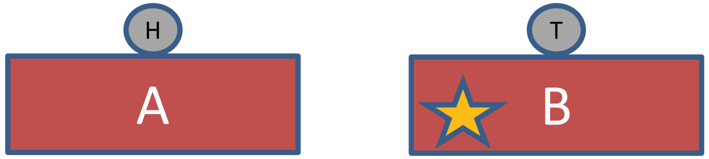
CS2040S

Data Structures and Algorithms

Hashing! (Part 1)



Puzzle of the Week:

You and your friend are a team.

I take you to a room with two boxes labeled ‘A’ and ‘B’. You see me putting a prize inside one of the boxes and then closing it. I put a coin on top of each box; each coin can be showing head or tail.

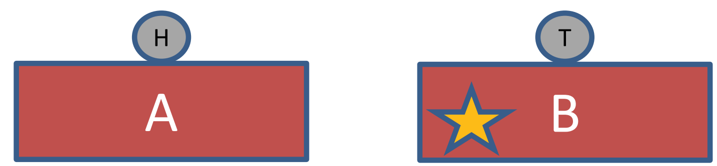
I ask you to flip exactly one of the two coins.

You then go out of the room, and your friend comes in. If she can guess the box with the prize, your team wins the prize. Otherwise, your team loses.

How should you choose the coin to flip? (You and your friend can strategize in advance.)

CS2040S
Data Structures and Algorithms

Hashing! (Part 1)



Puzzle of the Week:

You and your friend are a team.

I take you to a room with two boxes labeled ‘A’ and ‘B’. You see me putting a prize inside one of the boxes and then closing it. I put a coin on top of each box; each coin can be showing head or tail.

I ask you to flip exactly one of the two coins

You then go out of the room, and your f … guess the box with the prize, your team wins the prize. Otherwise, your team loses.

How should you choose the coin to flip?

Too easy? Solve this problem for 4 boxes!

Borrowed from https://www.theguardian.com/science/2023/jan/23/can-you-solve-it-prisoners-and-boxes

Plan: Rest of Semester

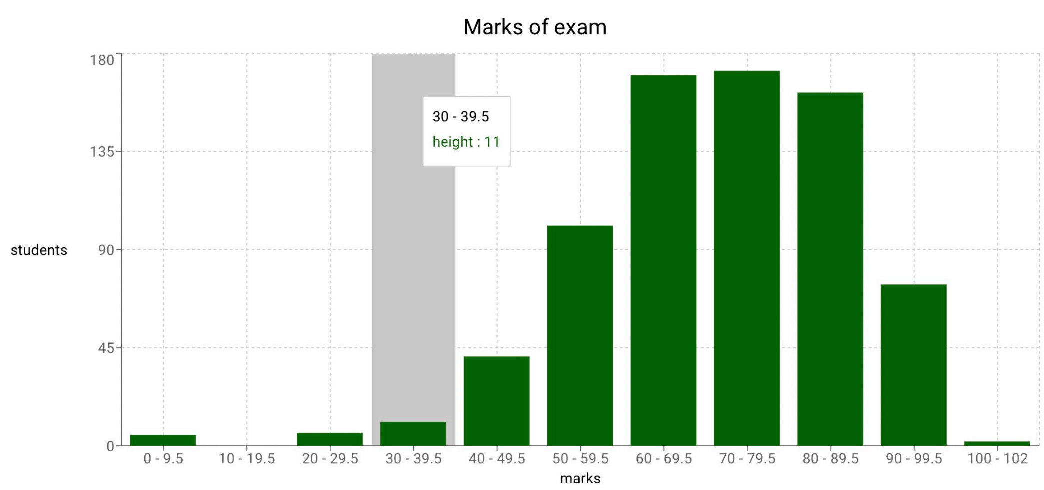
- Arnab (arnabb@nus.edu.sg) takes over the lectures.

- Both Seth and I are here to help you with the course!

Midterm

- Will release scores and solutions next week after make-up exam taken.

- Preliminary grading: Mean ≈ 71, Median ≈ 72.



Plan: this week and next

Three (or Four) Days of Hashing

- Applications
- Basic theory
- Handling collisions
- (Hashing in Java)
- Amortized analysis (doubling/shrinking)
- Sets and Bloom filters

Abstract Data Types

Symbol Table

```
public interface  SymbolTable

         void   insert(Key k, Value v)   insert (k,v) into table
        Value   search(Key k)            get value paired with k
         void   delete(Key k)            remove key k (and value)
      boolean   contains(Key k)          is there a value for k?
          int   size()                   number of (k,v) pairs
```

Note: no successor / predecessor queries.

Symbol Table

Examples:

Dictionary:	key = word
	value = definition
Phone Book	key = name
	value = phone number
Internet DNS	key = website URL
	value = IP address
Java compiler	key = variable name
	value = type and value

Implement symbol table with an AVL tree:
(C_I= cost insert, C_S=cost search)

1. $C_I = O(1)$, $C_S = O(1)$
2. $C_I = O(1)$, $C_S = O(\log n)$
3. $C_I = O(1)$, $C_S = O(n)$
4. $C_I = O(\log n)$, $C_S = O(\log n)$
5. $C_I = O(n)$, $C_S = O(\log n)$
6. $C_I = O(n)$, $C_S = O(n)$

ARCHIPELAGO
is open

Symbol Table

Implement a symbol table with:

- $C_I = O(1)$
- $C_S = O(1)$

Fast, fast, fast….

Dictionaries vs. Symbol Tables

What can you do with a dictionary but not a symbol table?

Dictionaries vs. Symbol Tables

Sorting with a dictionary:

1) Insert every item into the dictionary.
2) Search for the minimum item.
3) Repeat: find successor

ARCHIPELAGO
is open

Running time to implement sorting:

With an AVL tree/dictionary?

Dictionaries vs. Symbol Tables

Sorting with a dictionary:

1) Insert every item into the dictionary.

2) Search for the minimum item.

3) Repeat: find successor

Running time to implement sorting:

With an AVL tree/dictionary? O(n log n)

With a symbol table?

Dictionaries vs. Symbol Tables

Sorting with a dictionary:

1) Insert every item into the dictionary.

2) Search for the minimum item.

3) Repeat: find successor

Running time to implement sorting:

With an AVL tree/dictionary? $O(n \log n)$

With a symbol table? $O(n^2)$

- No efficient way to find minimum item!
- No ordering of elements.

Sorting (aside)

Isn't O(1) search/insert impossible? (Binary) search takes $\Omega(\log n)$ comparisons.

- Impossible to search in fewer than $\log(n)$ comparisons.
- But a symbol table finds an item in O(1) steps!!
- Conclusion: symbol table is not ***comparison-based***.

Building a Symbol Table

Direct Access Tables

Attempt #1: Use a table, indexed by keys.

0	null
1	null
2	item1
3	null
4	null
5	item3
6	null
7	null
8	item2
9	null

Universe U={0..9} of size m = 10.

(key, value)

(2, item1)
(8, item2)
(5, item3)

Assume keys are distinct.

Direct Access Tables

Attempt #1: Use a table, indexed by keys.

0	null
1	null
2	item1
3	null
4	null
5	item3
6	null
7	null
8	item2
9	null

Example: insert(4, Seth)

Direct Access Tables

Attempt #1: Use a table, indexed by keys.

0	null
1	null
2	item1
3	null
4	Seth
5	item3
6	null
7	null
8	item2
9	null

Example: insert(4, Seth)

Time: O(1) / insert, O(1) / search

Direct Access Tables

Problems:

- What if keys are not integers?
  - Where do you put the key/value “**(hippopotamus, bob)**”?
  - Where do you put 3.14159…?

Direct Access Tables

Pythagoras said, “Everything is a number.”

“The School of Athens” by Raphael

Direct Access Tables

Pythagoras said, “Everything is a number.”

- Everything is just a sequence of bits.
- Treat those bits as a number.

- English:
  - 26 letters => 5 bits/letter
  - Longest word = 34 letters (Supercalifragilisticexpialidocious?)
  - 34 letters * 5 bits = 170 bits
  - So we can store any English word in a direct-access array of size 2^{170}.

Direct Access Tables

Pythagoras said, "Everything is a number."

- Everything is just a sequence of bits.
- Treat those bits as a number.

- English:
  - 26 letters => 5 bits/letter
  - Longest word = 34 letters (Supercalifragilisticexpialidocious?)
  - 34 letters * 5 bits = 170 bits
  - So we can store any English word in a direct-access array of size 2^{170}. ≈ number of atoms in observable universe

Direct Access Tables

Problems:

- What if keys are not integers?
  - Where do you put the key/value "**(hippopotamus, bob)**"?
  - Where do you put 3.14159…?

  ➔ Can represent anything as a sequence of bits.

- Too much space
  - If keys are integers, then table-size > 4 billion

  ➔ Hashing

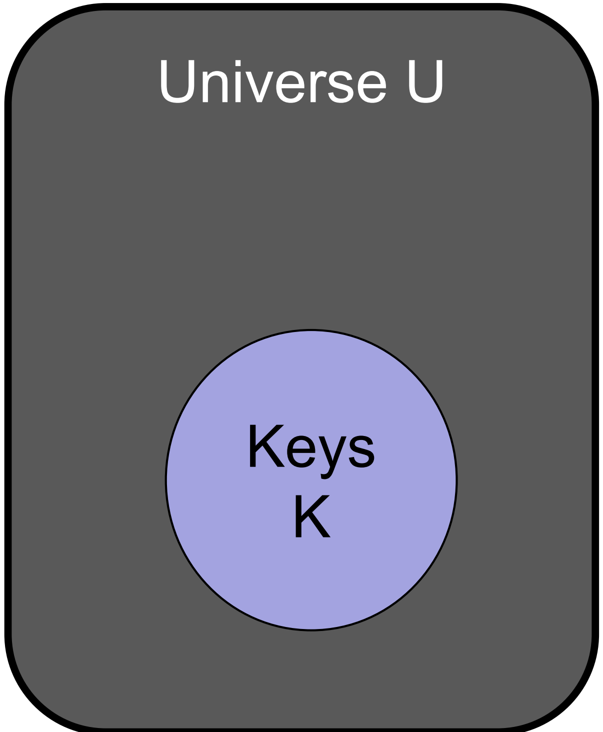
Hash Functions

Problem:

– Huge universe U of possible keys.

– Smaller number n of actual keys.

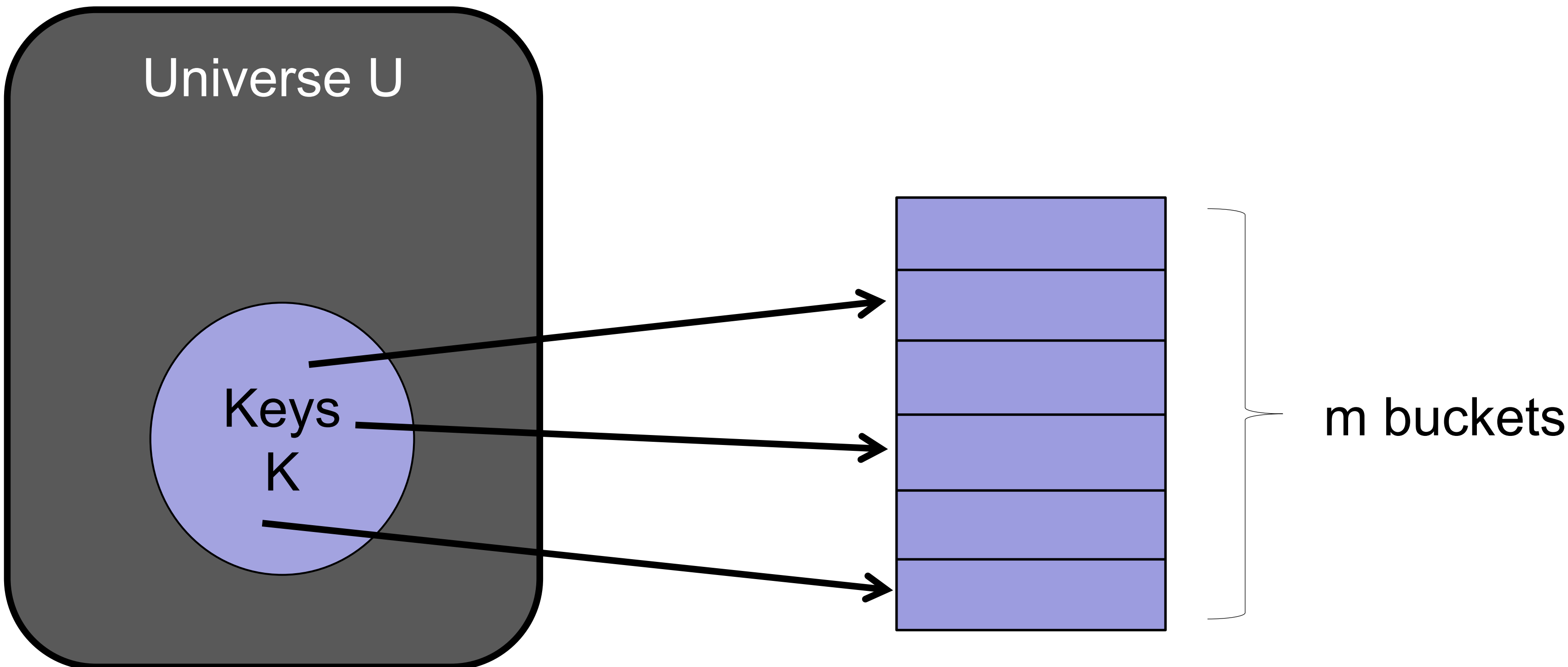
e.g., 2^{170}



Hash Functions

Problem:

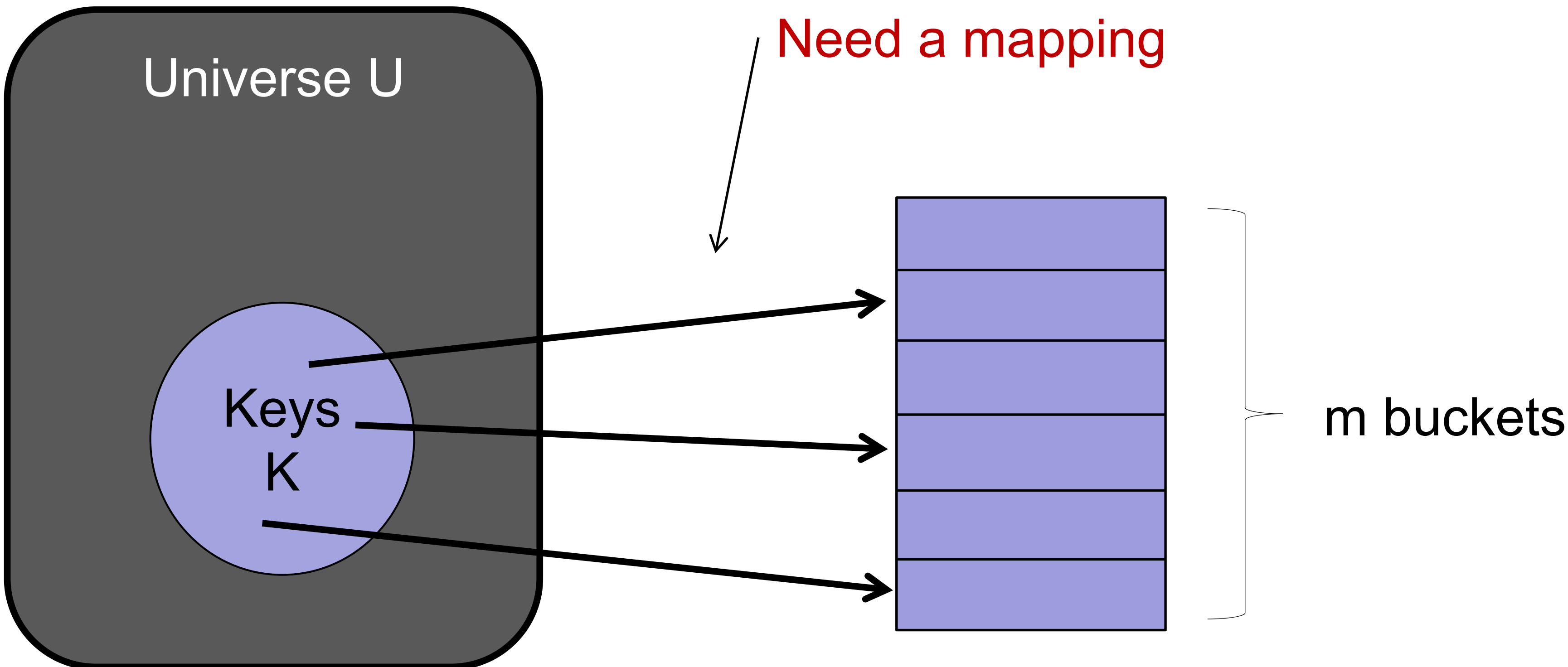
- Huge universe U of possible keys.
- Smaller number n of actual keys.
- How to map n keys to $m \approx n$ buckets?



Hash Functions

Problem:

- Huge universe U of possible keys.
- Smaller number n of actual keys.
- How to map n keys to $m \approx n$ buckets?

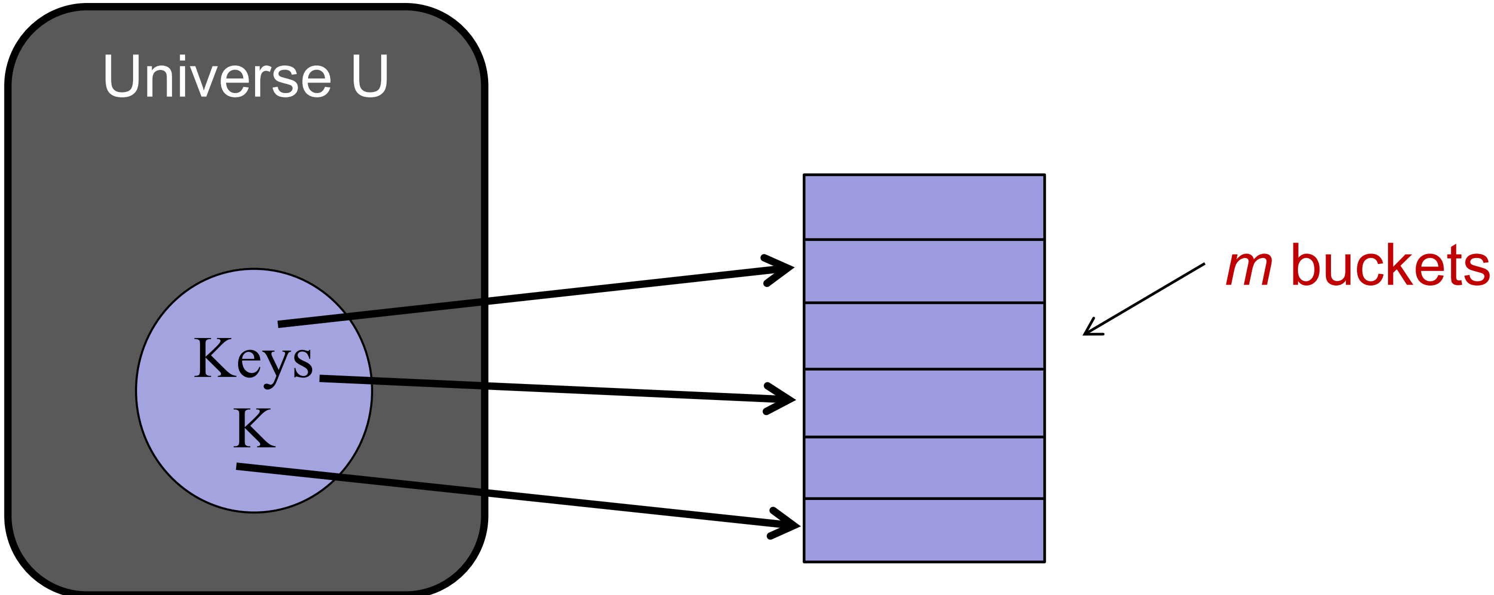


Hash Functions

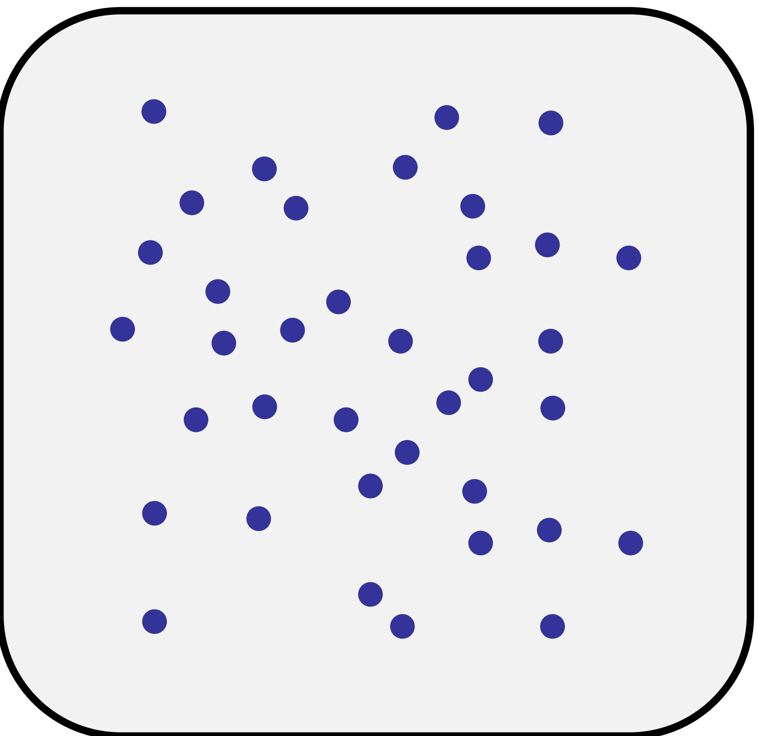
Define hash function $h : U \rightarrow \{1..m\}$

- Store key k in bucket $h(k)$.
- Time complexity:
  - Time to compute h + Time to access bucket
- Usually: assume hash function has cost 1 to compute.

unless otherwise specified, e.g., long strings.



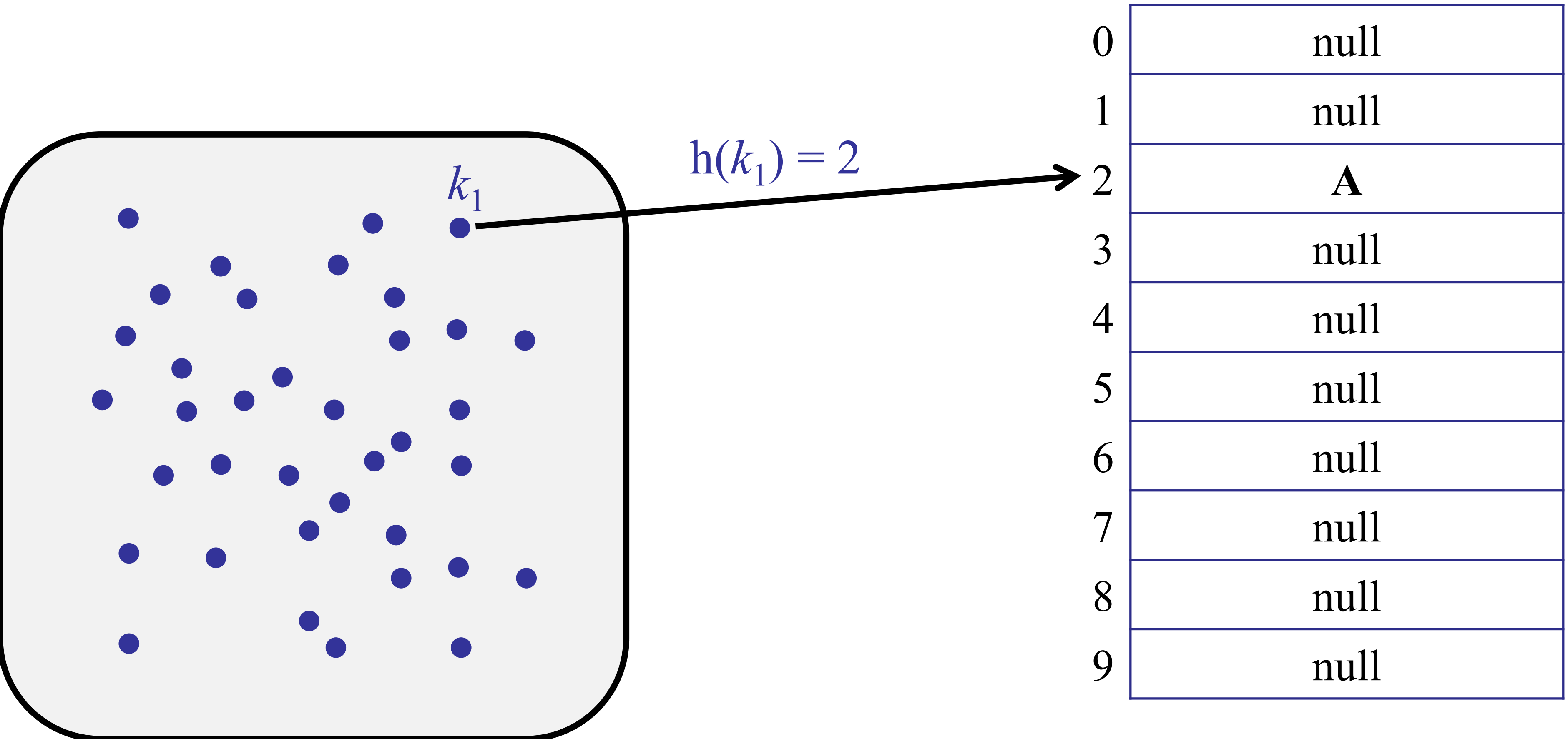
Hash Functions



0	null
1	null
2	null
3	null
4	null
5	null
6	null
7	null
8	null
9	null

Hash Functions

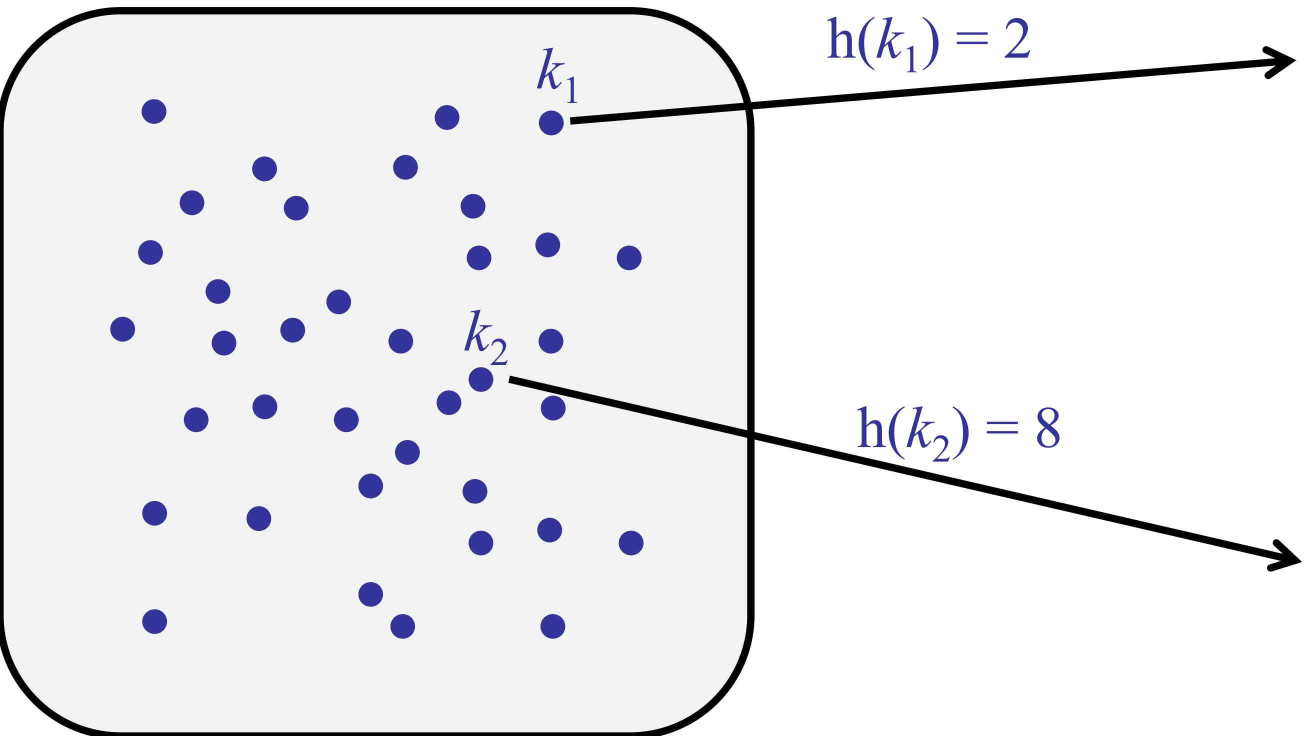
insert(k_1, A)



Hash Functions

insert(k_1, A)

insert(k_2, B)



0	null
1	null
2	**A**
3	null
4	null
5	null
6	null
7	null
8	**B**
9	null

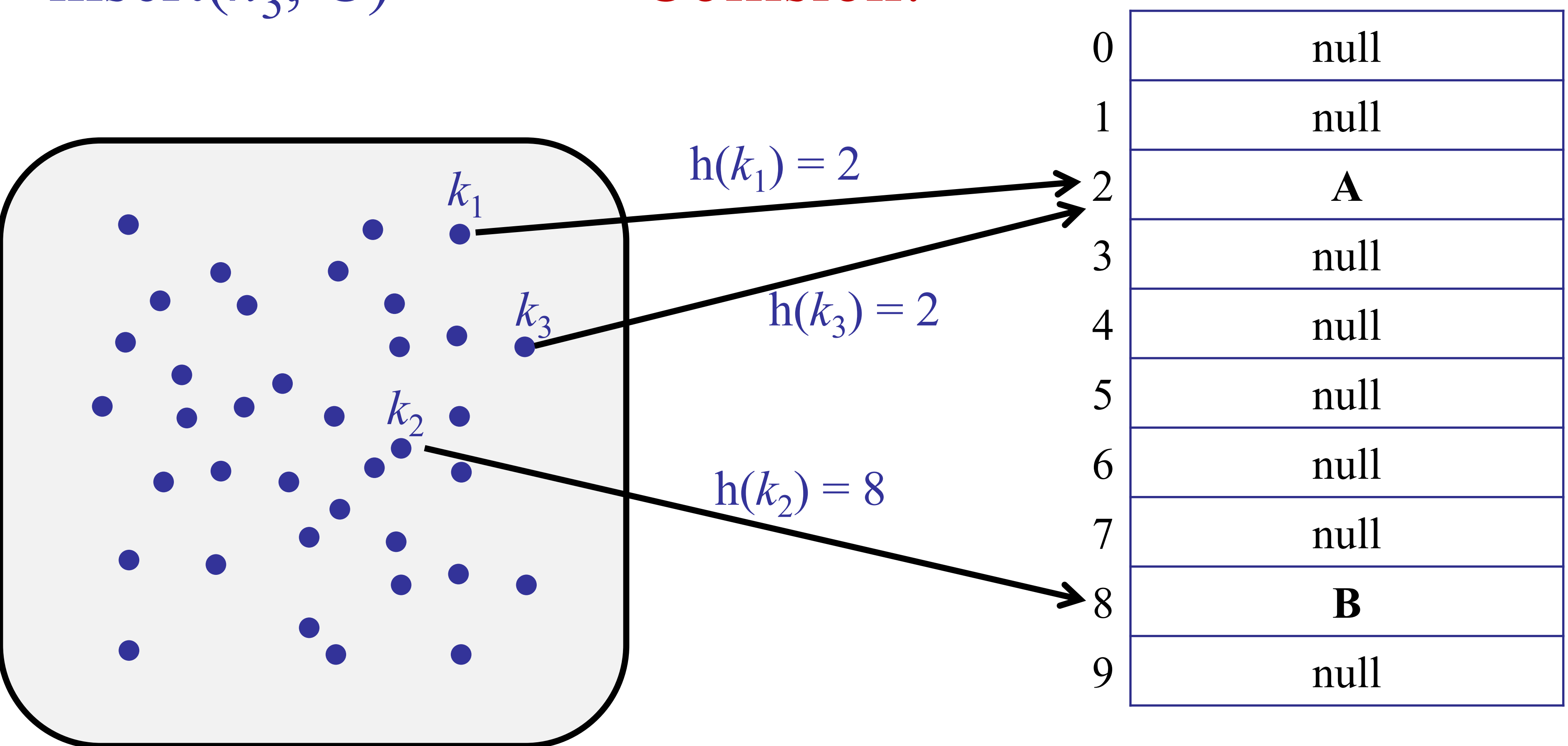
Hash Functions

insert(k_1, A)

insert(k_2, B)

insert(k_3, C)

Collision!



Hash Functions

Collisions:

- We say that two distinct keys k_1 and k_2 collide if:

$$h(k_1) = h(k_2)$$

Can we choose a hash function with no collisions?

1. Yes
2. Sometimes, if we choose carefully
3. No, impossible

ARCHIPELAGO

is open

Hash Functions

Collisions:

- We say that two <u>distinct</u> keys k_1 and k_2 collide if:

$$h(k_1) = h(k_2)$$

- Unavoidable!
  - The table size is smaller than the universe size.
  - The pigeonhole principle says:
    - There must exist two keys that map to the same bucket.
    - Some keys must collide!

Coping with Collision

Idea: choose a new, better hash functions

Coping with Collision

Idea: choose a new, better hash functions

- Hard to find.
- Requires re-copying the table.
- Eventually, there will be another collision.

Coping with Collision

Idea: choose a new, better hash functions

- Hard to find.
- Requires re-copying the table.
- Eventually, there will be another collision.

Idea: chaining (today)

- Put both items in the same bucket!

Idea: open addressing (next week)

- Find another bucket for the new item.

Coping with Collision

Idea: choose a new, better hash functions

- Hard to find.
- Requires re-copying the table.
- Eventually, there will be another collision.

Idea: chaining (today)

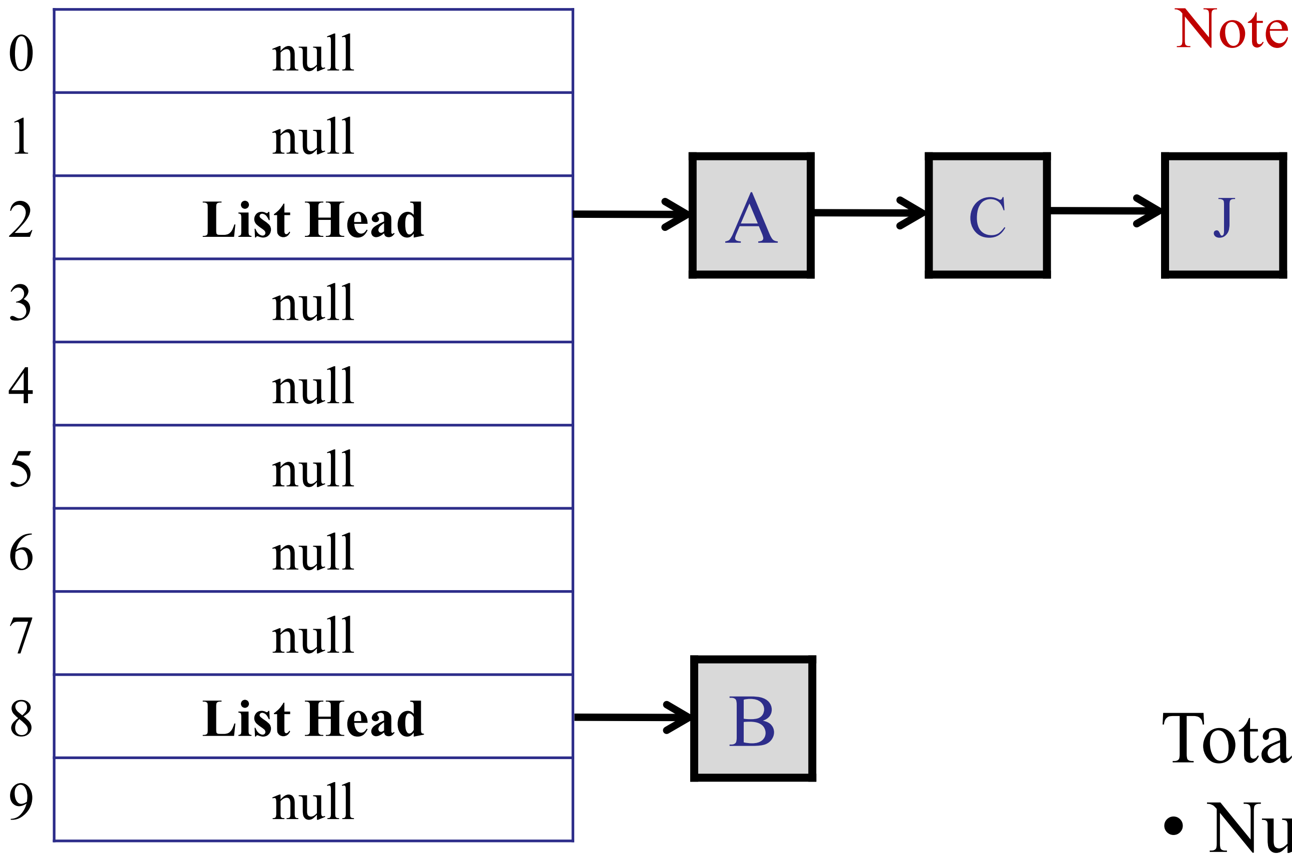
- Put both items in the same bucket!

Idea: open addressing (next week)

- Find another bucket for the new item.

Chaining

Each bucket contains a linked list of items.



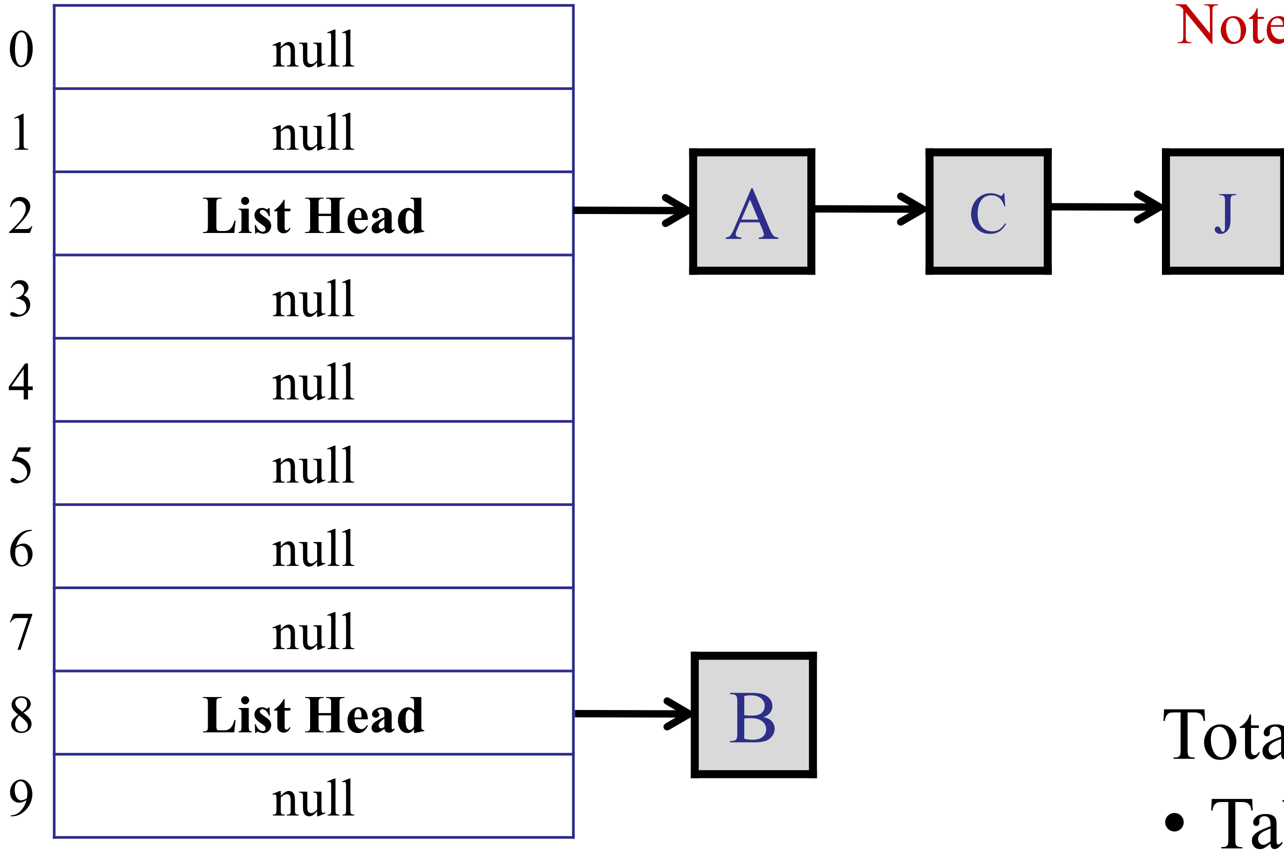
Note: h(A) == h(C) == h(J)

Total space:

- Number buckets: m
- Number entries: n

Chaining

Each bucket contains a linked list of items.



Note: h(A) == h(C) == h(J)

Total space: $O(m + n)$
- Table size: m
- Linked list size: n

Hashing with Chaining

Operations:

- insert(key, value)
  - Calculate h(key)
  - Lookup h(key) and add (key,value) to the linked list.

- search(key)
  - Calculate h(key)
  - Search for (key,value) in the linked list.

What is the worst-case cost of inserting a (key, value)? Assume cost(h) is cost of computing the hash function.

1. $O(1 + \text{cost}(h))$
2. $O(\log n + \text{cost}(h))$
3. $O(n + \text{cost}(h))$
4. $O(n\ \text{cost}(h))$
5. $O(n^2)$.

ARCHIPELAGO

is open

Do we care about duplicates?

➔ If so, the cost of insert is higher because we need to search for duplicates.

Hashing with Chaining

Operations:

Always O(1).

– insert(key, value)
  - Calculate h(key)
  - Lookup h(key) and add (key,value) to the linked list.

  (Note: this allows duplicate keys. Need to specify more precisely the behavior or insert!)

– search(key)
  - Calculate h(key)
  - Search for (key,value) in the linked list.

What is the worst-case cost of searching a (key, value)?

1. O(1 + cost(h))
2. O(log n + cost(h))
3. O(n + cost(h))
4. O(n*cost(h))
5. We cannot determine it without knowing h.

ARCHIPELAGO is open

Hashing with Chaining

Operations:

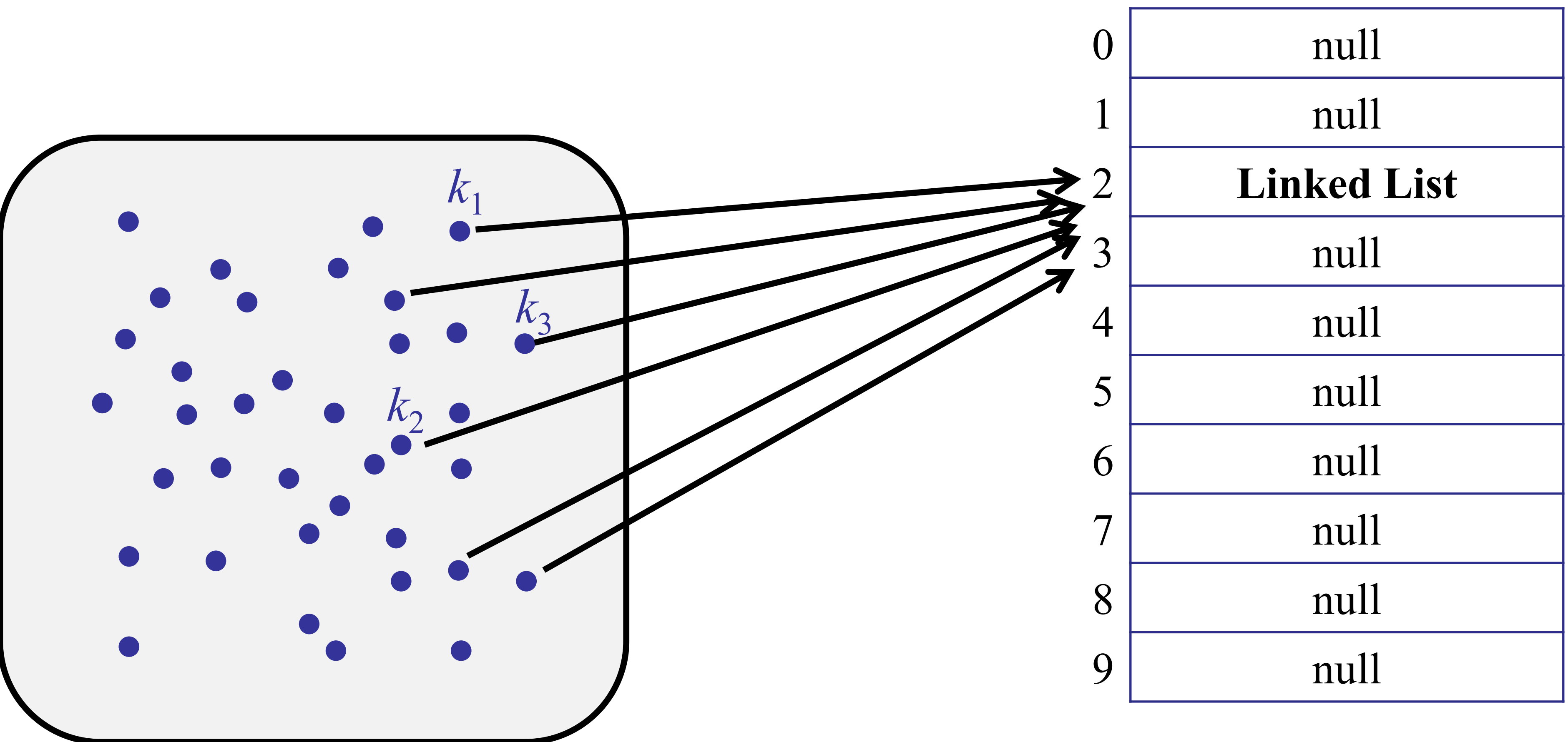
- insert(key, value)
  - Calculate h(key)
  - Lookup h(key) and add (key,value) to the linked list.

- search(key) → time depends on length of linked list
  - Calculate h(key)
  - Search for (key,value) in the linked list.

Hashing with Chaining

Assume all keys hash to the same bucket!

- Search costs O(n)
- Oh no!



Let's be optimistic today.

The Simple Uniform Hashing Assumption

- Every key is equally likely to map to every bucket.
- Keys are mapped independently.

Intuition:

- Each key is put in a random bucket.
- Then, as long as there are enough buckets, we won't get too many keys in any one bucket.

Why don’t we just insert each key into a random bucket (instead of using a hash function h)?

ARCHIPELAGO

is open

Why don't we just insert each key into a random bucket (instead of using hash function h)?

1. It would be slow to insert.
2. Computers don't have a real source of randomness.
3. By choosing the keys carefully, a user could force the random choices to create many collisions.
4. Searching would be very slow.
5. None of the above.

Hashing with Chaining

Searching:

- **Expected** search time = 1 + n/m = O(1) [Next lecture!]
- Worst-case search time = O(n)

Inserting:

- Worst-case insertion time = O(1)

Hashing: Recap

Problem: coping with large universe of keys

- Number of possible keys is very, very large.
- Direct Access Table takes too much space

Hash functions

- Use hash function to map keys to buckets.
- Sometimes, keys collide (inevitably!)
- Use linked list to store multiple keys in one bucket.

Analyze performance with simple uniform hashing.

- Expected number of keys / bucket is $O(n/m) = O(1)$.

Summary

Symbol Tables are pervasive

- You find them everywhere!

Hash tables are fast, efficient symbol tables.

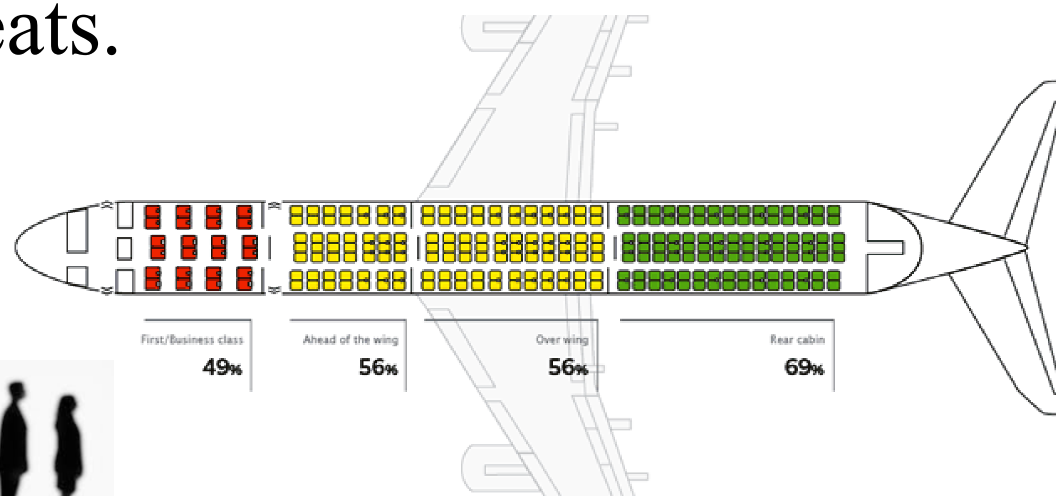
- Under optimistic assumptions, provably so.
- In the real world, often so.
- But be careful!

Beats BSTs:

- Operate directly on keys (i.e., indexing)
- Gave up: successor/predecessor/etc.

Puzzle Break

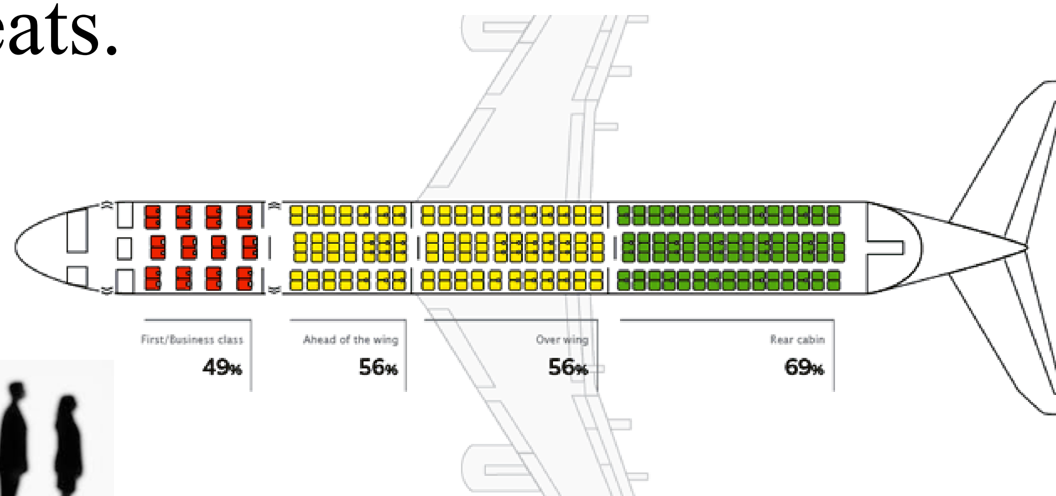
An airplane has 100 seats.



100 passengers board the airplane in a random order.

Puzzle Break

An airplane has 100 seats.



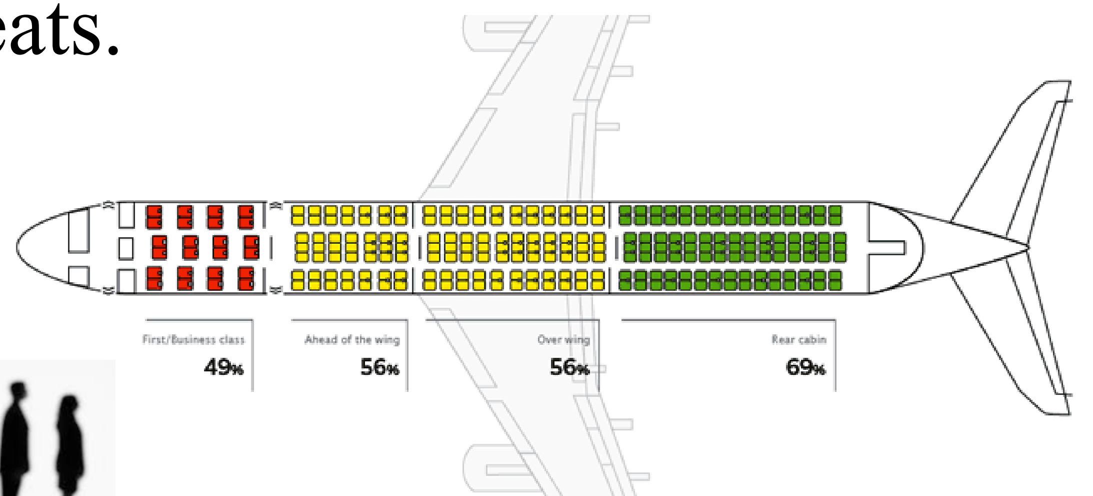
100 passengers board the airplane in a random order.

Passenger 1 is Mr. Burns.

Mr. Burns sits in a random seat.

Puzzle Break

An airplane has 100 seats.



Every other passenger:

- Sits in their assigned seat, if it is free.
- Otherwise, sits in a random seat.

Puzzle Break

An airplane has 100 seats.

You are passenger #100.

What is the probability your seat is free when you board?

Puzzle Break

An airplane has 100 seats.

What is the probability your seat is free when you board?

Problem Solving techniques:

Try a plane with 2 seats. Try a plane with 3 seats.