Locality Sensitive Hashing by Spark

Alain Rodriguez, Fraud Platform, Uber
Kelvin Chu, Hadoop Platform, Uber

June 08, 2016

UBER

Overlapping Routes

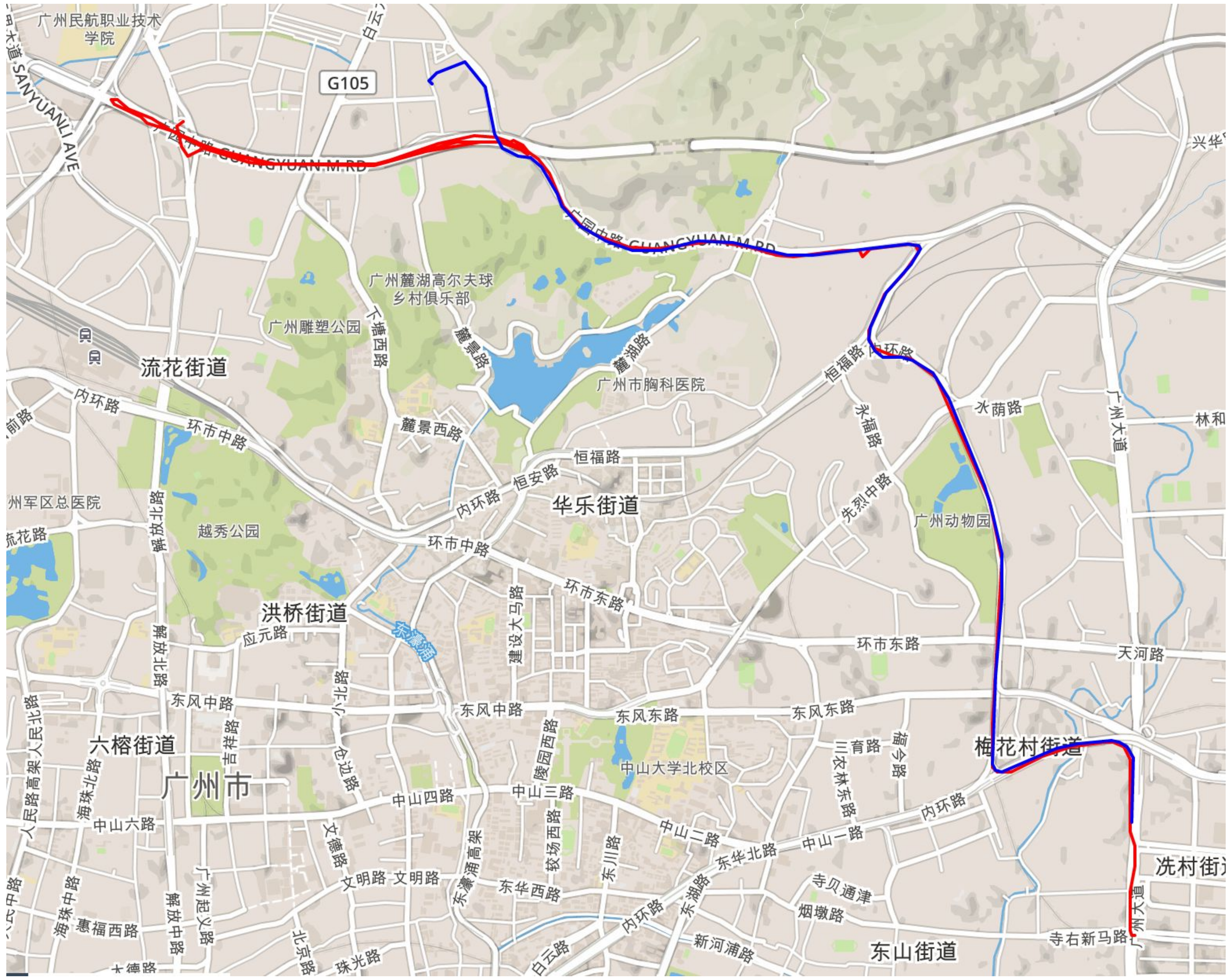
Finding similar trips in a city

The problem

Detect trips with a high degree of overlap

We are interested in detecting trips that have various degrees of overlap.

- Large number of trips
- Noisy, inconsistent GPS data
- **Not looking for exact matches**
- Directionality is important



Input Data

Millions of trips scattered over time and space

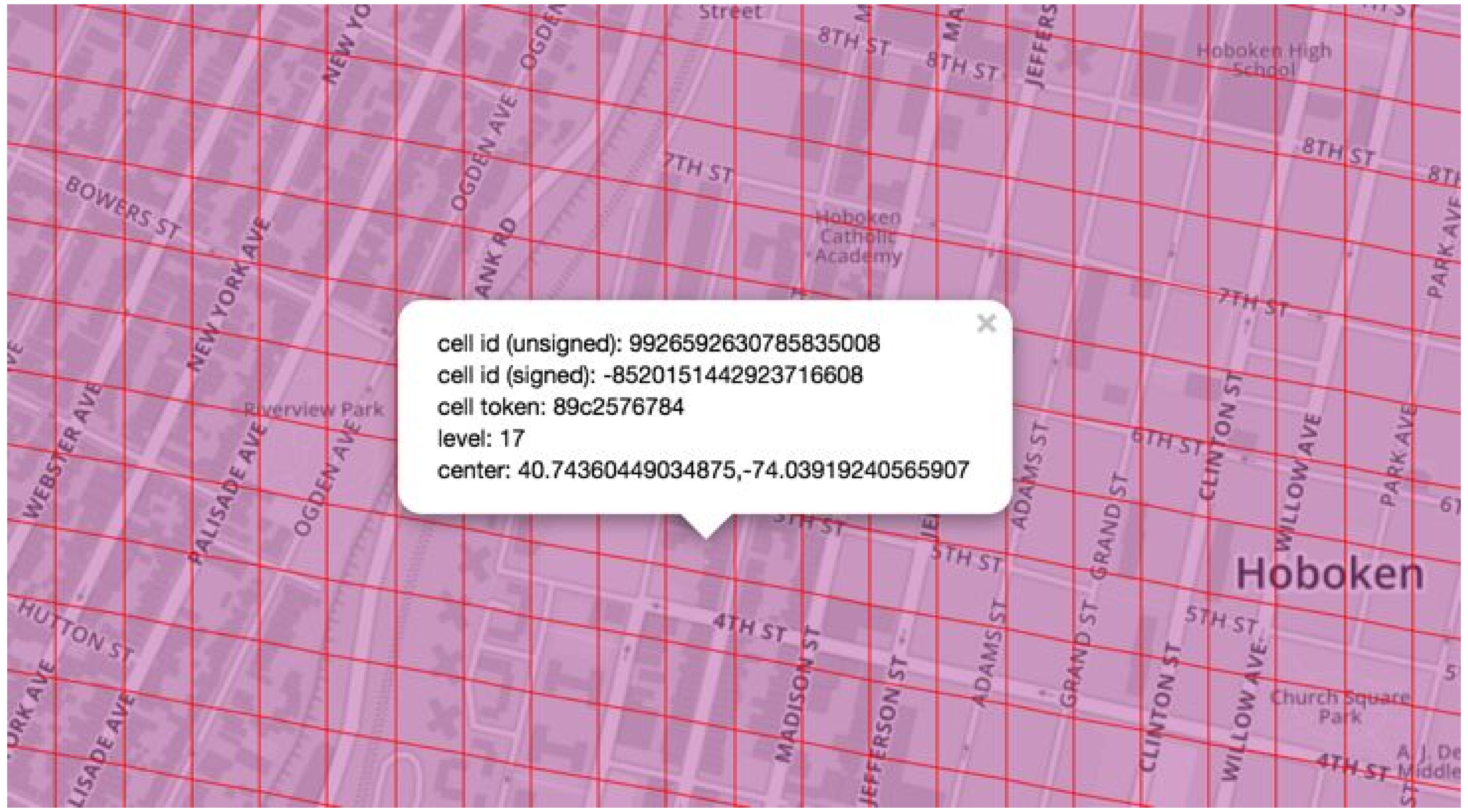
GPS traces are represented as an ordered list of **(latitude,longitude,time)** tuples.

- Coordinates are reals and have noise
- Traces can be dense or sparse, yet overlapping
- Large time and geographic search space


```
[
  {
    "latitude":25.7613453844,
    "epoch":1446577692,
    "longitude":-80.197244976
  },
  {
    "latitude":25.7613489535,
    "epoch":1446577693,
    "longitude":-80.1972450862
  },
  …
]
```

Google S2 Cells

Efficient geo hashing



Divides the world into consistently sized regions.

Area segments can be had of different sizes

Jaccard index

Set similarity coefficient

The Jaccard index can be used as a measure of set similarity

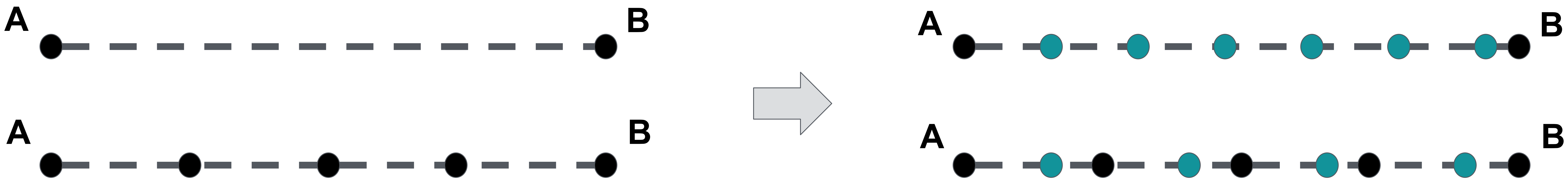
$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

A = {a, b, c}, B = {b, c, d}, C = {c, d, e}

J(A, A) = 1.0
J(A, B) = 0.5
J(A, C) = 0.2

Heuristic

Densify sparse traces



Sparse and dense traces should be matched

Different devices generate varying data densities. Two segments that start and end at the same location should be detected as overlapping.

Ensure points are at most X distance apart

Densification ensures that continuous segments are independently overlapping.

Heuristic

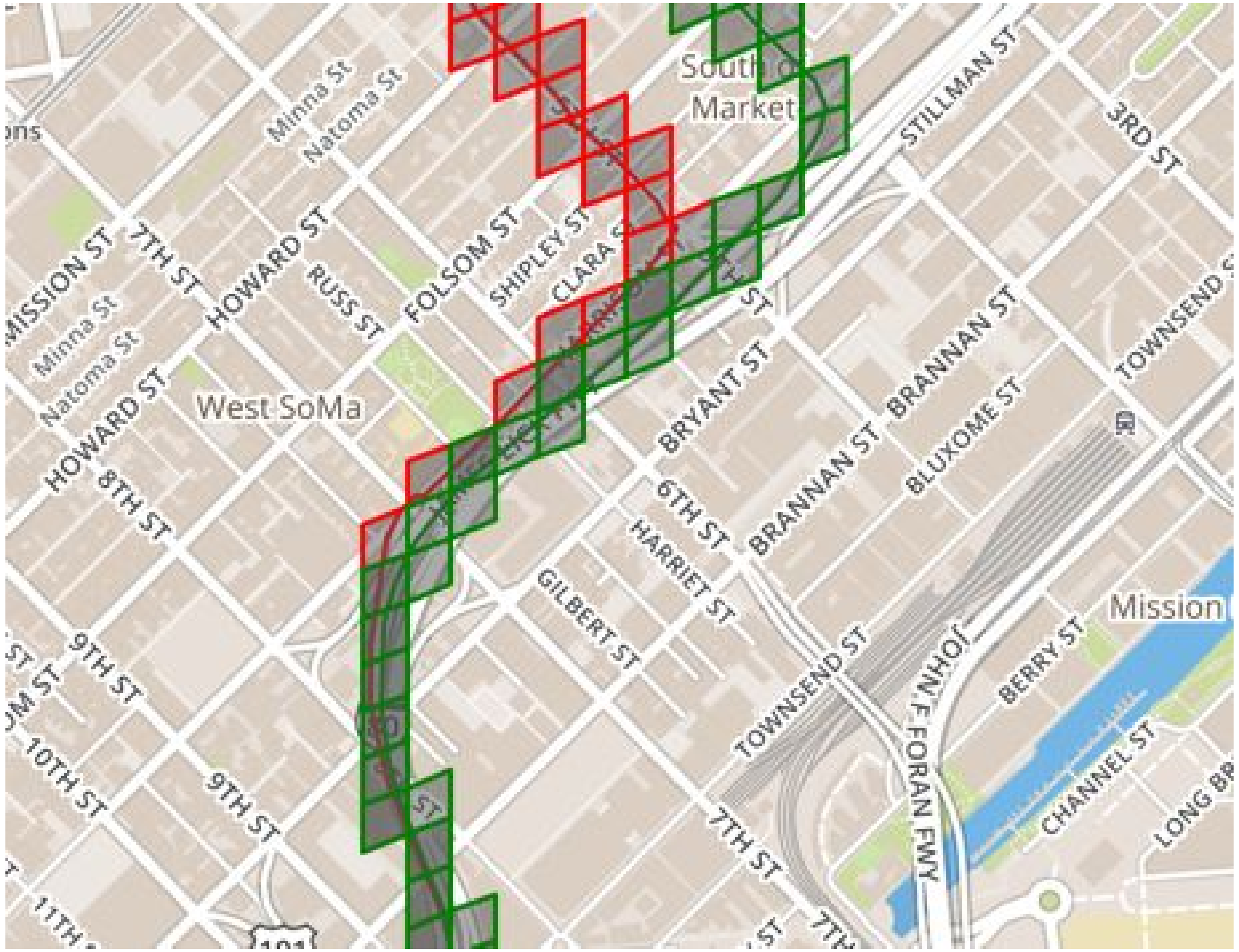
Discretize route segments

Discretize segments

Break down routes into equal size **area** segments; this eliminates route noise. Segment size determines matching sensitivity.

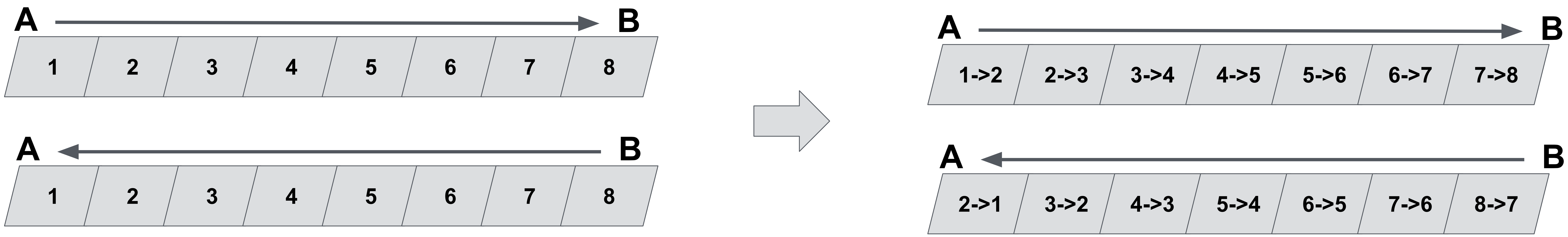
Remove contiguous duplicates

Remove noise resulting from a vehicle stopped at a light or a very chatty device.



Heuristic

Shingle contiguous area segments



Directionality matters

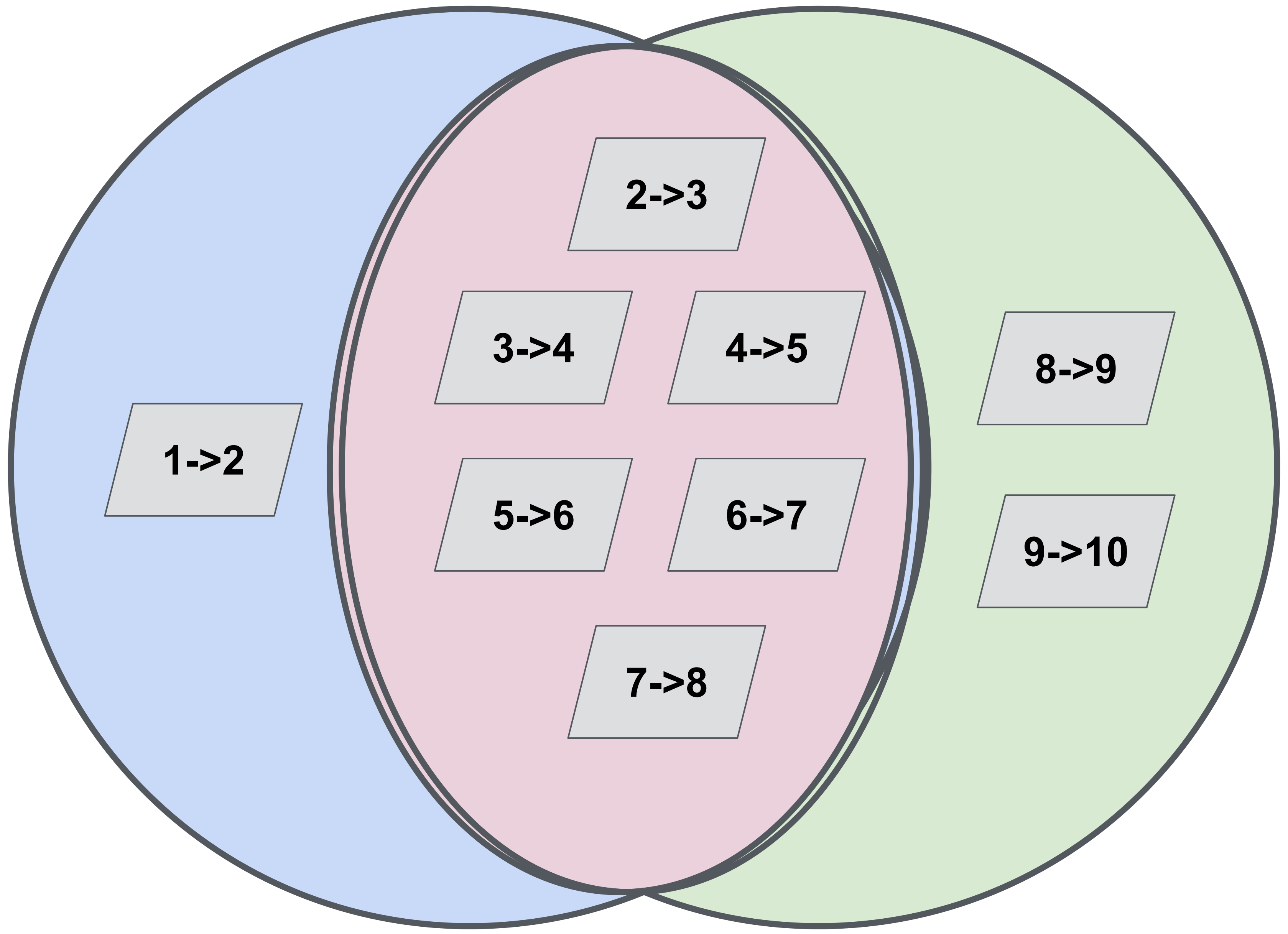
Two overlapping trips with opposite directions should not be matched.

Shingling captures directionality

Combining contiguous segments captures the sequence of moves from one segment to another.

Set overlap problem

Find traces that have the desired level of common shingles



N^2 takes forever

LSH to the rescue

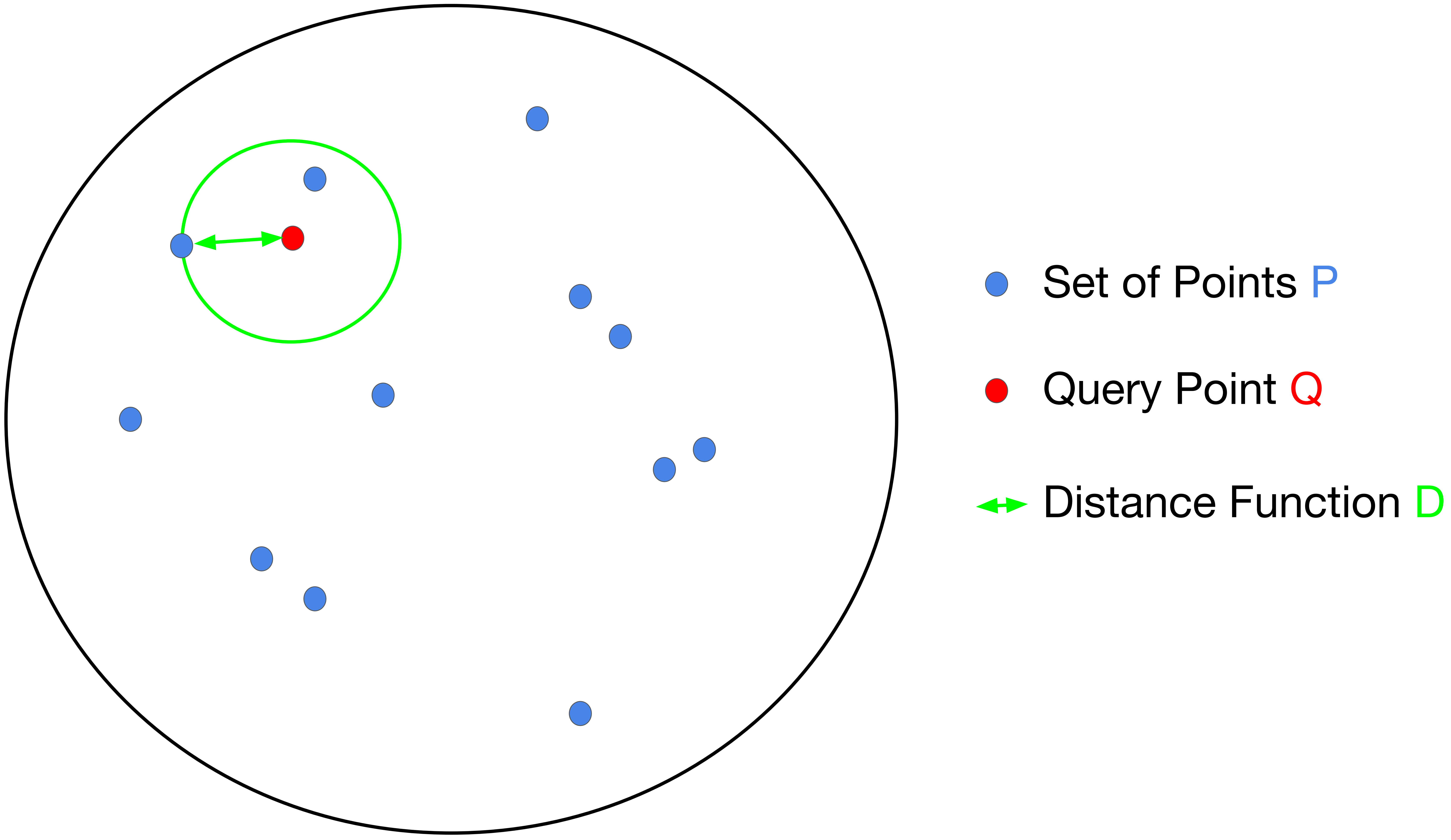
- Sifting through a month's worth of trips for a city takes forever with the N^2 approach

- Locality-Sensitive Hashing allows us to find most matches quickly. Spark provides the perfect engine.

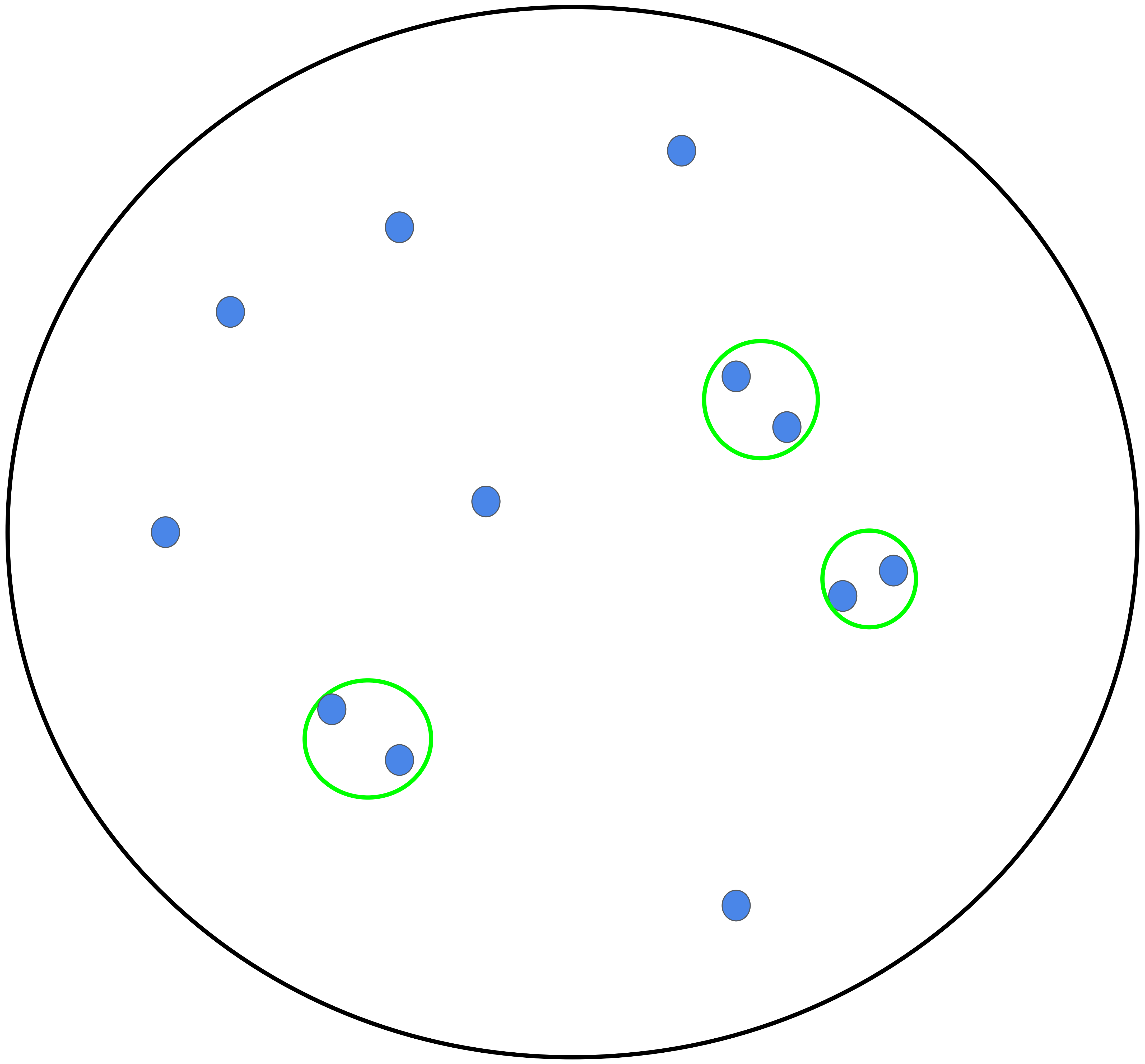
Locality-Sensitive Hashing (LSH)

Quick Introduction

Problem - Near Neighbors Search



Problem - Clustering



Set of Points P

Distance Function D

Curse of Dimensionality

1-Dimension e.g. single integer

Q: 7 Distance: 3

A Solution: Binary Tree e.g. Return 9, 4, 8, ...

2-Dimension e.g. GPS point

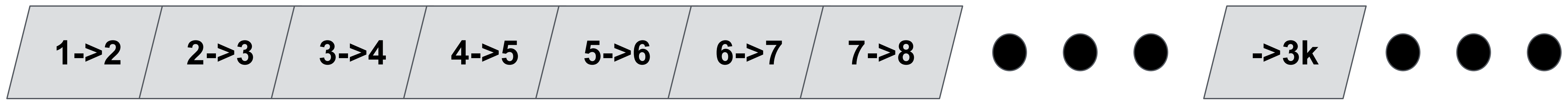
Q: (12.73, 61.45) Distance: 10

A Solution: Quadtree, R-tree, etc

Curse of Dimensionality

How about very high dimension?

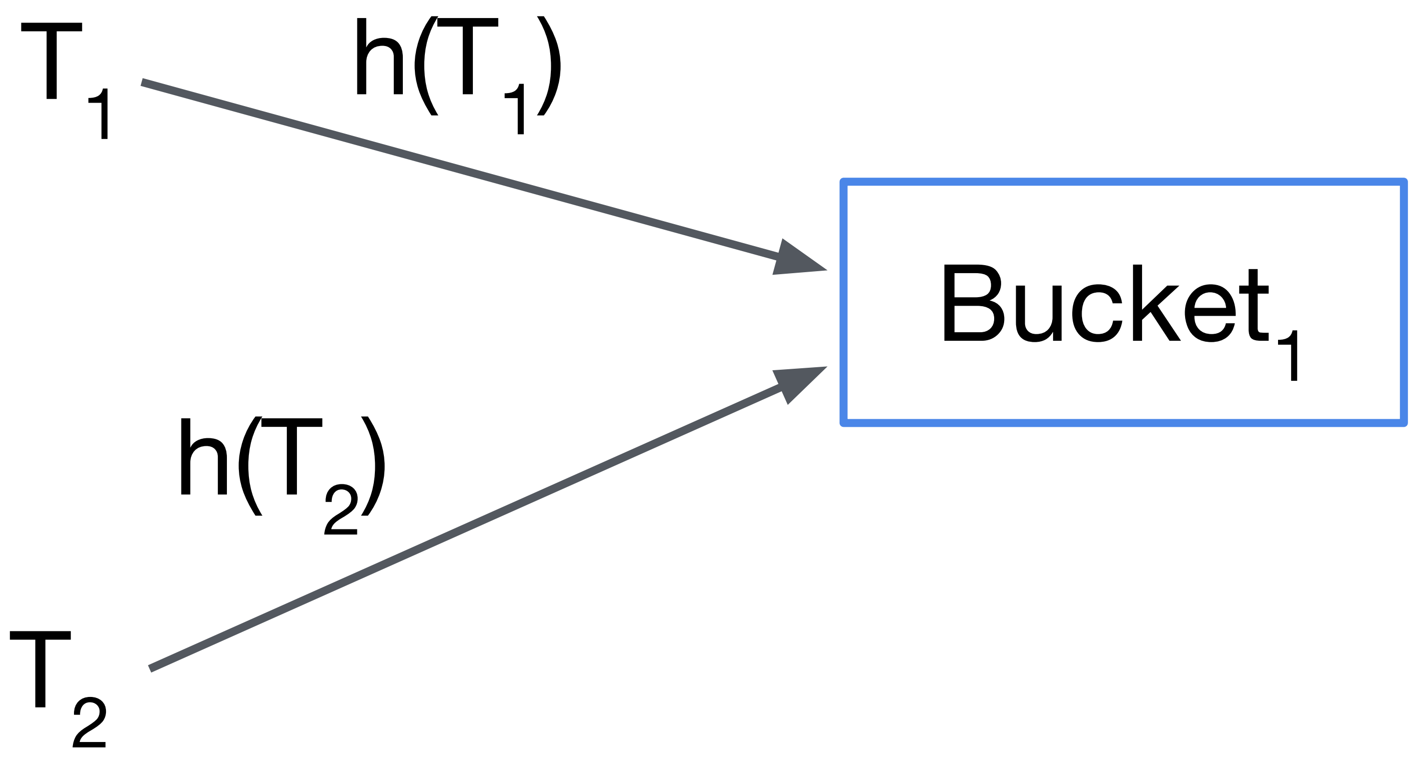
A trip often has thousands of shingles



Very hard problem

Approximate Solution

Trip T_1 & Trip T_2 are similar



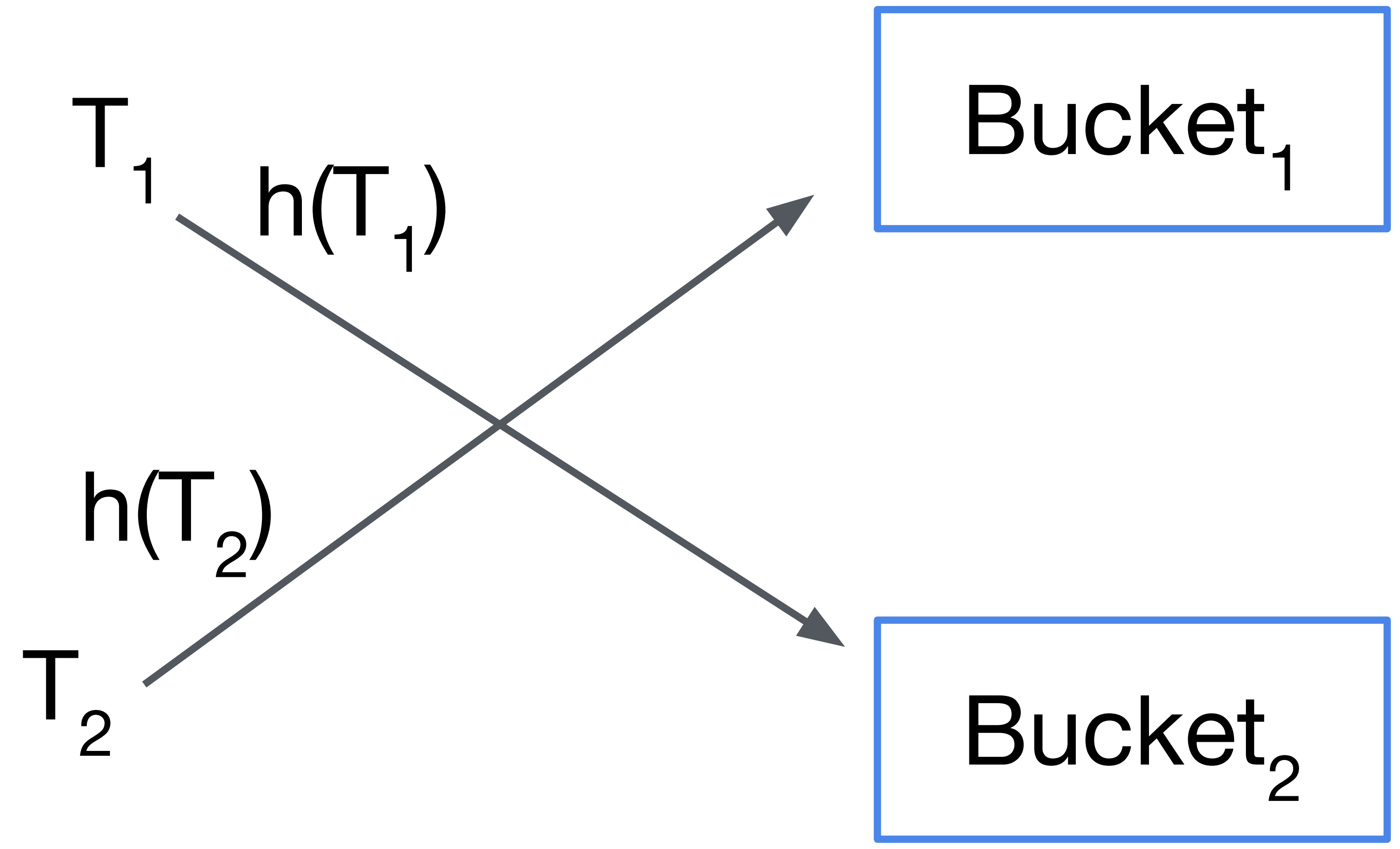
$D(T_1 , T_2)$ is small

$$\Downarrow$$

With high probability T_1 and T_2 are hashed into the same bucket.

Approximate Solution

Trip T_1 & Trip T_2 are not similar



$D(T_1, T_2)$ is large

$\Downarrow$

With high probability T_1 and T_2 are hashed into the different buckets.

Some distance functions have good companions of hash functions.

For Jaccard distance, it is MinHash function.

MinHash(S) = min { h(x) for all x in the set S }

h(x) is hash function such as (ax + b) % m where a & b are some *good* constants and m is the number of hash bins

Example:
S = {26, 88, 109}
h(x) = (2x + 7) % 8
MinHash(S) = min {3, 7, 1} = 1

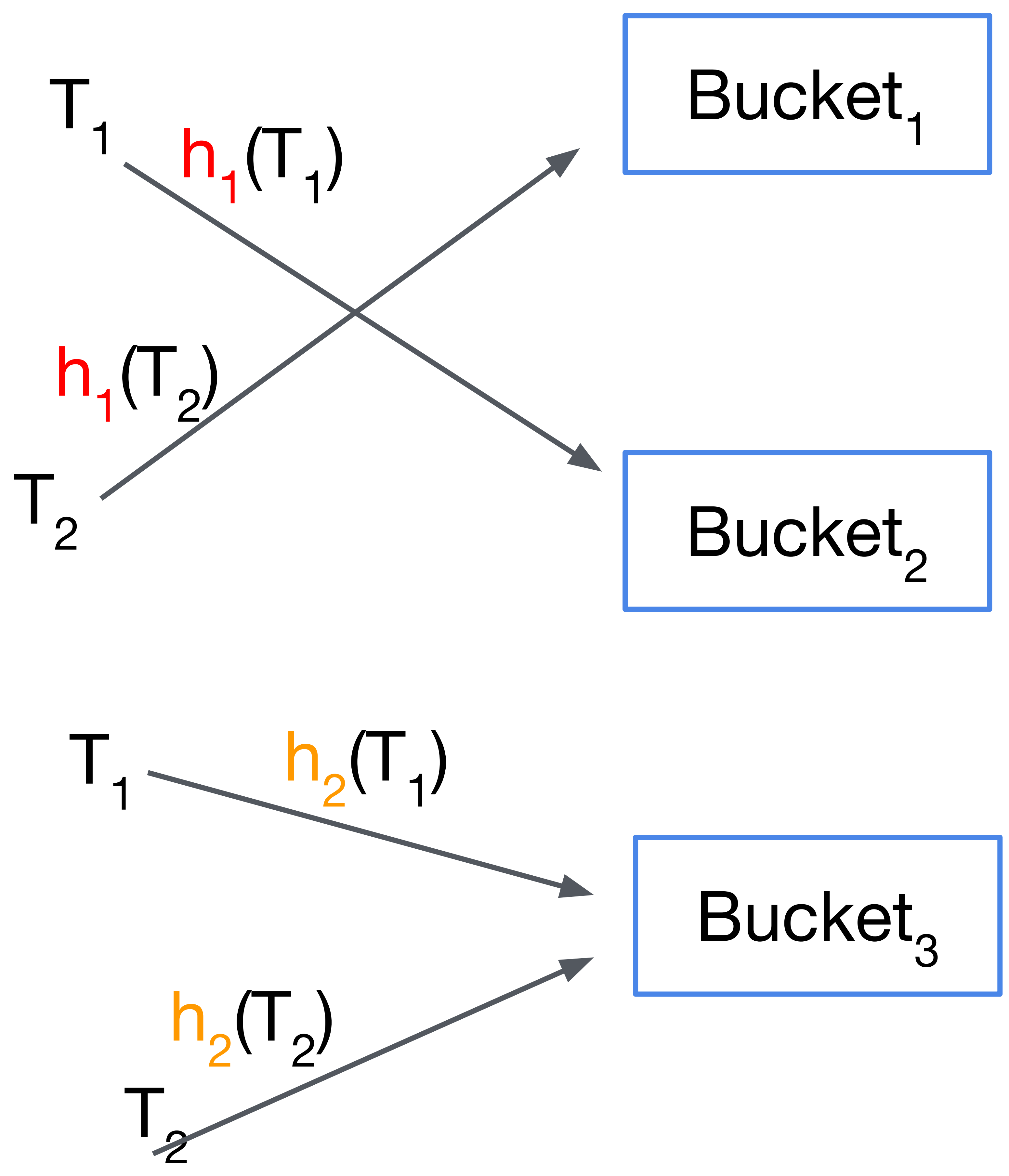
Some Other Examples

Distance	Hash Function
Jaccard	MinHash
Hamming	i-th value of vector x
Cosine	Sign of the dot product of x and a random vector

How to increase and control the probability?

It turns out the solution is very intuitive.

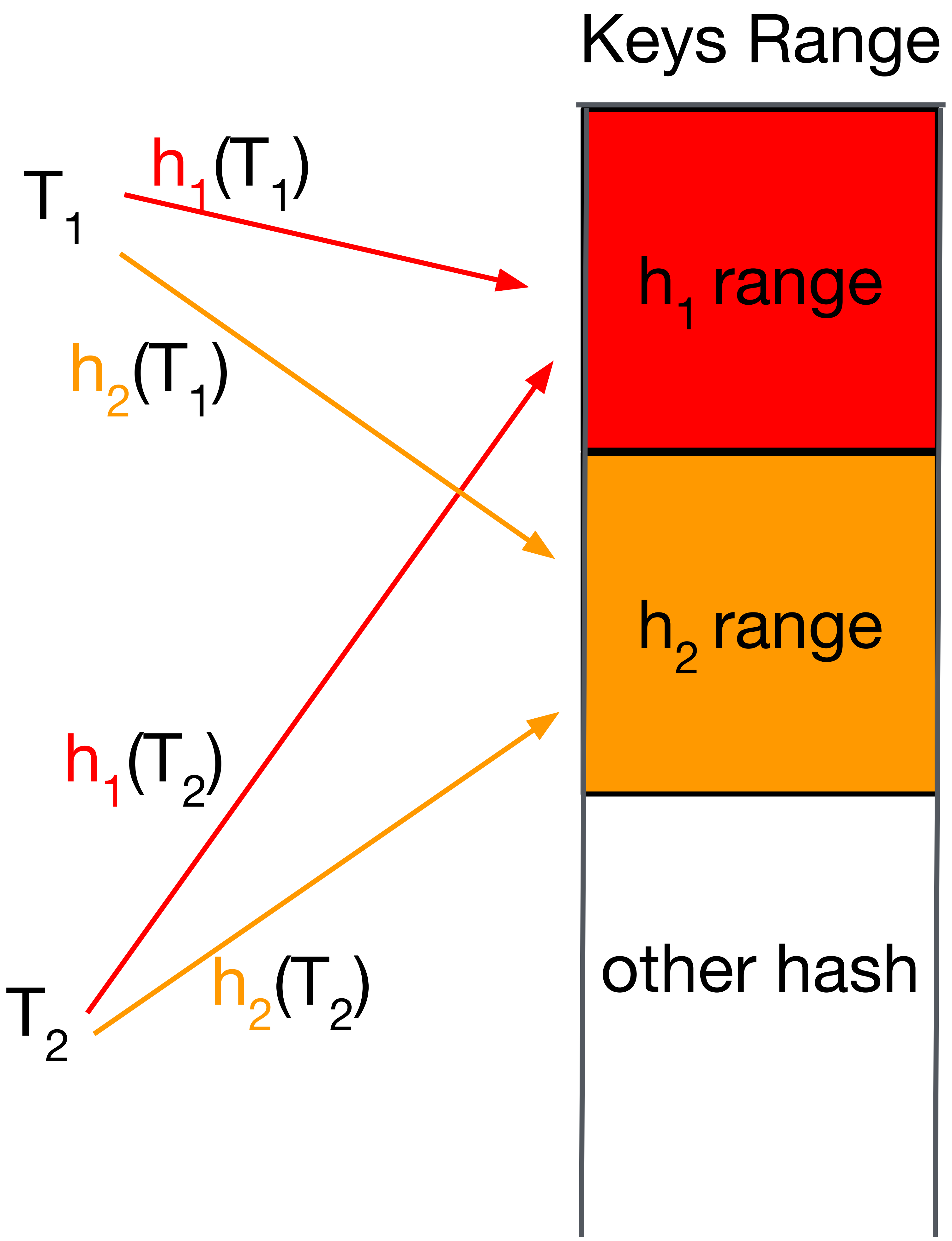
Use Multiple Hash



Both h_1 and h_2 are MinHash, but with different parameters (e.g. a & b)

Our Approach of LSH on Spark

Shuffle Keys



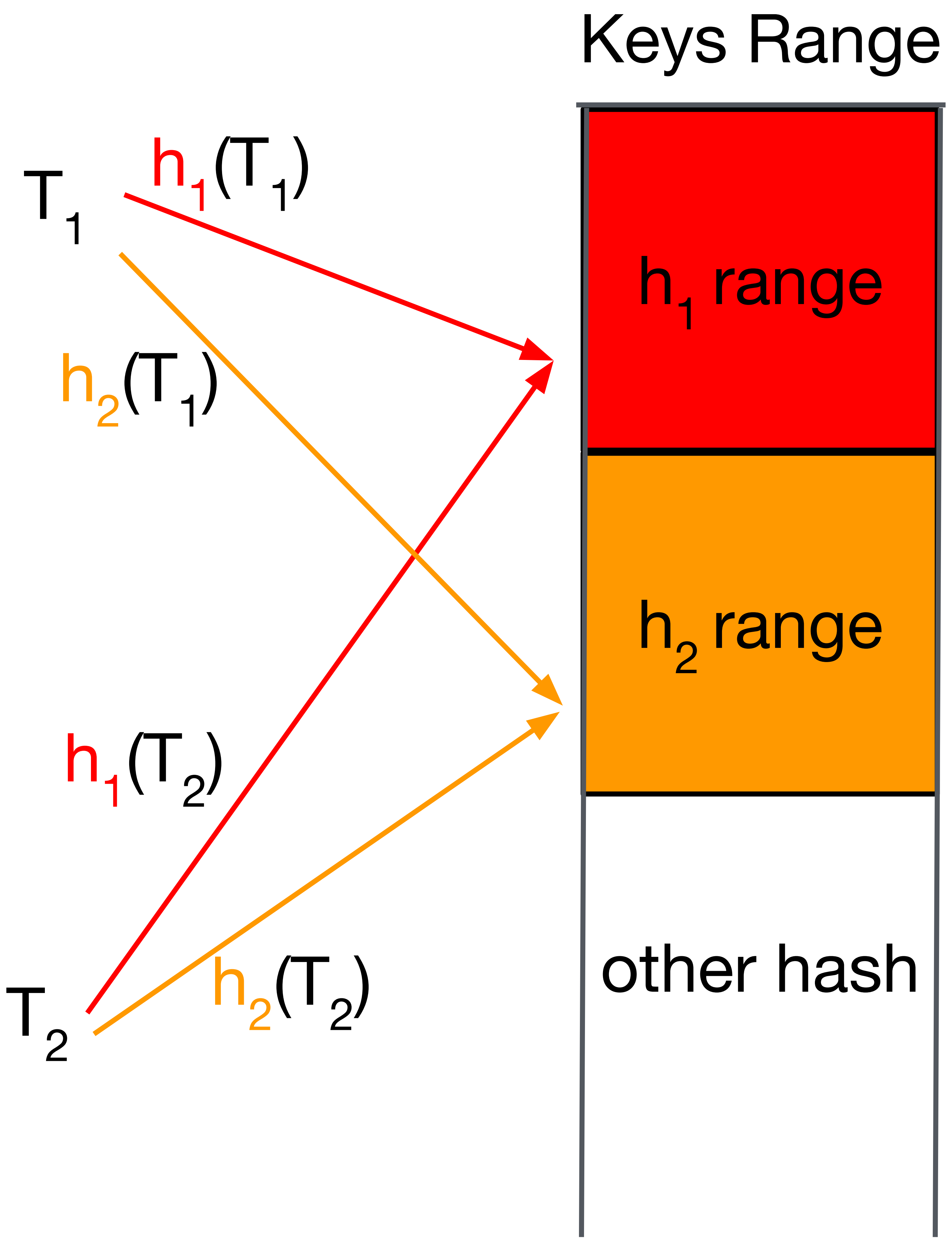
- RDD[Trip]
- The hash values are shuffle keys
- h_1 and h_2 have non-overlapping key ranges
- groupByKey()

Post Processing

Bucket$_1$ T_1, T_2

- If T_1 and T_2 are hashed into the same bucket, it's likely that they are similar.
- Compute the Jaccard distance.

Approach 2



- Same pair of trips are matched in both h_1 and h_2 buckets
- Use one more shuffle to dedup
- Network vs Distance Computation

Approach 3

- Don’t send the actual trip vector in the LSH and Dedup shuffles
- Send only the trip ID
- After dedup, join back with the trip objects with one more shuffle
  - Then compute the Jaccard distance of each pair of matched trips.
- When the trip object is large, Approach 3 saves a lot of network usage.

How to Generate Thousands of Hash Functions

- Naive approach
  - Generate thousands tuples of (a, b, m)
- Cache friendly approach - CPU register/L1/L2
  - Generate only two hash functions
  - $h_1(x) = (a_1x + b_1) \% m_1$
  - $h_2(x) = (a_2x + b_2) \% m_2$

$h_i(x) = h_1(x) + i * h_2(x)$ i from 1 to number of hash functions

Other Features

- Amplification
  - Improve the probabilities
  - Reduce computation, memory and network used in final post-processing
  - More hashing (usually insignificant compared to the cost in final post-processing)
- Near Neighbors Search
  - Used in information retrieval, instances based machine learning

Other Applications of LSH

- Search for top K similar items
  - Documents, images, time-series, etc
- Cluster similar documents
  - Similar news articles, mirror web pages, etc
- Products recommendation
  - Collaborative filtering

Future Work

- Migrate to Spark ML API
  - DataFrame as first class citizen
  - Integrate it into Spark
- Low latency inserts with Spark Streaming
  - Avoid re-hashing when new objects are streaming in

Thank you