

CS4225/CS5425 Big Data Systems for Data Science

MapReduce & Data Mining

Bryan Hooi
School of Computing
National University of Singapore
bhooi@comp.nus.edu.sg



Announcements

- Tutorial on Hadoop will be held today (after lecture, 8.30pm – 9.30pm). The lecture will end about 20 minutes before, so you can take a break and come back
- The tutorial link is different from the lecture link: you can get the link from LumiNUS > Conferencing

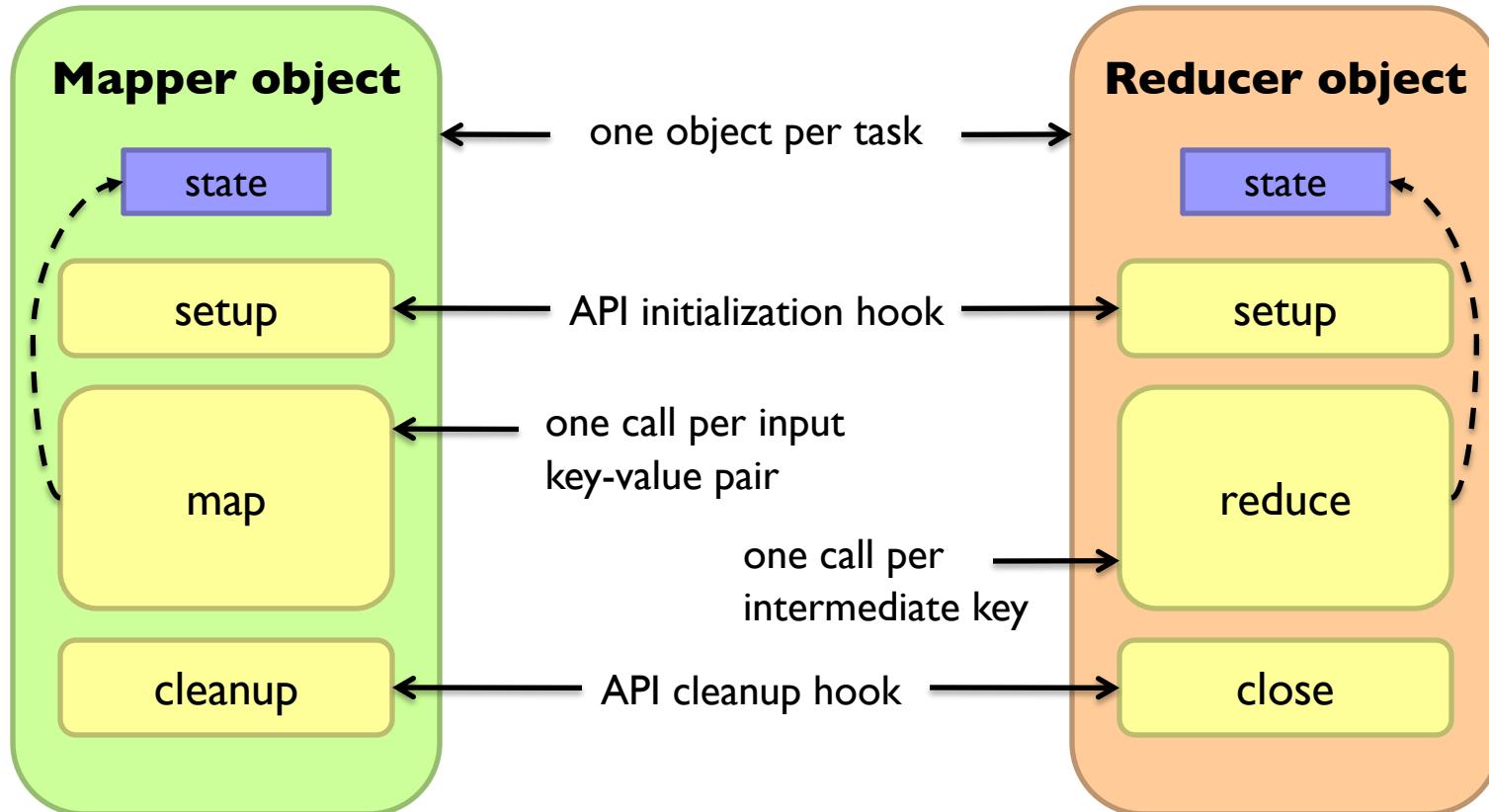
Week	Date	Topics	Tutorial	Due Dates
1	12 Aug	Overview and Introduction		
2	29 Aug	MapReduce - Introduction		
3	26 Aug	MapReduce and Relational Databases		
4	2 Sep	MapReduce and Data Mining	Tutorial: Hadoop	Assignment 1 released
5	9 Sep	NoSQL Overview 1		
6	16 Sep	NoSQL Overview 2		
Recess				
7	30 Sep	Apache Spark 1	Tutorial: NoSQL & Spark	Assignment 1 due, Assignment 2 released (3 Oct)
8	7 Oct	Apache Spark 2		
9	14 Oct	Large Graph Processing 1	Tutorial: Graph Processing	
10	21 Oct	Large Graph Processing 2		
11	28 Oct	Stream Processing	Tutorial: Stream Processing	Assignment 2 due (31 Oct)
12	4 Nov	Deepavali – No Class		
13	11 Nov	Test		

Announcements

- There is likely to be a delay of 1-2 weeks in releasing HWI, as the SoC cluster needed for running your assignment I code is still taking some time to be ready. Sorry for this!
- If the HWI release date is significantly delayed, we will try to reduce the length of the assignment (roughly proportionally), since you have less time to complete it, and the deadline will be pushed back as well
- Reminder: sign up for your SoC account if you don't have one

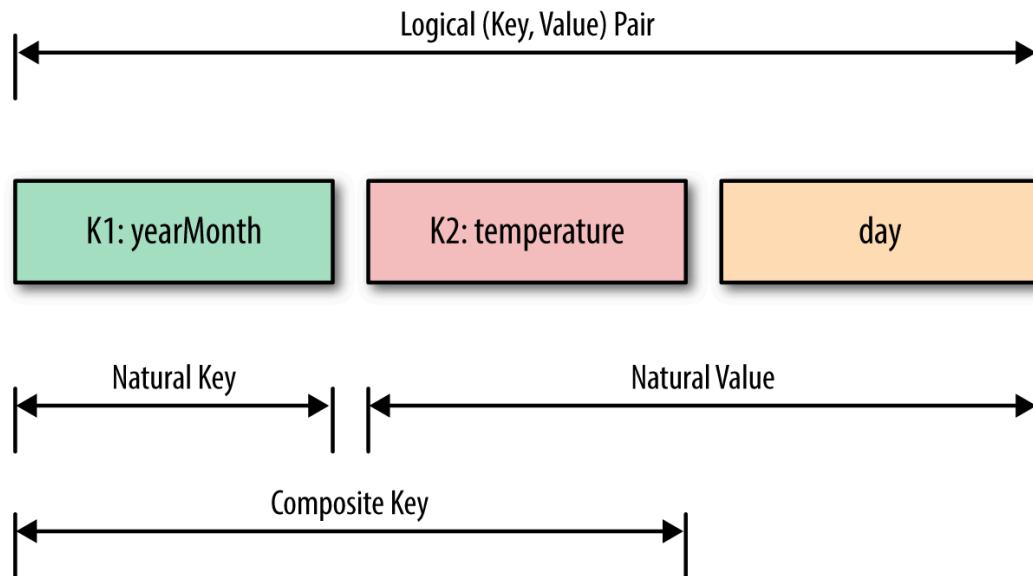
Week	Date	Topics	Tutorial	Due Dates
1	12 Aug	Overview and Introduction		
2	29 Aug	MapReduce - Introduction		
3	26 Aug	MapReduce and Relational Databases		
4	2 Sep	MapReduce and Data Mining	Tutorial: Hadoop	Assignment 1 released
5	9 Sep	NoSQL Overview 1		
6	16 Sep	NoSQL Overview 2		
Recess				
7	30 Sep	Apache Spark 1	Tutorial: NoSQL & Spark	Assignment 1 due, Assignment 2 released (3 Oct)
8	7 Oct	Apache Spark 2		
9	14 Oct	Large Graph Processing 1	Tutorial: Graph Processing	
10	21 Oct	Large Graph Processing 2		
11	28 Oct	Stream Processing	Tutorial: Stream Processing	Assignment 2 due (31 Oct)
12	4 Nov	Deepavali – No Class		
13	11 Nov	Test		

Recap: Preserving State in Mappers / Reducers



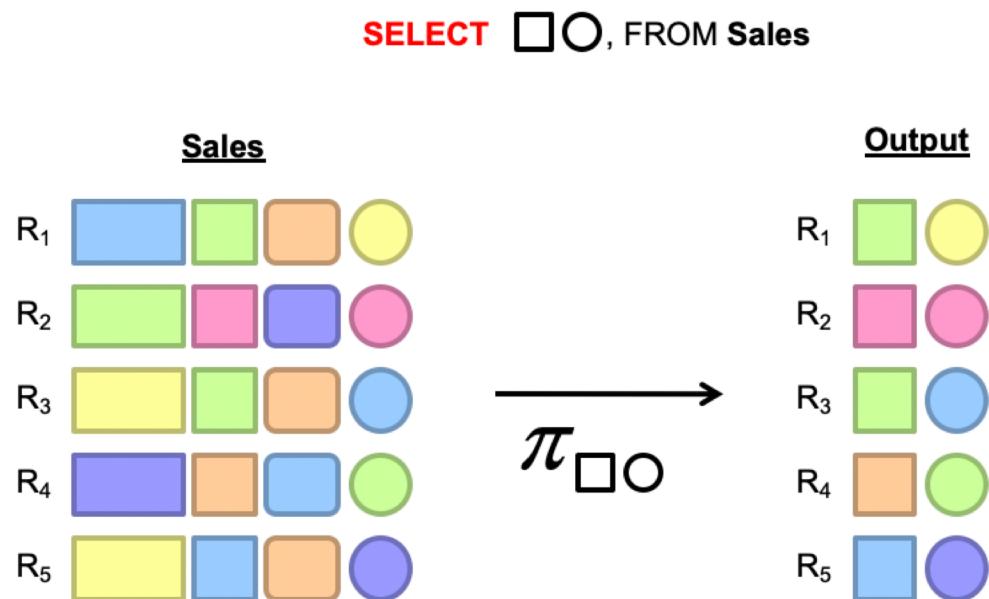
Recap: Secondary Sort

- **Problem:** each reducer's values arrive unsorted. But what if we want them to be sorted (e.g. sorted by temperature)?
- **Solution:** define a new ‘composite key’ as $(K1, K2)$, where $K1$ is the original key (“Natural Key”) and $K2$ is the variable we want to use to sort
 - **Partitioner:** now needs to be customized, to partition by $K1$ only, not $(K1, K2)$



Recap: Projection in MapReduce

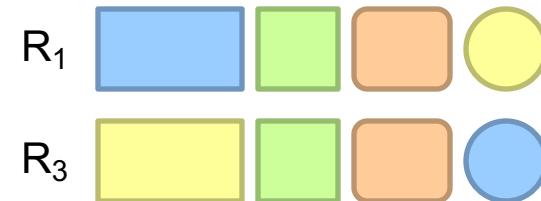
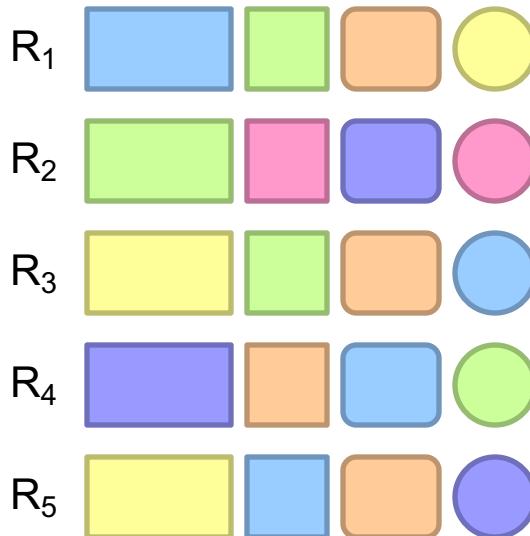
- **Map:** take in a tuple (with tuple ID as key), and emit new tuples with appropriate attributes
- No reducer needed (\Rightarrow no need shuffle step)



Recap: Selection in MapReduce

- **Map:** take in a tuple (with tuple ID as key), and emit only tuples that meet the predicate
- No reducer needed

predicate
↓
SELECT * FROM Sales WHERE (price > 10)



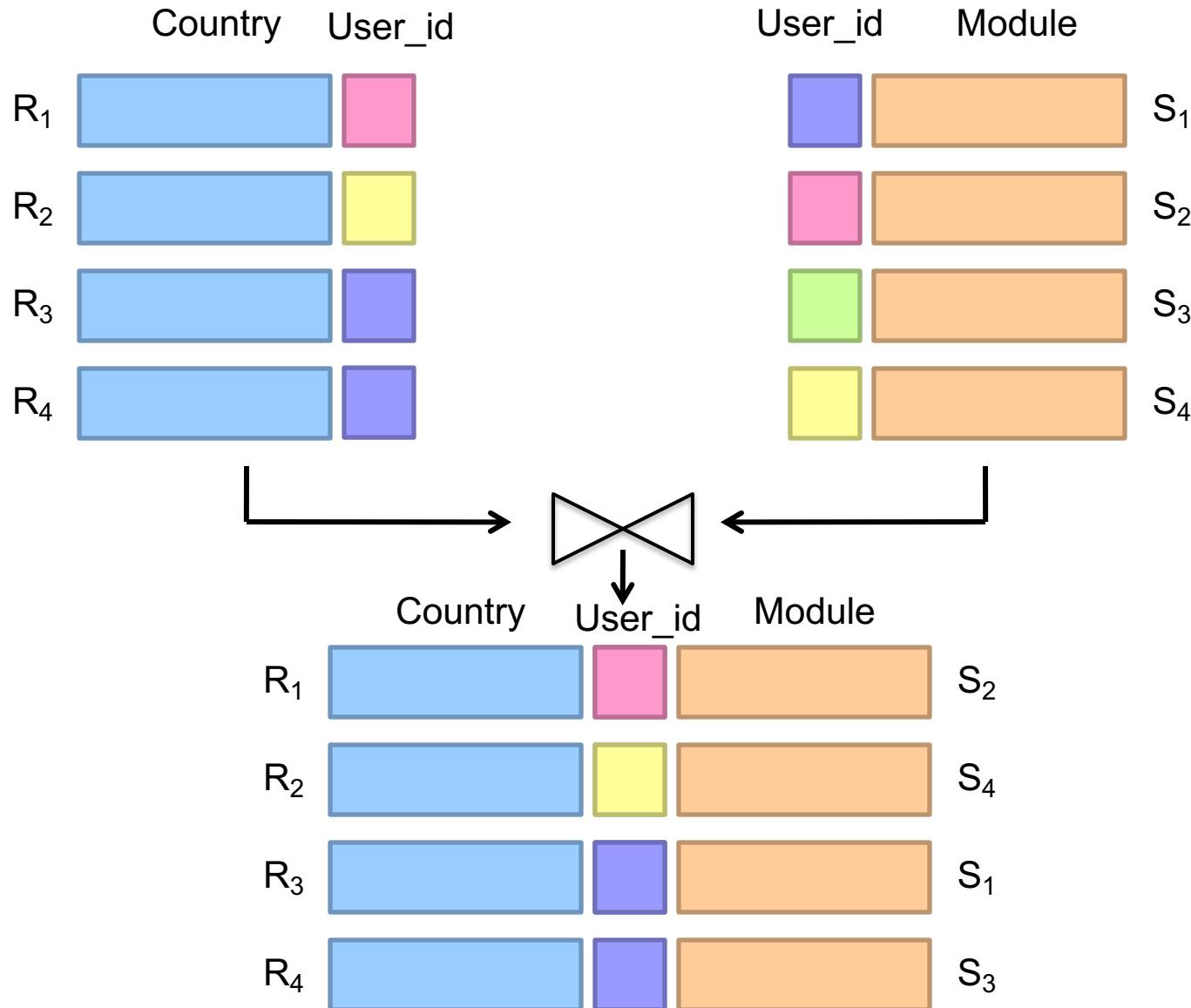
Recap: Group by... Aggregation

- Example: What is the average sale price per product?
- In SQL:
 - `SELECT product_id, AVG(price) FROM sales GROUP BY product_id`
- In MapReduce:
 - Map over tuples, emit `<product_id, price>`
 - Framework automatically groups values by keys
 - Compute average in reducer
 - Optimize with combiners

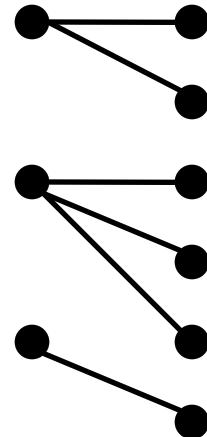
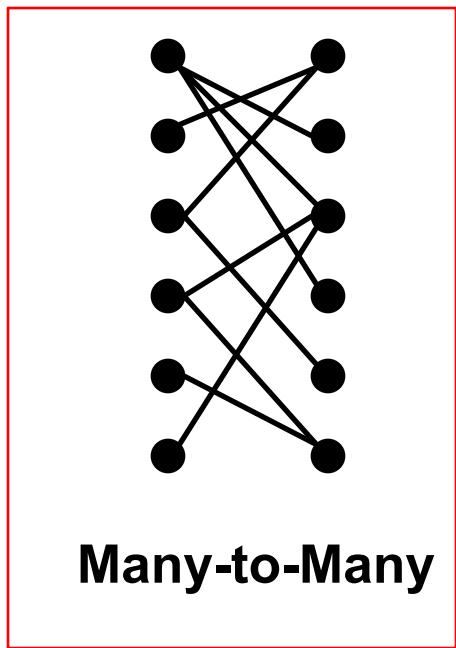
Overview

1. Relational Joins in MapReduce
2. Similarity Search
3. Clustering

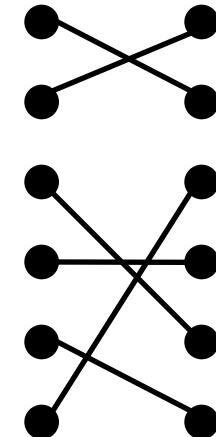
Relational Joins ('Inner Join')



Types of Relationships



One-to-Many

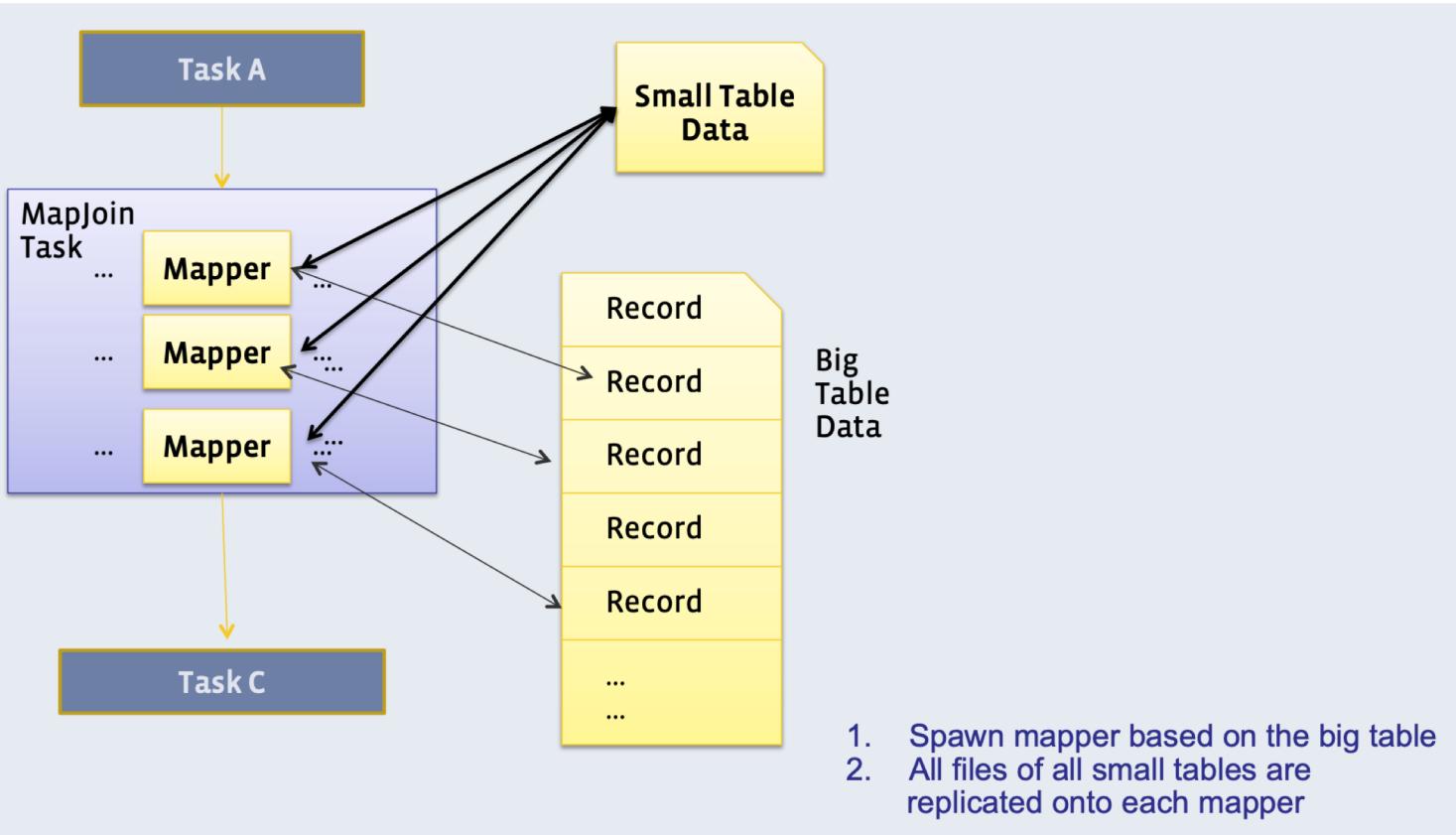


One-to-One

- “Many to many” relationship: e.g. a particular user ID can appear multiple times, in both tables

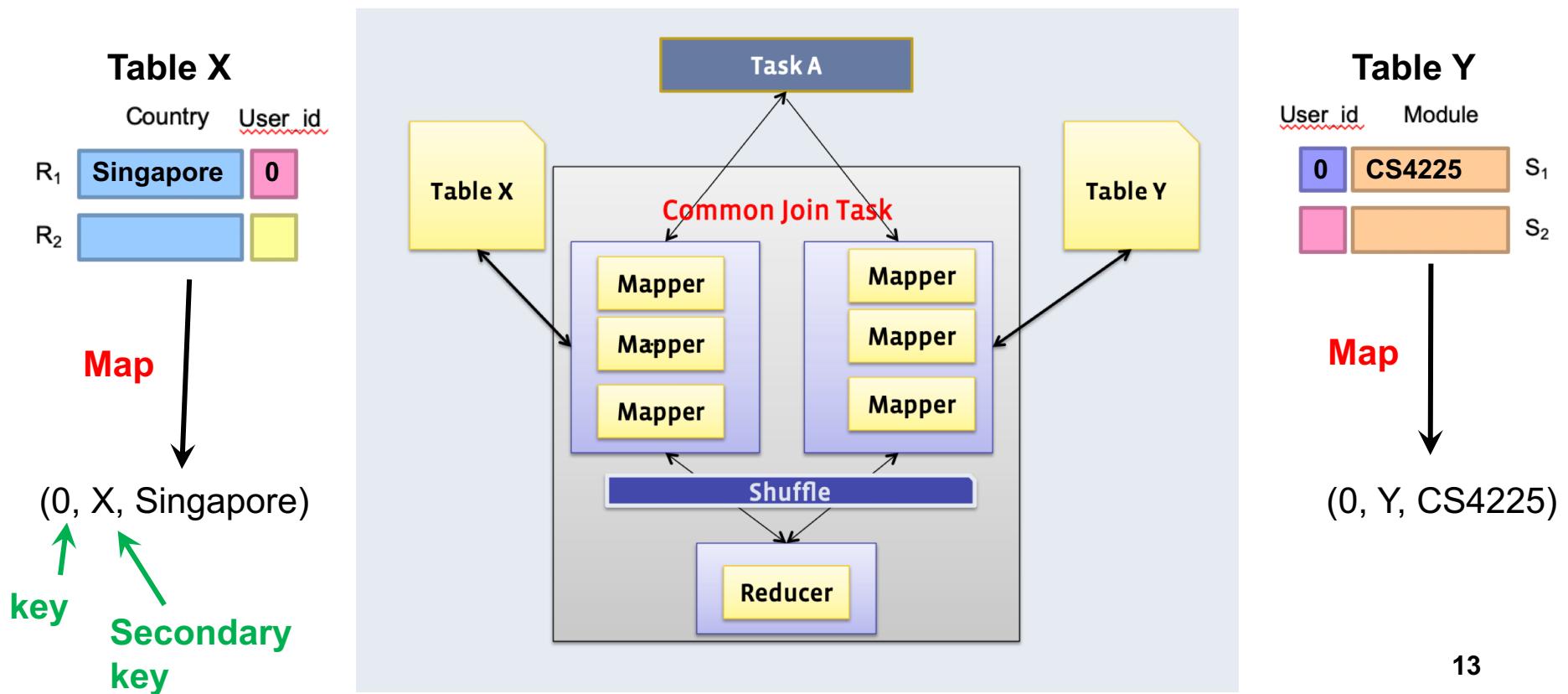
Method I: Map-Side Join

- Requires one of the tables to fit in memory
 - All mappers store a copy of the small table
 - They iterate over the big table, and join the records with the small table



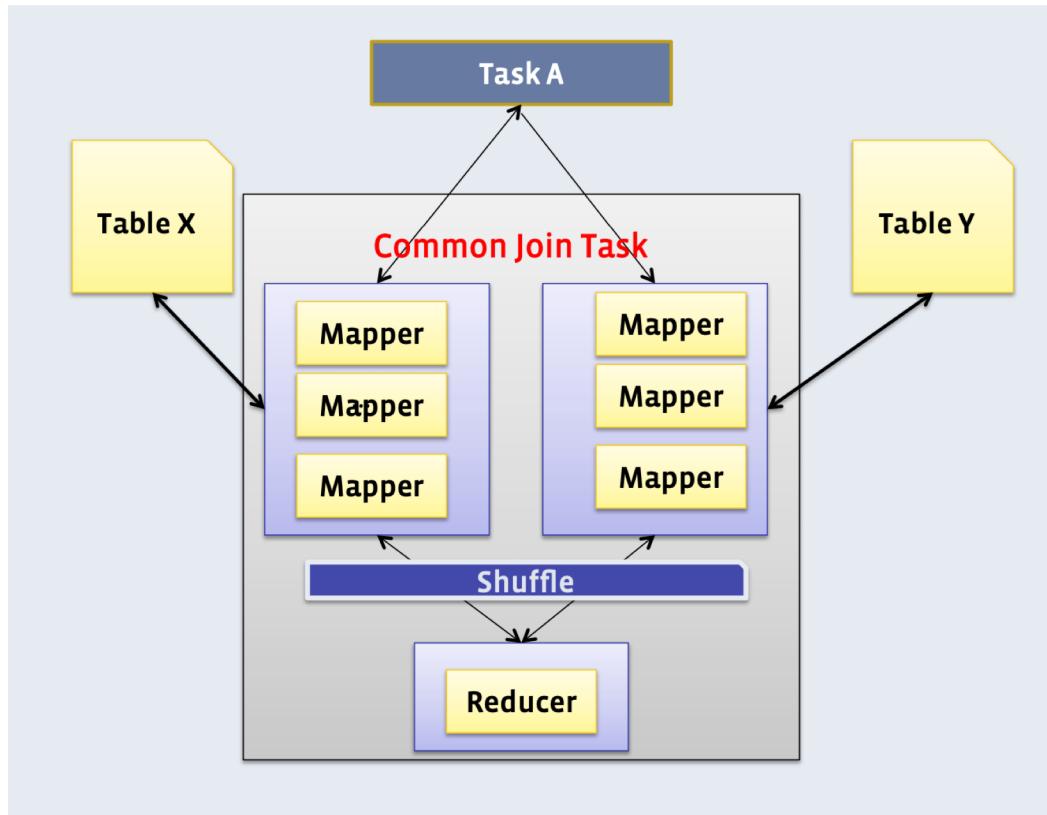
Method 2: Reduce-side ('Common') Join

- Doesn't require a dataset to fit in memory, but slower than map-side join
 - Different mappers operate on each table, and emit records, with key as the variable to join by



Method 2: Reduce-side ('Common') Join

- In reducer: we can use **secondary sort** to ensure that all keys from table X arrive before table Y
 - Then, hold the keys from table X in memory and cross them with records from table Y



In reducer (for key 0):

key	Secondary key	values
0	X	Singapore
0	X	
0	X	
0	Y	CS4225
0	Y	
0	Y	

Hold in memory

Cross with records from other set

Overview

1. Relational Joins in MapReduce
2. **Similarity Search**
3. Clustering

A Common Subroutine

- Many problems can be expressed as finding “similar” objects:
- **Examples:**
 - **Pages with similar words**
 - For duplicate detection, classification by topic
 - **Customers who purchased similar products**
 - Products with similar customer sets
 - **Users who visited similar websites**
 - Users with similar interests.

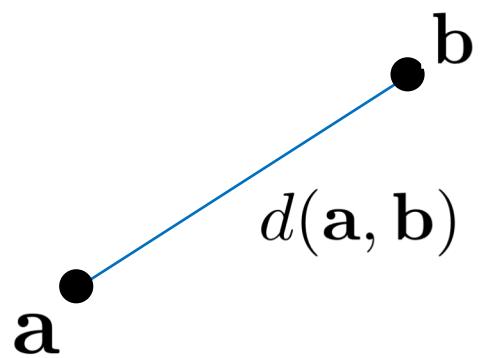
Distance / Similarity Measures

- We formally define “near neighbors” as points that are a “small distance” apart
- To measure the distance between objects x and y , we need a function $d(x, y)$ which we call a “distance measure”
- Similarity measures are the opposite: lower distance = higher similarity, and vice versa

Common distance / similarity metrics

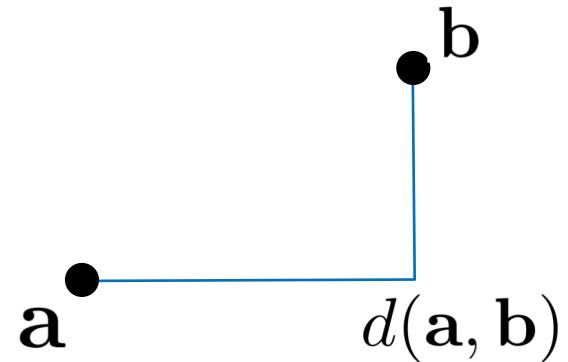
- Euclidean distance

$$d(\mathbf{a}, \mathbf{b}) = \|\mathbf{a} - \mathbf{b}\|_2 = \sqrt{\sum_{i=1}^D (a_i - b_i)^2}$$

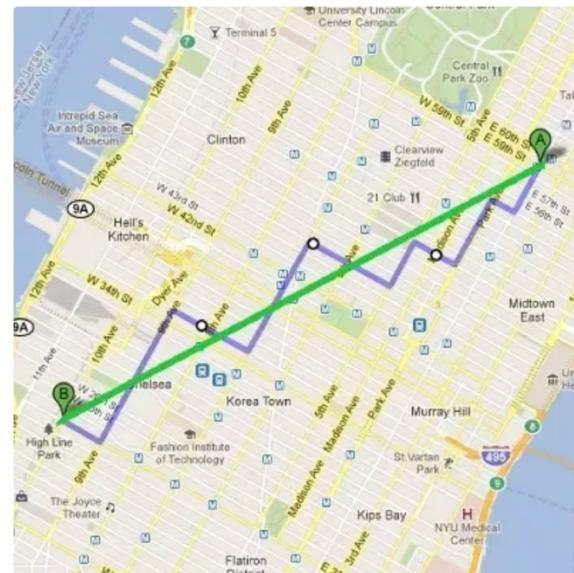


Common distance / similarity metrics

- **Manhattan distance**



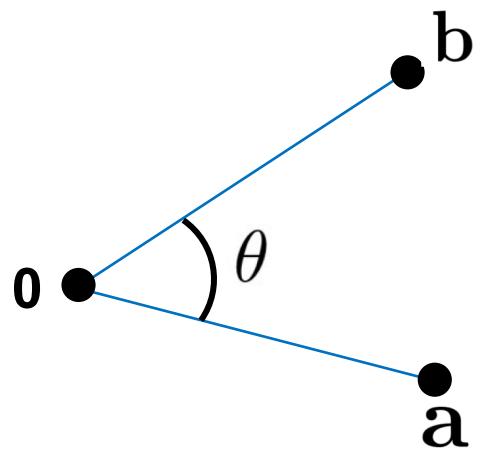
$$d(\mathbf{a}, \mathbf{b}) = \|\mathbf{a} - \mathbf{b}\|_1 = \sum_{i=1}^D |a_i - b_i|$$



Common distance / similarity metrics

- **Cosine similarity**

$$s(\mathbf{a}, \mathbf{b}) = \cos \theta = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \cdot \|\mathbf{b}\|}$$



Only considers direction: cosine similarity doesn't change if we scale a or b (i.e. multiplying them by a constant)

Common distance / similarity metrics

- **Jaccard Similarity**
(between **sets** A and B)

$$s_{\text{Jaccard}}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

$$A = \{ \text{bread} , \text{milk} \} \quad B = \{ \text{cheese} , \text{milk} \}$$

$$s_{\text{Jaccard}} = \frac{\text{milk}}{\text{cheese}, \text{bread}, \text{milk}} = 1/3$$

- **Jaccard Distance**

$$d_{\text{Jaccard}}(A, B) = 1 - s_{\text{Jaccard}}(A, B)$$

Task: Finding Similar Documents

○ Goals:

- **All pairs similarity:** Given a large number N of documents, find all “near duplicate” pairs, e.g. with Jaccard distance below a threshold
- **Similarity search:** Or: given a query document D, find all documents which are “near duplicates” with D

○ 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”

In order to show you the most relevant results, we have omitted some entries very similar to the 101 already displayed.

If you like, you can repeat the search with the omitted results included.

'Should have started this years back': Most hawker centre diners clear trays on first day of enforcement

CNA · 14 hours ago

- Hawker centre diners clear tables on first day of enforcement but some still flouting rule

The Straits Times · 20 hours ago



[View Full coverage](#)



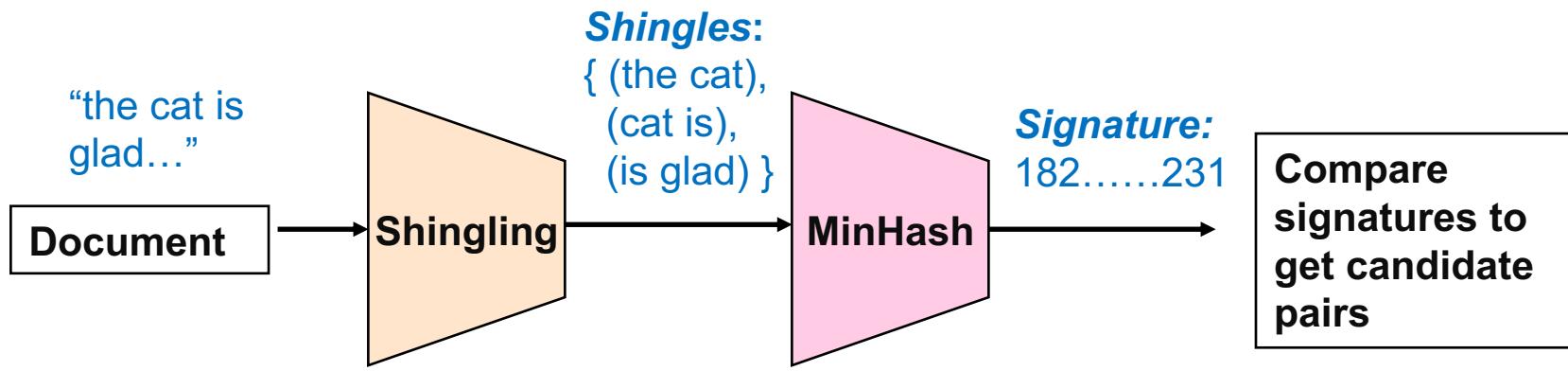
Task: Finding Similar Documents

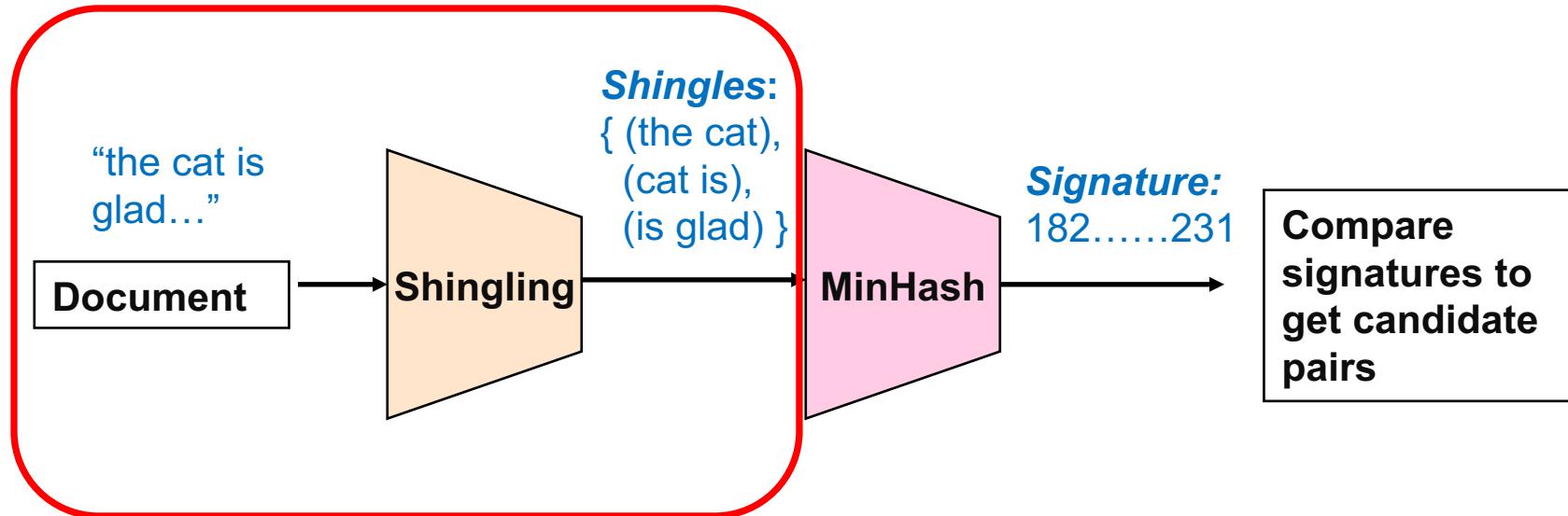
- **Goal:** Given a large number (N in the millions or billions) of documents, find “near duplicate” pairs
- **Applications:**
 - Mirror websites, or approximate mirrors
 - Don’t want to show both in search results
 - Similar news articles at many news sites
 - Cluster articles by “same story”
- **Problems:**
 - Too many documents to compare all pairs
 - Documents are so large or so many that they cannot be handled efficiently on a single machine.

2 Essential Steps for Similar Docs

1. ***Shingling:*** Convert documents to sets of short phrases (“shingles”)
2. ***Min-Hashing:*** Convert these sets to short signatures of each document, while preserving similarity
 - **Documents with the same signature are candidate pairs**

The Big Picture





Shingling

Step 1: *Shingling*: Convert documents to sets of phrases

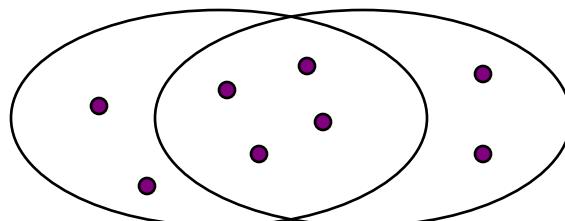
Define: Shingles

- A *k-shingle* (or *k-gram*) for a document is a sequence of k tokens that appears in the doc
- **Example: $k=2$** ; document $\mathbf{D}_1 = \text{"the cat is glad"}$
Set of 2-shingles: $\mathbf{S}(\mathbf{D}_1) = \{\text{"the cat"}, \text{"cat is"}, \text{"is glad"}\}$

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
- We measure similarity between documents as
Jaccard similarity:

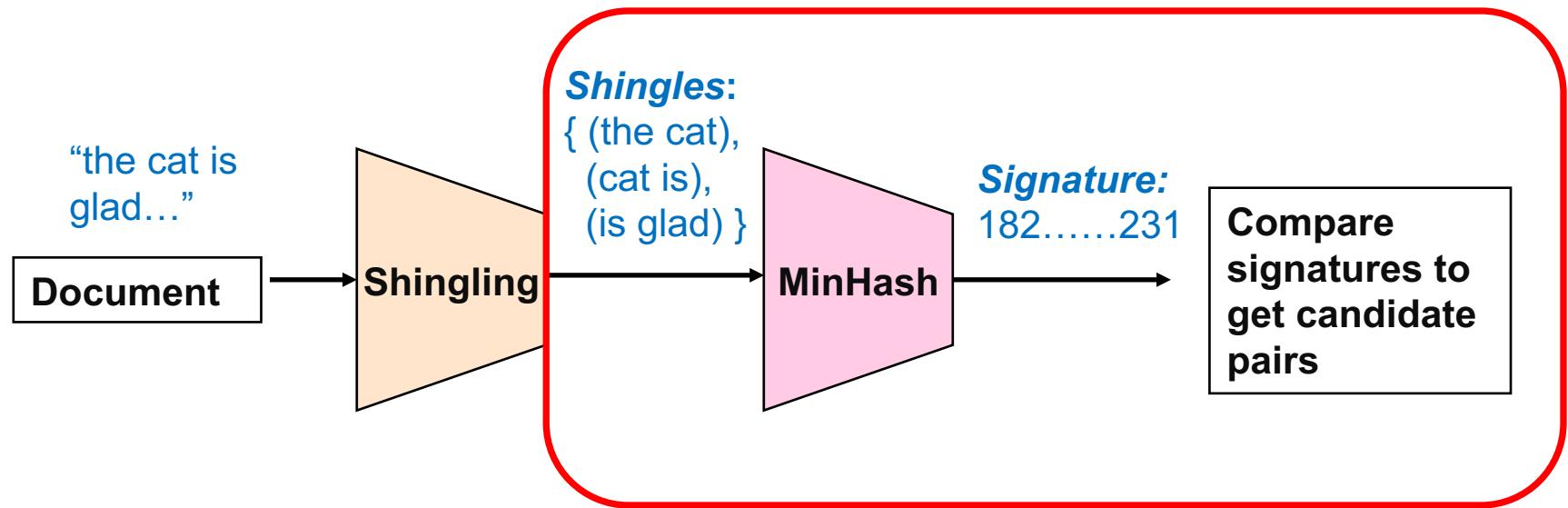
$$sim(D_1, D_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$$



Shingles	Documents			
1	1	1	1	0
1	1	0	1	1
0	1	0	1	1
0	0	0	1	1
1	0	0	1	1
1	1	1	0	1
1	0	1	0	1

Motivation for MinHash

- Suppose we have $N = 1$ million documents
- Naïvely, we would have to compute pairwise Jaccard similarities for every pair of docs
 - $N(N - 1)/2 \approx 5*10^{11}$ comparisons: too slow!
- MinHash gives us a fast approximation to the result of using Jaccard similarities to compare all pairs of documents



MinHashing

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

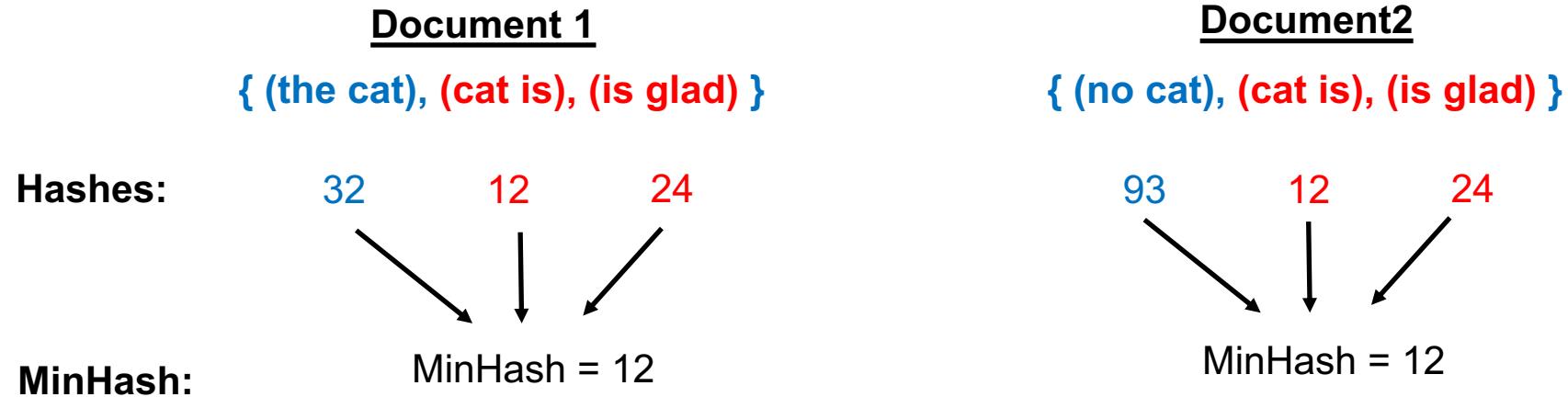
MinHash: Overview

- **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) highly similar documents usually have the same signature
- **Goal: Find a hash function $h(\cdot)$ such that:**
 - If $\text{sim}(C_1, C_2)$ is high, then with high probability, $h(C_1) = h(C_2)$
 - If $\text{sim}(C_1, C_2)$ is low, then with high probability, $h(C_1) \neq h(C_2)$

MinHash: Steps

- Given a set of shingles, { (the cat), (cat is), (is glad) }
- I. We have a **hash function** h that maps each shingle to an integer:
$$h(\text{"the cat"}) = 12, h(\text{"cat is"}) = 74, h(\text{"is glad"}) = 48$$
- 2. Then compute the **minimum** of these: $\min(12, 74, 48) = 12$

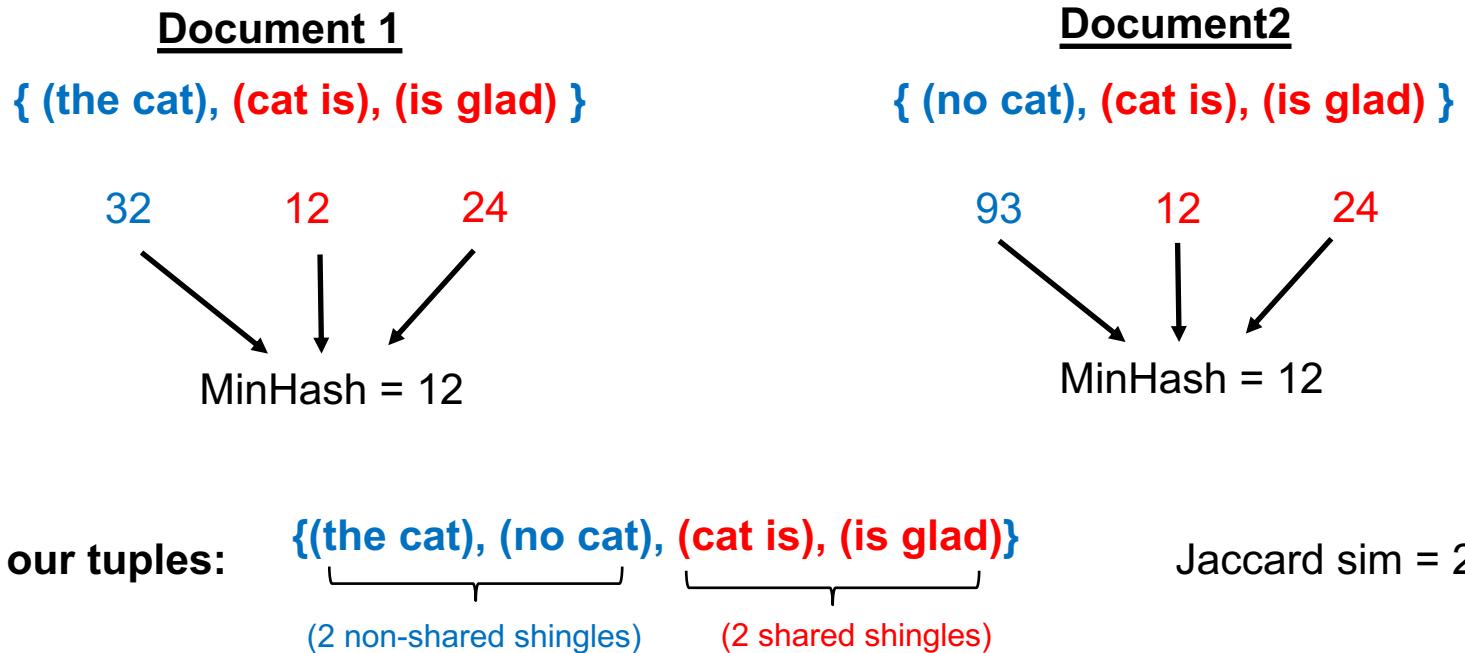
Example: MinHash on 2 sets of shingles



- Recall that we want to ensure that highly similar documents have high probability to have the same MinHash signature

Key Property of MinHash

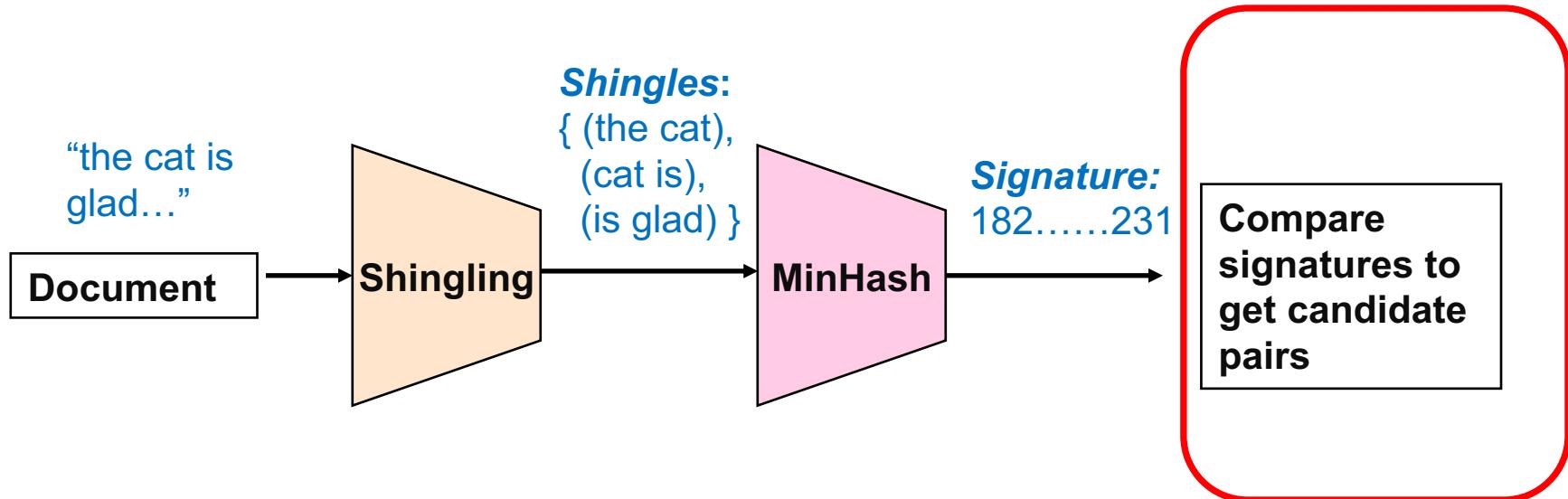
- **Key property:** the probability that two documents have the same MinHash signature is equal to their Jaccard similarity!
- Formally: $\Pr[h(C_1) = h(C_2)] = \text{Jaccard-Sim}(C_1, C_2)$
- **Proof** (not required knowledge):



Among these 4 tuples, each has the same probability of having the smallest hash value.
if one of the red shingles has the smallest hash, the documents will have the same MinHash.
Otherwise, they will have different MinHashes.

=> $\Pr[h(C_1) = h(C_2)] = (\text{red shingles} / \text{total shingles}) = (\text{intersection} / \text{union}) = \text{Jaccard similarity}$

Putting it all together



- **Multiple hashes:** we usually create multiple hash functions (e.g. $N=100$), and generate N signatures for each document, and concatenate them to get a final signature
- **Candidate pairs:** the documents with the same final signature are “candidate pairs”. We can either directly use them as our final output, or compare them one by one to check if they are actually similar pairs.

MapReduce Implementation

- Map

- Read over the document and extract its shingles
- Hash each shingle and take the min of them to get the MinHash signature
- Emit <signature, document_id>

- Reduce

- Receive all documents with a given MinHash signature
- Generate all candidate pairs from these documents
- (Optional): compare each such pair to check if they are actually similar

Any problem with this implementation?

Overview

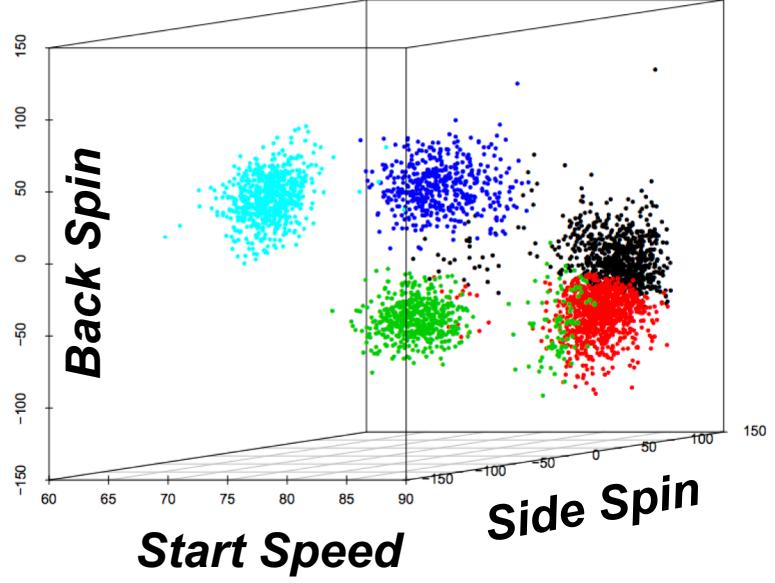
1. Relational Joins in MapReduce
2. Similarity Search
3. Clustering



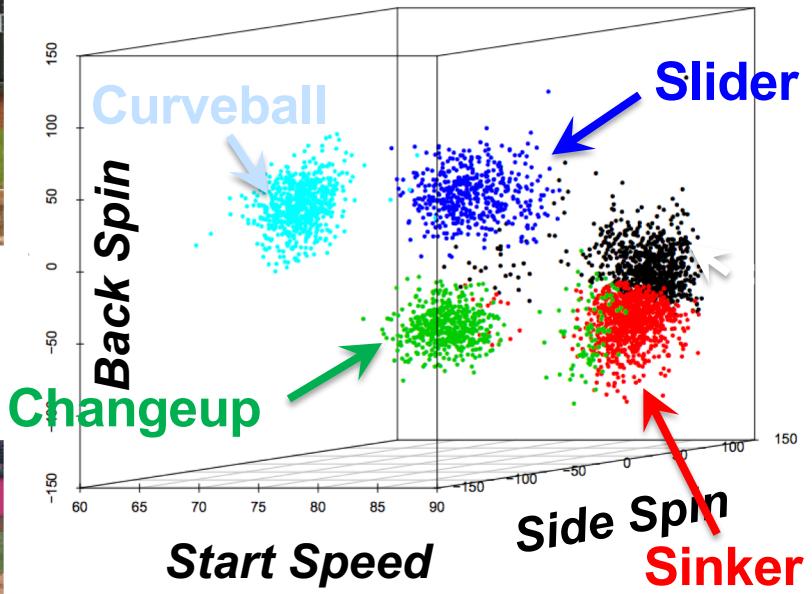




Pitches by Barry Zito



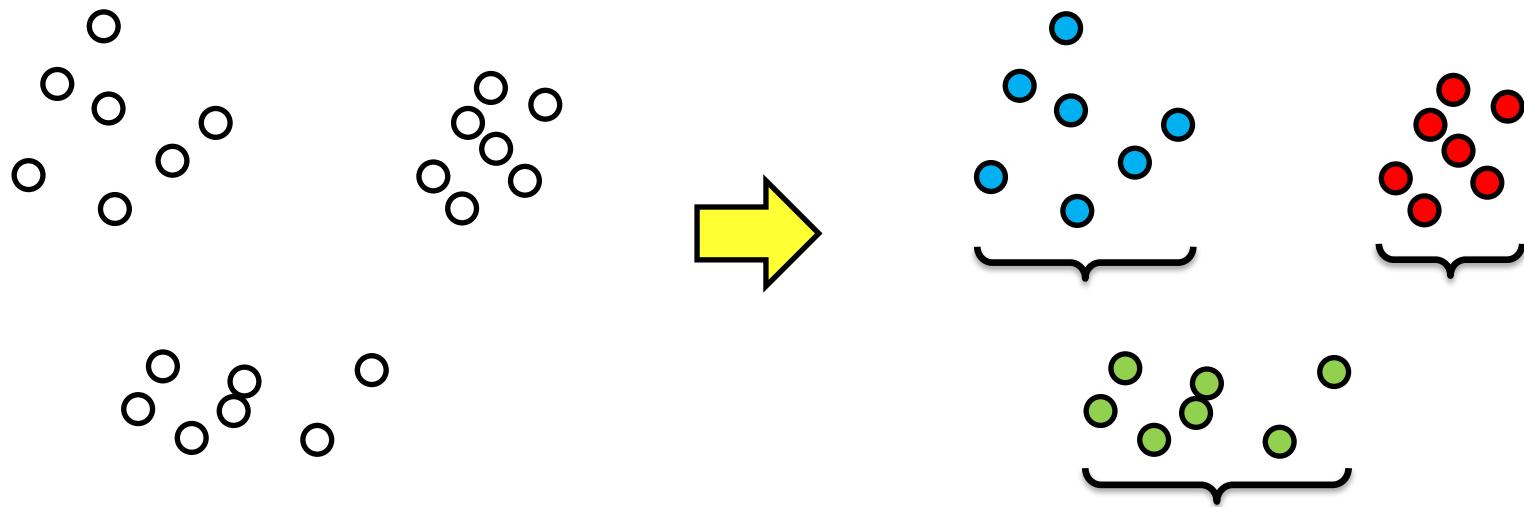
Pitches by Barry Zito



Goal of clustering

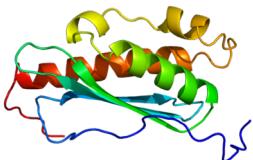
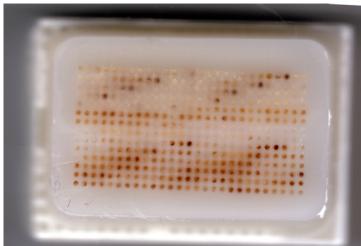
Clustering separates unlabelled data into groups of similar points.

Clusters should have high intra-cluster similarity, and low inter-cluster similarity.



Applications of clustering

Many applications:



Microbiology: find groups of related genes (or proteins etc.)



Recommendation & Social Networks: find groups of similar users

Google News

Trump, North Korea's Kim to hold second summit in late February

Channel NewsAsia • today

- Trump to hold second summit with Kim Jong Un in February

The Straits Times • today



[View more ▾](#)

Google

Introduction to K-means Clustering - DataScience.com

<https://www.datascience.com/blog/k-means-clustering> ▾

Dec 6, 2016 - Learn data science with data scientist Dr. Andrea Trevino's step-by-step tutorial on the K-means clustering unsupervised machine learning ...

K Means

stanford.edu/~cpiech/cs221/handouts/kmeans.html ▾

K-Means is one of the most popular "clustering" algorithms. K-means stores centroids that it uses to define clusters. A point is considered to be in a particular cluster if it is closer to that cluster's centroid than any other centroid.

Search & Information Retrieval: grouping similar search (or news etc.) results

K-Means Algorithm: Steps

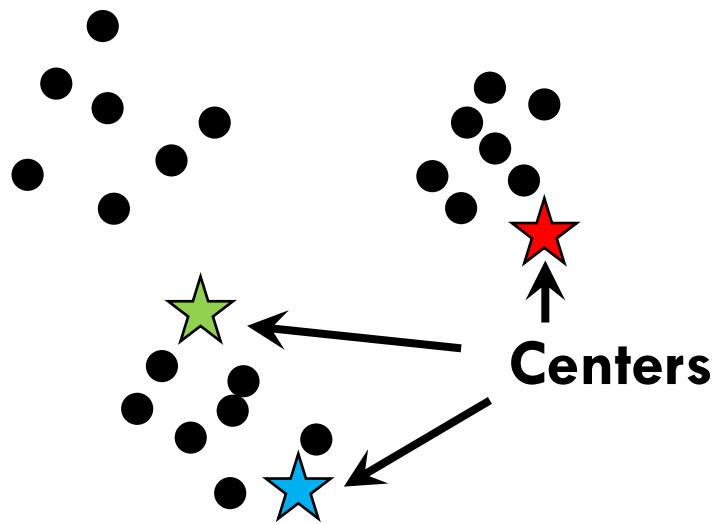
1. Initialization: Pick K random points as centers

2. Repeat:

a) **Assignment:** assign each point to nearest cluster

b) **Update:** move each cluster center to average of its assigned points

Stop if no assignments change



K-Means Algorithm: Steps

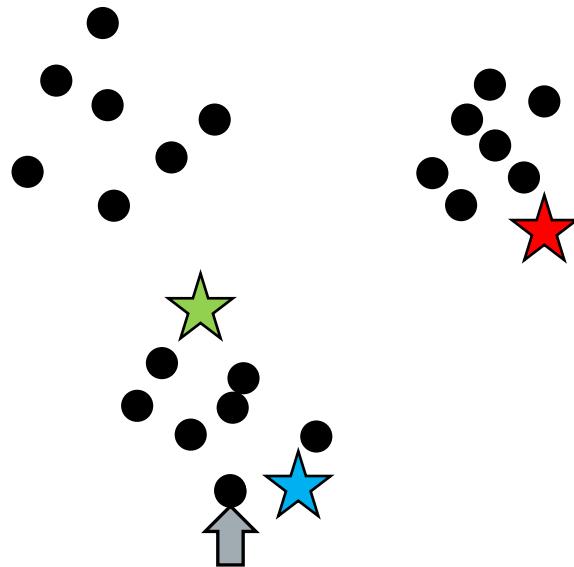
1. Initialization: Pick K random points as centers

2. Repeat:

a) **Assignment:** assign each point to nearest cluster

b) **Update:** move each cluster center to average of its assigned points

Stop if no assignments change



K-Means Algorithm: Steps

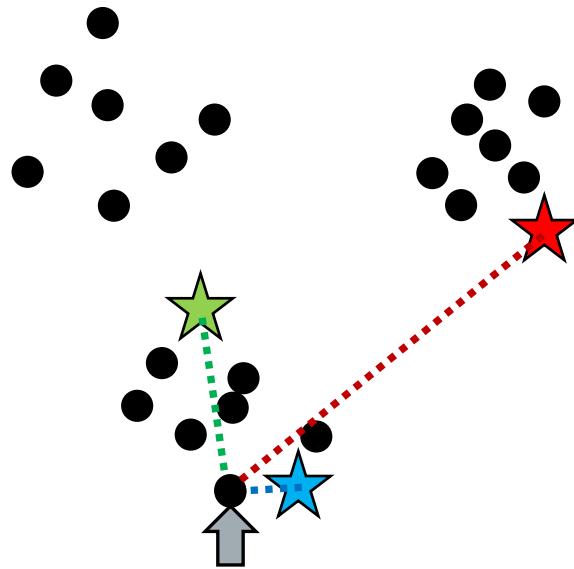
1. Initialization: Pick K random points as centers

2. Repeat:

a) **Assignment:** assign each point to nearest cluster

b) **Update:** move each cluster center to average of its assigned points

Stop if no assignments change



K-Means Algorithm: Steps

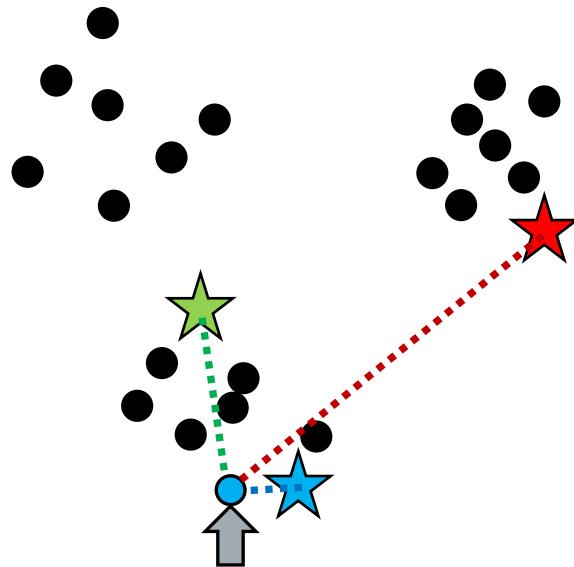
1. Initialization: Pick K random points as centers

2. Repeat:

a) **Assignment:** assign each point to nearest cluster

b) **Update:** move each cluster center to average of its assigned points

Stop if no assignments change



K-Means Algorithm: Steps

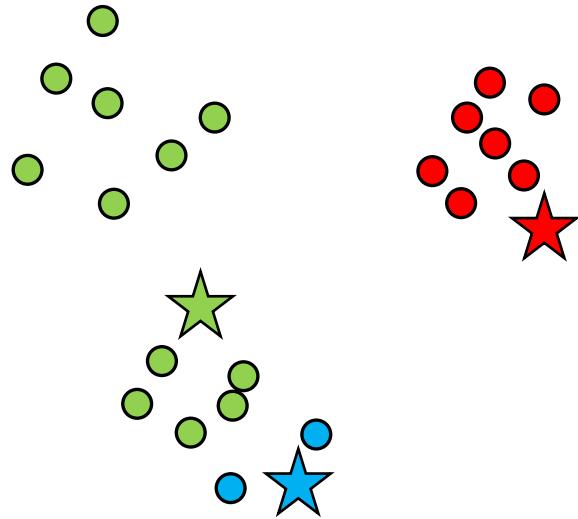
1. Initialization: Pick K random points as centers

2. Repeat:

a) **Assignment:** assign each point to nearest cluster

b) **Update:** move each cluster center to average of its assigned points

Stop if no assignments change



K-Means Algorithm: Steps

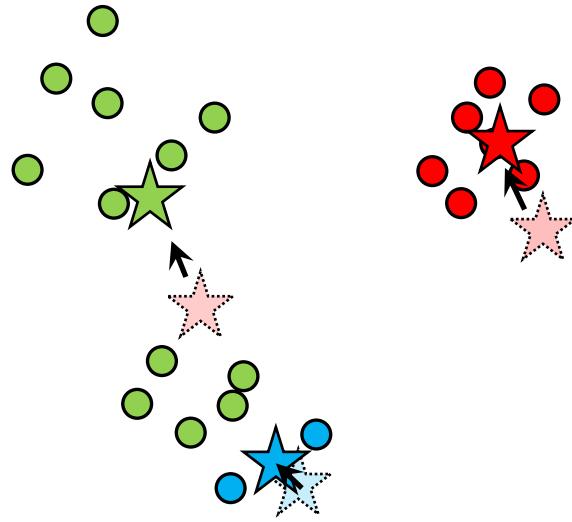
1. Initialization: Pick K random points as centers

2. Repeat:

a) **Assignment:** assign each point to nearest cluster

b) **Update:** move each cluster center to average of its assigned points

Stop if no assignments change



K-Means Algorithm: Steps

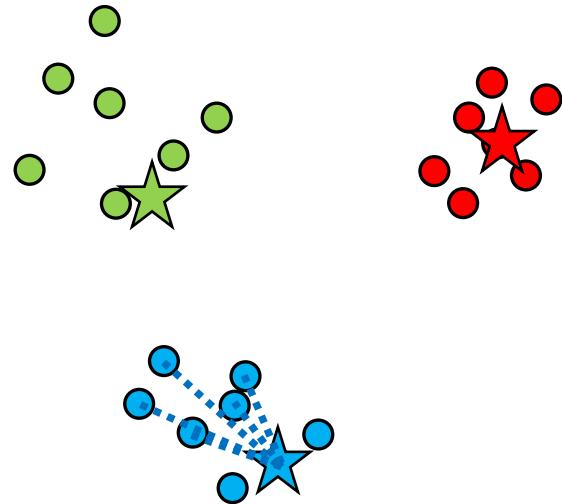
1. Initialization: Pick K random points as centers

2. Repeat:

a) **Assignment:** assign each point to nearest cluster

b) **Update:** move each cluster center to average of its assigned points

Stop if no assignments change



K-Means Algorithm: Steps

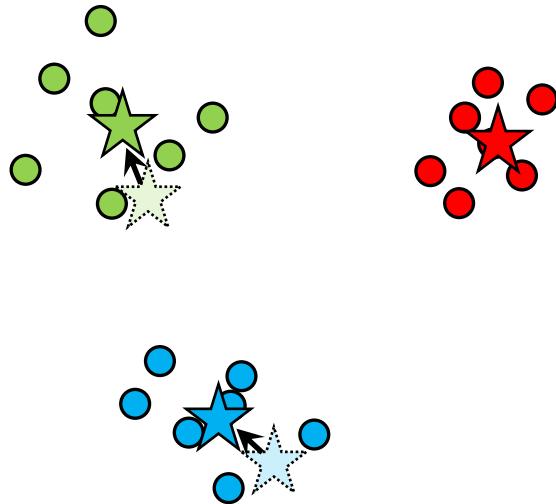
1. Initialization: Pick K random points as centers

2. Repeat:

a) **Assignment:** assign each point to nearest cluster

b) **Update:** move each cluster center to average of its assigned points

Stop if no assignments change



K-Means Algorithm: Steps

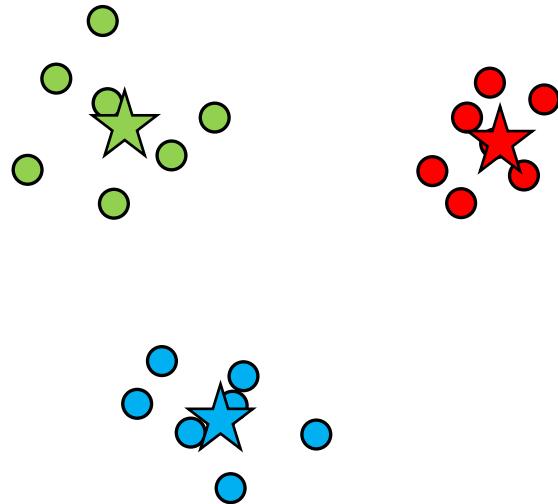
1. Initialization: Pick K random points as centers

2. Repeat:

a) **Assignment:** assign each point to nearest cluster

b) **Update:** move each cluster center to average of its assigned points

Stop if no assignments change



K-Means Algorithm: Steps

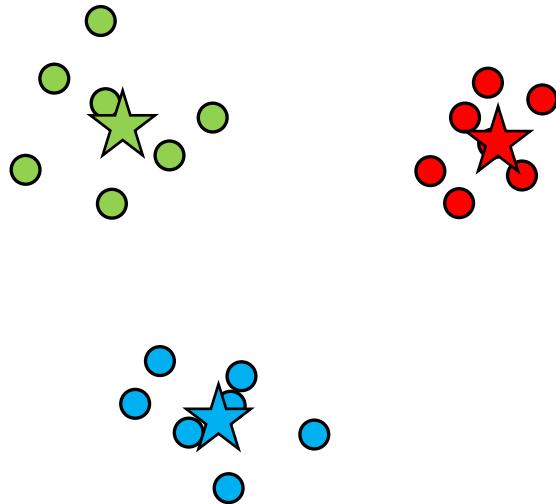
1. Initialization: Pick K random points as centers

2. Repeat:

a) Assignment: assign each point to nearest cluster

b) Update: move each cluster center to average of its assigned points

Stop if no assignments change



MapReduce Implementation v1

```
1: class MAPPER
2:   method CONFIGURE()
3:     c ← LOADCLUSTERS()
4:   method MAP(id i, point p)
5:     n ← NEARESTCLUSTERID(clusters c, point p) Network traffic!
6:     p ← EXTENDPOINT(point p)
7:     EMIT(clusterid n, point p)

1: class REDUCER
2:   method REDUCE(clusterid n, points [p1, p2, ...])
3:     s ← INITPOINTSUM()
4:     for all point p ∈ points do
5:       s ← s + p
6:     m ← COMPUTECENTROID(point s)
7:     EMIT(clusterid n, centroid m)
```

What is the problem?



Q: How much disk I/O is exchanged between the mappers and reducers, in MapReduce K-means v1?

(let n = no. of points, m = no. of iterations, d = dimensionality, k = no. of centers)

1. $O(n)$
2. $O(md)$
3. $O(nmd)$
4. $O(kmd)$



Q: How much disk I/O is exchanged between the mappers and reducers, in MapReduce K-means v1?

(let n = no. of points, m = no. of iterations, d = dimensionality, k = no. of centers)

1. $O(n)$
2. $O(md)$
3. $O(nmd)$
4. $O(kmd)$

MapReduce Implementation v2 (with in-mapper combiner)

```
1: class MAPPER
2:   method CONFIGURE()
3:      $c \leftarrow \text{LOADCLUSTERS}()$ 
4:      $H \leftarrow \text{INITASSOCIATIVEARRAY}()$ 
5:   method MAP(id  $i$ , point  $p$ )
6:      $n \leftarrow \text{NEARESTCLUSTERID}(\text{clusters } c, \text{ point } p)$ 
7:      $p \leftarrow \text{EXTENDPOINT}(\text{point } p)$ 
8:      $H\{n\} \leftarrow H\{n\} + p$ 
9:   method CLOSE()
10:    for all clusterid  $n \in H$  do
11:      EMIT(clusterid  $n$ , point  $H\{n\}$ )
12:
13: class REDUCER
14:   method REDUCE(clusterid  $n$ , points  $[p_1, p_2, \dots]$ )
15:      $s \leftarrow \text{INITPOINTSUM}()$ 
16:     for all point  $p \in \text{points}$  do
17:        $s \leftarrow s + p$ 
18:      $m \leftarrow \text{COMPUTECENTROID}(\text{point } s)$ 
19:     EMIT(clusterid  $n$ , centroid  $m$ )
```



Q: How much disk I/O is exchanged between the mappers and reducers, in MapReduce K-means v2?

(let n = no. of points, m = no. of iterations, d = dimensionality, k = no. of centers)

1. $O(n)$
2. $O(md)$
3. $O(nmd)$
4. $O(kmd)$



Q: How much disk I/O is exchanged between the mappers and reducers, in MapReduce K-means v2?

(let n = no. of points, m = no. of iterations, d = dimensionality, k = no. of centers)

1. $O(n)$
2. $O(md)$
3. $O(nmd)$
4. $O(kmd)$