Applications for Hashing

checksumming

hard disks / File systems

Applications for Hashing

checksumming

duplicate detection

similarity search

LSH = locality-sensitive hashing

Applications for Hashing

checksumming

duplicate detection

similarity search

encryption/signatures SHA MD5

Applications for Hashing

checksumming

duplicate detection

similarity search

encryption/signatures

pseudo-randomization

seed = 42

Applications for Hashing

checksumming

duplicate detection

similarity search

encryption/signatures

pseudo-randomization

building block for query processing

point queries

ID = 42

Applications for Hashing

checksumming

duplicate detection

similarity search

encryption/signatures

pseudo-randomization

building block for query processing

point queries

range queries

$42 \leq ID \leq 7000$

Applications for Hashing

- checksumming
- duplicate detection
- similarity search
- encryption/signatures
- pseudo-randomization
- building block for query processing
- point queries
- range queries
- ...

Point Queries

given:

an integer domain 0, ... , N-1

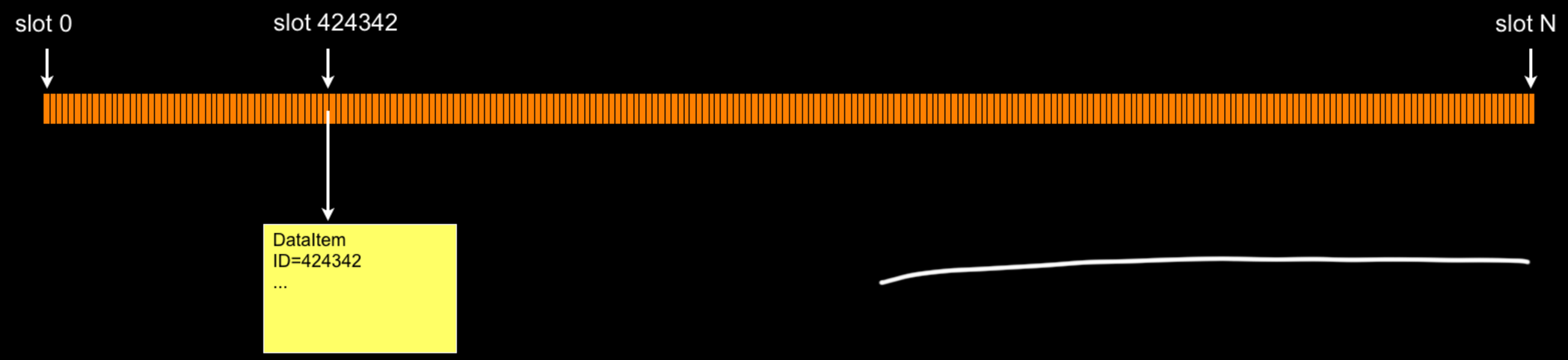
N is potentially large

point queries asking for data items having a key in 0, ... , N-1

Example:

What is the address of the student having ID=424342?

Naïve Solution: Array



= 기억

DataItem[] 배열 = new DataItem[N];

8B × N

Array Methods

```
insert(Integer key, DataItem value):
        array[key] = value;
```

aka put(..) or set(..)

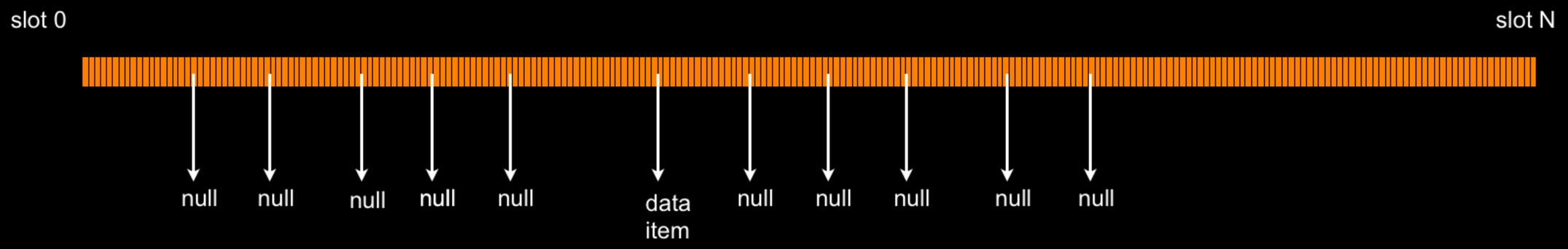
```
delete(Integer key):
        array[key] = null;
```

aka remove(..)

```
pointQuery(Integer key):
        return array[key];
```

aka get(..)

Space Usage



$$N = 100{,}000{,}000$$

$$X = 10{,}000$$

$$N/X = 10{,}000$$

Core Idea of Static Hashing

pick an M << N

allocate an array of size M only

transform any incoming key from

range 0, ... , N-1 to range 0, ... , M-1

Core Idea of Static Hashing

pick an M << N

allocate an array of size M only

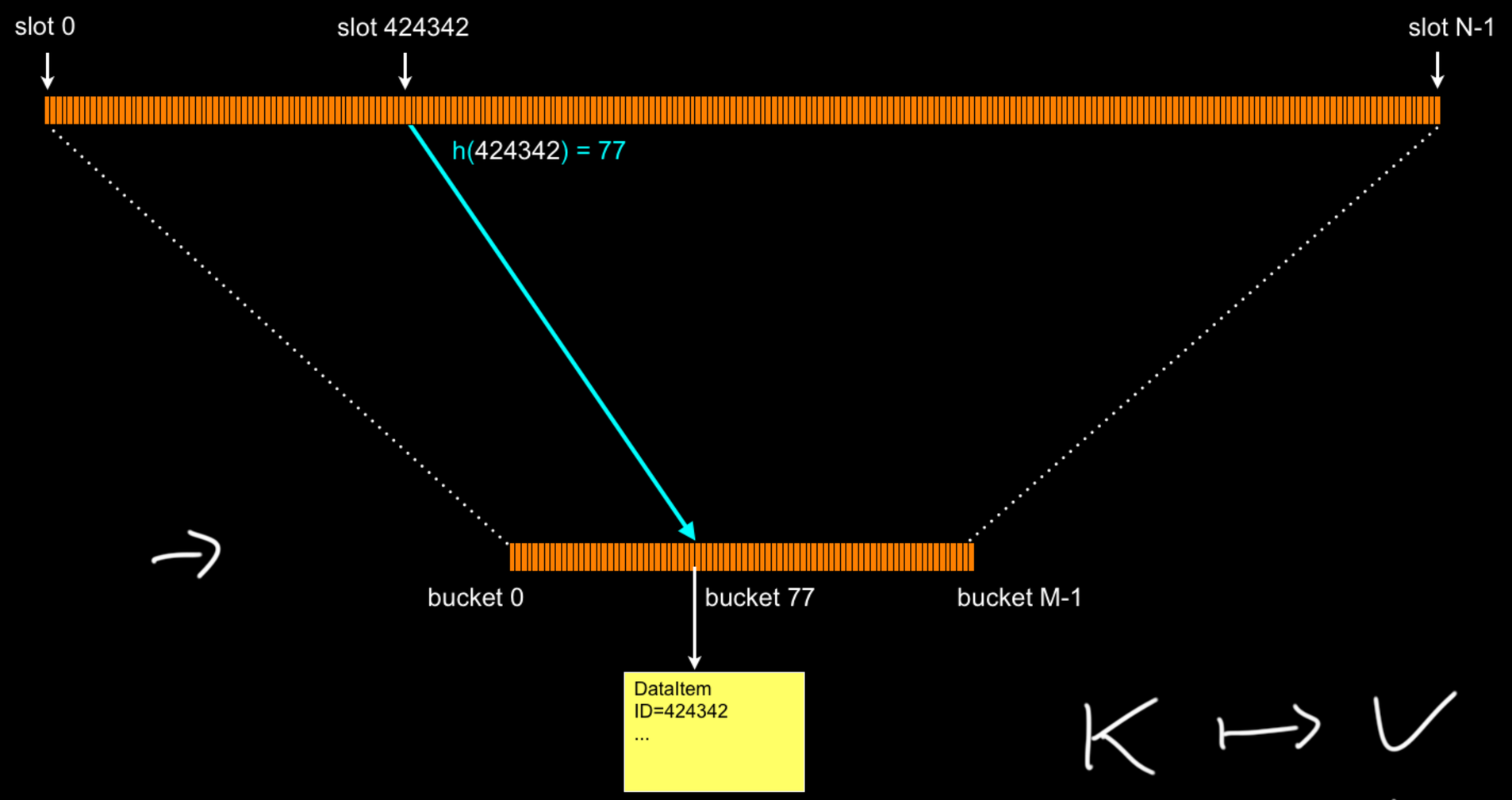
transform any incoming key from

range 0, ... , N-1 to range 0, ... , M-1

this is achieved by a so-called hash function h(key)

h(key) % M

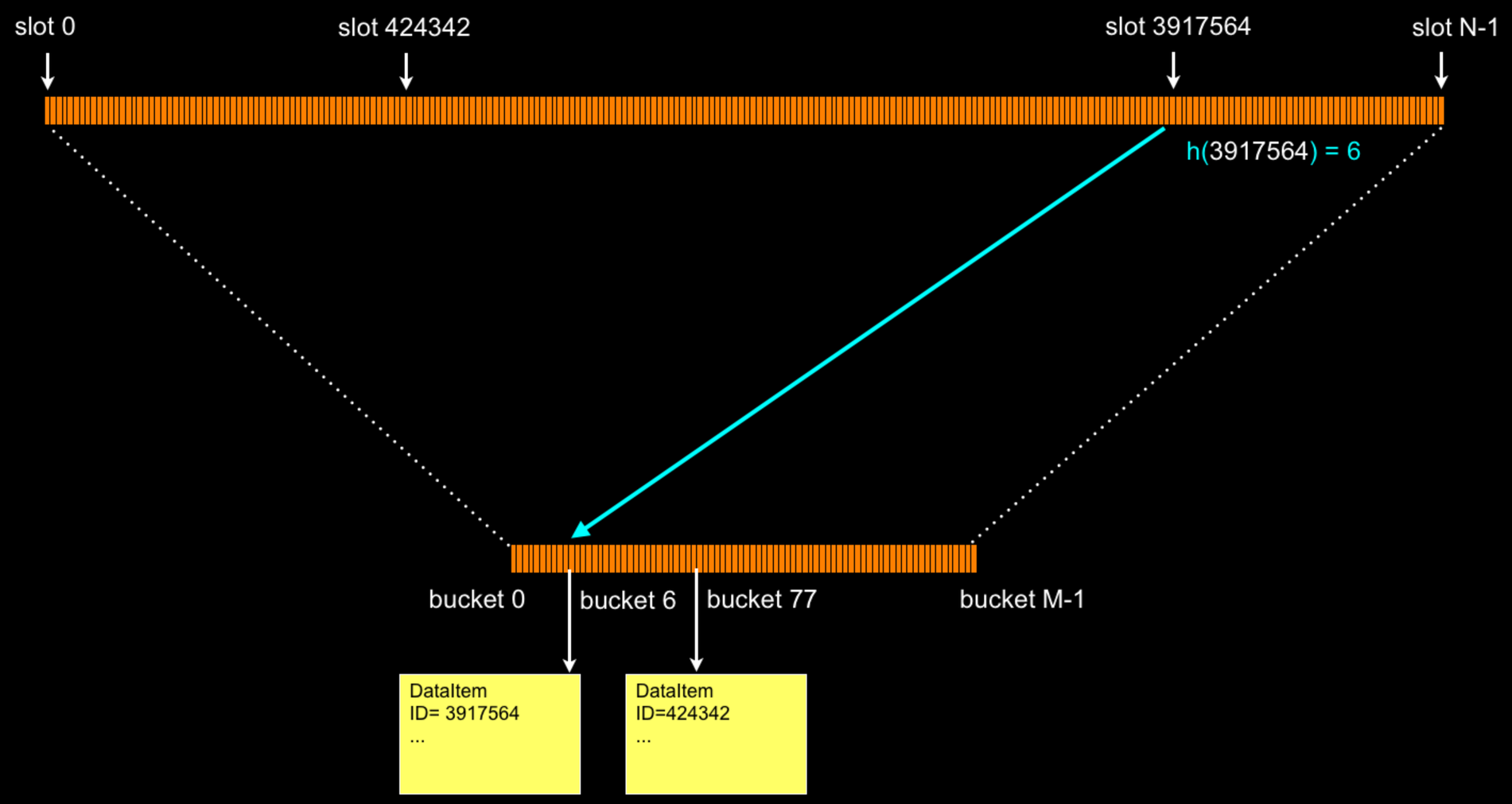
insert 424342



$K \mapsto V$

$ID \mapsto$ DataItem

insert 3917564



HashMap Methods (Arbitrary Type)

Note: this is ignoring collisions for the moment, see next slide for collisions.

insert(Key key, DataItem value): aka put(..) or set(..)

```
array[ h(key) ] = value;
```

$O(1)$

delete(Key key): aka remove(..)

```
array[ h(key) ] = null;
```

$O(1)$

pointQuery(Key key): aka get(..)

```
return array[ h(key) ];
```

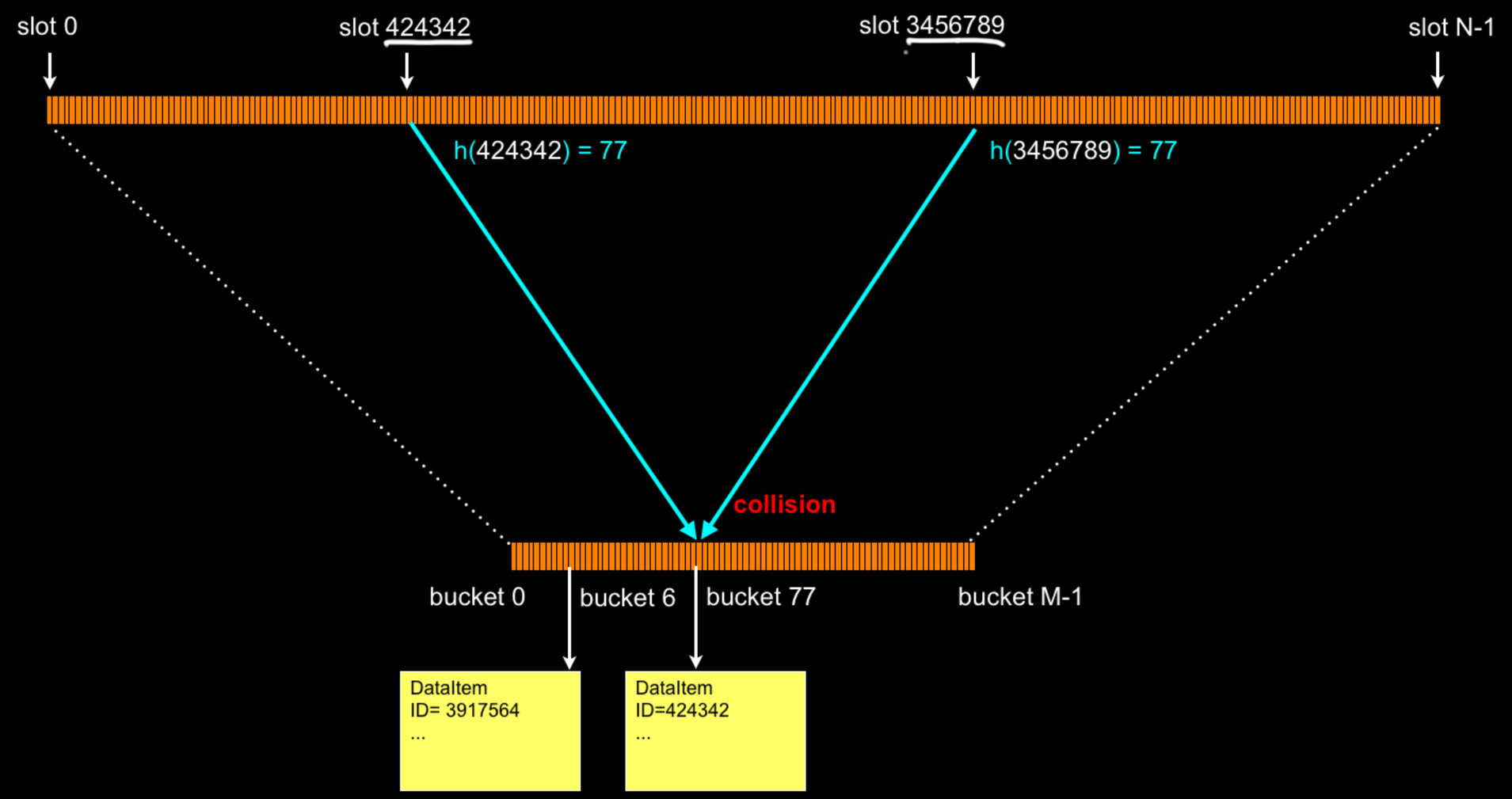
$O(1)$

$K \mapsto V$

String $\mapsto$ DataItem

h: String $\rightarrow$ Integer

Collisions? insert 3456789

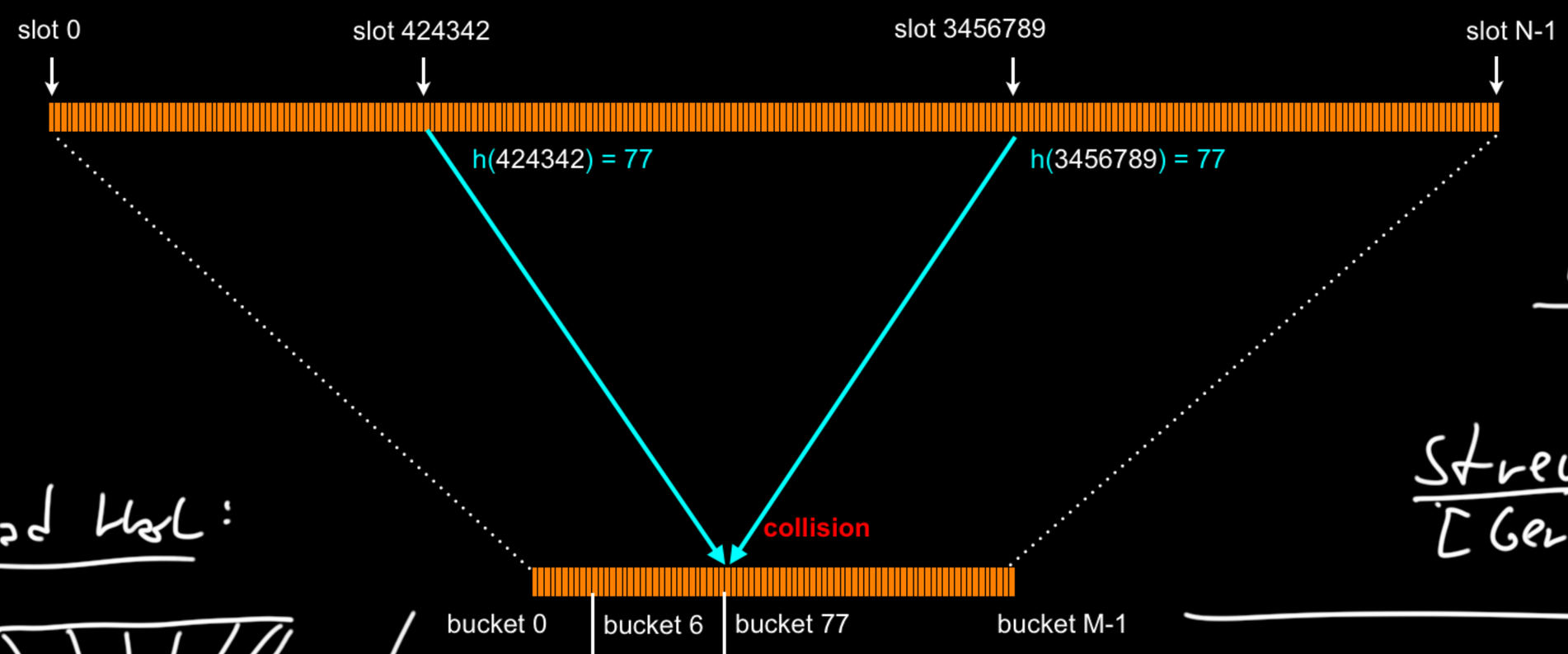


PK

||

No Duplicates

Overflow Chains

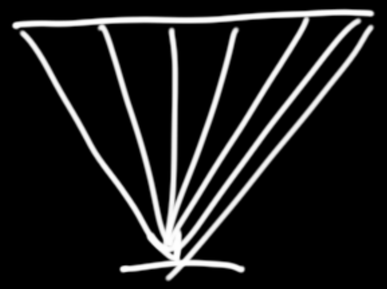


Hash

↓

Streutabelle
[German]

Bad Hash:



array[77] =

array[77] =

h(k)=42

Rehash!

pick an M‘ > M

create new hash table with M‘

re-insert all entries from old hash table into new hash table

rehash