Efficient Micro Frontend Development

with Webpack Module Federation

Caitlin Collins & Mike Eastes

OSN 2022

Who we are

Mike Eastes

Caitlin Collins

code42

Video Player

Share

Contact Us

Send

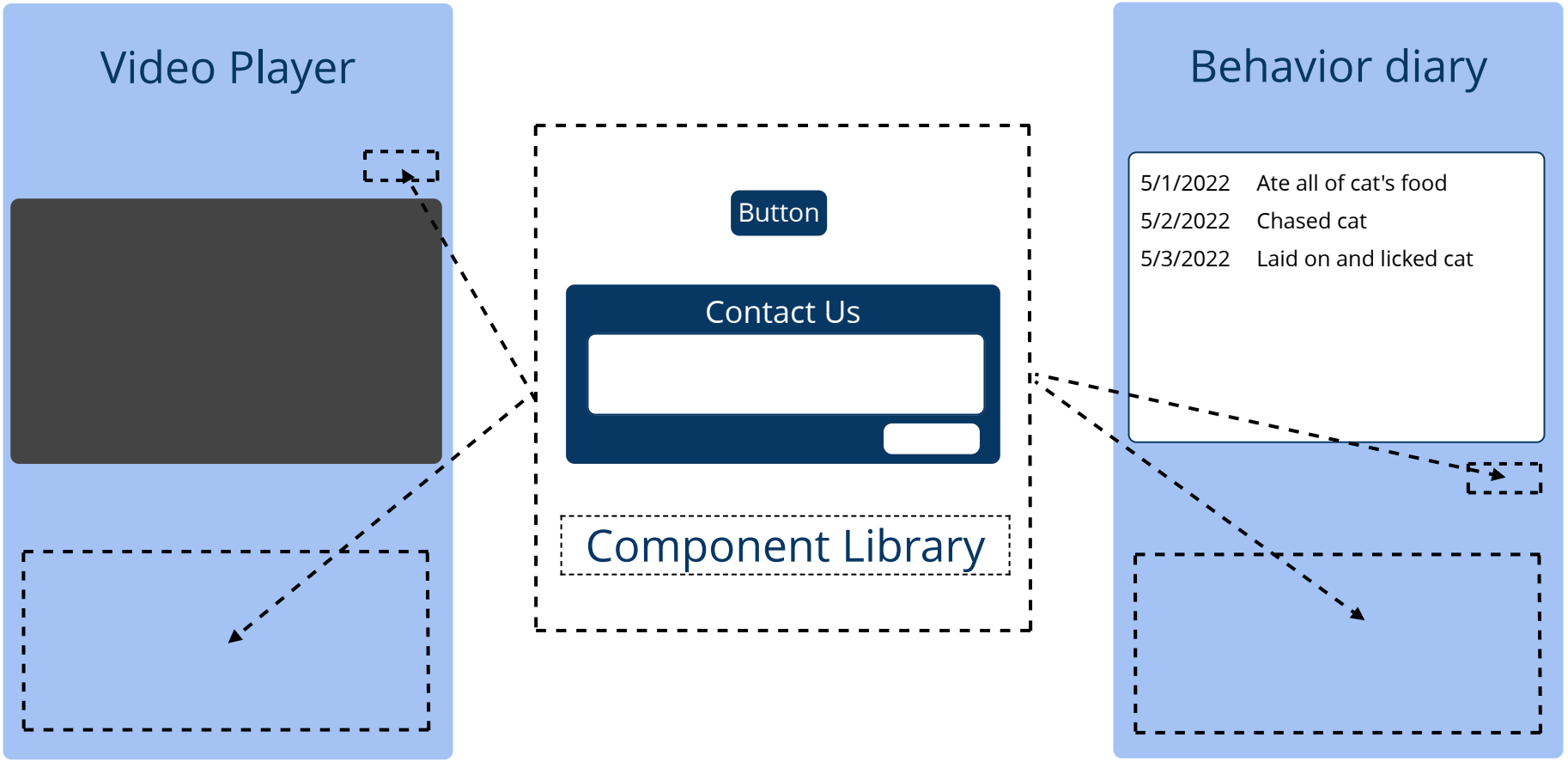
Behavior diary

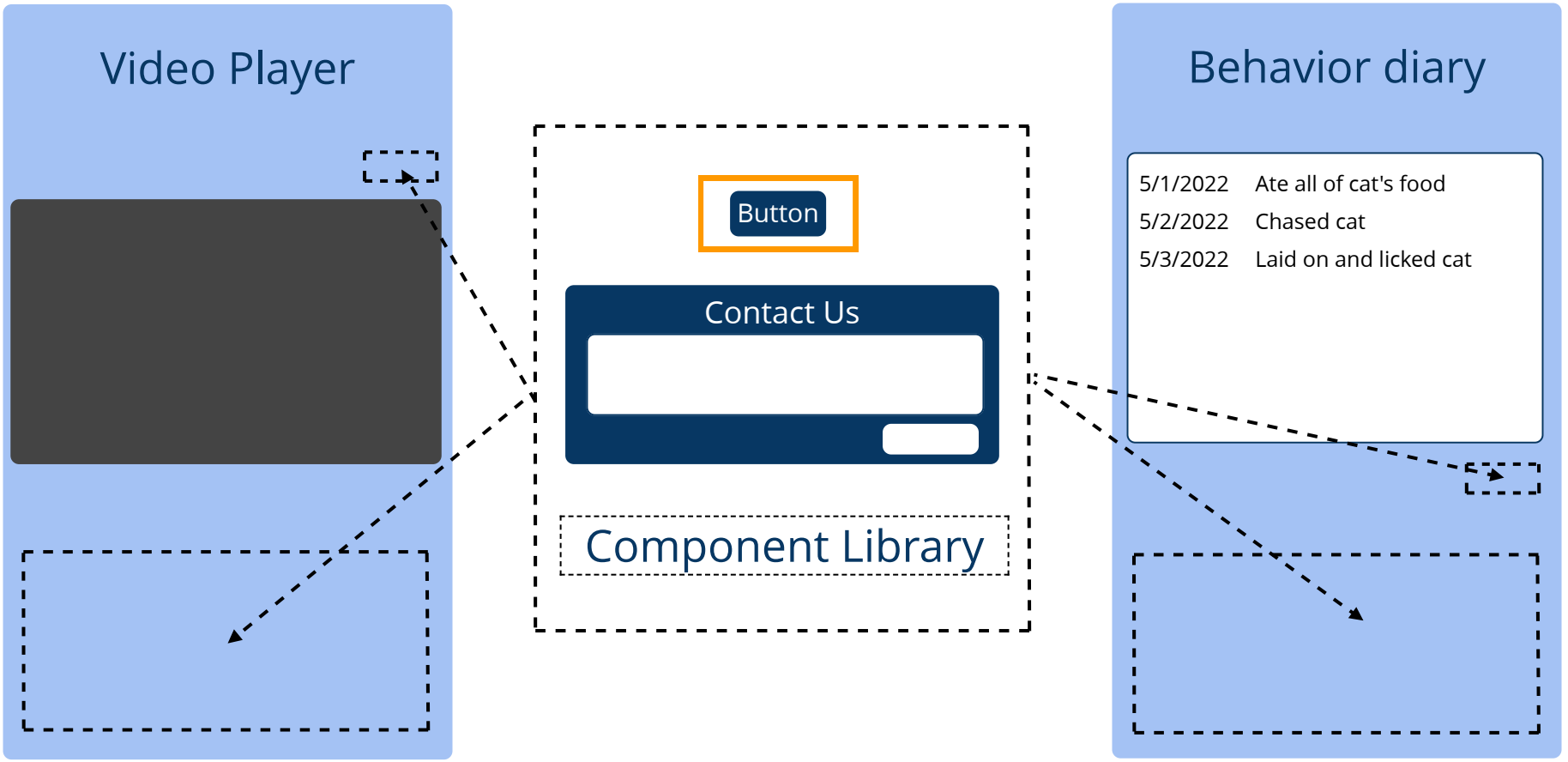
5/1/2022	Ate all of cat's food
5/2/2022	Chased cat
5/3/2022	Laid on and licked cat

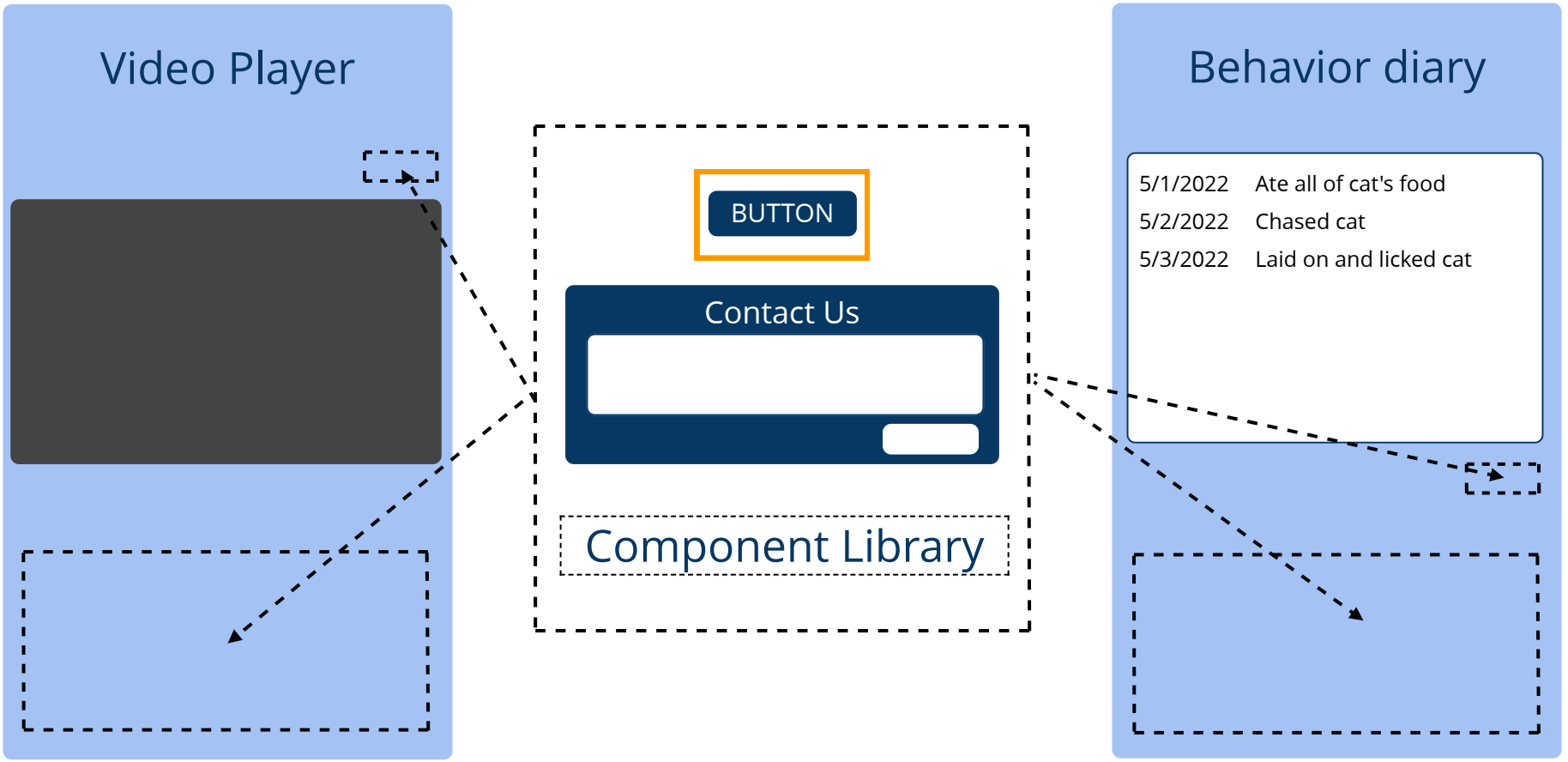
Save

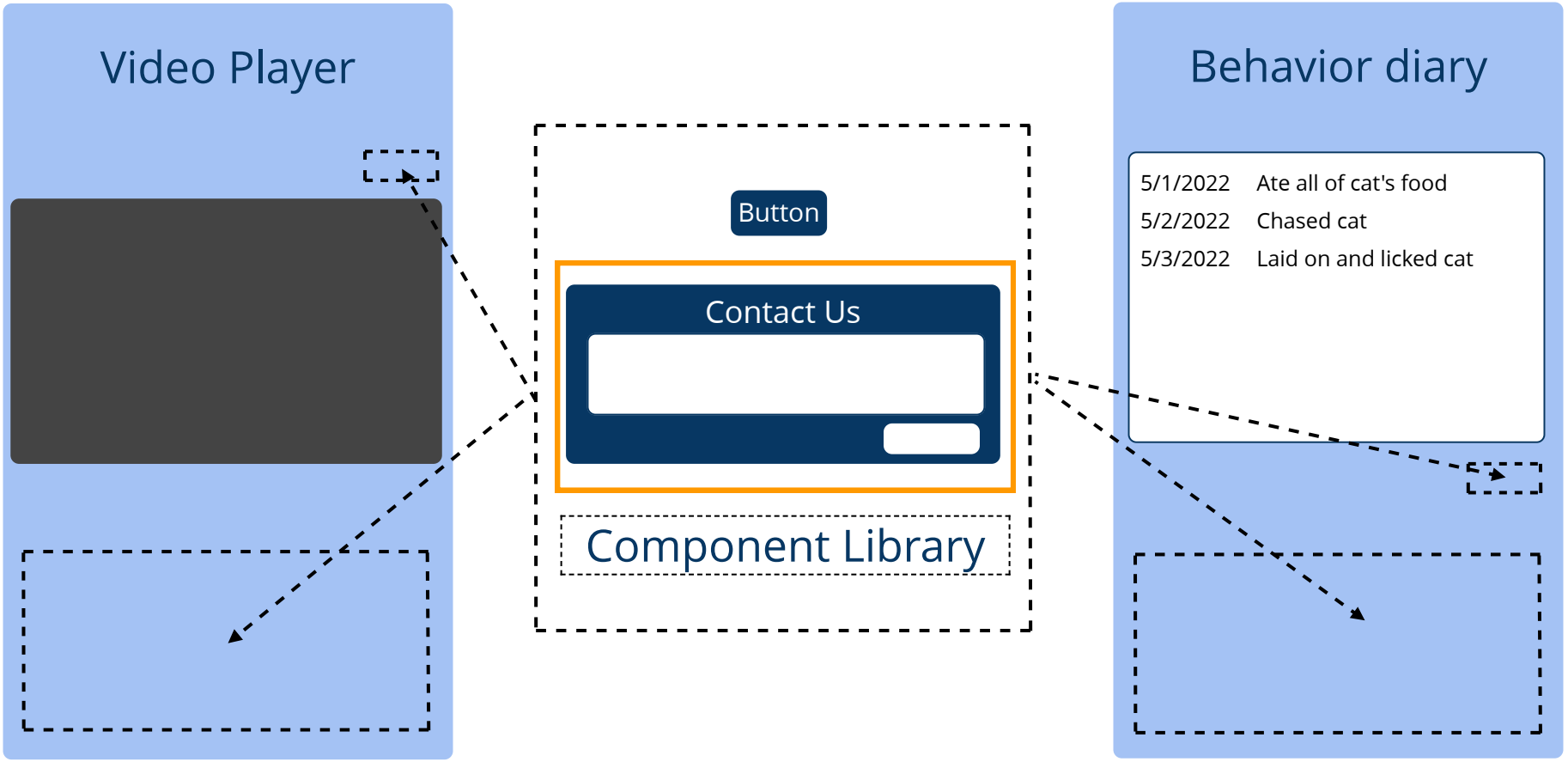
Contact Us

Send


Video Player
Button
Contact Us
Component Library
Behavior diary
5/1/2022 Ate all of cat's food
5/2/2022 Chased cat
5/3/2022 Laid on and licked cat


Video Player
Behavior diary
5/1/2022 Ate all of cat's food
5/2/2022 Chased cat
5/3/2022 Laid on and licked cat
Button
Contact Us
Component Library


Video Player
BUTTON
Contact Us
Component Library
Behavior diary
5/1/2022 Ate all of cat's food
5/2/2022 Chased cat
5/3/2022 Laid on and licked cat


Video Player
Button
Contact Us
Component Library
Behavior diary
5/1/2022 Ate all of cat's food
5/2/2022 Chased cat
5/3/2022 Laid on and licked cat

What is Webpack Module Federation?

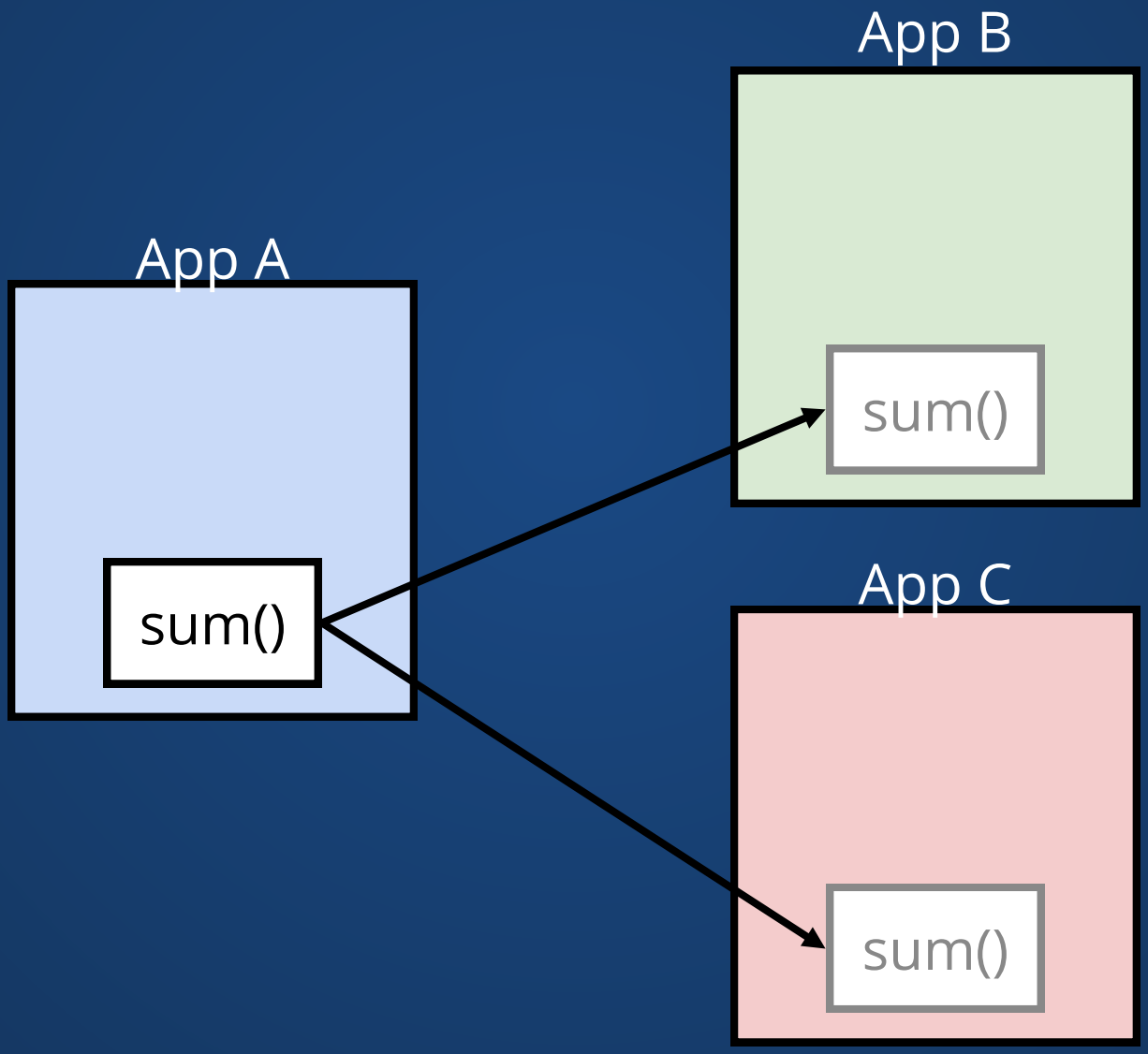
Webpack module federation

- Is a feature built-in to Webpack 5
- Allows apps to share code with each other **at runtime**
- Loads shared code asynchronously

Let's dive in

Our shared function

How it works



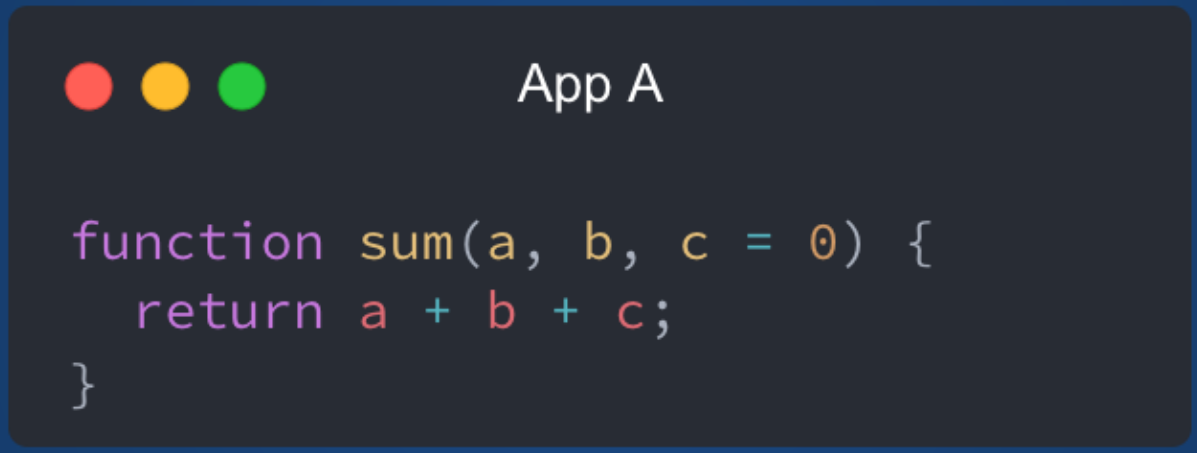
Consuming the function

App B

```
const sum = import('appA/sum');

sum(2, 2); // 4
```

What happens if the function changes?

Updated function



Consuming updates

App B

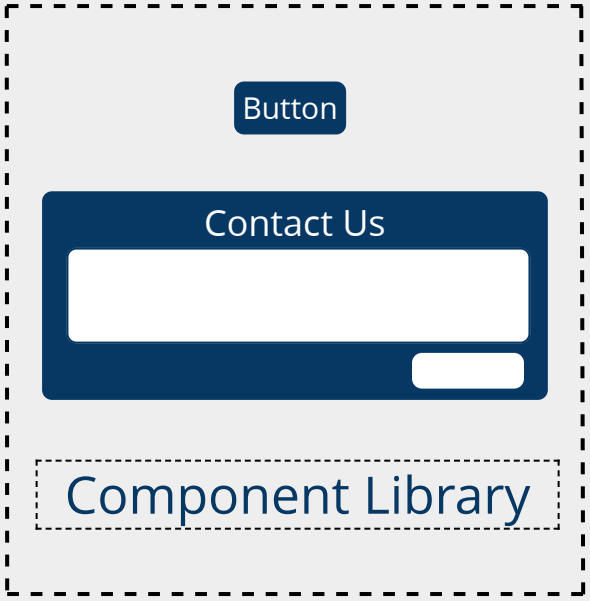
```
const sum = import('appA/sum');

sum(2, 2); // 4 - Still works!
```

Consuming updates

App B

```
const sum = import('appA/sum');

sum(2, 2); // 4 - Still works!
sum(2, 2, 2); // 6 - New functionality available immediately
```

Let's talk about UI components


Button
Contact Us
Component Library

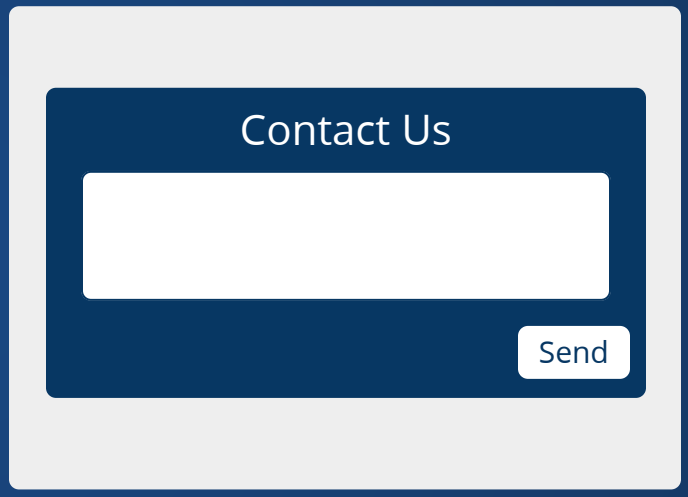
Component Library Challenges:

Cross-team coordination is hard.

Component Library Challenges:

Complex components slow down development

Webpack 5



Video Player

Share

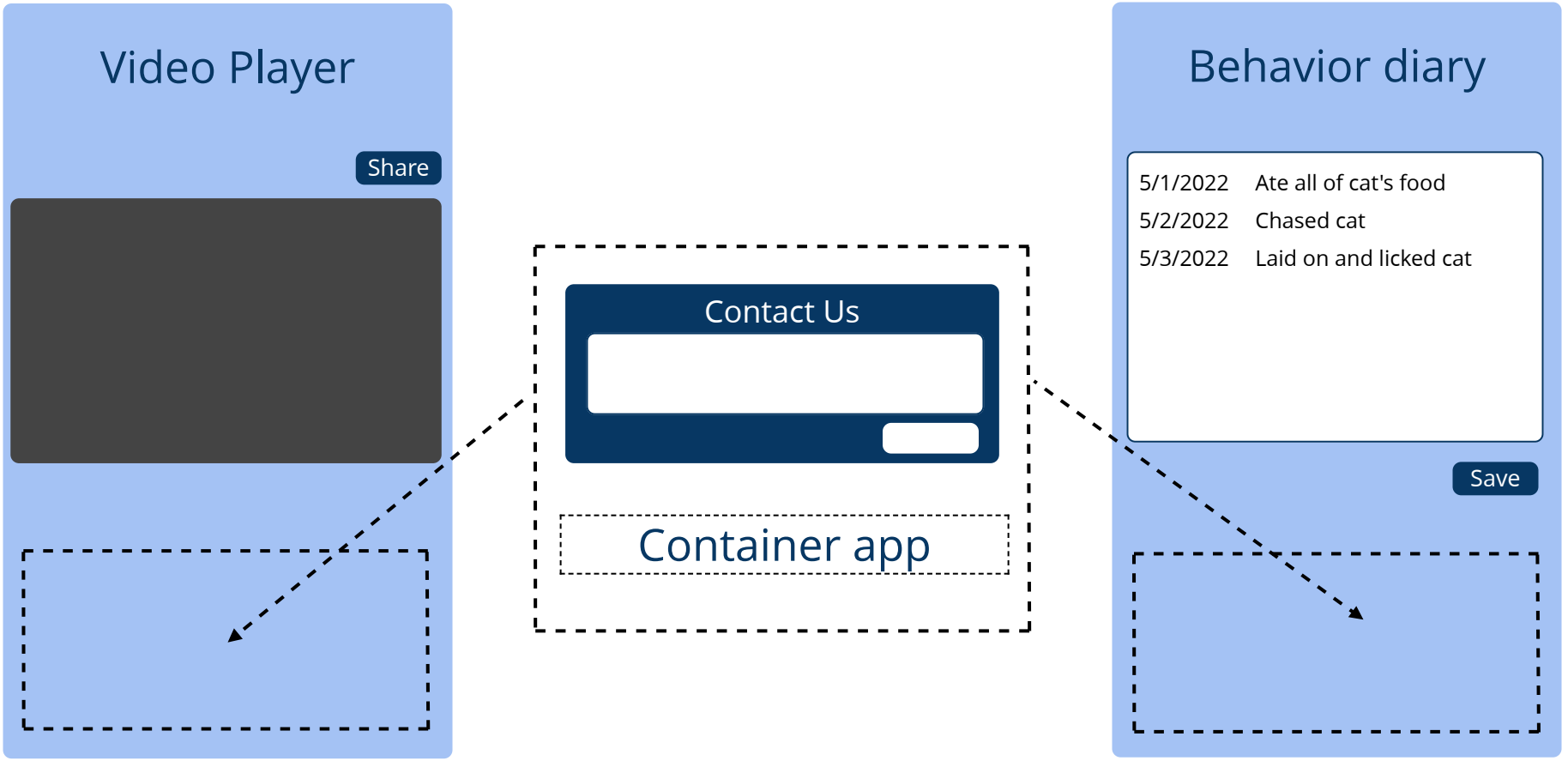
Contact Us

Send

Behavior diary

5/1/2022	Ate all of cat's food
5/2/2022	Chased cat
5/3/2022	Laid on and licked cat

Save


Video Player
Share
Contact Us
Container app
Behavior diary
5/1/2022 Ate all of cat's food
5/2/2022 Chased cat
5/3/2022 Laid on and licked cat
Save

Container app

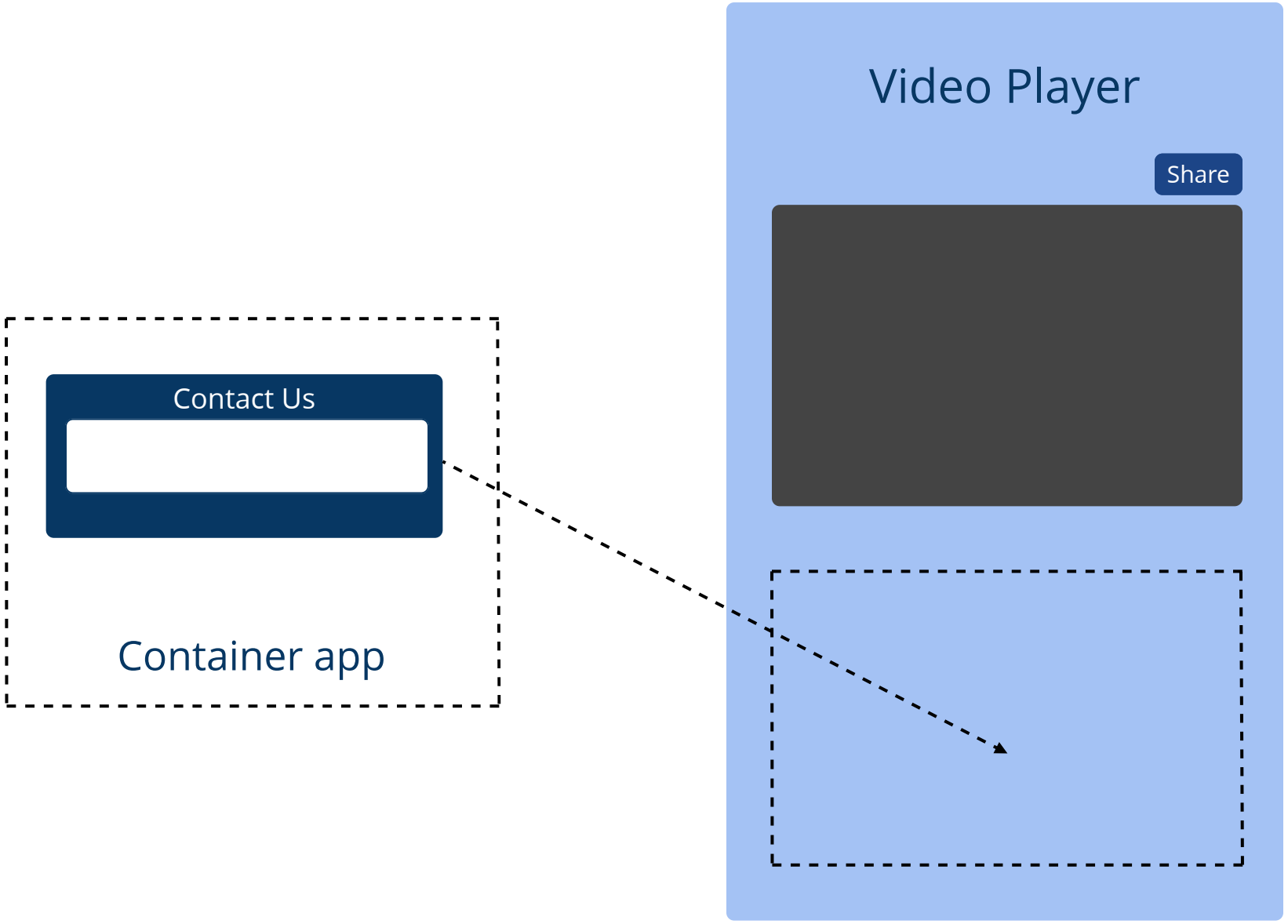
```
const { ModuleFederationPlugin } = require('webpack').container;

module.exports = {
  ...
  plugins: [
    new ModuleFederationPlugin({
      name: 'containerApp',
      filename: 'remoteEntry.js',
      exposes: {
        './ContactUs': './src/ContactUs',
      },
      shared: { react: { singleton: true }, 'react-dom': { singleton: true } },
    }),
    ...
  ],
};
```

Consuming app

```
const { ModuleFederationPlugin } = require('webpack').container;

module.exports = {
  ...
  plugins: [
    new ModuleFederationPlugin({
      name: 'videoPlayer',
      remotes: {
        containerApp: `"containerApp@[url]/remoteEntry.js"`,
      },
      shared: { react: { singleton: true }, 'react-dom': { singleton: true } },
    }),
    ...
  ],
};
```

Consuming app

```
import React from "react";

const RemoteContactUs = React.lazy(() => import("containerApp/ContactUs"));

const App = () => (
  <div>
    <h1>VideoPlayer</h1>
    ...
    <React.Suspense fallback={<div>Loading...</div>}>
      <ContactUs />
    </React.Suspense>
  </div>
);

export default App;
```


Video Player
Share
Contact Us
Container app

Challenges of Module Federation

Breaking changes

When making a breaking change

- In a library, increase the major version (1.0 -> 2.0)
- In a federated module...?

Federated modules are not versioned!

Our shared component

UserCard.tsx

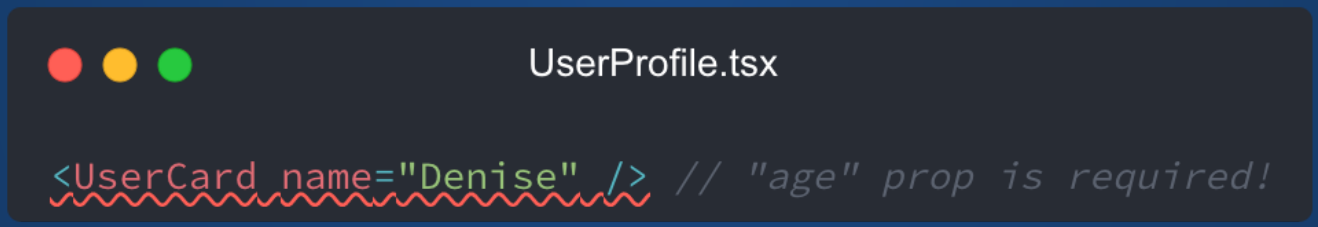
```tsx
interface Props {
  name: string;
}

const UserCard: FC<Props> = ({ name }) => {
  return (
    <Card name={name} />
  );
}
```

Breaking change

UserCard.tsx

```tsx
interface Props {
  name: string;
  age: number;
}

const UserCard: FC<Props> = ({ name, age })
=> {
  return (
    <Card name={name} age={age} />
  );
}
```

In the consuming app



Finding remote components

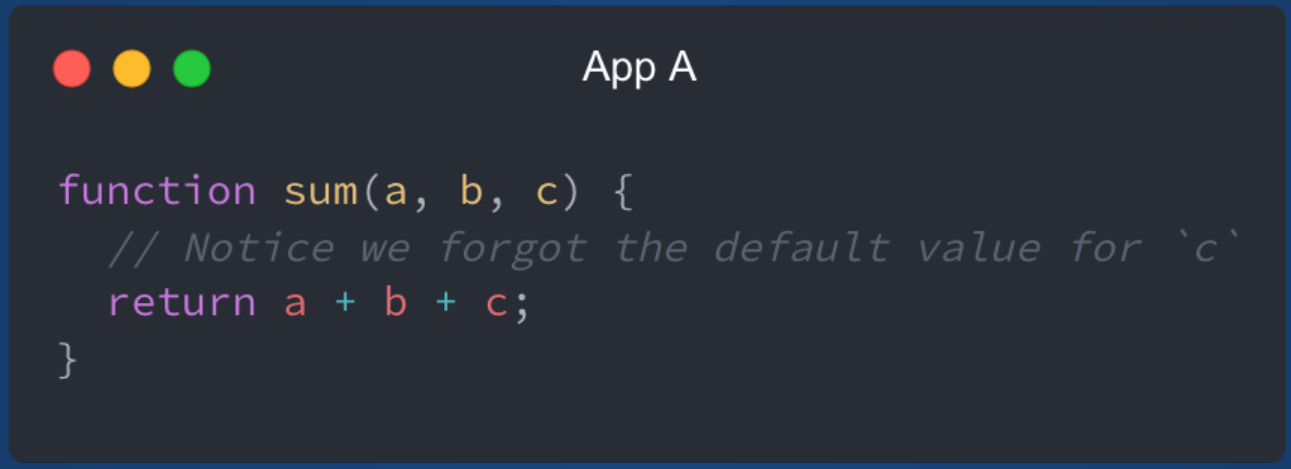
```
const { ModuleFederationPlugin } = require('webpack').container;

module.exports = {
  ...
  plugins: [
    new ModuleFederationPlugin({
      name: 'videoPlayer',
      remotes: {
        containerApp: `"containerApp@[url]/remoteEntry.js"`,
      },
      shared: { react: { singleton: true }, 'react-dom': { singleton: true } },
    }),
    ...
  ],
};
```

High stakes errors

What happens if we introduce a bug?

Back to this function

Update it the wrong way



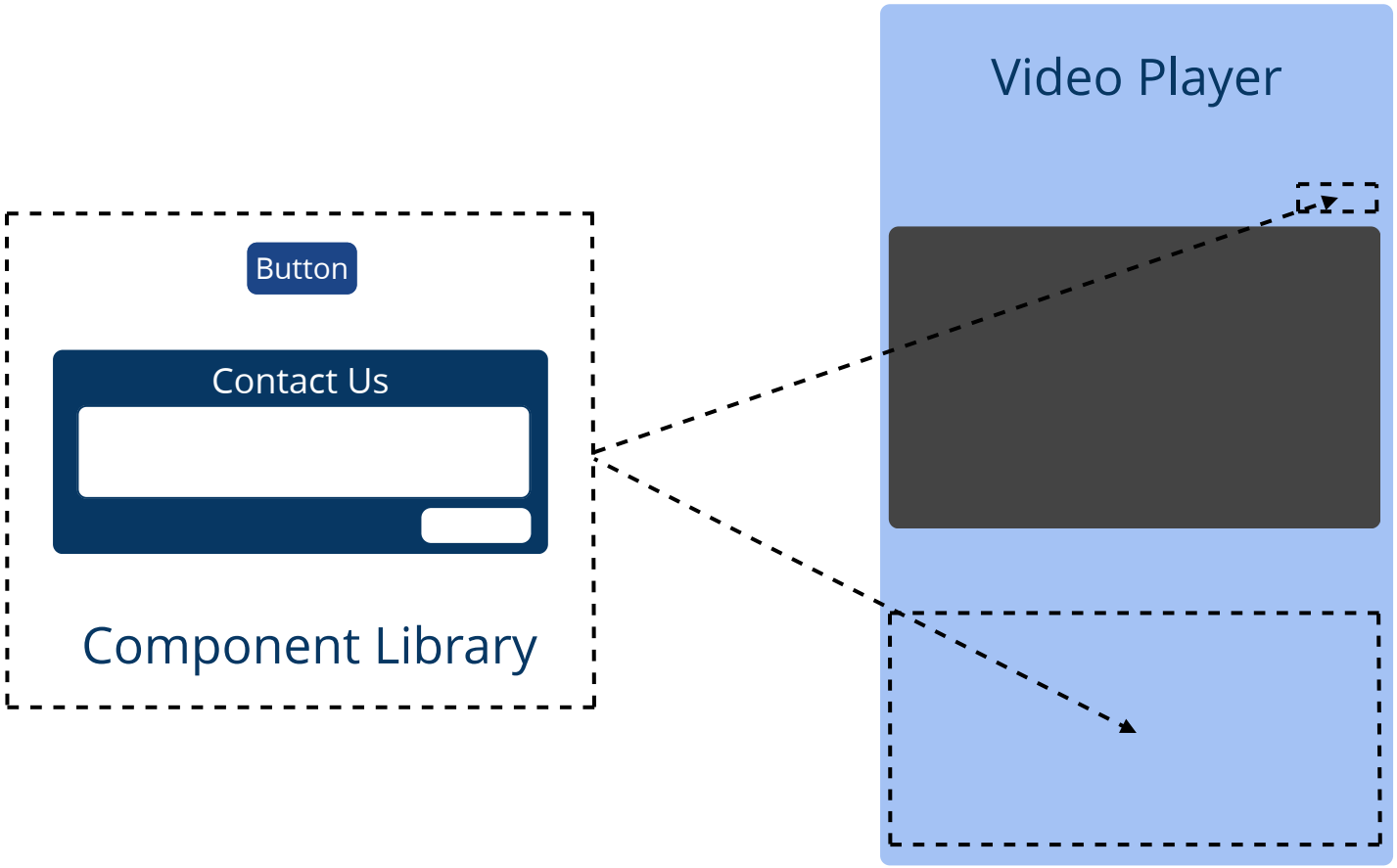
What happens to App B?

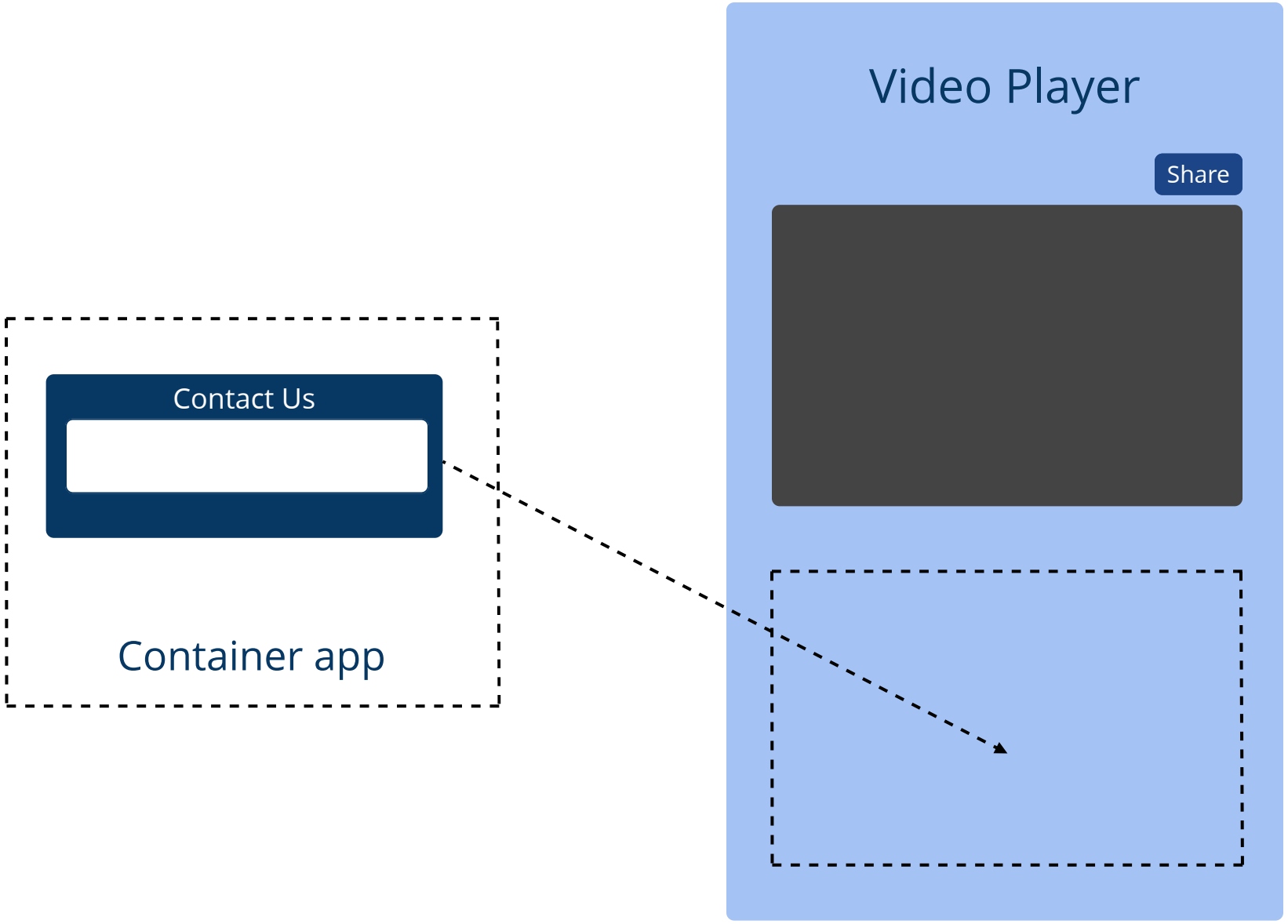
App B

```
const sum = import('appA/sum');

sum(2, 2); // 2 + 2 + undefined = NaN
```

What We Learned


Be Realistic
(about support costs)


Video Player
Button
Contact Us
Component Library


Video Player
Share
Contact Us
Container app


Use API-first design

What does that mean?

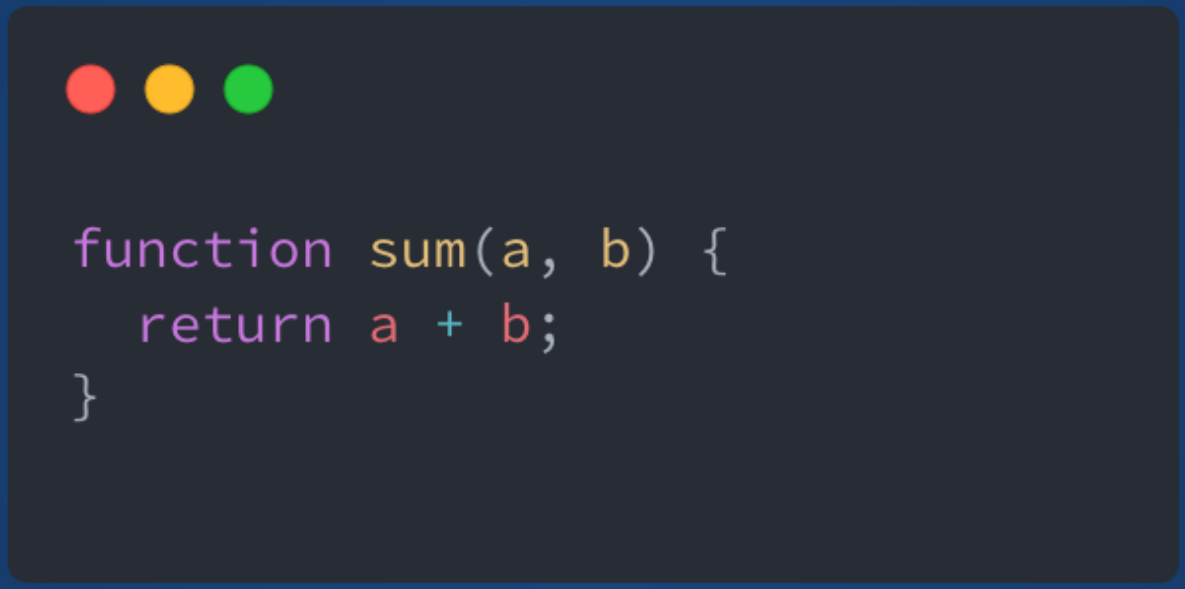
Think of your API as a first-class citizen that will become the center-point of all the changes you make to your component.

When designing an API, consider:

- What features are expected to be added
- What assumptions are you making
- In what ways is your API flexible

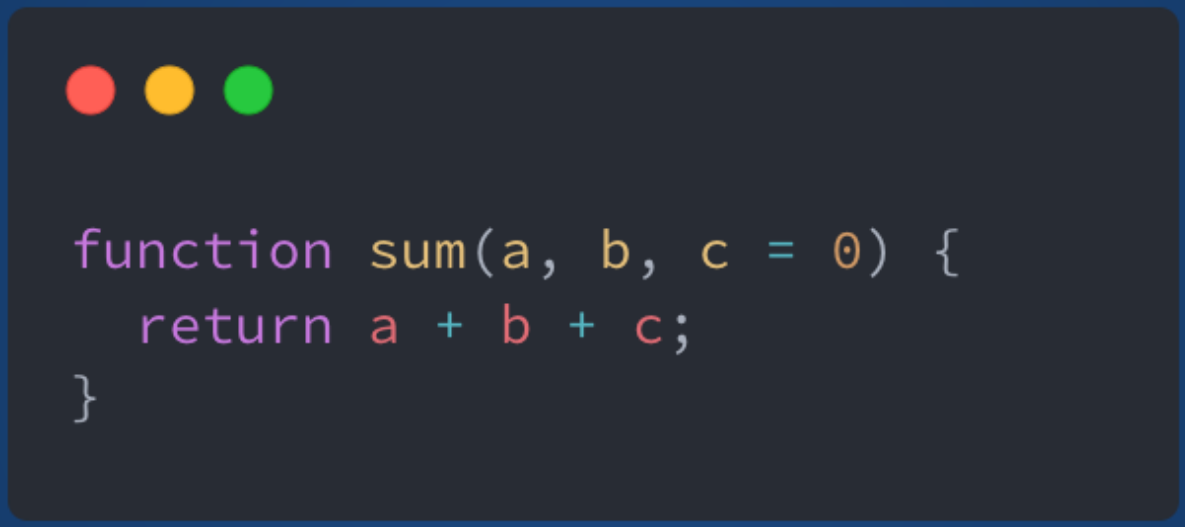
Reveal breaking changes in phases

Option 1: Make breaking changes non-breaking



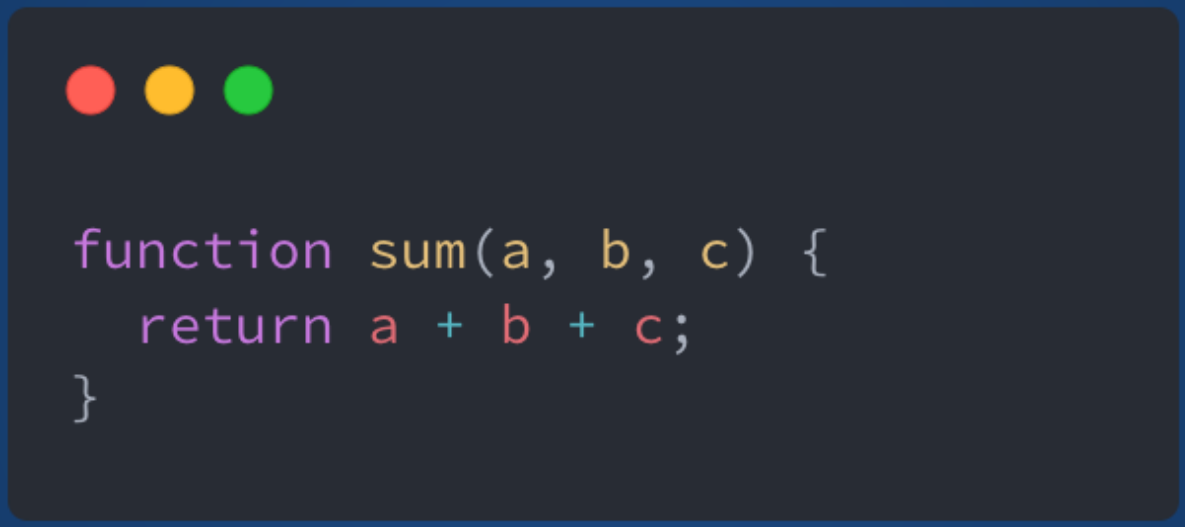
```
function sum(a, b) {
  return a + b;
}
```

Option 1: Make breaking changes non-breaking



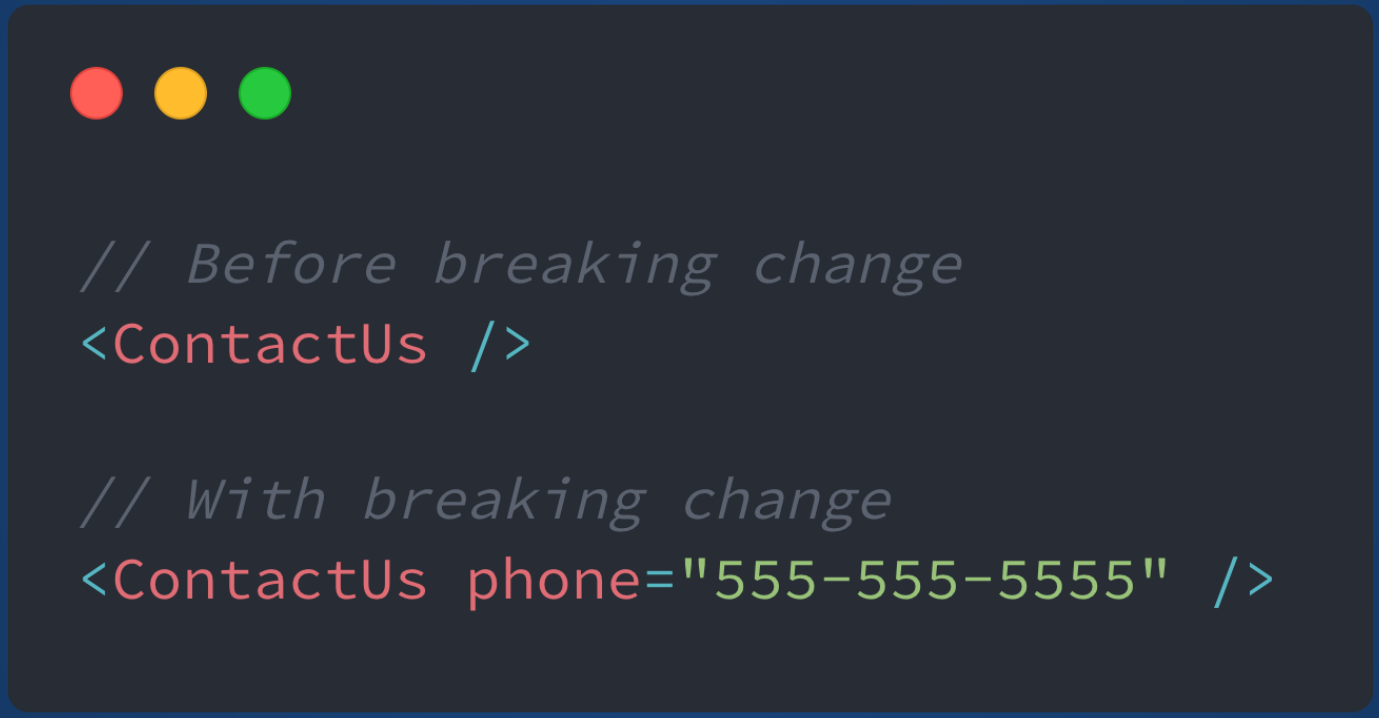
```
function sum(a, b, c = 0) {
  return a + b + c;
}
```

Option 1: Make breaking changes non-breaking



```
function sum(a, b, c) {
  return a + b + c;
}
```

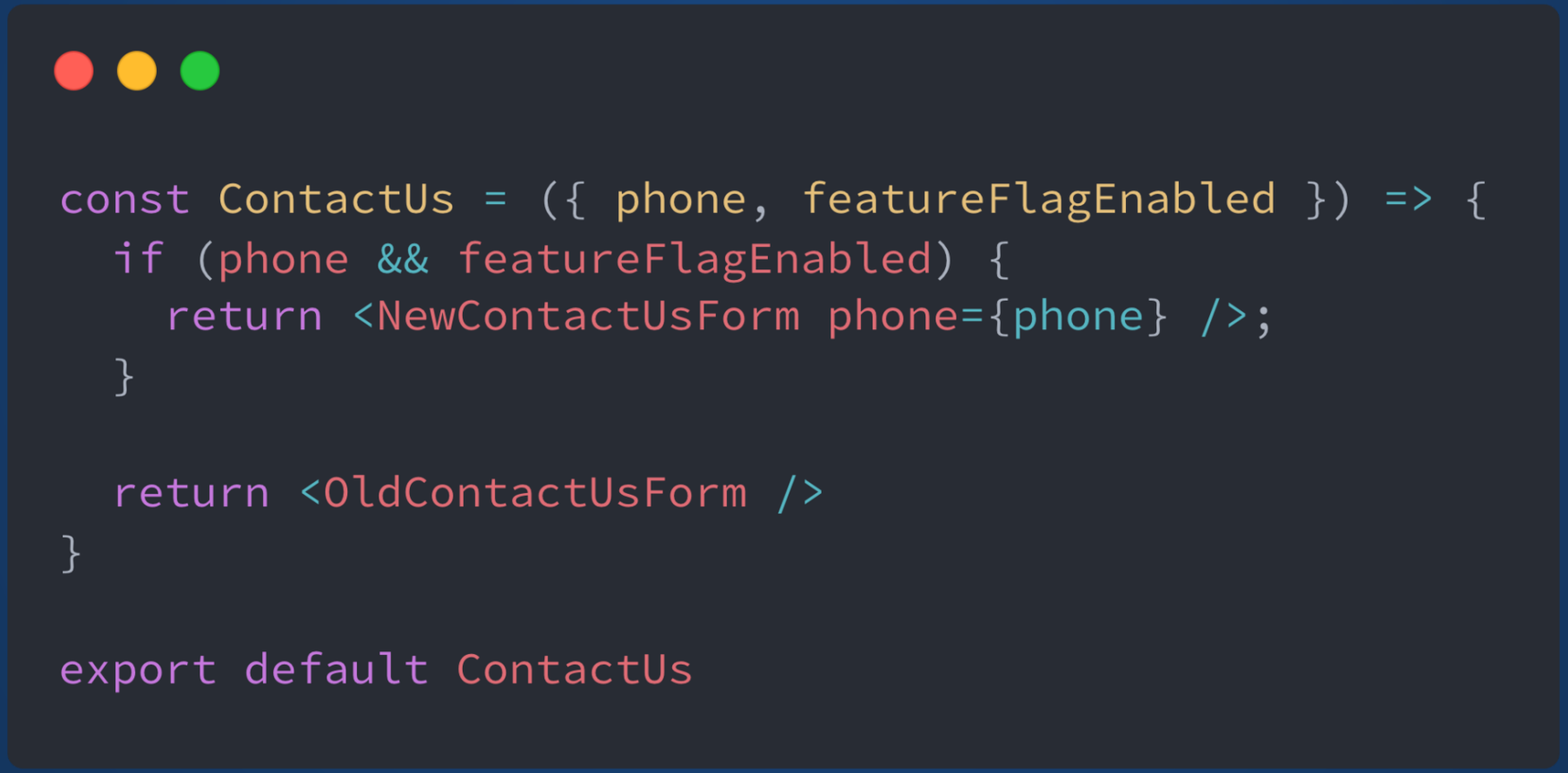
Option 2: Use conditional rendering



Option 2: Use conditional rendering

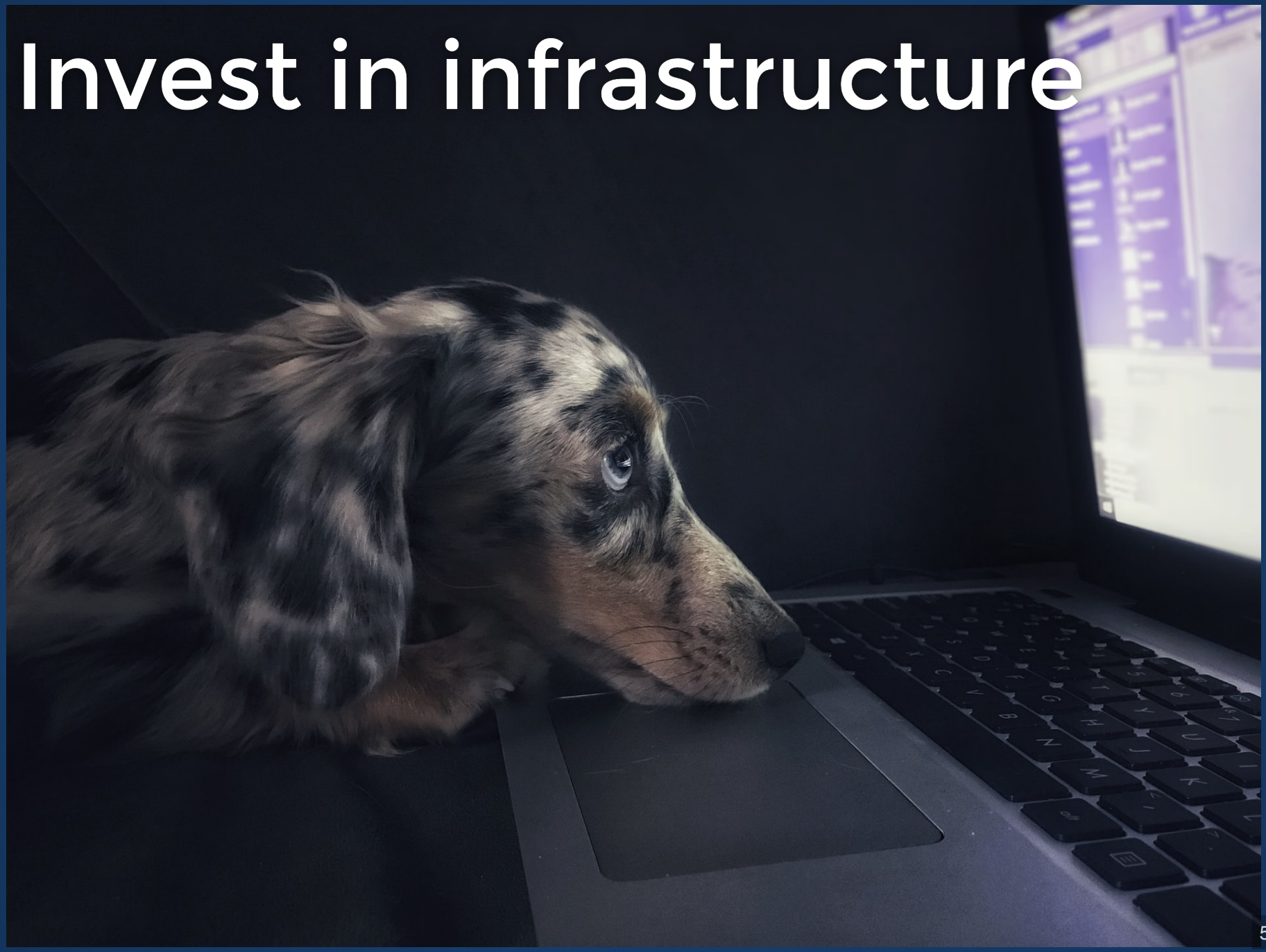
```
const ContactUs = ({ phone }) => {
  if (phone) {
    return <NewContactUsForm phone={phone} />;
  }

  return <OldContactUsForm />
}

export default ContactUs
```

Option 2: Use conditional rendering

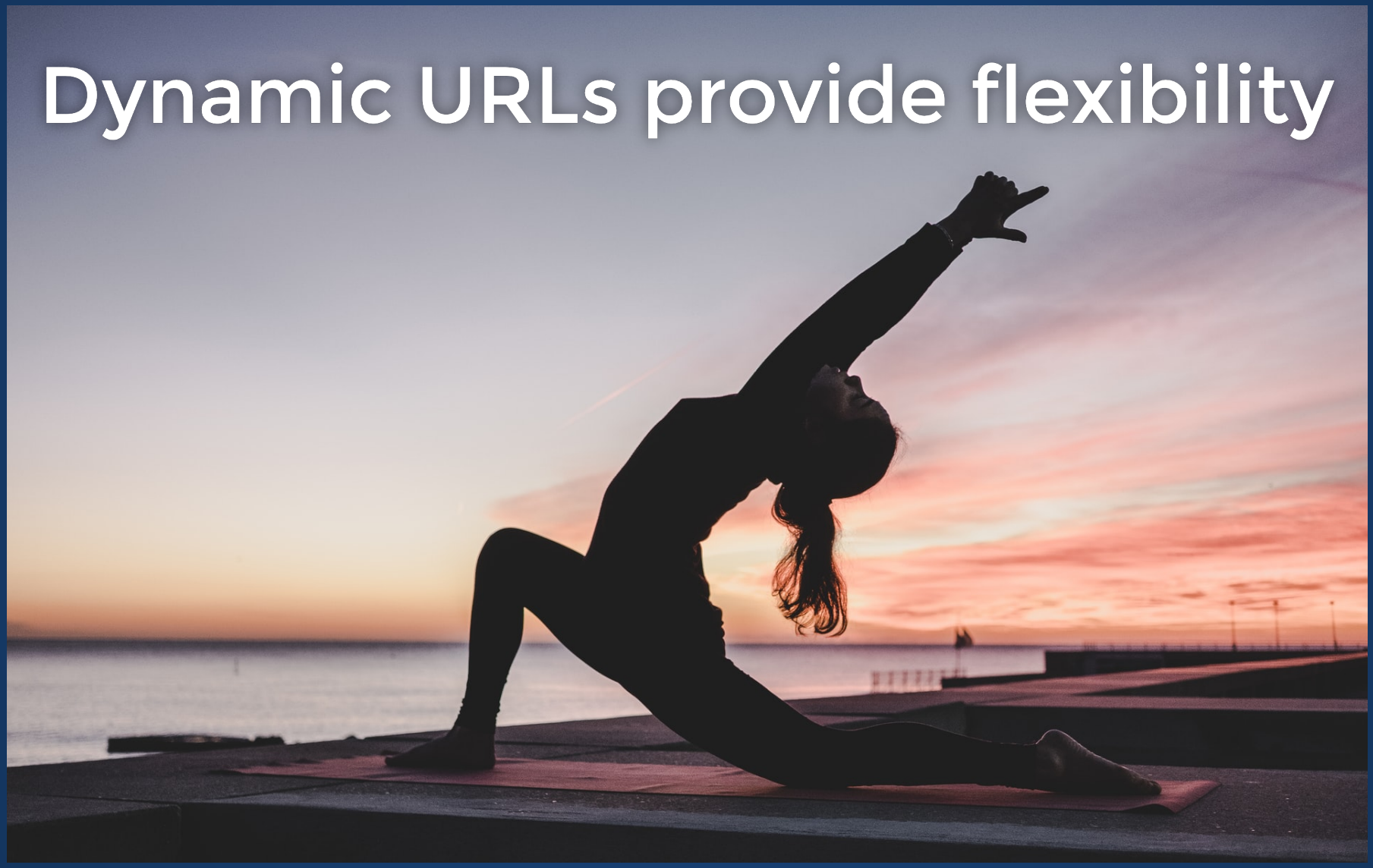


```
const ContactUs = ({ phone, featureFlagEnabled }) => {
  if (phone && featureFlagEnabled) {
    return <NewContactUsForm phone={phone} />;
  }

  return <OldContactUsForm />
}

export default ContactUs
```

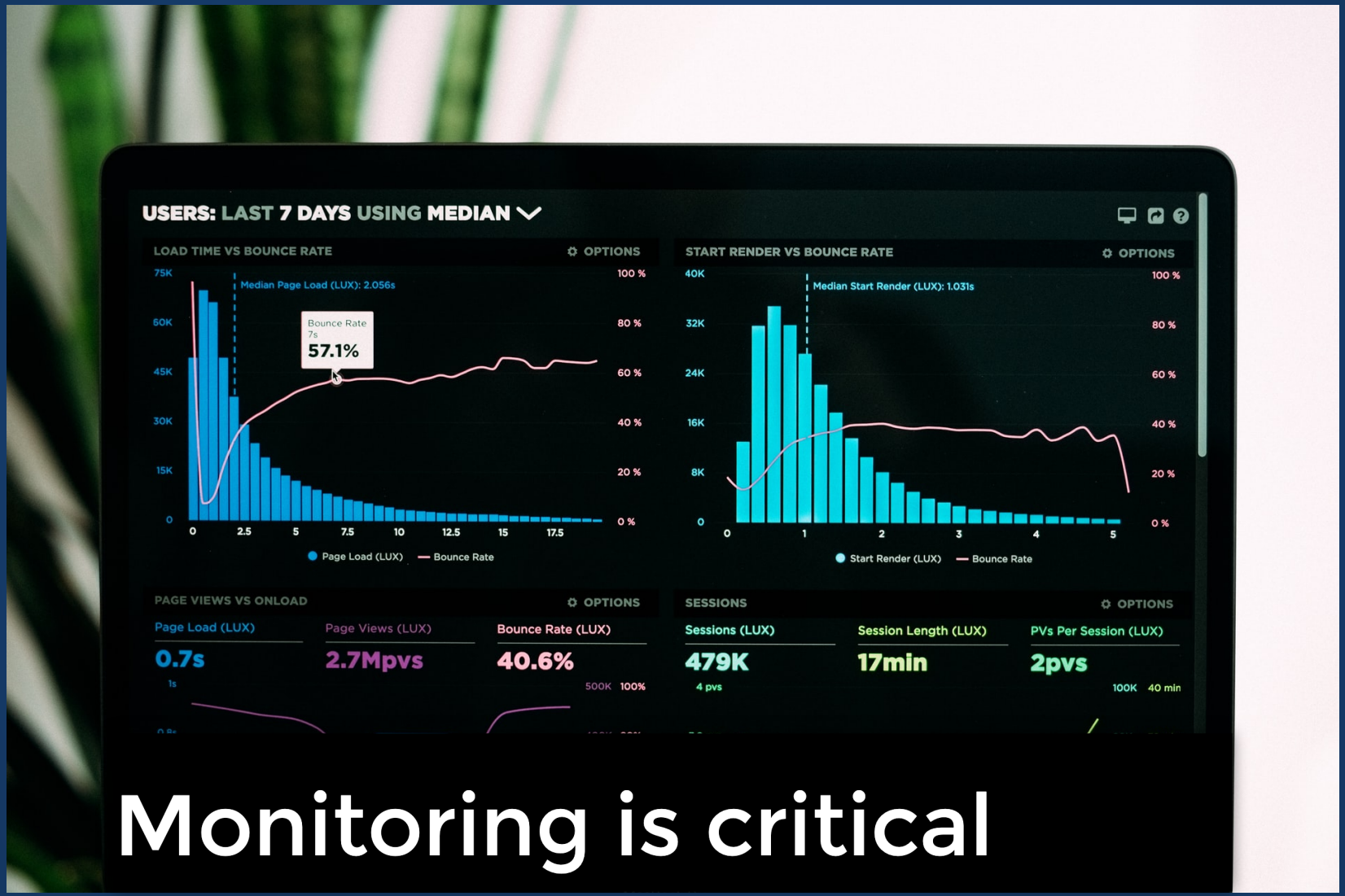
Invest in infrastructure



Dynamic URLs provide flexibility



Tests are your best defense


USERS: LAST 7 DAYS USING MEDIAN
LOAD TIME VS BOUNCE RATE
OPTIONS
Median Page Load (LUX): 2.056s
Bounce Rate
7s
57.1%
Page Load (LUX)
Bounce Rate
START RENDER VS BOUNCE RATE
OPTIONS
Median Start Render (LUX): 1.031s
Start Render (LUX)
Bounce Rate
PAGE VIEWS VS ONLOAD
OPTIONS
Page Load (LUX)
0.7s
Page Views (LUX)
2.7Mpvs
Bounce Rate (LUX)
40.6%
SESSIONS
OPTIONS
Sessions (LUX)
479K
4 pvs
Session Length (LUX)
17min
PVs Per Session (LUX)
2pvs
Monitoring is critical

Work together

to create standards

Top 3 Standards:

1. Requirements for error handling (error boundaries)
2. Minimum testing requirements (past versions)
3. Monitoring requirements


Start small

Give yourself time

Begin with a component that is:

- Non-critical
- Small in code size
- Shared across relatively few places


Wrap Up

Resources

Resources

- Wayfair Engineering's firsthand experience
- Using WFM to serve microfrontend apps
- Practical Guide to Module Federation ebook


THANK
YOU!