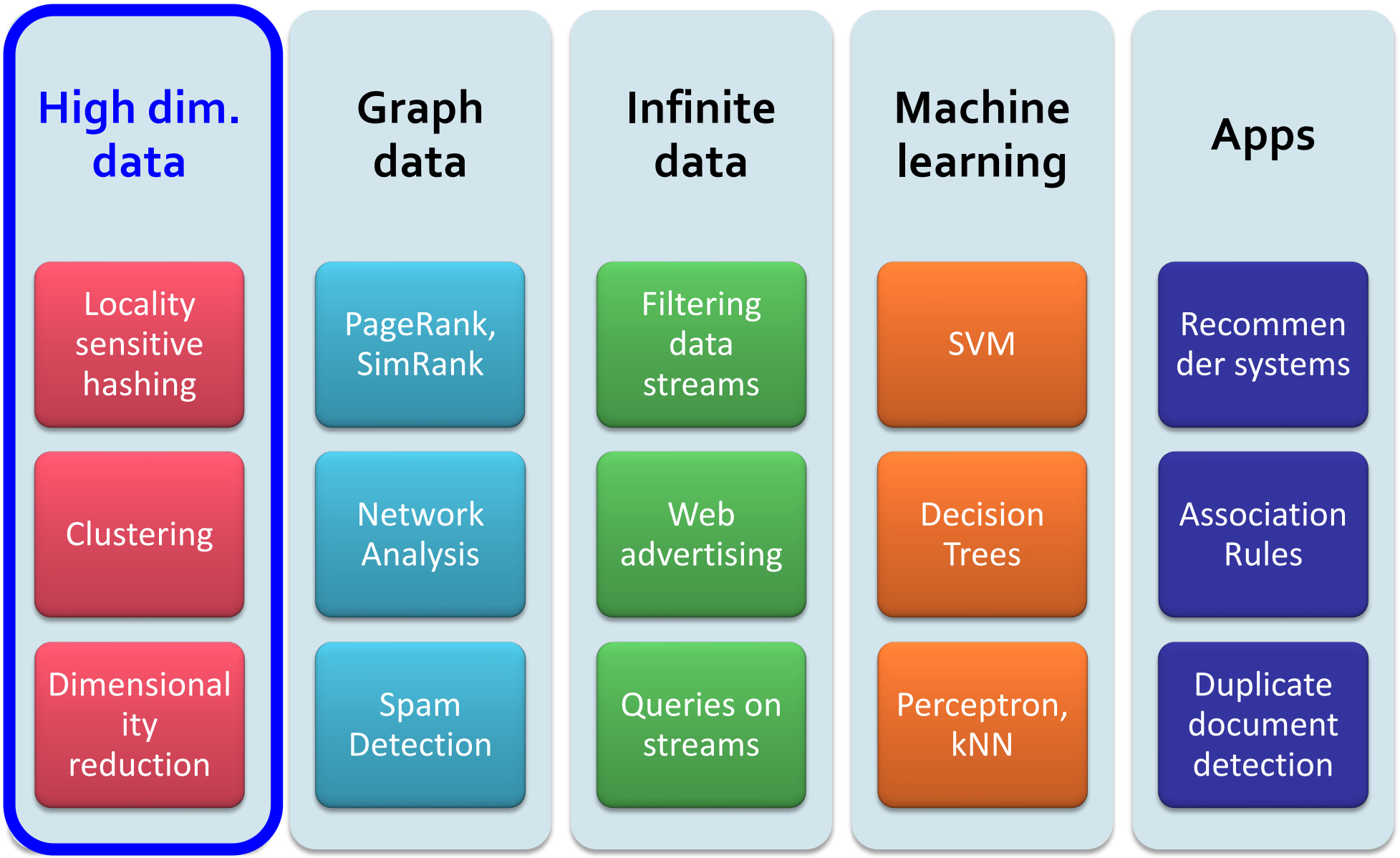
Finding Similar Items: Locality Sensitive Hashing

Mining of Massive Datasets
Jure Leskovec, Anand Rajaraman, Jeff Ullman
Stanford University
http://www.mmds.org

New thread: High dim. data

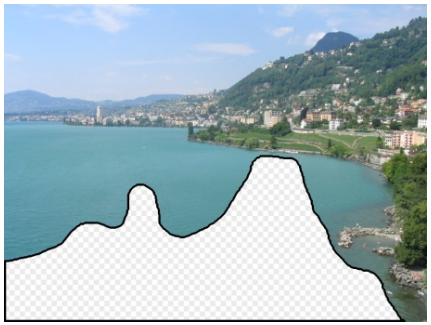


[Hays and Efros, SIGGRAPH 2007]

Scene Completion Problem

[Hays and Efros, SIGGRAPH 2007]

Scene Completion Problem



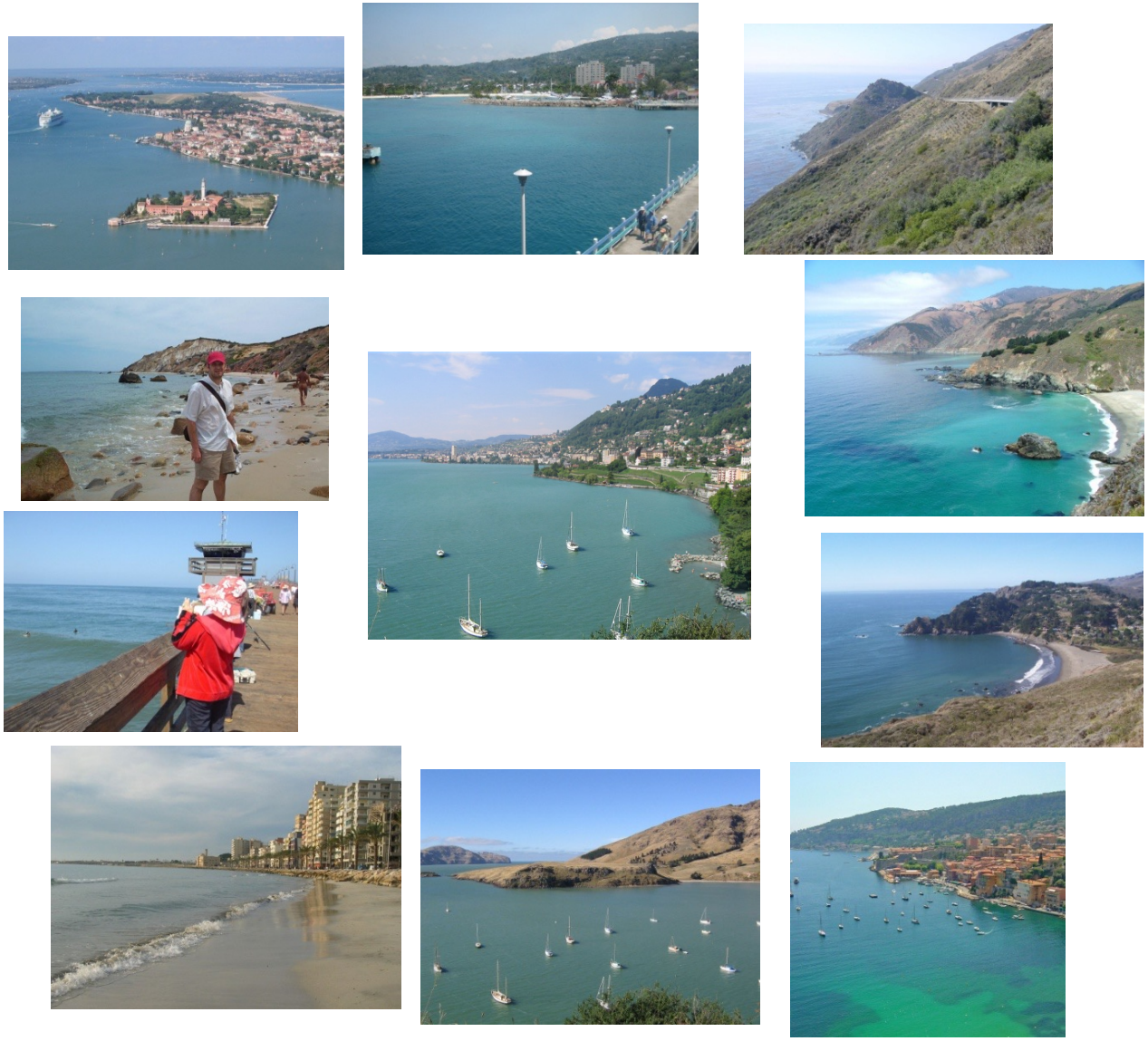
[Hays and Efros, SIGGRAPH 2007]

Scene Completion Problem

10 nearest neighbors from a collection of 20,000 images

[Hays and Efros, SIGGRAPH 2007]

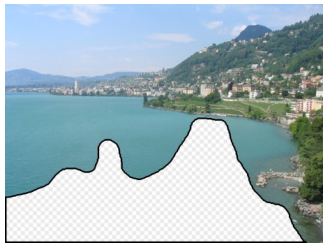
Scene Completion Problem



10 nearest neighbors from a collection of 2 million images

A Common Metaphor

- **Many problems can be expressed as finding "similar" sets:**
  - **Find near-neighbors in high-dimensional space**
- **Examples:**
  - **Pages with similar words**
    - For duplicate detection, classification by topic
  - **Customers who purchased similar products**
    - Products with similar customer sets
  - **Images with similar features**
    - Users who visited similar websites



Problem for Today's Lecture

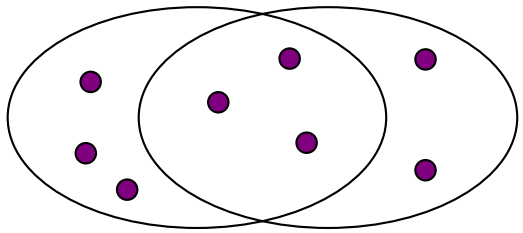
- **Given: High dimensional data points**
  - **For example:** Image is a long vector of pixel colors
- **And some distance function**
  - Which quantifies the "distance" between and
- **Goal:** Find **all pairs of data points** that are within some distance threshold
- **Note:** Naïve solution would take ☹

  where is the number of data points
- **MAGIC: This can be done in !! How?**

Finding Similar Items

Distance Measures

- **Goal: Find near-neighbors in high-dim. space**
  - We formally define "near neighbors" as points that are a "small distance" apart
- For each application, we first need to define what "**distance**" means
- **Today: Jaccard distance/similarity**
  - The **Jaccard similarity** of two **sets** is the size of their intersection divided by the size of their union: $sim(C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$
  - **Jaccard distance:** $d(C_1, C_2) = 1 - |C_1 \cap C_2| / |C_1 \cup C_2|$



3 in intersection
8 in union
Jaccard similarity= 3/8
Jaccard distance = 5/8

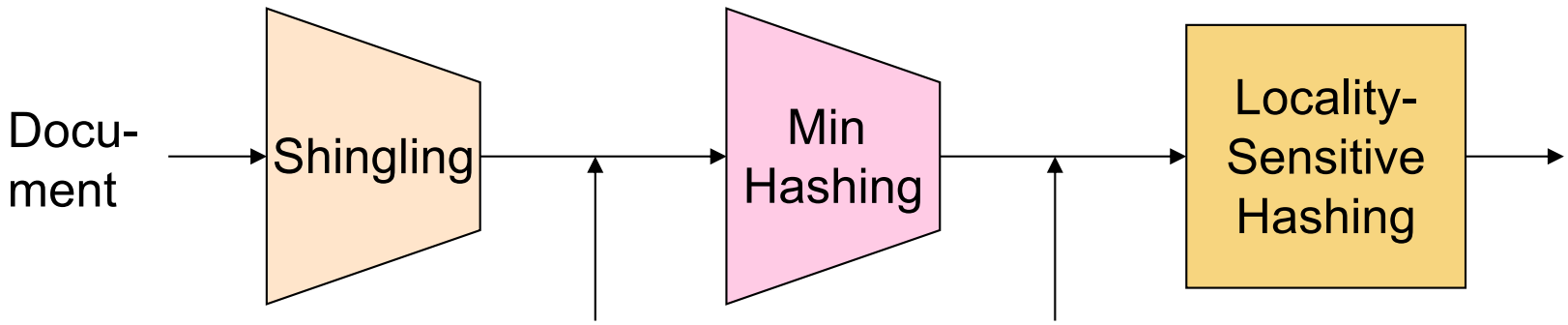
Task: Finding Similar Documents

- **Goal: Given a large number (N in the millions or billions) of documents, find "near duplicate" pairs**
- **Applications:**
  - Mirror websites, or approximate mirrors
    - Don't want to show both in search results
  - Similar news articles at many news sites
    - Cluster articles by "same story"
- **Problems:**
  - Many small pieces of one document can appear out of order in another
  - Too many documents to compare all pairs
  - Documents are so large or so many that they cannot fit in main memory

3 Essential Steps for Similar Docs

1. ***Shingling:*** Convert documents to sets
2. ***Min-Hashing:*** Convert large sets to short signatures, while preserving similarity
3. ***Locality-Sensitive Hashing:*** Focus on pairs of signatures likely to be from similar documents
   - **Candidate pairs!**

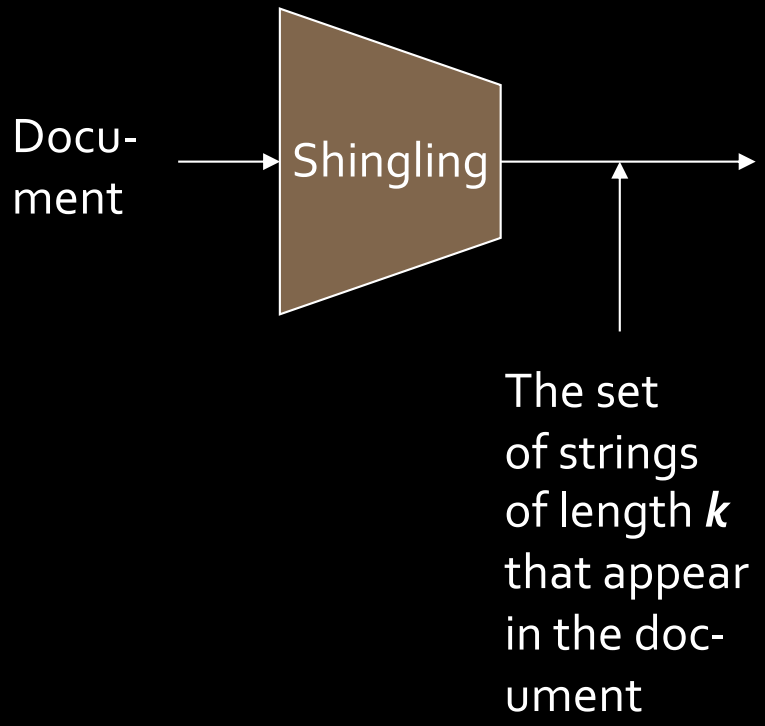
The Big Picture



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

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



Shingling

Step 1: ***Shingling:*** Convert documents to sets

Documents as High-Dim Data

- **Step 1: *Shingling:* Convert documents to sets**
- **Simple approaches:**
  - Document = set of words appearing in document
  - Document = set of "important" words
  - Don't work well for this application. Why?
- **Need to account for ordering of words!**
- A different way: **Shingles!**

Define: Shingles

- A *k*-shingle (or *k*-gram) for a document is a sequence of *k* tokens that appears in the doc
  - Tokens can be characters, words or something else, depending on the application
  - Assume tokens = characters for examples

- **Example: k=2**; document $\mathbf{D_1}$ = abcab
  Set of 2-shingles: $\mathbf{S(D_1)}$ = {ab, bc, ca}
  - **Option:** Shingles as a bag (multiset), count ab twice: $\mathbf{S'(D_1)}$ = {ab, bc, ca, ab}

Compressing Shingles

- To **compress long shingles**, we can **hash** them to (say) 4 bytes
- **Represent a document by the set of hash values of its *k*-shingles**
  - **Idea:** 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

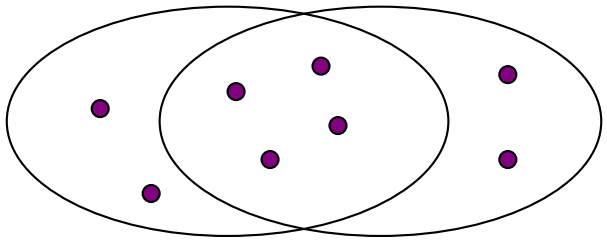
  Set of 2-shingles: $\mathbf{S(D_1)}$ = {ab, bc, ca}

  Hash the singles: $\mathbf{h(D_1)}$ = {1, 5, 7}

Similarity Metric for Shingles

- **Document D_1 is a set of its k-shingles $C_1=S(D_1)$**
- Equivalently, each document is a 0/1 vector in the space of *k*-shingles
  - Each unique shingle is a dimension
  - Vectors are very sparse
- **A natural similarity measure is the Jaccard similarity:**

$$sim(D_1, D_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$$



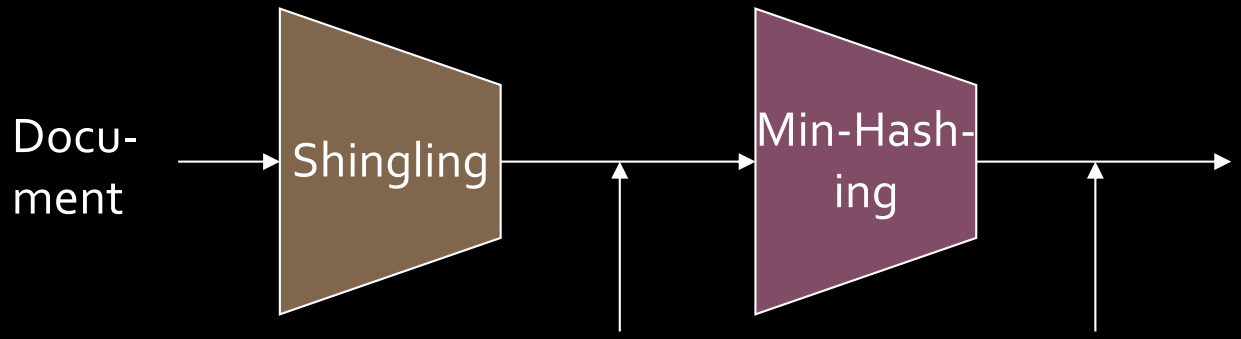
Working Assumption

- **Documents that have lots of shingles in common have similar text, even if the text appears in different order**

- **Caveat:** You must pick ***k*** large enough, or most documents will have most shingles
  - ***k*** = 5 is OK for short documents
  - ***k*** = 10 is better for long documents

Motivation for Minhash/LSH

- **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…



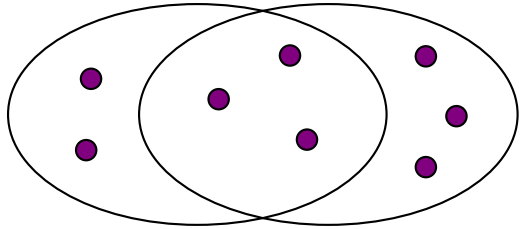
The set of strings of length *k* that appear in the doc-ument

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

MinHashing

Step 2: *Minhashing:* Convert **large sets** to **short signatures**, while **preserving similarity**

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**

From Sets to Boolean Matrices

- **Rows** = elements (shingles)
- **Columns** = sets (documents)
  - 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 corresponding sets (rows with value *1)*
  - **Typical matrix is sparse!**
- **Each document is a column:**
  - **Example: sim(C_1 ,C_2) = ?**
    - Size of intersection = 3; size of union = 6, Jaccard similarity (not distance) = 3/6
    - **d(C_1,C_2) = 1 – (Jaccard similarity) = 3/6**

Documents

Shingles				
	1	1	1	0
	1	1	0	1
	0	1	0	1
	0	0	0	1
	1	0	0	1
	1	1	1	0
	1	0	1	0

Outline: Finding Similar Columns

- **So far:**
  - Documents $\rightarrow$ Sets of shingles
  - Represent sets as boolean vectors in a matrix
- **Next goal: Find similar columns while computing small signatures**
  - **Similarity of columns == similarity of signatures**

Outline: Finding Similar Columns

- **Next Goal: Find similar columns, Small signatures**
- **Naïve approach:**
  - **1) Signatures of columns:** small summaries of columns
  - **2) Examine pairs of signatures** to find similar columns
    - **Essential:** Similarities of signatures and columns are related
  - **3) Optional:** Check that columns with similar signatures are really similar
- **Warnings:**
  - Comparing all pairs may take too much time: **Job for LSH**
    - These methods can produce false negatives, and even false positives (if the optional check is not made)

Hashing Columns (Signatures)

- **Key idea:** "hash" each column ***C*** to a small *signature* ***h(C)***, such that:
  - **(1)** ***h(C)*** is small enough that the signature fits in RAM
  - **(2)** ***sim(C_1, C_2)*** is the same as the "similarity" of signatures ***h(C_1)*** and ***h(C_2)***

- **Goal: Find a hash function *h(·)* such that:**
  - If ***sim(C_1,C_2)*** is high, then with high prob. ***h(C_1) = h(C_2)***
  - If ***sim(C_1,C_2)*** is low, then with high prob. ***h(C_1) ≠ h(C_2)***

- **Hash docs into buckets. Expect that "most" pairs of near duplicate docs hash into the same bucket!**

Min-Hashing

- **Goal: Find a hash function $h(\cdot)$ such that:**
  - if $sim(C_1,C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$
  - if $sim(C_1,C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$
- **Clearly, the hash function depends on the similarity metric:**
  - Not all similarity metrics have a suitable hash function
- **There is a suitable hash function for the Jaccard similarity:** It is called **Min-Hashing**

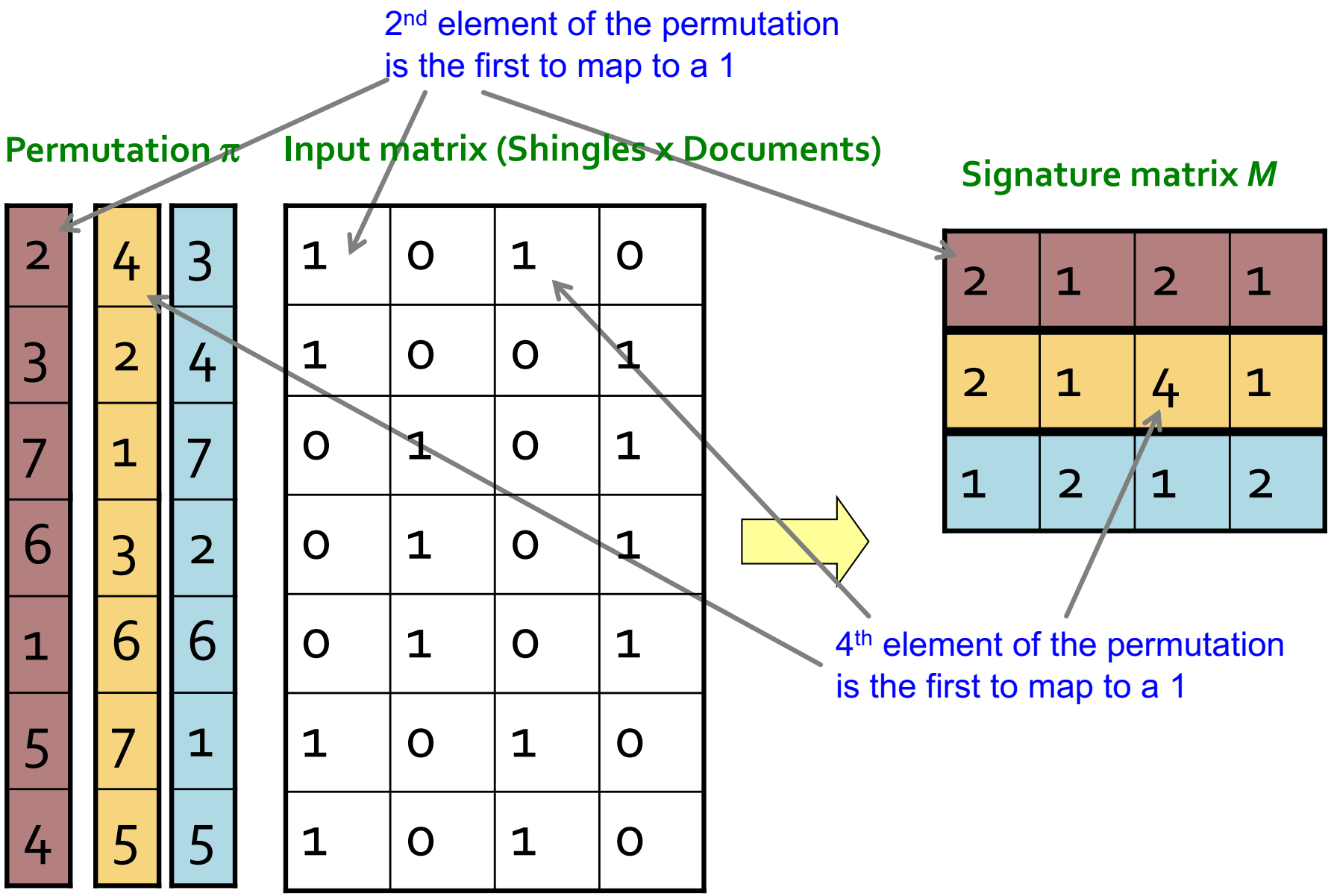
Min-Hashing

- Imagine the rows of the boolean matrix permuted under **random permutation** π
- Define a **"hash" function** $h_\pi(C)$ = the index of the **first** (in the permuted order π) row in which column ***C*** has value **1**:

$$h_\pi(C) = min_\pi \pi(C)$$

- Use several (e.g., 100) independent hash functions (that is, permutations) to create a signature of a column

Min-Hashing Example



The Min-Hash Property

0	0
0	0
1	**1**
0	0
0	1
1	0

- **Choose a random permutation π**
- **Claim:** $\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- **Why?**
  - Let **X** be a doc (set of shingles), $y \in X$ is a shingle
  - **Then:** $\Pr[\pi(y) = \min(\pi(X))] = 1/|X|$
    - It is equally likely that any $y \in X$ is mapped to the ***min*** element
  - Let ***y*** be s.t. $\pi(y) = \min(\pi(C_1 \cup C_2))$
  - **Then either:** $\pi(y) = \min(\pi(C_1))$ if $y \in C_1$, **or**
    $\pi(y) = \min(\pi(C_2))$ if $y \in C_2$

    One of the two cols had to have 1 at position ***y***
  - So the prob. that **both** are true is the prob. $y \in C_1 \cap C_2$
  - $\Pr[\min(\pi(C_1))=\min(\pi(C_2))] = |C_1 \cap C_2| / |C_1 \cup C_2| = sim(C_1, C_2)$

Similarity for Signatures

- We know: $\mathbf{Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)}$
- Now generalize to multiple hash functions

- **The *similarity of two signatures* is the fraction of the hash functions in which they agree**

- **Note:** Because of the Min-Hash property, the similarity of columns is the same as the expected similarity of their signatures

Min-Hashing Example

Permutation π

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

Input matrix (Shingles x Documents)

1	0	1	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0

Signature matrix *M*

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

Similarities:

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

Min-Hash Signatures

- **Pick K=100 random permutations of the rows**
- Think of ***sig*(C)** as a column vector
- ***sig*(C)[i] =** according to the *i*-th permutation, the index of the first row that has a 1 in column *C*

$$sig(C)[i] = \min(\pi_i(C))$$

- **Note:** The sketch (signature) of document *C* is small **~100 bytes!**

- **We achieved our goal! We "compressed" long bit vectors into short signatures**

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 $sig(C)[i] \leftarrow k_i(j)$

How to pick a random hash function h(x)?
Universal hashing:
$h_{a,b}(x)=((a \cdot x+b) \bmod p) \bmod N$
where:
a,b … random integers
p … prime number (p > N)

	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

$h(x) = x \bmod 5$

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

Row	C_1	C_2
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

$h(x) = x \bmod 5$

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

Row	C_1	C_2
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

$h(x) = x \bmod 5$

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

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
$h(4) = 4$	1	2
$g(4) = 4$	2	0

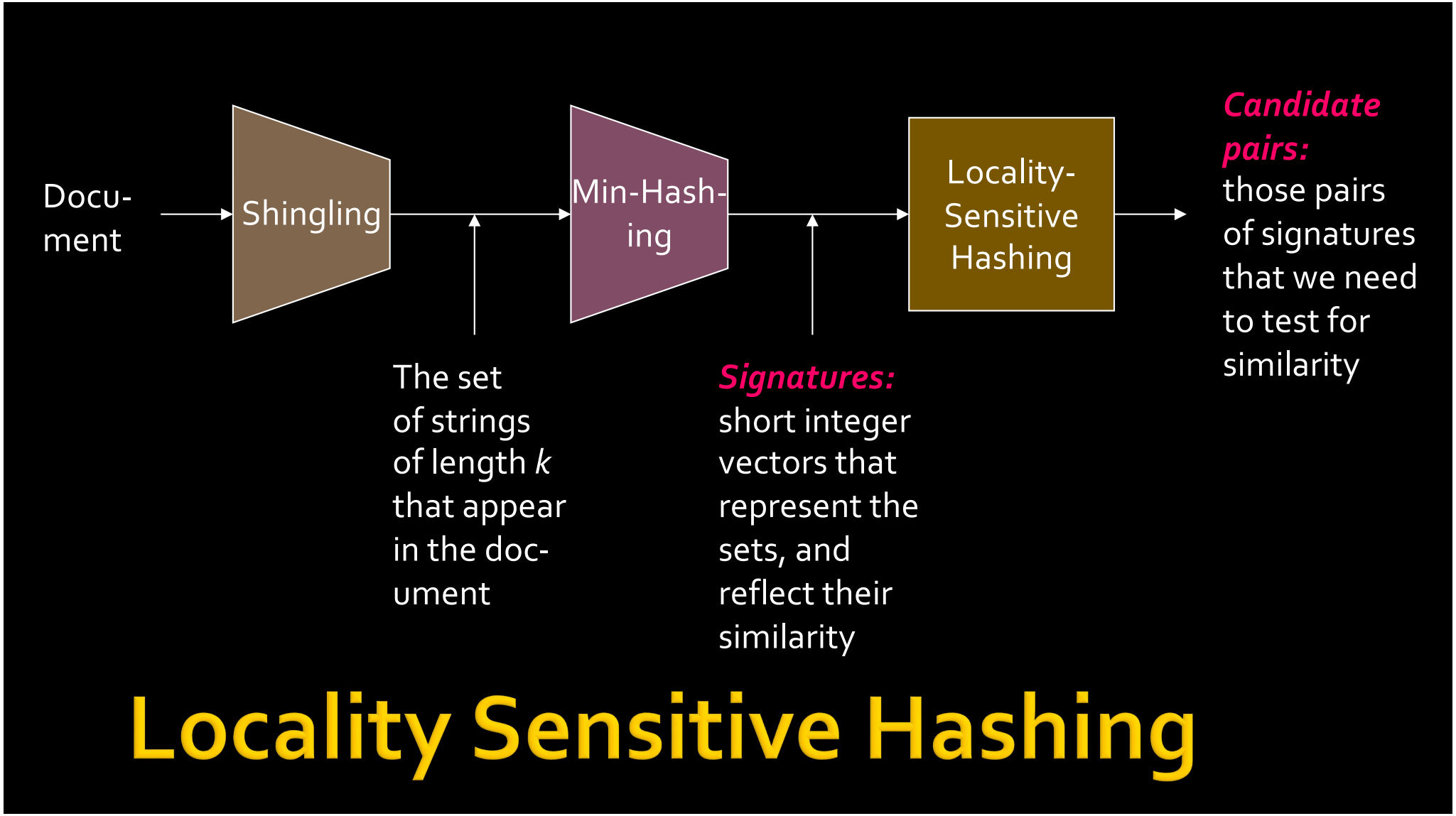
$h(x) = x \bmod 5$

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

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

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

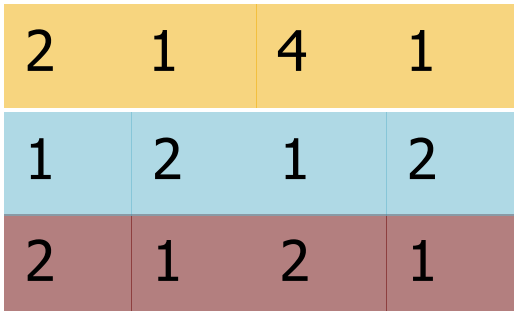
	Sig1	Sig2
$h(1) = 1$	1	∞
$g(1) = 3$	3	∞
$h(2) = 2$	1	2
$g(2) = 0$	3	0
$h(3) = 3$	1	2
$g(3) = 2$	2	0
$h(4) = 4$	1	2
$g(4) = 4$	2	0
$h(5) = 0$	1	0
$g(5) = 1$	2	0



Step 3: ***Locality-Sensitive Hashing:***

Focus on pairs of signatures likely to be from similar documents

LSH: First Cut



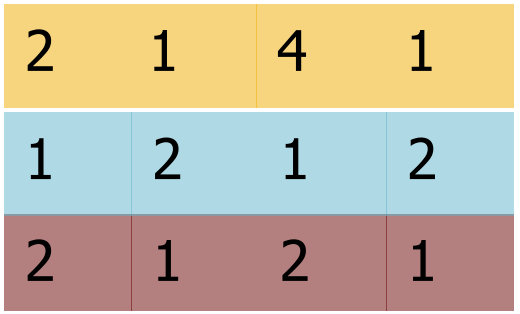
- **Goal:** Find documents with Jaccard similarity at least ***s*** (for some similarity threshold, e.g., ***s***=0.8)
- **LSH – General idea:** Use a function ***f(x,y)*** that tells whether ***x*** and ***y*** is a ***candidate pair:*** a pair of elements whose similarity must be evaluated
- **For Min-Hash matrices:**
  - Hash columns of signature matrix ***M*** to many buckets
  - Each pair of documents that hashes into the same bucket is a **candidate pair**

Candidates from Min-Hash

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

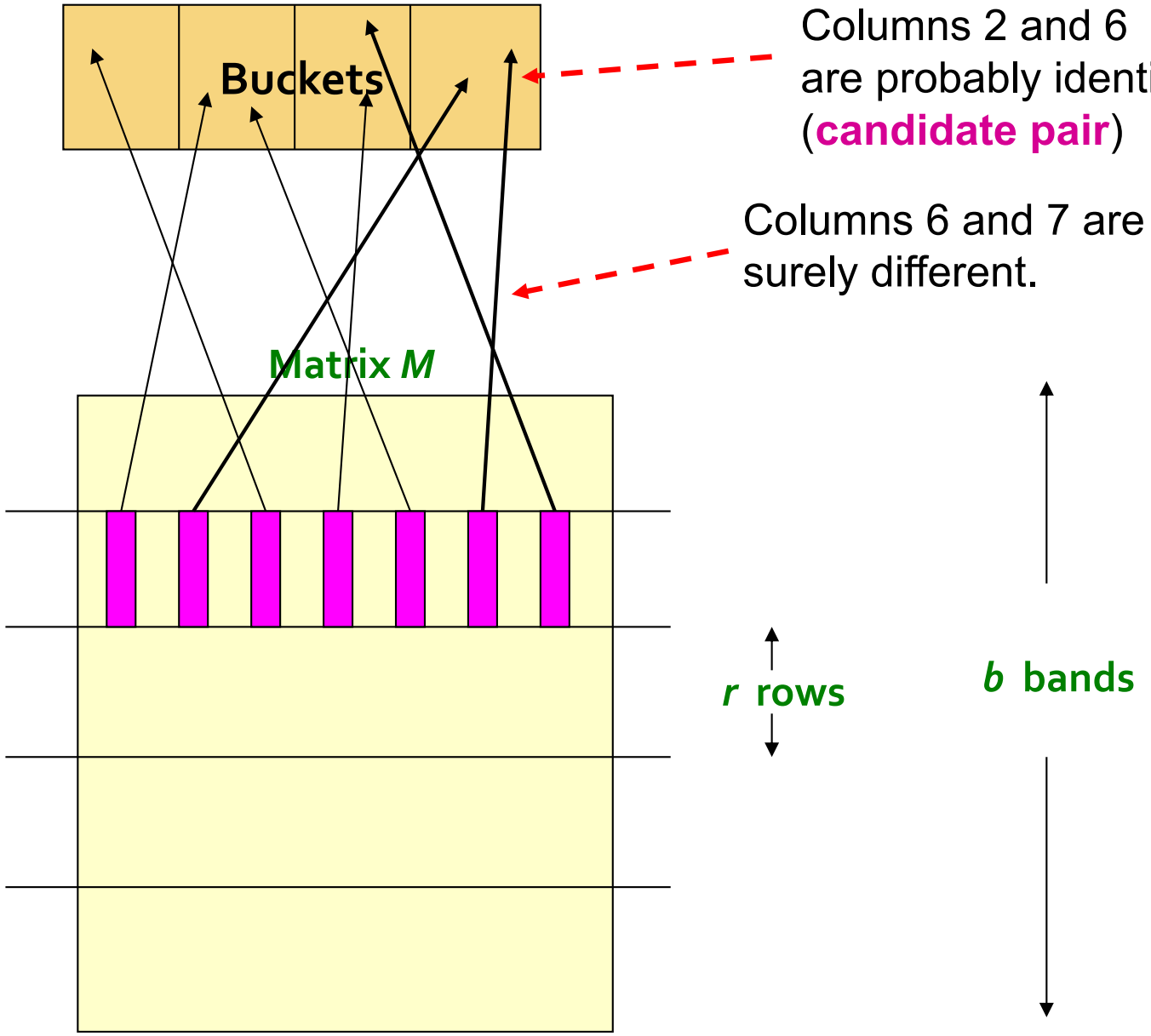
- **Pick a similarity threshold *s* (0 < s < 1)**
- Columns ***x*** and ***y*** of ***M*** are a **candidate pair** if their signatures agree on at least fraction ***s*** of their rows:
  M* (*i*, *x*) = *M* (*i*, *y*)** for at least frac. ***s values of ***i***
  - We expect documents ***x*** and ***y*** to have the same (Jaccard) similarity as their signatures

LSH for Min-Hash



- **Big idea: Hash columns of signature matrix *M* several times**
- Arrange that (only) **similar columns** are likely to **hash to the same bucket**, with high probability
- **Candidate pairs are those that hash to the same bucket**

Hashing Bands



Summary: 3 Steps

- **Shingling:** Convert documents to sets
  - We used hashing to assign each shingle an ID
- **Min-Hashing:** Convert large sets to short signatures, while preserving similarity
  - We used **similarity preserving hashing** to generate signatures with property $\mathbf{Pr}[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
  - We used hashing to get around generating random permutations
- **Locality-Sensitive Hashing:** Focus on pairs of signatures likely to be from similar documents
  - We used hashing to find **candidate pairs** of similarity $\geq$ **s**

Applications of LSH

Entity Resolution
Fingerprints
Similar News Articles

Mining of Massive Datasets
Leskovec, Rajaraman, and Ullman
Stanford University

Entity Resolution

- The *entity–resolution* problem is to examine a collection of records and determine which refer to the same entity.
  - *Entities* could be people, events, etc.
- Typically, we want to merge records if their values in corresponding fields are similar.

Matching Customer Records

- Imagine a consulting job solving the following problem:
  - Company A agreed to solicit customers for Company B, for a fee.
  - They then argued over how many customers.
  - Neither recorded exactly which customers were involved.

Customer Records – (2)

- Each company had about 1 million records describing customers that might have been sent from A to B.
- Records had name, address, and phone, but for various reasons, they could be different for the same person.

Customer Records – (3)

- Step 1: Design a measure ("*score* ") of how similar records are:
  - E.g., deduct points for small misspellings ("Jeffrey" vs. "Jeffery") or same phone with different area code.
- Step 2: Score all pairs of records that the LSH scheme identified as candidates; report high scores as matches.

Customer Records – (4)

- Problem: (1 million)^2 is too many pairs of records to score.
- Solution: A simple LSH.
  - Three hash functions: exact values of name, address, phone.
    - Compare iff records are identical in at least one.
  - Misses similar records with a small differences in all three fields.

Aside: Hashing Names, Etc.

- How do we hash strings such as names so there is one bucket for each string?
- Answer: Sort the strings instead.
- Another option was to use a few million buckets, and deal with buckets that contain several different strings.

Aside: Validation of Results

- We were able to tell what values of the scoring function were reliable in an interesting way.
- Identical records had a creation date difference of 10 days.
- We only looked for records created within 90 days of each other, so bogus matches had a 45–day average.

Validation – Generalized

- Any field not used in the LSH could have been used to validate, provided corresponding values were closer for true matches than false.
- Example: if records had a height field, we would expect true matches to be close, false matches to have the average difference for random people.

Fingerprint Matching

Minutiae
A New Way of Bucketing

Mining of Massive Datasets
Leskovec, Rajaraman, and Ullman
Stanford University

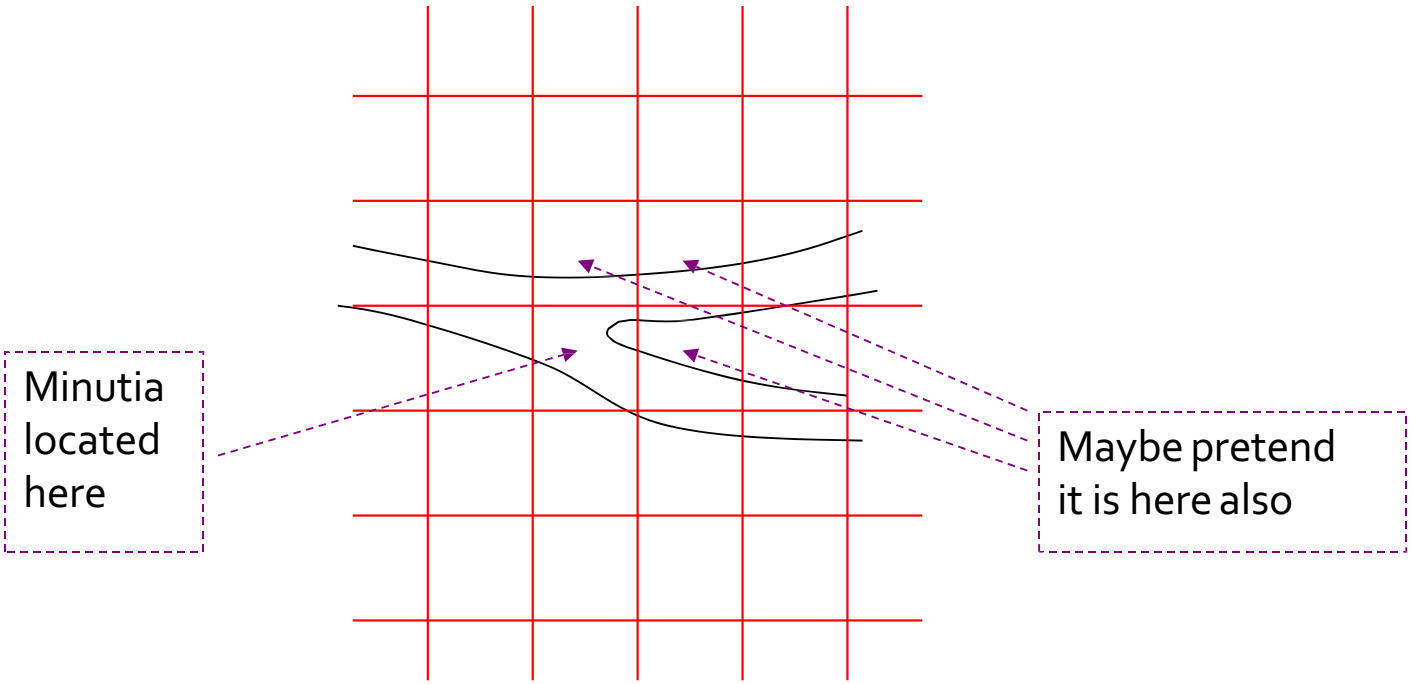
Fingerprint Comparison

- Represent a fingerprint by the set of positions of *minutiae*.
  - These are features of a fingerprint, e.g., points where two ridges come together or a ridge ends.

LSH for Fingerprints

- Place a grid on a fingerprint.
  - Normalize so identical prints will overlap.
- Set of grid squares where minutiae are located represents the fingerprint.
- Possibly, treat minutiae near a grid boundary as if also present in adjacent grid points.

Discretizing Minutiae



Applying LSH to Fingerprints

- Fingerprint = set of grid squares.
- No need to minhash, since the number of grid squares is not too large.
- Represent each fingerprint by a bit–vector with one position for each square.
  - 1 in only those positions whose squares have minutiae.

LSH/Fingerprints – (2)

- Pick 1024 (?) sets of 3 (?) grid squares (components of the bit vectors), randomly.
- For each set of three squares, two prints that each have 1 for all three squares are candidate pairs.
- Funny sort of ‘bucketization.”
  - Each set of three squares creates one bucket.
  - Prints can be in many buckets.

Example: LSH/Fingerprints

- Suppose typical fingerprints have minutiae in 20% of the grid squares.
- Suppose fingerprints from the same finger agree in at least 80% of their squares.
- Probability two random fingerprints each have minutiae in all three squares = $(0.2)^6$ = .000064.

Example: Continued

First print has has minutia in this square

Second print of the same finger also has minutia in that square

- Probability two fingerprints from the same finger each have 1's in three given squares = $((0.2)(0.8))^3 = .004096$.
- Prob. for at least one of 1024 sets of three points = $1-(1-.004096)^{1024} = .985$.
- But for random fingerprints: $1-(1-.000064)^{1024} = .063$.

1.5% false negatives

6.3% false positives

Thank you