IMMUTABLE DATA STRUCTURE

DI FAN

FSA-1706-FLEX

WHY
WHAT
HOW

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)

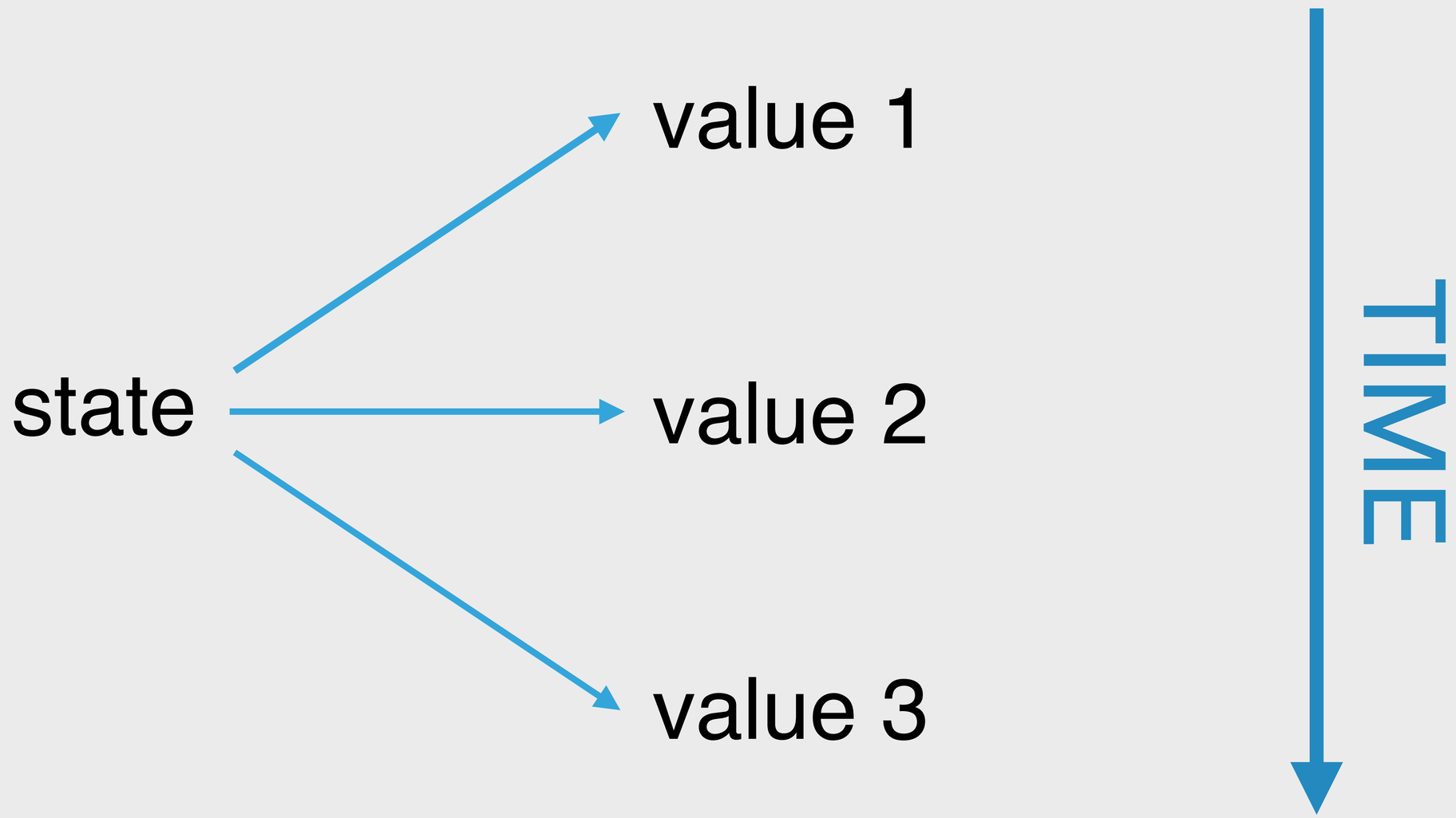
```
const state = { foo: ‘foo’}

UI = render(state) // { foo: ‘foo’}
setState(‘bar’)
UI = render(state) // { bar: ‘bar’}
setState(‘baz’)
UI = render(state) // { baz: ‘baz’}
```

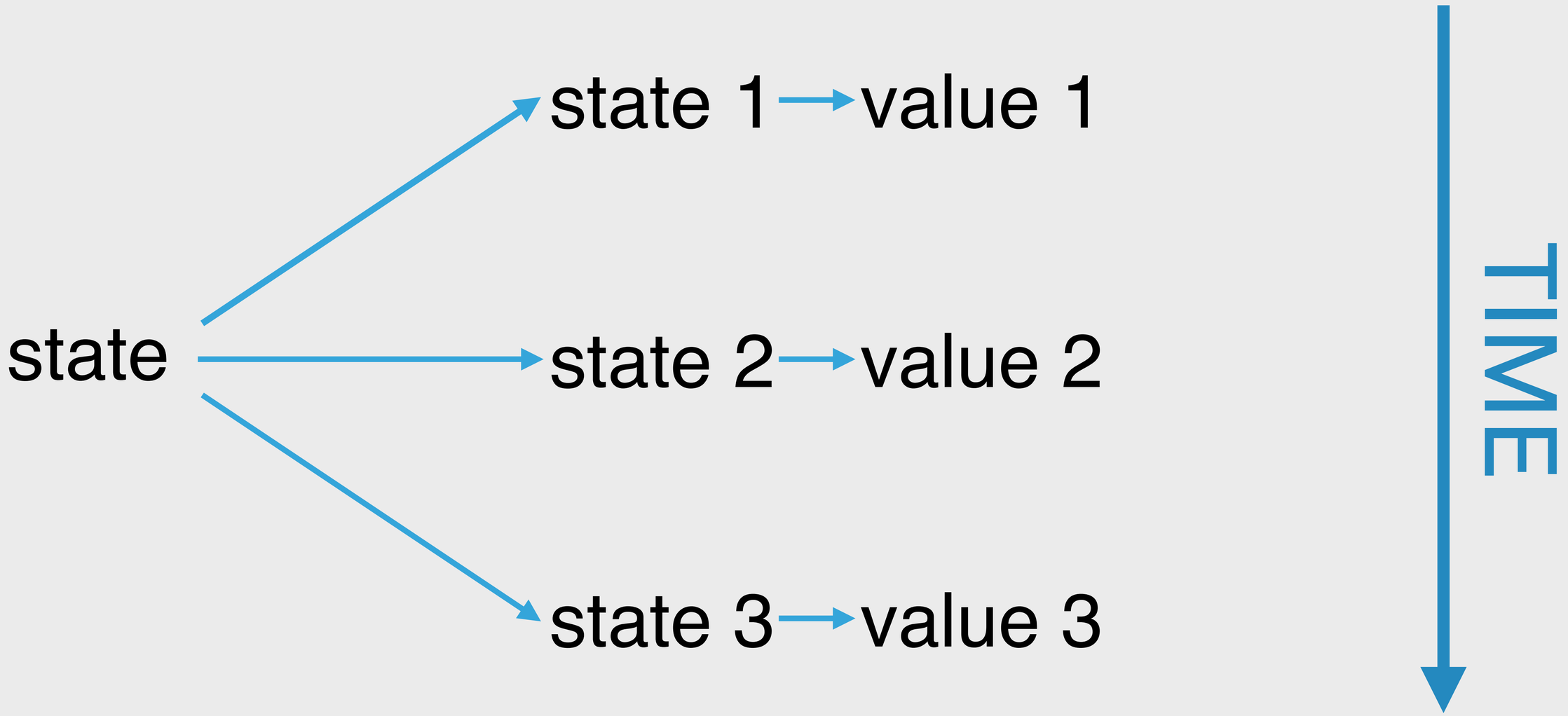
REDUX
“No Side Effect Please”

~~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’}
```


state
value 1
value 2
value 3
TIME


state
state 1
value 1
state 2
value 2
state 3
value 3
TIME

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

const state1 = setState(‘bar’)
UI = render(state1)

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

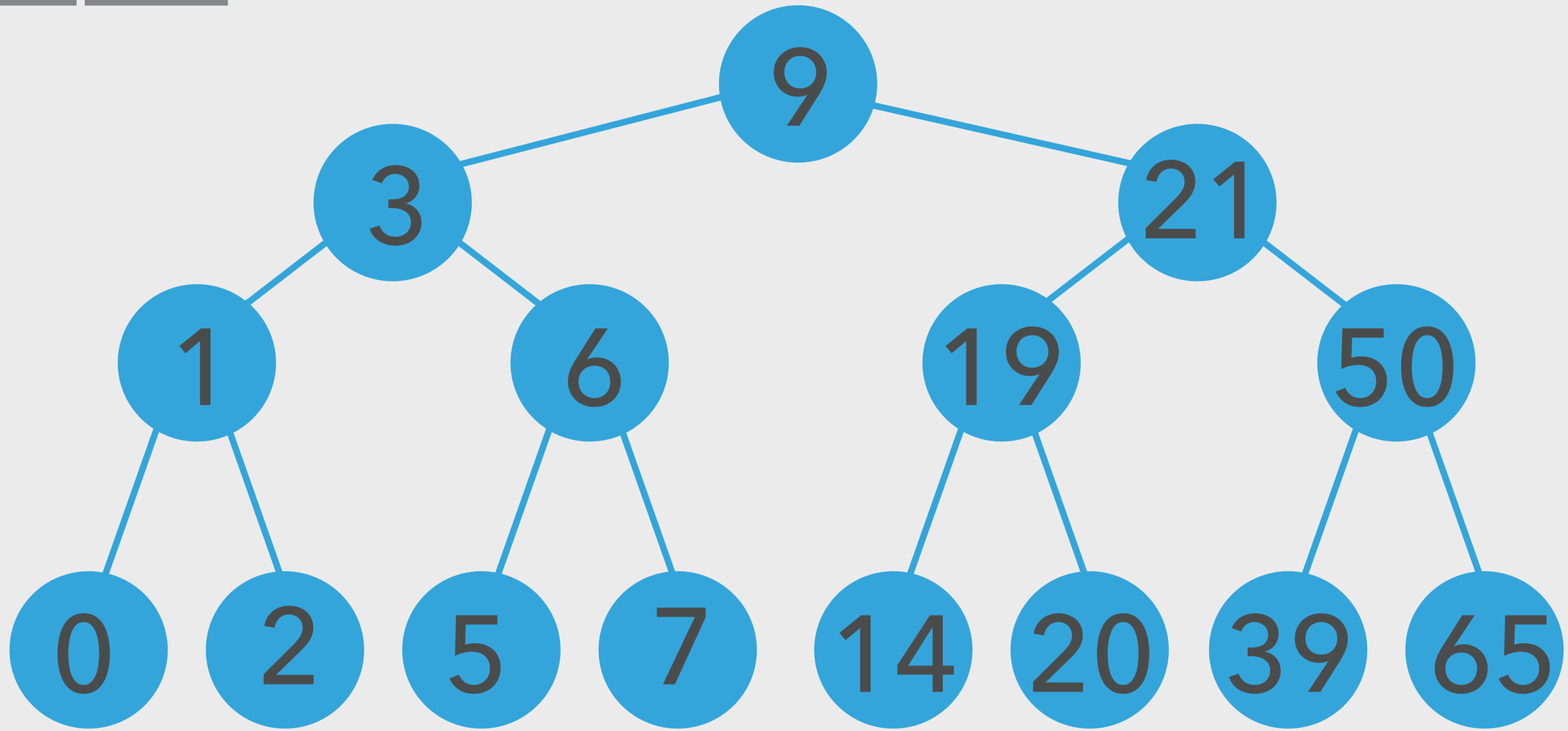
~~SIDE EFFECTS~~

~~MUTABILITY~~

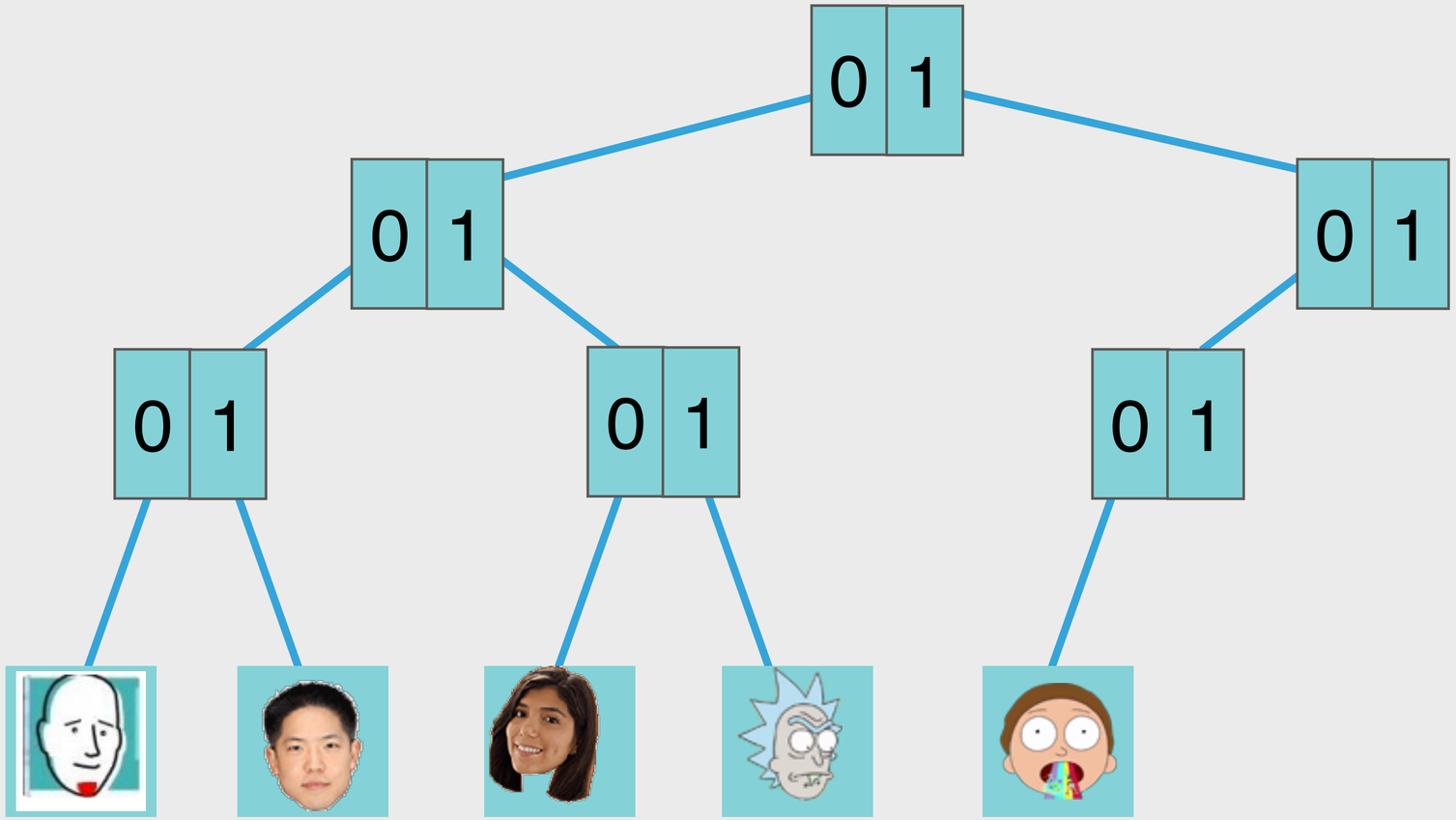

Oops...That's
SLOW & BIG

RETRIEVAL

TREE

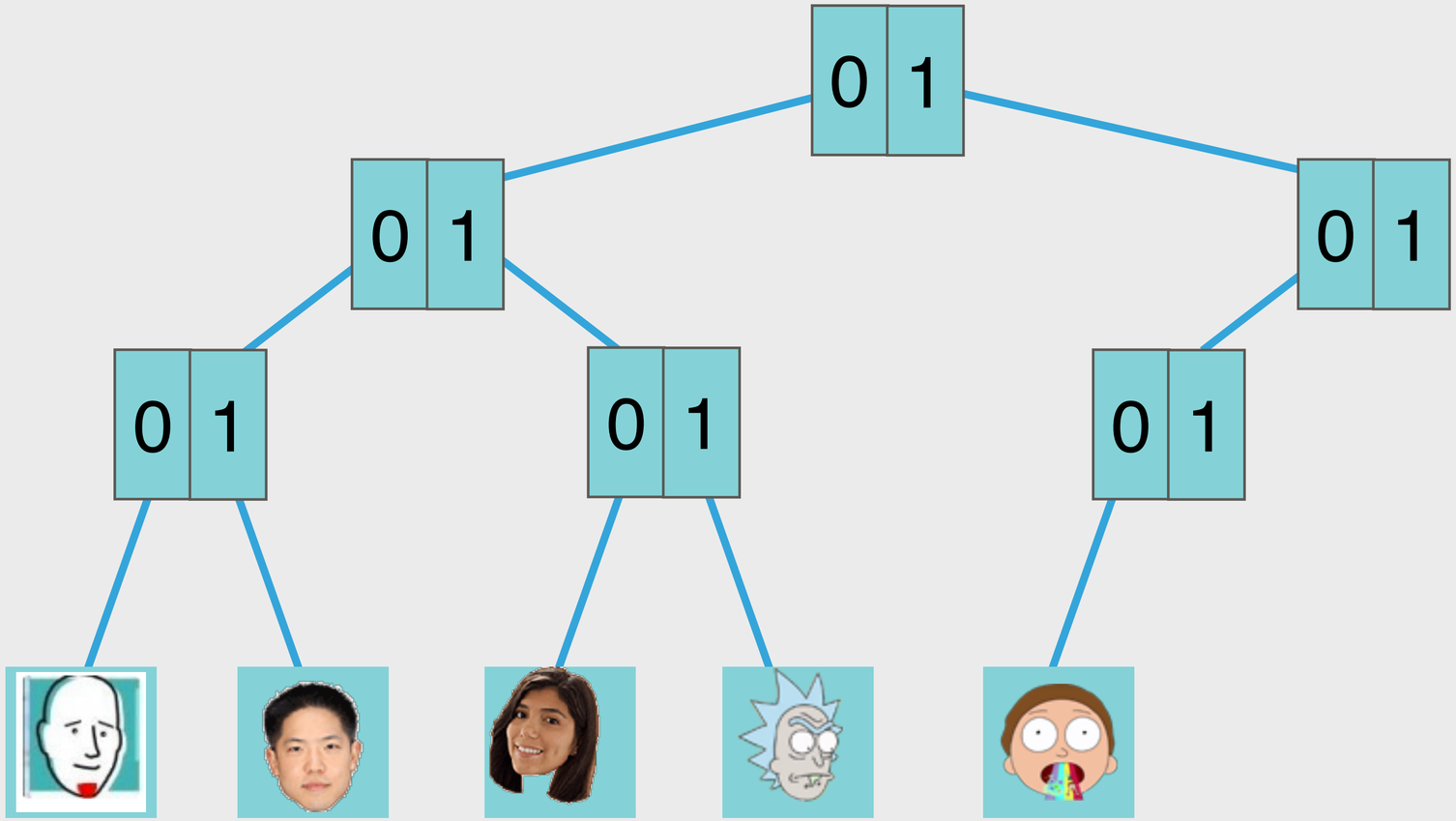


TRIE



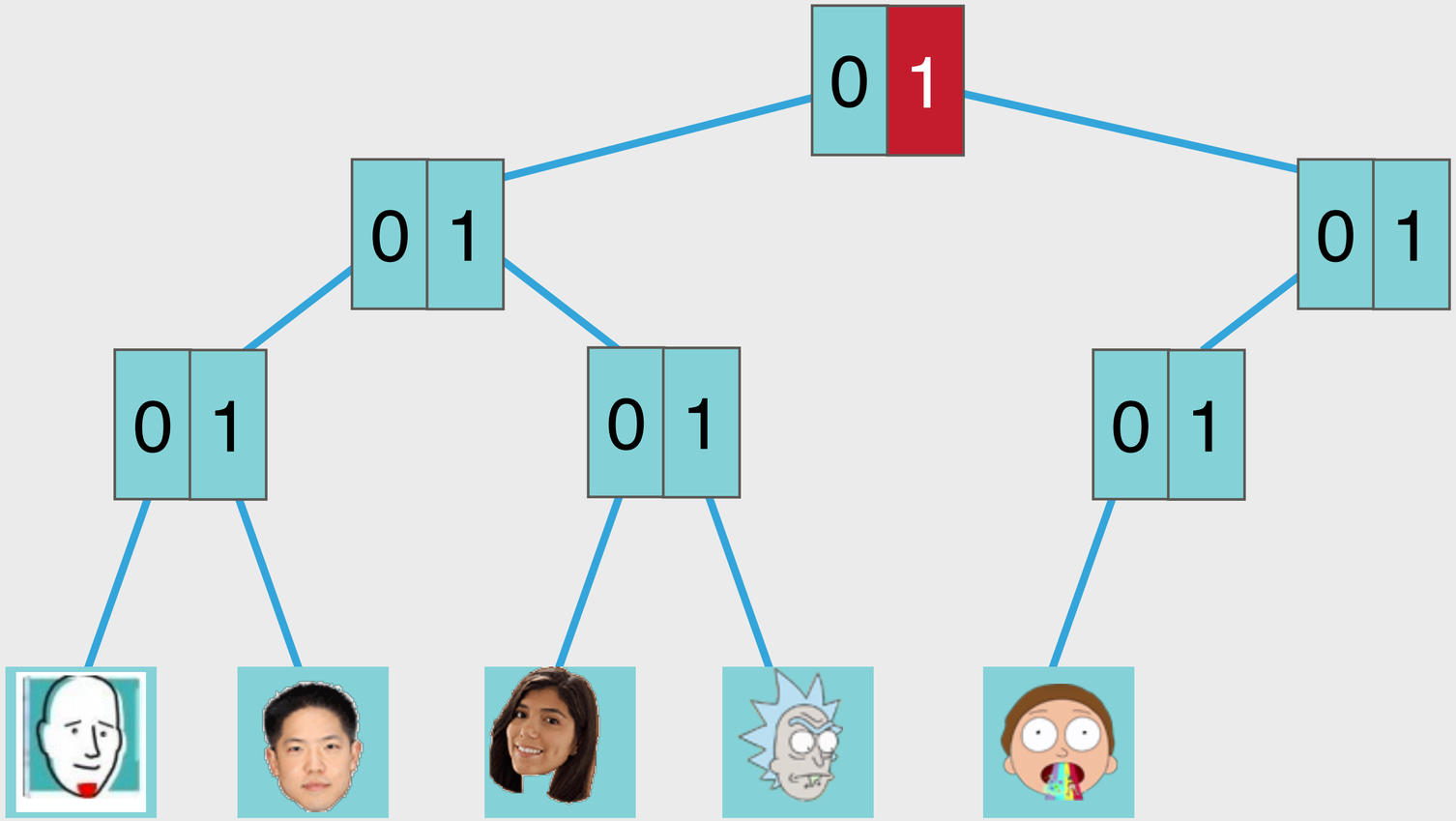
index = 5

index = ‘0b101‘



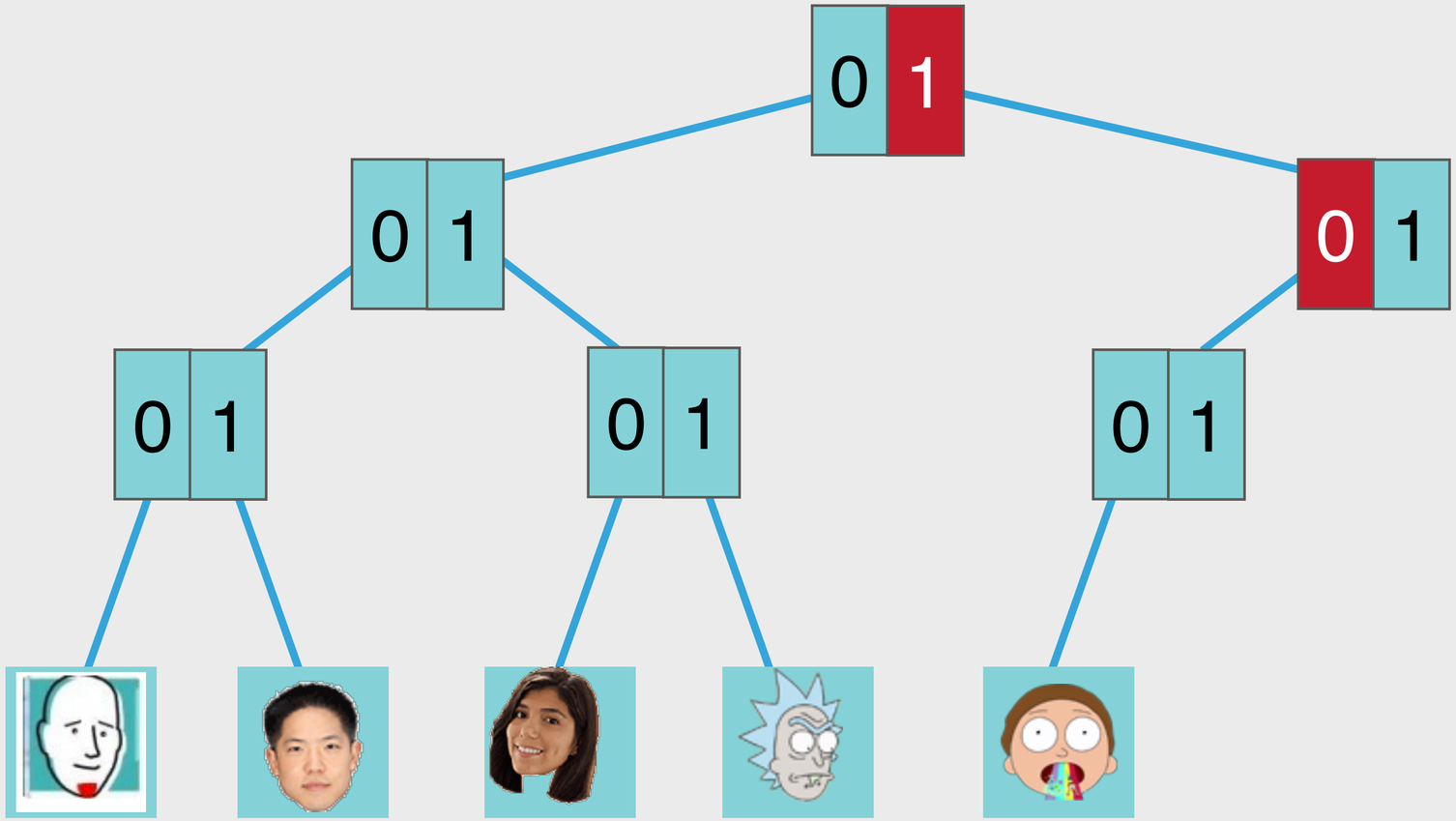
index = 5

index = ‘0b101‘



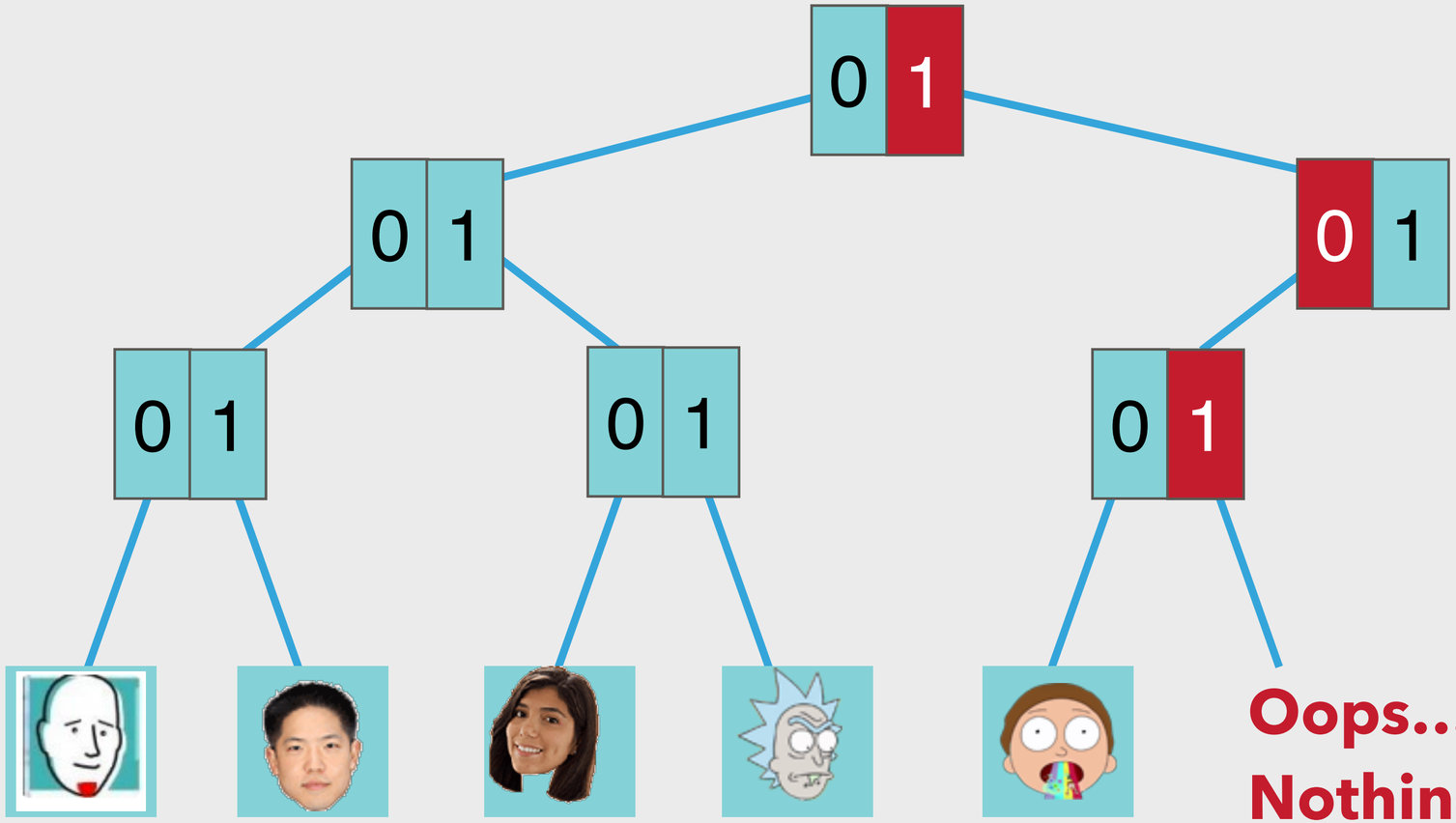
index = 5

index = ‘0b101‘

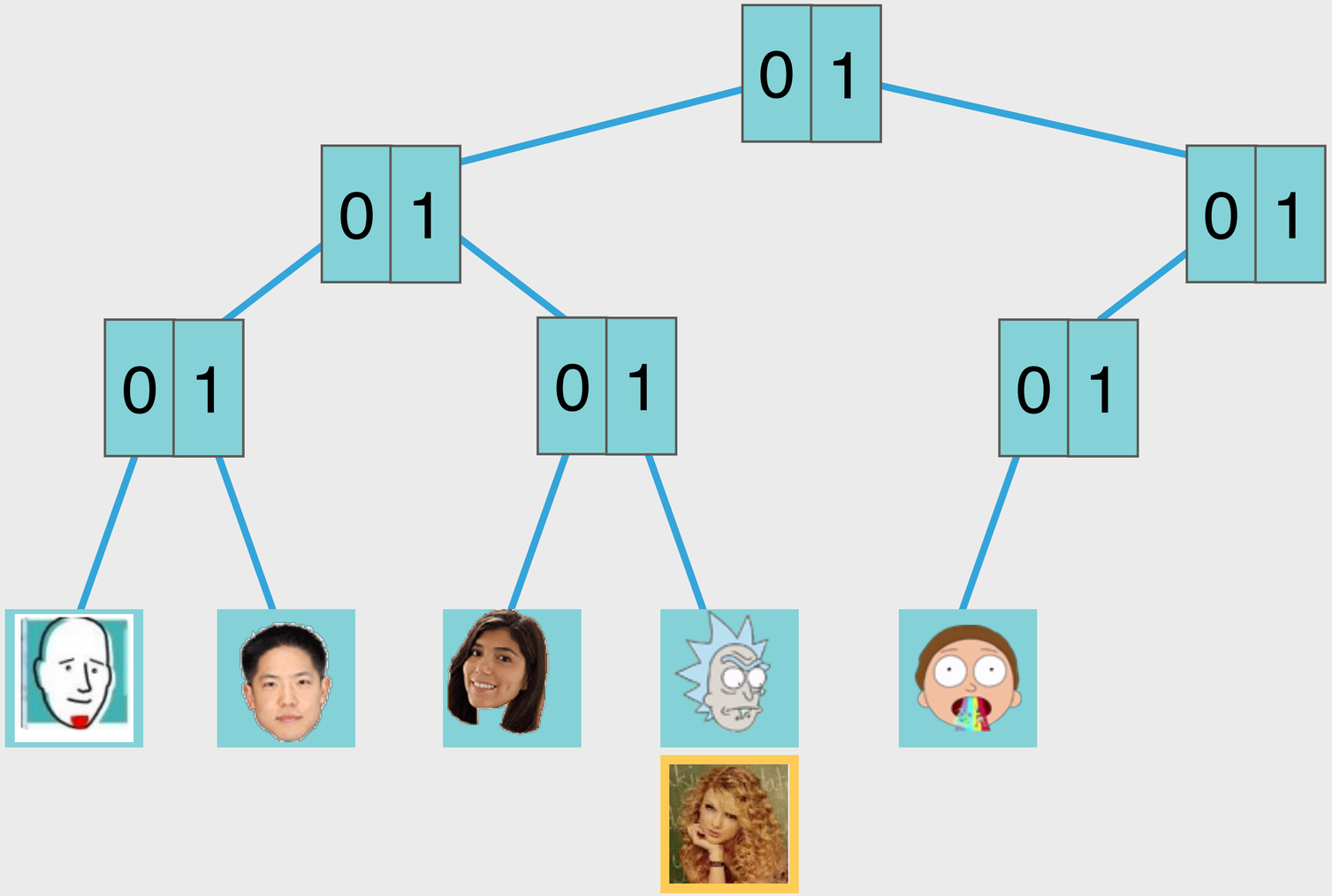


index = 5

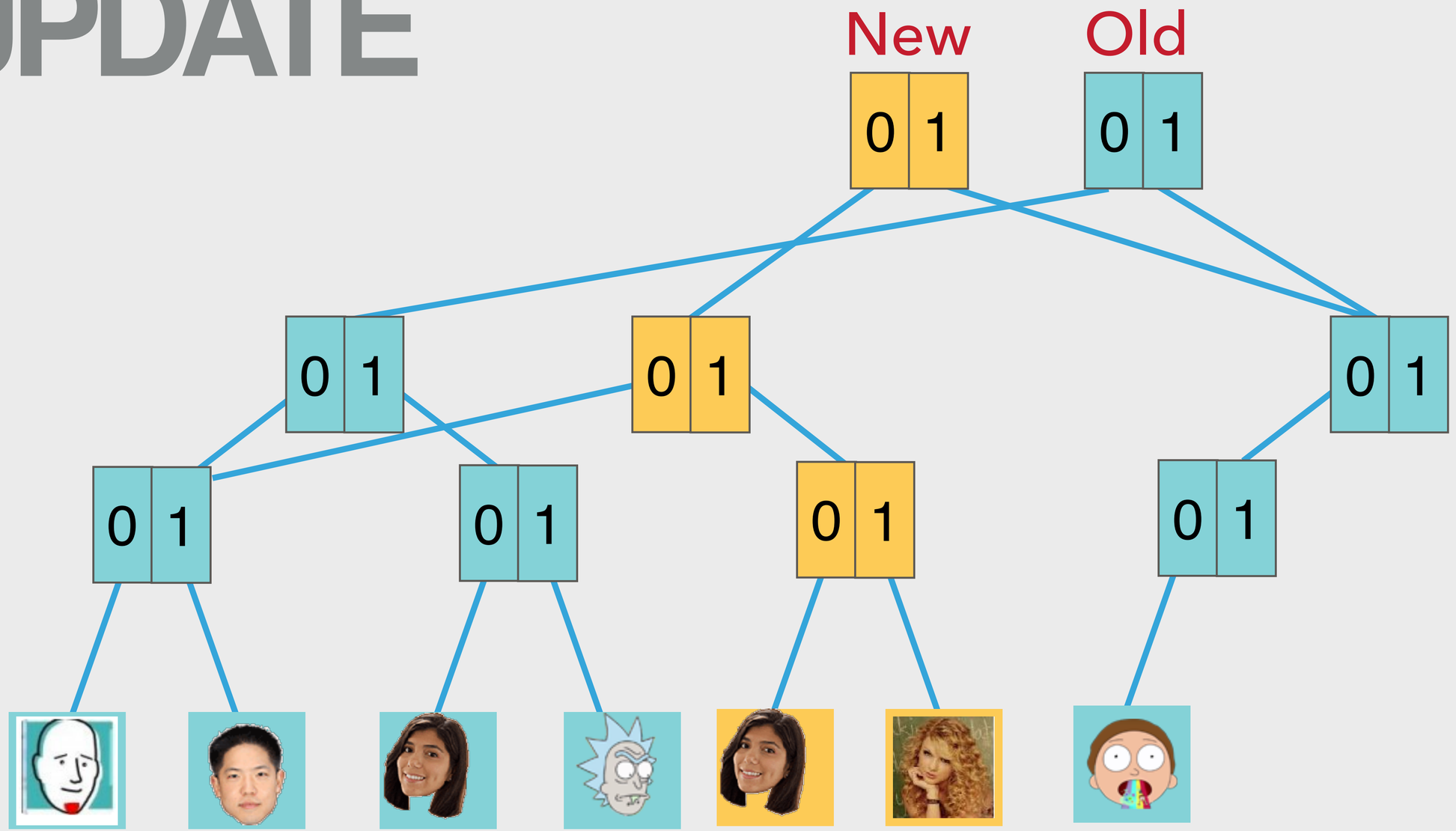
index = ‘0b101‘



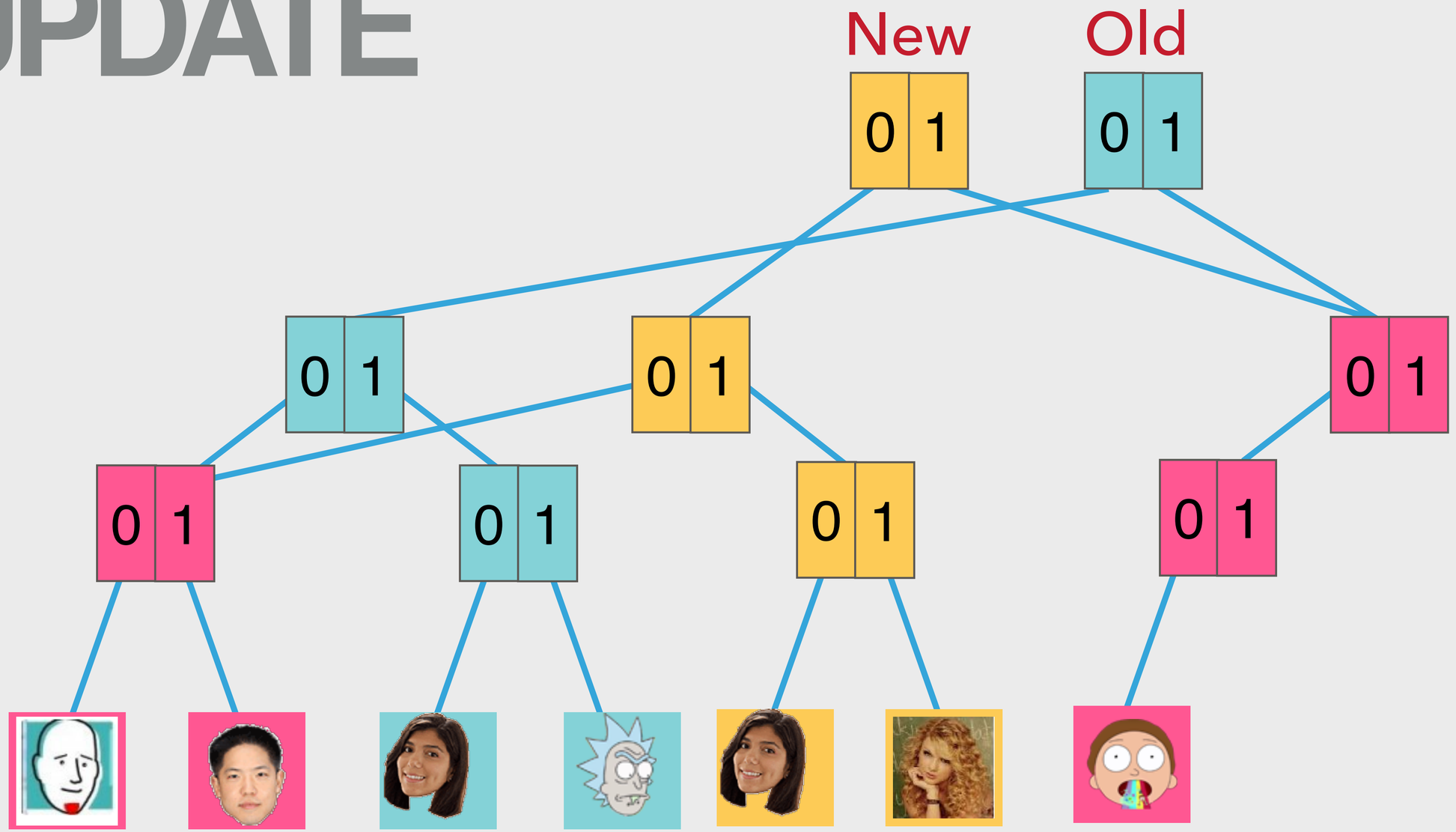
UPDATE



UPDATE

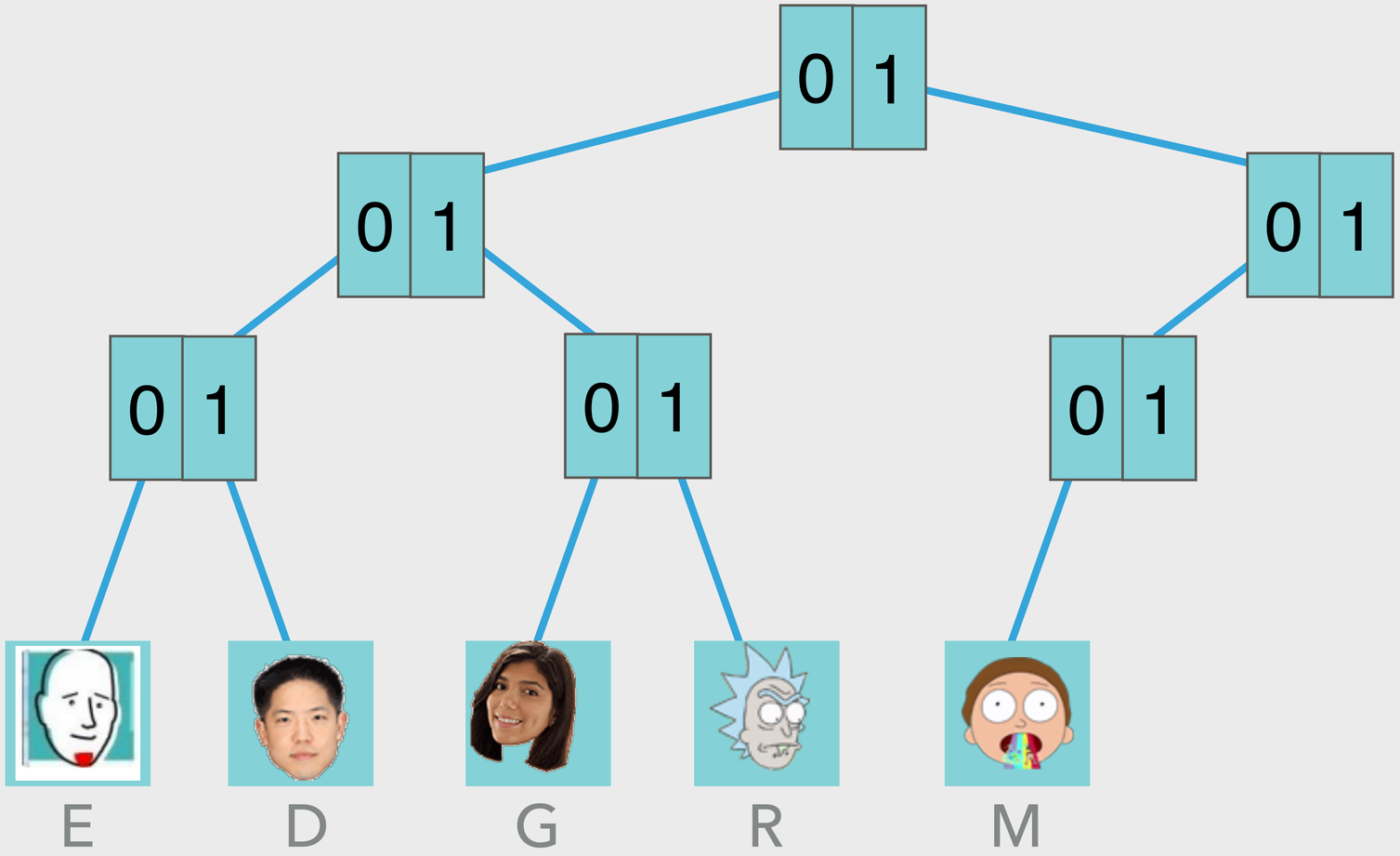


UPDATE



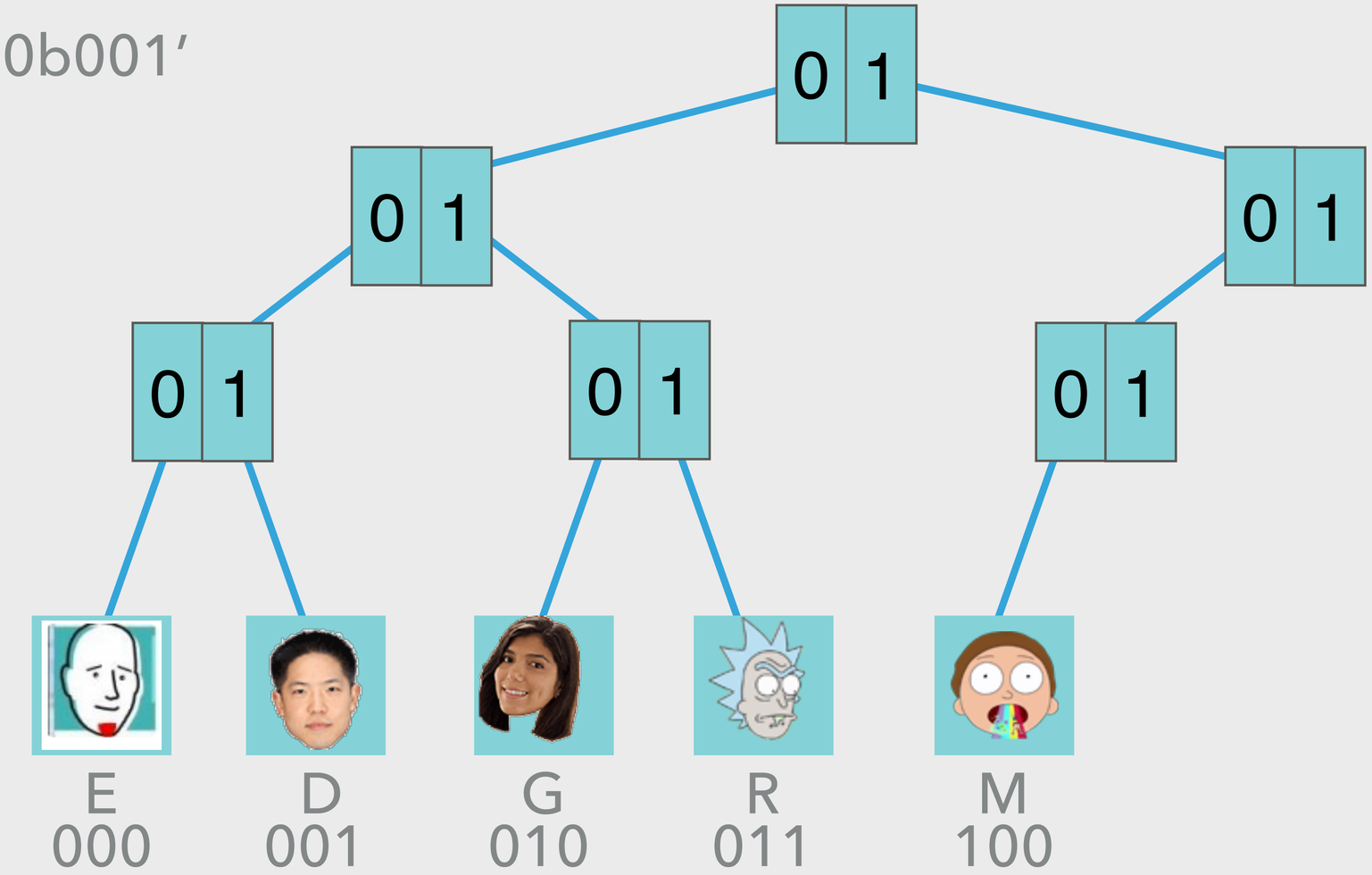
BITMAPPED VECTOR TRIE

WHAT ABOUT OBJECTS?


0 1
0 1
0 1
0 1
0 1
0 1
E
D
G
R
M

HASH KEYS

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



HASH ARRAY MAPPED TRIE

PATH COPYING

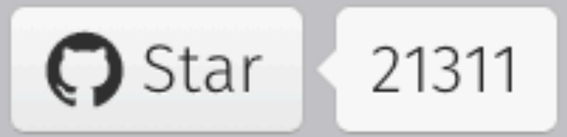
STRUCTURAL SHARING


Worry native performance no more...
SMALL & FAST

HOW TO USE ... IN JS

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

IMMUTABLE



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:

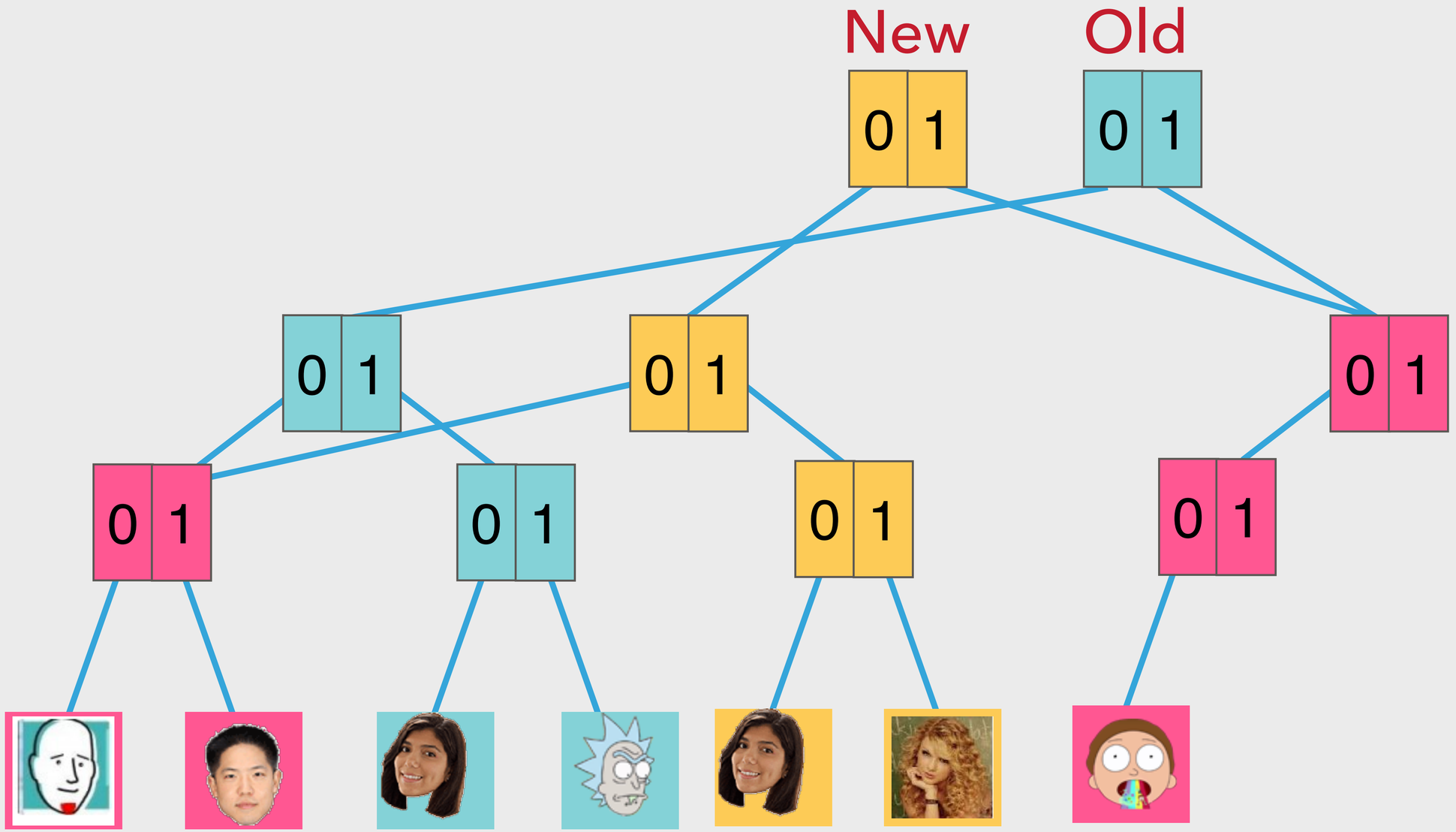
- Efficient immutable data structures - no cloning required
- Uniform iteration for all types
- Value based equality

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


New
Old
0 1
0 1
0 1
0 1
0 1
0 1
0 1
0 1
0 1

NO MORE MUTABILITY!!

REFERENCES

Persistent Data Structures and Managed References - *Rich Hickey*

React.js Conf 2015 - Immutable Data and React - *Lee Byron, Facebook*

Immutable data structures for functional JS | JSConf EU 2017 - *Anjana Vakil*