Reactive State Machines And Statecharts

@davidkpiano · React Finland 2018

loading

isLoading

`loading`

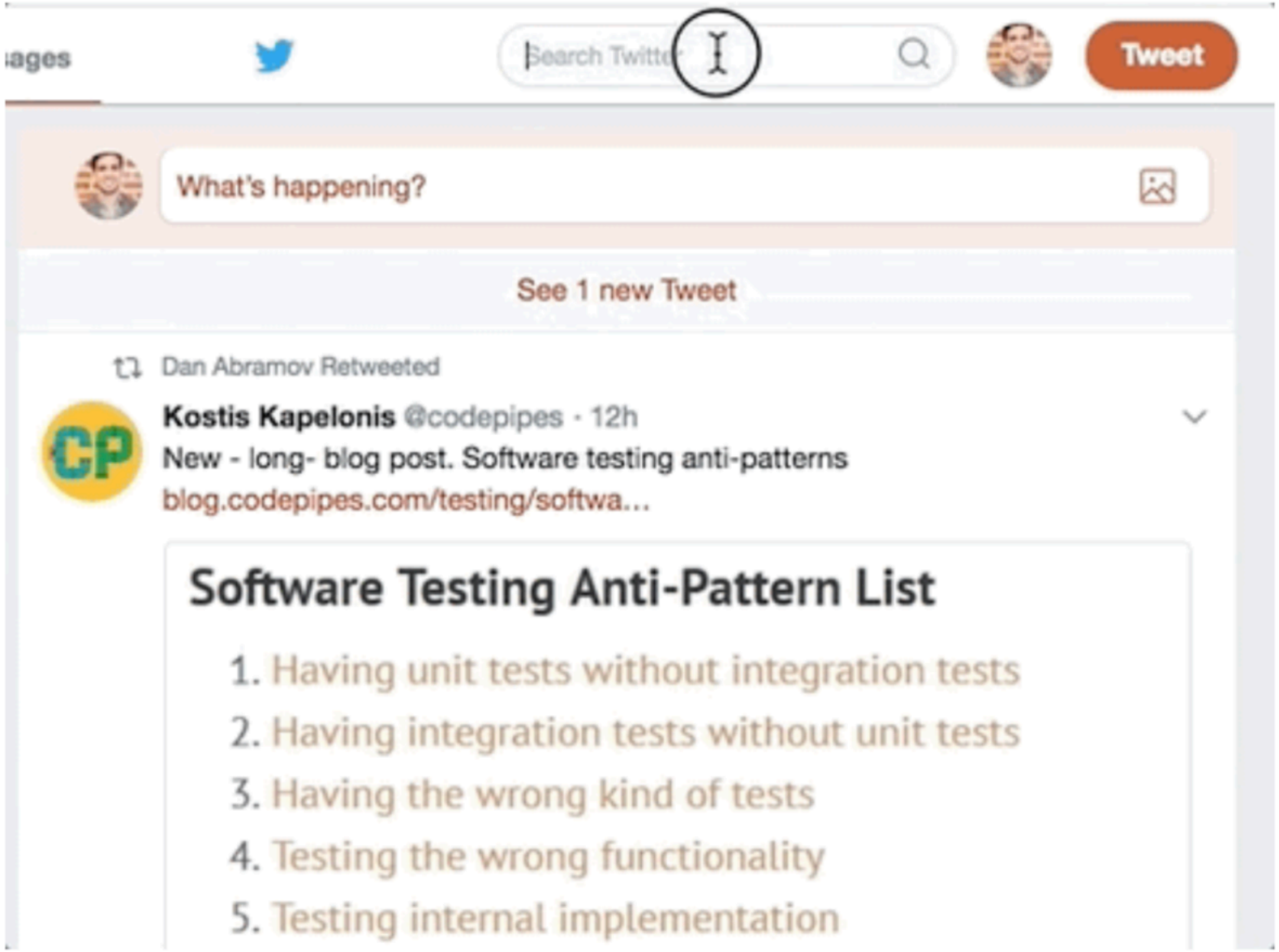
`isLoading`

shame on you

Missile Warning System

Send Actual Warning

Send Test Warning


ages
Search Twitter
Tweet
What's happening?
See 1 new Tweet
Dan Abramov Retweeted
Kostis Kapelonis @codepipes · 12h
New - long- blog post. Software testing anti-patterns
blog.codepipes.com/testing/softwa...
Software Testing Anti-Pattern List
1. Having unit tests without integration tests
2. Having integration tests without unit tests
3. Having the wrong kind of tests
4. Testing the wrong functionality
5. Testing internal implementation

The most neglected variable is **time**.

Callbacks

Promises

Async-Await

Observables

Callbacks

Promises

Async-Await

Observables

Callbacks

Promises

Async-Await

Observables

Callbacks

Promises

Async-Await

Observables

Callbacks

Promises

Async-Await

Observables

Callbacks

Promises

Async-Await

Observables

Callbacks

Promises

Async-Await

Observables

Callbacks

Promises

Async-Await

Observables

Callbacks

Promises

Async-Await

Observables

> *Any sufficiently complicated model class contains an ad-hoc, informally-specified, bug-ridden, slow implementation of* ***half of a state machine****.*

How I Learned to Stop Worrying and ❤️ the State Machine - Reginald Braithwaite

state management library

> *Any sufficiently complicated ~~model class~~ contains an ad-hoc, informally-specified, bug-ridden, slow implementation of* ***half of a state machine****.*

How I Learned to Stop Worrying and ❤️ the State Machine - Reginald Braithwaite

#Frameworkless

How can we model the **behavior** of user interfaces?

API

API

documented

API

documented

predictable

API

documented

predictable

testable

HumanAPI

documented

predictable

testable

HumanAPI

undocumented

predictable

testable

HumanAPI

undocumented

unpredictable

testable

HumanAPI

undocumented

unpredictable

untestable

```
// ...
onSearch(query) {
  fetch(FLICKR_API + '&tags=' + query)
    .then(data => this.setState({ data }));
}
// ...
```

Show data when results retrieved

```
// ...
onSearch(query) {
  this.setState({ loading: true });

  fetch(FLICKR_API + '&tags=' + query)
    .then(data => {
      this.setState({ data, loading: false });
    });
}
// ...
```

Show loading screen

Hide loading screen
Show data when results retrieved

```
// ...
onSearch(query) {
  this.setState({ loading: true });

  fetch(FLICKR_API + '&tags=' + query)
    .then(data => {
      this.setState({ data, loading: false });
    })
    .catch(error => {
      this.setState({
        loading: false,
        error: true
      });
    });
}
// ...
```

Show loading screen

Hide loading screen
Show data when results retrieved

Hide loading screen
Show error

```
// ...
onSearch(query) {
  this.setState({
    loading: true,
    error: false
  });

  fetch(FLICKR_API + '&tags=' + query)
    .then(data => {
      this.setState({
        data,
        loading: false,
        error: false
      });
    })
    .catch(error => {
      this.setState({
        loading: false,
        error: true
      });
    });
}
// ...
```

Show loading screen
Hide error

Hide loading screen
Show data when results retrieved
Hide error

Hide loading screen
Show error

```
// ...
onSearch(query) {
  if (this.state.loading) return;

  this.setState({
    loading: true,
    error: false,
    canceled: false
  });

  fetch(FLICKR_API + '&tags=' + query)
    .then(data => {
      if (this.state.canceled) {
        return;
      }

      this.setState({
        data,
        loading: false,
        error: false
      });
    })
    .catch(error => {
      // mitä vittua
      if (this.state.canceled) {
        return;
      }

      this.setState({
        loading: false,
        error: true
      });
    });
}

onCancel() {
  this.setState({
    loading: false,
    error: false,
    canceled: true
  });
}
```

Search in progress already
Show loading screen
Hide error
Cancel cancellation
Ignore results if cancelled
Hide loading screen
Show data when results retrieved
Hide error

Ignore error if cancelled
Hide loading screen
Show error

Cancel search

```
      error: false
    });
})
.catch(error => {
    // mitä vittua
    if (this.state.ca
      return;
    }
```

Search in progress already
Show loading screen
Hide error
Cancel cancellation
Ignore results if cancelled
Hide loading screen
Show data when results retrieved
Hide error

Ignore error if cancelled
Hide loading screen
Show error

Cancel search

🍑 🆙

The bottom-up approach

🍑 🆙

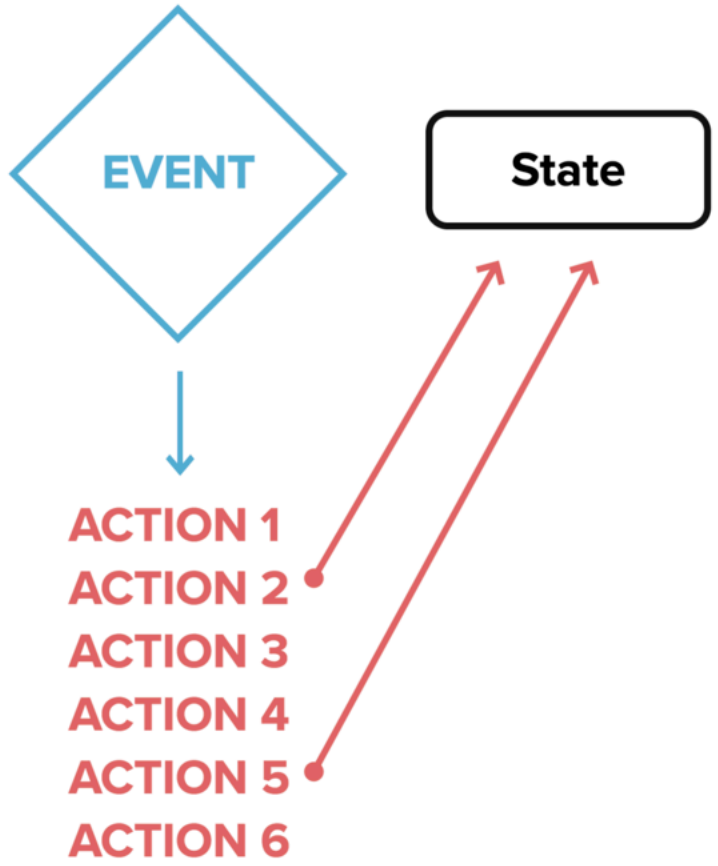
The bottom-up approach

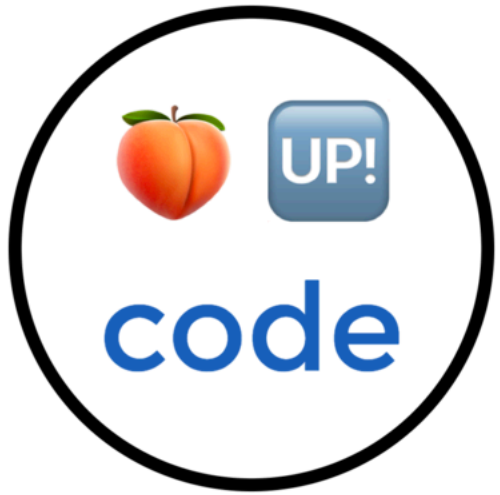
🍑 🆙

The bottom-up approach

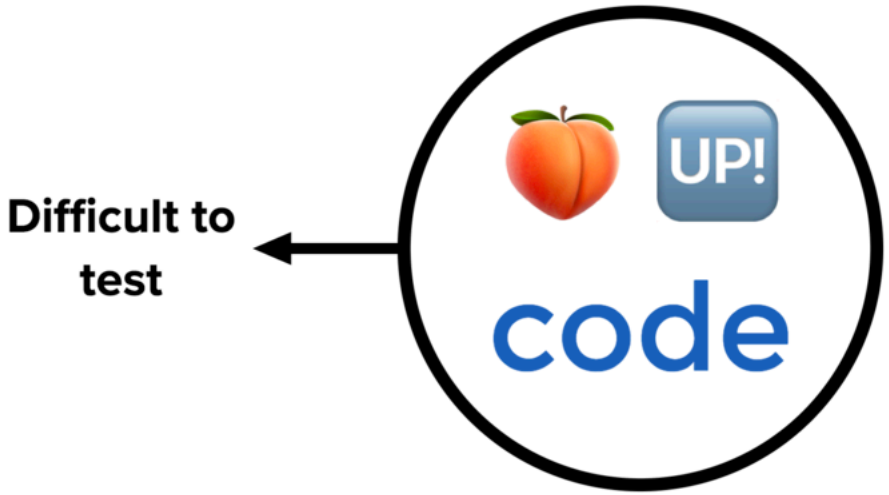
🍑 🆙

The bottom-up approach

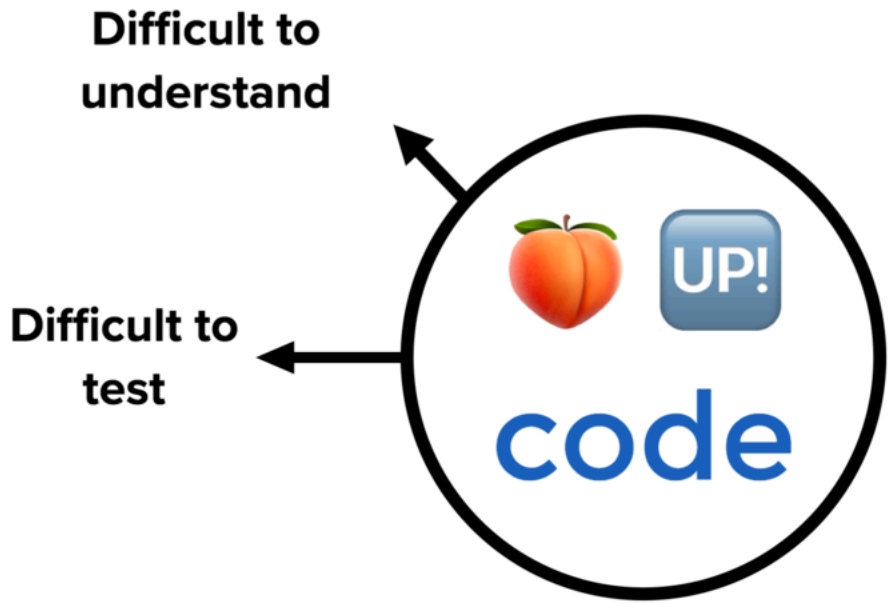




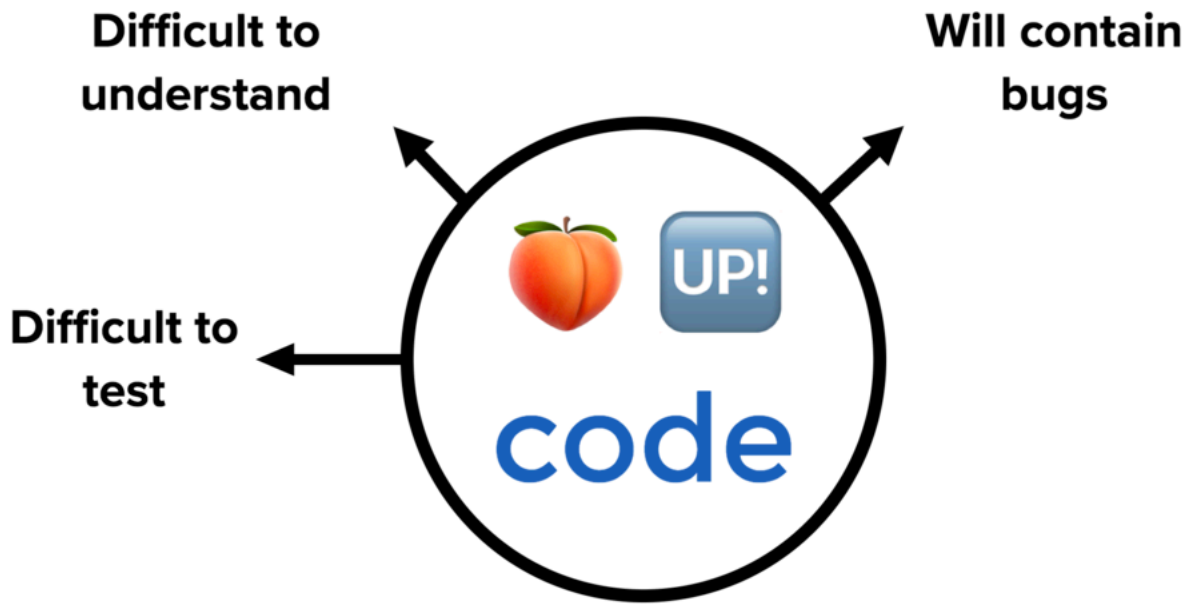
Source: Ian Horrocks, "Constructing the User Interface with Statecharts", ch. 3 pg. 17



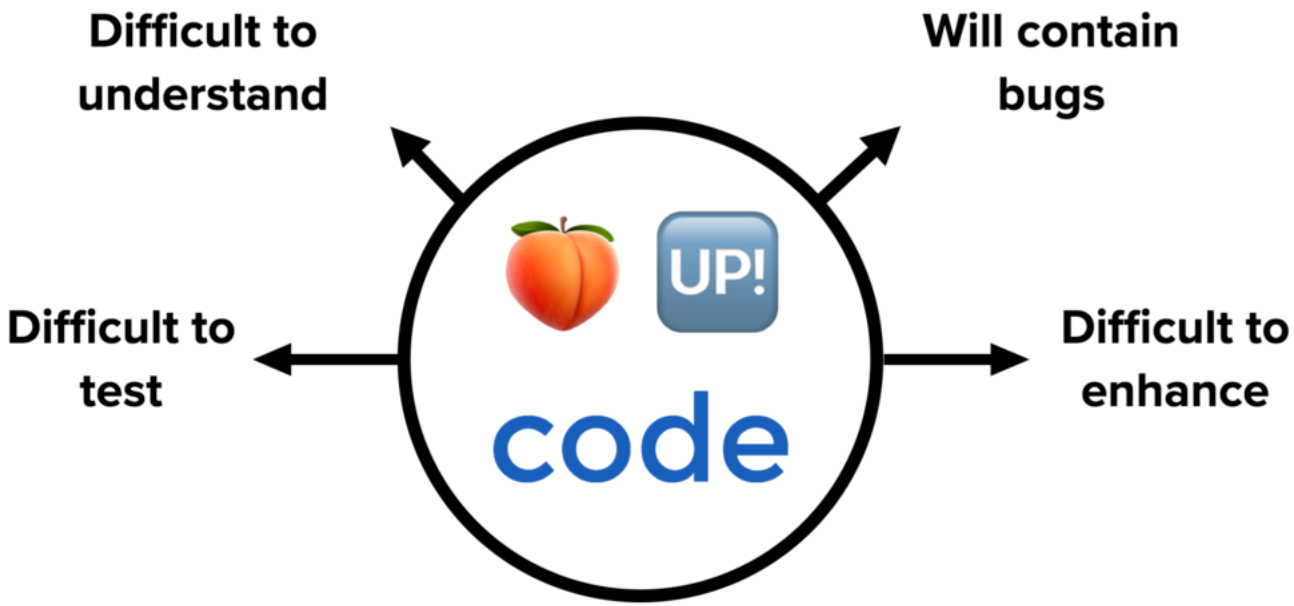
Source: Ian Horrocks, "Constructing the User Interface with Statecharts", ch. 3 pg. 17



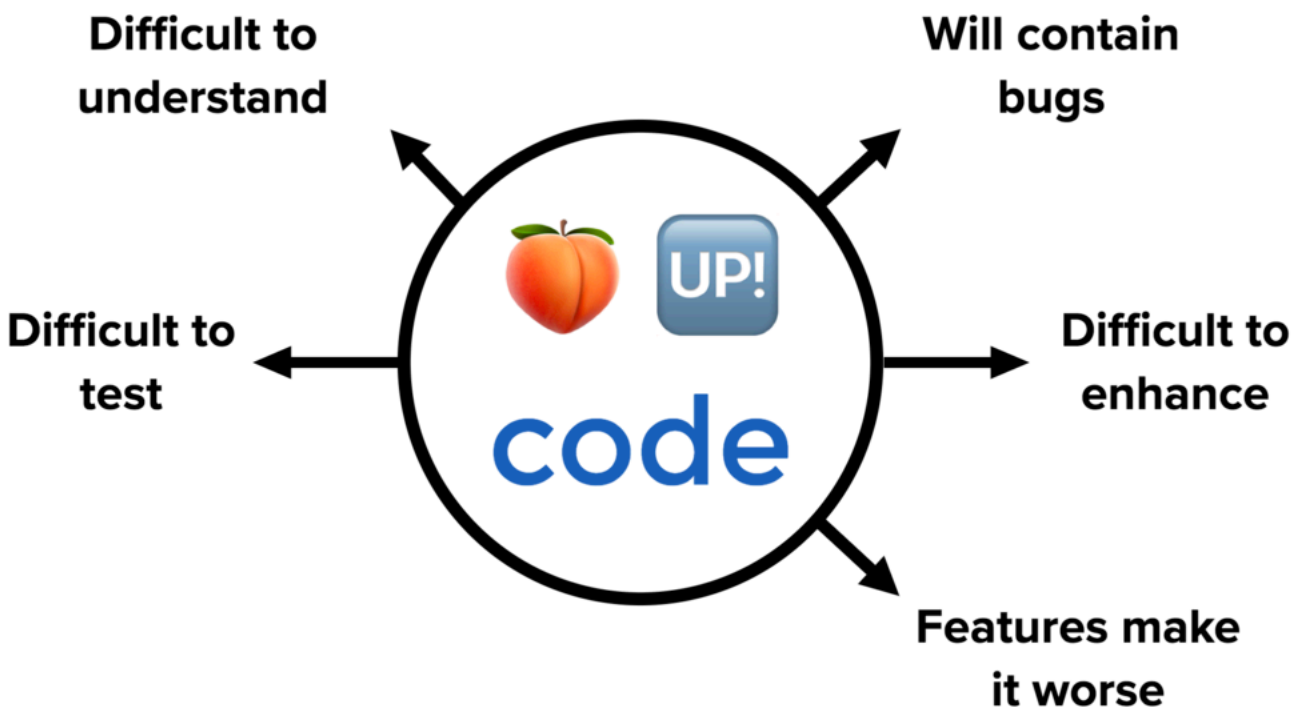
Source: Ian Horrocks, "Constructing the User Interface with Statecharts", ch. 3 pg. 17



Source: Ian Horrocks, "Constructing the User Interface with Statecharts", ch. 3 pg. 17

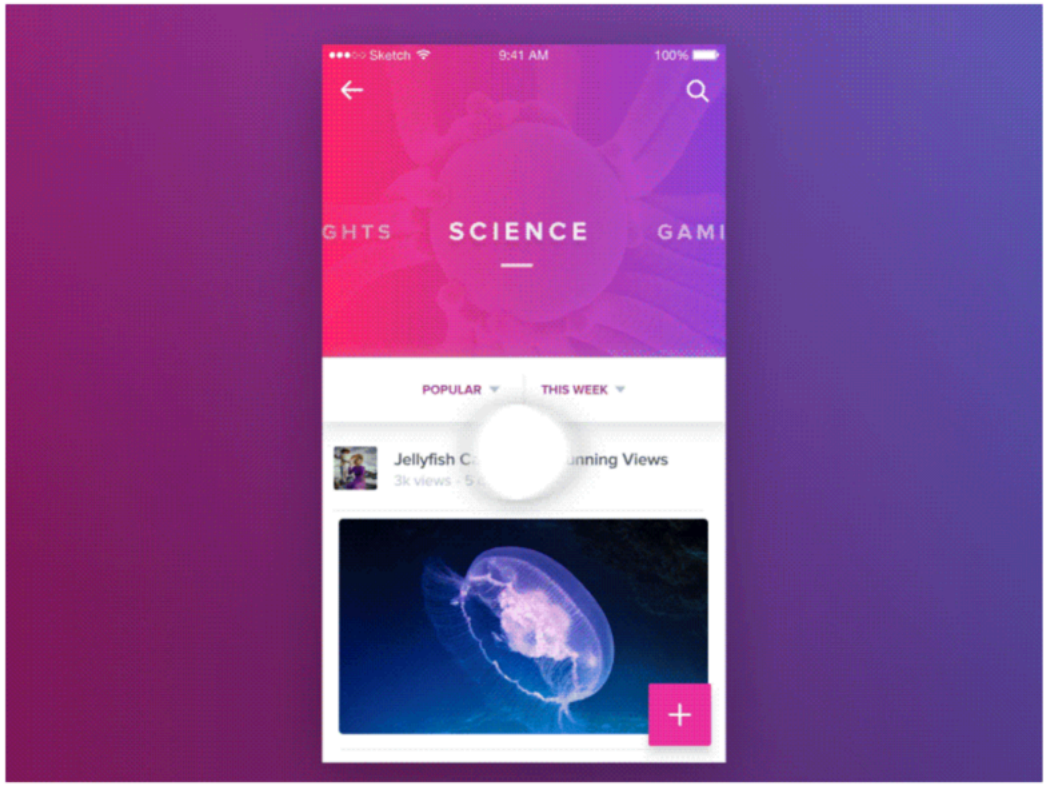


Source: Ian Horrocks, "Constructing the User Interface with Statecharts", ch. 3 pg. 17



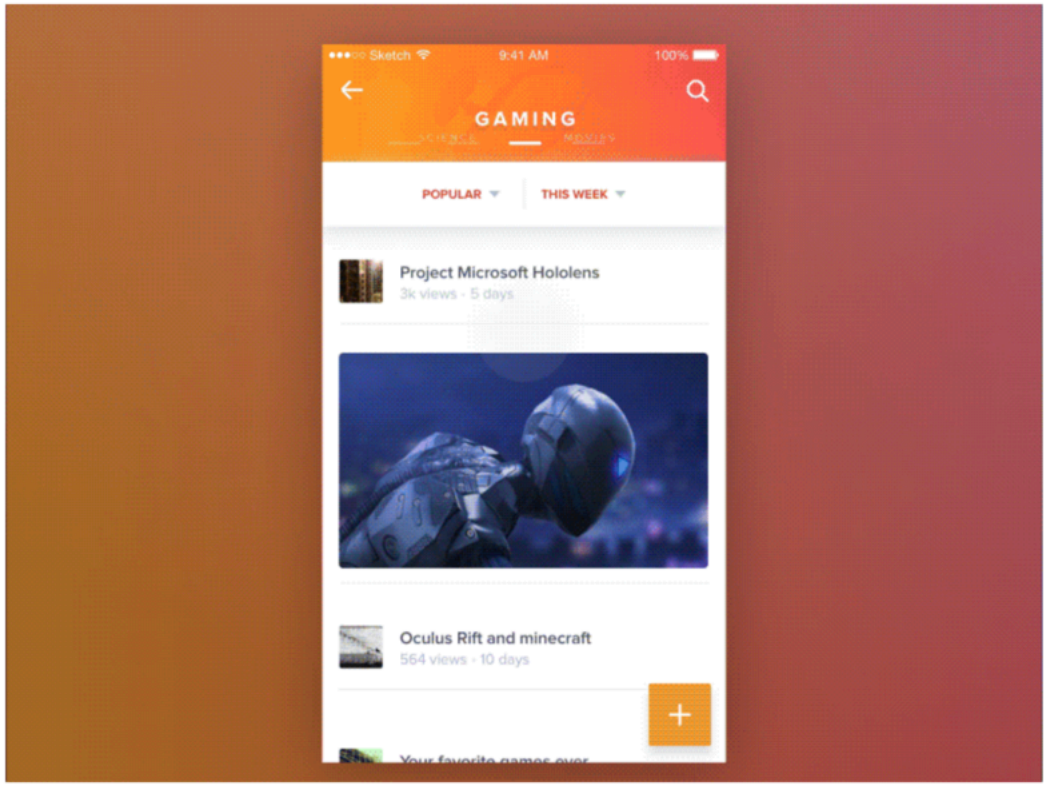
Source: Ian Horrocks, "Constructing the User Interface with Statecharts", ch. 3 pg. 17

Intuition


Sketch
9:41 AM
100%
GHTS
SCIENCE
GAMI
POPULAR
THIS WEEK
Jellyfish C
nning Views
3k views

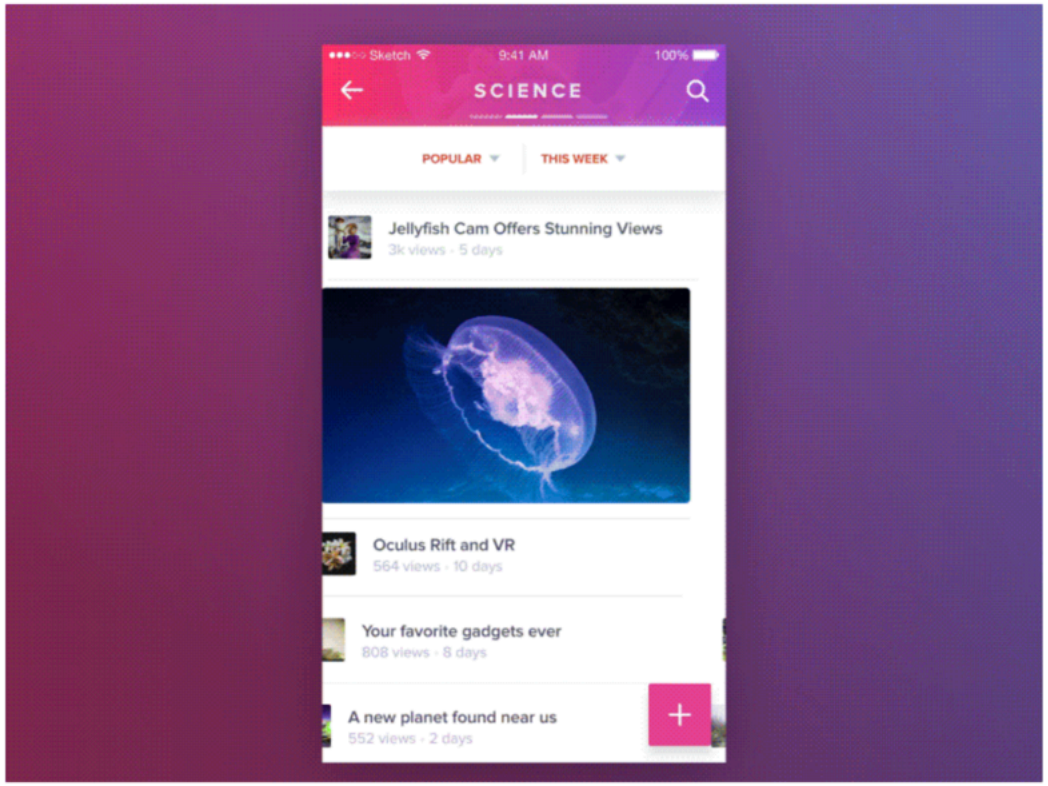
Intuition

UI components are **not independent**



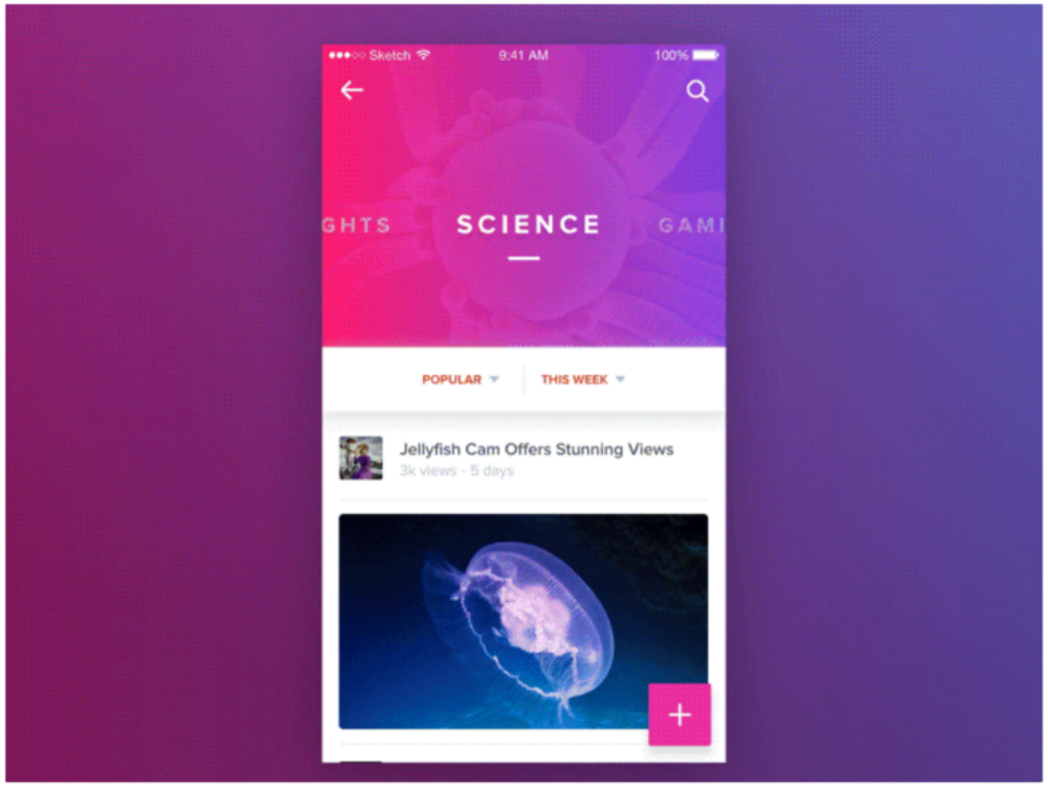
Intuition

UI components are **not independent**
Actions are based on **event & state**



Intuition

UI components are **not independent**
Actions are based on **event & state**
The event-action paradigm is **too simple**



Finite state machines and statecharts

Mathematical model [edit]

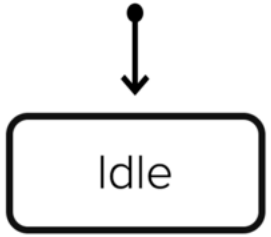
In accordance with the general classification, the following formal definitions are f

- A *deterministic finite state machine* or *acceptor deterministic finite state mach*
  $(\Sigma, S, s_0, \delta, F)$ where:
  - Σ is the input alphabet (a finite, non-empty set of symbols).
  - S is a finite, non-empty set of states.
  - s_0 is an initial state, an element of S.
  - δ is the state-transition function: $\delta : S \times \Sigma \to S$ (in a nondeterministic fi
    would be $\delta : S \times \Sigma \to \mathcal{P}(S)$, i.e., δ would return a set of states).
  - F is the set of final states, a (possibly empty) subset of S.

Finite state machines

Finite state machines

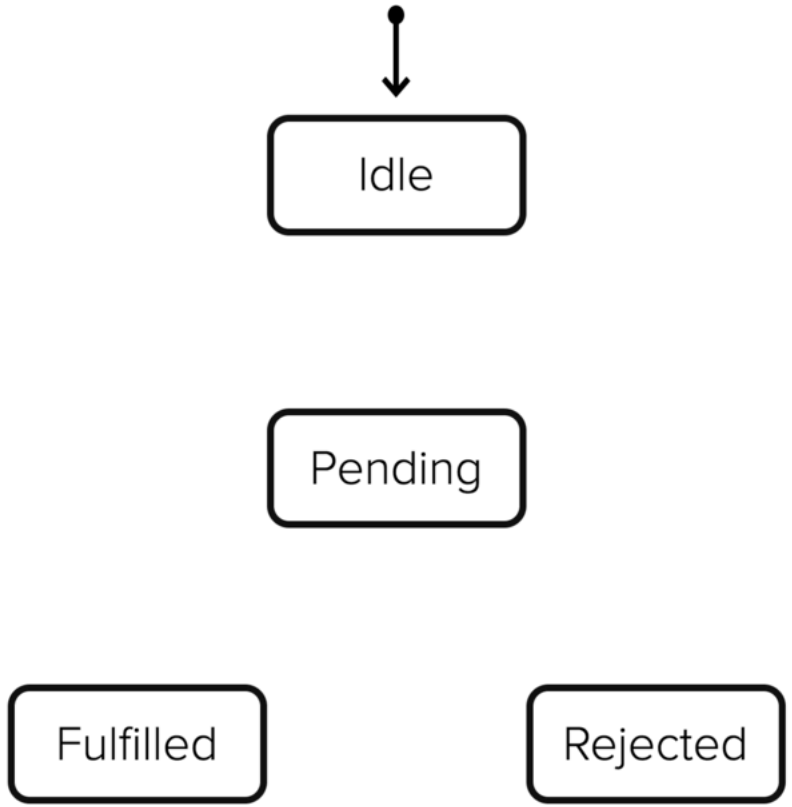
have one **initial state**



Finite state machines

have one **initial state**

a finite number of **states**

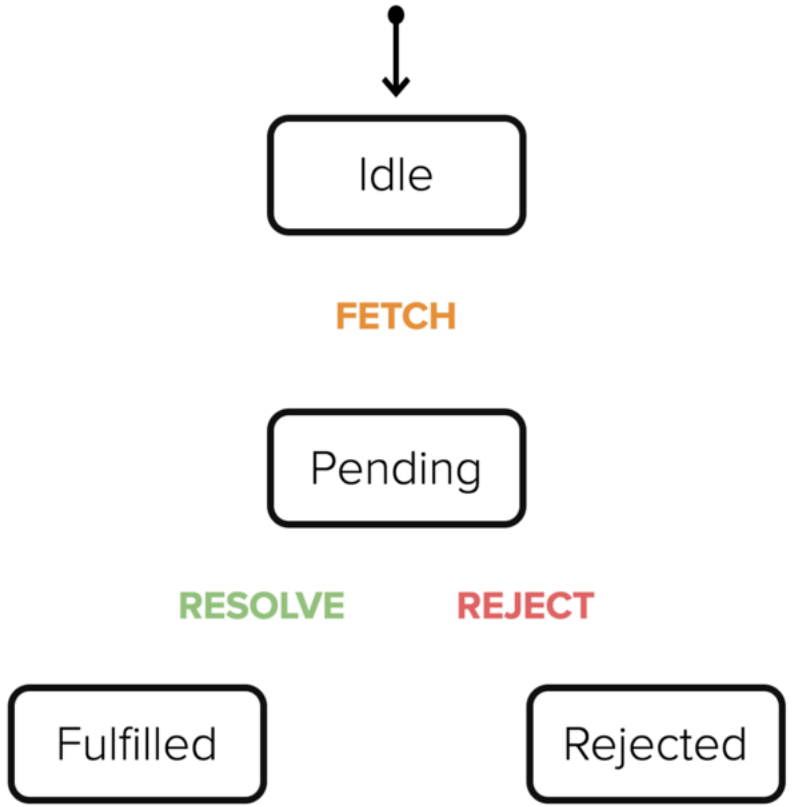


Finite state machines

have one **initial state**

a finite number of **states**

a finite number of **events**



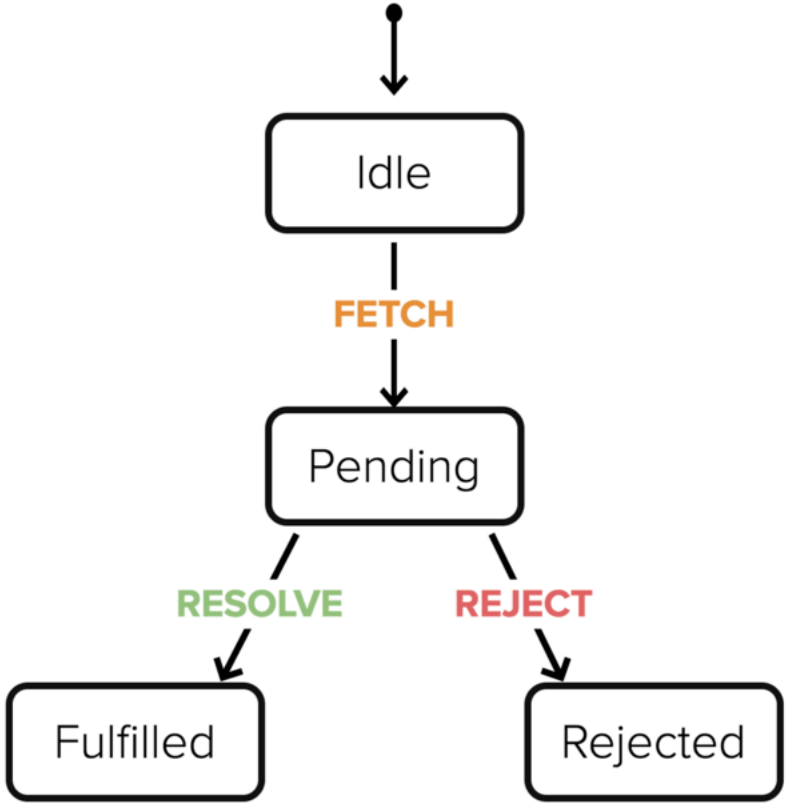
Finite state machines

have one **initial state**

a finite number of **states**

a finite number of **events**

a mapping of state **transitions** triggered by events



Finite state machines

have one **initial state**

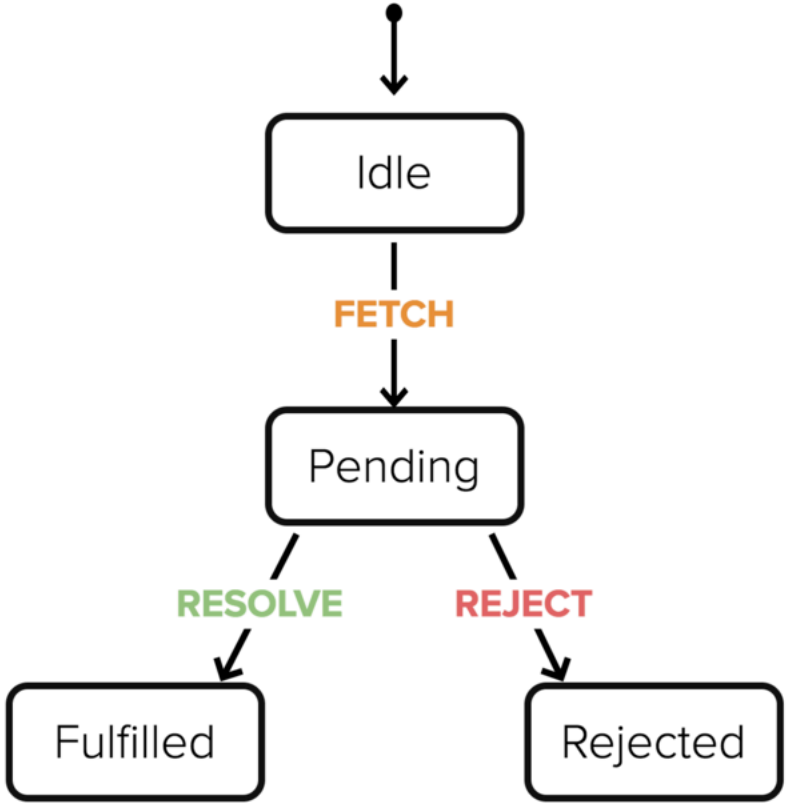
a finite number of **states**

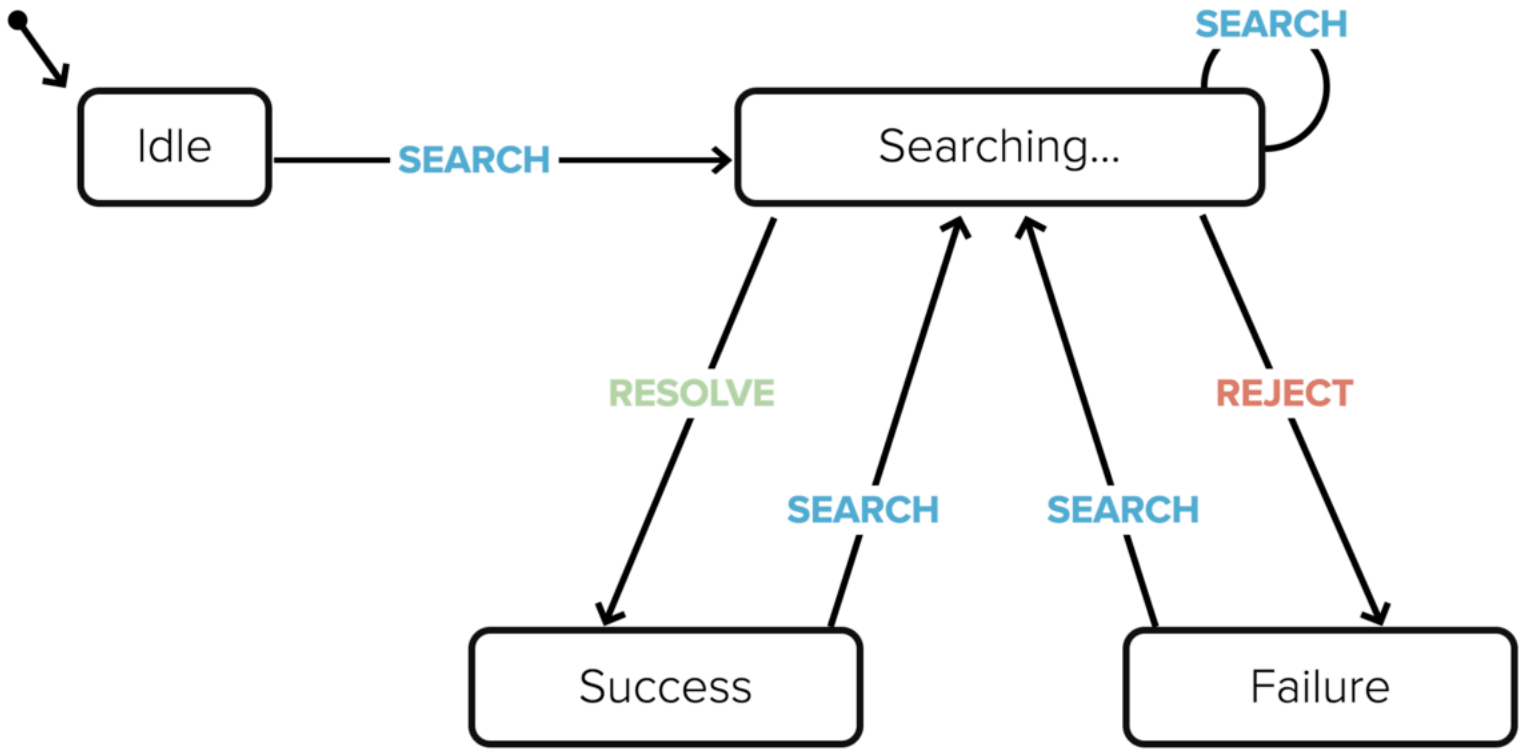
a finite number of **events**

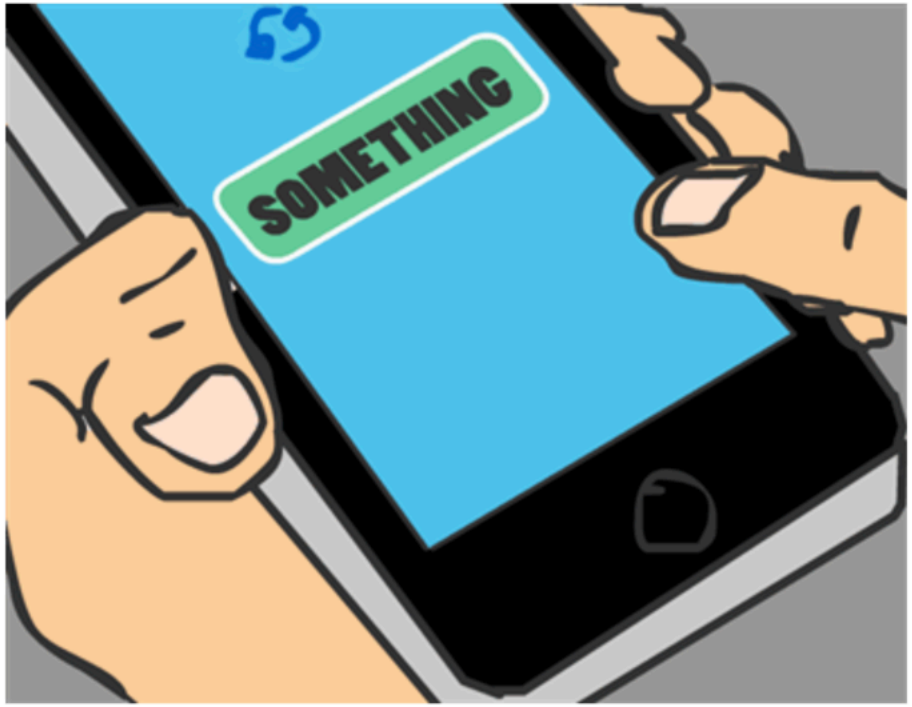
a mapping of state **transitions** triggered by events

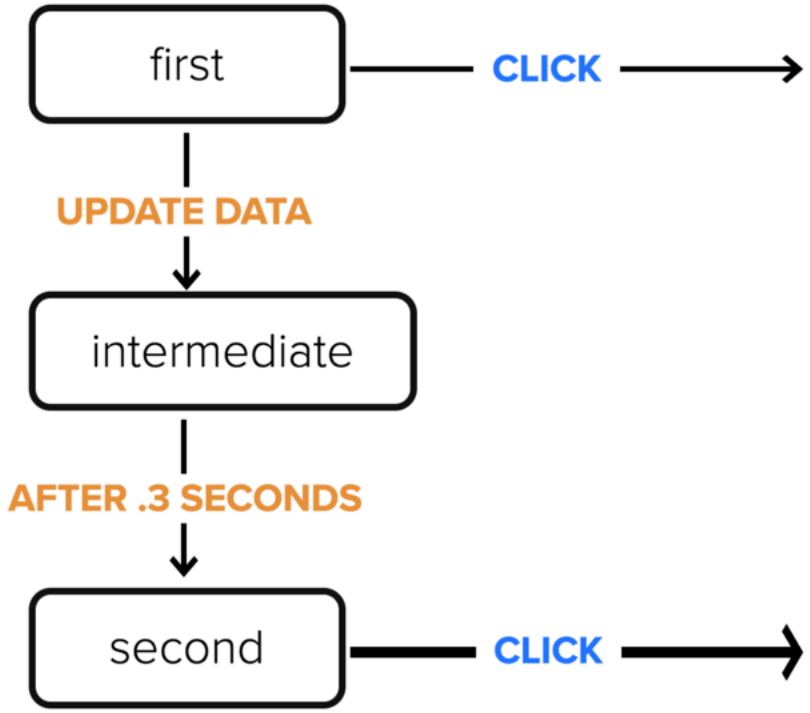
a finite number of **final states**

promisejs.org/implementing




SEARCH
Idle
SEARCH
Searching...
RESOLVE
REJECT
SEARCH
SEARCH
Success
Failure


SOMETHING


first
CLICK
UPDATE DATA
intermediate
AFTER .3 SECONDS
second
CLICK

```
const machine = {
  initial: 'idle',
  states: {
    idle: {
      on: { SEARCH: 'searching' }
    },
    searching: {
      on: {
        RESOLVE: 'success',
        REJECT: 'failure',
        SEARCH: 'searching'
      }
    },
    success: {
      on: { SEARCH: 'searching' }
    },
    failure: {
      on: { SEARCH: 'searching' }
    }
  }
};
```

Define **transitions** between **states** & **actions**

```
function transition(state, event) {
  return machine.states[state].on[event];
}
```

Transition function determines next state from state + event

```
const machine = {
  initial: 'idle',
  states: {
    idle: {
      on: { SEARCH: 'searching' }
    },
    searching: {
      on: {
        RESOLVE: 'success',
        REJECT: 'failure',
        SEARCH: 'searching'
      }
    },
    success: {
      on: { SEARCH: 'searching' }
    },
    failure: {
      on: { SEARCH: 'searching' }
    }
  }
};
```

Define **transitions** between **states** & **actions**

```
function transition(state, event) {
  return machine.states[state].on[event];
}
```

Transition function determines next state from state + event

```
const machine = {
  initial: 'idle',
  states: {
    idle: {
      on: { SEARCH: 'searching' }
    },
    searching: {
      on: {
        RESOLVE: 'success',
        REJECT: 'failure',
        SEARCH: 'searching'
      }
    },
    success: {
      on: { SEARCH: 'searching' }
    },
    failure: {
      on: { SEARCH: 'searching' }
    }
  }
};

function transition(state, event) {
  return machine.states[state].on[event];
}
```

Define **transitions** between **states** & **actions**

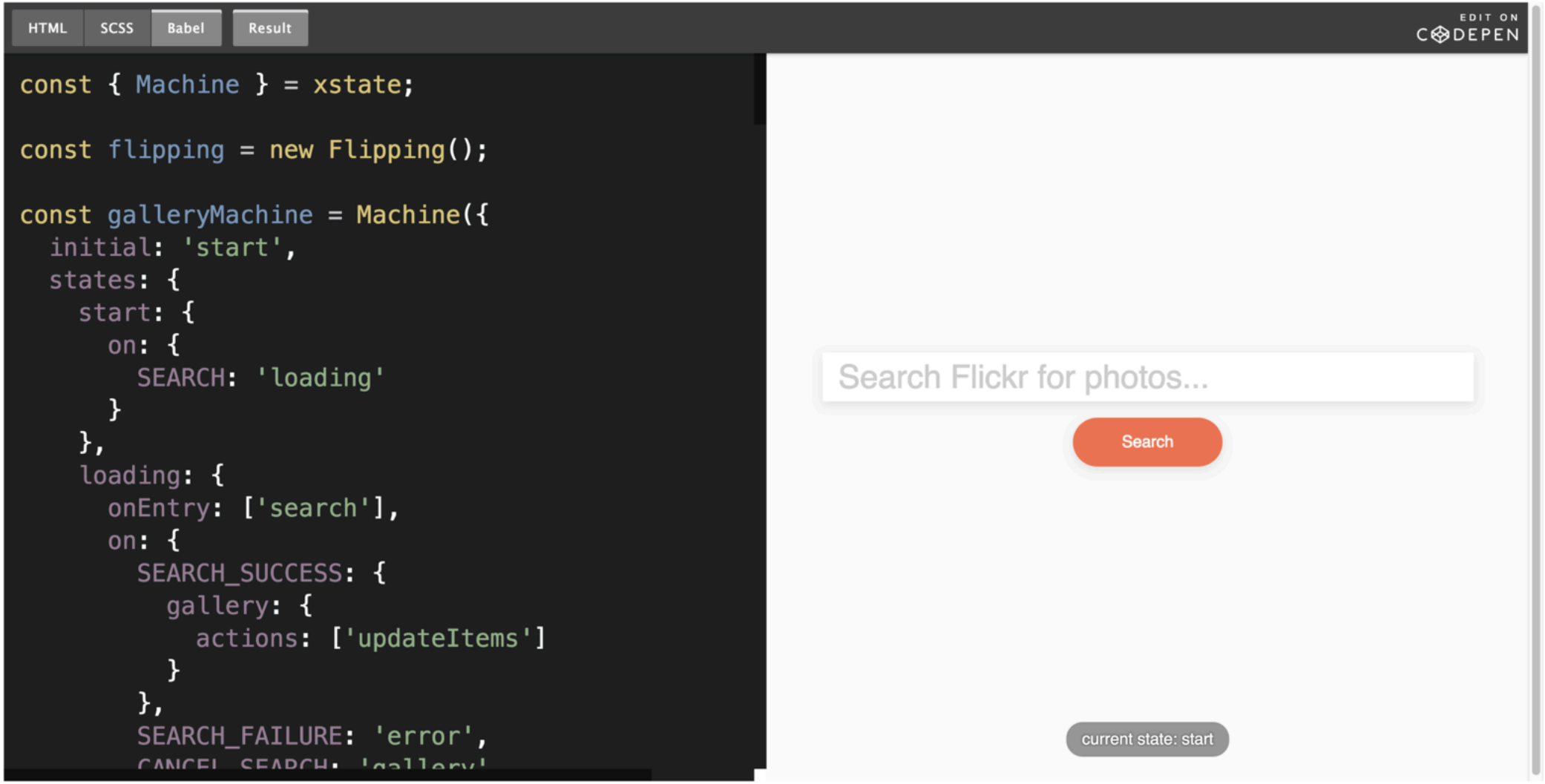
Transition function determines next state from state + event

```
const machine = {
  initial: 'idle',
  states: {
    idle: {
      on: { SEARCH: 'searching' }
    },
    searching: {
      on: {
        RESOLVE: 'success',
        REJECT: 'failure',
        SEARCH: 'searching'
      }
    },
    success: {
      on: { SEARCH: 'searching' }
    },
    failure: {
      on: { SEARCH: 'searching' }
    }
  }
};
```

Define **transitions** between **states** & **actions**

```
function transition(state, event) {
  return machine.states[state].on[event];
}
```

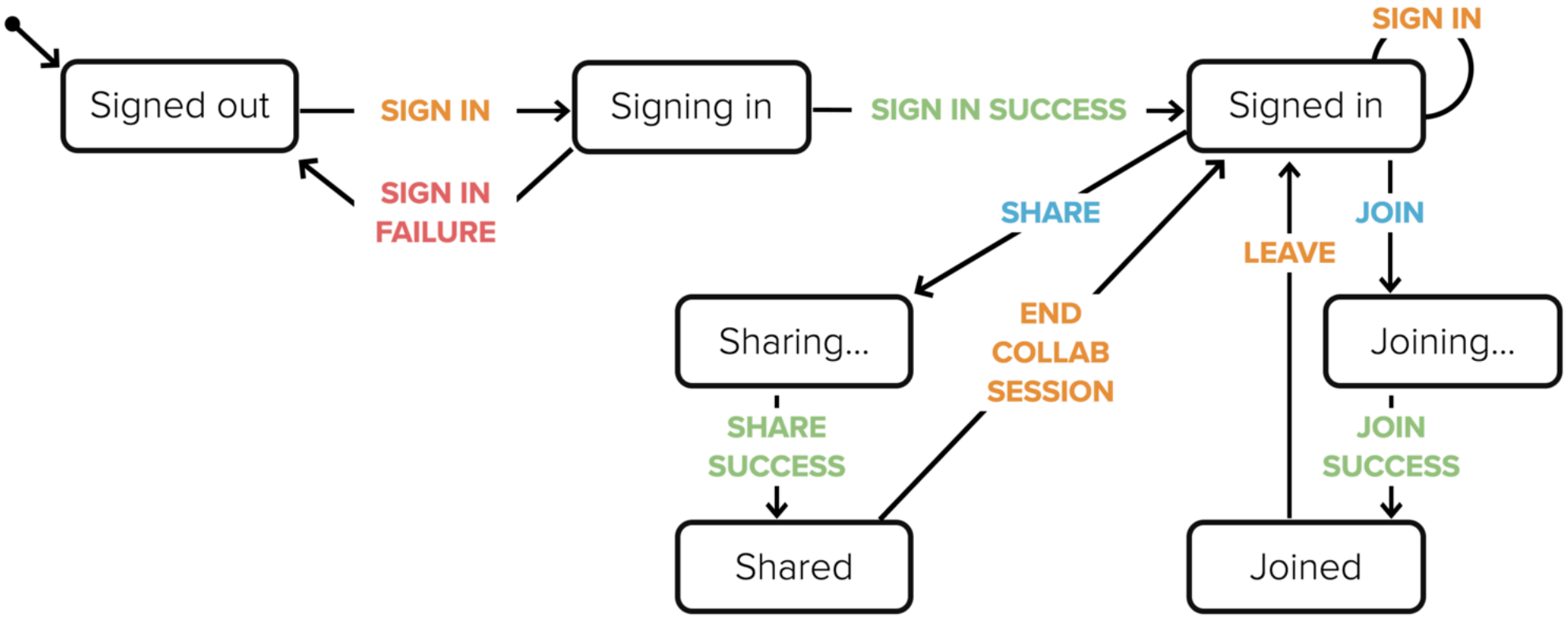
Transition function determines next state from state + event



Source: css-tricks.com/robust-react-user-interfaces-with-finite-state-machines

State machines in VS Live Share

http://aka.ms/vsls


Signed out
SIGN IN
Signing in
SIGN IN SUCCESS
Signed in
SIGN IN
SIGN IN FAILURE
SHARE
JOIN
LEAVE
Sharing...
END COLLAB SESSION
Joining...
SHARE SUCCESS
JOIN SUCCESS
Shared
Joined

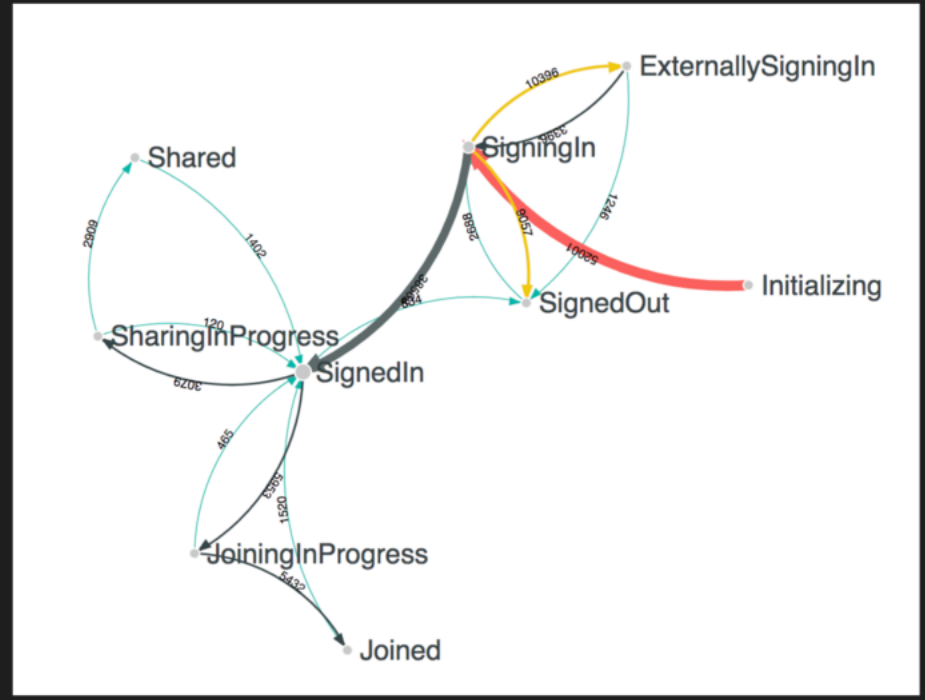
Using state machines for analytics

Using state machines for analytics

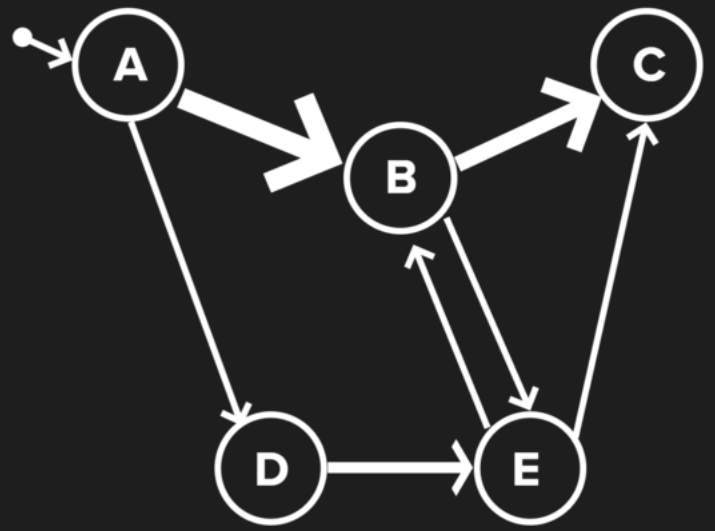
```
transition(currentState, event) {
  const nextState = // ...

  Telemetry.sendEvent(
    currentState,
    nextState,
    event
  );

  return nextState;
}
```

Using state machines for analytics

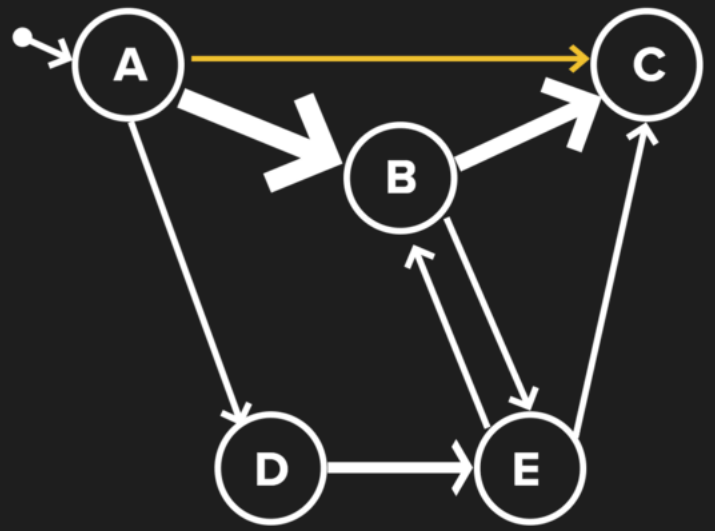
```
transition(currentState, event) {
  const nextState = // ...

  Telemetry.sendEvent(
    currentState,
    nextState,
    event
  );

  return nextState;
}
```



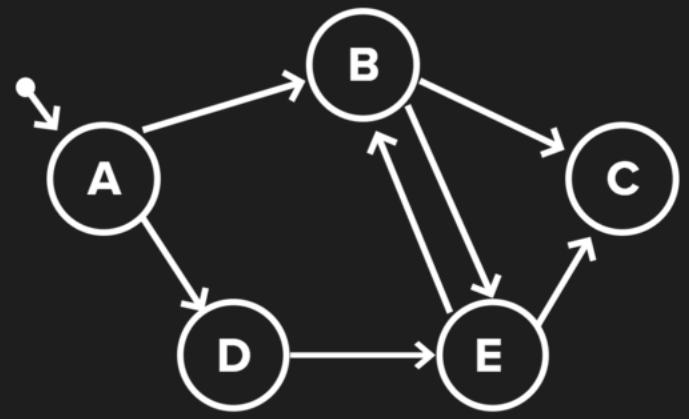
Using state machines for analytics



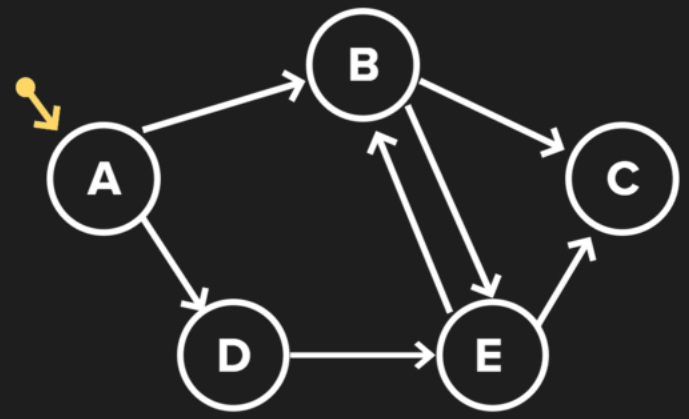
Using state machines for analytics



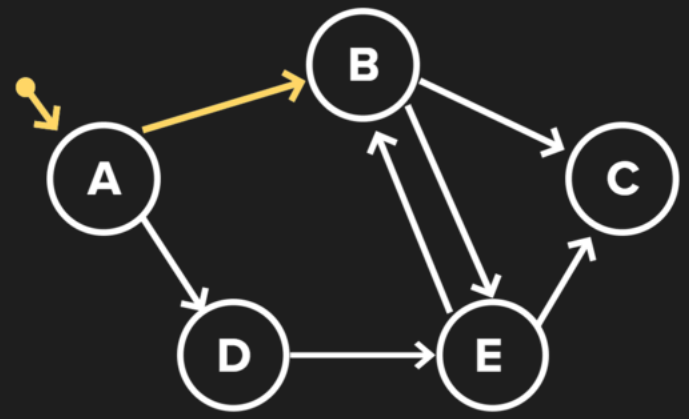
Using state machines for integration testing



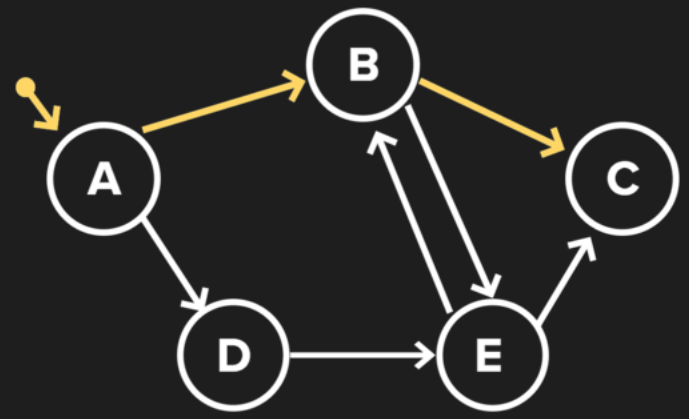
Using state machines for integration testing



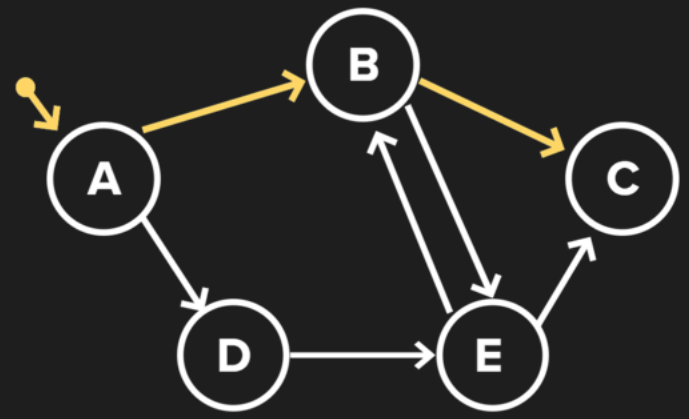
Using state machines for integration testing



Using state machines for integration testing



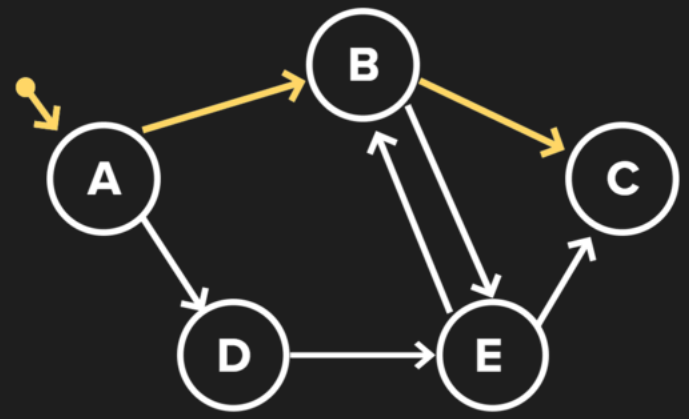
Using state machines for integration testing




```
A → B
A → B → C
A → D
A → D → E
```

Using state machines for integration testing

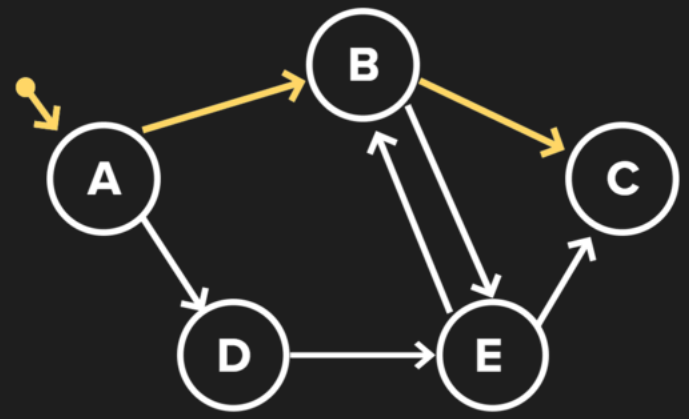
Shortest path algorithms (Dijkstra, Bellman-Ford, A* search, etc.)




```
A → B
A → B → C
A → D
A → D → E
```

Using state machines for integration testing

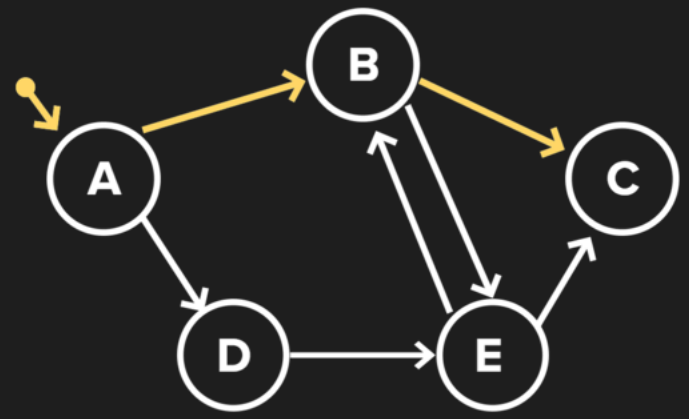
Shortest path algorithms (Dijkstra, Bellman-Ford, A* search, etc.)
Analytics provides weights




```
A → B
A → B → C
A → D
A → D → E
```

Using state machines for integration testing

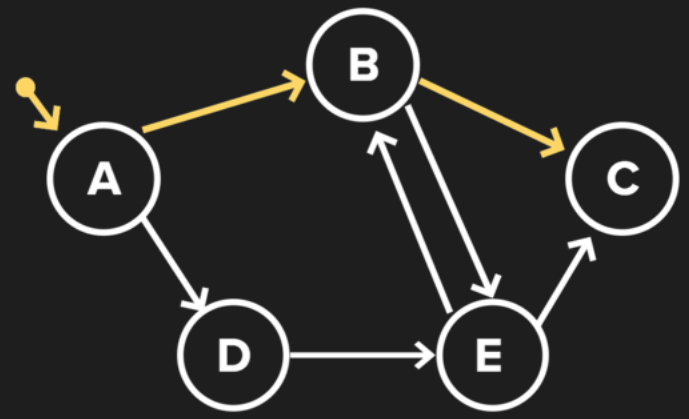
Shortest path algorithms (Dijkstra, Bellman-Ford, A* search, etc.)
Analytics provides weights
Represents all **happy paths**




```
A → B
A → B → C
A → D
A → D → E
```

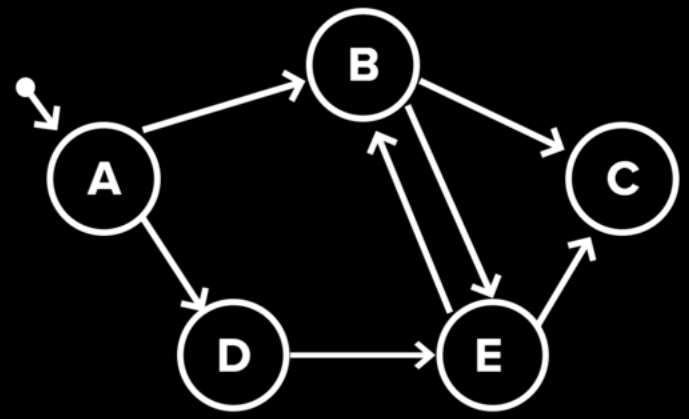
Using state machines for integration testing

Shortest path algorithms (Dijkstra, Bellman-Ford, A* search, etc.)
Analytics provides weights
Represents all **happy paths**
Can be **automatically generated**



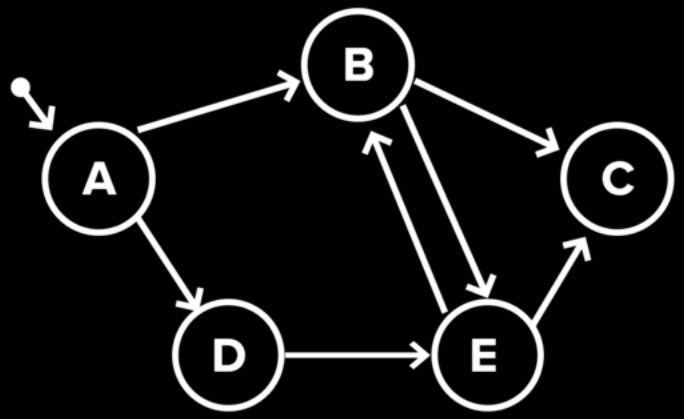

```
A → B
A → B → C
A → D
A → D → E
```

Using state machines for integration testing



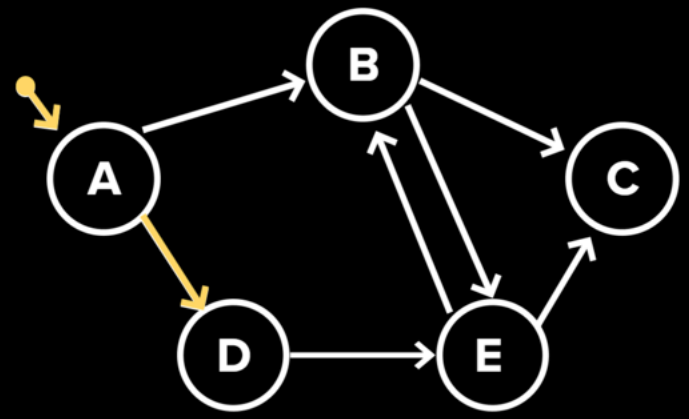
Using state machines for integration testing

Depth-first search (DFS) algorithm for finding all **simple paths**



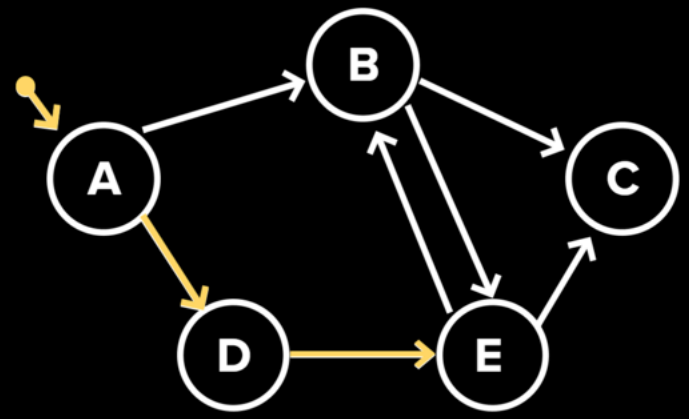
Using state machines for integration testing

Depth-first search (DFS) algorithm for finding all **simple paths**



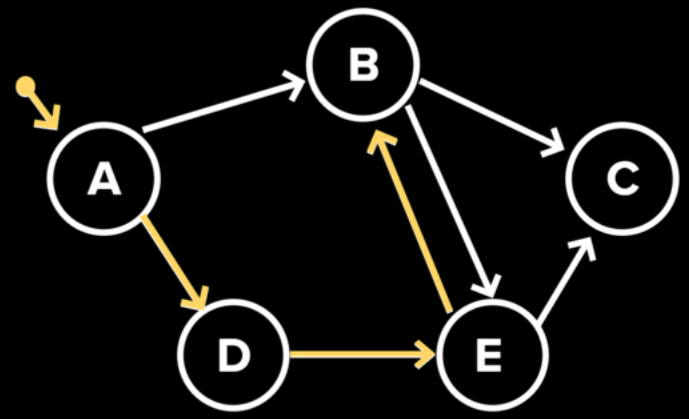
Using state machines for integration testing

Depth-first search (DFS) algorithm for finding all **simple paths**



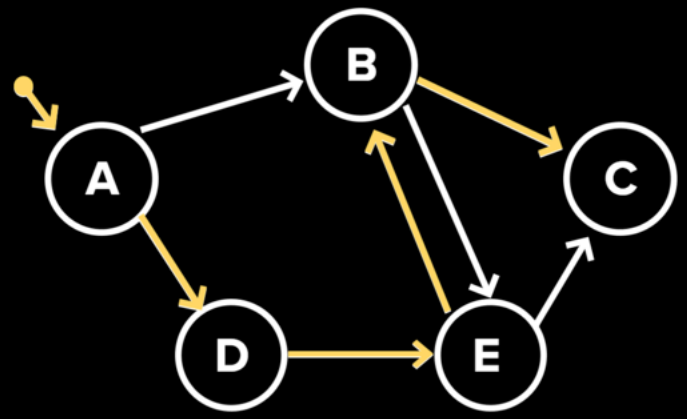
Using state machines for integration testing

Depth-first search (DFS) algorithm for finding all **simple paths**



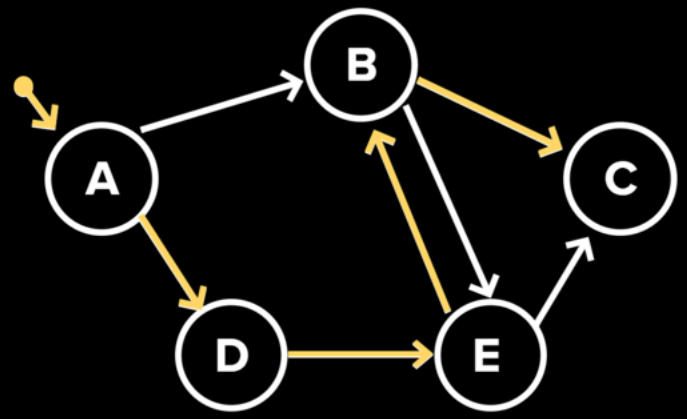
Using state machines for integration testing

Depth-first search (DFS) algorithm for finding all **simple paths**



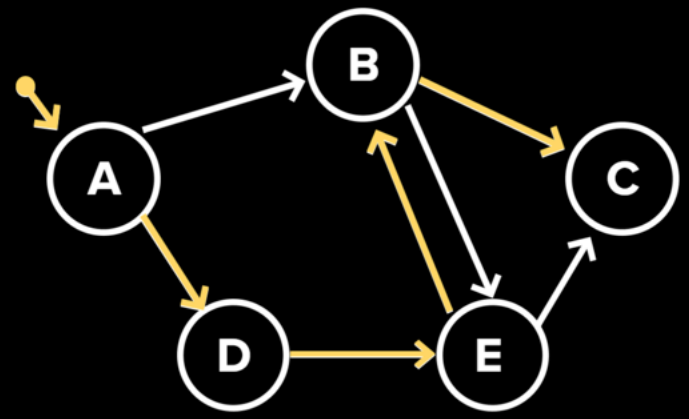
Using state machines for integration testing

Depth-first search (DFS) algorithm for finding all **simple paths**
Represents all possible **user flows**



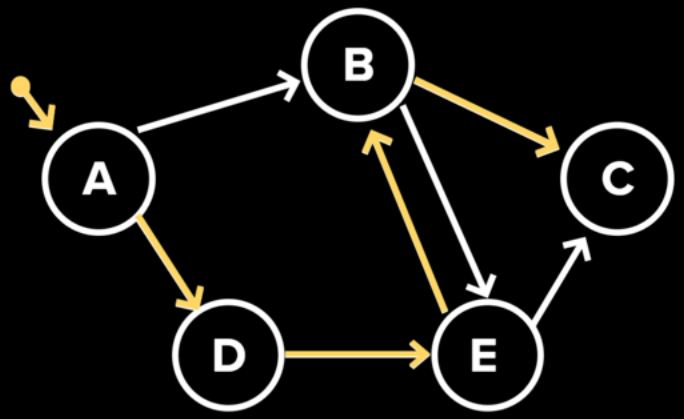
Using state machines for integration testing

Depth-first search (DFS) algorithm for finding all **simple paths**
Represents all possible **user flows**
Reveals all **edge cases**



Using state machines for integration testing

Depth-first search (DFS) algorithm for finding all **simple paths**
Represents all possible **user flows**
Reveals all **edge cases**
Can be **automatically generated** ⚠️




```
A → B
A → B → C
A → D
A → D → E
A → D → E → C
A → D → B → C
A → B → E → C
A → D → E → B → C
```

Using state machines for integration testing

Depth-first search (DFS) algorithm for finding all **simple paths**

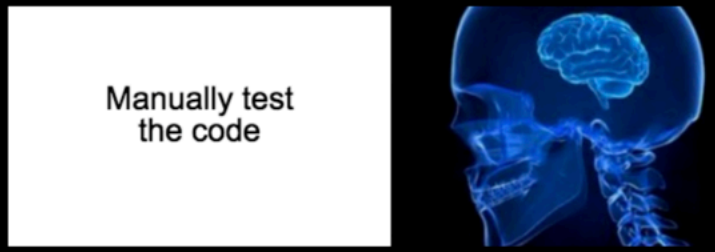
Represents all possible **user flows**

Reveals all **edge cases**

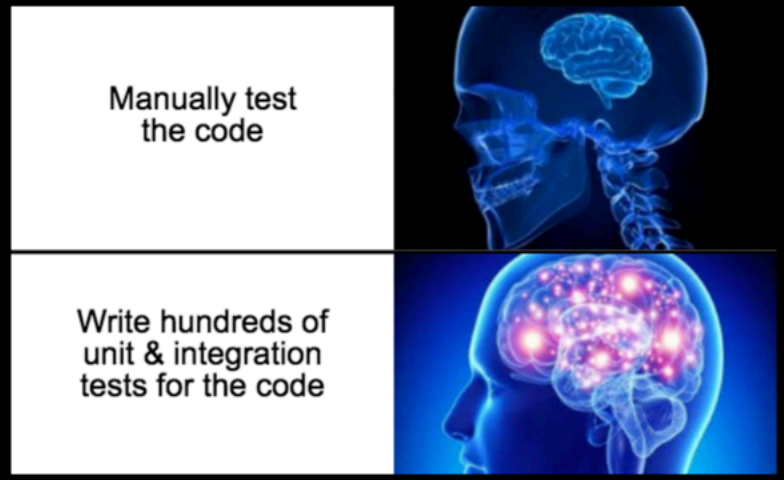
Can be **automatically generated** ⚠️

```
PASS src/App.test.js
  'question' state
    path 0
      ✓ is in 'question' (1ms)
  'thanks' state
    path 0
      ✓ is in 'question' and can send 'GOOD' (2ms)
      ✓ is in 'thanks'
    path 1
      ✓ is in 'question' and can send 'BAD' (3ms)
      ✓ is in 'form' and can send 'SUBMIT' (2ms)
      ✓ is in 'thanks'
  'closed' state
    path 0
      ✓ is in 'question' and can send 'GOOD' (2ms)
      ✓ is in 'thanks' and can send 'ESC'
      ✓ is in 'closed' (1ms)
    path 1
      ✓ is in 'question' and can send 'GOOD' (2ms)
      ✓ is in 'thanks' and can send 'CLOSE'
      ✓ is in 'closed' (1ms)
    path 2
      ✓ is in 'question' and can send 'BAD' (2ms)
      ✓ is in 'form' and can send 'ESC' (1ms)
      ✓ is in 'closed'
    path 3
      ✓ is in 'question' and can send 'BAD' (2ms)
      ✓ is in 'form' and can send 'SUBMIT' (1ms)
      ✓ is in 'thanks' and can send 'ESC'
      ✓ is in 'closed'
    path 4
      ✓ is in 'question' and can send 'BAD' (1ms)
      ✓ is in 'form' and can send 'SUBMIT' (1ms)
      ✓ is in 'thanks' and can send 'CLOSE' (1ms)
      ✓ is in 'closed'
    path 5
      ✓ is in 'question' and can send 'ESC'
      ✓ is in 'closed'
  'form' state
    path 0
      ✓ is in 'question' and can send 'BAD' (1ms)
      ✓ is in 'form'

Test Suites: 1 passed, 1 total
Tests:       27 passed, 27 total
Snapshots:   0 total
Time:        0.279s, estimated 1s
Ran all test suites related to changed files.
```

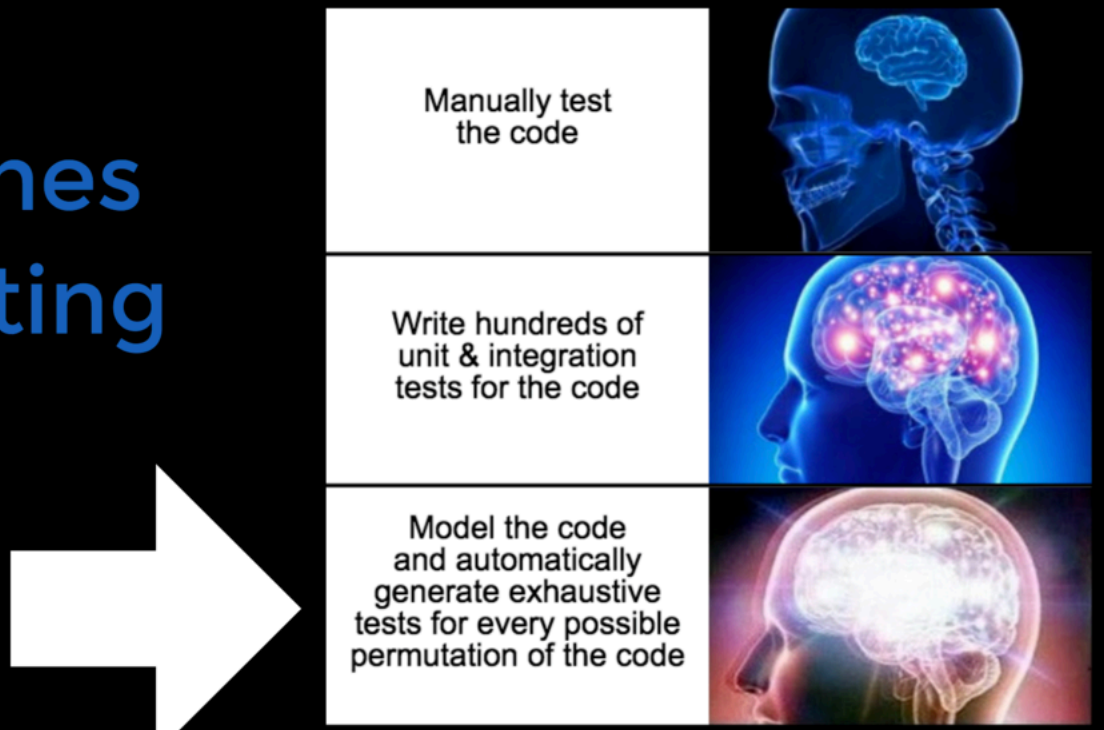
Using state machines for integration testing



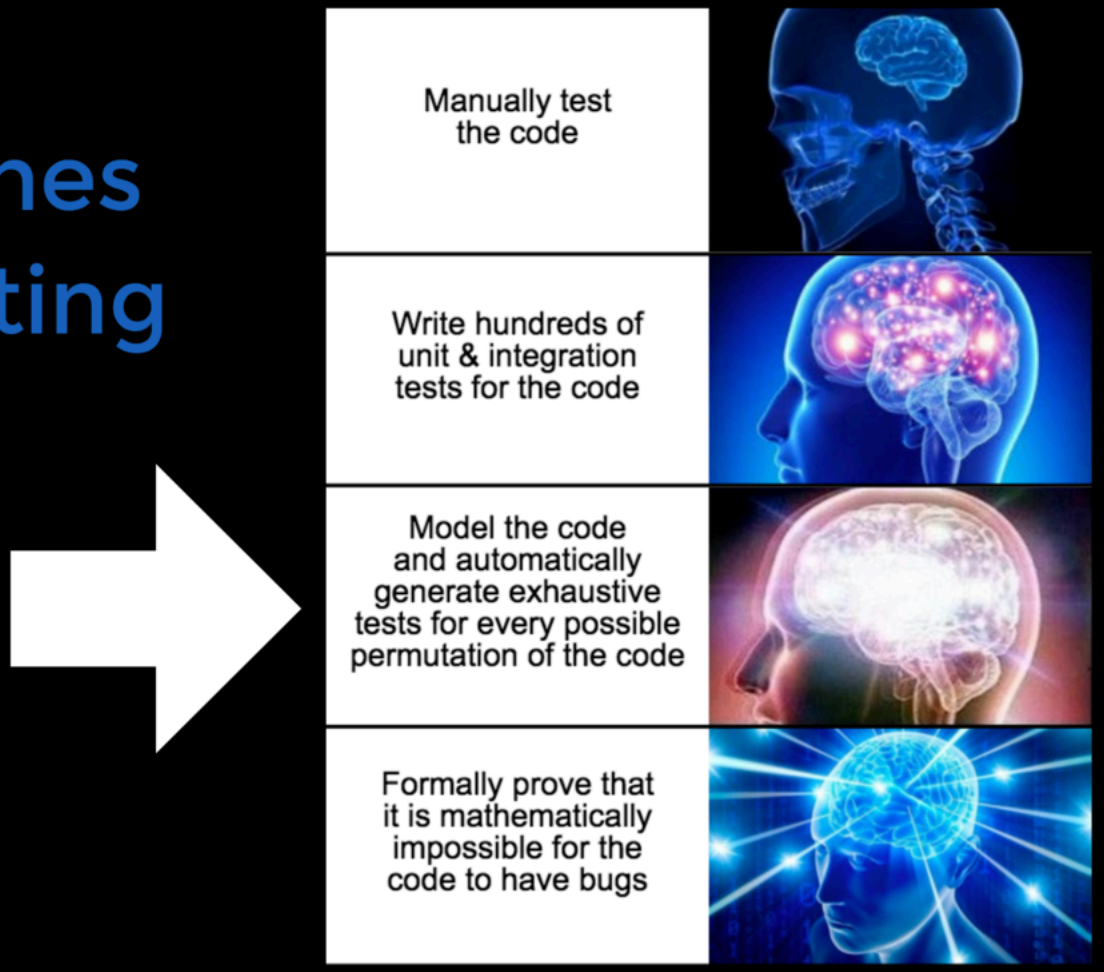
Using state machines for integration testing



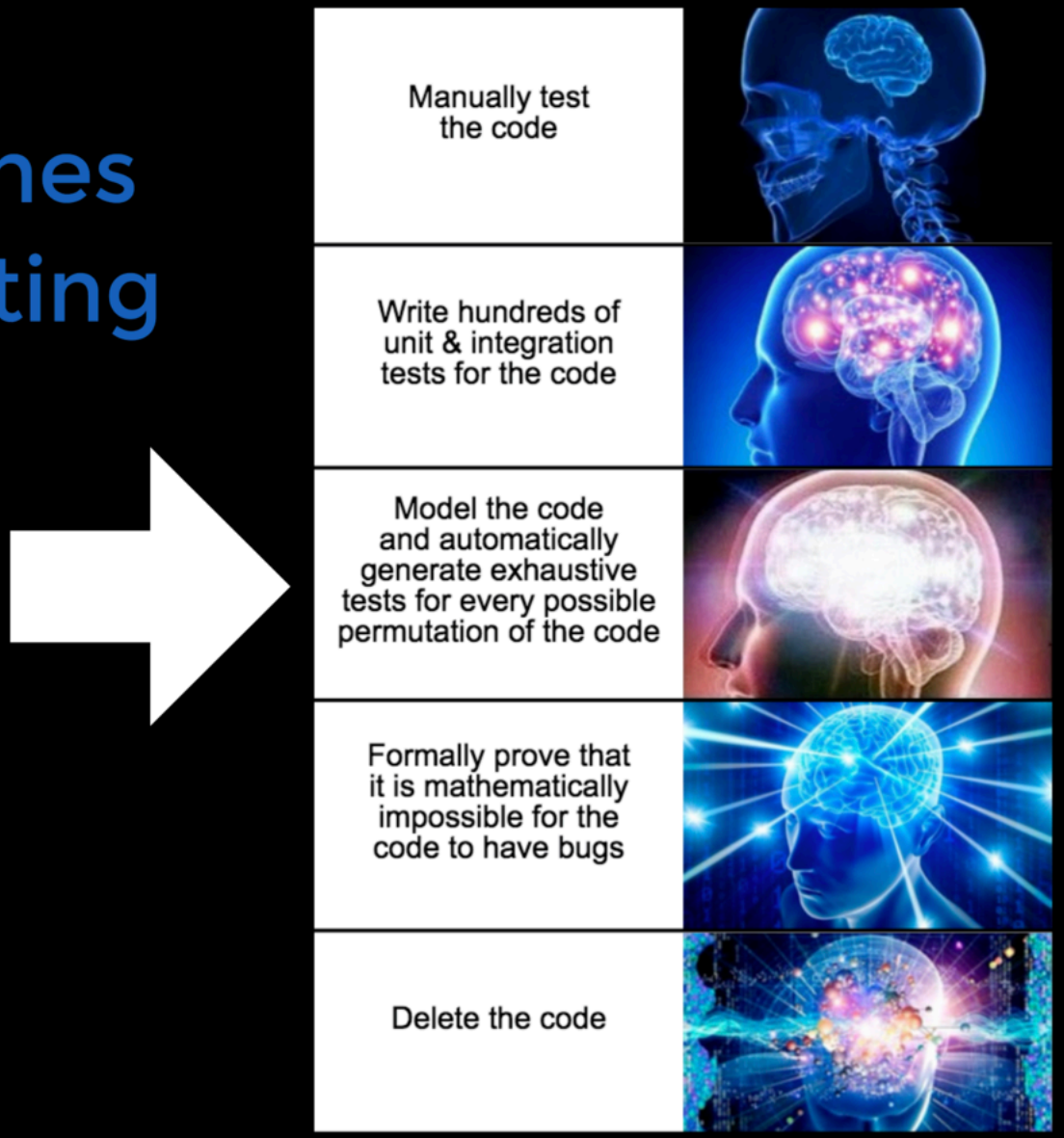
Using state machines for integration testing



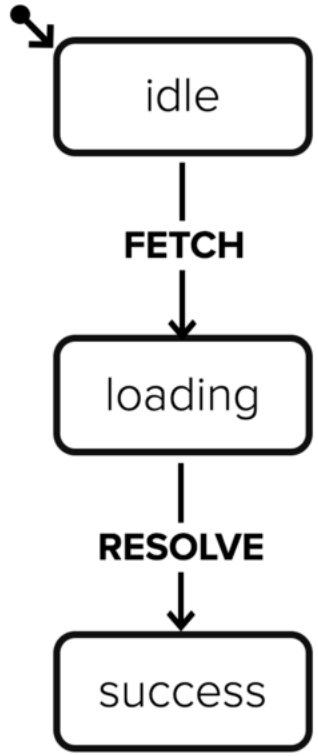
Using state machines for integration testing



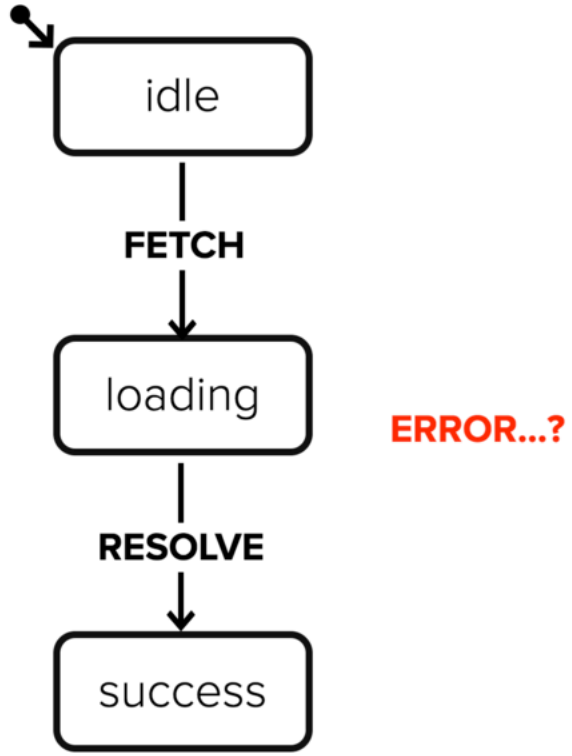
Using state machines for integration testing



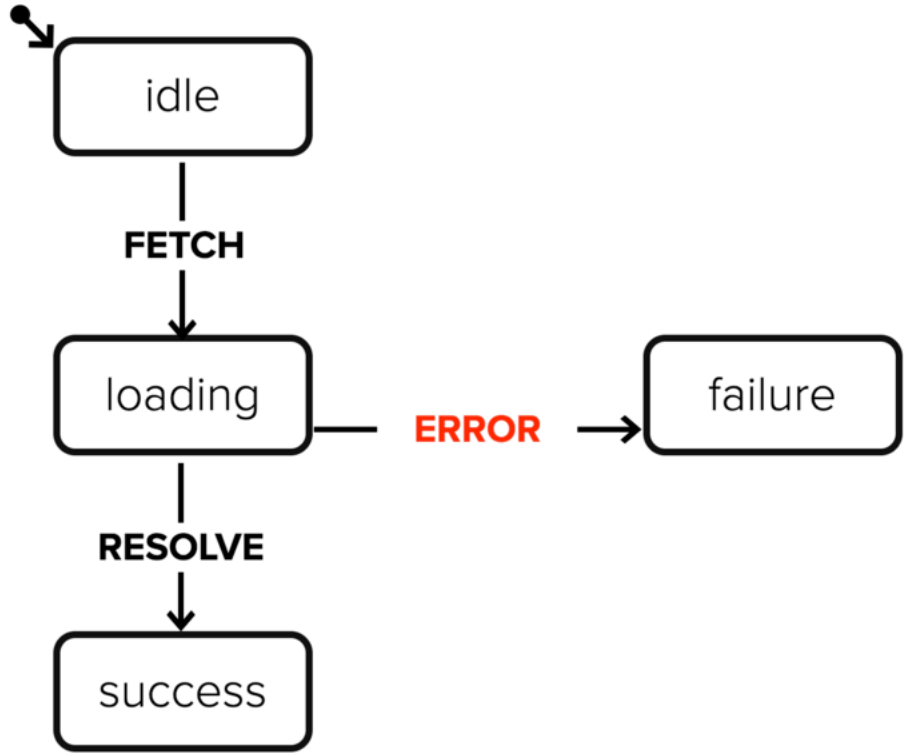
Software bugs are made visually clear



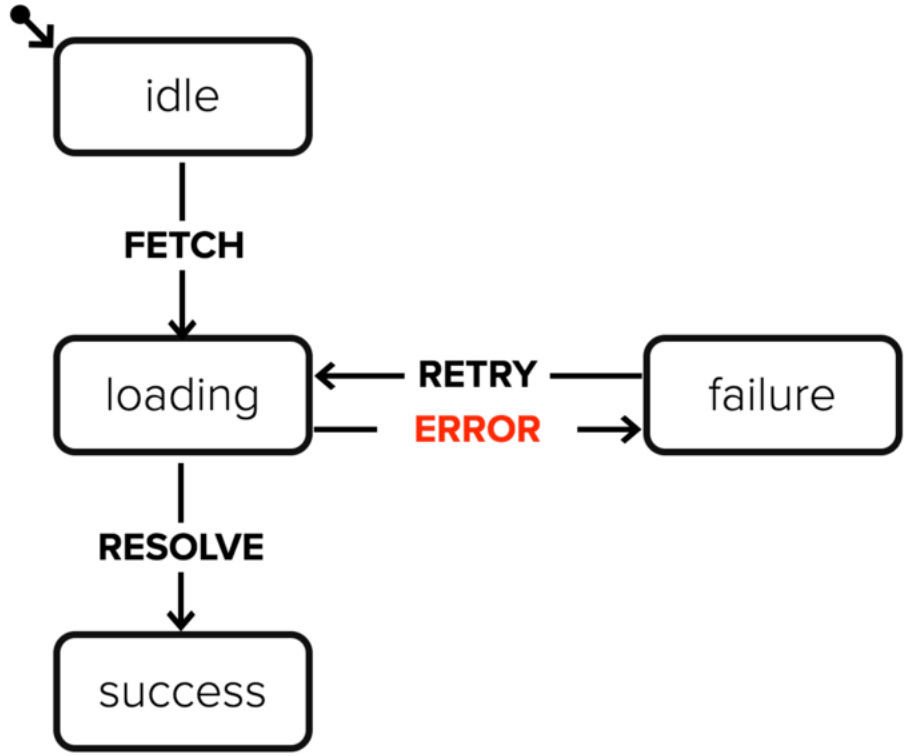
Software bugs are made visually clear



Software bugs are made visually clear



Software bugs are made visually clear



Does this scale?

Harel Statecharts
extended finite state machines

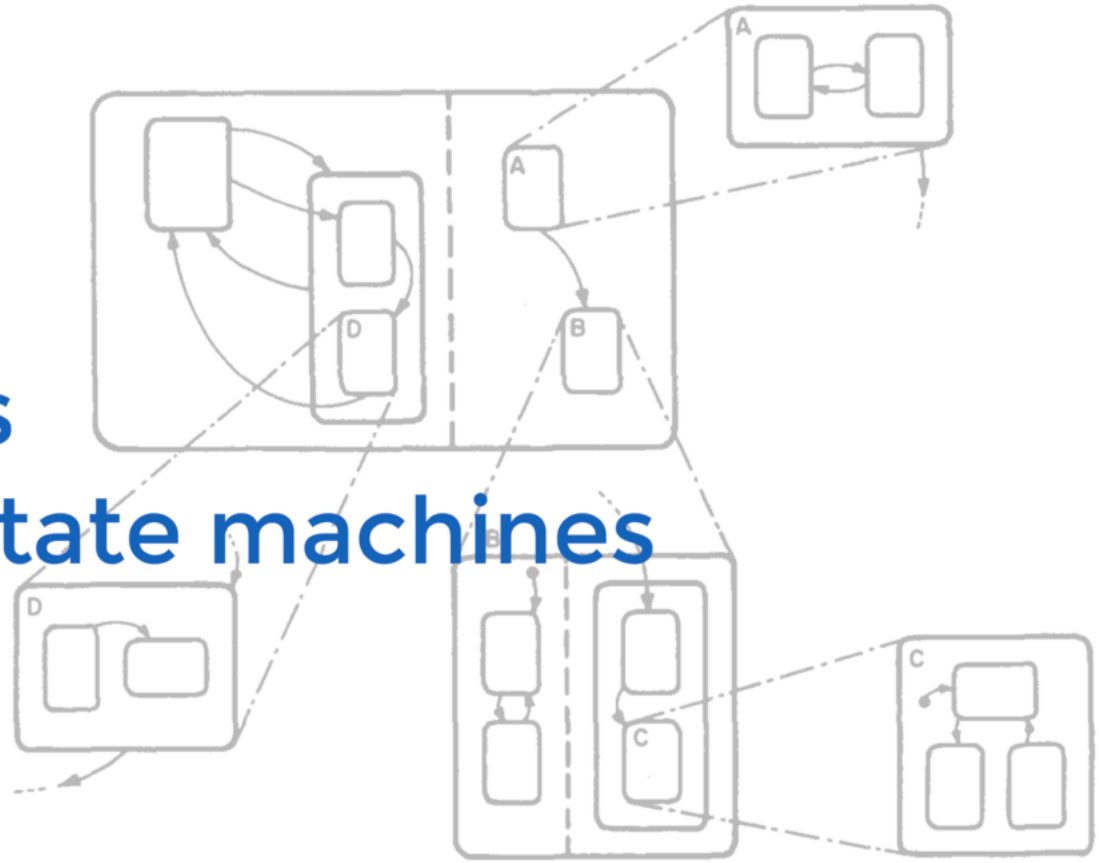
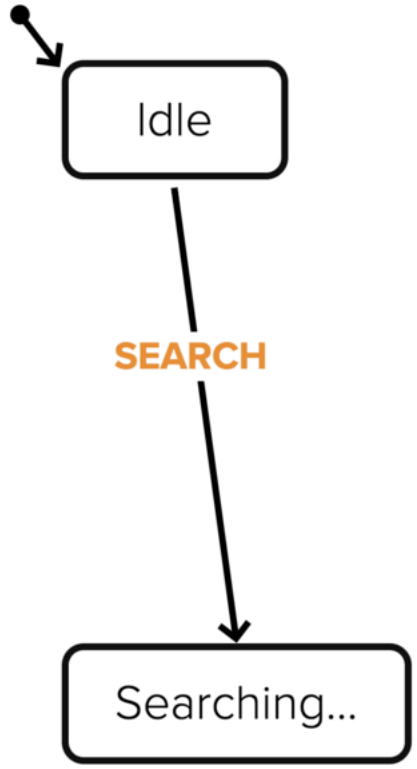


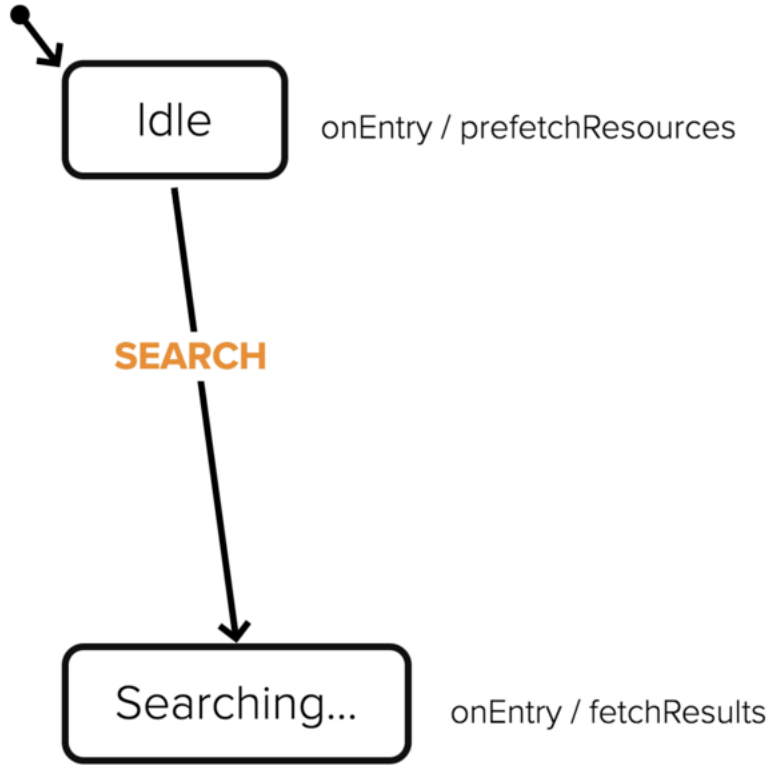
Fig. 36.

Statecharts



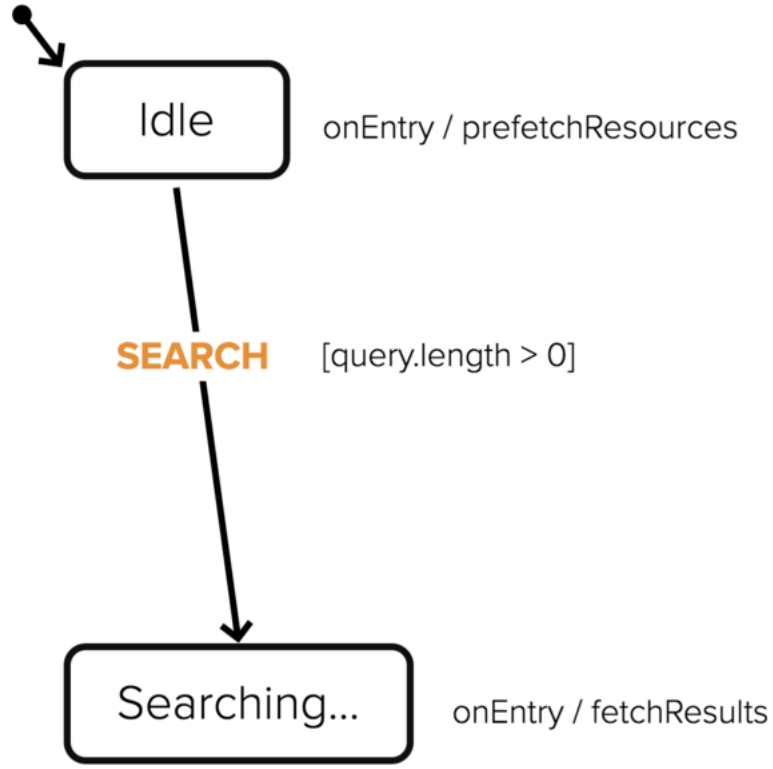
Statecharts

Actions - onEntry, onExit, transition



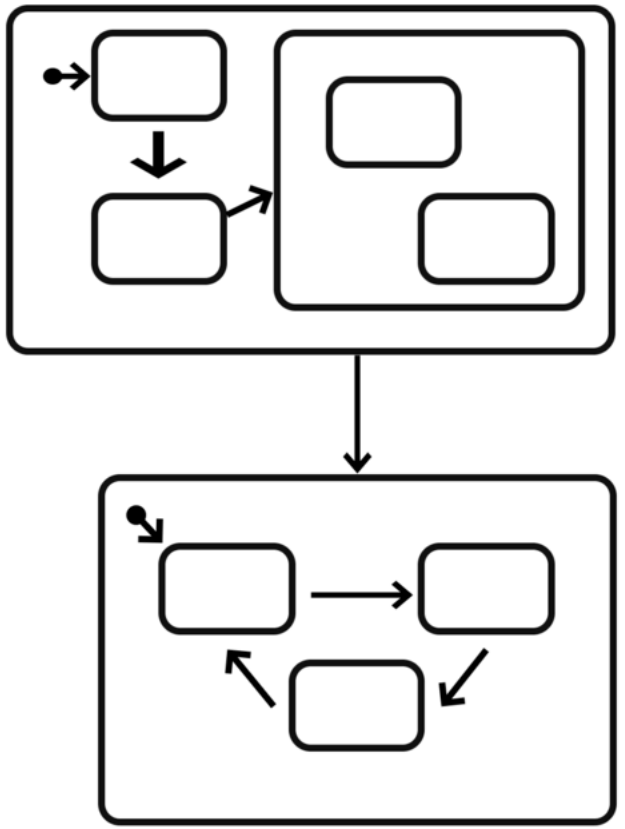
Statecharts

Actions - onEntry, onExit, transition
Guards - conditional transitions



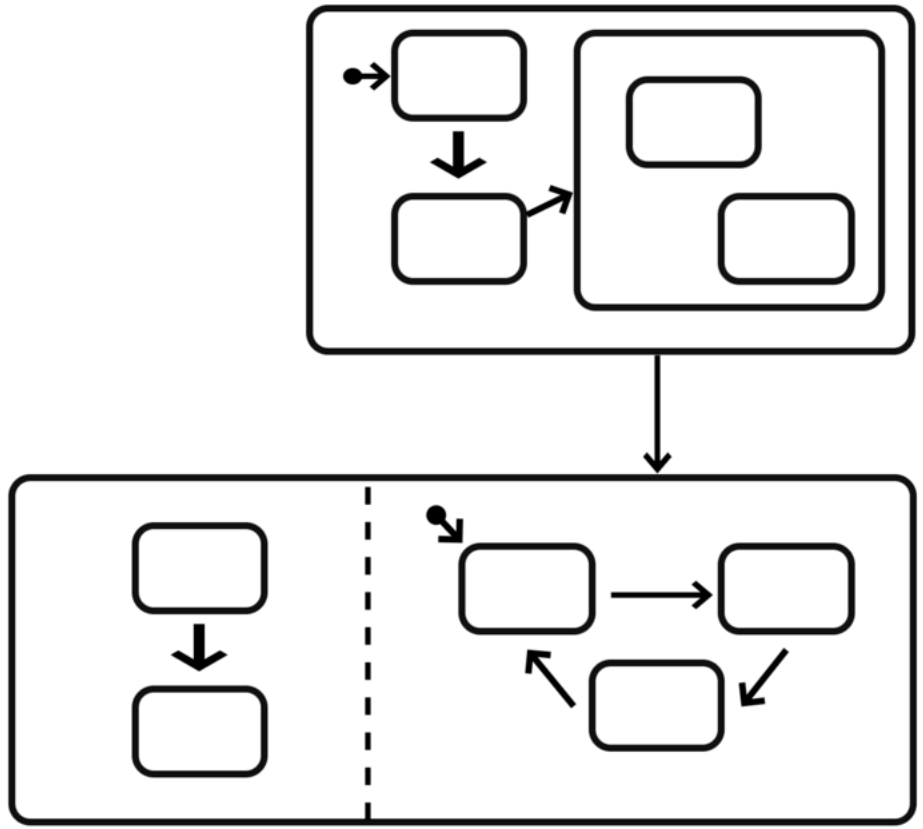
Statecharts

Actions - onEntry, onExit, transition
Guards - conditional transitions
Hierarchy - nested states



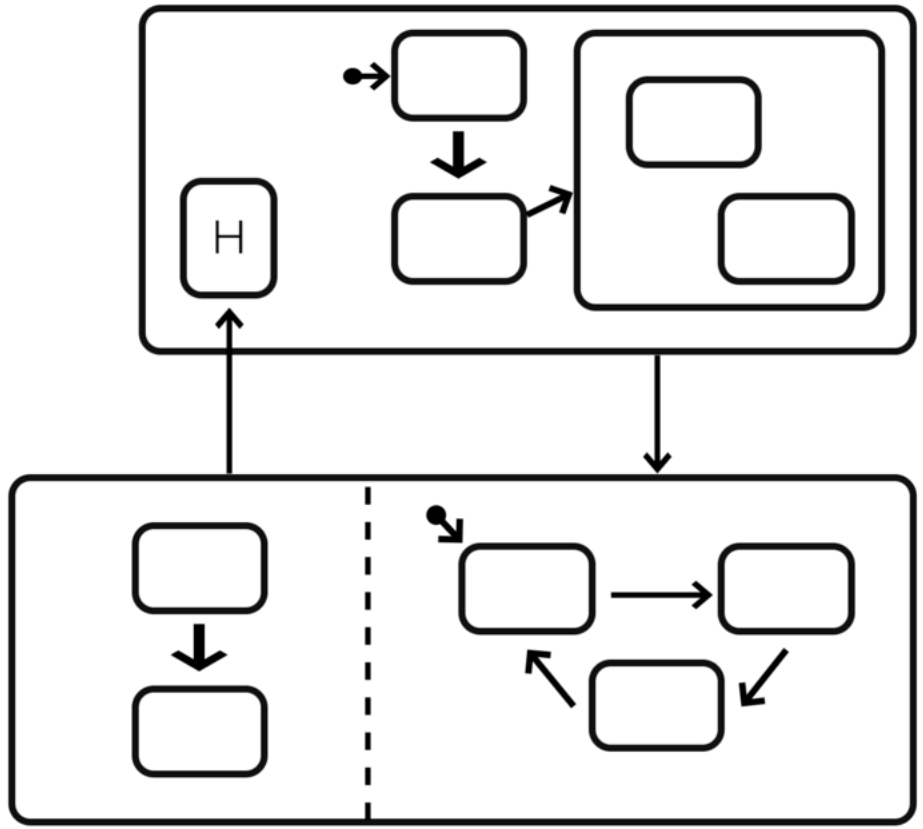
Statecharts

Actions - onEntry, onExit, transition
Guards - conditional transitions
Hierarchy - nested states
Orthogonality - parallel states



Statecharts

Actions - onEntry, onExit, transition
Guards - conditional transitions
Hierarchy - nested states
Orthogonality - parallel states
History - remembered states



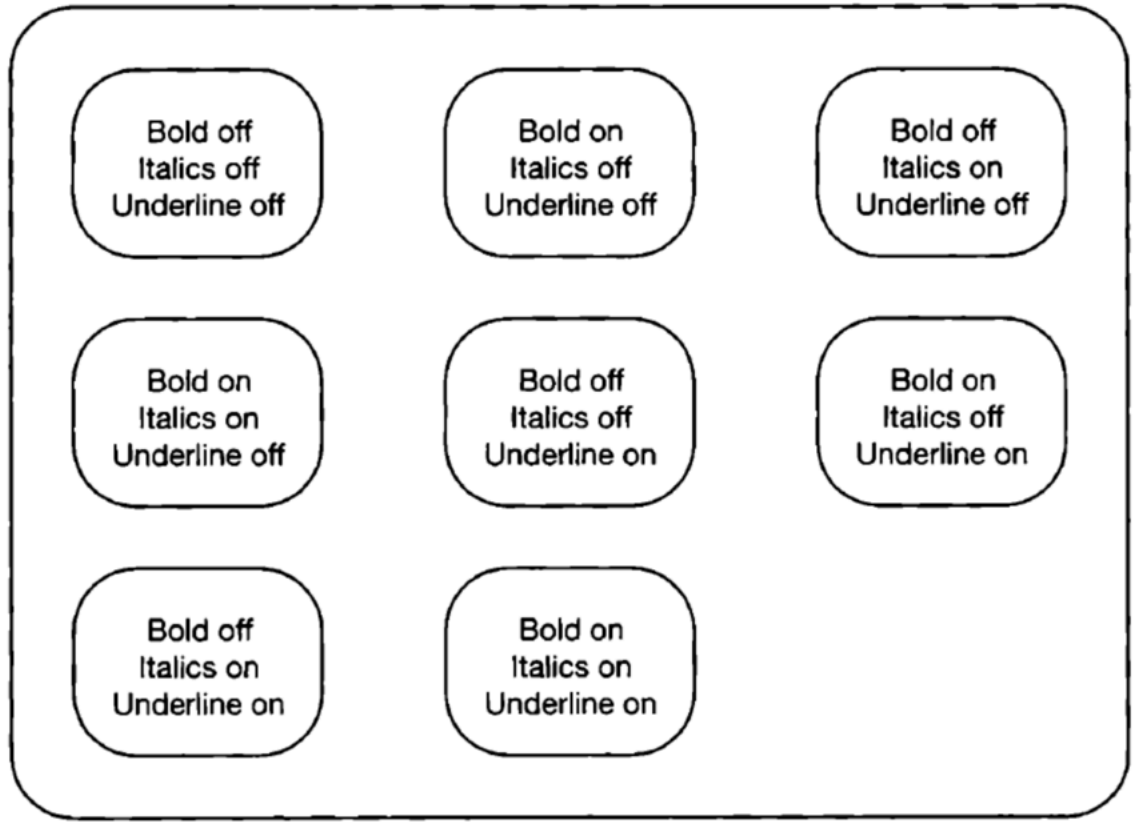
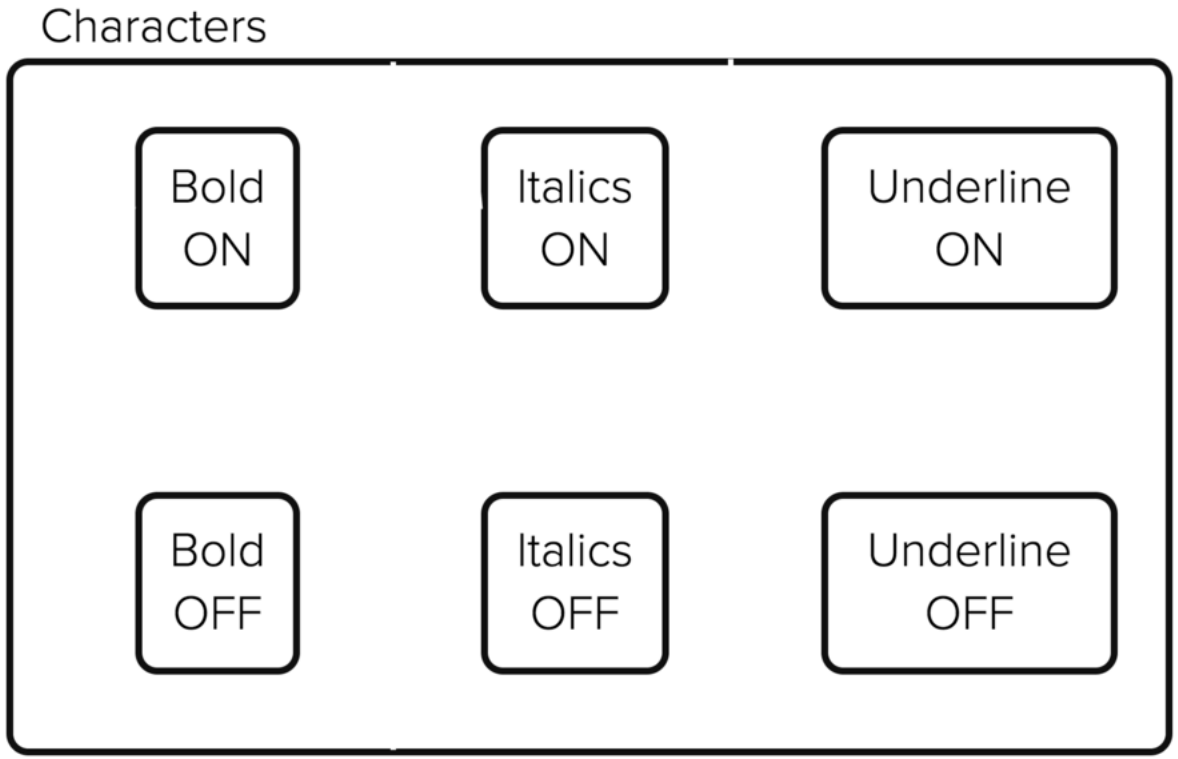
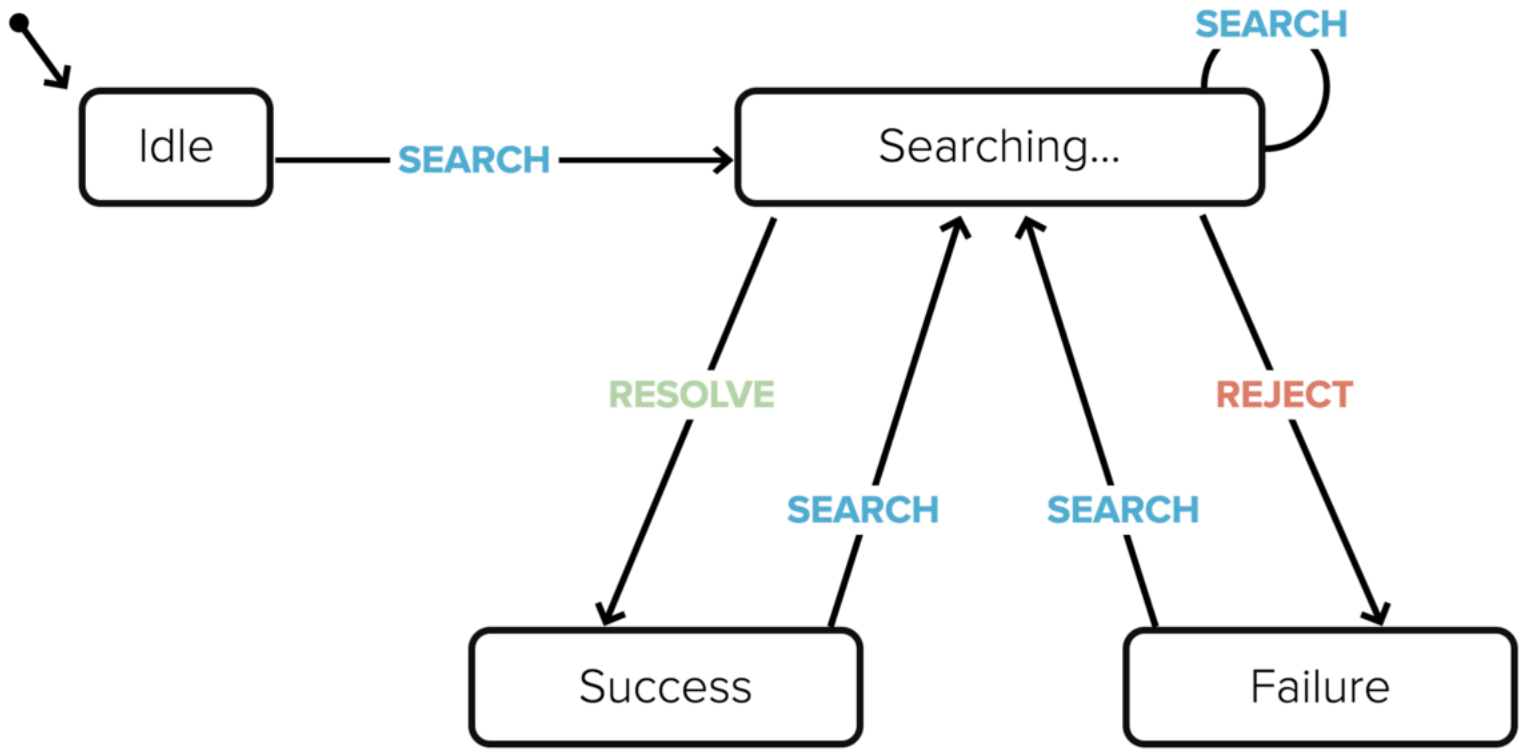
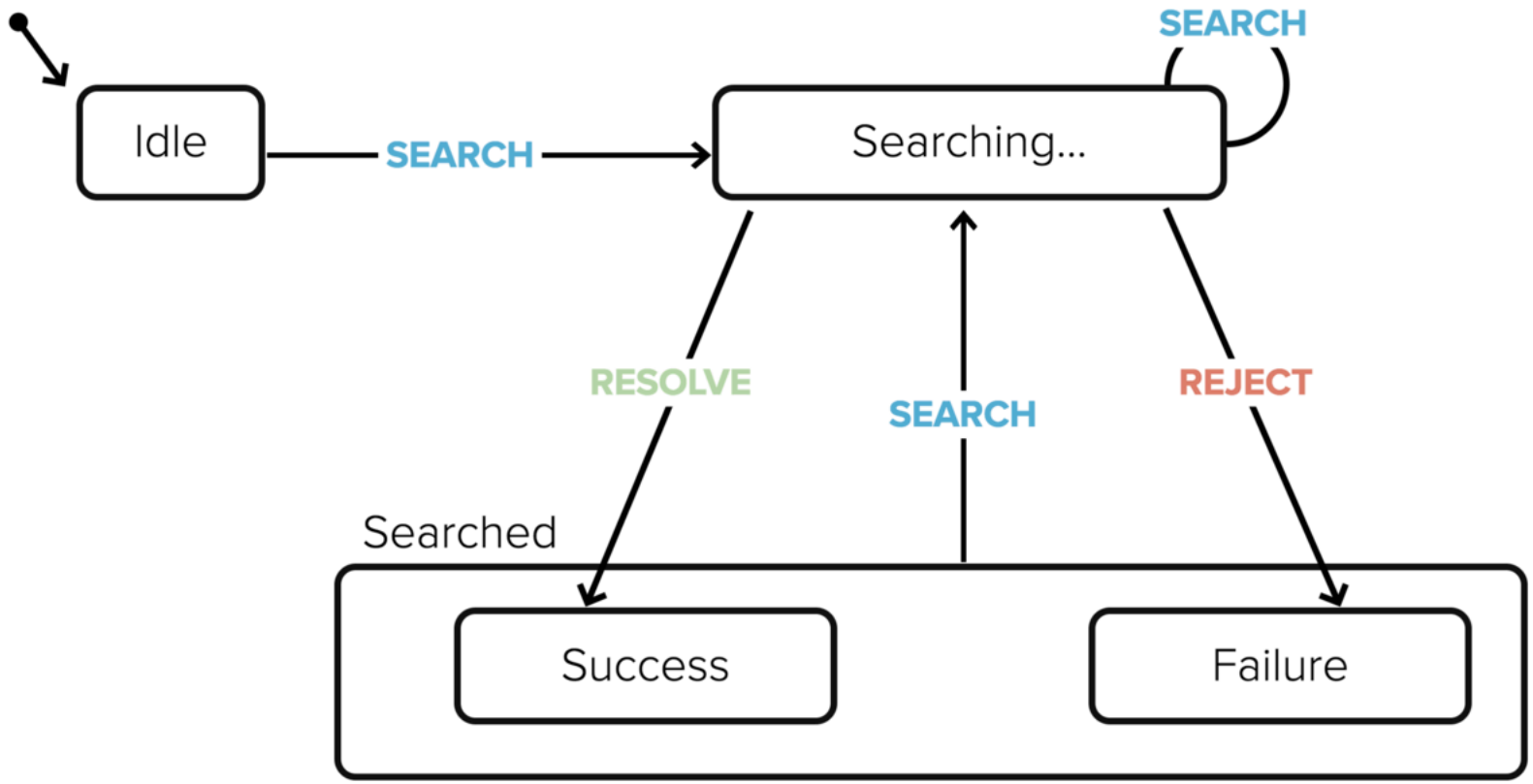

Bold off
Italics off
Underline off
Bold on
Italics off
Underline off
Bold off
Italics on
Underline off
Bold on
Italics on
Underline off
Bold off
Italics off
Underline on
Bold on
Italics off
Underline on
Bold off
Italics on
Underline on
Bold on
Italics on
Underline on

Figure 6.11


Characters
Bold
ON
Italics
ON
Underline
ON
Bold
OFF
Italics
OFF
Underline
OFF


SEARCH
Idle
SEARCH
Searching...
RESOLVE
REJECT
SEARCH
SEARCH
Success
Failure


SEARCH
Idle
SEARCH
Searching...
RESOLVE
SEARCH
REJECT
Searched
Success
Failure

Statecharts with xstate

```
npm install xstate --save
```

COPY

```
const lightMachine = Machine({
  initial: 'green',
  states: {
    green: {
      on: {
        TIMER: 'yellow'
      }
    },
    yellow: {
      on: {
        TIMER: 'red'
      }
    },
    red: {
      on: {
        TIMER: 'green'
      }
    }
  }
});

const nextState = lightMachine
  .transition('green', 'TIMER');

// State {
//   value: 'yellow'
// }
```

Statecharts with xstate

```
npm install xstate --save
```

Actions - onEntry, onExit, transition

COPY

```
const lightMachine = Machine({
  initial: 'green',
  states: {
    green: {
      on: {
        TIMER: 'yellow'
      }
    },
    yellow: {
      onEntry: ['activateYellow']
      on: {
        TIMER: 'red'
      }
    },
    red: {
      onExit: ['stopCountdown']
      on: {
        TIMER: 'green'
      }
    }
  }
});
```

Statecharts with xstate

`npm install xstate --save`

Actions - onEntry, onExit, transition
Guards - conditional transitions

```
const lightMachine = Machine({
  initial: 'green',
  states: {
    green: {
      on: {
        TIMER: {
          yellow: {
            cond: (xs, event) =>
              event.elapsed > 10000
          }
        }
      }
    },
    yellow: {
      on: {
        TIMER: 'red'
      }
    },
    red: {
      on: {
        TIMER: 'green'
      }
    }
  }
});
```

COPY

Statecharts with xstate

```
npm install xstate --save
```

Actions - onEntry, onExit, transition
Guards - conditional transitions
Hierarchy - nested states

COPY

```
const lightMachine = Machine({
  initial: 'green',
  states: {
    green: {
      on: {
        TIMER: 'yellow'
      }
    },
    yellow: {
      on: {
        TIMER: 'red'
      }
    },
    red: {
      initial: 'walk',
      states: {
        walk: {
          { on: { PED_COUNTDOWN: 'wait' } }
        },
        wait: {
          { on: { PED_COUNTDOWN_END: 'stop' } }
        },
        stop: {}
      }
    }
  }
});
```

Statecharts with xstate

`npm install xstate --save`

Actions - onEntry, onExit, transition
Guards - conditional transitions
Hierarchy - nested states
Orthogonality - parallel states

COPY

```
const lightsMachine = Machine({
  parallel: true,
  states: {
    northSouthLight: {
      initial: 'green',
      states: {
        // ...
      }
    },
    eastWestLight: {
      initial: 'red',
      states: {
        // ...
      }
    }
  }
});
```

Statecharts with xstate

```
npm install xstate --save
```

Actions - onEntry, onExit, transition
Guards - conditional transitions
Hierarchy - nested states
Orthogonality - parallel states
History - remembered states

COPY

```
const payMachine = Machine({
  initial: 'method',
  states: {
    method: {
      initial: 'card',
      states: {
        card: {
          on: { SELECT_CASH: 'cash' }
        },
        cash: {
          on: { SELECT_CARD: 'card' }
        }
      },
      on: {
        NEXT: 'review'
      }
    },
    review: {
      on: {
        PREV: 'method.$history'
      }
    }
  }
});
```

```
class App extends Component {
  static machine = Machine({ /* ... */ });

  state = {
    appState: machine.initialState
  };

  actions = { /* ... */ };

  send(event) {
    const nextState = machine.transition(
      this.state.appState,
      event.type,
      this.state
    );
    const { actions } = nextState;

    this.setState(
      { appState: nextState },
      () => {
        const nextExtState = actions
          .reduce((extState, action) => {
            const command = this.actions[action];
            // Execute the command
            return command(extState, eventType) || extState;
          }, this.state);

        this.setState(nextExtState);
      }
    );
  }
```

```
  render() {
    return (
      <form onSubmit={() => this.send('SUBMIT')}>
        <input
          type="text"
          onChange={e => this.send({
            type: 'CHANGE',
            value: e.target.value
          })}
        />

        <button>Submit</button>
      </form>
    );
  }
```

```rxjs
import { scan } from 'rxjs/operators';

const machine = new Machine({/* ... */});
const event$ = // ...

const state$ = event$.pipe(
  scan(machine.transition),
  tap(({ actions }) => {
    // execute actions
  });
```

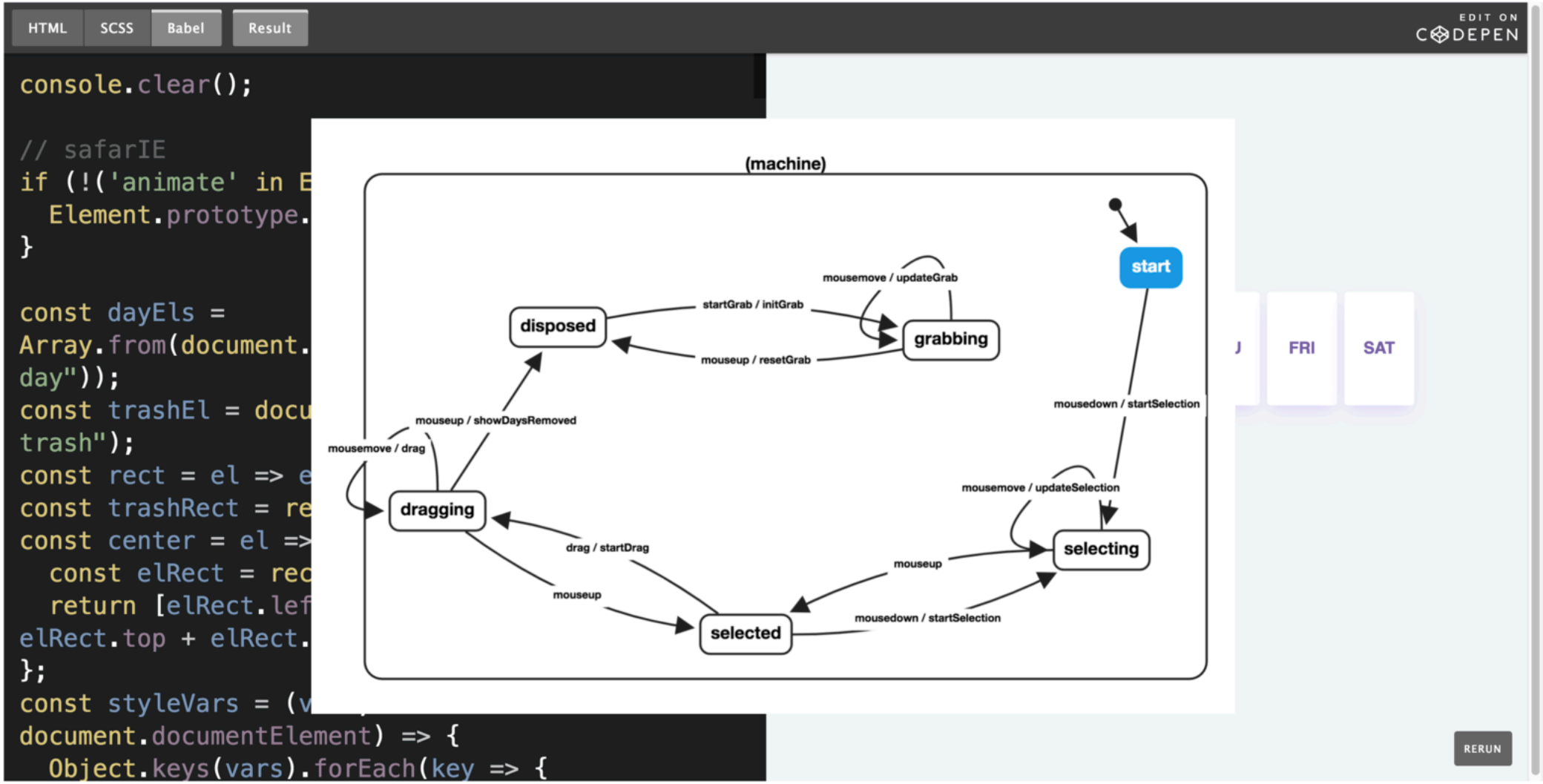
```redux
const machine = new Machine({/* ... */});

export const reducer = machine.transition;

// ...

class App extends Component() {
  // ...
  componentDidUpdate() {
    const { state } = this.props;
    const { actions } = state;

    const nextState = actions.reduce(action
      // execute the action commands
    }, this.state);

    // local state, or send it to Redux
    this.setState(nextState);
  }
  // ...
}
```

```
import { scan } from 'rxjs/operators';

const machine = new Machine({/* ... */});
const event$ = // ...

const state$ = event$.pipe(
  scan(machine.transition),
  tap(({ actions }) => {
    // execute actions
  });
```

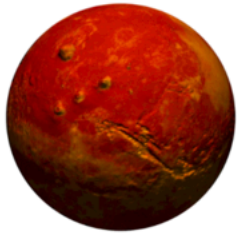
```
const machine = new Machine({/* ... */});

export const reducer = machine.transition;

// ...

class App extends Component() {
  // ...
  componentDidUpdate() {
    const { state, props } = this;
    const { actions } = state;

    const nextState = actions.reduce(action
    // execute the action commands
    // this.state
    // local state, or send it to Redux
    this.setState(nextState);
  }
  // ...
}
```

machine.transition() is just a reducer function!

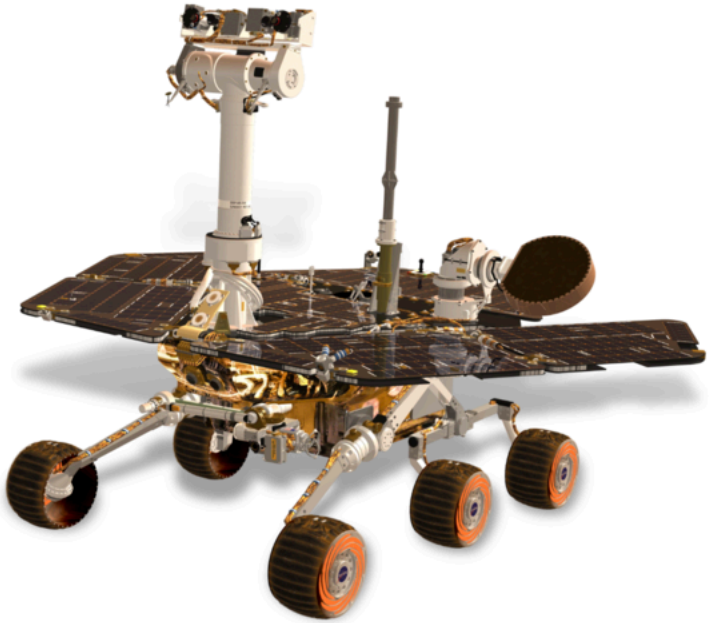
Source: codepen.io/davidkpiano/pen/zWrRye



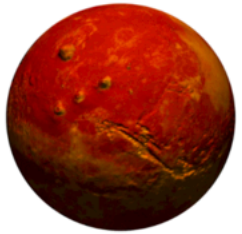
Source: codepen.io/davidkpiano/pen/zWrRye



Advantages of using statecharts

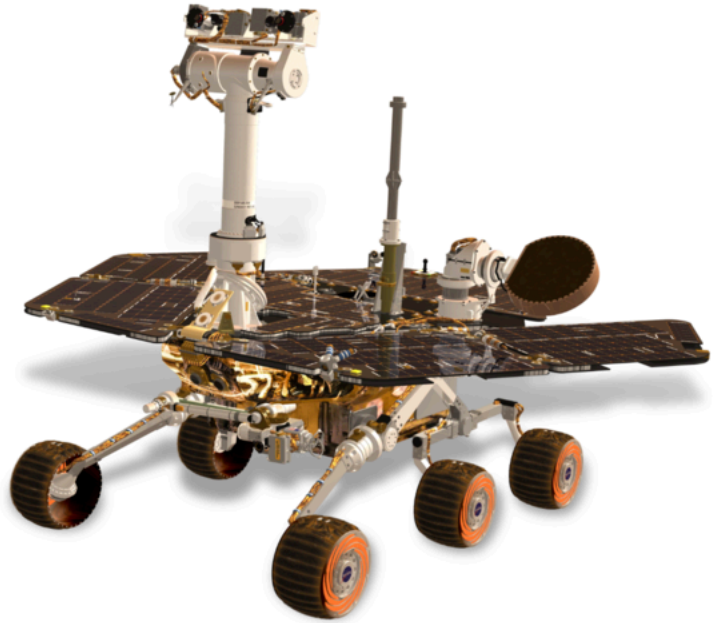


Statechart Autocoding for Curiosity Rover

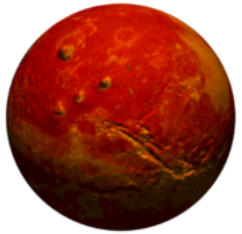


Advantages of using statecharts

Visualized **modeling**

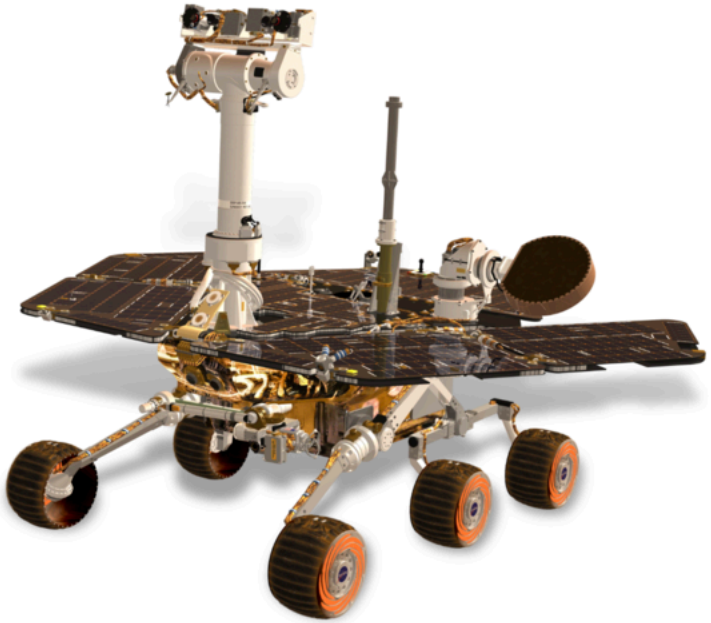


Statechart Autocoding for Curiosity Rover

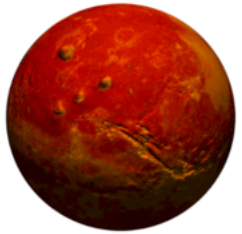


Advantages of using statecharts

Visualized **modeling**
Precise **diagrams**

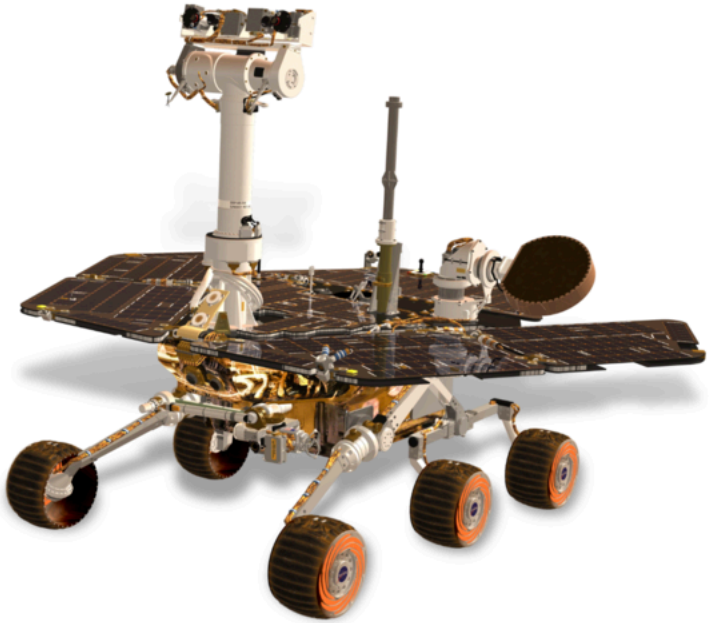


Statechart Autocoding for Curiosity Rover

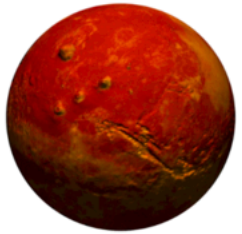


Advantages of using statecharts

Visualized **modeling**
Precise **diagrams**
Automatic **code generation**

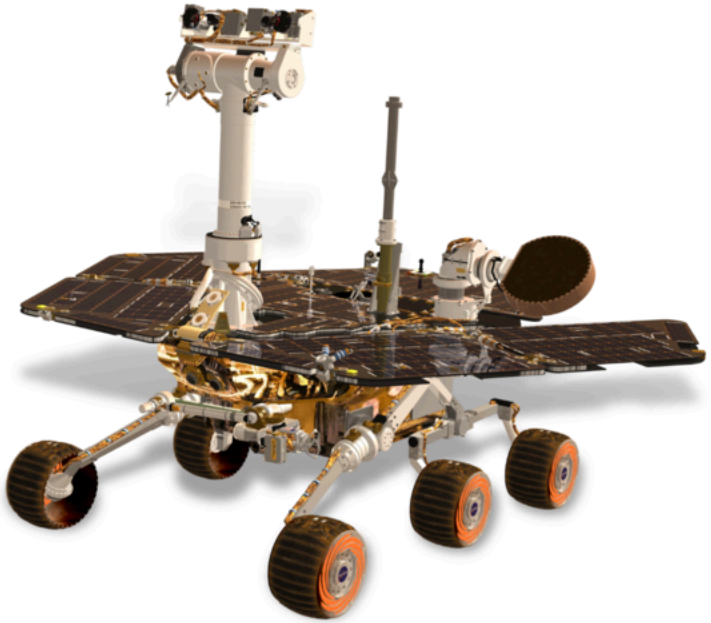


Statechart Autocoding for Curiosity Rover

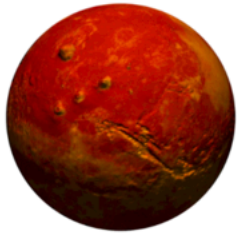


Advantages of using statecharts

Visualized **modeling**
Precise **diagrams**
Automatic **code generation**
Comprehensive **test coverage**

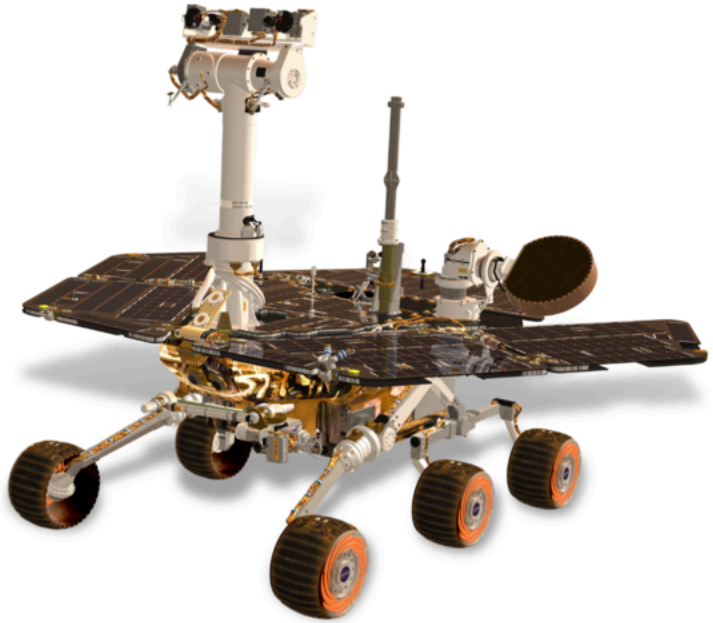


Statechart Autocoding for Curiosity Rover



Advantages of using statecharts

Visualized **modeling**
Precise **diagrams**
Automatic **code generation**
Comprehensive **test coverage**
Accommodation of late-breaking requirements **changes**



Statechart Autocoding for Curiosity Rover

Disadvantages of using statecharts

Disadvantages of using statecharts

Learning curve

Disadvantages of using statecharts

Learning curve

Modeling requires planning ahead

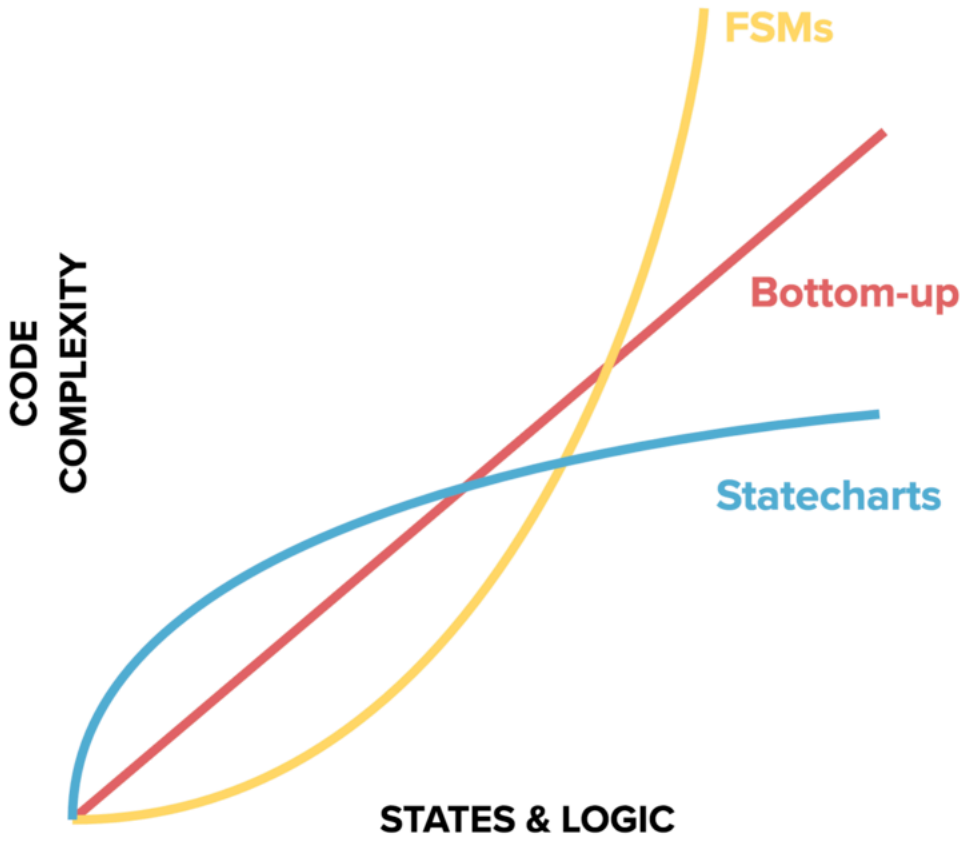
Disadvantages of using statecharts

Learning curve

Modeling requires planning ahead

Not everything can be modeled (yet)

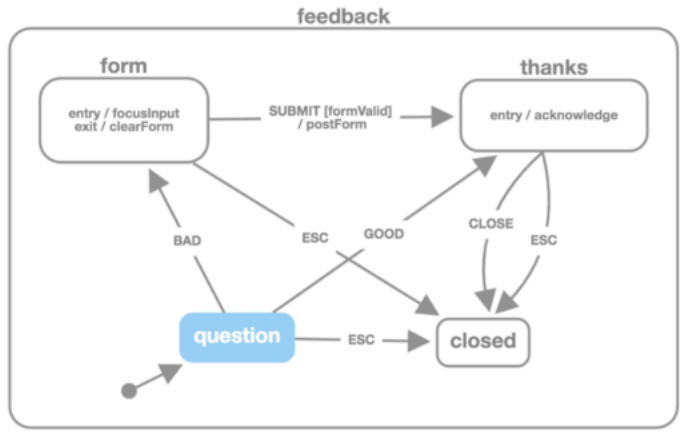
Complexity trade-offs



Statechart visualization?

Statechart visualization?

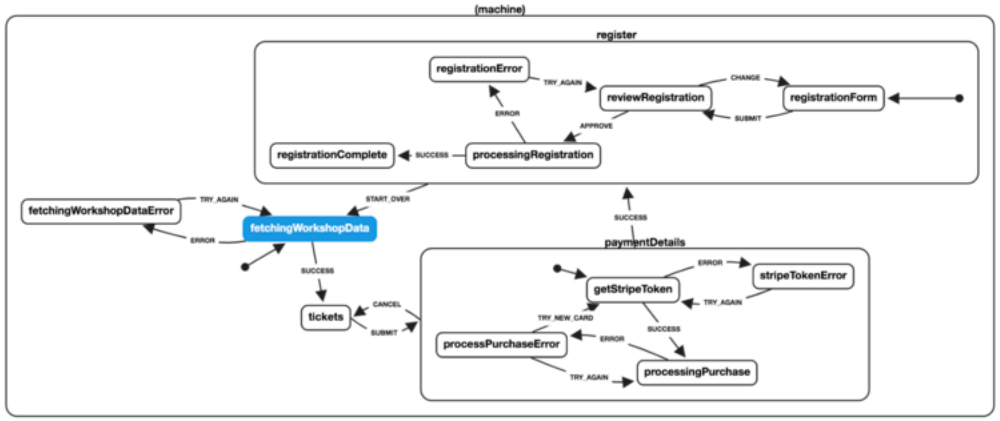
bit.ly/xstate-viz **BETA!**

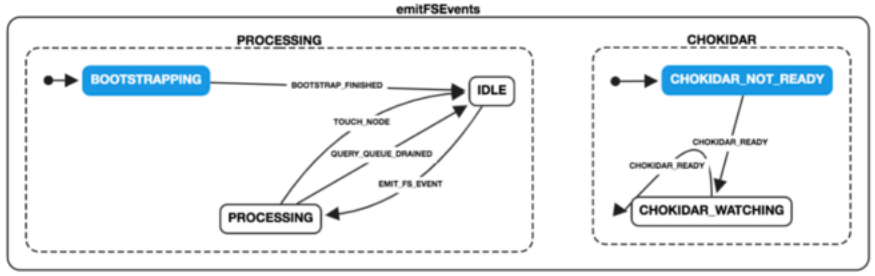


Workshop.me

Live tech training from the industry experts

Gatsby





I just pushed our first couple **statecharts to production** this afternoon. We have a medical calculator in our app that handles dose calculations.

I just pushed our first couple **statecharts to production** this afternoon. We have a medical calculator in our app that handles dose calculations.
The logic was slowly growing more confusing, especially with subtle UX tweaks as the component was maintained.

I just pushed our first couple **statecharts to production** this afternoon. We have a medical calculator in our app that handles dose calculations.
The logic was slowly growing more confusing, especially with subtle UX tweaks as the component was maintained.
We rewrote with a statechart and everything is much clearer! Hopefully less bugs going forward.

- @derekduncan on **spectrum.io/statecharts**

The future of xstate

Full **SCXML** support and conversion
A reactive **interpreter**
Editable **visualization tools**
Analysis and **testing tools**

The future of xstate

Full **SCXML** support and conversion
A reactive **interpreter**
Editable **visualization tools**
Analysis and **testing tools**

The future of xstate

Full **SCXML** support and conversion
A reactive **interpreter**
Editable **visualization tools**
Analysis and **testing tools**

xstate 3.2 just released!

Any state can transition to any state
Actions can be functions 🎉
Support for internal transitions
Support for raised events
Support for transient states/transitions
Support for conditional transition arrays
(WIP) More SCXML conversion

Resources and tools

The World of Statecharts - Erik Mogensen
Statecharts: A Visual Formalism for Complex Systems - David Harel (PDF)
Constructing the User Interface with Statecharts - Ian Horrocks (book)
xstate documentation
How to model the behavior of Redux apps using statecharts - Luca Matteis
React Automata - Michele Bertoli
Pure UI - Guillermo Rauch
Pure UI Control - Adam Solove

Write once, write anywhere

Write once, write anywhere

Learn once, write anywhere

Write once, write anywhere

Learn once, write anywhere

Model once, implement anywhere

Write once, write anywhere

Learn once, write anywhere

Model once, implement anywhere

Let's improve the way we develop.
Thank you React Finland!

@davidkpiano · React Finland 2018