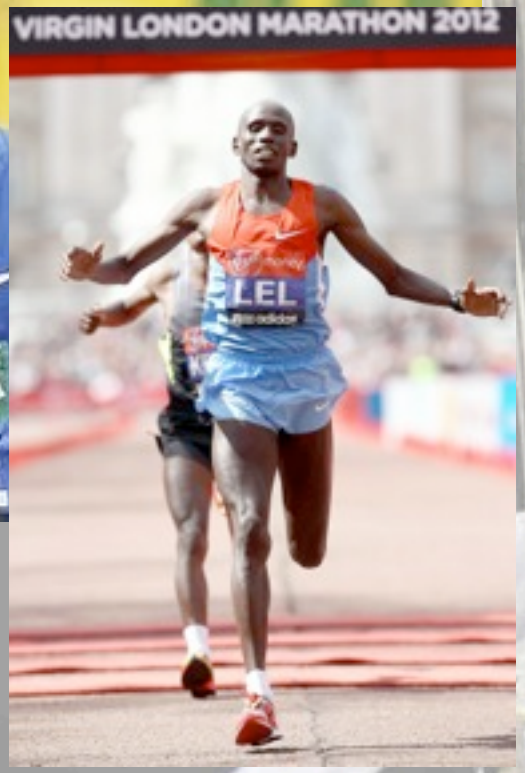
syncing async

kyle simpson
@getify
http://getify.me



parallel

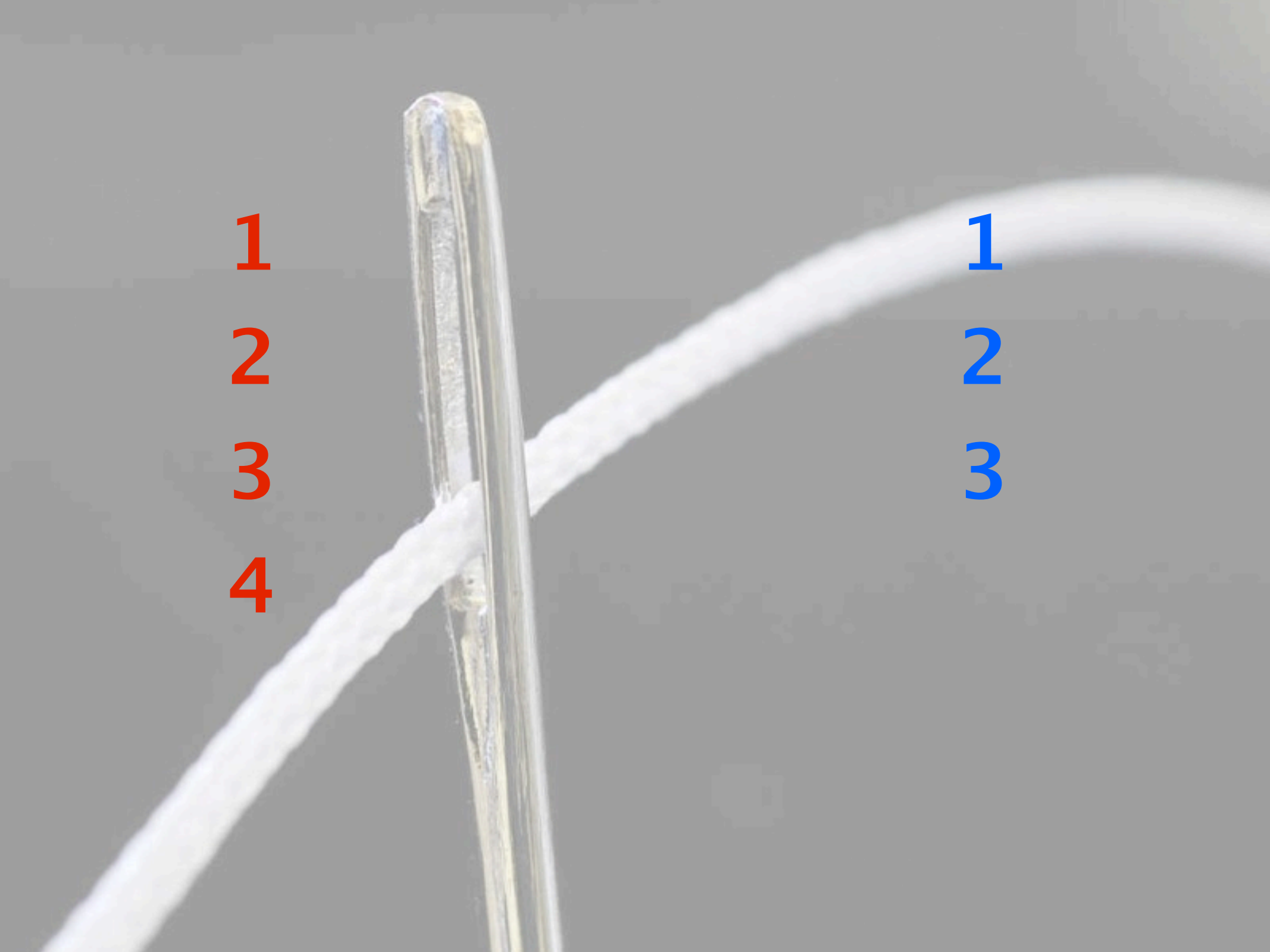
vs.

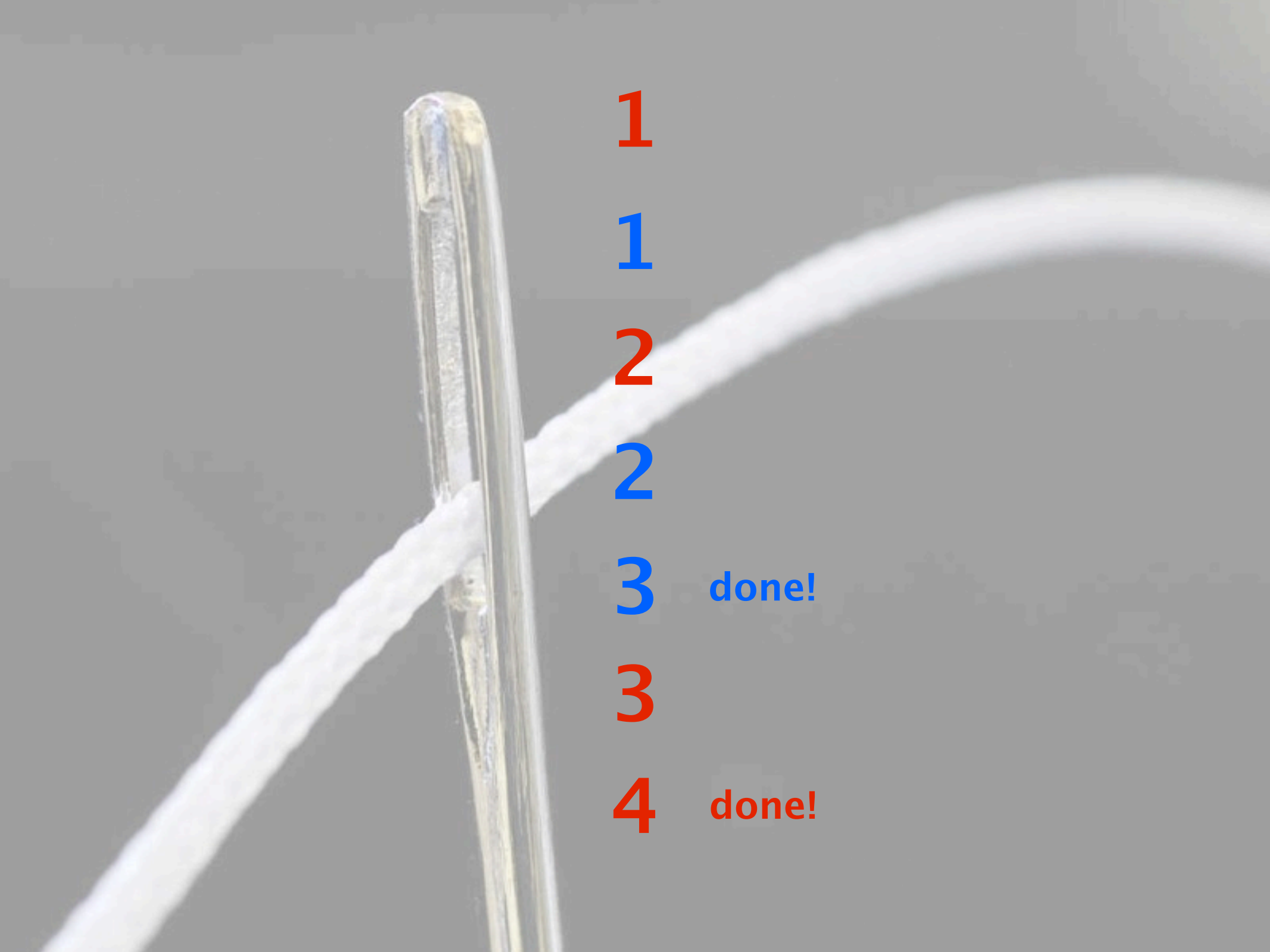
async

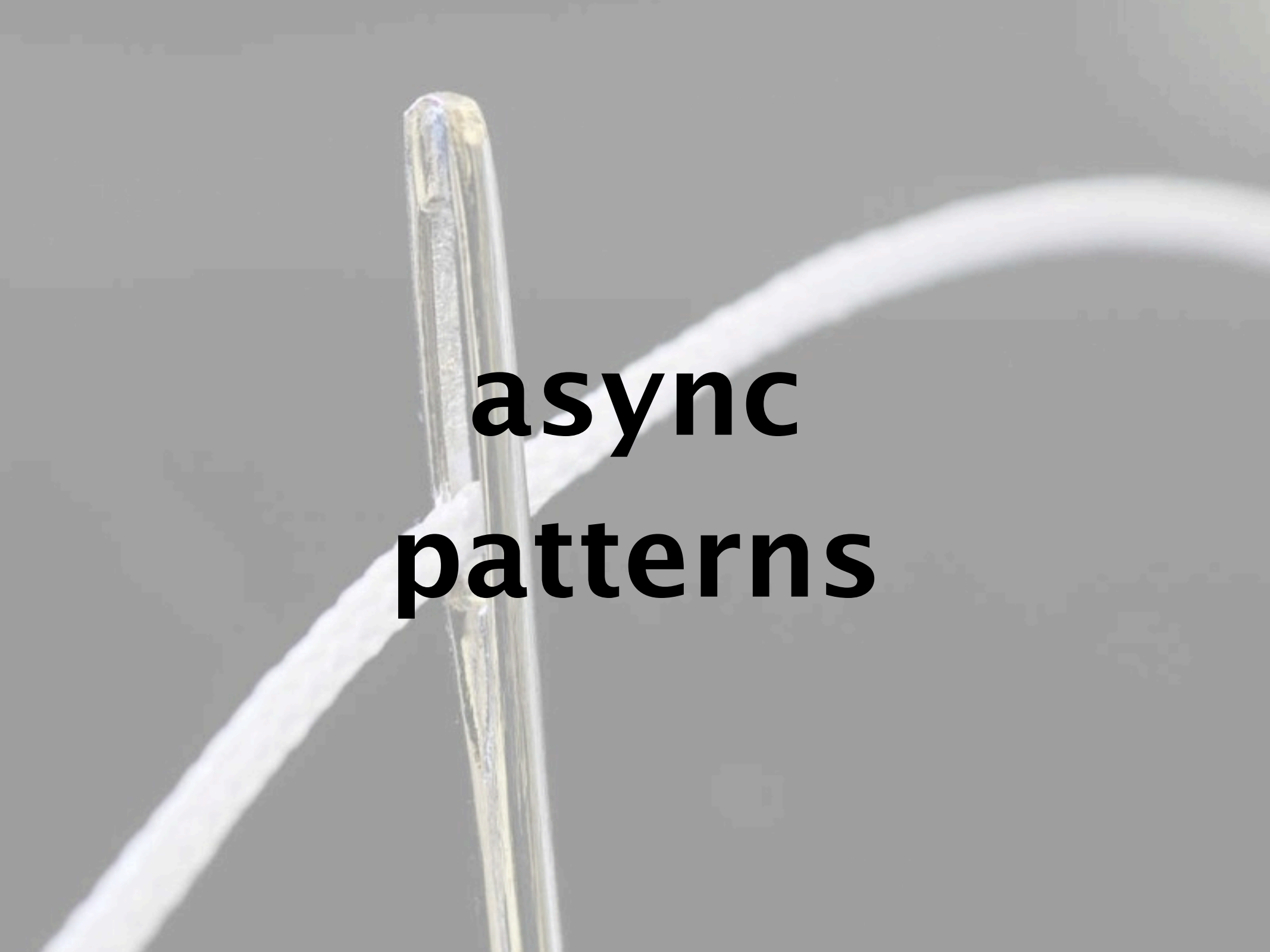
threads

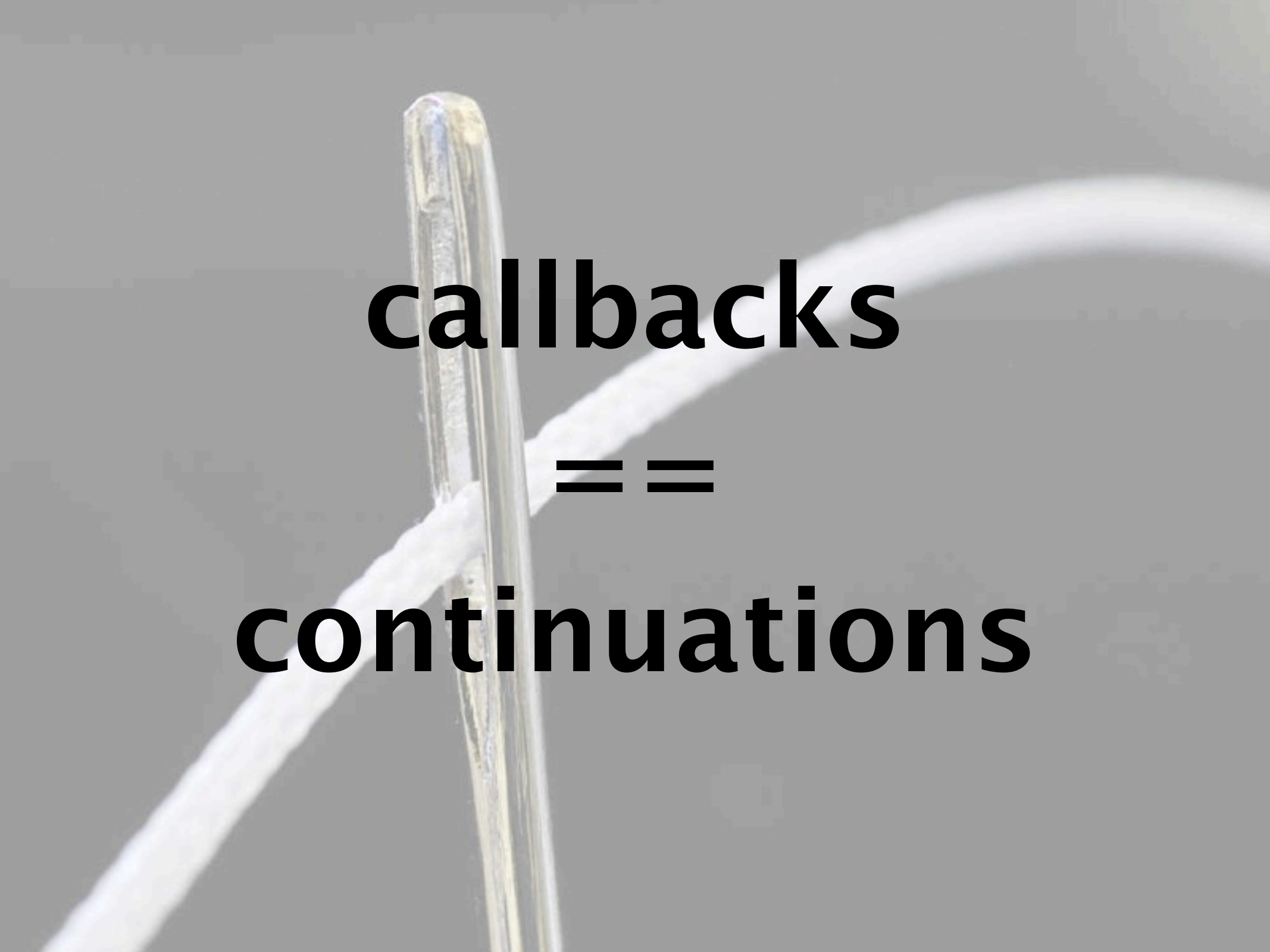

AROMA

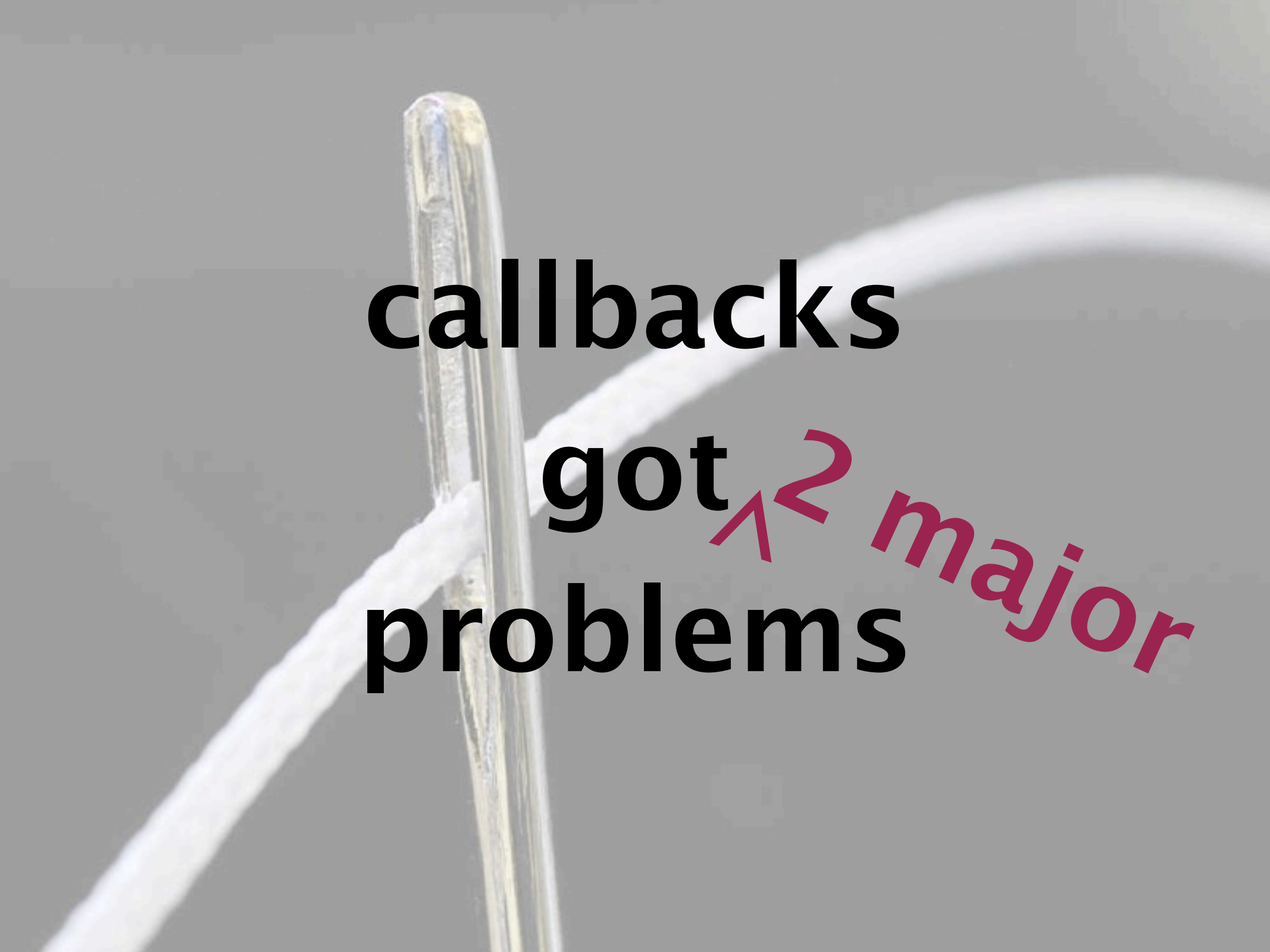

event-loop
concurrency


1
2
3
4
1
2
3


1
1
2
2
3 done!
3
4 done!


async
patterns


callbacks
==
continuations

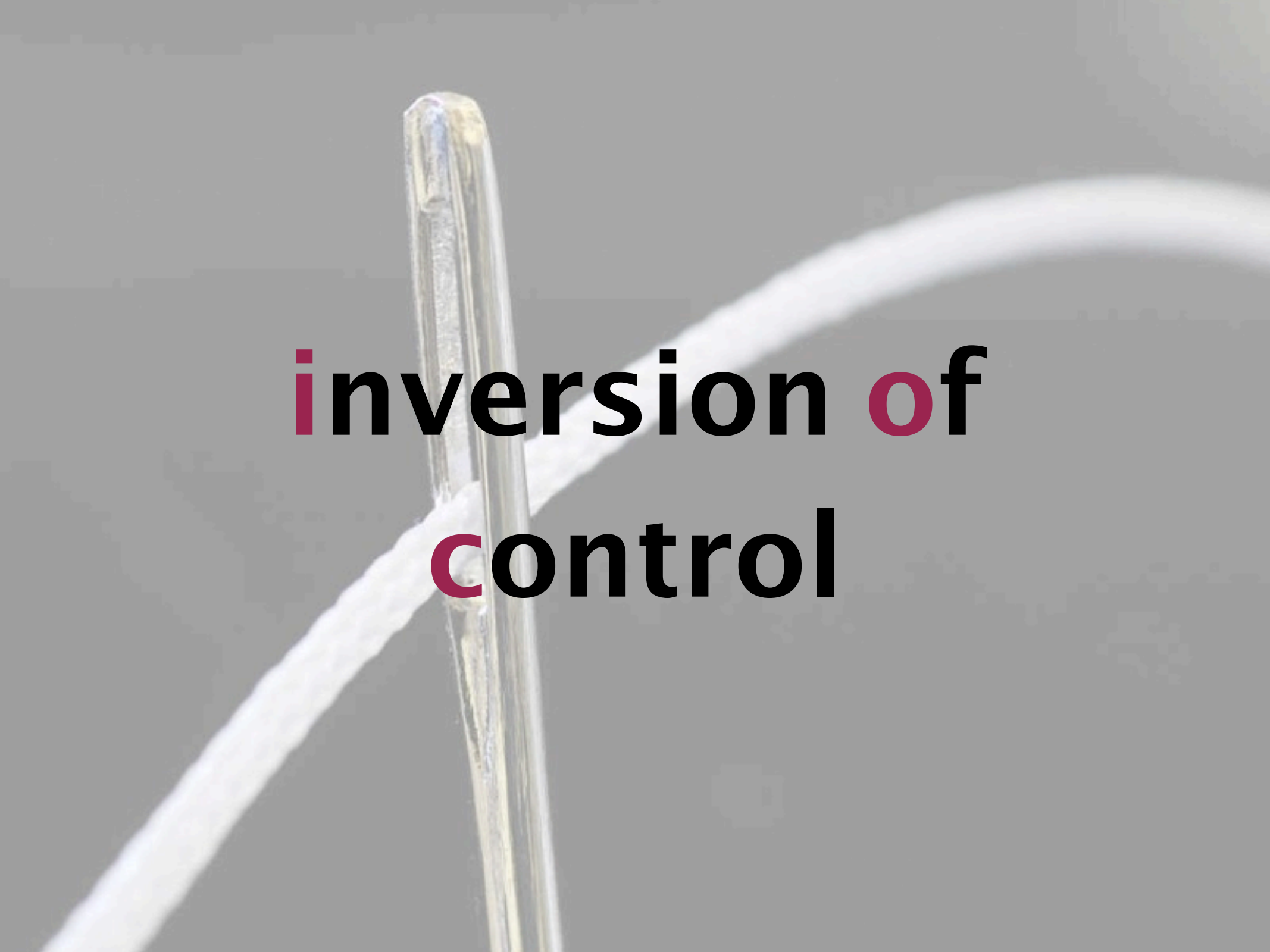

callbacks
got
problems
^2 major

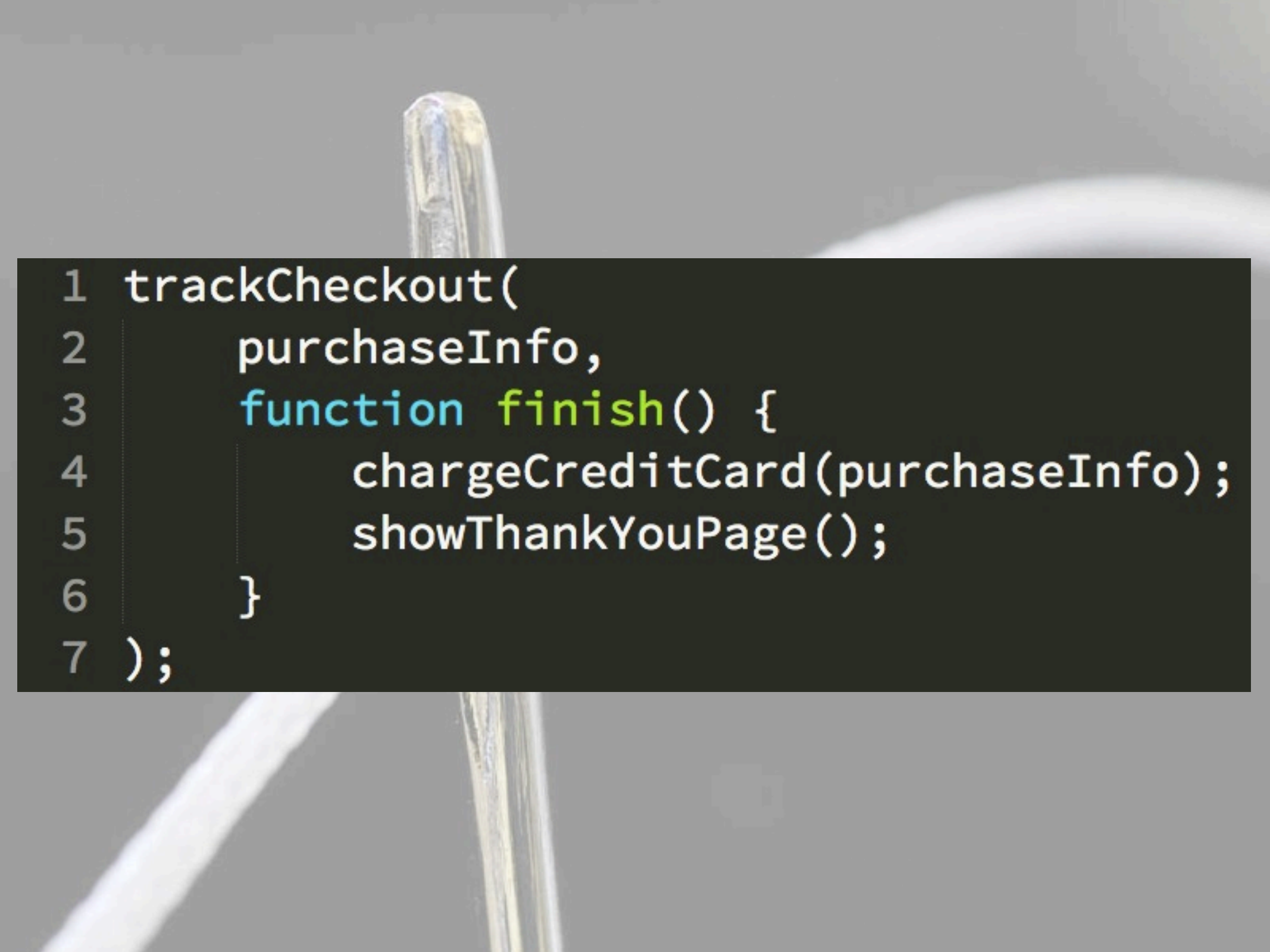
```
setTimeout(function(){
    console.log("one");
    setTimeout(function(){
        console.log("two");
        setTimeout(function(){
            console.log("three");
        },1000);
    },1000);
},1000);
```

“callback hell”

```
 1 function one(cb) {
 2     console.log("one");
 3     setTimeout(cb,1000);
 4 }
 5 function two(cb) {
 6     console.log("two");
 7     setTimeout(cb,1000);
 8 }
 9 function three() {
10     console.log("three");
11 }
12
13 one(function(){
14     two(three);
15 });
```

still “callback hell”

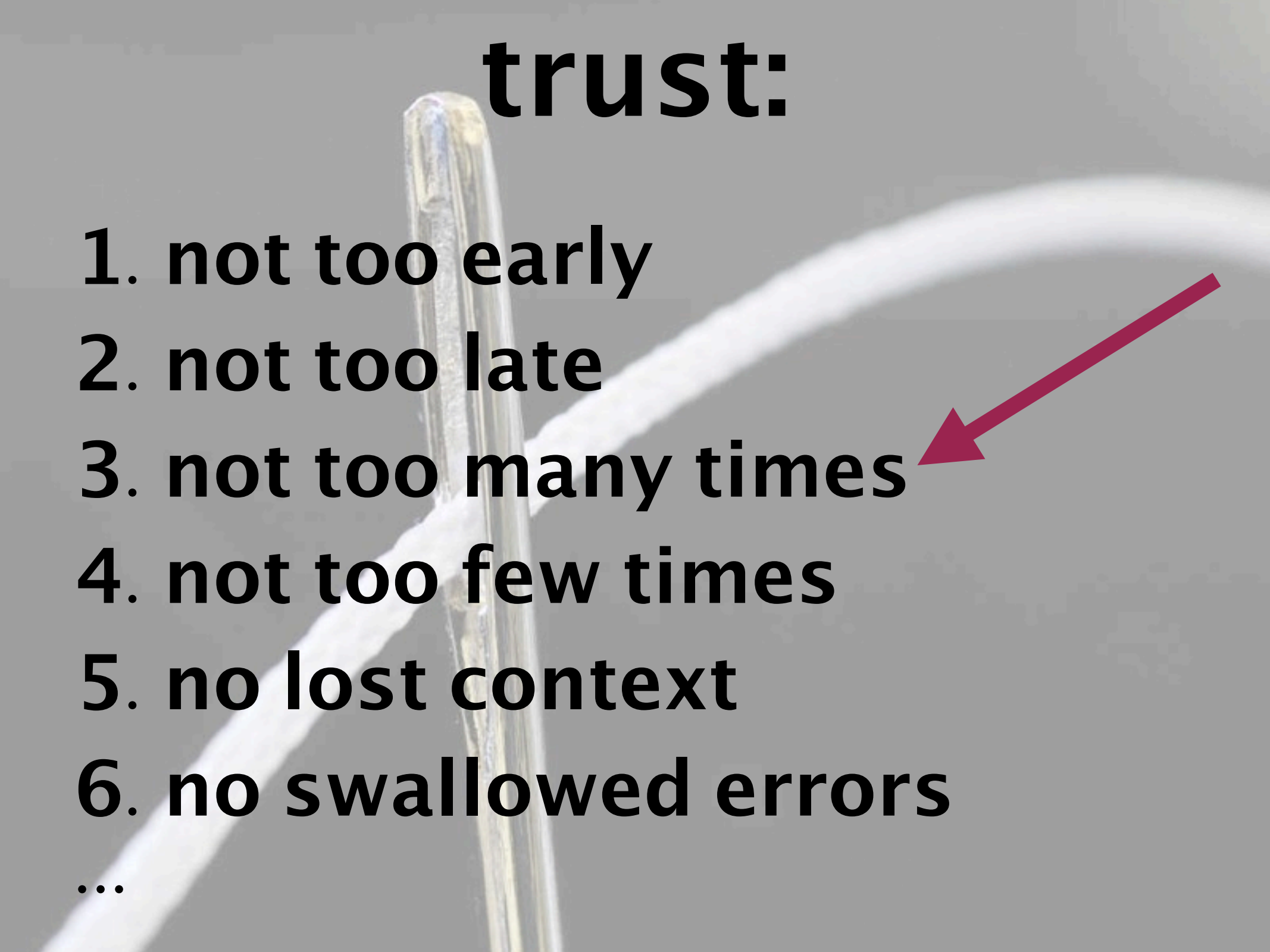

inversion of
control

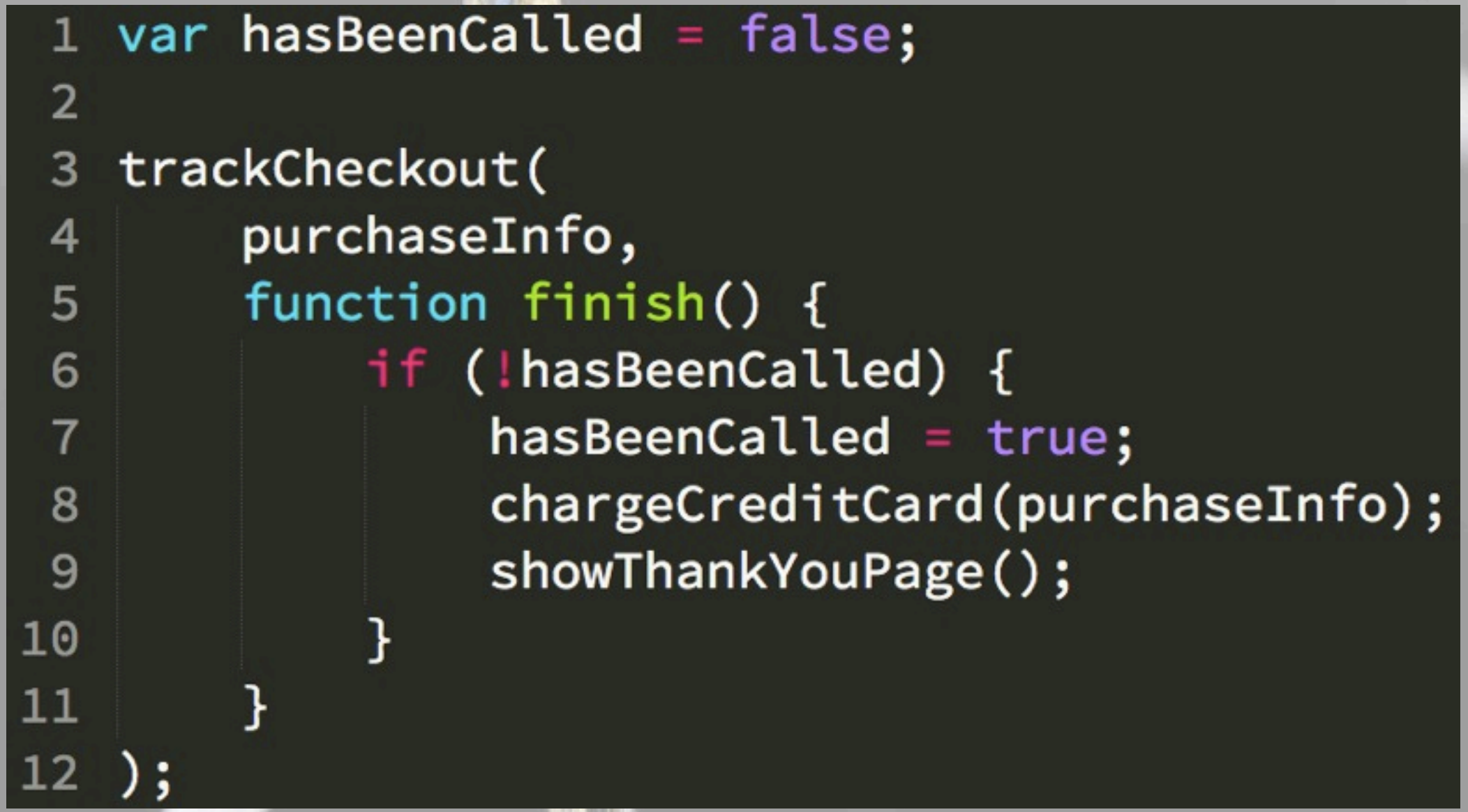

trackCheckout(
purchaseInfo,
function finish() {
chargeCreditCard(purchaseInfo);
showThankYouPage();
}
);

trust:

1. not too early
2. not too late
3. not too many times
4. not too few times
5. no lost context
6. no swallowed errors

…



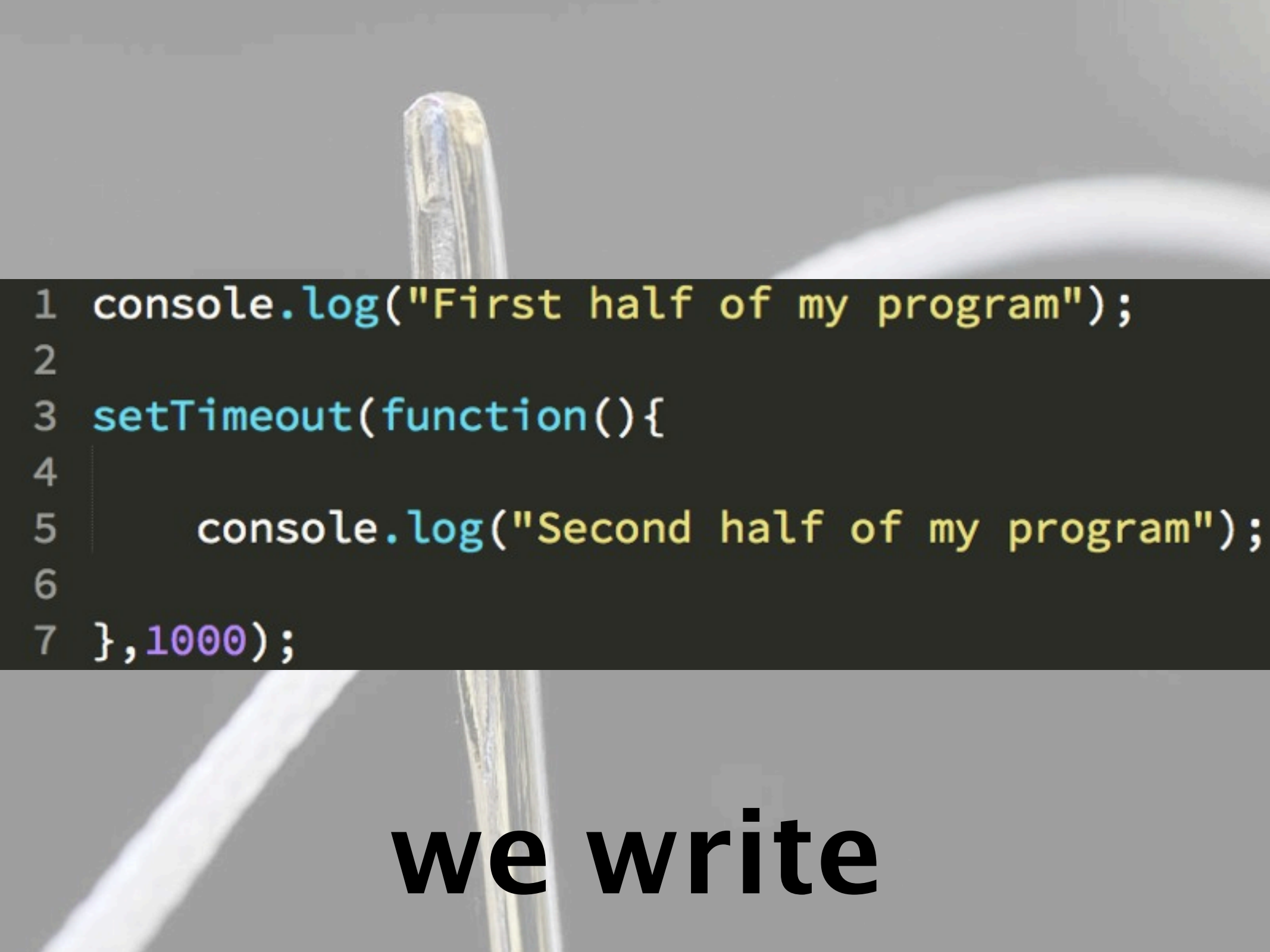


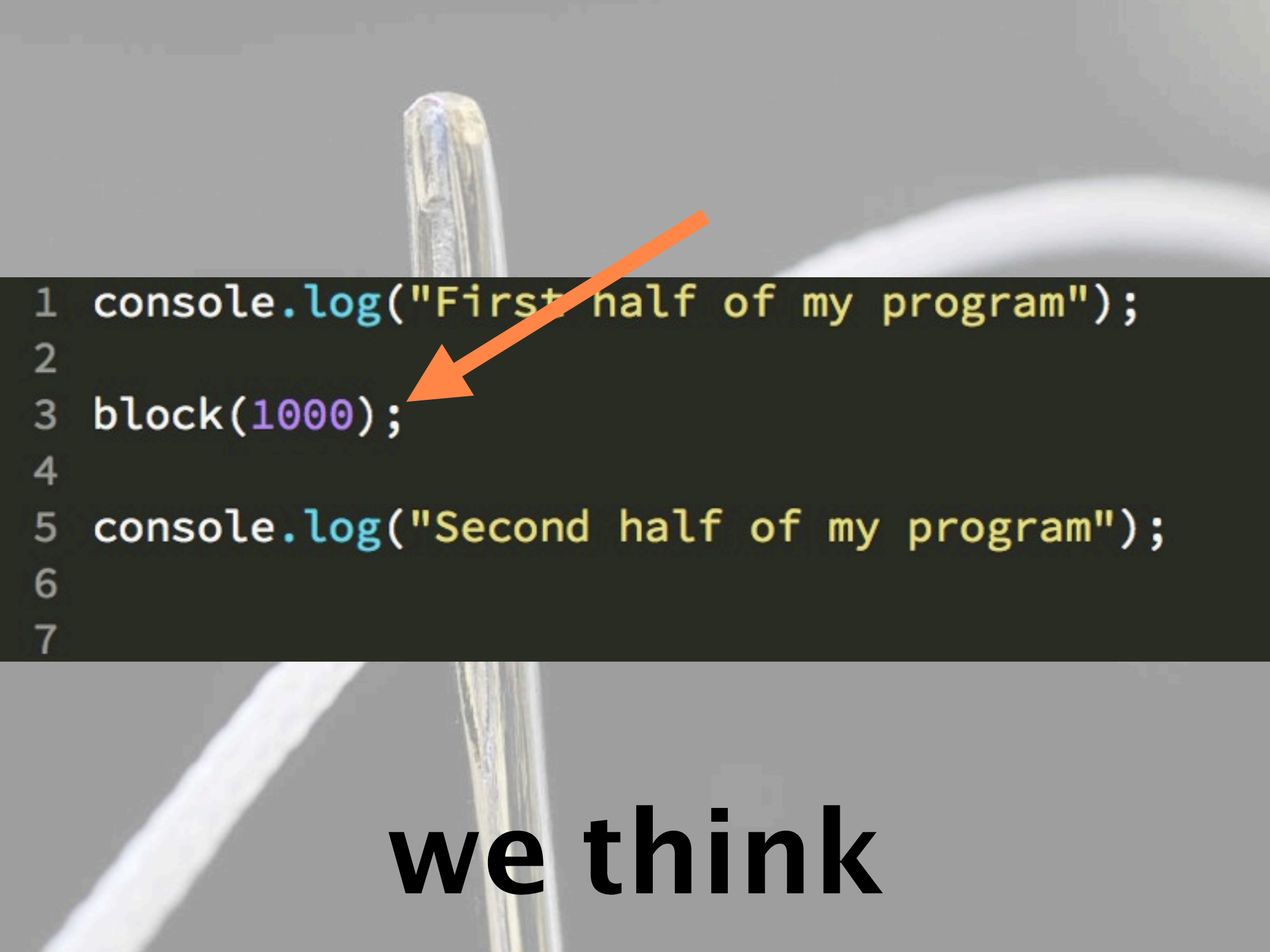
```
var hasBeenCalled = false;

trackCheckout(
    purchaseInfo,
    function finish() {
        if (!hasBeenCalled) {
            hasBeenCalled = true;
            chargeCreditCard(purchaseInfo);
            showThankYouPage();
        }
    }
);
```

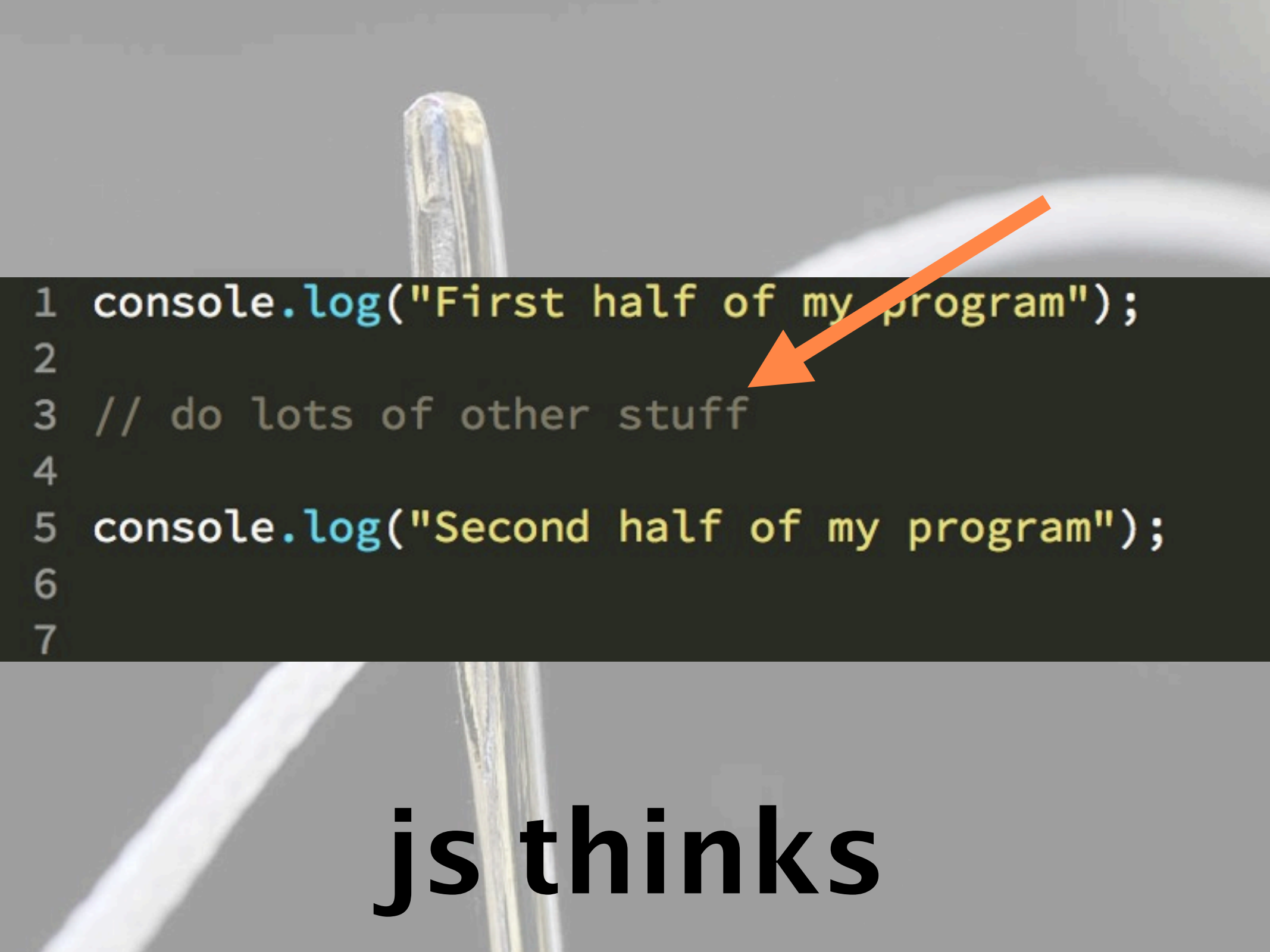

nested callbacks
are not
reasonable

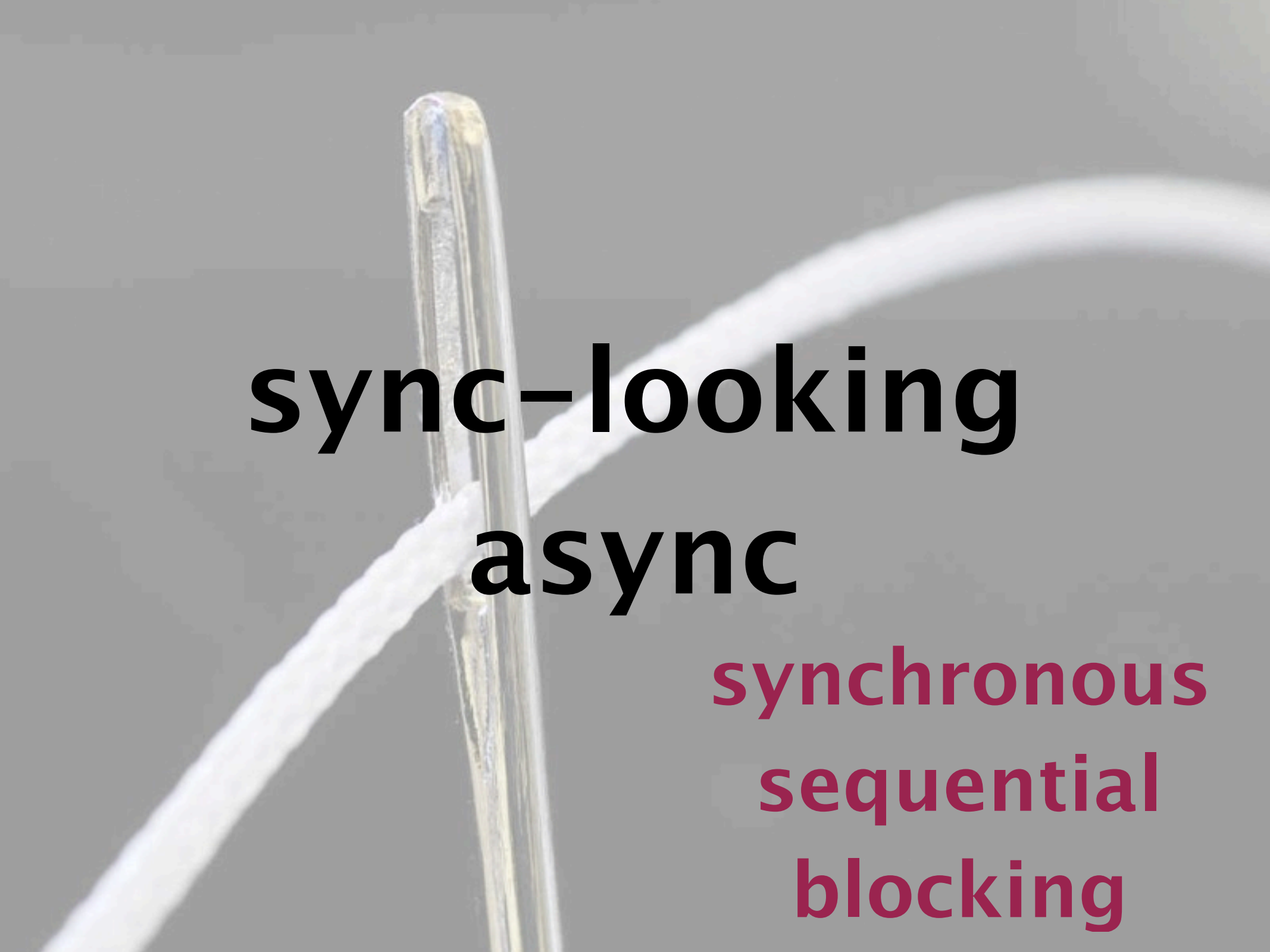
```
start task1:
    do some stuff
    pause

start task2:
    do some other stuff
    pause

resume task1:
    do more stuff
    pause

resume task2:
    finish stuff

resume task1:
    finish stuff
```

```
start task1:
    do some stuff
    pause

    resume task1:
        do more stuff
        pause

        resume task1:
            finish stuff


start task2:
    do some other stuff
    pause

    resume task2:
        finish stuff
```


1 console.log("First half of my program");
2
3 setTimeout(function(){
4
5 console.log("Second half of my program");
6
7 },1000);
we write


1 console.log("First half of my program");
2
3 block(1000);
4
5 console.log("Second half of my program");
6
7
we think


1 console.log("First half of my program");
2
3 // do lots of other stuff
4
5 console.log("Second half of my program");
6
7
js thinks

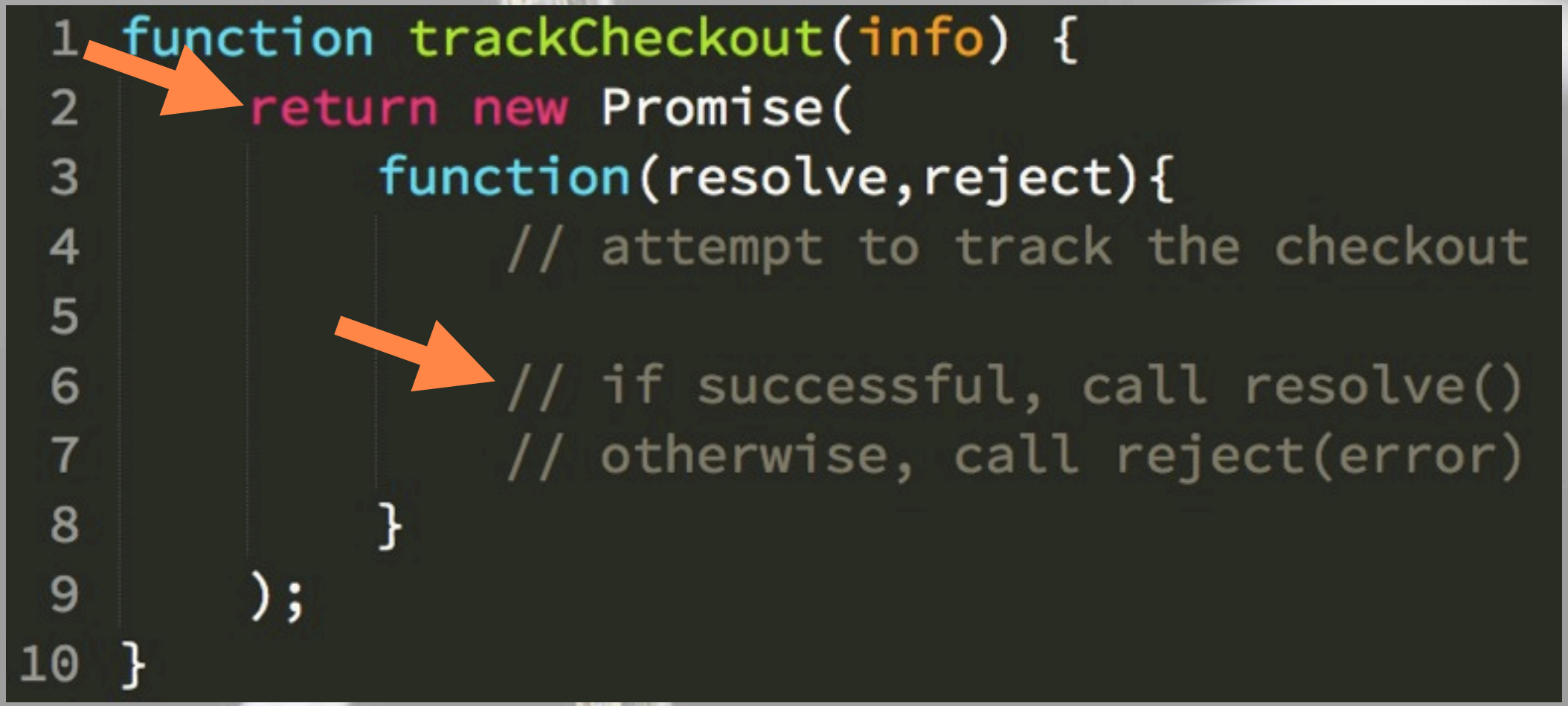

sync-looking
async
synchronous
sequential
blocking


hell is
callbacks
-Sartre

promise: future value

```
1  function finish(){
2      chargeCreditCard(purchaseInfo);
3      showThankYouPage();
4  }
5
6  function error(err){
7      logStatsError(err);
8      finish();
9  }
10
11 var listener = trackCheckout(purchaseInfo);
12
13 listener.on("completion",finish);
14 listener.on("error",error);
```

promise: “completion” event


function trackCheckout(info) {
return new Promise(
function(resolve,reject){
// attempt to track the checkout
// if successful, call resolve()
// otherwise, call reject(error)
}
);
}

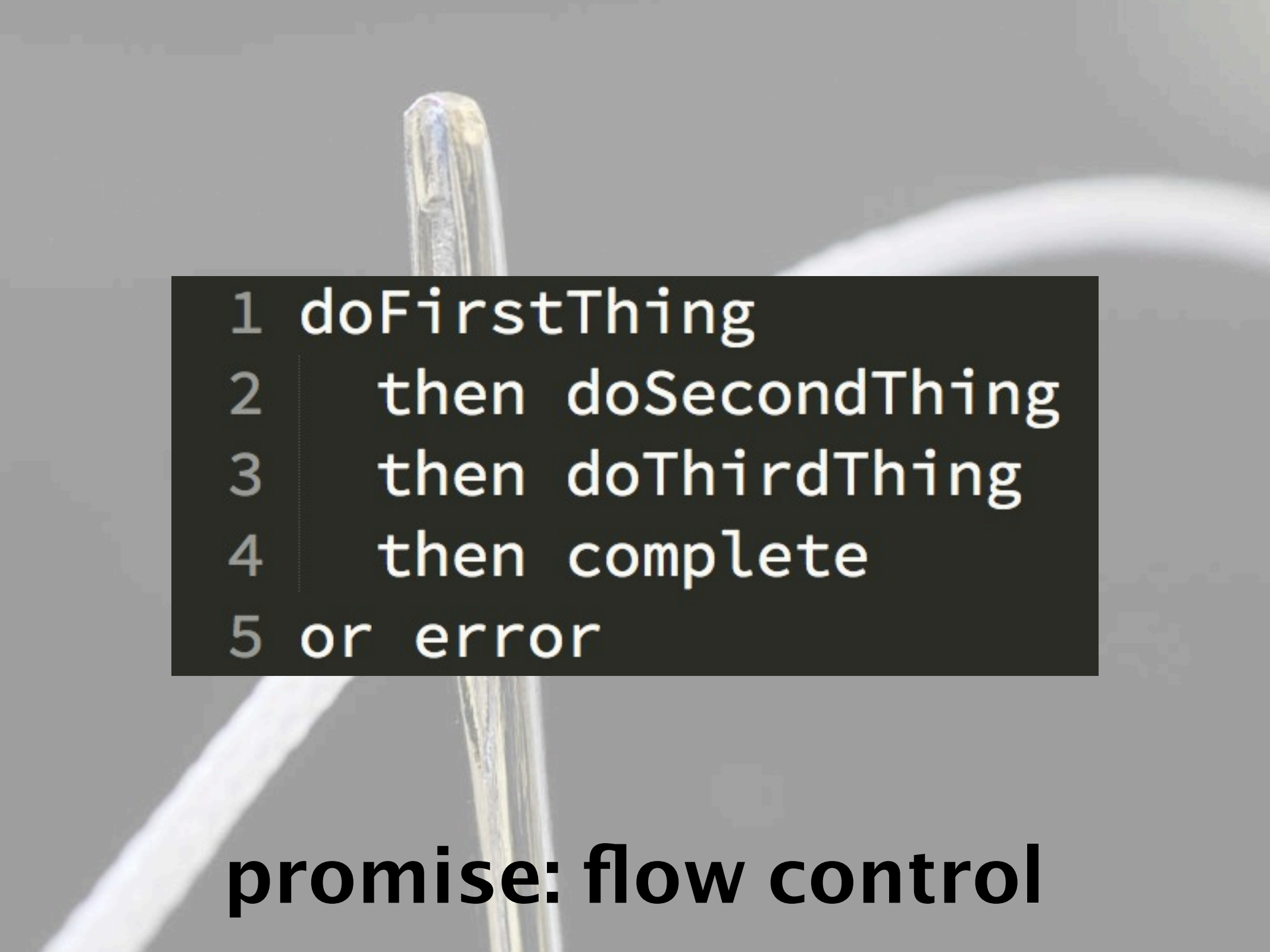
```
1  function finish(){
2      chargeCreditCard(purchaseInfo);
3      showThankYouPage();
4  }
5
6  function error(err){
7      logStatsError(err);
8      finish();
9  }
10
11 var promise = trackCheckout(purchaseInfo);
12
13 promise.then(
14     finish,
15     error
16 );
```

promises:

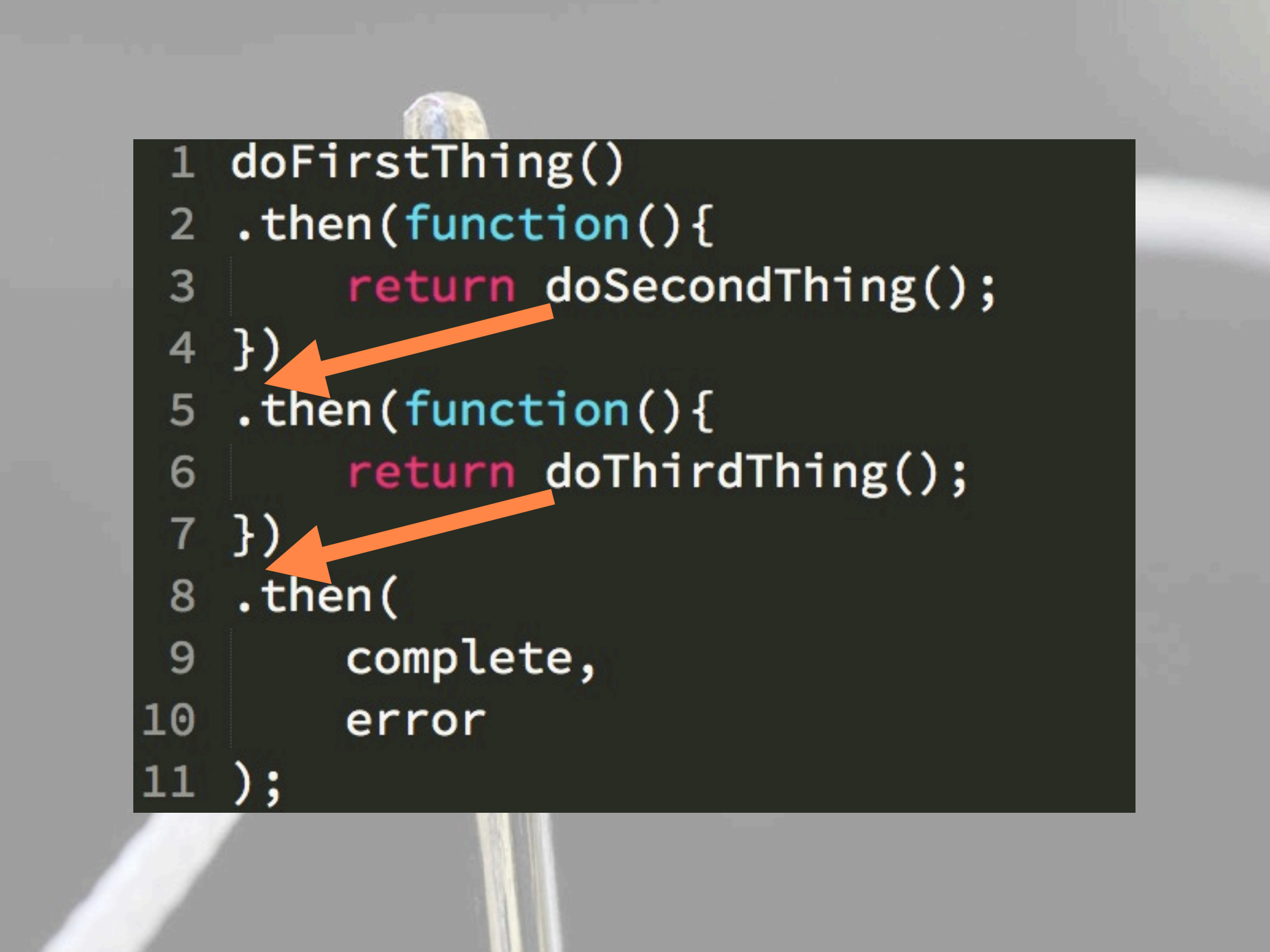
1. only resolved once
2. either success OR error
3. messages passed/kept
4. exceptions become errors
5. immutable once resolved

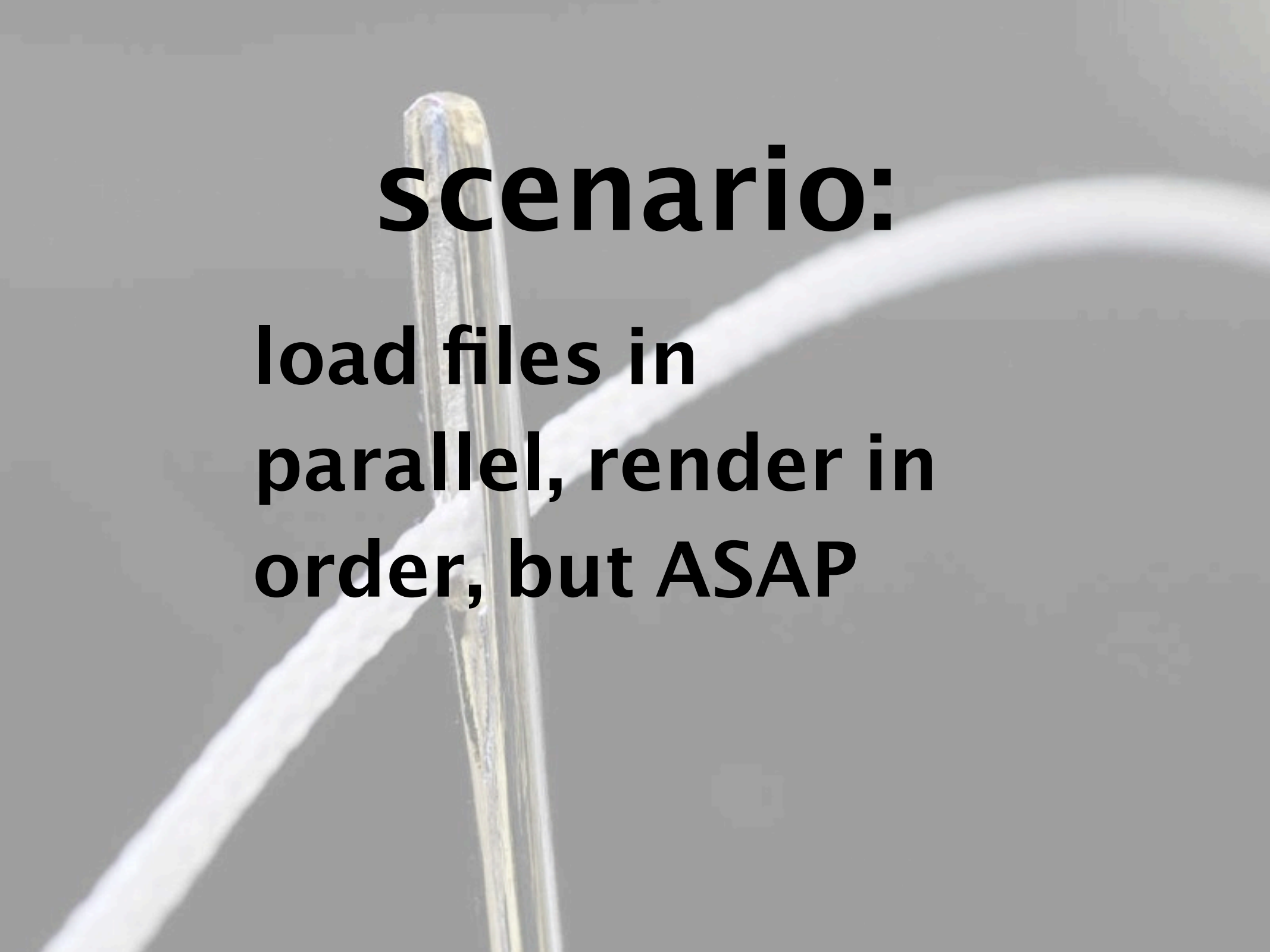

uninversion of
control


steps



promise: flow control


doFirstThing()
.then(function(){
return doSecondThing();
})
.then(function(){
return doThirdThing();
})
.then(
complete,
error
);



scenario:

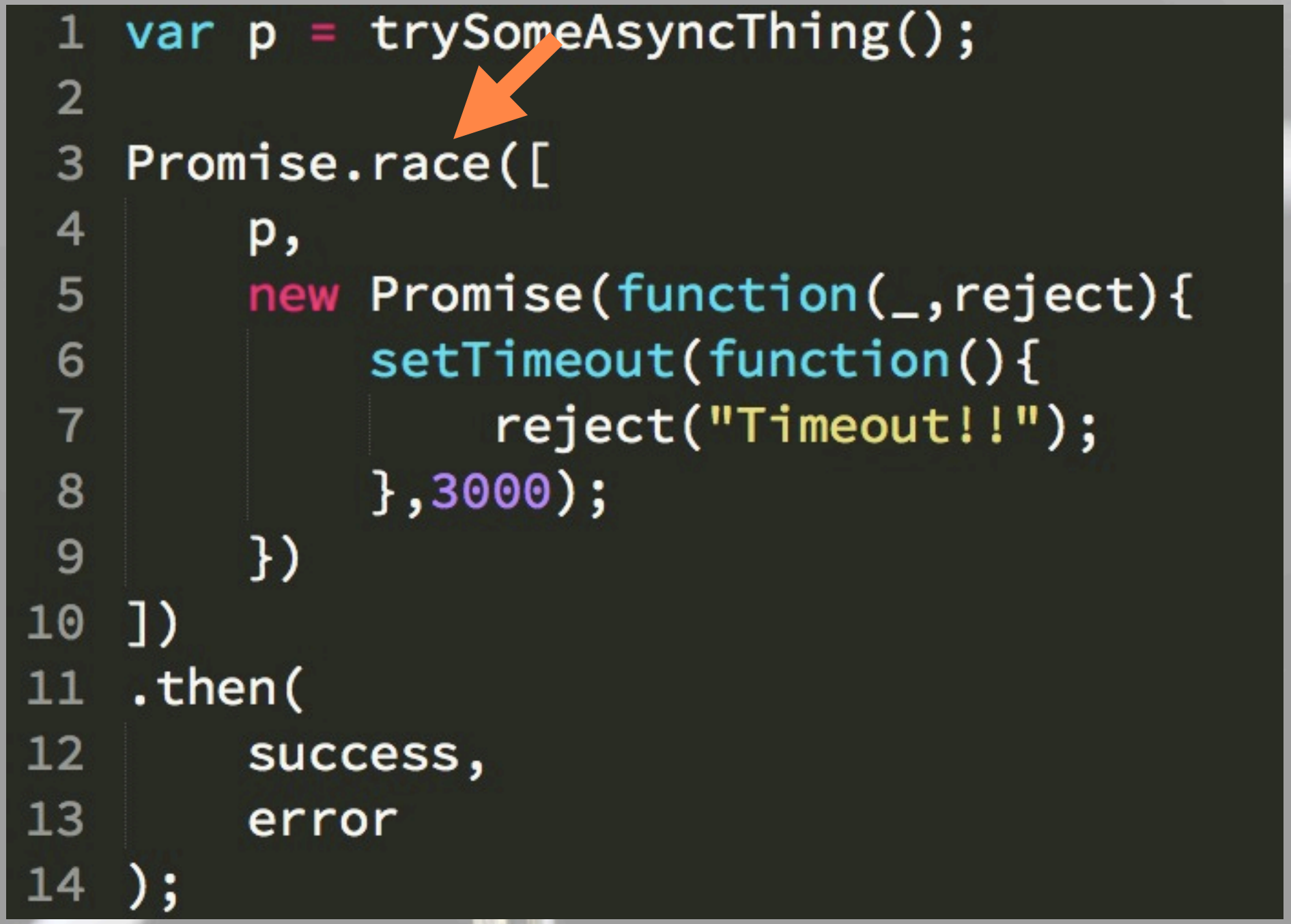
load files in parallel, render in order, but ASAP

```
// Request all files at once in "parallel"
var p1 = getFile("file1");
var p2 = getFile("file2");
var p3 = getFile("file3");

// Render as each one finishes, but only once
// previous rendering is done.
p1
.then(output)
.then(function(){
    return p2;
})
.then(output)
.then(function(){
    return p3;
})
.then(output)
.then(function(){
    output("Complete!");
});
```


cool story, bro.
but...

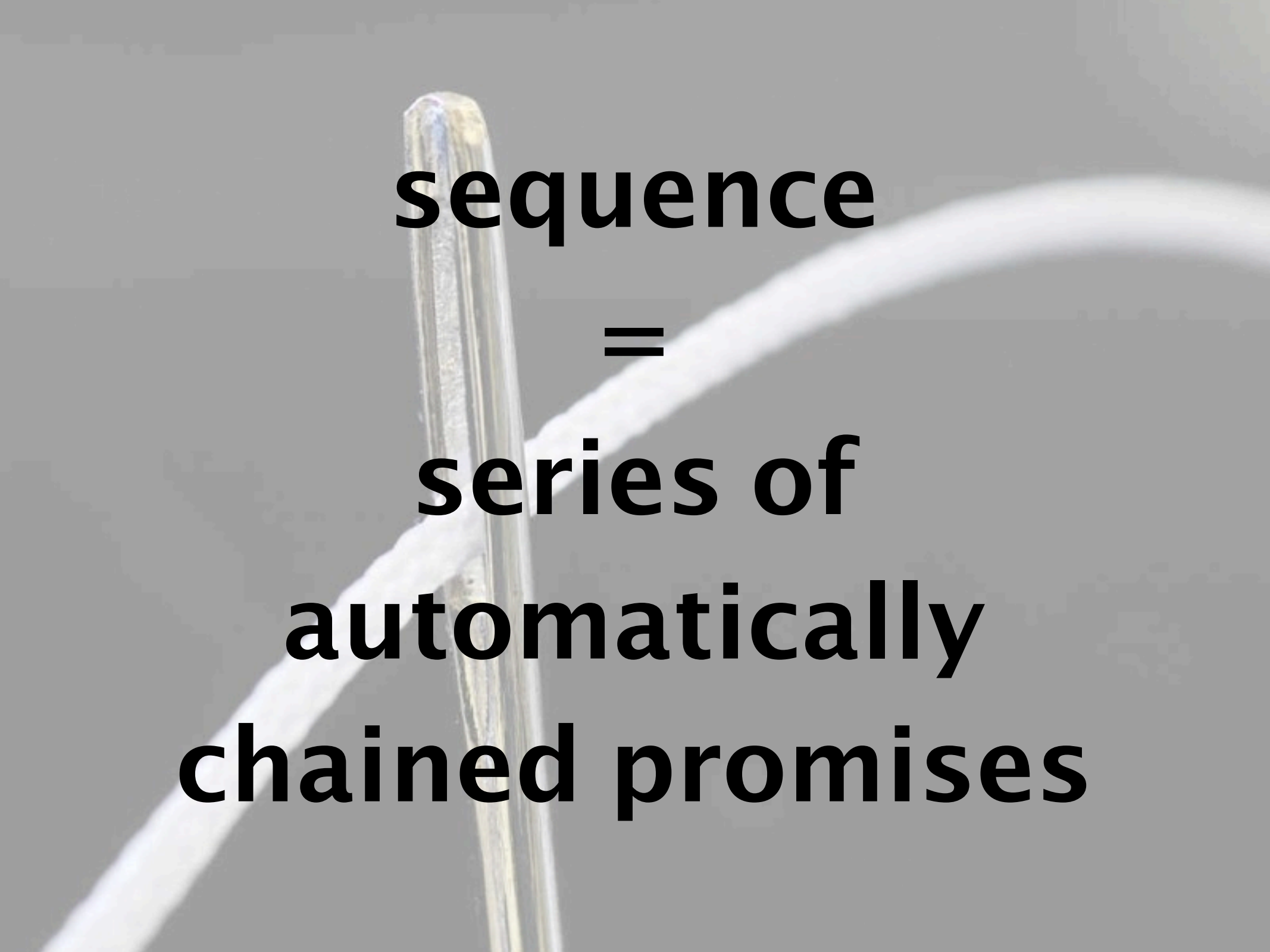
```
["file1","file2","file3"]
.map(getFile)
.reduce(
    function(chain,filePromise){
        return chain
            .then(function(){
                return filePromise;
            })
            .then(output);
    },
    // fulfilled promise to start chain
    Promise.resolve()
)
.then(function() {
    output("Complete!");
});
```

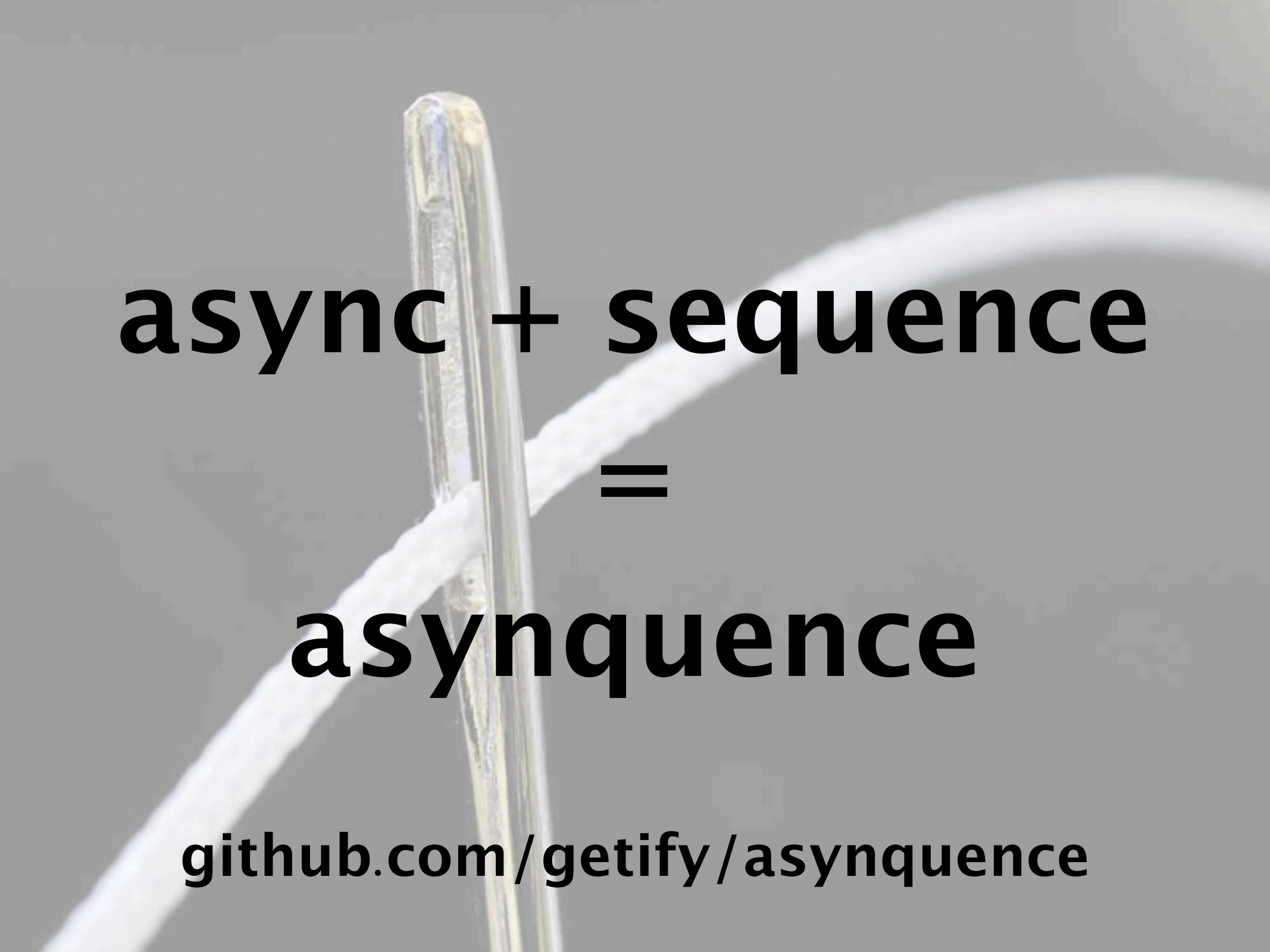

scenario:
timeout of a promise

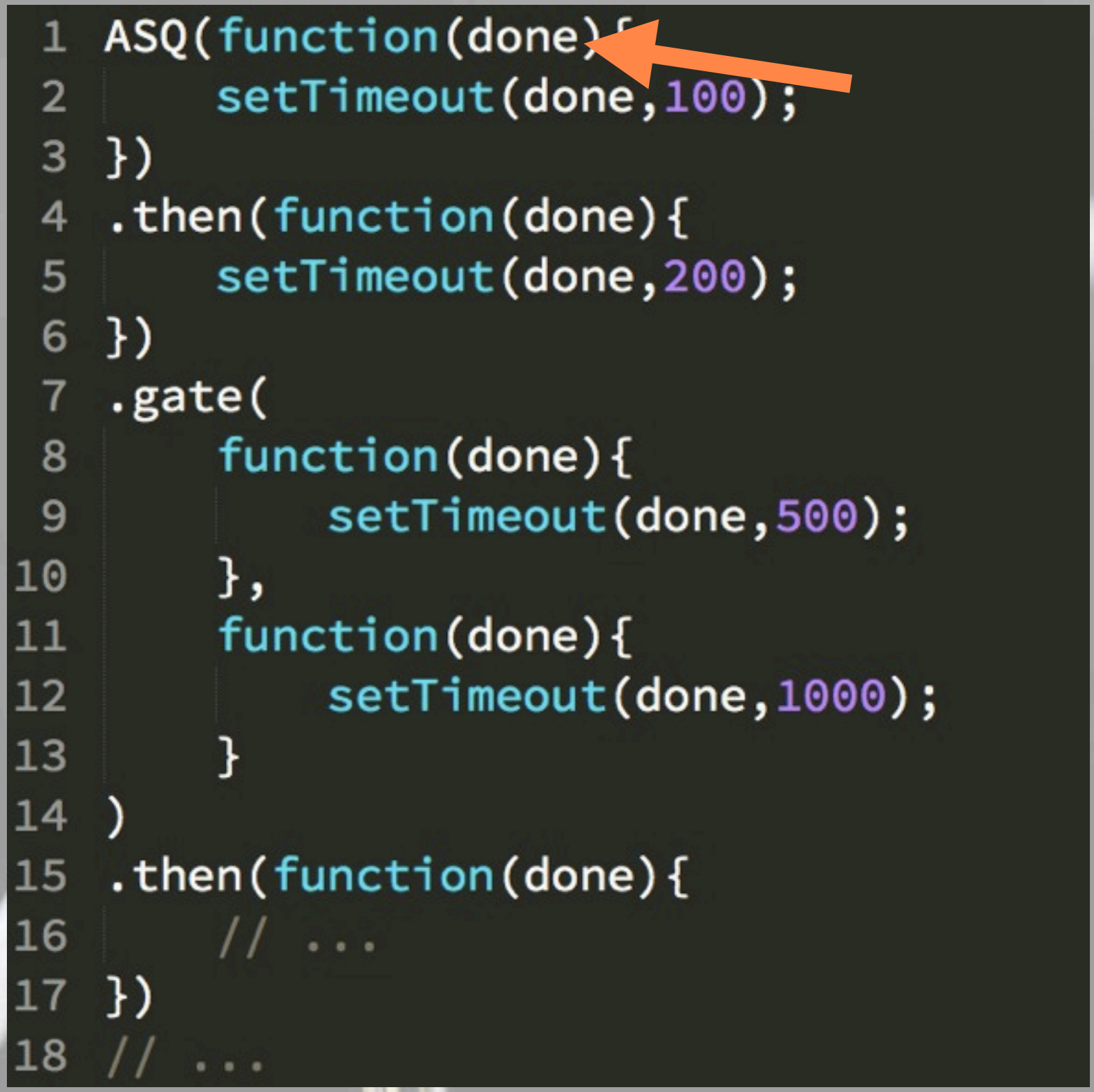

var p = trySomeAsyncThing();
Promise.race([
p,
new Promise(function(_,reject){
setTimeout(function(){
reject("Timeout!!");
},3000);
})
])
.then(
success,
error
);


blog.getify.com/promises-part-1/
github.com/getify/native-promise-only

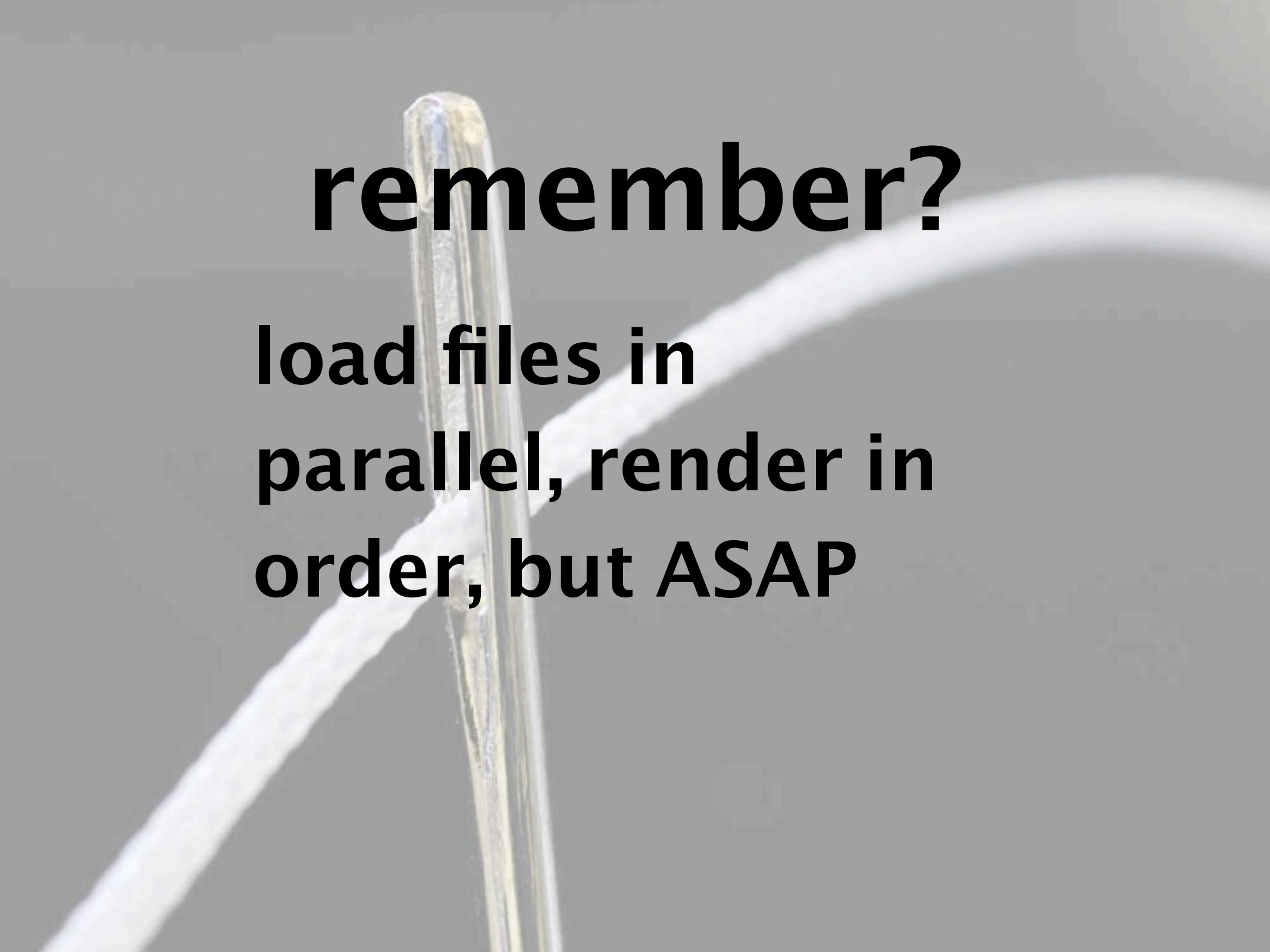

promise
abstractions

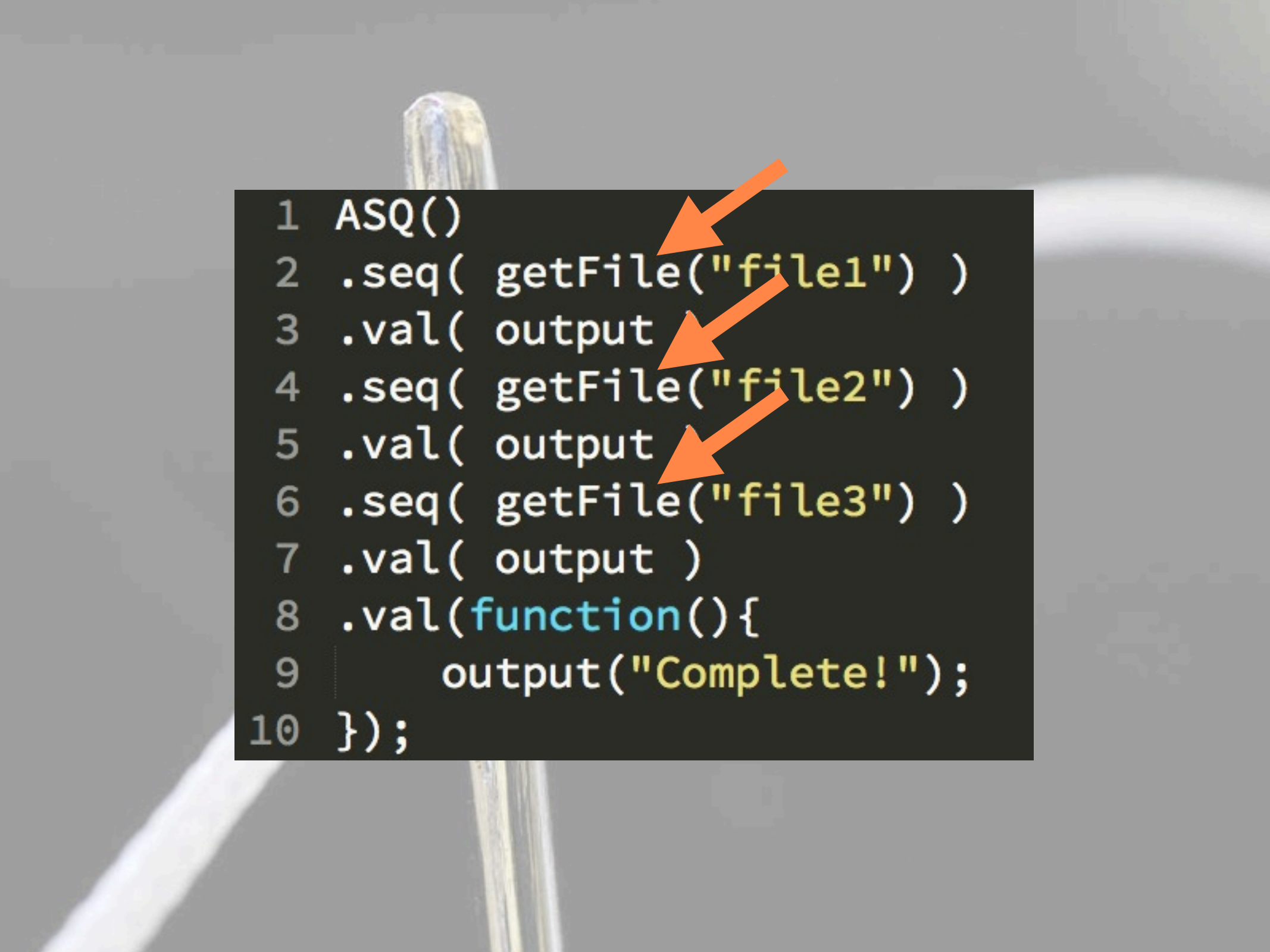

sequence
=
series of
automatically
chained promises


async + sequence
=
asynquence
github.com/getify/asynquence



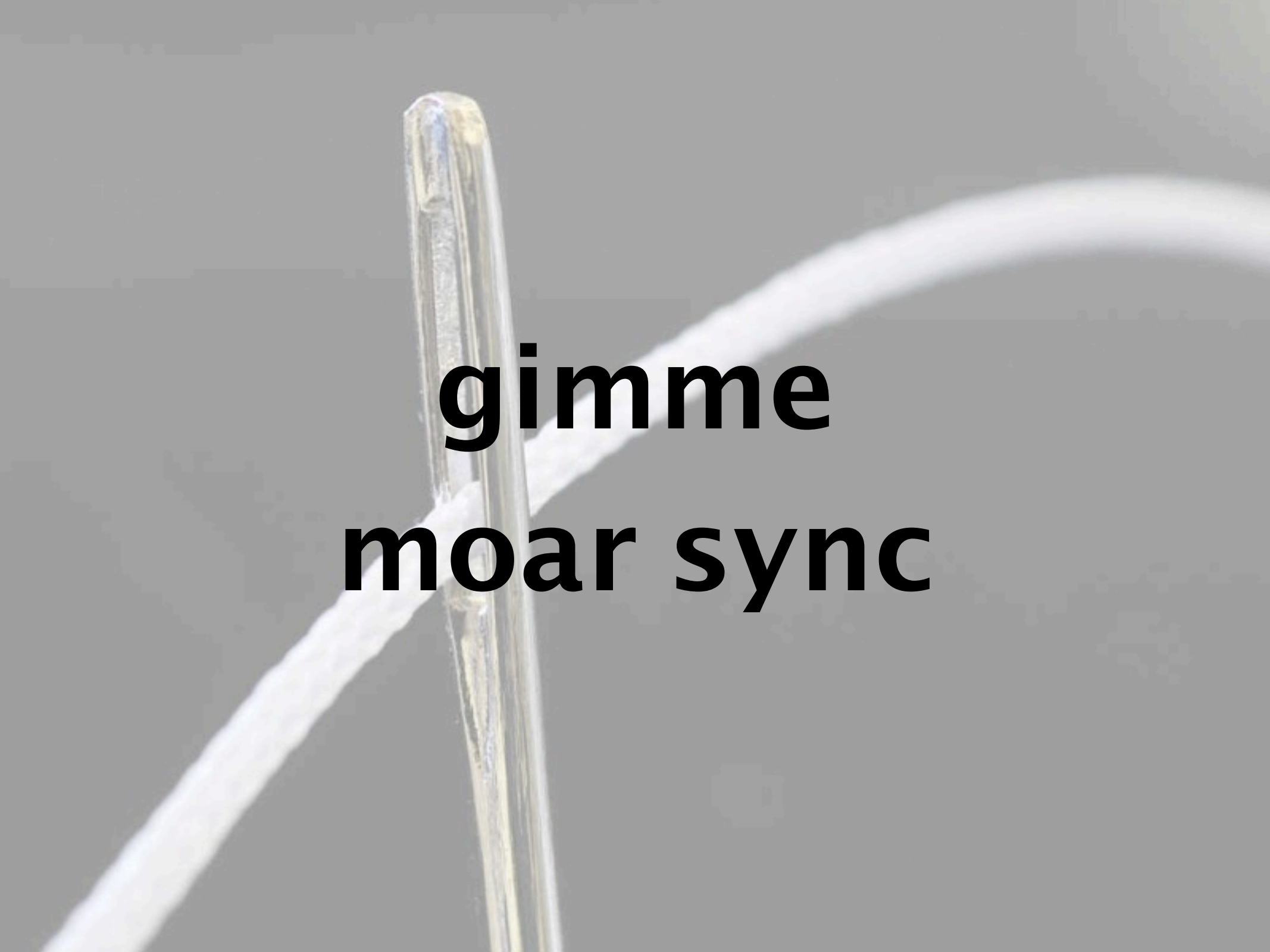
```
ASQ(function(done){
    setTimeout(done,100);
})
.then(function(done){
    setTimeout(done,200);
})
.gate(
    function(done){
        setTimeout(done,500);
    },
    function(done){
        setTimeout(done,1000);
    }
)
.then(function(done){
    // ...
})
// ...
```


remember?
load files in
parallel, render in
order, but ASAP


ASQ()
.seq(getFile("file1"))
.val(output)
.seq(getFile("file2"))
.val(output)
.seq(getFile("file3"))
.val(output)
.val(function(){
output("Complete!");
});

```
ASQ()
.seq.apply(null,
    ["file1","file2","file3"]
    .map(getFile)
    .map(function(sq){
        return function(){
            return sq.val(output);
        };
    })
)
.val(function(){
    output("Complete!");
});
```


promises
without all
the fuss

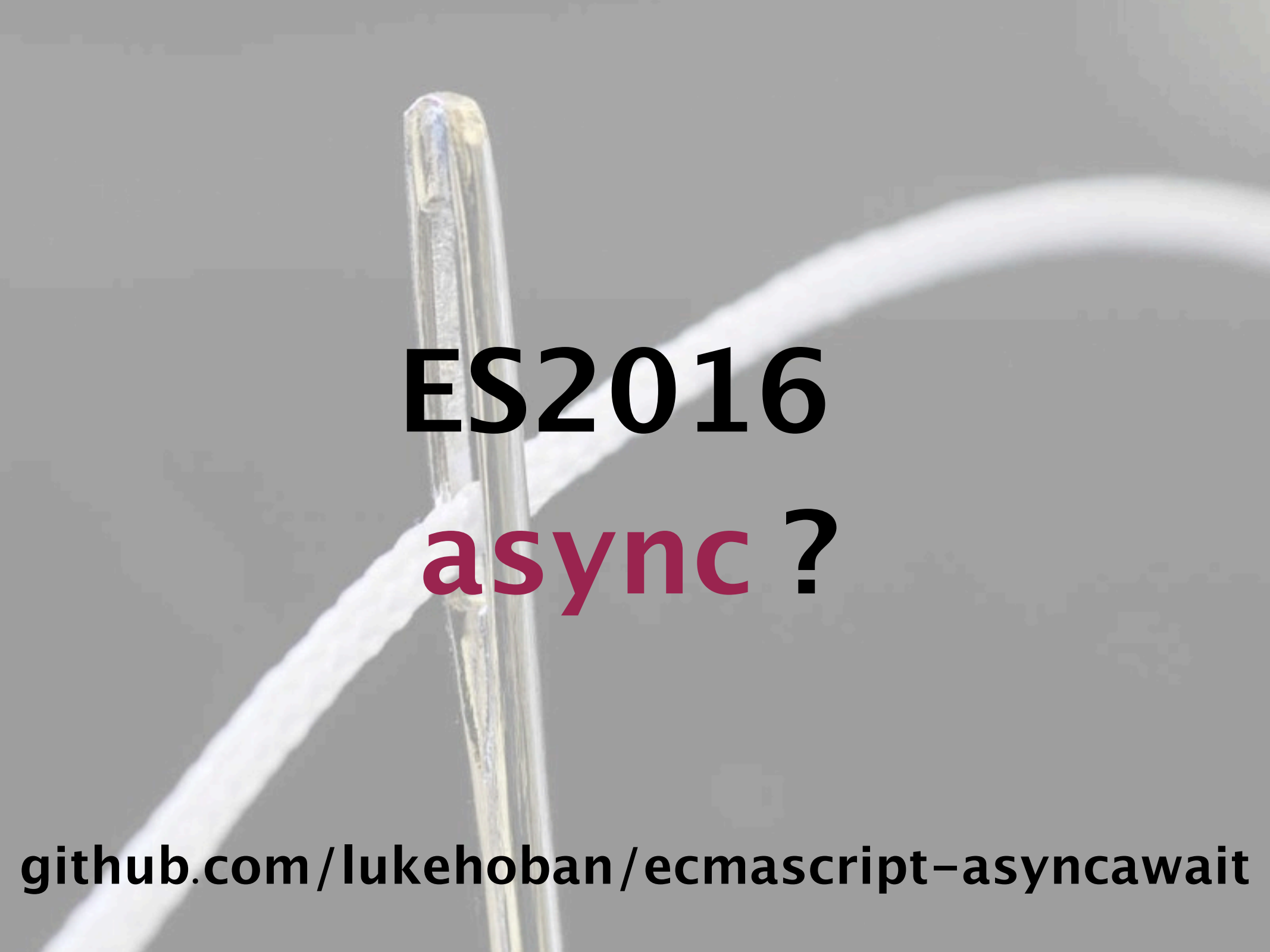

gimme
moar sync

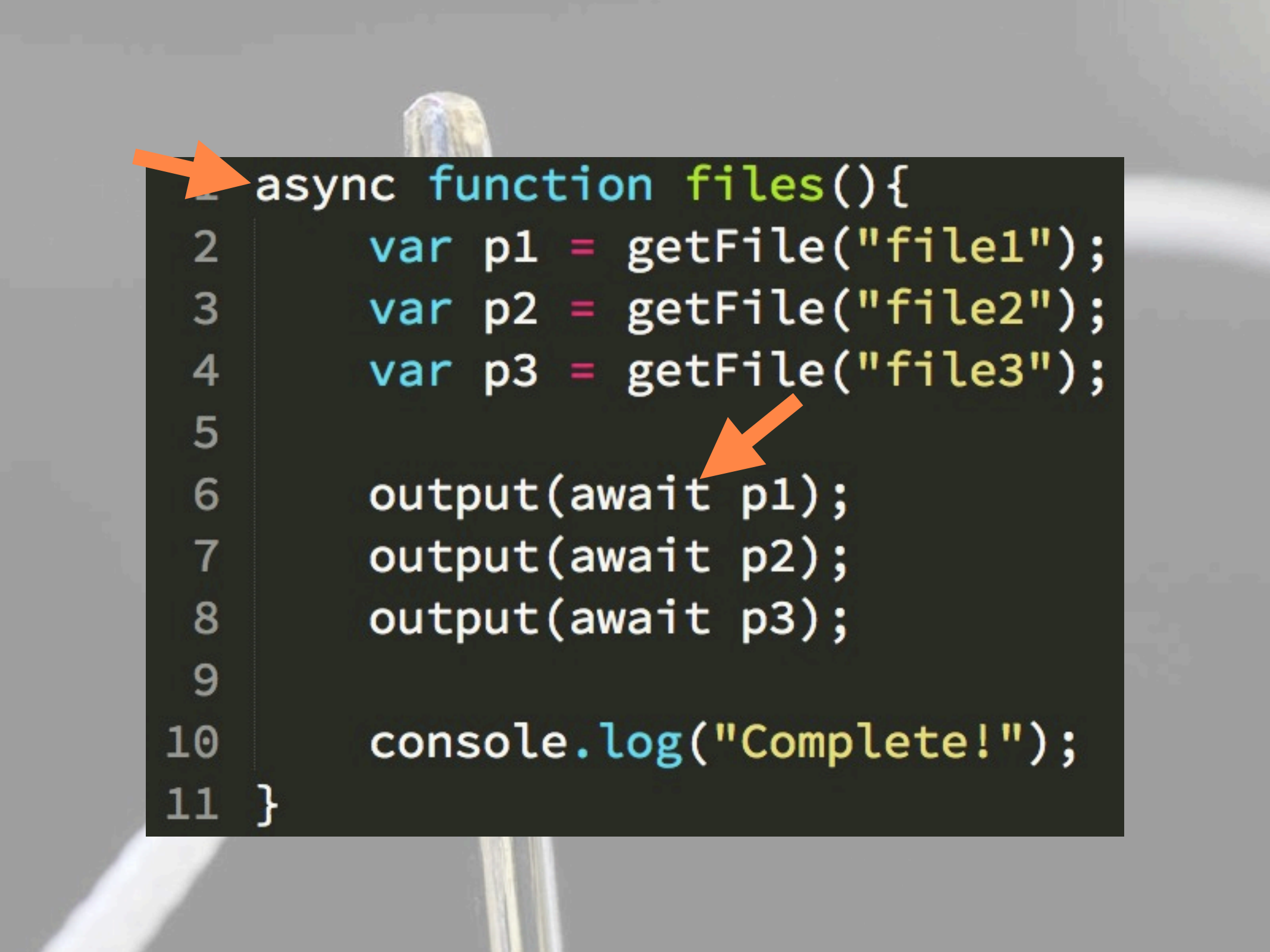

ES6
generators
davidwalsh.name/es6-generators

yield promises

```
run(function*(){
    var p1 = getFile("file1");
    var p2 = getFile("file2");
    var p3 = getFile("file3");

    output(yield p1);
    output(yield p2);
    output(yield p3);

    console.log("Complete!");
});
```

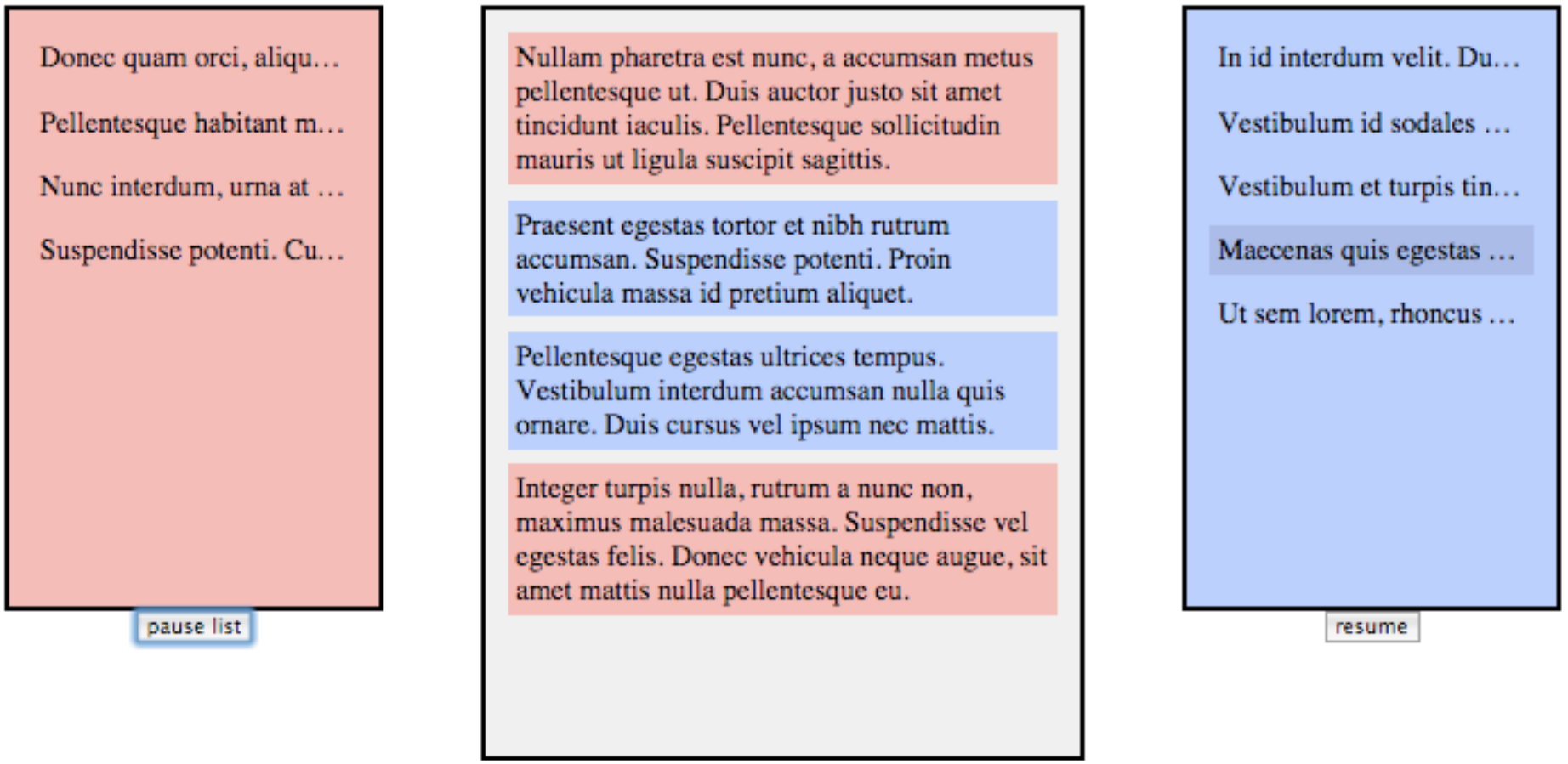
generator + promises


ES2016
async ?
github.com/lukehoban/ecmascript-asyncawait


async function files(){
var p1 = getFile("file1");
var p2 = getFile("file2");
var p3 = getFile("file3");
output(await p1);
output(await p2);
output(await p3);
console.log("Complete!");
}


One more thing...

A Tale Of Three Lists (Callbacks)



github.com/getify/a-tale-of-three-lists

“if you like it then you shoulda put async in it…”


thx!
kyle simpson
@getify
http://getify.me