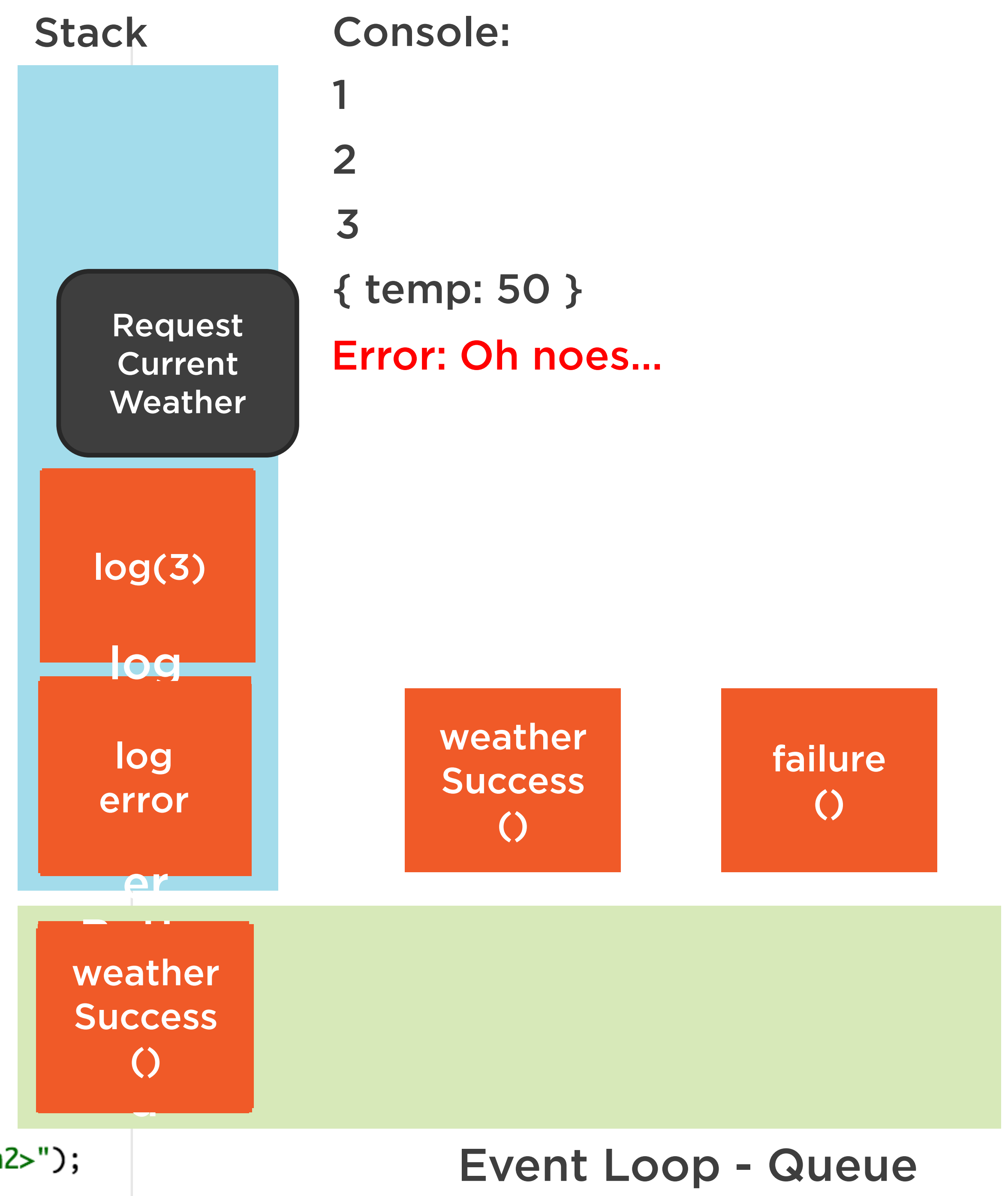
Reasoning Explicitly About When Code Executes

Wes Higbee

@g0t4 www.weshigbee.com

```
function load() {
   console.log("1")
   var weatherRequest = new XMLHttpRequest();
   weatherRequest.onload = weatherSuccess;
   weatherRequest.onerror = failure;
   weatherRequest.open('get', weatherUrl, true);
   weatherRequest.send();

   console.log("2")

   let fiveDayRequest = new XMLHttpRequest();
   fiveDayRequest.onload = fiveDaySuccess;
   fiveDayRequest.onerror = failure;
   fiveDayRequest.open('get', fiveDayUrl, true);
   fiveDayRequest.send();
   console.log("3")

   // code

   function weatherSuccess() {
     let data = JSON.parse(this.responseText);
     console.log("current weather", data);
     showResults(buildCurrentSummary(data));
   }

   function fiveDaySuccess() {
     var data = JSON.parse(this.responseText);
     console.log("five day", data);
     appendResults(buildForecastTable(data));
   }

   function failure(error) {
     showResults("<h2 style='color:red'>Oh no, something bad happened!</h2>");
     console.error(error);
   }
```



Key Takeaways

Single-threaded (bank with one teller)

Event Loop (line to use the bathroom)

Non-blocking (forgot papers, next person serviced)

Avoid blocking (don’t hog the bathroom)

Run to completion (satisfy customer before next)

Cooperative Concurrency (customers play nice)

Little Programs (customers in line)

Think explicitly about Asynchronous Seams

Timer delay not guaranteed (calendar / todo list)

Not always async (quack, waddle but not a duck)

Race conditions in order of queue