IMMUTABLE DATA STRUCTURES

Don’t you ever change!

TRAJECTORY

- What are immutable data structures?
- Why use immutable data structures?
- Immutable data structures and JavaScript

IMMUTABILITY: STATE CANNOT BE MODIFIED AFTER CREATION

IMMUTABLE DS

- What if, instead of mutating an array, using push or pop behaved like map or filter, and returned a new array?

- What if assigning a new key-value pair in an object was an operation that returned a new copy of the object?

```
let obj = {a: 'someData'}

lolFunction(obj)

console.log(obj) // what are we going to get?
```

```
const obj = {a: 'someData'}

const newObj = lolFunction(obj)

// Now I know that if I want to use obj, I'll use obj
// If I want newObj, I'll use newObj!
```

ADVANTAGES

- Predictability
- Referential transparency
- Easier to debug
- Can treat a collection of data like a single value

DISADVANTAGES

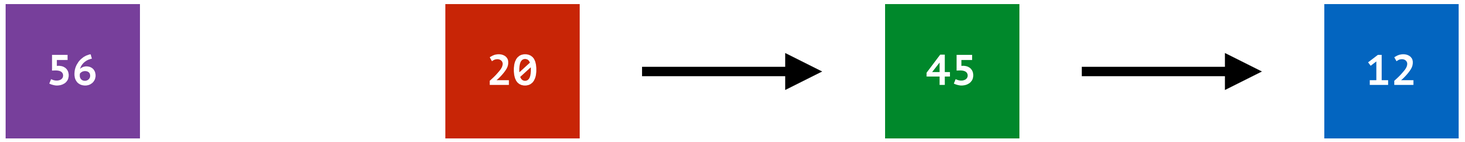
- **Requires more space**
  - Workaround: implement **structural sharing**
  - Still some dependency on garbage collection in your environment

- **Sometimes less performant, depending on what you're trying to do**

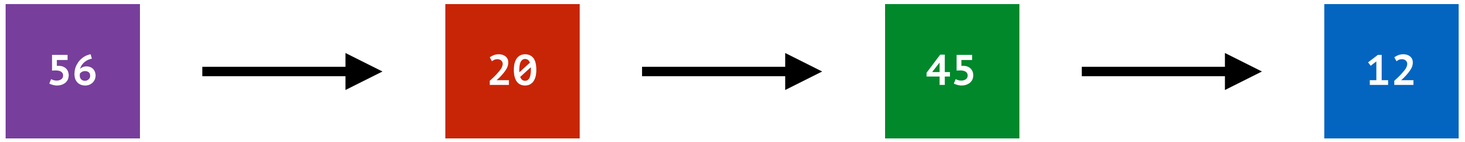
IMMUTABLE LINKED LIST OPERATIONS & STRUCTURAL SHARING

PREPEND

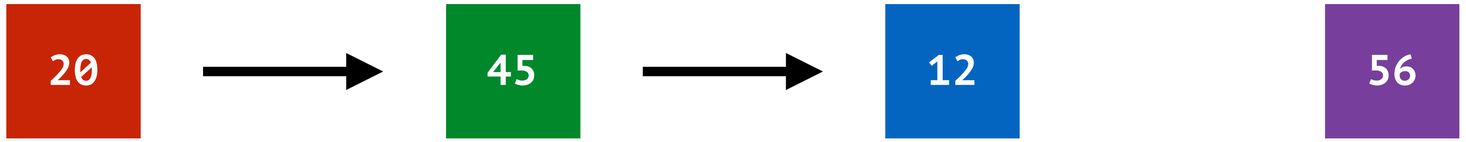
PREPEND



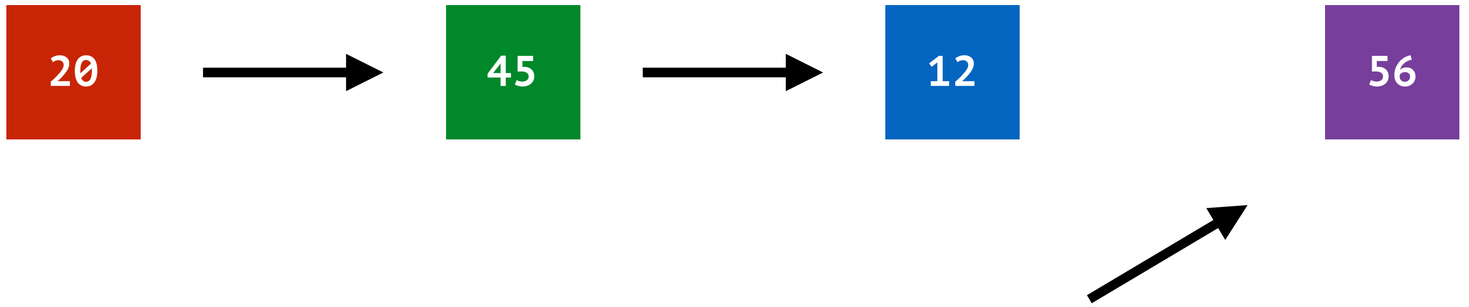
PREPEND



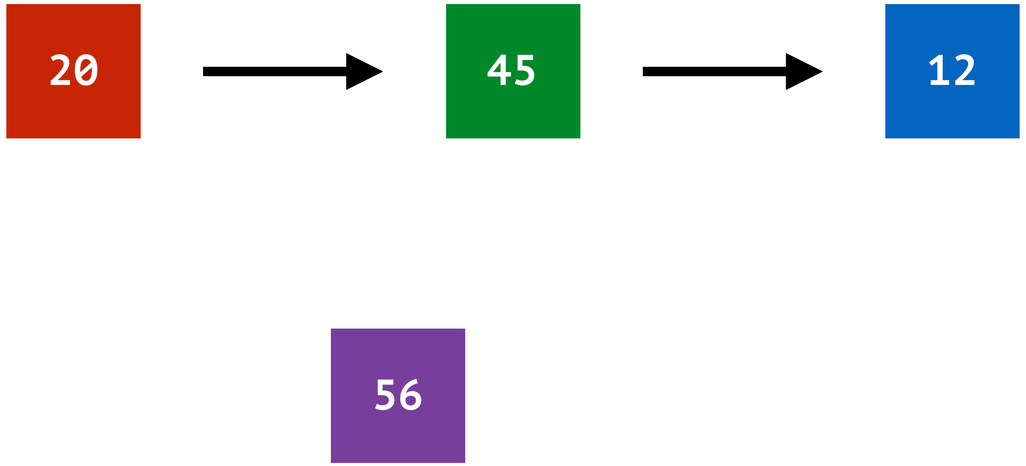
APPEND



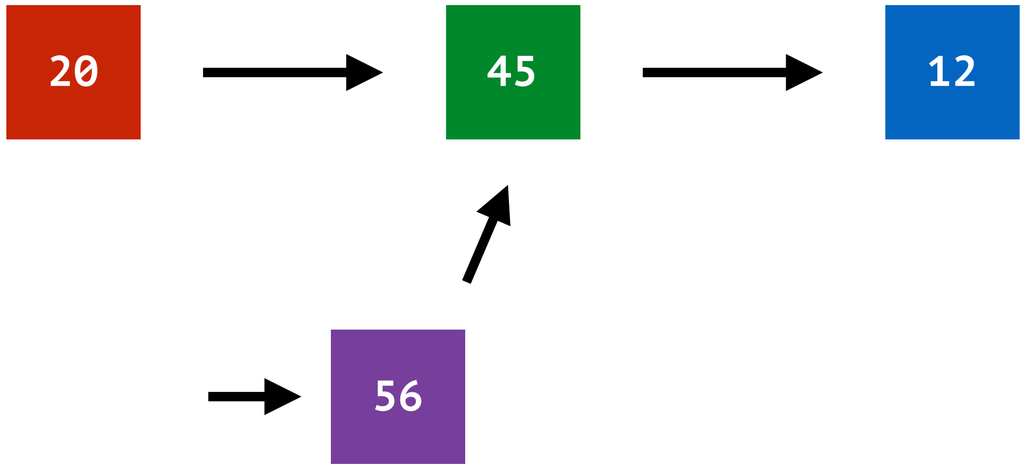
APPEND



INSERT



INSERT



COPY (MUTABLE DS)

COPY (MUTABLE DS)

COPY (IMMUTABLE)

COPY (IMMUTABLE)

Err…we're done?

- **Mutable single-linked list (assuming front, back and insertion nodes are known)**
  - Prepend: O(1)
  - Append: O(1)
  - Insert: O(1)
  - Find: O(n)
  - Copy: O(n)

- **Immutable single-linked list (assuming front, back and insertion nodes are known)**
  - Prepend: O(1)
  - Append: O(n)
  - Insert: O(n)
  - Find: O(n)
  - Copy: O(1)

WHAT ABOUT JS?

JAVASCRIPT

- **Primitive types are immutable**
  - Strings, Numbers, Booleans, Symbols

- **Objects (which includes Functions and Arrays) are mutable**

SPOT THE MUTANTS!

MUTATES! 💔

ARRAY.PROTOTYPE.PUSH

IMMUTABLE! 💖

ARRAY.PROTOTYPE.CONCAT

MUTATES! 💔

ARRAY.PROTOTYPE.SPLICE

IMMUTABLE! 💖

ARRAY.PROTOTYPE.SLICE

IMMUTABLE! 💖

ARRAY.PROTOTYPE.MAP

DOESN'T MUTATE...BUT... 😒

ARRAY.PROTOTYPE.FOREACH

MUTATIONS

```
const arr = [1, 2, 3]
```

MUTATIONS

```
const arr = [1, 2, 3]

arr.pop() // [1, 2]
```

MUTATIONS

```
const arr = [1, 2, 3]

arr.pop() // [1, 2]

arr.push(4) // [1, 2, 4]
```

MUTATIONS

```
const arr = [1, 2, 3]

arr.pop() // [1, 2]

arr.push(4) // [1, 2, 4]

arr.splice(1, 1) // [1, 4]
```

MUTATIONS

```
const obj = {}
```

MUTATIONS

```
const obj = {}

obj.a = 97 // {a: 97}
obj.b = 98 // {a: 97, b: 98}
```

MUTATIONS

```
const obj = {}

obj.a = 97 // {a: 97}
obj.b = 98 // {a: 97, b: 98}

delete obj.a // {b: 98}
```

IMMUTABILITY

```
const arr1 = [1, 2, 3]
```

IMMUTABILITY

```
const arr1 = [1, 2, 3]

const arr2 = arr1.slice(0, -1) // [1, 2]
```

IMMUTABILITY

```
const arr1 = [1, 2, 3]

const arr2 = arr1.slice(0, -1) // [1, 2]

const arr3 = [...arr2, 4] // [1, 2, 4]
```

IMMUTABILITY

```
const arr1 = [1, 2, 3]

const arr2 = arr1.slice(0, -1) // [1, 2]

const arr3 = [...arr2, 4] // [1, 2, 4]

const arr4 = arr3.filter((i) => i !== 2) // [1, 4]
```

IMMUTABILITY

```
const obj1 = {}
```

IMMUTABILITY

```
const obj1 = {}

const obj2 = {...obj1, a: 97} // {a: 97}
const obj3 = {...obj2, b: 98} // {a: 97, b: 98}
```

IMMUTABILITY

```
const obj1 = {}

const obj2 = {...obj1, a: 97} // {a: 97}
const obj3 = {...obj2, b: 98} // {a: 97, b: 98}

const {a, ...obj4} = obj3 // {b: 98}
```


IMMUTABLE

IMMUTABLE.JS

```
import {List} from 'immutable'
```

IMMUTABLE.JS

```
import {List} from 'immutable'

const l1 = List.of(1, 2, 3) // List [1, 2, 3]
```

IMMUTABLE.JS

```
import {List} from 'immutable'

const l1 = List.of(1, 2, 3) // List [1, 2, 3]
const l2 = l1.pop() // List [1, 2]
```

IMMUTABLE.JS

```
import {List} from 'immutable'

const l1 = List.of(1, 2, 3) // List [1, 2, 3]
const l2 = l1.pop() // List [1, 2]
const l3 = l2.push(4) // List [1, 2, 4]
```

IMMUTABLE.JS

```
import {List} from 'immutable'

const l1 = List.of(1, 2, 3) // List [1, 2, 3]
const l2 = l1.pop() // List [1, 2]
const l3 = l2.push(4) // List [1, 2, 4]

const l4 = l3.remove(1) // List [1, 4]
```

IMMUTABLE.JS

```
import {Map} from 'immutable'
```

IMMUTABLE.JS

```
import {Map} from 'immutable'

const m1 = new Map() // Map {}
```

IMMUTABLE.JS

```
import {Map} from 'immutable'

const m1 = new Map() // Map {}
const m2 = m1.set('a', 97) // Map {a: 97}
```

IMMUTABLE.JS

```
import {Map} from 'immutable'

const m1 = new Map() // Map {}
const m2 = m1.set('a', 97) // Map {a: 97}
const m3 = m2.set('b', 98) // Map {a: 97, b: 98}
```

IMMUTABLE.JS

```
import {Map} from 'immutable'

const m1 = new Map() // Map {}
const m2 = m1.set('a', 97) // Map {a: 97}
const m3 = m2.set('b', 98) // Map {a: 97, b: 98}

const m4 = m3.delete('a') // Map {b: 98}
```