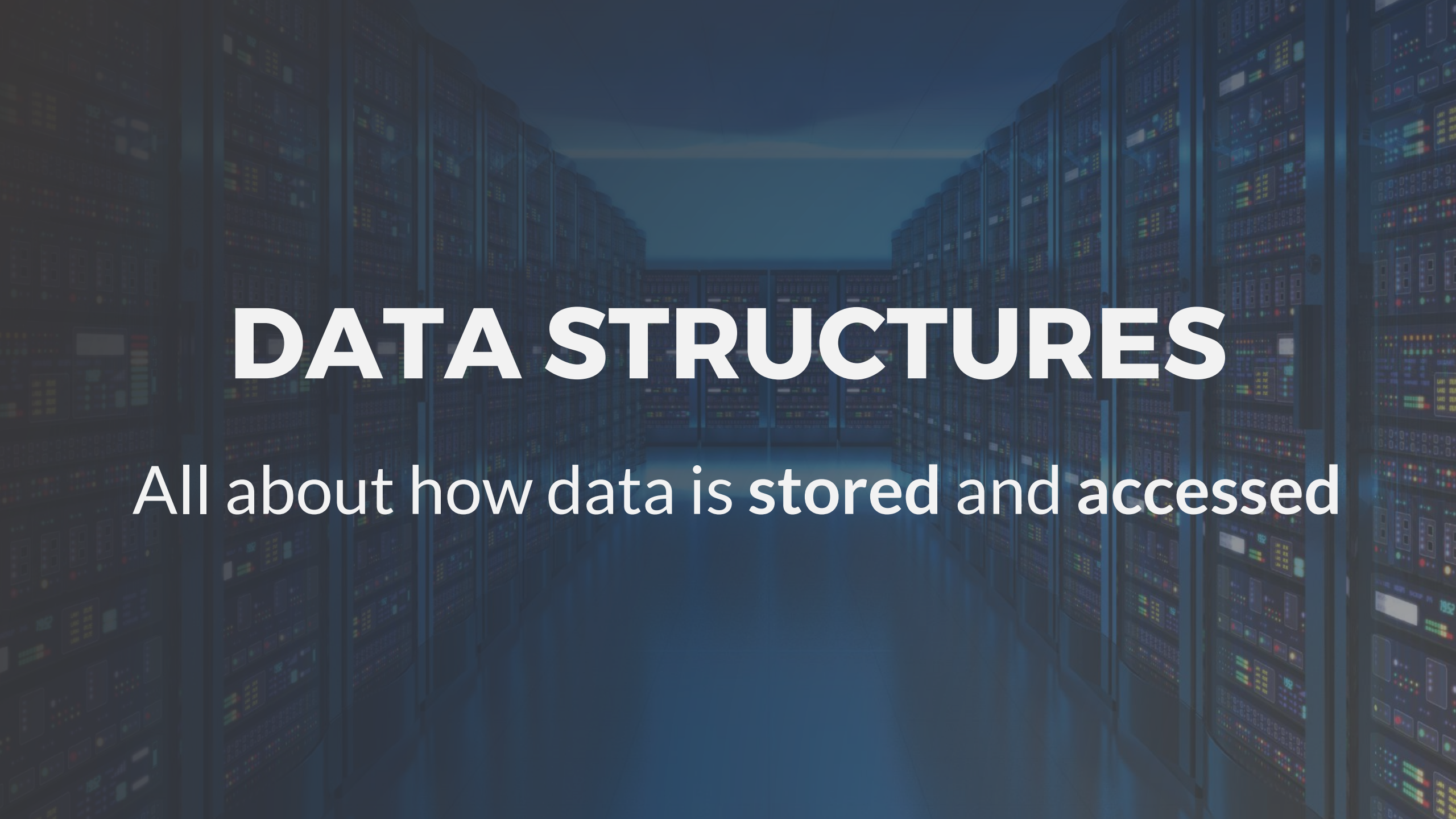
Data Structures



DATA STRUCTURES

All about how data is **stored** and **accessed**

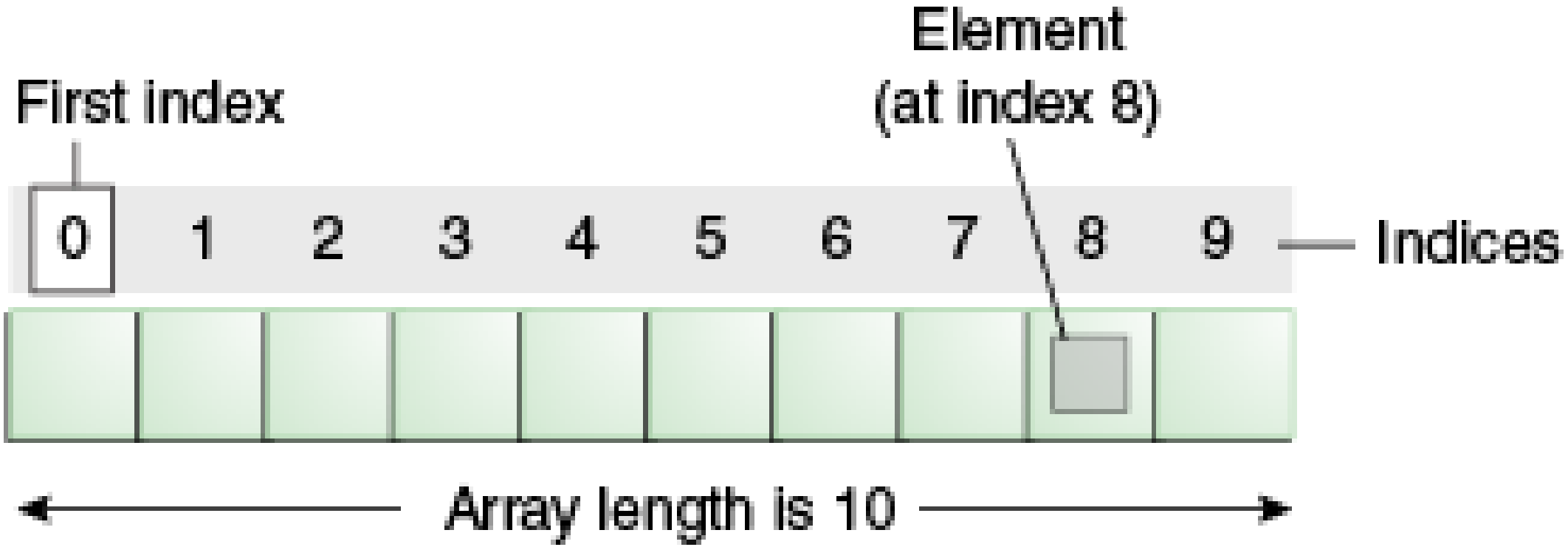
ARRAYS

What is an Array?

Definition: ordering by index of similar types of elements

Arrays hold multiple values of same data type into one variable, with each value identified by an index

ARRAYS


Element
(at index 8)
First index
0
1
2
3
4
5
6
7
8
9
Indices
Array length is 10

COLLECTIONS FRAMEWORK

What is Collections in Java?

Definition: set of classes and interfaces that handles common and complex data structures dynamically

Arrays

Fixed size

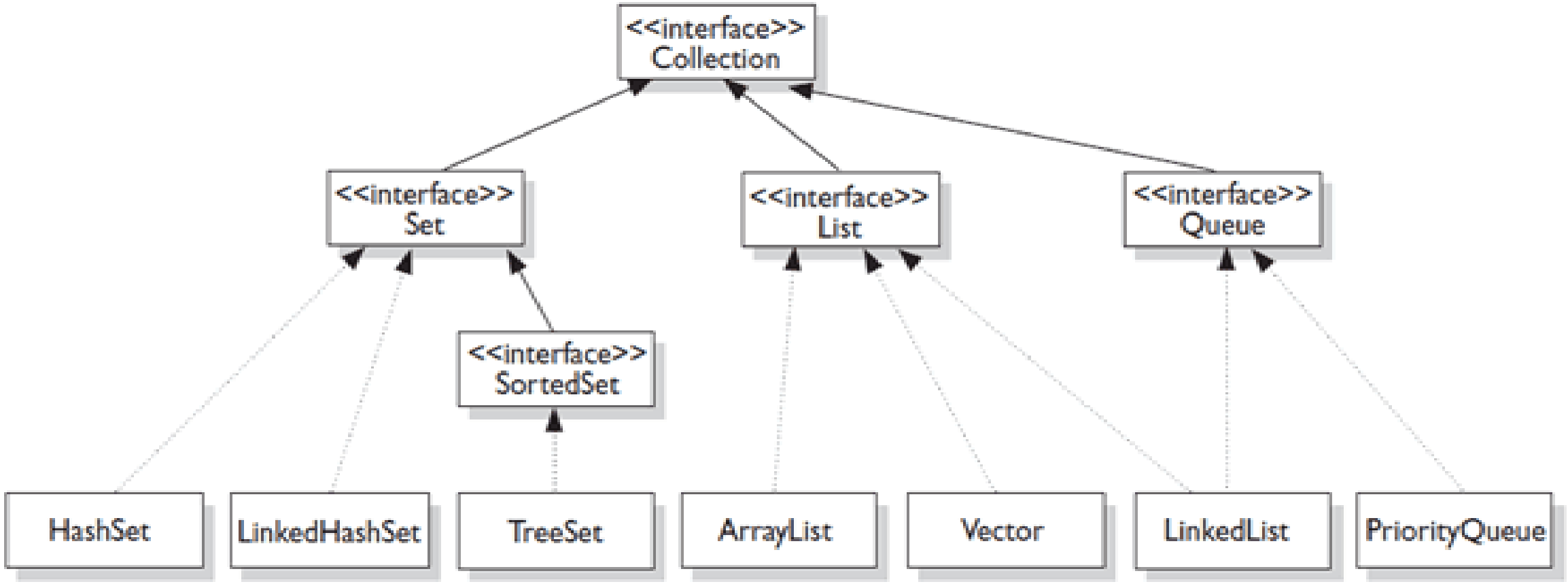
Fixed data type

Collections

Dynamic size

Multiple data types

The Collections Framework



List: Basic Data Structure

`.add()` `.remove()`

1. **ArrayList**
2. **LinkedList**

```
ListType<Type> name = new ListType<Type>()
```

1. ArrayList

```
ArrayList<Type> name = new ArrayList<Type>()
```

Methods:

.add()

.get()

.size()

.toString()

.remove() - removes element

.set() - defines element

1. ArrayList

```
ArrayList<Type> name = new ArrayList<Type>()
```

Used For:

- Simple Array
- Adding or removing elements to end of the list

Features:

- Quick addition
- Quick retrieval
- Slow removal
- Slow when adding/removing in middle of list

2. LinkedList

```
LinkedList<Type> name = new LinkedList<Type >()
```

Methods:

.pop() - retrieves and removes

.push() - adds add first element

.peak() - looks at first element

2. LinkedList

```
LinkedList<Type> name = new LinkedList<Type >()
```

Used For:

- Simple Arrays
- Adding or removing elements in the middle of the list

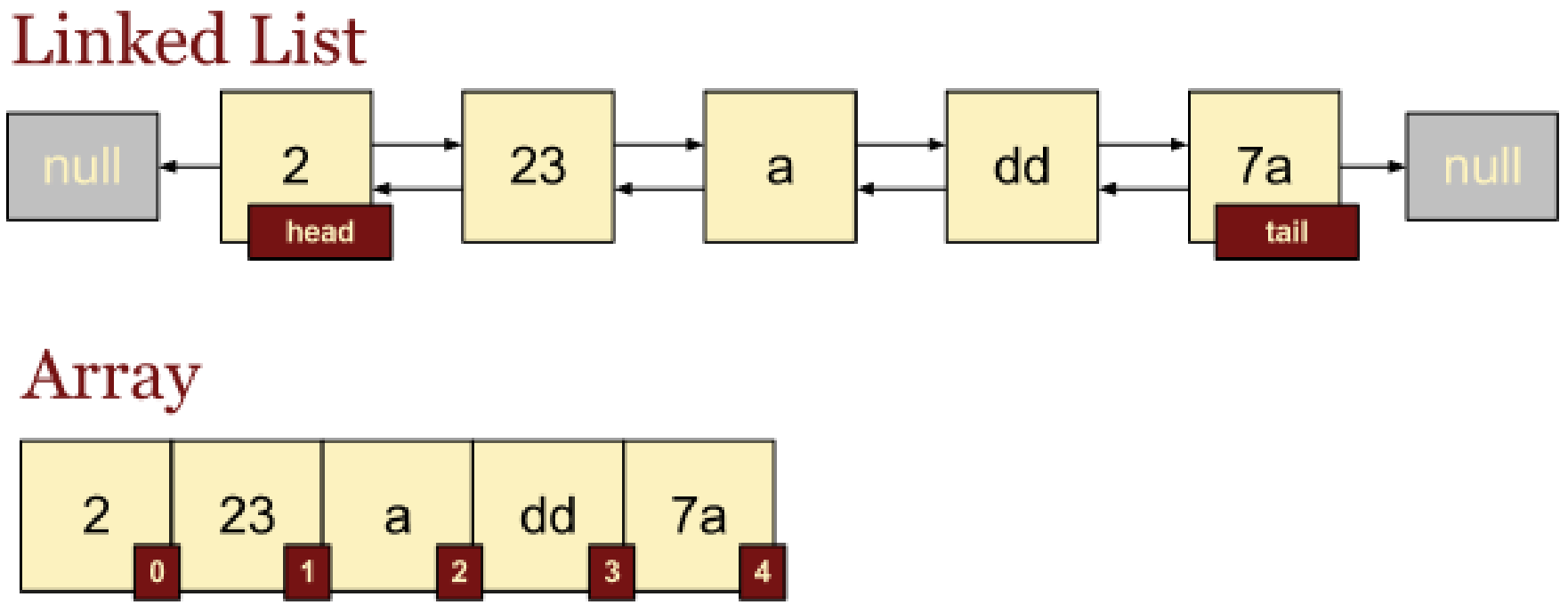
Features:

- Quick retrieval
- Quick removal
- Quick addition to middle of the list

ArrayList vs LinkedList

ArrayList manages arrays internally

LinkedList manages elements by reference to other elements



Maps: Key-Value Pairs

`.put(key,value)` `.get(key)`

1. **HashMap: does not retain order**
2. **LinkedHashMap: retains order**
3. **TreeMap: retains natural order (numerical / alphabetical)**

```
Map<Type1,Type2> name = new MapType<Type1,Type2>()
```

Ex: `Map<Integer,String> map1 = new HashMap<Integer,String>()`

Sets: Unique Lists

`.add()` `.remove()`

1. **HashSet: does not retain order**
2. **LinkedHashSet: retains order**
3. **TreeSet: retains natural order (numerical / alphabetical)**

```
Set<Type> name = new SetType<Type>()
```

Ex: `Map<Integer> map1 = new HashSet<Integer>()`