Revision, Part II

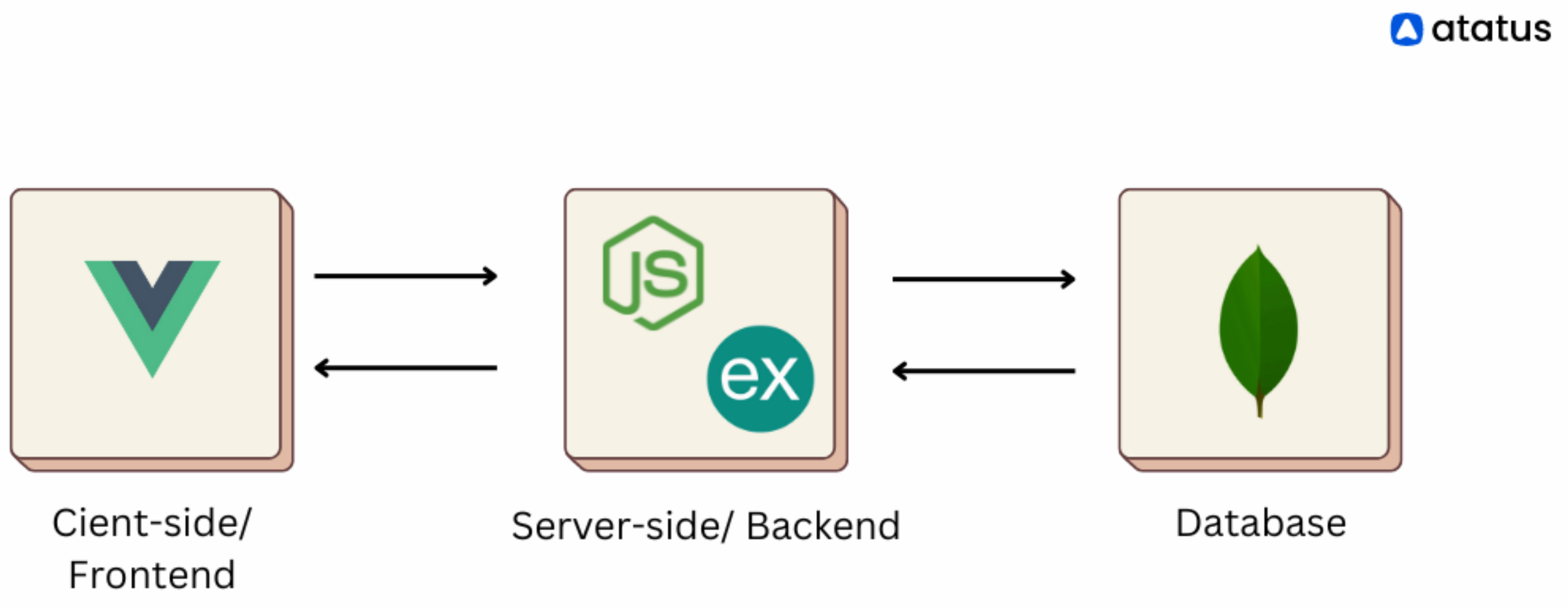
COMP7270 Web and Mobile Programming
& COMP7980 Dynamic Web and Mobile Programming

2rd April 2025, Dr Shichao MA

Vue.js

MEVN Stack

- M: MongoDB
- E: Express.js
- V: Vue.js
- N: Node.js

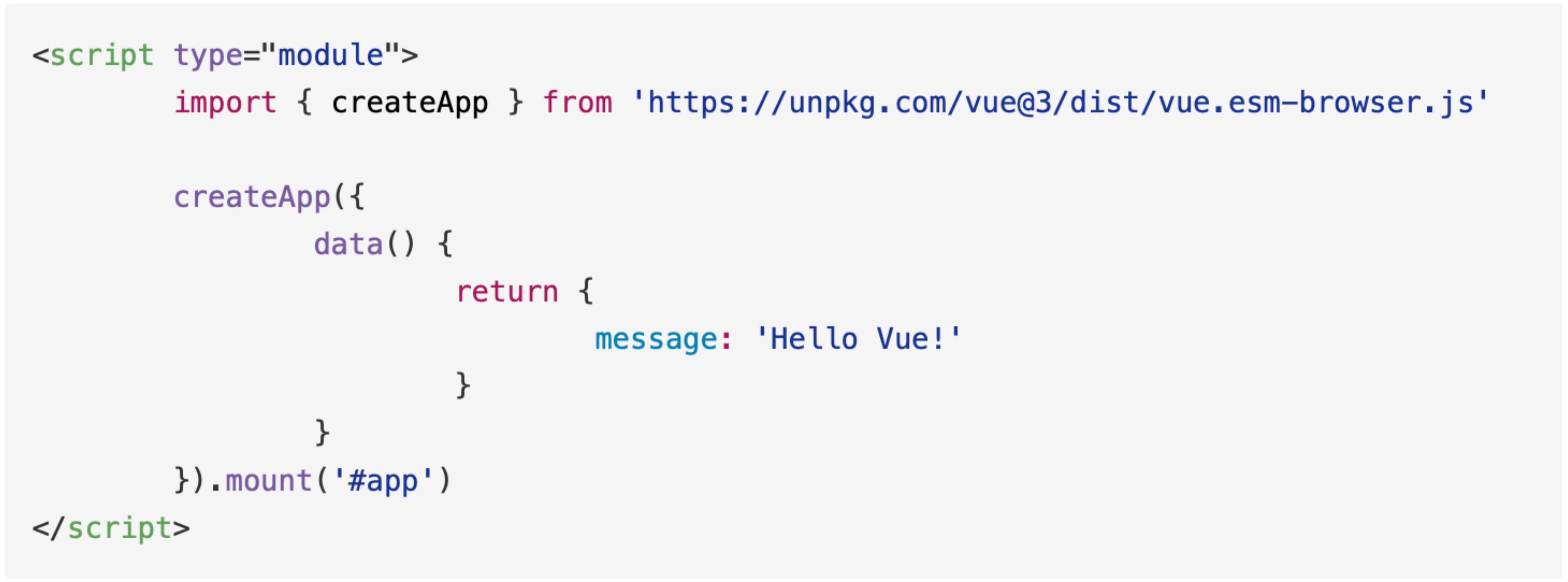


Vue.js Version 3

- Vue3 provides two APIs:
  - Go with Options API if you are not using build tools, or plan to use Vue primarily in low-complexity scenarios, e.g. progressive enhancement.
  - Go with **Composition API** + Single-File Components if you plan to build full applications with Vue.

CreateApp() & .mount()

For both APIs



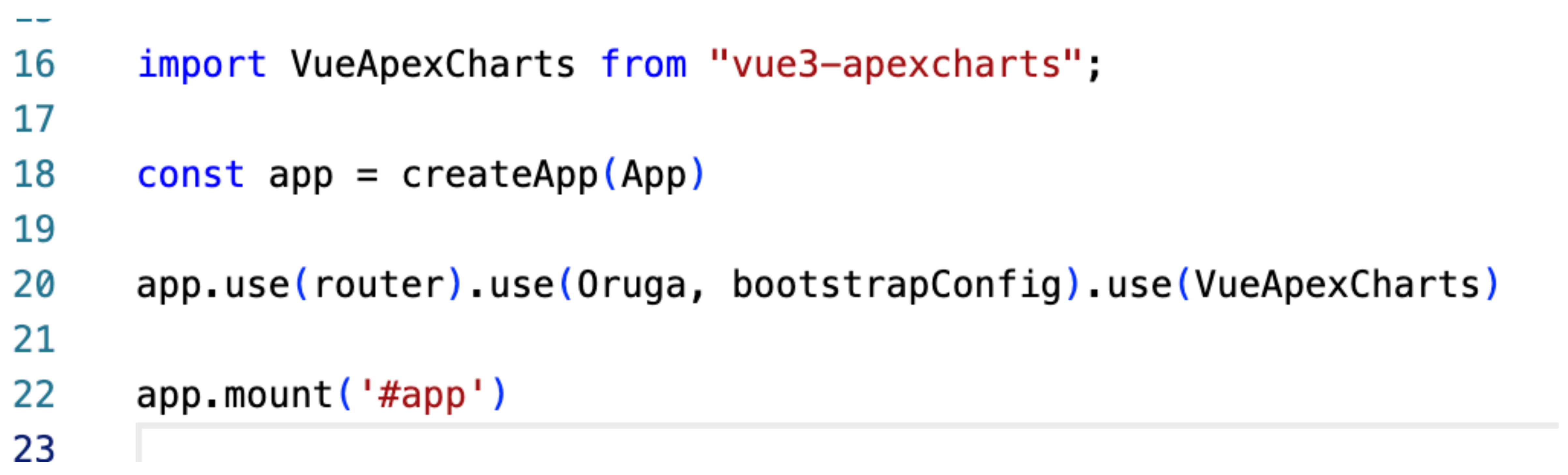
```
<script type="module">
        import { createApp } from 'https://unpkg.com/vue@3/dist/vue.esm-browser.js'

        createApp({
                data() {
                        return {
                                message: 'Hello Vue!'
                        }
                }
        }).mount('#app')
</script>
```

Options API

Composition API

In main.js



```
import VueApexCharts from "vue3-apexcharts";

const app = createApp(App)

app.use(router).use(Oruga, bootstrapConfig).use(VueApexCharts)

app.mount('#app')

```

Declarative Rendering

- At the core of Vue.js is a system that enables us to declaratively render data to the DOM using straightforward template syntax
- The most basic form of **data binding** is data interpolation and rendering using the “Mustache” syntax (double curly braces):

```
<div v-if="route.name == 'view-booking'">
    <h1>{{ booking.email }}</h1>
    <p>Number of Tickets: {{ booking.numTickets }}</p>
    <p>Payment Method: {{ booking.payment }}</p>
    <p>Favourite Team: {{ booking.team }}</p>
    <p>Favourite Hero: {{ booking.superhero }}</p>
    <p>Terms and Conditions: {{ booking.terms }}</p>
</div>
```

Directives

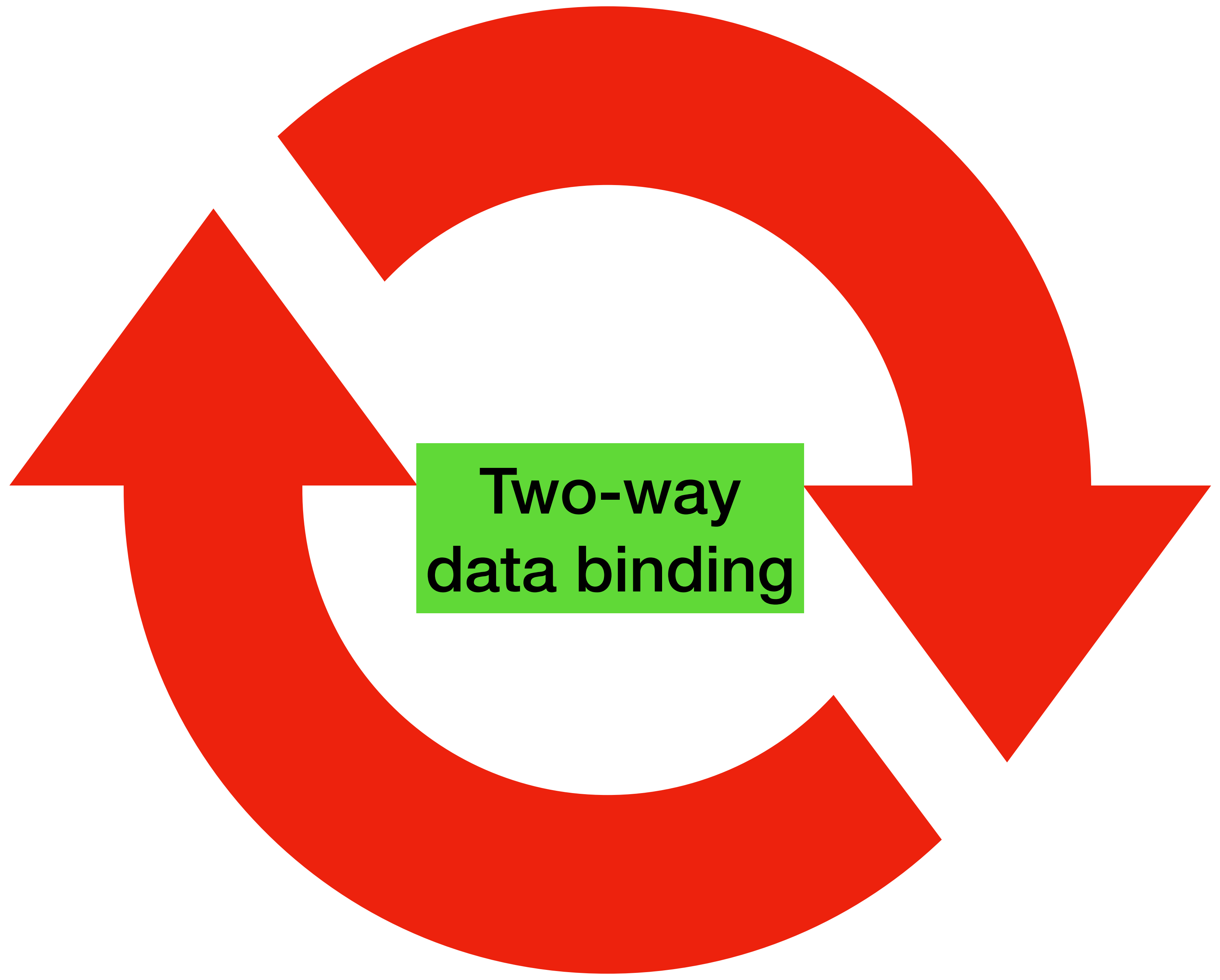
- At a high level, **directives are markers on a DOM element** that tell Vue's HTML compiler to attach a specified behavior to that DOM element, or even to transform the DOM element and its children.

- Vue.js comes with a set of these directives built-in, like v-if, v-for, v-on, and, v-model etc…

Form Input Bindings with v-model

- We can use the v-model directive to create **two-way data bindings** on form input and select elements, etc.

```
<input type="email" class="form-control" v-model="booking.email" required>
```



```
const booking = ref({
    email: '',
    numTickets: 1,
    payment: 'Credit Card',
    team: '',
    superhero: '',
    terms: false
})
```

Event Handling with v-on

- We can use the v-on directive to listen to DOM events and run some JavaScript when they're triggered.

- v-on accepts the name of an **instance method** that you want to call

```
<form class="container my-5" v-
on:submit.prevent="submitBooking" >
```

- v-on:submit could be shortened as @submit

```
<form class="container my-5"
@submit.prevent="submitBooking" >
```

```
const submitBooking = async function () {
    // post the booking to the backend
    const response = await fetch('/api/
bookings', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/
json'
        },
        body:
JSON.stringify(booking.value)
    });
    // convert the response to json
    const json = await response.json();
    // log the json
    console.log(json);
    // alert the user
    alert(JSON.stringify(json));
}
```

Computed Property

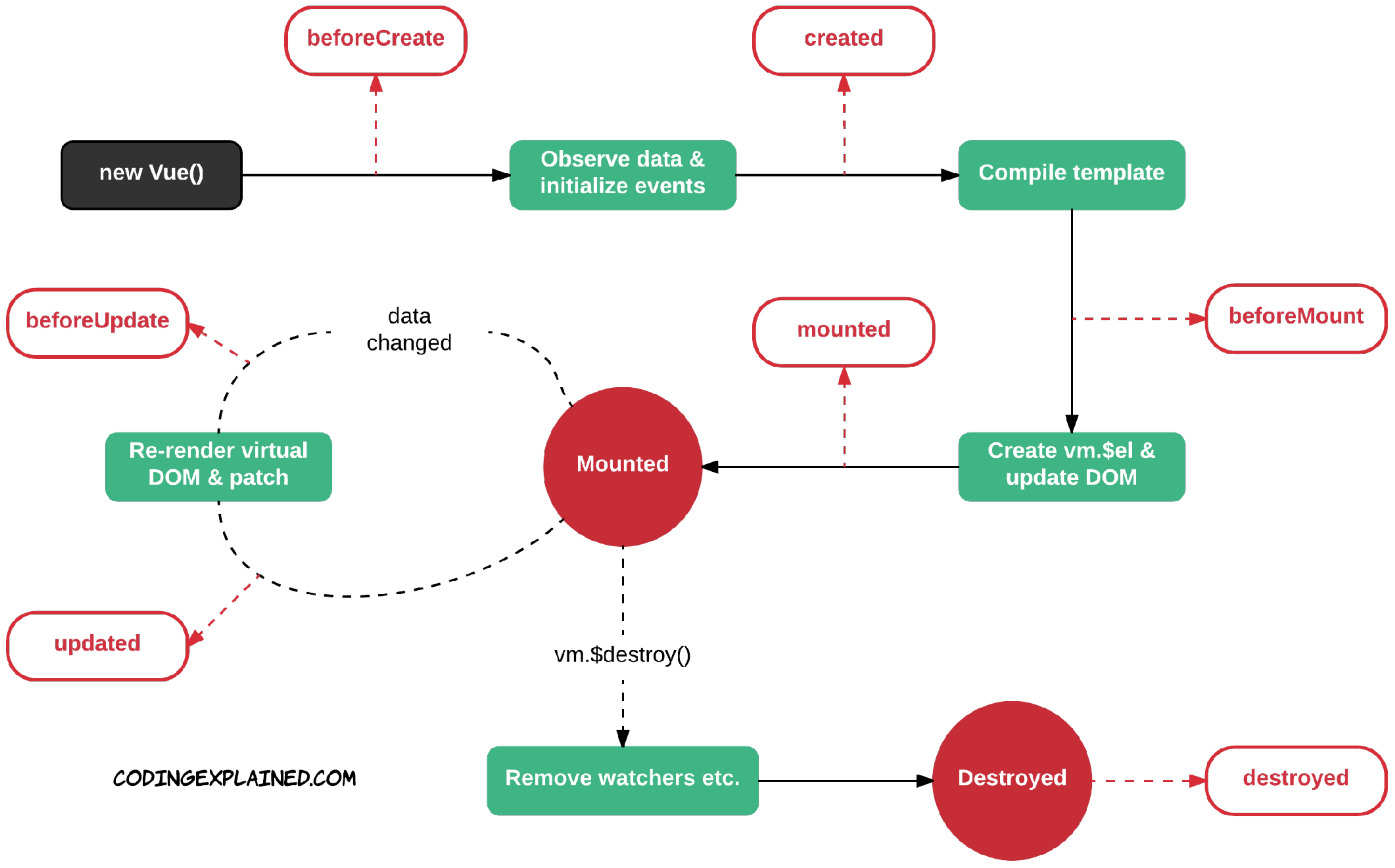
- Computed Properties are provided with custom getter functions.
- They will be re-computed **whenever the data properties got updated**. They could be used as normal data properties.

```
// Use computed property to get the superheroes
const superheroes = computed(() => {
    if (booking.value.team == 'Avengers') {
        return ["Captain America", "Iron Man",
"Thor", "Hulk", "Black Widow", "Hawkeye"];
    } else if (booking.value.team == "JLA") {
        return ["Superman", "Batman", "Wonder
Woman", "Flash", "Green Lantern”, "Aquaman"];
    } else {
        return [];
    }
});
```

LifeCycle Hooks

- Giving users the opportunity to add their own code at specific stages

```
onMounted(async () => {
    // if there is an id in the route
    if (route.params.id) {
        await getBooking();
    }
});
```



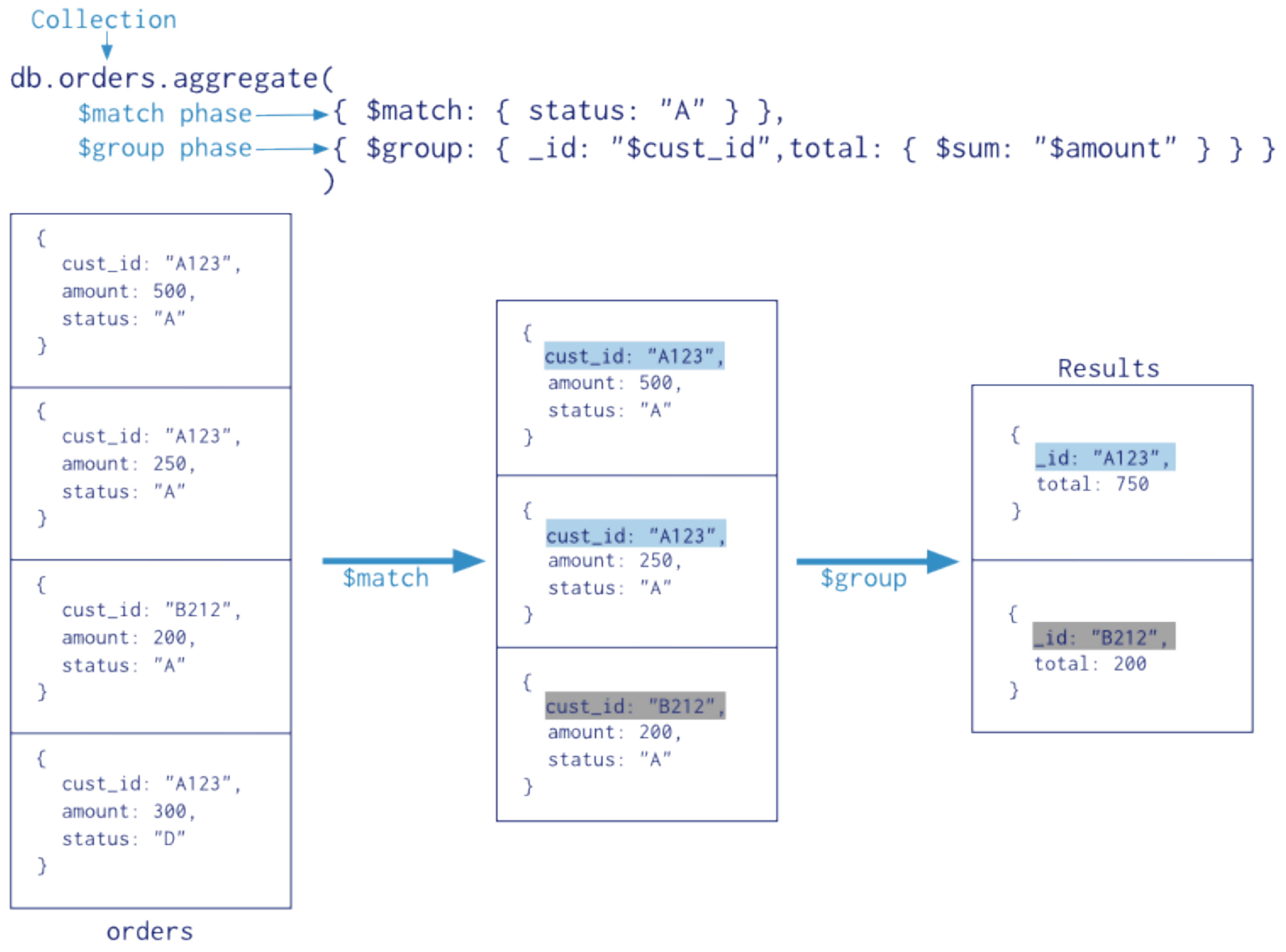
- onMounted— Execute after the instance is mounted to the DOM element.

Working with Data

JS Methods that Work with Data

- The push() method adds one or more elements to the end of an array and returns the new length of the array.
- concat(): **Concatenates** two or more arrays and returns a new array.
- map(): Creates a new array by applying a function to each element in the original array.
- filter(): Creates a new array with all elements that **pass a test specified by a function**.
- join(): Joins all elements of an array into **a string**, optionally separated by a specified delimiter.
- includes(): Checks **whether an array contains a specific element** and returns true or false.

Aggregate



```
db.orders.aggregate(
    { $match: { status: "A" } },
    { $group: { _id: "$cust_id",total: { $sum: "$amount" } } }
)
```

```
{
  cust_id: "A123",
  amount: 500,
  status: "A"
}
{
  cust_id: "A123",
  amount: 250,
  status: "A"
}
{
  cust_id: "B212",
  amount: 200,
  status: "A"
}
{
  cust_id: "A123",
  amount: 300,
  status: "D"
}
```

```
{
  cust_id: "A123",
  amount: 500,
  status: "A"
}
{
  cust_id: "A123",
  amount: 250,
  status: "A"
}
{
  cust_id: "B212",
  amount: 200,
  status: "A"
}
```

```
{
  _id: "A123",
  total: 750
}
{
  _id: "B212",
  total: 200
}
```

Aggregate Operations

- $match: Filters the documents based on specific criteria, similar to the find method. It allows you to select a subset of documents for further processing.

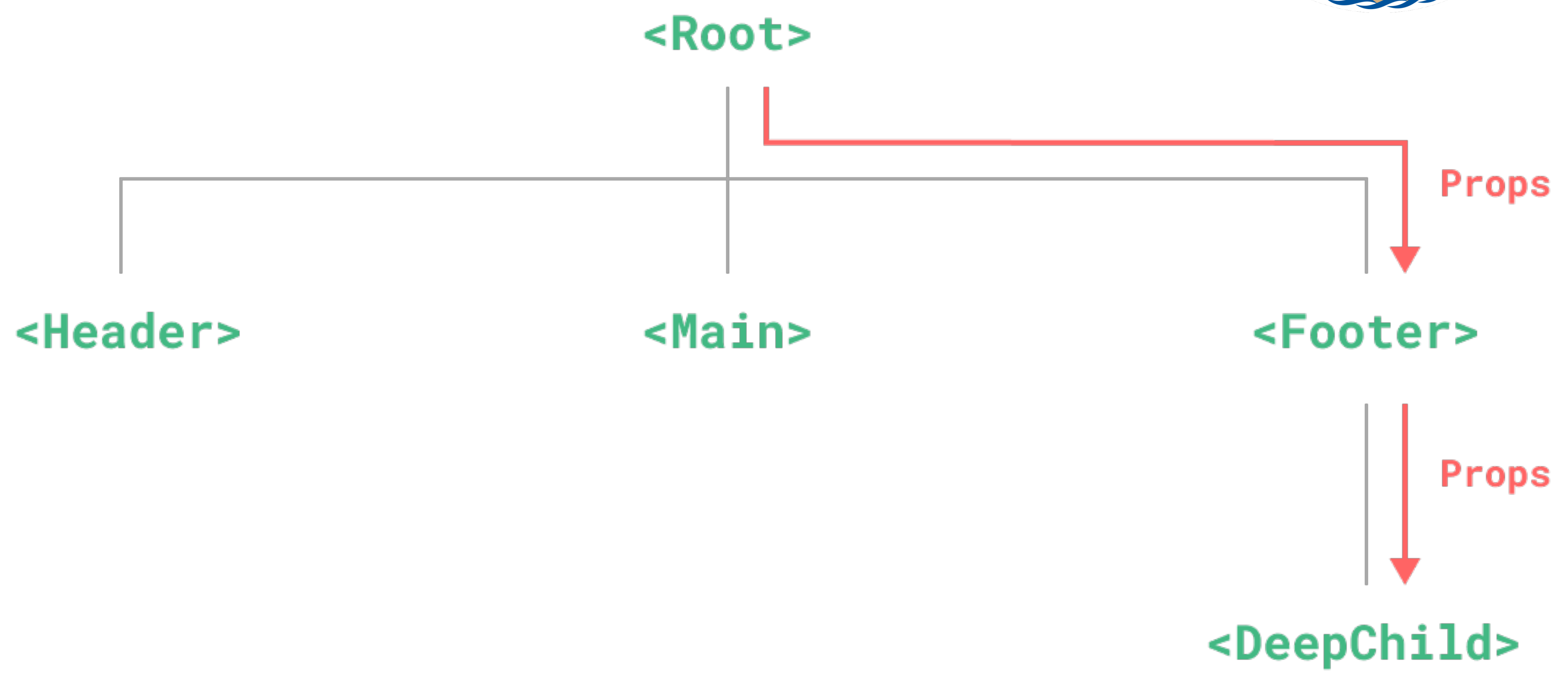
- $group: Groups the documents by a specified key and performs aggregations on each group. It is commonly used for tasks such as **calculating sums, averages, or counts**.

- $project: Specifies **which fields to include or exclude** from the output documents. It allows you to reshape the documents and create new computed fields.

Aggregate Operations

- $sort: Sorts the documents based on one or more fields, either in ascending or descending order.
- $limit and $skip: Controls the number of documents returned by the aggregation pipeline. $limit restricts the number of documents, while $skip skips a specified number of documents.
- $unwind: Deconstructs an array field from the input documents and **outputs one document for each element of the array**. It is useful when you need to perform further operations on individual elements of an array.

Props in Vue.js



- Usually, when we need to pass data from the parent to a child component, we use **Props**.
- Here, we provide **Props** (properties) via the template.

```
<DonutChart team="" />
<DonutChart team="Avengers" />
<DonutChart team="JLA" />
```

ChartsView,

The parent

DonutChart, the child element

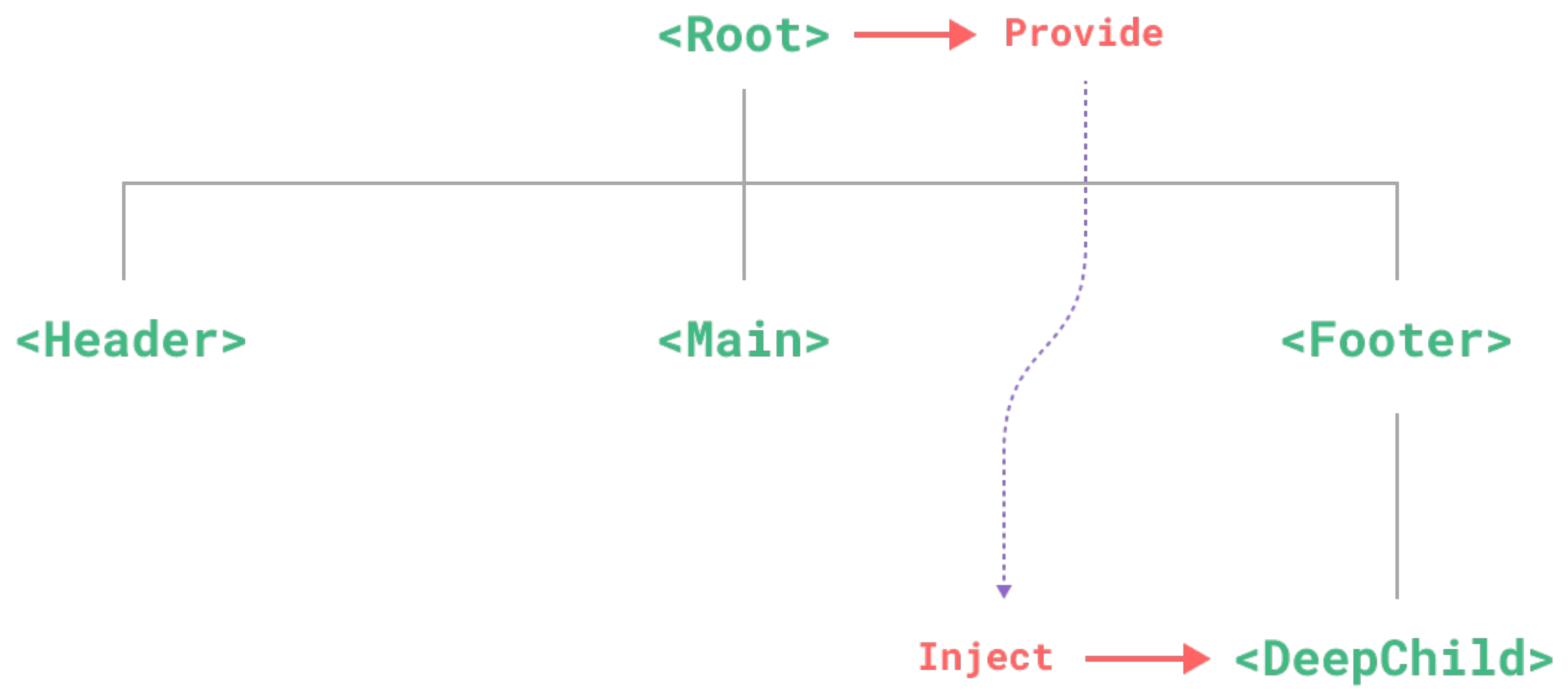
```
<script setup>
import { ref, onMounted, defineProps } from "vue";

const props = defineProps({
    team: String,
});
```

Provide / Inject

For Prop Drilling

- A parent component can serve as a dependency provider for all its descendants.
- Any component in the descendant tree, regardless of how deep it is, can **inject** dependencies **provided** by components up in its parent chain.



Token-based Authentication

Steps Involved

- User Authentication: When a user provides their credentials (such as username and password) during the login process, the server verifies the credentials and generates a token.
- Token Generation: Upon successful authentication, the server generates a token. This token is typically a long string of characters that contains encoded information about the user and any associated permissions or roles.
- **Token Storage**: The generated token is then **securely stored on the client-side**, usually in a cookie or **local storage**. It may also be stored on the server-side for validation purposes.

Steps Involved

- **Token Transmission**: For subsequent requests to authenticated endpoints, the token is included in the request, typically as an **HTTP header** (such as **Authorization: Bearer <token>**) or as a query parameter.

- Token Verification: When the server receives a request with a token, it validates the token to ensure its authenticity and integrity. This involves verifying the token's signature, checking its expiration, and validating any associated claims.

- Access Authorization: Once the token is successfully verified, the server uses the information contained in the token to determine if the user has the necessary permissions to access the requested resource. If authorized, the server processes the request and returns the appropriate response.

JSON Web Token

- A JWT consists of three parts: a header, a payload, and a signature.
  - Header: The header typically consists of two parts: the token type (which is JWT) and the signing algorithm used to generate the signature, such as HMAC, RSA, or ECDSA.
  - **Payload**: The payload contains the **claims**, which are **statements about the user or other data being transmitted**.
  - **Signature**: The signature is created by taking the encoded header, encoded payload, and a **secret key** known only to the server. It is used to verify the integrity of the token and ensure that it has not been tampered with.

localStorage

- localStorage is a web browser API that allows web applications to store and retrieve key-value pairs in a client-side storage area.
- It provides a simple way to store persistent data on the user's device, typically within the browser.
- localStorage data **persists even after the browser is closed and reopened**

Protecting an Express Route

- In Passport.js, the Bearer strategy is a popular authentication strategy used for authenticating API requests using bearer tokens. It is commonly used for token-based authentication, such as with JSON Web Tokens (JWTs).

- The Strategy is used in a **middleware function** to protect a route

```
router.patch('/:id/manage', passport.authenticate('bearer', { session: false }),
async function (req, res) {
```

$lookup

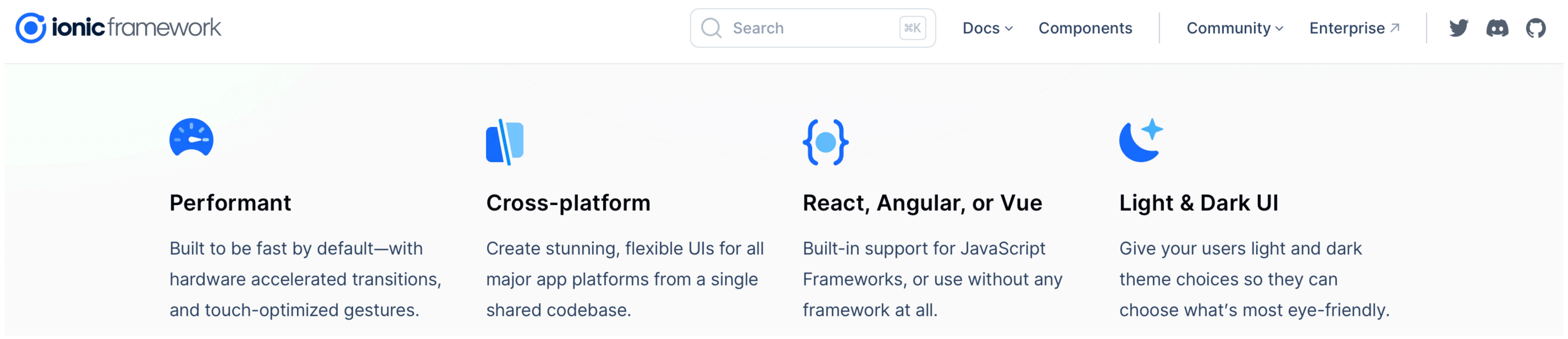
```
db.collection.aggregate([
  {
    $lookup: {
      from: <targetCollection>,
      localField: <fieldFromInputCollection>,
      foreignField: <fieldFromTargetCollection>,
      as: <outputArrayField>
    }
  }
])
```

- from: Specifies the **target collection** to join with.
- localField: Specifies the field from the input collection (the one on which the $lookup operation is performed).
- foreignField: Specifies the field from the **target collection** that should match the localField.
- as: Specifies the name of the output array field that will hold the joined documents.

Mobile App Development

Ionic Framework

- Could be exported to native codes
- Supports **cross-platform development**
- Works with modern frontend frameworks like Vue



Performant

Built to be fast by default—with hardware accelerated transitions, and touch-optimized gestures.

Cross-platform

Create stunning, flexible UIs for all major app platforms from a single shared codebase.

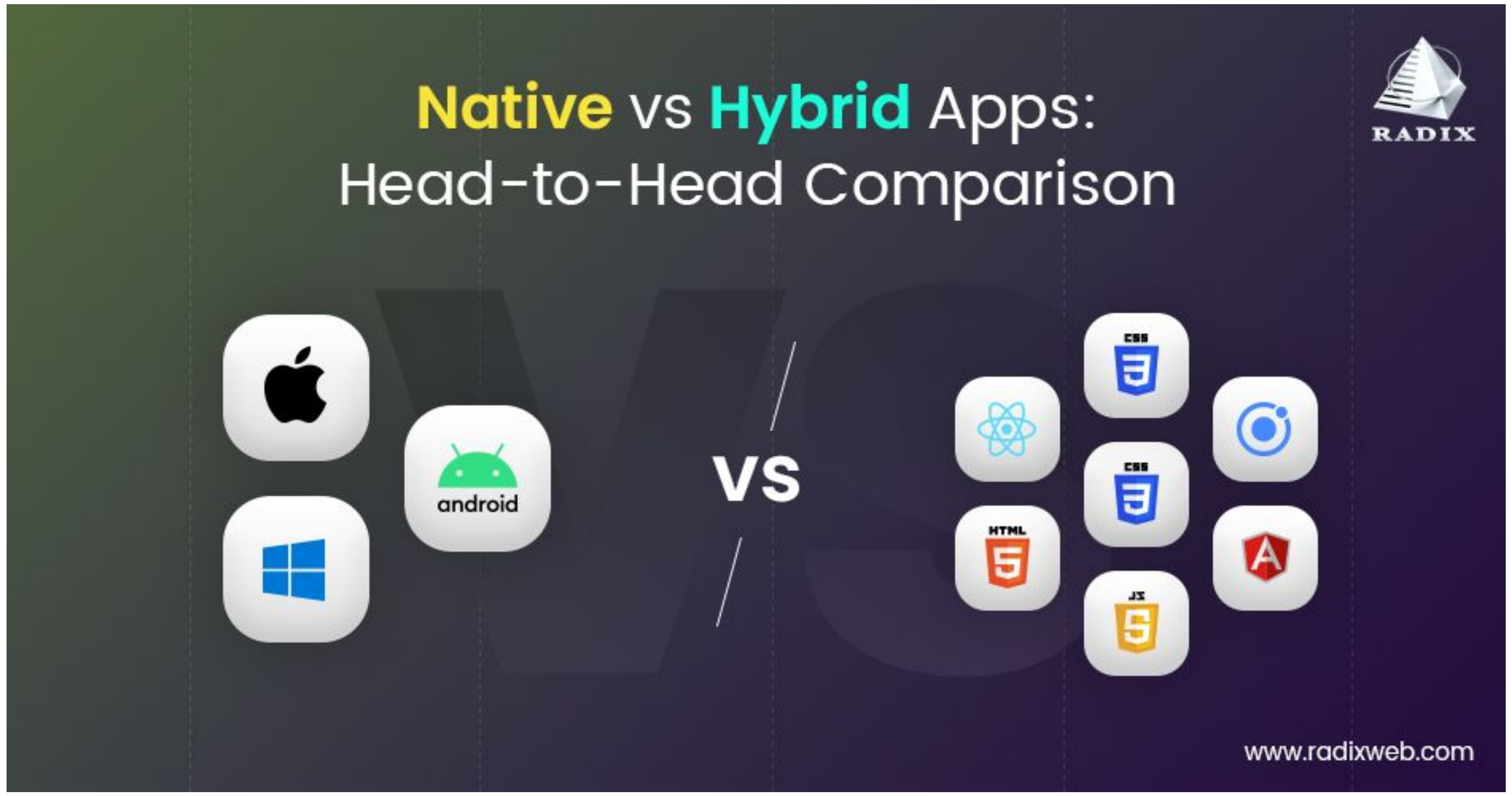
React, Angular, or Vue

Built-in support for JavaScript Frameworks, or use without any framework at all.

Light & Dark UI

Give your users light and dark theme choices so they can choose what's most eye-friendly.

Hybrid (or Web) App VS Native App



+

Code reusability
Faster prototyping
Lower costs
Cross-platform
…

-

Performance limitations
Limited access to native APIs
UI/UX Inconsistencies
…