React Explained in Five Visual Metaphors

Women of React Conf
April 2020
@mappletons

Drawing the Invisible

DRAW METAPHORS ◦ WIN FRIENDS ◦ INFLUENCE PEOPLE ◦ PROFIT

Maggie

maggieappleton.com
illustrated.dev

I make illustrations that help explain invisible, abstract programming concepts

egghead.io

Art Director
Illustrator
Metaphor Designer
Anthropologist


DECIMAL
BOOLEAN
INTEGER
STRING
STRING
TS


CSS

Illustrated explanations of web development, technology & a little bit of anthropology.

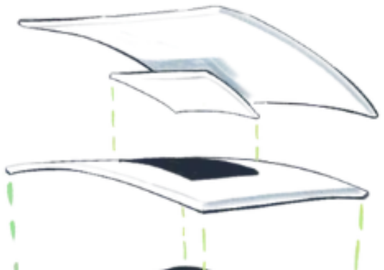
Illustrated Explanations & Notes

ALL POSTS EXPLAINERS ILLUSTRATED NOTES

The Art and Craft of Gatsby Themes

Updated April 18, 2020

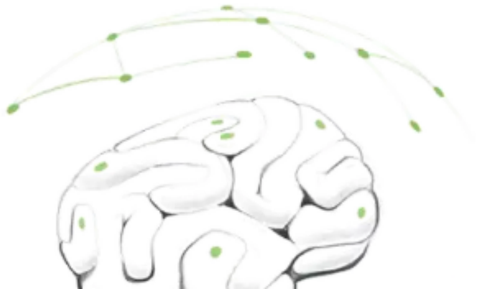
Illustrated Notes



How to Build a Second Brain – Part 2

Updated April 04, 2020

Illustrated Notes



A Chat with Henry Zhu on Open Source and Gift Economies

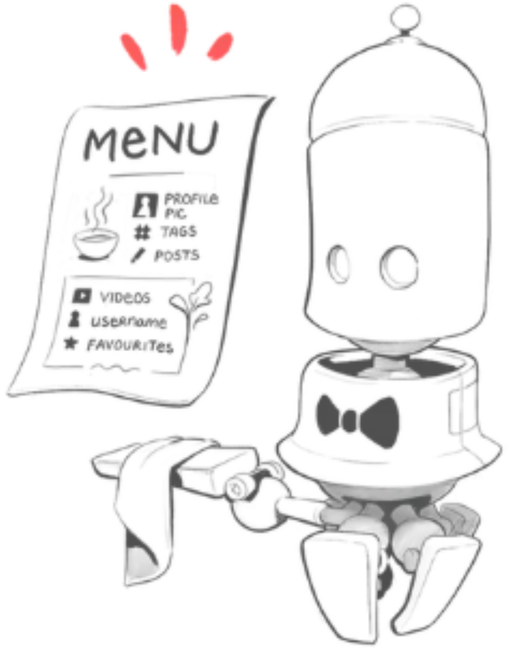
Updated March 30, 2020

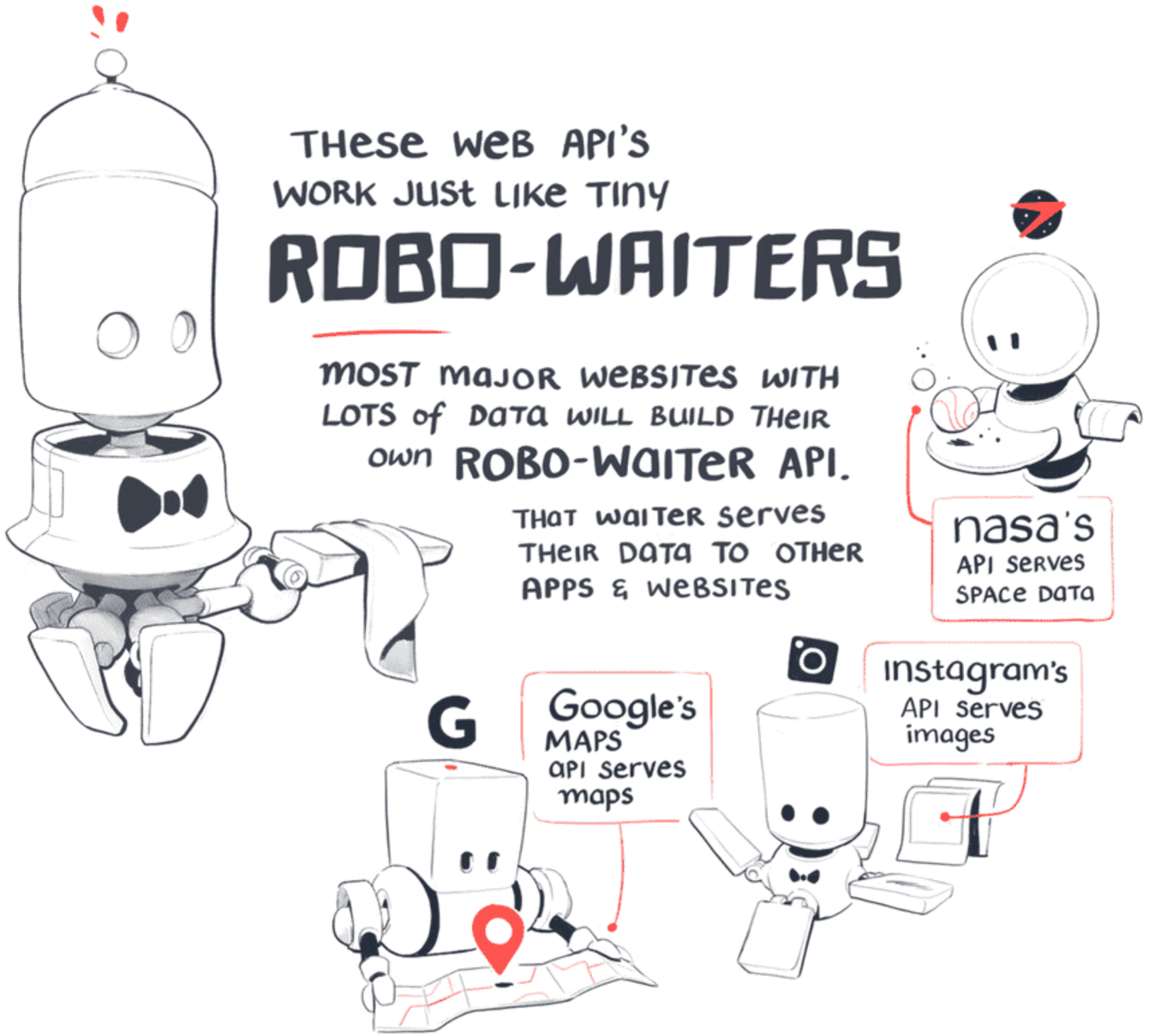
A Visual Explanation

Meet the Robowaiter APIs Serving Us Data

Updated April 10, 2019

A Visual Explanation

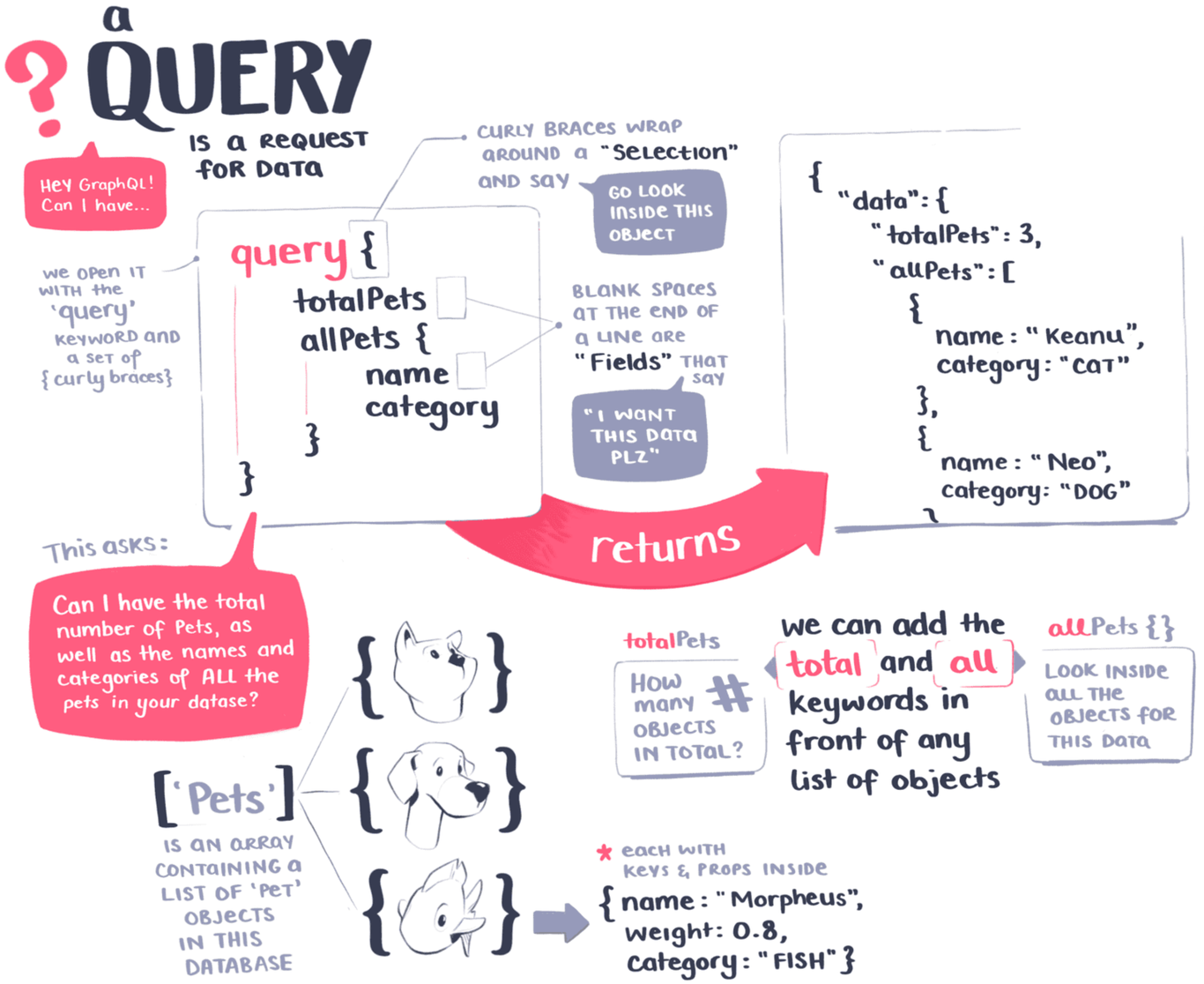




Speaking the GraphQL Query Language

Updated July 30, 2019

Illustrated Notes



All of these are about making

Abstract, Invisible, Intangible Programming Stuff

visible and easier to understand

Aka. Making

Visual Metaphors

Warning: metaphors are not in-depth code demos

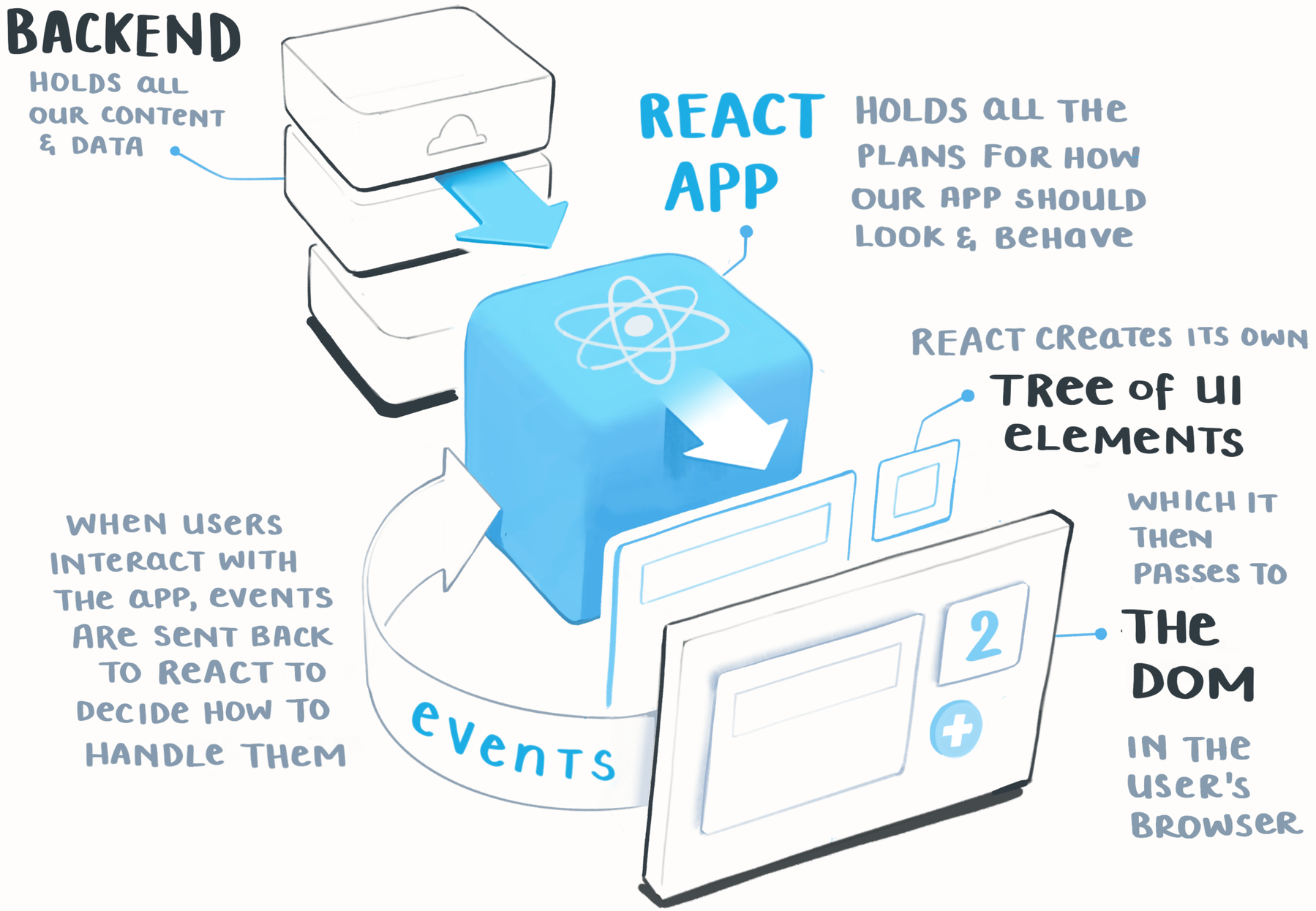
Metaphors are big picture, imprecise introductions

The Fundamentals of React...

In Five Visual Metaphors

Metaphor One

Drawing the Map of a React App


BACKEND
HOLDS ALL OUR CONTENT & DATA
REACT APP
HOLDS ALL THE PLANS FOR HOW OUR APP SHOULD LOOK & BEHAVE
REACT CREATES ITS OWN
TREE OF UI ELEMENTS
WHICH IT THEN PASSES TO
THE DOM
IN THE USER'S BROWSER
WHEN USERS INTERACT WITH THE APP, EVENTS ARE SENT BACK TO REACT TO DECIDE HOW TO HANDLE THEM
EVENTS
2

node_modules

react-dom

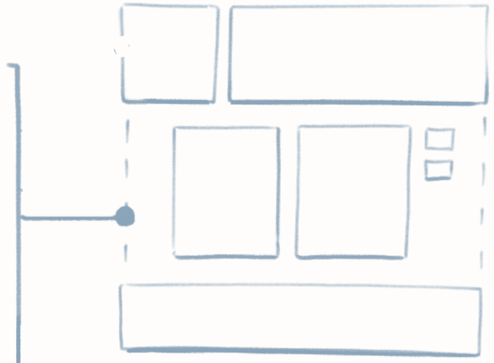
react

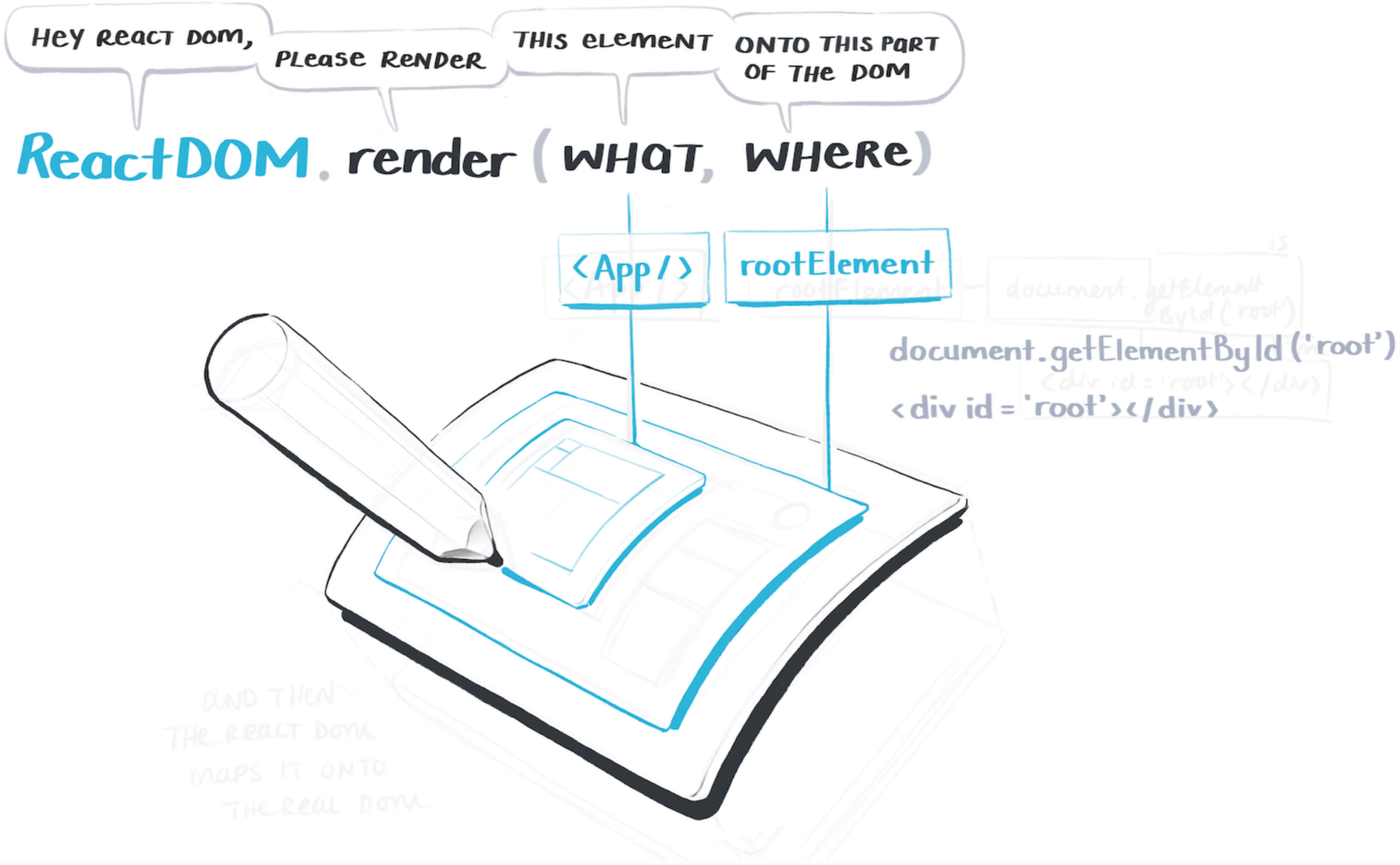
components

- Header.js
- Articles.js
- Footer.js

```
import React from 'react'
import ReactDOM from 'react-dom'

import Header from './components/Header'
import Articles from './components/Articles'
import Footer from './components/Footer'

const App = () => {
 return (
  <>
   <Header />
   <Articles />
   <Footer />
  </>
 )
}

const rootElement = document.getElementById('root')
ReactDOM.render(<App />, rootElement)
```

THIS IS WHERE YOU BUILD THE STRUCTURE OF YOUR APP




HEY REACT DOM,
PLEASE RENDER
THIS ELEMENT
ONTO THIS PART OF THE DOM
ReactDOM.render(WHAT, WHERE)
<App />
rootElement
document.getElementById('root')

AND THEN THE REACT DOM MAPS IT ONTO THE REAL DOM

node_modules

react-dom

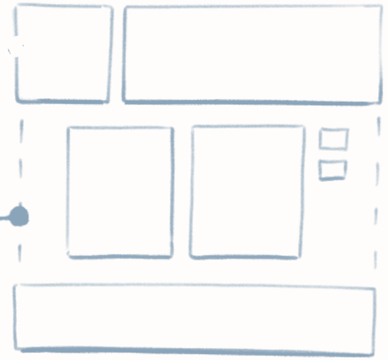
react

components

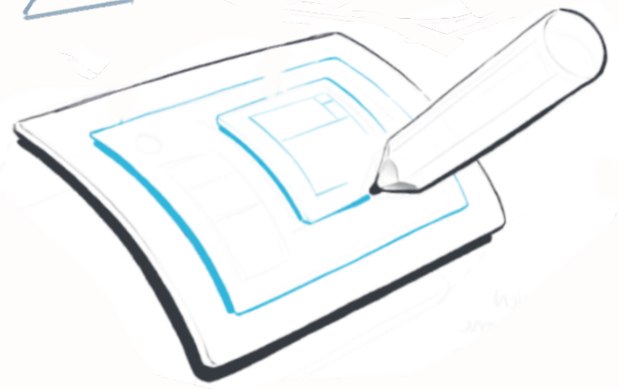
- Header.js
- Articles.js
- Footer.js

```
1  import React from 'react'
2  import ReactDOM from 'react-dom'
3
4  import Header from './components/Header'
5  import Articles from './components/Articles'
6  import Footer from './components/Footer'
7
8  const App = () => {
9   return (
10   <>
11    <Header />
12    <Articles />
13    <Footer />
14   </>
15  )
16 }
17
18 const rootElement = document.getElementById('root')
19 ReactDOM.render(<App />, rootElement)
```

THIS IS WHERE YOU BUILD THE STRUCTURE OF YOUR APP



RENDER TO THE DOM



Metaphor Two

Putting a JSX Cherry on Top of the Vanilla JS Stack

WE WRITE **React**
IN A SPECIAL FLAVOUR
OF JAVASCRIPT
CALLED **JSX** (X FOR EXTREME?)

IT'S AN EXTRA CHERRY & SPRINKLES ON TOP OF WHAT IS STILL MOSTLY

VANILLA JS

```
let color = "blue"                    JS
return (                              JSX
  < MainButton
    activeColor = {color} />
)
```

WHILE SOME PARTS OF JSX LOOK DIFFERENT TO VANILLA JS, IT'S JUST AN EXTENSION OF THE LANGUAGE.

IT ADDS

"Syntactic Sugar"

TO JAVASCRIPT

aka.

MAKES THE CODE LOOK NICER & IT'S EASIER TO READ

In JSX we swirl a bunch of

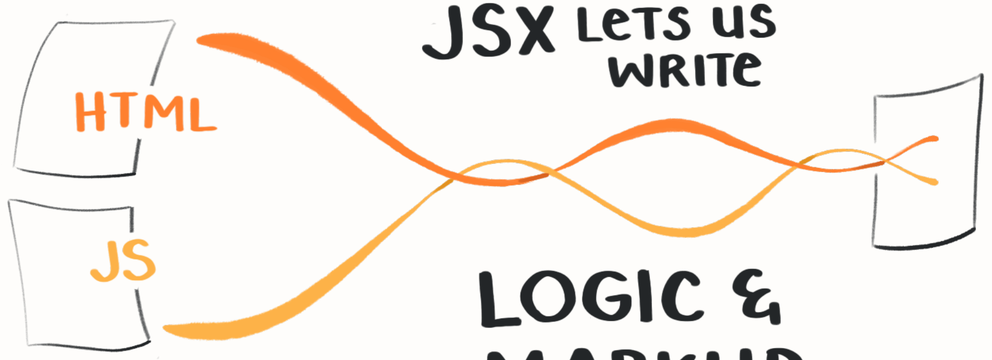
MARK UP

(THE SYNTAX WE WRITE HTML IN)

INTO OUR VANILLA JS ICE CREAM

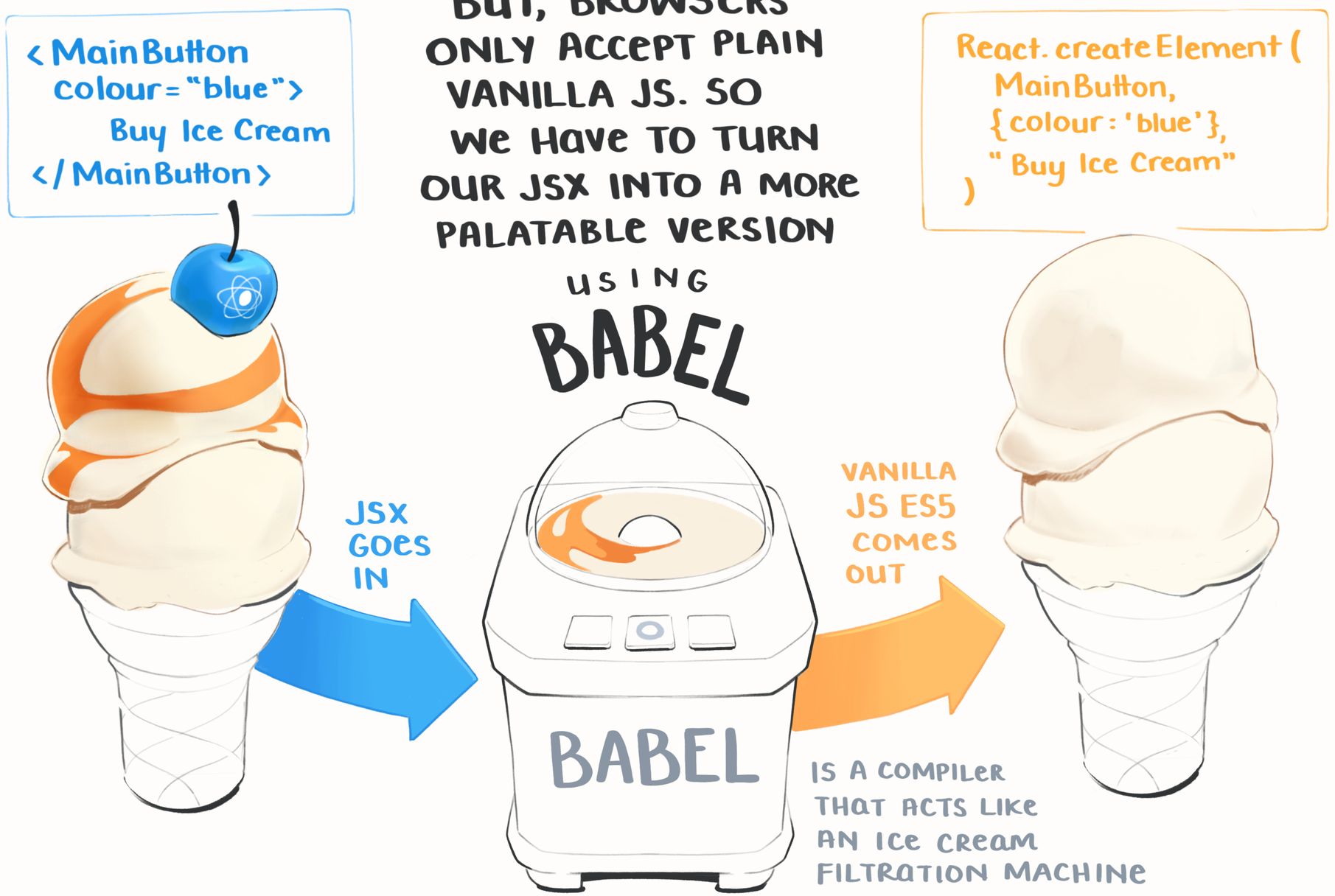
```
function Menu(){
  let flavour = "JSX Swirl"
  return(
    <div>
      <h1> {flavour} </h1>
    </div>
})
```

INSTEAD OF HAVING TWO SEPARATE FILES,

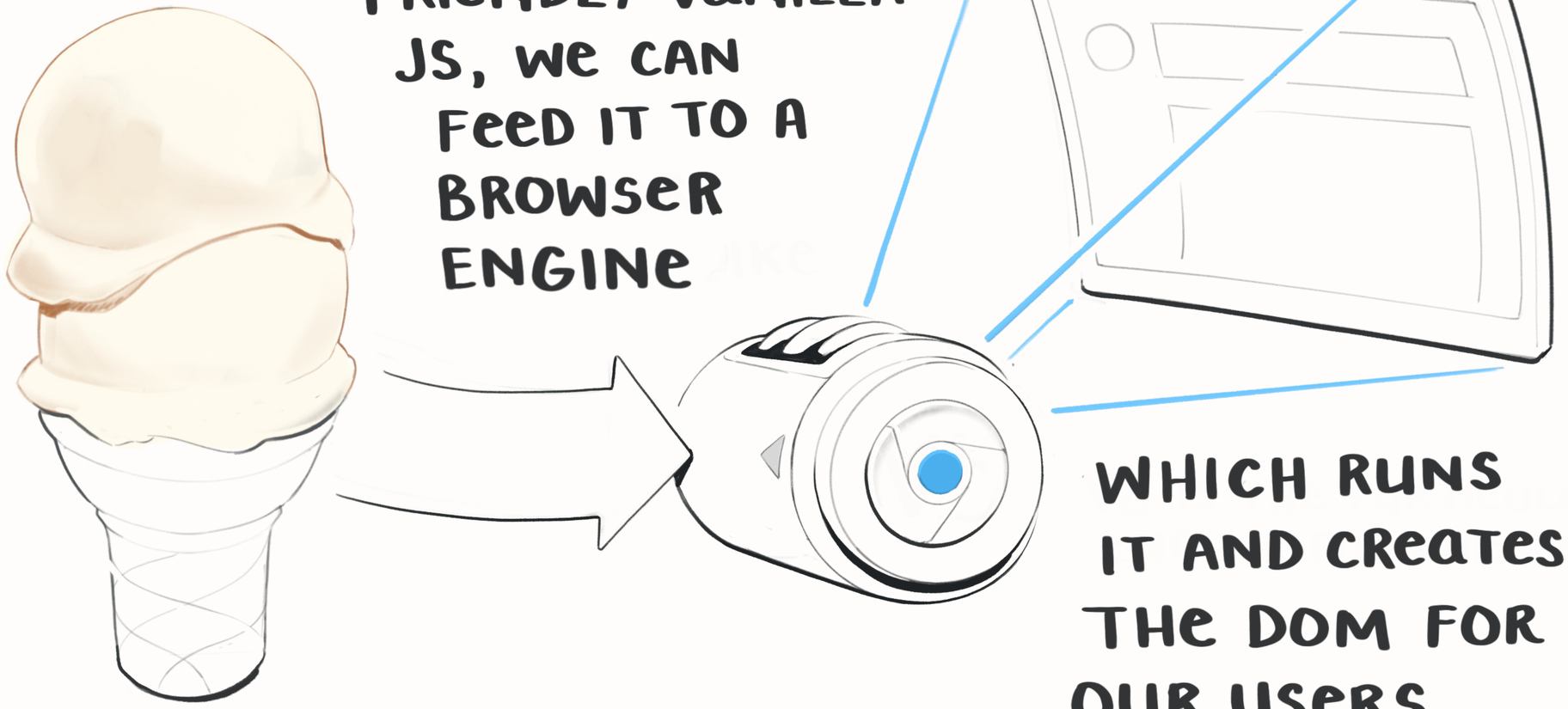


JSX LETS US WRITE

LOGIC & MARKUP SIDE-BY-SIDE


<MainButton
colour="blue">
Buy Ice Cream
</MainButton>
BUT, BROWSERS ONLY ACCEPT PLAIN VANILLA JS. SO WE HAVE TO TURN OUR JSX INTO A MORE PALATABLE VERSION
USING
BABEL
React.createElement(
MainButton,
{colour: 'blue'},
"Buy Ice Cream"
)
JSX GOES IN
VANILLA JS ES5 COMES OUT
BABEL
IS A COMPILER THAT ACTS LIKE AN ICE CREAM FILTRATION MACHINE

ONCE BABEL CREATES OUR BROWSER-FRIENDLY VANILLA JS, WE CAN FEED IT TO A BROWSER ENGINE

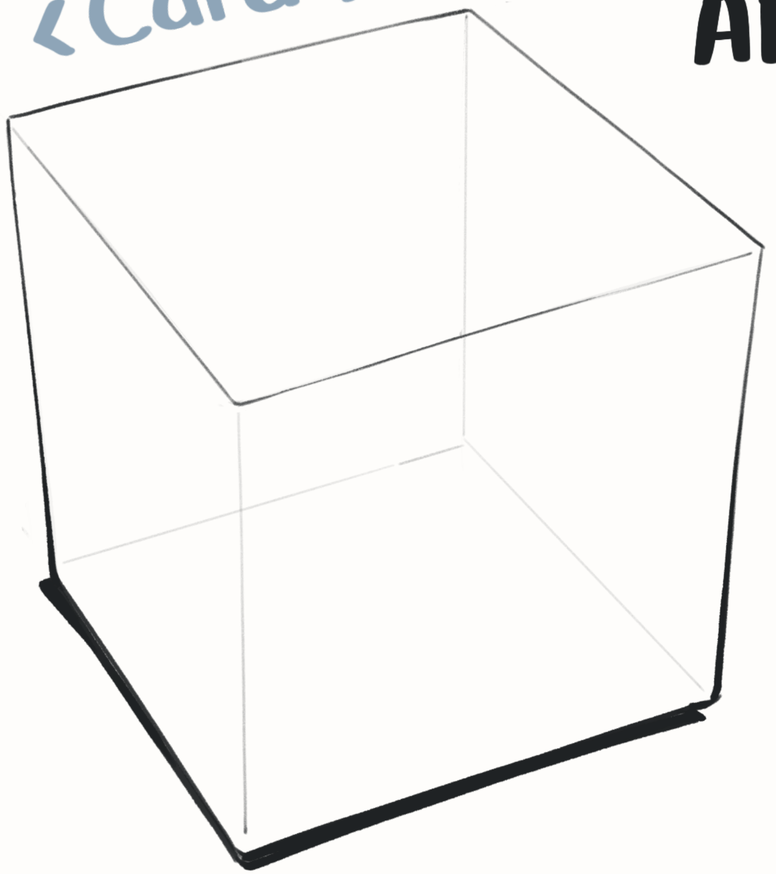


WHICH RUNS IT AND CREATES THE DOM FOR OUR USERS

Metaphor Three

Stage Play Components and Magic Carpet Props

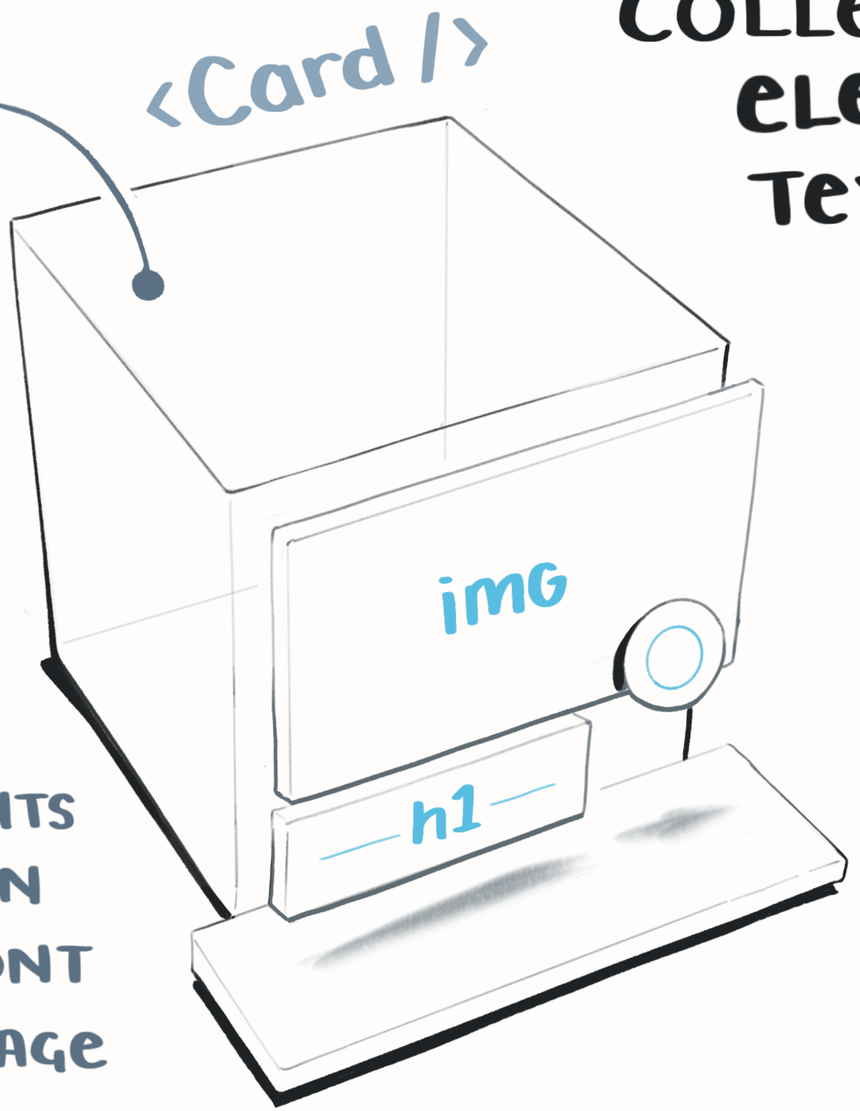
<Card />



COMPONENTS ARE CONTAINERS

COMPONENTS ARE COLLECTIONS OF UI ELEMENTS LIKE TEXT, IMAGES & BUTTONS

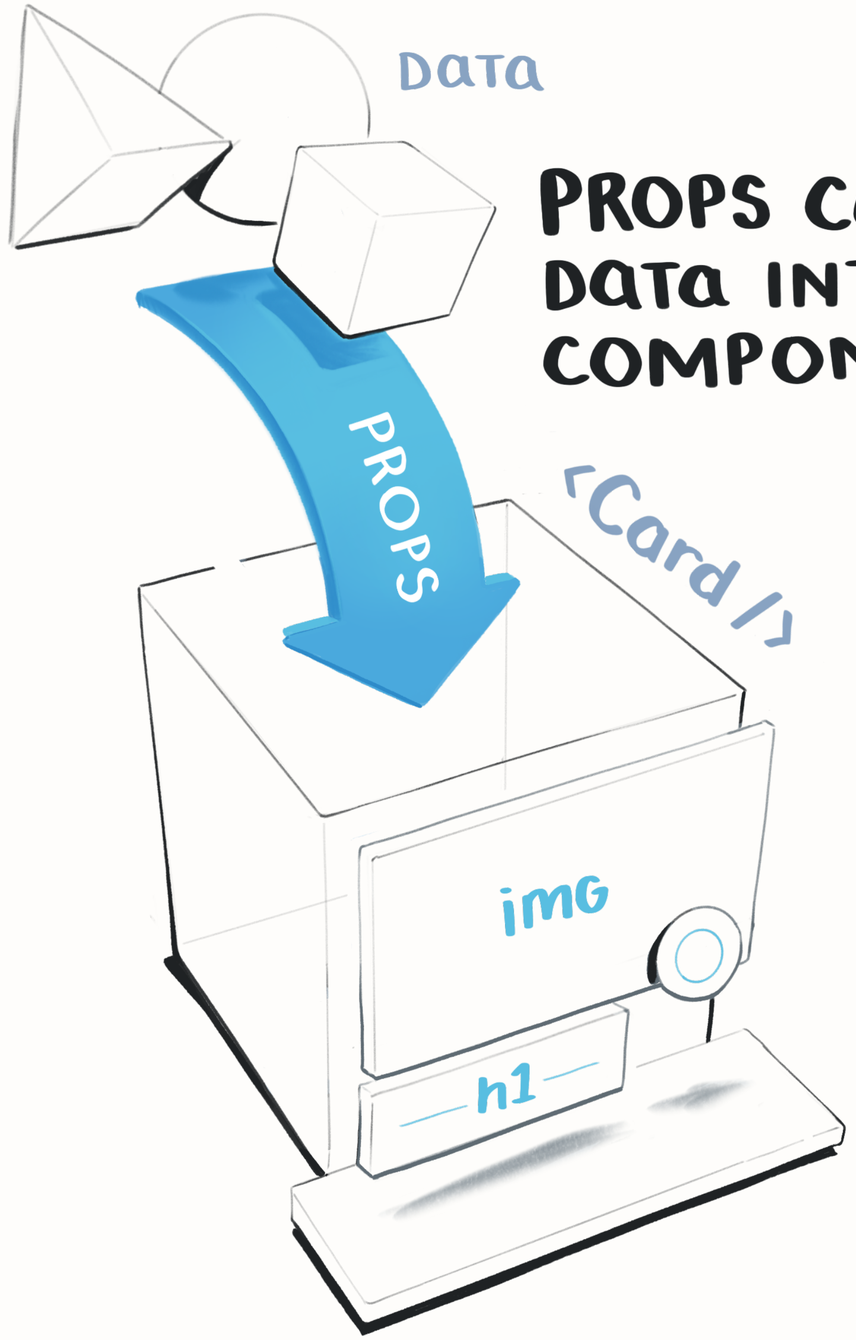
LOGIC & ACTION HAPPEN BACK STAGE



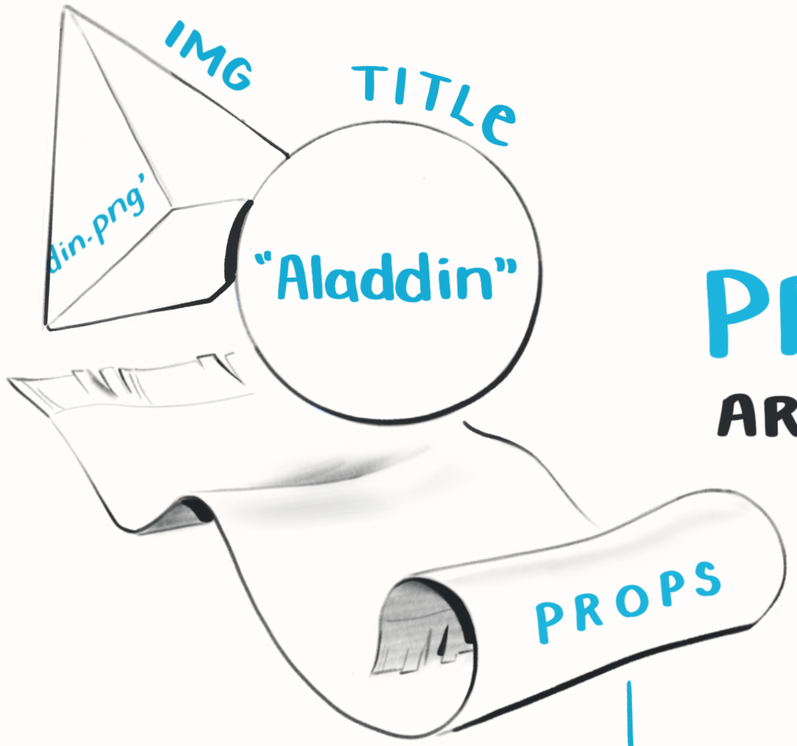
Like a Play, The elements perform on the front stage

```
function Card(props){
    return(
        <img src={props.img} />
        <button />
        <h1>{props.title}</h1>
    )
}
```

PROPS CARRY DATA INTO COMPONENTS




```
function Card(props){
    return(
        <img src={props.img} />
        <button />
        <h1>{props.title}</h1>
    )
}
```



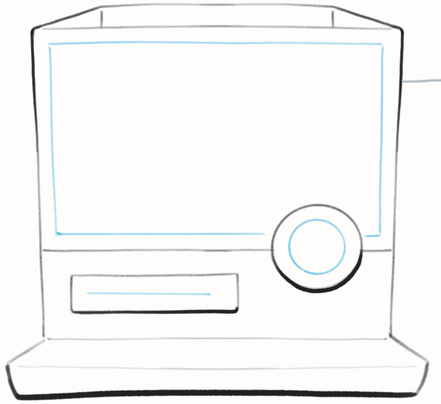
PROPS

ARE THE MAGIC CARPET OUR DATA RIDES IN ON

IT'S A JAVASCRIPT OBJECT THAT CARRIES THE PROPERTIES YOU DECLARE IN YOUR COMPONENT

```
function Card( props ){
  return(
    <img src = { props.img } />
    <button />
    <h1> { props.title } </h1>
  )
}
```

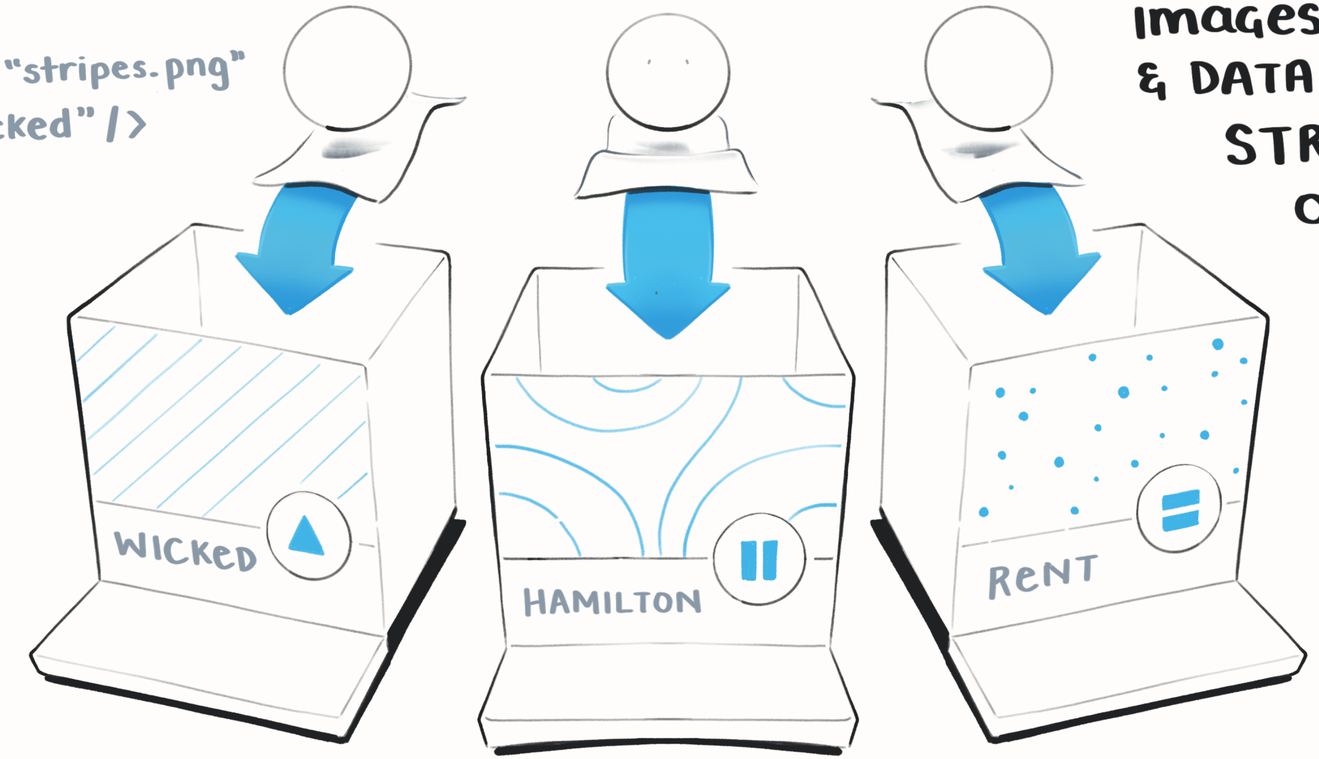
SO WE ONLY NEED TO SET THE STAGE FOR A COMPONENT **once**



```
function Card(props){
  return(
    <img src={props.img} />
    <button />
    <h1>{props.title}</h1>
  )
}
```

AND THEN we can pass different images, words, & data into that structure on the **PROPS CARPET**

```
<Card img="stripes.png"
  title="Wicked" />
```

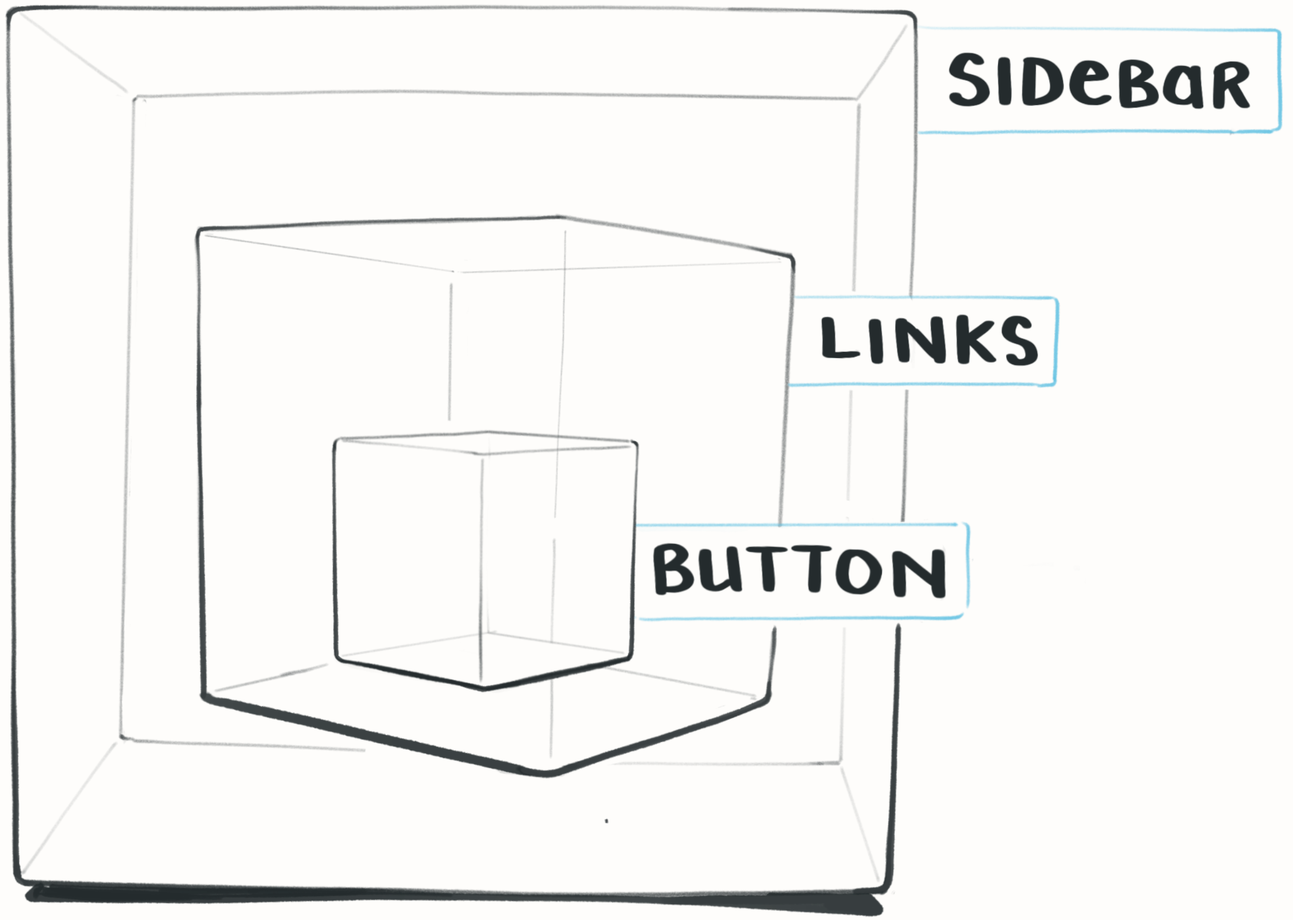


IT's a different play each time!

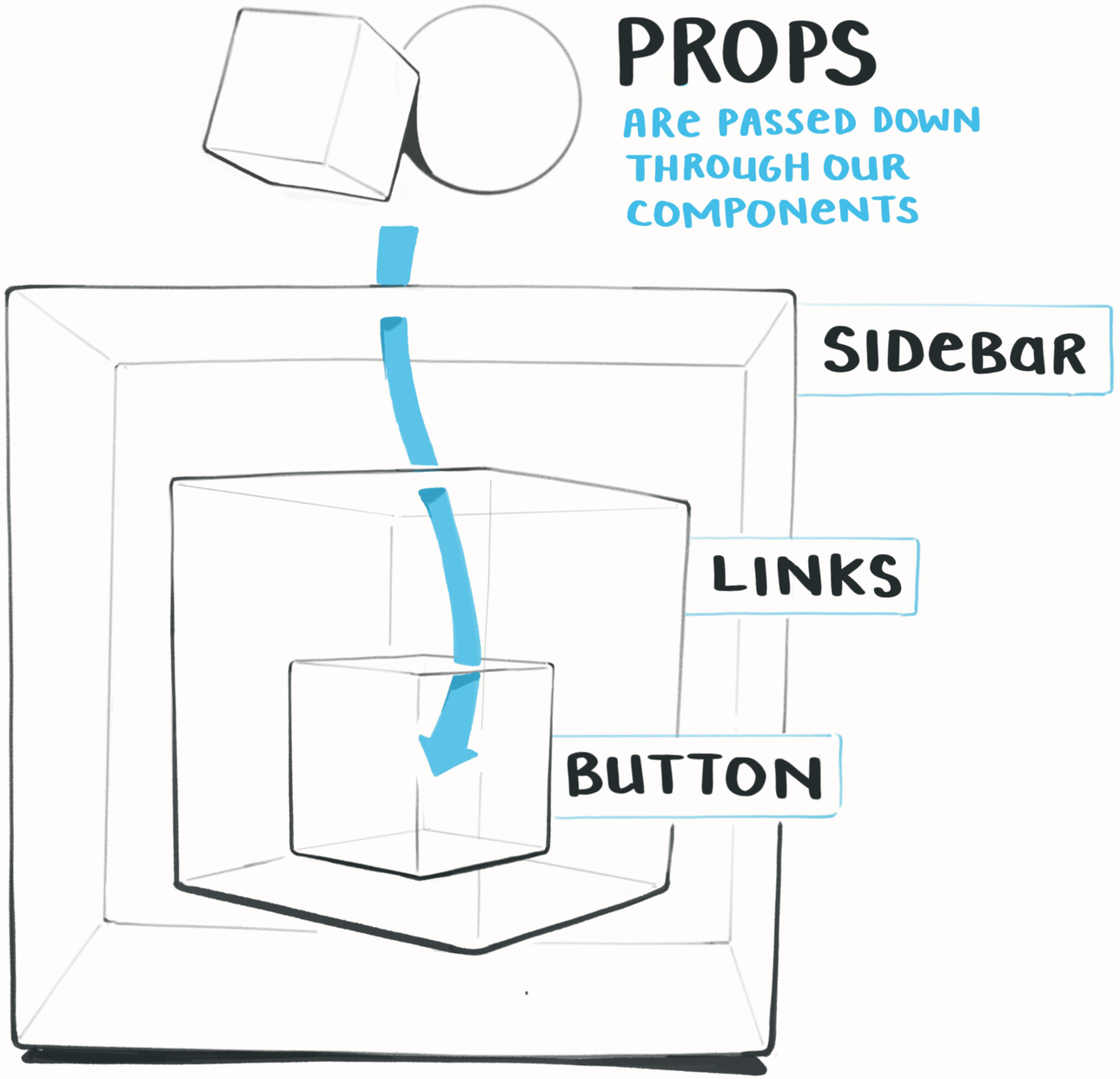
Metaphor Four

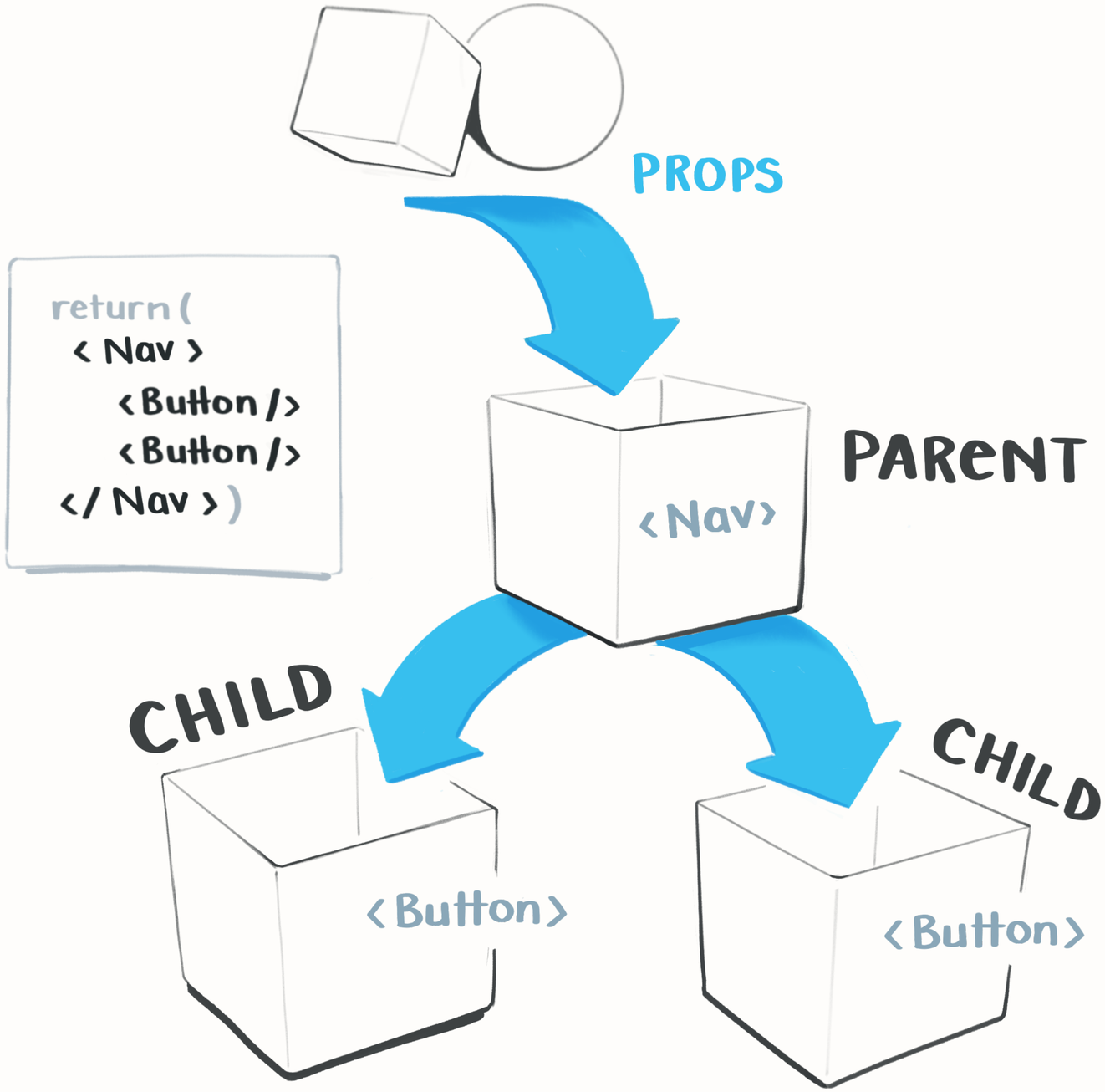
Tree Structures, Potato Plants and Data Flow


```
return (
  <Sidebar>
    <Links>
      <Button>
        Order
      </Button>
    </Links>
  </Sidebar> )
```

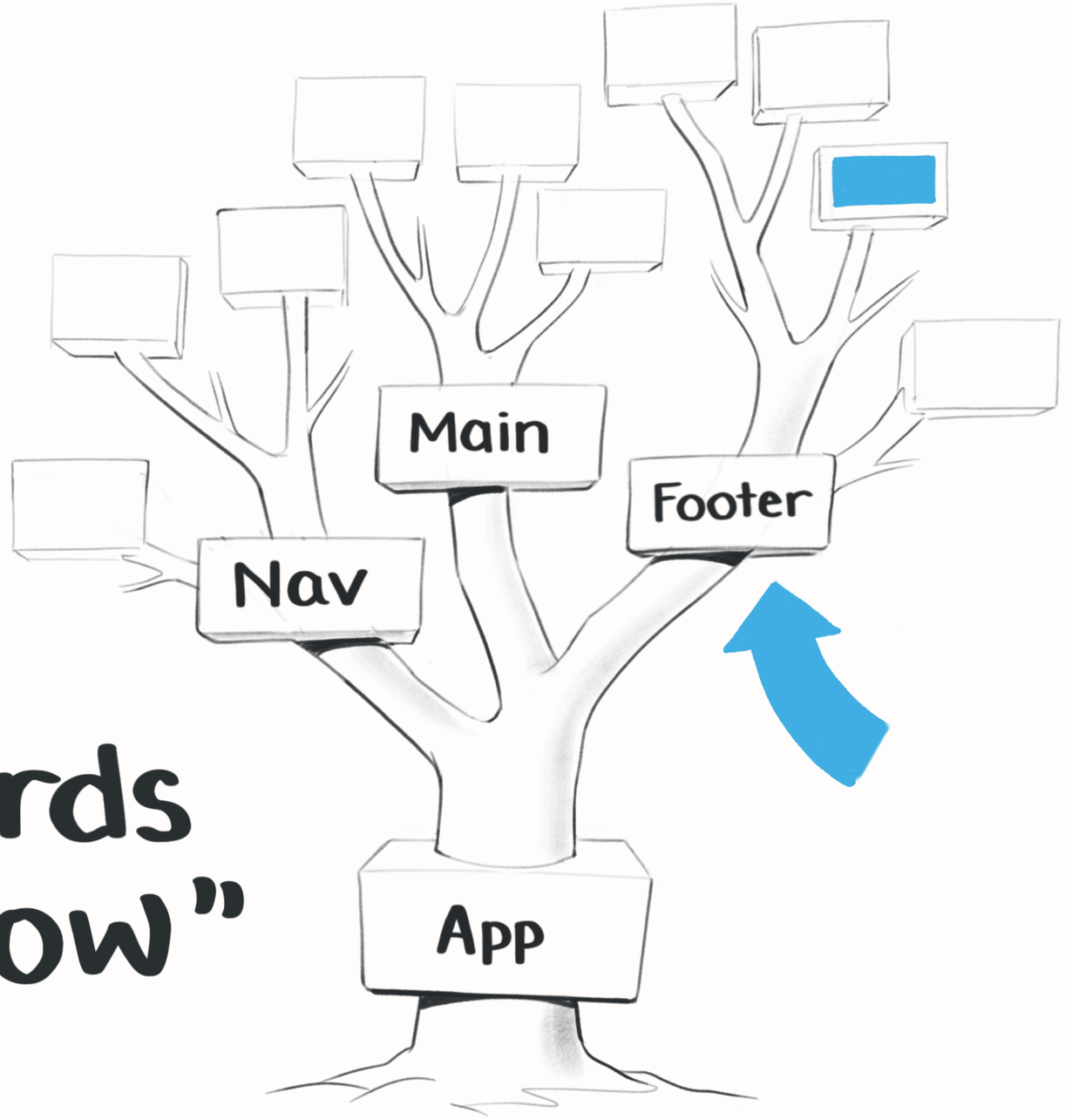



```
return (
  <Sidebar>
    <Links>
      <Button>
        Order
      </Button>
    </Links>
  </Sidebar> )
```

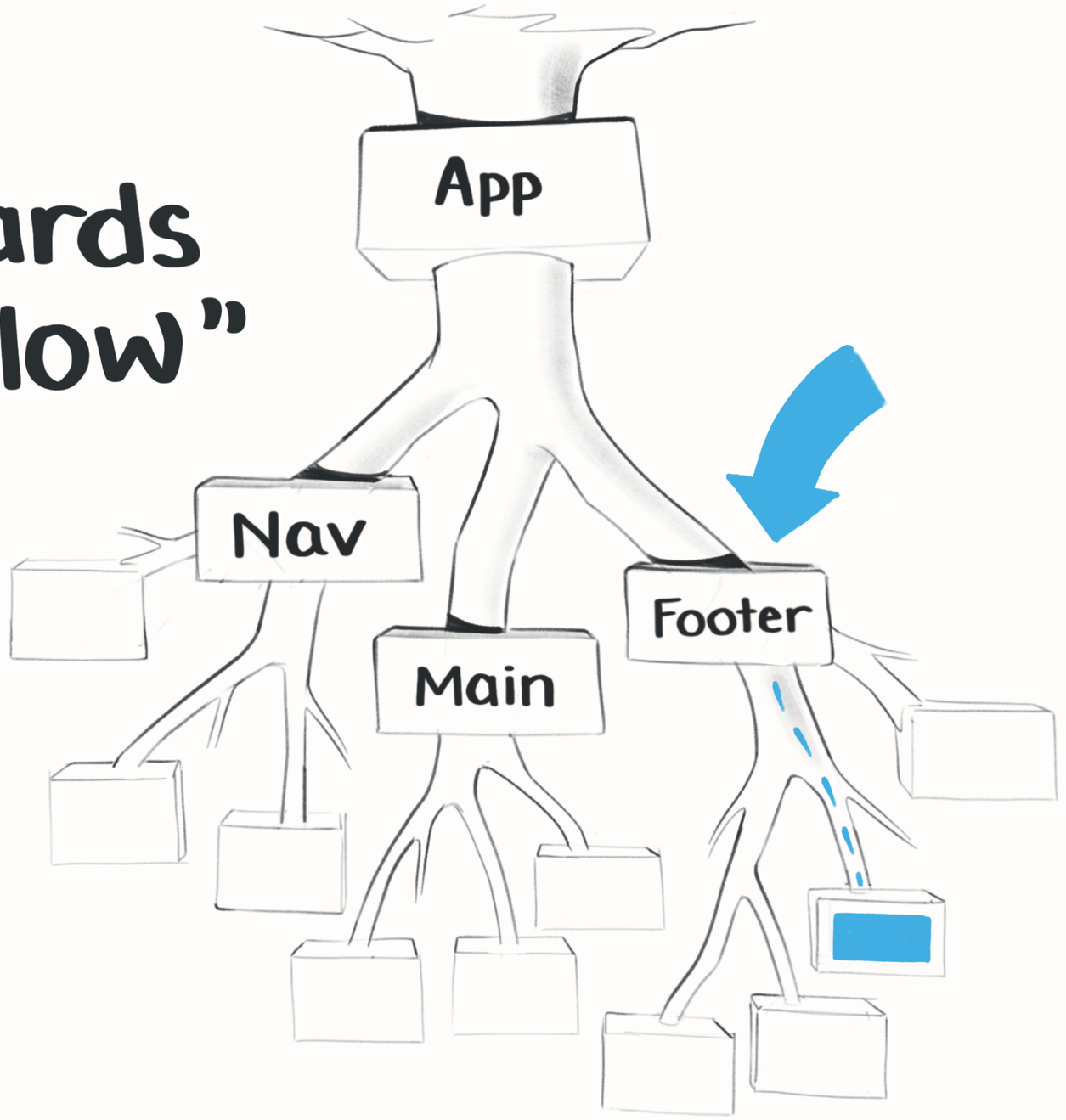



PROPS
return (
< Nav >
<Button />
<Button />
</ Nav >)
PARENT
<Nav>
CHILD
<Button>
CHILD
<Button>

"Downwards data flow"

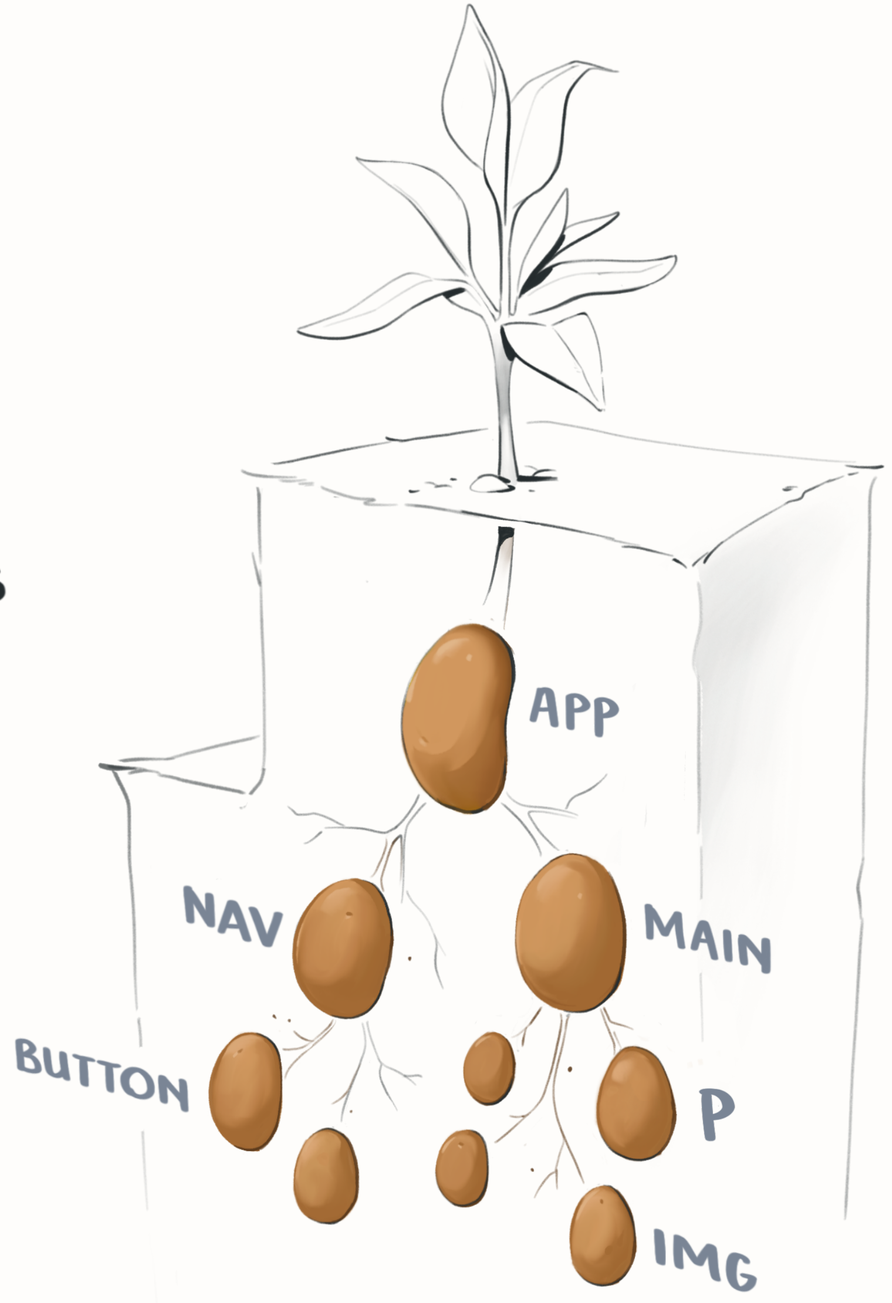


"Downwards data flow"



REACT IS A POTATO PLANT

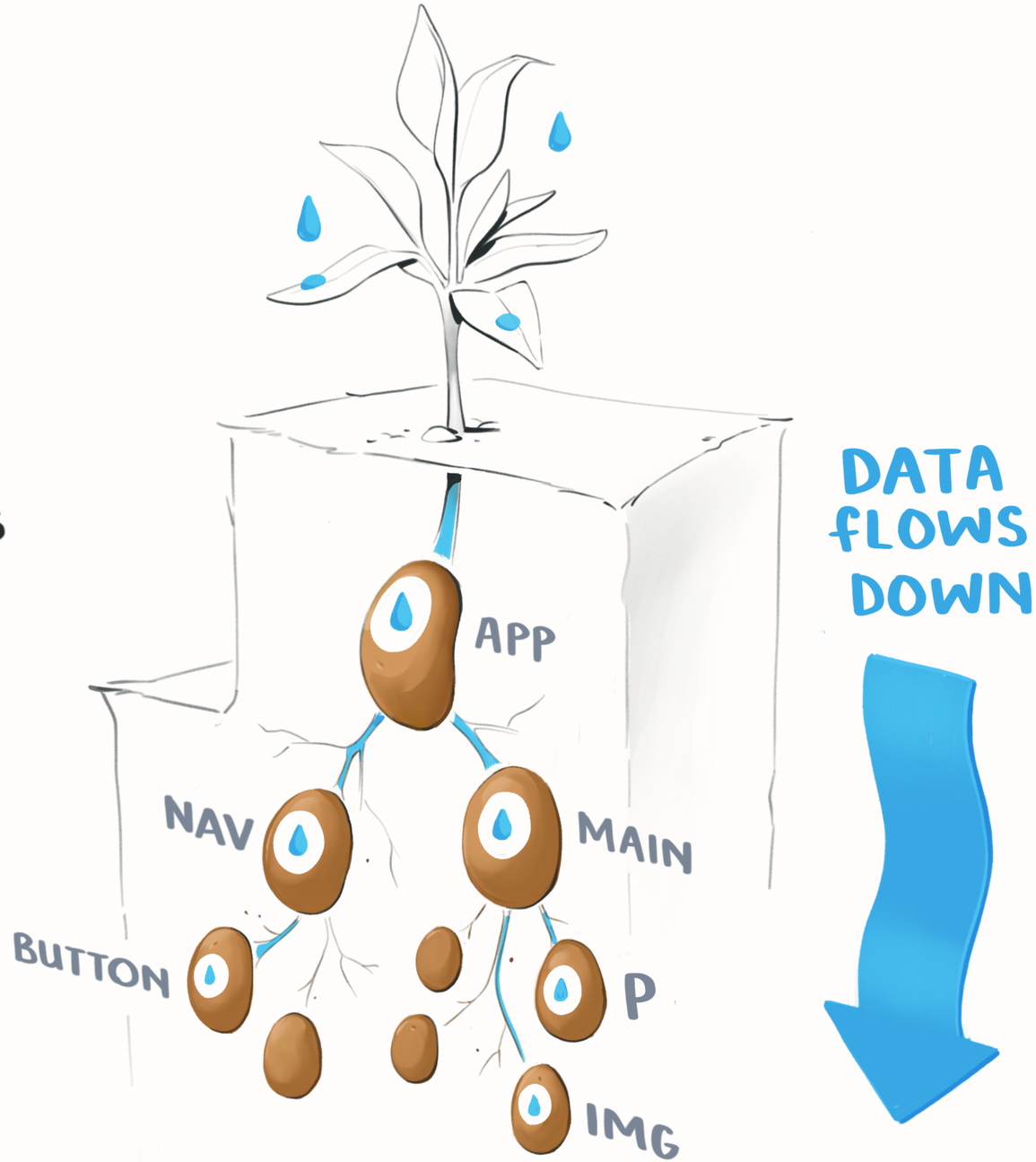
POTATO COMPONENTS ARE CONNECTED BY THE PROPS ROOT NETWORK



REACT IS A POTATO PLANT

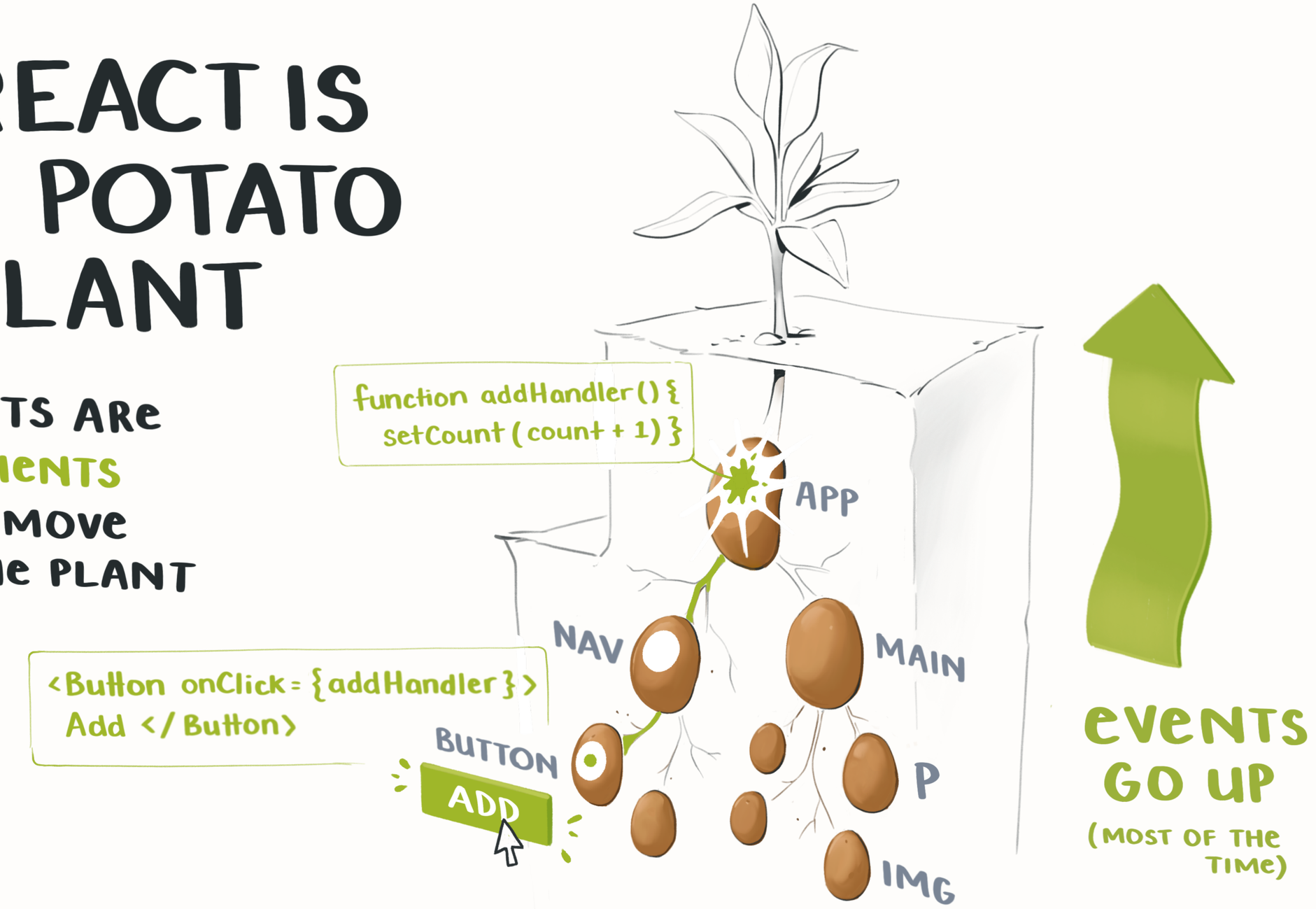
POTATO COMPONENTS ARE CONNECTED BY THE PROPS ROOT NETWORK

OUR DATA IS WATER FLOWING *DOWN* THE PLANT



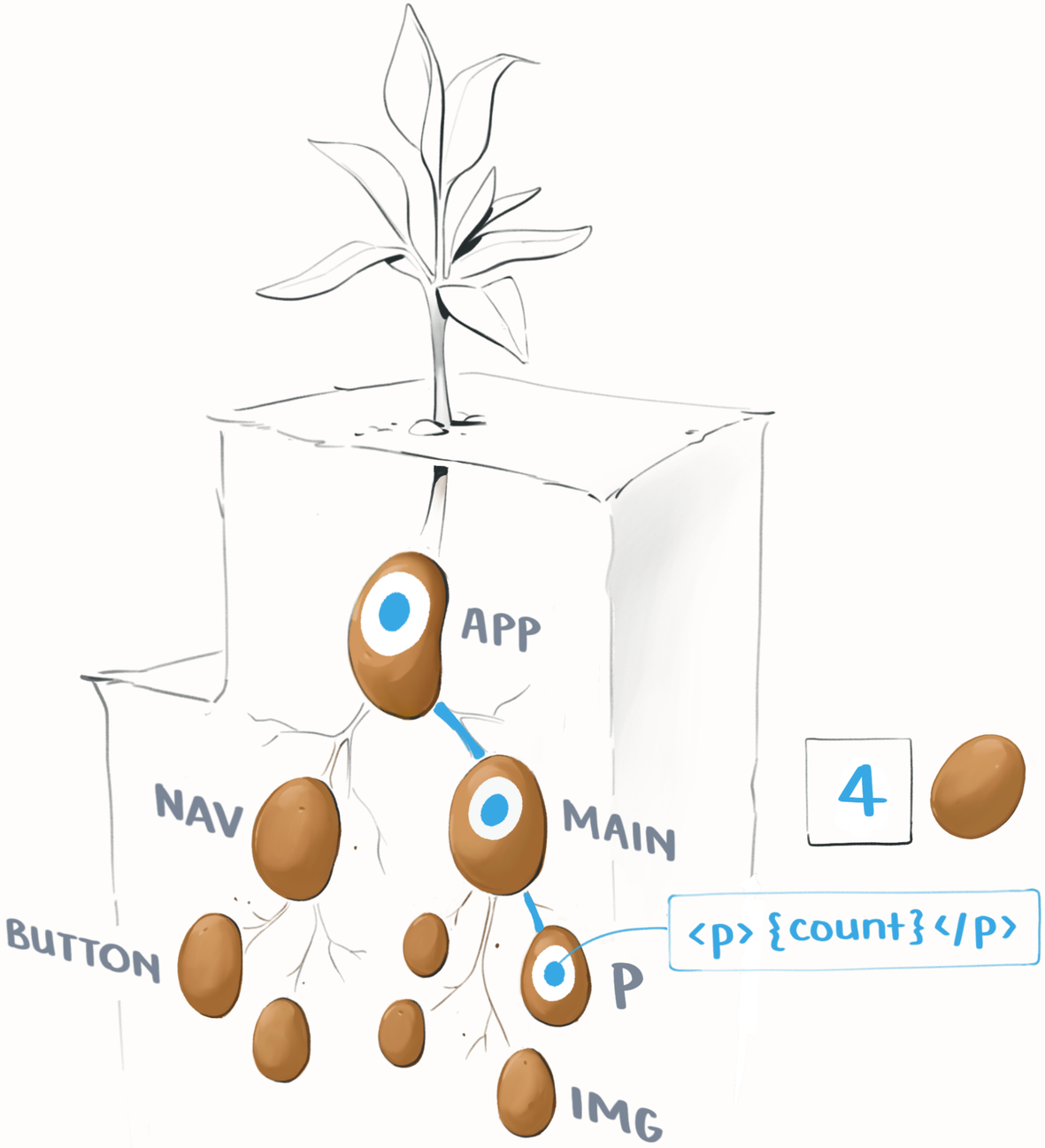
REACT IS A POTATO PLANT

EVENTS ARE **NUTRIENTS** THAT MOVE UP THE PLANT



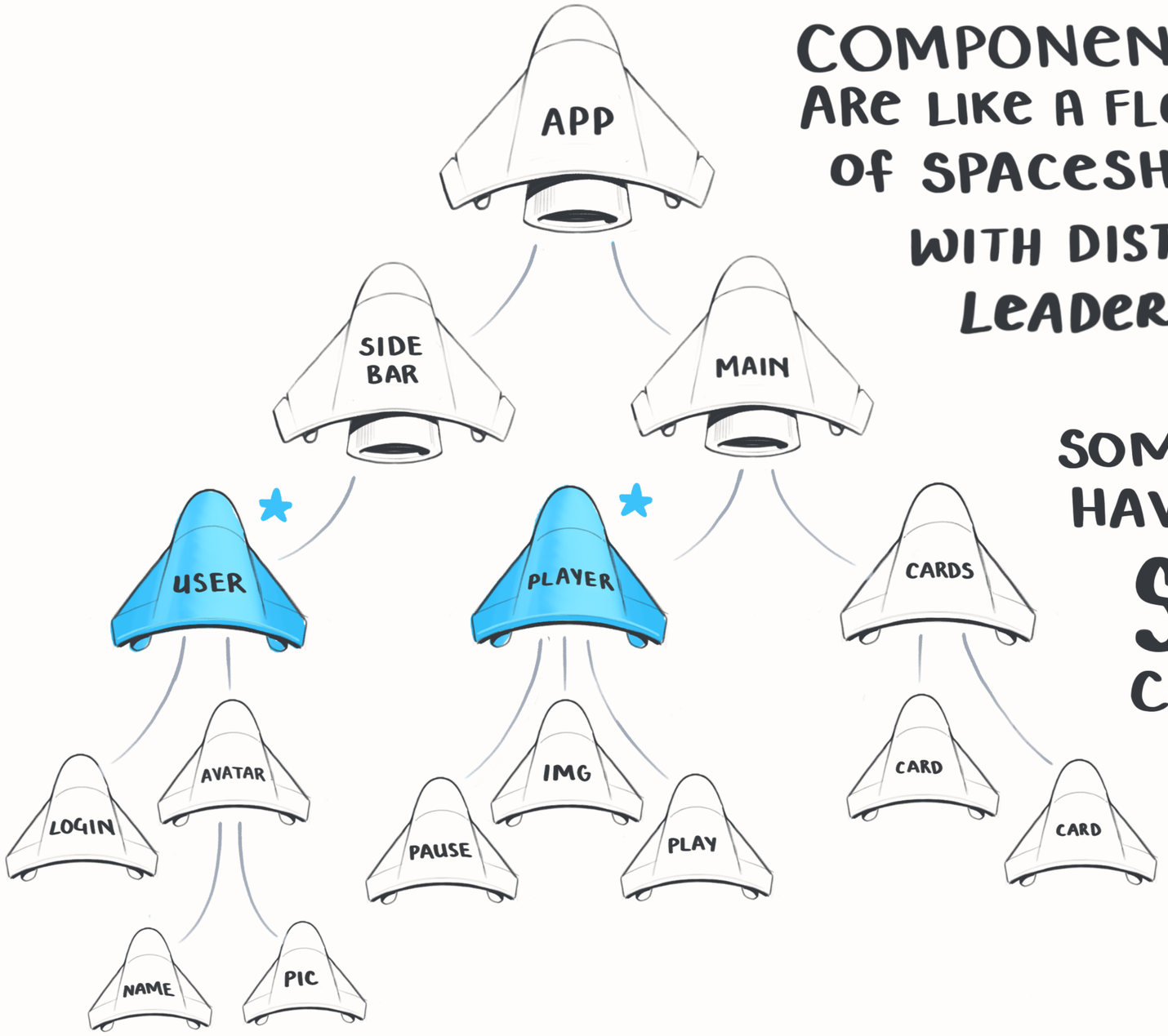
REACT IS A POTATO PLANT

EVENTS ARE NUTRIENTS THAT MOVE UP THE PLANT



Metaphor Five

Props vs State, and Commanding Spaceship Fleets

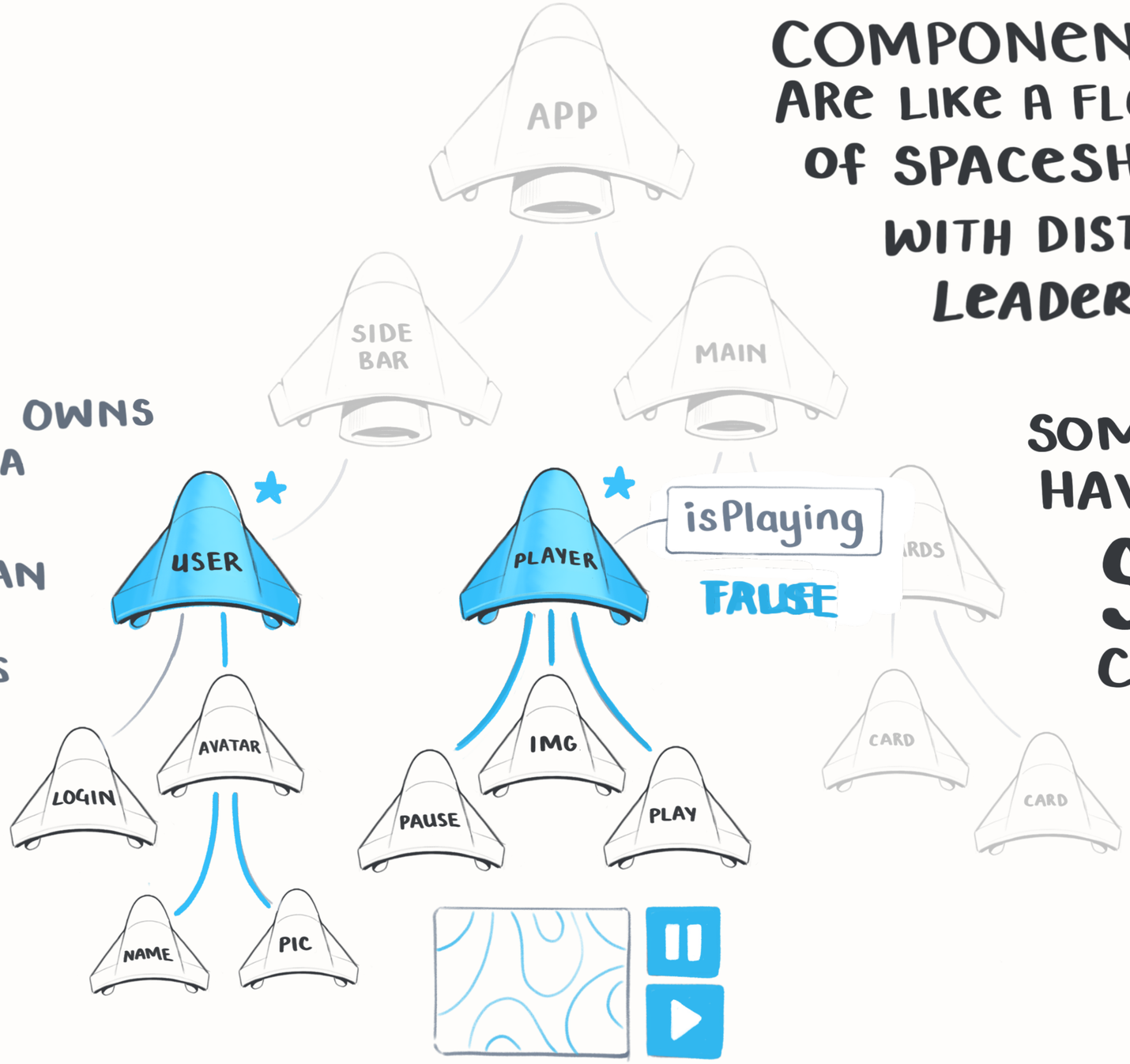


COMPONENTS ARE LIKE A FLEET OF SPACESHIPS WITH DISTRIBUTED LEADERSHIP.

SOME SHIPS HAVE A **STATE** COMMANDER INSIDE

COMPONENTS ARE LIKE A FLEET OF SPACESHIPS WITH DISTRIBUTED LEADERSHIP

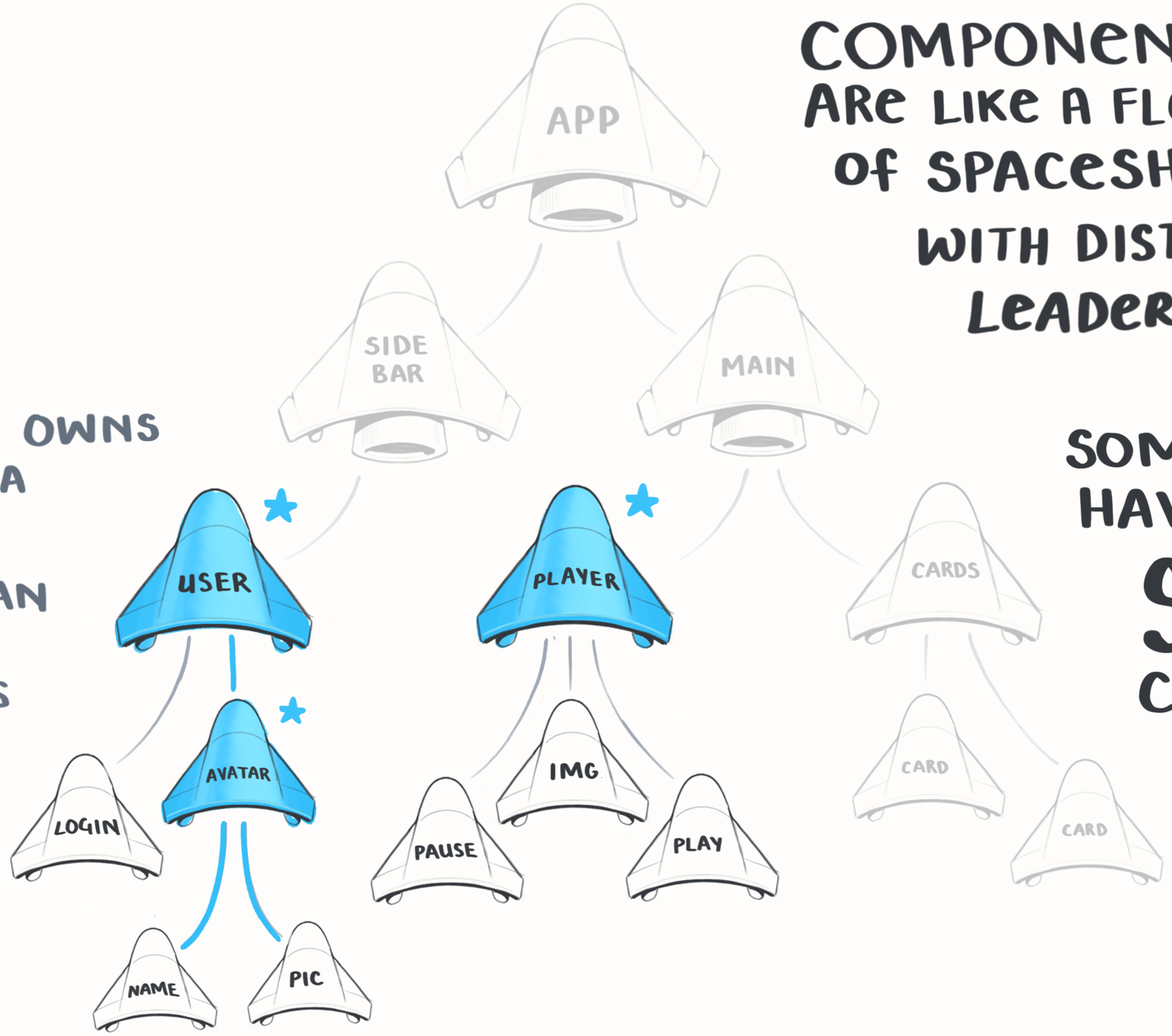
A STATE COMMANDER OWNS A SET OF DATA ABOUT THE MISSION & CAN TELL CHILD COMPONENTS TO UPDATE THEIR PROPS IF NEEDED



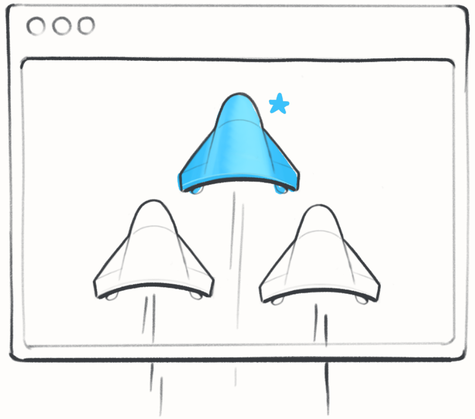
SOME SHIPS HAVE A **STATE** COMMANDER INSIDE

COMPONENTS ARE LIKE A FLEET OF SPACESHIPS WITH DISTRIBUTED LEADERSHIP.

A STATE COMMANDER OWNS A SET OF DATA ABOUT THE MISSION & CAN TELL CHILD COMPONENTS TO UPDATE THEIR PROPS IF NEEDED

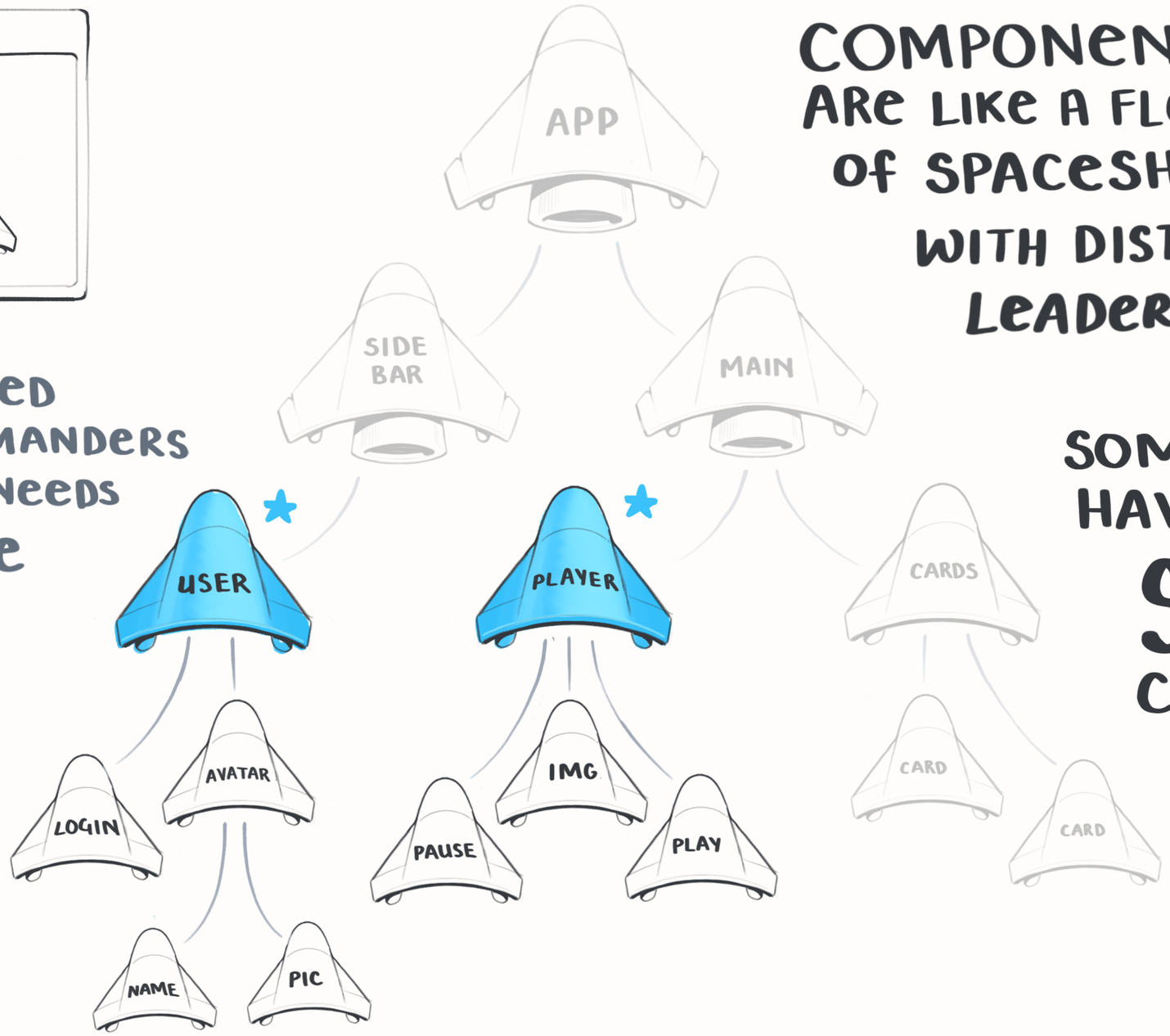


SOME SHIPS HAVE A **STATE** COMMANDER INSIDE



WE ONLY NEED STATE COMMANDERS IF OUR APP NEEDS TO *change* AFTER LAUNCH

eg. AFTER IT'S ALREADY ON THE DOM



COMPONENTS ARE LIKE A FLEET OF SPACESHIPS WITH DISTRIBUTED LEADERSHIP

SOME SHIPS HAVE A **STATE** COMMANDER INSIDE

Forget React

Let's talk about Visual Metaphor

And why programming needs it.

Here's the problem

Programming is always explained through walls of text

Highly technical jargon-filled text

Documentation is a Visual Desert

Limiting the medium means fewer people can access the information

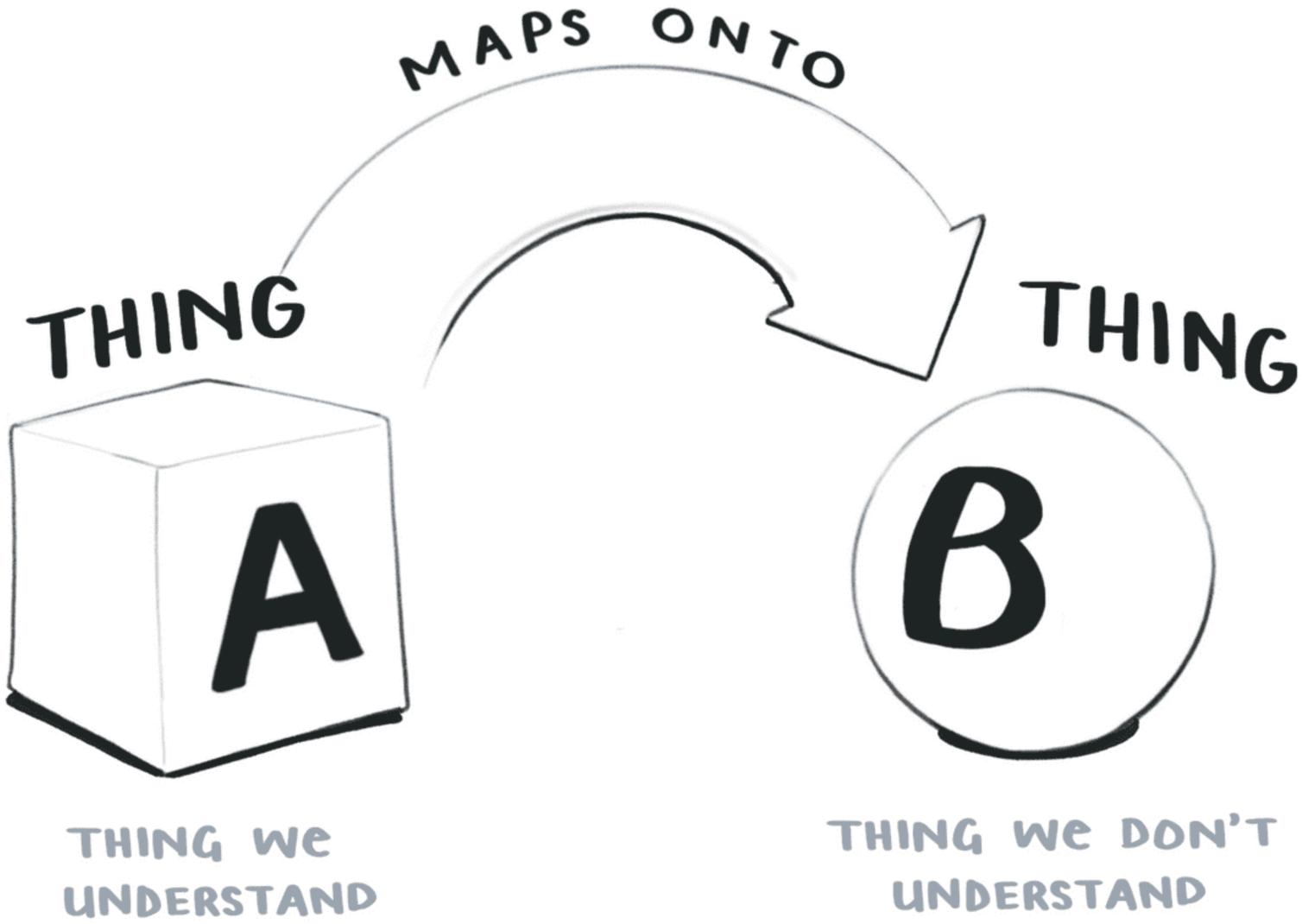
Our current ways of teaching limit who gets to join in

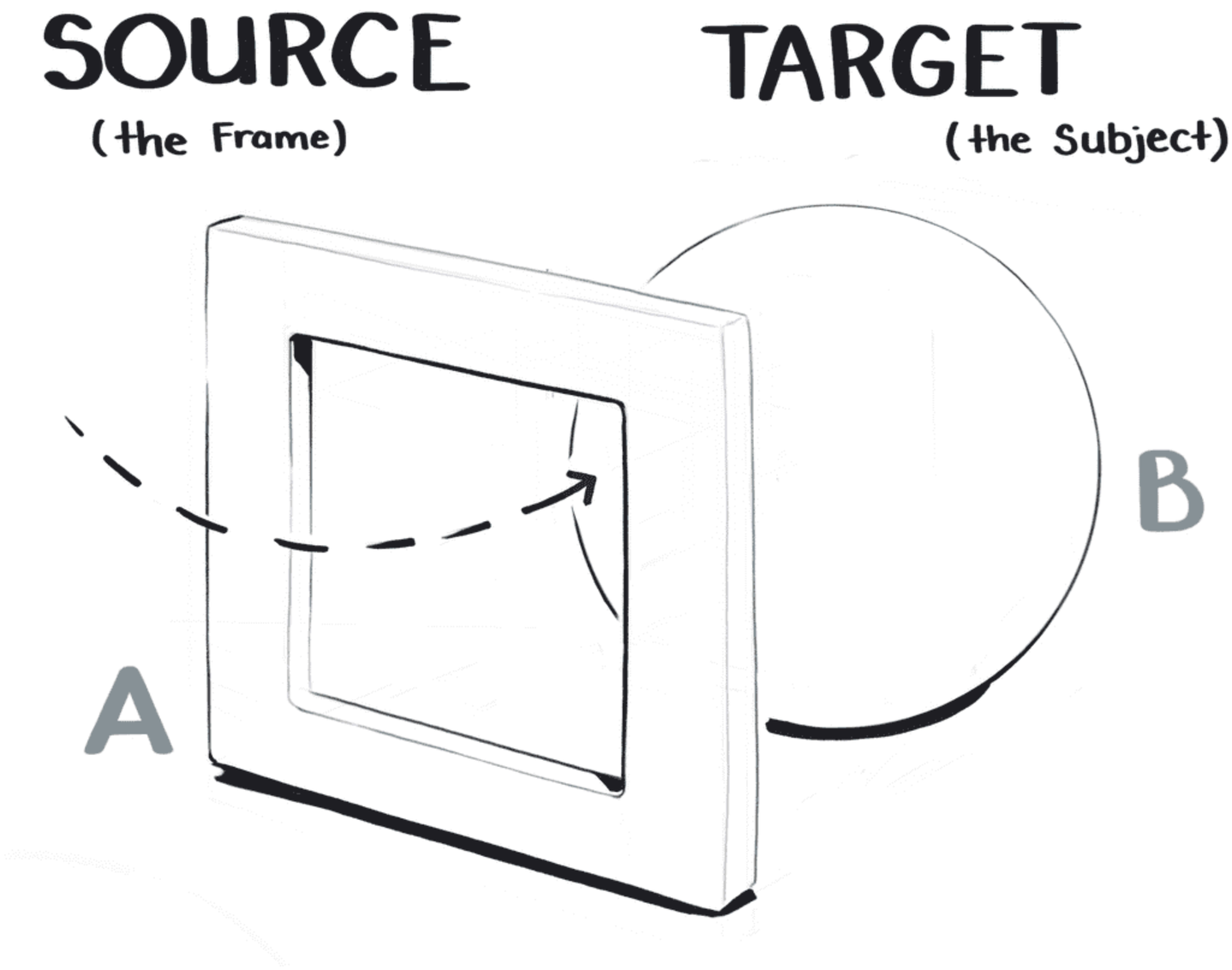
Metaphors will help lower the fortress walls

Bring more people in, make it more welcoming and accessible

What's a metaphor?

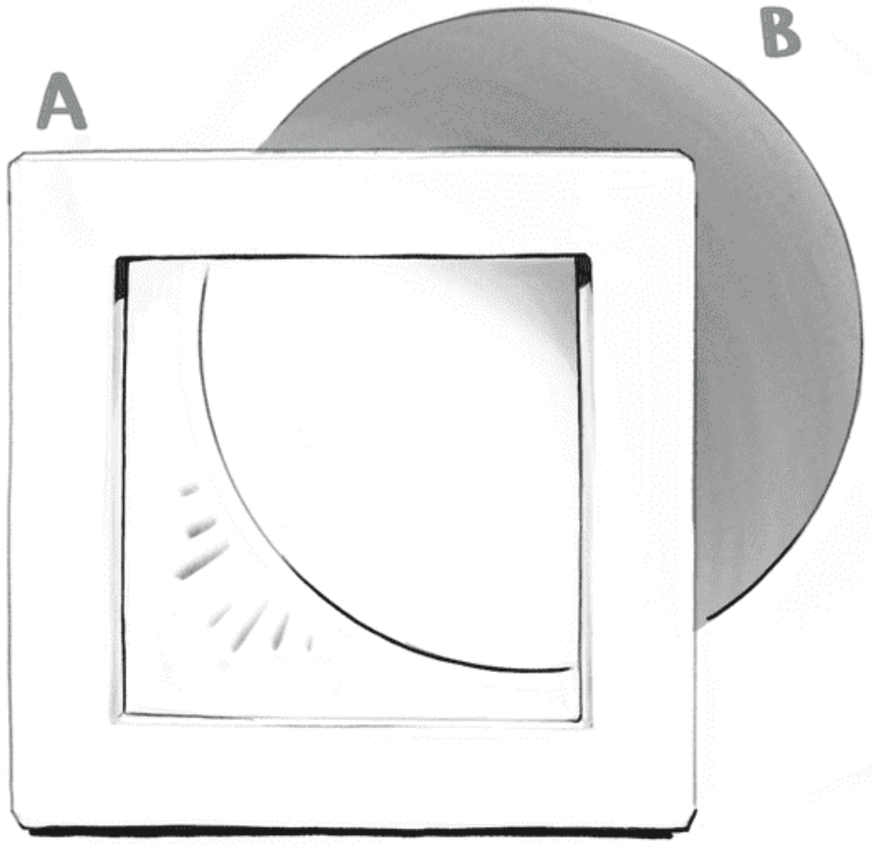
When we understand one thing in terms of another.


MAPS ONTO
THING
A
THING
B
THING WE UNDERSTAND
THING WE DON'T UNDERSTAND


SOURCE
(the Frame)
TARGET
(the Subject)
A
B

HIGH LIGHTS

THE SHARED QUALITIES



HIDES

QUALITIES THAT AREN'T SHARED

Aren't metaphors just a frivolous distraction??

This is programming, not kindergarten story-time!

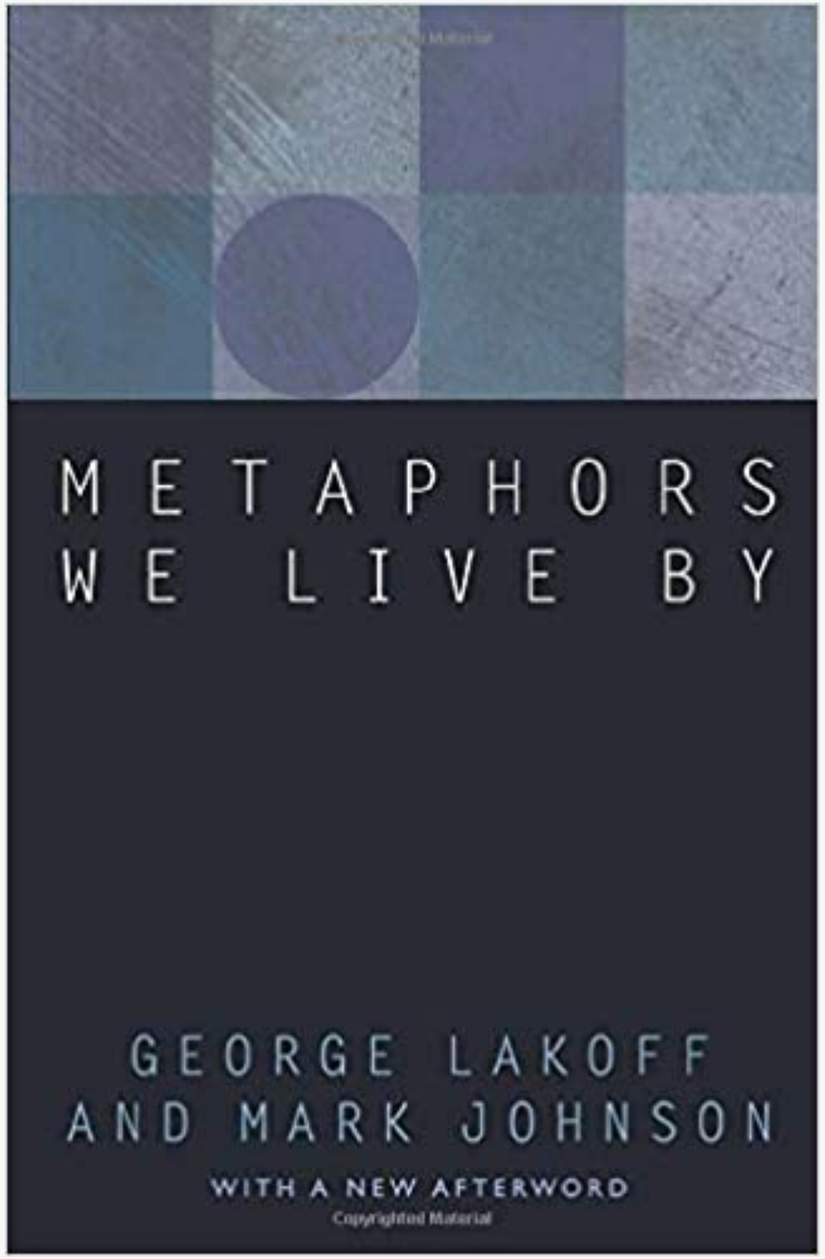
You're thinking of "figurative" metaphor

Common in poetry, literature, and mythology. Yes, it can get out of hand.

Cognitive Metaphor

Or "Conceptual" Metaphor

Metaphors that form the basis of all human thought



Metaphors We Live By

George Lakoff & Mark Johnson

Metaphors structure how we think, behave, perceive, and reason about the world.

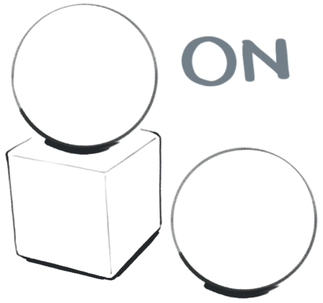
Metaphors make abstract thinking possible

We use physical, embodied terms to speak about abstract ideas

Abstract	Physical
Ideas	Physical Space
Emotions	Force / Motion
Experiences	Touch
Personalities	Taste
Programming	Vision / Light

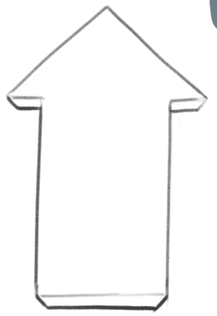
EMBODIED METAPHORS

emerge from spatial & physical experience



ON

OFF

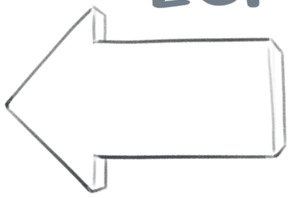


UP

BEHIND

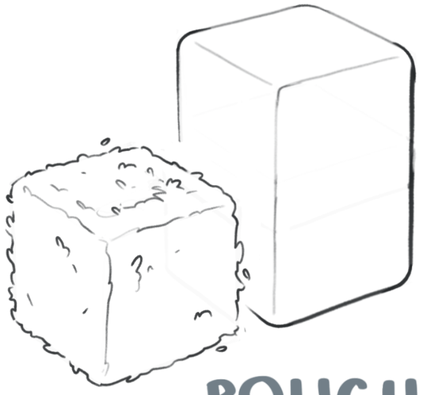
IN FRONT

LEFT

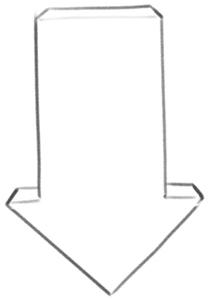


RIGHT

SMOOTH



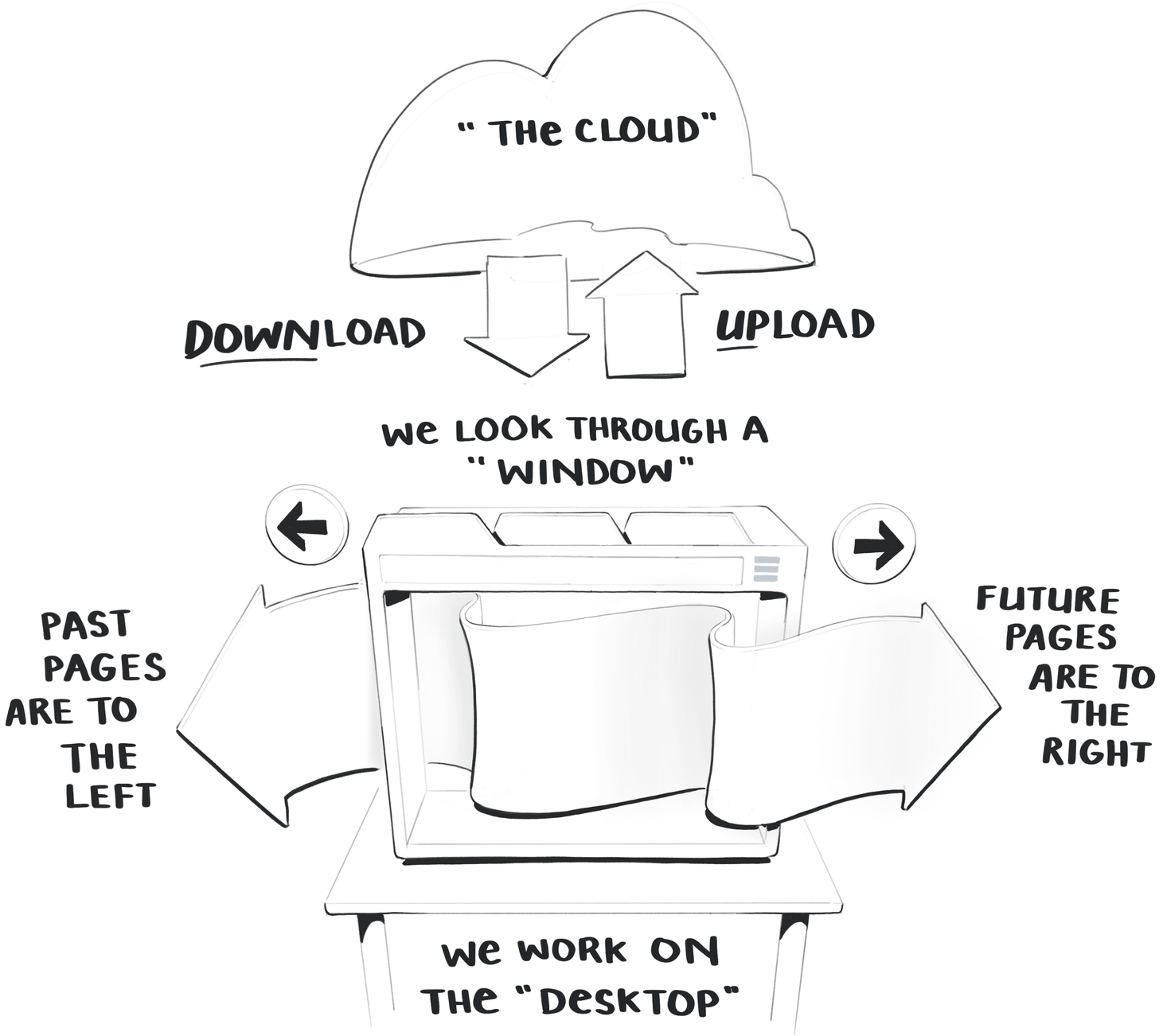
ROUGH



DOWN

SMALL

BIG


" THE CLOUD"
DOWNLOAD
UPLOAD
WE LOOK THROUGH A " WINDOW"
PAST PAGES ARE TO THE LEFT
FUTURE PAGES ARE TO THE RIGHT
WE WORK ON THE "DESKTOP"

And surprise, surprise...

Cognitive Metaphor is fundamental to programming

TLDR

You think in metaphor all day, every day.

You are already a metaphor maker

Metaphors love visuals

Visuals are the best way to communicate metaphor.

They make the implicit explicit.

How to Make Programming a Visual, Metaphorical Wonderland

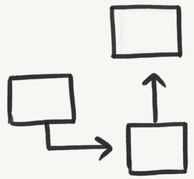
- *Contribute diagrams to documentation*
- *Draw sketches into blog posts*
- *Make educational animations*

"But I can't draw"

Good news!

It's not just about drawing...

Ways to Make Visuals



Diagrams

Emoji

Photography

Animation

Icons

Stick figures,
boxes, and arrows

Visual Tools

Figma

Miro

Whimsical

Excalidraw

Procreate

Keynote

Powerpoint

CSS, SVG or Greensock

illustrated.dev/resources

Resources

Want to improve your drawing and visual thinking skills?
This is a collection of my personal recommendations for online courses and books.

Learning Platforms & Courses

Schoolism

New Masters Academy

Society of Visual Storytelling

But wait...

How do I actually make metaphors?

The cheatsheet for

Designing Visual Metaphors

1

Identify the nouns and verbs

What things and actions do you need to explain?

Nouns

Components

Props

Data

Events

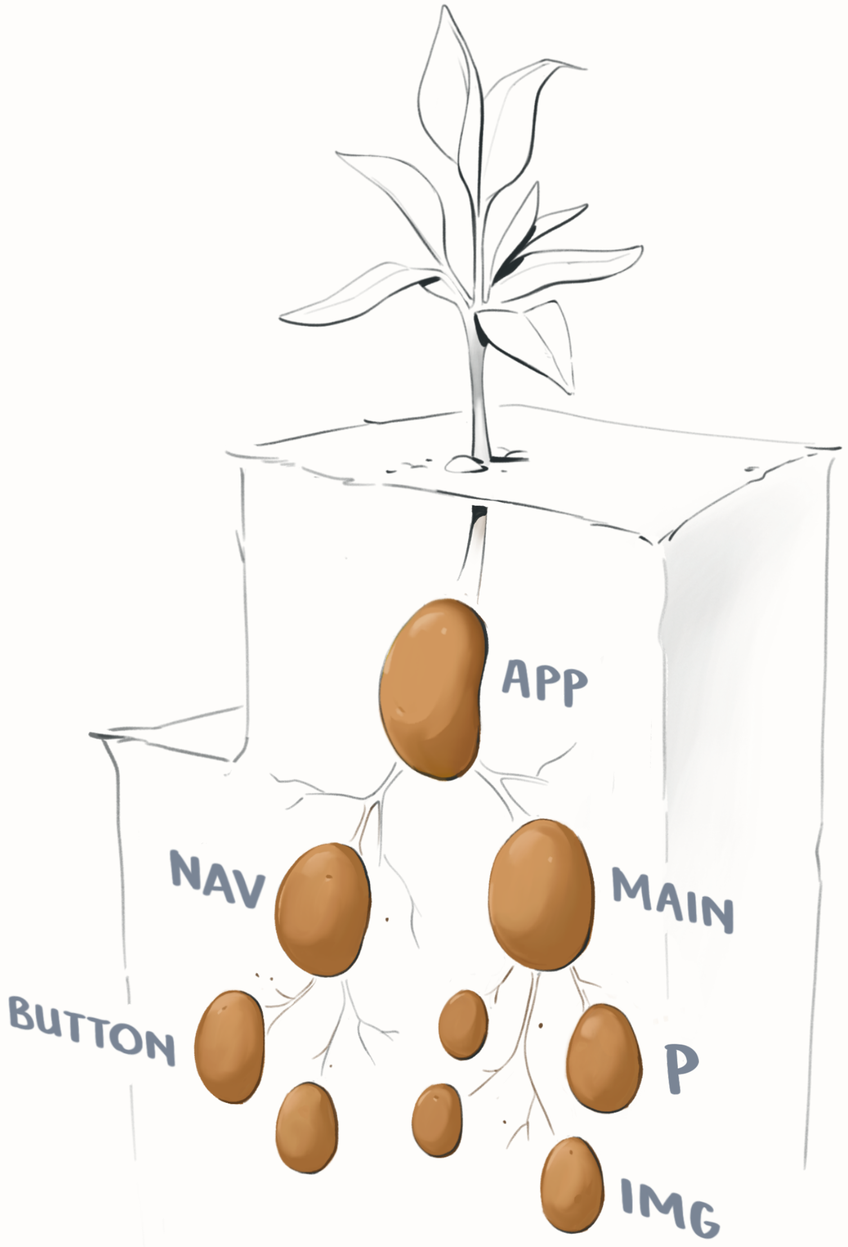
Verbs

Passing data

Passing events

Running a function

Updating data

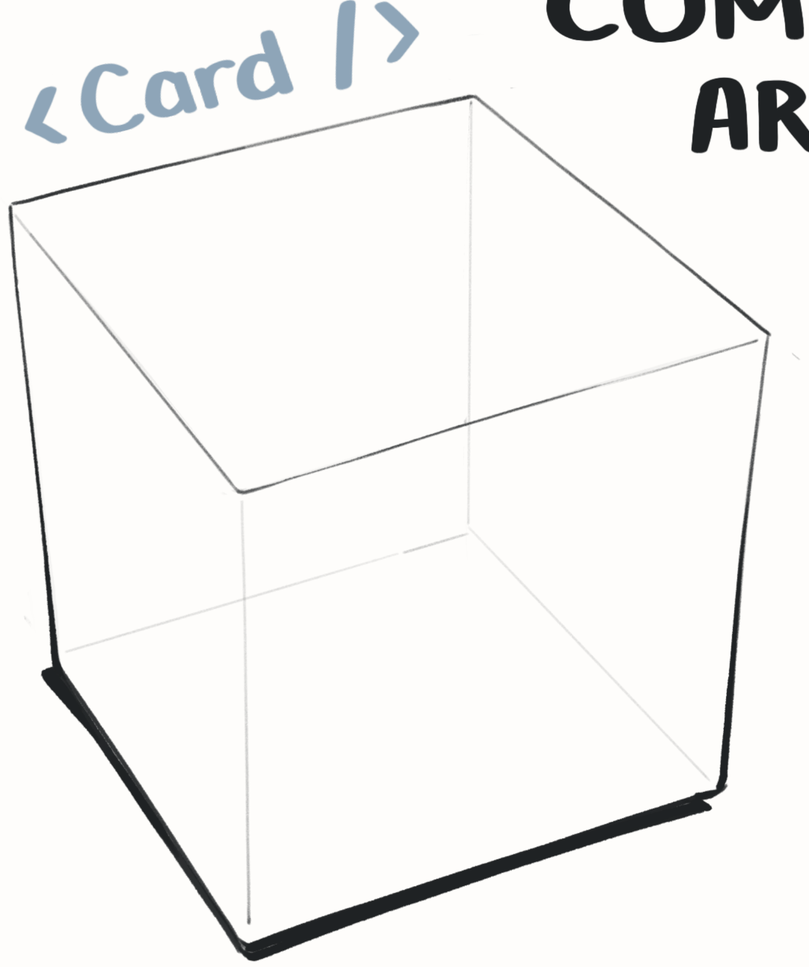


2

State it in simple, high-level terms

Look for physical properties and categories

COMPONENTS ARE CONTAINERS



3

Focus on key functions and qualities to highlight

What's important and what's not?

In JSX we swirl a bunch of

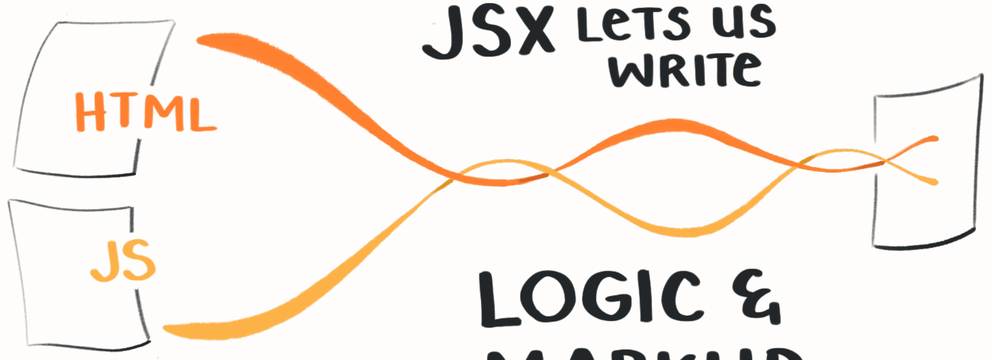
MARK UP

(THE SYNTAX WE WRITE HTML IN)

INTO OUR VANILLA JS ICE CREAM

```
function Menu(){
  let flavour = "JSX Swirl"
  return(
    <div>
      <h1> {flavour} </h1>
    </div>
})
```

INSTEAD OF HAVING TWO SEPARATE FILES,



JSX LETS US WRITE

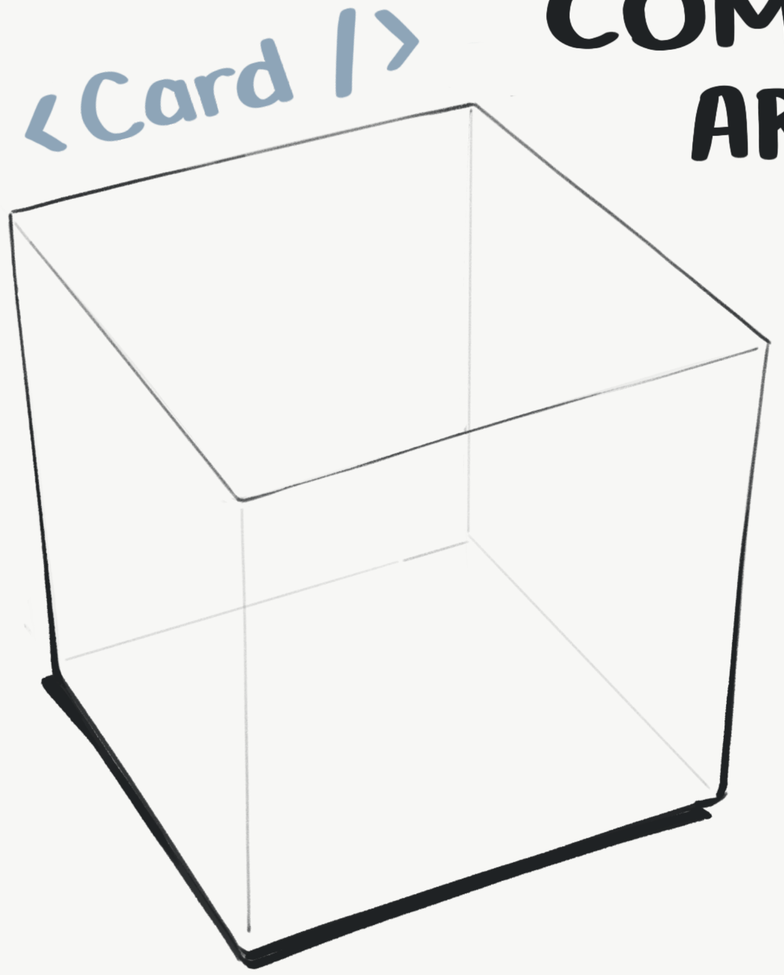
LOGIC & MARKUP SIDE-BY-SIDE

4

Think laterally to find alternatives

Look at unrelated disciplines

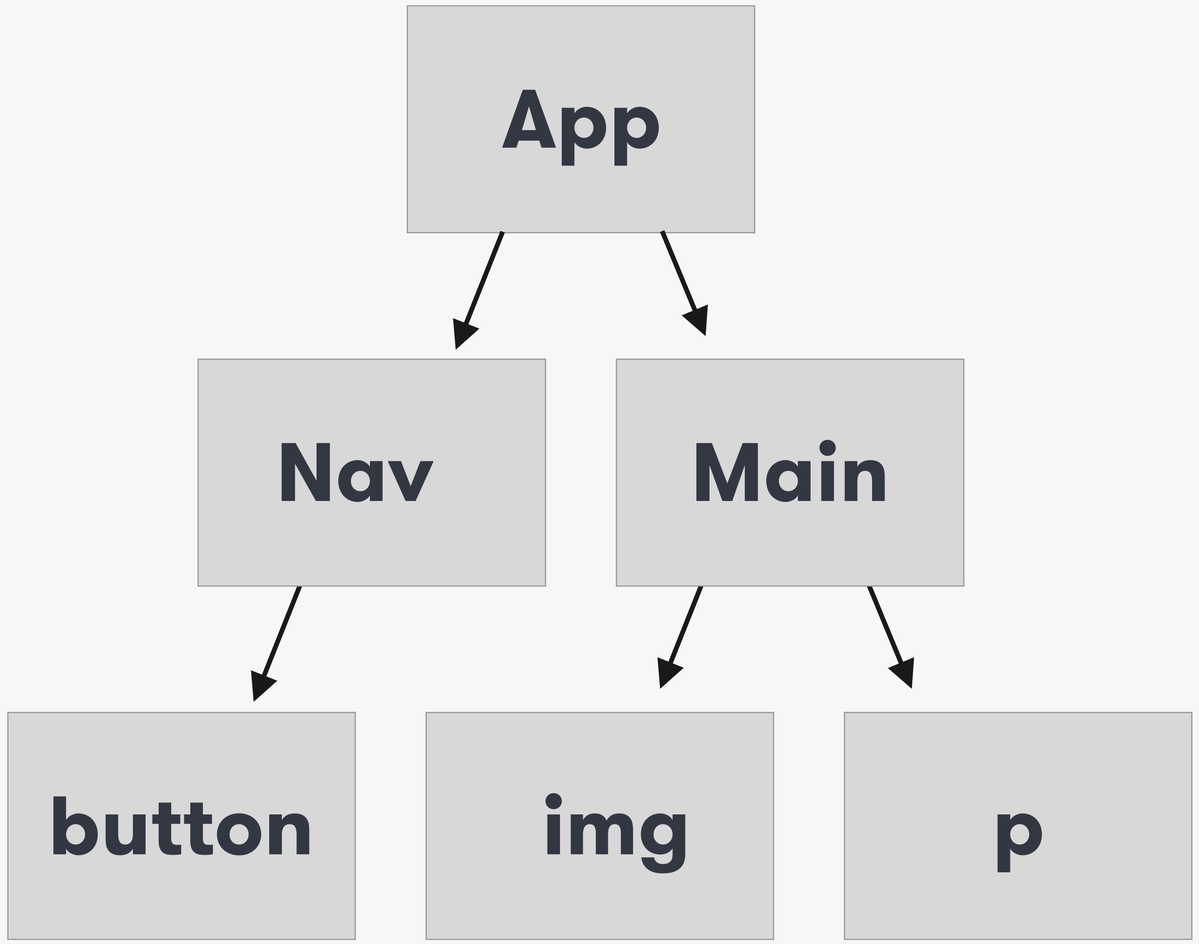
COMPONENTS ARE CONTAINERS

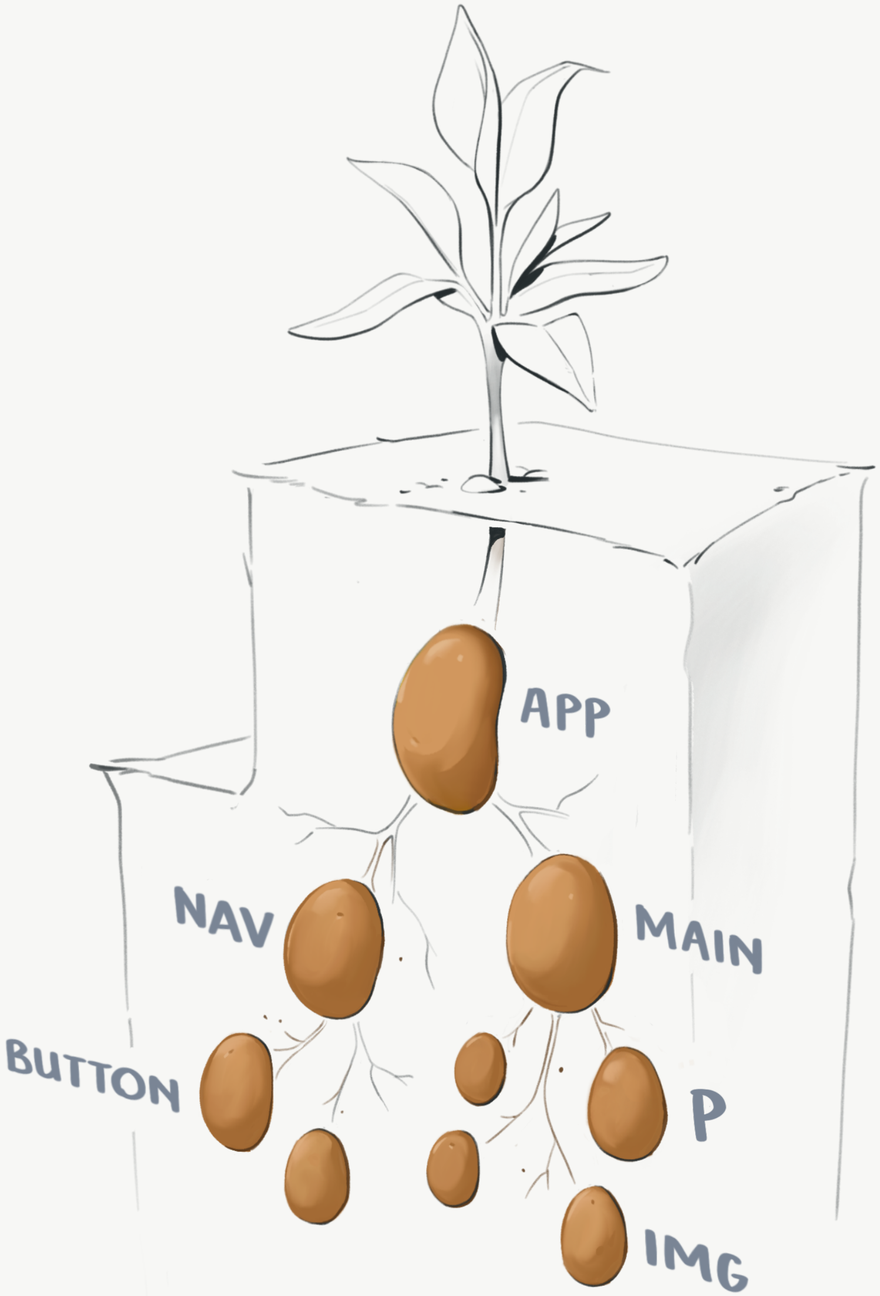


5

Add in specific storytelling elements

Speak human, not computer.


App
Nav
Main
button
img
p


APP
NAV
MAIN
BUTTON
P
IMG

Building Metaphors

1. *Find your nouns and verbs*
2. *State one main concept in simple terms*
3. *Pick functions and key qualities to highlight*
4. *Think laterally to find alternatives*
5. *Add in storytelling*

"Metaphor is just a conduit for the transportation of ideas between minds."

Tim Rohrer

Metaphors We Compute By

Go forth and

Make Metaphors

maggieappleton.com
illustrated.dev

Thank you!