

Scalable techniques for related table discovery

Prof. Giovanni Simonini

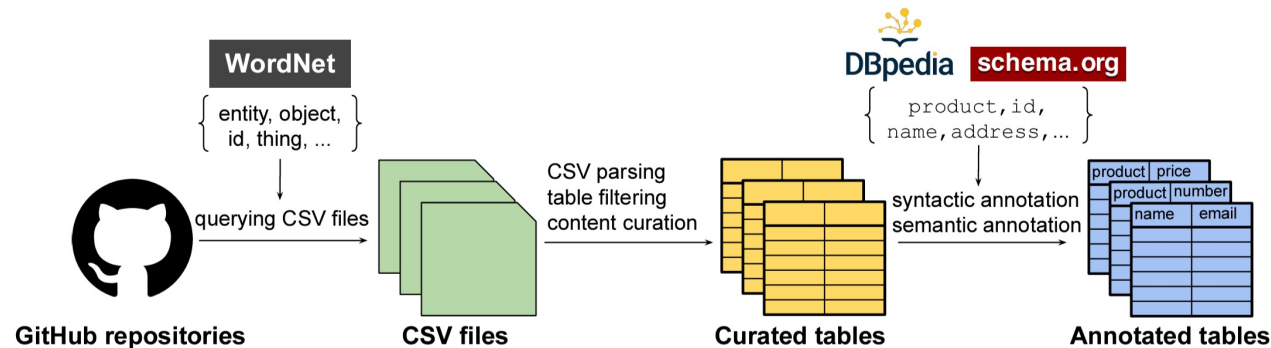
Università degli Studi di Modena e Reggio Emilia

simonini@unimore.it

Introduction

Web tables

- 14.1 billion HTML tables extracted from the English documents in Google main index, 154M distinct relational databases out of them... in 2008! [1]
- 2.13M tables in Wikipedia (September 2019) [2]
- Millions of relational tables on GitHub (at least 10M) [3]

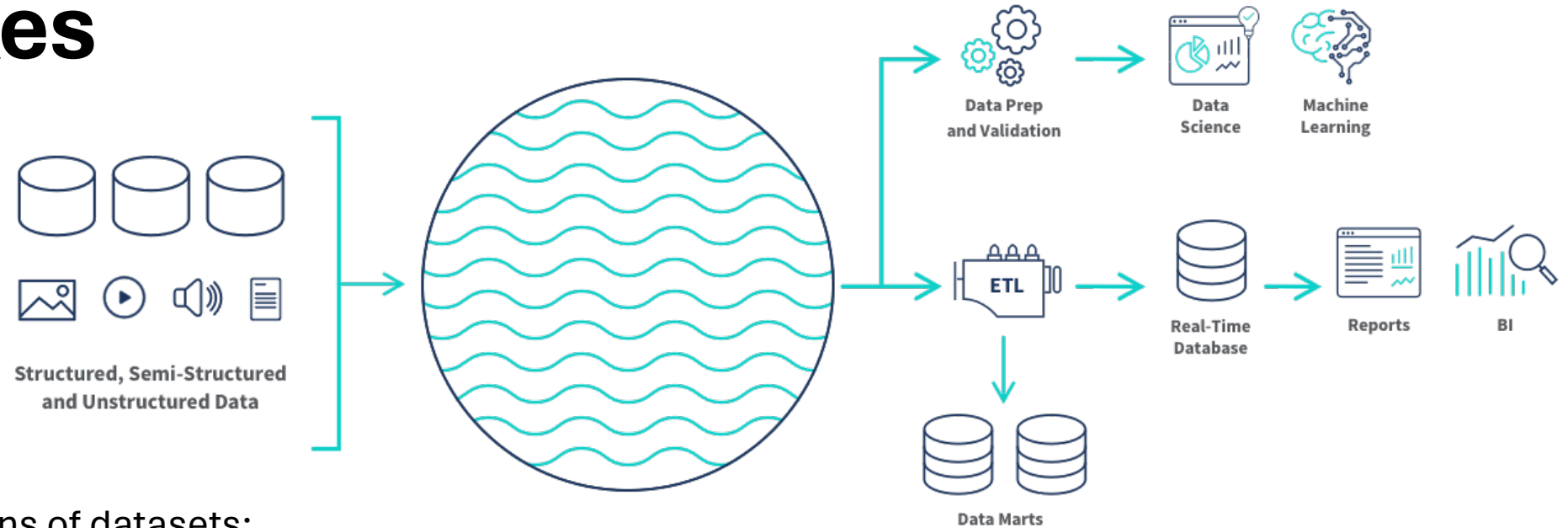


[1] Michael J. Cafarella, Alon Y. Halevy, Daisy Zhe Wang, Eugene Wu, Yang Zhang: “WebTables: Exploring the Power of Tables on the Web”. Proceedings of the VLDB Endowment (PVLDB), 2008. <https://doi.org/10.14778/1453856.1453916>

[2] Tobias Bleifuß, Leon Bornemann, Dmitri V. Kalashnikov, Felix Naumann, Divesh Srivastava: “The Secret Life of Wikipedia Tables”. Proceedings of the International Workshop on Search, Exploration, and Analysis in Heterogeneous Datastores (SEA Data @ VLDB), 2021. <https://ceur-ws.org/Vol-2929/paper4.pdf>

[3] Madelon Hulsebos, Çagatay Demiralp, Paul Groth: “GitTables: A Large-Scale Corpus of Relational Tables”. Proceedings of the ACM on Management of Data (PACMOD), 2023. <https://doi.org/10.1145/3588710>

Data lakes



- Massive collections of datasets:
 - Possibly hosted in different storage systems
 - Heterogeneous formats
 - No metadata or different formats for metadata
 - Change autonomously over time
- Storage layer for experimental data (both input and output of data analysis and learning tasks)
- In current data lakes, coupled with distributed computational frameworks (e.g., Spark) and tools for data management and governance → Infrastructure for sharing and reusing massive datasets

Data lakes

- Some of the data is extracted, transformed, and loaded into DBMSs or data warehouses
- Some data exclusively consumed on-demand to perform specific data analysis tasks
 - **On-demand query answering**: data discovery, extraction, cleaning, and integration done at query time on massive collections of datasets with possibly unknown structure, content, and data quality

Dataset discovery

Process of **identifying datasets that may meet an information need**

For instance, given a table, we may want to enhance its value by integrating it with additional information contained in **related tables** from a table corpus

But how to detect related tables in such large table corpora?

School1					
Name	Postcode	Type	Town	Gender	Borough
Goresbrook	RM9 4TX	Free	Dagenham	Mixed	Barking and Dagenham
St Mary's	NW3 6PG	Independent	London	Mixed	Camden
City of London	EC4V 3AL	Not applicable	London	Boys	City of London

School2					
Name	Address	Town	Postcode	Phase	LA_Name
Argyle	Tonbridge Street	London	WC1H 9EG	Primary	Camden
Millenium	50 John Harrison Way	London	SE10 0BG	Primary	Greenwich
Marshalls Park	Pettits Lane	Romford	RM1 4EH	Secondary	Havering

School3					
Name	Pupils	A*-C	At Least One	Average Points	LA_Name
St Mary's	167	34%	99%	325.5	Camden
Riverston	39	62%	100%	384.8	Greenwich

Deprivation			
Code	Name	Income Deprivation Rate	Deprivation Rate Quintile
E09000002	Barking and Dagenham	19.4%	1
E07000066	Camden	13.8%	2

Brownfield Sites				
Id	Organisation	Site Name Address	Hectares	Planning Status
670	Barking and Dagenham	9-10 The Triangle	0.05	Permissioned
19	London Borough of Camden	Park Village East/ Augustus St	0.01	Not permissioned

Related tables

Formalization of the related table discovery problem by Das Sarma et al. in 2012 [1]

A pair of tables T_1 and T_2 from a table corpus τ are related if it is possible to identify a virtual table T such that T_1 and T_2 are the results of applying two queries, Q_1 and Q_2 , over T

- The schemas may involve **renaming of the attributes**
- The table T must be **coherent**
- The queries should have **similar structures** (e.g., both projections/selections on T)

Two most common types of related tables:

- **Unionable tables** (i.e., *entity complement*)
- **Joinable tables** (i.e., *schema complement*)

[1] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Halevy, Hongrae Lee, Fei Wu, Reynold Xin, Cong Yu: “Finding Related Tables”. Proceedings of the ACM International Conference on Management of Data (SIGMOD), 2012. <https://doi.org/10.1145/2213836.2213962>

Related tables

Formalization of the related table discovery problem by Das Sarma et al. in 2012 [1]

A pair of tables T_1 and T_2 from a table corpus τ are related if it is possible to identify a virtual table T such that T_1 and T_2 are the results of applying two queries, Q_1 and Q_2 , over T

- The schemas may involve **renaming of the attributes**
- The table T must be **coherent**
- The queries should have **similar structures** (e.g., both projections/selections on T)

Two most common types of related tables:

- **Unionable tables** (i.e., *entity complement*)
- **Joinable tables** (i.e., *schema complement*)

[1] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Halevy, Hongrae Lee, Fei Wu, Reynold Xin, Cong Yu: “Finding Related Tables”. Proceedings of the ACM International Conference on Management of Data (SIGMOD), 2012. <https://doi.org/10.1145/2213836.2213962>

Unionable tables

Defined as **entity complement**:

- entity consistency
- entity expansion
- schema similarity

SINGLES







SINGLES LIVE

Top 100

All Countries

Current Week

The Pepperstone ATP Rankings, the historical merit-based method to determine tournament entry and seeding based on men's tennis rankings, is updated Sunday night (ET) after tournaments end. To learn more, visit our [FAQ](#) page.

Rank ^	+/- Rank ^		Player ^	Age ^	Points ^	+/- Points ^	Tourn Played ^	Dropping ^	Next Best ^
1	-		Novak Djokovic	36	9,945	-1,500	17	0	0
2	-		Carlos Alcaraz	20	8,455	-	17	0	0
3	-		Daniil Medvedev	27	7,200	-	21	0	0
4	-		Jannik Sinner	22	5,490	-	22	0	0
5	-		Andrey Rublev	26	4,805	-400	24	0	0
6	-		Stefanos Tsitsipas	25	4,235	-200	23	0	0

2023 Men Tennis Top 100 from ATP Tour

SINGLES

SINGLES LIVE







101-200

All Countries

Current Week

↺↻

The Pepperstone ATP Rankings, the historical merit-based method to determine tournament entry and seeding based on men's tennis rankings, is updated Sunday night (ET) after tournaments end. To learn more, visit our [FAQ](#) page.

Rank ^	+/- Rank ^		Player ^	Age ^	Points ^	+/- Points ^	Tourn Played ^	Dropping ^	Next Best ^
101	▼ -18		Hugo Gaston	23	609	▼ -92	29	0	0
102	▼ -1		Liam Broady	29	608	▼ -1	27	0	0
103	▼ -1		Denis Shapovalov	24	605	-	20	0	0
104	▲ +5		Alejandro Tabilo	26	602	▲ +30	23	0	0
105	▼ -1		David Goffin	32	594	-	27	0	0
106	▼ -1		Tomas Barrios Vera	25	586	-	27	0	0

2023 Men Tennis 100 - 200 from ATP Tour

Unionable tables

Defined as **entity complement**

Table $T_2 \in \tau$ is entity complement to $T_1 \in \tau$ if there exists a coherent virtual table T such that $Q_1(T) = T_1$ and $Q_2(T) = T_2$, where:

- Q_i takes the form $\sigma_{P_i(X)}(T)$, where X is a set of attributes in T and P_i is a selection predicate over X (*i.e., different selection predicates on the same set of attributes X*)
- The combination of Q_1 and Q_2 covers all tuples in T , and Q_2 covers some tuples not covered by Q_1
- Optionally, each Q_i renames or projects a set of attributes A (same for each Q_i) with the restriction that $\exists A' \subseteq A, A' \rightarrow X$ in T (*i.e., projections include the key attributes w.r.t. X*)

Joinable tables

ATP CLASSIFICA 2023

2023 ▾

ATP

WTA

GIOCATORI			PTS
1	  N. DJOKOVIC		9945
2	  C. ALCARAZ		8455
3	 D. MEDVEDEV		7200
4	  J. SINNER		5490
5	 A. RUBLEV		4805
6	  S. TSITSIPAS		4235

2023 Men Tennis 100 from Eurosport

Defined as **schema complement**:

- Coverage of entity set
- Schema expansion

SINGLES







SINGLES LIVE

Top 100

All Countries

Current Week

The Pepperstone ATP Rankings, the historical merit-based method to determine tournament entry and seeding based on men's tennis rankings, is updated Sunday night (ET) after tournaments end. To learn more, visit our [FAQ](#) page.

Rank ^	+/- Rank ^	Player ^	Age ^	Points ^	+/- Points ^	Tourn Played ^	Dropping ^	Next Best ^
1	-	 Novak Djokovic	36	9,945	-1,500	17	0	0
2	-	 Carlos Alcaraz	20	8,455	-	17	0	0
3	-	 Daniil Medvedev	27	7,200	-	21	0	0
4	-	 Jannik Sinner	22	5,490	-	22	0	0
5	-	 Andrey Rublev	26	4,805	-400	24	0	0
6	-	 Stefanos Tsitsipas	25	4,235	-200	23	0	0

2023 Men Tennis Top 100 from ATP Tour

Joinable tables

Defined as **schema complement**

Table $T_2 \in \tau$ is schema complement to $T_1 \in \tau$ if there exists a coherent virtual table T such that $Q_1(T) = T_1$ and $Q_2(T) = T_2$, where:

- Q_i takes the form $\pi_{A_i}(T)$, where A_i is the set of attributes in T to be projected (with optional renaming)
- $A_1 \setminus A_2 \neq \emptyset$, $A_1 \cup A_2$ covers all attributes in T , and $A_1 \cap A_2$ covers key attributes of A_1 and A_2 (i.e., $\exists X \subseteq A_1 \cap A_2, X \rightarrow A_i$)
- Optionally, each Q_i applies a fixed selection predicate P over the set of key attributes X

Similarity Joins

Similarity joins are needed when dealing with textual documents and we want to find records that exactly satisfies a provided predicate

Similarity joins

- Given a set of documents ***D***, a similarity function ***sim*** and a threshold ***t*** a similarity join retrieves all the pairs $r, s \in D \mid \text{sim}(r, s) \geq t$
- Similarity joins can be used in many applications such as:
 - Record Linkage
 - Deduplication
 - Data cleaning
 - Fraud detection (e.g. identify plagiarism)
 - ...

Usage example: Data Integration

Patients Data

ID	Name	Surname	Age	...

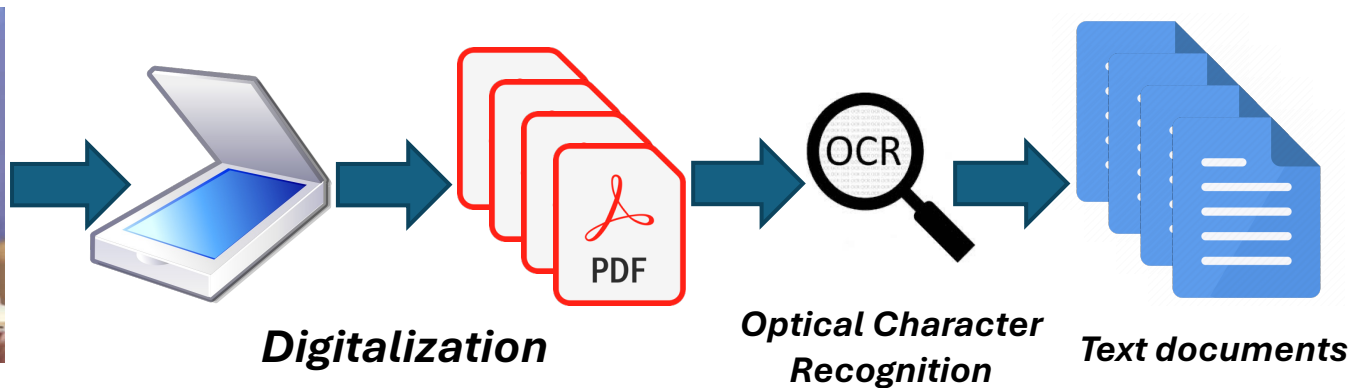
Similarity join
 $\text{sim}(r, d) \geq 0.9$

Mapping Table

Patient ID	Clinical record ID



Clinical records



Similarity measures

Token based

- Jaccard similarity

- $J(x, y) = \frac{|x \cap y|}{|x \cup y|}$

- Cosine similarity

- $C(x, y) = \frac{x \cdot y}{\|x\| \cdot \|y\|} = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2} \cdot \sqrt{\sum_i y_i^2}}$

- Dice similarity

- $D(x, y) = \frac{2 \cdot |x \cap y|}{|x| + |y|}$

- Overlap similarity

- $O(x, y) = |x \cap y|$

Character based

- Edit distance

- Edit distance between two string is the minimum number of

- Insertion

- Deletion

- Substitution

- Needed to transform one into the other

Note: all these similarity measures can be traced back to the overlap similarity

Type of Joins	Measure	Definition	Equivalent Overlap Threshold
character-based	Edit Distance	# character transformations	$\max(x , y) + 1 - (1 + \theta) \times q$
token-based	Overlap	$ x \cap y $	θ
	Cosine	$ x \cap y / \sqrt{ x \cdot y }$	$\theta \times \sqrt{ x \cdot y }$
	Dice	$2 \cdot x \cap y / (x + y)$	$\theta \times (x + y) / 2$
	Jaccard	$ x \cap y / (x + y - x \cap y)$	$\theta \times (x + y) / (1 + \theta)$

Indexing and filtering

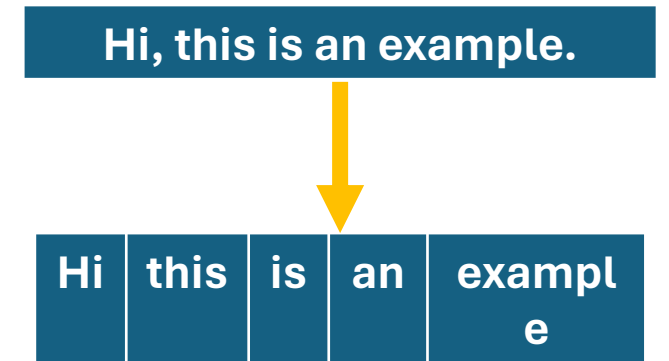
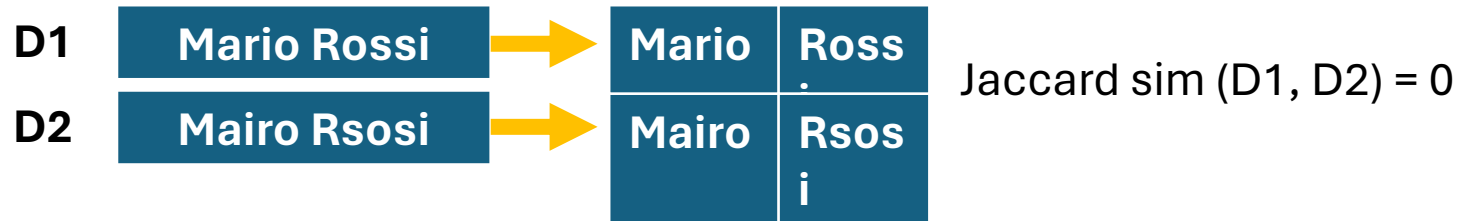
- As we saw before, it is not possible to compare all the possible pairs, due to the time required to perform all the comparisons;
- **Indexing** and **filtering** techniques are employed to reduce the number of comparisons discarding all the pairs that for sure cannot reach the request threshold;
- The most common used indexing technique is called ***prefix index*** (or ***prefix filter***)
- The most common used filters are:
 - ***Length filter***
 - ***Positional filter***

Tokenization

- To apply the techniques described in the previous slide, the documents must be tokenized;
- Many techniques could be used, for example:
 - Word tokenization
 - Q-grams
 - Soundex
 - More complex algorithms

Tokenization – Word tokenization

- Is the simplest technique, splits the text by punctuation and white spaces;
- Could be useful to compare large documents;
- Do not detect misspelled words



Tokenization – Q-grams

- Splits the text into chunks of length q , called q -grams;
- Useful to detecting misspelled words;
- In large documents generates a lot of tokens.

Example of 3-grams generation

D1	Mario	→	##m	#ma	mar	ari	rio	io#	o##	Jaccard sim (D1, D2) = 0.3
D2	Mairo	→	##m	#ma	mai	air	iro	ro#	o##	

Prefix filtering

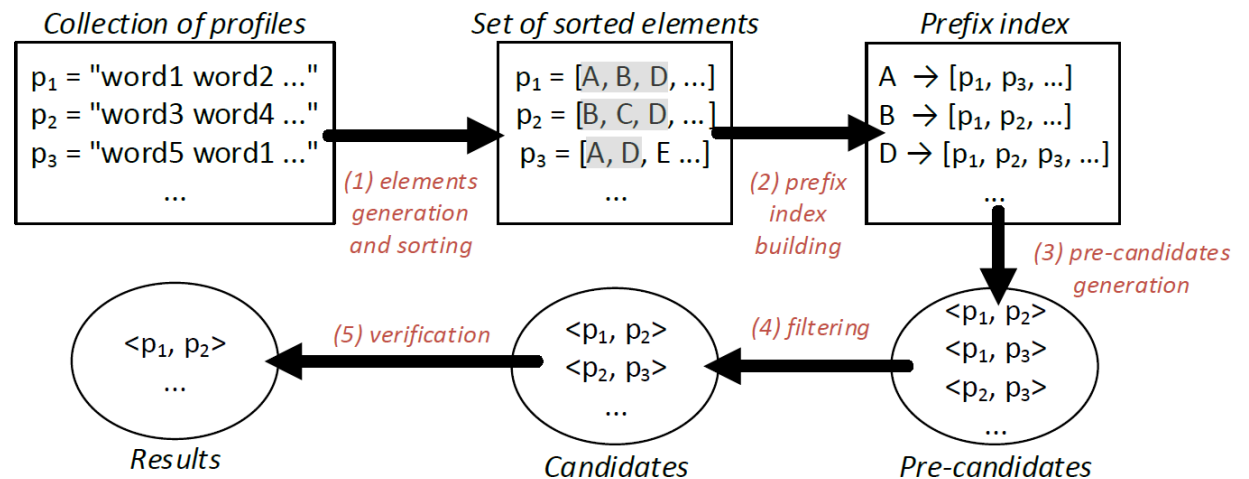
- Let's work with the overlap similarity
 - The formulas could be adapted to other similarity measures
- Given an overlap similarity threshold α , two documents can reach α only if they share at least one token in their first $\text{numDocTokens} - \alpha + 1$ tokens

Example. If $\alpha = 4$ this two documents can be safely discarded, because they share nothing in their prefix and the remaining elements to check are not sufficient to reach the requested overlapping.

D1	A	B	C	?	?	?
D2	D	E	?	?	?	

Prefix indexing

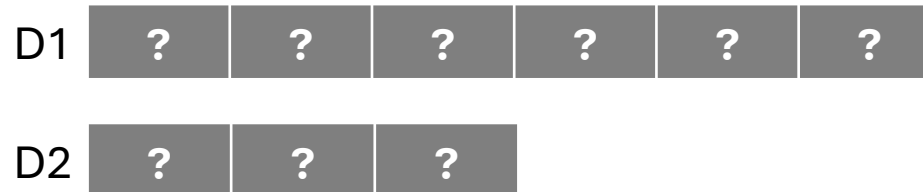
- By using the prefix filter idea, it is possible to index only the documents that share at least one token in their first $\text{numDocTokens} - \alpha + 1$ tokens;
- To perform prefix indexing tokens must be sorted according to a predefined order
 - Usually, term frequency (i.e. in how many documents a token appears) is used to sort the tokens, in this way rarest tokens are indexed generating a smaller index
- Prefix index is used to find the pre-candidate pairs that will be processed by other filtering techniques.



Length filter

- Given an overlap similarity threshold α , two documents can reach α only if they have enough tokens in common to reach α , i.e.
 $\min(|D1|, |D2|) \geq \alpha$

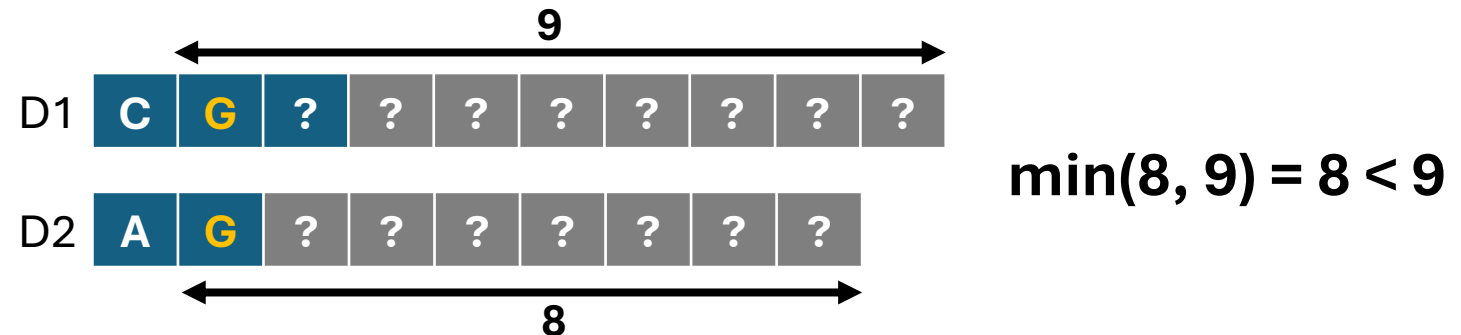
Example. If $\alpha = 4$ this two documents can be safely pruned, because they cannot have 4 tokens in common, since $|D2| = 3$



Positional filter

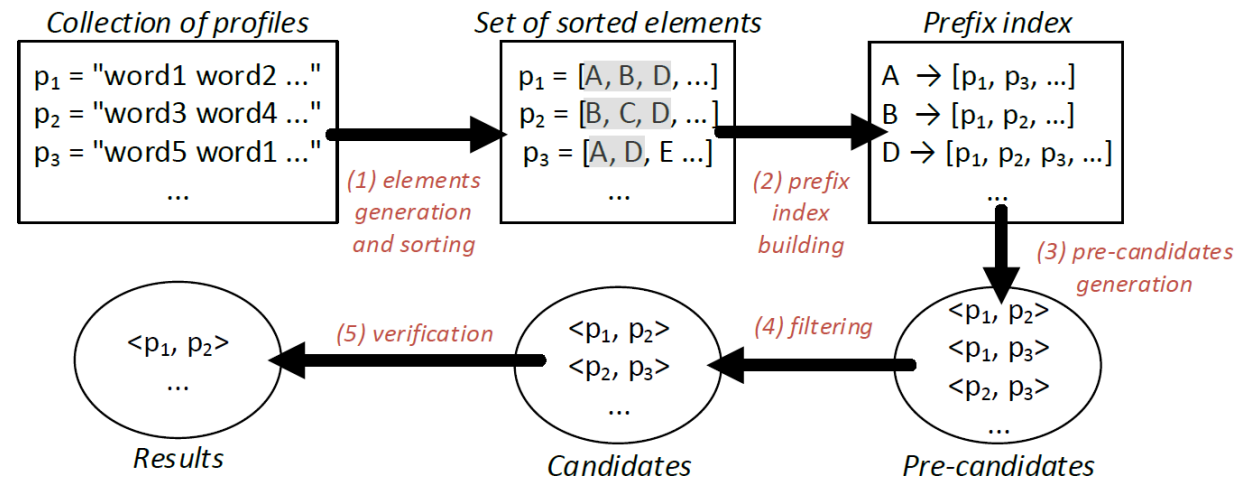
- Positional filter reasoning over the matching position of tokens in the prefix;
- It combines prefix filter with length filter;
- if after the position of the last common token in the prefix, there are no enough tokens to reach the requested threshold the pair is discarded.

Example. If $\alpha = 9$ this two documents can be safely discarded, because they will never reach the requested threshold.



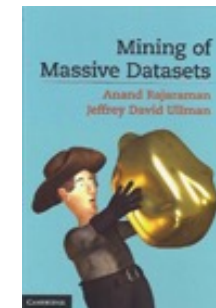
Verification

- Verification is the final phase of the similarity join;
- The similarity is computed for all the pairs of records that are retained after indexing and filtering by using one of the similarity functions described in previous slides;
- Many algorithms exist in literature, one of the best is called **PPJOIN** which applies all the algorithms previously described.



Approximate Similarity

MinHash and LSH



Adapted from Mining of Massive Datasets

Jure Leskovec, Anand Rajaraman, Jeff Ullman

Why Set Similarity?

- **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

Why is it hard?

- **Given: High dimensional data points x_1, x_2, \dots**
 - **For example:** Image is a long vector of pixel colors

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} \rightarrow [1 \ 2 \ 1 \ 0 \ 2 \ 1 \ 0 \ 1 \ 0]$$

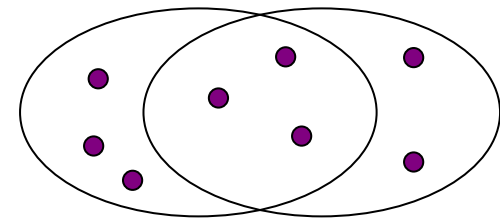
- **And some distance function $d(x_1, x_2)$**
 - Which quantifies the “distance” between x_1 and x_2
- **Goal:** Find **all pairs of data points (x_i, x_j)** that are within some distance threshold $d(x_i, x_j) \leq s$
- **Note:** Naïve solution would take $O(N^2)$ ☹
where N is the number of data points
- **MAGIC: This can be done in $O(N)$!! How?**

Why quadratic is bad?

- 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
 - 10^6 comparisons/sec
 - It would take **5 days**
- For $N = 10$ million, it takes more than a year...

Jaccard Similarity

- **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
- **Jaccard distance/similarity**
 - The **Jaccard similarity** of two **sets** is the size of their intersection divided by the size of their union:
$$\text{sim}(\mathbf{C}_1, \mathbf{C}_2) = |\mathbf{C}_1 \cap \mathbf{C}_2| / |\mathbf{C}_1 \cup \mathbf{C}_2|$$
 - **Jaccard distance:** $d(\mathbf{C}_1, \mathbf{C}_2) = 1 - |\mathbf{C}_1 \cap \mathbf{C}_2| / |\mathbf{C}_1 \cup \mathbf{C}_2|$



Jaccard similarity= 3/8
Jaccard distance = 5/8

Find similar documents

- **Goal:** Given a large number (millions/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”
- Find similar:
 - routes (e.g., Uber)
 - songs (e.g., Shazam)

Google Research

Deduplicating Training Data Makes Language Models Better

Katherine Lee^{*†}

Daphne Ippolito^{*†‡}

Andrew Nystrom[†]

Chiyuan Zhang[†]

Douglas Eck[†]

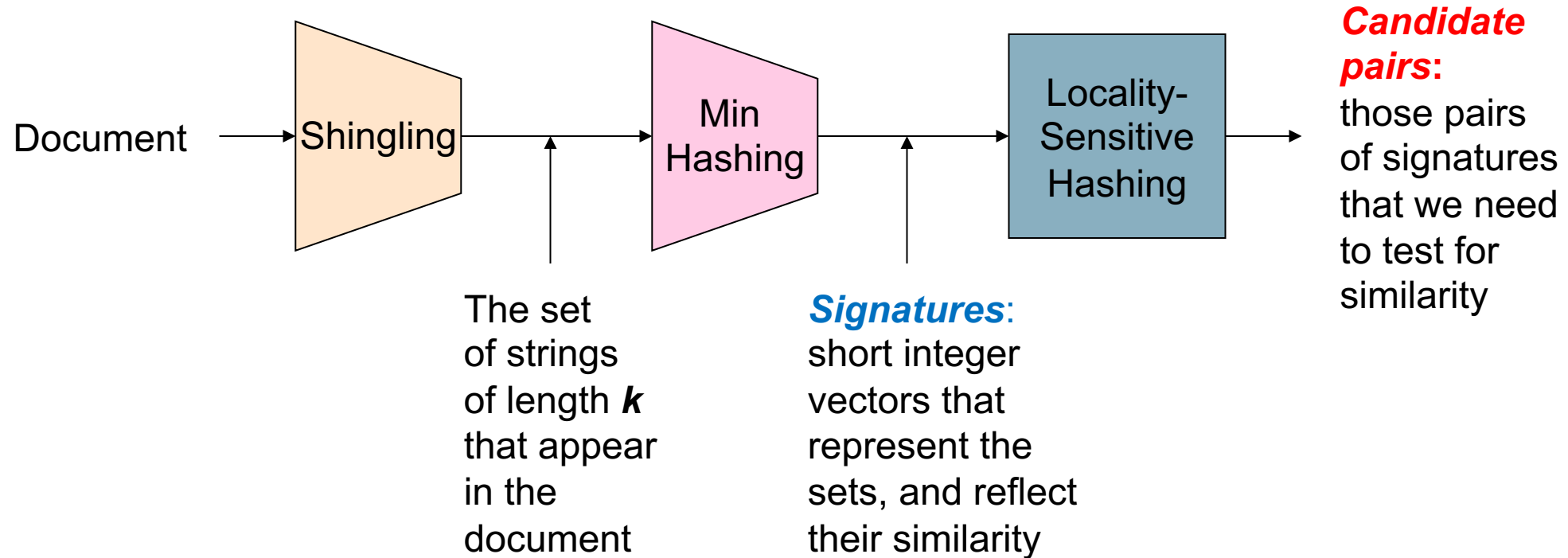
Chris Callison-Burch[‡]

Nicholas Carlini[†]

- **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

LSH: process overview



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
 - Might not 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 $D_1 = \text{abcab}$
Set of 2-shingles: $S(D_1) = \{\text{ab}, \text{bc}, \text{ca}\}$
 - **Option:** Shingles as a bag (multiset), count ab twice: $S'(D_1) = \{\text{ab}, \text{bc}, \text{ca}, \text{ab}\}$

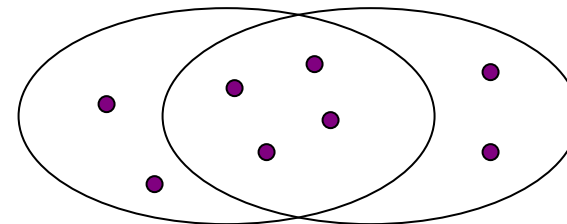
Compressing Shingles

- 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**
 - **Key collision is rarely a problem**
- **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

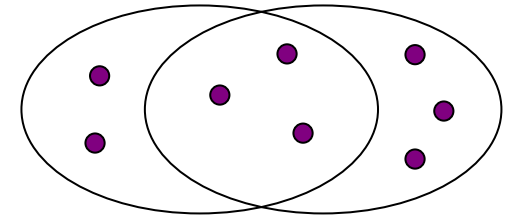
- **Document D_1 is a set of its k -shingles $C_1=S(D_1)$**
- Equivalently, each document is a 0/1 vector in the space of k -shingles
 - **Each unique shingle is a dimension**
 - **Vectors are very sparse**
- **A natural similarity measure is the Jaccard similarity:**

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



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



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 = 3/4**

From Sets to Boolean Matrices

- **Rows** = elements (shingles)
- **Columns** = sets (documents)
 - 1 in row *if the shingle is in the document*
 - 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 = $3/6$

	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
- **Still expensive:**
 - Sparse matrix
 - We still have the quadratic complexity
- **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 \mathbf{C} to a small **signature** $h(\mathbf{C})$, such that:
 - (1) $h(\mathbf{C})$ is small enough that the signature fits in RAM
 - (2) $\text{sim}(\mathbf{C}_1, \mathbf{C}_2)$ is the same as the “similarity” of signatures $h(\mathbf{C}_1)$ and $h(\mathbf{C}_2)$
- **Goal: Find a hash function $h(\cdot)$ such that:**
 - If $\text{sim}(\mathbf{C}_1, \mathbf{C}_2)$ is high, then with high prob. $h(\mathbf{C}_1) = h(\mathbf{C}_2)$
 - If $\text{sim}(\mathbf{C}_1, \mathbf{C}_2)$ is low, then with high prob. $h(\mathbf{C}_1) \neq h(\mathbf{C}_2)$
- **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 $\text{sim}(\mathbf{C}_1, \mathbf{C}_2)$ is high, then with high prob. $h(\mathbf{C}_1) = h(\mathbf{C}_2)$
 - if $\text{sim}(\mathbf{C}_1, \mathbf{C}_2)$ is low, then with high prob. $h(\mathbf{C}_1) \neq h(\mathbf{C}_2)$
- **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 $h_{\pi}(\mathbf{C})$ = the index of the **first** (in the permuted order π) row in which column \mathbf{C} has value **1**:

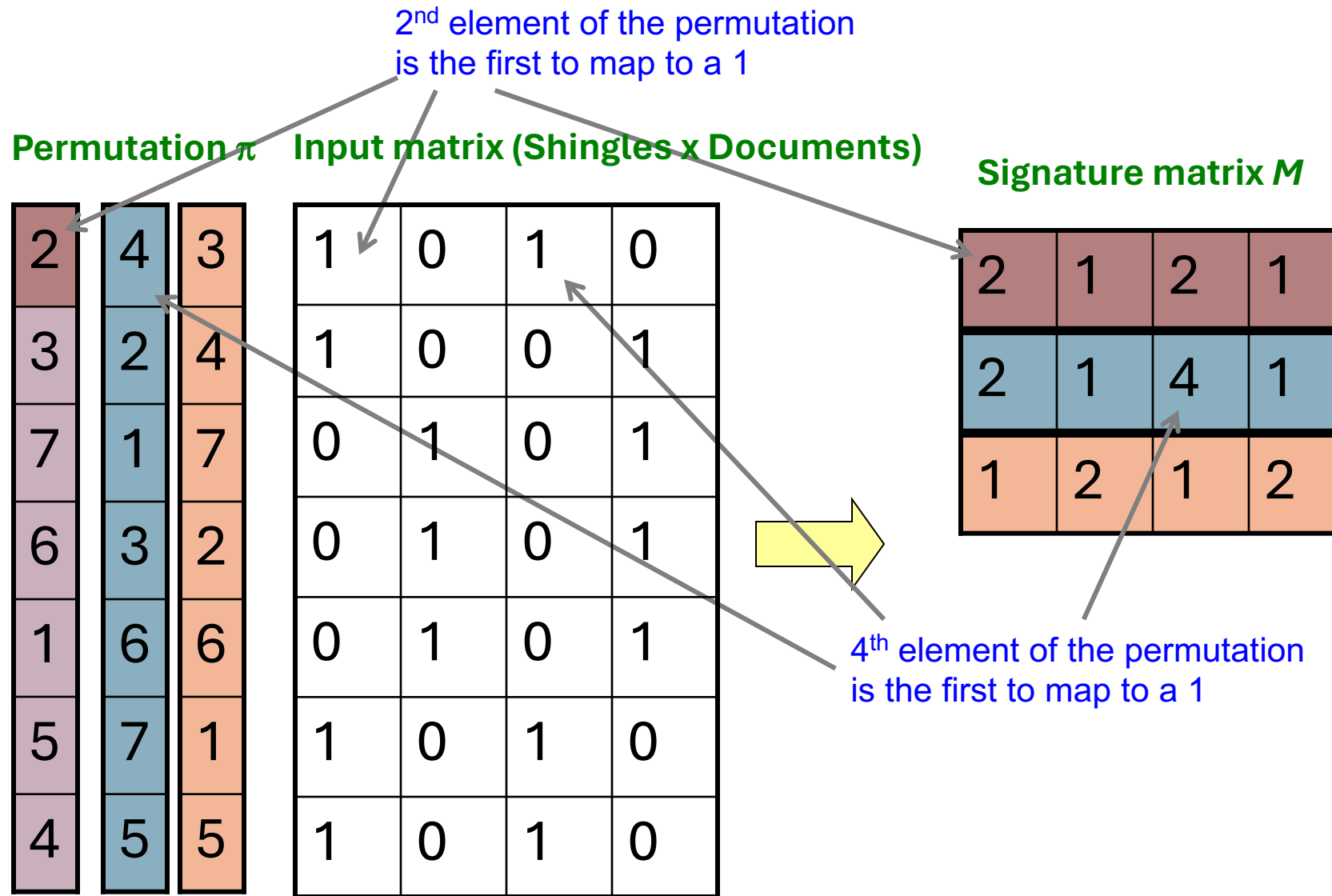
$$h_{\pi}(\mathbf{C}) = \min_{\pi} \pi(\mathbf{C})$$

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

Min-Hashing Example

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

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



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

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

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

Similarity for Signatures

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

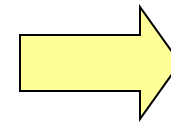
2	4	3
3	2	4
7	1	7
6	3	2
1	6	6
5	7	1
4	5	5

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 $\mathbf{sig}(\mathbf{C})$ as a column vector
- $\mathbf{sig}(\mathbf{C})[i]$ = according to the i -th permutation, the index of the first row that has a 1 in column C
$$\mathbf{sig}(\mathbf{C})[i] = \min (\pi_i(\mathbf{C}))$$
- **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 $\text{sig}(C)[i] = \infty$
 - **Scan rows looking for 1s**
 - Suppose row j has 1 in column C
 - Then for each k_i :
 - If $k_i(j) < \text{sig}(C)[i]$, then $\text{sig}(C)[i] \leftarrow k_i(j)$

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 ($p > N$)

Example

	h1	h2	h3						
→	2	4	3	1	0	1	0		
	3	2	4	1	0	0	1		
	7	1	7	0	1	0	1		
	6	3	2	0	1	0	1		
	1	6	6	0	1	0	1		
	5	7	1	1	0	1	0		
	4	5	5	1	0	1	0		

-	-	-	-
-	-	-	-
-	-	-	-

Example

	h1	h2	h3						
→	2	4	3	1	0	1	0		
	3	2	4	1	0	0	1		
	7	1	7	0	1	0	1		
	6	3	2	0	1	0	1		
	1	6	6	0	1	0	1		
	5	7	1	1	0	1	0		
	4	5	5	1	0	1	0		

2	-	2	-
4	-	4	-
3	-	3	-

Example

	h1	h2	h3				
	2	4	3	1	0	1	0
→	3	2	4	1	0	0	1
	7	1	7	0	1	0	1
	6	3	2	0	1	0	1
	1	6	6	0	1	0	1
	5	7	1	1	0	1	0
	4	5	5	1	0	1	0

2	-	2	3
2	-	4	2
3	-	3	4

Example

	h1	h2	h3				
	2	4	3	1	0	1	0
	3	2	4	1	0	0	1
→	7	1	7	0	1	0	1
	6	3	2	0	1	0	1
	1	6	6	0	1	0	1
	5	7	1	1	0	1	0
	4	5	5	1	0	1	0

2	7	2	3
2	1	4	1
3	7	3	4

Example

	h1	h2	h3				
	2	4	3	1	0	1	0
	3	2	4	1	0	0	1
	7	1	7	0	1	0	1
→	6	3	2	0	1	0	1
	1	6	6	0	1	0	1
	5	7	1	1	0	1	0
	4	5	5	1	0	1	0

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

Example

	h1	h2	h3				
	2	4	3	1	0	1	0
	3	2	4	1	0	0	1
	7	1	7	0	1	0	1
	6	3	2	0	1	0	1
	1	6	6	0	1	0	1
	5	7	1	1	0	1	0
→	4	5	5	1	0	1	0

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

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

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**

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

Candidates from Min-Hash

- 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

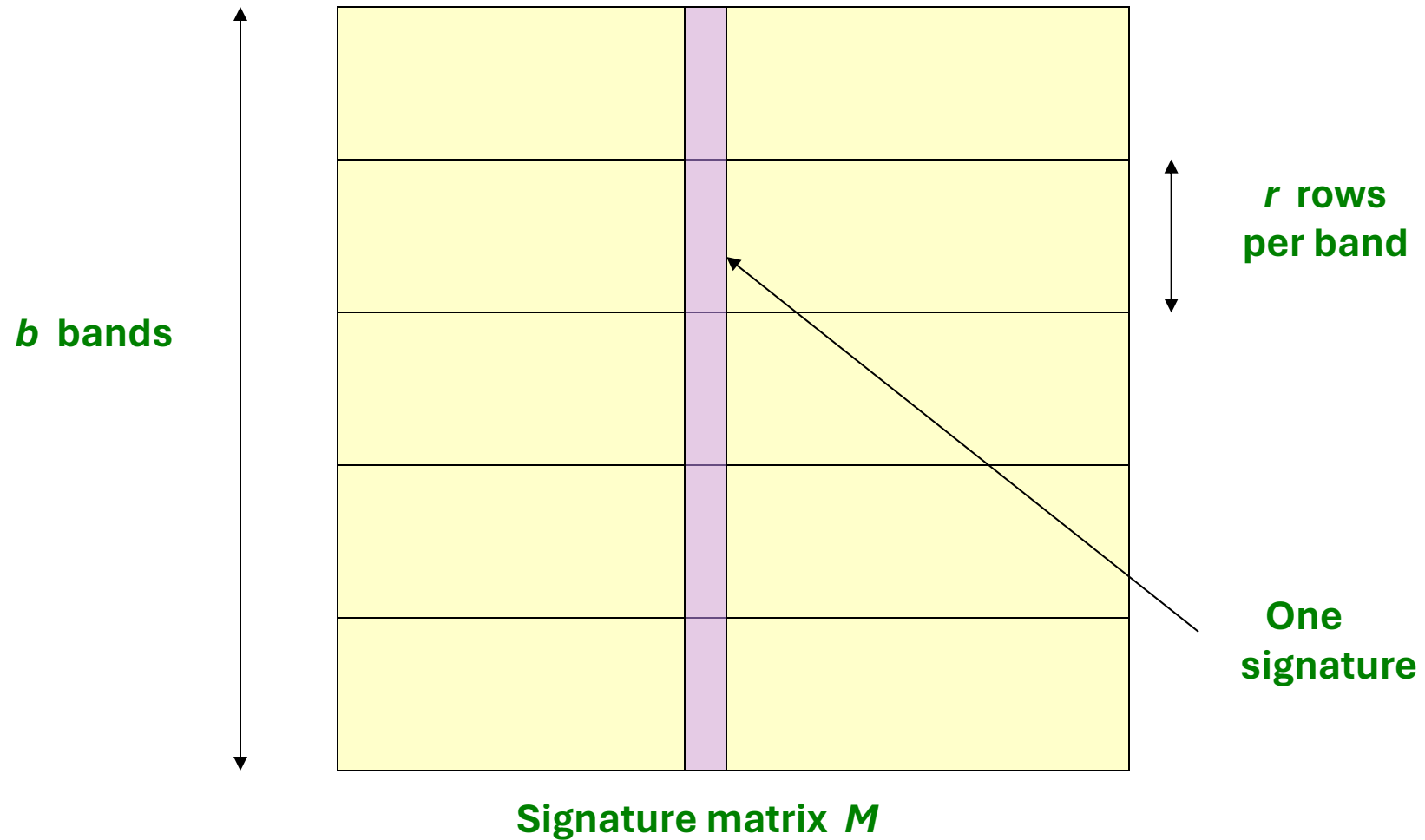
LSH for Min-Hash

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

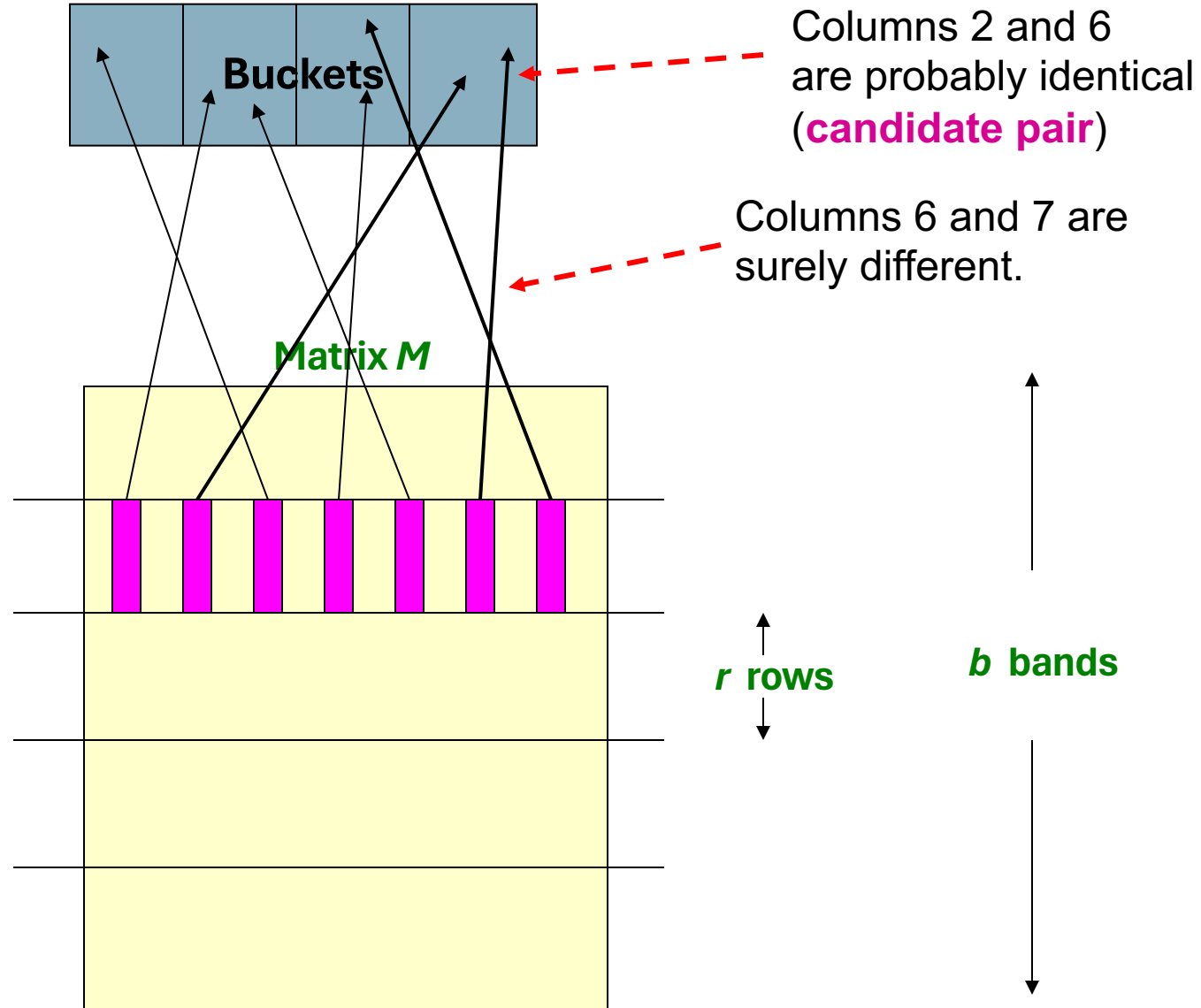
2	1	4	1
1	2	1	2
2	1	2	1



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

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

C_1, C_2 are 30% Similar

- 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

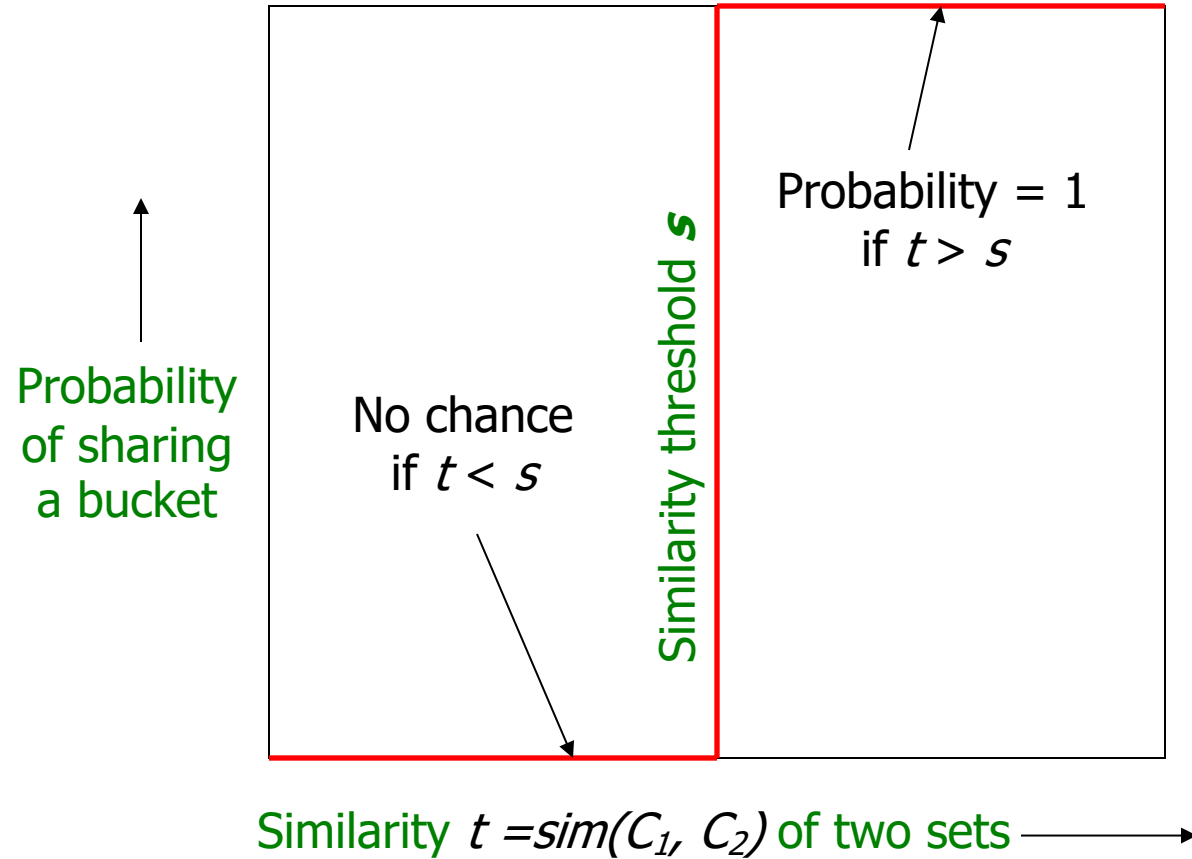
- **Pick:**

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

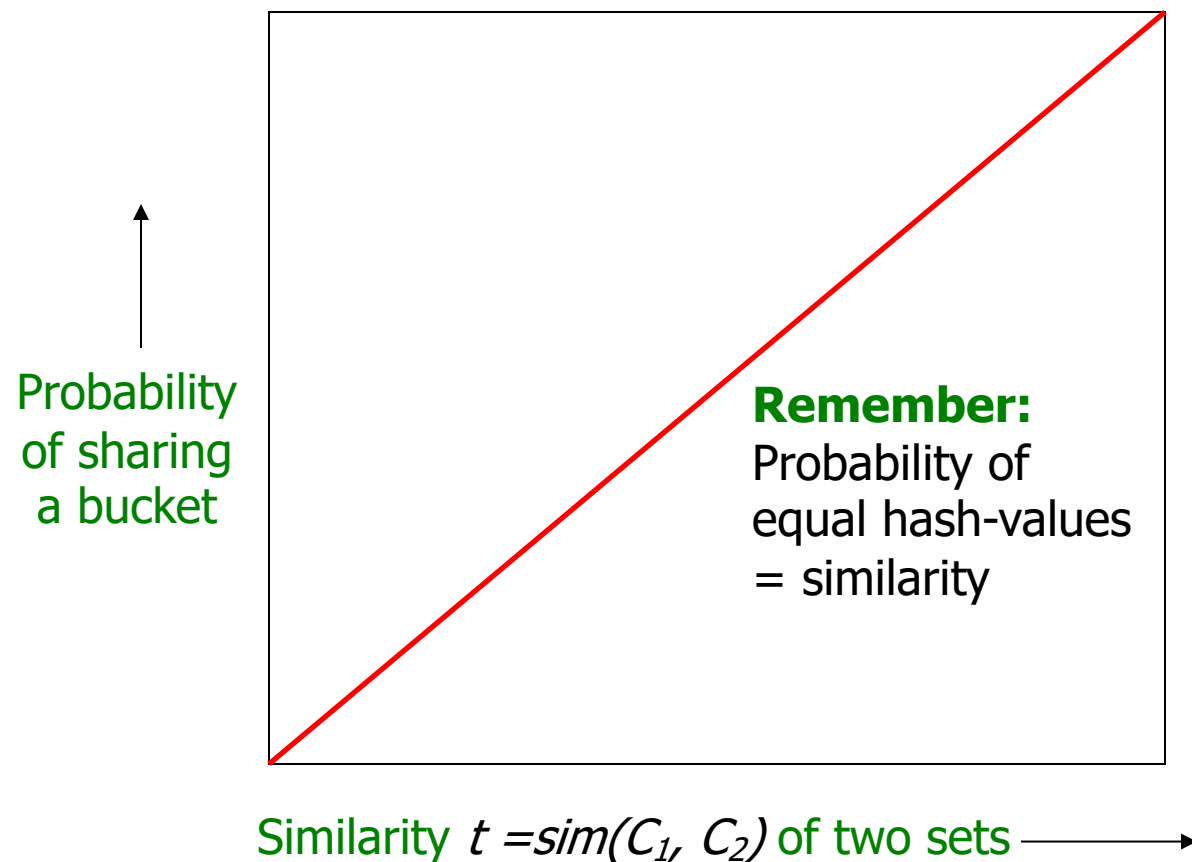
to balance false positives/negatives

- **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



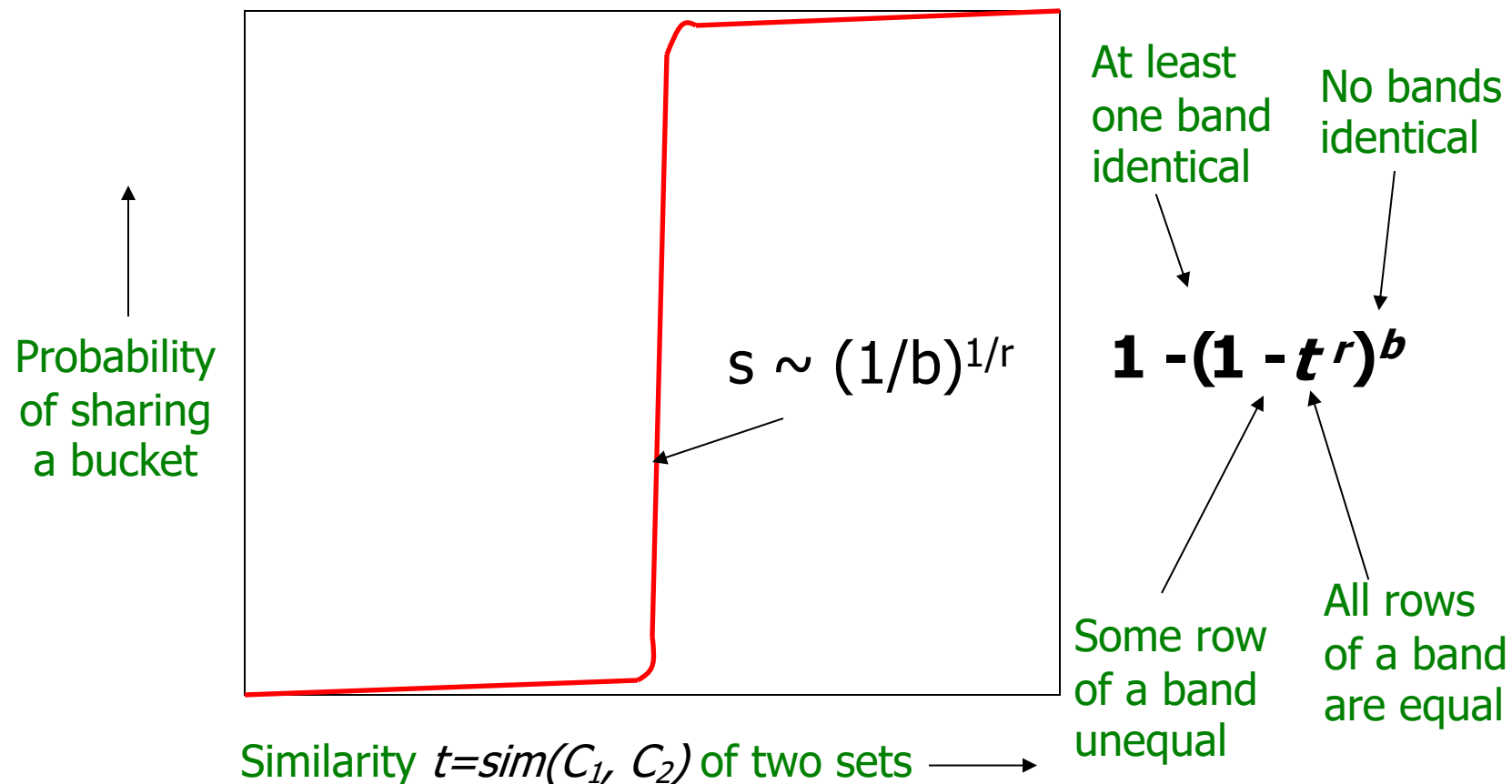
What 1 Band of 1 Row Gives You



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



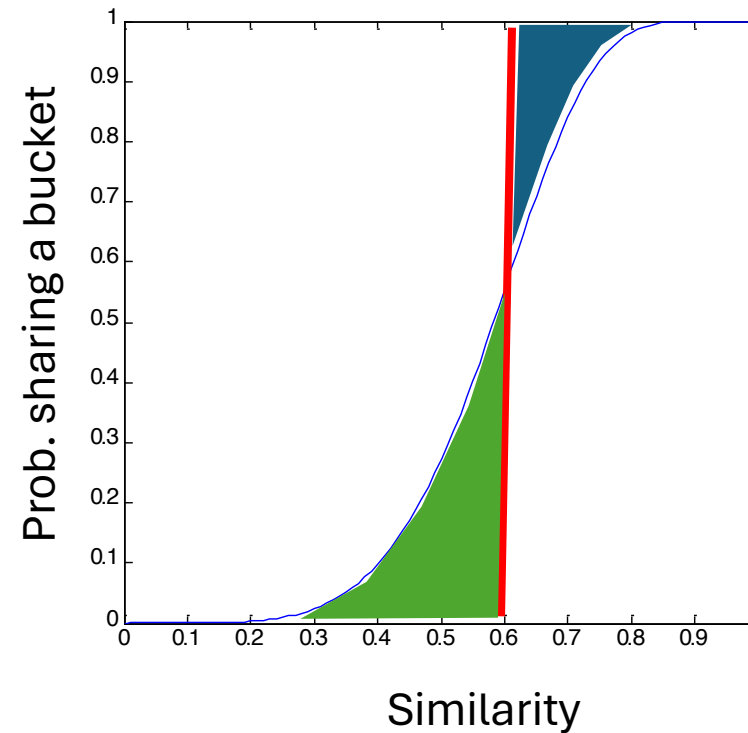
Example: $b = 20$; $r = 5$

- **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 $\geq s$

Compute sketches for cosine similarity

- *The probability that a random projection in the space of the elements “cut” two elements (i.e., vectors), is proportional to the angle between them*
- *For each vector, we compute the bit of the signatures with:*
 - *if $\bar{v} \cdot \bar{r} \geq 0 \rightarrow 1$*
else $\rightarrow 0$
 - *\bar{r} is a random projection, i. e., a random vector of 0s and 1s*

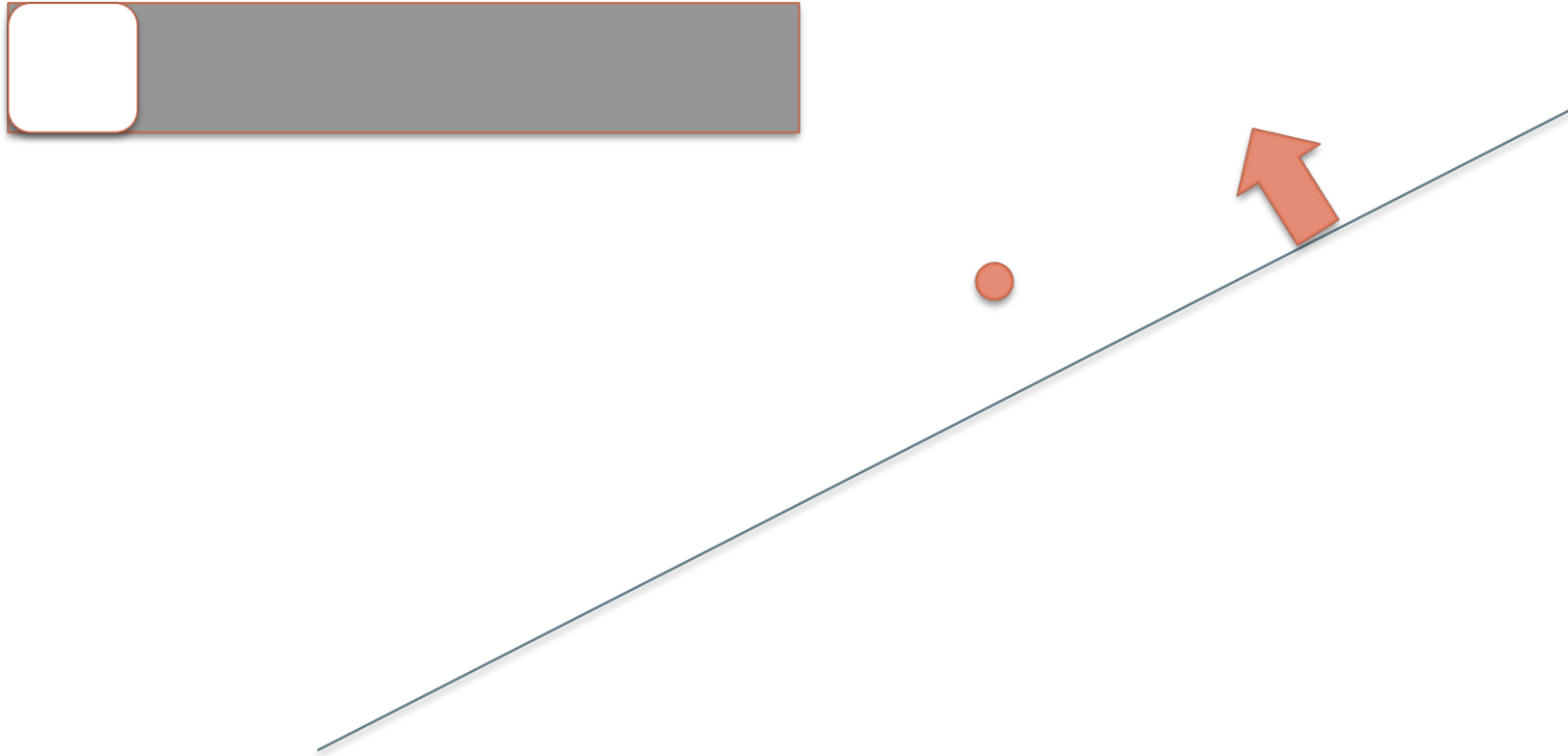
Locality Sensitive Hashing



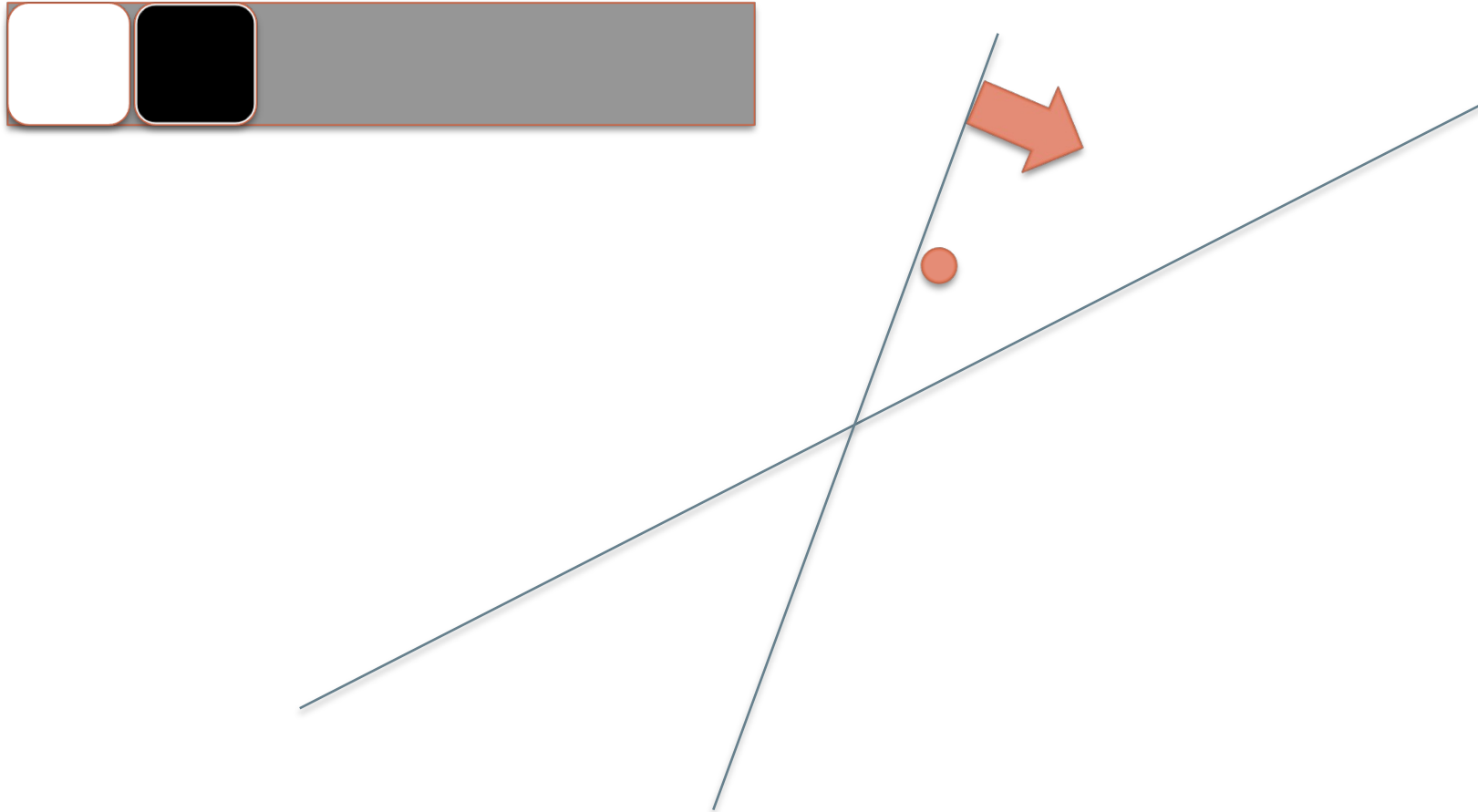
Locality Sensitive Hashing



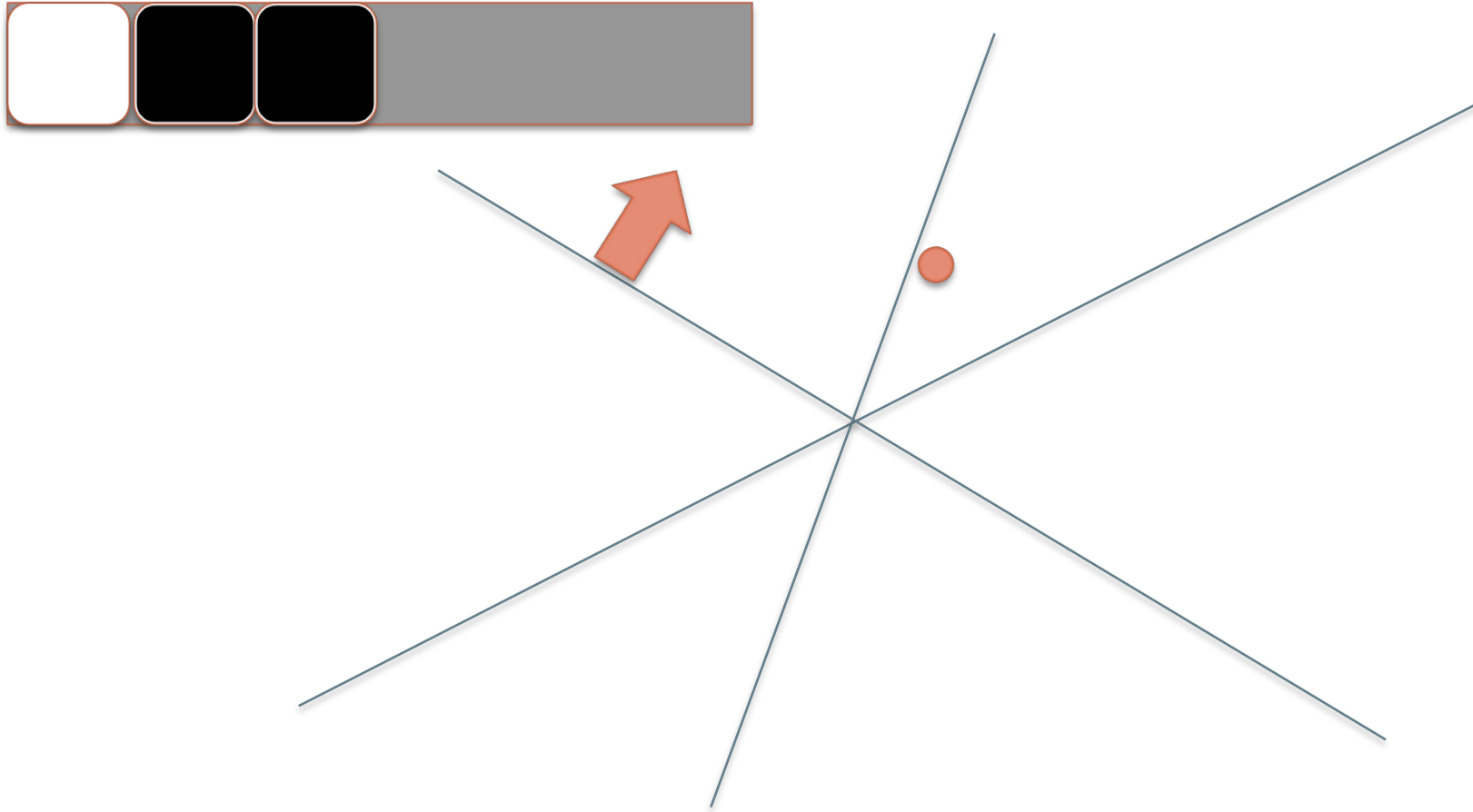
Locality Sensitive Hashing



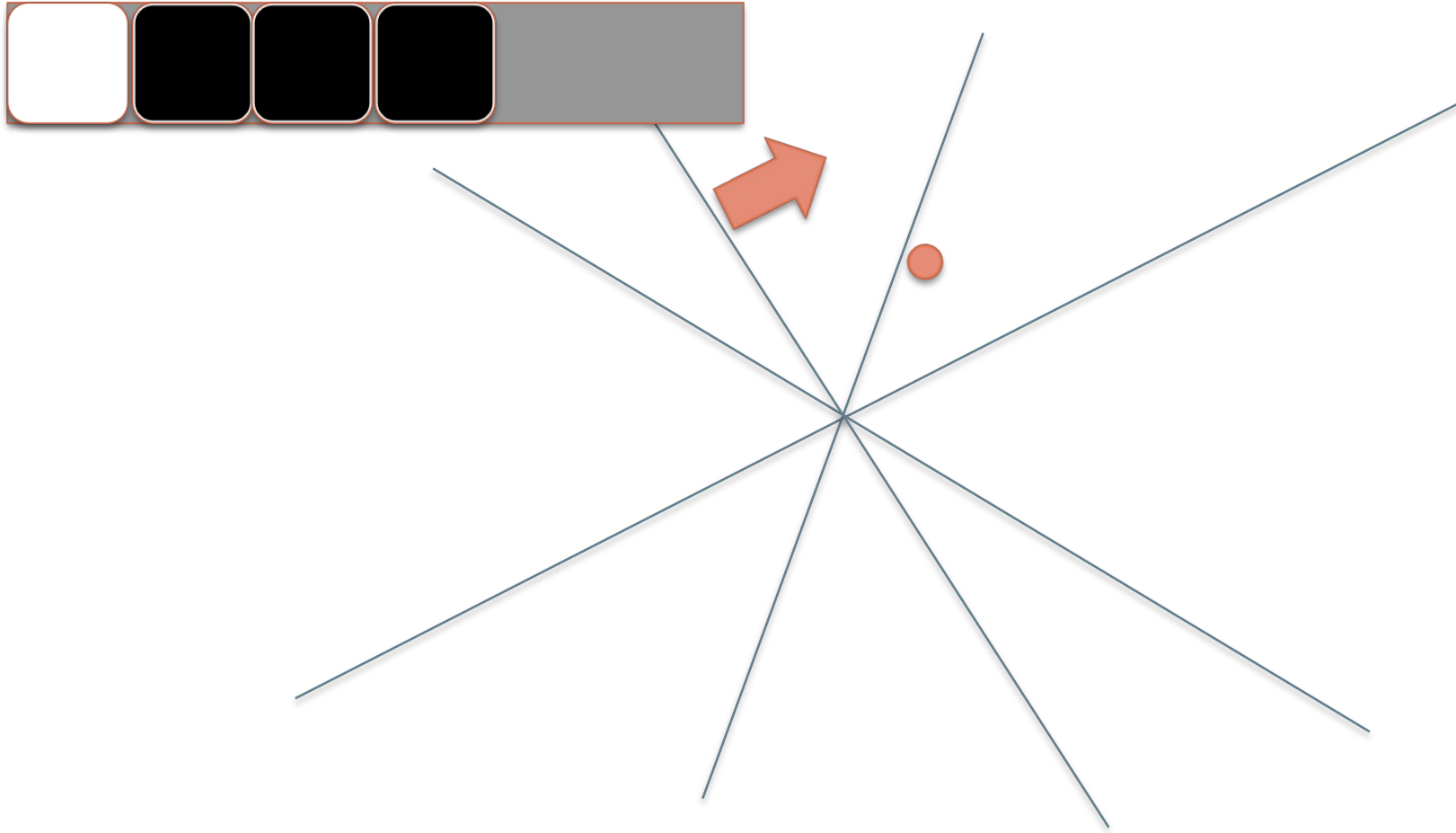
Locality Sensitive Hashing



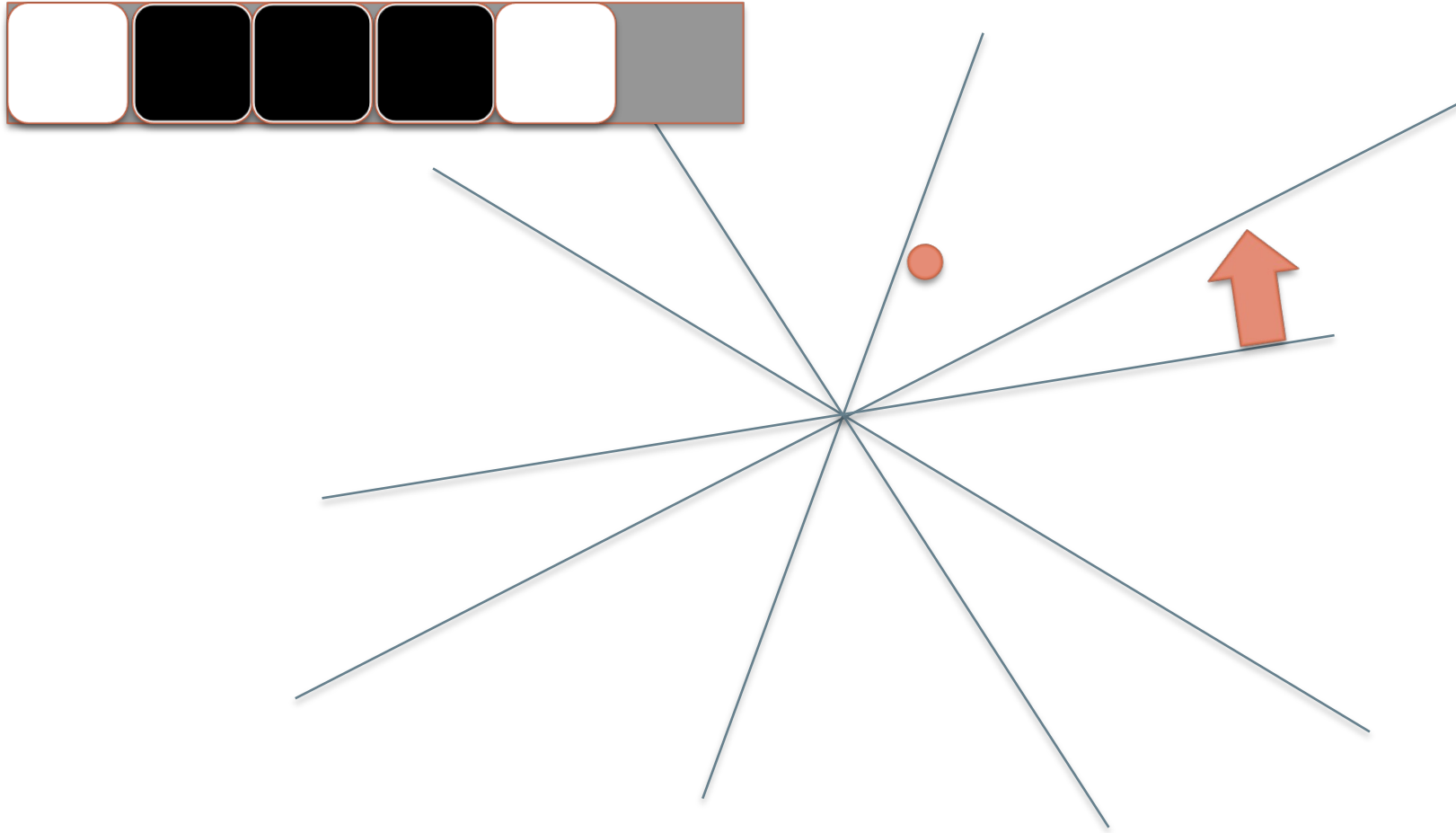
Locality Sensitive Hashing



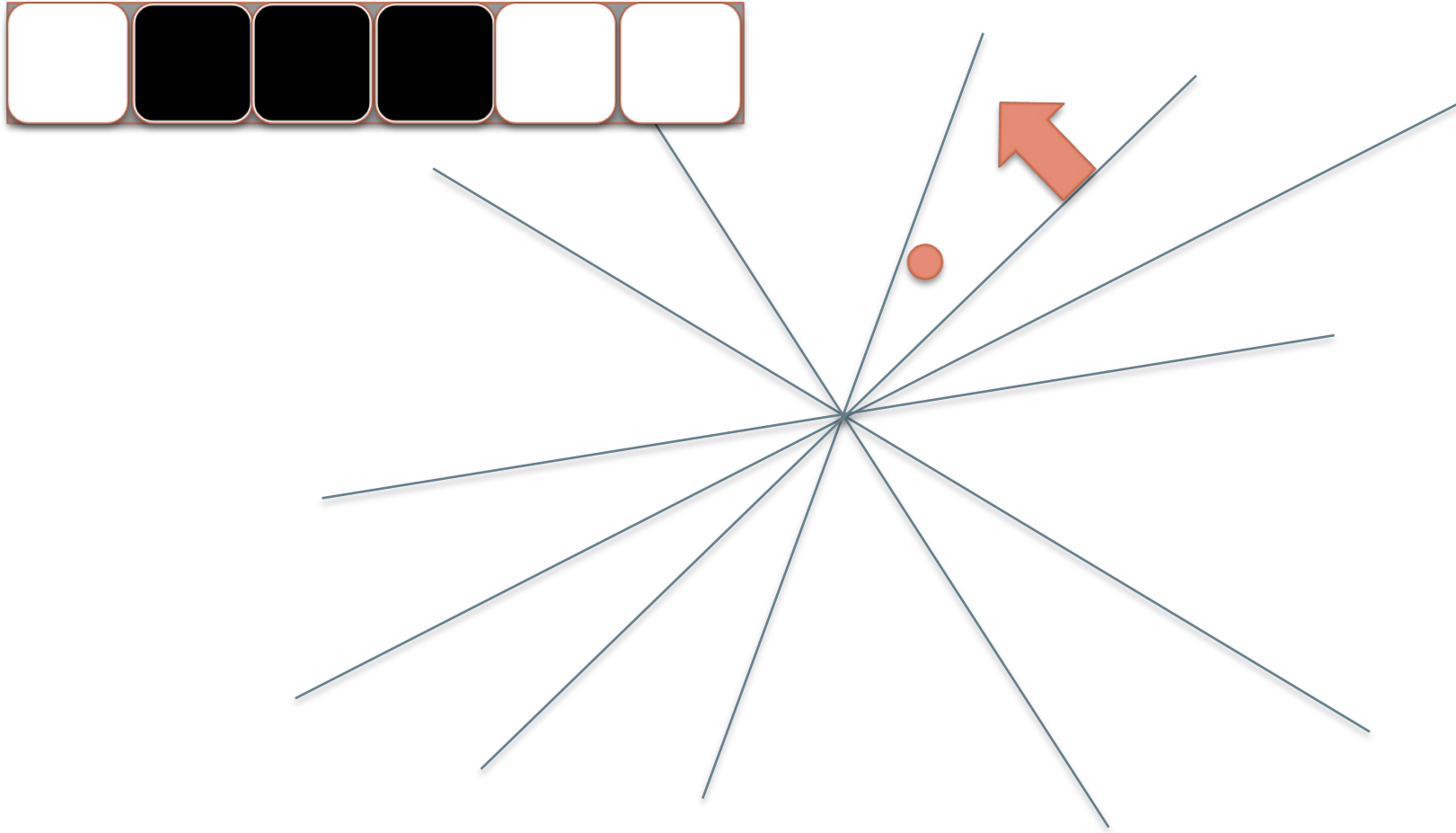
Locality Sensitive Hashing



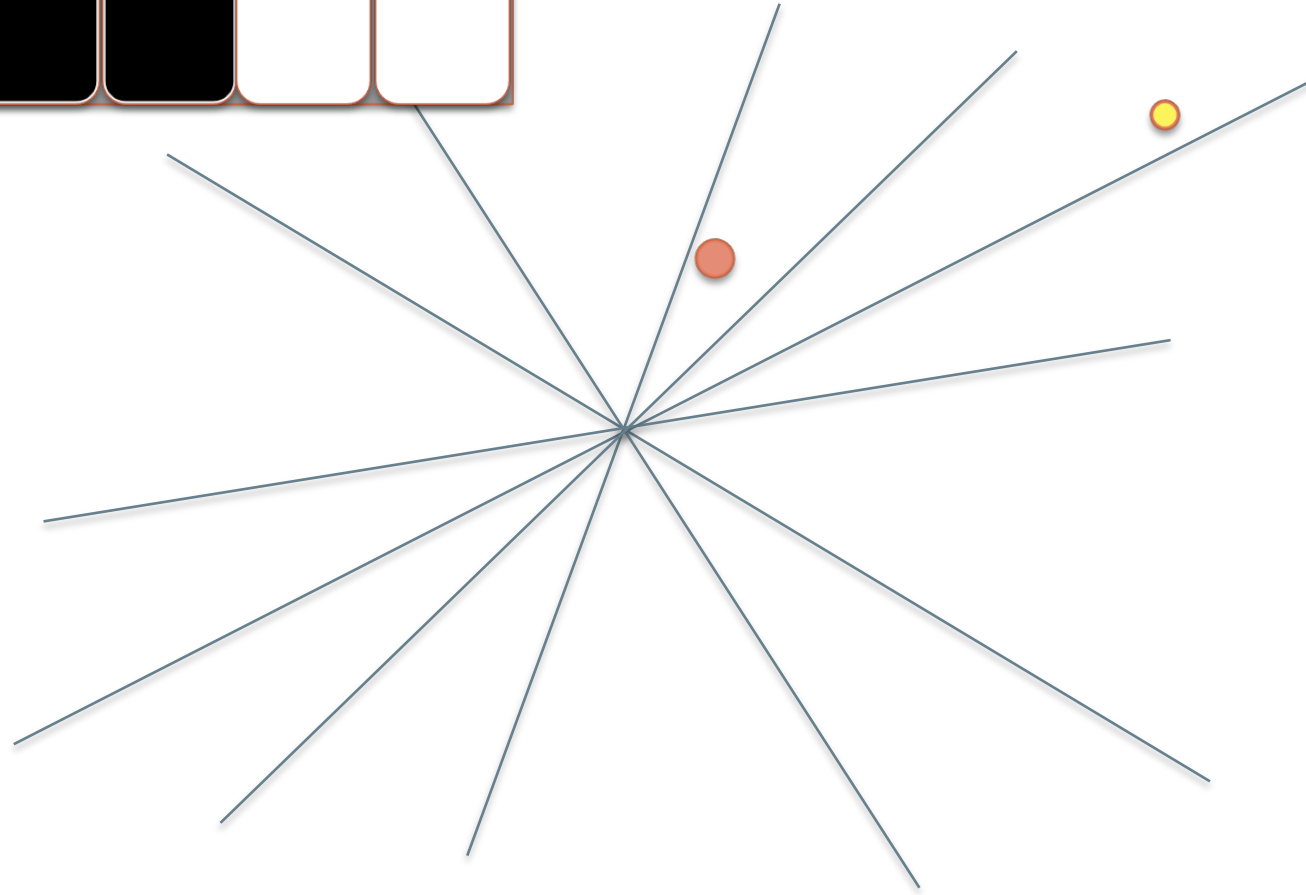
Locality Sensitive Hashing



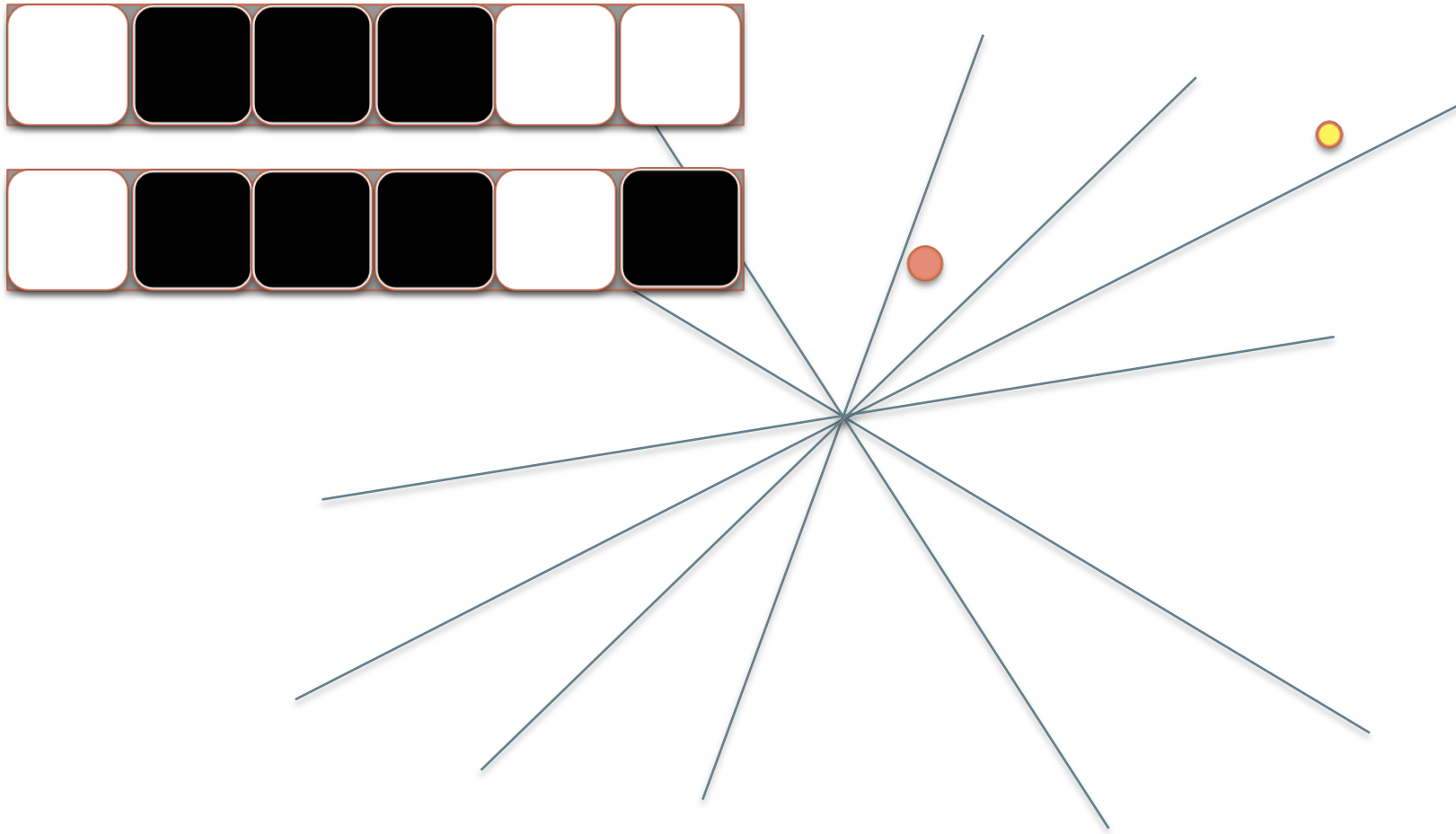
Locality Sensitive Hashing



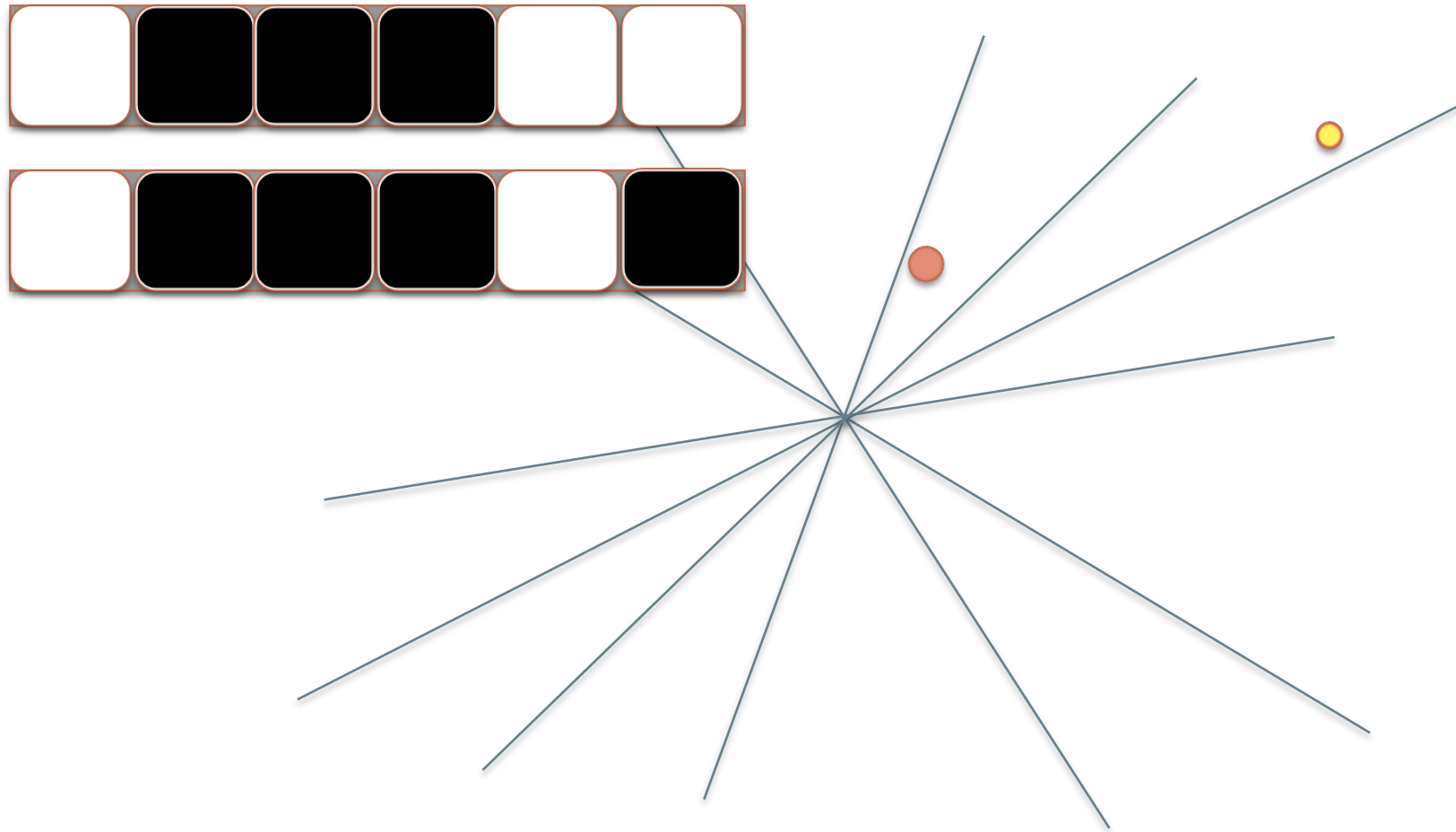
Locality Sensitive Hashing



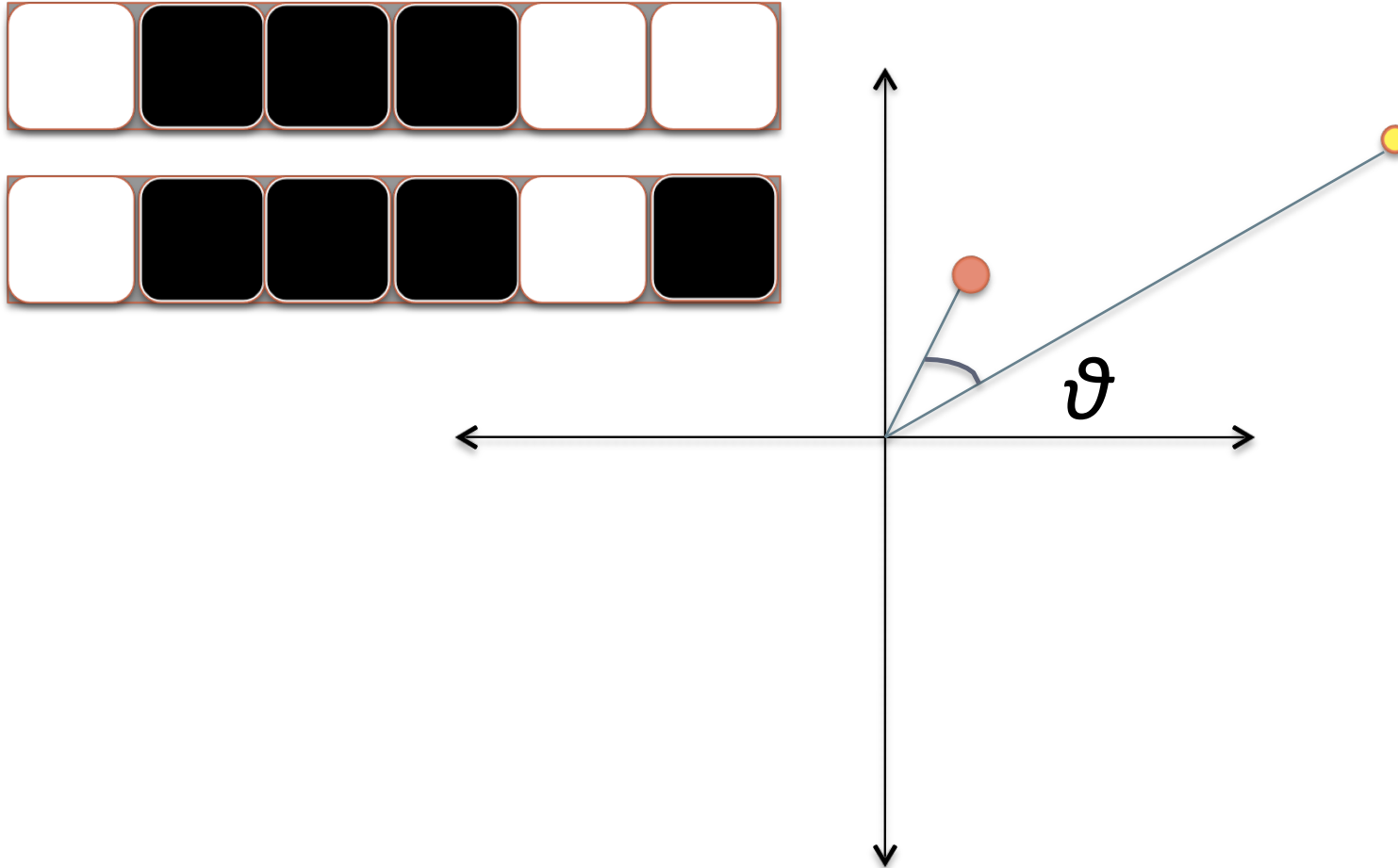
Locality Sensitive Hashing



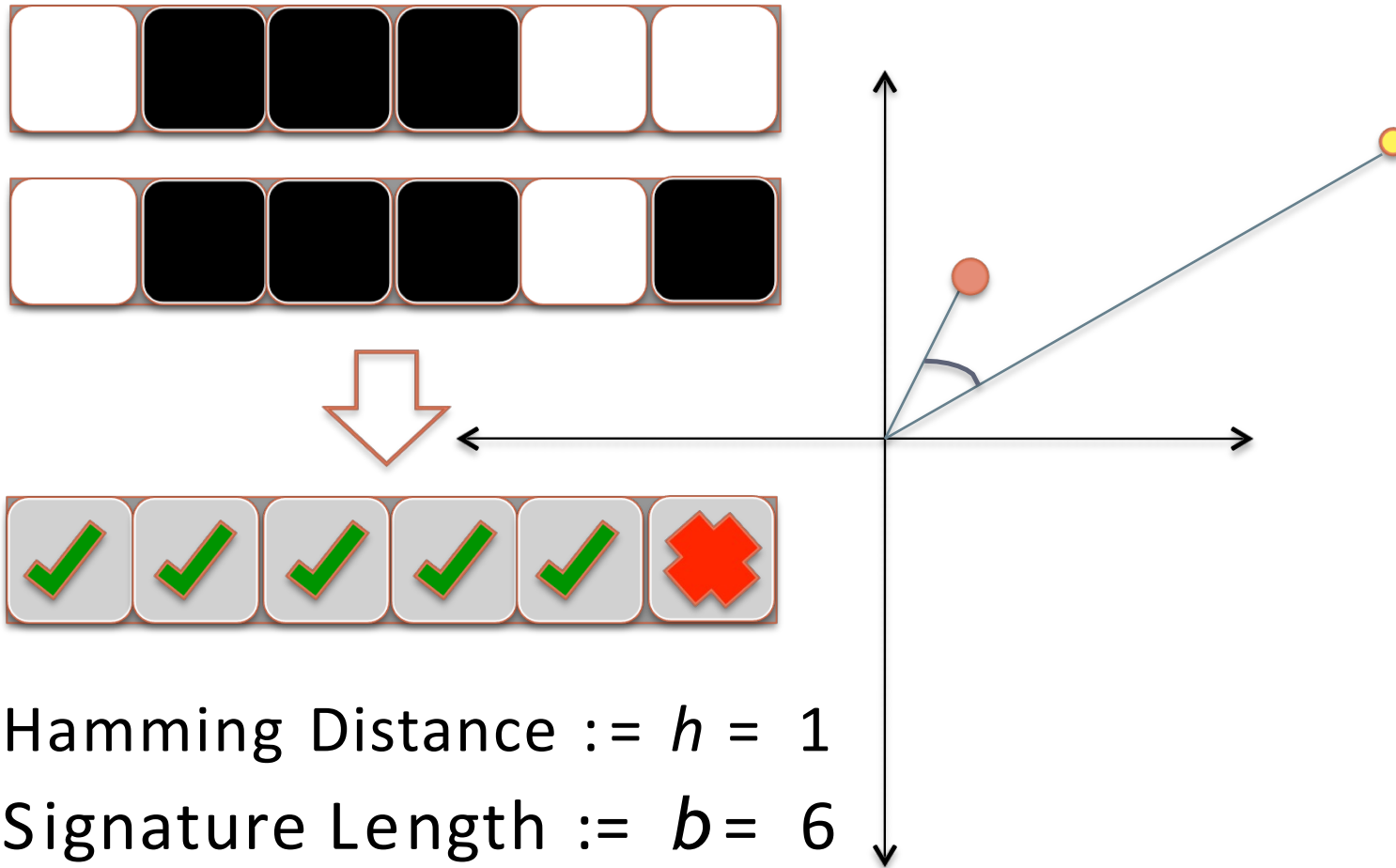
Locality Sensitive Hashing



Locality Sensitive Hashing



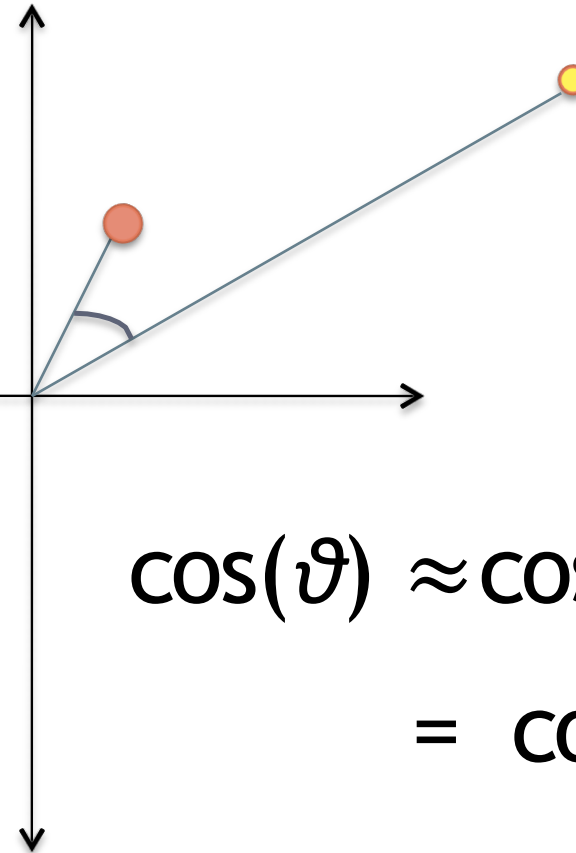
Locality Sensitive Hashing



Locality Sensitive Hashing



Hamming Distance $:= h = 1$
Signature Length $:= b = 6$



$$\begin{aligned}\cos(\vartheta) &\approx \cos\left(\frac{h}{b}\pi\right) \\ &= \cos\left(\frac{1}{6}\pi\right)\end{aligned}$$

Compute sketches for cosine similarity

- *For each vector, we compute the bit of the signatures with:*
 - *if $\bar{v} \cdot \bar{r} \geq 0 \rightarrow 1$
else $\rightarrow 0$*
 - *\bar{r} is a random projection, i.e., a random vector of 0s and 1s*
- Use XOR to count the bits that differ
 - i.e., the time that the random permutation “cut” the vectors
- SUM the XOR’ed bits and then

$$\cos(\vartheta) \approx \cos\left(\frac{h}{b}\pi\right)$$

SUM(XOR(...))

Signature Length

Bibliography

1. Xiao C, Wang W, Lin X, Yu JX, Wang G. Efficient similarity joins for near-duplicate detection. *ACM Transactions on Database Systems (TODS)*. 2011 Aug 26;36(3):1-41.
2. Leskovec J, Rajaraman A, Ullman JD. Mining of massive data sets. Cambridge university press; 2020 Jan 9.
3. Agrawal R, Faloutsos C, Swami A. Efficient similarity search in sequence databases. In *Foundations of Data Organization and Algorithms: 4th International Conference, FODO'93 Chicago, Illinois, USA, October 13–15, 1993 Proceedings 4* 1993 (pp. 69-84). Springer Berlin Heidelberg.
4. Andoni A, Indyk P, Laarhoven T, Razenshteyn I, Schmidt L. Practical and optimal LSH for angular distance. *Advances in neural information processing systems*. 2015;28.
5. Van Durme B, Lall A. Online generation of locality sensitive hash signatures. In *Proceedings of the ACL 2010 conference short papers* 2010 Jul (pp. 231-235).
6. Nargesian F, Zhu E, Miller RJ, Pu KQ, Arocena PC. Data lake management: challenges and opportunities. *Proceedings of the VLDB Endowment*. 2019 Aug 1;12(12):1986-9.
7. Khatiwada A, Shrager R, Gatterbauer W, Miller RJ. Integrating data lake tables. *Proceedings of the VLDB Endowment*. 2022 Dec 1;16(4):932-45.