Composing Components

Having fun with `scoped slots` & `provide/inject`

@linus_borg
Vue.js Roadtrip Berlin
23.11.2018

whoami?

Thorsten Lünborg
(Linusborg)

Vue.js core team member

Forum-Question-Answerer

product owner by day, developer by night

Author of portal-vue

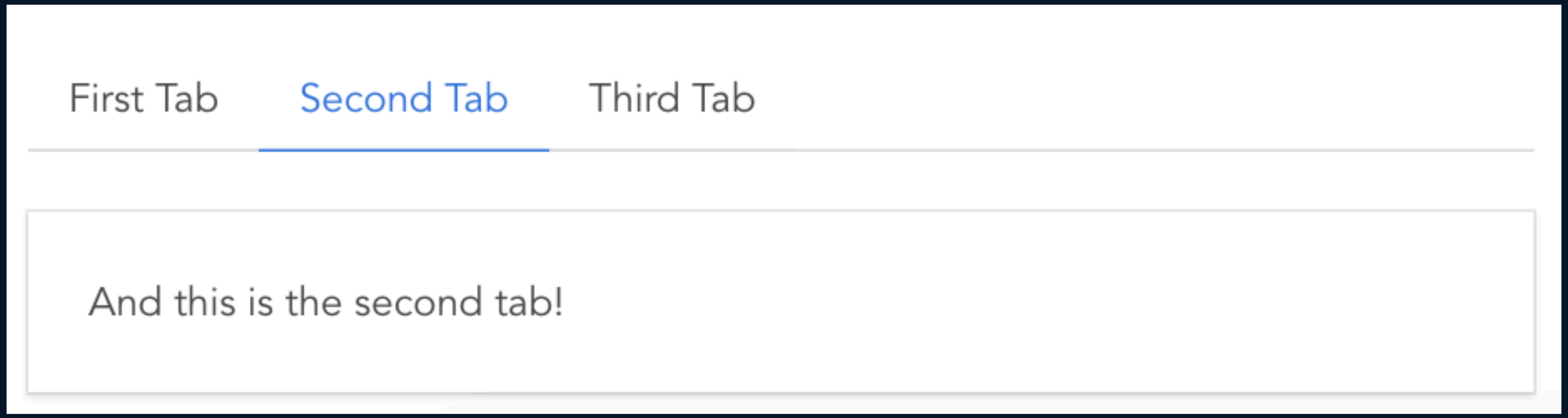
https://forum.vuejs.org

Github: linusborg
Twitter: @linus_borg

Scoped Slots

The Basics

```
<Tabs :tabs="tabs">
  <template slot-scope="{ active }“>

  </template>
</Tabs>
```



And this is the second tab!

```
<Tabs :tabs="tabs">

  <template slot-scope="{ active }“>

    <Tab v-if="active === 'First Tab'">
      <p class="first">This the fist tab!</p>
    </Tab>

    <Tab v-if="active === 'Second Tab'">
      <p class="second">And this is the second tab!</p>
    </Tab>

  </template>

</Tabs>
```

<Tabs>

```
<div>
  <div class="tabs">
    <ul>
      <li
        v-for="tabName in tabs"
        :key="tabName"
        :class="isActive(tabName)"
      >
        <a @click="active === tabName">
          {{ tabName }}
        </a>
      </li>
    </ul>
  </div>
  <div class="tabs-content">
    <slot :active="active" />
  </div>
</div>
```

```
export default {
  props: ['tabs'],
  data: () => ({
    active: null,
  }),
  methods: {
    isActive(name) {
      return this.active === name ? 'is-active' : null
    },
  },
}
```

<Tabs>

```
<div>
  <div class="tabs">
    <ul>
      <li
        v-for="tabName in tabs"
        :key="tabName"
        :class="isActive(tabName)"
      >
        <a @click="active === tabName">
          {{ tabName }}
        </a>
      </li>
    </ul>
  </div>
  <div class="tabs-content">
    <slot :active="active" />
  </div>
</div>
```

```
export default {
  props: ['tabs'],
  data: () => ({
    active: null,
  }),
  methods: {
    isActive(name) {
      return this.active === name ? 'is-active' : null
    },
  },
}
```

Scoped Slots

Wrapping State (Vuex)

```
<template>
  <div>
    <span>{{ count }}</span>
    <button @click="increment">Increment</button>
    <button @click="decrement">Increment</button>
  </div>
</template>
```

```
<script>
  import { mapState, mapActions } from 'vuex'
  export default {
    methods: {
      ...mapActions([increment, decrement])
    },
    computed: {
      ...mapState(['count'])
    }
  }
</script>
```

```
<template>
    <store>
      <div slot-scope="{state, dispatch }">
        <span>{{ state.count }}</span>
        <button @click="dispatch('increment')">Increment</button>
        <button @click="dispatch('decrement')">Increment</button>
      </div>
    </store>
</template>
```

```
<script>
  export default {
    // nothing to do here!
  }
</script>
```

<store>

```
export default {
  functional: true,

  render(h, { parent })  {

    const store = parent.$store

    return this.$scopedSlots.default({
      commit: store.commit,
      dispatch: store.dispatch,
      state: store.state,
      getters: store.getters,
    })[0]

  }
}
```

Scoped Slots

Wrapping behaviour: Promises

```
<template>
  <div>
    <span v-if="pending">Loading ...</span>
    <span v-elseif="error">An error happened!</span>
    <ul v-if="data.length">
      <li v-for="post in data" :key="post">{{ post }}</li>
    </ul>
  </div>
</template>
```

We're tracking promise state in our component

```
<script>
  export default {
    data: () => ({
      data: [],
      error: false,
      pending: false
    }),
    created() {
      this.pending = true
      this.data = await getPosts().catch(e => {
        this.error = true
      })
      this.pending = false
    }
  }
</script>
```

Promise state handled
in the template

```
<template>
  <div>
    <promised :promise="postsPromise">
      <div slot="combined" slot-scope="{data, pending, error }">
        <span v-if="pending">Loading ...</span>
        <span v-elseif="error">An error happened!</span>
        <ul v-if="data">
          <li v-for="post in data" :key="post">{{ post }}</li>
        </ul>
      </div>
    </promised>
  </div>
</template>
```

The component only takes care
of the promise

```
<script>
  export default {
    data: () => ({
      postsPromise: null,
    }),
    created() {
      this.postsPromise = getPosts()
    },
  }
</script>
```

https://github.com/posva/vue-promised

- UI Interactions
- APICalls
- Apollo
- Drag&Drop
- Authentication/Authorization Status
- …..

….but beware of …

Slot-Props Hell

"Callback hell all over again“

```
<template>
  <componentA>
    <componentB slot-scope="{ a, b }">
      <componentC slot-scope="{ c, d }">
        <TheComponentWeCareAbout
          slot-scope="{ e }"
          :a="a"
          :b="b"
          :c="c"
          :d="d"
          :e="e"
        />
      </componentC>
    </componentB>
  </componentA>
</template>
```

Scoped Slots

(…and Renderless Components)

👍

- High Flexibility for customisation
- Functionality can be abstracted away
- ..and explicitly accessed as slot props
- Less or no code in component necessary
- compose slot markup freely

👎

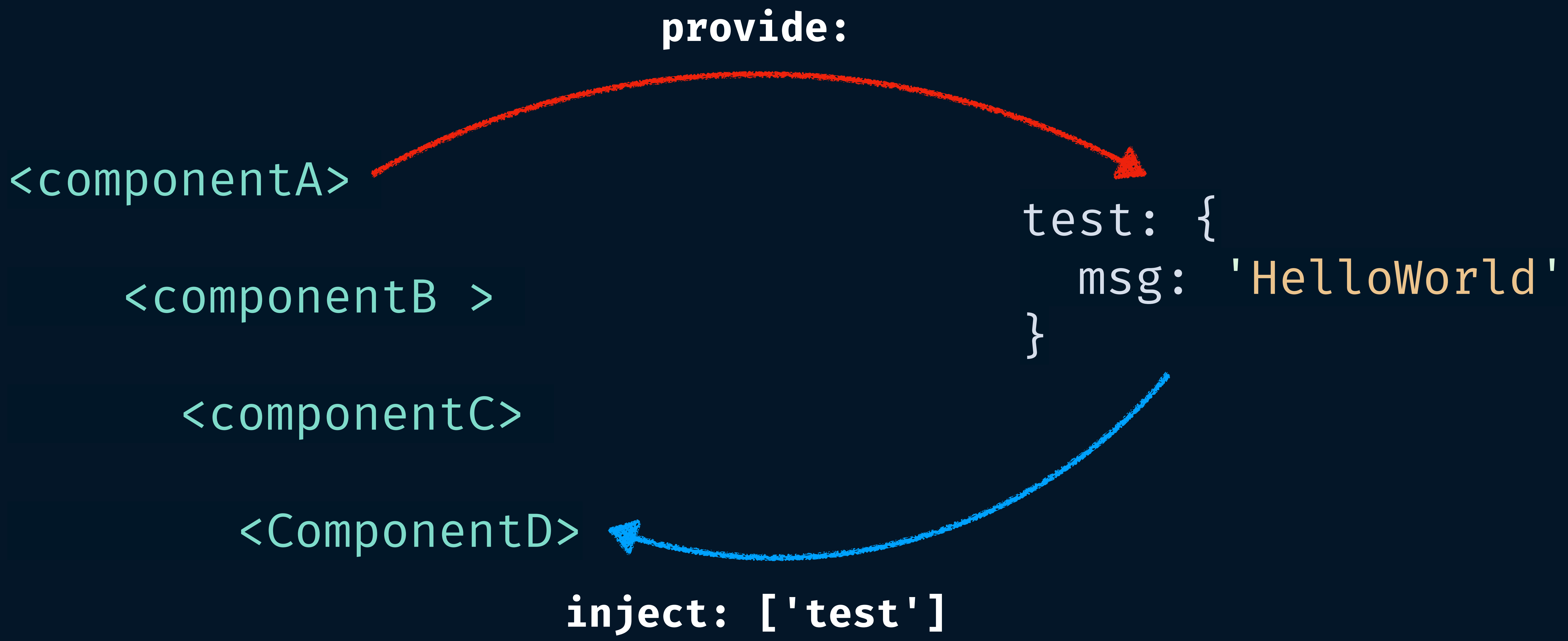
- „nested slots hell“ (callback hell 2.0)
- Access to exposed scope in component code is hard/impossible (esp. computed props)

Provide/Inject

The Basics

kind of like `Context`, for the React folks

🙃

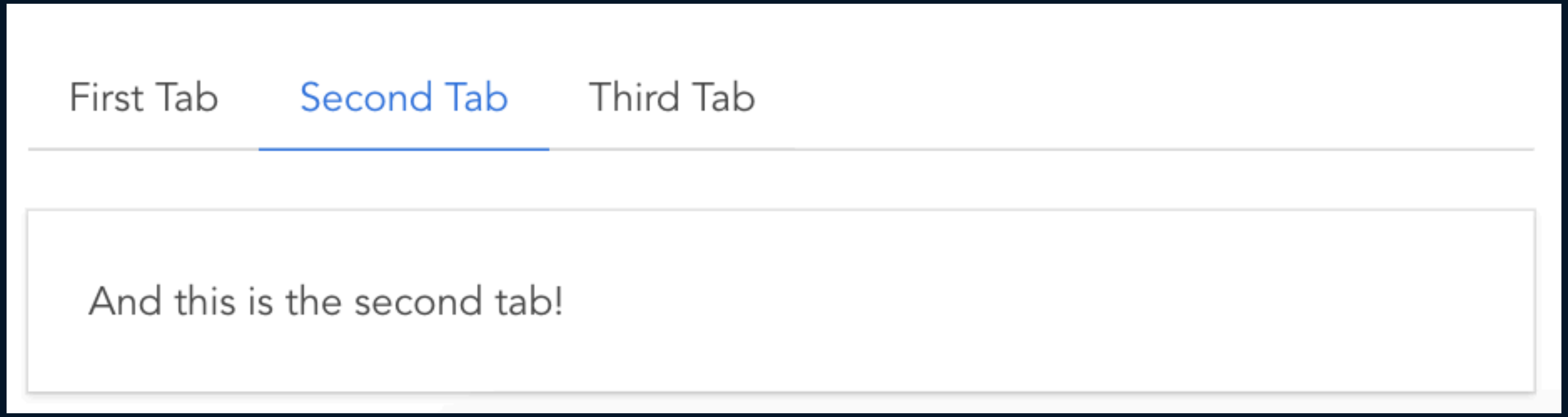


```
<componentA>
    <componentB >
        <componentC>
            <ComponentD>
```

```
test: {
  msg: 'HelloWorld'
}
```

We can share state & behaviour with grandchildren without passing props

```
<Tabs>

  <TabItem name="First Tab“ active>
    <p>This is content for the fist tab!</p>
  </TabItem>

  <TabItem name="Second Tab“>
    <h3>And this is the second tab!</h3>
  </TabItem>

</Tabs>
```



And this is the second tab!

<Tabs>

```
  <div class="tabs">
    <ul>
      <li
        v-for="tabName in tabsState.tabs"
        :key="tabName"
        :class="isActive(tabName)"
      >
        <a @click="tabsState.active = name">
          {{ tabName }}
        </a>
      </li>
    </ul>
  </div>

  <div class="tabs-content card">
    <div class=„card-content">
      <slot />
    </div>
  </div>

</div>
```

so it can be accessed from any (grand-)children

```
export default {
  data: vm => ({
    tabsState: {
      tabs: [],
      active: '',
    },
  }),

  provide() {
    return {
      tabsState: this.tabsState,
    }
  },

  methods: {
    isActive(name) {
      return this.tabsState.active === name
        ? 'is-active'
        : null
    },
  },
}
```

we provide the object

<TabItem>

```
<template>
  <div v-if=„isActive">
    <slot />
  </div>
</template>
```

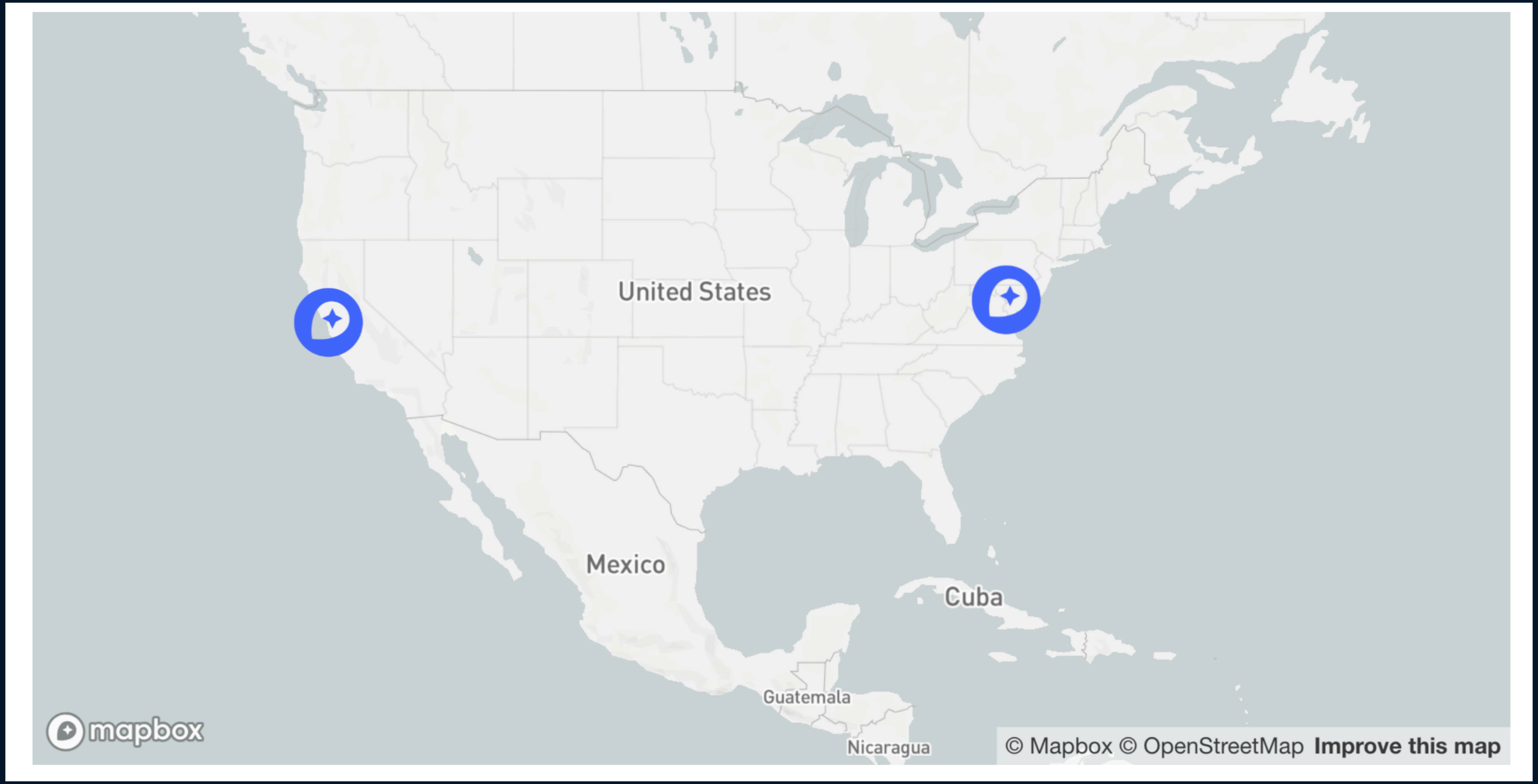
```
export default {
  inject: ['tabsState'],

  props: ['name', 'active'],

  created() {
    this.tabsState.push(this.name)
    if (this.active) {
      this.tabsState.active = this.name
    }
  },

  beforeDestroy() {
    removeFromArray(this.tabsState.tabs, this.name)
  },

  computed: {
    isActive() {
      return this.tabsState.active === this.name
    },
  },
}
```

- More complicated implementation
- … rewarding you with a cleaner template
- Ability for complicated parent - child interactions
- share state and behaviour between distant relatives

👍

Provide/Inject

Renderless Children



https://www.mapbox.com/help/custom-markers-gl-js/

```
var map = new mapboxgl.Map({
  container: 'map',
  style: 'mapbox://styles/mapbox/light-v9',
  center: [-96, 37.8],
  zoom: 3
});

var el = document.createElement('div');
el.className = 'marker';

new mapboxgl.Marker(el)
  .setLngLat(coordinates)
  .addTo(map);
```

```
<map>
  <marker :coordinates="coordinates"/>
</map>
```

<map>

```
<template>
  <div>
    <div ref="map" />
    <slot />
  </div>
</template>
```

mounting the map

providing the map to child components

```
<script>
  export default {
    mounted() {
      this.map = new mapboxgl.Map({
        container: this.$refs.map,
        style: 'mapbox://styles/mapbox/light-v9',
        center: [-96, 37.8],
        zoom: 3
      });
    },
    provide() {
      return {
        map: this.map
      }
    }
  }
</script>
```

<marker>

inject the map instance

we don’t render anyting!

```
<script>
  export default {
    inject: ['map'],

    render: () ⇒ null,

    mounted() {
      const el = document.createElement('DIV')
      this.marker = new mapboxgl.Marker(el)
        .setLngLat(coordinates)
        .addTo(this.map);
    },
    beforeDestroy() {
      this.marker.remove(this.map)
    }
  }
</script>
```

- We cache the marker on a property
- and add it to the map

When the component is destroyed, we remove the marker

Provide/Inject

👍

- component interactions can be abstracted away
- accessed implicitly in children

  .... or grand-grand(—)children !!!
- very clean template markup

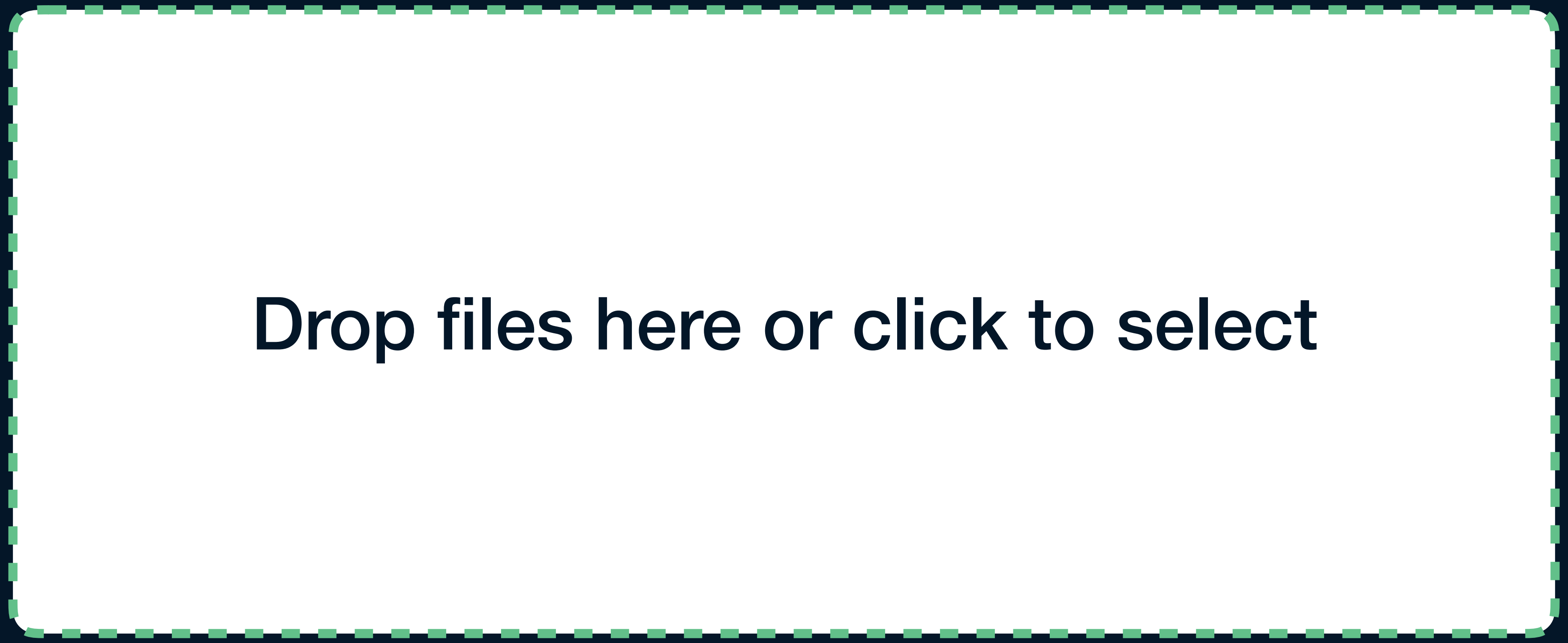
👎

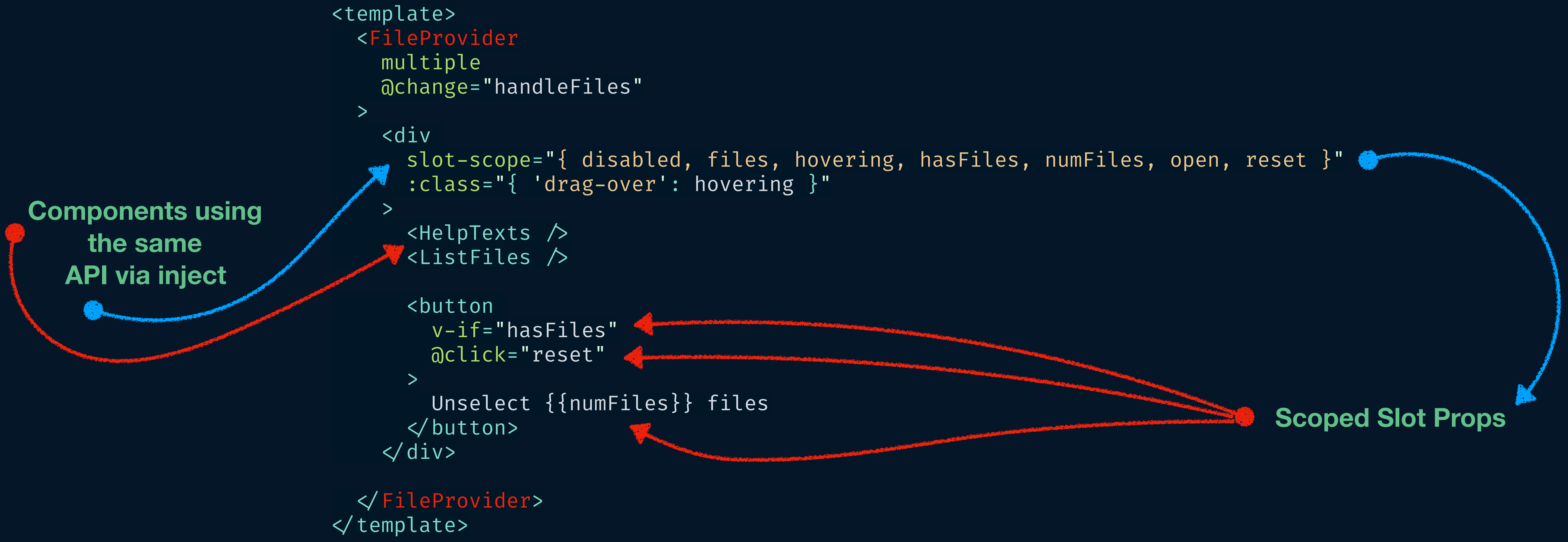
- implementation is harder to understand
  - not as explicit as scoped slots
  - some boilerplate necessary to pass reactive data
- adjusting styles etc. for child components

  can be cumbersome

Why not both?

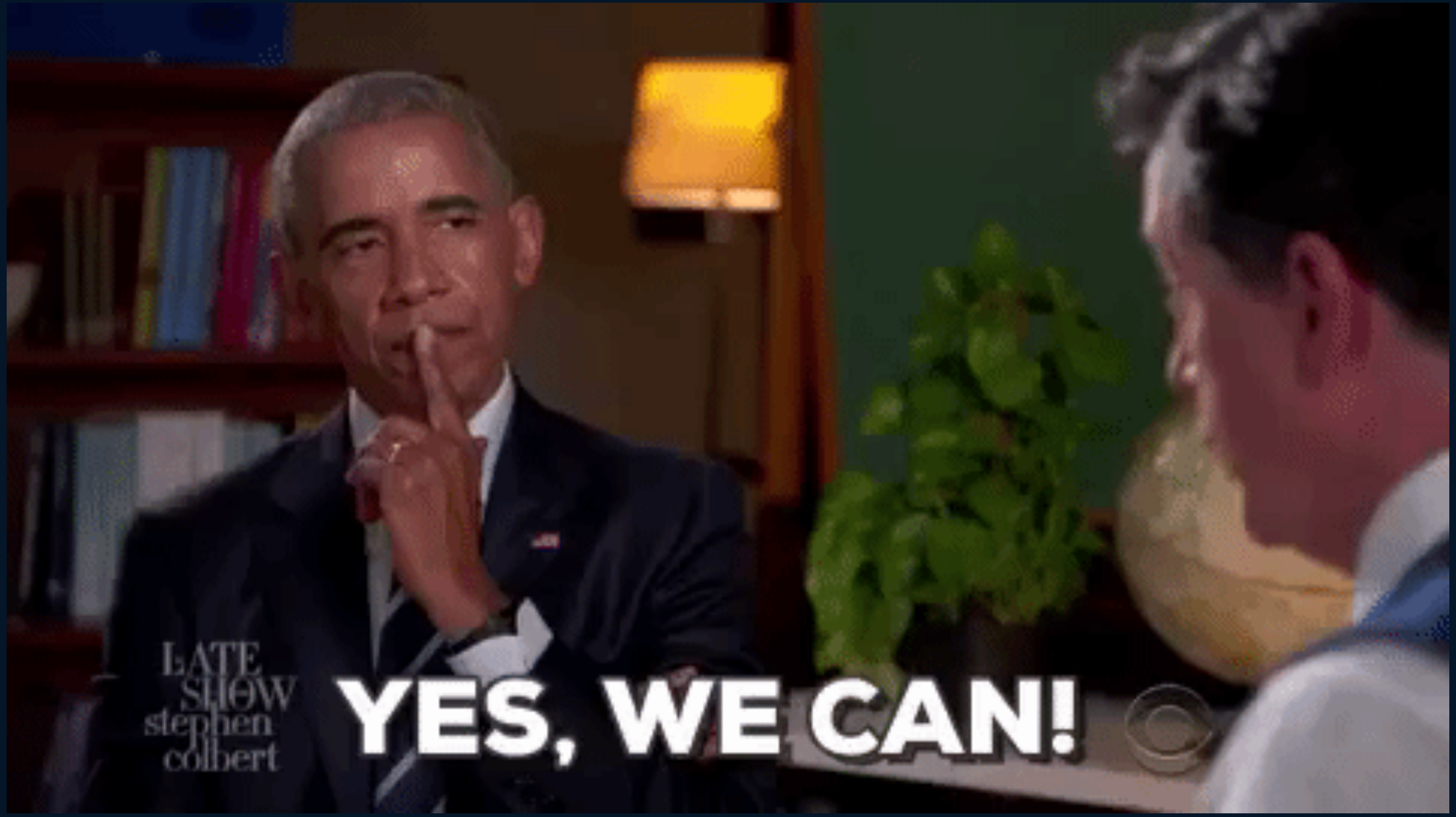
- Expose the same API via provide/inject and scoped slots
- Let the developer decide which fits the use case
- We can ship optional components ready to use, which real on provide/inject
- Developers can use these components or write their own markup





```
<template>
  <FileProvider
    multiple
    @change="handleFiles"
  >
    <div
      slot-scope="{ disabled, files, hovering, hasFiles, numFiles, open, reset }"
      :class="{ 'drag-over': hovering }"
    >
      <HelpTexts />
      <ListFiles />

      <button
        v-if="hasFiles"
        @click="reset"
      >
        Unselect {{numFiles}} files
      </button>
    </div>

  </FileProvider>
</template>
```

Can we do this?



<FileProvider>

```
<div
  @dragenter.prevent="hovering = true"
  @dragleave.prevent="hovering = false"
  @dragover.prevent
  @drop.prevent="onDrop"
>
</div>
```

```
export default {
  name: 'FileProvider',
  data: () ⇒ ({
    files: [],
    hovering: false,
  }),
  methods: {
    handleFiles(files) { … },
    onDrop(event) {
      this.handleFiles(event.dataTransfer.files)
    },
  },
}
```

<FileProvider>

```
<div
  @dragenter.prevent="hovering = true"
  @dragleave.prevent="hovering = false"
  @dragover.prevent
  @drop.prevent="onDrop"
>
  <input
    ref="file"
    :key="resetCount"
    :disabled="disabled"
    style="display: none;"
    type="file"
    @change="onInput"
  />
</div>
```

```
export default {
  name: 'FileProvider',
  props: {
    multiple: Boolean,
    disabled: Boolean,
  },
  data: () ⇒ ({
    files: [],
    hovering: false,
  }),
  methods: {
    handleFiles(files) { … },
    onDrop(event) {
      this.handleFiles(event.dataTransfer.files)
    },
  },
}
```

<FileProvider>

```
<div
  @dragenter.prevent="hovering = true"
  @dragleave.prevent="hovering = false"
  @dragover.prevent
  @drop.prevent="onDrop"
>
  <input
    ref="file"
    :key="resetCount"
    :disabled="disabled"
    style="display: none;"
    type="file"
    @change="onInput"
  />
</div>
```

```
export default {
  name: 'FileProvider',
  props: {
    multiple: Boolean,
    disabled: Boolean,
  },
  data: () ⇒ ({
    files: [],
    hovering: false,
  }),
  methods: {
    handleFiles(files) { … },
    onInput(event) {
      this.handleFiles(event.target.files)
    },
    onDrop(event) {
      this.handleFiles(event.dataTransfer.files)
    },
    open() { this.$refs.file.click() },
    reset() { this.files = [] },
  },
  computed: {
    numFiles() { return this.files.length },
    hasFiles() { return this.numFiles > 0 },
  },
  watch: {
    files(files) { this.$emit('input', files) }
  }
}
```

<FileProvider>

```
<div
  @dragenter.prevent="hovering = true"
  @dragleave.prevent="hovering = false"
  @dragover.prevent
  @drop.prevent="onDrop"
>
  <input
    ref="file"
    :key="resetCount"
    :disabled="disabled"
    style="display: none;"
    type="file"
    @change="onInput"
  />
  <slot v-bind="dropzone__api" />
</div>
```

```
export default {
  name: 'FileProvider',
  props: {
    multiple: Boolean,
    disabled: Boolean,
  },
  data: () => ({
    files: [],
    hovering: false,
  }),
  methods: {
    handleFiles(files) { … },
    onInput(event) {
      this.handleFiles(event.target.files)
    },
    onDrop(event) {
      this.handleFiles(event.dataTransfer.files)
    },
    open() { this.$refs.file.click() },
    reset() { this.files = [] },
  },
  computed: {
    numFiles() { return this.files.length },
    hasFiles() { return this.numFiles > 0 },
  },
  watch: {
    files(files) { this.$emit('input', files) }
  }
}
```

<FileProvider>

```
<div
  @dragenter.prevent="hovering = true"
  @dragleave.prevent="hovering = false"
  @dragover.prevent
  @drop.prevent="onDrop"
>
  <input
    ref="file"
    :key="resetCount"
    :disabled="disabled"
    style="display: none;"
    type="file"
    @change="onInput"
  />
  <slot v-bind="dropzone__api" />
</div>
```

which we use to pass
all properties to
the scoped slot

```
export default {
  name: 'FileProvider',
  props: {
    multiple: Boolean,
    disabled: Boolean,
  },
  data: () => ({
    files: [],
    hovering: false,
  }),
  reactiveProvide: {
    name: 'dropzone__api',
    include: [
      'disabled',
      'files',
      'numFiles',
      'open',
      'reset',
      'hovering',
    ],
  },
  methods: {
    …
  },
  computed: {
    …
  },
  // …
}
```

adds a „provide“
and
computed property
by the same name

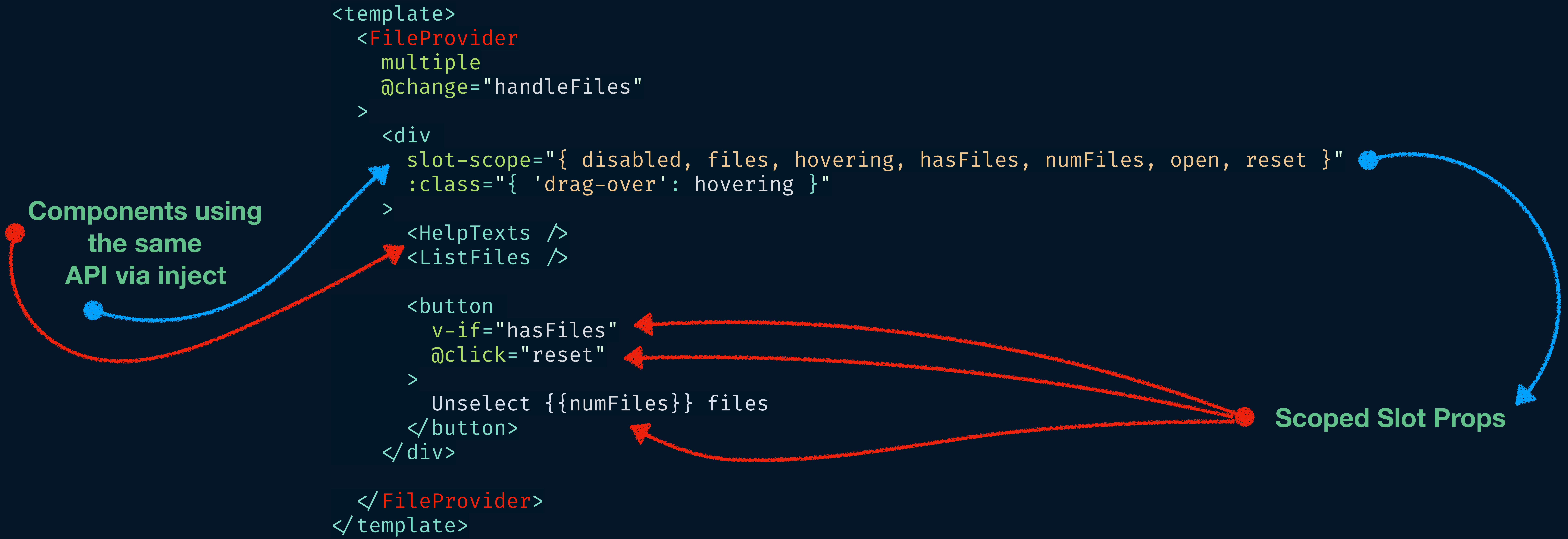
https://github.com/linusborg/vue-reactive-provide

<ListFiles>

Inject (and rename) the provided object

```
<template>
  <div v-if="dz.hasFiles">
    <ul>
      <li v-for="file in dz.files" :key="file.name">
        {{ file.name}} ( {{ file.size }})
      </li>
    </ul>
  </div>
</template>
```

```
<script>
export default {
  inject: {
    dropzone__api: 'dz',
  },
}
</script>
```



```
<template>
  <FileProvider
    multiple
    @change="handleFiles"
  >
    <div
      slot-scope="{ disabled, files, hovering, hasFiles, numFiles, open, reset }"
      :class="{ 'drag-over': hovering }"
    >
      <HelpTexts />
      <ListFiles />

      <button
        v-if="hasFiles"
        @click="reset"
      >
        Unselect {{numFiles}} files
      </button>
    </div>

  </FileProvider>
</template>
```

Conclusion

- scoped slots and provide/inject enable interesting patterns
- both can be misused and abused
- but used in the right situation, make you code cleaner and your life easier
- used together, they are especially useful unstoppable
- Don’t worry about „best practices“, instead follow your curiosity

🎶 so compose something, it’s fun!!! 🤩

Github: linusborg

Twitter: @linus_borg