Introduction to Hash Table Data Structure

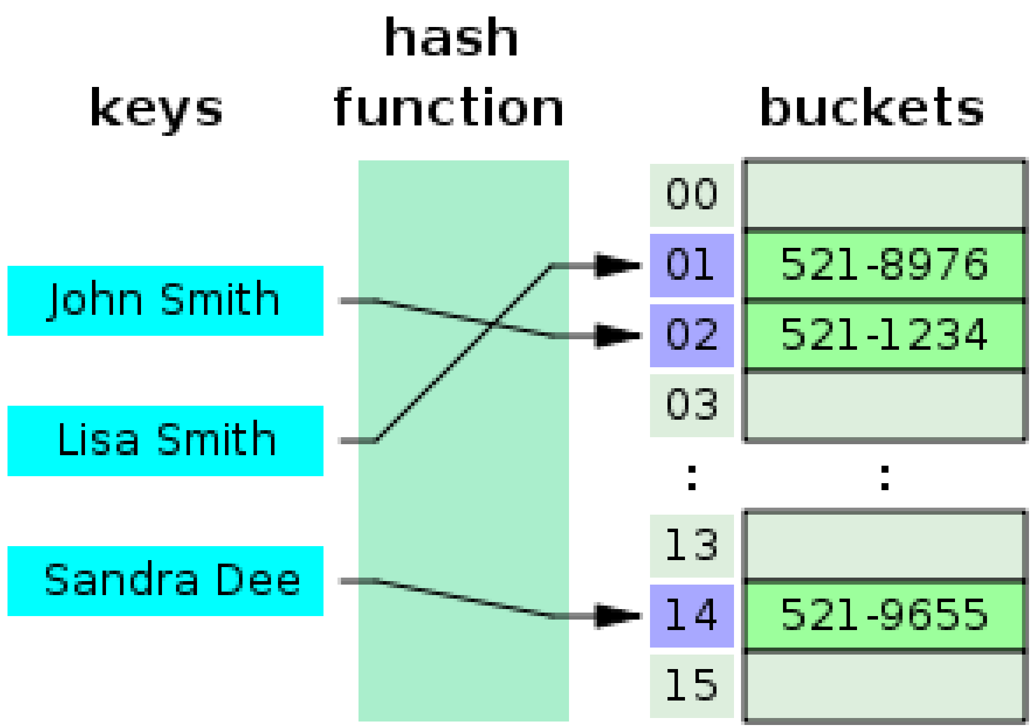
261217 Data Structures for Computer Engineers

Patiwet Wuttisarnwattana, Ph.D.

patiwet@eng.cmu.ac.th

Computer Engineering, Chiang Mai University

Hash Table Concept



Key Concept is the Hash Function
Hash function can map anything to an integer
The integer is an index of an array (table)

Hash function Application: Error checking

Bonus Time

$$x = \sum_{i=1}^{12} (14 - i) N_i \pmod{11}$$

$$x = (13N_1 + 12N_2 + 11N_3 + 10N_4 + 9N_5 + 8N_6 + 7N_7 + 6N_8 + 5N_9 + 4N_{10} + 3N_{11} + 2N_{12}) \pmod{11}$$

$$N_{13} = \begin{cases} 1 - x, & \text{if } x \leq 1 \\ 11 - x, & \text{if } x > 1 \end{cases}$$

Bonus Time

$$x = \sum_{i=1}^{12} (14 - i) N_i \pmod{11}$$

$$x = (13N_1 + 12N_2 + 11N_3 + 10N_4 + 9N_5 + 8N_6 + 7N_7 + 6N_8 + 5N_9 + 4N_{10} + 3N_{11} + 2N_{12}) \pmod{11}$$

$$N_{13} = \begin{cases} 1 - x, & \text{if } x \leq 1 \\ 11 - x, & \text{if } x > 1 \end{cases}$$

1 2345 67890 12 <u>3</u>

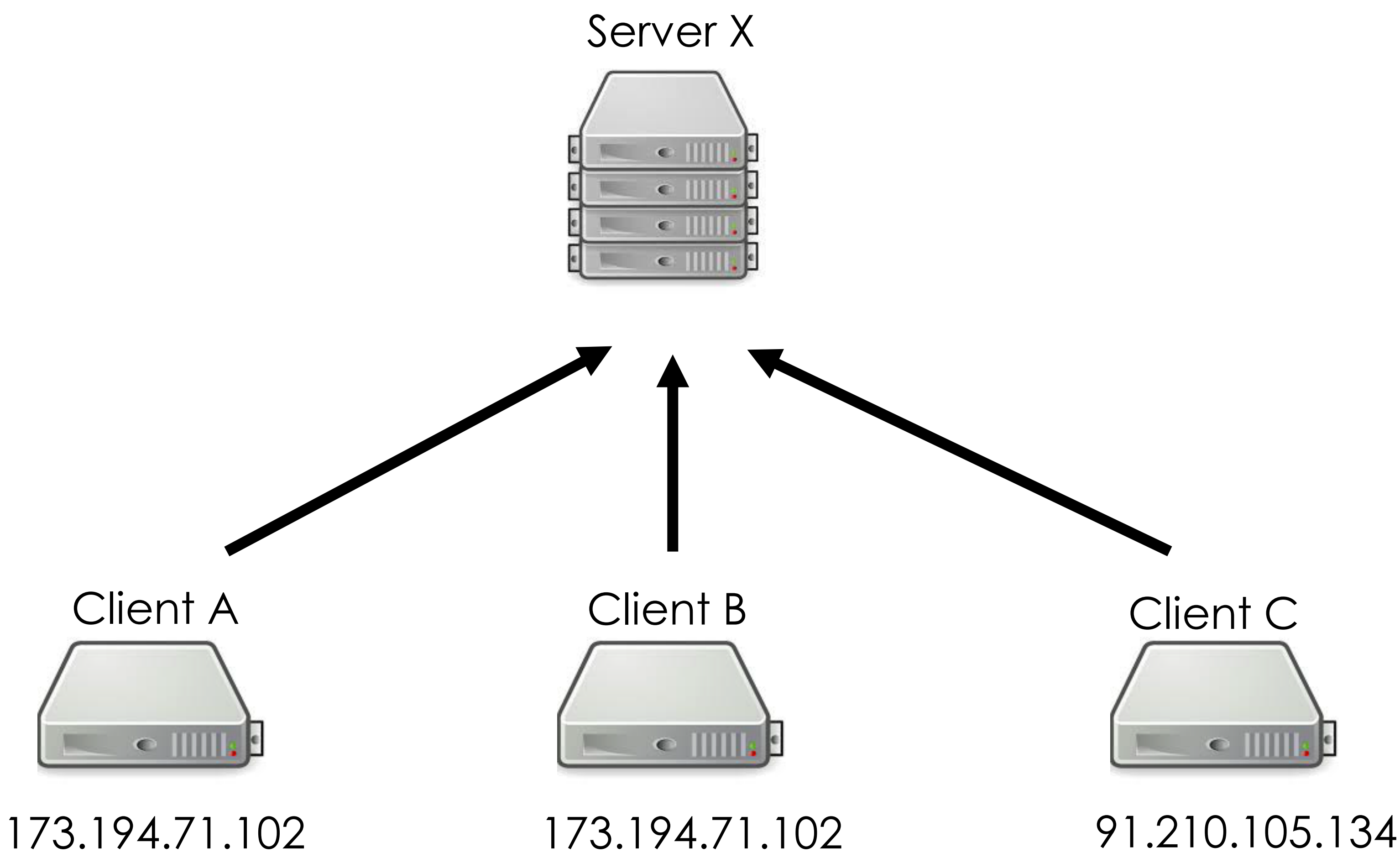
แสดงตัวเลขหลักที่ 13

Hash function Application: Password Storage

Hash function Application: Storage Optimization

Upload the file if the file is not uploaded yet
Guarantee No Duplication

Web Service


Server X
Client A
Client B
Client C
173.194.71.102
173.194.71.102
91.210.105.134

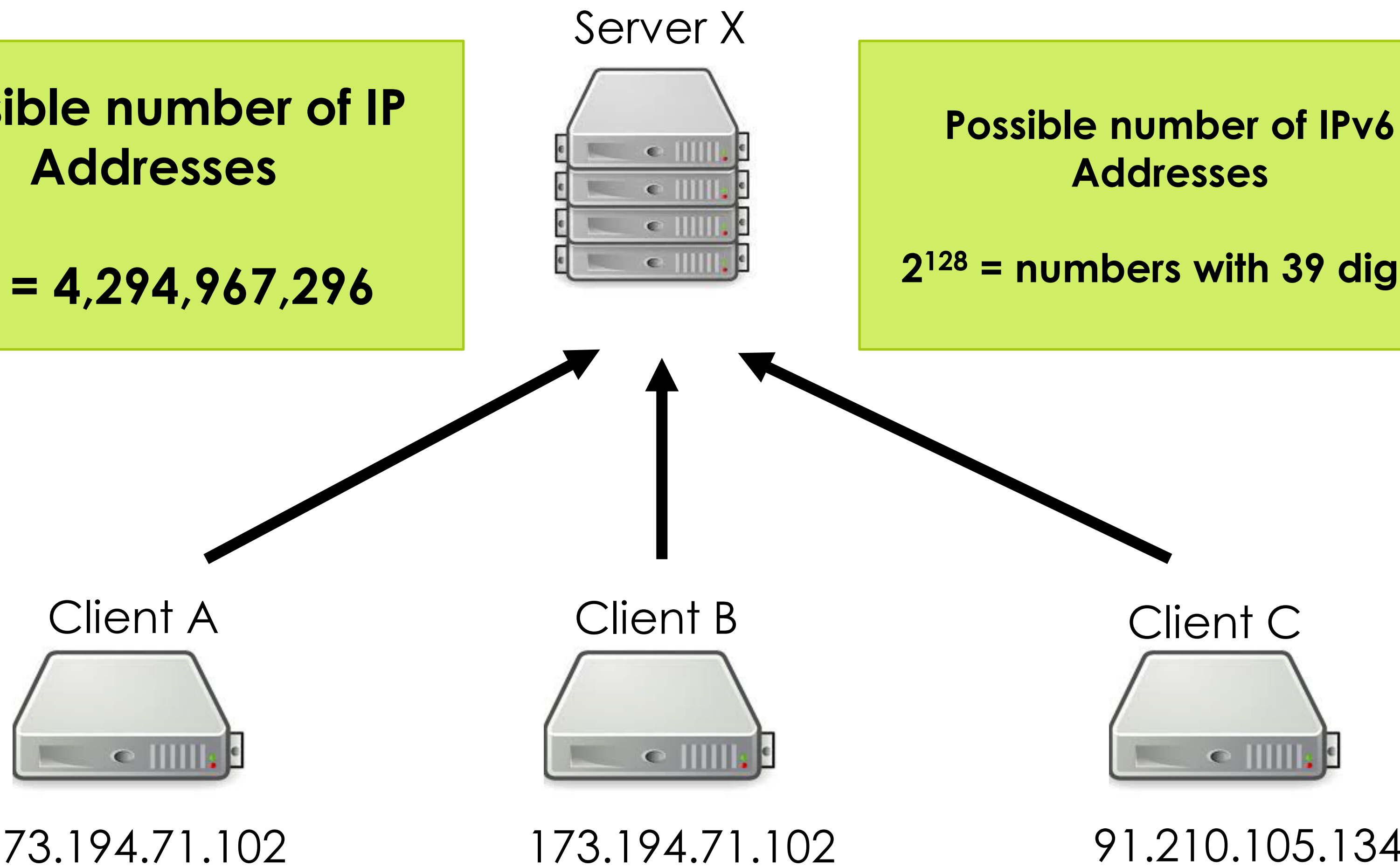
Web Service

Possible number of IP Addresses

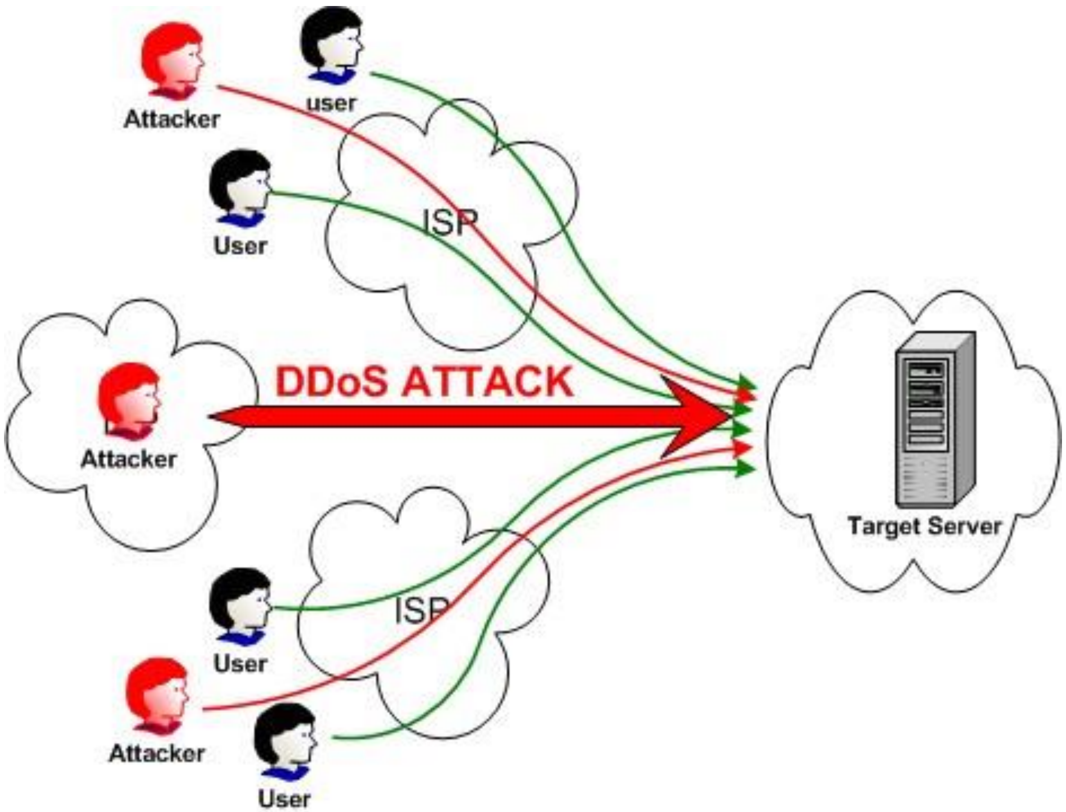
2^{32} = 4,294,967,296

Possible number of IPv6 Addresses

2^{128} = numbers with 39 digits



Denial of Service Attack



Access Log

Date	Time	IP address
09 Dec 2015	00:45:13	173.194.71.102
09 Dec 2015	00:45:15	69.171.230.68
...	...	...
...	...	...
09 Dec 2015	01:45:13	91.210.105.134

IP Access List

IP Access List

Analyse the access log and quickly answer queries: did anybody access the service from this *IP* during the last hour? How many times? How many *IP*s were used to access the service during the last hour?

Log Processing

- 1h of logs can contain millions of lines
- Too slow to process that for each query
- Keep count: how many times each IP appears in the last 1h of the access log
- **C** is some data structure to store the mapping from IP to counters
- We will learn later how to implement **C**

Log Processing

Decrease the counter

1 hour ago

Time	IP address
00:45:13	173.194.71.102
00:45:13	69.171.230.68
...	...
01:45:13	173.194.71.102
01:45:13	91.210.105.134

Now
(Need to update)

Increase the counter

Main Loop

log - array of log lines $(time, IP)$

C - mapping from IPs to counters

i - first unprocessed log line

j - first line in current 1h window

$i \leftarrow 0$

$j \leftarrow 0$

$C \leftarrow \emptyset$

Each second

  UpdateAccessList(log, i, j, C)

UpdateAccessList(log, i, j, C)

while $log[i].time \le Now()$:
  $C[log[i].IP] \leftarrow C[log[i].IP] + 1$
  $i \leftarrow i + 1$
while $log[j].time \le Now() - 3600$:
  $C[log[j].IP] \leftarrow C[log[j].IP] - 1$
  $j \leftarrow j + 1$

AccessedLastHour(IP, C)

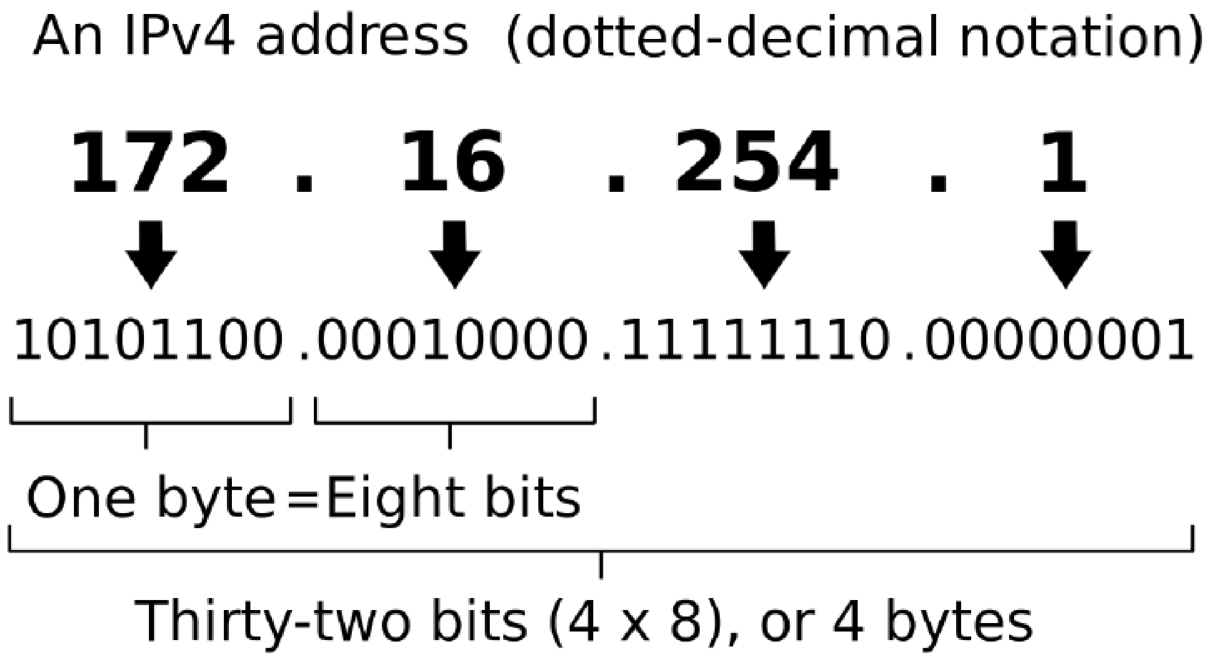
return $C[IP] > 0$

The Question is How to implement the mapping C?

Direct Addressing

- Need a data structure for **C**
- There are 2^{32} different IP(v4) addresses
  - $2^{32} = 4{,}294{,}967{,}296$
- Convert IP to 32-bit integer
- Create an integer array `A` of size 2^{32}
- Use `A[int(IP)]` as `C[IP]`

int(IP)



- int(0.0.0.1) = 1
- int(172.16.254.1) = 2,886,794,753
- int(69.171.230.68) = 1,168,893,508

$\mathtt{int}(IP)$

$\mathtt{return}\ \ IP[1]\cdot 2^{24} + IP[2]\cdot 2^{16} + IP[3]\cdot 2^{8} + IP[4]$

$\mathtt{UpdateAccessList}(log, i, j, A)$

$\mathtt{while}\ \ log[i].time \leq Now():$

$\quad A[\mathtt{int}(log[i].IP)] \leftarrow A[\mathtt{int}(log[i].IP)] + 1$

$\quad i \leftarrow i + 1$

$\mathtt{while}\ \ log[j].time \leq Now() - 3600:$

$\quad A[\mathtt{int}(log[j].IP)] \leftarrow A[\mathtt{int}(log[j].IP)] - 1$

$\quad j \leftarrow j + 1$

Is the server accessed by the IP in the last hour?

$$\texttt{AccessedLastHour}(IP)$$

$$\texttt{return}\ \ A[\texttt{int}(IP)] > 0$$

Big O Analysis of Direct Addressing Implementation

- UpdateAccessList is O(1) per log line
- AccessedLastHour is O(1)
- But need 2^{32} memory (4GB) even for few IPs
- IPv6: 2^{128} will not fit in memory
- In general: O(***U***) memory, ***U*** = number of possible IPs

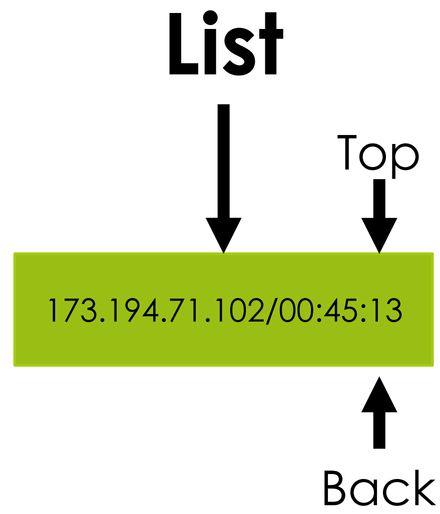
List-based Mapping

- Direct addressing requires too much memory
- Let’s store only active IPs
- Store them in a list
- Store only last occurrence of each IP
- Keep the order of occurrence

Access Log

Current Time: 00:45:15

Time	IP address
00:45:13	173.194.71.102
00:45:13	69.171.230.68
01:00:00	69.171.230.68
01:45:13	173.194.71.102
01:45:13	91.210.105.134



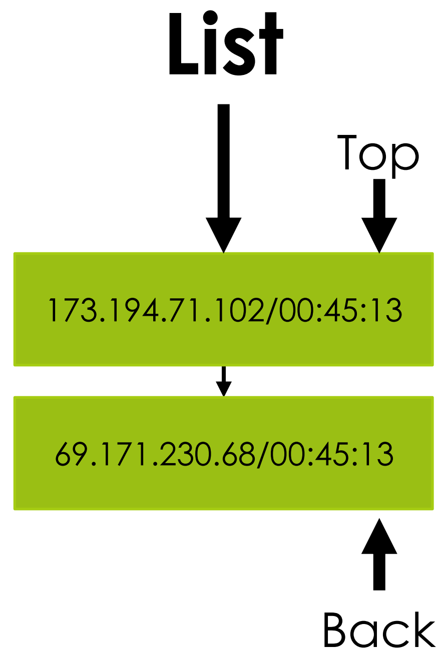
AccessedLastHour(69.171.230.68)?

AccessCountLastHour(69.171.230.68)?

Access Log

Current Time: 00:45:16

Time	IP address
00:45:13	173.194.71.102
00:45:13	69.171.230.68
01:00:00	69.171.230.68
01:45:13	173.194.71.102
01:45:13	91.210.105.134



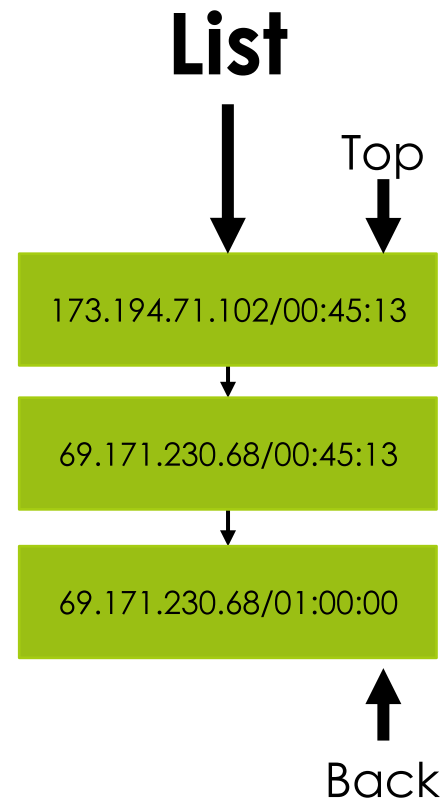
AccessedLastHour(69.171.230.68)?

AccessCountLastHour(69.171.230.68)?

Access Log

Current Time: 01:00:05

Time	IP address
00:45:13	173.194.71.102
00:45:13	69.171.230.68
01:00:00	69.171.230.68
01:45:13	173.194.71.102
01:45:13	91.210.105.134



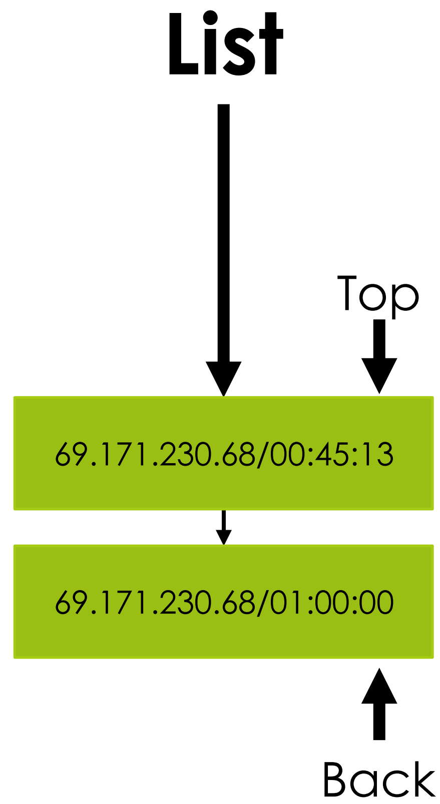
AccessedLastHour(69.171.230.68)?

AccessCountLastHour(69.171.230.68)?

Access Log

Current Time: 01:45:15

Time	IP address
00:45:13	173.194.71.102
00:45:13	69.171.230.68
01:00:00	69.171.230.68
01:45:13	173.194.71.102
01:45:13	91.210.105.134



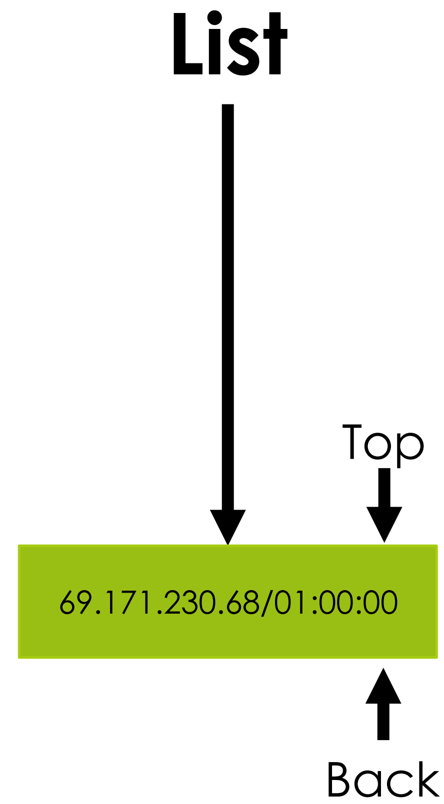
AccessedLastHour(69.171.230.68)?

AccessCountLastHour(69.171.230.68)?

Access Log

Current Time: 01:45:16

Time	IP address
00:45:13	173.194.71.102
00:45:13	69.171.230.68
01:00:00	69.171.230.68
01:45:13	173.194.71.102
01:45:13	91.210.105.134



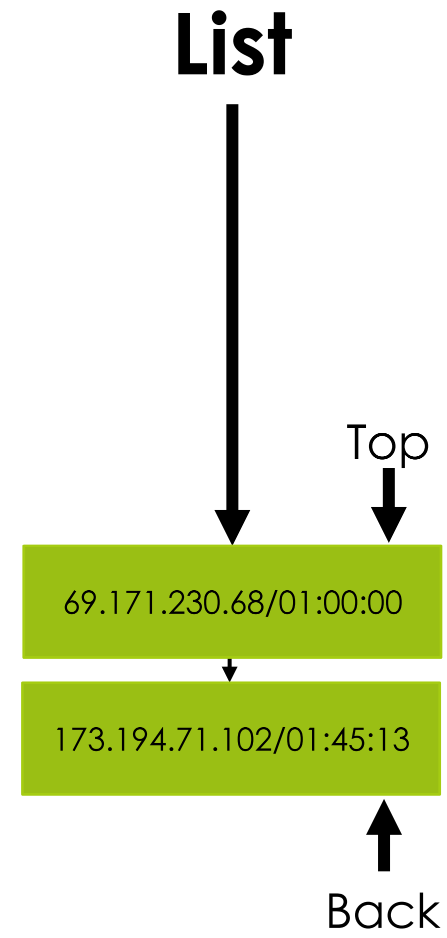
AccessedLastHour(69.171.230.68)?

AccessCountLastHour(69.171.230.68)?

Access Log

Current Time: 01:45:17

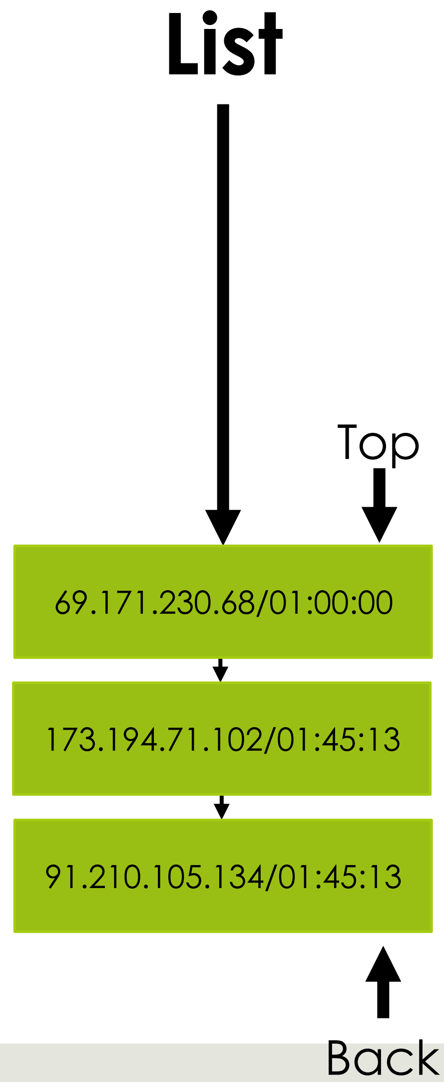
Time	IP address
00:45:13	173.194.71.102
00:45:13	69.171.230.68
01:00:00	69.171.230.68
01:45:13	173.194.71.102
01:45:13	91.210.105.134



Access Log

Current Time: 01:45:18

Time	IP address
00:45:13	173.194.71.102
00:45:13	69.171.230.68
01:00:00	69.171.230.68
01:45:13	173.194.71.102
01:45:13	91.210.105.134



UpdateAccessList(*log*, *i*, *L*)

UpdateAccessList(*log*, *i*, *L*)

```
while log[i].time ≤ Now()
     L.Append(log[i])
     i ← i + 1
while L.Top().time ≤ Now() - 3600
     L.Pop()
```

AccessedLastHour and AccessCountLastHour

AccessedLastHour(*IP*, *L*)

return *L*.FindByIP(*IP*) ≠ NULL

AccessCountLastHour(*IP*, *L*)

return *L*.CountIP(*IP*)

Big O Analysis of List-based Implementation

- n is number of active IPs
- Memory usage is O(n)
- `L.Append, K.Top, L.pop` are O(1)
- `UpdateAccessList` is O(1) per log line
- `L.FindByIP` and `L.CountIP` are O(n)
- `AccessedLastHour` and `AccessCountLastHour` are O(n)

Encoding (Hashing) IPs

- Encode/Hash IPs with small numbers
- For example, numbers from 0 to 999
- Different codes for currently active IPs

Hash Function

Definition

For any set of objects S and any integer $m > 0$, a function $h : S \to \{0, 1, \ldots, m-1\}$ is called a hash function.

Definition

m is called the cardinality of hash function h.

Bonus Time!!!

Desirable Properties

- **h** should be fast to compute
- Different values for different objects
- Direct addressing with $O(\mathbf{m})$ memory
- Want small cardinality **m**
- Impossible to have all different values if number of objects in *the universe* is more than **m**

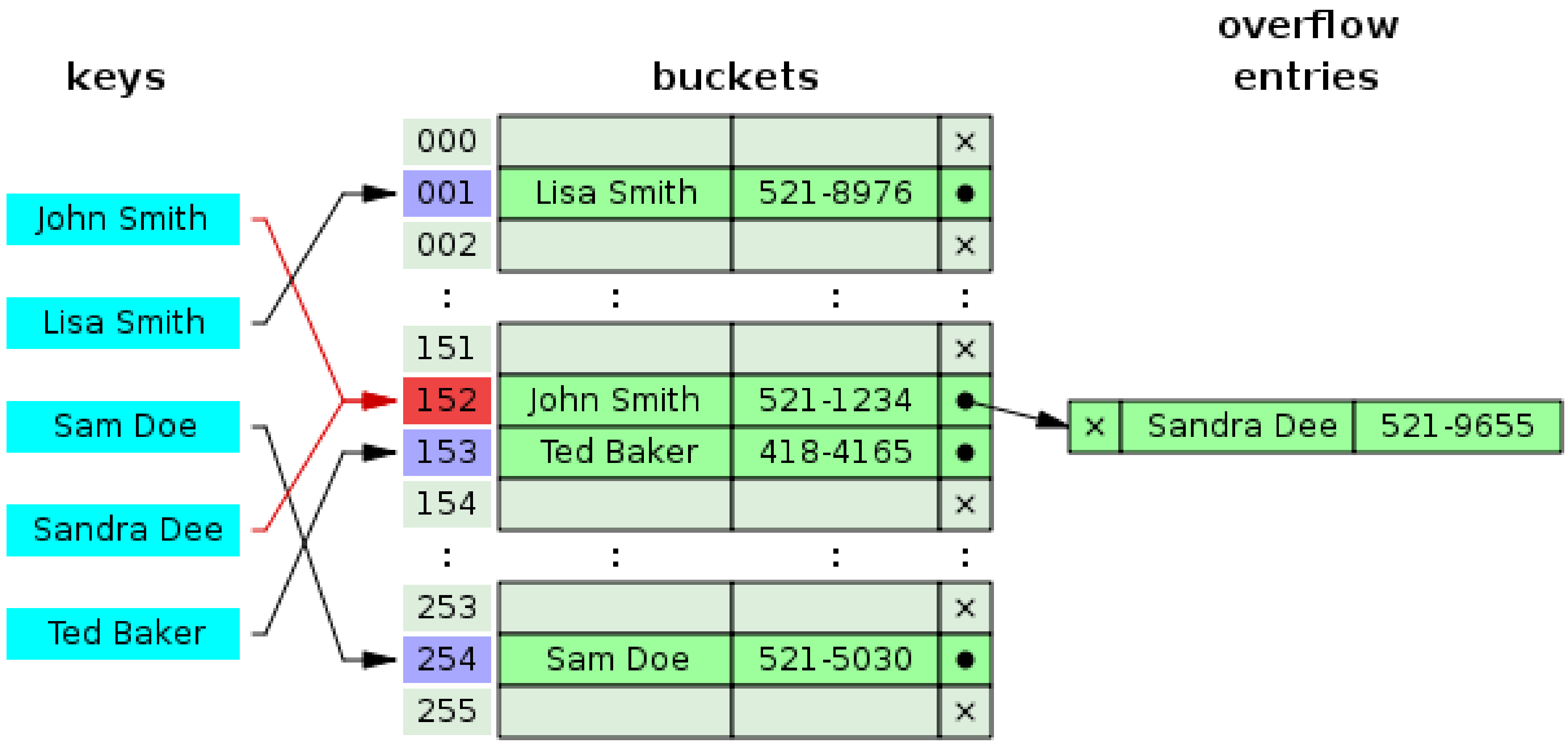
Collisions

Definition

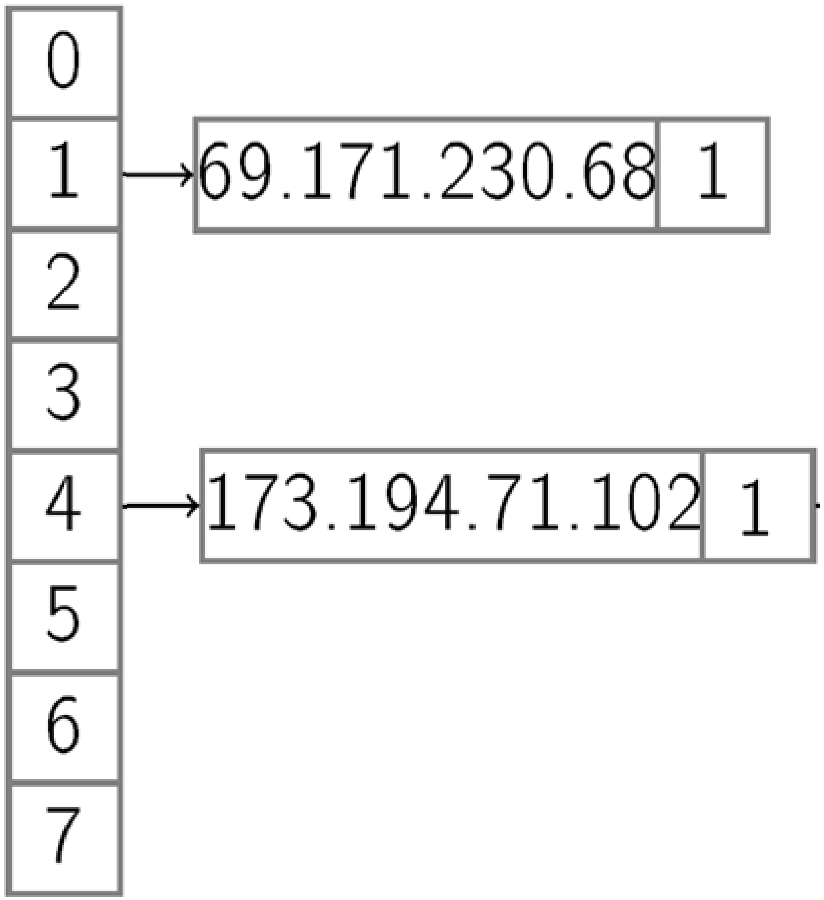
When $h(o_1) = h(o_2)$ and $o_1 \neq o_2$, this is a collision.

Collision Resolution of a Hash Table

Separate Chaining



Back to our example



$$h(173.194.71.102) = 4$$

$$h(69.171.230.68) = 1$$

$$h(173.194.71.102) = 4$$

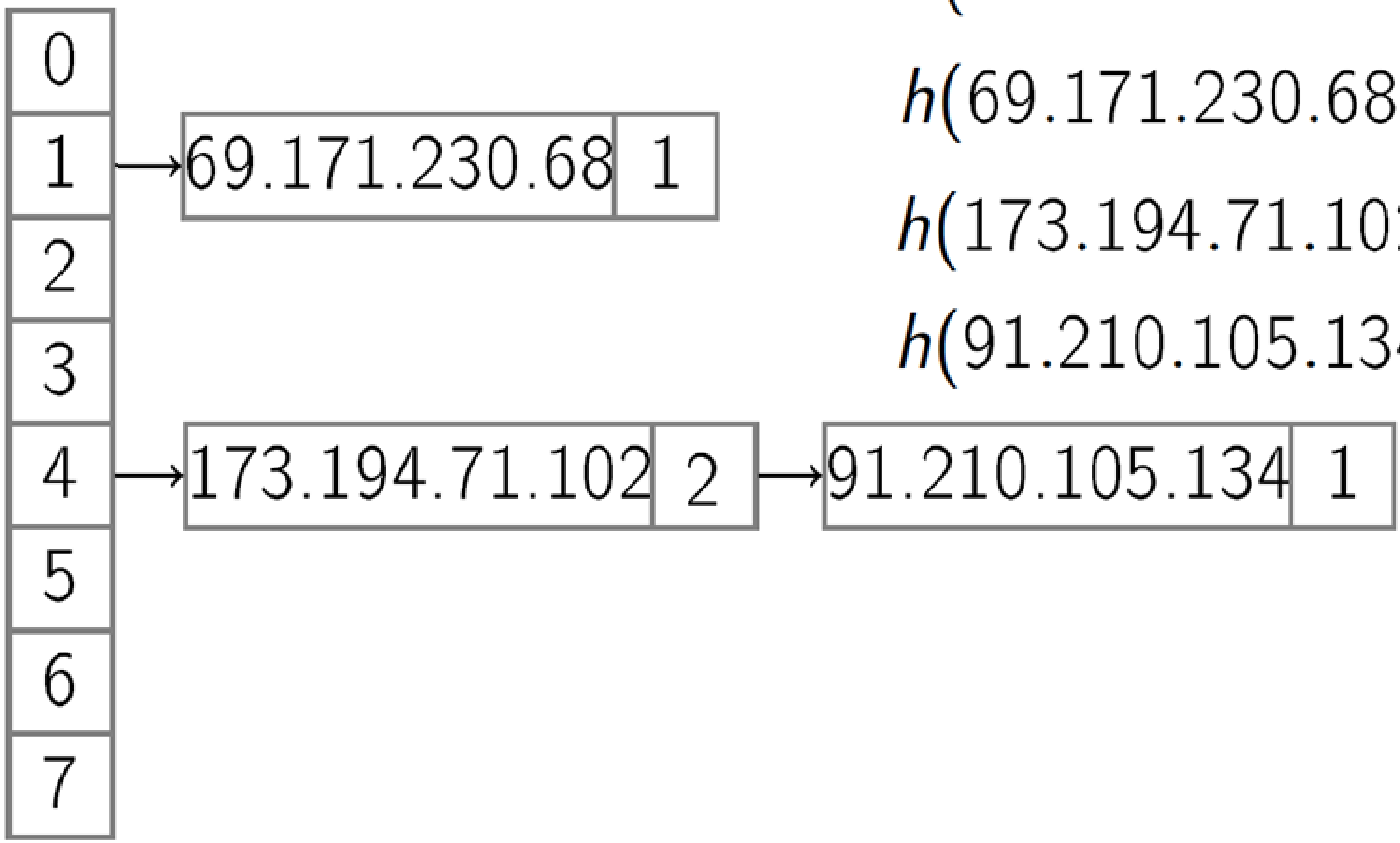
There is no collision

$$h(173.194.71.102) = 4$$

$$h(69.171.230.68) = 1$$

$$h(173.194.71.102) = 4$$

$$h(91.210.105.134) = 4$$



Map Data Structure

Dictionary Data Structure

Associative Array Data Structure

are Hash Table

Map (Dictionary, Associative Array)

- Arrays store items as an ordered collection and you can access thems with an index number.
- Maps store items in (Key, Value) pairs that you can access values by the corresponding keys (usually Strings)
  - Filename → location of the file on disk
  - Student ID → student name
  - Contact name → contact phone number
  - The first object is called "Key" or O
  - The second object is called "Value" or v
  - Map can have the following operations `HasKey(O)`, `Get(O)`, `Set(O,v)`

Map Data Structure

Map217

```
{
    {"580610615", "Kanokwan Pinthong"},
    {"580610616", "Kawewut Chujit"},
    {"580610618", "Kittitorn Rakpanyakeaw"}
}
```

- HasKey("580610615")
- HasKey("580610617")
- Get("580610618")
- Set("580610618", "Kittitat Boonkarn")
- Get("580610618")

Demo: Java HashMap

https://www.w3schools.com/java/java_hashmap.asp

Map Data Structure

Dictionary Data Structure

Associative Array Data Structure

can be implemented using Hash Table

$h : S \to \{0, 1, \ldots, m-1\}$

$O, O' \in S$

$v, v' \in V$

$A \leftarrow$ array of m lists (chains) of pairs (O, v)

`HasKey`(O)

$L \leftarrow A[h(O)]$

```
for (O′, v′) in L:
  if O′ == O:
    return true
return false
```

Get(O)

```
L ← A[h(O)]
for (O', v') in L:
    if O' == O:
        return v'
return n/a
```

$\texttt{Set}(O, v)$

$L \leftarrow A[h(O)]$

$\texttt{for } p \texttt{ in } L\texttt{:}$

$\quad \texttt{if } p.O == O\texttt{:}$

$\quad\quad p.v \leftarrow v$

$\quad\quad \texttt{return}$

$L.\texttt{Append}(O, v)$

Runtime Analysis

Lemma

Let c be the length of the longest chain in A. Then the running time of `HasKey`, `Get`, `Set` is $O(c+1)$.

Runtime Analysis

Lemma

Let n be the number of different keys O currently in the map and m be the cardinality of the hash function. Then the memory consumption for chaining is $\Theta(n + m)$.

Set

Definition

Set is a data structure with methods `Add`(O), `Remove`(O), `Find`(O).

Examples

- IPs accessed during last hour
- Students on campus
- Keywords in a programming language

Implementing Set

$h : S \to \{0, 1, \ldots, m-1\}$

$O, O' \in S$

$A \leftarrow$ array of m lists (chains) of objects O

Find(O)

$L \leftarrow A[h(O)]$

for O' in L:

  if $O' == O$:

    return true

return false

Implementing Set

$\mathtt{Add}(O)$

$L \leftarrow A[h(O)]$

$\mathtt{for}\ O'\ \mathtt{in}\ L:$

$\quad \mathtt{if}\ O' == O:$

$\qquad \mathtt{return}$

$L.\mathtt{Append}(O)$

Implementing Set

$\texttt{Remove}(O)$

$\texttt{if not Find}(O)\texttt{:}$

  $\texttt{return}$

$L \leftarrow A[h(O)]$

$L.\texttt{Erase}(O)$

Hash Table

Definition

An implementation of a set or a map using hashing is called a hash table.

Programming Languages

Set:

- `unordered_set` in C++
- `HashSet` in Java
- `set` in Python

Map:

- `unordered_map` in C++
- `HashMap` in Java
- `dict` in Python

Conclusion

- Chaining is a technique to implement a hash table
- Memory consumption is $O(n + m)$
- Operations works in time $O(c + 1)$
- How to make both m and c small?