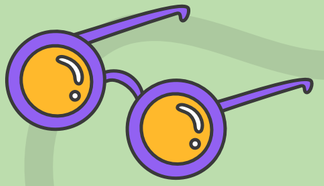


Hey it's Sohila Ashraf

Closed Hashing

strategy

we going to use the

space for time trade offs

- simply uses extra space to facilitate faster and more flexible access to the data

- We call this approach prestructuring

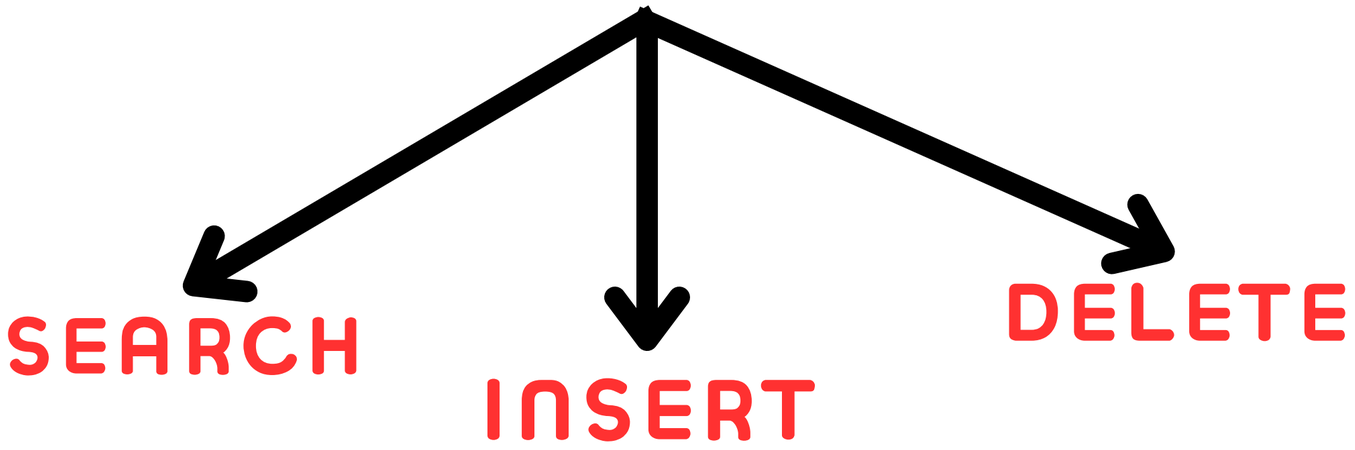

2

HASHING

- large keys are converted into small keys by using hash functions in hash table.
- keys could be numbers or string.
- How can we store them and do some operations as:

HASHING

- large keys are converted into small keys by using hash functions in hash table.
- keys could be numbers or string.
- How can we store them and do some operations as:



How to apply this

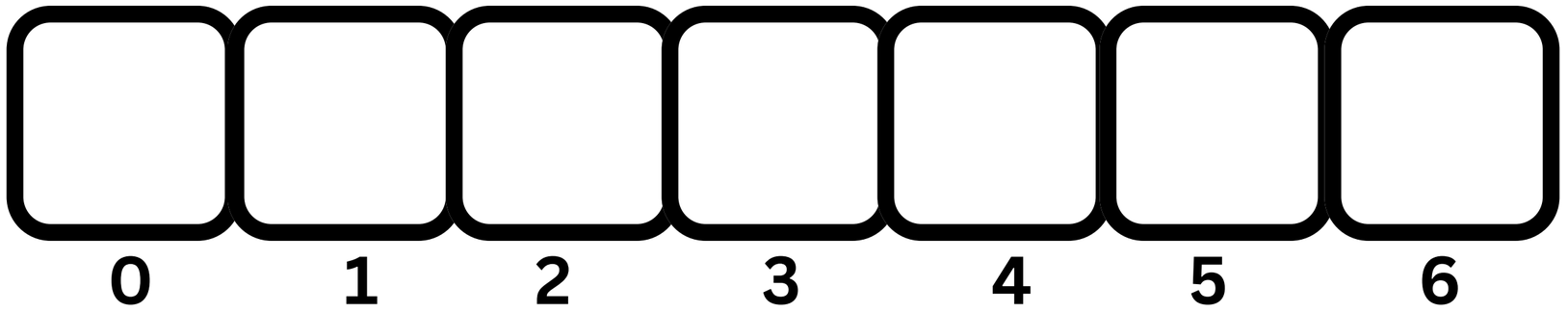
ab

bdb

cab

ddb

bbd



Hash table

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

ab

bdb

cab

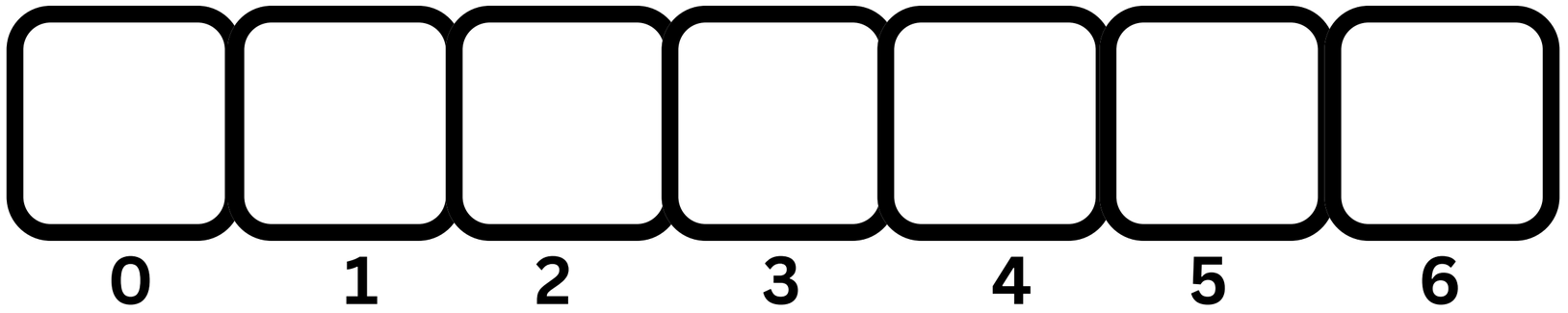
ddb

bbd

```
int StringHashFunc(string str, int TABLE_SIZE)
{
    int sum = 0;

    for(int i = 0; i < str.size(); i++)
            sum += str[i]-'a';

    return sum % TABLE_SIZE;                    // Compression
}
```

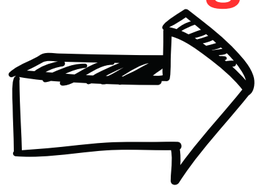


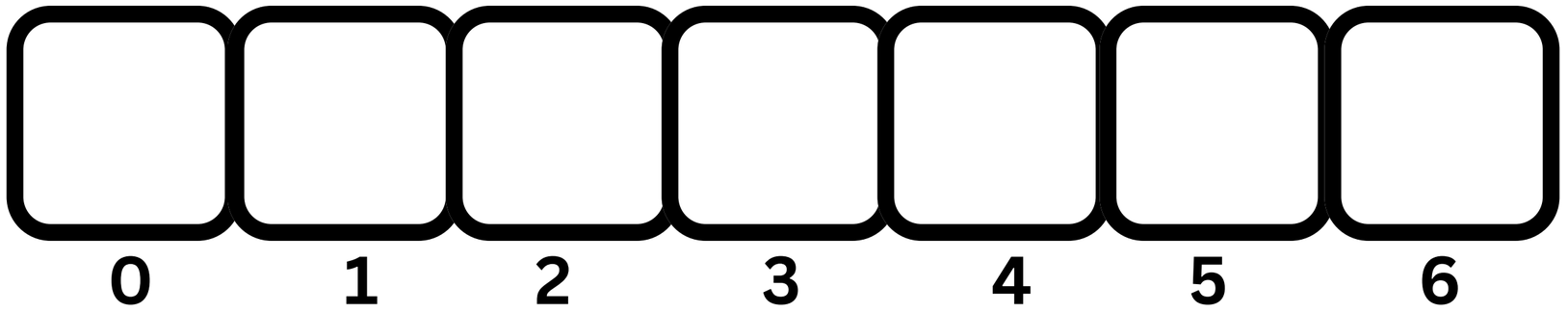
Hash table

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

ab	1
bdb	5
cab	2
ddb	7
bbd	5

Hashing

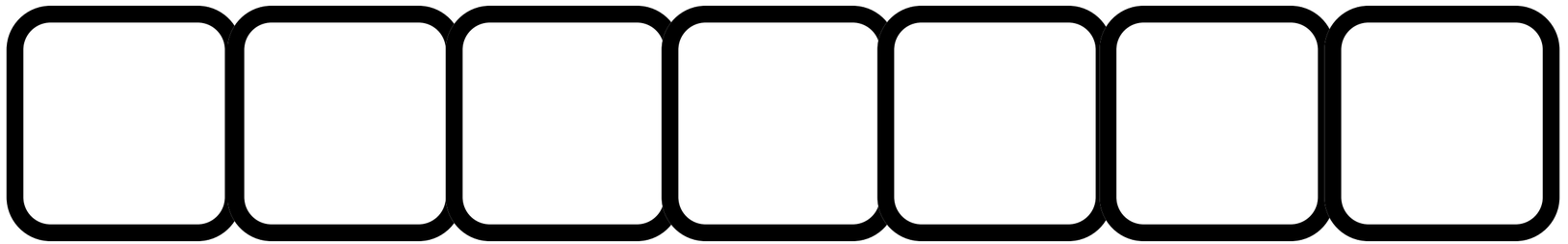




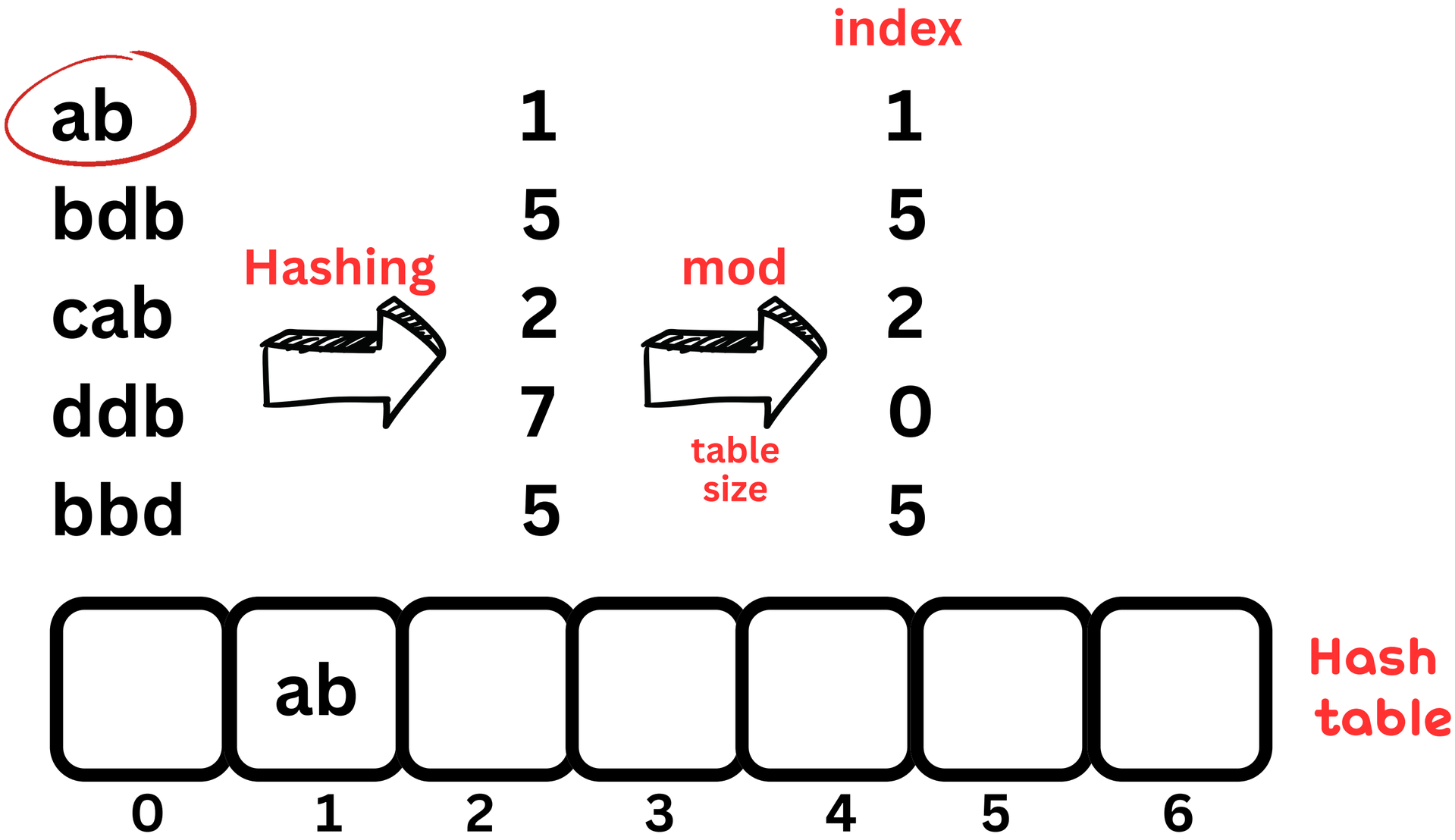
Hash table

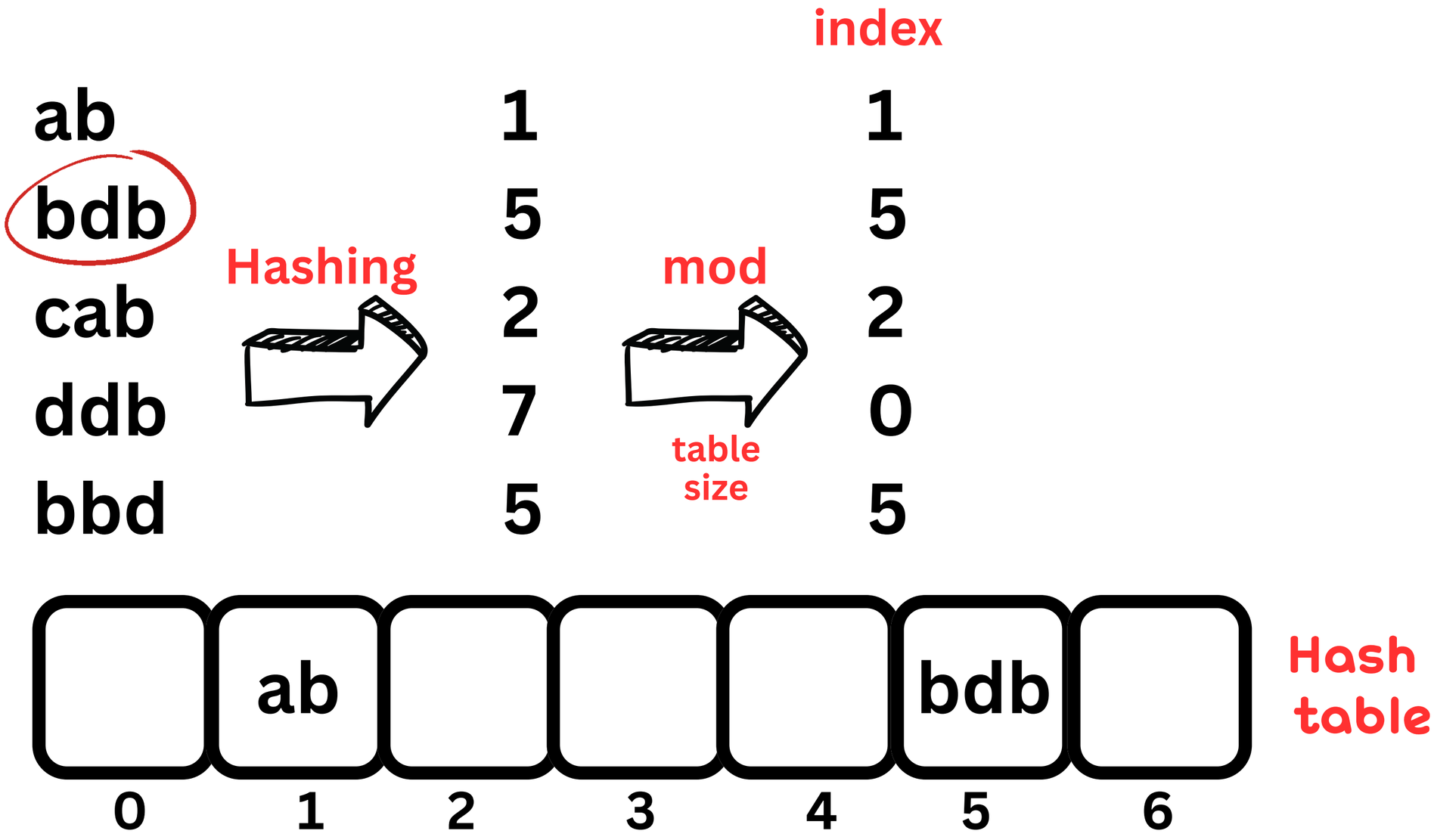
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

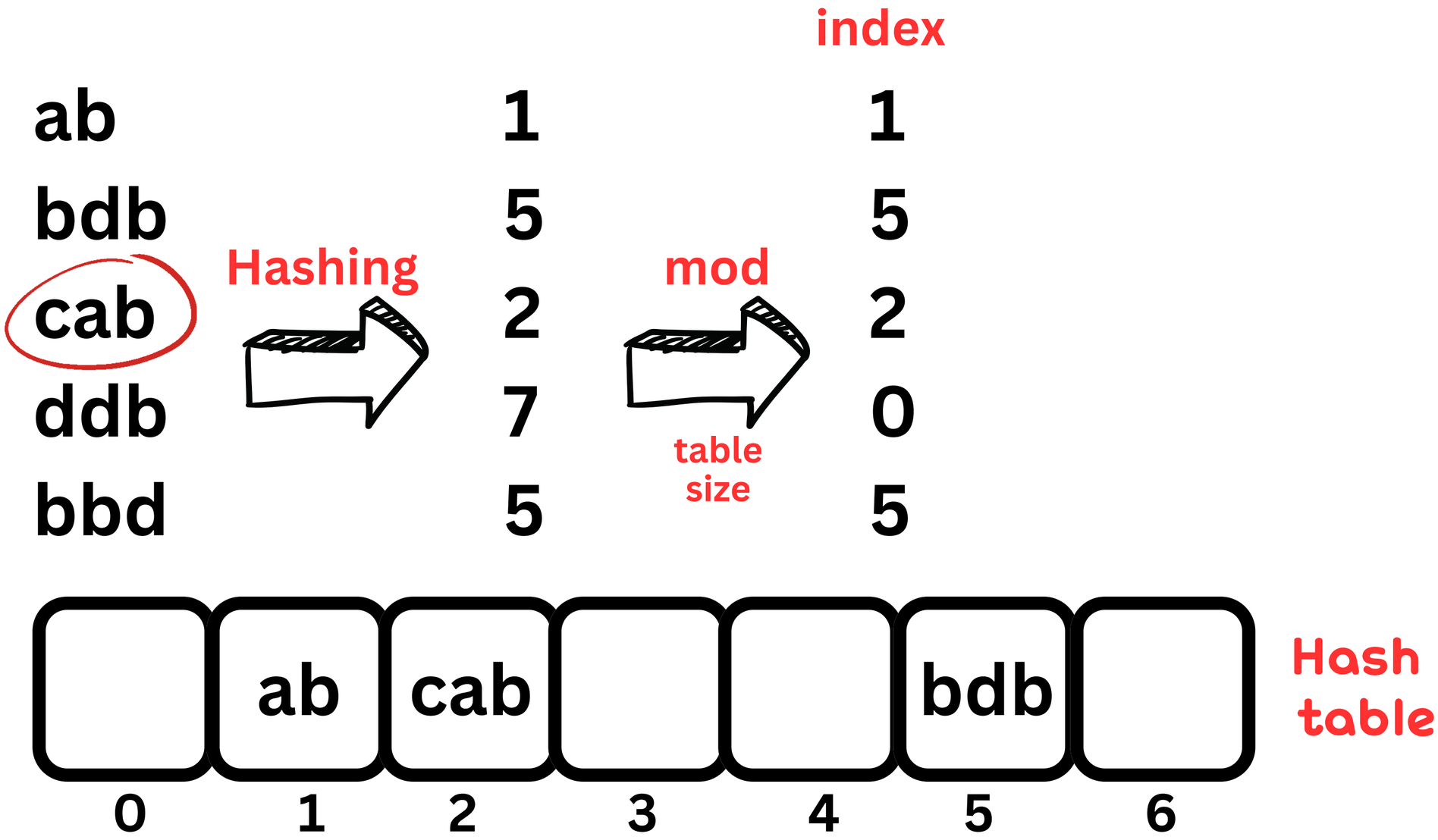
ab		1		1
bdb		5		5
cab	Hashing ⇨	2	mod table size ⇨	2
ddb		7		0
bbd		5		5

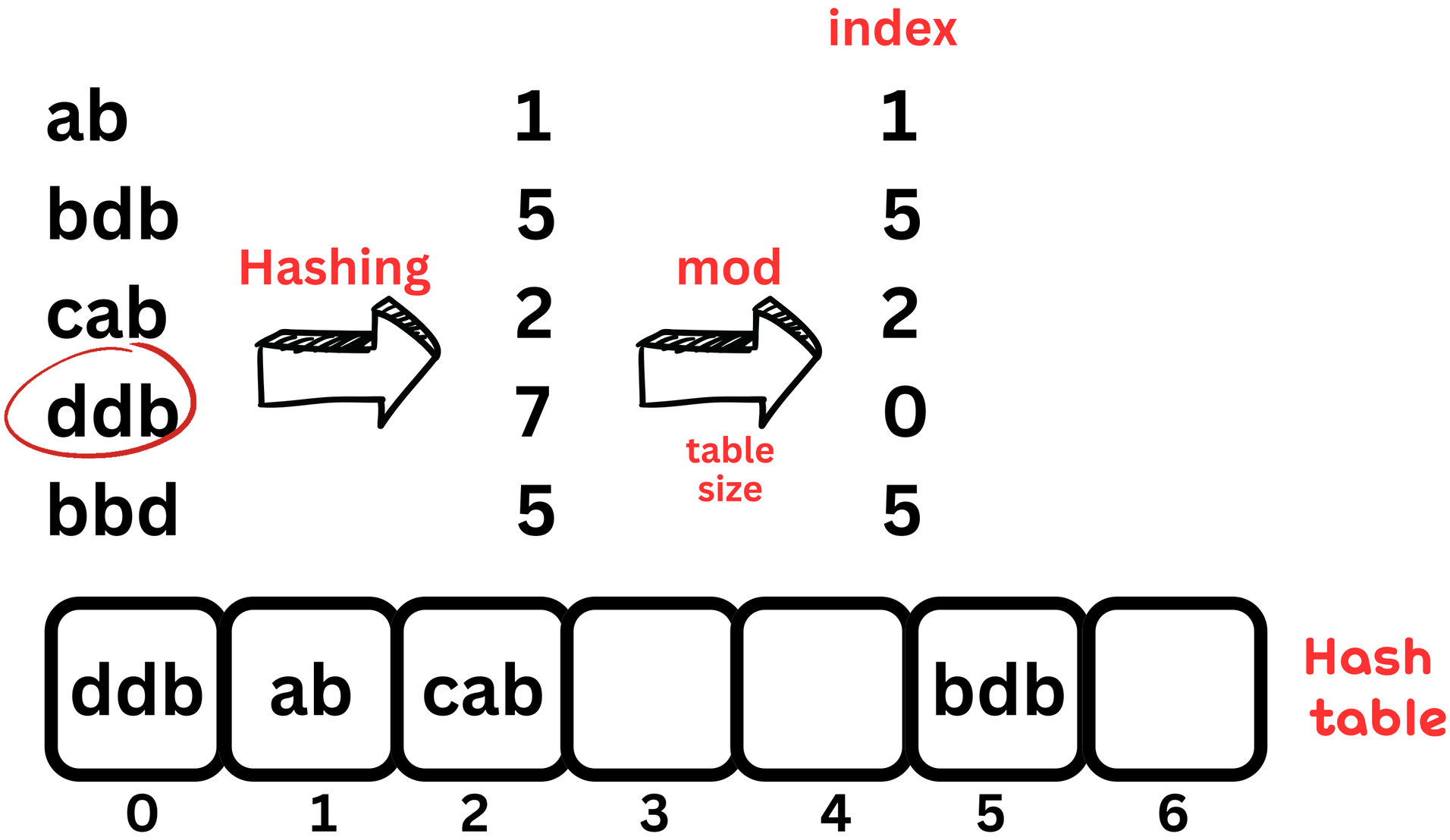


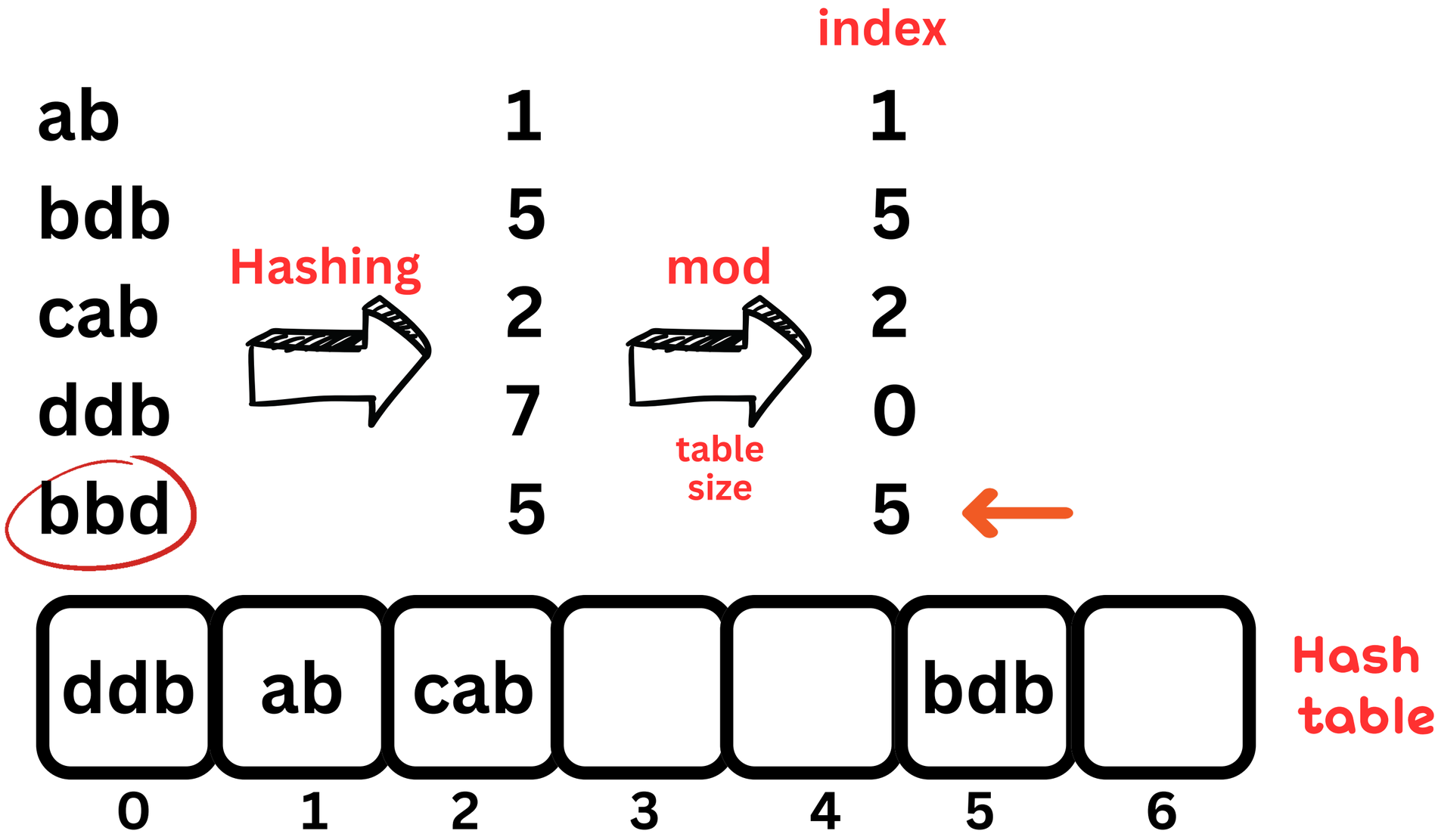
Hash table

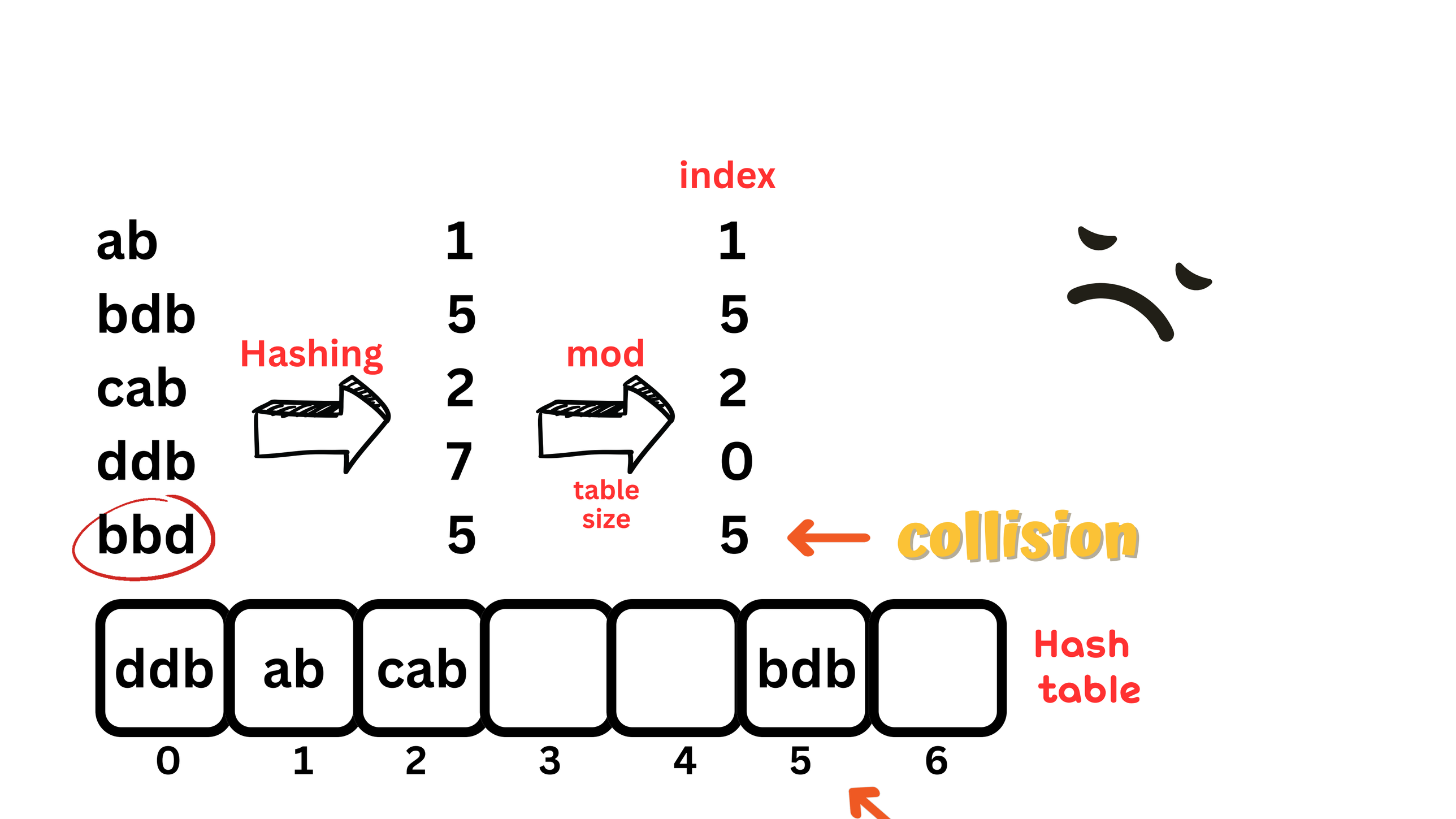

index
ab
bdb
cab
ddb
bbd
Hashing
1
5
2
7
5
mod
table
size
1
5
2
0
5
ab
0
1
2
3
4
5
6
Hash
table


index
ab
bdb
cab
ddb
bbd
Hashing
1
5
2
7
5
mod
table
size
1
5
2
0
5
ab
bdb
0
1
2
3
4
5
6
Hash
table


index
ab
bdb
cab
ddb
bbd
Hashing
1
5
2
7
5
mod
table
size
1
5
2
0
5
ab
cab
bdb
0
1
2
3
4
5
6
Hash
table


index
ab
bdb
cab
ddb
bbd
Hashing
1
5
2
7
5
mod
table
size
1
5
2
0
5
ddb
ab
cab
bdb
0
1
2
3
4
5
6
Hash
table

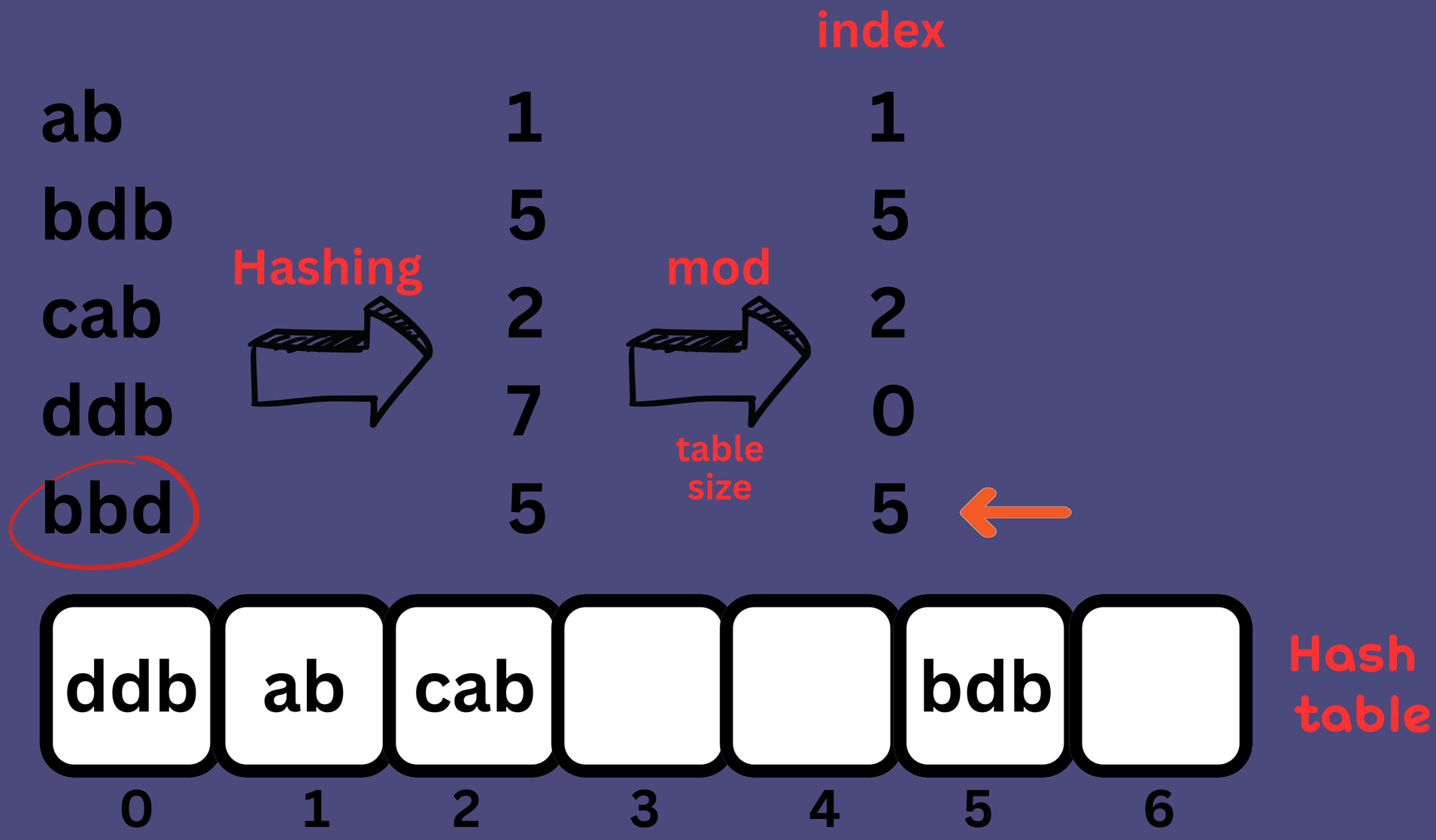

index
ab
bdb
cab
ddb
bbd
Hashing
1
5
2
7
5
mod
table
size
1
5
2
0
5
ddb
ab
cab
bdb
0
1
2
3
4
5
6
Hash
table

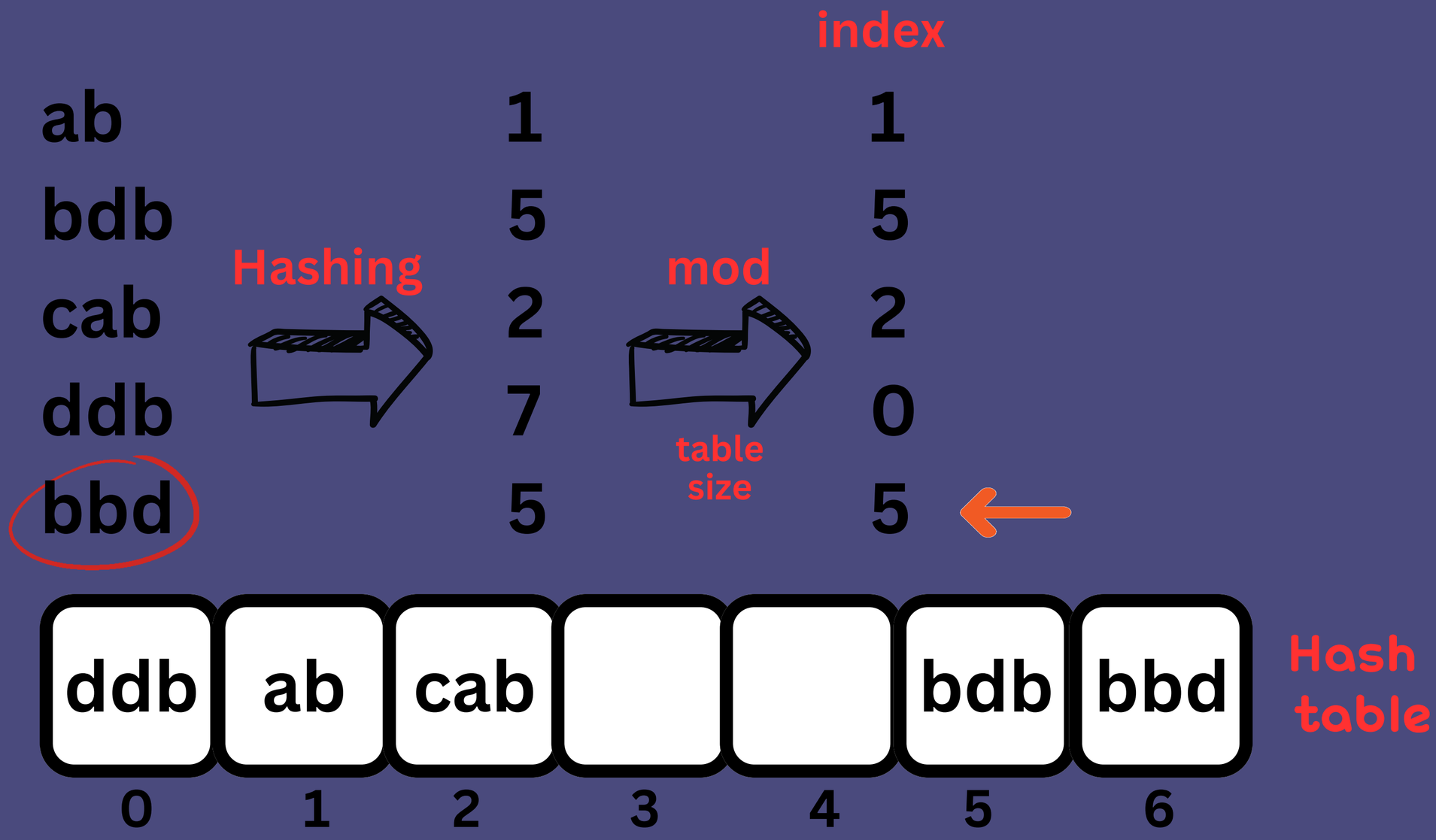

index
ab
bdb
cab
ddb
bbd
Hashing
1
5
2
7
5
mod
table size
1
5
2
0
5
collision
ddb
ab
cab
bdb
Hash table
0
1
2
3
4
5
6

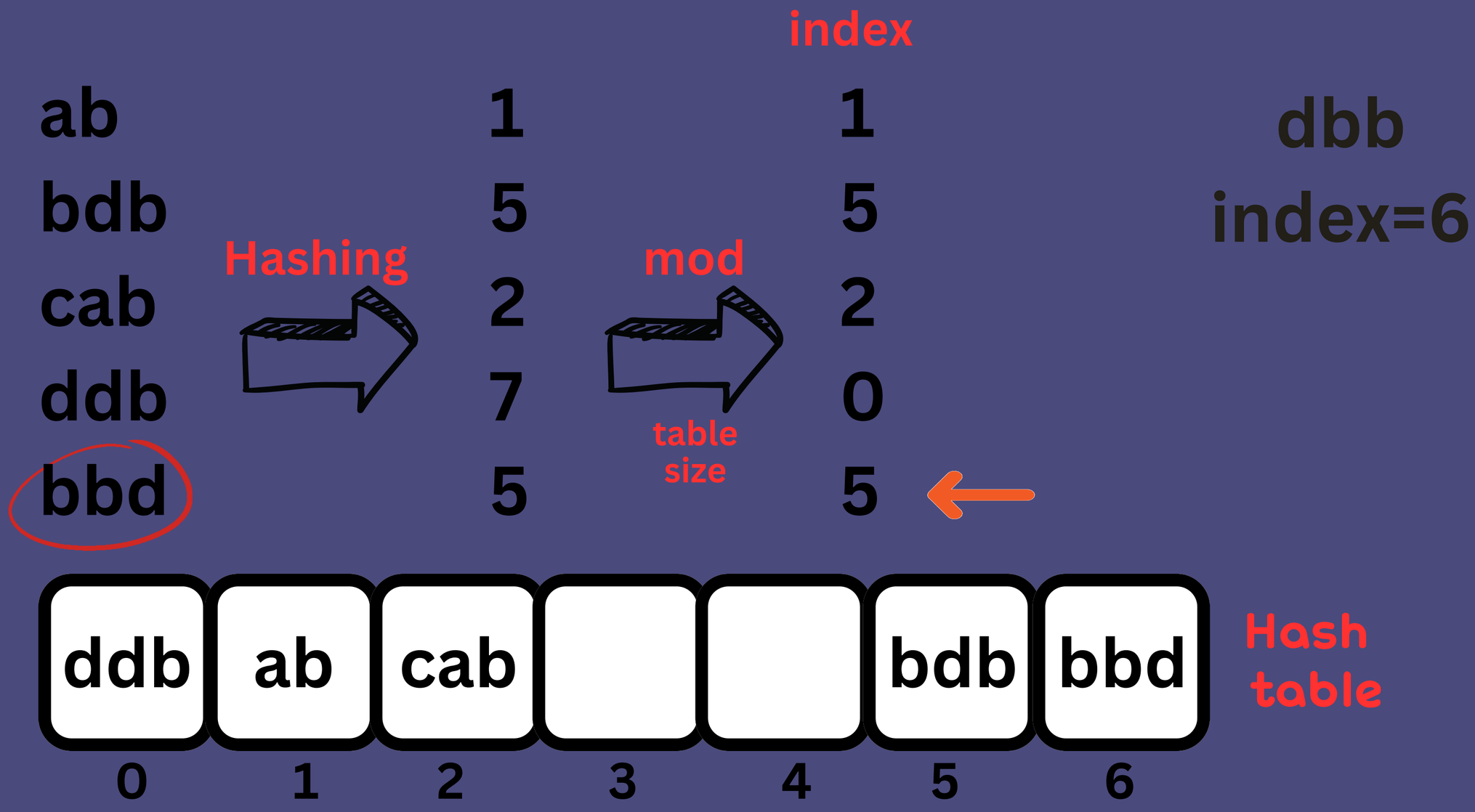
Resolving the collision using closed hashing (open addressing)

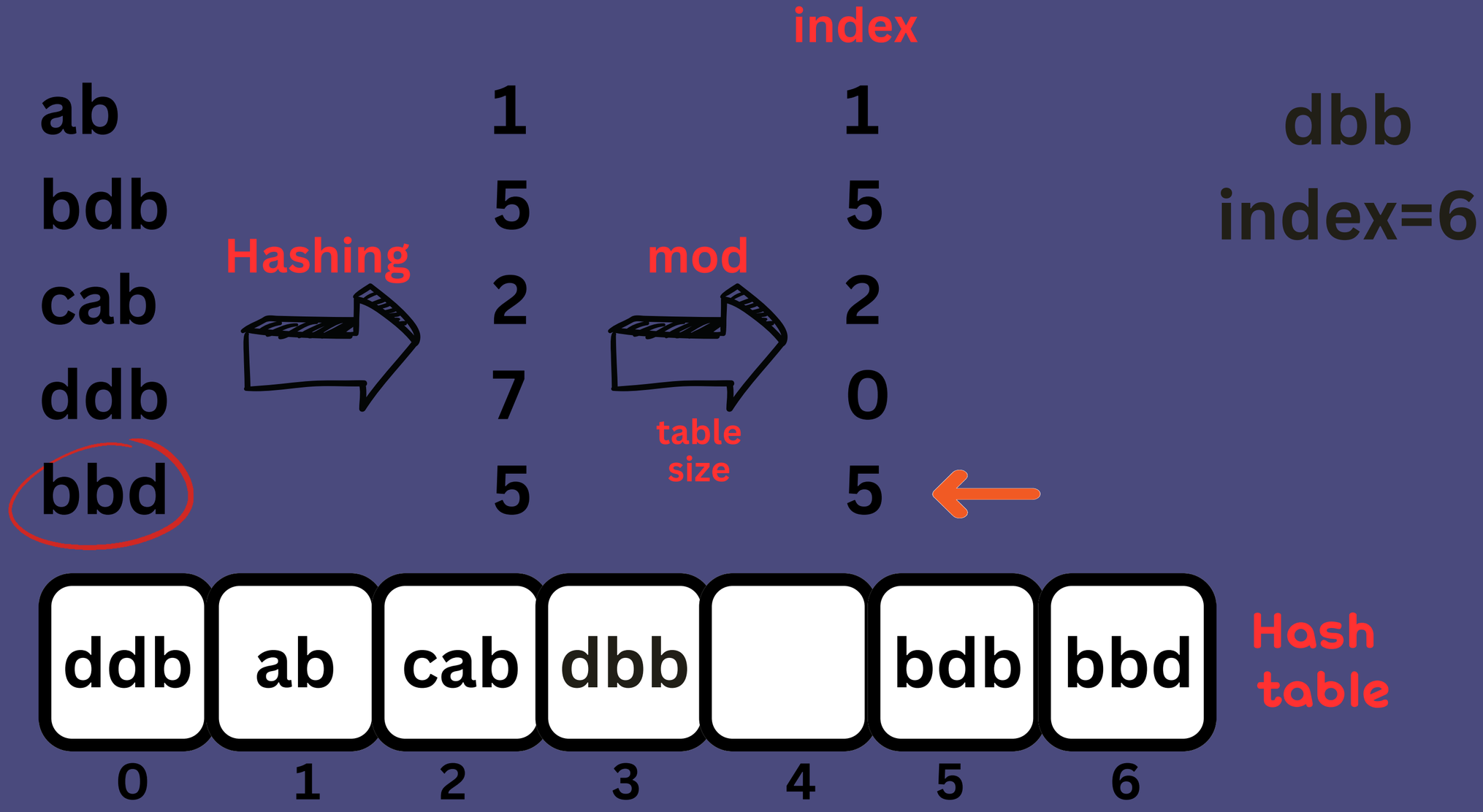
- Search for another cell by using:

  Linear Probing → Store the elements in the nearest empty cell in my hash table


index
ab
bdb
cab
ddb
bbd
Hashing
1
5
2
7
5
mod
table
size
1
5
2
0
5
ddb
ab
cab
bdb
0
1
2
3
4
5
6
Hash
table

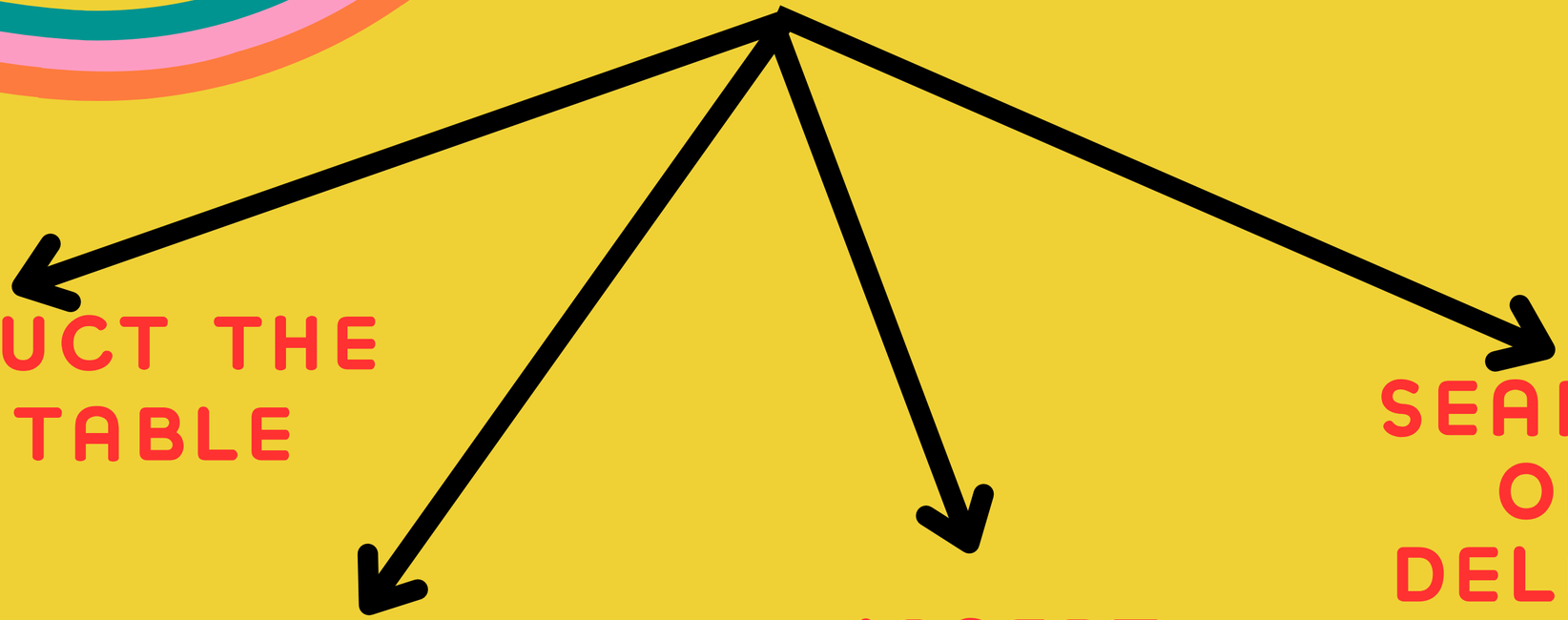

index
ab
bdb
cab
ddb
bbd
Hashing
1
5
2
7
5
mod
table
size
1
5
2
0
5
ddb
ab
cab
bdb
bbd
Hash
table
0
1
2
3
4
5
6


index
ab
bdb
cab
ddb
bbd
Hashing
1
5
2
7
5
mod
table size
1
5
2
0
5
dbb
index=6
ddb
ab
cab
bdb
bbd
0
1
2
3
4
5
6
Hash table


index
ab
bdb
cab
ddb
bbd
Hashing
1
5
2
7
5
mod
table size
1
5
2
0
5
dbb
index=6
ddb
ab
cab
dbb
bdb
bbd
0
1
2
3
4
5
6
Hash table

The Analysis

The Analysis



CONSTRUCT THE HASH TABLE

HASH FUNCTION

INSERT

SEARCH OR DELETE

Construct the hash table

$$\sum_{i=0}^{n-1} n-1-0+1 \quad \in o(n)$$

u-l+1
n:is the table size

Hash Function

$\in O(1)$

find the index to the target key

Insert

$$\sum_{i=0}^{n-1} n-1-0+1$$

$\in o(n)$ worst case

$\in O(1)$

Search and Delete

Worst Case

$$\sum_{i=0}^{n-1} n-1-0+1$$

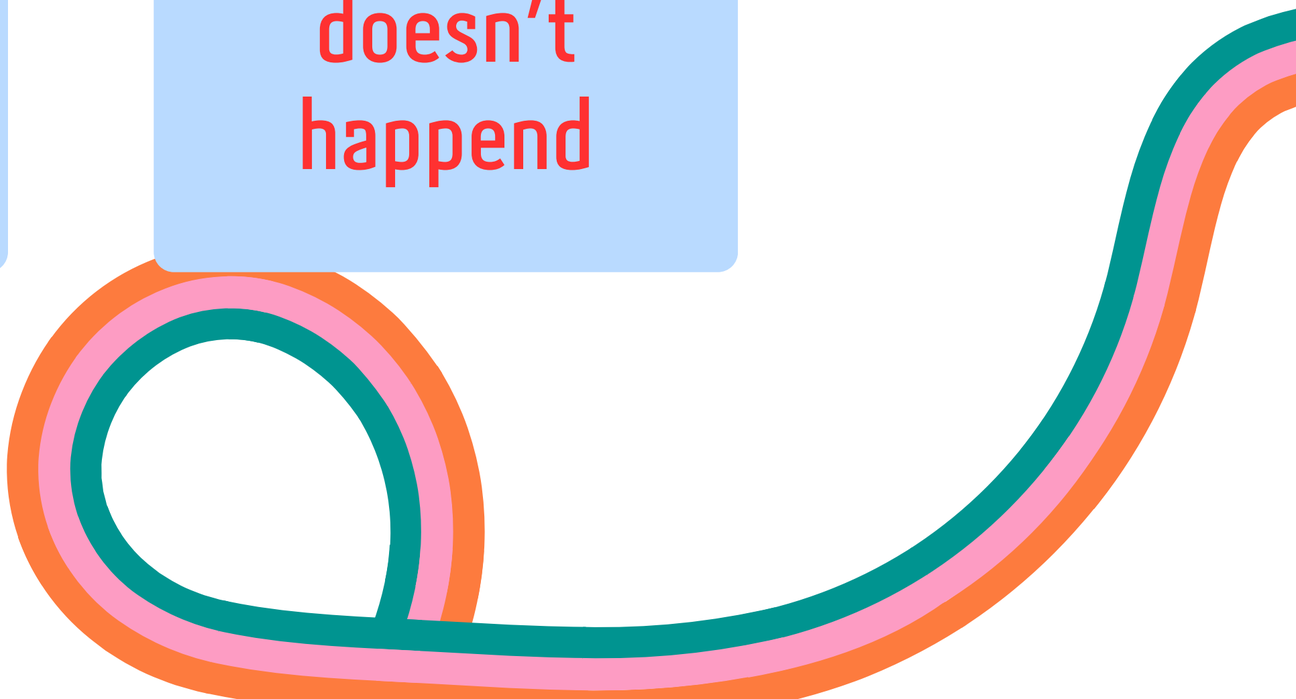
$\in o(n)$

collision happend

Best Case

$\in O(1)$

collision doesn't happend



THANK YOU