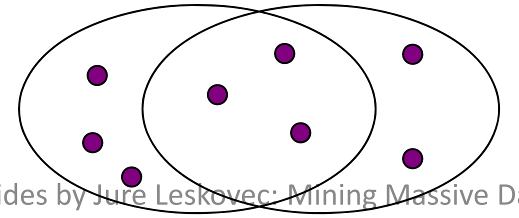
Finding Similar Items

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 the similar websites

Distance Measures

- We formally define "near neighbors" as points that are a "small distance" apart
- For each use case, we need to define what "**distance**" means
- **Today: Jaccard similarity/distance**
  - The *Jaccard Similarity/Distance* of two sets is the size of their intersection / the size of their union:
  - $sim(C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$
  - $d(C_1, C_2) = 1 - |C_1 \cap C_2| / |C_1 \cup C_2|$



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

Distance Measures

- We formally define "near neighbors" as points that are a "small distance" apart
- For each use case, we need to define what "**distance**" means
- **Two major classes of distance measures:**
  - A *Euclidean distance* is based on the locations of points in such a space
  - A *Non-Euclidean distance* is based on properties of points, but not their "location" in a space

Some Euclidean Distances

- ***L_2 norm:*** *d(p,q)* = square root of the sum of the squares of the differences between *p* and *q* in each dimension:

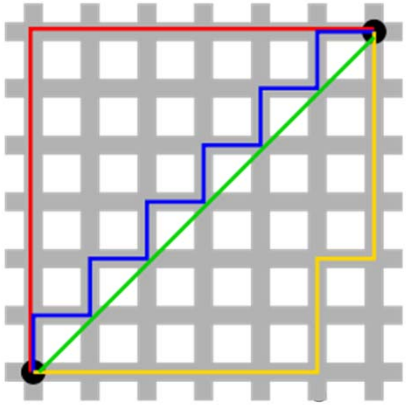
$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2 + (q_2-p_2)^2 + \cdots + (q_n-p_n)^2} = \sqrt{\sum_{i=1}^n (q_i-p_i)^2}.$$

  – The most common notion of "distance"

- ***L_1 norm:*** sum of the absolute differences in each dimension

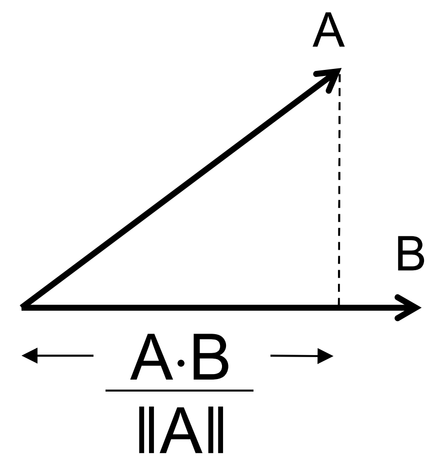
  – *Manhattan distance* = distance if you had to travel along coordinates only

$$d_1(\mathbf{p},\mathbf{q}) = \|\mathbf{p}-\mathbf{q}\|_1 = \sum_{i=1}^n |p_i - q_i|,$$



Non-Euclidean Distances: Cosine

- Think of a point as a vector from the origin (0,0,...,0) to its location
- Two vectors make an angle, whose cosine is normalized dot-product of the vectors:

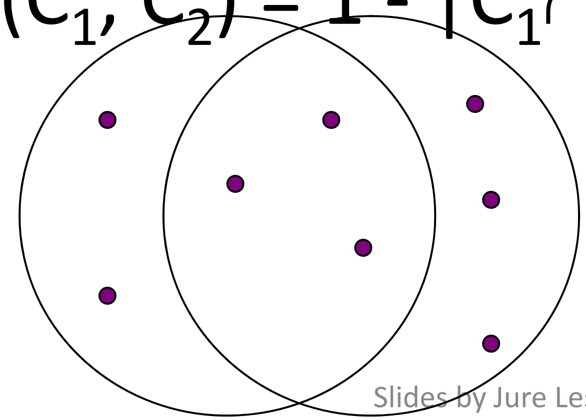


$$d(A,B) = \theta = \arccos\left(\frac{A \cdot B}{||A|| \cdot ||B||}\right)$$

- **Example:** A = 00111; B = 10011
  - A·B = 2; ‖A‖ = ‖B‖ = √3
  - cos(θ) = 2/3; θ is about 48 degrees

Non-Euclidean Distances: Jaccard

- The *Jaccard Similarity* of two sets is the size of their intersection / the size of their union:
  - $Sim(C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$
- The *Jaccard Distance* between sets is 1 minus their Jaccard similarity:
  - $d(C_1, C_2) = 1 - |C_1 \cap C_2| / |C_1 \cup C_2|$



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

Finding Similar Items

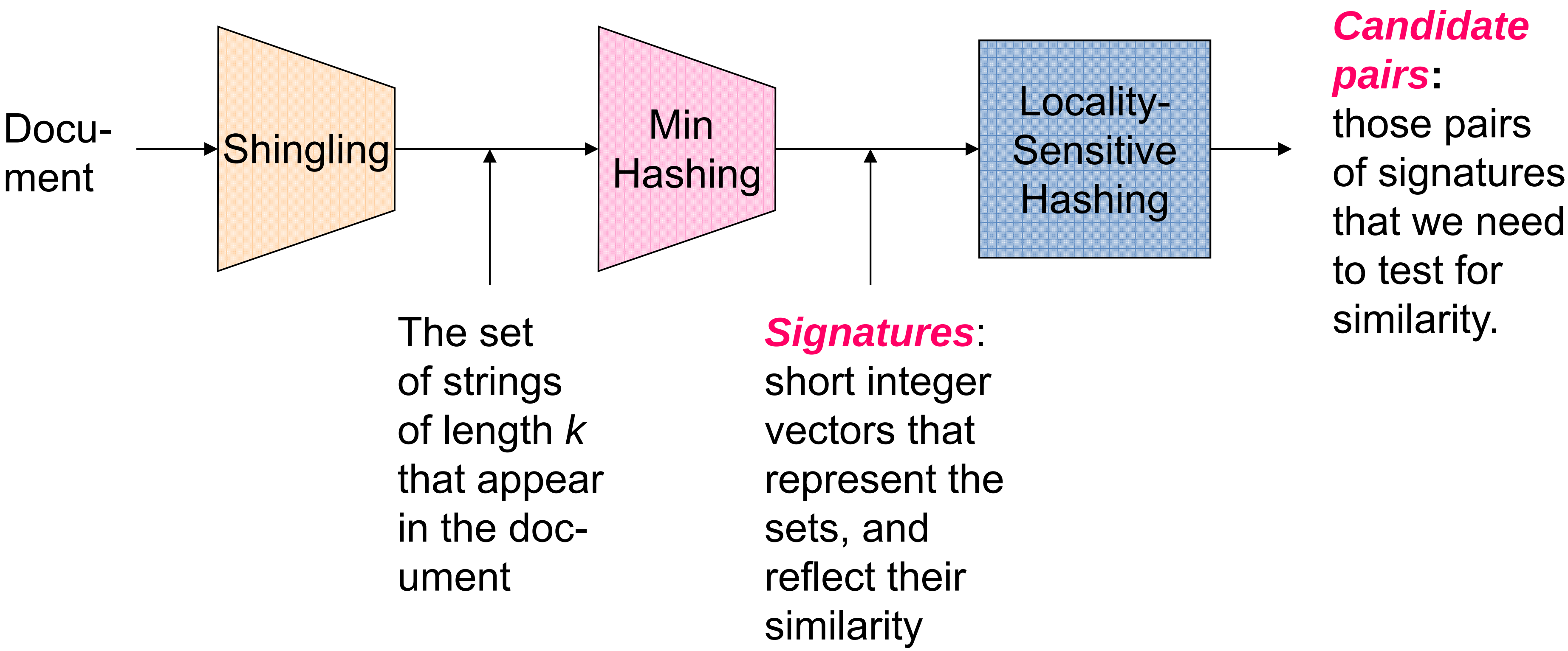
Finding Similar Documents

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

3 Essential Steps for Similar Docs

1. ***Shingling:*** Convert documents, emails, etc., to sets
2. ***Minhashing:*** Convert large sets to short signatures, while preserving similarity
3. ***Locality-sensitive hashing:*** Focus on pairs of signatures likely to be from similar documents

The Big Picture



Documents as High-Dim Data

- **Step 1:** ***Shingling:*** Convert documents, emails, etc., to sets

- **Simple approaches:**
  - Document = set of words appearing in doc
  - 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 application
  - Assume tokens = characters for examples

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

Compressing Shingles

- To **compress long shingles**, we can hash them to (say) 4 bytes
- **Represent a doc 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; D_1= abcab
  Set of 2-shingles: $S(D_1)$={ab, bc, ca}
  Hash the singles: $h(D_1)$={1, 5, 7}

Working Assumption

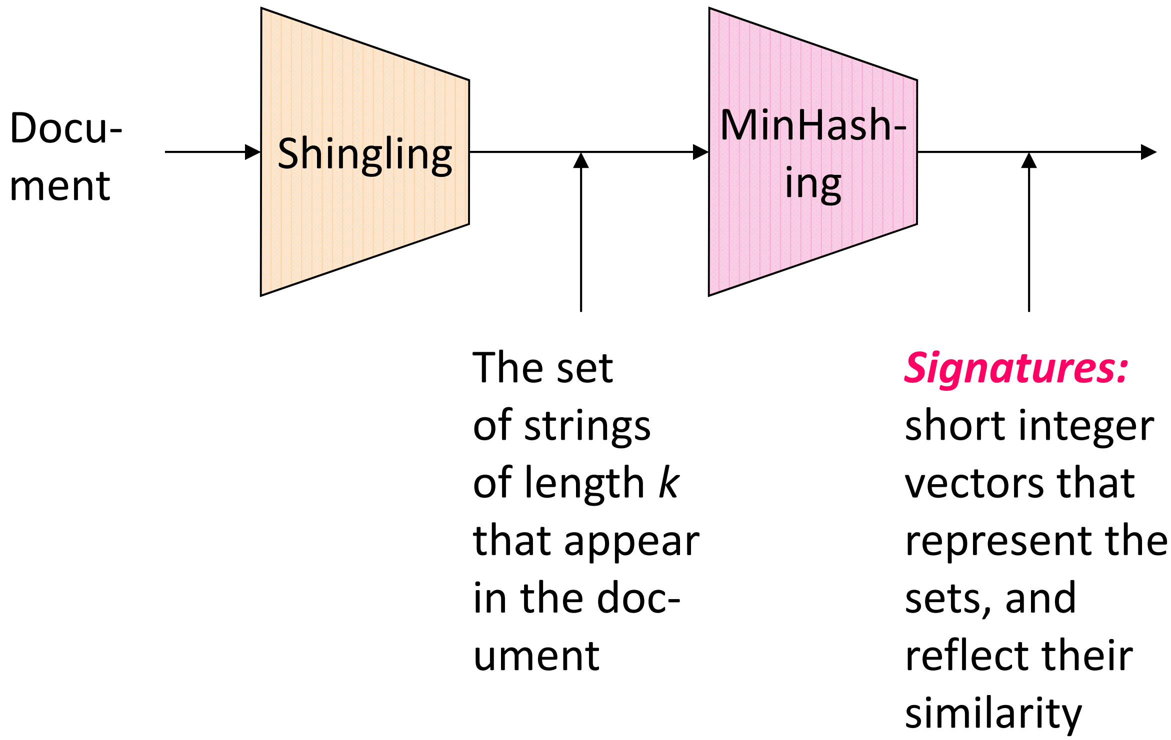
- Documents that have lots of shingles in common have similar text, even if the text appears in different order
- **Careful:** You must pick k large enough, or most documents will have most shingles
  - $k = 5$ is OK for short documents
  - $k = 10$ is better for long documents

Motivation for Minhash/LSH

- **Suppose we need to find near-duplicate documents among N=1 million documents**

- Naïvely, we'd have to compute pairwise Jaccard similarities for every pair of docs
  - i.e, $N(N-1)/2 \approx 5*10^{11}$ comparisons
  - At 10^5 secs/day and 10^6 comparisons/sec, it would take 5 days

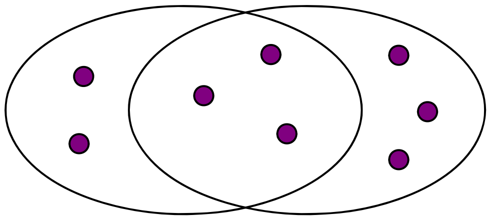
- For N = 10 million, it takes more than a year…



MinHashing

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

Encoding Sets as Bit Vectors



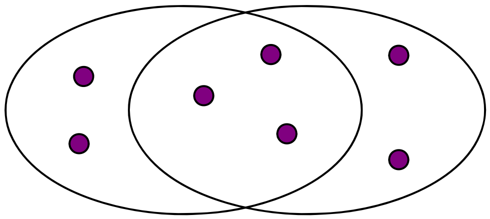
- Many similarity problems can be formalized as **finding subsets that have significant intersection**
- **Encode sets using 0/1 (bit, boolean) vectors**
  - One dimension per element in the universal set
- Interpret set intersection as bitwise **AND**, and set union as bitwise **OR**
- **Example:** C_1 = 10111; C_2 = 10011
  - Size of intersection = 3; size of union = 4, Jaccard similarity (not distance) = 3/4
  - $d(C_1,C_2)$ = 1 – (Jaccard similarity) = 1/4

From Sets to Boolean Matrices

- Rows = elements of the universal set
- 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



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

Example: Jaccard of Columns

- **Each document is a column:**
  - Example: C_1 = 1100011; C_2 = 0110010
    - Size of intersection = 2; size of union = 5, Jaccard similarity (not distance) = 2/5
    - $d(C_1,C_2)$ = 1 – (Jaccard similarity) = 3/5

Note:

- We might not really represent the data by a boolean matrix
- Sparse matrices are usually better represented by the list of places where there is a non-zero value

shingles

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

documents

Outline: Finding Similar Columns

- **So far:**
  - Documents → Sets of shingles
  - Represent sets as boolean vectors in a matrix
- **Next Goal: Find similar columns, Small signatures**
- **Approach:**
  - **1) Signatures of columns:** small summaries of columns
  - **2) Examine pairs of signatures** to find similar columns
    - **Essential:** Similarities of signatures & columns are related
  - **3) Optional:** check that columns with similar sigs. 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 (Singatures)

- **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. $h(C_1) \neq h(C_2)$
- Hash docs into buckets, and expect that "most" pairs of near duplicate docs hash into the same bucket

Min-Hashing

- **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. $h(C_1) \neq h(C_2)$
- 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 Jaccard similarity: **Min-hashing**

Min-Hashing

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

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

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

Min-Hashing Example

Permutation π

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

Signature matrix *M*

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

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

Surprising Property

- Choose a random permutation π
- then **Pr**$[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- **Why?**
  - Let X be a set of shingles, $X \subseteq [2^{64}]$, $x \in X$
  - **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 x be s.t. $\pi(x) = \min(\pi(C_1 \cup C_2))$
  - Then either: $\pi(x) = \min(\pi(C_1))$ if $x \in C_1$, **or**
    $\pi(x) = \min(\pi(C_2))$ if $x \in C_2$
  - So the prob. that both are true is the prob. $x \in C_1 \cap C_2$
  - $\Pr[\min(\pi(C_1)) = \min(\pi(C_2))] = |C_1 \cap C_2| / |C_1 \cup C_2| = sim(C_1, C_2)$

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 minhash property, the similarity of columns is the same as the expected similarity of their signatures

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

Similarities:

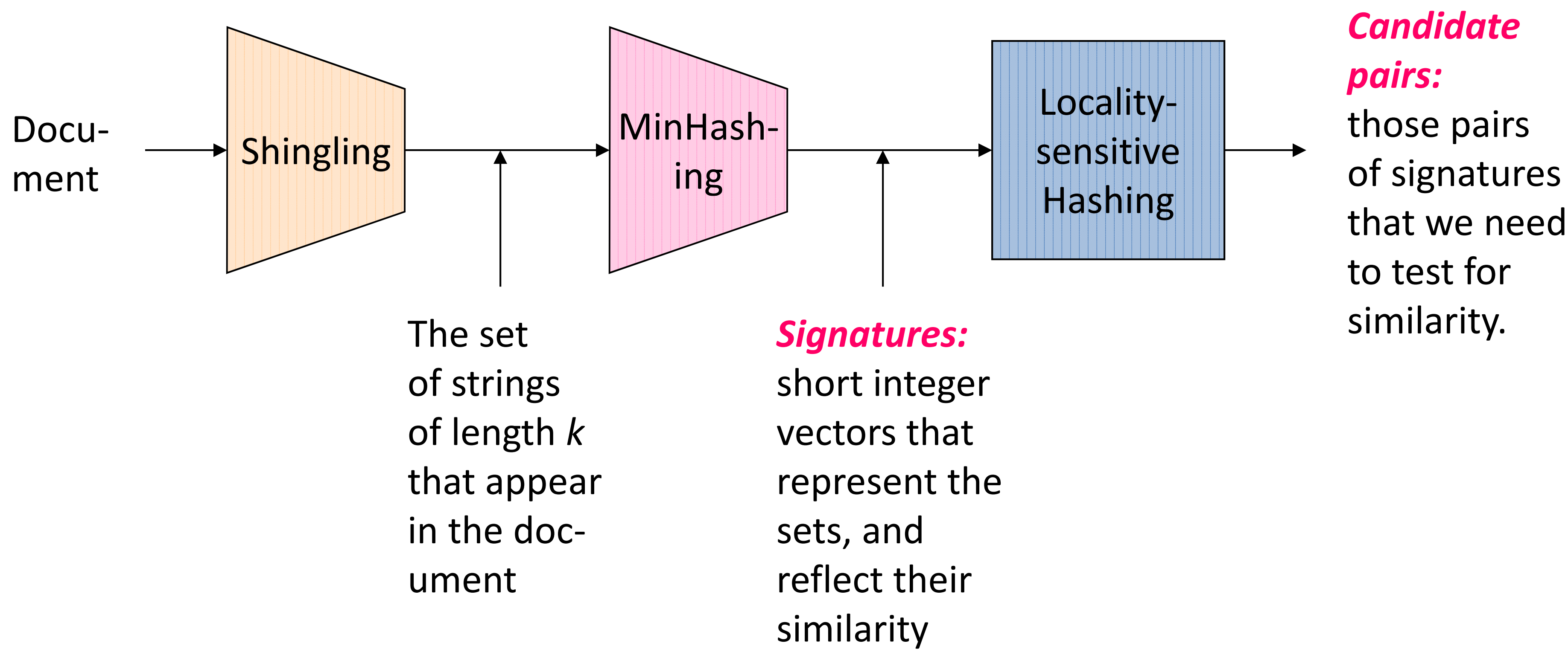
	1-3	2-4	1-2	3-4
Col/Col	0.75	0.75	0	0
Sig/Sig	0.67	1.00	0	0

MinHash Signatures

- Pick 100 random permutations of the rows
- Think of *sig*(C) as a column vector
- Let *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



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 minhash 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 Minha

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

- Pick a similarity threshold *s*, a fraction < 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 similarity as their signatures

LSH for Minhash

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

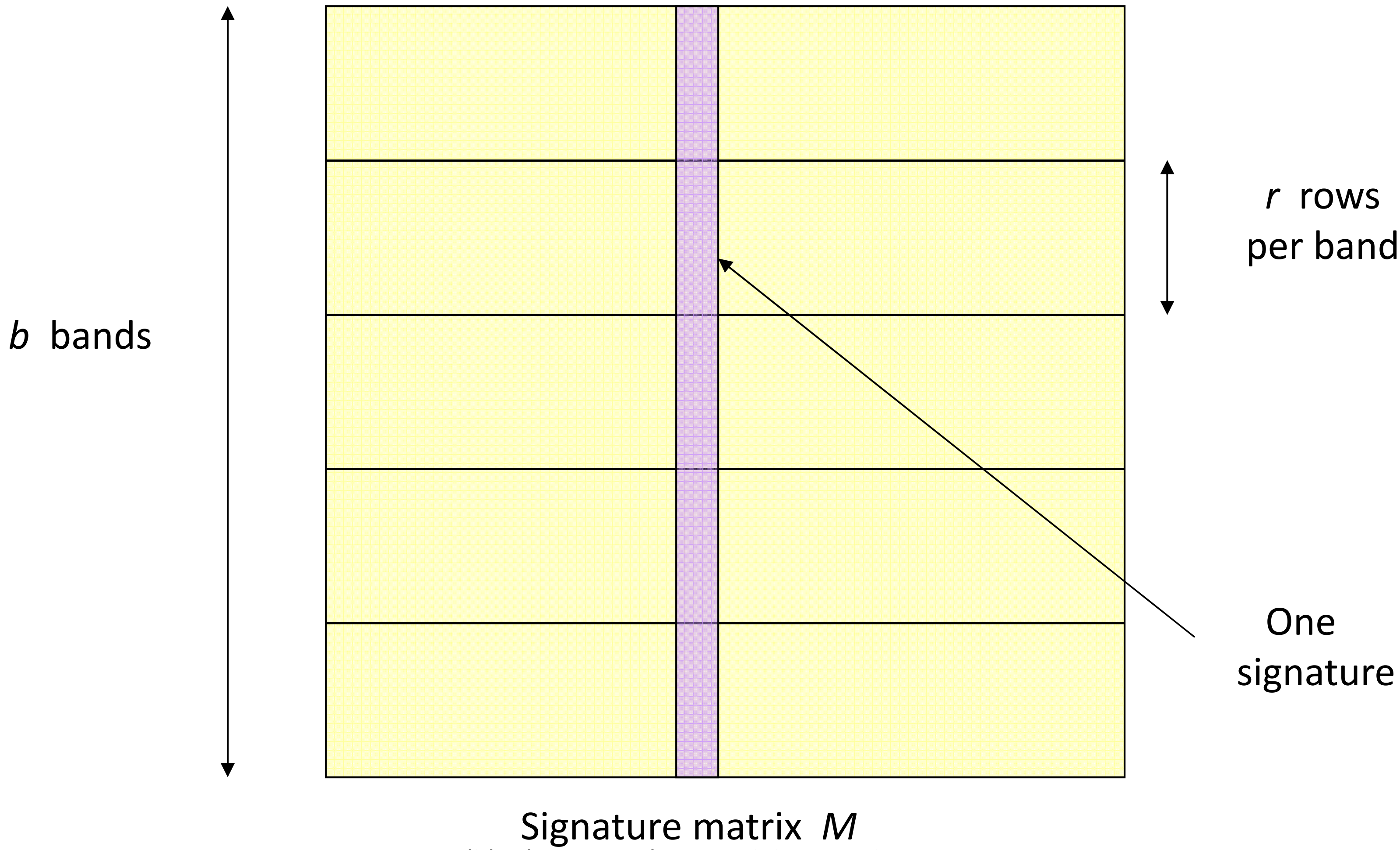
- **Big idea:** Hash columns of signature matrix *M* several times

- 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

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

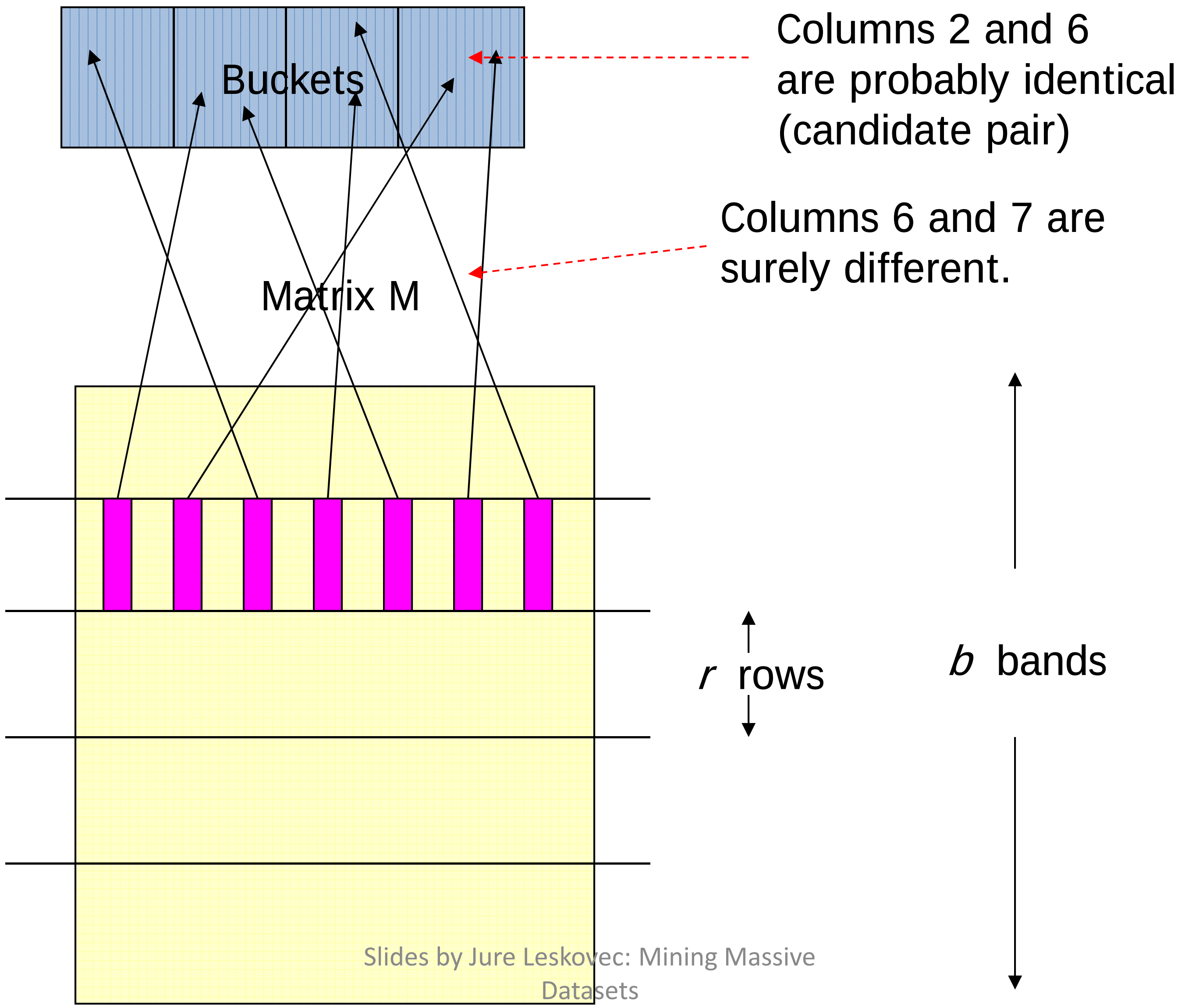
Partition M into Banc



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 ≥ 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 *M* (100k docs)
- Signatures of 100 integers (rows)
- Therefore, signatures take 40Mb
- Choose 20 bands of 5 integers/band

- **Goal:** Find pairs of documents that are at least s = 80% similar

C_1, C_2 are 80% Simila

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

- **Assume:** C_1, C_2 are 80% similar
  - Since s=80% we want C_1, C_2 to hash to at **least one 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: $(1\text{-}0.328)^{20} = 0.00035$
  - i.e., about 1/3000th of the 80%-similar column pairs are false negatives
  - We would find 99.965% pairs of truly similar documents

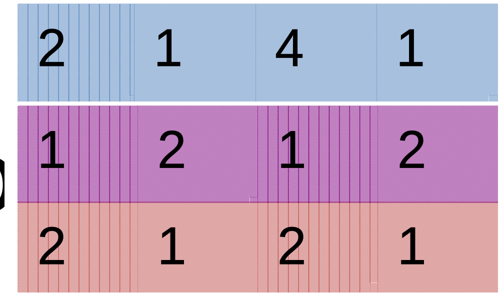
2	1	4	1
1	2	1	2
2	1	2	1

C_1, C_2 are 30% Simila

- **Assume:** C_1, C_2 are 30% similar
  - Since s=80% we want C_1, C_2 to hash to at **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 30% end up becoming candidate pairs -- **false positives**

LSH Involves a Tradeo

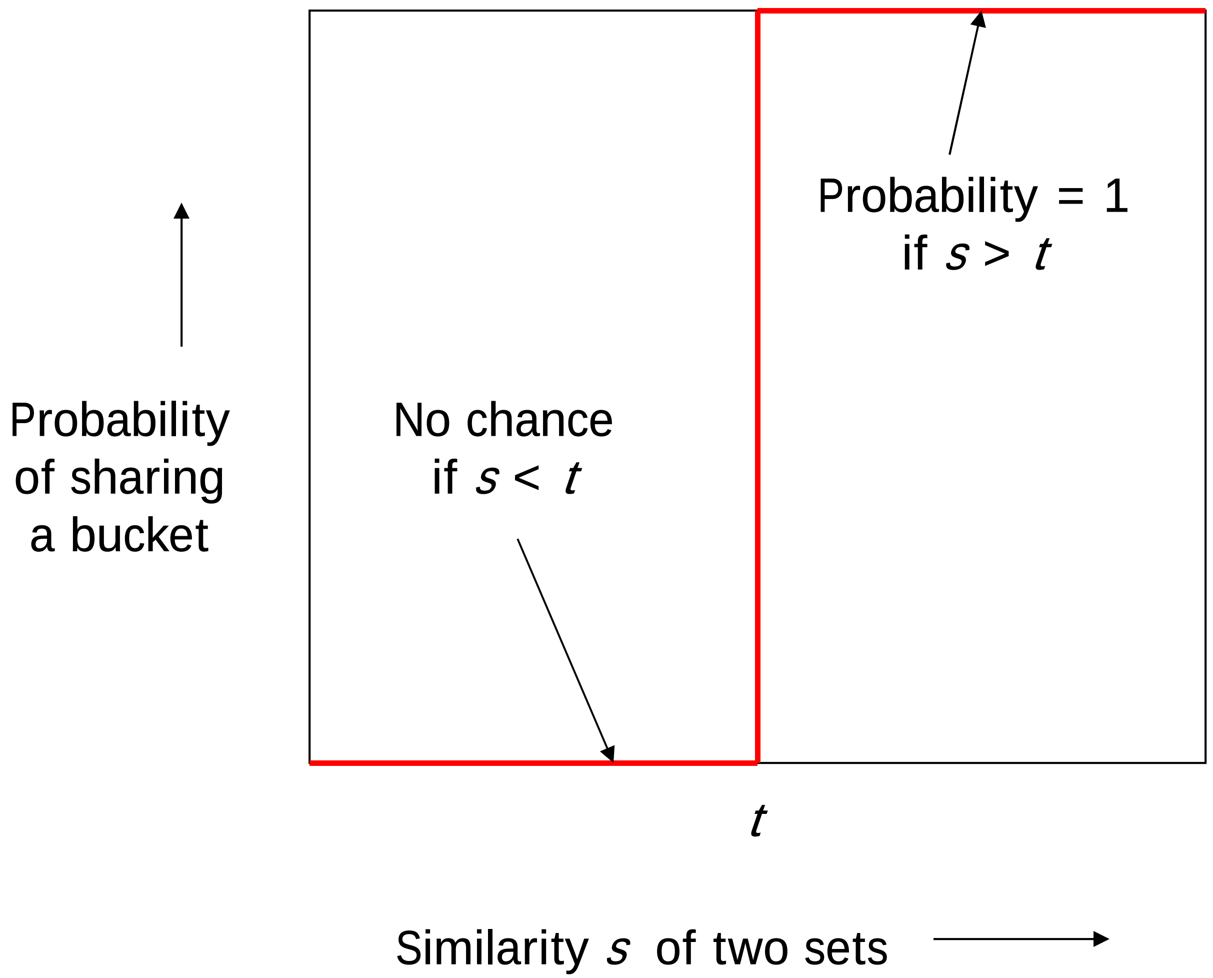


- **Pick:**
  - the number of minhashes (rows of M)
  - the number of bands *b*, and
  - the number of rows *r* per band

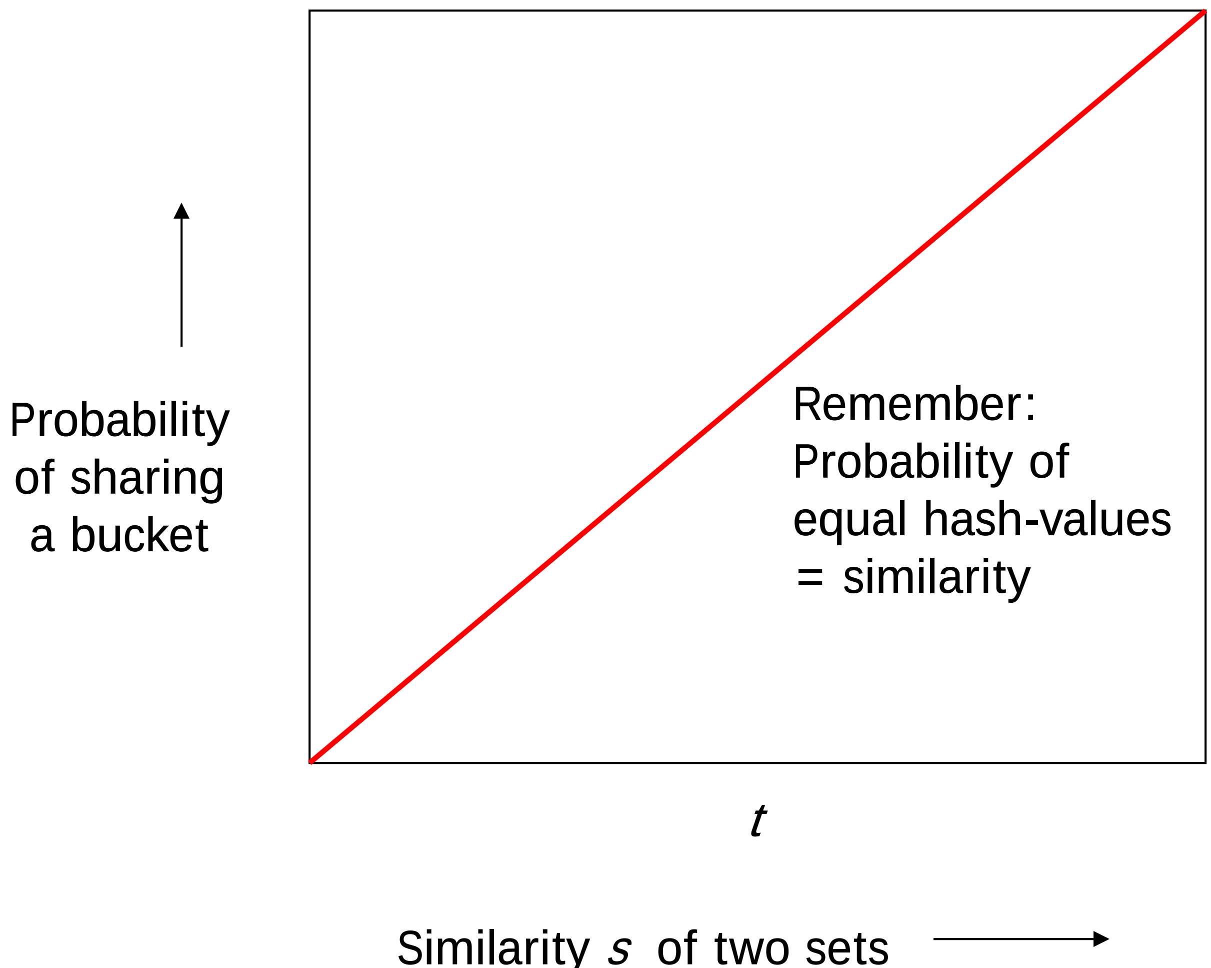
  to balance false positives/negatives

- **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

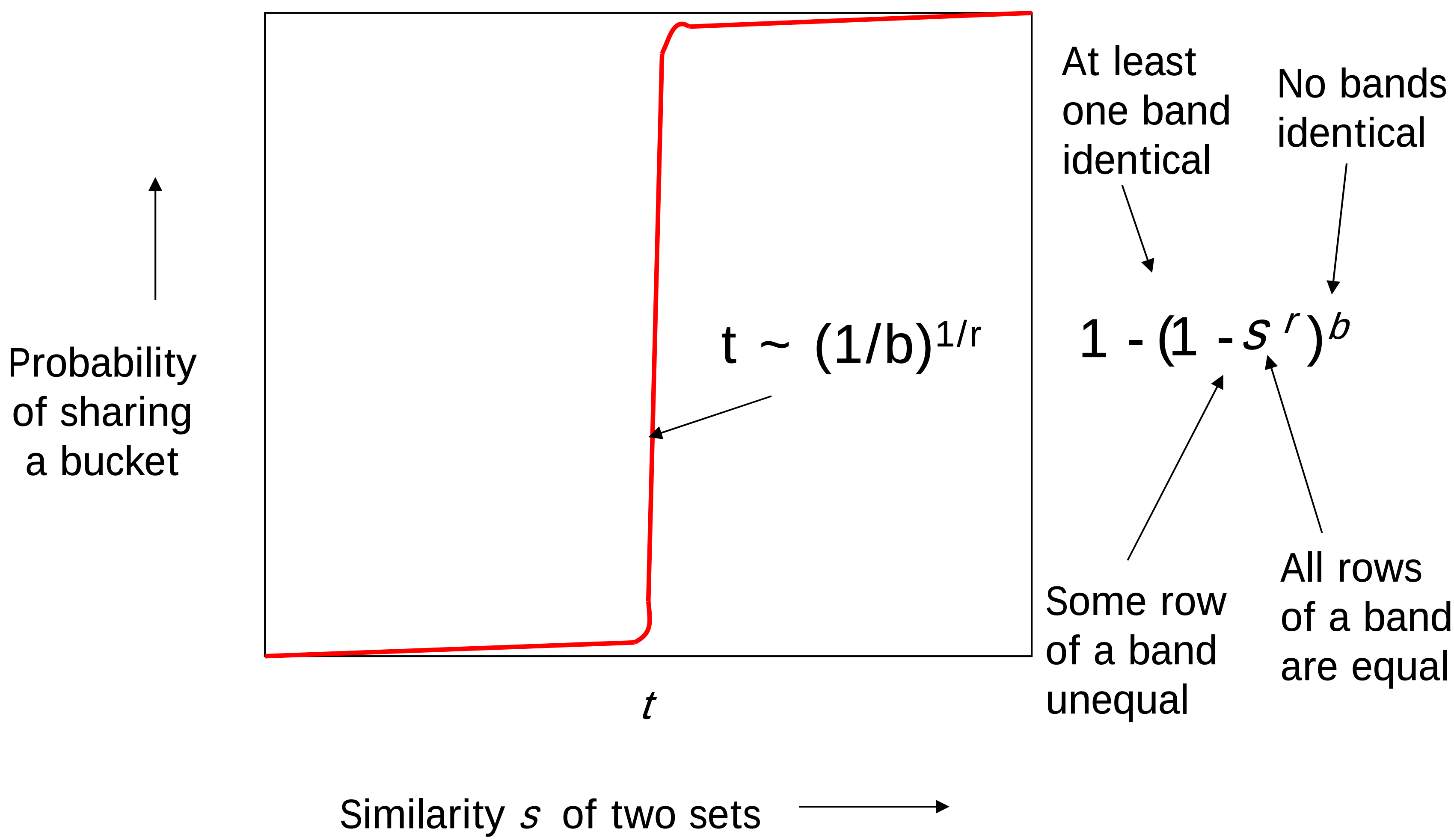
Analysis of LSH – What We Want



What 1 Band of 1 Row Gives You



What b Bands of r Rows Gives You



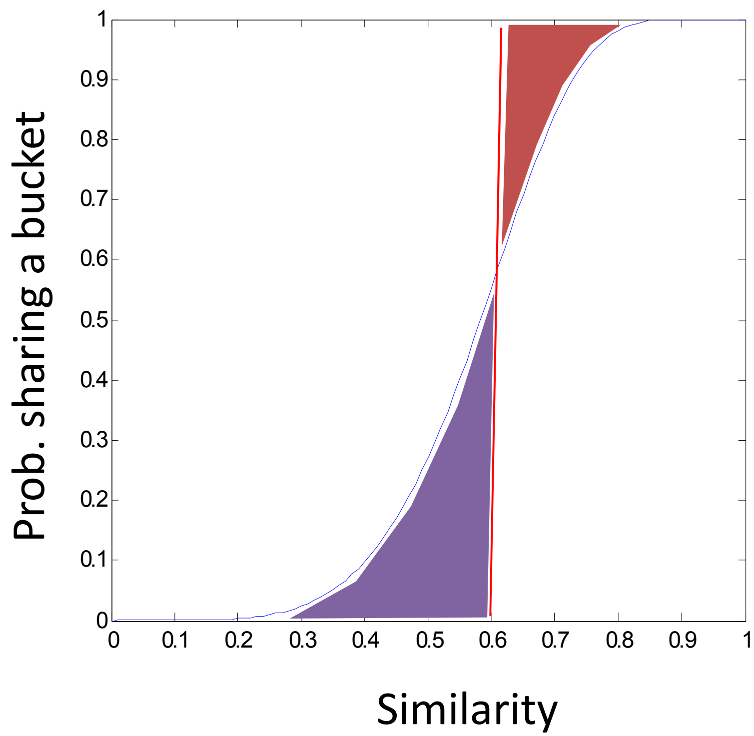
Example: $b = 20$; $r = 5$

- Similarity threshold s
- Prob. that at least 1 band 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 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

1. ***Shingling:*** Convert documents, emails, etc., to sets

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

3. ***Locality-sensitive hashing:*** Focus on pairs of signatures likely to be from similar documents