

Async Programming in Javascript

thinking about the future

Importance of async programming

- Javascript is Single Threaded (not exactly), Event driven, Non-blocking
 - This means Javascript code is packed with asynchronous code and callbacks!
- Your code will become cleaner - no more callback hell
- Cleaner code means less bugs
- Understanding the async conventions means you can understand better libraries that use async patterns
- Understanding async conventions means you will be able to write code that will be easy to understand and use.

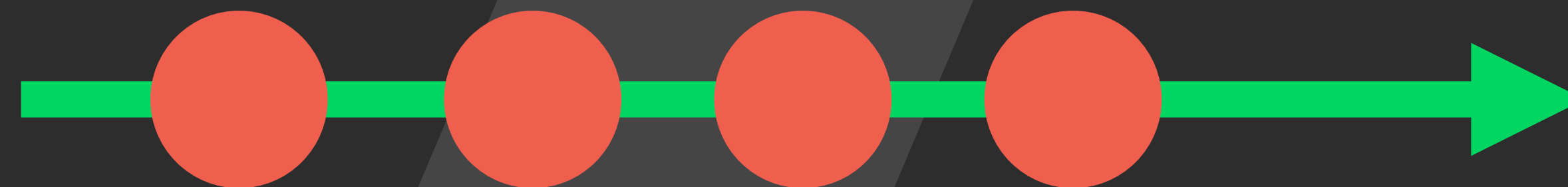
What we will cover in this chapter

- Part I: Defining and classifying asynchronous code
- Part II: Different patterns for different async code
- Part III: Error handling in async code
- Part IV: Freeing memory and memory leaks with async code

Let's improve our Javascript skills!

Part I

Understand and classify your async code



Motivation - why do I need to classify my async code?

- We will look at our async code as a problem we need to solve
 - Sending ajax request, waiting for keyboard event, etc.
- To solve a problem properly we have to define and understand it.
- Only after defining it we will know which tools and patterns we can use to solve our problem

What is Synchronous code?

Synchronous code

- Sync line of code will have to finish before moving to the next line of code
- Every line is blocking - you only get the green light when the line is finished

Synchronous code example

- Here's an example of some sync code lines

```
const todoArray = [  
  'Buy Tofu',  
  'Walk with Piglet',  
  'Clean home'  
];  
  
for (const item of todoArray) {  
  console.log(item);  
}
```

What is Asynchronous code?

Asynchronous code

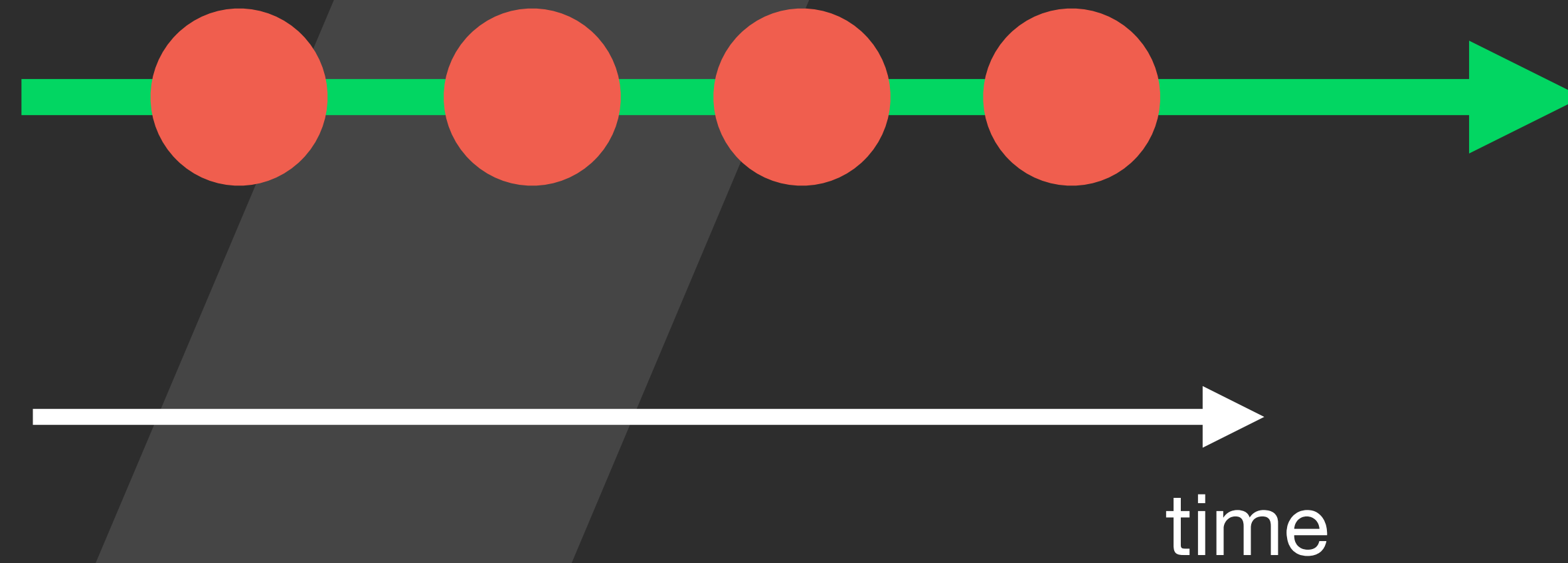
- Code that will run in the future
- We have to consider the element of time
- Non blocking - will run in the future when the stack is empty
- Asynchronous code in Javascript, usually means running a **function** in the future

Asynchronous code example

- Here's an example of async code that will print **hello world** in the future after 1 sec

```
setTimeout(function() {  
    console.log('Hello World');  
}, 1000);  
console.log('This will be printed first');
```

Async code diagram

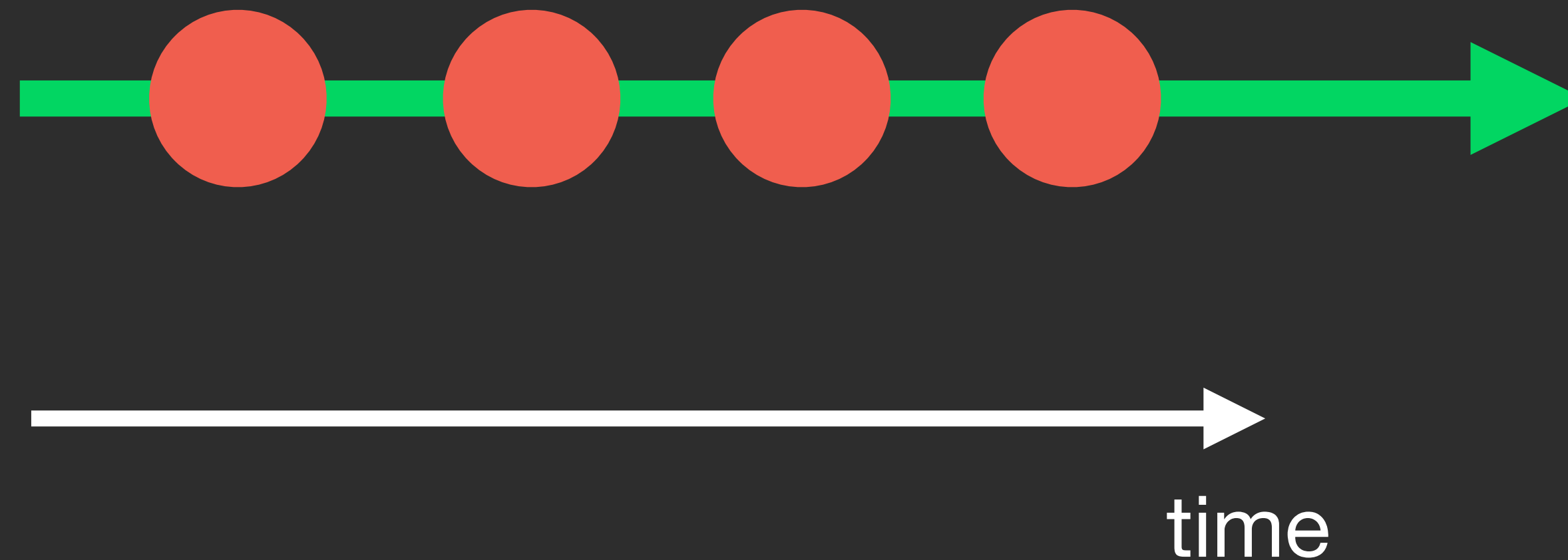


Async code diagram

- We are going to learn patterns to deal with async code properly, easy, and with less bugs
- To understand which pattern to use, we are going to have to categorise and identify our async code
 - Some patterns will be suited to certain types of async code
- We are going to use a diagram to help use identify our async code

How the diagram looks like

- Since async code will run in the future, we have to consider **time**, so time will be represented in the **X axis**
- Our async code is represented by the arrow with circles on it
 - The left side of the arrow is when the async code started to run
 - The circles represent our async code is running



Examples...

setTimeout

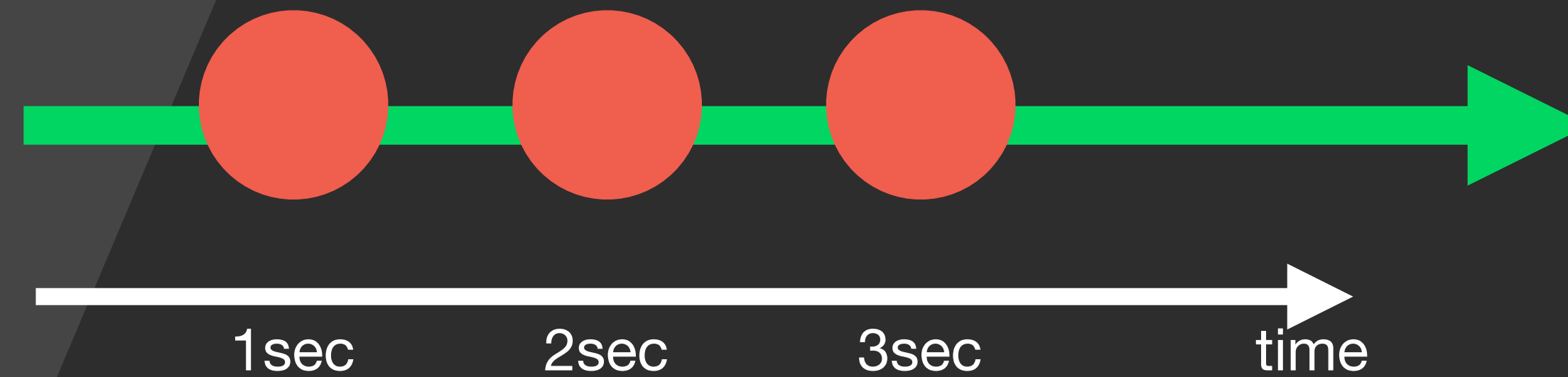
- Async code that runs **once** after a period of time has passed



```
setTimeout(function() {  
  console.log('Hello World');  
}, 1000);
```

setInterval

- Async code that runs **every time** a period of time has passed.



```
setInterval(function() {  
  console.log('Hello World');  
}, 1000);
```

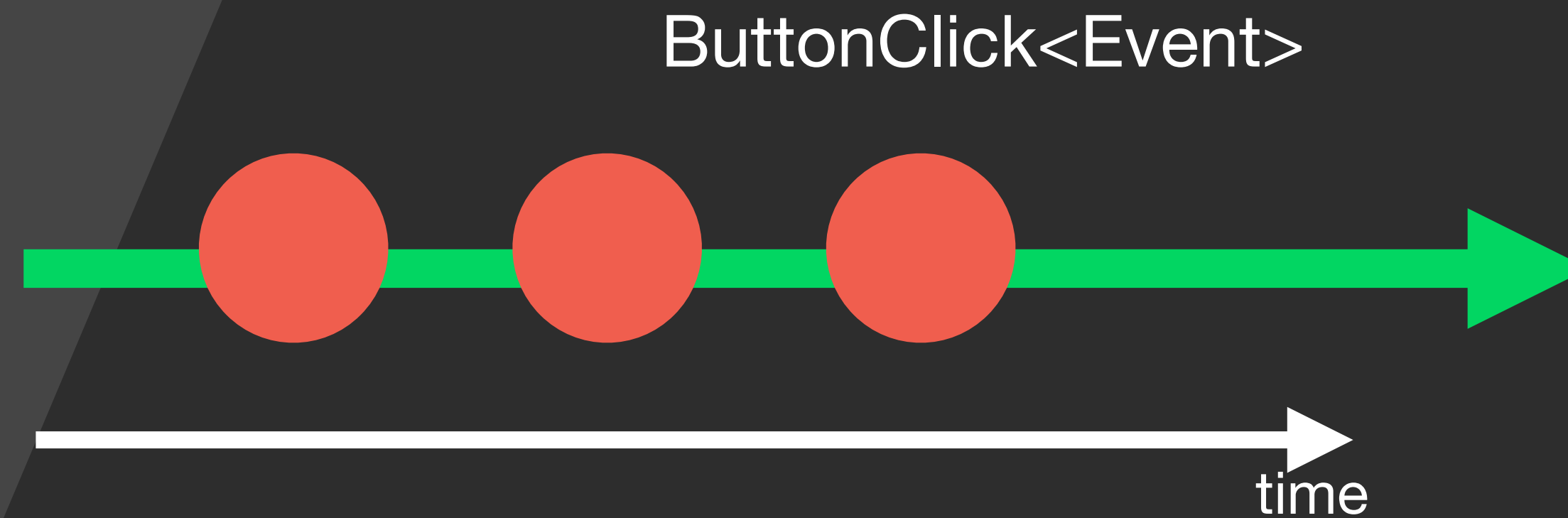
Async code with data

Async code with data

- Some async code will transfer data along
- This means that (for the most part) our async code function will get the data as an argument of the function
- It's common to keep the data consistent
- We can mark the data type in our diagram in these brackets: <string>

Async code with data - Example

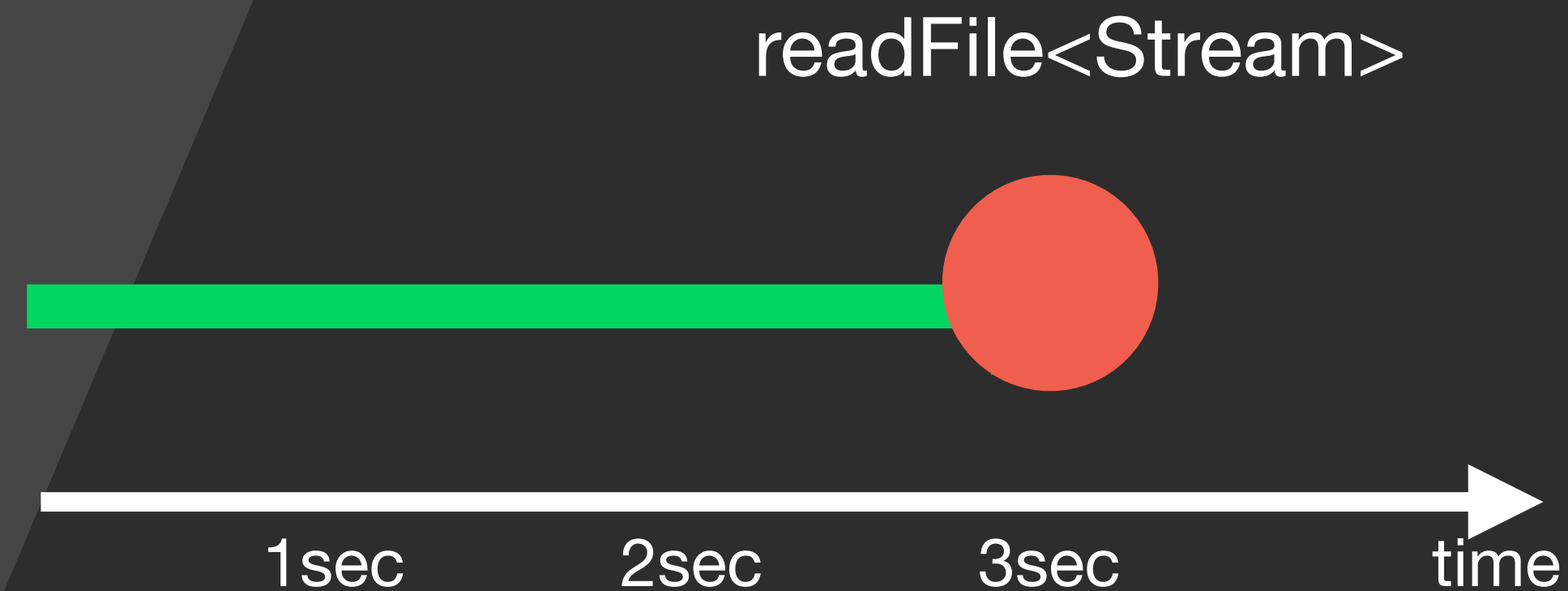
- Browser button click event



```
const button = document.getElementById('myButton');  
  
button.addEventListener('click', function(event) {  
  console.log('we are getting an event object describing the event');  
});
```

Async code with data - Example

- Node.js reading file

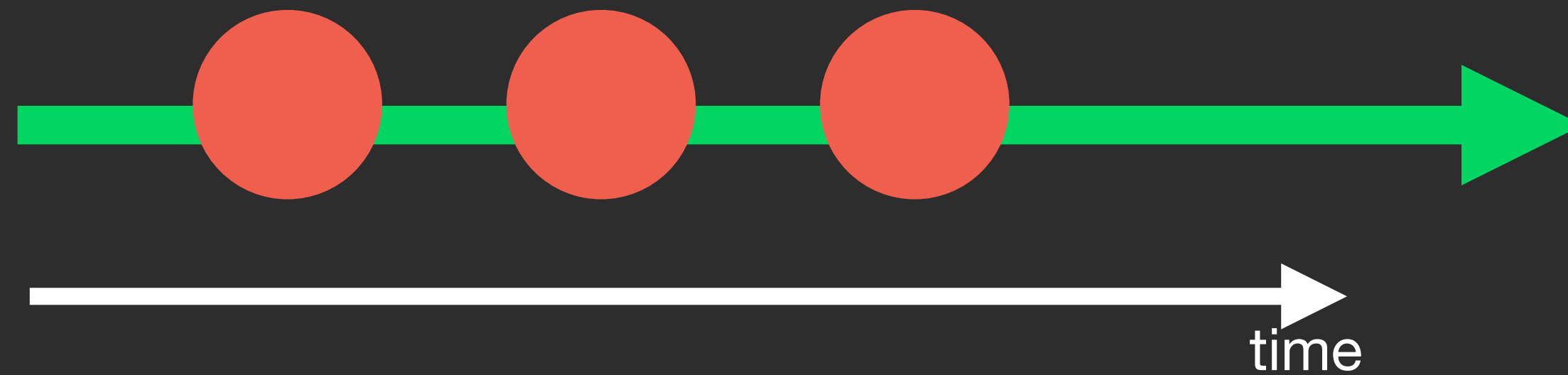


```
const fs = require('fs');

fs.readFile('some-file.txt', function(err, data) {
  console.log(`The content of the file is: ${data.toString()}`);
});
```

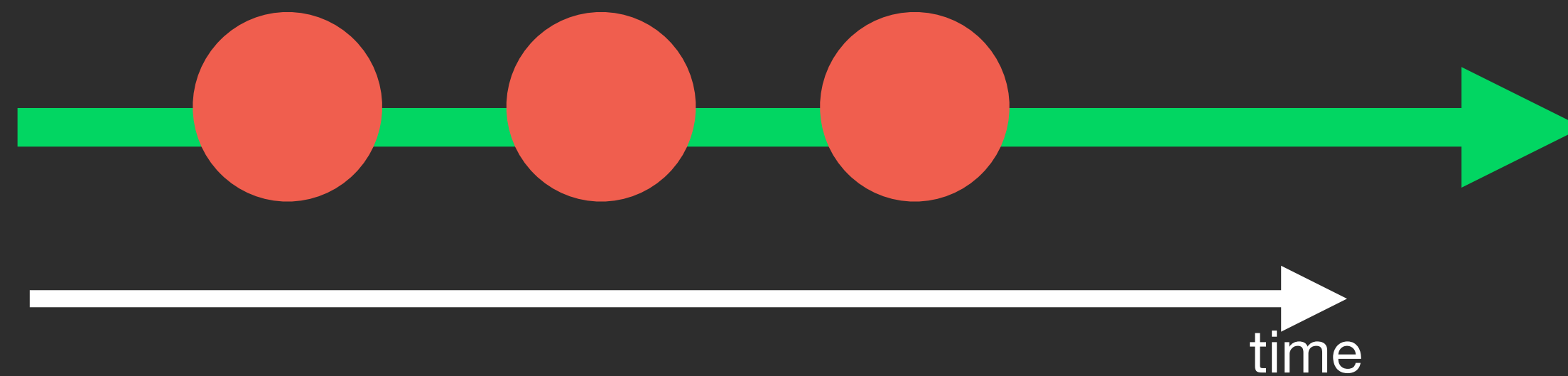
Important questions

- Few important questions you have to ask yourself about your async code
 - Will my async code end?
 - How many pulses do I have?



Summary

- Async code will usually call a callback function sometime in the future
- Sometimes that callback function will get data in the arguments
- We can define our async code in a diagram, where we have a line in the timeline with pulse every time the callback is called



Thank you

Continue to part II on async patterns

Part II

Patterns for dealing with async code