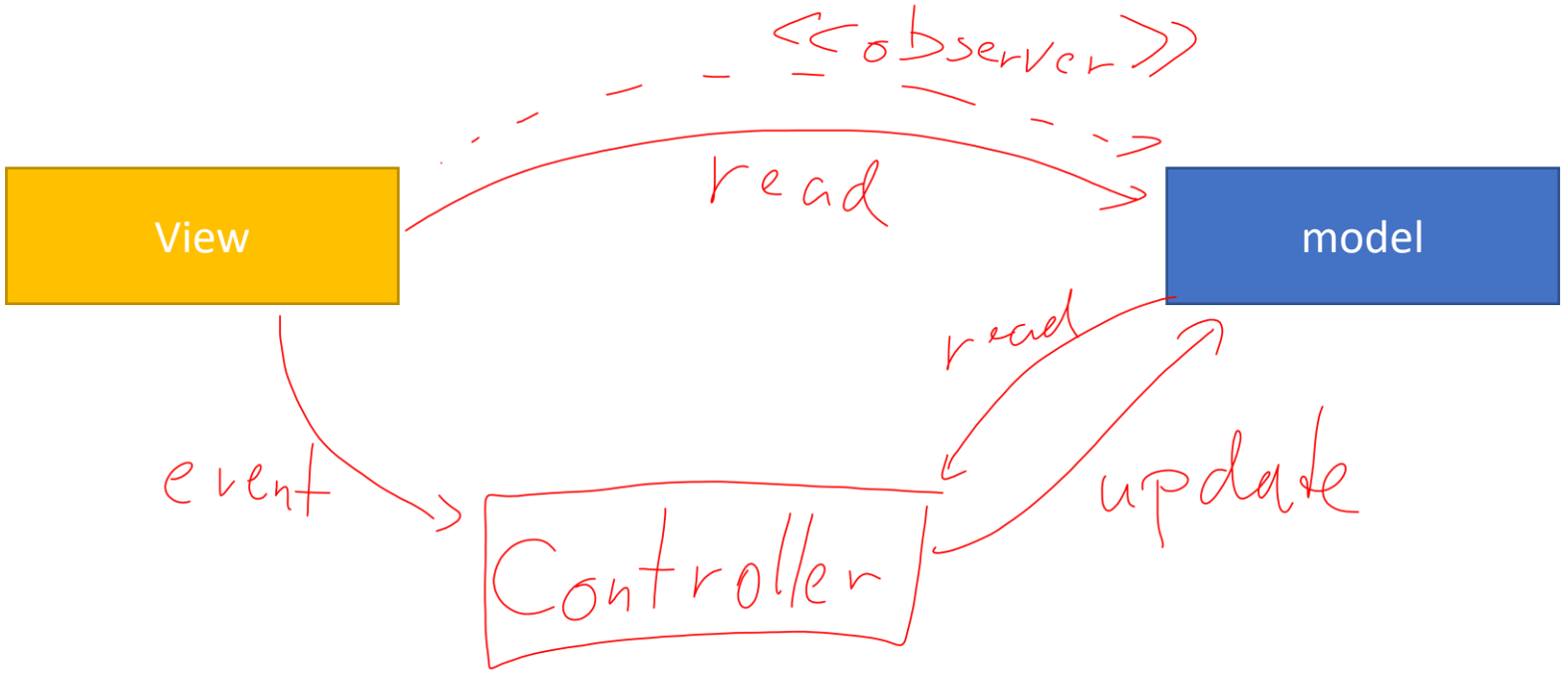
WEB3, Session 3

MVVM in Vue.js

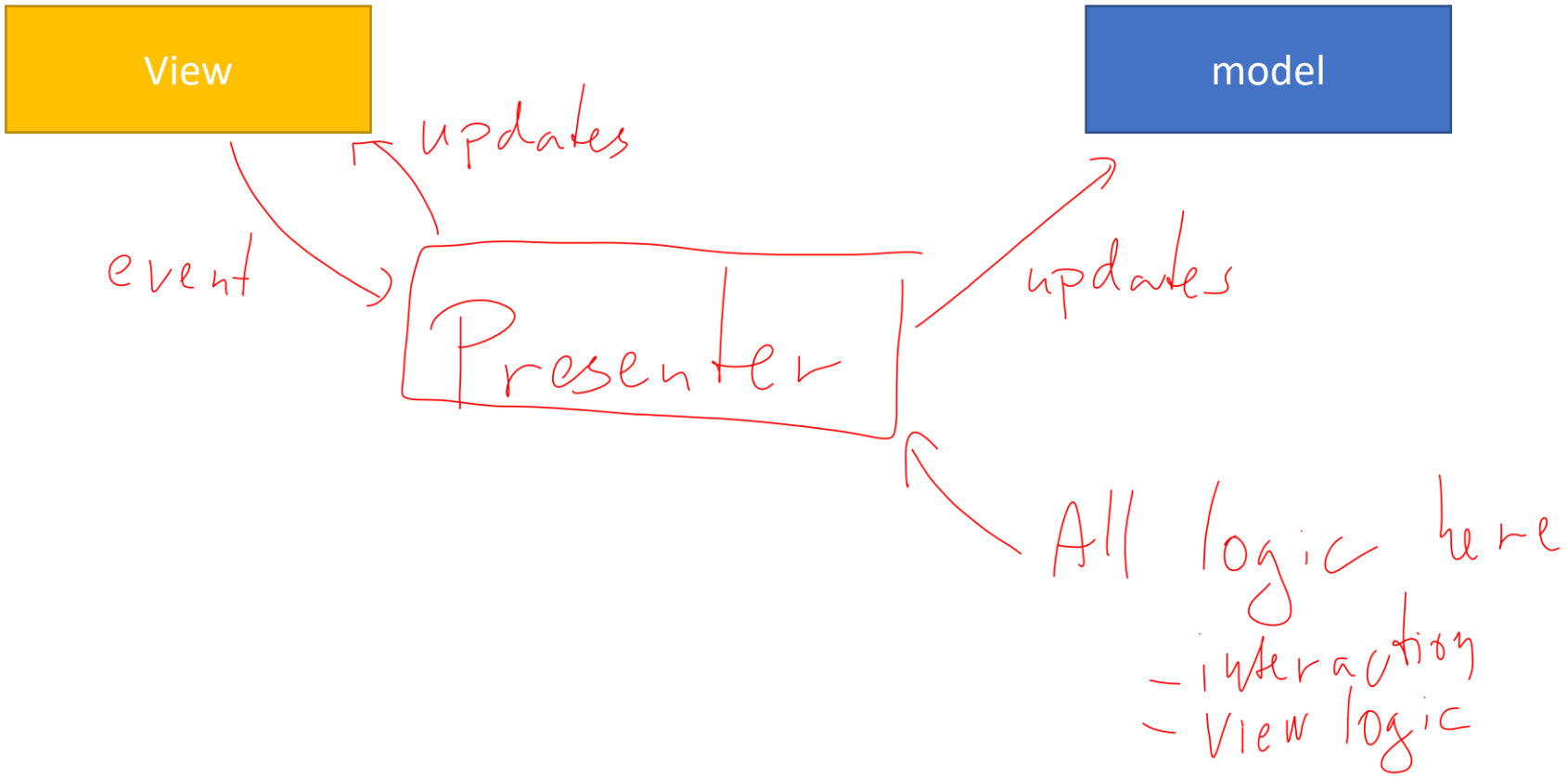
Model/view separation

- Single source of truth for any value in the UI (model/state)
- Helps with:
  - Consistency
  - Single responsibility
- What's between the model and the view?
  - Controller (MVC)
  - Presenter (MVP)
  - Model-View-ViewModel (MVVM)
    - Binding

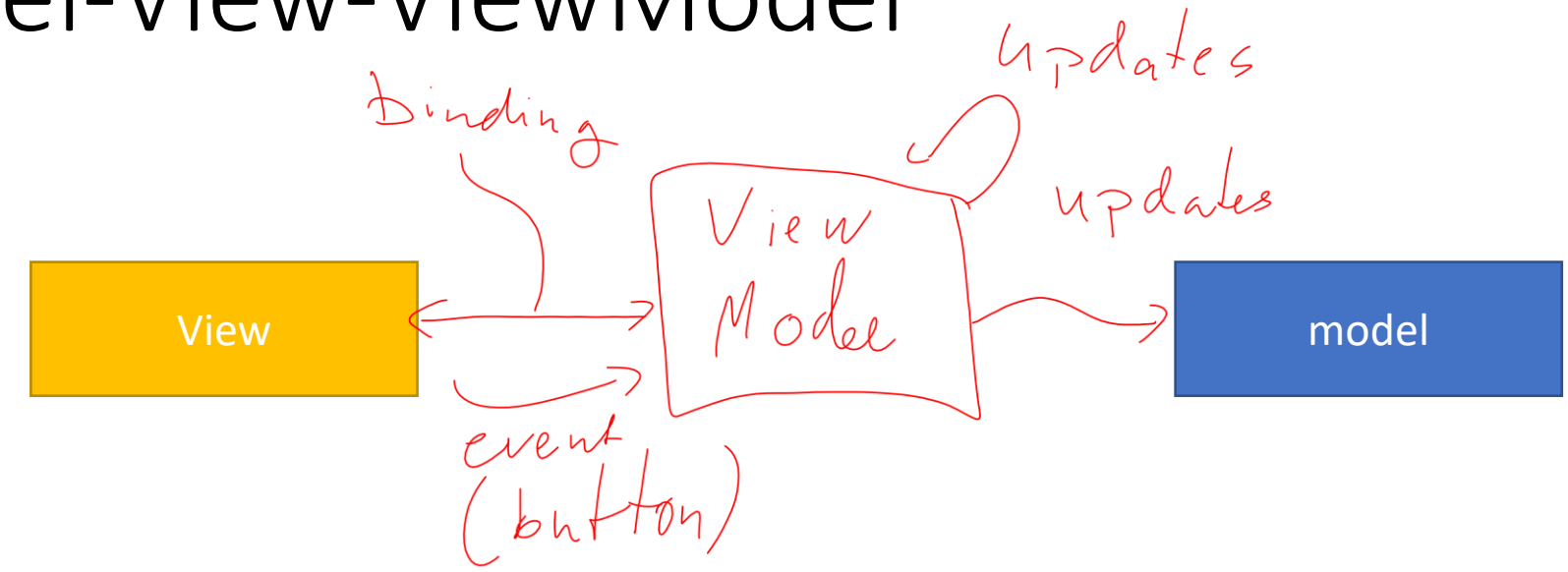
Model-View-Controller



Model-View-Presenter



Model-View-ViewModel



Binding

- Single vs double-bind
  - Single: Something in the view that reflects the VM
  - Double: Something in the view that both reflects the VM and updates the VM
- How?
  - Use an observer
    - JavaFX properties
    - Vue.js
    - DIY
  - Update after each call to a method that might change VM
    - Angular

Vue js

- View model
  - Options API
    - Create a view model as a JavaScript object
    - It needs certain predefined properties
    - This is the classic way - recommended for beginners
  - Composition API
    - Use some Vue functions that implicitly define the view model
    - Recommended for new projects and experienced developers
- Template language
  - HTML with some extra attributes and other features
- Both are defined in *.vue files (as of version 3)

View Model (Options API)

- In vue.js a view model consists of
  - data (properties)
  - computed properties
  - methods
  - …
- These are called *options*
- The view model is returned from the <script> section of the .vue file

A view model (from options/reverse/App.vue)

<script>

```
export default defineComponent({
    data() {
      return {
        s: "reverse"
      }
    },
    computed: {
      reversed: {
        get(): string { return reverse(this.s)},
        set(v: string): void { this.s = reverse(v) }
      }
    }
  })
```

</script>

initial data

current data object

The data() method

- The data method returns the initial value of the state
- Vue.js wraps creates observers on the state
- Triggers *re-renders* every time the state change
- This is called a *reactive* state
- In the view model the state is referred to by `this` - like `this.s` on the previous slide.

Computed properties

- Properties derived from the state
- May trigger re-render

```
computed: {
  reversed: {
    get(): string { return reverse(this.s)},
    set(v: string): void { this.s = reverse(v) }
  }
}
```

triggers re-render

Template

```
<template>
  <div id='base'>
    <input v-model='s' id='normal'>
    <input v-model='reversed' id='reversed'>
  </div>
</template>
```

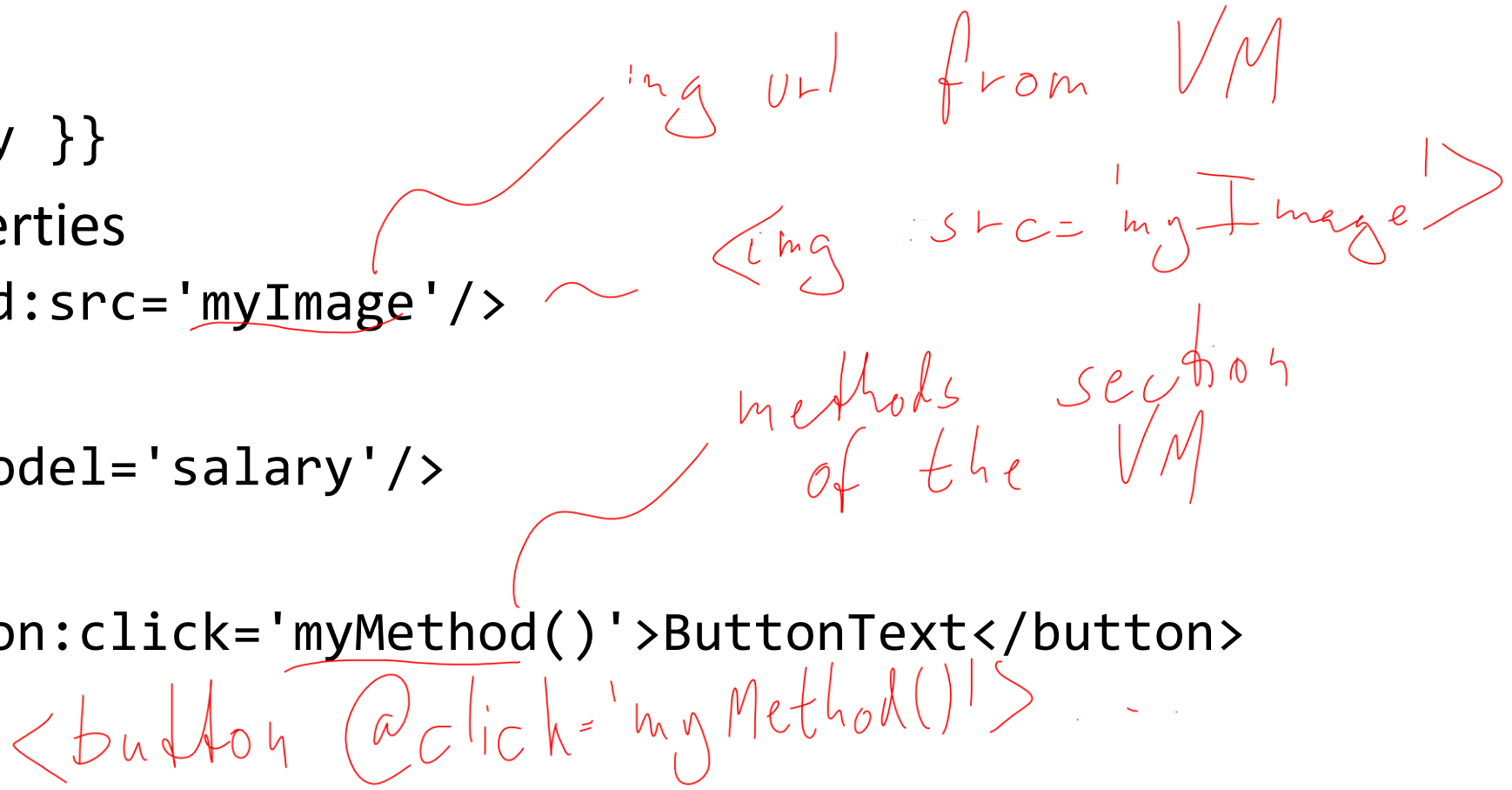
properties of the data + computed

Double bind

type in 'reversed' input → reversed.set(..) → this.s changes → re-render

Binding in the template language

- 1-way
  - `{{ property }}`
- 1-way w/ properties
  - `<img v-bind:src='myImage'/>`
- 2-way
  - `<input v-model='salary'/>`
- events
  - `<button v-on:click='myMethod()'>ButtonText</button>`



Method example (options/example/App.vue)

```
export default defineComponent({
  data: () => ({
    model: createModel(…)
    salary: 0,
    error: undefined
  }),
  …
```

```
  methods: {
   hire(id: number) {
    const person
     = this.model.personById(id)
    this.model.hire(person, this.salary)
    this.salary = 0
    this.error = undefined
   }
  }
})
```

Template - see options/example/App.vue

Snippets:

repeats this tag

```
<tr v-for='p in personData'>
```

p is in scope

```
</tr>
```

```
<button v-on:click='hire(p.id)'>Hire</button>
```

```
<div v-if="error !== undefined">{{error}}</div>
```

Control Structures

- `v-for`
  - Add as an attribute to indicate that the tag repeats for each element in an array. For instance, use it with `<tr>`, `<li>`, and of course `<div>`
- `v-if/v-else`
  - Display element if true, don’t if false.
- `<template>`
  - Use this to group elements that you can't group with anything else

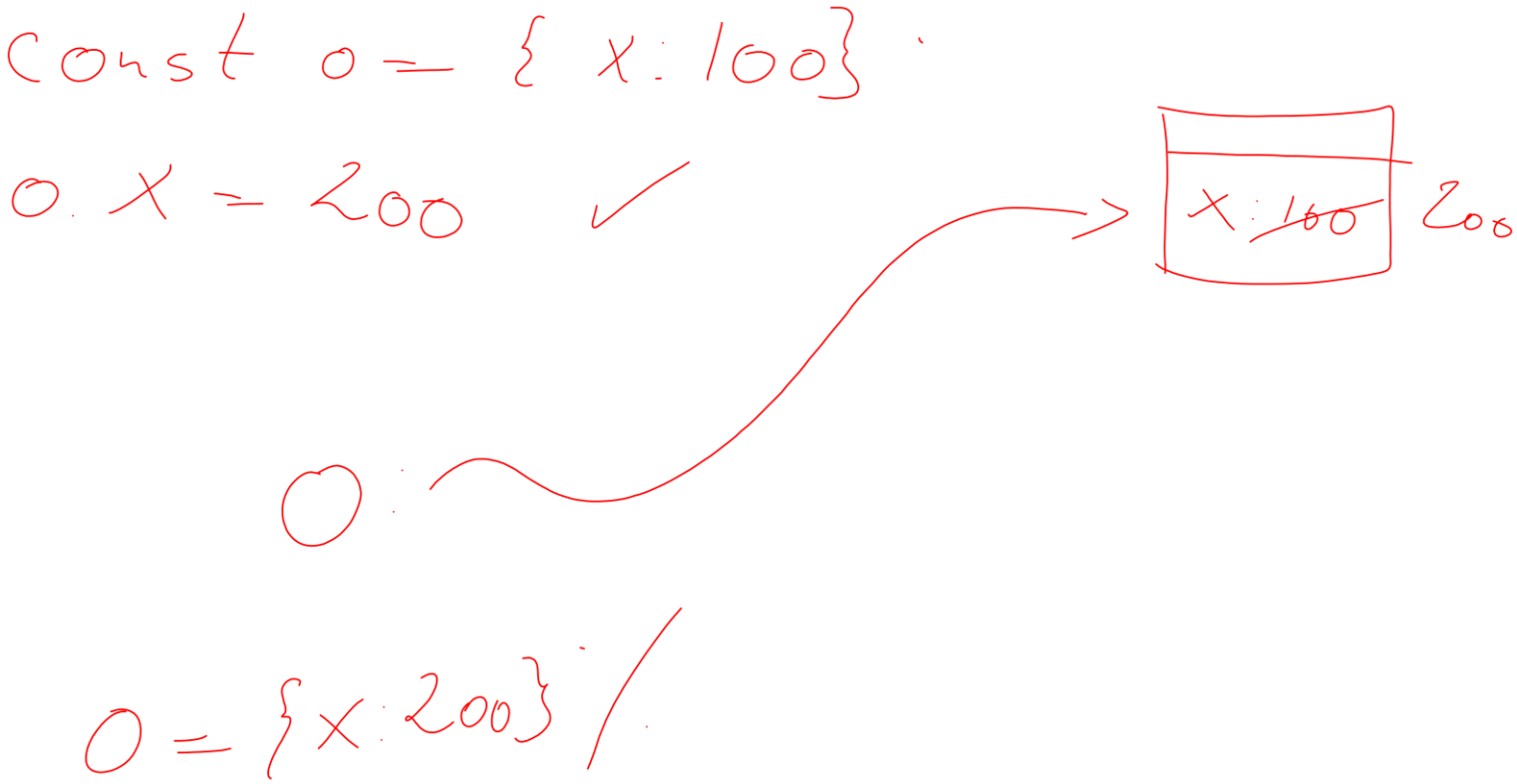
Composition API

- Composition vs options API is a question about how you will define the *view model*
  - Only the `<script>` part changes
- Denoting composition vs options API:
  - Options: `<script> / <script lang="ts">`
  - Composition: `<script setup> / <script setup lang="ts">`
- Choosing compositions vs options
  - Options has a strict structure - an object of a certain shape
  - Compositions has looser requirements
- The main difference is how you create *reactive* objects

Reactive objects

- A *reactive* object is an object that triggers a *re-rendering* when it changes
- Vue.js instruments the object with observers
- In options API:
  - Vue.js takes the object returned from `data()` and makes it reactive
  - You can access the object using this in the rest of the code
  - In the template you can access the properties of the reactive object directly
- In composition API:
  - You create a reactive object directly using `reactive()` or `ref()`
  - You access the properties through the object in both other code and template

State change: Mutable Object



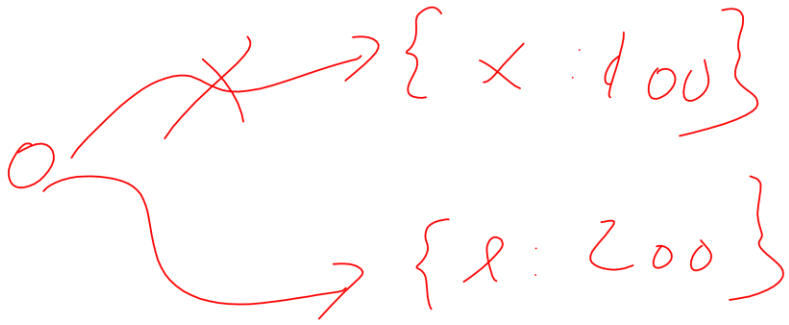
State change: changing variables

let i = 7

let o = {x: 100}

i++ // 8

o = {x: 200}

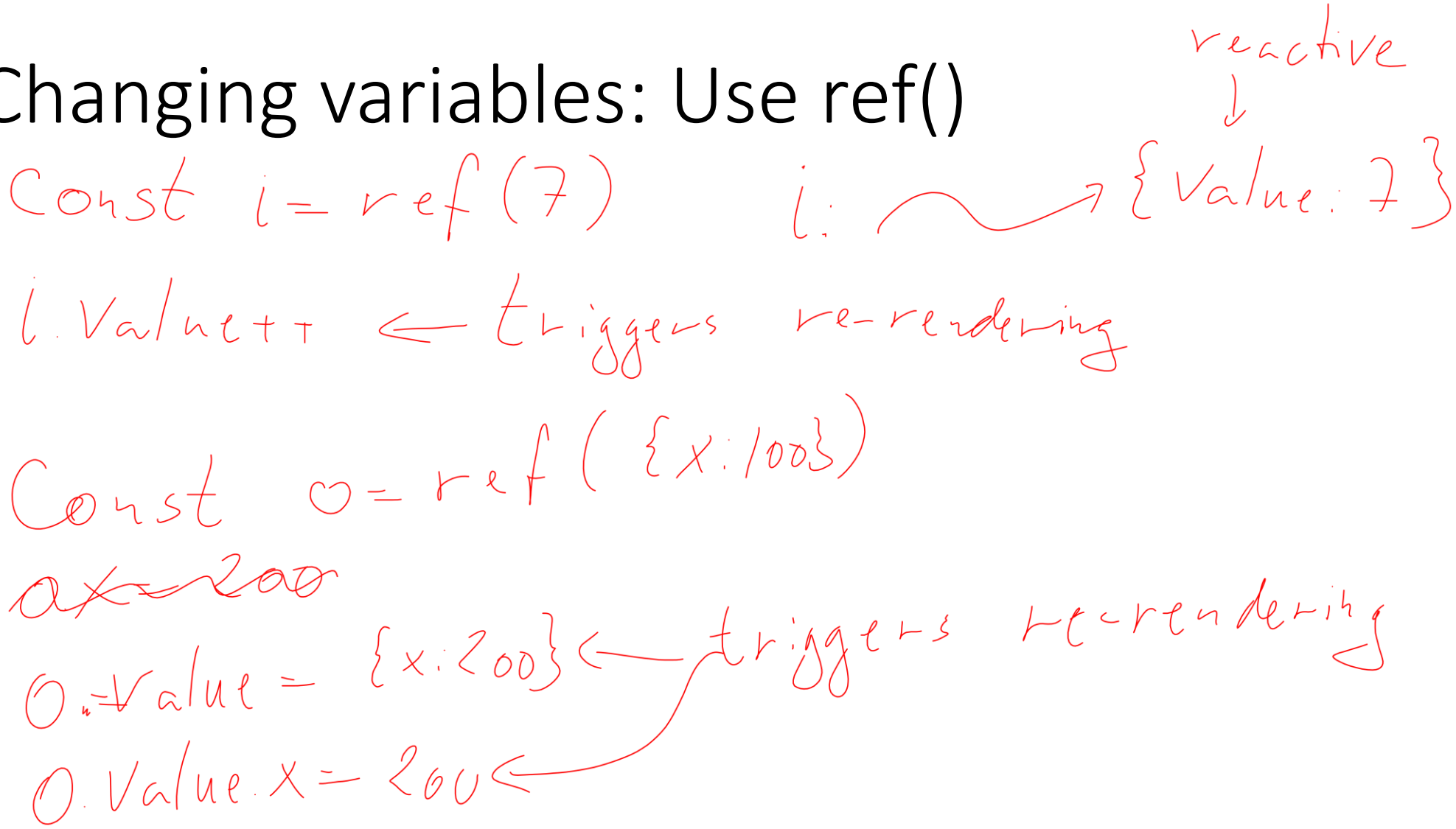


Mutable objects: reactive() is enough

const o = reactive({x: 100})

o.x = 200 ← triggers a re-render

Changing variables: Use ref()



Example

```
const s = ref("reverse")
const reversed = computed({
  get(): string { return reverse(s.value)},
  set(v: string): void { s.value = reverse(v) }
})

<input v-model='s' id='normal'>
<input v-model='reversed' id='reversed'>
```

unchanged

reactive()

```
const model = reactive(createModel(persons, employees))
```

Methods are just functions

```
function hire(id: number) {
  const person = model.personById(id)
  model.hire(person, this.salary)
  salary.value = 0
  error.value = undefined
}

<button v-on:click='hire(p.id)'>Hire</button>
```