MONASH University

MONASH
INFORMATION
TECHNOLOGY

FIT2004
Algorithms and Data Structures

Ian Wern Han Lim
lim.wern.han@monash.edu

Referencing materials by
Nathan Companez, Aamir Cheema, Arun Konagurthu and Lloyd Allison

GROUP OF EIGHT AUSTRALIA

Faculty of Information Technology, Monash University

MONASH University

Ready?

Agenda

- Hash Tables

Agenda

- Hash Tables
  - Cuckoo hashing

MONASH University

Let us begin…

Data Lookup

Getting what we stored…

- Very often, we store data/ information

Data Lookup

Getting what we stored…

- Very often, we store data/ information
- How do we retrieve it?

Data Lookup

Getting what we stored…

MONASH University

- Very often, we store data/ information
- How do we retrieve it?
  - Look it up… AKA SEARCH

Data Lookup

Getting what we stored…

MONASH University

- Very often, we store data/ information
- How do we retrieve it?
  - Look it up… AKA SEARCH

- So how do we search?

Data Lookup

Getting what we stored…

MONASH University

- Very often, we store data/ information
- How do we retrieve it?
  - Look it up… AKA SEARCH
- So how do we search?
  - By the key

Data Lookup

Getting what we stored…

MONASH University

- Very often, we store data/ information
- How do we retrieve it?
  - Look it up… AKA SEARCH

- So how do we search?
  - By the key
  - Key is unique
  - Data is then accessed

Data Lookup

Getting what we stored…

MONASH University

- Very often, we store data/ information
- How do we retrieve it?
  - Look it up… AKA SEARCH

- So how do we search?
  - By the key
  - Key is unique
    - Student ID
  - Data is then accessed
    - Name
    - Age
    - Address
    - Contact

Data Lookup

An array/ list

- Let say we store data in an array/ list

Data Lookup

An array/ list

MONASH University

- Let say we store data in an array/ list
- How fast can we find what we want?
  - Time complexity O(N) via linear search

Data Lookup

An array/ list

MONASH University

- Let say we store data in an array/ list
- How fast can we find what we want?
  - Time complexity O(N) via linear search
  - So can we make it faster?

Data Lookup

An sorted array/ ~~list~~

MONASH University

- If the data is sorted… Then we can use binary search!
  - Best case O(1)
  - Worst case O(log N)

Data Lookup

An sorted array/ ~~list~~

MONASH University

- If the data is sorted… Then we can use binary search!
  - Best case O(1)
  - Worst case O(log N)
- But we need to insert in order…
  - Giving us O(N) complexity still…

Data Lookup

An sorted array/ ~~list~~

- If the data is sorted… Then we can use binary search!
  - Best case O(1)
  - Worst case O(log N)
- But we need to insert in order…
  - Giving us O(N) complexity still…
- Likewise for delete, we need to shift
  - Giving us O(N) complexity still…

MONASH University

Questions?

Hash Table

Quick search!

MONASH University

- What if we have a way to know where we store the item?

Hash Table

Quick search!

MONASH University

- What if we have a way to know where we store the item?
- We need an array to access position in O(1) time

Hash Table

Quick search!

MONASH University

- What if we have a way to know where we store the item?
- We need an array to access position in O(1) time
  - Fixed size
  - O(1) random access

Hash Table

Direct-Addressing

MONASH University

- If we have N student
  - Student ID from 0 to N-1

Hash Table
Direct-Addressing

- If we have N student
  - Student ID from 0 to N-1
- We create an array N-size
  - StudentID = index

Hash Table
Direct-Addressing

MONASH University

- If we have N student
  - Student ID from 0 to N-1
- We create an array N-size
  - StudentID = index

- But what if our student ID is 8 digit?
  - Like Monash's?

Hash Table
Direct-Addressing

MONASH University

- If we have N student
  - Student ID from 0 to N-1
- We create an array N-size
  - StudentID = index

- But what if our student ID is 8 digit?
  - Like Monash's?
  - Make a very big array?

Hash Table
Direct-Addressing

MONASH University

- If we have N student
  - Student ID from 0 to N-1
- We create an array N-size
  - StudentID = index

- But what if our student ID is 8 digit?
  - Like Monash's?
  - Make a very big array?
- What if the student ID has alphabet?
  - Like Authcate?

Hash Table
Direct-Addressing

MONASH University

- If we have N student
  - Student ID from 0 to N-1
- We create an array N-size
  - StudentID = index

- But what if our student ID is 8 digit?
  - Like Monash's?
  - Make a very big array?
- What if the student ID has alphabet?
  - Like Authcate?

Hash functions

MONASH University

Questions?

Hash Table

Hash Functions

MONASH University

- A function to convert key into position in array
  - index = hash(key)

Hash Table

Hash Functions

MONASH University

- A function to convert key into position in array
  - index = hash(key)
  - Within the boundary of the array
    - % len(array)

Hash Table

Hash Functions

MONASH University

- A function to convert key into position in array
  - index = hash(key)
  - Within the boundary of the array
    - % len(array)
- Consider the following
  - StudentID = 322
  - len(array) = 10
  - Hash function = key % len(array)
  - What is the index?

Hash Table
Hash Functions

MONASH University

- A function to convert key into position in array
  - index = hash(key)
  - Within the boundary of the array
    - % len(array)

- Consider the following
  - StudentID = 322
  - len(array) = 10
  - Hash function = key % len(array)
  - Index = 2
  - Now we have another student, with the ID = 422
  - What is index?

Hash Table
Hash Functions

MONASH University

- A function to convert key into position in array
  - index = hash(key)
  - Within the boundary of the array
    - % len(array)

- Consider the following
  - StudentID = 322
  - len(array) = 10
  - Hash function = key % len(array)
  - Index = 2
  - Now we have another student, with the ID = 422
  - What is index? = 2 as well! COLLISION

MONASH University

Questions?

MONASH University

- So we see a collision…

Hash Table
Avoiding Collisions

MONASH University

- So we see a collision…
- Can we avoid it?

Hash Table
Avoiding Collisions

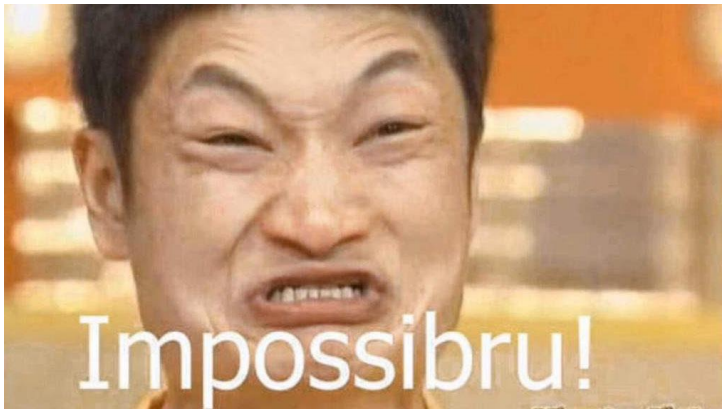
MONASH University

- So we see a collision…
- Can we avoid it?
  - Good hash function

Hash Table

Avoiding Collisions

- So we see a collision…
- Can we avoid it?
  - Good hash function



Understanding hash functions

- We want to use a hash table for a class of 50 students
- Assume that the hash function is based on birthdays (dd-mm), e.g., a student born on 01-Jan is hashed to index 1, 02-Jan on index 2, …, 31-Dec on 365.
- How likely is that two students will be hashed to the same index, e.g., how likely is the collision?

$$Prob(no\ collision) = \frac{365}{365} \times \frac{364}{365} \times \frac{363}{365} \cdots \times \frac{(366-N)}{365}$$

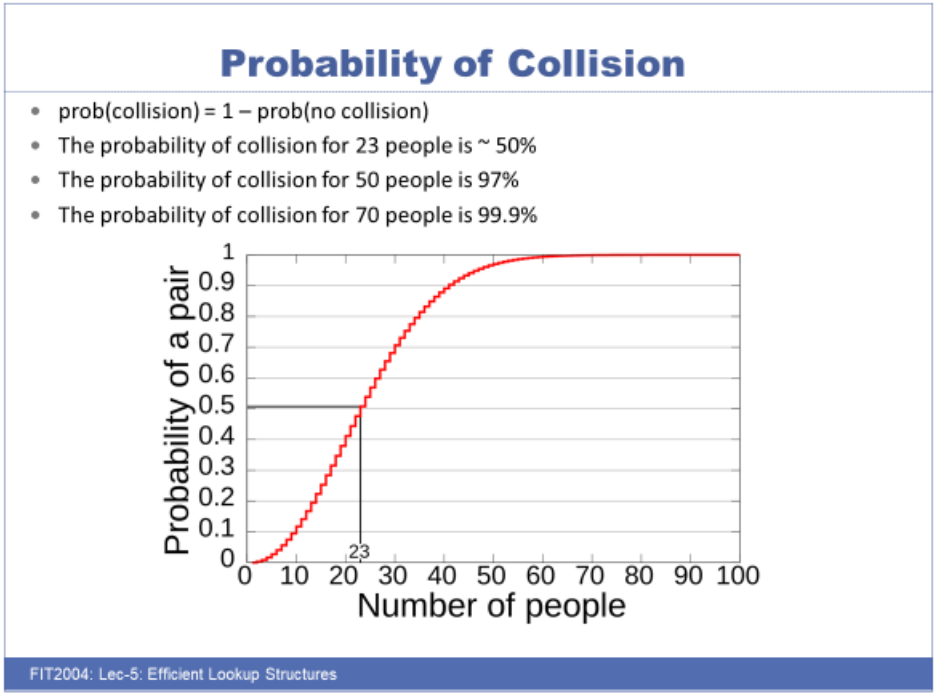
$$Prob(no\ collision) = \frac{365!}{365^N (365-N)!}$$

Visit https://pudding.cool/2018/04/birthday-paradox/ for an interactive explanation of the birthday paradox

FIT2004: Lec-5: Efficient Lookup Structures

Hash Table
Avoiding Collisions

- So we see a collision…
- Can we avoid it?
  - Good hash function
  - Kinda too hard…

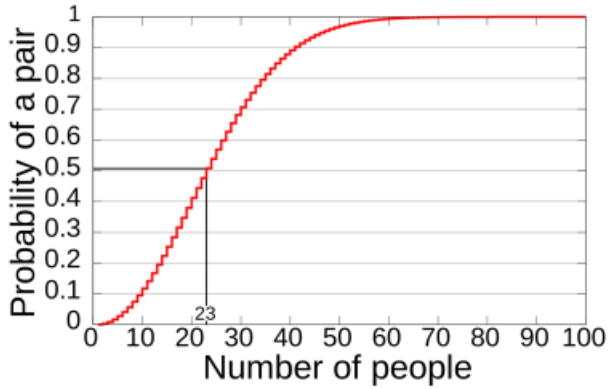


Hash Table
Avoiding Collisions

- So we see a collision…
- Can we avoid it?
  - Good hash function
  - Kinda too hard…

Probability of Collision

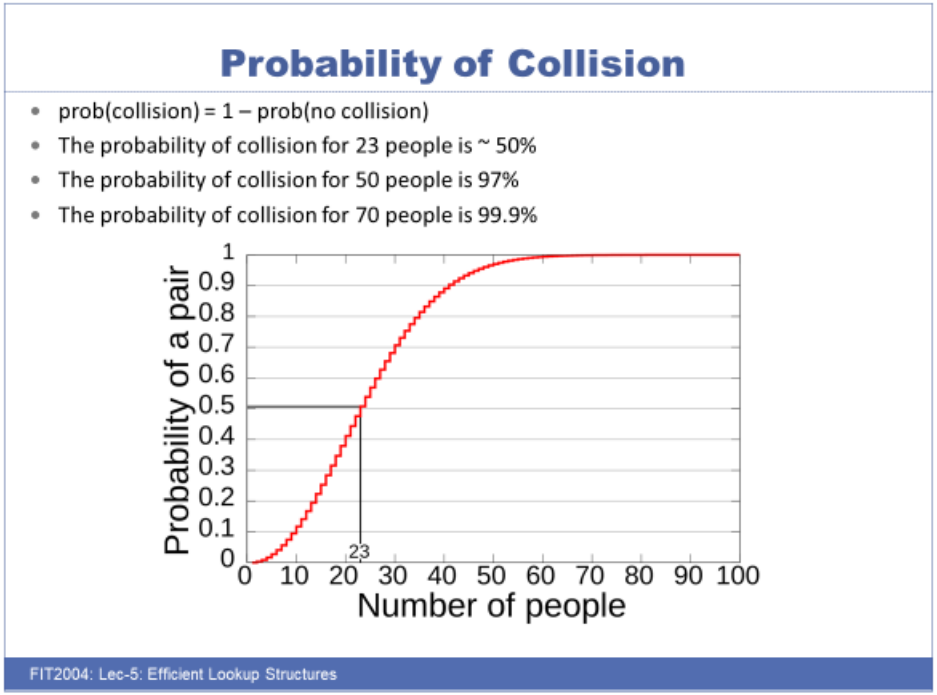
- prob(collision) = 1 – prob(no collision)
- The probability of collision for 23 people is ~ 50%
- The probability of collision for 50 people is 97%
- The probability of collision for 70 people is 99.9%



FIT2004: Lec-5: Efficient Lookup Structures

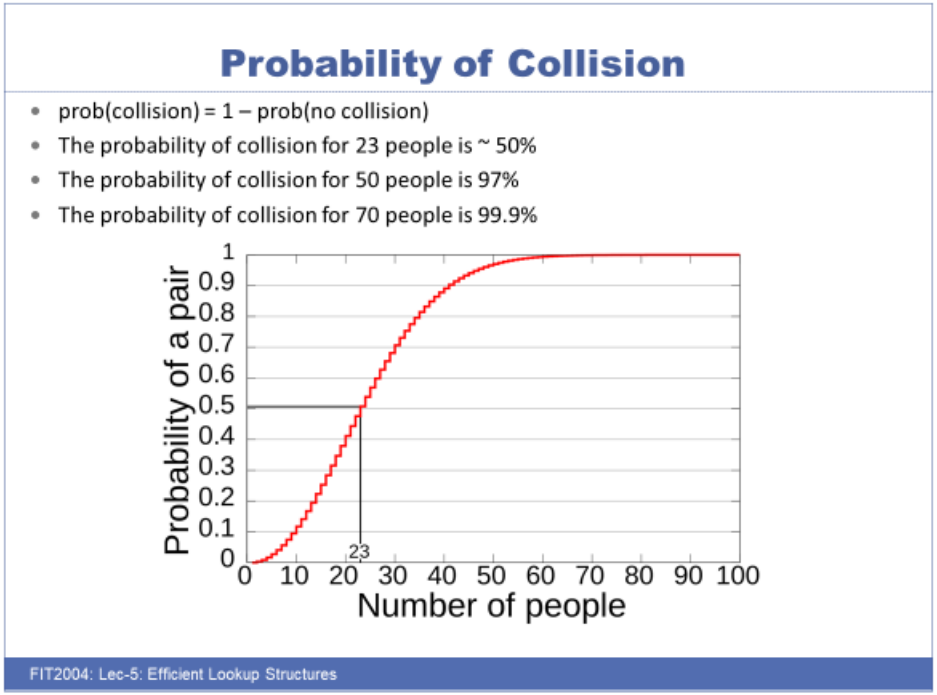
Hash Table
Avoiding Collisions

- So we see a collision…
- Can we avoid it?
  - Good hash function
  - Kinda too hard…
  - Unless our array is very very very very very big…



Hash Table
Avoiding Collisions

- So we see a collision…
- Can we avoid it?
  - Good hash function
  - Kinda too hard…
  - Unless our array is very very very very very big… M >> N



MONASH University

Questions?

Hash Table

Collision resolution

- We learnt from FIT1008 before

Hash Table

Collision resolution

- Open addressing
- Separate chaining

Hash Table

Collision resolution

MONASH University

- Open addressing
  - Linear probe
  - Quadratic probe
  - Double hashing
- Separate chaining

Hash Table
Collision resolution

MONASH University

- Open addressing aka closed hashing
  - Linear probe
  - Quadratic probe
  - Double hashing
- Separate chaining aka closed addressing

Hash Table

Collision resolution

- Open addressing aka closed hashing
  - Linear probe
  - Quadratic probe
  - Double hashing
- Separate chaining aka closed addressing

Hash Table

Collision resolution

- Open addressing aka closed hashing
  - Linear probe
  - Quadratic probe
  - Double hashing
  - Cuckoo hashing
- Separate chaining aka closed addressing

Hash Table

Collision resolution

- Open addressing aka closed hashing
  - Linear probe
  - Quadratic probe
  - Double hashing
  - Cuckoo hashing
- Separate chaining aka closed addressing

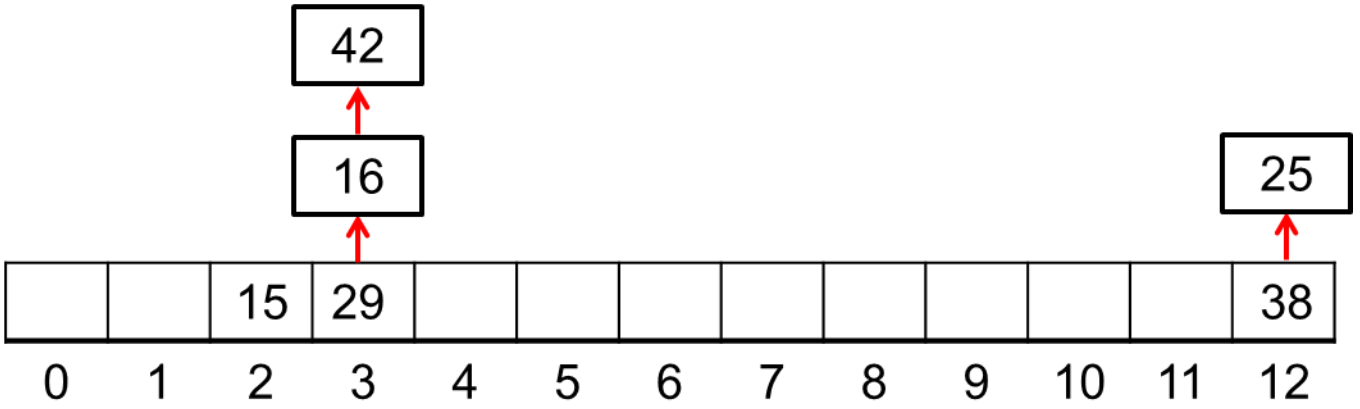
- Just use **open addressing** and **separate chaining**

MONASH University

Questions?

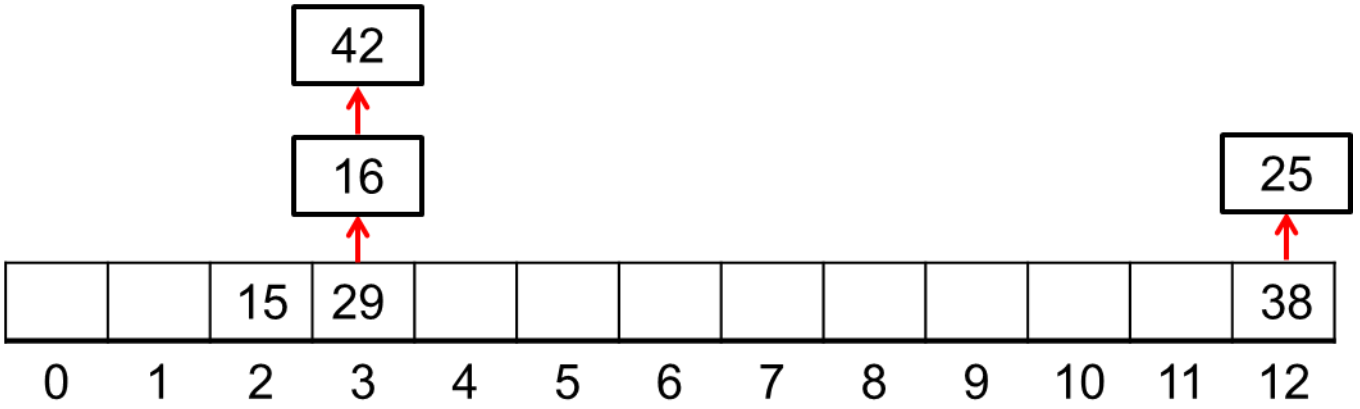
Hash Table
Collision resolution

- Separate chaining
  - Straight forward



Hash Table
Collision resolution

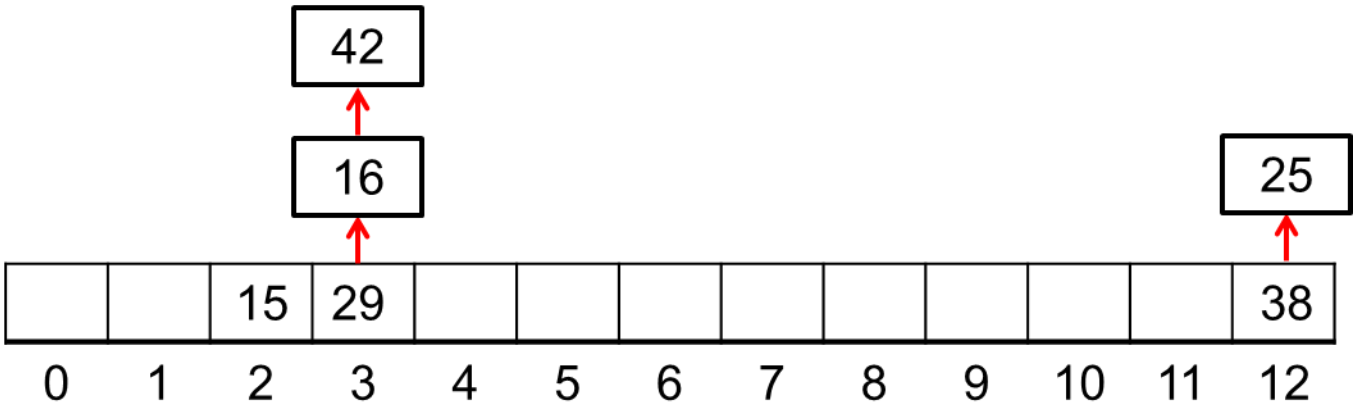
- Separate chaining
  - Straight forward
  - Complexity to travel through the chain for operations



Hash Table
Collision resolution

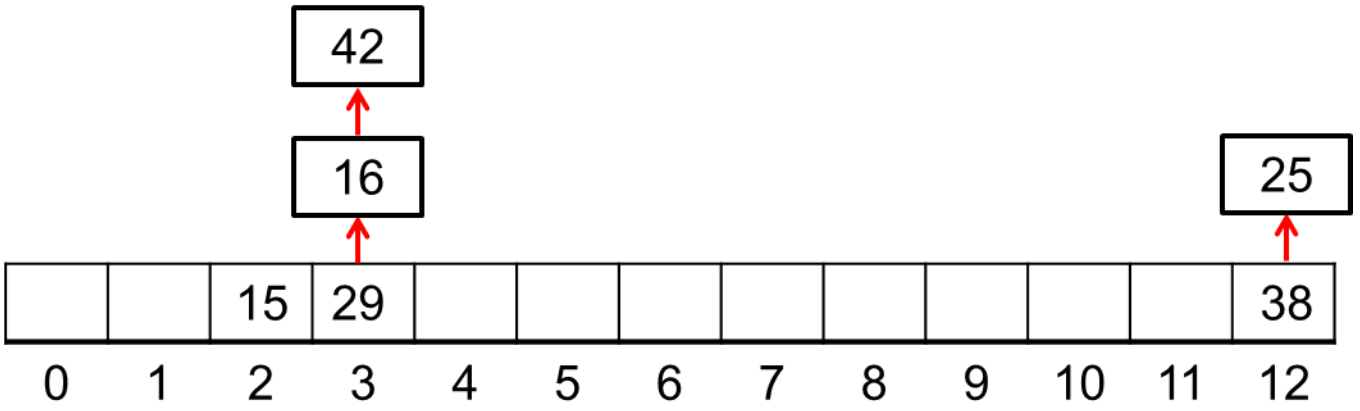
MONASH University

- Separate chaining
  - Straight forward
  - Complexity to travel through the chain for operations
  - The chain can be anything!



Hash Table
Collision resolution

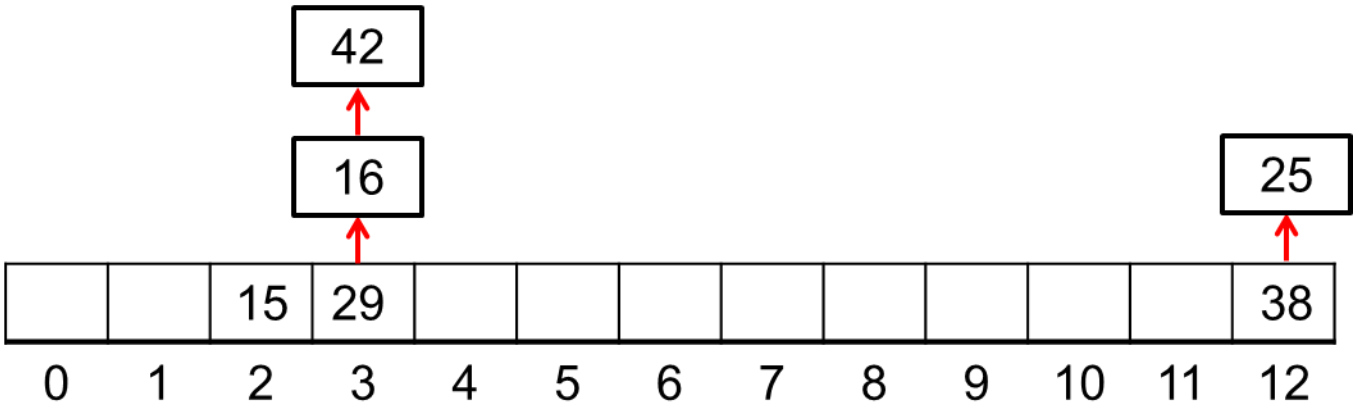
- Separate chaining
  - Straight forward
  - Complexity to travel through the chain for operations
  - The chain can be anything!
    - List
    - Another hash table
    - Tree



MONASH University

- Separate chaining
  - Straight forward
  - Complexity to travel through the chain for operations
  - The chain can be anything!
    - List
    - Another hash table
    - Tree

And many more!



MONASH University

Questions?

Hash Table
Collision resolution

MONASH University

- Open addressing
  - Linear probe
  - Quadratic probe

Hash Table
Collision resolution

- Open addressing
  - Linear probe
  - Quadratic probe
  - Both from FIT1008

Hash Table
Collision resolution

MONASH University

- Open addressing
  - Linear probe
  - Quadratic probe
  - Both from FIT1008
  - Complexity is travelling through the probe length…
    - Primary clustering
    - Secondary clustering

Hash Table
Collision resolution

MONASH University

- Open addressing
  - Linear probe
  - Quadratic probe
  - Both from FIT1008
  - Complexity is travelling through the probe length…
    - Primary clustering
    - Secondary clustering
    - Might need to resize
      (especially for quadratic probing as we might not find empty space)

Hash Table
Collision resolution

MONASH University

- Open addressing
  - Linear probe
  - Quadratic probe
  - Double hashing
  - Complexity is travelling through the probe length…
    - Primary clustering
    - Secondary clustering
    - Might need to resize
      (especially for quadratic probing as we might not find empty space)

Hash Table
Collision resolution

MONASH University

- Open addressing
  - Linear probe
  - Quadratic probe
  - Double hashing
    - 2nd hash function to determine the jumps/ probe
    - Instead of fixed
  - Complexity is travelling through the probe length…
    - Primary clustering
    - Secondary clustering
    - Might need to resize
      (especially for quadratic probing as we might not find empty space)

MONASH University

Questions?

MONASH University

- Now this is something new…

MONASH University

- Now this is something new…
  - In double hashing, we use a 2nd hash function

Hash Table
Cuckoo hashing

MONASH University

- Now this is something new…
  - In double hashing, we use a 2nd hash function
  - For cuckoo hashing, we have a 2nd hash table!

Hash Table
Cuckoo hashing

MONASH University

- Now this is something new…
  - In double hashing, we use a 2nd hash function
  - For cuckoo hashing, we have a 2nd hash table!
  - Items are kicked between tables when there is collision
    - From 1st table to 2nd table
    - From 2nd table to 1st table

Hash Table
Cuckoo hashing

MONASH University

- Now this is something new…
  - In double hashing, we use a 2nd hash function
  - For cuckoo hashing, we have a 2nd hash table!
  - Items are kicked between tables when there is collision
    - From 1st table to 2nd table
    - From 2nd table to 1st table
    - Let us look at an example (more to come in the tutorial)

MONASH University

- Let us try to insert the following keys…

MONASH University

- Let us try to insert the following keys…
  - 23
  - 36
  - 114
  - 49

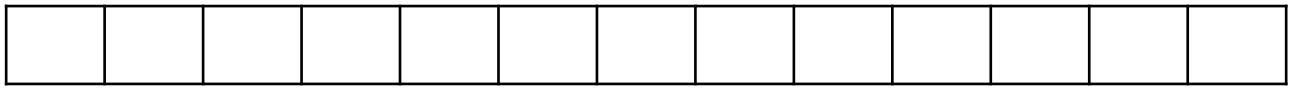
MONASH University

- Let us try to insert the following keys…
  - 23
  - 36
  - 114
  - 49
- Into the following tables…

Hash Table
Cuckoo hashing

MONASH University

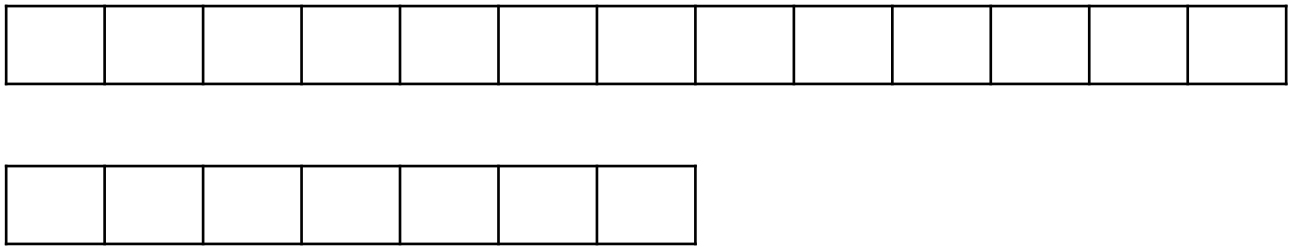
- Let us try to insert the following keys…
  - 23
  - 36
  - 114
  - 49
- Into the following tables…
  - Table 01, of size 13



Hash Table
Cuckoo hashing

MONASH University

- Let us try to insert the following keys…
  - 23
  - 36
  - 114
  - 49
- Into the following tables…
  - Table 01, of size 13
  - Table 02, of size 7

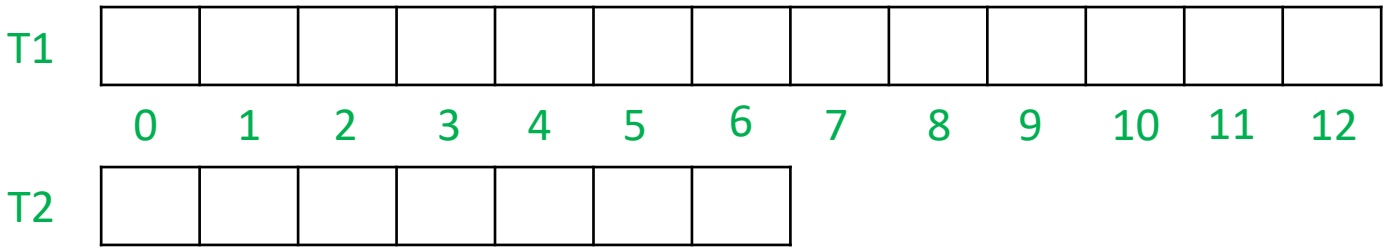


Hash Table

Cuckoo hashing

MONASH University

- Let us try to insert the following keys…
  - 23
  - 36
  - 114
  - 49
- Into the following tables…
  - Table 01, of size 13
  - Table 02, of size 7

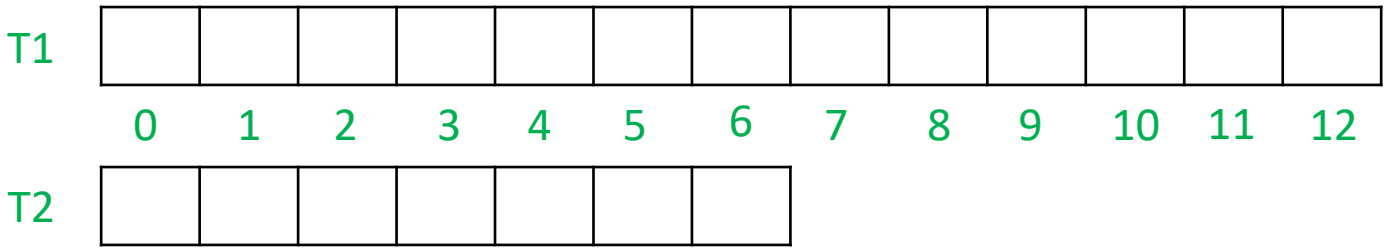


Hash Table

Cuckoo hashing

MONASH University

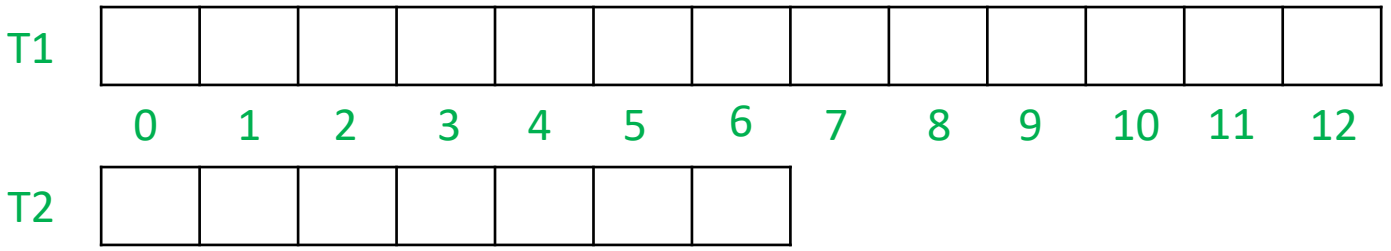
- Let us try to insert the following keys…
  - 23
  - 36
  - 114
  - 49
- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

- Insert 23
  - Hash1(23) =

- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7

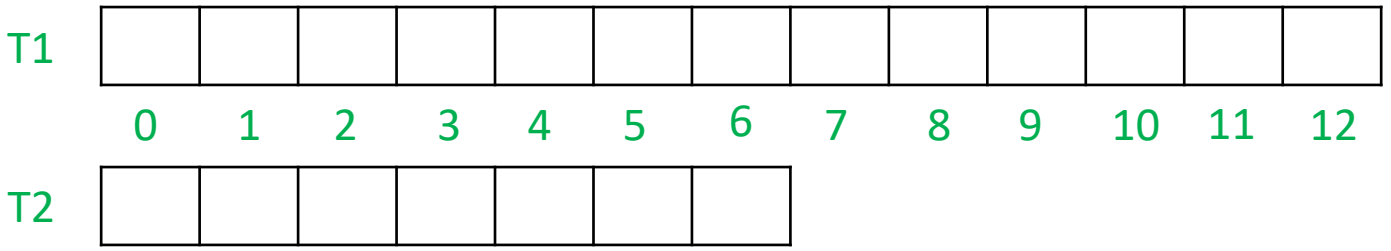


Hash Table
Cuckoo hashing

MONASH University

- Insert 23
  - Hash1(23) = 10 #QuikMath

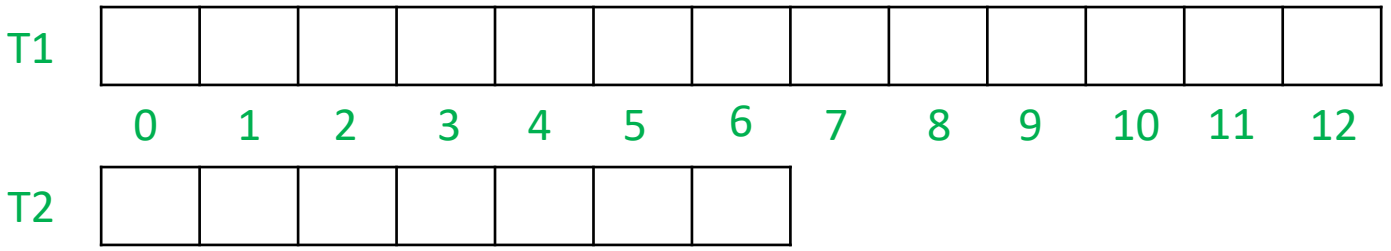
- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

- Insert 23
  - Hash1(23) = 10

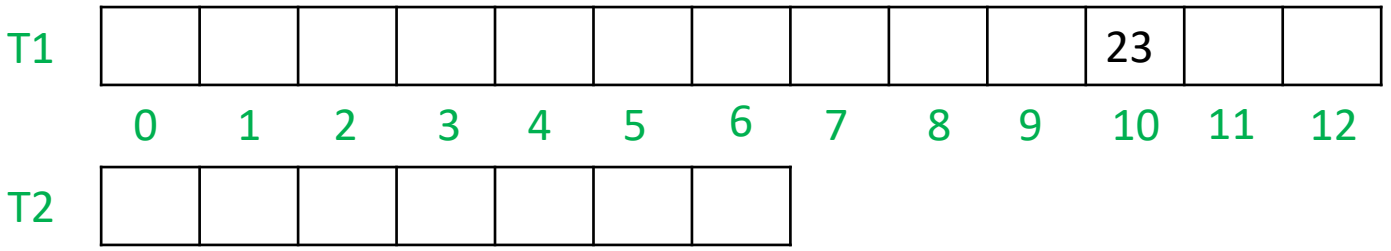
- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

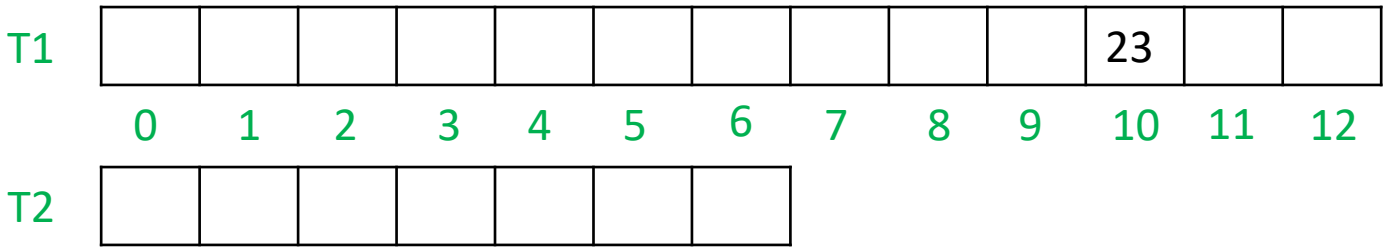
- Insert 23
  - Hash1(23) = 10

- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



- Insert 36
  - Hash1(36) = 10

- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7

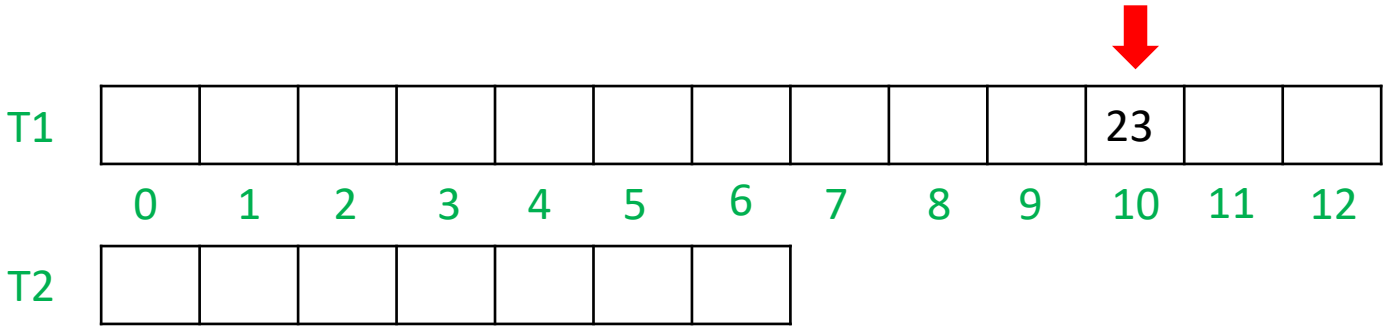


Hash Table
Cuckoo hashing

MONASH University

- Insert 36
  - Hash1(36) = 10 … we have collision

- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7

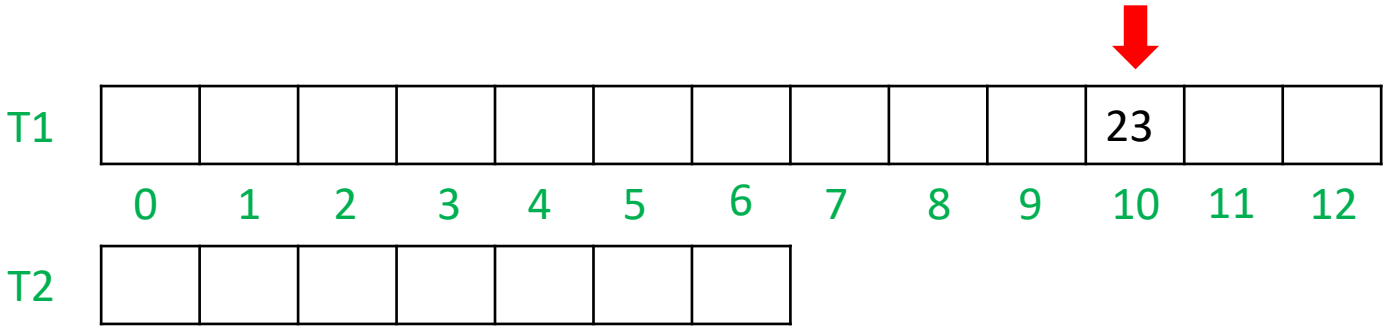


Hash Table
Cuckoo hashing

MONASH University

- Insert 36
  - Hash1(36) = 10 … we have collision

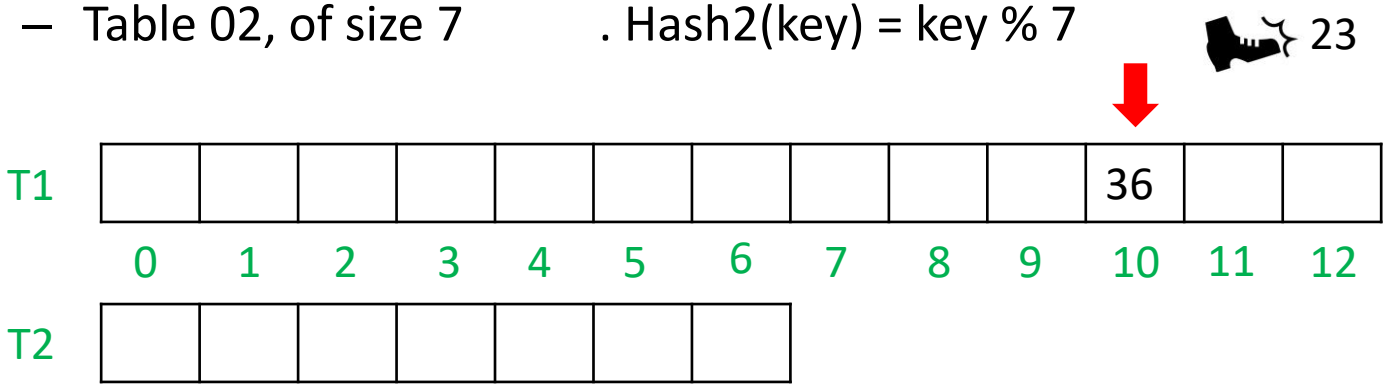
- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

- Insert 36
  - Hash1(36) = 10 … we have collision, so we kick 23 for 36

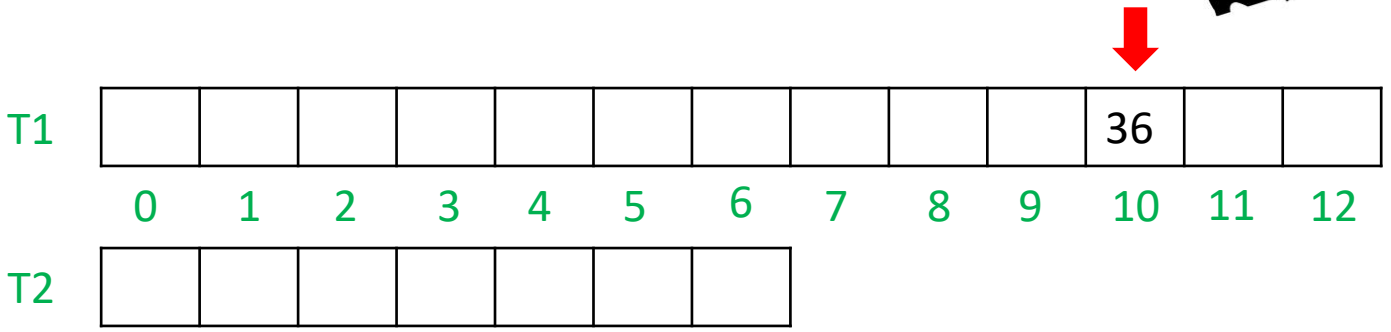
- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

- Insert 36
  - Hash1(36) = 10 … we have collision, so we kick 23 for 36
  - Hash2(23) = 2

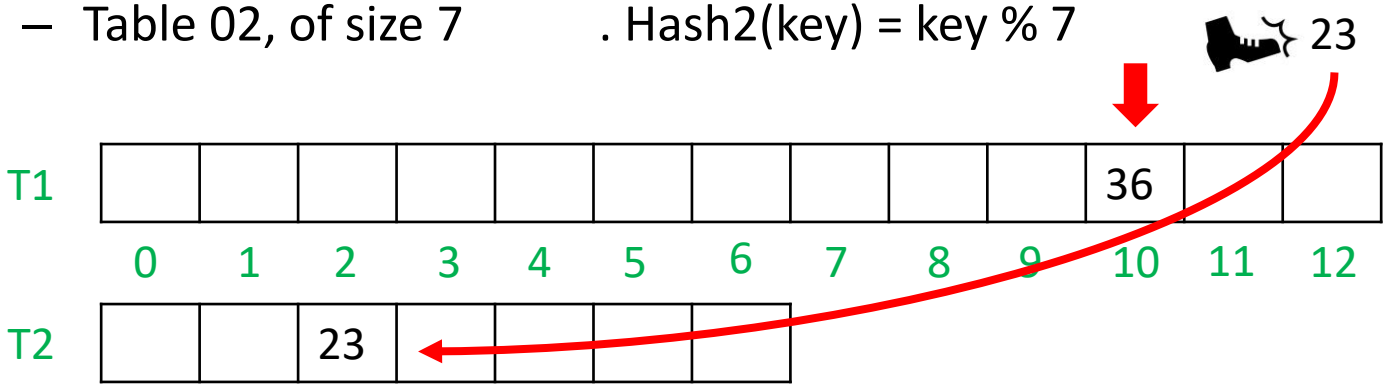
- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

- Insert 36
  - Hash1(36) = 10 … we have collision, so we kick 23 for 36
  - Hash2(23) = 2

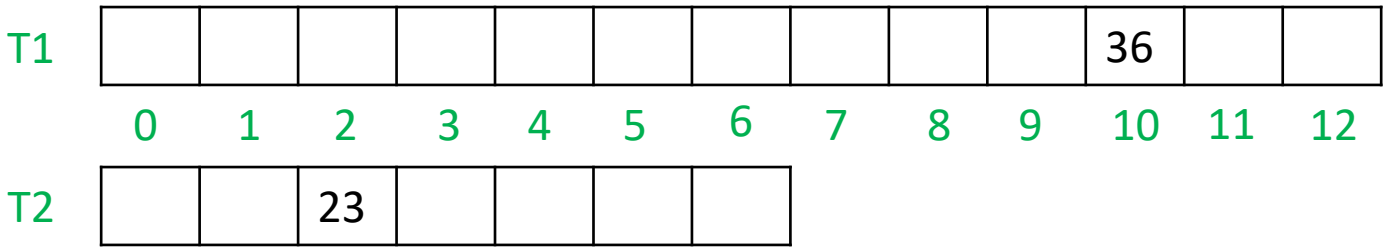
- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

- Insert 36
  - Hash1(36) = 10 … we have collision, so we kick 23 for 36
  - Hash2(23) = 2

- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7

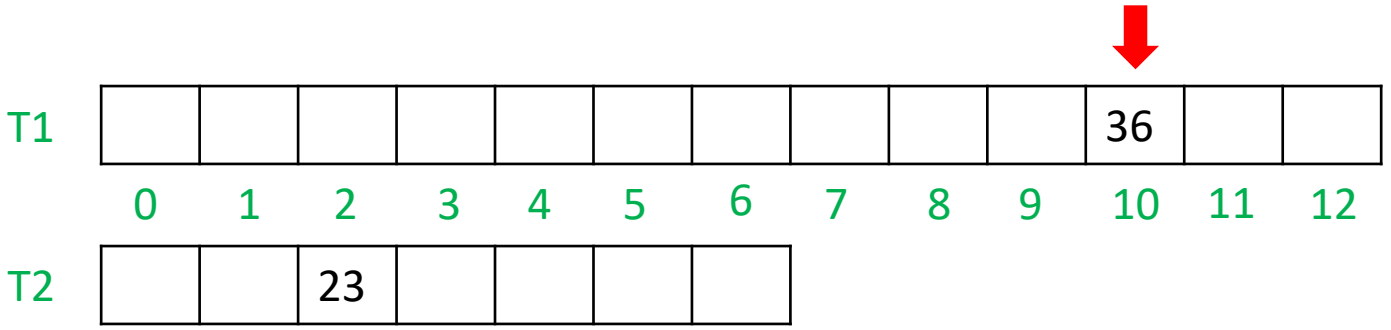


Hash Table
Cuckoo hashing

MONASH University

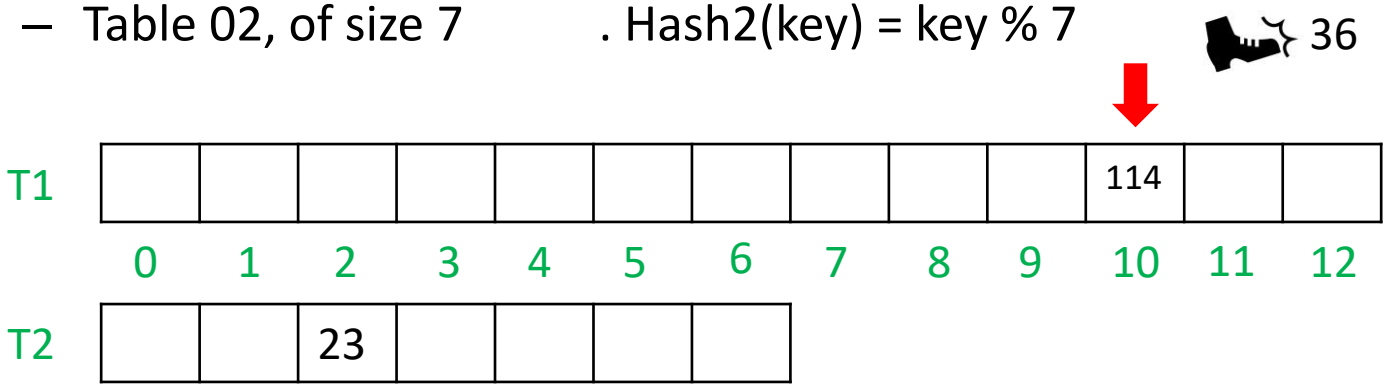
- Insert 114
  - Hash1(114) = 10 … we have collision

- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



- Insert 114
  - Hash1(114) = 10 … we have collision, so we kick 36 for 114

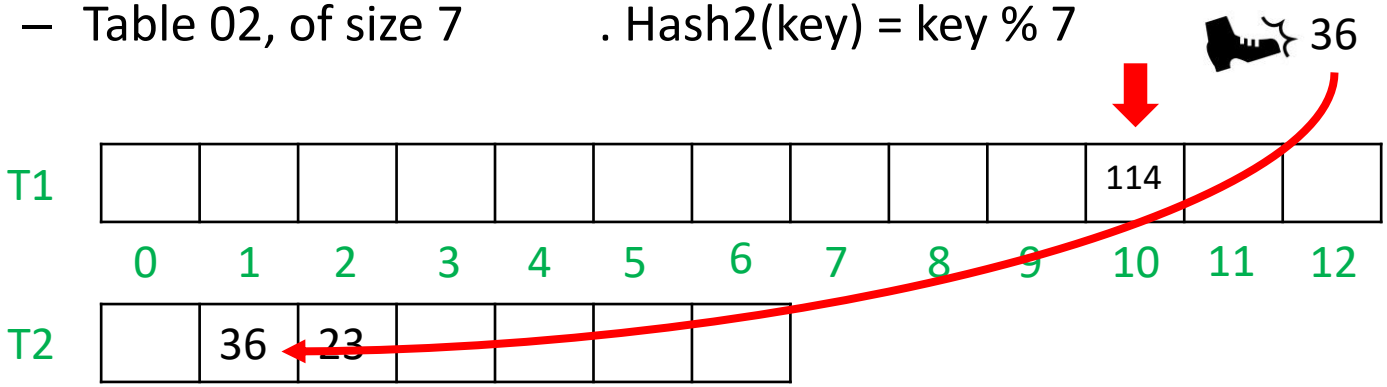
- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

- Insert 114
  - Hash1(114) = 10 … we have collision, so we kick 36 for 114
  - Hash2(36) = 1

- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7

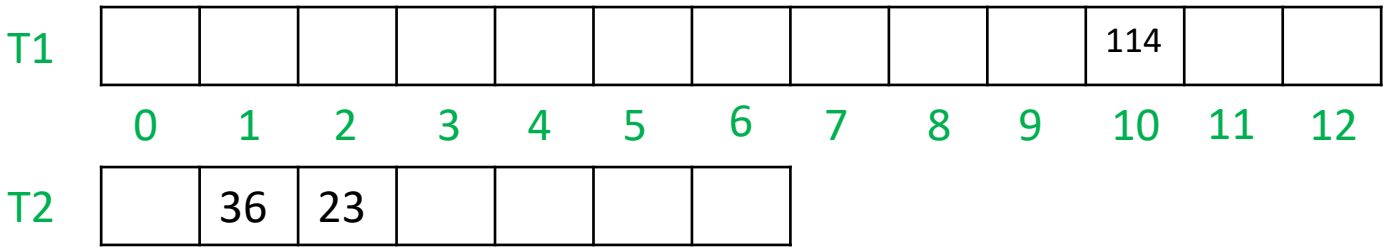


Hash Table
Cuckoo hashing

MONASH University

- Insert 114
  - Hash1(114) = 10 … we have collision, so we kick 36 for 114
  - Hash2(36) = 1

- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7

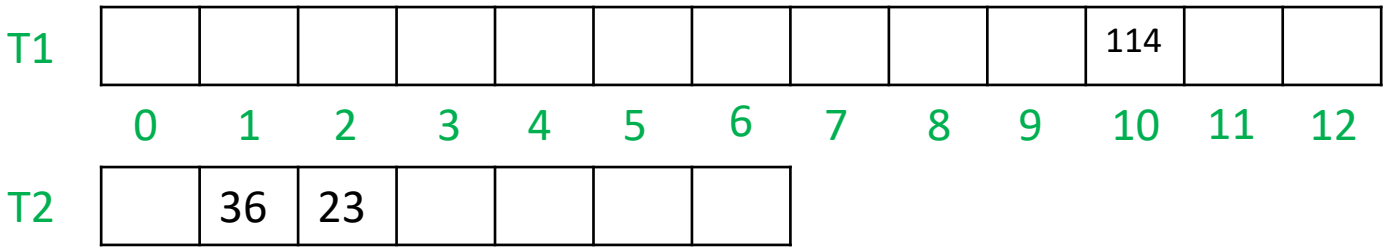


Hash Table
Cuckoo hashing

MONASH University

- Insert 49
  - Hash1(49) = 10

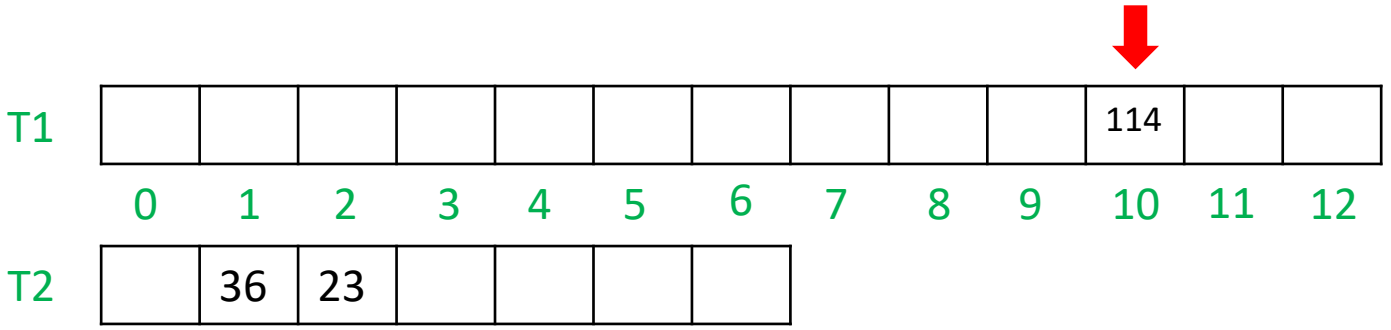
- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

- Insert 49
  - Hash1(49) = 10… collision!

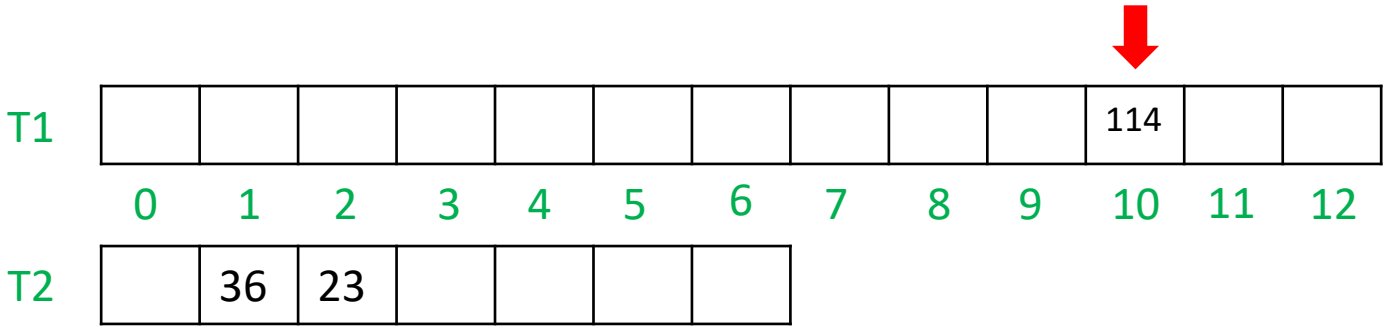
- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

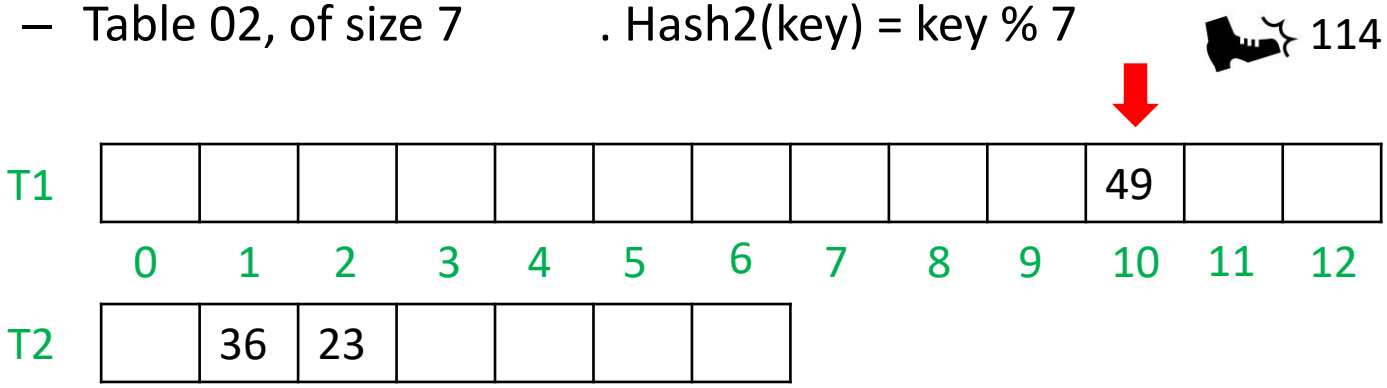
- Insert 49
  - Hash1(49) = 10… collision! Kick 114 for 49

- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



- Insert 49
  - Hash1(49) = 10… collision! Kick 114 for 49

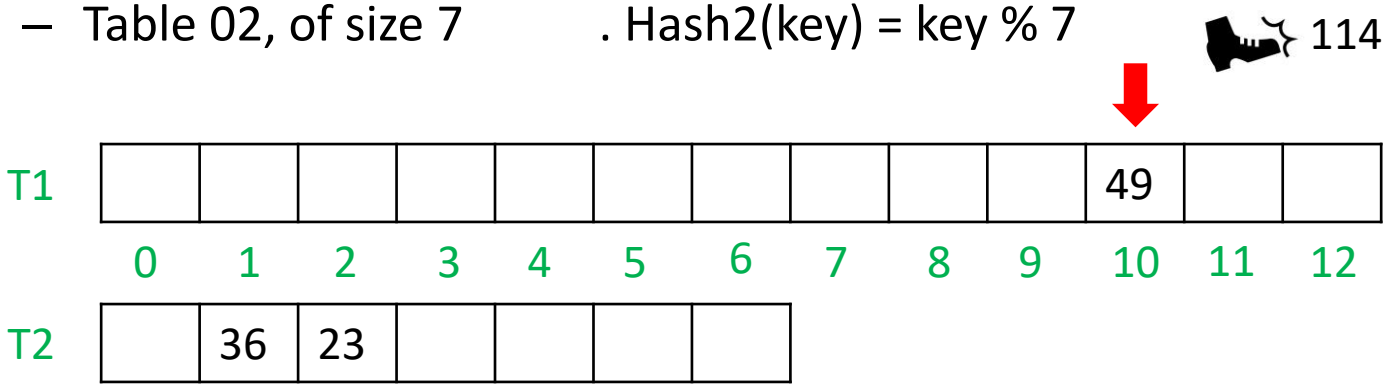
- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

- Insert 49
  - Hash1(49) = 10… collision! Kick 114 for 49
  - Hash2(114) = 2

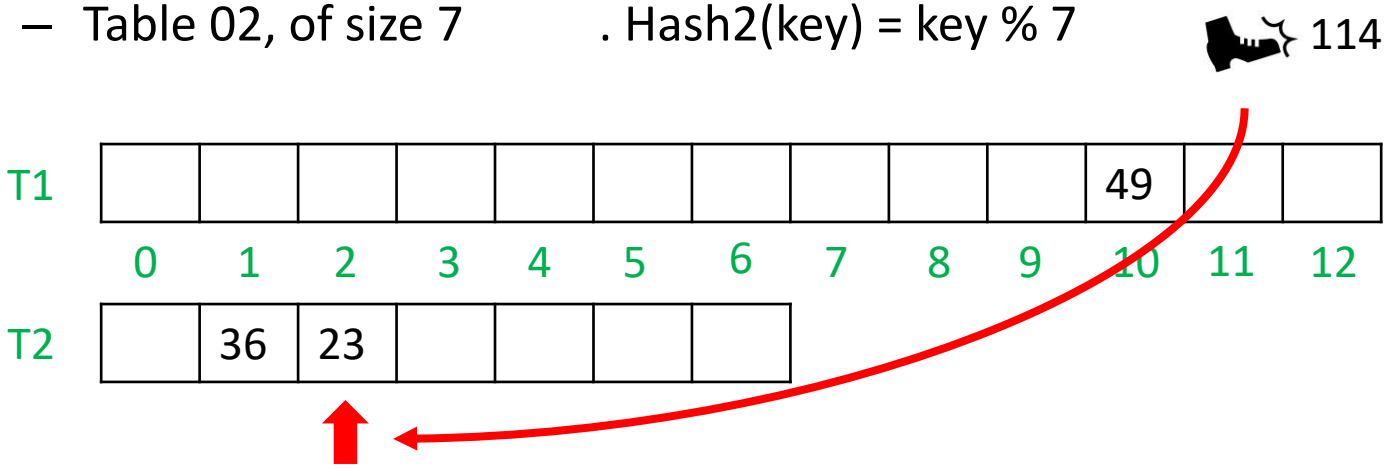
- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

- Insert 49
  - Hash1(49) = 10… collision! Kick 114 for 49
  - Hash2(114) = 2… collision!

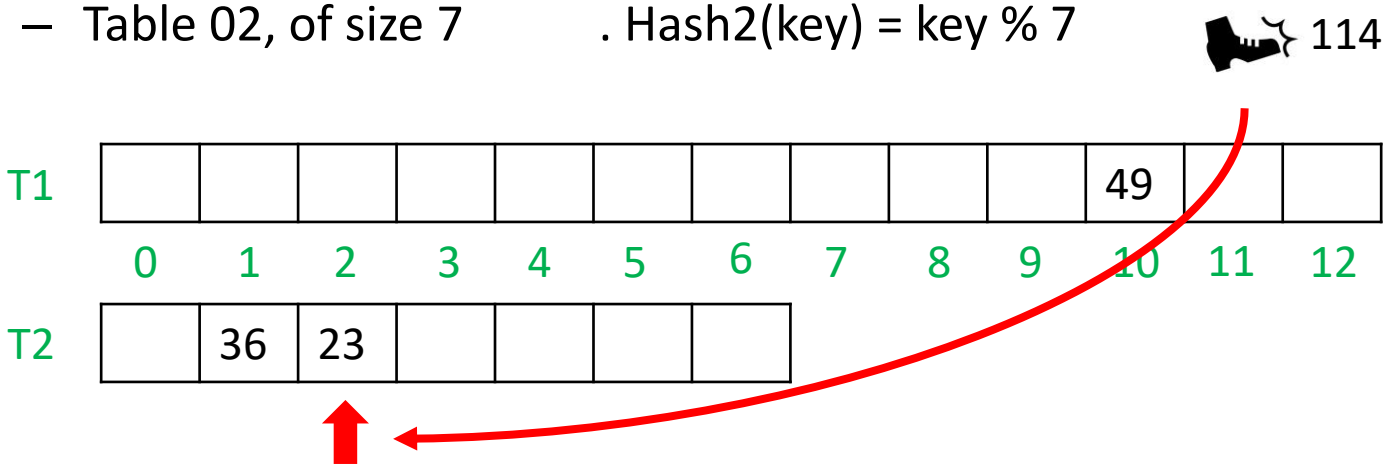
- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

- Insert 49
  - Hash1(49) = 10… collision! Kick 114 for 49
  - Hash2(114) = 2… collision! Kick 23 for 114

- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7

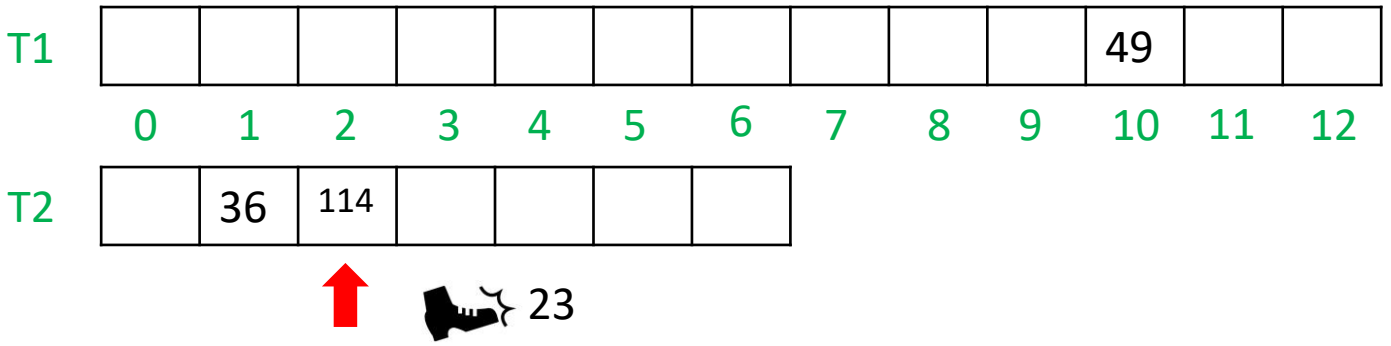


Hash Table
Cuckoo hashing

MONASH University

- Insert 49
  - Hash1(49) = 10… collision! Kick 114 for 49
  - Hash2(114) = 2… collision! Kick 23 for 114

- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7

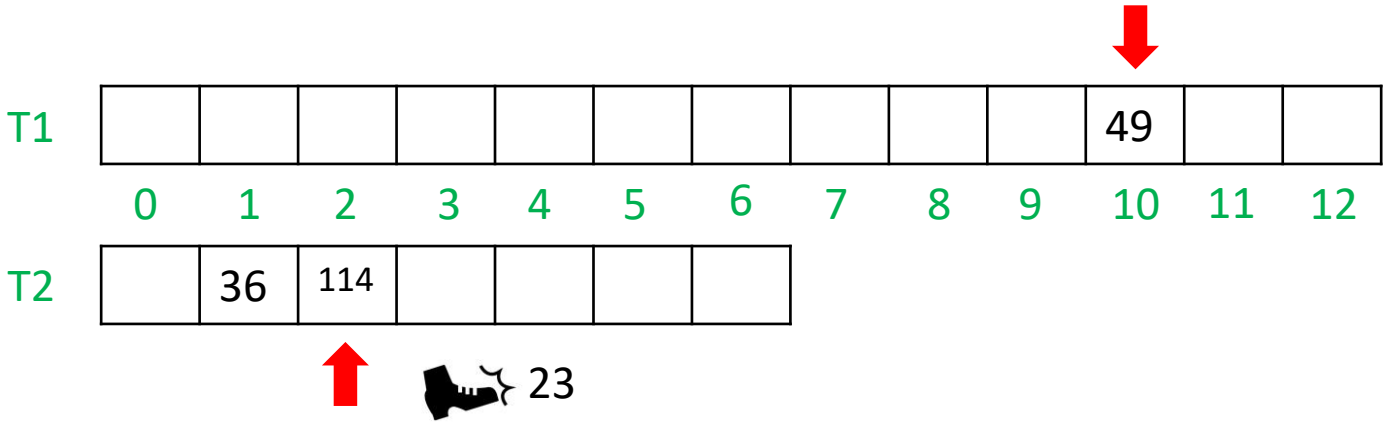


Hash Table
Cuckoo hashing

MONASH University

- Insert 49
  - Hash1(49) = 10… collision! Kick 114 for 49
  - Hash2(114) = 2… collision! Kick 23 for 114
  - Hash1(23) = 10

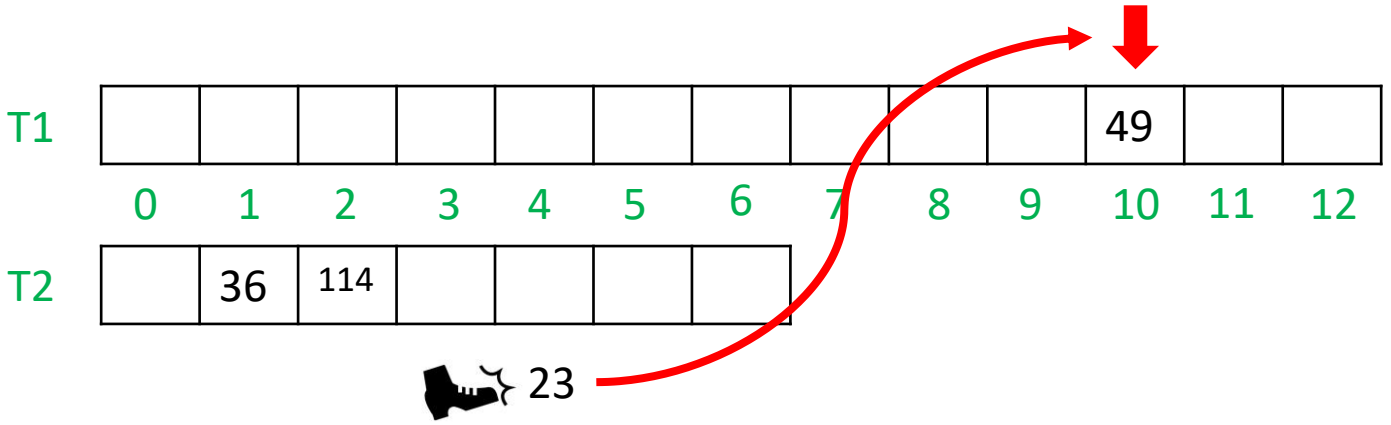
- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

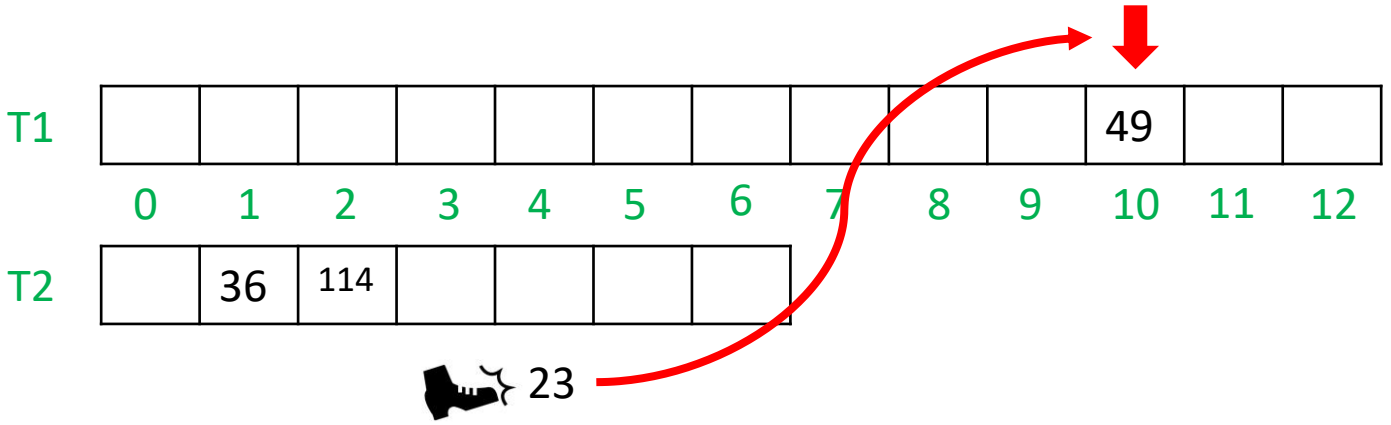
- Insert 49
  - Hash1(49) = 10… collision! Kick 114 for 49
  - Hash2(114) = 2… collision! Kick 23 for 114
  - Hash1(23) = 10… collision!

- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



- Insert 49
  - Hash1(49) = 10… collision! Kick 114 for 49
  - Hash2(114) = 2… collision! Kick 23 for 114
  - Hash1(23) = 10… collision! Kick 49 for 23

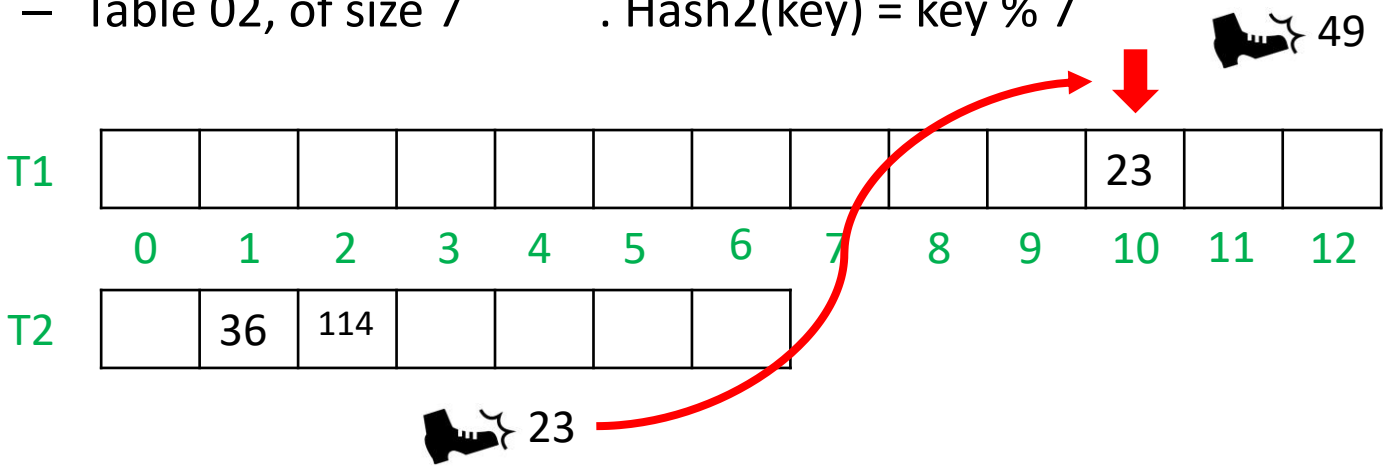
- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

- Insert 49
  - Hash1(49) = 10… collision! Kick 114 for 49
  - Hash2(114) = 2… collision! Kick 23 for 114
  - Hash1(23) = 10… collision! Kick 49 for 23

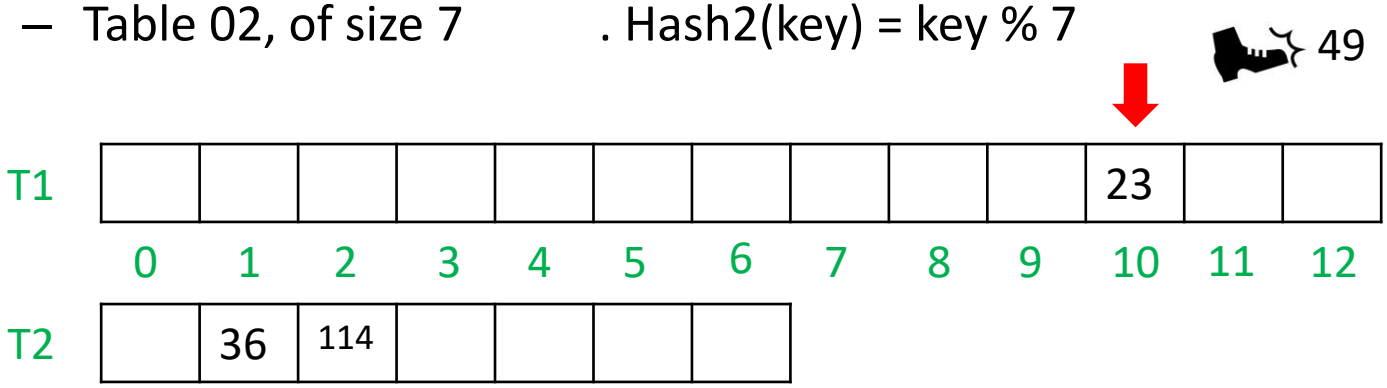
- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



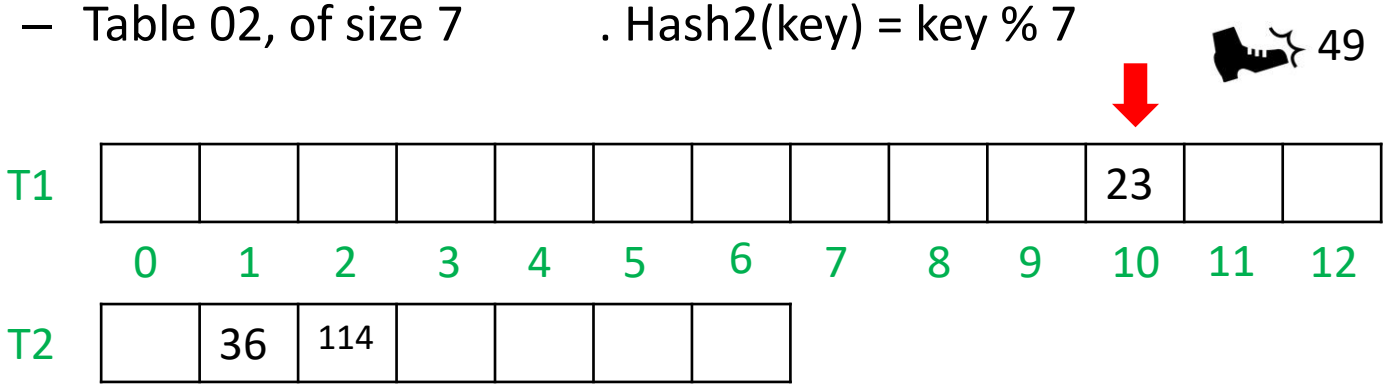
Hash Table
Cuckoo hashing

- Insert 49
  - Hash1(49) = 10… collision! Kick 114 for 49
  - Hash2(114) = 2… collision! Kick 23 for 114
  - Hash1(23) = 10… collision! Kick 49 for 23

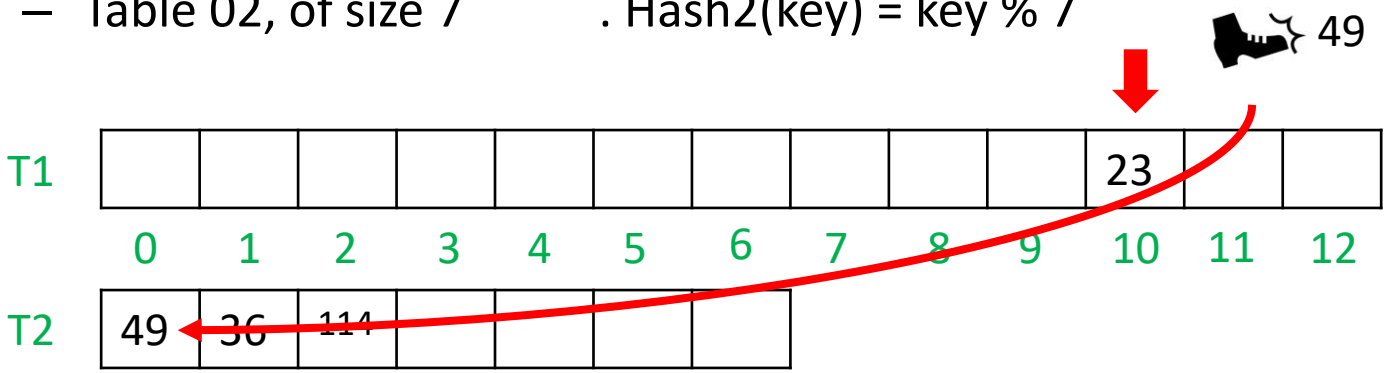
- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



- Insert 49
  - Hash1(49) = 10… collision! Kick 114 for 49
  - Hash2(114) = 2… collision! Kick 23 for 114
  - Hash1(23) = 10… collision! Kick 49 for 23
  - Hash1(49) = 0
- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7

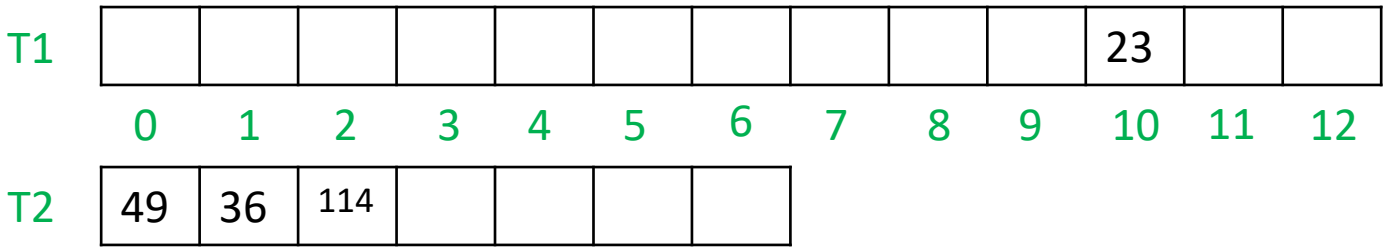


- Insert 49
  - Hash1(49) = 10… collision! Kick 114 for 49
  - Hash2(114) = 2… collision! Kick 23 for 114
  - Hash1(23) = 10… collision! Kick 49 for 23
  - Hash1(49) = 0
- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

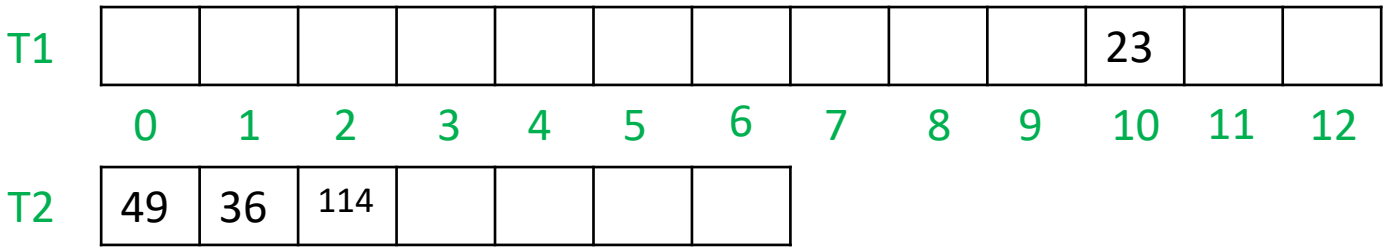
- Insert 49
  - Hash1(49) = 10… collision! Kick 114 for 49
  - Hash2(114) = 2… collision! Kick 23 for 114
  - Hash1(23) = 10… collision! Kick 49 for 23
  - Hash1(49) = 0 we r finished!
- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

MONASH University

- We have inserted all items in…
  - 23
  - 36
  - 114
  - 49
- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



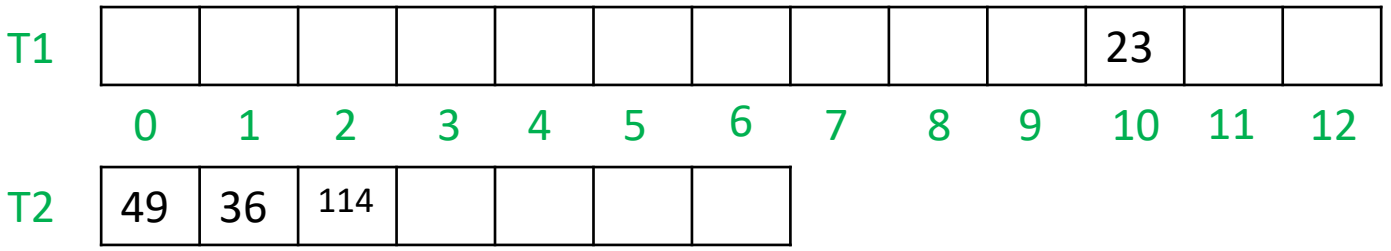
MONASH University

Questions?

MONASH University

- Now let us try and insert 140

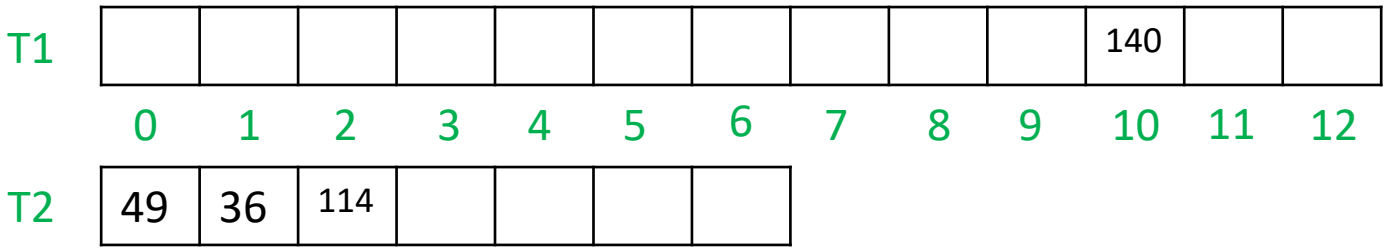
- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



MONASH University

- Now let us try and insert 140
  - Hash1(140) = 10… so we kick 23 out to T2

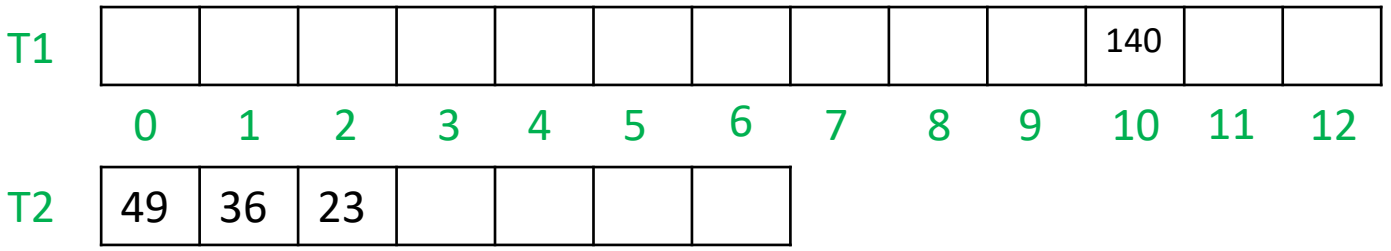
- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

- Now let us try and insert 140
  - Hash1(140) = 10… so we kick 23 out to T2
  - Hash2(23) = 2… so we kick 114 out to T1

- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7

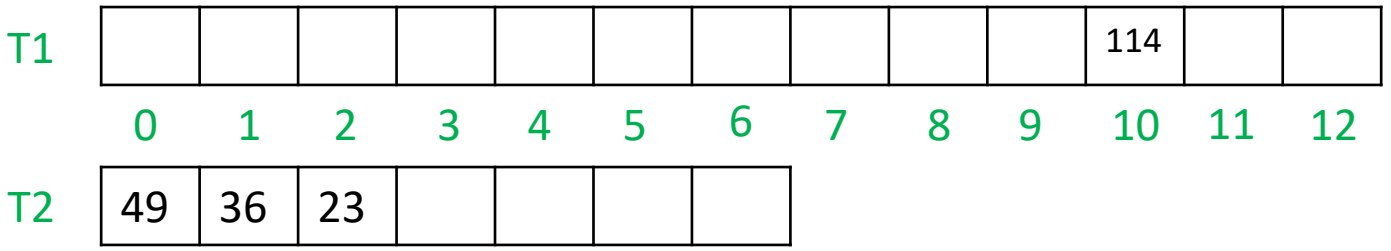


Hash Table
Cuckoo hashing

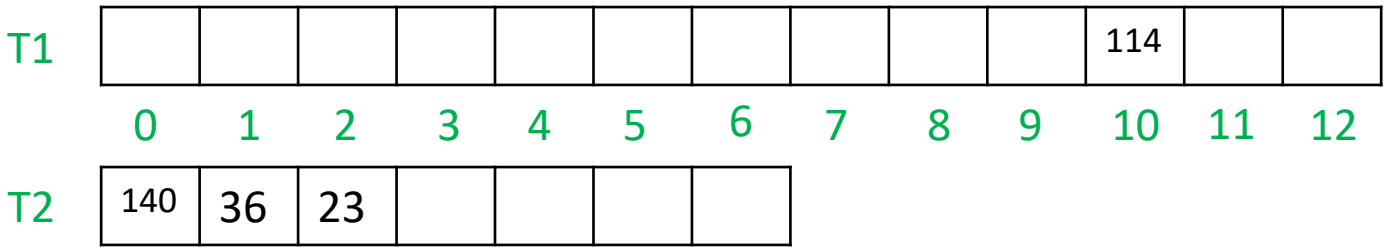
MONASH University

- Now let us try and insert 140
  - Hash1(140) = 10… so we kick 23 out to T2
  - Hash2(23) = 2… so we kick 114 out to T1
  - Hash1(114) = 10… so we kick 140 out to T2

- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7

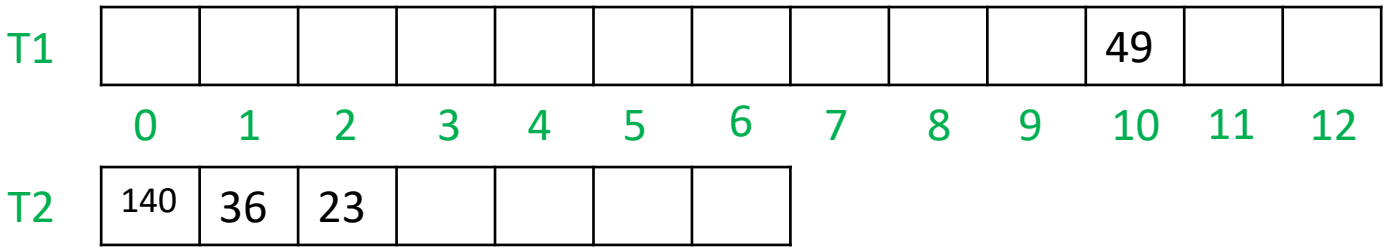


- Now let us try and insert 140
  - Hash1(140) = 10… so we kick 23 out to T2
  - Hash2(23) = 2… so we kick 114 out to T1
  - Hash1(114) = 10… so we kick 140 out to T2
  - Hash2(140) = 0… so we kick 49 out to T1
- Into the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7

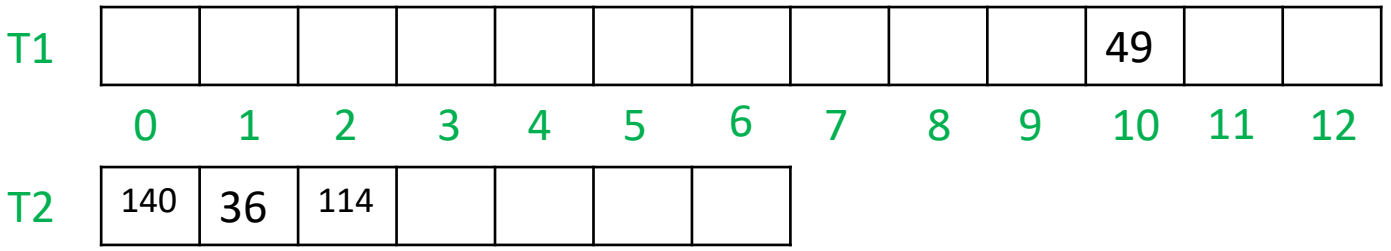


Hash Table
Cuckoo hashing

- Now let us try and insert 140
  - Hash1(140) = 10… so we kick 23 out to T2
  - Hash2(23) = 2… so we kick 114 out to T1
  - Hash1(114) = 10… so we kick 140 out to T2
  - Hash2(140) = 0… so we kick 49 out to T1
  - Hash1(49) = 10… so we kick 114 out to T2



- Now let us try and insert 140
  - Hash1(140) = 10… so we kick 23 out to T2
  - Hash2(23) = 2… so we kick 114 out to T1
  - Hash1(114) = 10… so we kick 140 out to T2
  - Hash2(140) = 0… so we kick 49 out to T1
  - Hash1(49) = 10… so we kick 114 out to T2
  - Hash2(114) = 2… so we kick 23 out to T1



Hash Table
Cuckoo hashing

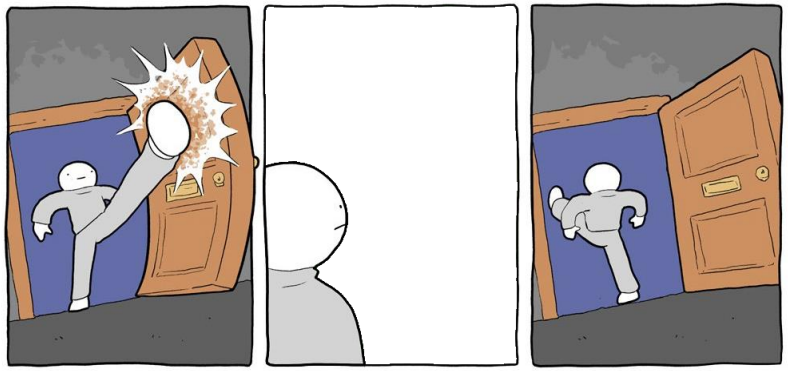
- Now let us try and insert 140
  - Hash1(140) = 10… so we kick 23 out to T2
  - Hash2(23) = 2… so we kick 114 out to T1
  - Hash1(114) = 10… so we kick 140 out to T2
  - Hash2(140) = 0… so we kick 49 out to T1
  - Hash1(49) = 10… so we kick 114 out to T2
  - Hash2(114) = 2… so we kick 23 out to T1
  - Hash1(23) = 10… so we kick….

	0	1	2	3	4	5	6	7	8	9	10	11	12
T1											49		
T2	140	36	114										

Hash Table

Cuckoo hashing

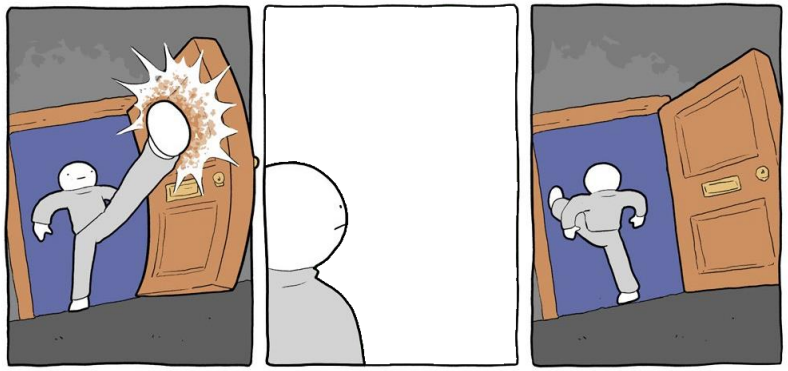
- It can be a lot of kicking…



	0	1	2	3	4	5	6	7	8	9	10	11	12
T1											49		
T2	140	36	114										

Hash Table
Cuckoo hashing

- It can be a lot of kicking…
  - Can even be infinite!

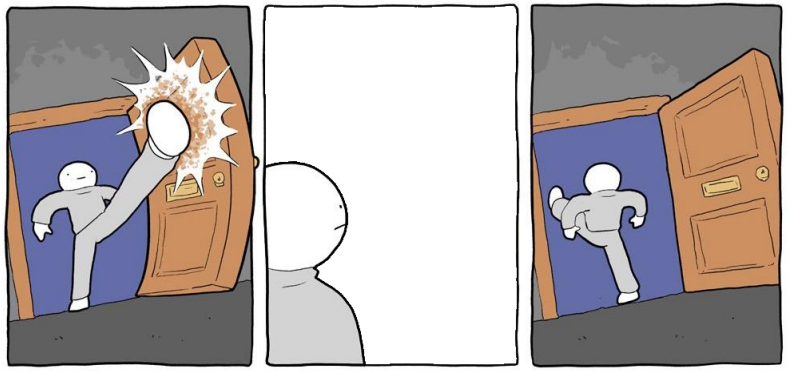


	0	1	2	3	4	5	6	7	8	9	10	11	12
T1											49		

T2	140	36	114			

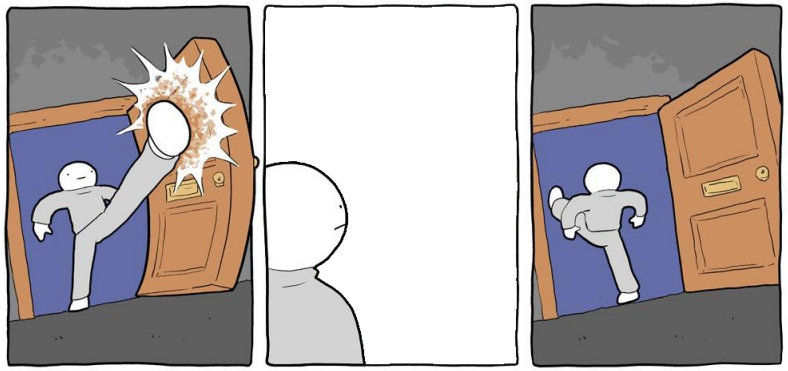
Hash Table
Cuckoo hashing

- It can be a lot of kicking…
  - Can even be infinite!
  - So how?



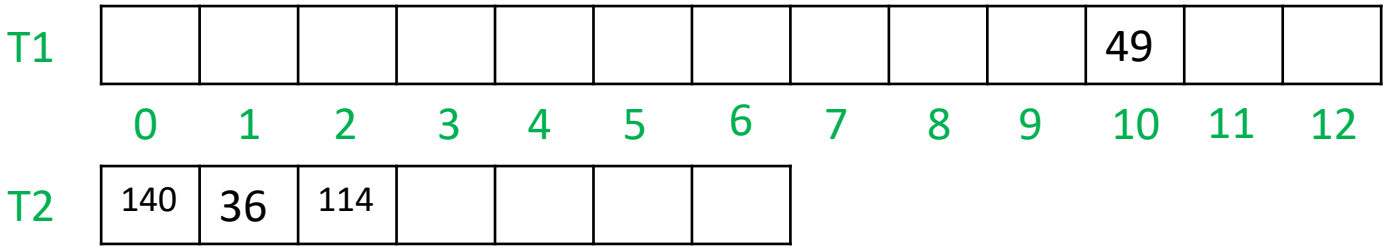
T1											49		
	0	1	2	3	4	5	6	7	8	9	10	11	12
T2	140	36	114										

Hash Table
Cuckoo hashing



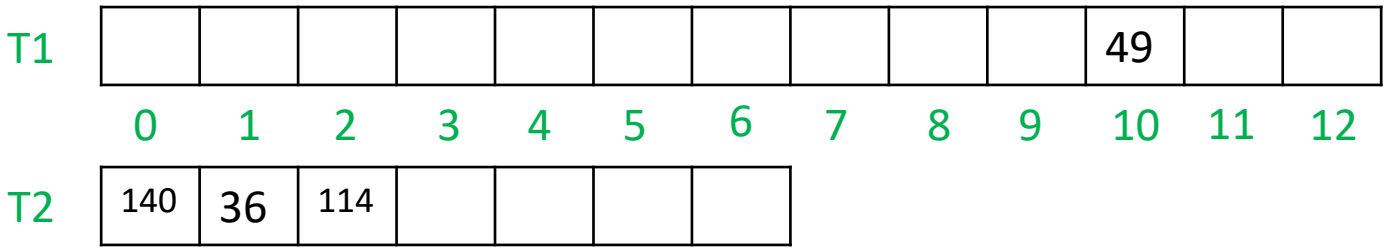
- It can be a lot of kicking…
  - Can even be infinite!
  - So how?

- Set a counter…
  - If count_kick >= max_kick: resize!



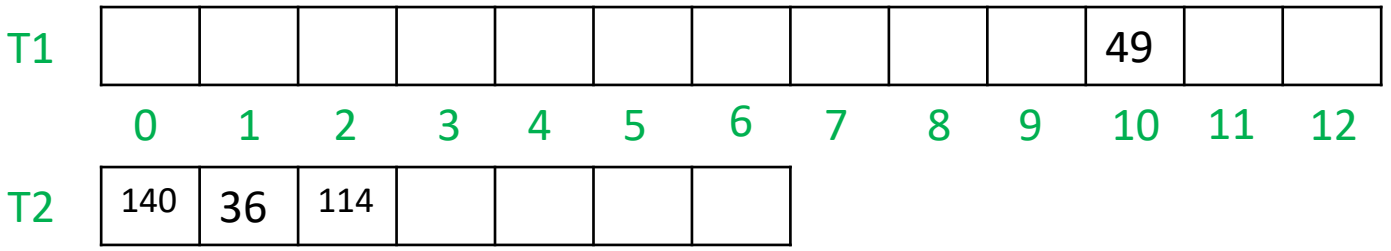
Hash Table
Cuckoo hashing

- Complexity?



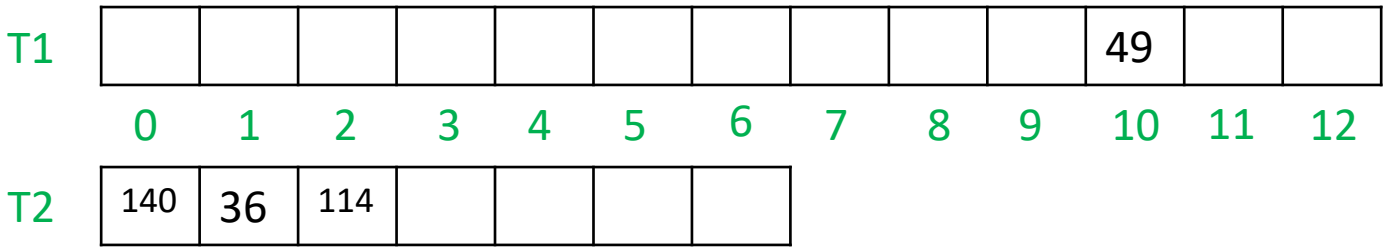
Hash Table
Cuckoo hashing

- Complexity?
  - Insert can be O(1) when the T1[Hash1(key)] is empty



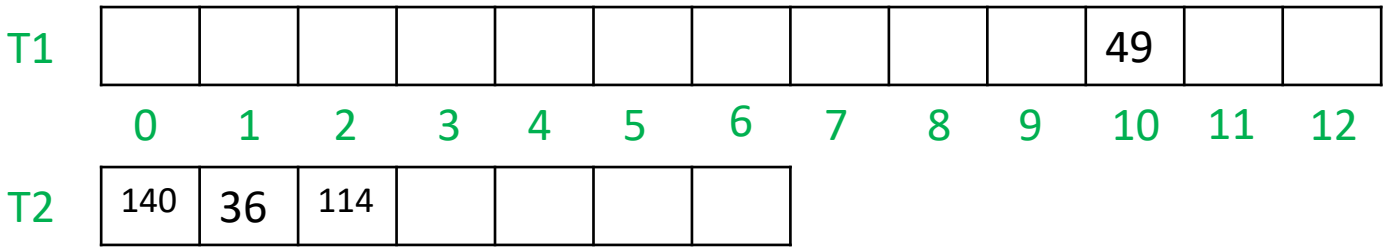
Hash Table
Cuckoo hashing

- Complexity?
  - Insert can be O(1) when the T1[Hash1(key)] is empty
  - Else we need to keep kicking…



Hash Table
Cuckoo hashing

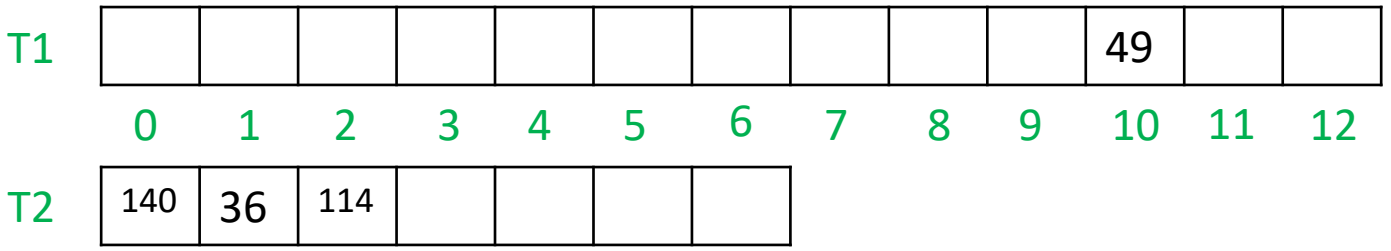
- Complexity?
  - Insert can be O(1) when the T1[Hash1(key)] is empty
  - Else we need to keep kicking…
    - Up to (K) where K is the maximum kick possible



Hash Table
Cuckoo hashing

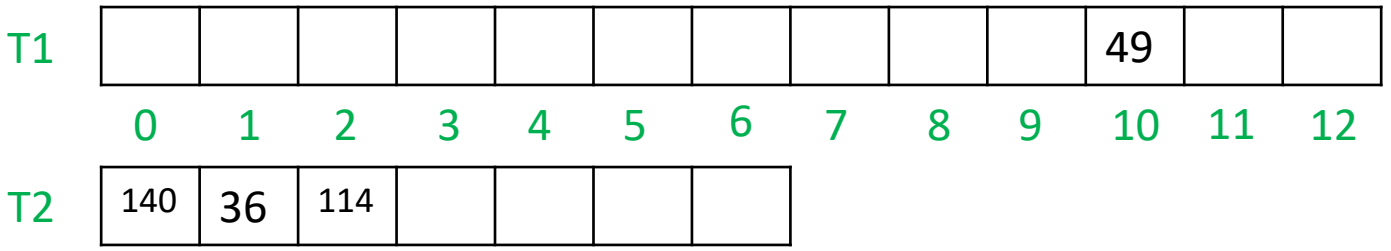
MONASH University

- Complexity?
  - Insert can be O(1) when the T1[Hash1(key)] is empty
  - Else we need to keep kicking…
    - Up to (K) where K is the maximum kick possible
  - If after K-kicks, still can't put key…
    - Resize!



Hash Table
Cuckoo hashing

- Complexity?
  - Insert can be O(1) when the T1[Hash1(key)] is empty
  - Else we need to keep kicking…
    - Up to (K) where K is the maximum kick possible
  - If after K-kicks, still can't put key…
    - Resize!
    - O(N) to resize one of the tables
    - Where N is the number of keys in the selected table



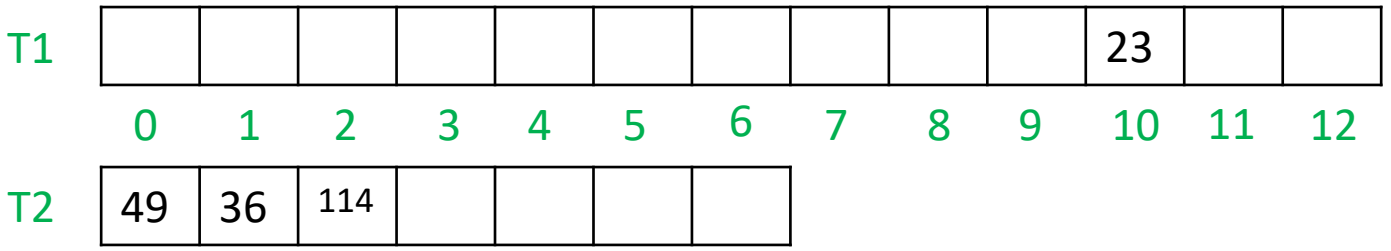
MONASH University

Questions?

MONASH University

- Searching

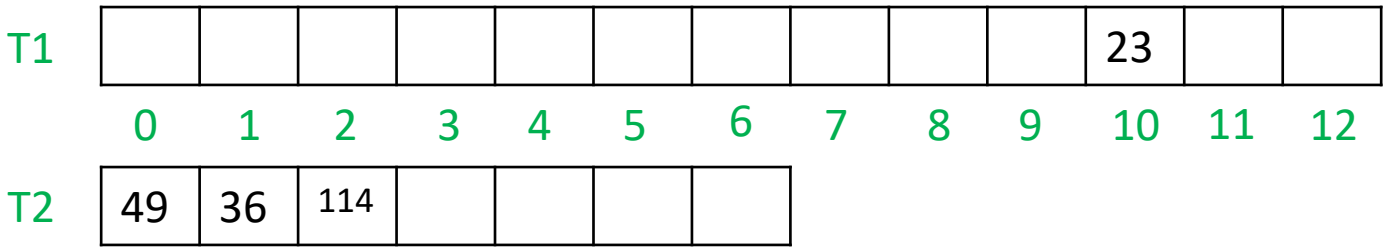
- Using the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

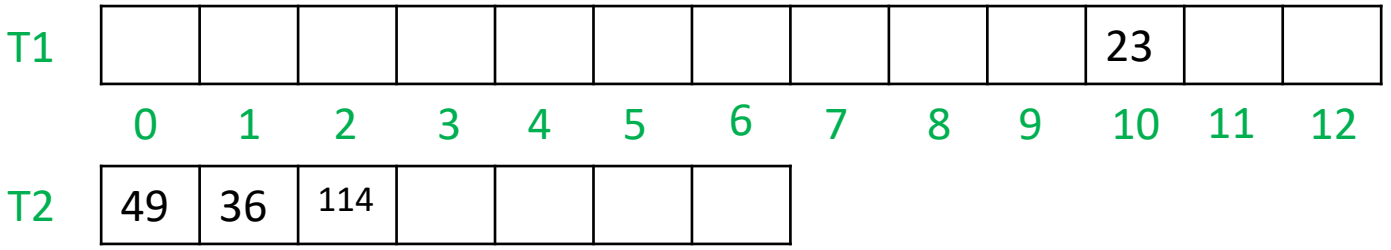
- Searching
  - The item can only appear in either T1 or T2…

- Using the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

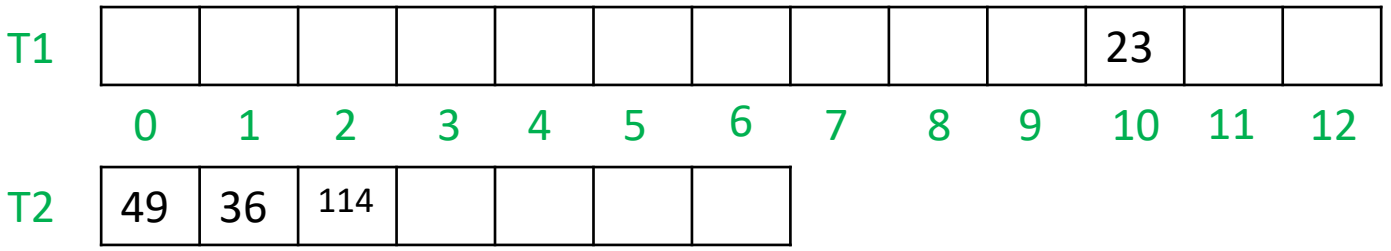
- Searching
  - The item can only appear in either T1 or T2…
  - So we only need to hash each item a maximum of 2-times
    - Hash for T1
    - If not in T1, hash for T2
- Using the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

- Searching for key 36

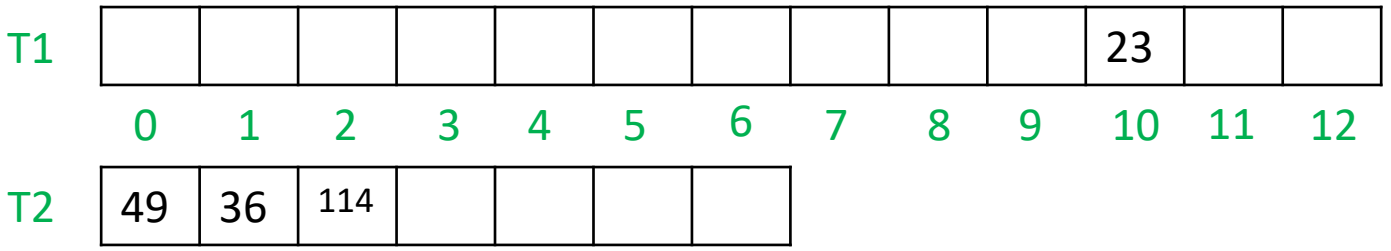
- Using the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

- Searching for key 36
  - Hash1(36) = 10

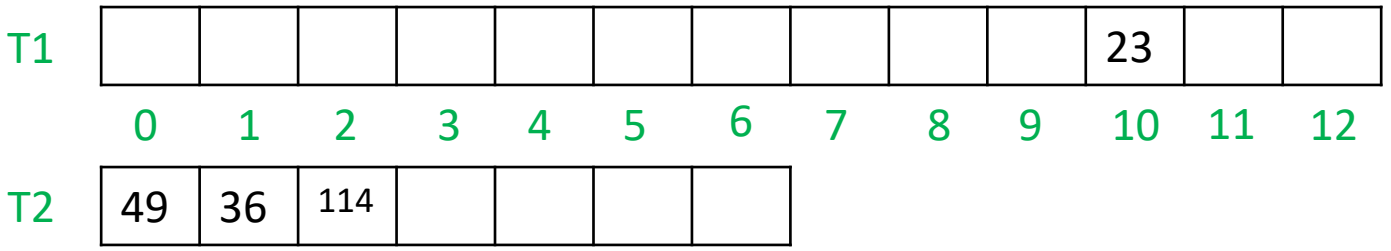
- Using the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



MONASH University

- Searching for key 36
  - Hash1(36) = 10, T1[10] is not 36

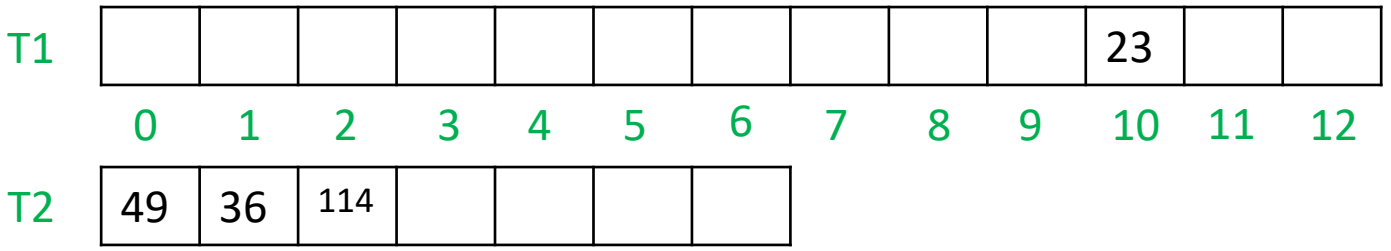
- Using the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

- Searching for key 36
  - Hash1(36) = 10, T1[10] is not 36
  - Hash2(36) = 1

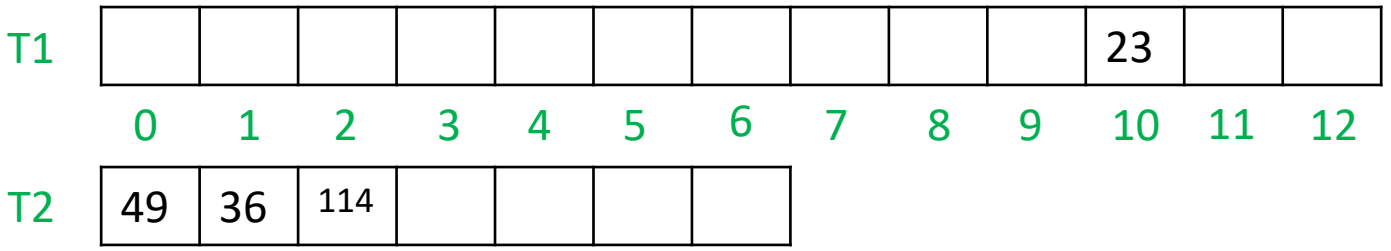
- Using the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



MONASH University

- Searching for key 36
  - Hash1(36) = 10, T1[10] is not 36
  - Hash2(36) = 1, T1[1] is 36… FOUND!

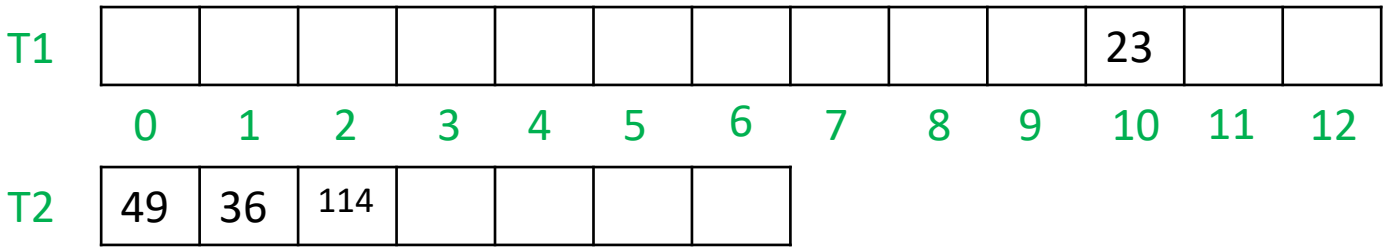
- Using the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



MONASH University

- Searching for key 10

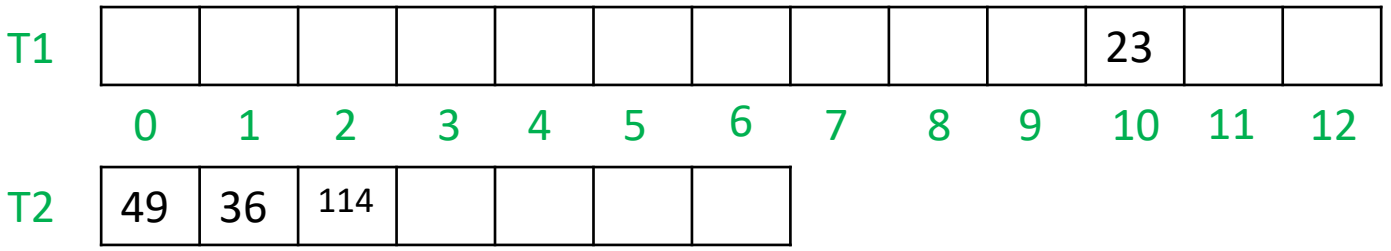
- Using the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

- Searching for key 10
  - Hash1(10) = 10

- Using the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7

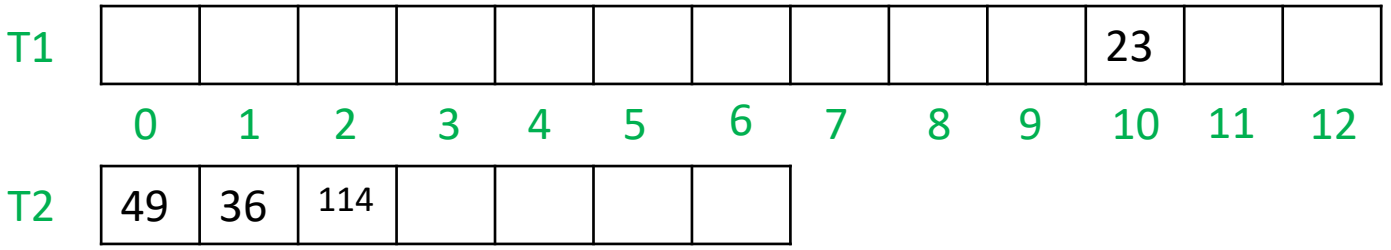


Hash Table
Cuckoo hashing

MONASH University

- Searching for key 10
  - Hash1(10) = 10, T1[10] is not 10

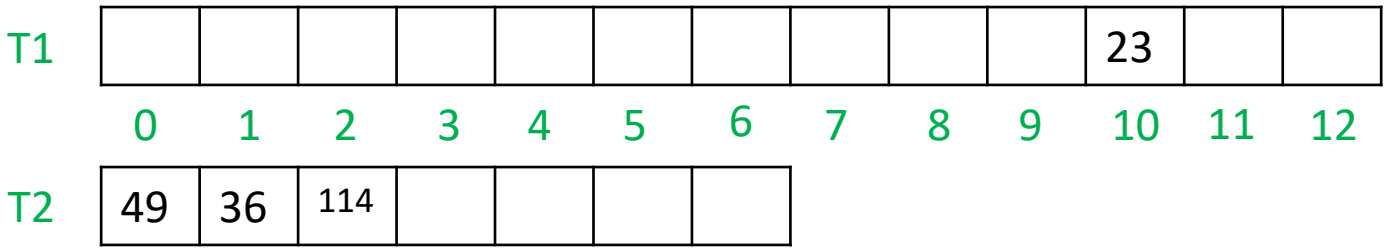
- Using the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

- Searching for key 10
  - Hash1(10) = 10, T1[10] is not 10
  - Hash2(10) = 3

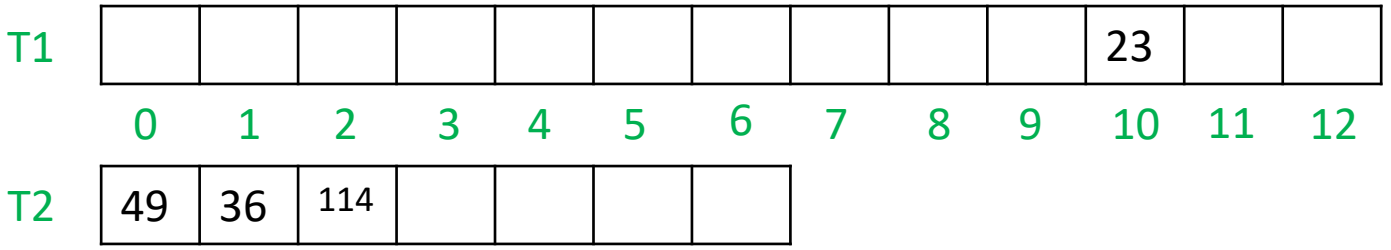
- Using the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

- Searching for key 10
  - Hash1(10) = 10, T1[10] is not 10
  - Hash2(10) = 3, T2[3] is not 10

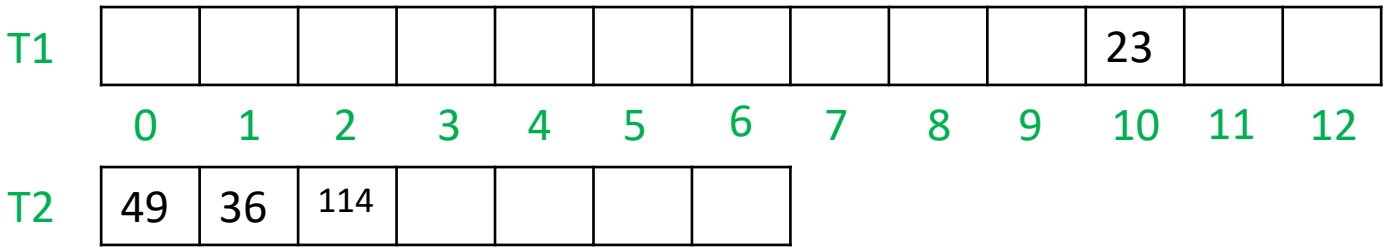
- Using the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



MONASH University

- Searching for key 10
  - Hash1(10) = 10, T1[10] is not 10
  - Hash2(10) = 3, T2[3] is not 10
  - So key 10 doesn't exist because it is not in T1 or T2

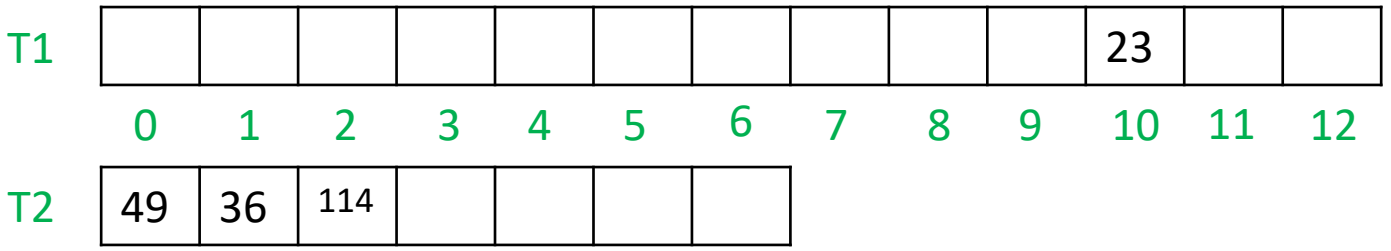
- Using the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7



Hash Table
Cuckoo hashing

- Guaranteed O(1) complexity!
  - For search

- Using the following tables…
  - Table 01, of size 13 . Hash1(key) = key % 13
  - Table 02, of size 7 . Hash2(key) = key % 7

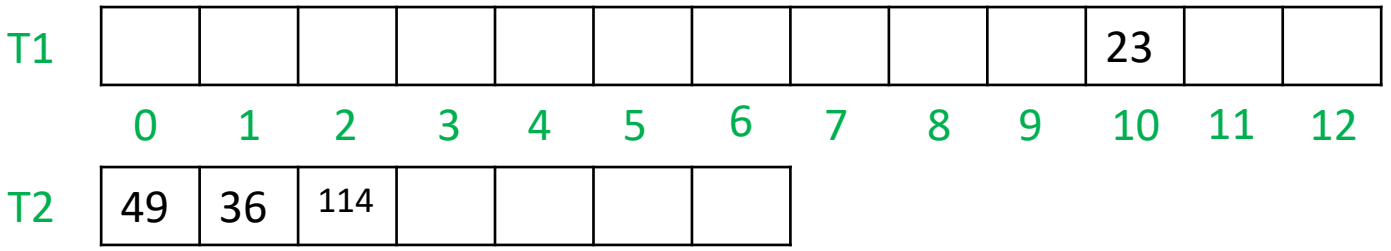


MONASH University

Questions?

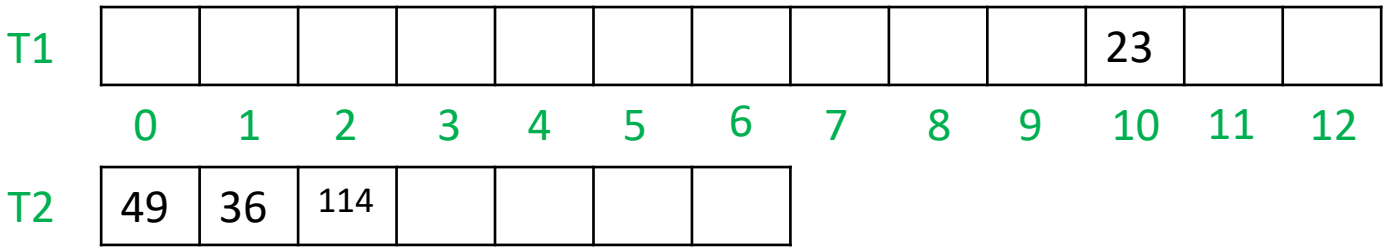
Hash Table
Cuckoo hashing

- What if we want to delete?

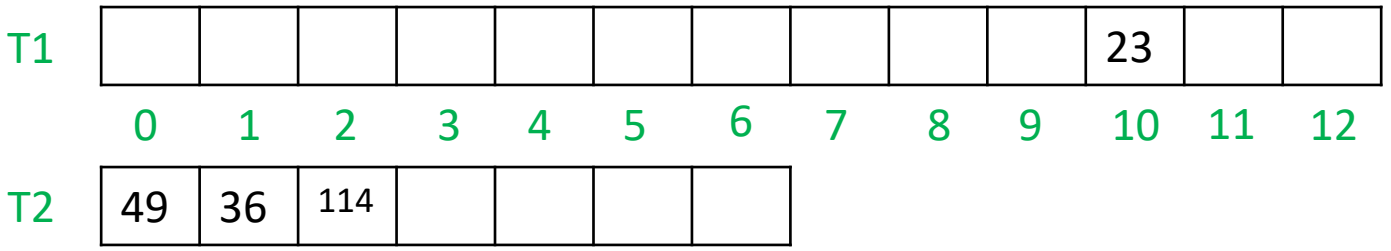


MONASH University

- What if we want to delete?
  - Same as search to look for key

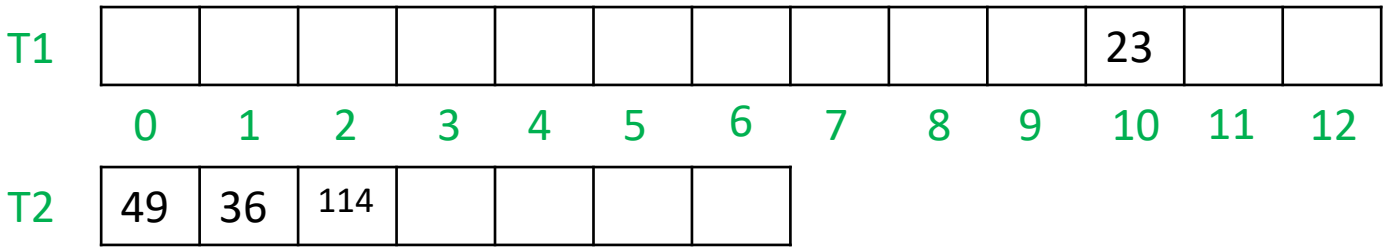


- What if we want to delete?
  - Same as search to look for key
  - If key is found, we delete



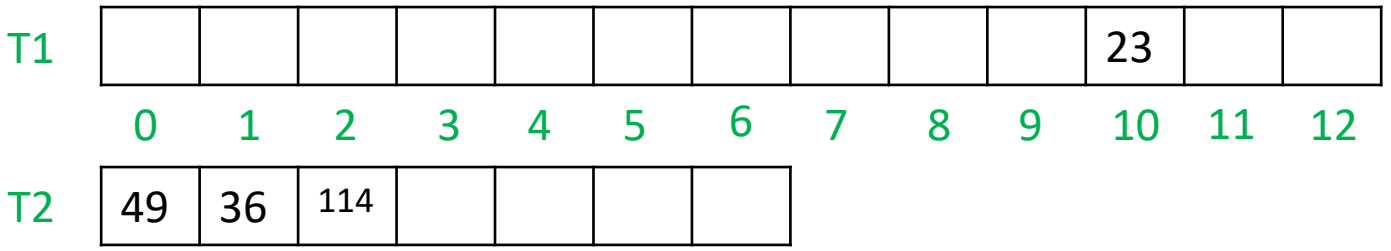
Hash Table
Cuckoo hashing

- What if we want to delete?
  - Same as search to look for key
  - If key is found, we delete
    - Removing the item but always search both tables



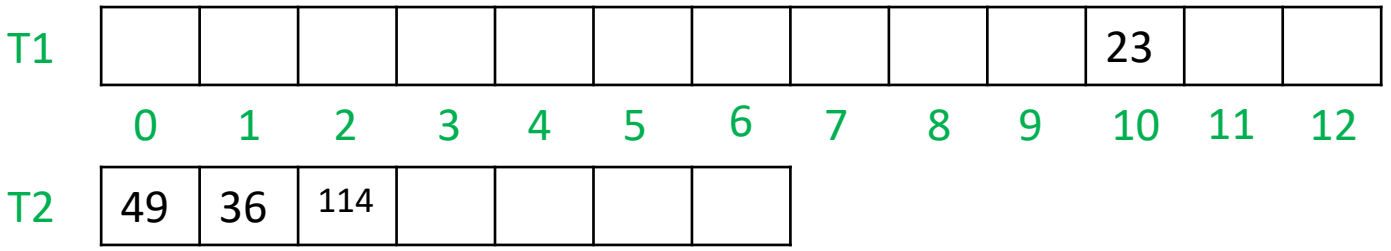
Hash Table
Cuckoo hashing

- What if we want to delete?
  - Same as search to look for key
  - If key is found, we delete
    - Removing the key but always search both tables
    - Replace key with FLAG and might not need to search both tables



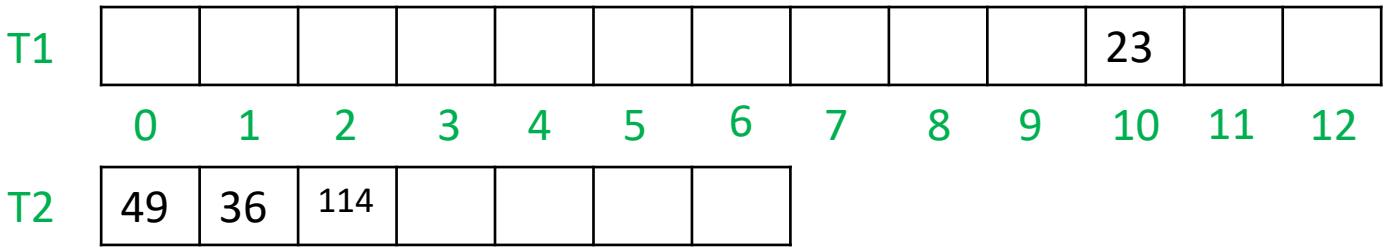
Hash Table
Cuckoo hashing

- What if we want to delete?
  - Same as search to look for key
  - If key is found, we delete
    - Removing the key but always search both tables
    - Replace key with FLAG and might not need to search both tables
    - Mainly just implementation decision



Hash Table
Cuckoo hashing

- What if we want to delete?
  - Same as search to look for key
  - If key is found, we delete
    - Removing the key but always search both tables
    - Replace key with FLAG and might not need to search both tables
    - Mainly just implementation decision



MONASH University

Questions?

MONASH University

- Now this is something new…
  - In double hashing, we use a 2nd hash function
  - For cuckoo hashing, we have a 2nd hash table!
  - Items are kicked between tables when there is collision
    - From 1st table to 2nd table
    - From 2nd table to 1st table

Hash Table
Cuckoo hashing

MONASH University

- Now this is something new…
  - In double hashing, we use a 2nd hash function
  - For cuckoo hashing, we have a 2nd hash table!
  - Items are kicked between tables when there is collision
    - From 1st table to 2nd table
    - From 2nd table to 1st table
  - Ideally, our complexity would be
    - O(1) for insert with 2 good hash functions (as to not resize often)
    - O(1) for search always!
    - O(1) for delete using flags

MONASH University

Questions?

Hash Table

Are great but…

- Hash tables are awesome…

Hash Table

Are great but…

- Hash tables are awesome… but…

Hash Table

Are great but…

- Hash tables are awesome… but…
  - Hard to produce good hash function
    - Even more perfect hash is not possible

Hash Table

Are great but…

MONASH University

- Hash tables are awesome… but…
  - Hard to produce good hash function
    - Even more perfect hash is not possible
  - Data are not ordered
    - So some operations are costly

Hash Table
Are great but…

MONASH University

- Hash tables are awesome… but…
  - Hard to produce good hash function
    - Even more perfect hash is not possible
  - Data are not ordered
    - So some operations are costly
    - What is max?
    - What is min?

Hash Table

Are great but…

MONASH University

- Hash tables are awesome… but…
  - Hard to produce good hash function
    - Even more perfect hash is not possible
  - Data are not ordered
    - So some operations are costly
    - What is max?
    - What is min?

- In reality, time is O(1)
  - But we use a lot of space
  - Sparse table for this to happen!

Hash Table
Are great but…

MONASH University

- Hash tables are awesome… but…
  - Hard to produce good hash function
    - Even more perfect hash is not possible
  - Data are not ordered
    - So some operations are costly
    - What is max?
    - What is min?

- In reality, time is O(1)
  - But we use a lot of space
  - Sparse table for this to happen!
    - Resize when load factor is reached
    - This resize cost of O(N*loadfactor) is added to the insert cost; but armotized to O(1)… Why?

Complexity of resizing - intuition

- Imagine spreading out the resize work over the insertions
- This concept is called "amortized analysis" (not examinable)

Table size	Total work for insertion	Total work for resize
m	m/2	2m + m/2
2m	m	4m + m
4m	2m	8m + 2m
…	…	…
$2^i m$	$2^{i-1} m$	$2^{i+1} m + (2^{i-1} m)$

- The amortized cost of each insert is O(1), even though most of the work occurs on one specific insert (the one which triggers a resize)

Hash Table

Are great but…

MONASH University

- Hash tables are awesome… but…
  - Hard to produce good hash function
    - Even more perfect hash is not possible
  - Data are not ordered
    - So some operations are costly
    - What is max?
    - What is min?

- In reality, time is O(1)
  - But we use a lot of space
  - Sparse table for this to happen!
    - Resize when load factor is reached
    - This resize cost of O(N*loadfactor) is added to the insert cost; but armotized to O(1)… Why? You don't resize for every insertion!

Hash Table

Are great but…

- Amortized analogy

Hash Table

Are great but…

MONASH University

- Amortized analogy
  - Everyday eat normal food $10
  - Everyday eat instant noodle $1, then once a week eat fancy $50

Hash Table

Are great but…

MONASH University

- Amortized analogy
  - Everyday eat normal food $10
    - Total in a week = $70
  - Everyday eat instant noodle $1, then once a week eat fancy $50
    - Total in a week = $56

Hash Table

Are great but…

MONASH University

- Amortized analogy
  - Everyday eat normal food $10
    - Total in a week = $70
  - Everyday eat instant noodle $1, then once a week eat fancy $50
    - Total in a week = $56
  - Basically it is the sum of complexity over a series of operations…
    - Here over an insertion of N items

Hash Table

Are great but…

MONASH University

- Amortized analogy
  - Everyday eat normal food $10
    - Total in a week = $70
  - Everyday eat instant noodle $1, then once a week eat fancy $50
    - Total in a week = $56
  - Basically it is the sum of complexity over a series of operations…
    - Here over an insertion of N items
    - We can add the first-k items in O(1)
    - Then the k+1-th item we resize with O(k)

Hash Table
Are great but…

MONASH University

- Amortized analogy
  - Everyday eat normal food $10
    - Total in a week = $70
  - Everyday eat instant noodle $1, then once a week eat fancy $50
    - Total in a week = $56
  - Basically it is the sum of complexity over a series of operations…
    - Here over an insertion of N items
    - We can add the first-k items in O(1)
    - Then the k+1-th item we resize with O(k)
    - Then after the k+1-th items, cause table is bigger we have less collision and insertion back to O(1)

Hash Table
Are great but…

MONASH University

- Amortized analogy
  - Everyday eat normal food $10
    - Total in a week = $70
  - Everyday eat instant noodle $1, then once a week eat fancy $50
    - Total in a week = $56
  - Basically it is the sum of complexity over a series of operations…
    - Here over an insertion of N items
    - We can add the first-k items in O(1)
    - Then the k+1-th item we resize with O(k)
    - Then after the k+1-th items, cause table is bigger we have less collision and insertion back to O(1)
    - Rinse and Repeat!

MONASH University

Questions?

MONASH University

Thank You