

IMMUTABLE DATA STRUCTURE

DI FAN

FSA-1706-FLEX

THE CHALLENGE

```
import { projectMapper, projectFilter } from 'util/helper'
```

```
let projects = []
```

```
...
```

```
projects.push(graceshopper, stackathon, capstone)
```

```
projectMapper(projects)
```

```
projectFilter(projects)
```

```
console.log(projects)
```

```
import { projectMapper, projectFilter } from 'util/helper'
```

```
let projects = []
```

```
...
```

```
projects.push(graceshopper, stackathon, capstone)
```

```
projectMapper(projects)
```

```
projectFilter(projects)
```

```
console.log(projects)
```

**MUTABLE
STATE**

MUTABLE

STATE

State

| stāt | noun

Value of an identity at a time.

— *Rich Hickey*

UI = render(state)


```
const state = { foo: 'foo'}
```

```
UI = render(state) // { foo: 'foo'}
```

```
setState('bar')
```

```
UI = render(state) // { bar: 'bar'}
```

```
setState('baz')
```

```
UI = render(state) // { baz: 'baz'}
```





REDUX
"No Side Effect Please"

~~SIDE EFFECTS~~

MUTABILITY

```
const state = { foo: 'foo'}
```

```
UI = render(state) // { foo: 'foo'}
```

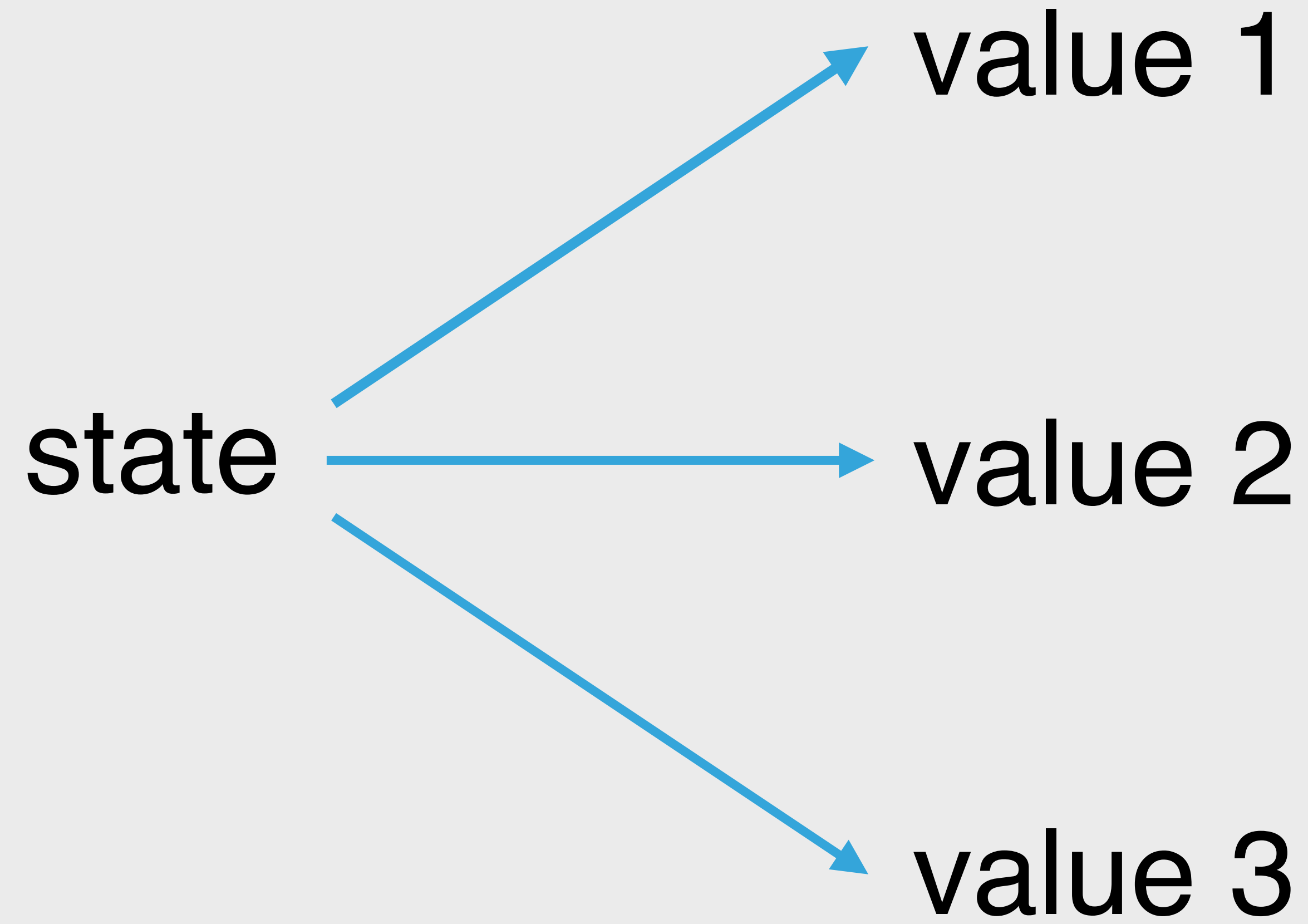
```
setState('bar')
```

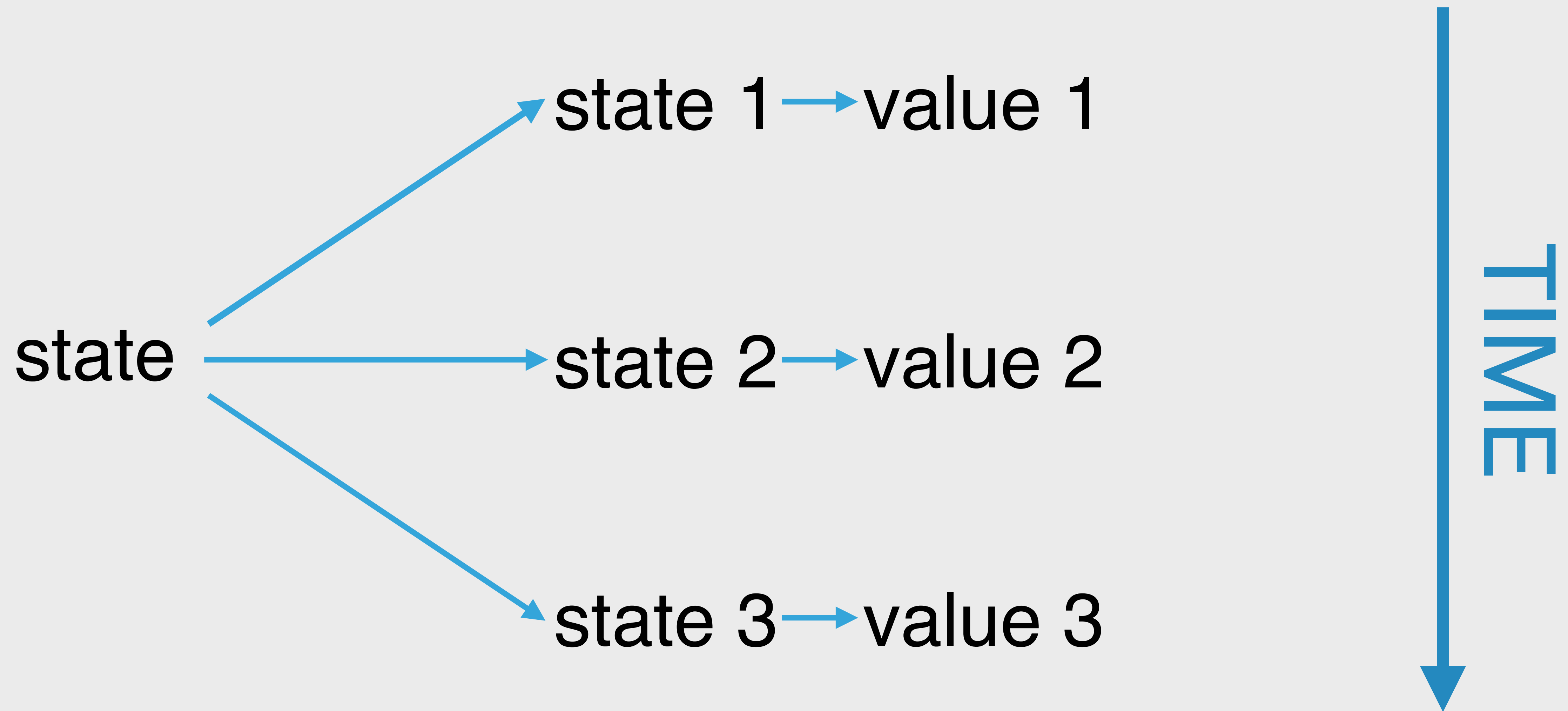
```
UI = render(state) // { bar: 'bar'}
```

```
setState('baz')
```

```
UI = render(state) // { baz: 'baz'}
```







```
const state0 = { foo: 'foo' }  
UI = render(state0)
```

```
const state1 = setState('bar')  
UI = render(state1)
```

```
/* Identity */  
state = state0 —> state = state1
```



~~SIDE EFFECTS~~

~~MUTABILITY~~


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

```
let list = List.of(1, 2, 3)
```

```
let list2 = list.push(4)
```

```
console.log(list.length) // 3
```

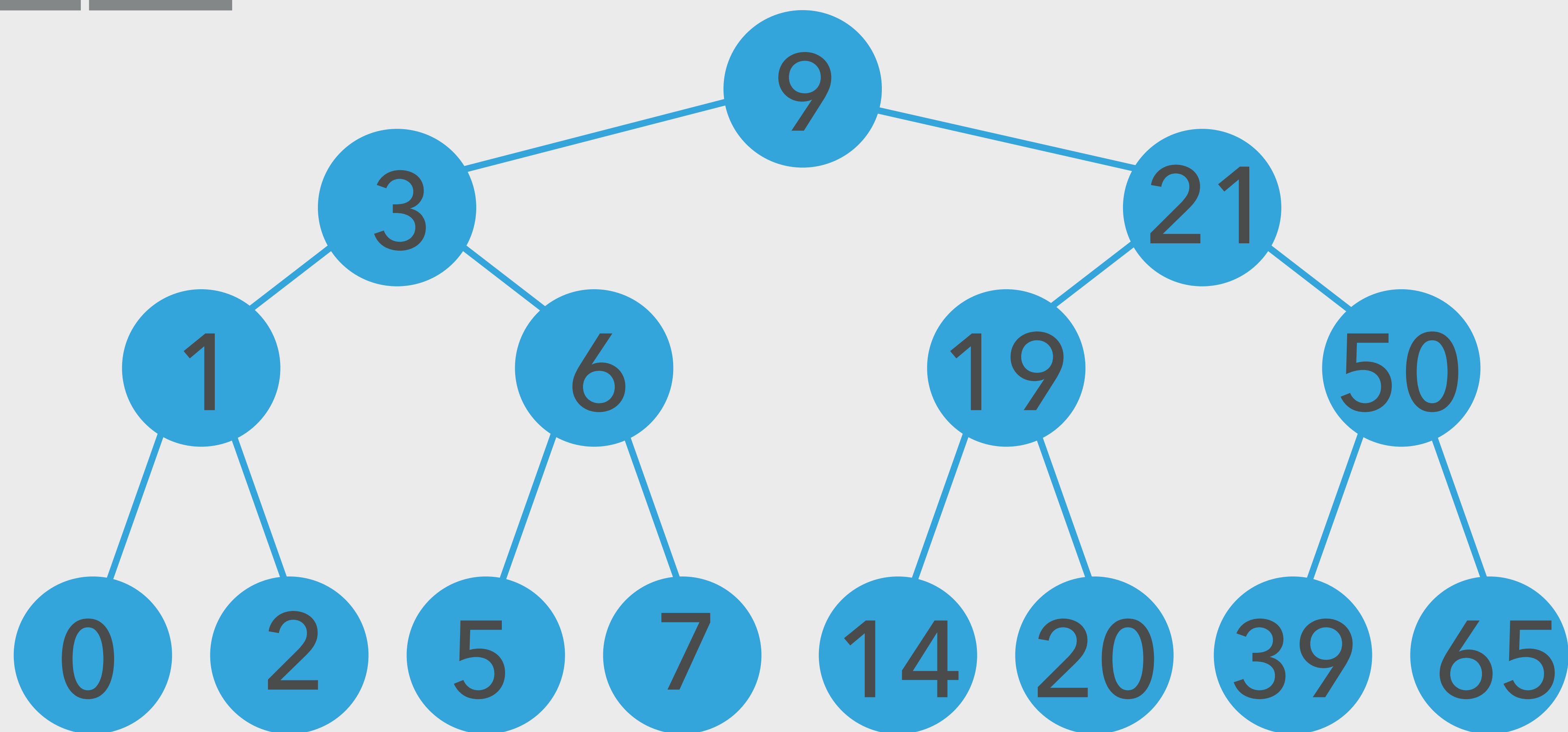
```
console.log(list2[3]) // 4
```


**Oops...That's
SLOW & BIG**

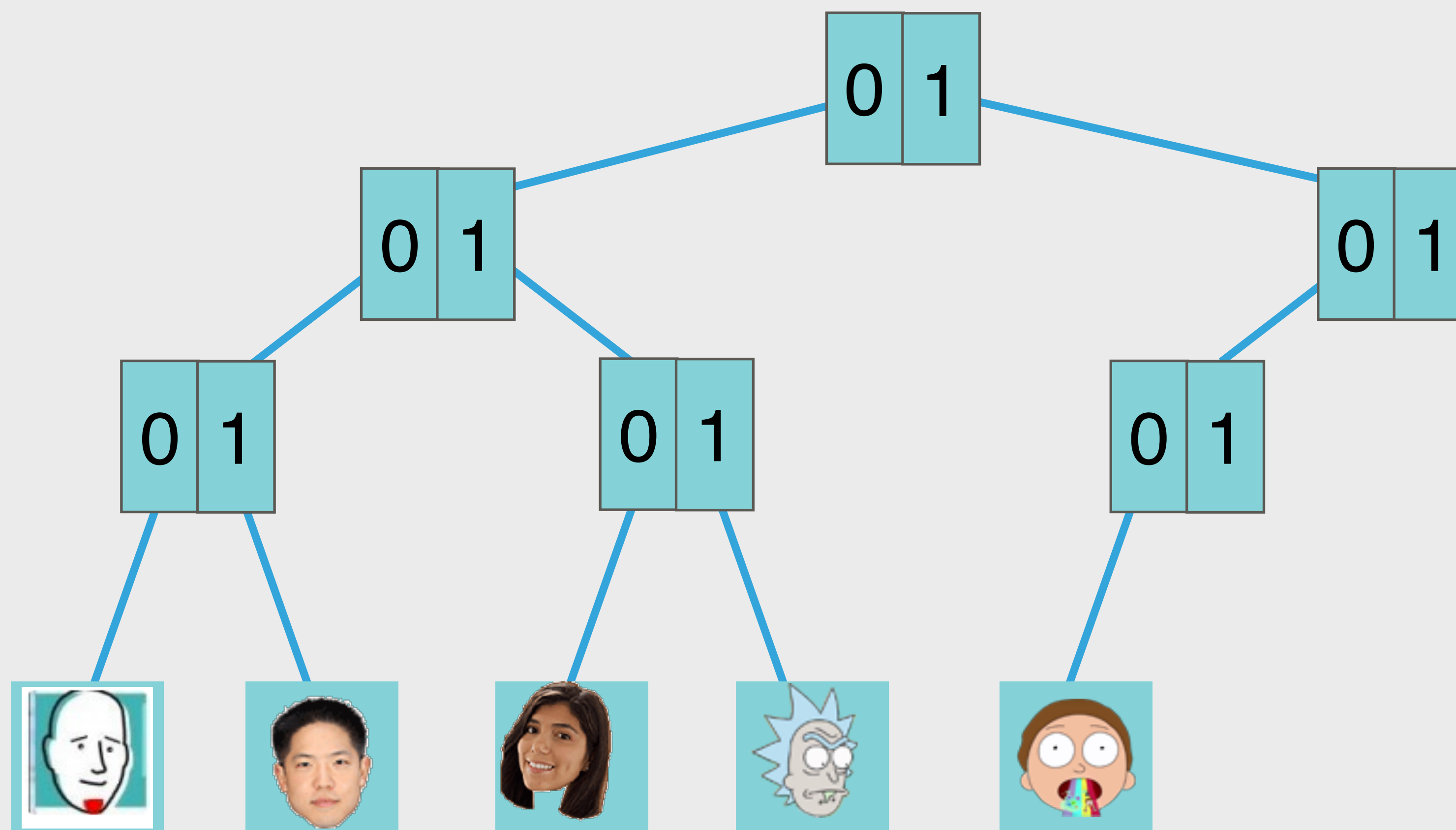


RETRIEVAL

TREE

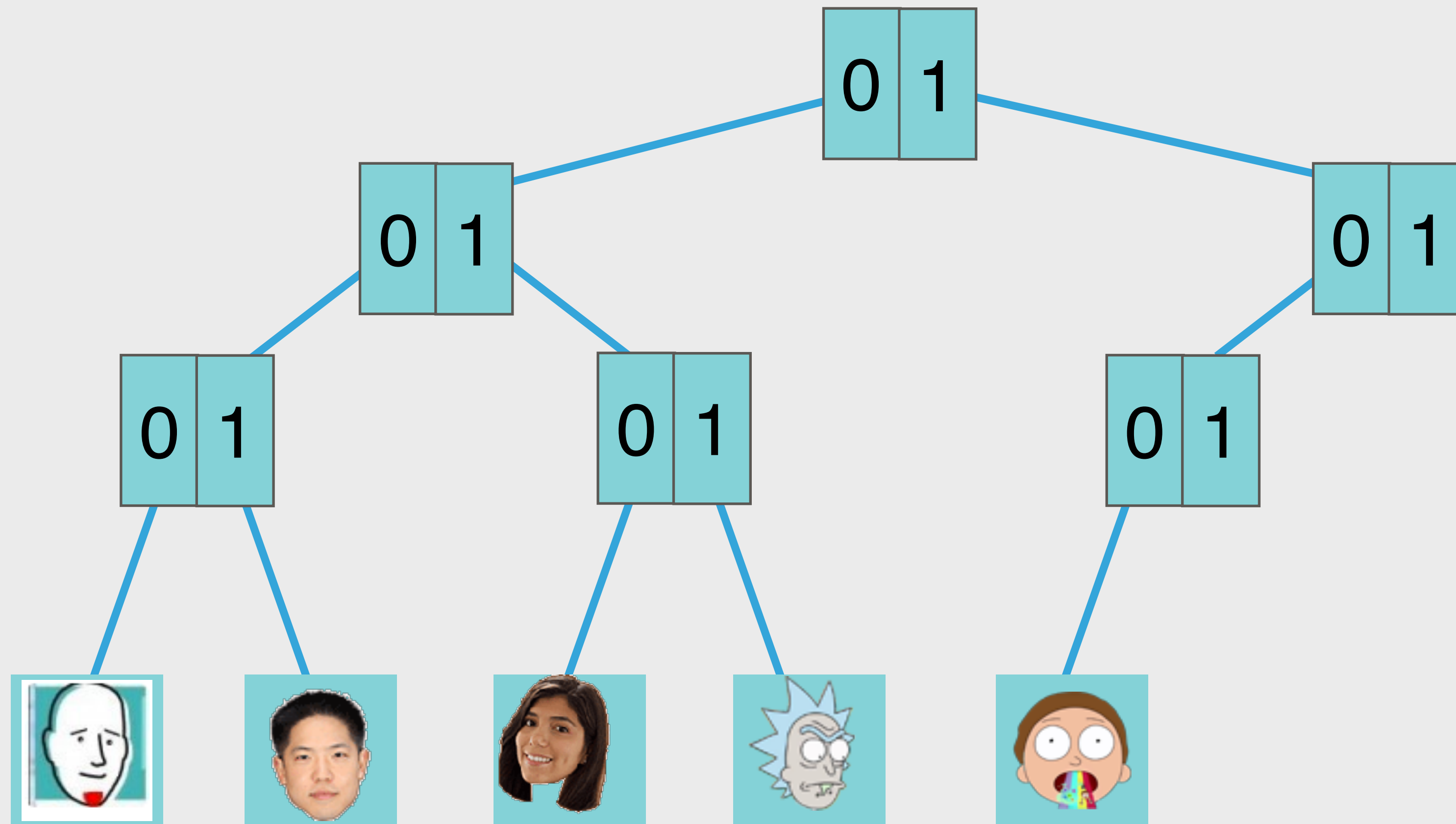


TRIE



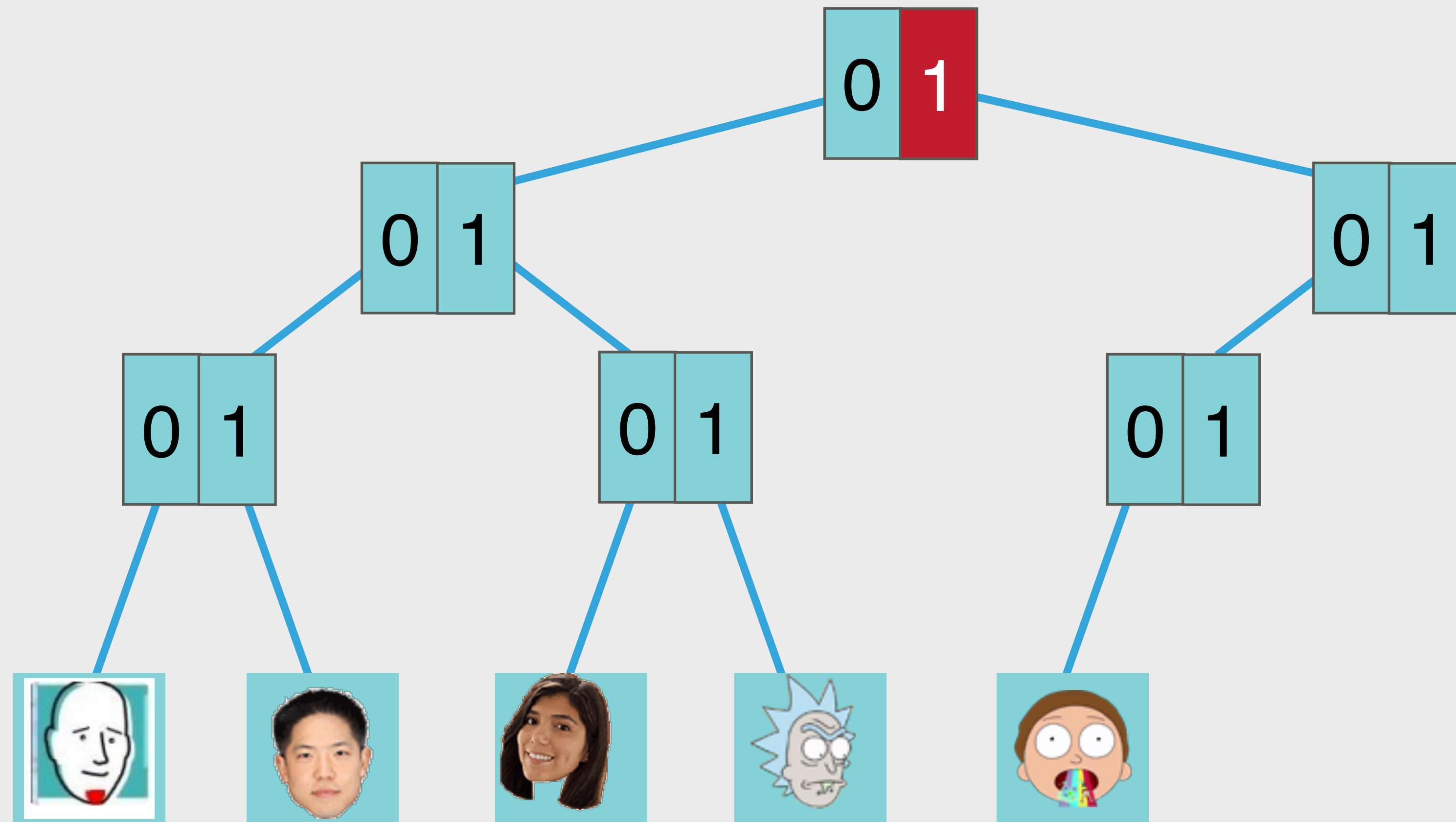
index = 5

index = '0b101'



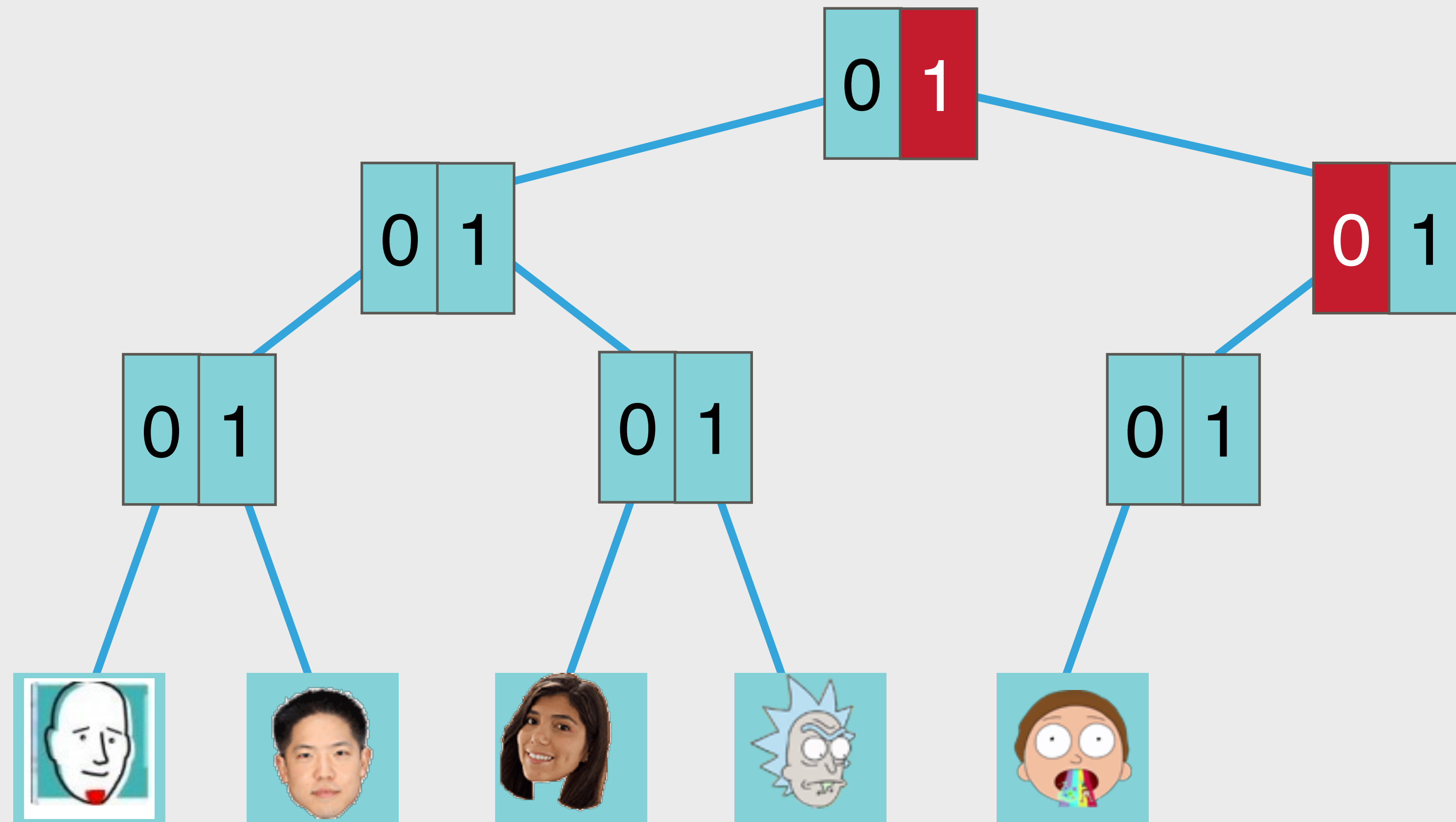
index = 5

index = '0b101'



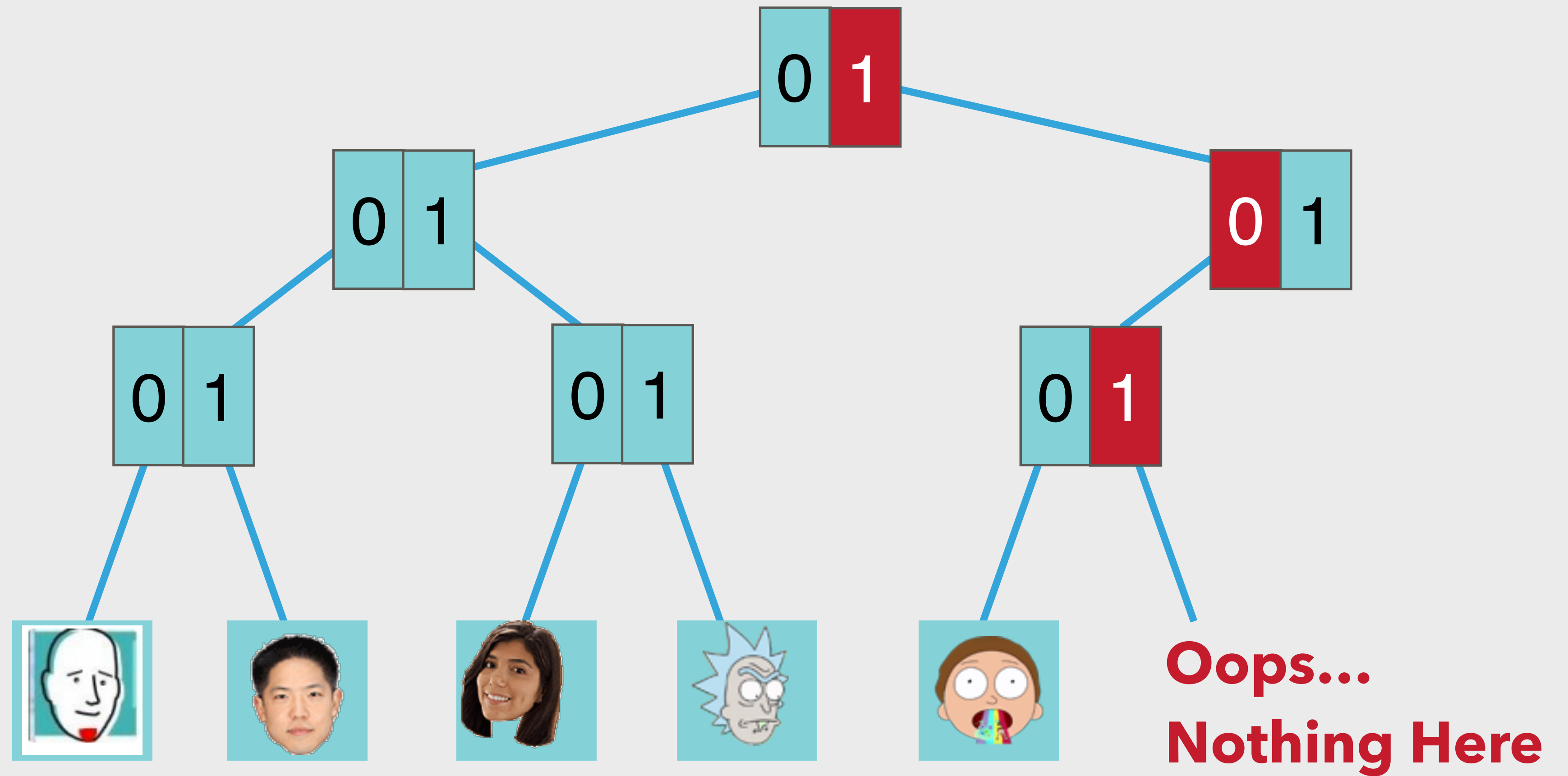
index = 5

index = '0b101'

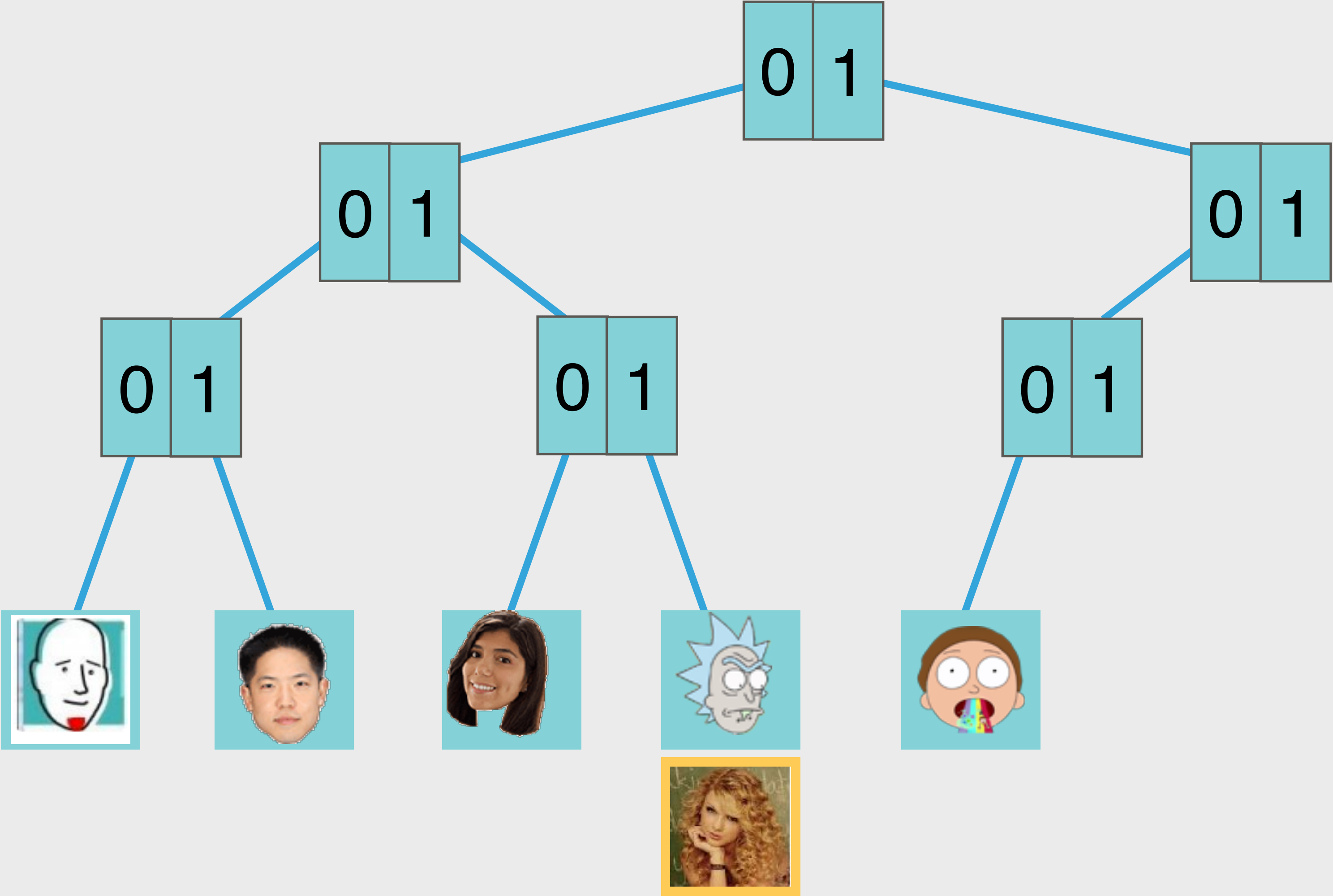


index = 5

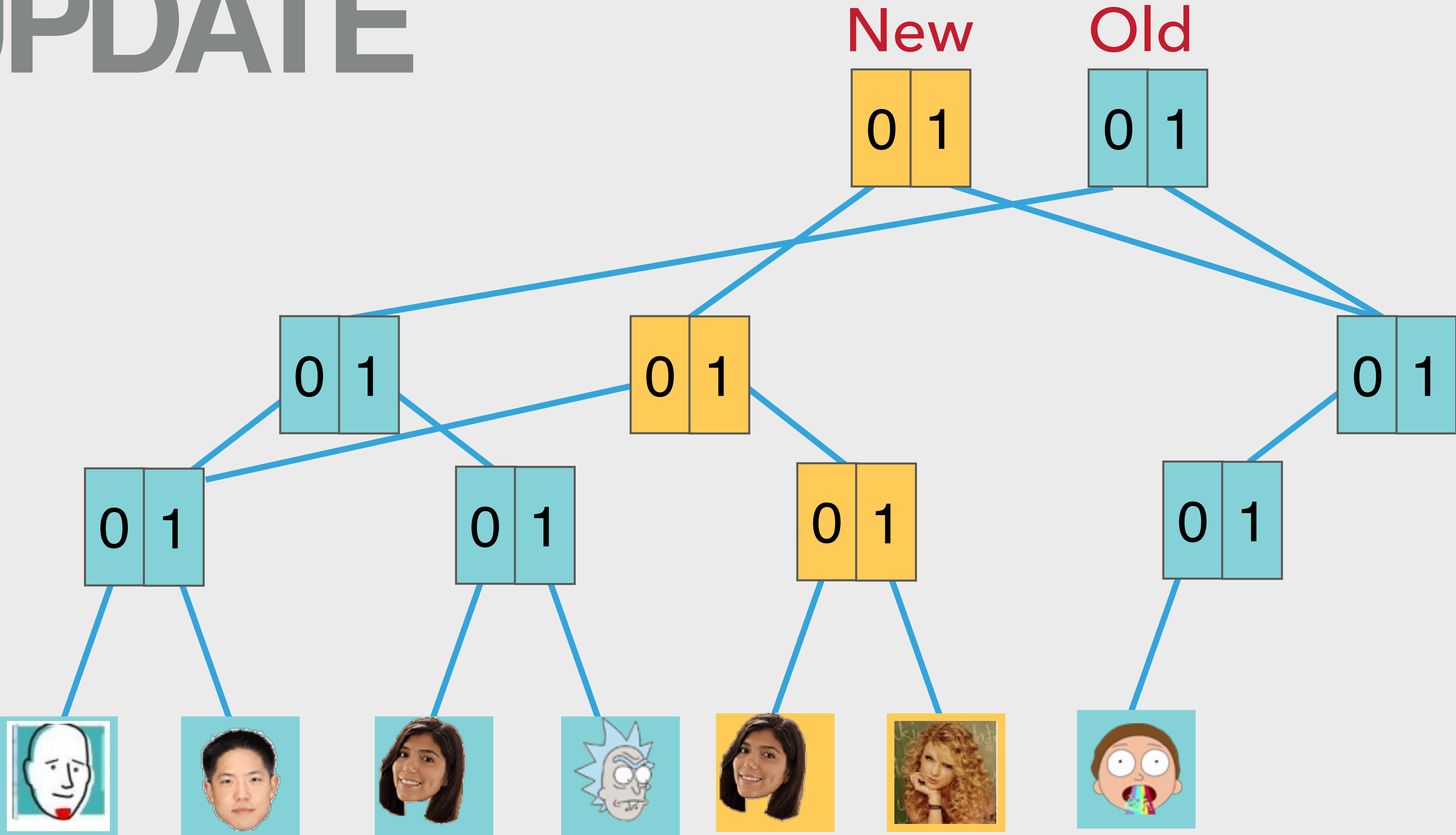
index = '0b101'



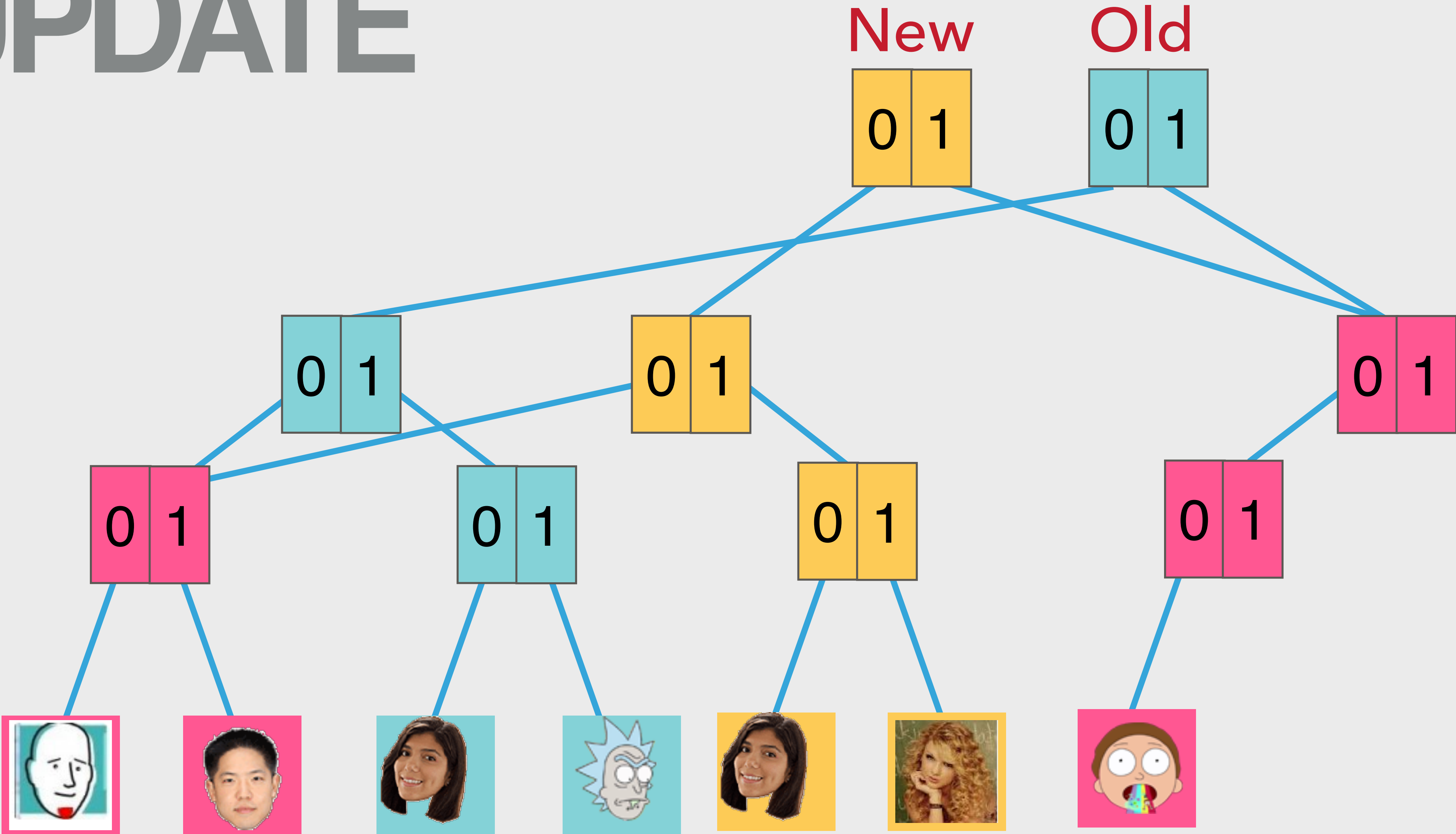
UPDATE



UPDATE



UPDATE



BITMAPPED VECTOR TRIE

```
list(5) // '0b101'
```

```
list(22832) // '0b101100100110000'
```

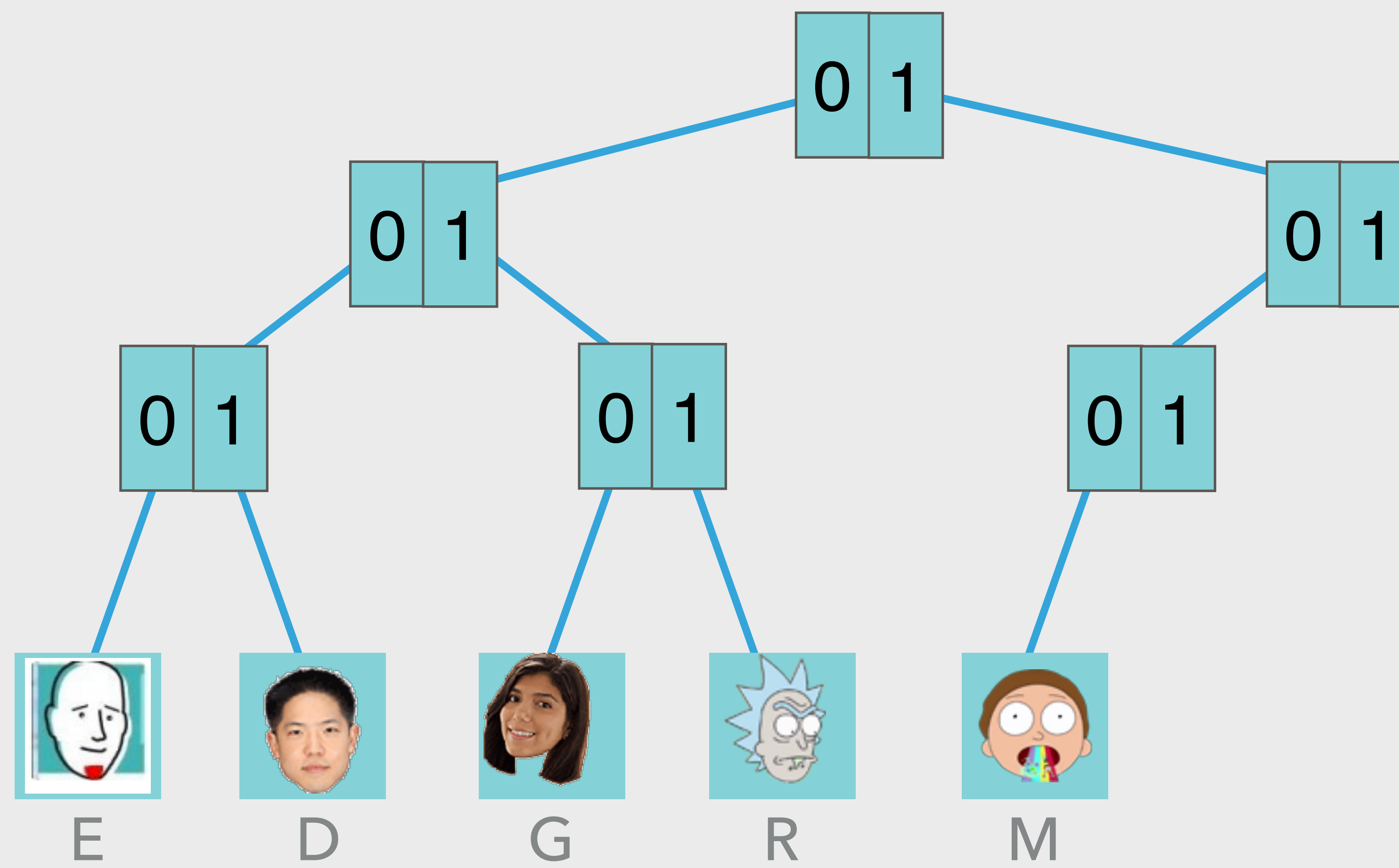
list(5) // '0b101'

list(22832) // '0b101100100110000'

index = 10110 01001 10000

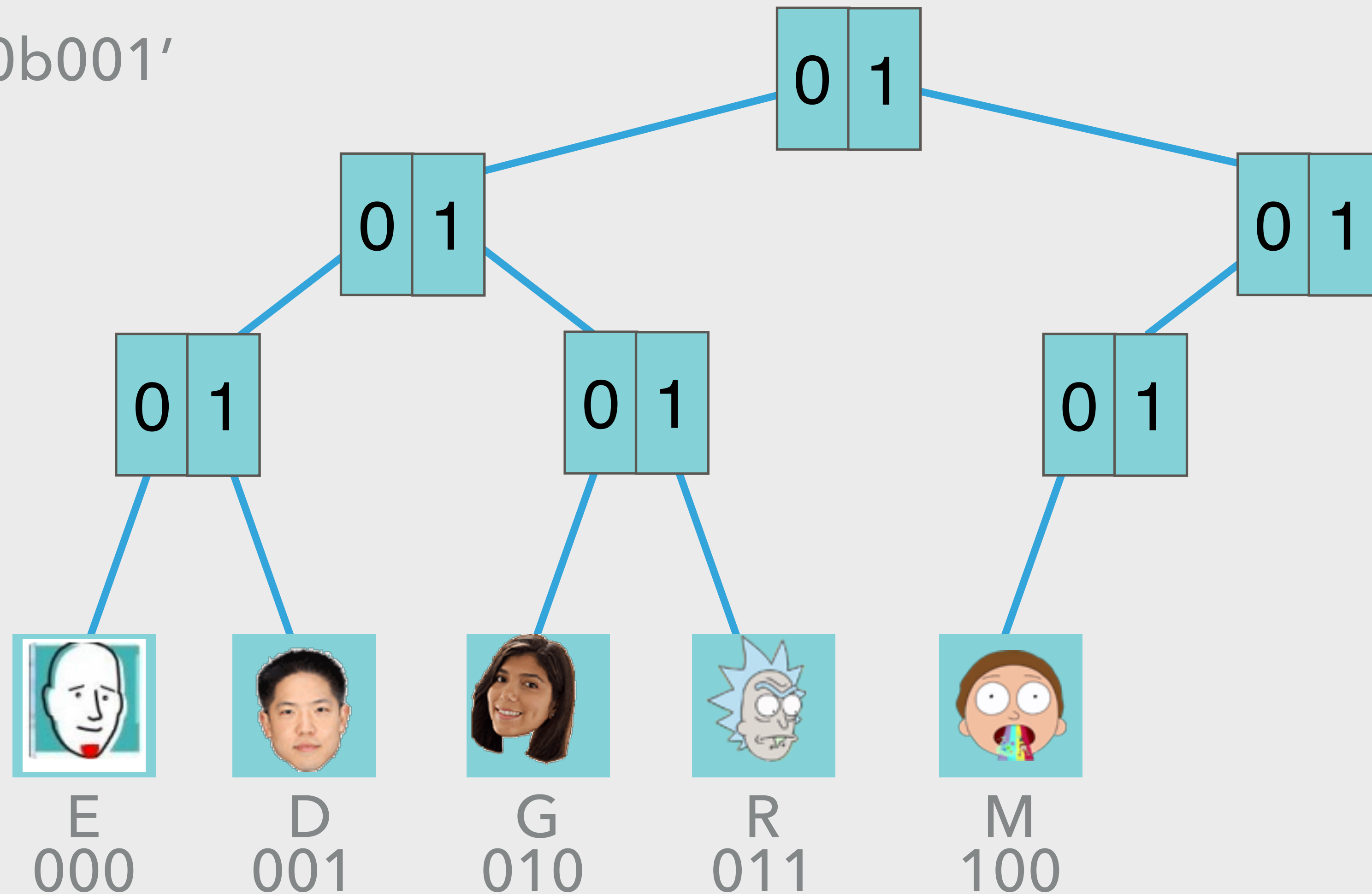
list —> 10110 —> 01001 —> 10000

**WHAT
ABOUT
OBJECTS?**



HASH KEYS

hash('D') = 1 // '0b001'



**HASH ARRAY
MAPPED TRIE**

PATH COPYING

STRUCTURAL

SHARING

Worry native performance no more...
SMALL & FAST



HOW
TO USE
... IN JS

<https://facebook.github.io/immutable-js/>

IMMUTABLE



Star

21311

Mori

Github Repository

Rationale

Immutability

Mori is not an island

Using Mori

Notation

Fundamentals

equals

hash

Type Predicates

isList

isSeq

isVector

isMap

isSet

isCollection

isSequential

isAssociative

mori <https://swannodette.github.io/mori/>

A library for using ClojureScript's persistent data structures and supporting API from the comfort of vanilla JavaScript.

Rationale

JavaScript is a powerful and flexible dynamic programming language with a beautiful simple associative model at its core. However this design comes at the cost of ubiquitous mutability. Mori embraces the simple associative model but leaves mutability behind. Mori delivers the following benefits to JavaScript:

- Efficient immutable data structures - no cloning required
- Uniform iteration for all types
- Value based equality

Modern JavaScript engines like V8, JavaScriptCore, and SpiderMonkey deliver the performance needed to implement persistent data structures well.


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

```
let list = List.of(1, 2, 3)
```

```
let list2 = list.push(4)
```

```
console.log(list.size) // 3
```

```
console.log(list2.get(3)) // 4
```

```
import mori from 'mori'
```

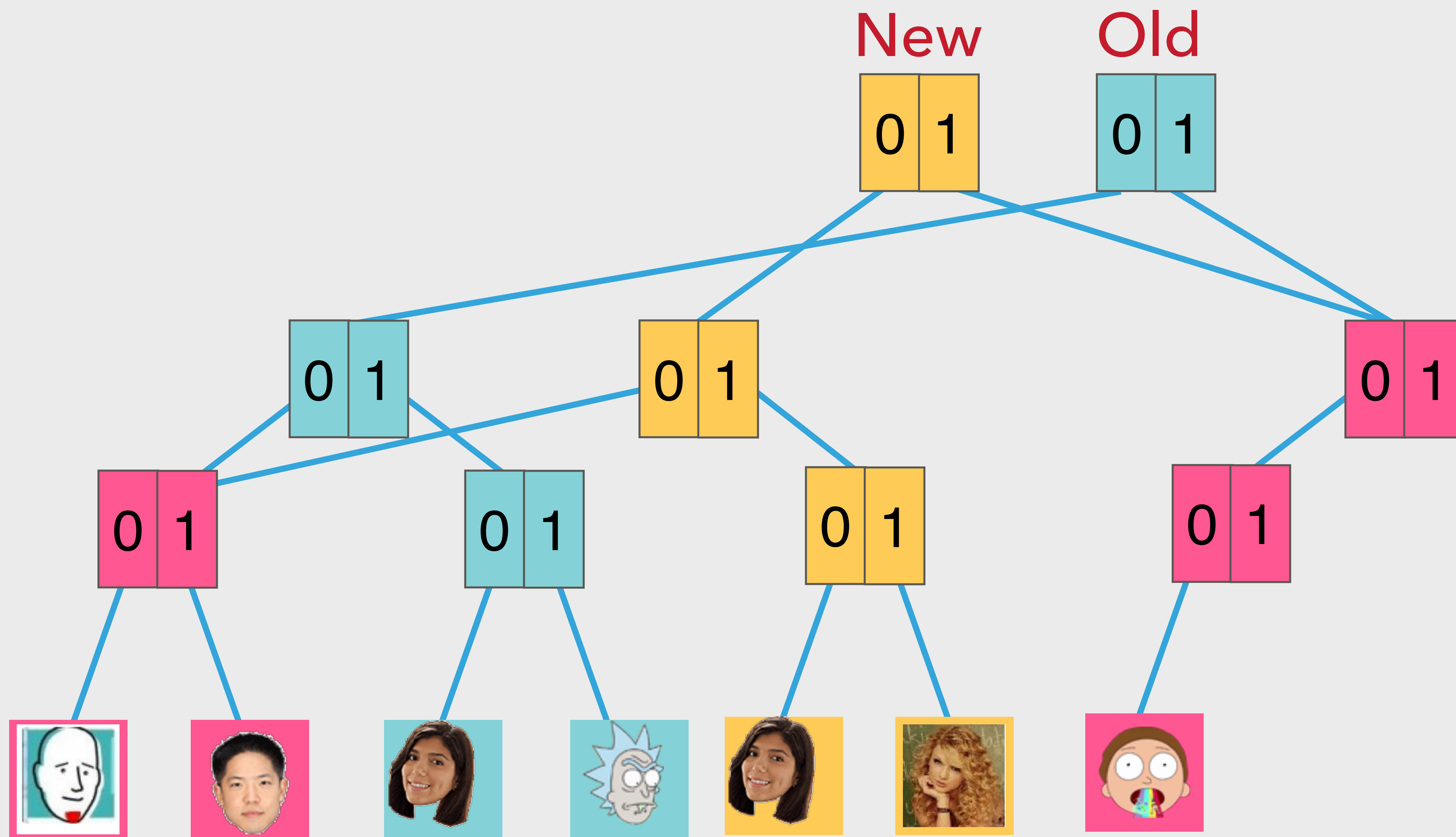
```
let v = mori.vector(1, 2, 3)
```

```
let v2 = mori.conj(v, 4)
```

```
console.log(mori.count(v)) // 3
```

```
console.log(mori.get(v2, 3)) // 4
```

RECONCILIATION



**NO MORE
MUTABILITY!!**

