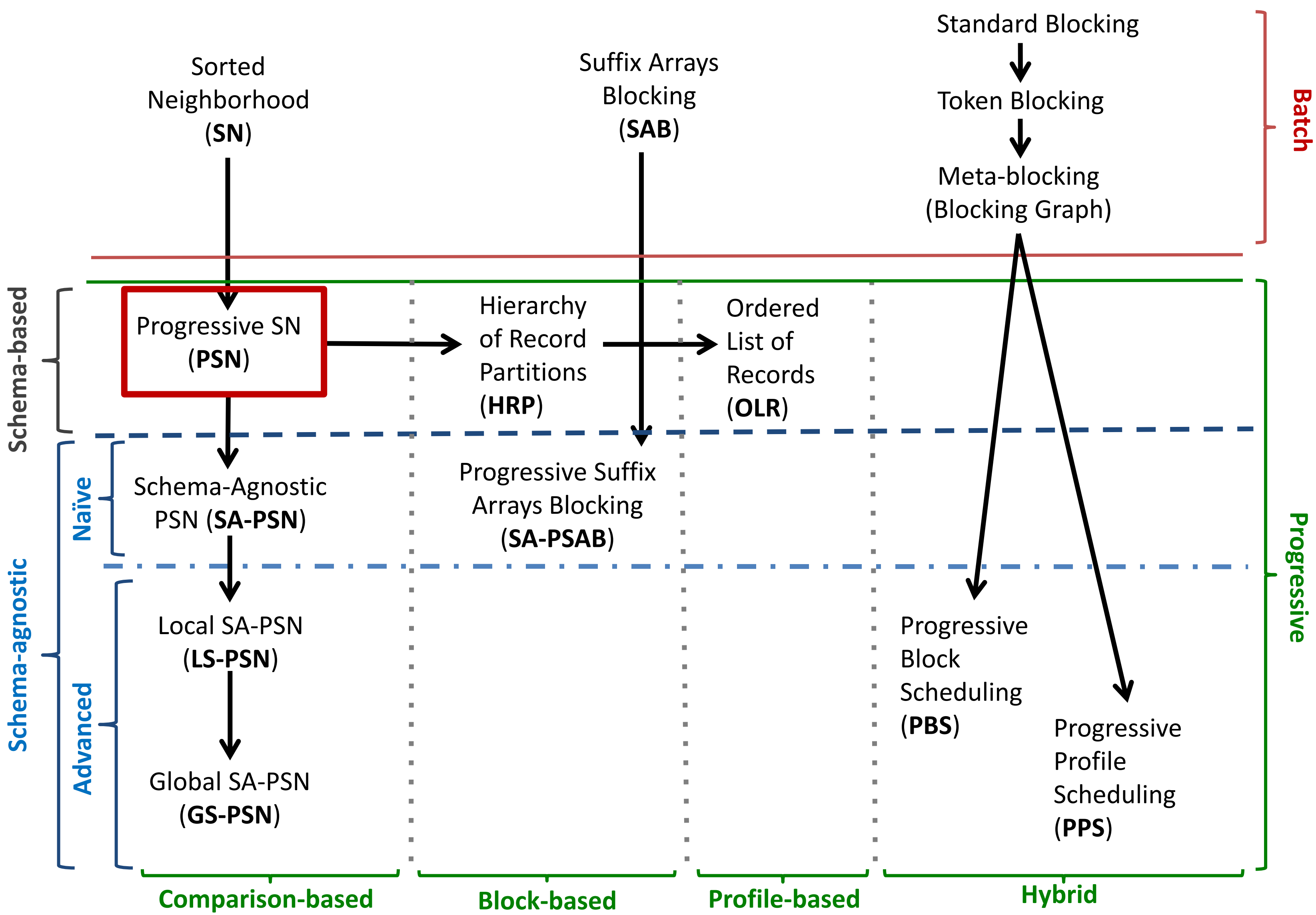
Static Progressive Methods

- Guide which records to compare first, **independently** of Entity Matching results

- Three flavors
  - *Sorted list of pairs*: a list of record pairs, ranked by the likelihood that the pairs match
  - *Hierarchy of partitions*: likely matching records in the form of partitions with different levels of granularity
  - *Sorted list of records*: maximize the number of matching records identified when the list is resolved sequentially

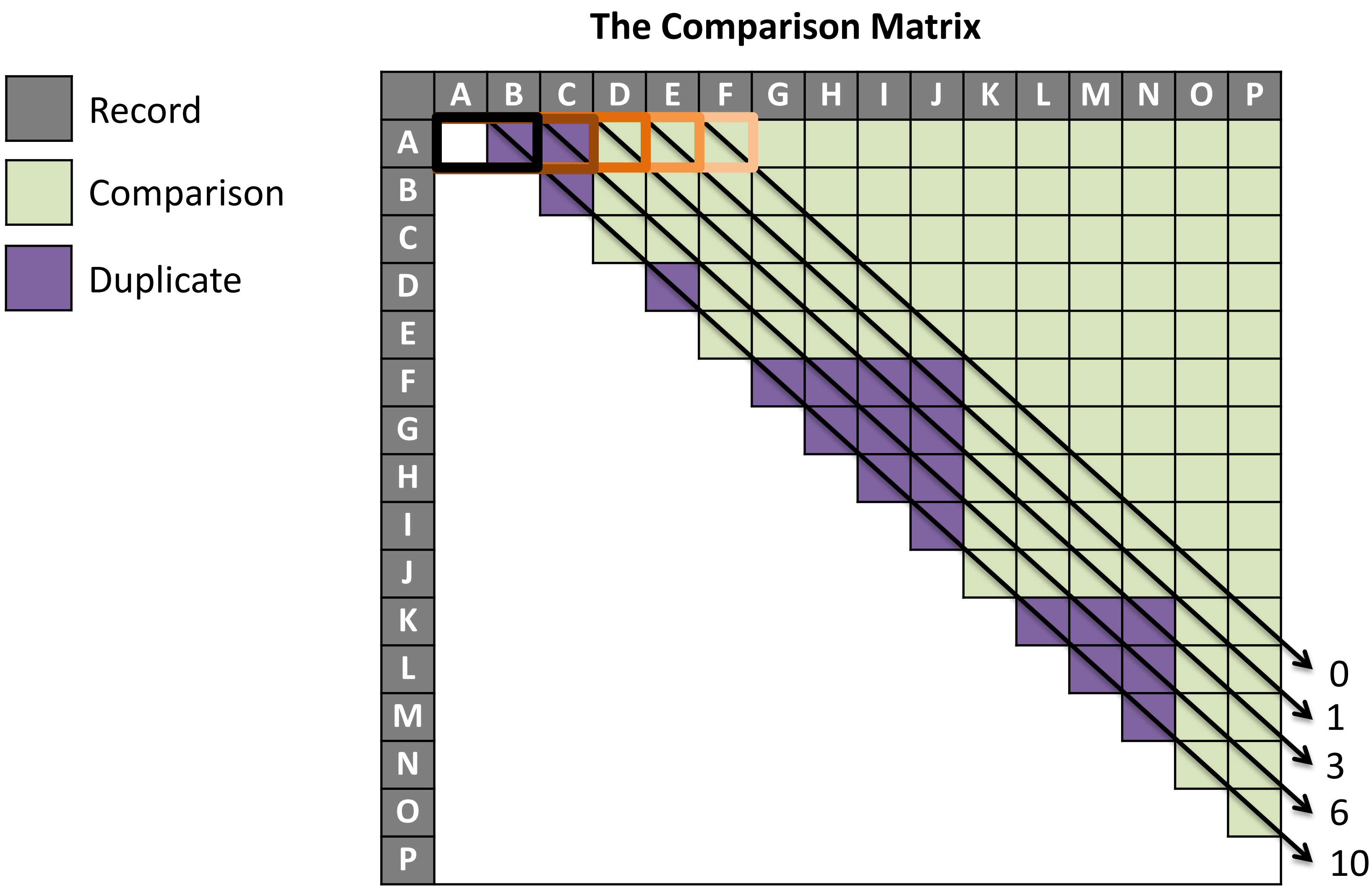
Courtesy of Steven Euijong Whang

Taxonomy of Progressive Methods



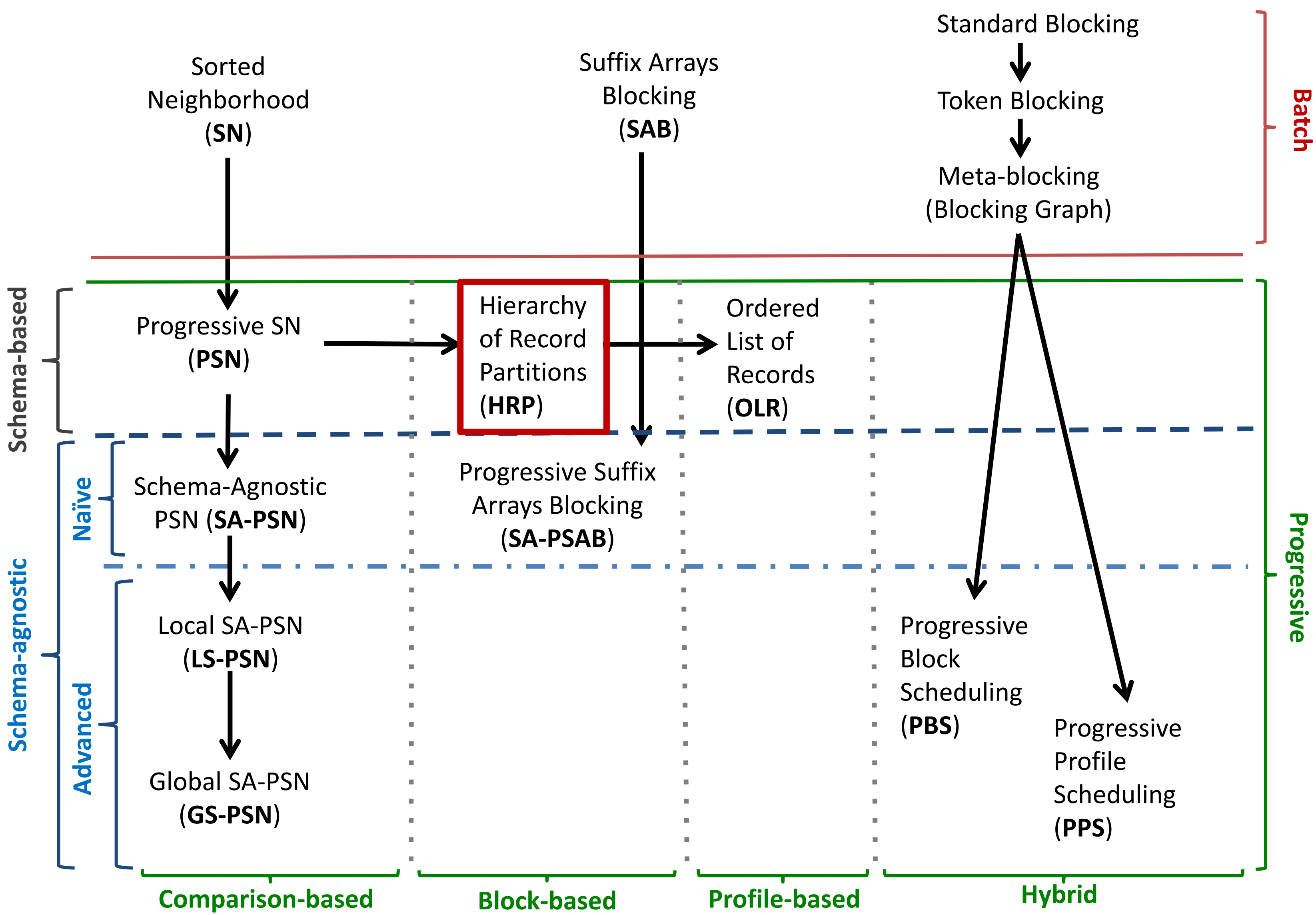
Progressive Sorted Neighborhood (PSN)

[Whang et al, IEEE TKDE, 2013]



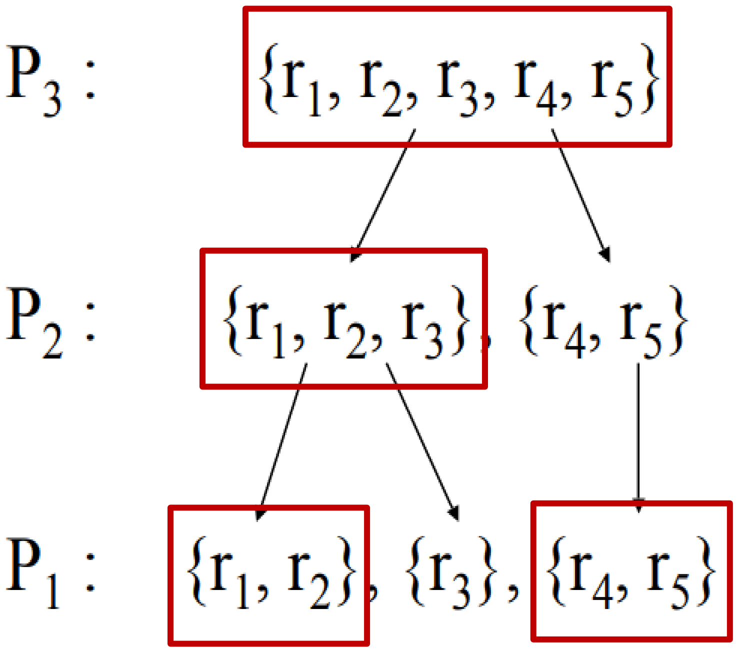
Courtesy of Thorsten Papenbrock

Taxonomy of Progressive Methods



Hierarchy of Partitions

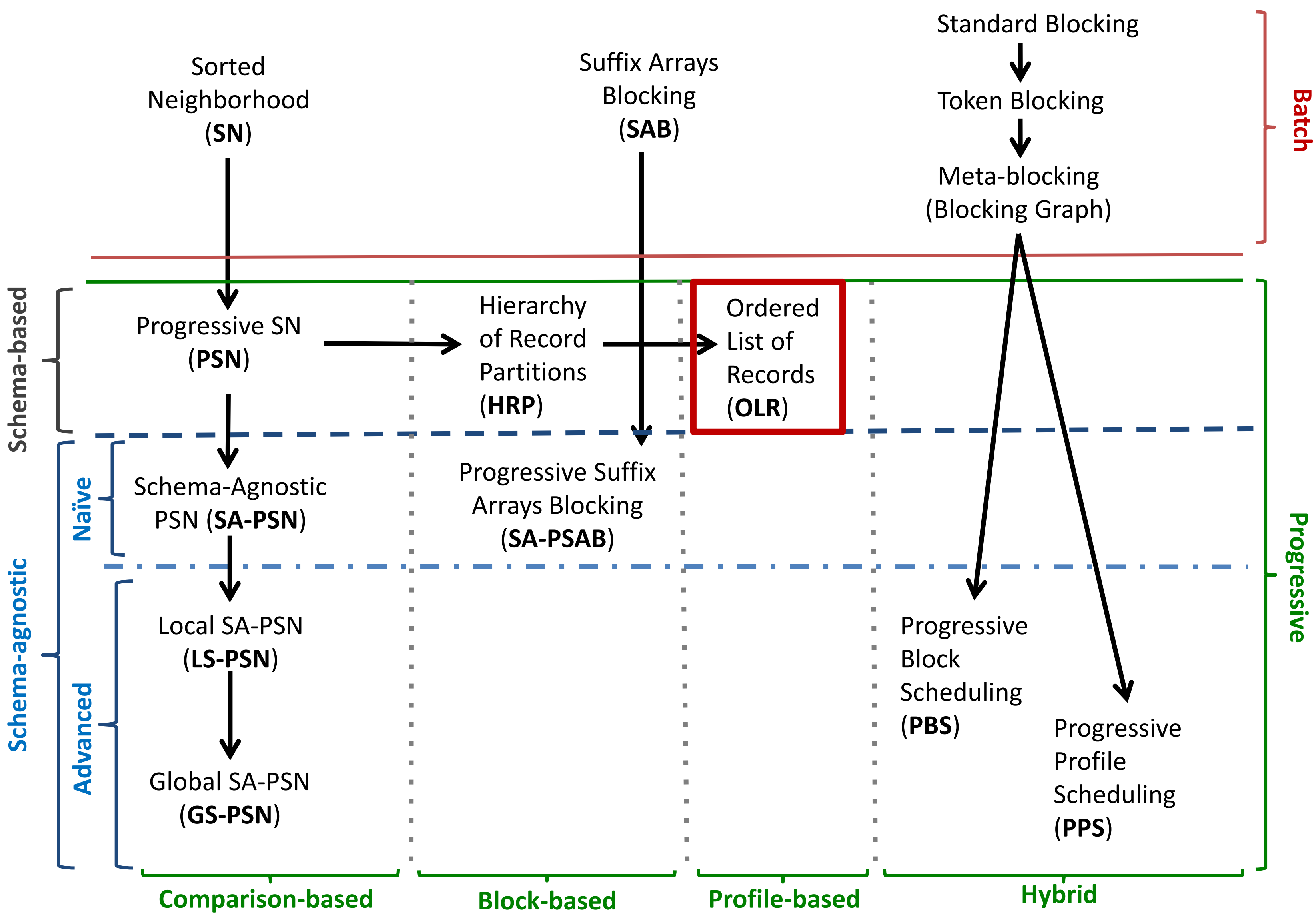
[Whang et al, IEEE TKDE, 2013]



1. Compare $\{r_1, r_2\}$ and $\{r_4, r_5\}$.
2. If there is more budget, compare $\{r_1, r_2, r_3\}$.
3. If there is still more budget, compare $\{r_1, r_2, r_3, r_4, r_5\}$.

Courtesy of Steven Euijong Whang

Taxonomy of Progressive Methods



Ordered List of Records

[Whang et al, IEEE TKDE, 2013]

Goal:

maximize the number of matching records identified while resolving the list sequentially

Advantages:

- zero space requirements
- no change in resolution algorithm

Generation:

Sort all entities according to a weight derived from the partitions that involve them, under the assumption that each partition is equally likely to be the correct ER outcome.

Schema-agnostic Progressive ER

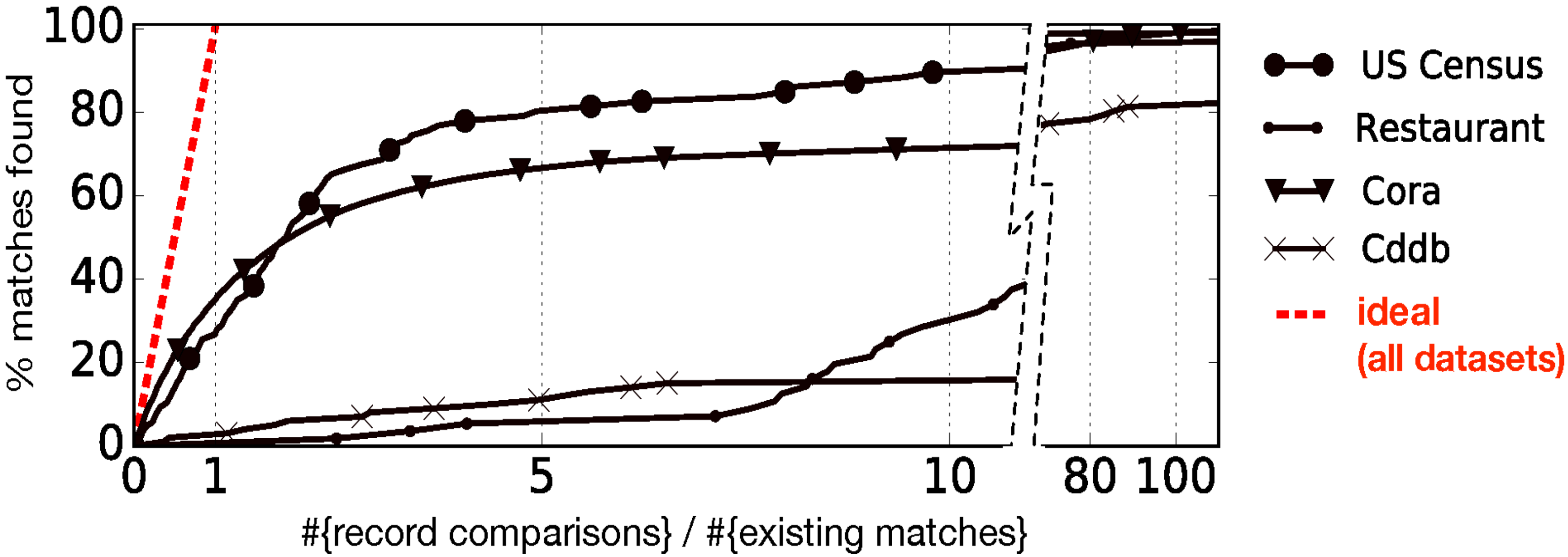
[Simonini et. al., ICDE 2018]

- Previous works assume structured data.
  - (Multiple) Schema-based blocking methods.

- Problems:
  - Inapplicable to Big Data, due to Volume and Variety.
  - Plenty of room for improvement
    - even for schema-based methods!

Schema-agnostic Progressive ER

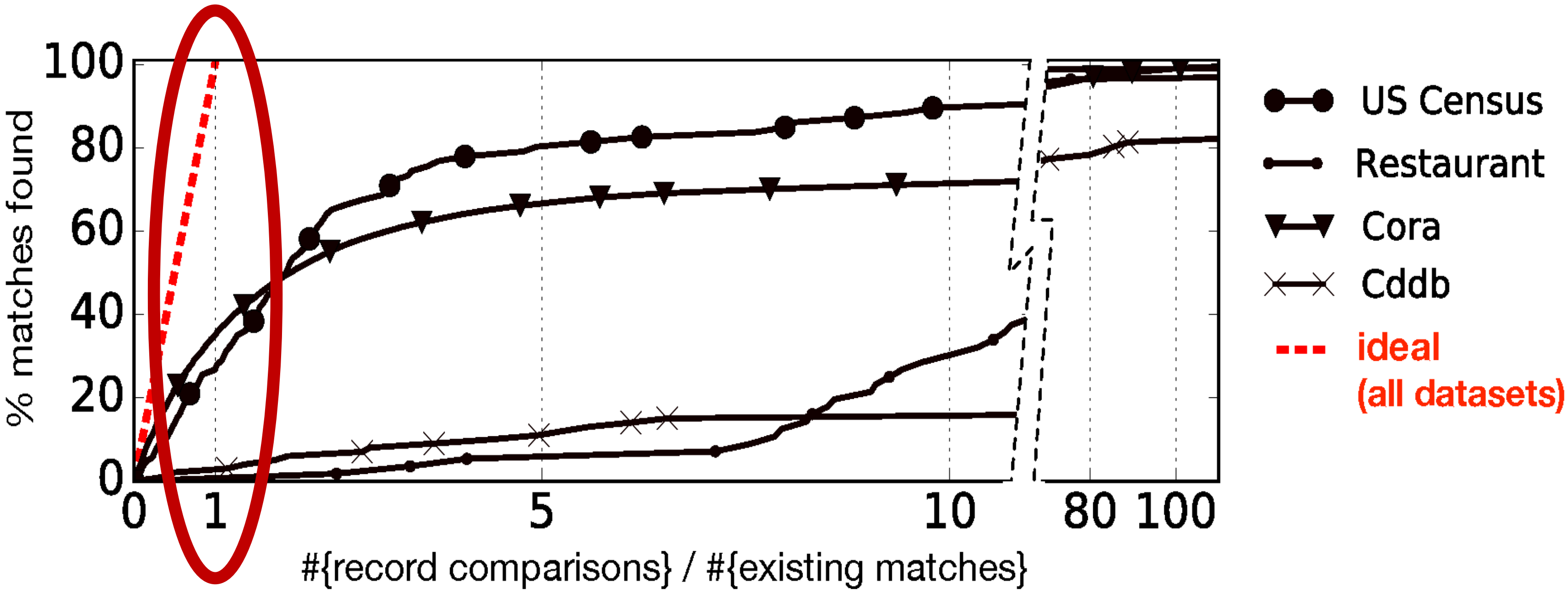
[Simonini et. al., ICDE 2018]



- state of the art schema-based solution: PSN

Schema-agnostic Progressive ER

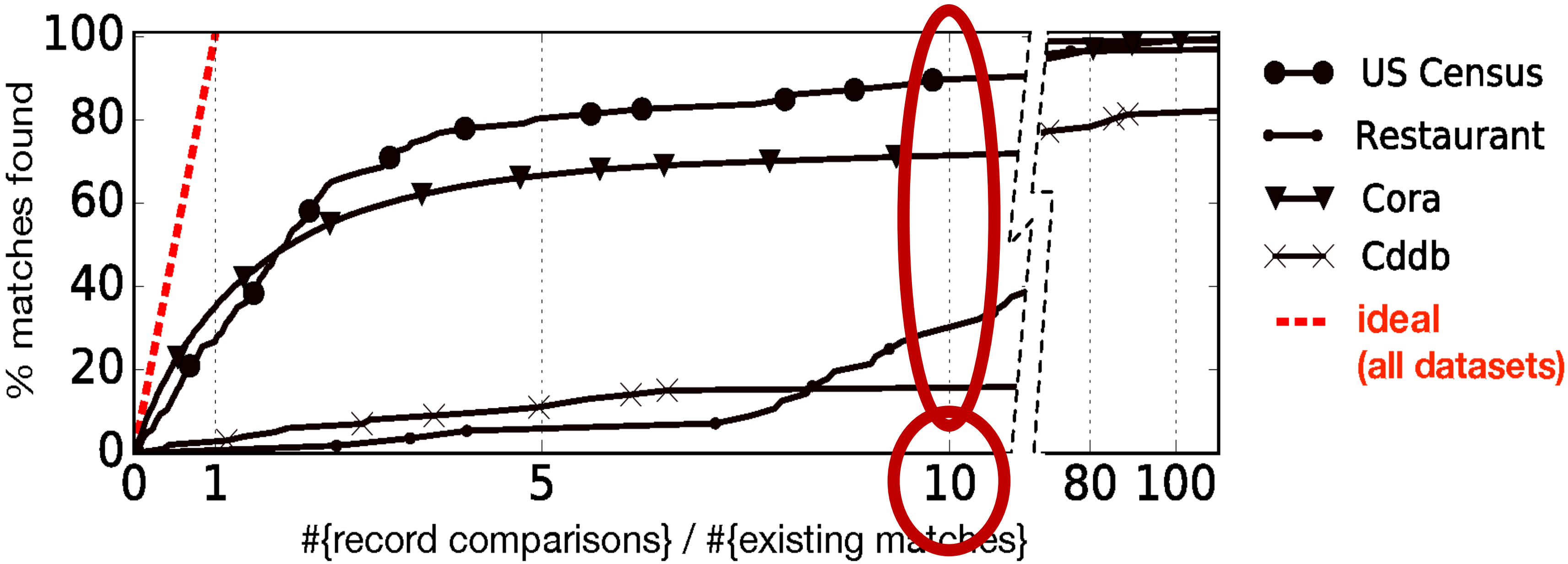
[Simonini et. al., ICDE 2018]



- state of the art schema-based solution: PSN
- when optimal finds 100% of matches, PSN finds 2-35% of matches

Schema-agnostic Progressive ER

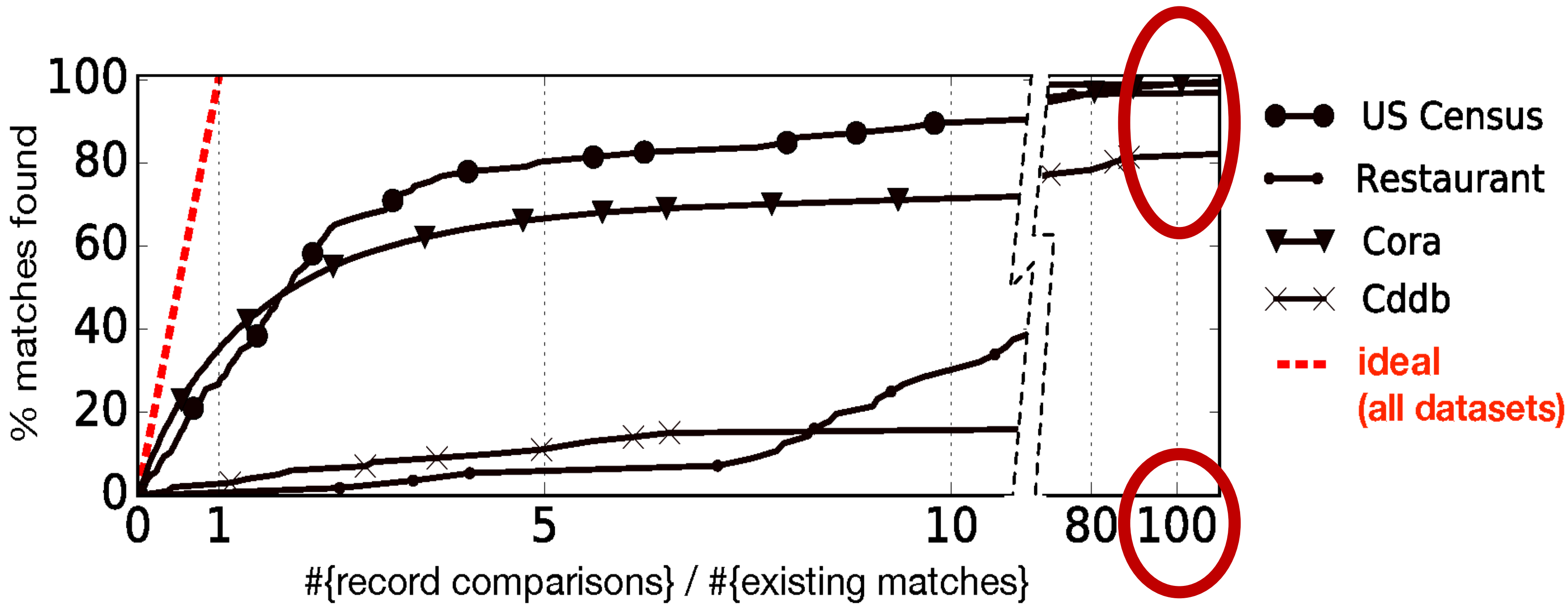
[Simonini et. al., ICDE 2018]



- state of the art schema-based solution: PSN
- when optimal finds 100% of matches, PSN finds 2-35% of matches
- after 10x the comparisons optimal needs, PSN finds 15-85% of matches

Schema-agnostic Progressive ER

[Simonini et. al., ICDE 2018]

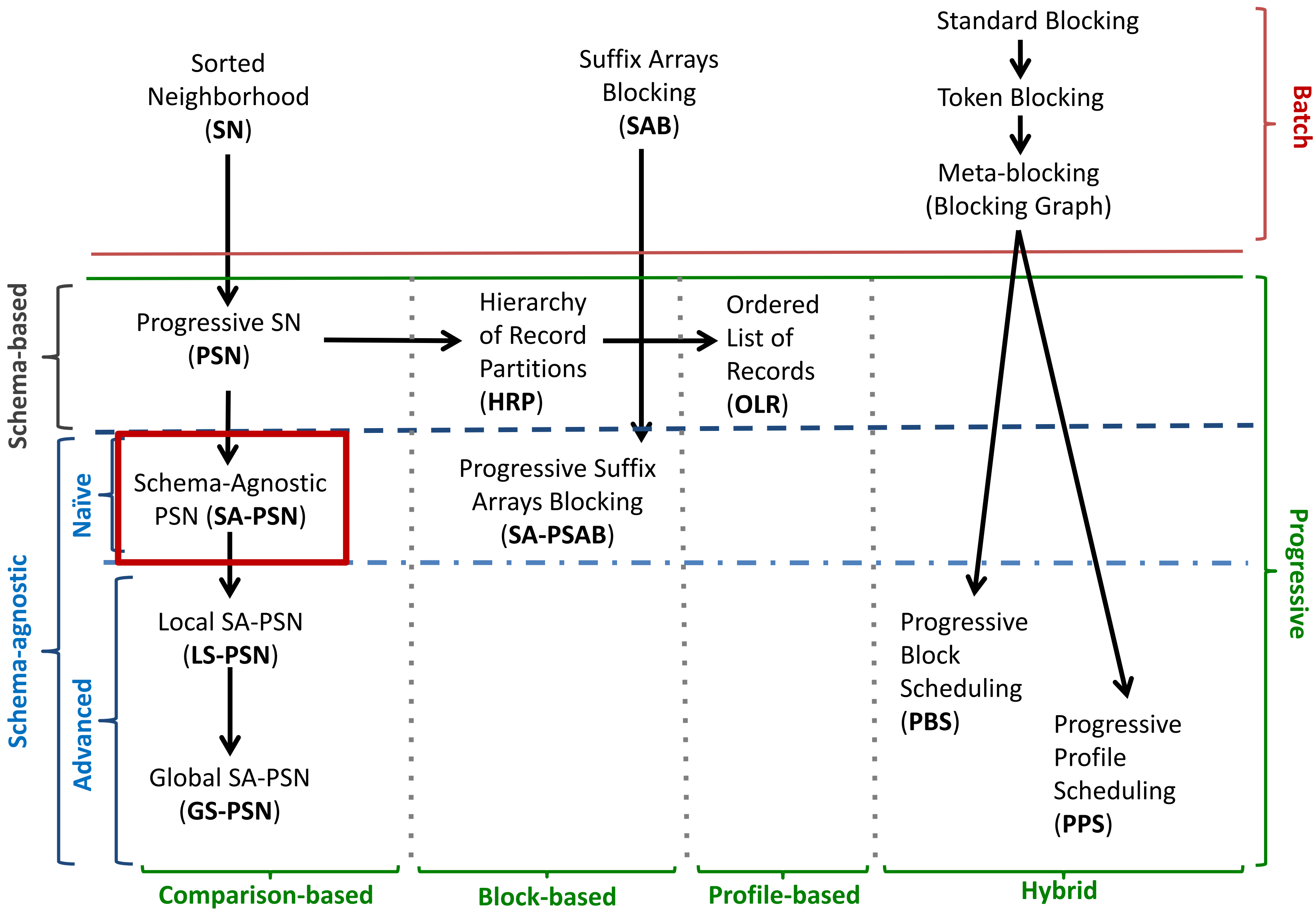


- state of the art schema-based solution: PSN
- when optimal finds 100% of matches, PSN finds 2-35% of matches
- after 10x the comparisons optimal needs, PSN finds 15-85% of matches
- after 100x the comparisons optimal needs, PSN finds 80-99% of matches

Schema-agnostic Progressive ER
[Simonini et. al., ICDE 2018]

- Solution:
  - schema-agnostic methods that are able to handle large, semi-structured, heterogeneous data
  - proposed solutions also applicable to structured data

Taxonomy of Progressive Methods



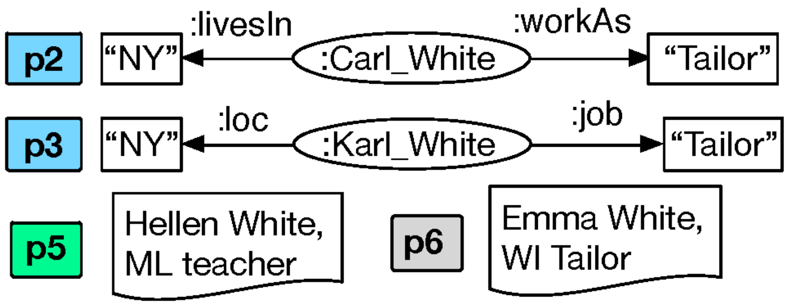
Schema-agnostic PSN [Simonini et. al., ICDE 2018]

- Use all attribute values as blocking keys – regardless of attribute values
- Sort them alphabetically
- Sort the entities accordingly → Neighbor List
  - The same entity might be placed in consecutive places
- Slide the incremental window over the Neighbor List
- Execute the valid comparisons

Example of Schema-agnostic PSN

Data Lake

	Name	Surname	City	Profession
p1	Carl	White	NY	Tailor
p4	Ellen	White	ML	Teacher



Blocking Keys

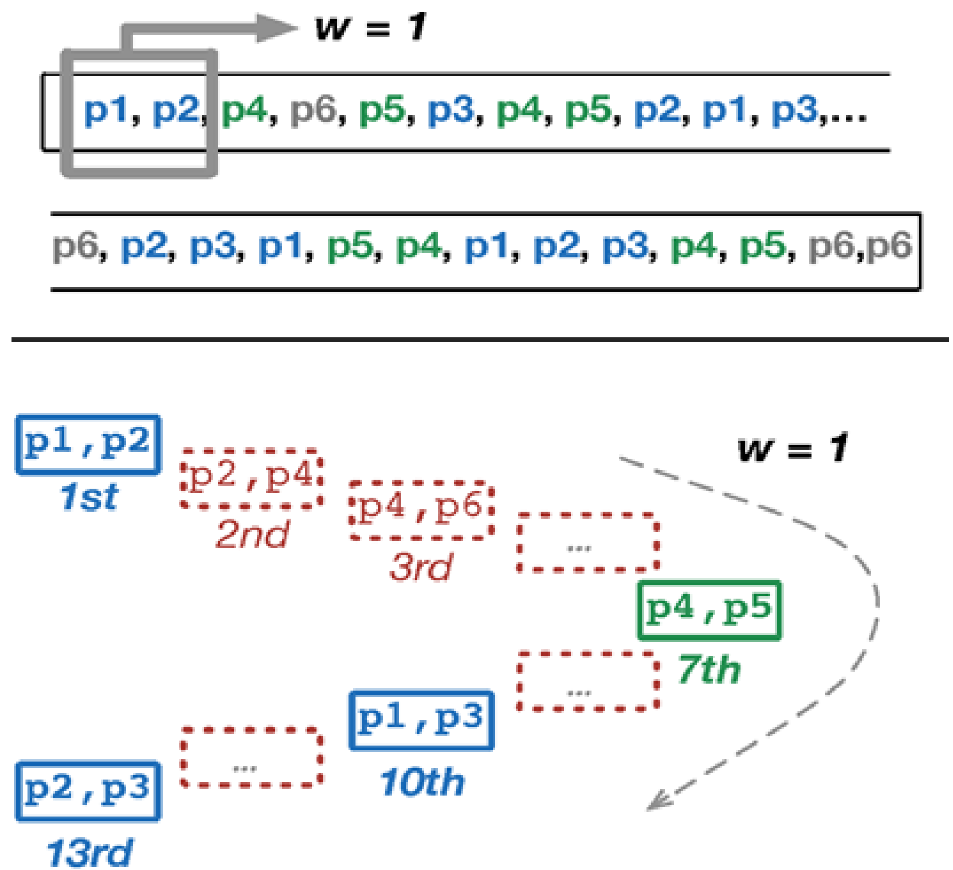
['carl','ellen','emma','hellen',
'karl','ml','ny','tailor',
'teacher','white','wi']

Neighbor List

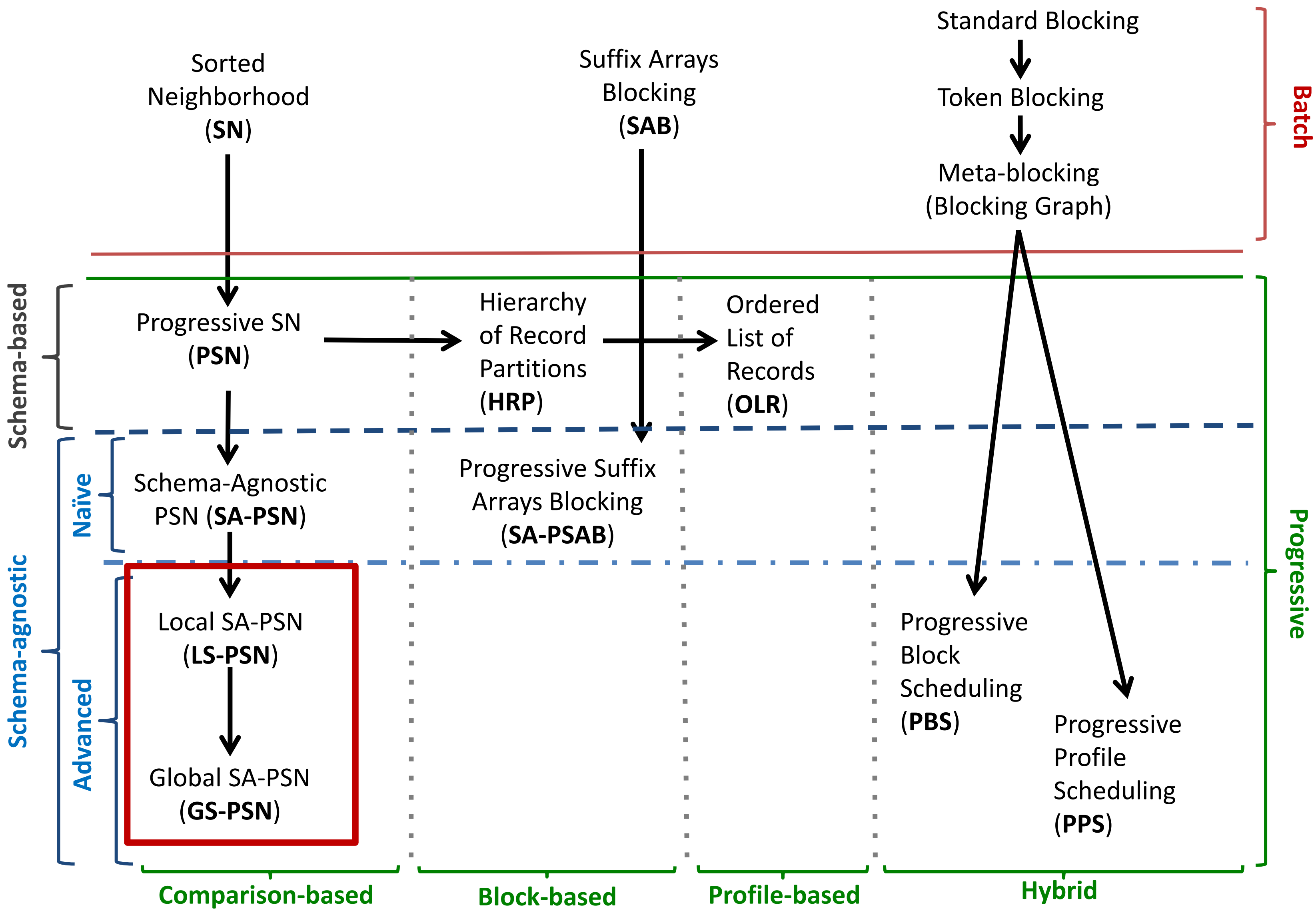
p1, p2, p4, p6, p5, p3, p4, p5, p2, p1, p3,...

p6, p2, p3, p1, p5, p4, p1, p2, p3, p4, p5, p6,p6

Sliding Window



Taxonomy of Progressive Methods



Local/Global Schema-agnostic PSN

[Simonini et. al., ICDE 2018]

- Drawbacks of SA-PSN
  - coincidental proximity → random ordering
  - redundant comparisons

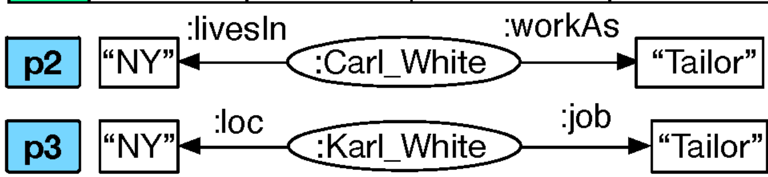
- LS-PSN:
  - discards redundant comparisons within the current window
  - local execution order through comparison weighting with Position Index
  - weighting scheme: Relative Co-occurrence Frequency (RCF)

- GS-PSN:
  - similar to LS-PSN, but defines a global execution order for all comparisons in a range of window sizes up to w_{max}

Example of LS-PSN

Data Lake

	Name	Surname	City	Profession
p1	Carl	White	NY	Tailor
p4	Ellen	White	ML	Teacher



p5 Hellen White, ML teacher

p6 Emma White, WI Tailor

Blocking Keys

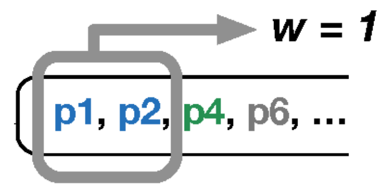
```
['carl','ellen','emma','hellen',
  'karl','ml','ny','tailor',
   'teacher','white','wi']
```

Neighbor List

p1, p2, p4, p6, p5, p3, p4, p5, p2, p1, p3,...

p6, p2, p3, p1, p5, p4, p1, p2, p3, p4, p5, p6,p6

Sliding Window



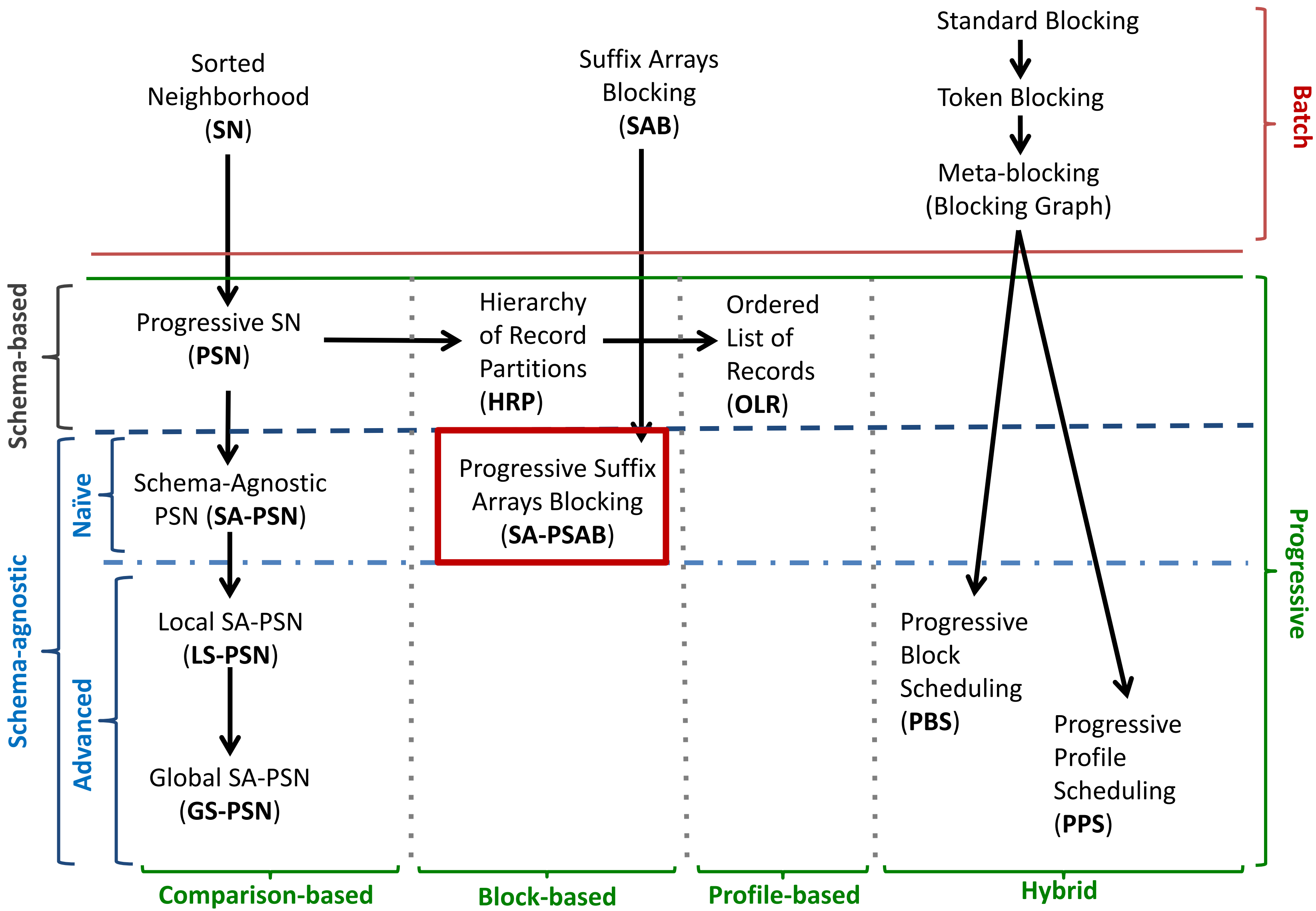
Comparison Weighting

	p2	p3	p4	p5	p6
p1	.6	.3	.1	.1	.1
p2		.6	.1	.1	.1
p3			.3	.1	.1
p4				.6	.1
p5					.3

Comparison Sorting

- p1,p2 *1st*
- p2,p3 *2nd*
- p4,p5 *3rd*
- p1,p3 *4th*

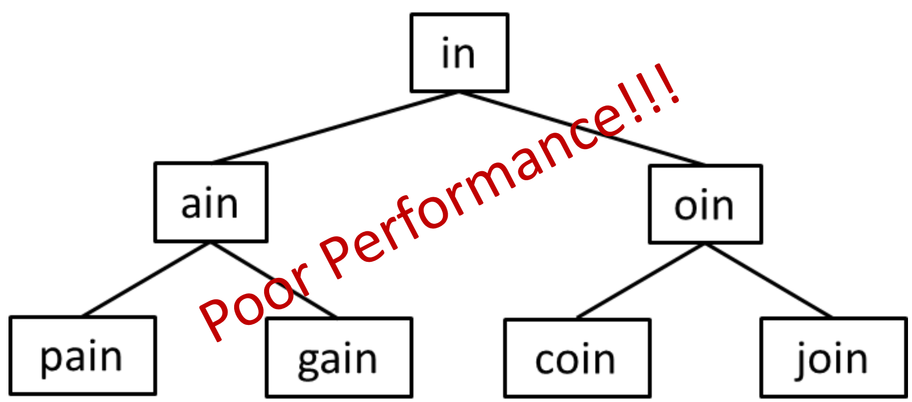
Taxonomy of Progressive Methods



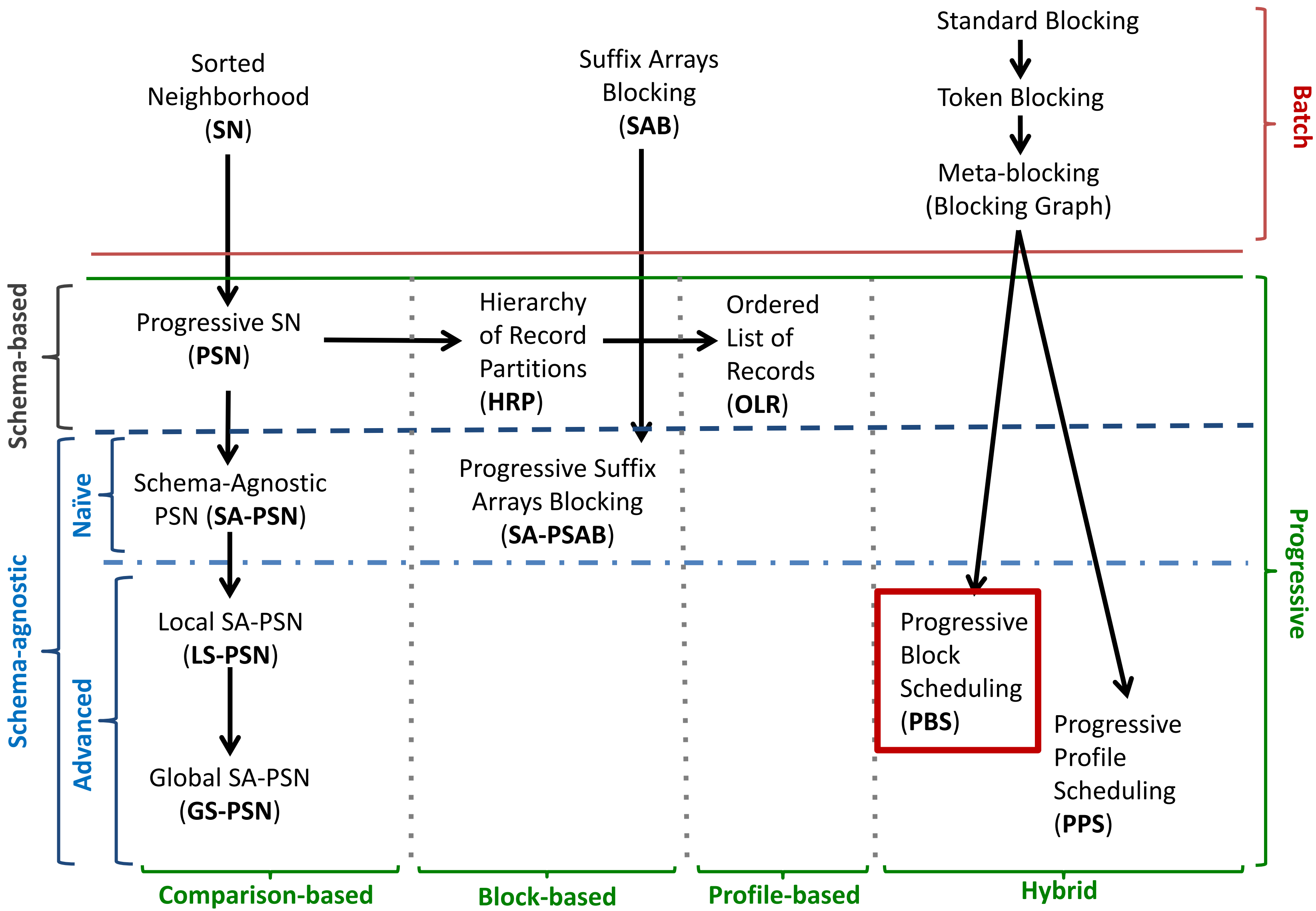
Progressive Suffix Arrays Blocking

[Simonini et. al., ICDE 2018]

- Every token in any attribute value is a blocking key
- Every key is converted to all suffixes with at least l_{min} characters
- Every suffix of minimum length creates a tree with that suffix at its root → Hierarchy of Partitions



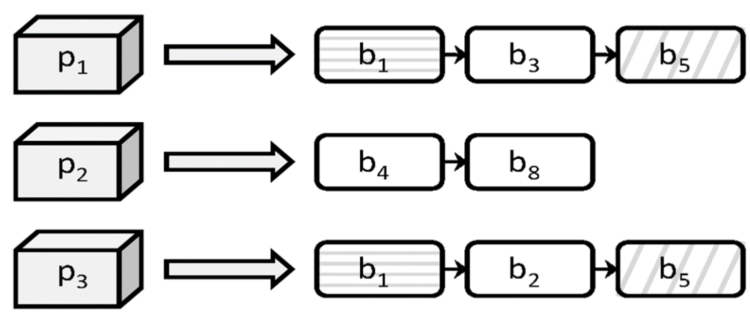
Taxonomy of Progressive Methods



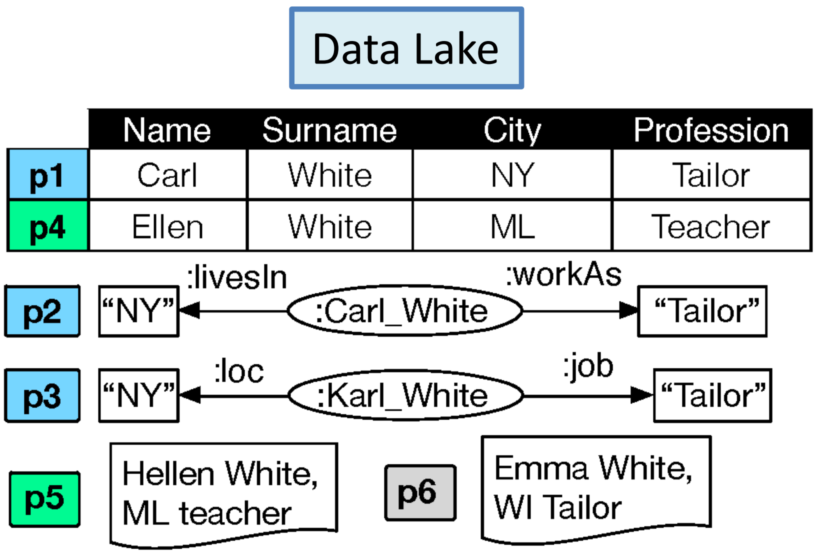
Progressive Block Scheduling (PBS)

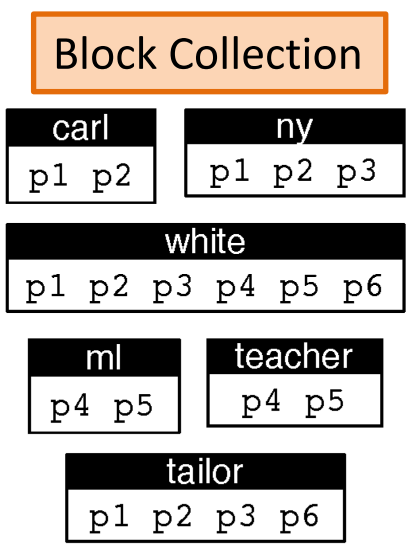
[Simonini et. al., ICDE 2018]

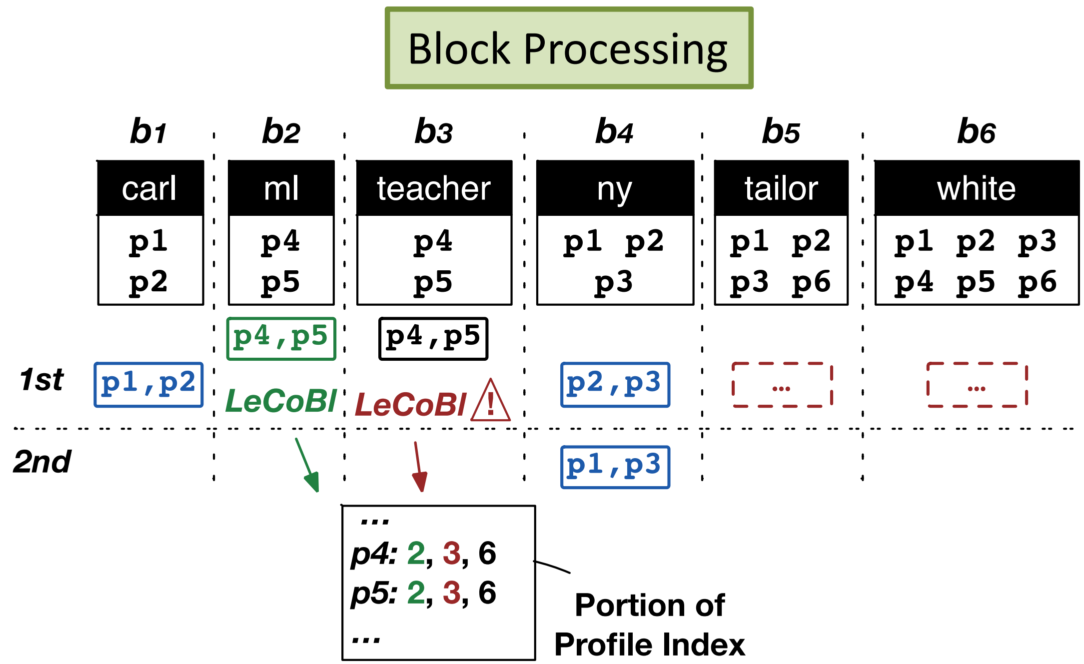
- Based on redundancy-positive blocking methods
- Orders blocks in increasing comparisons
- For each block:
  - Estimate the weight of all comparisons
  - Sort and process the non-redundant comparisons in decreasing weight
  - Relies on Entity (Profile) Index



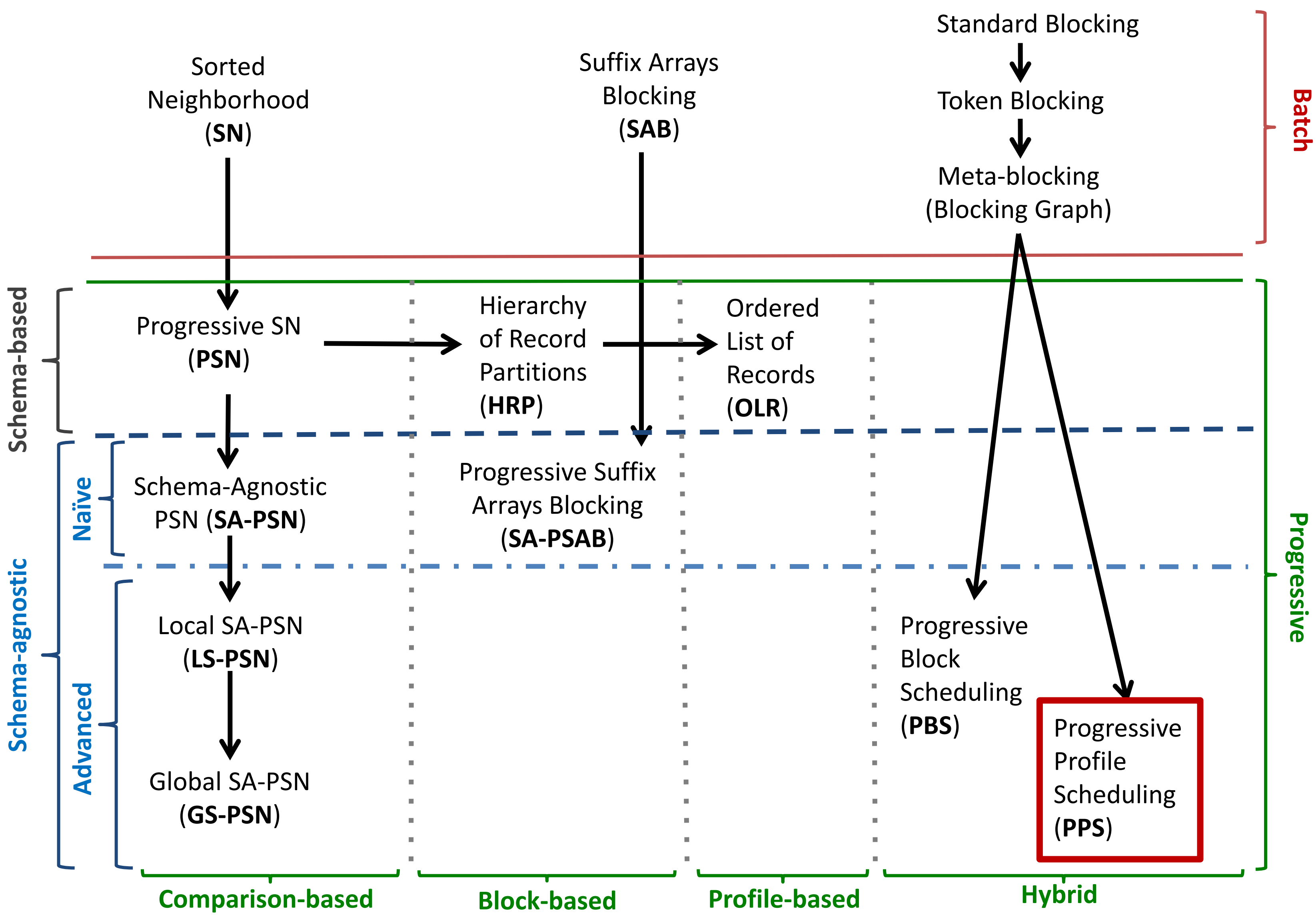
Example of PBS







Taxonomy of Progressive Methods



Progressive Profile Scheduling (PPS)

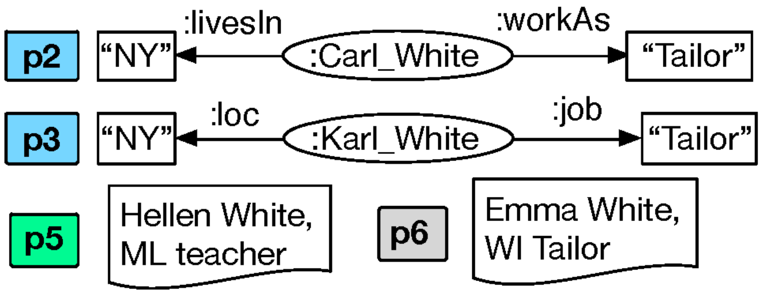
[Simonini et. al., ICDE 2018]

- Based on redundancy-positive blocking methods
- Orders entities in decreasing duplication likelihood
- Simultaneously, it aggregates the top-weighted comparison per entity → these are the first comparisons to be processed
- Processes one entity at a time
  - For each entity, it considers the k top-weighted co-occurring ones in decreasing edge weight

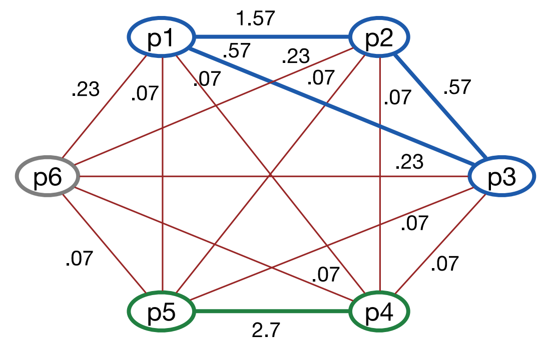
Example of PPS

Data Lake

	Name	Surname	City	Profession
p1	Carl	White	NY	Tailor
p4	Ellen	White	ML	Teacher



Blocking Graph



Sorted Profile List

p4	p5	p1	p2	p3	p6
1st	*2nd*	*3rd*	*4th*	*5th*	*6th*
W= .6	*W*= .6	*W*= .5	*W*= .5	*W*= .2	*W*= .1

Block Collection

carl
p1 p2

ny
p1 p2 p3

white
p1 p2 p3 p4 p5 p6

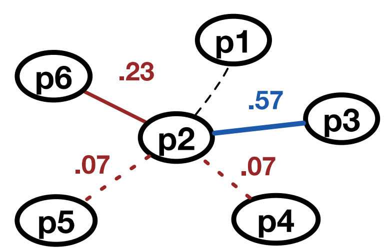
ml
p4 p5

teacher
p4 p5

tailor
p1 p2 p3 p6

Local Comparison List for p_2

1st	C23
2st	C26



Experimental Results [Simonini et. al., ICDE 2018]

- datasets used:

	ER type	$\|P\|$	#attr.	$\|\mathcal{D}_P\|$
Structured Datasets				
census	Dirty ER	841	5	344
restaurant	Dirty ER	864	5	112
cora	Dirty ER	1.3k	12	17k
cddb	Dirty ER	9.8k	106	300
Large, Heterogeneous Datasets				
movies	Clean-clean ER	28k—23k	4—7	23k
dbpedia	Clean-clean ER	1.2M—2.2M	30k—50k	893k
freebase	Clean-clean ER	4.2M—3.7M	37k—11k	1.5M

Experimental Results [Simonini et. al., ICDE 2018]

- datasets used:

	ER type	$\lvert P \rvert$	#attr.	$\lvert \mathcal{D}_P \rvert$
	Structured Datasets			
census	Dirty ER	841	5	344
restaurant	Dirty ER	864	5	112
cora	Dirty ER	1.3k	12	17k
cddb	Dirty ER	9.8k	106	300
	Large, Heterogeneous Datasets			
movies	Clean-clean ER	28k—23k	4—7	23k
dbpedia	Clean-clean ER	1.2M—2.2M	30k—50k	893k
freebase	Clean-clean ER	4.2M—3.7M	37k—11k	1.5M

Experimental Results [Simonini et. al., ICDE 2018]

- datasets used:

	ER type	$\|P\|$	#attr.	$\|\mathcal{D}_P\|$
Structured Datasets				
census	Dirty ER	841	5	344
restaurant	Dirty ER	864	5	112
cora	Dirty ER	1.3k	12	17k
cddb	Dirty ER	9.8k	106	300
Large, Heterogeneous Datasets				
movies	Clean-clean ER	28k—23k	4—7	23k
dbpedia	Clean-clean ER	1.2M—2.2M	30k—50k	893k
freebase	Clean-clean ER	4.2M—3.7M	37k—11k	1.5M

- measures used: $\boldsymbol{Recall} = \dfrac{\boldsymbol{\#\ emitted\ matches}}{\boldsymbol{\#\ existing\ matches}}$ $\quad \boldsymbol{ec}^* = \dfrac{\boldsymbol{\#\ emitted\ comparisons}}{\boldsymbol{\#\ existing\ matches}}$

  - $\boldsymbol{ec}^*$ measures # of comparisons as multiples of all comparisons of optimal

Experimental Results [Simonini et. al., ICDE 2018]

- datasets used:

	ER type	$\|P\|$	#attr.	$\|\mathcal{D}_P\|$
Structured Datasets				
census	Dirty ER	841	5	344
restaurant	Dirty ER	864	5	112
cora	Dirty ER	1.3k	12	17k
cddb	Dirty ER	9.8k	106	300
Large, Heterogeneous Datasets				
movies	Clean-clean ER	28k—23k	4—7	23k
dbpedia	Clean-clean ER	1.2M—2.2M	30k—50k	893k
freebase	Clean-clean ER	4.2M—3.7M	37k—11k	1.5M

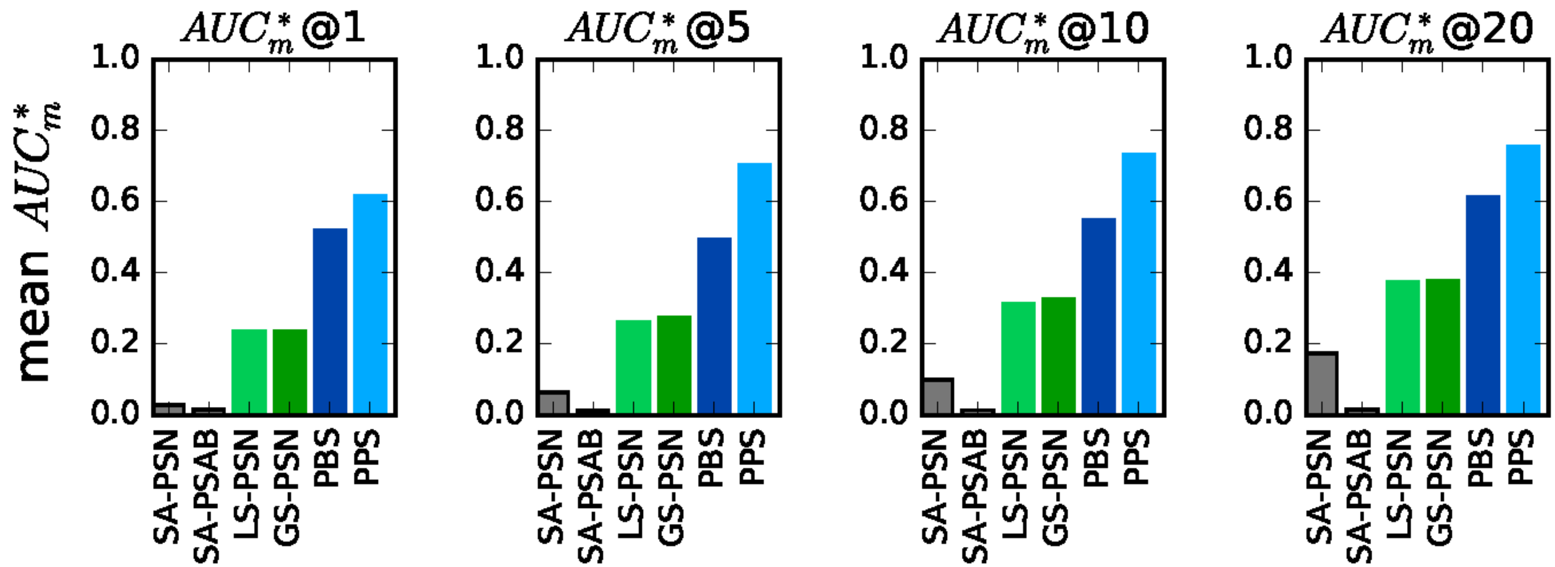
- measures used: $\boldsymbol{Recall} = \frac{\#\ \boldsymbol{emitted\ matches}}{\#\ \boldsymbol{existing\ matches}}$ $\quad \boldsymbol{ec}^* = \frac{\#\ \boldsymbol{emitted\ comparisons}}{\#\ \boldsymbol{existing\ matches}}$

$$\boldsymbol{area\ under\ curve}: \quad \boldsymbol{AUC}_{\boldsymbol{m}}^*@\boldsymbol{ec}^* = \int_0^{\boldsymbol{ec}^*} \frac{\boldsymbol{Recall\ Curve\ of\ m}}{\boldsymbol{ideal\ Recall\ Curve}}$$

  - $\boldsymbol{ec}^*$ measures # of comparisons as multiples of all comparisons of optimal
  - $\boldsymbol{AUC}_{\boldsymbol{m}}^*@\boldsymbol{ec}^*$ measures performance of method *m* for effort $\boldsymbol{ec}^*$
    - the higher the $\boldsymbol{AUC}_{\boldsymbol{m}}^*@\boldsymbol{ec}^*$, the better (optimal has $\boldsymbol{AUC}_{\boldsymbol{m}}^*@\boldsymbol{ec}^*$=1)

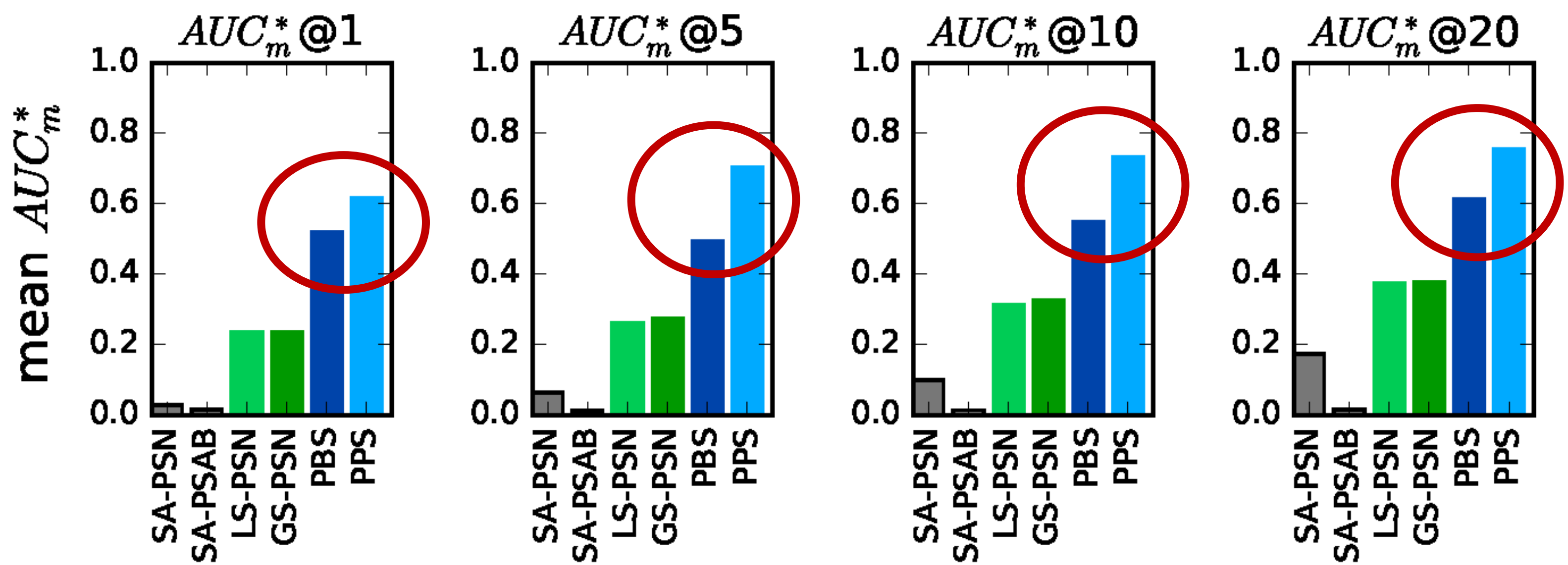
Experimental Results [Simonini et. al., ICDE 2018]

- performance over heterogeneous datasets



Experimental Results [Simonini et. al., ICDE 2018]

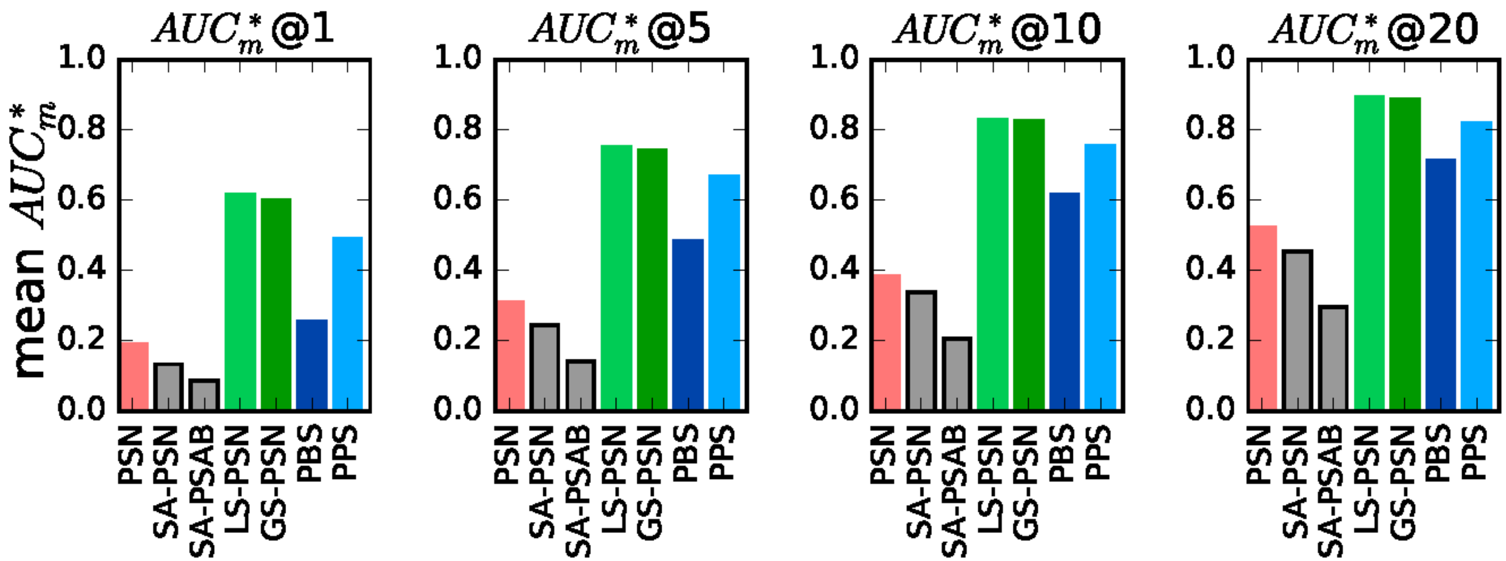
- performance over heterogeneous datasets



- methods based on redundancy-positive blocking perform significantly better
- PPS is the winner

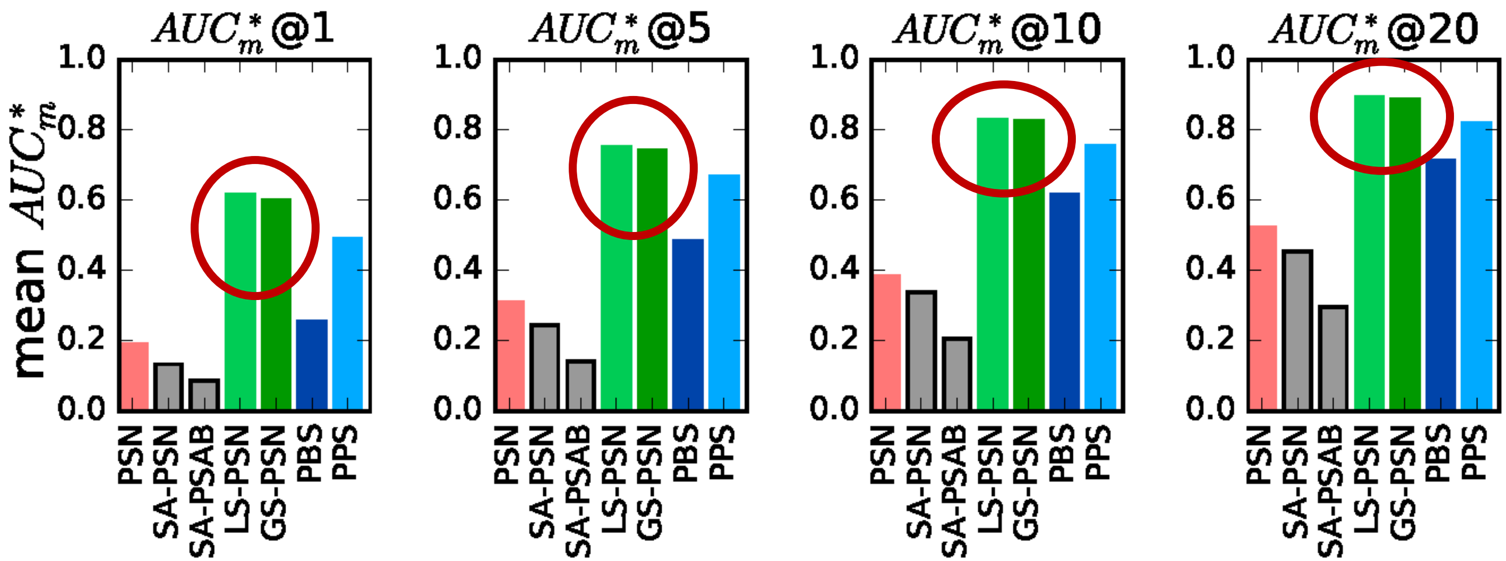
Experimental Results [Simonini et. al., ICDE 2018]

- performance over structured datasets



Experimental Results [Simonini et. al., ICDE 2018]

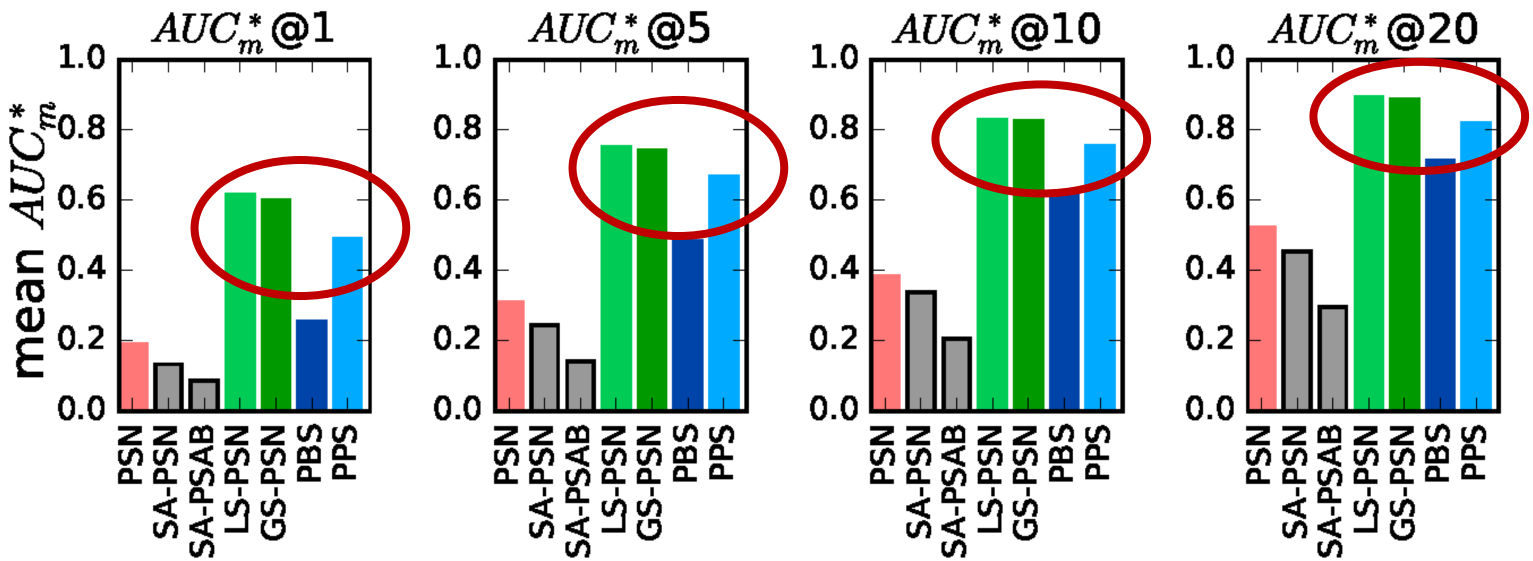
- performance over structured datasets



- the LS/GS-PSN methods perform significantly better

Experimental Results [Simonini et. al., ICDE 2018]

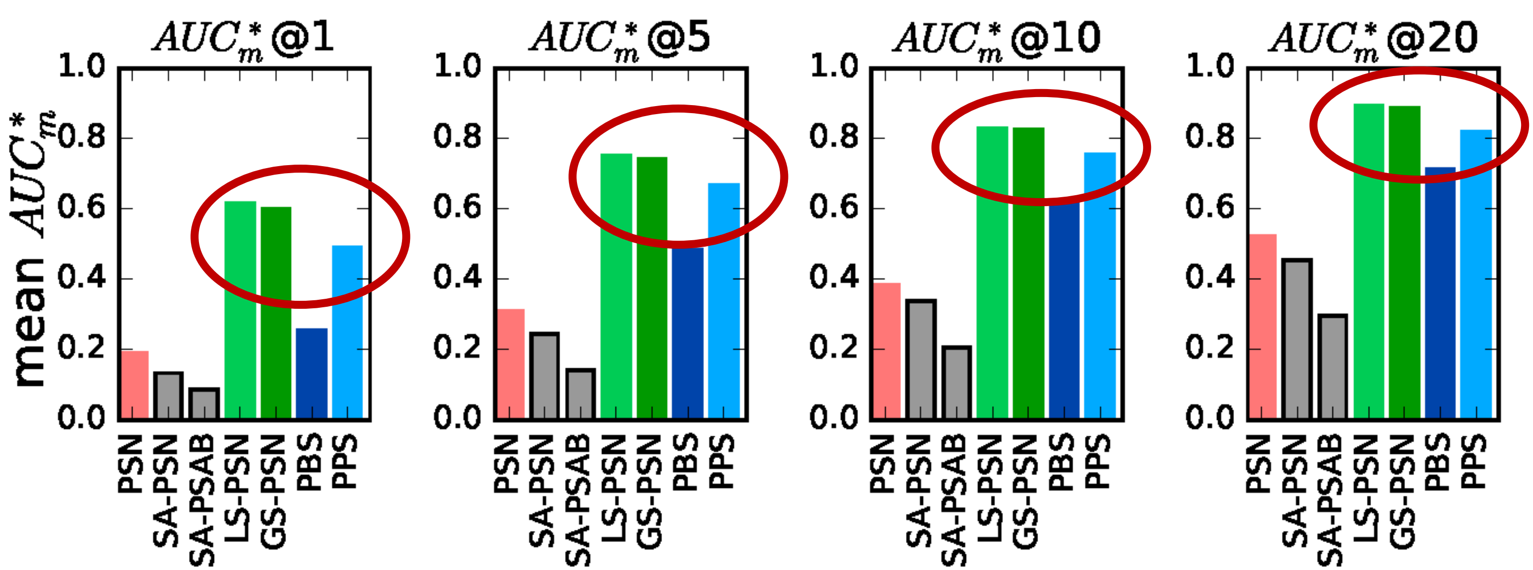
- performance over structured datasets



- the LS/GS-PSN methods perform significantly better
- PPS achieves almost the same performance

Experimental Results [Simonini et. al., ICDE 2018]

- performance over structured datasets



- the LS/GS-PSN methods perform significantly better
- PPS achieves almost the same performance

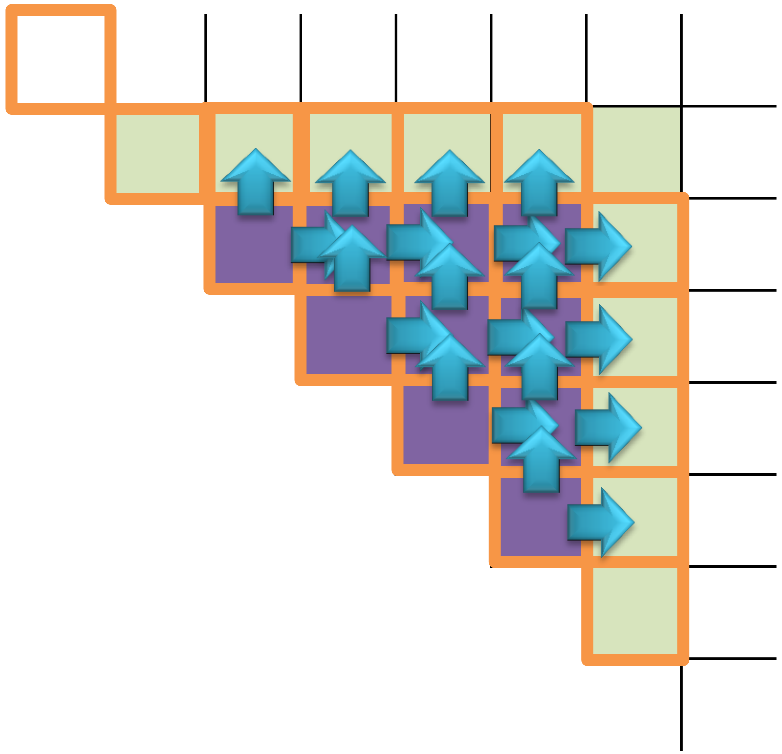
- overall, PPS is method of choice for progressive ER on both structured/ heterogeneous data

Dynamic Progressive Methods

- Problem of Static Methods:
  - The order of comparisons is immutable.
- Impact:
  - The algorithm cannot react to a skewed distribution of duplicates.
- Solution:
  - If *(i,j)* is a duplicate, then check *(i+1,j)* and *(i,j+1)* as well.
  - Assumption:
    - Oracle for Entity Matching

Dynamic Progressive SN

[Papenbrock et a., IEEE TKDE, 2015]



Courtesy of Thorsten Papenbrock

Part 10:
JedAI Toolkit

What is the JedAI Toolkit?

JedAI can be used in three ways:

1. As an open source library that implements numerous state-of-the-art methods for all steps of an established end-to-end ER workflow.

2. As a desktop application for ER with an intuitive Graphical User Interface that is suitable for both expert and lay users.

3. As a workbench for comparing all performance aspects of various (configurations of) end-to-end ER workflows.

JedAI vs other tools

Magellan

JedAI

× limited variety of (blocking) methods

✓ rich variety available methods for every step in the end-to-end workflow

JedAI vs other tools

Magellan

× limited variety of (blocking) methods

× restricted to relational data only

JedAI

✓ rich variety available methods for every step in the end-to-end workflow

✓ applies to both structured and non-structured data

JedAI vs other tools

Magellan

- × limited variety of (blocking) methods
- × restricted to relational data only
- × targeted to expert users, focusing on development of tailor-made methods

JedAI

- ✓ rich variety available methods for every step in the end-to-end workflow
- ✓ applies to both structured and non-structured data
- ✓ hands-off functionality through default configuration of every method, but also extensible

JedAI vs other tools

Magellan

JedAI

× limited variety of (blocking) methods

× restricted to relational data only

× targeted to expert users, focusing on development of tailor-made methods

× offers command-line interface, no GUI

✓ rich variety available methods for every step in the end-to-end workflow

✓ applies to both structured and non-structured data

✓ hands-off functionality through default configuration of every method, but also extensible

✓ intuitive GUI with guidelines even for novice users

JedAI vs other tools

Magellan

- × limited variety of (blocking) methods
- × restricted to relational data only
- × targeted to expert users, focusing on development of tailor-made methods
- × offers command-line interface, no GUI

JedAI

- ✓ rich variety available methods for every step in the end-to-end workflow
- ✓ applies to both structured and non-structured data
- ✓ hands-off functionality through default configuration of every method, but also extensible
- ✓ intuitive GUI with guidelines even for novice users
- ✓ multi-core execution (coming soon)

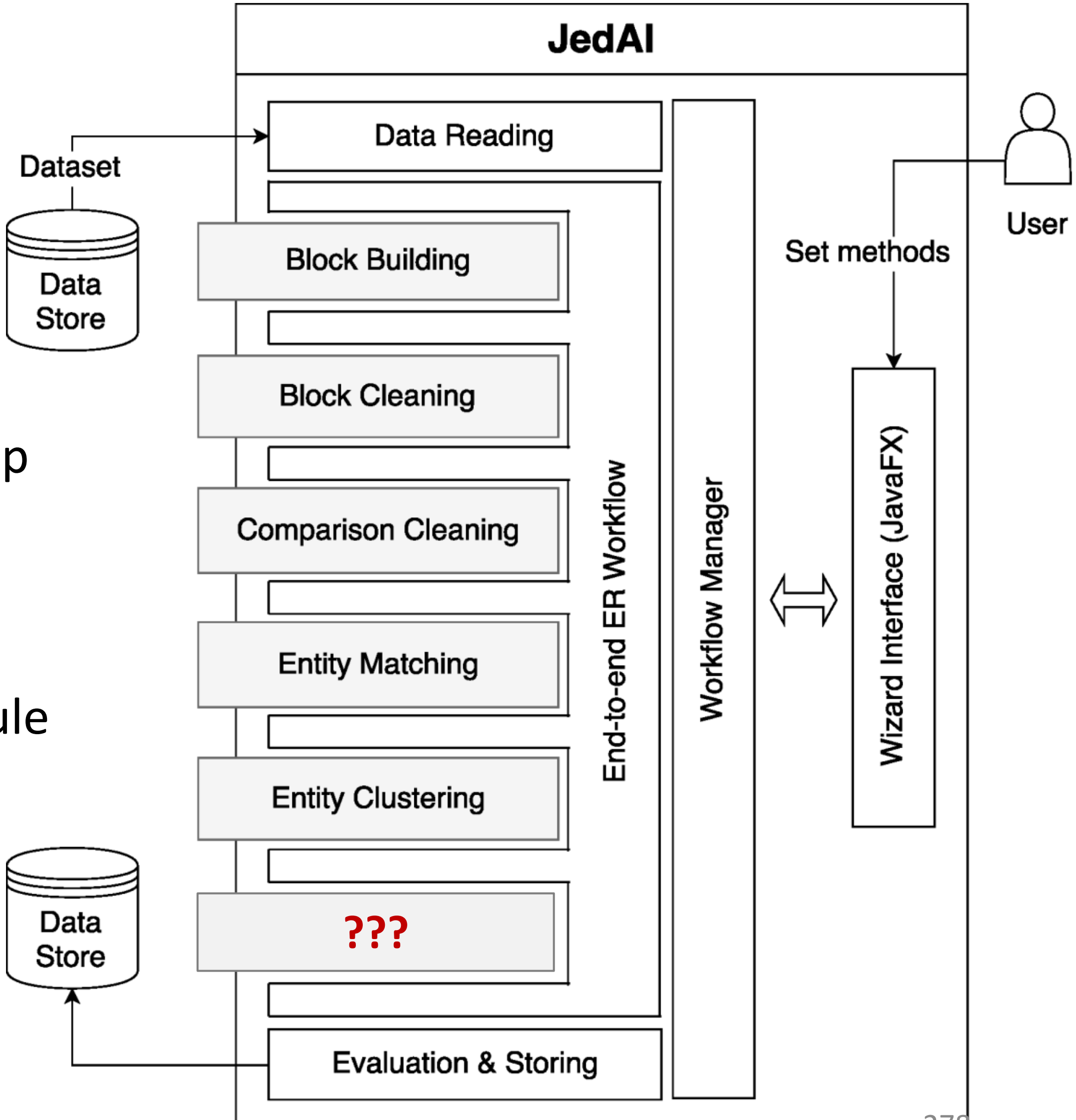
How does the JedAI Toolkit work?

JedAI implements the following schema-agnostic, end-to-end workflow for both Clean-Clean and Dirty ER:

Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7
Data Reading →	**Block Building** →	**Block Cleaning** →	**Comparison Cleaning** →	**Entity Matching** →	**Entity Clustering** →	**Evaluation & Storing**
Reads files containing the entity profiles and the golden standard.	Creates overlapping blocks.	Optional step that cleans blocks from useless comparisons (repeated, superfluous).	Optional step that operates on the level of individual comparisons to remove the useless ones.	Executes all retained comparisons.	Partitions the similarity graph into equivalence clusters.	Stores and presents performance results w.r.t. numerous measures.

How is the JedAI Toolkit structured?

- Modular architecture
  - one module per workflow step
- Extensible architecture
  - e.g., ontology matching module



How can I build an ER workflow?

JedAI supports several established methods for each workflow step:

Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7
Data Reading →	**Block Building** →	**Block Cleaning** →	**Comparison Cleaning** →	**Entity Matching** →	**Entity Clustering** →	**Evaluation & Storing**
Possible to read CSV, RDF/XML files, SPARQL endpoints & relational DBs in any combination	Choose 1 out of 7 methods.	Specify any combination of 3 complementary methods	Choose 1 out of 7 methods (including Meta-blocking)	Combine 1 out of 2 methods with 12 textual representation models and 10 similarity measures	Choose 1 out of 6 methods for Dirty ER. For Clean-Clean ER, 1 method is available	Store results as a CSV file

Which Data Formats are supported?

Data Formats
CSV files
Relational databases
XML/RDF/OWL files
SPARQL endpoints
Java Serialized Objects (using JedAI data model)

Which Blocking Methods are supported?

Block Building	Block Cleaning	Comparison Cleaning
Token Blocking	Block Filtering	Comparison Propagation
Sorted Neighborhood	Size-based Block Purging	Cardinality Edge Pruning (CEP)
Extended Sorted Neighborhood	Cardinality-based Block Purging	Cardinality Node Pruning (CNP)
Q-Grams Blocking	Block Scheduling	Weighted Edge Pruning (WEP)
Extended Q-Grams Blocking		Weighted Node Pruning (WNP)
Suffix Arrays		Reciprocal CNP
Extended Suffix Arrays		Reciprocal WNP

Which Entity Matching/Clustering Methods are supported?

Entity Matching	Entity Clustering
Group Linkage*	**Center Clustering**
Profile Matcher*	**Connected Components**
	Cut Clustering
* In combination with bag and graph textual models based on token and character n-grams and various established string similarity measures	**Markov Clustering**
	Merge-Center Clustering
	Ricochet SR Clustering
	Unique Mapping Clustering

Which Datasets are included?

Several datasets are available for testing

Clean-Clean ER (real)	D1 Entities	D2 Entities
Abt-Buy	1,076	1,076
DBLP-ACM	2,616	2,294
DBLP-Scholar	2,516	61,353
Amazon-GP	1,354	3,039
Movies	27,615	23,182
DBPedia	1,190,733	2,164,040

can be used for Dirty ER, too

Dirty ER (synthetic)	Entities
10K	10,000
50K	50,000
100K	100,000
200K	200,00
300K	300,00
1M	1,000,000
2M	2,000,000

What are the next steps?

- Version 2.0:
  - Includes support for schema clustering, multicore functionality, GNU Trove for higher time efficiency.
  - Available at the end of August, 2018.

- Version 3.0:
  - Includes support for data fusion, progressive ER as well as a workflow builder.
  - Available at the end of December, 2018.

- Version 4.0:
  - All functionality is implemented in Apache Spark.
  - Available at the end of December, 2019.

Where can I find JedAI Toolkit?

- Project website: http://jedai.scify.org
- Documentation (slides, videos, etc) available at github

- Github repositories:
  - JedAI Library: https://github.com/scify/JedAIToolkit
  - JedAI Desktop Application and Workbench: https://github.com/scify/jedai-ui .
  - All code is implemented using Java 8.
  - All code is publicly available under Apache License V2.0.

Where can I find JedAI Toolkit?

- Project website: http://jedai.scify.org
- Documentation (slides, videos, etc) available at github

- Github repositories:
  - JedAI Library: https://github.com/scify/JedAIToolkit
  - JedAI Desktop Application and Workbench: https://github.com/scify/jedai-ui .
  - All code is implemented using Java 8.
  - All code is publicly available under Apache License V2.0.

- JedAI already used in the industry, and in university courses

- When using JedAI, please cite:
  *George Papadakis, Leonidas Tsekouras, Emmanouil Thanos, George Giannakopoulos, Themis Palpanas and Manolis Koubarakis: "**JedAI: The Force behind Entity Resolution**", in ESWC 2017*

Part 11:

Challenges

Automatic Configuration

Facts:

- Several parameters in every blocking workflow
  - Both for lazy and proactive methods
- Blocking performance sensitive to internal configuration
  - Experimentally verified in [Papadakis et. al., VLDB 2016]
- Manual fine-tuning required

Open Research Directions:

- Plug-and-play blocking
- Data-driven configuration

Privacy Preserving Blocking

Facts:

- several applications ask for privacy-preserving ER
- lots of interest in this area

  [Christen, PADM 2006][Karakasidis et al., 2012][Ziad et al, BTW 2015]

Open Research Directions:

- What is the role of blocking workflow techniques?
  - block building, block filtering, comparison cleaning
- How can existing blocking techniques be adjusted?
- Novel blocking methods for this context

Incremental Blocking

Facts:

- Velocity in Web Data
- Dynamic ER
- Incremental ER [Gruenheid et. al., VLDB 2014]
  - Blocking → black box

Open Research Directions:

- Incremental (Meta-)Blocking

Distributed Blocking

Facts:

- Velocity in Big Data
- Need for even faster/more scalable ER solutions

Open Research Directions:

- What is the best way to use the modern distributed platforms/paradigms?
  - Flink/Spark
- How can we further improve performance of Parallel Meta-blocking?
  - Gelly/Gradoop/GraphX
- Minimize both time performance and total CPU cycles

Part 12:

Conclusions

Conclusions – Block Building

- Traditional proactive blocking methods only suitable for **relational data**
  - background schema knowledge should be available for their configuration

- Recent lazy blocking methods scale well to heterogeneous, semi-structured **Big Data**
  - **Variety** is addressed with schema-agnostic keys
  - **Volume** is addressed with Block and Comparison Cleaning methods → they trade slightly lower recall, for much higher precision
  - Token Blocking → the only parameter-free blocking method

Conclusions – Block Cleaning

- Coarse-grained functionality:
  - operation at the level of entire blocks
  - low cost (fast) methods

- Only applicable to lazy blocking methods

- They boost the overall performance to a large extent:
  - comparisons drop by orders of magnitude
  - recall drops to a controllable extent (~1-2%)

- Mostly complementary methods
  - multiple Block Cleaning methods can be combined in a single workflow

Conclusions – Comparison Cleaning

- Fine-grained functionality:
  - operate at the level of individual comparisons → computationally intensive process
- Apply to both lazy and proactive methods

- Meta-blocking is the current state-of-the-art
  - Discards both superfluous and redundant comparisons
  - Necessary for reducing comparisons to manageable levels
    - **reduces comparisons by orders of magnitude, with recall > 98%**
  - Naturally parallelizable

Big Data Research (BDR) Journal

http://www.journals.elsevier.com/big-data-research/

- New Elsevier journal on topics related to big data
  - advances in big data management/processing
  - interdisciplinary applications
- Editor in Chief for BDR
  - submit your work
  - propose special issues
- google: bdr journal

thank you!

questions?

http://sourceforge.net/projects/erframework

google: themis palpanas

-> publications -> tutorials

References – Part A

[Aizawa et. al., WIRI 2005] Akiko N. Aizawa, Keizo Oyama, "A Fast Linkage Detection Scheme for Multi-Source Information Integration" in WIRI, 2005.

[Baxter et. al., KDD 2003] R. Baxter, P. Christen, T. Churches, “A comparison of fast blocking methods for record linkage”, in Workshop on Data Cleaning, Record Linkage and Object Consolidation at KDD, 2003.

[Bilenko et. al., ICDM 2006] Mikhail Bilenko, Beena Kamath, Raymond J. Mooney, "Adaptive Blocking: Learning to Scale Up Record Linkage", in ICDM 2006.

[Christen, PADM 2006] Christen P: Privacy-preserving data linkage and geocoding: Current approaches and research directions. PADM held at IEEE ICDM, Hong Kong, 2006.

[Christen, TKDE 2011] P. Christen, "A survey of indexing techniques for scalable record linkage and deduplication.” in IEEE TKDE 2011.

[Efthymiou et. al., BigData 2015] Vasilis Efthymiou, George Papadakis, George Papastefanatos, Kostas Stefanidis, Themis Palpanas, "Parallel meta-blocking: Realizing scalable entity resolution over large, heterogeneous data", in IEEE Big Data 2015.

[Fellegi et. al., JASS 1969] P. Fellegi, A. Sunter, “A theory for record linkage,” in Journal of the American Statistical Society, vol. 64, no. 328, 1969.

[Fisher et. al., KDD 2015] Jeffrey Fisher, Peter Christen, Qing Wang, Erhard Rahm, "A Clustering-Based Framework to Control Block Sizes for Entity Resolution" in KDD 2015.

[Gravano et. al., VLDB 2001] L. Gravano, P. Ipeirotis, H. Jagadish, N. Koudas, S. Muthukrishnan, D. Srivastava, “Approximate string joins in a database (almost) for free’, in VLDB, 2001.

[Gruenheid et. al., VLDB 2014] Anja Gruenheid, Xin Luna Dong, Divesh Srivastava, "Incremental Record Linkage", in PVLDB 2014.

[Hernandez et. al., SIGMOD 1995] M. Hernandez, S. Stolfo, “The merge/purge problem for large databases”, in SIGMOD, 1995.

References – Part B

[Karakasidis et al., SAC 2012] Karakasidis A and Verykios VS: Reference table based k-anonymous private blocking. Symposium on Applied Computing, 2012.

[Kenig et. al., IS 2013] Batya Kenig, Avigdor Gal, "MFIBlocks: An effective blocking algorithm for entity resolution", in Inf. Syst. 2013.

[Ma et. Al., WSDM 2013] Y. Ma, T. Tran, "TYPiMatch: type-specific unsupervised learning of keys and key values for heterogeneous web data integration", in WSDM 2013.

[McCallum et. al., KDD 2000] A. McCallum, K. Nigam, L. Ungar, "Efficient clustering of high-dimensional data sets with application to reference matching", in KDD, 2000.

[Michelson et. al., AAAI 2006] Matthew Michelson, Craig A. Knoblock, "Learning Blocking Schemes for Record Linkage", in AAAI 2006.

[Papadakis et. al., EDBT 2016] George Papadakis, George Papastefanatos, Themis Palpanas, Manolis Koubarakis, "Scaling Entity Resolution to Large, Heterogeneous Data with Enhanced Meta-blocking", in EDBT 2016.

[Papadakis et al., iiWAS 2010] G. Papadakis, G. Demartini, P. Fankhauser, P. Karger, "The missing links: discovering hidden same-as links among a billion of triples", in iiWAS 2010.

[Papadakis et al., JCDL 2011] G. Papadakis, E. Ioannou, C. Niederee, T. Palpanas, W. Nejdl, "Eliminating the redundancy in blocking-based entity resolution methods", in JCDL 2011.

[Papadakis et al., SWIM 2011] G. Papadakis, E. Ioannou, C. Niederee, T. Palpanas, W. Nejdl, "To Compare or Not to Compare: making Entity Resolution more Efficient", in SWIM workshop (collocated with SIGMOD), 2011.

[Papadakis et. al., TKDE 2013] George Papadakis, Ekaterini Ioannou, Themis Palpanas, Claudia Niederee, Wolfgang Nejdl, "A Blocking Framework for Entity Resolution in Highly Heterogeneous Information Spaces", in IEEE TKDE 2013.

References – Part C

[Papadakis et. al., TKDE 2014] George Papadakis, Georgia Koutrika, Themis Palpanas, Wolfgang Nejdl, "Meta-Blocking: Taking Entity Resolution to the Next Level", in IEEE TKDE 2014.

[Papadakis et. al., VLDB 2014] G. Papadakis, G. Papastefanatos, G. Koutrika, "Supervised Meta-blocking", in PVLDB 2014.

[Papadakis et. al., VLDB 2015] George Papadakis, George Alexiou, George Papastefanatos, Georgia Koutrika, "Schema-agnostic vs Schema-based Configurations for Blocking Methods on Homogeneous Data", in PVLDB 2015.

[Papadakis et. al., VLDB 2016] George Papadakis, Jonathan Svirsky, Avigdor Gal, Themis Palpanas, "Comparative Analysis of Approximate Blocking Techniques for Entity Resolution", in PVLDB 2016.

[Papadakis et al., WSDM 2011] G. Papadakis, E. Ioannou, C. Niederee, P. Fankhauser, "Efficient entity resolution for large heterogeneous information spaces", in WSDM 2011.

[Papadakis et al., WSDM 2012] G. Papadakis, E. Ioannou, C. Niederee, T. Palpanas, W. Nejdl, "Beyond 100 million entities: large-scale blocking-based resolution for heterogeneous data", in WSDM 2012.

[Papenbrock et. al., TKDE 2015] Thorsten Papenbrock, Arvid Heise, Felix Naumann, "Progressive Duplicate Detection", in IEEE TKDE 2015.

[Sarma et. al, CIKM 2012] Anish Das Sarma, Ankur Jain, Ashwin Machanavajjhala, Philip Bohannon, "An automatic blocking mechanism for large-scale de-duplication tasks" in CIKM 2012.

[Simonini et. al, VLDB 2017] Giovanni Simonini, Sonia Bergamaschi and H.V. Jagadish, "Blast: a Loosely schema-aware Meta-blocking Approach for Entity Resolution" in VLDB 2017.

[Whang et. Al, SIGMOD 2009] Whang, D. Menestrina, G. Koutrika, M. Theobald, H. Garcia-Molina, "Entity resolution with iterative blocking", in SIGMOD 2009.

[Whang et. al., TKDE 2013] Steven Euijong Whang, David Marmaros, Hector Garcia-Molina, "Pay-As-You-Go Entity Resolution", in IEEE TKDE 2013.

[Ziad et al, BTW 2015] Ziad Sehili, Lars Kolb, Christian Borgs, Rainer Schnell, Erhard Rahm, "Privacy Preserving Record Linkage with PPJoin", in BTW 2015.

References – Part D

[APR-JGraph04] J. A. Aslam, E. Pelekhov, D. Rus, "The star clustering algorithm for static and dynamic information organization", in Journal of Graph Algorithms and Appl 2004.

[BBC-ML04] N. Bansal, A. Blum, S. Chawla, "Correlation clustering", in Machine Learning" 2004.

[BCKT-VLDB07] N. Bansal, F. Chiang, N. Koudas, F. W. Tompa, "Seeking stable clusters in the blogosphere", in VLDB 2007.

[Dongen-Thesis00] S. van Dongen, "Graph clustering by flow simulation", PhD thesis, University of Utrecht, 2000.

[Jain&Dubes88] A. Jain, R. Dubes, "Algorithms for Clustering Data", Prentice Hall, 1988.

[FTT-IM04] G. W. Flake, R. E. Tarjan, K. Tsioutsiouliklis, "Graph clustering and minimum cut trees", in Internet Mathematics 2004.

[HGI-WebDB'00] T. H. Haveliwala, A. Gionis, P. Indyk, "Scalable Techniques for Clustering the Web", in WebDB 2000.

[HM-VLDBJ09] O. Hassanzadeh, R. J. Miller, "Creating Probabilistic Databases from Duplicated Data", VLDB J. 2009.

[WB-TR07] D. T. Wijaya, S. Bressan, "Ricochet: A family of unconstrained algorithms for graph clustering", Technical report, National University of Singapore, 2007.

[WB-DASFAA'09] D. T. Wijaya, S. Bressan. Ricochet: A Family of Unconstrained Algorithms for Graph Clustering. In Proc. of the Int'l Conf. on Database Systems for Advanced Applications (DASFAA), pages 153–167, Brisbane, Australia, 2009.

[On et al., ICDE 2007] Byung-Won On, Nick Koudas, Dongwon Lee, Divesh Srivastava, "Group Linkage", in ICDE 2007.

[Lacoste-Julien et al., KDD 2013] Simon Lacoste-Julien, Konstantina Palla, Alex Davies, Gjergji Kasneci, Thore Graepel, Zoubin Ghahramani, "SIGMa: simple greedy matching for aligning large knowledge bases", in KDD 2013.

[Suchanek et al., PVLDB 2011] Fabian M. Suchanek, Serge Abiteboul, Pierre Senellart, "PARIS: Probabilistic Alignment of Relations, Instances, and Schema", in PVLDB 2011.

[Simonini et. al., ICDE 2018] Giovanni Simonini, George Papadakis, Themis Palpanas, Sonia Bergamaschi, "Schema-agnostic Progressive Entity Resolution", in ICDE 2018.

References – Part E

[Dong et al., Book 2015] Xin Luna Dong, Divesh Srivastava, “Big Data Integration”, in Synthesis Lectures on Data Management, Morgan & Claypool Publishers, 2015.

[Elmagarmid et al., TKDE 2007] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, Vassilios S. Verykios, “Duplicate Record Detection: A Survey”, in IEEE TKDE 2007.

[Papadakis et al., Semantics 2017] George Papadakis, Konstantina Bereta, Themis Palpanas, Manolis Koubarakis, “Multi-core Meta-blocking for Big Linked Data”, in SEMANTICS 2017.

[Böhm et al., CIKM 2012] Christoph Böhm, Gerard de Melo, Felix Naumann, Gerhard Weikum, “LINDA: distributed web-of-data-scale entity matching”, in CIKM 2012.

[Efthymiou et al., IS 2017] Vasilis Efthymiou, George Papadakis, George Papastefanatos, Kostas Stefanidis, Themis Palpanas, “Parallel meta-blocking for scaling entity resolution over big heterogeneous data” in Information Systems 2017.

[Hassanzadeh et al., VLDB 2009] Oktie Hassanzadeh, Fei Chiang, Renée J. Miller, Hyun Chul Lee, “Framework for Evaluating Clustering Algorithms in Duplicate Detection”, in PVLDB 2009.

[Stefanidis et al, WWW 2014] Kostas Stefanidis, Vasilis Efthymiou, Melanie Herschel, Vassilis Christophides, “Entity resolution in the web of data”, in WWW (Companion Volume) 2014.

[Benjelloun et al., VLDBJ 2009] Omar Benjelloun, Hector Garcia-Molina, David Menestrina, Qi Su, Steven Euijong Whang, Jennifer Widom, “Swoosh: a generic approach to entity resolution”, in VLDB Journal 2009.

[Kolb et al., PVLDB 2012] Lars Kolb, Andreas Thor, Erhard Rahm, “Dedoop: Efficient Deduplication with Hadoop”, in PVLDB, 2012.

[Singh et al., PVLDB 2017] Rohit Singh, Venkata Vamsikrishna Meduri, Ahmed K. Elmagarmid, Samuel Madden, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Armando Solar-Lezama, Nan Tang, “Synthesizing Entity Matching Rules by Examples”, in PVLDB 2017.

References – Part F

[Dal Bianco et al., Information Systems, 2018] Guilherme Dal Bianco, Marcos André Gonçalves, Denio Duarte, "BLOSS: Effective meta-blocking with almost no effort", in Information Systems 2018.