KINGSLAND UNIVERSITY

Routing

Browser Routing

Table of Contents

SPA, Multi Page

Types of Web Applications

Multi Page Applications

- **Reloads** the entire page
- **Displays** the **new page** when a user interacts with the web app
- When a data is exchanged, a **new page** is **requested** from the server to display in the web browser

Multi Page Pros and Cons

Pros

- Performs well on the **search engine**
- Provides a **visual map** of the web app to the user

Cons

- Comparatively **complex development**
- Coupled backend and frontend

Single Page Applications

- A next evolution from multi-page website
- Web apps that load a **single HTML file**
- SPAs use **AJAX** and **HTML5** to create fluid and responsive Web apps
- **No** constant **page reloads**

Single Page Applications

- **Re-renders** its content in response to navigation actions, **without reloading** of the page
- Can use **state** from **external source** or track state internally
  - Internal state SPAs are **limited** - only one "entry"
  - With location-based SPAs, the location is always updating
  - Location-based SPAs need a special object "Router"

SPA Pros and Cons

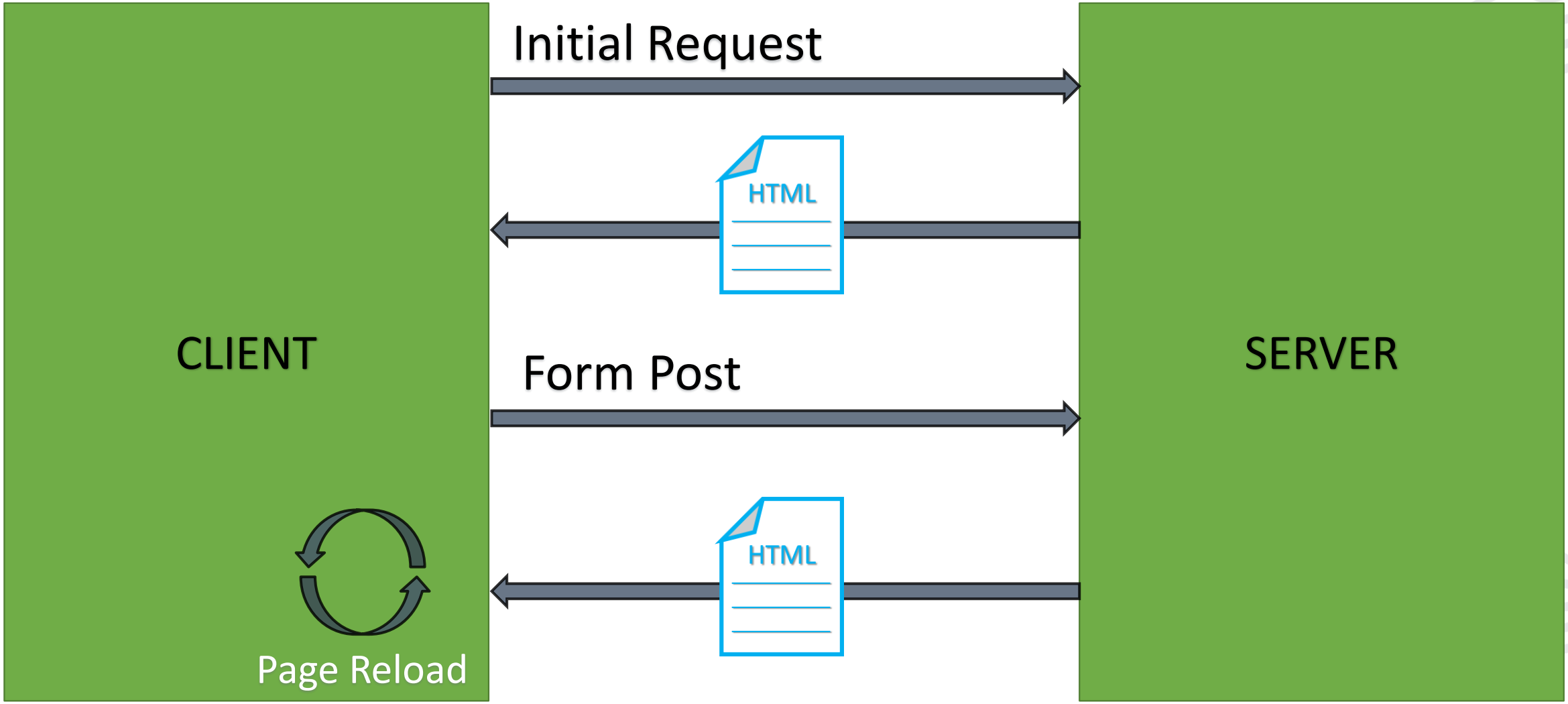
Pros

- Load all scripts **only once**
- **Maintain state** across multiple pages
- Browser **history** can be used
- Better **UX**

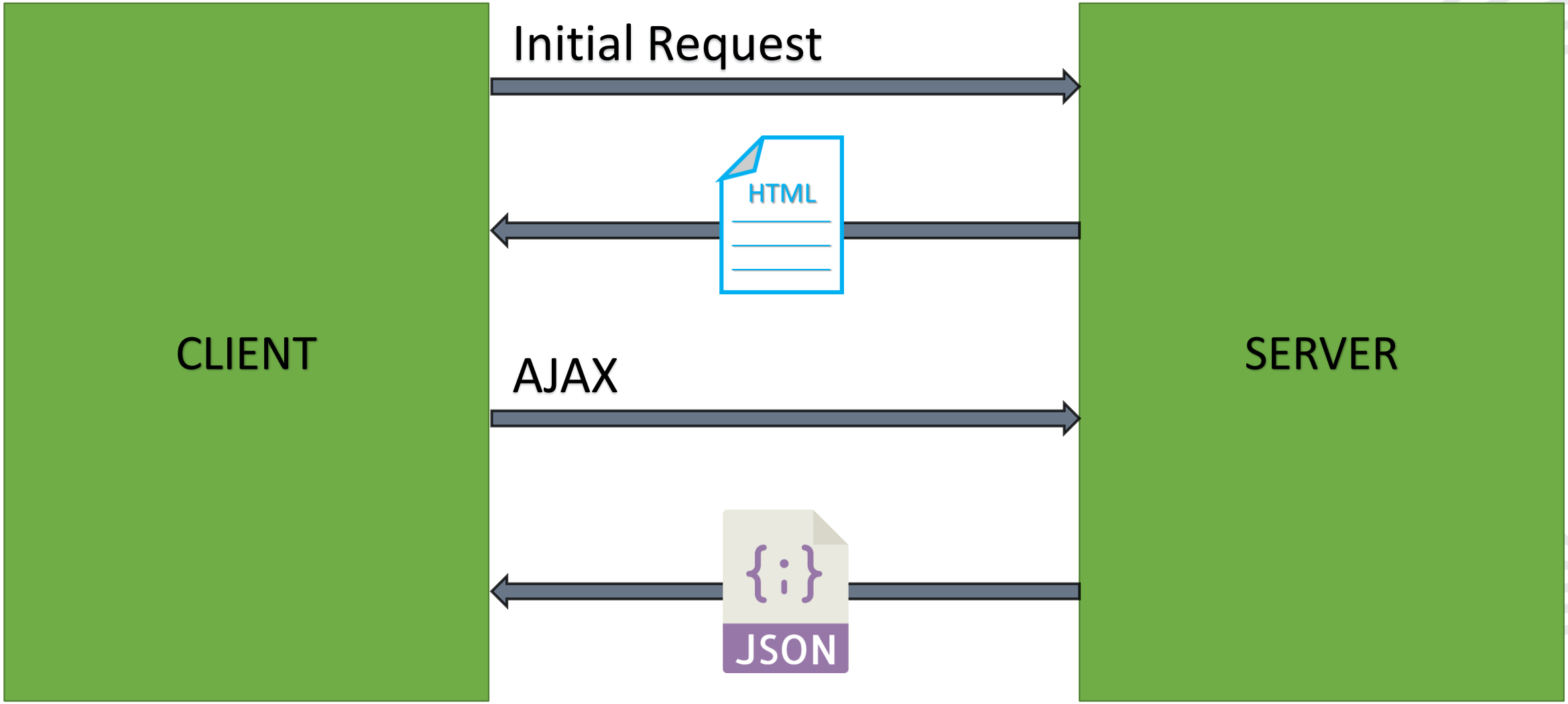
Cons

- Perform poor on the search engine
  - Server-side rendering helps
- Provide **single sharing link**
- Less secure

Multi Page Application Lifecycle

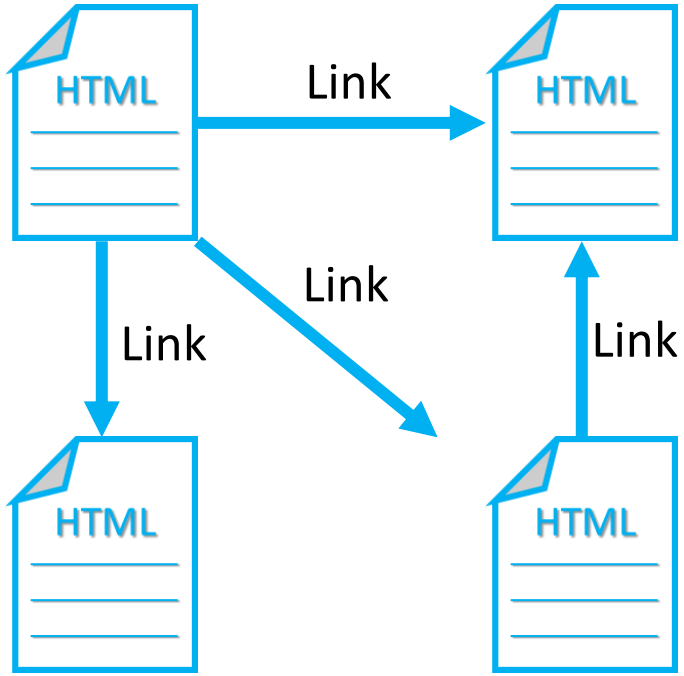


SPA Lifecycle

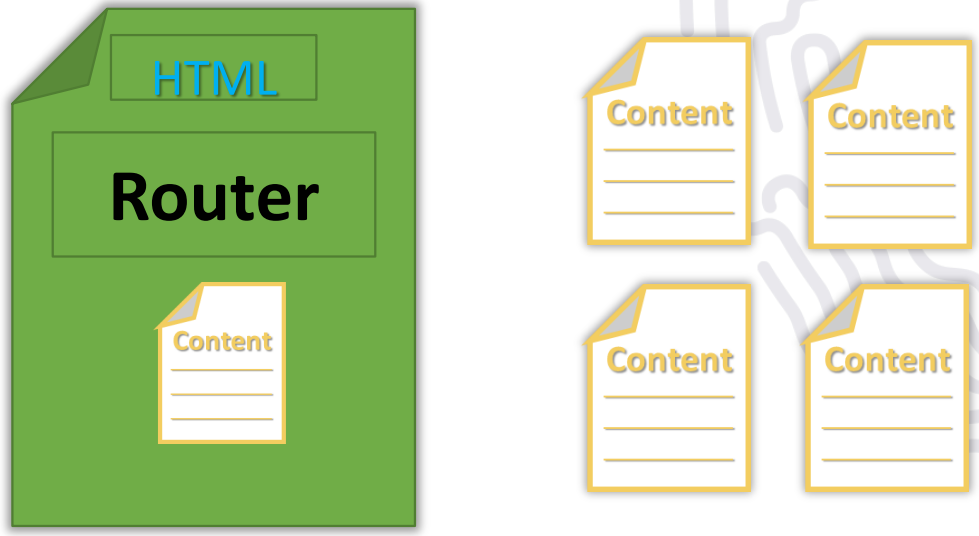


Navigation Types

- Standard Navigation



- Navigation using **Routing** - allows navigation, **without reloading** the page



Query Parameters

- Allow for additional application **state** to be **serialized** into the URL
- Common **use cases**
  - Representing the current page number in a paginated collection
  - Filter criteria
  - Sorting criteria

Location

https://www.example.com/one?key=value#trending

```
location = {
  protocol: "https:",
  hostname: "www.example.com",
  pathname: "/one",
  search: "?key=value",
  hash: "#trending"
}
```

Navigation for Single Page Apps

Routing Concepts

How Routers Work

- A **Router** loads the appropriate content when the **location changes**
  - E.g. when the user manually **enters an address**
- Conversely, a change in content is reflected in the address bar
  - E.g. when the user **clicks on a link**

Hash-based Routing

- Using the **#hash** part of the URL to simulate different content
- The routing is possible because changes in the hash **don't trigger page reload**

Example

- Extracting the hash from the entire URL

```
let hash = window.location.href.split('#')[1] || '';
```

- Changing the path

```
let changePath = function (path) {
  let currentPath = window.location.href;
  window.location.href =
    currentPath.replace(/#(.*)$/, '') + '#'+ path;
}
```

Example

- Subscribe for changes

```
let url = undefined;
let getCurrent = function () {
  return window.location.hash;
};
let listen = function () {
  let current = getCurrent();
  if (current !== url) {
    url = current;
  }
  setTimeout(listen, 200);
};
listen();
```

Push-Based Routing

- You can actually surface real **server-side data** to support things like SEO and Facebook Open Graph
- It helps with **analytics**
- It helps fix **hash tag issues**
- You can actually use hash tag for what is was meant for, **deep linking** to sections of long pages

History API

- Provides access to the browser's history through the **`history`** object
- **HTML5** introduced the **`history.pushState()`** and **`history.replaceState()`**
  - They allow you to add and modify **history entries**
  - These methods work in conjunction with the **`popstate`** event

The PushState() Method

- Adds new object to the history of the browser
- Takes three parameters:
  - **State**
    - Object which is associated with the new history entry
  - **Title**
    - Browsers currently ignore this parameter
  - **URL**
    - The new history entry's URL is given by this parameter
    - It must be of the **same origin** as the current URL

The ReplaceState() Method

- **Modifies the current history entry** instead of creating a new one
- It is particularly useful when you want to update the **state object** or **URL** of the current history entry

```
let stateObj = { facNum: "56789123" };
history.pushState(stateObj, "", "student.html");

history.replaceState(stateObj, "", "newStudent.html");
```

The Popstate Event

- Dispatched to the window every time the active history entry changes
- If the history entry being activated was created by a call to **pushState** or affected by a call to **`replaceState`**,
- The **`popstate`** event's **`state` property** contains a copy of the history entry’s state object
- You can read the state of the current history entry without waiting for a **`popstate`** event using the **`history.state` property**

Live Demo

Summary

- Multi Page Application
  - **Reloads** the entire page
- Single Page Application
  - **Re-renders** its content
- Routing
  - **Hash-based**
  - **Push-based**
    - History API-provides access to the **browser's history**

Questions?

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © Kingsland University – https://kingslanduniversity.com

KINGSLAND UNIVERSITY

THANK YOU