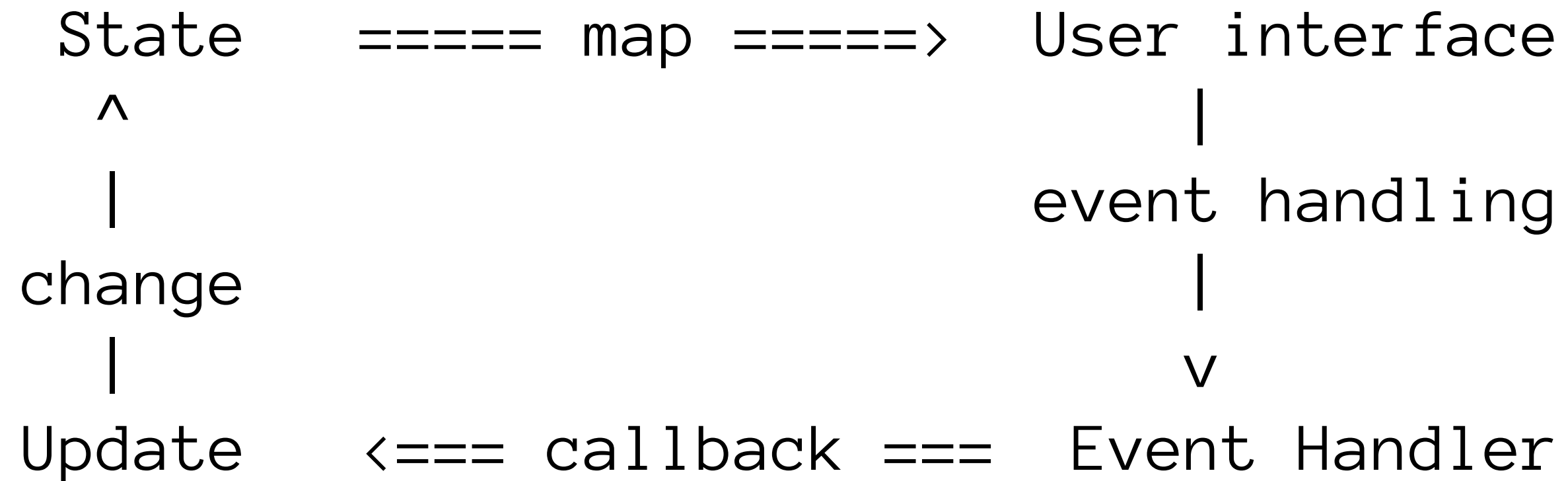


User Interface as State

Mental Model



Simple Example

```
<p>0</p>
```

```
<button>Increment</button>
```

```
let counter = 0
```

```
document.querySelector('button').addEventListener('click', function (event) {  
  counter++
```

```
  const counterElement = document.querySelector('p')  
  counterElement.textContent = counter  
})
```

Turn state into user interface

In the example we are using a *local* variable to track the *state* of the count and then *manually* updating the user interface.

What if the user interface was simply a representation of state?

More complex example

Empty HTML

```
let counter = 0

function render() {
  const html = `
  <p>${counter}</p>
  <button>Increment</button>
  `

  document.body.innerHTML = html

  // After we create the HTML we can
  // now setup our listener for clicks
  document.querySelector('button').addEventListener('click', function (event) {
    counter++

    render()
  })
}

render()
```

State change => HTML refresh

Every time we update counter we repaint the entire user interface.

More complex
example: scoreboard

Scoreboard

Take contents of the <body>

HTML

CSS

```
import './style.css'

function render() {
  const html = `
<header><h1>My Score Board</h1></header>
<main>
  <section class="team1">
    <h2>Team 1</h2>
    <h3>0</h3>
    <fieldset><input type="text" placeholder="Name" /></fieldset>

    <fieldset><i class="add fas fa-2x fa-plus-circle"></i><i class="subtract fas fa-2x fa-minus-circle"></i></fieldset>
  </section>

  <section class="team2">
    <h2>Team 2</h2>
    <h3>0</h3>
    <fieldset><input type="text" placeholder="Name" /></fieldset>

    <fieldset><i class="add fas fa-2x fa-plus-circle"></i><i class="subtract fas fa-2x fa-minus-circle"></i></fieldset>
  </section>
</main>`

  document.body.innerHTML = html

  // Setup event listeners here
}

render()
```

Setup state

```
let teamOneName = 'Team 1'
```

```
let teamOneScore = 0
```

```
let teamTwoName = 'Team 2'
```

```
let teamTwoScore = 0
```

Update render to use variables

Use string interpolation.

Change static text such as:

```
<h2>Team 1</h2>  
<h3>0</h3>  
<fieldset><input type="text" placeholder="Name" /></fieldset>
```

to

```
<h2>${teamOneName}</h2>  
<h3>${teamOneScore}</h3>  
<fieldset>  
  <input type="text" placeholder="Name" value="${teamOneName}" />  
</fieldset>
```

See that if we change our initial state the UI changes!

Seemingly simple and innocuous. However, this is a powerful idea.

Create event listeners

The event listening functions should update the appropriate variable and call render

```
document.querySelector('.team1 .add')?.addEventListener('click', function () {
  teamOneScore++
  render()
})
document
  .querySelector('.team1 .subtract')
  ?.addEventListener('click', function () {
    teamOneScore--
    render()
  })
document
  .querySelector('.team1 input')
  ?.addEventListener('input', function (event) {
    const target = event.target as HTMLInputElement

    teamOneName = target?.value
    render()
  })

document.querySelector('.team2 .add')?.addEventListener('click', function () {
  teamTwoScore++
  render()
})
document
  .querySelector('.team2 .subtract')
  ?.addEventListener('click', function () {
    teamTwoScore--
    render()
  })
document
  .querySelector('.team2 input')
  ?.addEventListener('input', function (event) {
    const target = event.target as HTMLInputElement

    teamTwoName = target?.value
    render()
  })
```

Change state to be an object for each team

```
interface Team {  
  name: string  
  score: number  
}  
  
const teamOne: Team = {  
  name: 'Team 1',  
  score: 0,  
}  
  
const teamTwo: Team = {  
  name: 'Team 2',  
  score: 0,  
}
```

Change render function and handlers

Change:

```
<h2>${teamOneName}</h2>
```

to

```
<h2>${teamOne.name}</h2>
```

Extract out a method to render a team and pass in the state

- Add an id to each team object
- Update render to call that function
- Move query listeners

Render

```
function renderTeam(team) {
  const html = `
  <section class="team${team.id}">
  <h2>${team.name}</h2>
  <h3>${team.score}</h3>
  <fieldset>
    <input type="text" placeholder="Name" value="${team.name}" />
  </fieldset>

  <fieldset>
    <i class="add fas fa-2x fa-plus-circle"></i>
    <i class="subtract fas fa-2x fa-minus-circle"></i>
  </fieldset>
</section>
`

  return html
}
```

Listeners

```
function setupListeners(team: Team) {
  document
    .querySelector(`.team${team.id} .add`)
    ?.addEventListener('click', function () {
      team.score++
      render()
    })
  document
    .querySelector(`.team${team.id} .subtract`)
    ?.addEventListener('click', function () {
      team.score--
      render()
    })
  document
    .querySelector(`.team${team.id} input`)
    ?.addEventListener('input', function (event) {
      const target = event.target as HTMLInputElement
      team.name = target?.value
      render()
    })
}
```

```
function render() {  
  const html = `  
<header><h1>My Score Board</h1></header>  
<main>  
${renderTeam(teamOne)}  
${renderTeam(teamTwo)}  
</main>`  
  
  document.body.innerHTML = html  
  setupListeners(teamOne)  
  setupListeners(teamTwo)  
}
```

Using Arrays

- Change state to be an array
- Change render to iterate over the array
- Change event handlers to use the supplied element of the array

State

```
const teams: Team[] = [  
  {  
    id: 1,  
    name: 'Team 1',  
    score: 0,  
  },  
  
  {  
    id: 2,  
    name: 'Team 2',  
    score: 0,  
  },  
]
```

Render

```
function render() {  
  const html = `  
    <header>  
      <h1>My Score Board</h1>  
    </header>  
    <main>  
    ${teams  
      .map(function (team: Team) {  
        return renderTeam(team)  
      })  
      .join('')}  
    </main>  
  `
```

```
  document.body.innerHTML = html
```

```
  teams.forEach(function (team: Team) {  
    setupListeners(team)  
  })  
}
```

Add more teams!

```
{  
  id: 3,  
  name: 'Team 3',  
  score: 10,  
},
```

Add a reset button

```
<footer>  
  <button>reset</button>  
</footer>
```

```
footer {  
  display: flex;  
  justify-content: center;  
}
```

```
footer button {  
  font-size: 3rem;  
  text-transform: uppercase;  
}
```

```
document.querySelector('button')?.addEventListener('click', function (event) {  
  // Reset the teams  
  teams = [  
    { id: 1, name: 'Team 1', score: 0 },  
    { id: 2, name: 'Team 2', score: 0 },  
    { id: 3, name: 'Team 3', score: 0 },  
  ]  
  
  render()  
})
```

**Invented a low-
fidelity version of
react**