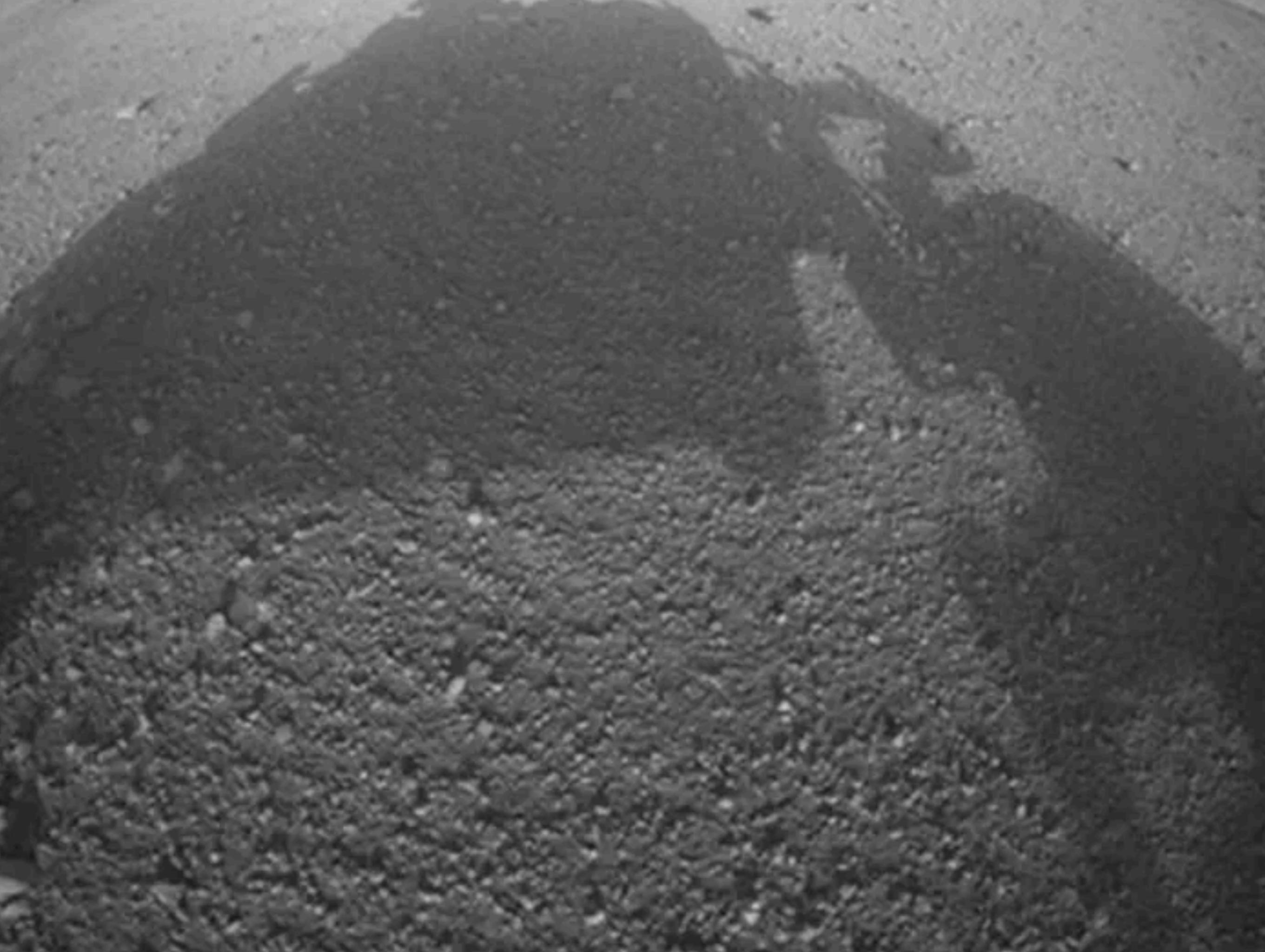


Mastering views with Finite State Machines



Rubén Sospedra

@sospedra_r

Javascript hacker



SENIOR SOFTWARE ENGINEER AT mytaxi





“develop
user interfaces
is not easy”

David Khourshid, 2017

4GIFs.com

Please insert
**BANK OF
MEXICO**
Card and Remove
Card.
Por favor inserte y retire su
tarjeta.



 MCR

Copying...



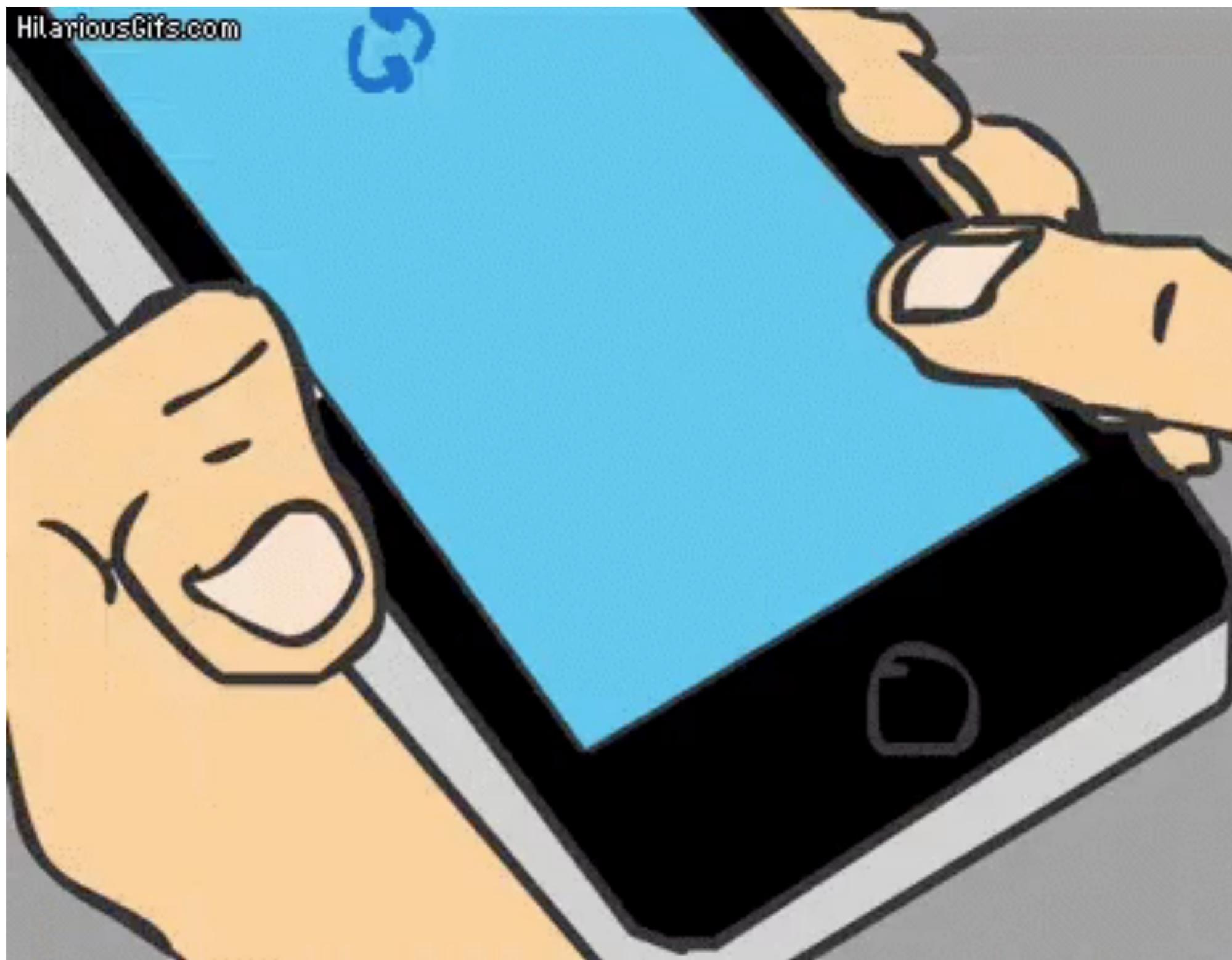
Big File

From 'C:\\' to 'D:\\'



Cancel

HilariousGifs.com





defensive programming

```
  . . . fetchUser () {-
    . . . . . fetch('https://api.domain.com/user').then((response) => {
    . . . . . . response.json().then((payload) => {
    . . . . . . . this.setState({ user: payload.user })
    . . . . . . })
    . . . . . })
    . . . . }
    . . . }

  . . . render () {-
    . . . . <Main>
    . . . . . . <Text>{this.state.user.name}</Text>
    . . . . </Main>
    . . . }
```

```
  fetchUser () {  
    this.setState({ isLoading: true })  
    fetch('https://api.domain.com/user').then((response) => {  
      response.json().then((payload) => {  
        this.setState({  
          user: payload.user,  
          isLoading: false  
        })  
      })  
    })  
  }  
  
  render () {  
    <Main>  
      {this.state.isLoading?  
        ? <Loading />  
        : <Text>{this.state.user.name}</Text>}  
    </Main>  
  }  
}
```

```
fetchUser () {  
  this.setState({ isLoading: true })  
  fetch('https://api.domain.com/user').then((response) => {  
    response.json().then((payload) => {  
      this.setState({  
        user: payload.user,  
        isLoading: false  
      })  
    }).catch(() => this.setState({ isError: true, isLoading: false }))  
  }).catch(() => this.setState({ isError: true, isLoading: false }))  
}  
  
render () {  
  <Main>  
    {this.state.isError && <Error />}  
    {!this.state.isError && this.state.isLoading  
     ? <Loading />  
     : <Text>{this.state.user.name}</Text>}  
  </Main>  
}
```

```
  .fetchUser () {-
    .  .  .this.setState({ isLoading: true })-
    .  .  .fetch('https://api.domain.com/user').then((response) => {-
    .  .  .  .if (this.state.hasCancel) return-
    .  .  .  .response.json().then((payload) => {-
    .  .  .  .  .this.setState({-
    .  .  .  .  .  .user: payload.user,-
    .  .  .  .  .  .isLoading: false-
    .  .  .  .  .})-
    .  .  .}).catch(() => {-
    .  .  .  .if (this.state.hasCancel) return-
    .  .  .  .this.setState({ isError: true, isLoading: false })-
    .  .  .}).catch(() => {-
    .  .  .  .if (this.state.hasCancel) return-
    .  .  .  .this.setState({ isError: true, isLoading: false })-
    .  .  .})-
    .  .}-
    .  .render () {-
    .  .  <Main>-
    .  .  .{this.state.isError && <Error />}-
    .  .  .{!this.state.isError && this.state.isLoading-
    .  .  .  ? <Loading />-
    .  .  .  : <Text>{this.state.user.name}</Text>}-
    .  .  .<button onClick={() => this.setState({ hasCancel: true })}>-
    .  .  .  .Cancel-
    .  .  .  </button>-
    .  .}</Main>-
    .  .}-
  .}
```

```
this.setState({ isLoading: true })  
fetch('https://api.domain.com/user').then((response) => {  
  if (this.state.hasCancel) return  
  response.json().then((payload) => {  
    this.setState({  
      user: payload.user,  
      isLoading: false  
    })  
  }).catch(() => {  
    if (this.state.hasCancel) return  
    this.setState({ isError: true, isLoading: false })  
  })  
}).catch(() => {  
  if (this.state.hasCancel) return  
  this.setState({ isError: true, isLoading: false })  
})  
}  
  
render () {  
  <Main>  
  {this.state.isError && <Error />}  
}
```

```
  this.setState({ isLoading: true });

  fetch('https://api.domain.com')
    .then(response => response.json())
    .then(payload =>
      this.setState({
        user: payload.user,
        isLoading: false
      })
    )
    .catch(() => {
      // ...
    });

```

```
setState({ isLoading: true })  
  .then(() =>  
    axios.get('https://api.domain.com/cart')  
      .then(response =>  
        const { items } = response.data  
        setState({  
          items,  
          isLoading: false  
        })  
      )  
      .catch(error =>  
        console.error(error)  
        setState({  
          error,  
          isLoading: false  
        })  
      )  
  )  
  .catch(error =>  
    console.error(error)  
    setState({  
      error  
    })  
  )  
}  
;
```

```
setState({ isLoading: true });

('https://api.domain.com/api/v1/categories')
  .then(response => response.json())
  .then(categories => {
    const categoryList = categories.map(category => (
      

A brown poop emoji with a smiling face, positioned over the word "response" in the code.



const categoryList = categories.map(category =>

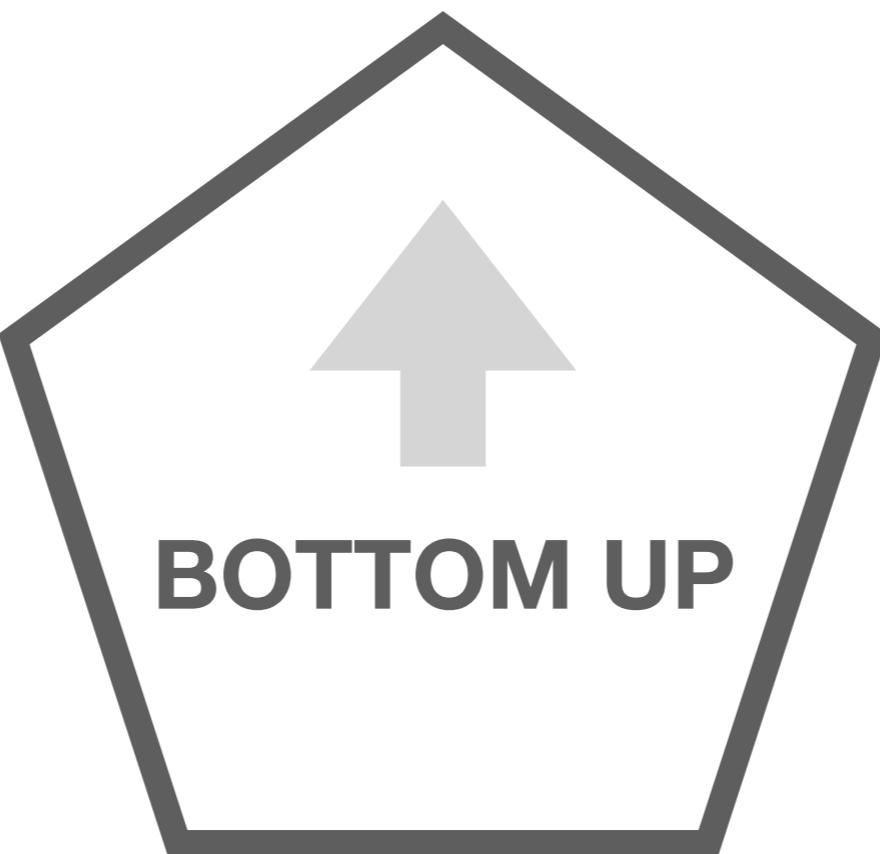

    ))
    .then(categoryList => {
      this.setState({
        categories: categoryList,
        isLoading: false
      });
    })
  })

```

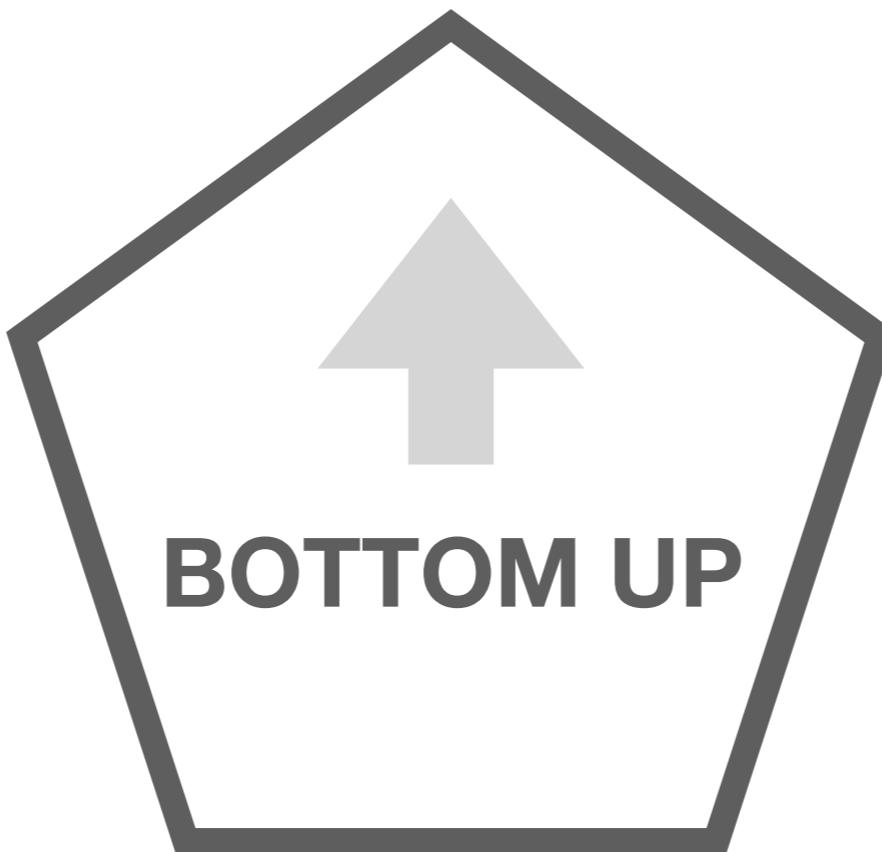


but why?

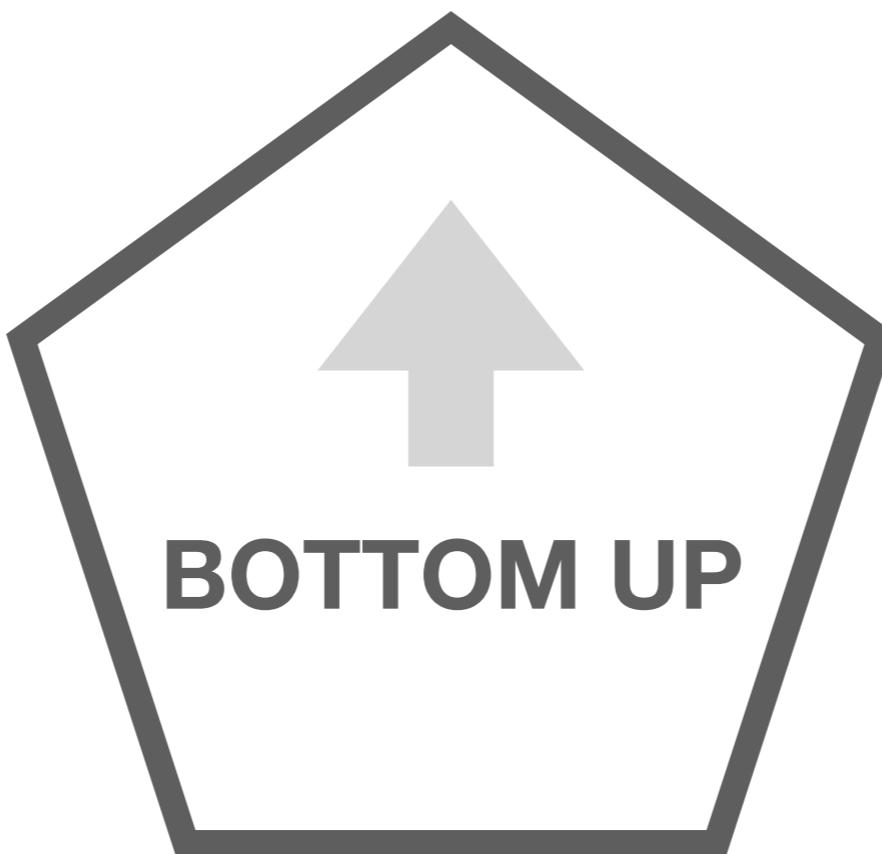
Understand



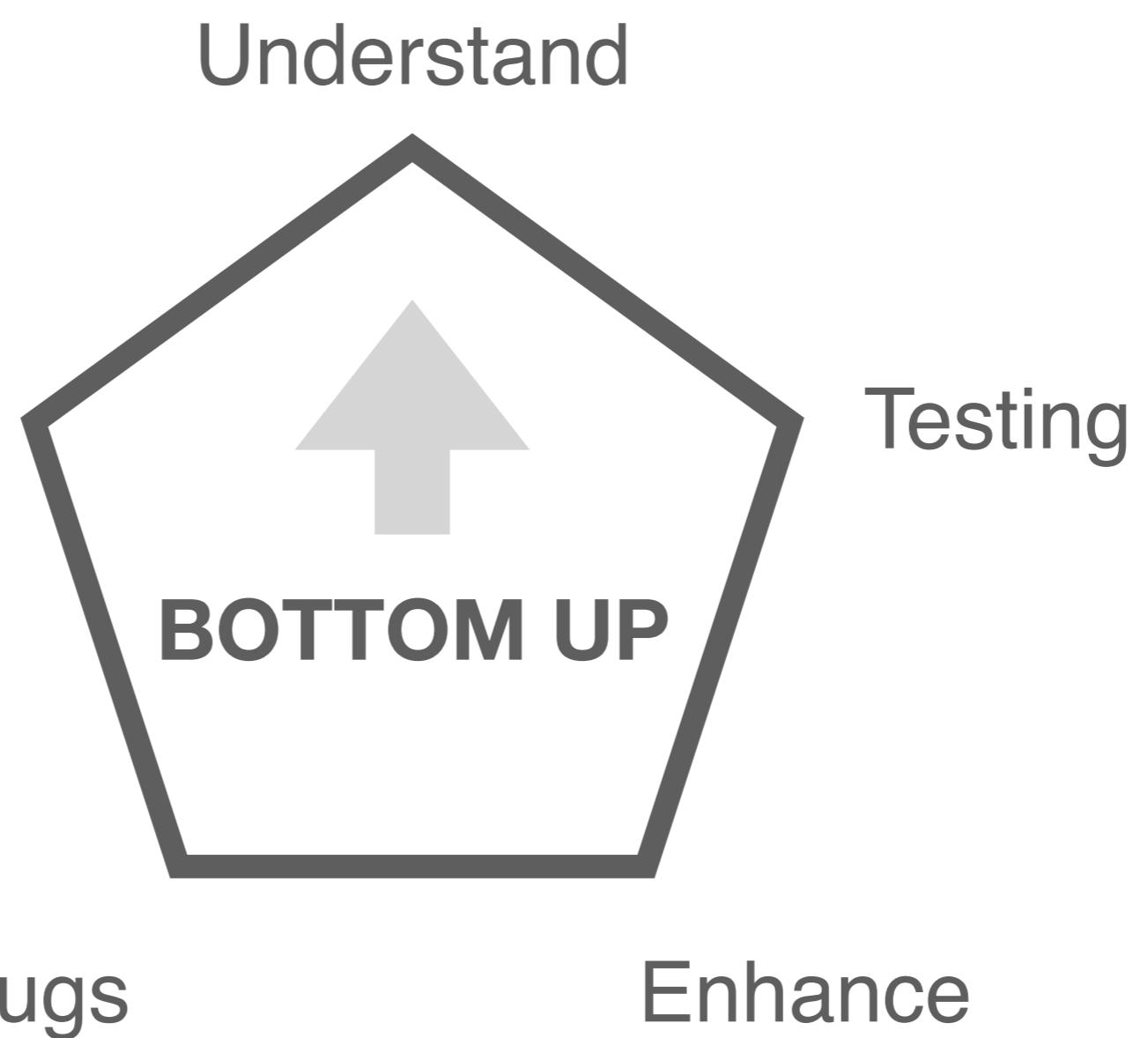
Understand

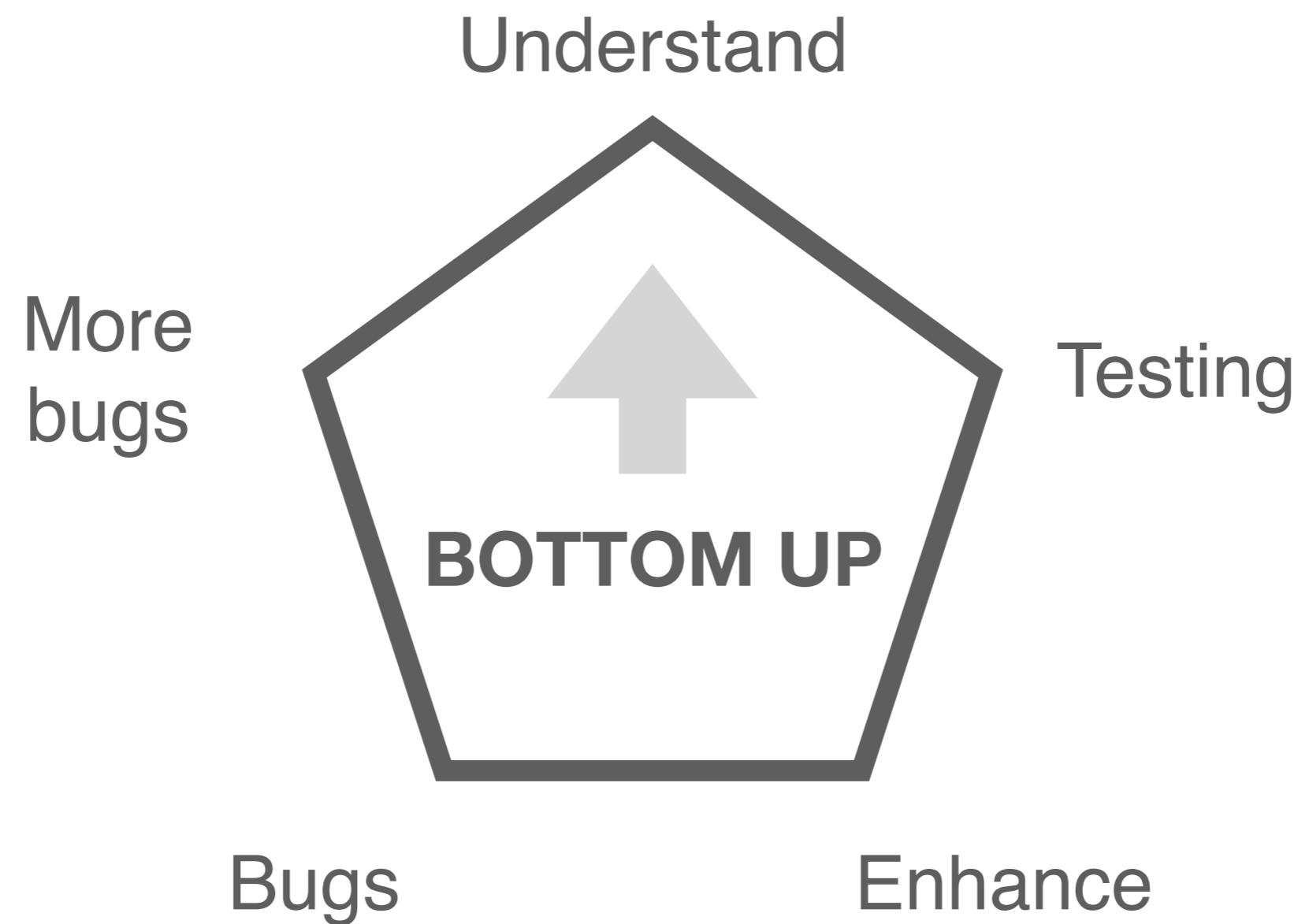


Understand



Enhance

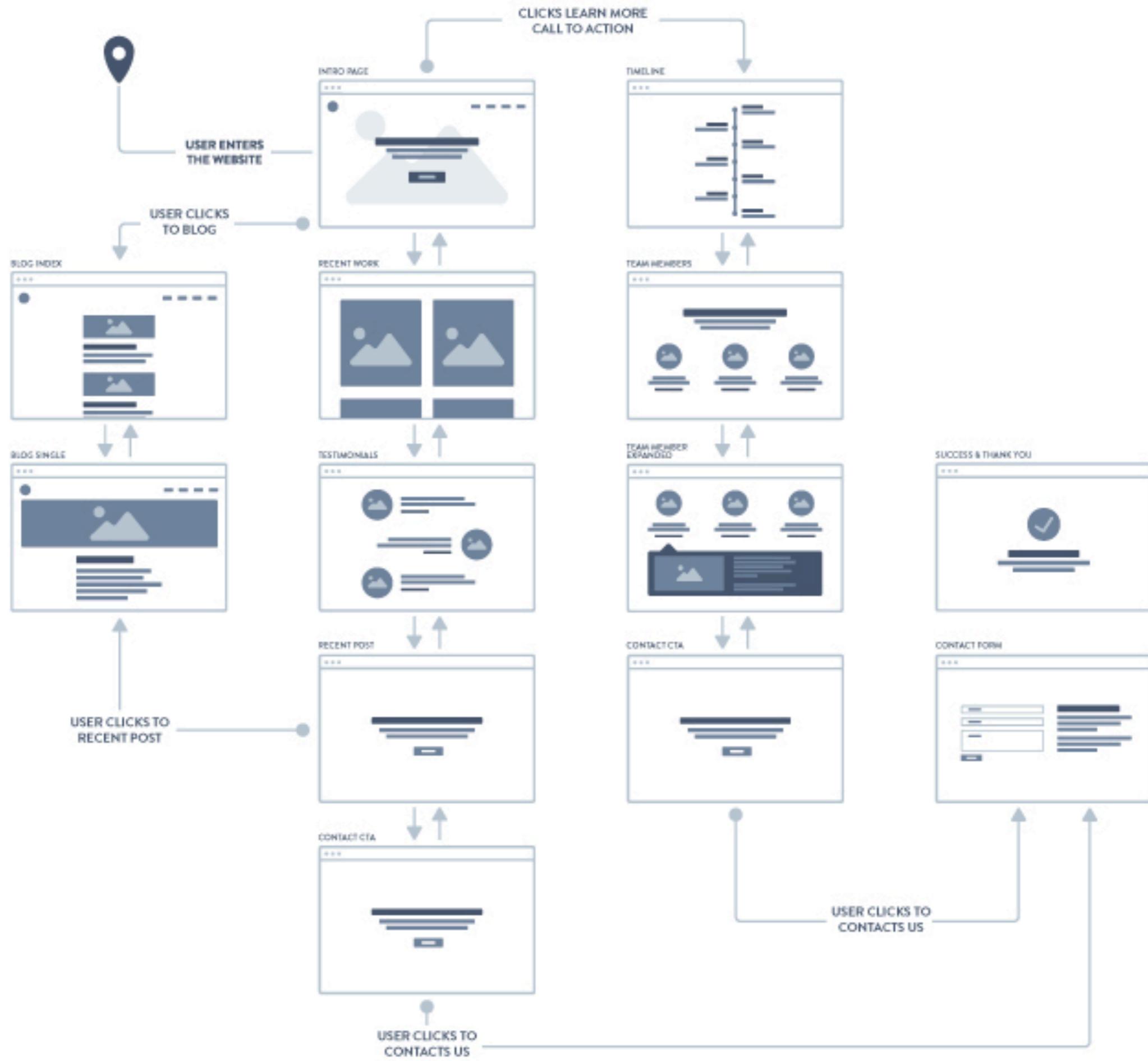








top-down



finite state machines



In accordance with the general classification, the following formal definitions are found:

- A *deterministic finite state machine* or *acceptor deterministic finite state machine* is a [quintuple](#) $(\Sigma, S, s_0, \delta, F)$, 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: $\delta : S \times \Sigma \rightarrow S$
 - F is the set of final states, a (possibly empty) subset of S .





Σ is the input alphabet (finite non-empty set)



Σ is the input alphabet (finite non-empty set)

```
const alphabet = {  
  - LOCK: 'LOCK',  
  - UNLOCK: 'UNLOCK'  
}
```



S is a finite, non-empty set of states



S is a finite, non-empty set of states

```
const states = {  
    - · · LOCK: { ... },  
    - · · UNLOCK: { ... }  
}
```



s_0 is an initial state, an element of S



s_0 is an initial state, an element of S

```
const initial = states[FIRST] -
```



δ is the state-transition function: $\delta : S \times \Sigma \rightarrow S$



δ is the state-transition function: $\delta : S \times \Sigma \rightarrow S$

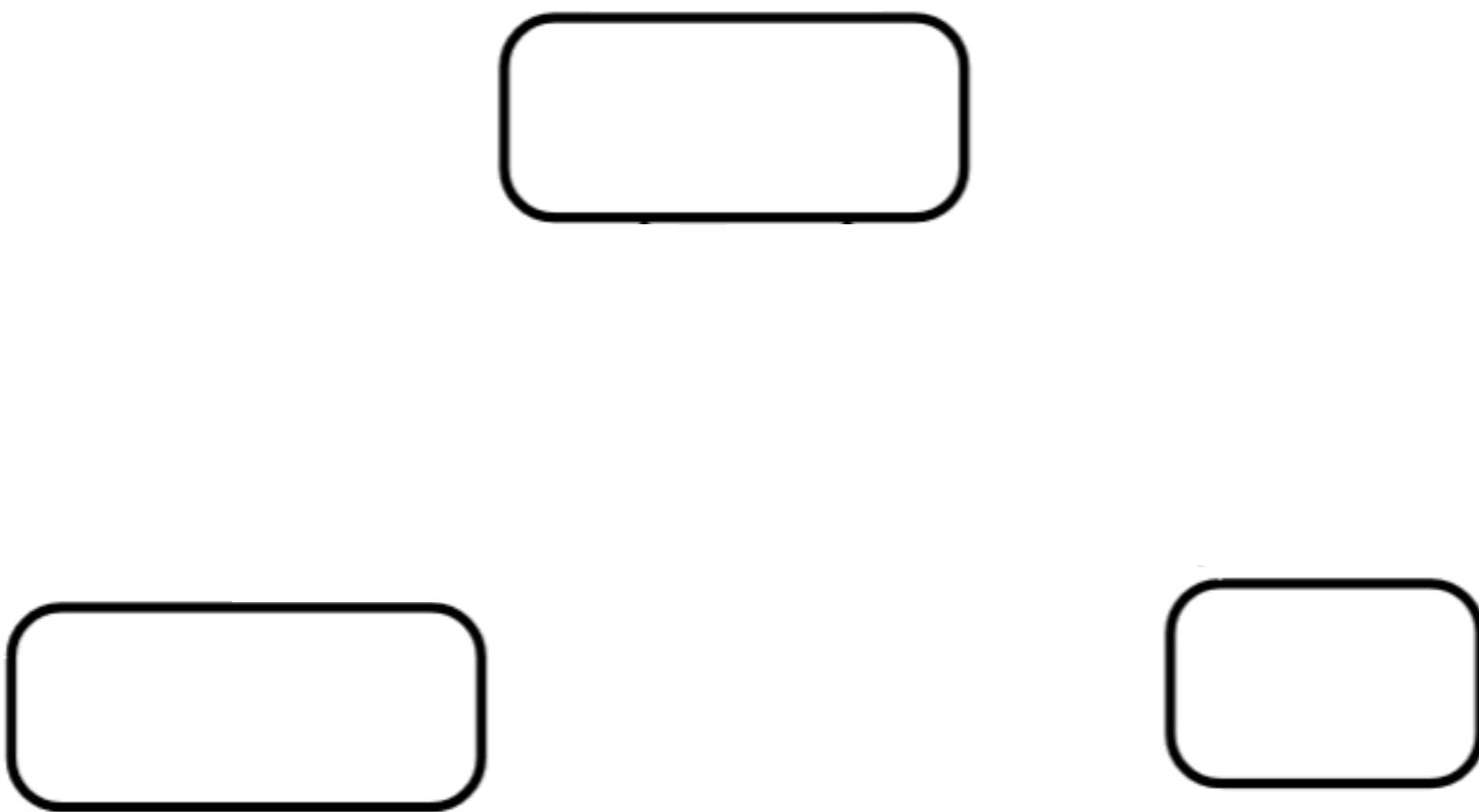
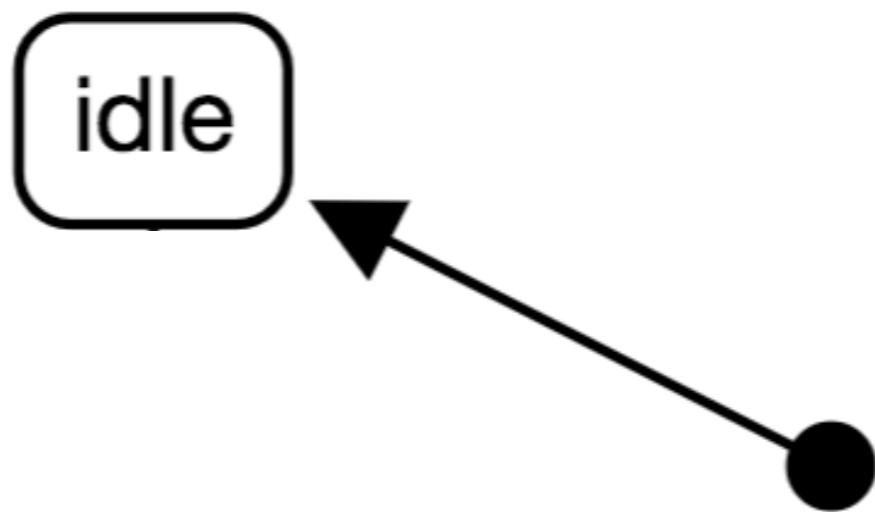
```
const transition = (states, alpha) => {
  return states[alpha]
}
```

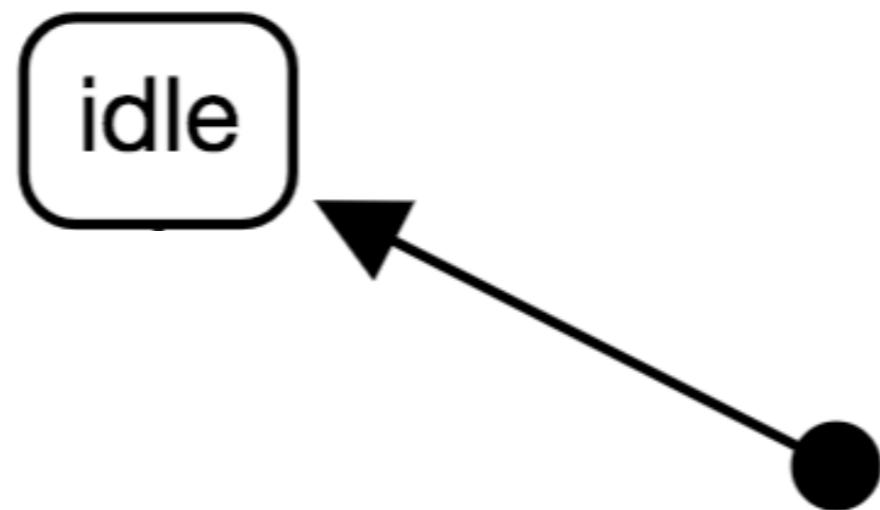
```
const machine = {  
  LOCK: {  
    next: UNLOCK  
  },  
  UNLOCK: {  
    next: LOCK  
  }  
}  
  
const initial = LOCK  
  
const transition = (current) => {  
  return machine[current].next  
}
```



anatomy



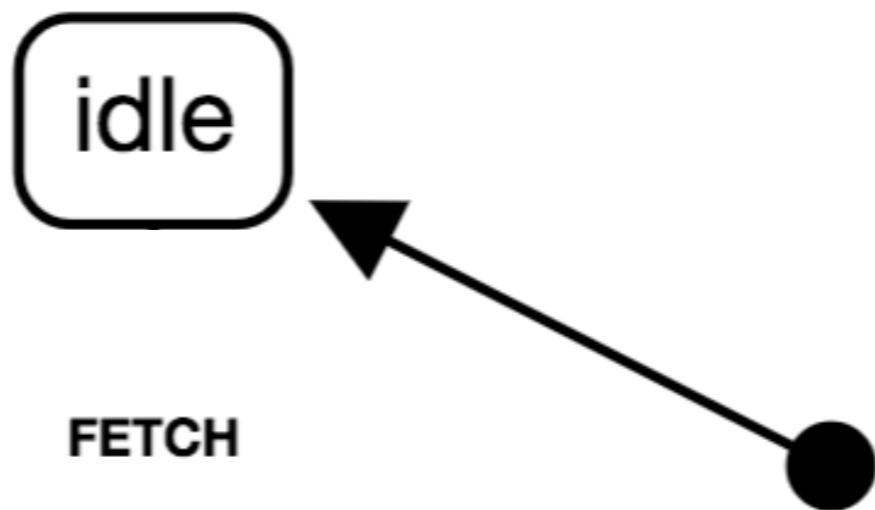




fetching

success

error



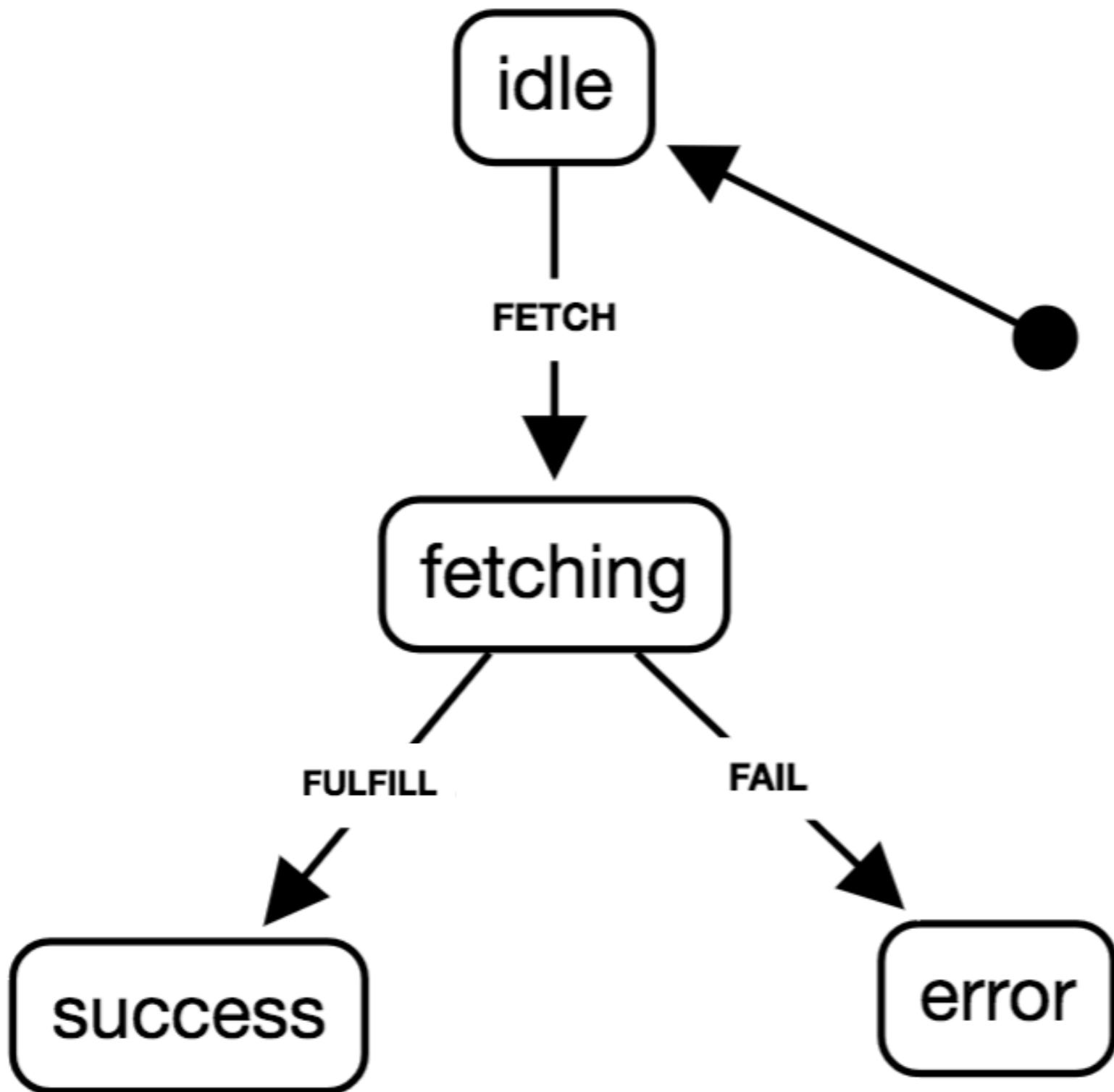
fetching

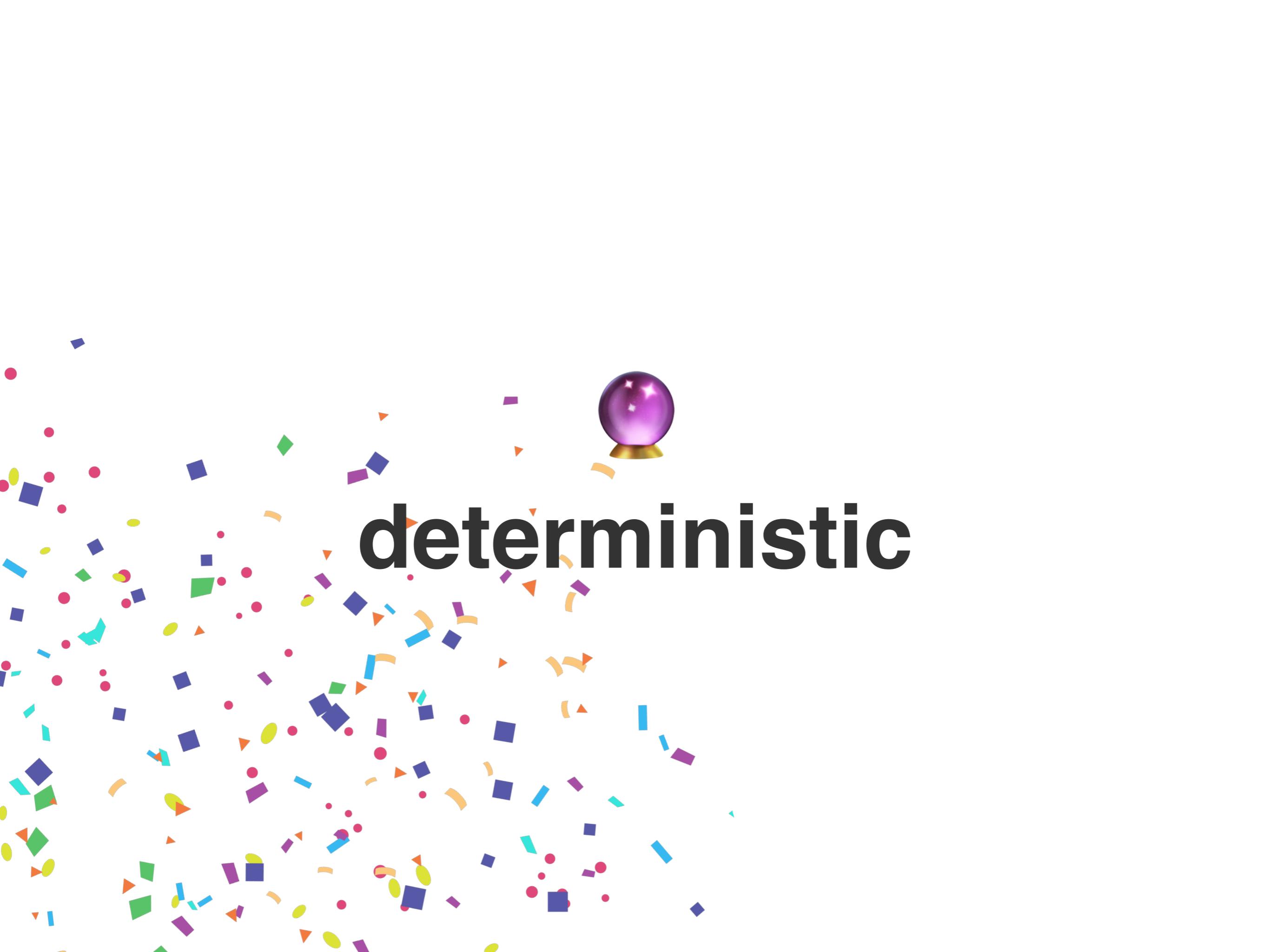
FULFILL

FAIL

success

error





deterministic



automata





END

finite



“a pure system that
can only be in one of
a set of finite states”

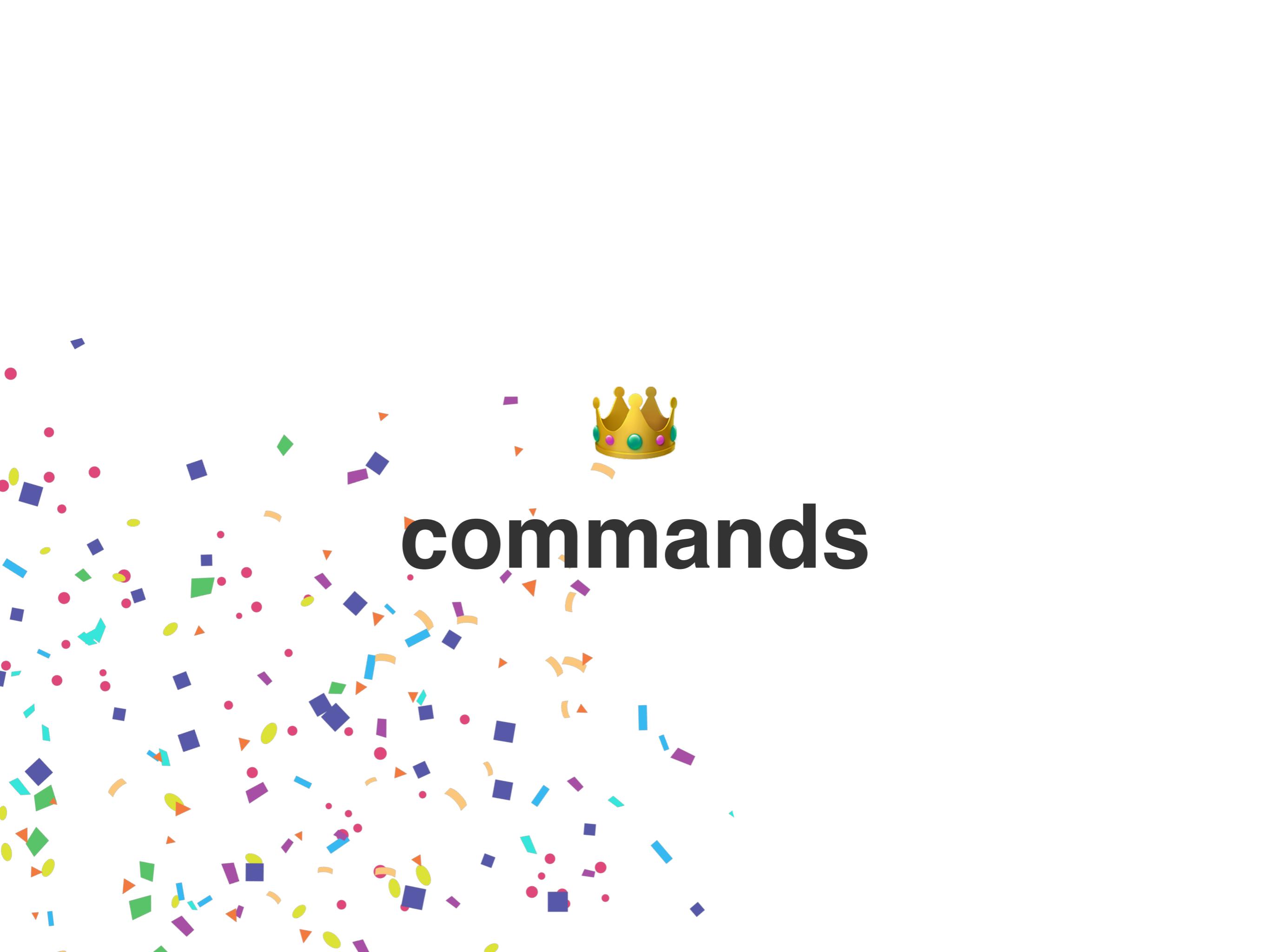
Me, now





demo





commands

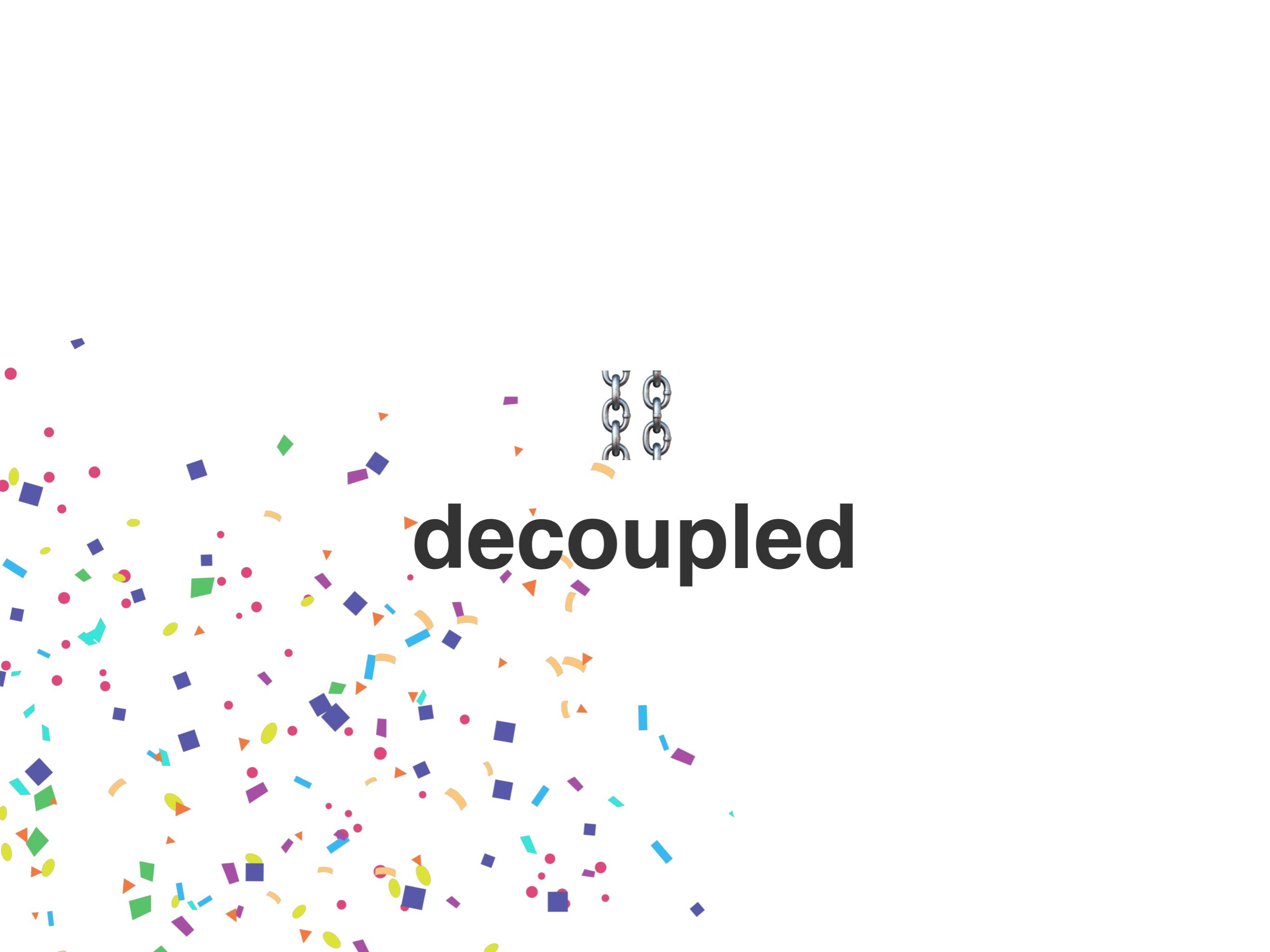
WOAX



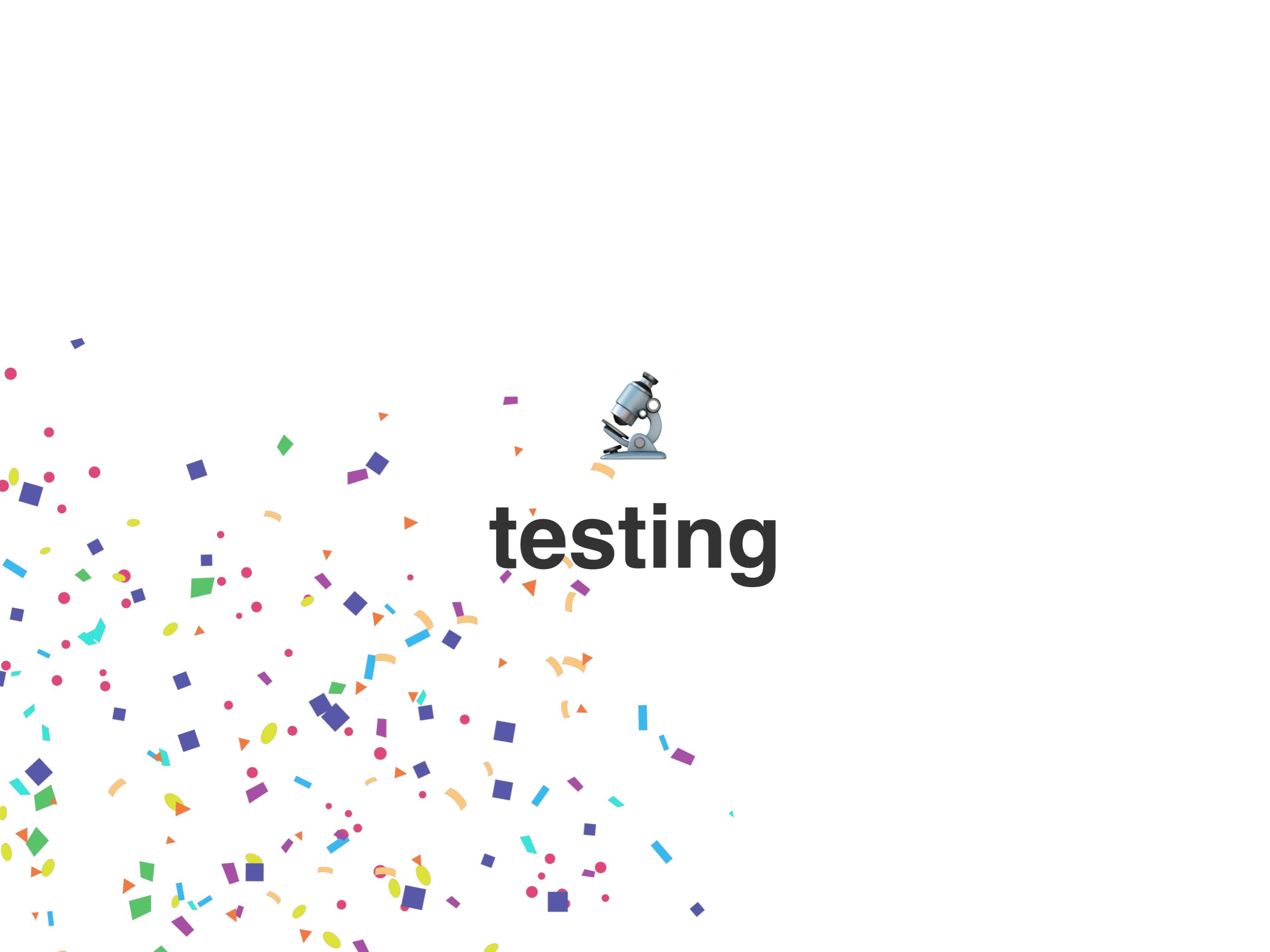


**U CANT HAVE RACE
CONDITIONS**

IF U DONT HAVE CONDITIONS



decoupled



testing





[adult swim]



statecharts



“a visual formalism for complex systems”

David Harel, 1987

clustering

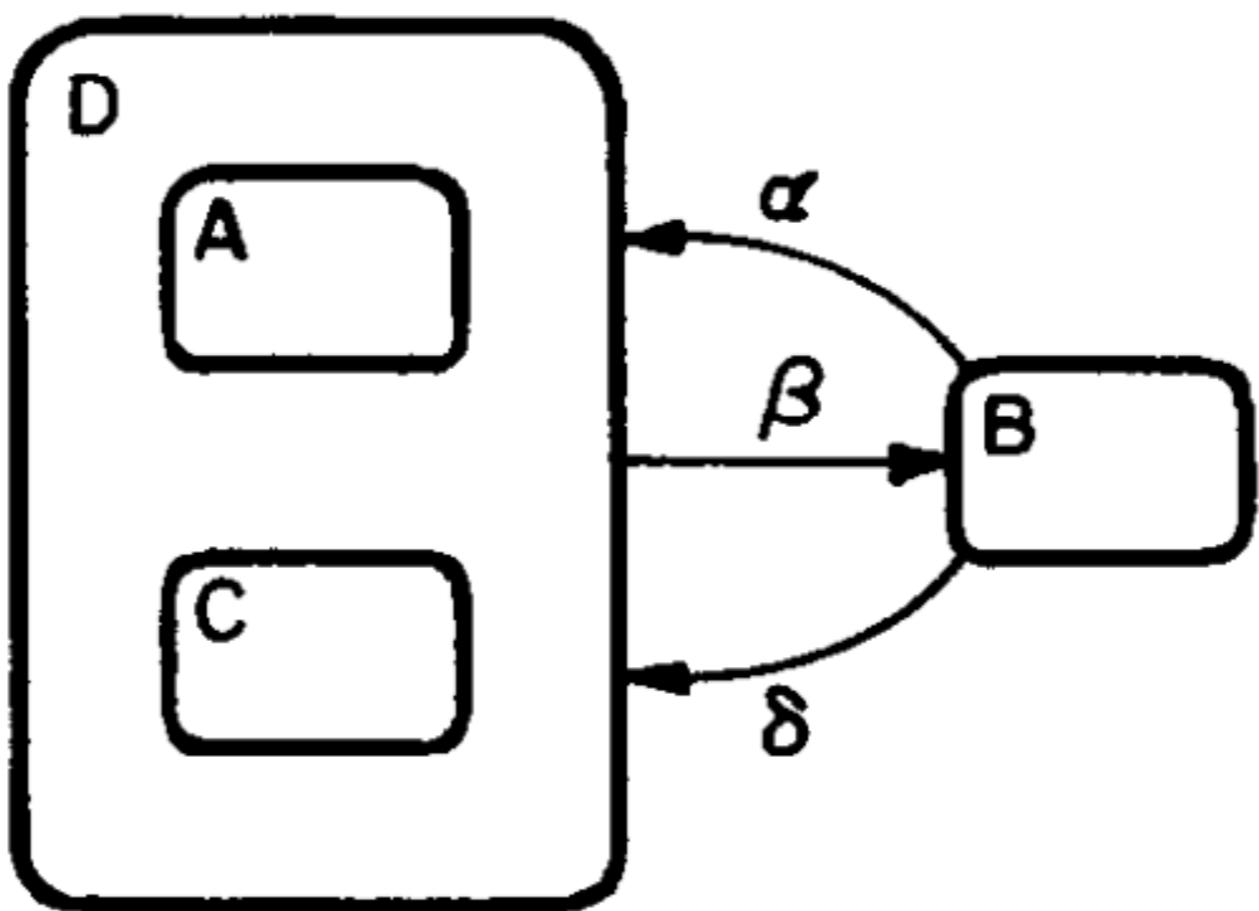


Fig. 4.

orthogonality

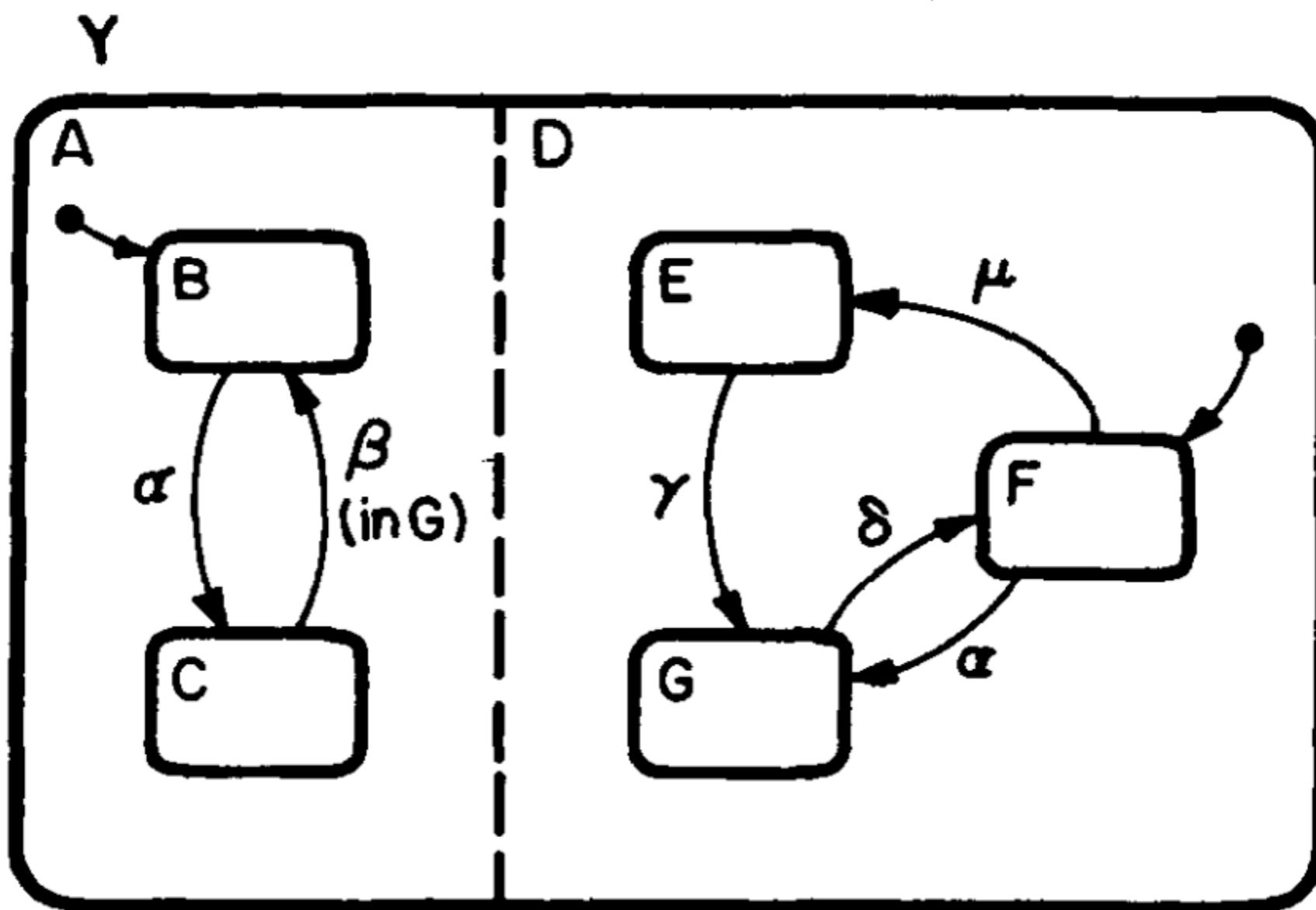


Fig. 19.

guards

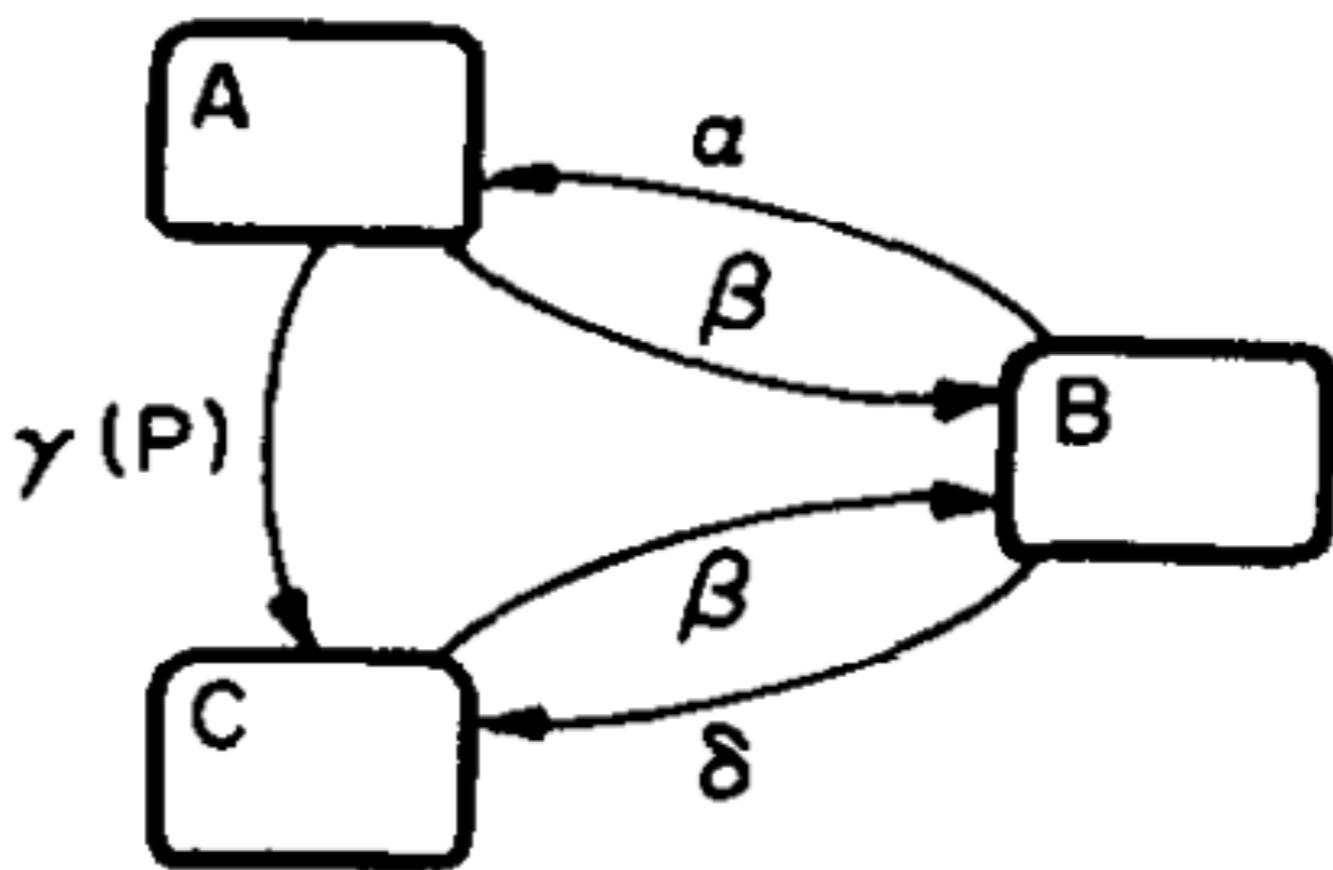


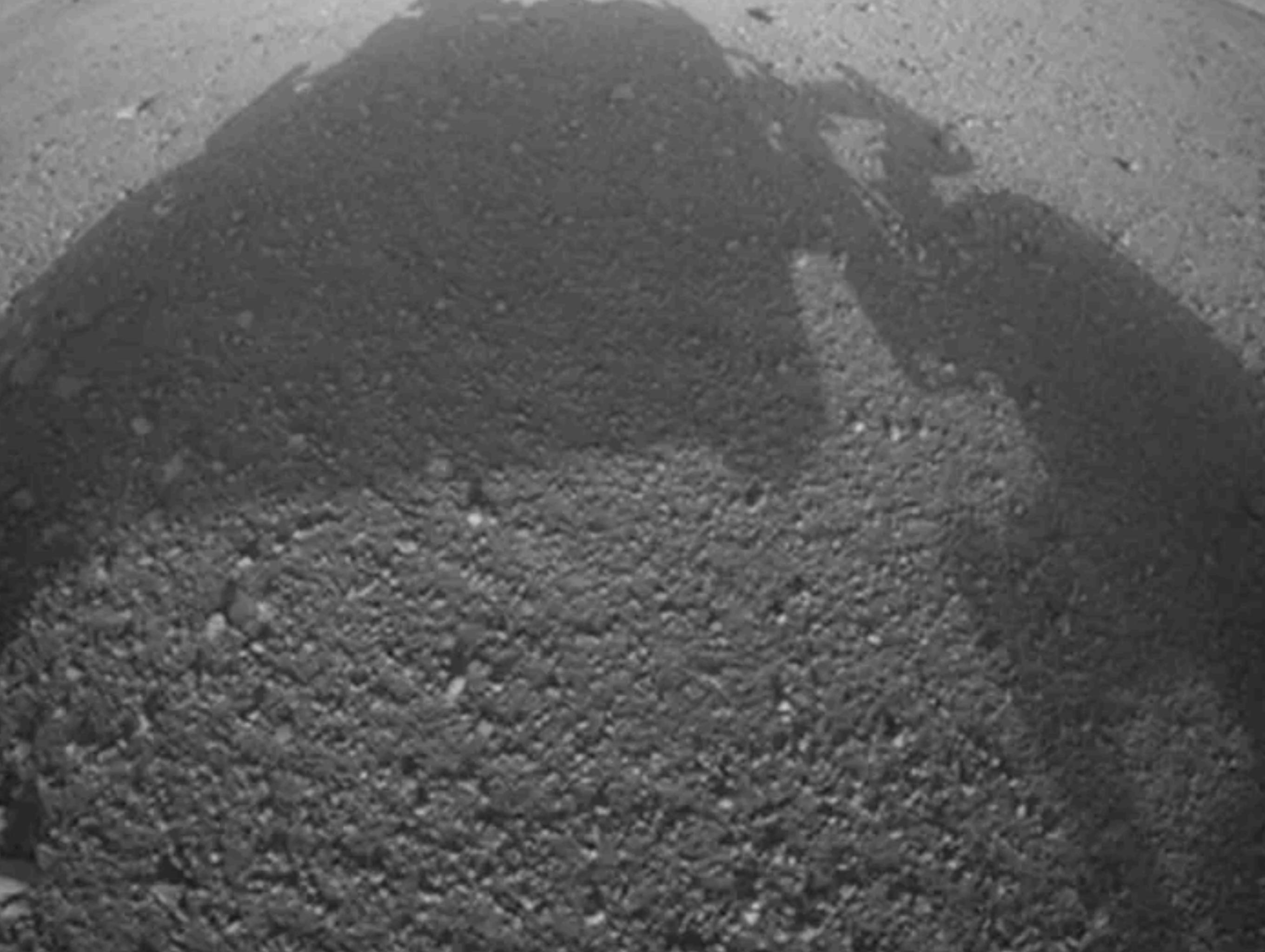
Fig. 1.

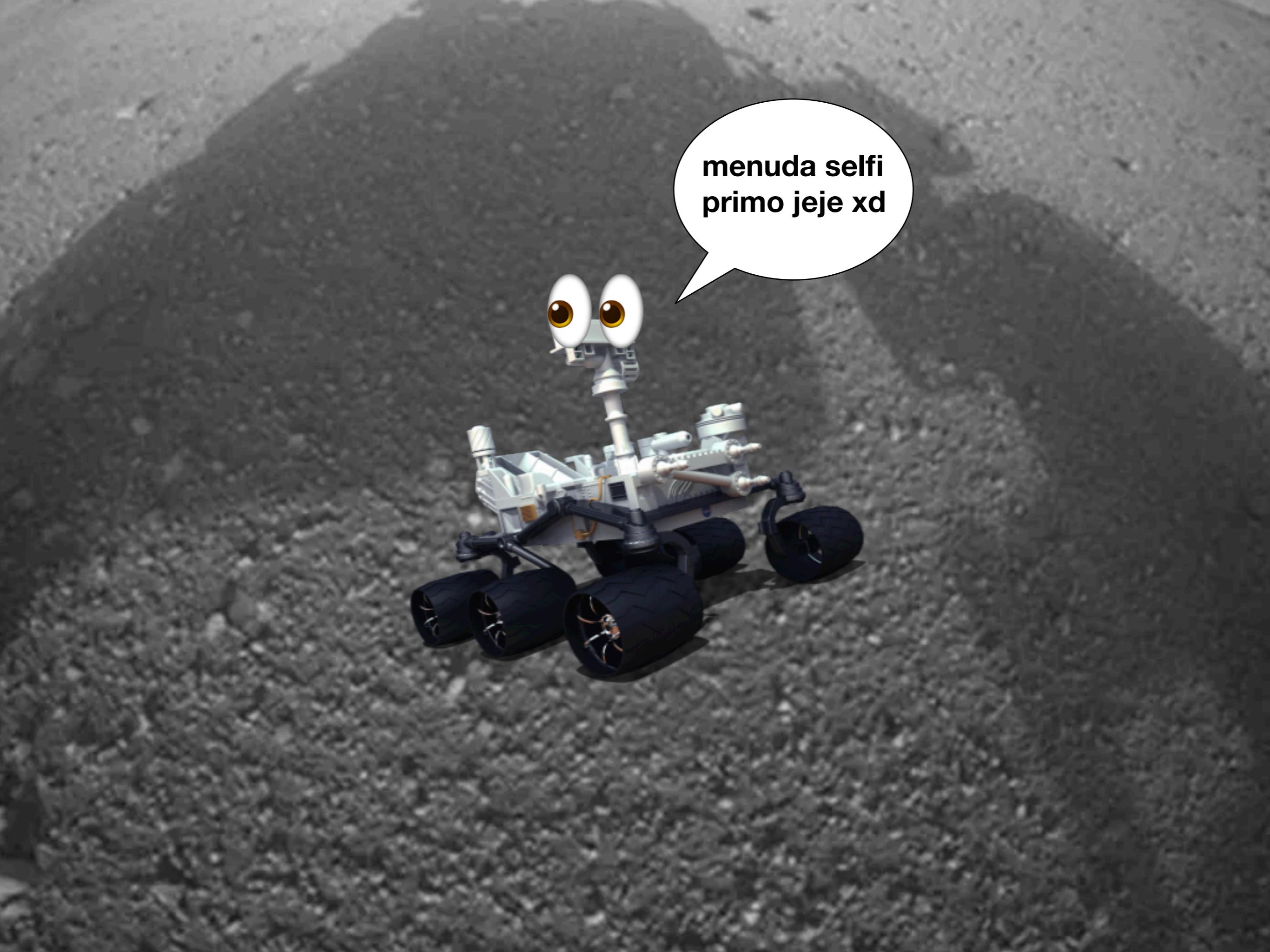


resources



- **Libraries**
 - <https://github.com/davidkpiano/xstate>
 - <https://github.com/MicheleBertoli/react-automata>
 - <https://github.com/cytoscape/cytoscape>
- **Reads**
 - “Constructing the UI with Statecharts” by Ian Horrocks
 - <https://rauchg.com/2015/pure-ui>
 - http://www.inf.ed.ac.uk/teaching/courses/seoc/2005_2006/resources/statecharts.pdf
 - <https://statecharts.github.io/>
- **Real world examples**
 - **VSCode telemetry**





A Curiosity Mars rover is shown from a low angle, appearing to float above a dark, rocky surface. A white, oval-shaped speech bubble originates from the top of the rover's mast, containing the text "menuda selfi primo jeje xd". The rover has a white body, black wheels, and a mast with two large, expressive eyes and a small mouth.

**menuda selfi
primo jeje xd**

- **Libraries**
 - <https://github.com/davidkpiano/xstate>
 - <https://github.com/MicheleBertoli/react-automata>
 - <https://github.com/cytoscape/cytoscape>
- **Reads**
 - “Constructing the UI with Statecharts” by Ian Horrocks
 - <https://rauchg.com/2015/pure-ui>
 - http://www.inf.ed.ac.uk/teaching/courses/seoc/2005_2006/resources/statecharts.pdf
 - <https://statecharts.github.io/>
- **Real world examples**
 - VSCode telemetry
 - Curiosity rover

We're hiring!

r.sospedra@mytaxi.com

Cities

Hamburg (HQ)

Berlin

Barcelona

Roles

iOS & Android Developer

Backend & Frontend Developer

QA Engineer

BI & Data Science





thank you

<https://github.com/sospedra/talks>

<https://github.com/sospedra/finite-state-machine-demo>