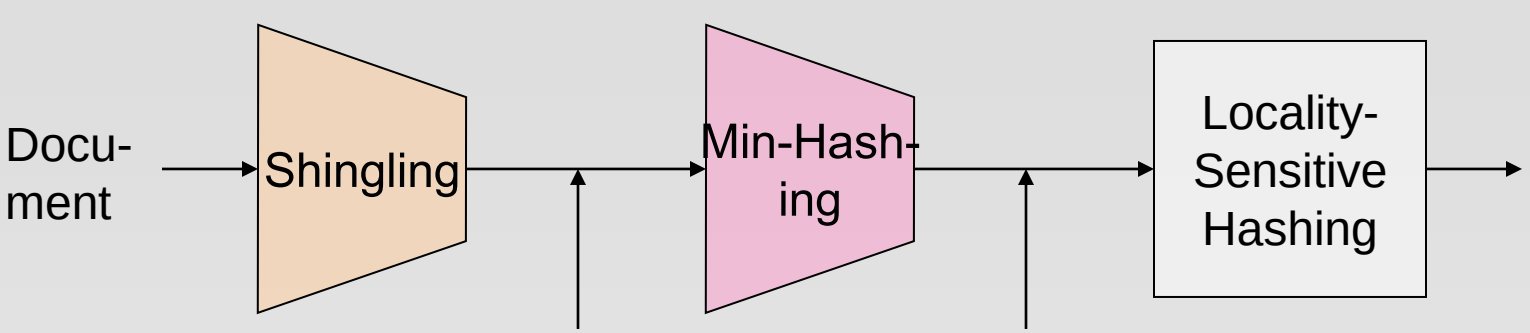
COMP9313: Big Data Management

Lecturer: Xin Cao

Course web site: http://www.cse.unsw.edu.au/~cs9313/

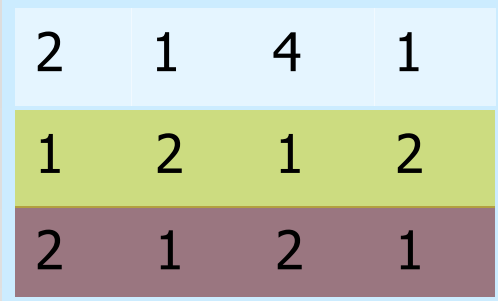
Chapter 7.2: Finding Similar Items



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

Focus on pairs of signatures likely to be from similar documents

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**

Candidates from Min-Hash

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

- **Pick a similarity threshold *s* (0 < s < 1)**

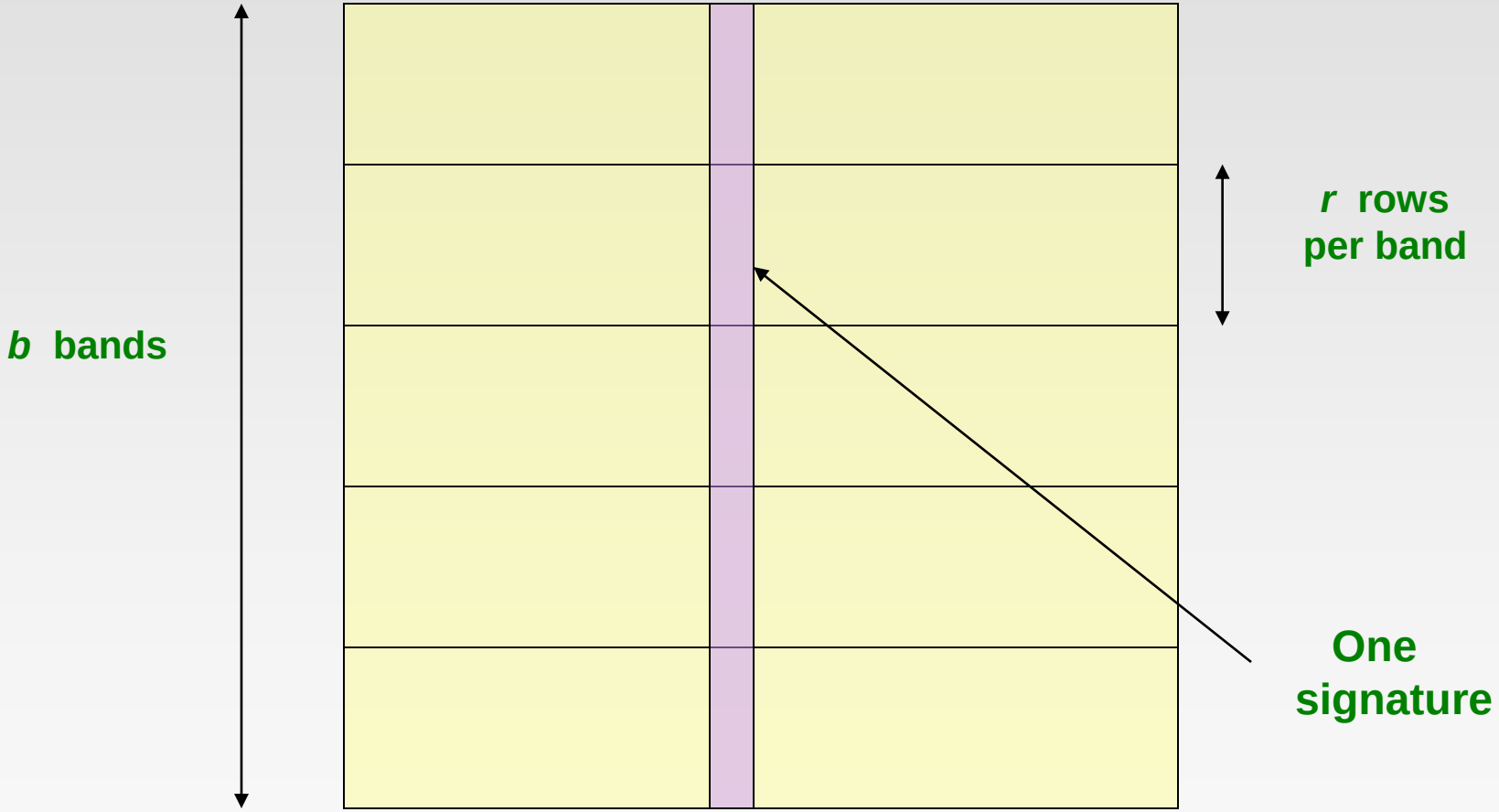
- Columns ***x*** and ***y*** of ***M*** are a **candidate pair** if their signatures agree on at least fraction ***s*** of their rows:
  M* (*i*, *x*) = *M* (*i*, *y*)** for at least frac. ***s values of ***i***
  - We expect documents ***x*** and ***y*** to have the same (Jaccard) similarity as their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

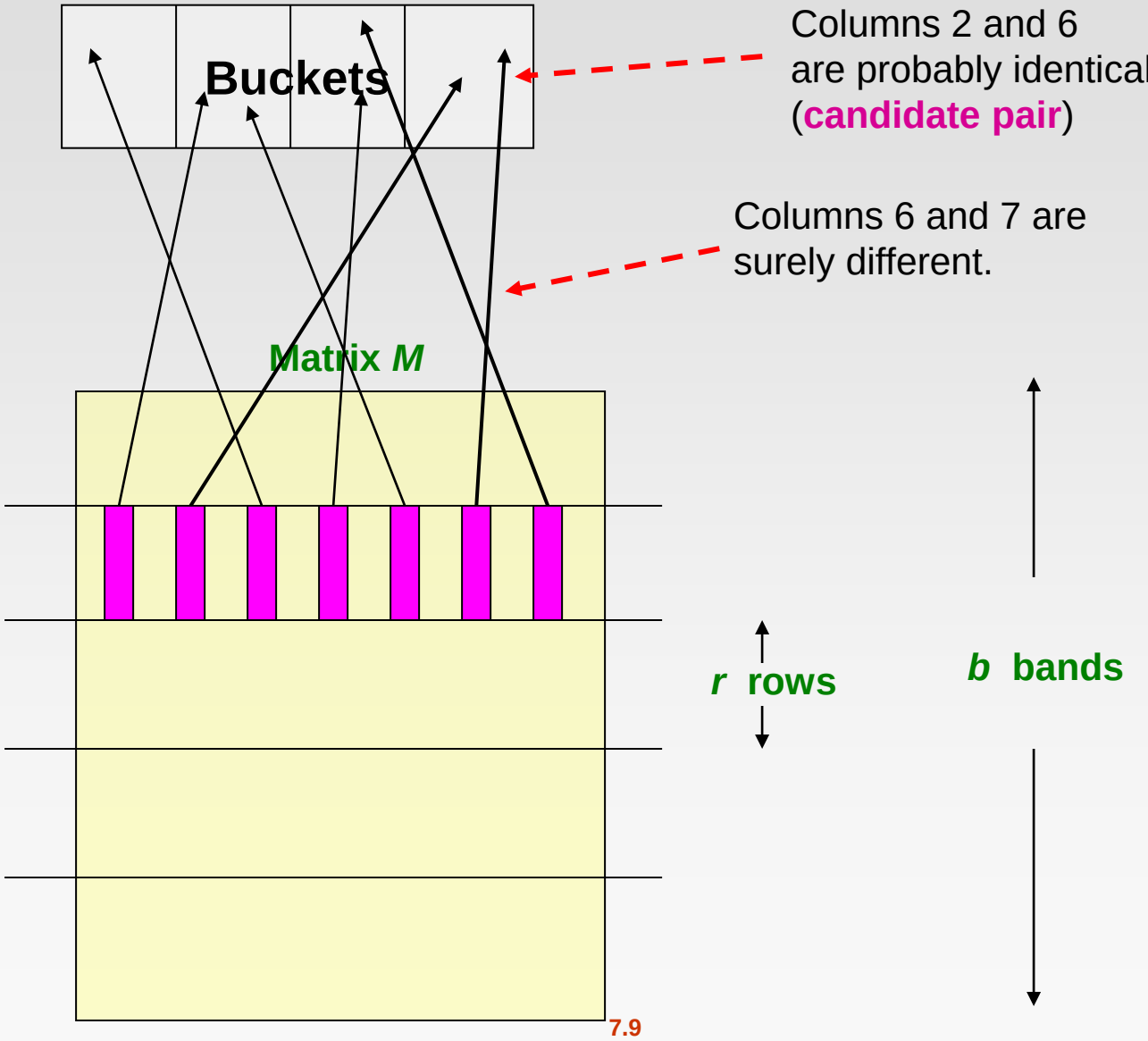
Partition *M* into *b* Bands



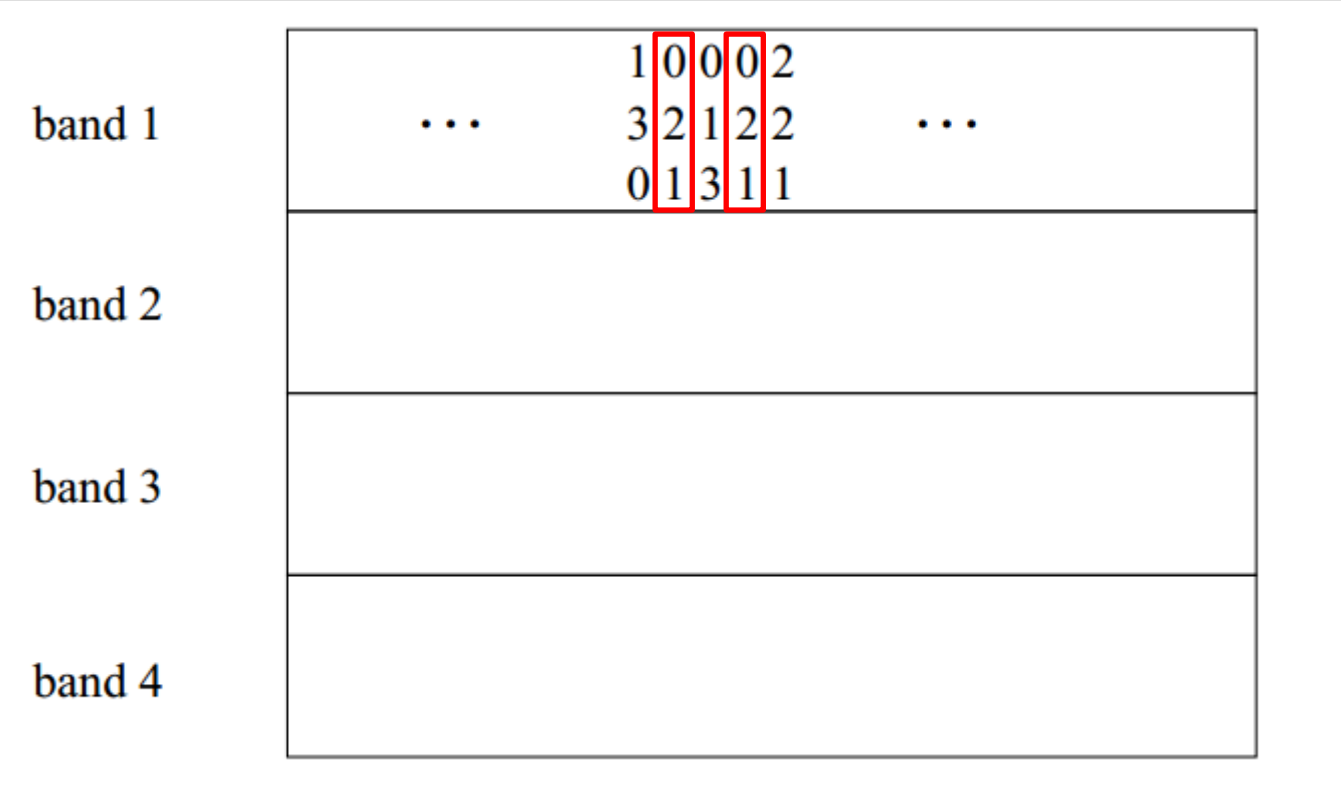
Partition M into Bands

- 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



Hashing Bands



- Regardless of what those columns look like in the other three bands, this pair of columns will be a candidate pair
- Two columns that do not agree in band 1 have three other chances to become a candidate pair; they might be identical in any one of these other 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

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

- **Find pairs of ≥ s**=0.8 similarity, set **b**=20, **r**=5
- **Assume:** sim(C_1, C_2) = 0.8
  - Since sim(C_1, C_2) ≥ **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

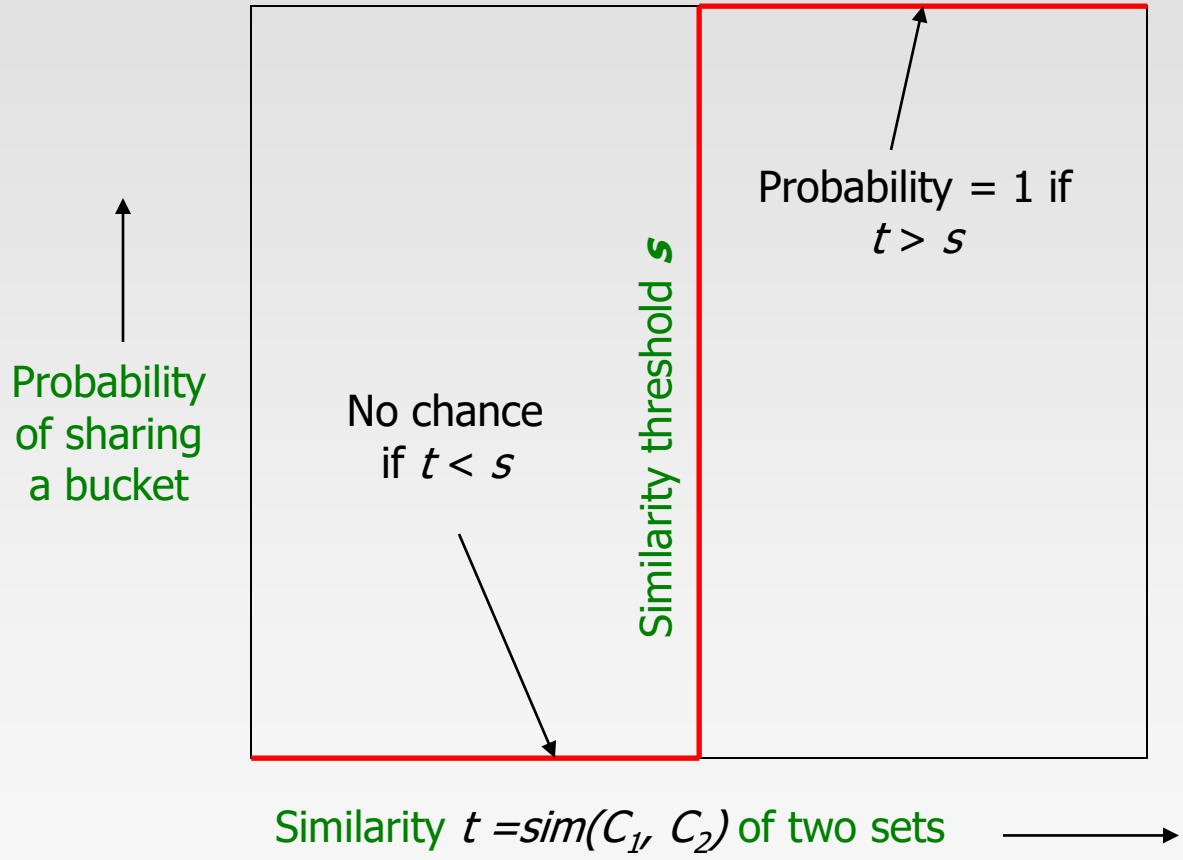
- **Find pairs of ≥ s**=0.8 similarity, set **b**=20, **r**=5
- **Assume:** sim(C_1, C_2) = 0.3
  - Since sim(C_1, C_2) < **s** we want C_1, C_2 to hash to **NO common buckets** (all bands should be different)
- **Probability C_1, C_2 identical in one particular band:** $(0.3)^5$ = 0.00243
- Probability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

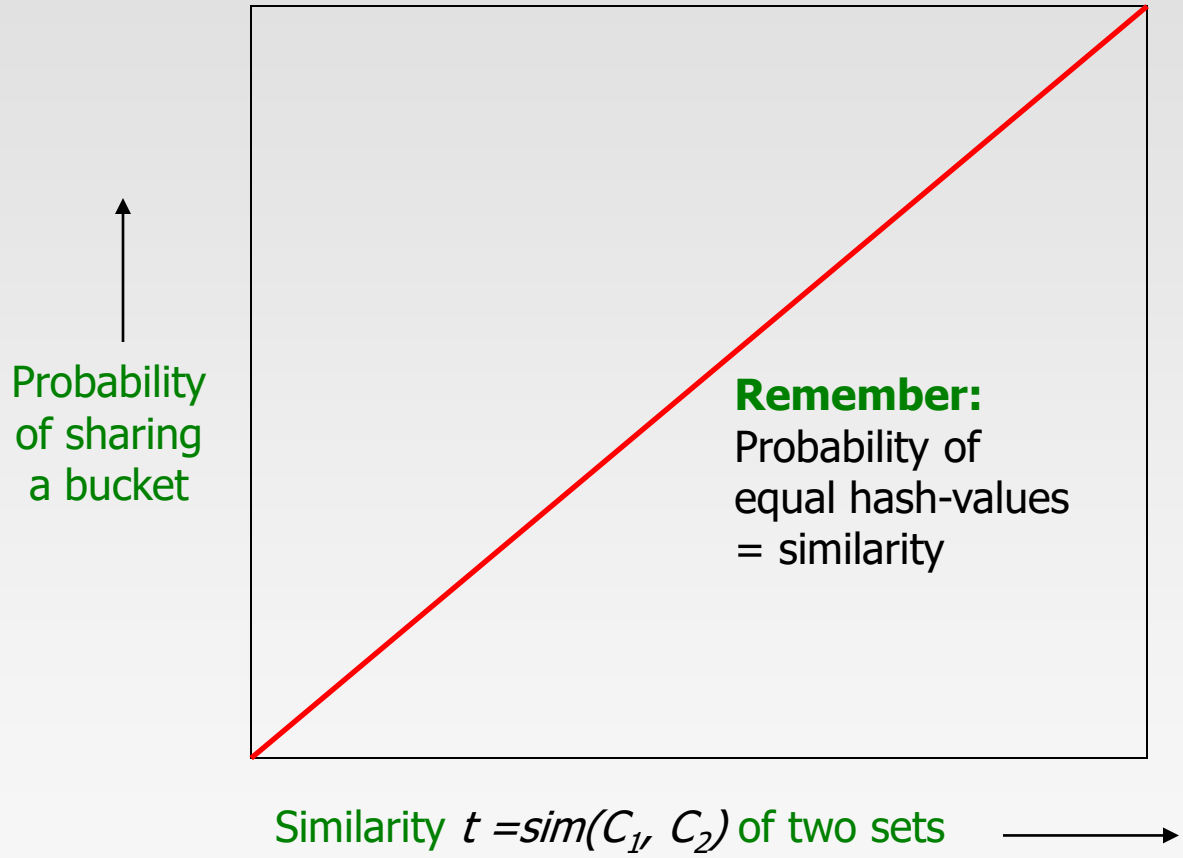
- **Pick:**
  - The number of Min-Hashes (rows of ***M***)
  - The number of bands ***b***, and
  - The number of rows ***r*** per band to balance false positives/negatives

- **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

- The probability that the minhash signatures for the documents agree in any one particular row of the signature matrix is ***t*** ($sim(C_1, C_2)$)

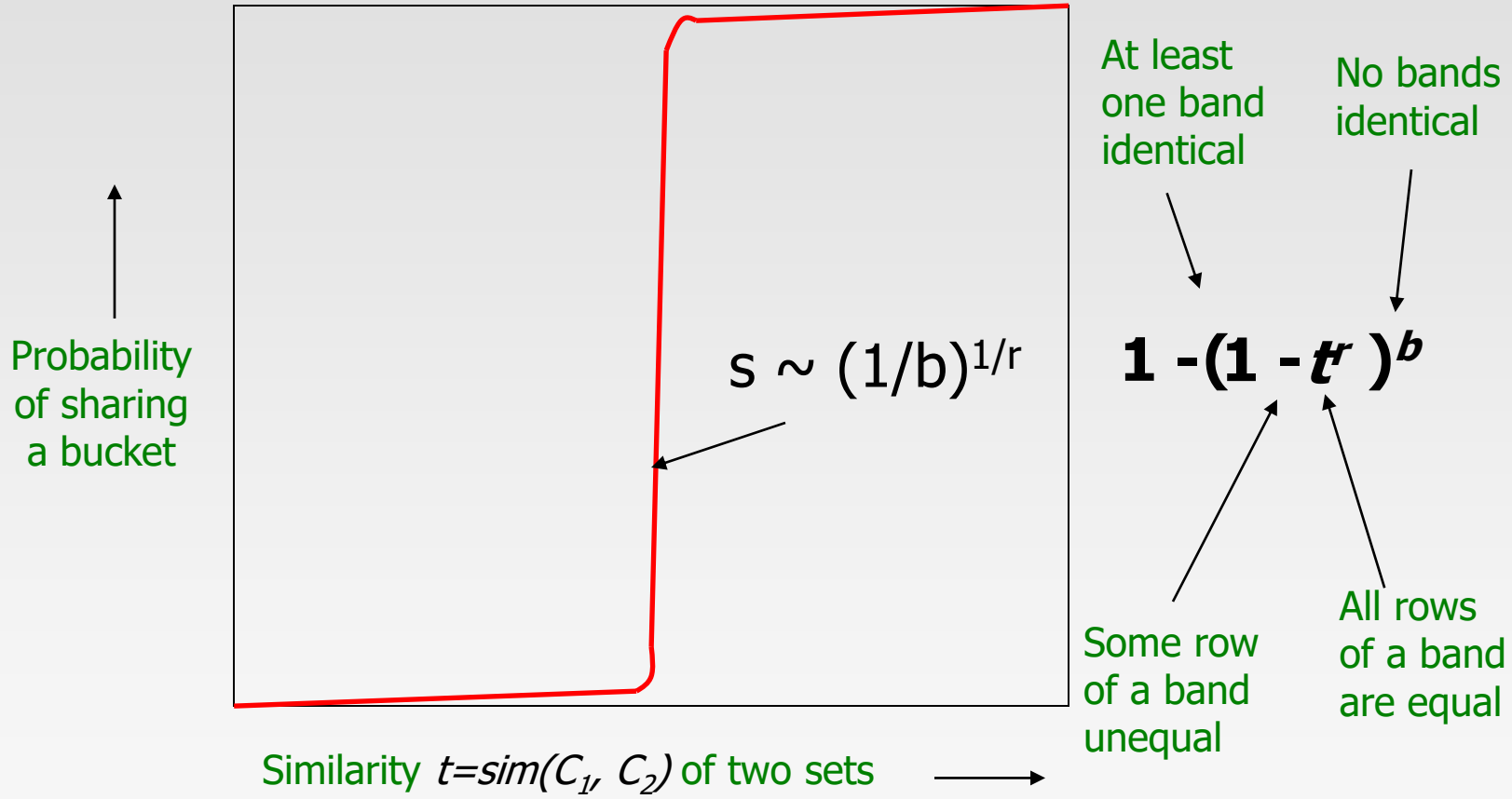
- Pick any band (***r*** rows)
  - Prob. that all rows in band equal = t^r
  - Prob. that some row in band unequal = $1 - t^r$

- 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

这种band的思想，使得这个判断函数具有很明显的断崖式结果，这就是我们想要的
我们希望当相似度低于某个值的时候，被hash到一个桶的概率很低



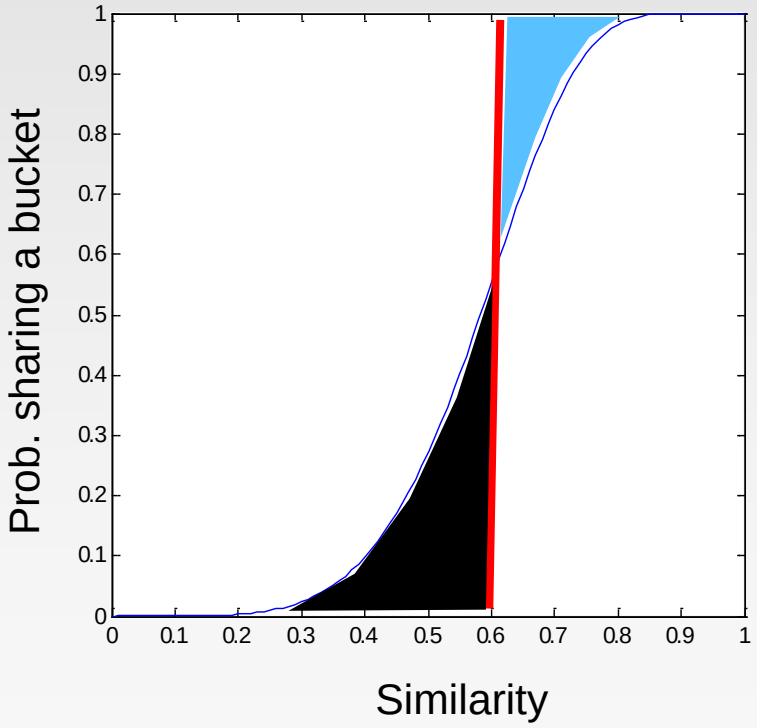
Example: $b = 20$, $r = 5$

- 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
Black 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

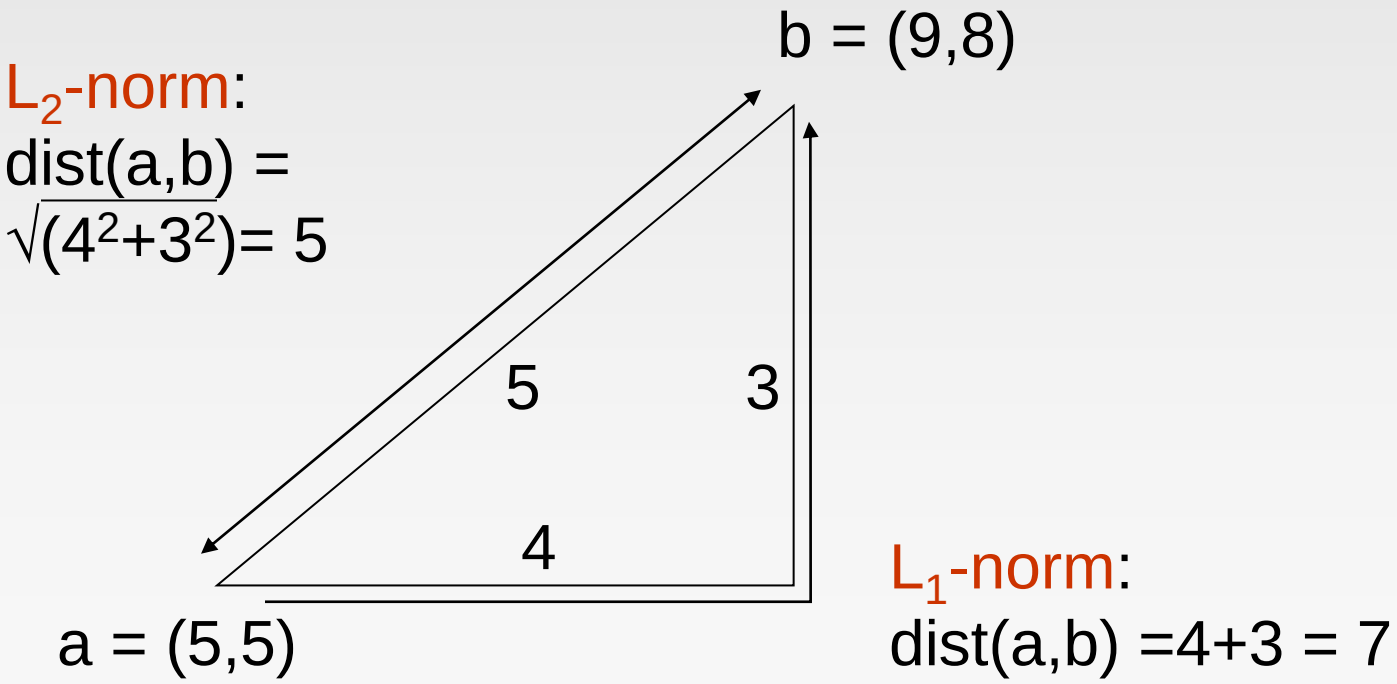
- **Shingling:** Convert documents to sets
  - We used hashing to assign each shingle an ID
- **Min-Hashing:** Convert large sets to short signatures, while preserving similarity
  - We used **similarity preserving hashing** to generate signatures with property $\mathbf{Pr[}h_\pi\mathbf{(C_1) =} h_\pi\mathbf{(C_2)] =} sim\mathbf{(C_1, C_2)}$
  - 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**

Distance Measures

- Generalized LSH is based on some kind of "distance" between points.
  - Similar points are "close."
- Example: Jaccard similarity is not a distance; 1 minus Jaccard similarity is.
- *d* is a *distance measure* if it is a function from pairs of points to real numbers such that:
  1. $d(x,y) \geq 0$.
  2. $d(x,y) = 0$ iff $x = y$.
  3. $d(x,y) = d(y,x)$.
  4. $d(x,y) \leq d(x,z) + d(z,y)$ (*triangle inequality*).

Some Euclidean Distances

- *L_2 norm*: d(x,y) = square root of the sum of the squares of the differences between *x* and *y* in each dimension.
  - The most common notion of "distance."
- *L_1 norm*: sum of the differences in each dimension.
  - *Manhattan distance* = distance if you had to travel along coordinates only.



Some Non-Euclidean Distances

- *Jaccard distance* for sets = 1 minus Jaccard similarity.
- *Cosine distance* for vectors = angle between the vectors.
- *Edit distance* for strings = number of inserts and deletes to change one string into another.

Cosine Distance

- Think of a point as a vector from the origin [0,0,…,0] to its location.
- Two points' vectors make an angle, whose cosine is the normalized dot-product of the vectors: $p_1.p_2/|p_2||p_1|$.
  - Example: p_1 = [1,0,2,-2,0]; p_2 = [0,0,3,0,0].
  - $p_1.p_2 = 6$; $|p_1| = |p_2| = \sqrt{9} = 3$.
  - $\cos(\theta) = 6/9$; θ is about 48 degrees.

Edit Distance

- The *edit distance* of two strings is the number of inserts and deletes of characters needed to turn one into the other.
- An equivalent definition: d(x,y) = |x| + |y| - 2|LCS(x,y)|.
  - LCS = *longest common subsequence* = any longest string obtained both by deleting from *x* and deleting from *y*.
- Example:
  - *x* = *abcde* ; *y* = *bcduve*.
  - Turn *x* into *y* by deleting *a*, then inserting *u* and *v* after *d*.
    - Edit distance = 3.
  - Or, computing edit distance through the LCS, note that LCS(x,y) = *bcde*.
  - Then:|x| + |y| - 2|LCS(x,y)| = 5 + 6 –2*4 = 3 = edit distance.

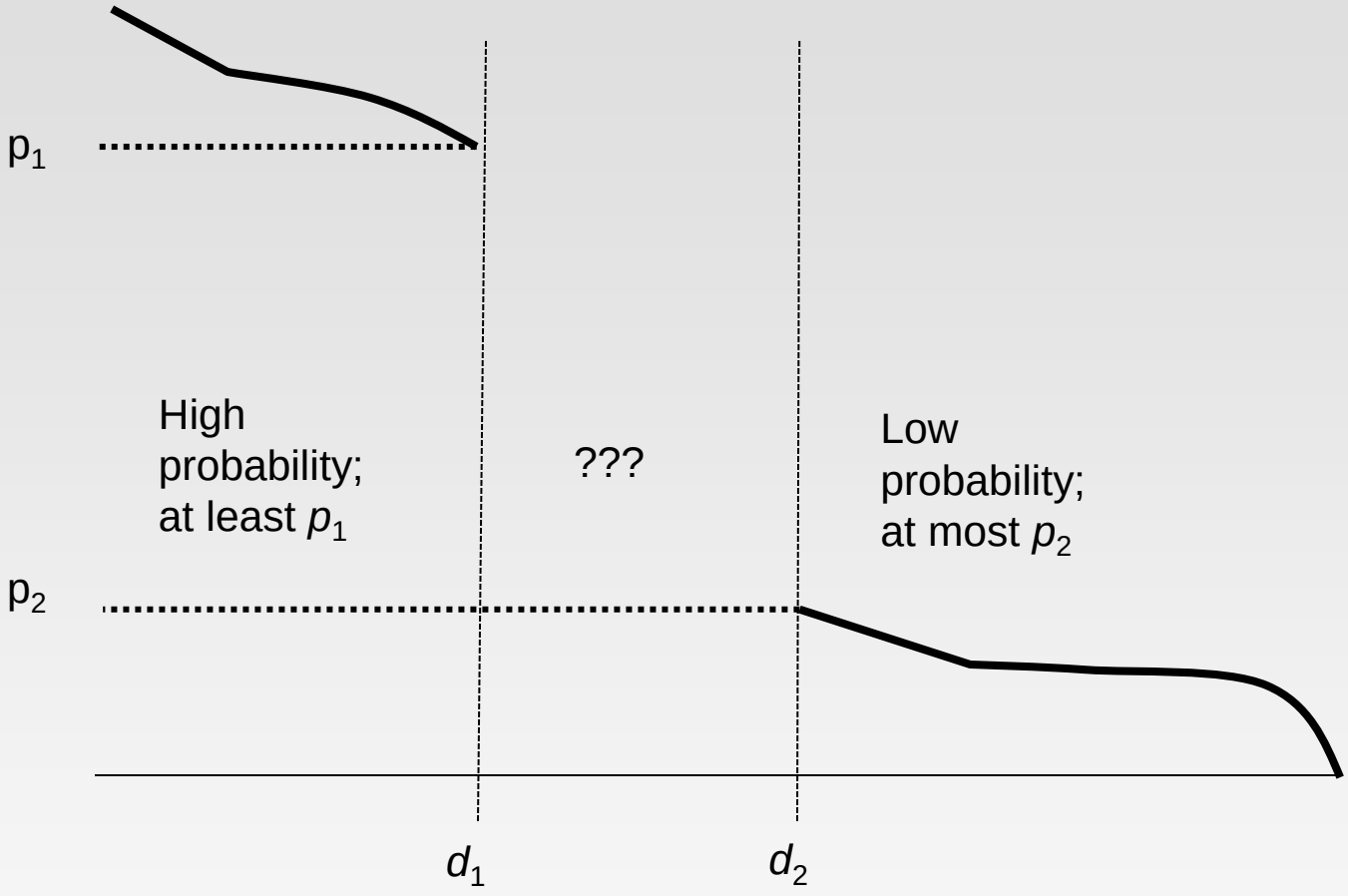
Hash Functions Decide Equality

- There is a subtlety about what a “hash function” is, in the context of LSH families.
- A hash function h really takes two elements x and y, and returns a decision whether x and y are candidates for comparison.
- Example: the family of minhash functions computes minhash values and says “yes” iff they are the same.
- Shorthand: “h(x) = h(y)” means h says “yes” for pair of elements x and y.

LSH Families Defined

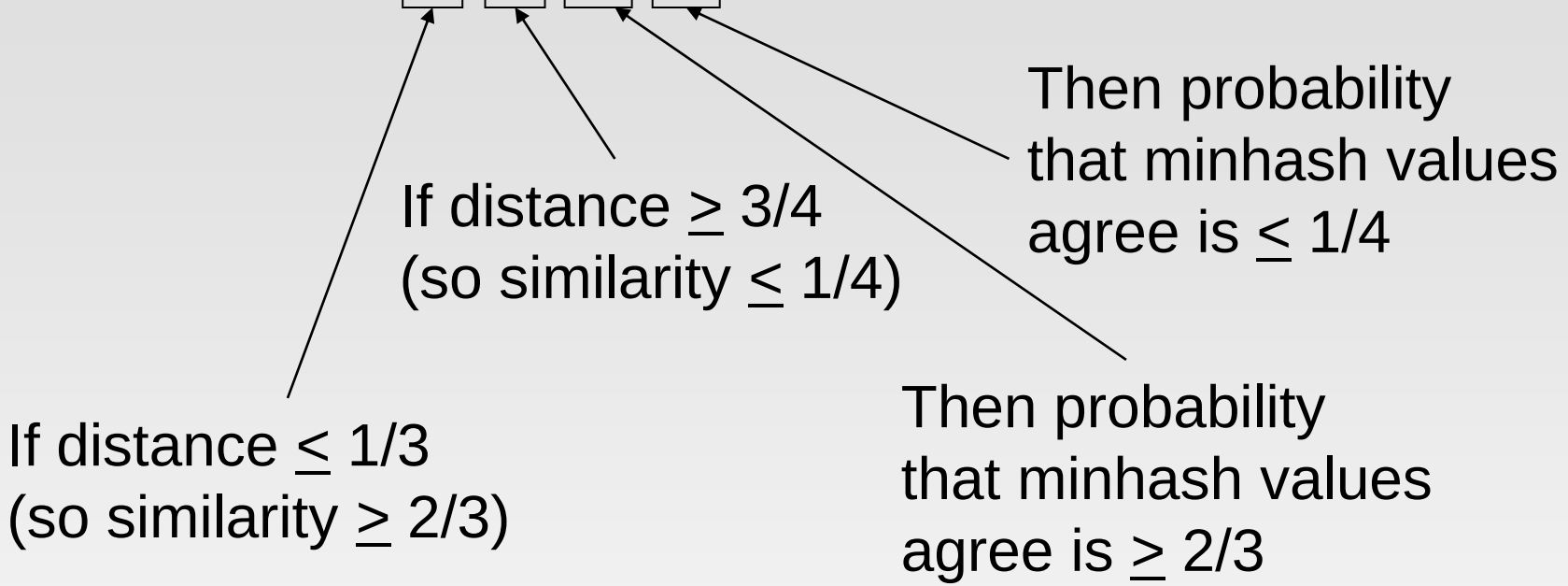
- Suppose we have a space *S* of points with a distance measure *d*.
- A family **H** of hash functions is said to be (d_1,d_2,p_1,p_2)-*sensitive* if for any *x* and *y* in *S*:
  1. If $d(x,y) \leq d_1$, then the probability over all *h* in **H**, that h(x) = h(y) is at least p_1.
  2. If $d(x,y) \geq d_2$, then the probability over all *h* in **H**, that h(x) = h(y) is at most p_2.

LS Families: Illustration



Example: LS Family – (2)

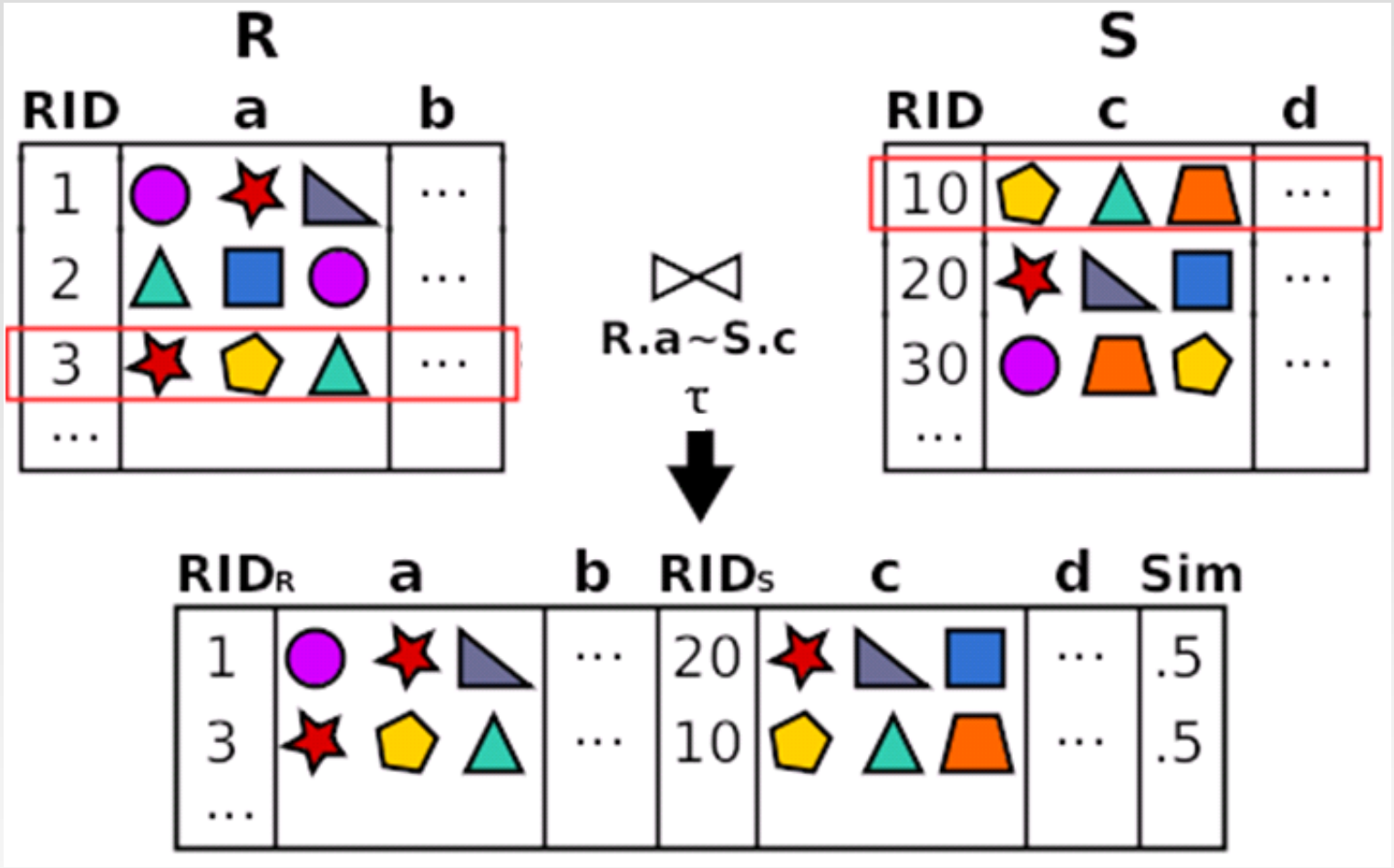
- **Claim**: **H** is a (1/3, 3/4, 2/3, 1/4)-sensitive family for *S* and *d*.



For Jaccard similarity, minhashing gives us a $(d_1,d_2,(1-d_1),(1-d_2))$-sensitive family for any $d_1 < d_2$.

Part 2: Exact Approach to Finding Similar Items

Set-Similarity Join



Finding pairs of records with a **similarity** on their join attributes > t

Set-Similarity Join

- Given two collections of records R and S, a similarity function sim(., .), and a threshold τ, the set similarity join between R and S, is to find all record pairs r (from R) and s (from S), such that sim(r, s) >= τ.

id	set
r_1	$\{e_1, e_4, e_5, e_6\}$
r_2	$\{e_2, e_3, e_6\}$
r_3	$\{e_4, e_5, e_6\}$

(a) $\mathcal{R}$ sets

id	set
s_1	$\{e_1, e_4, e_6\}$
s_2	$\{e_2, e_5, e_6\}$
s_3	$\{e_3, e_5\}$

(b) $\mathcal{S}$ sets

- Given the above example, and set τ=0.5, the results are: (r1, s1) (similarity 0.75), (r2, s2) (similarity 0.5), (r3, s1) (similarity 0.5), (r3, s2) (similarity 0.5).
- LSH can solve this problem approximately.

Application: Record linkage

Table R

Star
Keanu Reeves
Samuel Jackson
Schwarzenegger
…

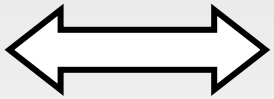
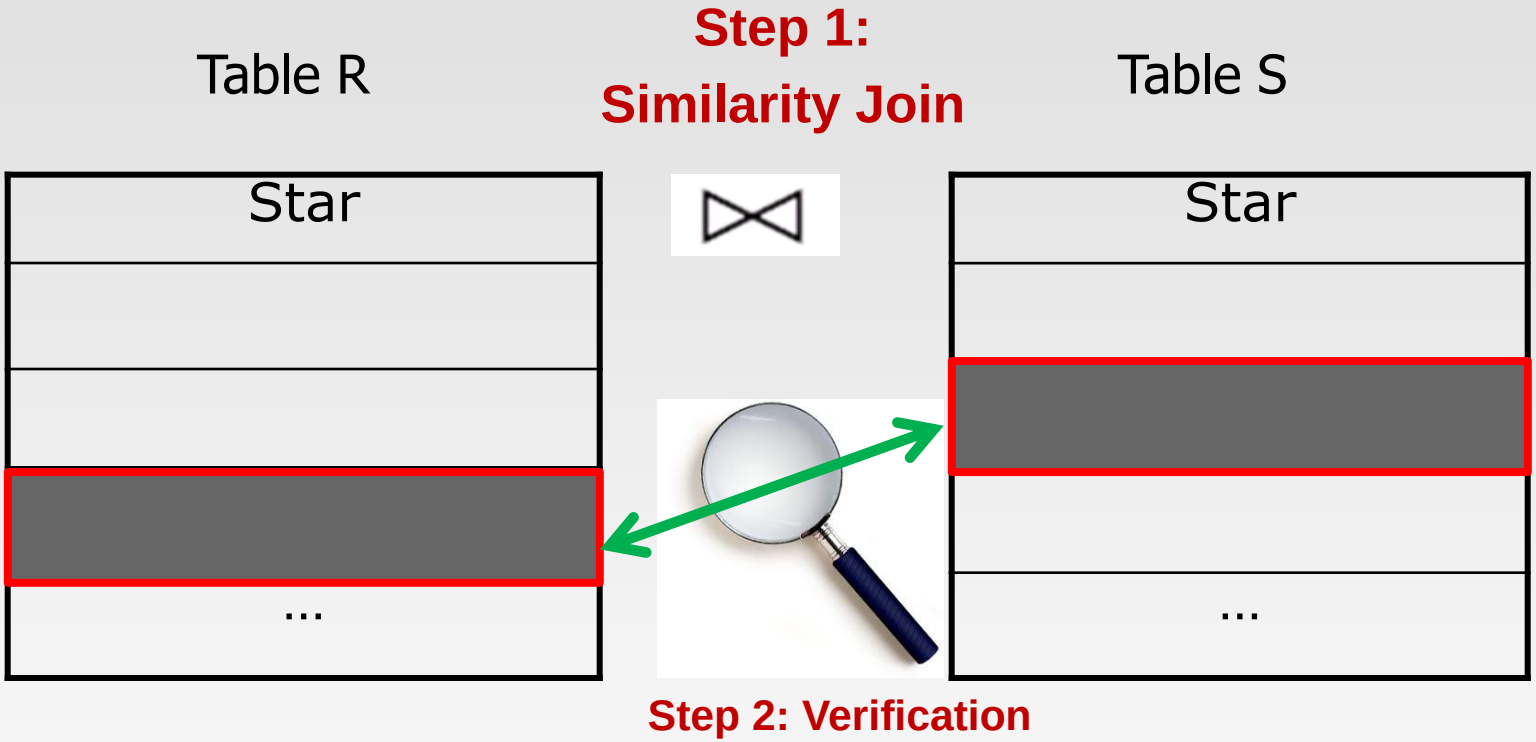


Table S

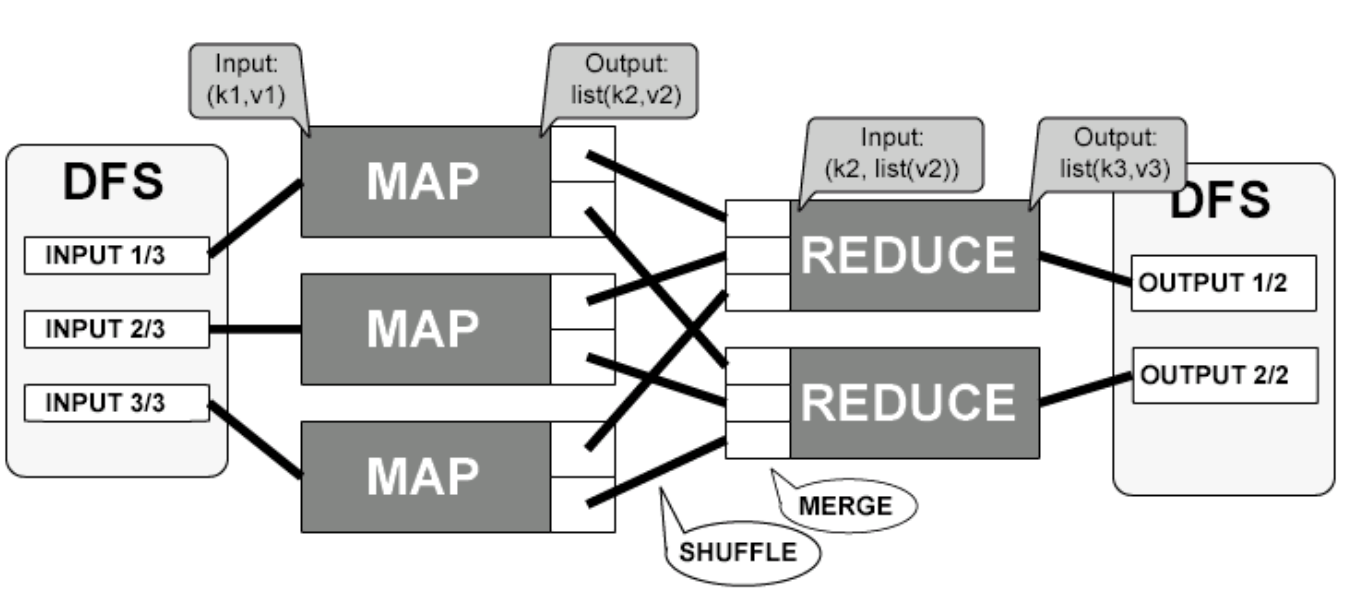
Star
Keanu Reeves
Samuel L. Jackson
Schwarzenegger
…

Two-step Solution



A Naïve Solution

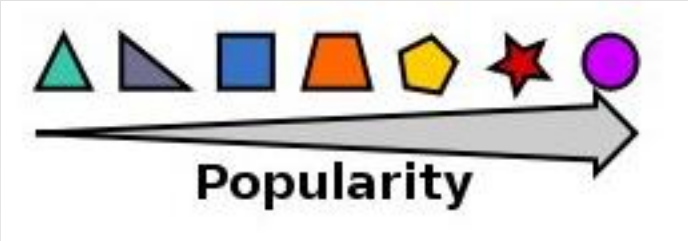
- Map: <23, (a,b,c)> → (a, 23), (b, 23), (c, 23)
- Reduce: (a,23),(a,29),(a,50), … → Verify each pair (23, 29), (23, 50), (29, 50) … …



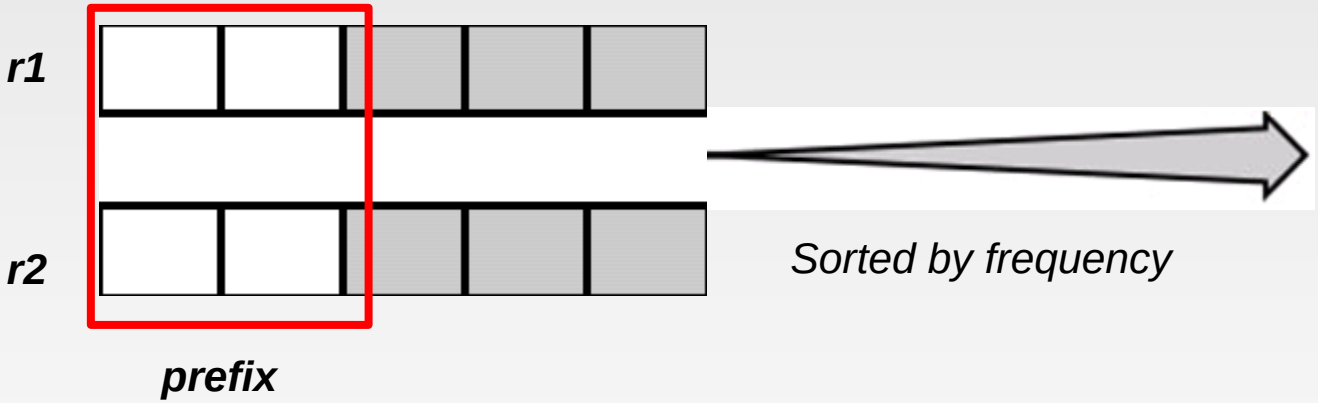
- Too much data to transfer ☹
- Too many pairs to verify ☹

Solving frequency skew: prefix filtering

- Sort tokens by frequency (ascending)

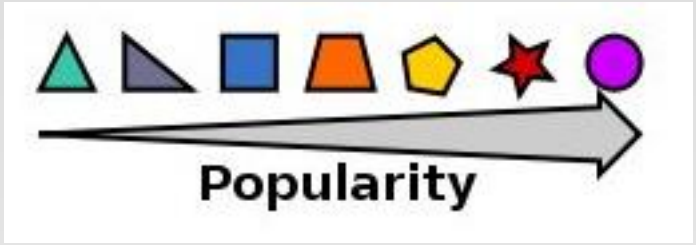


- Prefix of a set: least frequent tokens



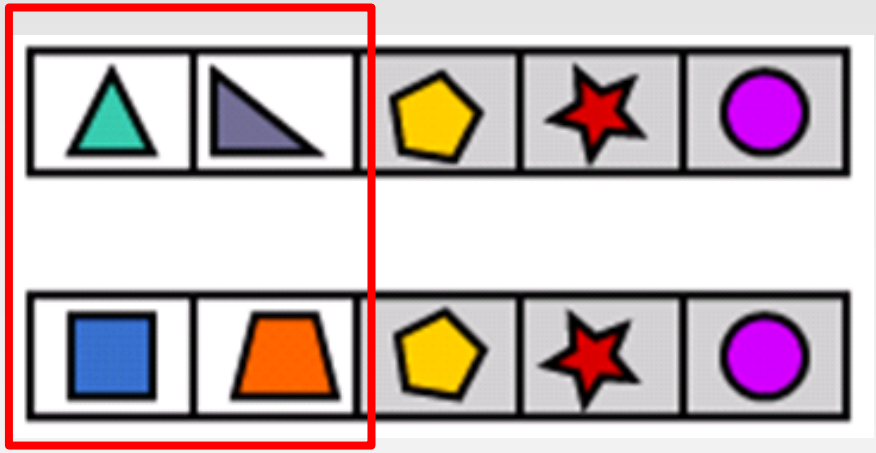
- Prefixes of similar sets should share tokens

Prefix filtering: example



Record 1

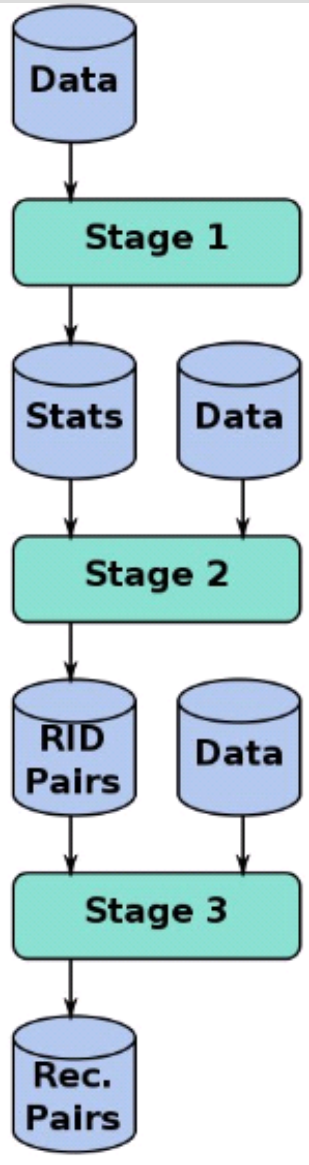
Record 2



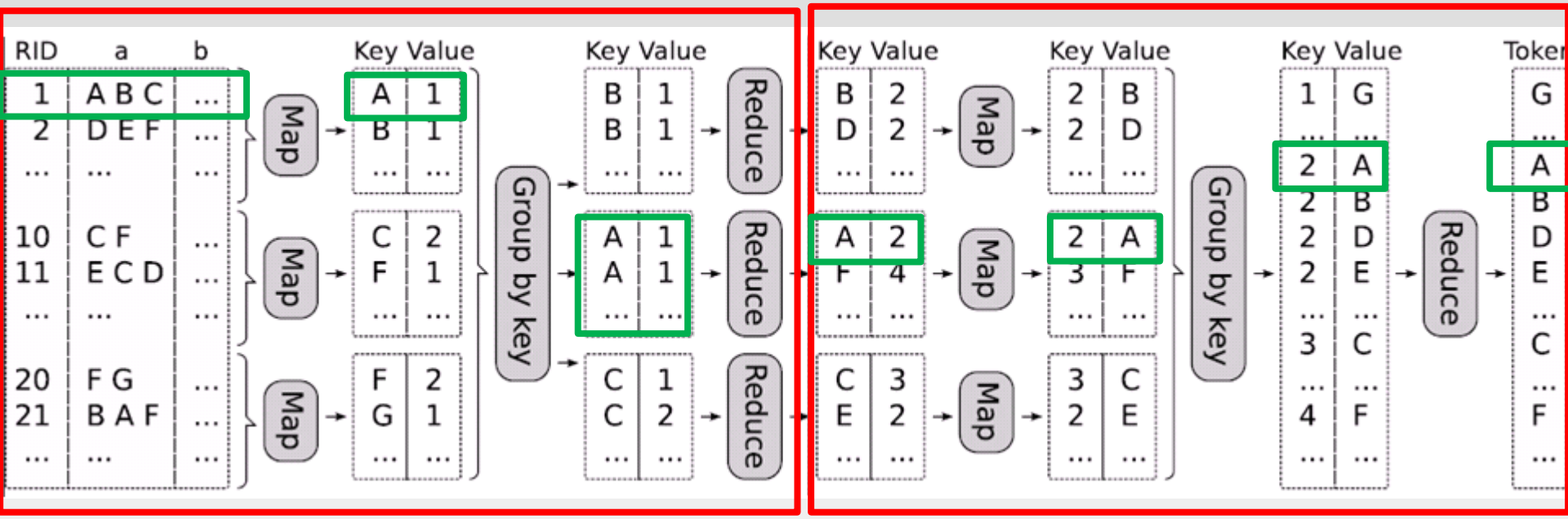
- Each set has 5 tokens
- “Similar”: they share at least 4 tokens
- Prefix length: 2

Hadoop Solution: Overview

- Stage 1: Order tokens by frequency
- Stage 2: Finding “similar” id pairs (verification)
- Stage 3: remove duplicates



Stage 1: Sort tokens by frequency



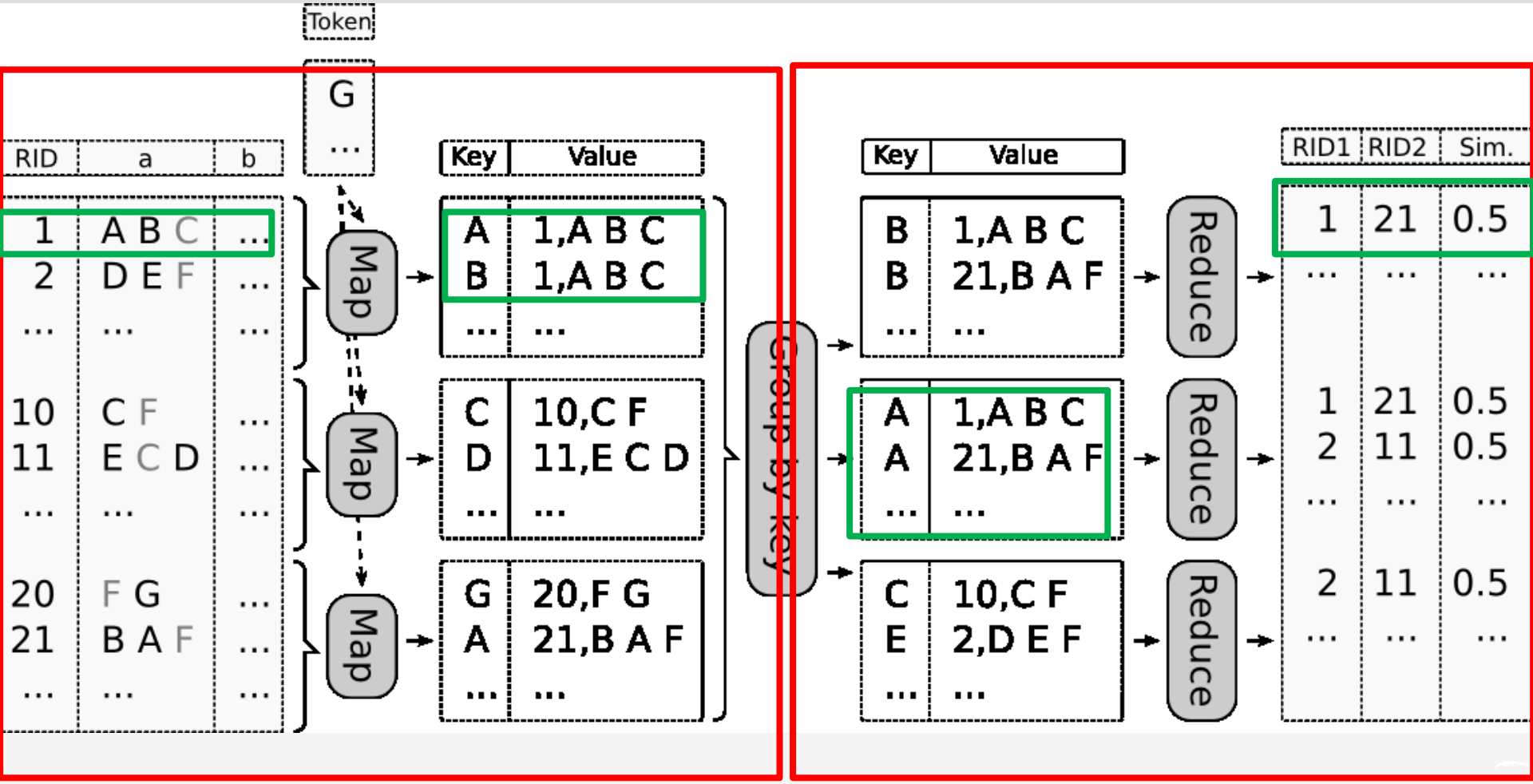
Compute token frequencies

Sort them

MapReduce phase 1

MapReduce phase 2

Stage 2: Find "similar" id pairs



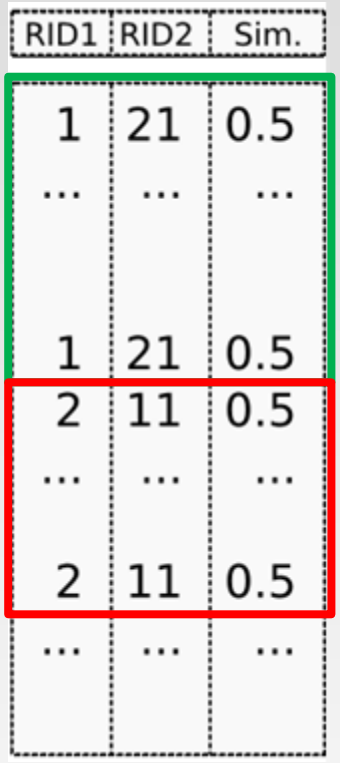
Partition using prefixes

Verify similarity

Compute the Length of Shared Tokens

- Jaccard Similarity: **sim(r, s) = |r∩s|/|r∪s|**
- If **sim(r, s) >= τ, l = |r∩s| >= |r∪s| * τ >= max(|r|, |s|) * τ**
- Given a record r, you can compute the prefix length as p = |r| - l + 1
- r and s is a candidate pair, they must share at least one token in the first (|r| - l + 1) tokens
- Given a record r = (A, B, C, D) and p = 2, the mapper emits (A, r) and (B, r)

Stage 3: Remove Duplicates



More Optimization Strategies

- Project 3: Do it using Spark on Google Dataproc
- It is your job to design more optimization strategies. The faster the better!
- Thinking:
  - How to compute the prefix length of a single record when processing it?
  - How to pass the sorted list to each worker?
  - Is it necessary to generate duplicate pairs?

References

- Chapter 3 of Mining of Massive Datasets.

End of Chapter 7.2