Hash Tables

Robert Horvick
SOFTWARE ENGINEER

@bubbafat www.roberthorvick.com

Hash Table Overview

- Associative Array

Hashing Algorithms

- Stable
- Uniform Distribution
- Secure

Hash Table Operations

- Add
- Search
- Remove

Demo: State Level Caching

Associative Array

A collection of key/value pairs where the key can only exist once in the collection

Associative Array Examples

HTTP Headers

Application Configuration

Environment Variables

Key/Value Database

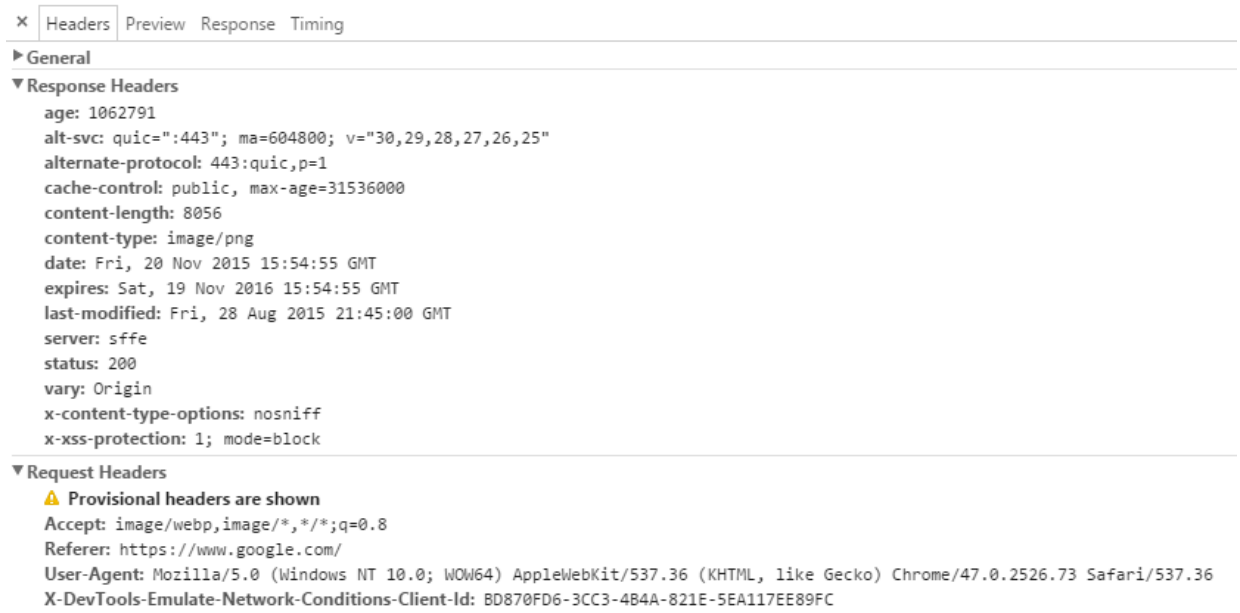
Request URL

Response body

Browser version

Referrer

Etc...



```
HttpHeader headers;

headers["content-length"] = "8056";

headers["content-type"] = "image/png";
```

HTTP Headers

The header name and value become the key and value in the associative array

Environment Variables

```
IFS=$' \t\n'
JAVA_HOME=/usr/local/jdk
LESSOPEN='||/usr/bin/lesspipe.sh %s'
LINES=50
LOCALLIBUSER=autumnpa
LOGNAME=autumnpa
LS_COLORS='rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=4
:*.arj=01;31:*.taz=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.zip=01;31:*.z=01;31
z2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.rar=01;31:*.ace=01;31:*.zoo=01;31:*.c
.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35
1;35:*.mkv=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=
v=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.axv
idi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.axa=00;36:*.
LS_OPTIONS='--color=tty -F -a -b -T 0'
MACHTYPE=x86_64-redhat-linux-gnu
MAIL=/var/spool/mail/autumnpa
MAILCHECK=60
OLDPWD=/home7/autumnpa/www/drupal
OPTERR=1
OPTIND=1
OSTYPE=linux-gnu
PATH=/usr/local/jdk/bin:/usr/lib64/qt-3.3/bin:/home7/autumnpa/perl5/bin:/ramdisk/php/54/bin:/us
home7/autumnpa/bin
PERL5LIB=/home7/autumnpa/perl5/lib/perl5
PERL_LOCAL_LIB_ROOT=/home7/autumnpa/perl5
PERL_MB_OPT='--install_base "/home7/autumnpa/perl5"'
PERL_MM_OPT=INSTALL_BASE=/home7/autumnpa/perl5
PIPESTATUS=([0]="0")
PPID=26704
```

```
Environment env;

env[”SSH_TTY"] = ”/dev/pts/0";

env[”PATH"] = ”/usr/local/jdk/bin/...";
```

Environment Variables

The header name and value become the key and value in the associative array

Hash Table

An associative array container that provides O(1) insert, delete and search operations.

```
HashTable<String, String> headers;


headers["content-length"] = "8056";

headers["content-type"] = "image/png";
```

Hash Table

The hash table stores HTTP header data where the key and value are both strings

```
HashTable<String, HttpHeaderValue> headers;


headers["content-length"] = IntHeader(8056)

headers["content-type"] = StringHeader("image/png");
```

Hash Table

Hash table key and value types do not have to be the same

Hash Function

A function that maps data of arbitrary size to data of a fixed size.

Hash Function Examples

Verifying downloaded data

Storing passwords in a database

Hash tables key lookup

Stability

Uniformity

Security

hash = f(value)

Stability

A hash function always generates the same output given the same input.

Hash Function Stability

Stable

```
public int StableHash(string input)
{
   int result = 0;

   foreach(byte ascii in input)
   {
      result += ascii;
   }

   return result;
}
```

Unstable

```
public int UnstableHash(string input)
{
   int result = DateTime.Now.Second;

   foreach(byte ascii in input)
   {
      result += ascii;
   }

   return result;
}
```

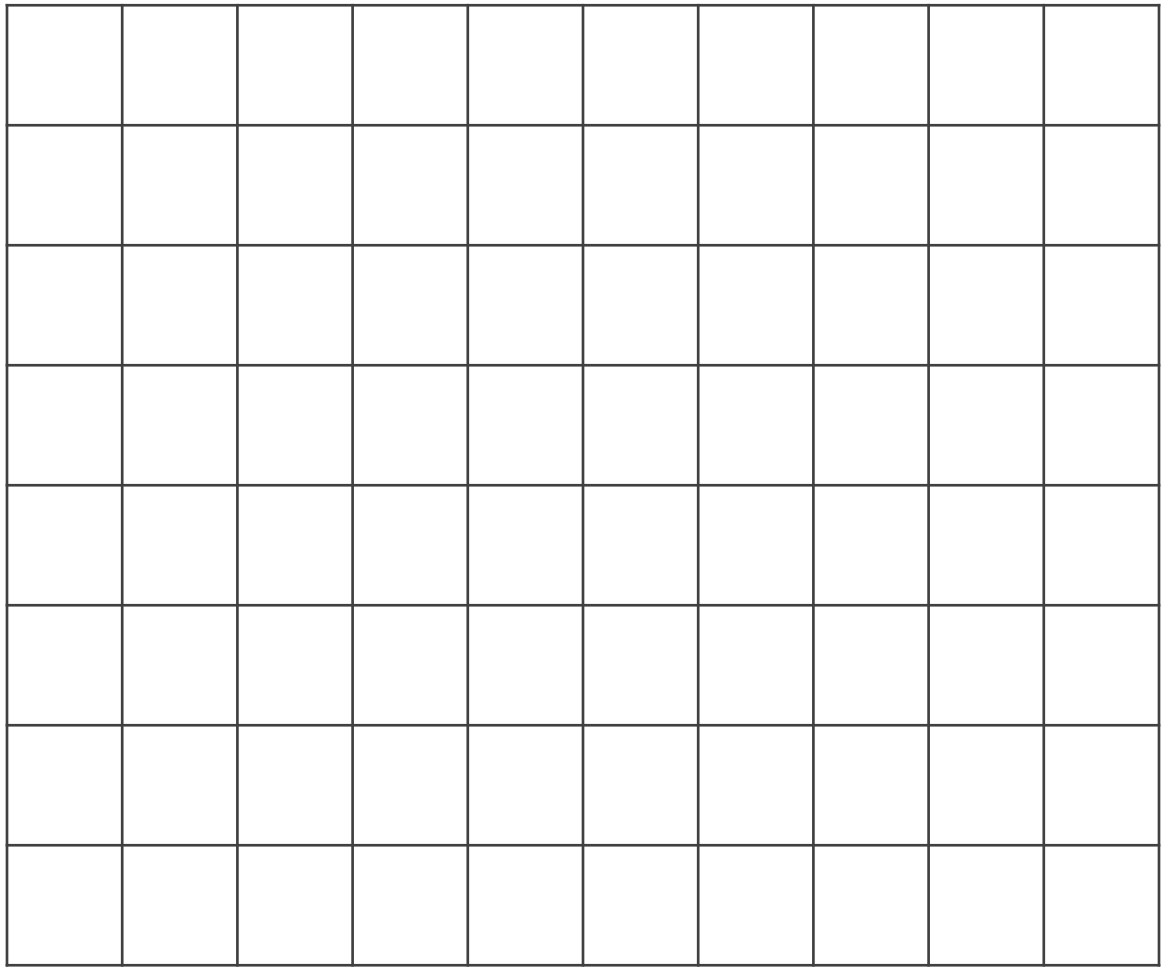
Hash Function Stability

Stable	Unstable
StableHash("foo");	UnstableHash("foo");
StableHash("foo");	UnstableHash("foo");
StableHash("foo");	UnstableHash("foo");
324	1449447443
324	1449447444
324	1449447445

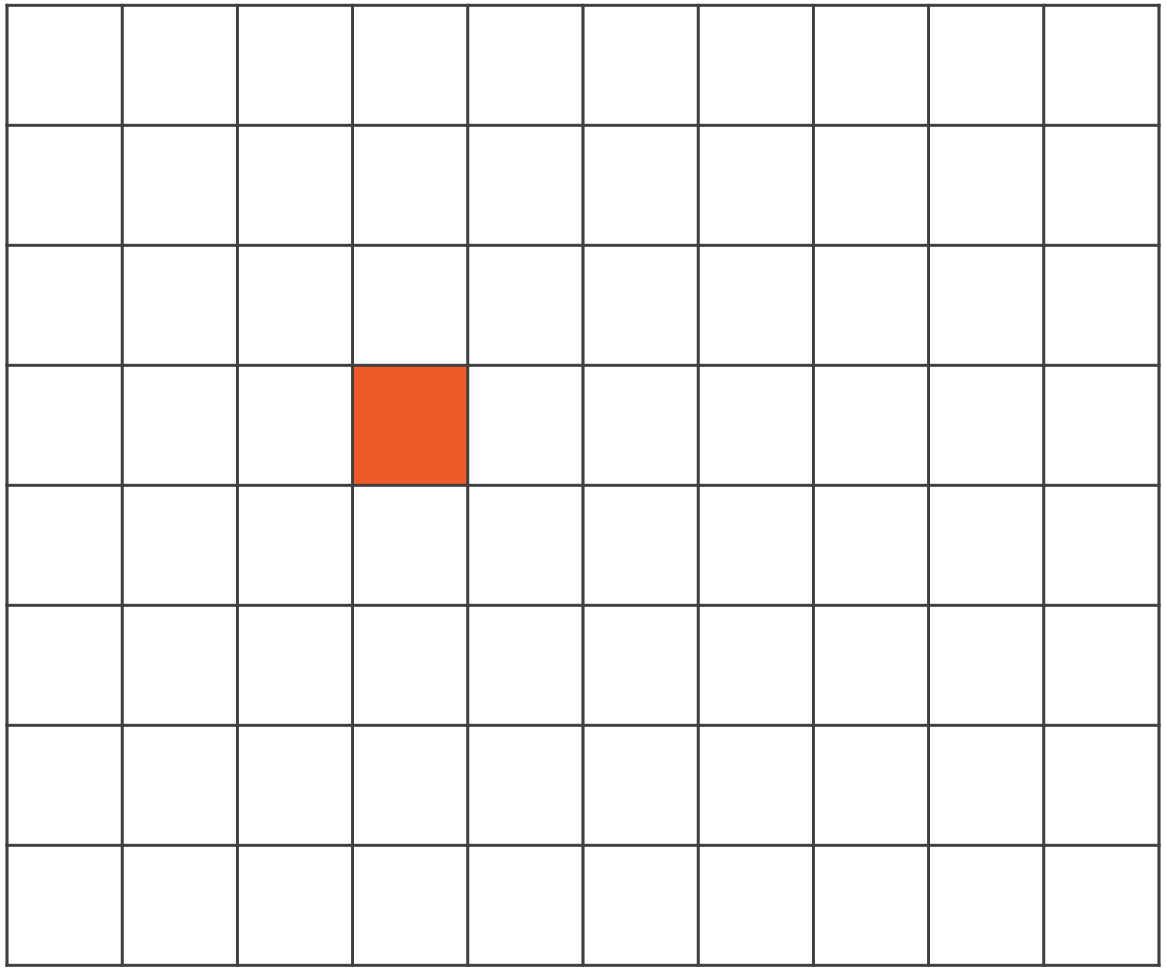
Uniformity

A hash algorithm should distribute its resulting hash value uniformly throughout the output space.

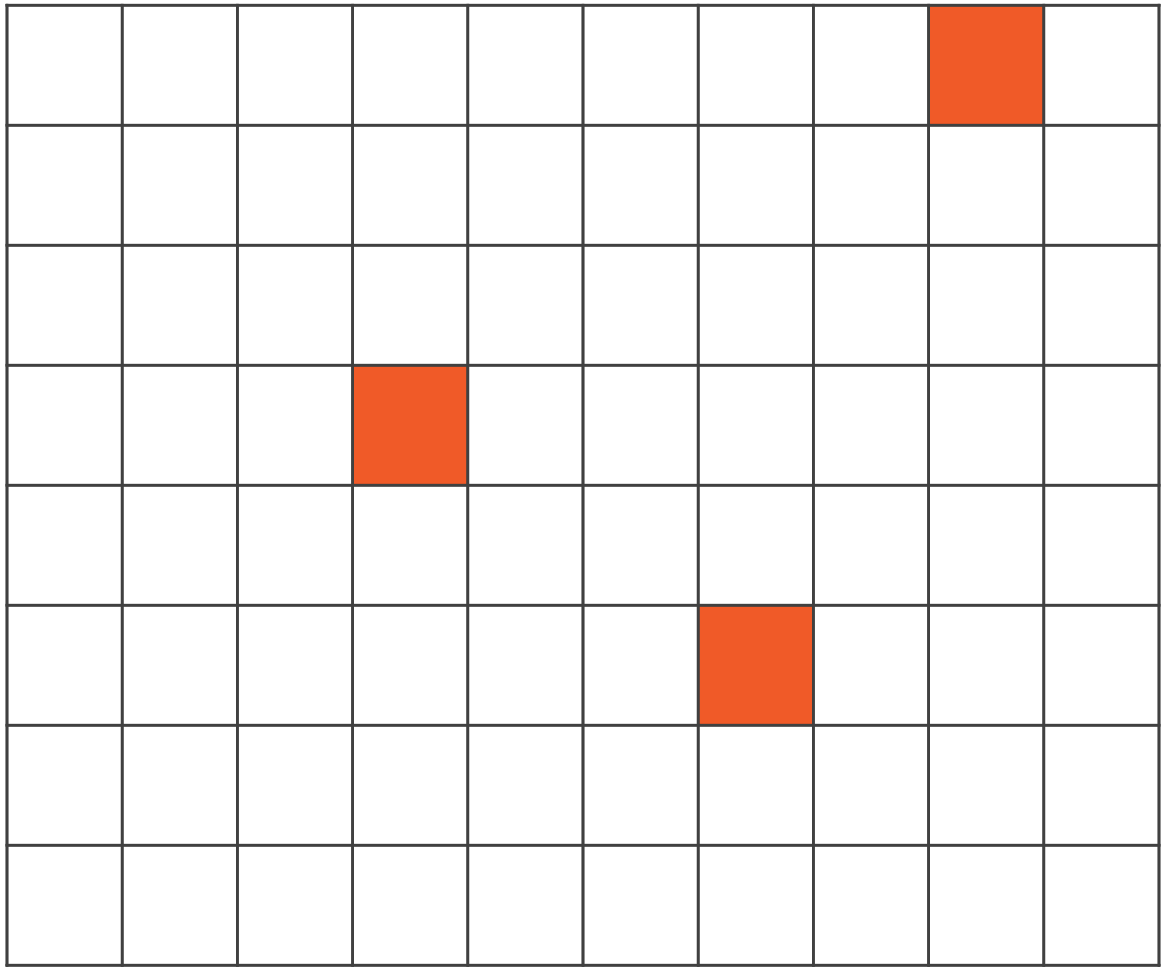
Uniform Distribution



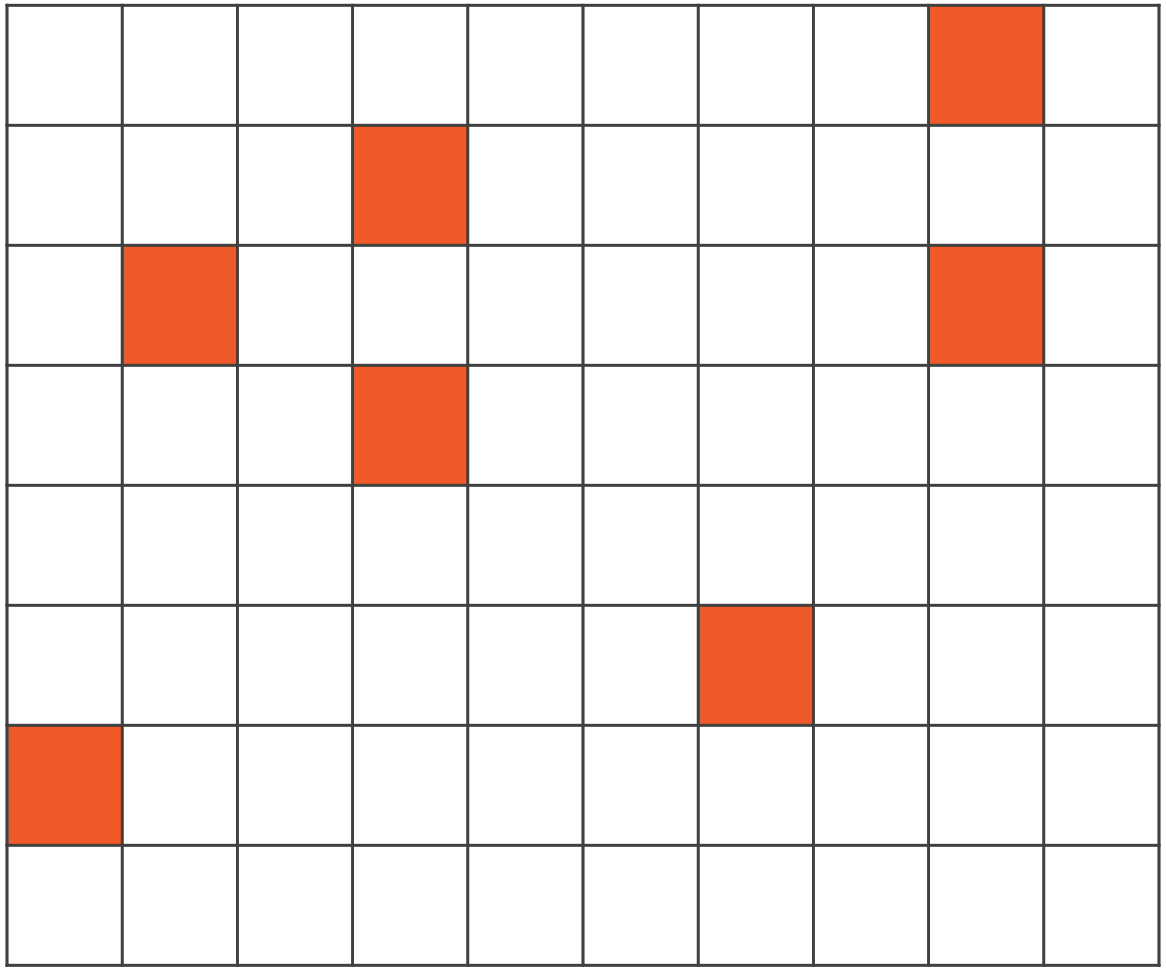
Uniform Distribution



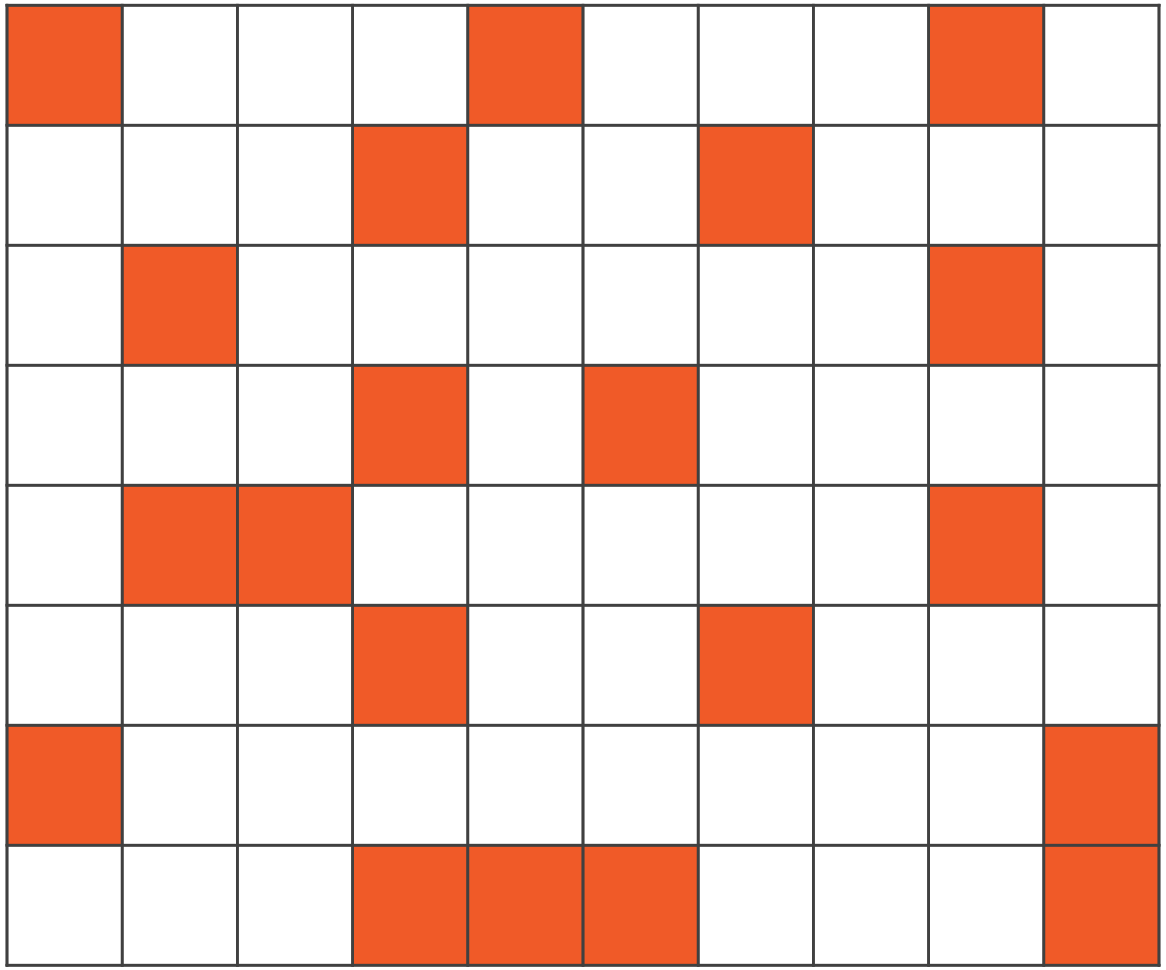
Uniform Distribution



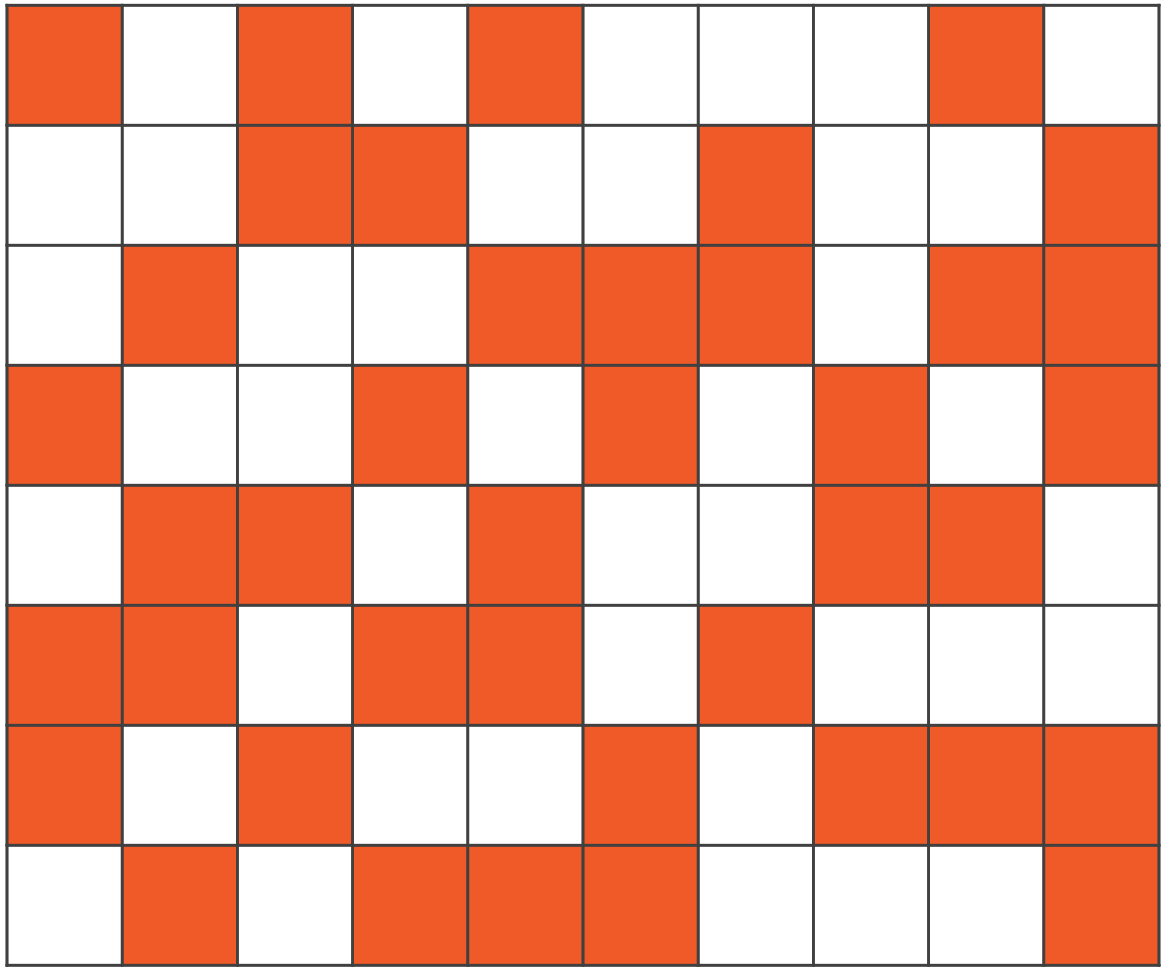
Uniform Distribution



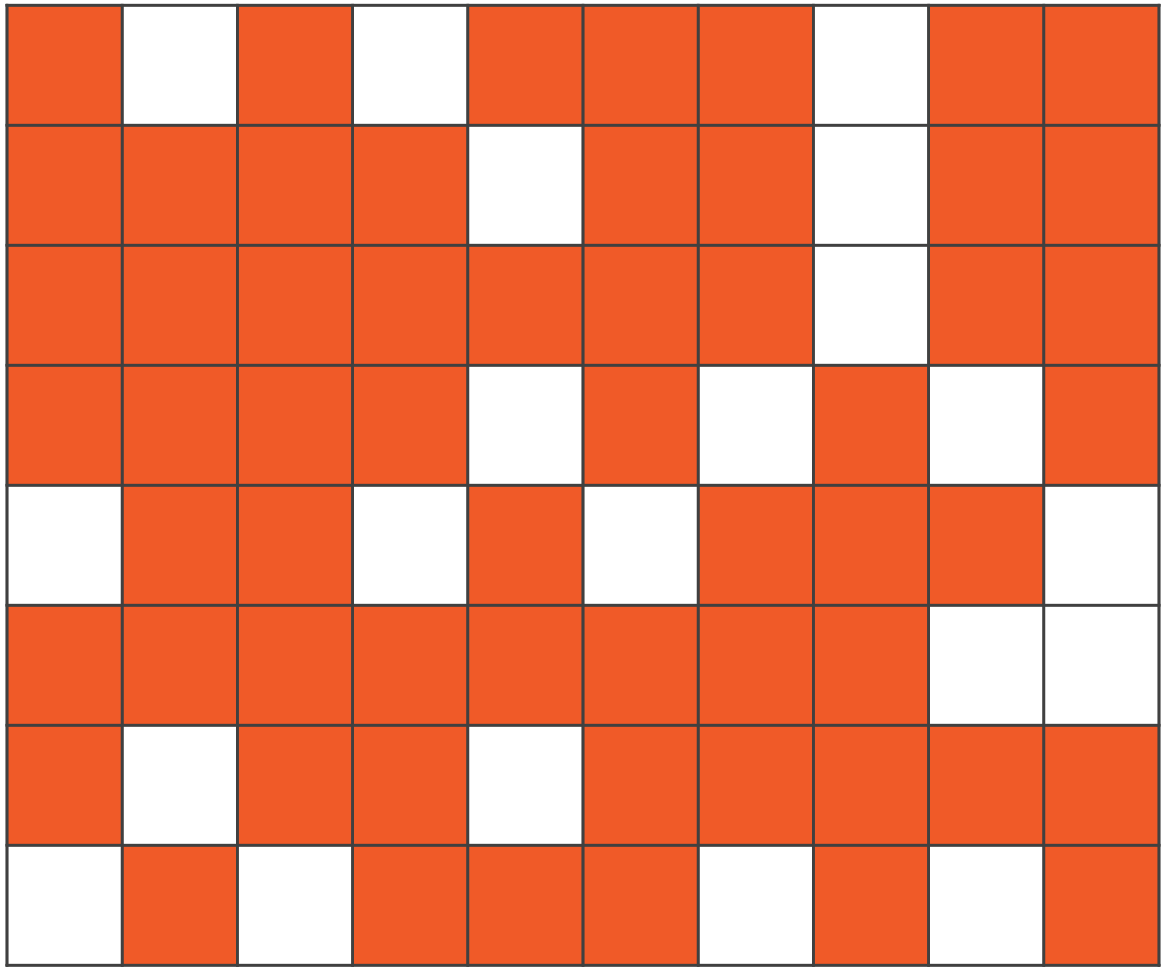
Uniform Distribution



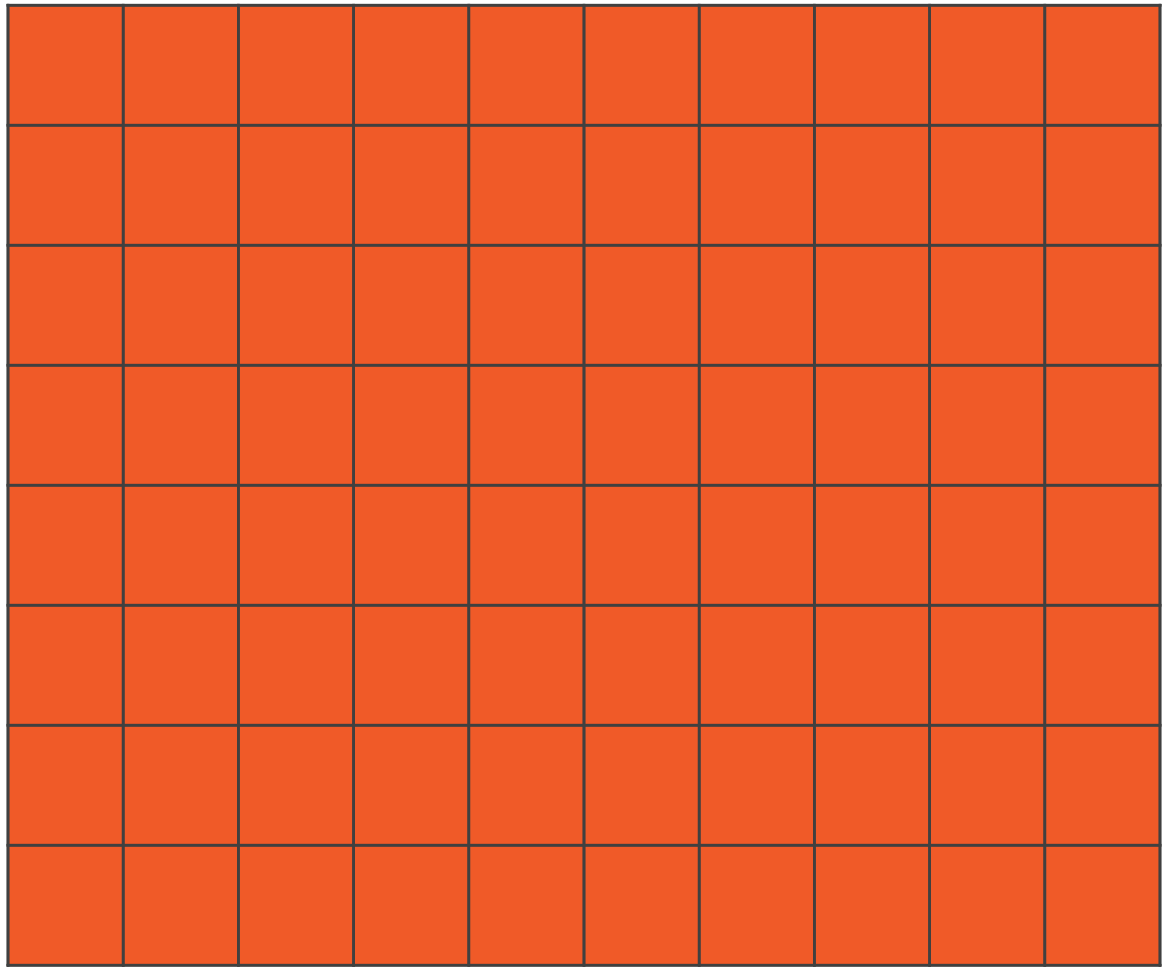
Uniform Distribution



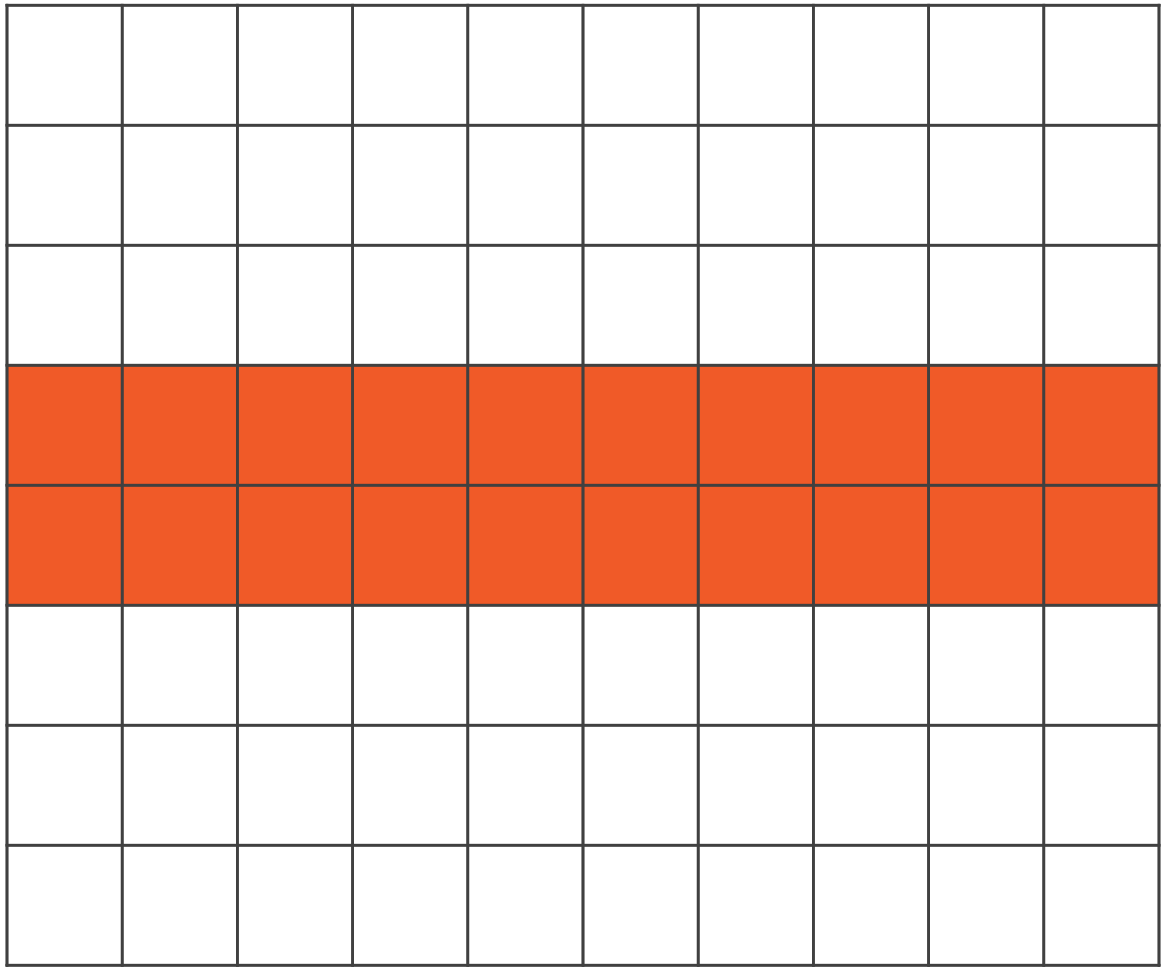
Uniform Distribution



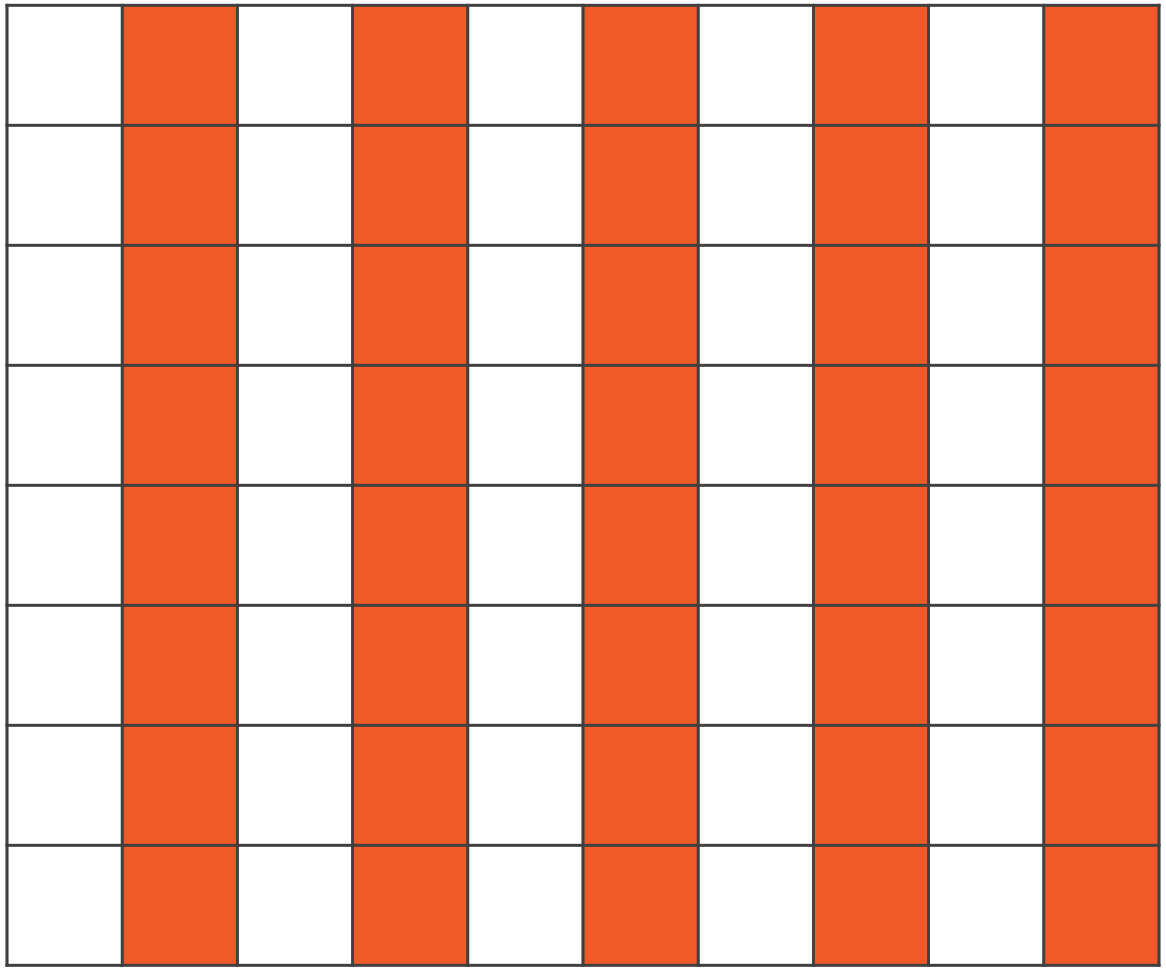
Uniform Distribution



Uniform Distribution



Uniform Distribution



Uniform Distribution

(More) Uniform

```
public uint SDBMHash(string input)
{
    uint hash = 0;

    foreach (byte ascii in input)
    {
        hash = hash * 65599 + ascii;
    }

    return hash;
}
```

Non-uniform

```
public int StableHash(string input)
{
    int result = 0;

    foreach(byte ascii in input)
    {
        result += ascii;
    }

    return result;
}
```

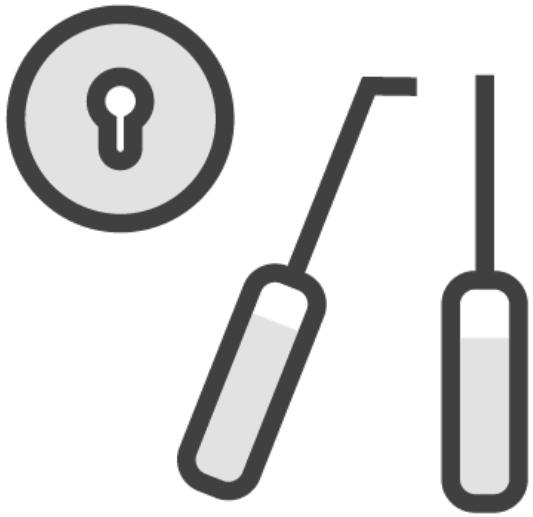
Uniform Distribution

(More) Uniform	Non-uniform
SDBMHash(”foo");	StableHash("foo");
SDBMHash(”oof");	StableHash(”oof");
SDBMHash(”ofo");	StableHash(”ofo");
849955110	324
924308646	324
923718264	324

Security

A secure hashing algorithm cannot be inverted (the input derived from the output hash).

Username	Password Hash (sdbm)
evelyn	3471203675
brian	969889485
will	1978836480

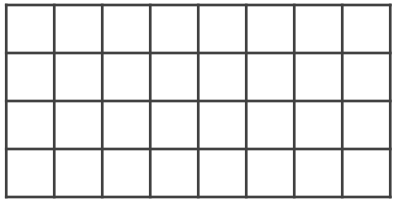


Password	Hash
aaaaaaaa	3834880256
aaaaaaab	3834880257
aaaaaaac	3834880258
aaaaaaad	3834880259
...	...
zzzzzzzz	528284160

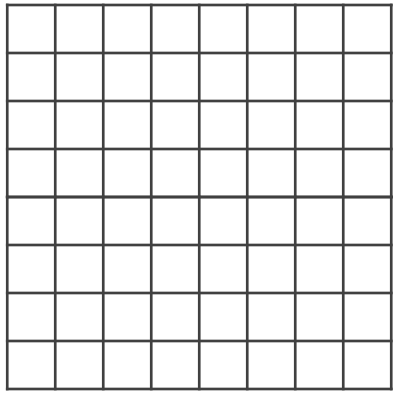
Username	Password Hash (sdbm)
evelyn	3471203675
brian	969889485
will	1978836480

Hash	Match	Actual
3471203675	aagkDhA4	password
969889485	aac0xWRs	football
1978836480	aaabhqCy	jedi

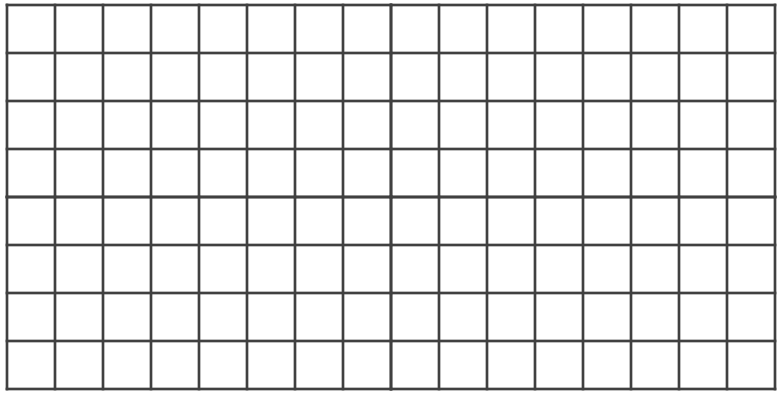
Output Size (32)



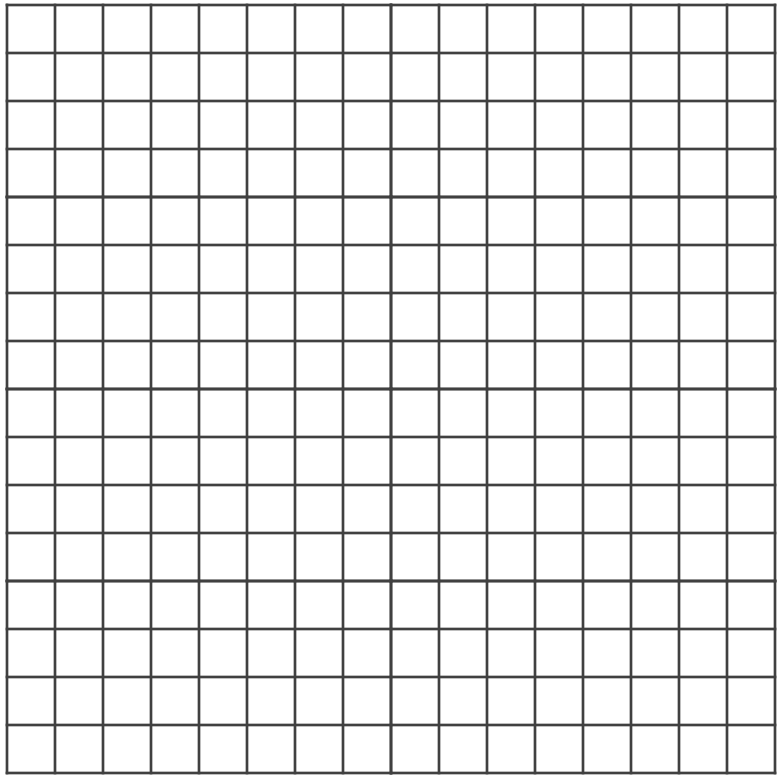
Output Size (64)



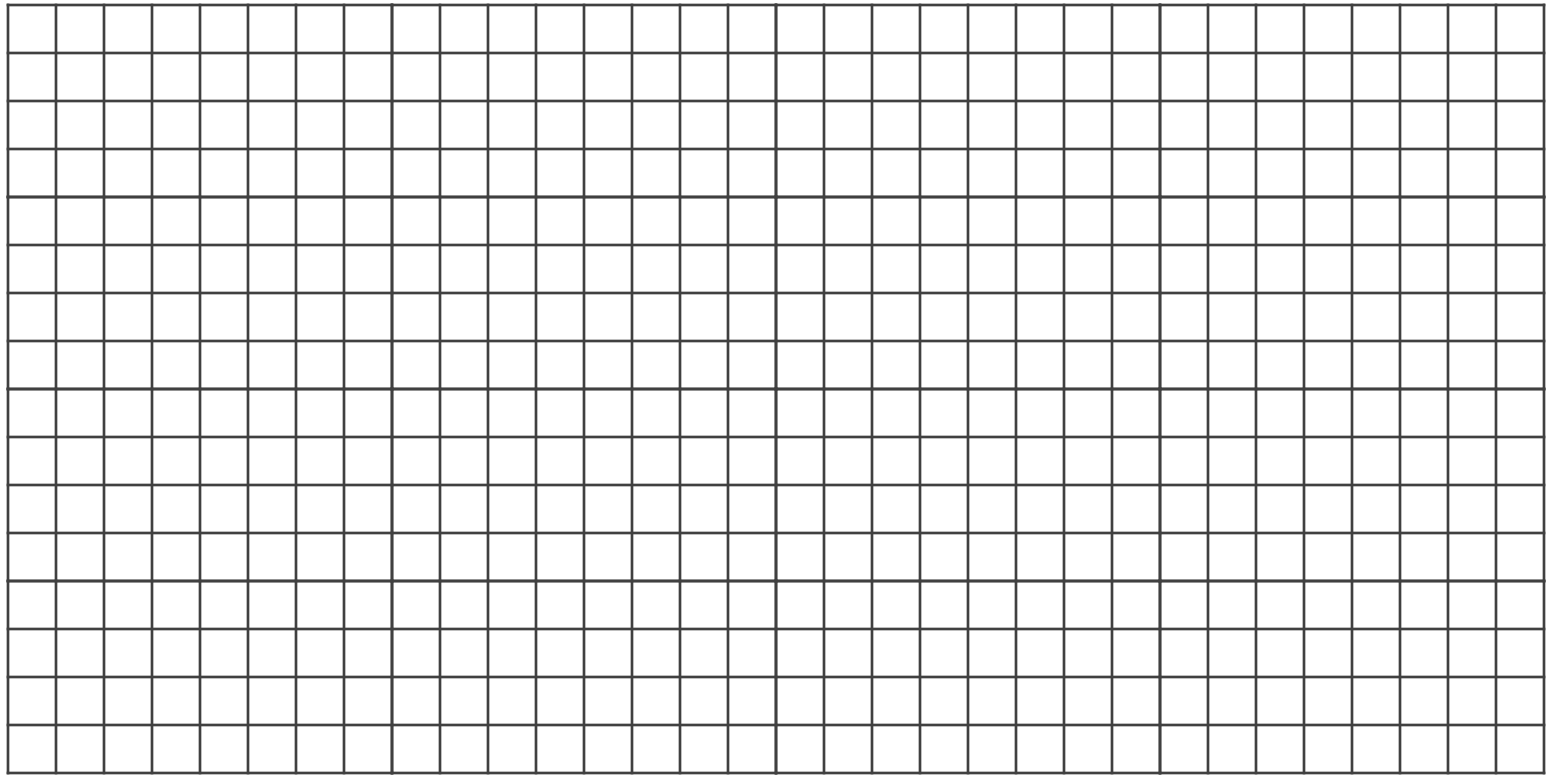
Output Size (128)



Output Size (256)



Output Size (512)



Hash Output Sizes

2^{32} (4294967296)

Checks/sec	Time
1000	49 Days
10,000	4.9 Days
100,000	12 hours
1,000,000	1 hour
10,000,000	7 minutes
100,000,000	42 seconds
1,000,000,000	4 seconds

2^{512} (1.340781e+154)

Checks/sec	Time
1000	4.25e+143 years
10,000	4.25e+142 years
100,000	4.25e+141 years
1,000,000	4.25e+140 years
10,000,000	4.25e+139 years
100,000,000	4.25e+138 years
1,000,000,000	4.25e+137 years

Sample Hash Algorithms

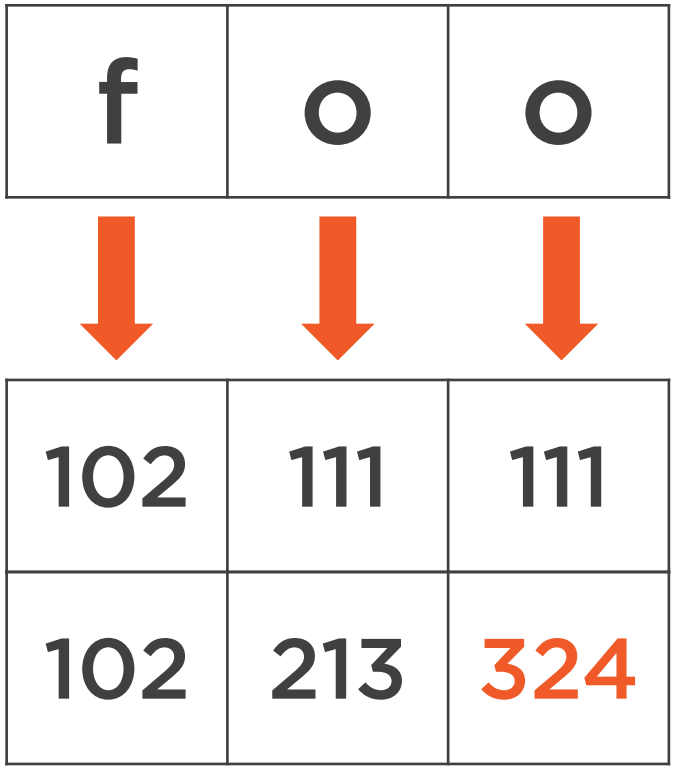
Additive Hash

Pros

- Stable
- Fast

Cons

- Poor uniformity
- Poor security



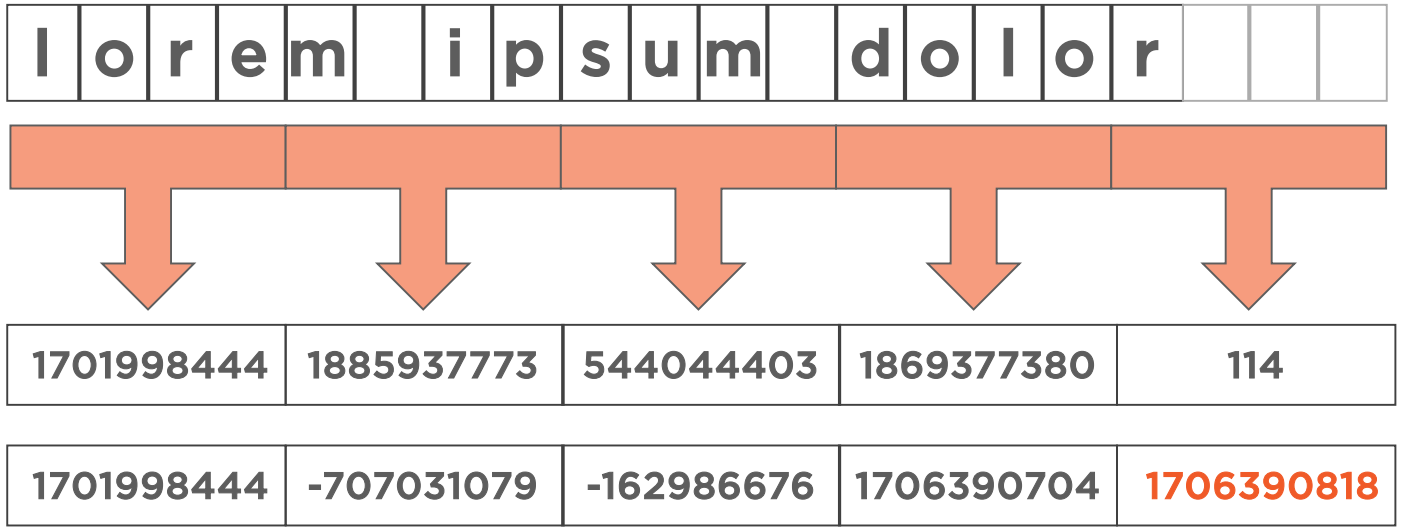
Folding Hash

Pros

- Stable
- Fast
- Better Uniformity

Cons

- Poor Security

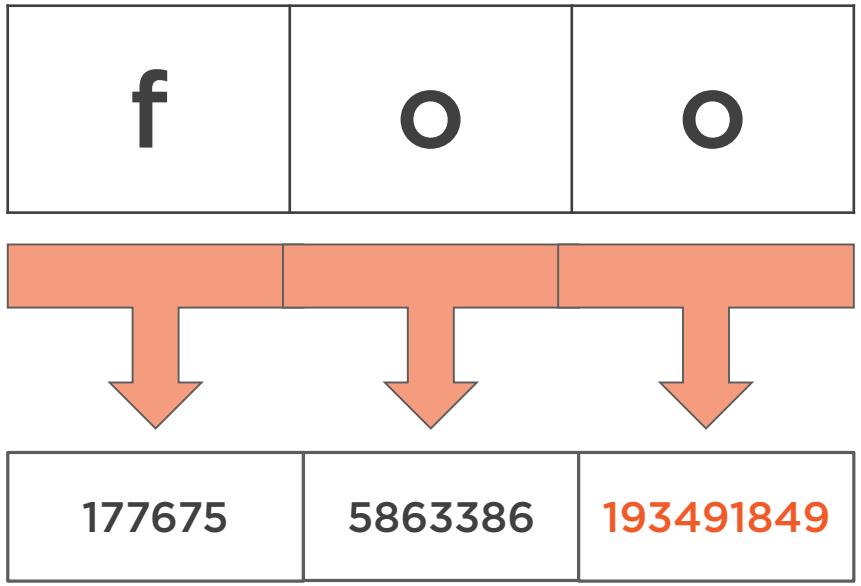


Dbj2 Hash

```
public ulong Dbj2Hash(string input)
{
    ulong hash = 5381;

    foreach(byte c in input)
    {
        hash = hash * 33 + c;
    }

    return hash;
}
```



Hash Function Comparison

Name	Output Size	Stable	Uniform	Secure
Additive	32	YES	NO	NO
Folding	32	YES	YES	NO
Dbj2	64	YES	YES	NO
MD5	128	YES	YES	*NO**
SHA-1	160	YES	YES	*NO**
SHA-2	224/384	YES	YES	*NO**
SHA-2	256-512	YES	YES	YES

**Once considered secure, this hash should no longer be used for secure applications.*

Adding Items

```
public class HashTable<TKey, TValue>
{
   TValue[] table = new TValue[4];

   private uint Hash(TKey key) { ... }

   public TValue this[TKey key]
   {
      get => table[Index(key)];
      set => table[Index(key)] = value;
   }
   private uint Index(TKey key)
   {
      return Hash(key) % table.Length;
   }
}
```

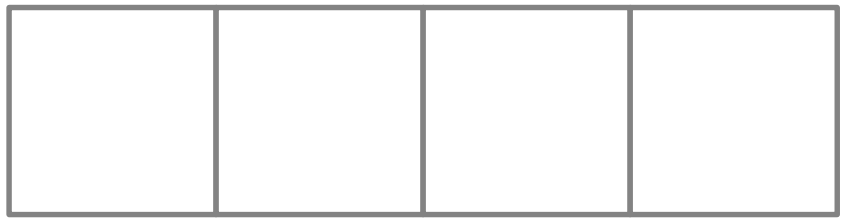
- Key and value type parameters
- Backing array defaults to size of 4
- A private hash function
- Functions to support indexed get and set operations using the key type
- Retrieve the value with that key
- Set the value with that key

```
HashTable<string, string> table = new HashTable<string, string>();


table["content-length"] = "8056";

table["content-type"] = "image/png";
```

Adding Values

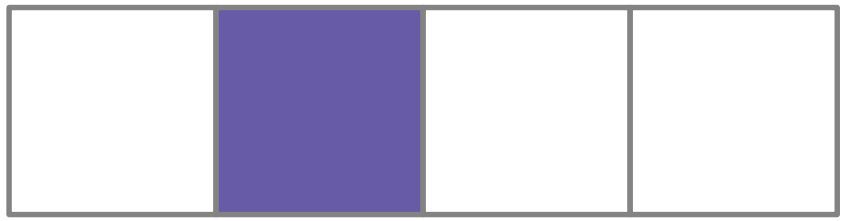


```
HashTable<string, string> table = new HashTable<string, string>();


table["content-length"] = "8056";

table["content-type"] = "image/png";
```

Adding Values

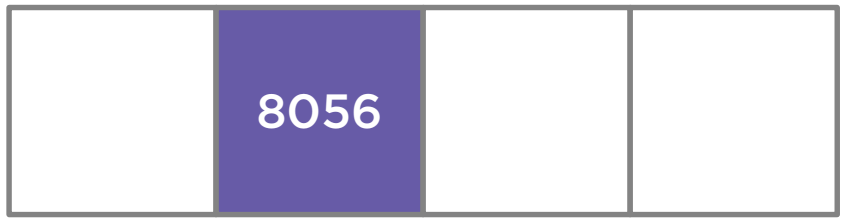


```
HashTable<string, string> table = new HashTable<string, string>();


table["content-length"] = "8056";

table["content-type"] = "image/png";
```

Adding Values

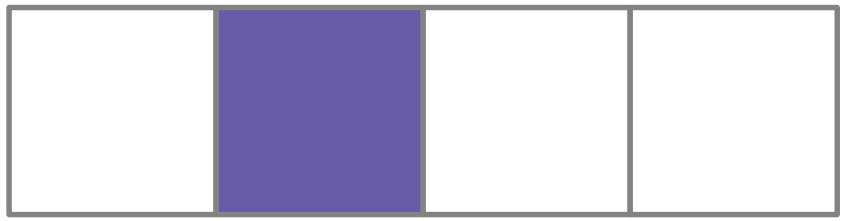


```
HashTable<string, string> table = new HashTable<string, string>();


table["content-length"] = "8056";

table["content-type"] = "image/png";
```

Adding Values

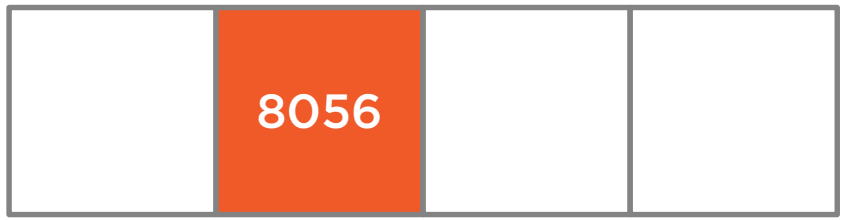


```
HashTable<string, string> table = new HashTable<string, string>();


table["content-length"] = "8056";

table["content-type"] = "image/png";
```

Adding Values



Hash Collision

When multiple distinct keys would be inserted at the same hash table index.

Separate Chaining

Collisions in a hash table are chained together into a linked list whose root node is the hash table array entry.

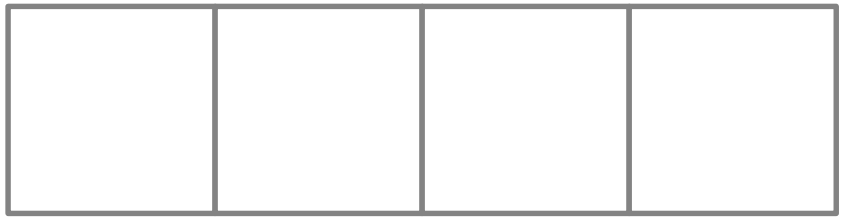
```
internal class HashTableEntry<TKey, TValue>
{
    public TKey Key;
    public TValue Value;
    public HashTableEntry<TKey, TValue> Next;
}
```

Separate Chaining

The hash table handles collisions by linking all the values with the same table index into a linked list of entries.

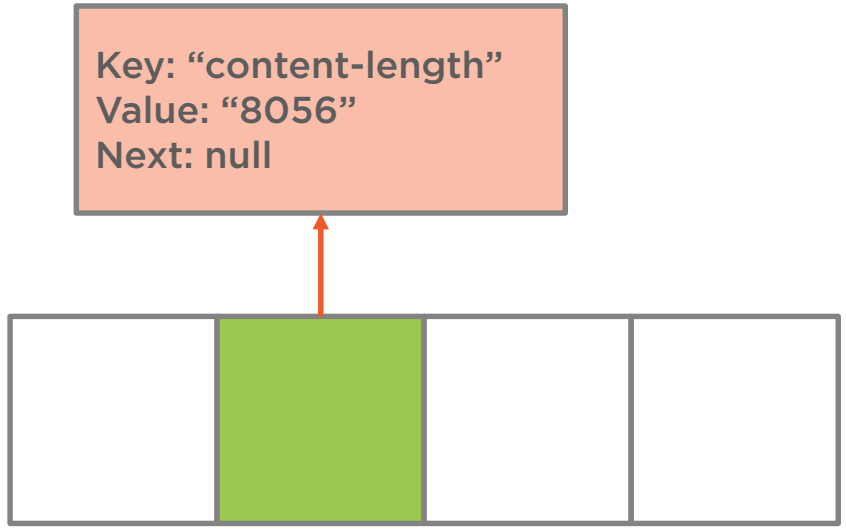
```
table["content-length"] = "8056";

table["content-type"] = "image/png";
```



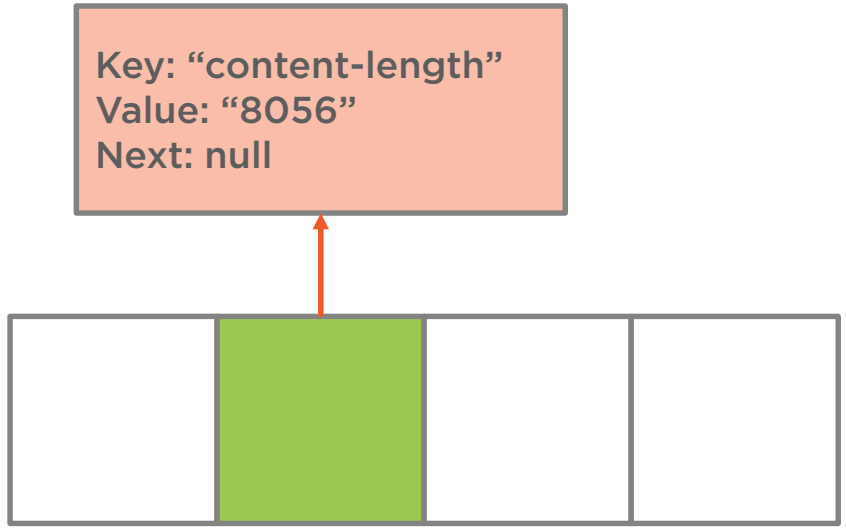
```
table["content-length"] = "8056";

table["content-type"] = "image/png";
```



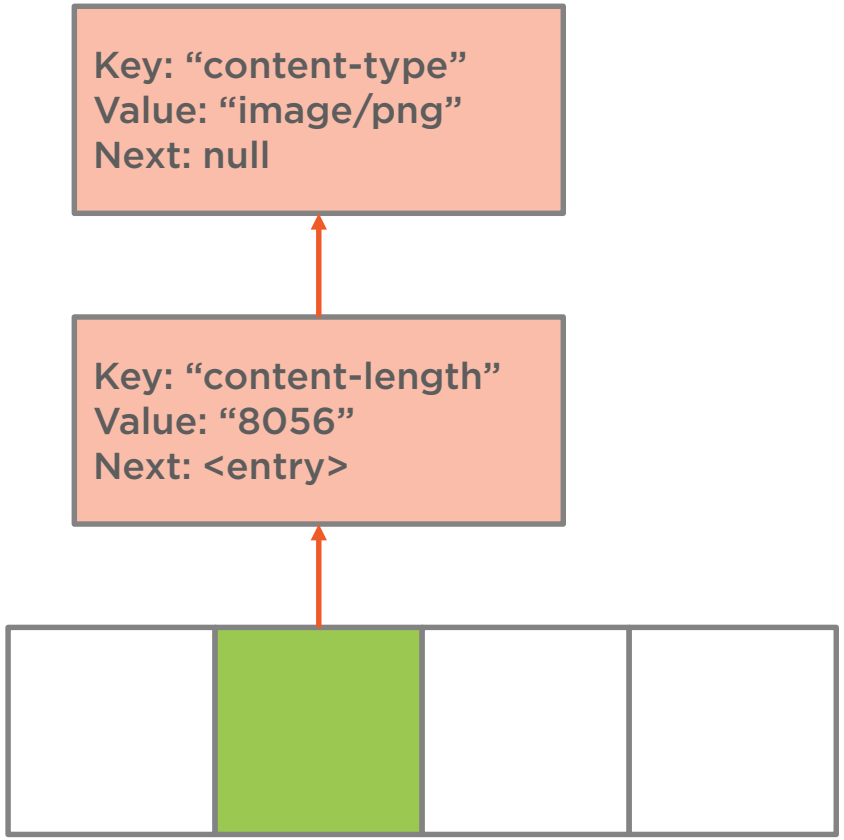
```
table["content-length"] = "8056";

table["content-type"] = "image/png";
```

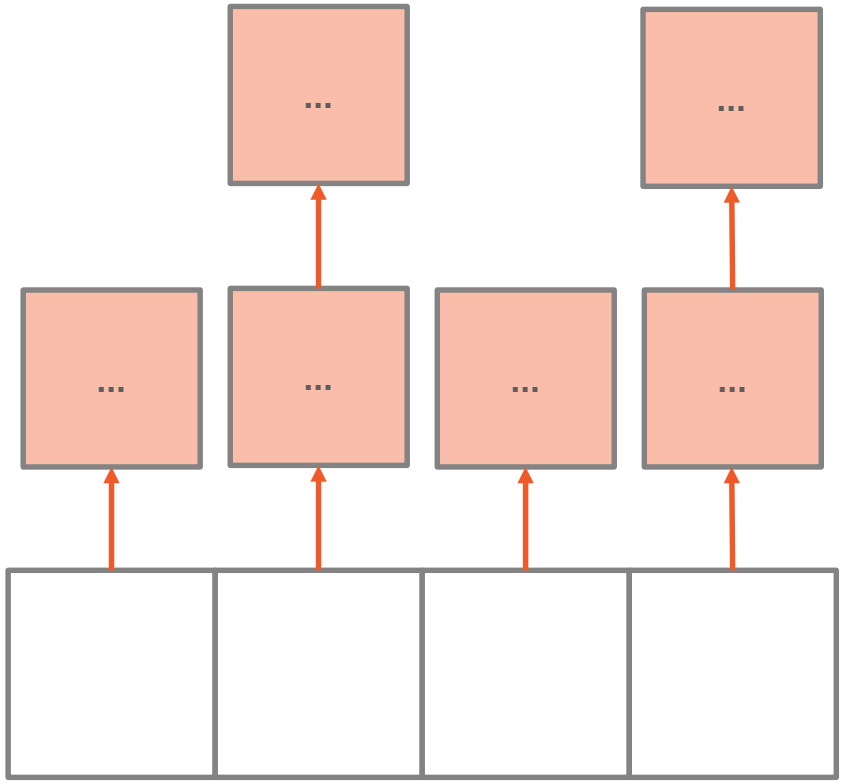


```
table["content-length"] = "8056";

table["content-type"] = "image/png";
```



Hash Table Collisions



Fill Factor

The percentage of capacity representing the maximum number of entries before the table will grow. E.g., 0.80

Growth Factor

The multiple to increase the capacity of the hash table when the fill factor has been exceeded. E.g., 1.50

Hash Table Growth

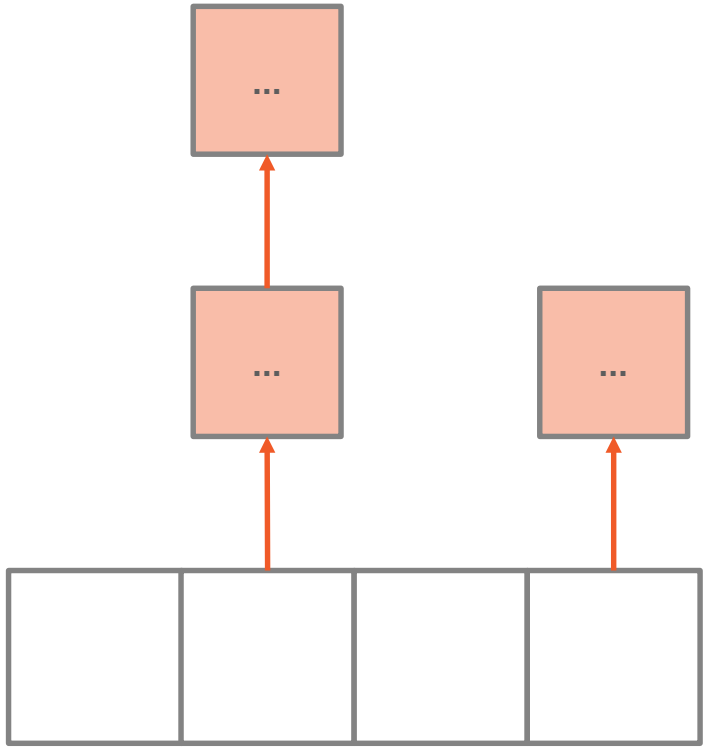
Use the fill factor to determine if growth is needed

Use the growth factor to allocate a larger array

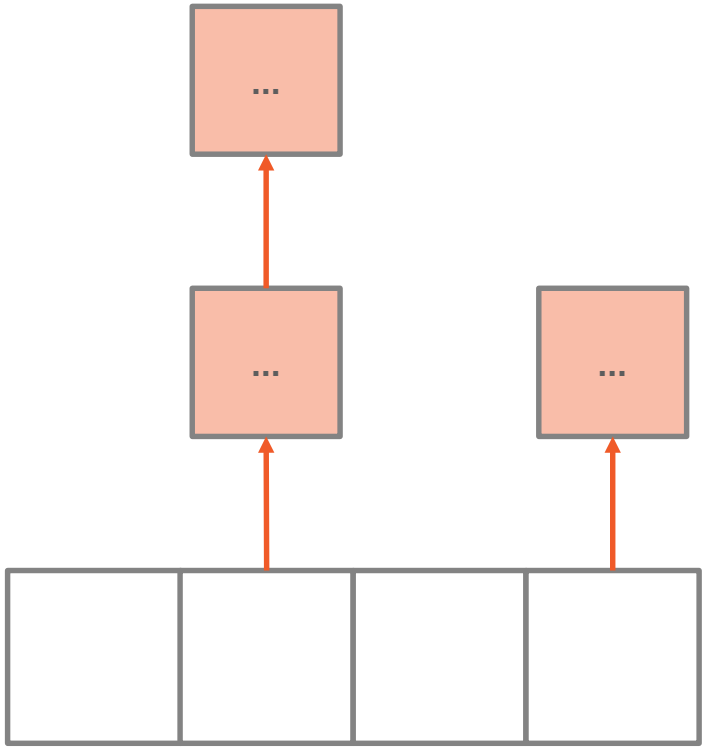
Determine the new index for the existing items in the hash table

Update the hash table to use the new array

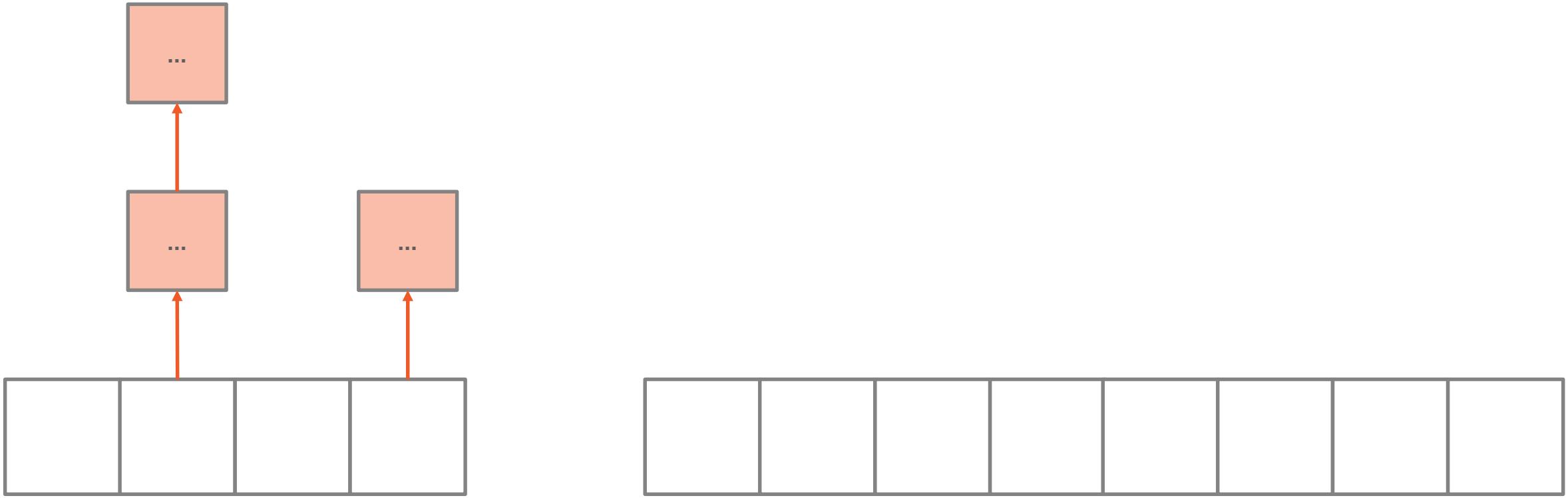
Hash Table Growth



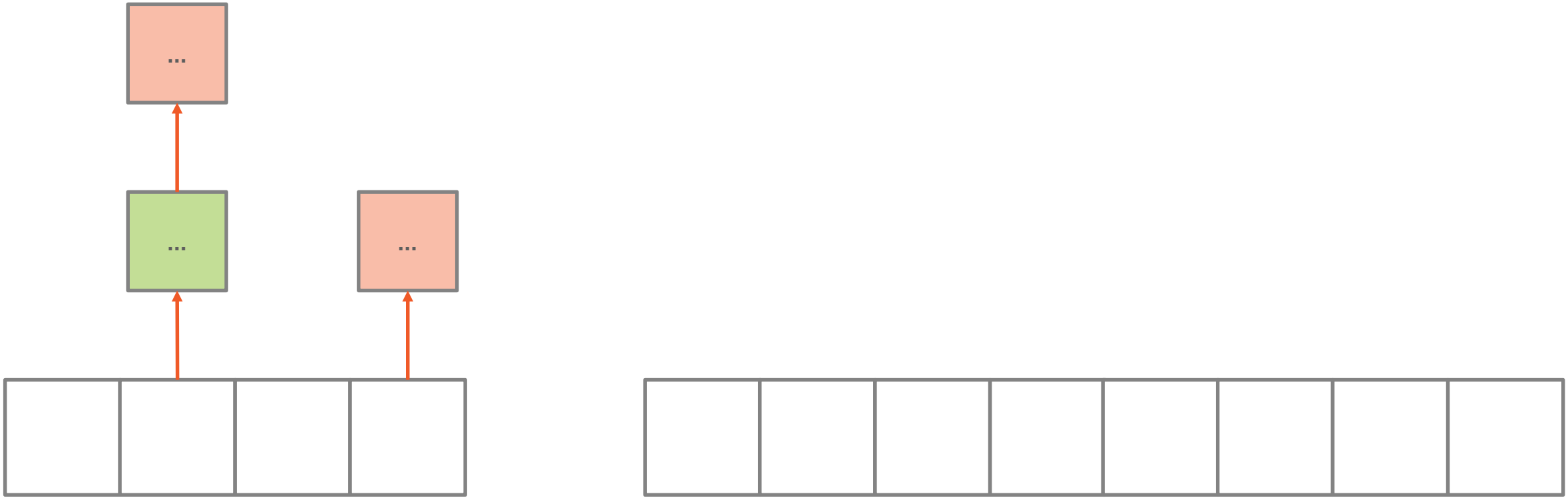
Hash Table Growth



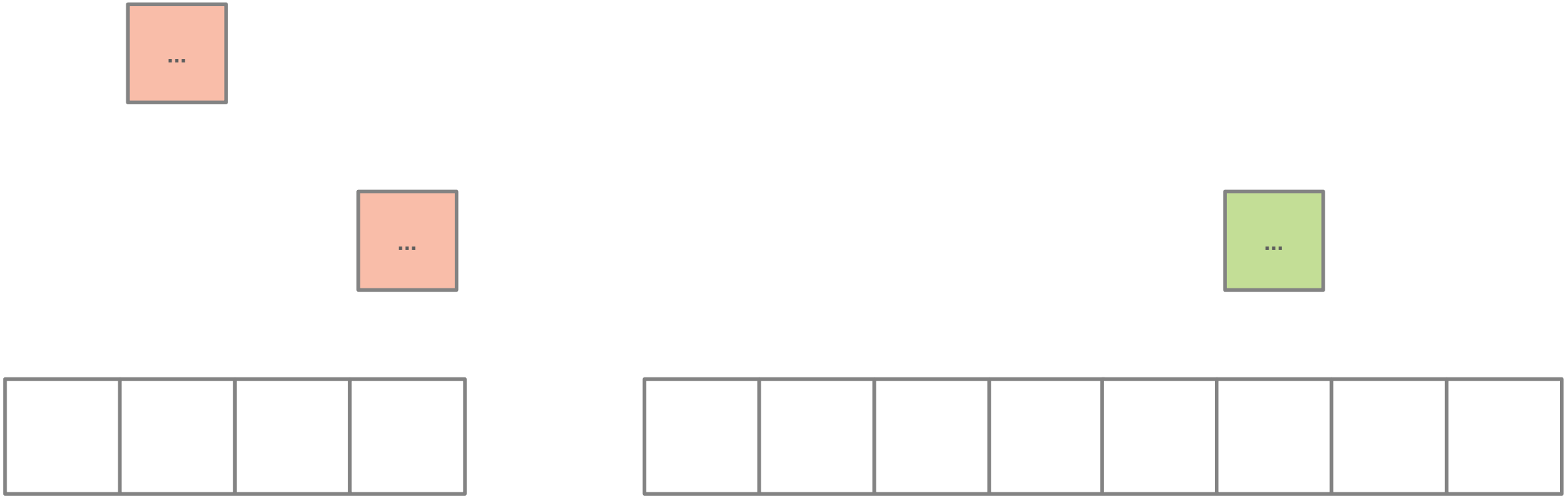
Hash Table Growth



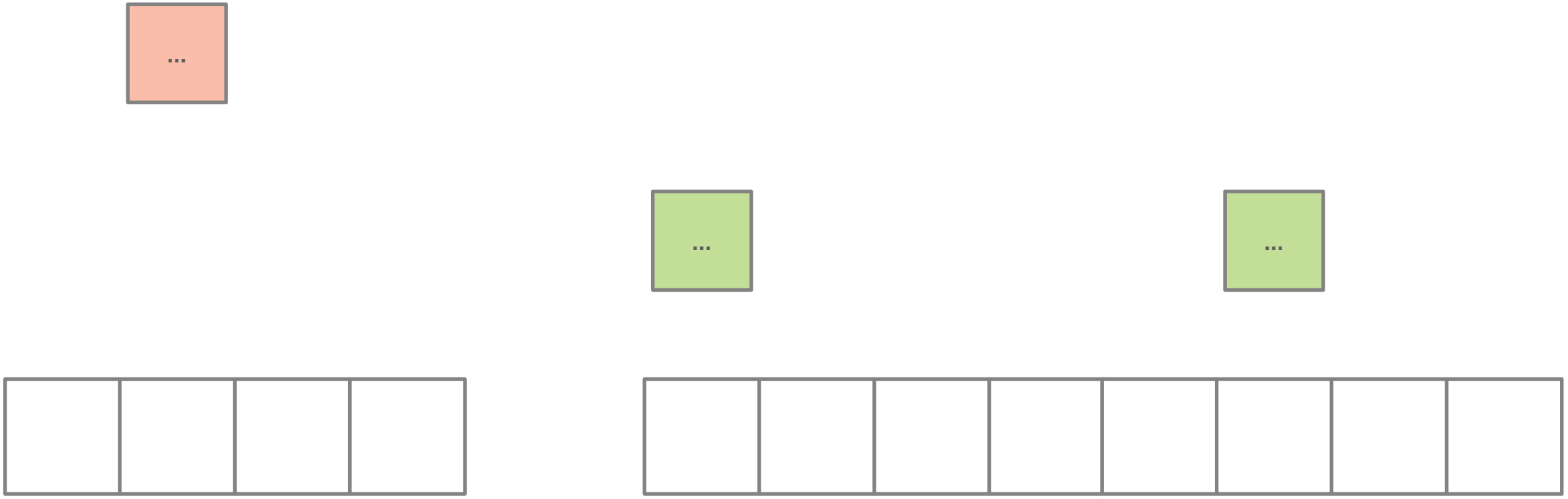
Hash Table Growth



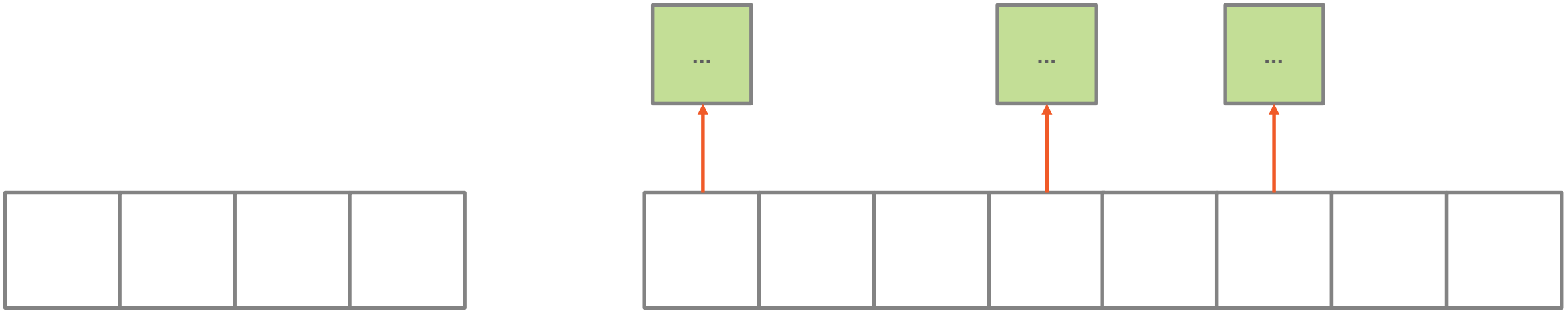
Hash Table Growth



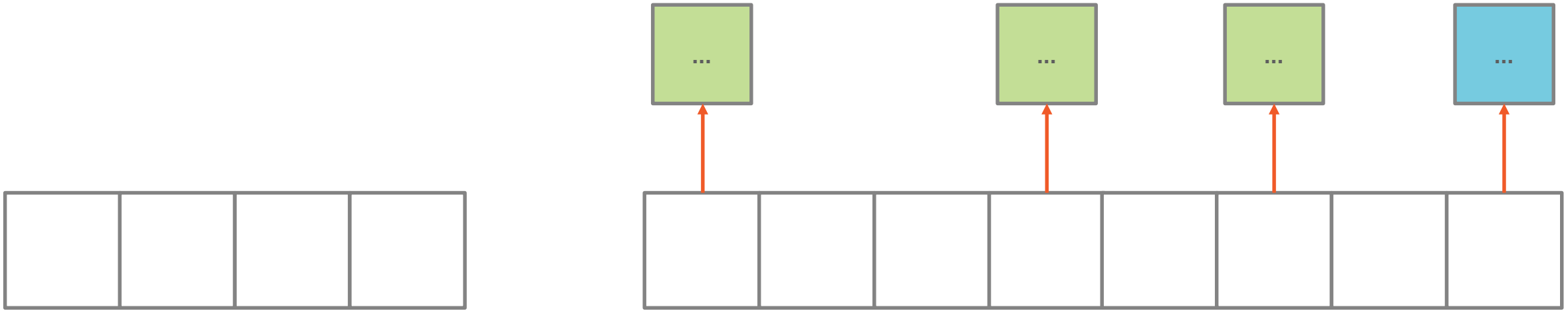
Hash Table Growth



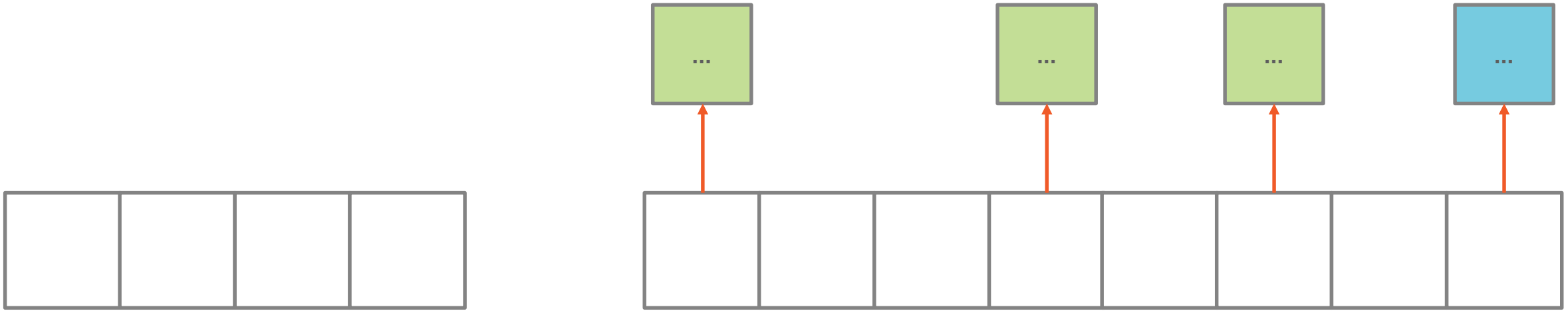
Hash Table Growth



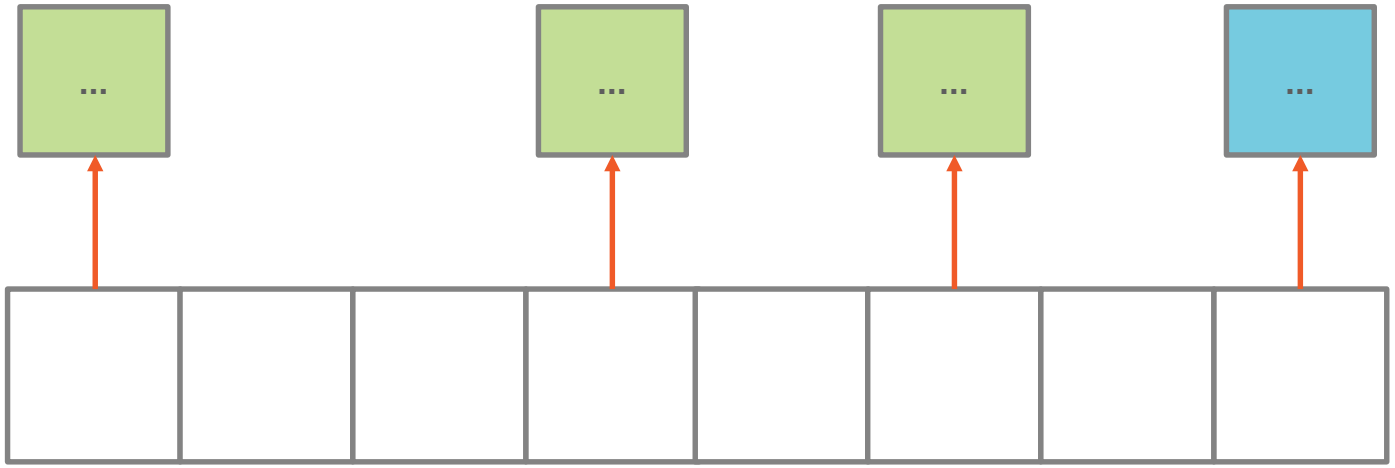
Hash Table Growth



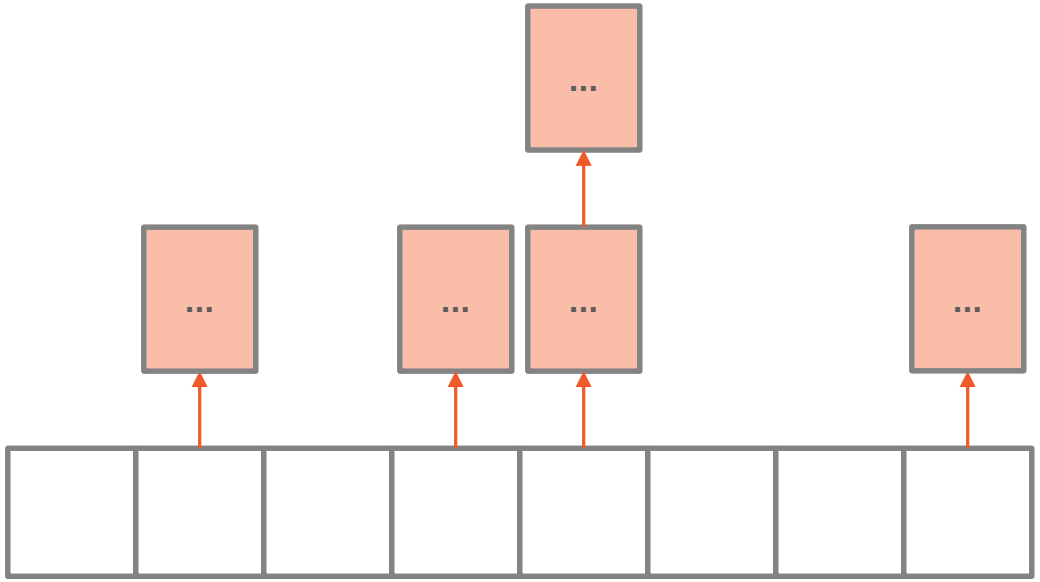
Hash Table Growth



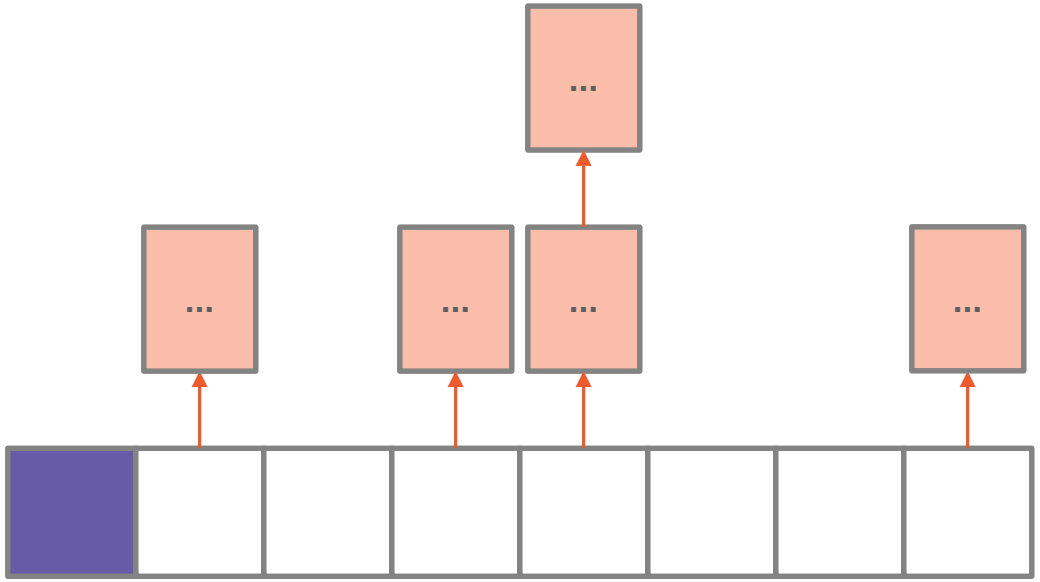
Hash Table Growth



Iteration

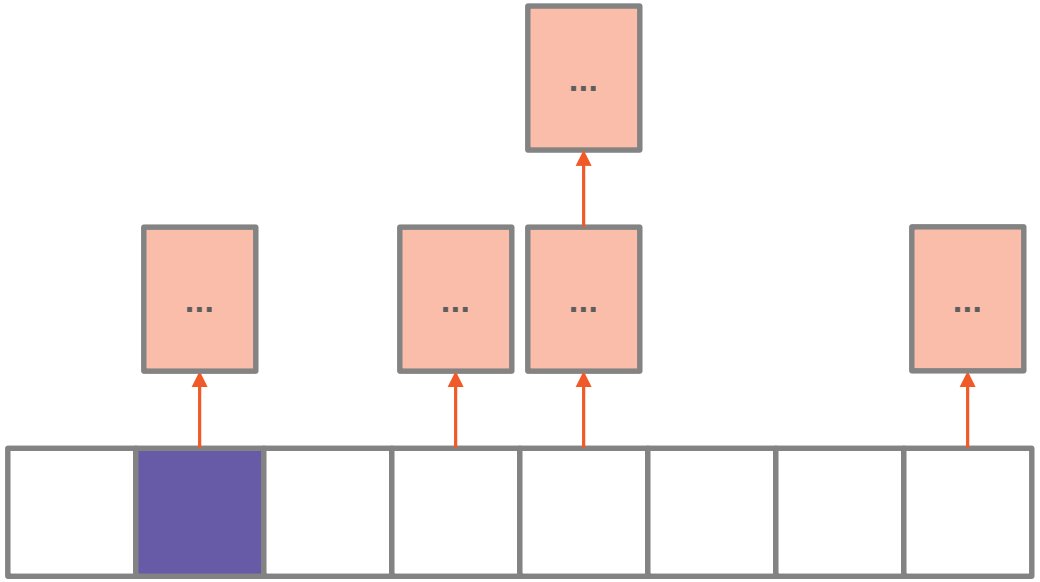


Begin at first index



Begin at first index

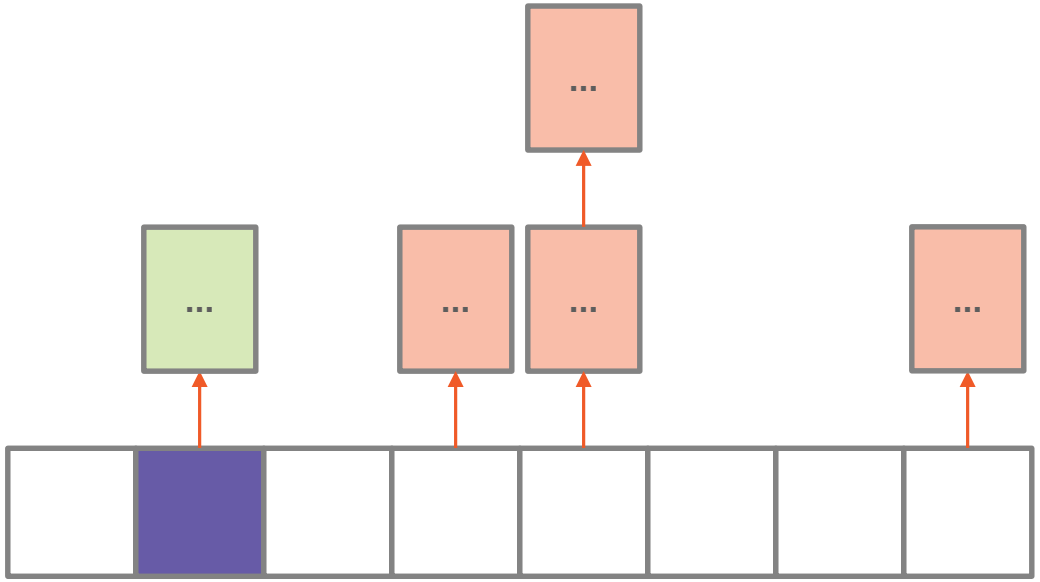
Visit each index



Begin at first index

Visit each index

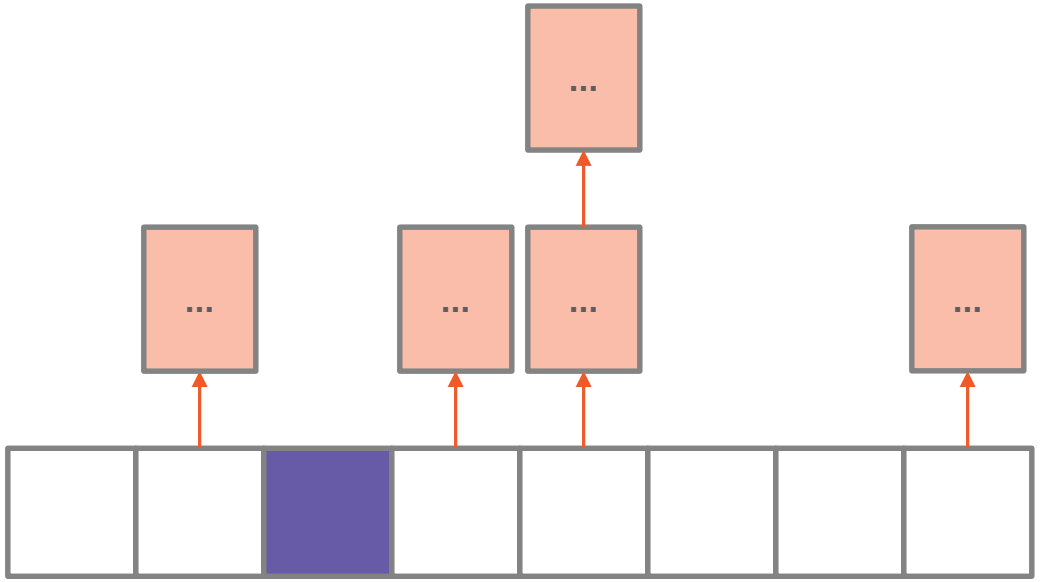
Visit each entry



Begin at first index

Visit each index

Visit each entry

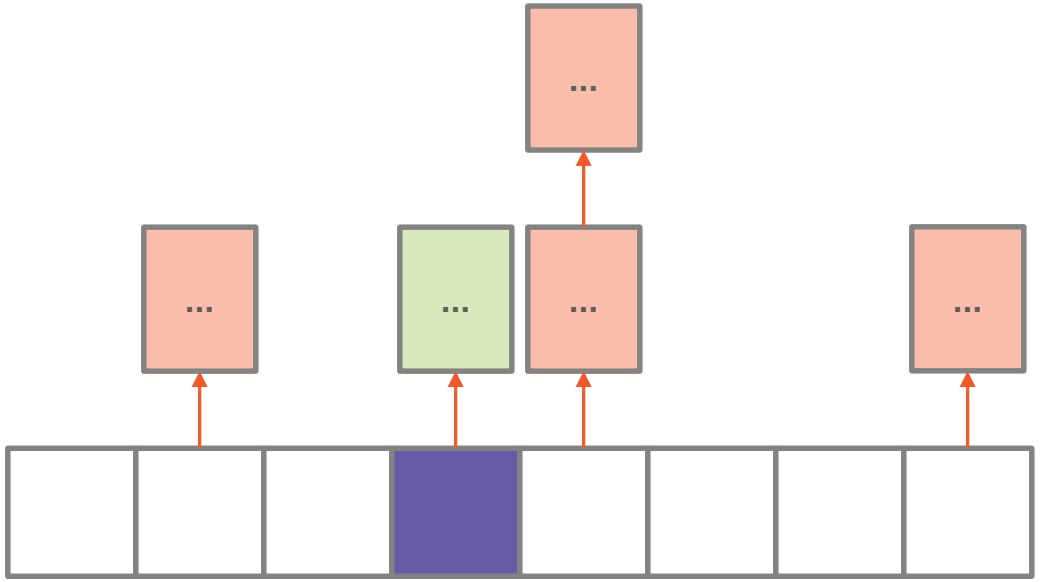


Begin at first index

Visit each index

Visit each entry

Continue to the end

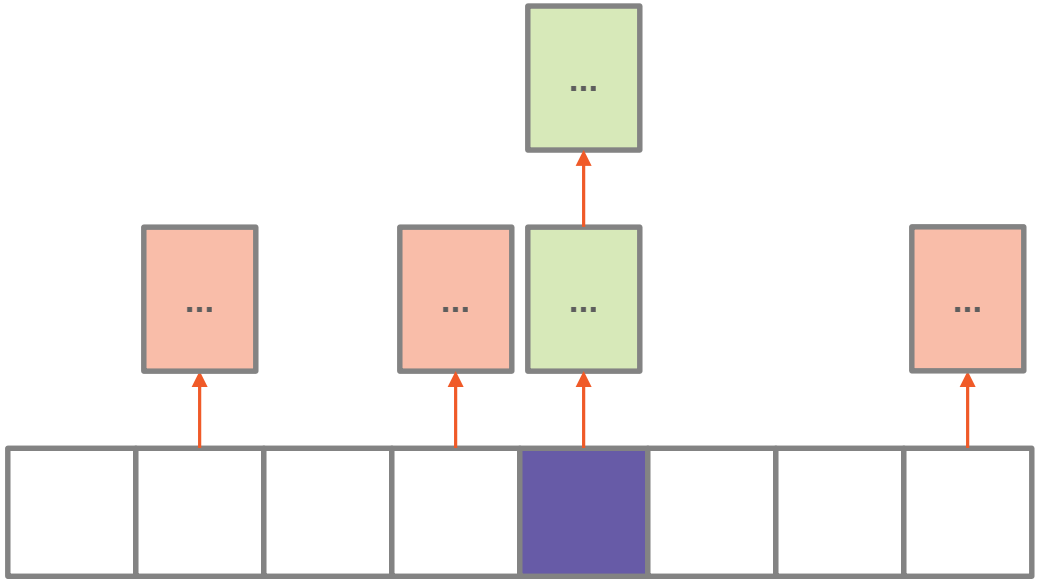


Begin at first index

Visit each index

Visit each entry

Continue to the end

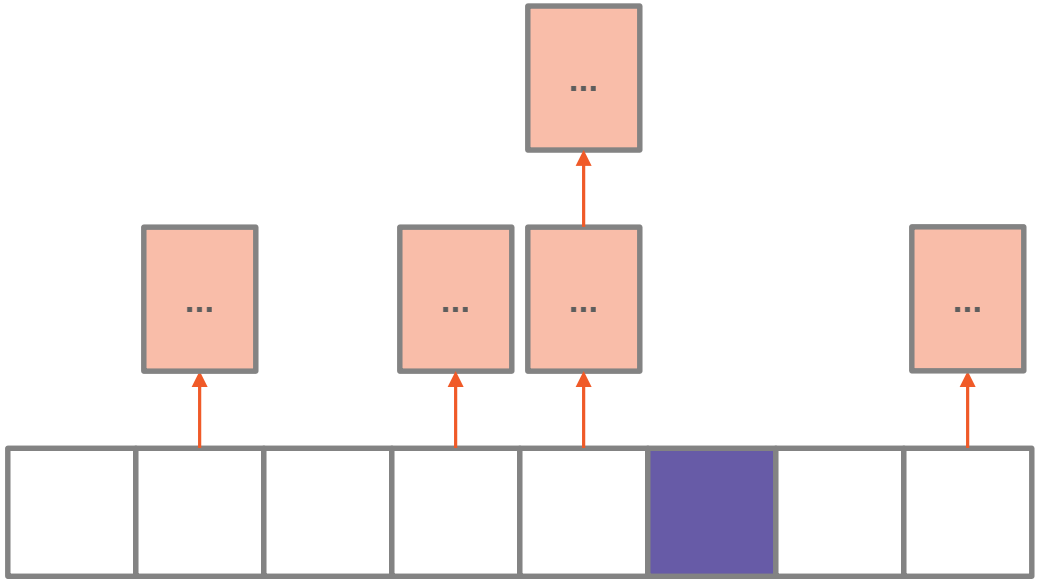


Begin at first index

Visit each index

Visit each entry

Continue to the end

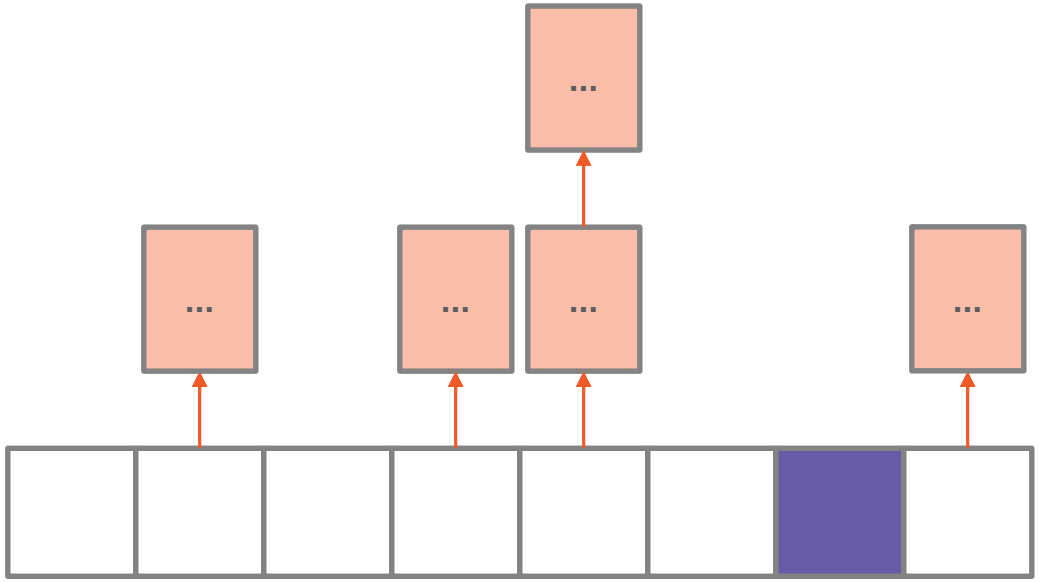


Begin at first index

Visit each index

Visit each entry

Continue to the end

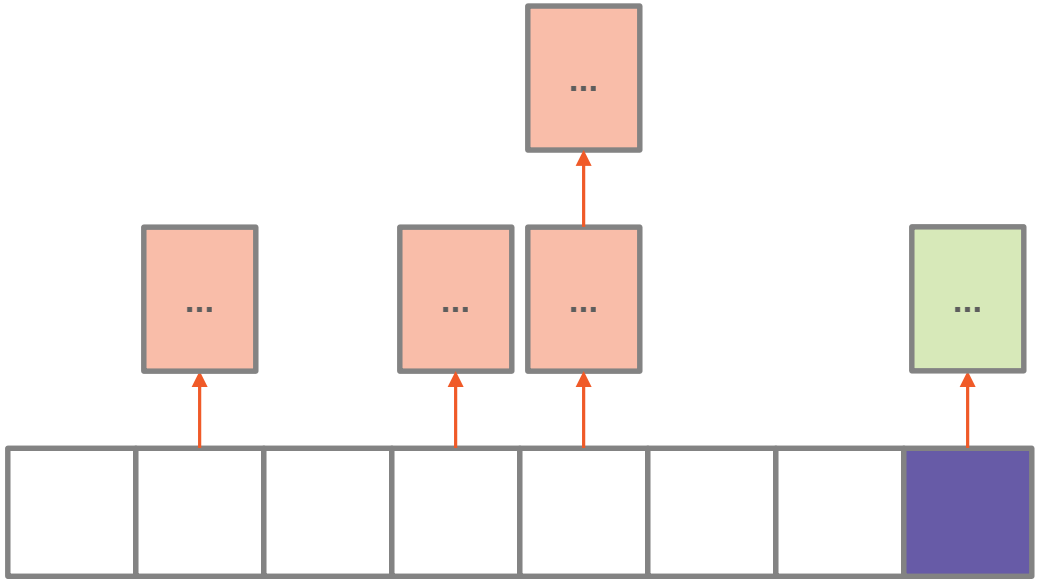


Begin at first index

Visit each index

Visit each entry

Continue to the end

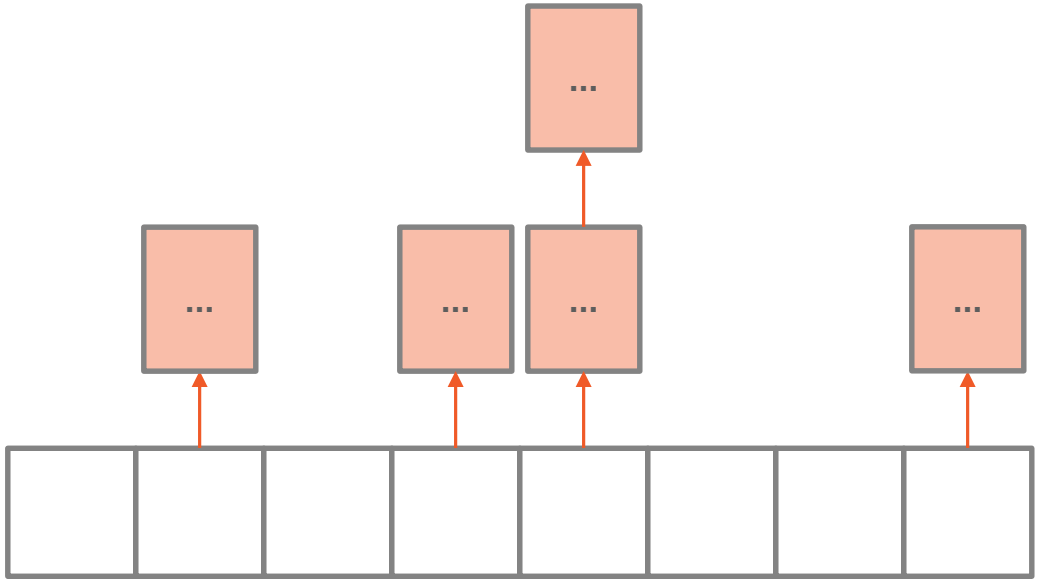


Begin at first index

Visit each index

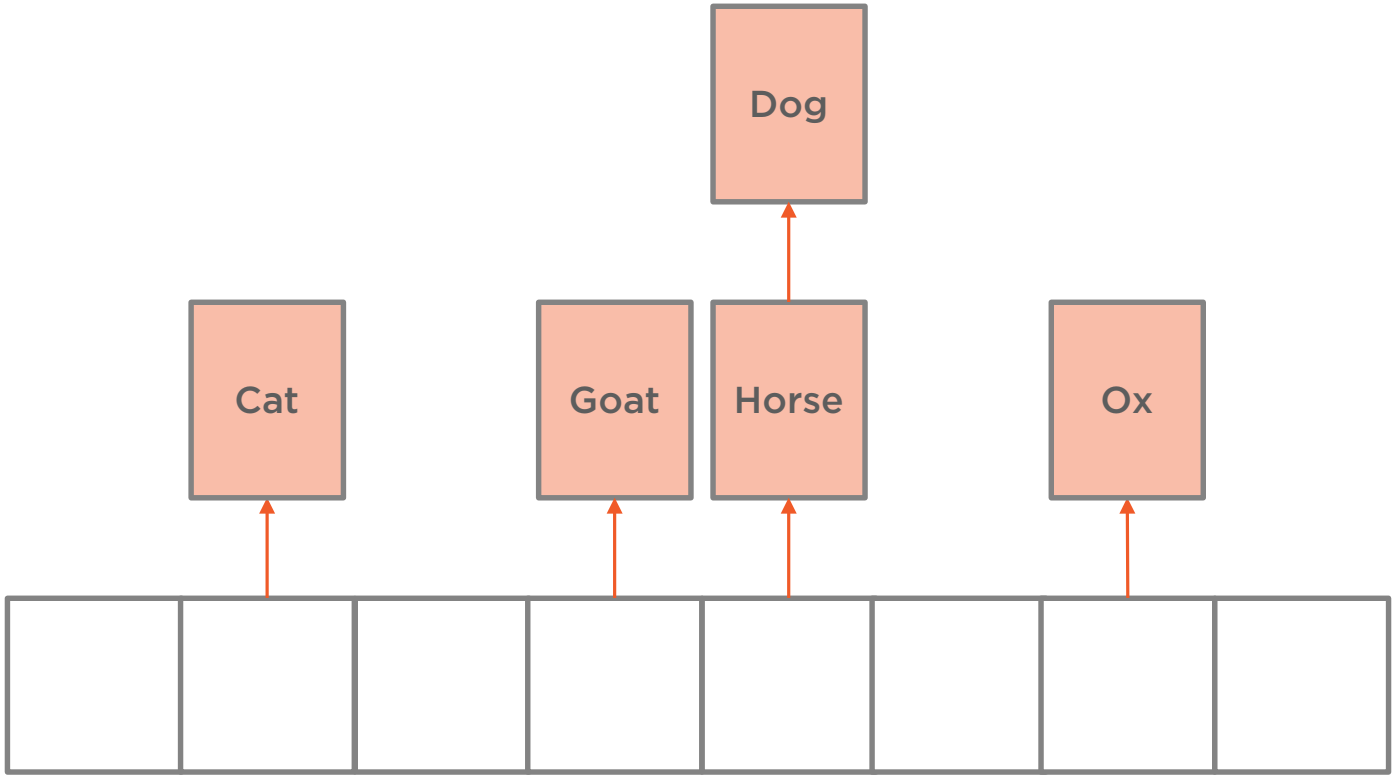
Visit each entry

Continue to the end

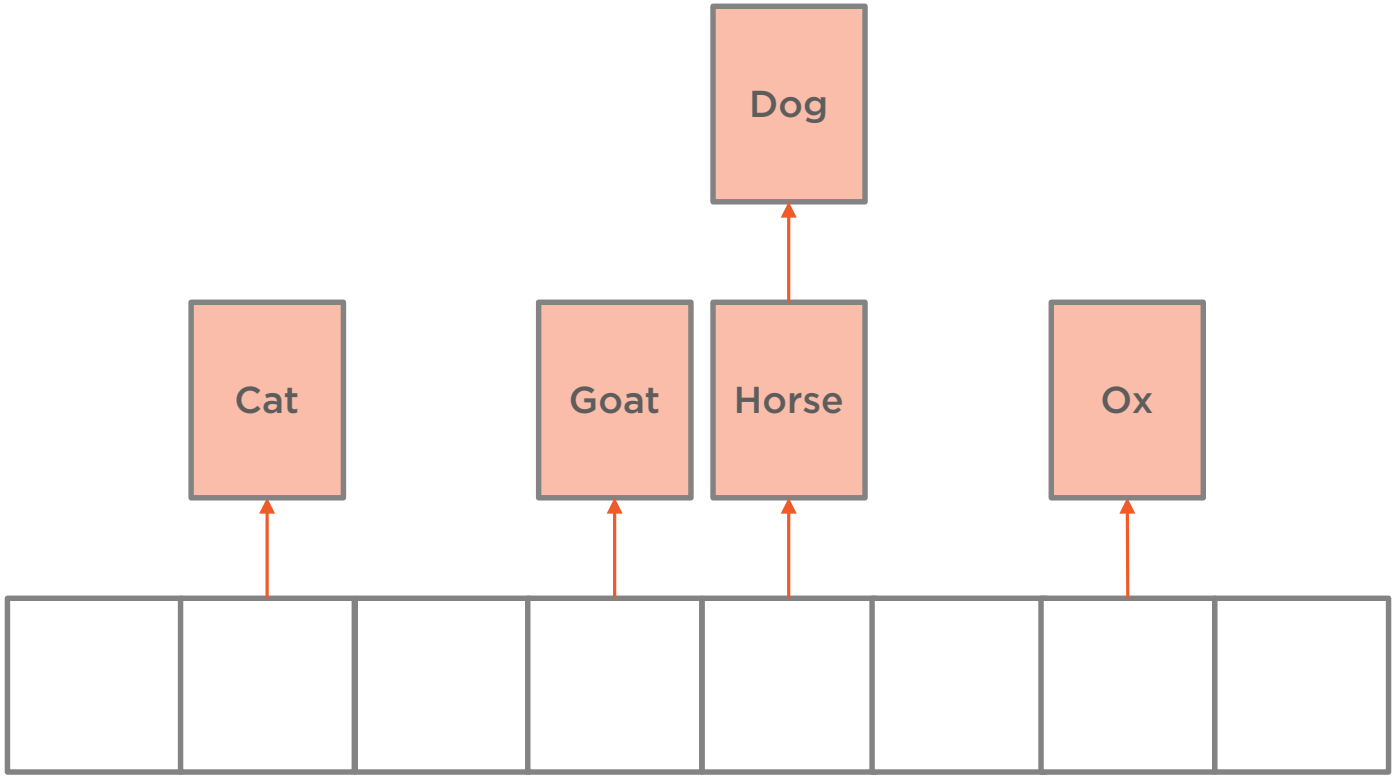


Items are iterated in the order they are stored in the table.

Finding Items

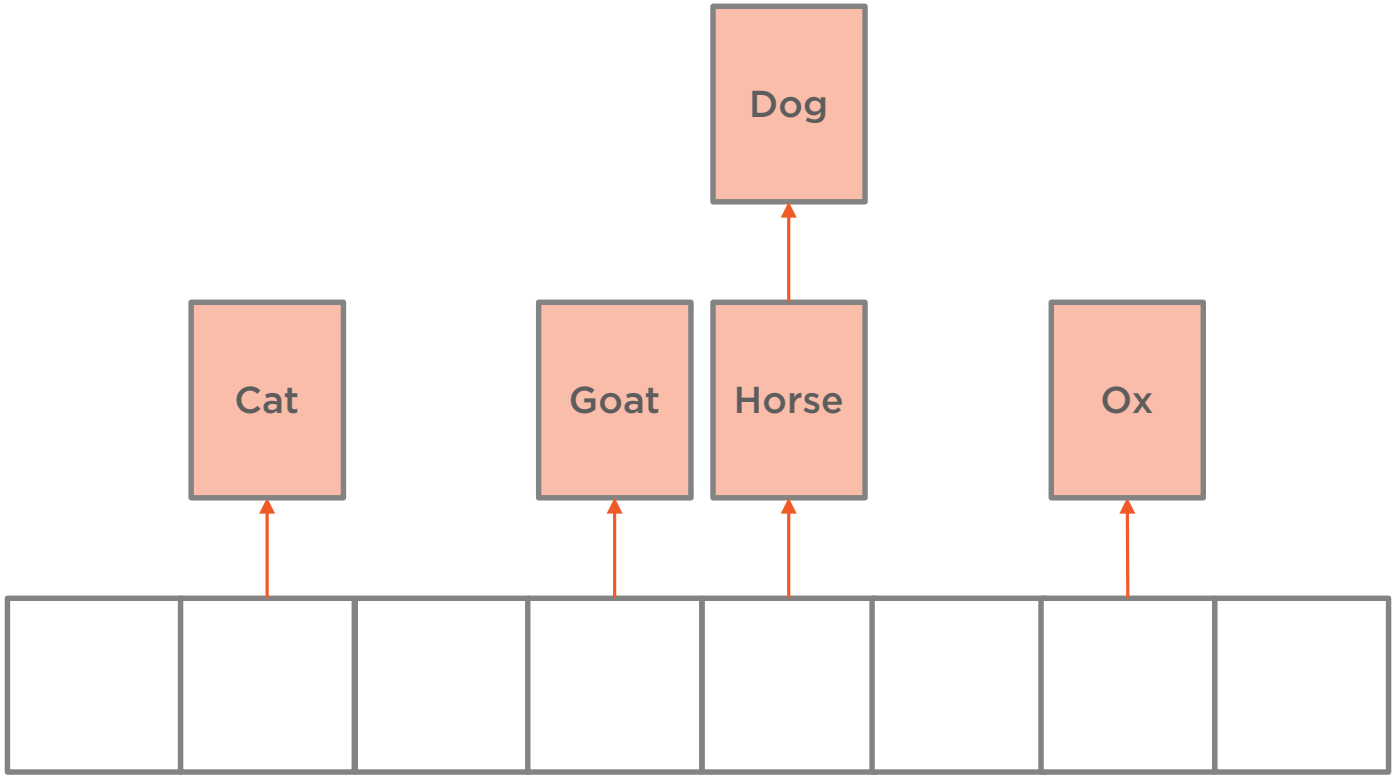

Dog
Cat
Goat
Horse
Ox

Find the index



Find the index

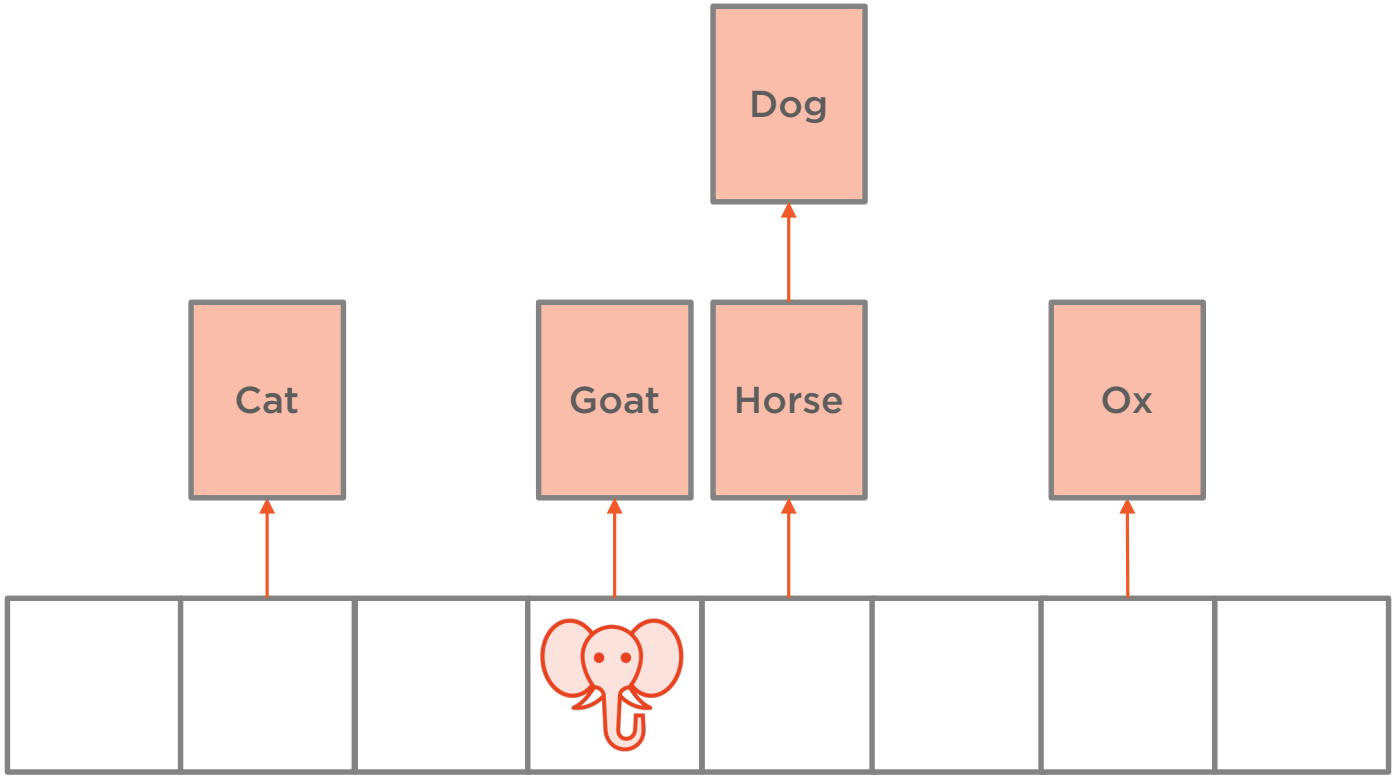
Use the hash function



Find the index

Use the hash function

Modulo the hash

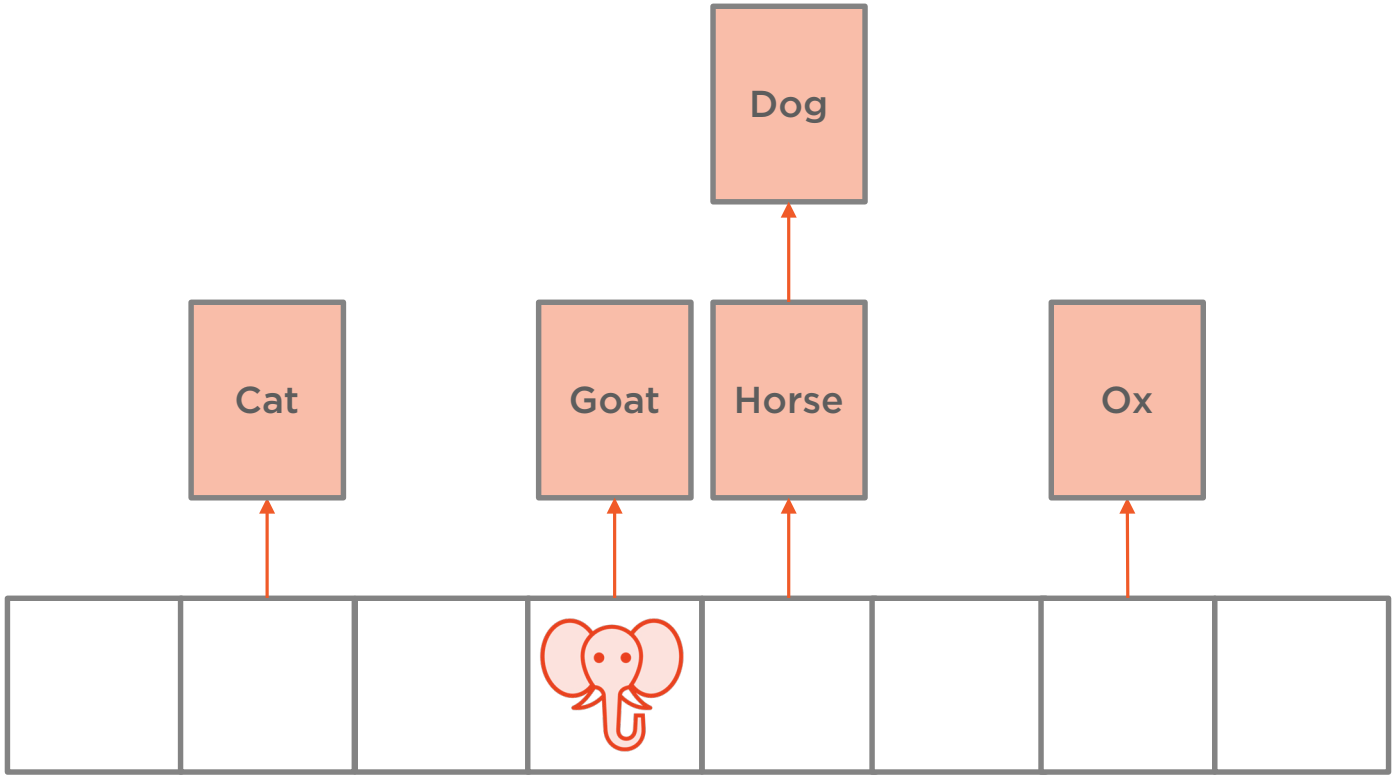


Find the index

Use the hash function

Modulo the hash

Search the entry list

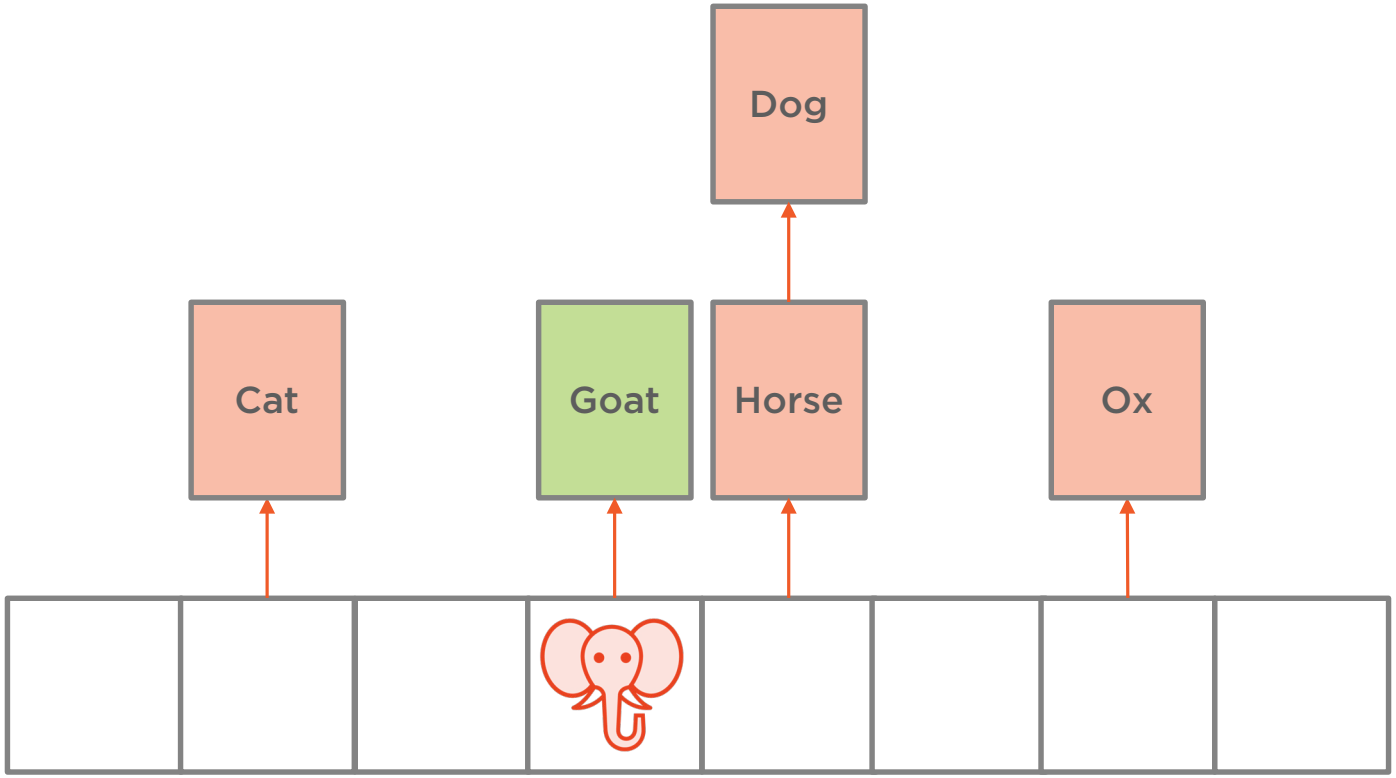


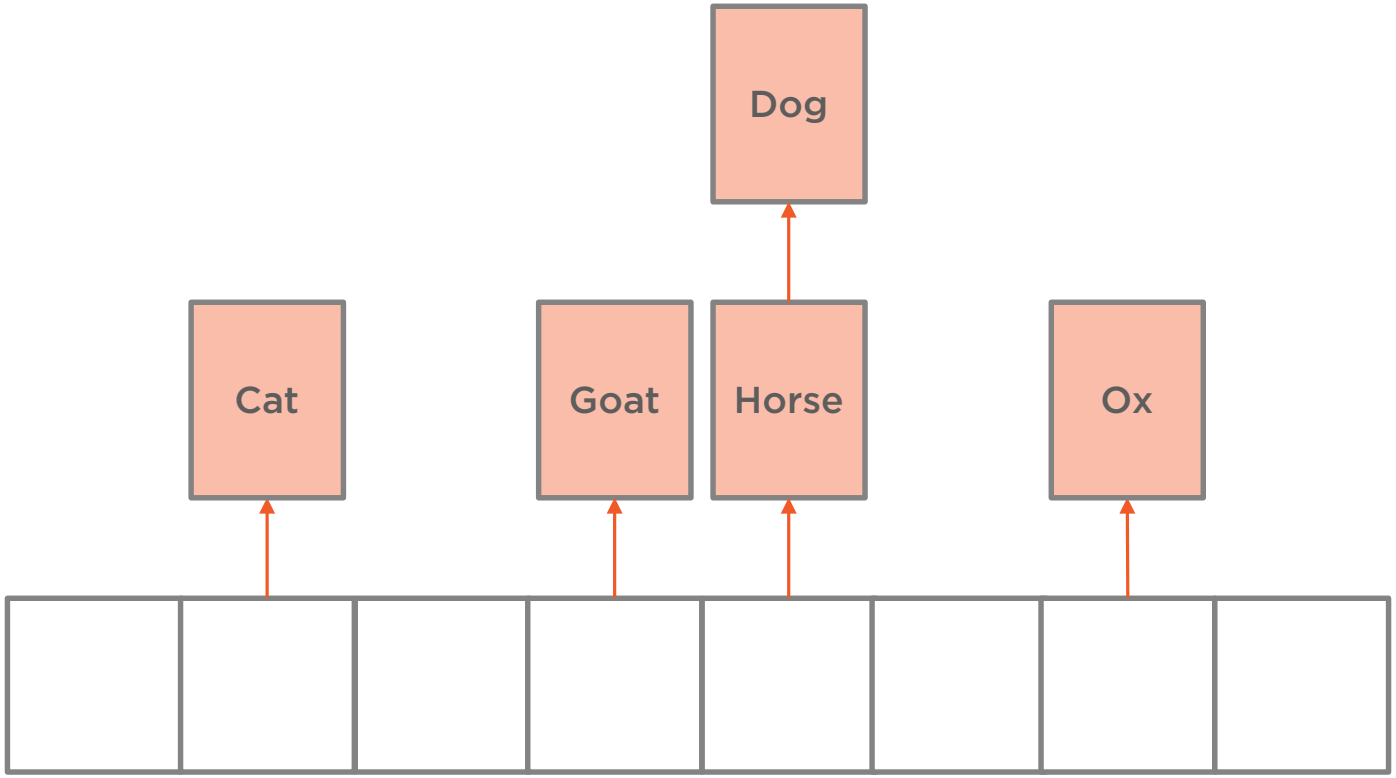
Find the index

Use the hash function

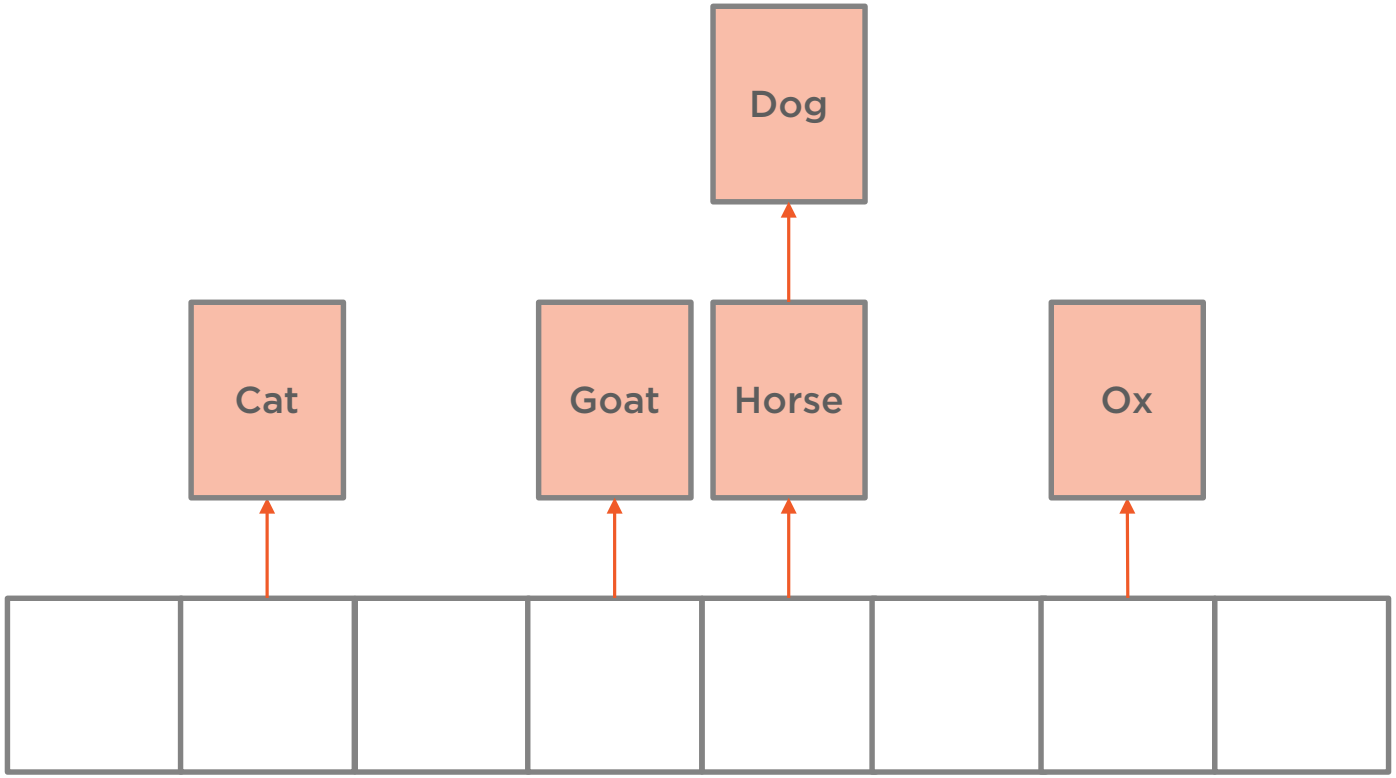
Modulo the hash

Search the entry list



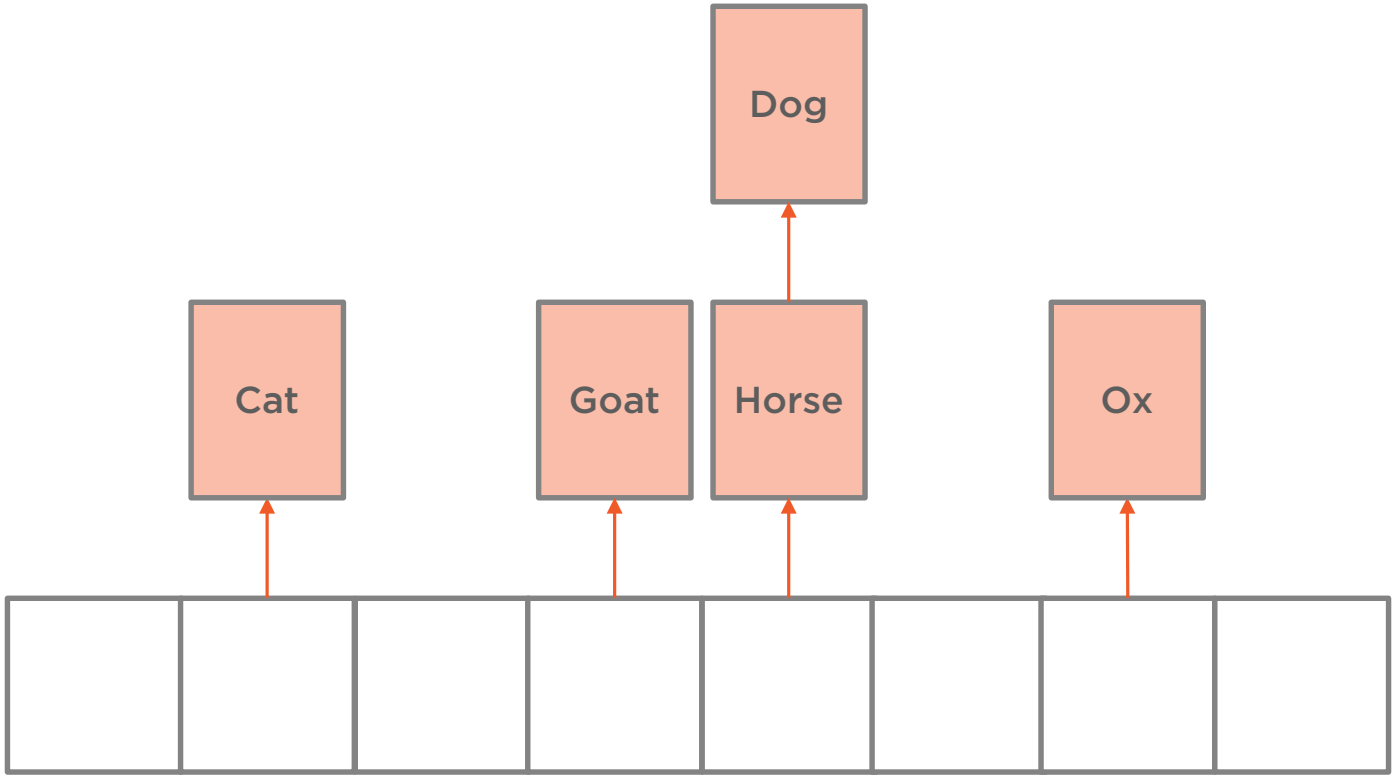

Dog
Cat
Goat
Horse
Ox

Find the index



Find the index

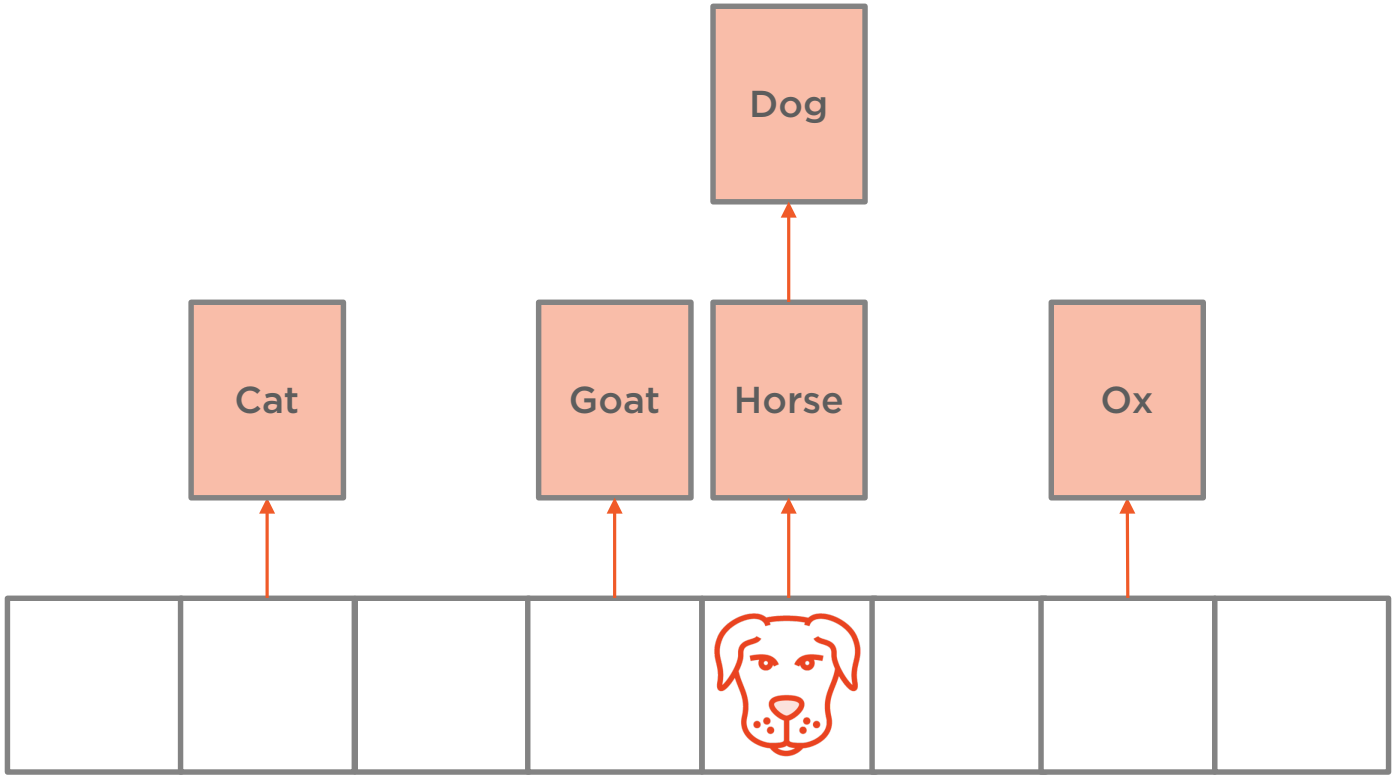
Use the hash function



Find the index

Use the hash function

Modulo the hash

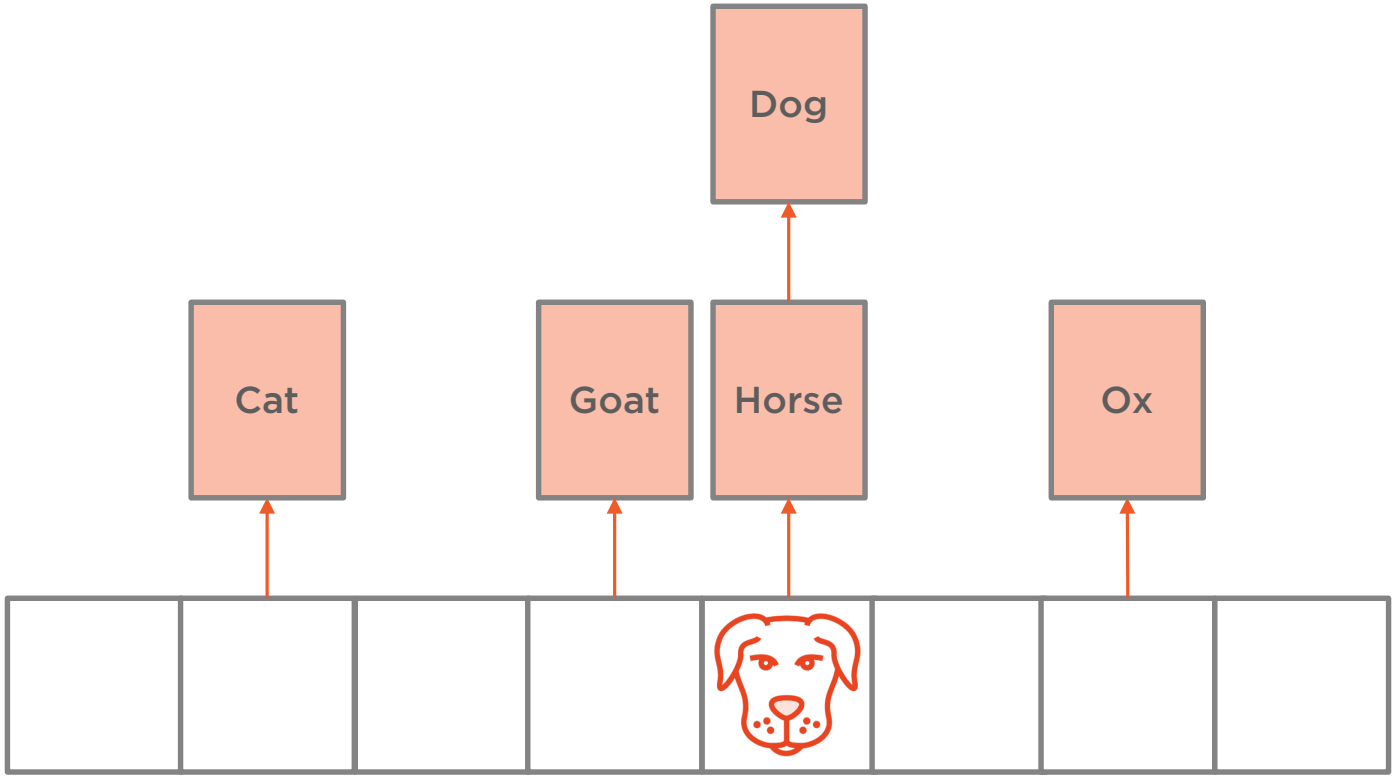


Find the index

Use the hash function

Modulo the hash

Search the entry list

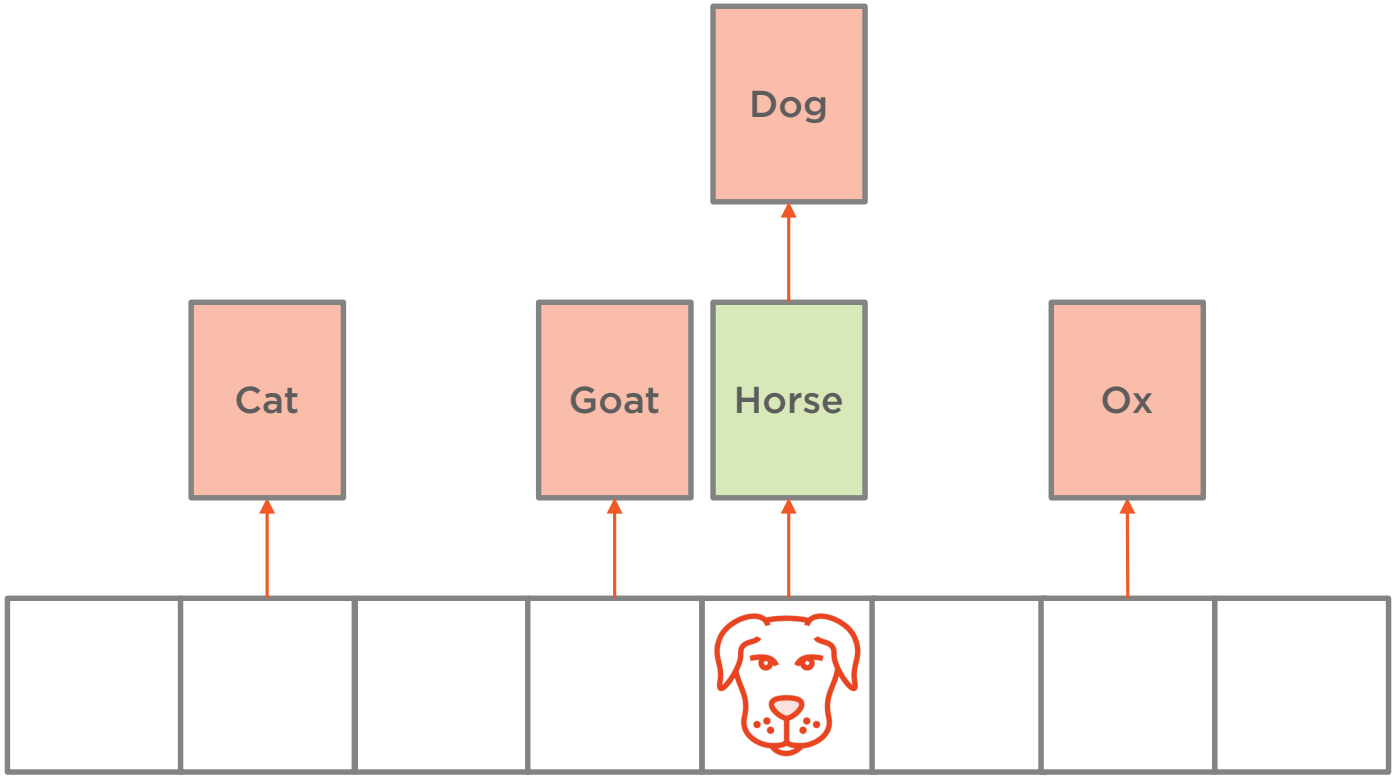


Find the index

Use the hash function

Modulo the hash

Search the entry list

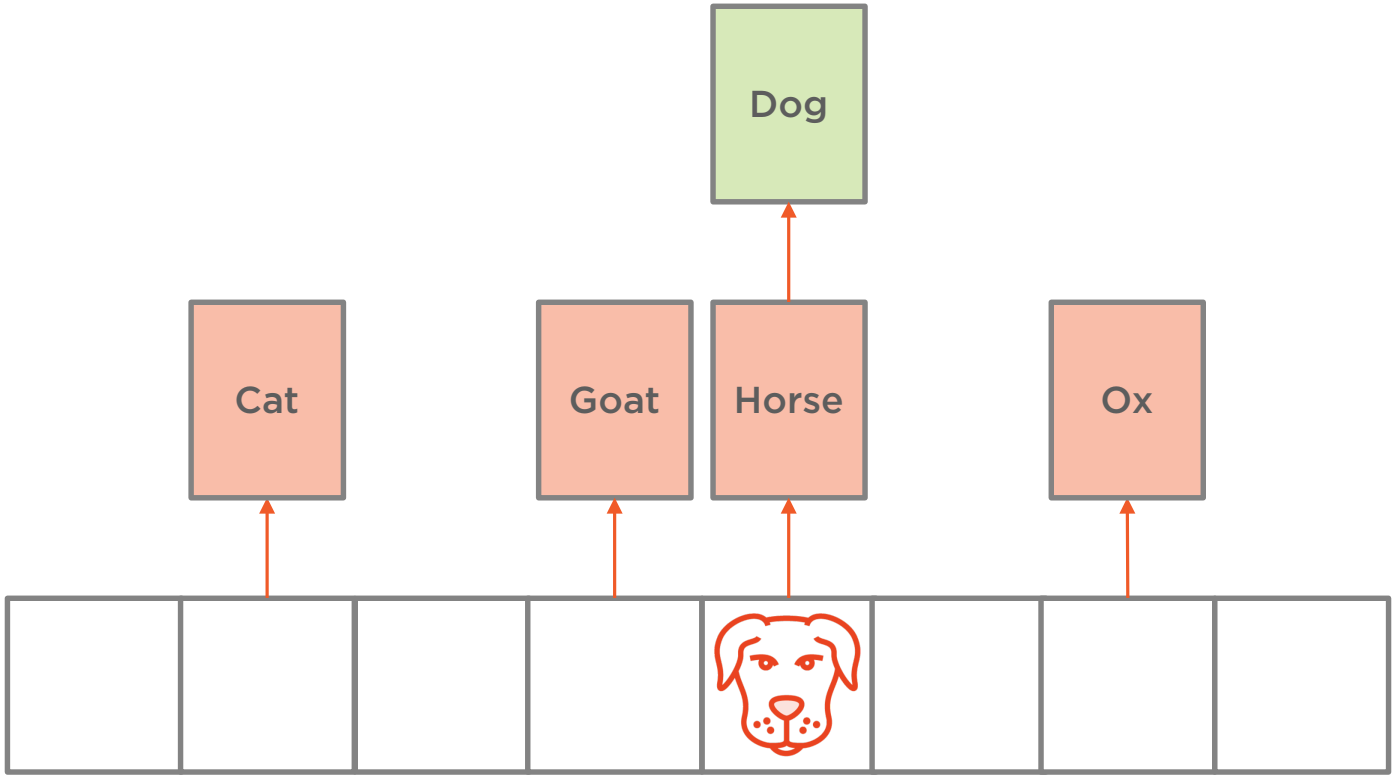


Find the index

Use the hash function

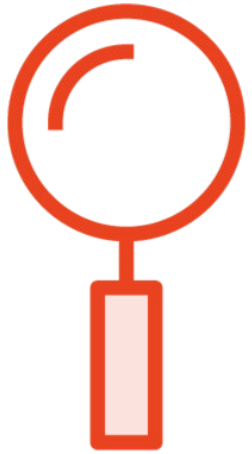
Modulo the hash

Search the entry list



Removing Items

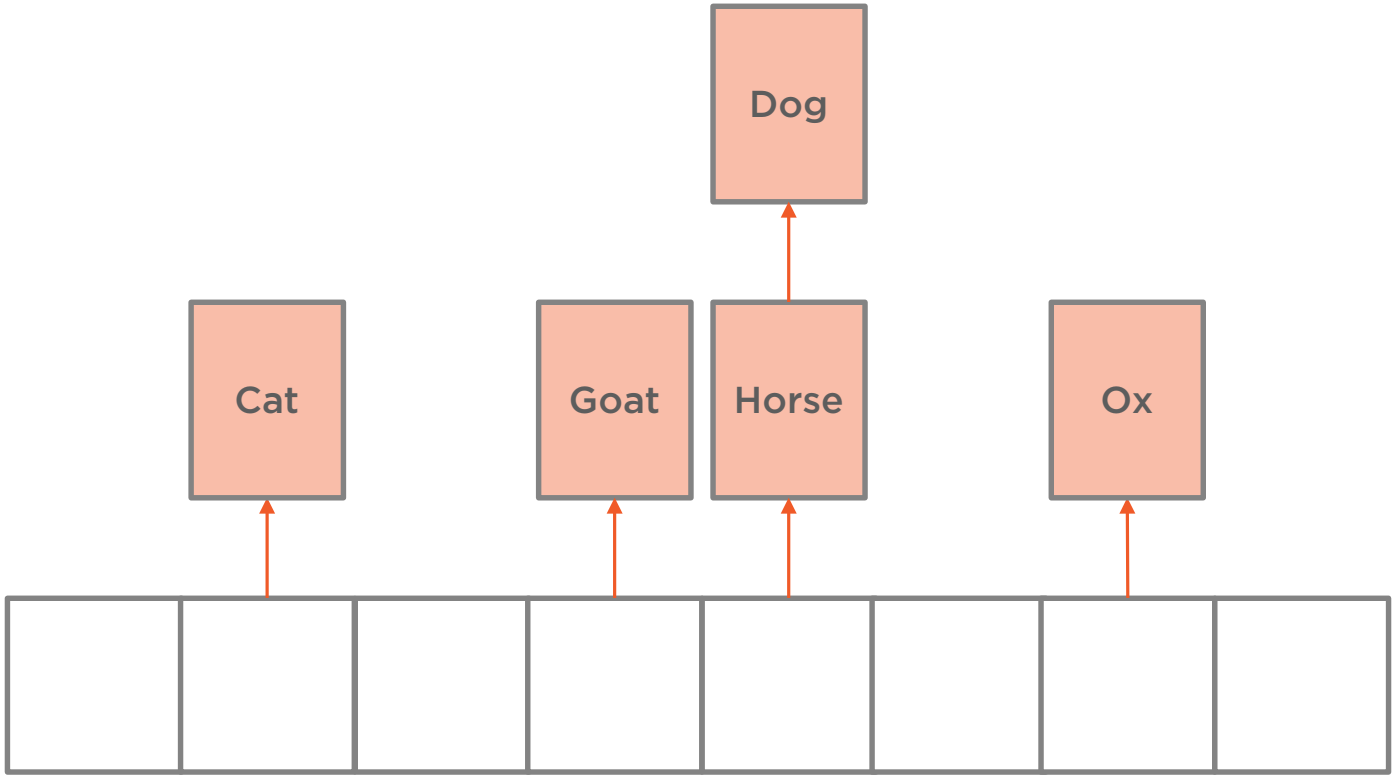
Removing Items



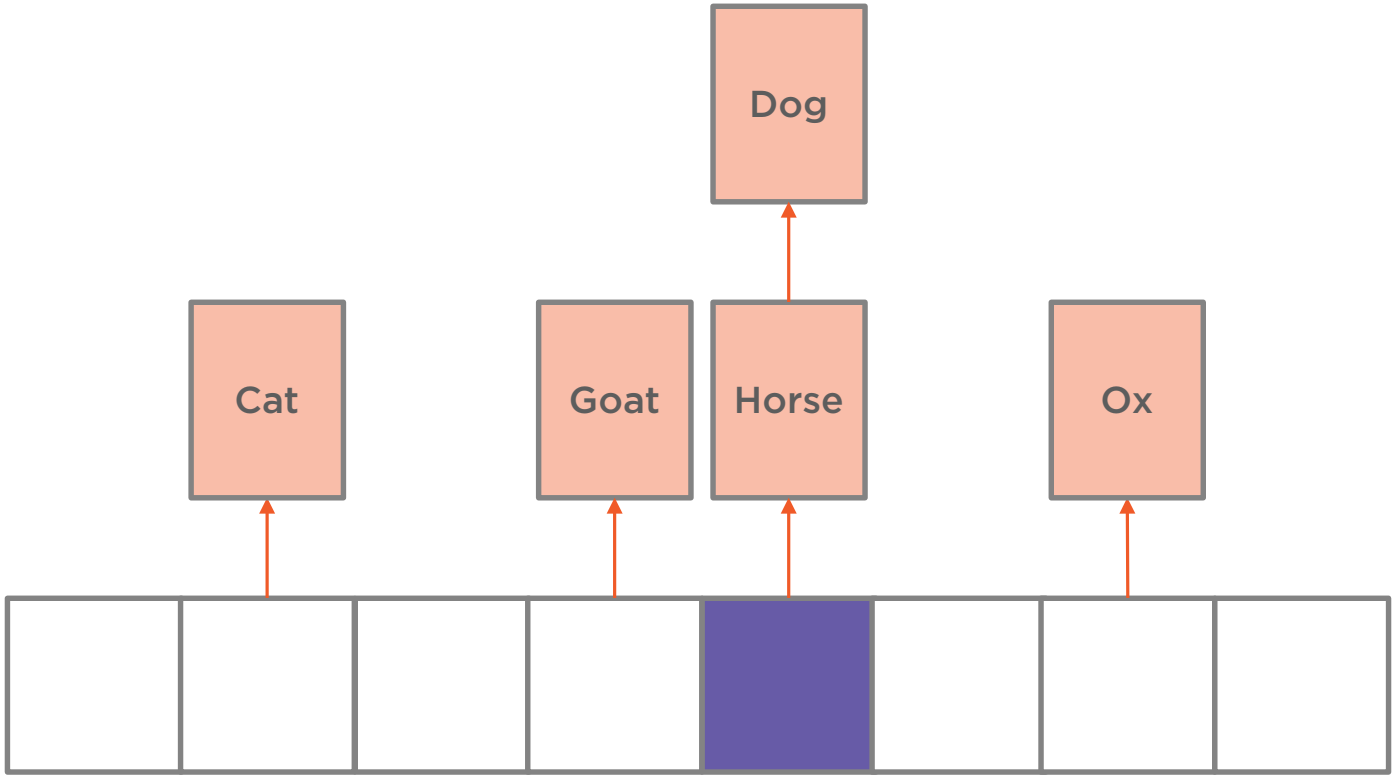
Determine the index

Search the entry list

Remove the entry

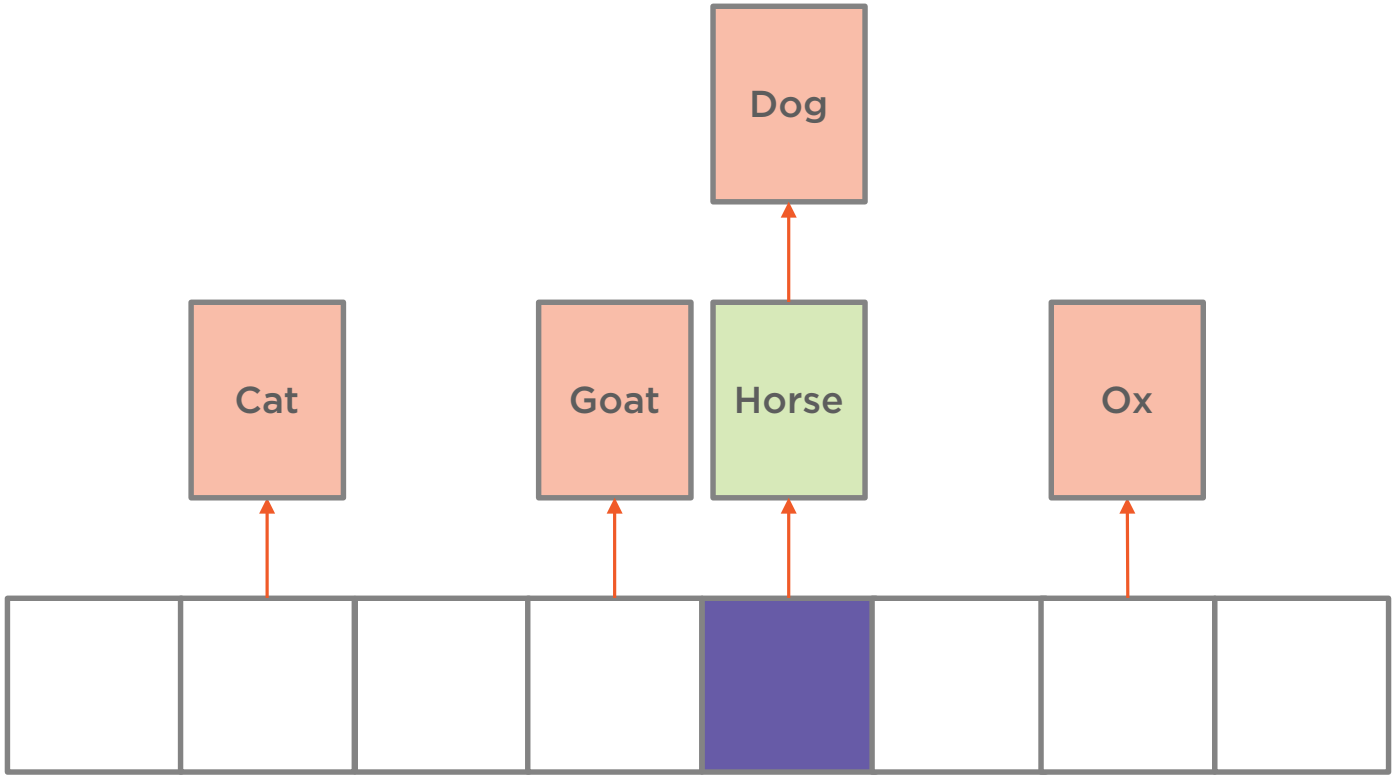

Dog
Cat
Goat
Horse
Ox

Find the index



Find the index

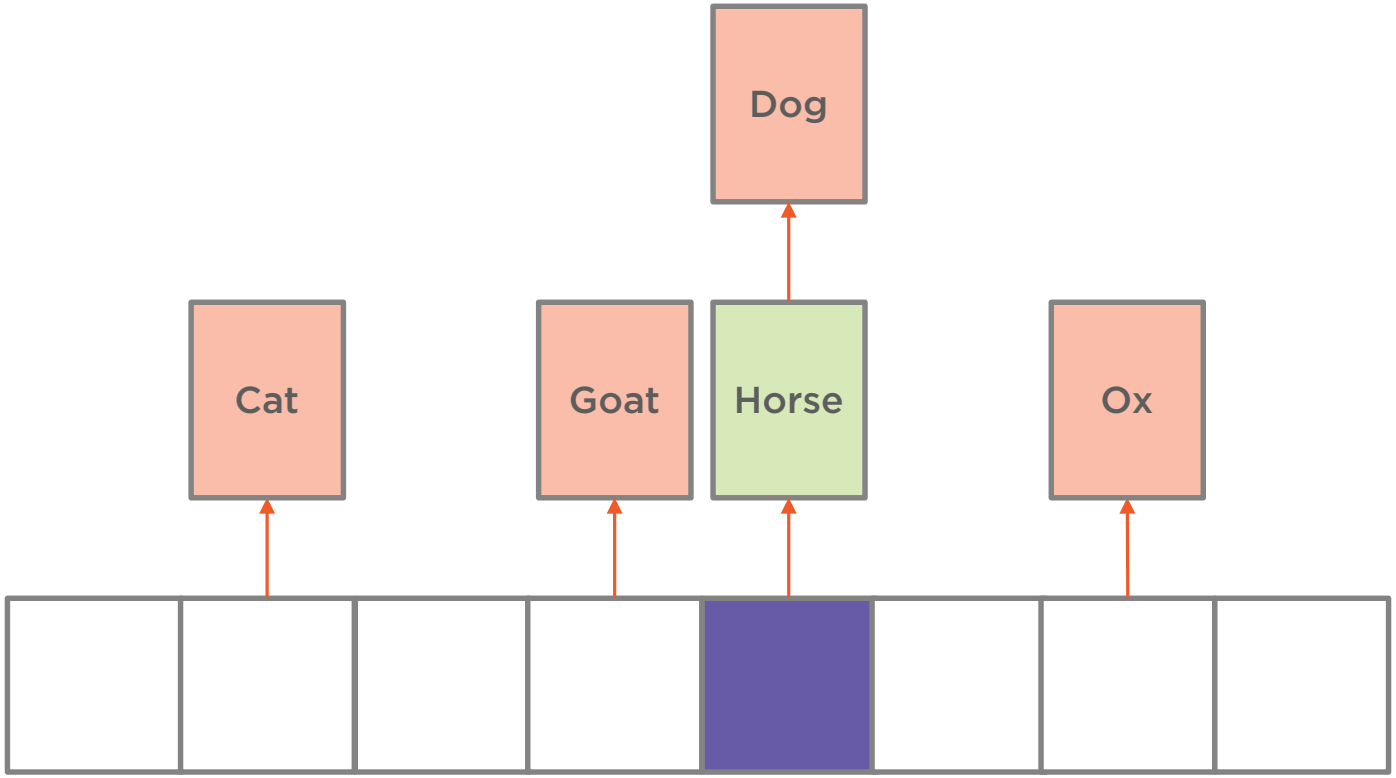
Search the entry list



Find the index

Search the entry list

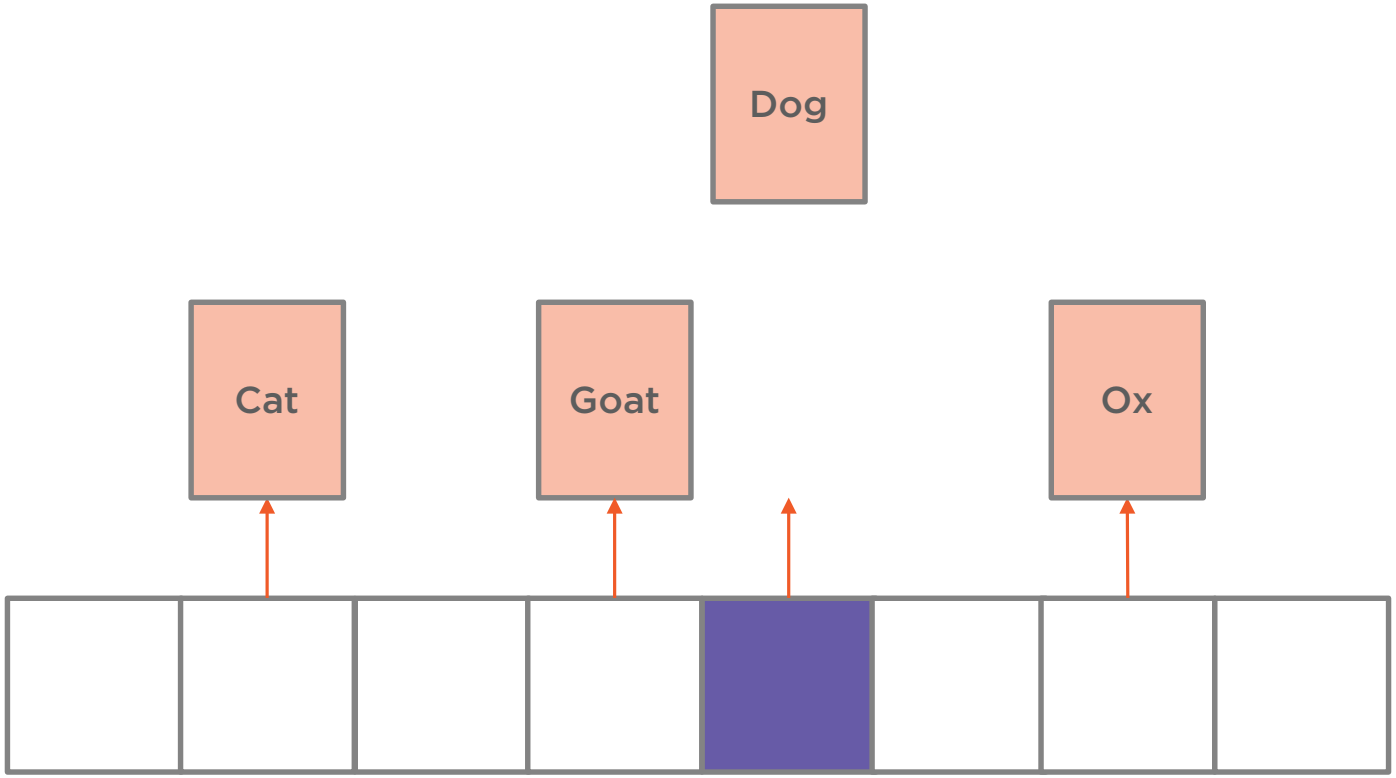
Remove the entry



Find the index

Search the entry list

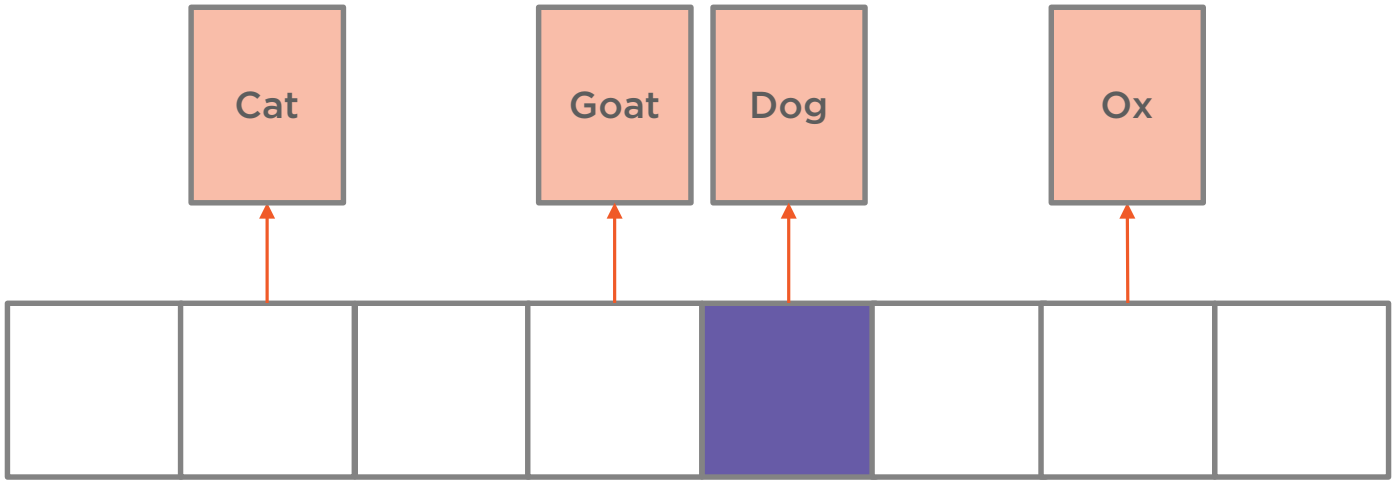
Remove the entry



Find the index

Search the entry list

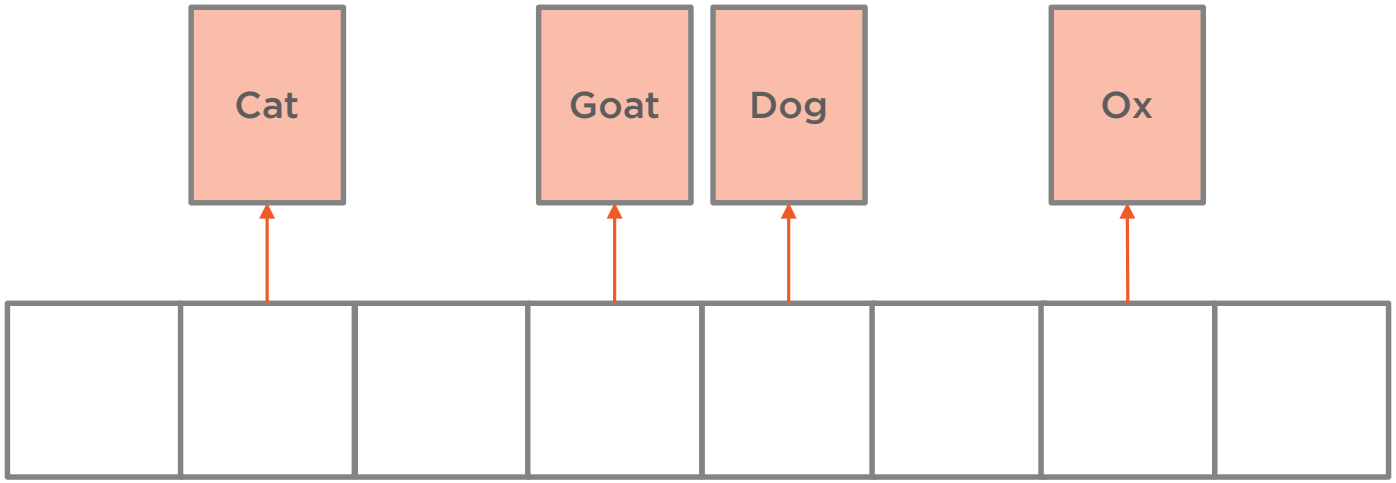
Remove the entry



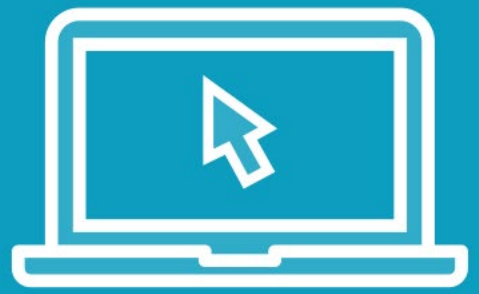
Find the index

Search the entry list

Remove the entry



Demo



Add state-level caching