JAVASCRIPT

OBJECT ORIENTED PROGRAMMING (OOP) WITH JAVASCRIPT

WHAT IS OBJECT-ORIENTED PROGRAMMING?

Object-oriented programming (OOP) is a programming paradigm based on the concept of objects

We use objects to model (describe) real-world or abstract features

Objects may contain data (properties) and code (methods). By using objects, we pack data and the corresponding behavior into one block

In OOP, objects are self-contained pieces/blocks of code

Objects are building blocks of applications, and interact with one another

Interactions happen through a public interface (API): methods that the code outside of the object can access and use to communicate with the object

OOP was developed with the goal of organizing code, to make it more flexible and easier to maintain

THE 4 FUNDAMENTAL OOP PRINCIPLES

ABSTRACTION

Ignoring or hiding details that don't matter, allowing us to get an overview perspective of the thing we're implementing, instead of messing with details that don't really matter to our implementation

ENCAPSULATION

Keeping properties and methods private inside the class, so they are not accessible from outside the class. Some methods can be exposed as a public interface (API)

INHERITANCE

Making all properties and methods of a certain class available to a child class, forming a hierarchical relationship between classes. This allows us to reuse common logic and to model real-world relationships

POLYMORPHISM

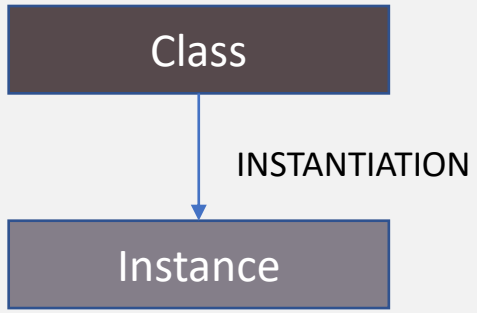
A child class can overwrite a method it inherited from a parent class

PROTOTYPES

OOP IN JAVASCRIPT

OOP IN JAVASCRIPT: PROTOTYPES

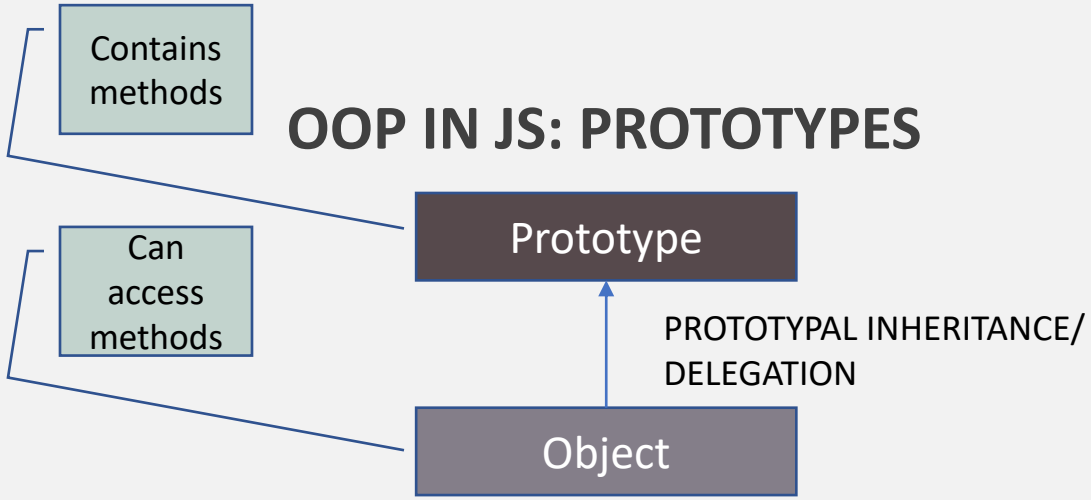
"CLASSICAL OOP": CLASSES



Objects (instances) are instantiated from a class, which functions like a blueprint

Behavior (methods) is copied from class to all instances.

OOP IN JS: PROTOTYPES



Objects are linked to a prototype object;

The prototype contains methods (behavior) that are accessible to all objects linked to that prototype;

Behavior is delegated to the linked prototype object.

3 WAYS OF IMPLEMENTING PROTOTYPAL INHERITANCE IN JAVASCRIPT

1 CONSTRUCTOR FUNCTIONS

- Technique to create objects from a function;
- This is how built-in objects like Arrays, Maps or Sets are actually implemented.

2 ES6 CLASSES

- Modern alternative to constructor function syntax;
- “Syntactic sugar”: behind the scenes, ES6 classes work exactly like constructor functions;
- ES6 classes do NOT behave like classes in “classical OOP”.

3 OBJECT.CREATE()

- The easiest and most straightforward way of linking an object to a prototype object.

JAVASCRIPT NEW FEATURES

ES6 AND MORE

ES6 & MORE – NEW FEATURES

ARROW FUNCTION	DESTRUCTURING	REST / SPREAD
TEMPLATE LITERALS	BLOCK SCOPING	MAP/SET
CLASSES	DEFAULT PARAMETERS	ITERATORS & GENERATORS

ARROW FUNCTION =>

Arrow functions are handy for one-liner functions

Arrow Function Flavors

Without Curly braces
(…args) => expression

With curly braces
(…args) => { body

Limitations

Don’t have this keyword

Don’t have arguments keyword

Cant call with new operator

BLOCK SCOPING

Restricts the scope of variables to the nearest curly braces { }

Variables Types

const: converts the variable to a constant

let: for all type of variables

const !== immutable

REST / SPREAD OPERATOR

REST	A function can be called with any number of arguments, no matter how it is defined.
	The rest parameters must be at the end.
	Usage: create functions that accept any number of arguments
SPREAD	Spread operator looks similar to rest parameters, also using (...), but does quite the opposite.
	It is used in the function call, it “expands” an iterable object into the list of arguments.
	Usage: pass an array to functions that normally require a list of many

DESTRUCTURING

Destructuring assignment is a special syntax that allows us to “unpack” arrays or objects into a bunch of variables.

OBJECT DESTRUCTURING

- We have an existing object at the right side, that we want to split into variables.

ARRAY DESTRUCTURING

- The array is destructured into variables, but the array itself is not modified.

NESTED DESTRUCTURING

- If an object or an array contain other objects and arrays, we can use more complex left side patterns to extract deeper portions.

COMPLEX DATA STRUCTURES : MAP

Map is a collection of keyed data items, just like an Object.

METHODS AND PROPERTIES	DESCRIPTION
new Map()	Creates a Map
map.set (key, value)	Stores the value by the key
map.get(key)	Returns the value by the key, undefined if key doesn’t exist in map
map.has(key)	Returns true if the key exists, false otherwise
map.delete(key)	Removes the value by the key
map.clear()	Removes everything from the map
map.size	Returns the current element count

COMPLEX DATA STRUCTURES : SET

A Set is a special type collection – “set of values” (without keys), where each value may occur only once.

METHODS AND PROPERTIES	DESCRIPTION
new Set(iterable)	Creates the set, and if an iterable object is provided (usually an array), copies values from it into the se
set.add(value)	Adds a value, returns the set itself
set.delete(value)	Removes the value, returns true if value existed at the moment of the call, otherwise false
set.has(value)	Returns true if the value exists in the set, otherwise false
set.clear()	Removes everything from the set
map.size	Returns the current element count

WEAKMAP AND WEAKSET

WeakMap is Map-like collection that allows only objects as keys and removes them together with associated value once they become inaccessible by other means.

WeakSet is Set-like collection that stores only objects and removes them once they become inaccessible by other means.

Not having support for clear, size, keys, values etc

WeakMap and WeakSet are used as “secondary” data structures in addition to the “primary” object storage. Once the object is removed from the primary storage, if it is only found as the key of WeakMap or in a WeakSet, it will be cleaned up automatically.

Tagged Template Literals

A syntax that makes string interpolation possible in JavaScript instead of ugly string concatenation

Tagged template literals offers you the opportunity to parse template literals by combining functions with template literals

There are two parts of a tagged template literal, the first one being the *tag function* and the second, the *template literal*

ITERATORS AND GENERATORS

Iterator

- In JavaScript an iterator is an object which defines a sequence and potentially a return value upon its termination.
- An iterator object can be iterated explicitly by repeatedly calling *next()*
- An iterator is any object which implements the *Iterator protocol* by having a next() method that returns an object with two properties: *value & done*

ITERATORS AND GENERATORS

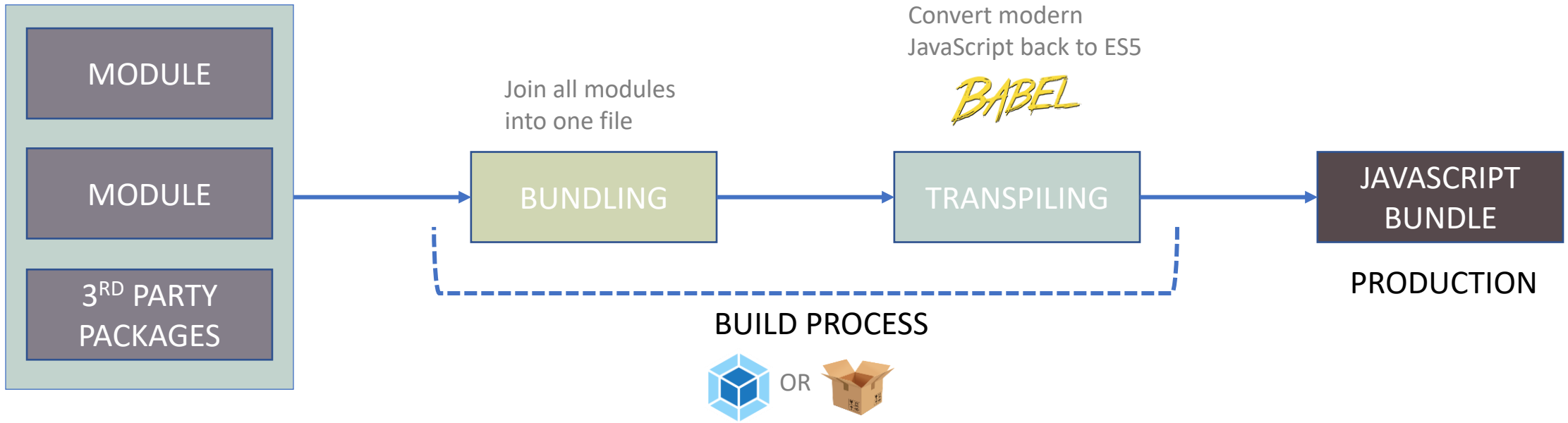
Generators

- Generator allow you to define an iterative algorithm by writing a single function whose execution is not continuous
- Generator functions are written using the *function** syntax
- Generators return a special type of iterator, called a Generator. When a value is consumed by calling the generator's *next()* method, the Generator function executes until it encounters the *yield* keyword

MODULES AND TOOLING

AN OVERVIEW OF MODERN JAVASCRIPT DEVELOPMENT

MODERN JAVASCRIPT DEVELOPMENT



NODE PACKAGE MANAGER

- 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.)

AN OVERVIEW OF MODULES

Reusable piece of code that encapsulates implementation details

Usually a standalone file, but it doesn't have to be

Why Modules?

Compose software: Modules are small building blocks that we put together to build complex applications;

Isolate components: Modules can be developed in isolation without thinking about the entire codebase;

Abstract code: Implement low-level code in modules and import these abstractions into other modules;

Organized code: Modules naturally lead to a more organized codebase;

Reuse code: Modules allow us to easily reuse the same code, even across multiple projects.

ASYNCHRONOUS JAVASCRIPT

PROMISES, ASYNC/ AWAIT AND MORE

SYNCHRONOUS CODE

```
const p = document.querySelector("#paragraph");
p.textContent = "Hello World";
alert("Who's this?");
p.style.color = "red";
```

Most code is synchronous

Synchronous code is executed line by line

Each line of code waits for previous line to finish

Long-running operations block code execution

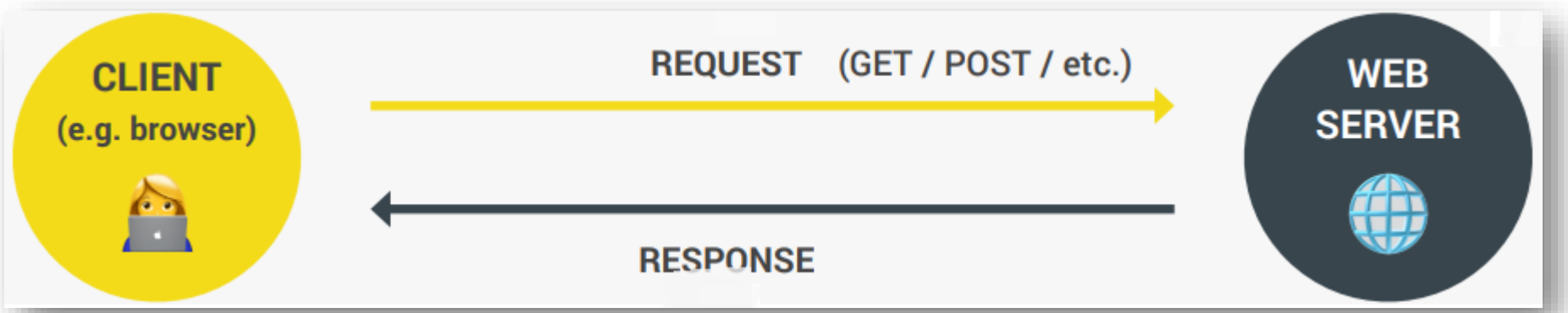
ASYNCHRONOUS CODE

```
const p = document.querySelector("#paragraph");
setTimeout(() => {
    alert("Who's this?")
}, 1000);
p.style.color = "red";
```

- Asynchronous code is executed after a task that runs in the “background” finishes
- Asynchronous code is non-blocking
- Execution doesn’t wait for an asynchronous task to finish its work
- Callback functions alone do NOT make code asynchronous

WHAT ARE AJAX CALLS?

Asynchronous JavaScript And XML: Allows us to communicate with remote web servers in an asynchronous way. With AJAX calls, we can request data from web servers dynamically



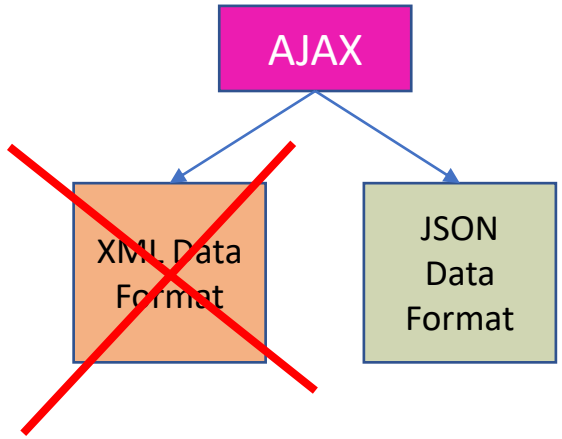
WHAT IS AN API?

Piece of software that can be used by another piece of software, in order to allow applications to talk to each other

There are be many types of APIs in web development: DOM API, Geolocation API, Books API, Online API etc

“Online” API: Application running on a server, that receives requests for data, and sends data back as response

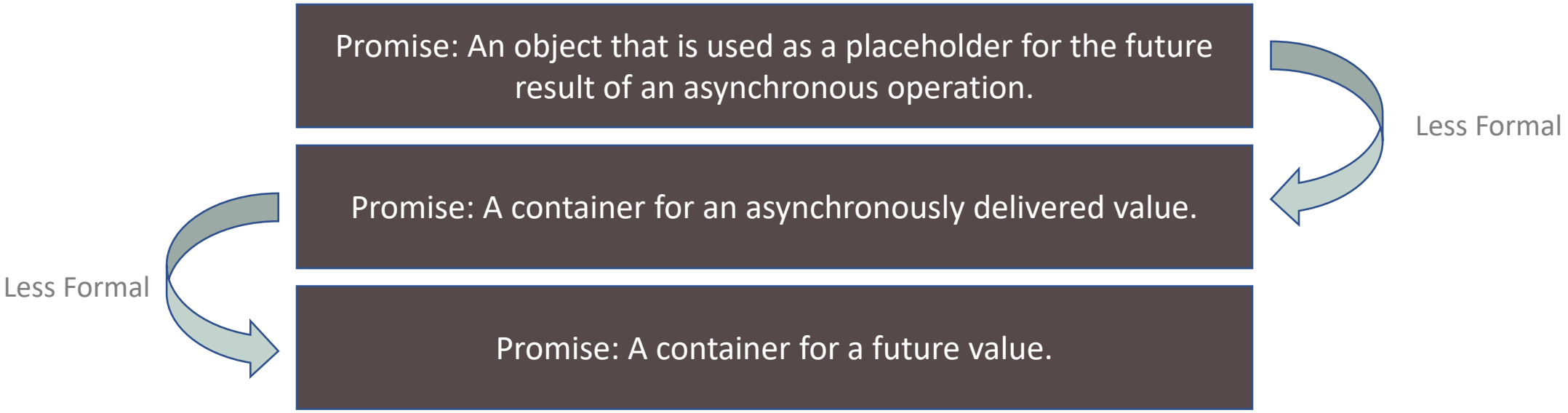
We can build our own web APIs (requires back-end development, e.g. with node.js) or use 3rd-party APIs



Examples of 3rd Party APIs

- Weather data
- Data about countries
- Flights data
- Currency conversion data
- APIs for sending email or SMS
- Google Maps
- Millions of possibilities.

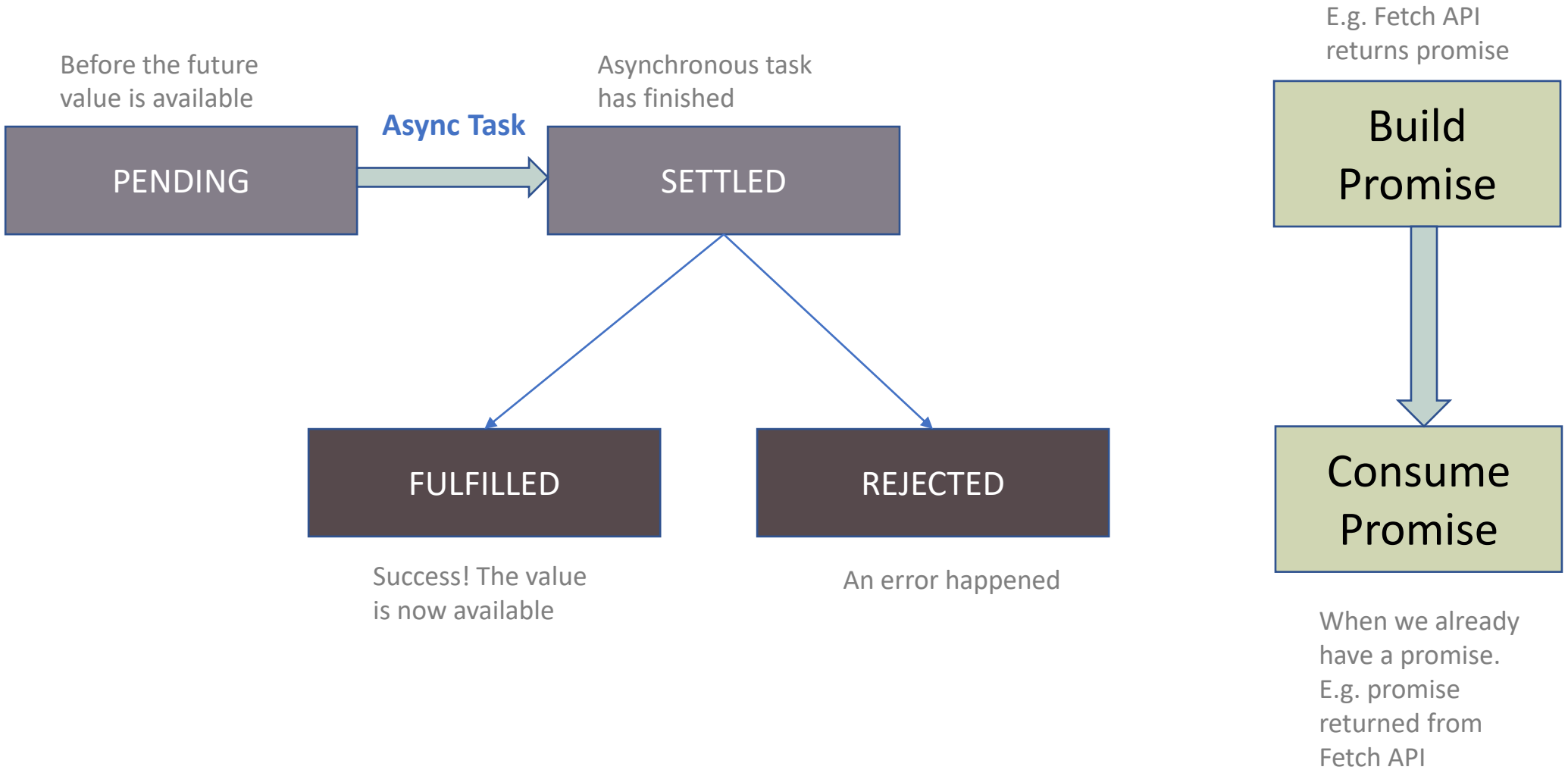
WHAT ARE PROMISES?



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

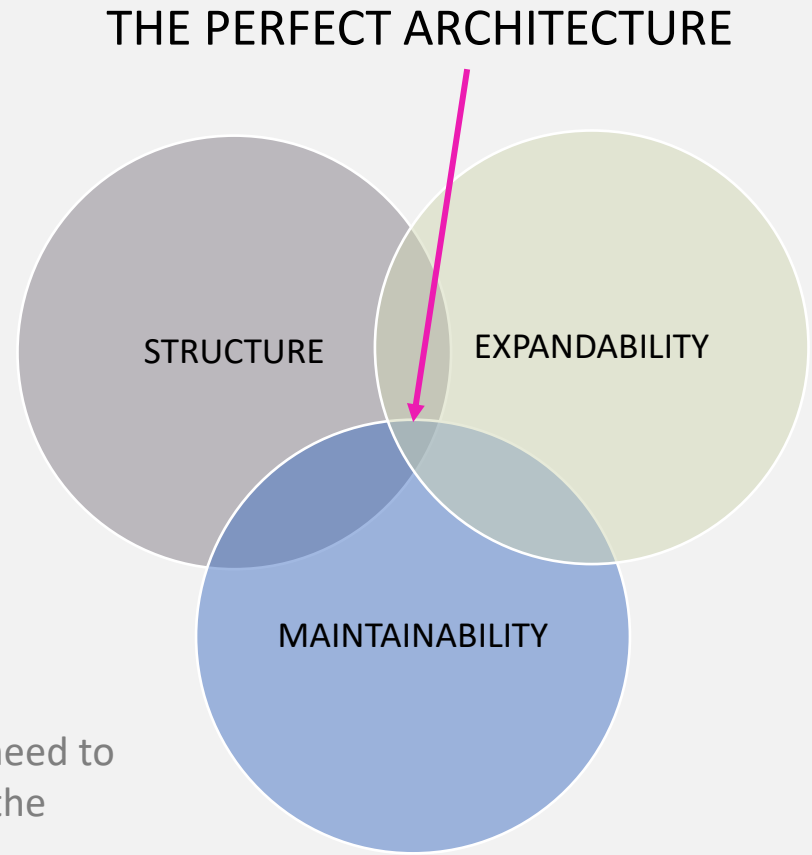
THE PROMISE LIFECYCLE



THE MVC ARCHITECTURE

WHY WORRY ABOUT ARCHITECTURE?

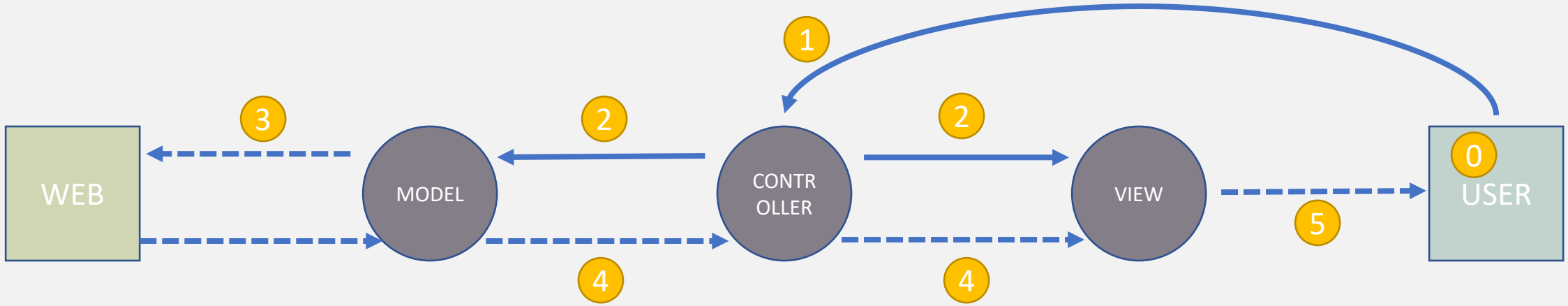
Like a house, software needs a structure: the way we organize our code



We also need to be able to easily add new features

A project is never done! We need to be able to easily change it in the future

THE MODEL-VIEW-CONTROLLER (MVC) ARCHITECTURE



BUSINESS LOGIC

STATE

HTTP LIBRARY

APPLICATION LOGIC

Bridge between model and views (which don't know about one another)

Handles UI events and dispatches tasks to model and view

PRESENTATION LOGIC

Connected by function call and import

Data Flow

MODERN JAVASCRIPT DEVELOPMENT

WRITING CLEAN AND MODERN JAVASCRIPT

REVIEW: MODERN AND CLEAN CODE

READABLE CODE

- Write code so that others can understand it
- Write code so that you can understand it in 1 year
- Avoid too “clever” and overcomplicated solutions
- Use descriptive variable names: what they contain
- Use descriptive function names: what they do

FUNCTIONS

- Generally, functions should do only one thing
- Don’t use more than 3 function parameters
- Use default parameters whenever possible
- Generally, return same data type as received
- Use arrow functions when they make code more readable

REVIEW: MODERN AND CLEAN CODE

GENERAL

- Use DRY principle (refactor your code)
- Don’t pollute global namespace, encapsulate instead
- Don’t use var
- Use strong type checks (=== and !==)

OOP

- Use ES6 classes
- Encapsulate data and don’t mutate it from outside the class
- Implement method chaining
- Do not use arrow functions as methods (in regular objects)

REVIEW: MODERN AND CLEAN CODE

AVOID NESTED CODE

- Use early return (guard clauses)
- Use ternary (conditional) or logical operators instead of if
- Use multiple if instead of if/else-if
- Avoid for loops, use array methods instead
- Avoid callback-based asynchronous APIs

ASYNCHRONOUS CODE

- Consume promises with async/await for best readability
- Whenever possible, run promises in parallel (Promise.all)
- Handle errors and promise rejections

REFERENCES

READING MATERIAL

- https://javascript.info/
- https://developer.mozilla.org/en-US/docs/Web/JavaScript

VIDEO LINKS

- https://www.youtube.com/watch?v=NCwa_xi0Uuc
- https://www.youtube.com/watch?v=nZ1DMMsyVyI