

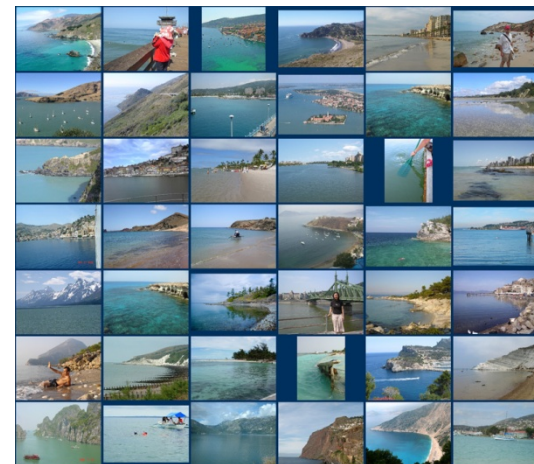
# Finding Similar Items: Locality Sensitive Hashing

Ref: Mining of Massive Datasets

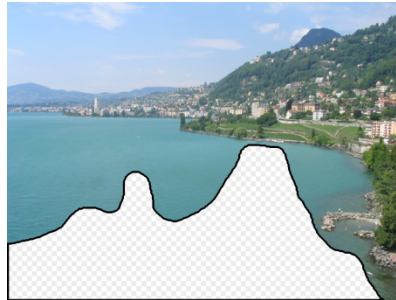
Jure Leskovec, Anand Rajaraman, Jeff Ullman

Stanford University

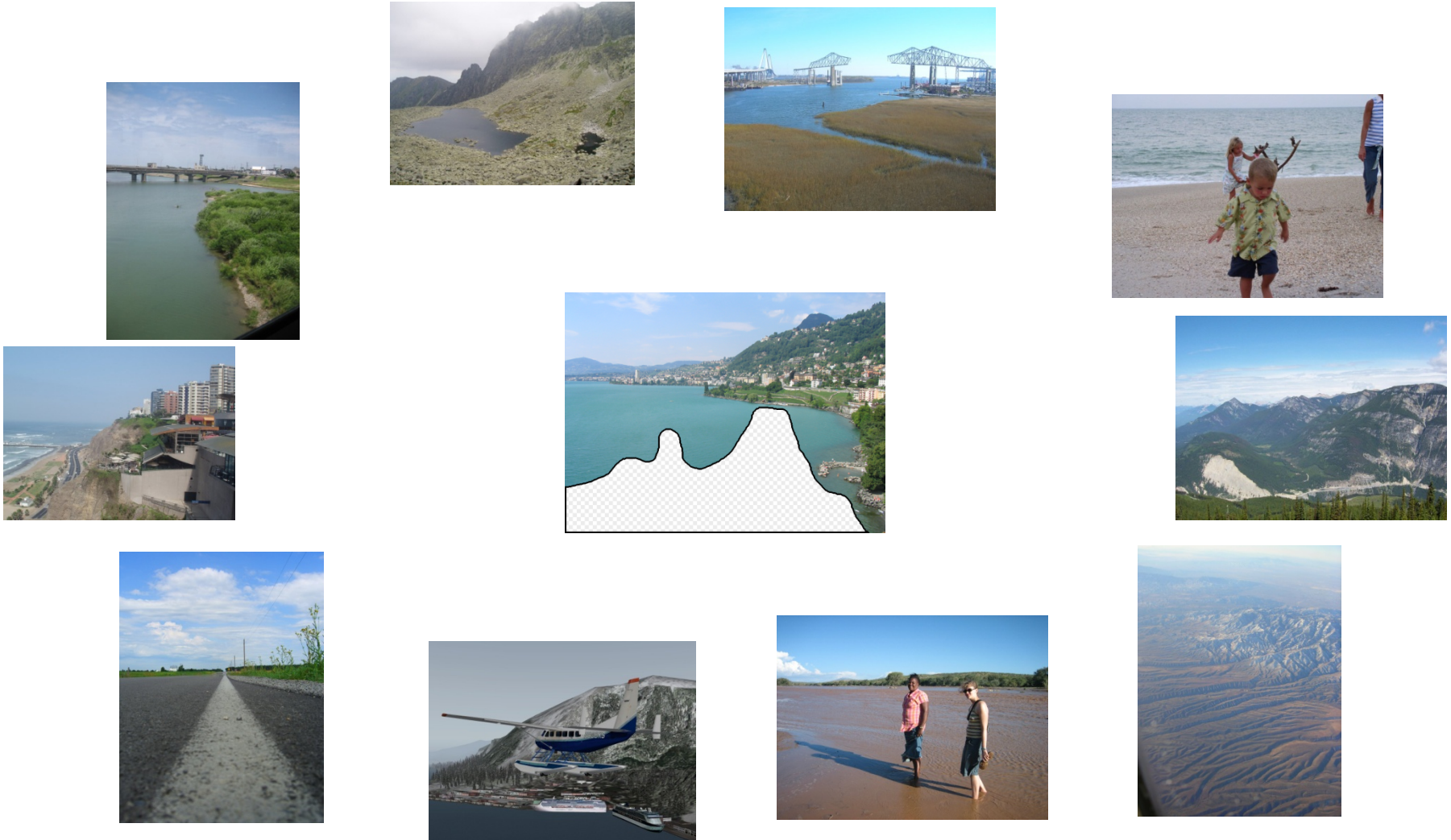
# Scene Completion Problem



# Scene Completion Problem



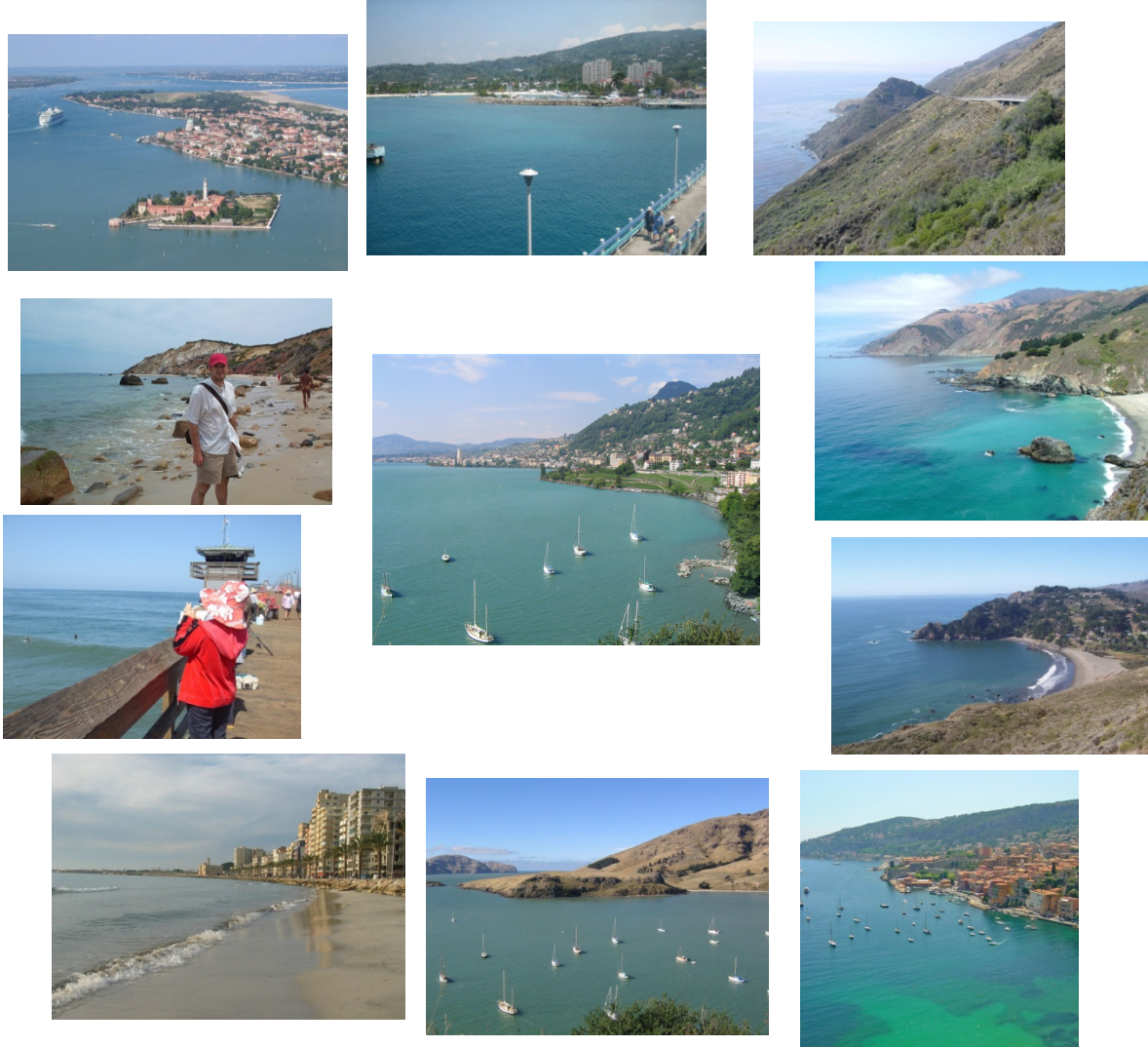
# Scene Completion Problem



**10 nearest neighbors from a collection of 20,000 images**



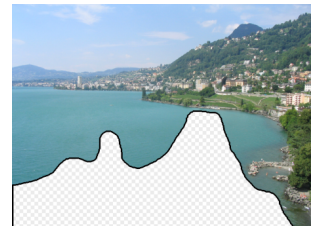
# Scene Completion Problem



**10 nearest neighbors from a collection of 2 million images**

# A Common Metaphor

- Many problems can be expressed as finding “similar” items:
  - Find near-neighbors in high-dimensional space
- **Examples:**
  - Pages with similar words
    - For duplicate detection, classification by topic
  - Customers who purchased similar products
  - Images with similar features



# Nearest Neighbor Search Problem

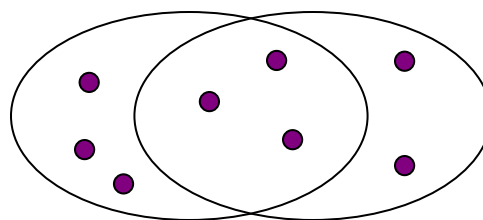
- **Given: High dimensional data points  $x_1, x_2, \dots$** 
  - **For example:** Image is a long vector of pixel colors
$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} \rightarrow [1 \ 2 \ 1 \ 0 \ 2 \ 1 \ 0 \ 1 \ 0]$$
- **And some distance function  $d(x_1, x_2)$** 
  - Which quantifies the “distance” between  $x_1$  and  $x_2$
- **Goal:** Find **all pairs of data points  $(x_i, x_j)$**  that are within some distance threshold  $d(x_i, x_j) \leq s$
- **Note:** Naïve solution would take  $O(N^2)$  ☹  
where  $N$  is the number of data points
- **MAGIC: This can be done in  $O(N)$ !! 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
- **Our focus: Jaccard distance/similarity**
  - The **Jaccard similarity** of two **sets** is the size of their intersection divided by the size of their union:  
$$\text{sim}(C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$$
  - **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 =  $5/8$

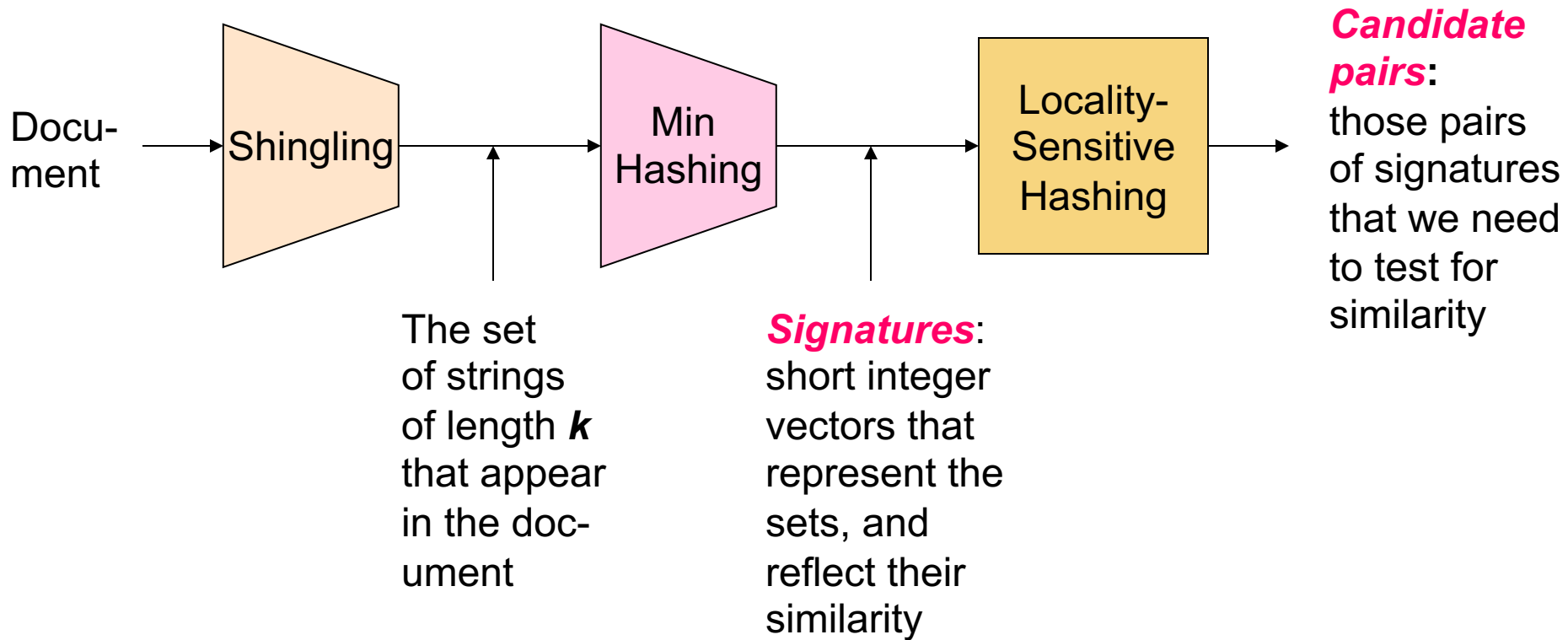
# Task: Finding Similar Documents

- **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

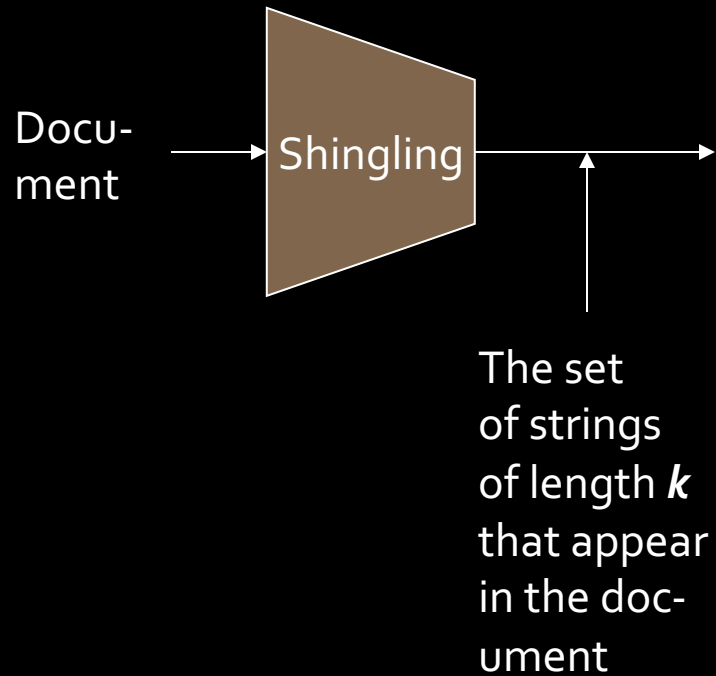
# 3 Essential Steps for Similar Docs

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







# 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  $D_1 = \text{ab cab}$   
Set of 2-shingles:  $S(D_1) = \{\text{ab}, \text{bc}, \text{ca}\}$ 
  - **Option:** Shingles as a bag (multiset), count ab twice:  $S'(D_1) = \{\text{ab}, \text{bc}, \text{ca}, \text{ab}\}$

# Compressing Shingles

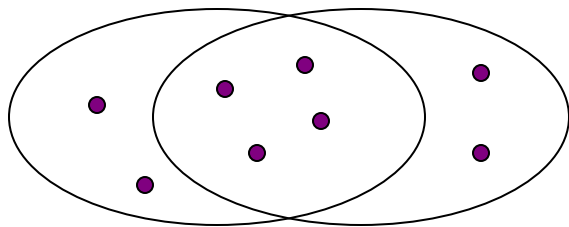
- To **compress long shingles**, we can **hash** them to (say) 4 bytes
- **Represent a document by the set of hash values of its  $k$ -shingles**
- **Example:**  $k=2$ ; document  $D_1 = \text{abcab}$   
Set of 2-shingles:  $S(D_1) = \{\text{ab}, \text{bc}, \text{ca}\}$   
Hash the shingles:  $h(D_1) = \{1, 5, 7\}$



# 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**:

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

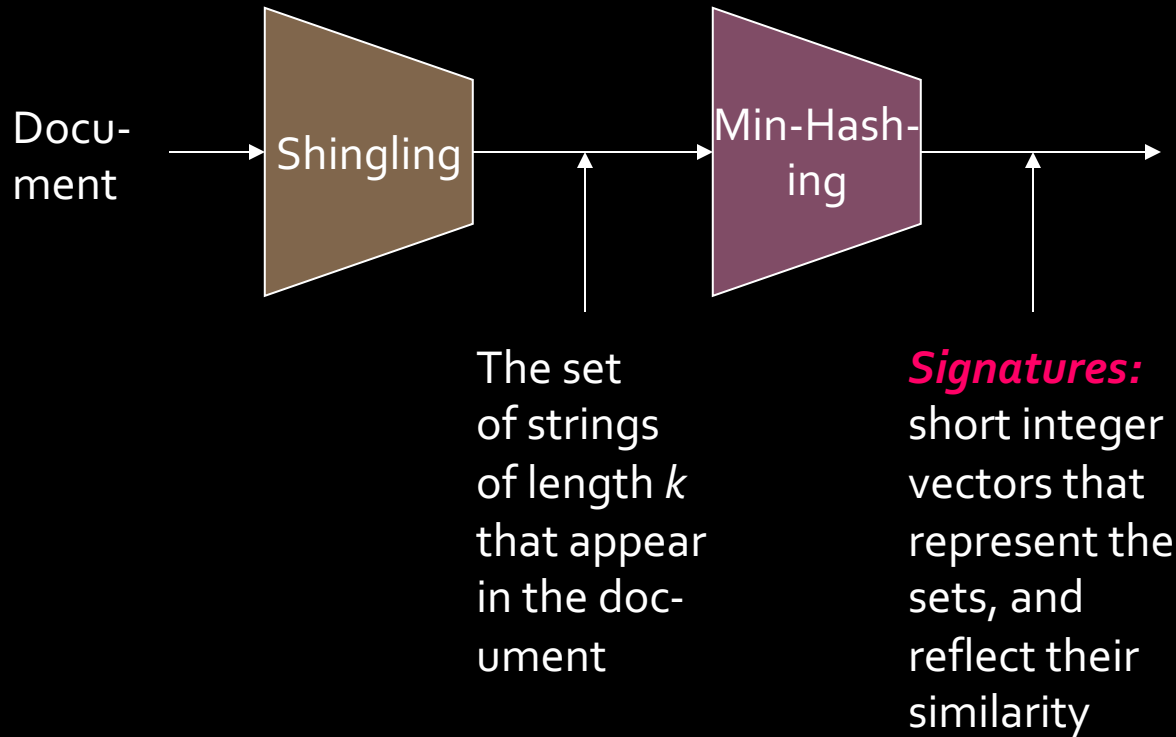


# Working Assumption

- 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**
  - $N(N - 1)/2 \approx 5 \cdot 10^{11}$  comparisons
  - At  $10^5$  secs/day and  $10^6$  comparisons/sec, it would take **5 days**
- For  $N = 10$  million, it takes more than a year...



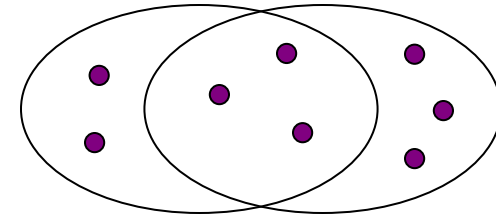
# MinHashing

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



# 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:**  $\text{sim}(C_1, C_2) = ?$ 
    - Size of intersection = 3; size of union = 6, Jaccard similarity (not distance) =  $3/6$
    - $d(C_1, C_2) = 1 - (\text{Jaccard similarity}) = 3/6$

	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 → 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)  $\text{sim}(C_1, C_2)$  is the same as the “similarity” of signatures  $h(C_1)$  and  $h(C_2)$
- **Goal: Find a hash function  $h(\cdot)$  such that:**
  - If  $\text{sim}(C_1, C_2)$  is high, then with high prob.  $h(C_1) = h(C_2)$
  - If  $\text{sim}(C_1, C_2)$  is low, then with high prob.  $h(C_1) \neq h(C_2)$
- **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  $\text{sim}(C_1, C_2)$  is high, then with high prob.  $h(C_1) = h(C_2)$
  - if  $\text{sim}(C_1, C_2)$  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**

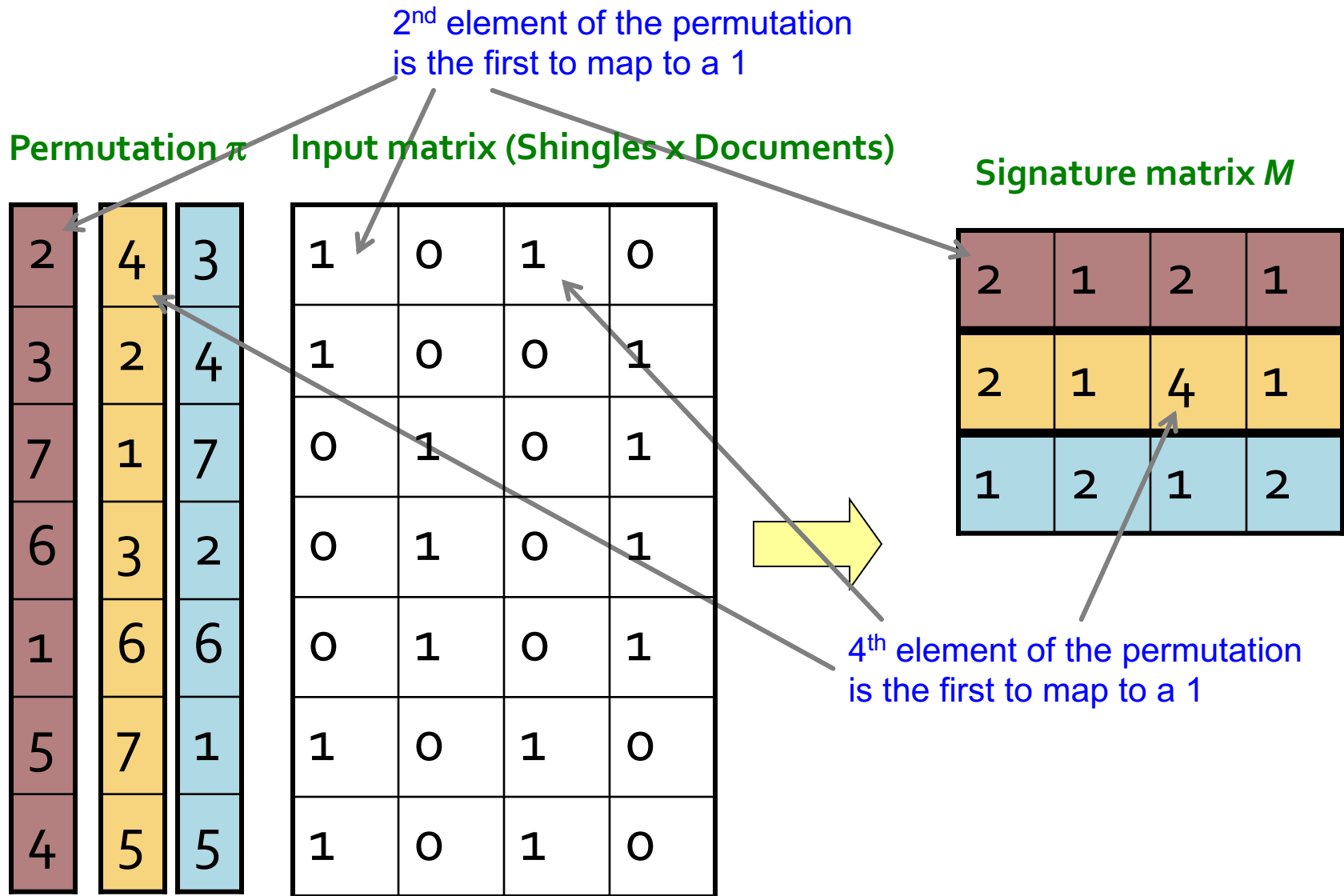
# Min-Hashing

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

$$h_{\pi}(\mathbf{C}) = \min_{\pi} \pi(\mathbf{C})$$

- 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  $\pi$
- Claim:  $\Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)$
- 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.  $y \in C_1 \cap C_2$
  - $\Pr[\min(\pi(C_1)) = \min(\pi(C_2))] = |C_1 \cap C_2| / |C_1 \cup C_2| = \text{sim}(C_1, C_2)$

# Four Types of Rows

- Given cols  $C_1$  and  $C_2$ , rows may be classified as:

	<u><math>C_1</math></u>	<u><math>C_2</math></u>
A	1	1
B	1	0
C	0	1
D	0	0

- $a$  = # rows of type A, etc.
- **Note:**  $\text{sim}(C_1, C_2) = a/(a + b + c)$
- **Then:**  $\Pr[h(C_1) = h(C_2)] = \text{Sim}(C_1, C_2)$ 
  - Look down the cols  $C_1$  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

- We know:  $\Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)$
- 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  $\pi$

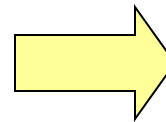
2	4	3
3	2	4
7	1	7
6	3	2
1	6	6
5	7	1
4	5	5

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:

Col/Col  
Sig/Sig

	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  $\text{sig}(\mathbf{C})$  as a column vector
- $\text{sig}(\mathbf{C})[i]$  = according to the  $i$ -th permutation, the index of the first row that has a 1 in column  $C$

$$\text{sig}(\mathbf{C})[i] = \min (\pi_i(\mathbf{C}))$$

- **Note:** The sketch (signature) of document  $C$  is small  $\sim 100$  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  $sig(C)[i] = \infty$
  - **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

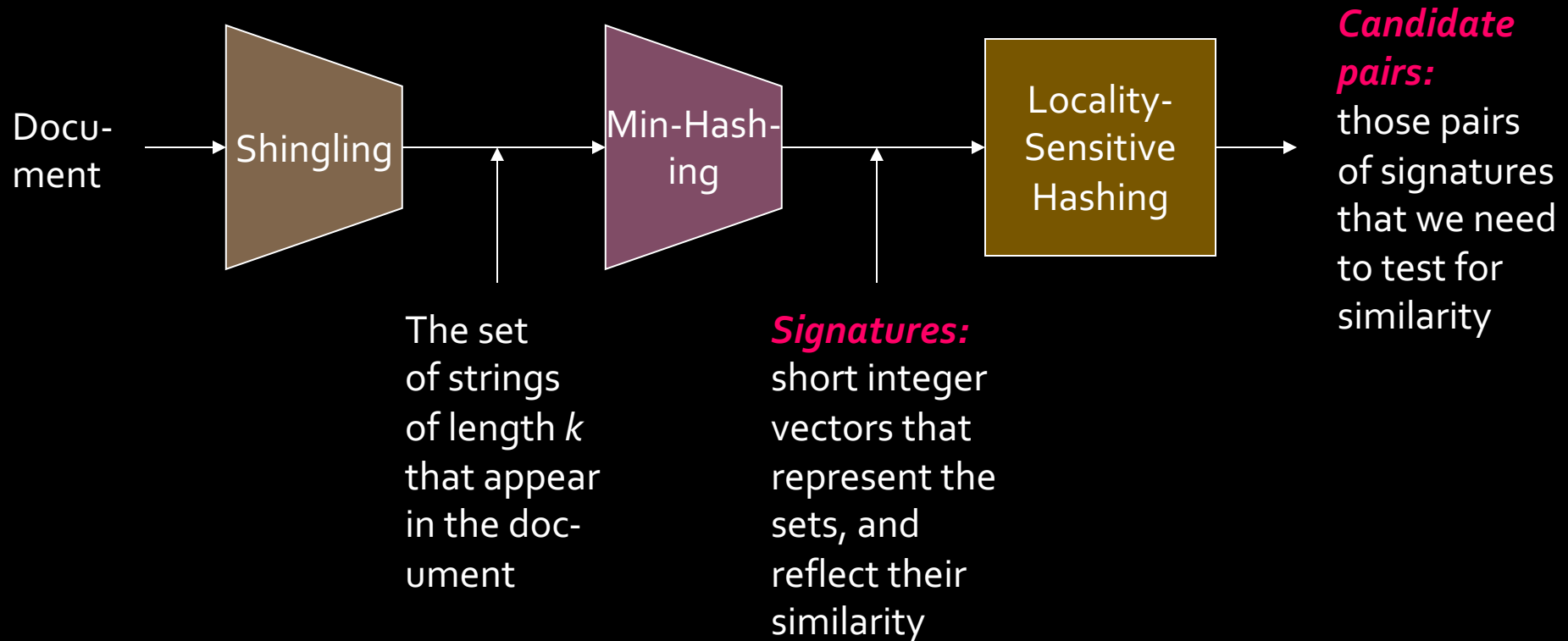
$p$  ... prime number ( $p > N$ )

# Implementation Trick

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

# Locality Sensitive Hashing

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

Focus on pairs of signatures likely to be from similar documents



# LSH: First Cut

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

- **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  $\mathbf{x}$  and  $\mathbf{y}$  of  $\mathbf{M}$  are a **candidate pair** if their signatures agree on at least fraction  $s$  of their rows:  
 $M(i, \mathbf{x}) = M(i, \mathbf{y})$  for at least frac.  $s$  values of  $i$ 
  - We expect documents  $\mathbf{x}$  and  $\mathbf{y}$  to have the same (Jaccard) similarity as their signatures

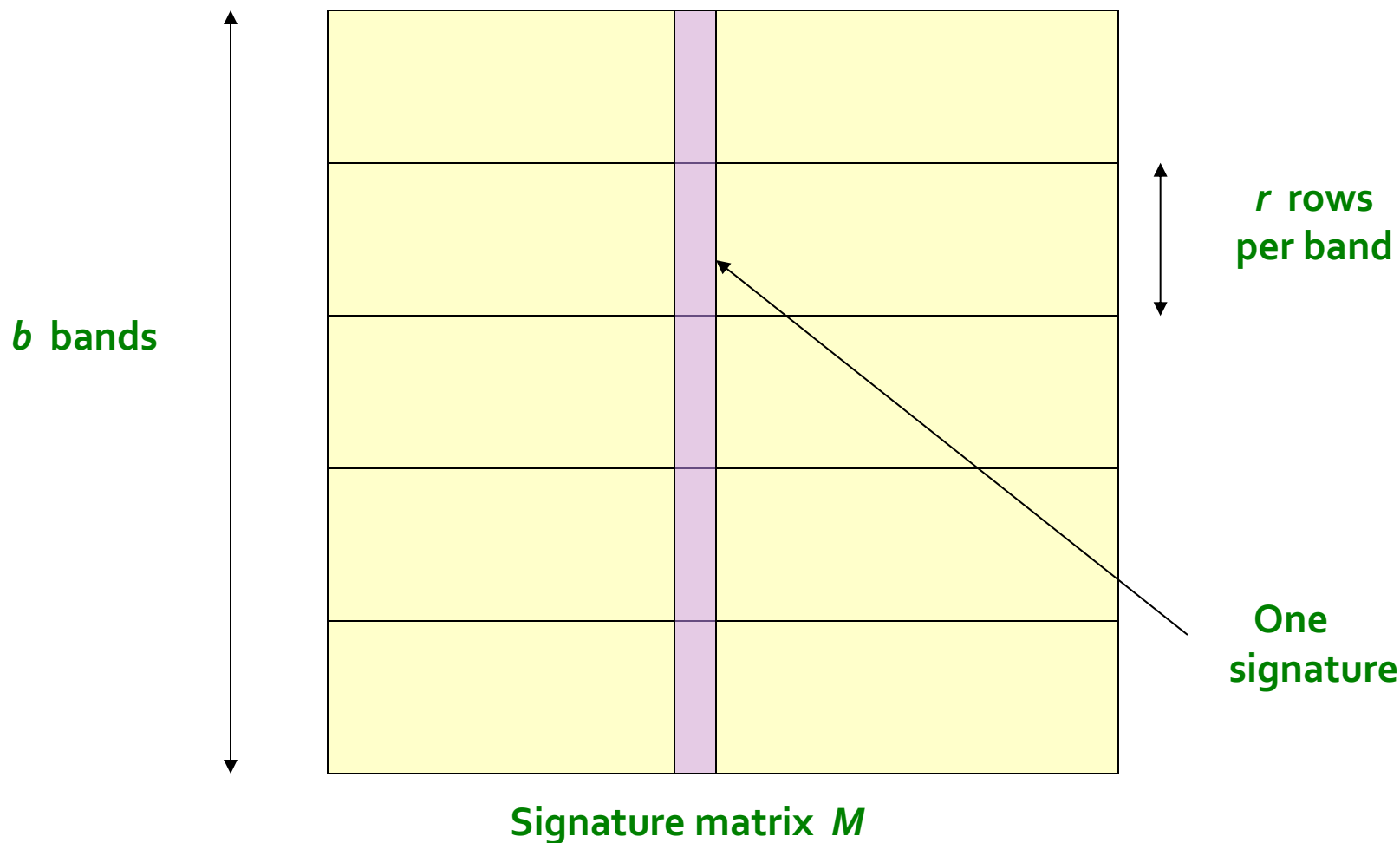
# LSH for Min-Hash

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

- **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

# Partition $M$ into $b$ Bands

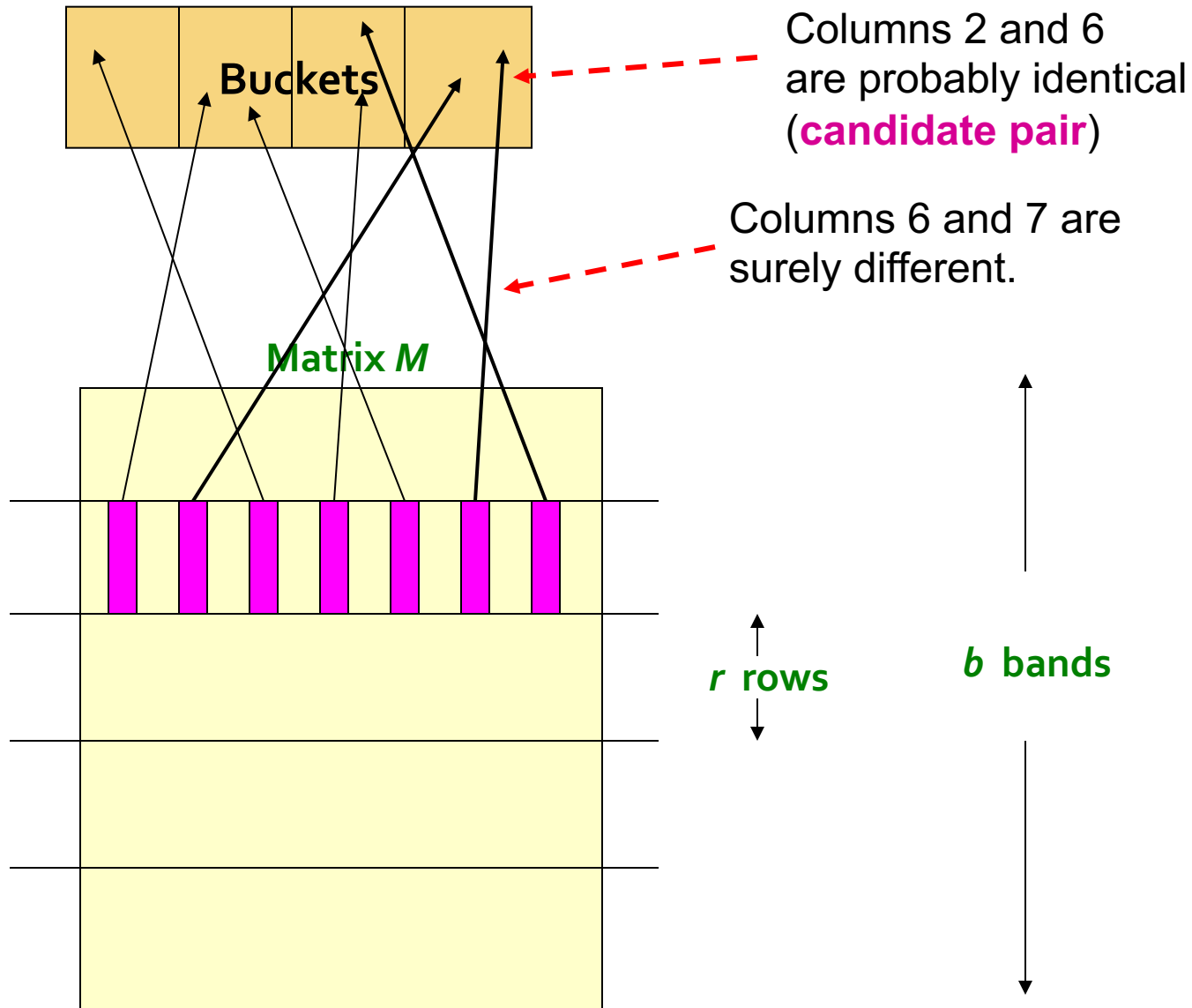
2	1	4	1
1	2	1	2
2	1	2	1



# Partition $M$ into Bands

- 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  $\geq 1$  band
- Tune  $b$  and  $r$  to catch most similar pairs, but few non-similar pairs

# Hashing Bands



# Simplifying Assumption

- There are **enough buckets** that columns are unlikely to hash to the same bucket unless they are **identical** in a particular band
- Hereafter, we assume that “**same bucket**” means “**identical in that band**”
- Assumption needed only to simplify analysis, not for correctness of algorithm

# Example of Bands

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

## Assume the following case:

- Suppose 100,000 columns of  $\mathbf{M}$  (100k docs)
- Signatures of 100 integers (rows)
- Therefore, signatures take 40Mb
- Choose  $b = 20$  bands of  $r = 5$  integers/band
- **Goal:** Find pairs of documents that are at least  $s = 0.8$  similar



# $C_1, C_2$ are 80% Similar

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

- Find pairs of  $\geq s=0.8$  similarity, set  $b=20$ ,  $r=5$
- **Assume:**  $\text{sim}(C_1, C_2) = 0.8$ 
  - Since  $\text{sim}(C_1, C_2) \geq s$ , we want  $C_1, C_2$  to be a **candidate pair**: We want them to hash to at **least 1 common bucket** (at least one band is identical)
- **Probability  $C_1, C_2$  identical in one particular band:**  $(0.8)^5 = 0.328$
- Probability  $C_1, C_2$  are **not** similar in all of the 20 bands:  $(1-0.328)^{20} = 0.00035$ 
  - i.e., about 1/3000th of the 80%-similar column pairs are **false negatives** (we miss them)
  - **We would find 99.965% pairs of truly similar documents**

# $C_1, C_2$ are 30% Similar

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

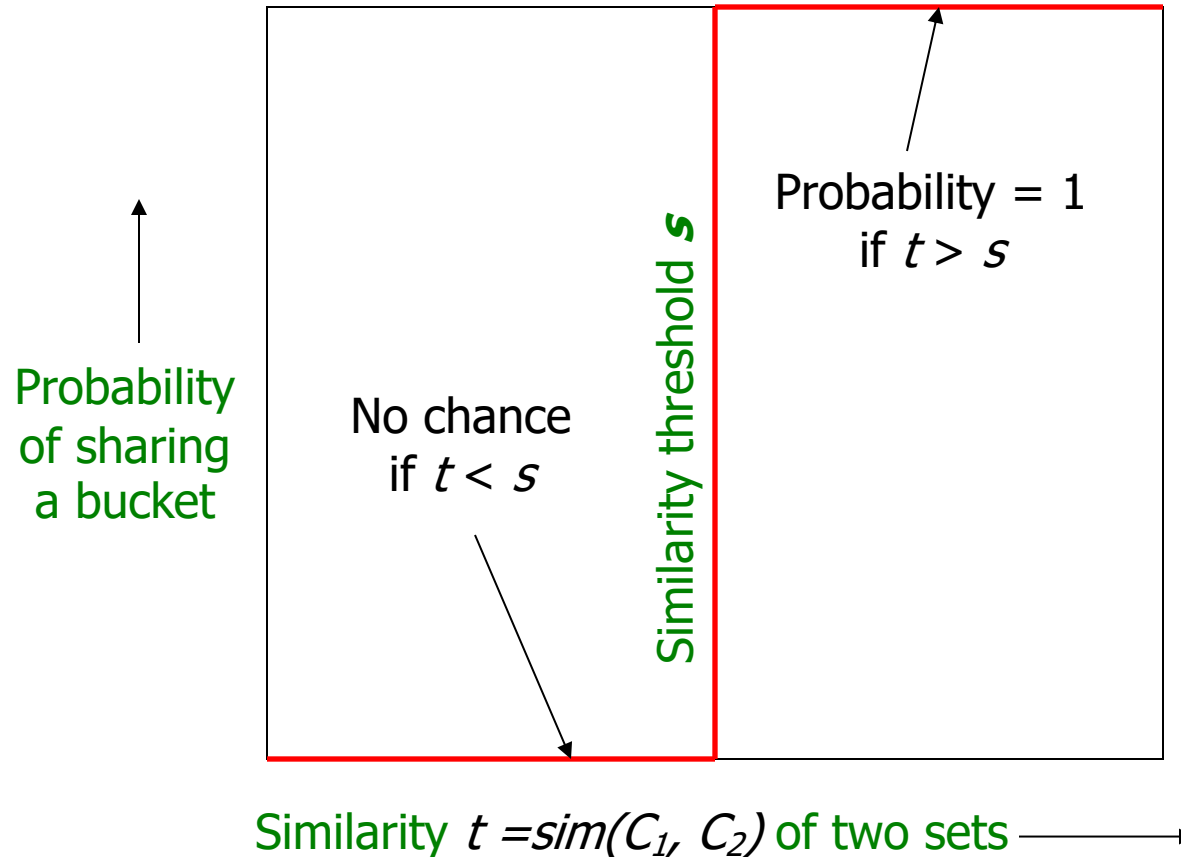
- Find pairs of  $\geq s=0.8$  similarity, set  $b=20, r=5$
- **Assume:**  $\text{sim}(C_1, C_2) = 0.3$ 
  - Since  $\text{sim}(C_1, C_2) < s$  we want  $C_1, C_2$  to hash to **NO common buckets** (all bands should be different)
- **Probability  $C_1, C_2$  identical in one particular band:**  $(0.3)^5 = 0.00243$
- Probability  $C_1, C_2$  identical in at least 1 of 20 bands:  $1 - (1 - 0.00243)^{20} = 0.0474$ 
  - In other words, approximately 4.74% pairs of docs with similarity 0.3% end up becoming **candidate pairs**
    - They are **false positives** since we will have to examine them (they are candidate pairs) but then it will turn out their similarity is below threshold  $s$

# LSH Involves a Tradeoff

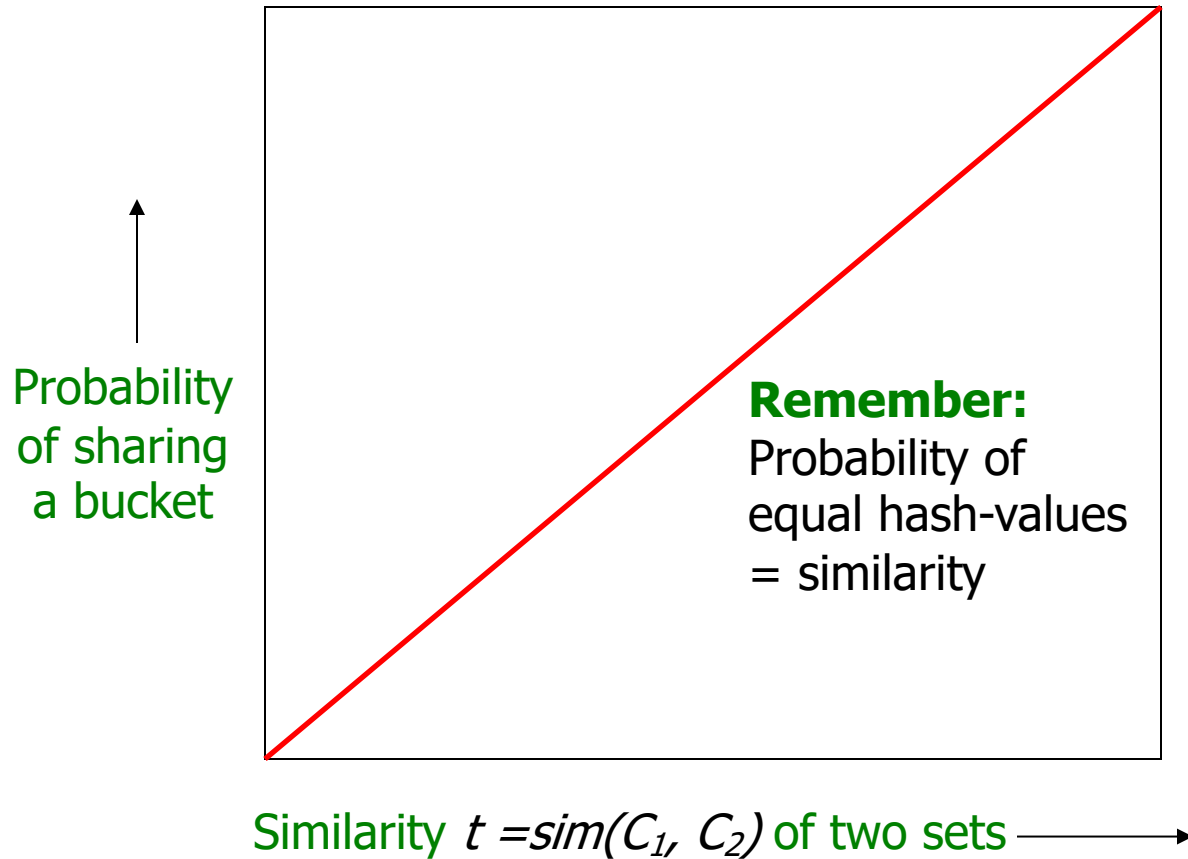
2	1	4	1
1	2	1	2
2	1	2	1

- **Pick:**
  - The number of Min-Hashes (rows of  $\mathbf{M}$ )
  - The number of bands  $\mathbf{b}$ , and
  - The number of rows  $\mathbf{r}$  per bandto balance false positives/negatives
- **Example:** If we had only 15 bands of 5 rows, the number of false positives would go down, but the number of false negatives would go up

# Analysis of LSH – What We Want



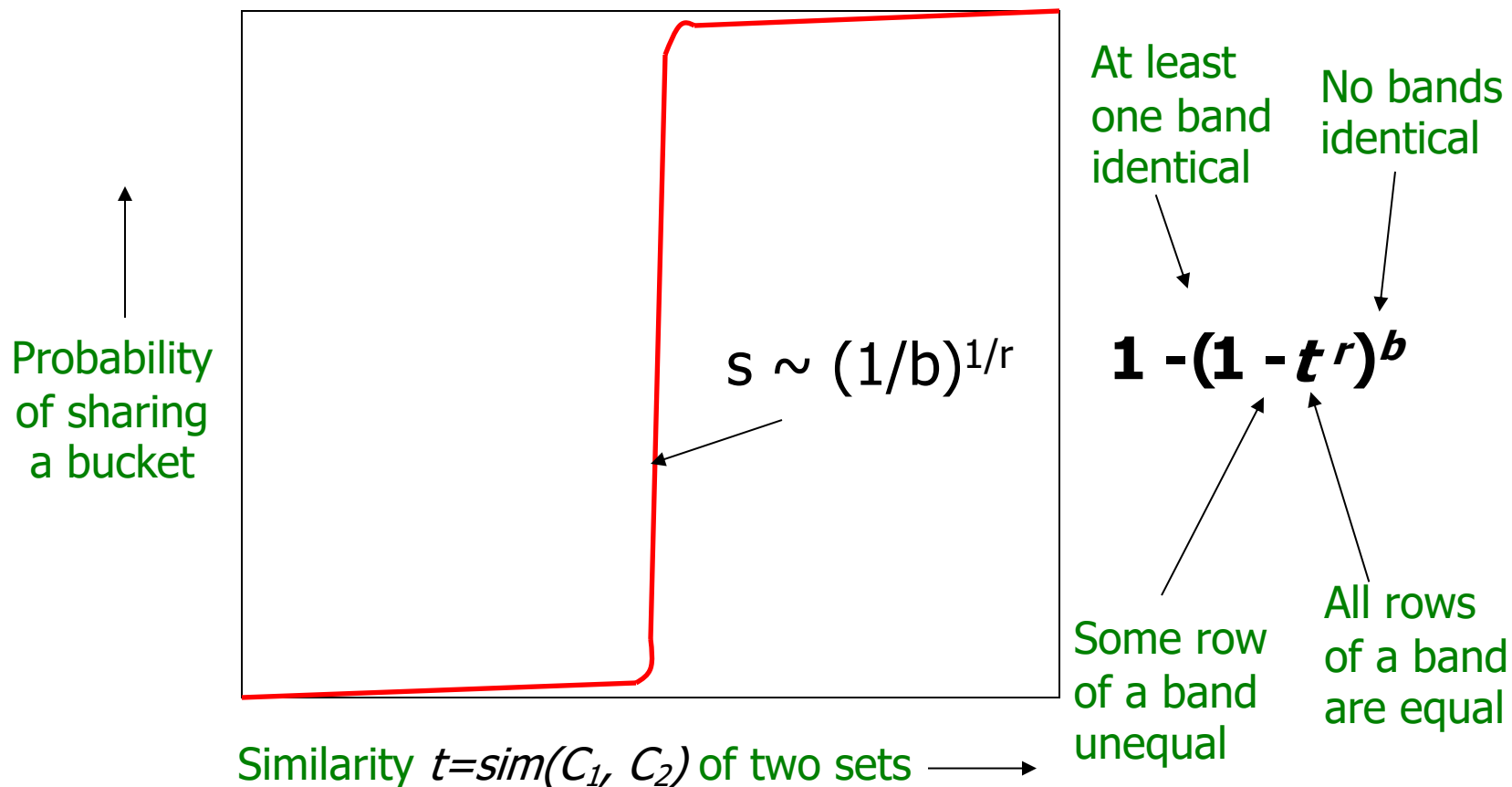
# What 1 Band of 1 Row Gives You



# $b$ bands, $r$ rows/band

- Columns  $C_1$  and  $C_2$  have similarity  $t$
- Pick any band ( $r$  rows)
  - Prob. that all rows in band equal =  $t^r$
  - Prob. that some row in band unequal =  $1 - t^r$
- Prob. that no band identical =  $(1 - t^r)^b$
- Prob. that at least 1 band identical =  $1 - (1 - t^r)^b$

# What $b$ Bands of $r$ Rows Gives You



# Example: $b = 20; r = 5$

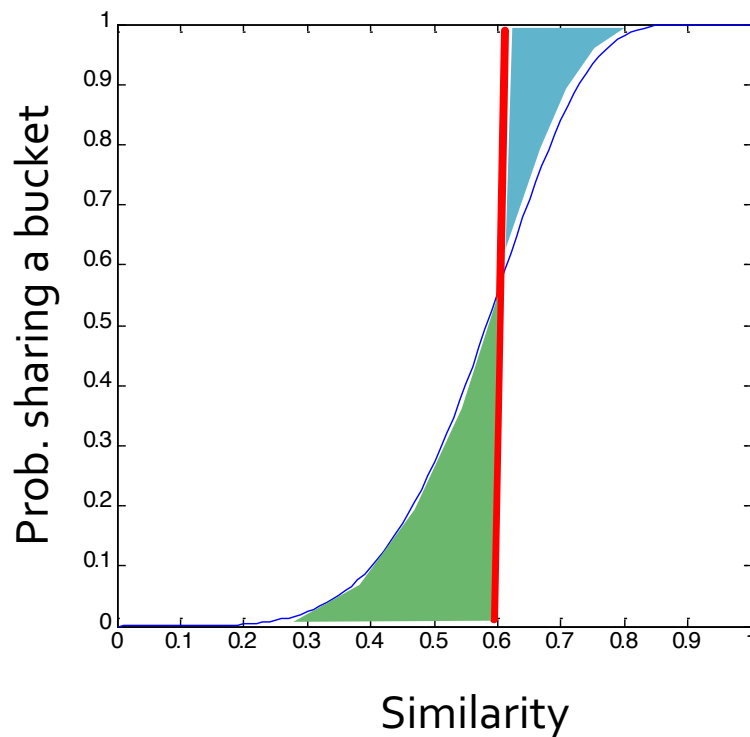
- Similarity threshold  $s$
- Prob. that at least 1 band is identical:

$s$	$1-(1-s^r)^b$
.2	.006
.3	.047
.4	.186
.5	.470
.6	.802
.7	.975
.8	.9996



# Picking $r$ and $b$ : The S-curve

- Picking  $r$  and  $b$  to get the best S-curve
  - 50 hash-functions ( $r=5$ ,  $b=10$ )



Blue area: False Negative rate

Green area: False Positive rate

# LSH Summary

- Tune  $M, b, r$  to get almost all pairs with similar signatures, but eliminate most pairs that do not have similar signatures
- Check in main memory that **candidate pairs** really do have **similar signatures**
- **Optional:** In another pass through data, check that the remaining candidate pairs really represent similar documents

# 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  $\Pr[h_{\pi}(C_1) = h_{\pi}(C_2)] = \text{sim}(C_1, C_2)$
  - 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  $\geq s$