Svelte

Outline

- What is Svelte
- Svelte Components
  - Structure
  - Lifecycle
- Core Features
- Svelte Compiler
- Svelte Kit
- Who benefits from Svelte?

What is Svelte?

- A UI Framework used for building reusable components

- A Compiler

  Generates a minimal and **highly optimized JavaScript** code from our declarative component code at **build time**.

  Frameworkless logic sent to the browser = highly performant

  Zero runtime dependencies, only devDependencies!

Svelte Components

- Classic JS in HTML syntax with a **.svelte** extension
- Superset of HTML, allowing JS expressions such as conditions, loops
- Components have a scoped and isolated CSS block
- JavaScript block that runs when a component instance is created.

App.svelte

```svelte
<script>
  let count = 0;

  function handleClick() {
    count += 1;
  }
</script>

<h1>
  Counter example
</h1>
<button on:click={handleClick}>
  Clicked {count} {count === 1 ? 'time' : 'times'}
</button>

<style>
  h1 {
    color: red;
  }
</style>
```

Svelte Component lifecycle

- **onMount:** Runs after the component is first rendered on the DOM
  - **return** functions will be called when the component is destroyed
- **onDestroy:** Contains logic to be run when the component is destroyed
- **beforeUpdate:** Runs immediately before the DOM is updated
- **afterUpdate:** Runs immediately after the DOM is updated and in sync with our data

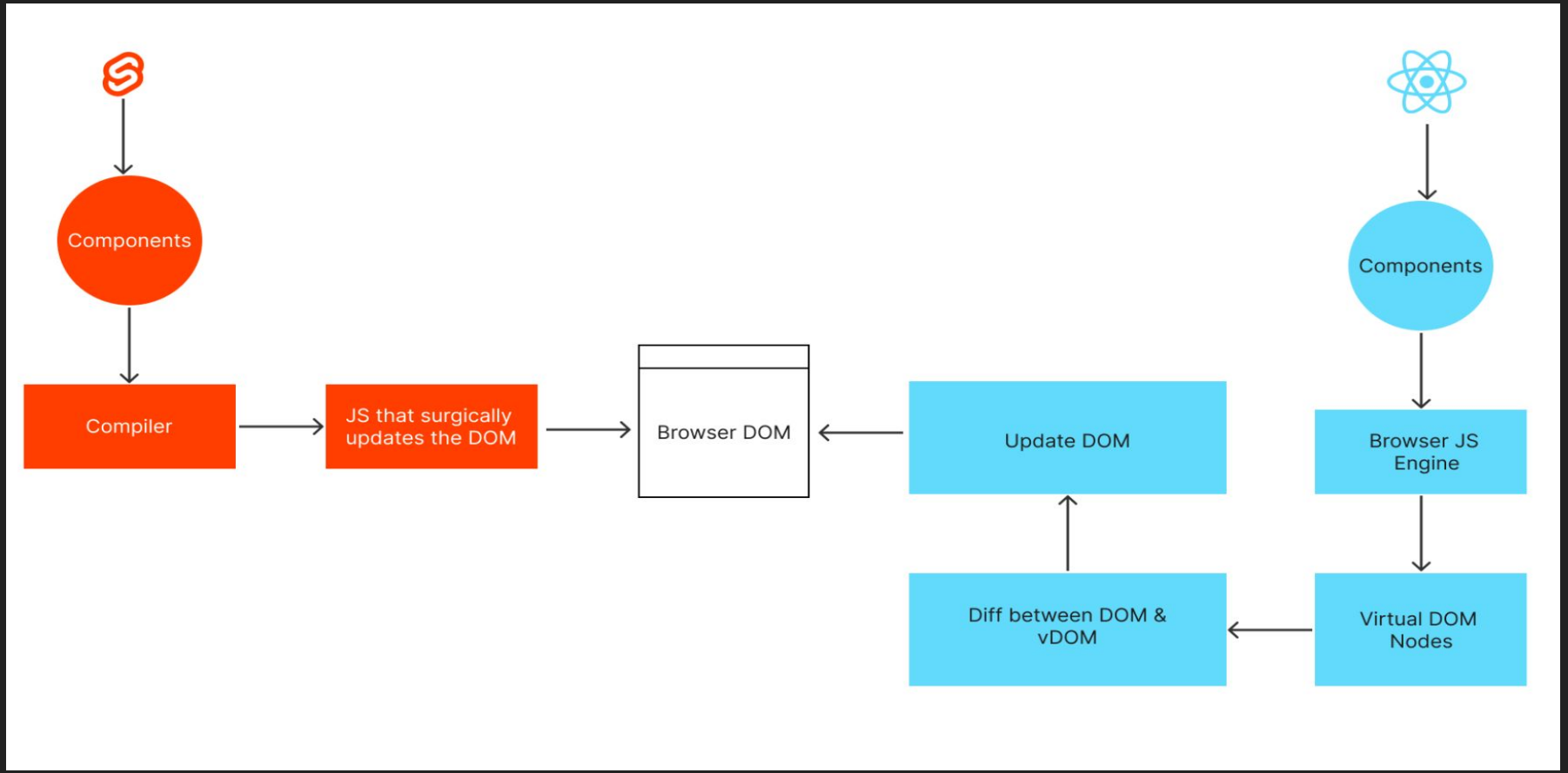
Life cycle functions do not run in SSR except onDestroy: Avoid fetching data that can be loaded lazily after the component has been mounted

Svelte Core Features

- No Virtual DOM
- Truly Reactive
- Write Less Code

No Virtual DOM

- vDOM’s benefit: It is easier to manipulate than the real DOM
- Virtual DOM diffing brings overhead to browser
- Svelte runs at build time converting declarative component code into highly efficient imperative code that surgically updates the DOM


Components
Compiler
JS that surgically updates the DOM
Browser DOM
Update DOM
Diff between DOM & vDOM
Components
Browser JS Engine
Virtual DOM Nodes

Truly Reactive

1. Variable Assignments
2. Derived State

Write less code

- More code = more bugs
- Multiple top level elements
- Bindings
- State updates are as simple as assignment statements
- Reactive declarations (as opposed to using useMemo, etc)

Svelte Compiler

- Declaratively described code to highly optimized vanilla JS
- Traditionally done by web frameworks like Vue, React using vDOM
- What if we shift the framework magic to the build time?

Svelte Compiler Steps

- Parse the Svelte Code
- Static Analysis
- Rendering
  - Generate for DOM
  - Generate for SSR
- Output as js + css

1. Parsing the code

- Svelte has its own parser for
  - HTML templates
  - Logic Blocks
- For parsing JS content - it uses **Acron**
- For parsing CSS - it uses **CSSTree**

Svelte code is converted into tokens and arranged into Svelte AST

- Output: AST representation of svelte code

App.svelte*

```
<script>
  let count = 0;

  $: double = count * 2;

  function handleCount() {
    count+=1;
  }
</script>

<button>
  Click Me
</button>
<h1>Count {count}</h1>
<h2>
  Double {double}
</h2>

<style>
  h1{
    color: red;
  }
</style>
```

2. Static Analysis

1. Traverse the Script tag
2. Traverse the Template
3. Traverse the Script again (optimization)
4. Traverse the Style tag (Add unique prefixes)

Variables

- count referenced(script) mutated referenced(template)
- double injected referenced(template)

- Output: Analyzes the code without executing it to figure out the variables and their behaviors

App.svelte*

```
<script>
  let count = 0;

  $: double = count * 2;

  function handleCount() {
    count+=1;
  }
</script>

<button>
  Click Me
</button>
<h1>Count {count}</h1>
<h2>
  Double {double}
</h2>

<style>
  h1{
    color: red;
  }
</style>
```

3. Render Phase

- Depending on our compiler options, it either
  - Generates JS for DOM
  - Generates JS for SSR

Output: JavaScript Code

Svelte Kit

- A web app framework built on top of Svelte
- Utilizes Vite to build code
- Functionalities include
    - File Based Routing
    - Built in **load** functions for fetching data before the component runs
    - Built in **layout** component for handling Default Application Layouts
    - SSR
    - SEO enhancement
    - State Management
    - Asset Handling
    - Form Handling
    - Eslint, Prettier, TypeScript, configured out of the box (Zero Config)

Use cases

- Intended for low-end devices with limited processing power: Due to its small bundle sizes
- Highly interactive pages with large number of DOM elements: No runtime overhead
- Small learning curve and is perfect for onboarding developers with basic HTML, CSS and JS knowledge

Conclusion

- Svelte is UI framework & a compiler
- It is fast, because of its no vDOM implementation
- Used to build interactive websites
- Svelte Kit for structuring our code