



syncing async

kyle simpson
@getify
<http://getify.me>



parallel

vs.

async





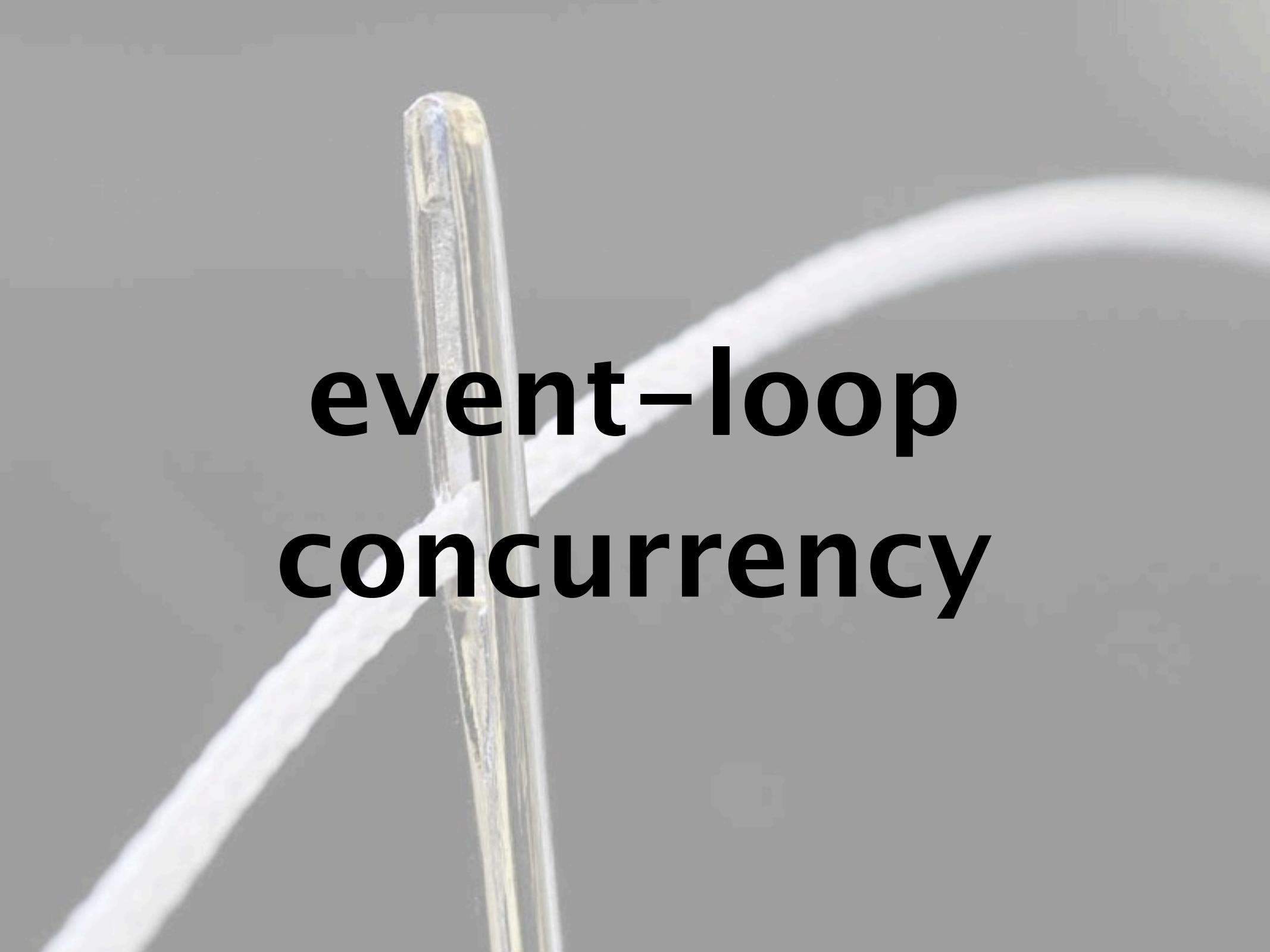
threads



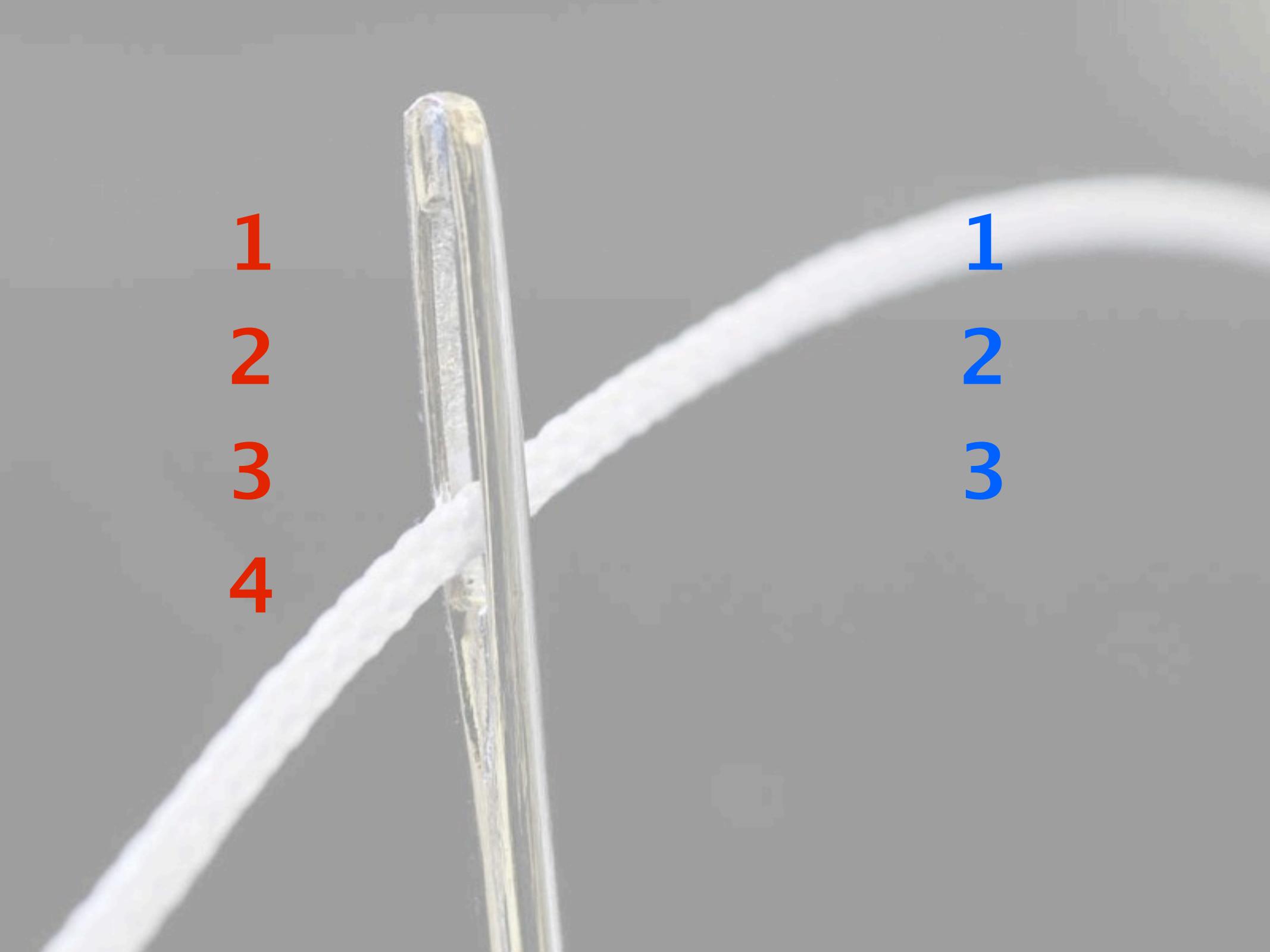








**event-loop
concurrency**



1

2

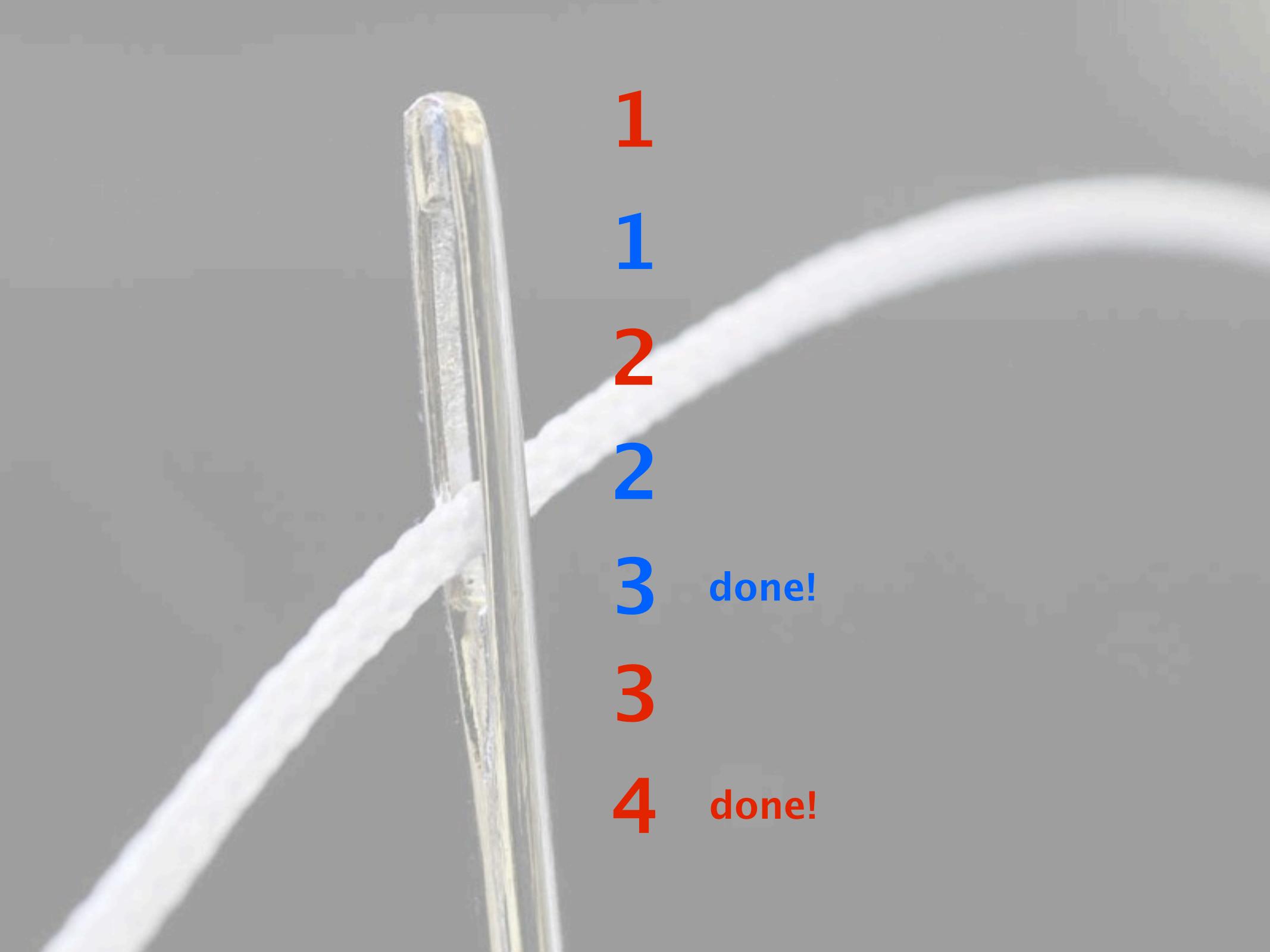
3

4

1

2

3



1

1

2

2

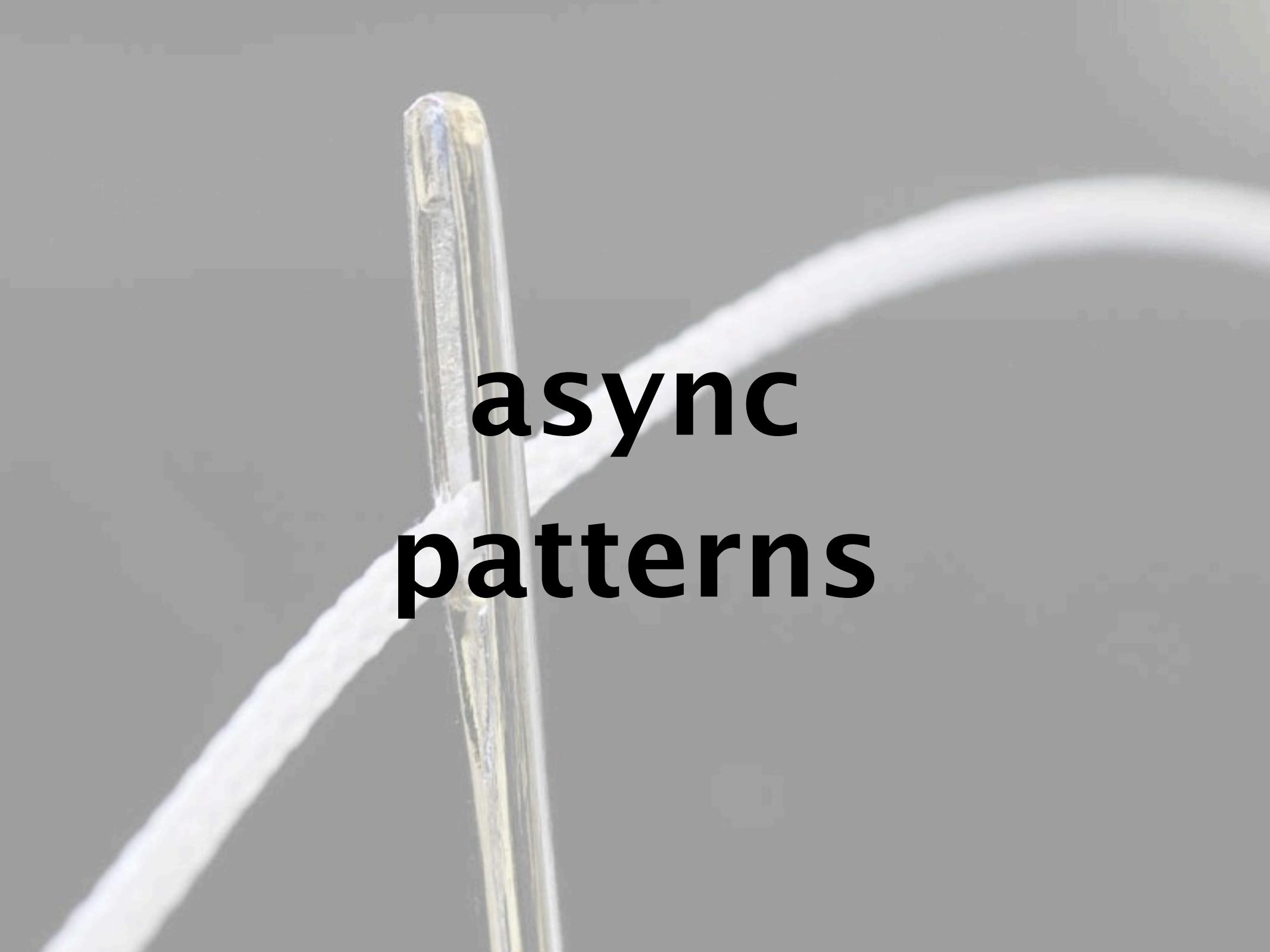
3

done!

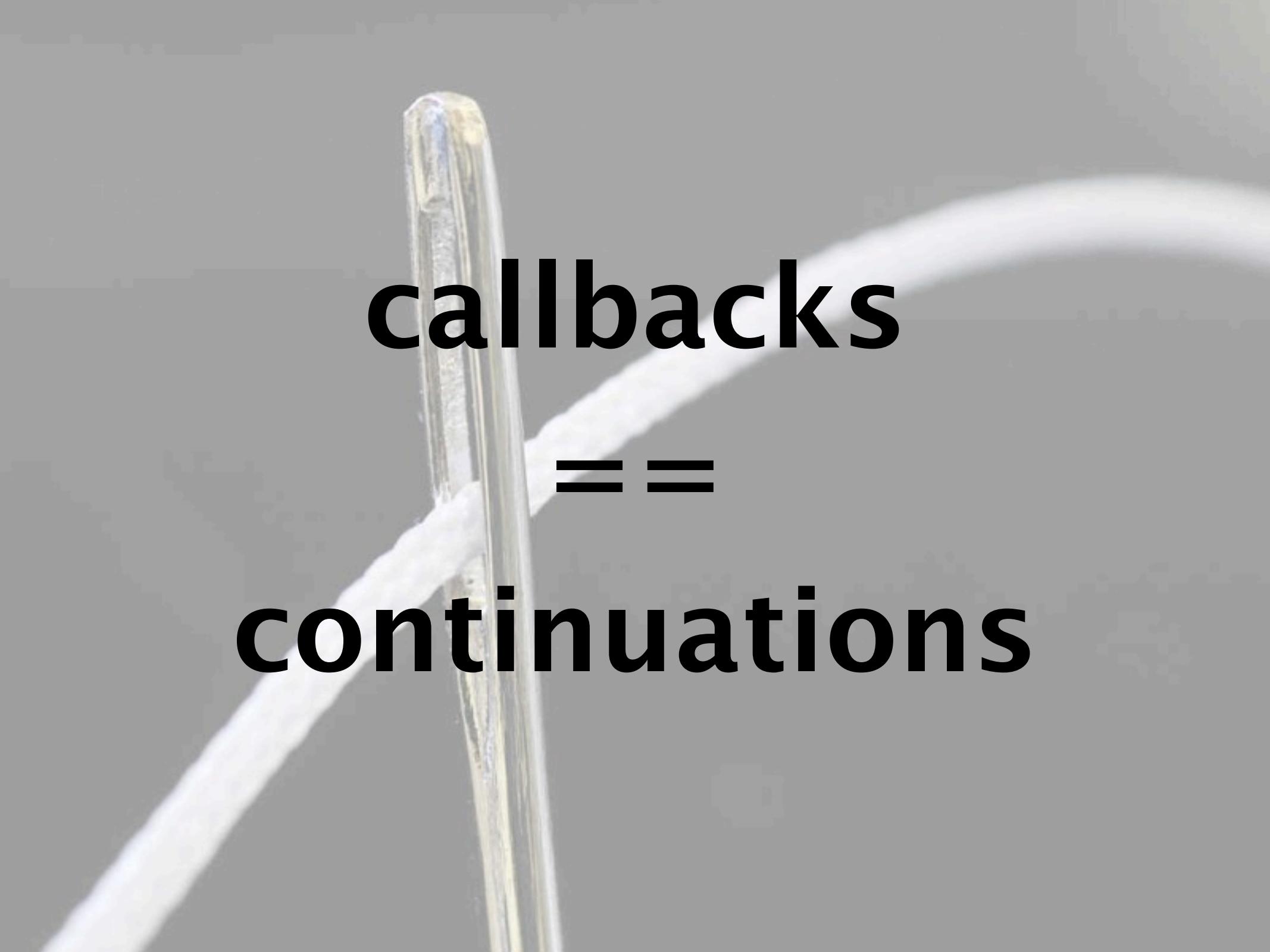
3

4

done!



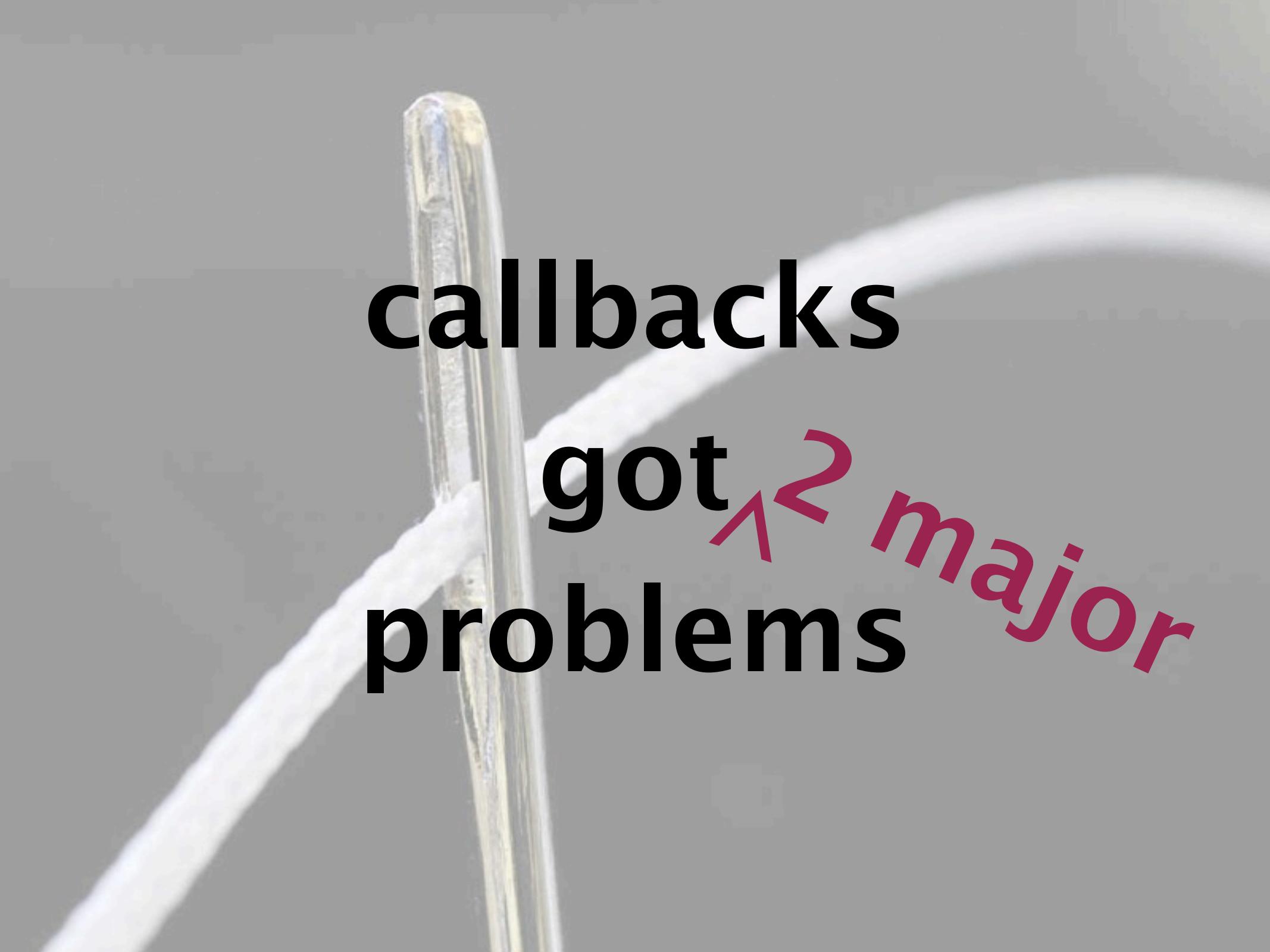
**async
patterns**



callbacks

==

continuations



callbacks

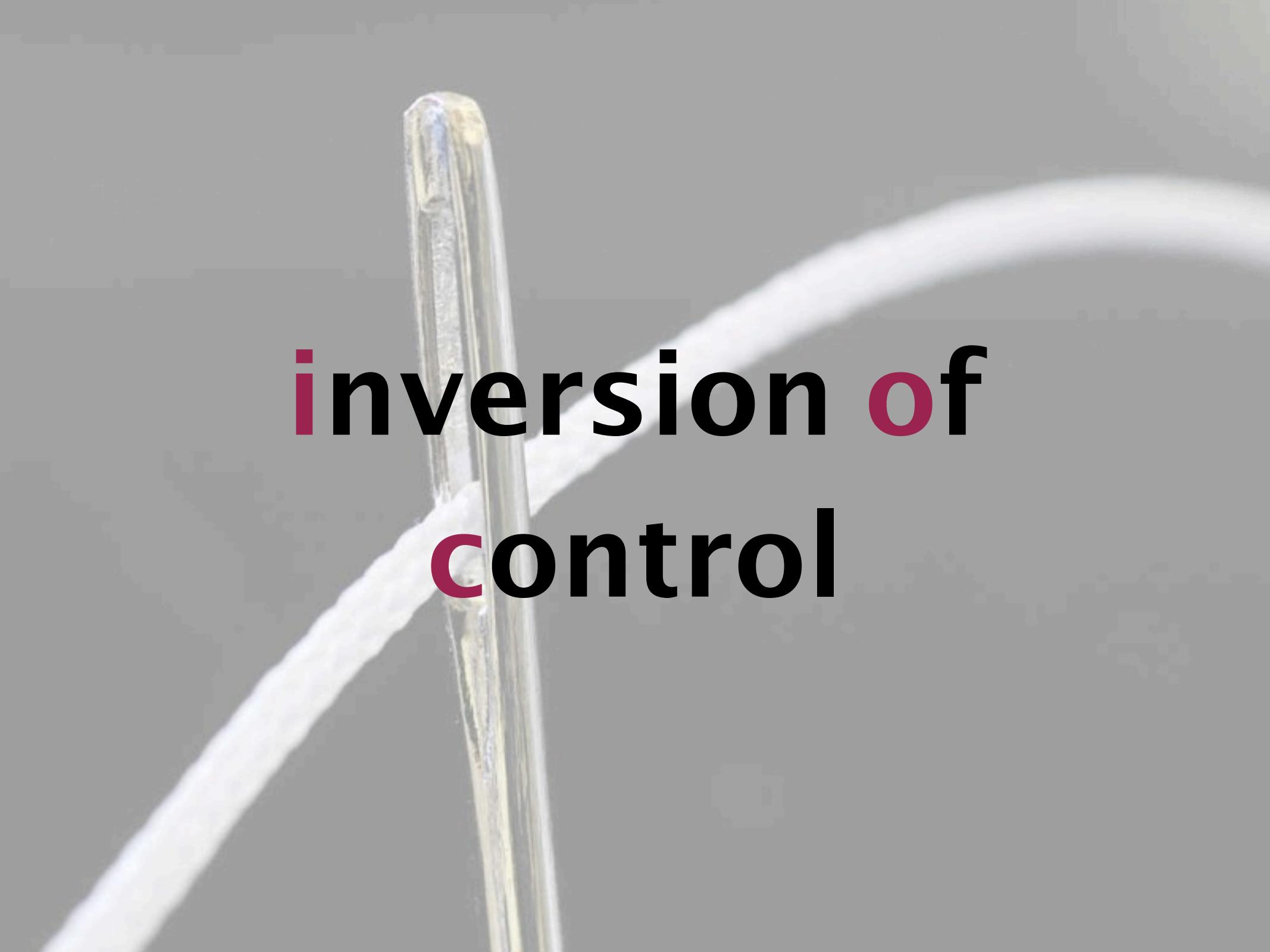
got ¹² major
problems

```
1 setTimeout(function(){
2     console.log("one");
3     setTimeout(function(){
4         console.log("two");
5         setTimeout(function(){
6             console.log("three");
7             },1000);
8         },1000);
9     },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**

```
1 trackCheckout(  
2     purchaseInfo,  
3     function finish() {  
4         chargeCreditCard(purchaseInfo);  
5         showThankYouPage();  
6     }  
7 );
```

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

...

```
1 var hasBeenCalled = false;  
2  
3 trackCheckout(  
4     purchaseInfo,  
5     function finish() {  
6         if (!hasBeenCalled) {  
7             hasBeenCalled = true;  
8             chargeCreditCard(purchaseInfo);  
9             showThankYouPage();  
10        }  
11    }  
12 );
```



nested callbacks
are not
reasonable

```
1 start task1:  
2     do some stuff  
3     pause  
4  
5 start task2:  
6     do some other stuff  
7     pause  
8  
9 resume task1:  
10    do more stuff  
11    pause  
12  
13 resume task2:  
14    finish stuff  
15  
16 resume task1:  
17    finish stuff
```

```
1 start task1:  
2     do some stuff  
3     pause  
4  
5     resume task1:  
6         do more stuff  
7         pause  
8  
9     resume task1:  
10        finish stuff  
11  
12  
13 start task2:  
14     do some other stuff  
15     pause  
16  
17     resume task2:  
18        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

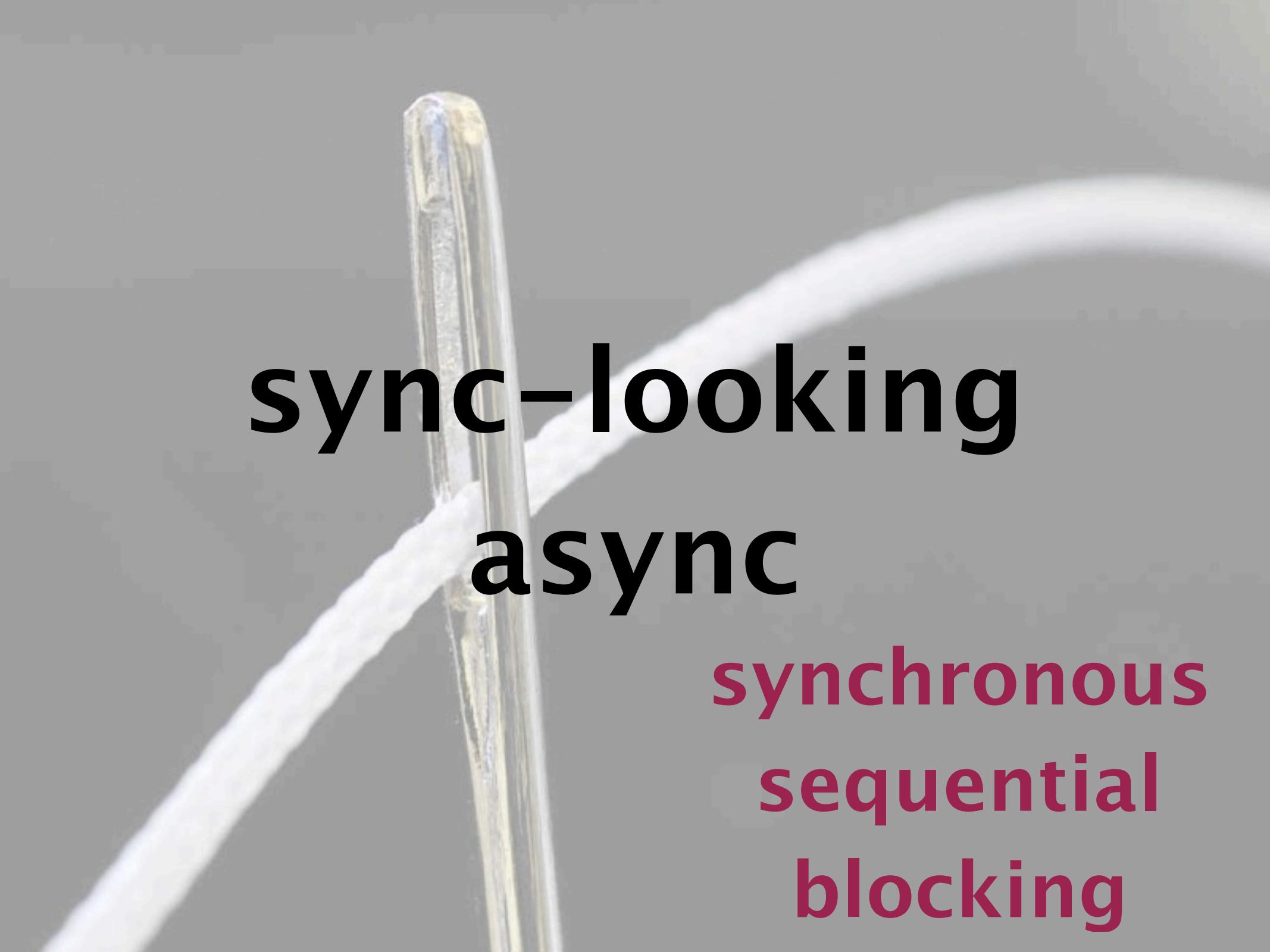
```
1 console.log("First half of my program");
2
3 block(1000);
4
5 console.log("Second half of my program");
6
7
```

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

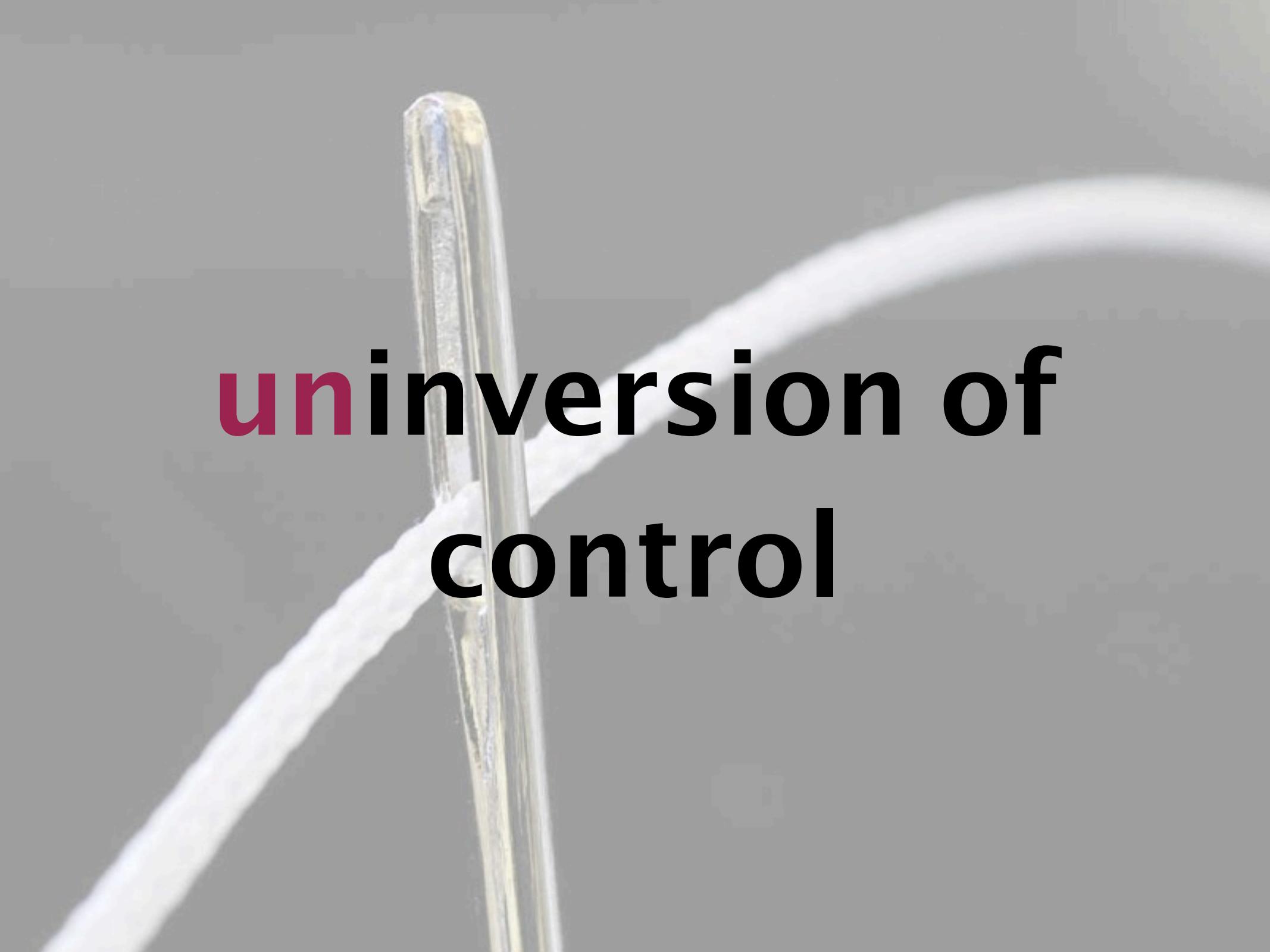
```
1 function trackCheckout(info) {  
2     return new Promise(  
3         function(resolve,reject){  
4             // attempt to track the checkout  
5             // if successful, call resolve()  
6             // otherwise, call reject(error)  
7         }  
8     );  
9 }  
10 }
```

```
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**

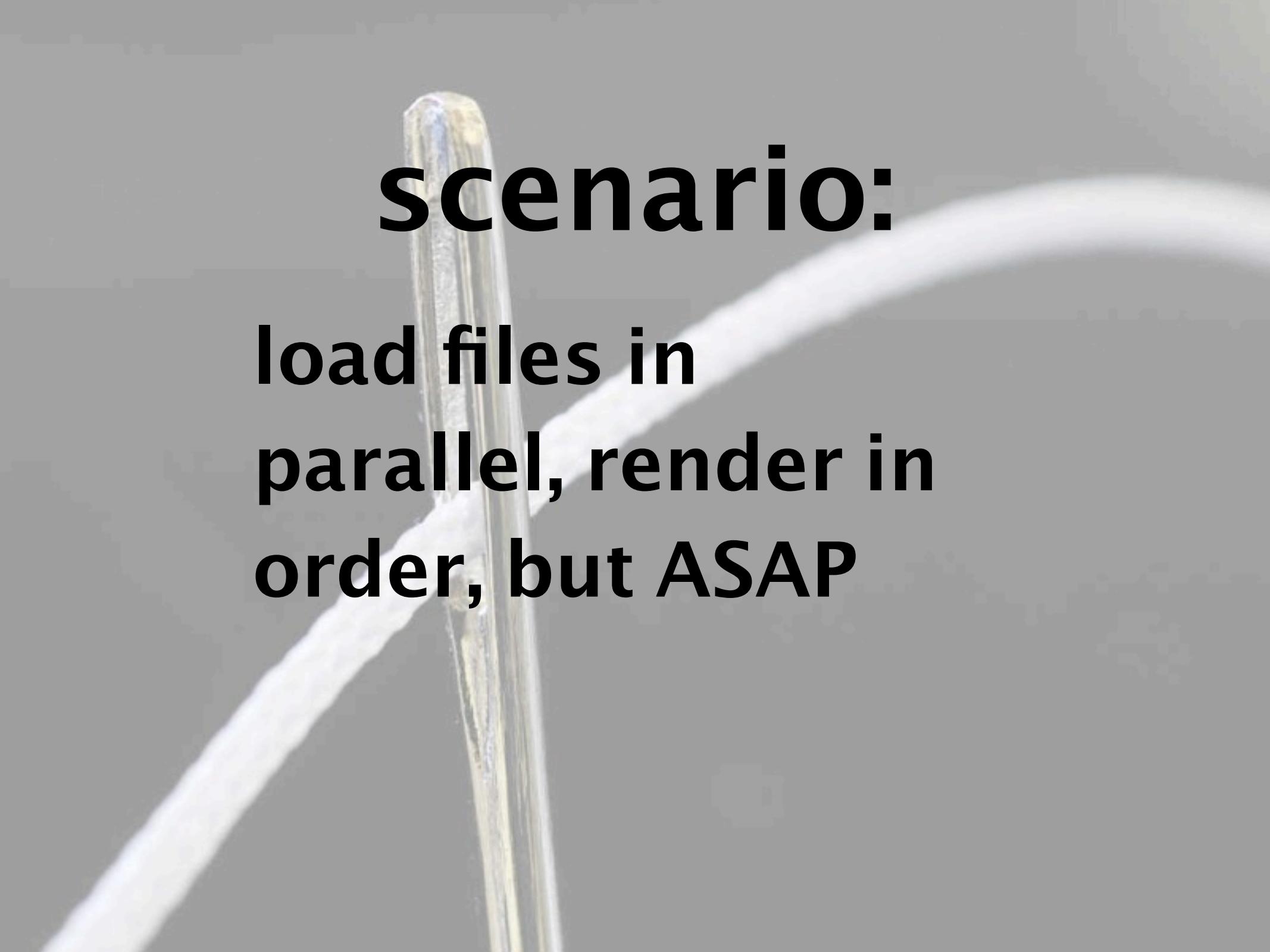


steps

```
1 doFirstThing
2   then doSecondThing
3   then doThirdThing
4   then complete
5 or error
```

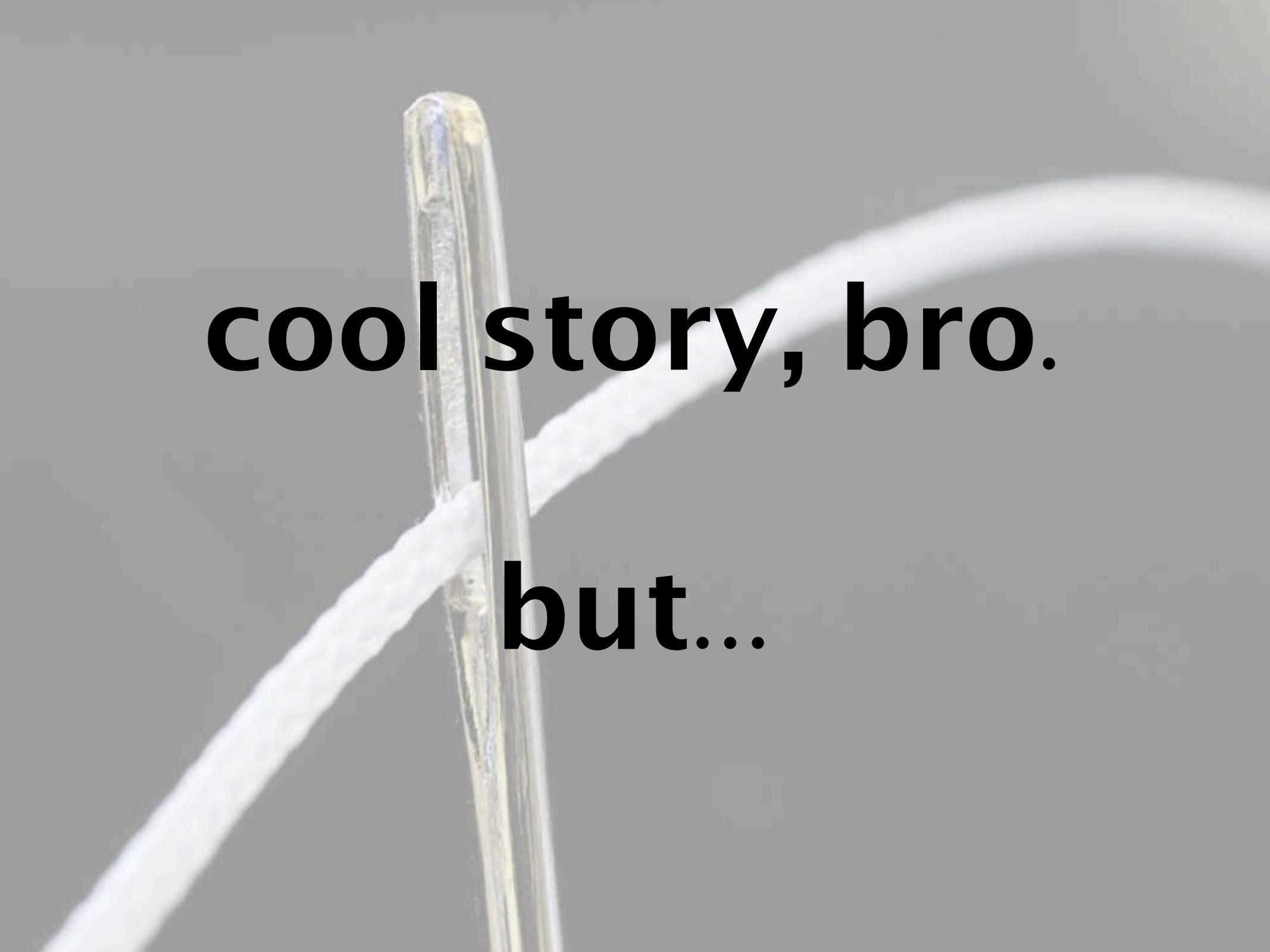
promise: flow control

```
1 doFirstThing()  
2 .then(function(){  
3     return doSecondThing();  
4 })  
5 .then(function(){  
6     return doThirdThing();  
7 })  
8 .then(  
9     complete,  
10    error  
11 );
```



scenario:
load files in
parallel, render in
order, but ASAP

```
1 // Request all files at once in "parallel"
2 var p1 = getFile("file1");
3 var p2 = getFile("file2");
4 var p3 = getFile("file3");
5
6 // Render as each one finishes, but only once
7 // previous rendering is done.
8 p1
9 .then(output)
10 .then(function(){
11     return p2;
12 })
13 .then(output)
14 .then(function(){
15     return p3;
16 })
17 .then(output)
18 .then(function(){
19     output("Complete!");
20 }));
```



cool story, bro.

but...

```
1 ["file1","file2","file3"]
2 .map(getFile)
3 .reduce(
4     function(chain,filePromise){
5         return chain
6             .then(function(){
7                 return filePromise;
8             })
9             .then(output);
10    },
11    // fulfilled promise to start chain
12    Promise.resolve()
13 )
14 .then(function() {
15     output("Complete!");
16 });
```



scenario:
timeout of a promise

```
1 var p = trySomeAsyncThing();
2
3 Promise.race([
4     p,
5     new Promise(function(_,reject){
6         setTimeout(function(){
7             reject("Timeout!!");
8         },3000);
9     })
10 ])
11 .then(
12     success,
13     error
14 );
```

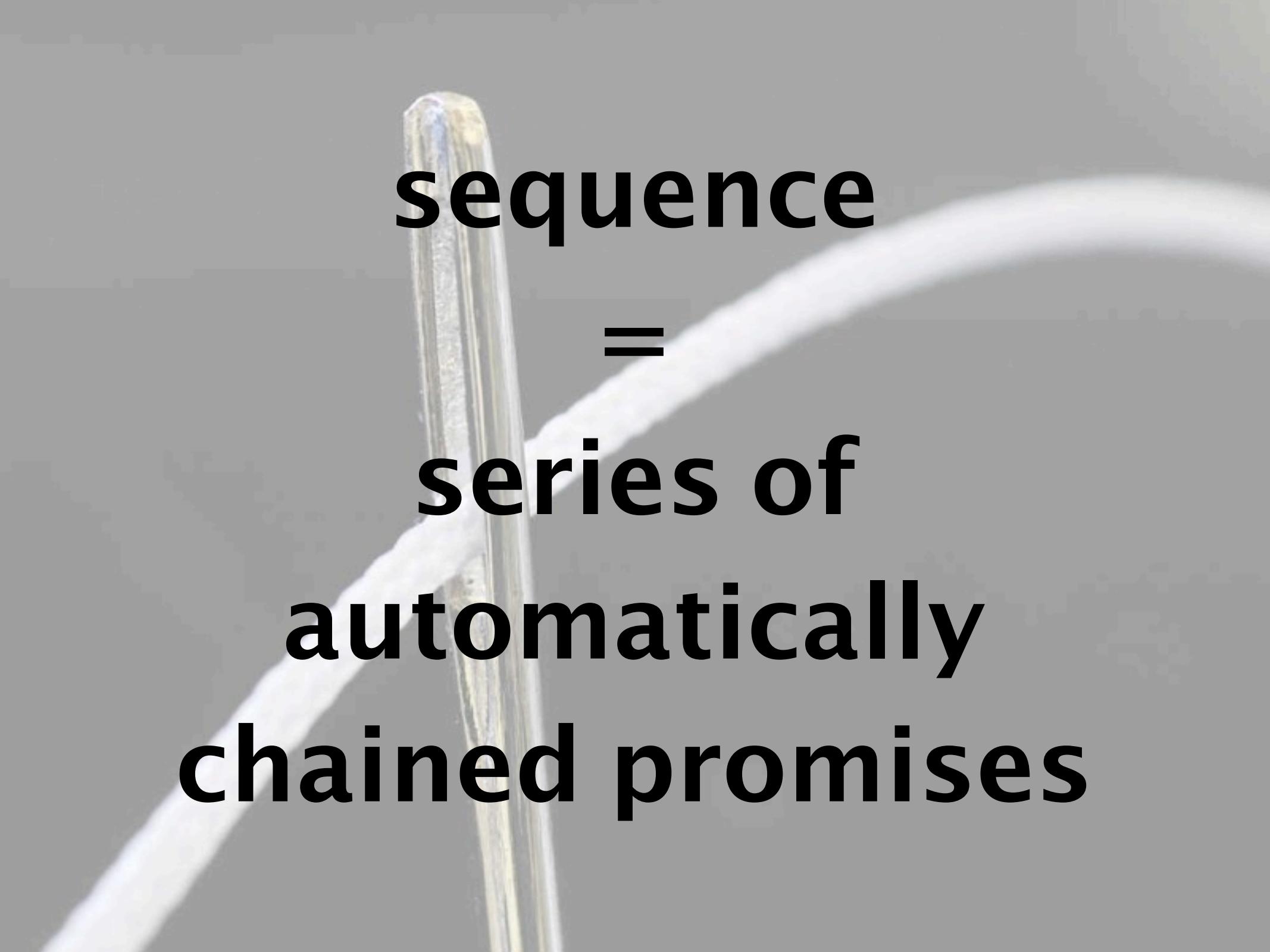


blog.getify.com/promises-part-1/

github.com/getify/native-promise-only

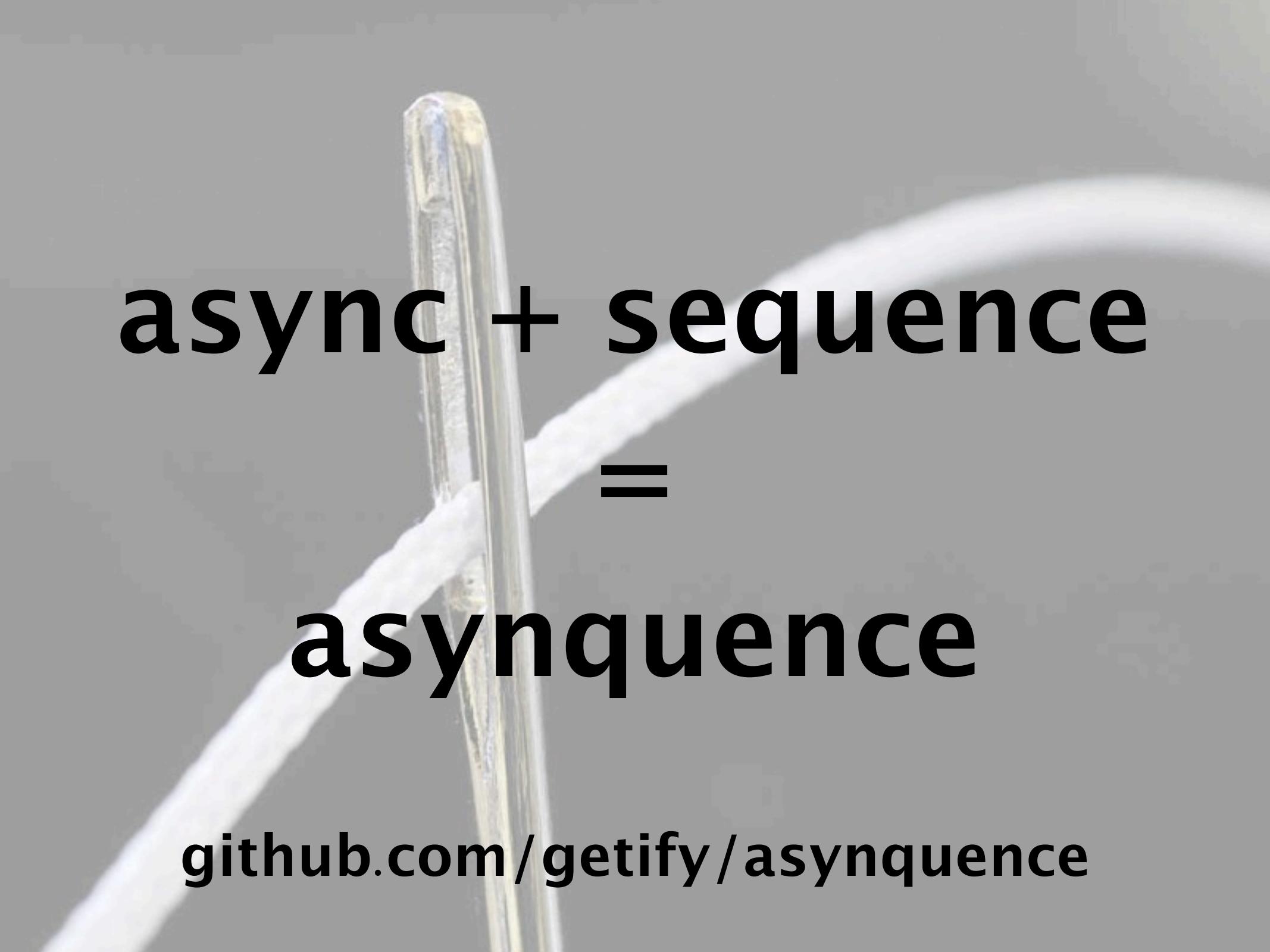


**promise
abstractions**



sequence
=

**series of
automatically
chained promises**



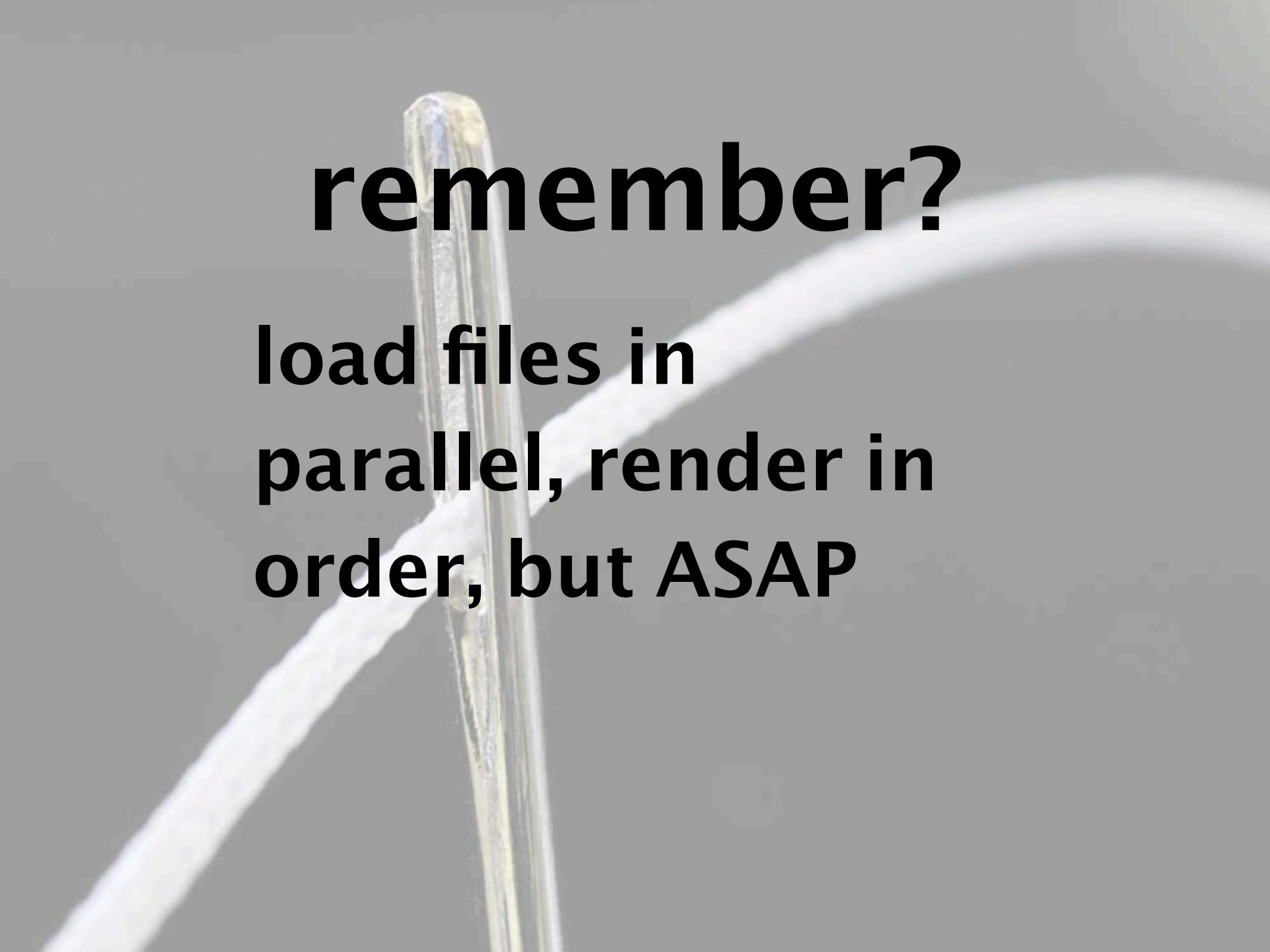
async + sequence

=

asynquence

github.com/getify/asynquence

```
1 ASQ(function(done){  
2     setTimeout(done,100);  
3 })  
4 .then(function(done){  
5     setTimeout(done,200);  
6 })  
7 .gate(  
8     function(done){  
9         setTimeout(done,500);  
10    },  
11    function(done){  
12        setTimeout(done,1000);  
13    }  
14 )  
15 .then(function(done){  
16     // ...  
17 })  
18 // ...
```



remember?

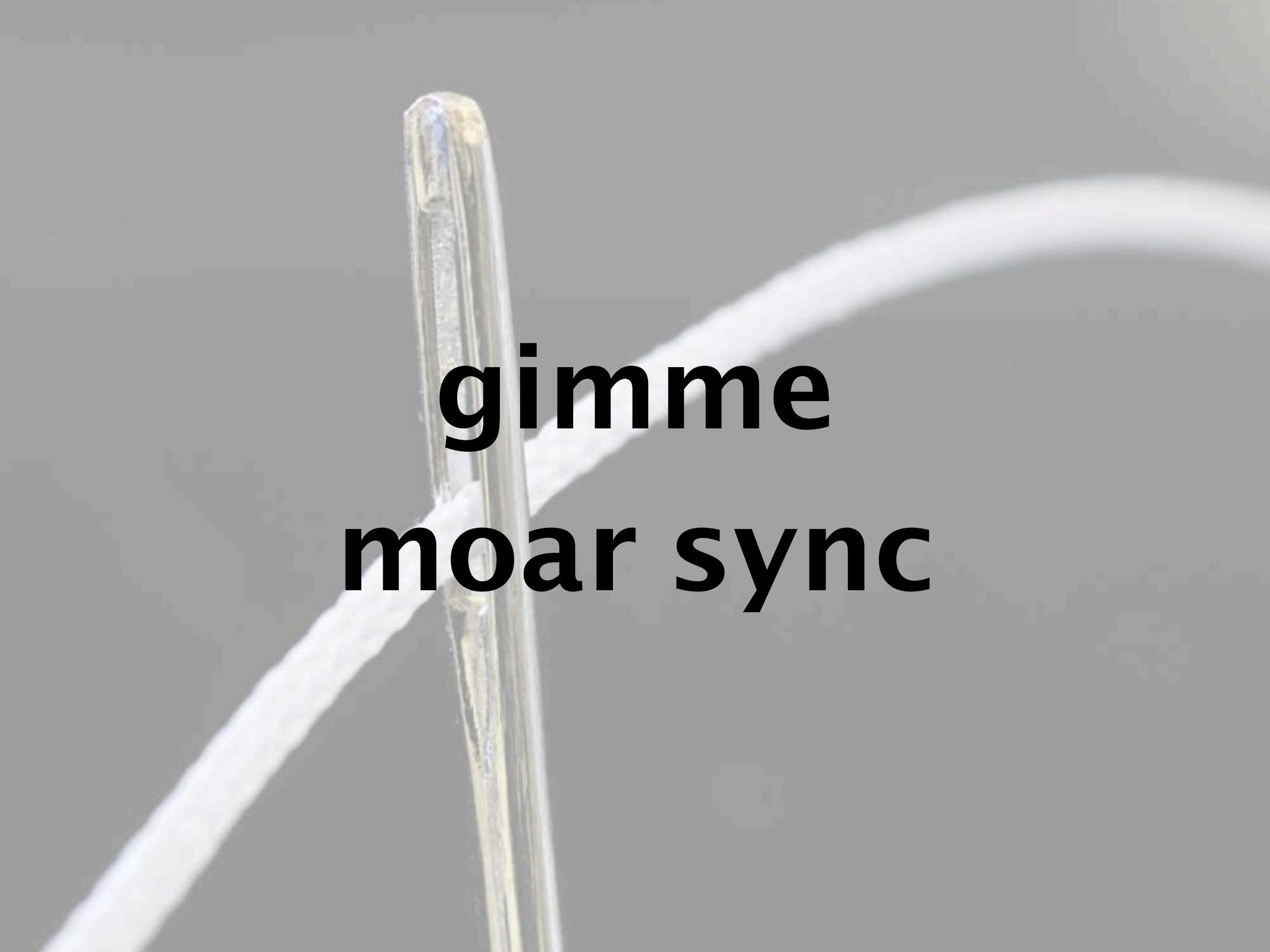
**load files in
parallel, render in
order, but ASAP**

```
1 ASQ()  
2 .seq( getFile("file1") )  
3 .val( output )  
4 .seq( getFile("file2") )  
5 .val( output )  
6 .seq( getFile("file3") )  
7 .val( output )  
8 .val(function(){  
9     output("Complete!");  
10});
```

```
1 ASQ()  
2 .seq.apply(null,  
3   ["file1","file2","file3"]  
4   .map(getFile)  
5   .map(function(sq){  
6     return function(){  
7       return sq.val(output);  
8     };  
9   })  
10 )  
11 .val(function(){  
12   output("Complete!");  
13 });
```



**promises
without all
the fuss**



**gimme
moar sync**

A close-up photograph of a vintage-style incandescent lightbulb. The bulb is cylindrical with a clear glass body showing internal filaments. It is illuminated, emitting a warm, yellowish-orange glow that fades into the surrounding darkness. The background is a solid, dark gray.

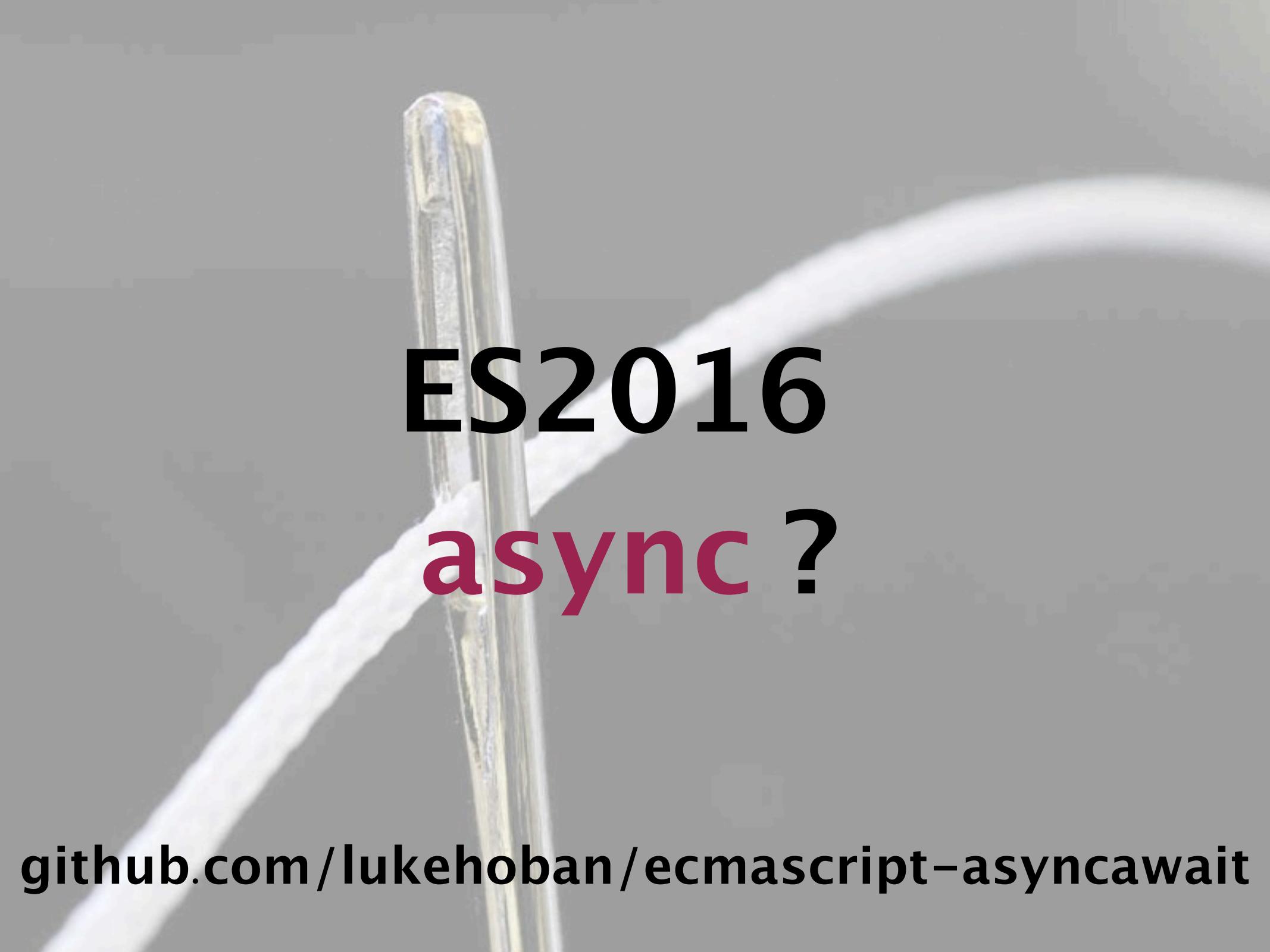
ES6 generators

davidwalsh.name/es6-generators

yield promises

```
1 run(function*(){
2     var p1 = getFile("file1");
3     var p2 = getFile("file2");
4     var p3 = getFile("file3");
5
6     output(yield p1);
7     output(yield p2);
8     output(yield p3);
9
10    console.log("Complete!");
11});
```

generator + promises



ES2016 async ?

github.com/lukehoban/ecmascript-asyncawait

```
1  async function files(){
2      var p1 = getFile("file1");
3      var p2 = getFile("file2");
4      var p3 = getFile("file3");
5
6      output(await p1);
7      output(await p2);
8      output(await p3);
9
10     console.log("Complete!");
11 }
```

One more thing...



A Tale Of Three Lists (Callbacks)

Donec quam orci, aliqu...

Pellentesque habitant m...

Nunc interdum, urna at ...

Suspendisse potenti. Cu...

[pause](#) [list](#)

Nullam pharetra est nunc, a accumsan metus
pellentesque ut. Duis auctor justo sit amet
tincidunt iaculis. Pellentesque sollicitudin
mauris ut ligula suscipit sagittis.

Praesent egestas tortor et nibh rutrum
accumsan. Suspendisse potenti. Proin
vehicula massa id pretium aliquet.

Pellentesque egestas ultrices tempus.
Vestibulum interdum accumsan nulla quis
ornare. Duis cursus vel ipsum nec mattis.

Integer turpis nulla, rutrum a nunc non,
maximus malesuada massa. Suspendisse vel
egestas felis. Donec vehicula neque augue, sit
amet mattis nulla pellentesque eu.

In id interdum velit. Du...

Vestibulum id sodales ...

Vestibulum et turpis tin...

Maecenas quis egestas ...

Ut sem lorem, rhoncus ...

[resume](#)

“if you like it then you shoulda put **async** in it...”





thx!

kyle simpson
@getify
<http://getify.me>