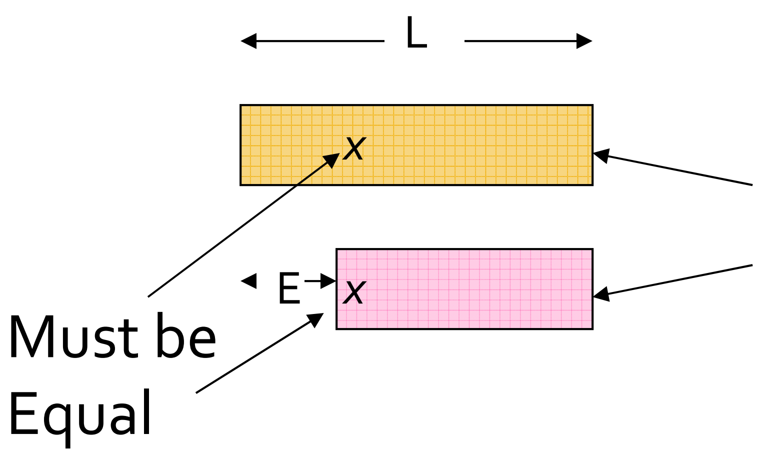
The Prefix of a String

Indexing by Symbols
Prefixes

Example: Prefix-Based Indexing

- If two strings are 90% similar, they must share some symbol in their prefixes whose length is just above 10% of the length of each string.
- Thus, we can base an index on symbols in just the first $\lfloor JL+1 \rfloor$ positions of a string of length L.
  - That's the *prefix* of the string.

Why the Limit on Prefixes?



Extreme case: second string has none of the first E symbols of the first string, but they agree thereafter.

If two strings do not share any of the first E symbols, then $J \geq E/L$.

Thus, $E = JL$ is possible, but any larger E is impossible. Index E+1 positions.

Indexing Prefixes

- Think of a bucket for each possible symbol.
- Each string of length L is placed in the bucket for each of its first $\lfloor JL+1 \rfloor$ positions.
- A B-tree with symbol as key leads to the strings.

Lookup

- Given a *probe* string *s* of length L, with J the limit on Jaccard distance:

```
for (each symbol a among the
 first ⌊JL+1⌋ positions of s)
   look for other strings in
    the bucket for a;
```

Example: Indexing Prefixes

- Let J = 0.2.
- String abcdef is indexed under *a* and *b*.
- String acdfg is indexed under *a* and *c*.
- String bcde is indexed only under *b*.
- If we search for strings similar to cdef, we need look only in the bucket for *c*.

Distance Measures

- Generalized LSH is based on some kind of "distance" between points.
  - Similar points are "close."
- Jaccard similarity is not a distance; 1 minus Jaccard similarity is.
- Two major classes of distance measure:

1. *Euclidean*
2. *Non-Euclidean*

Euclidean Vs. Non-Euclidean

- A *Euclidean space* has some number of real-valued dimensions and “dense” points.
  - There is a notion of “average” of two points.
  - A *Euclidean distance* is based on the locations of points in such a space.
- Any other space is *Non-Euclidean*.
  - Distance measures for non-Euclidean spaces are based on properties of points, but not their “location” in a space.

Axioms of a Distance Measure

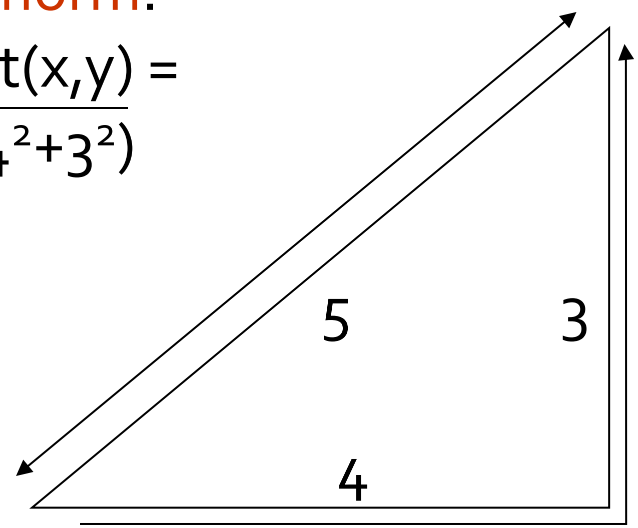
- *d* is a *distance measure* if it is a function from pairs of points to real numbers such that:
  1. $d(x,y) \geq 0$.
  2. $d(x,y) = 0$ iff $x = y$.
  3. $d(x,y) = d(y,x)$.
  4. $d(x,y) \leq d(x,z) + d(z,y)$ (*triangle inequality*).

Some Euclidean Distances

- *L_2 norm*: d(x,y) = square root of the sum of the squares of the differences between *x* and *y* in each dimension.
  - The most common notion of "distance."
- *L_1 norm*: sum of the differences in each dimension.
  - *Manhattan distance* = distance if you had to travel along coordinates only.

Examples of Euclidean Distances

L_2-norm:
dist(x,y) =
$\sqrt{(4^2+3^2)}$
= 5



L_1-norm:
dist(x,y) =
4+3 = 7

More Euclidean Distances

- *L_∞ norm*: d(x,y) = the maximum of the differences between *x* and *y* in any dimension.
- Note: the maximum is the limit as *r* goes to ∞ of the L_r norm: what you get by taking the r^{th} power of the differences, summing and taking the r^{th} root.

Non-Euclidean Distances

- *Jaccard distance* for sets = 1 minus Jaccard similarity.
- *Cosine distance* for vectors = angle between the vectors.
- *Edit distance* for strings = number of inserts and deletes to change one string into another.
- *Hamming Distance* for bit vectors = the number of positions in which they differ.

Example: Jaccard Distance

- Consider x = {1,2,3,4} and y = {1,3,5}
- Size of intersection = 2; size of union = 5, Jaccard similarity (not distance) = 2/5.
- d(x,y) = 1 – (Jaccard similarity) = 3/5.

Why J.D. Is a Distance Measure

- $d(x,y) \geq 0$ because $|x \cap y| \leq |x \cup y|$.
- $d(x,x) = 0$ because $x \cap x = x \cup x$.
  - And if $x \neq y$, then the size of $x \cap y$ is strictly less than the size of $x \cup y$.
- $d(x,y) = d(y,x)$ because union and intersection are symmetric.
- $d(x,y) \leq d(x,z) + d(z,y)$ trickier – next slide.

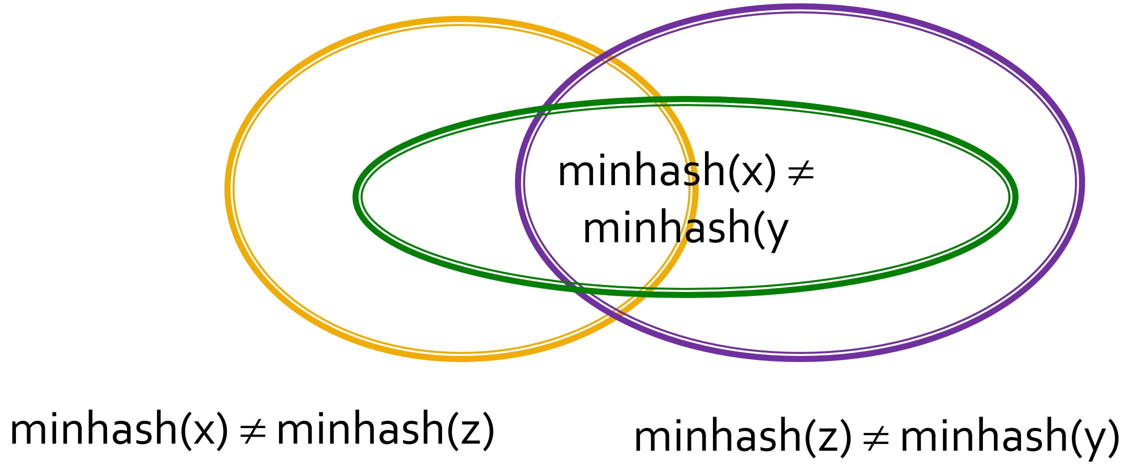
Triangle Inequality for J.D.

$$1 - \frac{|x \cap z|}{|x \cup z|} + 1 - \frac{|y \cap z|}{|y \cup z|} \geq 1 - \frac{|x \cap y|}{|x \cup y|}$$

- Remember: $|a \cap b|/|a \cup b|$ = probability that minhash(a) = minhash(b).
- Thus, $1 - |a \cap b|/|a \cup b|$ = probability that minhash(a) $\neq$ minhash(b).

Triangle Inequality – (2)

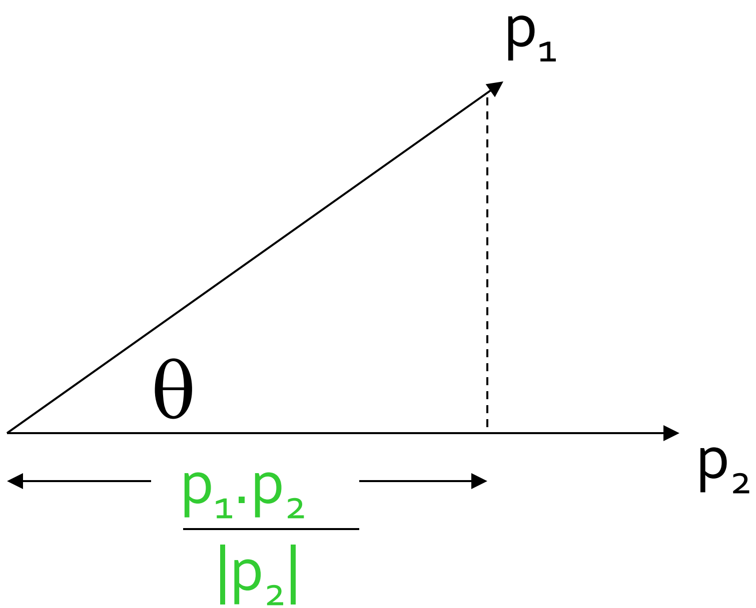
- Claim: prob[minhash(x) $\neq$ minhash(y)] $\leq$ prob[minhash(x) $\neq$ minhash(z)] + prob[minhash(z) $\neq$ minhash(y)]
- Proof: whenever minhash(x) $\neq$ minhash(y), at least one of minhash(x) $\neq$ minhash(z) and minhash(z) $\neq$ minhash(y) must be true.



Cosine Distance

- Think of a point as a vector from the origin (0,0,…,0) to its location.
- Two points' vectors make an angle, whose cosine is the normalized dot-product of the vectors: $p_1.p_2/|p_2||p_1|$.
  - Example: p_1 = 00111; p_2 = 10011.
  - $p_1.p_2 = 2$; $|p_1| = |p_2| = \sqrt{3}$.
  - $\cos(\theta) = 2/3$; θ is about 48 degrees.

Cosine-Measure Diagram



$d\ (p_1, p_2) = \theta = \arccos(p_1.p_2/|p_2||p_1|)$

Why C.D. Is a Distance Measure

- d(x,x) = 0 because arccos(1) = 0.
- d(x,y) $\geq$ 0 because any two intersecting vectors make an angle in the range 0 to 180 degrees.
- d(x,y) = d(y,x) by symmetry.
- Triangle inequality: physical reasoning. If I rotate an angle from *x* to *z* and then from *z* to *y*, I can't rotate less than from *x* to *y*.

Edit Distance

- The *edit distance* of two strings is the number of inserts and deletes of characters needed to turn one into the other.
- An equivalent definition: $d(x,y) = |x| + |y| - 2|LCS(x,y)|$.
  - LCS = *longest common subsequence* = any longest string obtained both by deleting from *x* and deleting from *y*.

Example

- *x* = *abcde* ; *y* = *bcduve*.
- Turn *x* into *y* by deleting *a*, then inserting *u* and *v* after *d*.
  - Edit distance = 3.
- Or, LCS(x,y) = *bcde*.
- Note: |x| + |y| - 2|LCS(x,y)| = 5 + 6 –2*4 = 3 = edit distance.

Why Edit Distance Is a Distance Measure

- $d(x,x) = 0$ because 0 edits suffice.
- $d(x,y) \geq 0$: no notion of negative edits.
- $d(x,y) = d(y,x)$ because insert/delete are inverses of each other.
- Triangle inequality: changing *x* to *z* and then to *y* is one way to change *x* to *y*.

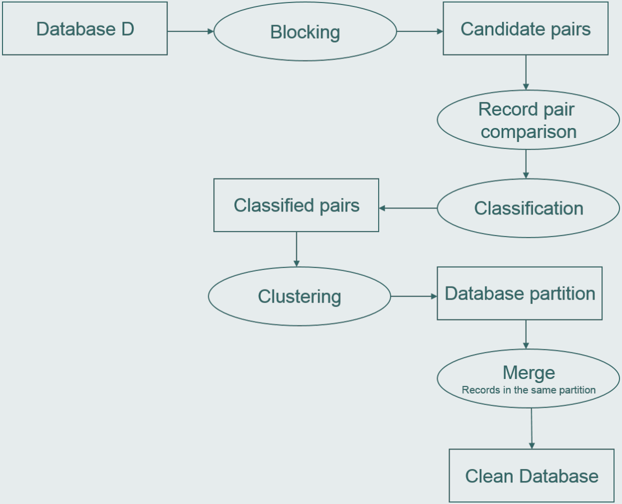
Hamming Distance

- *Hamming distance* is the number of positions in which bit-vectors differ.
- Example: p_1 = 10101; p_2 = 10011.
- $d(p_1, p_2) = 2$ because the bit-vectors differ in the 3rd and 4th positions.

Why Hamming Distance Is a Distance Measure

- d(x,x) = 0 since no positions differ.
- d(x,y) $\geq$ 0 since strings cannot differ in a negative number of positions.
- d(x,y) = d(y,x) by symmetry of "different from."
- Triangle inequality: changing *x* to *z* and then to *y* is one way to change *x* to *y*.

The ER Process



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

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!)

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!

Fundamental Assumptions

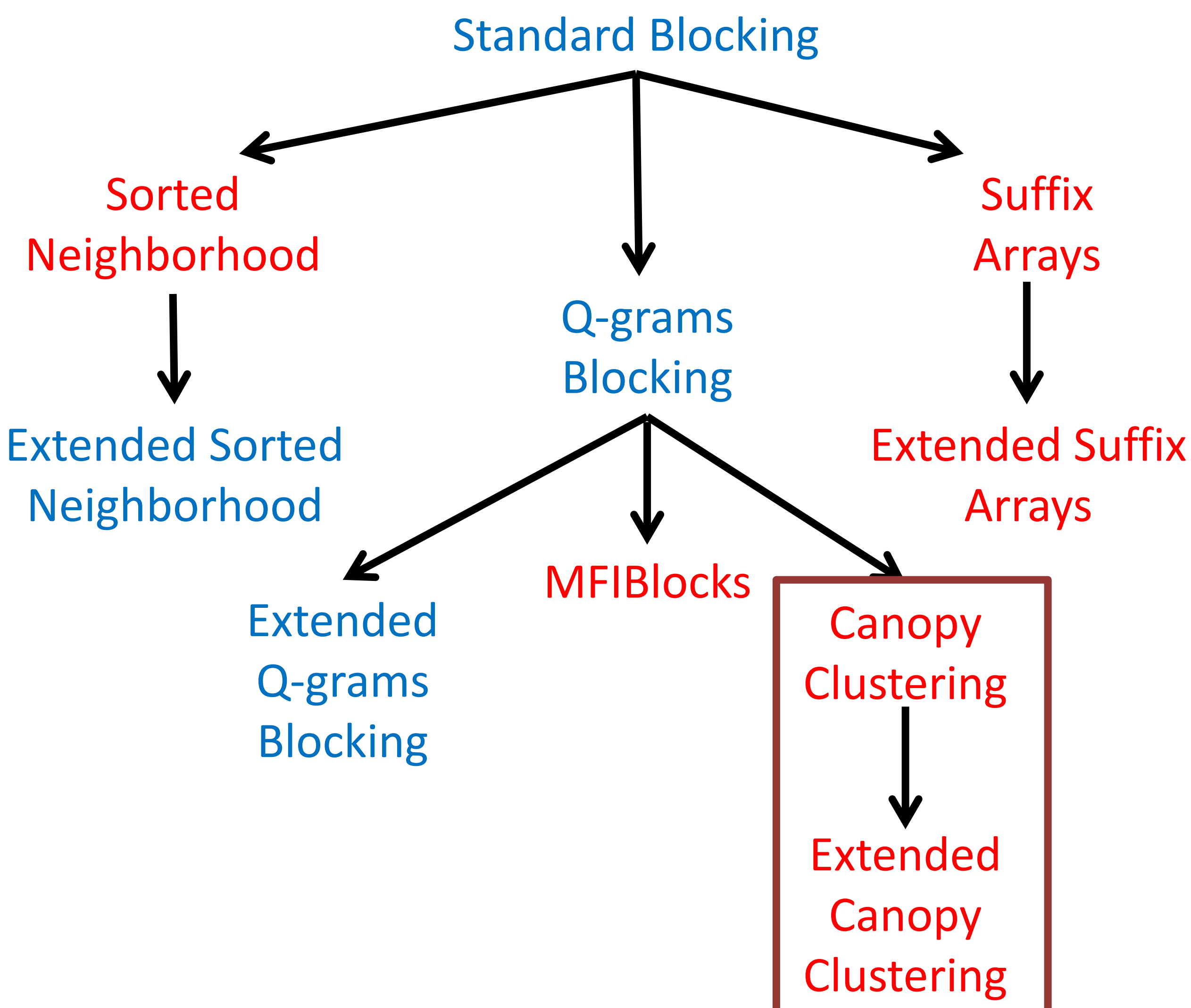
1. Every entity profile consists of a *uniquely identified* set of name-value pairs.

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.

Overview of Schema-based Methods



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

Token Blocking [Papadakis et al., WSDM2011]

Functionality:

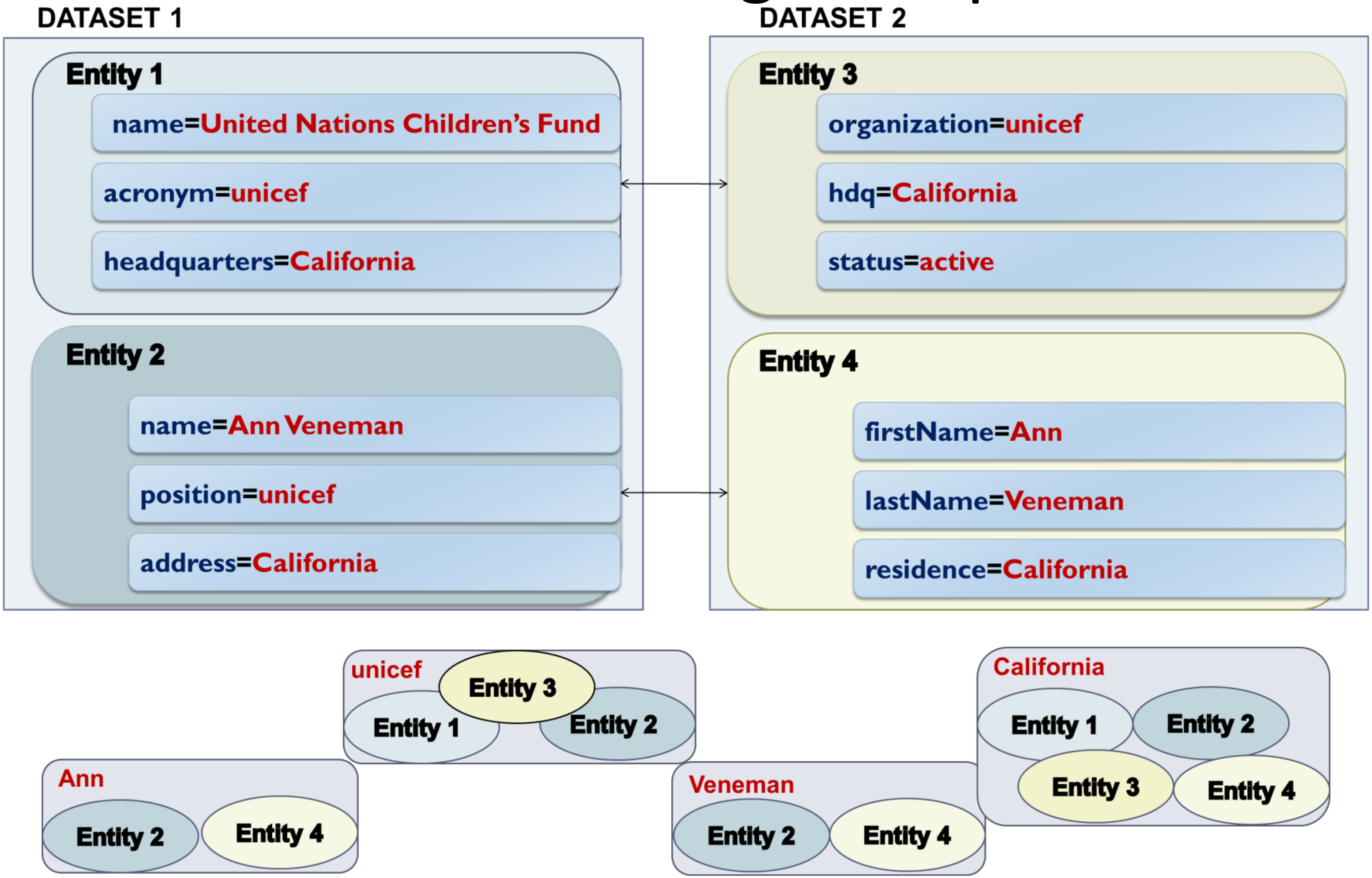
1. given an entity profile, it extracts all tokens that are contained in its attribute values.
2. creates one block for every distinct token → each block contains all entities with the corresponding token*****.

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.*

Token Blocking Example

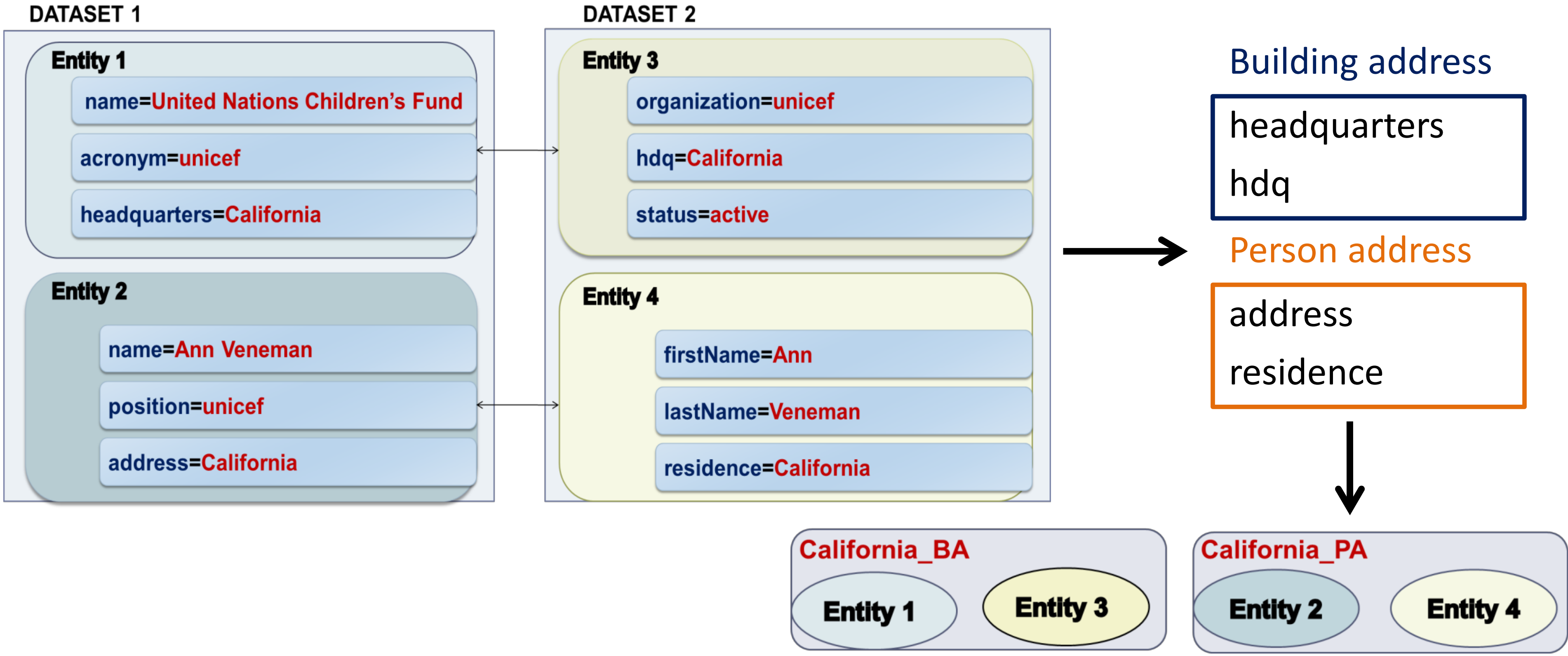


Attribute-Clustering Blocking

[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 Functionality

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 $<n_i,n_j>$
- 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.