OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

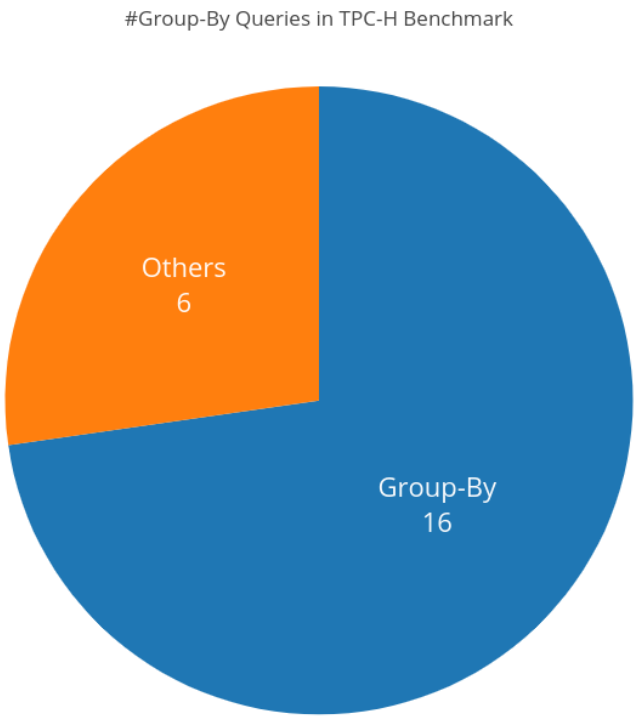
INF FAKULTÄT FÜR INFORMATIK

SIMD Vectorized Hashing for Grouped Aggregation

Bala Gurumurthy, David Broneske, Marcus Pinnecke, Gabriel Campero Durand and Gunter Saake

Grouped Aggregation

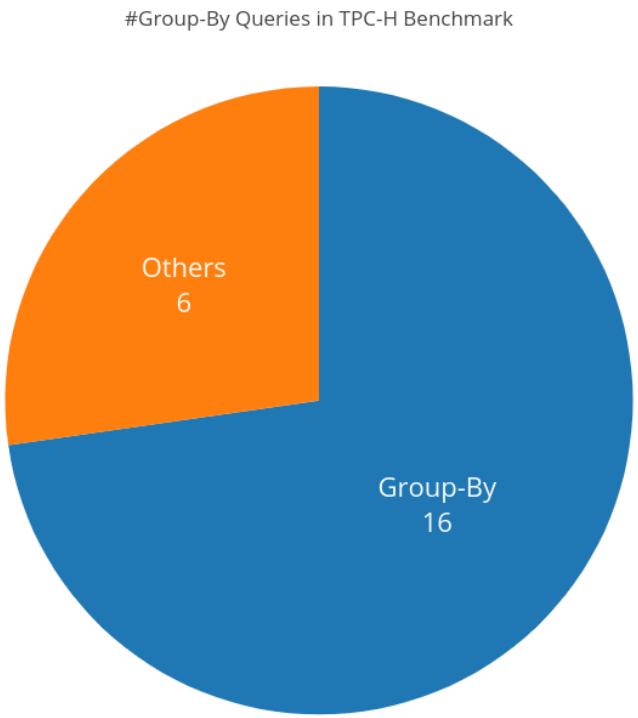
- Commonly-used and time-consuming operation



Based on analysis by Boncz et al.[1]

Grouped Aggregation

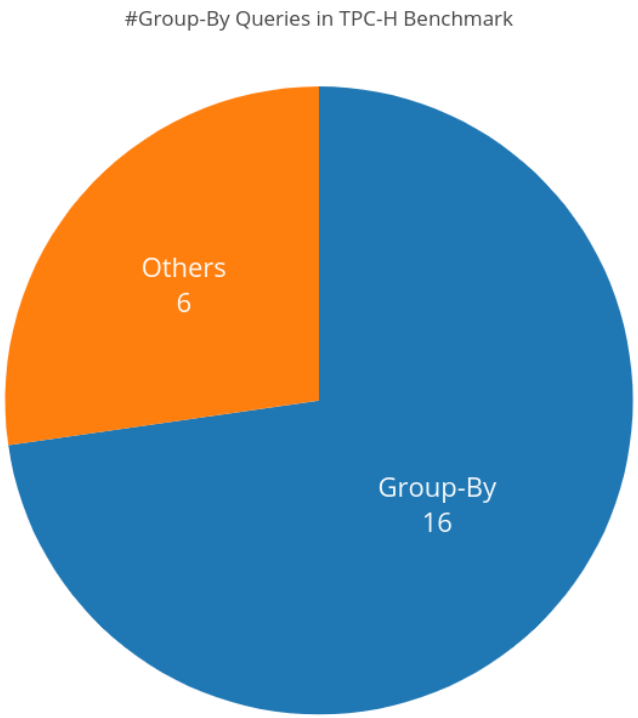
- Commonly-used and time-consuming operation
- All input must be consumed for single output



Based on analysis by Boncz et al.[1]

Grouped Aggregation

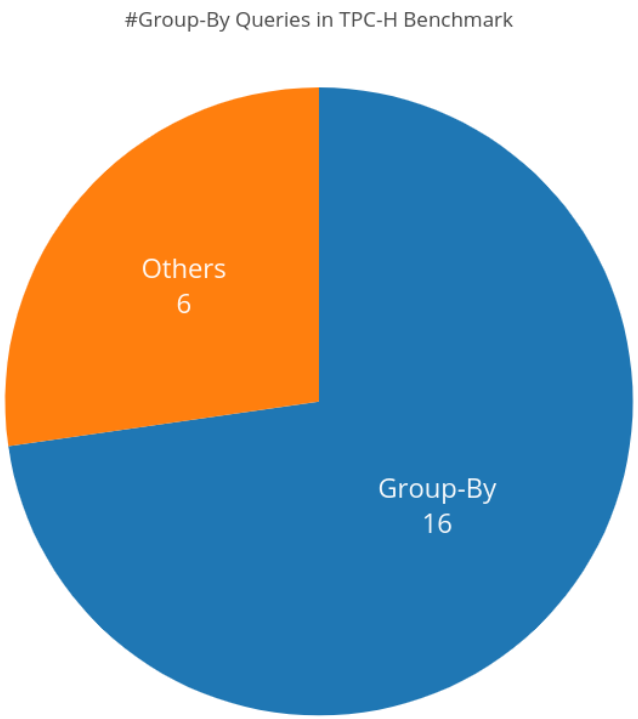
- Commonly-used and time-consuming operation
- All input must be consumed for single output
- Faster input processing = higher throughput



Based on analysis by Boncz et al.[1]

Grouped Aggregation

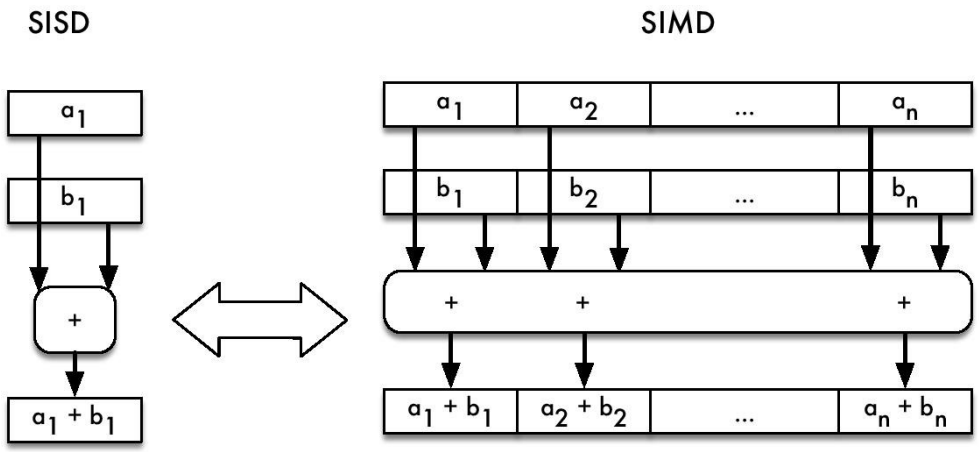
- Commonly-used and time-consuming operation
- All input must be consumed for single output
- Faster input processing = higher throughput
- Improving underlying technique improves efficiency

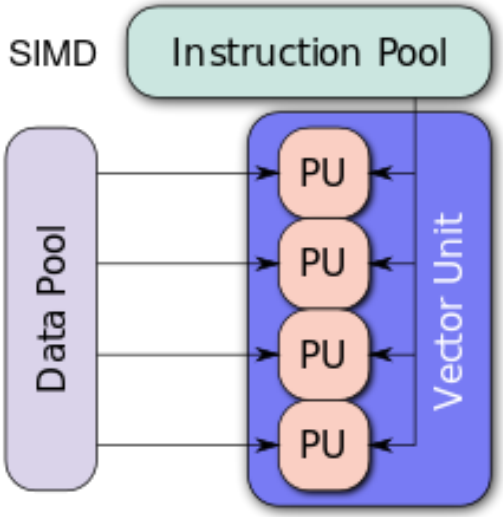


Based on analysis by Boncz et al.[1]

SIMD Capability in Modern Processors

- SIMD – Single Instruction Multiple Data



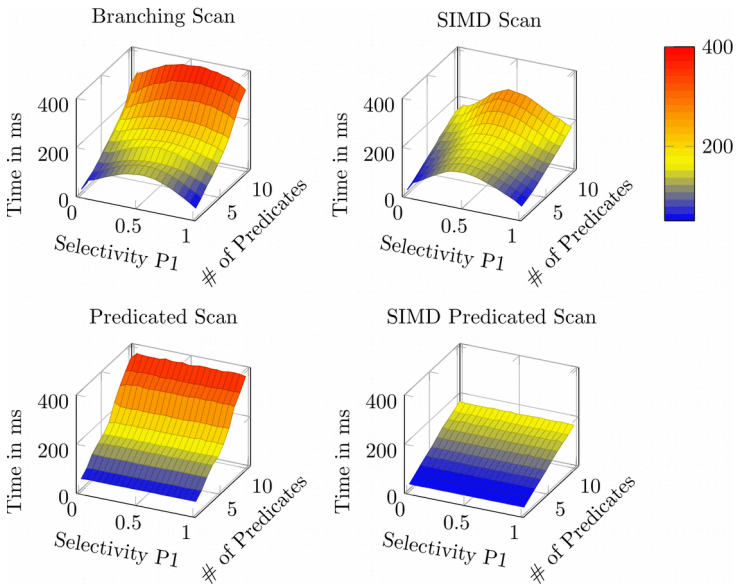


SIMD Capability in Modern Processors

- SIMD – Single Instruction Multiple Data
- Allows vectorized execution in modern processors
  - Reduces overall execution time of an operation

SIMD Capability in Modern Processors

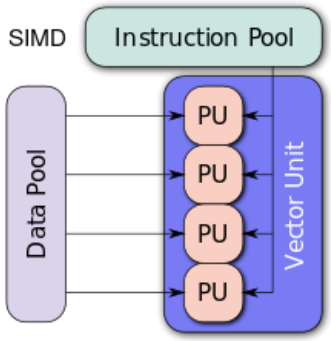
- SIMD – Single Instruction Multiple Data
- Allows vectorized execution in modern processors
  - Reduces overall execution time of an operation
- SIMD is shown to increase throughput in orders of magnitude for DBMS operation [2] [3]



SIMD accelerated selection [3]

SIMD for Grouped Aggregation

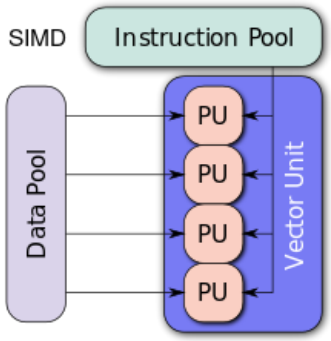
- SIMD acceleration of hashing techniques improves throughput



+ **Group-By** = High throughput

SIMD for Grouped Aggregation

- SIMD acceleration of hashing techniques improves throughput



+ **Group-By** = High throughput

How to incorporate SIMD for Grouped Aggregation?

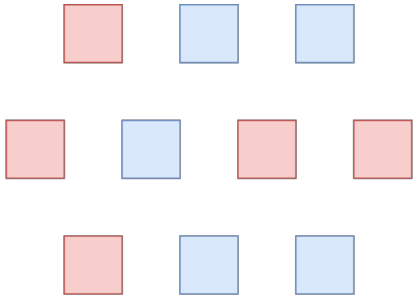
What is the impact of SIMD?

Hash Based Aggregation

- Grouped aggregation commonly implemented using hashing techniques

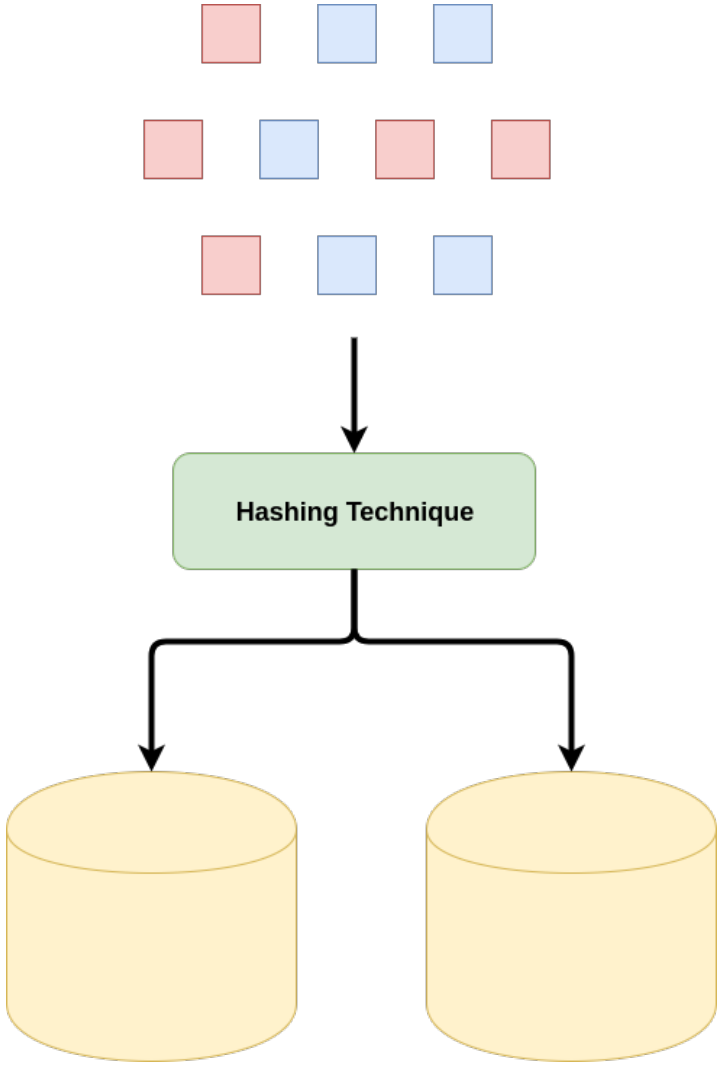
Hash Based Aggregation

- Grouped aggregation commonly implemented using hashing techniques



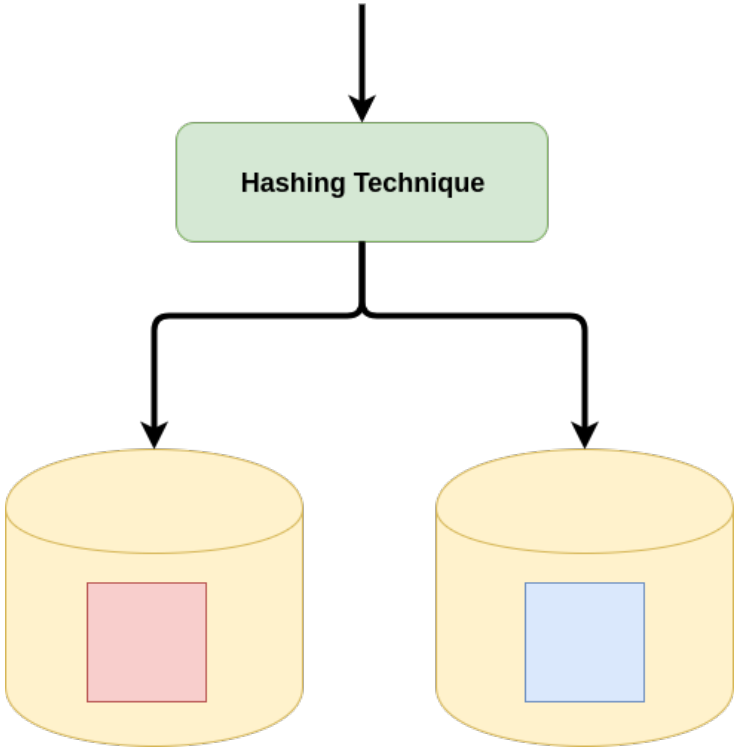
Hash Based Aggregation

- Grouped aggregation commonly implemented using hashing techniques
- Separates groups into buckets

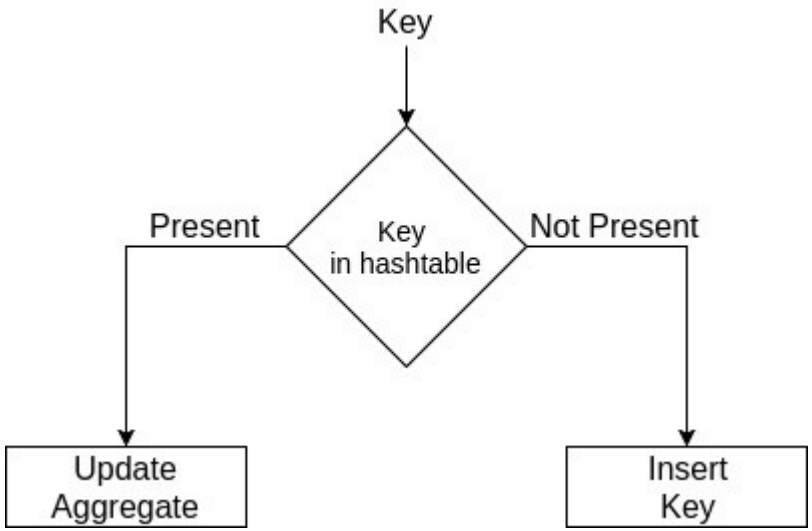


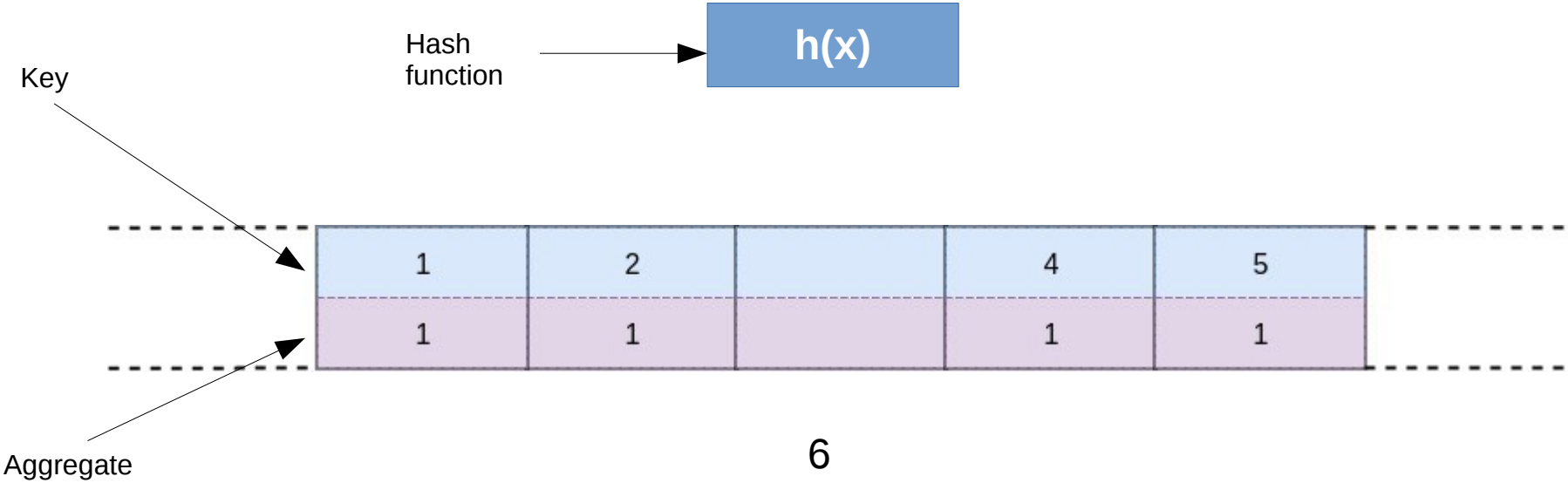
Hash Based Aggregation

- Grouped aggregation commonly implemented using hashing techniques
- Separates groups into buckets
- Aggregation done within each buckets

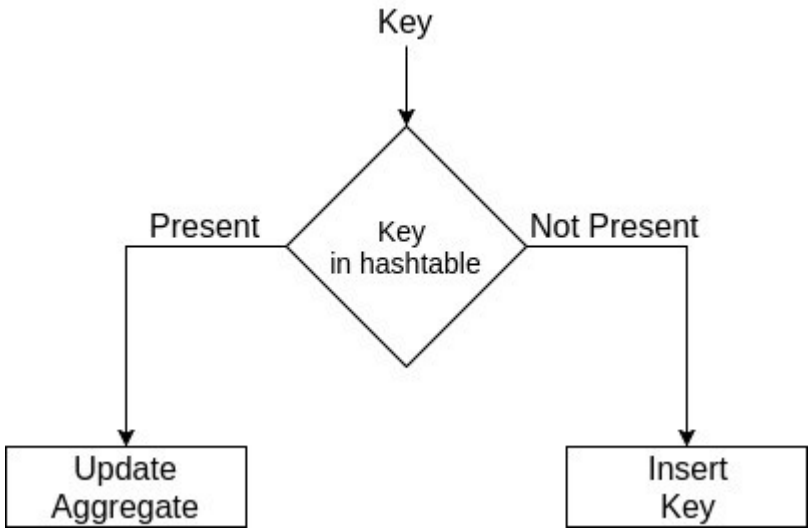


Hash Based Aggregation: Example

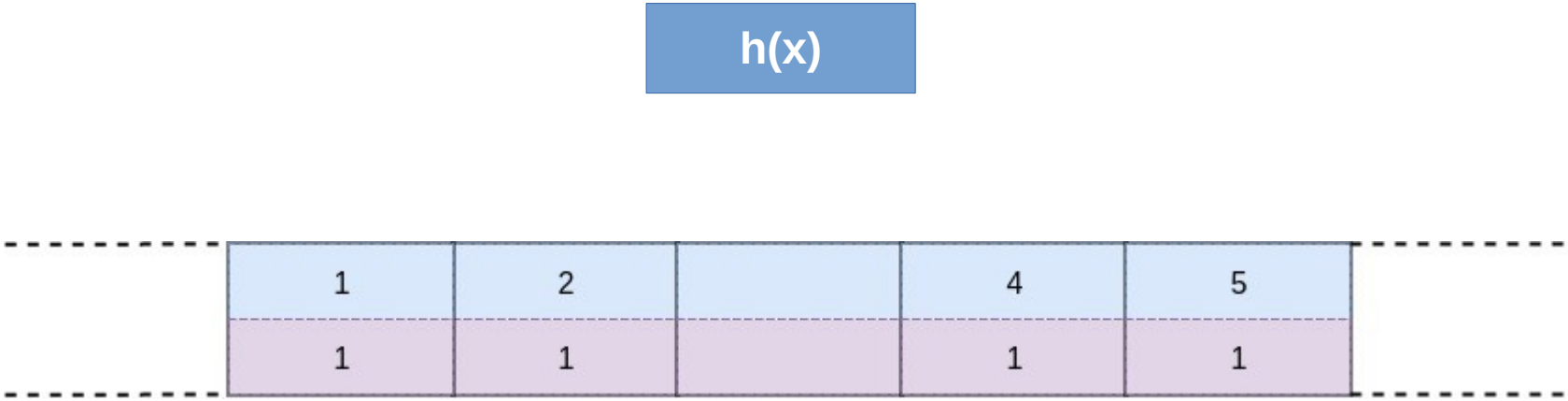




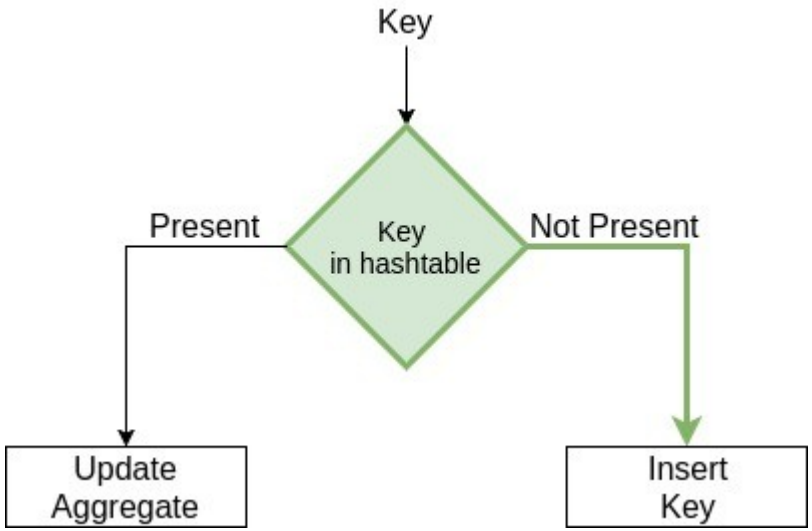
Hash Based Aggregation: Example



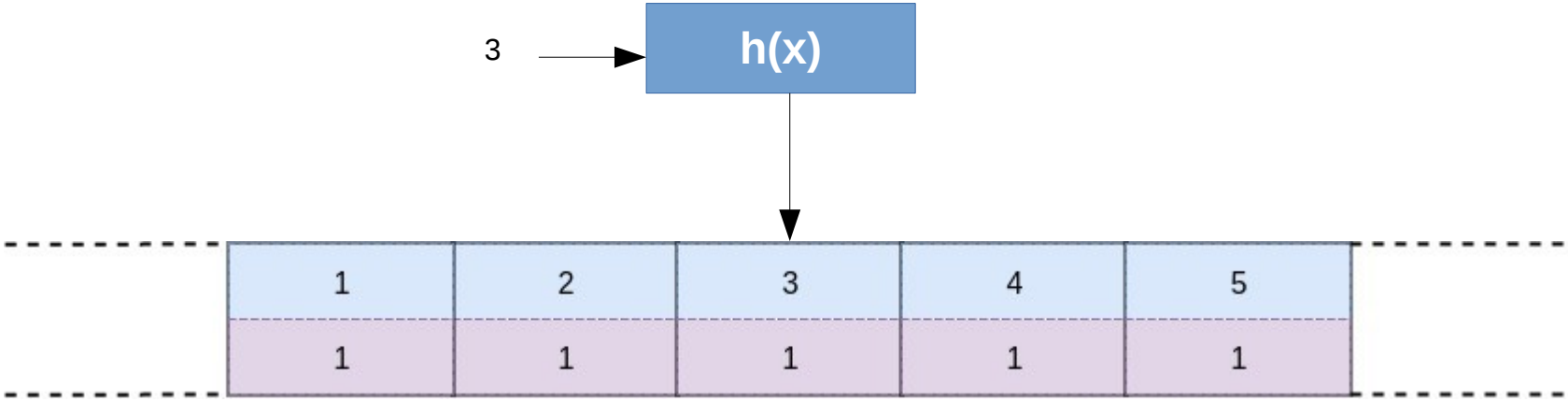
Input: 3



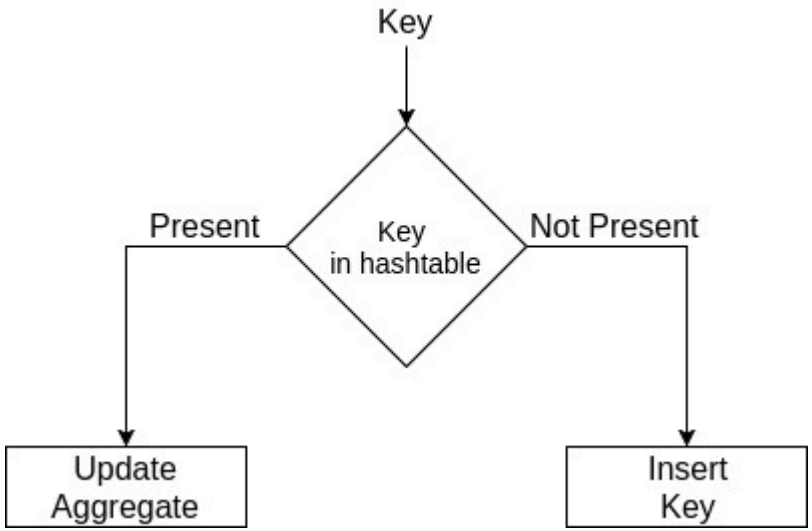
Hash Based Aggregation: Example



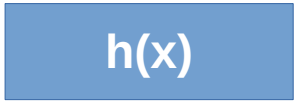
Input: 3



Hash Based Aggregation: Example

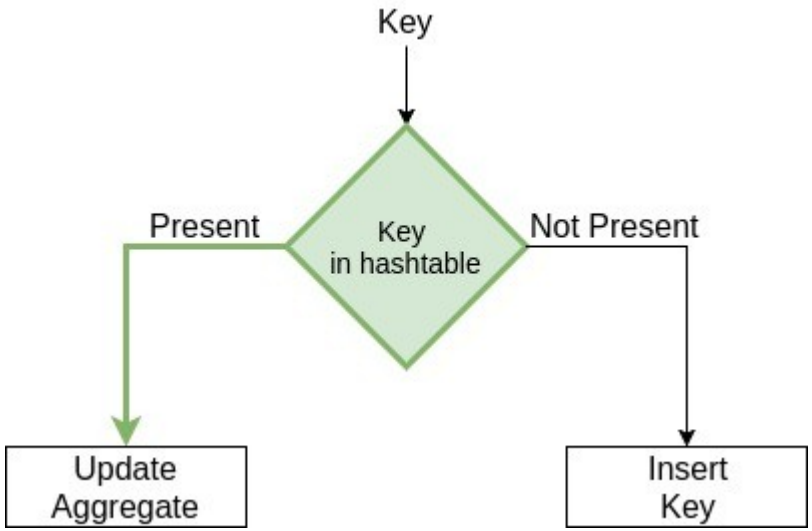


Input: 3

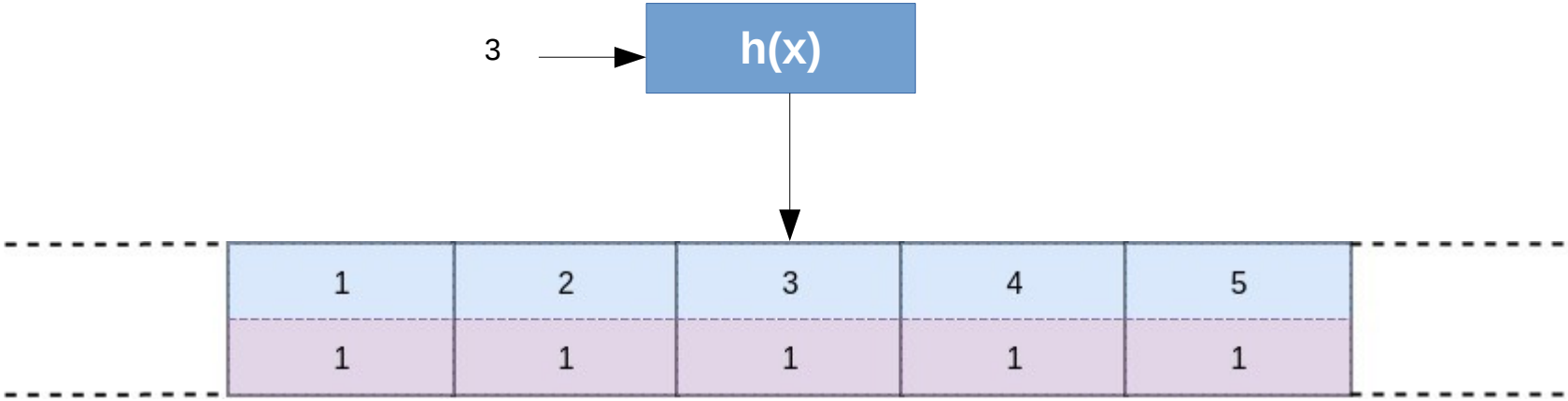


1	2	3	4	5
1	1	1	1	1

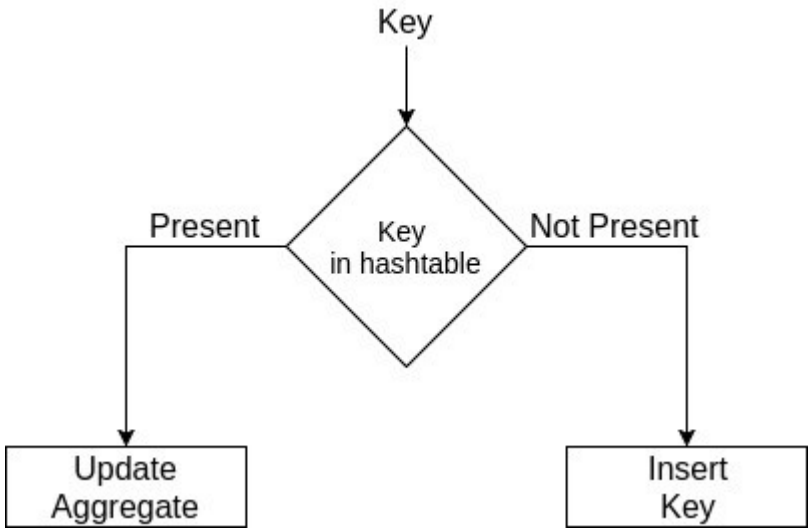
Hash Based Aggregation: Example



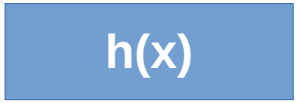
Input: 3



Hash Based Aggregation: Example



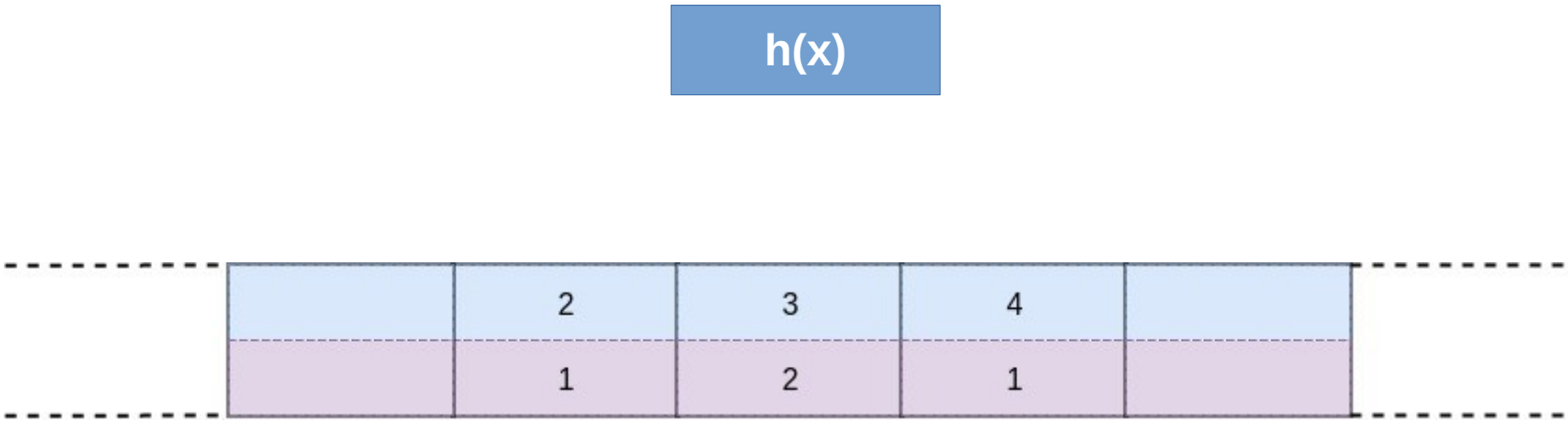
Input: 3



1	2	3	4	5
1	1	2	1	1

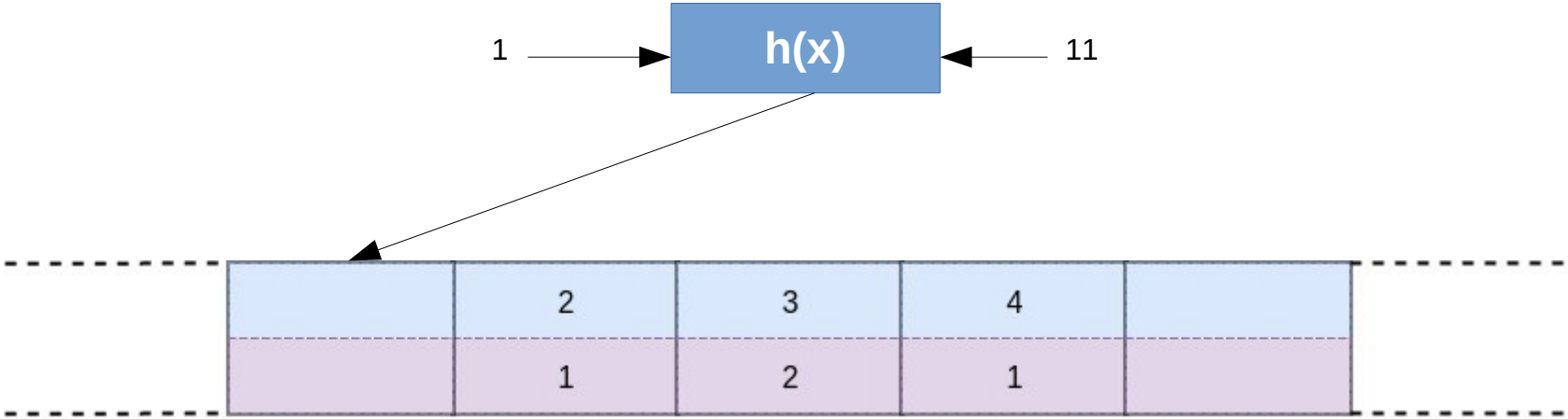
Collision: Increasing Complexity

- Not all keys have unique location



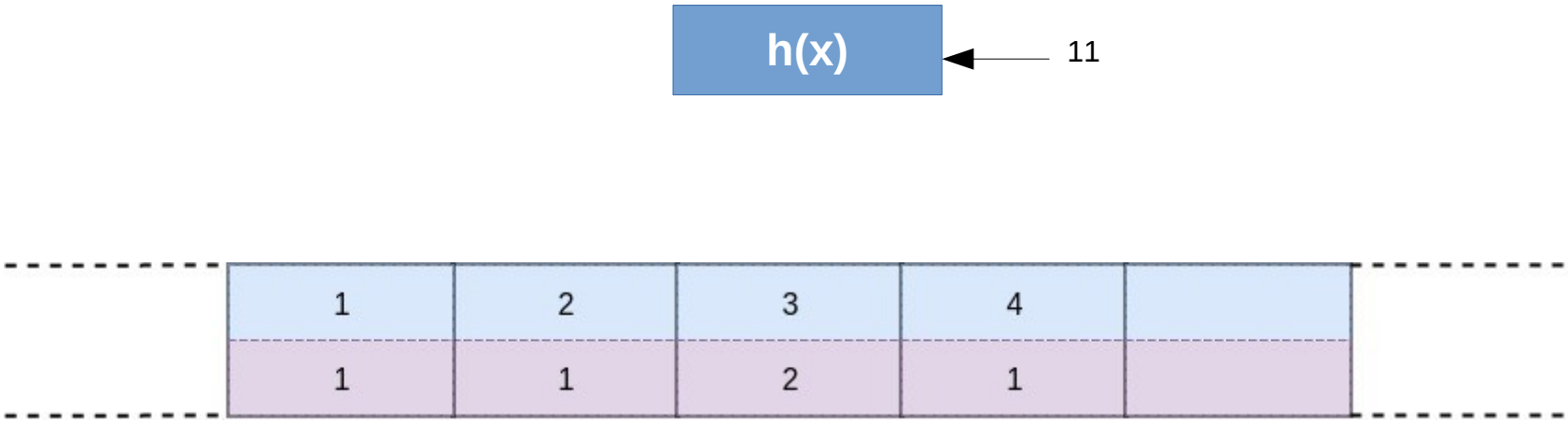
Collision: Increasing Complexity

- Not all keys have unique location
- Two keys might hash to same slots



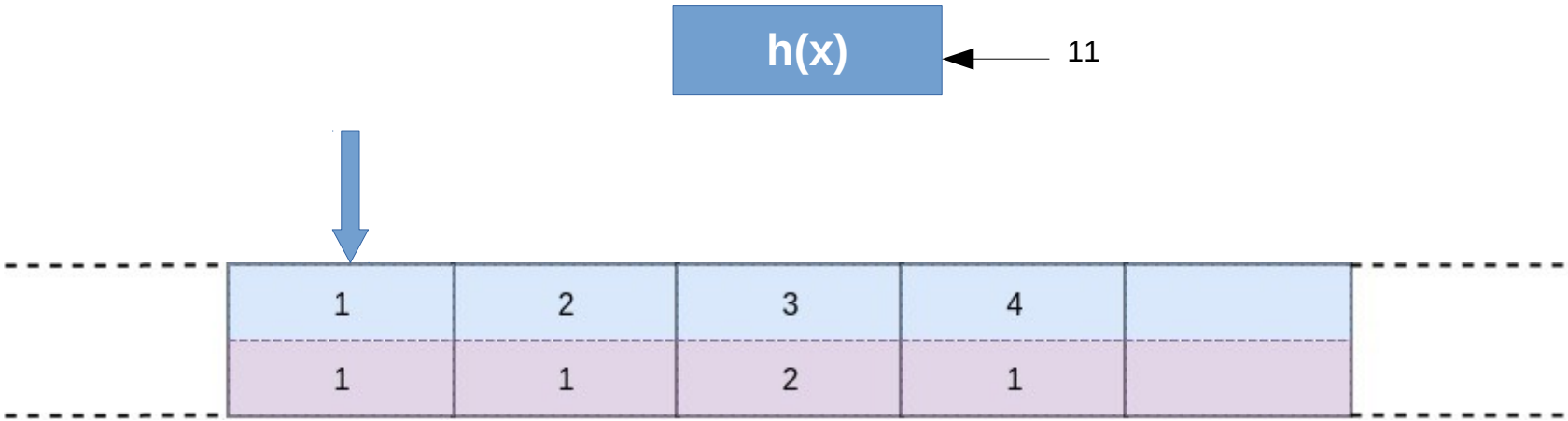
Collision: Increasing Complexity

- Not all keys have unique location
- Two keys might hash to same slots



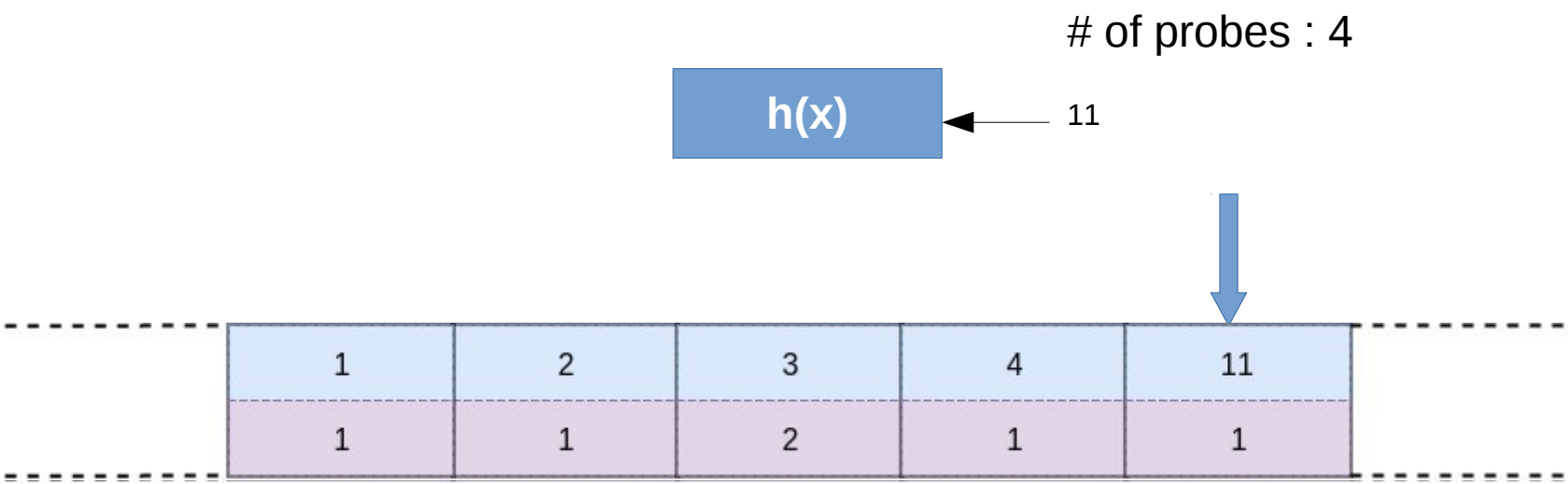
Collision: Increasing Complexity

- Not all keys have unique location
- Two keys might hash to same slots
- Hash table must be probed for alternative location



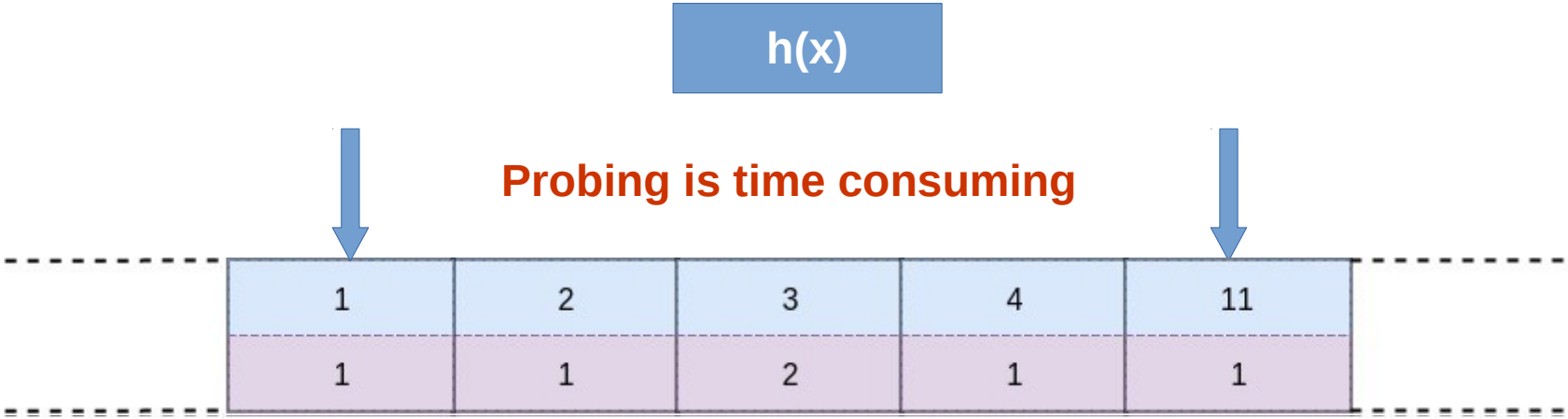
Collision: Increasing Complexity

- Not all keys have unique location
- Two keys might hash to same slots
- Hash table must be probed for alternative location



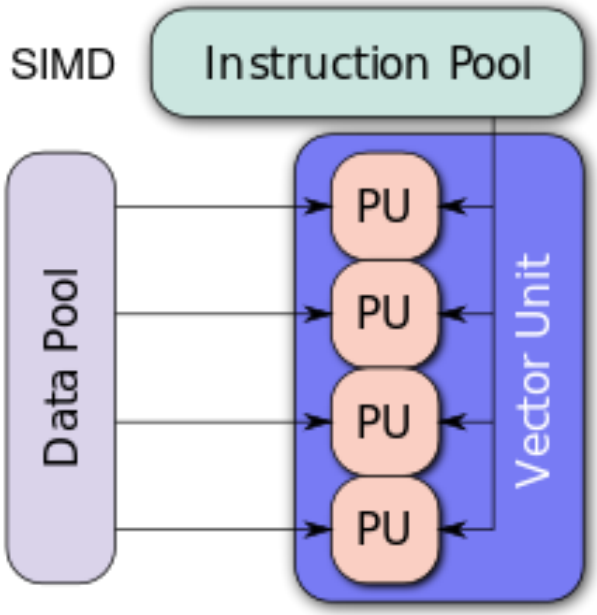
Collision: Increasing Complexity

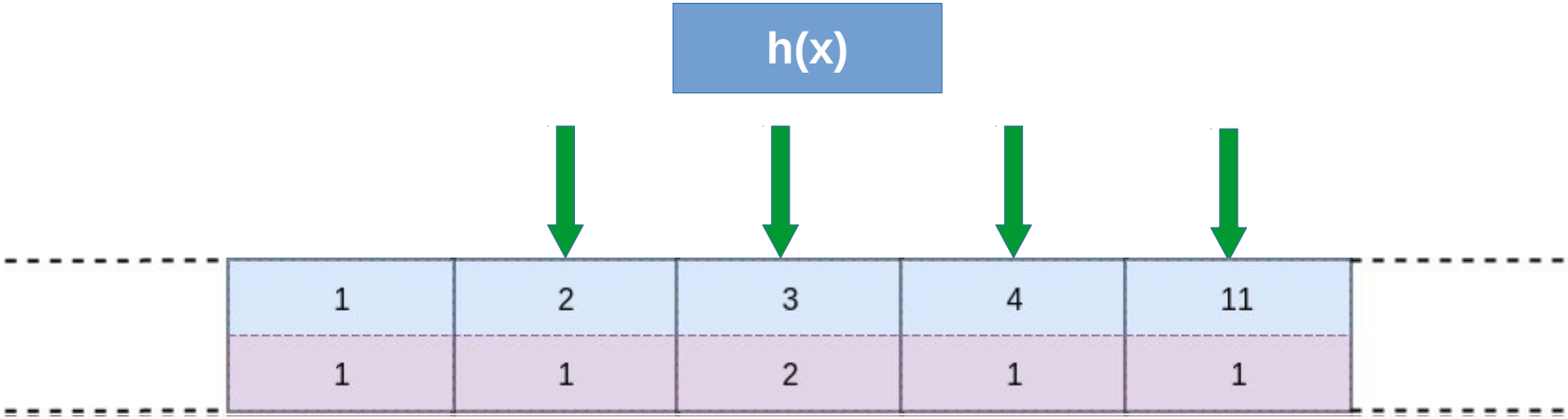
- Not all keys have unique location
- Two keys might hash to same slots
- Hash table must be probed for alternative location



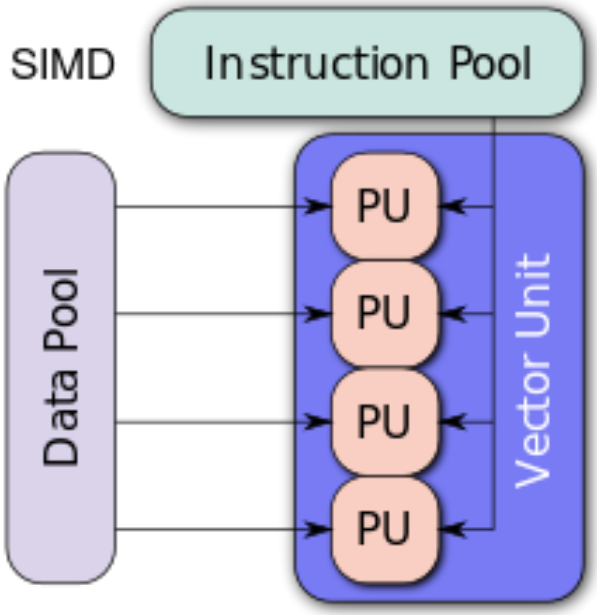
SIMD for Probing

- Multiple slots are probed in an instant using SIMD

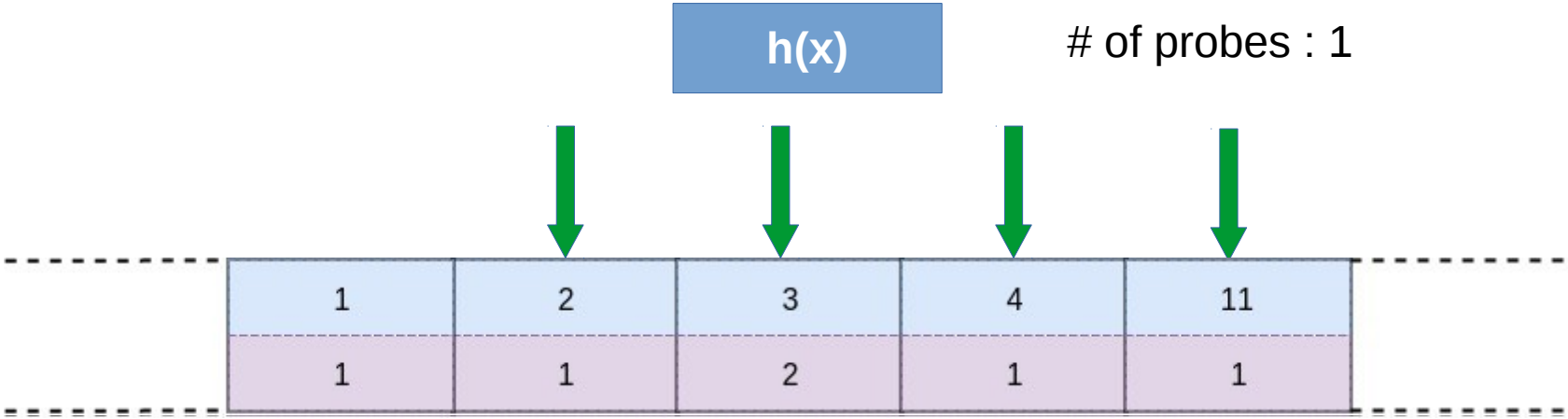




SIMD for Probing



- Multiple slots are probed in an instant using SIMD
- Reduces overall number of probes



SIMD Accelerated Hash Probing

- Each hashing techniques have their own collision resolution mechanism

SIMD Accelerated Hash Probing

- Each hashing techniques have their own collision resolution mechanism
- We use open-addressing hashing techniques

SIMD Accelerated Hash Probing

- Each hashing techniques have their own collision resolution mechanism
- We use open-addressing hashing techniques
    - Have constant hashtable size

SIMD Accelerated Hash Probing

- Each hashing techniques have their own collision resolution mechanism
- We use open-addressing hashing techniques
    - Have constant hashtable size
    - Suitable for SIMD

SIMD Accelerated Hash Probing

- Each hashing techniques have their own collision resolution mechanism
- We use open-addressing hashing techniques
  - Have constant hashtable size
  - Suitable for SIMD

Hashing techniques used are

- Cuckoo hashing
- Linear probing
- Two-choice hashing
- Hopscotch hashing

Cuckoo Hashing

- Stores keys in multiple hash tables

Hash(x) = x%5

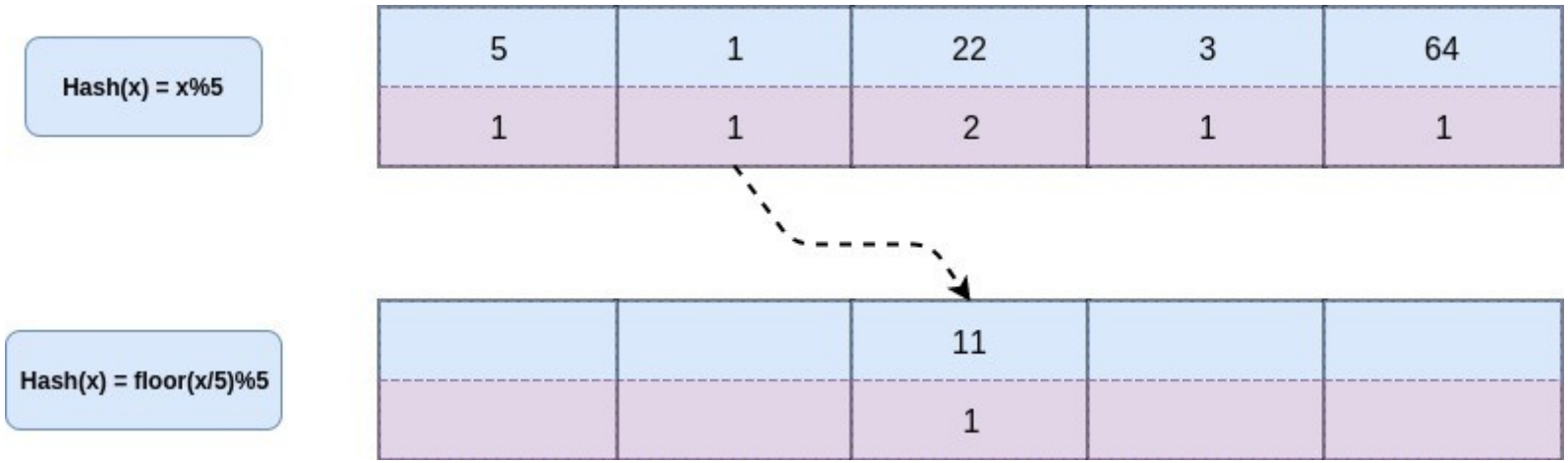
5	11	22	3	64
1	1	2	1	1

Hash(x) = floor(x/5)%5

Cuckoo Hashing

- Stores keys in multiple hash tables
- On collision swaps current value to alternative tables

Input : 1



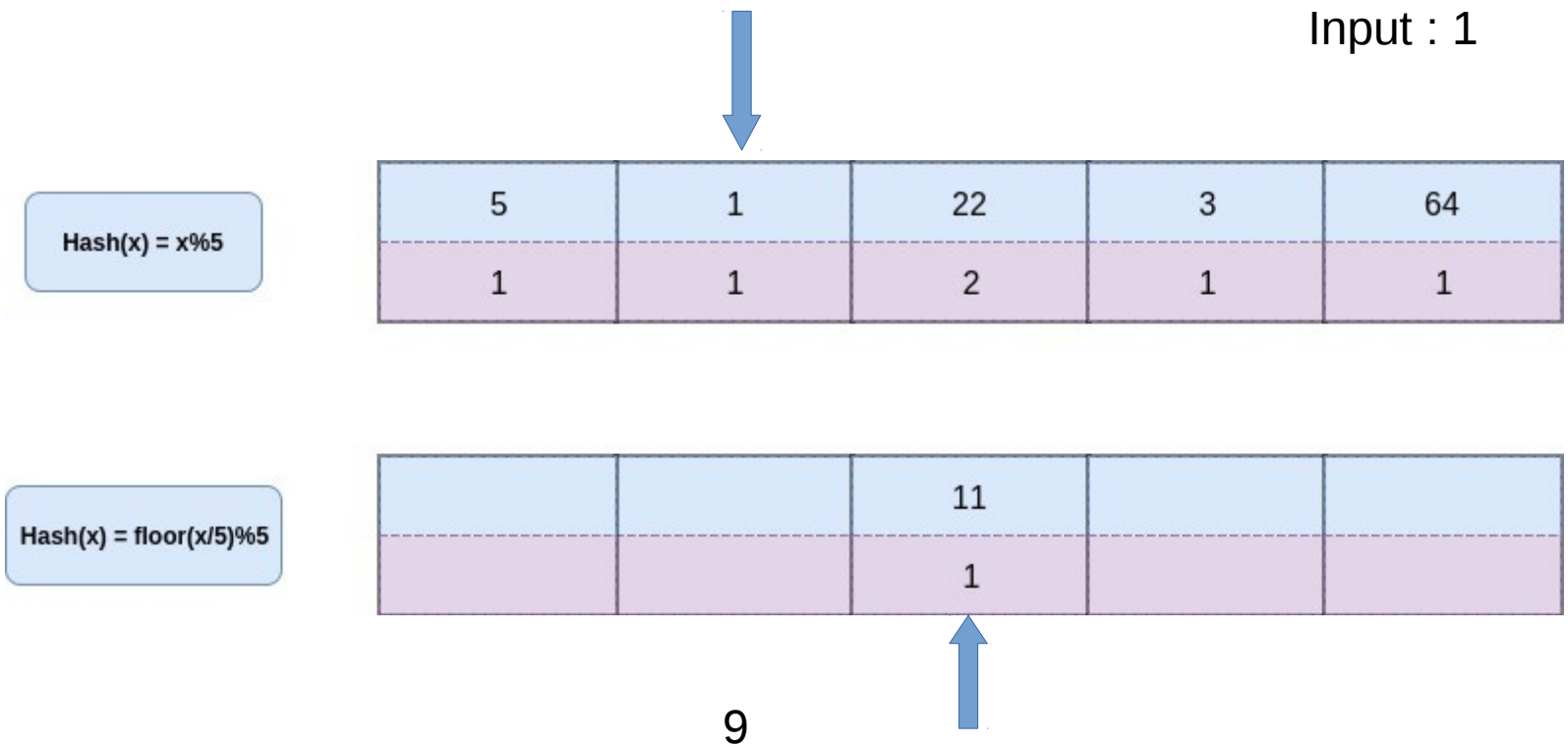
Cuckoo Hashing

- Stores keys in multiple hash tables
- On collision swaps current value to alternative tables
  - Might form swap loop; solved using a threshold

Input : 111

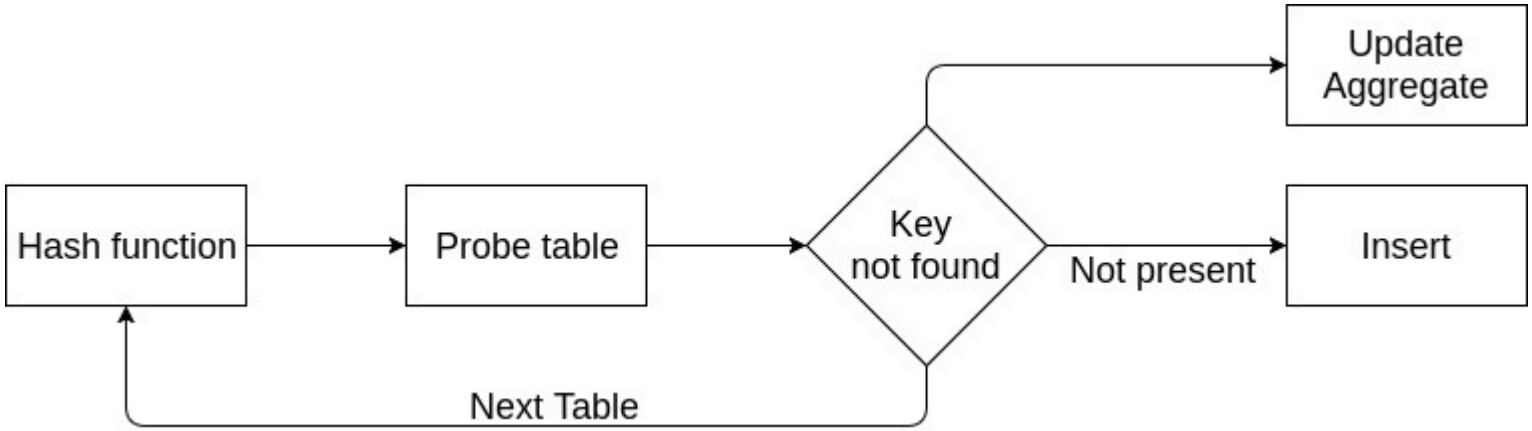
Cuckoo Hashing

- Stores keys in multiple hash tables
- On collision swaps current value to alternative tables
  - Might form swap loop; solved using a threshold
- Has constant look-up time



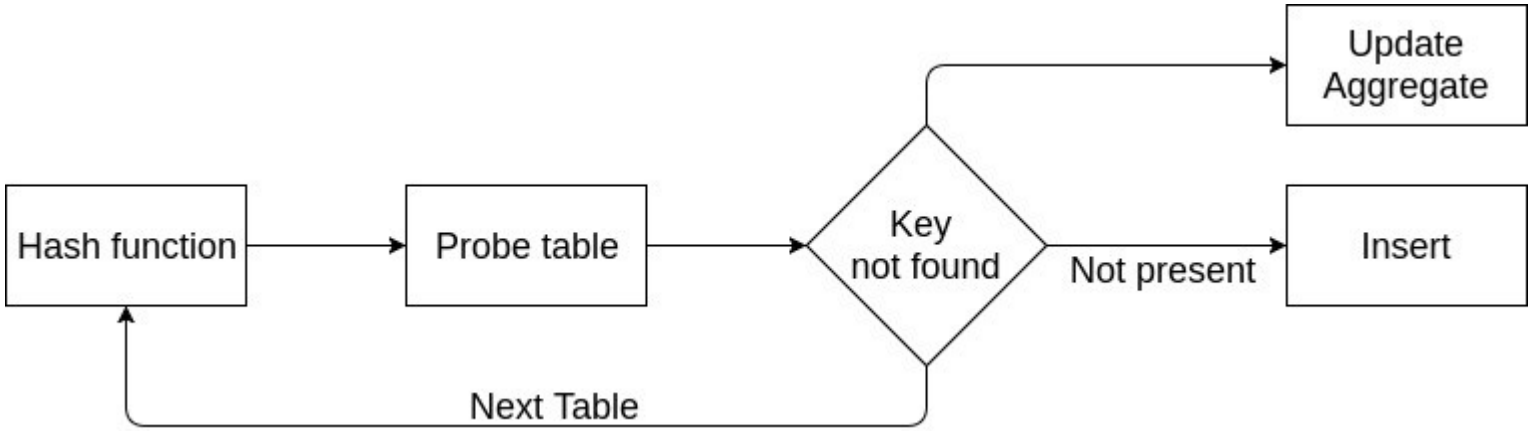
Grouped Aggregation using Cuckoo Hashing

- Probe for key in each hash table



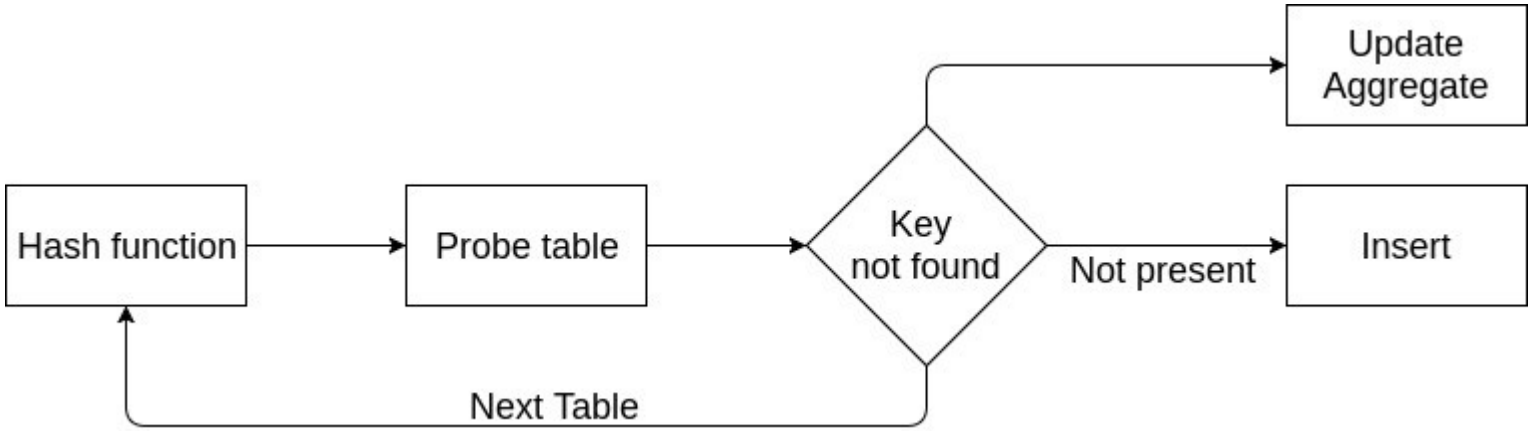
Grouped Aggregation using Cuckoo Hashing

- Probe for key in each hash table
  - If found update aggregate



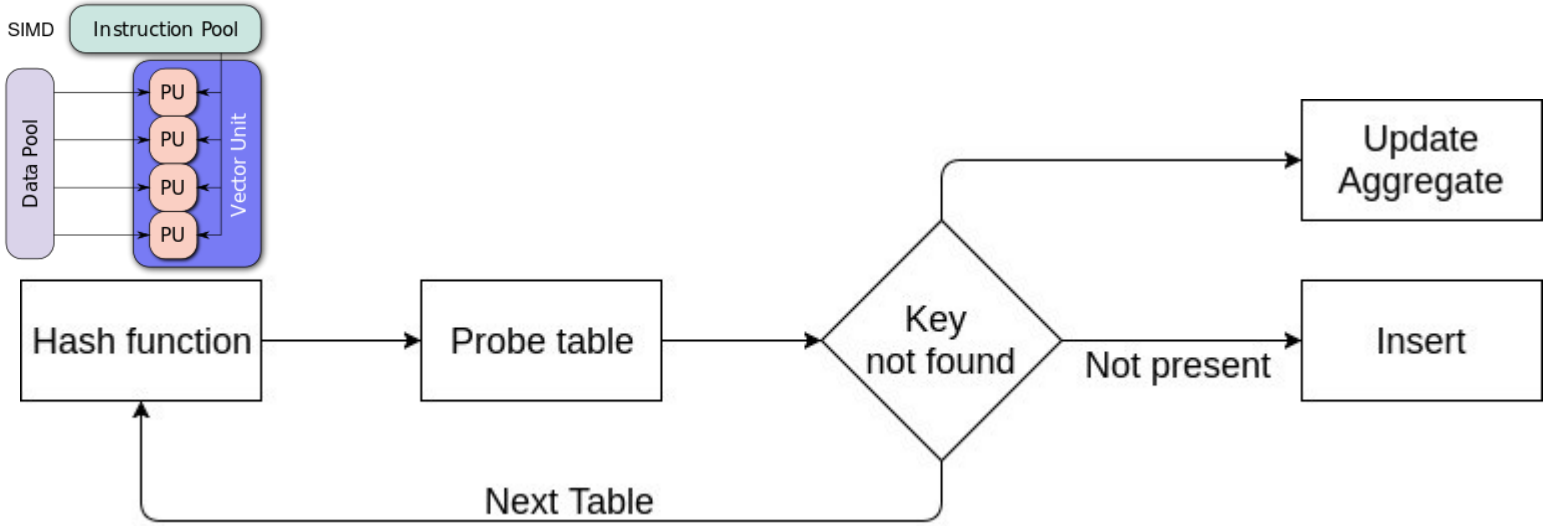
Grouped Aggregation using Cuckoo Hashing

- Probe for key in each hash table
  - If found update aggregate
  - Else, insert the key



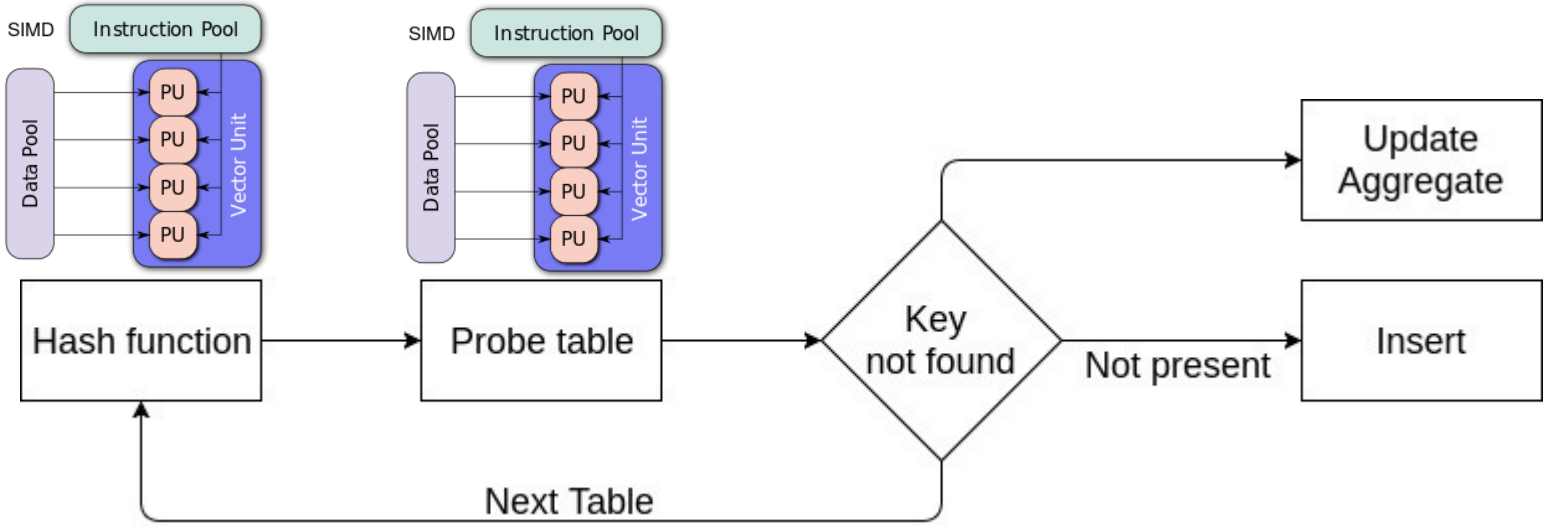
SIMD Optimization of Cuckoo Hashing

- Hash function computed for multiple tables



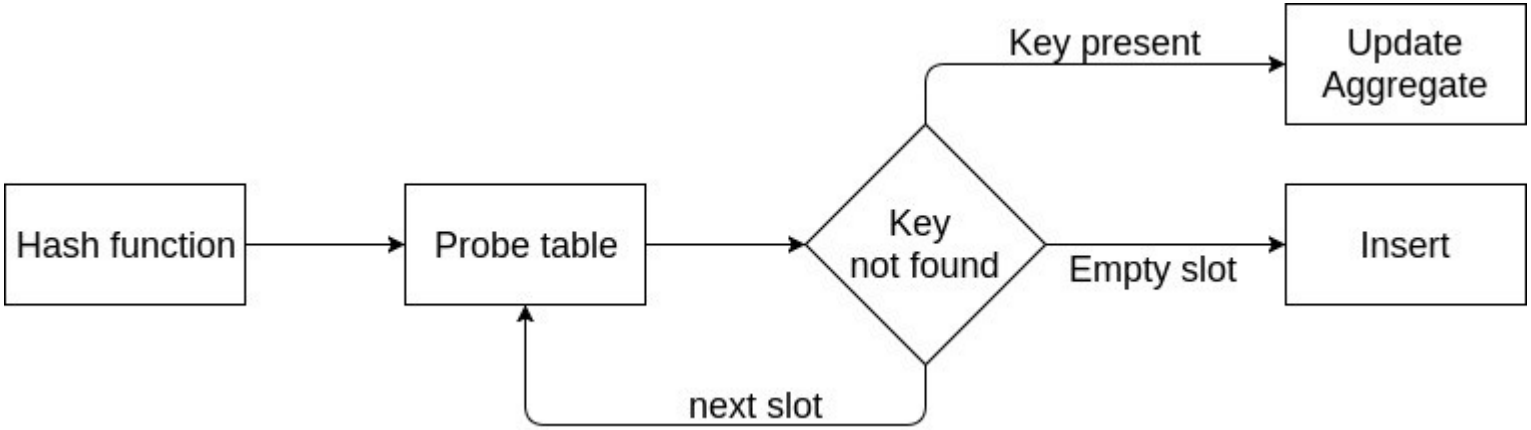
SIMD Optimization of Cuckoo Hashing

- Hash function computed for multiple tables
- Multiple slots probed in parallel



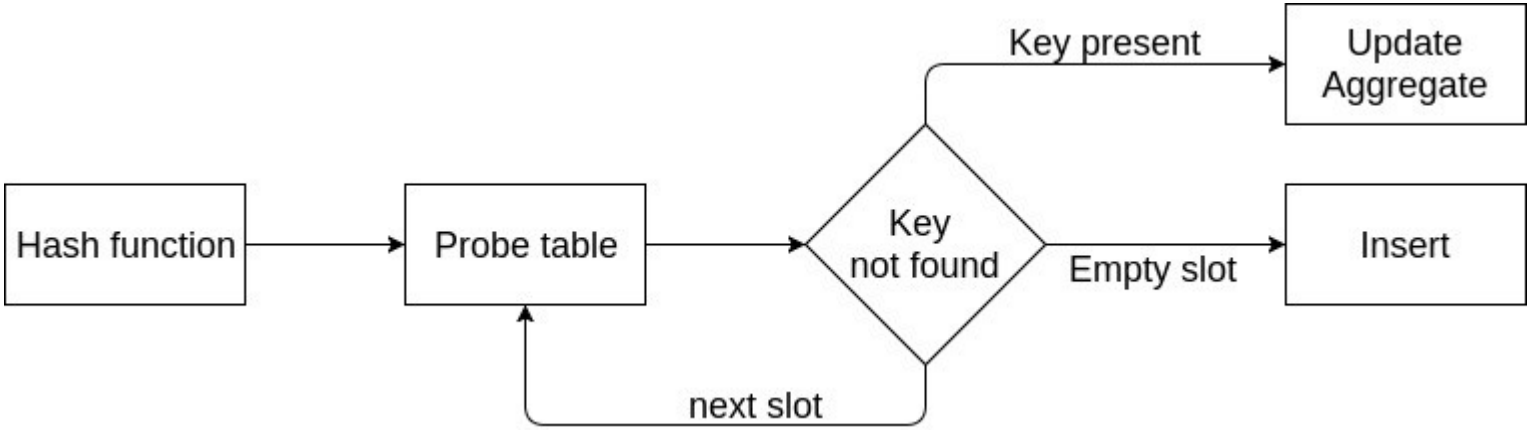
Linear Probing

- Straight forward approach for collision resolution



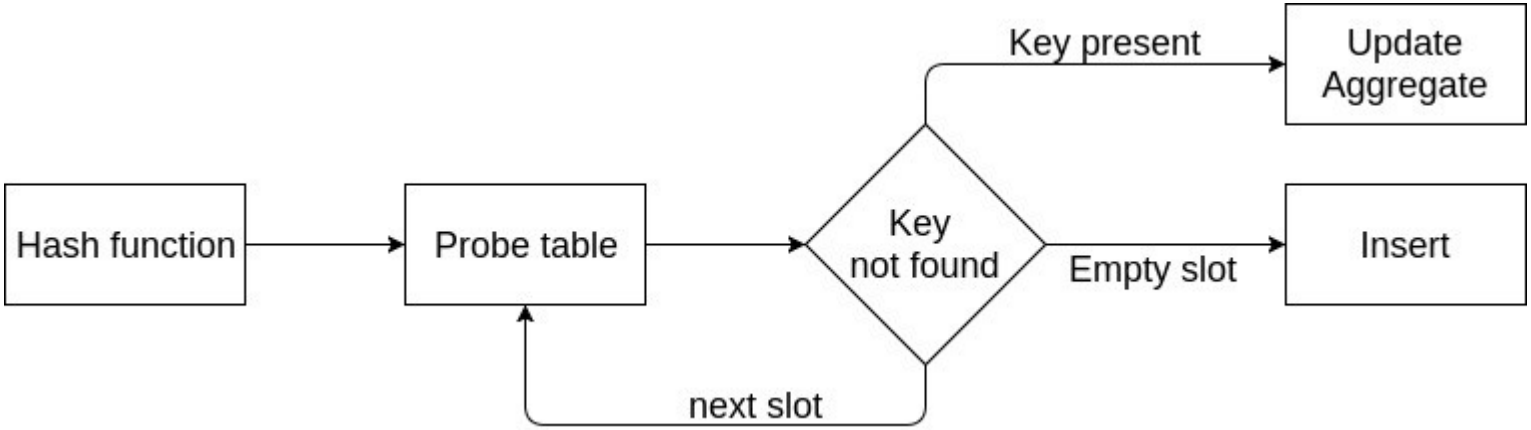
Linear Probing

- Straight forward approach for collision resolution
- Probes hash table linearly for alternative location



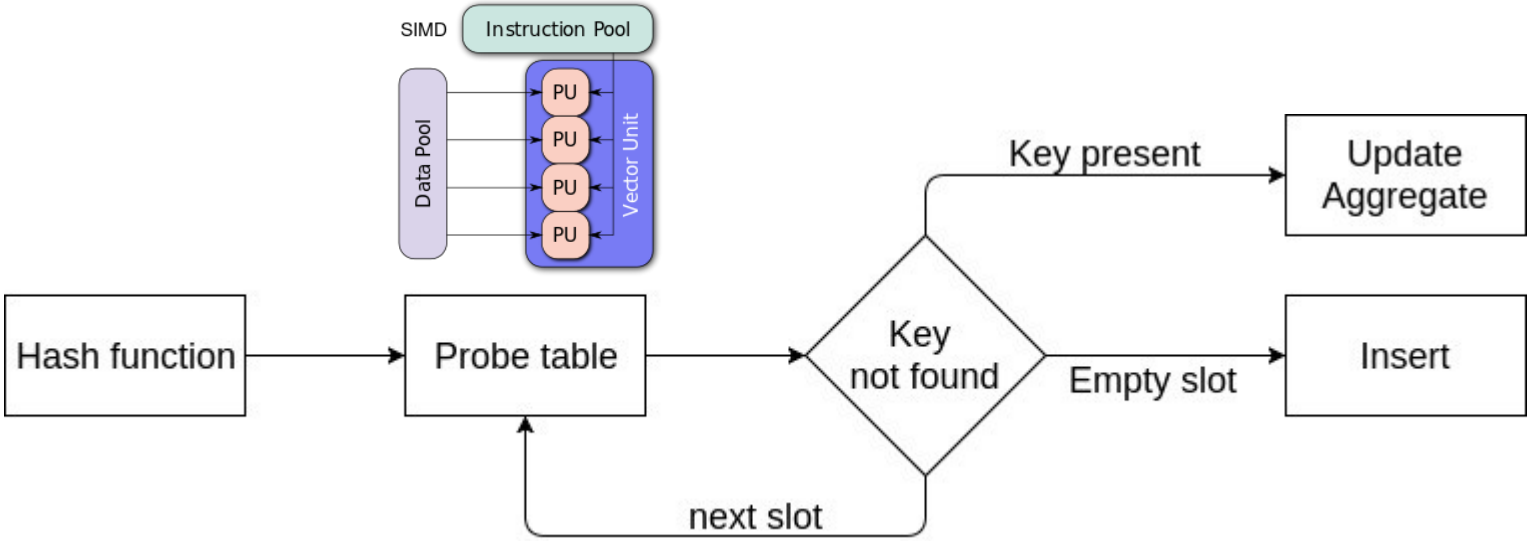
Linear Probing

- Straight forward approach for collision resolution
- Probes hash table linearly for alternative location
- Empty location encountered : insert key



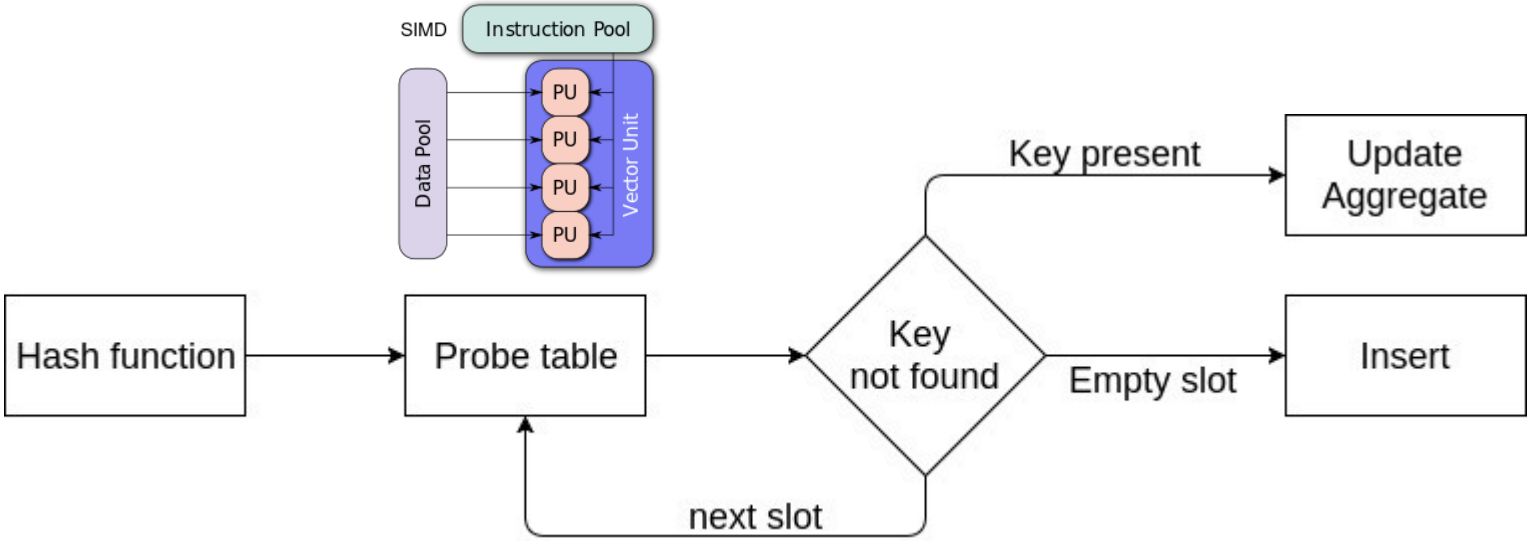
SIMD Optimization of Linear Probing

- Multiple slot is probed



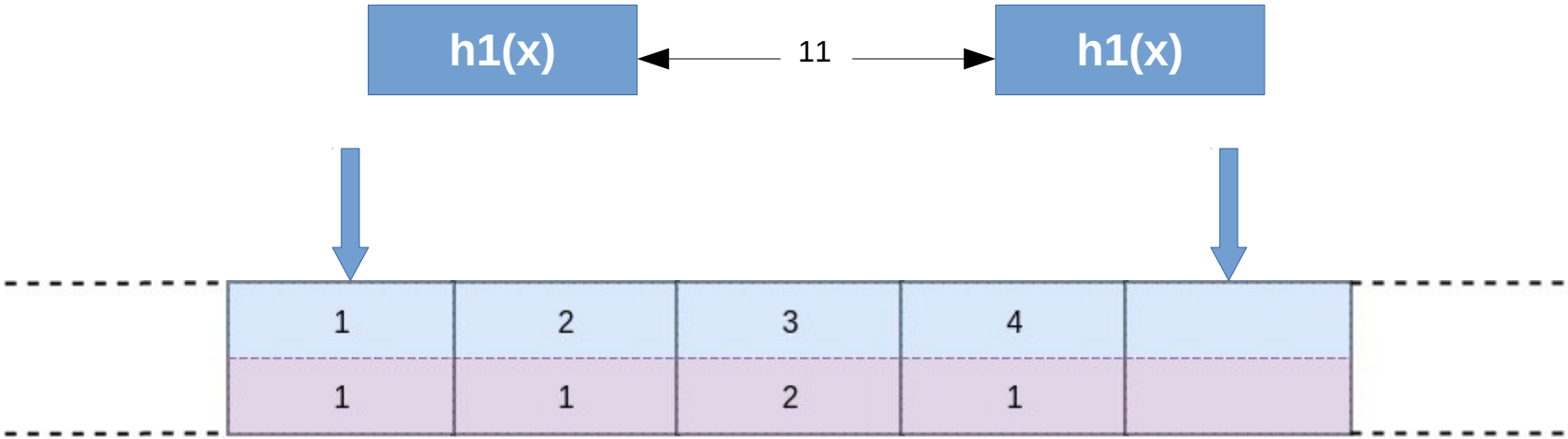
SIMD Optimization of Linear Probing

- Multiple slot is probed
- Comparison mask used for updating aggregate



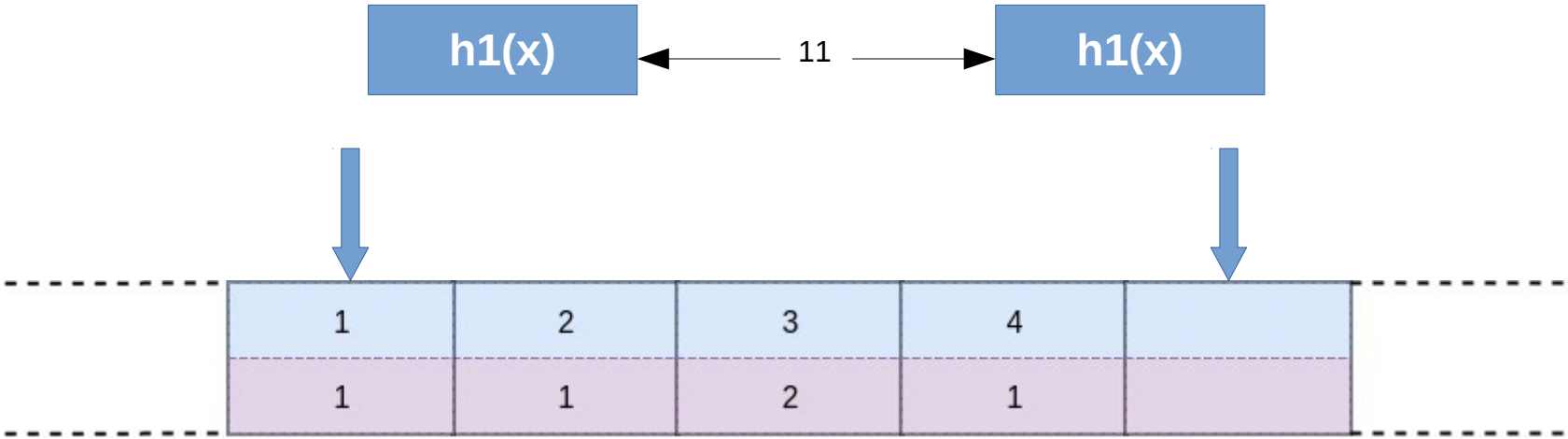
Two-choice Hashing

- Improvement on linear probing



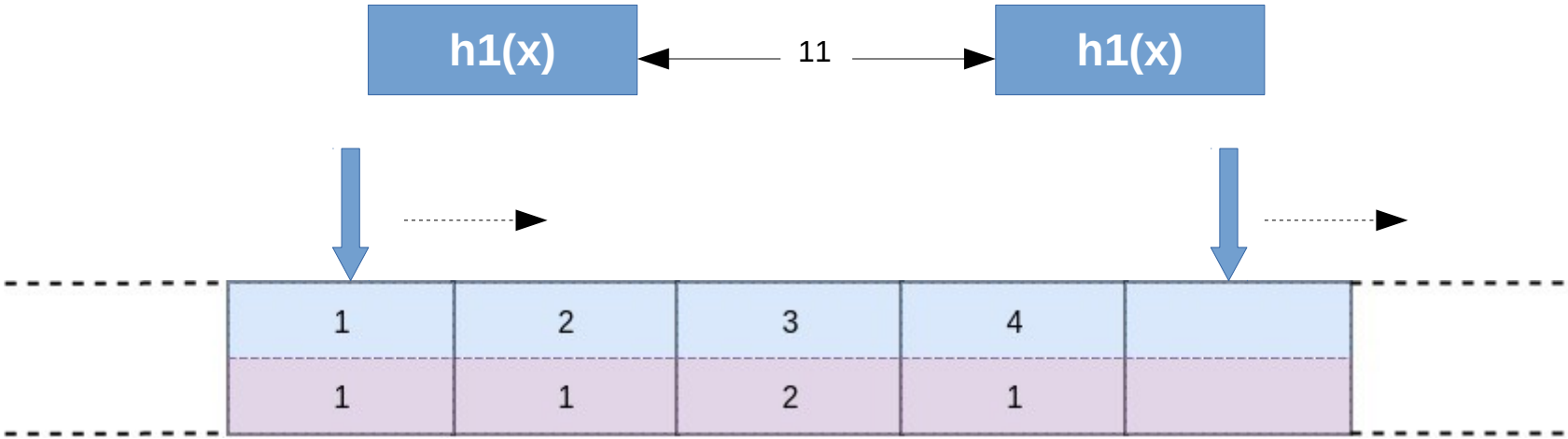
Two-choice Hashing

- Improvement on linear probing
- Two hashing function for same hash table (can be more)



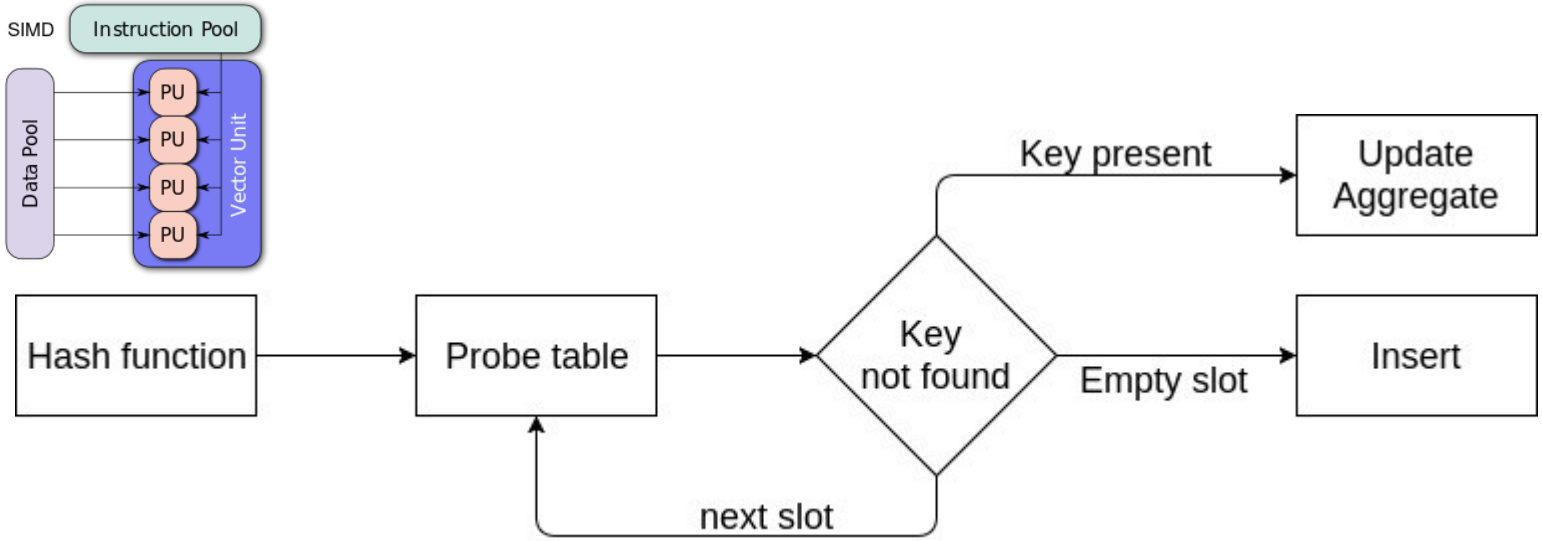
Two-choice Hashing

- Improvement on linear probing
- Two hashing function for same hash table (can be more)
- If all slots occupied, probing done from both slots



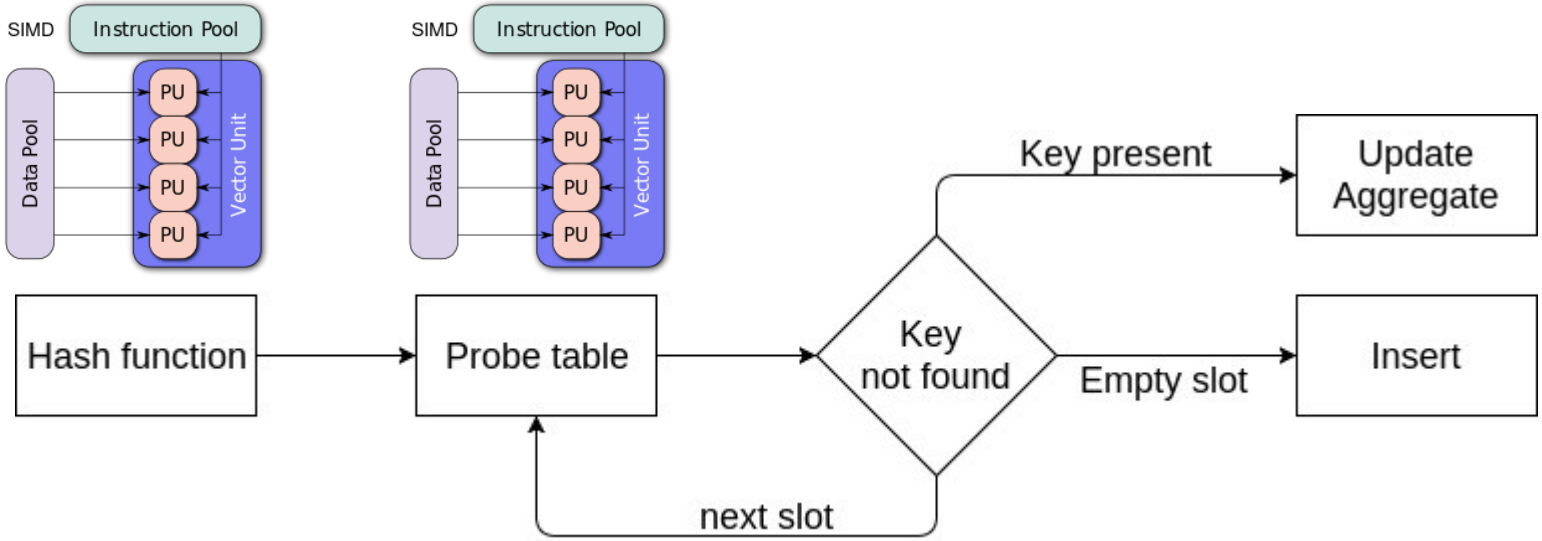
SIMD Optimization of Two-Choice Hashing

- SIMD hash function from cuckoo hashing



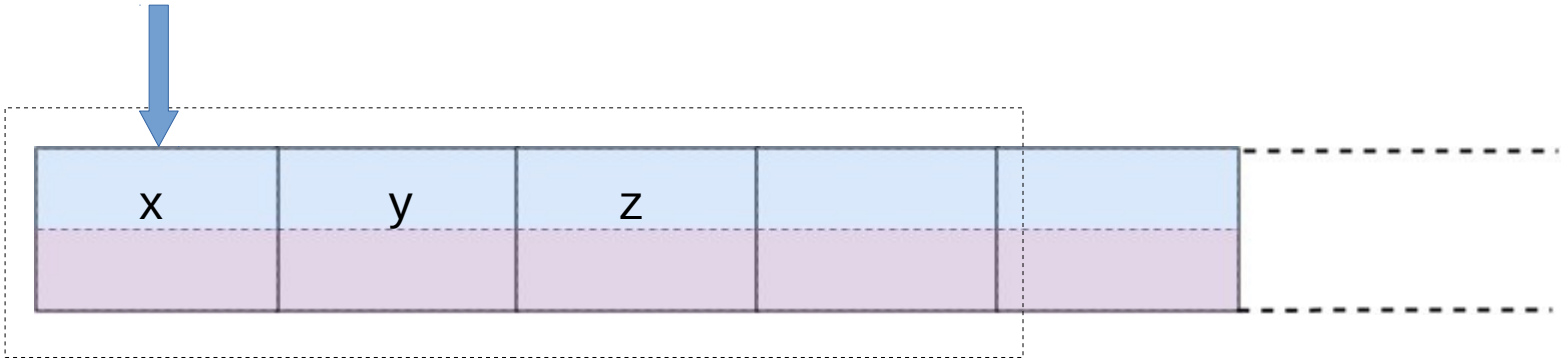
SIMD Optimization of Two-Choice Hashing

- SIMD hash function from cuckoo hashing
- SIMD probing from linear probing



Hopscotch Hashing

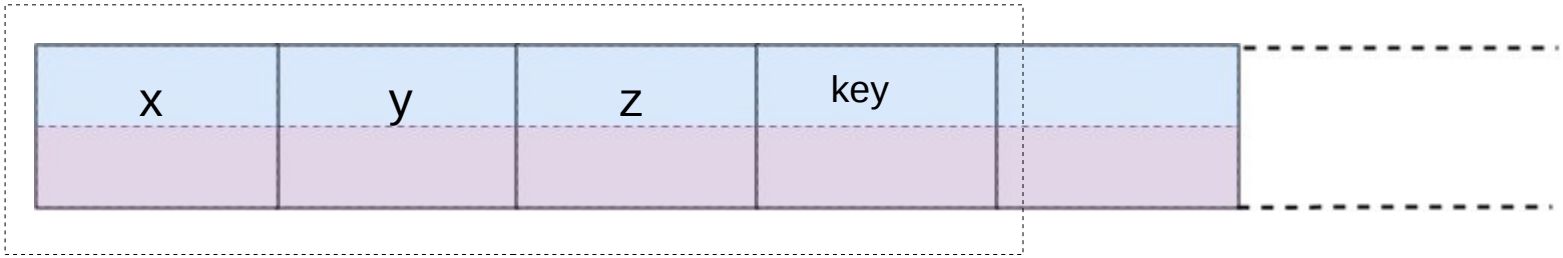
- Has a limited probe length known as neighborhood



Neighborhood : 3

Hopscotch Hashing

- Has a limited probe length known as neighborhood
- A key will be available within neighborhood distance from hash slot

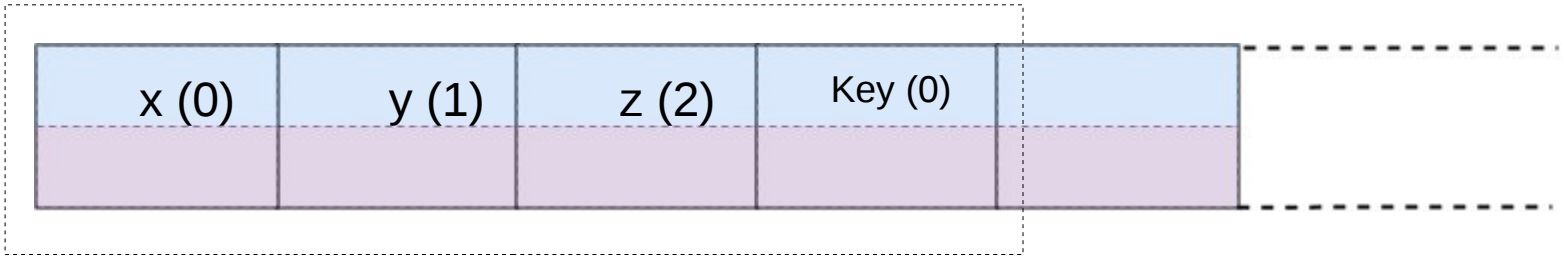


Neighborhood : 3

Hopscotch Hashing

- Has a limited probe length known as neighborhood
- A key will be available within neighborhood distance from hash slot
- If no slot available within neighborhood, table is rearranged by swapping keys

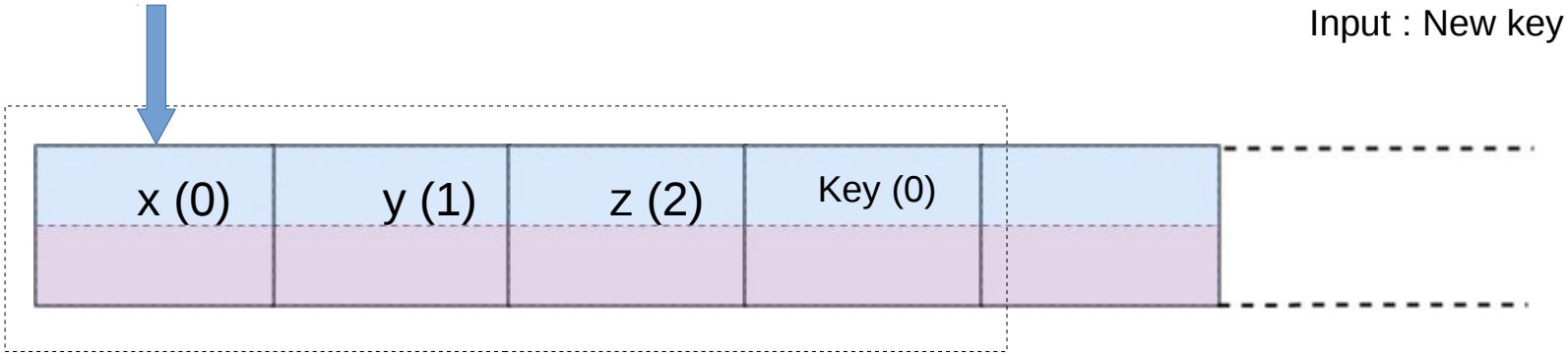
Input : New key



Neighborhood : 3

Hopscotch Hashing

- Has a limited probe length known as neighborhood
- A key will be available within neighborhood distance from hash slot
- If no slot available within neighborhood, table is rearranged by swapping keys

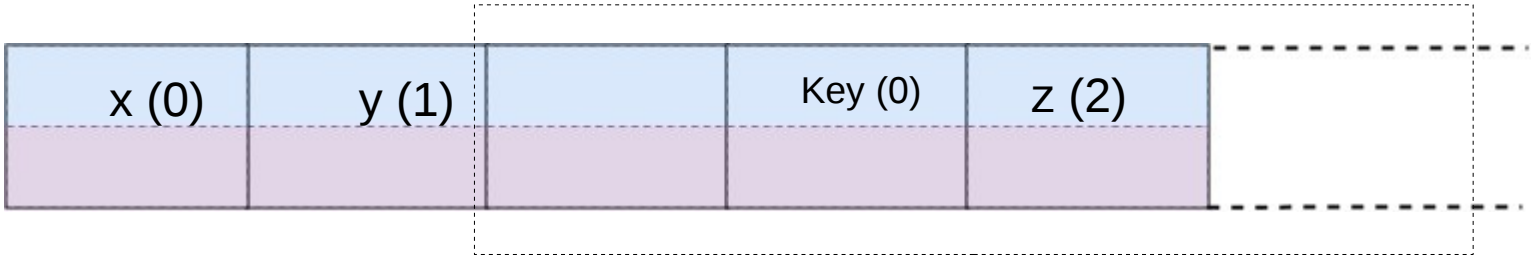


Neighborhood : 3

Hopscotch Hashing

- Has a limited probe length known as neighborhood
- A key will be available within neighborhood distance from hash slot
- If no slot available within neighborhood, table is rearranged by swapping keys

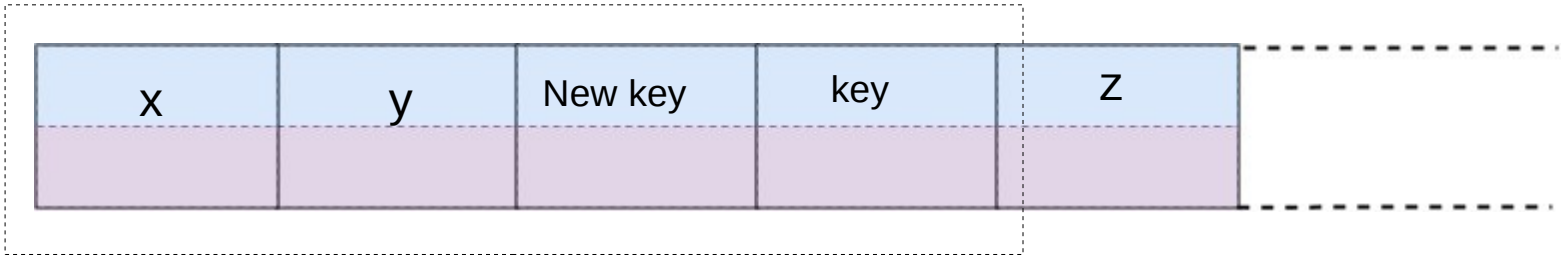
Input : New key



Neighborhood : 3

Hopscotch Hashing

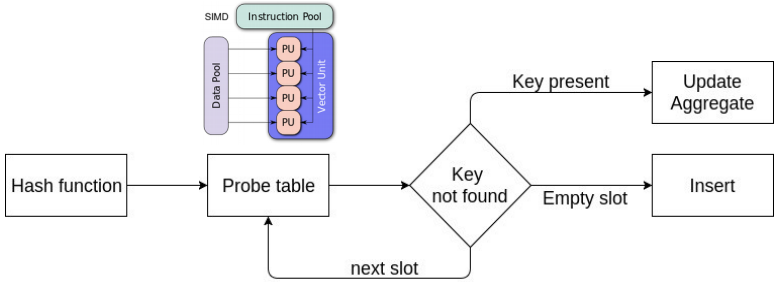
- Has a limited probe length known as neighborhood
- A key will be available within neighborhood distance from hash slot
- If no slot available within neighborhood, table is rearranged by swapping keys



Neighborhood : 3

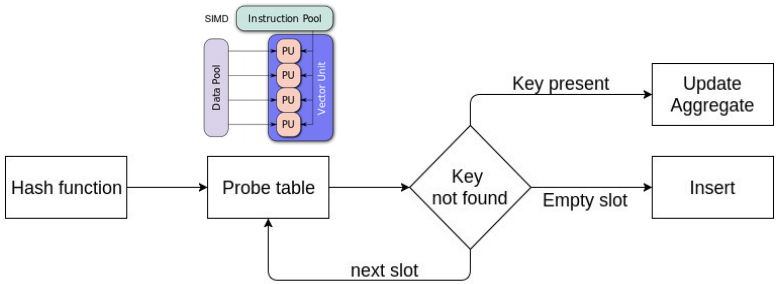
SIMD Optimization of Hopscotch Hashing

- **SIMD for Probing**
  - Uses SIMD linear probing

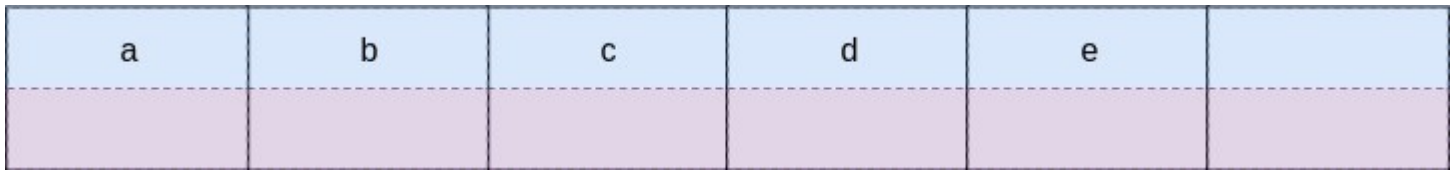


SIMD Optimization of Hopscotch Hashing

- **SIMD for Probing**
  - Uses SIMD linear probing

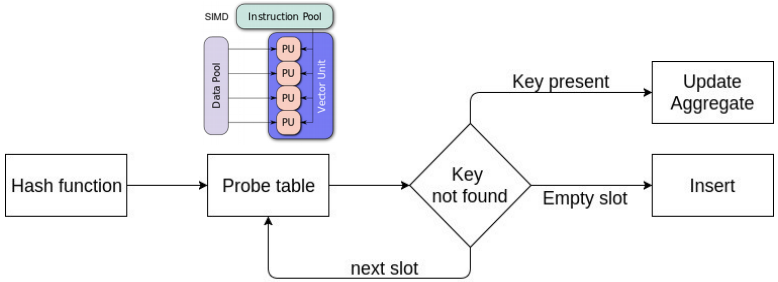


- **SIMD for Insertion**
  - Swapping multiple values in an instant

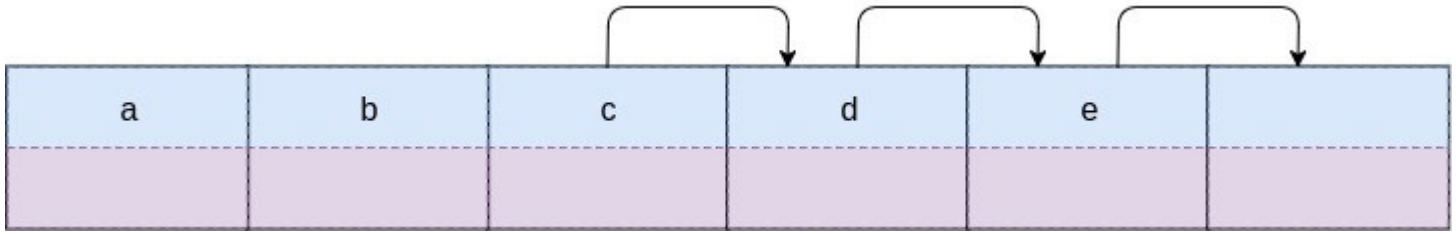


SIMD Optimization of Hopscotch Hashing

- **SIMD for Probing**
  - Uses SIMD linear probing

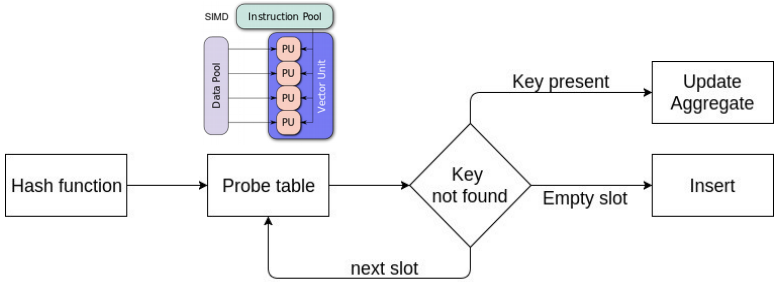


- **SIMD for Insertion**
  - Swapping multiple values in an instant

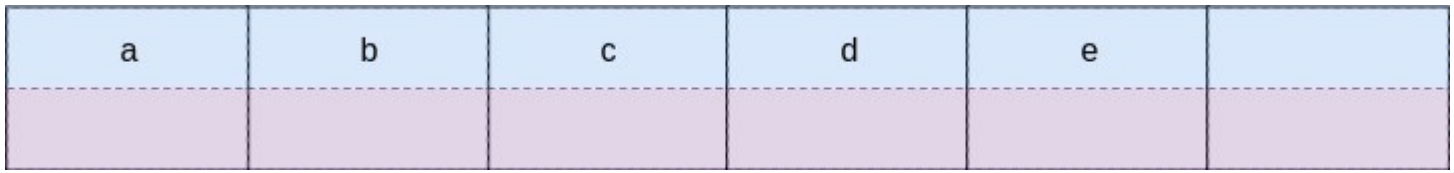


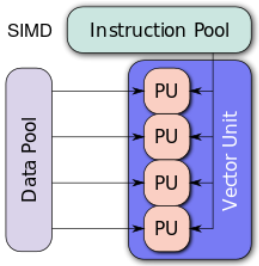
SIMD Optimization of Hopscotch Hashing

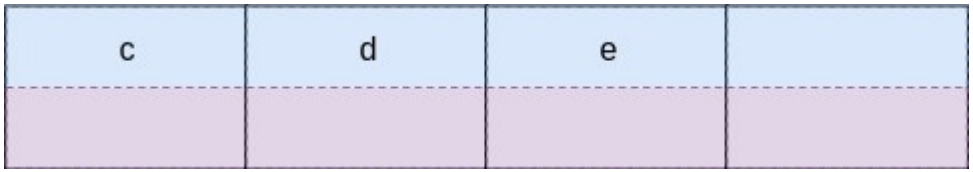
- **SIMD for Probing**
  - Uses SIMD linear probing



- **SIMD for Insertion**
  - Swapping multiple values in an instant
  - Collect values using SIMD Gather

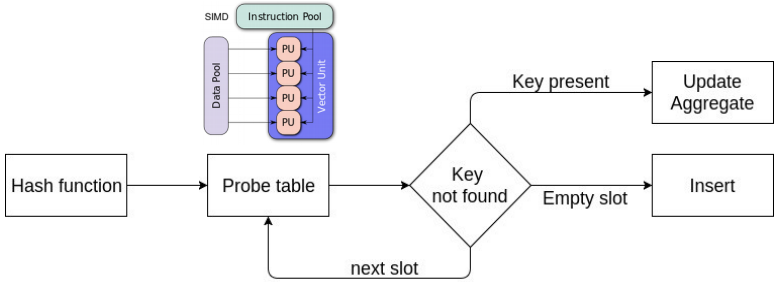




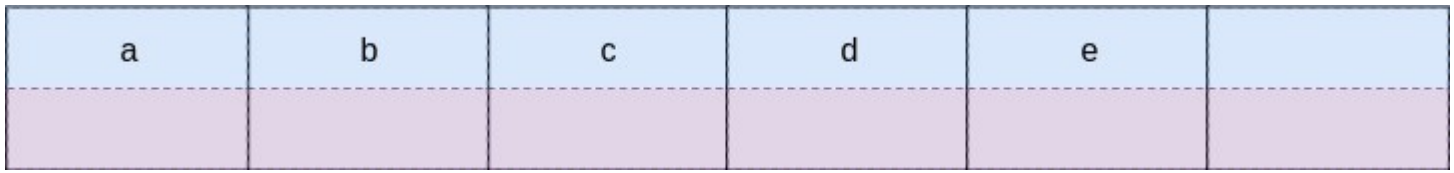


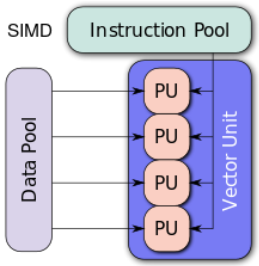
SIMD Optimization of Hopscotch Hashing

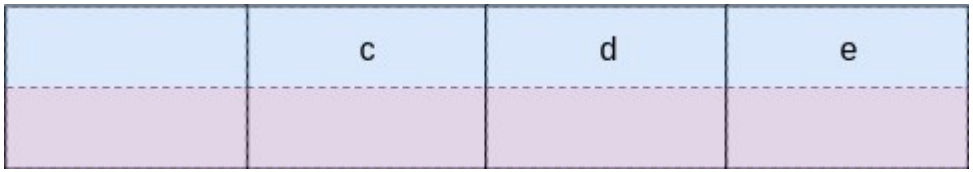
- **SIMD for Probing**
  - Uses SIMD linear probing



- **SIMD for Insertion**
  - Swapping multiple values in an instant
  - Collect values using SIMD Gather
  - Swap them

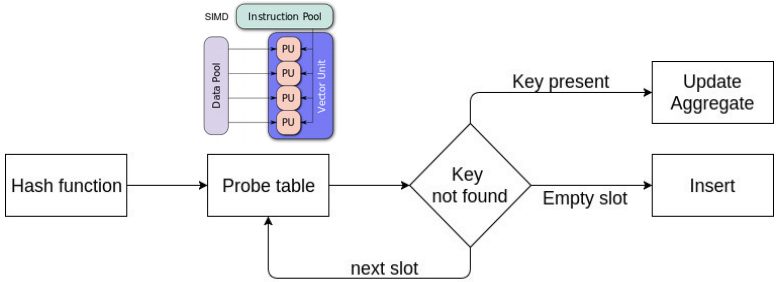




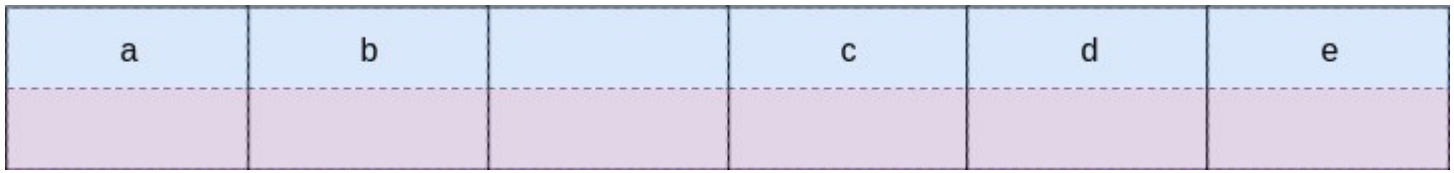


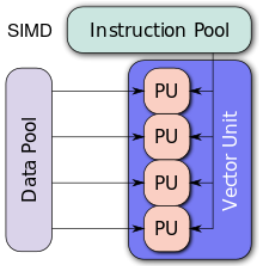
SIMD Optimization of Hopscotch Hashing

- **SIMD for Probing**
  - Uses SIMD linear probing



- **SIMD for Insertion**
  - Swapping multiple values in an instant
  - Collect values using SIMD Gather
  - Swap them
  - Store back

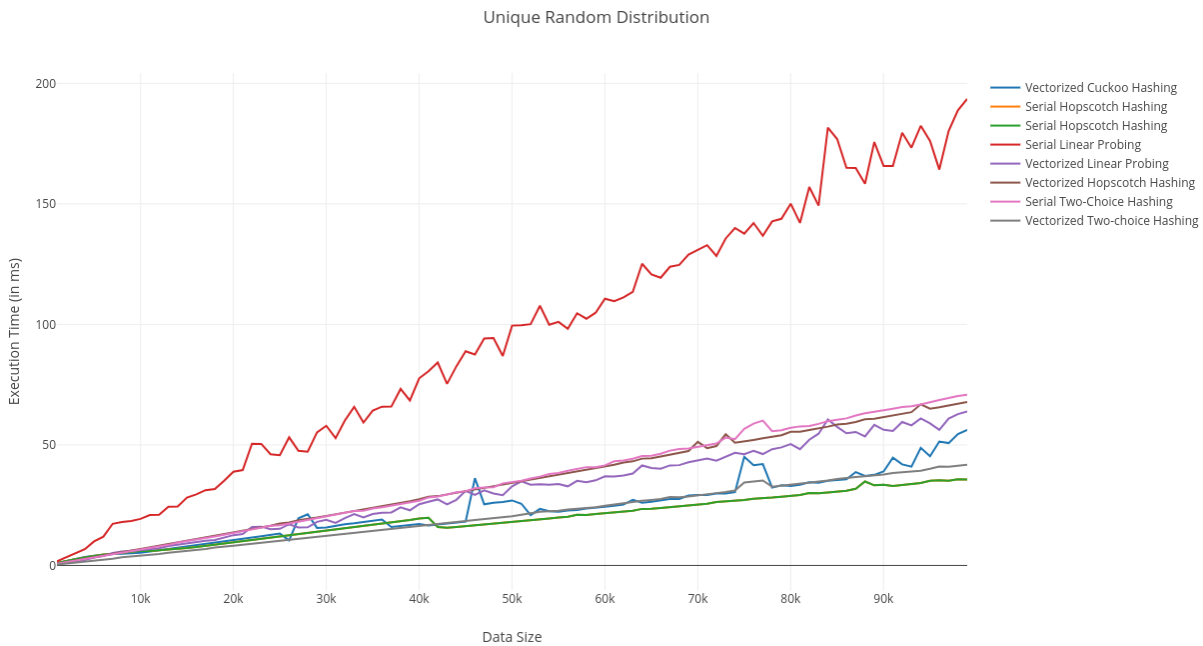




Evaluation Setup

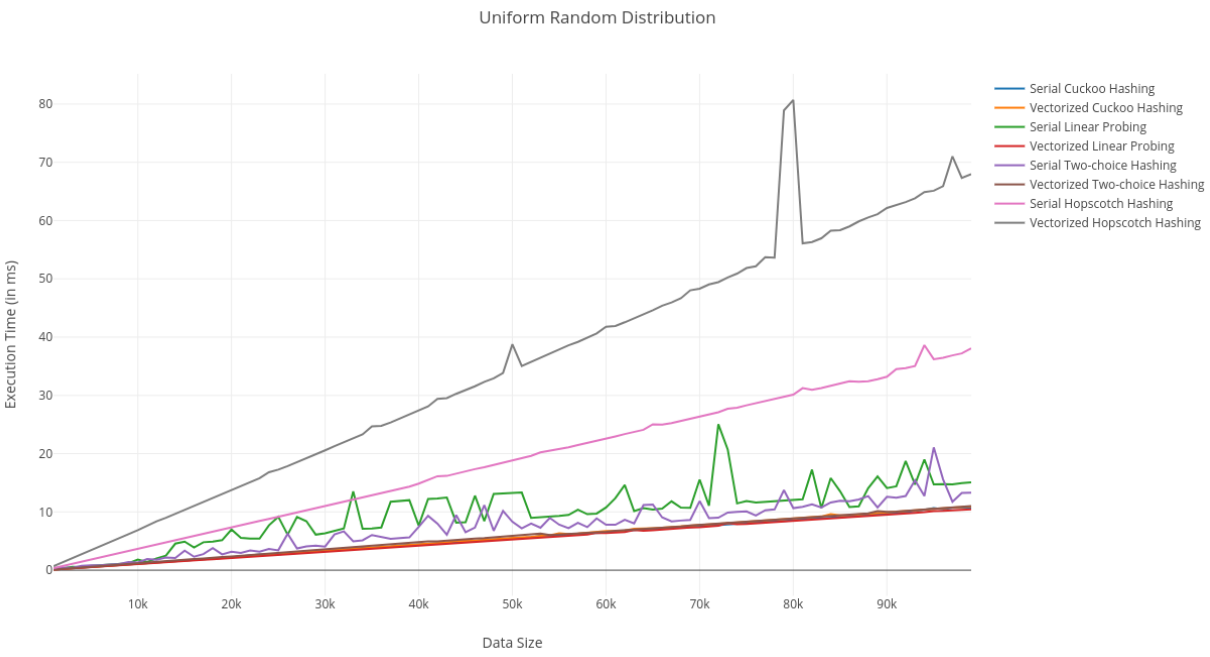
- **Environment**
  - Processor : Intel Octa Core Xeon E5-2630
  - OS: Linux
  - SIMD : AVX2 instruction set
- **Grouped Aggregation**
  - Keys : 32 bit integers (0 is not valid)
  - Aggregation : count()
- **Hashing technique**
  - Hash function : Multiplicative function – h(x) = Ax%tableSize
    - A => knuth's number
- **Data Distribution**
  - Uniform random
  - Sequential
  - Unique random
  - Moving cluster

Unique Random Distribution



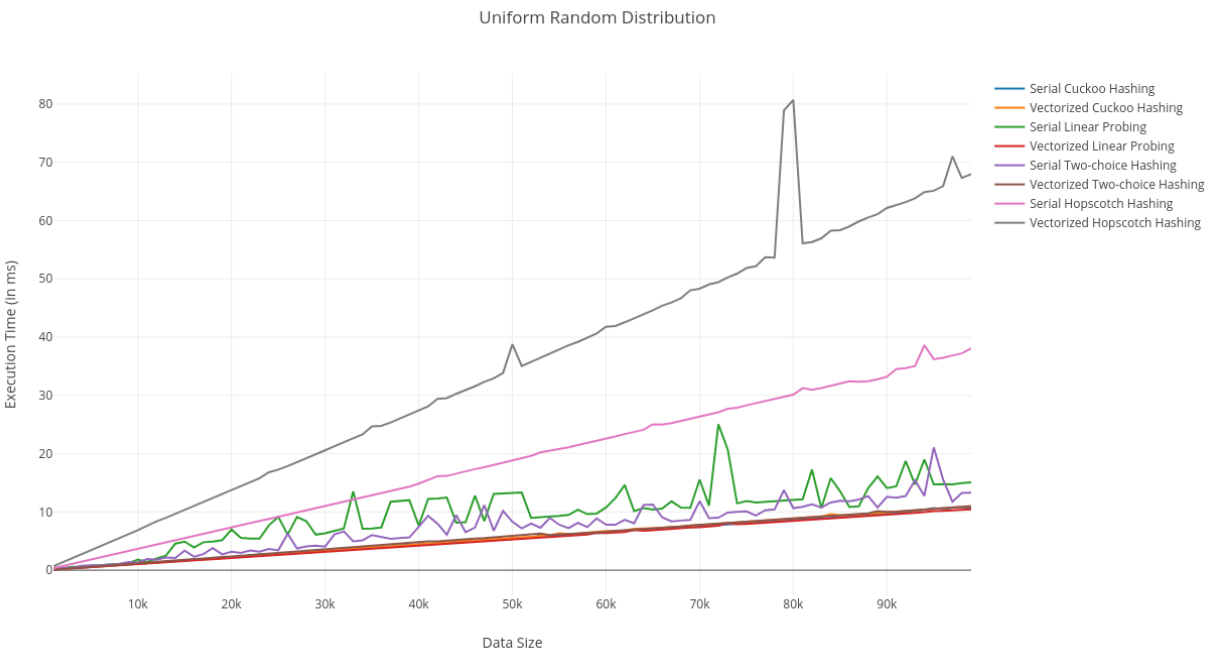
- Serial linear probing has worse time
- Serial hopscotch hashing is efficient
- Vectorized two-choice hashing competes with Hopscotch hashing

Uniform Random Distribution



- Both serial and vectorized hopscotch has worse efficiency
  - Vectorized with the worst time
- Vectorized Two-choice hashing has best execution time

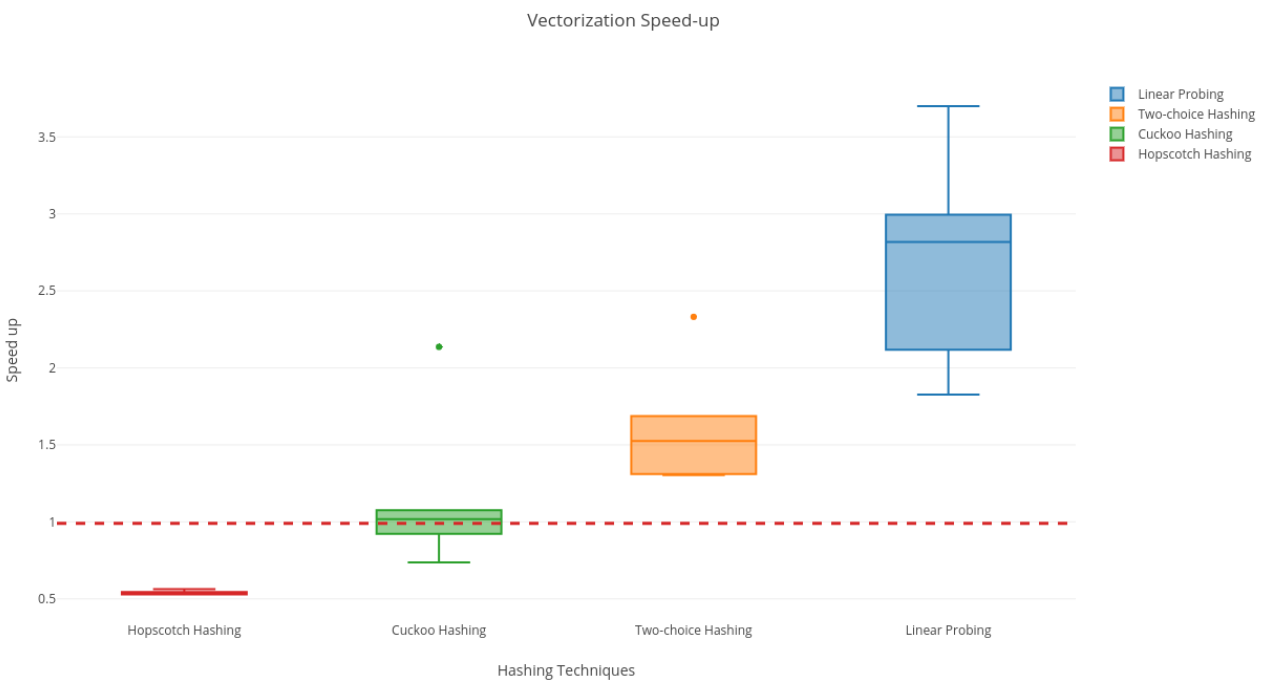
Uniform Random Distribution



- Both serial and vectorized hopscotch has worse efficiency
  - Vectorized with the worst time
- Vectorized Two-choice hashing has best execution time

Other distributions have the same characteristics

Overall Speed-up



- SIMD has worse impact on hopsctoch hashing
  - Nearly 2x slower
  - Due to random access for insertion
- Linear probing has highest SIMD impact

Summary

- Hash probing reduces efficiency of grouped aggregation

Summary

- Hash probing reduces efficiency of grouped aggregation
- SIMD accelerated probing not always improves efficiency
  - Specifically, hopscotch hashing has negative impact
  - Linear probing has up to 3.5x speed up due to SIMD

Summary

- Hash probing reduces efficiency of grouped aggregation
- SIMD accelerated probing not always improves efficiency
  - Specifically, hopscotch hashing has negative impact
  - Linear probing has up to 3.5x speed up due to SIMD
- Hashing technique related parameters also improve the efficiency

Summary

- Hash probing reduces efficiency of grouped aggregation
- SIMD accelerated probing not always improves efficiency
  - Specifically, hopscotch hashing has negative impact
  - Linear probing has up to 3.5x speed up due to SIMD
- Hashing technique related parameters also improve the efficiency
- Further, improvement can be extended using SIMD for multiple insertion

Questions?

Thank You

References

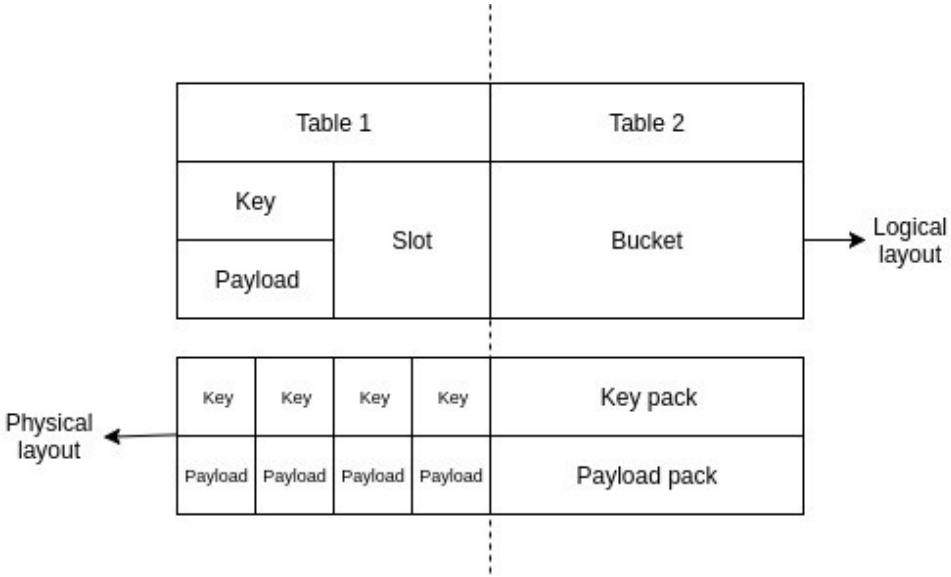
[1] Boncz, P., Neumann, T., & Erling, O. (2014). TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark. Springer International Publishing.

[2] Orestis Polychroniou, Arun Raghavan, and Kenneth A. Ross. (2015). Rethinking SIMD Vectorization for In-Memory Databases. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15)

[3] Broneske, D., Meister, A., & Saake, G. (2017). Hardware-Sensitive Scan Operator Variants for Compiled Selection Pipelines. BTW

SIMD Cuckoo Hashing – Table Structure

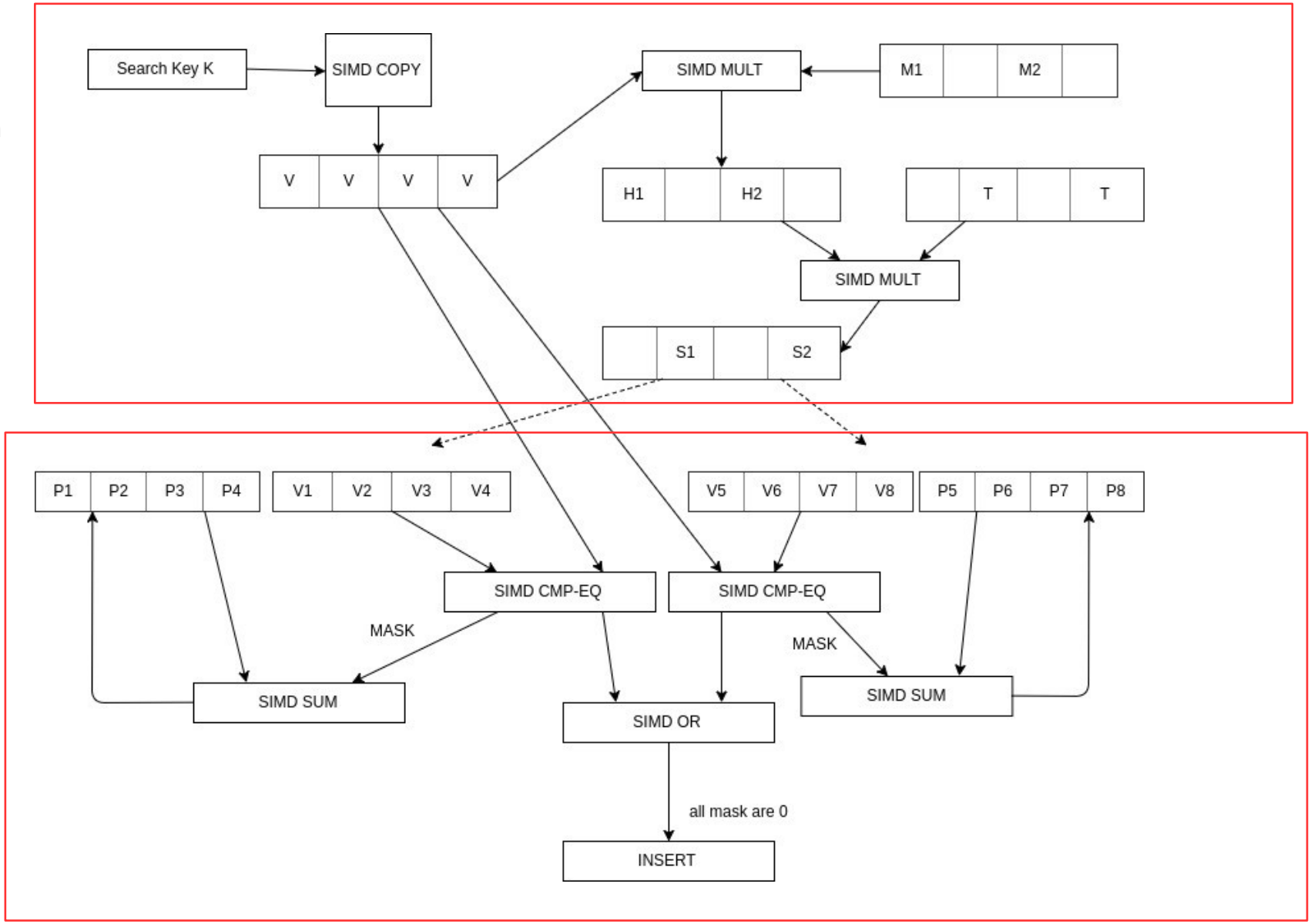
- **Table Structure**
  - Packed key and payload
  - Each bucket contain one key, payload pack
- Pack size = SIMD vector size
- Ross et al. explored SIMD probing [4]



SIMD Acclerated Cuckoo Hashing

Hash Function Compute (Multiplicative hashing)

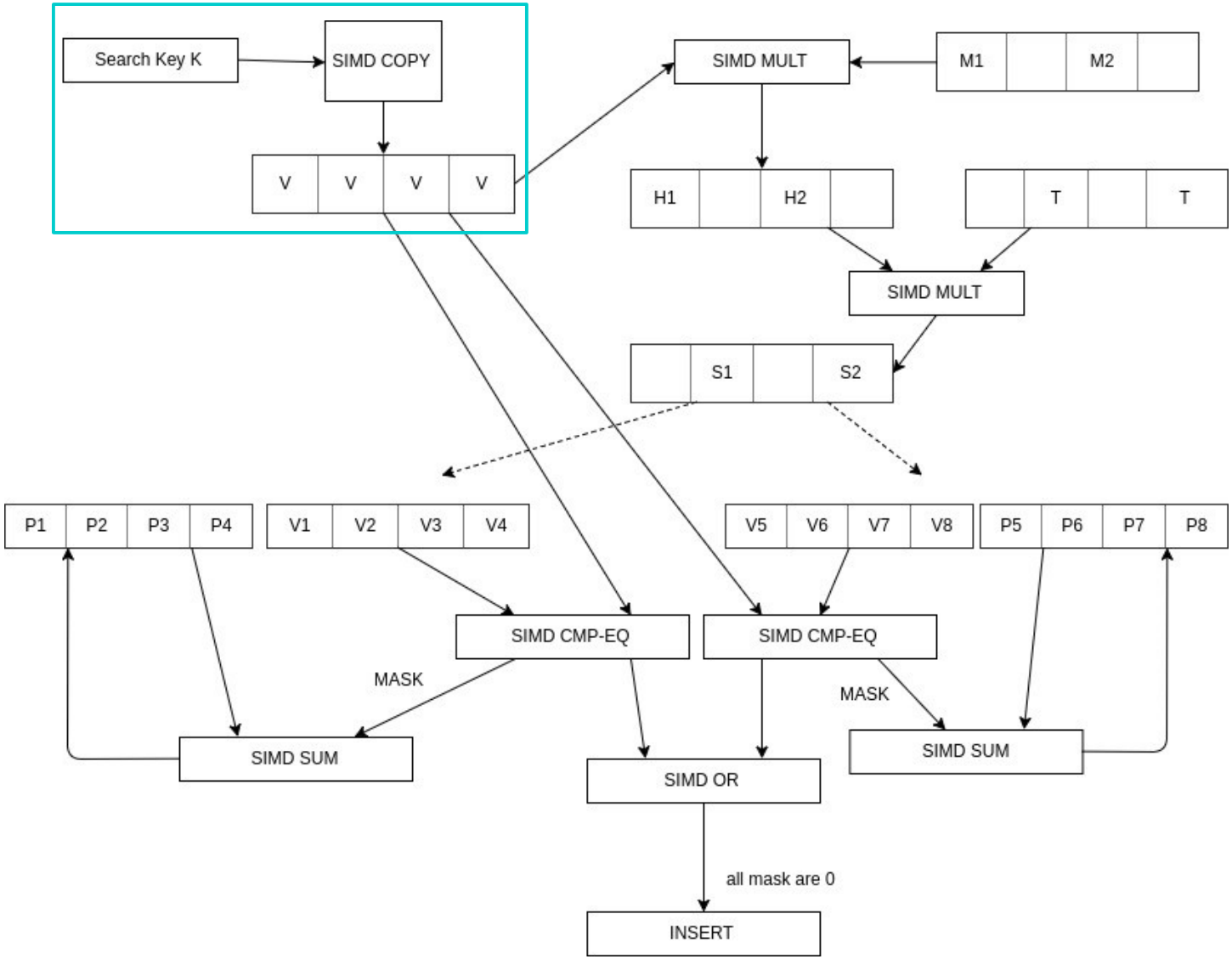
Table Probe



Adapted from [6]

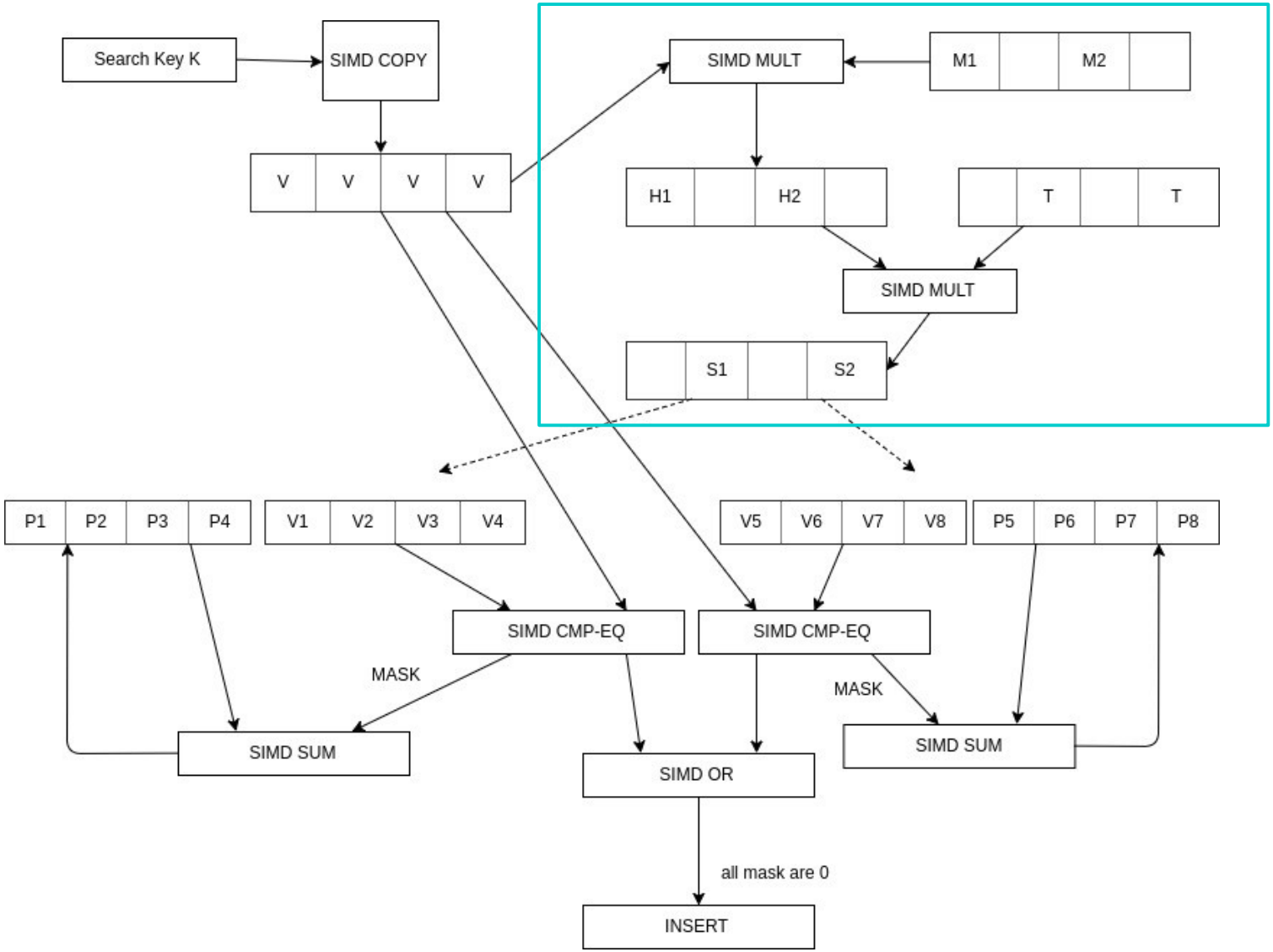
SIMD Acclerated Cuckoo Hashing

Search key – K is duplicated



Adapted from [6]

SIMD Acclerated Cuckoo Hashing



- Hashing function
- Returns two bucket positions

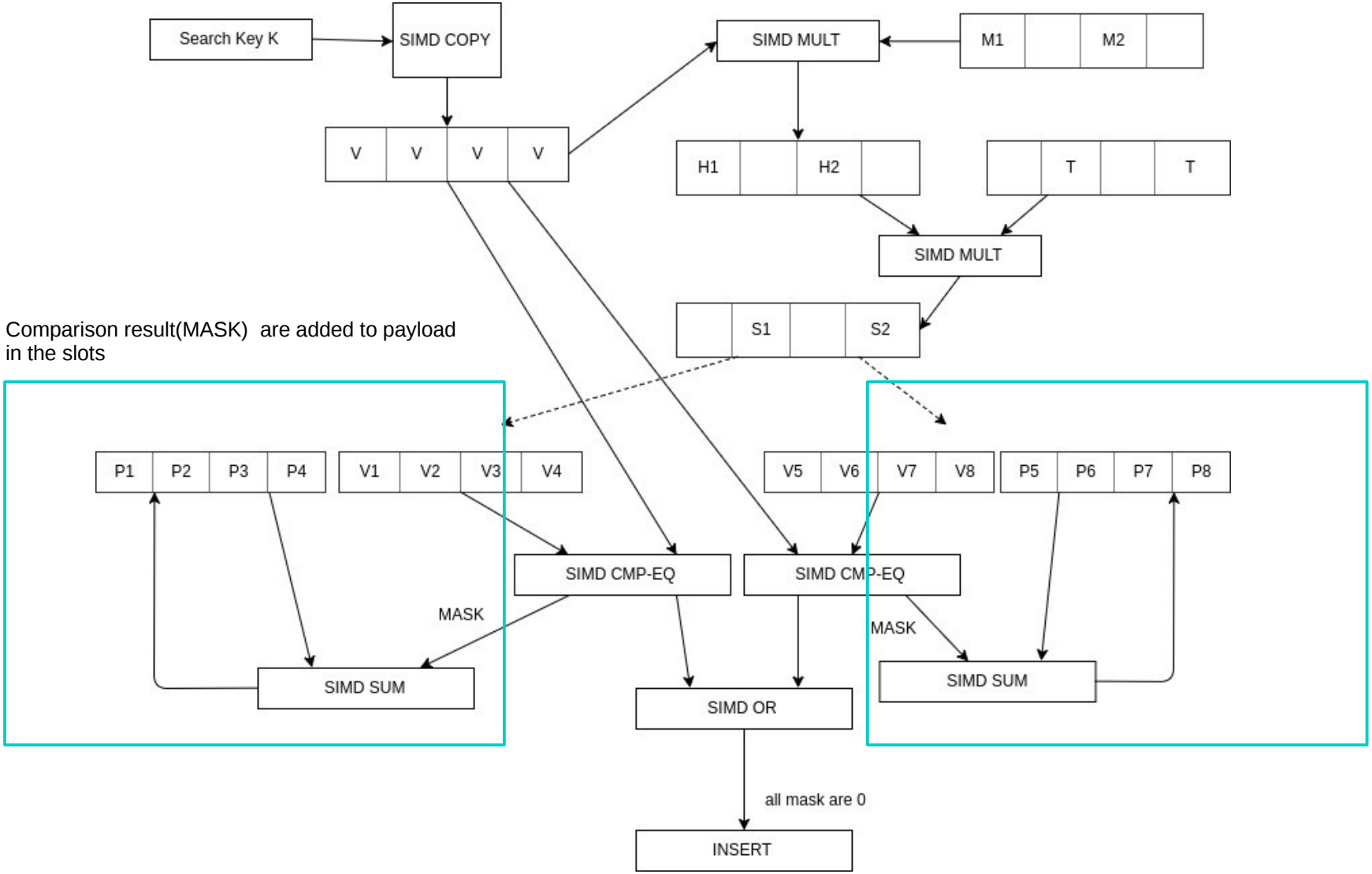
Adapted from [6]

SIMD Acclerated Cuckoo Hashing

- Probe values in the table
- Compare with search key

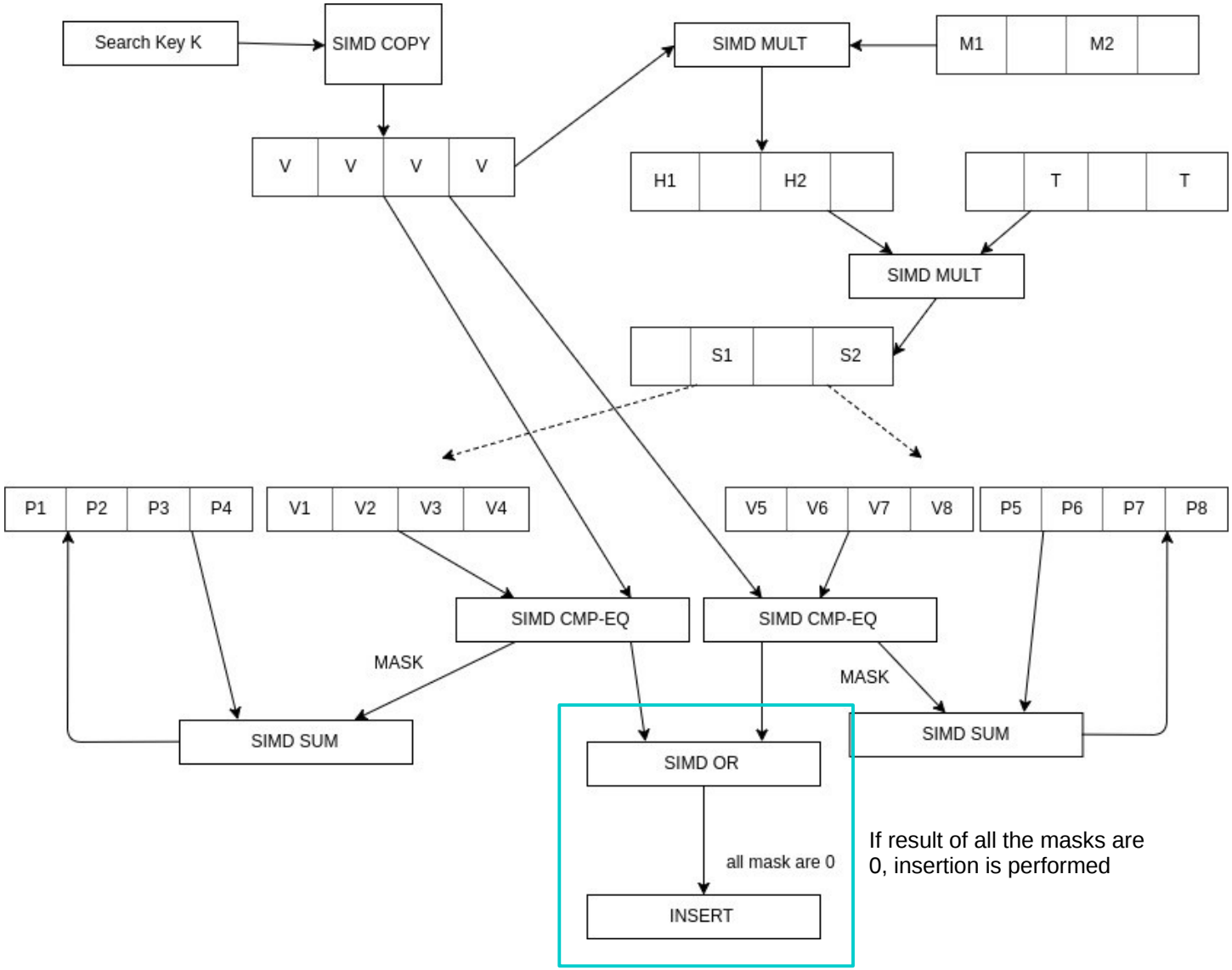
Adapted from [6]

SIMD Acclerated Cuckoo Hashing



Adapted from [6]

SIMD Acclerated Cuckoo Hashing

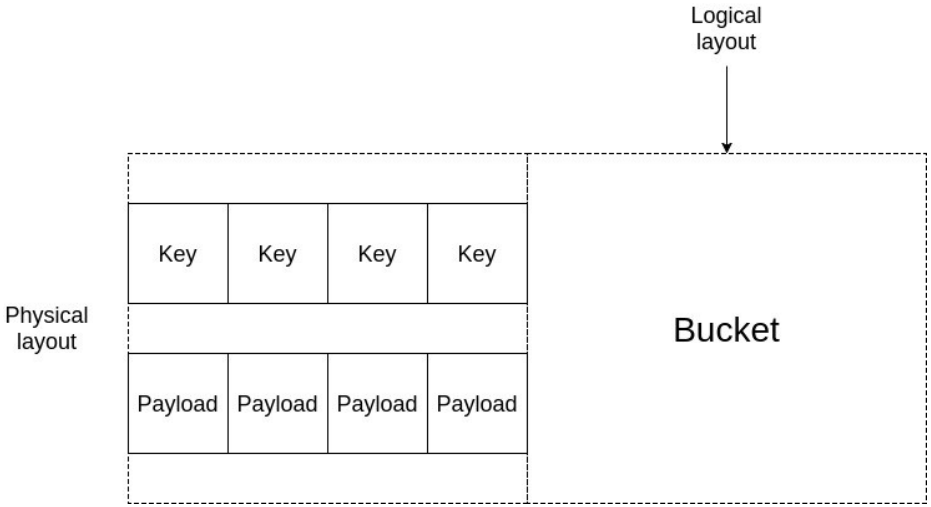


If result of all the masks are 0, insertion is performed

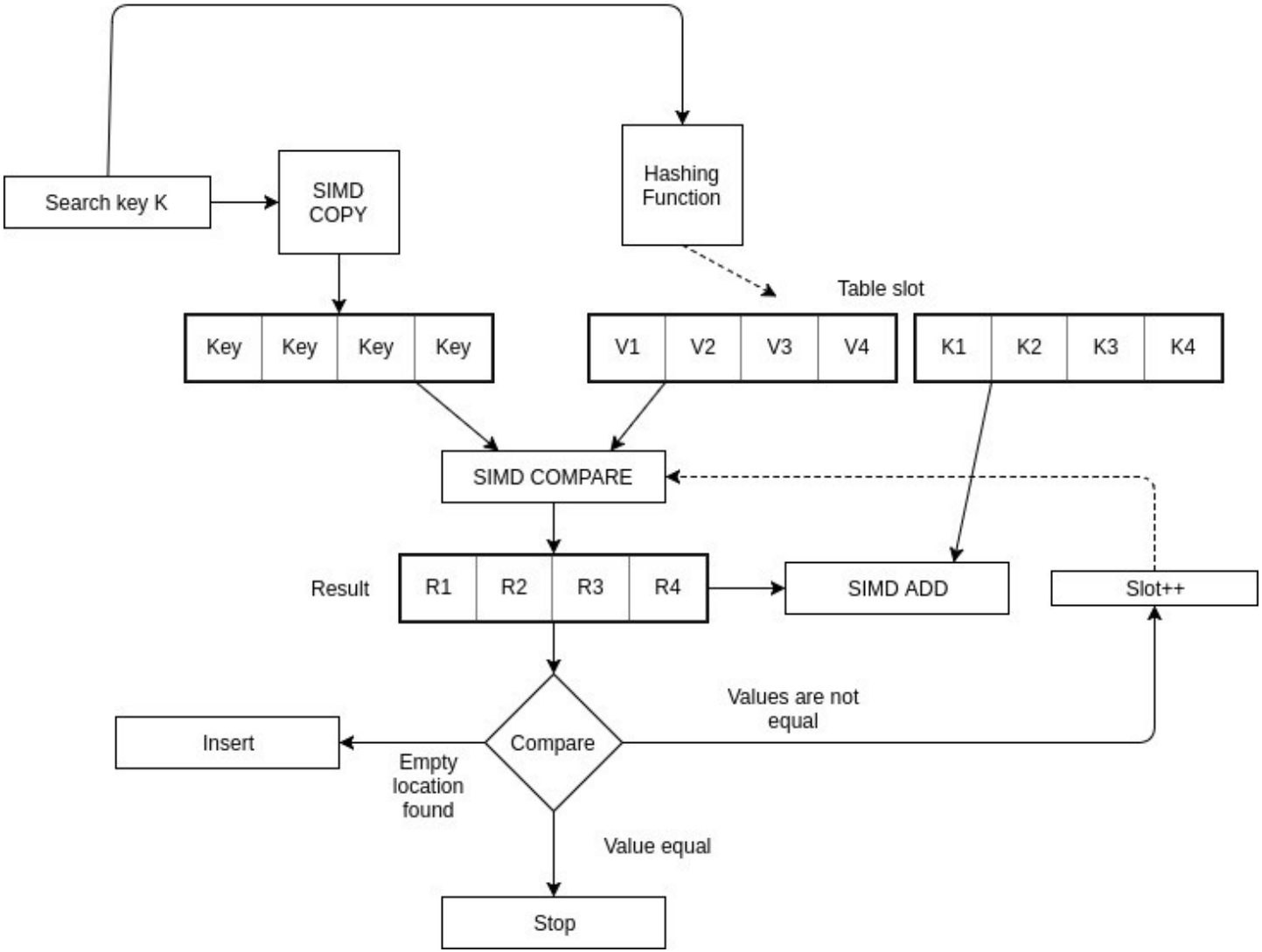
Adapted from [6]

Linear Probing

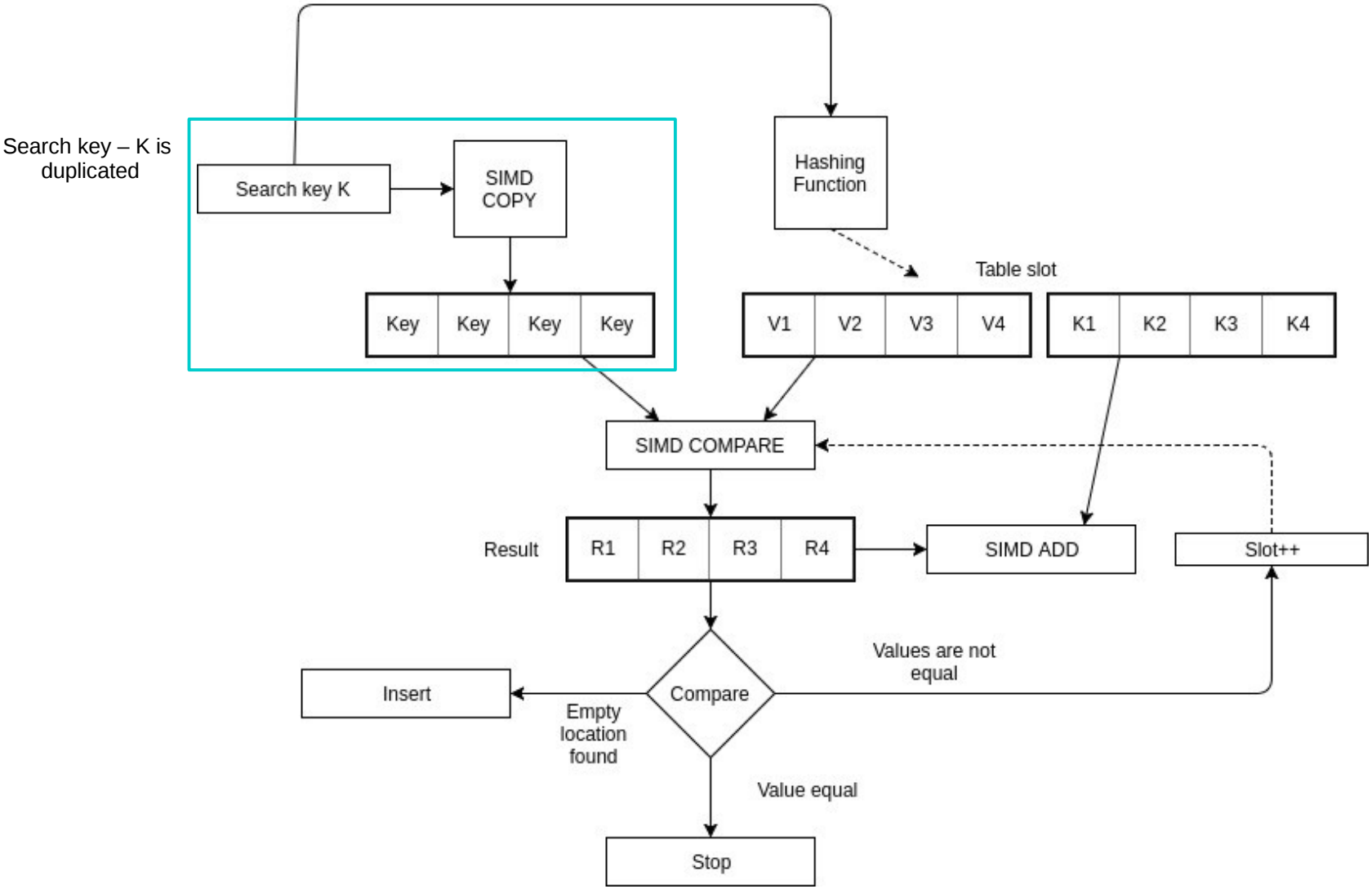
- **Table Structure**
  - Has SoA (Structure of Array format)
  - Key and payload in individual array
  - payload in corresponding key position



Linear Probing – SIMD Code Optimization

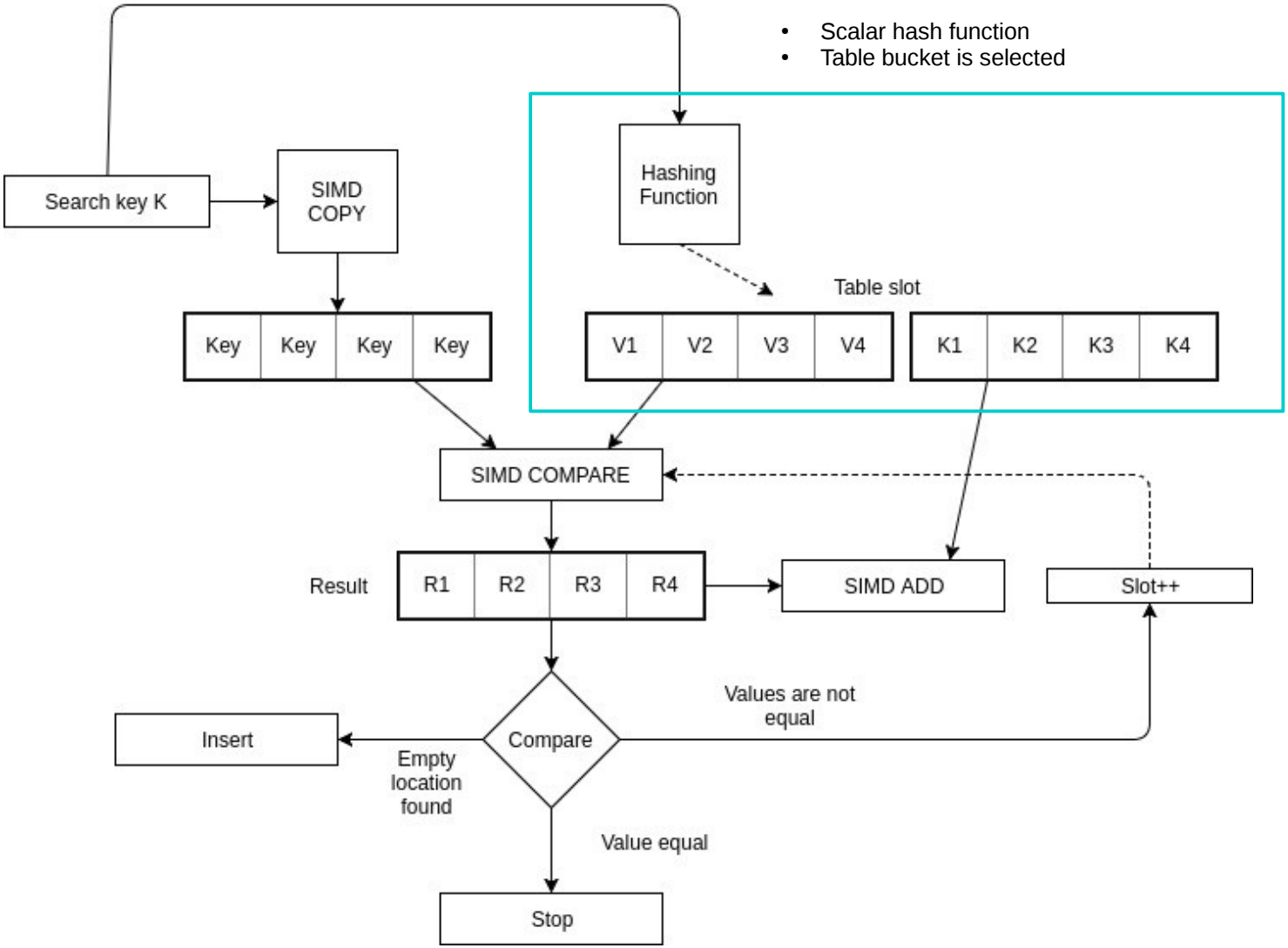


Linear Probing – SIMD Code Optimization

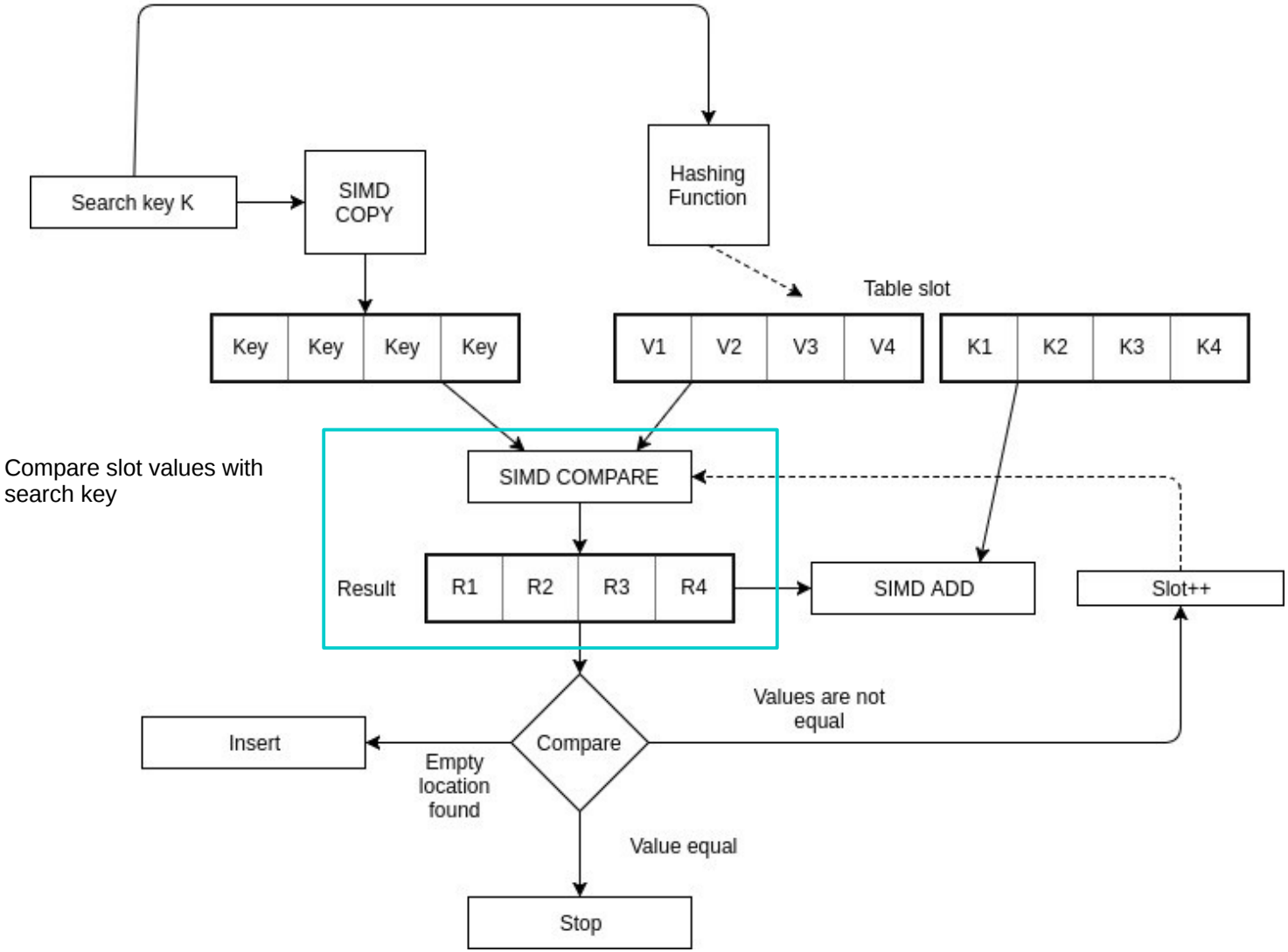


Linear Probing – SIMD Code Optimization

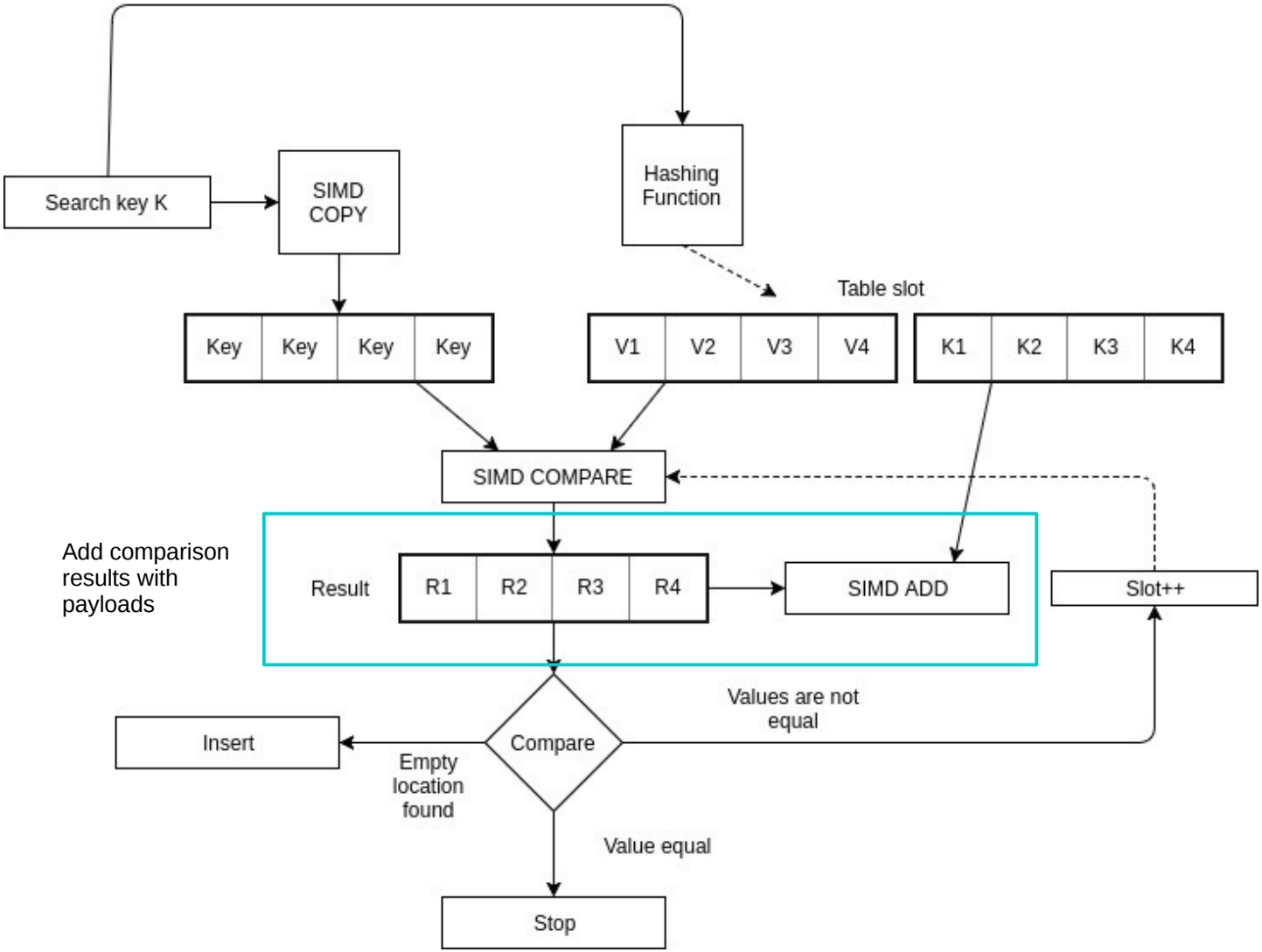
- Scalar hash function
- Table bucket is selected



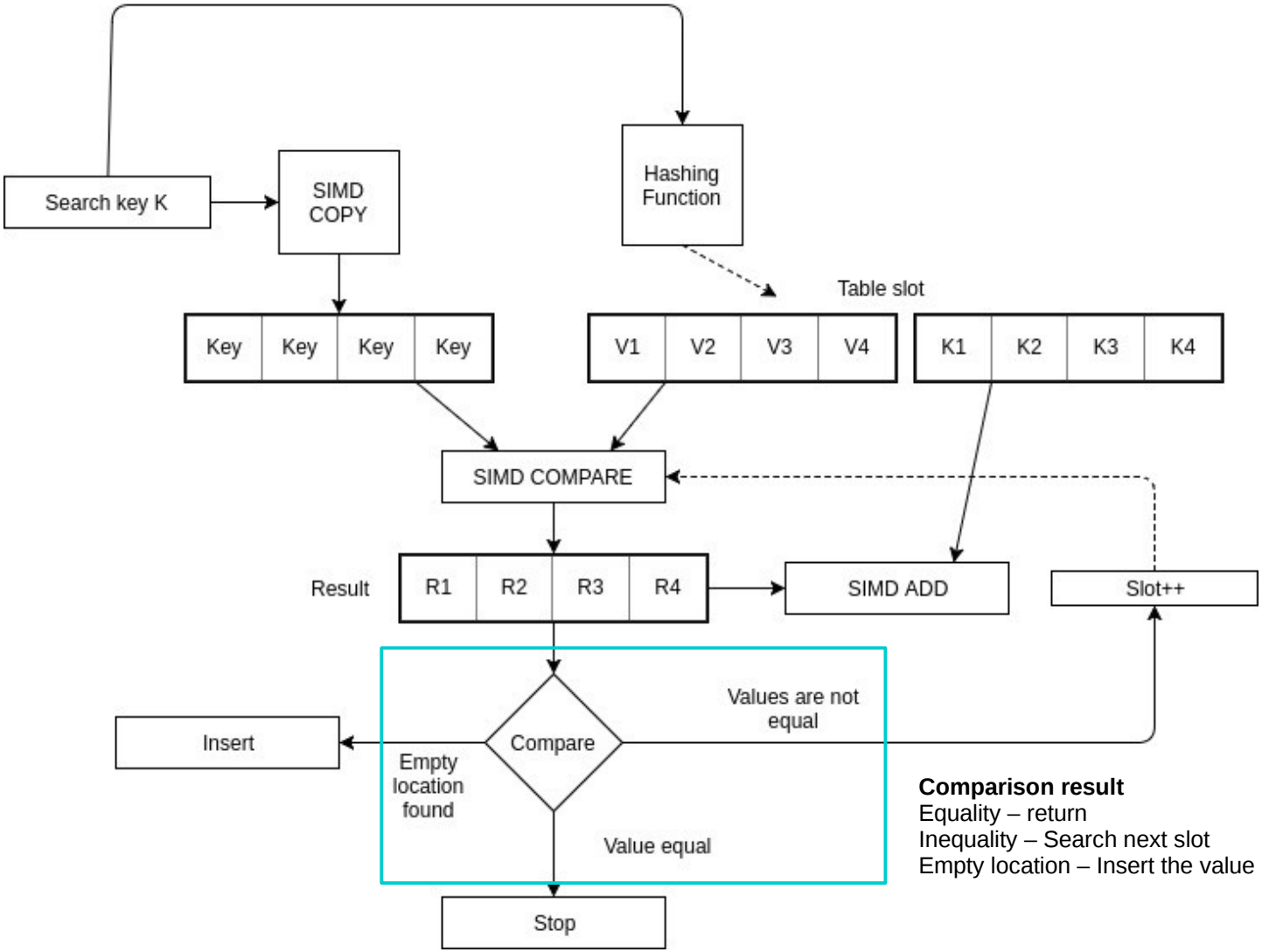
Linear Probing – SIMD Code Optimization



Linear Probing – SIMD Code Optimization



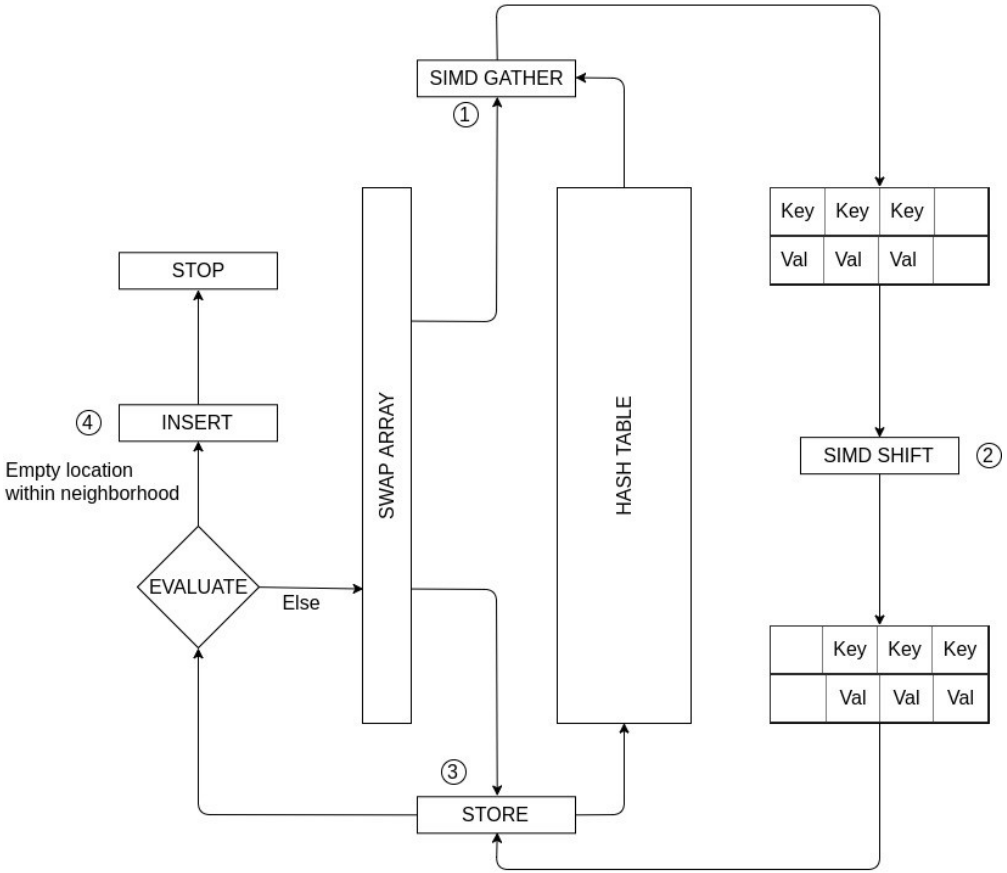
Linear Probing – SIMD Code Optimization



Comparison result
Equality – return
Inequality – Search next slot
Empty location – Insert the value

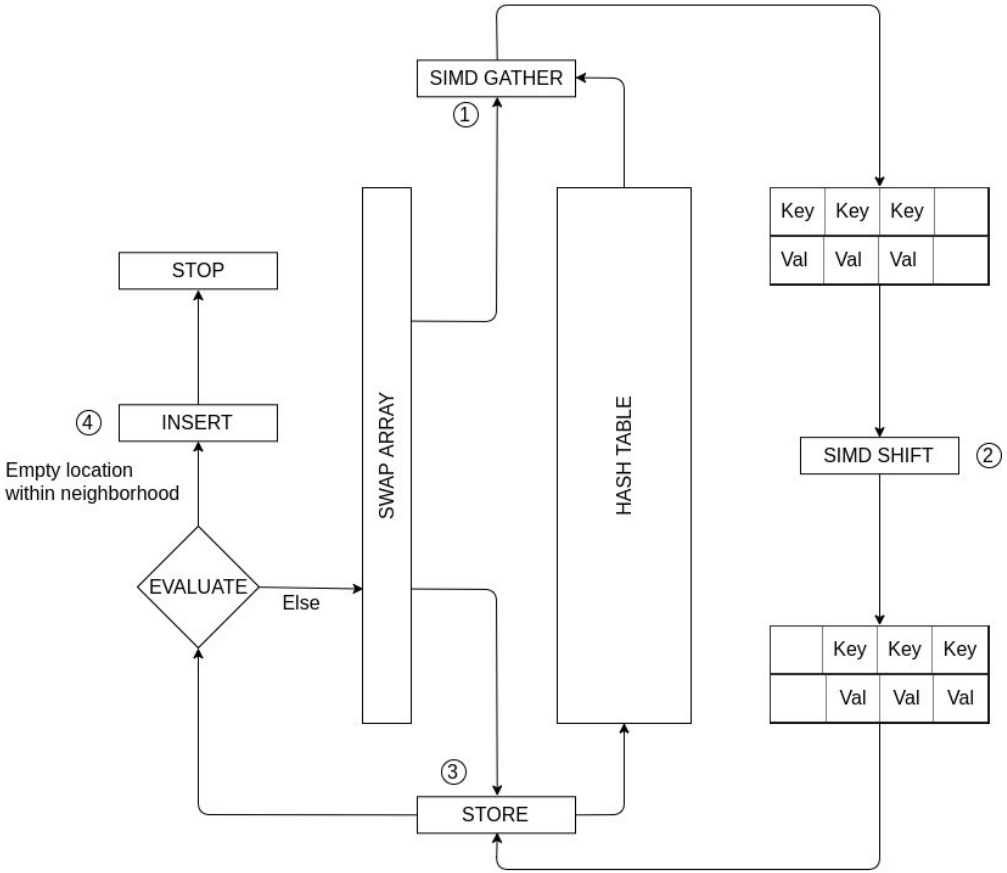
SIMD Accelerated Hopscotch Hashing

- **SIMD for Probing**
  - Uses SIMD linear probing
  - Probe key within Neighborhood
  - Probe empty space outside
- **SIMD For Insertion**
  - Starts when empty space is found



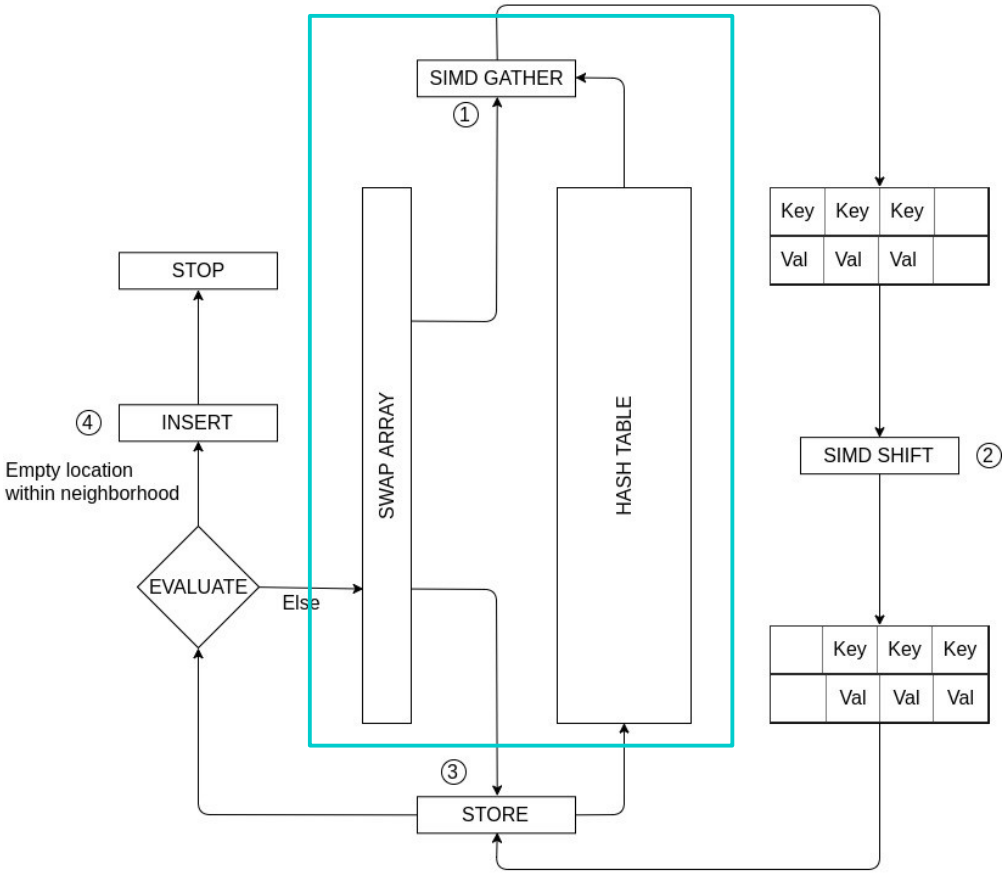
SIMD Accelerated Hopscotch Hashing

- **SIMD for Probing**
  - Uses SIMD linear probing
  - Probe key within Neighborhood
  - Probe empty space outside
- **SIMD For Insertion**
  - Starts when empty space is found
  - Swap previous values until empty space is inside neighborhood



SIMD Accelerated Hopscotch Hashing

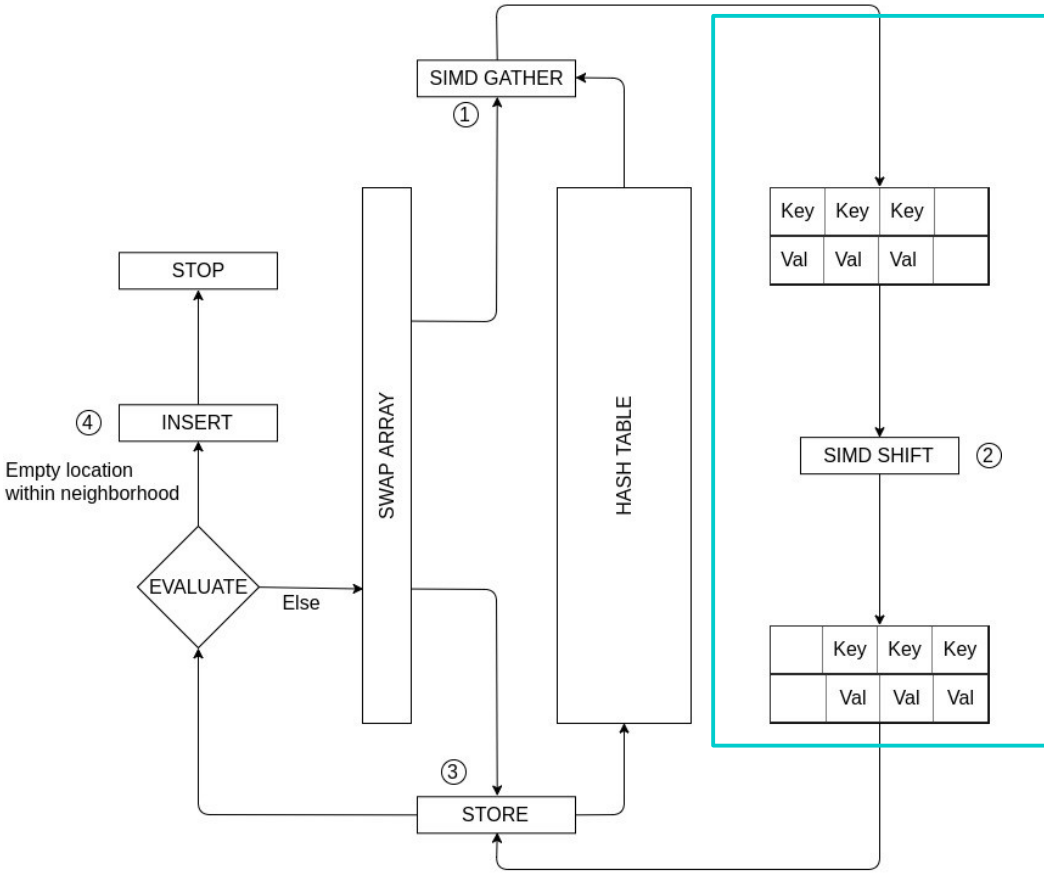
- **SIMD for Probing**
  - Uses SIMD linear probing
  - Probe key within Neighborhood
  - Probe empty space outside
- **SIMD For Insertion**
  - Starts when empty space is found
  - Swap previous values until empty space is inside neighborhood



Swap array holds key to swap
Use SIMD gather to collect keys from hash table

SIMD Accelerated Hopscotch Hashing

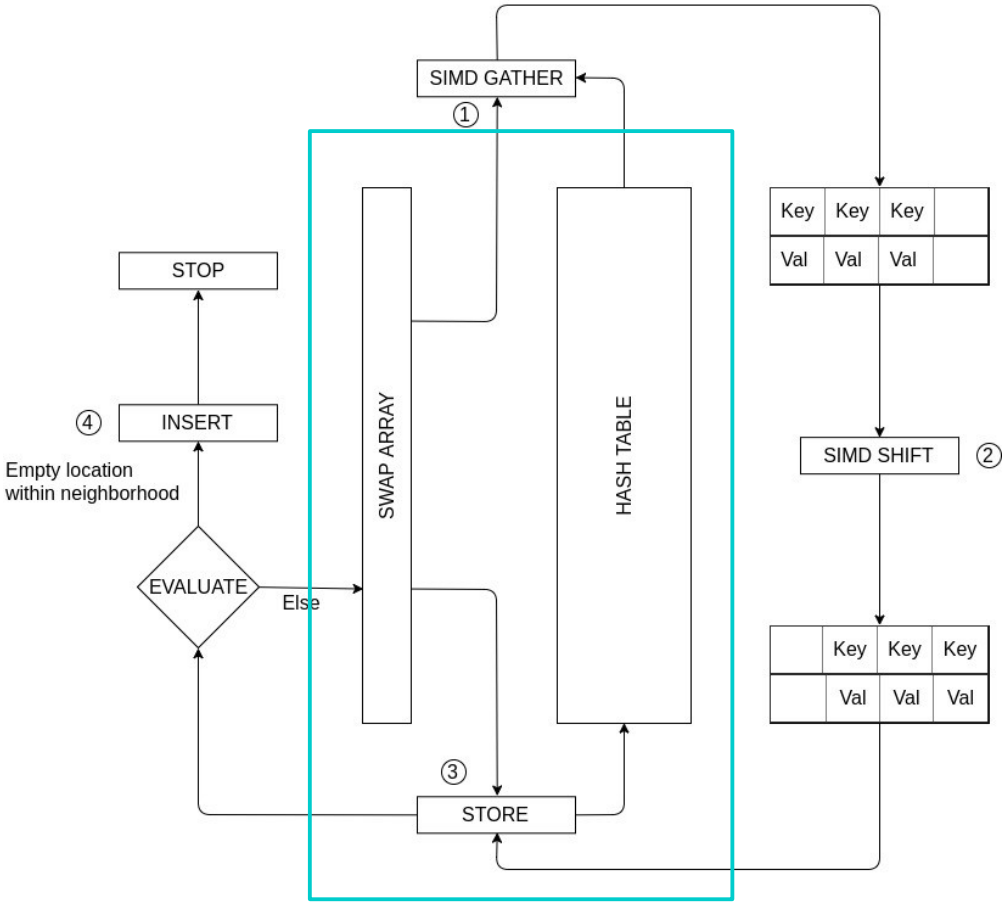
- **SIMD for Probing**
  - Uses SIMD linear probing
  - Probe key within Neighborhood
  - Probe empty space outside
- **SIMD For Insertion**
  - Starts when empty space is found
  - Swap previous values until empty space is inside neighborhood



Using SIMD shift to move the keys one step

SIMD Accelerated Hopscotch Hashing

- **SIMD for Probing**
  - Uses SIMD linear probing
  - Probe key within Neighborhood
  - Probe empty space outside
- **SIMD For Insertion**
  - Starts when empty space is found
  - Swap previous values until empty space is inside neighborhood



Using the position the new values are written back