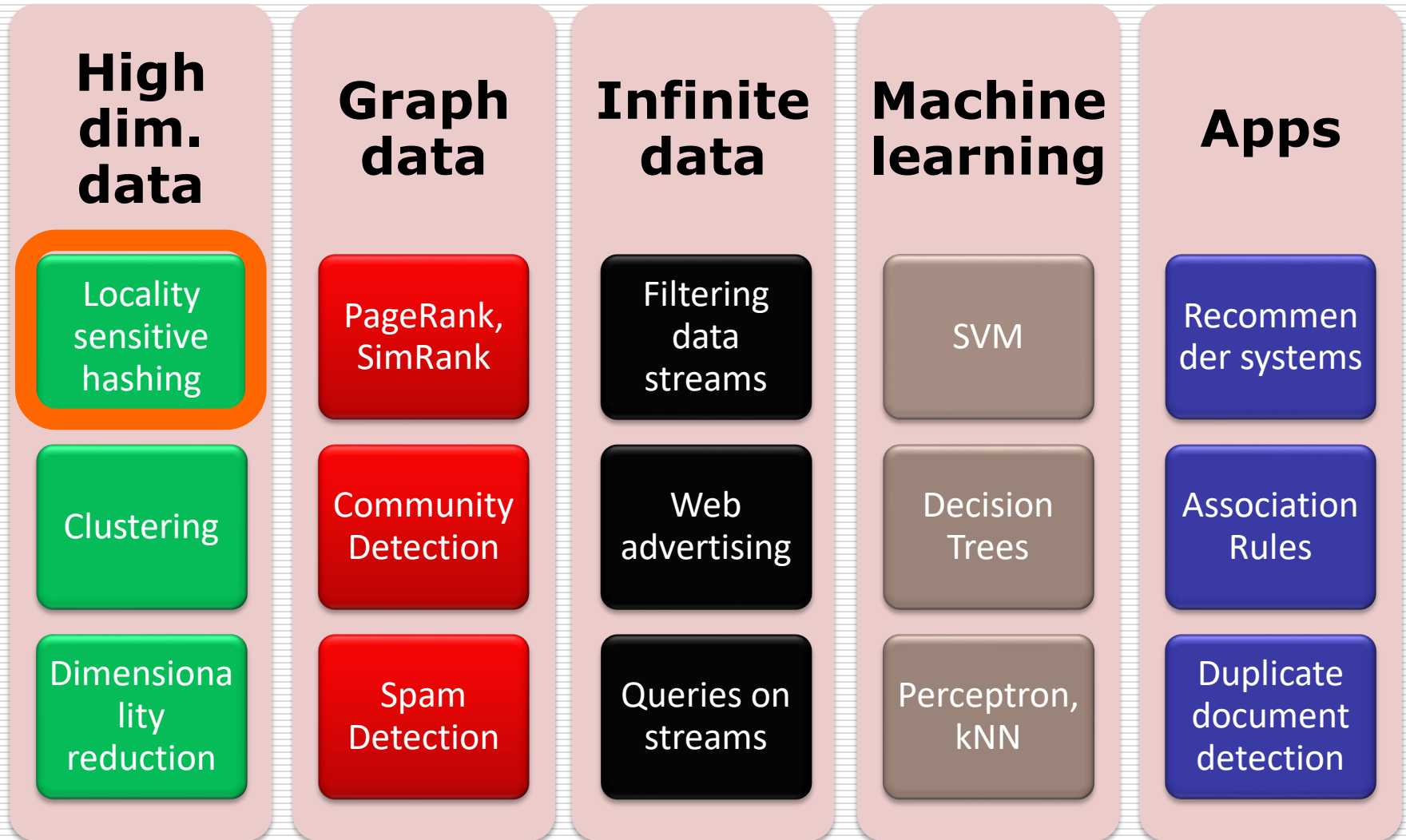


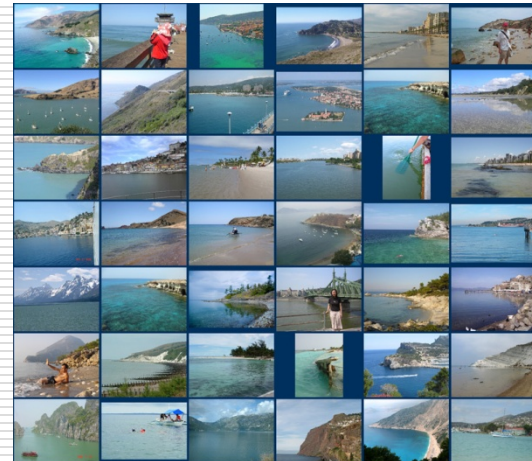
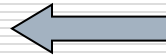
大数据计算及应用(三)

Finding Similar Items: Locality Sensitive Hashing

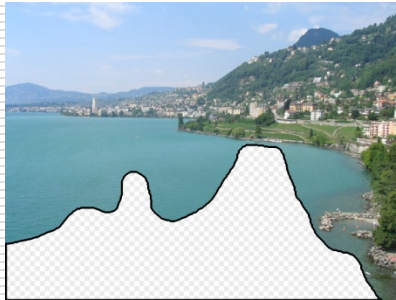
Agenda



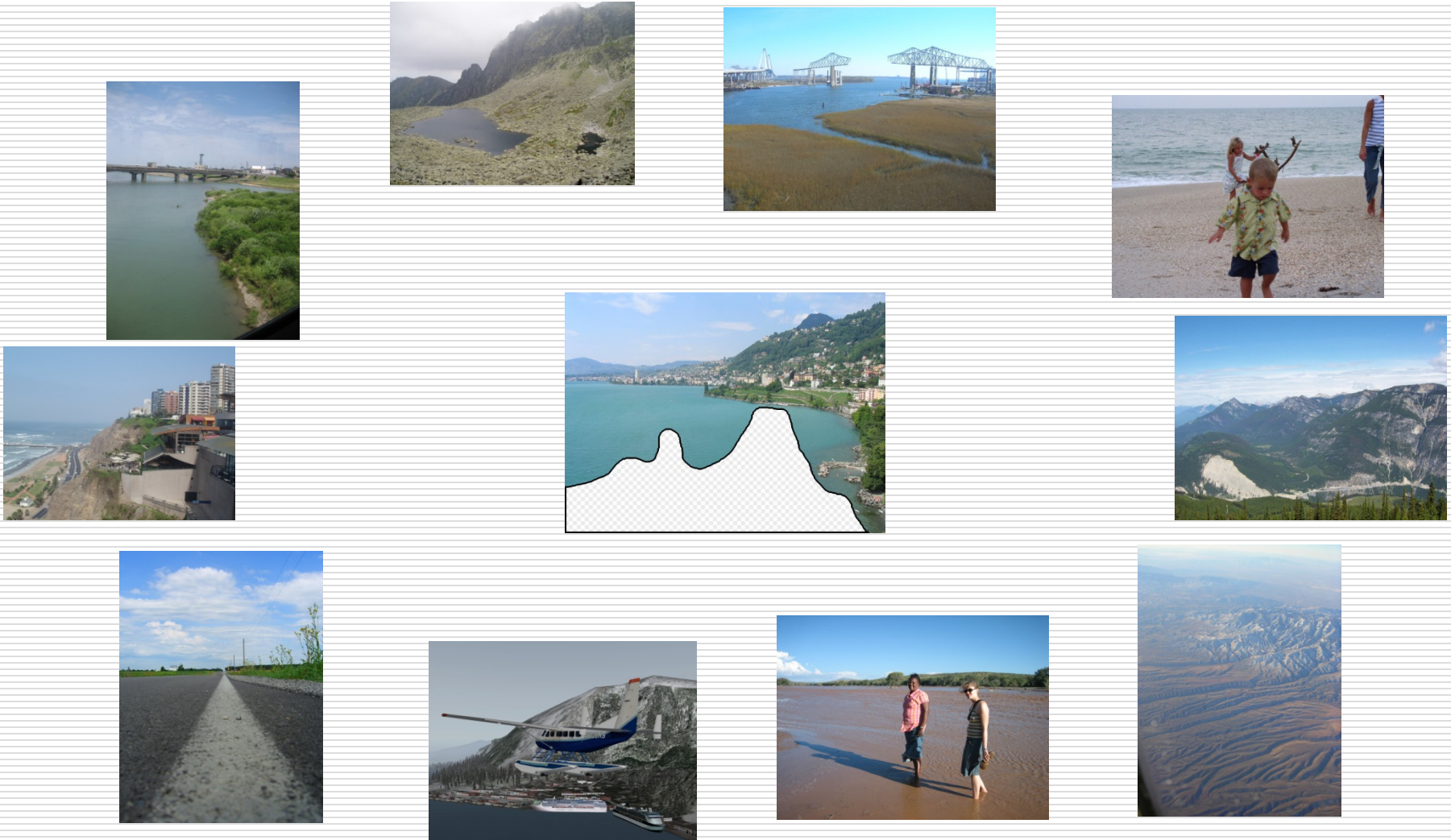
Scene Completion Problem



Scene Completion Problem

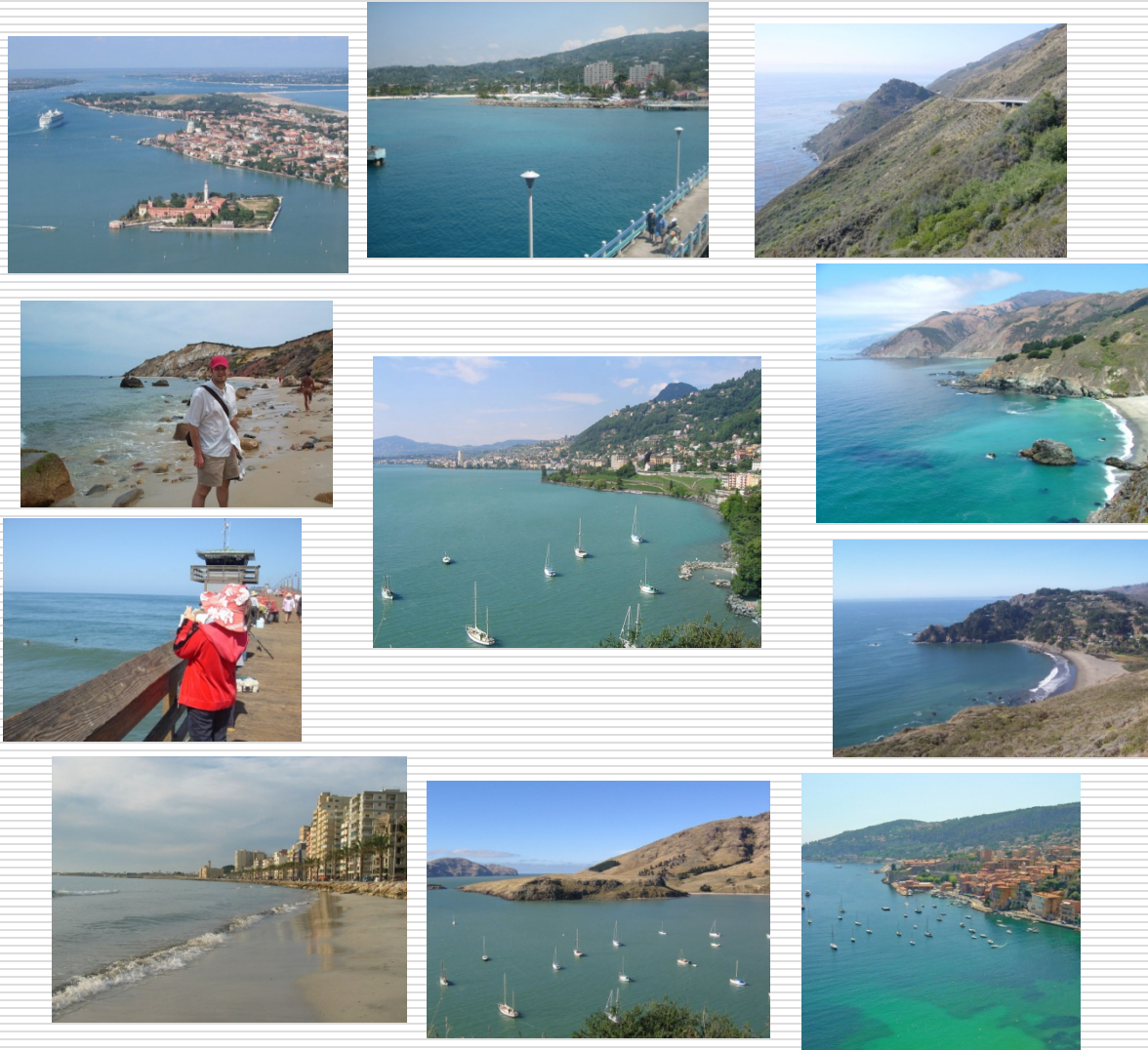


Scene Completion Problem



10 nearest neighbors from a collection of 2.3M images

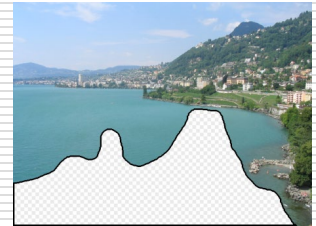
Scene Completion Problem



10 nearest neighbors from a collection of 2.3M 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 x_1, x_2, \dots

■ For example: Image is a long vector of pixel colors

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \rightarrow [1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0]$$

<i>img1</i>	<i>img2</i>
1	1
0	1
1	1
0	1
0	0
1	1
0	1
1	1
0	0

Is *img1* similar to *img2*?

Problem for Today's Lecture

□ Given: High dimensional data points x_1, x_2, \dots

■ Another example:

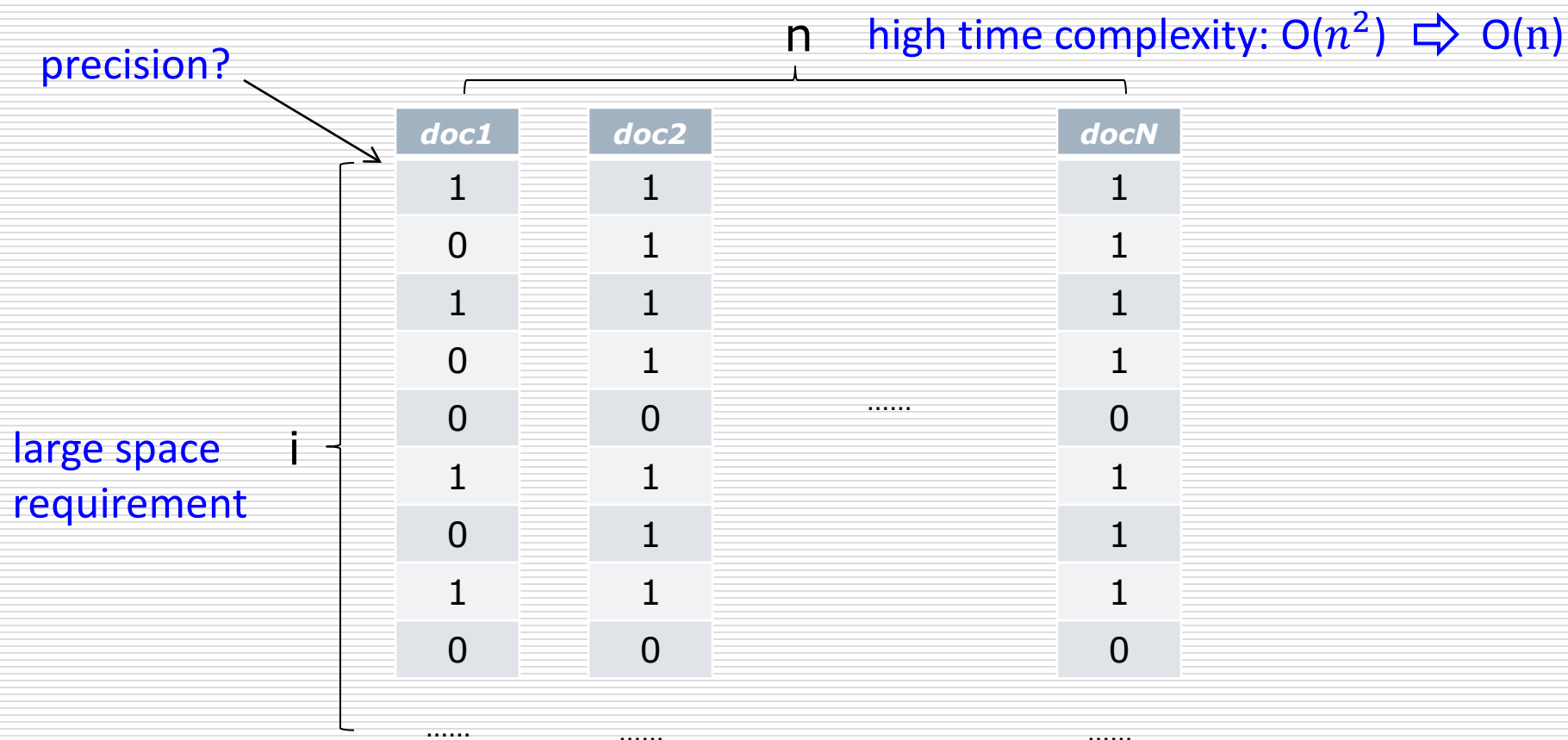
Document is a long vector of words existence

	<i>doc1</i>	<i>doc2</i>
angle	1	1
before	0	1
country	1	1
end	0	1
food	0	0
good	1	1
house	0	1
live	1	1
use	0	0

Is *doc1* similar to *doc2*?

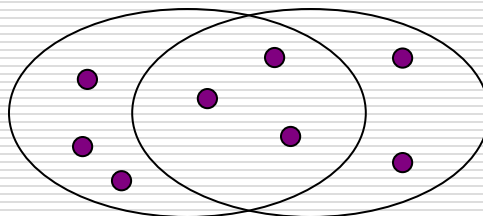
Problem for Today's Lecture

- Goal: Find **all pairs of documents** (x_i, x_j) that are within some distance threshold $d(x_i, x_j) \leq s$



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:
$$\text{sim}(C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$$
 - **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

Encoding Sets as Bit Vectors

- 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: $d(C_1, C_2) = 1 - (\text{Jaccard similarity}) = 1/4$

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
 - Documents are so large or so many that they cannot fit in main memory
 - Too many documents to compare all pairs

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!

Problem for Today's Lecture

precision? Shingling

n

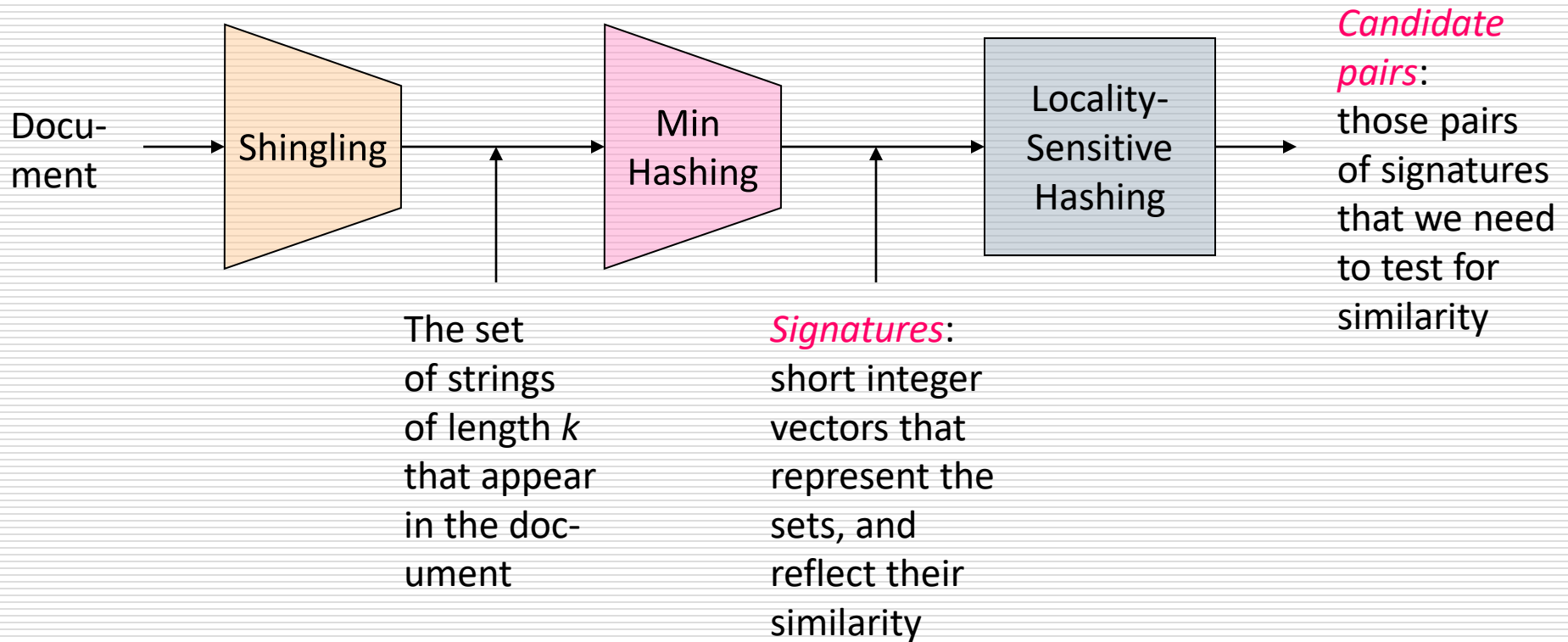
high time complexity: $O(n^2) \Rightarrow O(n)$

LSH

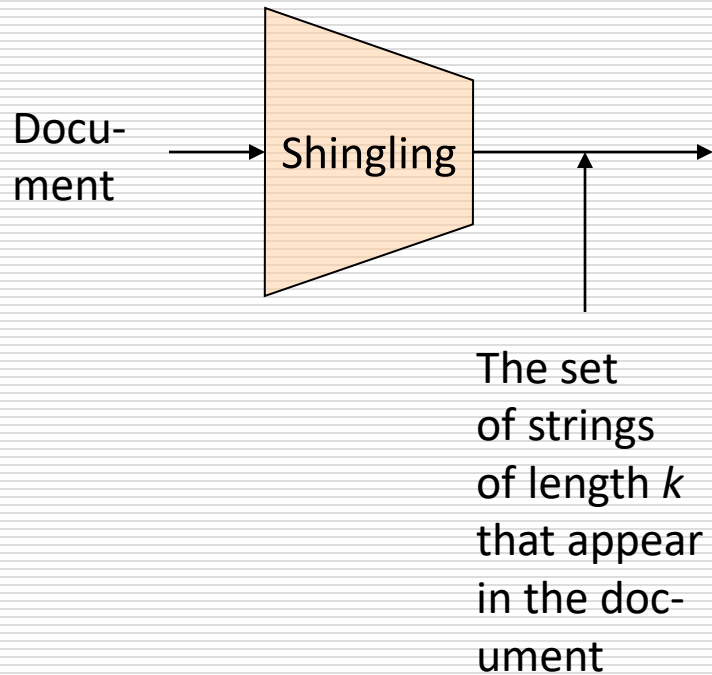
large space
requirement:
MinHash

doc1	doc2	docN
1	1		1
0	1		1
1	1		1
0	1		1
0	0	0
1	1		1
0	1		1
1	1		1
0	0		0
.....

The Big Picture



The Big Picture



□ 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*!

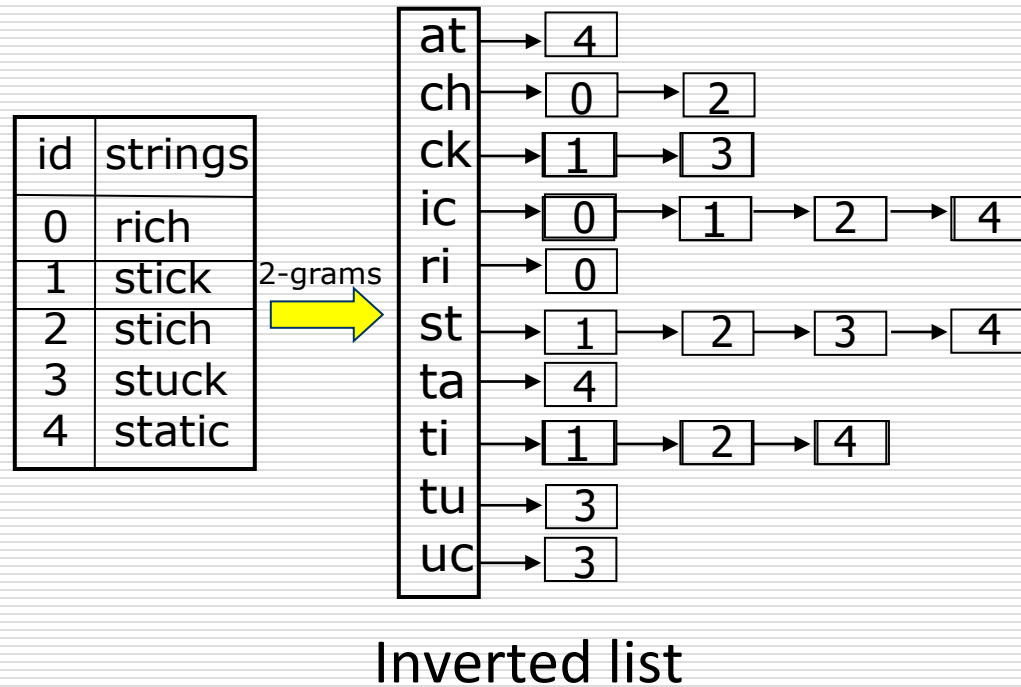
Define: Shingles (Grams)

- 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*: $k=2$; string $S_1 = \text{abcaab}$
Set of 2-shingles: $S(S_1) = \{\text{ab}, \text{bc}, \text{ca}\}$

Define: Shingles (Grams)

- Application: similar string search

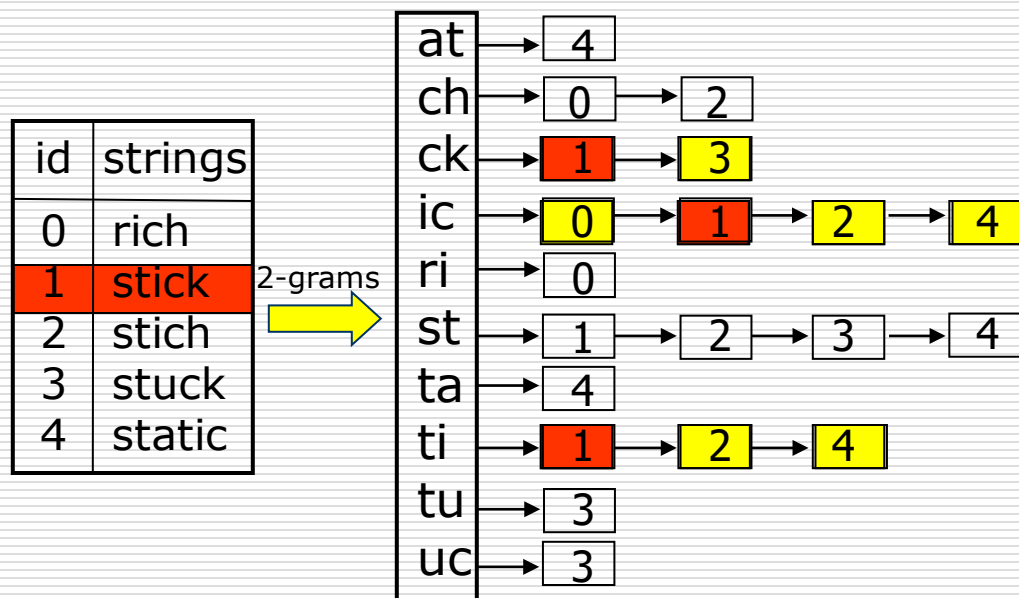


Define: Shingles (Grams)

- Application: similar string search

Query: "shtick"

sh ht ti ic ck



Inverted list

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 $D_1 = \text{ab cab}$
Set of 2-shingles: $S(D_1) = \{\text{ab}, \text{bc}, \text{ca}\}$
Hash the singles: $h(D_1) = \{1, 5, 7\}$

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

	<i>doc1</i>	<i>doc2</i>
go out	1	1
like travel	0	1
Every day	1	1

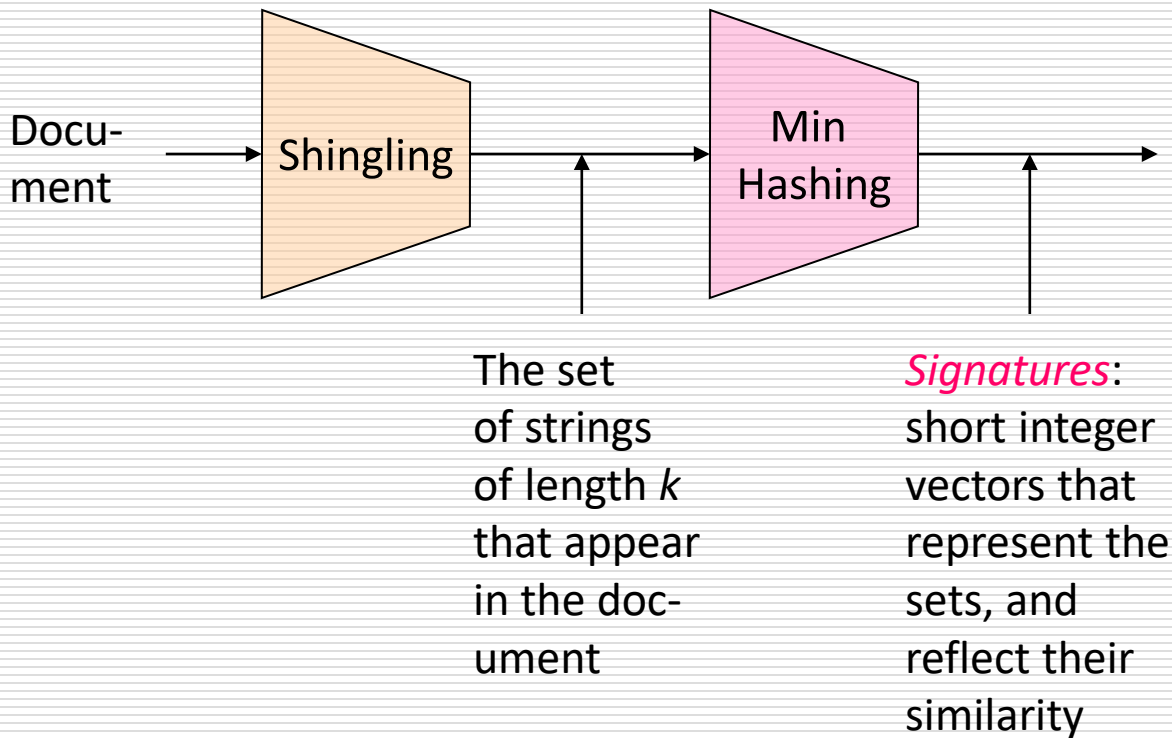
.....

.....

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

The Big Picture

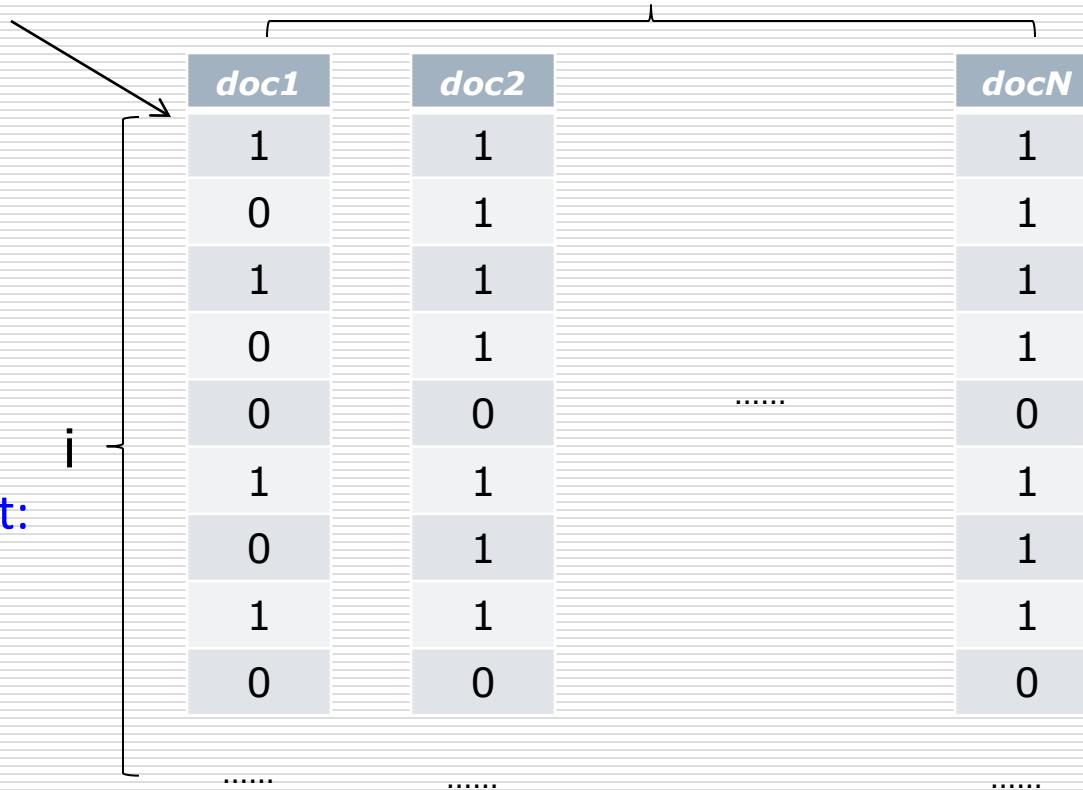


- Step 2: *Minhashing*: Convert large sets to short signatures, while preserving similarity

Problem for Today's Lecture

precision? Shingling

large space
requirement:
MinHash



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
		C_1	C_2		

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

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. $h(C_1) \neq h(C_2)$

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** 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:

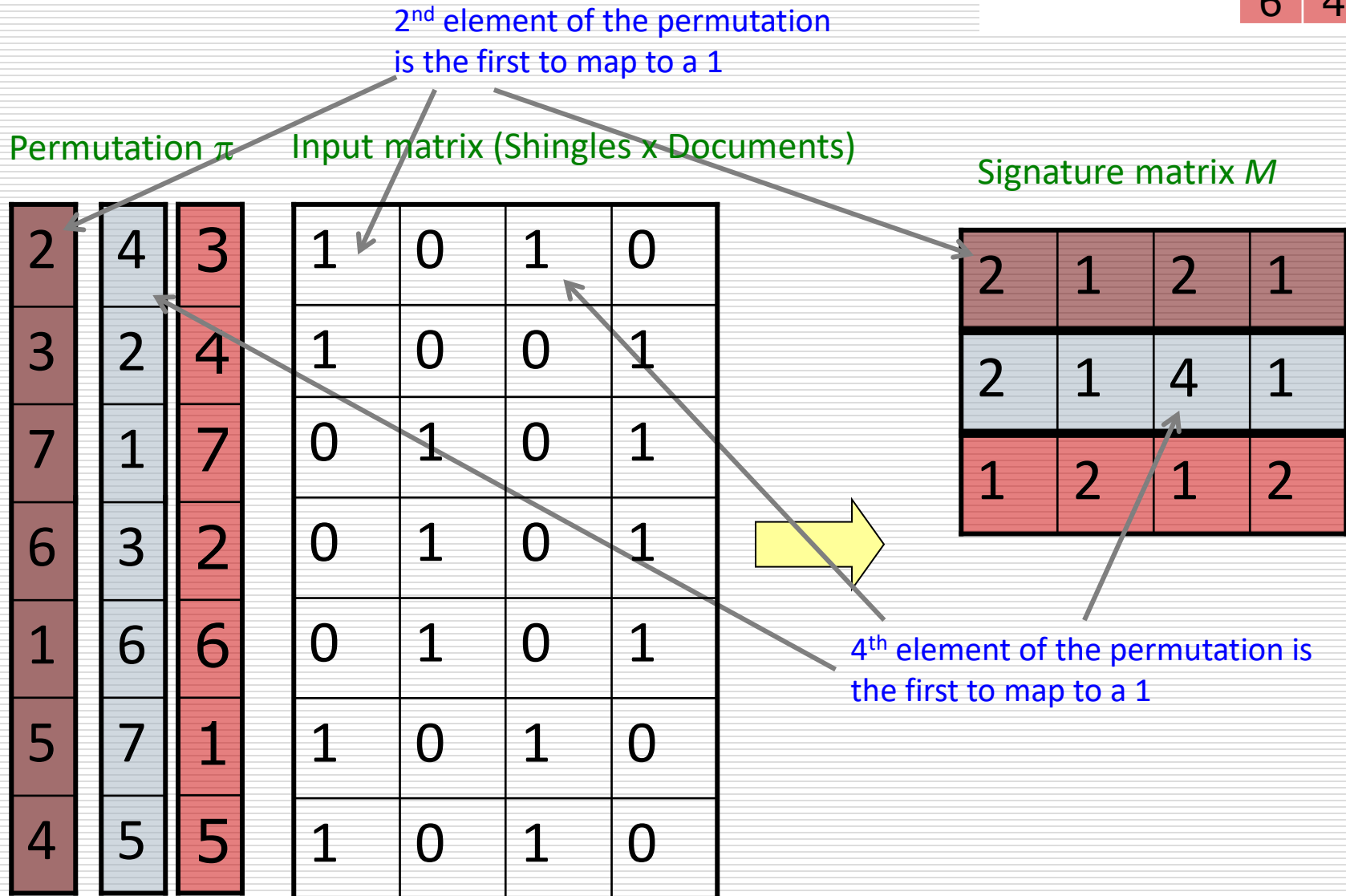
$$h_{\pi}(C) = \min_{\pi} \pi(C)$$

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

Min-Hashing Example

Note: Another (equivalent) way is to store row indexes:

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



Four Types of Rows

- Given cols 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

- 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

The Min-Hash Property

- Choose a random permutation π
- Claim: $\Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)$
- Why?

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

One of the two
cols had to have
1 at position y

- 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 $y \in C_2$
- So the prob. that both are true is the prob. $y \in C_1 \cap C_2$
- $\Pr[\min(\pi(C_1)) = \min(\pi(C_2))] = |C_1 \cap C_2| / |C_1 \cup C_2| = \text{sim}(C_1, C_2)$

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 π

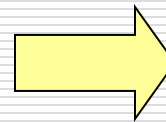
2	4	3
3	2	4
7	1	7
6	3	2
1	6	6
5	7	1
4	5	5

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 $sig(C)[i] = \infty$
 - 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 $h(x)$?

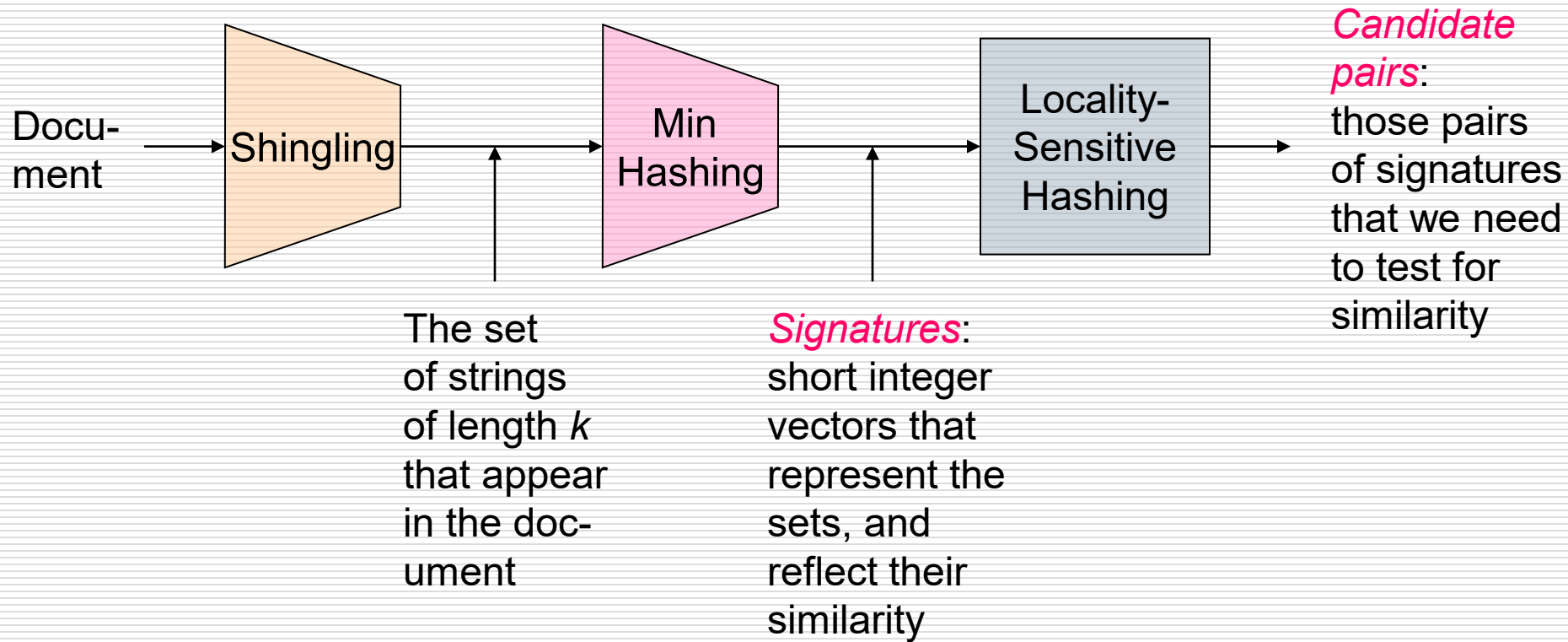
Universal hashing:

$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod N$
where:

$a, b \dots$ random integers

$p \dots$ prime number ($p > N$)

The Big Picture



- Step 3: *Locality-Sensitive Hashing:*
Focus on pairs of signatures likely to be from similar documents

Motivation for 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...

Problem for Today's Lecture

precision? Shingling

n

high time complexity: $O(n^2) \Rightarrow O(n)$

LSH

large space
requirement:
MinHash

	doc1	doc2	docN
	1	1		1
	0	1		1
	1	1		1
	0	1		1
	0	0	0
i	1	1		1
	0	1		1
	1	1		1
	0	0		0

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 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

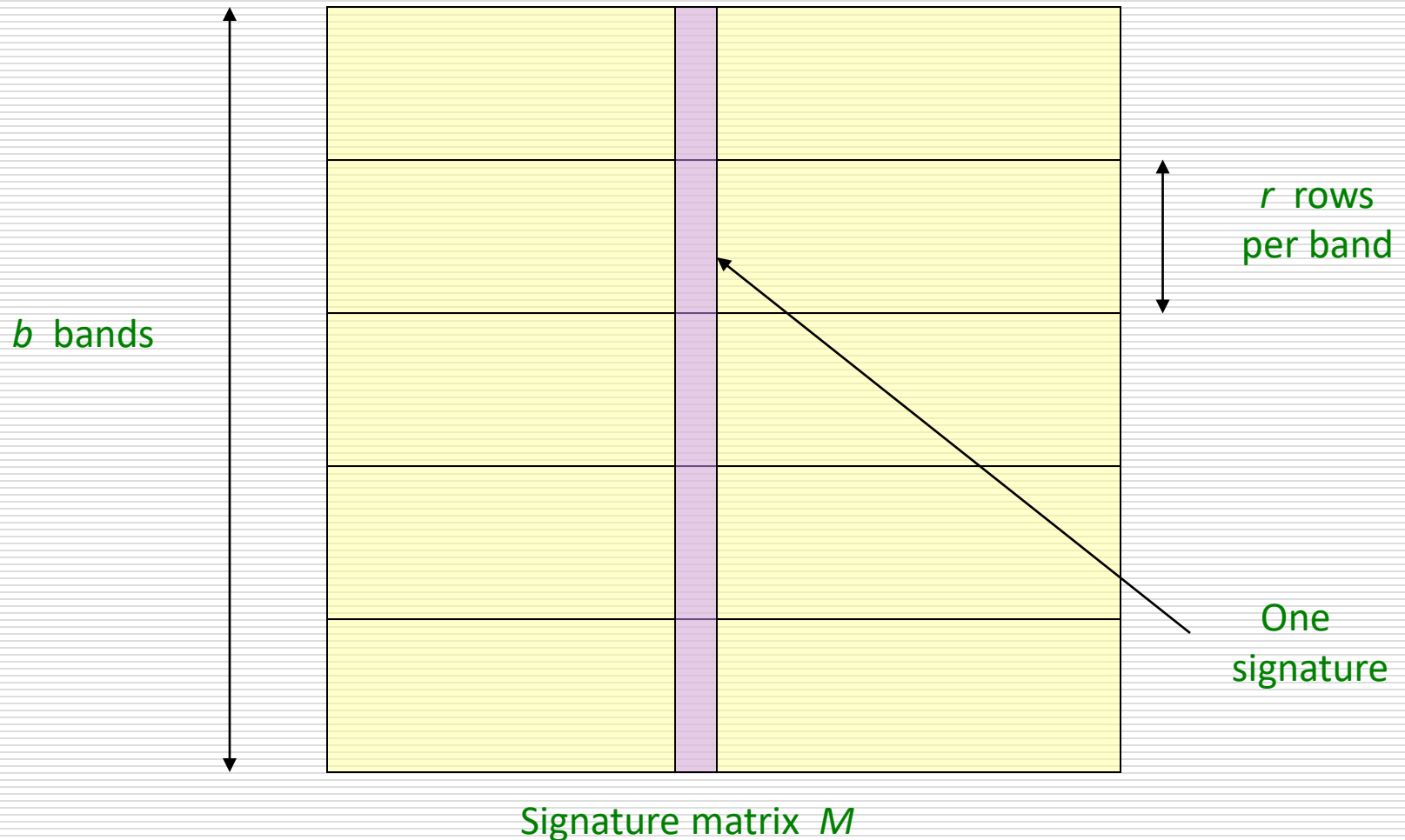
LSH for Min-Hash

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

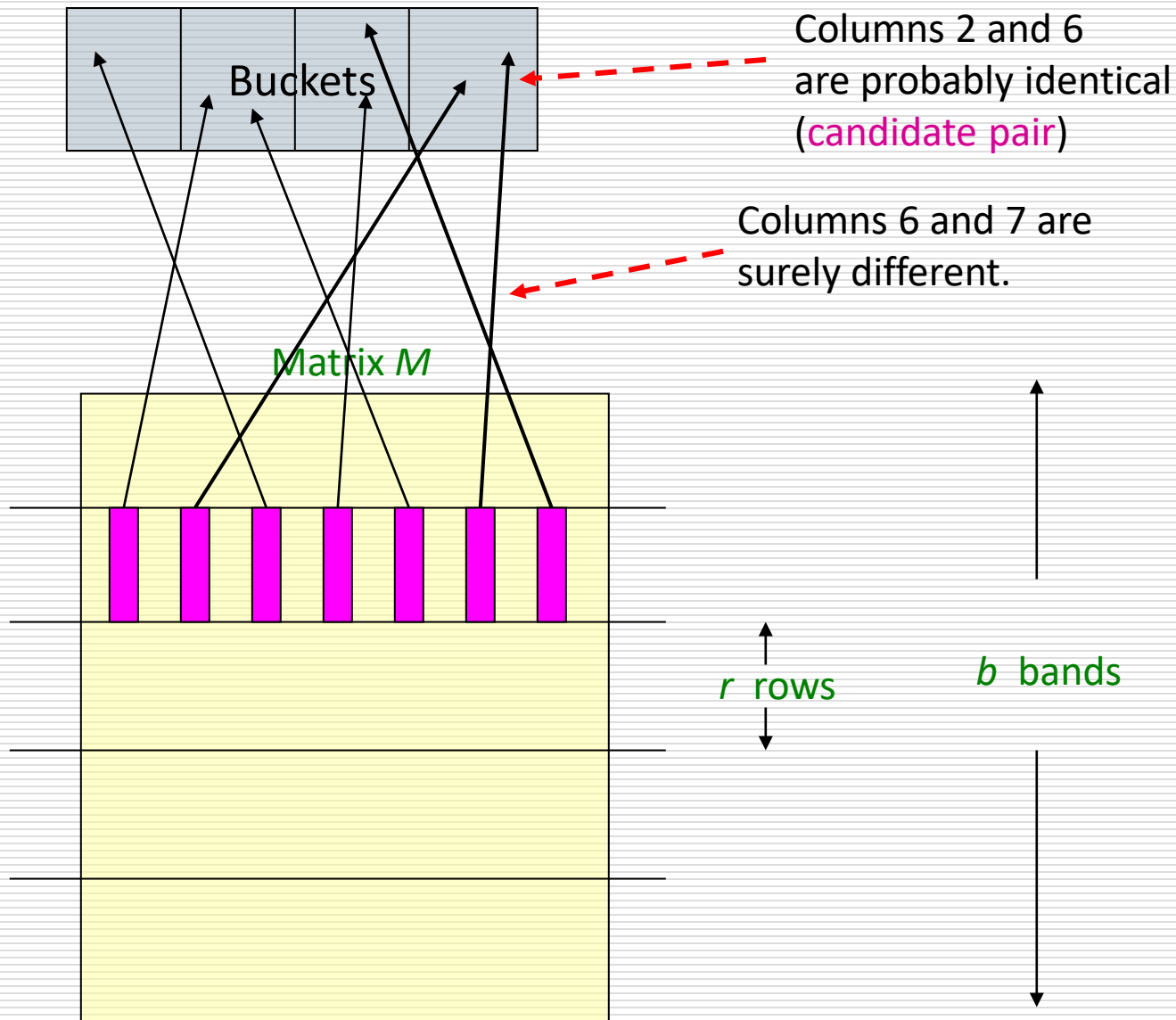
2	1	4	1
1	2	1	2
2	1	2	1



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 ≥ 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 $b = 20$ 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

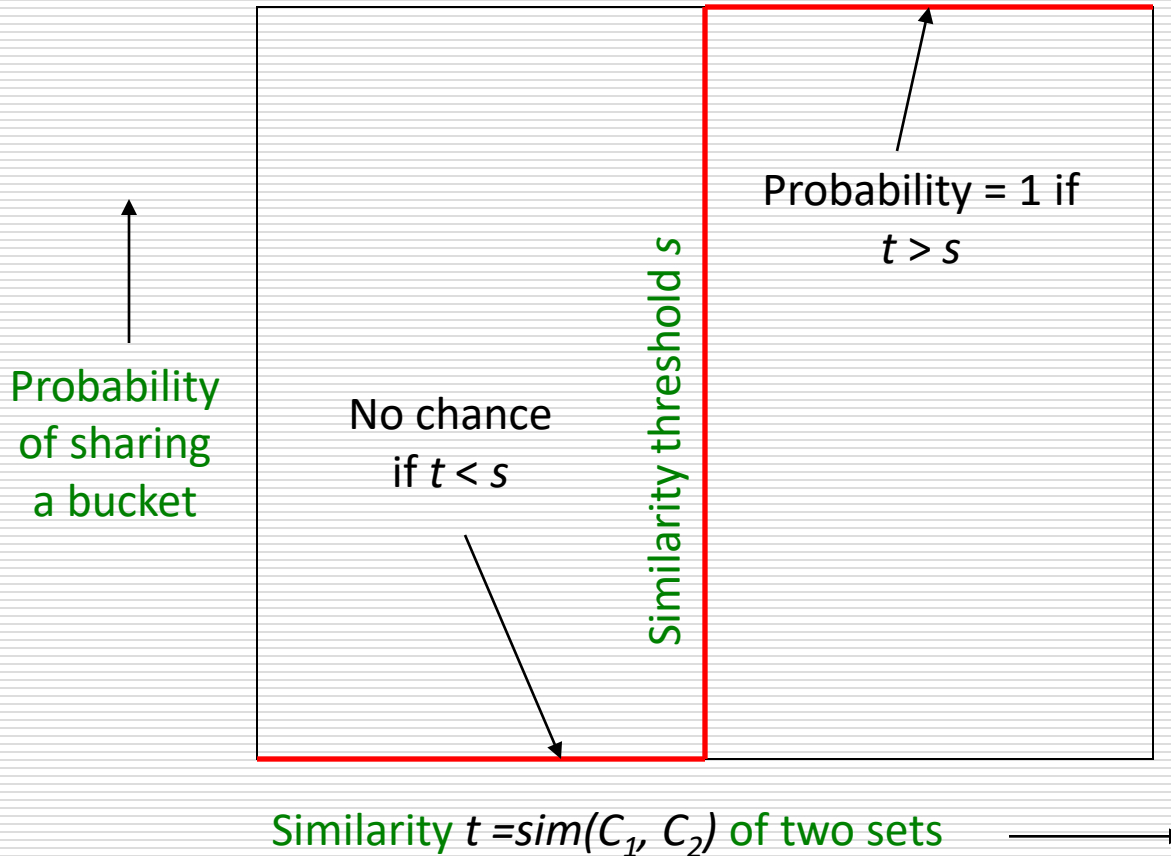
□ Pick:

- The number of Min-Hashes (rows of M)
- The number of bands b , and
- The number of rows r per band

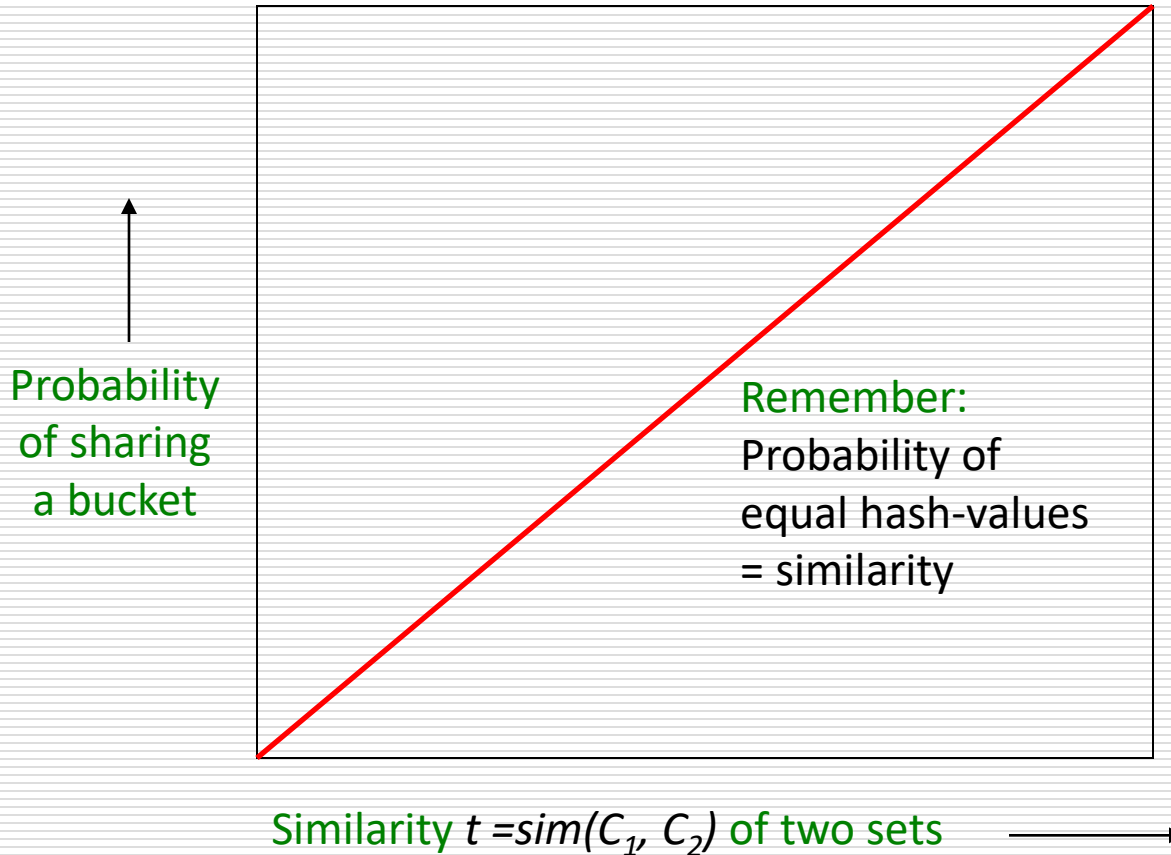
to balance false positives/negatives

- **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



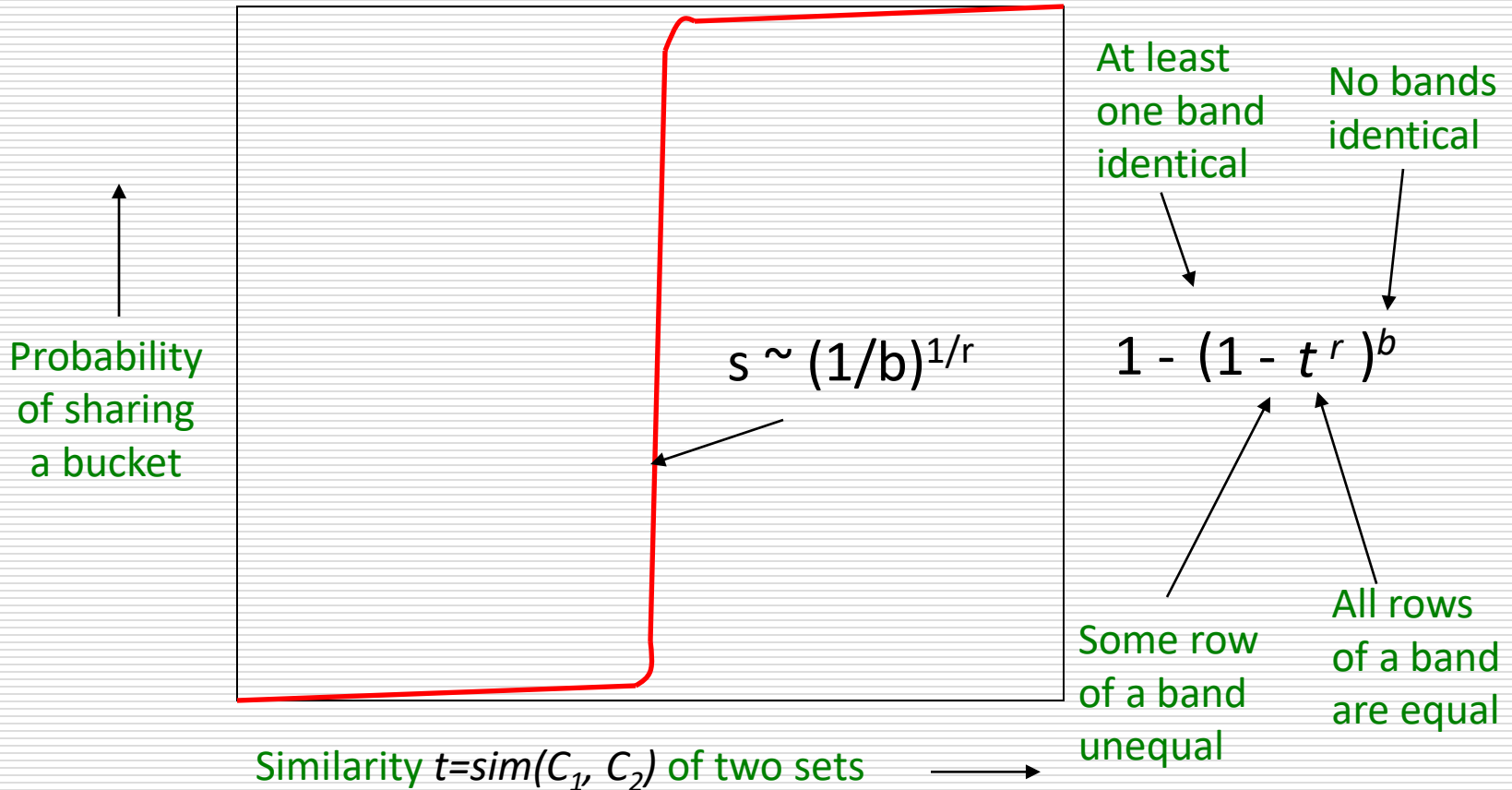
What 1 Band of 1 Row Gives You



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**

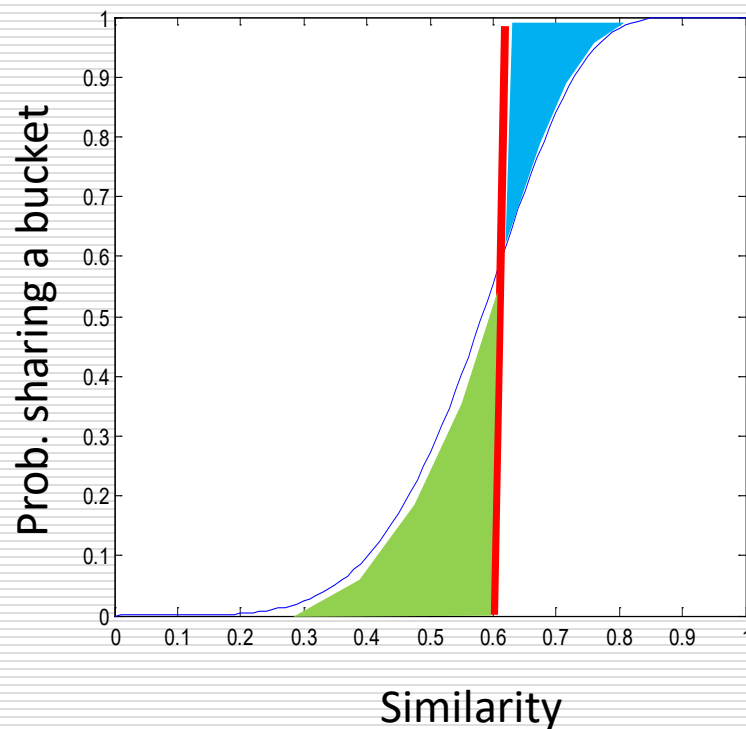
□ **Prob. that at least 1 band is identical:**

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

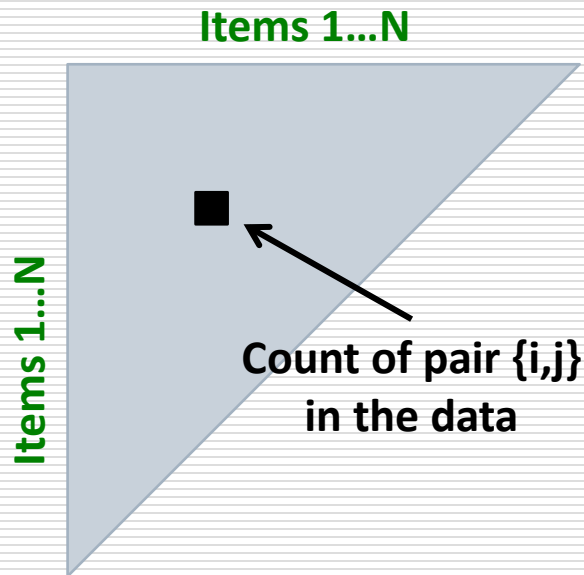
- Tune M, b, r to get almost all pairs with similar signatures, but eliminate most pairs that do not have similar signatures
- Check in main memory that **candidate pairs** really do have **similar signatures**
- **Optional:** In another pass through data, check that the remaining candidate pairs really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

Relation to Previous Lecture

□ Last time: Finding frequent pairs

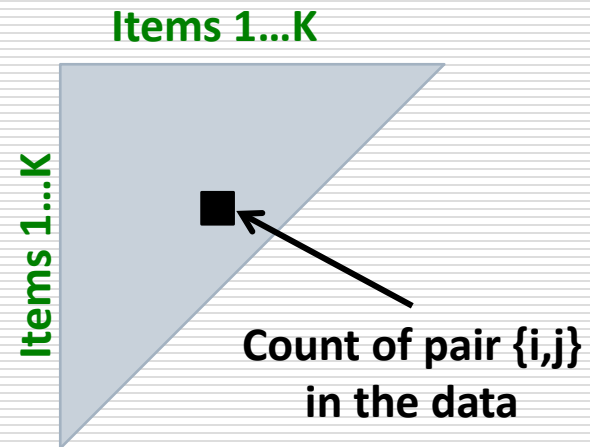


Naïve solution:

Single pass but requires space quadratic in the number of items

N ... number of distinct items

K ... number of items with support $\geq s$



A-Priori:

First pass: Find frequent singletons

For a pair to be a frequent pair candidate, its singletons have to be frequent!

Second pass:

Count only candidate pairs!

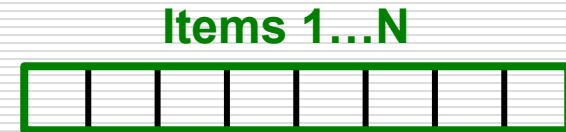
Relation to Previous Lecture

□ Last time: Finding frequent pairs

□ Further improvement: PCY

■ Pass 1:

□ Count exact frequency of each item:



□ Take pairs of items $\{i,j\}$, hash them into B buckets and count of the number of pairs that hashed to each bucket:



Basket 1: ~~$\{1,2,3\}$~~

Pairs: $\{1,2\}$ $\{1,3\}$ $\{2,3\}$

Relation to Previous Lecture

□ Last time: Finding frequent pairs

□ Further improvement: PCY

■ Pass 1:

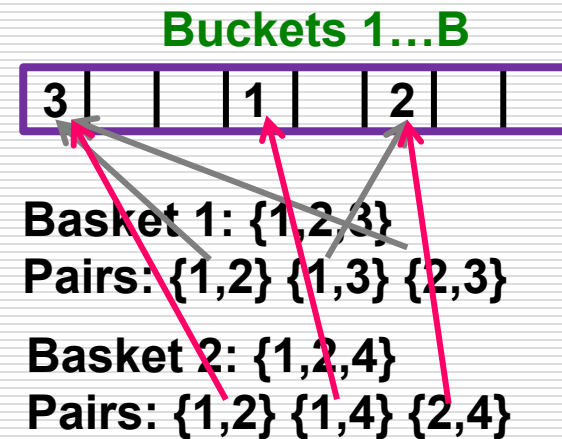
□ Count exact frequency of each item:



□ Take pairs of items $\{i,j\}$, hash them into B buckets and count of the number of pairs that hashed to each bucket:

■ Pass 2:

□ For a pair $\{i,j\}$ to be a **candidate for a frequent pair**, its singletons $\{i\}$, $\{j\}$ have to be frequent and the pair has to hash to a frequent bucket!



Relation to Previous Lecture

□ Last time: Finding frequent pairs

□ **Previous lecture: A-Priori**

■ Main idea: Candidates

Instead of keeping a count of each pair, only keep a count of candidate pairs!

Today's lecture: Find pairs of similar docs

Main idea: Candidates

■ -- Pass 1: Take documents and hash them to buckets such that documents that are similar hash to the same bucket

-- Pass 2: Only compare documents that are candidates (i.e., they hashed to a same bucket)

Benefits: Instead of $O(N^2)$ comparisons, we need $O(N)$ comparisons to find similar documents



3}

{2,4}

Acknowledgement

- Slides are adapted from:
 - Prof. Jeffrey D. Ullman
 - Dr. Anand Rajaraman
 - Dr. Jure Leskovec