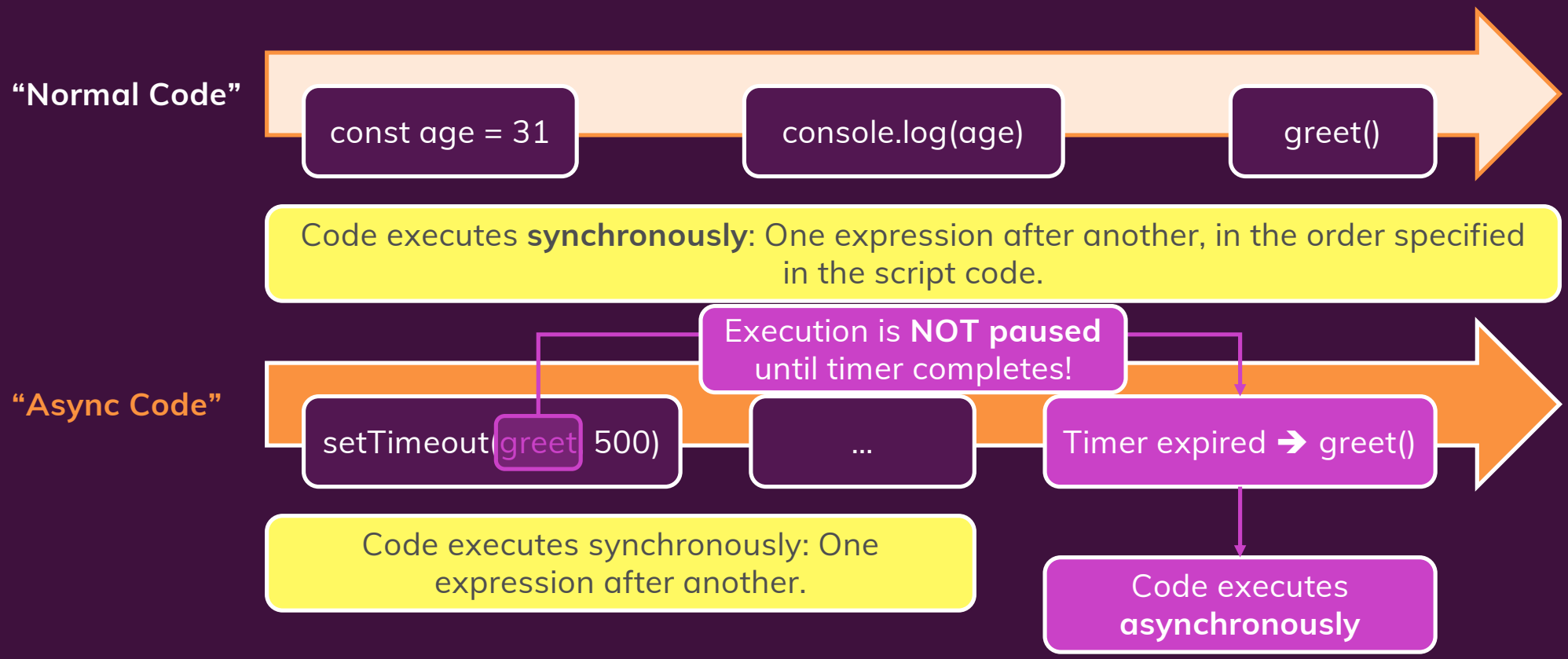
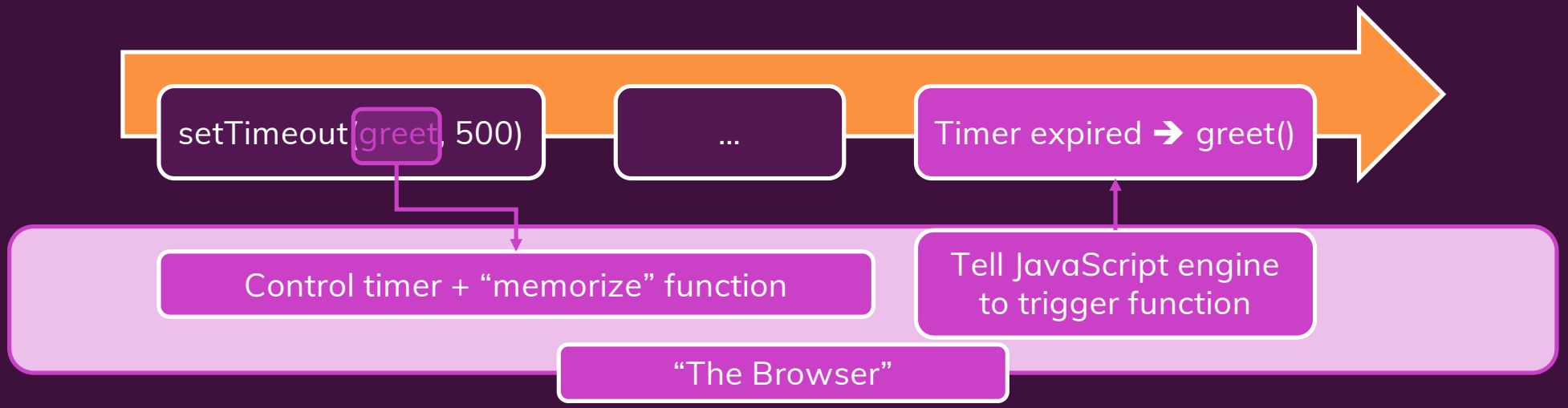
ACADEMIND

Understanding Asynchronous Code



But JavaScript is Single-Threaded!



Only one action at a time can be performed. JavaScript **can’t** keep a timer running and do something else at the same time.

But the browser is able to do that!

ACADEMIND

Dealing with Asynchronous Operations

Callback Functions	Define which function should eventually be executed. *Examples: setTimeout(), addEventListener()*
Promises	Define a step-by-step “chain” for async operations. *Example: fetch()*
async/ await	Syntactic sugar for Promises to simplify using them *Example: fetch()*

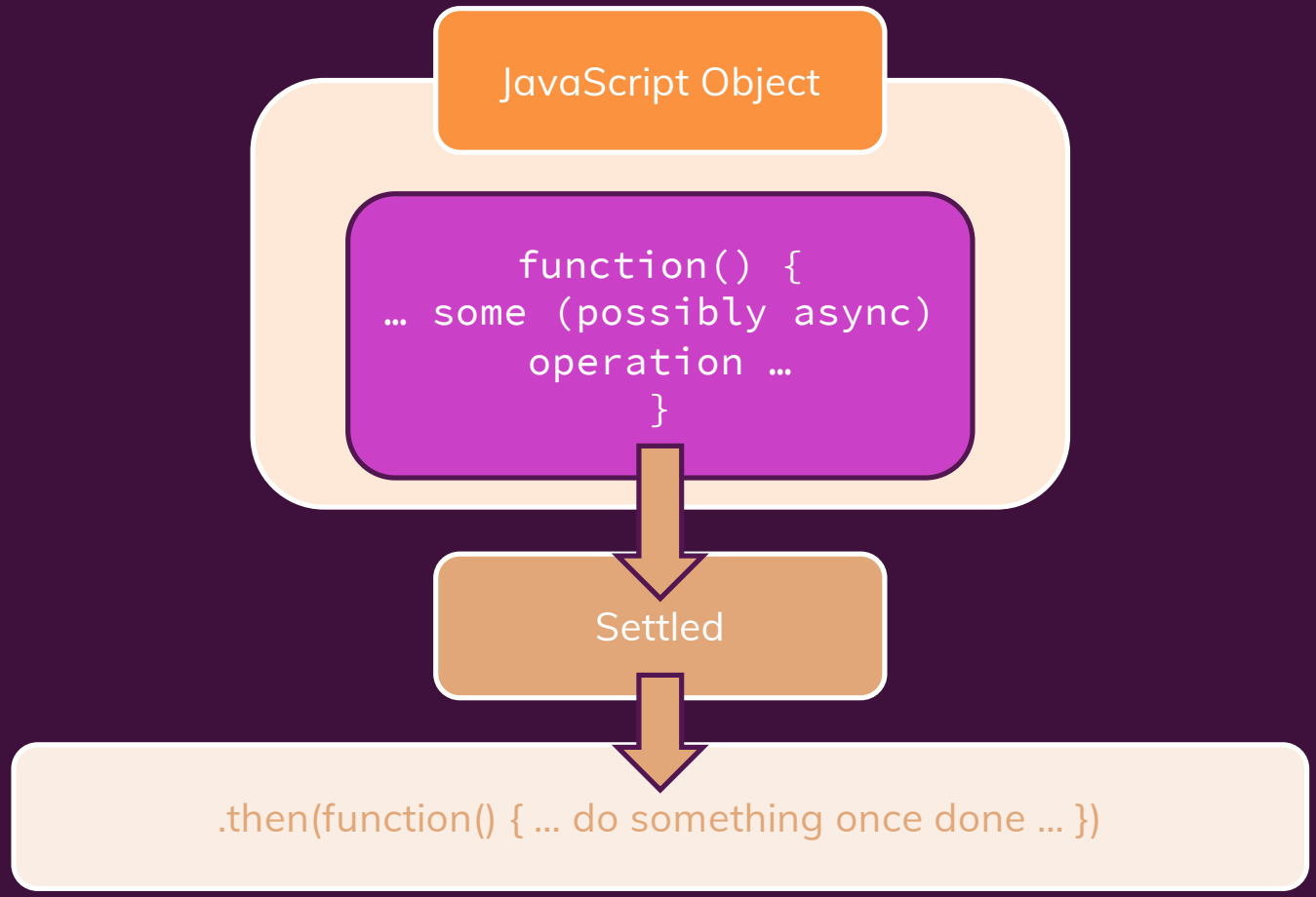
ACADEMIND

Callback Hell

```
el.addEventListener('click', function() {
  setTimeout(function() {
    doSomethingElse(function() {
      …
    });
  }, 2000);
})
```

```
addListener(el, 'click')
  .then(function() {
    return setTimer(2000);
  })
  .then(function() {
    return doSomethingElse();
  })
  .then(…);
```

What's a Promise?



ACADEMIND

Promise States

Pending		Initial state, code in promise has not produced a result yet
Fulfilled	Settled	Promise has been resolved (i.e. the operation completed successfully)
Rejected		Promise has been rejected (i.e. the operation failed)

ACADEMIND

Summary – Async Code

Asynchronous code is code that "**doesn't finish immediately**" (e.g. a timer, a http request, ...). JavaScript handles such code by **NOT pausing execution** but instead by **handing the task off** to the engine environment (e.g. to the browser).

JavaScript **is single-threaded**, hence it can only **do one thing at a time** – therefore it **can't manage** a timer and continue executing other code.

Hence async operations are **handed off to the environment** (e.g. browser) which is not single-threaded.

Async operations can be handled **with two key features** (which are also available for non-async tasks): **Callback functions** and **Promises**.

For **built-in APIs** (e.g. `setTimeout()`, `fetch()`), you **can't choose which mechanism** you want to use.

Instead of `then()` / `catch()`, you can use **`async/await`** (with `try-catch`) to work with **Promises**.