GDSC SSU
SEMINAR

WEB
FRONTEND

WEB FRONTEND

View
&
Interaction

HTML, CSS & JavaScript

```html
<!DOCTYPE html>
<html>
<head>
  <title>2nd GDSC Soongsil</title>
</head>
<body>
  <div class=“container”>
    <p class=“text”>Hello!</p>
  </div>
</body>
</html>
```

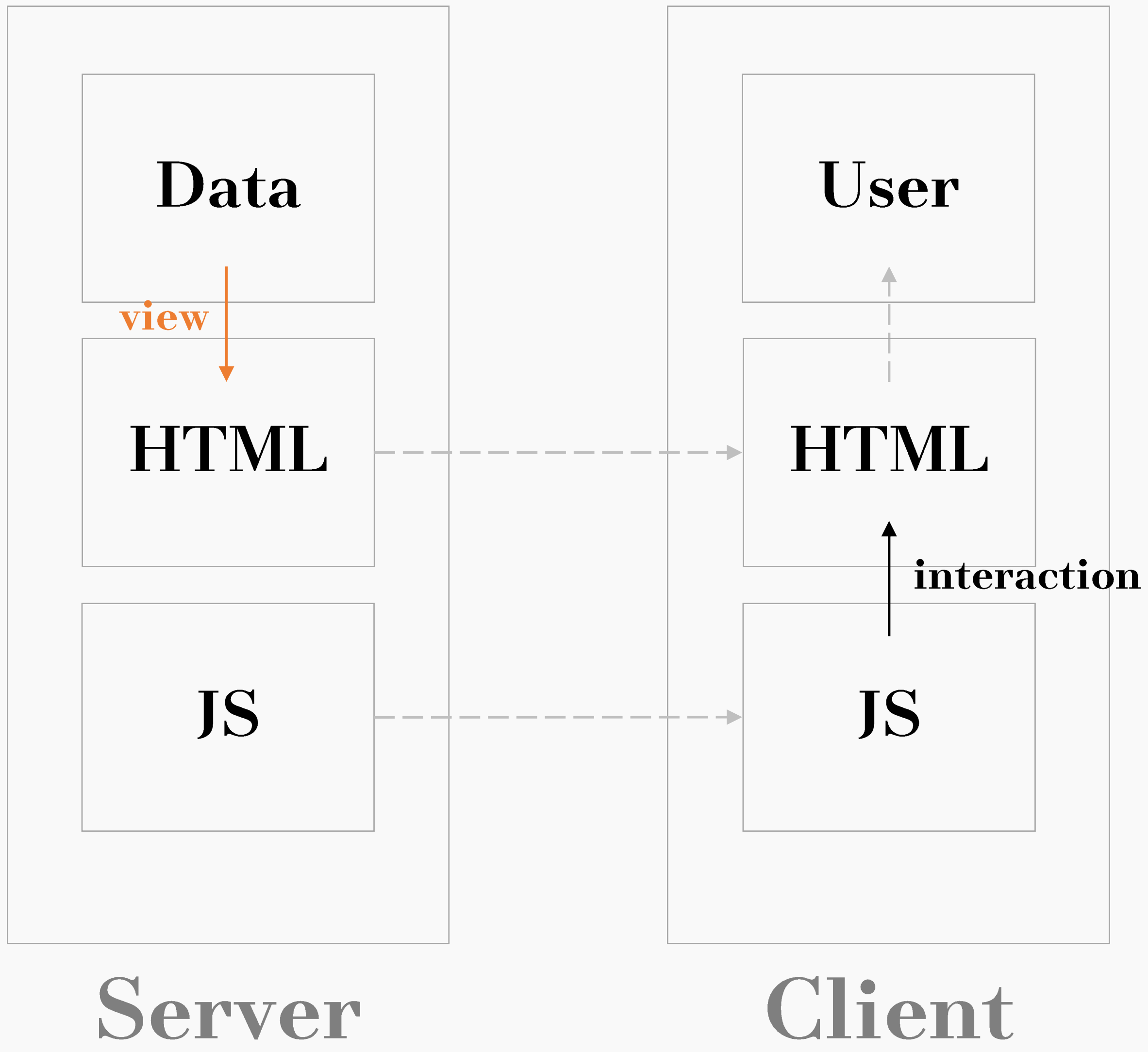
```javascript
const container =
  document.querySelector(‘.container’);

container.addEventListener(‘click’, () => {
  console.log(“Hello World!”);
});
```

```
<!DOCTYPE html>
<html>
<head>
  <title>2nd GDSC Soongsil</title>
</head>
<body>
{% for o in some_list %}
  <div class="container">
    <p class="text">Hello {{ o }}!</p>
  </div>
{% endfor %}
</body>
</html>
```

```
const container =
  document.querySelector('.container');

container.addEventListener('click', () => {
  console.log("Hello World!");
});
```

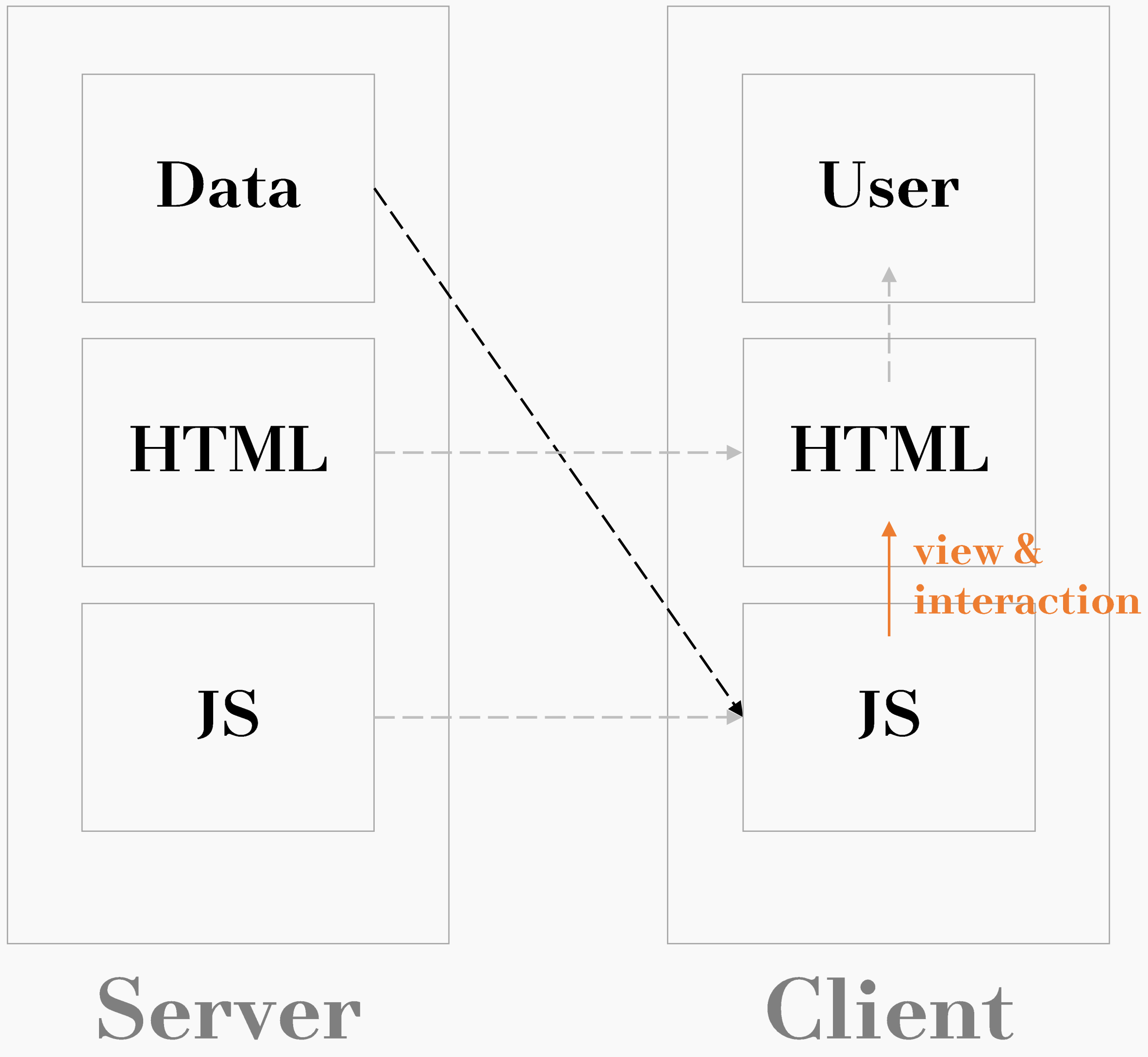
```html
<!DOCTYPE html>
<html>
<head>
  <title>2nd GDSC Soongsil</title>
</head>
<body>
  <div class="container">
    <p class="text">Hello a!</p>
  </div>
  <div class="container">
    <p class="text">Hello b!</p>
  </div>
  <div class="container">
    <p class="text">Hello c!</p>
  </div>
</body>
</html>
```

```javascript
const container =
  document.querySelector('.container');

container.addEventListener('click', () => {
  console.log("Hello World!");
});
```


Data
view
HTML
JS
Server
User
HTML
interaction
JS
Client

```html
<!DOCTYPE html>
<html>
<head>
  <title>2nd GDSC Soongsil</title>
</head>
<body>
{% for o in some_list %}
  <div class="container">
    <p class="text">Hello {{ o }}!</p>
  </div>
{% endfor %}
</body>
</html>
```

```javascript
const body =
  document.querySelector('body');

let containers = '';

const someList = await fetch( /* ... */ );

someList.forEach((o) => {
  containers += `
    <div class="container">
      <p class="text">Hello ${o}!</p>
    </div>
  `;
});

body.innerHTML = containers;

const container =
  document.querySelector('.container');

container.addEventListener('click', () => {
  console.log("Hello World!");
});
```


Data
HTML
JS
User
HTML
view &
interaction
JS
Server
Client

Server

Data Processing
(CRUD)

Client

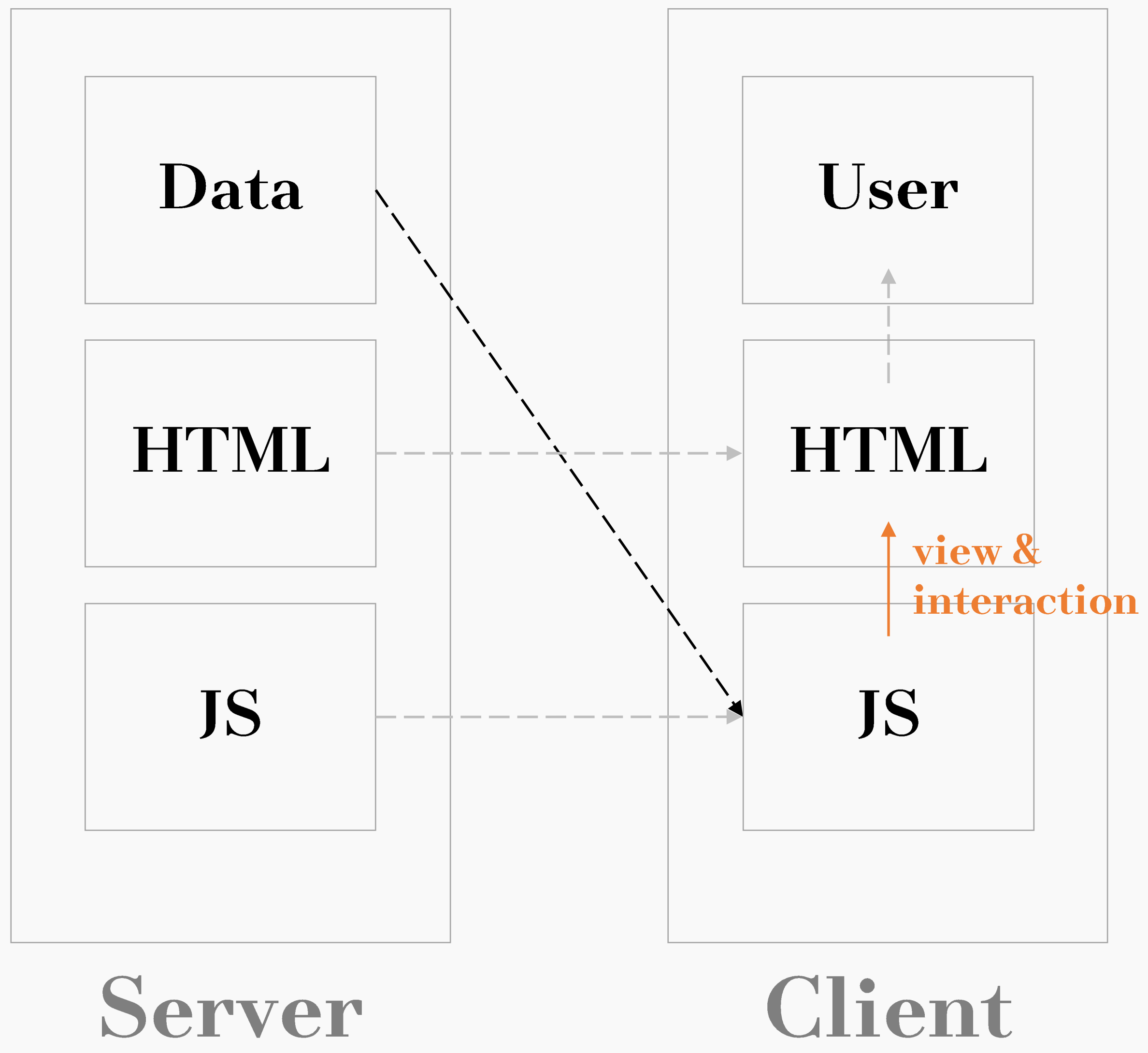
View &
Interaction

1.

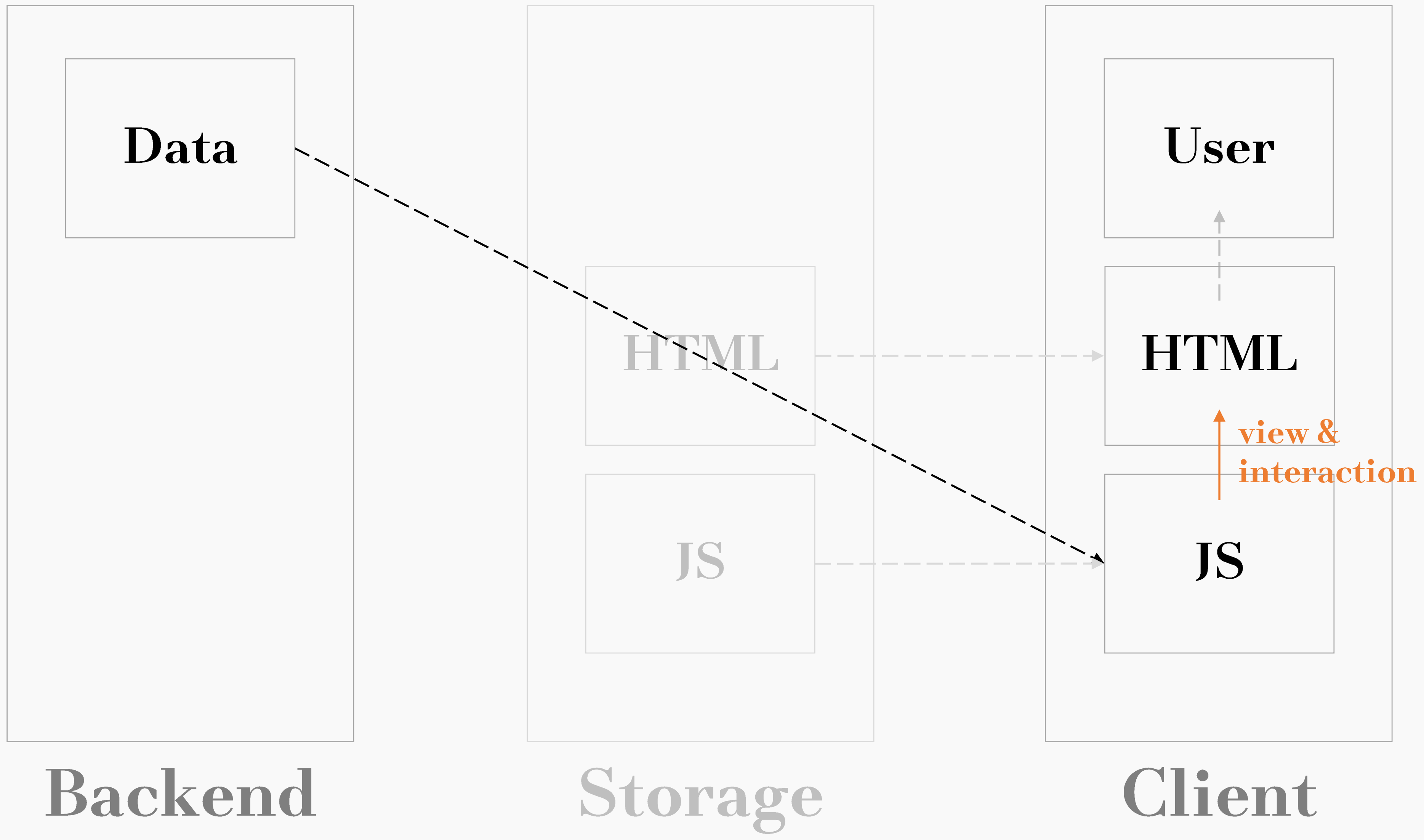
Server or Client

1. Server or Client

Client Side Rendering



1. Server or Client



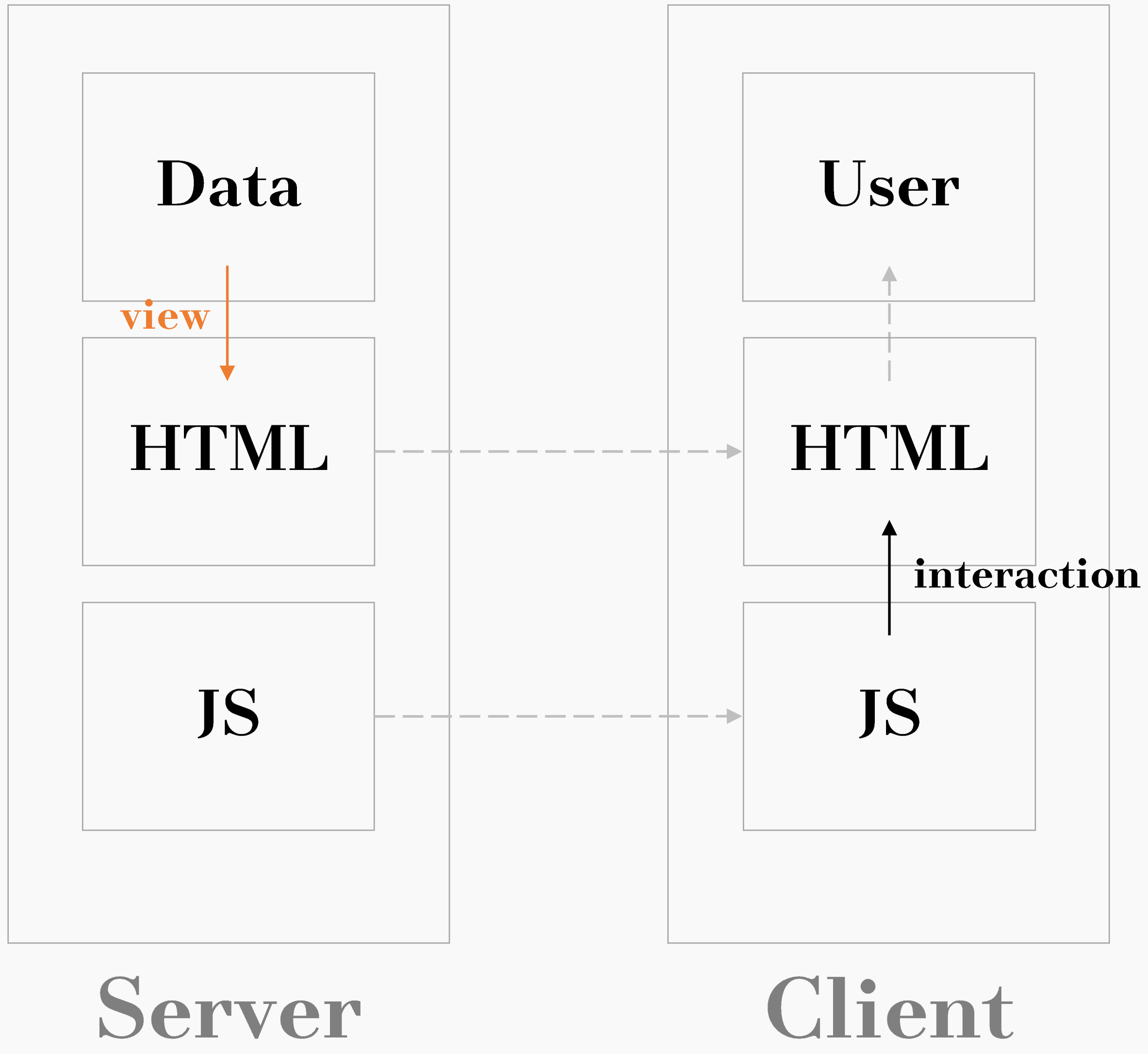
1. Server or Client

React, JSX

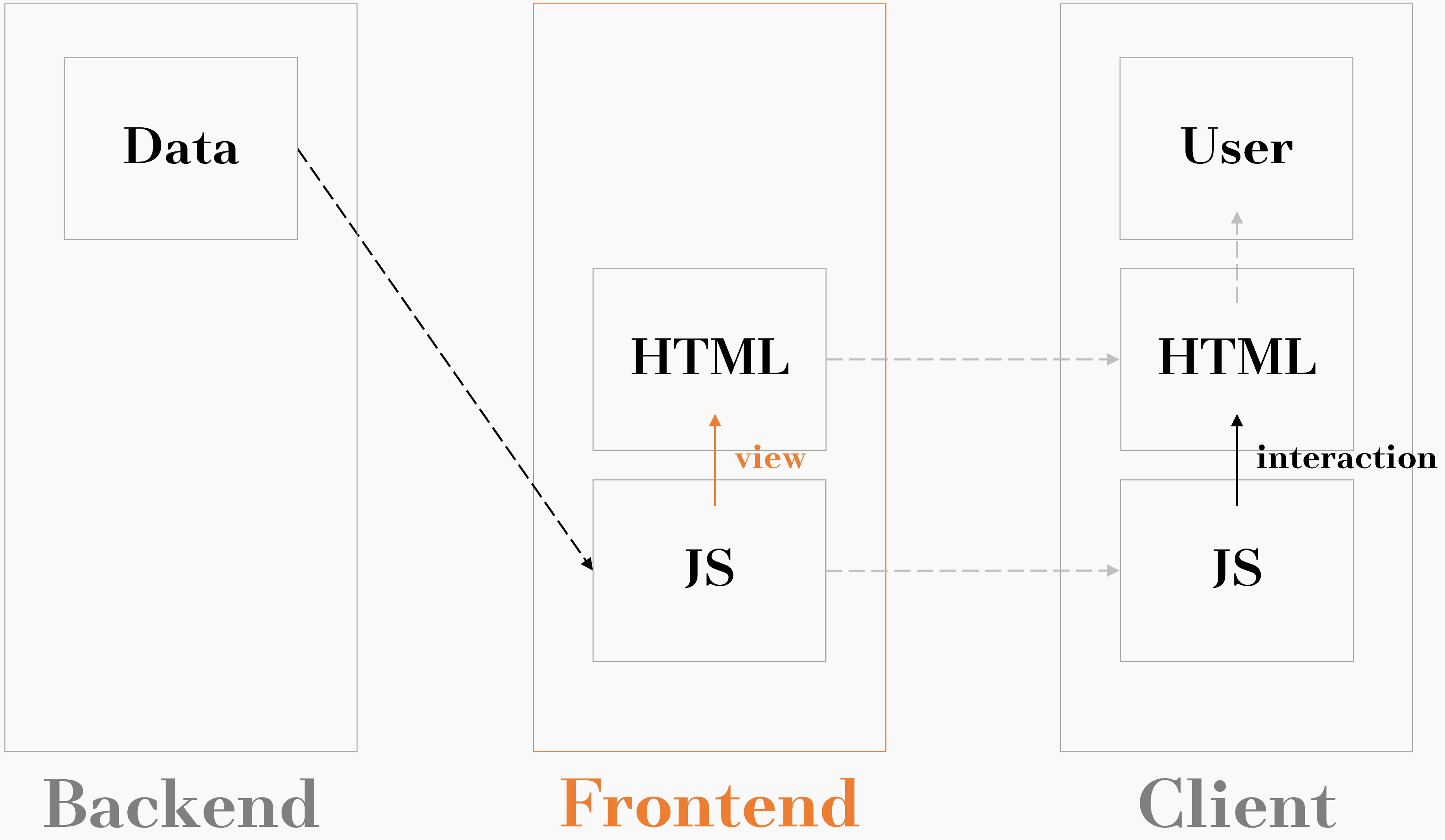
```
function Component({text}) {
  return (
    <div className="container">
      <p className="text">
        {text}
      </p>
    </div>
  );
}

function App() {
  return (
    <>
      {someList.map((o) => (
        <Component text={o} />
      ))}
    </>
  );
}
```

1. Server or Client

Server Side Rendering



1. Server or Client

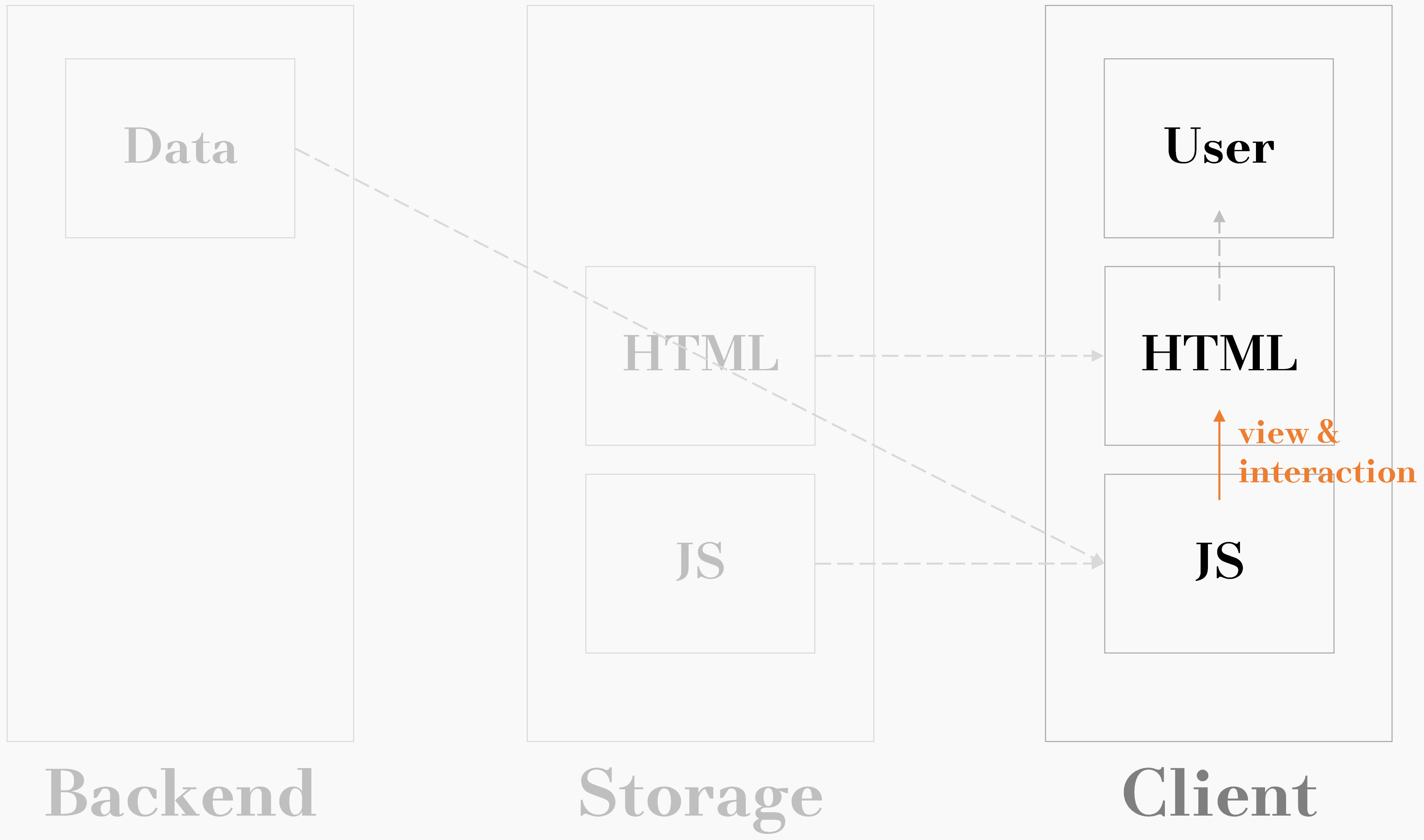


1. Server or Client

Next.js, Gatsby

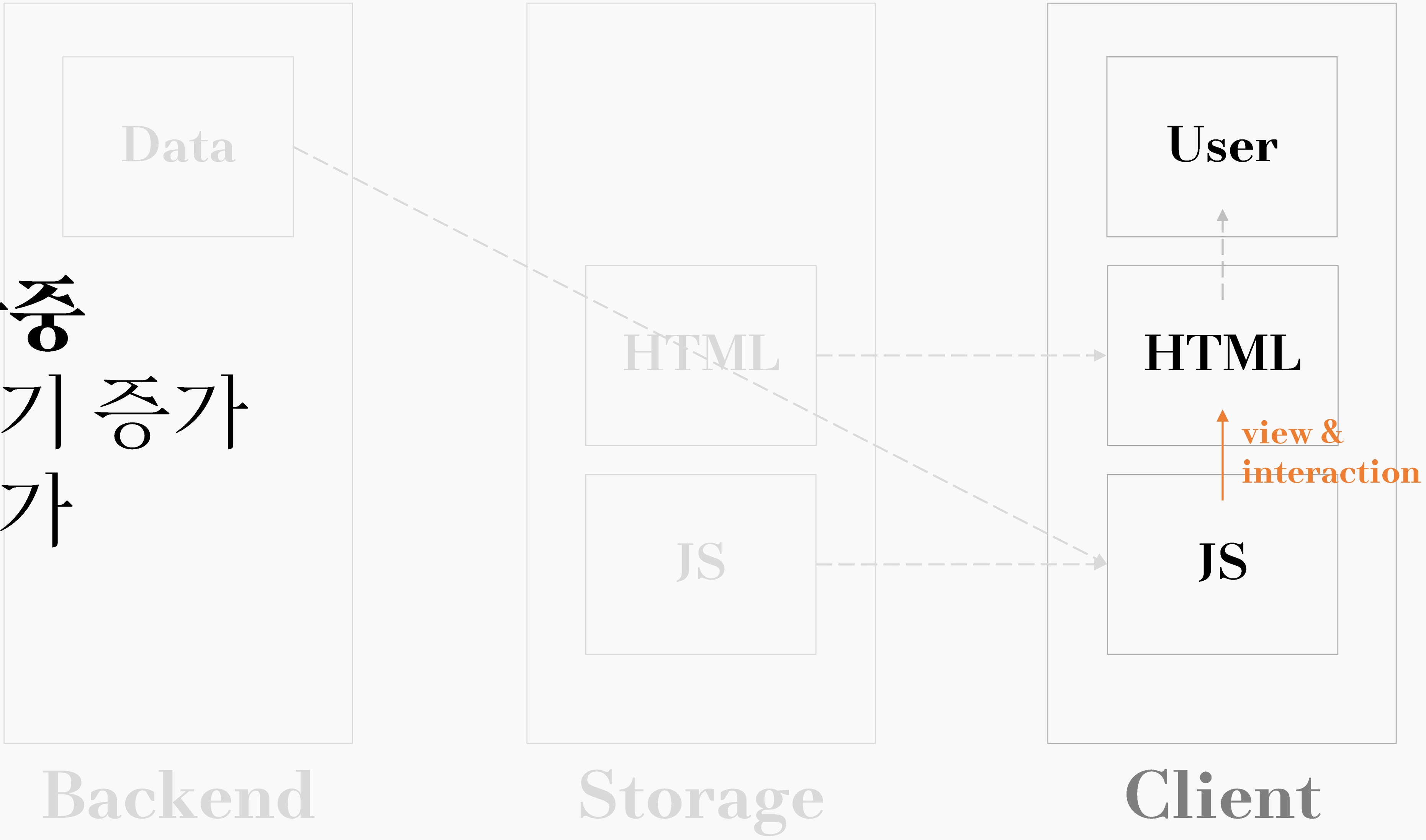
Nuxt.js

Astro, Fresh


Data
HTML
JS
User
HTML
JS
view &
interaction
Backend
Storage
Client

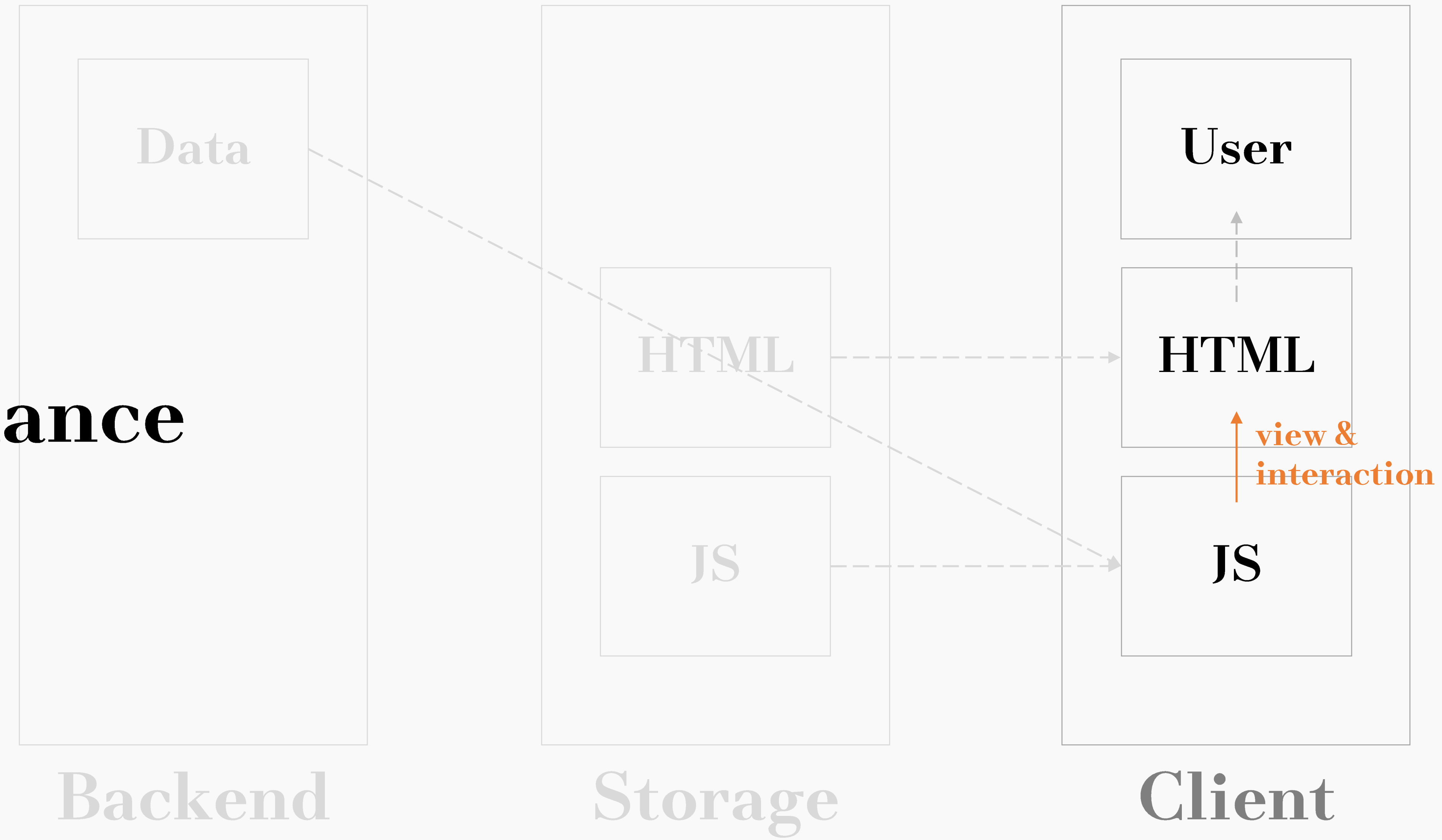
JS 역할 과중

- 파일 크기 증가
- 연산 증가

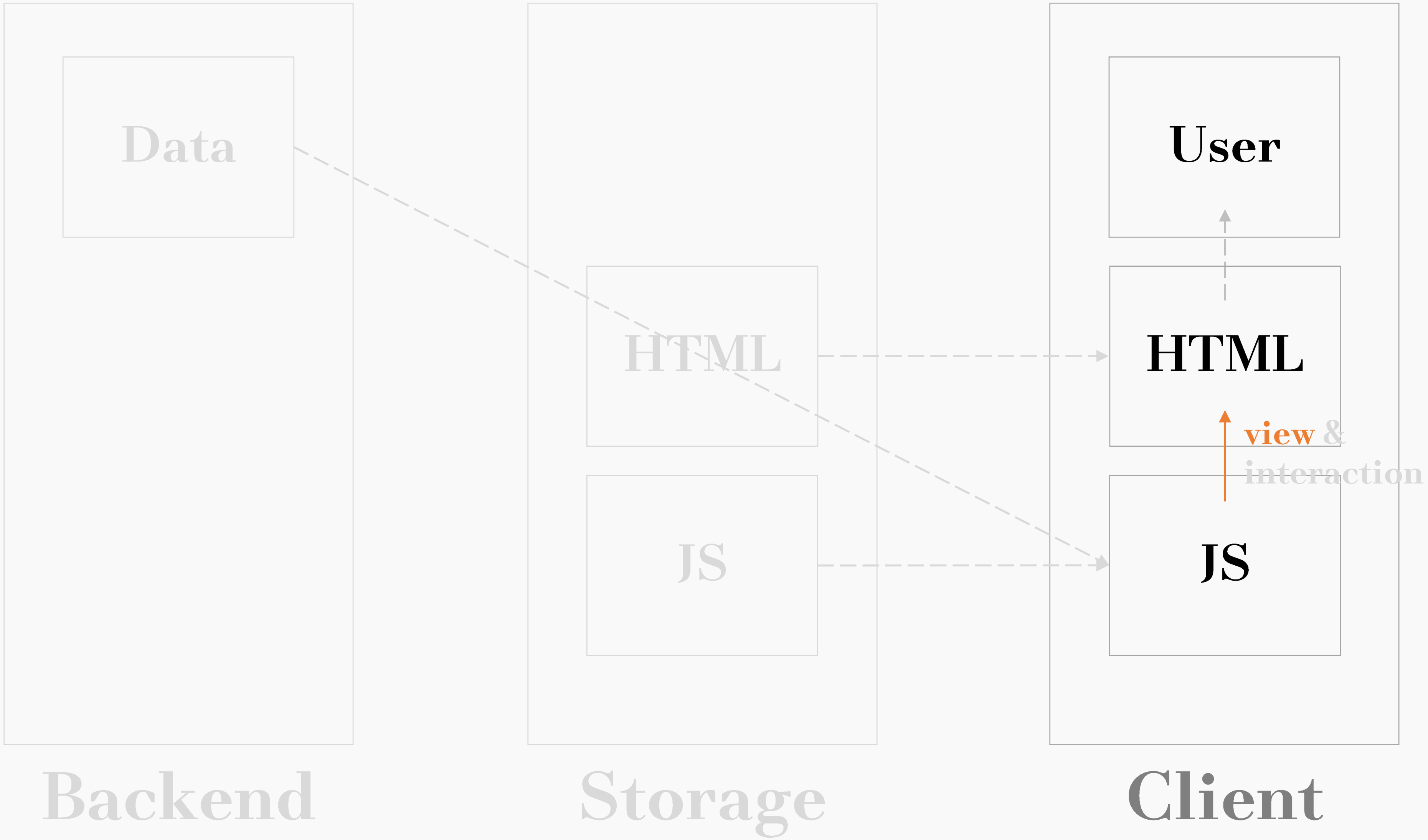


2.

Performance

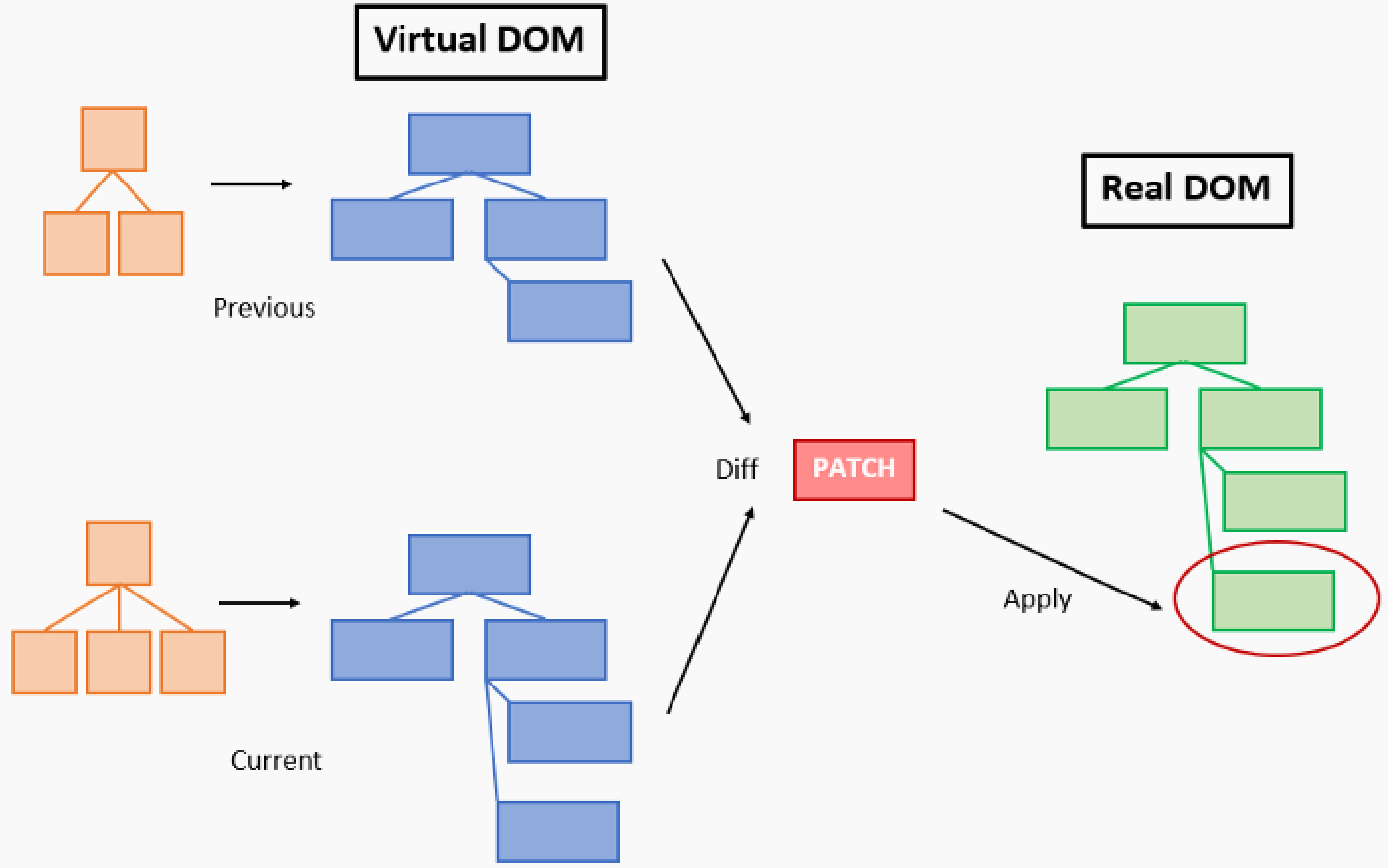


2. Performance



2. Performance

Virtual DOM

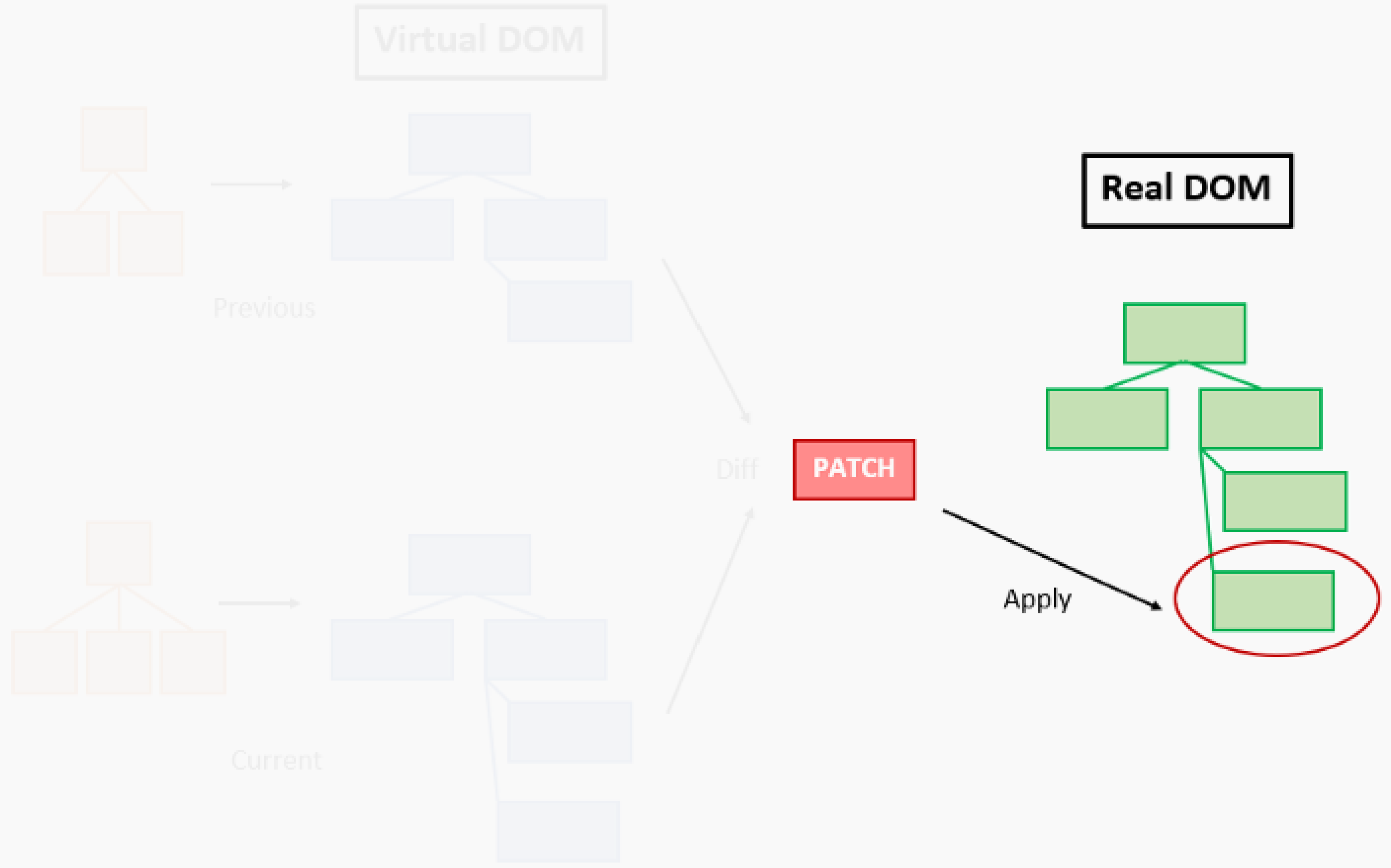


React, Vue

Source : 'Incremental vs Virtual DOM': Chameera Dulanga

2. Performance

Reactivity

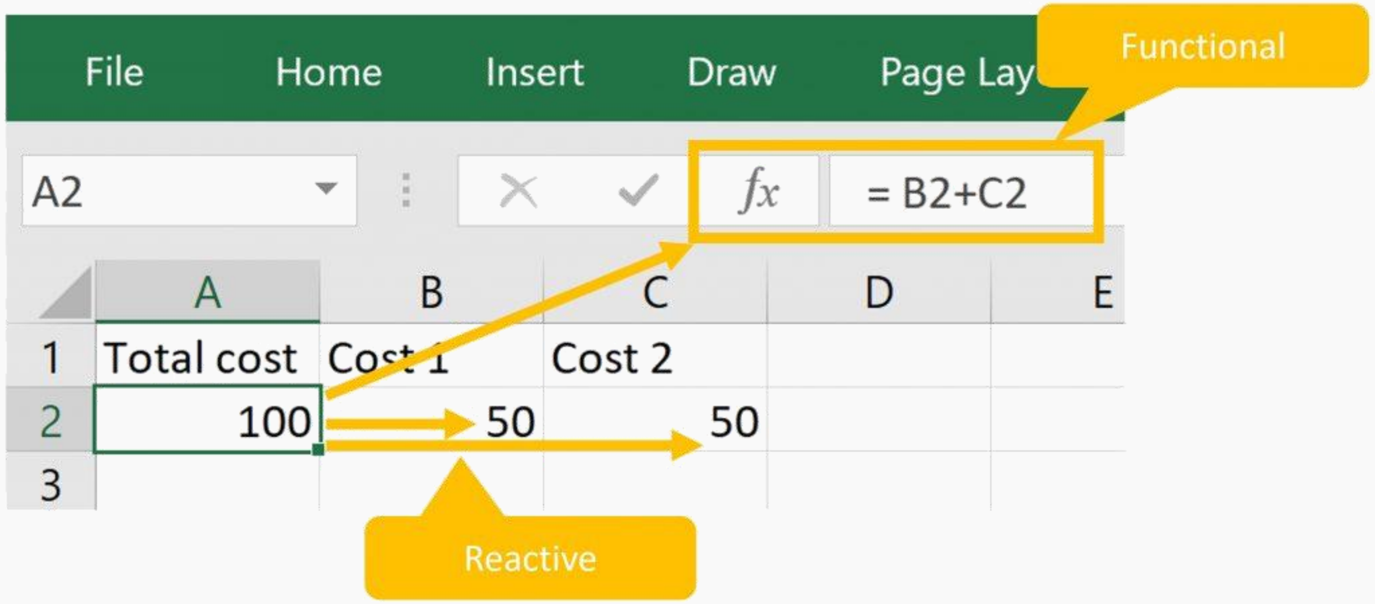


Source : 'Incremental vs Virtual DOM': Chameera Dulanga

Svelte, Solid.js

2. Performance

Reactivity

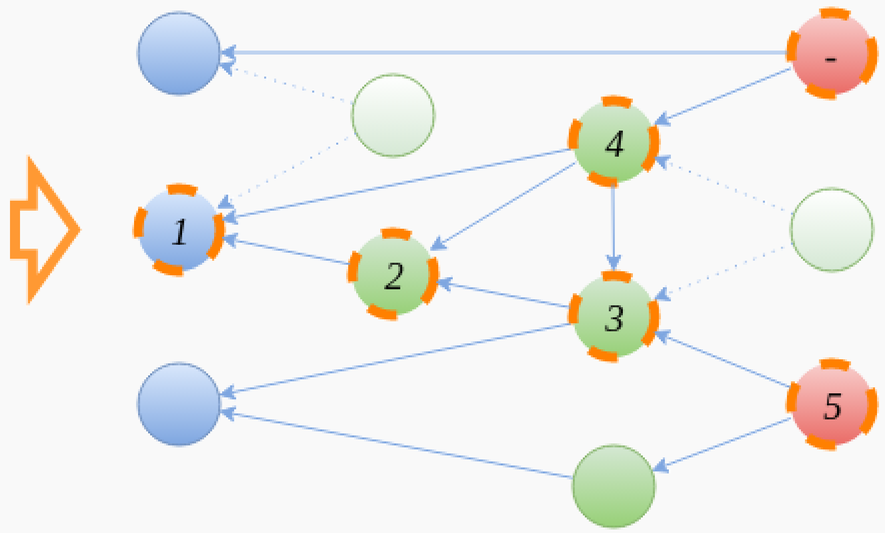


Source : What is Functional Reactive Programming (FRP)?: Malte Bucksch

Svelte, Solid.js

2. Performance

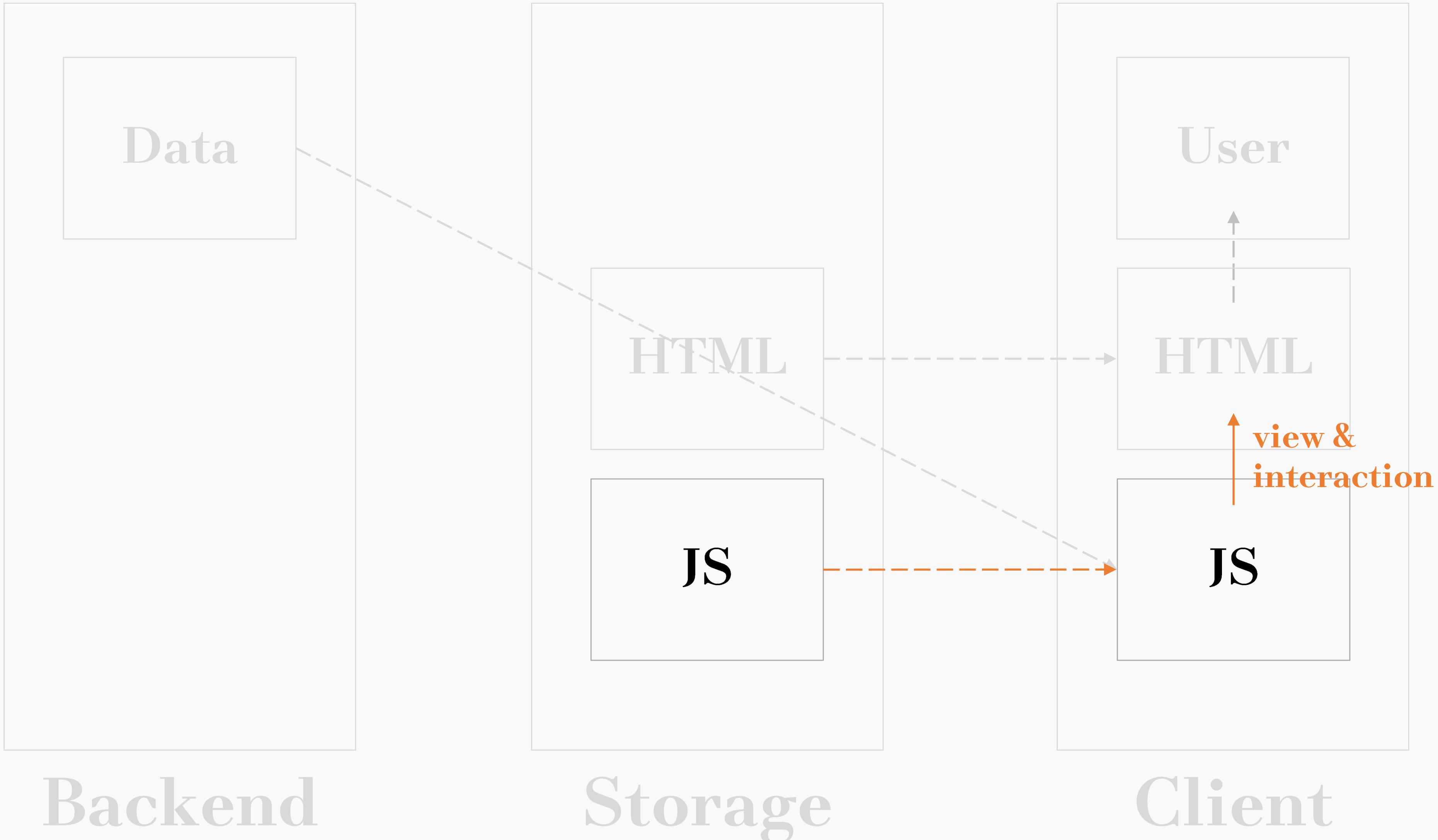
Reactivity



Source : Becoming fully reactive: an in-depth explanation of MobX: Michel Weststrate

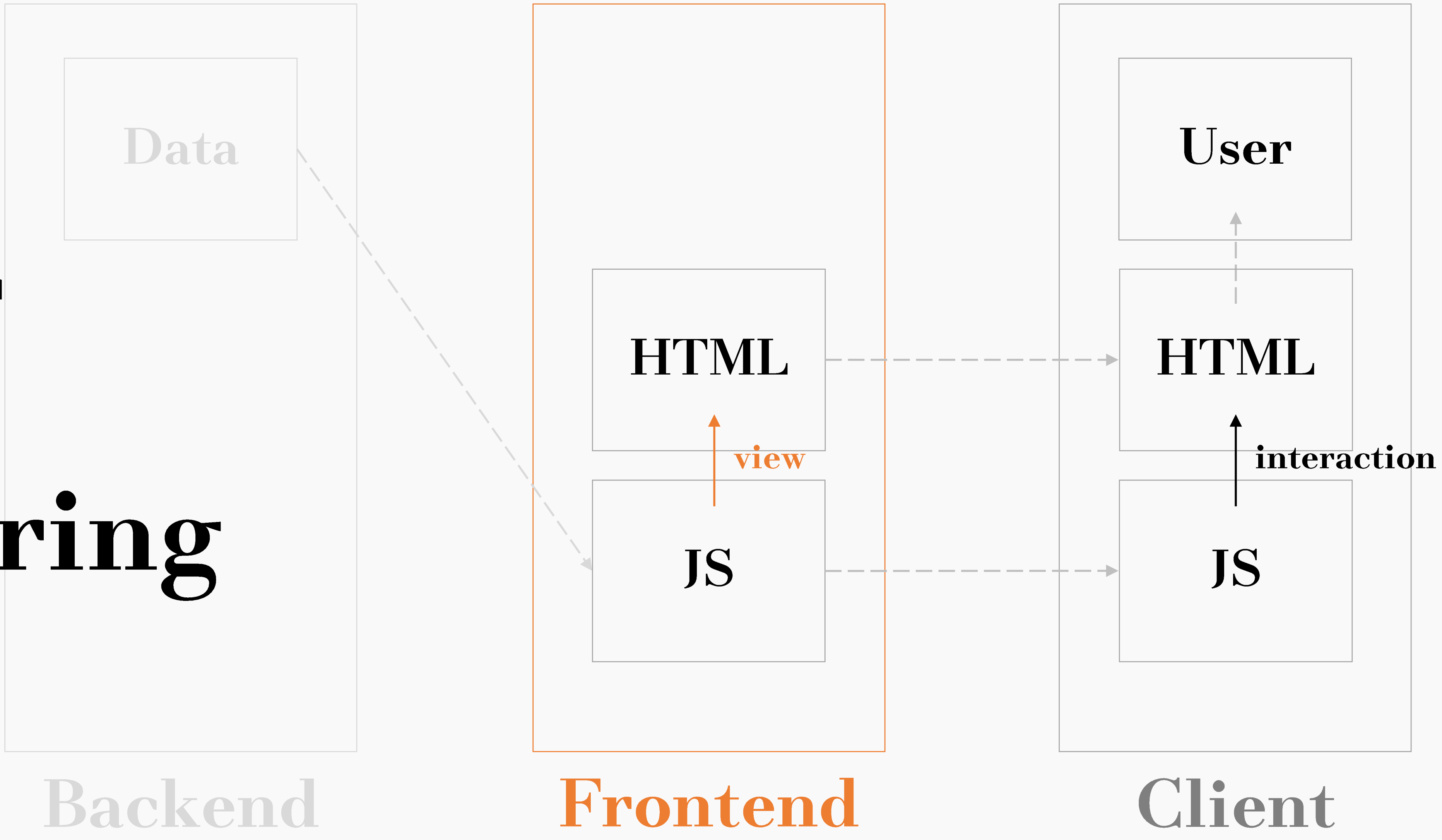
Svelte, Solid.js

2. Performance



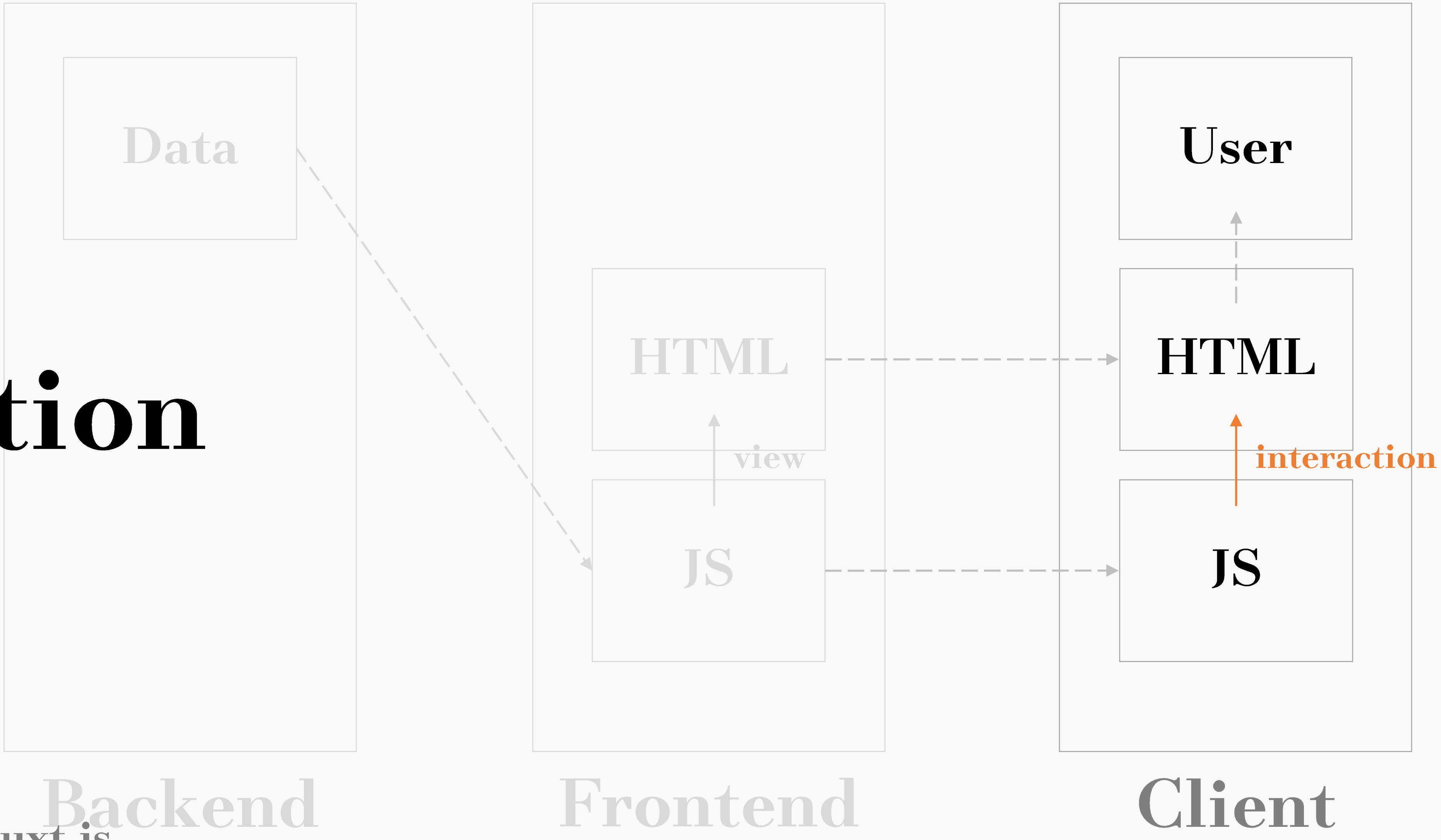
2. Performance

Server Side Rendering



2. Performance

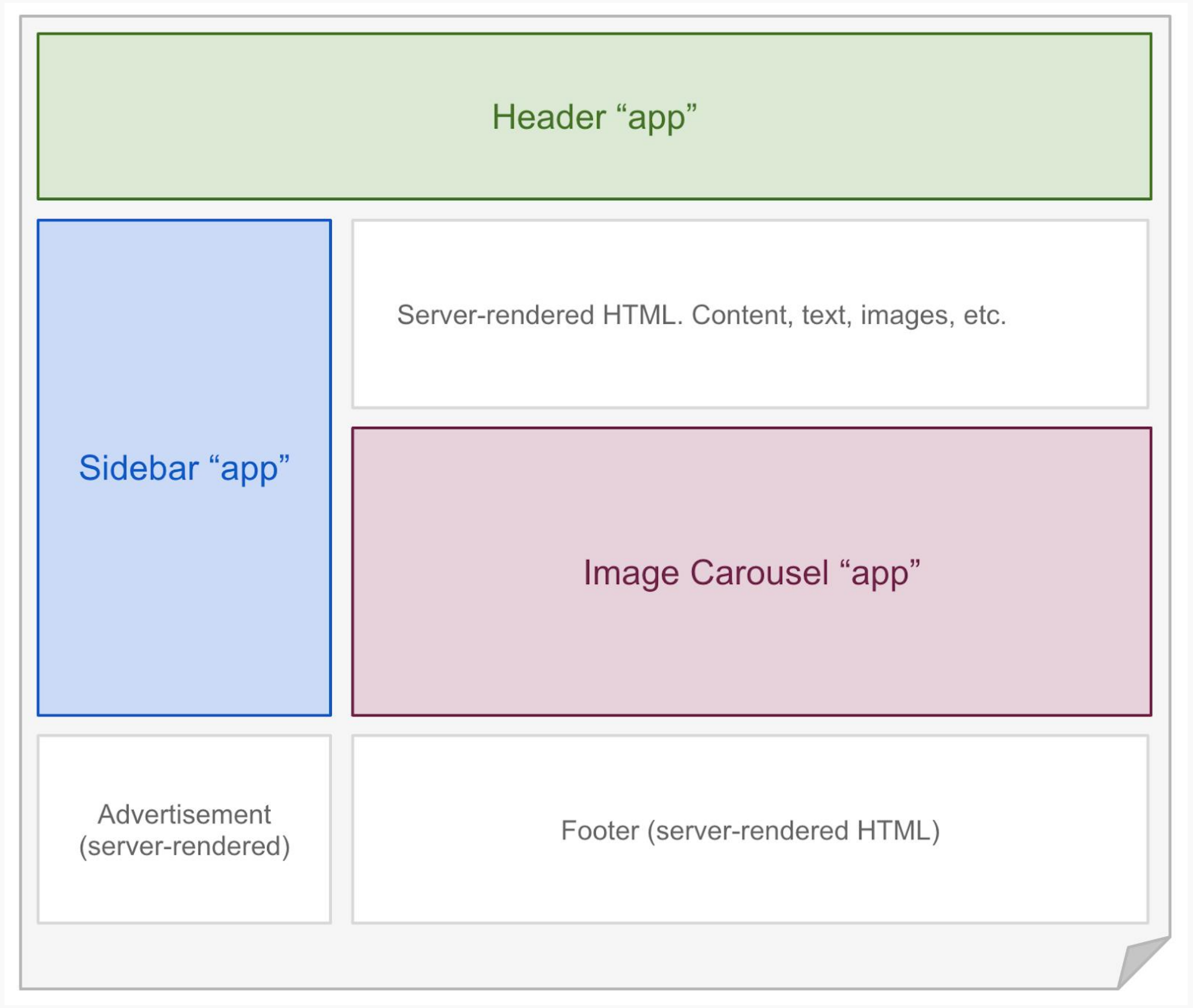
Hydration



Next.js, Gatsby, Nuxt.js

2. Performance

Partial Hydration

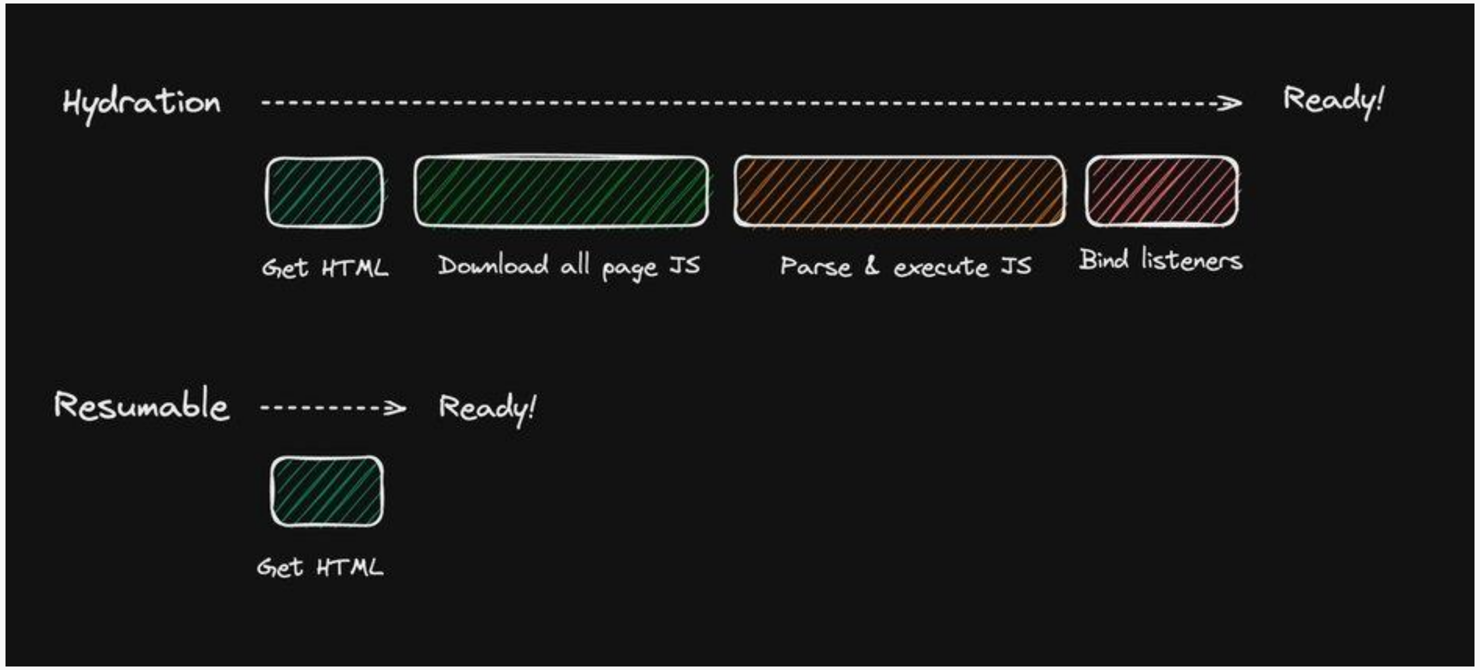

Header “app”
Sidebar “app”
Server-rendered HTML. Content, text, images, etc.
Image Carousel “app”
Advertisement (server-rendered)
Footer (server-rendered HTML)

Astro, Fresh, Marko

Source : Islands Architecture: Jason Miller

2. Performance

Resumable

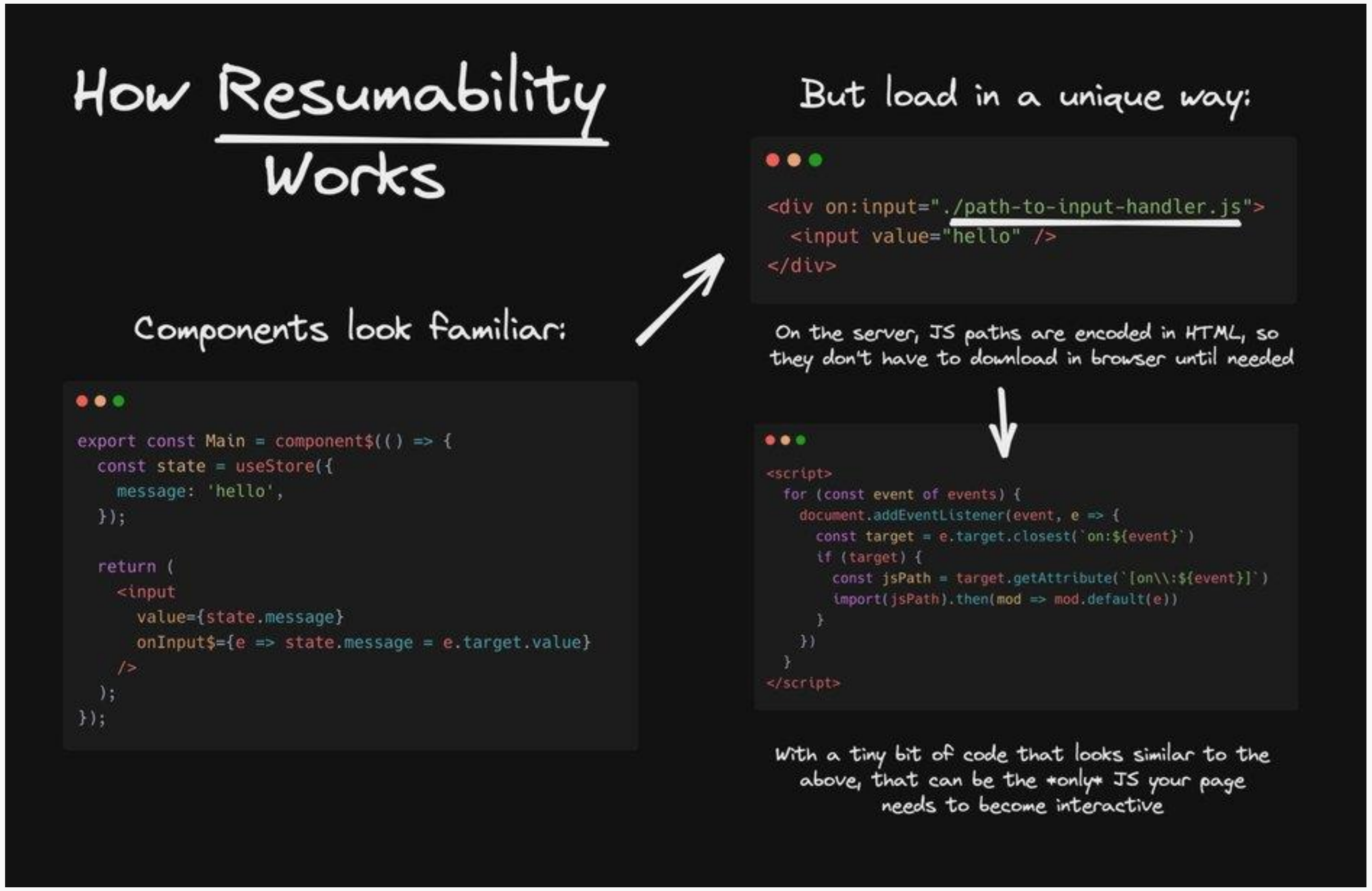


Source : Hydration is Pure Overhead: MIŠKO HEVERY

Qwik

2. Performance

Resumable


How Resumability Works

Components look familiar:

```
export const Main = component$(() => {
  const state = useStore({
    message: 'hello',
  });

  return (
    <input
      value={state.message}
      onInput$={e => state.message = e.target.value}
    />
  );
});
```

But load in a unique way:

```
<div on:input="./path-to-input-handler.js">
  <input value="hello" />
</div>
```

On the server, JS paths are encoded in HTML, so they don't have to download in browser until needed

```
<script>
  for (const event of events) {
    document.addEventListener(event, e => {
      const target = e.target.closest(`on:${event}`)
      if (target) {
        const jsPath = target.getAttribute(`[on\\:${event}]`)
        import(jsPath).then(mod => mod.default(e))
      }
    })
  }
</script>
```

With a tiny bit of code that looks similar to the above, that can be the *only* JS your page needs to become interactive

Source : Hydration is Pure Overhead: MIŠKO HEVERY

Qwik

DX **Small** **Compiled**