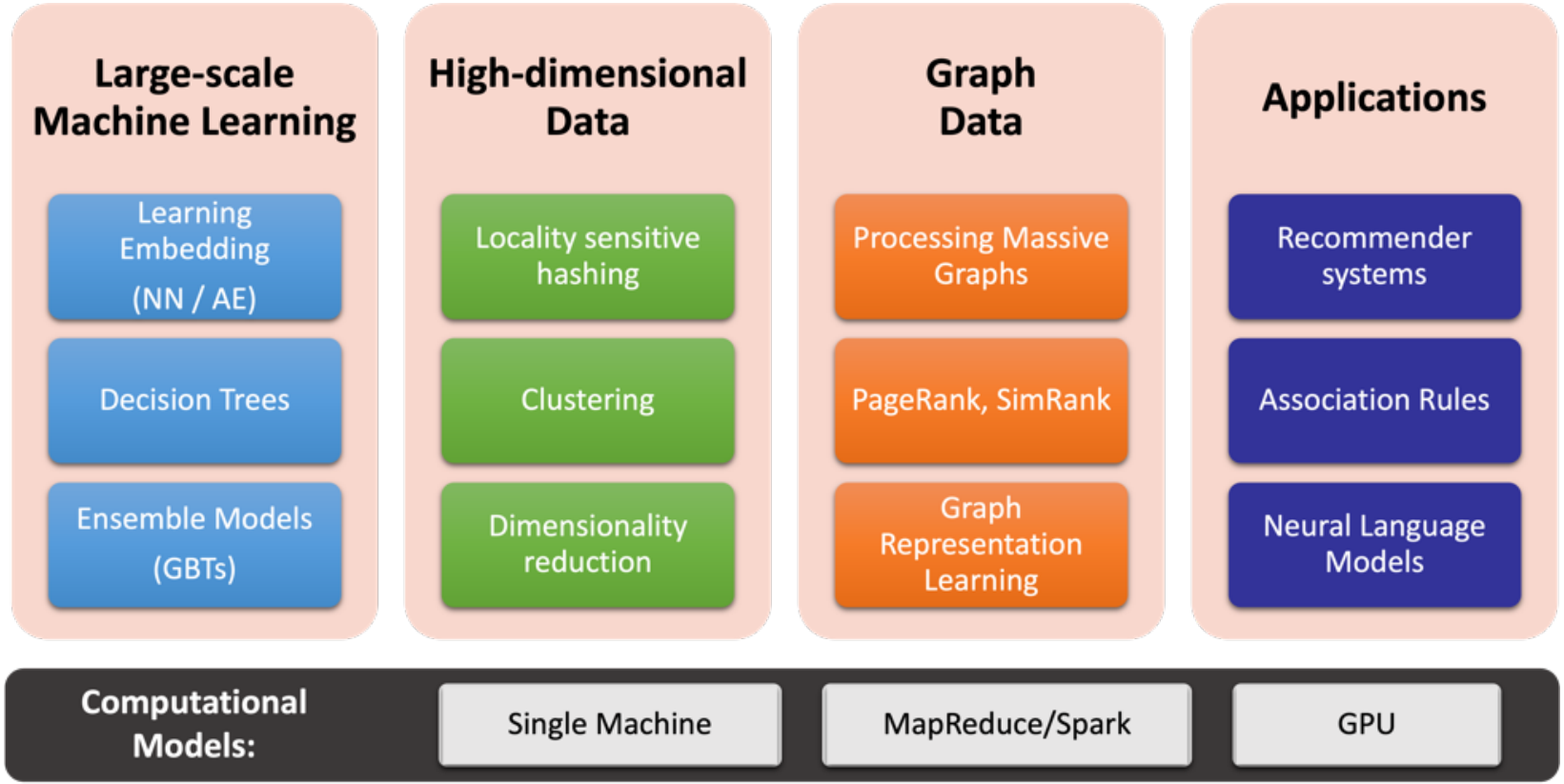
MIE524 Data Mining

Locality Sensitive Hashing

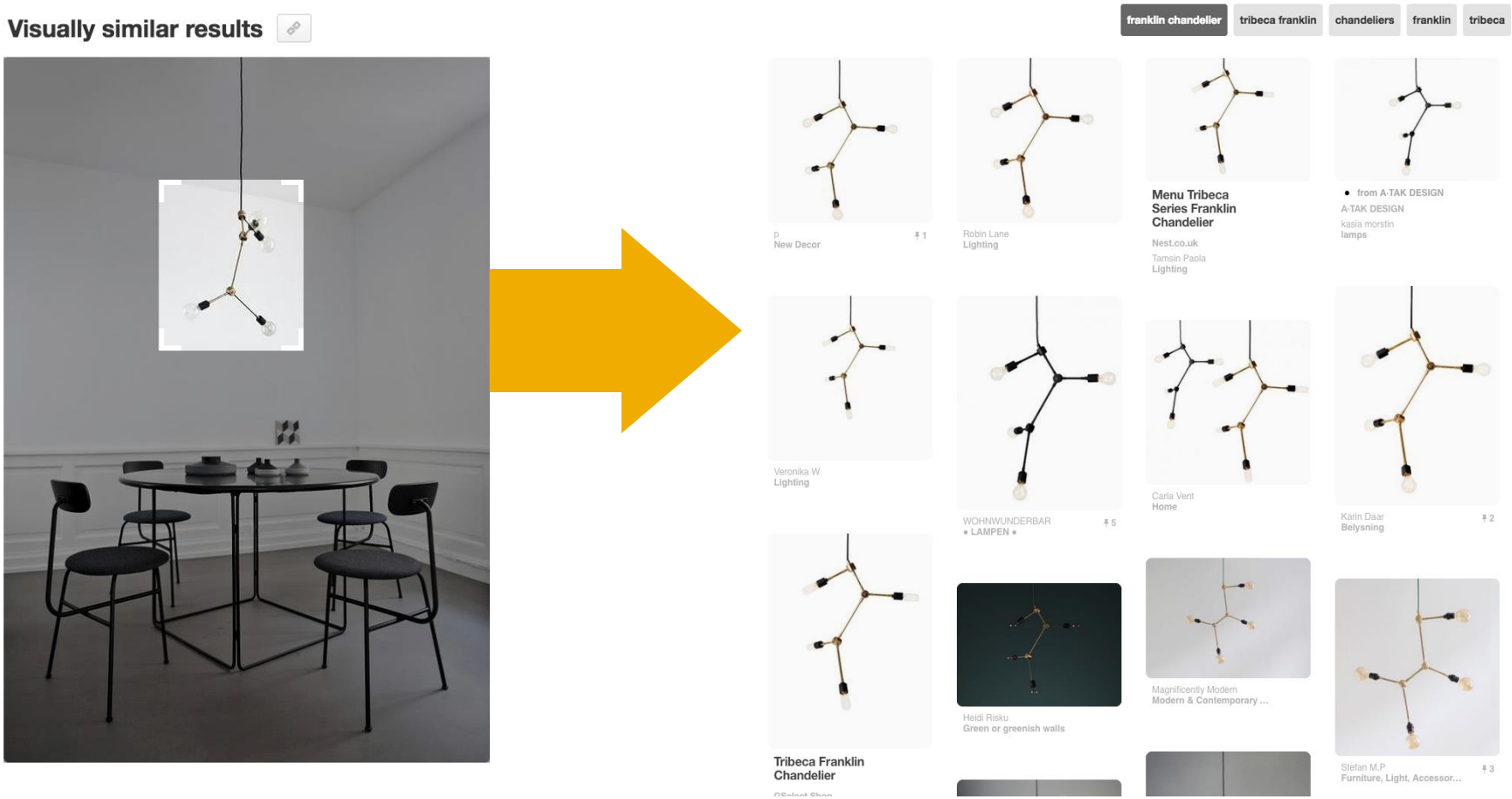
Slides Credits:
Slides from Leskovec, Rajaraman, Ullman (http://www.mmds.org), Leskovec & Ghashami

MIE524: Course Topics (Tentative)

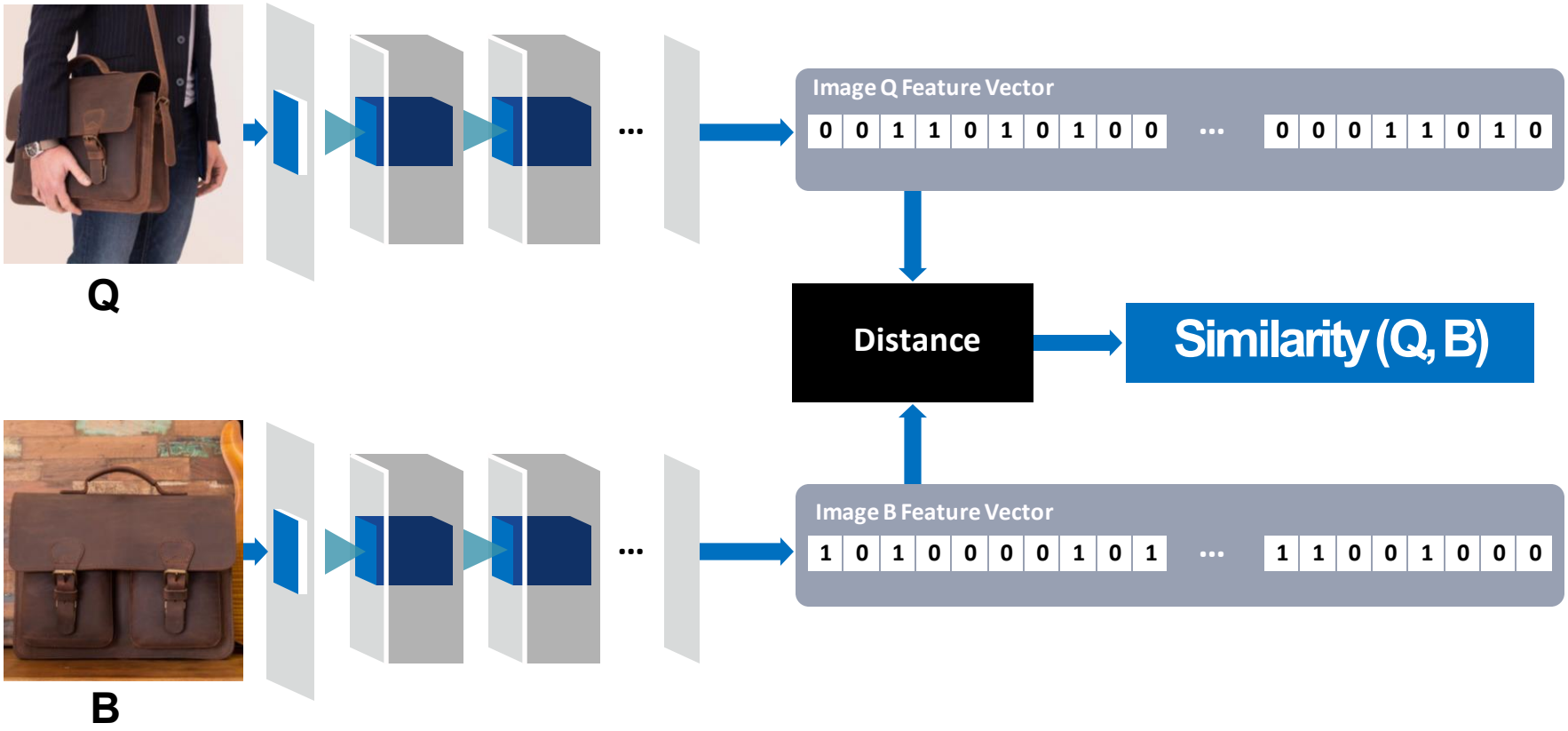


Pinterest Visual Search

Given a **query image patch**, find similar images



How does it work?



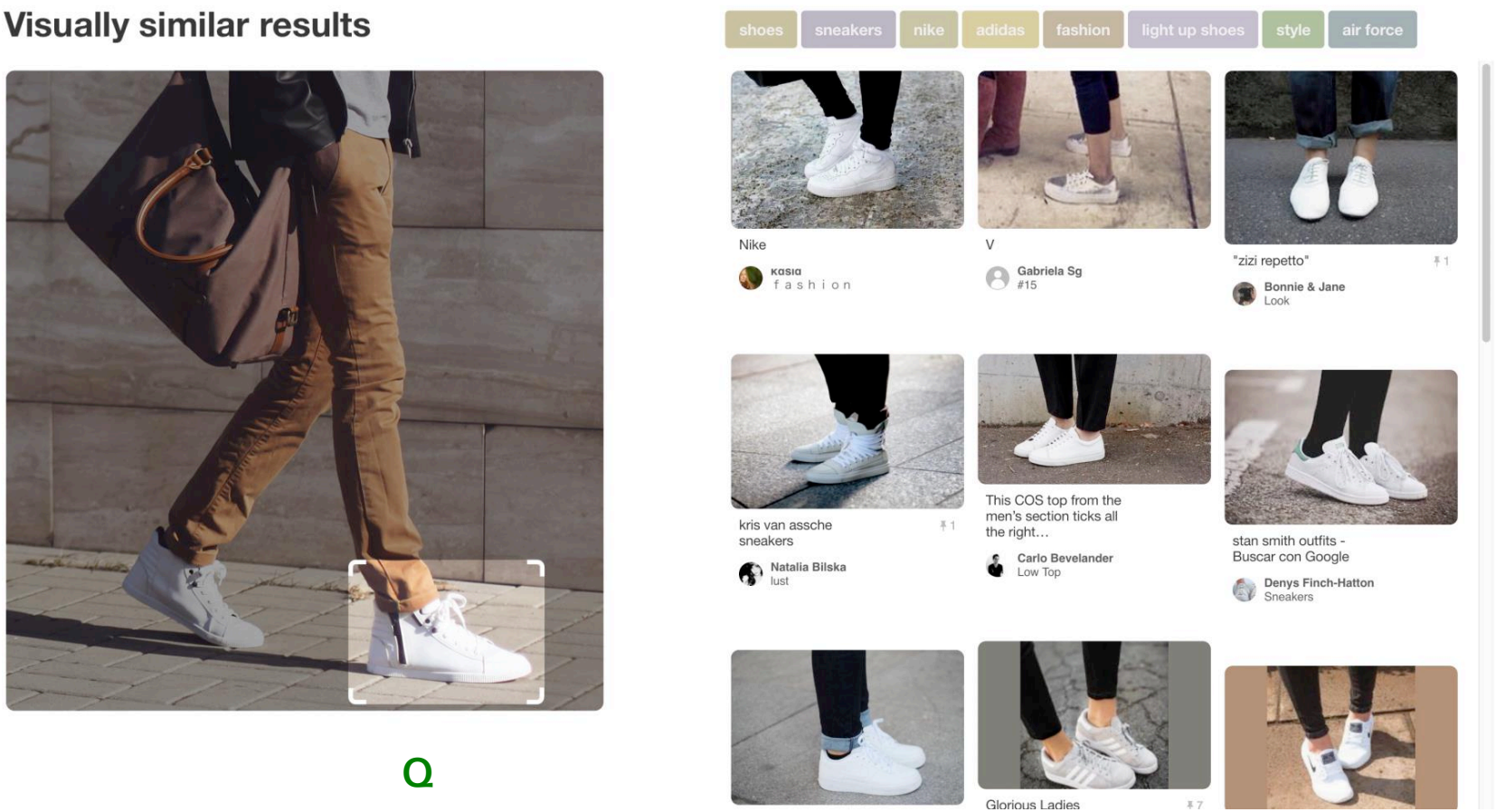
- Collect billions of images
- Determine feature vector for each image (4k dim)
- **Given a query Q, find nearest neighbors FAST**

How does it work?

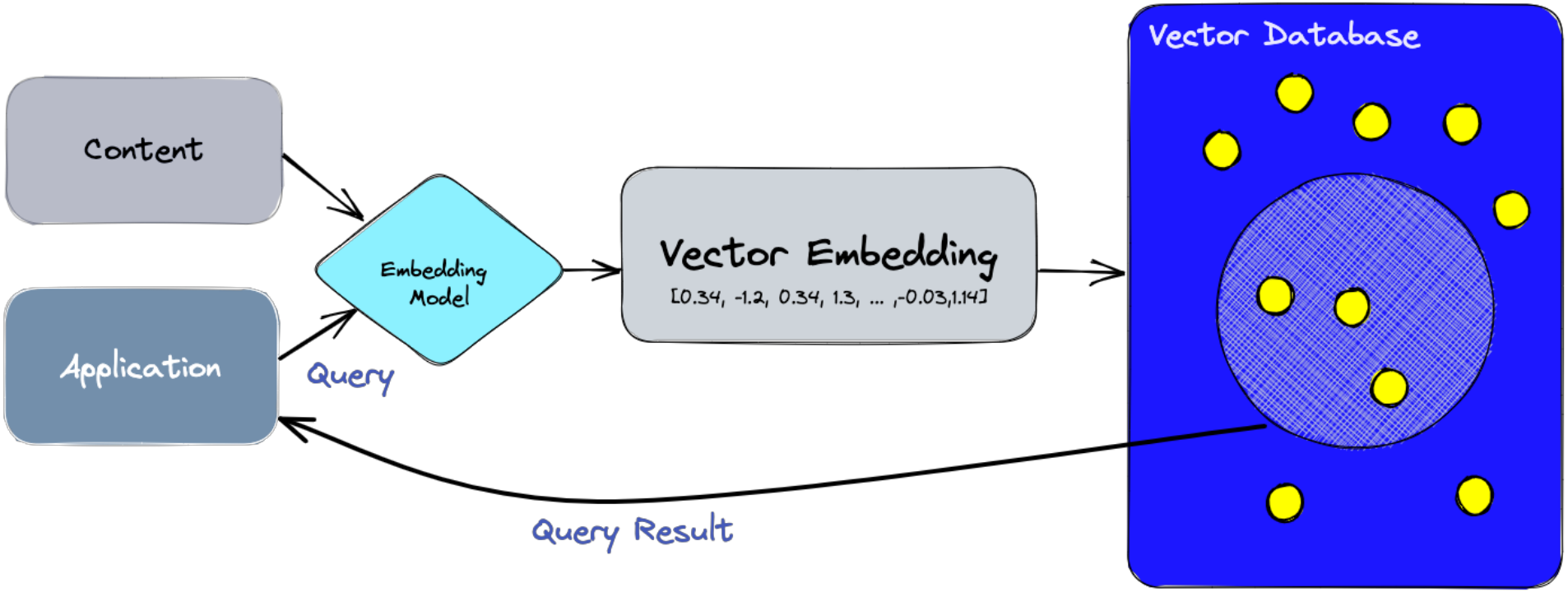
Q

Nearest neighbor query
in the embedding space

Application: Visual Search



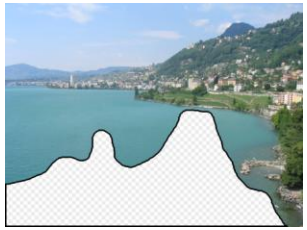
Technology Behind Vector DBs



https://www.pinecone.io/learn/vector-database

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
  - **Customers who purchased similar products**
    - Products with similar customer sets
  - **Images with similar features**
    - Image completion
  - **Recommendations and search**



Problem for today's lecture (1)

- **Given: High dimensional data points $x_1, x_2, \ldots$**
  - **For example:** Image is a long vector of pixel colors
- **And some distance function $d(x_1, x_2)$**
  - which quantifies the "distance" between x_1 and x_2

- **Goal:** Given q, find **data points x_j** that are within distance threshold $d(q, x_j) \leq s$
- **Note:** Naïve solution would take $O(N)$

  where N is the number of data points
- **MAGIC: This can be done in $O(1)$!! How??**

Problem for today's lecture (2)

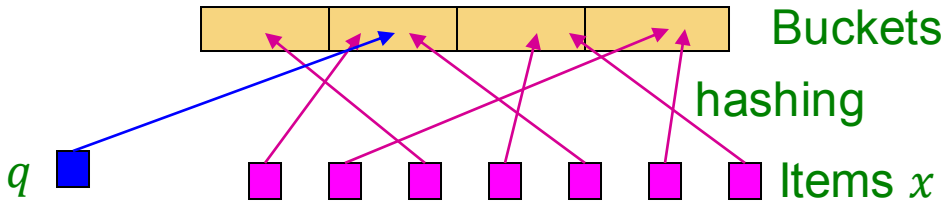
- **Given: High dimensional data points $x_1, x_2, \ldots$**
  - **For example:** Image is a long vector of pixel colors
- **And some distance function $d(x_1, x_2)$**
  - which quantifies the "distance" between x_1 and x_2

- **Goal:** Find **all pairs of data points (x_i, x_j)** that are within 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??**

Overview of LSH: The Bigfoot of CS

- LSH is really a family of related techniques
- In general, one throws items into buckets using several different "hash functions".
- You examine only those pairs of items that share a bucket for at least one of these hashings.



- **Upside:** Designed correctly, only a small fraction of points are ever examined
- **Downside:** There are *false negatives* – there might be similar items that get missed

Motivating Application: Finding Similar Documents

Motivation for Min-Hash/LSH

- **Suppose we need to find near-duplicate documents among $N = 1$ million documents**
  - Naïvely, we would have to compute **pairwise similarities** for **every pair of docs**
    - $N(N-1)/2 \approx 5*10^{11}$ 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…
- Similarly, we have a dataset of 10B documents, quickly find the document that is most similar to query document q.

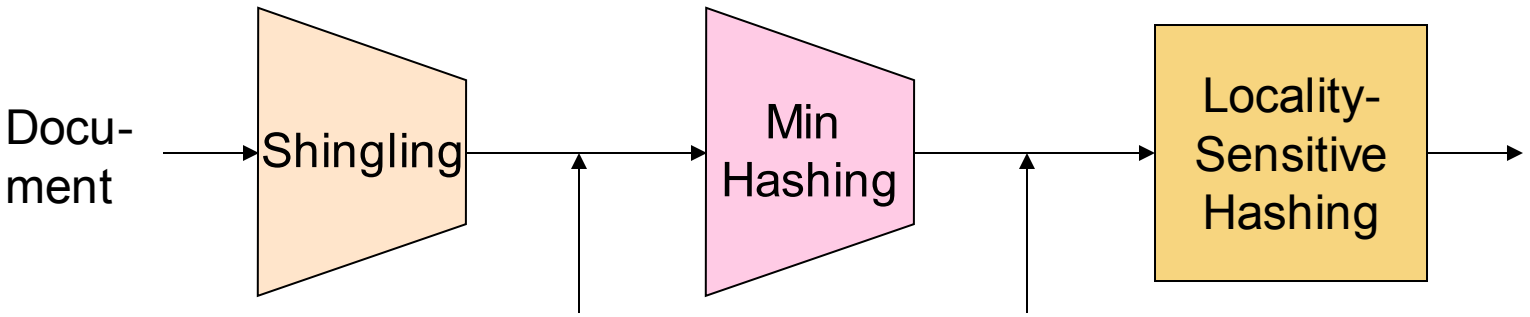
3 Essential Steps for Similar Docs

1. ***Shingling:*** Converts a document into a set representation (Boolean vector)

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!**

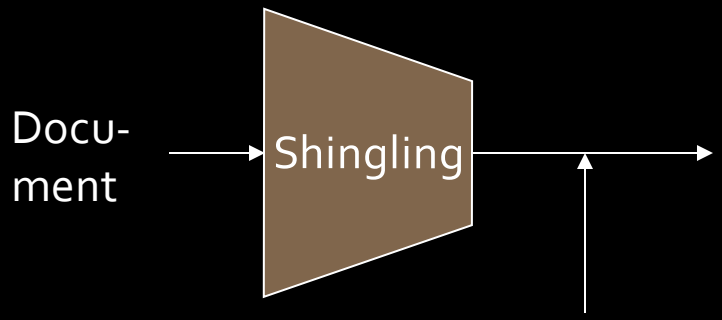
The Big Picture



The set of strings of length ***k*** that appear in the document

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 a document into a set

Documents as High-Dim Data

Step 1: *Shingling:* Converts a document into a set

- 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

- **Represent a document by the set of its *k*-shingles**

Compressing Shingles

- **Example: k=2**; document $\mathbf{D_1}$= abcab
  Set of 2-shingles: $\mathbf{S(D_1)}$ = {ab, bc, ca}

- ***k*** = 8, 9, or 10 is often used in practice

- **Benefits of shingles:**
  - Documents that are intuitively similar will have many shingles in common
  - Changing a word only affects k-shingles within distance k-1 from the word

Shingles and Similarity

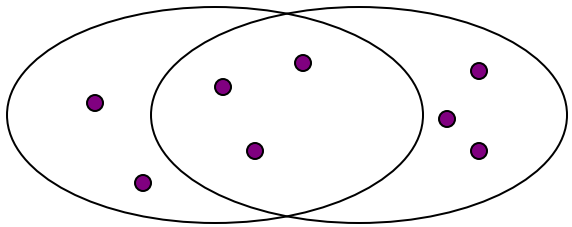
- Documents that are textually similar will have many shingles in common.
- Changing a word only affects k-shingles within distance k-1 from the word.
- Reordering paragraphs only affects the 2k shingles that cross paragraph boundaries.
- Example: k=3, “The dog which chased the cat” versus “The dog that chased the cat”.
  - Only 3-shingles replaced are g_w, _wh, whi, hic, ich, ch_, and h_c.

Similarity Metric for Shingles

- **Document D_i is represented by a set of its k-shingles $C_i = S(D_i)$**
- **A natural similarity measure is the Jaccard similarity:**

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

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

From Sets to Boolean Matrices

Encode sets using 0/1 (bit, Boolean) vectors

- **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

We don't really construct the matrix; just imagine it exists

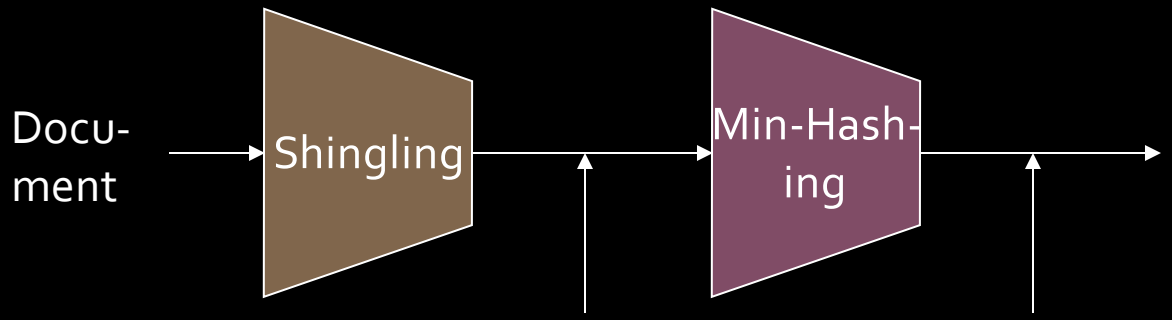
Example: Column Similarity

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

$Sim(C_1, C_2) = 2/5 = 0.4$

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**
- **Warnings:**
  - Comparing all pairs takes too much time: **Job for LSH**
    - These methods can produce false negatives, and even false positives (if the optional check is not made)



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

Min-Hashing

Step 2: *Min-Hashing:* Convert **large sets** to **short signatures**, while **preserving similarity**

Hashing Columns (Signatures)

- **Key idea:** "hash" each column ***C*** to a small *signature* ***h(C)***, such that:
  - ***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)***
- **Idea: Hash docs into buckets. Expect that "most" pairs of near duplicate docs hash into the same bucket!**

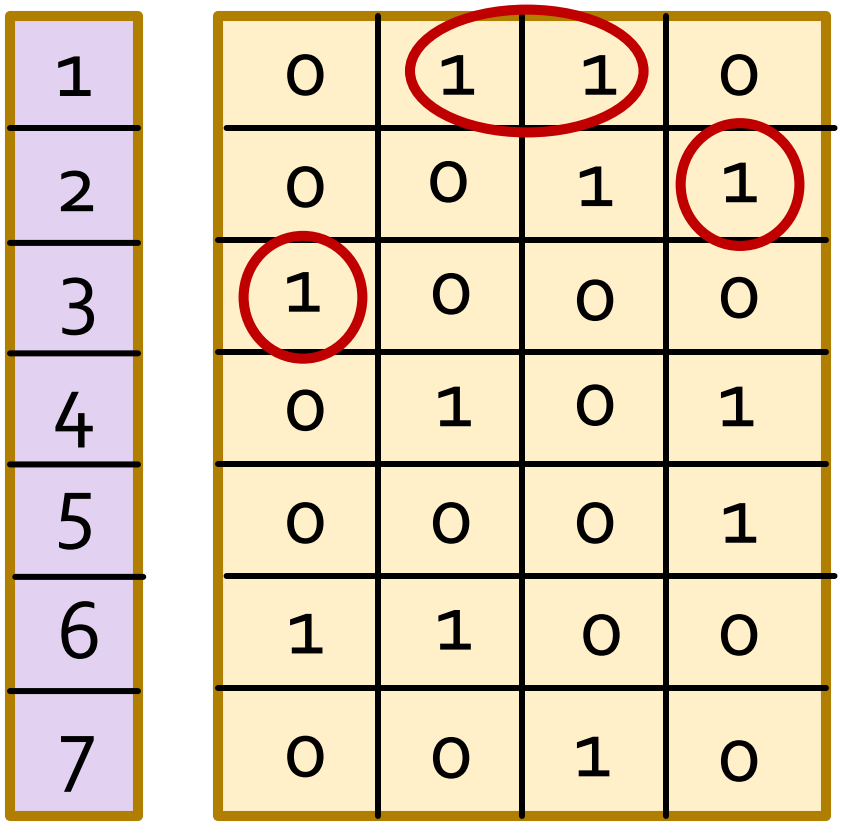
Min-Hashing: Goal

- **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: Overview

- Permute the rows of the Boolean matrix using some permutation π
  - Thought experiment – not real
- Define minhash function for this permutation π, $\mathbf{h_\pi(C)}$ = the number of the first (in the permuted order) row in which column *C* has value 1.
  - Denoted this as: $\boldsymbol{h_\pi(C) = min_\pi \pi(C)}$
- Apply, to all columns, several randomly chosen permutations π to create a signature for each column
- **Result is a signature matrix:** Columns = sets, Rows = minhash values for each permutation π

Example: Minhashing

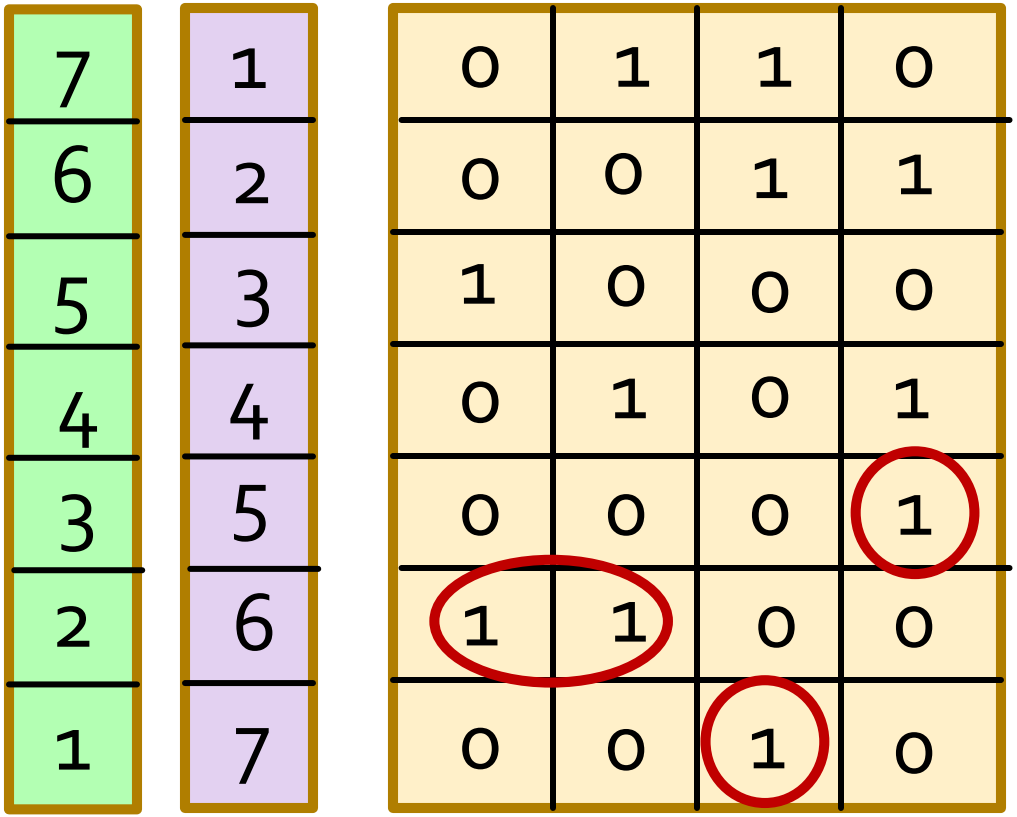


Input Matrix

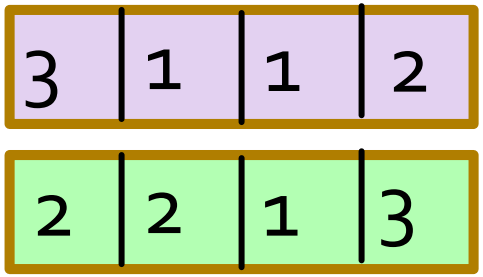
3	1	1	2

Signature Matrix

Example: Minhashing

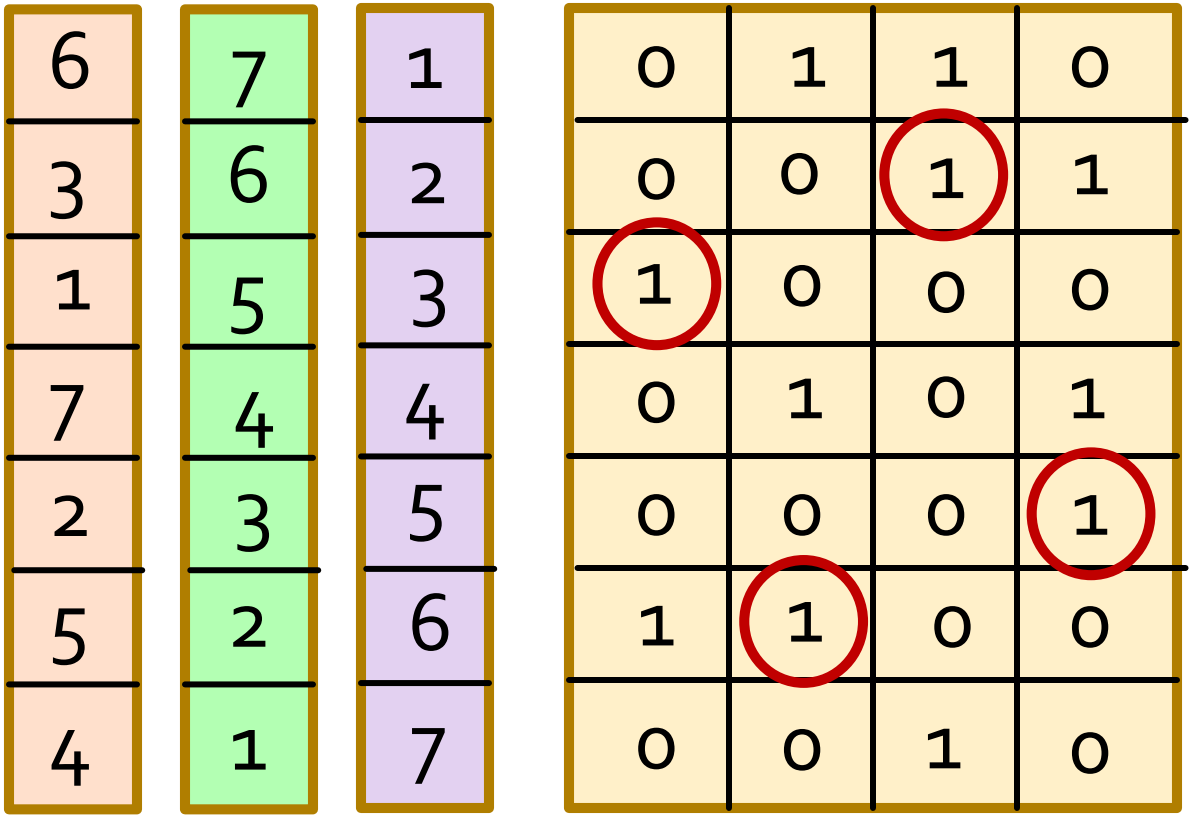


Input Matrix

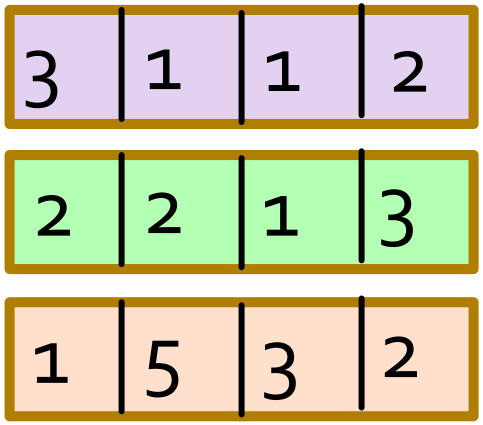


Signature Matrix

Example: Minhashing

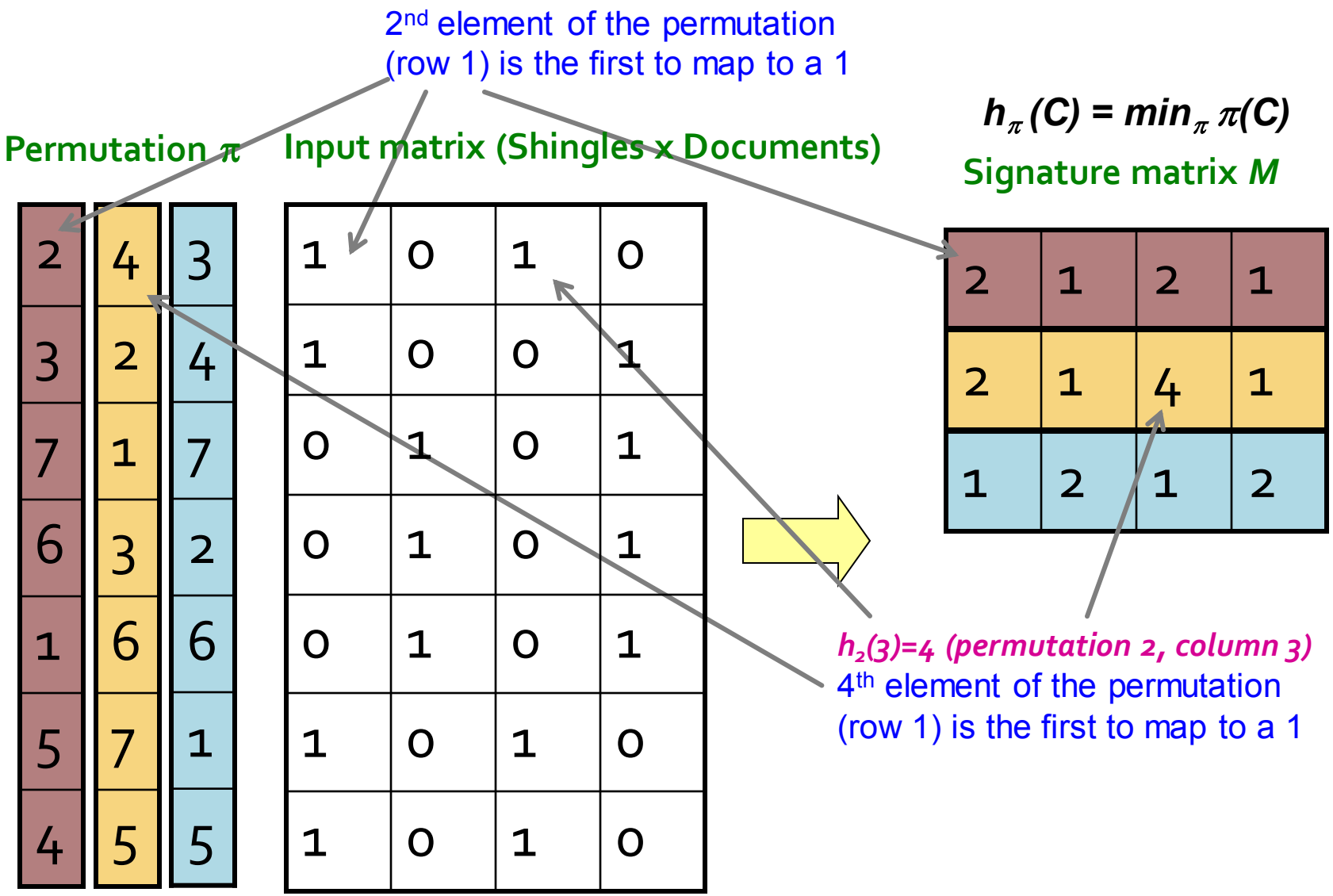


Input Matrix



Signature Matrix

Min-Hashing Example #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?
  - Already know $Sim(C_1, C_2) = a/(a+b+c)$.
  - 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.

Surprising Property

- **Given cols C_1 and C_2, rows are classified as:**

	C_1	C_2
A	1	1
B	1	0
C	0	1
D	0	0

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

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

Similarity for Signatures

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

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

- 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

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)

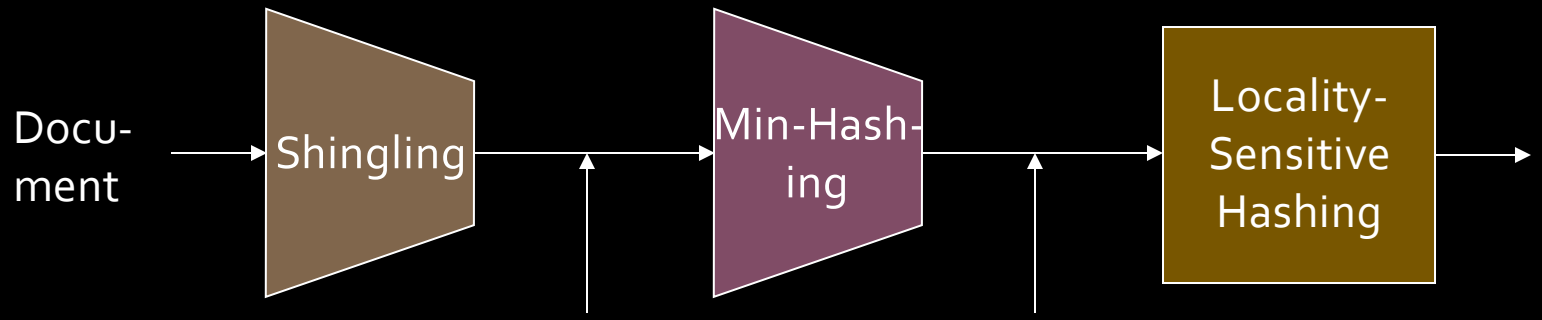
1	0	1	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0

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

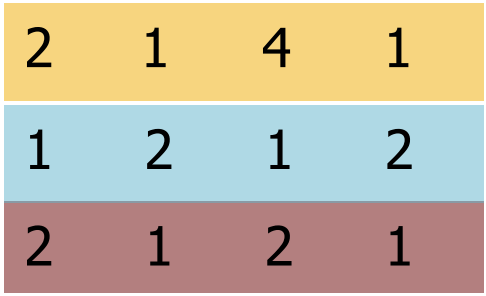


Locality Sensitive Hashing

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

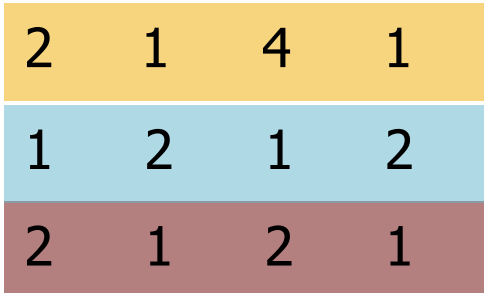
Focus on pairs of signatures likely to be from similar documents

LSH: Overview



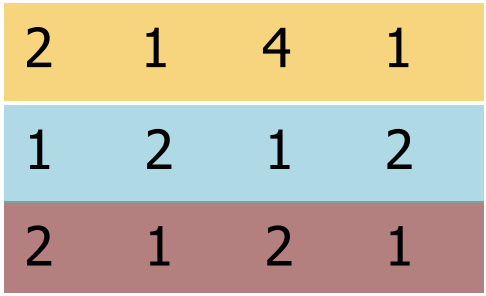
- **Goal:** Find documents with Jaccard similarity at least ***s*** (for some similarity threshold, e.g., ***s***=0.8)
- **LSH – General idea:** Use a hash function that tells whether ***x*** and ***y*** is a ***candidate pair:*** a pair of elements whose similarity must be evaluated
- **For Min-Hash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

LSH: Overview



- **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

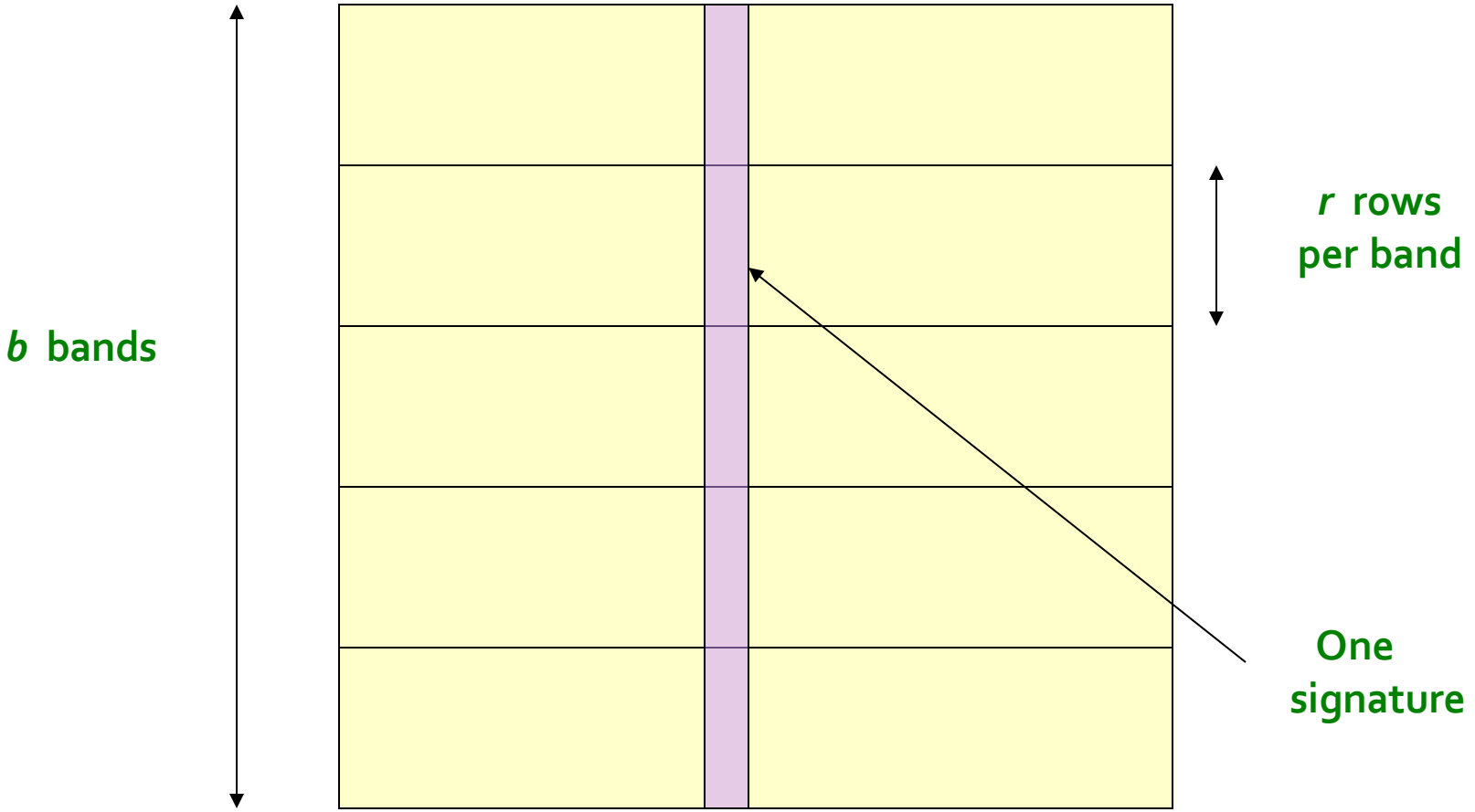
LSH for Min-Hash



- **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**

Partition *M* into *b* Bands

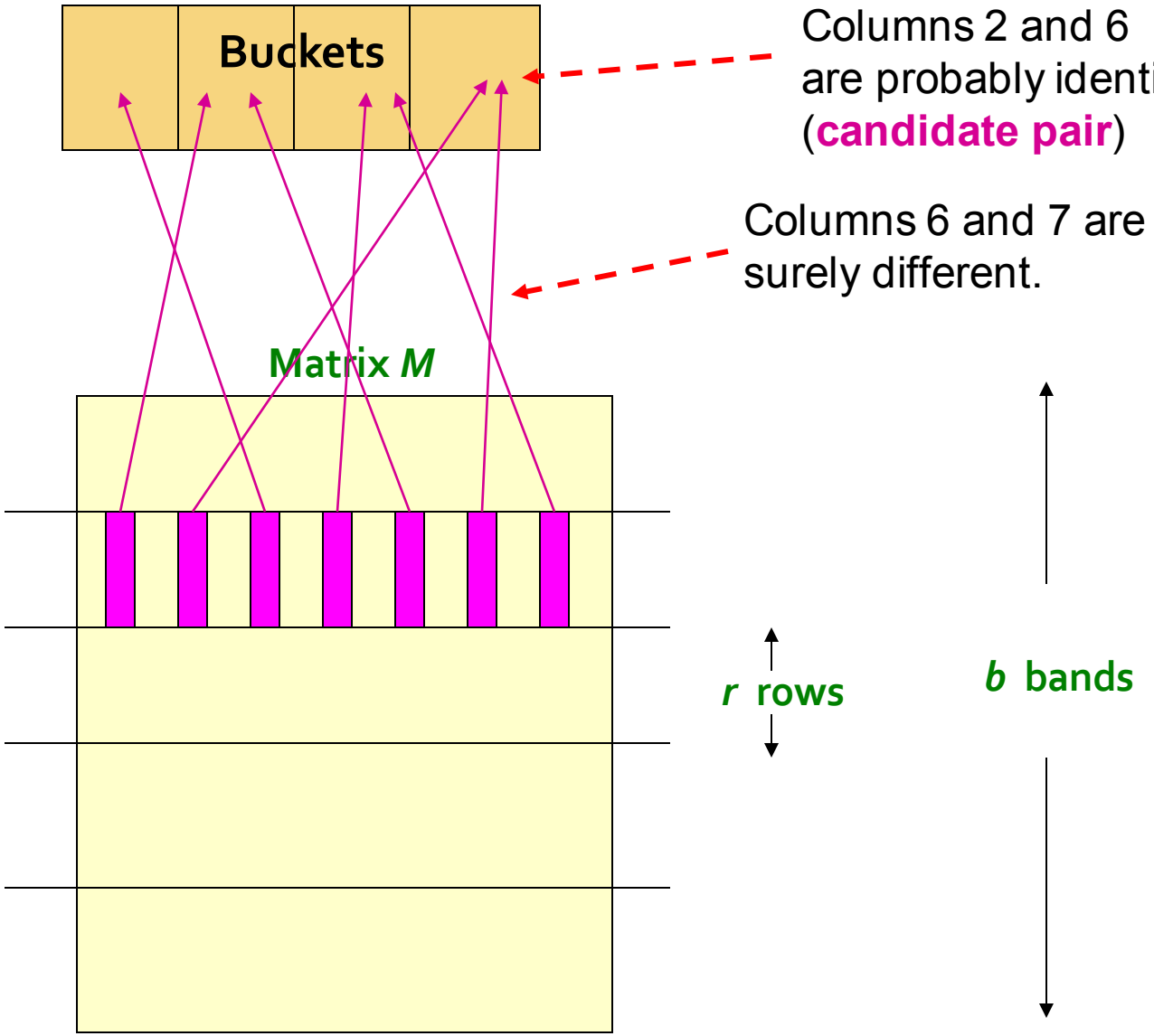
2	1	4	1
1	2	1	2
2	1	2	1



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** bands
- Tune ***b*** and ***r*** to catch most similar pairs, but few non-similar pairs

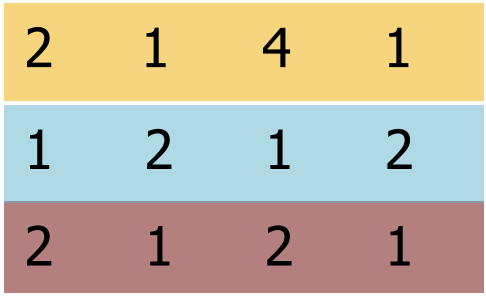
Hashing Bands



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**".

Example of Bands



Assume the following case:

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

If C_1, C_2 are 80% Similar

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

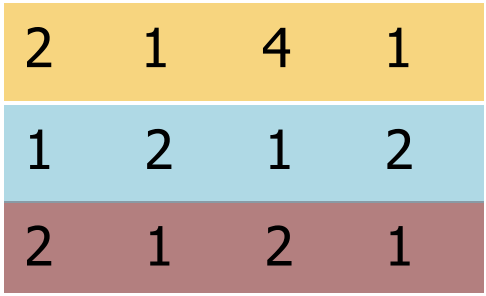
- **Find pairs of $\geq s$=0.8 similarity, set b=20, r=5**
- **Assume:** $\text{sim}(C_1, C_2) = 0.8$
  - Since $\text{sim}(C_1, C_2) \geq \mathbf{s}$, we want C_1, C_2 to be a **candidate pair**: We want them to hash to at **least 1 common bucket** (at least one band is identical)
- **Probability C_1, C_2 identical in one particular band:** $(0.8)^5 = 0.328$
- Probability C_1, C_2 are ***not*** similar in all 20 bands: $(1\text{-}0.328)^{20} = 0.00035$
  - 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**

If C_1, C_2 are 30% Similar

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

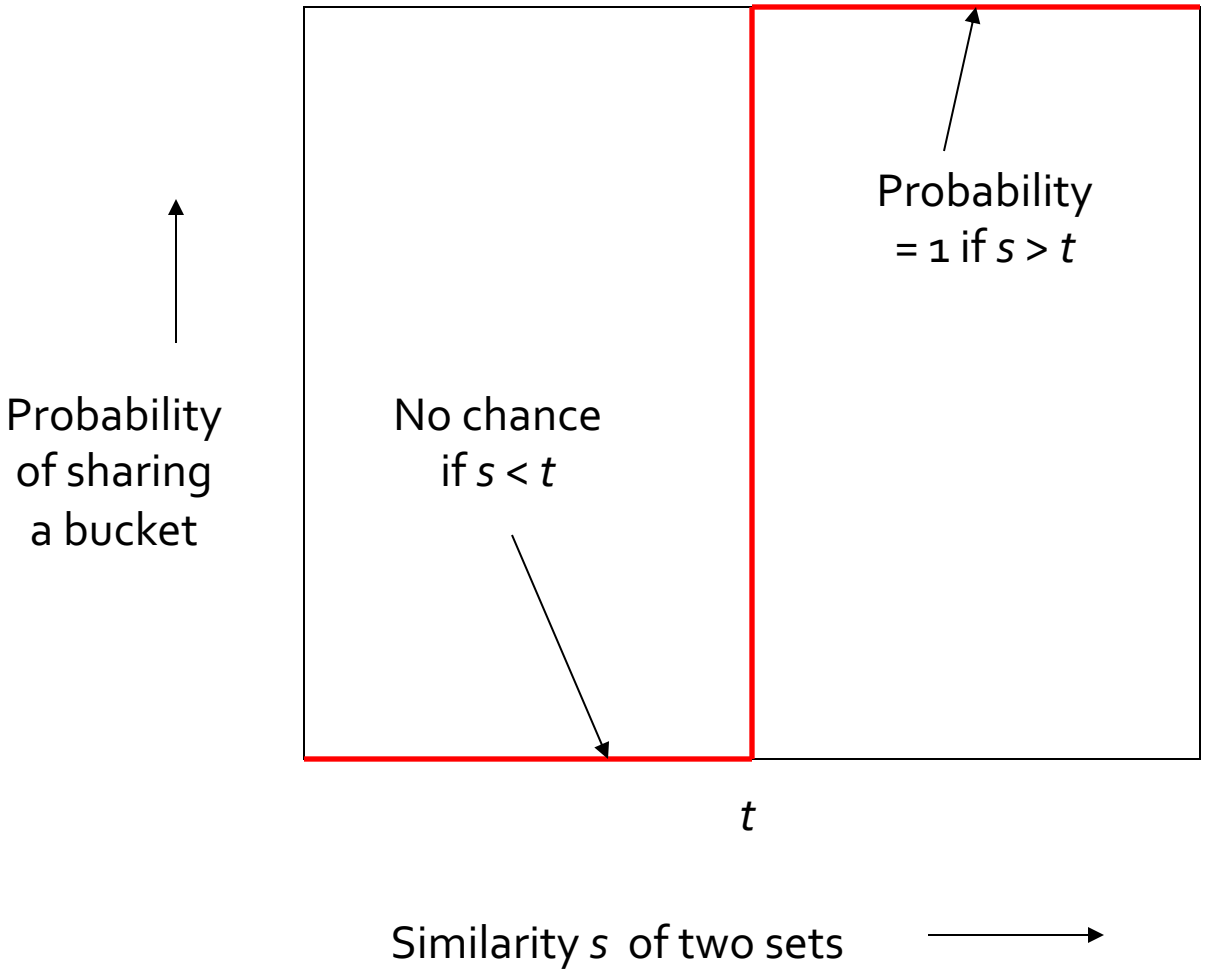
- **Find pairs of $\geq s$=0.8 similarity, set b=20, r=5**
- **Assume:** $\text{sim}(C_1, C_2) = 0.3$
  - Since $\text{sim}(C_1, C_2) < s$ we want C_1, C_2 to hash to **NO common buckets** (all bands should be different)
- **Probability C_1, C_2 identical in one particular band:** $(0.3)^5 = 0.00243$
- 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

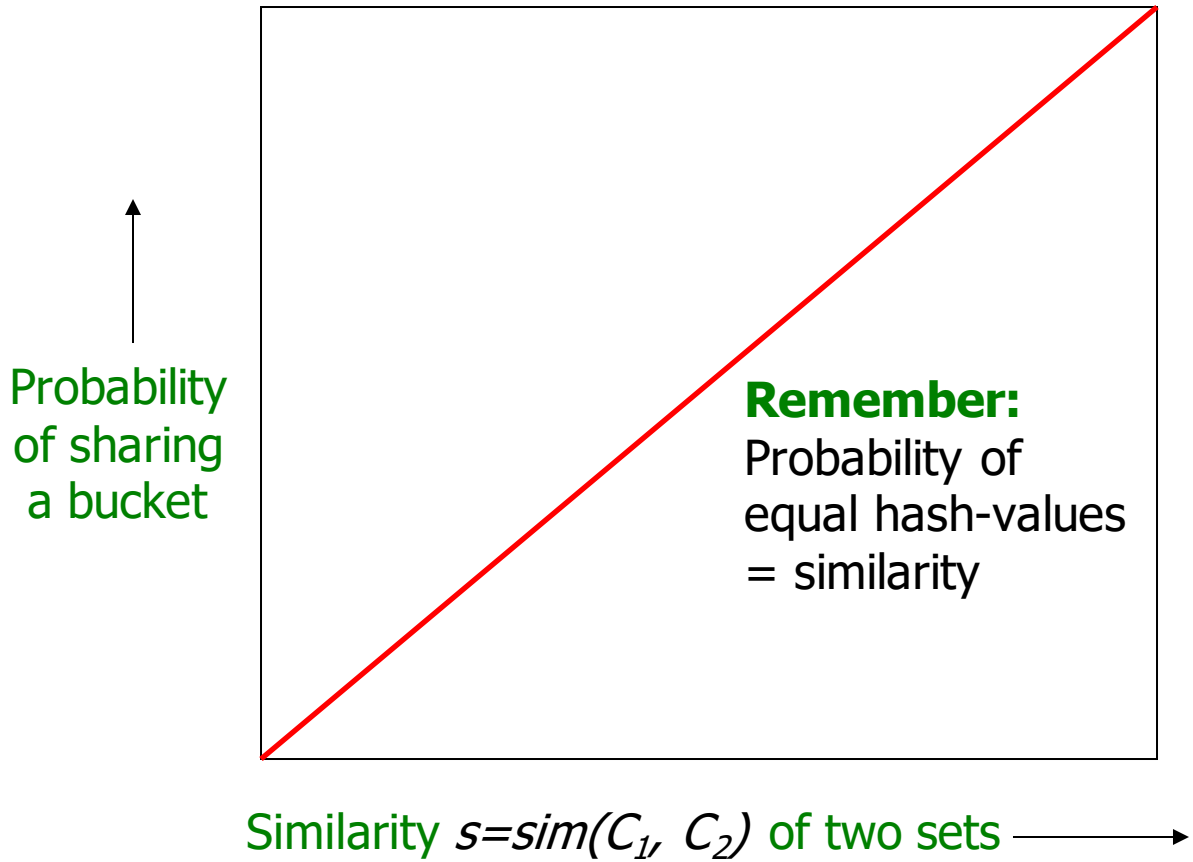


- **Pick:**
  - The number of Min-Hashes (rows of ***M***)
  - The number of bands ***b***, and
  - The number of rows ***r*** per band to balance false positives/negatives
    - Note, M=b*r
- **Example:** If we had only 10 bands of 10 rows, the number of false positives would go down, but the number of false negatives would go up

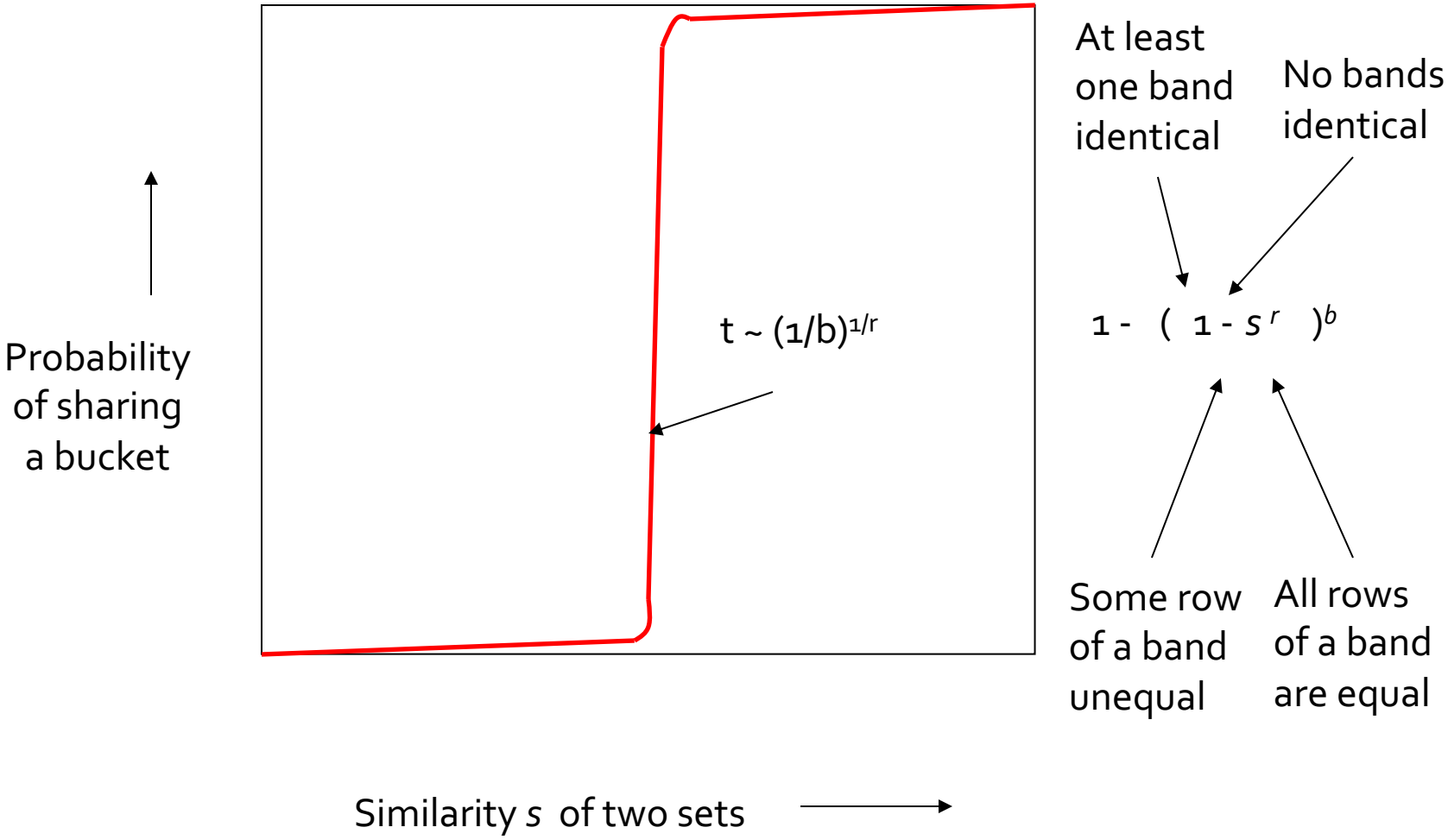
Analysis of LSH – What We Want



What 1 Band of 1 Row Gives You



What *b* Bands of *r* Rows Gives You



b bands, r rows/band

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

$$1 - (1 - s^r)^b$$

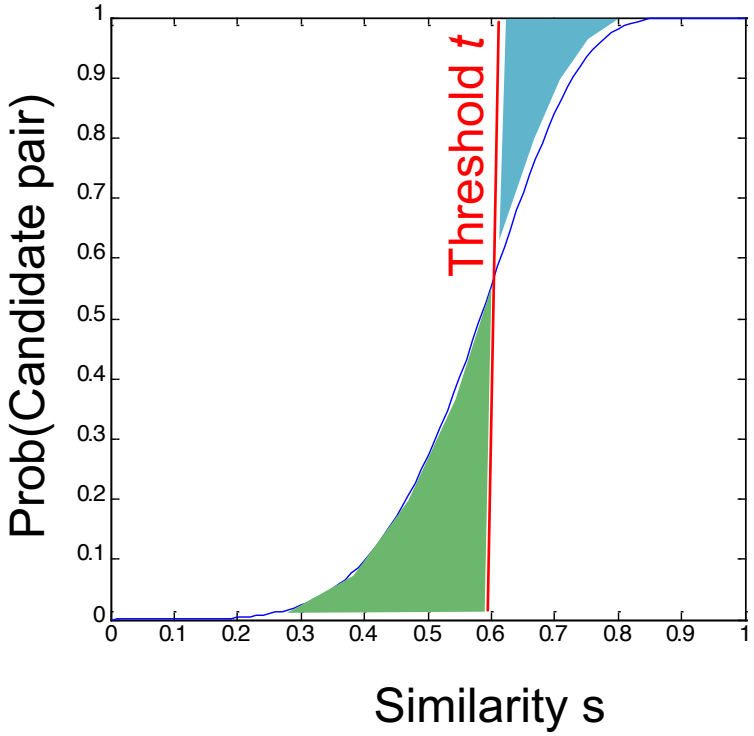
Example: $b = 20$; $r = 5$

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

s	$1-(1-s^r)^b$
0.2	0.006
0.3	0.047
0.4	0.186
0.5	0.470
0.6	0.802
0.7	0.975
0.8	0.9996

Picking *r* and *b*: The S-curve

- **Picking *r* and *b* to get desired performance**
  - **50 hash-functions** (***r*** *= 5,* ***b*** *= 10*)

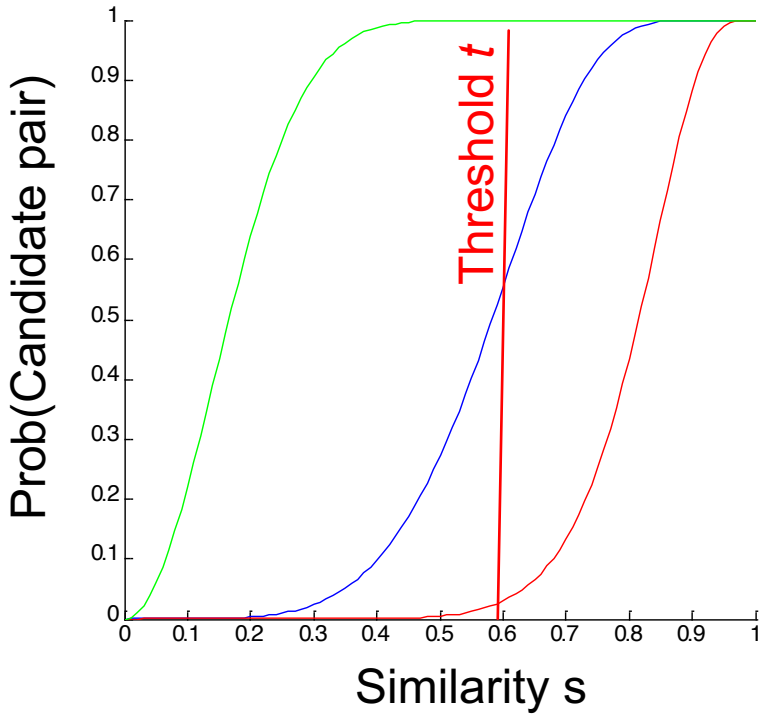


Blue area *X*: False Negative rate
These are pairs with ***sim* > *t*** but the ***X*** fraction won't share a band and then will **never become candidates.** This means we will never consider these pairs for (slow/exact) similarity calculation!

Green area Y: False Positive rate
These are pairs with ***sim* < *t*** but we will consider them as candidates. This is not too bad, we will consider them for (slow/exact) similarity computation and discard them.

Picking *r* and *b*: The S-curve

- **Picking *r* and *b* to get desired performance**
  - **50 hash-functions (*r* * *b* = *50*)**



r=2, b=25
r=5, b=10
r=10, b=5

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 set representation
- **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**