Bringing **components** to **legacy code**

“You **can’t use** React,
our system is incompatible”
—Stakeholder

“You **can use** whatever,
we just need to adapt”

—A Reasonable Engineer

What is a **component**?

A component is a software package, a web service, a web resource, or a module that encapsulates a set of related functions (or data).

—wikipedia.org

A **nice** way to
group & reuse code

—Me

ember®

— **Reusability** and **composition** of code

— More **organized** code

— **Easy** to implement and expand

— Developers are **familiar** with them

What is a **legacy system**?

An **existing codebase**

— **Too big** to refactor

— Necessity to **keep existing code** running

— Opportunities to **improve** and **scale**

— Grow the team, **hire** more people

A **restrictive system**

— **Less control** of the system

— Back-end responsible for **HTML rendering**

— Need to **compose pages** based on generated HTML

Magento®

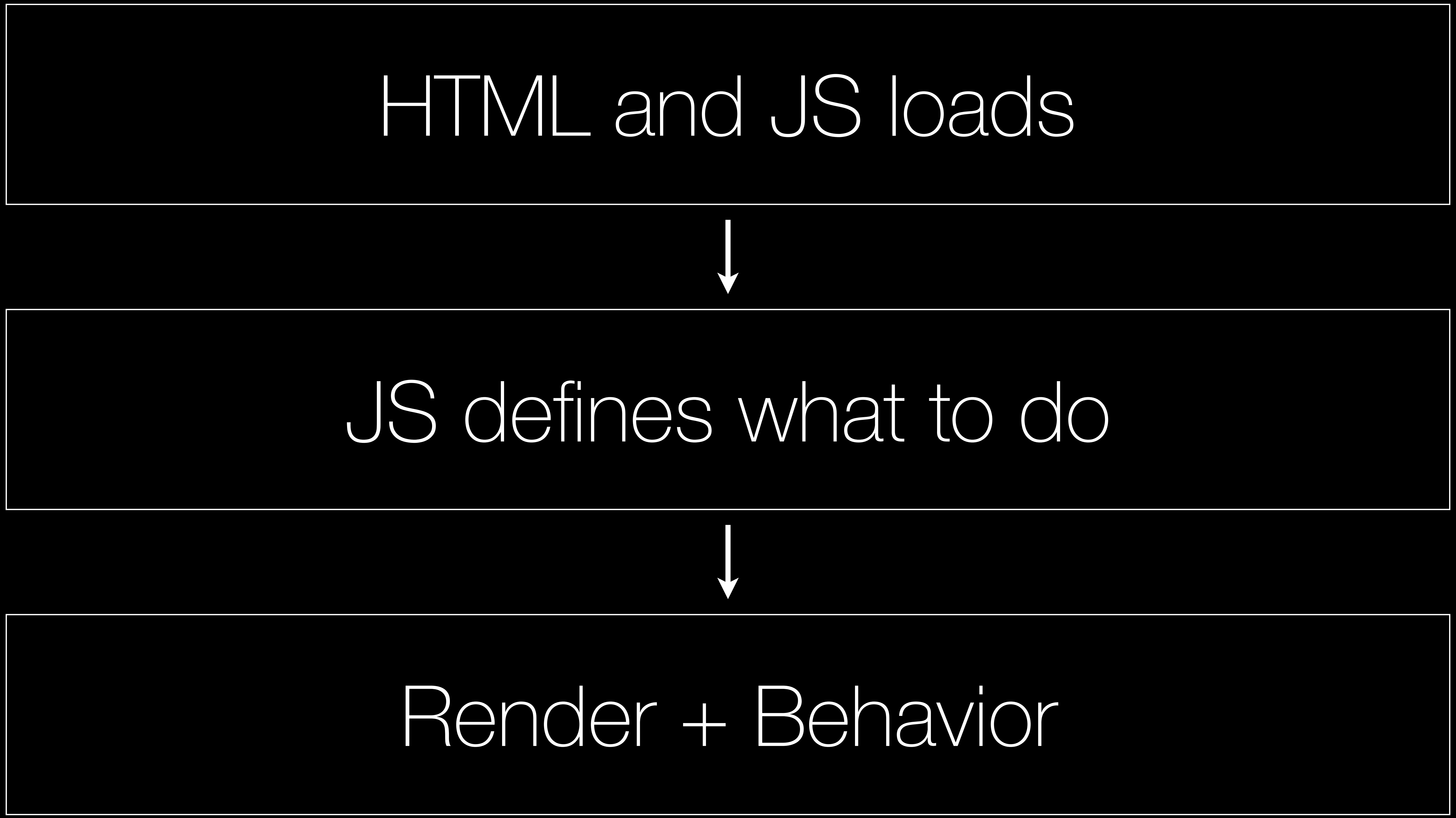
Drupal™

Booking.com

Meetup

Typical framework behavior

JavaScript
sets the rules


HTML and JS loads
JS defines what to do
Render + Behavior

React Controls it All

```html
<div id="App"></div>
```

React Controls it All

```
render(<App />, document.getElementById('App'));
```

jQuery

```html
<div class="Todo">
  <ul class="Todo--Items"></ul>
  <button class="js-todo-new">New Item</button>
</div>
```

jQuery

```
$('.js-todo-new').on('click', addNewItem);
```

— All JS frameworks require **specific elements** to exist, configured in the JS file

— Some need **total control of the DOM** for rendering

— They don't expect other things running in **parallel**

How does it play with a

legacy system

- You **can’t replace** the entire system with a modern application
- The system might provide **customization through HTML**

What do we **do**?

— It’s a **strategic decision** to bring a different system into place

— How does the framework **enable** the requirements of the new features?

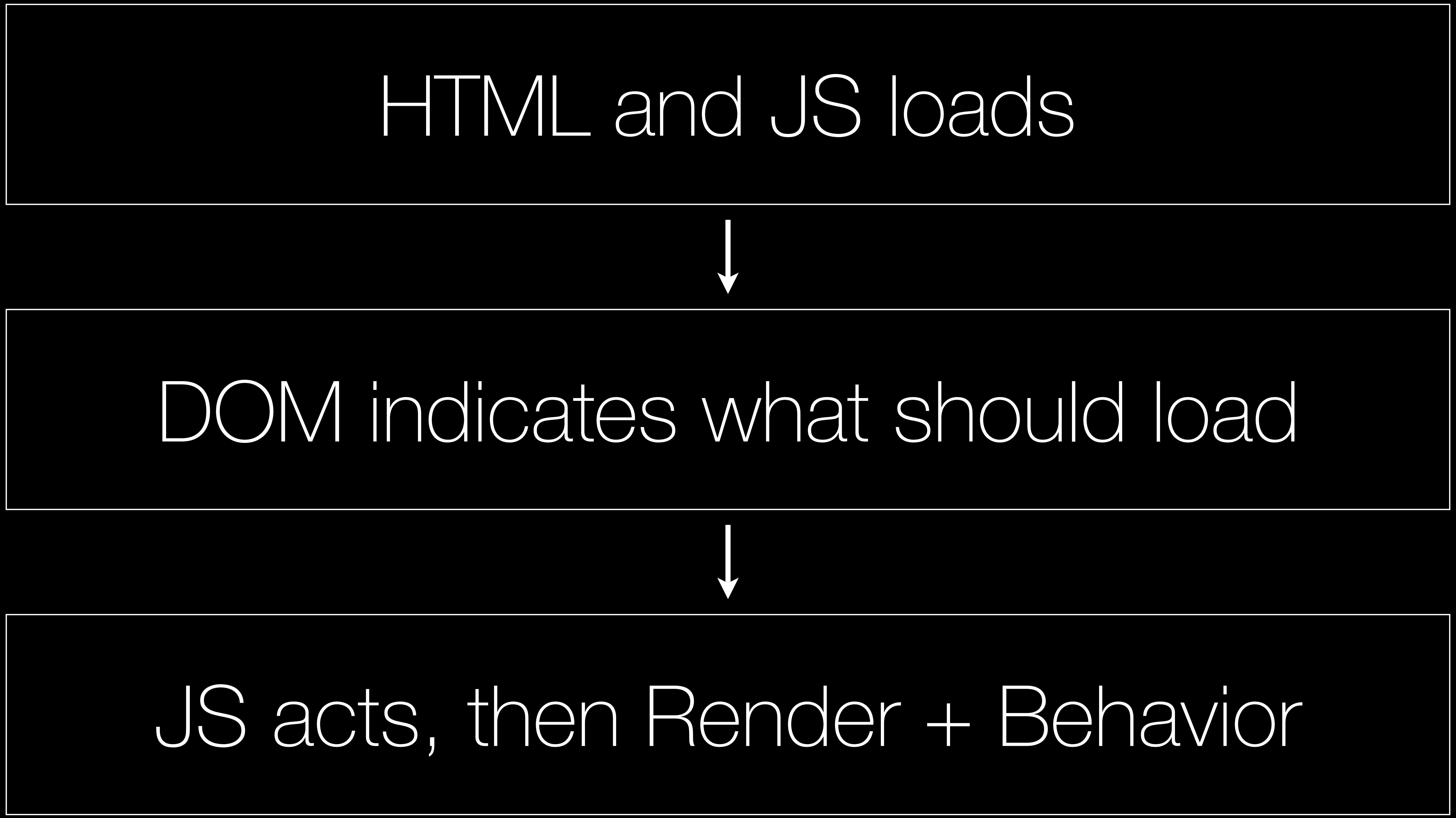
We **flip** the responsibilities

Before:

JavaScript
sets the rules

After:

The DOM
sets the rules


HTML and JS loads
DOM indicates what should load
JS acts, then Render + Behavior

Using **React**

React Controls it All

```
<div id="App"></div>
```

After: Multiple React Apps

```
<div data-component="Header"></div>

(...)

<a href='#' onclick='veryOldThing();void(0)'>

(...)

<div data-component="PostHeader">
  <script type="application/json" data-props>
    {“title”:"Bringing Components to Legacy Code"}
  </script>
</div>
```

After: Multiple React Apps

```
<div data-component="Header"></div>

(...)

<a href='#' onclick='veryOldThing();void(0)'>

(...)

<div data-component="PostHeader">
  <script type="application/json" data-props>
    {“title”:"Bringing Components to Legacy Code"}
  </script>
</div>
```

After: Multiple React Apps

```
import Header from './components/Header';

const componentList = {
  Header: Header,
}

const collection = document.querySelectorAll('[data-component]');
Array.from(collection).forEach(element => {
  const componentName = element.getAttribute('data-component');
  const Component = componentList[componentName];

  render(<Component />, element);
});
```

After: Multiple React Apps

```
import Header from './components/Header';

const componentList = {
  Header: Header,
}

const collection = document.querySelectorAll('[data-component]');
Array.from(collection).forEach(element => {
  const componentName = element.getAttribute('data-component');
  const Component = componentList[componentName];

  render(<Component />, element);
});
```

After: Multiple React Apps

```
import Header from './components/Header';

const componentList = {
  Header: Header,
}

const collection = document.querySelectorAll('[data-component]');
Array.from(collection).forEach(element => {
  const componentName = element.getAttribute('data-component');
  const Component = componentList[componentName];

  render(<Component />, element);
});
```

After: Multiple React Apps

```
import Header from './components/Header';

const componentList = {
  Header: Header,
}

const collection = document.querySelectorAll('[data-component]');
Array.from(collection).forEach(element => {
  const componentName = element.getAttribute('data-component');
  const Component = componentList[componentName];

  render(<Component />, element);
});
```

After: Multiple React Apps

```
import Header from './components/Header';

const componentList = {
  Header: Header,
}

const collection = document.querySelectorAll('[data-component]');
Array.from(collection).forEach(element => {
  const componentName = element.getAttribute('data-component');
  const Component = componentList[componentName];

  render(<Component />, element);
});
```

⌘ + Tab

In the end

— **Data attributes** define which components are present

— Multiple systems can **coexist** in different areas of the page

— A strict back-end can **personalize** a page’s components and provide data

Consider this

— Be aware of the **impact** of multiple framework applications

— **Bundling** might be a challenge

— Remember to share **state management**

This is also great for
simple applications

— Helps **organize your code** in components

— Adds **flexibility** on what code runs on your pages

— Very **small** footprint

data-components

github.com/rafaelrinaldi/data-components

References

@WesleydeSouza

@NYCReact

Thank you!

@WesleydeSouza

@NYCReact