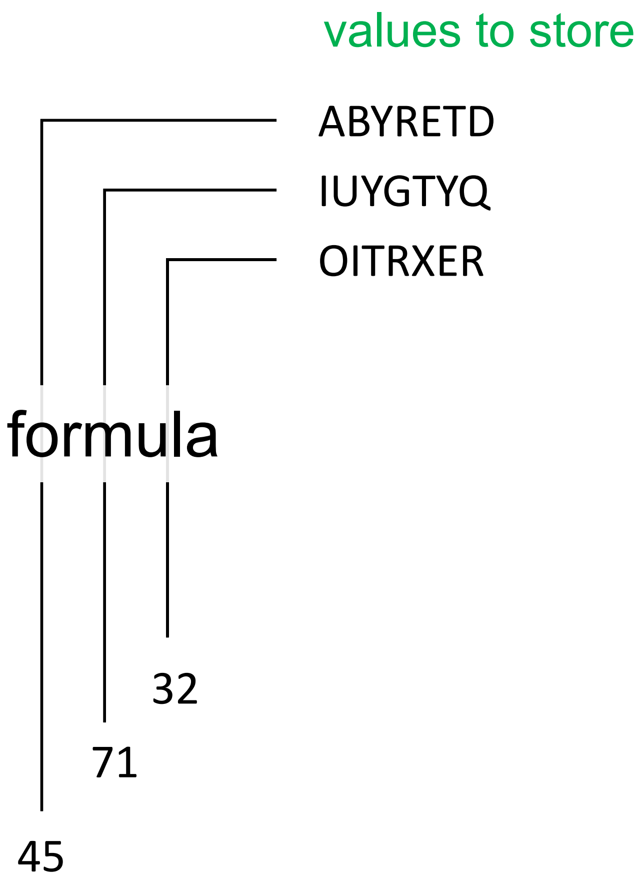
DS:
Hashing

Liwei

What is Hashing

- Hashing is a method of sorting and indexing data.
- The idea behind hashing is to allow large amounts of data to be indexed using keys commonly created by formulas.



values to store

ABYRETD

IUYGTYQ

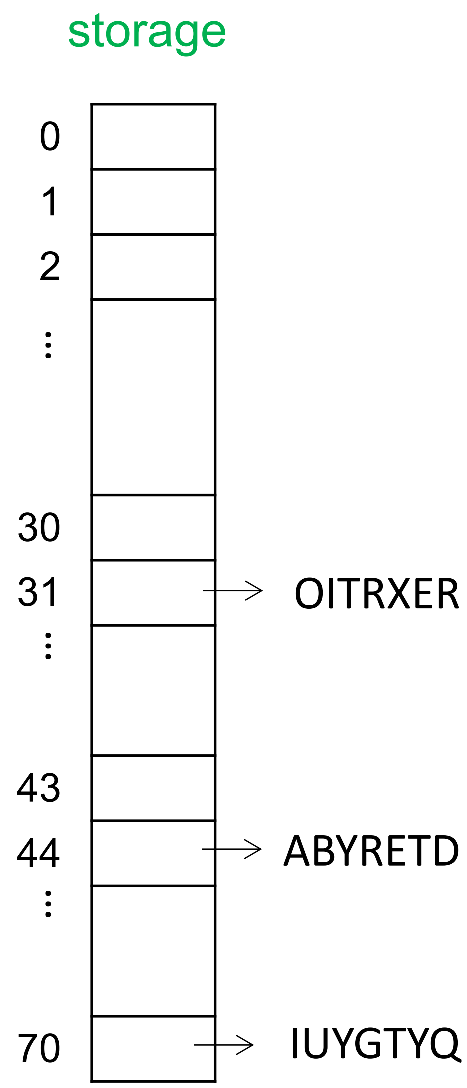
OITRXER

formula

32

71

45



storage

0

1

2

⋮

30

31 → OITRXER

⋮

43

44 → ABYRETD

⋮

70 → IUYGTYQ

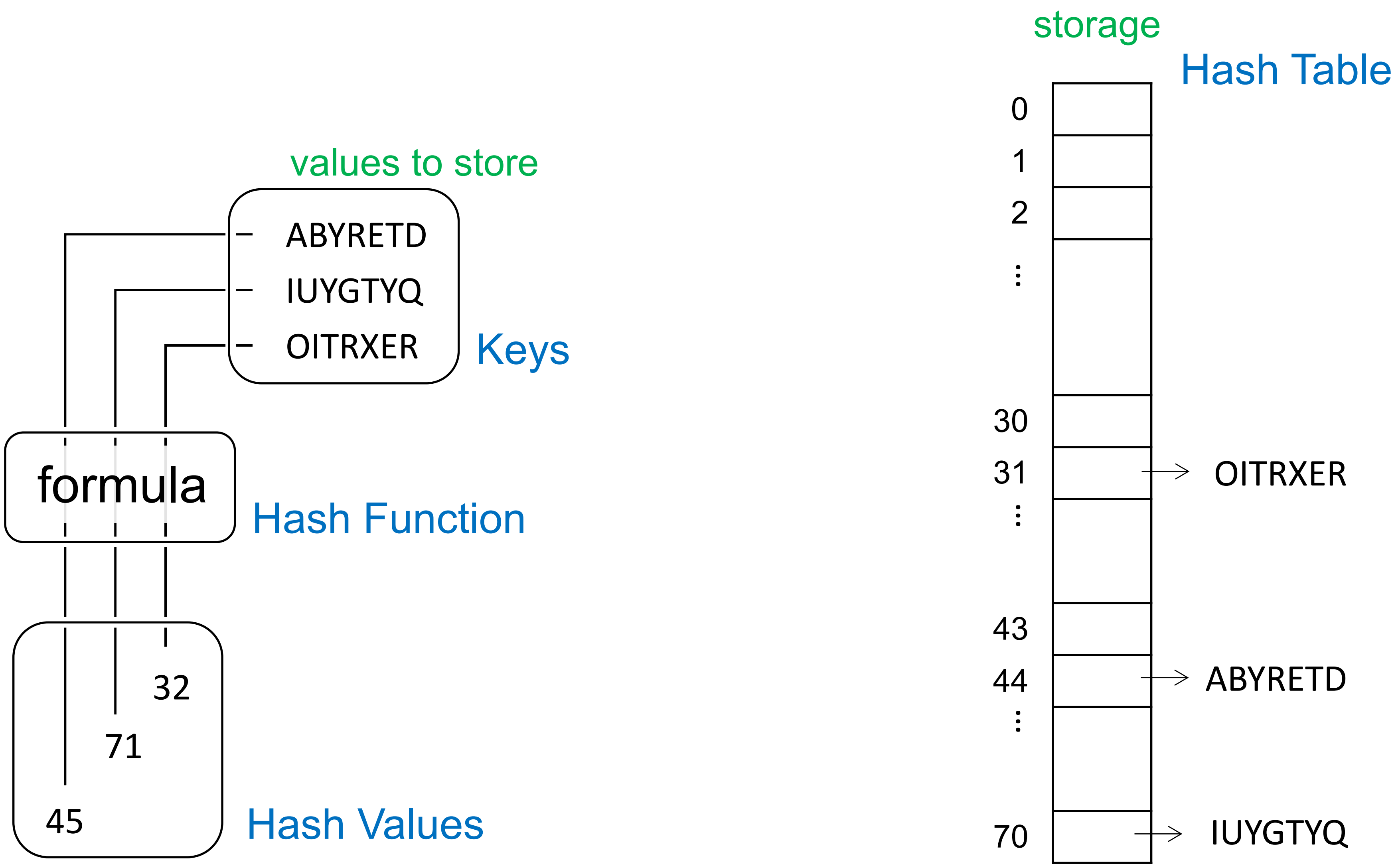
Why we need Hashing

- Time efficiency

Data Structures	Time Complexity for search operation
Array	O(log n)
Linked List	O(n)
Tree	O(log n)
Hashing	O(1) / O(n)

Terminologies

- **Hash Function:** A hash function is any function that can be used to map data of arbitrary size to data of fixed size.
- **Key:** input data given by user
- **Hash value:** The values returned by a hash function are called hash values, hash codes, digests, or simply hashes.
- **Hash Table:** it is a data structure which implements an associative array abstract data type, a structure that can map keys to values.
- **Collision:** a collision occurs when two different key to a hash function produce the same output called hash value. ???



values to store

ABYRETD

IUYGTYQ

OITRXER

Keys

formula

Hash Function

32

71

45

Hash Values

storage

Hash Table

0

1

2

⋮

30

31 → OITRXER

⋮

43

44 → ABYRETD

⋮

70 → IUYGTYQ

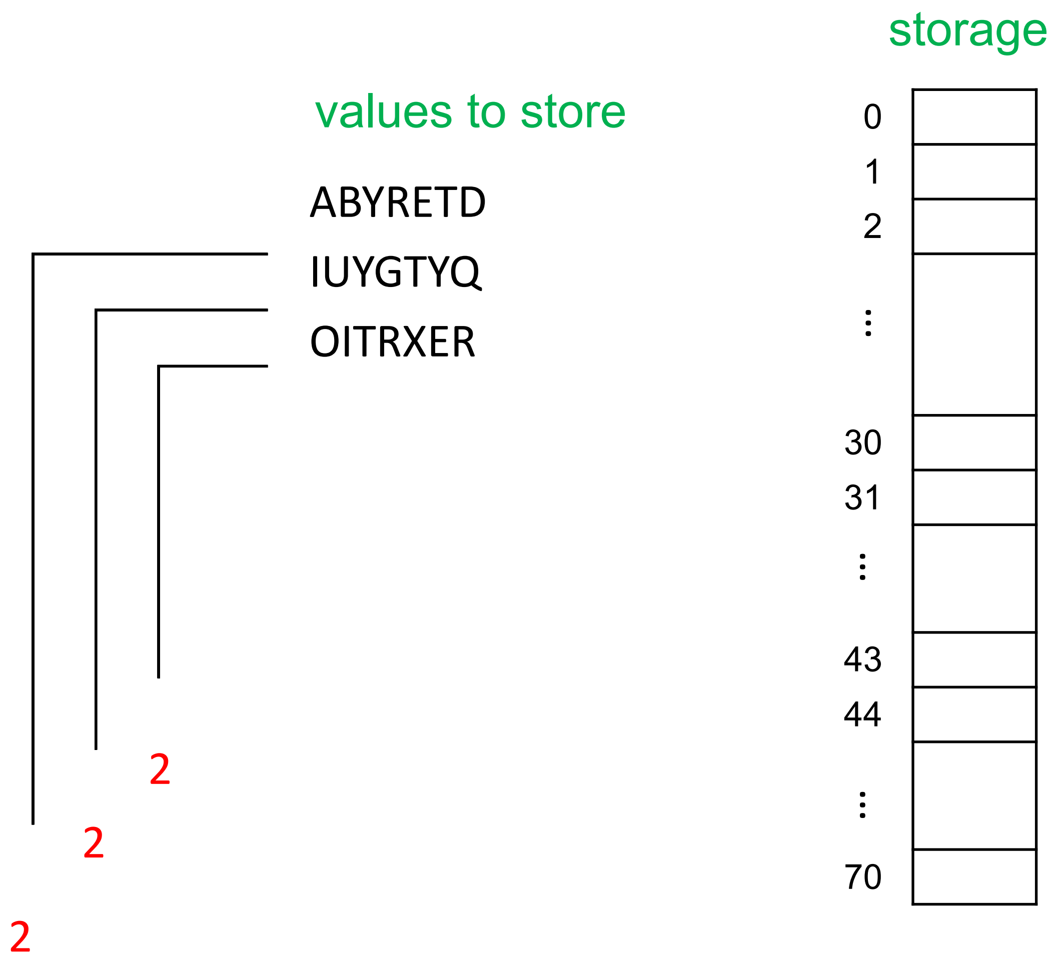
Sample good Hash function

- Simple Mod Function (for integer inputs)

Characteristics of good Hash Function

- It distributes hash values uniformly across the hash table
- The hash function uses all the input data

Collision Resolution Techniques



values to store

ABYRETD

IUYGTYQ

OITRXER

2

2

2

storage

0

1

2

⋮

30

31

⋮

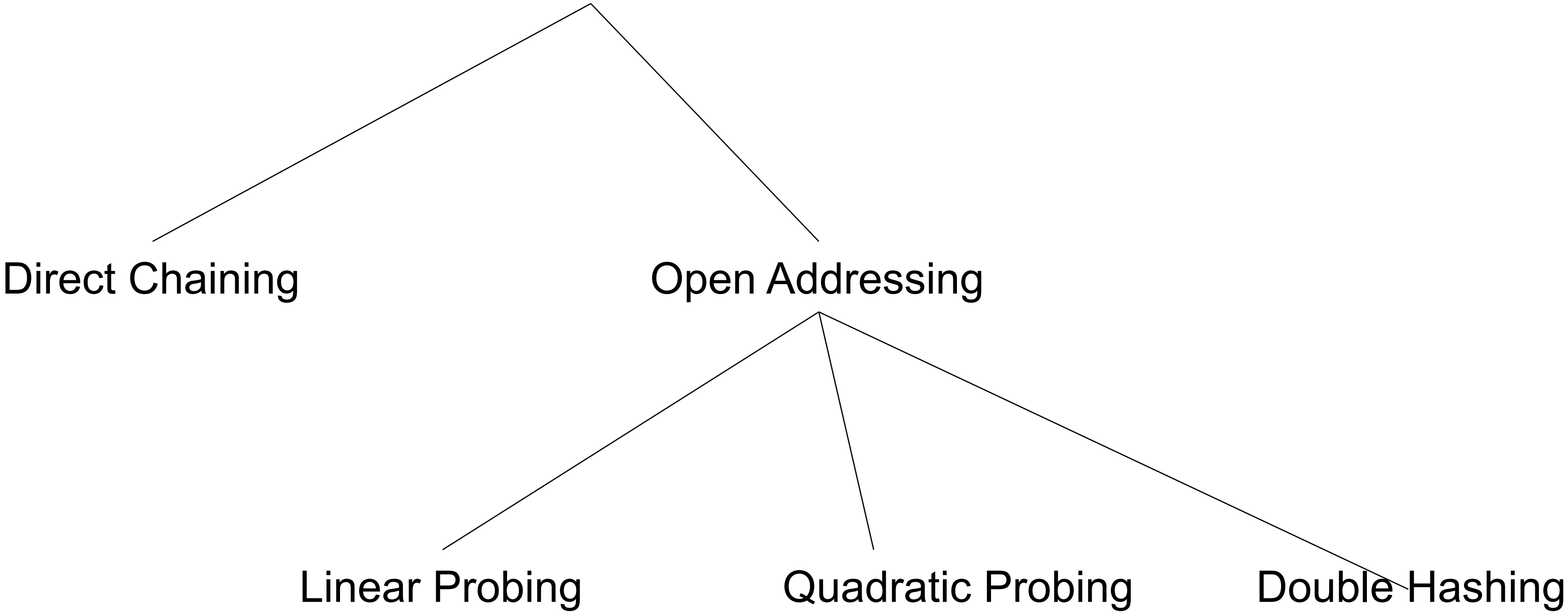
43

44

⋮

70

Collision Resolution Techniques



Direct Chaining

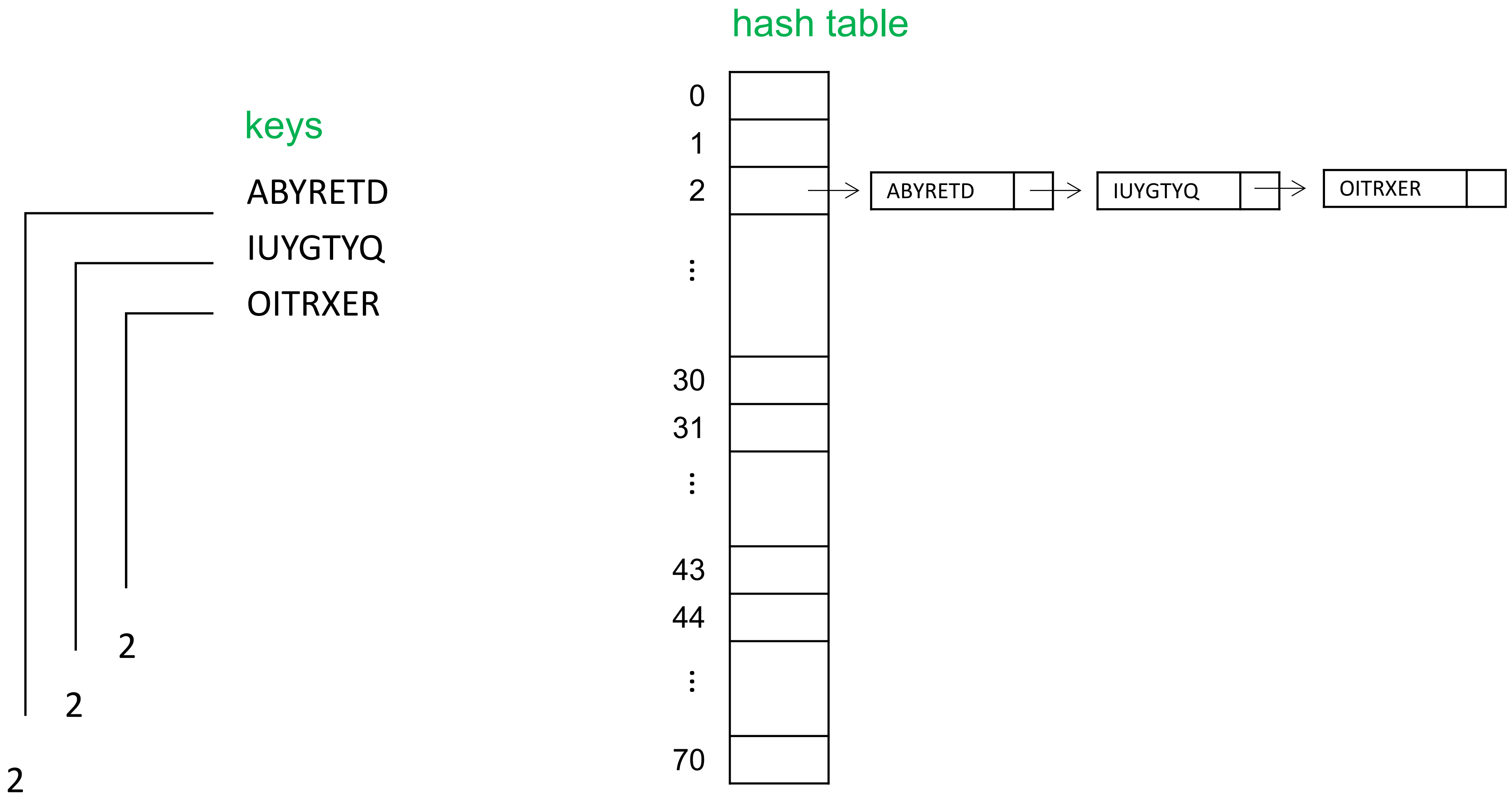
Open Addressing

Linear Probing

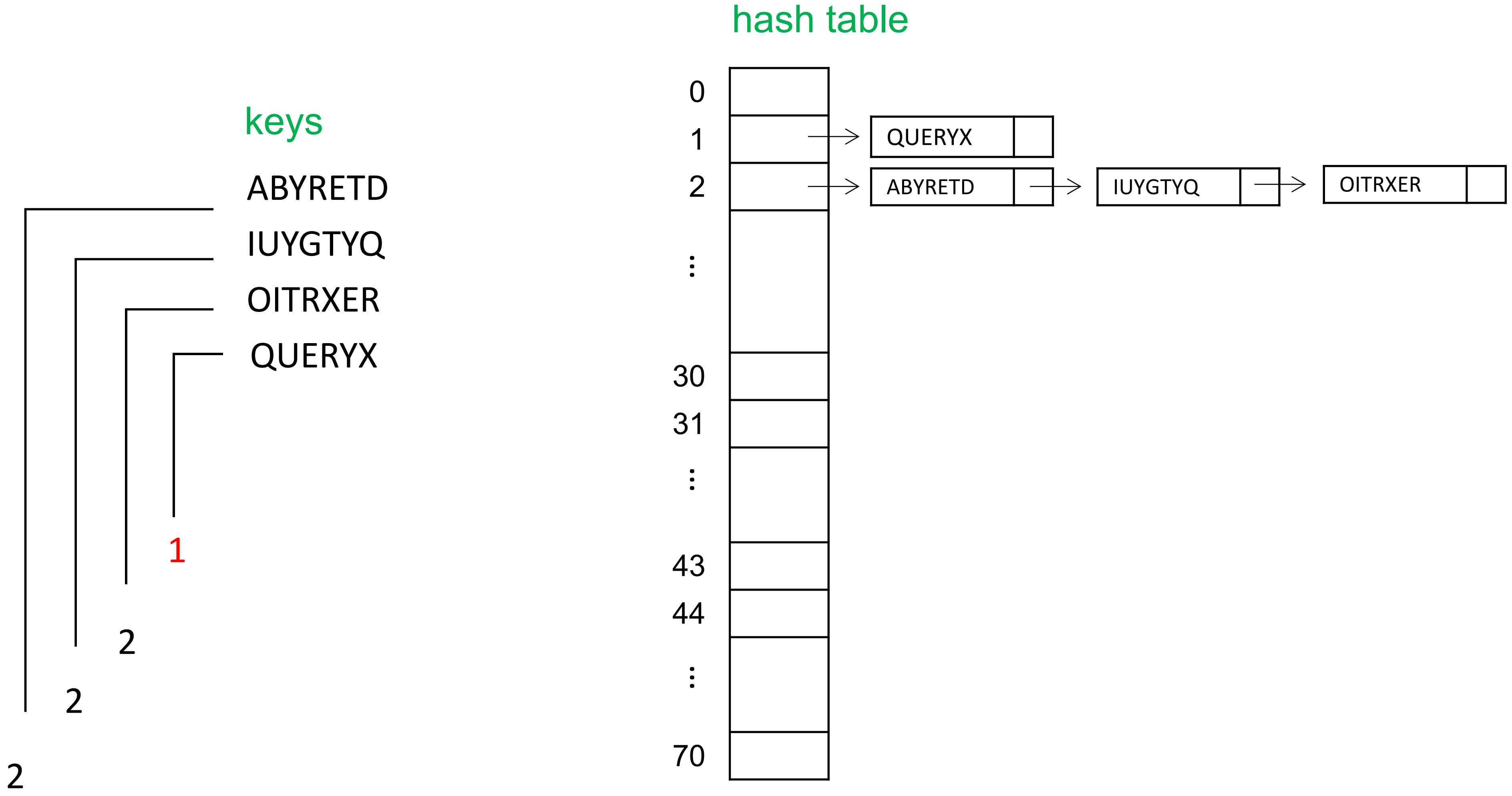
Quadratic Probing

Double Hashing

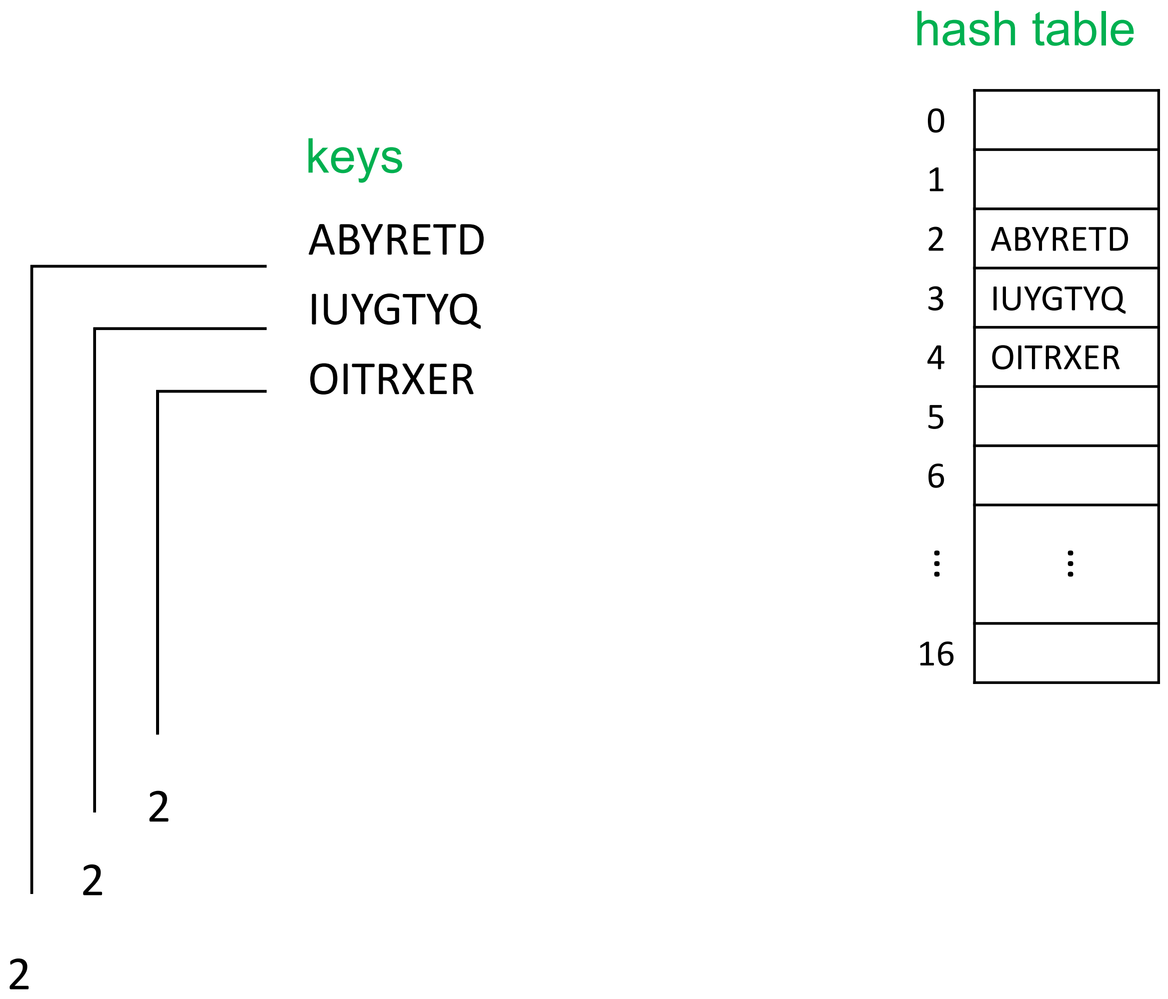
Direct Chaining Technique



Direct Chaining Technique



Linear Probing



keys

ABYRETD

IUYGTYQ

OITRXER

2

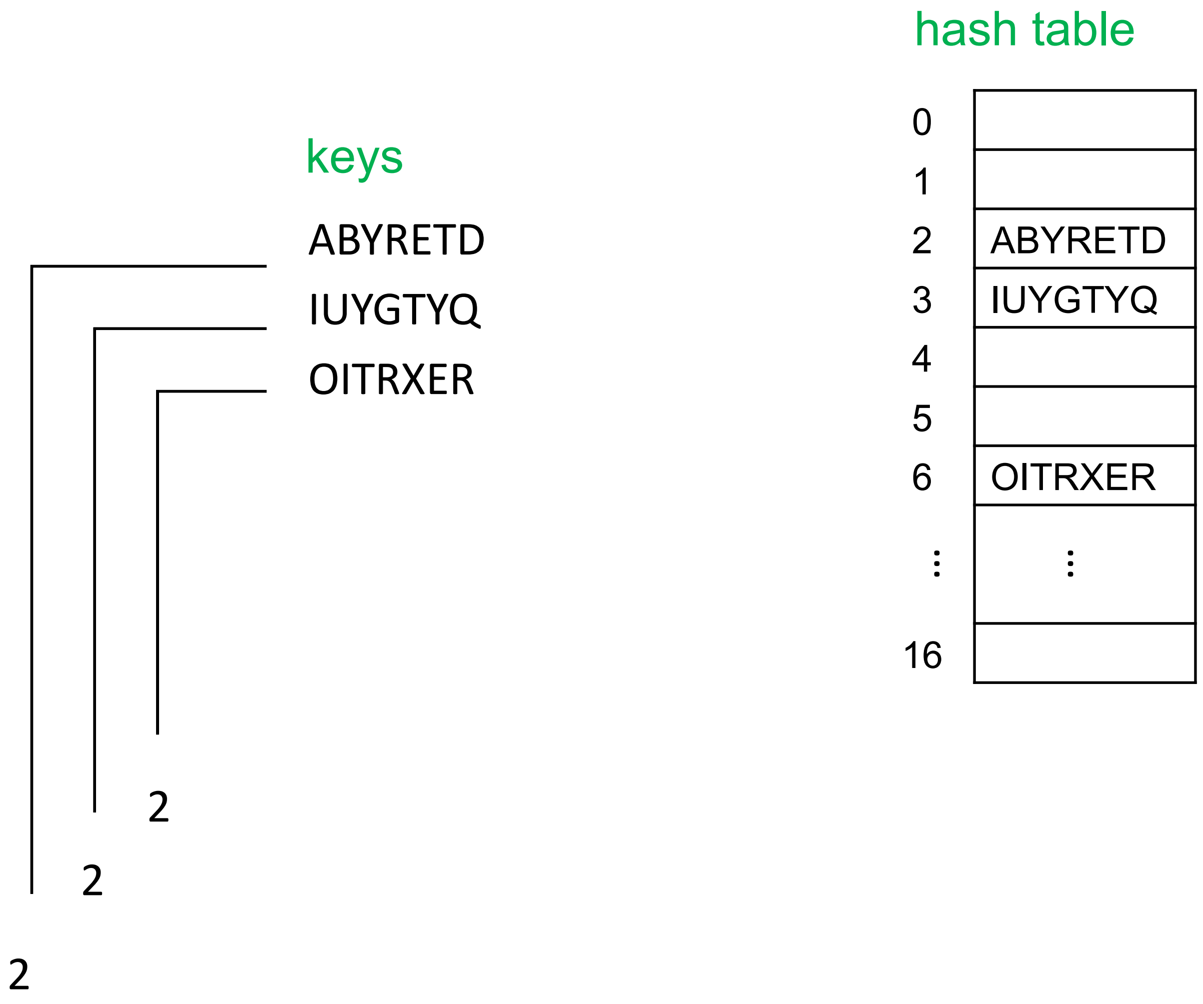
2

2

hash table

0	
1	
2	ABYRETD
3	IUYGTYQ
4	OITRXER
5	
6	
⋮	⋮
16	

Quadratic Probing



keys

ABYRETD

IUYGTYQ

OITRXER

2

2

2

hash table

0	
1	
2	ABYRETD
3	IUYGTYQ
4	
5	
6	OITRXER
⋮	⋮
16	

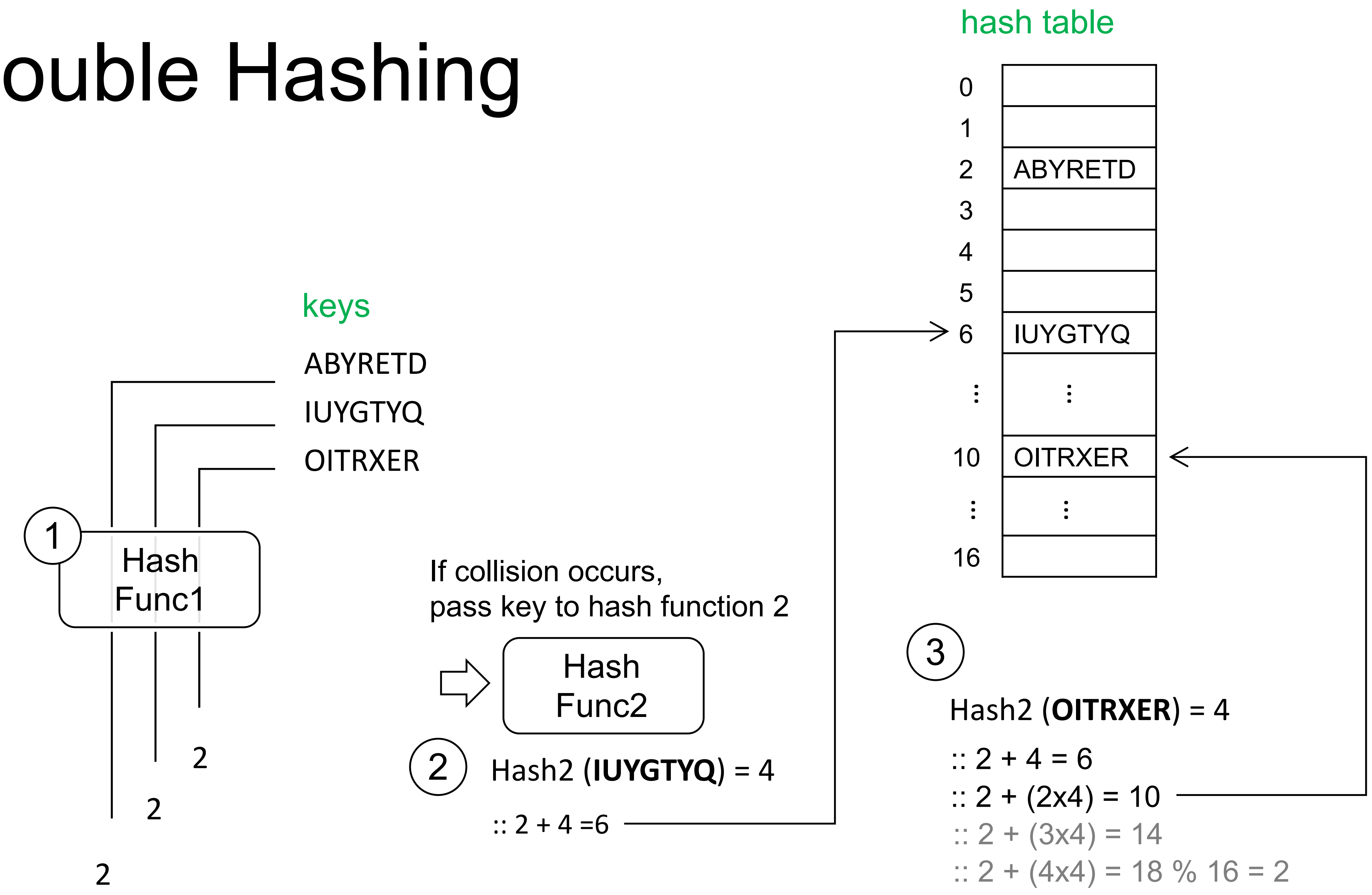
1^2, 2^2, 3^2, 4^2

$2+1^2 = 3$

$2+2^2 = 6$

$2+3^2 = 11$

Double Hashing



keys

ABYRETD

IUYGTYQ

OITRXER

1 Hash Func1

2

2

2

If collision occurs,
pass key to hash function 2

Hash Func2

2 Hash2 (**IUYGTYQ**) = 4

:: 2 + 4 =6

hash table

0	
1	
2	ABYRETD
3	
4	
5	
6	IUYGTYQ
⋮	⋮
10	OITRXER
⋮	⋮
16	

3

Hash2 (**OITRXER**) = 4

:: 2 + 4 = 6

:: 2 + (2x4) = 10

:: 2 + (3x4) = 14

:: 2 + (4x4) = 18 % 16 = 2

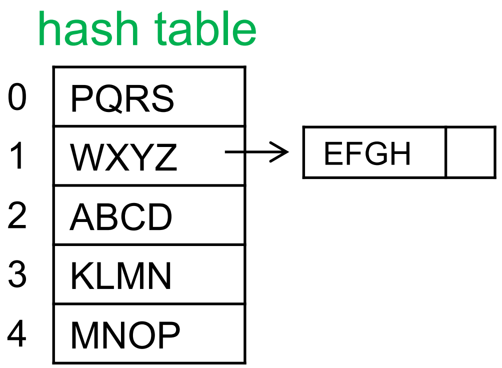
- **Direct Chaining:**
  - implements the buckets as linked lists. Colliding elements are stored in these lists.
- **Open Addressing:**
  - colliding elements are stored in other vacant buckets. During storage and lookup, these are found through so-called “probing”
  - **Linear Probing:**
    - is a strategy for resolving collisions, by placing the new key into the closest following empty cell.
  - **Quadratic Probing:**
    - operates by taking the original hash index and adding successive values of an arbitrary quadratic polynormial until an open slot is found.
  - **Double Hashing:**
    - interval between probes is computed by another hash function.

What happens when Hash Table is full?

- **Direct Chaining:**
  - This situation will never arise.
- **Open Addressing:**
  - Need to create 2x size of current Hash Table and redo Hashing for existing keys

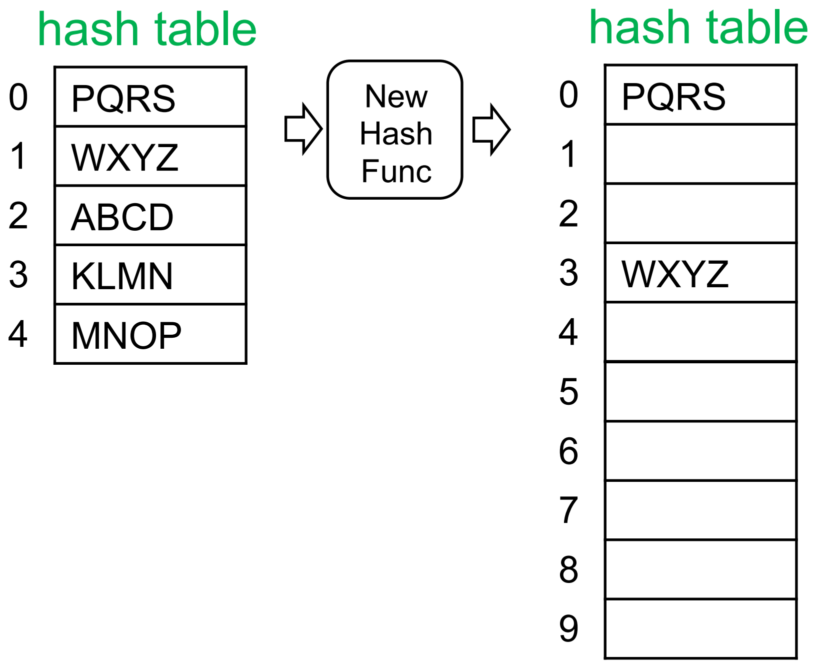
In Cases of Direct Chaining

keys	hash values
ABCD	→ 2
PQRS	→ 0
MNOP	→ 4
WXYZ	→ 1
KLMN	→ 3
EFGH	→ 1



In Cases of Open Addressing

keys	hash values
ABCD	→ 2
PQRS	→ 0
MNOP	→ 4
WXYZ	→ 1
KLMN	→ 3
EFGH	→ 1



hash table

0	PQRS
1	WXYZ
2	ABCD
3	KLMN
4	MNOP

hash table

0	PQRS
1	
2	
3	WXYZ
4	
5	
6	
7	
8	
9	

Pros &Cons of Collision Resolution Techniques

- **Direct Chaining:**
  - No fear of exhausting Hash Table buckets
  - Fear of big Linked List (can effect performance big-O time).
- **Open Addressing:**
  - Easy implementation
  - Fear of exhausting Hash Table buckets

- If input size is known then always use "Open Addressing", else can use any of the two.
- If frequency of deletion is high, then we should always go for "Direct Chaining".

In Cases of Open Addressing

keys	hash values
WXYZ	→ 1
PQRS	→ 0
MNOP	→ 4
ABCD	→ 1
KLMN	→ 1

	hash table
0	PQRS
1	WXYZ
2	~~ABCD~~
3	KLMN
4	MNOP

← Deletion creates to “holes” in the hash table,
which can impact time complexity.

one may think KLMN is not in the table when following
linear probe strategy where this hole stops probing further.

In this case, we do “restructure”,
which is to rebuild the hash table from scratch,
so “the holes“ are filled.

Personal computer

Username: liwei@abc.com
Password: 123456

Facebook server

1. Save the password as it is. i.e 123456
2. Convert the Key (e.g., password) into hash value and save the hash value instead of password. Say ruh67#87Fg6hye@%

Pros &Cons of Hashing

- Pros:
  - On an average Insertion/Deletion/Search operation takes O(1) time.
- Cons:
  - In the worst case Insertion/Deletion/Search might take O(n) time. (when hash function is not good enough)

	Array	Linked List	Tree	Hashing
Insertion	O(n)	O(n)	O(logn)	O(1) / O(n)
Deletion	O(n)	O(n)	O(logn)	O(1) / O(n)
Searching	O(n)	O(n)	O(logn)	O(1) / O(n)