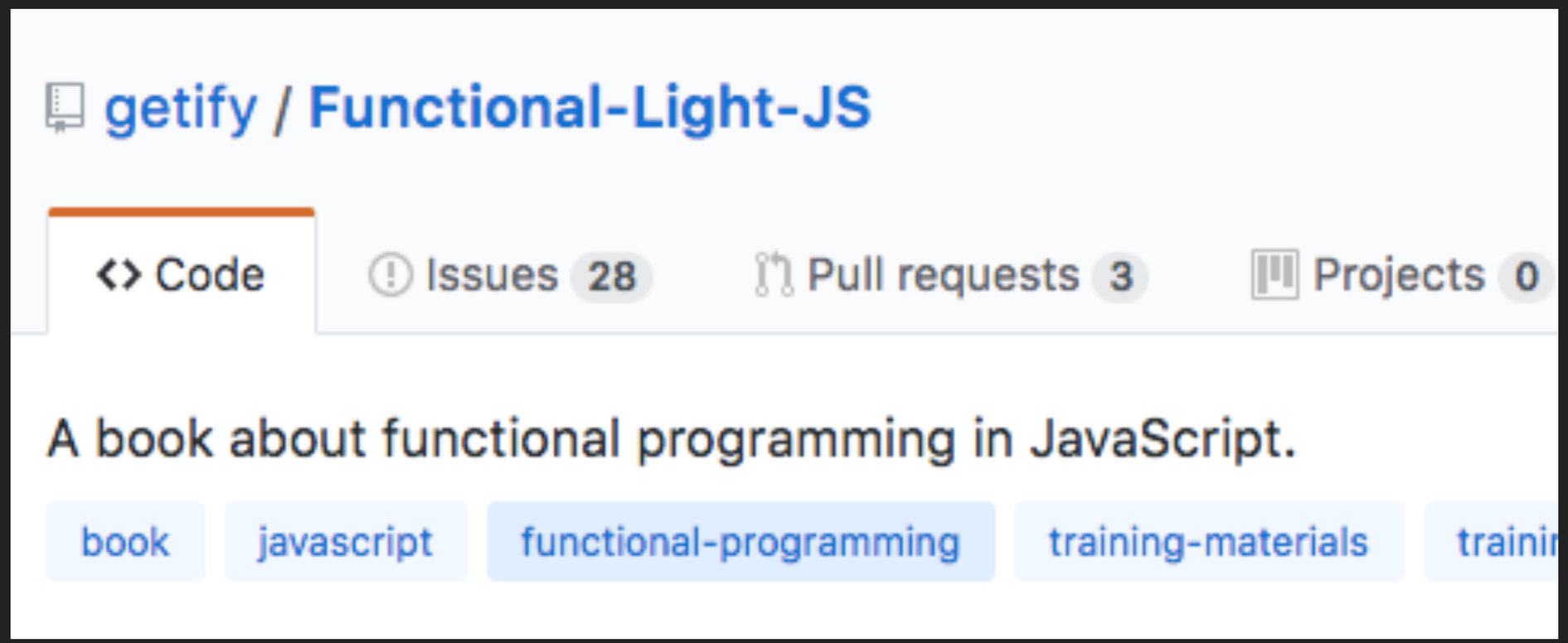
KYLE SIMPSON GETIFY@GMAIL.COM

FUNCTIONAL-LIGHT JS



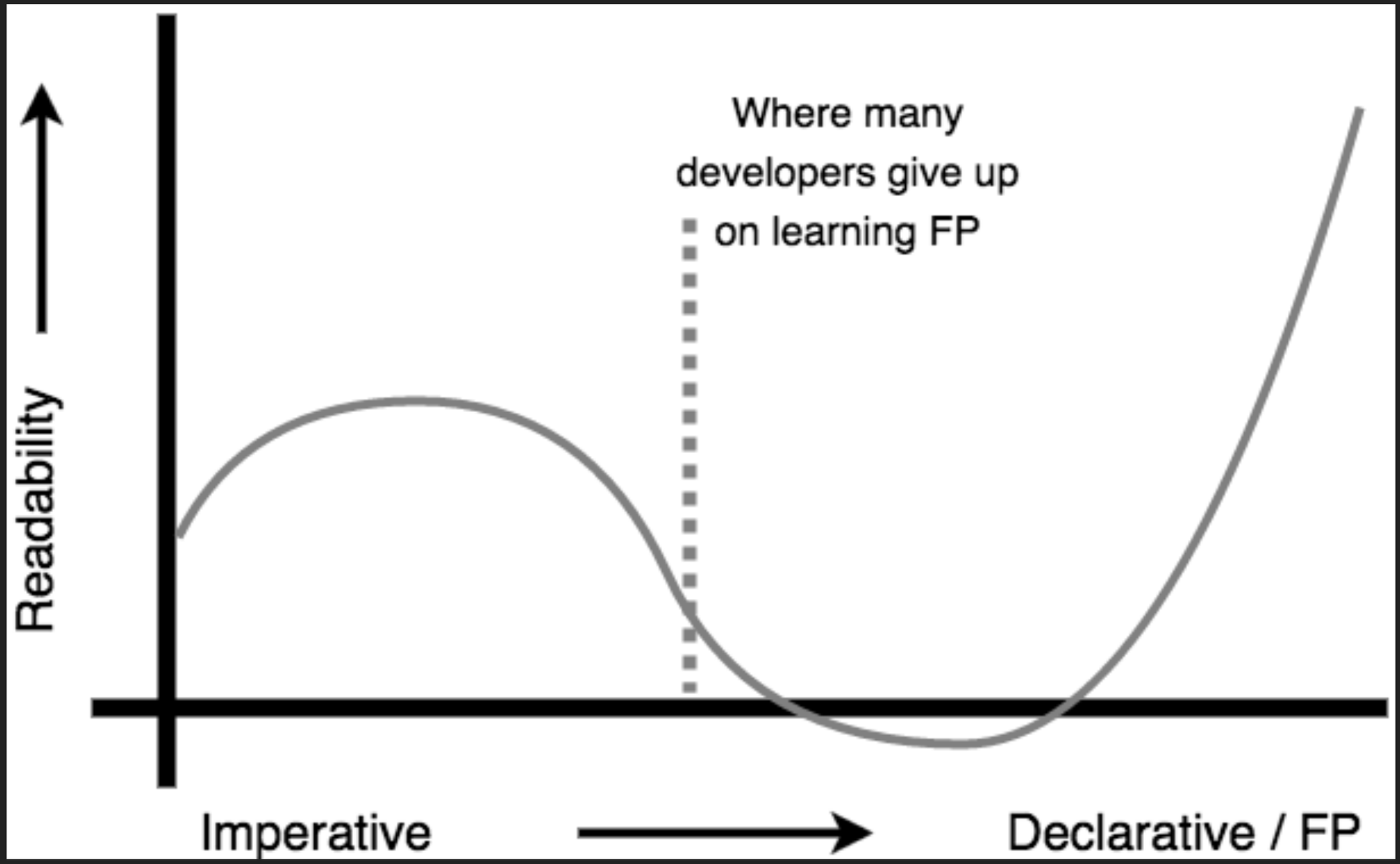
github.com/getify/Functional-Light-JS

WHY FP?

IMPERATIVE

VS

DECLARATIVE


Where many
developers give up
on learning FP
Readability
Imperative
Declarative / FP

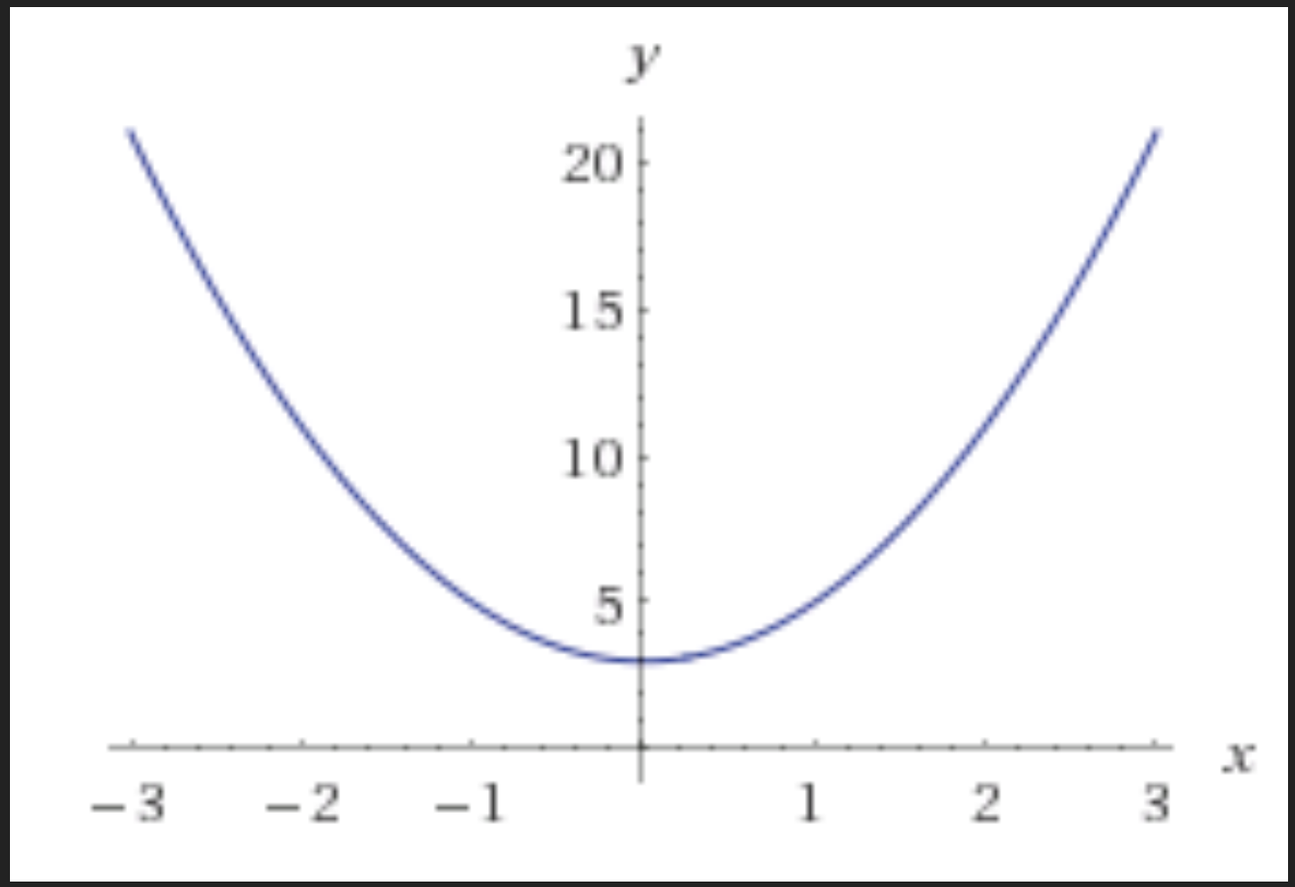
PROVABLE

LESS TO READ

FUNCTIONS

```
function foo(x,y,z,w) {
    console.log( x, y, z, w );
}

function bar(x = 2,...args) {
    return foo(x,42,...args);
}


bar();              // 2 42 undefined undefined

bar(3,8,11);        // 3 42 8 11

bar(...[6,5]);      // 6 42 5 undefined
```

```
function foo(x,y) {
    return [x + 1, y - 1];
}

var [a,b] = foo(...[5,10]);

a;                  // 6
b;                  // 9
```



$$f(x) = 2x^2 + 3$$

```
function f(x) {
    return 2 * Math.pow(x,2) + 3;
}
```

```
1  function f() {
2      y = 2 * Math.pow(x,2) + 3;
3  }
4
5  var x, y;
6
7  x = 0;
8  f();
9  y;                    // 3
10
11 x = 2;
12 f();
13 y;                    // 11
```

~~SIDE EFFECTS~~

```
1  function f(x) {
2      return 2 * Math.pow(x,2) + 3;
3  }
4
5  var y;
6
7  y = f(0);
8  // 3
9
10 y = f(2);
11 // 11
12
13 y = f(-1);
14 // 5
```

PURE FUNCTIONS

```
1 // pure
2 function foo(x,y) {
3     return x + y;
4 }
5
6 // impure
7 function bar(x,y) {
8     return x + y + z;
9 }
```

PURIFYING

```
1  function f() {
2      y = 2 * Math.pow(x,2) + 3;
3  }
4
5  var x, y;
6
7  x = 0;
8  f();
9  y;              // 3
10
11 x = 2;
12 f();
13 y;              // 11
```

```
function F(x) {
    var y;
    f(x);
    return y;

    function f() {
        y = 2 * Math.pow(x,2) + 3;
    }
}

var y;

y = F(0);
// 3

y = F(2);
// 11
```

```
function f() {
    y = 2 * Math.pow(x,2) + 3;
}

function F(curX) {
    var [origX,origY] = [x,y];
    x = curX;
    f();
    var newY = y;
    [x,y] = [origX,origY];
    return newY;
}

var x, y;

F(0);
// 3

F(2);
// 11
```

EXERCISE 1

```
1 const y = 1;
2
3 function foo(x) {
4     return x + y;
5 }
6
7 foo(1);          // 2
```

```
1 function foo(x) {
2     return bar(x);
3 }
4
5 function bar(y) {
6     return y + 1;
7 }
8
9 foo(1);          // 2
```

```
1 function foo(bar) {
2         return function(x){
3             return bar(x);
4         };
5 }
6
7 foo(function(v){
8         return v * 2;
9 })(3);
10 // 6
```

```
1 function getId(obj) {
2     return obj.id;
3 }
4
5 getId({
6     get id() {
7         return Math.random();
8     }
9 });
```

ARGUMENTS

```
1 // unary
2 function increment(x) {
3     return sum(x,1);
4 }
5
6 // binary
7 function sum(x,y) {
8     return x + y;
9 }
```

```
function unary(fn) {
    return function one(arg){
        return fn(arg);
    };
}

function binary(fn) {
    return function two(arg1,arg2){
        return fn(arg1,arg2);
    };
}

function f(...args) {
    return args;
}

var g = unary(f);
var h = binary(f);

g(1,2,3,4);     // [1]
h(1,2,3,4);     // [1,2]
```

```
function flip(fn) {
    return function flipped(arg1,arg2,...args){
        return fn(arg2,arg1,...args);
    };
}

function f(...args) {
    return args;
}

var g = flip(f);

g(1,2,3,4);     // [2,1,3,4]
```

```
function reverseArgs(fn) {
    return function reversed(...args){
        return fn(...args.reverse());
    };
}

function f(...args) {
    return args;
}

var g = reverseArgs(f);

g(1,2,3,4);     // [4,3,2,1]
```

```
function spreadArgs(fn) {
    return function spread(args){
        return fn(...args);
    };
}

function f(x,y,z,w) {
    return x + y + z + w;
}

var g = spreadArgs(f);

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

POINT-FREE

```
1 foo(function(v){
2         return bar(v);
3 });
4
5 foo(bar);
6
```

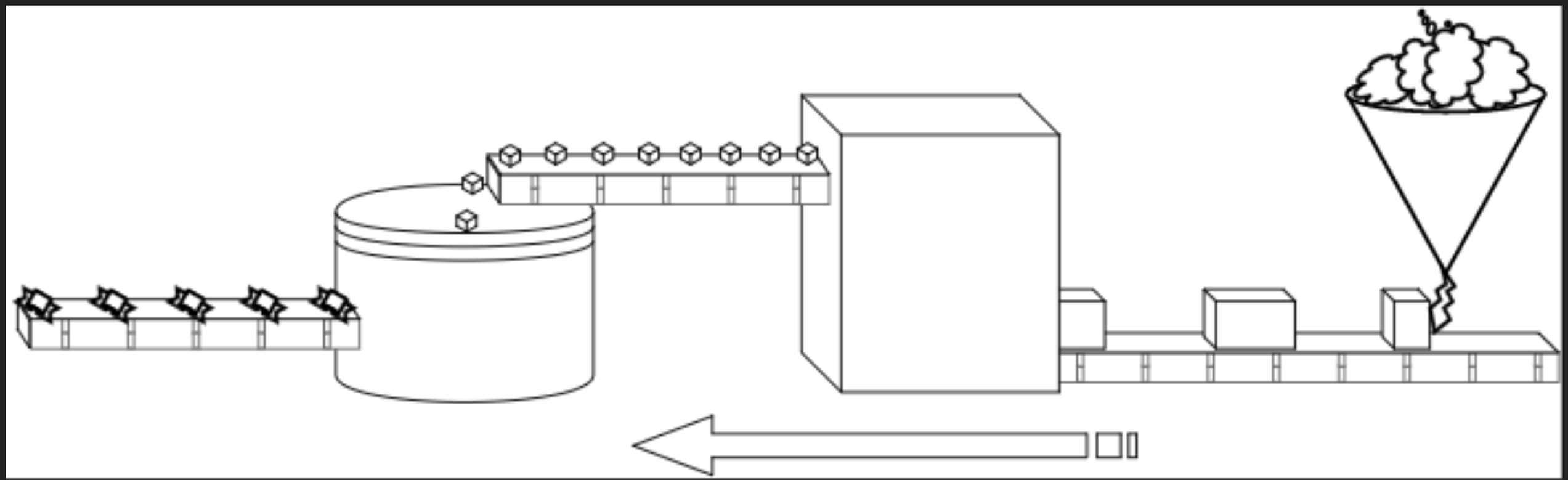
```
1 function isOdd(v) {
2     return v % 2 == 1;
3 }
4
5 function isEven(v) {
6     return !isOdd(v);
7 }
8
9 isEven(4);          // true
```

```
function not(fn) {
    return function negated(...args){
        return !fn(...args);
    };
}

function isOdd(v) {
    return v % 2 == 1;
}

var isEven = not(isOdd);

isEven(4);          // true
```

EXERCISE 2

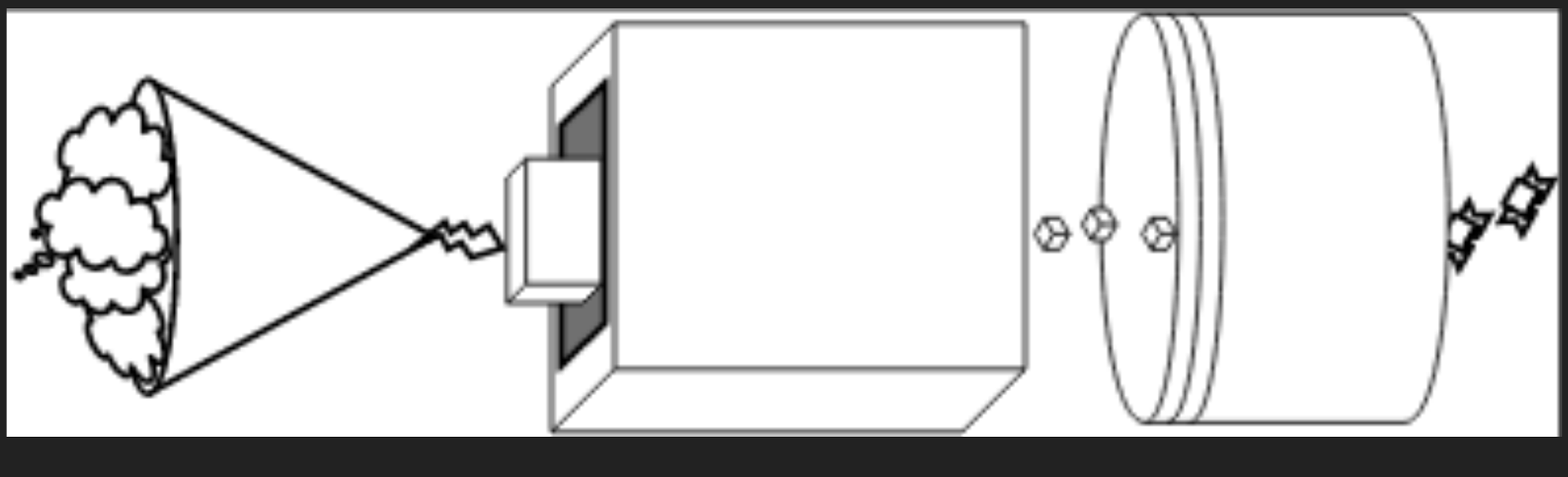
COMPOSITION

```
1  function sum(x,y) {
2      return x + y;
3  }
4
5  function mult(x,y) {
6      return x * y;
7  }
8
9  // (3 * 4) + 5
10 var x_y = mult( 3, 4 );
11 sum( x_y , 5 );                    // 17
```

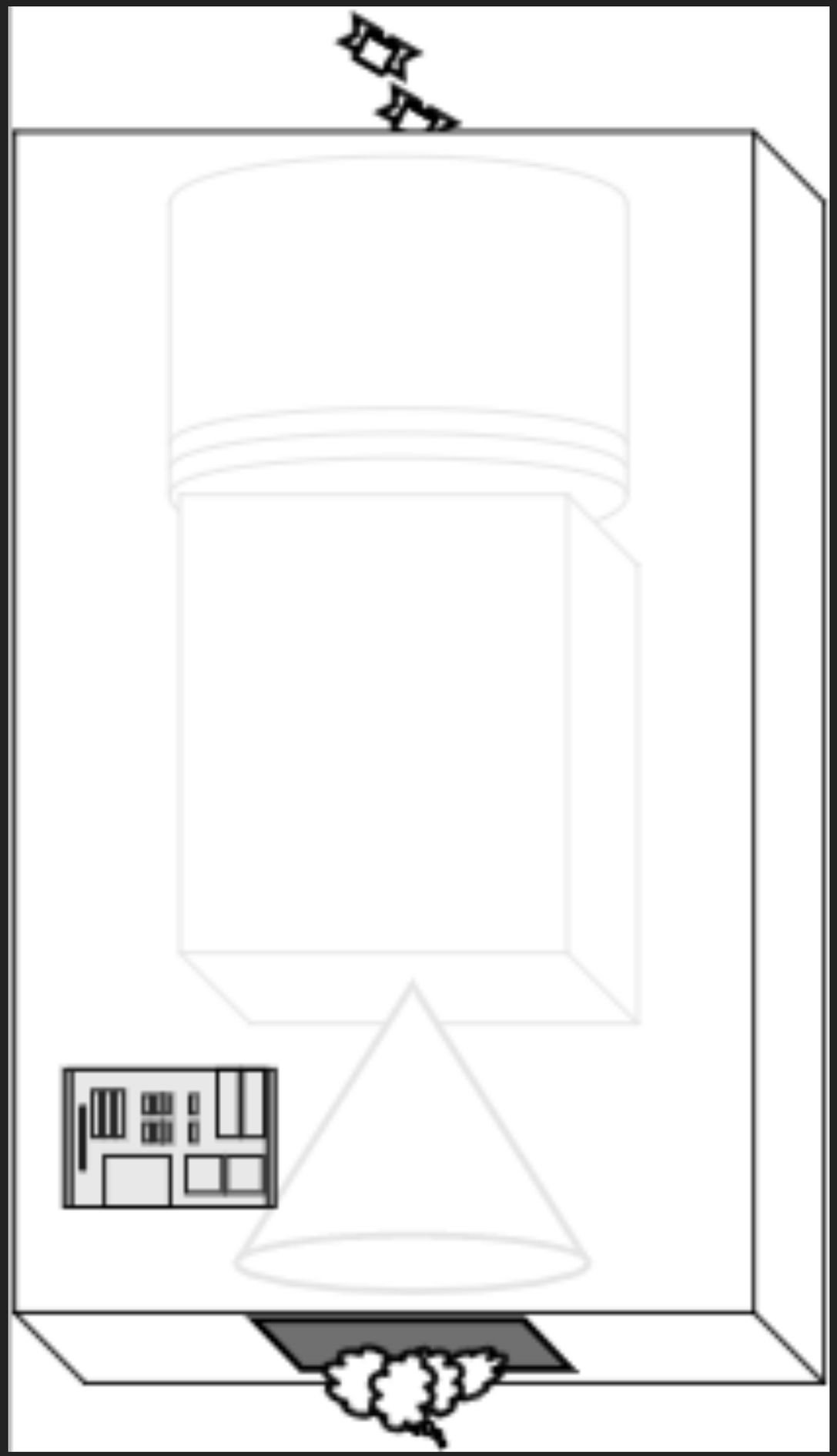


(RIGHT-TO-LEFT)

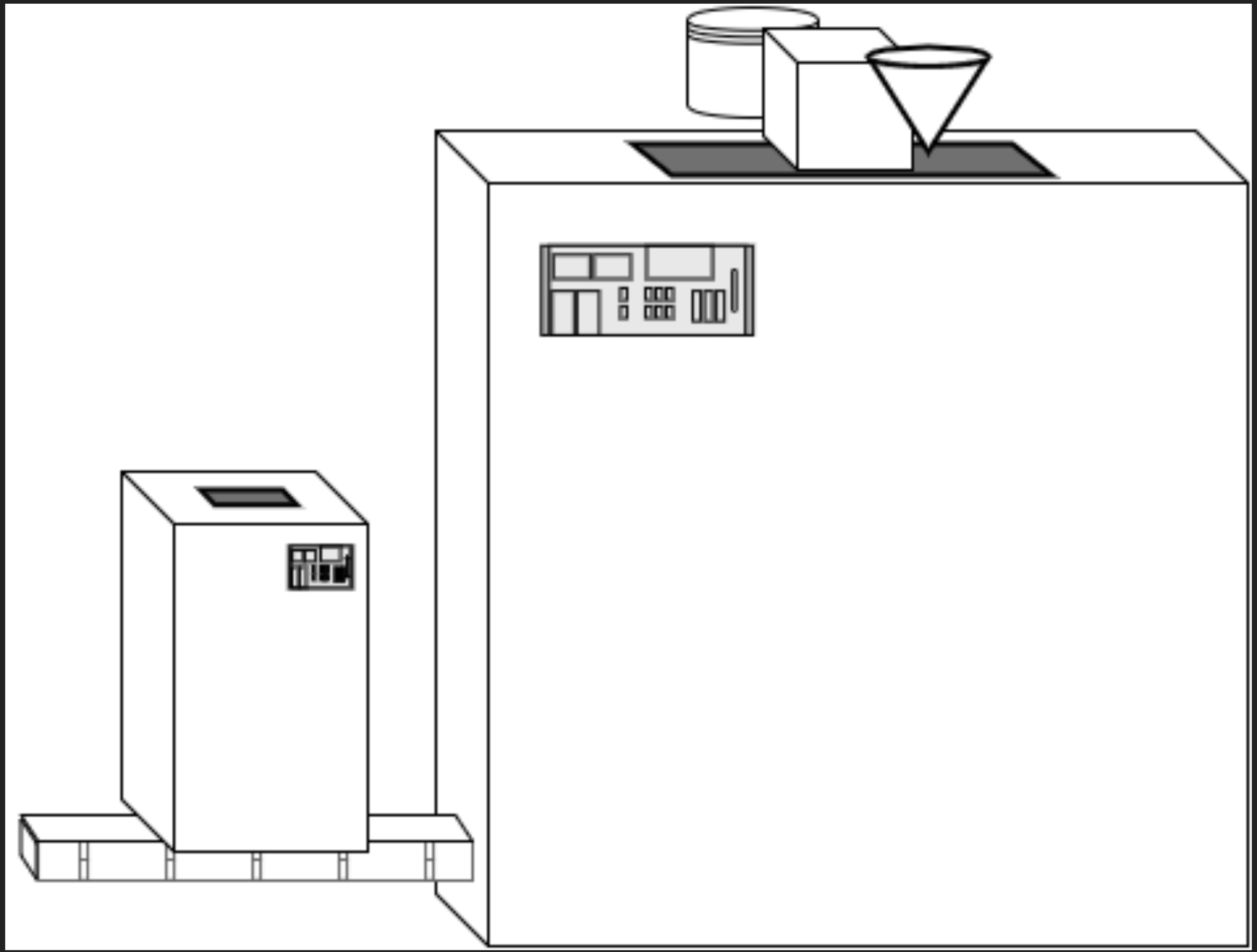
```
1 function sum(x,y) {
2     return x + y;
3 }
4
5 function mult(x,y) {
6     return x * y;
7 }
8
9 // (3 * 4) + 5
10 sum( mult( 3, 4 ), 5 ); // 17
```



```
function sum(x,y) {
    return x + y;
}

function mult(x,y) {
    return x * y;
}

function multAndSum(x,y,z) {
    return sum( mult( x, y ), z );
}

// (3 * 4) + 5
multAndSum(3,4,5);          // 17
```



```
function sum(x,y) {
    return x + y;
}

function mult(x,y) {
    return x * y;
}

function pipe2(fn1,fn2) {
    return function piped(arg1,arg2,arg3){
        return fn2(
            fn1(arg1,arg2),
            arg3
        );
    };
}

var multAndSum = pipe2(mult,sum);

// (3 * 4) + 5
multAndSum(3,4,5);      // 17
```



```
1 foo(bar(baz(2)));
2
3 compose(foo,bar,baz)(2);
4
5 pipe(baz,bar,foo)(2);
```

(RIGHT-TO-LEFT)

(LEFT-TO-RIGHT)

```
function composeRight(fn2,fn1) {
    return function comp(...args){
        return fn2(fn1(...args));
    };
}
```

```
function increment(x) {
    return x + 1;
}

function double(x) {
    return x * 2;
}

var f = composeRight(increment,double);
var p = composeRight(double,increment);

f(3);    // 7
p(3);    // 8
```

EXERCISE 3

IMMUTABILITY

```
1 var x = 2;
2 x++;                          // allowed
3
4 const y = 3;
5 y++;                          // not allowed
6
7 const z = [4,5,6];
8 z = 10;                       // not allowed
9 z[0] = 10;                    // allowed!
```

```
1 {
2       const y = 3;
3       var x = x + y;
4       var z = foo(x);
5       bar(x,y,z);
6       z = [x*2,y*3,z*4];
7       bam(z);
8 }
```

```
1 var z = Object.freeze([4,5,6,[7,8,9]]);
2
3 z[0] = 10;          // not allowed
4 z[3][0] = 10;       // allowed!
```

```
1  var state = Immutable.List.of( 1, 2, 3, 4 );
2  var newState = state.set( 42, "meaning of life" );
3
4  state === newState;                          // false
5
6  state.get( 2 );                              // 3
7  newState.get( 2 );                           // 3
8
9  state.get( 42 );                             // undefined
10 newState.get( 42 );                          // "meaning of life"
11
12 newState.toArray().slice( 1, 3 );            // [2,3]
```

facebook.github.io/immutable-js

```
1  function doubleThemMutable(list) {
2      for (var i=0; i<list.length; i++) {
3          list[i] = list[i] * 2;
4      }
5  }
6
7  var arr = [3,4,5];
8  doubleThemMutable(arr);
9
10 arr;      // [6,8,10]
```

```
function doubleThemImmutable(list) {
    var newList = [];
    for (var i=0; i<list.length; i++) {
        newList[i] = list[i] * 2;
    }
    return newList;
}

var arr = [3,4,5];
var arr2 = doubleThemImmutable(arr);

arr;     // [3,4,5]
arr2;    // [6,8,10]
```

EXERCISE 4

CLOSURE

Closure is when a function "remembers" the variables around it even when that function is executed elsewhere.

```
1 function unary(fn) {
2     return function one(arg){
3         return fn(arg);
4     };
5 }
```

```
1 function composeRight(fn2,fn1) {
2     return function comp(...args){
3         return fn2(fn1(...args));
4     };
5 }
```

EXERCISE 5

GENERALIZED
TO SPECIALIZED

```
1  function add(x,y) {
2      return x + y;
3  }
4
5  function partial(fn,...firstArgs) {
6      return function applied(...lastArgs){
7          return fn(...firstArgs,...lastArgs);
8      };
9  }
10
11 var addTo10 = partial(add,10);
12
13 addTo10(32);     // 42
```

```
1  var add3 = curry(function add3(x,y,z) {
2      return x + y + z;
3  });
4
5  var f = add3(3);
6
7  var p = f(4);
8
9  p(5);                 // 12
10
11 add3(3)(4)(5);   // 12
```

RECURSION

```
function sumIter(...nums) {
    var sum = 0;
    for (var i=0; i<nums.length; i++) {
        sum = sum + nums[i];
    }
    return sum;
}

sumIter(3,4,5,6,7,8,9);      // 42
```

```
function sumIter(sum,...nums) {
    for (var i=0; i<nums.length; i++) {
        sum = sum + nums[i];
    }
    return sum;
}

sumIter(3,4,5,6,7,8,9);     // 42
```

```
function sumRecur(sum, ...nums) {
    if (nums.length == 0) return sum;
    return sum + sumRecur(...nums);
}

sumRecur(3,4,5,6,7,8,9);     // 42
```

```
function sumRecur(sum,num,...nums) {
    if (nums.length == 0) return sum + num;
    return sum + sumRecur(num,...nums);
}

sumRecur(3,4,5,6,7,8,9);     // 42
```

EXERCISE 6

```
1 function sumRecur(sum,num,...nums) {
2     if (nums.length == 0) return sum + num;
3     return sum + sumRecur(num,...nums);
4 }
5
6 sumRecur(3,4,5,6,7,8,9);     // 42
```

PTC
PROPER TAIL CALLS

```
1  "use strict";
2
3  function foo(x) {
4      if (x < 10) return x;
5      return bar(x);
6  }
7
8  function bar(x) {
9      return x / 2;
10 }
11
12 foo(42);           // 21
```

```
1  "use strict";
2
3  function foo(x) {
4      if (x % 2 == 1) {
5          x = Math.round(x / 3);
6      }
7      else {
8          x = x / 2;
9      }
10
11     if (x < 10) return x;
12     return foo(x);
13 }
14
15 foo(42);          // 7
```

```
1 function sumRecur(sum,num,...nums) {
2     if (nums.length == 0) return sum + num;
3     return sum + sumRecur(num,...nums);
4 }
5
6 sumRecur(3,4,5,6,7,8,9);     // 42
```

sum + sum + sum + sum + ..

```
"use strict";

function sumRecur(...nums) {
    return recur(...nums);

    // **************************
    function recur(sum,num,...nums) {
        sum += num;
        if (nums.length == 0) return sum;
        return recur(sum,...nums);
    }
}

sumRecur(3,4,5,6,7,8,9);    // 42
```

```
1  "use strict";
2
3  var sumRecur = (function(){
4      return function(...nums){
5          return recur(...nums);
6      };
7
8      // ************************
9      function recur(sum,num,...nums) {
10         sum += num;
11         if (nums.length == 0) return sum;
12         return recur(sum,...nums);
13     }
14 })();
15
16 sumRecur(3,4,5,6,7,8,9);     // 42
```

```
1 "use strict";
2
3 function sumRecur(sum,num,...nums) {
4     sum += num;
5     if (nums.length == 0) return sum;
6     return sumRecur(sum,...nums);
7 }
8
9 sumRecur(3,4,5,6,7,8,9);     // 42
```

CPS

```
"use strict";

var sumRecur = (function(...nums) {
    return function(...nums){
        return recur(nums,v=>v);
    };

    // ************************
    function recur([sum,...nums],cont) {
        if (nums.length == 0) return cont(sum);
        return recur(nums,function(v){
            return cont(sum + v);
        });
    }
})();

sumRecur(3,4,5,6,7,8,9);    // 42
```

TRAMPOLINES

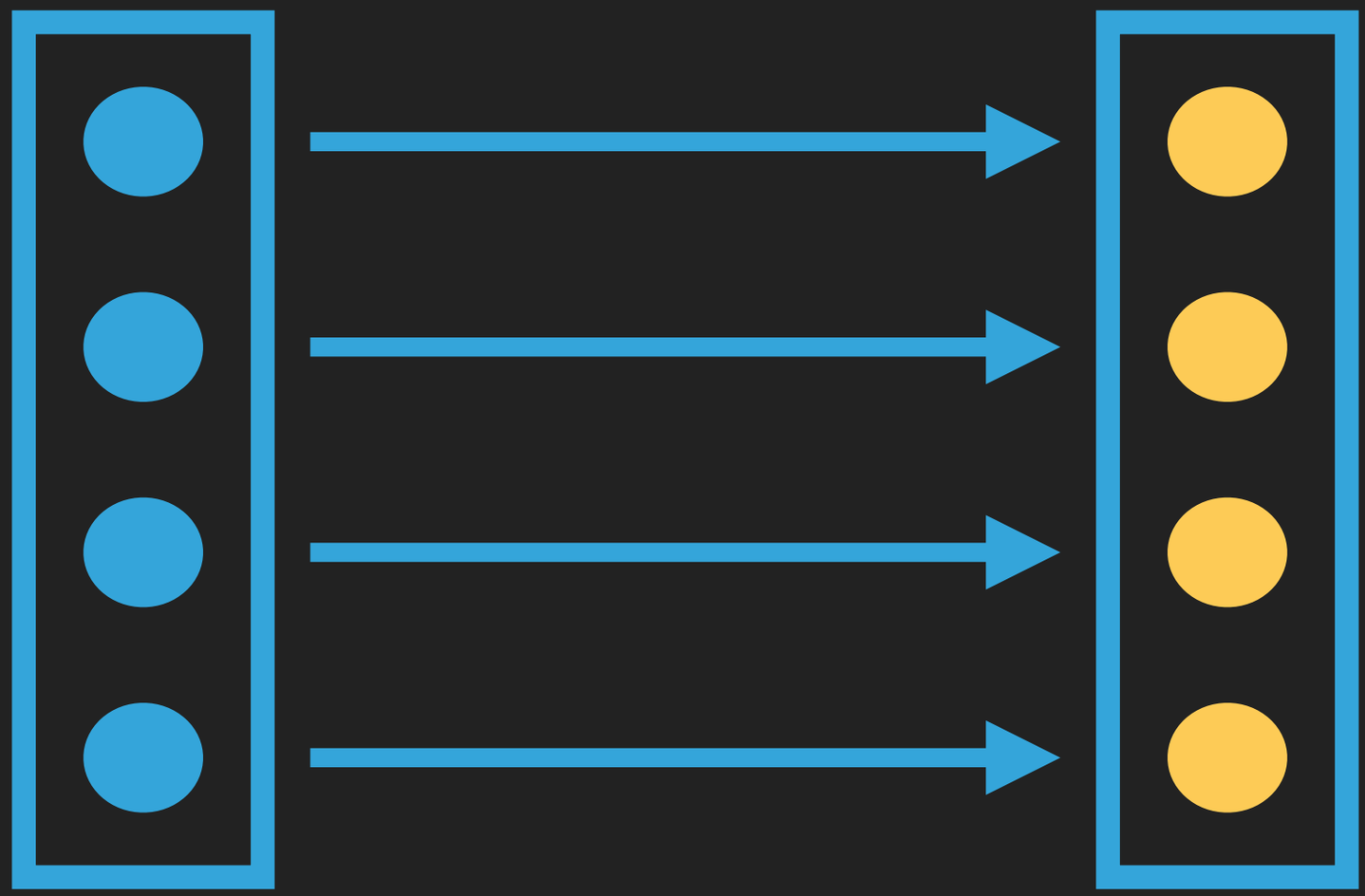
```
var sumTrampolined =
    trampoline(function f(sum,num,...nums) {
        sum += num;
        if (nums.length == 0) return sum;
        return function(){
            return f(sum,...nums);
        };
    });

sumTrampolined(3,4,5,6,7,8,9);  // 42
```

```
function trampoline(fn) {
    return function trampolined(...args) {
        var result = fn(...args);

        while (typeof result == "function") {
            result = result();
        }

        return result;
    };
}
```

If you can do something awesome,
keep doing it repeatedly.

LISTS

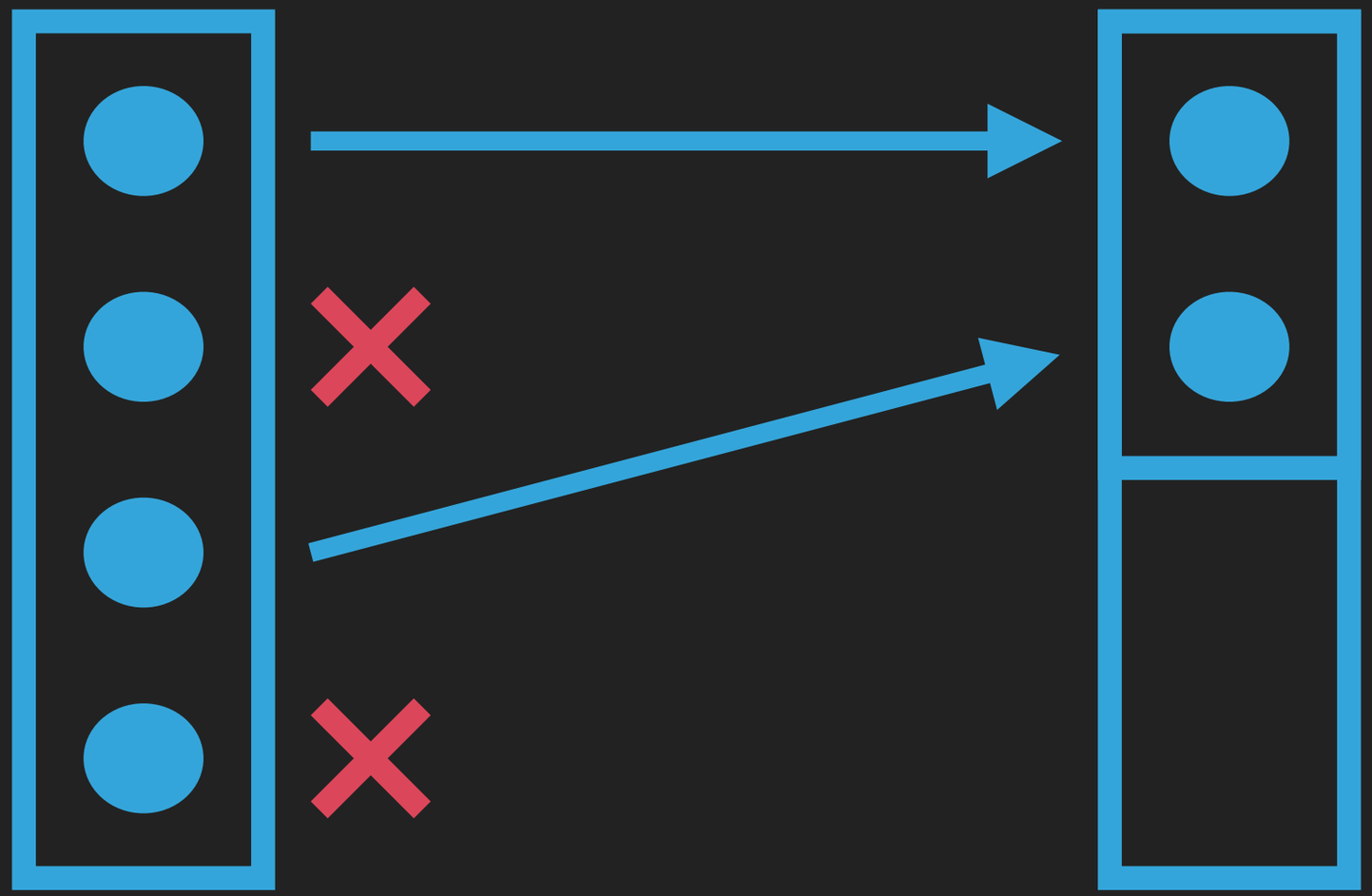
actually, data structures



MAP: TRANSFORMATION

```
1  function doubleIt(v) { return v * 2; }
2
3  function transform(arr,fn) {
4      var list = [];
5      for (var i=0; i<arr.length; i++) {
6          list[i] = fn(arr[i]);
7      }
8      return list;
9  }
10
11
12 transform([1,2,3,4,5],doubleIt);
13 // [2,4,6,8,10]
```

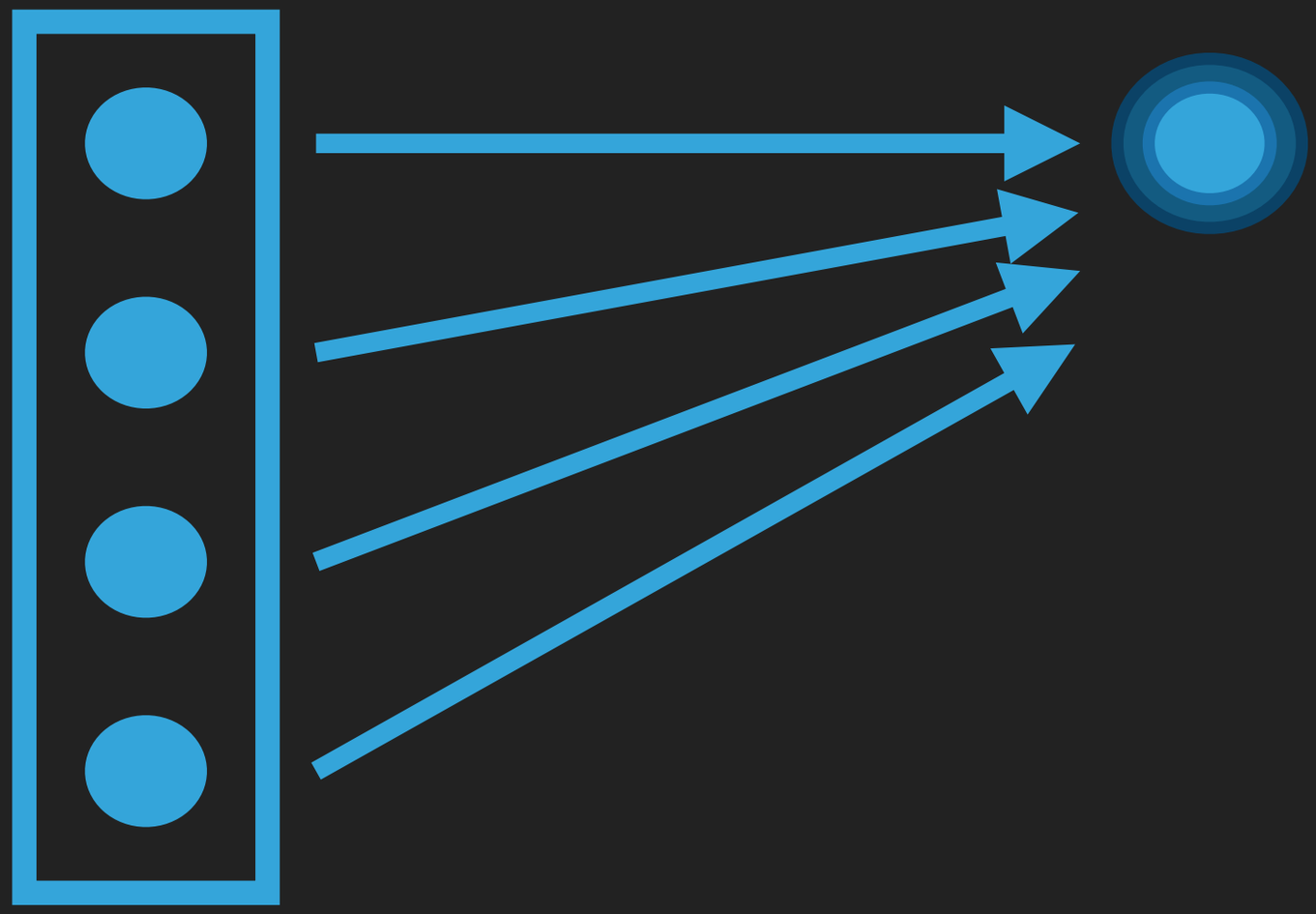
```
1 function doubleIt(val) {
2     return val * 2;
3 }
4
5 [1,2,3,4,5].map(doubleIt);
6 // [2,4,6,8,10]
```



FILTER: EXCLUSION

```
function isOdd(v) { return v % 2 == 1; }

function exclude(arr,fn) {
    var list = [];
    for (var i=0; i<arr.length; i++) {
        if (fn(arr[i])) {
            list.push(arr[i]);
        }
    }
    return list;
}


exclude([1,2,3,4,5],isOdd);
// [1,3,5]
```

```
1 function onlyOdds(val) {
2     return val % 2 == 1;
3 }
4
5 [1,2,3,4,5].filter(onlyOdds);
6 // [1,3,5]
```



REDUCE: COMBINING

```
function mult(x,y) { return x * y; }

function combine(arr,fn,initial) {
    var result = initial;
    for (var i=0; i<arr.length; i++) {
        result = fn(result,arr[i]);
    }
    return result;
}

combine([1,2,3,4,5],mult,1);
// 120
```

```
function acronym(str,word) {
    return str + word.charAt(0);
}

["Functional","Light","JavaScript","Stuff"]
.reduce(acronym,"");
// FLJS
```

EXERCISE 7

FUSION

```
1 function add1(v) { return v + 1; }
2 function mul2(v) { return v * 2; }
3 function div3(v) { return v / 3; }
4
5 var list = [2,5,8,11,14,17,20];
6
7 list
8 .map( add1 )
9 .map( mul2 )
10 .map( div3 );
11 // [2,4,6,8,10,12,14]
```

```
function add1(v) { return v + 1; }
function mul2(v) { return v * 2; }
function div3(v) { return v / 3; }

function composeRight(fn1,fn2) {
    return function(...args){
        return fn1(fn2(...args));
    };
}

var list = [2,5,8,11,14,17,20];

list
.map(
    [div3,mul2,add1].reduce(composeRight)
);
// [2,4,6,8,10,12,14]
```

TRANSDUCING

```
function add1(v) { return v + 1; }
function isOdd(v) { return v % 2 == 1; }
function sum(total,v) { return total + v; }

var list = [2,5,8,11,14,17,20];

list
.map( add1 )
.filter( isOdd )
.reduce( sum );
// 48
```

```
function mapWithReduce(arr,mappingFn) {
    return arr.reduce(function reducer(list,v){
        list.push( mappingFn(v) );
        return list;
    }, [] );
}

function filterWithReduce(arr,predicateFn) {
    return arr.reduce(function reducer(list,v){
        if (predicateFn(v)) list.push(v);
        return list;
    }, [] );
}

var list = [2,5,8,11,14,17,20];

list = mapWithReduce( list, add1 );
list = filterWithReduce( list, isOdd );
list.reduce( sum );
// 48
```

```
function mapReducer(mappingFn) {
    return function reducer(list,v){
        list.push( mappingFn(v) );
        return list;
    };
}

function filterReducer(predicateFn) {
    return function reducer(list,v){
        if (predicateFn(v)) list.push(v);
        return list;
    };
}

var list = [2,5,8,11,14,17,20];

list
.reduce( mapReducer(add1), [] )
.reduce( filterReducer(isOdd), [] )
.reduce( sum );
// 48
```

```
function listCombination(list,v) {
    list.push(v);
    return list;
}

function mapReducer(mappingFn) {
    return function reducer(list,v){
        return listCombination( list, mappingFn(v) );
    };
}

function filterReducer(predicateFn) {
    return function reducer(list,v){
        if (predicateFn(v)) return listCombination( list, v );
        return list;
    };
}

var list = [2,5,8,11,14,17,20];

list
.reduce( mapReducer(add1), [] )
.reduce( filterReducer(isOdd), [] )
.reduce( sum );
// 48
```

```
1  function listCombination(list,v) {
2      list.push(v);
3      return list;
4  }
5
6  var mapReducer = curry(function mapReducer(mappingFn,combineFn){
7      return function reducer(list,v){
8          return combineFn( list, mappingFn(v) );
9      };
10 });
11
12 var filterReducer = curry(function filterReducer(predicateFn,combineFn){
13     return function reducer(list,v){
14         if (predicateFn(v)) return combineFn( list, v );
15         return list;
16     };
17 });
18
19 var list = [2,5,8,11,14,17,20];
20
21 list
22 .reduce( mapReducer(add1)(listCombination), [] )
23 .reduce( filterReducer(isOdd)(listCombination), [] )
24 .reduce( sum );
25 // 48
```

```
function listCombination(list,v) {
    list.push(v);
    return list;
}

var mapReducer = curry(function mapReducer(mappingFn,combineFn){
    return function reducer(list,v){
        return combineFn( list, mappingFn(v) );
    };
});

var filterReducer = curry(function filterReducer(predicateFn,combineFn){
    return function reducer(list,v){
        if (predicateFn(v)) return combineFn( list, v );
        return list;
    };
});

var transducer = compose( mapReducer(add1), filterReducer(isOdd) );

var list = [2,5,8,11,14,17,20];

list
.reduce( transducer(listCombination), [] )
.reduce( sum );
// 48
```

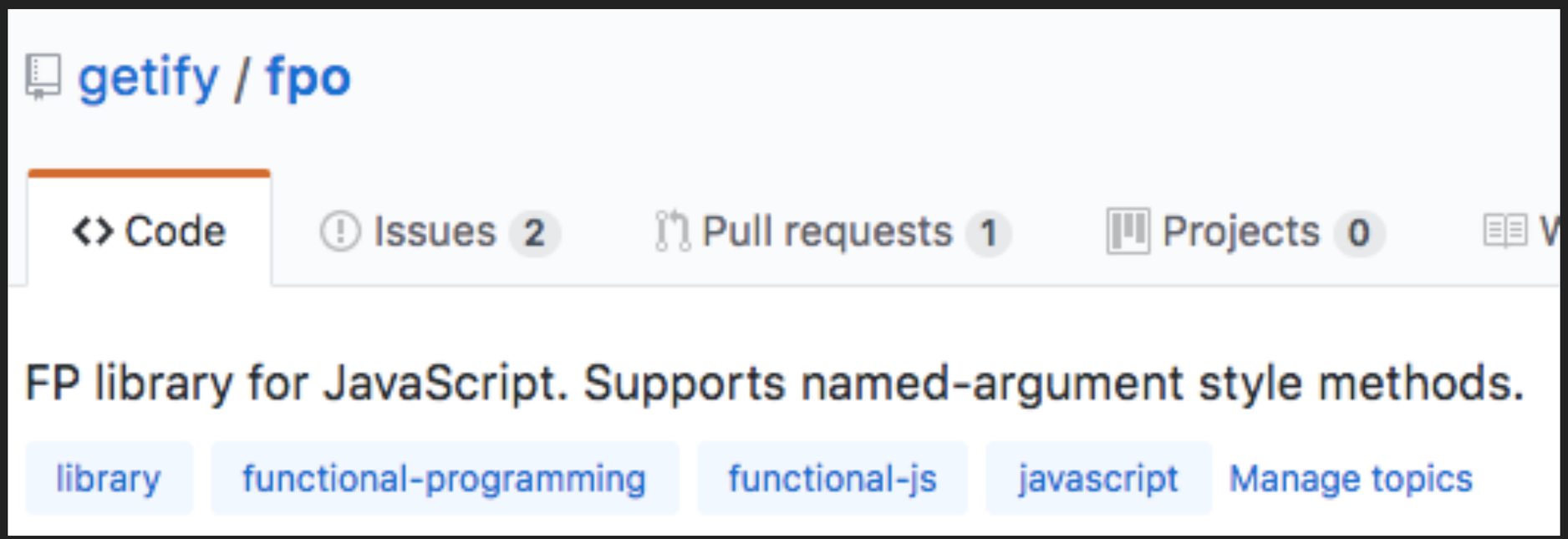
```
var mapReducer = curry(function mapReducer(mappingFn,combineFn){
    return function reducer(list,v){
        return combineFn( list, mappingFn(v) );
    };
});

var filterReducer = curry(function filterReducer(predicateFn,combineFn){
    return function reducer(list,v){
        if (predicateFn(v)) return combineFn( list, v );
        return list;
    };
});

var transducer = compose( mapReducer(add1), filterReducer(isOdd) );

var list = [2,5,8,11,14,17,20];

list
.reduce( transducer(sum), 0 );
// 48
```

```
function transduce(transducer,combineFn,initialValue,list) {
    var reducer = transducer(combineFn);
    return list.reduce(reducer,initialValue);
}

var transducer = compose( mapReducer(add1), filterReducer(isOdd) );

transduce( transducer, sum, 0, [2,5,8,11,14,17,20] );
// 48
```

DATA STRUCTURE OPERATIONS

```
function mapObj(mapperFn,o) {
    var newObj = {};
    var keys = Object.keys(o);
    for (let key of keys) {
        newObj[key] = mapperFn( o[key] );
    }
    return newObj;
}

var obj = {
    a: "Hello",
    b: "World"
};

mapObj(function upper(val){
    return val.toUpperCase();
},obj);
// {a: "HELLO", b: "WORLD" }
```

EXERCISE 8



github.com/getify/fpo

```
// the classic/traditional method style
// (on the `FPO.std.*` namespace)
FPO.std.reduce(
    (acc,v) => acc + v,
    undefined,
    [3,7,9]
); // 19

// FPO named-argument method style
FPO.reduce({
    arr: [3,7,9],
    fn: ({acc,v}) => acc + v
}); // 19
```

```
var f = curry(
    flip(partialRight(reduce,[[3,7,9]]))
)(0);

f((acc,v) => acc + v);  // 19
f((acc,v) => acc * v);  // 189
```

```
var f = FPO.reduce({ arr: [3,7,9] });

// later:
f({ fn: ({acc,v}) => acc + v });  // 19
f({ fn: ({acc,v}) => acc * v });  // 189
```

ASYNC

```
1 var a = [1,2,3];
2
3 var b = a.map(function(v){
4     return v * 2;
5 });
6
7 b;                  // [2,4,6]
```

SYNCHRONOUS

FP OVER TIME

EAGER
VS
LAZY

```
var a = [];

var b = mapLazy(a,function(v){
    return v * 2;
});

a.push(1);

a[0];          // 1
b[0];          // 2

a.push(2);

a[1];          // 2
b[1];          // 4
```

```
var a = new LazyArray();

setInterval(function everySecond(){
    a.push(Math.random());
},1000);

// ************************************

var b = a.map(function(v){
    return v * 2;
});

b.forEach(function onValue(v){
    console.log(v);
});
```

LAZYARRAY

OBSERVABLE

```
var a = new Rx.Subject();

setInterval(function everySecond(){
    a.next(Math.random());
},1000);

// ********************************

var b = a.map(function(v){
    return v * 2;
});

b.subscribe(function onValue(v){
    console.log(v);
});
```

EXERCISE 9

RECAP:

- Functions (~~side effects~~, point-free)
- Composition
- Immutability
- Closure
- List & Data Structure operations
- Async (observables)

THANKS!!!!

KYLE SIMPSON GETIFY@GMAIL.COM

FUNCTIONAL-LIGHT JS