Functional Programming: What? Why? How?

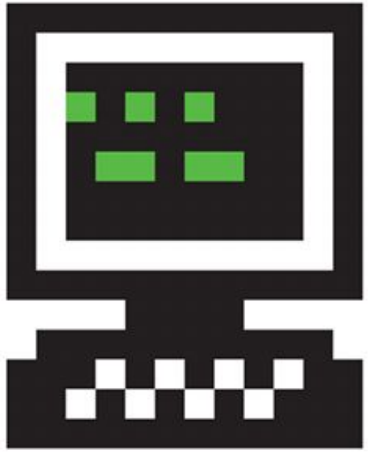
@AnjanaVakil

HolyJS 2017

Who?

@AnjanaVakil

The Recurse Center

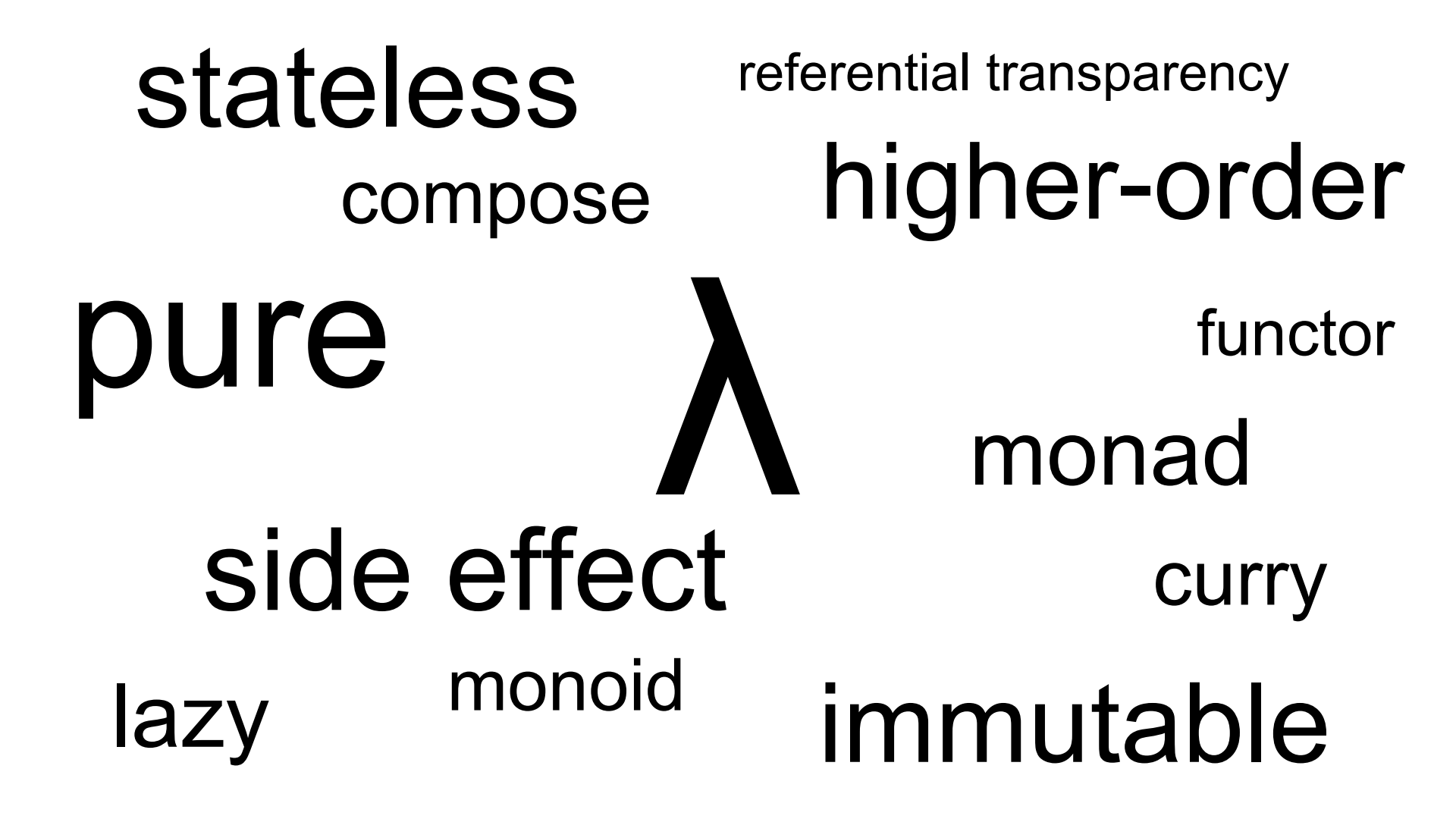


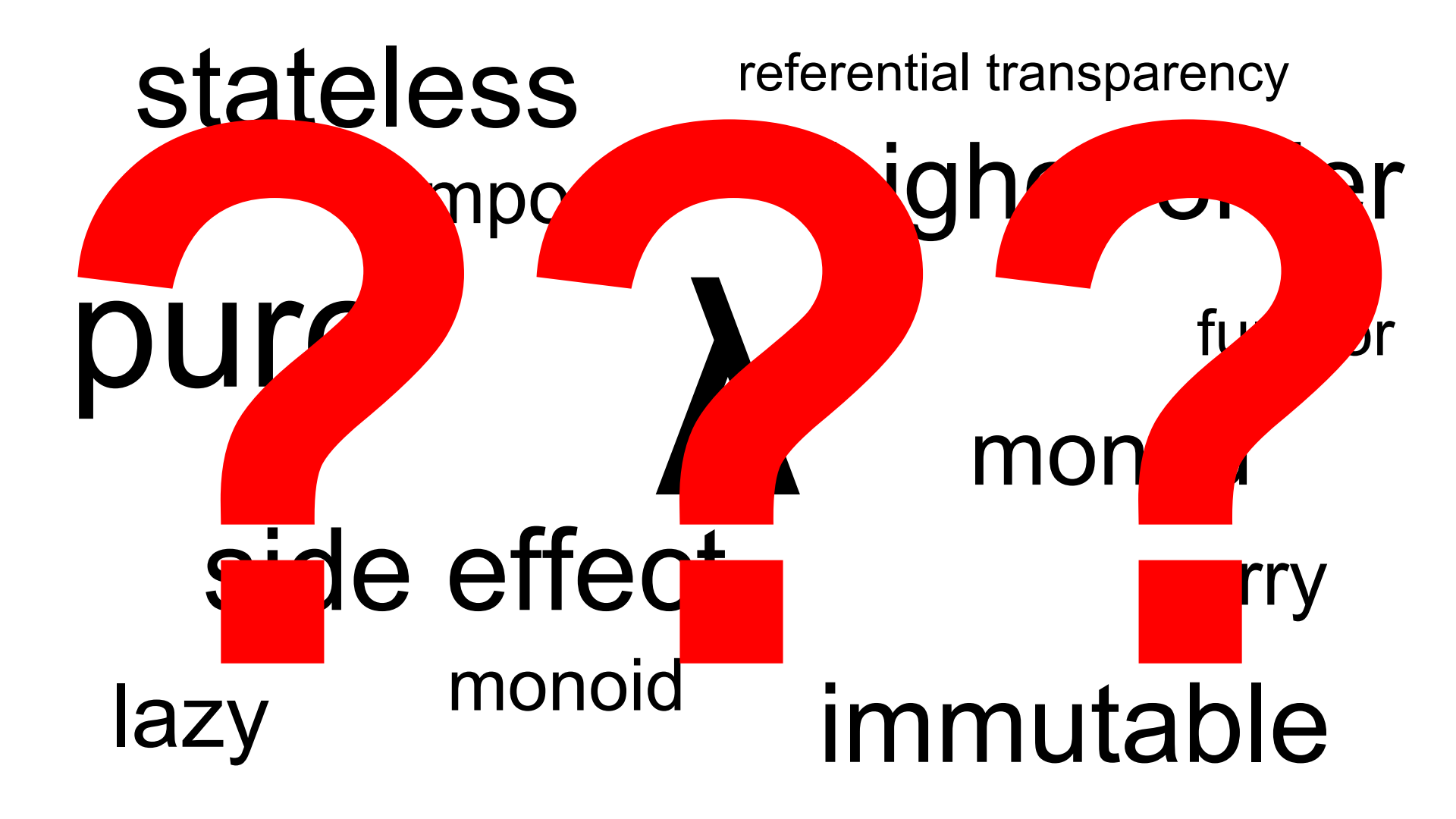
https://www.recurse.com/

What is functional programming?

What is functional programming?

a buzz-wordy trend


stateless
referential transparency
compose
higher-order
pure
λ
functor
monad
side effect
curry
monoid
lazy
immutable


stateless
referential transparency
pur
λ
side effect
monoid
lazy
immutable

What is functional programming?

a coding style
supported by some languages

F#

Erlang

Haskell

JavaScript!

Clojure

Elm

Scala

OCaml

What is functional programming?

a programming paradigm
(worldview, mindset)

Imperative

follow my commands
do this, then that

Declarative

this is what I want
I don't care how you do it

Object-Oriented

keep state to yourself
send/receive messages

Functional

???

What is functional programming?

one simple idea

CODE WORDS – ISSUE ONE

An introduction to functional programming

Mary Rose Cook

Many functional programming articles teach abstract functional techniques. That is, composition, pipelining, higher order functions. This one is different. It shows examples of imperative, unfunctional code that people write every day and translates these examples to a functional style.

https://codewords.recurse.com/issues/one/an-introduction-to-functional-programming

pure functions

only input in
only output out

Not pure:

```
var name = "Saint Petersburg";

function greet() {
   console.log("Hello, " + name + "!");
}

greet(); // Hello, Saint Petersburg!

name = "HolyJS";
greet(); // Hello, HolyJS!
```

Pure:

```
function greet(name) {
   return "Hello, " + name + "!";
}

greet("Saint Petersburg");
// "Hello, Saint Petersburg!"

greet("HolyJS");
// "Hello, HolyJS!"
```

Why functional programming?

Why functional programming?

more predictable, safer

Why functional programming?

easier to test/debug

Why functional programming?

~~makes you look/feel smart~~
~~is The Best™ paradigm~~

Why functional JavaScript?

Why functional JavaScript?

object-oriented JS gets tricky
(prototypes? `this`?!?)

Why functional JavaScript?

established community/tools

OK, let’s do it!

OK, let’s do it!

...how?

Do everything with functions

program === function(s)

Imperative:

```
var city = "St. Petersburg";
var greeting = "Hi";

console.log(greeting + ", " + city + "!");
// Hi, St. Petersburg!

greeting = "Привет";
console.log(greeting + ", " + city + "!");
// Привет, St. Petersburg!
```

Functional:

```
function greet(greeting, name) {
   return greeting + ", " + name + "!";
}

greet("Hi", "St. Petersburg");
// "Hi, St. Petersburg!"

greet("Привет", "HolyJS");
// "Привет, HolyJS!"
```

Avoid side effects

do nothing but return output

Side effects:

```
var conf = {name: "SaintJS", date: 2017};

function renameConf(newName) {
   conf.name = newName;
   console.log("Renamed!");
}

renameConf("HolyJS"); // Renamed!
conf; // {name: "HolyJS", date: 2017}
```

No side effects:

```
var conf = {name: "SaintJS", date: 2017};

function renameConf(oldConf, newName) {
   return {name: newName, date: oldConf.date}
}

var newConf = renameConf(conf, "HolyJS");
newConf; // {name: "HolyJS", date: 2017}
conf;    // {name: "SaintJS", date: 2017}
```

Avoid mutability

don't change in-place;
instead, replace

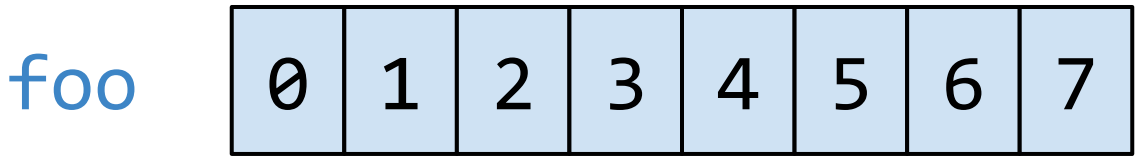
Mutation (dangerous!):

```
var numSysts = ["Roman", "Arabic", "Chinese"];

numSysts[1] = "Hindu-" + numSysts[1];

numSysts;
// ["Roman", "Hindu-Arabic", "Chinese"]
```

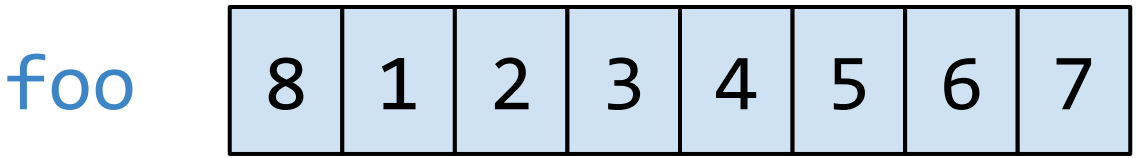
No mutation (safer!):

```
const numSysts = ["Roman", "Arabic", "Chinese"];

const newNumSysts = numSysts.map((num) => {
  if (num === "Arabic") { return "Hindu-" + num; }
  else { return num; }
});

newNumSysts;//["Roman", "Hindu-Arabic", "Chinese"]
numSysts;   //["Roman", "Arabic", "Chinese"]
```

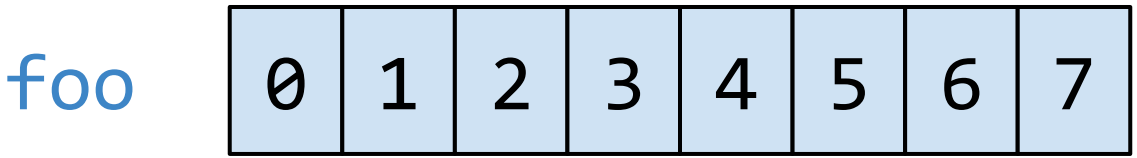
Mutable data



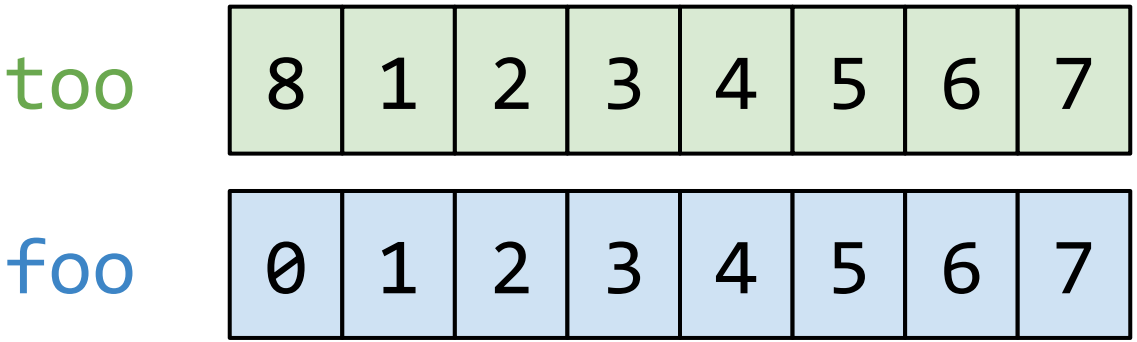
Mutable data



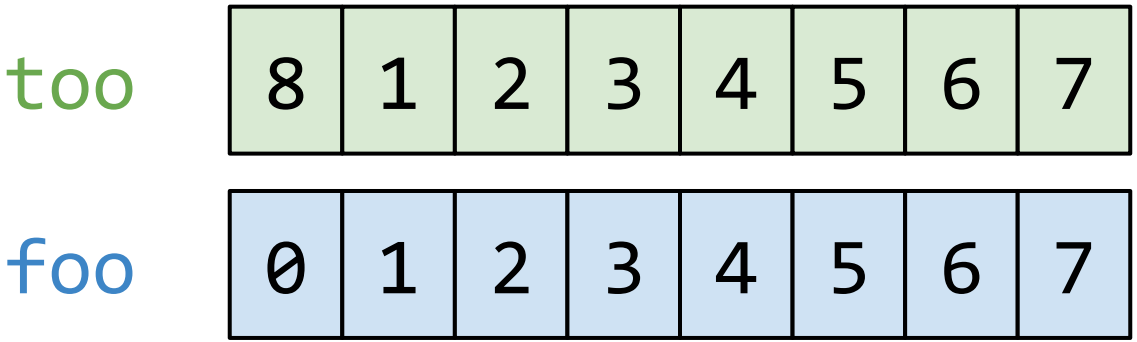
Immutable data



Immutable data



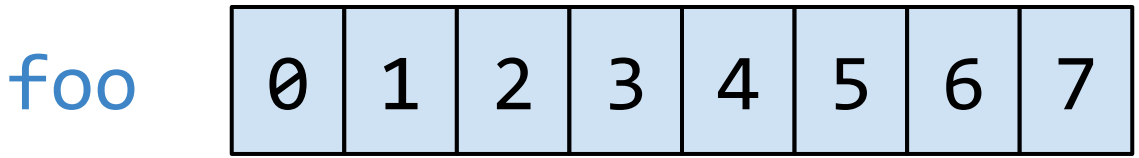
Immutable data (inefficient)



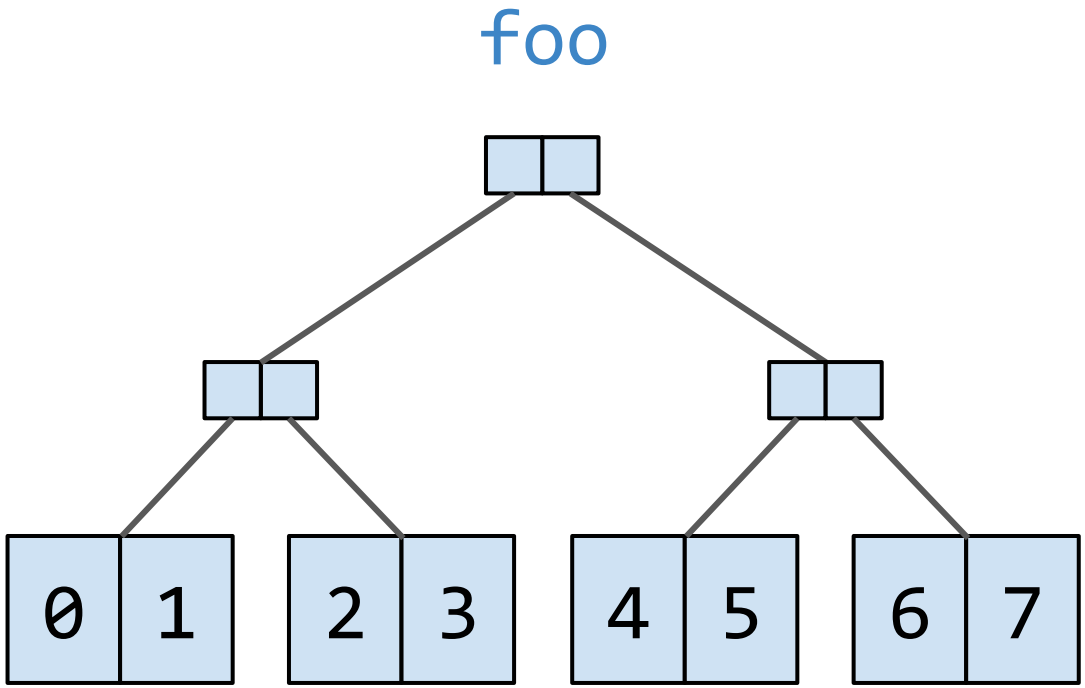
Use persistent data structures

for efficient immutability

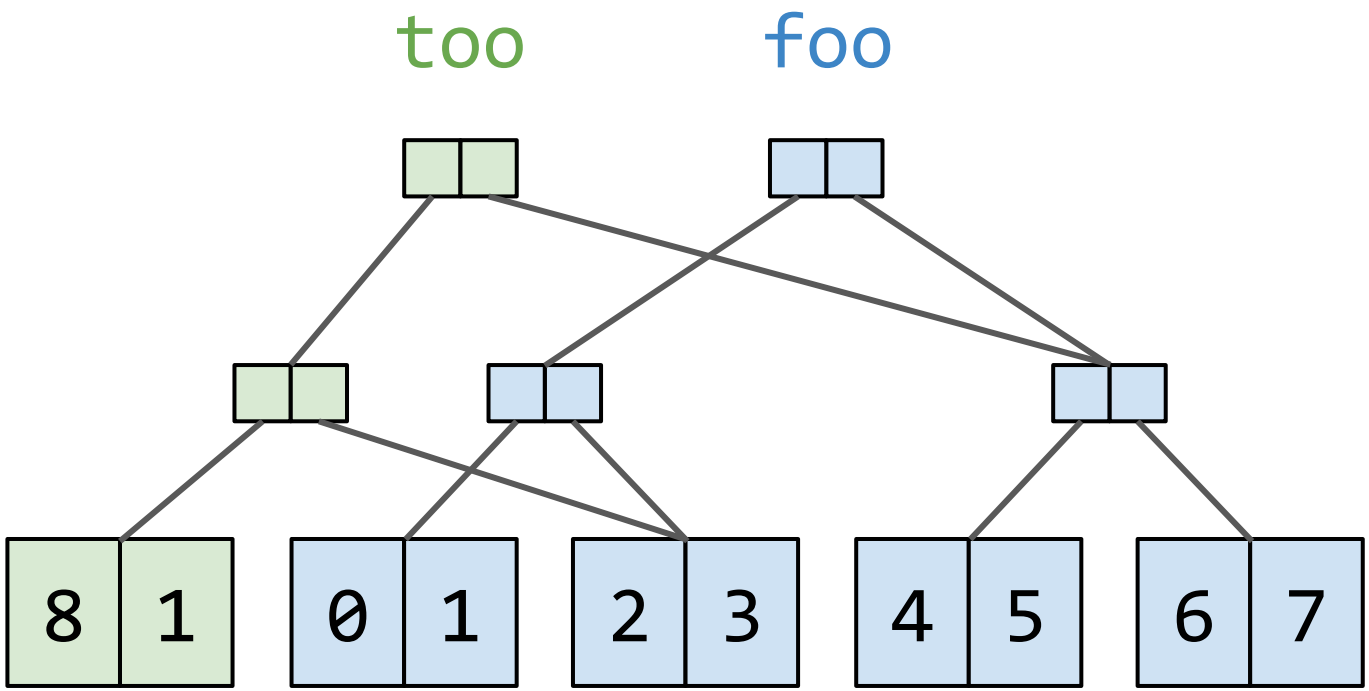
Immutable data (efficient)



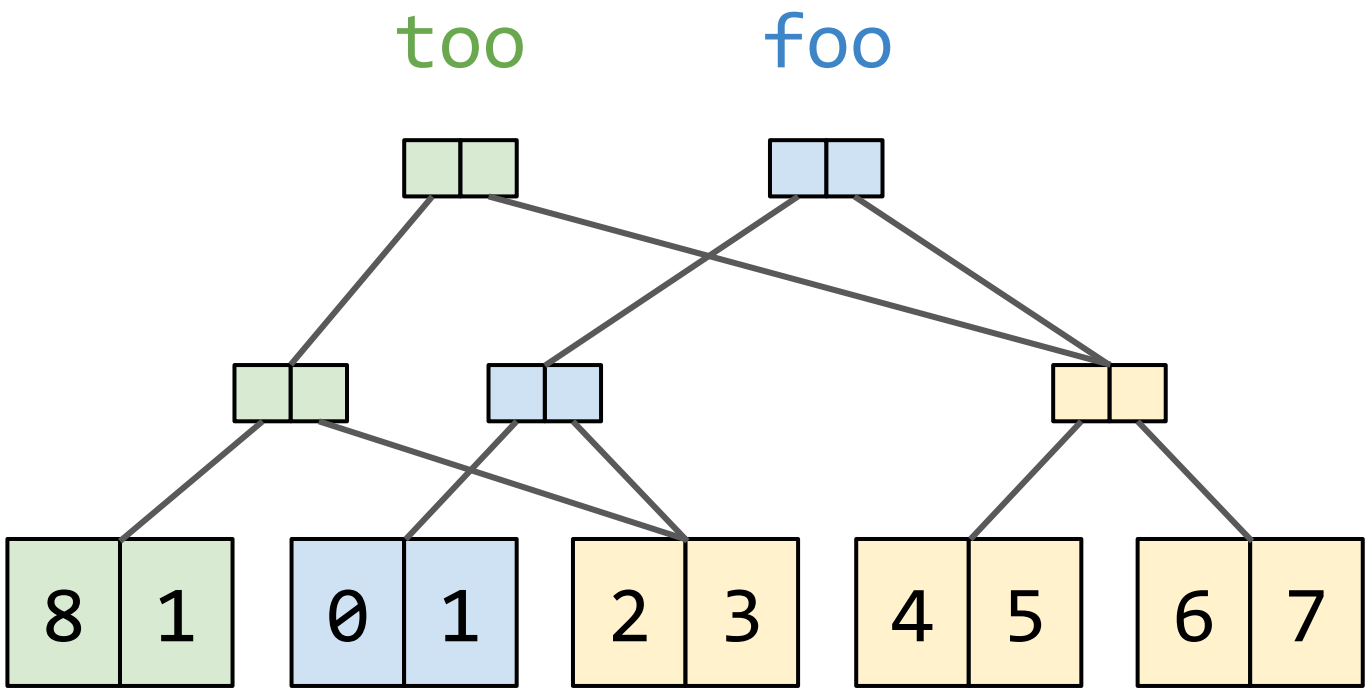
Immutable data (efficient)



Immutable data (efficient)



Persistent (immutable) data structures



structural sharing

Persistent data structures

Mori

https://swannodette.github.io/mori

```
var f = mori.vector(1,2);
var t = mori.conj(f, 3);
```

- ClojureScript port
- Functional API
- Fast

Immutable.js

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

```
var f = Immutable.List.of(1,2);
var t = f.push(3);
```

- JS through & through
- Public methods
- A bit smaller than Mori

Don’t iterate

recurse

Iteration:

```
function sum (numbers) {
  let total = 0;
  for (i = 0; i < numbers.length; i++) {
    total += numbers[i];
  }
  return total;
}

sum([0,1,2,3,4]); // 10
```

Recursion:

```
function sum (numbers) {
  if (numbers.length === 1) {
    return numbers[0];
  } else {
    return numbers[0] + sum(numbers.slice(1));
  }
}

sum([0,1,2,3,4]); // 10
```

Don’t loop

use `map`, `reduce`, `filter`

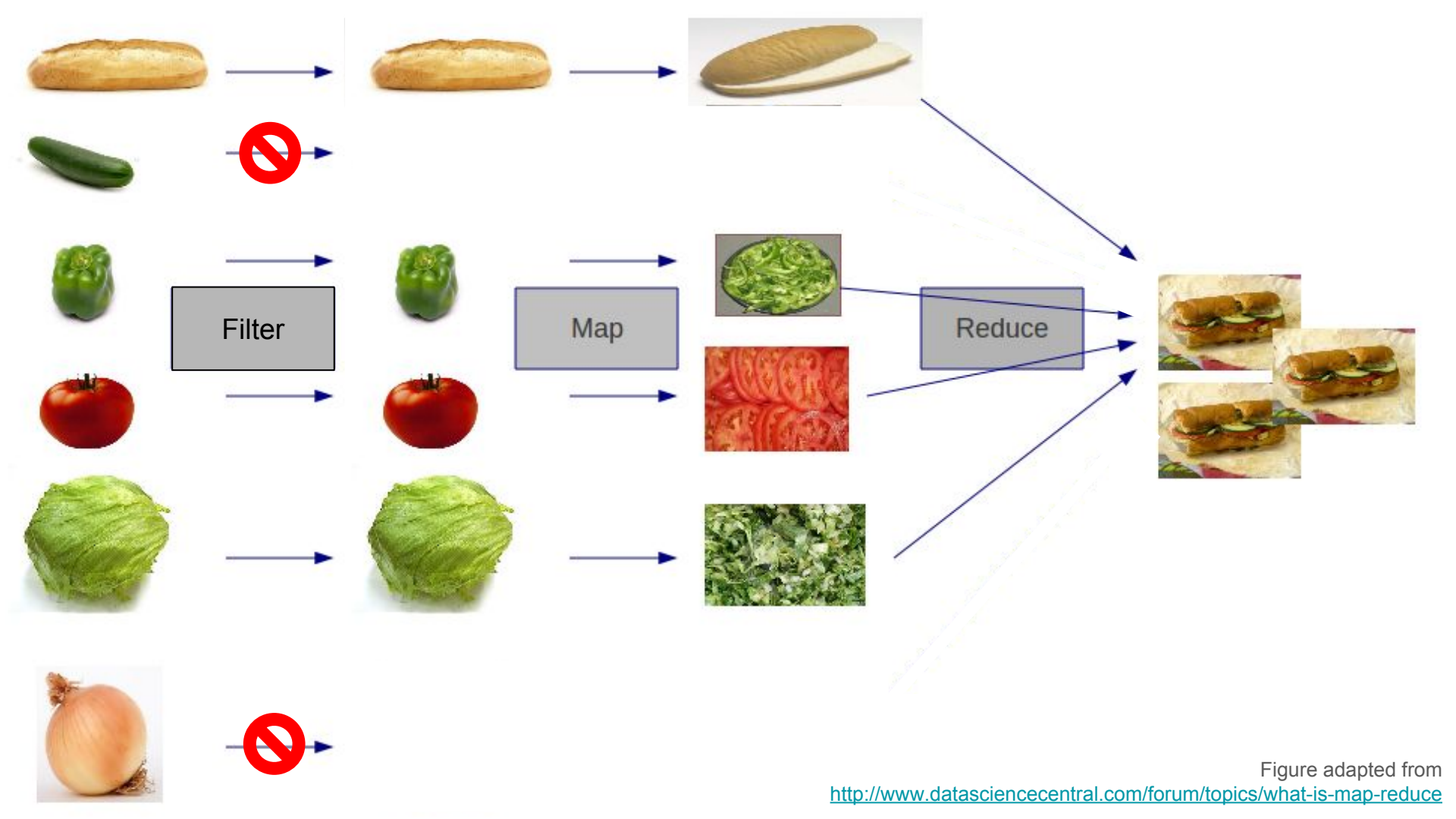


Figure adapted from
http://www.datasciencecentral.com/forum/topics/what-is-map-reduce

Use higher-order functions

functions with functions
as inputs/outputs

Closures:

```
function makeAdjectifier(adjective) {
   return function (noun) {
     return adjective + " " + noun;
   };
}

var holify = makeAdjectifier("holy");
holify("JS");  // "holy JS"
holify("cow"); // "holy cow"
```

Flow data through functions

outputs become inputs become outputs

Outputs become next inputs:

```
var ender = (ending) => (input) => input + ending;

var adore = ender(" rocks");
var announce = ender(", y'all");
var exclaim = ender("!");

var hypeUp = (x) => exclaim(announce(adore(x)));
hypeUp("JS"); // "JS rocks, y'all!"
hypeUp("FP"); // "FP rocks, y'all!"
```

Function composition/pipelining:

```
var r = require("ramda");

var rtlHype = r.compose(adore, announce, exclaim);
rtlHype("FP");  // "FP!, y'all rocks"

var ltrHype = r.pipe(adore, announce, exclaim);
ltrHype("FP");  // "FP rocks, y'all!"
```

FP libraries for JS

- Mori (swannodette.github.io/mori)
- Immutable.js (facebook.github.io/immutable-js)
- Ramda (ramdajs.com)
- Underscore (underscorejs.org)
- Lodash (lodash.com)
- ...and more!

Further reading/watching

Code Words by the Recurse Center (codewords.recurse.com)

- Mary Rose Cook, "An introduction to functional programming"
- Sal Becker, "This just isn't functional"
- Patrick Dubroy, "Immutability is not enough"

J. Kerr, “Functional Principles for OO Development”, GOTO Chicago '14 youtu.be/GpXsQ-NIKXY

Me, "Immutable data structures for functional JS", JSConf EU '17 youtu.be/Wo0qiGPSV-s

D. Nolen, “Immutability, interactivity & JS”, FutureJS '14 youtu.be/mS264h8KGwk

L. Byron, “Immutable Data & React”, React.js Conf '15 youtu.be/I7IdS-PbEgI

B. Stokke, "The Miracle of Generators", GOTO Chicago '16 youtu.be/6mCkLZ0cwAI

Thanks for listening!

I'm @AnjanaVakil

Huge thanks to:

Mary Rose Cook, Sal Becker,
Khalid Ali & The Recurse Center
HolyJS organizers