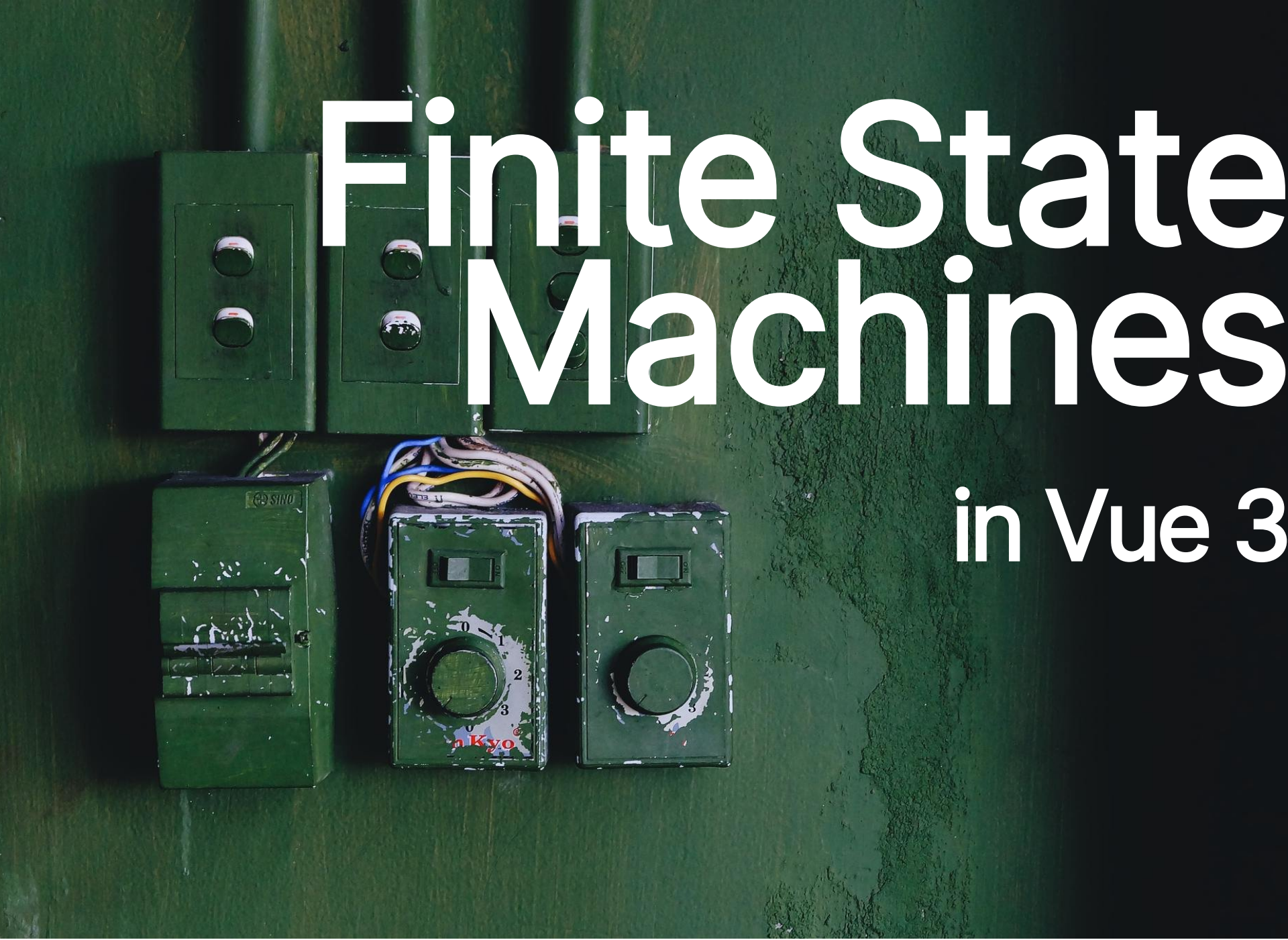

Finite State Machines
in Vue 3

In web apps, state is everywhere.

# State can be strings, numbers

```
<template>
  <p>My name is {{ name }} and I'm {{ age }} years old.</p>
</template>

<script>
import { ref, computed } from "vue";

export default {
  setup() {
    const name = ref("Sarah Dayan");
    const birthYear = ref(1990);
    const age = computed(() => new Date().getYear() - birthYear.value);

    return { name, age };
  },
};
```

# State can be arrays, objects

```
<template>
  <p>My favorite snacks are:</p>
  <ul>
    <li v-for="{ label, emoji } in food" :key="label">
      {{ label }} ({{ emoji }})
    </li>
  </ul>
</template>

<script>
import { ref } from "vue";

export default {
  setup() {
    const food = ref([
      { label: "croissant", emoji: "🥐" },
```

# State can be booleans

```
<template>
  <p>I have a secret...</p>
  <p v-if="shouldReveal">{{ secret }}</p>
  <button v-else @click="reveal">Please tell!</button>
</template>

<script>
import { ref } from "vue";

export default {
  setup() {
    const shouldReveal = ref(false);
    const secret = ref("Don't repeat it!");

    function reveal() {
      shouldReveal.value = true;
```

# Data fetching with loading state

```
export default {
  setup() {
    const { data, loading, error } = toRefs(
      reactive({ data: [], loading: true, error: null })
    );

    fetch("https://example.com/api/users")
      .then((users) => (data.value = users))
      .catch((err) => (error.value = err))
      .finally(() => (loading.value = false));

    return { data, loading, error };
  },
};
```

# Data fetching with loading state

```
export default {
  setup() {
    const { data, loading, error } = toRefs(
      reactive({ data: [], loading: true, error: null })
    );

    fetch("https://example.com/api/users")
      .then((users) => (data.value = users))
      .catch((err) => (error.value = err))
      .finally(() => (loading.value = false));

    return { data, loading, error };
  },
};
```

# Data fetching with loading state

```
export default {
  setup() {
    const { data, loading, error } = toRefs(
      reactive({ data: [], loading: true, error: null })
    );

    fetch("https://example.com/api/users")
      .then((users) => (data.value = users))
      .catch((err) => (error.value = err))
      .finally(() => (loading.value = false));

    return { data, loading, error };
  },
};
```

# Data fetching with loading state

```
export default {
  setup() {
    const { data, loading, error } = toRefs(
      reactive({ data: [], loading: true, error: null })
    );

    fetch("https://example.com/api/users")
      .then((users) => (data.value = users))
      .catch((err) => (error.value = err))
      .finally(() => (loading.value = false));

    return { data, loading, error };
  },
};
```

# Data fetching with loading state

```
export default {
  setup() {
    const { data, loading, error } = toRefs(
      reactive({ data: [], loading: true, error: null })
    );

    fetch("https://example.com/api/users")
      .then((users) => (data.value = users))
      .catch((err) => (error.value = err))
      .finally(() => (loading.value = false));

    return { data, loading, error };
  },
};
```

# Data fetching with loading state

```
<template>
  <div v-if="loading">Loading users...</div>
  <table v-if="data.length">
    <td :key="user.id" v-for="user in data">
      <div>{{ user.name }}</div>
      <div>{{ user.email }}</div>
    </td>
  </table>
  <div v-if="error">Uh-ho! Something unexpected happened.</div>
</template>
```

# Data fetching with loading state

```
<template>
  <div v-if="loading">Loading users...</div>
  <table v-if="data.length">
    <td :key="user.id" v-for="user in data">
      <div>{{ user.name }}</div>
      <div>{{ user.email }}</div>
    </td>
  </table>
  <div v-if="error">Uh-ho! Something unexpected happened.</div>
</template>
```

# Data fetching with loading state

```
<template>
  <div v-if="loading">Loading users...</div>
  <table v-if="data.length">
    <td :key="user.id" v-for="user in data">
      <div>{{ user.name }}</div>
      <div>{{ user.email }}</div>
    </td>
  </table>
  <div v-if="error">Uh-ho! Something unexpected happened.</div>
</template>
```

# Data fetching with loading state

```
<template>
  <div v-if="loading">Loading users...</div>
  <table v-if="data.length">
    <td :key="user.id" v-for="user in data">
      <div>{{ user.name }}</div>
      <div>{{ user.email }}</div>
    </td>
  </table>
  <div v-if="error">Uh-ho! Something unexpected happened.</div>
</template>
```

# Data fetching with loading state

```
const { data, loading, error } = toRefs(
  reactive({ data: [], loading: true, error: null })
);

fetch("https://example.com/api/users")
  .then((users) => (data.value = users))
  .catch((err) => (error.value = err));

// loading.value is still `true`

return { data, loading, error };
```

Loading...

Tommy Miller
tommy.miller@example.com

Ellie Williams
ellie.williams@example.com

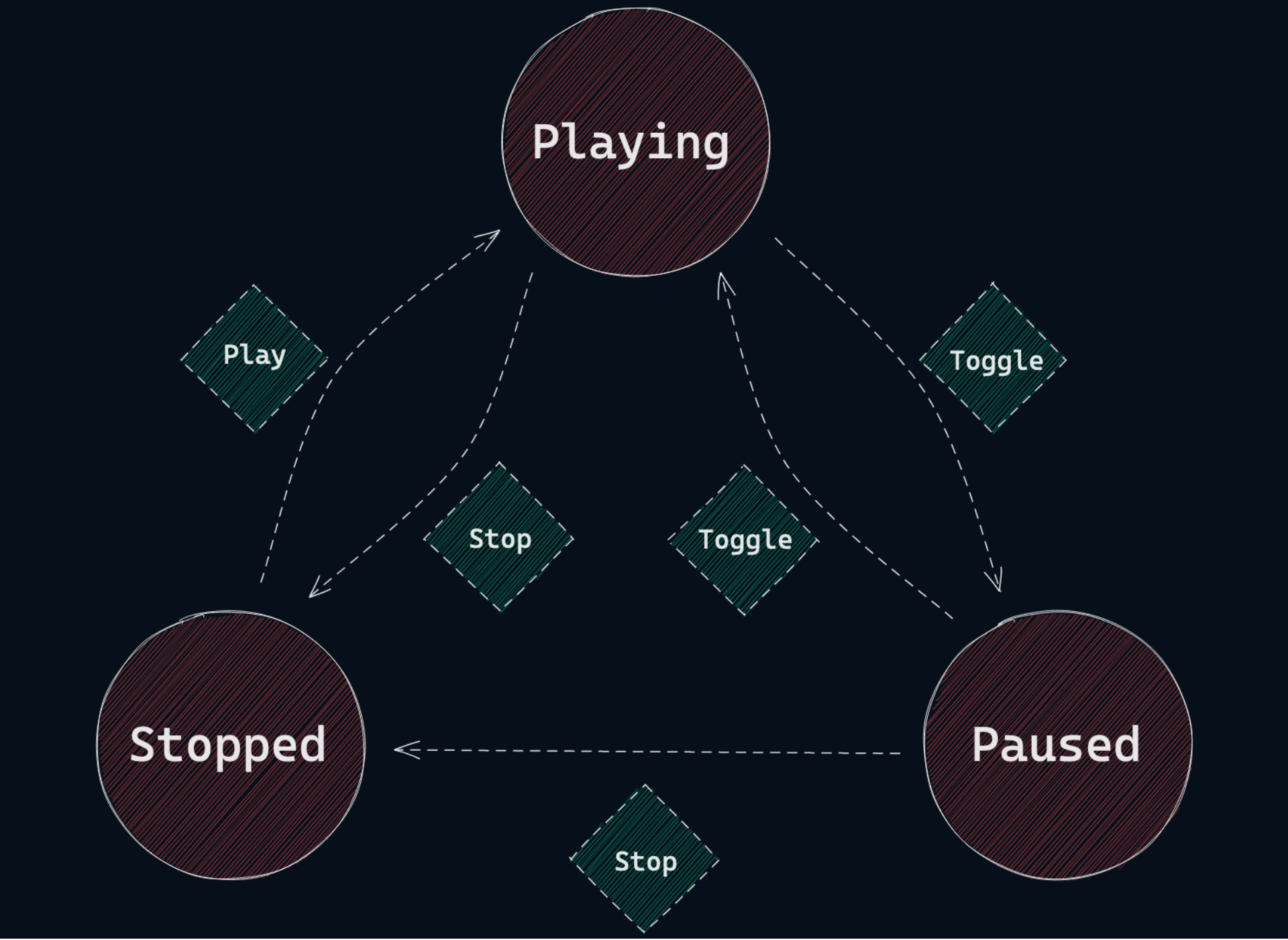
Joel Miller
joel.miller@example.com

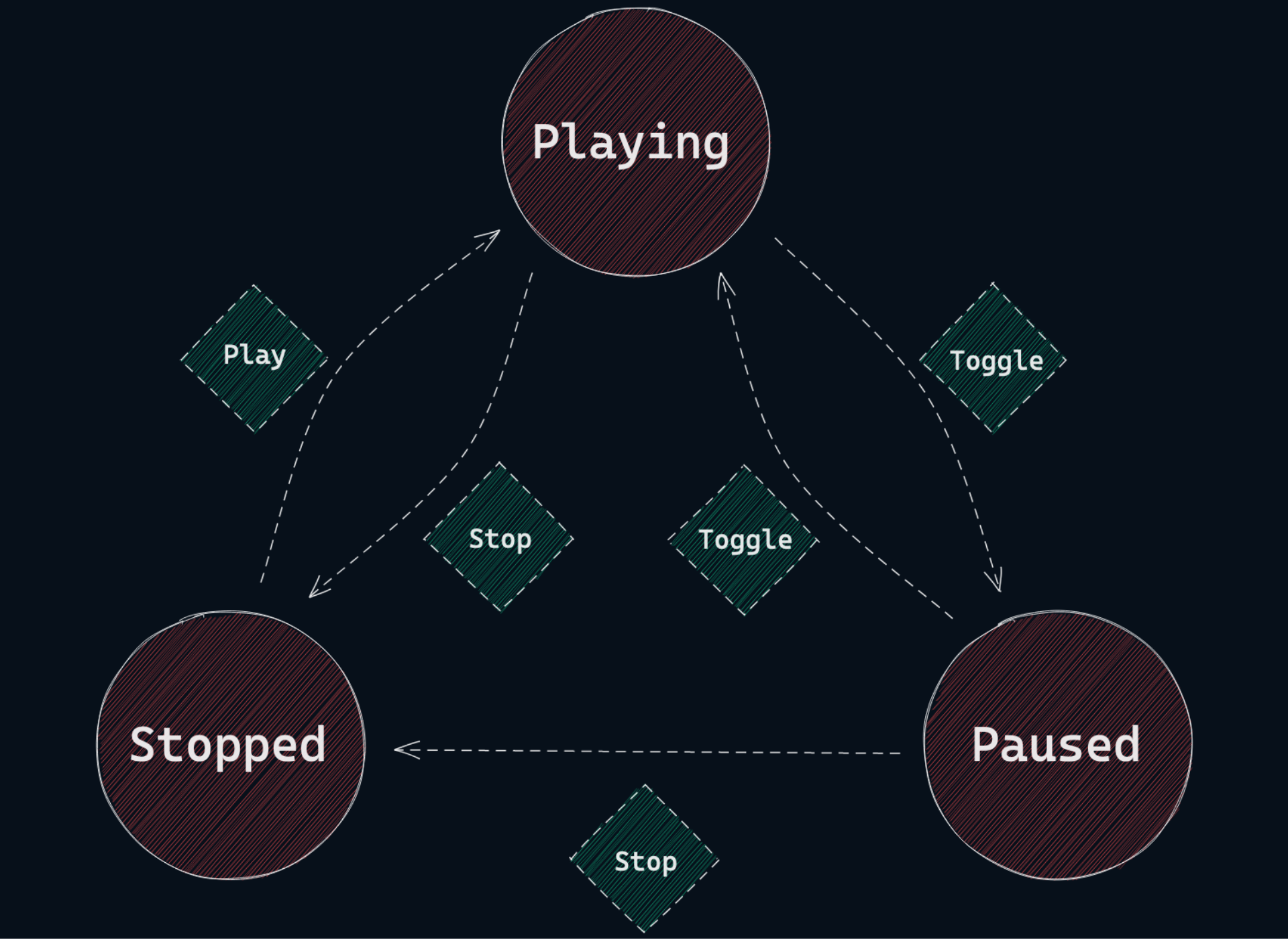
Imperative code
is more error-prone.

We should only control the what, not the how.

We should only control the what, not the how.

# State machines.


Playing
Play
Toggle
Stop
Toggle
Stopped
Paused
Stop


Playing
Play
Toggle
Stop
Toggle
Stopped
Paused
Stop


STATE


XSTATE