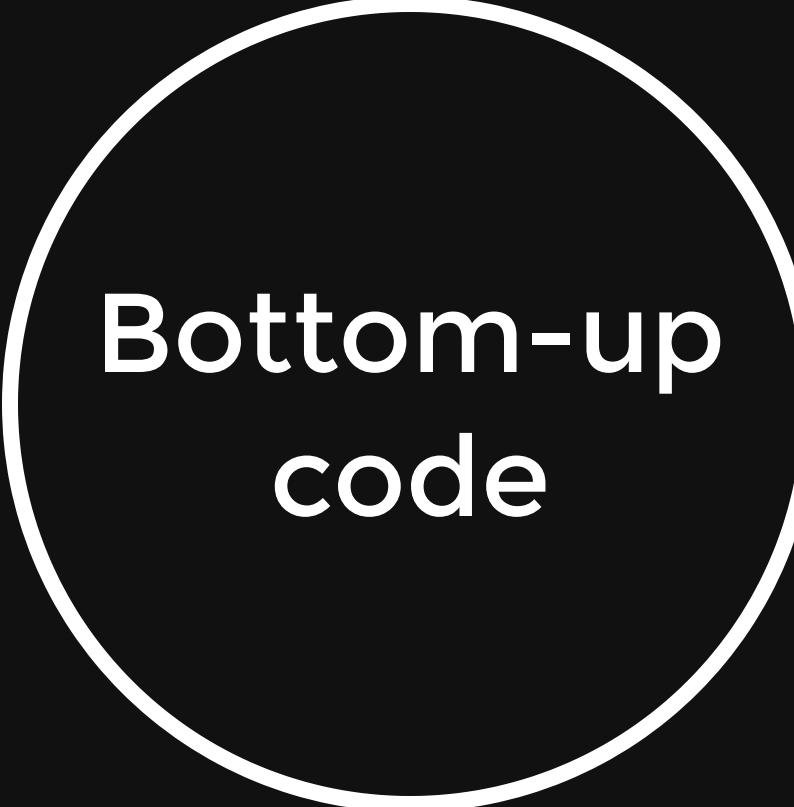


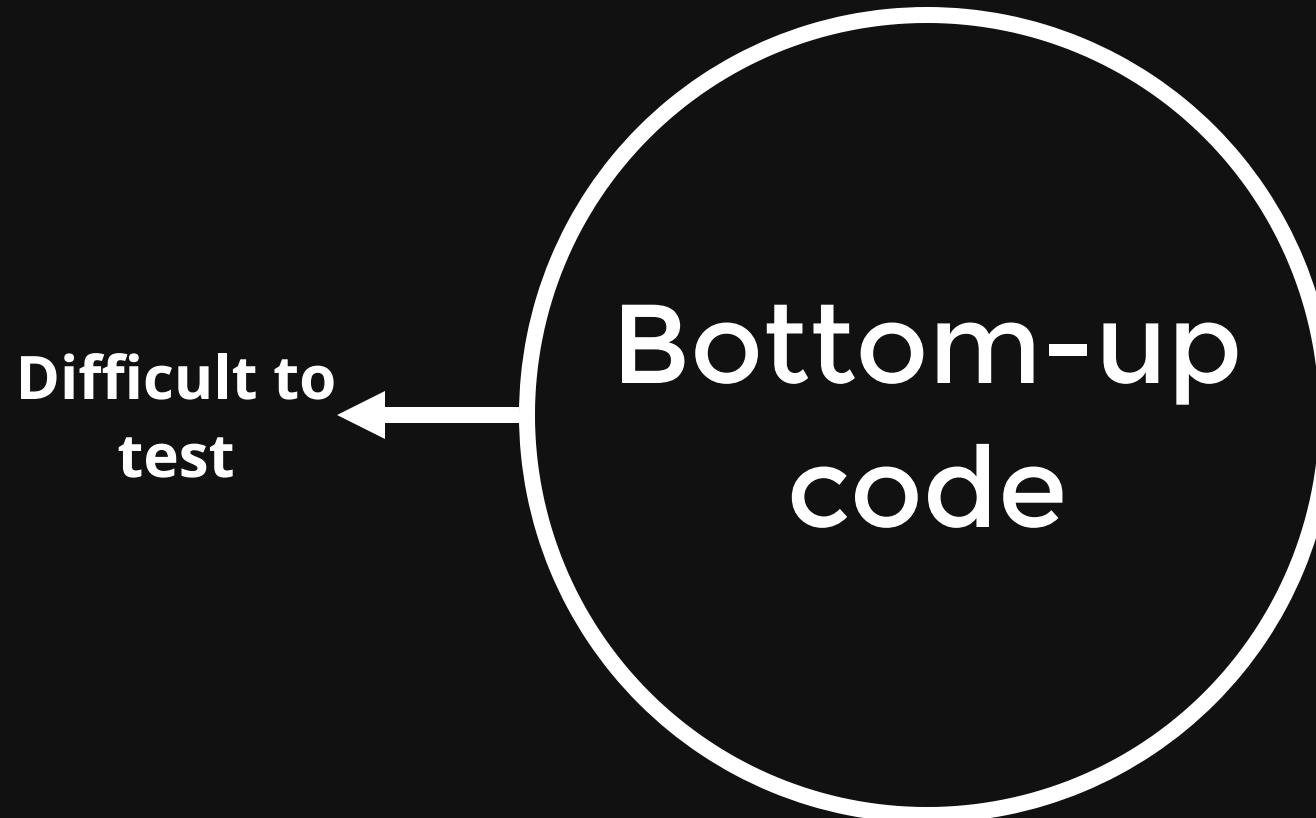
Frontend **Masters**

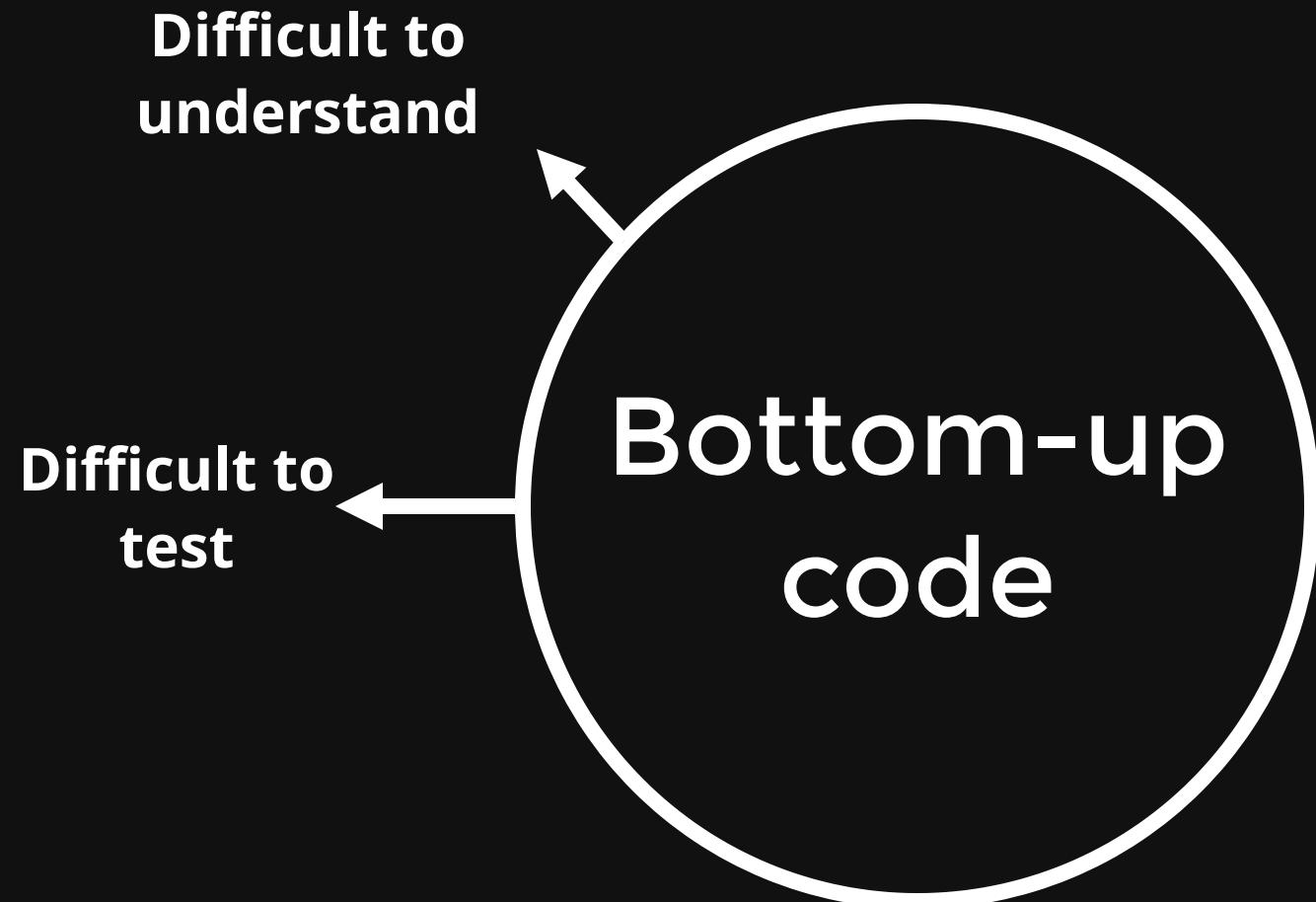
JavaScript State Machines & XSTATE

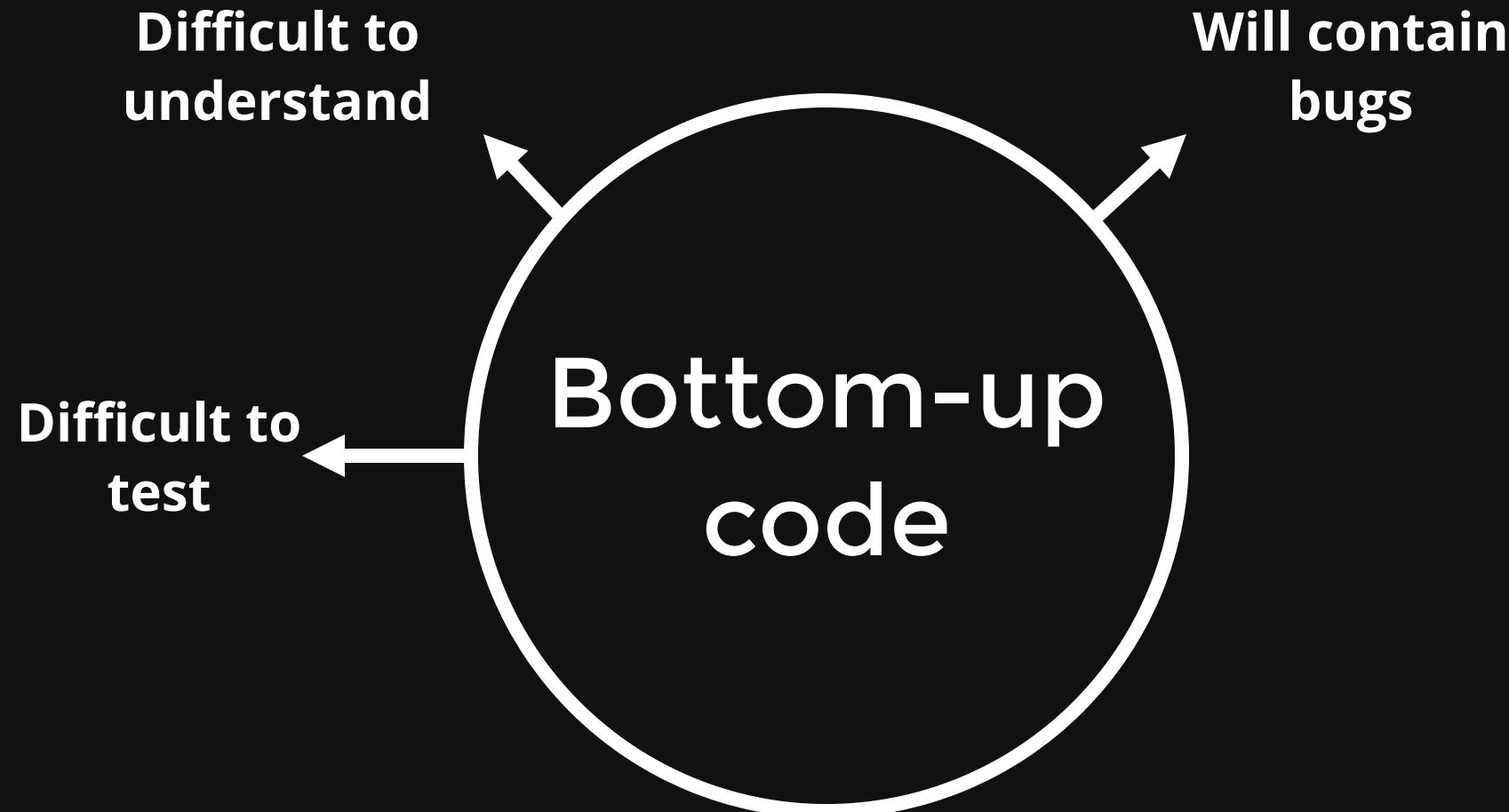
David Khourshid · [@davidkpiano](https://twitter.com/davidkpiano)

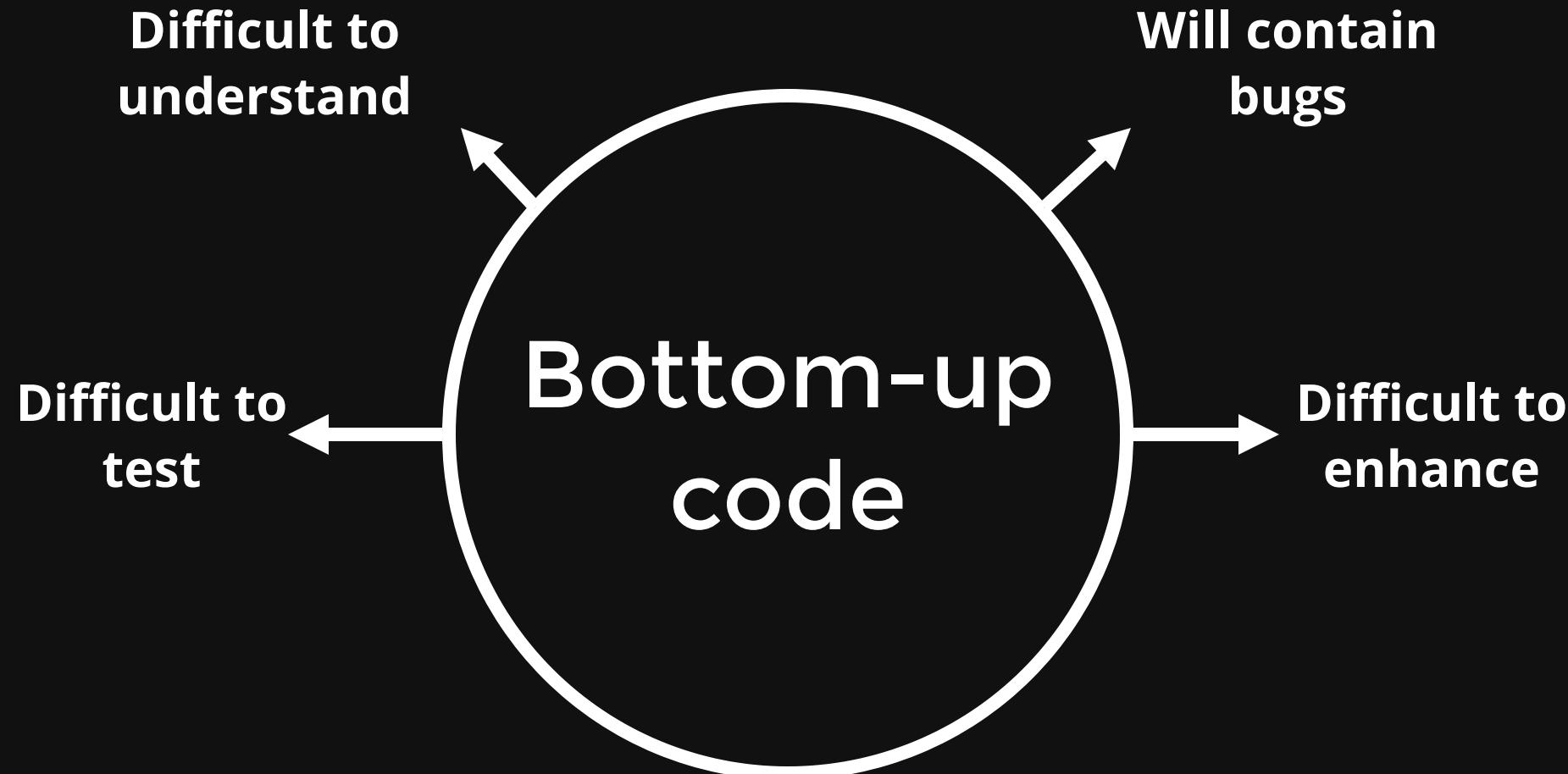


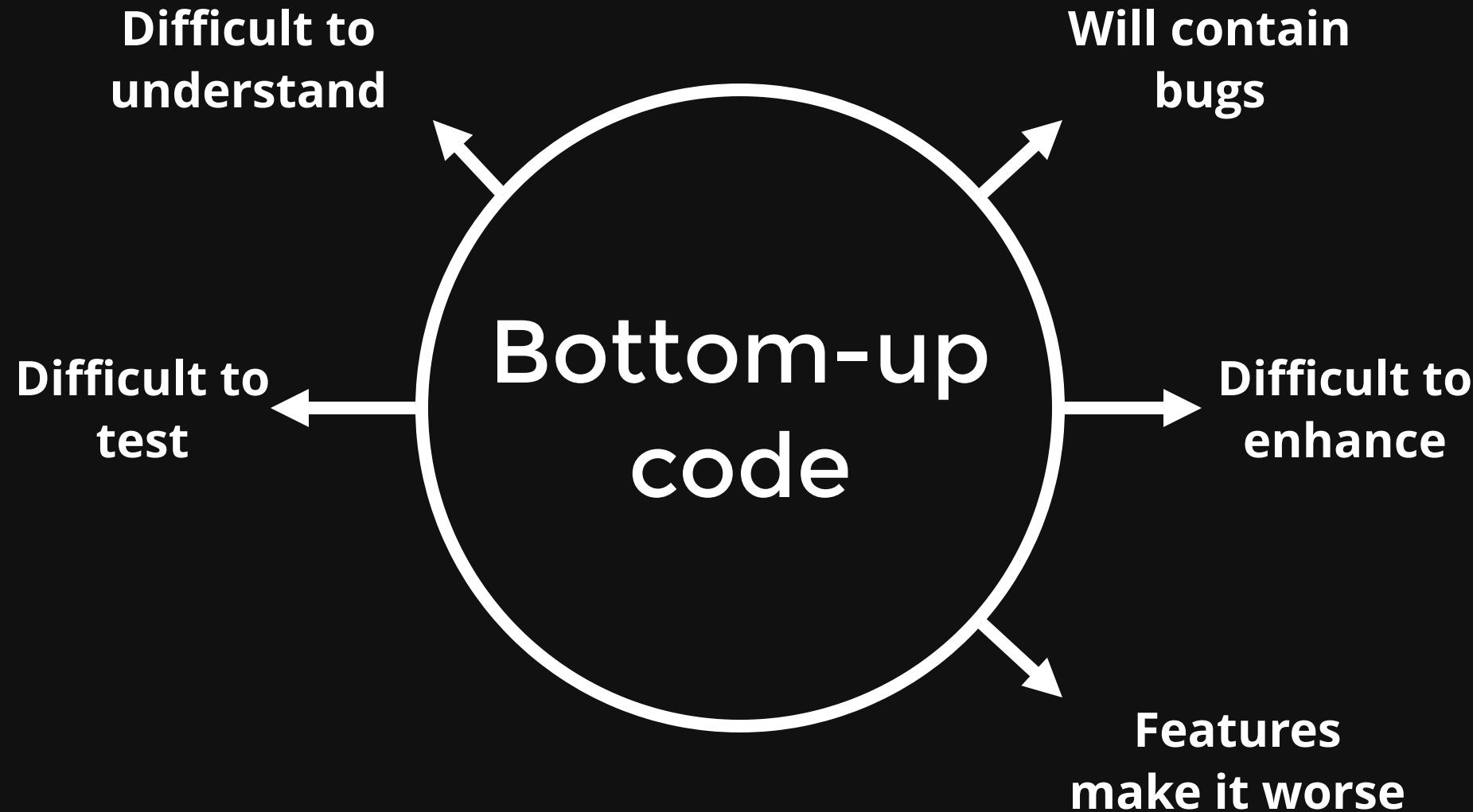
Bottom-up
code



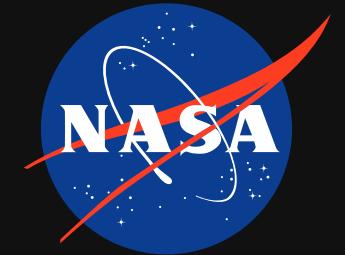




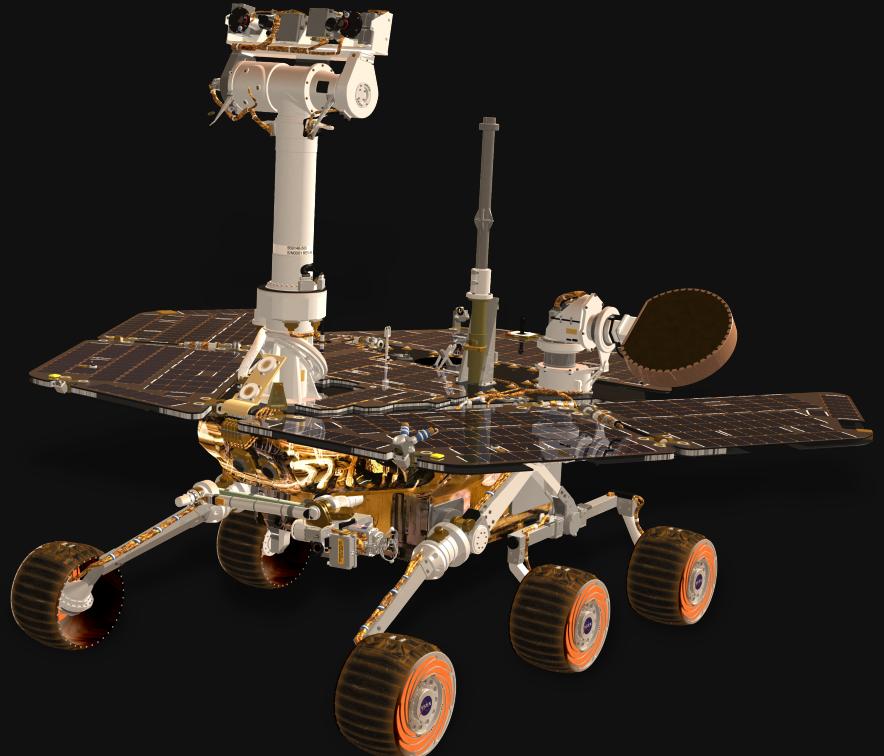




Why use state machines & statecharts?

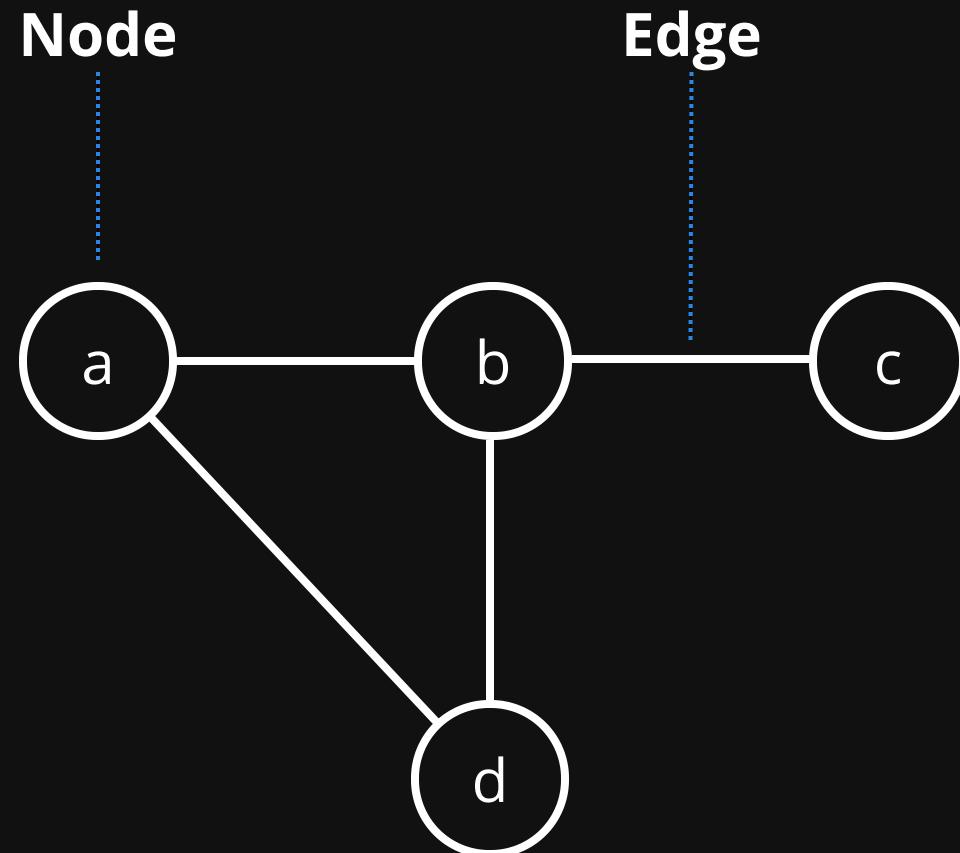


- Visualized **modeling**
- Precise **diagrams**
- Automatic **code generation**
- Comprehensive **test coverage**
- Accommodation of late-breaking requirements **changes**



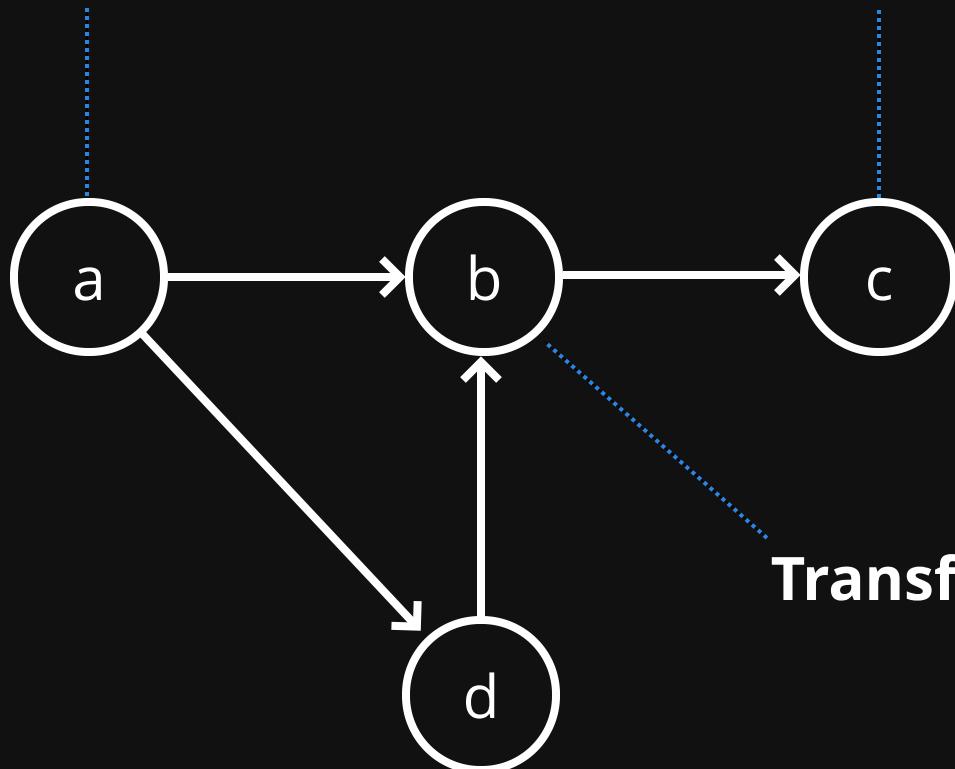
Statechart Autocoding for Curiosity Rover

Graphs



Directed graphs

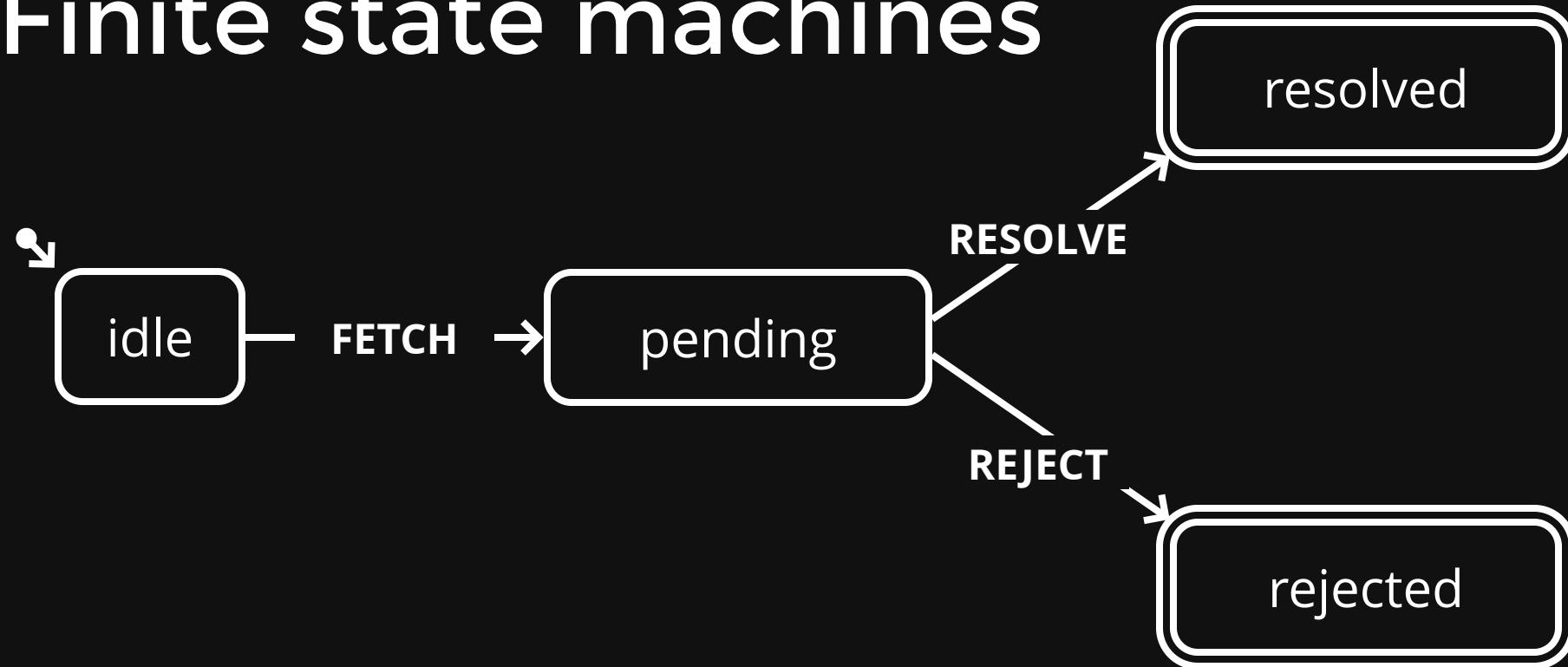
Source node



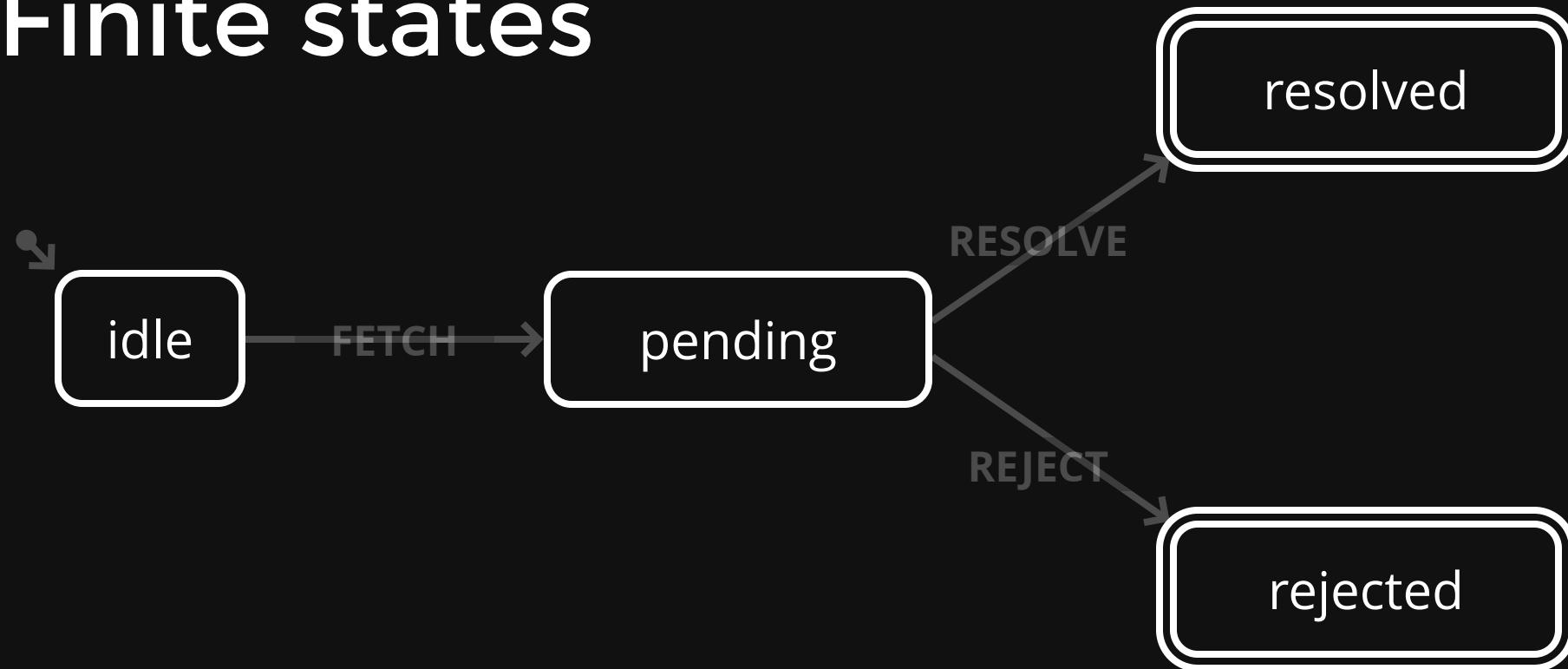
Sink node

Transfer node

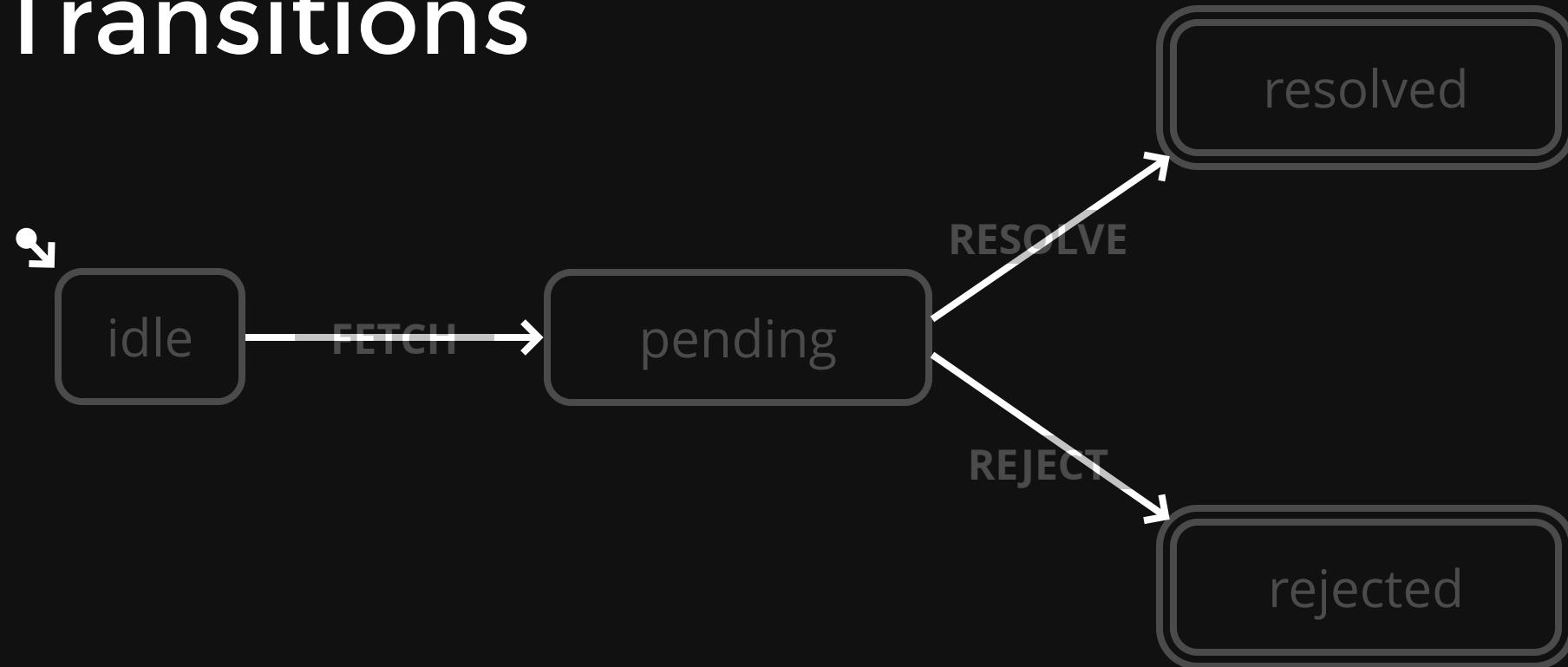
Finite state machines



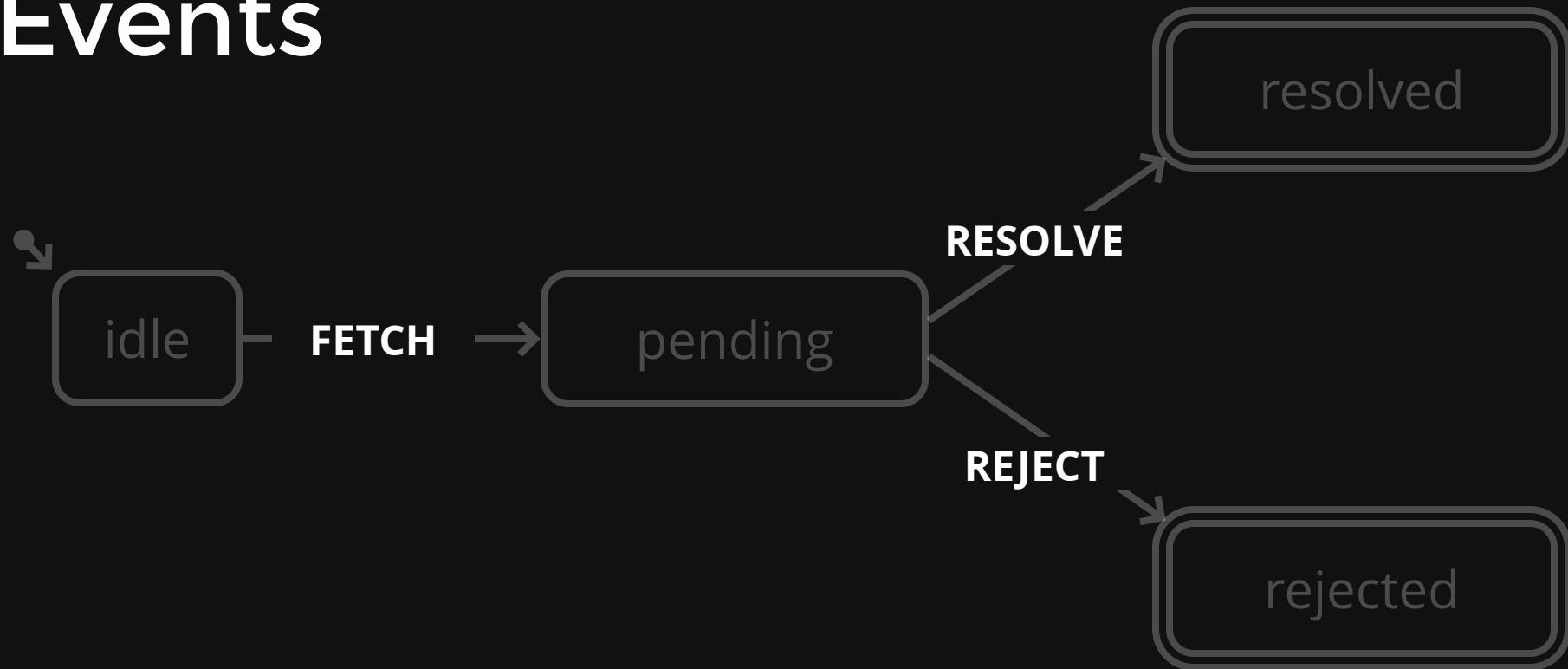
Finite states



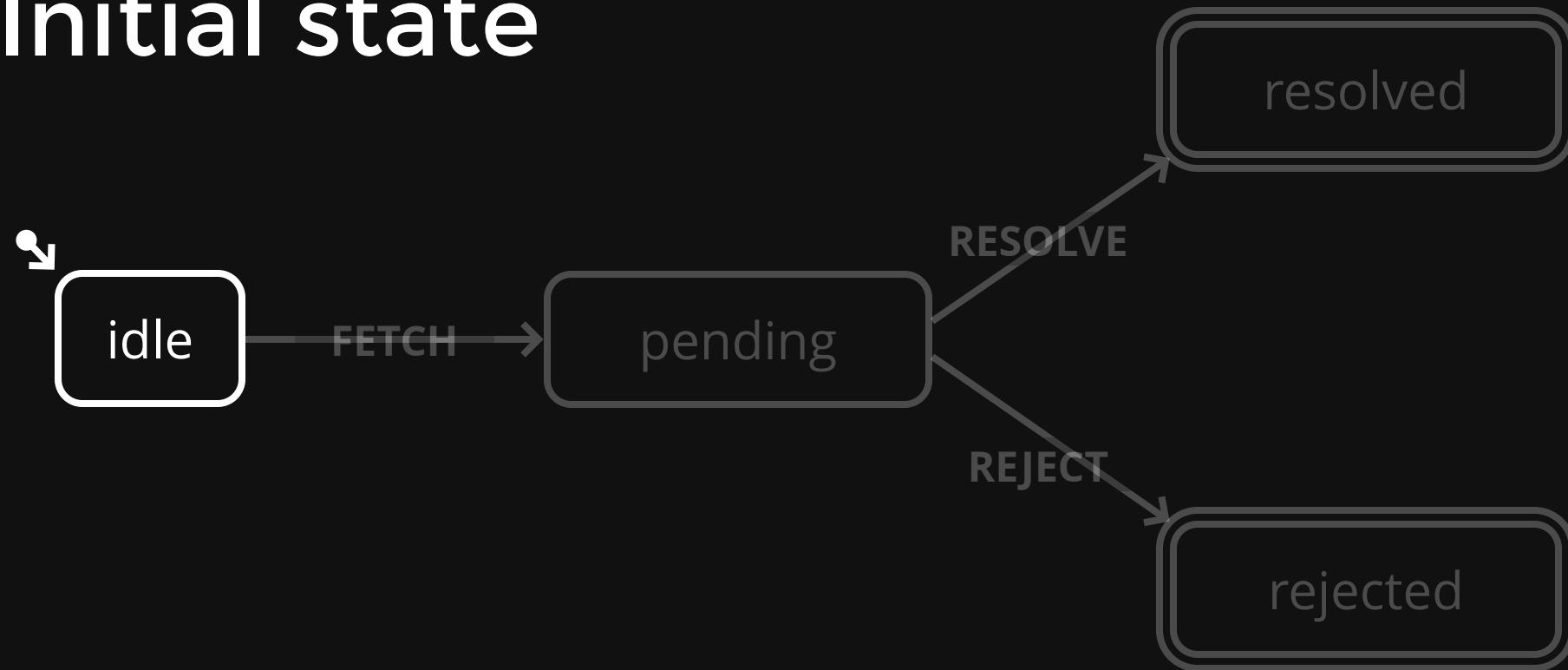
Transitions



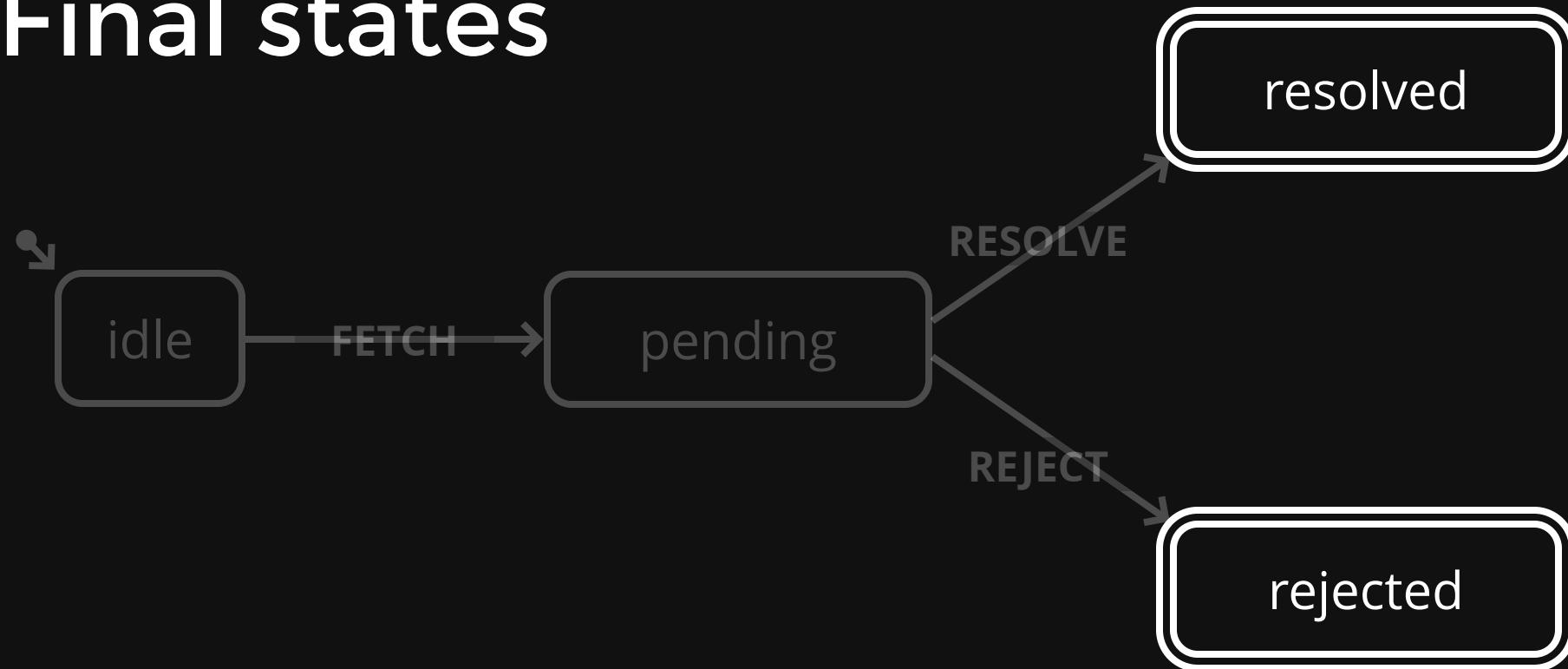
Events



Initial state



Final states



Using switch statements

```
1
2 function transition(state, event) {
3     switch (state) {
4         case 'idle':
5             switch (event) {
6                 case 'FETCH':
7                     return 'pending';
8                 default:
9                     return state;
10            }
11        case 'pending':
12            switch (event) {
13                case 'RESOLVE':
14                    return 'resolved';
15                case 'REJECT':
16                    return 'rejected';
17                default:
18                    return state;
19            }
20        case 'resolved':
21        case 'rejected':
22        default:
23            return state;
24    }
25 }
```

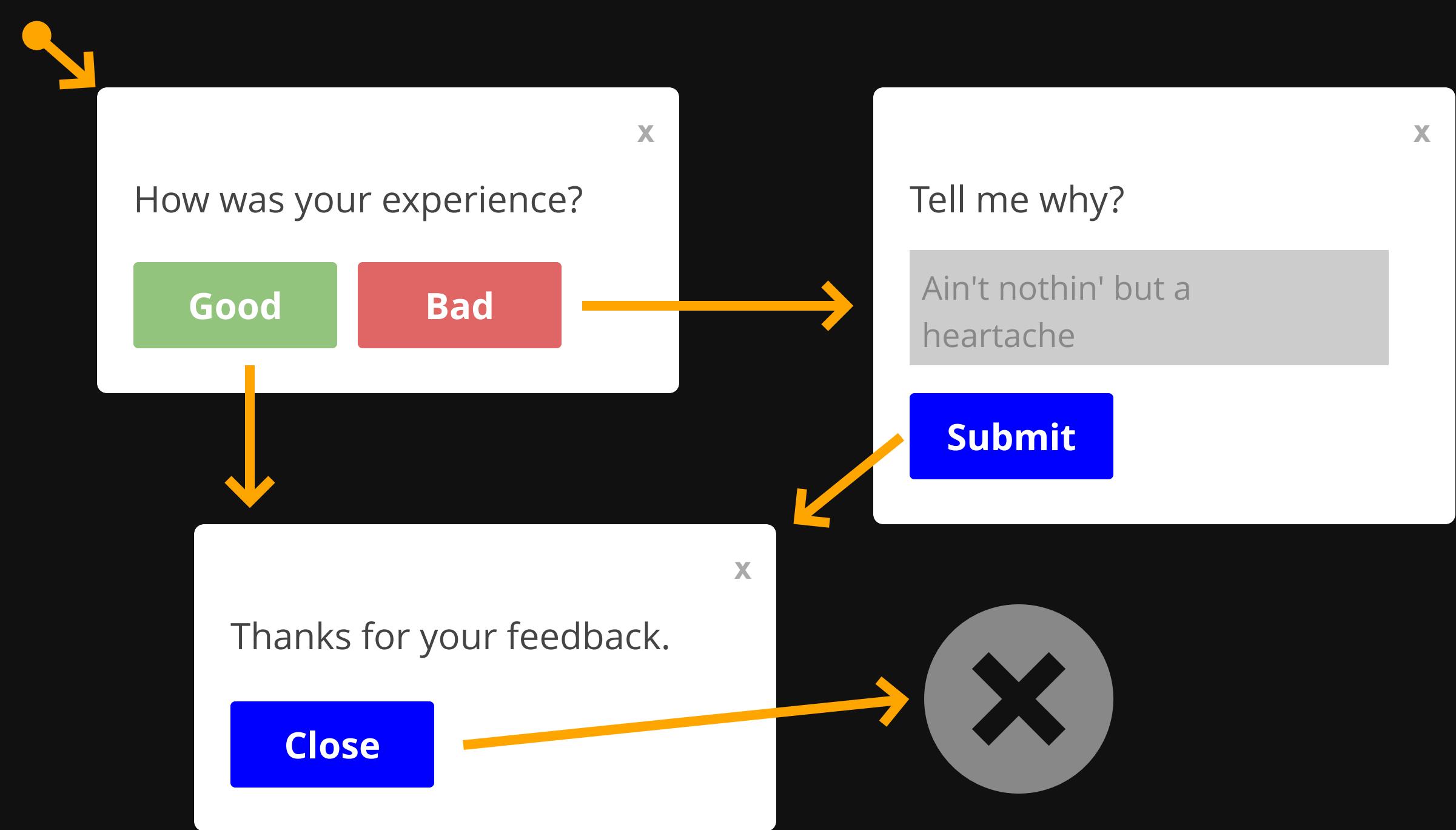
Using objects

```
1
2 const machine = {
3   initial: 'idle',
4   states: {
5     idle: {
6       on: {
7         FETCH: 'pending'
8       }
9     },
10    pending: {
11      on: {
12        RESOLVE: 'resolved',
13        REJECT: 'rejected'
14      }
15    },
16    resolved: {},
17    rejected: {}
18  }
19}
20
21 function transition(state, event) {
22   return machine
23     .states[state]?
24     .on?.[event]
25     || state;
26 }
```

Interpreting state machines

```
1 // Keep track of the current state
2 let currentState = machine.initial;
3
4 // Receive events
5 function send(event) {
6
7     // Determine the next state
8     const nextState =
9         transition(currentState, event);
10
11    // Update the current state
12    currentState = nextState;
13 }
14
15 // Send some event
16 send("CLICK");
```

Exercise 01



Add states

```
1 // Setup the machine
2 const feedbackMachine = {
3   initial: 'question',
4   states: {
5     question: {
6       // ...
7     },
8     form: {
9       // ...
10    },
11    thanks: {
12      // ...
13    },
14    closed: {
15      // ...
16    }
17  }
18 }
```

Add events

```
1 // Setup the machine
2 const feedbackMachine = {
3   initial: 'question',
4   states: {
5     question: {
6       on: {
7         CLICK_GOOD: 'thanks',
8         CLICK_BAD: 'form'
9       }
10    },
11    form: {
12      on: {
13        SUBMIT: 'thanks'
14      }
15    },
16    thanks: {
17      on: {
18        CLOSE: 'closed'
19      }
20    },
21    closed: {}
22  }
23};
```

Add transition function

```
1 // Setup the machine
2 const feedbackMachine = {
3   // ...
4 };
5
6 // Setup the state transition function
7 function transition(state, event) {
8   return feedbackMachine
9     .states[state]?
10       .on?.[event]
11     || state;
12 }
```

Track current state

```
1 // Setup the machine
2 const feedbackMachine = {
3   // ...
4 };
5
6 let currentState = feedbackMachine.initial;
7
8 // Setup the state transition function
9 function transition(state, event) {
10   return feedbackMachine
11     .states[state]?
12     .on?.[event]
13     || state;
14 }
```

Receive events

```
1 // Setup the machine
2 const feedbackMachine = {
3   // ...
4 };
5
6 let currentState = feedbackMachine.initial;
7
8 // Setup the state transition function
9 function transition(state, event) {
10   return feedbackMachine
11     .states[state]?
12       .on?.[event]
13     || state;
14 }
15
16 function send(event) {
17   const nextState = transition(currentState, eve
18
19   currentState = nextState;
20 }
```

Interpreting state machines

```
1
2 function interpret(machine) {
3   // Keep track of the current state
4   // in a closure
5   let currentState = machine.initial;
6
7   // Receive events
8   const send = (event) => {
9     const nextState =
10       transition(currentState, event);
11
12     // Update the current state
13     currentState = nextState;
14   }
15
16   // Expose the send function
17   return {
18     send
19   };
20 }
```

Interpreting state machines

```
1
2 function interpret(machine) {
3   let currentState = machine.initial;
4
5   // Keep track of listeners
6   const listeners = new Set();
7
8   const send = (event) => {
9     const nextState =
10       transition(currentState, event);
11
12     currentState = nextState;
13   }
14
15   // Register listeners
16   const onTransition = (listener) => {
17     listeners.add(listener);
18   }
19
20   return {
21     // Expose way to register listeners
22     onTransition,
23     send
24   };
25 }
```

Interpreting state machines

```
1  function interpret(machine) {
2    let currentState = machine.initial;
3    let status = 1; // 1 means started, 2 means stopped
4
5    const listeners = new Set();
6
7    const send = (event) => {
8      // Don't accept events if stopped
9      if (status === 2) { return; }
10     // ...
11   }
12
13
14  const onTransition = (listener) => {
15    listeners.add(listener);
16  }
17
18  // Cleanup
19  const stop = () => {
20    status = 2;
21    listeners.clear();
22  }
23
24  return {
25    onTransition,
26    // Expose cleanup
27    stop,
28    send
29  };
}
```

X STATE

npm install xstate

yarn add xstate

 xstate.js.org/docs

Creating a machine

```
1 import { createMachine } from 'xstate';
2
3 const feedbackMachine = createMachine({
4   initial: 'question',
5   states: {
6     question: {
7       // ...
8     },
9     form: {
10      // ...
11    },
12    thanks: {
13      // ...
14    },
15    closed: {
16      // ...
17    }
18  }
19});
```

Adding transitions

Shorthand syntax

```
1 import { createMachine } from 'xstate';
2
3 const feedbackMachine = createMachine({
4   initial: 'question',
5   states: {
6     question: {
7       on: {
8         CLICK_GOOD: 'thanks',
9         CLICK_BAD: 'form'
10      }
11    },
12    form: {
13      on: {
14        SUBMIT: 'thanks'
15      }
16    },
17    thanks: {
18      on: {
19        CLOSE: 'closed'
20      }
21    },
22    closed: {
23      type: 'final'
24    }
25  }
26});
```

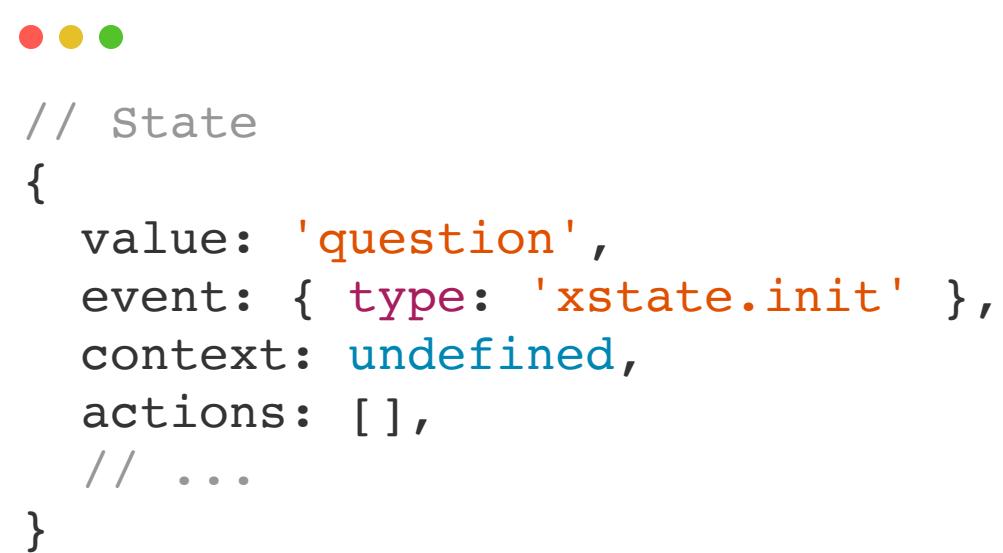
Adding transitions

Object syntax

```
1 import { createMachine } from 'xstate';
2
3 const feedbackMachine = createMachine({
4   initial: 'question',
5   states: {
6     question: {
7       on: {
8         CLICK_GOOD: 'thanks',
9         CLICK_BAD: 'form'
10      }
11    },
12    form: {
13      on: {
14        SUBMIT: {
15          target: 'thanks'
16        }
17      }
18    },
19    thanks: {
20      on: {
21        CLOSE: 'closed'
22      }
23    },
24    closed: {
25      type: 'final'
26    }
27  }
28});
```

State objects

```
1 import { createMachine } from 'xstate';
2
3 const feedbackMachine = createMachine({
4   // ...
5 });
6
7 const initialState = feedbackMachine.initialState;
8
9 console.log(initialState);
```



The screenshot shows a macOS terminal window with three colored window control buttons (red, yellow, green) at the top. The main area contains the following code output:

```
// State
{
  value: 'question',
  event: { type: 'xstate.init' },
  context: undefined,
  actions: [ ],
  // ...
}
```

Event objects

```
1 import { createMachine } from 'xstate';
2
3 const feedbackMachine = createMachine({
4   // ...
5 });
6
7 const initialState = feedbackMachine.initialState;
8
9 // An event object
10 const clickGoodEvent = {
11   type: 'CLICK_GOOD'
12 };
13
14 // An event object with payload
15 const submitEvent = {
16   type: 'SUBMIT',
17   feedback: 'Very good, thank you'
18 };
```

Transitioning

```
1 import { createMachine } from 'xstate';
2
3 const feedbackMachine = createMachine({
4   // ...
5 });
6
7 const initialState = feedbackMachine.initialState;
8
9 const clickGoodEvent = {
10   type: 'CLICK_GOOD'
11 };
12
13 // Determine next state based on
14 // current state and event
15 const nextState = feedbackMachine
16   .transition(initialState, clickGoodEvent);
17
18 // The event type can be used
19 // if there is no "payload"
20 const otherState = feedbackMachine
21   .transition(nextState, "CLOSE");
22
23 // This is equivalent to passing
24 // in an event object
25 const otherState = feedbackMachine
26   .transition(nextState, { type: "CLOSE" }));
```

Interpreting machines

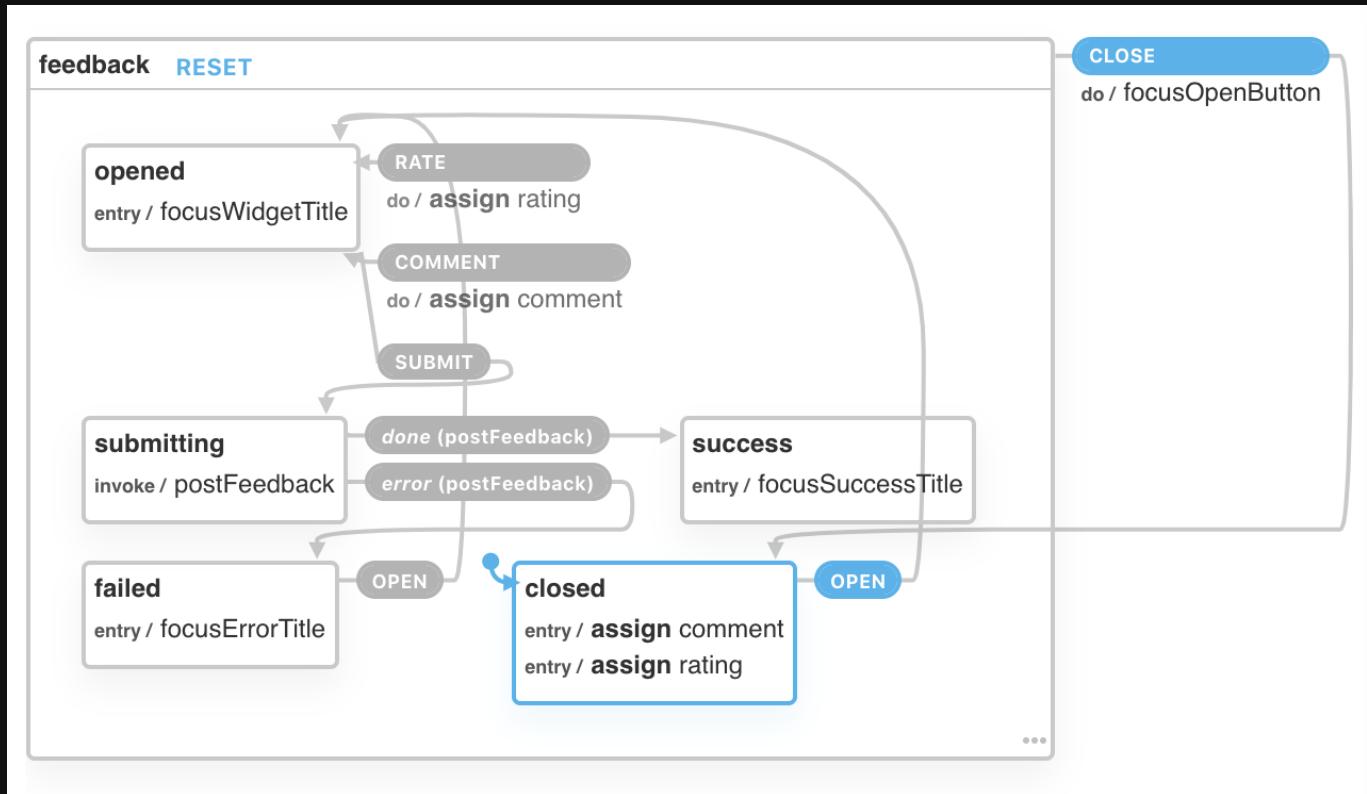
```
1 import { createMachine, interpret } from 'xstate';
2
3 const feedbackMachine = createMachine({
4   // ...
5 });
6
7 // Create a "service" instance
8 const feedbackService = interpret(feedbackMachine);
9
10 // You can add a listener that receives
11 // the current state on every transition
12 feedbackService.onTransition(state => {
13   console.log(state);
14 });
15
16 // At first, the service didn't start yet...
17 // So start it!
18 feedbackService.start();
```

Interpreting machines

```
1 import { createMachine, interpret } from 'xstate';
2
3 const feedbackMachine = createMachine({
4   // ...
5 });
6
7 // Create a "service" instance and start it
8 const feedbackService = interpret(feedbackMachine);
9   .onTransition(state => {
10     console.log(state);
11   })
12   .start();
13
14 // Events can be sent to the service
15 service.send({ type: 'CLICK_GOOD' });
16
17 // If there is no payload, only the
18 // event type is necessary
19 service.send("CLOSE");
20
21 // When the service no longer needs to run,
22 // stop it for disposal/cleanup
23 service.stop();
```

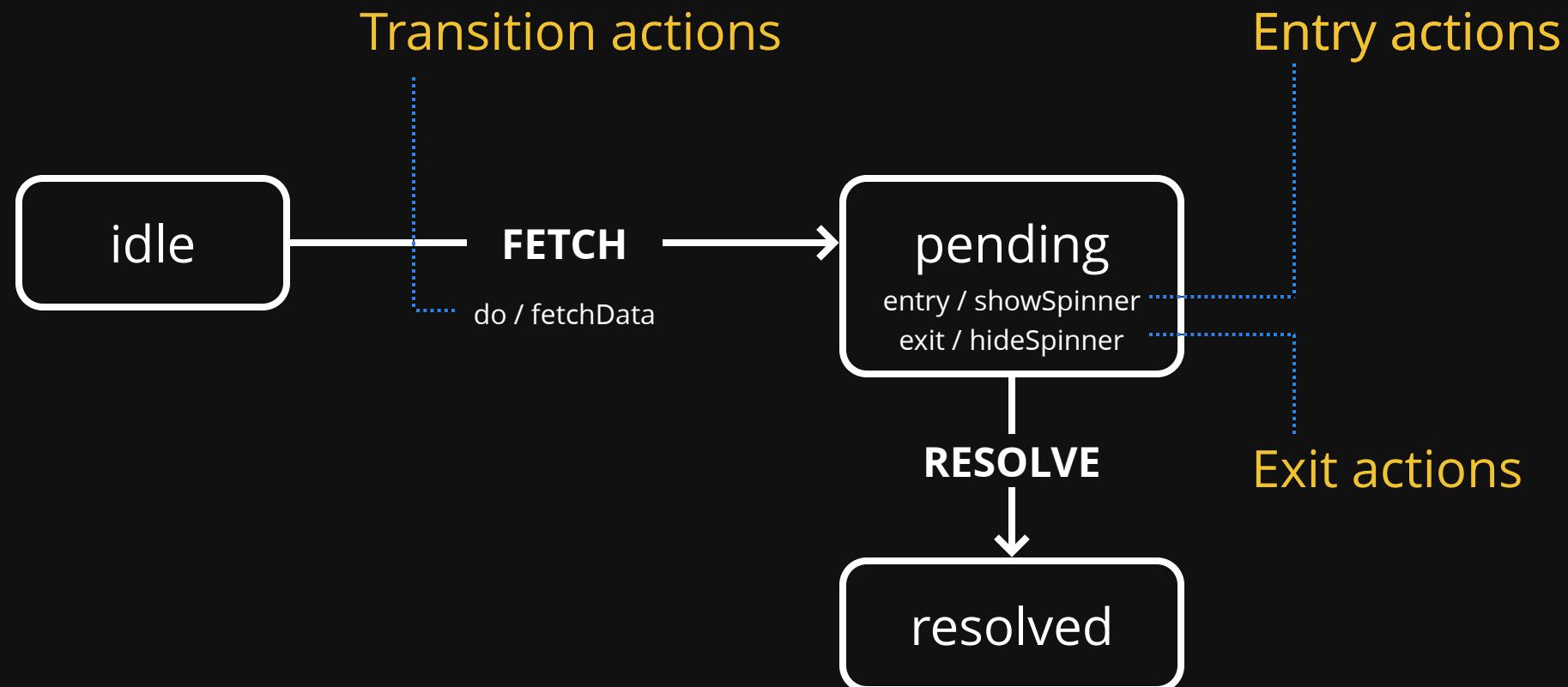
Visualizing machines

xstate.js.org/viz

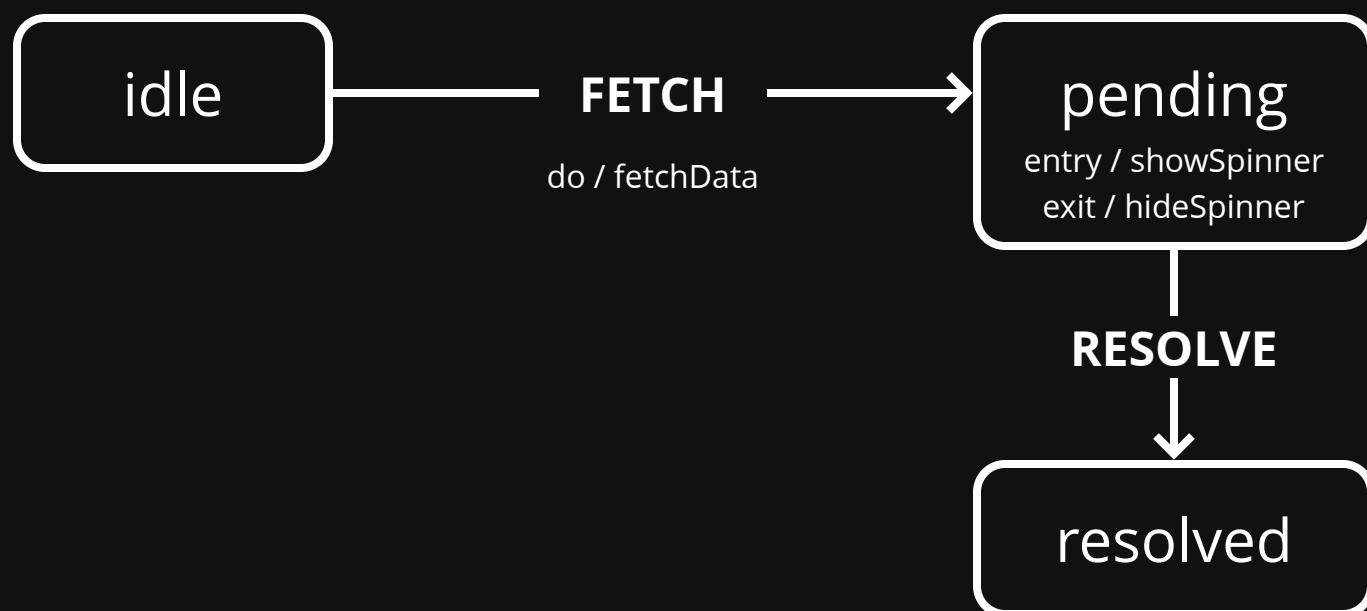


Exercise 03

Actions



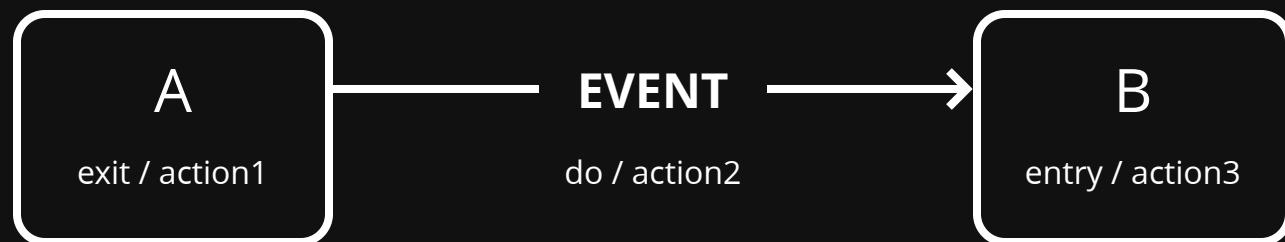
Actions



- state: **idle**
- event: **FETCH**
- state: **pending**
 - action: **fetchData**
 - action: **showSpinner**
- event: **RESOLVE**
- state: **resolved**
 - action: **hideSpinner**

Action order

1. Exit action(s)
2. Transition action(s)
3. Entry action(s)



Actions in XState

```
1 import { createMachine } from 'xstate';
2
3 const someMachine = createMachine({
4   initial: 'active',
5   states: {
6     active: {
7       entry: (context, event) => {
8         console.log('Entered active');
9       },
10    on: {
11      CLICK: {
12        target: 'inactive',
13        actions: (context, event) => {
14          console.log('Clicked on active');
15          console.log(event);
16        }
17      }
18    },
19    exit: (context, event) => {
20      console.log('Exited active')
21    }
22  },
23  inactive: {
24    entry: (context, event) => {
25      console.log('Entered inactive');
26    }
27  }
28}
29});
```

Actions in XState

```
1 import { createMachine } from 'xstate';
2
3 const enterActive = () => {
4   console.log('Entered active');
5 };
6
7 const clickActive = () => {
8   console.log('Clicked on active');
9 };
10
11 const exitActive = () => {
12   console.log('Exited active');
13 };
14
15 const enterInactive = () => {
16   console.log('Entered inactive');
17 };
18
19 const someMachine = createMachine({
20   initial: 'active',
21   states: {
22     active: {
23       entry: enterActive,
24       on: {
25         CLICK: {
26           target: 'inactive',
27           actions: clickActive
28         }
29       },
30     },
31   },
32   actions: {
33     click: {
34       target: 'inactive',
35       actions: clickActive
36     }
37   }
38 }, {
39   actions: {
40     click: {
41       target: 'inactive',
42       actions: clickActive
43     }
44   }
45 }
```

Actions in XState

Multiple actions

```
1 import { createMachine } from 'xstate';
2
3 const sendTelemetry = () => {/* ... */}
4
5 const enterActive = () => {
6   console.log('Entered active');
7 };
8
9 const clickActive = () => {
10   console.log('Clicked on active');
11 };
12
13 const exitActive = () => {
14   console.log('Exited active');
15 };
16
17 const enterInactive = () => {
18   console.log('Entered inactive');
19 };
20
21 const someMachine = createMachine({
22   initial: 'active',
23   states: {
24     active: {
25       entry: [enterActive, sendTelemetry],
26       on: {
27         CLICK: {
28           target: 'inactive',
29           actions: clickActive
30         }
31       }
32     }
33   }
34 });
35
36 someMachine.on('CLICK').then(() => {
37   console.log('Clicked');
38 });
39
40 someMachine.send('CLICK');
```

Actions in XState

Multiple actions

```
1 import { createMachine } from 'xstate';
2
3 const someMachine = createMachine({
4   initial: 'active',
5   states: {
6     active: {
7       entry: [ 'enterActive', 'sendTelemetry' ],
8       on: {
9         CLICK: {
10           target: 'inactive',
11           actions: 'clickActive'
12         }
13       },
14       exit: 'exitActive'
15     },
16     inactive: {
17       entry: 'enterInactive'
18     }
19   }
20 }, {
21   actions: {
22     sendTelemetry: () => {/* ... */},
23     enterActive: () => {
24       console.log('Entered active');
25     },
26     clickActive: () => {
27       console.log('Clicked on active');
28     },
29     exitActive: () => {
```

Exercise 04

Statecharts

Statecharts

Invented by David Harel

Statecharts: A Visual Formalism
for Complex Systems (1987)

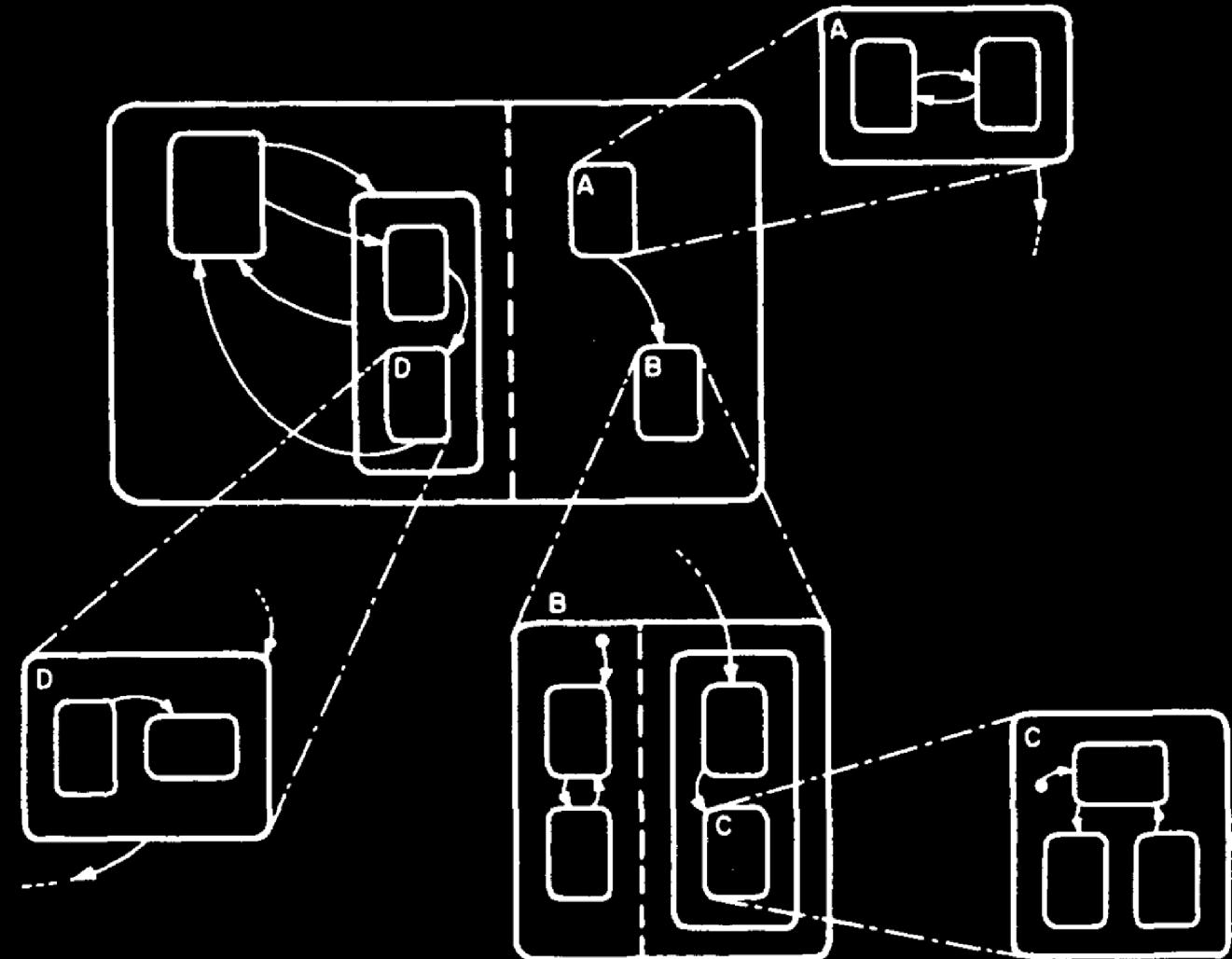


Fig. 36.

Extended state

```
1 import { createMachine } from 'xstate';
2
3 const lightbulbMachine = createMachine({
4   initial: 'turnedOff',
5   states: {
6     turnedOff: {
7       on: {
8         TOGGLE: {
9           target: 'turnedOn',
10          actions: 'turnOn'
11        }
12      }
13    },
14    turnedOn: {
15      on: {
16        TOGGLE: {
17          target: 'turnedOff',
18          actions: 'turnOff'
19        }
20      }
21    }
22  }
23});
```

Extended state

```
1 import { createMachine } from 'xstate';
2
3 const lightbulbMachine = createMachine({
4   initial: 'turnedOff',
5   context: {
6     count: 0 // # of times it was turned on
7   },
8   states: {
9     turnedOff: {
10       on: {
11         TOGGLE: {
12           target: 'turnedOn',
13           actions: 'turnOn'
14         }
15       }
16     },
17     turnedOn: {
18       on: {
19         TOGGLE: {
20           target: 'turnedOff',
21           actions: 'turnOff'
22         }
23       }
24     }
25   }
26 }) ;
```

Assignment

```
1 import { createMachine, assign } from 'xstate';
2
3 const lightbulbMachine = createMachine({
4   initial: 'turnedOff',
5   context: {
6     count: 0 // # of times it was turned on
7   },
8   states: {
9     turnedOff: {
10       on: {
11         TOGGLE: {
12           target: 'turnedOn',
13           actions: 'turnOn'
14         }
15       }
16     },
17     turnedOn: {
18       entry: assign({
19         count: (context, event) => {
20           return context.count + 1;
21         }
22       }),
23       on: {
24         TOGGLE: {
25           target: 'turnedOff',
26           actions: 'turnOff'
27         }
28       }
29     }
30   }
31 }
```

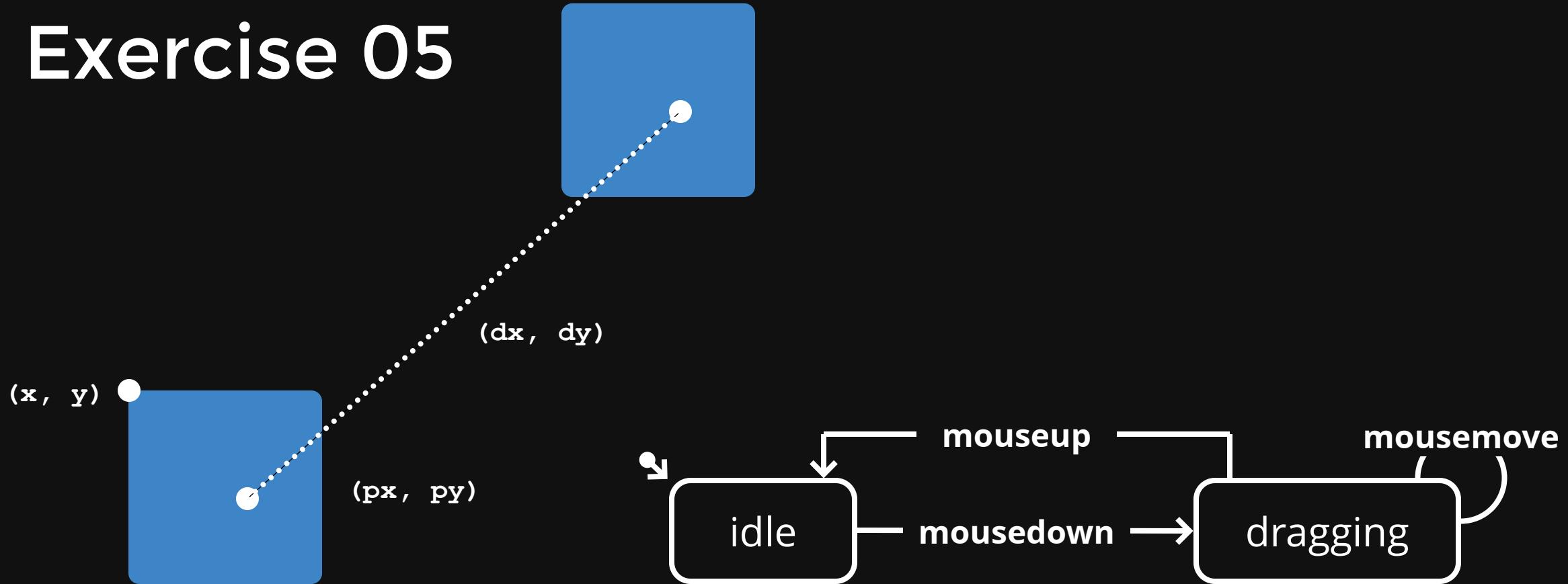
Assignment

```
1 import { createMachine, assign } from 'xstate';
2
3 const assignAction = assign({
4   // static assignment
5   message: 'Hello',
6
7   // assignment based on context + event
8   count: (context, event) => {
9     return context.count + event.value;
10  }
11 });
12
13 console.log(assignAction);
```

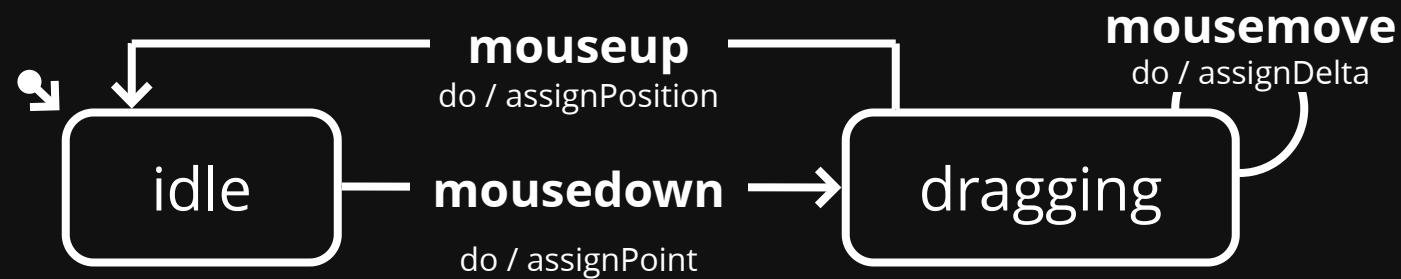
Assignment

```
1 import { createMachine, assign } from 'xstate';
2
3 const assignAction = assign(context, event) => {
4   return {
5     message: 'hello',
6     count: context.count + event.value
7   };
8 });
9
10 console.log(assignAction);
```

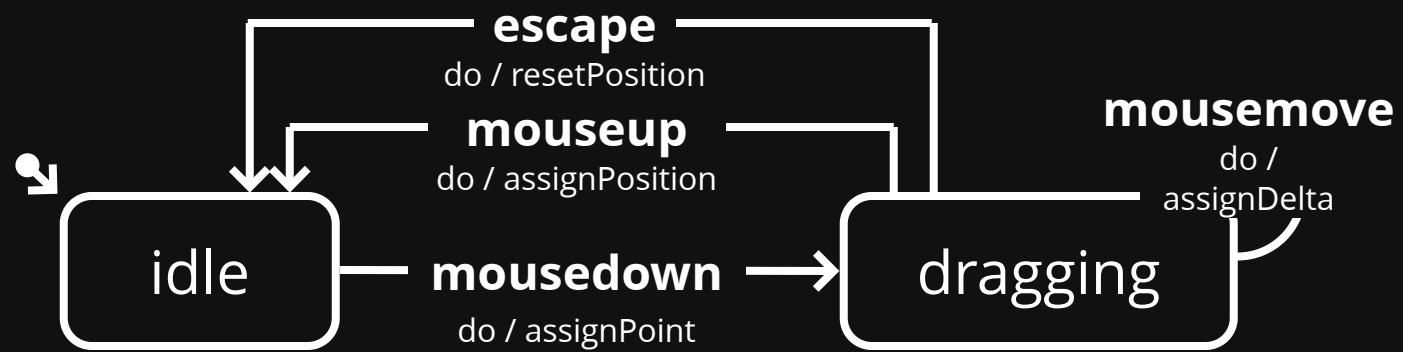
Exercise 05



Exercise 05

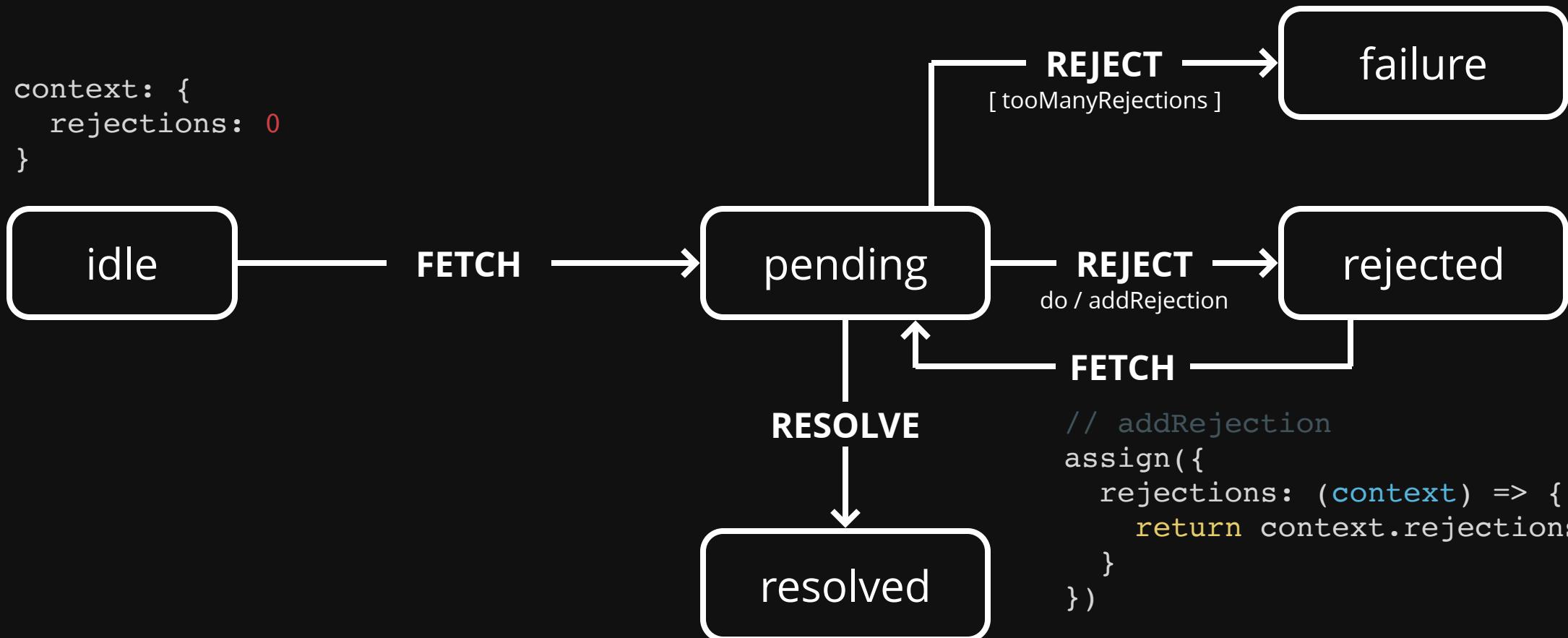


Exercise 05



Guarded transitions

```
context: {  
  rejections: 0  
}
```



```
const tooManyRejections = (context, event) =>  
  return context.rejections >= 5;  
}  
  
// addRejection  
assign({  
  rejections: (context) => {  
    return context.rejections + 1;  
  }  
})
```

Guarded transitions

```
1 import { createMachine } from 'xstate';
2
3 const authMachine = createMachine({
4   initial: 'idle',
5   context: {
6     rejections: 0
7   },
8   states: {
9     idle: {
10       on: {
11         FETCH: 'pending'
12       }
13     },
14     pending: {
15       on: {
16         REJECT: [
17           {
18             target: 'failure',
19             cond: (context) => {
20               return context.rejections >= 5
21             }
22           },
23           {
24             target: 'rejected',
25             actions: assign({
26               retries: (context) => context.retries + 1
27             })
28           }
29         ],
30       }
31     }
32   }
33 })
```

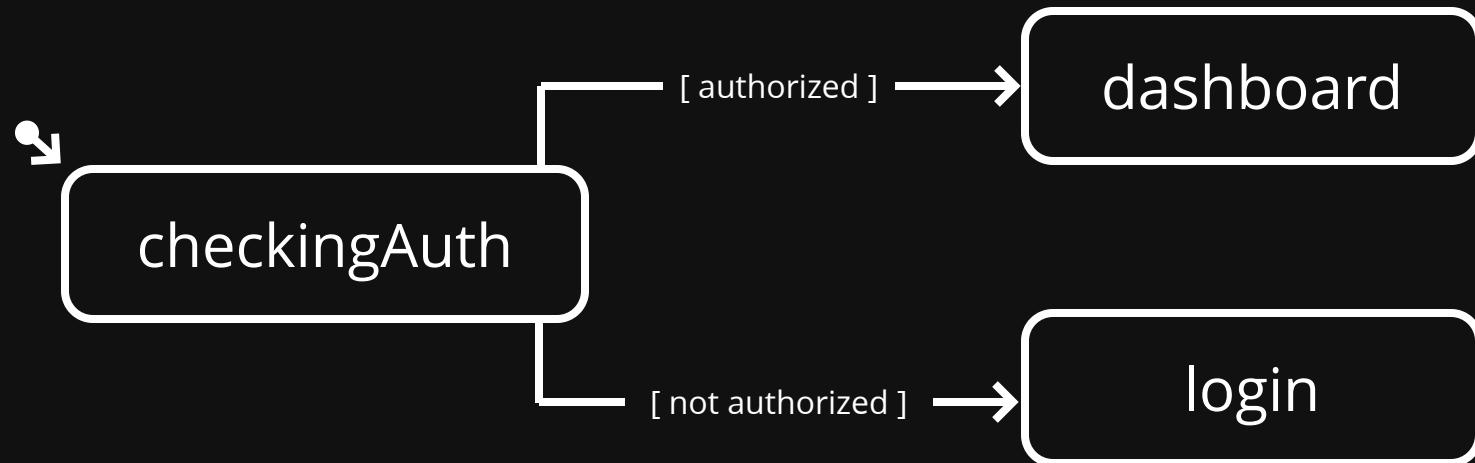
Guarded transitions

```
1 import { createMachine } from 'xstate';
2
3 const tooManyRejections = (context) => {
4   return context.rejections >= 5
5 };
6
7 const addRejection = assign({
8   retries: (context) => context.retries + 1
9 });
10
11 const authMachine = createMachine({
12   initial: 'idle',
13   context: {
14     rejections: 0
15   },
16   states: {
17     idle: {
18       on: {
19         FETCH: 'pending'
20       }
21     },
22     pending: {
23       on: {
24         REJECT: [
25           {
26             target: 'failure',
27             cond: tooManyRejections
28           },
29           {

```

Exercise 06

Transient transitions



Transient transitions

```
1 import { createMachine } from 'xstate';
2
3 const isAuthorized = (context, event) => {
4   return !!context.user;
5 }
6
7 const appMachine = createMachine({
8   initial: 'checkingAuth',
9   context: {
10     user: null
11   },
12   states: {
13     checkingAuth: {
14       on: {
15         '' : [
16           {
17             target: 'dashboard',
18             cond: isAuthorized
19           },
20           { target: 'login' }
21         ]
22       }
23     },
24     login: {},
25     dashboard: {}
26   }
27 });
```

Transient transitions

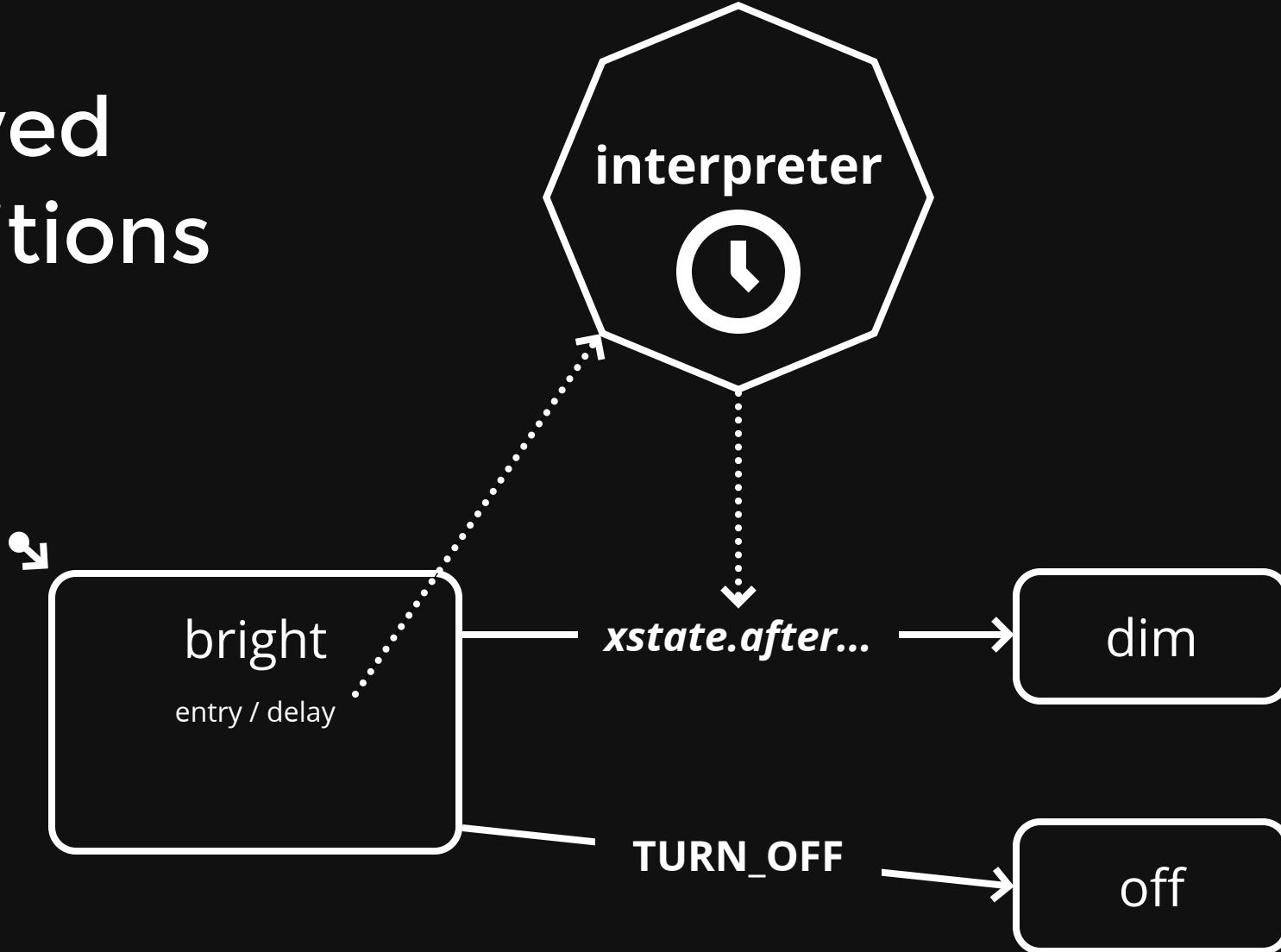
```
1 import { createMachine } from 'xstate';
2
3 const isAuthorized = (context, event) => {
4   return !!context.user;
5 }
6
7 const appMachine = createMachine({
8   initial: 'checkingAuth',
9   context: {
10     user: null
11   },
12   states: {
13     checkingAuth: {
14       on: {
15         '' : [
16           {
17             target: 'dashboard',
18             cond: 'isAuthorized'
19           },
20           { target: 'login' }
21         ]
22       }
23     },
24     login: {},
25     dashboard: {}
26   }
27 }, {
28   guards: {
29     isAuthorized
```

Exercise 07

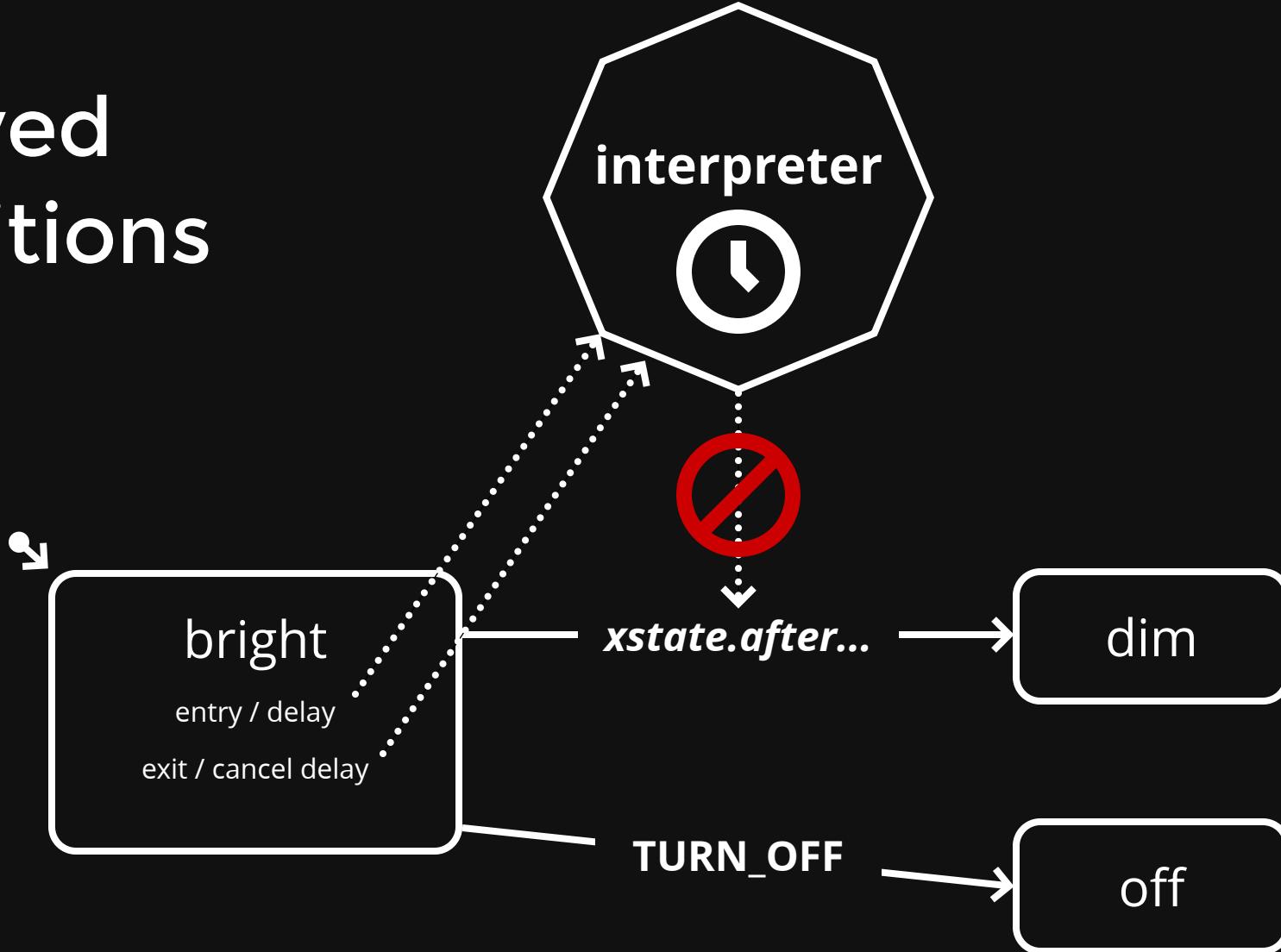
Delayed transitions



Delayed transitions

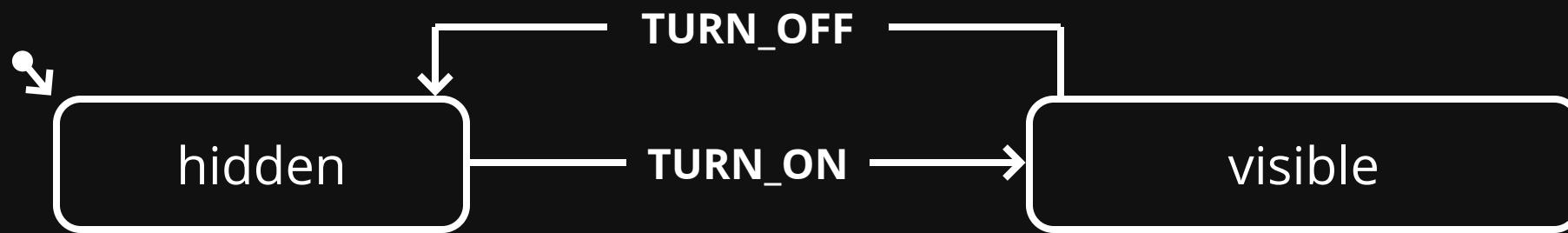


Delayed transitions

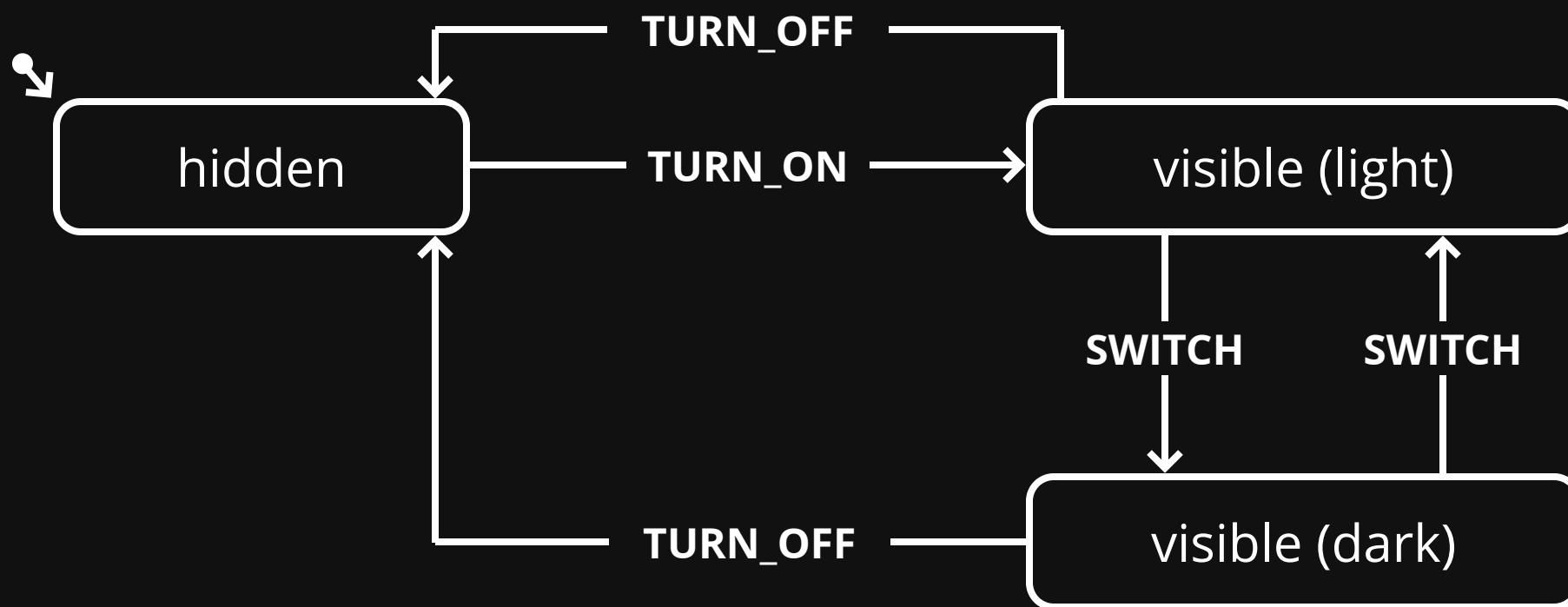


Exercise 08

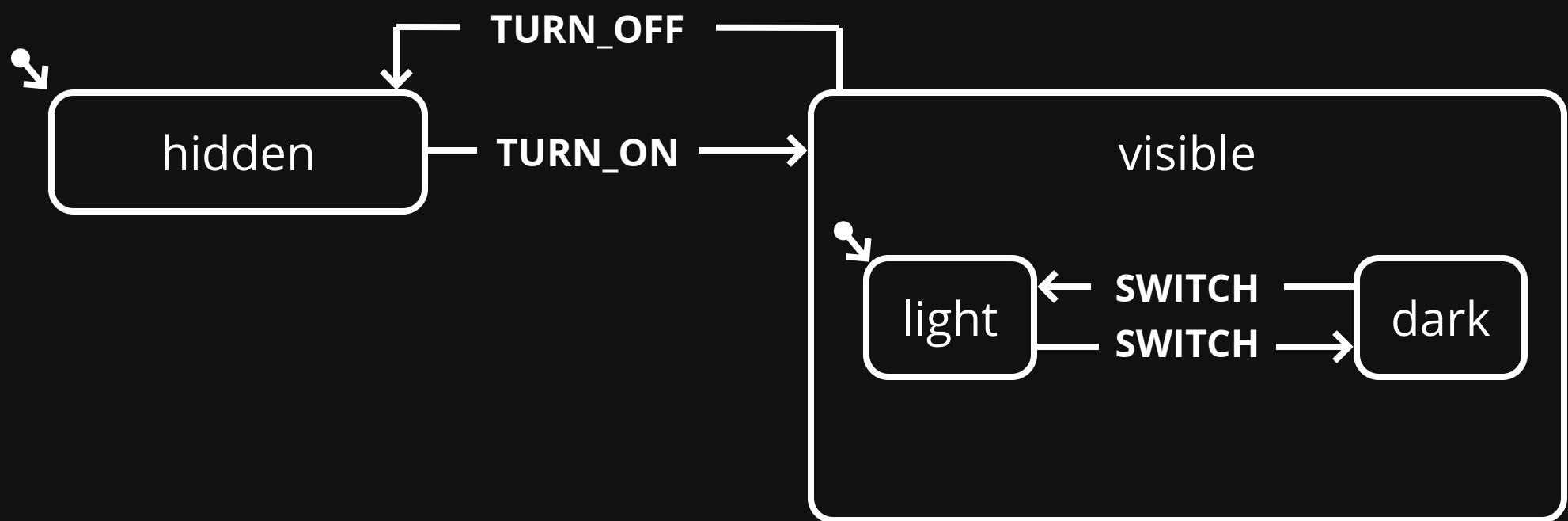
Hierarchical states



Hierarchical states

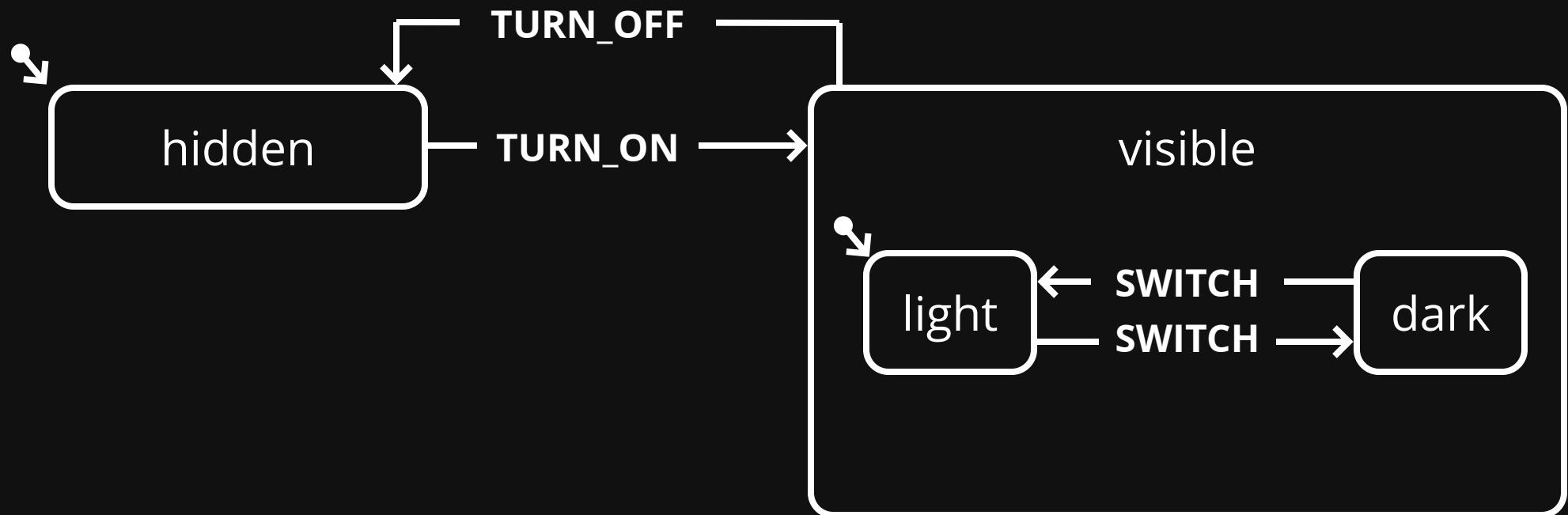


Hierarchical states

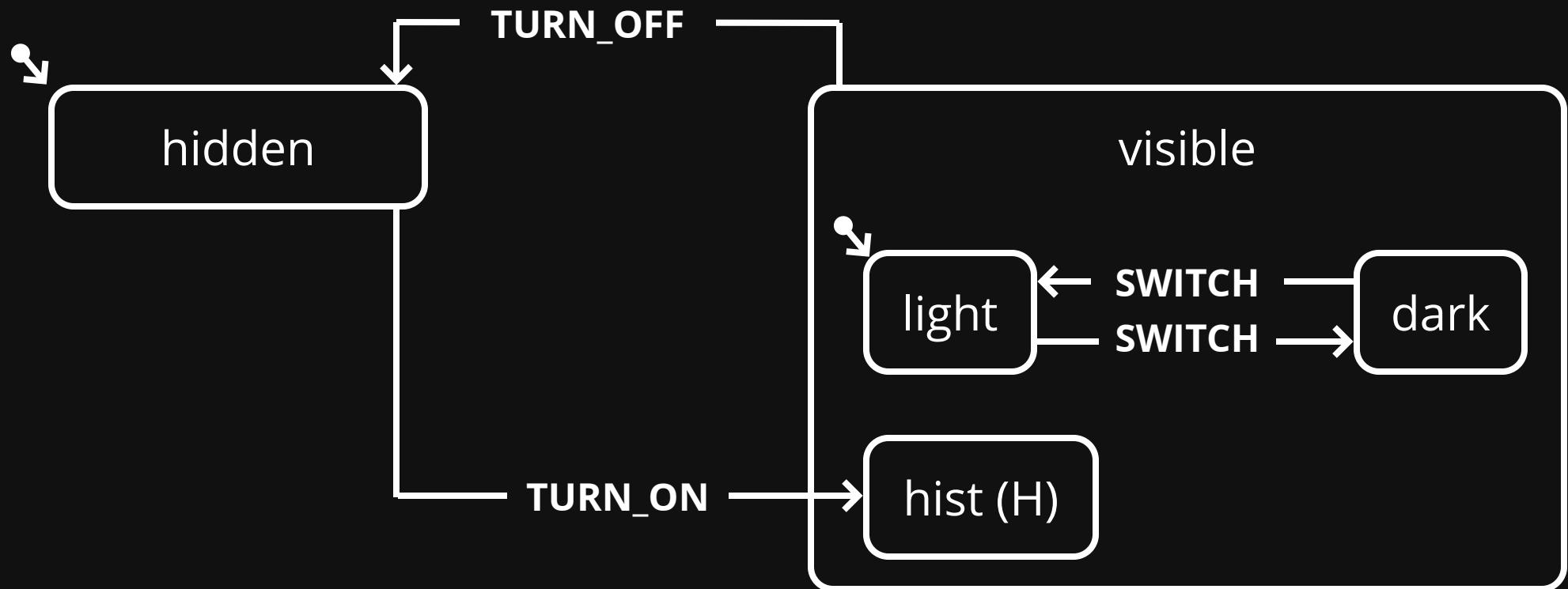


Exercise 09

History states

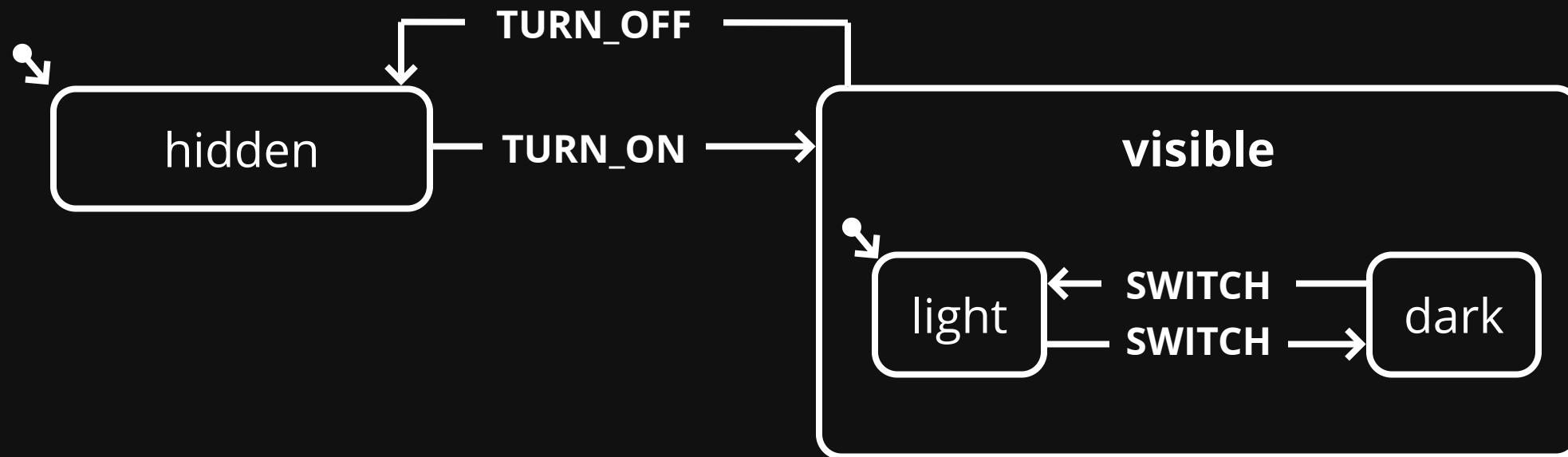


History states

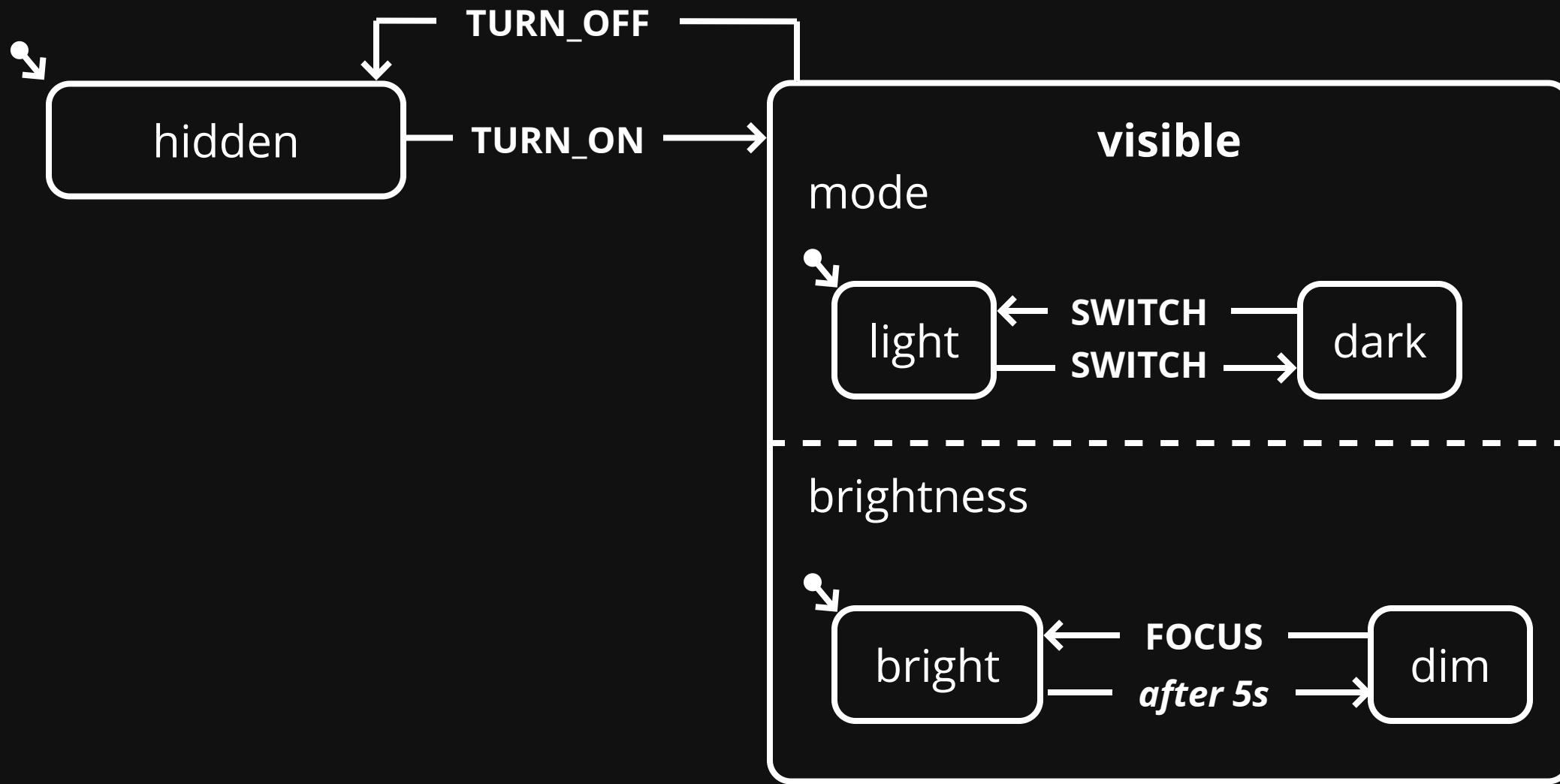


Exercise 10

Parallel states



Parallel states

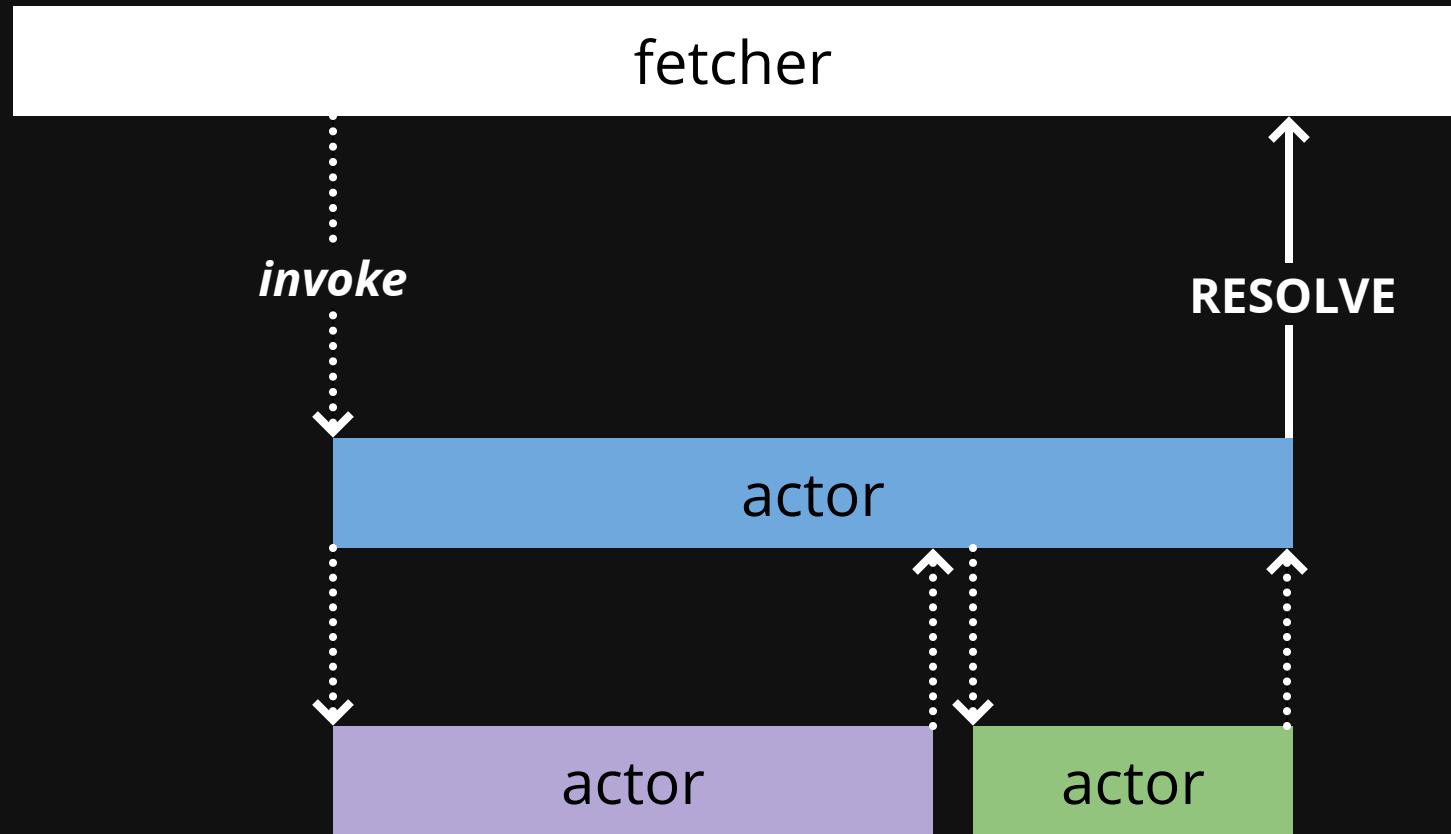


Exercise 11

The actor model

An actor can:

- send a message
- spawn new actor(s)
- change its behavior



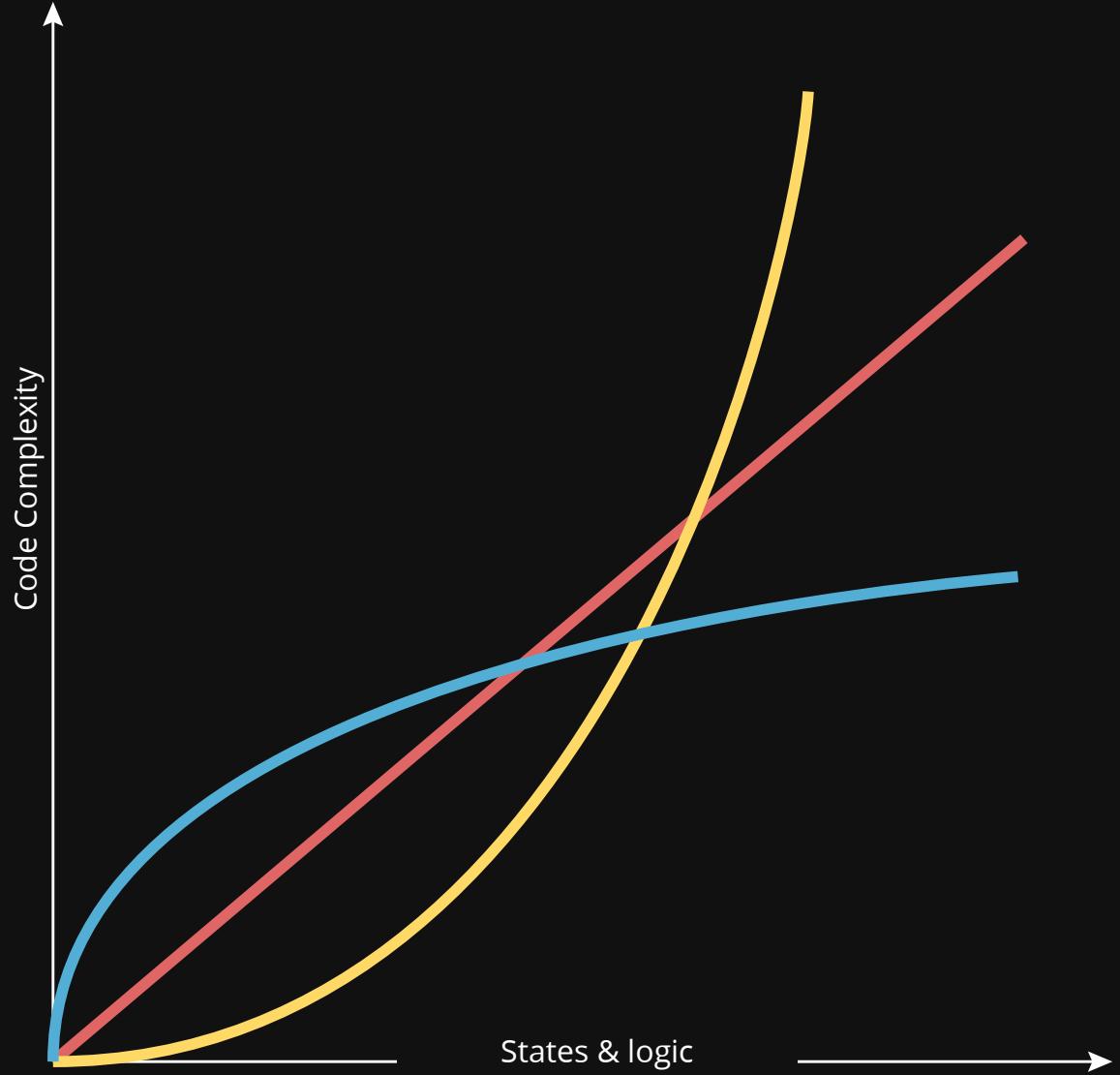
Exercise 12

Further topics

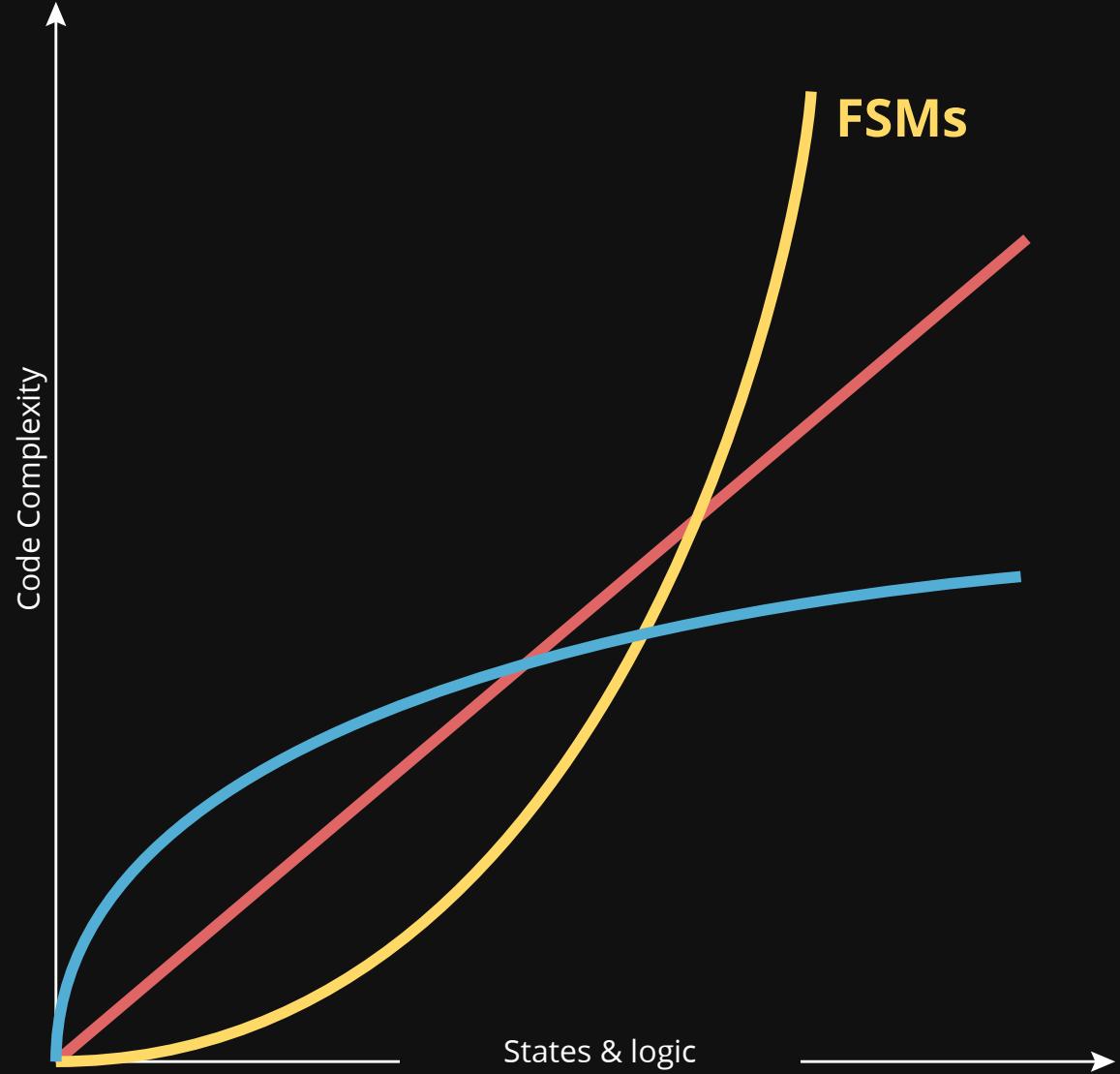
- XState + React with **@xstate/react**
- XState + Vue with **@xstate/vue**
- *XState + Angular (coming soon)*
- *XState + Svelte (coming soon)*
- Model-based-testing with **@xstate/test**
- Graph generation with **@xstate/graph**

 xstate.js.org/docs

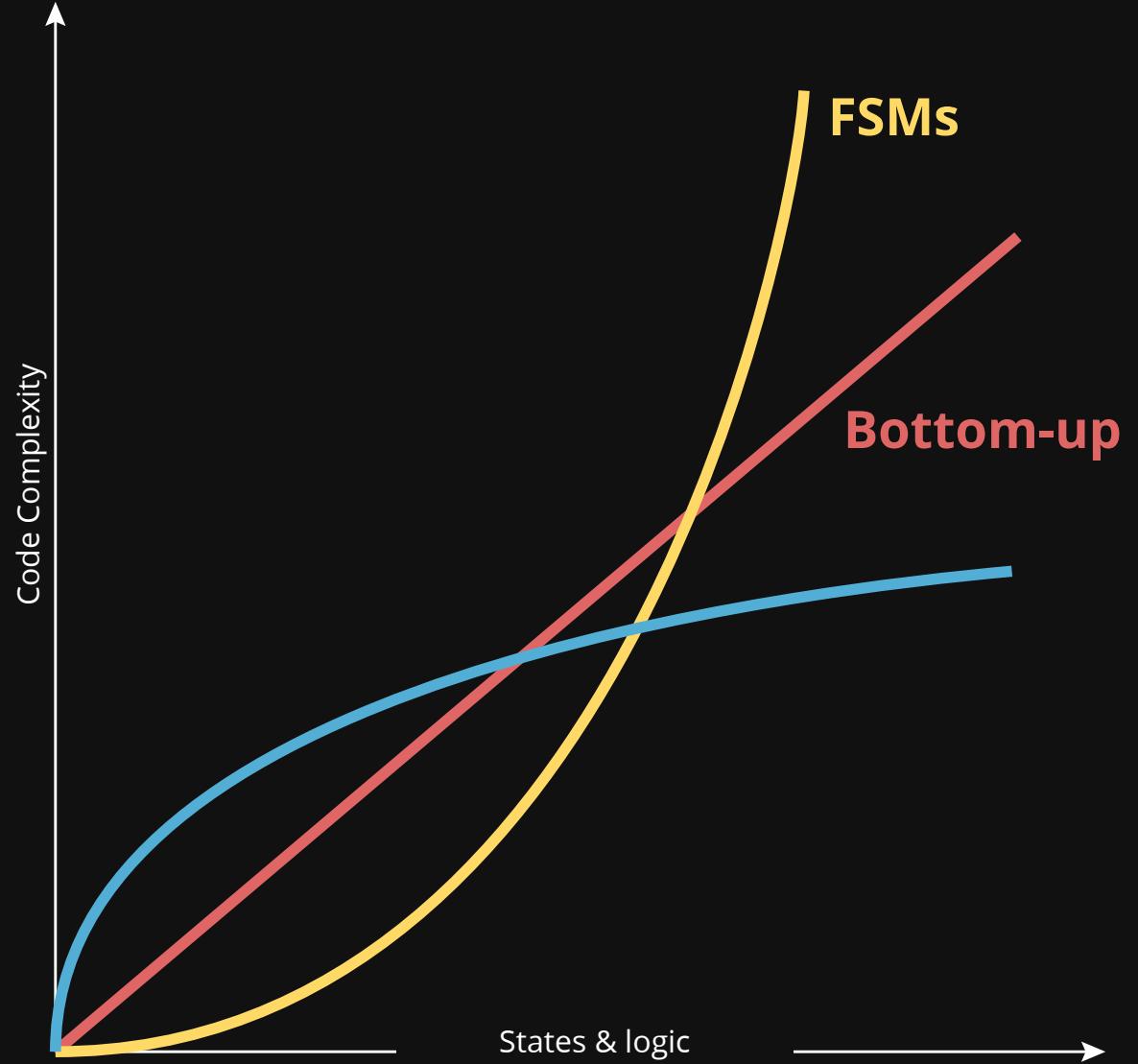
Complexity trade-offs



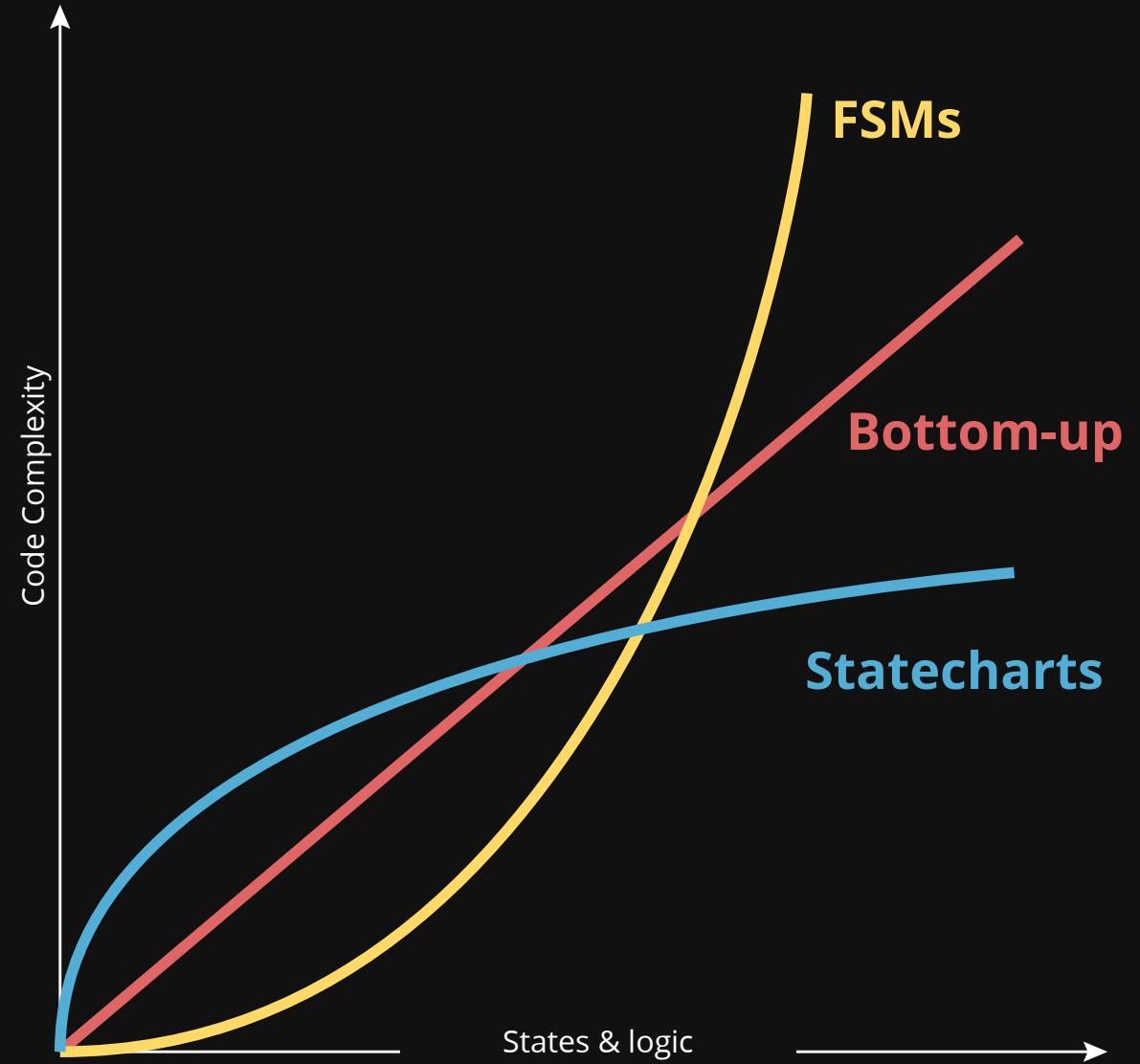
Complexity trade-offs



Complexity trade-offs



Complexity trade-offs



The world of statecharts

statecharts.github.io

Spectrum community

spectrum.chat/statecharts

XState Documentation

xstate.js.org/docs

