THE COMPLETE JAVASCRIPT COURSE

FROM ZERO TO EXPERT!

SECTION

ASYNCHRONOUS JAVASCRIPT: PROMISES, ASYNC/AWAIT AND AJAX

LECTURE

PROMISES AND THE FETCH API

JS

WHAT ARE PROMISES?

PROMISE

- **Promise:** An object that is used as a placeholder for the future result of an asynchronous operation.

  ↓ Less formal

- **Promise:** A container for an asynchronously delivered value.

  ↓ Less formal

- **Promise:** A container for a future value.

  Example: Response from AJAX call

- We no longer need to rely on events and callbacks passed into asynchronous functions to handle asynchronous results;
- Instead of nesting callbacks, we can **chain promises** for a sequence of asynchronous operations: **escaping callback hell** 🎉

🎟 **Promise** that I will receive money if I guess correct outcome

🤞 I buy lottery ticket (promise) right now

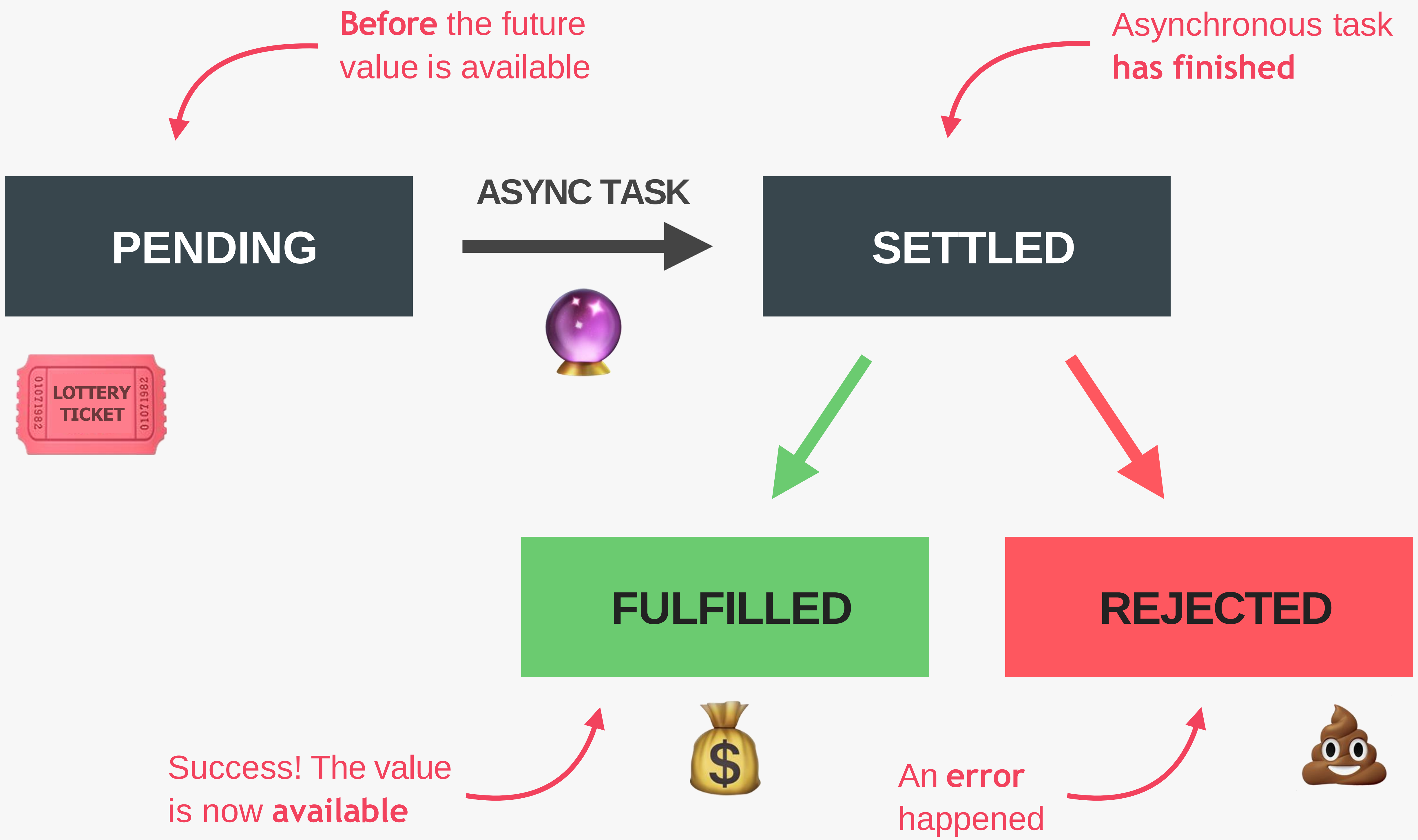
↓

🔮 Lottery draw happens asynchronously

↓

💰 If correct outcome, I receive money, because it was promised

THE PROMISE LIFECYCLE



Before the future value is available

PENDING

ASYNC TASK

SETTLED

Asynchronous task **has finished**

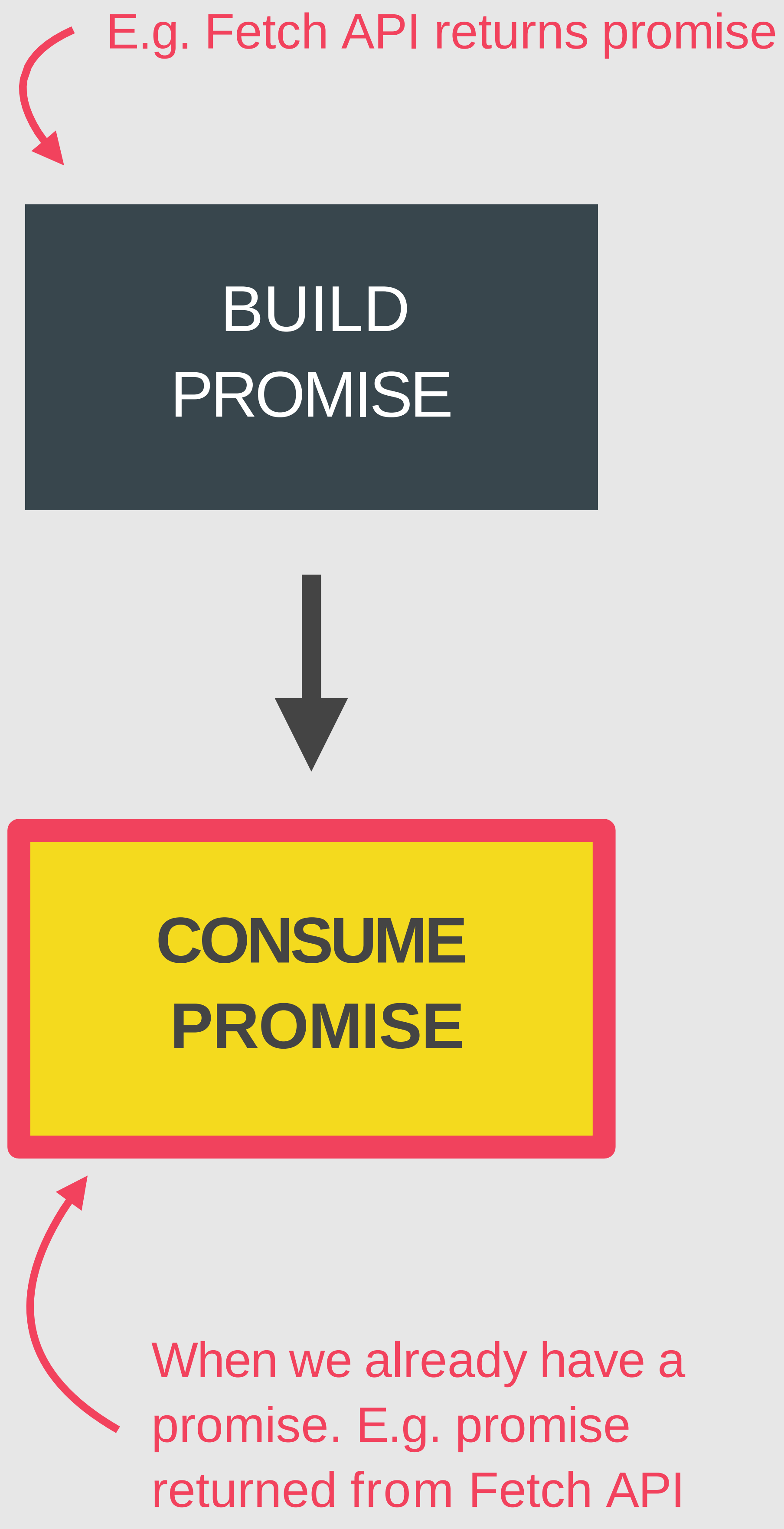
FULFILLED

REJECTED

Success! The value is now **available**

An **error** happened

👉 We are able **handle** these different states in our code!



E.g. Fetch API returns promise

BUILD PROMISE

CONSUME PROMISE

When we already have a promise. E.g. promise returned from Fetch API

THE COMPLETE JAVASCRIPT COURSE

FROM ZERO TO EXPERT!

SECTION

ASYNCHRONOUS JAVASCRIPT: PROMISES, ASYNC/AWAIT AND AJAX

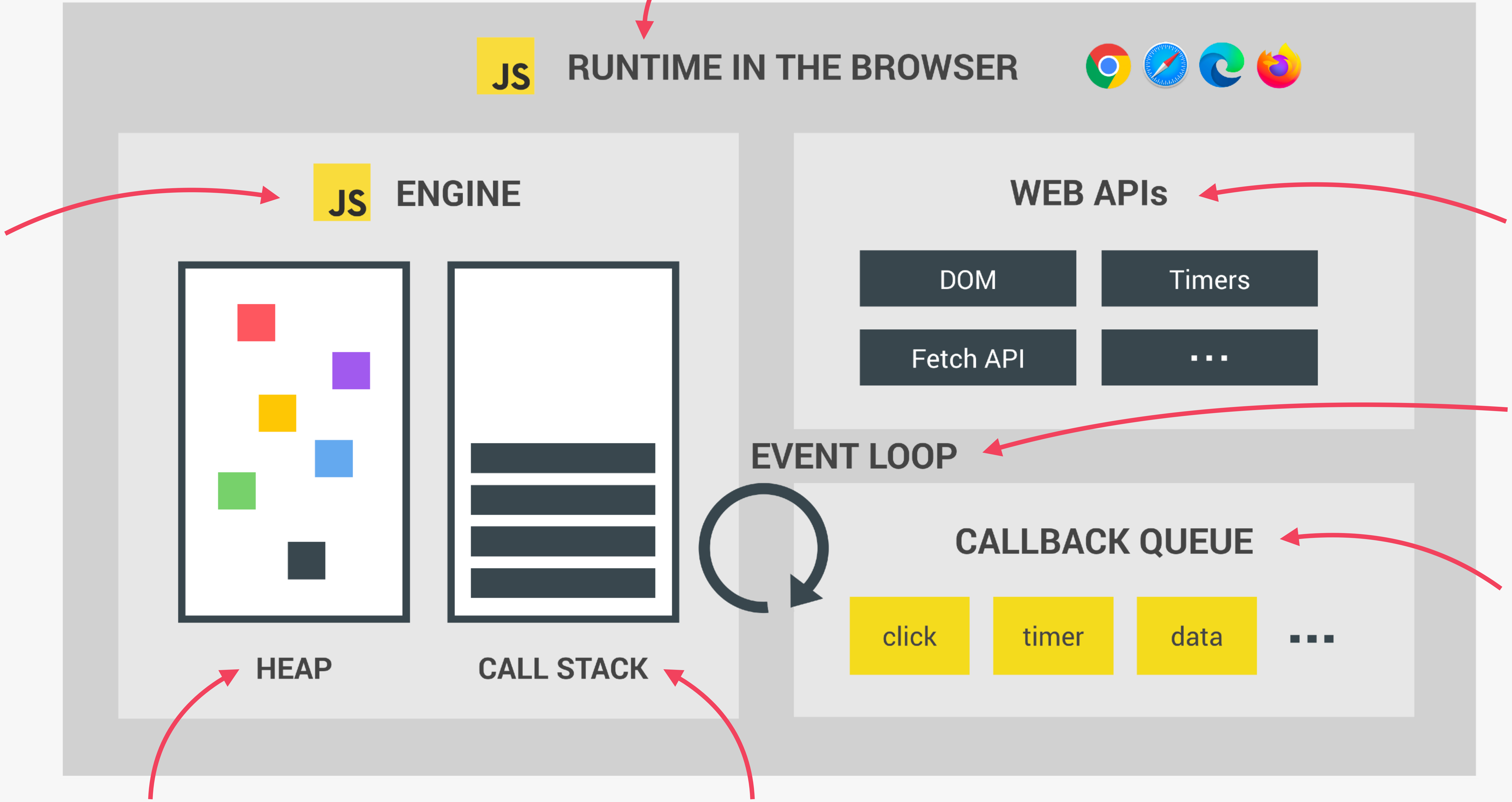
LECTURE

ASYNCHRONOUS BEHIND THE SCENES: THE EVENT LOOP

JS

REVIEW: JAVASCRIPT RUNTIME

👉 **Concurrency model**: How JavaScript handles multiple tasks happening at the same time.



"Container" which includes all the pieces necessary to execute JavaScript code

"Heart" of the runtime

APIs provided to the engine

Sends callbacks from queue to call stack

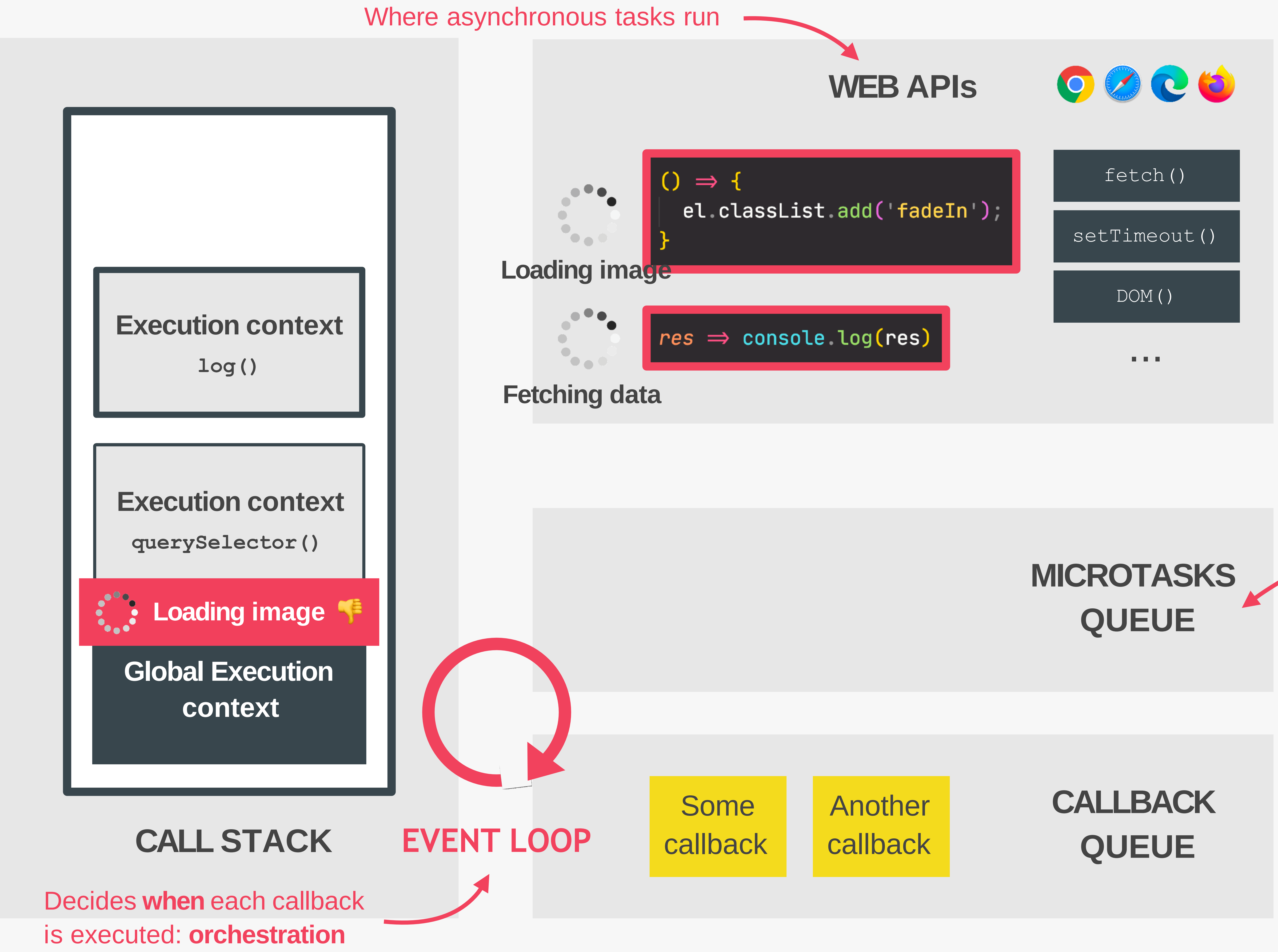
Ready-to-be-executed callback functions (coming from events)

Where object are stored in memory

Where code is actually executed

Only ONE thread of execution. No multitasking!

HOW ASYNCHRONOUS JAVASCRIPT WORKS BEHIND THE SCENES



Where asynchronous tasks run

WEB APIs

```
() => {
  el.classList.add('fadeIn');
}
```

Loading image

```
res => console.log(res)
```

Fetching data

fetch()

setTimeout()

DOM()

...

Execution context
log()

Execution context
querySelector()

Loading image 👎

Global Execution context

CALL STACK

EVENT LOOP

Decides **when** each callback is executed: **orchestration**

MICROTASKS QUEUE

Some callback

Another callback

CALLBACK QUEUE

```
el = document.querySelector('img');
el.src = 'dog.jpg';
el.addEventListener('load', () => {
  el.classList.add('fadeIn');
});

fetch('https://someurl.com/api')
  .then(res => console.log(res));

// More code...
```

Like callback queue, but for callbacks related to **promises**. Has **priority** over callback queue!

🤔 How can **asynchronous** code be executed in a **non-blocking way**, if there is **only one thread** of execution in the engine?

MODERN JAVASCRIPT DEVELOPMENT: MODULES AND TOOLING

THE COMPLETE JAVASCRIPT COURSE

FROM ZERO TO EXPERT!

SECTION

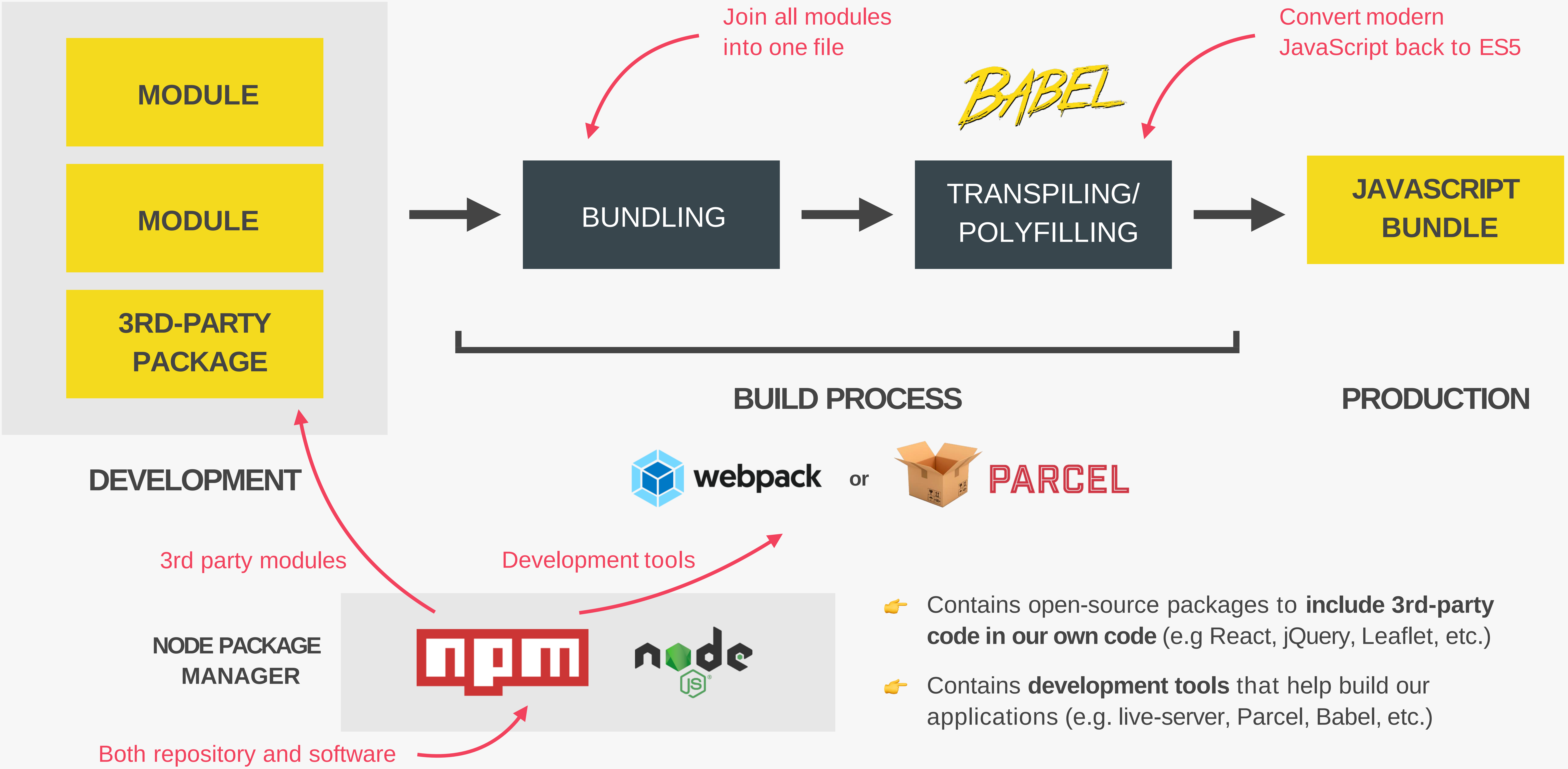
MODERN JAVASCRIPT DEVELOPMENT: MODULES AND TOOLING

LECTURE

AN OVERVIEW OF MODERN JAVASCRIPT DEVELOPMENT

JS

MODERN JAVASCRIPT DEVELOPMENT



MODULE

MODULE

3RD-PARTY PACKAGE

DEVELOPMENT

Join all modules into one file

BUNDLING

Convert modern JavaScript back to ES5

TRANSPILING/ POLYFILLING

JAVASCRIPT BUNDLE

BUILD PROCESS

PRODUCTION

3rd party modules

Development tools

NODE PACKAGE MANAGER

Both repository and software

- Contains open-source packages to **include 3rd-party code in our own code** (e.g React, jQuery, Leaflet, etc.)
- Contains **development tools** that help build our applications (e.g. live-server, Parcel, Babel, etc.)