Map (or dictionary or associative array)

- Behaviour
  - Associates some meaningful data (the ”key”) with a value
  - Stores and accesses values using the key
  - No specific ordering of the data
- Operations:
  - Put (key, value)
  - Get (key) -> value
  - Size () -> integer
  - ContainsKey (key) -> Boolean
  - ContainsValue (value) -> Boolean (optional)
  - Remove (key) -> Boolean (optional)

Map examples

- Trivial map
  - Key is the sequence of integers 1, 2, 3, 4, …
  - Implementation: a standard array

- More complex map
  - Key is your Banner ID
  - Value is your netid
  - Implementation: hash table

DALHOUSIE UNIVERSITY
Inspiring Minds

Recognizing a spot for a standard ADT

- Stack
  - Doing a set of operations that might need undoing in the reverse order
  - Exploring options that involve backtracking (changing or removing the most recent choice)
  - Recursion (implicit or explicit stack)
  - Situations where proper nesting is involved
  - Exploring connected problems that handle depth of coverage before breadth of coverage
- Queue
  - Simulations of scheduling with items arriving at different times
  - Processing a growing list of items in a way that ensures that each item is handled in a “fair” timeframe
  - Exploring connected problems that handle breadth of coverage before depth of coverage

Recognizing a spot for a standard ADT

- Priority Queue
  - I need to store items and retrieve them in an order that I define (order can change)
  - Often used in scheduling
- Set
  - I have a collection if items to store
    - I just care about having one copy
  - I want to access the items randomly
  - I want to iterate over the set
  - I don’t have any particular order needed for the data
- List
  - I have a collection of items to store
    - I might have several copies of the same thing
  - I want to access items randomly
  - I want to be able to impose an order to the set by sorting
  - I want to iterate over the set in the sorted order
- Map
  - Random access to key, value pairs when *exact matches to keys* is all that is needed

What is stored in an ADT?

- Basic data types, like “int” (for integers)
  - Behave exactly as we expect them to

- Objects
  - Important to understand exactly what is being stored

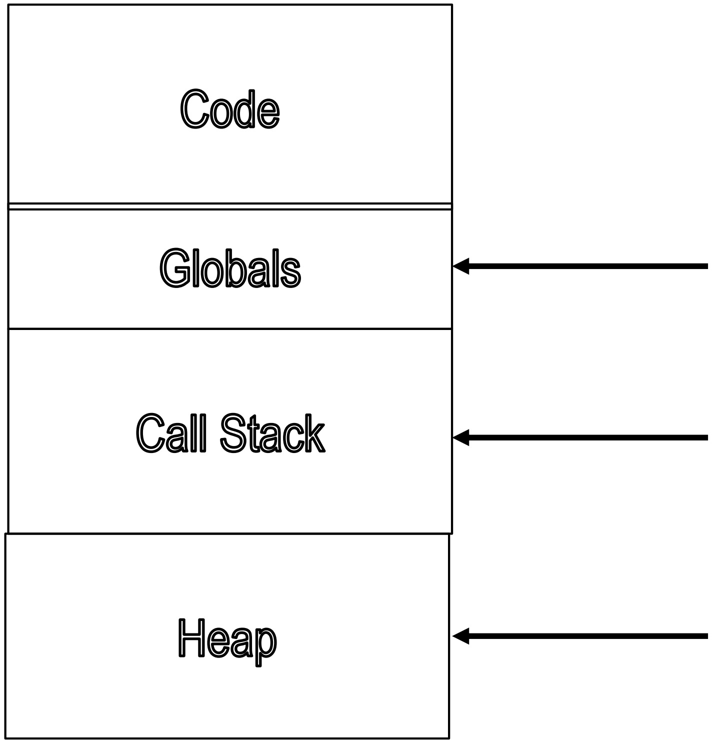
Is there a difference?

```
int a;
int b;

a = 10;
b = a;
a = 20;

System.out.println( b );
```

```
myIntClass a;
myIntClass b;

a = new myIntClass( 10 );
b = a;
a.setValue( 20 );

System.out.println( b.getValue() );
```

```
Public class myIntClass {
            int value;
            public void setValue( int val ) {
                        value  = val;
            }
            public int getValue( ) {
                        return value;
            }
}
```

Elements of a Process



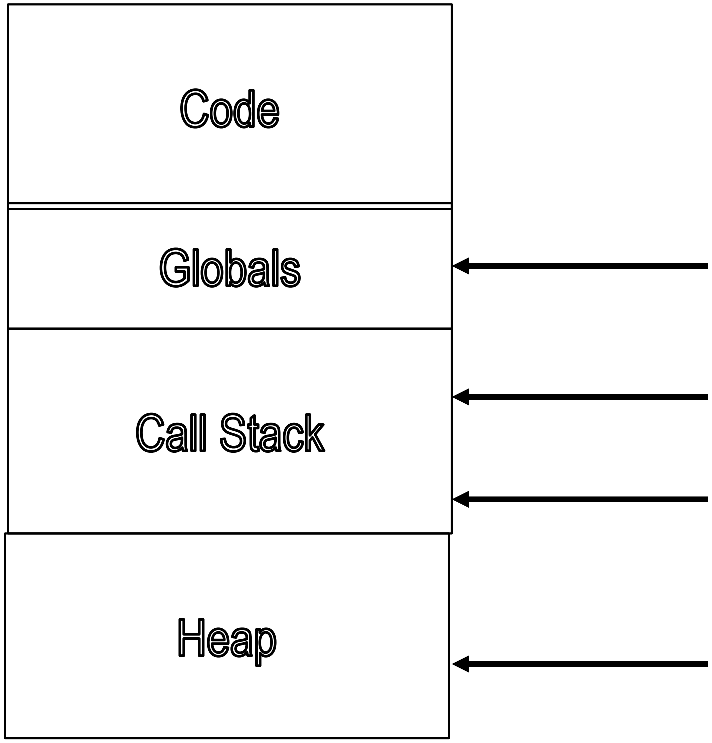
“static” variables in Java

Local variables

Objects created with “new”

Storing Objects

```
public int myMethod() {
    int someInt;
    static boolean test;
    OtherClass anObject;

    anObject = new OtherClass();
}
```



Code

Globals

Call Stack

Heap

test

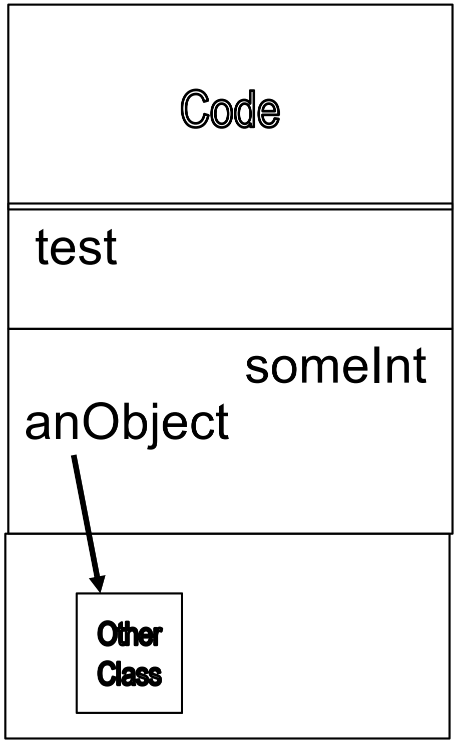
someInt

anObject – just space for a reference to an object

the “new” object

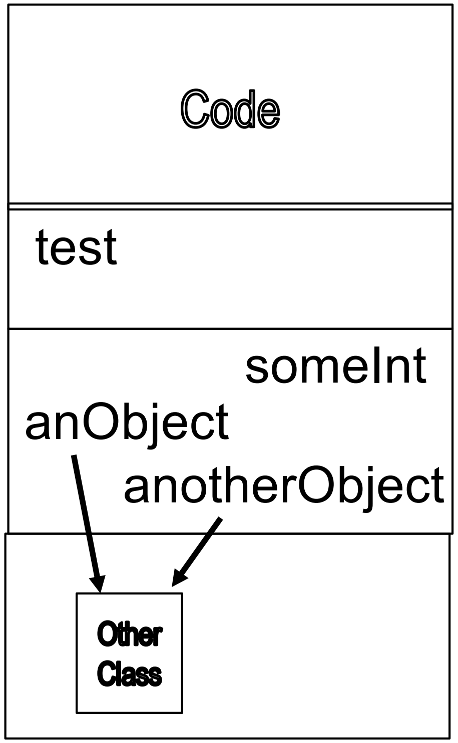
Storing Objects

```
public int myMethod() {
    int someInt;
    static boolean test;
    OtherClass anObject;

    anObject = new OtherClass();
}
```



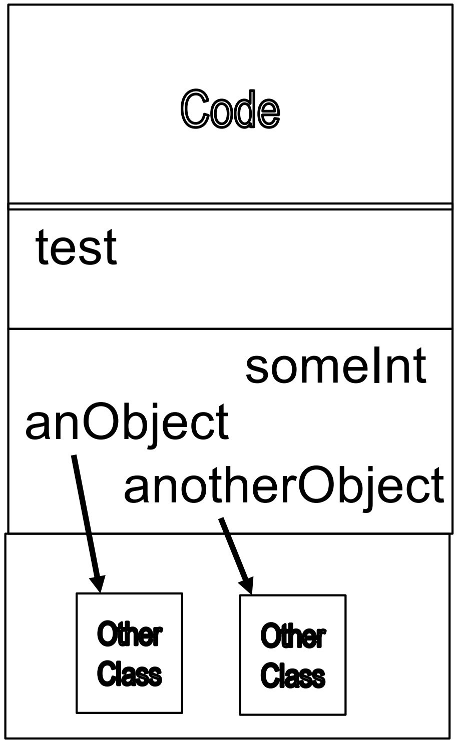
Storing Objects

```
public int myMethod() {
    int someInt;
    static boolean test;
    OtherClass anObject;

    anObject = new OtherClass();

    OtherClass anotherObject;

    anotherObject = anObject;
}
```



When we assign object values, we are copying the reference to the object. We are not copying the content.

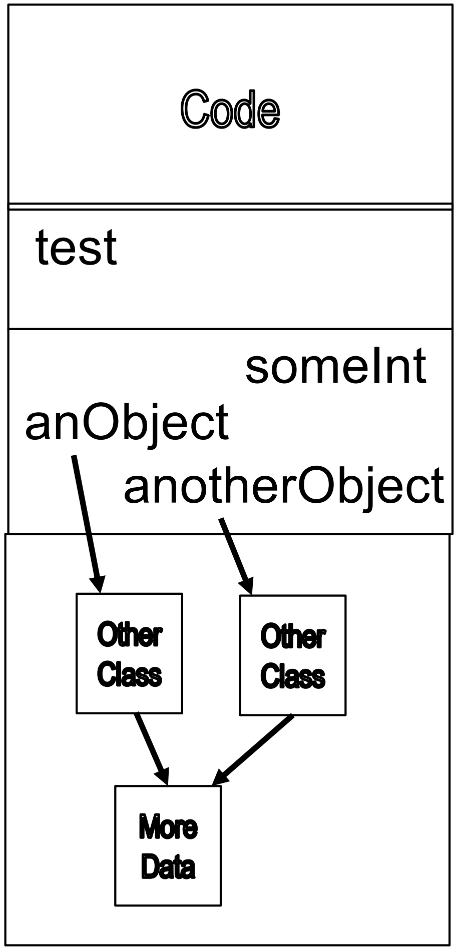
Storing Objects



```
public int myMethod() {
    int someInt;
    static boolean test;
    OtherClass anObject;

    anObject = new OtherClass();

    OtherClass anotherObject;

    anotherObject.copy(anObject);
}
```

Classes often have a “copy” method to make an actual copy of the class instead.

Storing Objects

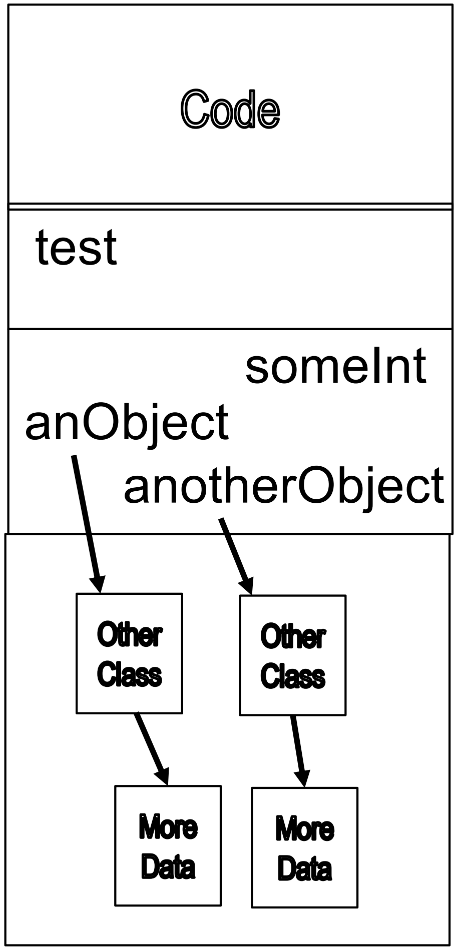


```
public int myMethod() {
    int someInt;
    static boolean test;
    OtherClass anObject;

    anObject = new OtherClass();

    OtherClass anotherObject;

    anotherObject.copy(anObject);
}
```

An object of a class like OtherClass may reference other objects.
A "copy" method may not always copy the content of those references. That kind of copy is called a "shallow copy".

DALHOUSIE UNIVERSITY *Inspiring Minds*

Storing Objects



```
public int myMethod() {
    int someInt;
    static boolean test;
    OtherClass anObject;

    anObject = new OtherClass();

    OtherClass anotherObject;

    anotherObject.deepCopy(
            anObject);
}
```

A copy method that copies all of the underlying objects is often called a “deep copy” of the object.

What is stored in an ADT?

- Take-away:
  - When you put an object into two ADTs, know whether you expect to have each copy be shared or independent
    - If shared, then put the reference into each ADT
    - If independent then put a copy into each ADT
      - Understand if you need a shallow or a deep copy

Combine ADTs

- You can combine ADTs to meet the need.
- Example
  - You want to store all items in your house to be retrieved by their colour.
    - Store all items of the same colour in one set.
    - Store these sets in a map where the key is the colour name and the value is the set

Data structures with a fixed size

Array

- A fixed-size linear sequence of items
- Uses integers to identify the order of items in the sequence
    - Start at index number 0 in many programming languages
        - Historical context based on implementation efficiency

Declaring an array in Java

String[ ] anArray; ← Creates a reference to an array, but there is no actual array to store data yet.

String[ ] anArray = new String[10]; ← Creates the space for 10 entries in an array. We see that an array is treated like an object of its own.

How would you create a 2d array?

- Integer[][] arrayName = new Integer[20][15];

Hash table

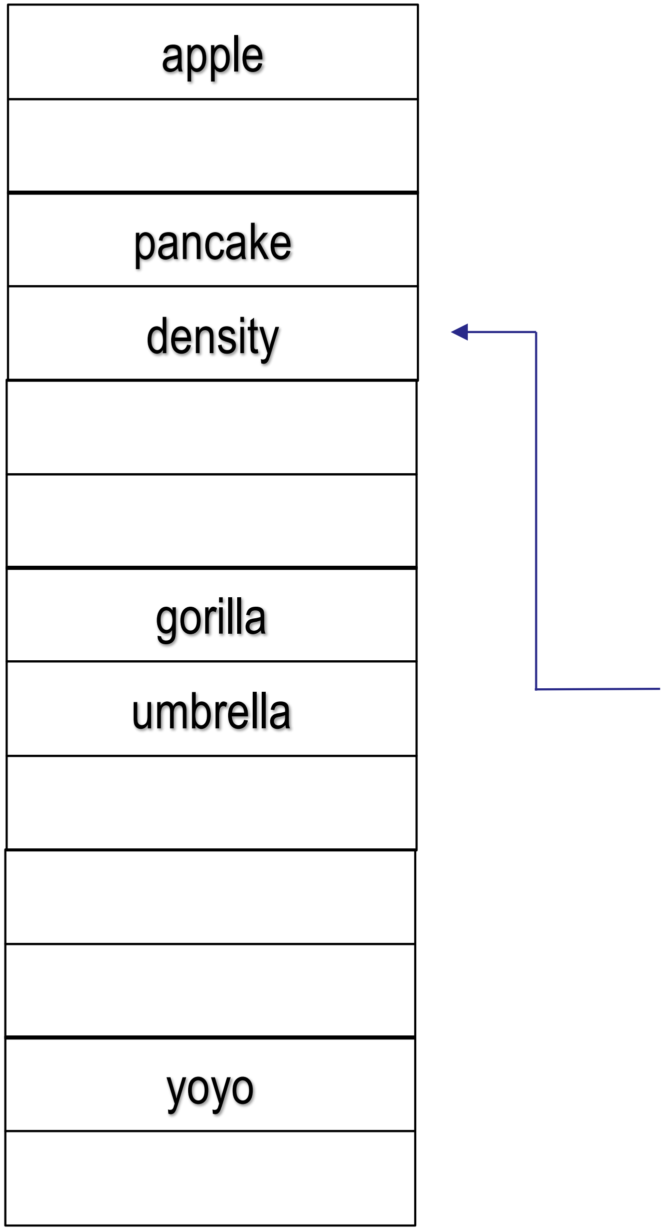
- An organization of data in an array to let us search for an entry quickly.
- Key concept:
  - Use a formula to convert the key to store into an array index
    - Called the "hash function"
  - Store the value in the array at the computed index value
  - Have rules to handle the case where two values are converted to the same array index
    - Called a "collision" in the hash table
- In a moderately-filled array, you expect to find a search value in constant time.

Hash table example

- Array size: 13
- Data stored: alphabetic lower-case strings
- Hash function: the position in the alphabet of the first letter of the string (starting at position 0)
- Array index: take the hash value modulo 13

- Expected collisions:
  - All the strings that start with the same letter end up at the same index
  - Two letters of the alphabet converge on the same index

Hash table example



apple
pancake
density
gorilla
umbrella
yoyo

Add “quiet”

Hash value is 16 (‘q’ – ‘a’)

Index is 16 mod 13 = 3

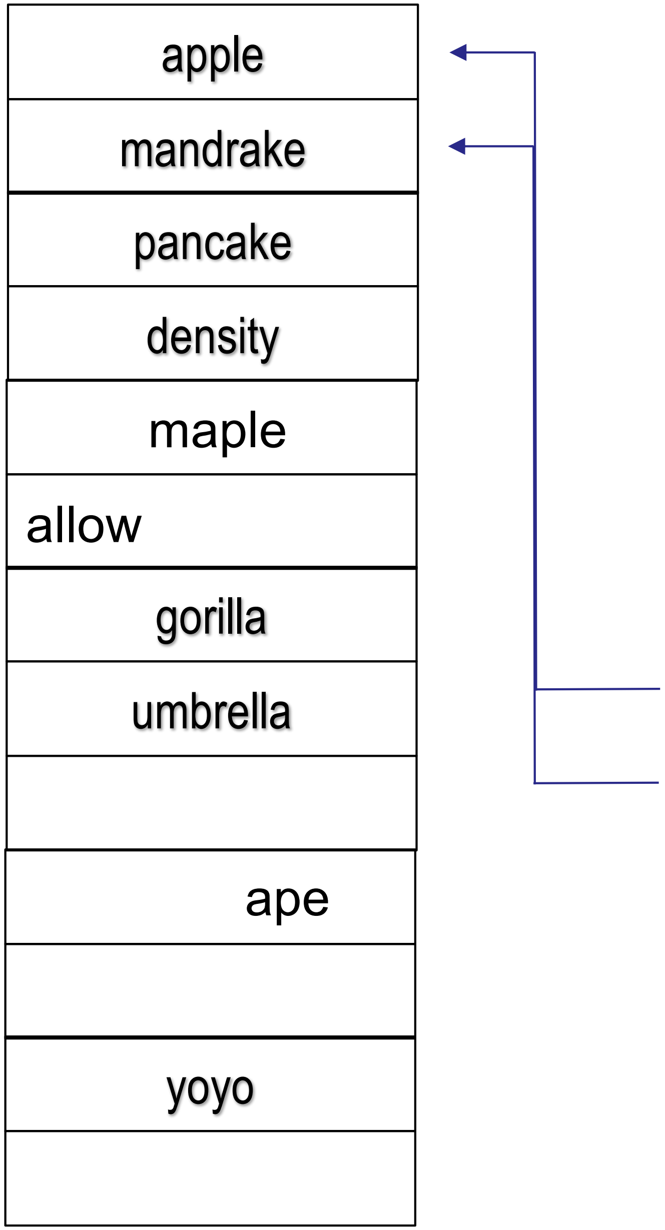
Store “quiet” here

How to deal with hash table collisions

- Have a data structure at each array index to catch all values that belong at the index (called “open hashing”)
  - Linked list, binary tree, …
- In-place: look for another “predictable” place in the array to store the entry (called “closed hashing”)
  - Move forward k entries in the array until you find an entry spot
    - Linear probing: k=1
    - Quadratic probing: k follows a sequence 1^2, 2^2, 3^2, 4^2, …
    - Double hashing: k is the result of applying a second hash function to the value to be stored
  - More complex resolution schemes
    - Eg. Cuckoo hashing

DALHOUSIE UNIVERSITY
Inspiring Minds

Hash table example with linear probing



apple
mandrake
pancake
density
maple
allow
gorilla
umbrella
ape
yoyo

Add “mandrake”

Hash value is 13 (‘m’ – ‘a’)

Index is 13 mod 13 = 0

Try to store “mandrake” here, but the entry is full

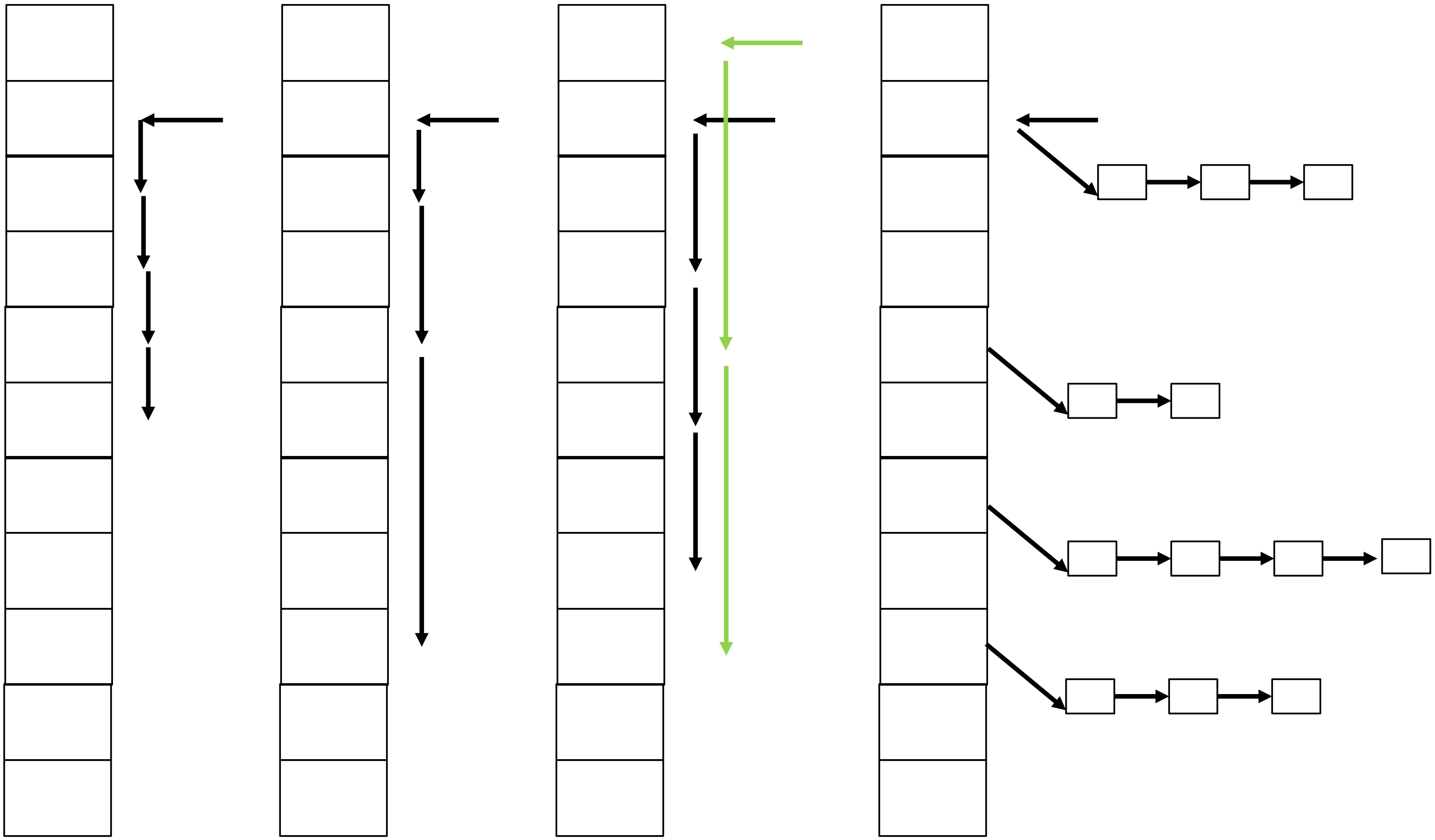
Linear probing: advance by 1 until we find an empty entry

Store “mandrake” in this empty entry

Hash table collisions

- Other ways to handle hash table collisions
  - Linear probing (already seen)
  - Quadratic probing
  - Double hashing
    - Use another hash function to tell you how much to jump ahead
  - Store a secondary data structure at each entry in the hash table and put all items that map to entry into the secondary structure
    - Often use a linked list at each entry and call it “chaining”
- Other specialized approaches, like cuckoo hashing and Robin Hood hashing, also exist.

Collision management



Linear probing

Quadratic probing

Double hashing

Chaining