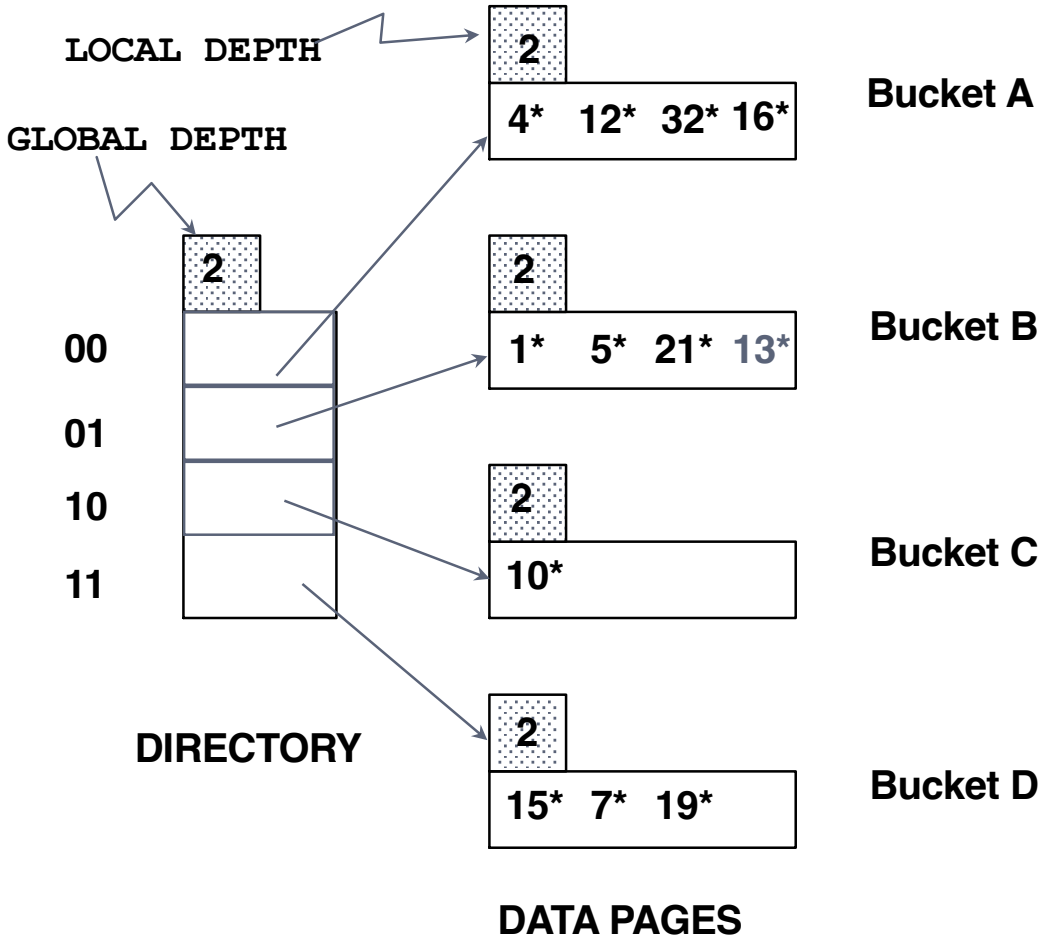
Hash Functions



Hash function = last 2 bits in its binary representation

Finding Similar Items

Acknowledgements

- Slides provided by:
  - L. Leskovec, A. Rajaraman, J. Ullman
  - Themis Palpanas, George Papadakis, Ekaterini Ioannou
  - Avigdor Gal
  - Yannis Velegrakis

Connection to Previous Lecture

- Entity Identification is based on similarity
  - i.e., finding similar items
- For small number of entities of relativelly small size things are manageable.
  - What if we have too many entities ?
  - What if the entities are too large?
    - Too many comparisons (Time)
    - Too large memory requirements (Space)
- Approach: See entities as a set
  - Finding similar items reduced to finding similar sets

Problem Statement

- **Given:**
  - **A set of high dimensional data points**
  - **And some distance function**
    - Which quantifies the "distance" between data points
- **Goal:**
  - Find **all pairs of data points** that are within some distance threshold

Applications of Set-Similarity

Many data−mining problems can be expressed as finding "similar" sets:

1. Entity resolution.
2. Pages with similar words
   - For classification by topic
   - For plagiarism
   - For repetition avoidance
3. NetFlix users with similar tastes in movies
   - For recommendation systems.
4. Dual: movies with similar sets of fans.
   - For movie classification

Focus on Finding Similar Documents

- Goal: Given a large number (N in the millions or billions) of documents, find "near duplicate" pairs
- Problems:
  - Too many documents to compare all pairs
  - Documents are so large or so many that they cannot fit in main memory
  - Many small pieces of one document can appear out of order in another (i.e., order is important)

Converting Documents to Sets

- Set of “words” (or “important words”) not effective
- Shingling
  - Generating a set of strings of length k that appear in the document
- A k –shingle (or k –gram) for a document is a sequence of k characters or k works or any other token
  - Example: k=2; doc = abcab. Doc2=abfef
    - Set of 2–shingles= {ab, bc, ca}. 2-shingles2={ab bf fe ef}
    - [ab bc ca bf fe ef]
    - D1: [1,1,1,0,0,0]. D2: [1,0, 0, 1,1,,1]
- A doc is represented by its set of k–shingles
  - Each document is an 0/1 Vector in the space of shingles
  - Sparse Vectors

Converting Documents to Sets

- Documents that are intuitively similar will have many shingles in common
- Changing a word only affects k−shingles within distance k from the word.
- Reordering paragraphs only affects the 2k shingles that cross paragraph boundaries.
  - Example: k=3, “The dog which chased the cat” versus “The dog that chased the cat”.
    - Only 3−shingles replaced are:

      g_w, _wh, whi, hic, ich, ch_, and h_c

Challenges in using shingles

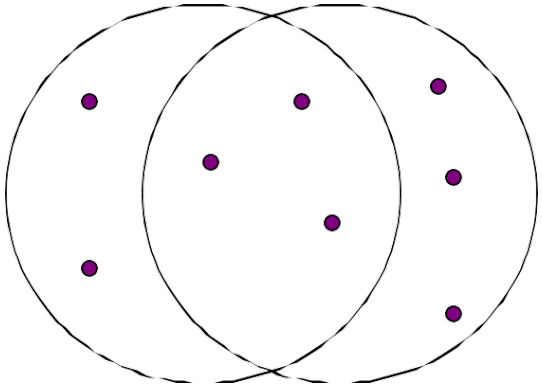
- Selecting the right k
  - K=5 is ok for short documents
  - K=10 is better for long documents
- Selecting the right token
- Hashing Shingles
  - Represent a doc by its tokens, that is, the set of hash values of its k−shingles.
  - Two documents could (rarely) appear to have shingles in common, when in fact only the hash−values were shared.
  - **Example: k=2**; document $\mathbf{D_1}$= abcab
    Set of 2-shingles: $\mathbf{S(D_1)}$ = {ab, bc, ca}
    Hash the singles: $\mathbf{h(D_1)}$ = {1, 5, 7}

Computing Similarities between Document Representations as Sets

Jaccard Similarity

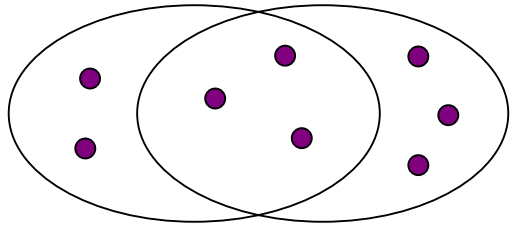
- A natural similarity measure
- The Jaccard similarity of two sets is the size of their intersection divided by the size of their union.
- Jaccard(C1, C2) = $|C1 \cap C2| / |C1 \cup C2|$.

Example: Jaccard Similarity



3 in intersection.
8 in union.
Jaccard similarity
= 3/8

Encoding Sets as Bit Vectors



- Many similarity problems can be formalized as **finding subsets that have significant intersection**
- **Encode sets using 0/1 (bit, boolean) vectors**
  - One dimension per element in the universal set
- Interpret set intersection as bitwise **AND**, and set union as bitwise **OR**

- **Example:** $\mathbf{C_1}$ = 10111; $\mathbf{C_2}$ = 10011
  - Size of intersection **= 3**; size of union **= 4**,
  - **Jaccard similarity** (not distance) **= 3/4**
  - **Distance: d(C_1,C_2) = 1 – (Jaccard similarity) = 1/4**

The challenge

- **Suppose we need to find near-duplicate documents among $N = 1$ million documents**
- Naïvely, we would have to compute **pairwise Jaccard similarities** for **every pair of docs**
  - $\boldsymbol{N(N-1)/2 \approx 5*10^{11}}$ comparisons
  - At 10^5 secs/day and 10^6 comparisons/sec, it would take **5 days**
- For $\boldsymbol{N = 10}$ million, it takes more than a year…

The solution

- Construct signatures of the items we compare and compare the signatures instead
  - Signatures being much smaller of course
- *Minhashing* : convert large sets to short signatures, while preserving similarity.
- [X, Y] X = 0 or 1, Y=0 or 1
- [0,1] D1. [1,1]. D2

From Sets to Boolean Matrices

Rows = elements of the universal set.

Example: the set of all k–shingles.

Columns = sets.

1 in row *e* and column *S* if and only if *e* is a member of *S*.

Column similarity is the Jaccard similarity of the sets of their rows with 1.

Typical matrix is sparse.

Example: Column Similarity

C_1	C_2		
0	1		*
1	0		*
1	1	*	*
0	0		
1	1	*	*
0	1		*

$Sim(C_1, C_2) =$

$2/5 = 0.4$

Four Types of Rows

Given columns C_1 and C_2, rows may be classified as:

	C_1	C_2
a	1	1
b	1	0
c	0	1
d	0	0

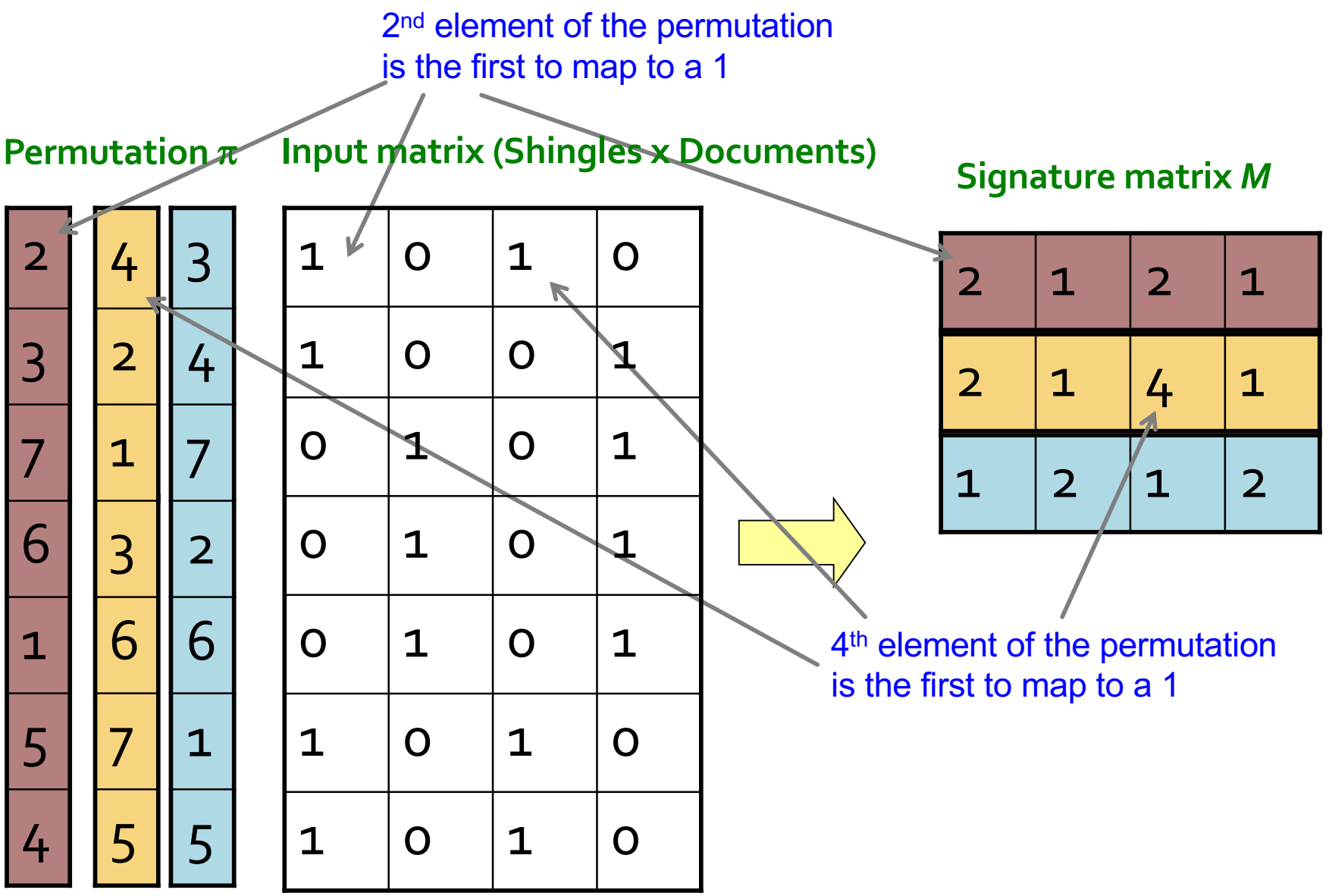
Also, *a* = # rows of type *a* , etc.

Note $Sim(C_1, C_2) = a/(a + b + c)$.

Minhashing

Imagine the rows permuted randomly.
Define *minhash function* $h(C)$ = the number of the first (in the permuted order) row in which column C has 1.
Use several (e.g., 100) independent hash functions to create a signature for each column.
The signatures can be displayed in another matrix – the *signature matrix* – whose columns represent the sets and the rows represent the minhash values, in order for that column.

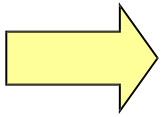
Min-Hashing Example



Minhashing Example

Input matrix

1	4	3	1	0	1	0
3	2	4	1	0	0	1
7	1	7	0	1	0	1
6	3	6	0	1	0	1
2	6	1	0	1	0	1
5	7	2	1	0	1	0
4	5	5	1	0	1	0



Signature matrix *M*

2	1	2	1
2	1	4	1
1	2	1	2

Surprising Property

The probability (over all permutations of the rows) that $h(C_1) = h(C_2)$ is the same as $Sim(C_1, C_2)$.

Both are $a/(a+b+c)$!

Why?

Look down the permuted columns C_1 and C_2 until we see a 1.

If it's a type–*a* row, then $h(C_1) = h(C_2)$. If a type–*b* or type–*c* row, then not.

Similarity for Signatures

The *similarity of signatures* is the fraction of the minhash functions in which they agree.

Thinking of signatures as columns of integers, the similarity of signatures is the fraction of rows in which they agree.

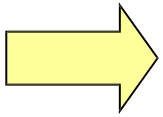
Thus, the expected similarity of two signatures equals the Jaccard similarity of the columns or sets that the signatures represent.

And the longer the signatures, the smaller will be the expected error.

Min Hashing – Example

Input matrix

1	4	3	1 0 1 0
3	2	4	1 0 0 1
7	1	7	0 1 0 1
6	3	6	0 1 0 1
2	6	1	0 1 0 1
5	7	2	1 0 1 0
4	5	5	1 0 1 0



Signature matrix *M*

2	1	2	1
2	1	4	1
1	2	1	2

	1-3	2-4	1-2
Col/Col	0.75	0.75	0
Sig/Sig	0.67	1.00	0

Implementation of Minhashing

Suppose 1 billion rows.
Hard to pick a random permutation of 1…billion.
Representing a random permutation requires 1 billion entries.
Accessing rows in permuted order leads to thrashing.

Implementation Trick

- **Permuting rows even once is prohibitive**
- **Row hashing!**
  - Pick **K = 100** hash functions k_i
  - Ordering under k_i gives a random row permutation!
- **One-pass implementation**
  - For each column ***C*** and hash-func. k_i keep a "slot" for the min-hash value
  - Initialize all ***sig(C)[i] =*** ∞
  - **Scan rows looking for 1s**
    - Suppose row ***j*** has 1 in column ***C***
    - Then for each k_i :
      - If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

Implementation – (3)

for each row *r* **do begin**
  for each hash function h_i **do**
    compute $h_i(r)$;
  for each column *c*
    if the document c has 1 in row *r*
      for each hash function h_i **do**
        if $h_i(r)$ is smaller than $M(i, c)$ **then**
          $M(i, c) := h_i(r)$;
end;

Example

	Sig1	Sig2
$h(1) = 1$	1	∞
$g(1) = 3$	3	∞

Row	C1	C2
1	1	0
2	0	1
3	1	1
4	1	0
5	0	1

$h(x) = x \bmod 5$
$g(x) = (2x+1) \bmod 5$

Example

Row	C1	C2
1	1	0
2	0	1
3	1	1
4	1	0
5	0	1

	Sig1	Sig2
$h(1) = 1$	1	∞
$g(1) = 3$	3	∞
$h(2) = 2$	1	2
$g(2) = 0$	3	0

$h(x) = x \bmod 5$
$g(x) = (2x+1) \bmod 5$

Example

Row	C1	C2
1	1	0
2	0	1
3	1	1
4	1	0
5	0	1

	Sig1	Sig2
$h(1) = 1$	1	∞
$g(1) = 3$	3	∞
$h(2) = 2$	1	2
$g(2) = 0$	3	0
$h(3) = 3$	1	2
$g(3) = 2$	2	0

$h(x) = x \bmod 5$
$g(x) = (2x+1) \bmod 5$

Example

Row	C1	C2
1	1	0
2	0	1
3	1	1
4	1	0
5	0	1

$h(x) = x \bmod 5$
$g(x) = (2x+1) \bmod 5$

	Sig1	Sig2
$h(1) = 1$	1	∞
$g(1) = 3$	3	∞
$h(2) = 2$	1	2
$g(2) = 0$	3	0
$h(3) = 3$	1	2
$g(3) = 2$	2	0
$h(4) = 4$	1	2
$g(4) = 4$	2	0

Example

Row	C1	C2
1	1	0
2	0	1
3	1	1
4	1	0
5	0	1

$h(x) = x \bmod 5$
$g(x) = (2x+1) \bmod 5$

	Sig1	Sig2
$h(1) = 1$	1	∞
$g(1) = 3$	3	∞
$h(2) = 2$	1	2
$g(2) = 0$	3	0
$h(3) = 3$	1	2
$g(3) = 2$	2	0
$h(4) = 4$	1	2
$g(4) = 4$	2	0
$h(5) = 0$	1	0
$g(5) = 1$	2	0

Implementation – (4)

Often, data is given by column, not row.

Example: columns = documents, rows = shingles.

If so, sort matrix once so it is by row.

Focusing on Similar MinHash Signatures

- *Locality–sensitive hashing* : focus on pairs of signatures likely to be similar.
-

Locality-Sensitive Hashing

General idea: Generate from the collection of all elements (signatures in our example) a small list of *candidate pairs*: pairs of elements whose similarity must be evaluated.

For signature matrices: Hash columns to many buckets, and make elements of the same bucket candidate pairs.

Candidate Generation From Minhash Signatures

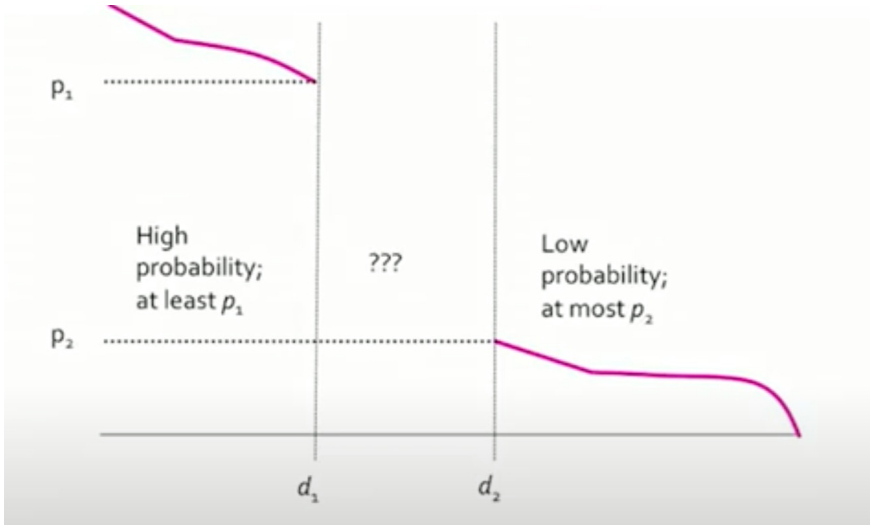
Pick a similarity threshold *t*, a fraction < 1. We want a pair of columns *c* and *d* of the signature matrix M to be a *candidate pair* if and only if their signatures agree in at least fraction *t* of the rows.

I.e., $M(i, c) = M(i, d)$ for at least fraction *t* values of *i*.

Assumptions for LSH

- Consider a space S of points with distance measure d
- Hash functions to be (d1,d2,p1,p2)-sensitive if for any two points x and y:
  - If $d(x,y) <= d_1$ then $h(x)=h(y) >= p_1$, $\forall h$
  - If $(d,y) \geq d_2$, then $h(x)=h(y) \leq p_2$, $\forall h$

- For Jaccard distance & minhash:
  - Prob[h(x)=h(y)] = 1 – d(x,y)



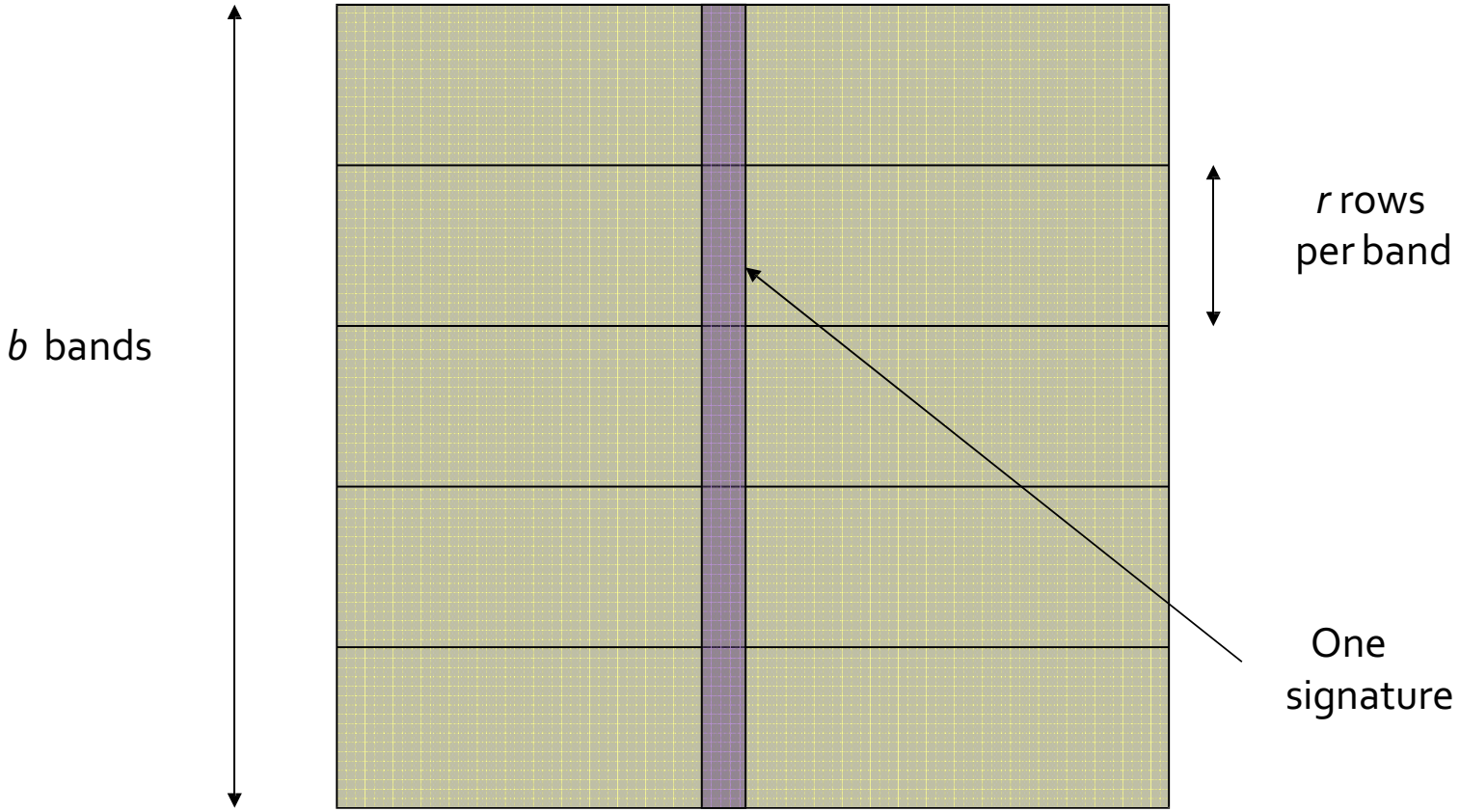
LSH for Minhash Signatures

Big idea: hash columns of signature matrix *M* several times.

Arrange that (only) similar columns are likely to hash to the same bucket.

Candidate pairs are those that hash at least once to the same bucket.

Partition Into Bands



Partition into Bands – (2)

Divide matrix *M* into *b* bands of *r* rows.

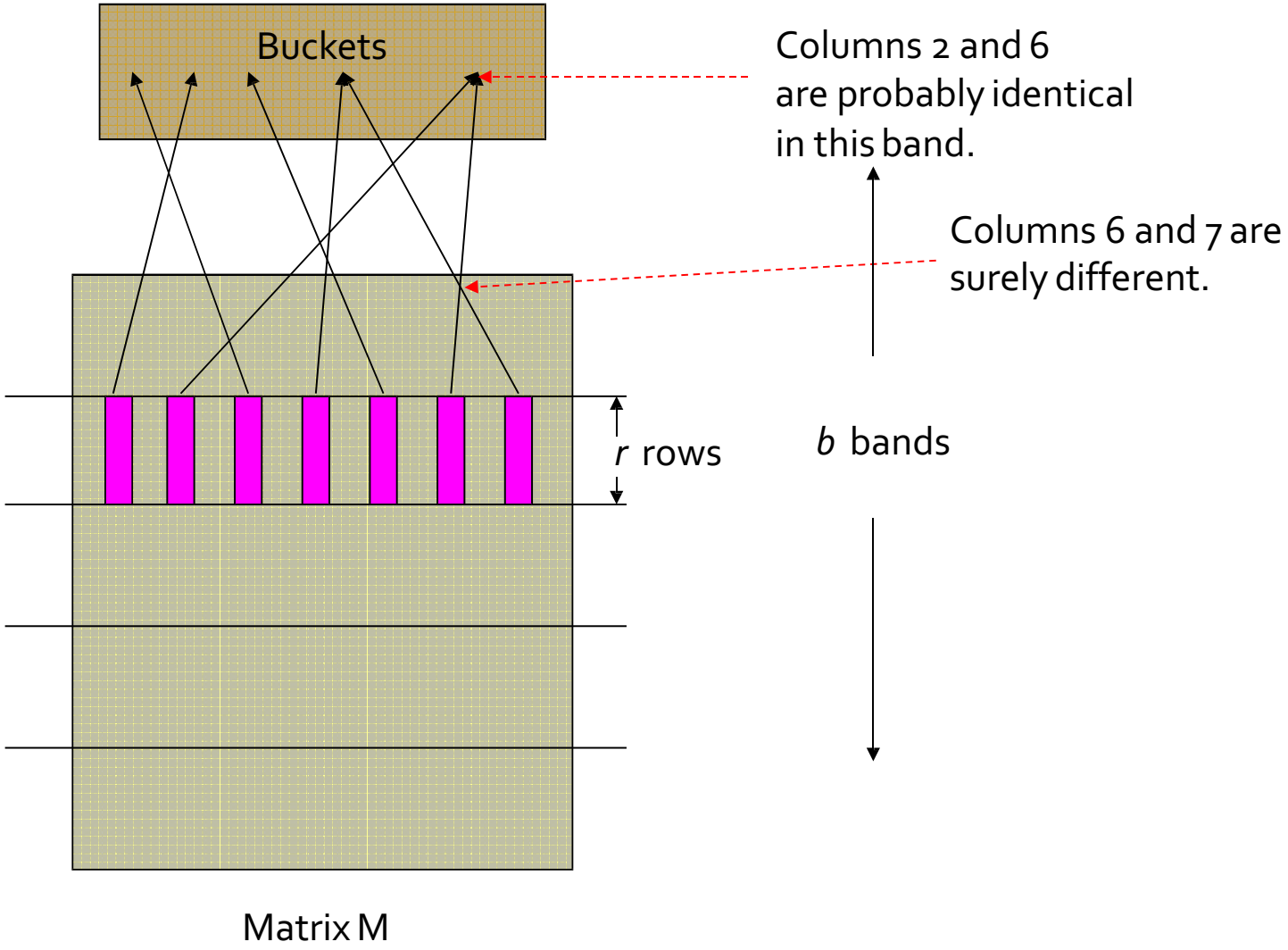
For each band, hash its portion of each column to a hash table with *k* buckets.

Make *k* as large as possible.

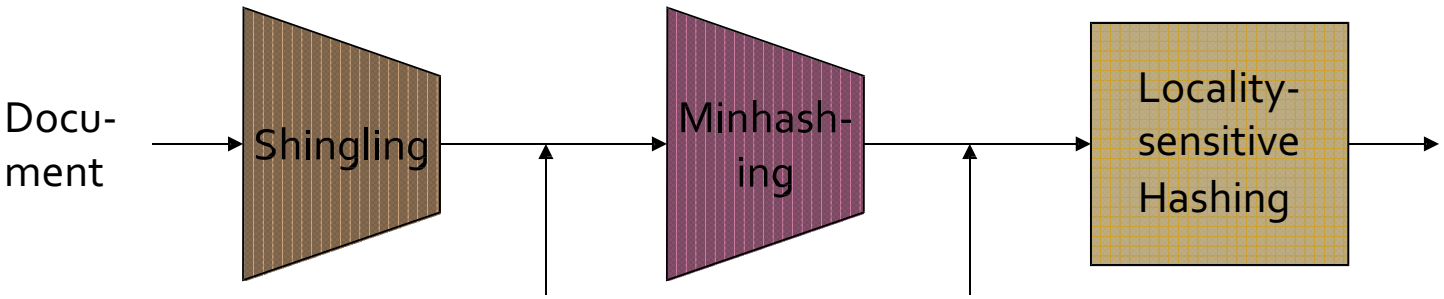
Candidate column pairs are those that hash to the same bucket for ≥ 1 band.

Tune *b* and *r* to catch most similar pairs, but few nonsimilar pairs.

Hash Function for One Bucket



The Big Picture



Candidate pairs : those pairs of signatures that we need to test for similarity.

The set of strings of length *k* that appear in the doc-ument

Signatures : short integer vectors that represent the sets, and reflect their similarity

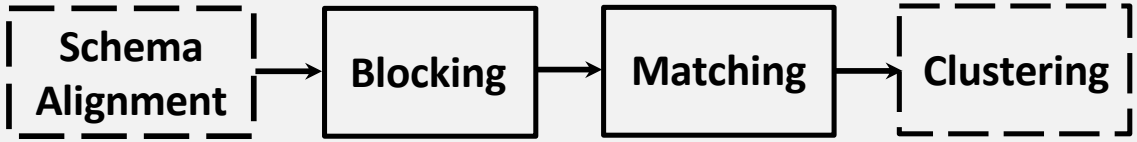
Methods for High Degree of Similarity

- LSH is good for relativelly low similarities
- Finding Identical Items
  - Hash on the first few letters
  - Hash the whole document
  - Hash fixed positions only
- Representing Sets as Strings
- Length based Filtering
  - Sort strings on length and compare only with what follows.
- Prefix Indexing

LSH idea in entity Linking: Blocking

1. Represent each entity by *one or more* signatures called blocking keys
   - Focus on string values
2. Place into blocks all entities having the *same or similar* blocking key
3. Two matching profiles can be detected as long as they co-occur in at least one block
   - Trade-off between recall and precision!

Modern Entity Linking

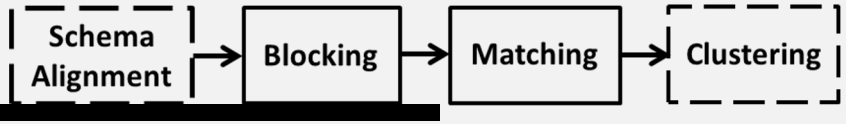


- Requires 4 steps

Step 1: Schema Alignment / Matching

- Scope: Remove heterogeneity in names
  - Record Linkage
- Goal:
  - Create mappings between equivalent attributes of the two schemata, e.g., *profession* ≡ *job*
- Types of Solutions: Schema Matching Methods
  - Structure-based
  - Instance-based
  - Usage-based
  - Hybrid

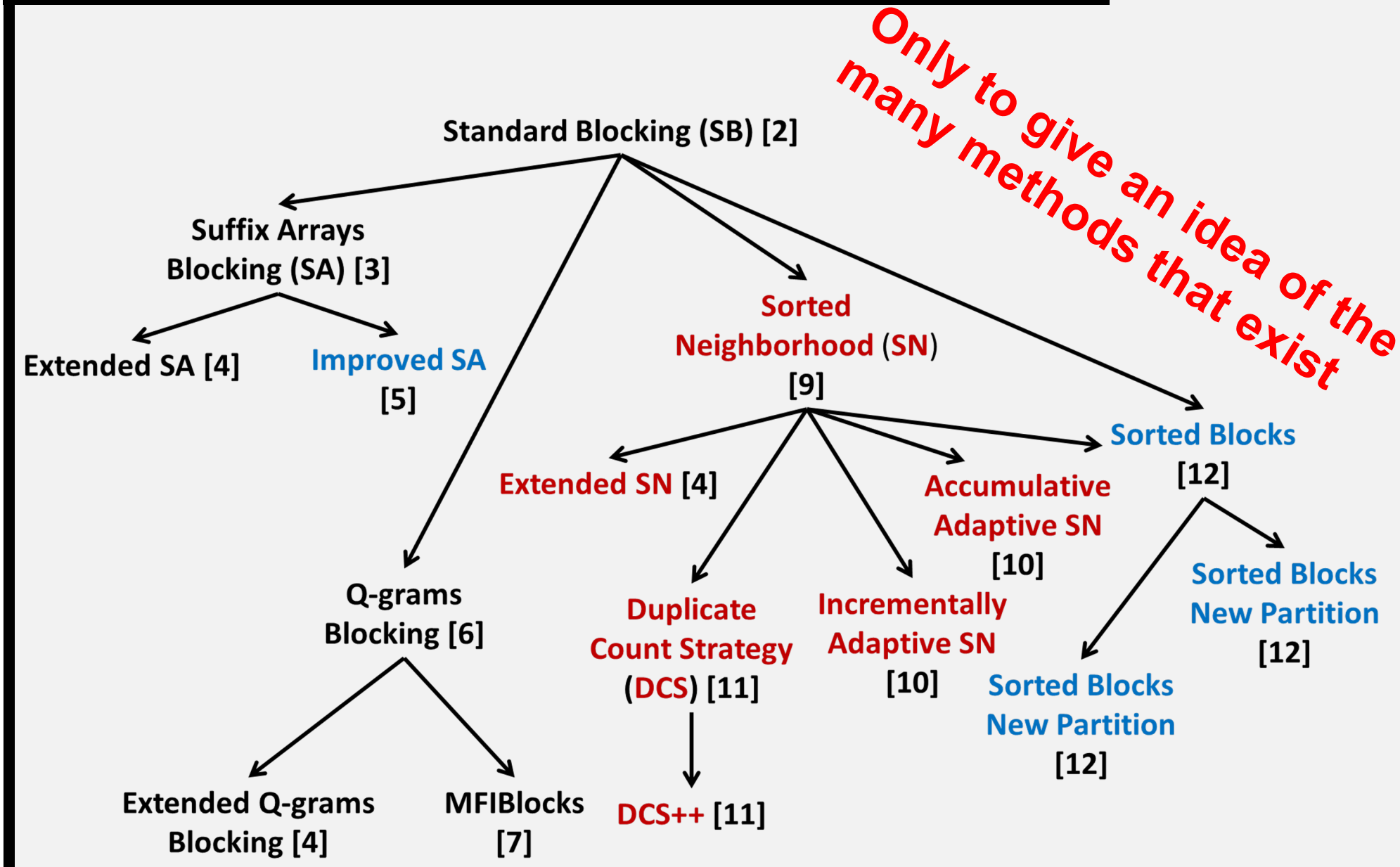
Step 2: Blocking



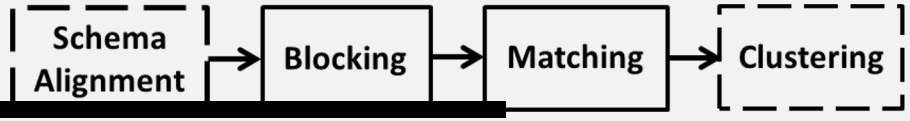
- Scope:
  - Both Deduplication and Record Linkage
- Goal:
  - ER is an inherently quadratic problem, $O(n^2)$: every entity has to be compared with all others
  - Blocking groups similar entities into blocks
    - Comparisons are executed only inside each block
    - Complexity is now quadratic to the size of the block (much smaller than dataset size!)

The Minhash/LSH Principle

Genealogy Tree of Non-learning Blocking Methods



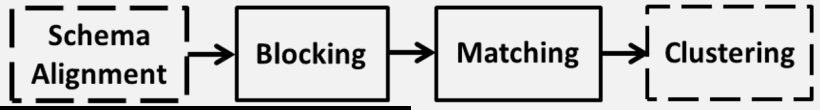
Step 3: Matching



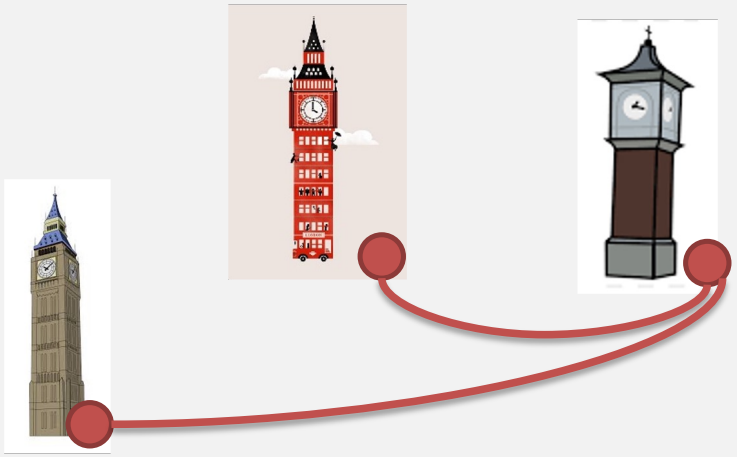
- Estimates the similarity of candidate matches.
- Input
  - A set of blocks
    - Every distinct comparison in any block is a candidate match
- Output
  - Similarity Graph
    - Nodes → entities
    - Edges → candidate matches
    - Edge weights → matching likelihood (based on similarity score)

We have already seen this

Step 4: Clustering



- Partitions the matched pairs into equivalence clusters
  i.e., groups of entity profiles describing the same real-world object



- Input
  - Similarity Graph:
    - Nodes → entities
    - Edges → candidate matches
    - Edge weights → matching likelihood (based on similarity score)

- Output
  - Equivalence Clusters

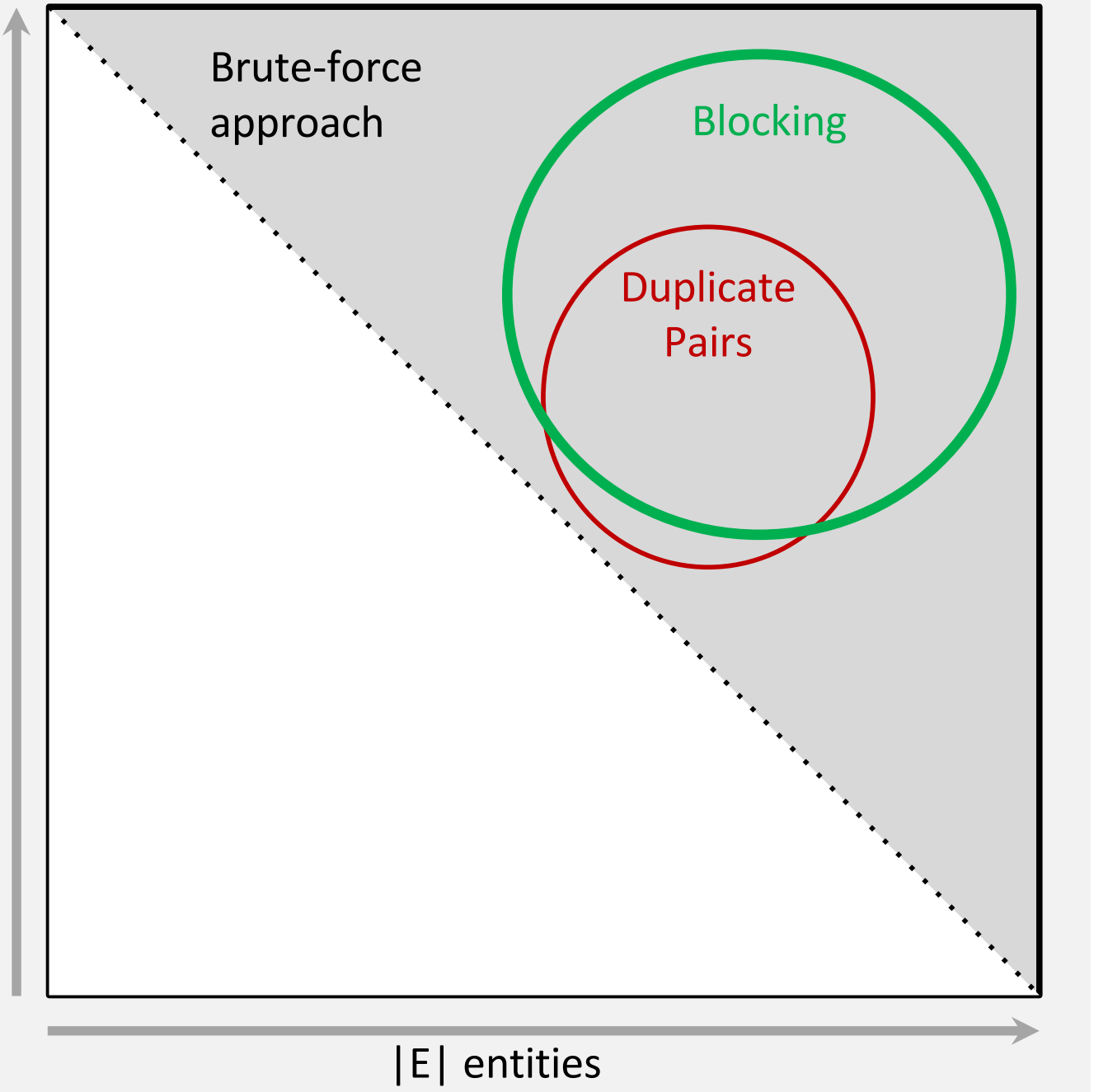
- THANK YOU

Computational cost

Input:
Entity Collection E

|E| entities

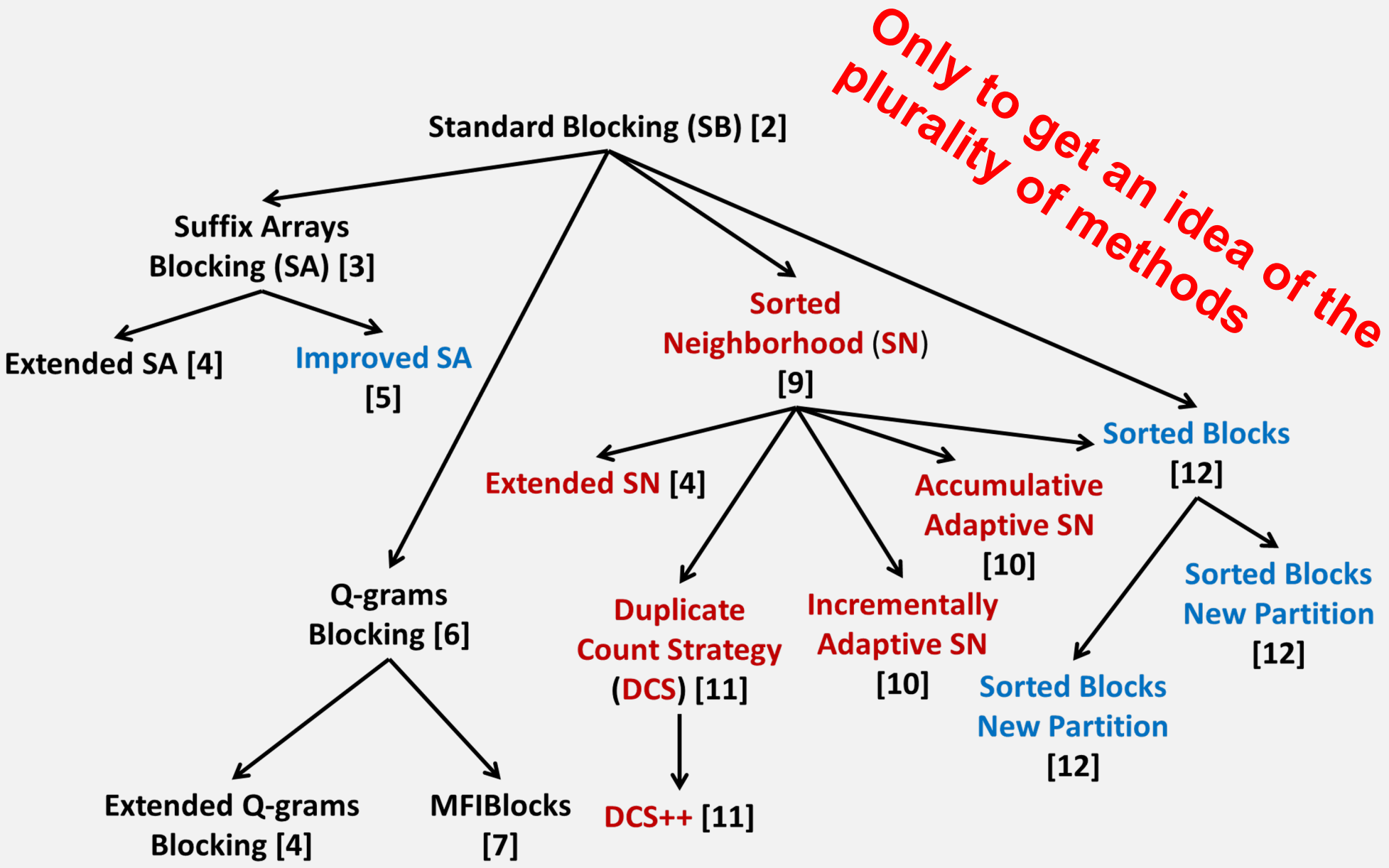
E.g.: For a dataset with 100,000 entities:
~10^{10} comparisons,
If 0.05 msec each →
>100 hours in total



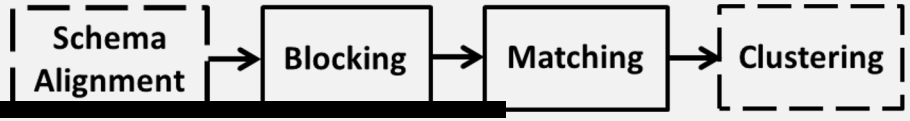
General Principles of Blocking

1. Represent each entity by *one or more* signatures called blocking keys
   - Focus on string values
2. Place into blocks all entities having the *same or similar* blocking key
3. Two matching profiles can be detected as long as they co-occur in at least one block
   - Trade-off between recall and precision!

Genealogy Tree of Non-learning Blocking Methods [1]



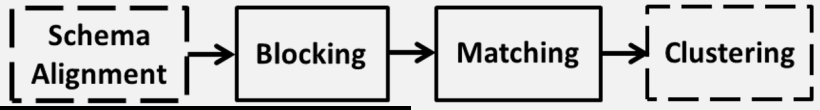
Step 3: Matching



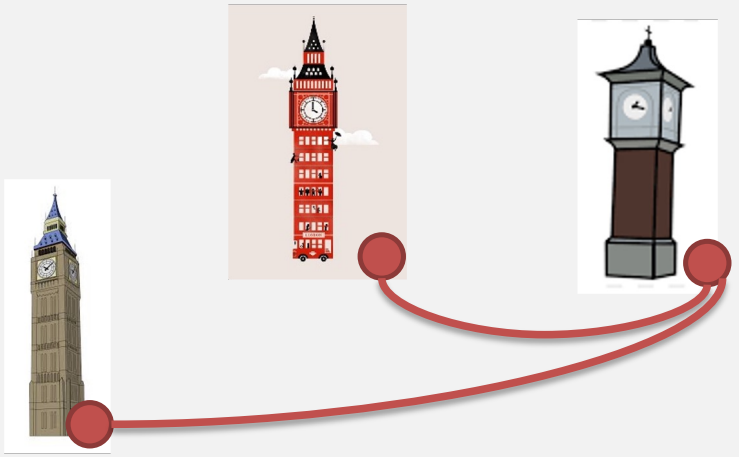
- Estimates the similarity of candidate matches.
- Input
  - A set of blocks
    - Every distinct comparison in any block is a candidate match
- Output
  - Similarity Graph
    - Nodes → entities
    - Edges → candidate matches
    - Edge weights → matching likelihood (based on similarity score)

We have already seen this

Step 4: Clustering



- Partitions the matched pairs into equivalence clusters
  i.e., groups of entity profiles describing the same real-world object

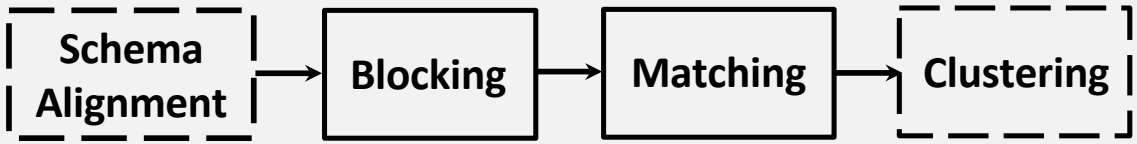


- Input
  - Similarity Graph:
    - Nodes → entities
    - Edges → candidate matches
    - Edge weights → matching likelihood (based on similarity score)

- Output
  - Equivalence Clusters

Part of the Data Mining Course

Generation 2: Tackling **Volume** and Veracity



- Same workflow as Generation 1
- Scope:
  - (tens of) millions of structured entity profiles
- Goals:
  - High accuracy despite noise
  - High time efficiency despite the size of data
- Assumptions:
  - Known schema → custom, schema-based solutions

Solution: Parallelization

Two types:

- Multi-core parallelization
  - Single system → shared memory
  - Distribute processing among available CPUs
- Massive parallelization
  - Cluster of independent systems
  - **Map-Reduce** paradigm [1]
    - Data partitioned across the nodes of a cluster
    - Fault-tolerant, optimized execution
    - Map Phase: transforms a data partition into (key, value) pairs
    - Reduce Phase: processes pairs with the same key

Any Questions?