CSCI 232:

Data Structures and Algorithms

Hashing (Part 1)

Reese Pearsall

Summer 2025

Program 1 due **tomorrow** at 11:59PM

No in-person office hours tomorrow
(email me if you need anything)

Map / Dictionary

A **map** or **dictionary** is an unordered collection of key/value pairs.

Maps a **key** to a **value**

Keys		**Values**
Dallas	→	Cowboys
Chicago	→	Bears
New England	→	Patriots
Denver	→	Broncos
Pittsburgh	→	Steelers
Kansas City	→	Chiefs
Miami	→	Dolphins
Tennessee	→	Titans
New York	→	Giants
Buffalo	→	Bills
Atlanta	→	Falcons

General Rules

1. Keys should not be shared

(no duplicate keys)

New York : Jets
New York : Giants ✗

1. Keys should not be mutable

String ✓
int ✓
double ✓

Arrays ✗
Objects 〰

Map / Dictionary

A **map** or **dictionary** is an unordered collection of key/value pairs.

Maps a **key** to a **value**

Keys		Values
Dallas	→	Cowboys
Chicago	→	Bears
New England	→	Patriots
Denver	→	Broncos
Pittsburgh	→	Steelers
Kansas City	→	Chiefs
Miami	→	Dolphins
Tennessee	→	Titans
New York	→	Giants
Buffalo	→	Bills
Atlanta	→	Falcons

Implementation?

General Rules

1. Keys should not be shared
(no duplicate keys)

New York : Jets
New York : Giants ✗

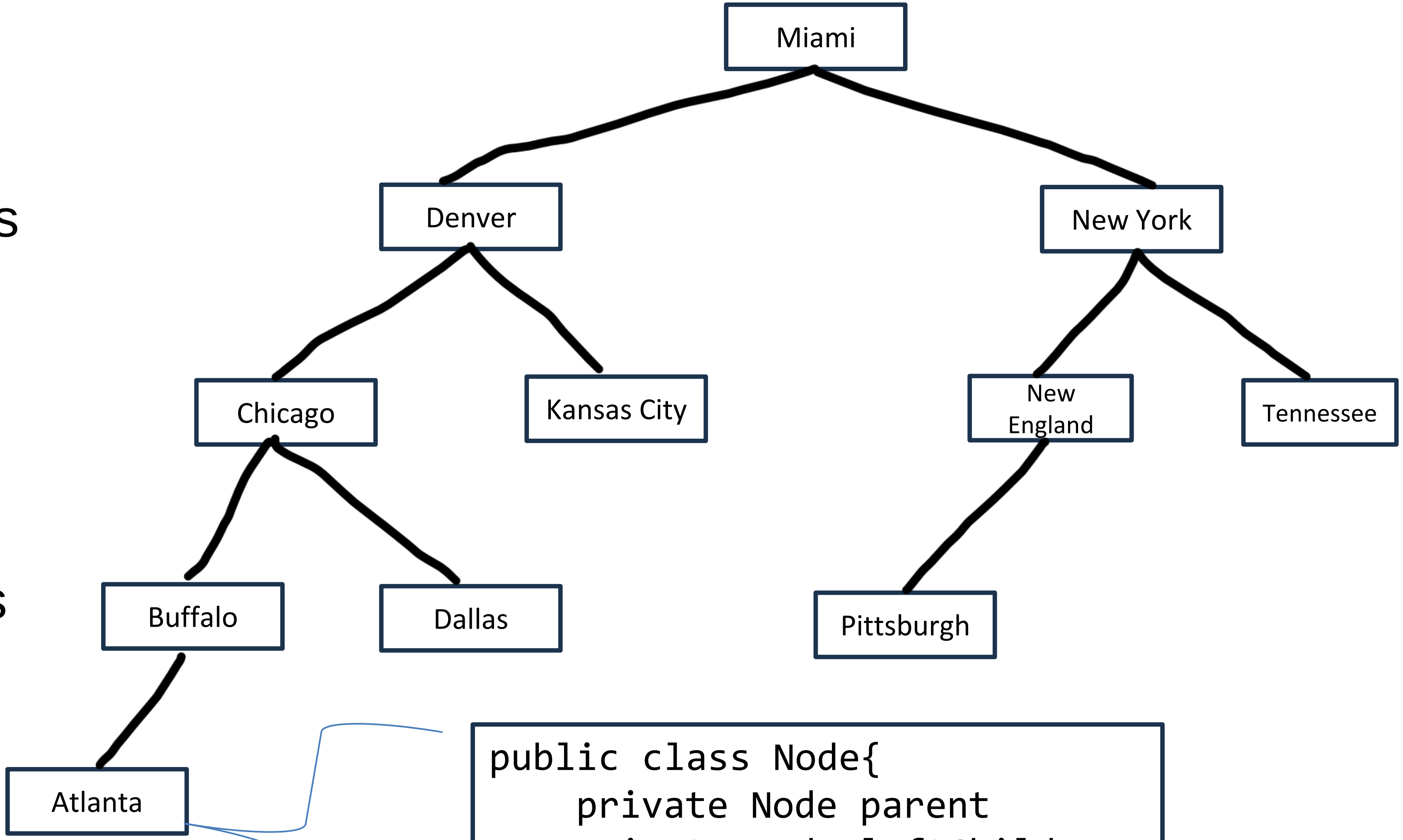
1. Keys should not be mutable

String ✓
int ✓
double ✓

Arrays ✗
Objects ~

Map / Dictionary

Keys		Values
Dallas	→	Cowboys
Chicago	→	Bears
New England	→	Patriots
Denver	→	Broncos
Pittsburgh	→	Steelers
Kansas City	→	Chiefs
Miami	→	Dolphins
Tennessee	→	Titans
New York	→	Giants
Buffalo	→	Bills
Atlanta	→	Falcons

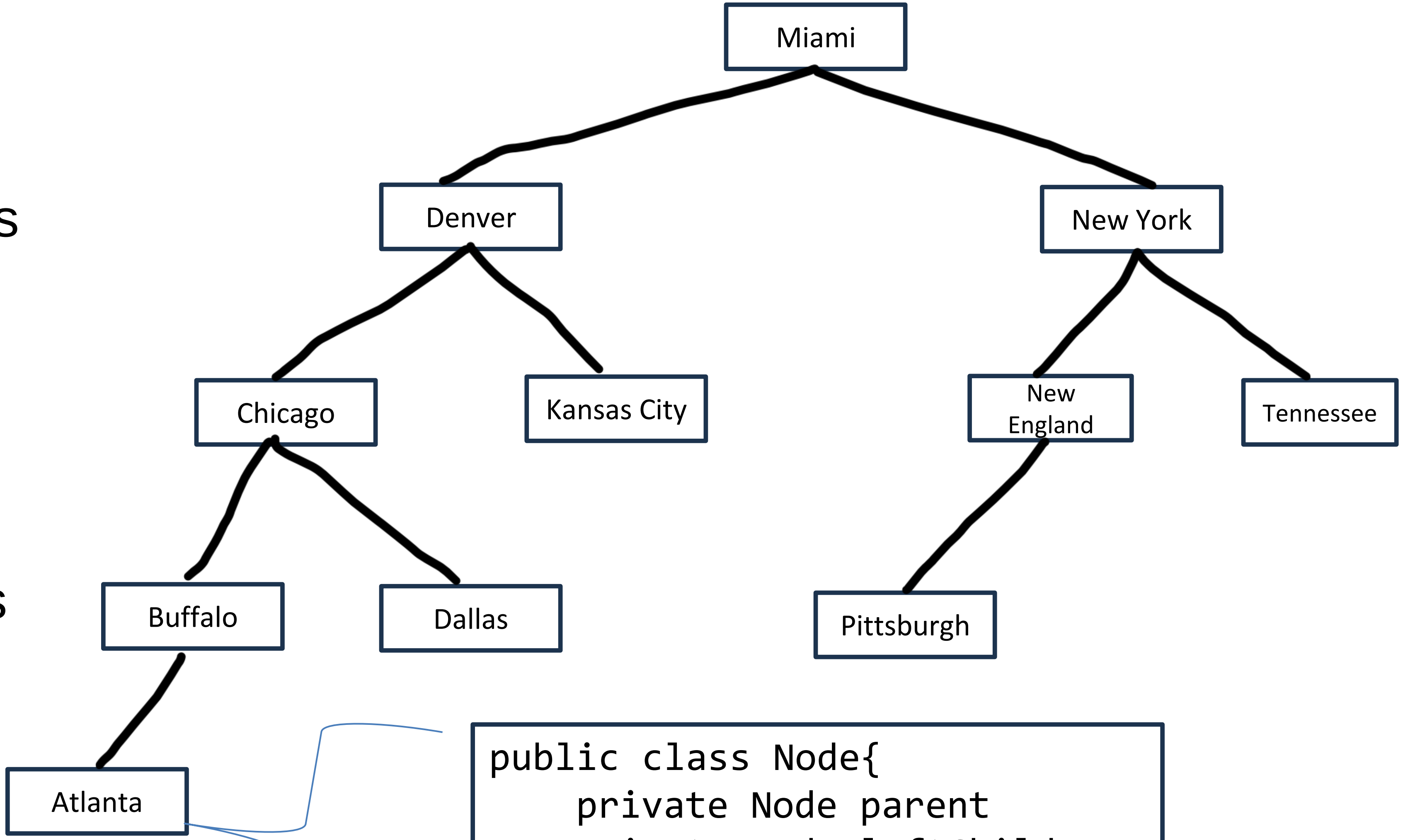


```
public class Node{
    private Node parent
    private Node leftChild
    private Node rightChild

    private String key;
    private String value;
```

Map / Dictionary

Keys		Values
Dallas	→	Cowboys
Chicago	→	Bears
New England	→	Patriots
Denver	→	Broncos
Pittsburgh	→	Steelers
Kansas City	→	Chiefs
Miami	→	Dolphins
Tennessee	→	Titans
New York	→	Giants
Buffalo	→	Bills
Atlanta	→	Falcons

1. Build a BST based on Node key



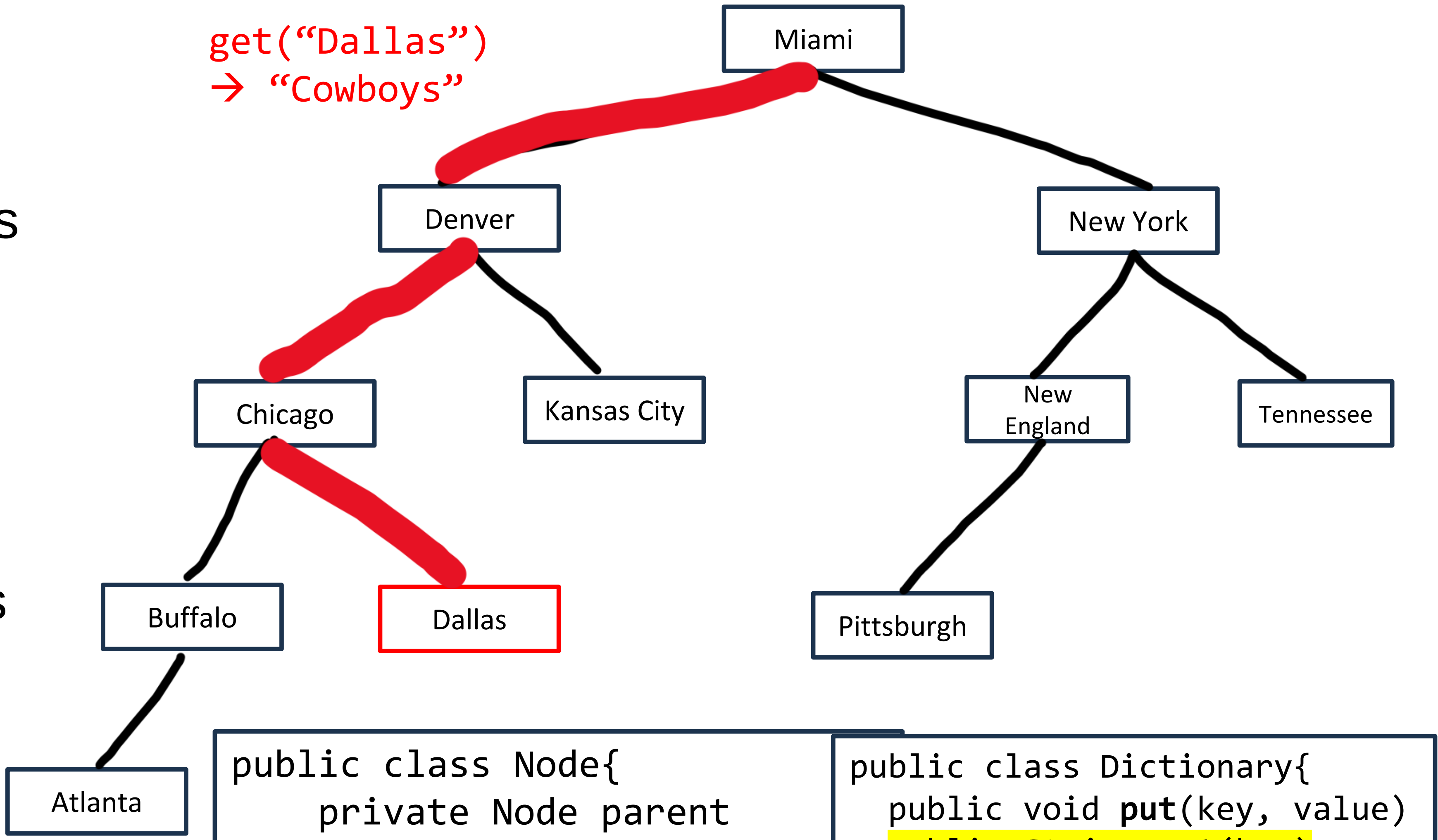
```
public class Node{
    private Node parent
    private Node leftChild
    private Node rightChild

    private String key;
    private String value;
```

Map / Dictionary

Keys		**Values**
Dallas	→	Cowboys
Chicago	→	Bears
New England	→	Patriots
Denver	→	Broncos
Pittsburgh	→	Steelers
Kansas City	→	Chiefs
Miami	→	Dolphins
Tennessee	→	Titans
New York	→	Giants
Buffalo	→	Bills
Atlanta	→	Falcons

1. Build a BST based on Node key
2. Search for value using BST, return value of Node

```
get("Dallas")
→ "Cowboys"
```



```
public class Node{
    private Node parent
    private Node leftChild
    private Node rightChild

    private String key;
    private String value;
```

```
public class Dictionary{
  public void put(key, value)
  public String get(key)
  public void delete(key)
  …
}
```

Map / Dictionary

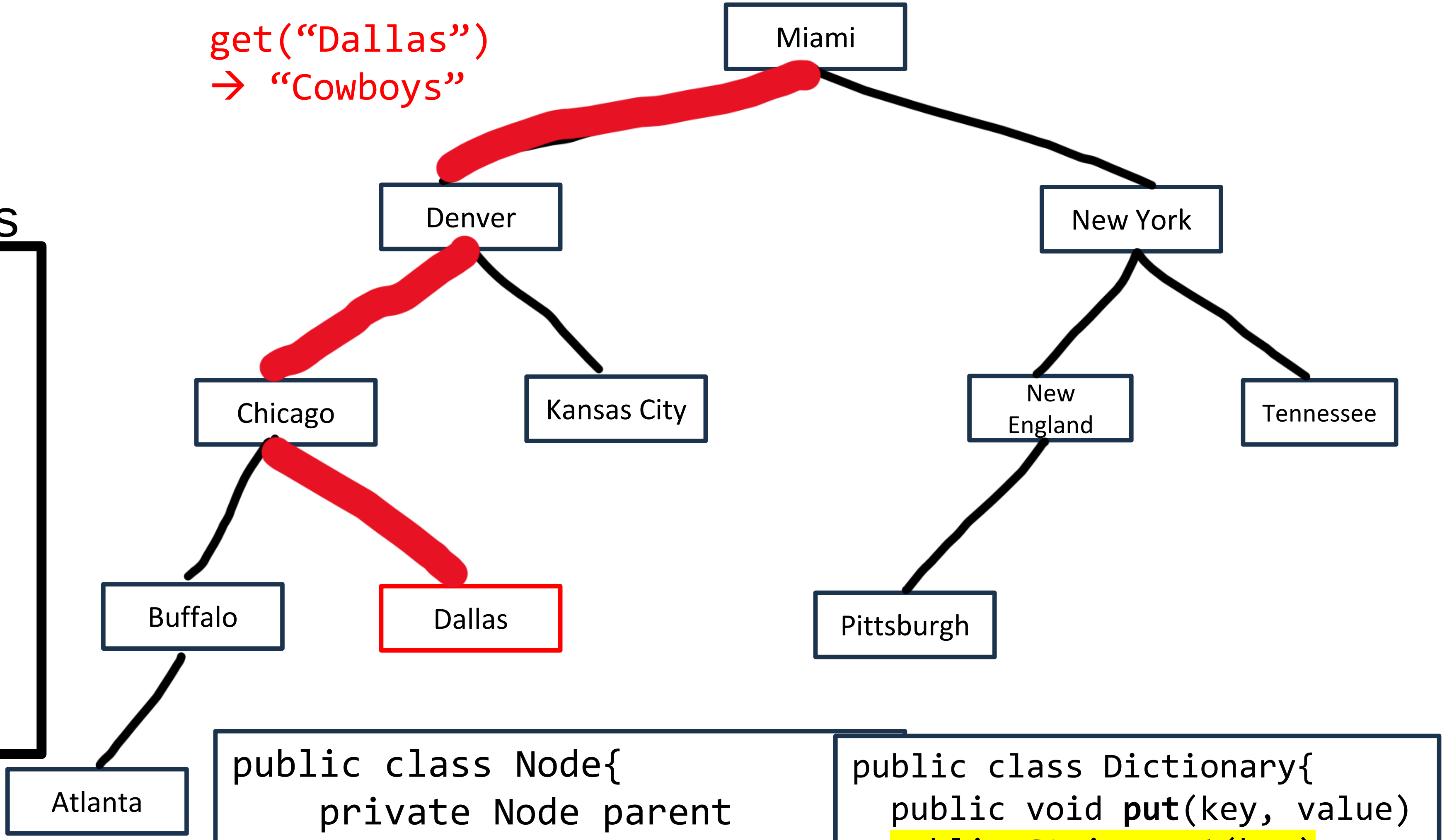
Keys		**Values**
Dallas	→	Cowboys
Chic		
New		
Den		
Pitts		
Kan		
Mia		
Ten		
New		
Buffalo	→	Bills
Atlanta	→	Falcons

Lookup time?

O(log n)

1. Build a BST based on Node key
2. Search for value using BST, return value of Node

`get("Dallas")`
`→ "Cowboys"`



```
public class Node{
    private Node parent
    private Node leftChild
    private Node rightChild

    private String key;
    private String value;
```

```
public class Dictionary{
  public void put(key, value)
  public String get(key)
  public void delete(key)
  …
}
```

Pokedex

Key (Pokemon #)	**Value** (Pokemon)
1	Bulbasaur
2	Ivysaur
3	Venasaur
…	…
98	Krabby
99	Kingler

Pokedex

Key (Pokemon #)	**Value** (Pokemon)
1	Bulbasaur
2	Ivysaur
3	Venasaur
…	…
98	Krabby
99	Kingler

→

Index	
0	(null)
1	Bulbasuar
2	Ivysaur
3	Venasaur
…	…
98	Krabby
99	Kingler

Pokedex

Key (Pokemon #)	**Value** (Pokemon)
1	Bulbasaur
2	Ivysaur
3	Venasaur
…	…
98	Krabby
99	Kingler

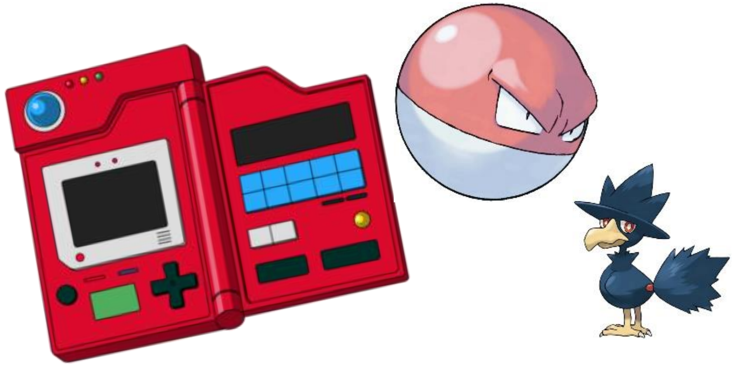
Index	
0	(null)
1	Bulbasuar
2	Ivysaur
3	Venasaur
…	…
98	Krabby
99	Kingler

Lookup time?

O(1) **!!**

Pokedex

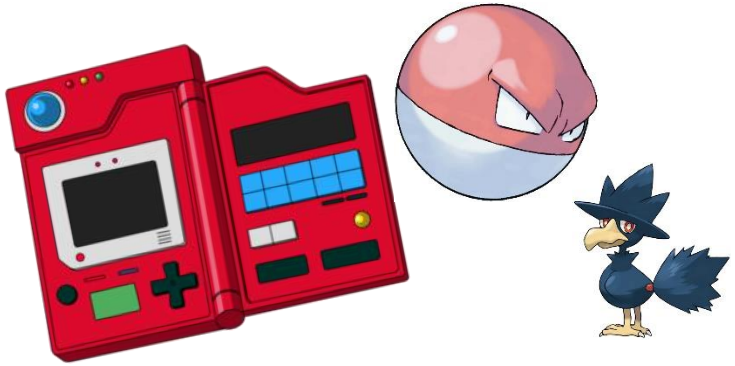
Key (Pokemon #)	Value (Pokemon)
100	Voltorb
101	Electrode
102	Exeggcute
…	…
198	Murkrow
199	Slowking



Index	
0	null
…	…
99	null
100	Voltorb
101	Electrode
102	Exeggcute
103	Exeggutor
…	…
198	Murkrow
199	Slowking

Pokedex

Key (Pokemon #)	Value (Pokemon)
100	Voltorb
101	Electrode
102	Exeggcute
…	…
198	Murkrow
199	Slowking

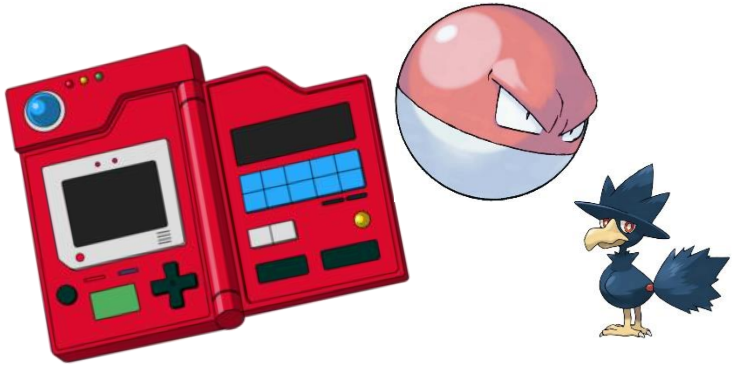


Lots of wasted space that won't be used… not ideal

Index	
0	null
…	…
99	null
100	Voltorb
101	Electrode
102	Exeggcute
103	Exeggutor
…	…
198	Murkrow
199	Slowking

Pokedex

Key (Pokemon #)	**Value** (Pokemon)
100	Voltorb
101	Electrode
102	Exeggcute
…	…
198	Murkrow
199	Slowking

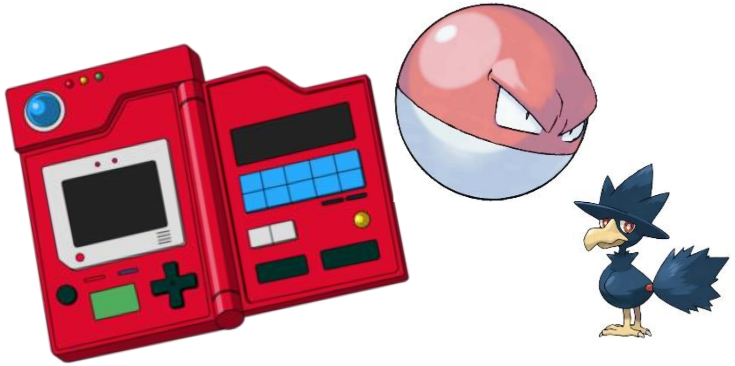


Index	
0	Voltorb
1	Electrode
2	Exeggcute
3	Exeggutor
…	…
98	Murkrow
99	Slowking

Pokedex

Key (Pokemon #)	**Value** (Pokemon)
100	Voltorb
101	Electrode
102	Exeggcute
…	…
198	Murkrow
199	Slowking

Index	
0	Voltorb
1	Electrode
2	Exeggcute
3	Exeggutor
…	…
98	Murkrow
99	Slowking



What array index does Pokemon number **x** go into ?

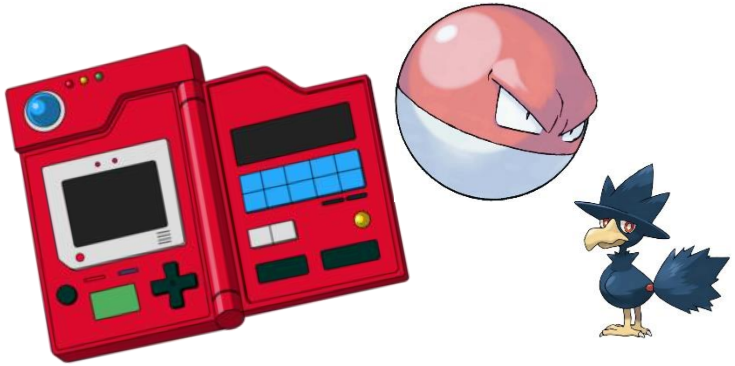
Pokedex

Key (Pokemon #)	**Value** (Pokemon)
100	Voltorb
101	Electrode
102	Exeggcute
…	…
198	Murkrow
199	Slowking

Index	
0	Voltorb
1	Electrode
2	Exeggcute
3	Exeggutor
…	…
98	Murkrow
99	Slowking

X % 100

What array index does Pokemon number x go into ?



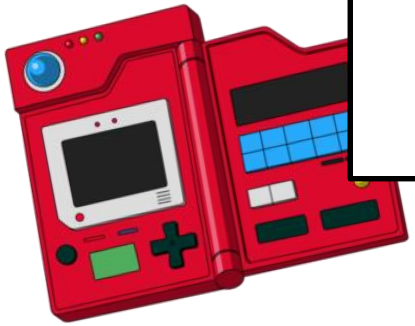
Pokedex

Key (Pokemon #)	Value
100	V
101	E
102	E
…	..
198	M
199	S

Index

% - modulo operator

a % b = remainder when a is divided by b



Pokemon number x go into ?

Pokedex

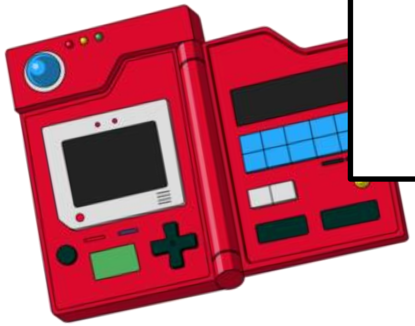
Key (Pokemon #)	Value
100	V
101	E
102	E
…	..
198	M
199	S

Index

% - modulo operator

a % b = remainder when a is divided by b

12 % 7 =



Pokemon number x go into ?

Pokedex

Key (Pokemon #)	**Value**
100	V
101	E
102	E
…	..
198	M
199	S

Index

% - modulo operator

a % b = remainder when a is divided by b

12 % 7 = 5

7 % 12 =

Pokemon number x go into ?

Pokedex

Key (Pokemon #)	Value
100	V
101	E
102	E
…	..
198	M
199	S

Index

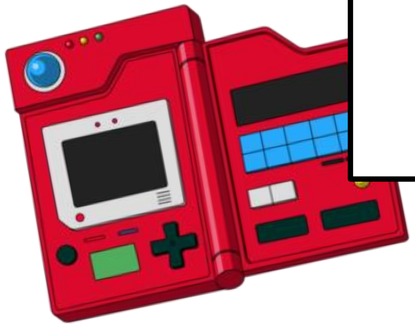
% - modulo operator

a % b = remainder when a is divided by b

12 % 7 = 5

7 % 12 = 7

132 % 100 =



Pokemon number x go into ?

Pokedex

Key (Pokemon #)	**Value**
100	V
101	E
102	E
…	..
198	M
199	S

Index

% - modulo operator

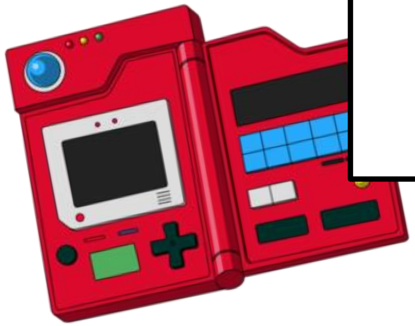
a % b = remainder when a is divided by b

12 % 7 = 5

7 % 12 = 7

132 % 100 = 32

100 % 100 =



Pokemon number x go into ?

Pokedex

Key (Pokemon #)	**Value**
100	V
101	E
102	E
…	..
198	M
199	S

Index

% - modulo operator

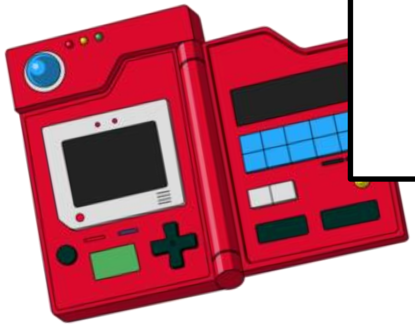
a % b = remainder when a is divided by b

12 % 7 = 5

7 % 12 = 7

132 % 100 = 32

100 % 100 = 0



Pokemon number x go into ?

Pokedex

Index

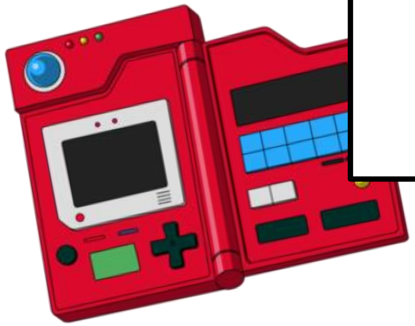
Key (Pokemon #)	Value
100	V
101	E
102	E
…	..
198	M
199	S

% - modulo operator

a % b = remainder when a is divided by b

X % 100

Possible output values?



Pokemon number x go into ?

Pokedex

Key (Pokemon #)	Value
100	V
101	E
102	E
…	..
198	M
199	S

Index

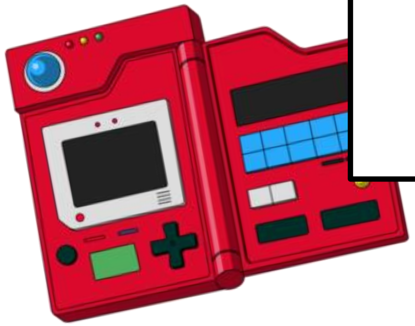
% - modulo operator

a % b = remainder when a is divided by b

X % 100

Possible output values?

0, 1, 2, 3, … , 98, 99



Pokemon number x go into ?

Pokedex

Key
(Pokemon #)

Value

Index

100 V
101 E
102 E
… ..
198 M
199 S

% - modulo operator

a % b = remainder when a is divided by b

X % 100

Possible output values?

0, 1, 2, 3, … , 98, 99

All array spots are used!

Pokemon number x go into ?

Pokedex

Key (Pokemon #)	Value (Pokemon)
100	Voltorb
101	Electrode
102	Exeggcute
…	…
198	Murkrow
199	Slowking

Index	
0	Voltorb
1	Electrode
2	Exeggcute
3	Exeggutor
…	…
98	Murkrow
99	Slowking

Why 100?

X % 100

What array index does Pokemon number x go into ?

Pokedex

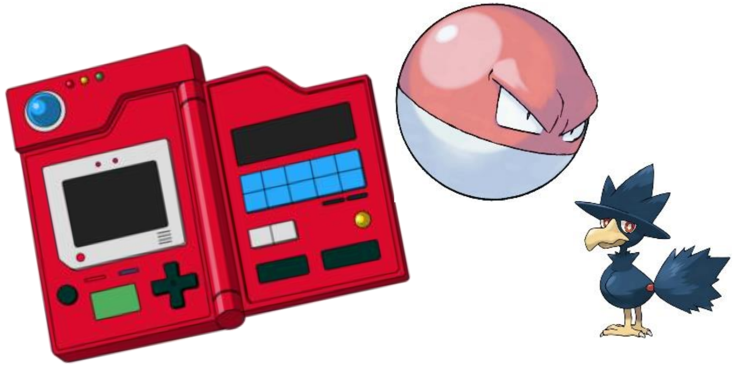
Key (Pokemon #)	**Value** (Pokemon)
100	Voltorb
101	Electrode
102	Exeggcute
…	…
198	Murkrow
199	Slowking

Index	
0	Voltorb
1	Electrode
2	Exeggcute
3	Exeggutor
…	…
98	Murkrow
99	Slowking

X % 100

This is our (simple) **hash function**

Hash Function: Function that translates keys into array indices (hash values)



Pokedex

Key (Pokemon #)	**Value** (Pokemon)
100	Voltorb
101	Electrode
102	Exeggcute
…	…
198	Murkrow
199	Slowking

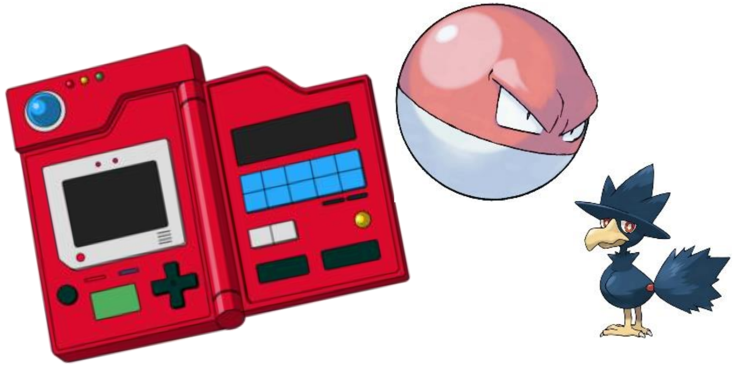
Index	
0	Voltorb
1	Electrode
2	Exeggcute
3	Exeggutor
…	…
98	Murkrow
99	Slowking

X % 100

This is our (simple) **hash function**

Can accept any arbitrary sized input!

Hash Function: Function that translates keys into array indices (hash values)



Pokedex

Key (Pokemon #)	**Value** (Pokemon)
100	Voltorb
101	Electrode
102	Exeggcute
…	…
198	Murkrow
199	Slowking

Index	
0	Voltorb
1	Electrode
2	Exeggcute
3	Exeggutor
…	…
98	Murkrow
99	Slowking

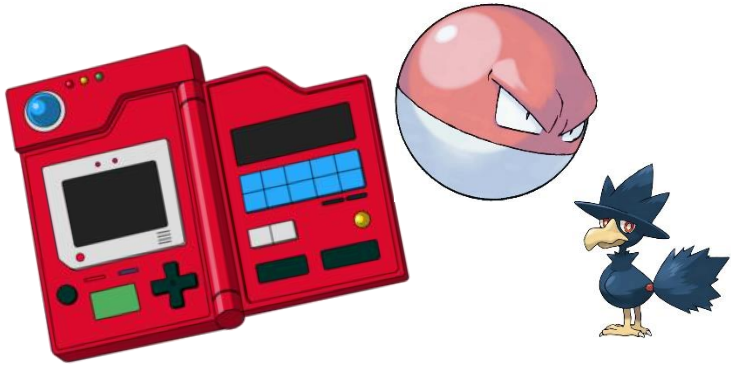
Runs in O(1) time

X % 100

This is our (simple) **hash function**

Can accept any arbitrary sized input!

Hash Function: Function that translates keys into array indices (hash values)



Pokedex

Key (Pokemon #)	**Value** (Pokemon)
100	Voltorb
101	Electrode
102	Exeggcute
…	…
198	Murkrow
199	Slowking

X % 100

What could possibly go wrong?

Index	
0	Voltorb
1	Electrode
2	Exeggcute
3	Exeggutor
…	…
98	Murkrow
99	Slowking

Pokedex

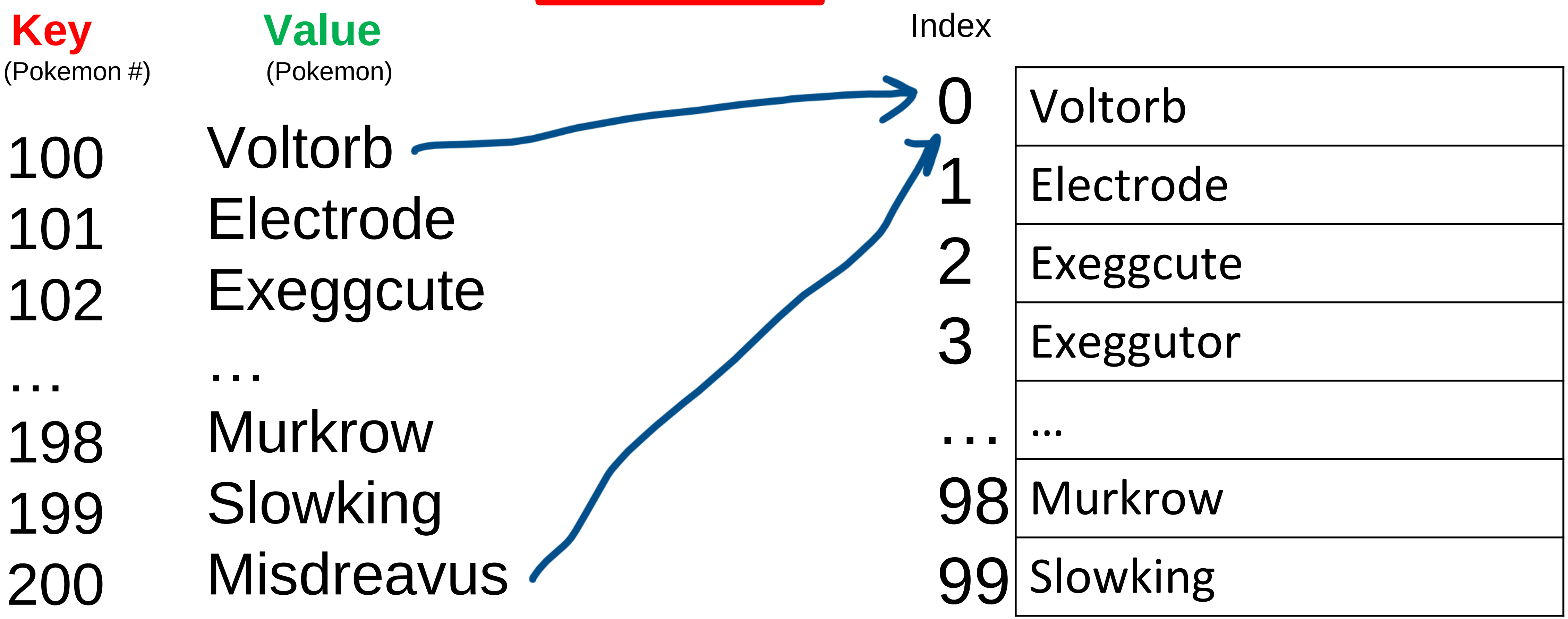
Key (Pokemon #)	**Value** (Pokemon)
100	Voltorb
101	Electrode
102	Exeggcute
…	…
198	Murkrow
199	Slowking
200	Misdreavus

X % 100

Index	
0	Voltorb
1	Electrode
2	Exeggcute
3	Exeggutor
…	…
98	Murkrow
99	Slowking

Pokedex

X % 100



Key (Pokemon #)	**Value** (Pokemon)
100	Voltorb
101	Electrode
102	Exeggcute
…	…
198	Murkrow
199	Slowking
200	Misdreavus

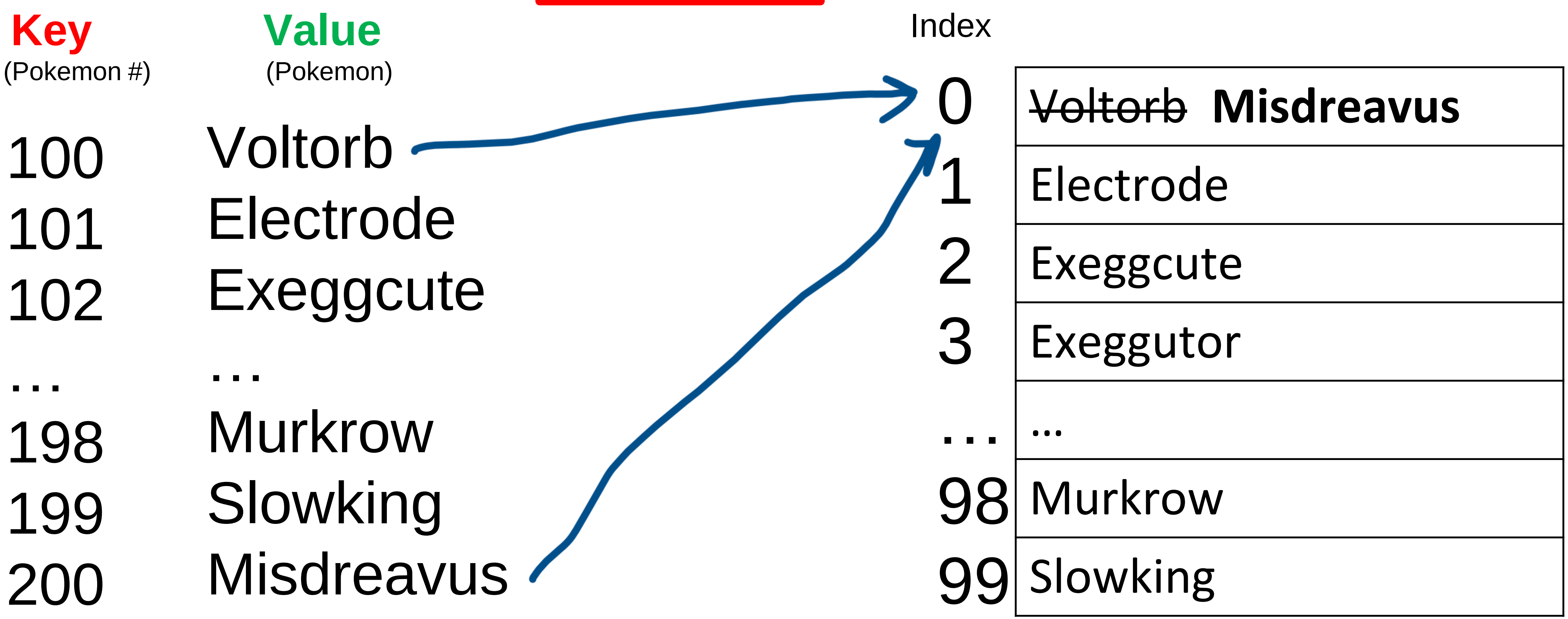
Index	
0	Voltorb
1	Electrode
2	Exeggcute
3	Exeggutor
…	…
98	Murkrow
99	Slowking

We have two keys that map to the same "bucket" (array index)

→ A **collision**

Pokedex

X % 100



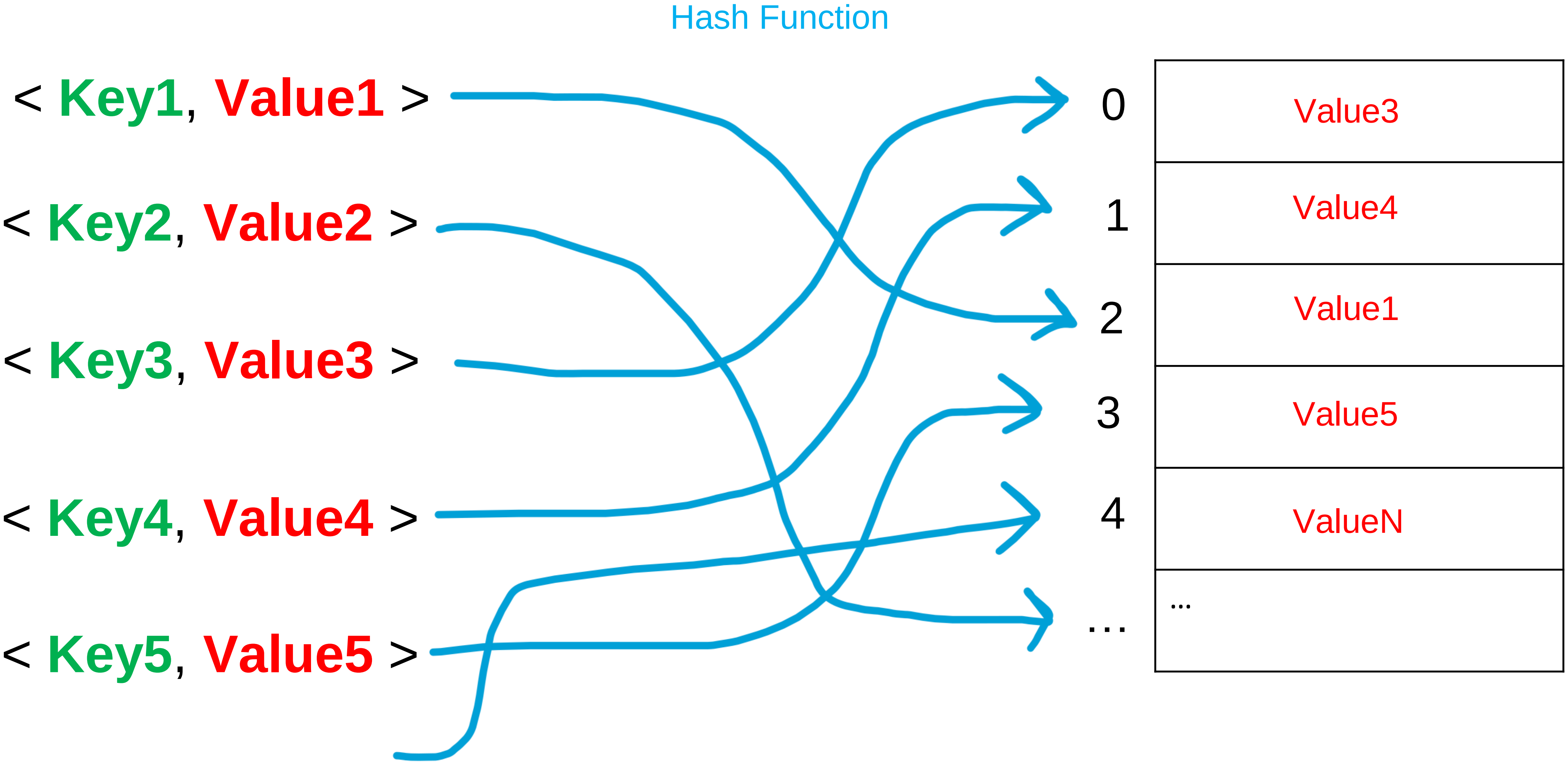
Key (Pokemon #)	**Value** (Pokemon)
100	Voltorb
101	Electrode
102	Exeggcute
…	…
198	Murkrow
199	Slowking
200	Misdreavus

Index	
0	~~Voltorb~~ **Misdreavus**
1	Electrode
2	Exeggcute
3	Exeggutor
…	…
98	Murkrow
99	Slowking

We have two keys that map to the same “bucket” (array index)

→ A **collision**

Hash Tables 101



Hash Function

< **Key1**, **Value1** >

< **Key2**, **Value2** >

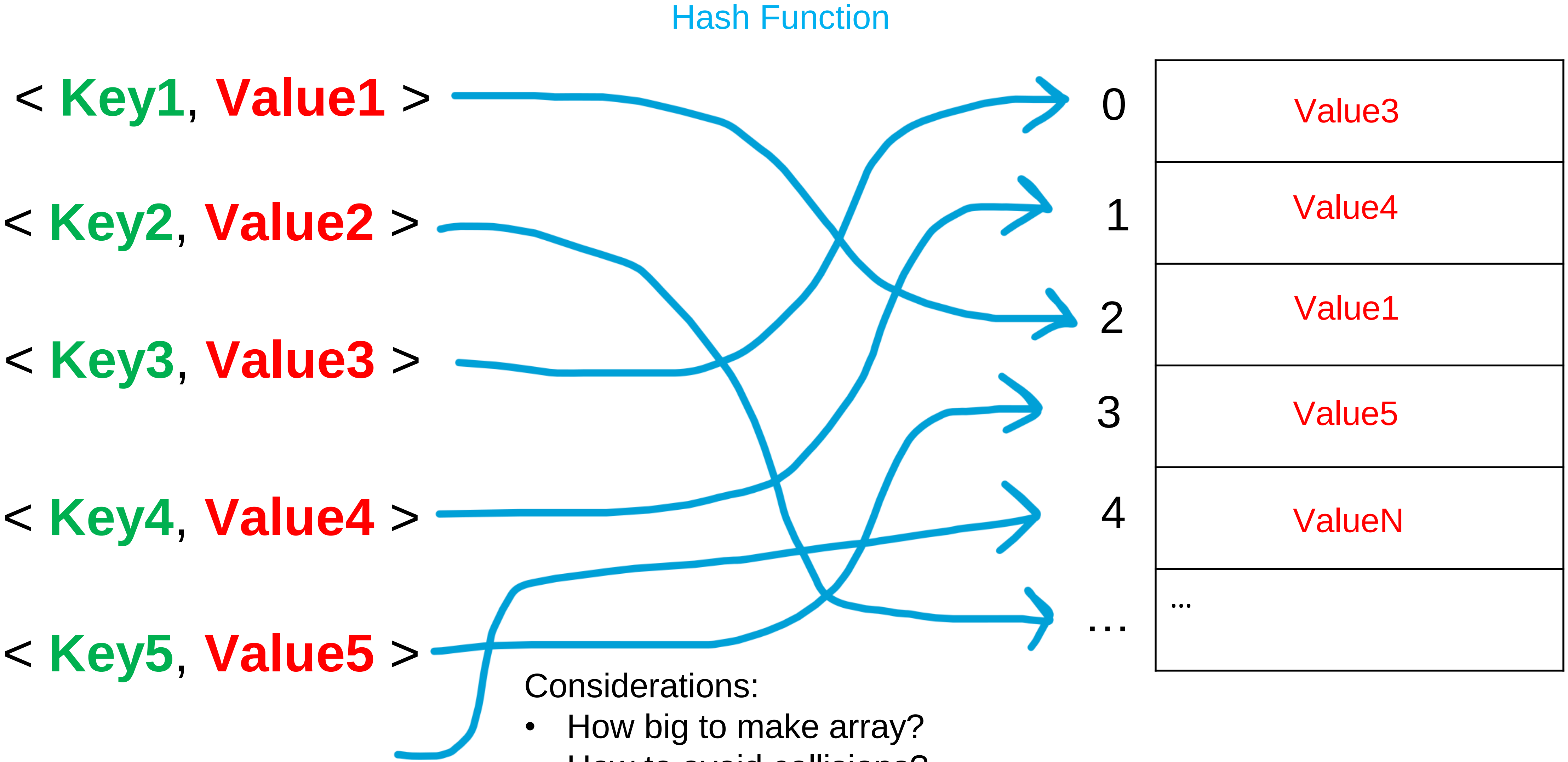
< **Key3**, **Value3** >

< **Key4**, **Value4** >

< **Key5**, **Value5** >

0	Value3
1	Value4
2	Value1
3	Value5
4	ValueN
…	…

Hash Tables 101



Hash Function

< **Key1**, **Value1** >

< **Key2**, **Value2** >

< **Key3**, **Value3** >

< **Key4**, **Value4** >

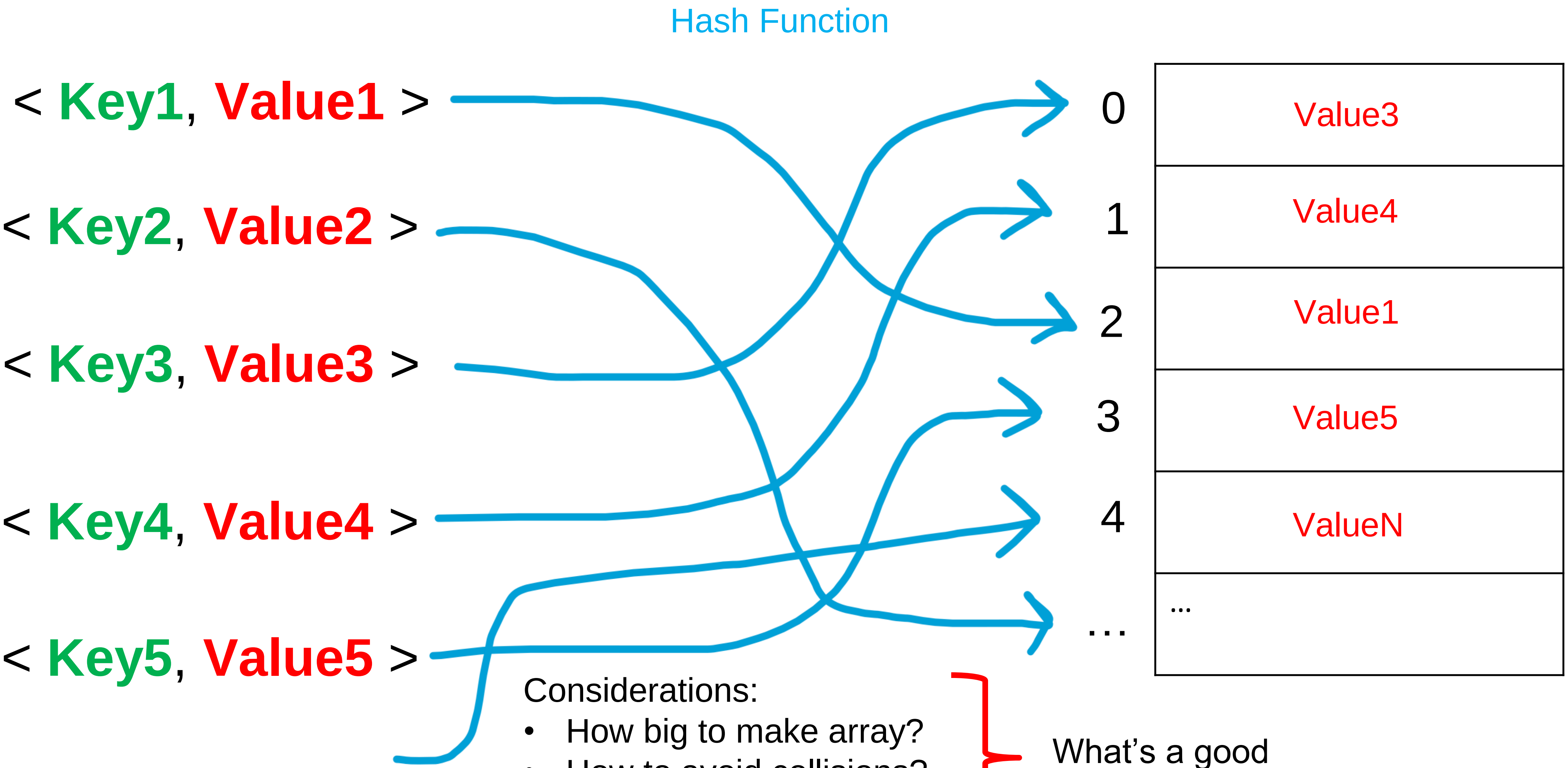
< **Key5**, **Value5** >

0	Value3
1	Value4
2	Value1
3	Value5
4	ValueN
…	…

Considerations:

- How big to make array?
- How to avoid collisions?
- How to handle collisions?

Hash Tables 101



Hash Function

< **Key1**, **Value1** >

< **Key2**, **Value2** >

< **Key3**, **Value3** >

< **Key4**, **Value4** >

< **Key5**, **Value5** >

0	Value3
1	Value4
2	Value1
3	Value5
4	ValueN
…	…

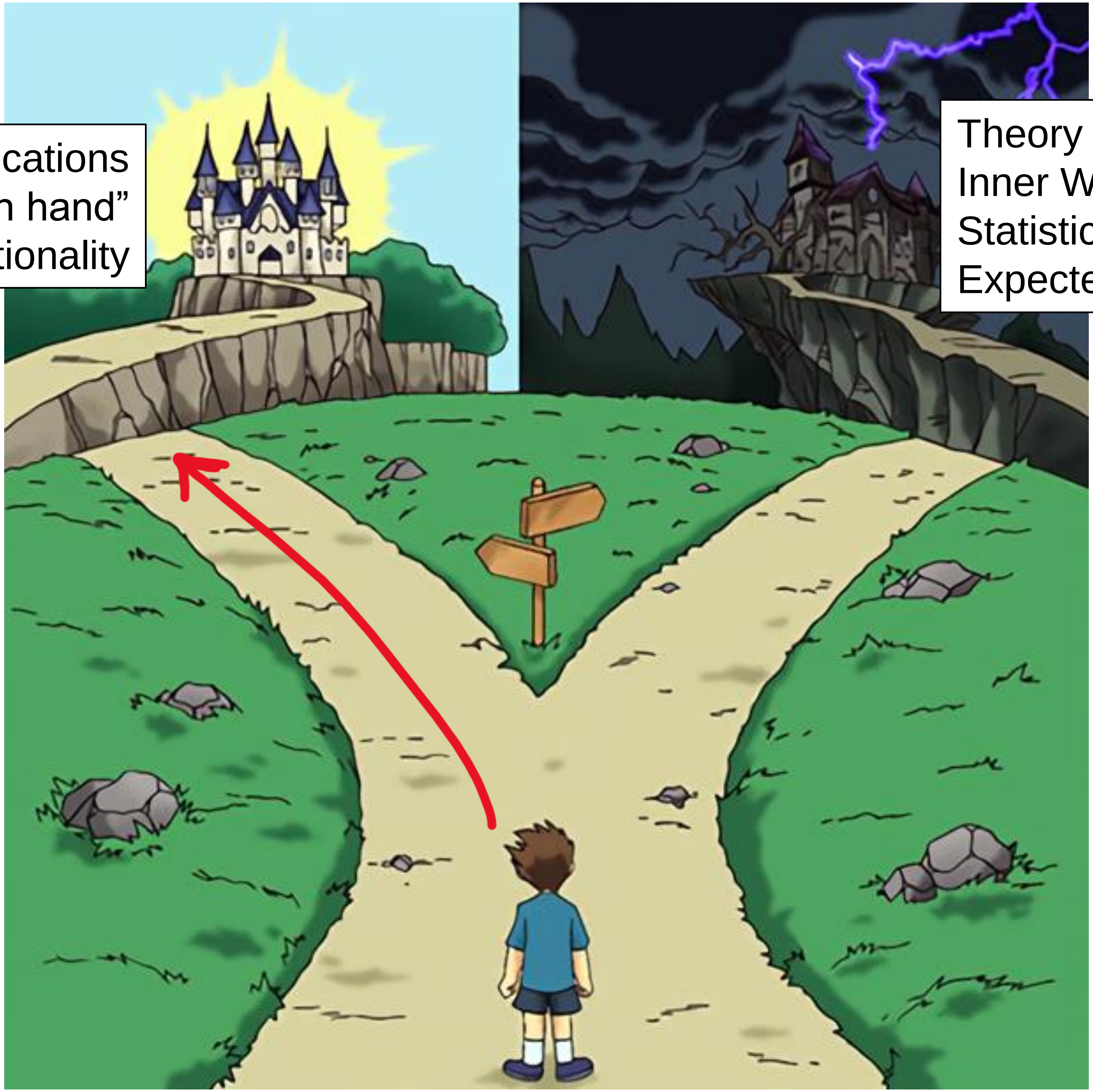
Considerations:

- How big to make array?
- How to avoid collisions?
- How to handle collisions?

What's a good hash function?

Hash Tables 101

Hash Tables 101



Applications
“Tools in hand”
Java Functionality

Theory
Inner Workings of Hash Functions
Statistical Likelihood
Expected Performance

Hash Tables are probably the most useful thing you learn in this class

Hash Tables 101

I use HashMap, HashTable, and Dictionary interchangeably, but there are very small differences between these

Let’s build a Hash Table for a **Student Database**

Keys need to be unique, what could we use for a key ?

Hash Tables 101

I use HashMap, HashTable, and Dictionary interchangeably, but there are very small differences between these

Let’s build a Hash Table for a **Student Database**

Keys need to be unique, what could we use for a key? Student ID!

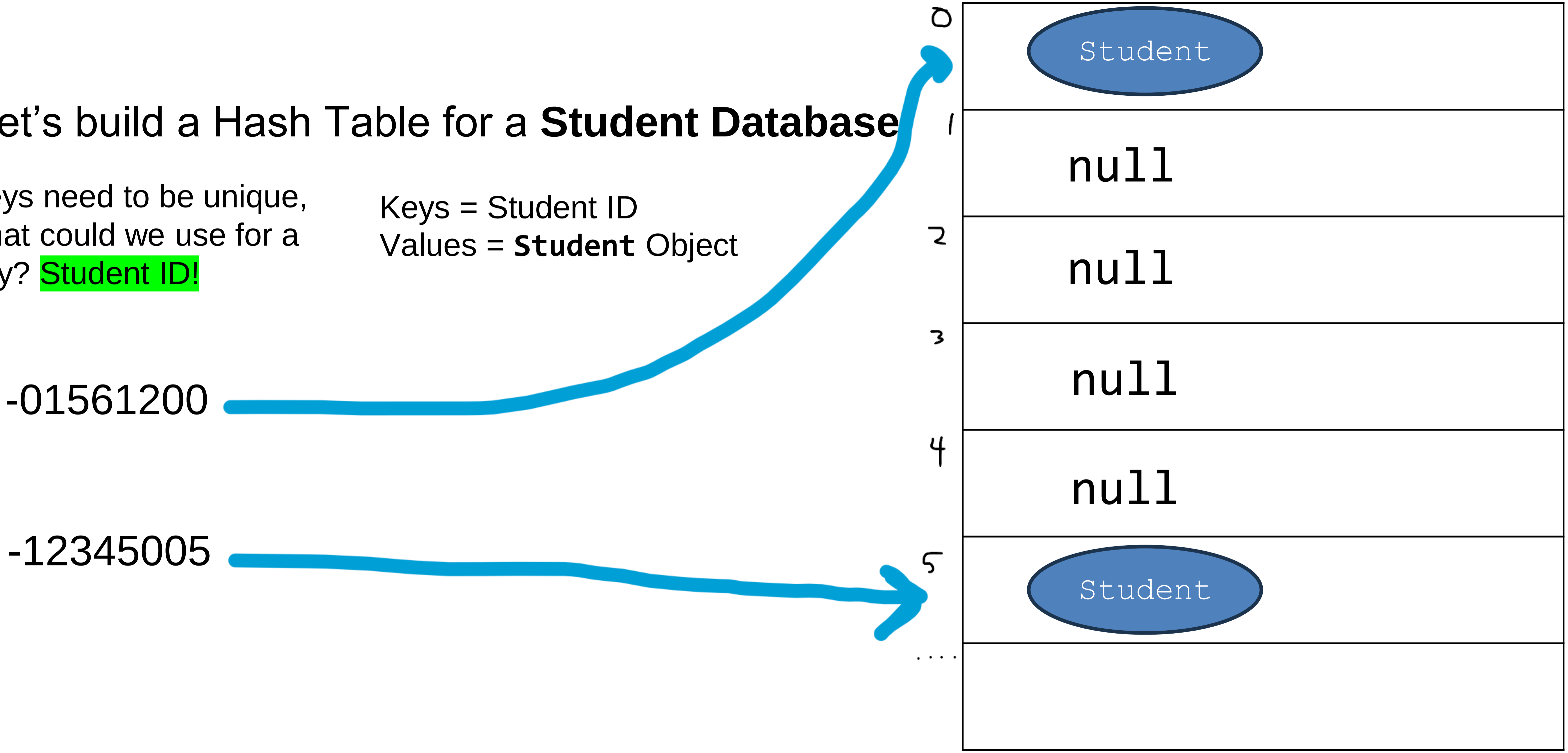
Hash Tables 101

I use HashMap, HashTable, and Dictionary interchangeably, but there are very small differences between these

Let’s build a Hash Table for a **Student Database**

Keys need to be unique, what could we use for a key? Student ID!

Keys = Student ID
Values = **`Student`** Object



-01561200

-12345005