Async JavaScript and Promises

Callbacks

- Functions are first class citizens in JavaScript
- Functions can takes functions as argument and call them when they are done

```
console.log("Before readFile");
fs.readFile("file1.txt", function(file1Content) {
  console.log("File Contents here");
  return "#" + file1Content + "#";
});
console.log("After readFile");
```

Callbacks

- Functions are first class citizens in JavaScript
- Functions can takes functions as argument and call them when they are done

```
console.log("Before readFile");
fs.readFile("file1.txt", function(file1Content) {
  console.log("File Contents here");
  return "#" + file1Content + "#";
});
console.log("After readFile");
```

Is JavaScript really Async?

- All of the JavaScript engines including V8 are single-threaded
- But all I/O is evented and asynchronous
- It is possible via the EventLoop

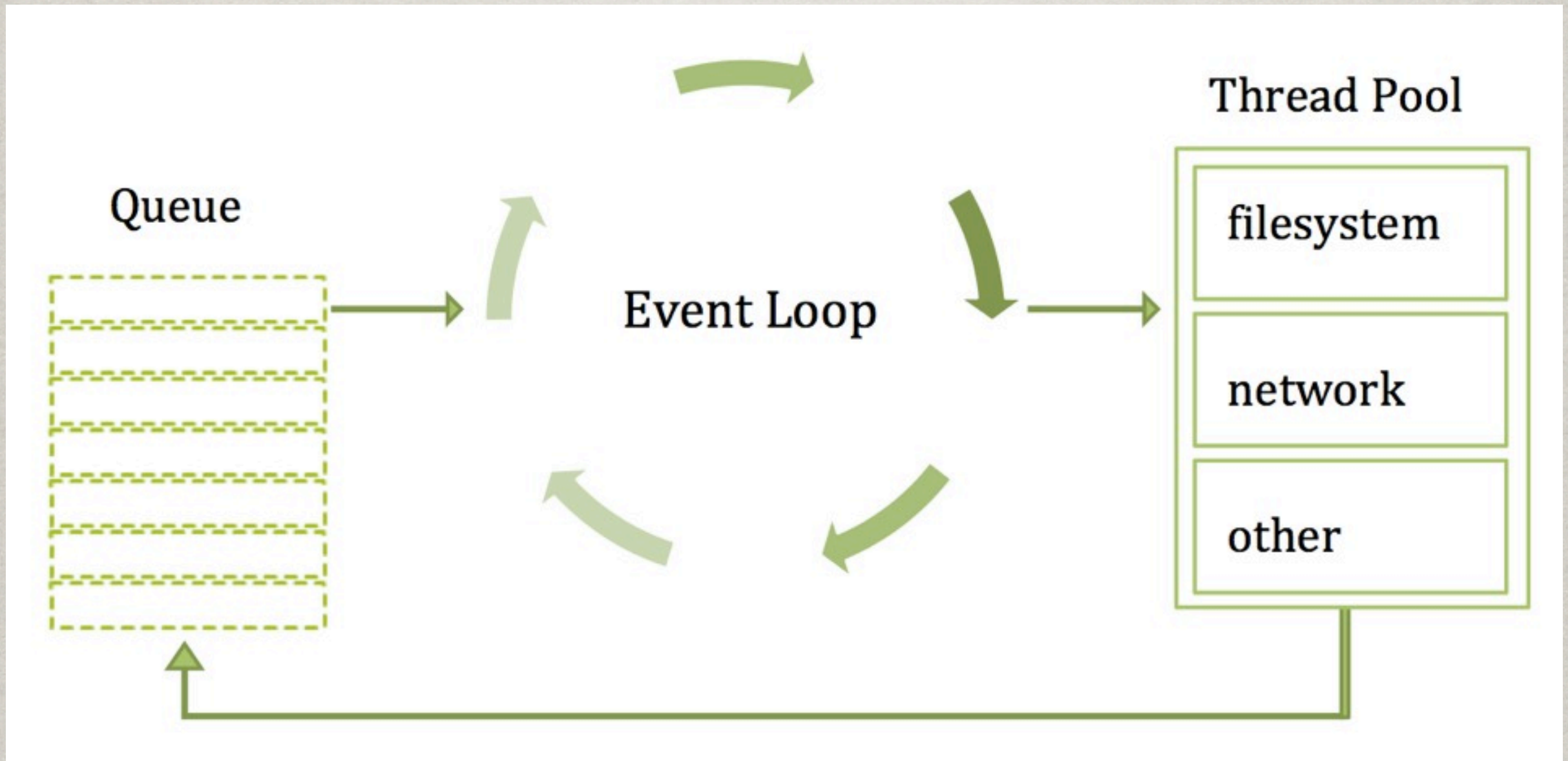
Event Loops

- An entity that handles and processes external events and converts them into callback invocations

- Every asynchronous operation adds itself to the EventLoop

How EventLoop works?

Event Loop



Making Code Asynchronous

- setImmediate
- process.nextTick
- setTimeout / setInterval

Promises

- A Promise is an object that represents a one-time event, typically the outcome of an async task like an AJAX call.

- At first, a Promise is in a pending state. Eventually, it's either resolved or rejected.

- Once a Promise is resolved or rejected, it'll remain in that state forever, and its callbacks will never fire again.

What are Promises For?

Callback Hell

Promises can solve the problem of "Callback Hell"

```
step1(function (value1) {
  step2(value1, function(value2) {
    step3(value2, function(value3) {
      step4(value3, function(value4) {
        // Do something with value4
      });
    });
  });
});
```

```
Q.fcall(promisedStep1)
.then(promisedStep2)
.then(promisedStep3)
.then(promisedStep4)
.then(function (value4) {
    // Do something with value4
})
.done();
```

Handling exceptions

Promises also helps you handle exceptions in a cleaner way

```
var handleError = console.log;

step1(function (value1) {
  step2(value1, function(value2) {
    step3(value2, function(value3) {
      step4(value3, function(value4) {
        // Do something with value4
      }, handleError);
    }, handleError);
  }, handleError);
}, handleError);
```

```
Q.fcall(promisedStep1)
.then(promisedStep2)
.then(promisedStep3)
.then(promisedStep4)
.then(function(value4) {
    // Do something with value4
})
.fail(function(error) {
    // Handle any error from all above
steps
})
.done();
```

It makes the code Async

- Q adds the functions in the chain to the Event loop
- When the promised function returns, the promises are rejected/resolved during the next pass of the runloop

Thank you

Code snippets used for the demo can be downloaded from
https://gist.github.com/senthilkumarv/6326192

Contact at senthilkumar@thoughtworks.com and kuldeepg@thoughtworks.com