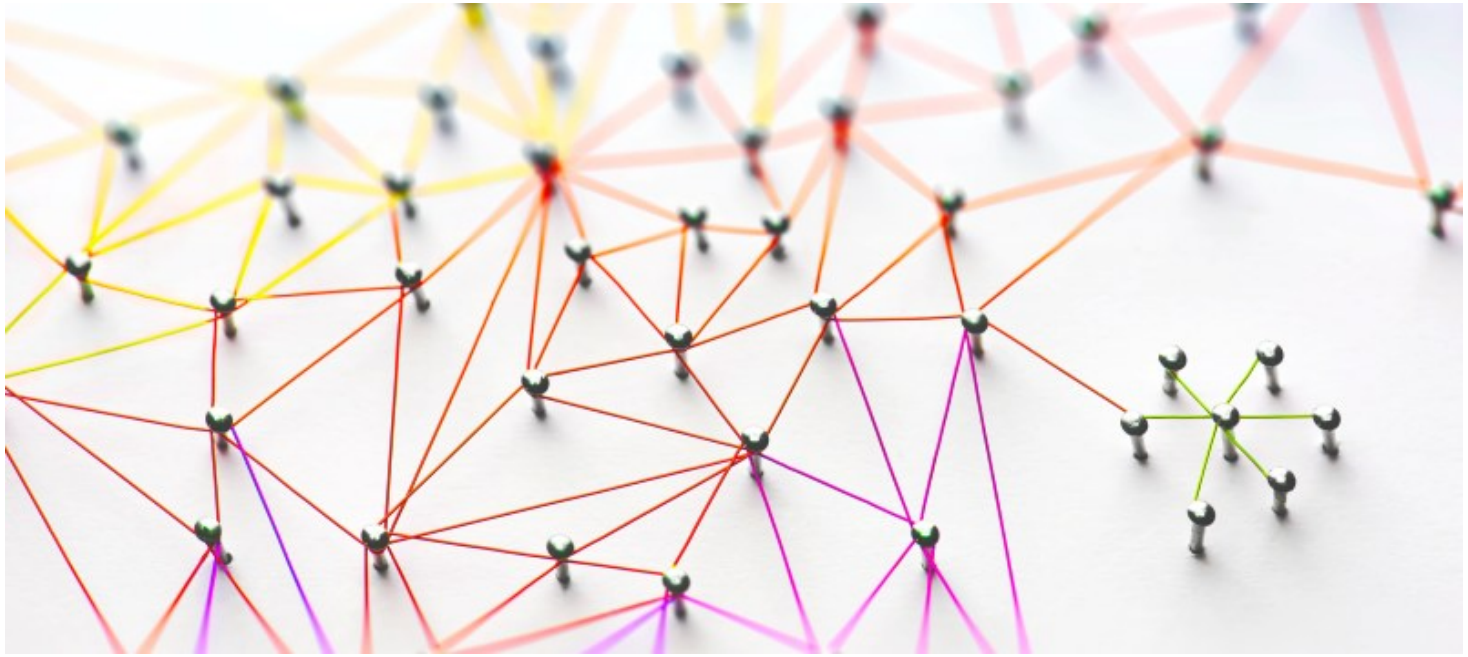
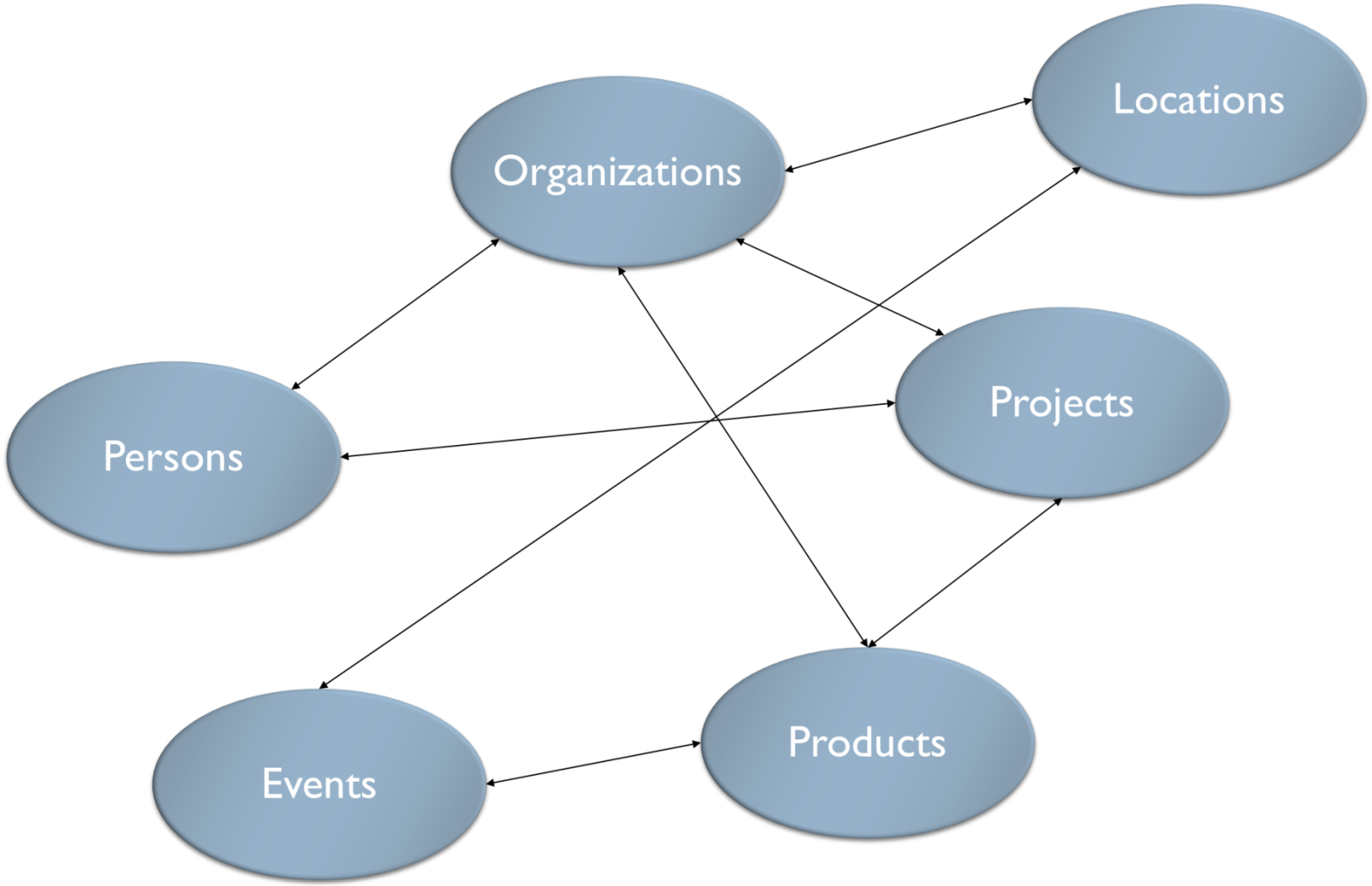
Entity Resolution



Giorgos Alexiou
ATHENA Research Center
&
Athens University of Economics and Business
galexiou@athenarc.gr

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"?

London 런던 ലണ്ടൻ लंडन लंदन ලන්ඩන් ለንደን ロンドン लन्डन 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
- …

- London, Jack
  2612 Almes Dr
  Montgomery, AL
  (334) 272-7005

- London, Jack R
  2511 Winchester Rd
  Montgomery, AL 36106-3327
  (334) 272-7005

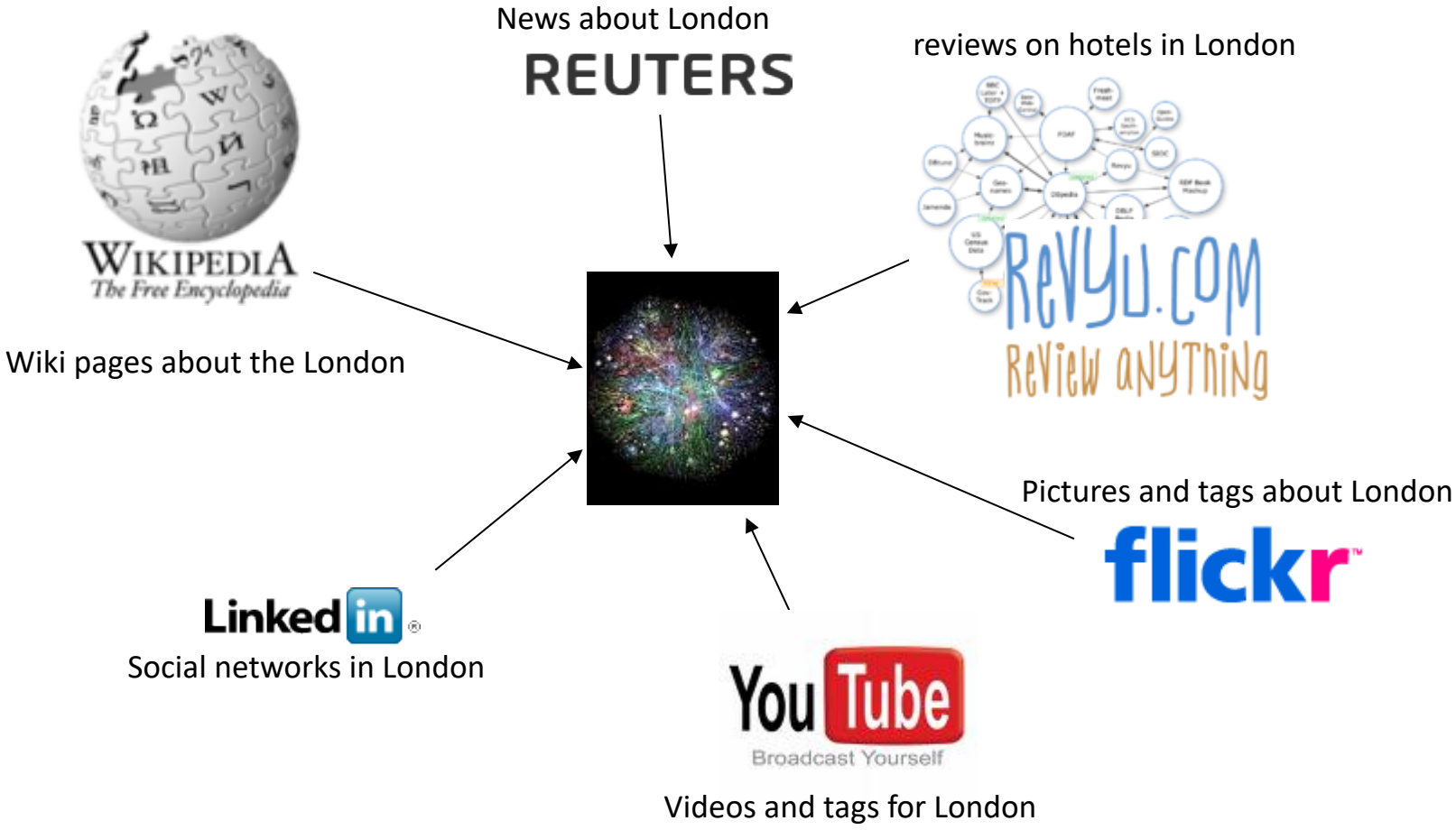
- London, Jack
  1222 Whitetail Trl
  Van Buren, AR 72956-7368
  (479) 474-4136

- 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"?



Entity Resolution (ER)

Entity Resolution [Christen, TKDE2011]:

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

Application areas:

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

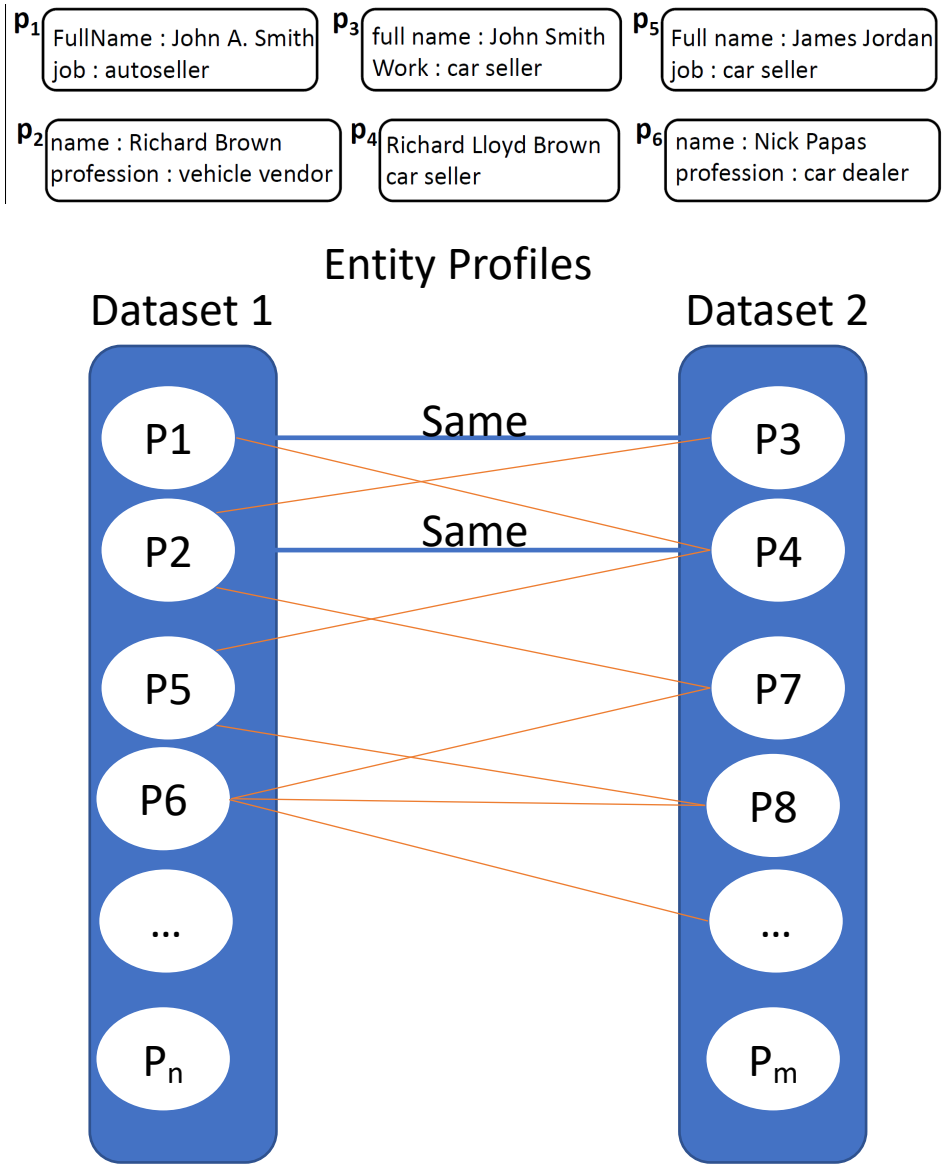
Useful because:

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

Ironically, ER has duplicate names…

- Record linkage
- Duplicate detection
- Deduplication
- Reference reconciliation
- Reference matching
- Object consolidation
- Fuzzy matching
- Entity clustering
- Hardening soft databases
- …

Entity Resolution (ER)



* Entity profile → a uniquely identified set of name-value pairs that corresponds to a single real-world object.

Time Complexity →quadratic

$O(n^2)$

Every entity is compared with all others.

Types of Entity Resolution

The input of ER consists of entity collections that can be of two types [Christen, TKDE2011]:

- **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, CiteseerX

Types of Entity Resolution

The input of ER consists of entity collections that can be of two types [Christen, TKDE2011]:

- **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, CiteseerX

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*)

Computational cost

ER is an inherently quadratic problem (i.e., $O(n^2)$):

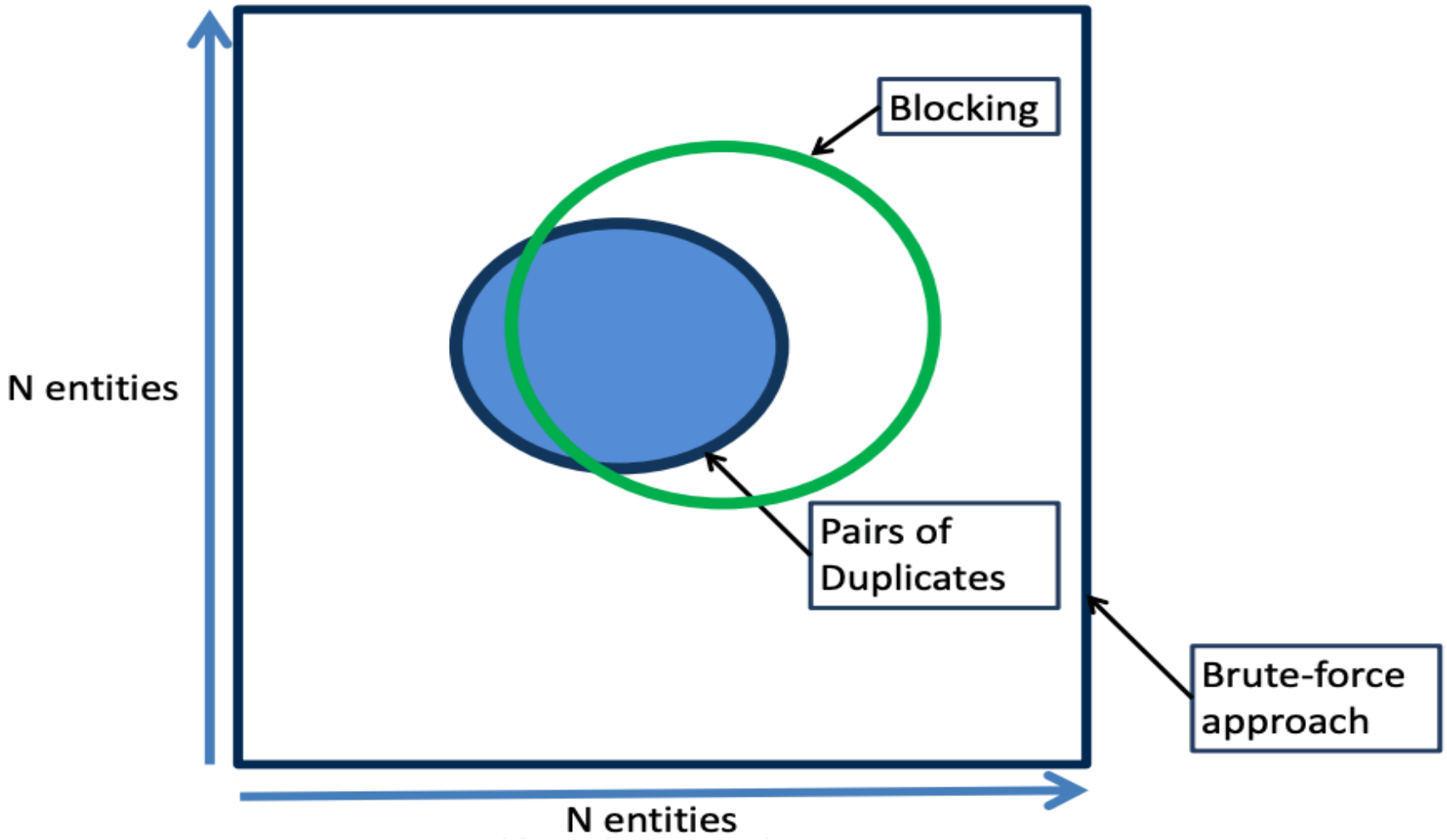
every entity has to be compared with all others

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

Solution: **Blocking**

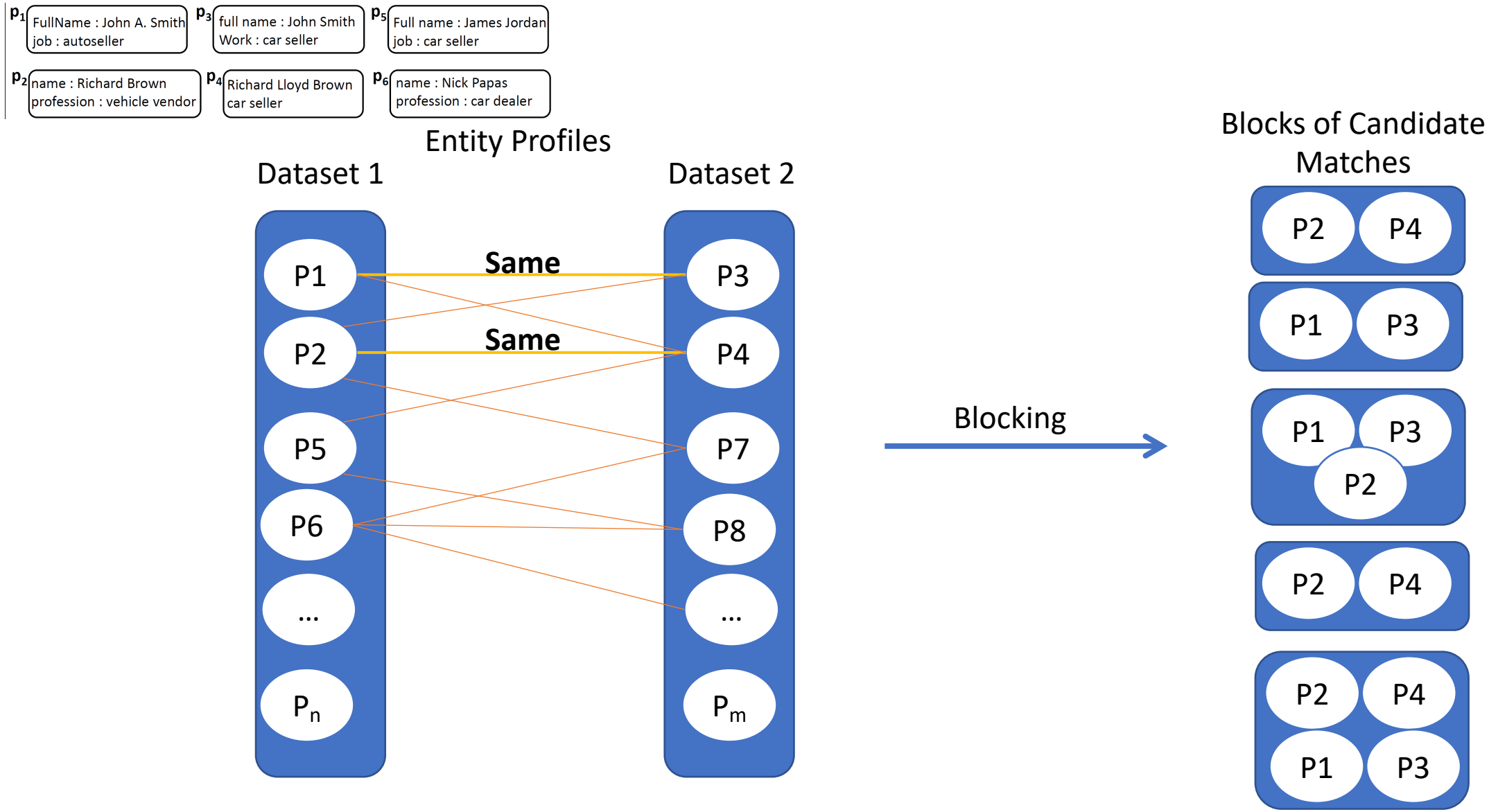
- group similar entities into blocks
- execute comparisons only inside blocks
- approximate solution

Computational cost



Blocking

Most brute-force comparisons involve non-matching entities



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 cooccur in at least one block.

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:

- Pairs Completeness: $PC = \frac{detected\ matches}{existing\ matches}$ (recall)
- Pairs Quality: $PQ = \frac{detected\ matches}{executed\ comparisons}$ (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 *PC* and *PQ* are maximized.

caution:

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

disclaimer:

Precision of entity matching is dependent on the entity similarity measures, and is orthogonal to the above problem.

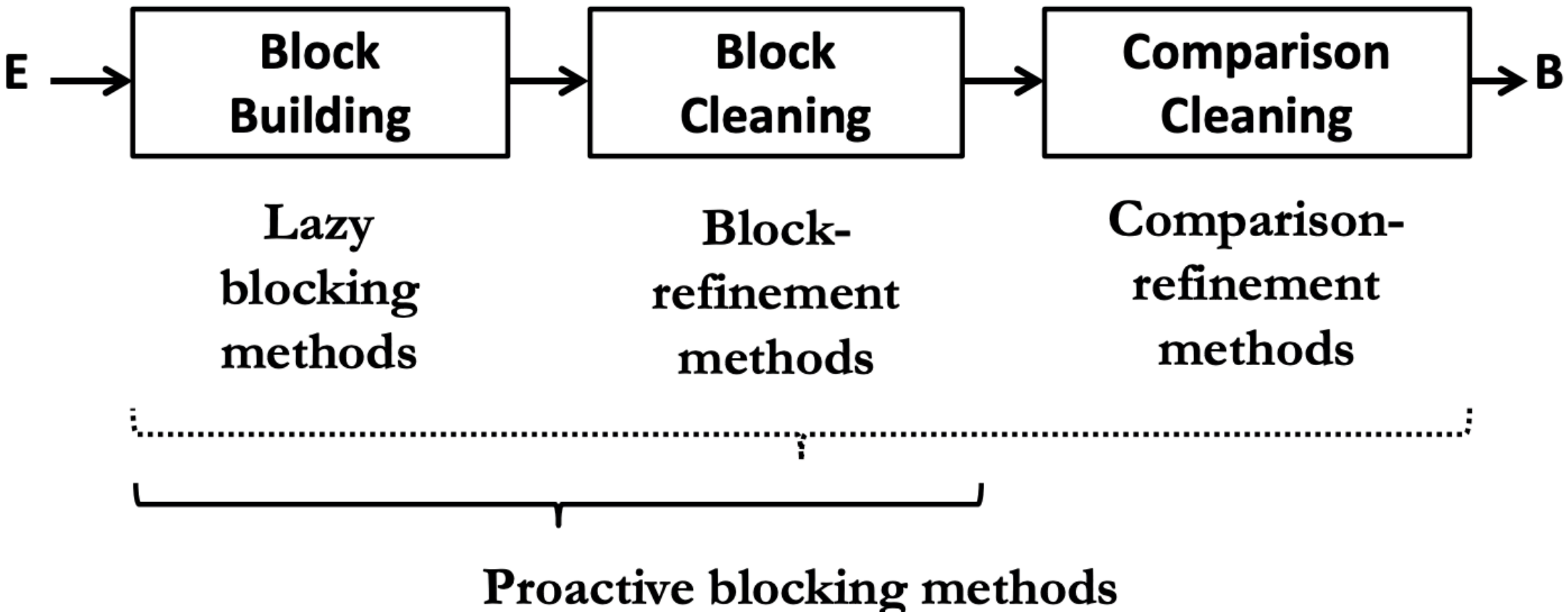
Categorization of Blocking Methods

1. Definition of blocking keys
   - Supervised
   - Unsupervised
2. Dependency on schema
   - Schema-based
   - Schema-agnostic
3. Redundancy
   - Disjoint blocks
   - Overlapping blocks
     - Redundancy-positive
     - Redundancy-neutral
     - Redundancy-negative

Definition of blocking keys

- **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
- **Unsupervised** Methods
  - Generic, popular methods

Blocking Workflow



Dependency on schema

- Schema-based
  - A-priori known schema → no noise in attribute names
  - Relies on domain knowledge
  - Blocking keys are (parts of) values from attribute names with low noise and high discriminativeness

- Schema-agnostic
  - Disregards schema knowledge
  - Every token in any attribute value is a blocking key.

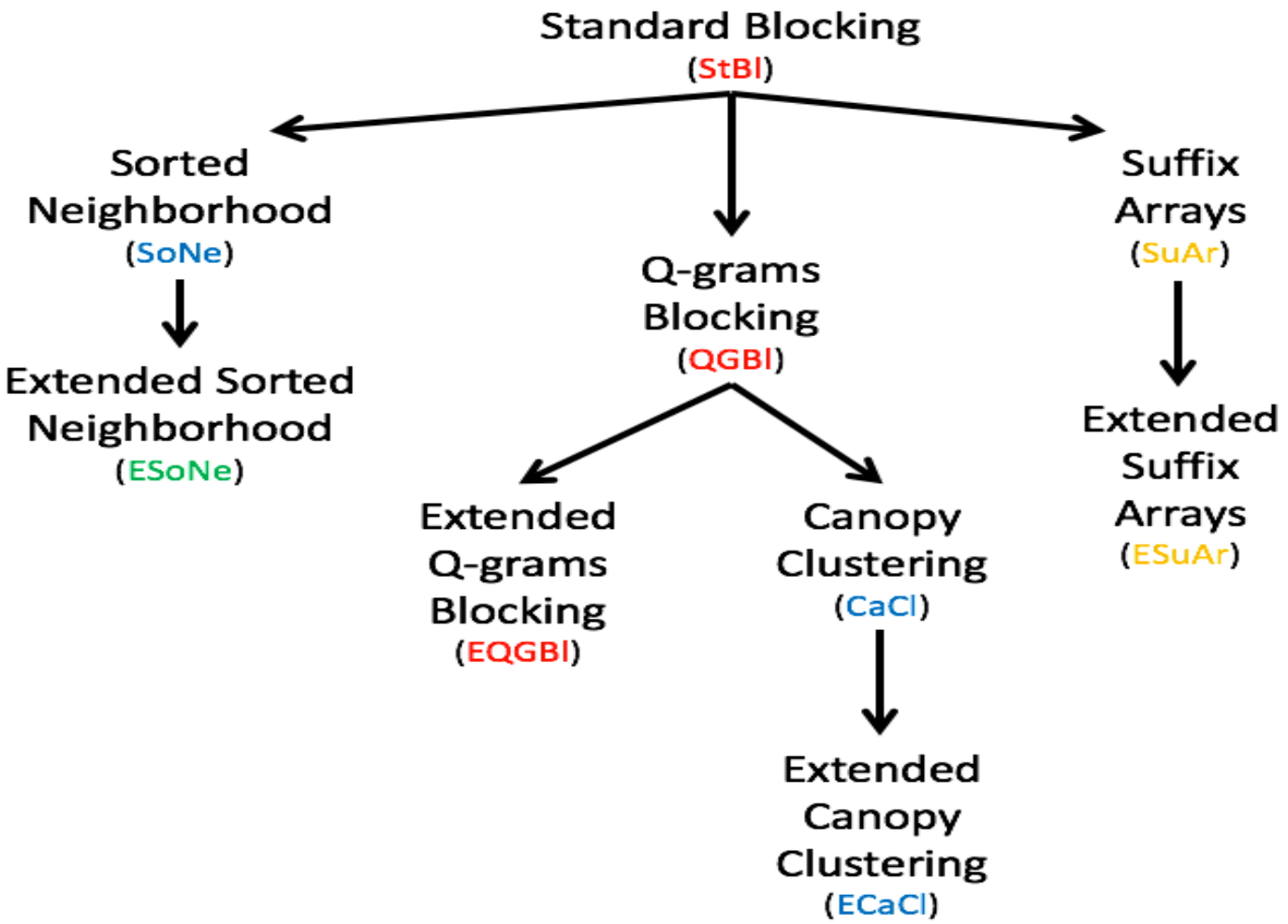
Pros and Cons

- Scope
  - The schema-based configuration is suitable **only** for structured (homogeneous) data
  - The schema-agnostic configuration is suitable for semi-structured (heterogeneous) data, **as well**
- Sensitivity
  - For the schema-based configuration, minor modifications in blocking keys lead to major differences in quality
  - The schema-agnostic configuration involves a straightforward, predetermined definition of keys
- Fine-tuning
  - Either manually or in a supervised way for schema-based configuration.
  - Completely unsupervised for schema-agnostic configuration

Blocks- and Signature-wise Categorization of Block Building Methods

	Disjoint Blocks	Overlapping Blocks		
		Redundancy-negative	**Redundancy-neutral**	**Redundancy-positive**
Schema-based	Standard Blocking	Canopy Clustering	Sorted Neighborhood	1.Q-grams Blocking 2.Suffix Array
Schema-agnostic	-	-	Semantic Indexing	1. Token Blocking 2. Agnostic Clustering 3. URI Semantics 4. TYPiMatch

Blocking Methods



**See also: Comparative Analysis of Approximate Blocking Techniques for Entity Resolution. George Papadakis, Jonathan Svirsky, Avigdor Gal, Themis Palpanas, VLDB2016.*

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

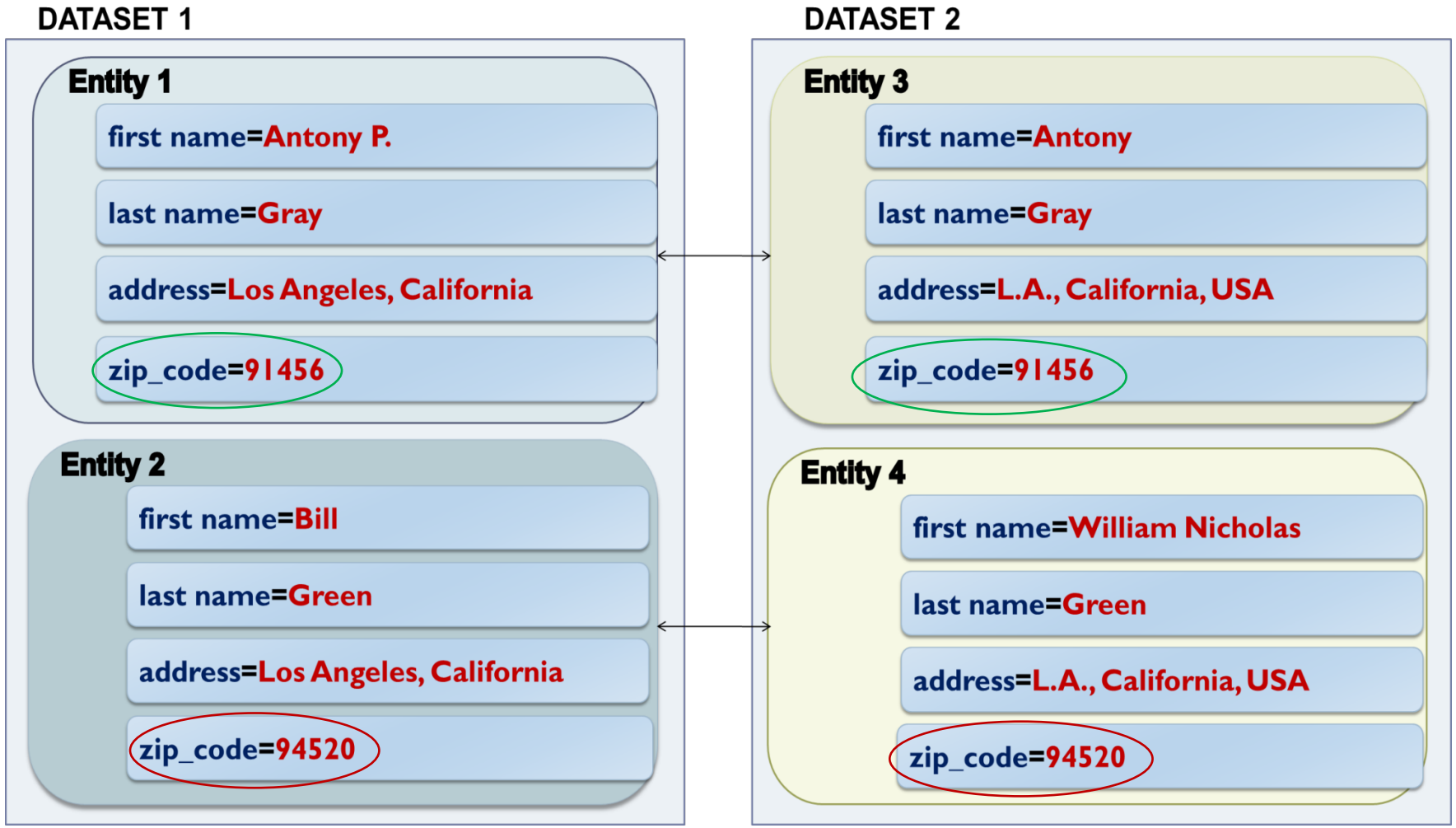
Earliest, simplest form of blocking.

Algorithm:

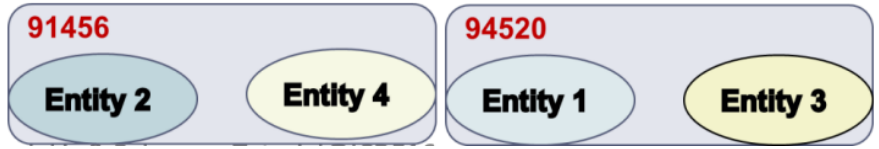
1. Select the most appropriate attribute name(s) w.r.t. noise and distinctiveness.
2. Transform the corresponding value(s) into a Blocking Key (BK)
3. For each BK, create one block that contains all entities having this BK in their transformation.

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

Example of Standard Blocking



Blocks on zip_code



Token Blocking [Papadakis et al., WSDM2011]

Functionality:

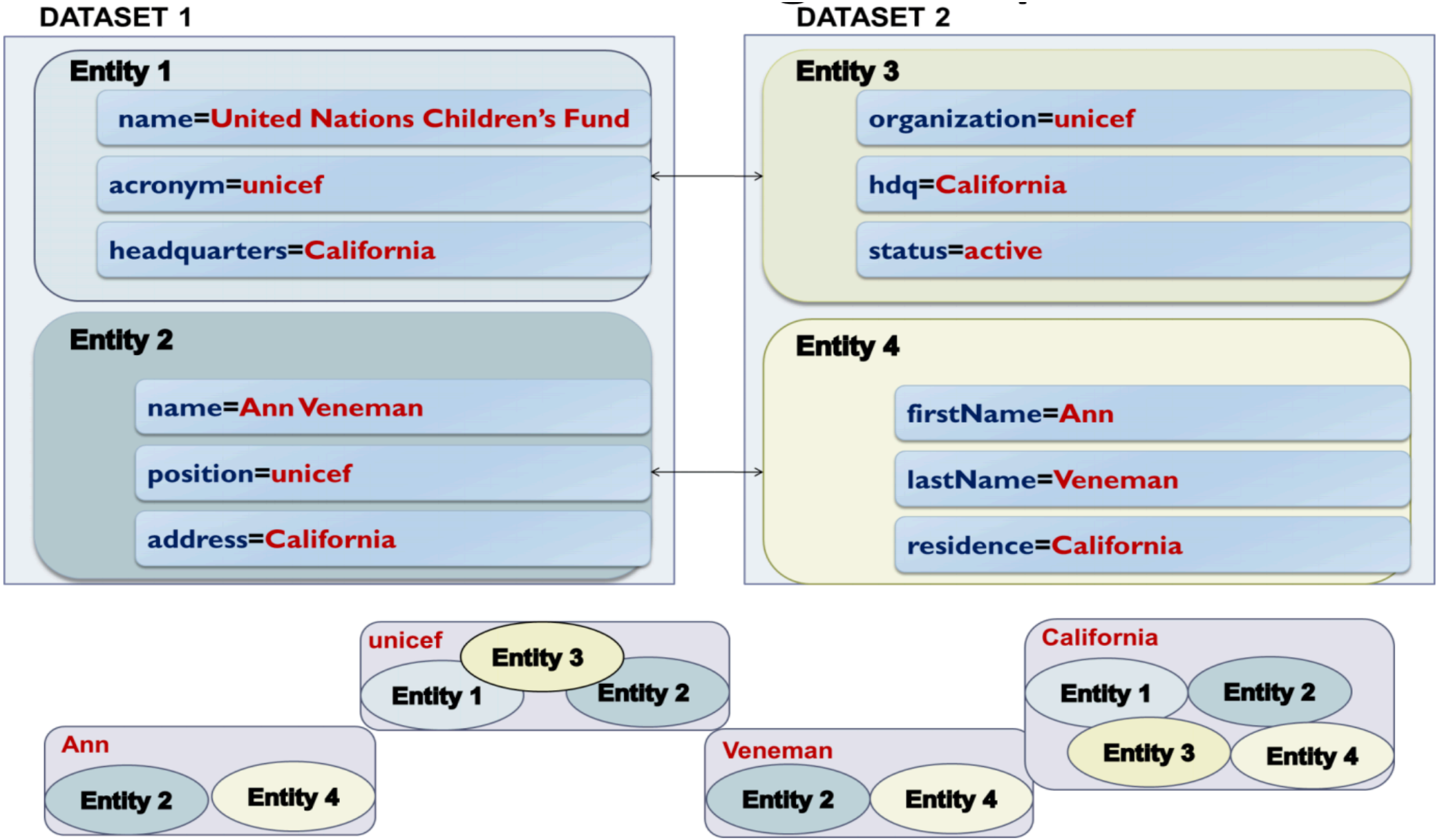
1. given an entity profile, it extracts all tokens that are contained in its attribute values.
2. creates one block for every distinct token → each block contains all entities with the corresponding token*.

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.

Example of Token Blocking



Attribute-Clustering Blocking

Goal:

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

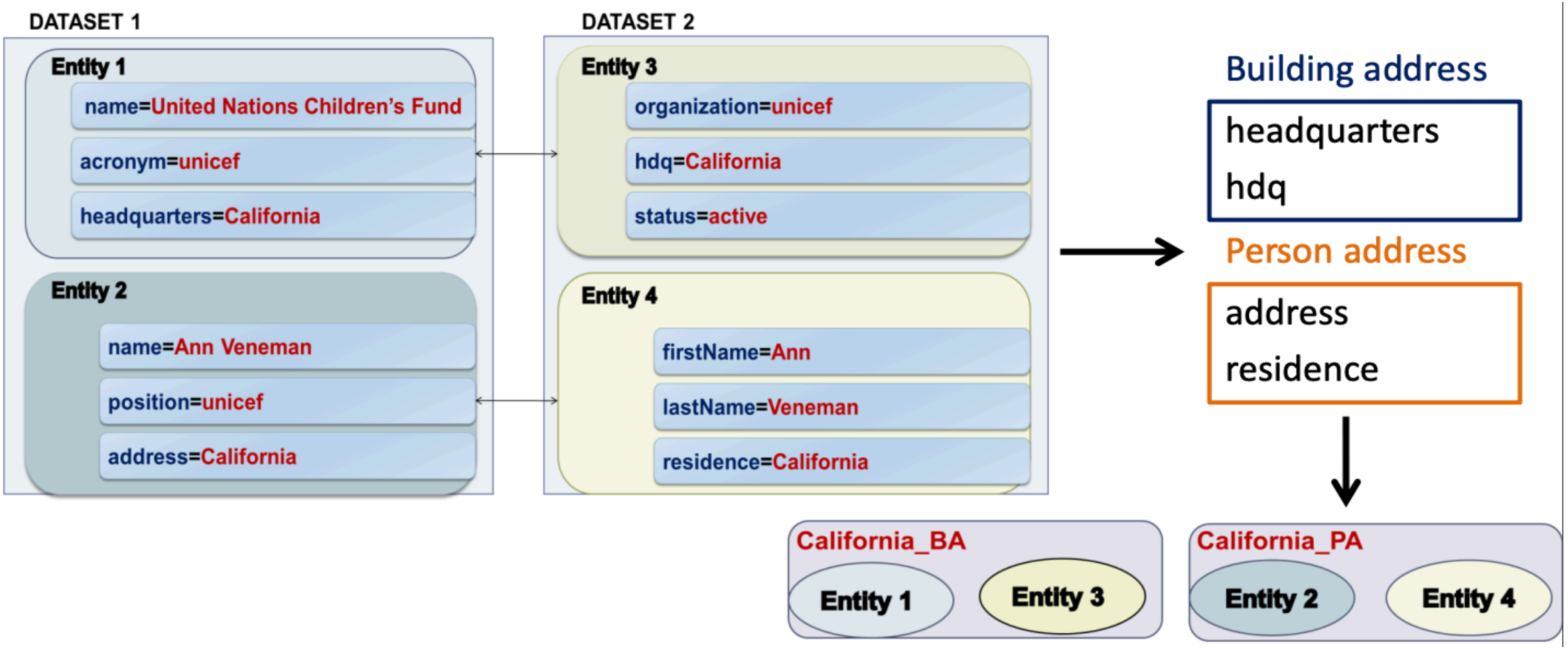
Algorithm

- Create a graph, where every node corresponds to an attribute name and aggregates 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
  - 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 Blocking Example



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.

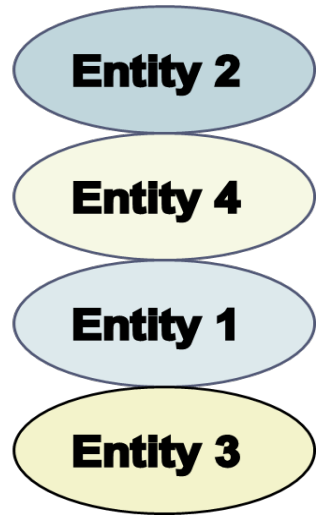
2. All singleton attribute names are associated with each other.

3. Unlike Schema Matching, it scales to the very high levels of heterogeneity of Web Data

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

Blocks on the similarity of BKs.

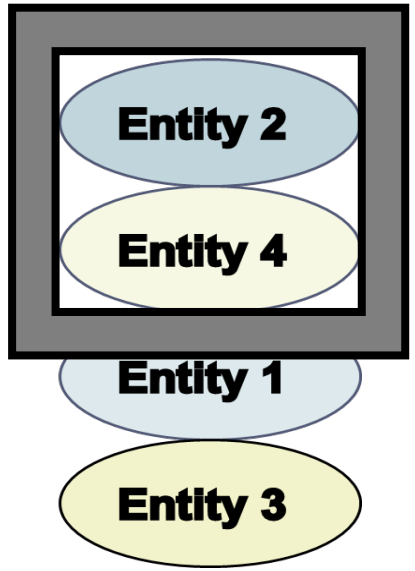
1. Entities are sorted in alphabetic order of BKs.
2. A window of fixed size slides over the sorted list of entities.
3. Entities within the window placed in the same block.



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

Blocks on the similarity of BKs.

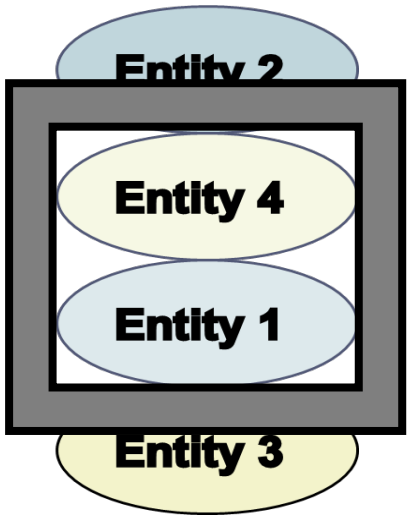
1. Entities are sorted in alphabetic order of BKs.
2. A window of fixed size slides over the sorted list of entities.
3. Entities within the window placed in the same block.



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

Blocks on the similarity of BKs.

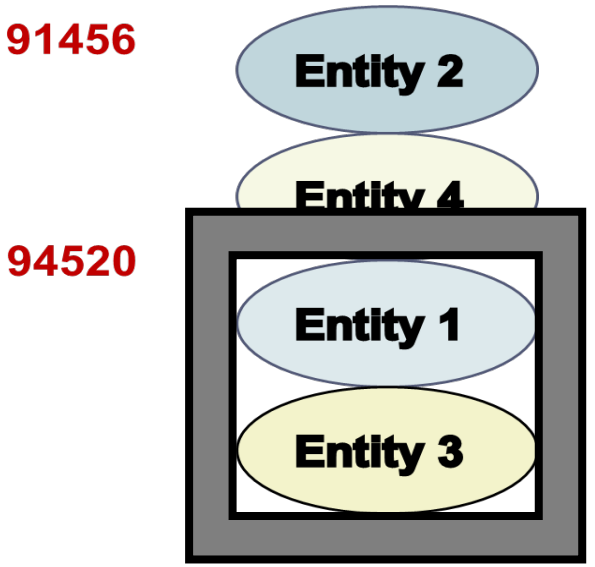
1. Entities are sorted in alphabetic order of BKs.
2. A window of fixed size slides over the sorted list of entities.
3. Entities within the window placed in the same block.



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

Blocks on the similarity of BKs.

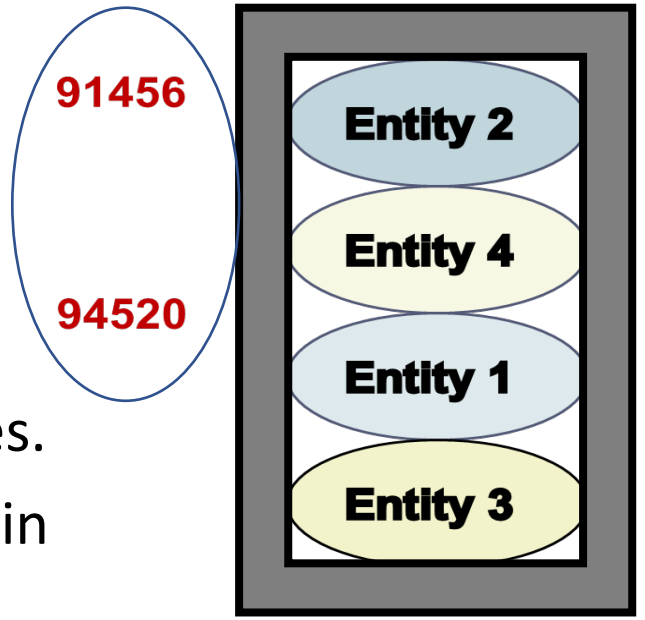
1. Entities are sorted in alphabetic order of BKs.
2. A window of fixed size slides over the sorted list of entities.
3. Entities within the window placed in the same block.



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. Entities within the window placed in the same block.



Extended Sorted Neighborhood [Christen, TKDE 2011]

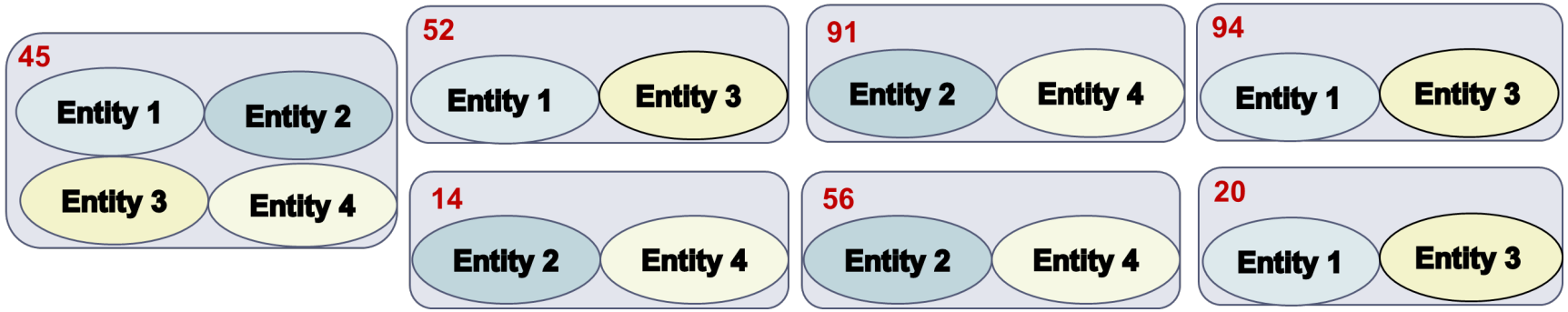
2'. A window of fixed size slides over the sorted list of **BKs**.

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:

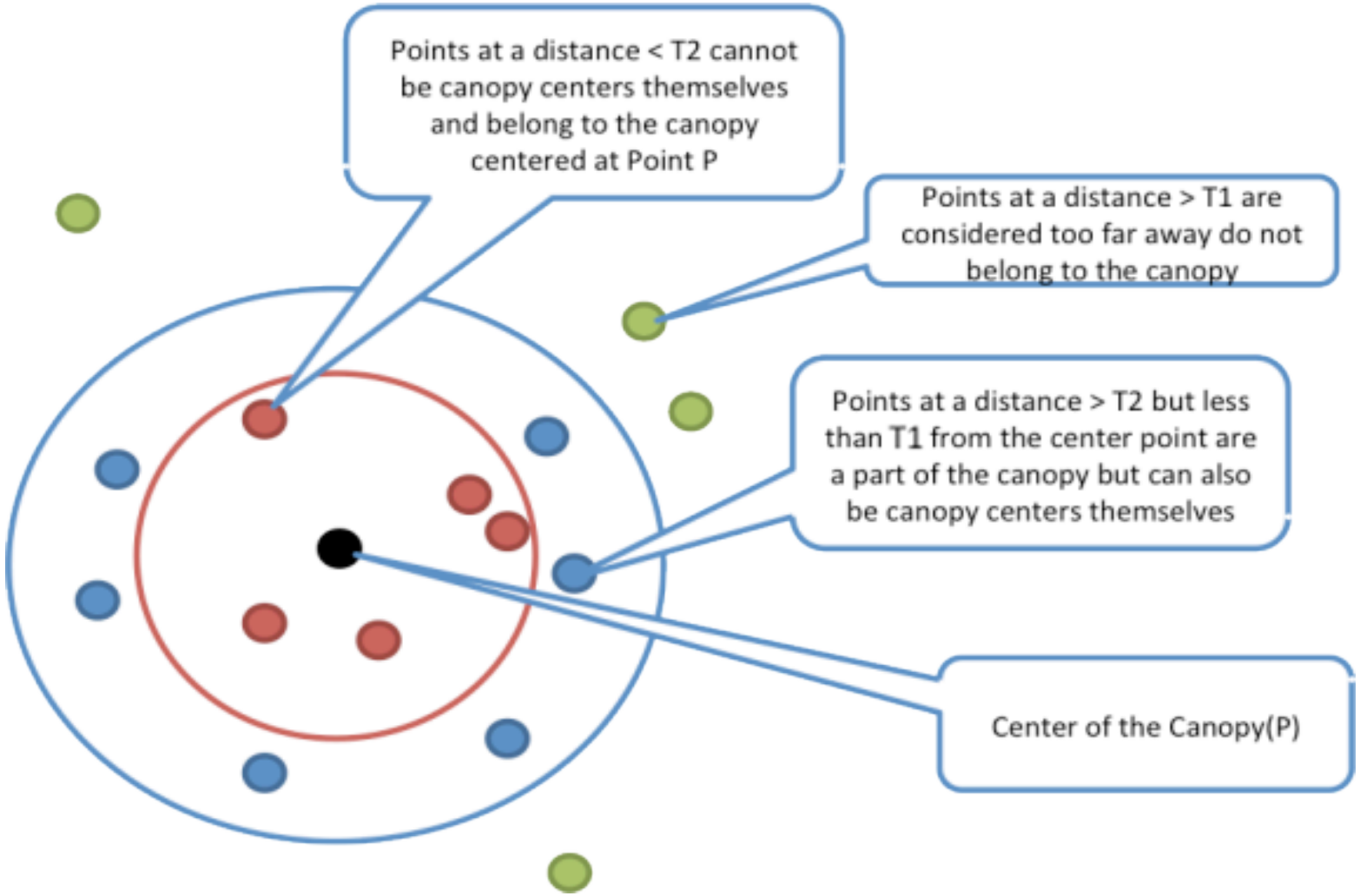
threshold $t \in (0,1)$ specifies the minimum number of *q*-grams per BK as follows: $\mathrm{l} = \boldsymbol{max}(\mathbf{1}, \lfloor \mathbf{k} \cdot \boldsymbol{t} \rfloor)$, where k is the number of *q*-grams from the original BK

Example:

for BK= 91456, q=2 and t=0.9,
we have l=3 and the following valid BKs:
91_14_45_56
91_14_45
91_14_56
91_45_56
14_45_56

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

- Same blocking keys as Q-grams Blocking
- 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]

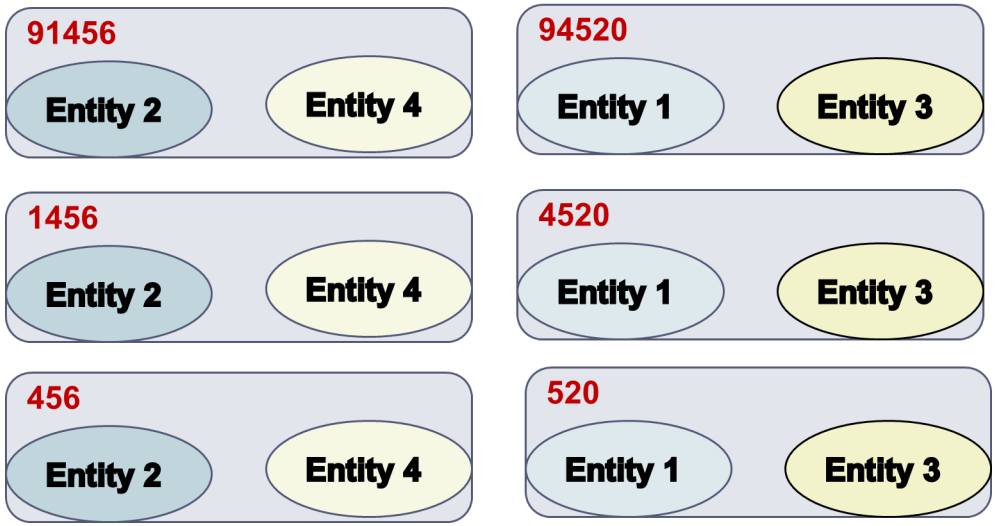
- cardinality thresholds instead of weight thresholds
- for each center of a canopy:
  - the n_1 nearest entities are placed in its block
  - the n_2 ($\leq n_1$) nearest entities are removed from the pool

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 $\mathbf{l_{min}}$.

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



Frequent suffixes are discarded with the help of the parameter $\mathbf{b_M}$:

- 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

Taxonomy of blocking methods

		Size Limit	
		Size-free	**Size-constrained**
Block Definition	**Equality-based**	• Standard Blocking (**StBl**) • Token Blocking (**tbl**) • Q-Grams Blocking (**QGBl**) • Extended Q-Grams Blocking (**EQGBl**)	• Suffix Arrays (**SuAr**) • Extended Suffix Arrays (**ESuAr**)
	Similarity-based	• Extended Sorted Neighborhood (**ESoNe**)	• Sorted Neighborhood (**SoNe**) • Canopy Clustering (**CaCl**) • Extended Canopy Clustering (**ECaCl**)

Summary of Blocking for Databases

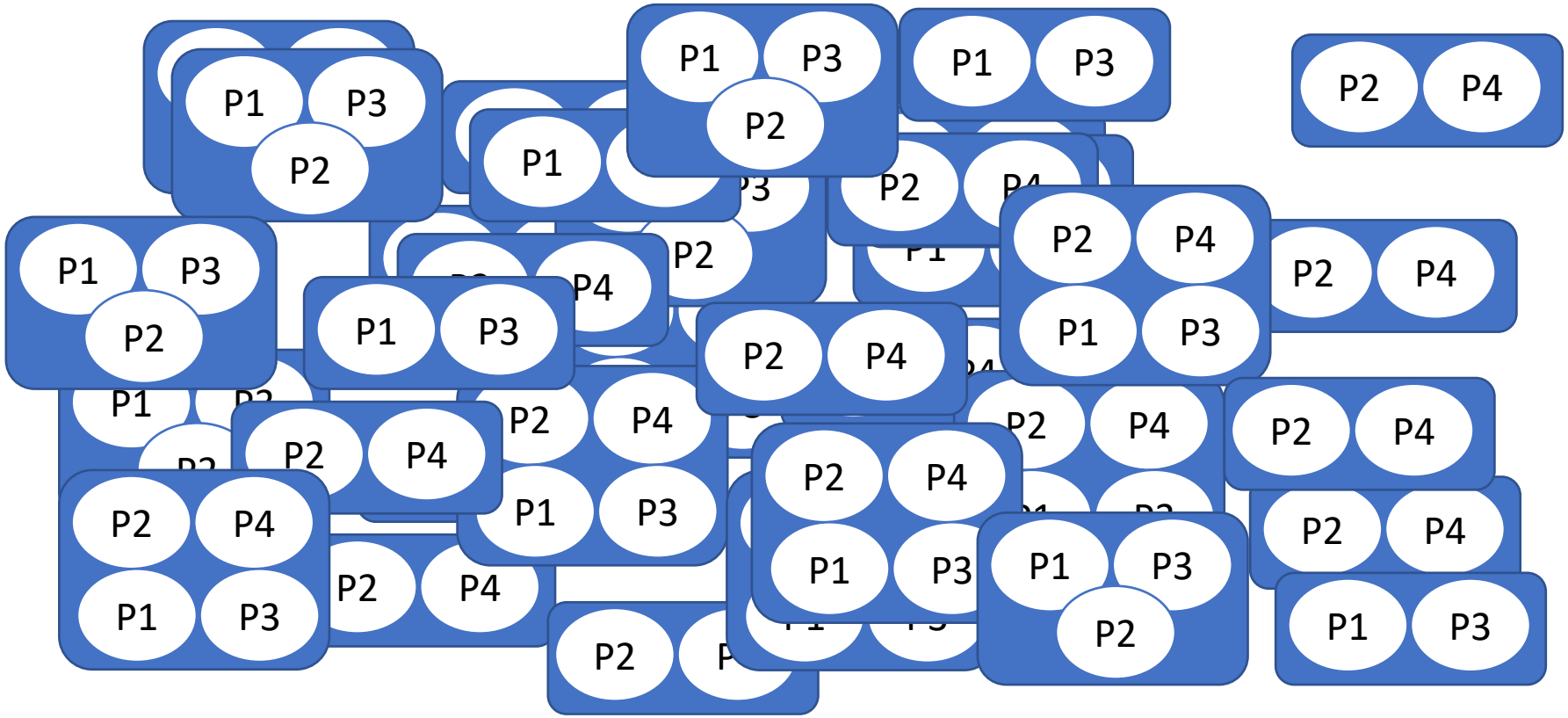
They typically employ **redundancy** to ensure robustness in the context of noise at the cost of lower efficiency.

Drawbacks:

1. Too many parameters to be configured Canopy Clustering has the following parameters:
   - String matching method
   - Threshold t_1
   - Threshold t_2
2. Schema-dependent -> manual definition of BKs

Blocking in Voluminous Data

Too many blocks – too many comparisons



Prune blocks & comparisons without missing matching entities

Meta-blocking

Goal:

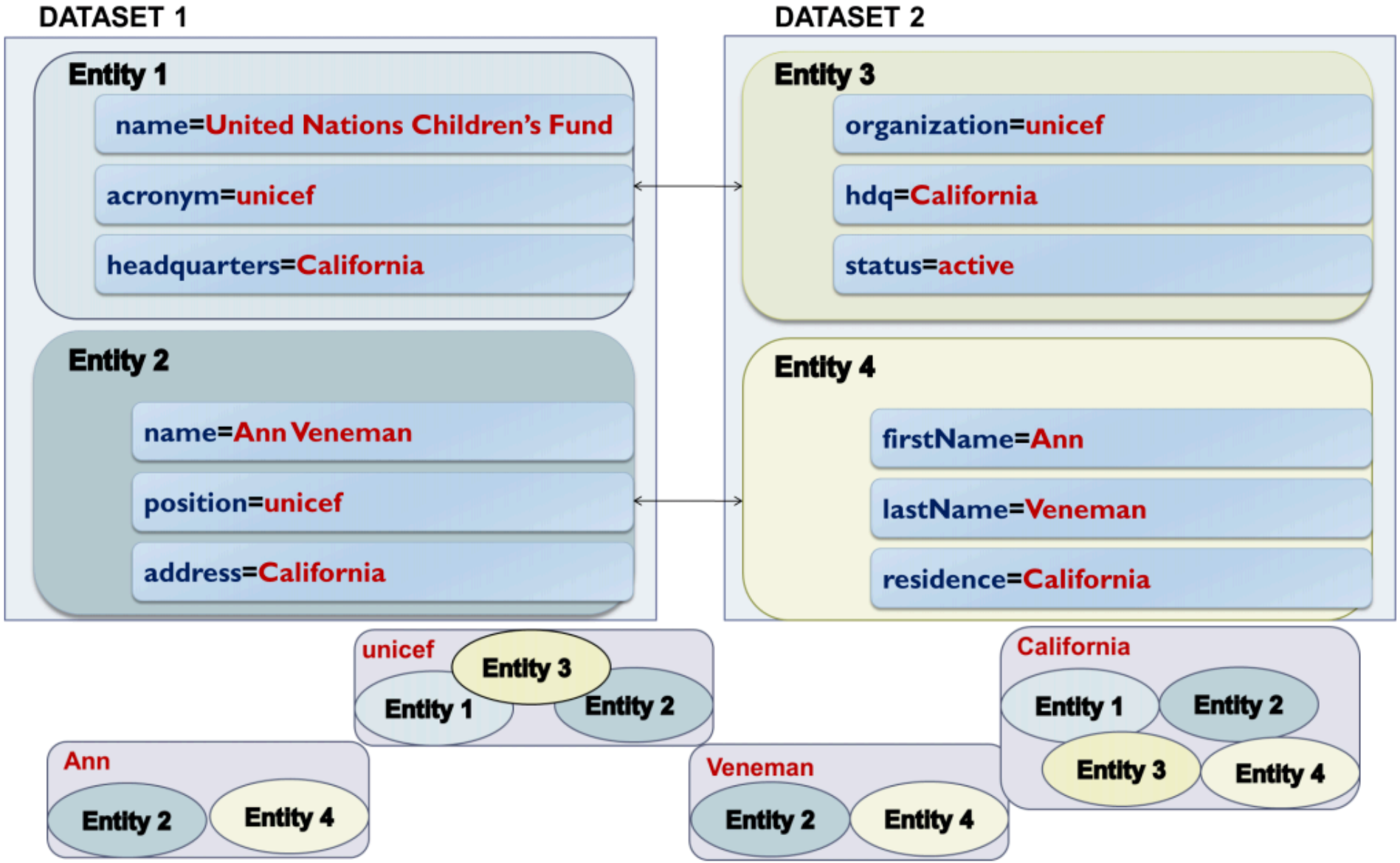
restructure a redundancy-positive block collection into a new one that contains a substantially lower number of comparisons, while being equally effective (ΔPC≈0, ΔPQ»0).

Types of pair-wise comparisons

Every comparison between entity profiles p_i and p_j belongs to one of the following types:

1. **Matching** if $p_i \equiv p_j$.
2. **Redundant** if p_i and p_j co-occur and will be compared in another block.
3. **Superfluous** if p_i or p_j or both of them have been matched to some other entity (Clean-Clean ER).
4. **Non-matching** if $p_i \neq p_j$ and the comparison is not redundant (for Dirty ER). For Clean-Clean ER, it should not be superfluous either.

Token Blocking Example



Meta-blocking

Goal:

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

Meta-blocking

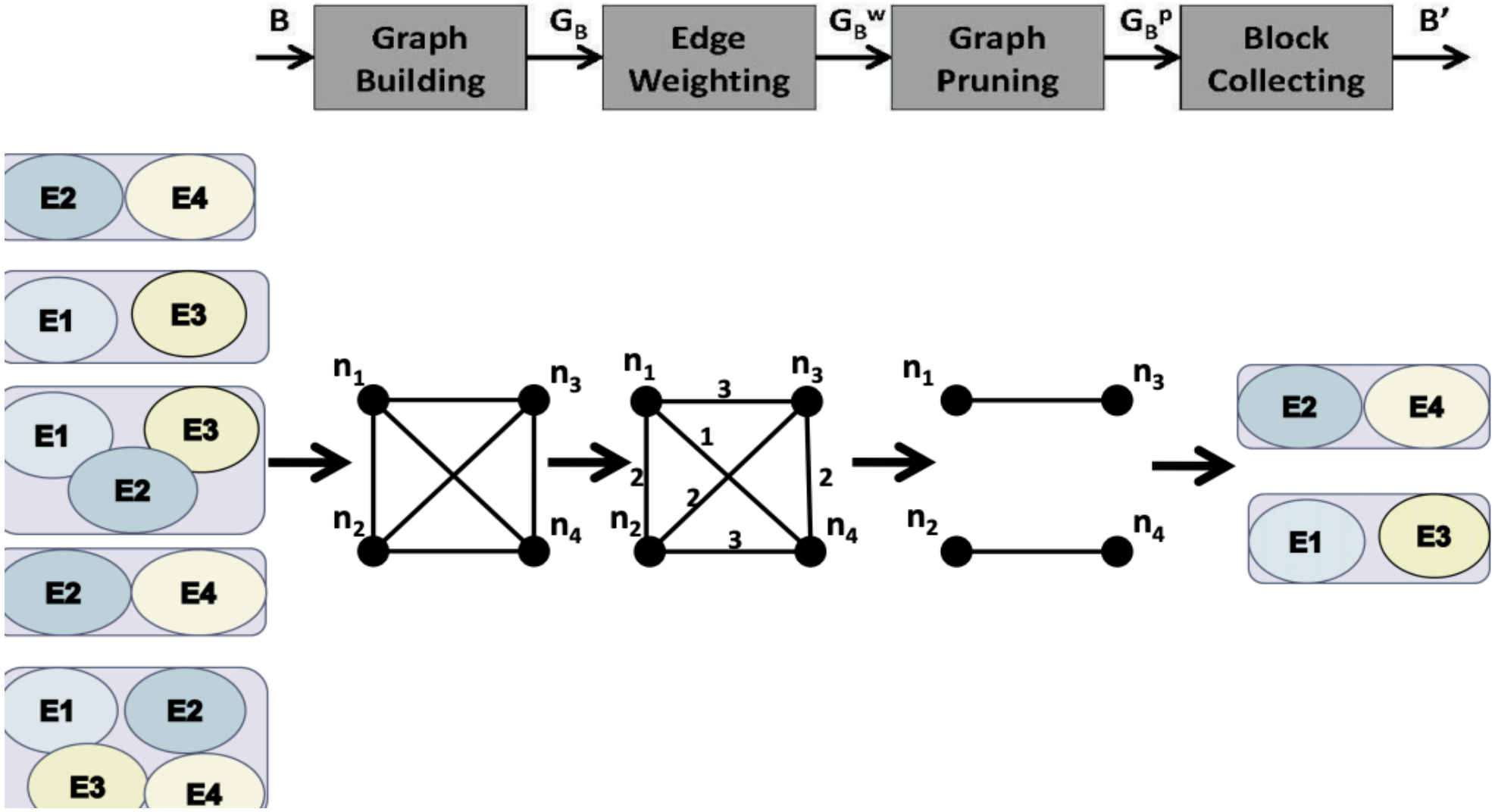
Goal:

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

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 or bit arrays.

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: minimum weight across all edges

2. Cardinality Edge Pruning (CEP)
   - threshold: K = BPE·|E|/2 3.

3. Weighted Node Pruning (WNP)
   - threshold: for each node, the average weight of the adjacent edges 4.

4. Cardinality Node Pruning (CNP)
   - threshold: for each node, k=BPE-1

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)

Block Collecting

Transform the pruned blocking graph into a new block collection.

For **undirected** blocking graphs:

- every retained edge creates a block of minimum size

For **directed** blocking graphs:

- for every node (with retained outgoing edges), we create a new block containing the corresponding entities

Block Processing Techniques

General Principles

Goals:

1. eliminate **repeated** comparisons,
2. discard **superfluous** comparisons,
3. avoid **non-matching** comparisons.

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

Taxonomy of techniques:

Granularity	Comparison's Type: Repeat Method	Superfluity Method	Non-match method	Scheduling method
Block-refinement	-	-	1. Block Purging 2. Block Pruning	Block Scheduling
Comparison-refinement	Comparison Propagation	Duplicate Propagation	Comparison Pruning	Comparison Scheduling

Block Purging

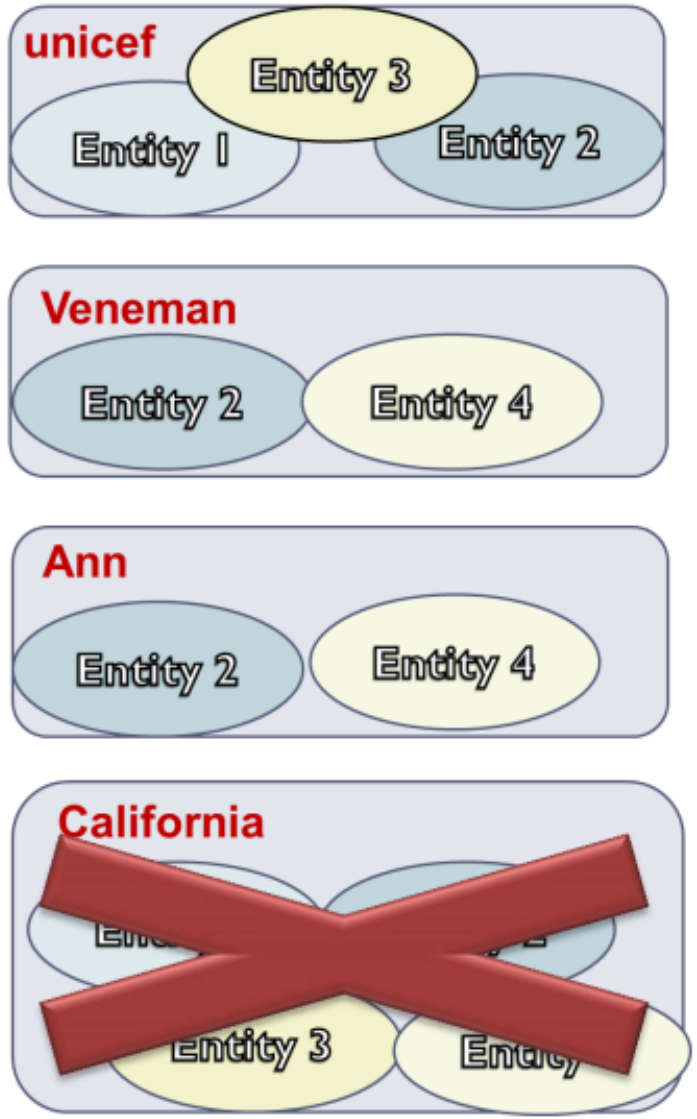
Oversized blocks: many, unnecessary comparisons (redundant, non-matching, superfluous).

Block Purging: discards oversized blocks 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]

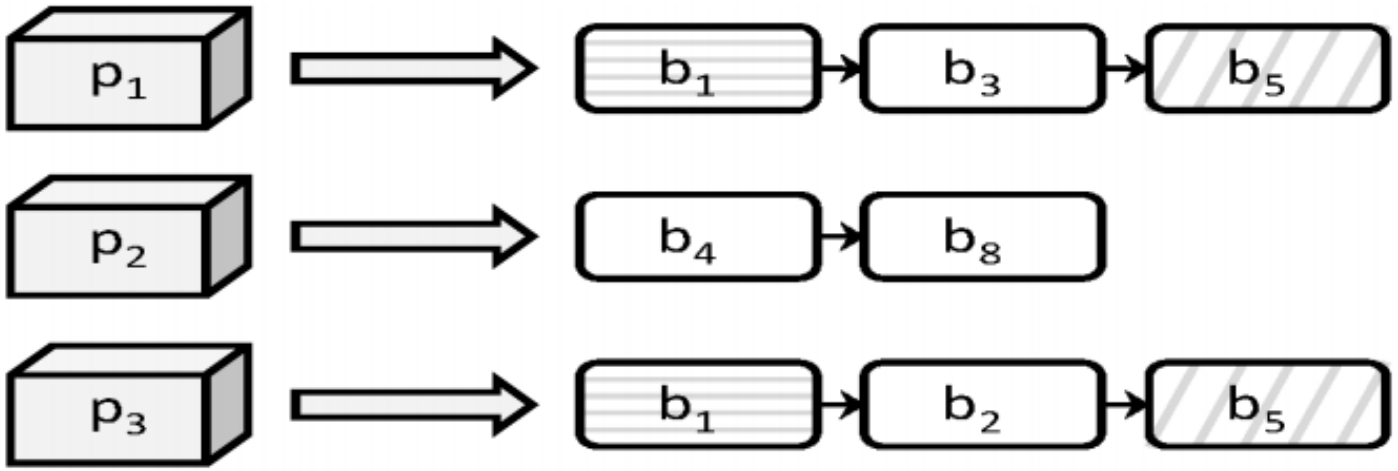
Core method:

- Low computational cost.
- Low impact on effectiveness.



Comparison Propagation

- Eliminates all redundant comparisons at no cost in recall
  → naïve approach does not scale
- Enumerates Blocks
- Least Common Block Index condition.



Comparison Execution

Matching features

Give two records , compute a “comparison” vector of similarity scores for corresponding features

- E.g., to match two bibliographical references, compute ⟨1st-author-match-score, title-match-score, venue-match-score, year-match-score, …⟩
- Score can be Boolean (match, or mismatch), or reals (based on some distance function)

Quick tour of matching features

- Difference between numeric values
- Domain-specific, like Jaro (for names)
- Edit distance: good for typos in strings
  - Levenshtein, Smith-Waterman, affine gap
- Phonetic-based
  - Soundex
- Translation-based
- Set similarity
  - Jaccard, Dice
  - For text fields (set of words) or relational features (e.g., set of authors of a paper)
- Vector-based
  - Cosine similarity, TF/IDF (good for text)

Jaro

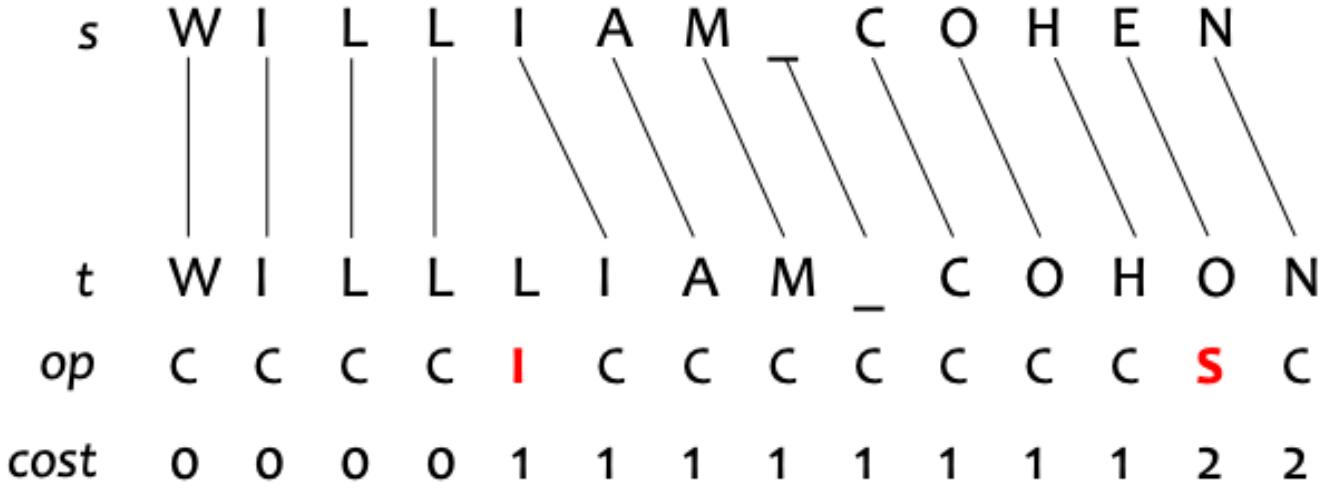
- Specifically designed for named by US Census
- The Jaro similarity sim_j of two given strings s_1 and s_2 is:

$$\text{Jaro similarity} = \frac{1}{3}\left(\frac{m}{|s|} + \frac{m}{|t|} + \frac{m-x}{2m}\right)$$

- Where:
  - $|s_i|$ is the length of the string s_i
  - **m** is the number of matching characters
  - **t** is half the number of transpositions
- Two characters from s_1 and s_2 respectively, are considered matching only if they are the same

Levenshtein

- Distance between strings s and t = shortest sequence of edit commands that transform s to t
  - Copy character from s over to t
  - Delete a character in s (cost 1)
  - Insert a character in t (cost 1)
  - Substitute one character for another (cost 1)



Computing Levenshtein

$D(i,j)$ = score of best alignment between $s_1 s_2 \cdots s_i$ and $t_1 t_2 \cdots t_j$

$$= \min \begin{cases} D(i-1,j-1) + d(s_i, t_j) & \text{sub/copy} \\ D(i-1,j) + 1 & \text{delete} \\ D(i,j-1) + 1 & \text{insert} \end{cases}$$

where $d(s_i, t_j) = \mathbf{1}[s_i \neq t_j]$,

and let $D(0,0) = 0, D(i,0) = i$, and $D(0,j) = j$

- Can then normalize using lengths of s and t:

$$1 - D(|s|, |t|) / \max(|s|, |t|)$$

Jaccard

- The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

- The Jaccard distance, which measures dissimilarity between sample sets, is complementary to the Jaccard coefficient and is obtained by:

$$d_J(A,B) = 1 - J(A,B)$$