QUOIN

<title>Server Side Rendering with React and Redux</title>

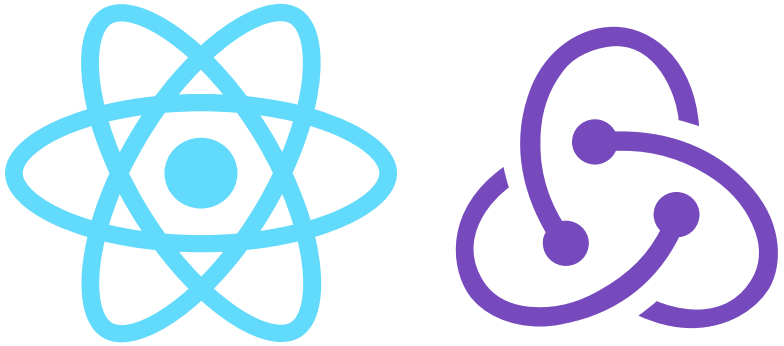
Presented by Fiax

Date: 2020-08-20

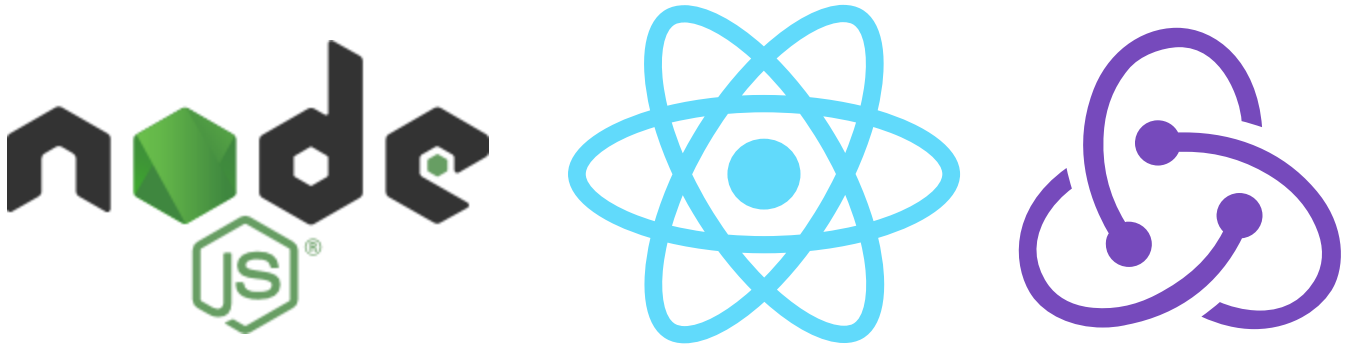
<h3>Introduction</h3>

  Traditional Single Page Application (SPA) flow in React
  Server Side Rendering (SSR) challenges
  SSR concepts graphics and diagrams
  Analize snapshot codes

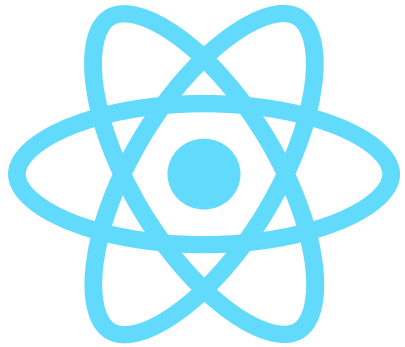
Technologies



Technologies



Technologies

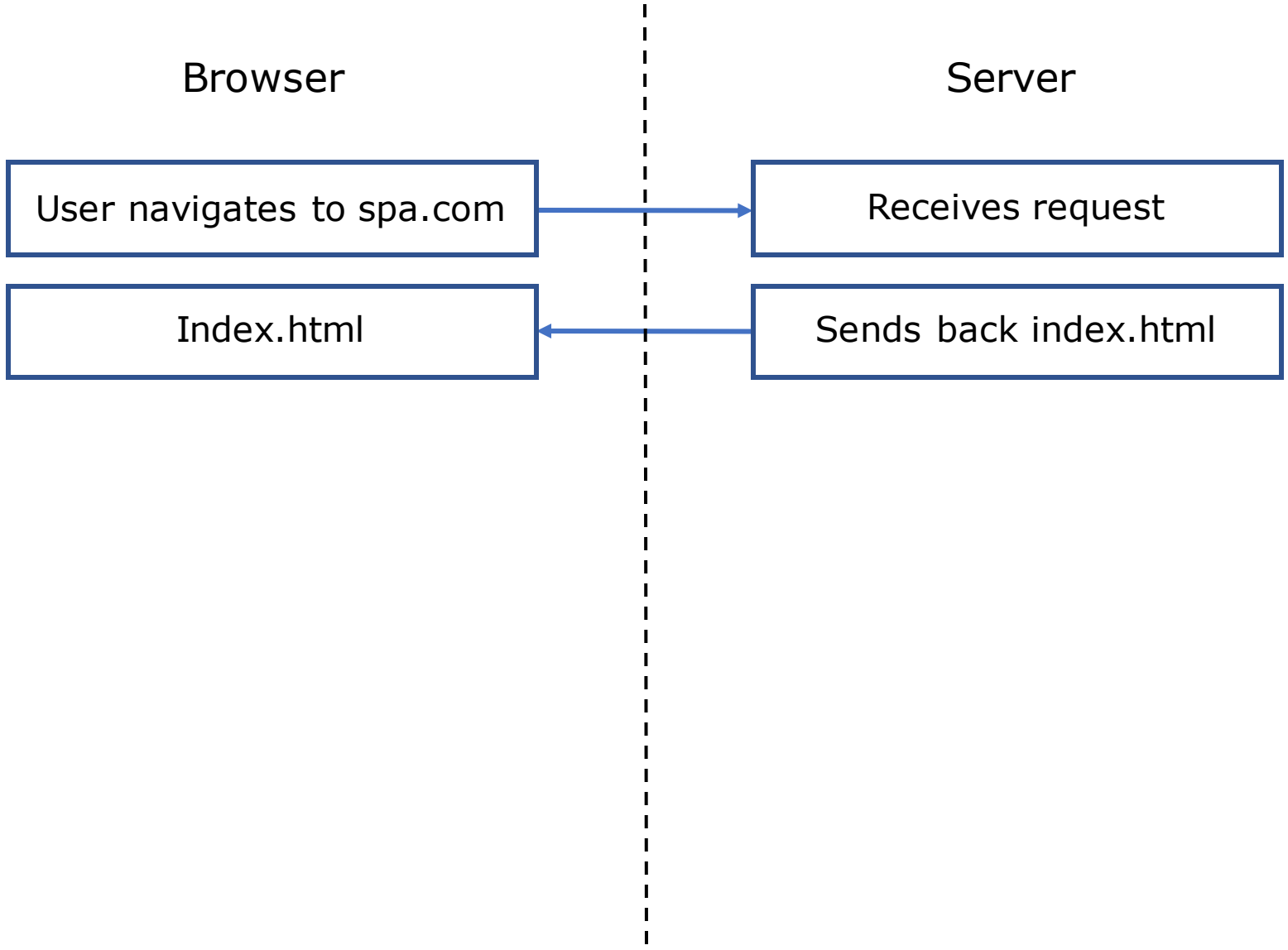


<h2>Why Server Side Rendering (SSR) is needed?</h2>

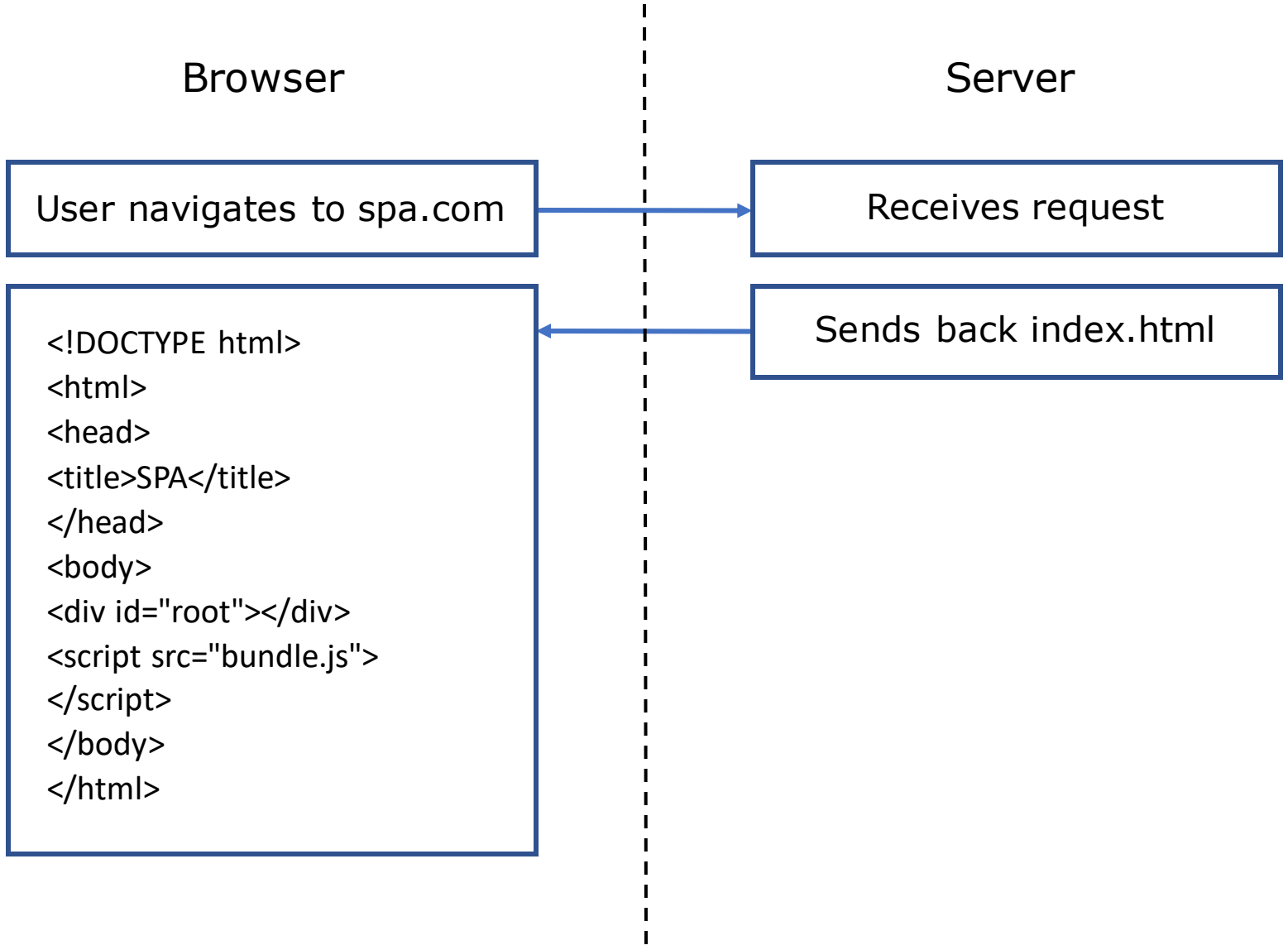
<h1>Javascript ✊</h1>

<pre>Browsers are more powerful. We started to create entire websites and web apps with client-side JavaScript. We started to call this "Single Page Application" (SPA).</pre>

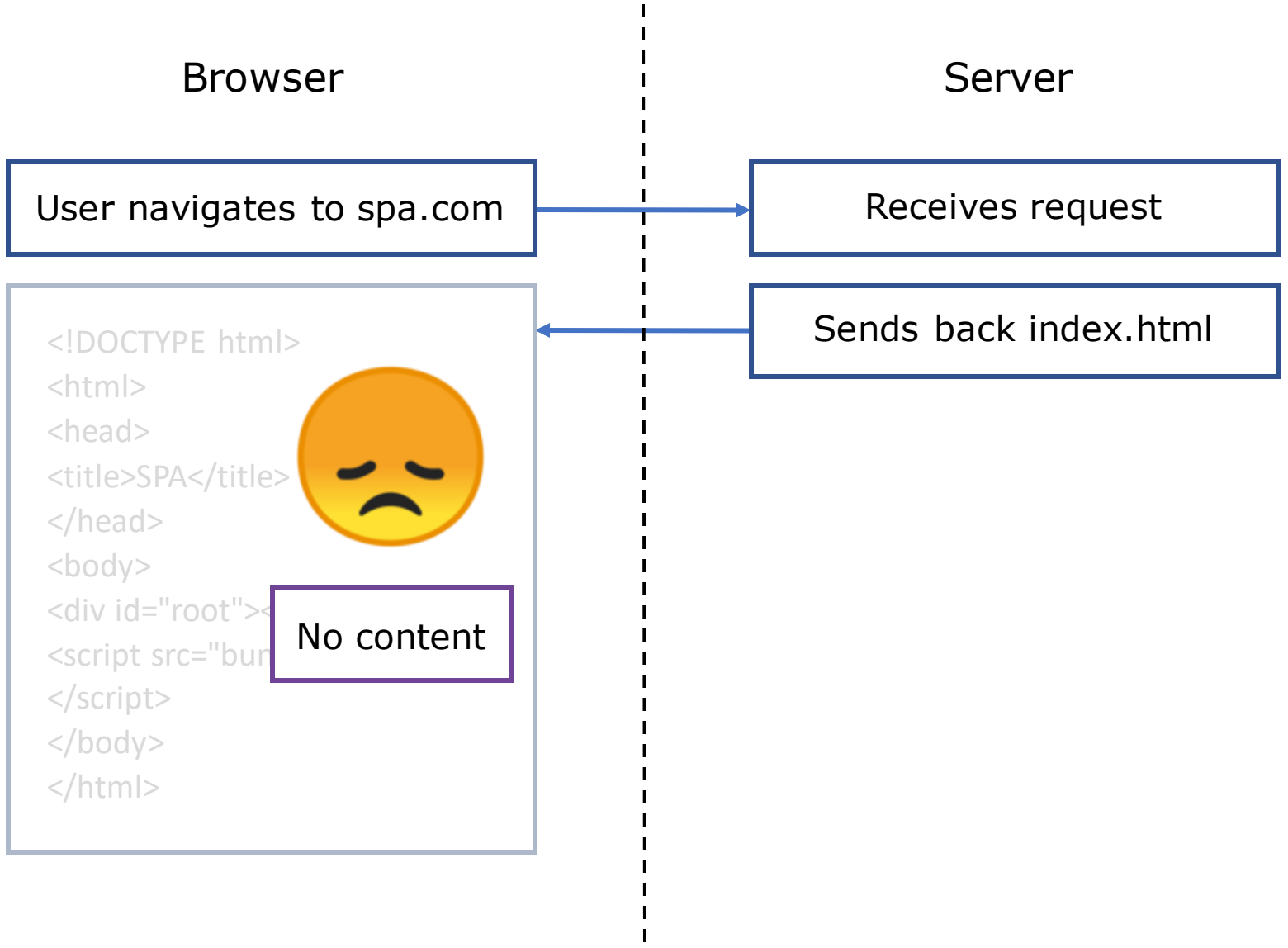
Single Page Application



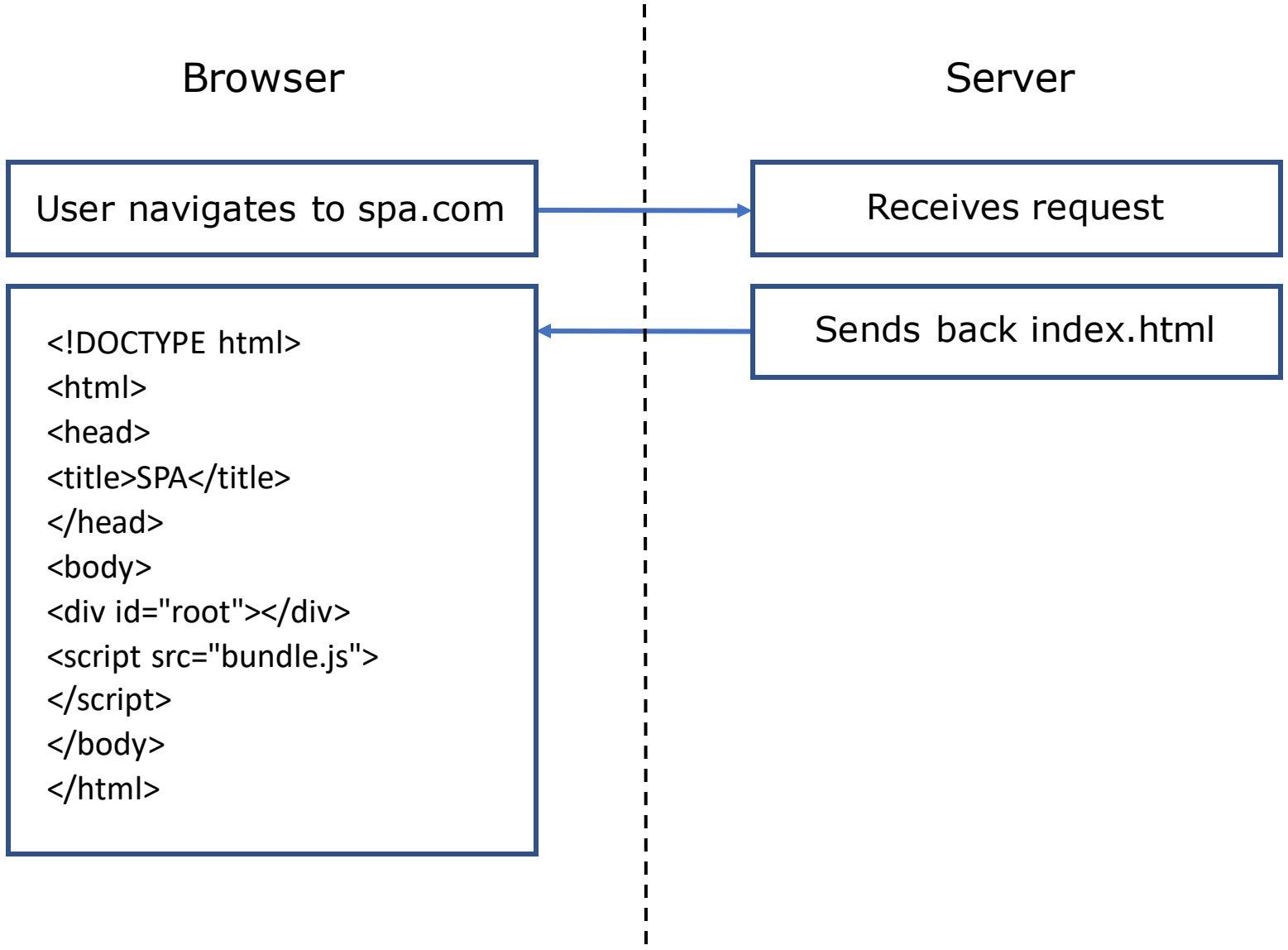
Single Page Application



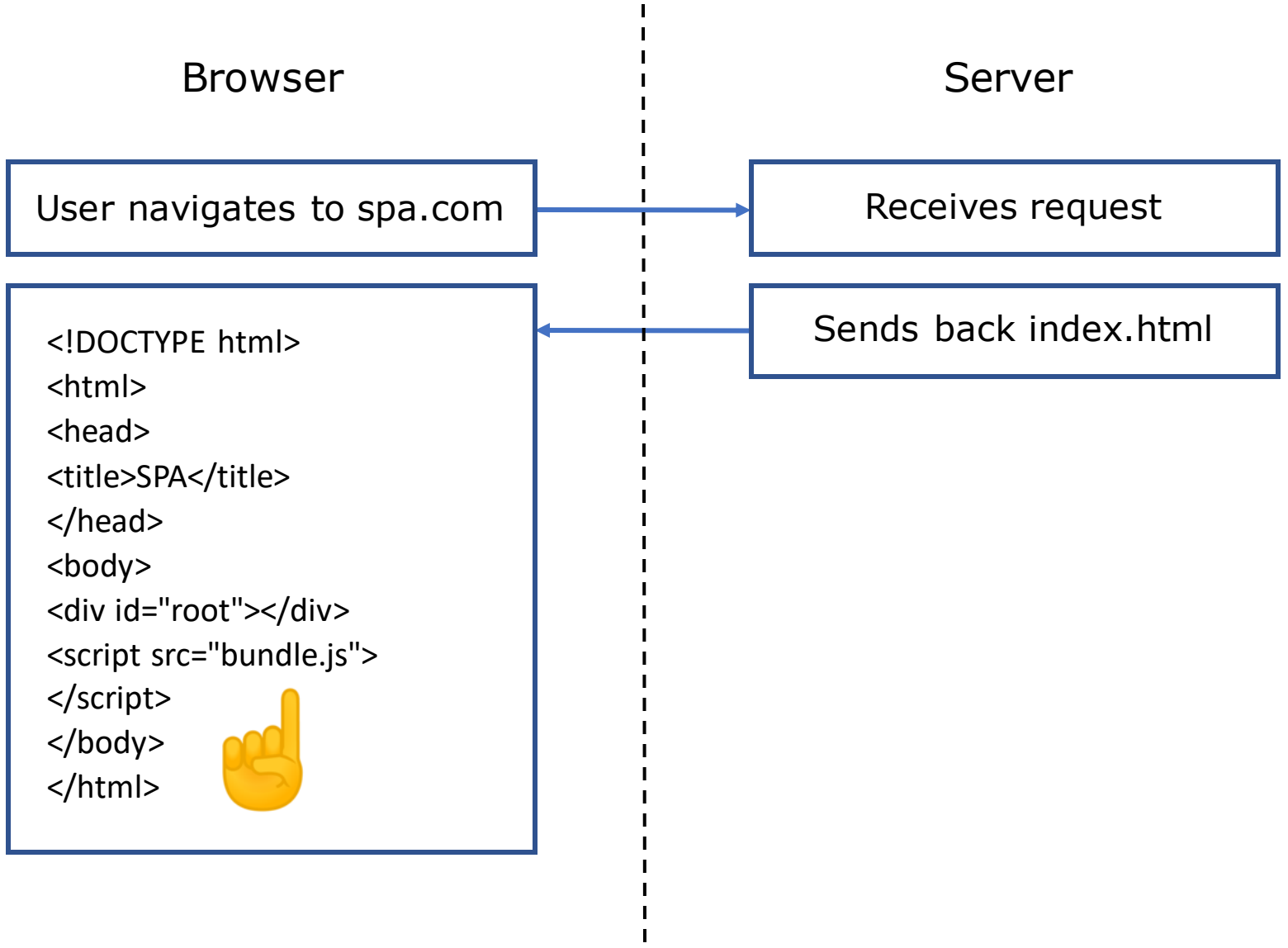
Single Page Application



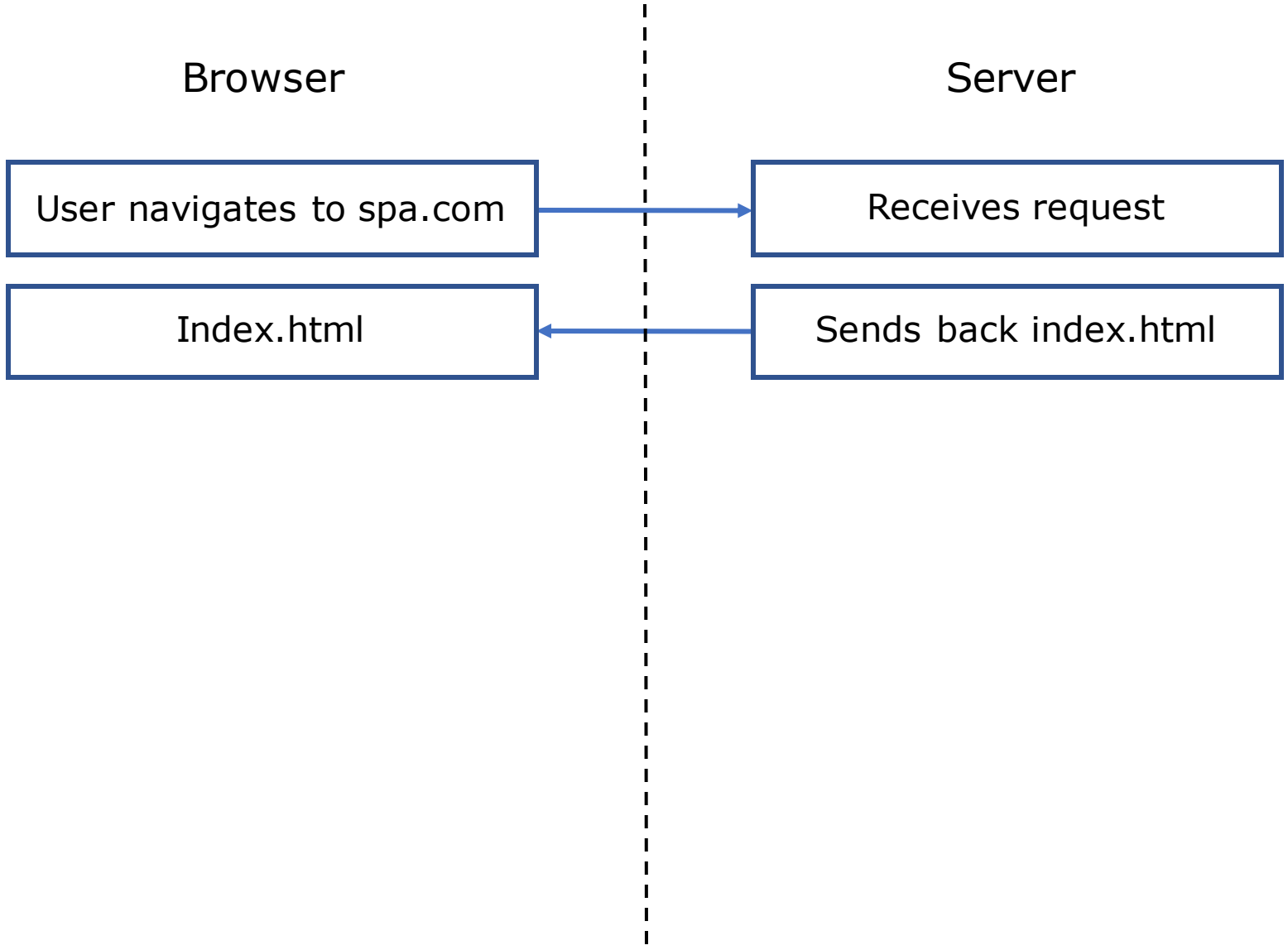
Single Page Application



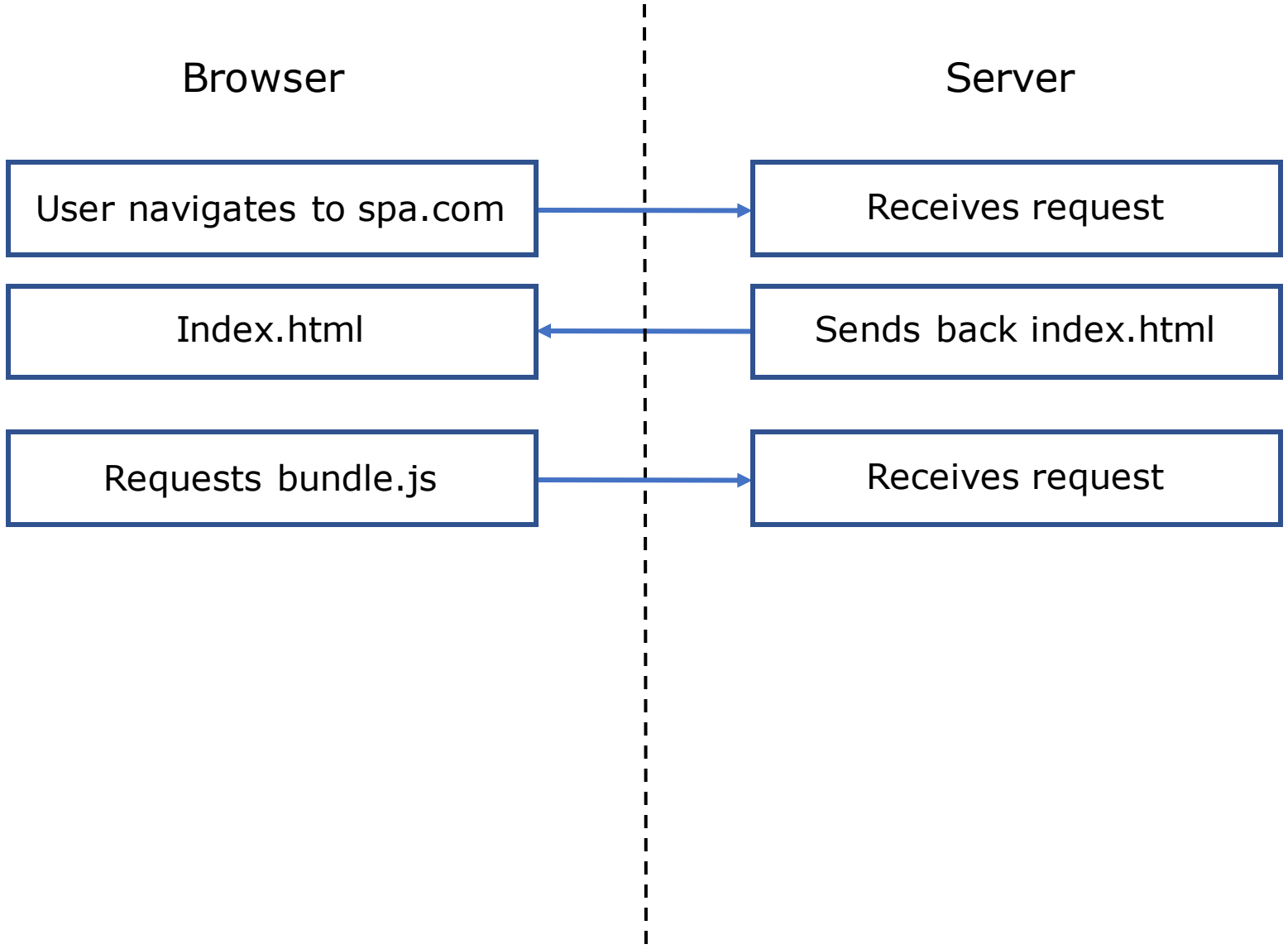
Single Page Application



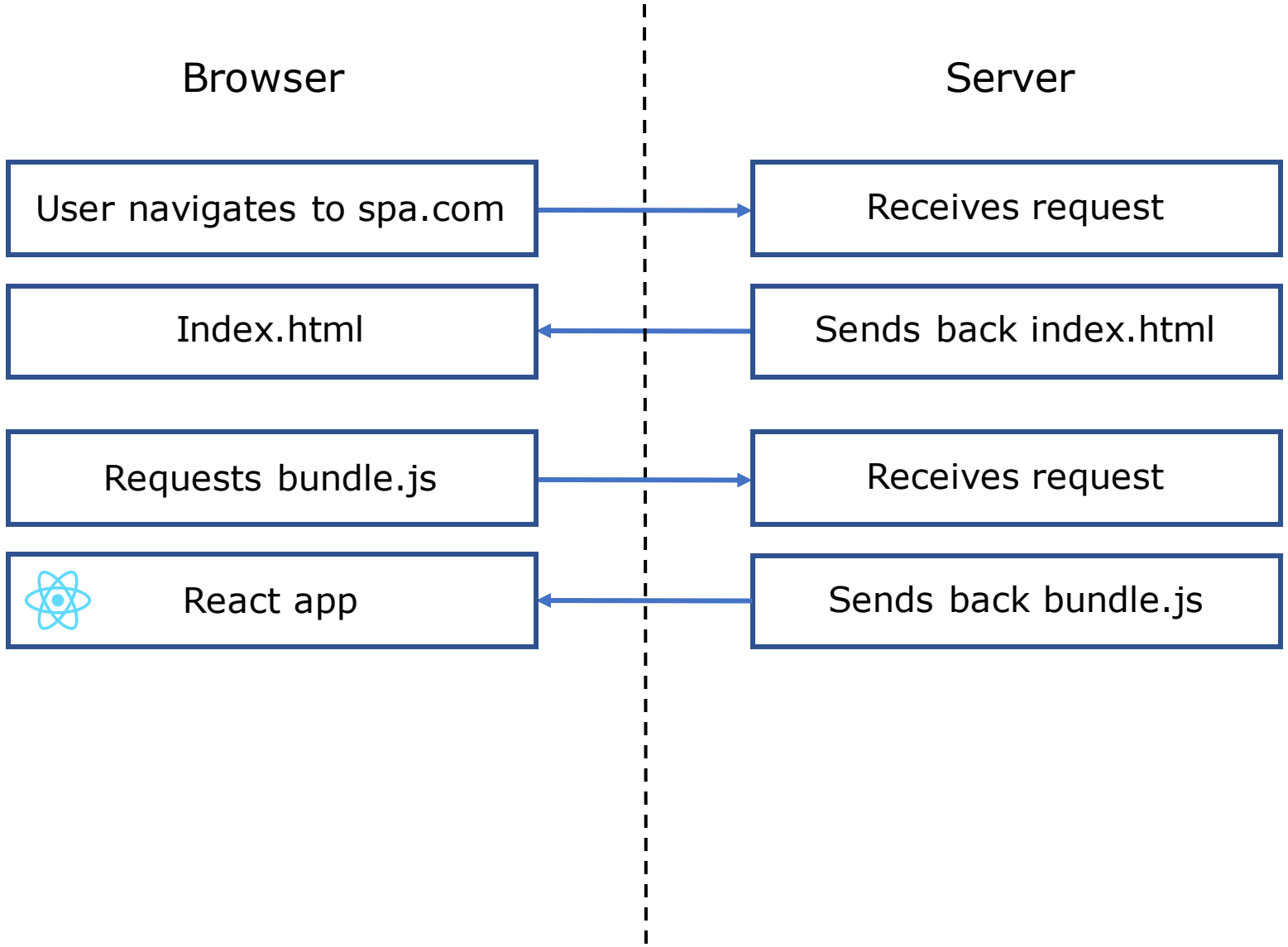
Single Page Application



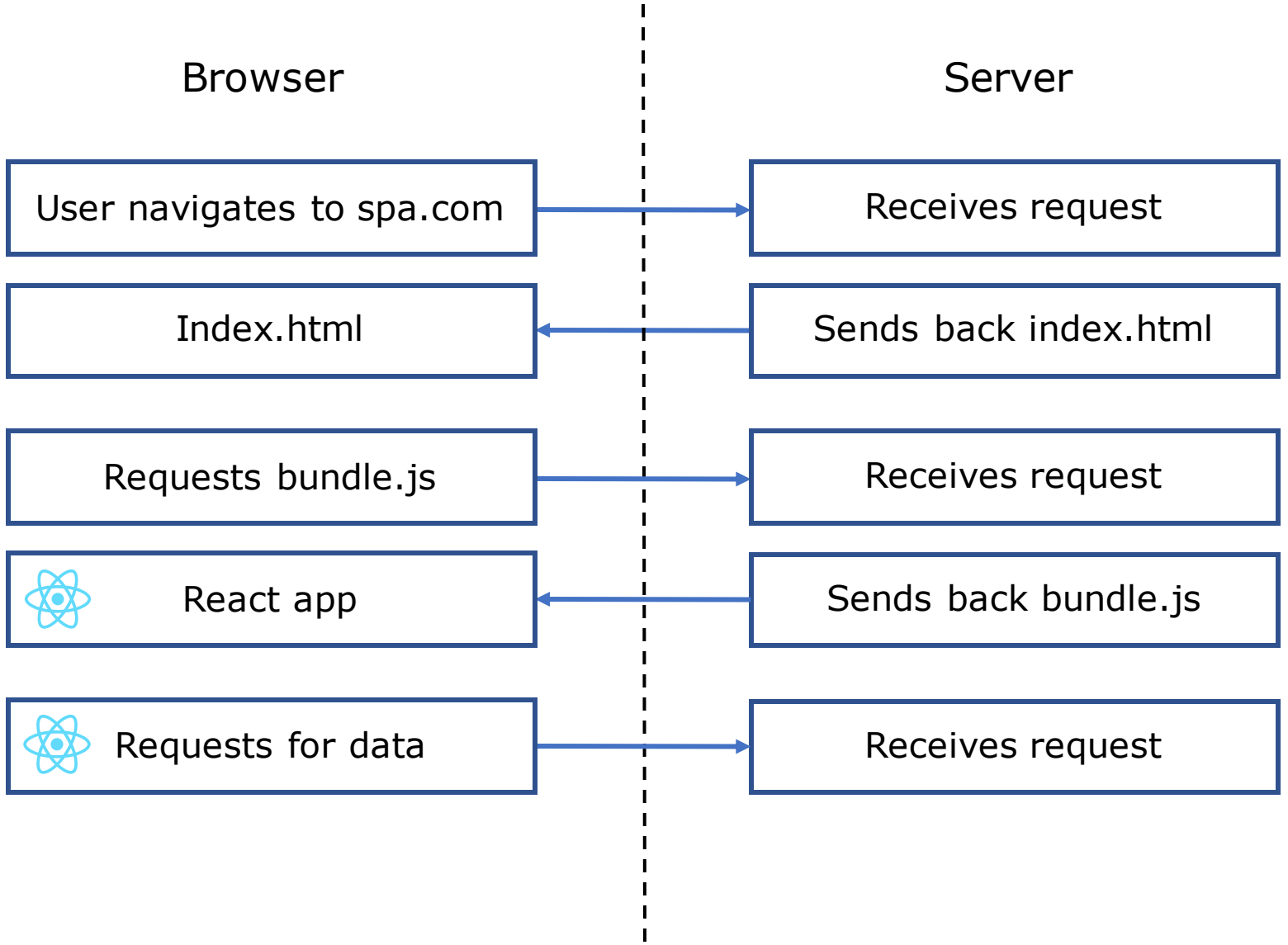
Single Page Application



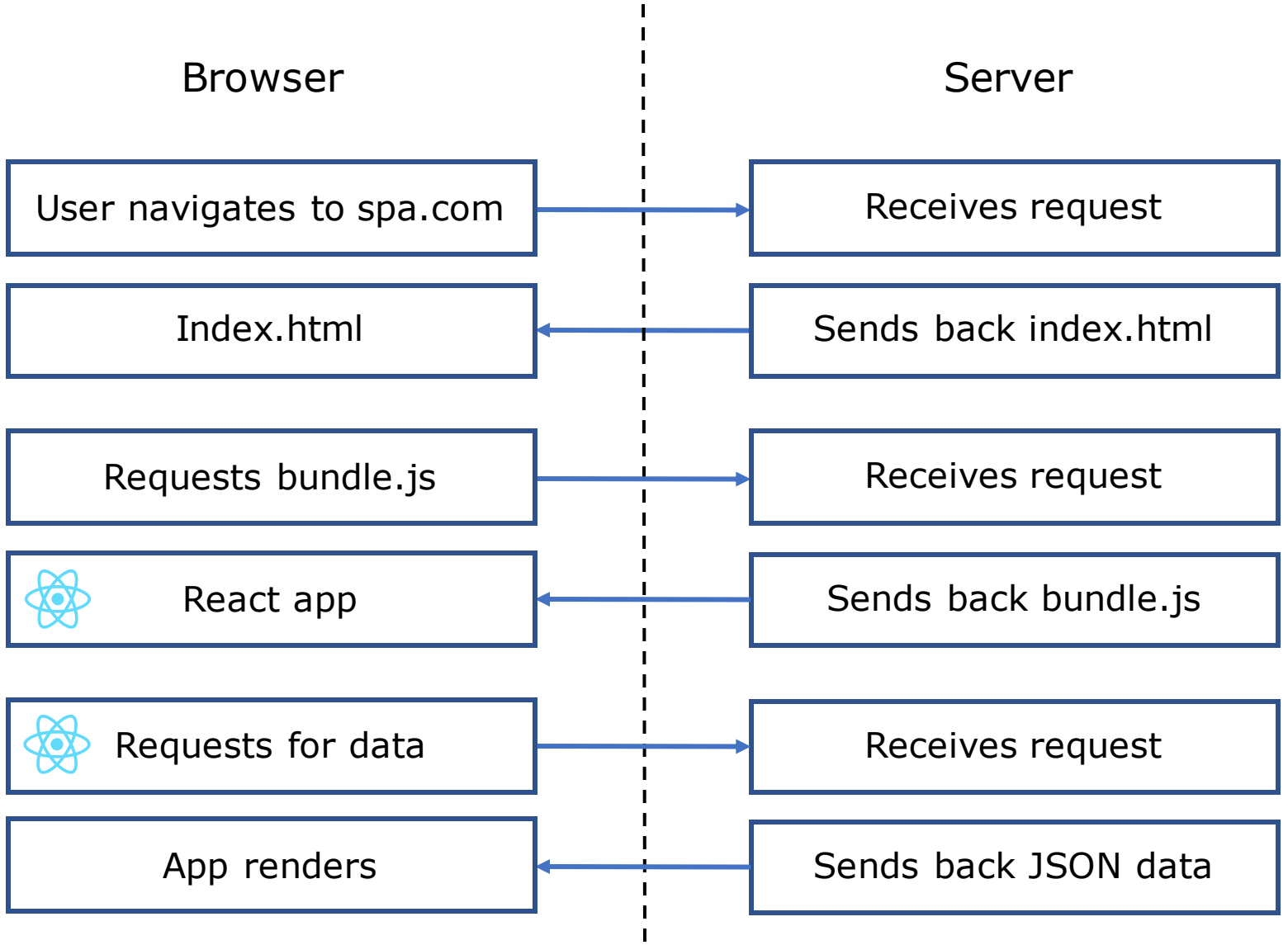
Single Page Application



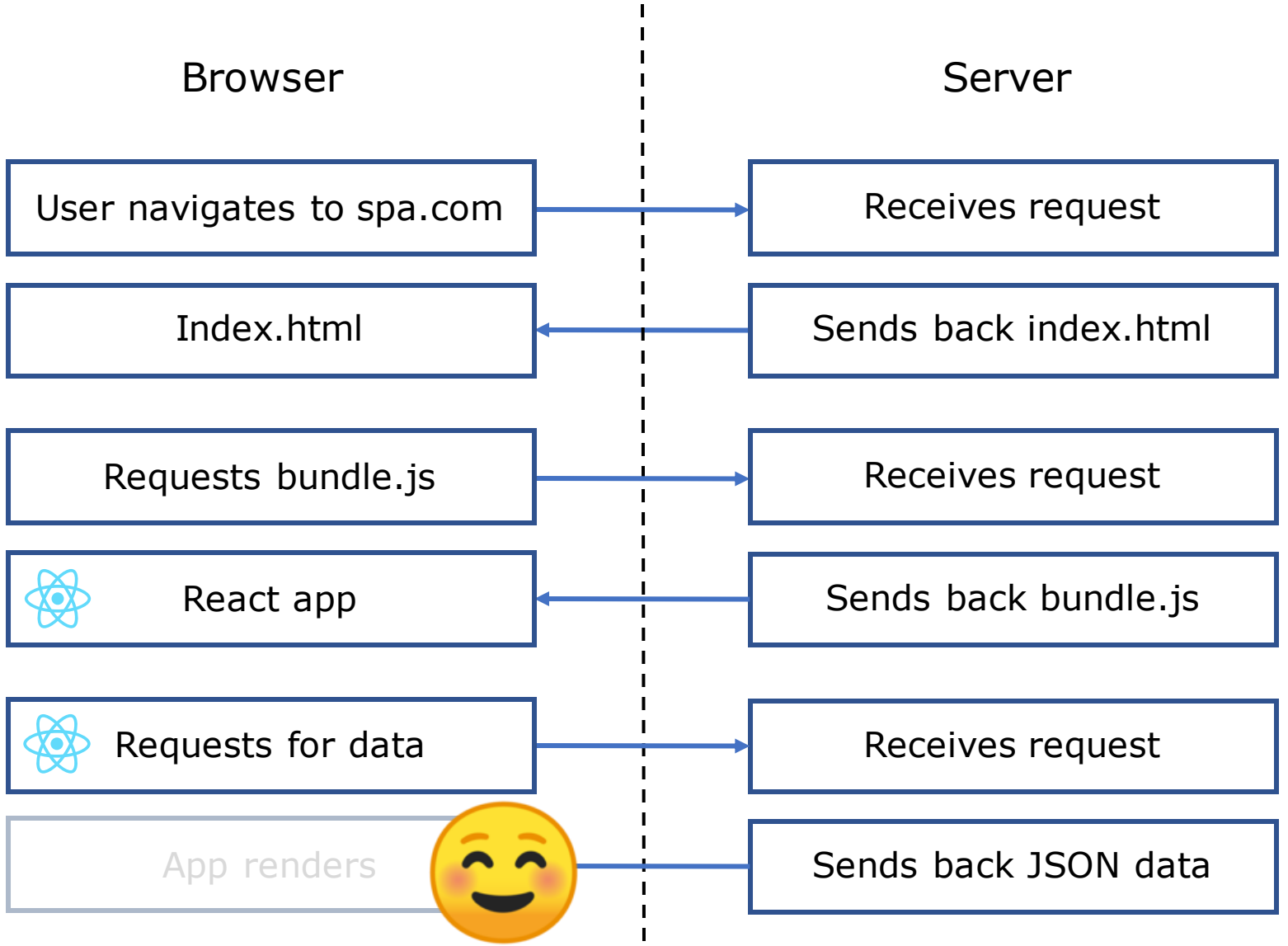
Single Page Application



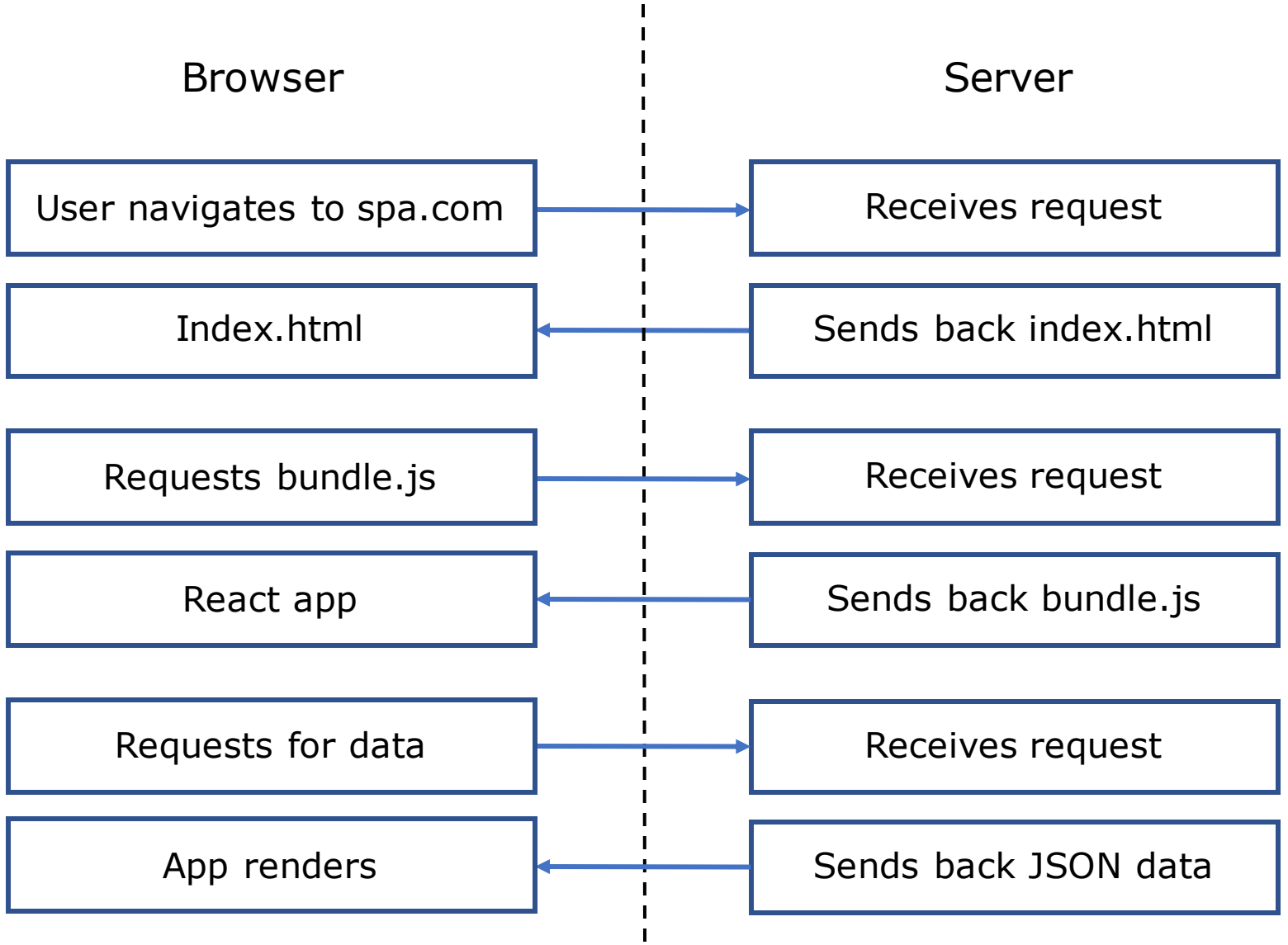
Single Page Application



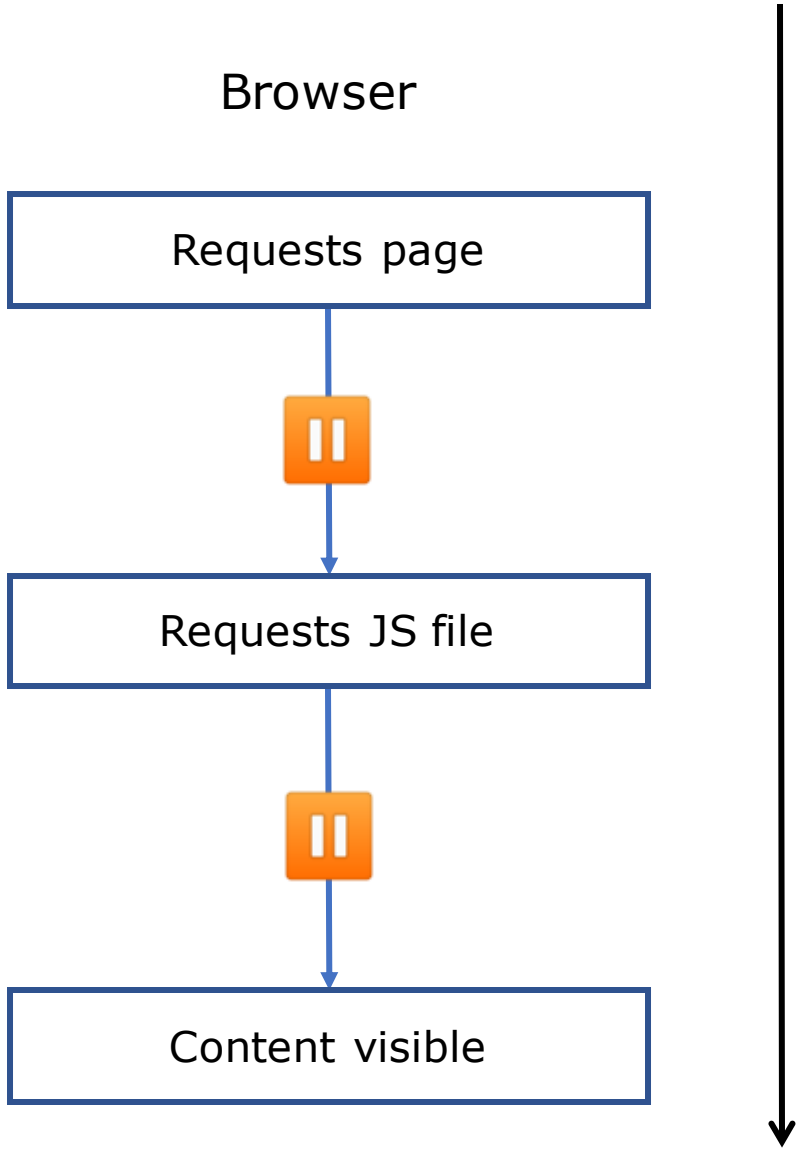
Single Page Application



Single Page Application

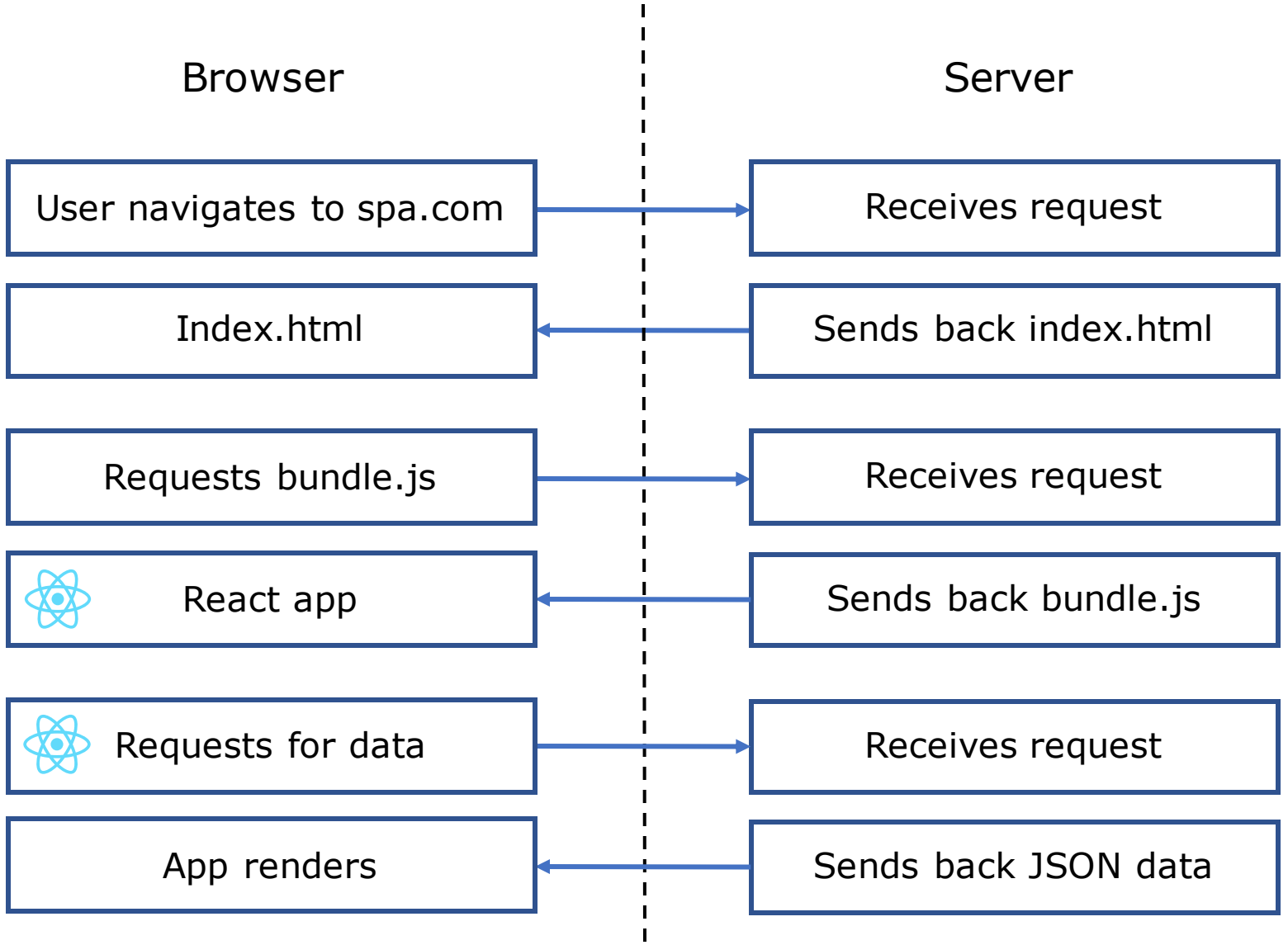


Single Page Application

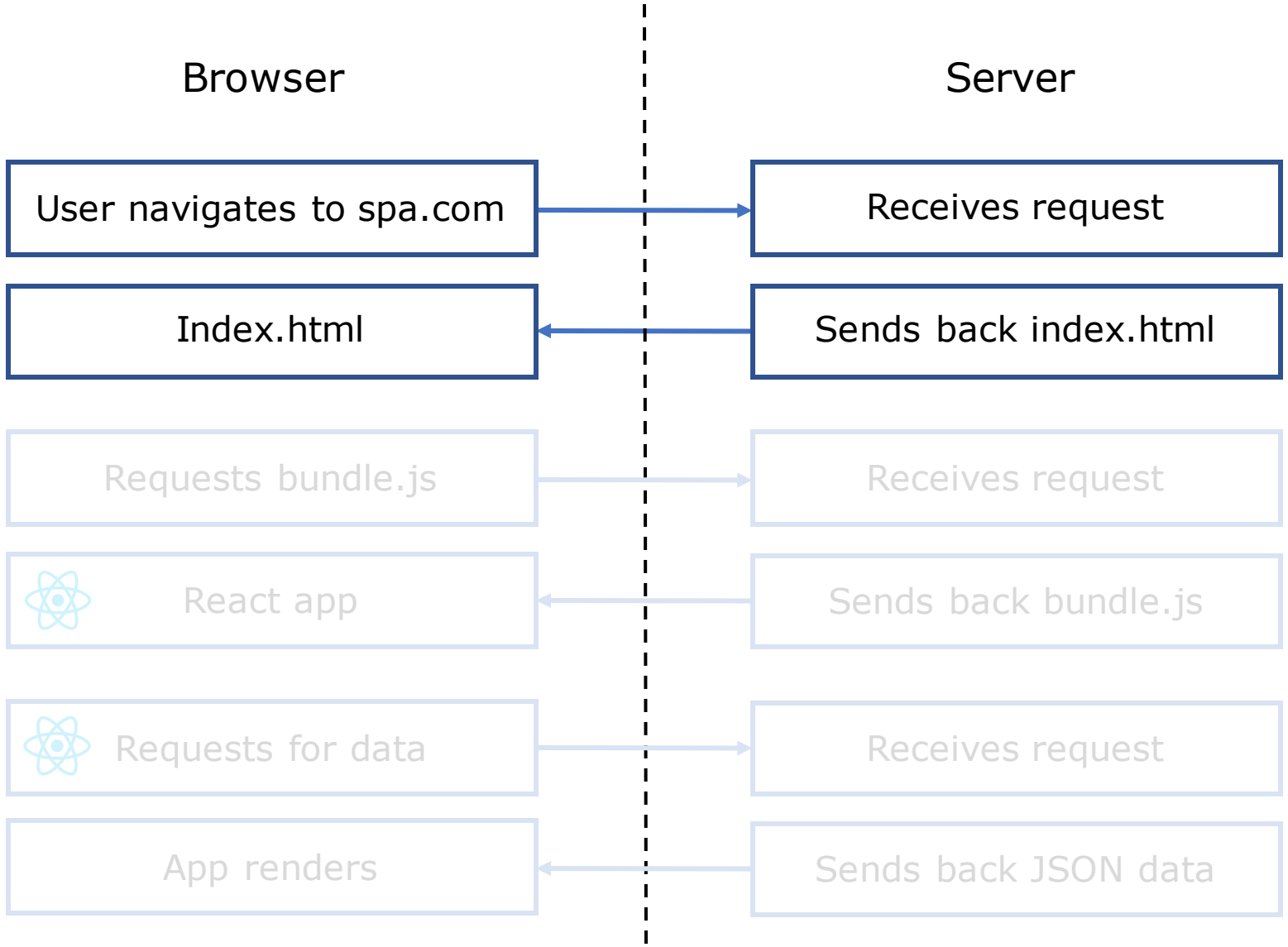


<h2>Why SSR is needed?</h2>

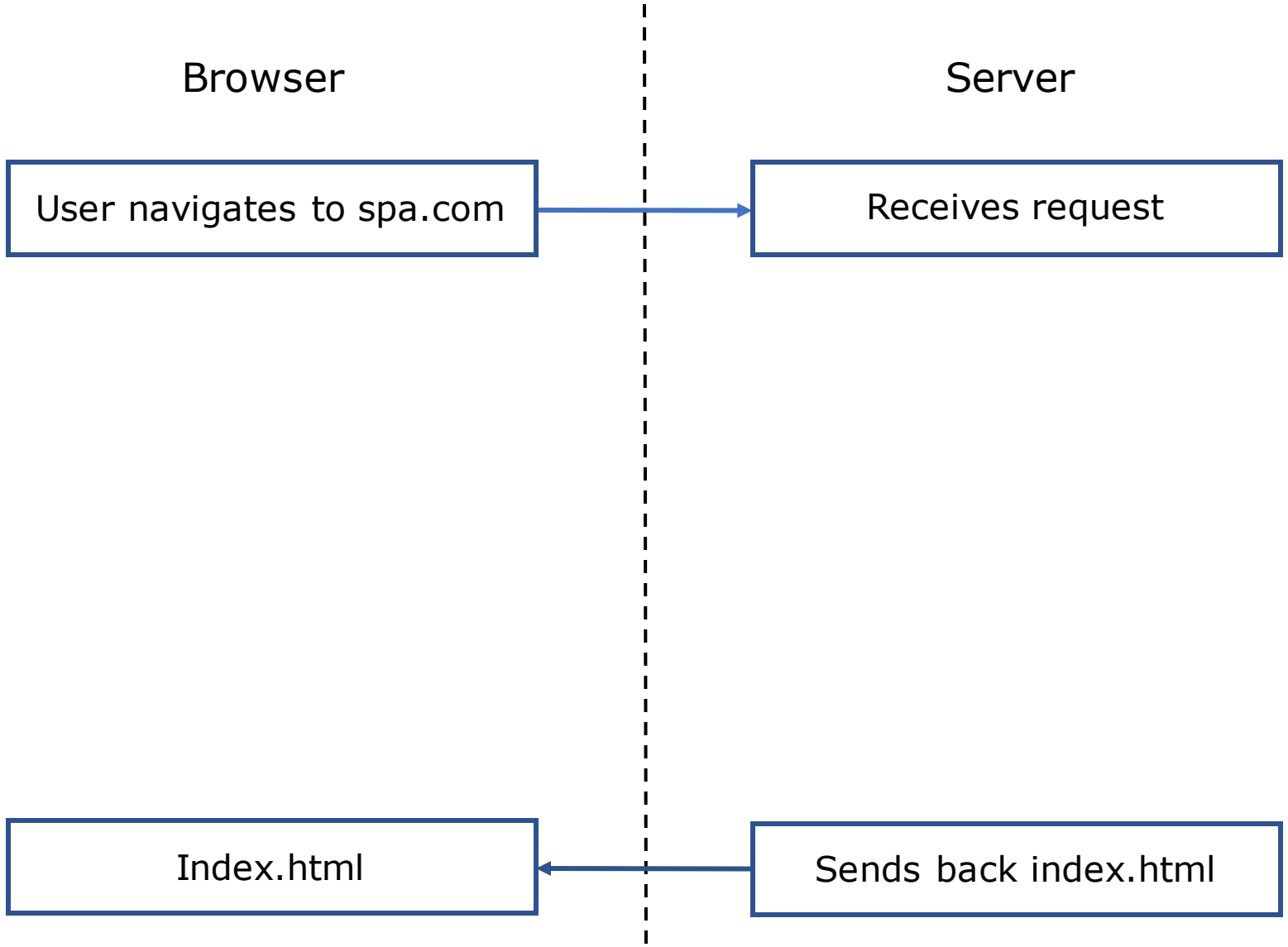
Server Side Rendering



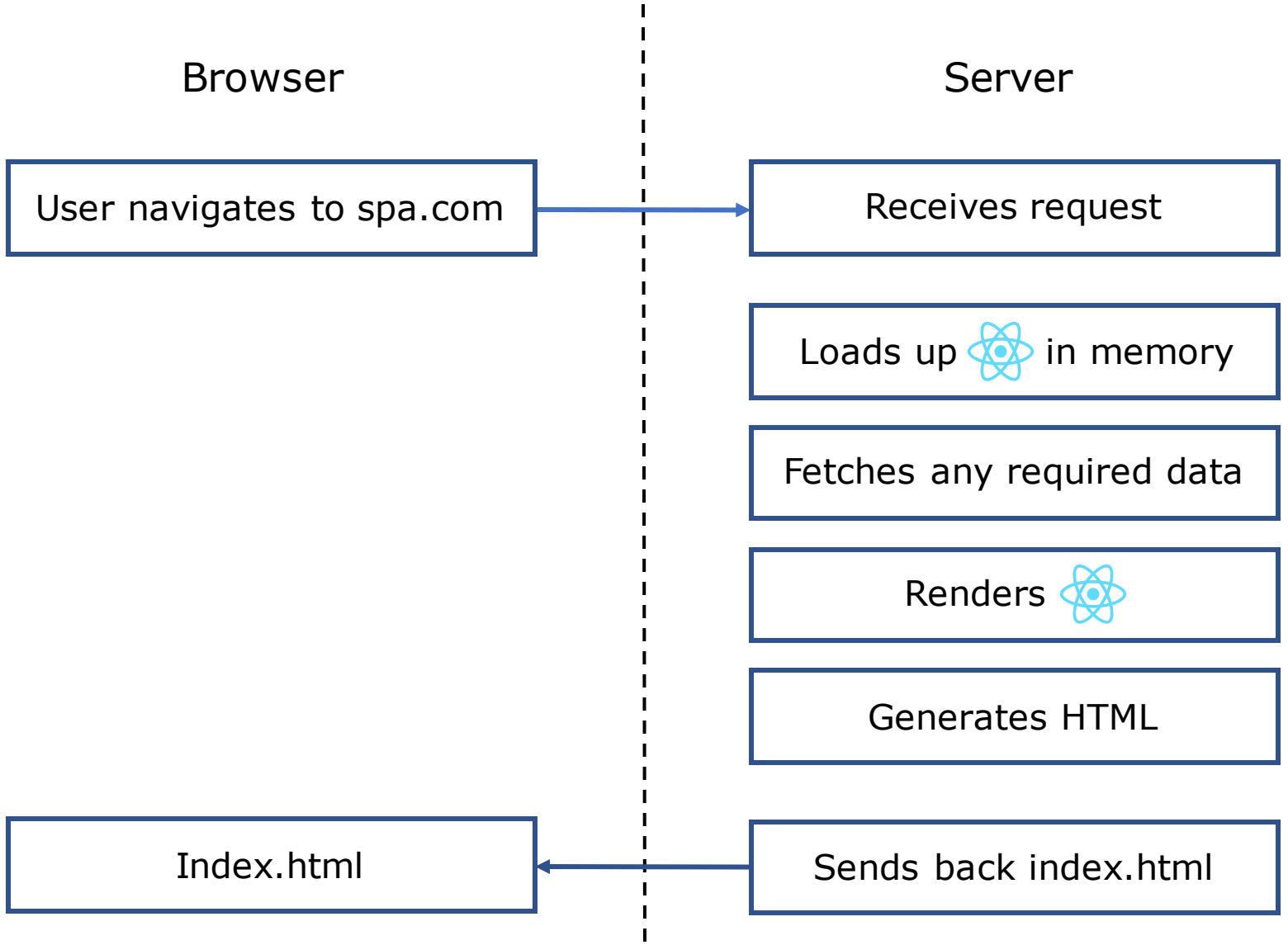
Server Side Rendering



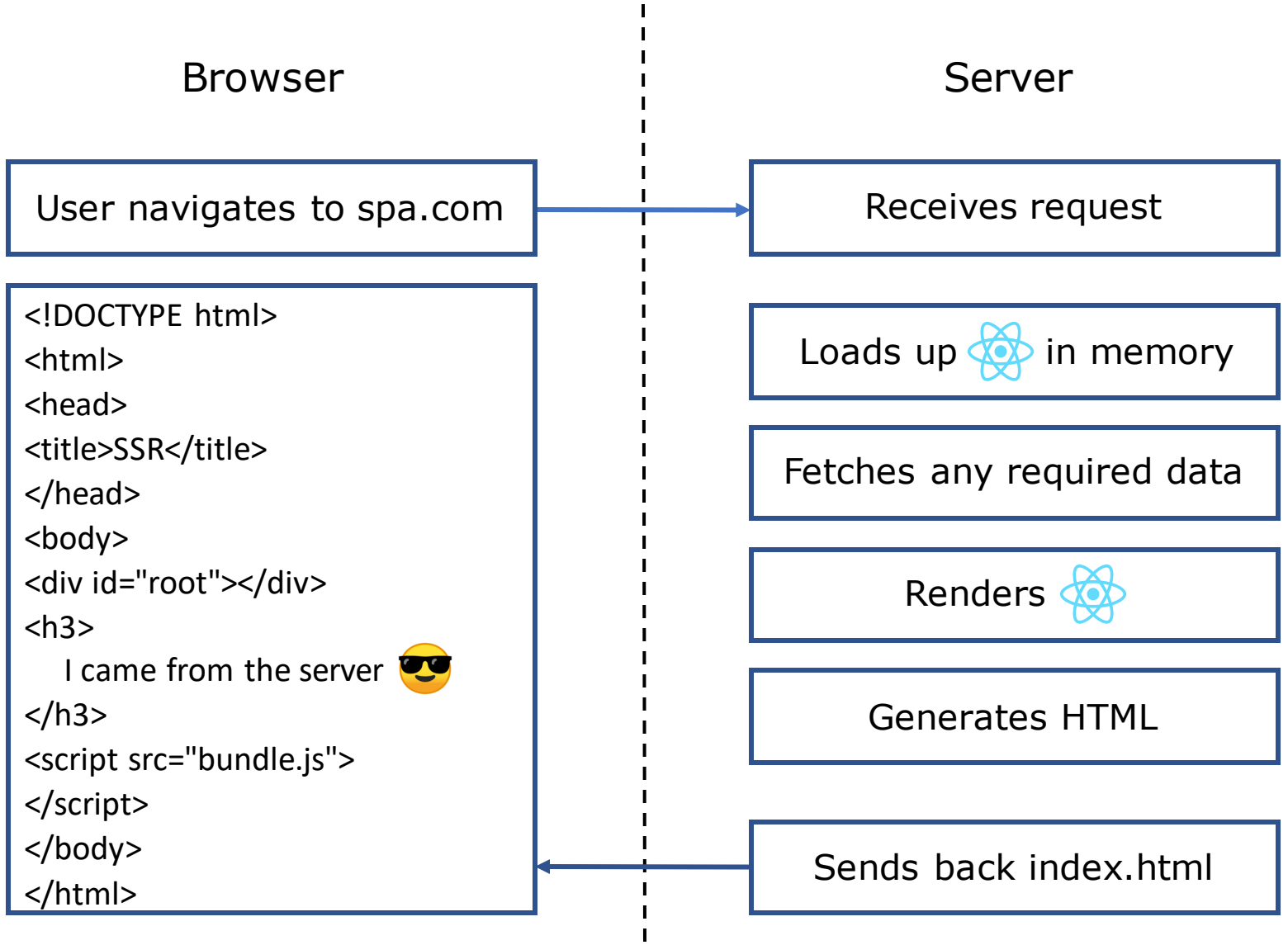
Server Side Rendering



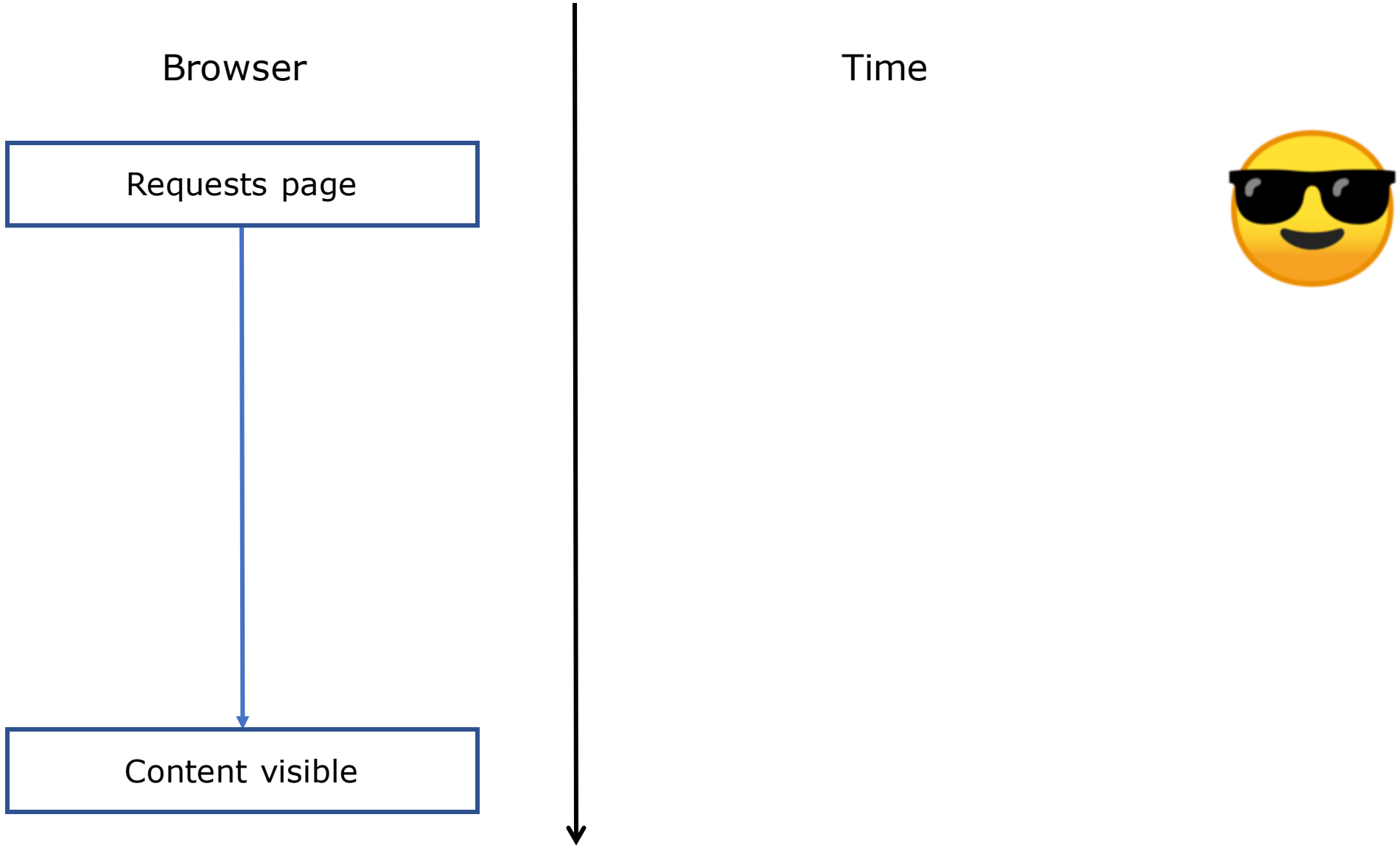
Server Side Rendering



Server Side Rendering



Server Side Rendering



SSR + SPA

SSR + SPA =

SSR + SPA = Universal App

<h3>Benefits from SSR</h3>

Faster times for the initial page render
Fully indexable HTML pages

<h3>Challenges on SSR</h3>

  JSX on the server
  Need to turn components into HTML
  Need state rehydration on the browser
  Redux needs different configuration on browser vs server
  Routing on the server
  Need to detect when all initial data load are completed on server

<h3>JSX on the server</h3>

```
// All imports required in server.js file


const app = express()

app.get('*', (request, response) => {
        const content = renderToString(<App />)

        res.send(content)
})

app.listen(port, () => {
        console.log(`Server listening to port ${port} on ${host}`)
})
```

<h3>JSX on the server</h3>

```
// All imports required in server.js file


const app = express()

app.get('*', (request, response) => {
        const content = renderToString(<App />)

        res.send(content)
})

app.listen(port, () => {
        console.log(`Server listening to port ${port} on ${host}`)
})
```

<h3>JSX on the server</h3>

```
// All imports required in server.js file


const app = express()

app.get('*', (request, response) => {
        const content = renderToString(<App />)

        res.send(content)
})

app.listen(port, () => {
        console.log(`Server listening to port ${port} on ${host}`)
})
```

<h3>JSX on the server</h3>

```
// All imports required in server.js file


const app = express()

app.get('*', (request, response) => {
        const content = renderToString(<App />)

        res.send(content)
})

app.listen(port, () => {
        console.log(`Server listening to port ${port} on ${host}`)
})
```

<h3>JSX on the server</h3>

```
// All imports required in server.js file


const app = express()

app.get('*', (request, response) => {
        const content = renderToString(<App />)

        res.send(content)
})

app.listen(port, () => {
        console.log(`Server listening to port ${port} on ${host}`)
})
```

$ node server.js

<h3>JSX on the server</h3>

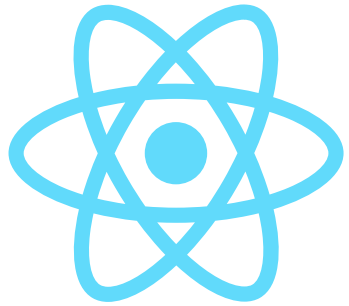
```
// All imports required in server.js file


const app = express()

app.get('*', (request, response) => {
        const content = renderToString(<App />)

        res.send(content)
})

app.listen(port, () => {
        console.log(`Server listening to port ${port} on ${host}`)
})
```

$ node server.js

Syntax Error: Unexpected token

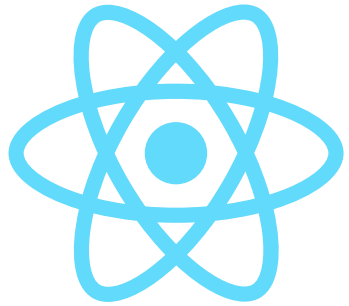
<h3>JSX on the server</h3>

```
// All imports required in server.js file


const app = express()

app.get('*', (request, response) => {
        const content = renderToString(<App />)

        res.send(content)
})

app.listen(port, () => {
        console.log(`Server listening to port ${port} on ${host}`)
})
```



<h3>JSX on the server</h3>

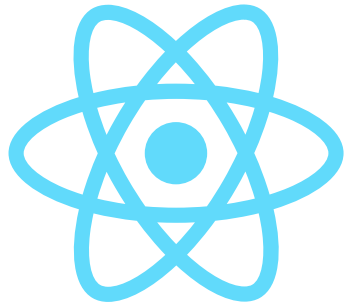
```
// All imports required in server.js file
import { renderToString } from 'react-dom/server'

const app = express()

app.get('*', (request, response) => {
        const content = renderToString(<App />)
        const html = `<html>
                        <head></head>
                        <body>
                                <div id="root">${content}</div>
                                <script src="bundle.js"></script>
                        </body>
                        </html>`
        res.send(html )
})
```

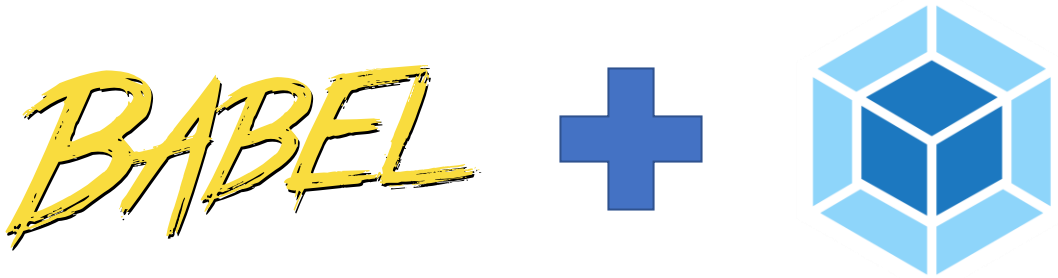


Challenge: Need to turn components into HTML

<h3>JSX on the server</h3>

```
// All imports required in server.js file


const app = express()

app.get('*', (request, response) => {
        const content = renderToString(<App />)

        res.send(content)
})

app.listen(port, () => {
        console.log(`Server listening to port ${port} on ${host}`)
})
```




BABEL

```
// webpack.base.js file

module.exports = {

        module: {
                rules: [{
                        test: /\.js?$/,
                        loader: 'babel-loader',
                        exclude: /node_modules/,
                        options: {
                                presets: [
                                        'react',
                                        'stage-0',
                                        ['env', { targets: { browsers: ['last 2 versions'] } }]
                                ]
                        }
                }]
        }
}
```

```
// webpack.client.js file

const path = require('path')
const merge = require('webpack-merge')
const baseConfig = require('./webpack.base.js')

const config = {

    entry: './src/client/client.js',

    output: {
        filename: 'bundle.js',
        path: path.resolve(__dirname, 'public')
    }
}

module.exports = merge(baseConfig, config)
```

```
// webpack.server.js file

const path = require('path')
const merge = require('webpack-merge')
const baseConfig = require('./webpack.base.js')

const config = {
        target: 'node',
        entry: './src/client/client.js', entry: './src/server.js',

        output: {
                filename: 'bundle.js',
                path: path.resolve(__dirname, 'public') path: path.resolve(__dirname, 'build')
        }
}

module.exports = merge(baseConfig, config)
```

<h3>JSX on the server</h3>

```
// All imports required in server.js file

const app = express()

app.get('*', (request, response) => {
        const content = renderToString(<App />)

        res.send(content)
})

app.listen(port, () => {
        console.log(`Server listening to port ${port} on ${host}`)
})
```

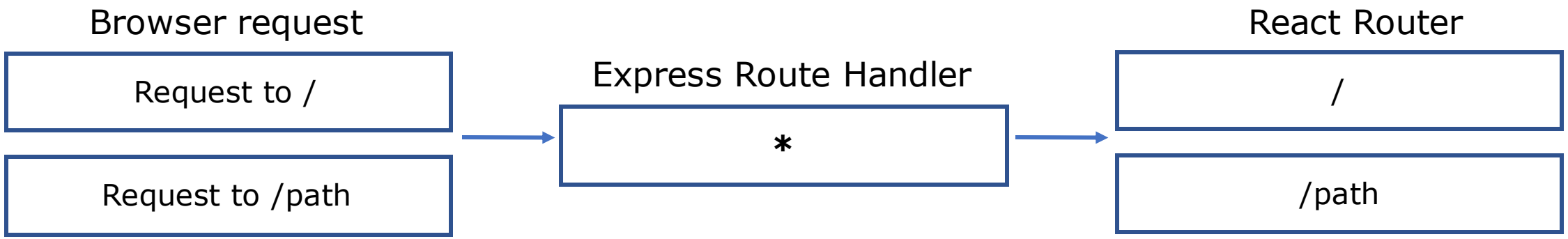

```
$ node bundle.js
```

<h3>Routing</h3>

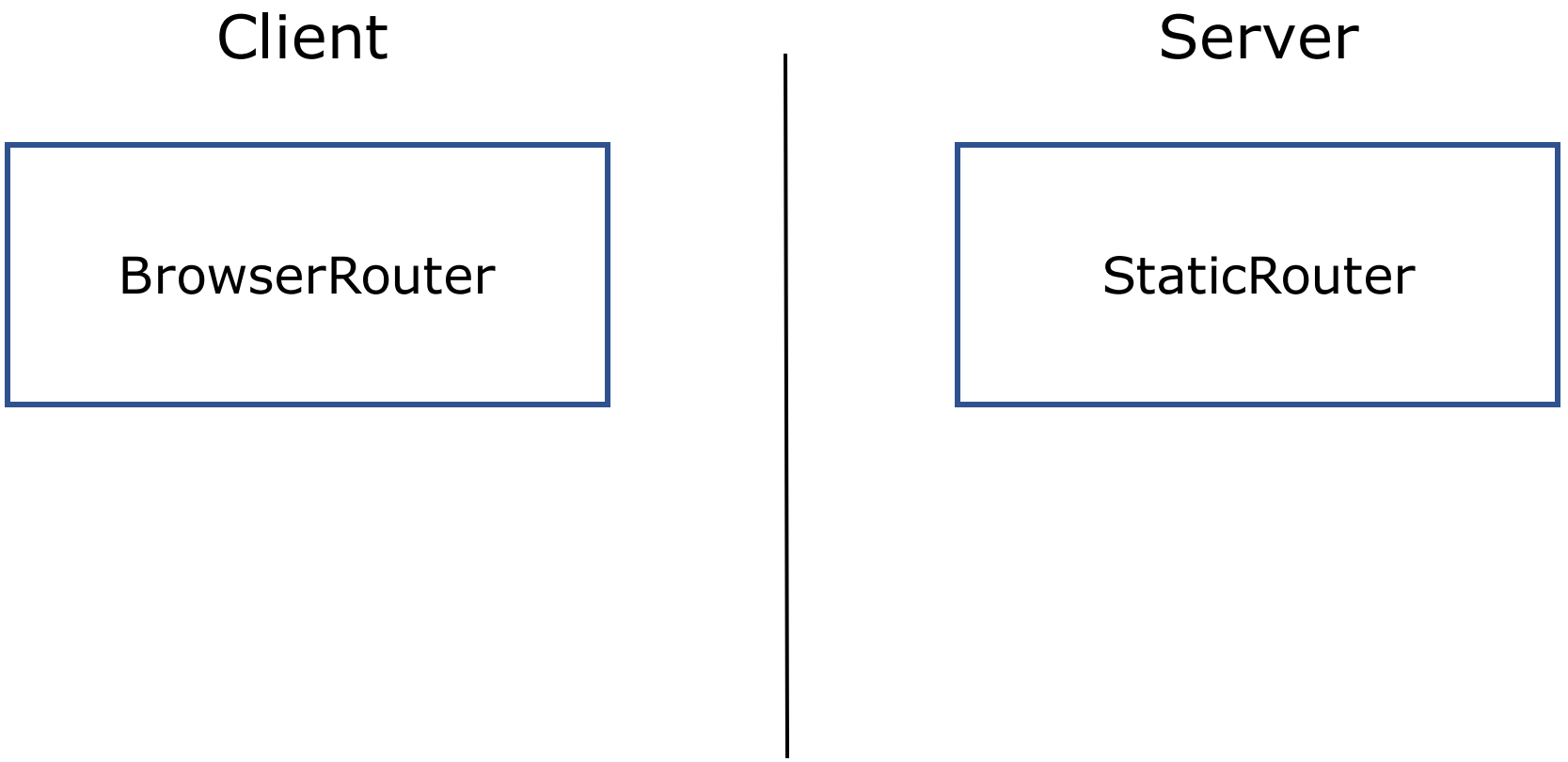
```
// All imports required in server.js file

const app = express()

app.get('*', (request, response) => {
        const content = renderToString(<App />)

        res.send(content)
})

app.listen(port, () => {
        console.log(`Server listening to port ${port} on ${host}`)
})
```

```
$ node bundle.js
```

<h3>Router</h3>


Browser request
Request to /
Request to /path
Express Route Handler
*
React Router
/
/path

<h3>Router</h3>



<h3>Router</h3>

Client

```
import { renderRoutes } from 'react-router-config'
import { BrowserRouter} from 'react-router-dom'

ReactDOM.hydrate(
    <Provider store={store}>
        <BrowserRouter>
            {renderRoutes(Routes)}
        </BrowserRouter>
    </Provider>,
    document.querySelector('#root')
)
```

Server

```
import { renderRoutes } from 'react-router-config'
import { StaticRouter } from 'react-router-dom'

export default (request, store) => {
    const content = renderToString(
    <Provider store={store}>
        <StaticRouter location={request.url}>
            {renderRoutes(Routes)}
        </StaticRouter>
    </Provider>
)}
```

<h3>Router</h3>

Client

```
import { renderRoutes } from 'react-router-config'
import { BrowserRouter} from 'react-router-dom'

ReactDOM.hydrate(
    <Provider store={store}>
        <BrowserRouter>
            {renderRoutes(Routes)}
        </BrowserRouter>
    </Provider>,
    document.querySelector('#root')
)
```

Server

```
import { renderRoutes } from 'react-router-config'
import { StaticRouter } from 'react-router-dom'

export default (request, store) => {
    const content = renderToString(
    <Provider store={store}>
        <StaticRouter location={request.url}>
            {renderRoutes(Routes)}
        </StaticRouter>
    </Provider>
)}
```

<h3>Router</h3>

Client

```
import { renderRoutes } from 'react-router-config'
import { BrowserRouter} from 'react-router-dom'

ReactDOM.hydrate(
      <Provider store={store}>
            <BrowserRouter>
                  {renderRoutes(Routes)}
            </BrowserRouter>
      </Provider>,
      document.querySelector('#root')
)
```

Server

```
import { renderRoutes } from 'react-router-config'
import { StaticRouter } from 'react-router-dom'

export default (request, store) => {
      const content = renderToString(
      <Provider store={store}>
            <StaticRouter location={request.url}>
                  {renderRoutes(Routes)}
            </StaticRouter>
      </Provider>
)}
```

<h3>Routes</h3>

```
// Routes.js

export default [{
        component: App,
        loadData: ({ dispatch }) => dispatch(fetchData()),
        routes: [
                {
                        component: Home,
                        path: '/',
                        exact: true
                },
                {
                        component: _Component,
                        path: '/path'
                }
        ]
}]
```

<h3>Routes</h3>

```
// Routes.js

export default [{
        component: App,
        loadData: ({ dispatch }) => dispatch(fetchData()),
        routes: [
                {
                        component: Home,
                        path: '/',
                        exact: true
                },
                {
                        component: _Component,
                        path: '/path'
                }
        ]
}]
```

<h3>Routes</h3>

```
// Routes.js

export default [{
    component: App,
    loadData: ({ dispatch }) => dispatch(fetchData()),
    routes: [
        {
            component: Home,
            path: '/',
            exact: true
        },
        {
            component: _Component,
            path: '/path'
        }
    ]
}]
```

Server

Time

Figure out what components would have rendered (based on URL)

Call 'loadData' function attached to each of those components

Detect all requests are completed

Render the app with the collected data

Send result to browser

```
/* ... */

const promises = matchRoutes(Routes, request.url).map(({ route }) =>
{
        return route.loadData ? route.loadData(store) : null
}).map(promise => {
        if (promise) {
                return new Promise((resolve, reject) => {
                promise.then(resolve).catch(resolve)
                })
        }
})

Promise.all(promises).then(() => {
        const reduxState = store.getState( )
        const html= htmlTemplate(request, reduxState )

        res.send(html)
})
```

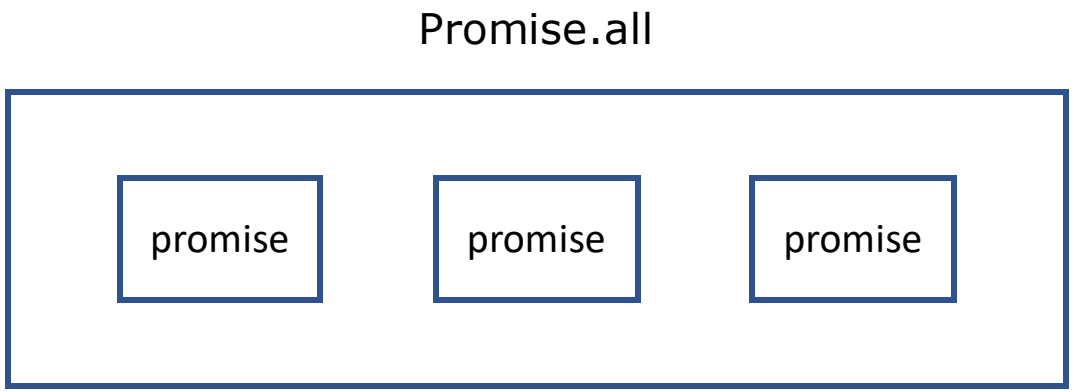
```
/* ... */

const promises = matchRoutes(Routes, request.url).map(({ route }) =>
{
        return route.loadData ? route.loadData(store) : null
}).map(promise => {
        if (promise) {
                return new Promise((resolve, reject) => {
                promise.then(resolve).catch(resolve)
                })
        }
})

Promise.all(promises).then(() => {
        const reduxState = store.getState( )
        const html= htmlTemplate(request, reduxState )

        res.send(html)
})
```

```
// Routes.js

export default [{
        component: App,
        loadData: ({ dispatch }) => dispatch(fetchData()),
        routes: [
                {
                        component: Home,
                        path: '/',
                        exact: true
                },
                {
                        component: _Component,
                        path: '/path'
                }
        ]
}]
```

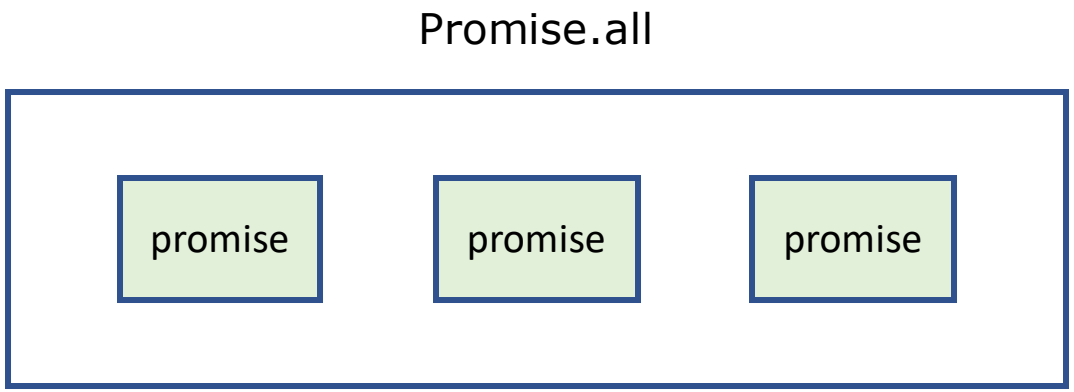
```
/* ... */

const promises = matchRoutes(Routes, request.url).map(({ route }) =>
{
        return route.loadData ? route.loadData(store) : null
}).map(promise => {
        if (promise) {
                return new Promise((resolve, reject) => {
                promise.then(resolve).catch(resolve)
                })
        }
})

Promise.all(promises).then(() => {
        const reduxState = store.getState( )
        const html= htmlTemplate(request, reduxState )

        res.send(html)
})
```

```
// Routes.js

export default [{
        component: App,
        loadData: ({ dispatch }) => dispatch(fetchData()),
        routes: [
                {
                        component: Home,
                        path: '/',
                        exact: true
                },
                {
                        component: _Component,
                        path: '/path'
                }
        ]
}]
```

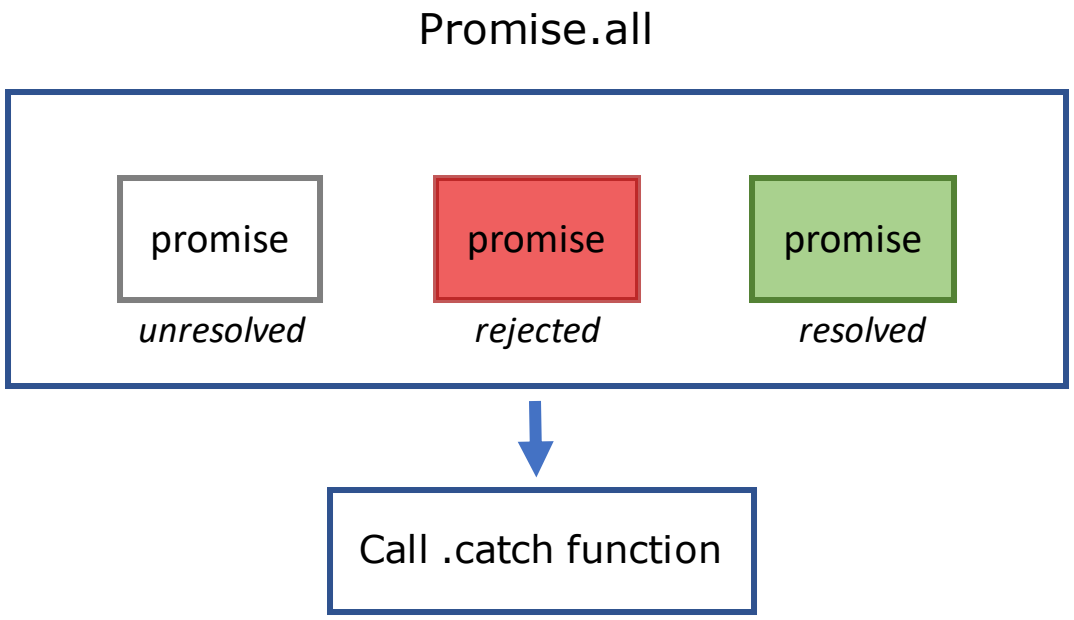
```
/* ... */

const promises = matchRoutes(Routes, request.url).map(({ route }) =>
{
        return route.loadData ? route.loadData(store) : null
}).map(promise => {
        if (promise) {
                return new Promise((resolve, reject) => {
                promise.then(resolve).catch(resolve)
                })
        }
})

Promise.all(promises).then(() => {
        const reduxState = store.getState( )
        const html= htmlTemplate(request, reduxState )

        res.send(html)
})
```



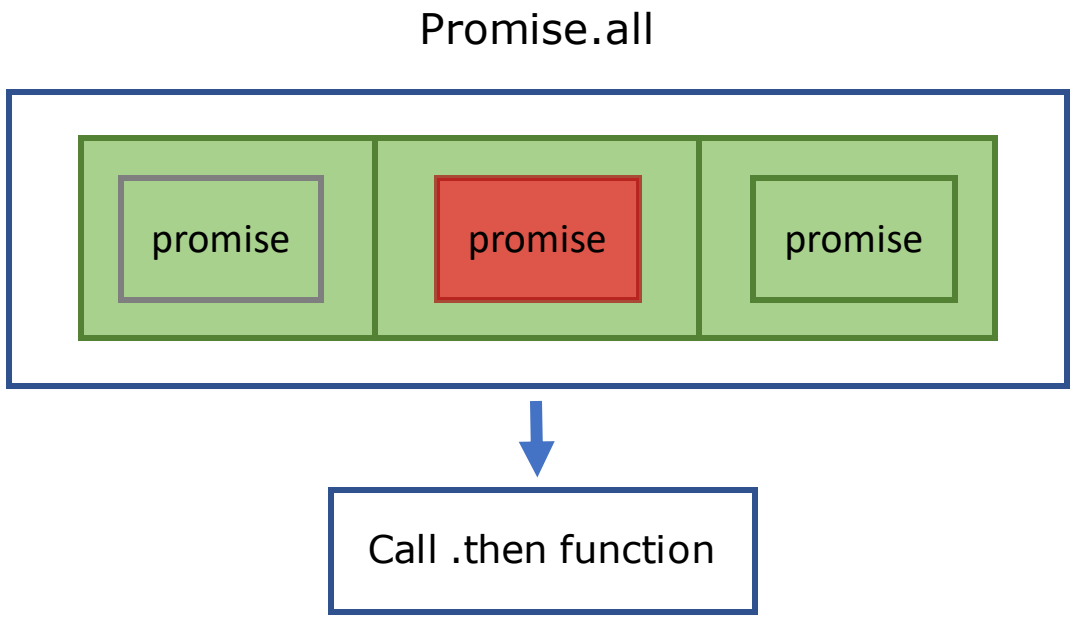
```
/* ... */

const promises = matchRoutes(Routes, request.url).map(({ route }) =>
{
        return route.loadData ? route.loadData(store) : null
}).map(promise => {
        if (promise) {
                return new Promise((resolve, reject) => {
                promise.then(resolve).catch(resolve)
                })
        }
})

Promise.all(promises).then(() => {
        const reduxState = store.getState( )
        const html= htmlTemplate(request, reduxState )

        res.send(html)
})
```



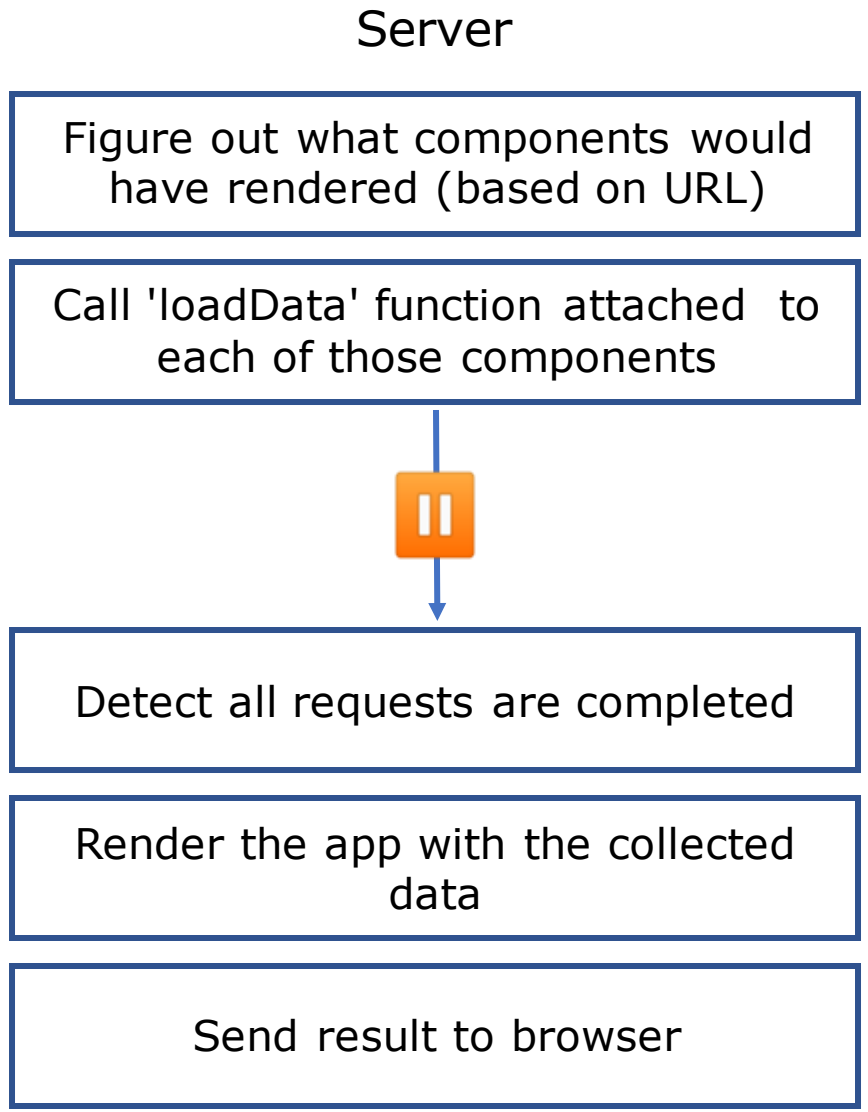
```
/* ... */

const promises = matchRoutes(Routes, request.url).map(({ route }) =>
{
        return route.loadData ? route.loadData(store) : null
}).map(promise => {
        if (promise) {
                return new Promise((resolve, reject) => {
                promise.then(resolve).catch(resolve)
                })
        }
})

Promise.all(promises).then(() => {
        const reduxState = store.getState( )
        const html= htmlTemplate(request, reduxState )

        res.send(html)
})
```



```
/* ... */

const promises = matchRoutes(Routes, request.url).map(({ route }) =>
{
        return route.loadData ? route.loadData(store) : null
}).map(promise => {
        if (promise) {
                return new Promise((resolve, reject) => {
                promise.then(resolve).catch(resolve)
                })
        }
})

Promise.all(promises).then(() => {
        const reduxState = store.getState( )
        const html= htmlTemplate(request, reduxState )

        res.send(html)
})
```



```
/* ... */

const promises = matchRoutes(Routes, request.url).map(({ route }) =>
{
        return route.loadData ? route.loadData(store) : null
}).map(promise => {
        if (promise) {
                return new Promise((resolve, reject) => {
                promise.then(resolve).catch(resolve)
                })
        }
})

Promise.all(promises).then(() => {
        const reduxState = store.getState( )
        const html= htmlTemplate(request, reduxState )

        res.send(html)
})
```



```
/* ... */

const promises = matchRoutes(Routes, request.url).map(({ route }) =>
{
        return route.loadData ? route.loadData(store) : null
}).map(promise => {
        if (promise) {
                return new Promise((resolve, reject) => {
                promise.then(resolve).catch(resolve)
                })
        }
})

Promise.all(promises).then(() => {
        const reduxState = store.getState( )
        const html= htmlTemplate(request, reduxState )

        res.send(html)
})
```

<h3>Redux principles</h3>

  Single source of truth
  State is read-only
  Changes are made with pure functions


```
/* ... */

const content = renderToString(
        <Provider store={store}>
                <StaticRouter location={request.url}>
                        {renderRoutes(Routes)}
                </StaticRouter>
        </Provider>
)

const html = `<html>
                        <head></head>
                        <body>
                                <div id="root">${content}</div>
                                <script>
                                        window.INITIAL_STATE = ${JSON.stringify(store.getState())}
                                </script>
                                <script src="bundle.js"></script>
                        </body>
                        </html>`
```

```
/* ... */

const content = renderToString(
        <Provider store={store}>
                <StaticRouter location={request.url}>
                        {renderRoutes(Routes)}
                </StaticRouter>
        </Provider>
)

const html = `<html>
                        <head></head>
                        <body>
                                <div id="root">${content}</div>
                                <script>
                                        window.INITIAL_STATE = ${JSON.stringify(store.getState())}
                                </script>
                                <script src="bundle.js"></script>
                        </body>
                        </html>`
```

```
/* ... */

const content = renderToString(
      <Provider store={store}>
            <StaticRouter location={request.url}>
                  {renderRoutes(Routes)}
            </StaticRouter>
      </Provider>
)

const html = `<html>
                  <head></head>
                  <body>
                        <div id="root">${content}</div>
                        <script>
                              window.INITIAL_STATE = ${JSON.stringify(store.getState())}
                        </script>
                        <script src="bundle.js"></script>
                  </body>
                  </html>`
```

Client

```
const initialState = window.INITIAL_STATE ? window.INITIAL_STATE : {}
const store = createStore(reducers, initialState)

ReactDOM.hydrate(<Provider store={store}>
                        <BrowserRouter>
                              {renderRoutes(Routes)}
                        </BrowserRouter>
                  </Provider>, document.querySelector('#root'))
```

```
/* ... */

const content = renderToString(
      <Provider store={store}>
            <StaticRouter location={request.url}>
                  {renderRoutes(Routes)}
            </StaticRouter>
      </Provider>
)

const html = `<html>
                  <head></head>
                  <body>
                        <div id="root">${content}</div>
                        <script>
                              window.INITIAL_STATE = ${JSON.stringify(store.getState())}
                        </script>
                        <script src="bundle.js"></script>
                  </body>
                  </html>`
```

Client

```
const initialState = window.INITIAL_STATE ? window.INITIAL_STATE : {}
const store = createStore(reducers, initialState)

ReactDOM.hydrate(<Provider store={store}>
                        <BrowserRouter>
                              {renderRoutes(Routes)}
                        </BrowserRouter>
                  </Provider>, document.querySelector('#root'))
```

Challenge: Need state rehydration on the browser

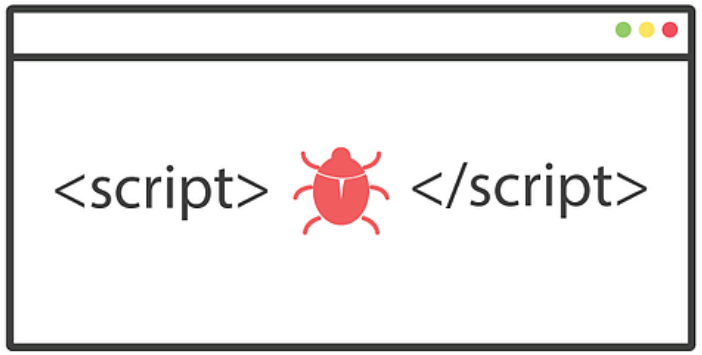
<h3>Search Engine Optimization (SEO)</h3>

```
<ul>
  <li>Helps people to find your website using search engines</li>
  <li>Brings more online business by improving search engine ranking</li>
</ul>
```

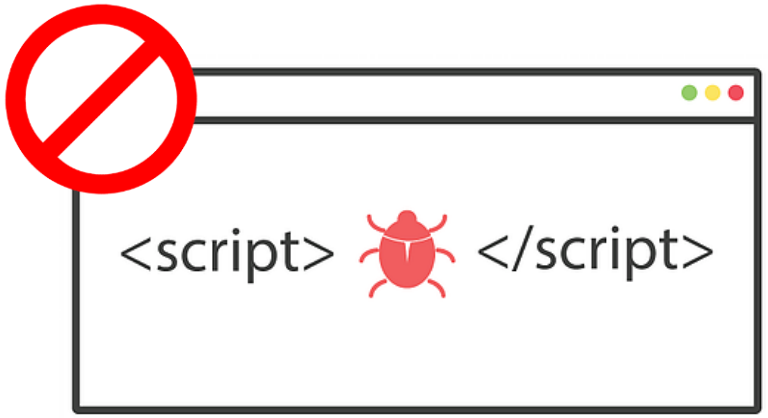
```
/* ... */


const html = `<html>
                <head>
                </head>
                <body>
                    <div id="root">${content}</div>
                    <script>
                        window.INITIAL_STATE = ${JSON.stringify(store.getState())}
                    </script>
                    <script src="bundle.js"></script>
                </body>
                </html>`
```

```
import { Helmet } from 'react-helmet'

const helmet = Helmet.renderStatic()

const html = `<html>
                <head>
                ${helmet.title.toString()}
                ${helmet.meta.toString()}
                </head>
                <body>
                    <div id="root">${content}</div>
                    <script>
                        window.INITIAL_STATE = ${JSON.stringify(store.getState())}
                    </script>
                    <script src="bundle.js"></script>
                </body>
                </html>`
```

```
import { Helmet } from 'react-helmet'

const helmet = Helmet.renderStatic()

const html = `<html>
                <head>
                ${helmet.title.toString()}
                ${helmet.meta.toString()}
                </head>
                <body>
                    <div id="root">${content}</div>
                    <script>
                        window.INITIAL_STATE = ${JSON.stringify(store.getState())}
                    </script>
                    <script src="bundle.js"></script>
                </body>
                </html>`
```



```
import serialize from 'serialize-javascript'

const helmet = Helmet.renderStatic()

const html = `<html>
                <head>
                ${helmet.title.toString()}
                ${helmet.meta.toString()}
                </head>
                <body>
                    <div id="root">${content}</div>
                    <script>
                        window.INITIAL_STATE = ${serialize(store.getState())}
                    </script>
                    <script src="bundle.js"></script>
                </body>
                </html>`
```



<h3>More Challenges</h3>

  Authentication in SSR
  Error Handling and Redirects

QUOIN

<title>SSR with React and Redux</title>

Presented by Fiax

Date: 2020-08-20

QUOIN

<title>SSR with React and Redux</title> 🤞

Presented by Fiax

Date: 2020-08-20