

# Finite State Machines

in Vue 3







In web apps, state is  
everywhere.

# State can be strings, numbers

```
1 <template>
2   <p>My name is {{ name }} and I'm {{ age }} years old.</p>
3 </template>
4
5 <script>
6 import { ref, computed } from "vue";
7
8 export default {
9   setup() {
10     const name = ref("Sarah Dayan");
11     const birthYear = ref(1990);
12     const age = computed(() => new Date().getFullYear() - birthYear.value);
13
14     return { name, age };
15   },
16 }
```

# State can be arrays, objects

```
1 <template>
2   <p>My favorite snacks are:</p>
3   <ul>
4     <li v-for="{ label, emoji } in food" :key="label">
5       {{ label }} ({{ emoji }})
6     </li>
7   </ul>
8 </template>
9
10 <script>
11 import { ref } from "vue";
12
13 export default {
14   setup() {
15     const food = ref([
16       { label: "croissant", emoji: "🥐" },
17       { label: "apple", emoji: "🍏" },
18       { label: "banana", emoji: "🍌" },
19       { label: "orange", emoji: "🍊" },
20       { label: "grapes", emoji: "🍇" }
21     ])
22     return { food }
23   }
24 }
25
26 <script>
27 import { defineComponent } from "vue";
28
29 export default defineComponent({
30   template: `
31     <div>
32       <h1>My Snack List</h1>
33       <ul>
34         <li v-for="item in food" :key="item.label">
35           {{ item.label }} ({{ item.emoji }})
36         </li>
37       </ul>
38     </div>
39   `,
40   setup() {
41     const food = ref([
42       { label: "croissant", emoji: "🥐" },
43       { label: "apple", emoji: "🍏" },
44       { label: "banana", emoji: "🍌" },
45       { label: "orange", emoji: "🍊" },
46       { label: "grapes", emoji: "🍇" }
47     ])
48     return { food }
49   }
50 }
51
52 <script>
53 import { createApp } from "vue";
54 import App from "./App.vue";
55
56 createApp(App).mount("#app");
57
```

# State can be **booleans**

```
1 <template>
2   <p>I have a secret...</p>
3   <p v-if="shouldReveal">{{ secret }}</p>
4   <button v-else @click="reveal">Please tell!</button>
5 </template>
6
7 <script>
8 import { ref } from "vue";
9
10 export default {
11   setup() {
12     const shouldReveal = ref(false);
13     const secret = ref("Don't repeat it!");
14
15     function reveal() {
16       shouldReveal.value = true;
```

# Data fetching with loading state

```
1 export default {
2   setup() {
3     const { data, loading, error } = toRefs(
4       reactive({ data: [], loading: true, error: null })
5     );
6
7     fetch("https://example.com/api/users")
8       .then((users) => (data.value = users))
9       .catch((err) => (error.value = err))
10      .finally(() => (loading.value = false));
11
12     return { data, loading, error };
13   },
14 }
```

# Data fetching with loading state

```
1 export default {
2   setup() {
3     const { data, loading, error } = toRefs(
4       reactive({ data: [], loading: true, error: null })
5     );
6
7     fetch("https://example.com/api/users")
8       .then((users) => (data.value = users))
9       .catch((err) => (error.value = err))
10      .finally(() => (loading.value = false));
11
12     return { data, loading, error };
13   },
14 }
```

# Data fetching with loading state

```
1 export default {
2   setup() {
3     const { data, loading, error } = toRefs(
4       reactive({ data: [], loading: true, error: null })
5     );
6
7     fetch("https://example.com/api/users")
8       .then((users) => (data.value = users))
9       .catch((err) => (error.value = err))
10      .finally(() => (loading.value = false));
11
12     return { data, loading, error };
13   },
14 }
```

# Data fetching with loading state

```
1 export default {
2   setup() {
3     const { data, loading, error } = toRefs(
4       reactive({ data: [], loading: true, error: null })
5     );
6
7     fetch("https://example.com/api/users")
8       .then((users) => (data.value = users))
9       .catch((err) => (error.value = err))
10      .finally(() => (loading.value = false));
11
12     return { data, loading, error };
13   },
14 }
```

# Data fetching with loading state

```
1 export default {
2   setup() {
3     const { data, loading, error } = toRefs(
4       reactive({ data: [], loading: true, error: null })
5     );
6
7     fetch("https://example.com/api/users")
8       .then((users) => (data.value = users))
9       .catch((err) => (error.value = err))
10      .finally(() => (loading.value = false));
11
12     return { data, loading, error };
13   },
14 }
```

# Data fetching with loading state

```
1 <template>
2   <div v-if="loading">Loading users...</div>
3   <table v-if="data.length">
4     <td :key="user.id" v-for="user in data">
5       <div>{{ user.name }}</div>
6       <div>{{ user.email }}</div>
7     </td>
8   </table>
9   <div v-if="error">Uh-ho! Something unexpected happened.</div>
10 </template>
```

# Data fetching with loading state

```
1 <template>
2   <div v-if="loading">Loading users...</div>
3   <table v-if="data.length">
4     <td :key="user.id" v-for="user in data">
5       <div>{{ user.name }}</div>
6       <div>{{ user.email }}</div>
7     </td>
8   </table>
9   <div v-if="error">Uh-ho! Something unexpected happened.</div>
10 </template>
```

# Data fetching with loading state

```
1 <template>
2   <div v-if="loading">Loading users...</div>
3   <table v-if="data.length">
4     <td :key="user.id" v-for="user in data">
5       <div>{{ user.name }}</div>
6       <div>{{ user.email }}</div>
7     </td>
8   </table>
9   <div v-if="error">Uh-ho! Something unexpected happened.</div>
10 </template>
```

# Data fetching with loading state

```
1 <template>
2   <div v-if="loading">Loading users...</div>
3   <table v-if="data.length">
4     <td :key="user.id" v-for="user in data">
5       <div>{{ user.name }}</div>
6       <div>{{ user.email }}</div>
7     </td>
8   </table>
9   <div v-if="error">Uh-ho! Something unexpected happened.</div>
10 </template>
```

# Data fetching with loading state

```
1 const { data, loading, error } = toRefs(  
2   reactive({ data: [], loading: true, error: null }))  
3 );  
4  
5 fetch("https://example.com/api/users")  
6   .then((users) => (data.value = users))  
7   .catch((err) => (error.value = err));  
8  
9 // loading.value is still `true`  
10  
11 return { data, loading, error };
```

Loading...

**Tommy Miller**

[tommy.miller@example.com](mailto:tommy.miller@example.com)

**Ellie Williams**

[ellie.williams@example.com](mailto:ellie.williams@example.com)

**Joel Miller**

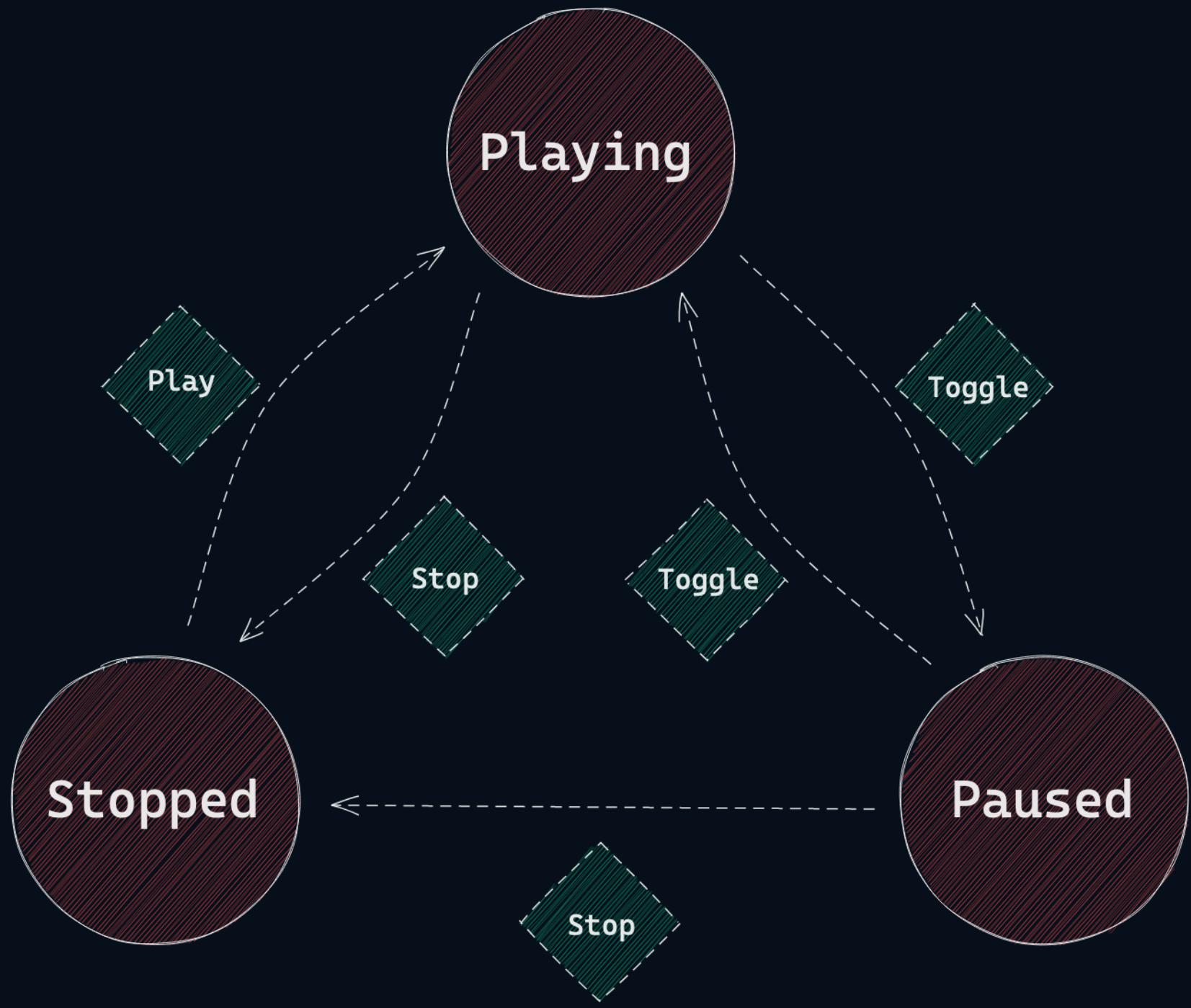
[joel.miller@example.com](mailto:joel.miller@example.com)

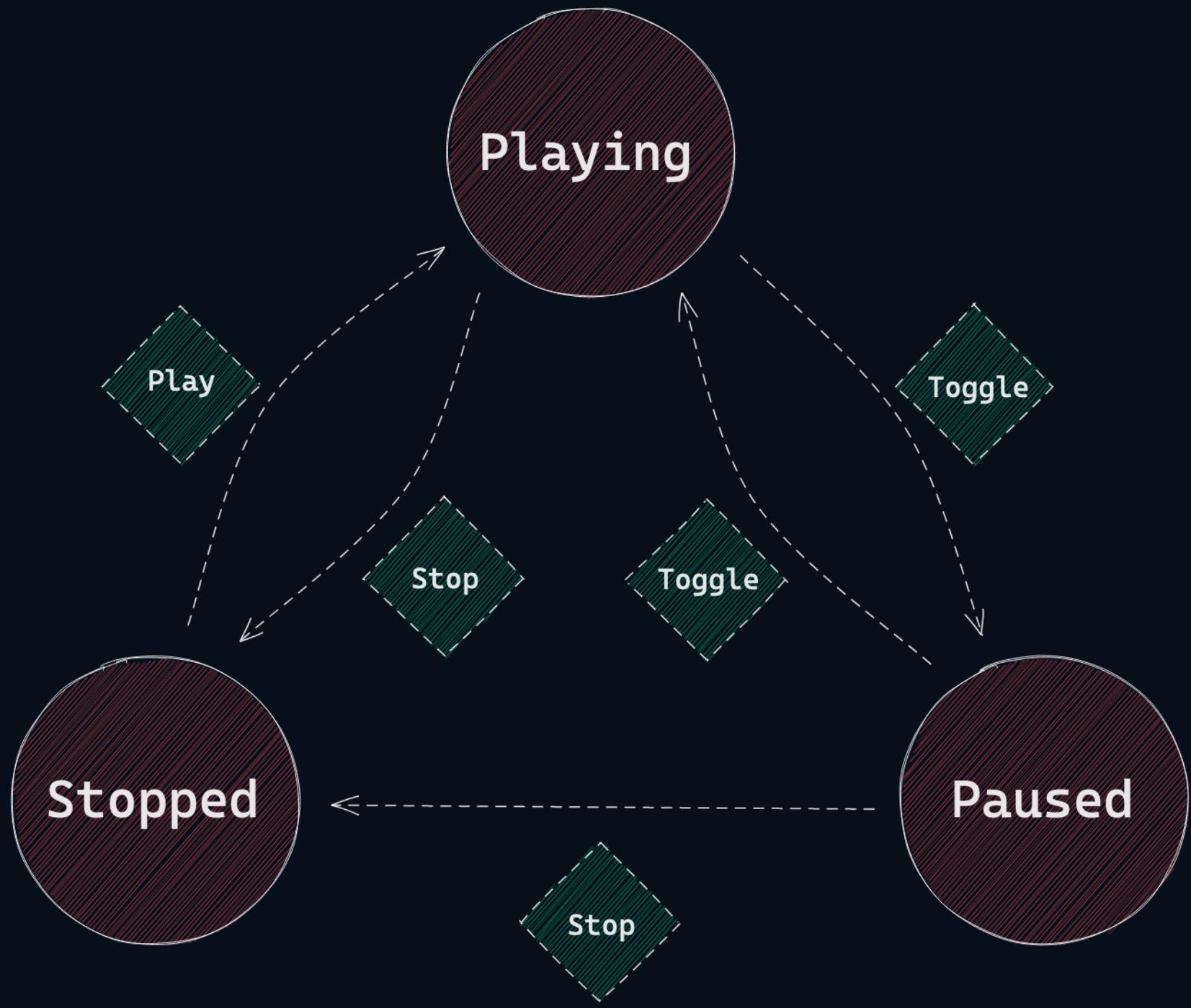
Imperative code  
is more error-prone.

We should only control  
the what, not the how.

We should only control  
the what, not the how.

# State machines.





X STATE

X STATE V