Become the best version of yourself

ECMAScript 2015+

Understanding the Essentials

What is a programming language?

- A programming language is a formal language comprising a set of instructions that produce various kinds of output.
- These languages are used in computer programming to implement algorithms and manipulate data structures.
- Essentially, they are the tools that developers use to build software applications, scripts, or other sets of instructions for computers to execute.

What a programming language provides?

- **Set of Keywords**: These are reserved words with special meanings defined by the language, such as if, for, while, class, and return. These keywords cannot be used as identifiers (like variable names) because they are used to construct the programming syntax.
- **Loops and Statements**: Control structures like loops (for, while) and statements (if, else, switch) control the flow of execution of the program. They allow for the repetition of certain operations and making decisions.
- **Data Types**: These define the nature of data that can be processed, such as integers, floating points, booleans, and strings. Data types help in understanding the kind of operations that can be performed on a particular piece of data.
- **Syntax and Semantic Rules**: Syntax refers to the structure of the language and the rules about how elements of the language can be combined. Semantics deals with the meanings of these syntactical elements—what the instructions mean when executed.

Installation

- Install Node Latest LTS version (v18) - https://nodejs.org/en/
- Install / Update Visual Studio Code to Latest Release - https://code.visualstudio.com/
- Extensions of Visual Studio Code
  - AutoFileName
  - vscode-icons
  - JavaScript (ES6) code snippets
- File Menu -> Preferences -> Theme -> File Icon Theme -> Select VScode Icons

Babel

- Babel is a JavaScript compiler.
- Babel is a toolchain that is mainly used to convert ECMAScript 2015+ code into a backward compatible version of JavaScript in current and older browsers or environments.
- Here are the main things Babel can do for you:
  - Transform syntax
  - Polyfill features that are missing in your target environment (through a third-party polyfill such as core-js)
  - Source code transformations (codemods)

Commands

- Create a new Folder and open it in terminal
- npm init -y
- npm i -D @babel/core @babel/preset-env @babel/cli
- npm i core-js
- Created babel.config.json file
- To Compile
  - npx babel --version
  - npx babel src --out-dir dist

Steps to run the application

- Download and Extract the zip file / clone it from Gitlab Repo.
- Open the folder in Visual Studio Code
- Open the terminal window on the folder path and run
  - npm install
- To Compile
  - npm run build
- Once the installation completes, run
  - node dist/main.js
  - The output will not run on the browser because it needs module loader or module bundler

What is Application Build?

- In the context of software development, "build" refers to the process of converting source code into an executable or deployable form.
- The build process is an essential part of the software development lifecycle, and it typically occurs after the development phase when the source code is ready for deployment or testing.
- The primary goal of the build process is to transform the source code into a usable form that can be executed or deployed in a target environment.
- It involves various tasks such as compiling code, bundling assets, optimizing resources, running tests, and generating artifacts that can be used for deployment or distribution.

How will you Build - Build Workflow

- A build workflow, also known as a build process or build pipeline, refers to the series of steps and processes involved in transforming source code into a deployable and optimized version of a software project.
- It encompasses tasks such as compiling, bundling, testing, optimizing, and preparing the code for production or deployment.
- Creating build workflows involves defining the sequence of tasks and processes required to build, test, and optimize your project.
- The specific workflow will vary depending on your project's requirements.

Task Runners

- Task runners, automate repetitive tasks and workflows in the development process.
- They provide a way to define and execute a series of tasks, such as compiling code, concatenating files, minifying assets, running tests, and more.
- Commonly used Task Runners:
  - Grunt
  - Gulp

Use a Task Runner (e.g., Gulp or Grunt)

- Your project primarily involves automating development tasks and workflows, such as compiling code, running tests, optimizing assets, and copying files.
- You have a relatively simple project with a small number of build-related tasks.
- Your build process requires flexibility in defining custom workflows and tasks.
- Dependency management and package installation are an important part of your build process.
- You have an existing familiarity and expertise with a particular task runner.

Module Bundlers

- Module bundling, on the other hand, involves the process of combining multiple modules and their dependencies into a single file or a set of files, often referred to as bundles.
- The goal of module bundling is to optimize the size and performance of the application by reducing the number of separate HTTP requests required to load the code.
- The bundler resolves dependencies, applies transformations through loaders (e.g., transpiling, minification), and creates a bundle that can be efficiently delivered to the browser.
- Bundling is commonly used to package JavaScript, CSS, images, and other assets together, reducing the network overhead and improving the performance of the application.

The Purpose of Bundlers

Module Bundlers

- Some of the commonly used Module Bundlers are:
  - Webpack
  - Rollup
  - Parcel
  - Browserify

Use a Bundler

- Your project involves module bundling, dependency management, and code optimization.
- Code splitting, lazy loading, or other advanced code optimization techniques are required.
- You need to bundle and optimize assets such as JavaScript, CSS, images, and other resources.
- Tree shaking (removing unused code) or transpiling code to a specific target environment is necessary.
- You want to leverage a rich ecosystem of plugins and loaders specific to bundlers.
- You require advanced optimizations for production builds, including minification and compression.

Using Both

- You can utilize the task runner to automate general development tasks, such as running tests or copying files
- While leverage the bundler to handle module bundling, code optimization, and asset management.
- By combining the strengths of both tools, you can create a comprehensive build process that meets the specific needs of your project.
- For simpler projects, one tool might suffice.

Webpack

- Webpack is a popular module bundler for JavaScript applications.
- It is widely used in modern web development workflows to manage and bundle various assets, such as JavaScript files, CSS stylesheets, images, and more.
- Webpack takes your project's dependencies, processes them through loaders and plugins, and creates optimized bundles that can be served to the browser.

How Webpack Executes?

- Webpack executes in a multi-step process that involves analyzing dependencies, transforming code, and generating bundled output.
- The steps involved in Webpack's execution process:
  - Entry Point Resolution
  - Dependency Graph Creation
  - Module Loading and Transformation
  - Code Splitting
  - Module Resolution
  - Bundling and Output Generation
  - Output Files

Configuring Webpack

- The configuration steps for Webpack involve setting up the necessary configuration file to define how Webpack should behave and process your project's source code.
- Here are the main steps for configuring Webpack:
  - Create a Configuration File
  - Entry Points
  - Output Configuration
  - Loaders
  - Plugins
  - Resolve Configuration
  - Optimization Settings
  - Development vs. Production Mode
  - Additional Customizations

Webpack Client-Side Build

Manual Configuration
- Learn Node JS
- Learn Webpack
- Learn Babel & Config.
- Learn Polyfill

Build & Deploy your Project on Local Server
npm start

WEBPACK-DEV-SERVER

Build your project
npm run build

WEBPACK CLI

WEBPACK

ES 2015 + (Multiple Files)

Babel +
Preset (Env) +
Polyfill (core-js)

Browser Compatible Files (Multiple Files)

Bundling

Single File / Set of Files

BROWSER

Local HTTP Server

Deploy

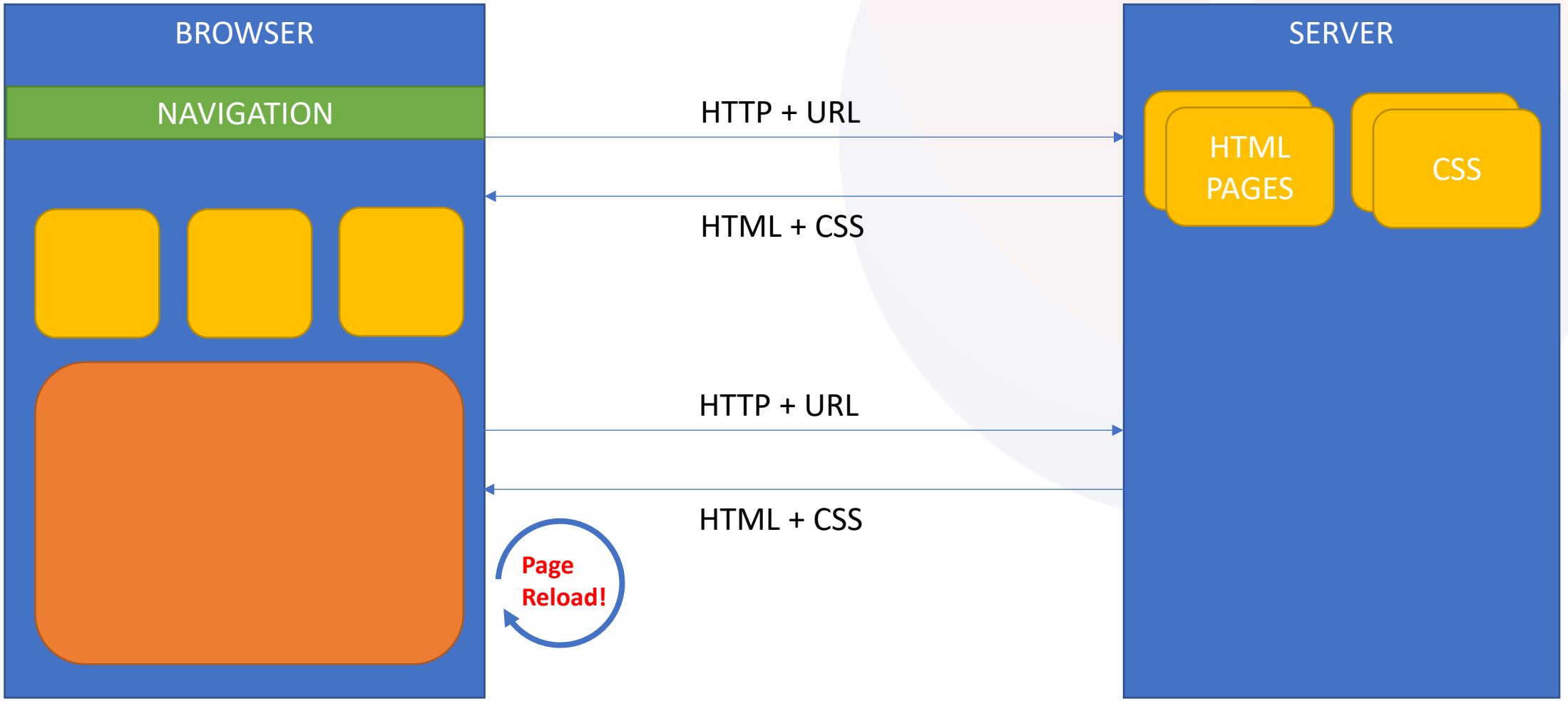
HTML PAGE

Inject Bundles

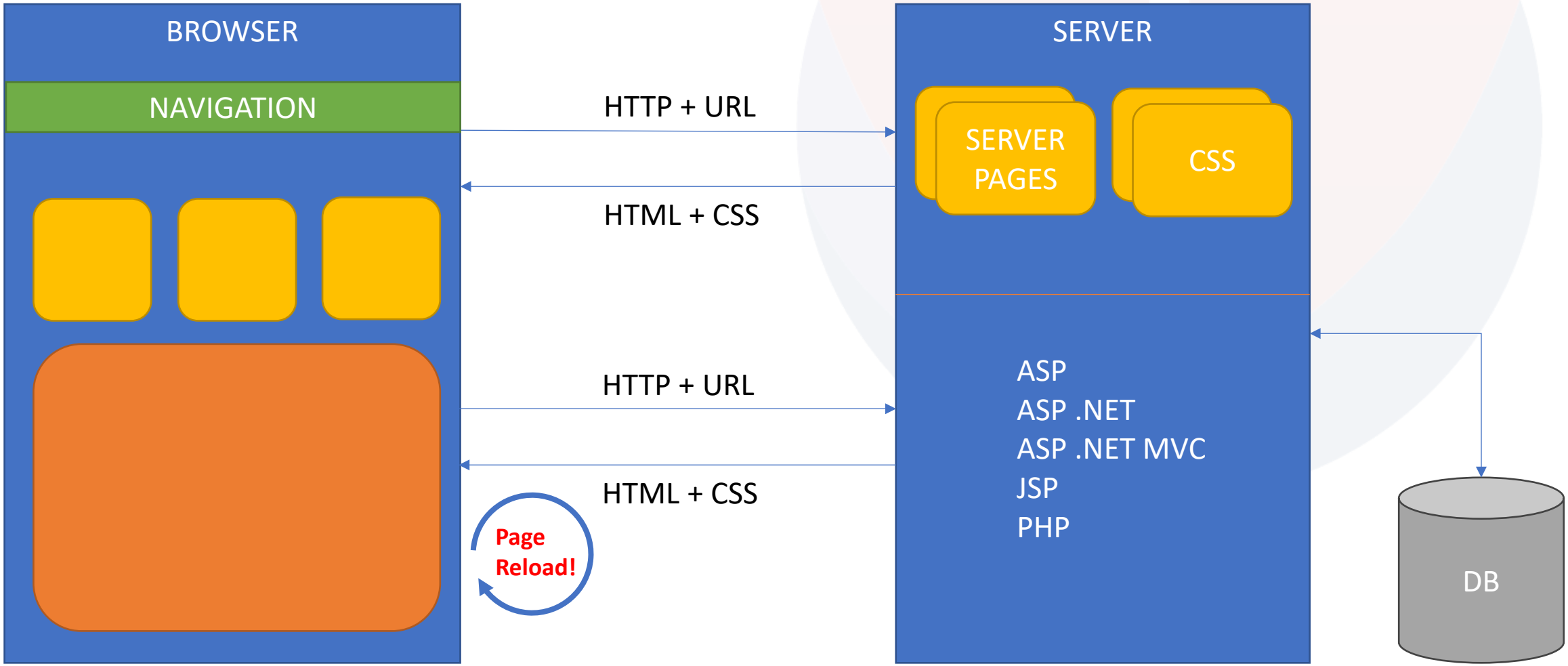
Steps to run the application

- Download and Extract the zip file / clone it from Gitlab Repo.
- Open the extracted folder in Visual Studio Code
- Open the terminal window on the folder path and run
  - npm install
- Once the installation completes, run
  - npm start – Starts the Development Server on localhost:3000
  - npm run build – To create a production build in dist folder

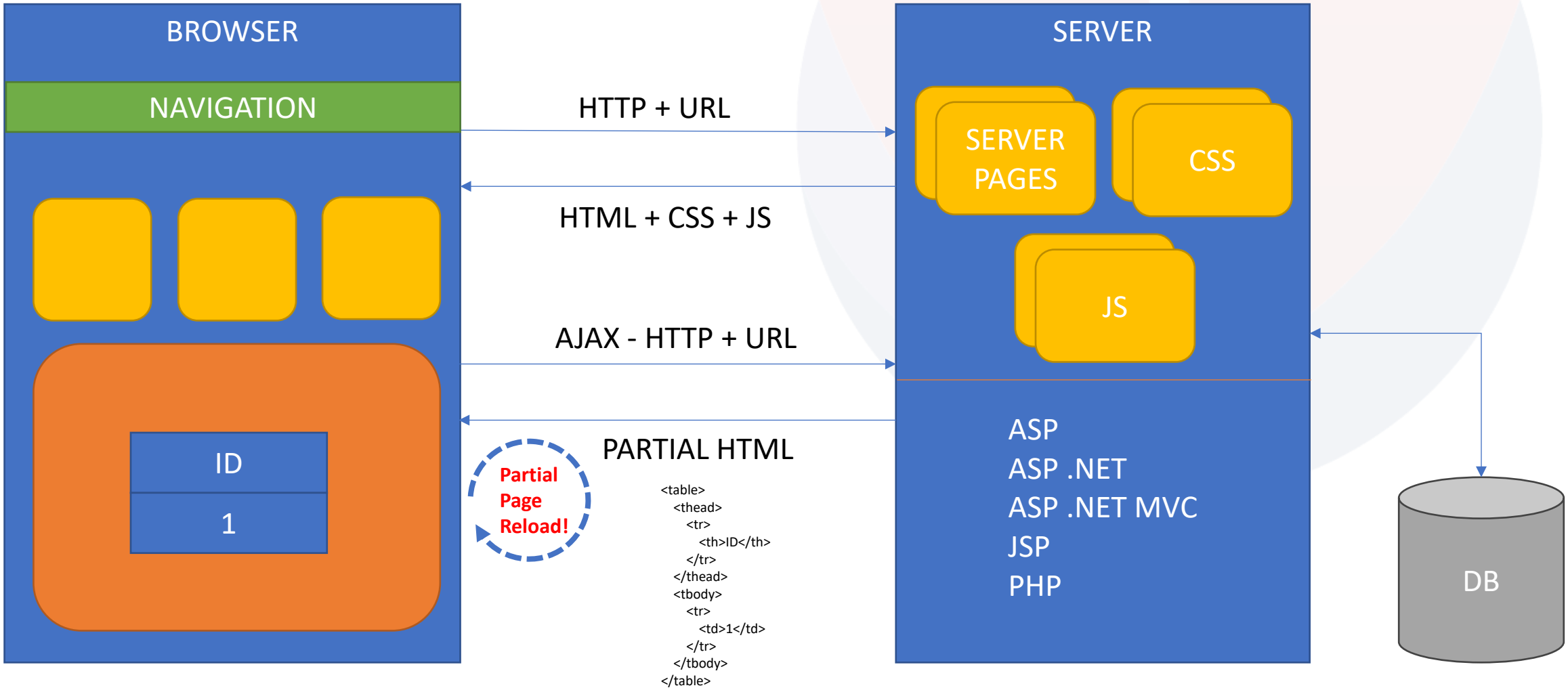
Traditional Website Architecture



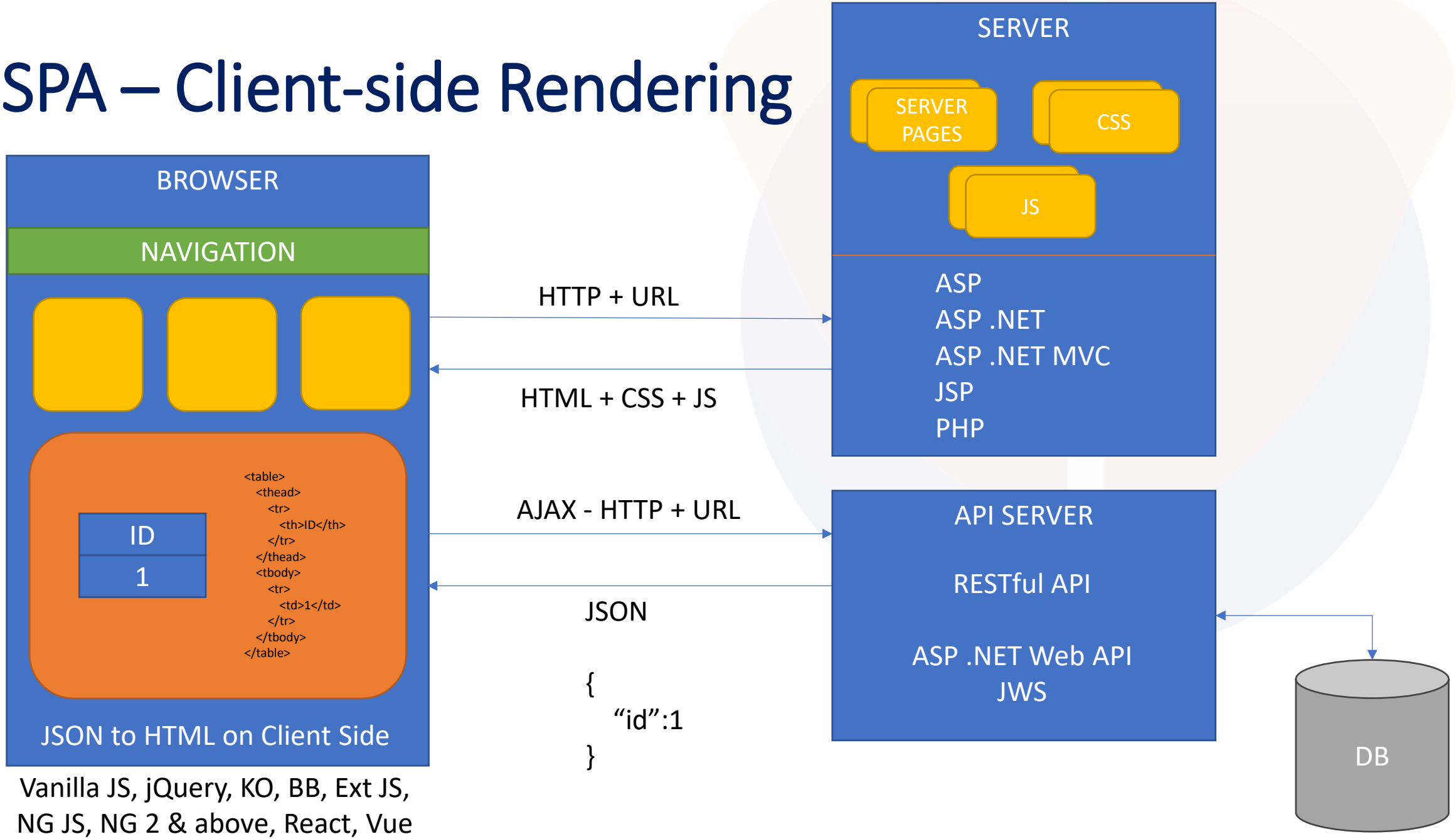
Traditional Web Application Architecture



Traditional Web Application Architecture With Partial Page Rendering (Post Back)



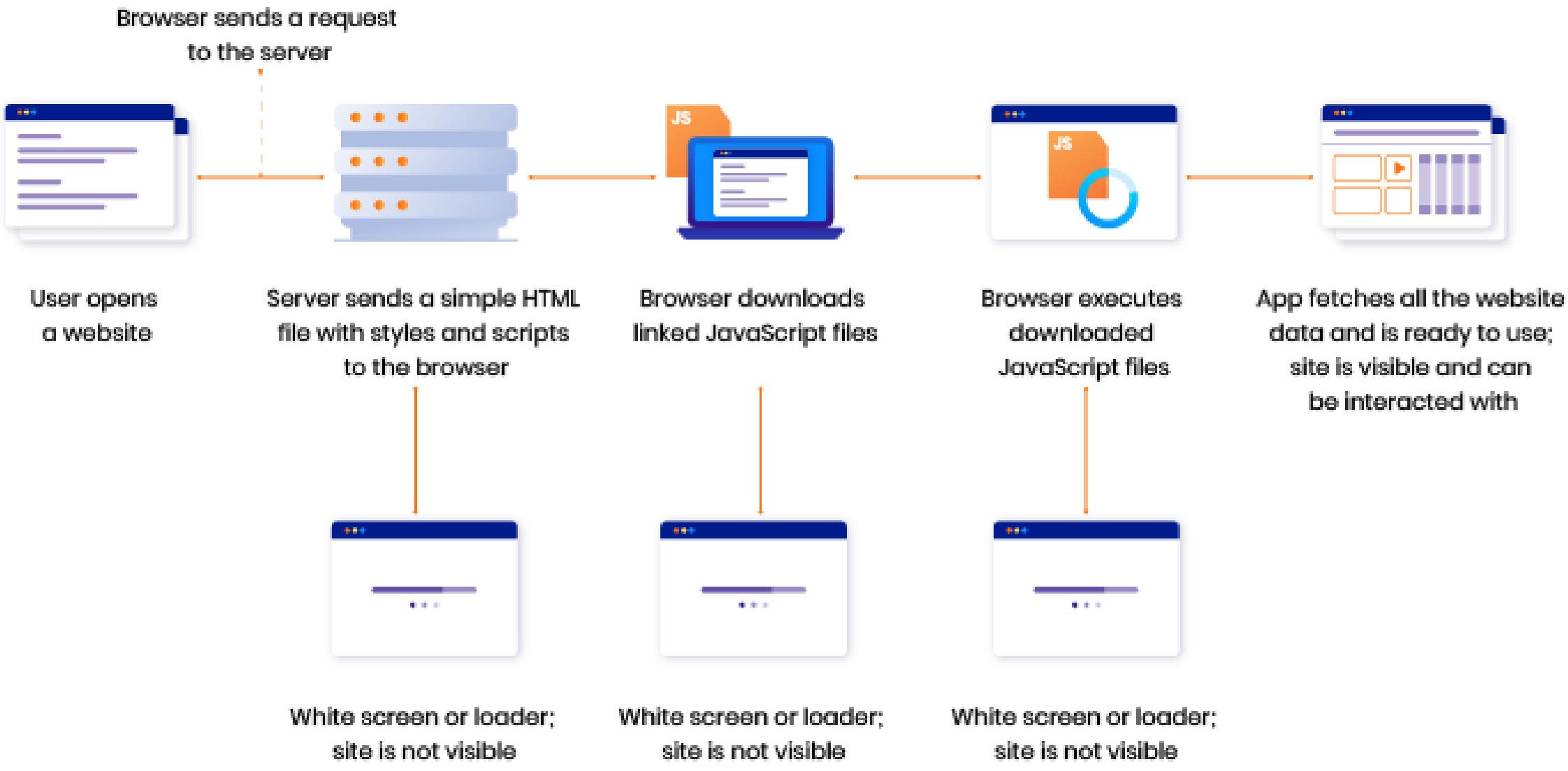
SPA – Client-side Rendering

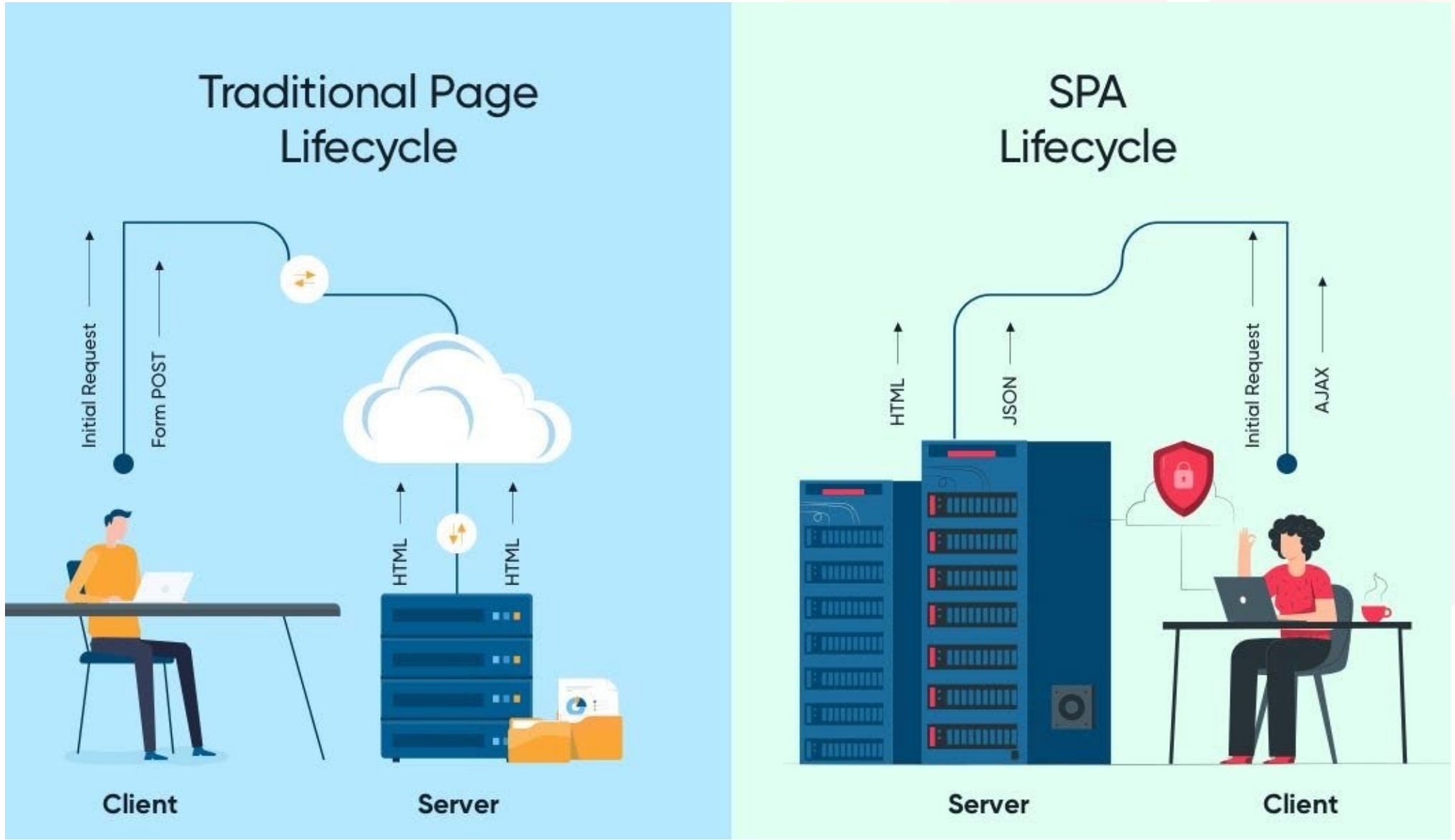


Single Page Application Pros - CSR

- Quick Loading Time
- Seamless User Experience
- Ease in Building Feature-rich Apps
- Uses Less Bandwidth

Client-side rendering




Traditional Page Lifecycle
Initial Request
Form POST
HTML
HTML
Client
Server
SPA Lifecycle
HTML
JSON
Initial Request
AJAX
Server
Client

Single Page Application Cons (Purely CSR)

- Doesn’t Perform Well With SEO
  - One of the metrics that search engines use is the number of pages a website has. However, since SPAs only load a single page, it serves as a disadvantage when ranking on search engines
- Uses a Lot of Browser Resources
  - SPAs require many resources from the web browser since the browser is doing most of the tasks for the SPAs.
- Security Issues
  - As compared to multi-page apps, SPAs are more prone to cross-site scripting attacks.

Variables

- Variables are containers for storing data (storing data values).
- There are 4 Ways to Declare a JavaScript Variable:
  - Using var
  - Using let
  - Using const
  - Using nothing (Not Recommended)
- Always declare JavaScript variables with var, let, or const.
- The var keyword is used in all JavaScript code from 1995 to 2015.
- The let and const keywords were added to JavaScript in 2015.
- If you want your code to run in older browser, you must use var.

Datatypes

- The set of types in the JavaScript language consists of **primitive values** and **objects**.
  - Primitive Values - Immutable datum represented directly at the lowest level of the language
    - Null – null
    - Undefined – undefined
    - Boolean() – boolean
    - Number() – number
    - String() – string
    - Symbol (ECMAScript 2015) – symbol – **We cannot use new keyword with Symbol()**
  - Objects - An object is a mutable value in memory which is possibly referenced by an identifier.
    - Object() - object

JavaScript types

- All types except objects define immutable values (that is, values which can't be changed)
- **Boolean type:** Boolean represents a logical entity and can have two values: true and false. See Boolean and Boolean for more details.
- **Null type:** The Null type has exactly one value: null. See null and Null for more details.
- **Undefined type:** A variable that has not been assigned a value has the value undefined. See undefined and Undefined for more details.
- **Number type:** The Number type is a double-precision 64-bit binary format value (numbers between -(2^53 − 1) and 2^53 − 1). In addition to representing floating-point numbers, the number type has three symbolic values: +Infinity, -Infinity, and NaN ("Not a Number").
- **String type:** JavaScript's String type is used to represent textual data. It is a set of "elements" of 16-bit unsigned integer values. Each element in the String occupies a position in the String.
- **Symbol type:** A Symbol is a unique and immutable primitive value and may be used as the key to an Object property.
- **Objects:** An object is a value in memory which is possibly referenced by an identifier. objects can be seen as a collection of properties. Properties are identified using key values. A key value is either a String value or a Symbol value.

Mutable vs Immutable

- JavaScript Engine handles mutable and immutable objects differently.
- Immutable are quicker to access than mutable objects.
- Mutable objects are great to use when you need to change the size of the object, example array, set etc..
- Immutable are used when you need to ensure that the object you made will always stay the same.
- Immutable objects are fundamentally expensive to “change”, because doing so involves creating a copy.
- Changing mutable objects is cheap.

Standard built-in objects

- Value properties
  - Infinity
  - NaN
  - undefined
  - globalThis

- Function properties
  - eval()
  - uneval()
  - isFinite()
  - isNaN()
  - parseFloat()
  - parseInt()
  - encodeURI()
  - encodeURIComponent()
  - decodeURI()
  - decodeURIComponent()

- Fundamental objects
  - Object
  - Function
  - Boolean
  - Symbol

- Error objects
  - Error
  - AggregateError
  - EvalError
  - InternalError
  - RangeError
  - ReferenceError
  - SyntaxError
  - TypeError
  - URIError

Standard built-in objects

- Numbers and dates
  - Number
  - Math
  - Date
- Text processing
  - String
  - RegExp
- Indexed collections
  - Array
  - Int8Array
  - Uint8Array
  - Uint8ClampedArray
  - Int16Array
  - Uint16Array
  - Int32Array
  - Uint32Array
  - Float32Array
  - Float64Array

Standard built-in objects

- Keyed collections
  - Map
  - Set
  - WeakMap
  - WeakSet
- Structured data
  - ArrayBuffer
  - DataView
  - JSON
- Control abstraction objects
  - Promise
  - Generator
  - GeneratorFunction
  - AsyncFunction
  - AsyncGenerator
  - AsyncGeneratorFunction
- Reflection
  - Reflect
  - Proxy

Statements & Loops

- Control Flow
  - Block { }
  - break
  - continue
  - Empty
  - if…else
  - switch
  - throw
  - try…catch
- Iterations
  - for
  - for...in
  - for…of
  - while
  - do…while

Functions

- Functions are one of the fundamental building blocks in JavaScript.
- A function in JavaScript is like a procedure—a set of statements that performs a task or calculates a value, but for a procedure to qualify as a function, it should take some input and return an output.
- To use a function, you must define it somewhere in the scope from which you wish to call it.
- In JavaScript, functions are first-class objects, because they can have properties and methods just like any other object.
- What distinguishes them from other objects is that functions can be called.
- Every JavaScript function is a Function object.

Functions

- function
  - Declares a function with the specified parameters.
- function*
  - Generator Functions, enable writing iterators more easily
- async function
  - Declares an async function with the specified parameters.

Function Parameters & Arguments

- Function **Parameters** are the names that are define in the function definition and real values passed to the function in function definition are known as **arguments**.
- Parameter Rules:
  - There is no need to specify the data type for parameters in JavaScript function definitions.
  - It does not perform type checking based on passed in JavaScript function.
  - It does not check the number of received arguments.
- If a function is called with missing arguments (less than declared), the missing values are set to undefined.

The Arguments Object

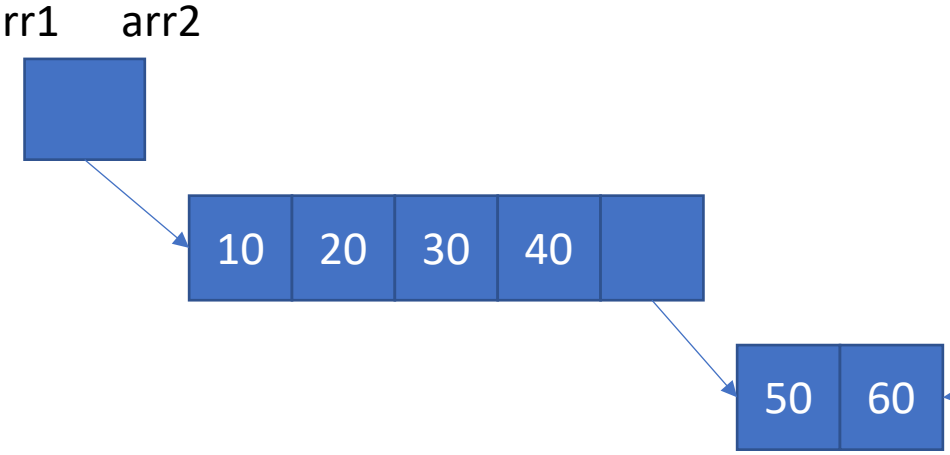
- JavaScript functions have a built-in object called the arguments object.
- The argument object contains an array of the arguments used when the function was called (invoked).
- If a function is called with too many arguments (more than declared), these arguments can be reached using the arguments object.

Rest & Spread

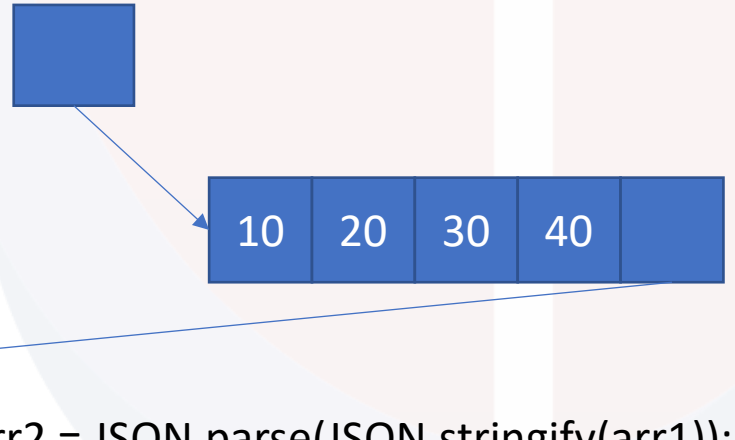
- The rest operator (…) allows us to call a function with any number of arguments and then access those excess arguments as an array. The rest operator also allows us in destructuring array or objects.
- The spread operator (…) allows us to expand an iterable like array into its individual elements.

Reference Vs Shallow Vs Deep Copy

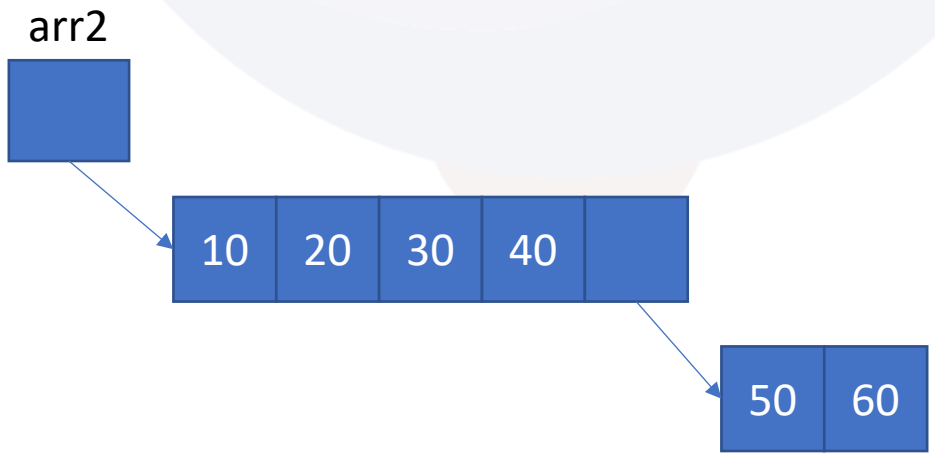
var arr2 = arr1;



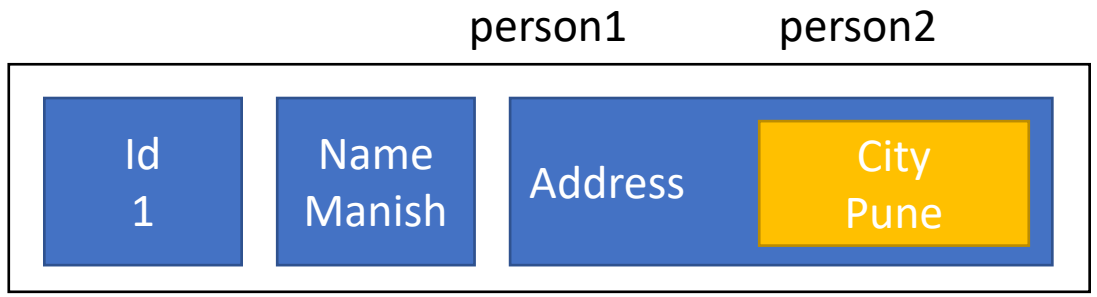
var arr2 = […arr1];



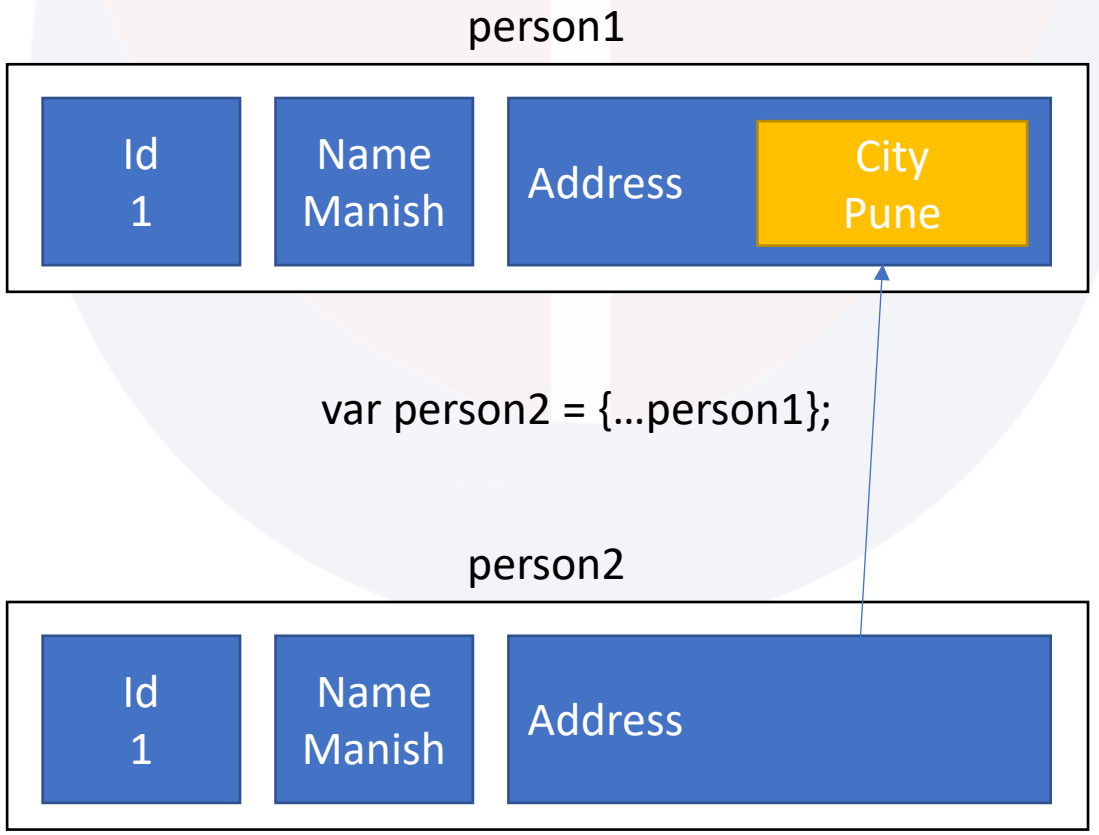
var arr2 = JSON.parse(JSON.stringify(arr1));



Reference Copy Vs Shallow Copy



var person2 = person1;



var person2 = {...person1};

Pure and Impure Functions

- A side-effect is any operation your function performs that is not related to computing the final output, including but not limited to:
  - Modifying a global variable
  - Modifying an argument
  - Making HTTP requests
  - DOM manipulation
  - Reading/writing files
- A pure function must both be predictable and without side-effects. If either of these criteria is not met, we're dealing with an impure function.
- A pure function, returns same result if same arguments are passed in, no matter how many times it runs.
- An impure function, may return different result if same arguments are passed in, on multiple runs

IIFE

- **Immediately Invoked Function Expression** (IIFE) is one of the most popular design patterns in JavaScript.
- It is pronounced as, iify.
- As name suggest, IIFE is a function expression that automatically invokes after completion of the definition.
- IEFE solves this problem by having its own scope and restricting functions and variables to become global.
- The functions and variables declare inside IIFE will not pollute global scope even they have same name as global variables & functions.

Function Overloading

- JavaScript does **not** support function overloading.
- If we create two function with same name, the second function will overwrite the first function.
- To handle same function names in JavaScript
  - The first recommendation is to use a different and self-explanatory name for functions.
  - Using default values for the parameters.
  - Checking the number of arguments and type of arguments.
  - Passing the last parameter to describe the kind of option the function should perform. We can use switch or if...else to perform the required operation.

Callback

- A callback is a function that is passed as an argument to another function, which is expected to invoke it either immediately or at some point in the future.
- Callbacks can be seen as a form of the continuation-passing style (CPS), in which control is passed explicitly in the form of a continuation; in this case the callback passed as an argument represents a continuation.
- There are two types of callbacks.
  - A callback passed to a function can be invoked **synchronously** before return
  - Or it can be deferred to execute **asynchronously** some time after return.

JavaScript ‘this’ keyword

- In JavaScript, **“context”** refers to an object.
- Within an object, the keyword **“this”** refers to that object, and provides an interface to the properties and methods that are members of that object.
- When a function is executed, the keyword “this” refers to the object that the function is executed in.
- Scenarios
  - When a function executes in the global context, “this” refers to the global, or “window” object
  - When a function is a method of an Object, “this” refers to that object (unless it is manually executed in the context of a different object)
  - When a function executes inside of another function (no matter how deeply nested), “this” refers to the object whose context within which it is executed
  - When you instantiate a constructor function, inside of the instance object, “this” refers to the instance object

Closure

- A closure is a function or reference to a function together with a referencing environment — a table storing a reference to each of the non-local variables of that function.
- Closures are functions that refer to independent (free) variables. In other words, the function defined in the closure 'remembers' the environment in which it was created.
- A closure is a special kind of object that combines two things: a function, and the environment in which that function was created. The environment consists of any local variables that were in-scope at the time that the closure was created.

Function Currying

- Currying is a way of constructing functions that allows partial application of a function’s arguments.
- What this means is that you can pass all the arguments a function is expecting and get the result or pass a subset of those arguments and get a function back that’s waiting for the rest of the arguments.

Higher Order Functions

- Functions that operate on other functions, by taking them as arguments and by returning them, are called higher-order functions.
- Higher-order functions allow us to abstract over actions, not just values.

Indexed collections

- Array
- Int8Array
- Uint8Array
- Uint8ClampedArray
- Int16Array
- Uint16Array
- Int32Array
- Uint32Array
- Float32Array
- Float64Array

Arrays

- **Arrays** are an ordered collection that can hold data of any type. In JavaScript, arrays are created with square brackets [...] or using Array constructor and allow duplicate elements.
- JavaScript objects and arrays were the most important data structures to handle collections of data.
- The combination of objects and arrays was able to handle data in many scenarios, however, there were a few shortcomings as follows
  - Object keys can only be of type string.
  - Objects don't maintain the order of the elements inserted into them.
  - Objects lack some useful methods, which makes them difficult to use in some situations. For example, you can't compute the size (length) of an object easily. Also, enumerating an object is not that straightforward.
  - Arrays are collections of elements that allow duplicates. Supporting arrays that only have distinct elements requires extra logic and code.

Keyed collections

- Map
- Set
- WeakMap
- WeakSet

Map

- Map is a collection of key-value pairs where the key can be of any type. Map remembers the original order in which the elements were added to it, which means data can be retrieved in the same order it was inserted in.
- In other words, Map has characteristics of both Object and Array:
  - Like an object, it supports the key-value pair structure.
  - Like an array, it remembers the insertion order.
- A new Map can be created using Map Constructor.
- Use Map when:
  - You may want to create keys that are non-strings. Storing an object as a key is a very powerful approach. Map gives you this ability by default.
  - You need a data structure where elements can be ordered. Regular objects do not maintain the order of their entries.
  - You are looking for flexibility without relying on an external library like lodash.

Set

- A Set is a collection of unique elements that can be of any type.
- Set is also an ordered collection of elements, which means that elements will be retrieved in the same order that they were inserted in.
- A Set in JavaScript behaves the same way as a mathematical set.
- Use Set when
  - You need to maintain a distinct set of data to perform set operations like union, intersection, difference, and so on.

Weak Map

- The WeakMap object is a collection of key/value pairs in which the keys are objects only and the values can be arbitrary values.
- The object references in the keys are held weakly, meaning that they are a target of garbage collection (GC) if there is no other reference to the object anymore.
- One difference to Map objects is that WeakMap keys are not enumerable.
- One use case of WeakMap objects is to store private data for an object, or to hide implementation details.

Weak Set

- WeakSet objects are collections of objects.
- An object in the WeakSet may only occur once.
- It is unique in the WeakSet's collection, and objects are not enumerable.
- The main differences to the Set object are:
  - In contrast to Sets, WeakSets are collections of objects only, and not of arbitrary values of any type.
  - The WeakSet is weak: References to objects in the collection are held weakly. If there is no other reference to an object stored in the WeakSet, they can be garbage collected. That also means that there is no list of current objects stored in the collection.
  - WeakSets are not enumerable.
- The use cases of WeakSet objects are limited.
  - They will not leak memory, so it can be safe to use DOM elements as a key and mark them for tracking purposes.

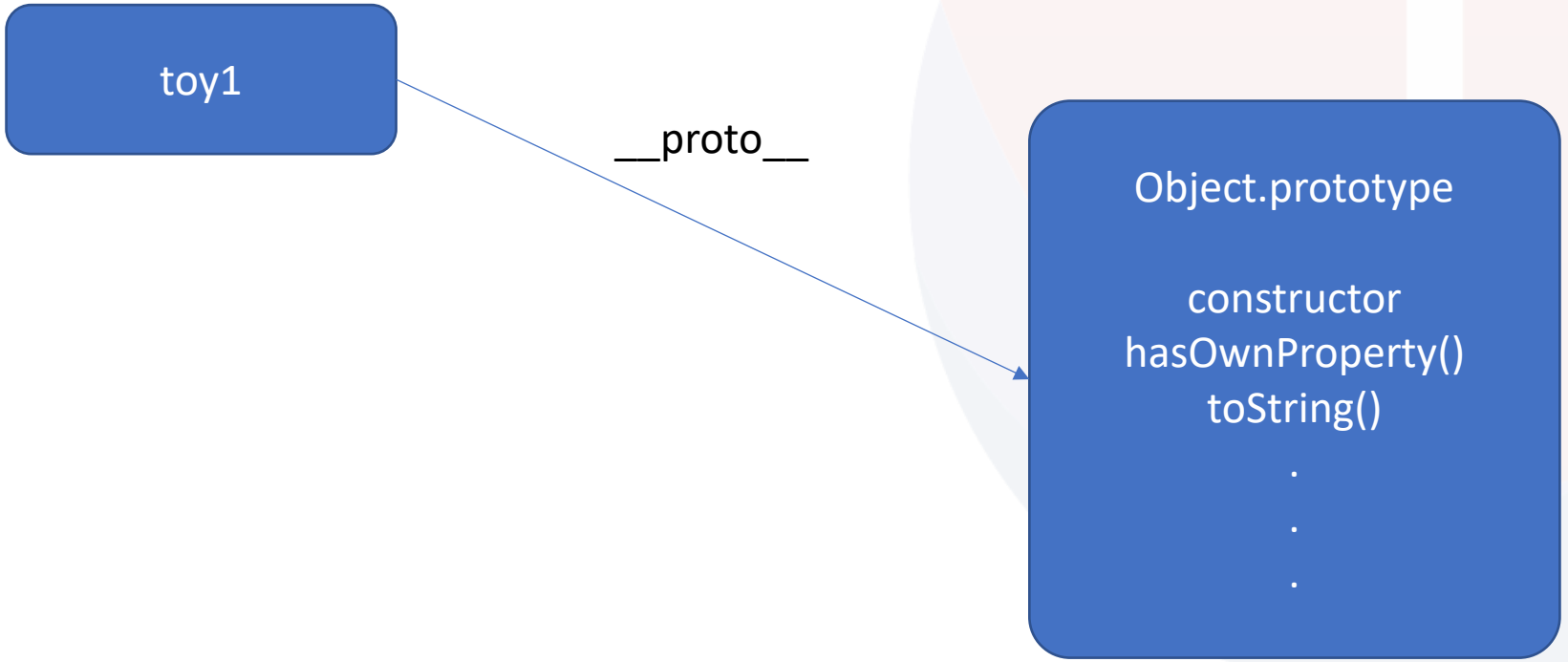
Objects

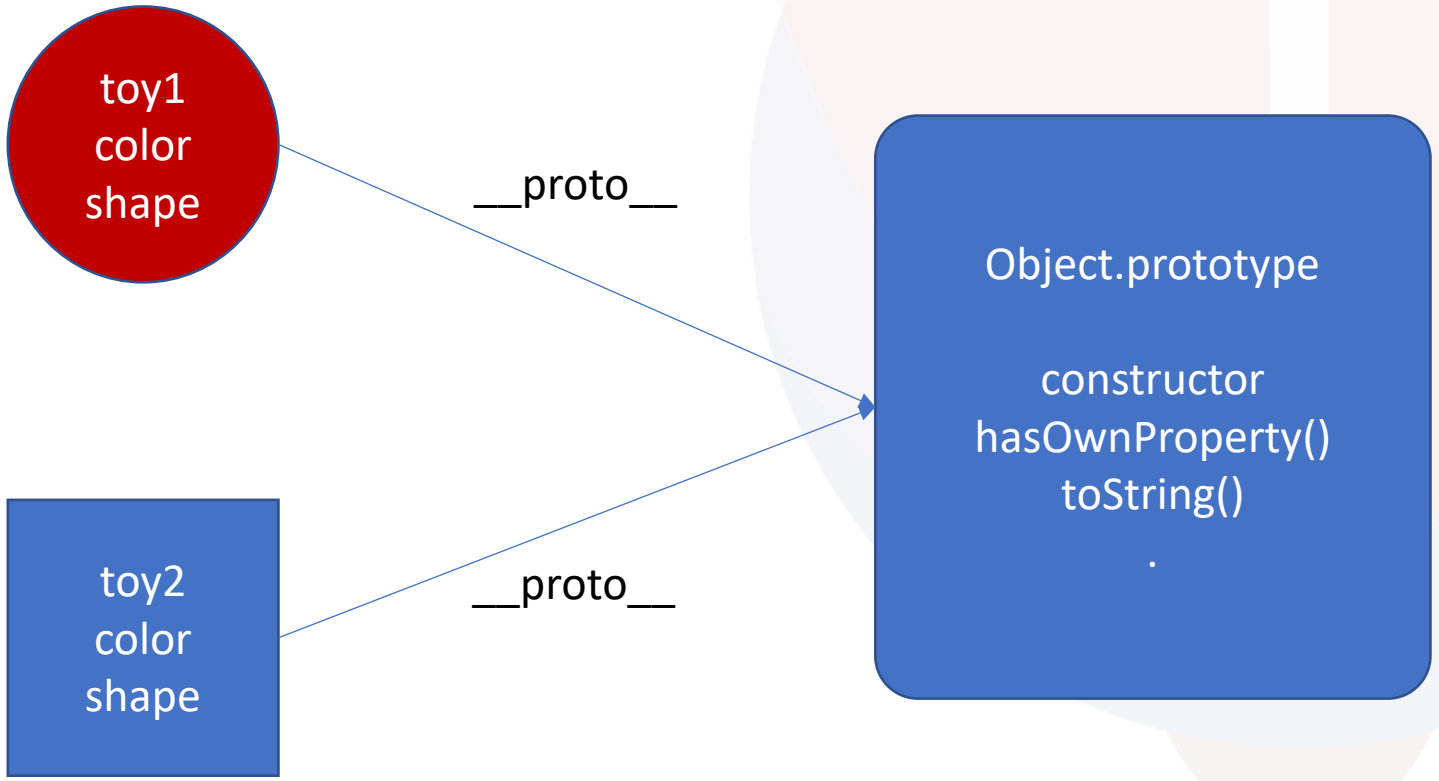
- A JavaScript object is an entity having state and behavior (properties and method). For example: car, pen, bike, chair, glass, keyboard, monitor etc.
- JavaScript is an object-based language. Everything is an object in JavaScript.
- We can create objects directly.
  - By object literal
  - By creating instance of Object directly (using new keyword)
  - By using an object constructor (using new keyword)

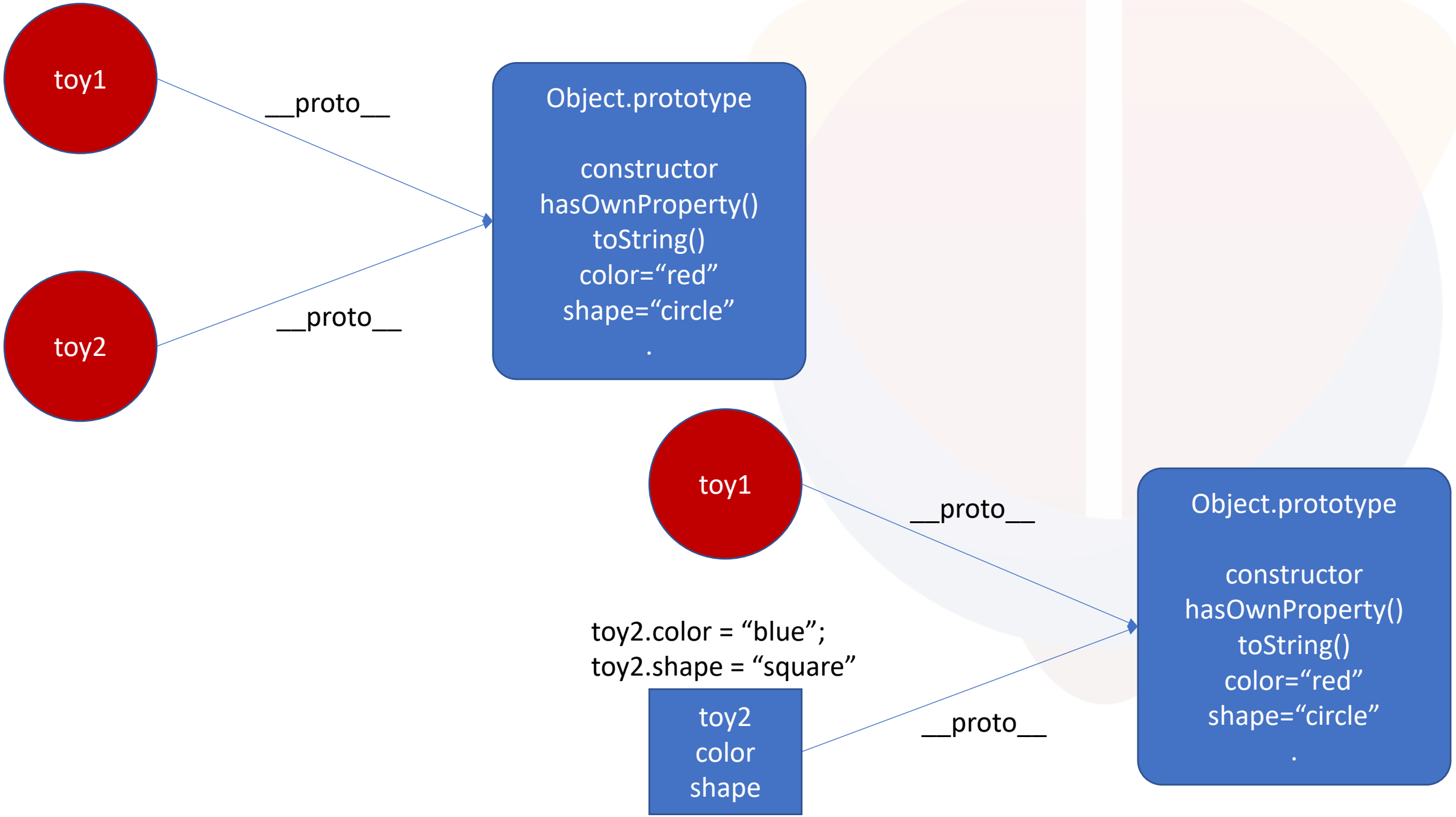
JSON

- JSON is a format for storing and transporting data.
- JSON is often used when data is sent from a server to a web page.
- What is JSON?
  - JSON stands for JavaScript Object Notation
  - JSON is a lightweight data interchange format
  - JSON is language independent *
  - JSON is "self-describing" and easy to understand
- The JSON syntax is derived from JavaScript object notation syntax, but the JSON format is text only.
- Code for reading and generating JSON data can be written in any programming language.

var toy1 = new Object()




toy1
color
shape
__proto__
toy2
color
shape
__proto__
Object.prototype
constructor
hasOwnProperty()
toString()
.

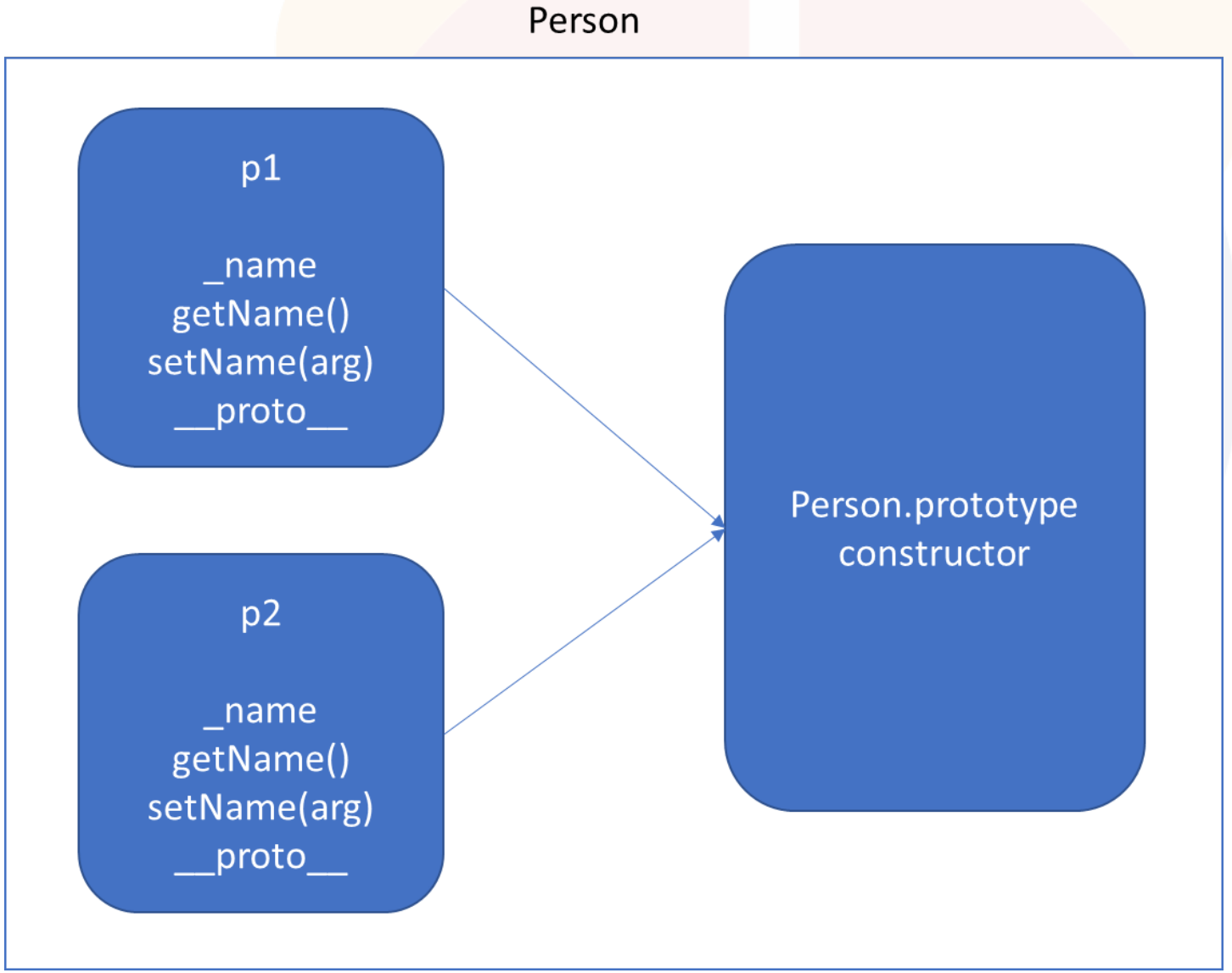

toy1
__proto__
Object.prototype
constructor
hasOwnProperty()
toString()
color="red"
shape="circle"
.
toy2
__proto__
toy1
__proto__
Object.prototype
constructor
hasOwnProperty()
toString()
color="red"
shape="circle"
.
toy2.color = "blue";
toy2.shape = "square"
toy2
color
shape
__proto__

Constructor Function

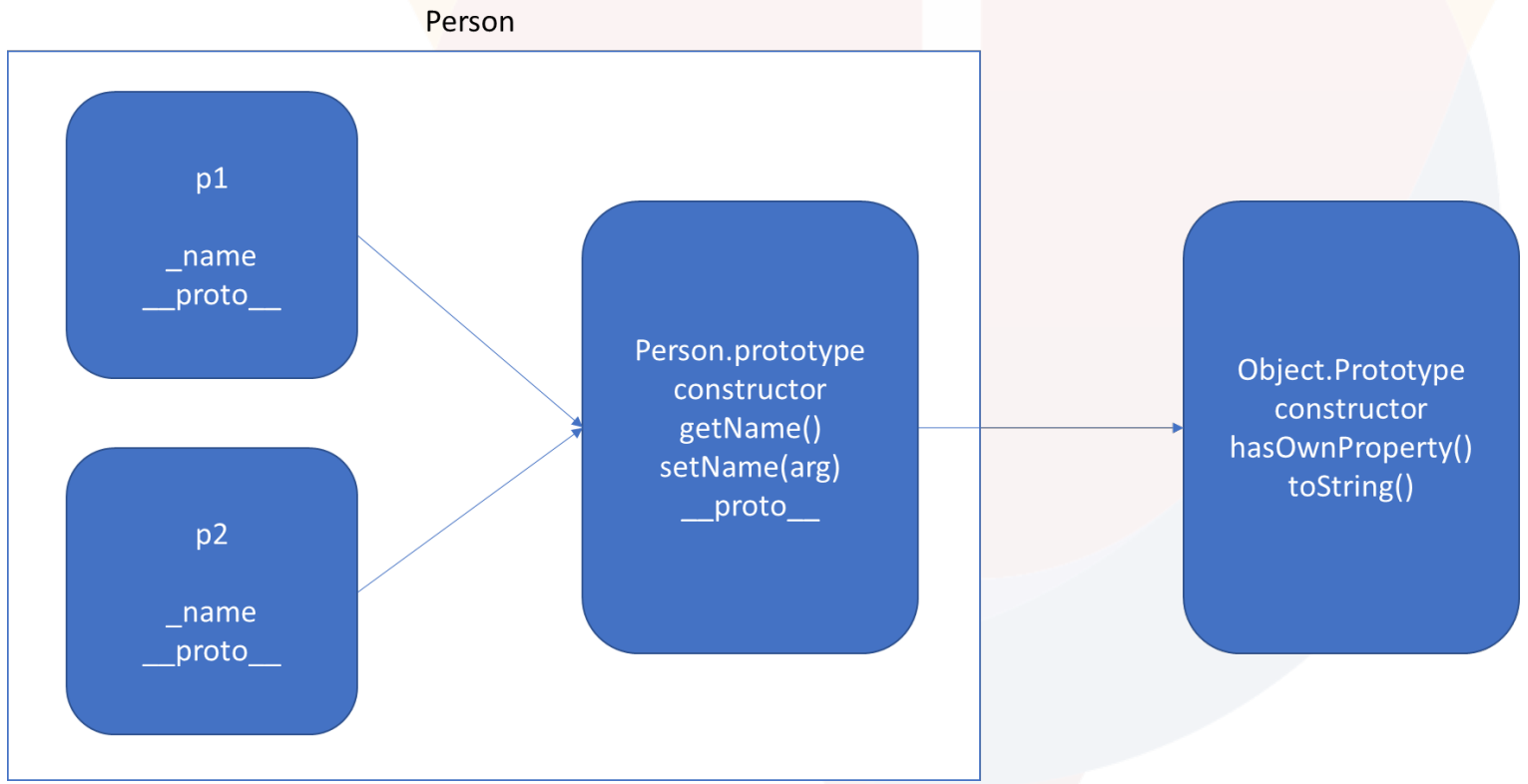
- In JavaScript, a constructor function is used to create objects.
- To create an object from a constructor function, we use the new keyword.
- It is considered a good practice to capitalize the first letter of your constructor function.
- When this keyword is used in a constructor function, this refers to the object when the object is created.

Function Using this keyword

*Will increase memory Usage



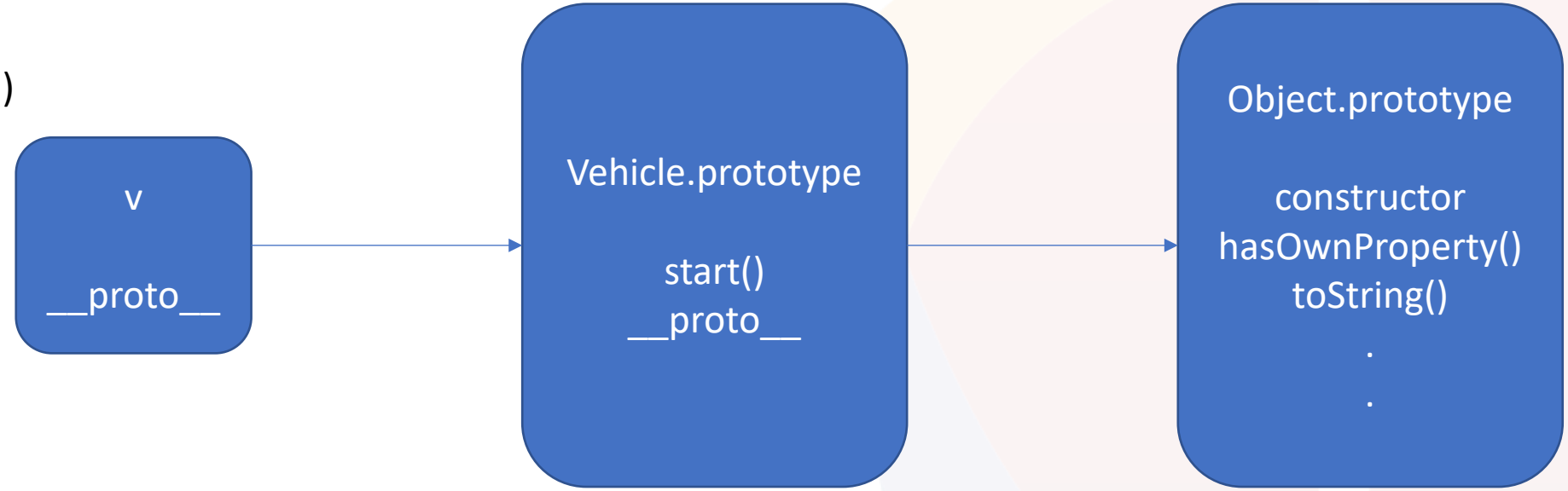
Function Using prototype



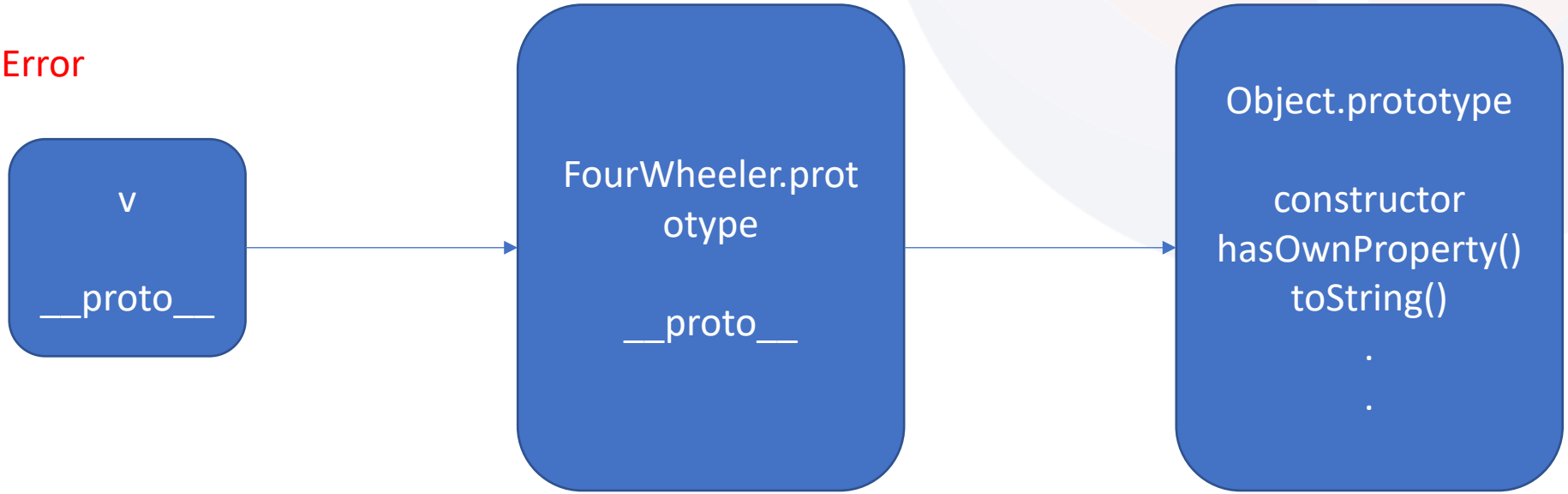
static

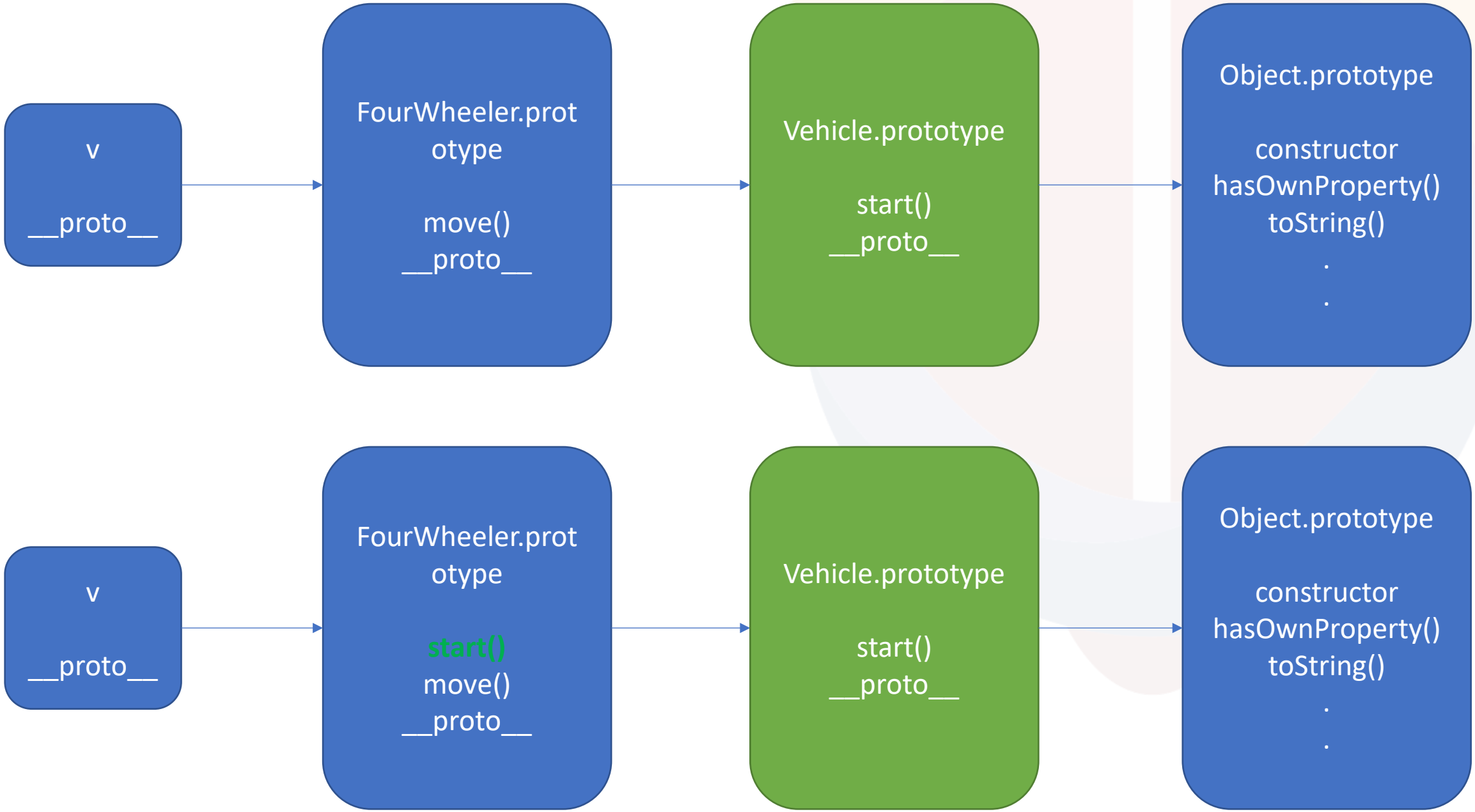
- The static keyword defines a static method or property for a class, or a class static initialization block.
- Neither static methods nor static properties can be called on instances of the class. Instead, they're called on the class itself.
- Static methods are often utility functions, such as functions to create or clone objects, whereas static properties are useful for caches, fixed-configuration, or any other data you don't need to be replicated across instances.

v.start()
v.toString()



v.start() - Error




v
__proto__
FourWheeler.prototype
move()
__proto__
Vehicle.prototype
start()
__proto__
Object.prototype
constructor
hasOwnProperty()
toString()
.
.
v
__proto__
FourWheeler.prototype
start()
move()
__proto__
Vehicle.prototype
start()
__proto__
Object.prototype
constructor
hasOwnProperty()
toString()
.
.

Well-Known symbols

- A list of symbols that JavaScript treats specially is published as well-known symbols.
- Well-known symbols are used by built-in JavaScript algorithms. For example **Symbol.iterator (@@iterator)** is utilized to iterate over items in arrays, strings, or even to define your own iterator function.

Well-known symbol

Specification Name	[[Description]]	Value and Purpose
@@hasInstance	"Symbol.hasInstance"	A method that determines if a constructor object recognizes an object as one of the constructor's instances. Called by the semantics of the instanceof operator.
@@isConcatSpreadable	"Symbol.isConcatSpreadable"	A Boolean valued property that if true indicates that an object should be flattened to its array elements by Array.prototype.concat.
@@iterator	"Symbol.iterator"	A method that returns the default Iterator for an object. Called by the semantics of the for-of statement.
@@match	"Symbol.match"	A regular expression method that matches the regular expression against a string. Called by the String.prototype.match method.
@@replace	"Symbol.replace"	A regular expression method that replaces matched substrings of a string. Called by the String.prototype.replace method.
@@search	"Symbol.search"	A regular expression method that returns the index within a string that matches the regular expression. Called by the String.prototype.search method.
@@species	"Symbol.species"	A function valued property that is the constructor function that is used to create derived objects.
@@split	"Symbol.split"	A regular expression method that splits a string at the indices that match the regular expression. Called by the String.prototype.split method.
@@toPrimitive	"Symbol.toPrimitive"	A method that converts an object to a corresponding primitive value. Called by the ToPrimitive abstract operation.
@@toStringTag	"Symbol.toStringTag"	A String valued property that is used in the creation of the default string description of an object. Accessed by the built-in method Object.prototype.toString.
@@unscopables	"Symbol.unscopables"	An object valued property whose own property names are property names that are excluded from the with environment bindings of the associated object.

Iterators & Generators

- Iterators and Generators bring the concept of iteration directly into the core language and provide a mechanism for customizing the behavior of for...of loops.
- In JavaScript an iterator is an object which defines a sequence and potentially a return value upon its termination.
- Specifically, an iterator is any object which implements the Iterator protocol by having a next() method that returns an object with two properties:
  - value
    - The next value in the iteration sequence.
  - done
    - This is true if the last value in the sequence has already been consumed. If value is present alongside done, it is the iterator's return value.
- Once created, an iterator object can be iterated explicitly by repeatedly calling next().

Generators

- In JavaScript, generators provide a new way to work with functions and iterators. Generators provide an easier way to implement iterators.
- Using a generator,
  - you can stop the execution of a function from anywhere inside the function
  - and continue executing code from a halted position
- To create a generator, you need to first define a generator function with function* symbol.
- You can pause the execution of a generator function without executing the whole function body by using yield keyword.
- The yield expression returns a value. However, unlike the return statement, it doesn't terminate the program. That's why you can continue executing code from the last yielded position.

Generator Methods

Method	Description
next()	Returns a value of yield
return()	Returns a value and terminates the generator
throw()	Throws an error and terminates the generator

Uses of Generators

- Generators let us write cleaner code while writing asynchronous tasks.
- Generators provide an easier way to implement iterators.
- Generators execute its code only when required.
- Generators are memory efficient.

Modules

- As our application grows bigger, we want to split it into multiple files, so called “modules”.
- Until ES6, JavaScript existed without a language-level module syntax.
- JavaScript community invented a variety of ways to organize code into modules, special libraries to load modules on demand.
  - AMD – one of the most ancient module systems, initially implemented by the library require.js.
  - CommonJS – the module system created for Node.js server.
  - UMD – one more module system, suggested as a universal one, compatible with AMD and CommonJS.
- The language-level module system appeared in the standard in ECMASCRIPT 2015

Modules

- A module is just a file. One script is one module. As simple as that.
- A module may contain a class or a library of functions for a specific purpose.
- Modules can load each other and use special directives export and import to interchange functionality, call functions of one module from another one:
  - **export** keyword labels variables and functions that should be accessible from outside the current module.
  - **import** allows the import of functionality from other modules.

Promise

- The Promise object represents the eventual completion (or failure) of an asynchronous operation, and its resulting value.
- It allows you to associate handlers with an asynchronous action's eventual success value or failure reason.
- This lets asynchronous methods return values like synchronous methods: instead of immediately returning the final value, the asynchronous method returns a promise to supply the value at some point in the future.
- Essentially, a promise is a returned object to which you attach callbacks, instead of passing callbacks into a function.

Promise vs Callbacks

- There are several advantages of using a promise over a callback in JavaScript. Here are some of the advantages of using a promise:
  - They are composable, unlike callbacks, therefore we can avoid callback hells
  - You can easily execute code with Promise.all when multiple responses are returned
  - You can wait for only one result from concurrent pending promises with the help of Promise.race
  - You can write asynchronous code in a synchronous manner if you use it in conjunction with async / await.
- While they are a perfect choice over callbacks, there are some minor disadvantages we need to cover as well. Here are some:
  - They can only operate on a single value at a time
  - They are not available in older browser; they must be polyfilled

Promise Guarantees

- Unlike old-fashioned passed-in callbacks, a promise comes with some guarantees:
  - Callbacks added with then() will never be invoked before the completion of the current run of the JavaScript event loop.
  - These callbacks will be invoked even if they were added after the success or failure of the asynchronous operation that the promise represents.
  - Multiple callbacks may be added by calling then() several times. They will be invoked one after another, in the order in which they were inserted.
  - One of the great things about using promises is chaining.
    - A common need is to execute two or more asynchronous operations back-to-back, where each subsequent operation starts when the previous operation succeeds, with the result from the previous step. We accomplish this by creating a **promise chain**.

Fetch API

- The Fetch API provides a JavaScript interface for accessing and manipulating parts of the HTTP pipeline, such as requests and responses.
- It also provides a global fetch() method that provides an easy, logical way to fetch resources asynchronously across the network.
- This kind of functionality was previously achieved using XMLHttpRequest.
- Fetch provides a better alternative that can be easily used by other technologies such as Service Workers.
- Fetch also provides a single logical place to define other HTTP-related concepts such as CORS and extensions to HTTP.

Beyond ES6

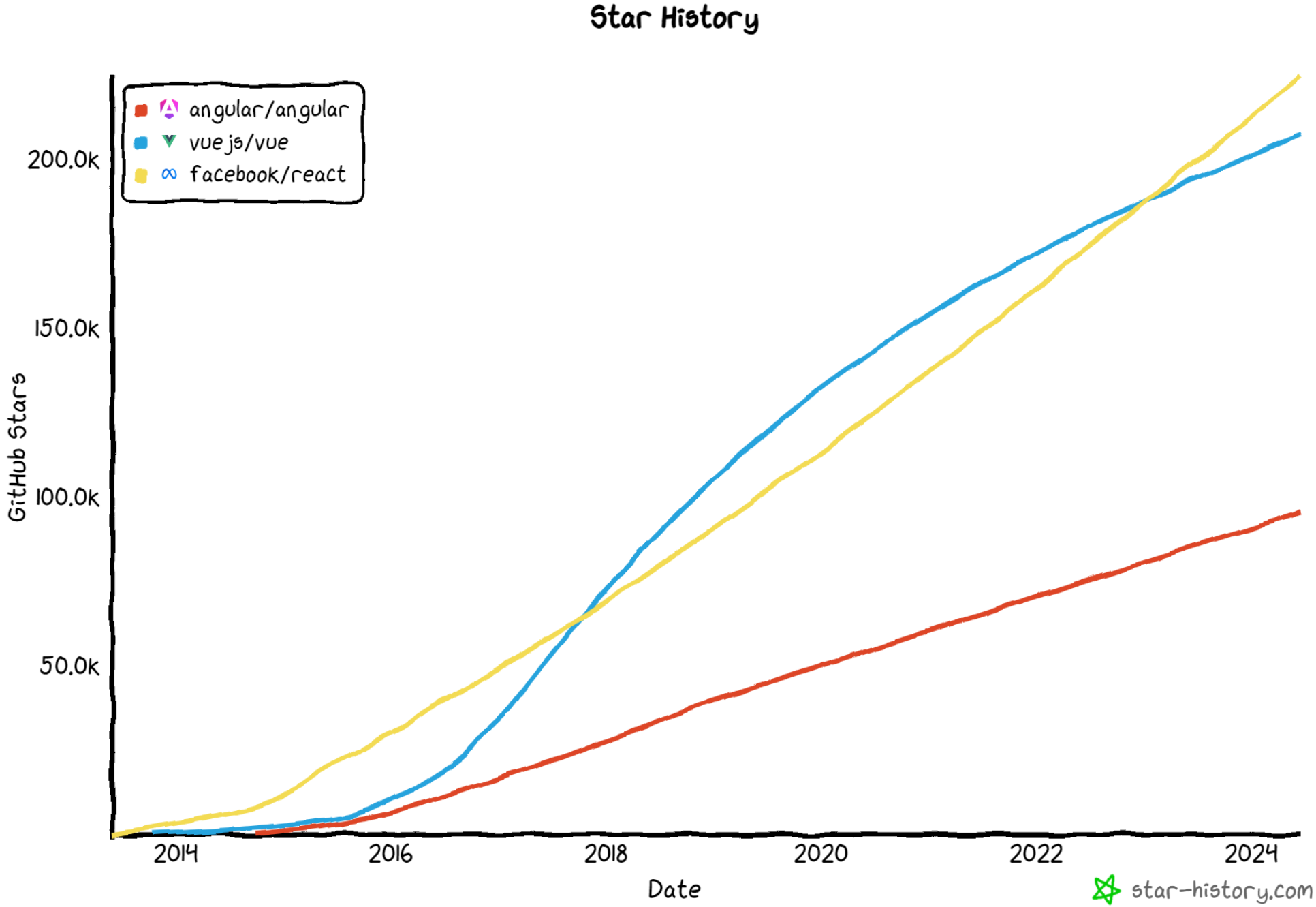
- ES 7 (ECMAScript 2016) New Features
  - Exponentiation Operator
  - Array: includes()
- ES 8 (ECMAScript 2017) New Features
  - Object Static Methods
    - Object: entries() & values()
    - Object: getOwnPropertyDescriptors()
  - String Padding
  - Trailing commas in Function
  - Async Functions
    - Async Await Flow
    - Async Await
- ES 9 (ECMAScript 2018) New Features
  - Object rest/spread properties
  - Asynchronous Iteration
  - Promise: finally()
  - Template literal Revision

React With Redux

Understanding the Essentials

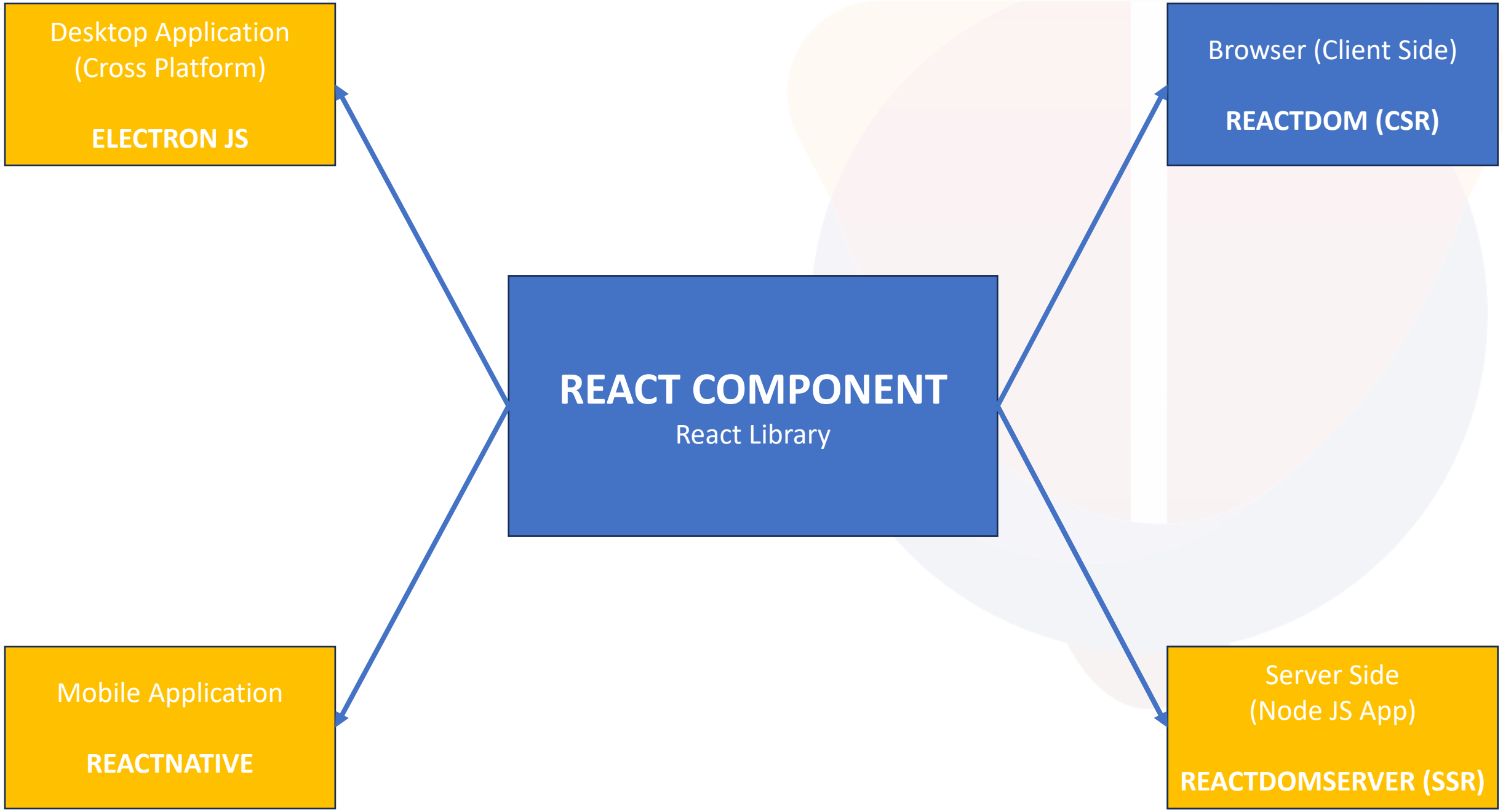
What is a JavaScript Library?

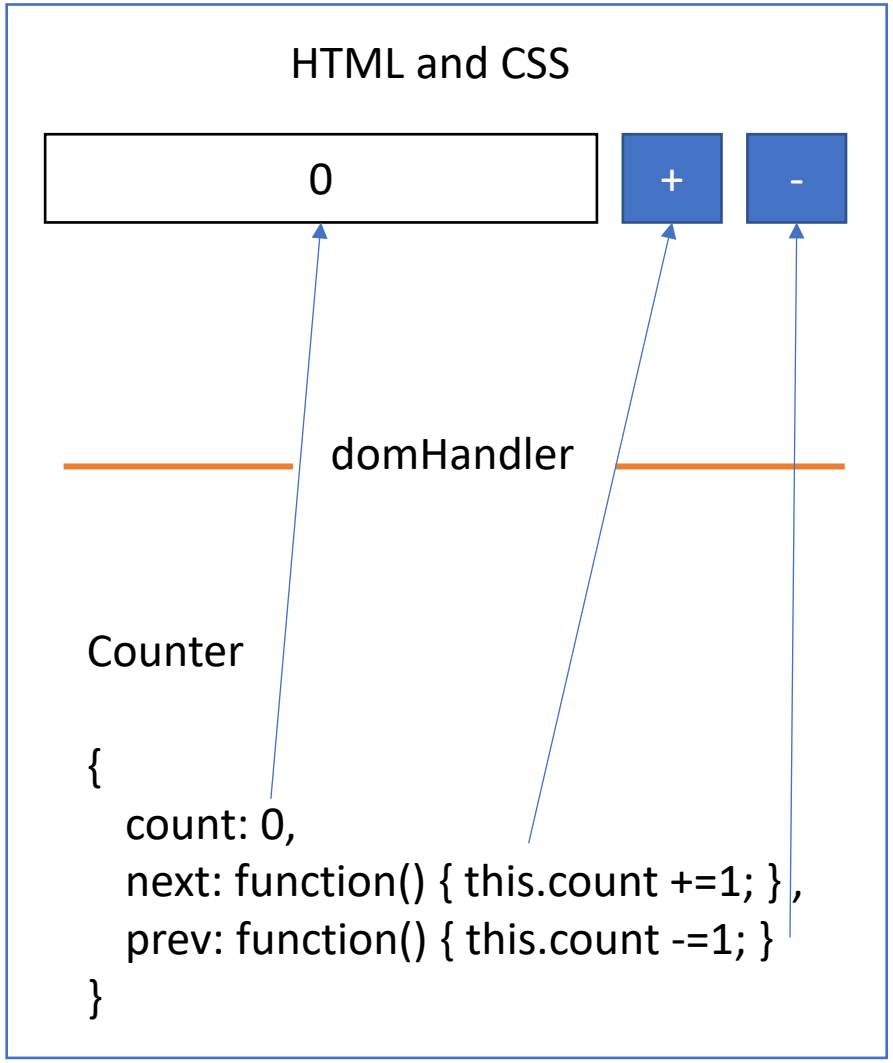
- JavaScript libraries are collections of prewritten code snippets that can be used and reused to perform common JavaScript functions.
- A particular JavaScript library code can be plugged into the rest of your project’s code on an as-needed basis.
- This led to faster development and fewer vulnerabilities to have errors.

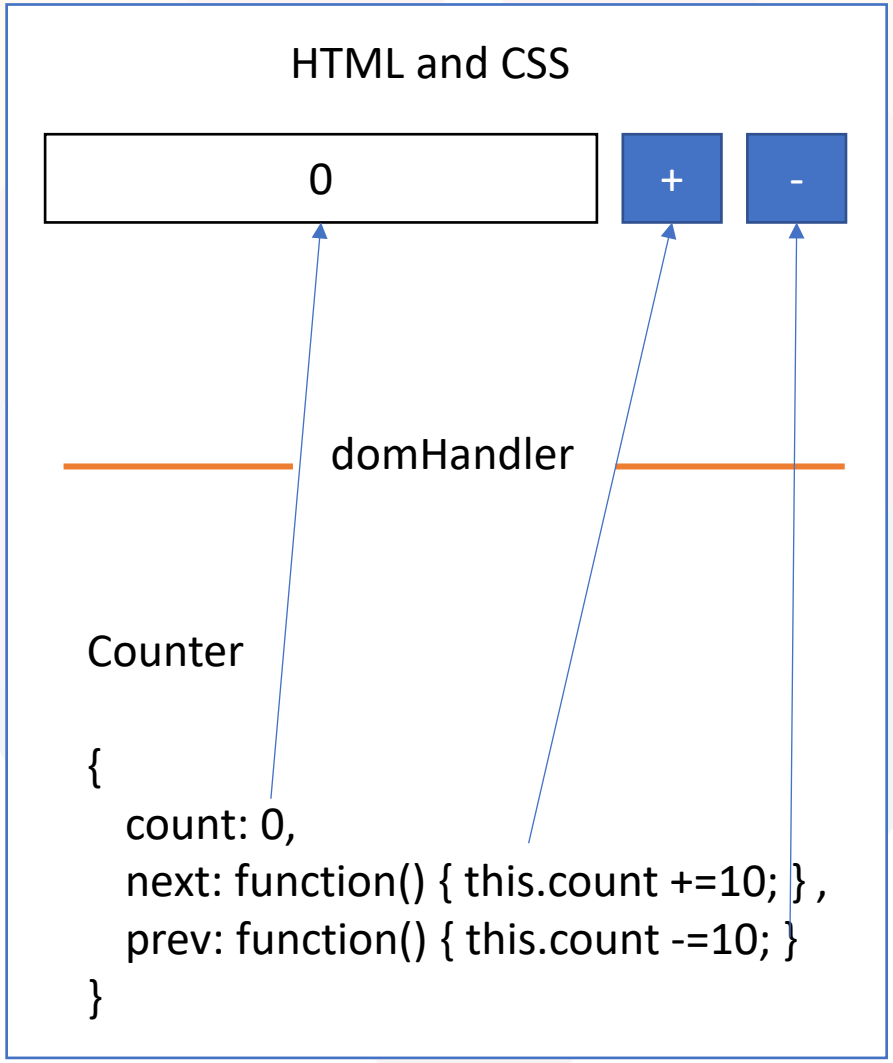

Star History
angular/angular
vuejs/vue
facebook/react
200.0k
150.0k
100.0k
50.0k
GitHub Stars
2014
2016
2018
2020
2022
2024
Date
star-history.com

React JS

- React JS is a popular JavaScript library for building user interfaces (UIs) for web applications.
- It was developed by Facebook and is now maintained by a community of developers.
- React JS allows developers to build reusable UI components that can be composed together to create complex, interactive web interfaces.
- One of the key features of React JS is its virtual DOM (Document Object Model) implementation, which allows for efficient updates to the UI without having to re-render the entire page. This results in faster performance and a better user experience.
- React JS also provides several other features and tools, such as a JSX syntax for writing HTML-like code in JavaScript, a component lifecycle for managing the state and behavior of components, and a powerful set of tools for debugging and testing.
- React JS has become a popular choice for web developers due to its flexibility, performance, and ease of use.
- It is often used in conjunction with other libraries and frameworks, such as Redux for state management, React Router for navigation, and Next.js for server-side rendering.


Desktop Application
(Cross Platform)
ELECTRON JS
Browser (Client Side)
REACTDOM (CSR)
REACT COMPONENT
React Library
Mobile Application
REACTNATIVE
Server Side
(Node JS App)
REACTDOMSERVER (SSR)





<Counter />

<Counter interval="10"/>

REACT COMPONENT

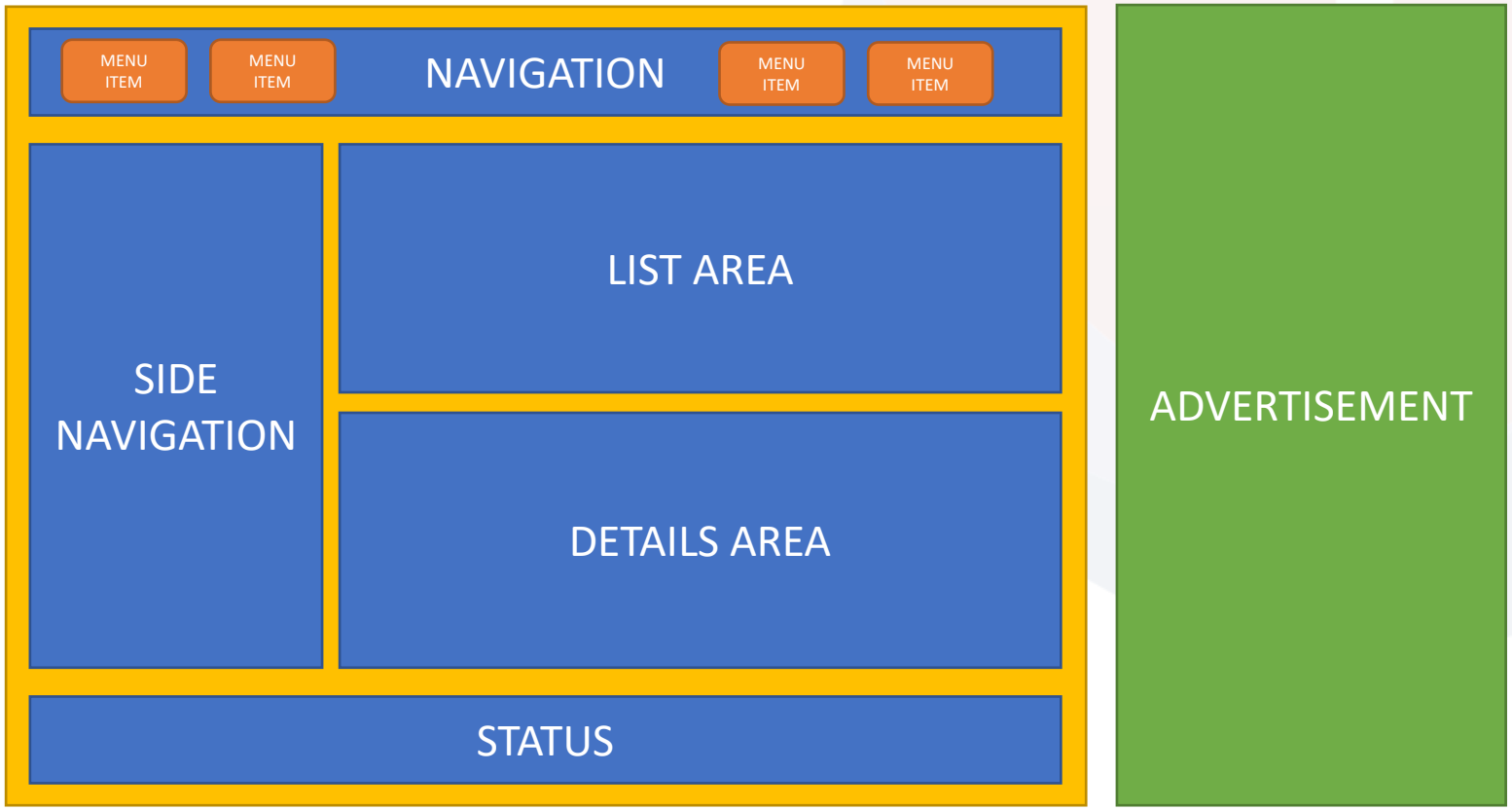
View* (UI) – JS / JSX

Style (CSS) – JSX / CSS

Data (Inside) – State
Data (Outside) – Props

Behaviour – Methods
(UI Interaction)

UI Composition (Composite UI Pattern)



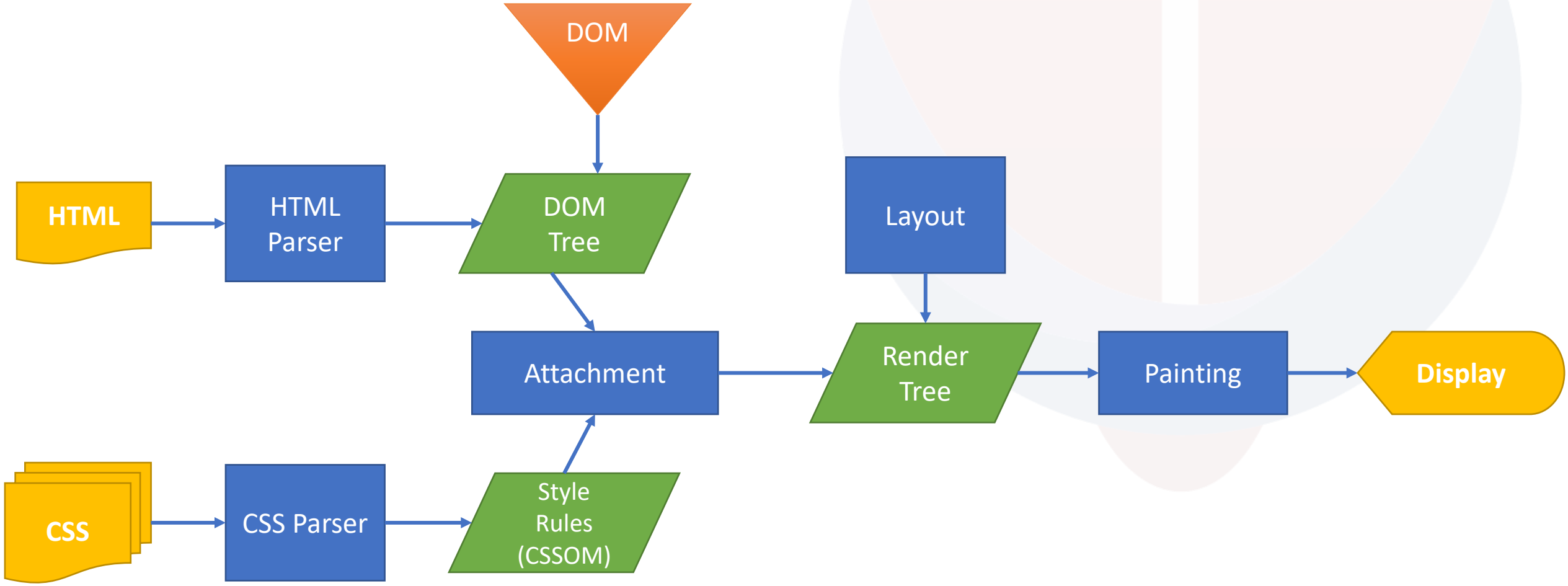
Installation

- Install Node LTS (v18) version - https://nodejs.org/en/
- Install / Update Visual Studio Code to Latest Release - https://code.visualstudio.com/
- Extensions of Visual Studio Code
  - AutoFileName
  - vscode-icons
  - JavaScript (ES6) code snippets
  - CSS Formatter
  - IntelliSense for CSS class names in HTML
  - Reactjs code snippets
- After Extensions are installed
  - File Menu -> Preferences -> Theme -> File Icon Theme -> Select VScode Icons

Browser Extensions Installation

- Install Google Chrome Extensions
  - React Developer Tools
    - https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi
  - Redux DevTools
    - https://chrome.google.com/webstore/detail/redux-devtools/lmhkpmbekcpmknklioeibfkpmmfibljd

How Browser Renders a Web Page?



Webpack Client-Side Build

Manual Configuration
- Learn Node JS
- Learn Webpack
- Learn Babel & Config.
- Learn Polyfill

Auto Configuration
- create-react-app (CLI)

react-scripts

Build & Deploy your Project on Local Server
npm start
WEBPACK-DEV-SERVER

Build your project
npm run build
WEBPACK CLI

WEBPACK

ES 2015 + (Multiple Files)

Babel +
Preset (React)
Preset (Env) +
Polyfill (core-js)

Browser Compatible Files (Multiple Files)

Bundling

Single File / Set of Files

BROWSER

Local HTTP Server

Deploy

HTML PAGE

Inject Bundles

React toolchains

- Use an integrated toolchain for the best user and developer experience.
- React toolchains help with tasks like:
  - Scaling to many files and components.
  - Using third-party libraries from npm.
  - Detecting common mistakes early.
  - Live-editing CSS and JS in development.
  - Optimizing the output for production.

Toolchains

- The React team primarily recommends these solutions:
- If you're learning **React or creating a new single-page app**, use **Create React App**.
- If you're building a **server-rendered website with Node.js**, try **Next.js**.
- If you're building a **static content-oriented website**, try **Gatsby**.

Create React App

- Create React App is a comfortable environment for learning React and is the best way to start building a new single-page application in React.
- It sets up your development environment so that you can use the latest JavaScript features, provides a nice developer experience, and optimizes your app for production.
- You'll need to have Node >= 14.0.0 and npm >= 5.6 on your machine.
- Create React App doesn't handle backend logic or databases; it just creates a frontend build pipeline, so you can use it with any backend you want. Under the hood, it uses Babel and webpack, but you don't need to know anything about them.
- When you're ready to deploy to production, running npm run build will create an optimized build of your app in the build folder.

Using React CLI

- To Create a new application
  - npx create-react-app first-app
  - cd first-app
  - Commands available in the application
    - npm start
      - Starts the development server.
    - npm run build
      - Bundles the app into static files for production.
    - npm run test
      - Starts the test runner.
    - npm run eject
      - Removes this tool and copies build dependencies, configuration files and scripts into the app directory. If you do this, you can’t go back!

Using Bootstrap

- Install
  - npm i bootstrap
  - npm i bootstrap-icons

Or

  - npm i bootstrap bootstrap-icons
- Include bootstrap and bootstrap-icons CSS file in index.js
- Include bootstrap module in index.js

Assignment

- Create and render two React components with the following specifications:
  - Component One:
    - Display the text "Hello from Component One".
    - Apply the Bootstrap text-success class to style the text.
  - Component Two:
    - Display the text "Hello from Component Two".
    - Apply the Bootstrap text-primary class to style the tex

Render Both Components Using a Single ReactDOM.render

- Key Points:
  - Both ComponentOne and ComponentTwo are rendered directly within a div element.
  - There is no parent component that can manage their state or facilitate communication between them.
  - If ComponentOne needs to communicate with ComponentTwo (e.g., share state), it becomes challenging because there is no shared parent to handle this.

Render both Components using two different ReactDOM.render

- Key Points
  - Each component is rendered separately in different DOM elements.
  - Two separate ReactDOM.render calls are made, each targeting a different root element in the HTML.
  - Useful if you need to render components in different parts of the HTML document.
  - Allows for isolated updates if the components don't need to interact with each other.

Render Both Components in Root and Render Root Using ReactDOM.render

- Key Points:
  - Both ComponentOne and ComponentTwo are rendered inside an App component.
  - The App or Root component serves as a parent component that can manage state and facilitate communication between ComponentOne and ComponentTwo.
  - The App or Root component can pass props or use context to manage and share state between its children.

Ways to Style React Components

- There are different ways to styling React JS components. Few of them are:
  - Inline Styles
    - In inline styling basically we create objects of style. And render it inside the components in style attribute using the React technique to incorporate JavaScript variable inside the JSX (Using '{ }')
  - Normal (External) CSS
    - In the external CSS styling technique, we basically create an external CSS file for each component and do the required styling of classes.
    - And use those class names inside the component.
    - It is a convention that name of the external CSS file same as the name of the component with '.css' extension.
    - It is better if the name of the classes used, follow the format 'componentName-context' (here context signifies where we use this classname).
  - CSS module
    - A CSS module is a simple CSS file but a key difference is by default when it is imported every class name and animation inside the CSS module is scoped locally to the component that is importing it also CSS file name should follow the format 'filename.module.css'.
    - This allows us to use a valid name for CSS classes without worrying about conflicts with other class names in your application.
  - Styled Components
    - The styled-components allows us to style the CSS under the variable created in JavaScript.
    - Style components is a third-party package using which we can create a component as a JavaScript variable that is already styled with CSS code and used that styled component in our main component.
    - styled-components allow us to create custom reusable components which can be less of a hassle to maintain.

State

- The state is an updatable structure that is used to contain data or information about the component and can change over time.
- The change in state can happen as a response to user action or system event.
- It is the heart of the react component which determines the behavior of the component and how it will render.
- It represents the component's local state or information.
- It can only be accessed or modified inside the component or by the component directly.

Props

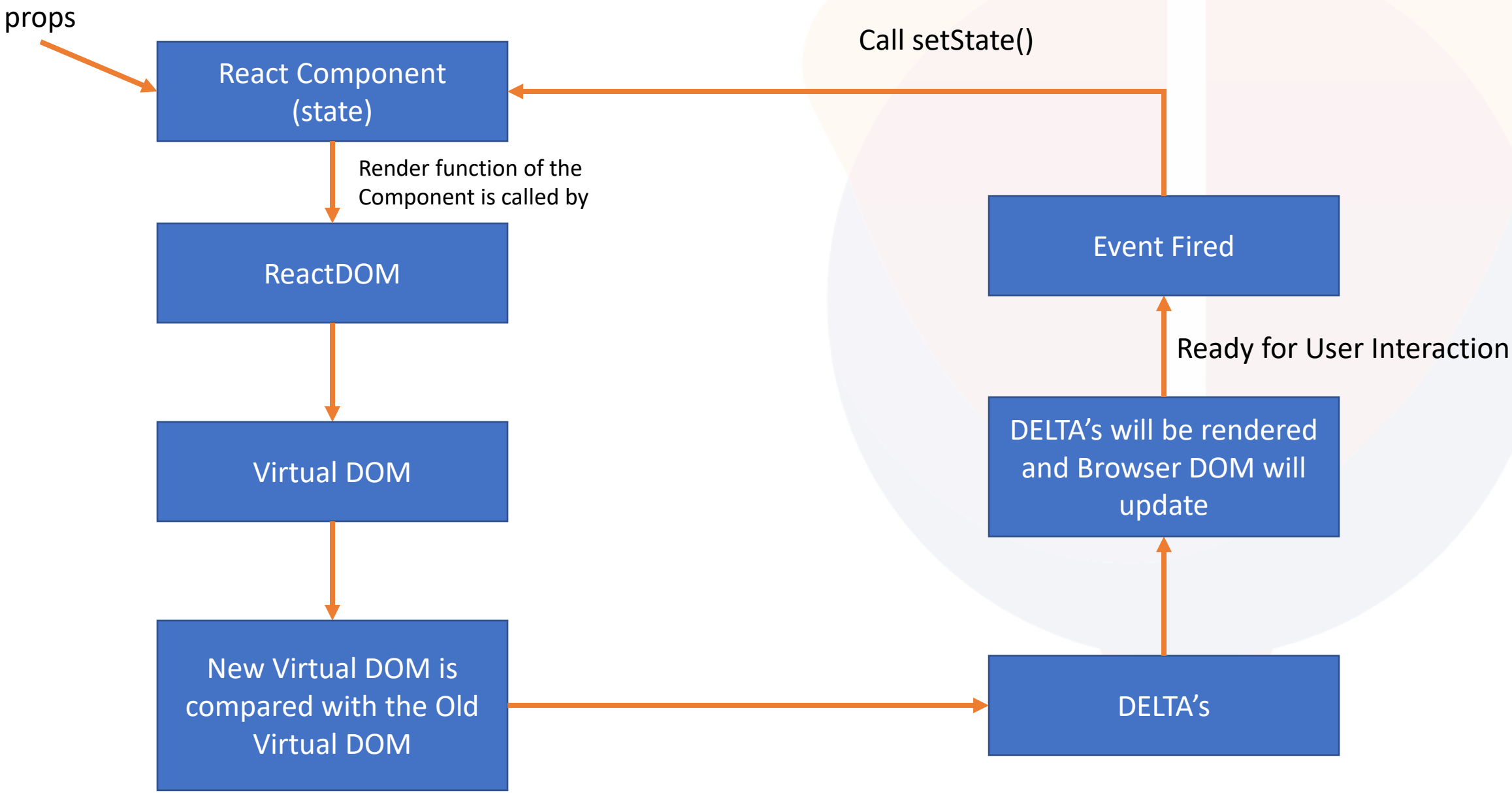
- **Read-Only Structure**: Props are immutable, meaning they cannot be modified once set. They are used to pass data from a parent component to a child component, similar to how attributes work in HTML.
- **Object Storage**: Props are stored as an object, holding the values of the attributes assigned to a component.
- **Data Passing**: Props facilitate the flow of data from one component to another, enabling components to communicate and share information.
- **Immutability**: Since props are immutable, any attempt to modify them inside the component will not work. This immutability helps maintain the predictability of the component's behavior.

Props vs State

Props	State
Props are read-only.	State changes can be asynchronous.
Props are immutable.	State is mutable.
Props allow you to pass data from one component to other components as an argument.	State holds information about the components.
Props can be accessed by the child component.	State cannot be accessed by child components.
Props are used to communicate between components.	States can be used for rendering dynamic changes within the component.
Props make components reusable.	State cannot make components reusable.
Props are external and controlled by whatever renders the component.	The State is internal and controlled by the React Component itself.

Type checking With PropTypes

- As your app grows, you can catch a lot of bugs with typechecking.
- To run typechecking on the props for a component, you can assign the special propTypes property.
- React.PropTypes has moved into a different package since React v15.5.
- Use prop-types library
  - npm i prop-types
- PropTypes exports a range of validators that can be used to make sure the data you receive is valid.


props
React Component (state)
Call setState()
Render function of the Component is called by
ReactDOM
Event Fired
Ready for User Interaction
Virtual DOM
DELTA's will be rendered and Browser DOM will update
New Virtual DOM is compared with the Old Virtual DOM
DELTA's

Virtual DOM

- Virtual DOM is an in-memory representation of real DOM.
- It is a lightweight JavaScript object which is a copy of Real DOM.
- Whenever setState() method is called, ReactJS creates the whole Virtual DOM from scratch.
- At any given time, ReactJS maintains two virtual DOM, one with the updated state Virtual DOM and other with the previous state Virtual DOM.
- ReactJS using diff algorithm compares both the Virtual DOM to find the minimum number of steps to update the Real DOM.

Virtual DOM

- Updating virtual DOM in ReactJS is faster because ReactJS uses
  - Efficient diff algorithm
  - Batched update operations
  - Efficient update of subtree only
  - Uses observable instead of dirty checking to detect the change

Terminologies

- **Reconciliation** is the process of keeping 2 DOM Trees in sync by a library like ReactDOM. It is done by using **Reconciler** and a **Renderer**.
- **Reconciler** uses Diffing Algorithm to find differences between Current Tree and Work in Progress Tree and sends computed changes to the Renderer.
- The **Renderer** is the one that updates the app's UI. Different devices can have different Renderers while sharing the same Reconciler.
- In React 16, they created a new Reconciler from scratch which uses a new data structure called **fiber**. Hence it is called **Fiber Reconciler**. The main aim was to make the reconciler **asynchronous** and **smarter by executing work on the basis of priority**.

React Fiber

- **React Fiber** needs to be asynchronous by taking advantage of cooperative scheduling and should be able to do :
  - Pause work and come back to it later
  - Assign priority to different types of work
  - Reuse previously completed work
  - Abort work if it’s no longer needed

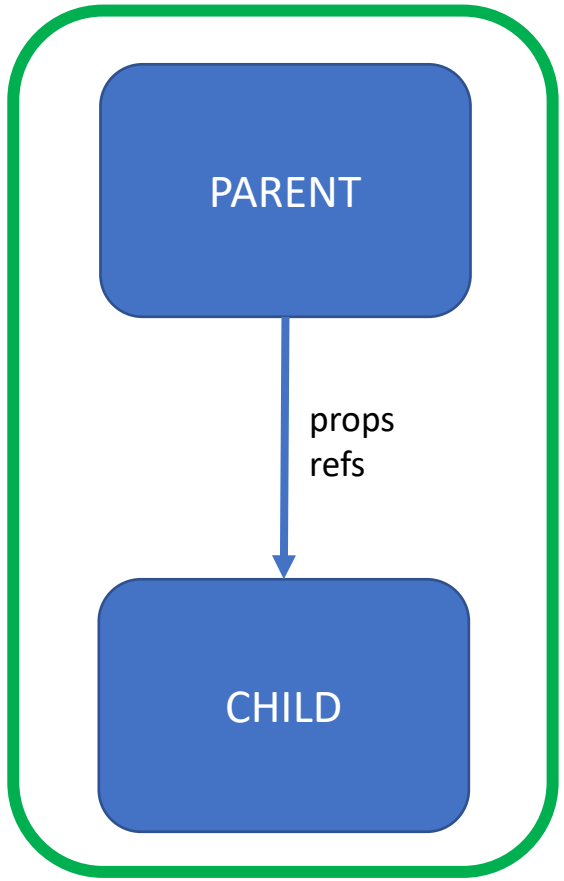
SyntheticEvent Object

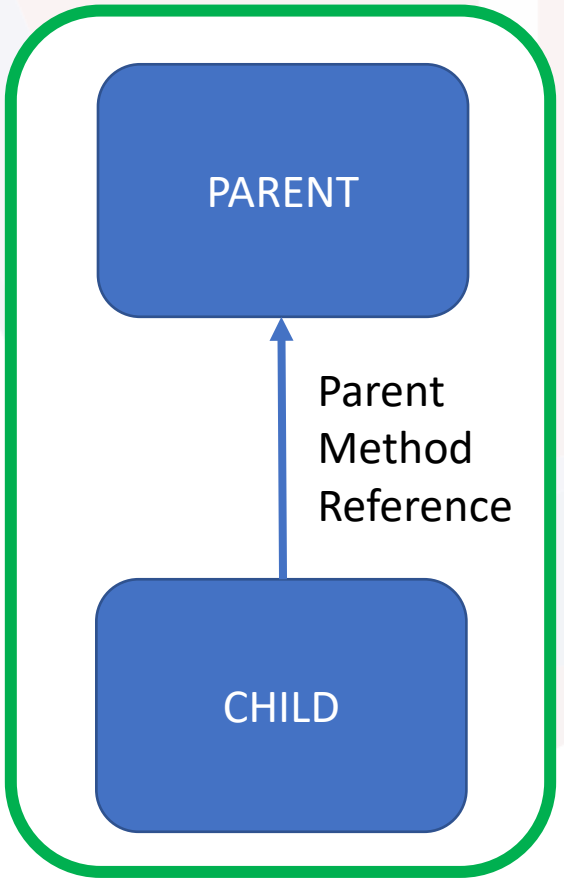
- Your event handlers will be passed instances of SyntheticEvent, a cross-browser wrapper around the browser’s native event.
- It has the same interface as the browser’s native event, including stopPropagation() and preventDefault(), except the events work identically across all browsers.
- The synthetic events are different from, and do not map directly to, the browser’s native events.

SyntheticEvent object attributes

- boolean bubbles
- boolean cancelable
- DOMEventTarget currentTarget
- boolean defaultPrevented
- number eventPhase
- boolean isTrusted
- DOMEvent nativeEvent
- void preventDefault()
- boolean isDefaultPrevented()
- void stopPropagation()
- boolean isPropagationStopped()
- void persist()
- DOMEventTarget target
- number timeStamp
- string type

Parent Child Communication





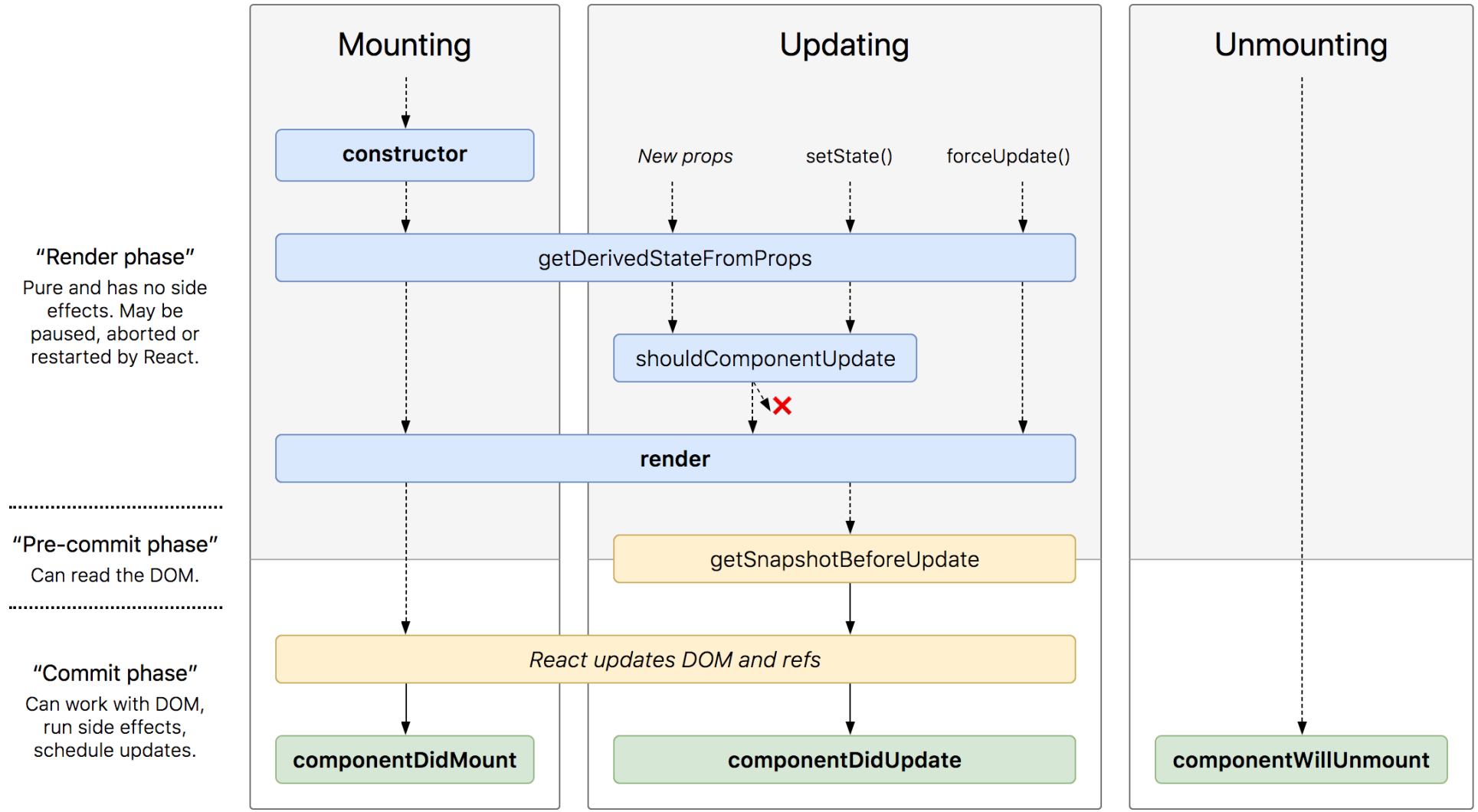
Parent Child Communication

- Parent to Child Communication:
  - Pass data as props from parent to child.
- Child to Parent Communication:
  - Pass a callback function from parent to child, which the child can call with data.
- Using Ref
  - Use useRef() to create a ref in a functional component or React.createRef() in a class component.
  - Use refs in parent for Accessing Child Methods
- Bidirectional Communication:
  - Combine both methods to enable communication in both directions.

Methods & Properties in a React Component Class

- Instance Methods
  - constructor()
  - render()
  - setState()
  - componentDidMount()
  - componentDidUpdate()
  - shouldComponentUpdate()
  - getSnapshotBeforeUpdate()
  - forceUpdate()
  - componentDidCatch()
  - componentWilUnmount()
- Instance Properties
  - state
  - props
  - context
- Class Methods (static)
  - getDerivedStateFromProps()
  - getDerivedStateFromError()
- Class Properties (static)
  - defaultProps
  - propTypes
  - displayName
  - contextType

Lifecycle Methods



Lifecycle Methods in a React Component Class

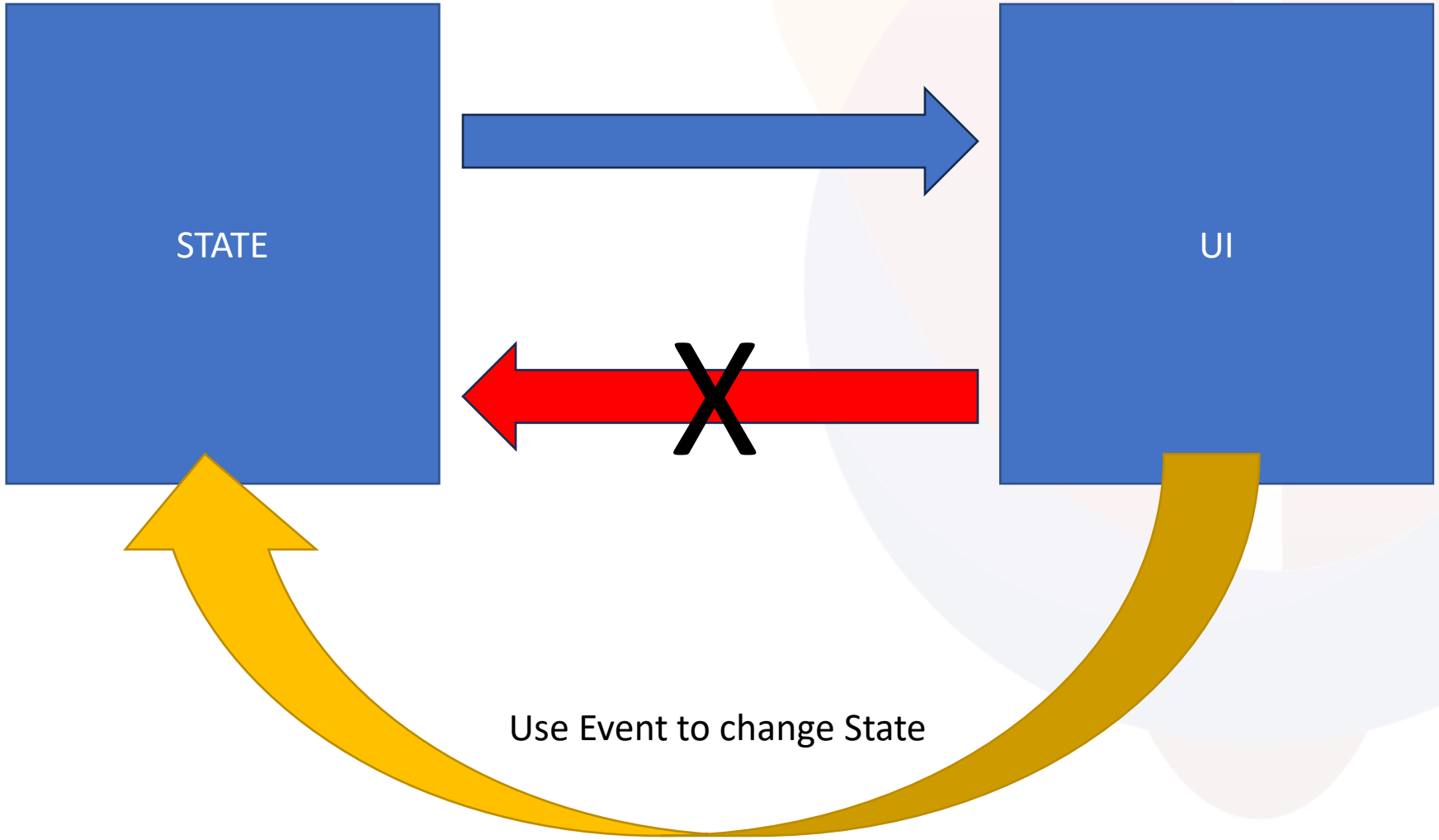
- Mounting Phase (Once)
  - constructor()
  - static getDerivedStateFromProps()
  - render()
  - componentDidMount()
- Updating Phase (On Changes)
  - static getDerivedStateFromProps()
  - shouldComponentUpdate()
  - render()
  - getSnapshotBeforeUpdate()
  - componentDidUpdate()
- Unmounting Phase (Once)
  - componentWillUnmount()
- Error Phase (Once)
  - componentDidCatch()
  - static getDerivedStateFromError()

Hooks

- Hooks are a feature introduced in React 16.8 that allow you to use state and other React features in functional components.
- Hooks provide a more direct API to the React concepts you already know, such as state, lifecycle, context, refs, and more.
- They enable you to write cleaner, more reusable code without the need for class components.

Built-in Hooks of React

- Basic Hooks:
  - useState: Lets you add state to functional components.
  - useEffect: Lets you perform side effects in functional components.
  - useContext: Lets you subscribe to React context without introducing nesting.
- Additional Hooks:
  - useReducer: An alternative to useState for more complex state logic.
  - useCallback: Returns a memoized callback.
  - useMemo: Returns a memoized value.
  - useRef: Returns a mutable ref object whose .current property is initialized to the passed argument.
  - useImperativeHandle: Customizes the instance value that is exposed when using ref.
  - useLayoutEffect: Similar to useEffect, but it fires synchronously after all DOM mutations.
  - useDebugValue: Can be used to display a label for custom hooks in React DevTools.
- Concurrent Mode Hooks (Experimental):
  - useTransition: Lets you update state without blocking the UI.
  - useDeferredValue: Returns a deferred version of the value that will defer updating until the next idle period.
  - useId: Generates unique IDs that are stable across the server and client.
  - useSyncExternalStore: Used for reading and subscribing to external data sources in a way that’s compatible with concurrent rendering features.

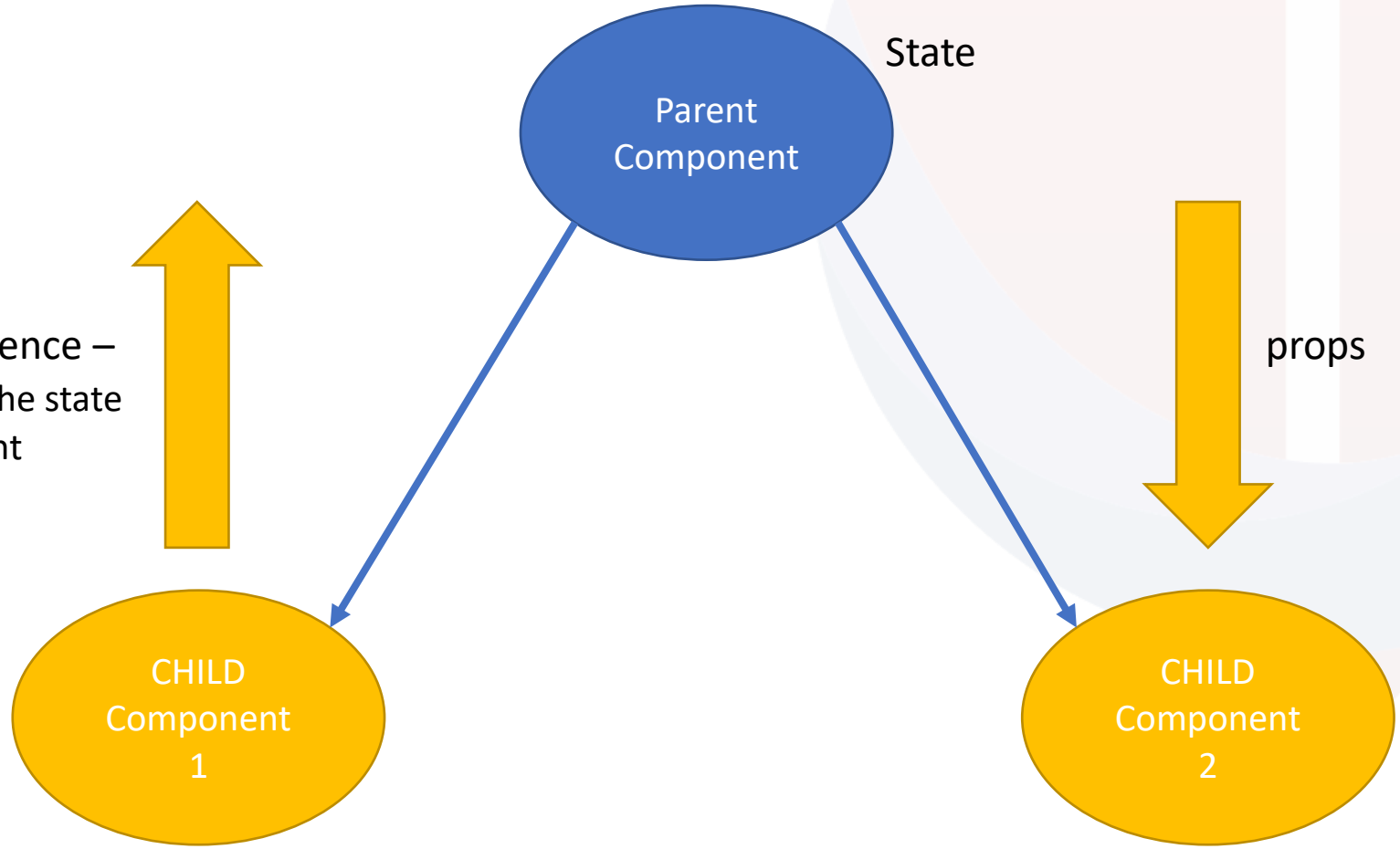

STATE
UI
Use Event to change State

Rules to remember

- The UI will update, only when setState() is called to modify the state.
- The state of a component, can be modified only by the method of the same component using setState().
- Never create a data in the state, if you are not using it on the UI.
- The data flows unidirectionally in React,
  - from state to the template of the component

    or
  - from parent to child.

Sibling Communication Using Parent State



Prop Drilling

- Prop drilling is a term used in React to describe the process where you pass data from a top-level component down to child components through multiple levels of intermediary components.
- This can lead to a scenario where intermediate components only serve as a conduit to pass props down to their children, without actually using those props themselves.

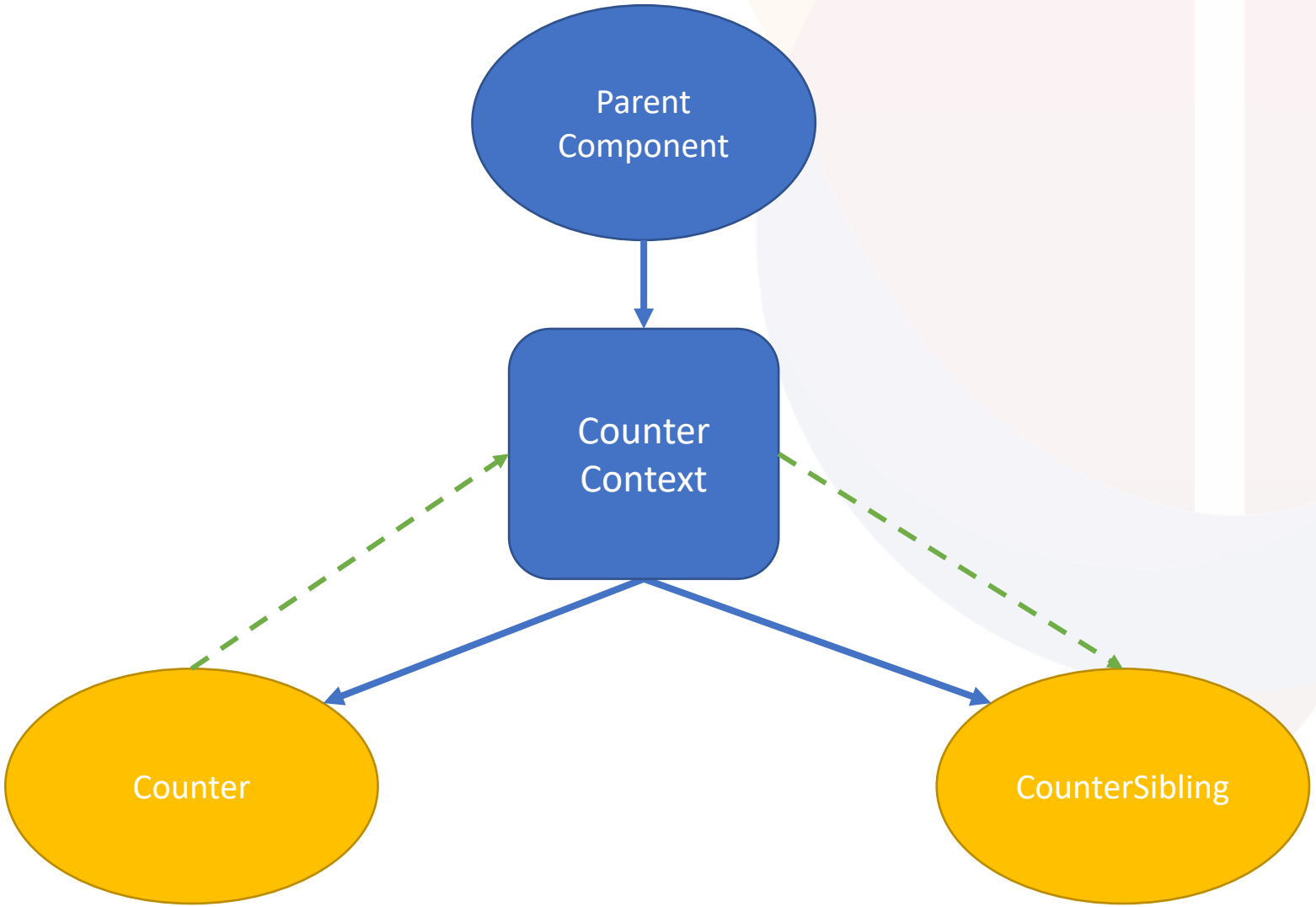
Context

- In a typical React application, data is passed top-down (parent to child) via props, but such usage can be cumbersome for certain types of props that are required by many components within an application.
- The Context API in React is a powerful feature that allows you to manage and share state across the entire component tree without having to pass props down manually at every level.
- It provides a way to pass data through the component tree by using a Provider and Consumer mechanism.

How the Context API Works

- Create a Context:
  - Use React.createContext to create a Context object.
- Provider:
  - Use the Provider component to make the context value available to all components within its subtree.
- Consumer:
  - Use the useContext hook or Context.Consumer component to access the context value in a component.

Counter Sibling Communication using Context



Controlled Component

- A controlled component is bound to a value, and its changes will be handled in code by using event-based callbacks.
- The input form element is handled by the react itself rather than the DOM.
- The mutable state is kept in the state property and will be updated only with setState() method.
- Controlled components have functions that govern the data passing into them on every onChange event occurs.
- The data is then saved to state and updated with setState() method. It makes component have better control over the form elements and data.

Uncontrolled Component

- It is similar to the traditional HTML form inputs. Here, the form data is handled by the DOM itself.
- It maintains their own state and will be updated when the input value changes.
- To write an uncontrolled component, there is no need to write an event handler for every state update.
- You can use a ref to access the value of the form from the DOM.

Controlled vs Uncontrolled

Controlled	Uncontrolled
It does not maintain its internal state.	It maintains its internal states.
Here, data is controlled by the parent component.	Here, data is controlled by the DOM itself.
It accepts its current value as a prop.	It uses a ref for their current values.
It allows validation control.	It does not allow validation control.
It has better control over the form elements and data.	It has limited control over the form elements and data.

Error Boundaries

- If the errors are not handled, the react component will crash and show a blank UI (in production mode).
- Error boundaries are React components that **catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI** instead of the component tree that crashed.
- Error boundaries catch errors during rendering, in lifecycle methods, and in constructors of the whole tree below them.
- Error boundaries do not catch errors for:
  - Event handlers
  - Asynchronous code (e.g. setTimeout or requestAnimationFrame callbacks)
  - Server side rendering
  - Errors thrown in the error boundary itself (rather than its children)

Create an Error Boundary

- A class component becomes an error boundary if it defines either (or both) of the lifecycle methods ***static getDerivedStateFromError()*** or ***componentDidCatch()***
- Use static getDerivedStateFromError() to render a fallback UI after an error has been thrown.
- Use componentDidCatch() to log error information.
- The granularity of error boundaries is up to you.
  - You may wrap top-level components to display a “Something went wrong” message to the user, just like how server-side frameworks often handle crashes.
  - You may also wrap individual components in an error boundary to protect them from crashing the rest of the application.

Array to UI Transformation

- Rendering arrays in React is a common task, typically handled with the map function.
- For simple arrays, you can use the map function to iterate over the array and render each item.
- When rendering an array of objects, it's better to use unique identifiers (such as IDs) as keys instead of indices.
- Best Practices
  - Use Unique Keys: Always provide a unique key for each item when rendering lists. Preferably, use a unique identifier from your data.
  - Avoid Using Indices as Keys: Using indices can lead to unexpected behavior, especially if the list changes dynamically.
  - Consistent Keys: Ensure that keys remain consistent between renders to allow React to manage updates efficiently.

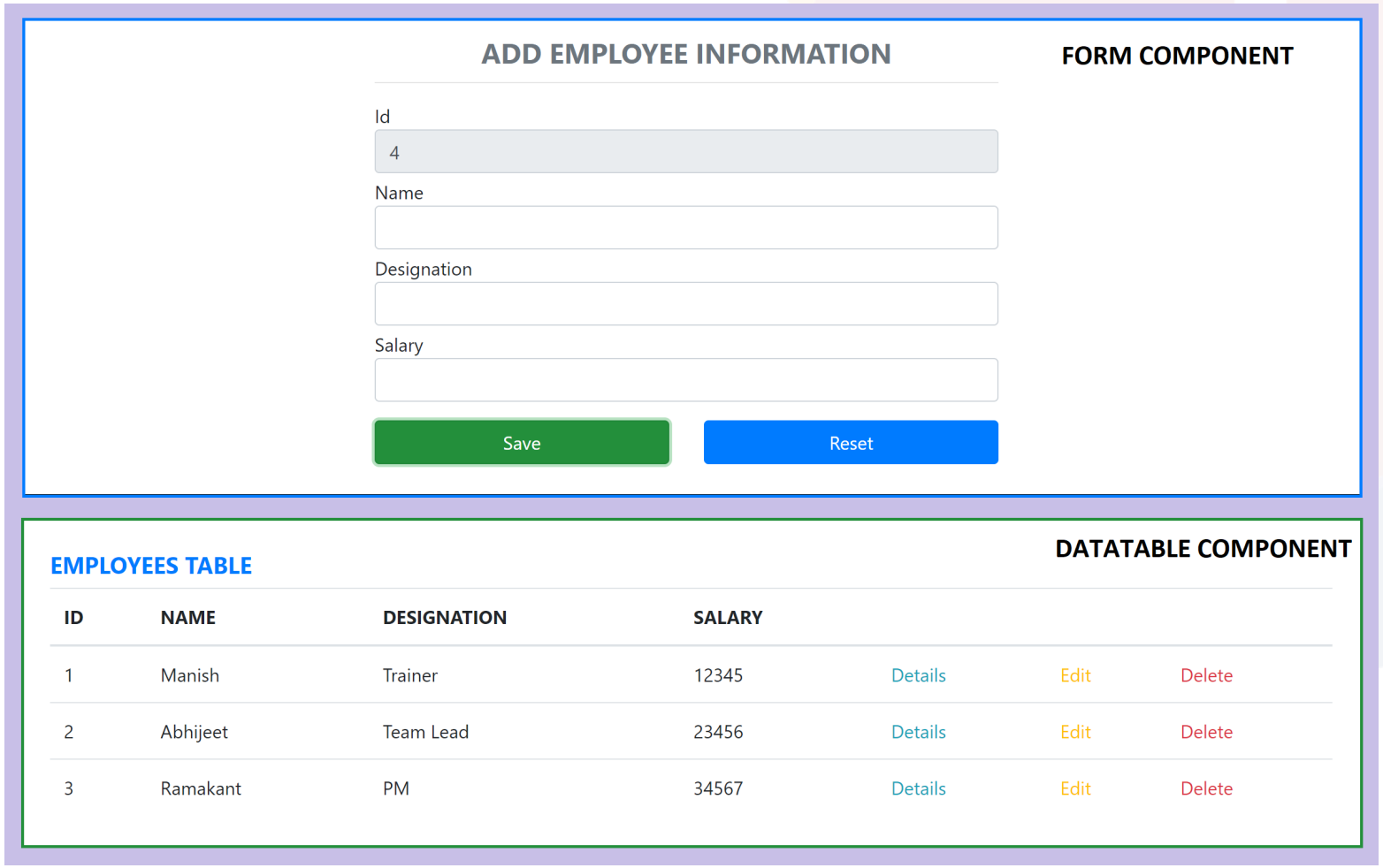
Array to UI Transformation - Why key Prop is Necessary

- Unique Identification: Keys help React identify which items have changed, are added, or are removed. This is particularly important when rendering dynamic lists of elements.
- Efficient Updates: When the state of a list changes, React uses the key prop to determine which elements need to be updated. By associating each element with a unique key, React can update only the elements that have changed, rather than re-rendering the entire list.
- Maintaining Element Order: Keys help maintain the order of elements. If keys are not used or are not unique, React may not be able to preserve the correct order of elements, leading to unexpected behavior in the UI.
- Preserving Component State: Keys help preserve the state of components. For example, if you have a list of input fields, changing the order of the elements without keys can result in the loss of user input because React cannot correctly associate the state of each input field with its corresponding element in the DOM.

Assignment

- Complete the CRUD assignment as per the screenshot shared in the next slide, use **array** to keep the data.
  - Create form-component and data-table-component as sibling components.
  - Id should be auto generated after every insert, update and delete
  - When we click details or edit, data will be displayed in Form Component.
  - In Details, Form should display the data, but the form will be disabled for editing.
  - On delete, ask for confirmation, before delete
- Second, Use Context API to Share Data between both the components.

Assignment


ADD EMPLOYEE INFORMATION
FORM COMPONENT
Id
4
Name
Designation
Salary
Save
Reset
EMPLOYEES TABLE
DATATABLE COMPONENT
ID NAME DESIGNATION SALARY
1 Manish Trainer 12345 Details Edit Delete
2 Abhijeet Team Lead 23456 Details Edit Delete
3 Ramakant PM 34567 Details Edit Delete

React-Bootstrap

- React-Bootstrap is a popular library that integrates Bootstrap, a widely-used front-end framework, with React.
- React-Bootstrap provides a collection of React components that are styled using Bootstrap, enabling developers to create responsive and visually appealing web applications with ease.
- React-Bootstrap replaces the Bootstrap JavaScript. Each component has been built from scratch as a true React component, without unneeded dependencies like jQuery.

Key Features of React-Bootstrap

- Pre-Built Components: React-Bootstrap offers a wide range of pre-built components such as buttons, forms, modals, navbars, and more, which are styled according to Bootstrap's design principles.
- React-Friendly: It is designed to work seamlessly with React, allowing developers to use Bootstrap's components and styling within a React application without having to use jQuery or additional libraries.
- Customizable: Components can be customized and extended as needed, providing flexibility to adapt the look and feel of the application.
- Theming: React-Bootstrap supports theming, enabling developers to easily switch between different visual themes or create their own custom themes.
- Accessibility: React-Bootstrap components are built with accessibility in mind, ensuring that applications are usable by people with disabilities.

React Hook Form

- React Hook Form is a library designed to manage form state and validation in React applications using React hooks.
- It simplifies form handling by reducing the amount of boilerplate code required and improving performance.
- React Hook Form embraces uncontrolled components and native HTML inputs, which makes it lightweight and fast.

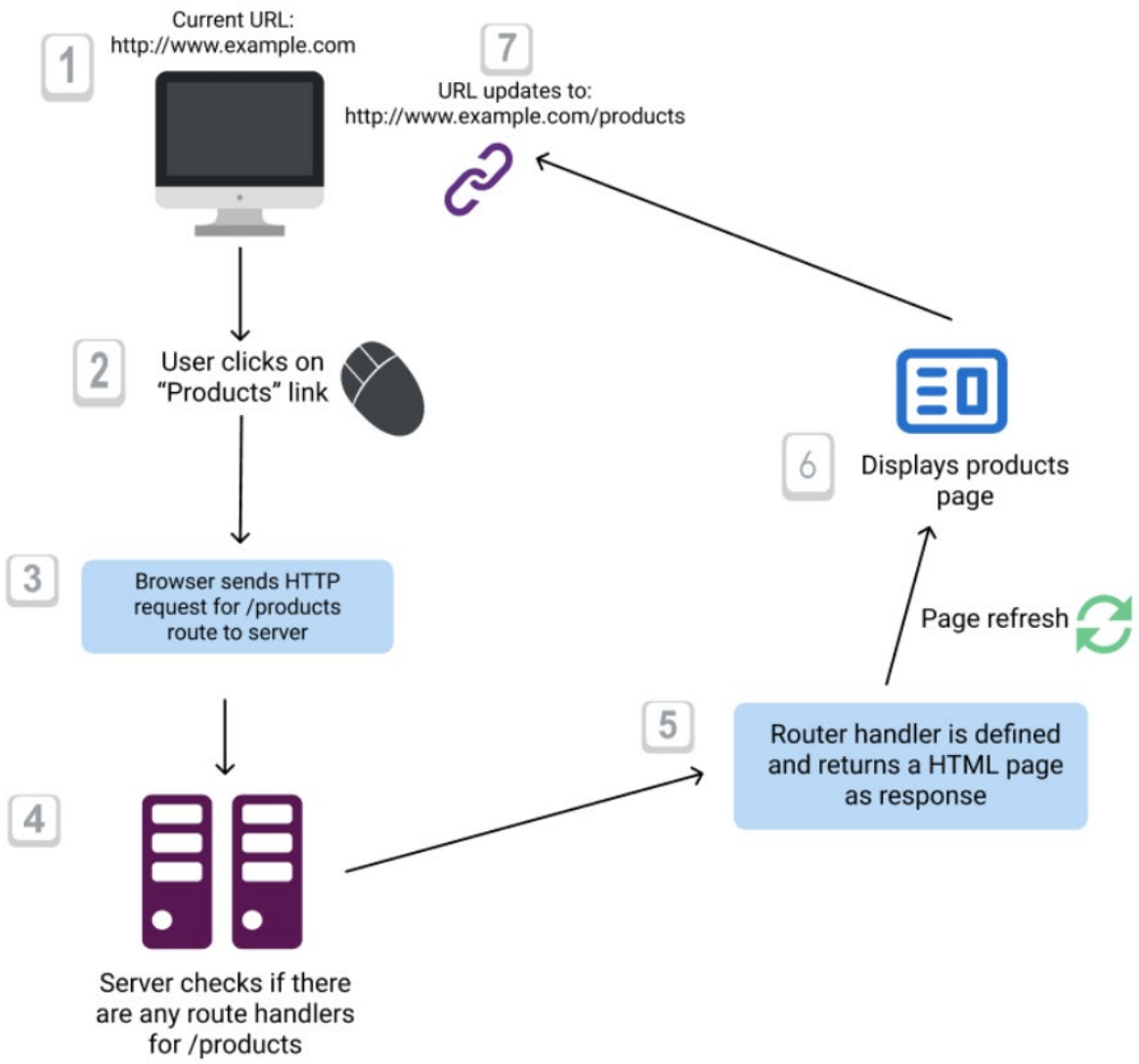
Key Features

- Performance: React Hook Form is built with performance in mind. It leverages uncontrolled components and minimizes re-renders, resulting in faster form interactions.
- Easy Integration: It integrates seamlessly with existing form validation libraries like Yup, Zod, Joi, and others.
- Minimal Re-renders: React Hook Form reduces the number of re-renders caused by form changes, improving the overall performance of your application.
- Simple API: It provides a simple and intuitive API for handling forms, making it easy to learn and use.
- Built-in Validation: The library offers built-in validation methods and supports custom validation.

AJAX and APIs

- You can use any AJAX library you like with React.
- Some popular libraries are:
  - Axios
  - jQuery AJAX
  - **window.fetch**
  - XMLHttpRequest
- You should populate data with AJAX calls in the commit phase lifecycle methods like componentDidMount().
- Use setState to update your component when the data is retrieved.

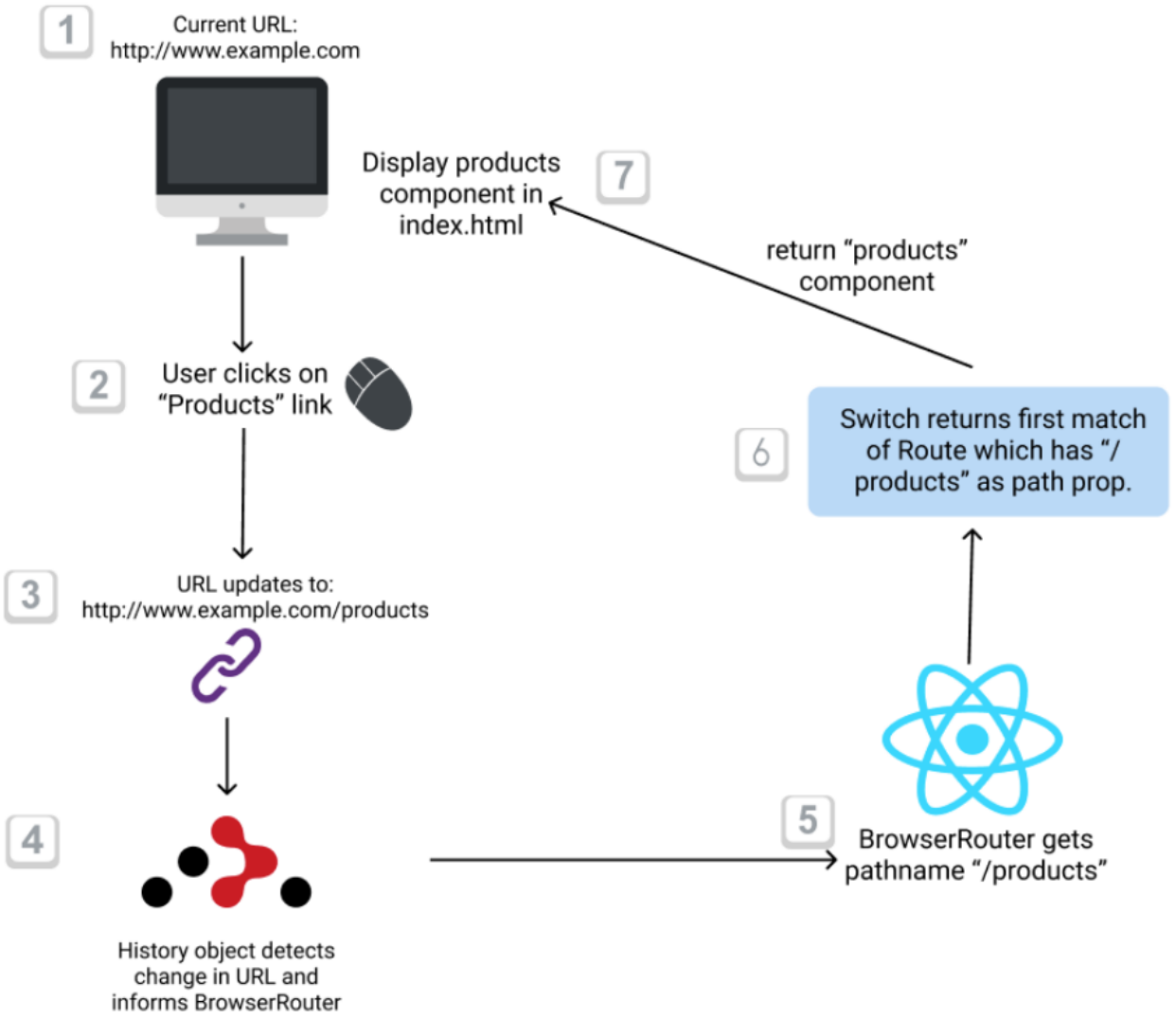
Conventional Routing

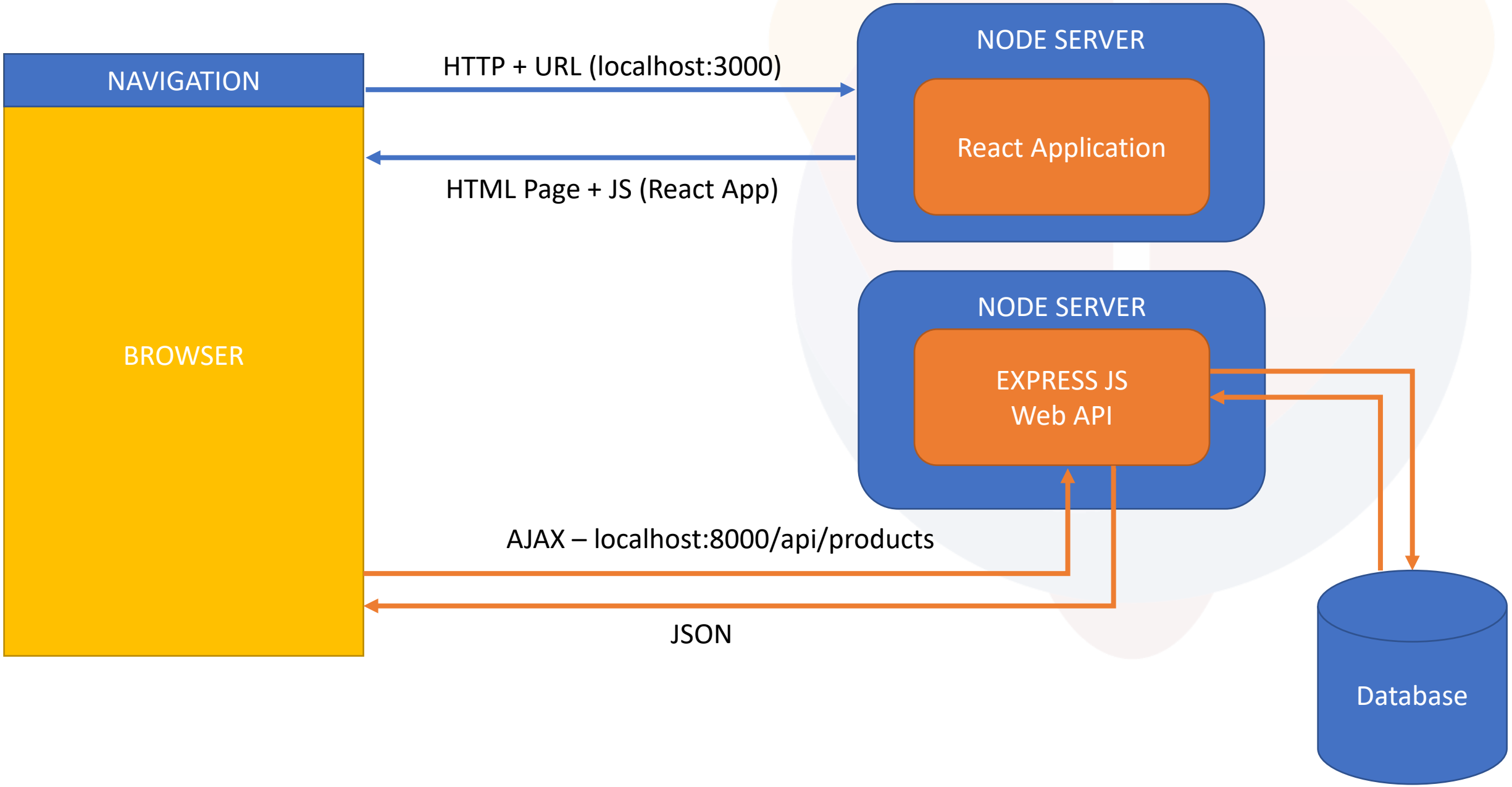


React Router

- React Router is the de-facto React routing library, and it’s one of the most popular projects built on top of React.
- React at its core is a very simple library, and it does not dictate anything about routing.
- Routing in a Single Page Application is the way to introduce some features to navigating the app through links, which are expected in normal web applications:
- The browser should change the URL when you navigate to a different screen
  - Deep linking should work: if you point the browser to a URL, the application should reconstruct the same view that was presented when the URL was generated.
  - The browser back (and forward) button should work like expected.
  - Routing links together your application navigation with the navigation features offered by the browser: the address bar and the navigation buttons.
- React Router offers a way to write your code so that it will show certain components of your app only if the route matches what you define.

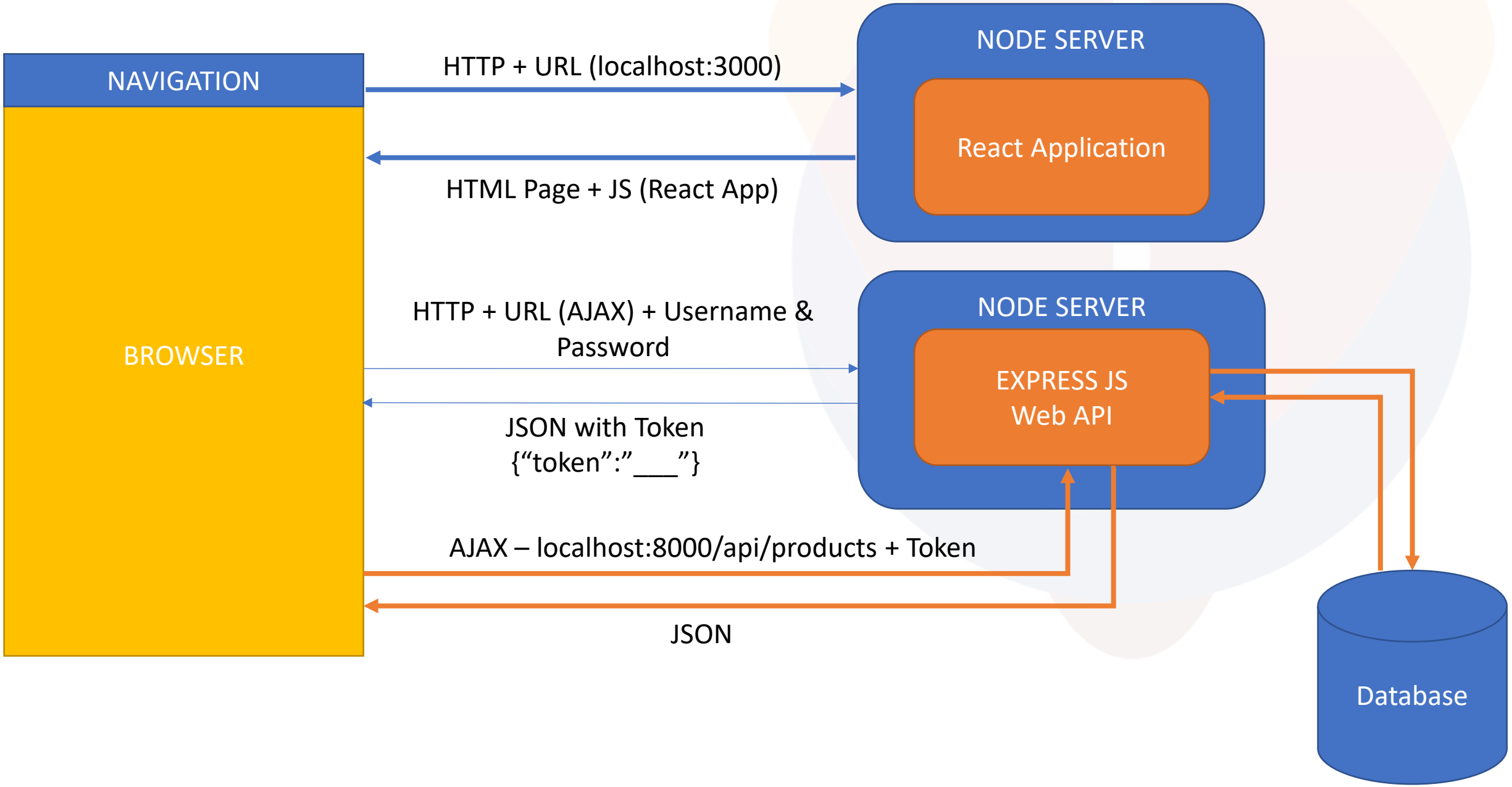
React Client-side Routing




NAVIGATION
BROWSER
HTTP + URL (localhost:3000)
HTML Page + JS (React App)
NODE SERVER
React Application
NODE SERVER
EXPRESS JS
Web API
AJAX – localhost:8000/api/products
JSON
Database

How is it working?

- User Clicks on Admin Link, React router maps the URL and displays the AdminComponent.
- On componentDidMount, Call to the API is done to access Data from the API Server.
- On success of the API call, the data is displayed in AdminComponent using DataTable Component.
- On error, AdminComponent shows error messages.

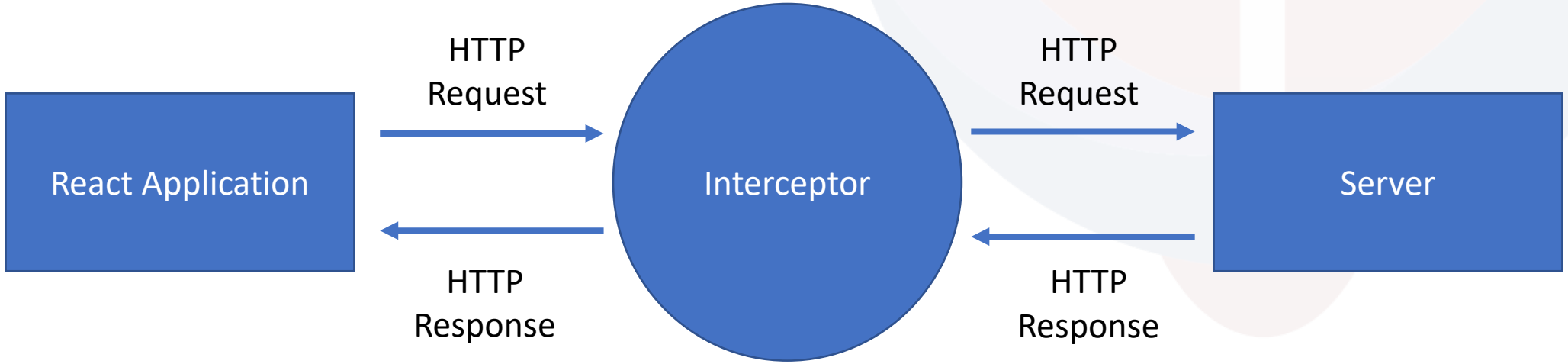

NAVIGATION
BROWSER
HTTP + URL (localhost:3000)
HTML Page + JS (React App)
NODE SERVER
React Application
HTTP + URL (AJAX) + Username & Password
JSON with Token
{“token”:”___”}
NODE SERVER
EXPRESS JS
Web API
AJAX – localhost:8000/api/products + Token
JSON
Database

Secure Application using Custom Route

- When user clicks the Admin Link, we need to verify that we have a token in session storage, and if the token in not there in the session storage, redirect the request to login component. (SecuredRoute)
  - SecuredRoute uses AuthenticatorClient's isAuthenticated to verify.
    - If false, the request is redirected to Login Component.
    - If true, the request is allowed for Admin Component.
- Logging and Token Management (AuthenticatorClient)
  - On Login Component, user must give username and password, which will be sent to Node API Server
  - If the username & password is correct, in the response; token will come from the Server
  - Received token will be stored in the local storage of the browser and isAuthenticated will be set to true.
- Reading and Attaching Token
  - Whenever the AJAX request is made for api, we must read and attach the token in the request header.
  - We use AuthenticatorClient getToken() to read the token from local storage.

Fetch Interceptor

- Interceptors are an excellent way to modify the HTTP request or response to make it more simple and understandable.
- fetch-intercept monkey patches the global fetch method and allows you the usage in Browser, Node and Webworker environments.
- https://www.npmjs.com/package/fetch-intercept



We can manipulate or modify
the request or response

Secure Application using Custom Route

- When user clicks the Admin Link, we need to verify that we have a token in session storage, and if the token in not there in the session storage, redirect the request to login component. (SecuredRoute)
  - SecuredRoute uses AuthenticatorClient's isAuthenticated to verify.
    - If false, the request is redirected to Login Component.
    - If true, the request is allowed for Admin Component.
- Logging and Token Management (AuthenticatorClient)
  - On Login Component, user must give username and password, which will be sent to Node API Server
  - If the username & password is correct, in the response; token will come from the Server
  - Received token will be stored in the session storage of the browser and isAuthenticated will be set to true.
- Reading and Attaching Token (Fetch Interceptor)
  - Whenever the AJAX request is made for api, we must read and attach the token in the request header.
  - We use AuthenticatorClient getToken() to read the token from session storage.

HOC (Higher Order Components)

- HOC is a pattern that emerges from React's compositional nature.
- A higher-order component (HOC) is an advanced technique in React for reusing component logic.
- Component transforms props into UI, **a higher-order component transforms a component into another component**.
- **A higher-order component is a function that takes a component and returns a new component**.

useReducer

- It is an alternative to the useState hook and is particularly useful when the state logic is complex and involves multiple sub-values or when the next state depends on the previous one.
- When to Use useReducer Over useState
  - When you have complex state logic that involves multiple sub-values.
  - When the next state depends on the previous state.
  - When you want to centralize state management logic for better readability and maintainability.
  - When you have a lot of state updates that need to be tracked and managed systematically.

Reducer Function

- A reducer function takes the current state and an action and returns a new state. It is a pure function, meaning it should not have side effects and should always produce the same output given the same input.

Benefits of Using useReducer

- Better State Management for Complex Logic: When state transitions are more complex and involve multiple sub-values, useReducer can help organize this logic more effectively than useState.
- Improved Readability: It separates state logic from component logic, making the component code more readable and easier to maintain.
- Predictable State Updates: Because reducer functions are pure, state updates become more predictable and easier to debug.
- Centralized State Logic: All state updates are handled in one place (the reducer function), making it easier to track changes and manage state transitions.
- Easier to Test: Reducer functions are pure functions, making them easier to test in isolation compared to component logic that might include side effects.
- Scalability: As your application grows, using useReducer can help manage more complex state transitions and make the codebase more scalable.

REACT COMPONENT → DISPATCHER (ACTION) → REDUCER → CONTEXT STATE →REACT COMPONENT

Questions

- When 2 or more components want to share data, where should we keep the data?
  - Parent Component
  - Context API (Provider & Consumer)
- How will you maintain data related to multiple subcomponents in an application?
  - Multiple Contexts
- How will you manage all the contexts in the application?

Redux

- Redux is a pattern and library for managing and updating application state, using events called "actions".
- It serves as a centralized store for state that needs to be used across your entire application, with rules ensuring that the state can only be updated in a predictable fashion.
- **The patterns and tools provided by Redux make it easier to understand when, where, why, and how the state in your application is being updated, and how your application logic will behave when those changes occur**.

When Should I Use Redux?

- Redux helps you deal with shared state management, but like any tool, it has tradeoffs. There are more concepts to learn, and more code to write. It also adds some indirection to your code and asks you to follow certain restrictions. It's a trade-off between short term and long-term productivity.
- Redux is more useful when:
  - You have large amounts of application state that are needed in many places in the app
  - The app state is updated frequently over time
  - The logic to update that state may be complex
  - The app has a medium or large-sized codebase, and might be worked on by many people

Not all apps need Redux. Take some time to think about the kind of app you're building and decide what tools would be best to help solve the problems you're working on.

Redux Terminology

- **Actions** - An action is a plain JavaScript object that has a type field. **You can think of an action as an event that describes something that happened in the application.**
- **Action Creators** - An action creator is a function that creates and returns an action object. We typically use these, so we don't have to write the action object by hand every time
- **Reducers** - A reducer is a function that receives the current state and an action object, decides how to update the state if necessary, and returns the new state. **You can think of a reducer as an event listener which handles events based on the received action (event) type.**
- **Store** - The current Redux application state lives in an object called the store . The store is created by passing in a reducer.
- **Dispatch** - The Redux store has a method called dispatch. The only way to update the state is to call store.dispatch() and pass in an action object. The store will run its reducer function and save the new state value inside. **You can think of dispatching actions as "triggering an event"** in the application.
- **Selectors** - Selectors are functions that know how to extract specific pieces of information from a store state value.

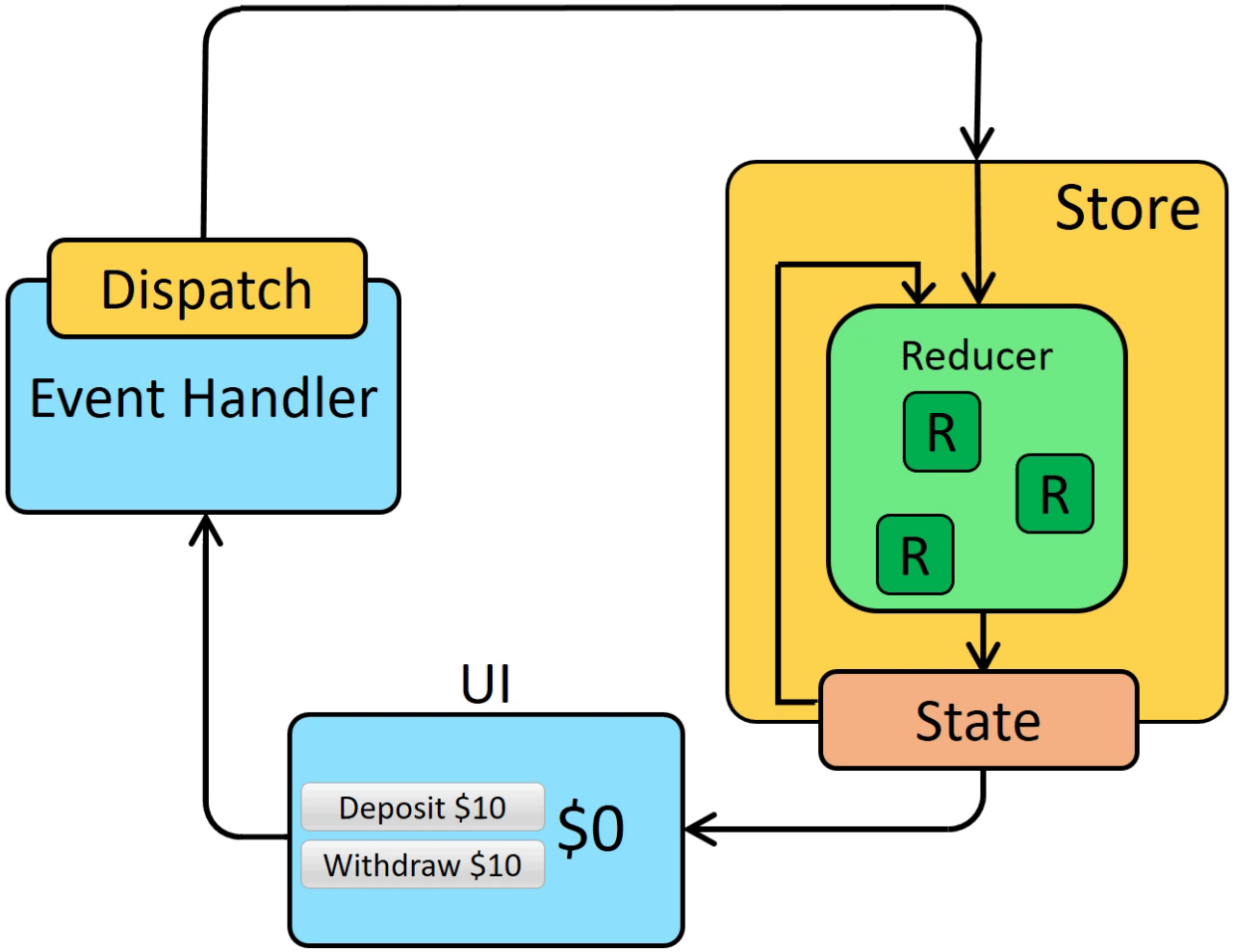
Redux Slice

- A modular piece of Redux state and its logic, created using the createSlice function from Redux Toolkit.
- createSlice function simplifies the process of managing state in a Redux application.
- It provides a way to define a slice of the Redux state along with the associated reducers and actions in a single, concise manner.

Key Concepts of Redux Slices

- State: The portion of the Redux state managed by the slice.
- Reducers: Functions that specify how the state changes in response to actions.
- Actions: Automatically generated action creators for each reducer function.
- Selectors: Functions that extract specific pieces of the state. Though not part of the slice directly, they are commonly defined alongside slices.

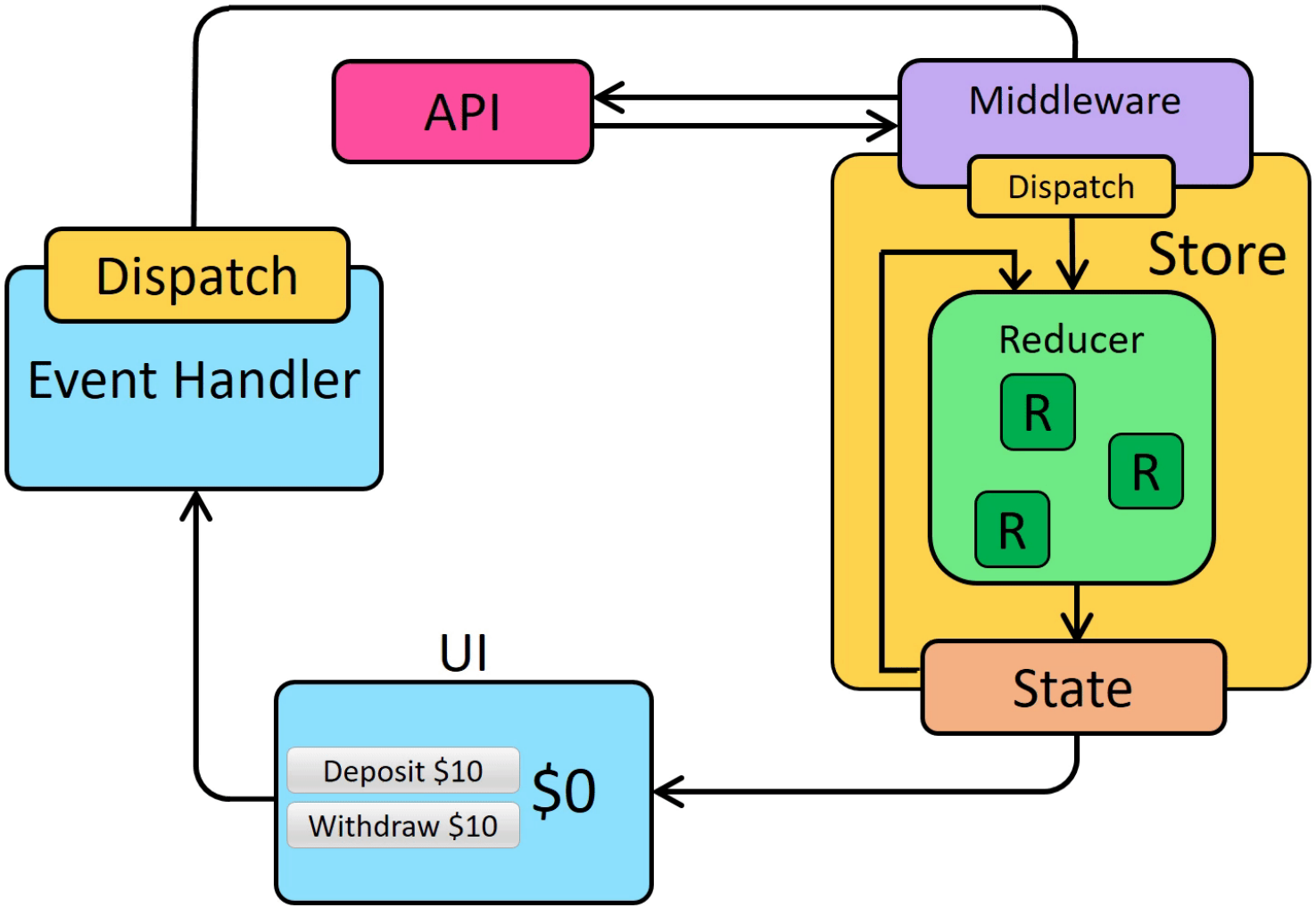
Redux Application Data Flow



Thunk Middleware – Async Data Flow

- Thunk is a logical concept in programming where you deal with a function that is primarily used to delay the calculation or evaluation of any operation.
- Redux Thunk acts as a middleware that will return you a function instead of an object while calling through the action creators.
- The returned function receives the dispatch method from the store and then later it is used to dispatch synchronously inside the body of function once the asynchronous actions have been completed.

Redux Application Async Data Flow



REACT COMPONENT → DISPATCHER (ACTION) → REDUCER → STORE →REACT COMPONENT

REACT COMPONENT → DISPATCHER (ACTION) → MIDDLEWARE → REDUCER → STORE →REACT COMPONENT

createAsyncThunk

- createAsyncThunk is a utility provided by Redux Toolkit that simplifies the process of handling asynchronous operations in a Redux application.
- It helps manage the state associated with these operations, such as loading, success, and error states, in a standardized way.

Benefits of createAsyncThunk

- Simplifies Async Logic: createAsyncThunk abstracts the complexity of writing async actions. It handles dispatching of pending, fulfilled, and rejected actions automatically.
- Automatic Action Types: It generates action types for each stage of the async operation (pending, fulfilled, rejected), reducing the need for manual action type creation.
- Built-in Loading and Error Handling: Automatically manages loading and error states, making it easier to update the UI based on the status of the async operation.
- Consistent Patterns: Provides a consistent pattern for handling asynchronous logic across the application, making the code more predictable and easier to maintain.
- Type Safety: Improves type safety in TypeScript projects by providing clear types for actions and payloads.
- Integration with Slices: Integrates seamlessly with Redux slices, allowing you to handle async logic directly within the slice's extraReducers.

Assignments

- Complete the CRUD Assignment Application (Day3) with Context API.
- Complete the CRUD Assignment Application (Day3) with Context API and Reducer.
- Use Redux with the same Application.
- Consume a secured API with Redux.

Links

- https://compat-table.github.io/compat-table/es6/
- https://create-react-app.dev/
- https://react-bootstrap.netlify.app/
- https://react-hook-form.com/get-started
- https://reactrouter.com/en/main
- https://redux-toolkit.js.org/

Further Learning

Do it only after you have completed the code and assignments and comfortable with what we covered in the training

Redux Saga Middleware

- Saga pattern is way a to handle long running transactions with side effects or potential failures.
- Redux-saga is a redux middleware library, that is designed to make handling side effects in your redux app nice and simple.
- It achieves this by leveraging an ES6 feature called Generators, allowing us to write asynchronous code that looks synchronous.

Thunk vs Saga

Thunk	Saga
Created by Redux Creator	Created by third party developers
Runs in any JavaScript Context	Only runs in ES6 environments that support yield
Has no built-in solution for asynchronous calls	Uses yield and generator functions to simplify async
No way to orchestrate side-effects between thunks	Uses effects and plain actions to coordinate sagas

React Hook Form

- React Hook Form reduces the amount of code you need to write while removing unnecessary re-renders.
- Features
  - Built with performance, UX and DX in mind
  - Embraces native HTML form validation
  - Out of the box integration with UI libraries
  - Small size and no dependencies
  - Support Yup, Zod, AJV, Superstruct, Joi, Vest, class-validator, io-ts, nope and custom build

React Component Libraries

- React Bootstrap
- React Material
- Kendo React
- Prime React
- Syncfusion React

TECHNIZER
Inspired by Impossible

TECHNIZER
EDGE
Smarter Systems, Sharper Teams

Contact Me

Manish Sharma | Gmail

Manish Sharma | WhatsApp

Manish Sharma | Facebook

Manish Sharma | LinkedIn