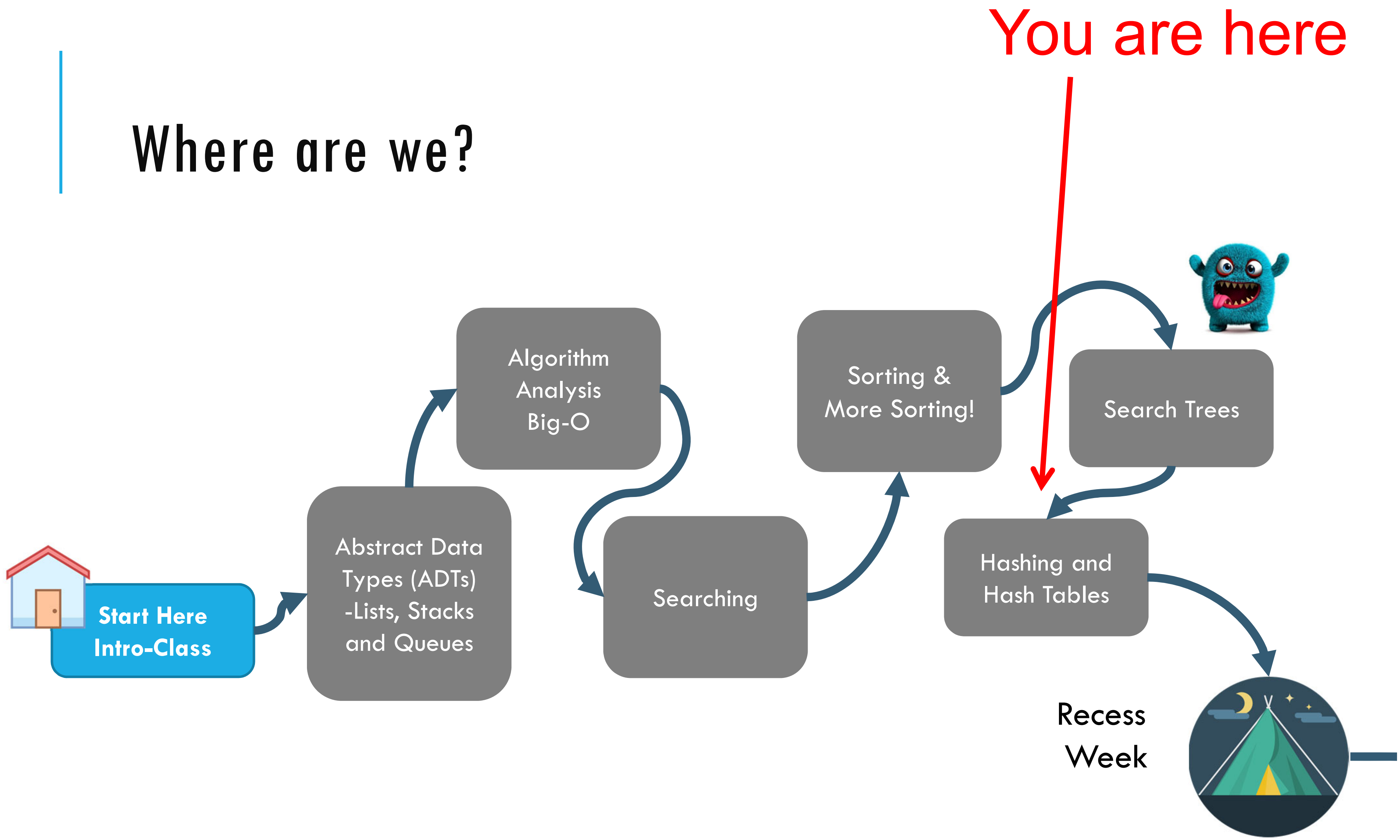
CS2040S
Data Structures and Algorithms

Hashing!

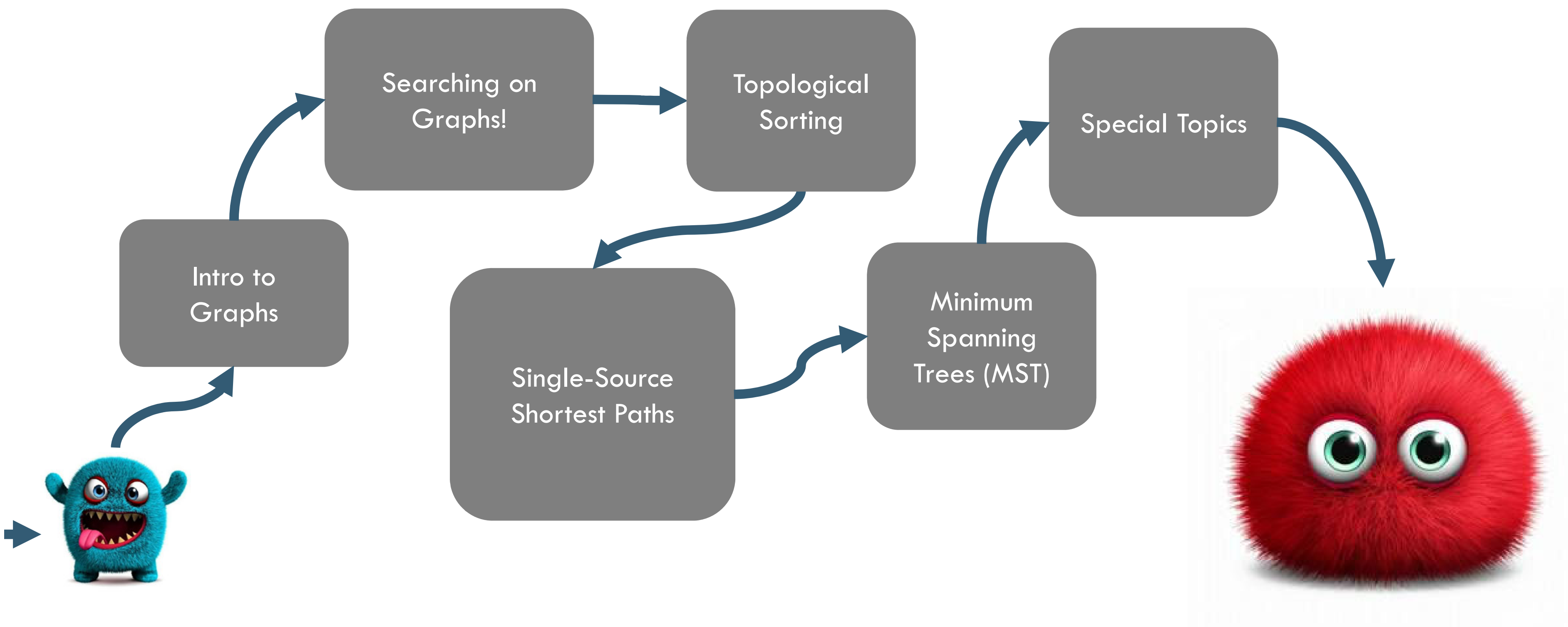
(Part 1)

Where are we?



Part I: Organizing your Data

Where are we?



Part II: Modelling and Solving Problems

Announcements

Midterm : Tuesday March 9, 4pm

(Note: In person, face-to-face)

Will post last year’s midterm next week…

Will discuss how to prepare after recess week…

(Hint: practice!)

Plan: this week and next

Three Days of Hashing

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

Topic of the Week: Hash Tables

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. $C_I = O(n)$, $C_S = O(n)$

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

Running time to implement sorting:

With an AVL tree/dictionary?

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?

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? $O(n^2)$

Sorting (aside)

Isn't O(1) search/insert impossible?

- Sorting takes $\Omega(n \log n)$ comparisons.

Sorting (aside)

Isn't O(1) search/insert impossible?

Sorting takes $\Omega(n \log n)$ comparisons.

- How do you sort with a symbol table?
- Only search/insert/delete.

Sorting (aside)

Isn't O(1) search/insert impossible?

Sorting takes $\Omega(n \log n)$ comparisons.

- How do you sort with a symbol table?
- Only search/insert/delete.

(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***.

Symbol Tables in Java

Symbol Tables in Java

java.util.Map

public interface java.util.Map<Key, Value>

void	clear()	*removes all entries*
boolean	containsKey(Object k)	*is k in the map?*
boolean	containsValue(Object v)	*is v in the map?*
Value	get(Object k)	*get value for k*
Value	put(Key k, Value v)	*adds (k,v) to table*
Value	remove(Object k)	*remove mapping for k*
int	size()	*number of entries*

Note: no successor / predecessor queries.

Symbol Tables in Java

java.util.Map

- Parameterized by key and value.
- Not necessarily comparable

`public interface java.util.Map<Key, Value>`

`void`	`clear()`	*removes all entries*
`boolean`	`containsKey(Object k)`	*is k in the map?*
`boolean`	`containsValue(Object v)`	*is v in the map?*
`Value`	`get(Object k)`	*get value for k*
`Value`	`put(Key k, Value v)`	*adds (k,v) to table*
`Value`	`remove(Object k)`	*remove mapping for k*
`int`	`size()`	*number of entries*

Note: no successor / predecessor queries.

Symbol Tables in Java

- Search by key.

java.util.Map

public interface	java.util.Map<Key, Value>	
void	clear()	*removes all entries*
boolean	containsKey(Object k)	*is k in the map?*
boolean	containsValue(Object v)	*is v in the map?*
Value	get(Object k)	*get value for k*
Value	put(Key k, Value v)	*adds (k,v) to table*
Value	remove(Object k)	*remove mapping for k*
int	size()	*number of entries*

Note: no successor / predecessor queries.

Symbol Tables in Java

java.util.Map

- Search by key.
- Search by value.

(May not be efficient.)

```
public interface  java.util.Map<Key, Value>

      void  clear()                      removes all entries
   boolean  containsKey(Object k)        is k in the map?
   boolean  containsValue(Object v)      is v in the map?
     Value  get(Object k)                get value for k
     Value  put(Key k, Value v)          adds (k,v) to table
     Value  remove(Object k)             remove mapping for k
       int  size()                       number of entries
```

Note: no successor / predecessor queries.

Symbol Tables in Java

java.util.Map

• Can use any Object as key?

```
public interface  java.util.Map<Key, Value>
```

void	clear()	*removes all entries*
boolean	containsKey(Object k)	*is k in the map?*
boolean	containsValue(Object v)	*is v in the map?*
Value	get(Object k)	*get value for k*
Value	put(Key k, Value v)	*adds (k,v) to table*
Value	remove(Object k)	*remove mapping for k*
int	size()	*number of entries*

Note: no successor / predecessor queries.

Symbol Tables in Java

java.util.Map

• Put new (key, value) in table.

`public interface java.util.Map<Key, Value>`

`void`	`clear()`	*removes all entries*
`boolean`	`containsKey(Object k)`	*is k in the map?*
`boolean`	`containsValue(Object v)`	*is v in the map?*
`Value`	`get(Object k)`	*get value for k*
`Value`	`put(Key k, Value v)`	*adds (k,v) to table*
`Value`	`remove(Object k)`	*remove mapping for k*
`int`	`size()`	*number of entries*

Note: no successor / predecessor queries.

Map Interface in Java

java.util.Map<Key, Value>

- No duplicate keys allowed.
- No *mutable* keys
  - If you use an *object* as a key, then you can’t modify that object later.

Symbol Table

Key Mutability

What time does this plane depart at?

```
SymbolTable<Time, Plane> t =
           new SymbolTable<Time, Plane>();

Time  t1 = new Time(9:00);
Time  t2 = new Time(9:15);

t.insert(t1, “SQ0001”);
t.insert(t2, “SQ0002”);

t1.setTime(10:00);

x = new Time(9:00);
t.search(x);
```

Symbol Table

Key Mutability

Moral: Keys should be immutable.

Examples: Integer, String

```
SymbolTable<Time, Plane> t =
          new SymbolTable<Time, Plane>();

Time  t1 = new Time(9:00);
Time  t2 = new Time(9:15);

t.insert(t1, “SQ0001”);
t.insert(t2, “SQ0002”);

t1.setTime(10:00);

x = new Time(9:00);
t.search(x);
```

Symbol Tables in Java

java.util.Map

public interface	**java.util.Map<Key, Value>**	
`Set<Map.Entry<Key, Value>`	`entrySet()`	*set of all mappings*
`Set<Key>`	`keySet()`	*set of all keys*
`Collection<Value>`	`values()`	*collection of all values*

Note: not sorted

not necessarily efficient to work with these sets/collections.

What is wrong here?

Example:

There is a bug here!

```
Map<String, Integer> ageMap = new Map<String, Integer>();

ageMap.put(“Alice”, 32);
ageMap.put(“Bernice”, 84);
ageMap.put(“Charlie”, 7);

Integer age = ageMap.get(“Alice”)
```

- Key-type: String
- Value-type: Integer

What is wrong here?

Example:

Map is an interface!
Cannot instantiate an interface.

```
Map<String, Integer> ageMap = new Map<String, Integer>();

ageMap.put("Alice", 32);
ageMap.put("Bernice", 84);
ageMap.put("Charlie", 7);

Integer age = ageMap.get("Alice")
```

- Key-type: String
- Value-type: Integer

Map Class in Java

Example: HashMap

```
Map<String, Integer> ageMap = new HashMap<String, Integer>();

ageMap.put("Alice", 32);
ageMap.put("Bernice", 84);
ageMap.put("Charlie", 7);

Integer age = ageMap.get("Alice");
System.out.println("Alice's age is: " + age + ".");
```

- Key-type: String
- Value-type: Integer

Map Class in Java

Example: HashMap

```
Map<String, Integer> ageMap = new HashMap<String, Integer>();

ageMap.put("Alice", 32);
ageMap.put("Bernice", null);
ageMap.put("Charlie", 7);

Integer age = ageMap.get("Bob");
if (age==null){
    System.out.println("Bob's age is unknown.");
}
```

- Returns "null" when key is not in map.
- Returns "null" when value is null.

Map Classes in Java

HashMap

Symbol Table

- containsKey
- containsValue
- entrySet
- get
- isEmpty
- keySet
- put
- putAll
- remove
- values

TreeMap

Dictionary

- containsKey
- containsValue
- entrySet
- get
- isEmpty
- keySet
- put
- putAll
- remove
- values

Map Classes in Java

HashMap

Symbol Table

TreeMap

Dictionary

- ceilingEntry
- ceilingKey
- descendingKeySet
- firstEntry
- firstKey
- floorEntry
- floorKey
- headMap
- higherEntry
- higherKey
- … (and more)

Lots of functionality

Wide Interfaces

vs.

Narrow Interfaces

Limited functionality

Lots of functionality
- Java
- No guarantee of efficiency.
- Easy to use (badly).

Wide Interfaces

vs.

Narrow Interfaces

Limited functionality
- Enforces proper use.
- Restricts usage.

Symbol Tables are Useful

Examples:

1. Spelling correction (key=misspelled word, data=word)
2. Scheme interpreter (key=variable, data=value)
3. Web server
   - Lots of simultaneous network connections.
   - When a packet arrives, give it to the right process to handle the connection.
   - key=ip address, data = connection handler

In this cases, O(log n) often isn't fast enough!

Symbol Tables are Useful

Example 1: Pilot Scheduling

1. Check to see if feasible to schedule at time *t*.

   No two airplanes can land with 15 minutes of each other.

2. Find schedule of pilot *p*.

   Get a list of all the planes that are being flown by a specified pilot.

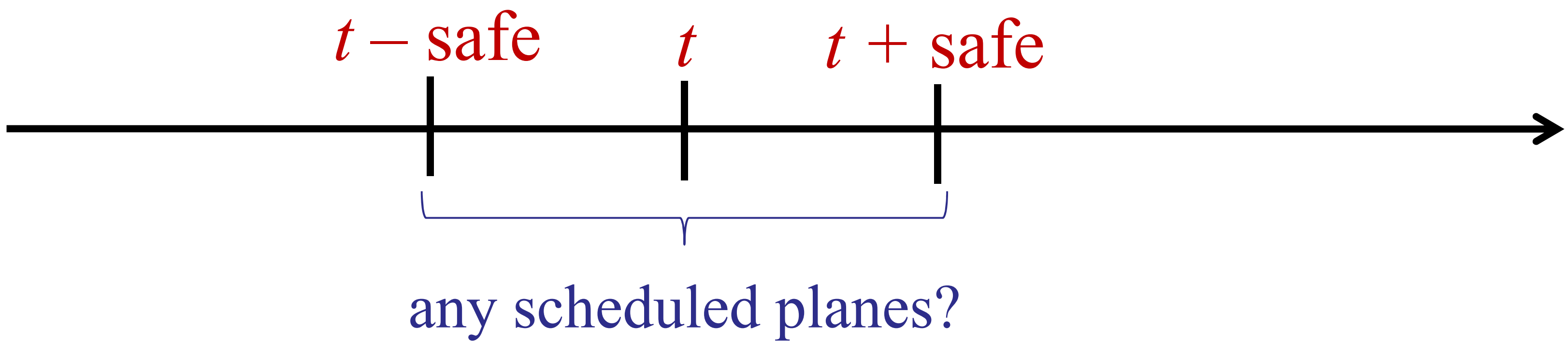
Which can be efficiently solved with a symbol table?

1. Both: scheduling and pilots info.
2. Only scheduling.
3. Only pilot info.
4. Neither.

Symbol Tables are Useful

Example 1: Pilot Scheduling

1. Check to see if feasible to schedule at time t.
    - Hard with a symbol table!
    - Need to find out if there are any planes scheduled in the interval $[t, t \pm \text{safe\_distance}]$



Symbol Tables are Useful

Example 1: Pilot Scheduling

1. Check to see if feasible to schedule at time *t*.
2. Find schedule of pilot *p*.
   - Perfect for a symbol table!
   - Can insert new pilots.
   - Can search for (and update) existing pilots.
   - Listing all pilots?

Symbol Tables are Useful

Example 2: Document Distance

- Given two documents A and B, how similar are they?
  - Two documents are *similar* if they have similar words in similar frequencies.
  - Formally, define each text as a vector with one entry per word.
  - The distance between the two texts is the angle between the two vectors.

Symbol Tables are Useful

Example 2: Document Distance

- Step 1: Read in each document
  - Read the file as a string.
  - Parse the file into words.
  - Sort the list of words.
  - Count the frequency of each word.
- Step 2: Compare the two documents
  - Calculate the norm $|A|$ and $|B|$ of each vector
  - Calculate the dot product AB.
  - Calculate the angle between A and B.

Symbol Tables are Useful

Example 2: Document Distance

– Step 1: Read in each document

$O(n)$ • Read the file as a string.

$O(n)$ • Parse the file into words.

$O(n \log n)$ • Sort the list of words.

$O(n)$ • Count the frequency of each word.

– Step 2: Compare the two documents

$O(n)$ • Calculate the norm $|A|$ and $|B|$ of each vector

$O(n)$ • Calculate the dot product AB.

$O(n)$ • Calculate the angle between A and B.

Performance Profiling (Sorting)

(Dracula vs. *Lewis & Clark)*

Step	Function	Running Time
Create vectors:	Read each file	1.03s
	Parse each file	1.23s
	Sort words in each file	2.04s
	Count word frequencies	0.41s
Dot product:		6.10s
Norm:		0.01s
Angle:		0.02s
Total:		**12.75s**

Performance Profiling (Symbol Table)

(Dracula vs. *Lewis & Clark)*

Step	Function	Running Time
Create vectors:	Read each file	1.19s
	Parse each file	1.37s
	Sort words in each file	0
	Count word frequencies	0
Dot product:		0.03s
Norm:		0.01s
Angle:		0.02s
Total:		**2.43s**

Symbol Tables are Useful

Example 2: Document Distance

– Step 1: Read in each document

  - Read and parse the file.
  - Put each (word, count) in a HashMap.

  Symbol Table:

  - key (String) = word
  - value (Integer) = count (# times in doc)

Document Distance

(Dracula vs. Lewis & Clark)

Version	Change	Running Time
Version 1		4,311.00s
Version 2	Better file handling	676.50s
Version 3	Faster sorting	6.59s
Version 4	Symbol Table	2.35s

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 $m = 10$.

<u>(key, value)</u>

(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:

- Too much space
  - If keys are integers, then table-size > 4 billion
- 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 = 28 letters (antidisestablishmentarianism?)
  - 28 letters * 5 bits = 140 bits
  - So we can store any English word in a direct-access array of size 2^{140}.

Direct Access Tables

Pythagoras said, "Everything is a number."

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

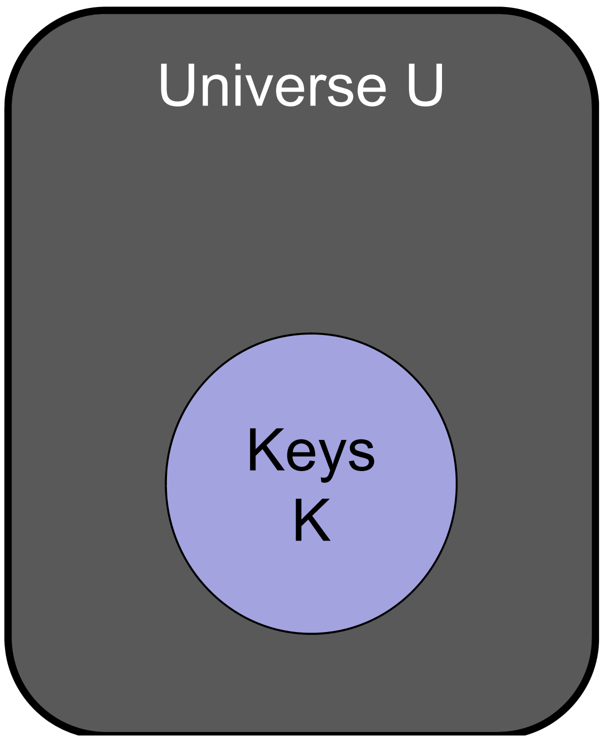
- English:
  - 26 letters => 5 bits/letter
  - Longest word = 28 letters (antidisestablishmentarianism?)
  - 28 letters * 5 bits = 140 bits
  - So we can store any English word in a direct-access array of size 2^{140}. ≈ number of atoms in observable universe

Hash Functions

Problem:

- Huge universe U of possible keys.
- Smaller number n of actual keys.

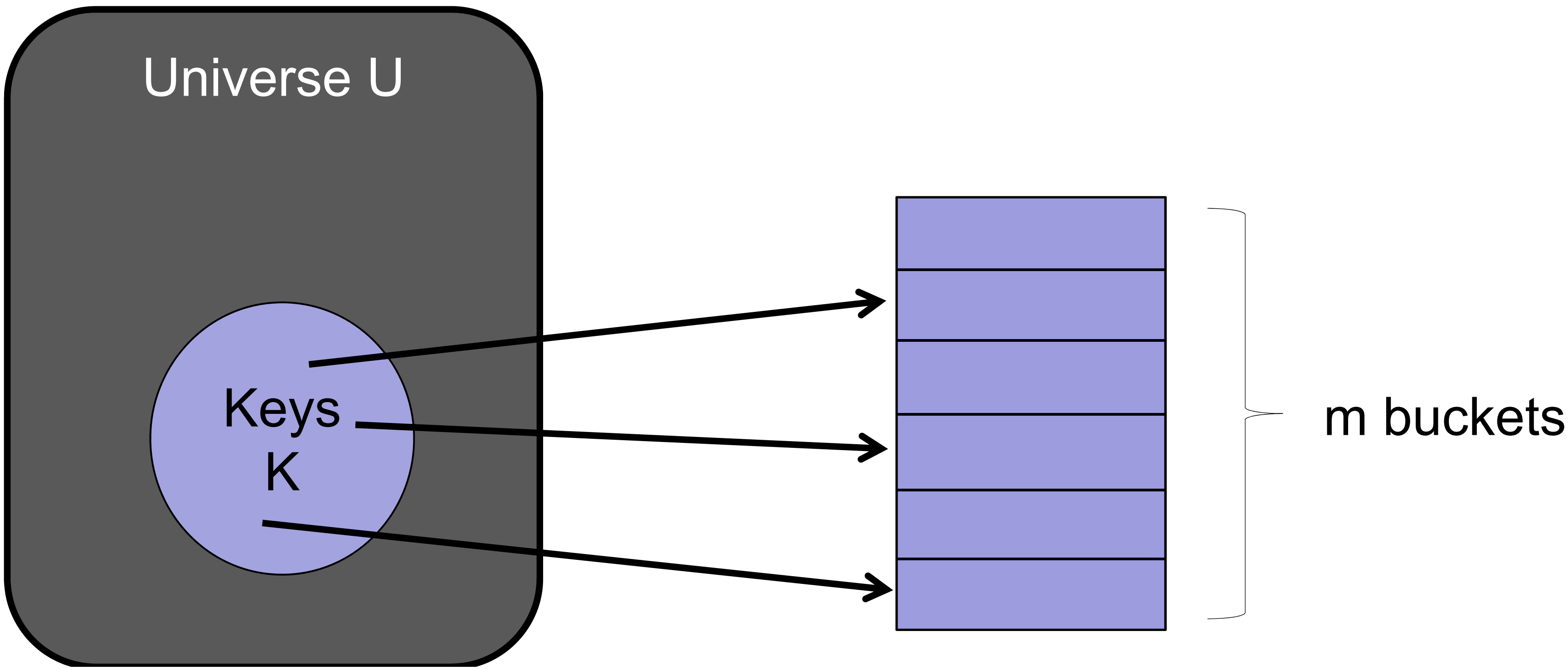
e.g., 2^{140}



Hash Functions

Problem:

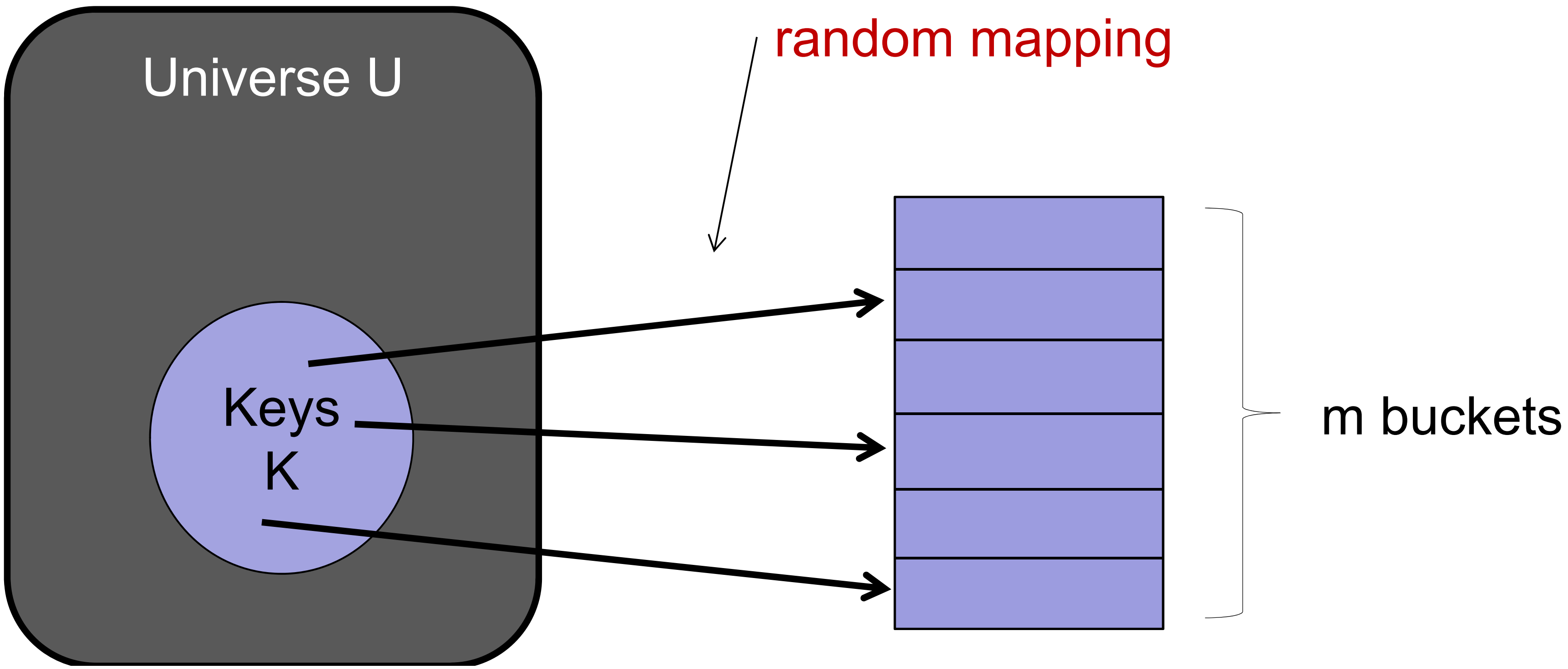
- Huge universe U of possible keys.
- Smaller number n of actual keys.
- How to map n keys to $m \approx n$ buckets?



Hash Functions

Problem:

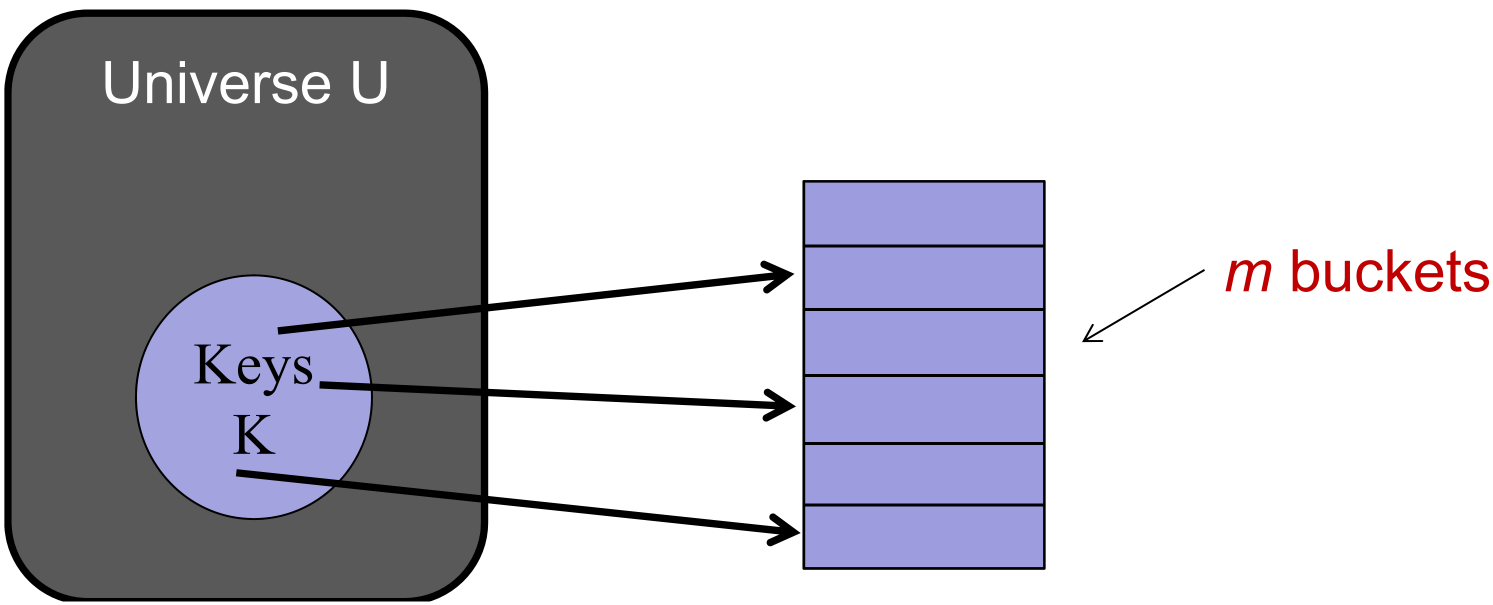
- Huge universe U of possible keys.
- Smaller number n of actual keys.
- How to map n keys to $m \approx n$ buckets?



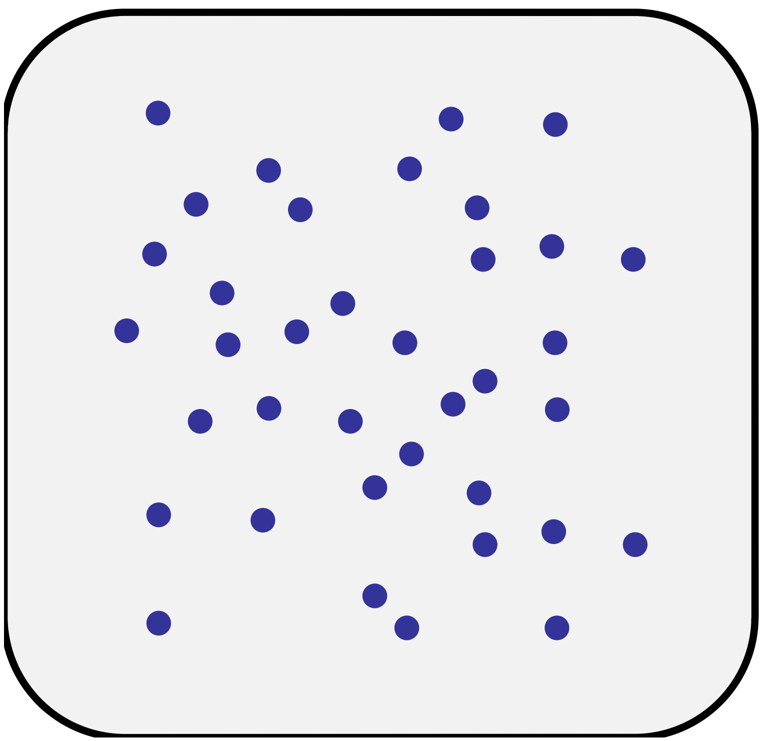
Hash Functions

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
- For now: assume hash function has cost 1 to compute.



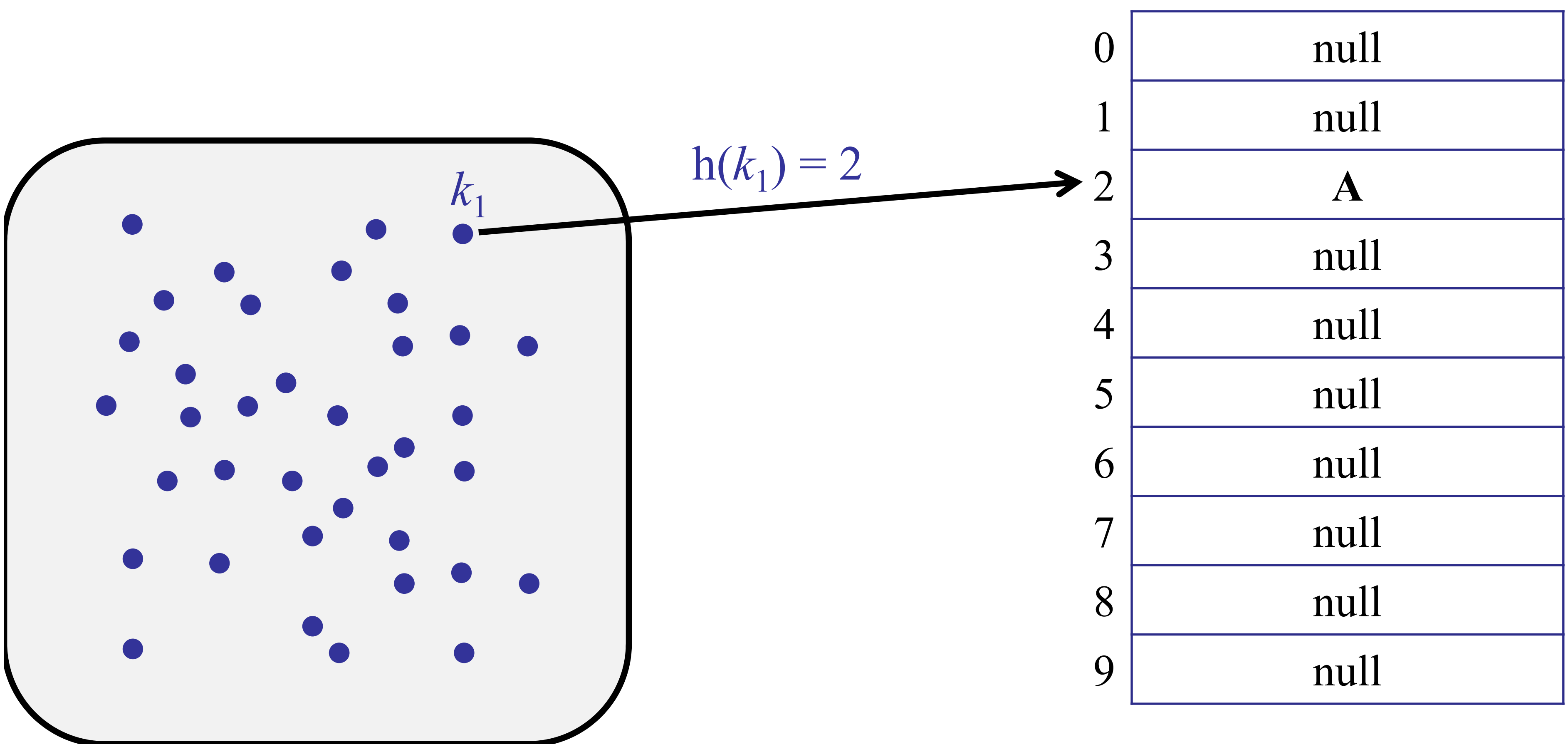
Hash Functions



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

Hash Functions

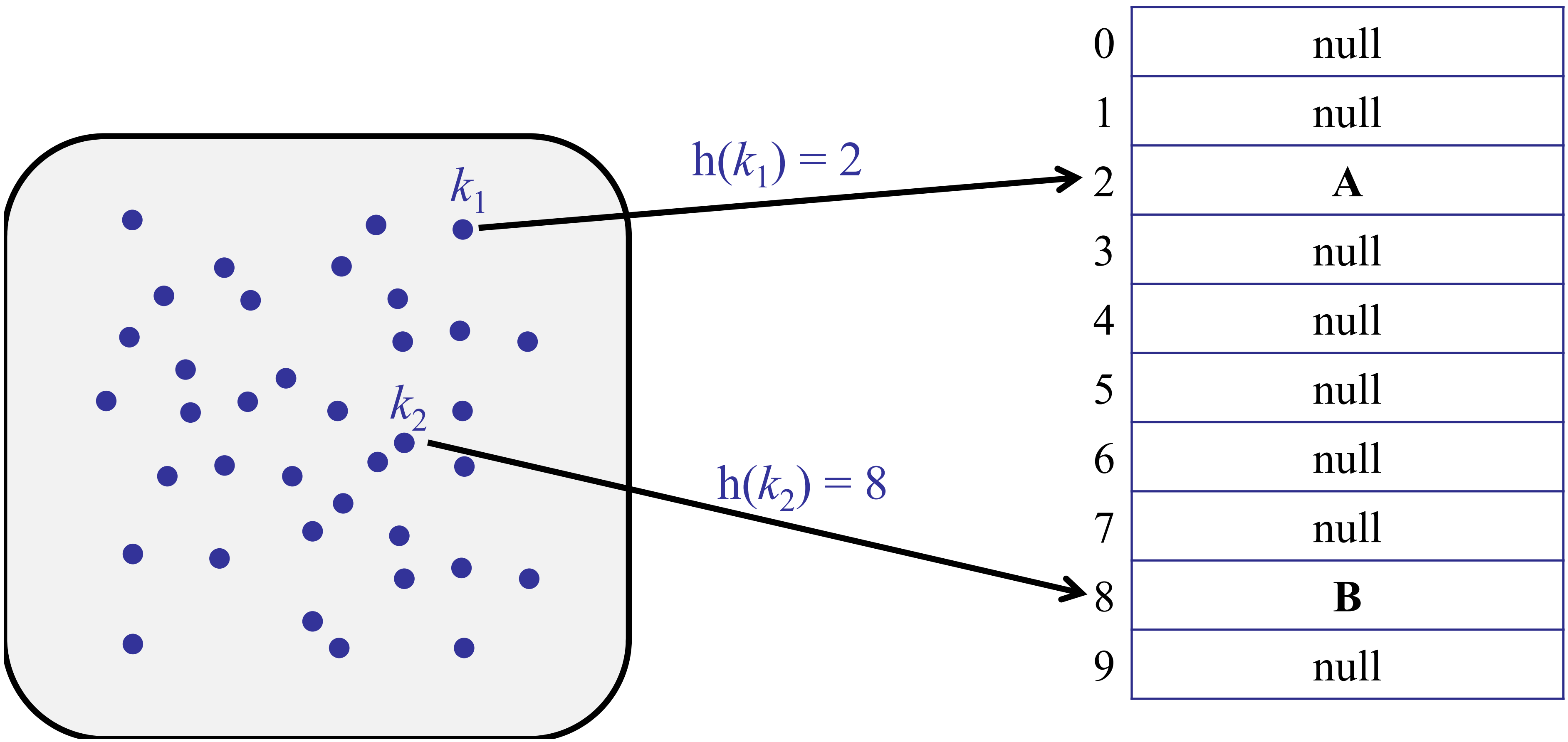
insert(k_1, A)



Hash Functions

insert(k_1, A)

insert(k_2, B)



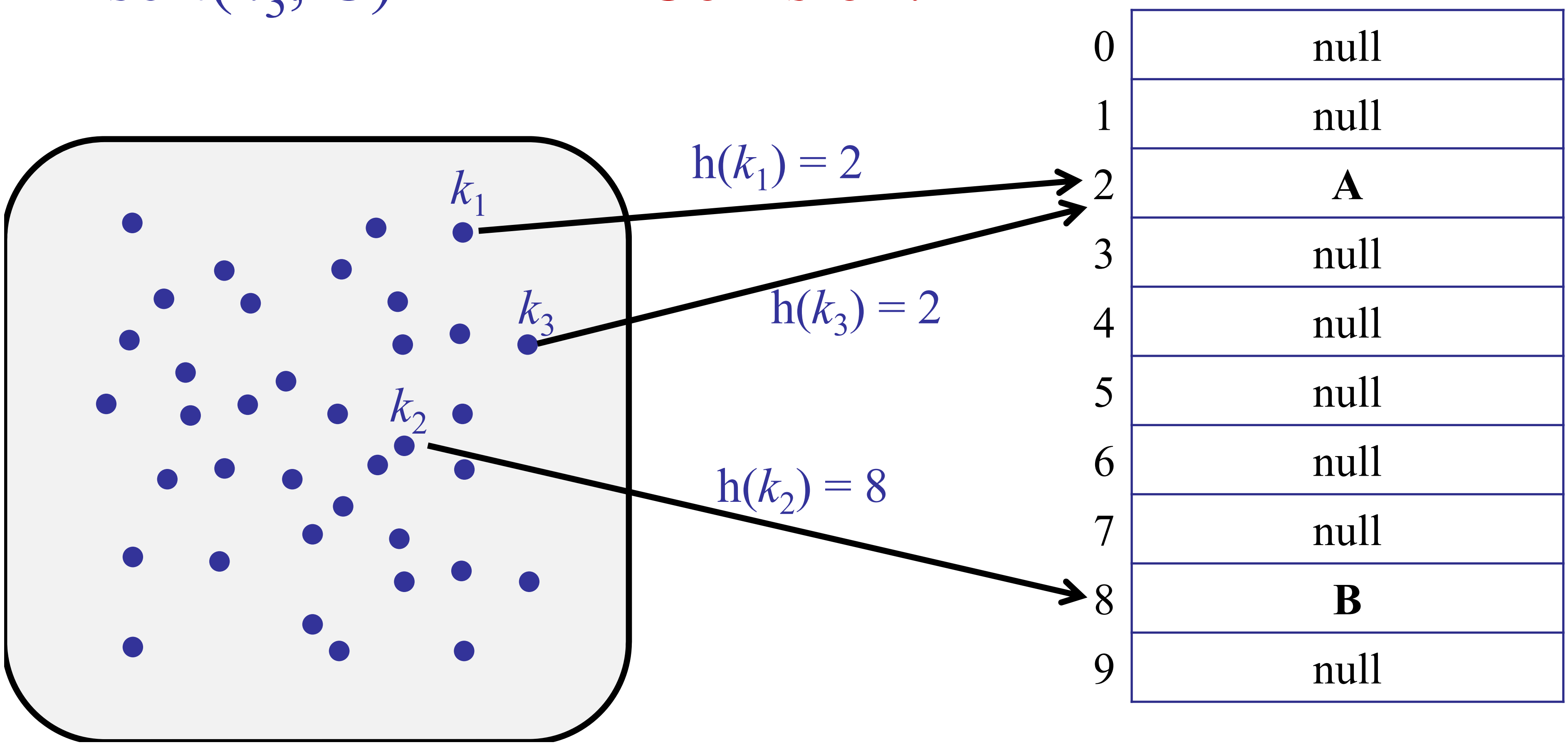
Hash Functions

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

Hash Functions

Collisions:

- We say that two distinct 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)

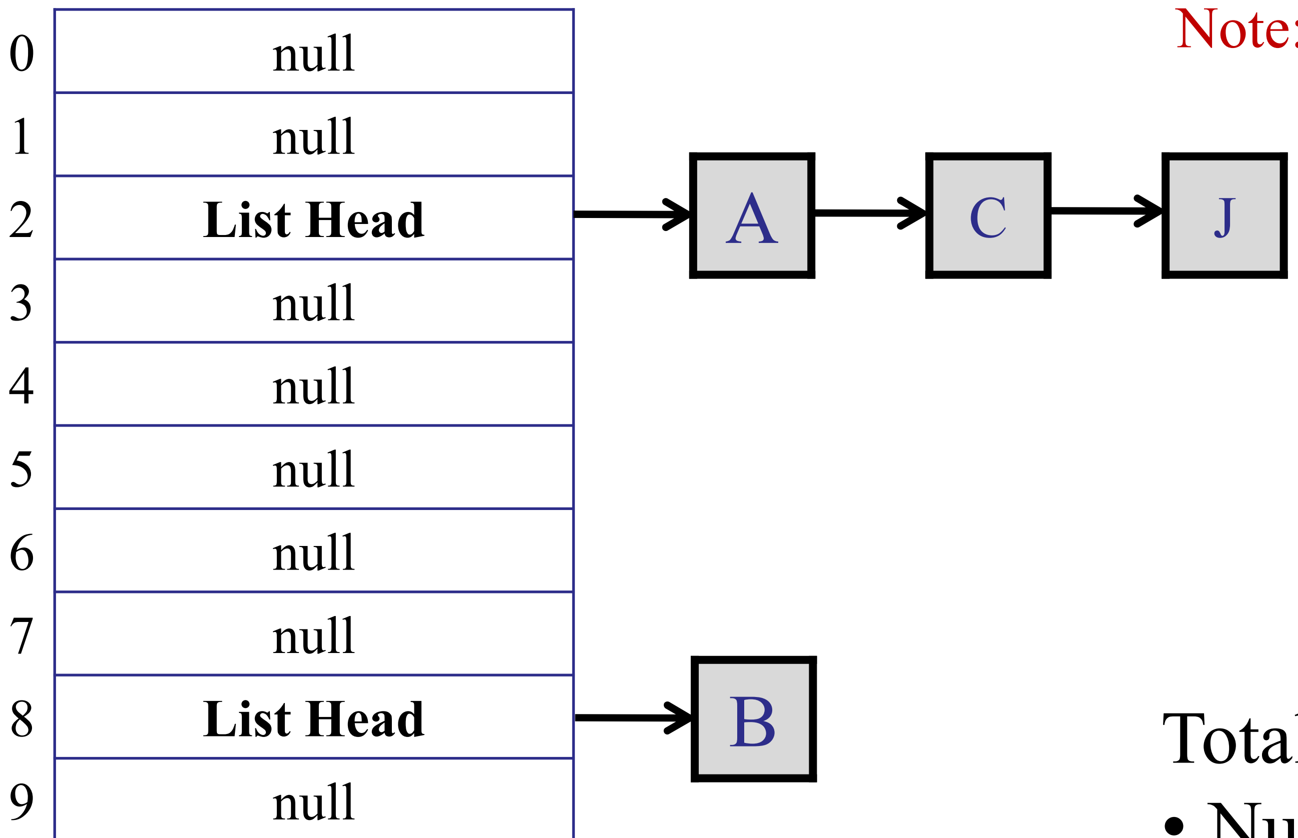
- Put both items in the same bucket!

Idea: open addressing (next week)

- Find another bucket for the new item.

Chaining

Each bucket contains a linked list of items.



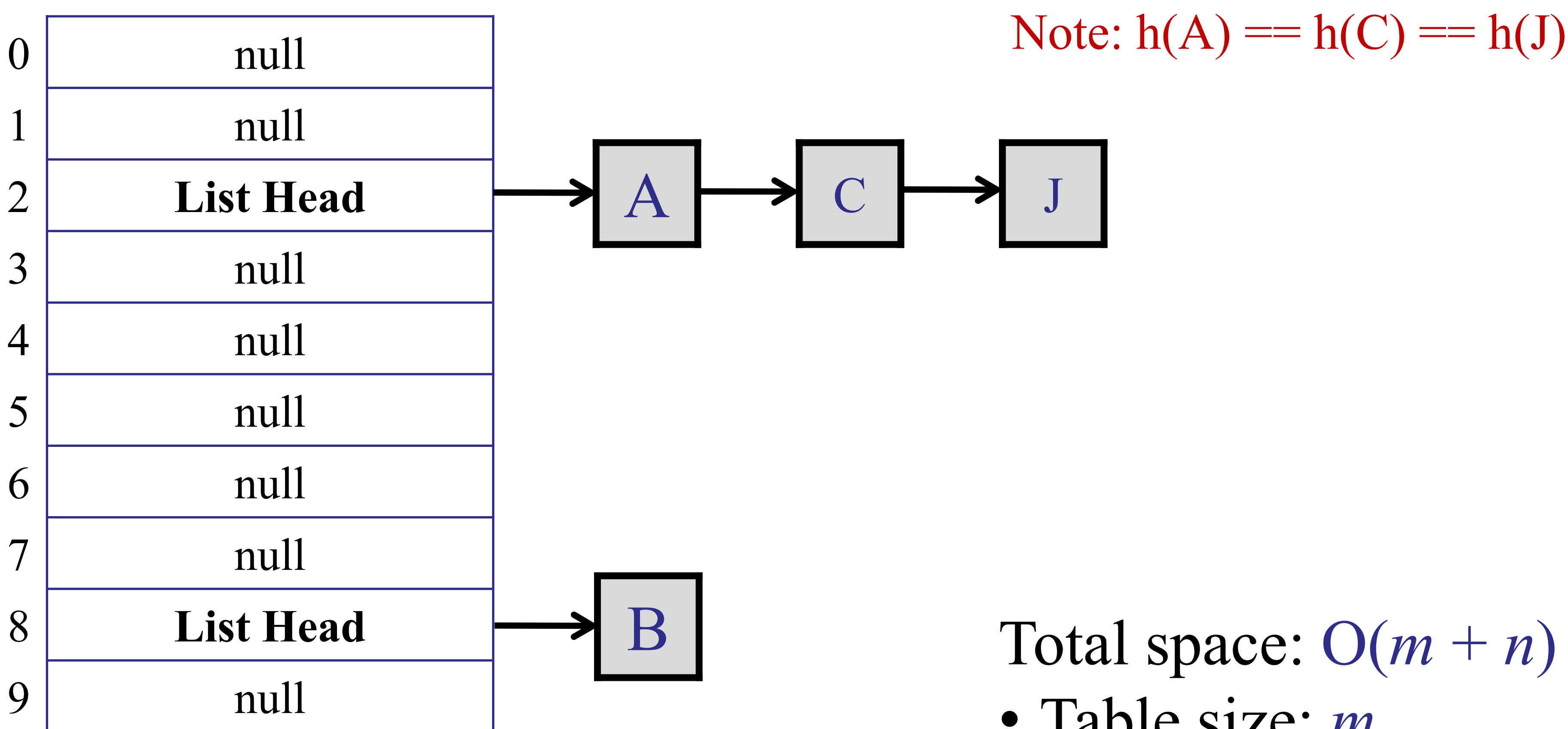
Note: h(A) == h(C) == h(J)

Total space:

- Number buckets: m
- Number entries: n

Chaining

Each bucket contains a linked list of items.



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)?

✓ 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. $O(n^2)$.

Hashing with Chaining

Operations:

Always O(1).

- 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 searching a (key, value)?

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. We cannot determine it without knowing h.

Hashing with Chaining

Operations:

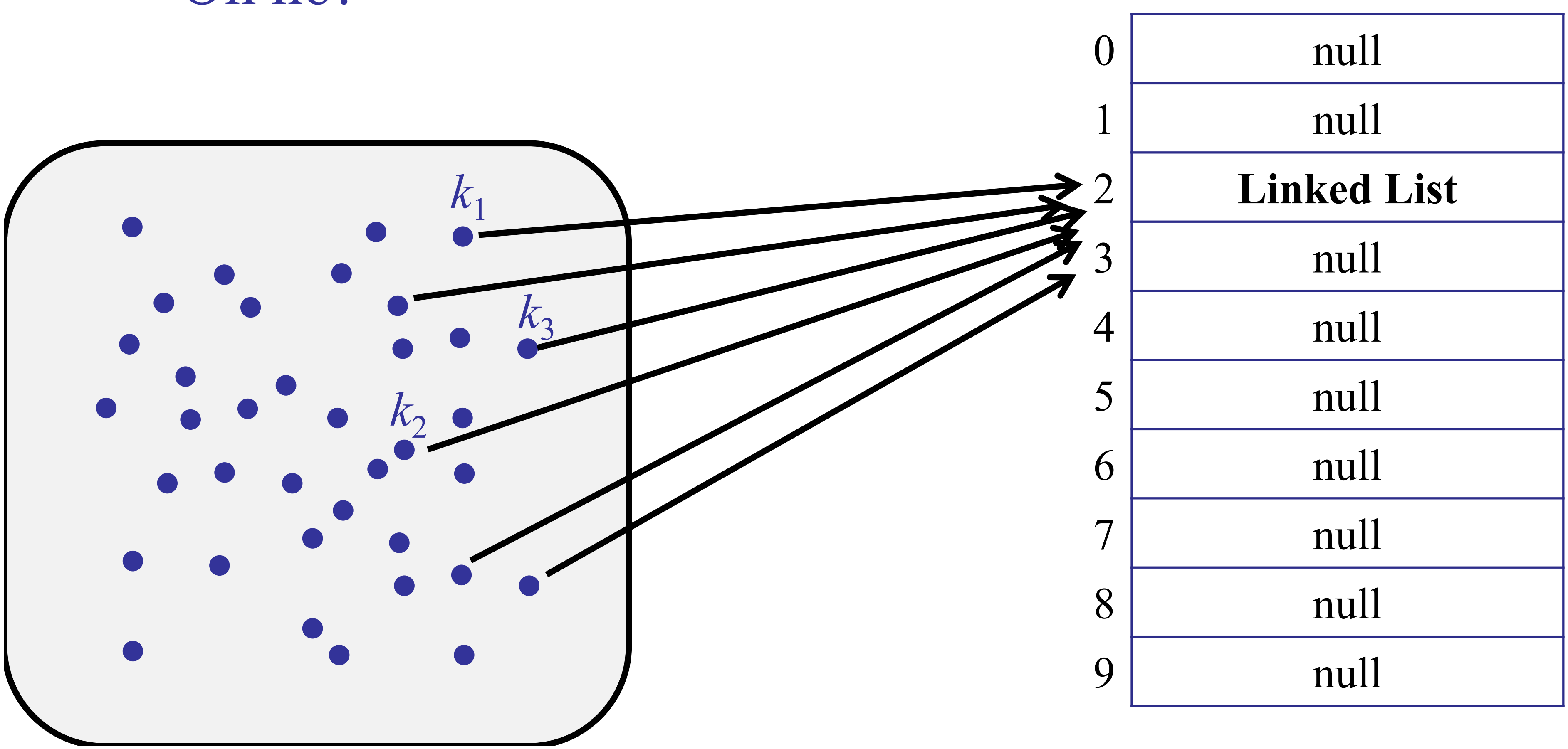
- insert(key, value)
  - Calculate h(key)
  - Lookup h(key) and add (key,value) to the linked list.

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

Hashing with Chaining

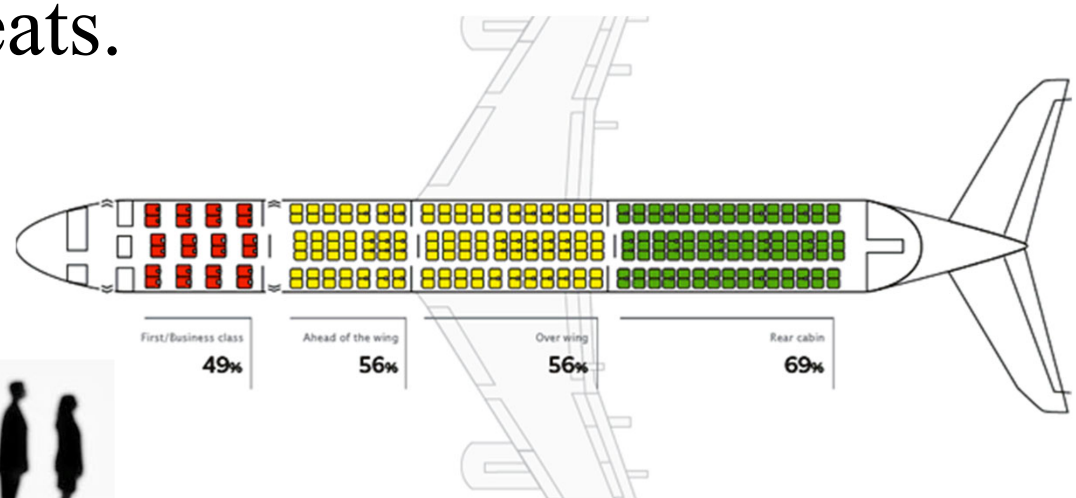
Assume all keys hash to the same bucket!

- Search costs O(n)
- Oh no!



Puzzle Break

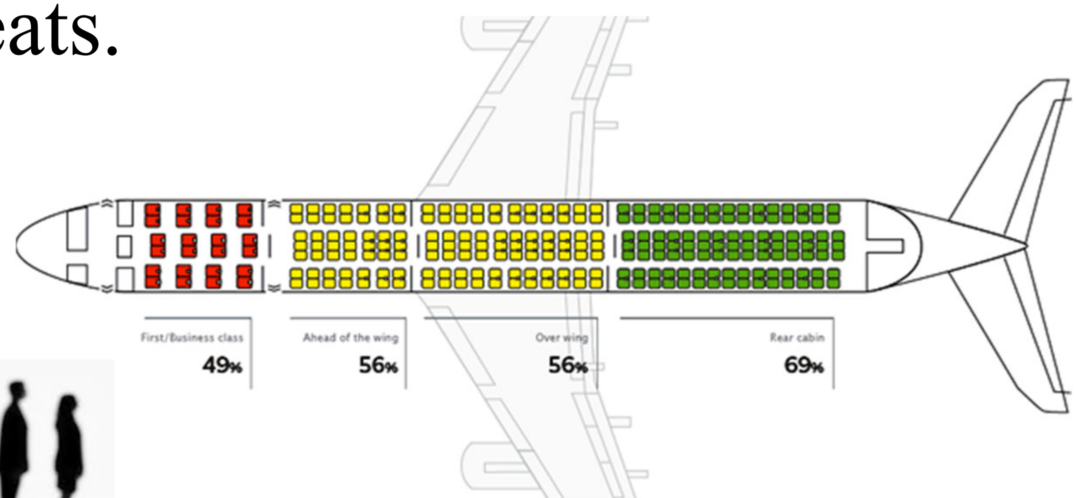
An airplane has 100 seats.



100 passengers board the airplane in a random order.

Puzzle Break

An airplane has 100 seats.



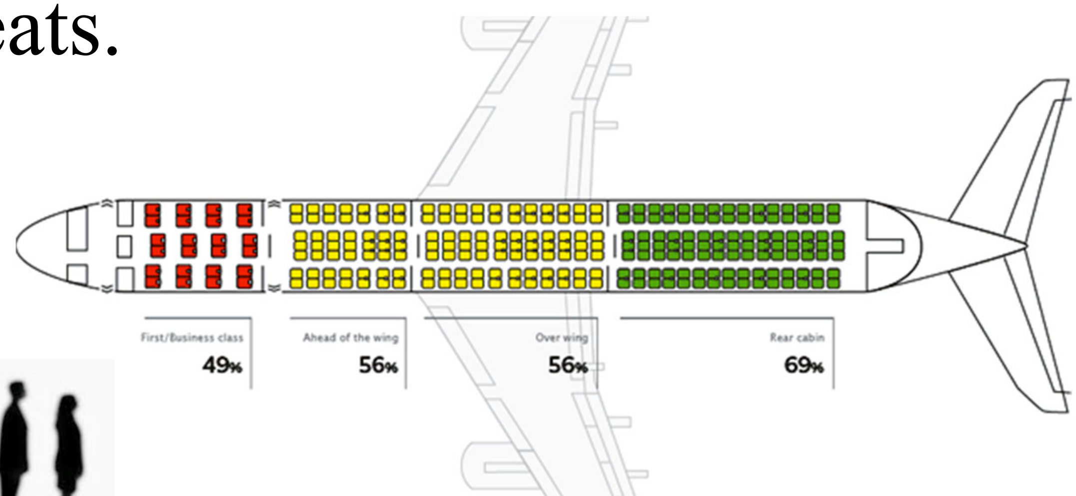
100 passengers board the airplane in a random [illegible]er.

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.

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.