Prepared by Julian García, based on material by
David Albrecht and María García de la Banda

Lecture 31
Collision Resolution II

FIT 1008
Introduction to Computer Science

MONASH University
Information Technology

https://upload.wikimedia.org/wikipedia/commons/5/50/Japanese_car_accident_blur.jpg

Hash Table operations: Insert

Objectives for this lecture

- To understand two of the main methods of conflict resolution:
  - Open addressing:
    - Linear Probing
    - Quadratic probing
    - Double Hashing
  - Separate Chaining

- To understand their advantages and disadvantages

- To be able to implement them

http://blog.mlive.com/annarbornews_impact/2008/12/large_wcc_parking_121708.jpg

Collisions: two main approaches

- Open addressing:
  - Each array position contains a single item
  - Upon collision, use an empty space to store the item (which empty space depends on which technique)

- Separate chaining:
  - Each array position contains <u>a linked list of items</u>
  - Upon collision, the element is added to the linked list

http://2.bp.blogspot.com/-YA-pB_QYUGI/TnbKKJ-eYxI/AAAAAAAAAFA/34OrD2S6xBc/s1600/cbe9caa5_f433bdd9_ae77_4546_95be_776c36fbea07.jpg

```python
def __str__(self):
    result = ""
    for item in self.array:
        if item is not None:
            (key, value) = item
            result += "(" + str(key) + "," + str(value) + ")"
    return result
```

Take the thing there and consider the first part as key and the second as value

Open Addressing: Linear Probing

- Search for an item with hash value N:
  - Perform a linear search from array[N] until either the item or an empty space is found
- But careful, you must deal again with:
  - <u>Full table</u> (to avoid going into an infinite loop)
  - Restarting from position 0 if the end of table is reached.

search(key)

- Get the position N using the hash function, N = hash(key)
- If array[N] is empty return None.
- If there is <u>already an item there</u>:
  - If there is already something there, with the same key return the associated data.
  - If there is already something there with a different key, you need to find the key and return data

__setitem__(self, key, data)

insert or update item (key, data)

__getitem__(self, key)

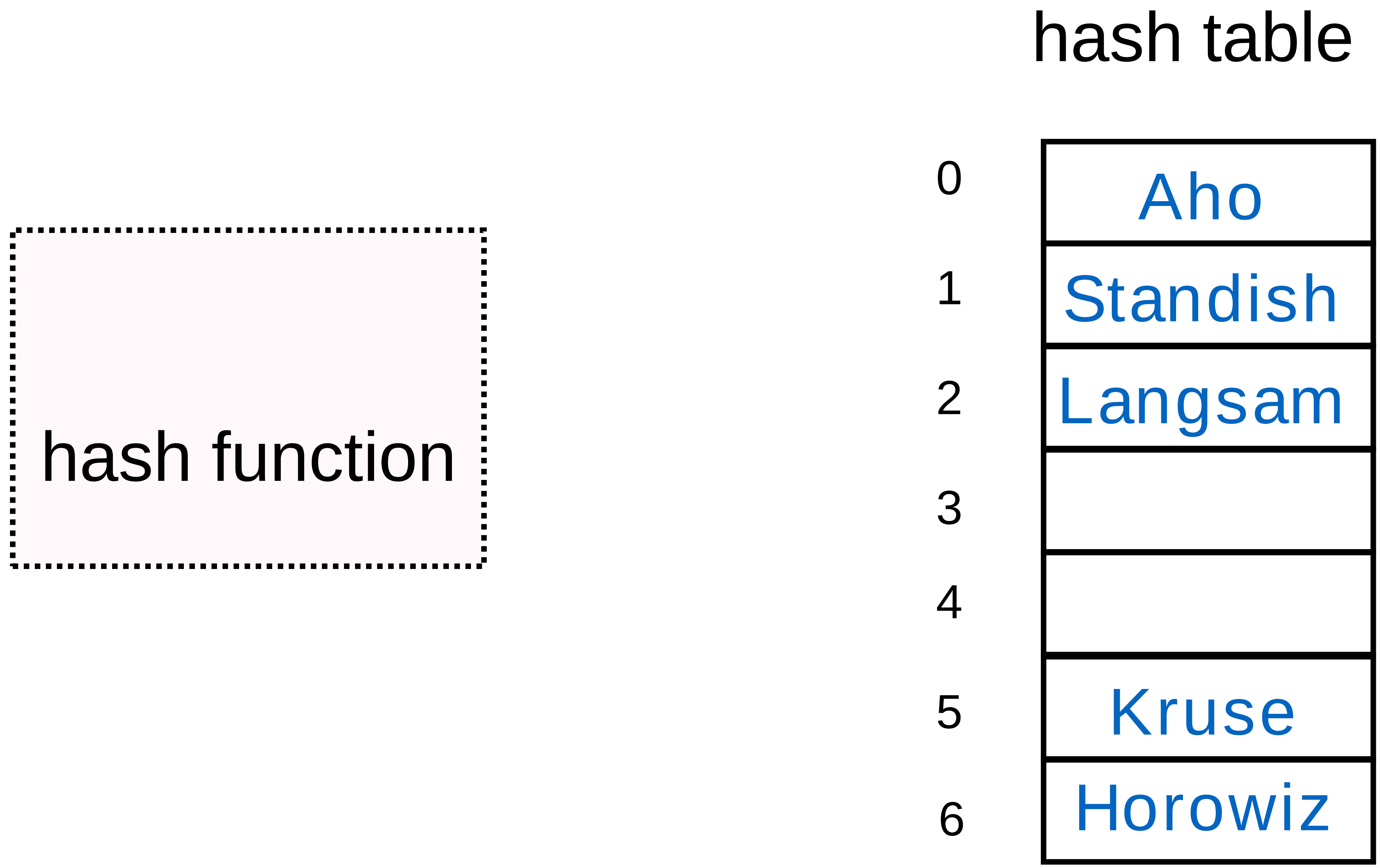
give me the data associated to a key

```python
def __getitem__(self, key):
    position = self.hash(key)
    for i in range(self.table_size):
        if self.array[position] is None:  # found empty slot
            raise KeyError(key)
        elif self.array[position][0] == key:  # found key
            return self.array[position][1]
        else:  # there is something there but different key, try next
            position = (position + 1) % self.table_size
    raise KeyError(key)
```

I showed you this last week but we didn’t use Key Errors

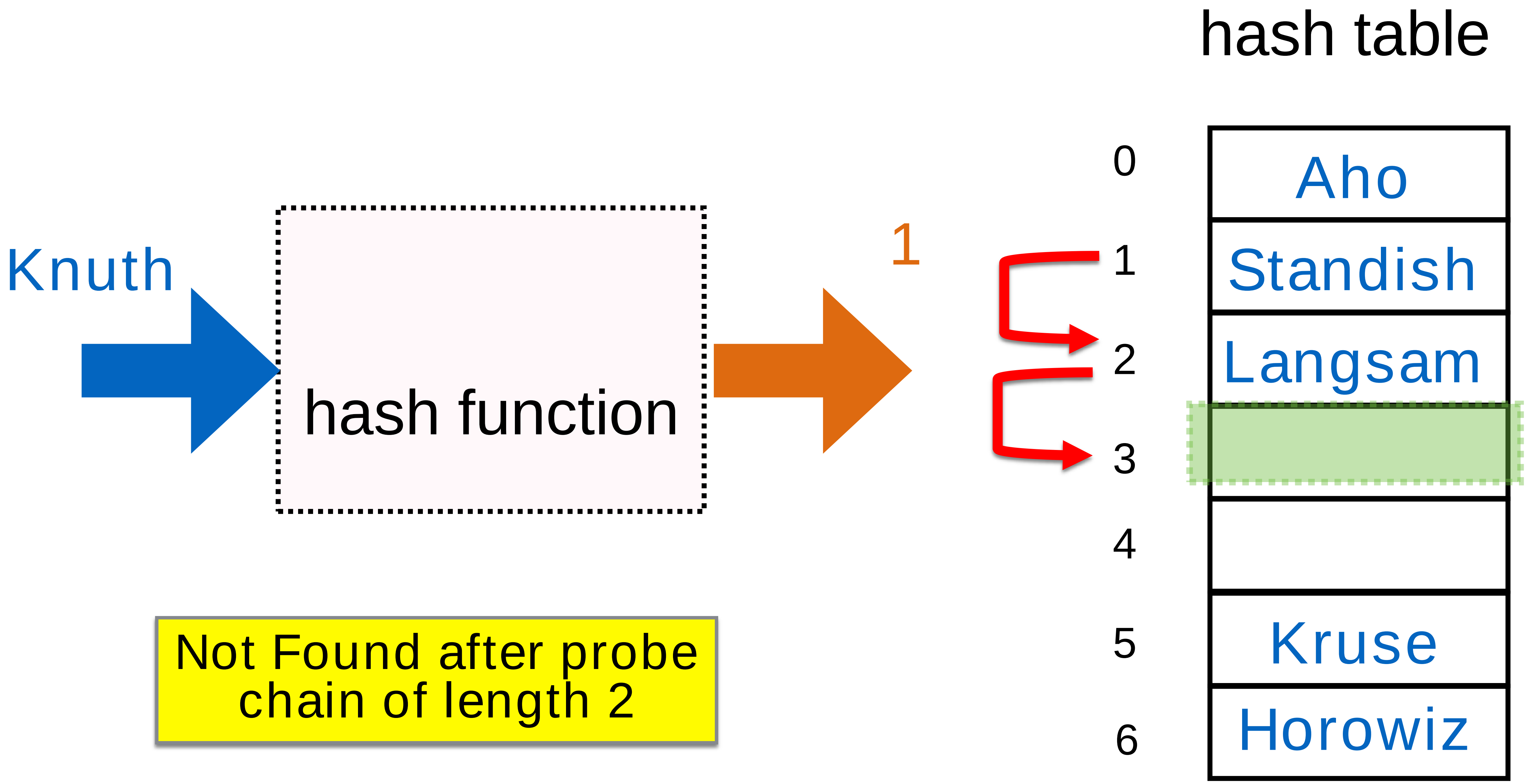
Example: search

key: Langsam



Example: search

key: Knuth



Open Addressing: Linear Probing

- What about delete?
- One possibility:
  - Use the search function to find the item
  - If found at N delete and reinsert every item from N+1 to the first empty position

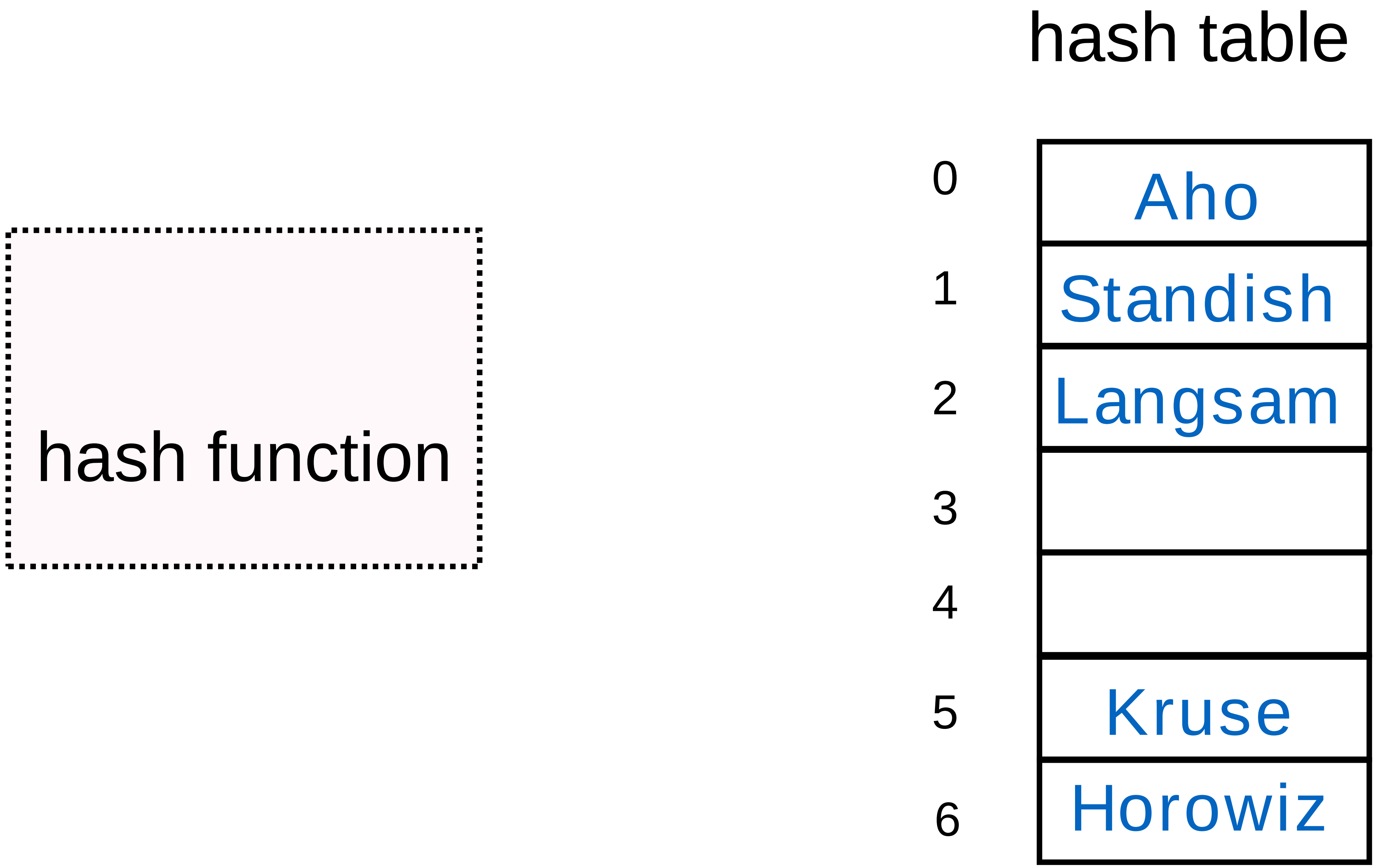
    Time consuming! (though should not be many)

What if I do not reinsert?

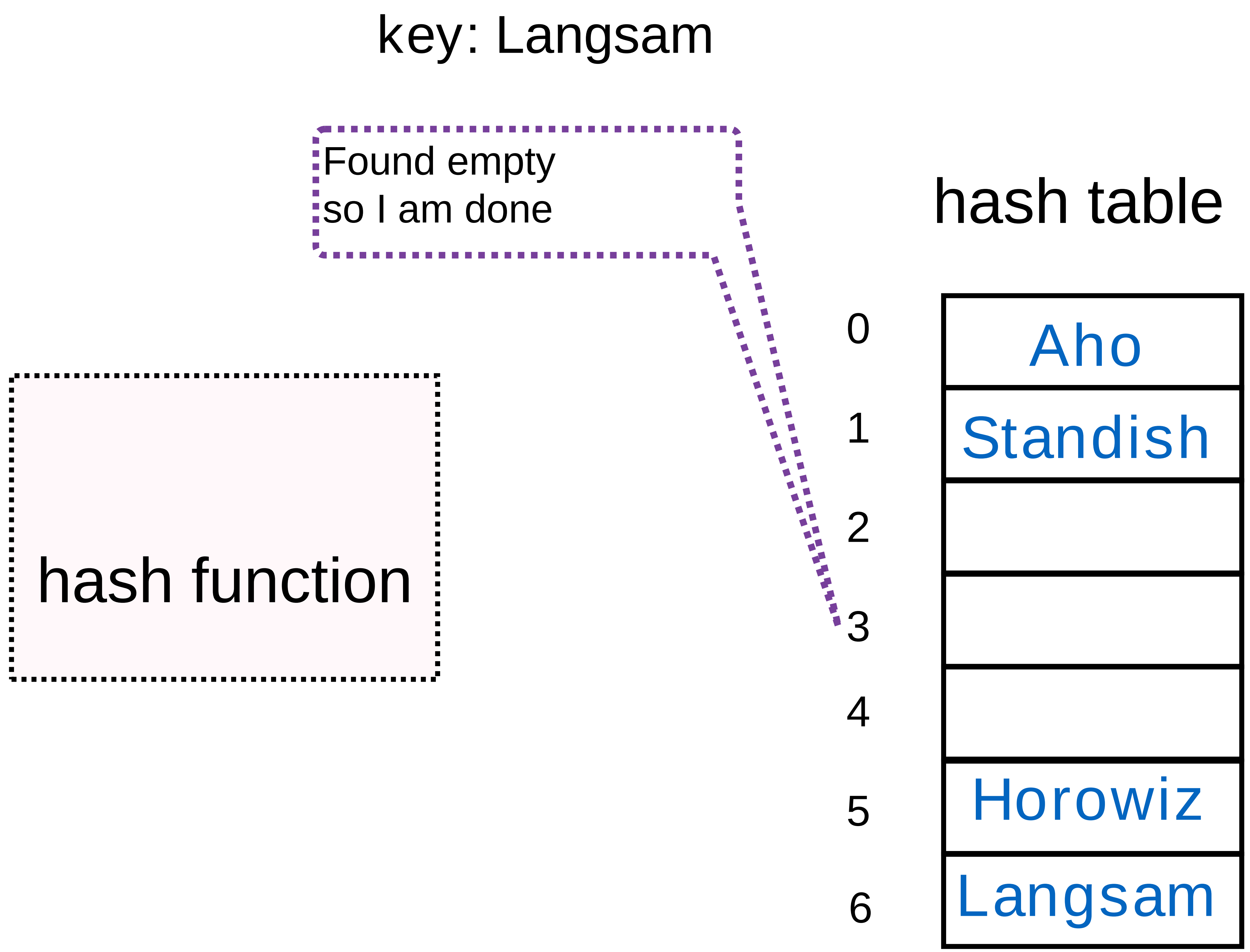
search may incorrectly report some items as not found

Example: delete

key: Kruse



Example: delete



Open Addressing: Linear Probing

- Load factor: total number of items/TABLESIZE

- Cluster: sequence of full hash table slots (i.e., without an empty slot)

- Clusters once formed, tend to grow…
  - Items that hash to a value within the cluster, get inserted at the end making it bigger
  - This might involve more than one hash value

- Cluster can form even when the load is small

Example of cluster

- All 5 elements are part of a cluster
- Langsam, Kruse and Horowiz allhave same hash value (5)
- Aho and Standish have values 0 and 1
- From then on, any element mapped to 0,1,2,5 or 6 will be part of the cluster

hash table

0	Aho
1	Standish
2	Langsam
3	
4	
5	Kruse
6	Horowiz

Linear Probing: Problems

- Tendency for clustering to occur as the load is > 0.5
- Low speed on clustering. We start under-delivering on the promise of constant time search and insert.
- Deletion of records is difficult
- If implemented in arrays – table may become full fairly quickly, resizing is time and resource consuming

Can we reduce clustering by taking bigger and bigger steps?

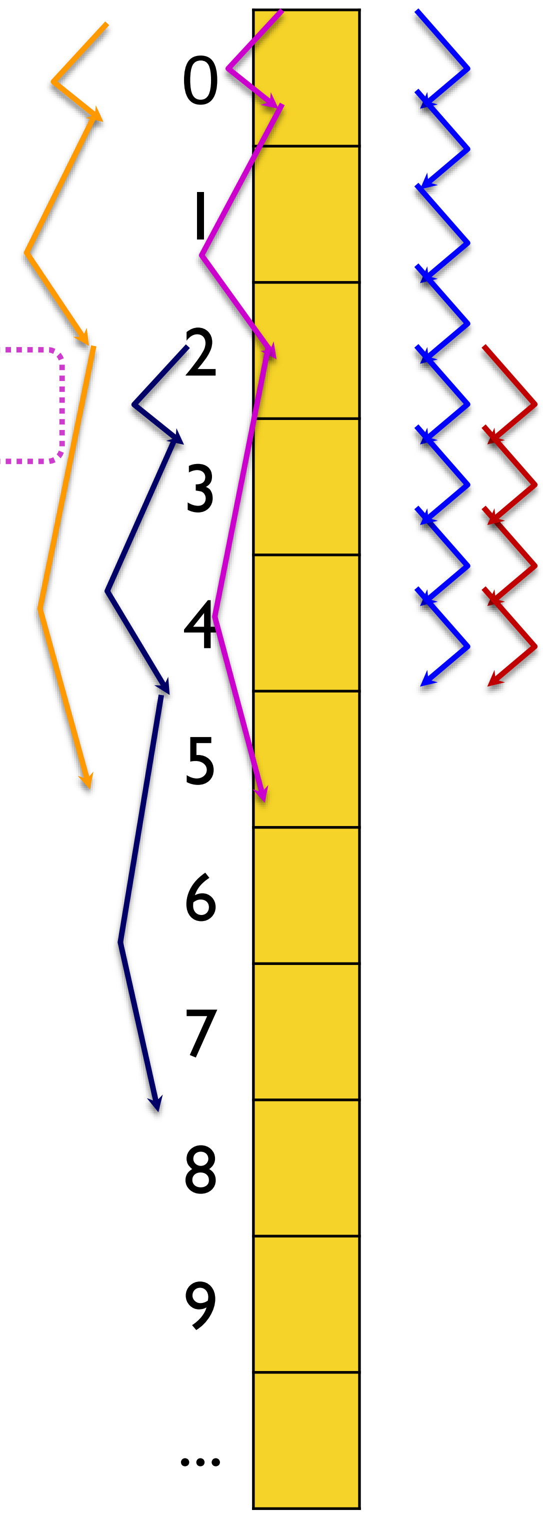
Open Addressing: Quadratic Probing

N+S
S is the step

- Linear probing: search at N+1, N+2, N+3…
- Quadratic probing: search at N+1, N+4, N+9, N+16

$N+S^2$

- primary clustering: keys with different hash values have same probe chains (as in linear probing)
- secondary clustering: keys with same hash values have the same probe chains
- Advantage: Quadratic probing eliminates primary clustering, but can suffer from secondary clustering
- There's a better method: Double Hashing



Open Addressing: Double Hashing

- If a collision occurs, use a second hash function to determine the step.

- Second hash function:
  - Cannot hash to 0
  - Use primes: table size & step size are co-primes (avoid revisiting the same positions)

Greatest Common Denominator = 1

- Eliminates both primary and secondary clustering

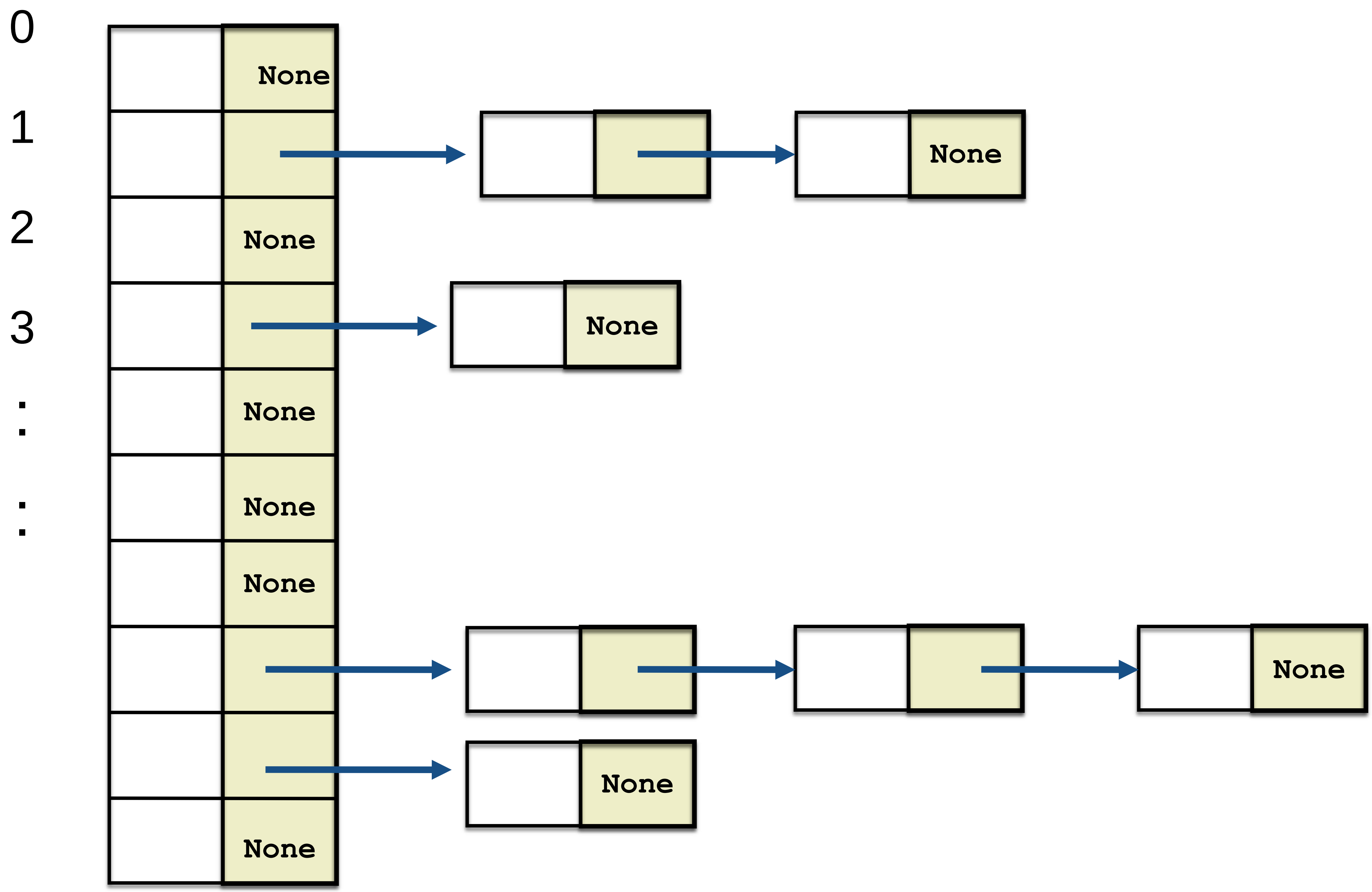
Separate Chaining

Separate Chaining

- Uses a Linked List at each position in the Hash Table.
- Linked list at a position contains all the items that 'hash' to that position.

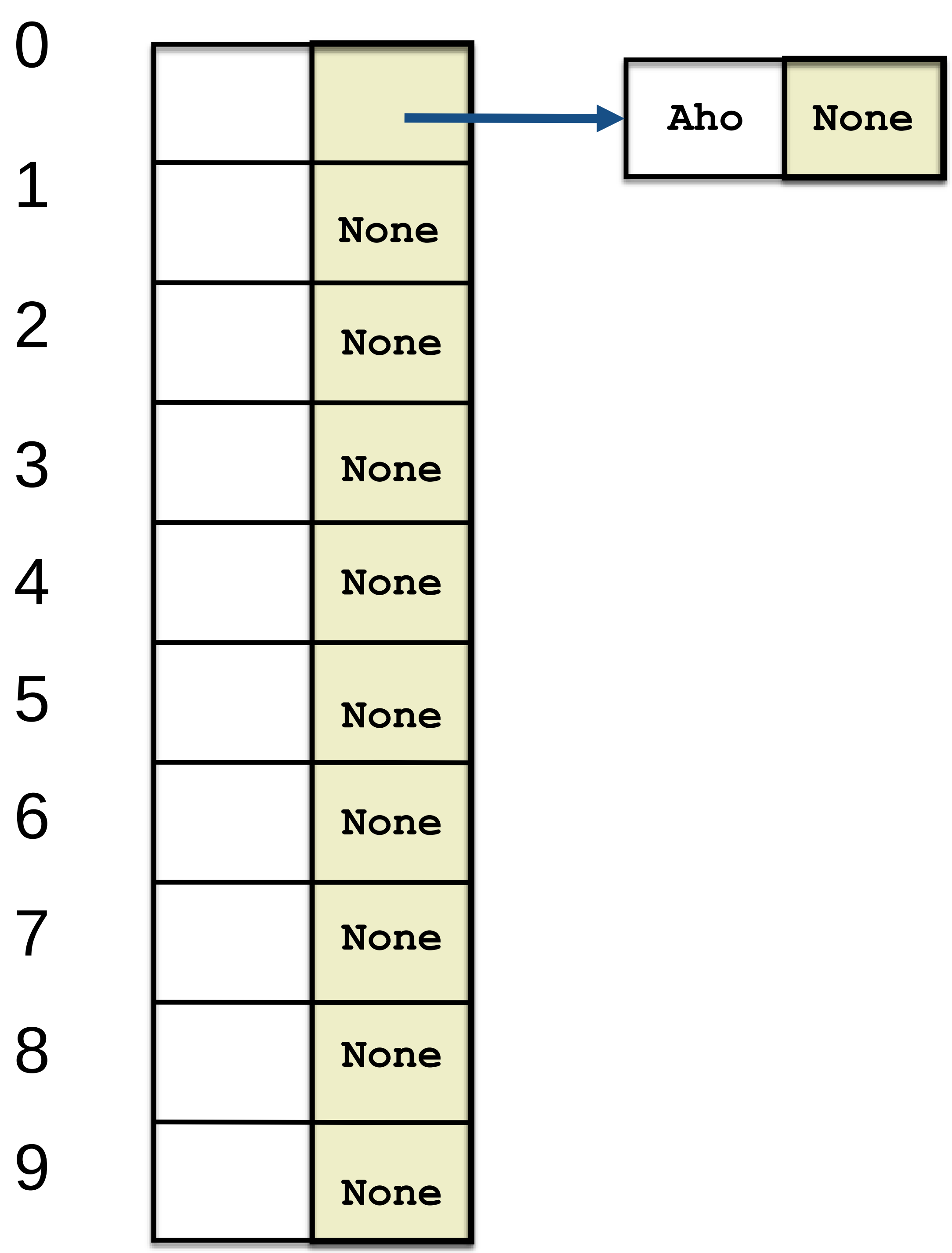
http://2.bp.blogspot.com/-YA-pB_QYUGI/TnbKKJ-eYxI/AAAAAAAAAFA/34QrD2S6xBc/s1600/cbe9caa5_f433bdd9_ae77_4546_95be_776c36fbea07.jpg

hash table



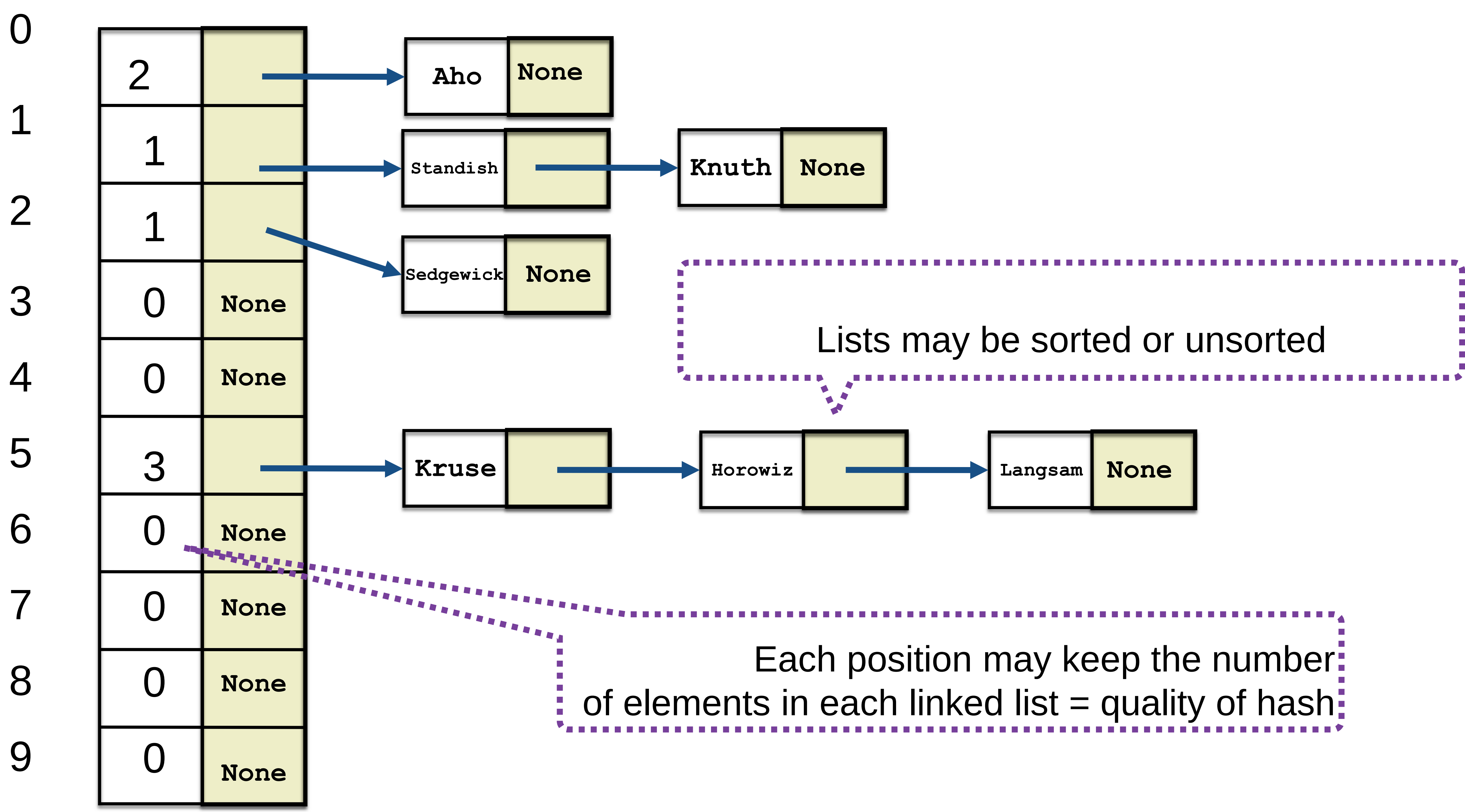
Aho, Kruse, Standish, Horowiz, Langsam, Sedgwick, Knuth

0, 5, 1, 5, 5, 2, 1



Aho, Kruse, Standish, Horowiz, Langsam, Sedgwick, Knuth

0, 5, 1, 5, 5, 2, 1



Separate Chaining

- Apply hash function to get a position N in the array
- Insert: Insert key into the Linked List at position N
- Search: Search for key in the Linked List at position N
- Delete: Search for key; delete the node in the Linked List at position N

Separate Chaining

Advantages:

- Conceptually simpler
- Insertions and deletions are easy and quick
- Naturally resizable, allows a varying number of records to be stored

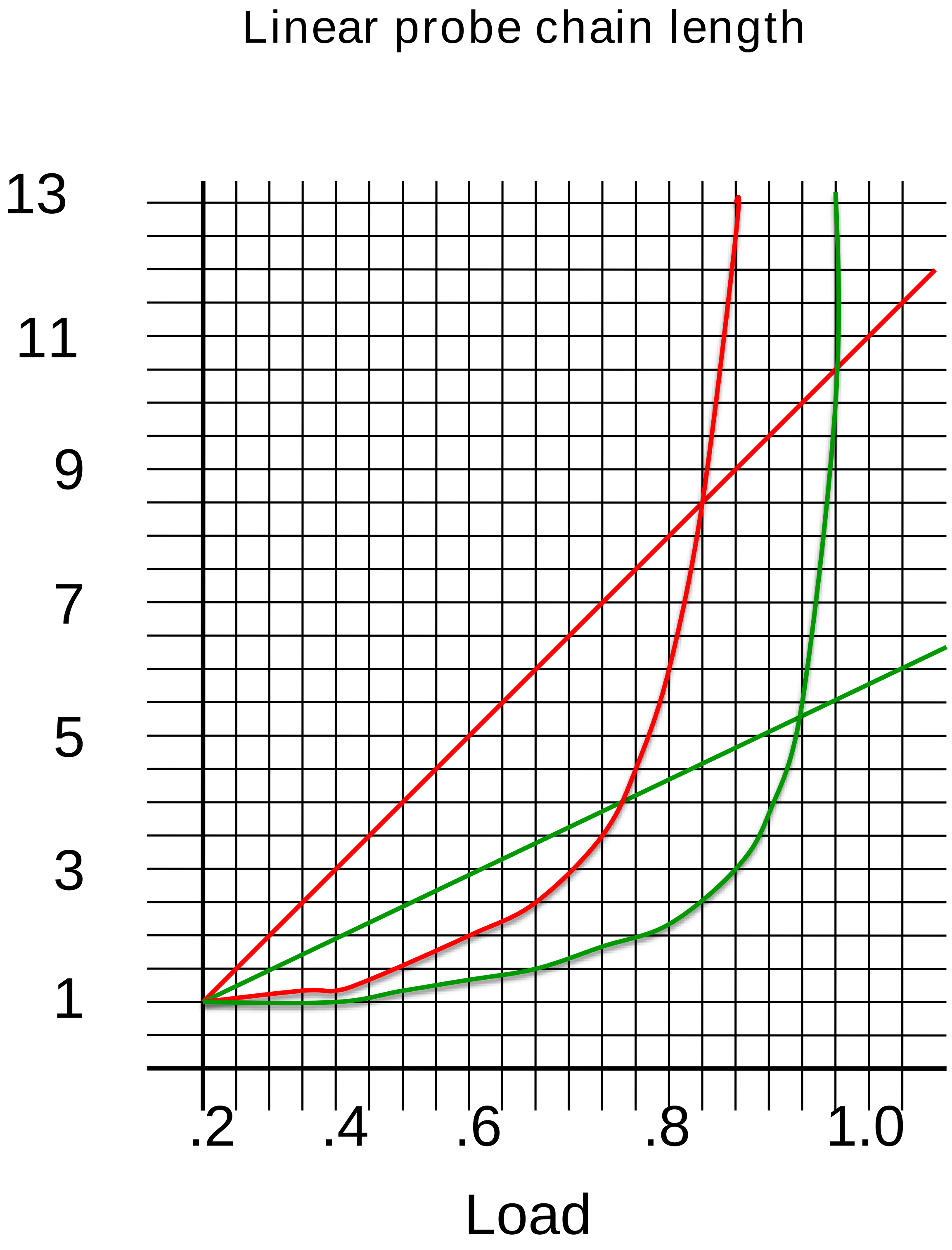
Disadvantages

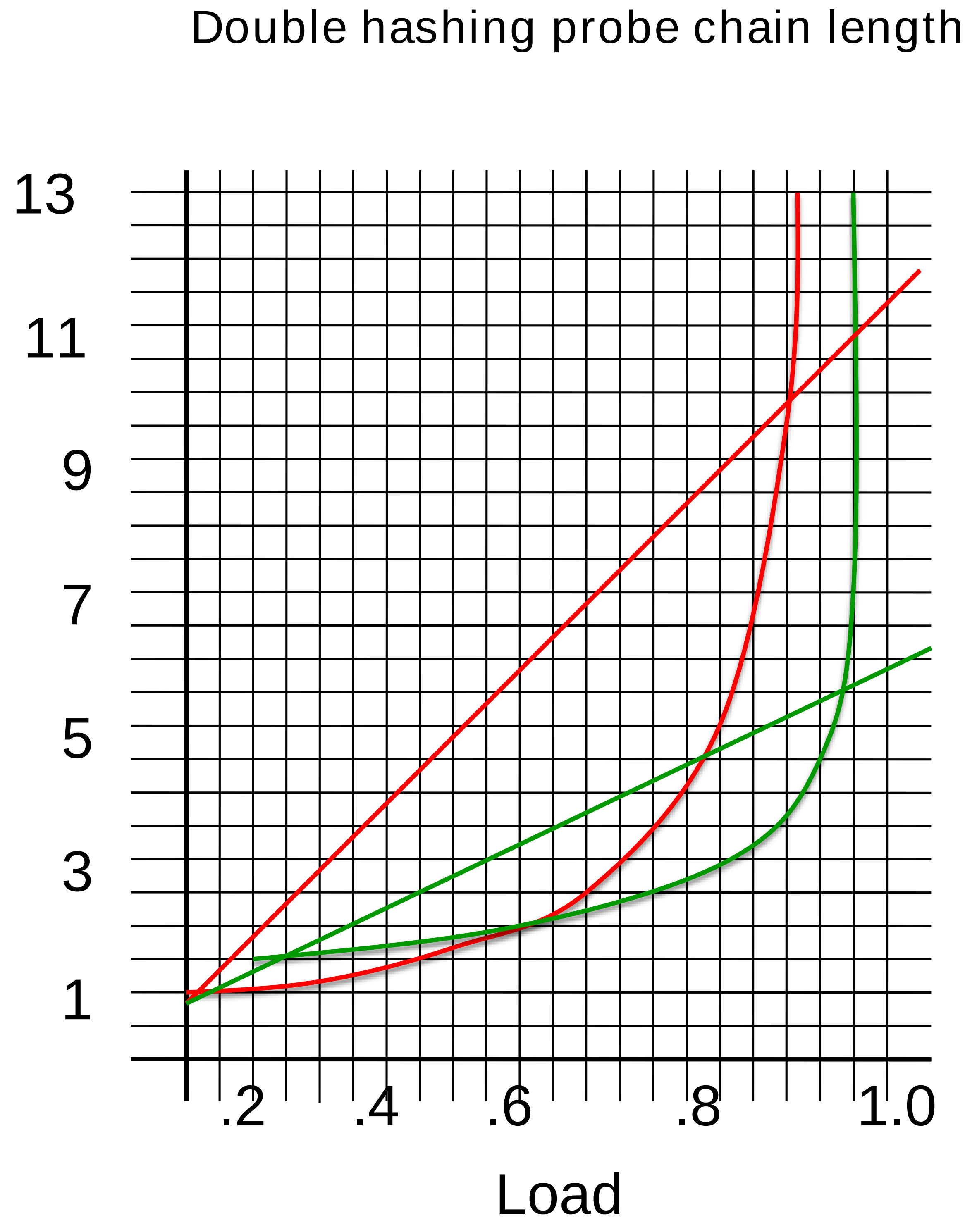
- Requires extra space for the links
- Requires linear search for elements in a list

Comparison: general

- Choice depends on the particular application
  - Linear Probing: fast if memory allows for a large table.
  - Double hashing: efficient use of memory but needs to compute a second hash
  - Separate chaining: simple, extra memory, resizable, fast insert and fast delete
- When the load approaches 1: double hashing far outperforms linear probing
- Open addressing: keep load under 2/3 …. even better 1/2
- Separate chaining: efficiency degrades linearly with load

Superimposing Separate Chaining





found/not found

Dynamic Hashing

- Each time the load in an open address method gets greater than desirable
  - 1/2 for linear probing
  - 2/3 for double hashing

  we expand the table by doubling its size

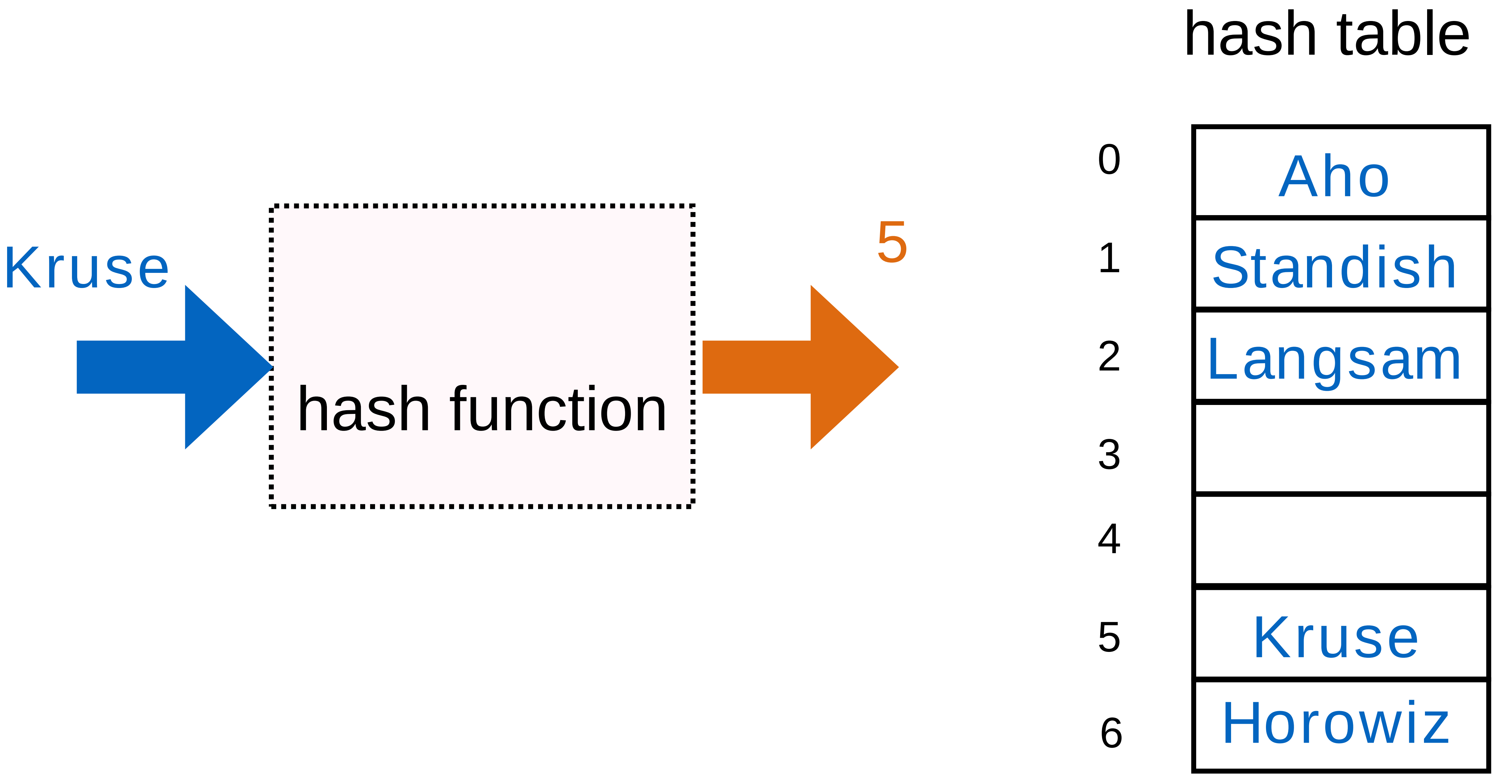
- This requires
  - Creating the new array
  - Rehashing every item in the old table into the new one (due to the use of TABLESIZE in the hash function)

Other ways to delete

- Lazy deletion
- When you remove something, mark it as gone
- Treat these as empty for insertion, and wrong key for search

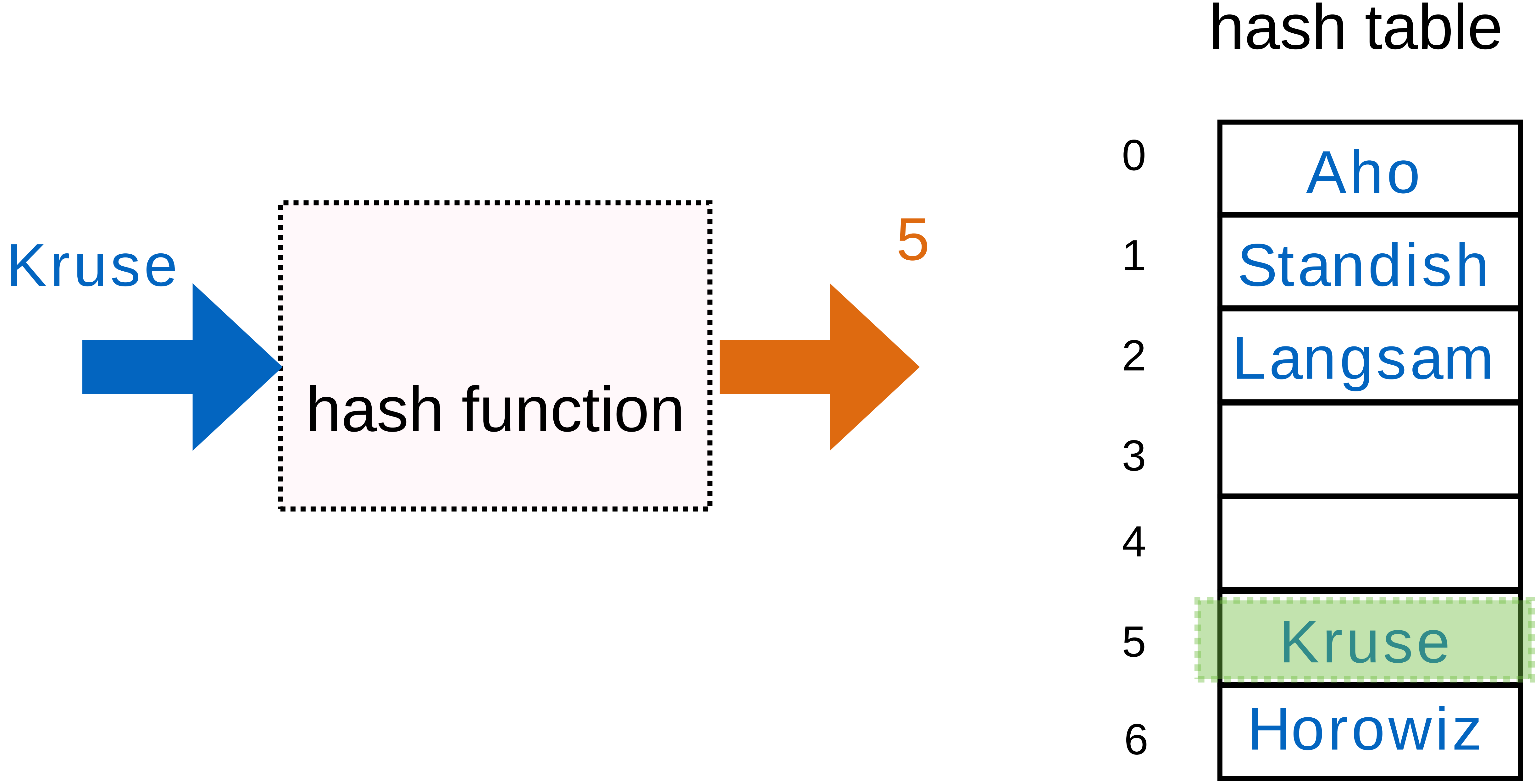
Example: delete (lazy)

key: Kruse



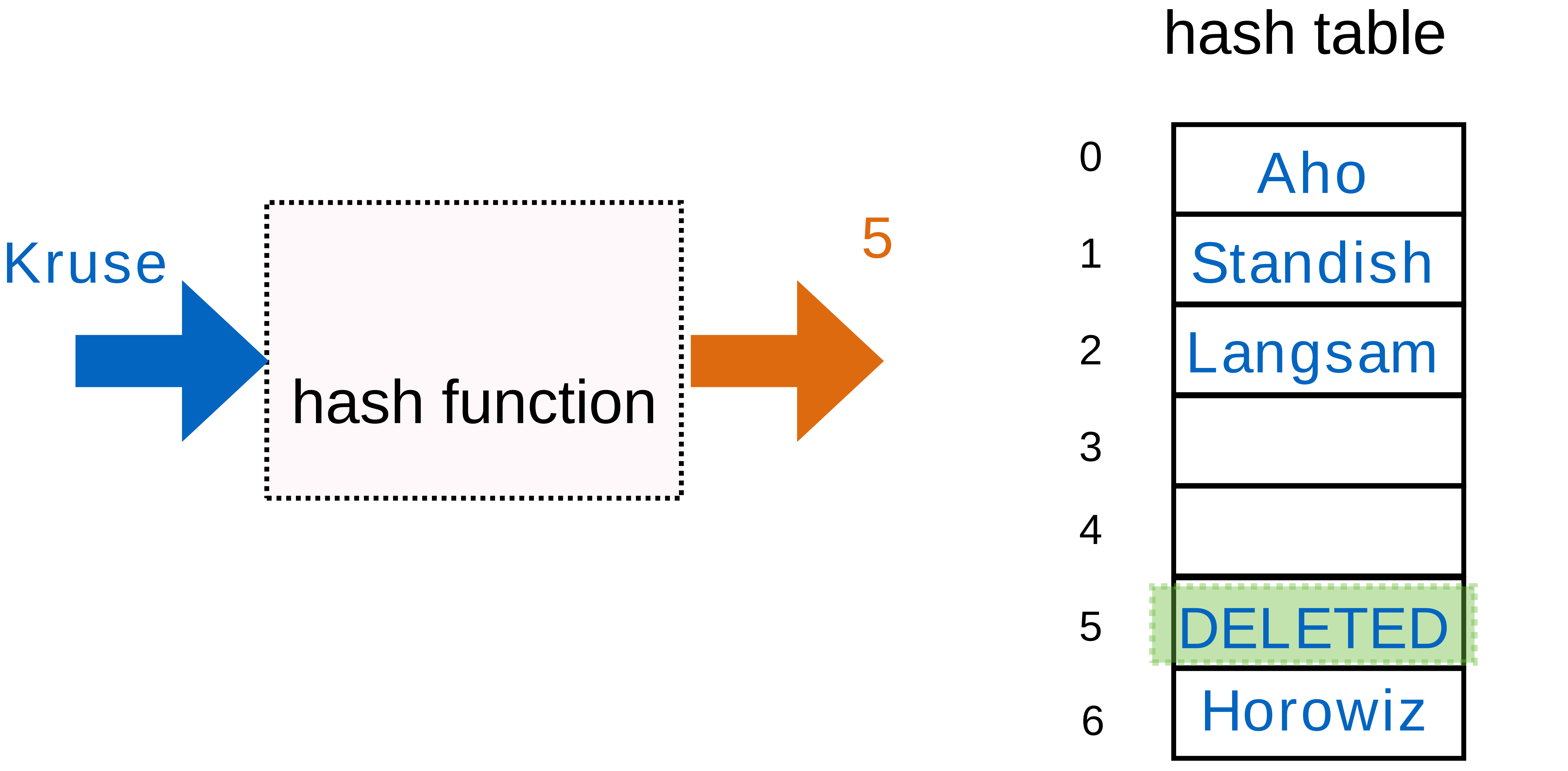
Example: delete (lazy)

key: Kruse



Example: delete (lazy)

key: Kruse

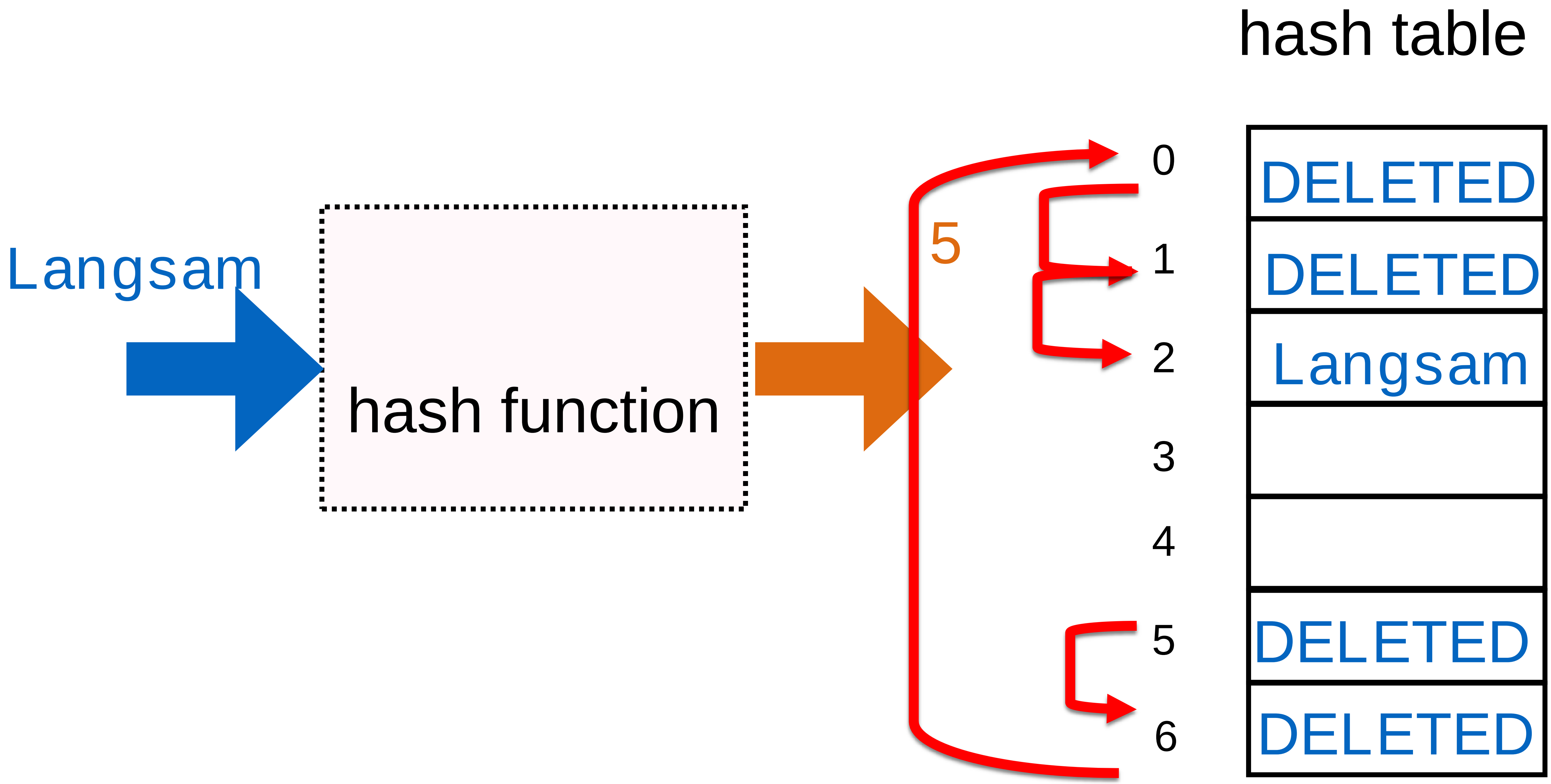


Consequences of lazy deletion

- When inserting, I need to search to the end of the probe chain to confirm it isn’t there
- But I get to insert at the first deleted if it’s not
- Searching in general means quite long chains if I do many deletions

searching after lazy deletion

key: Langsam



Conclusion

- Hash Tables are one of the most used data types
- You have a very good chance of using them in your career
- Choice of hash function, collision handling and load factor are crucial to maintaining an efficient hash table
- They are very simple conceptually:
  - A significant amount of experimental evaluation is usually needed to fine tune the hash function and the TABLESIZE