Real-Time Clustering for Large Sparse Online Visitor Data

Gromit Yeuk-Yin Chan[1], Fan Du[2], Ryan Rossi[2], Anup Rao[2], Eunyee Koh[2], Claudio T. Silva[1], Juliana Freire[1]

[1] New York University

[2] Adobe

THE WEB CONFERENCE | TAIPEI 2020

Online Visitor Behavior are Large and Sparse

Predefined traits

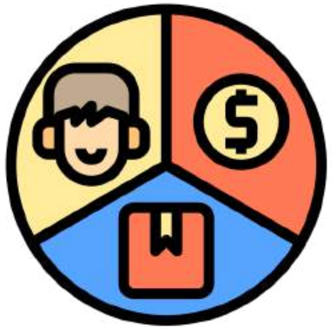
Millions of online customers everyday

How to identify similar online behavior? ➔ Cluster the similar customers!

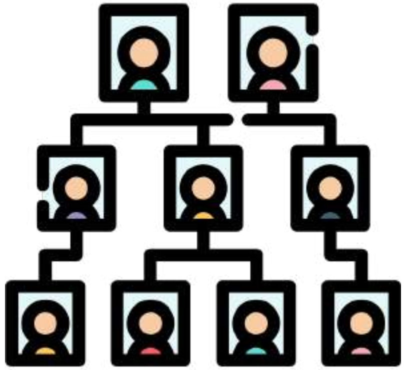
Interactive Cluster Analysis Is Important

Marketing Analyst

Identify optimal clusters



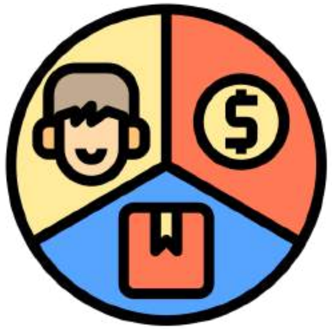
Multi-level clusters



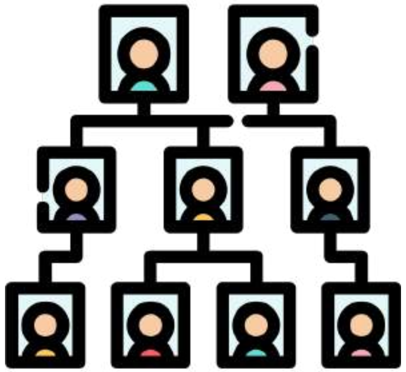
Interactive Analysis Is Important

Marketing Analyst

Identify optimal clusters



Multi-level segments



Interactive Data Analysis
(playing with different parameters)

Challenges as a Data Platform Provider

- Lots of recalculations
  - To find the “best” clusters, users need to try different combinations of parameters.

- Resource Utilization
  - Each time a clustering is started from scratch, a job is submitted to the distributed system.

Challenges as a Data Platform Provider

- Lots of recalculations
  - To find the "best" clusters, users need to try different combinations of parameters.
- Resource Utilization
  - Each time a clustering is started from scratch, a job is submitted to the distributed system.

Solution 1	Solution 2
lightweight and fast end-to-end clustering (e.g. K Means)	Pre-compute a hierarchy of cluster membership (e.g. linkage clustering)

Challenges as a Data Platform Provider

- Lots of recalculations
  - To find the “best” clusters, users need to try different combinations of parameters.

- Resource Utilization
  - Each time a clustering is started from scratch, a job is submitted to the distributed system.

Solution 1	Solution 2 ✓
lightweight and fast end-to-end clustering (e.g. K Means)	Pre-compute a hierarchy of cluster membership (e.g. linkage clustering)

The quality of interactive analysis will be affected if the result does not arrive within 500ms (Liu & Heer TVCG 2014)

Motivation

- Linkage Clustering (main technique to construct a hierarchy)
  - Requires a pairwise distance matrix
  - Impossible in terms of memory and time ($O(n^2)$) for moderate size data

- Application to Distributed System
  - Parallel algorithm
  - Even data distribution among the computation nodes.

Motivation

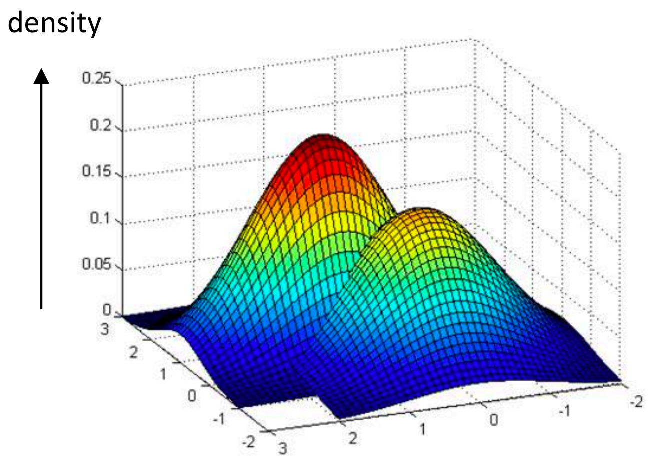
- Linkage Clustering (main technique to construct a hierarchy)
  - Requires a pairwise distance matrix
  - Impossible in terms of memory and time ($O(n^2)$) for moderate size data

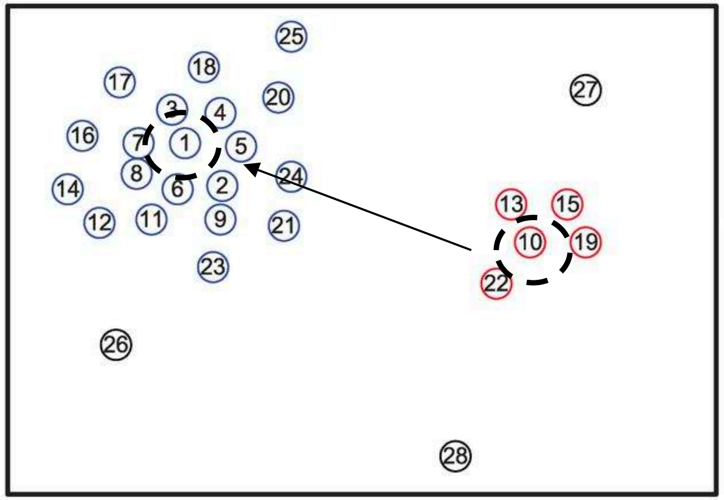
- Application to Distributed System
  - Parallel algorithm
  - Even data distribution among the computation nodes.

Can we create linkage without all pairwise similarity calculations in the distributed system?

Density Peaks Clustering

- Density Peaks (DP) Clustering
  - Density: number of neighbors around a point.
  - Assumption: if a point's **closest higher density point** is far away, the point is likely to be a cluster center.
  - Parameter: cutoff distance (d_{cutoff})





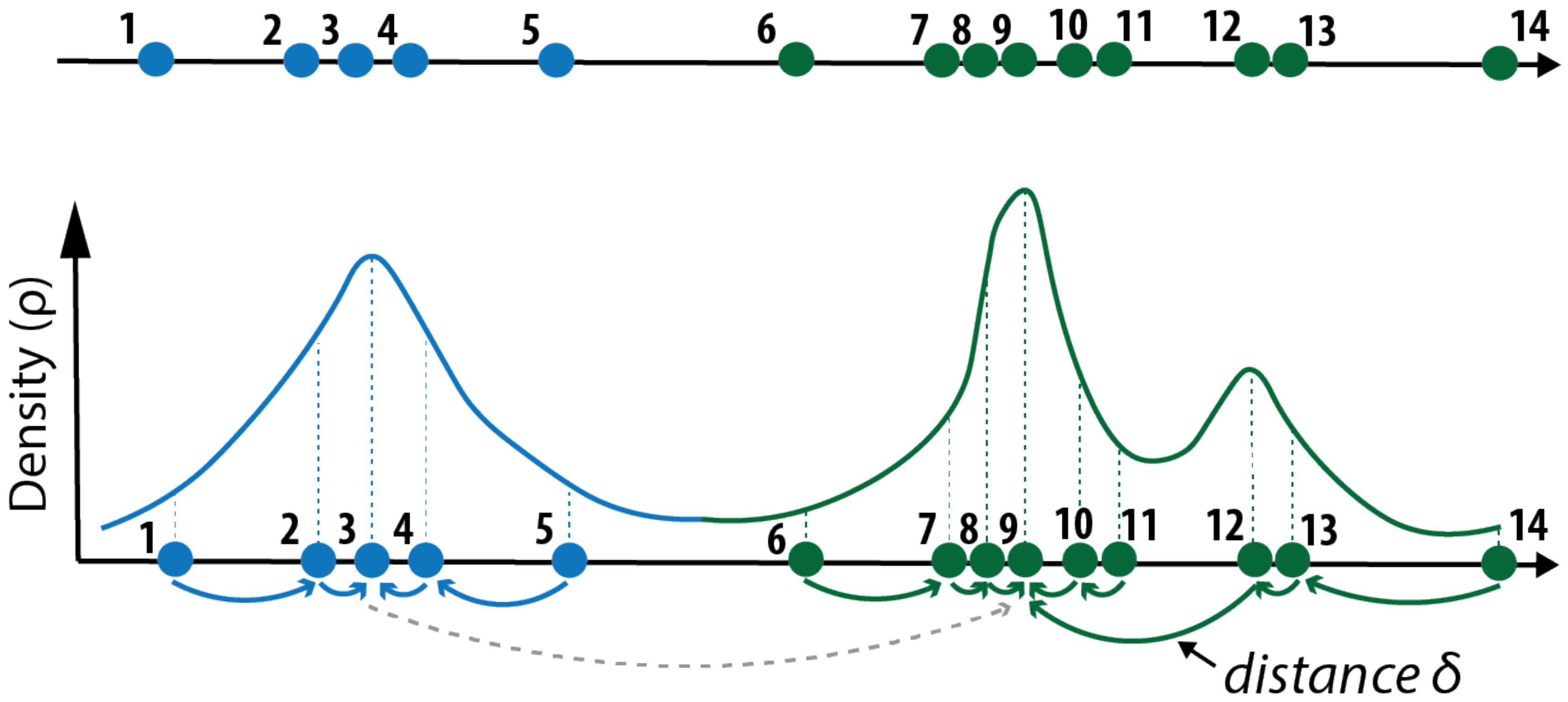
Rodriguez, Alex, and Alessandro Laio. "Clustering by fast search and find of density peaks." Science 344.6191 (2014): 1492-1496.

Density Peaks Clustering

Consider a 1-D example, we want to group the points with 2 clusters.

Density Peaks Clustering

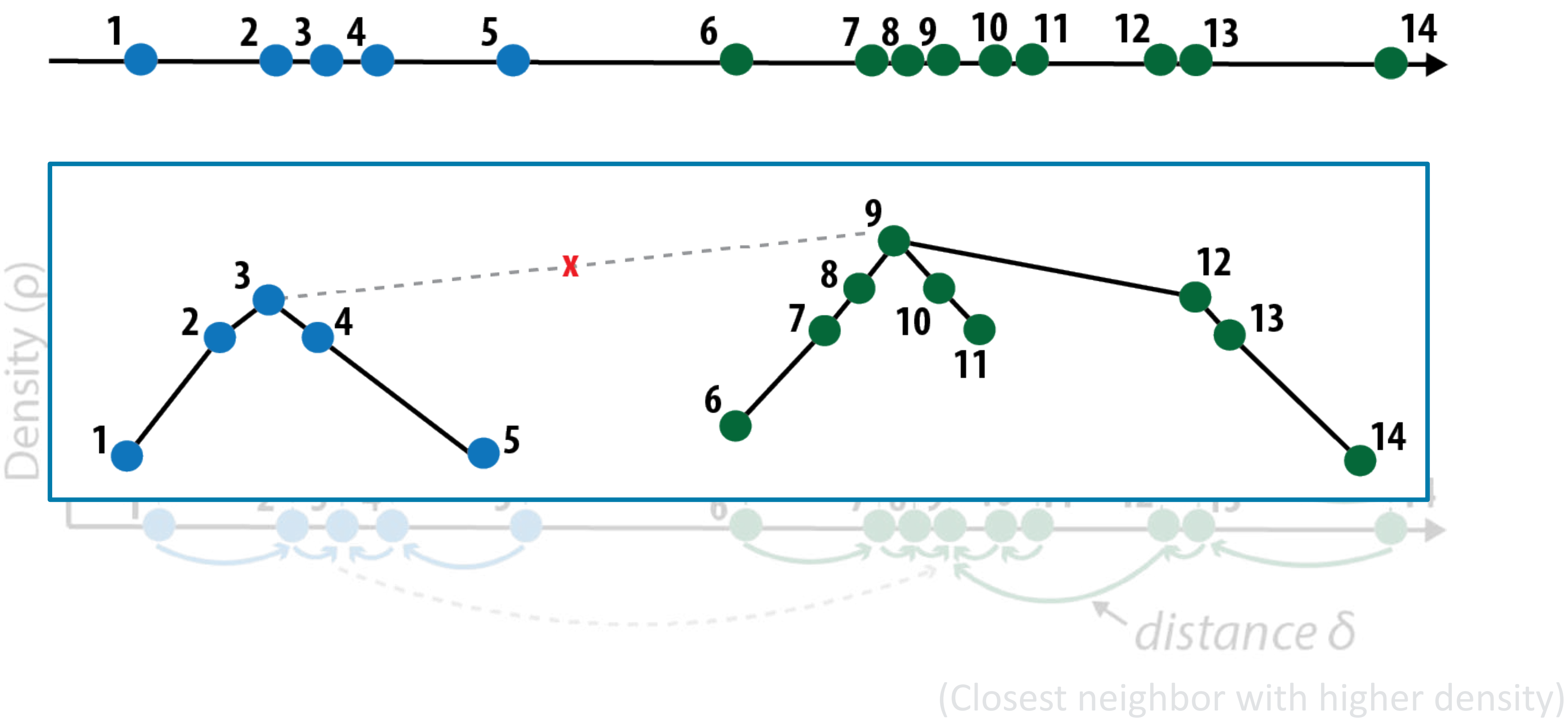
Consider an 1-D example, we want to group the points with 2 clusters.



(Closest neighbor with higher density)

Density Peaks Clustering

Consider a 1-D example, we want to group the points with 2 clusters.



Density Peaks Clustering

- Density Peaks (DP) Clustering Algorithm

1. Calculate density **ρ**

2. Calculate shortest distance of higher density Points **δ**

3. Sort the elements by density, then assign them to cluster same as the nearest neighbor of higher density.

Density Peaks Clustering

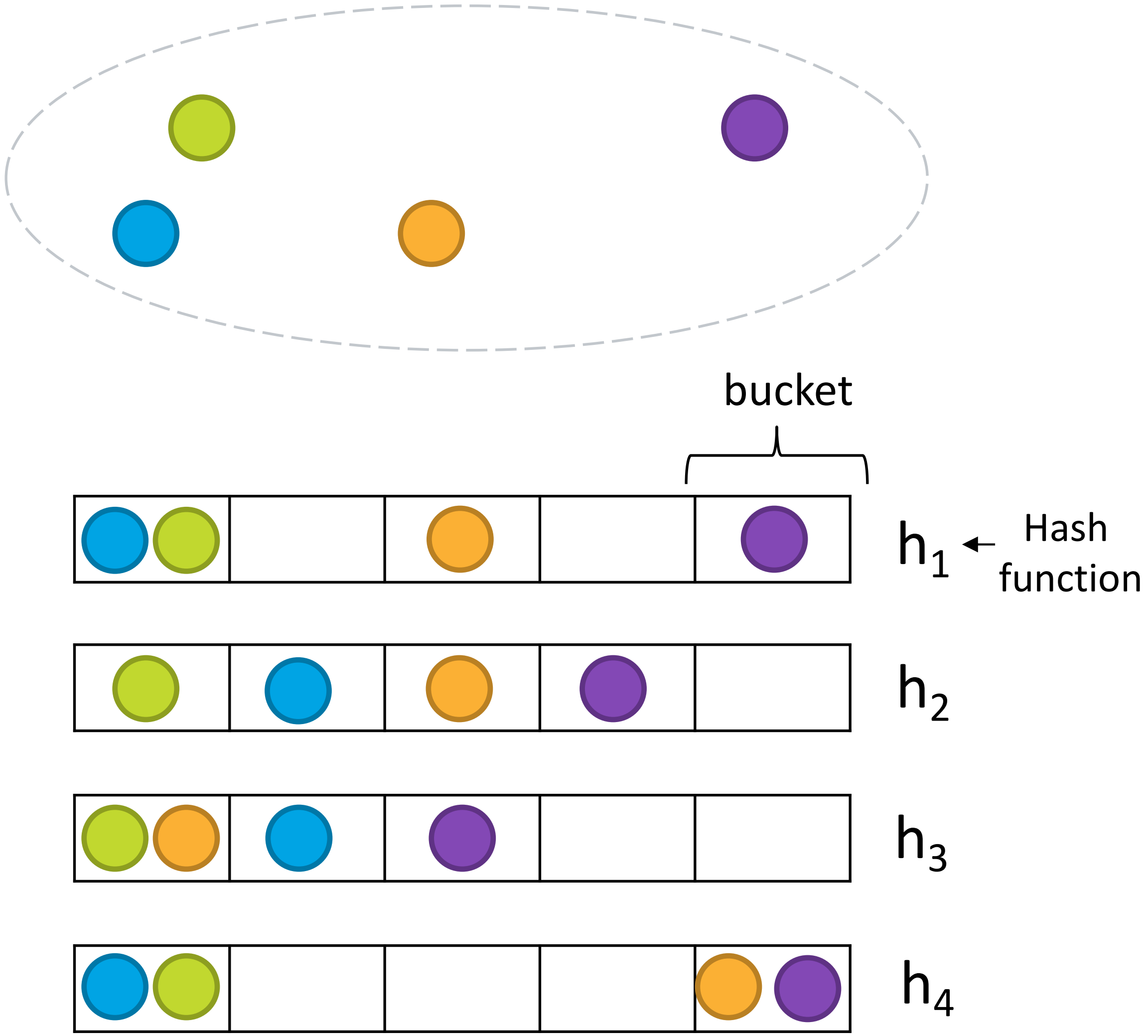
- Density Peaks (DP) Clustering Algorithm

1. Calculate density **ρ** $O(n^2)$
2. Calculate shortest distance of higher density Points **δ** $O(n^2)$
3. Sort the elements by density, then assign them to cluster same as the nearest neighbor of higher density. $O(n)$

Slow

Now the question:
How to make the preprocessing faster?

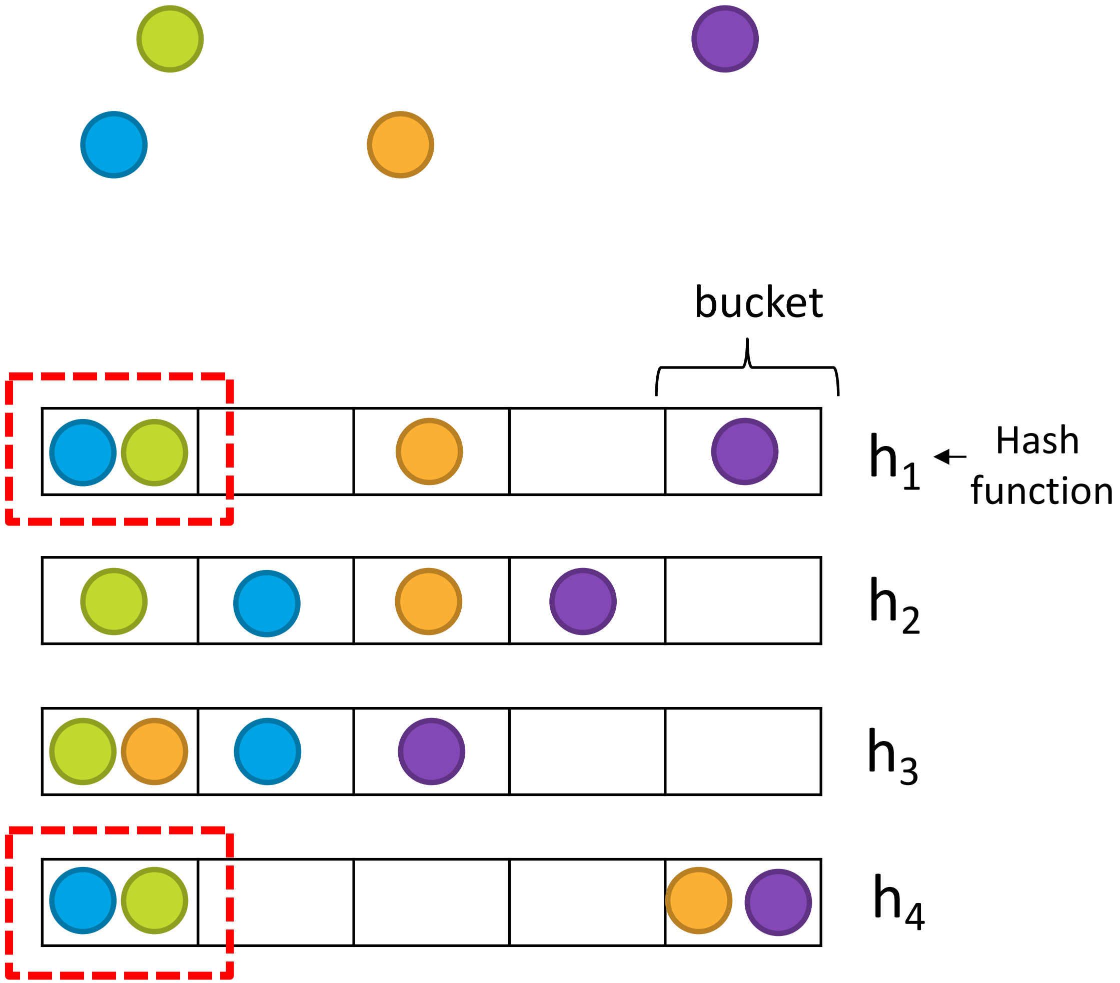
Locality Sensitive Hashing for Neighbor Querying



Locality-sensitive hashing (LSH) hashes similar items to the same hash “bucket” with high probability.

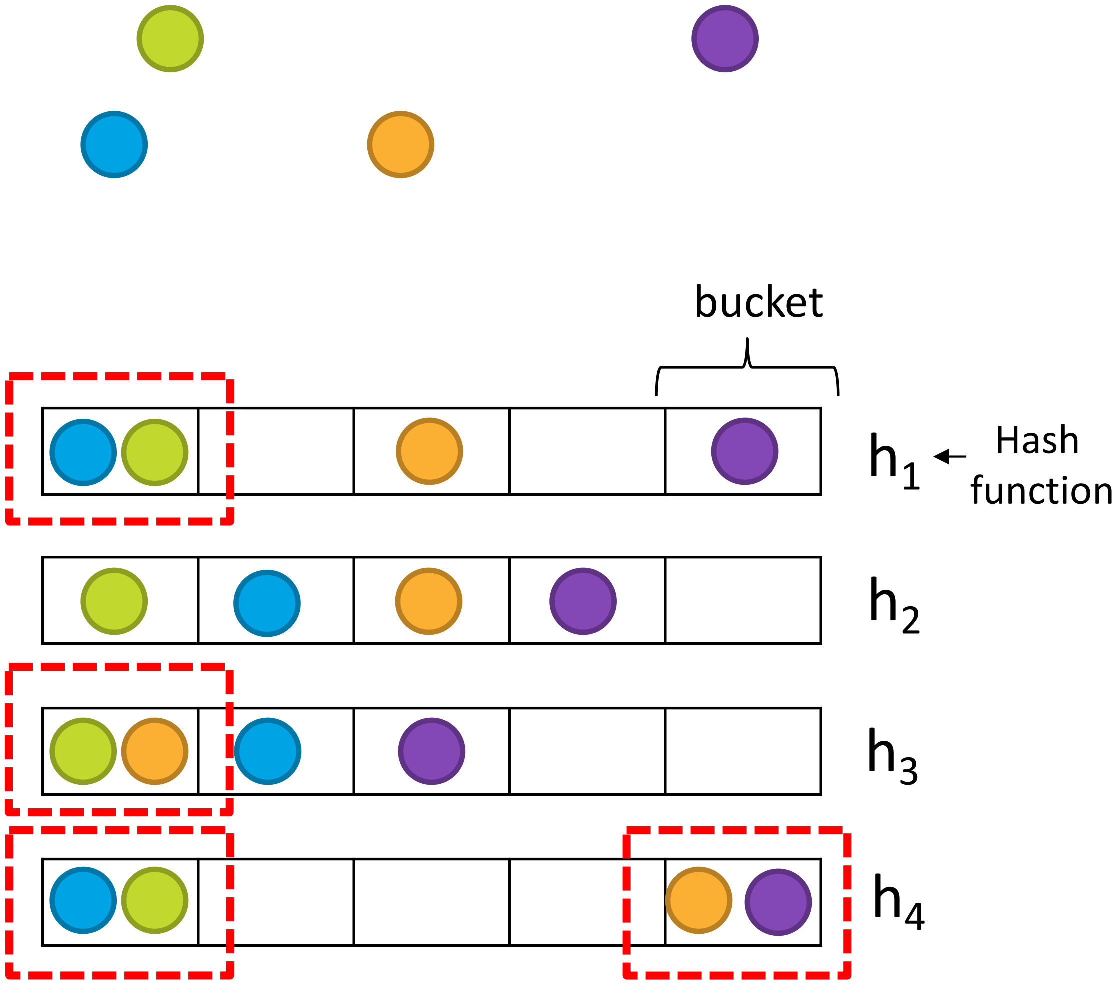
The greater number of times two items are hashed in the same bucket (*collision*), the higher the similarity they are.

Applying LSH to Density Peaks Calculation



To calculate the density **ρ** of each point, we query the points with high number of collisions.

Applying LSH to Density Peaks Calculation

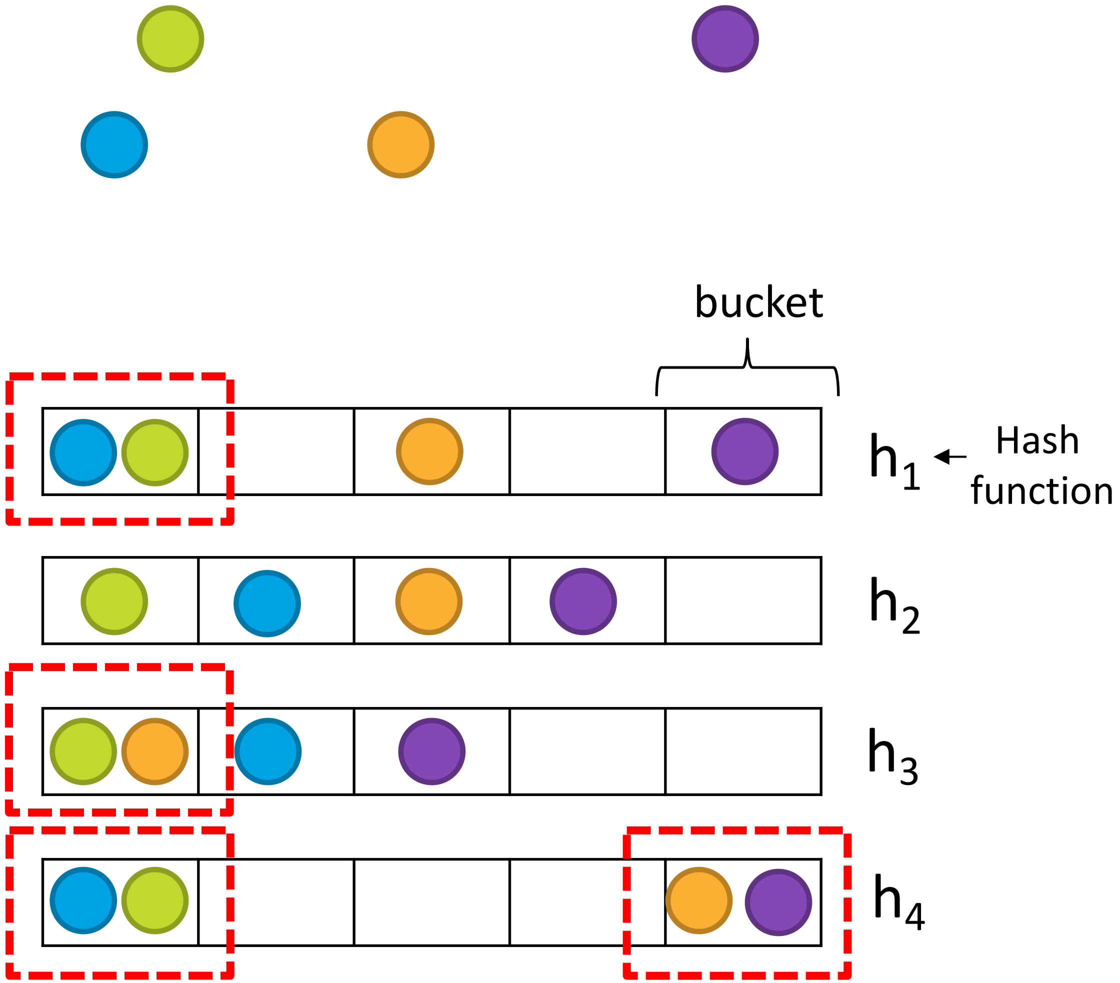


To calculate the density **ρ** of each point, we query the points with high number of collisions.

To retrieve distance **δ** of the nearest neighbor, we query the points with at least one collisions.

If a point does not have any queried result, we directly compare it with points with highest density.

Applying LSH to Density Peaks Calculation



After the query, we also have the following information:

Exact similarity between the points

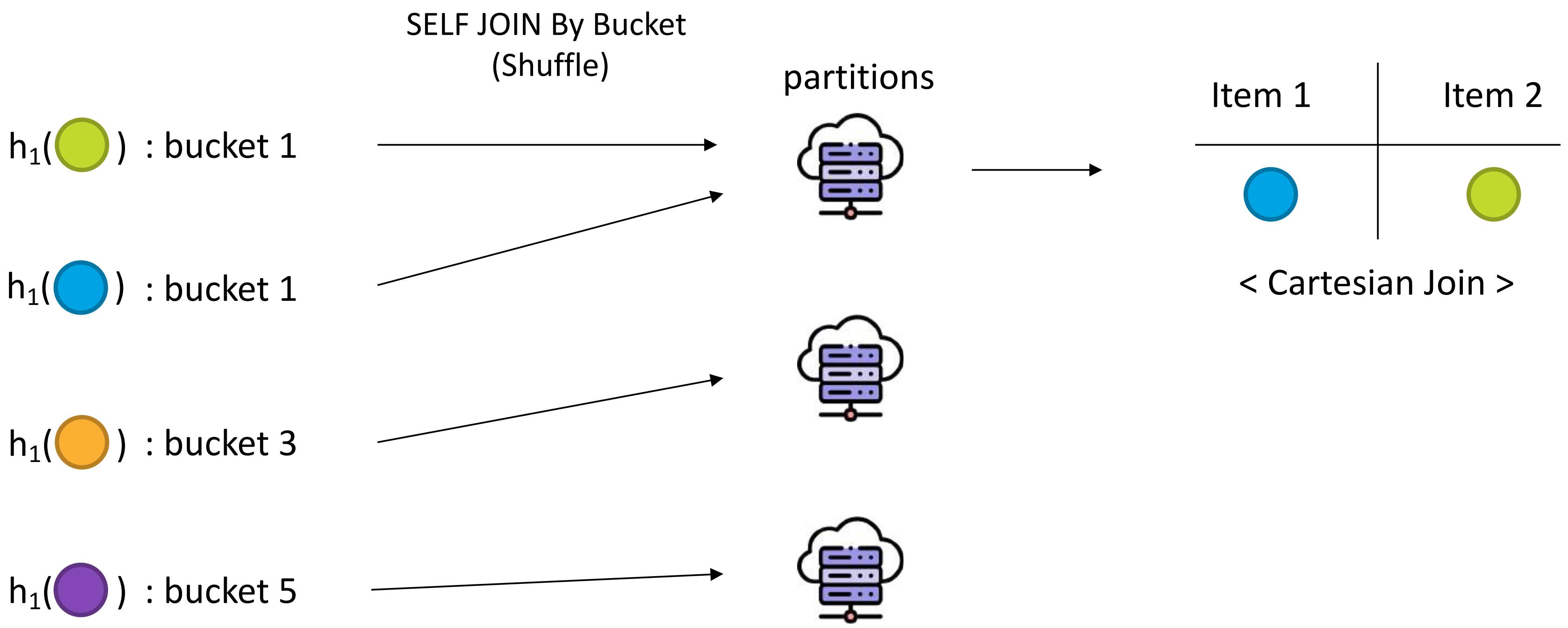
↓

Probability of collision between the points

Thus, it is possible to calculate the *join size estimation* to refine the *density and* the *accuracy* of *nearest neighbor* query. (Zhang et. al. TKDE 2016)

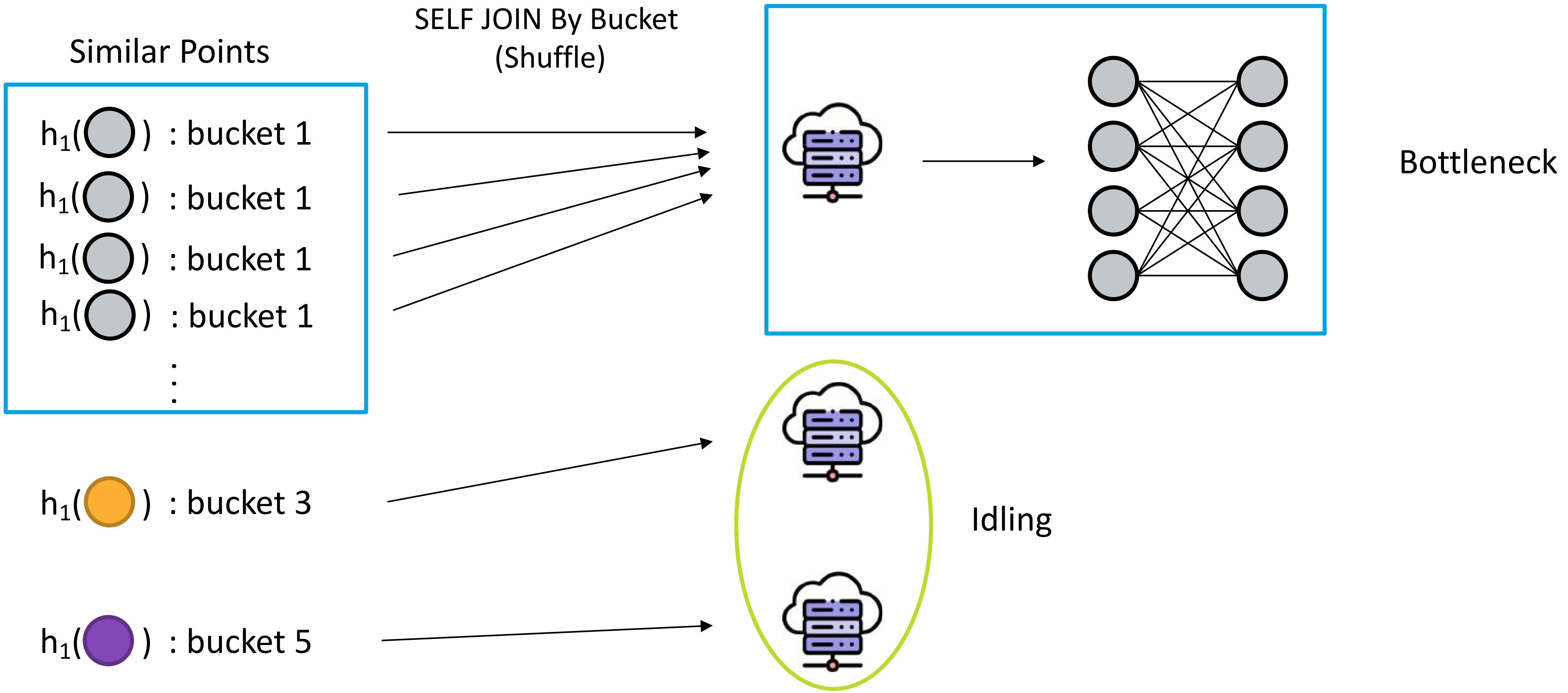
Parallelizing LSH Query in Spark

- Finding the items that collide with each other is a parallel process.



Parallelizing LSH Query in Spark

- A straightforward JOIN can lead to uneven data distributions.



Salting Strategies to Increase Parallelization

- Create Random JOIN keys to ''scatter'' the data

$h_1(\bigcirc)$: bucket 1

$h_1(\bigcirc)$: bucket 1

$h_1(\bigcirc)$: bucket 1

Salting Strategies to Increase Parallelization

- Create Random JOIN keys to ''scatter'' the data

Random Keys

h_1( ) : bucket 1

h_1( ) : bucket 1

h_1( ) : bucket 1

Salting Strategies to Increase Parallelization

- Create Random JOIN keys to ’’scatter’’ the data

Random Keys

$h_1($ $)$: bucket 1

$h_1($ $)$: bucket 1

$h_1($ $)$: bucket 1

Replicate 1

$h_1($ $)$: bucket 1

$h_1($ $)$: bucket 1

$h_1($ $)$: bucket 1

Replicate 2

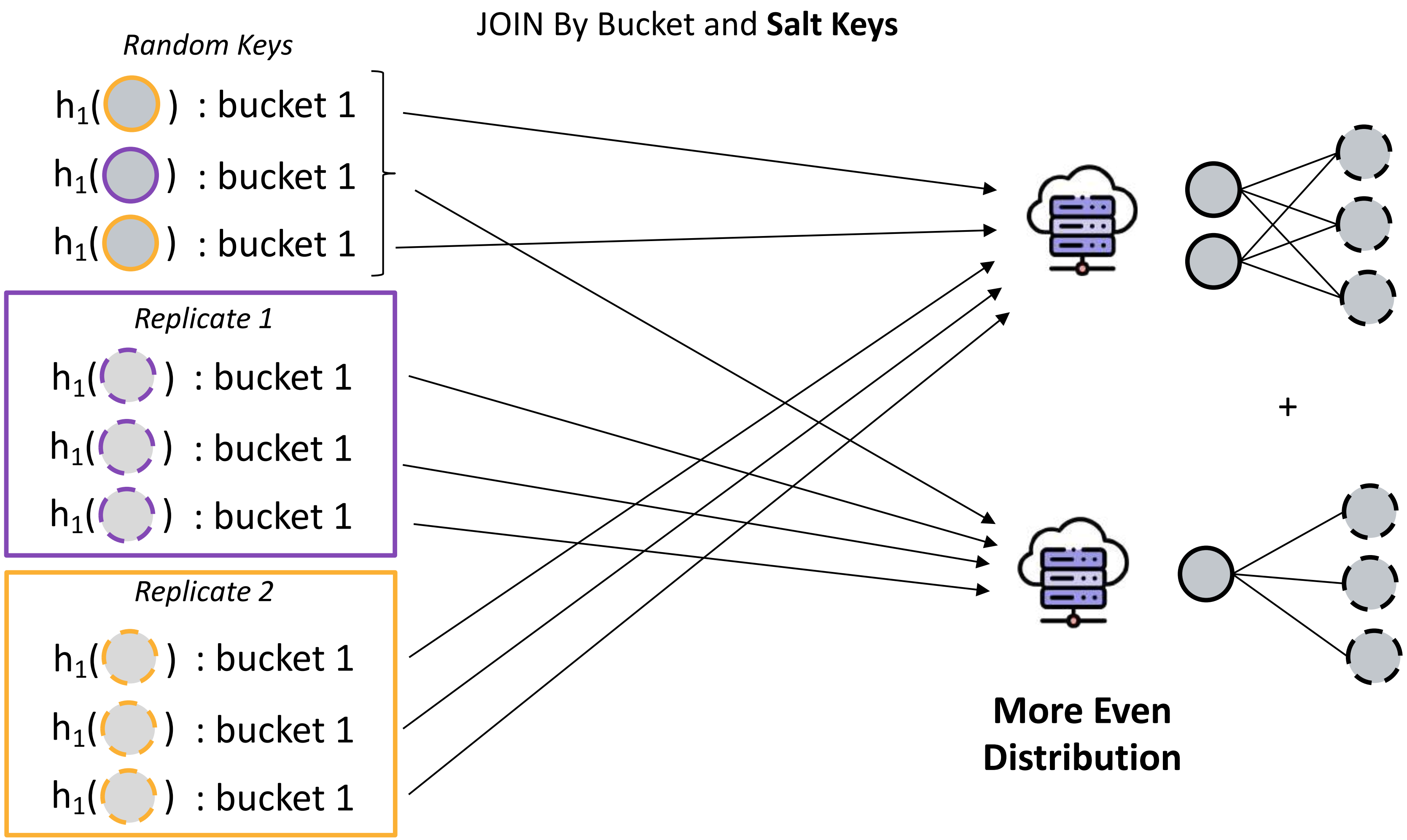
$h_1($ $)$: bucket 1

$h_1($ $)$: bucket 1

$h_1($ $)$: bucket 1

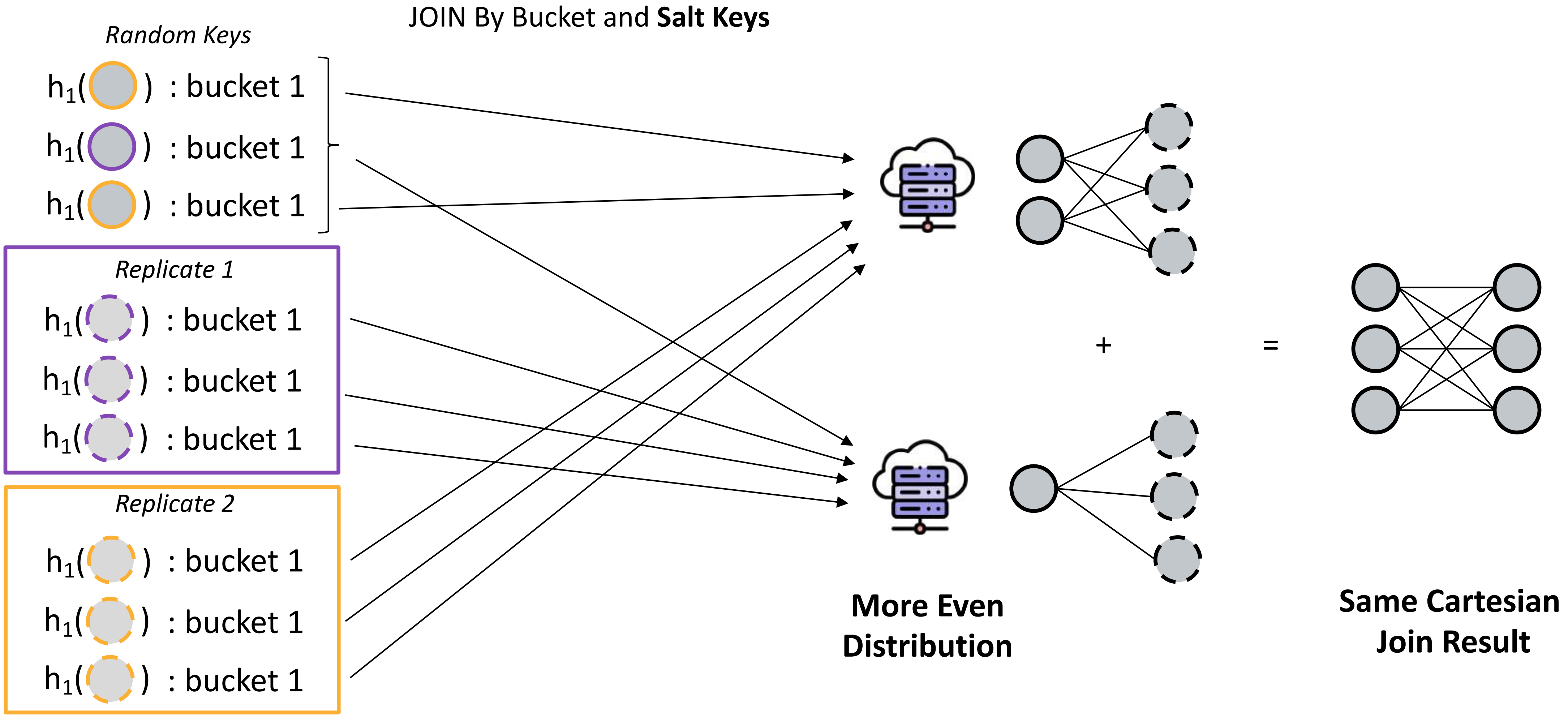
Salting Strategies to Increase Parallelization

- Create Random JOIN keys to ”scatter” the data



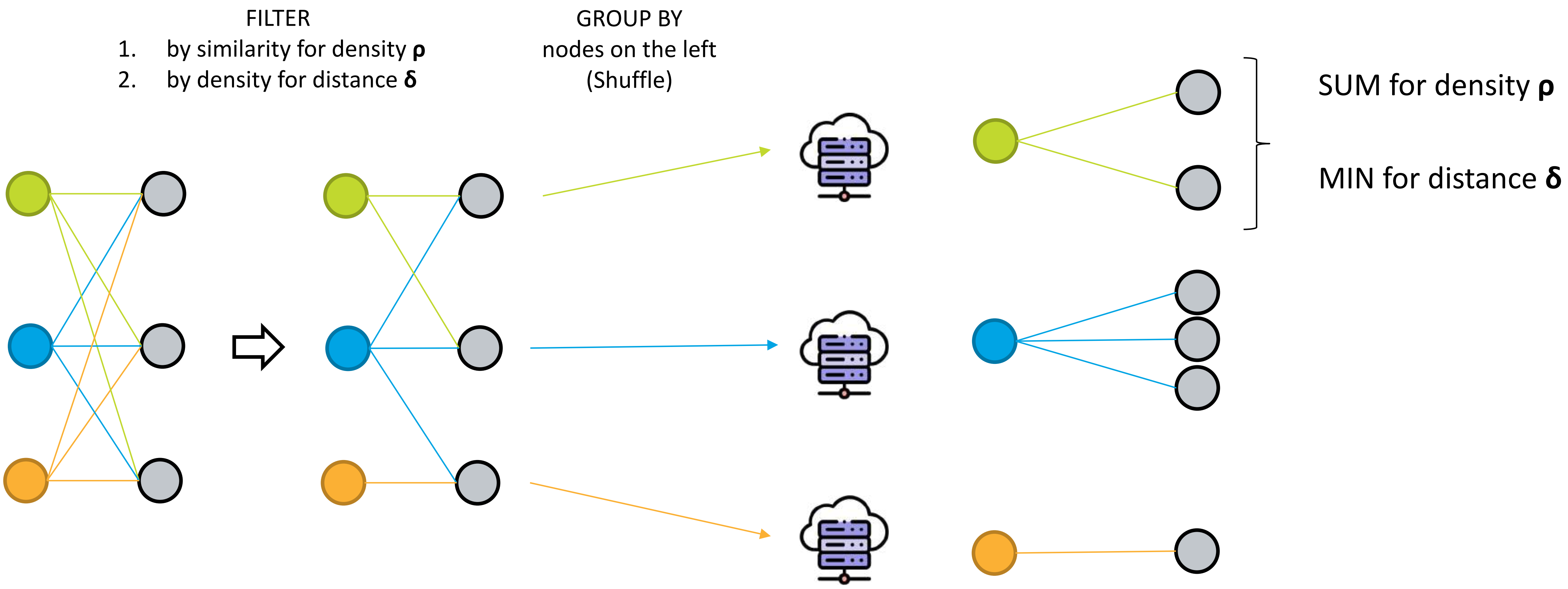
Salting Strategies to Increase Parallelization

- Create Random JOIN keys to ”scatter” the data



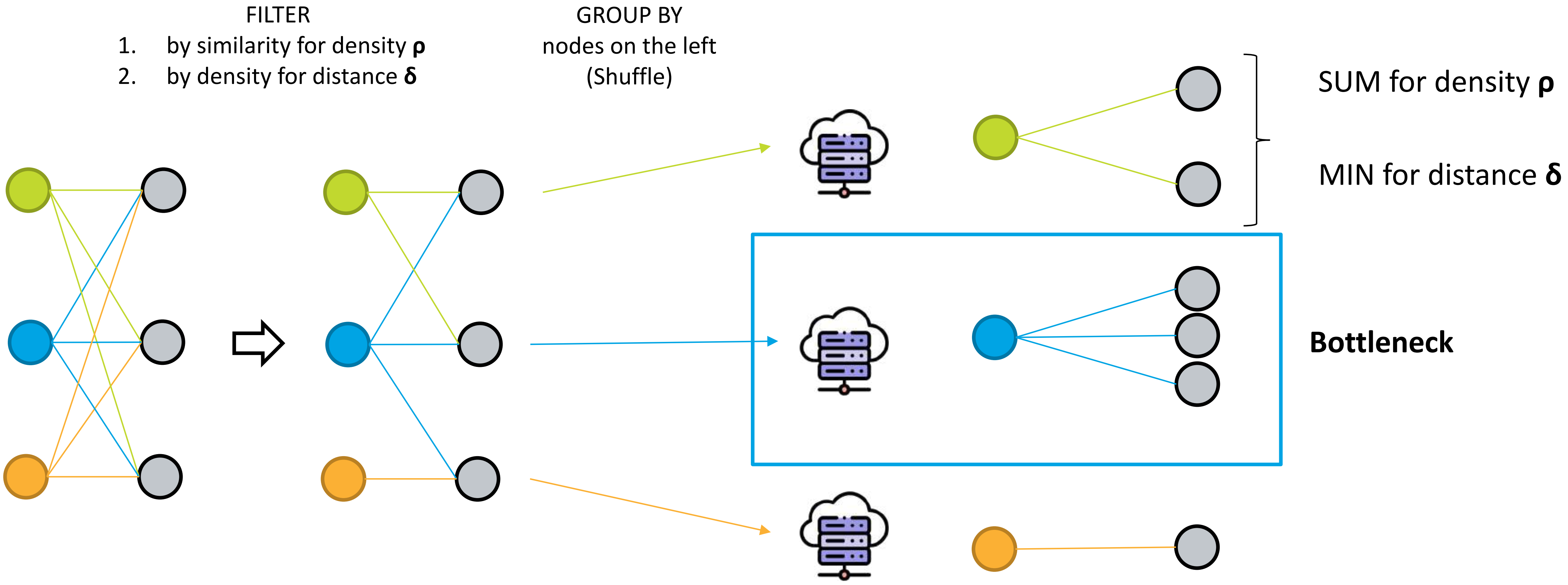
Group By Operations for Density and Distance

- The second stage to compute the Density Peaks can also be parallelized.



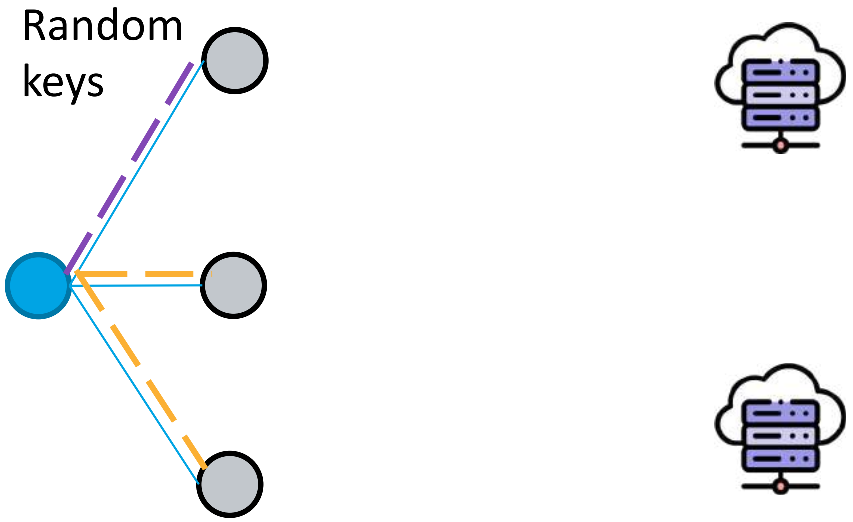
Group By Operations for Density and Distance

- The second stage to compute the Density Peaks can also be parallelized.



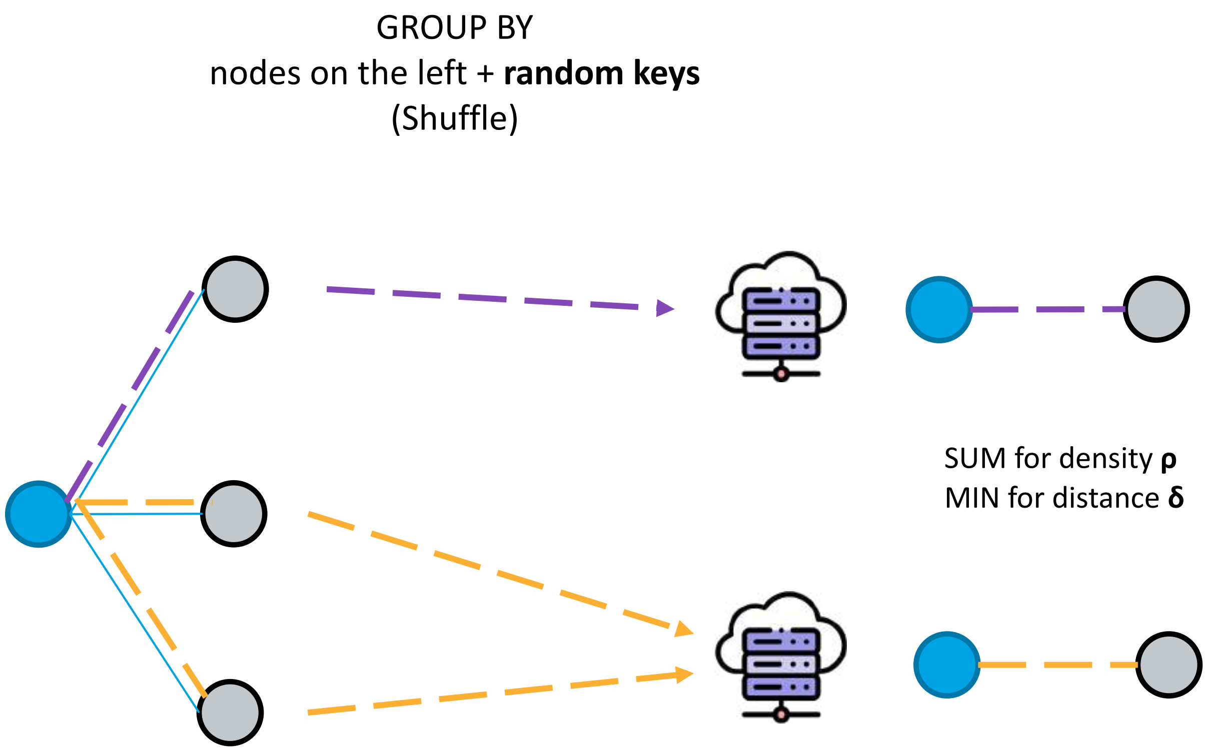
Salting Strategies for Group By Operations

- The second stage to compute the Density Peaks can also be parallelized.



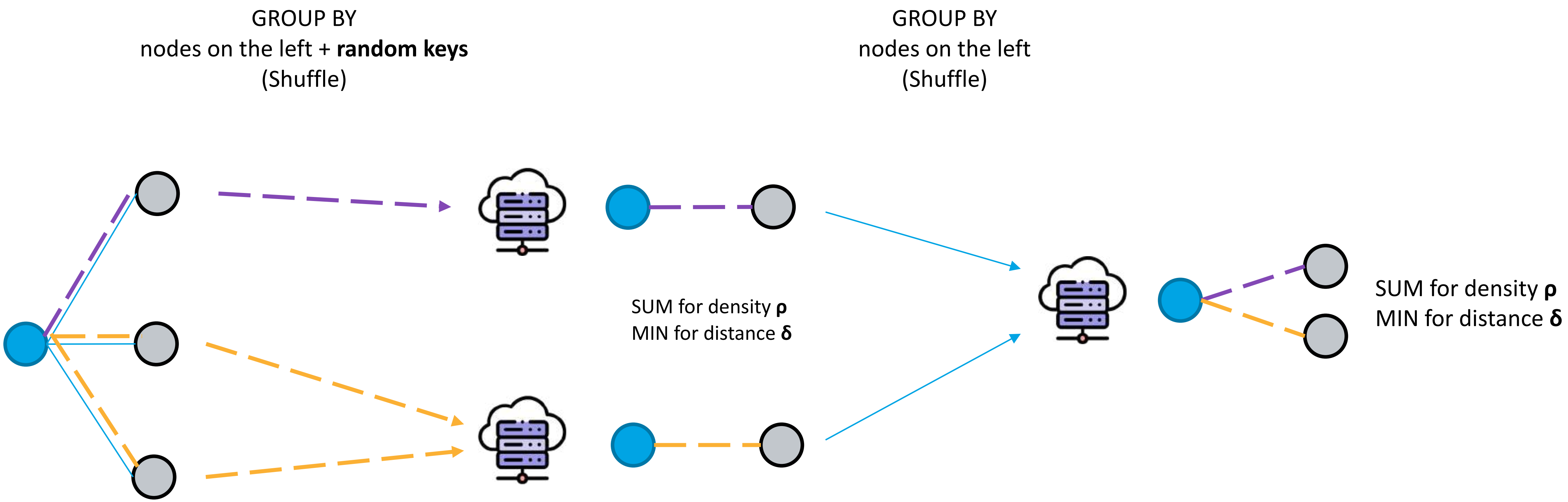
Salting Strategies for Group By Operations

- The second stage to compute the Density Peaks can also be parallelized.



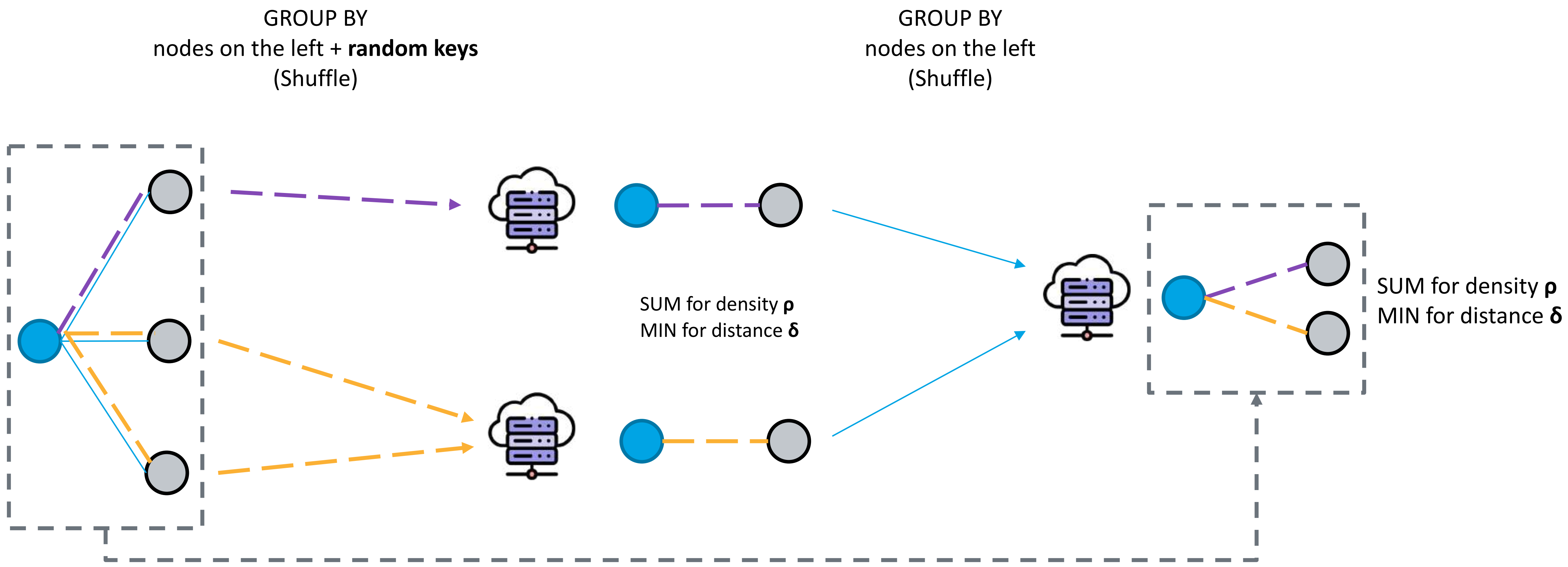
Salting Strategies for Group By Operations

- The second stage to compute the Density Peaks can also be parallelized.



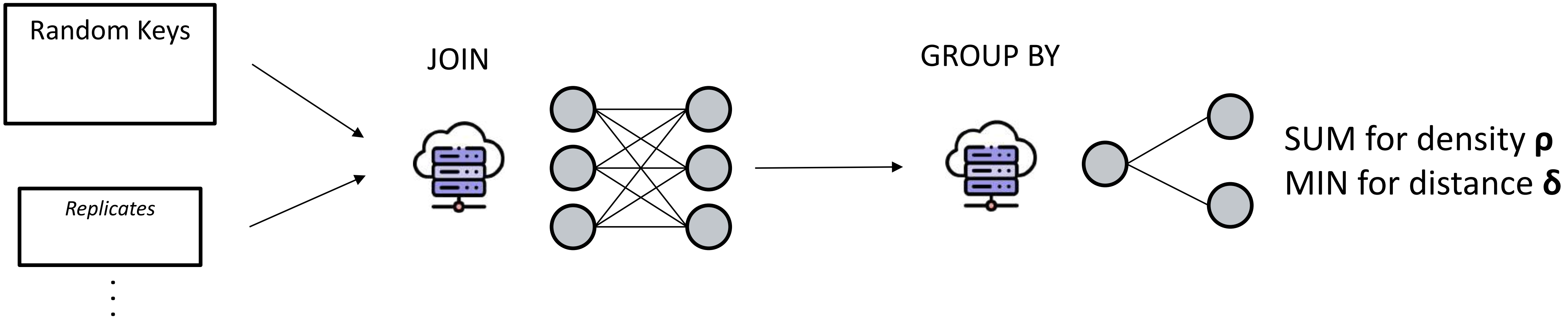
Salting Strategies for Group By Operations

- The second stage to compute the Density Peaks can also be parallelized.



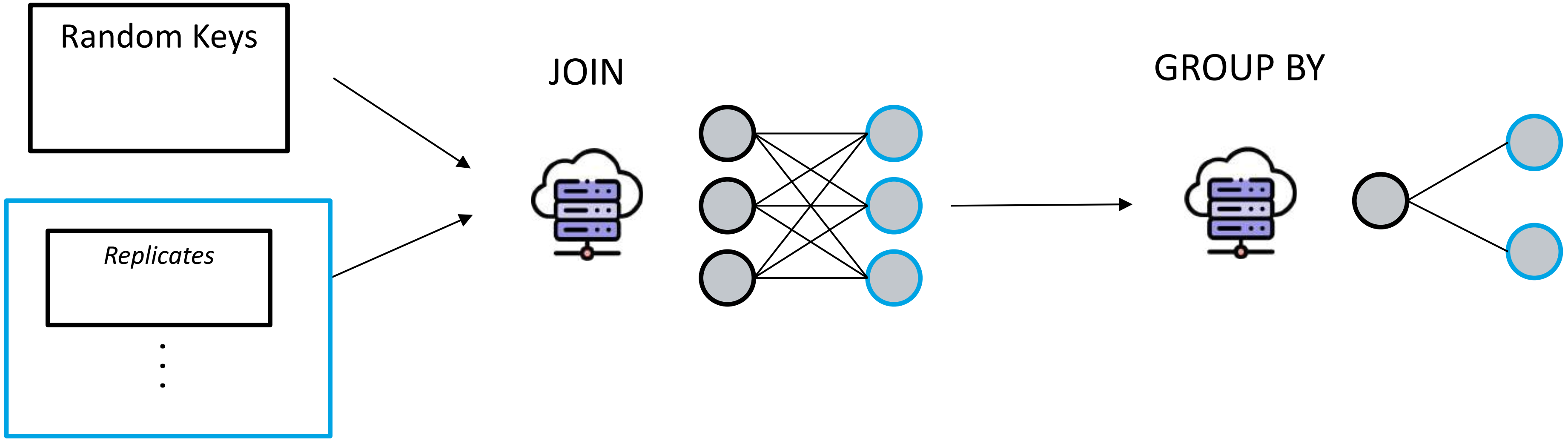
Batch Processing in the Pipeline

- The whole pipeline allows taking batches one by one.



Batch Processing in the Pipeline

- The whole pipeline allows taking batches one by one.

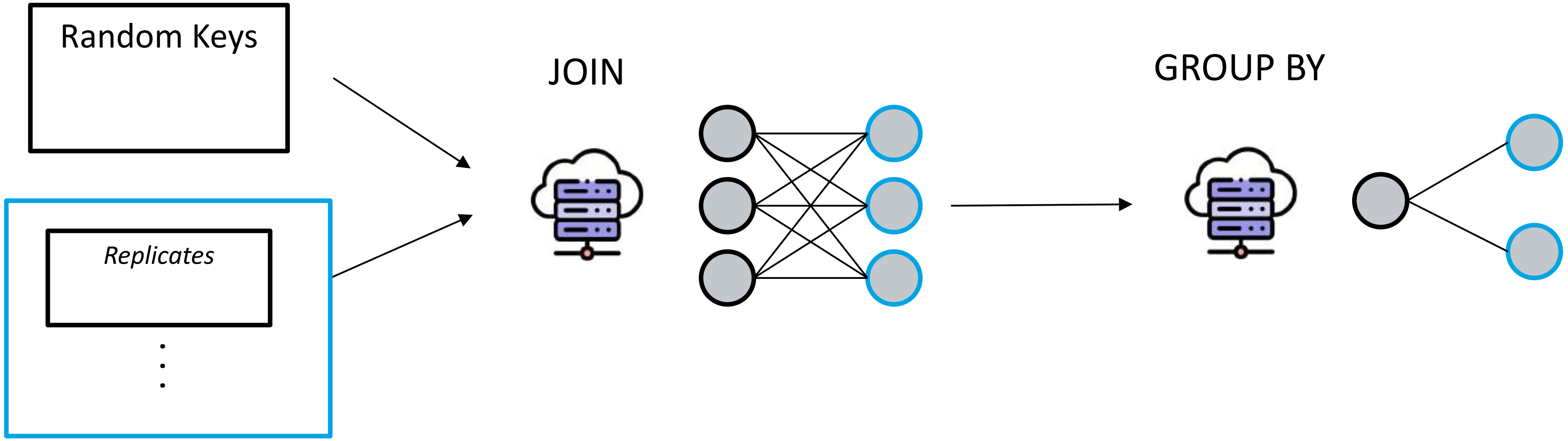


SUM for density **ρ**
MIN for distance **δ**

Instead of taking all data, we replace it with replicate for a small batch.

Batch Processing in the Pipeline

- The whole pipeline allows taking batches one by one.

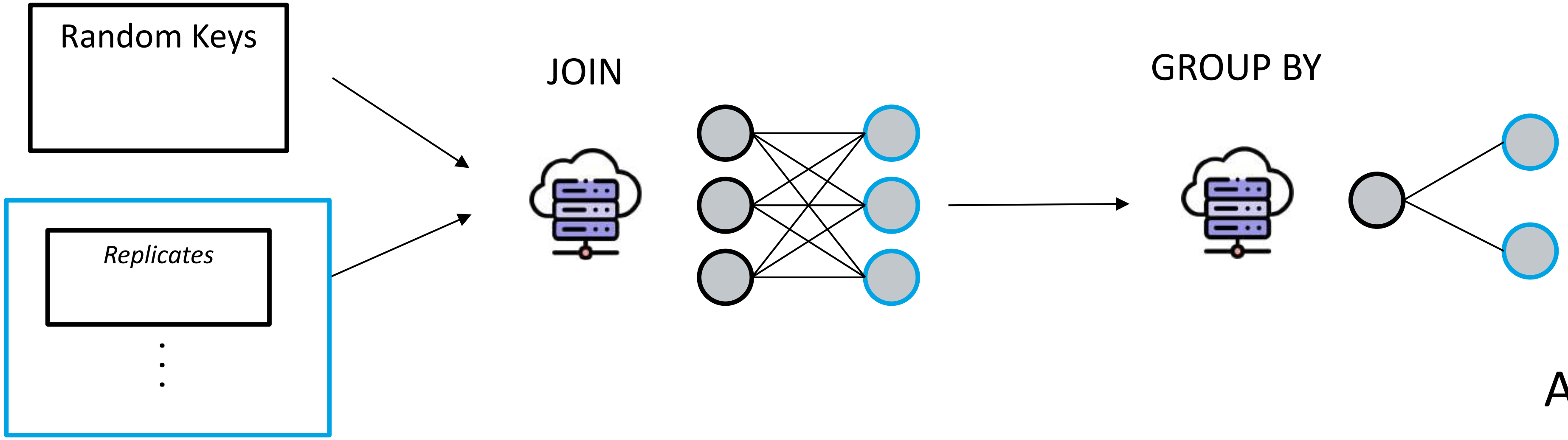


SUM for density $\boldsymbol{\rho}_t$
MIN for distance $\boldsymbol{\delta}_t$

Instead of taking all data, we replace it with replicate for a small batch.

Batch Processing in the Pipeline

- The whole pipeline allows taking batches one by one.



SUM for density **ρ**
MIN for distance **δ**

At batch t,

$\boldsymbol{\rho}_t$ = SUM + $\boldsymbol{\rho}_{t-1}$

$\boldsymbol{\delta}_t$ = min(MIN, $\boldsymbol{\delta}_{t-1}$)

Instead of taking all data, we replace it with replicate for a small batch.

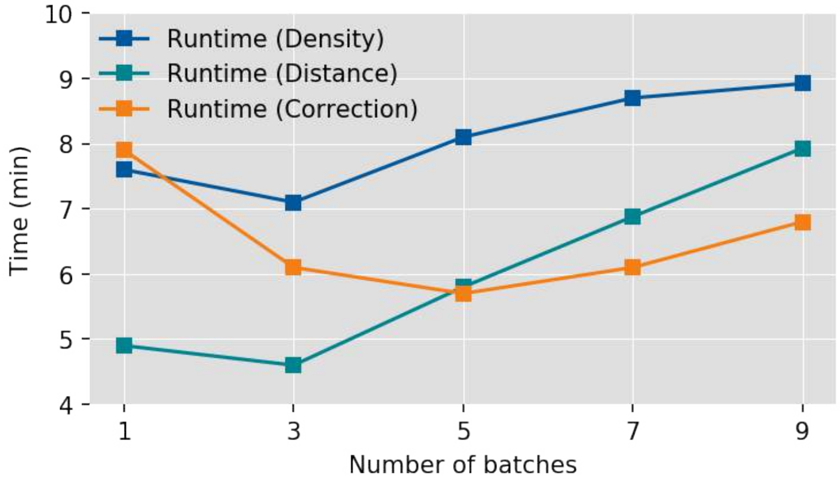
Tradeoff between Number of Batches and Shuffles

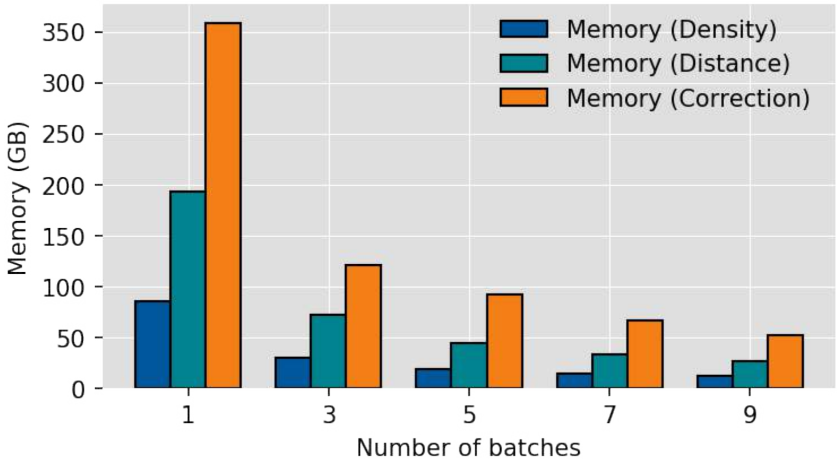
The greater number of batches, the more shuffle stages are needed.

The greater number of batches, the fewer chances to have long bottlenecks and large shuffle data.

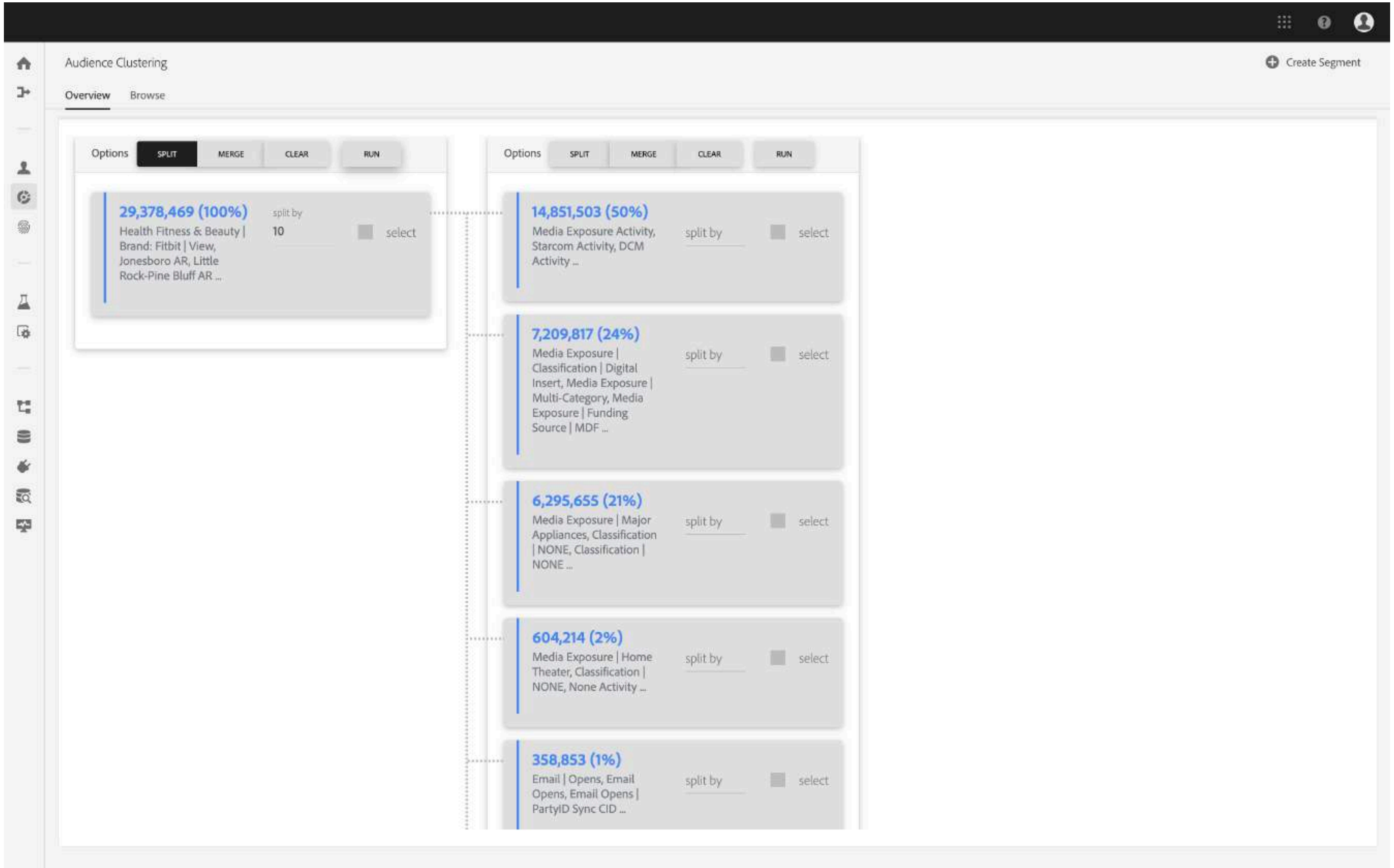
∴ While increasing number of batches can reduce the memory needed, there is an equilibrium in runtime.

The equilibrium depends on hardware specifications.

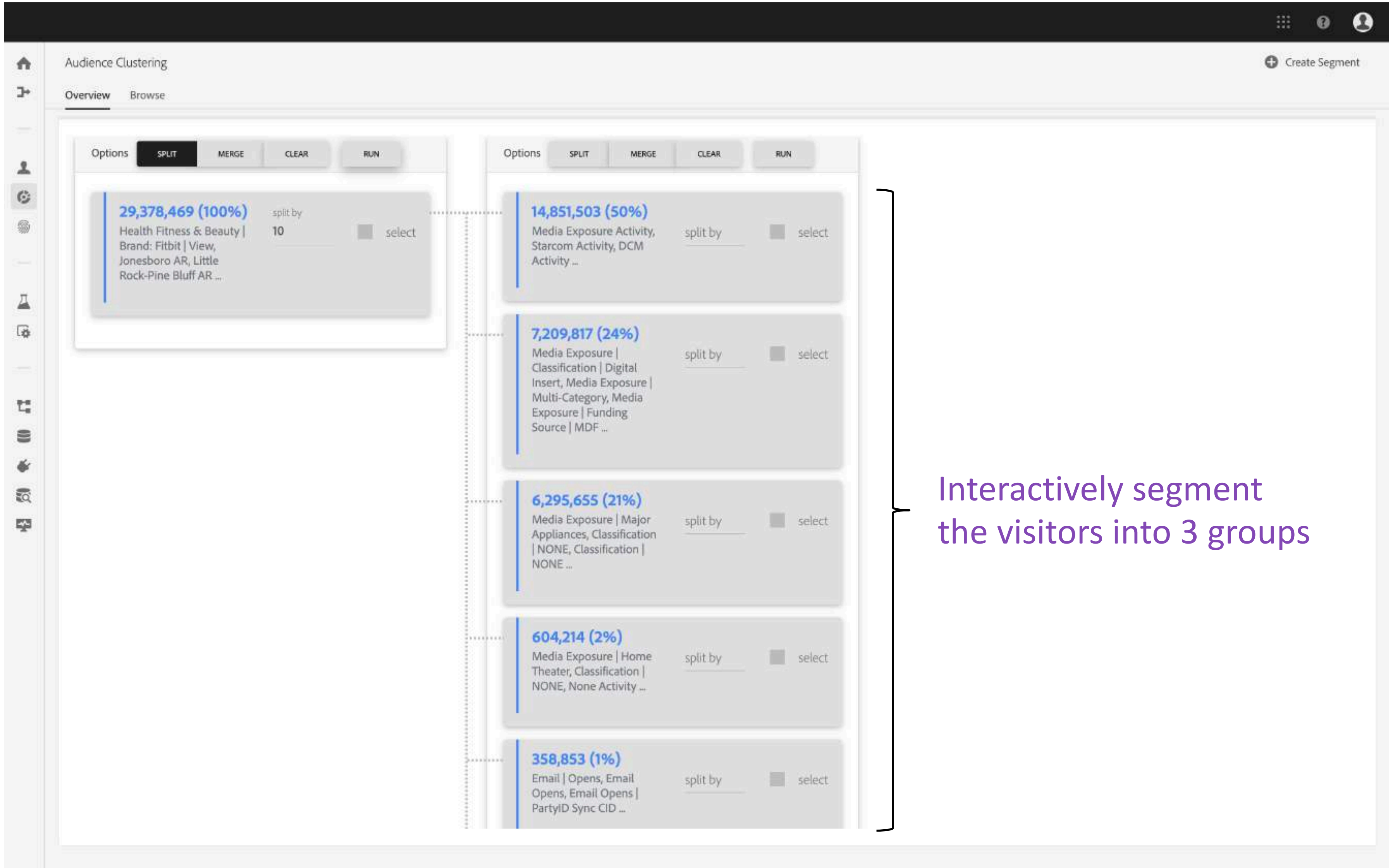




User Interface for Interactive Visitor Clustering

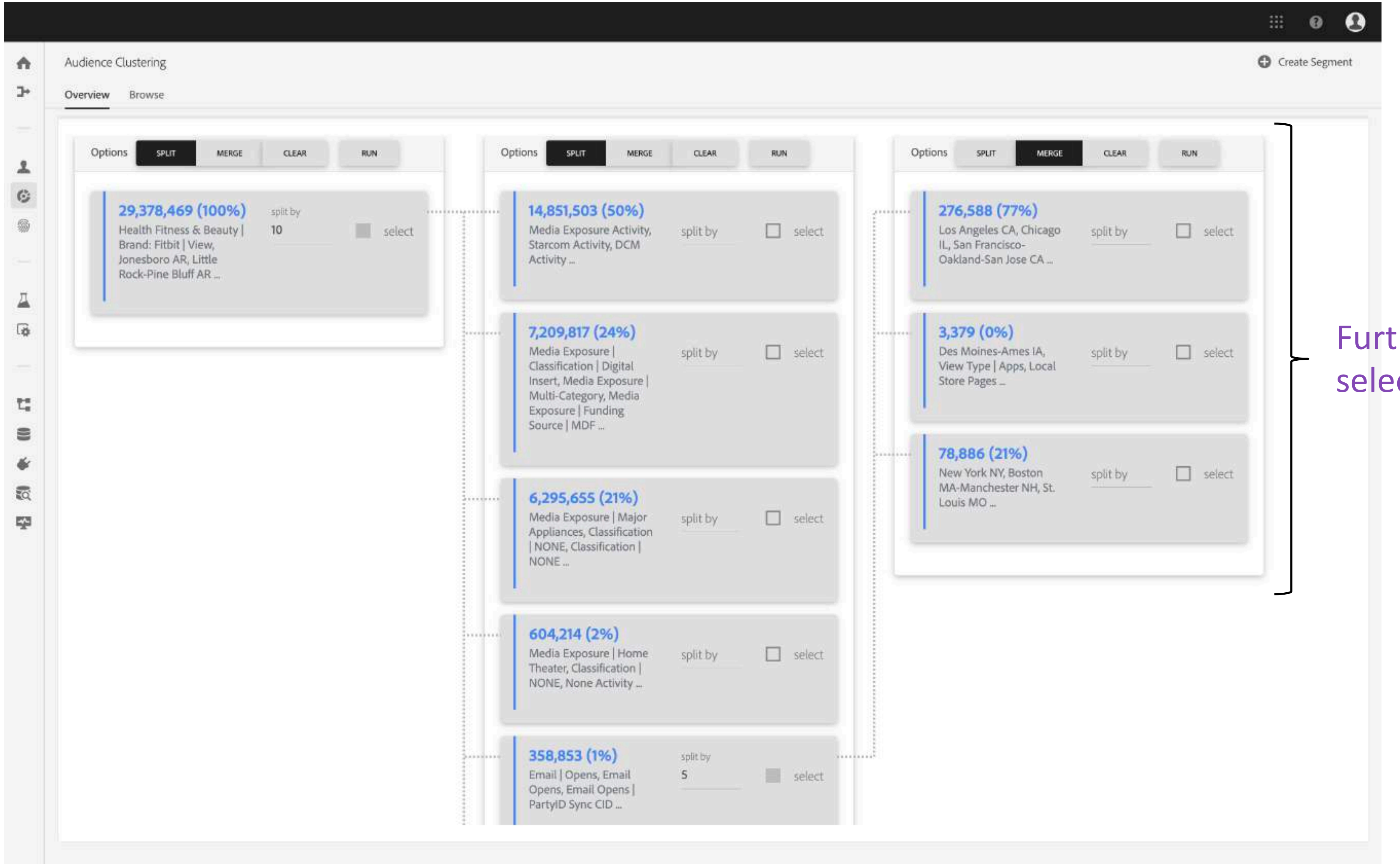


User Interface for Interactive Visitor Clustering



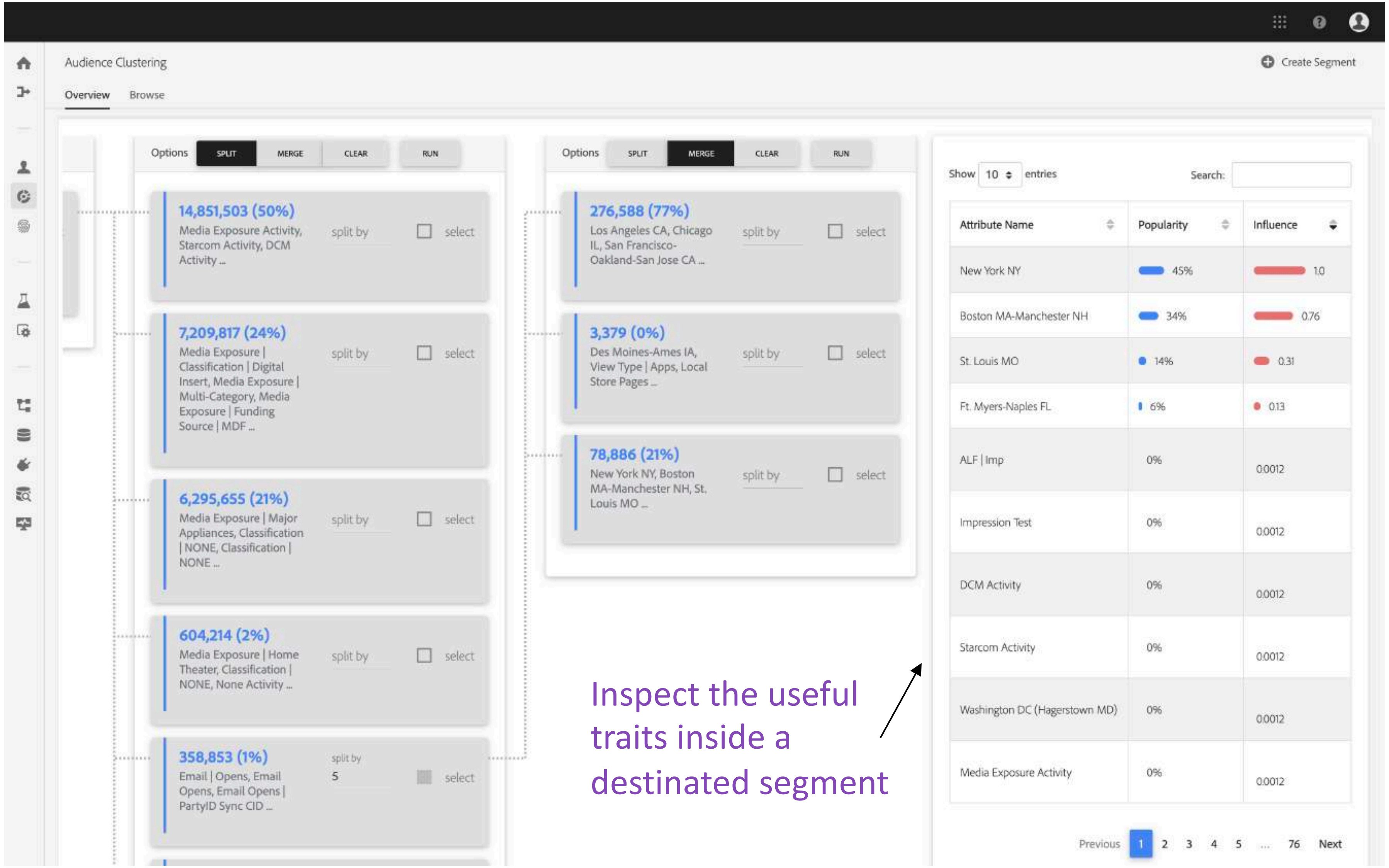
Interactively segment the visitors into 3 groups

User Interface for Interactive Visitor Clustering



Further split a selected segment

User Interface for Interactive Visitor Clustering



Inspect the useful traits inside a destinated segment

Conclusions and Future Work

- Applying LSH and DP clustering to enable interactive clustering on sparse online visitor data.
- Designing speed up strategies for clustering pipeline in a distributed environment.

- Future Work:
  - Further reduce the uneven data distribution in LSH Join
  - The increasing number of hash tables in LSH worsen the data distribution easily.

Thank You

Twitter: @GromitC
Email: gromit.chan@nyu.edu

Gromit Yeuk-Yin Chan, Fan Du, Ryan A. Rossi, Anup B. Rao, Eunyee Koh, Cláudio T. Silva, and Juliana Freire. 2020. Real-Time Clustering for Large Sparse Online Visitor Data. In Proceedings of The Web Conference 2020 (WWW ’20). Association for Computing Machinery, New York, NY, USA, 1049–1059.