Enhancing SSR in Low-Thread Web Servers

A Comprehensive Approach for Progressive Server-Side Rendering with Any Asynchronous API and Multiple Data Models

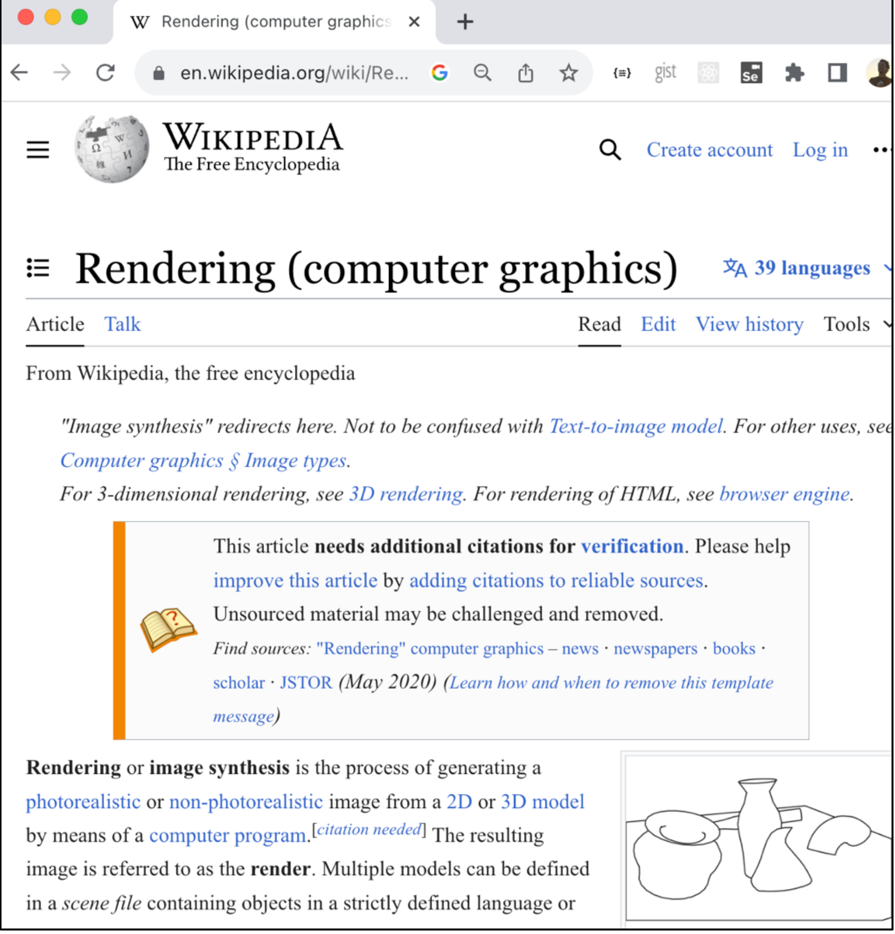
Fernando Miguel Carvalho (ISEL)

Novembro 2023

Enhancing SSR in Low-Thread Web Servers

A Comprehensive Approach for Progressive Server-Side Rendering with Any Asynchronous API and Multiple Data Models

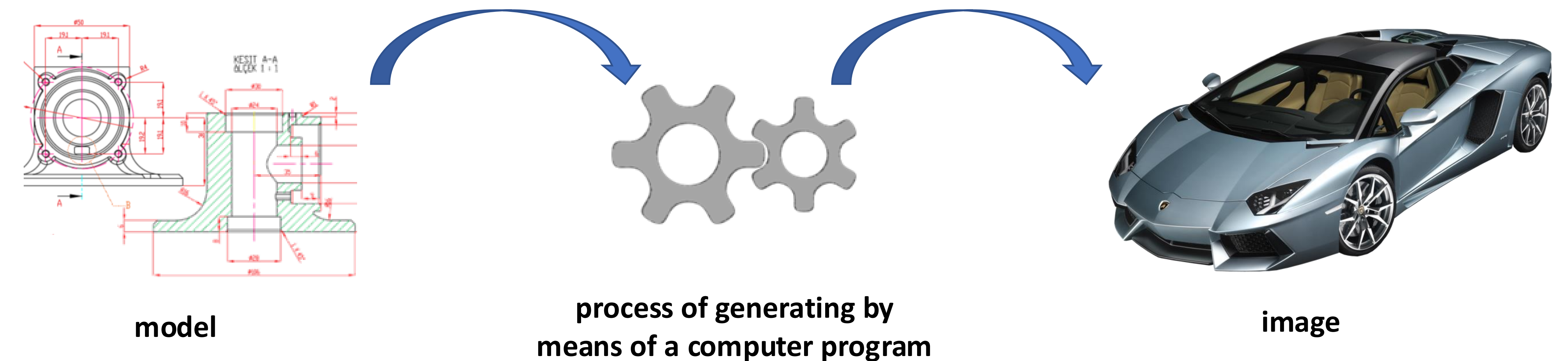
Progressive Server-Side **Rendering**



*"Rendering is the **process of generating** a photorealistic or non-photorealistic **image** from a 2D or 3D **model** by means of a **computer program**"*

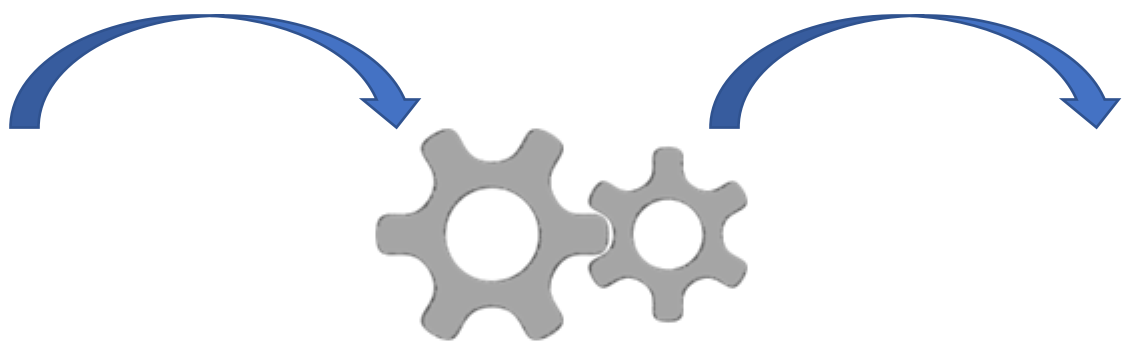
Progressive Server-Side Rendering

"Rendering is the ***process of generating*** *a photorealistic or non-photorealistic* ***image*** *from a 2D or 3D* ***model*** *by means of a* ***computer program****"*



Progressive Server-Side **Rendering**

Web Frontend



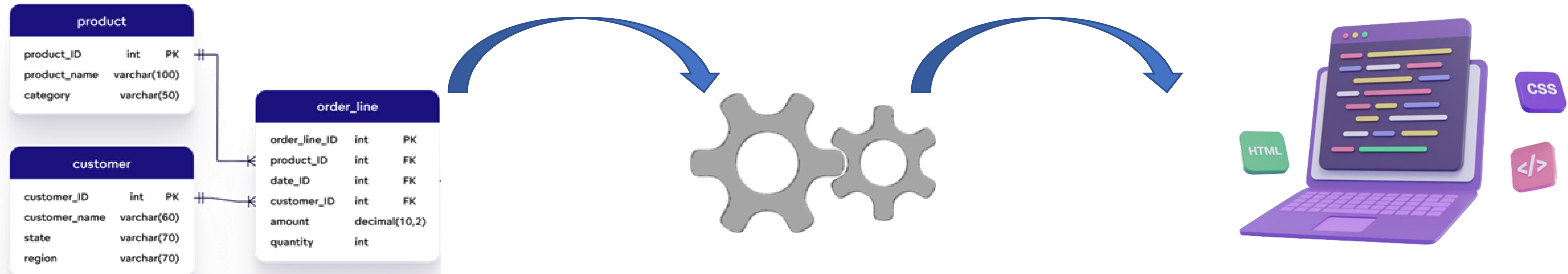
model

process of generating by means of a computer program

image

Progressive Server-Side Rendering

Web Frontend

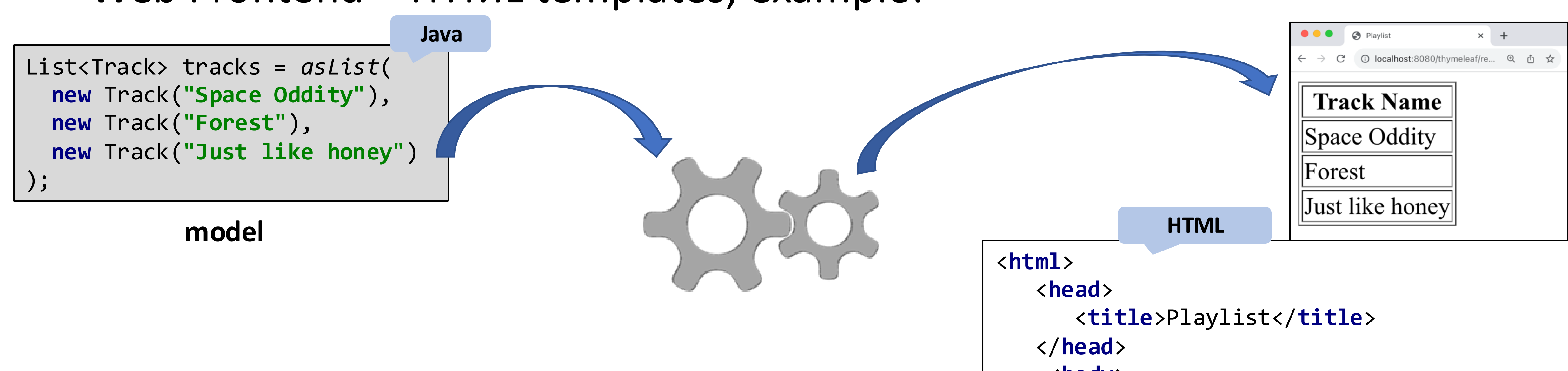


model

Frontend

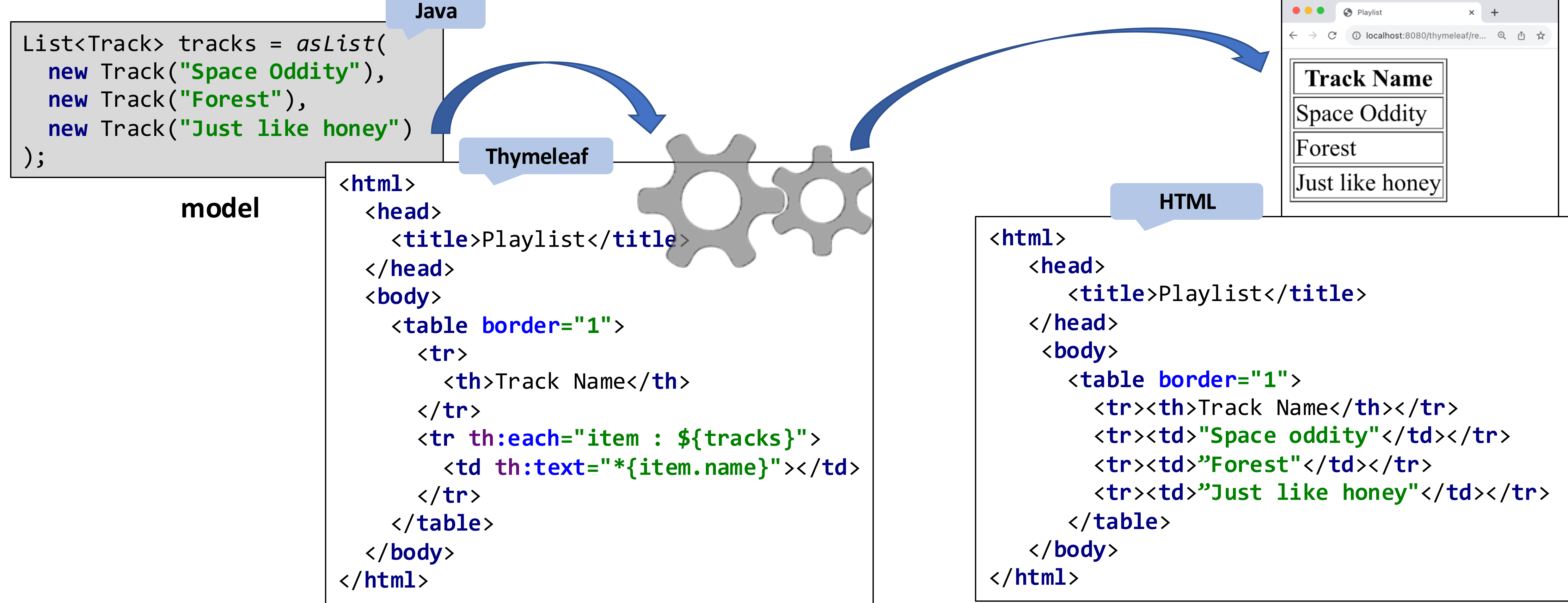
Progressive Server-Side Rendering

Web Frontend – HTML templates, example:



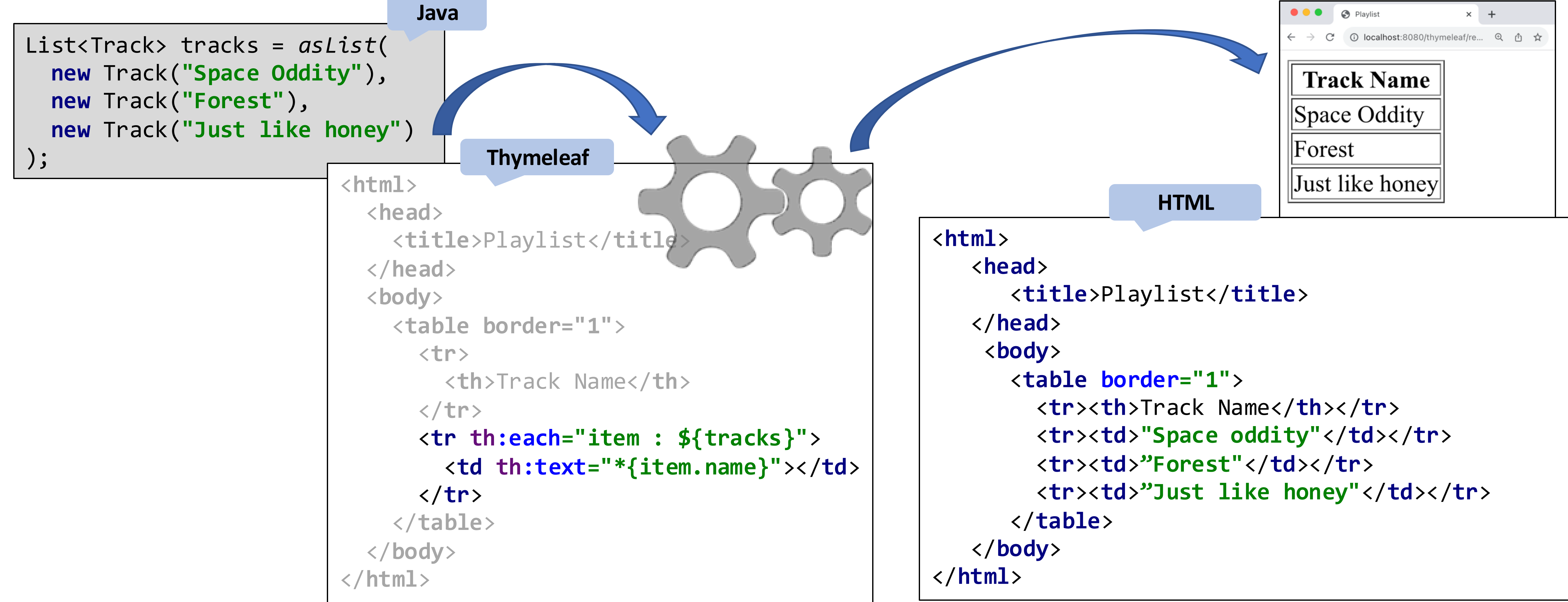
Progressive Server-Side Rendering

Web Frontend – HTML templates, example:



Progressive Server-Side Rendering

Web Frontend – HTML templates, example:



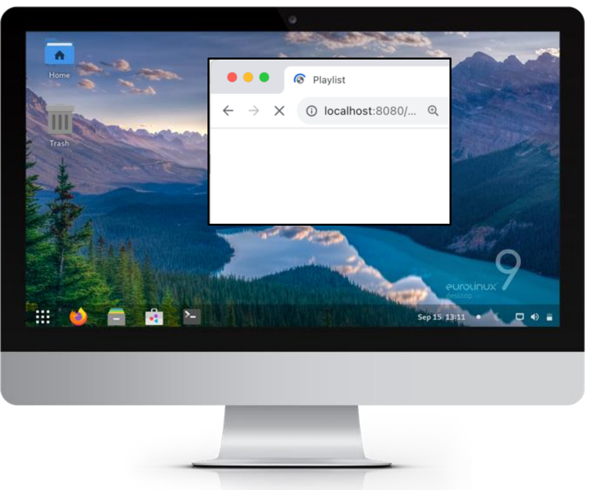
Progressive Server-Side Rendering

```java
List<Track> tracks = asList(
  new Track("Space Oddity"),
  new Track("Forest"),
  new Track("Just like honey")
);
```

```html
<html>
  <head>
    <title>Playlist</title>
  </head>
  <body>
    <table border="1">
      <tr>
        <th>Track Name</th>
      </tr>
      <tr th:each="item : ${tracks}">
        <td th:text="*{item.name}"></td>
      </tr>
    </table>
  </body>
</html>
```

```html
<html>
   <head>
      <title>Playlist</title>
   </head>
    <body>
      <table border="1">
        <tr><th>Track Name</th></tr>
        <tr><td>"Space oddity"</td></tr>
        <tr><td>”Forest"</td></tr>
        <tr><td>”Just like honey"</td></tr>
      </table>
   </body>
</html>
```

Progressive Server-Side Rendering



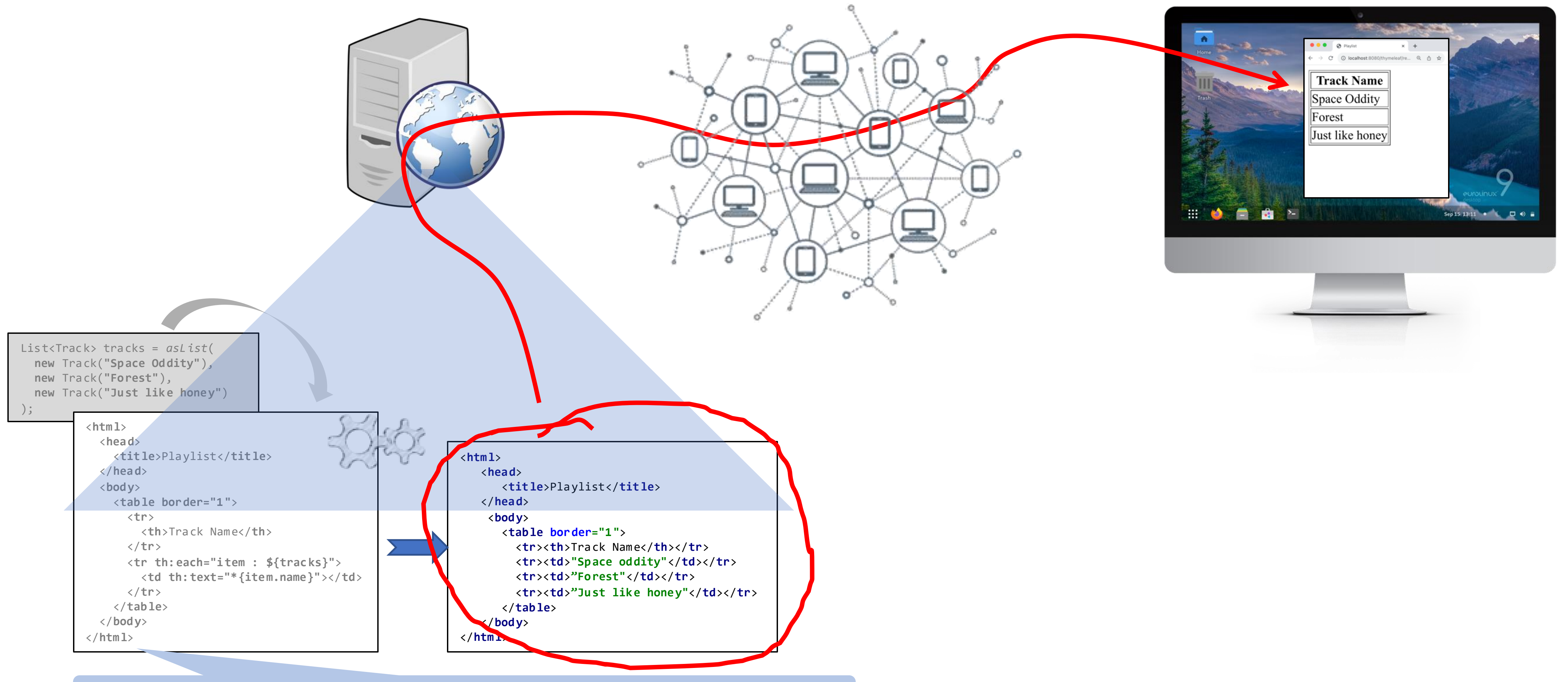

```
List<Track> tracks = asList(
  new Track("Space Oddity"),
  new Track("Forest"),
  new Track("Just like honey")
);
```

```
<html>
  <head>
    <title>Playlist</title>
  </head>
  <body>
    <table border="1">
      <tr>
        <th>Track Name</th>
      </tr>
      <tr th:each="item : ${tracks}">
        <td th:text="*{item.name}"></td>
      </tr>
    </table>
  </body>
</html>
```

```
<html>
   <head>
      <title>Playlist</title>
   </head>
    <body>
      <table border="1">
        <tr><th>Track Name</th></tr>
        <tr><td>"Space oddity"</td></tr>
        <tr><td>”Forest"</td></tr>
        <tr><td>”Just like honey"</td></tr>
      </table>
   </body>
</html>
```

E.g. Thymeleaf, JSP, PHP, Jade, Rythm, Pebble, Rocker, Handlebars, etc

Progressive Server-Side Rendering



E.g. Thymeleaf, JSP, PHP, Jade, Rythm, Pebble, Rocker, Handlebars, etc

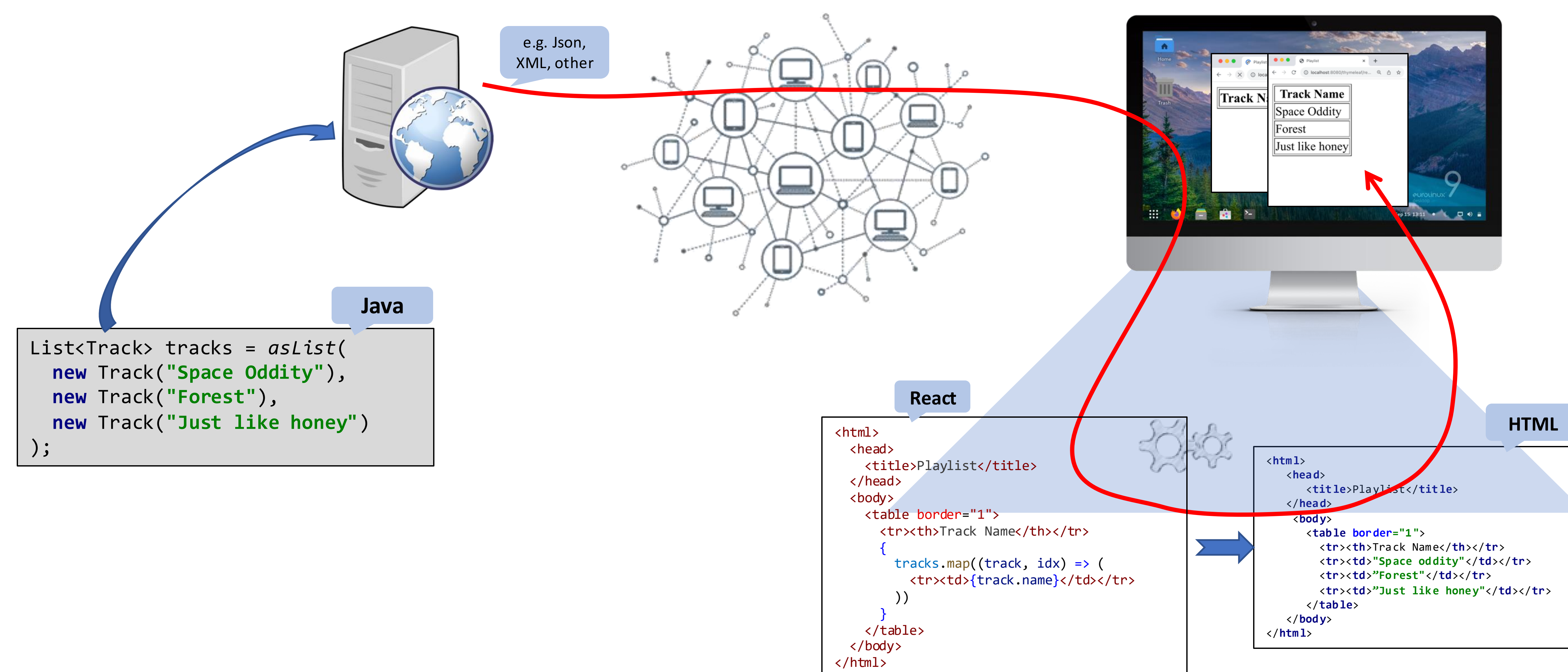
Progressive Client-Side Rendering

```
List<Track> tracks = asList(
  new Track("Space Oddity"),
  new Track("Forest"),
  new Track("Just like honey")
);
```

```
<html>
  <head>
    <title>Playlist</title>
  </head>
  <body>
    <table border="1">
      <tr>
        <th>Track Name</th>
      </tr>
      <tr th:each="item : ${tracks}">
        <td th:text="*{item.name}"></td>
      </tr>
    </table>
  </body>
</html>
```

```
<html>
   <head>
      <title>Playlist</title>
   </head>
    <body>
      <table border="1">
        <tr><th>Track Name</th></tr>
        <tr><td>"Space oddity"</td></tr>
        <tr><td>”Forest"</td></tr>
        <tr><td>”Just like honey"</td></tr>
      </table>
   </body>
</html>
```

Progressive Client-Side Rendering



Server-Side <versus> Client-Side Rendering

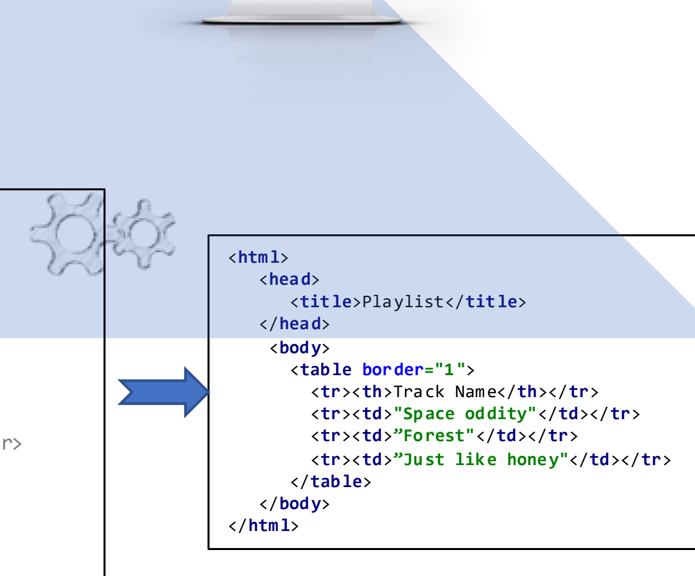
```java
List<Track> tracks = asList(
  new Track("Space Oddity"),
  new Track("Forest"),
  new Track("Just like honey")
);
```

```html
<html>
  <head>
    <title>Playlist</title>
  </head>
  <body>
    <table border="1">
      <tr>
        <th>Track Name</th>
      </tr>
      <tr th:each="item : ${tracks}">
        <td th:text="*{item.name}"></td>
      </tr>
    </table>
  </body>
</html>
```

```html
<html>
   <head>
      <title>Playlist</title>
   </head>
    <body>
      <table border="1">
        <tr><th>Track Name</th></tr>
        <tr><td>"Space oddity"</td></tr>
        <tr><td>”Forest"</td></tr>
        <tr><td>"Just like honey"</td></tr>
      </table>
   </body>
</html>
```

E.g. Thymeleaf, JSP, PHP, Jade, Rythm, Pebble, Rocker, Handlebars, etc

```html
<html>
  <head>
    <title>Playlist</title>
  </head>
  <body>
    <table border="1">
      <tr><th>Track Name</th></tr>
      {
        tracks.map((track, idx) => (
          <tr><td>{track.name}</td></tr>
        ))
      }
    </table>
  </body>
</html>
```



```html
<html>
   <head>
      <title>Playlist</title>
   </head>
    <body>
      <table border="1">
        <tr><th>Track Name</th></tr>
        <tr><td>"Space oddity"</td></tr>
        <tr><td>”Forest"</td></tr>
        <tr><td>"Just like honey"</td></tr>
      </table>
   </body>
</html>
```

E.g. React, Vue, Angular

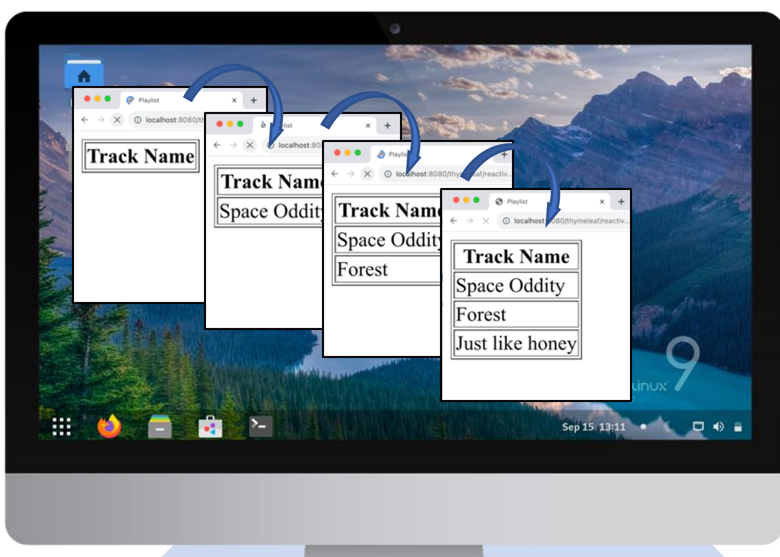
Progressive Rendering

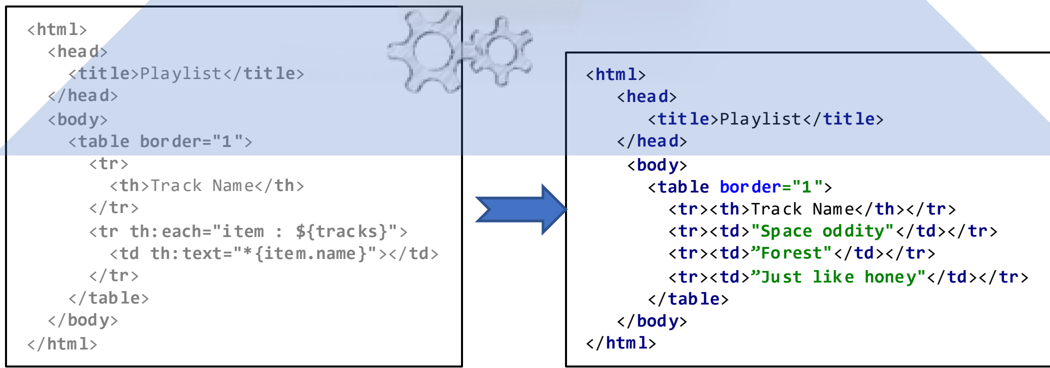
```
List<Track> tracks = asList(
  new Track("Space Oddity"),
  new Track("Forest"),
  new Track("Just like honey")
);
```

```
<html>
  <head>
    <title>Playlist</title>
  </head>
  <body>
    <table border="1">
      <tr>
        <th>Track Name</th>
      </tr>
      <tr th:each="item : ${tracks}">
        <td th:text="*{item.name}"></td>
      </tr>
    </table>
  </body>
</html>
```

```
<html>
  <head>
    <title>Playlist</title>
  </head>
  <body>
    <table border="1">
      <tr><th>Track Name</th></tr>
      <tr><td>"Space oddity"</td></tr>
      <tr><td>"Forest"</td></tr>
      <tr><td>"Just like honey"</td></tr>
    </table>
  </body>
</html>
```

E.g. Thymeleaf, JSP, PHP, Jade, Rythm, Pebble, Rocker, Handlebars, etc





E.g. React, Vue, Angular

SSR dominance

- 2022, registered over 200 million actively websites (W3Techs, 2022)
- 95% of them use SSR (e.g. PHP)
- CSR and SPA (single-page application):
  - incurs **higher complexity**
  - NOT useful for most web applications.

SSR web templates – Issues

- Proliferation of templating dialects

- Unresponsive behaviour

- Inability to handle asynchronous data models

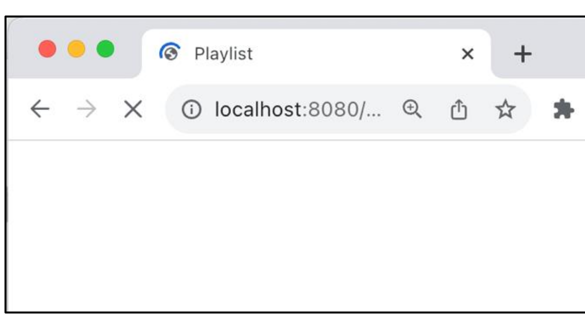
SSR web templates – Issues

- Proliferation of templating dialects

- Unresponsive behaviour

  e.g. dealing with large data sets

  or data source latency



- Inability to handle asynchronous data models

SSR web templates – Issues

- Proliferation of templating dialects
  - Different between template technologies, such as, Thymeleaf, Jade, Rocker, Handlebars, JSP, Pebble, Rythm, Chunk, etc.

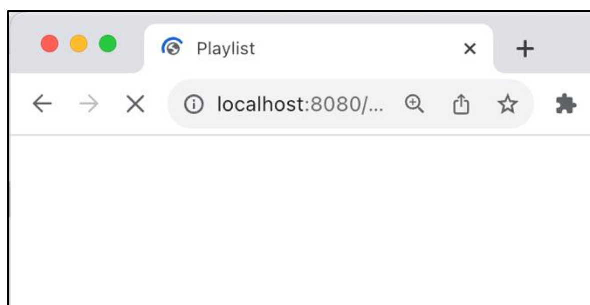
Thymeleaf

```html
<body>
  <table border="1">
    <tr>
      <th>Track Name</th>
    </tr>
    <tr th:each="item : ${tracks}">
      <td th:text="*{item.name}"></td>
    </tr>
  </table>
</body>
```

- Unresponsive behaviour

  e.g. dealing with large data sets

  or data source latency



- Inability to handle asynchronous data models

SSR web templates – Issues

- Proliferation of templating dialects
  - Different between template technologies, such as, Thymeleaf, Jade, Rocker, Handlebars, JSP, Pebble, Rythm, Chunk, etc.

Thymeleaf

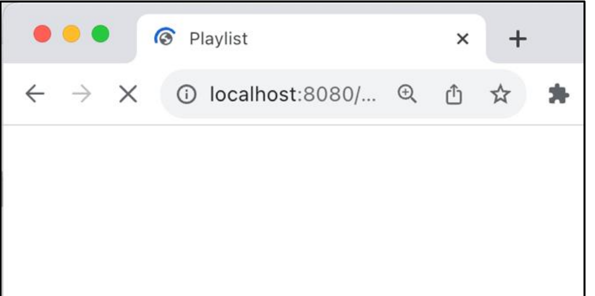
```
<body>
  <table border="1">
    <tr>
      <th>Track Name</t
    </tr>
    <tr th:each="item :
      <td th:text="*{it
    </tr>
  </table>
</body>
```

```
<tr th:each="prod : ${prods}">
<div th:switch="${user.role}">
<p th:case="#{roles.manager}">
<p th:text="*{home.welcome}”
...
```

- Unresponsive behaviour

  e.g. dealing with large data sets

  or data source latency



- Inability to handle asynchronous data models

SSR web templates – Issues - Solutions

- Proliferation of templating dialects

  => Use Host language (e.g. Java, Kotlin, **JavaScript**)

  => e.g. **React**, j2html, HtmlFlow, KotlinX.html

  => HTML Domain Specific Languages

JSX ≃ JavaScript + HTML

```
<div>
  <table border="1">
    <tr><th>Track Name</th></tr>
    {
      tracks.map((track, idx) => (
        <tr key={idx}><td>{track.name}</td></tr>
      ))
    }
  </table>
</div>
```

- Unresponsive behaviour

  e.g. dealing with large data sets

- Inability to handle asynchronous data models

SSR web templates – Issues - Solutions

- Proliferation of templating dialects
  - => Use Host language (e.g. Java, **Kotlin**, JavaScript)
  - => e.g. React, j2html, HtmlFlow, **KotlinX.html**
  - => HTML Domain Specific Languages

KotlinX.html

```
html {
  head { title("Playlist") }
  body {
    table {
      attributes["border"] = "1"
      tr { th { text("Track Name") } }
      tracks
        .forEach { track ->
          tr { td { text(track.name) } }
        }
    } // table
  } // body
} // html
```

- Unresponsive behaviour

  e.g. dealing with large data sets

- Inability to handle asynchronous data models

SSR web templates – Issues - Solutions

- Proliferation of templating dialec

  => Use Host language (e.g. **Java**, JavaScript)
  => e.g. React, j2html, **HtmlFlow**, KotlinX.html
  => HTML Domain Specific Languages

HtmlFlow

```
html()
  .head().title().text("Playlist").__().__()
  .body()
    .table().attrBorder(EnumBorderType._1)
      .tr().th().text("Track name").__().__()
      .dynamic<Iterable<Track>>{ table, tracks -> tracks
        .forEach{ track -> table
          .tr().td().text(track.name).__().__()
        }
    .__() // table
  .__() // body
.__() // html
```

- Unresponsive behaviour

  e.g. dealing with large data sets

- Inability to handle asynchronous data models

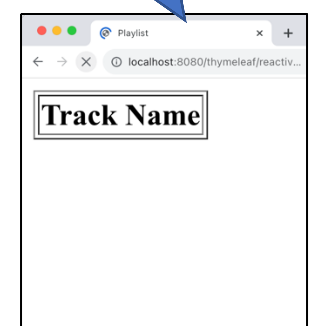
SSR web templates – Issues - Solutions

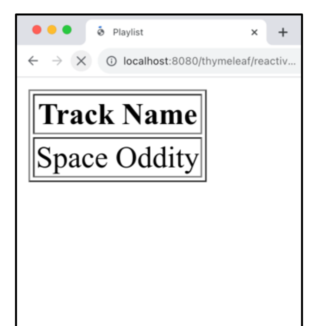
- Proliferation of templating dialec
  - => Use Host language (e.g. Java, JavaScript)
  - => e.g. React, j2html, HtmlFlow, KotlinX.html
  - => HTML Domain Specific Languages
- Unresponsive behaviour

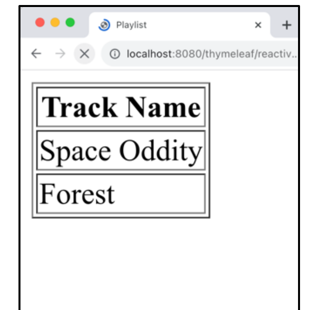
  e.g. dealing with large data sets
  - => Avoid template text files
  - => Higher-order functions
  - => **Progressive** rendering
- Inability to handle asynchronous data models

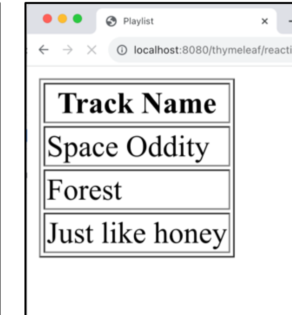
HtmlFlow

```
html()
  .head().title().text("Playlist").__().__()
  .body()
    .table().attrBorder(EnumBorderType._1)
      .tr().th().text("Track name").__().__()
      .dynamic<Iterable<Track>>{ table, tracks -> tracks
        .forEach{ track -> table
          .tr().td().text(track.name).__().__()
        }
    .__() // table
  .__() // body
.__() // html
```









SSR web templates – Issues - Solutions

- Proliferation of templating dialec

  => Use Host language (e.g. Java, JavaScript)

  => e.g. React, j2html, HtmlFlow, KotlinX.html

  => HTML Domain Specific Languages

HtmlFlow

```
html()
  .head().title().text("Playlist").__().__()
  .body()
    .table().attrBorder(EnumBorderType._1)
      .tr().th().text("Track name").__().__()
      .dynamic<Iterable<Track>>{ table, tracks -> tracks
        .forEach{ track -> table
          .tr().td().text(track.name).__().__()
        }
    .__() // table
  .__() // body
.__() // html
```

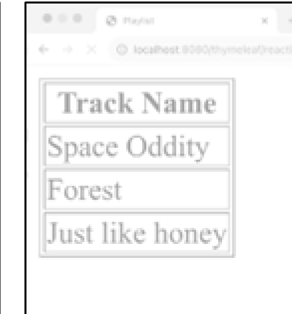
- Unresponsive behaviour

  e.g. dealing with large data sets

  => Avoid template text files

  => Higher-order functions

  => **Progressive** rendering

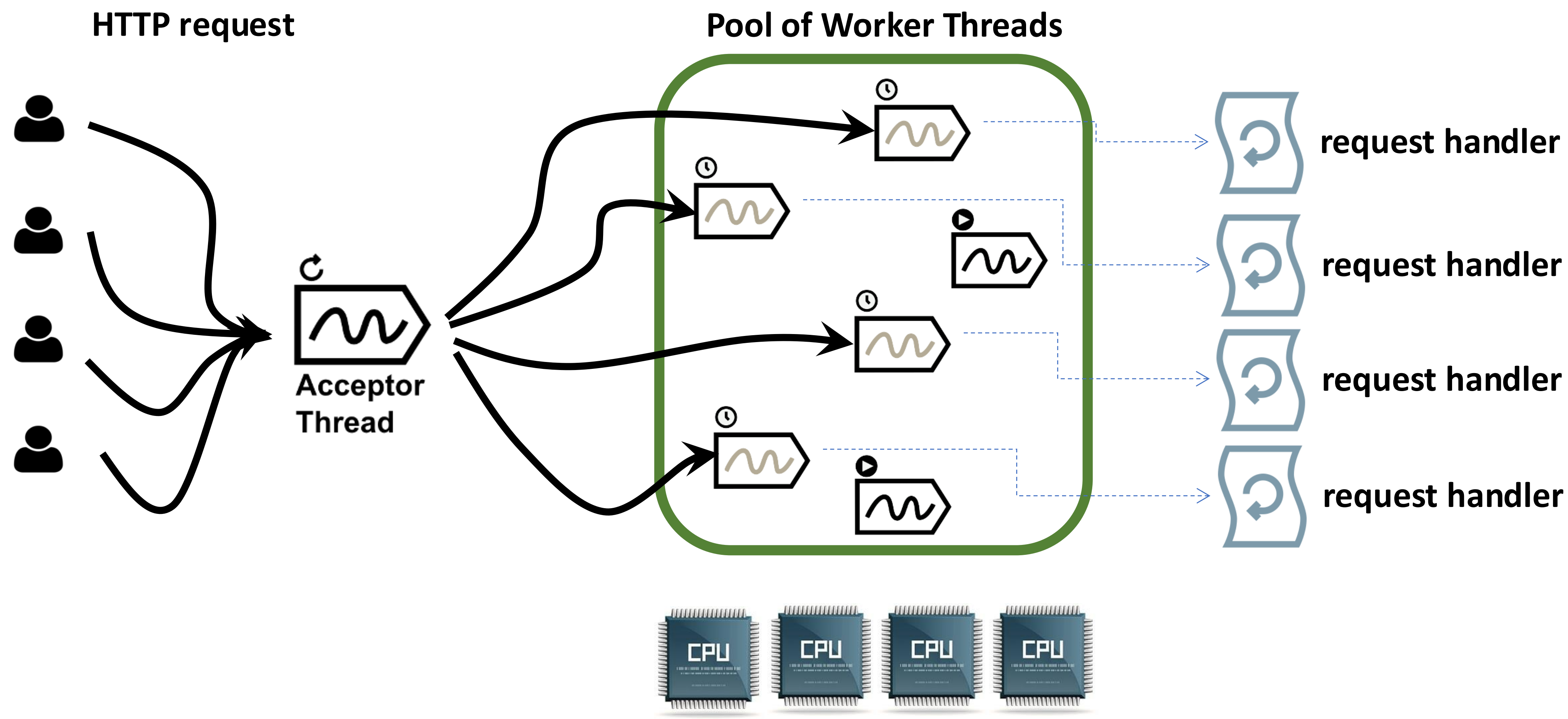


- Inability to handle asynchronous data models

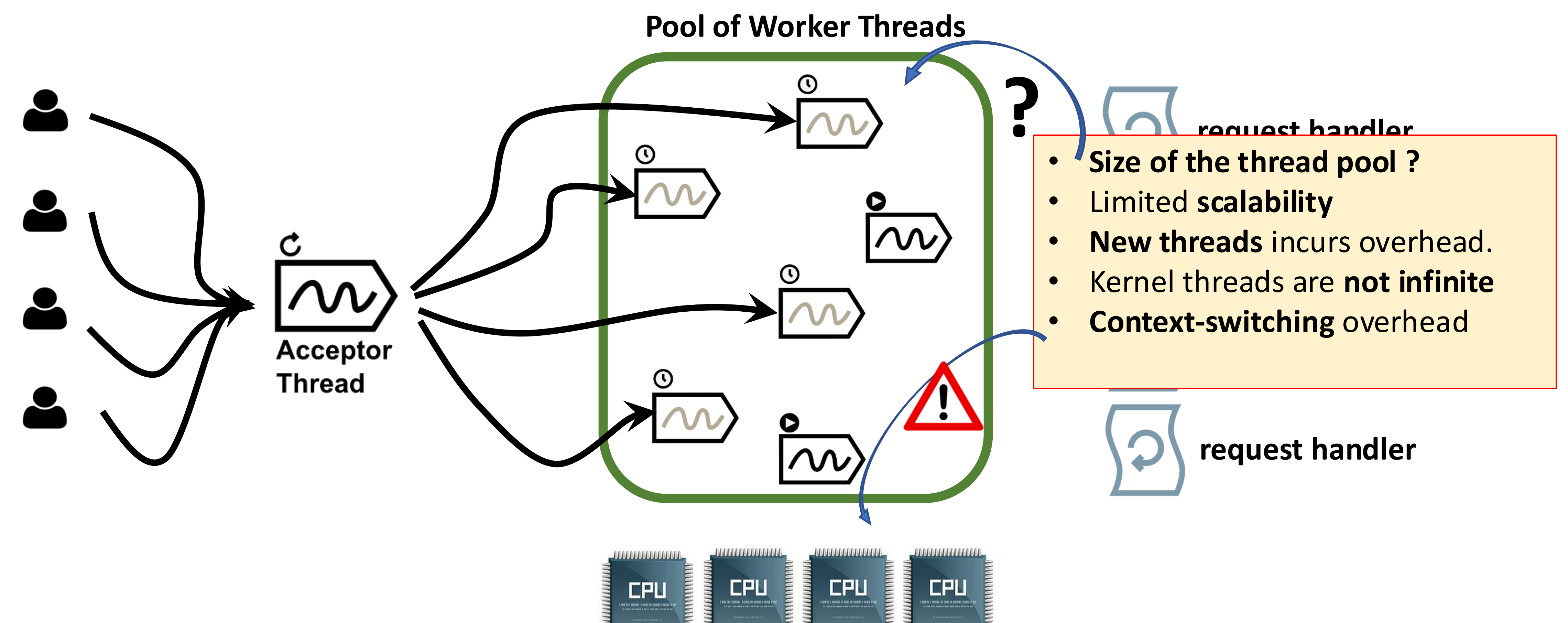
Enhancing SSR in Low-Thread Web Servers

A Comprehensive Approach for Progressive Server-Side Rendering with Any Asynchronous API and Multiple Data Models

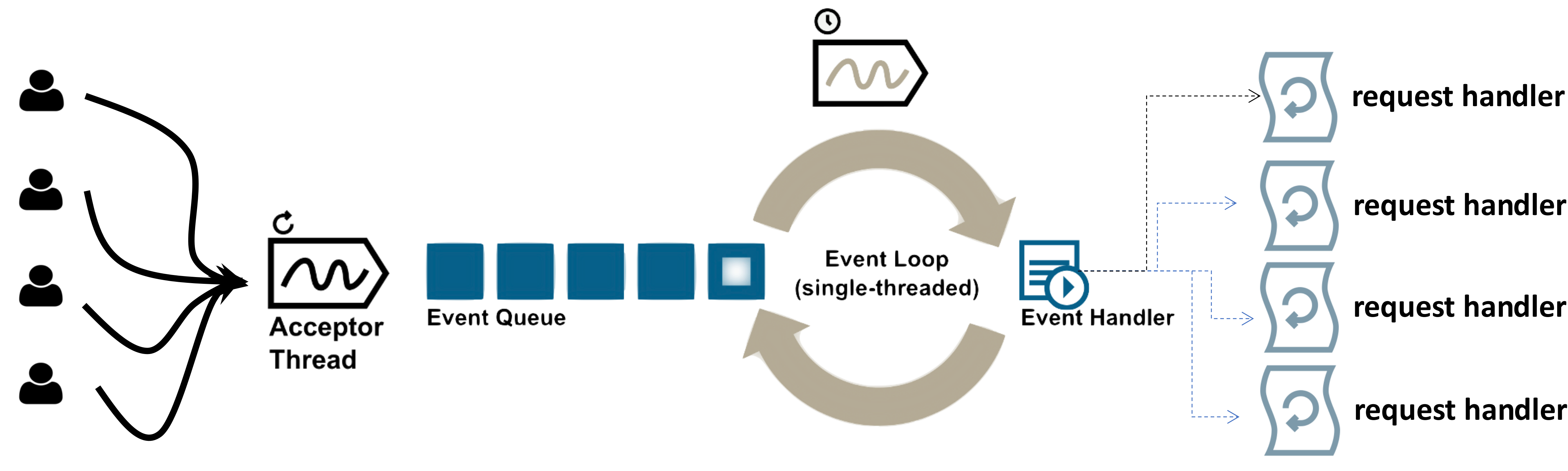
Thread-per-request architecture



Thread-per-request architecture - Issues



Low-thread architectures (aka event-driven)



Low-thread architectures (aka event-driven)

Acceptor Thread

Event Queue

Event Loop (single-threaded)

Event Handler

Handle many concurrent request

request handler

request handler

request handler

request handler

Cannot block

Low-thread architectures (aka event-driven)

- Node.js

- Netty

- Akka HTTP

- Vert.X

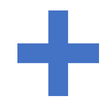
- Spring WebFlux

Low-thread architectures - Issues

- Low-thread use non-blocking I/O => Asynchronous

- SSR Web Templates => Synchronous (e.g. java Iterator)

```java
List<Track> tracks = asList(
  new Track("Space Oddity"),
  new Track("Forest"),
  new Track("Just like honey")
);
```



KotlinX.html

```kotlin
html {
  head { title("Playlist") }
  body {
    table {
      attributes["border"] = "1"
      tr { th { text("Track Name") } }
      tracks
        .forEach { track ->
          tr { td { text(track.name) } }
        }
    } // table
  } // body
} // html
```

```html
<html>
  <head>
    <title>Playlist</title>
  </head>
   <body>
    <table border="1">
      <tr><th>Track Name</th></tr>
      <tr><td>"Space oddity"</td></tr>
      <tr><td>"Forest"</td></tr>
      <tr><td>"Just like honey"</td></tr>
    </table>
  </body>
</html>
```

Synchronous <versus> Asynchronous

Iterator

```
<T> void resolve(Iterator<T> tracks) {
    ...
    T track = tracks.next();
    appendHtml(track.name);
    ...
    ...
    ...
}
```

Synchronous <versus> Asynchronous

Iterator

```
<T> void resolve(Iterator<T> tracks) {
    ...
    T track = tracks.next();
    appendHtml(track.name);
    ...
    ...
    ...
}
```

Observable

```
<T> void resolve(Observable<T> tracks) {
    ...
    tracks.doOnNext(track -> {
        appendHtml(track.name);
        ...
    });
    ...
}
```

Synchronous <versus> Asynchronous

Iterator

```
<T> void resolve(Iterator<T> tracks) {
    ...
    T track = tracks.next();
    appendHtml(track.name);
    ...
    ...
    ...
}
```

Termination

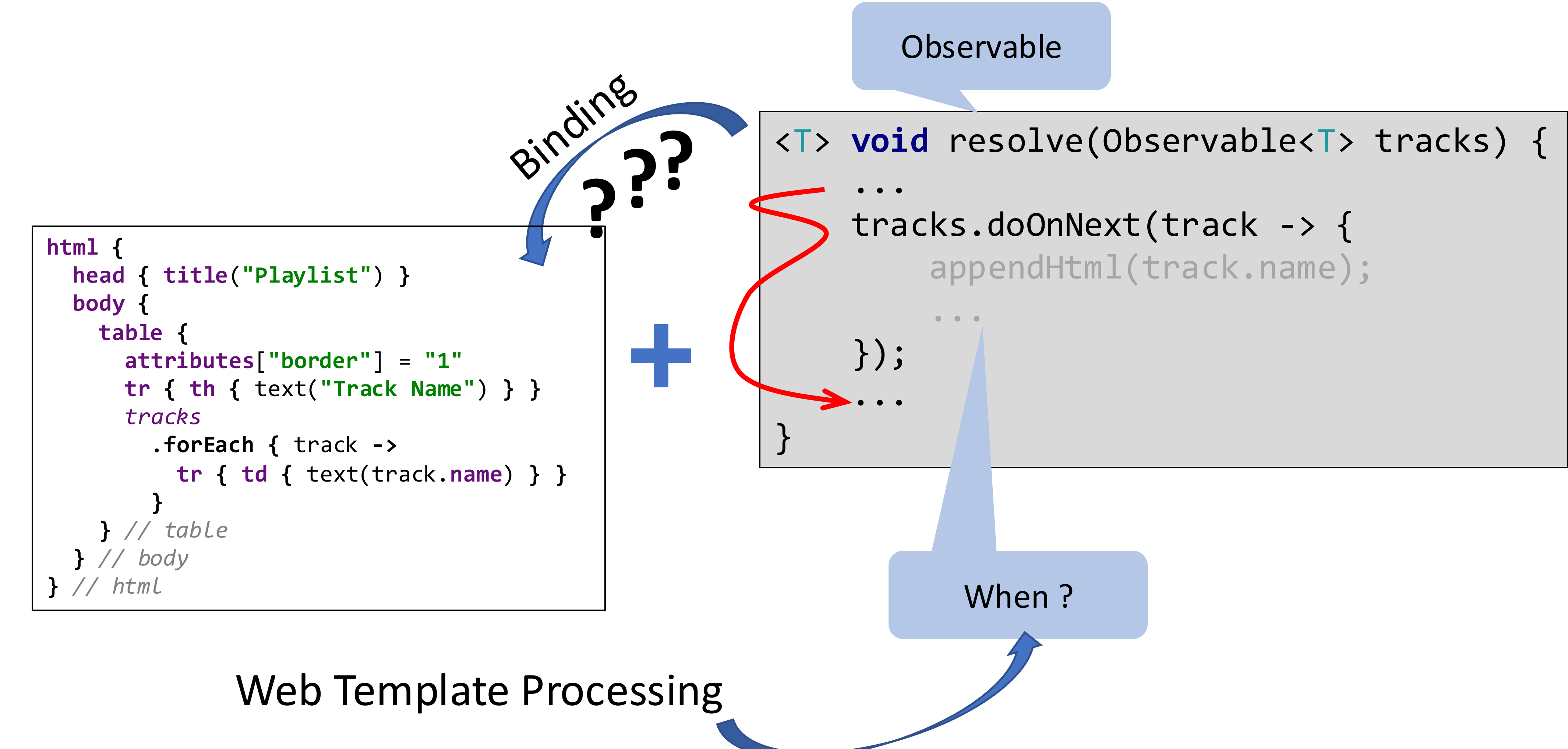
Observable

```
<T> void resolve(Observable<T> tracks) {
    ...
    tracks.doOnNext(track -> {
        appendHtml(track.name);
        ...
    });
    ...
}
```

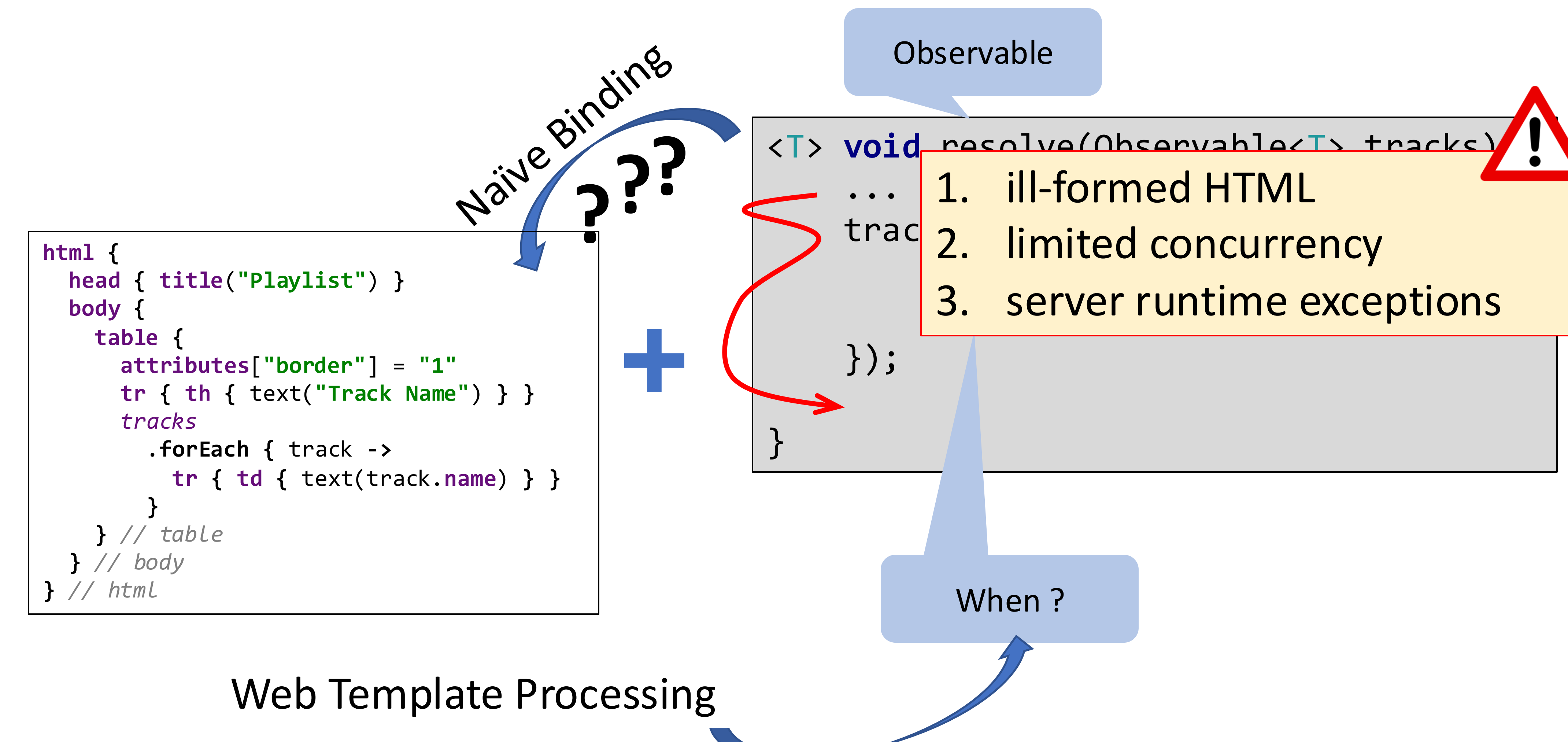
When ?

Web Template Processing

Synchronous <versus> Asynchronous Binding



Synchronous <versus> Asynchronous Binding Issues



Naïve Asynchronous Binding Issues

e.g. HTTP Response channel

Model

```
fun template(resp: Appendable, tracks: Iterable<Track>) {
  resp
    .appendHTML()
      .html {
        head { title("Playlist") }
        body {
          table {
            attributes["border"] = "1"
            tr { th { text("Track Name") } }
            tracks
              .forEach { track ->
                tr { td { text(track.name) } }
              }
          } // table
        } // body
      } // html
}
```

Synchronous

Naïve Asynchronous Binding Issues

e.g. HTTP Response channel

Model

Asynchronous

```
fun template(resp: Appendable, tracks: Observable<Track>) {
  resp
    .appendHTML()
      html {
        head { title("Playlist") }
        body {
          table {
            attributes["border"] = "1"
            tr { th { text("Track Name") } }
            tracks
              .doOnNext { track ->
                tr { td { text(track.name) } }
              }
              .subscribe()
          } // table
        } // body
      } // html
}
```

Naïve Asynchronous Binding Issues

e.g. HTTP Response channel

Model

```
fun template(resp: Appendable, tracks: Observable<Track>) {
  resp
    .appendHTML()
      html {
        head { title("Playlist") }
        body {
          table {
            attributes["border"] = "1"
            tr { th { text("Track Name") } }
            tracks
              .doOnNext { track ->
                tr { td { text(track.name) } }
              }
              .subscribe()
          } // table
        } // body
      } // html
}
```

Asynchronous

Later

Naïve Asynchronous Binding Issues

e.g. HTTP Response channel

Model

```
fun template(resp: Appendable, tracks: Observable<Track>) {
  resp
    .appendHTML()
      html {
        head { title("Playlist") }
        body {
          table {
            attributes["border"] = "1"
            tr { th { text("Track Name") } }
            tracks
              .doOnNext { track ->
                tr { td { text(track.name) } }
              }
              .subscribe()
          } // table
        } // body
      } // html
}
```

Asynchronous

Later

```
<html>
  <body>
    <table border="1">
      <tr><th>Track name</th></tr>
    </table>
  </body>
</html>
<tr><td>Space Oddity</td></tr>
<tr><td>Forest</td></tr>
<tr><td>Just like honey</td></tr>
```

Playlist

localhost:8080/kotlinx/reactive/pl...

Track Name

Space Oddity Forest Just like honey

Naïve Asynchronous Binding Issues

e.g. HTTP Response channel

Model

```
fun template(resp: Appendable, tracks: Observable<Track>) {
  resp
    .appendHTML()
      html {
        head { title("Playlist") }
        body {
          table {
            attributes["border"] = "1"
            tr { th { text("Track Name") } }
            tracks
              .doOnNext { track ->
                tr { td { text(track.name) } }
              }
              .subscribe()
          } // table
        } // body
      } // html
}
```

Asynchronous

Later

1. ill-formed HTML
2. limited concurrency
3. server runtime exceptions

```
<html>
  <body>
    <table border="1">
      <tr><th>Track name</th></tr>
    </table>
  </body>
</html>
<tr><td>Space Oddity</td></tr>
<tr><td>Forest</td></tr>
<tr><td>Just like honey</td></tr>
```

Playlist

localhost:8080/kotlinx/reactive/pl...

Track Name

Space Oddity Forest Just like honey

Naïve Asynchronous Binding Issues

- Asynchronous to Synchronous ?

```
fun template(resp: Appendable, tracks: Observable<Track>) {
  resp
    .appendHTML()
      html {
        head { title("Playlist") }
        body {
          table {
            attributes["border"] = "1"
            tr { th { text("Track Name") } }
            tracks
              .doOnNext { track ->
                tr { td { text(track.name) } }
              }
              .subscribe()
          } // table
        } // body
      } // html
}
```

Naïve Asynchronous Binding Issues

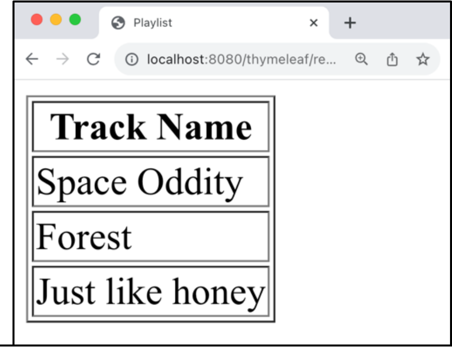
- Asynchronous to Synchronous ?

```
fun template(resp: Appendable, tracks: Observable<Track>) {
  resp
   .appendHTML()
    html {
      head { title("Playlist") }
      body {
        table {
          attributes["border"] = "1"
          tr { th { text("Track Name") } }
          tracks.blockingIterable()
            .forEach { track ->
              tr { td { text(track.name) } }
            }
        } // table
      } // body
    } // html
}
```

Naïve Asynchronous Binding Issues

- Asynchronous to Synchronous ?

```
fun template(resp: Appendable, tracks: Observable<Track>) {
  resp
    .appendHTML()
      html {
        head { title("Playlist") }
        body {
          table {
            attributes["border"] = "1"
            tr { th { text("Track Name") } }
            tracks.blockingIterable()
              .forEach { track ->
                tr { td { text(track.name) } }
              }
          } // table
        } // body
      } // html
}
```




```
<html>
   <head>
      <title>Playlist</title>
   </head>
    <body>
      <table border="1">
        <tr><th>Track Name</th></tr>
        <tr><td>"Space oddity"</td></tr>
        <tr><td>”Forest"</td></tr>
        <tr><td>”Just like honey"</td></tr>
      </table>
   </body>
</html>
```

Naïve Asynchronous Binding Issues

- Asynchronous to Synchronous ?

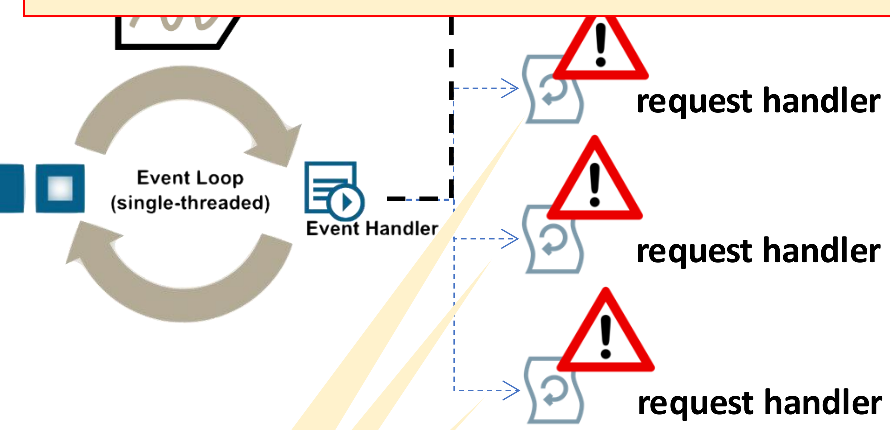
```
fun template(resp: Appendable, tracks: Observable<Track>) {
  resp
    .appendHTML()
     html {
       head { title("Playlist") }
       body {
         table {
           attributes["border"] = "1"
           tr { th { text("Track Name") } }
           tracks.blockingIterable()
             .forEach { track ->
               tr { td { text(track.name) } }
             }
         } // table
       } // body
     } // html
}
```

Naïve Asynchronous Binding Issues

- Asynchronous to Synchronous ?

```
fun template(resp: Appendable, tracks: Observable<Track>) {
  resp
   .appendHTML()
    html {
     head { title("Playlist") }
     body {
      table {
       attributes["border"] = "1"
       tr { th { text("Track Name") } }
       tracks.blockingIterable()
        .forEach { track ->
         tr { td { text(track.name) } }
        }
      } // table
     } // body
    } // html
}
```

1. ill-formed HTML
2. limited concurrency
3. server runtime exceptions



Waiting !!!

Naïve Asynchronous Binding Issues

- Asynchronous to Synchronous ?

```
fun template(resp: Appendable, tracks: Observable<Track>) {
  resp
   .appendHTML()
    html {
      head { title("Playlist") }
      body {
        table {
          attributes["border"]
          tr { th { text("Track
          tracks.blockingIterab
           .forEach { track ->
             tr { td { text(tr
           }
        } // table
      } // body
    } // html
}
```

IllegalStateException: blocking, which is not supported!

```
spring-asynchronous-views — java -Xmx64m -Xms64m -Dorg.gradle.appname=gradlew -classpath ~/Library/CloudStorage/Dropbox/ISE

 :: Spring Boot ::                (v2.7.8)

2023-11-09 15:41:02.802 ERROR 22556 --- [ctor-http-nio-1] a.w.r.e.AbstractErrorWebExceptionHandler : [8e
/blocking/artist/onerepublic"

java.lang.IllegalStateException: block()/blockFirst()/blockLast() are blocking, which is not supported i
        at reactor.core.publisher.BlockingSingleSubscriber.blockingGet(BlockingSingleSubscriber.java:83)
        Suppressed: reactor.core.publisher.FluxOnAssembly$OnAssemblyException:
Error has been observed at the following site(s):
        *__checkpoint ⇢ HTTP GET "/thymeleaf/blocking/artist/onerepublic" [ExceptionHandlingWebHandler]
```

Naïve Asynchronous Binding Issues

- Asynchronous to Synchronous ?

```
fun template(resp: Appendable, tracks: Observable<Track>) {
  resp
   .appendHTML()
    html {
     head { title("Playlist") }
     body {
      table {
       attributes["border"] = "1"
       tr { th { text("Track Name") } }
       tracks.blockingIterable()
        .forEach { track ->
         tr { td { text(track.name) } }
        }
      } // table
     } // body
    } // html
}
```

1. ill-formed HTML
2. limited concurrency
3. server runtime exceptions

Enhancing SSR in Low-Thread Web Servers

A Comprehensive Approach for Progressive Server-Side Rendering with Any Asynchronous API and Multiple Data Models

Higher Order Templates – Background Sync

e.g. HTTP Response channel

Model

```
fun template(resp: Appendable, tracks: Iterable<Track>) {
  resp
    .appendHTML()
      .html {
        head { title("Playlist") }
        body {
          table {
            attributes["border"] = "1"
            tr { th { text("Track Name") } }
            tracks
              .forEach { track ->
                tr { td { text(track.name) } }
              }
          } // table
        } // body
      } // html
}
```

```
template(appendableResponse, tracks)
```

Higher Order Templates – Background Sync

KotlinX.Html

```kotlin
fun template(resp: Appendable, tracks: Iterable<Track>) {
  resp
    .appendHTML()
      .html {
        head { title("Playlist") }
        body {
          table {
            attributes["border"] = "1"
            tr { th { text("Track Name") } }
            tracks
              .forEach { track ->
                tr { td { text(track.name) } }
              }
          } // table
        } // body
      } // html
}
```

HtmlFlow

```kotlin
val view = HtmlFlow.view { page -> page
  .html()
    .head().title().text("Playlist").__().__()
    .body()
      .table().attrBorder(EnumBorderType._1)
        .tr().th().text("Track name").__().__()
        .dynamic<Iterable<Track>>{ table, tracks ->
          tracks
            .forEach{ track -> table
              .tr().td().text(track.name).__().__()
            }
      .__() // table
    .__() // body
  .__() // html
}
```

```kotlin
template(appendableResponse, tracks)
```

```kotlin
view.setOut(appendableResponse).write(tracks)
```

Higher Order Templates – Background Sync

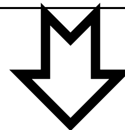
KotlinX.Html

```
fun template(resp: Appendable, tracks: Iterable<Track>) {
  resp
    .appendHTML()
      .html {
        head { title("Playlist") }
        body {
          table {
            attributes["border"] = "1"
            tr { th { text("Track Name") } }
            tracks
              .forEach { track ->
                tr { td { text(track.name) } }
              }
          } // table
        } // body
      } // html
}
```

HtmlFlow

```
val view = HtmlFlow.view { page -> page
  .html()
    .head().title().text("Playlist").__().__()
    .body()
      .table().attrBorder(EnumBorderType._1)
        .tr().th().text("Track name").__().__()
        .dynamic<Iterable<Track>>{ table, tracks ->
          tracks
            .forEach{ track -> table
              .tr().td().text(track.name).__().__()
            }
      .__() // table
    .__() // body
  .__() // html
}
```

- Web Templates
- Pure Functions
- Composable



```
template(appendableResponse, tracks)
```

```
view.setOut(appendableResponse).write(tracks)
```

Higher Order Templates

Higher Order Templates – Background Sync

KotlinX.Html

```
fun template(resp: Appendable, tracks: Iterable<Track>) {
  resp
    .appendHTML()
      .html {
        head { title("Playlist") }
        body {
          table {
            attributes["border"] = "1"
            tr { th { text("Track Name") } }
            tracks
              .forEach { track ->
                tr { td { text(track.name) } }
              }
          } // table
        } // body
      } // html
}
```

1. Progressive rendering
2. Any kind of data model API
3. Well-formed HTML

HtmlFlow

```
val view = HtmlFlow.view { page -> page
  .html()
    .head().title().text("Playlist").__().__()
    .body()
      .table().attrBorder(EnumBorderType._1)
        .tr().th().text("Track name").__().__()
        .dynamic<Iterable<Track>>{ table, tracks ->
          tracks
            .forEach{ track -> table
              .tr().td().text(track.name).__().__()
            }
      .__() // table
    .__() // body
  .__() // html
}
```

```
<html>
<body>
     <table border="1">
       <tr><th>Track Name</th></tr>
       <tr><td>"Space oddity"</td></tr>
       <tr><td>”Forest"</td></tr>
       <tr><td>”Just like honey"</td></tr>
     </table>
   </body>
</html>
```

Track Name
Space Oddity
Forest
Just like honey

Higher Order Templates – Asynchronous ?

KotlinX.Html

```kotlin
fun template(resp: Appendable, tracks: Observable<Track>) {
  resp
    .appendHTML()
      .html {
        head { title("Playlist") }
        body {
          table {
            attributes["border"] = "1"
            tr { th { text("Track Name") } }
            tracks
              .doOnNext { track ->
                tr { td { text(track.name) } }
              }
              .subscribe()
          } // table
        } // body
      } // html
}
```

1. Progressive rendering ✔
2. Any kind of data model API ✔
3. **Ill-formed HTML** ⚠
4. Non-blocking ✔
5. No server runtime exceptions ✔

HtmlFlow

```kotlin
val view = HtmlFlow.view { page -> page
  .html()
    .head().title().text("Playlist").__().__()
    .body()
      .table().attrBorder(EnumBorderType._1)
        .tr().th().text("Track name").__().__()
        .dynamic<Observable<Track>>{ table, tracks ->
          tracks
            .doOnNext { track -> table
              .tr().td().text(track.name).__().__()
            }
            .subscribe()
      .__() // table
    .__() // body
  .__() // html
}
```

```html
<html>
  <body>
    <table border="1">
      <tr><th>Track name</th></tr>
    </table>
  </body>
</html>
<tr><td>Space Oddity</td></tr>
<tr><td>Forest</td></tr>
<tr><td>Just like honey</td></tr>
```

Playlist

localhost:8080/html...

Track name

Space Oddity

localhost:8080/html...

Track name

Space Oddity Forest

Playlist

localhost:8080/html...

Track name

Space Oddity Forest Just like honey

Higher Order Templates – Asynchronous ?

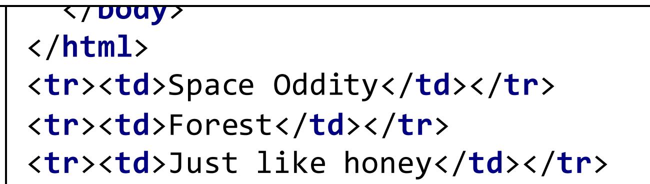
KotlinX.Html

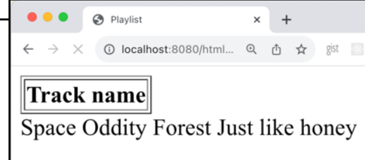
```
fun template(resp: Appendable, tracks: Observa
  resp
    .appendHTML()
      .html {
        head { title("Playlist") }
        body {
          table {
            attributes["border"] = "1"
            tr { th { text("Track Name") } }
            tracks
              .doOnNext { track ->
                tr { td { text(track.name) } }
              }
              .subscribe()
          } // table
        } // body
      } // html
}
```

HtmlFlow

```
val view = HtmlFlow.view { page -> page
  .html()
    .head().title().text("Playlist").__().__()
    .body()
      .table().attrBorder(EnumBorderType._1)
        .tr().th().text("Track name").__().__()
        .dynamic<Observable<Track>>{ table, tracks ->
          tracks
            .doOnNext { track -> table
              .tr().td().text(track.name).__().__()
            }
            .subscribe()
      .__() // table
    .__() // body
  .__() // html
}
```

1. Progressive rendering ✓
2. Any kind of data model API ✓
3. **Ill-formed HTML** ⚠
4. Non-blocking ✓
5. No server runtime exceptions ✓





Higher Order Templates – HtmlFlow - Background

Goals

- Progressive Render

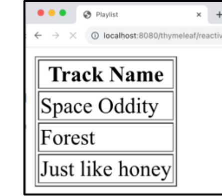
- Type Safety

- Performance

Higher Order Templates – HtmlFlow

Goals

- Progressive Render



Playlist
localhost:8080/thymeleaf/reactiv...
Track Name
Space Oddity
Forest
Just like honey

```
.html()
  .head().title().text("Playlist").__().__()
  .body()
    .table().attrBorder(EnumBorderType._1)
      .tr().th().text("Track name").__().__()
      .dynamic<List<Track>>{ table, tracks -> tracks
        .forEach { track -> table
          .tr().td().text(track.name).__().__()
        }
      }
    .__() // table
  .__() // body
.__() // html
```

- Type Safety

- Performance

Higher Order Templates – HtmlFlow – **Performance**

Goals

- Progressive Render
- Type Safety
- Performance

Track Name
Space Oddity

Track Name
Space Oddity
Forest

Track Name
Space Oddity
Forest
Just like honey

```
.html()
  .head().title().text("Playlist").__().__()
  .body()
    .table().attrBorder(EnumBorderType._1)
      .tr().th().text("Track name").__().__()
      .dynamic<Observable<Track>>{ table, tracks -> tracks
        .forEach { track -> table
          .tr().td().text(track.name).__().__()
        }
      }
    .__() // table
  .__() // body
.__() // html
```

```
"<html>
  <head>
    <title>Playlist</title>
  </head>
  <body>
  <table border="1">
    <tr><th>Track Name</th></tr>"
```

Static HTML resolved 1 x

```
"    </table>
  </body>
</html>"
```

Higher Order Templates – HtmlFlow

HtmlFlow 4.0

```
val view = HtmlFlow.view { page -> page
  .html()
    .body()
      .h3().dynamic<Artist> {
        h3, m -> h3.text(m.artistName)
      }
      .__() // h3
      .h3().text("MusicBrainz info:").__()
      .ul().dynamic<Artist> { ul, m -> m.musicBrainz
        ...
      }
      .__() // ul
      .b().text("Spotify popular tracks:").__()
      .div().dynamic<Artist> { div, m -> m.spotify
        ...
      }
      .__() // div
    .__() // body
  .__() // html
}
```

```
view.setOut(appendableResp).write(tracks)
```

localhost:8080/htmlflow/react

localhost:8080/...

David Bowie

MusicBrainz info:

- Founded: 1960
- From: New York
- Genre: Pop Rock

Spotify popular tracks: Heroes, Starman, Rebel Rebel, Space Oddity, Let's dance,

Higher Order Templates – HtmlFlow

HtmlFlow 4.0

```
HtmlFlow.view { page -> page
  .html()
    .body()
      .h3().dynamic<Artist> {
        h3, m -> h3.text(m.artistName)
      }
      .__() // h3
      .h3().text("MusicBrainz info:").__()
      .ul().dynamic<Artist> { ul, m -> m.musicBrainz
        ...
      }
      .__() // ul
      .b().text("Spotify popular tracks:").__()
      .div().dynamic<Artist> { div, m -> m.spotify
        ...
      }
      .__() // div
    .__() // body
  .__() // html
}
```

```
view.setOut(appendableResp).write(tracks)
```

“<html><body><h3>”

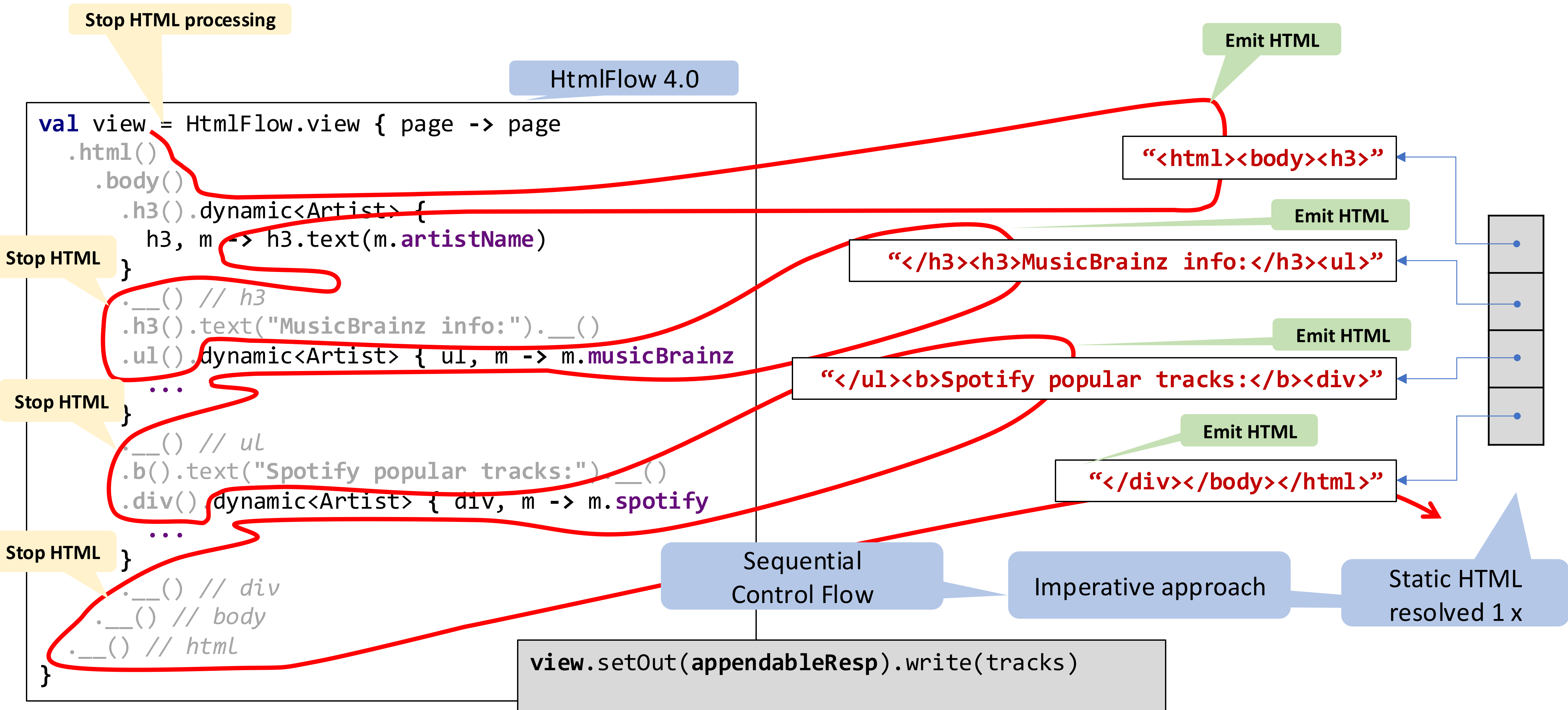
“</h3><h3>MusicBrainz info:</h3>
”

“
Spotify popular tracks:<div>”

“</div></body></html>”

Static HTML resolved 1 x

Higher Order Templates – HtmlFlow



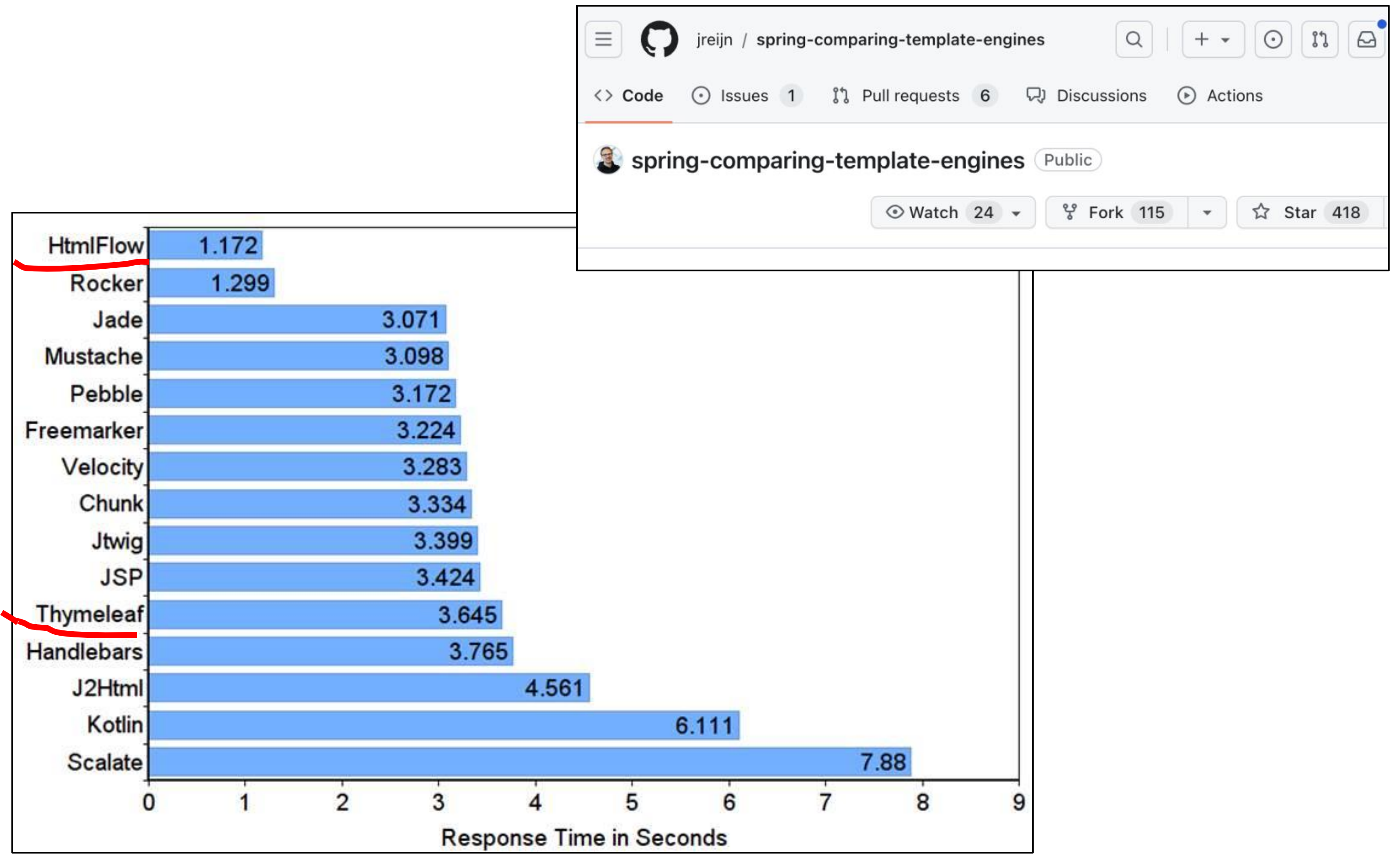
Higher Order Templates – HtmlFlow – **Performance**

Goals

- Progressive Render

- Type Safety

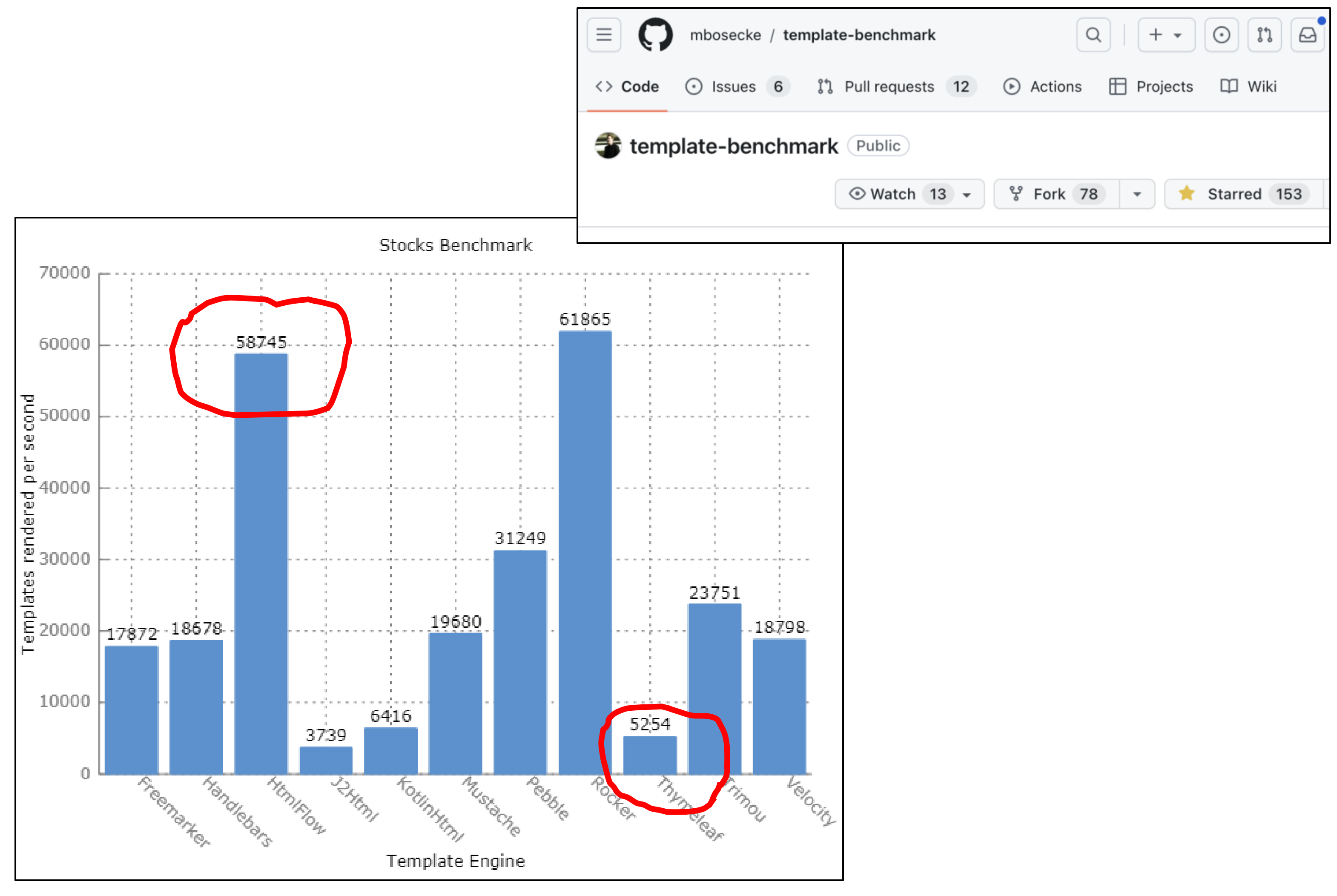
- Performance



Higher Order Templates – HtmlFlow – **Performance**

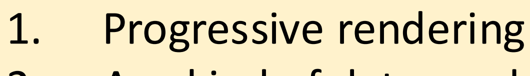
Goals

- Progressive Render
- Type Safety
- Performance



Higher Order Templates – HtmlFlow – Asynchronous

How to solve de **asynchronous** issue with ill-formed HTML?



1. Progressive rendering
2. Any kind of data model API
3. **<u>Ill-formed HTML</u>**
4. Non-blocking
5. No server runtime exceptions

Higher Order Templates – HtmlFlow

HtmlFlow 4.0

```
HtmlFlow.view { page -> page
  .html()
    .body()
      .h3().dynamic<Artist> {
        h3, m -> h3.text(m.artistName)
      }
      .__() // h3
      .h3().text("MusicBrainz info:").__()
      .ul().dynamic<Artist> { ul, m -> m.musicBrainz
        ...
      }
      .__() // ul
      .b().text("Spotify popular tracks:").__()
      .div().dynamic<Artist> { div, m -> m.spotify
        ...
      }
      .__() // div
    .__() // body
  .__() // html
}
```

```
view.setOut(appendableResp).write(tracks)
```

“<html><body><h3>”

“</h3><h3>MusicBrainz info:</h3>
”

“
Spotify popular tracks:<div>”

“</div></body></html>”

Static HTML resolved 1 x

Turn in Continuations

```
interface HtmlContinuation<U> {
  void execute(U model);
}
```

HtmlFlow 4.0

```
HtmlFlow.view { page -> page
  .html()
    .body()
      .h3().dynamic<Artist> {
        h3, m -> h3.text(m.artistName)
      }
      .__() // h3
      .h3().text("MusicBrainz info:").__()
      .ul().dynamic<Artist> { ul, m -> m.musicBrainz
        ...
      }
      .__() // ul
      .b().text("Spotify popular tracks:").__()
      .div().dynamic<Artist> { div, m -> m.spotify
        ...
      }
      .__() // div
    .__() // body
  .__() // html
}
```

```
model -> { visitor.write("<html><body><h3>"); next() }

model -> { visitor.write("</h3><h3>Music…"); next() }

model -> { visitor.write("</ul><b>Spotify…"); next() }

model -> { visitor.write("</div></body></html>"); next() }
```

Static HTML resolved 1 x

```
view.setOut(appendableResp).write(tracks)
```

Turn in Continuations

```
interface HtmlContinuation<U> {
  void execute(U model);
}
```

HtmlFlow 4.0

```
val view = HtmlFlow.view { page -> page
  .html()
    .body()
      .h3().dynamic<Artist> {
        h3, m -> h3.text(m.artistName)
      }
      .__() // h3
      .h3().text("MusicBrainz info:").__()
      .ul().dynamic<Artist> { ul, m -> m.musicBrainz
         ...
      }
      .__() // ul
      .b().text("Spotify popular tracks:").__()
      .div().dynamic<Artist> { div, m -> m.spotify
         ...
      }
      .__() // div
    .__() // body
  .__() // html
}
```

```
model -> { visitor.write("<html><body><h3>"); next() }
model -> { consumer.accept(element, model); next() }
model -> { visitor.write("</h3><h3>Music…"); next() }
model -> { consumer.accept(element, model); next() }
model -> { visitor.write("</ul><b>Spotify…"); next() }
model -> { consumer.accept(element, model); next() }
model -> { visitor.write("</div></body></html>"); next() }
```

Chain-of-responsibility pattern

Turn in Continuations

```
interface HtmlContinuation<U> {
  void execute(U model);
}
```

HtmlFlow 4.0

```
val view = HtmlFlow.view { page -> page
  .html()
    .body()
      .h3().dynamic<Artist> {
        h3, m -> h3.text(m.artistName)
      }
      .__() // h3
      .h3().text("MusicBrainz info:").__()
      .ul().dynamic<Artist> { ul, m -> m.musicBrainz
        ...
      }
      .__() // ul
      .b().text("Spotify popular tracks:").__()
      .div().dynamic<Artist> { div, m -> m.spotify
        ...
      }
      .__() // div
    .__() // body
  .__() // html
}
```

```
model -> { visitor.write(“<html><body><h3>”); next() }
model -> { consumer.accept(element, model); next() }
model -> { visitor.write(“</h3><h3>Music…”); next() }
model -> { consumer.accept(element, model); next() }
model -> { visitor.write(“</ul><b>Spotify…”); next() }
model -> { consumer.accept(element, model); next() }
model -> { visitor.write(“</div></body></html>”); next() }
```

Turn in Continuations - Async

```
interface HtmlContinuation<U> {
  void execute(U model);
}
```

HtmlFlow 4.0

```
val view = HtmlFlow.view { page -> page
  .html()
    .body()
      .h3().dynamic<Artist> {
        h3, m -> h3.text(m.artistName)
      }
      .__() // h3
      .h3().text("MusicBrainz info:").__()
      .ul().dynamic<Artist> { ul, m -> m.musicBrainz.doOnNext { mb ->
        ...
      }}
      .b().text("Spotify popular tracks:").__()
      .div().dynamic<Artist> { div, m -> m.spotify.doOnNext { sp ->
        ...
      }
      .__() // div
    .__() // body
  .__() // html
}
```

When to call the next continuation?

```
r.write("<html><body><h3>"); next() }
er.accept(element, model); next() }
r.write("</h3><h3>Music…"); next() }
er.accept(element, model); next() }
r.write("</ul><b>Spotify…"); next() }
er.accept(element, model); next() }
r.write("</div></body></html>"); next() }
```

When completed?

? ? ? ?

Turn in Continuations - Async

```
interface HtmlContinuation<U> {
  void execute(U model);
}
```

```
public interface AwaitConsumer<T extends Element, U> {
    void accept(T parent, U model, OnCompletion cb);
}
```

HtmlFlow 4.0

```
val view = HtmlFlow.view { page -> page
  .html()
    .body()
      .h3().dynamic<Artist> {
        h3, m -> h3.text(m.artistName)
      }
      .__() // h3
      .h3().text("MusicBrainz info:").__()
      .ul().await<Artist> { ul, m, cb -> m.musicBrainz.doOnNext { mb ->
        ...
        cb.finish()
      }}
      .__() // ul
      .b().text("Spotify popular tracks:").__()
      .div().await<Artist> { div, m, cb -> m.spotify.doOn
        ...
      }
      .__() // div
    .__() // body
  .__() // html
}
```

```
r.write("<html><body><h3>"); next() }
er.accept(element, model); next() }
r.write("</h3><h3>Music…"); next() }
model -> { consumer.accept(element, model); next() }
r.write("</ul><b>Spotify…"); next() }
model -> { consumer.accept(element, model); next() }
r.write("</div></body></html>"); next() }
```

Idea of the *resume* function in coroutines (Haynes et al., 1984)

Turn in Continuations - Async

```
interface HtmlContinuation<U> {
  void execute(U model);
}
```

```
public interface AwaitConsumer<T extends Element, U> {
    void accept(T parent, U model, OnCompletion cb);
}
```

HtmlFlow 4.0

```
val view = HtmlFlow.view { page -> page
  .html()
    .body()
      .h3().dynamic<Artist> {
        h3, m -> h3.text(m.artistName)
      }
      .__() // h3
      .h3().text("MusicBrainz info:").__()
      .ul().await<Artist> { ul, m, cb -> m.musicBrainz.doOnNext { mb ->
        ...
        cb.finish()
      }}
      .__() // ul
      .b().text("Spotify popular tracks:").__()
      .div().await<Artist> { div, m, cb -> m.spotify.doOn
        ...
        cb.finish()
      }
      .__() // div
    .__() // body
  .__() // html
}
```

```
r.write("<html><body><h3>"); next() }
er.accept(element, model); next() }
r.write("</h3><h3>Music…"); next() }
model -> { consumer.accept(element, model, () -> next()) }
r.write("</ul><b>Spotify…"); next() }
model -> { consumer.accept(element, model, () -> next()) }
r.write("</div></body></html>"); next() }
```

Idea of the *resume* function in coroutines (Haynes et al., 1984)

Turn in Continuations - Async

```java
interface HtmlContinuation<U> {
  void execute(U model);
}
```

```java
public interface AwaitConsumer<T extends Element, U> {
    void accept(T parent, U model, OnCompletion cb);
}
```

HtmlFlow 4.0

```kotlin
val view = HtmlFlow.view { page -> page
  .html()
    .body()
      .h3().dynamic<Artist> {
        h3, m -> h3.text(m.artistName)
      }
      .__() // h3
      .h3().text("MusicBrainz info:").__()
      .ul().await<Artist> { ul, m, cb -> m.musicBrainz.doOnNext { mb ->
        ...
        cb.finish()
      }}
      .__() // ul
      .b().text("Spotify popular tracks:").__()
      .div().await<Artist> { div, m, cb -> m.spotify.doOn
        ...
        cb.finish()
      }
      .__() // div
    .__() // body
  .__() // html
}
```

1. Progressive rendering
2. Any kind of data model API
3. **Well-formed HTML** ⚠
4. Non-blocking
5. No server runtime exceptions

```
ody><h3>"); next() }
, model); next() }
r.write("</h3><h3>Music…"); next() }
model -> { consumer.accept(element, model, () -> next()) }
r.write("</ul><b>Spotify…"); next() }
model -> { consumer.accept(element, model, () -> next()) }
r.write("</div></body></html>"); next() }
```

Idea of the *resume* function in coroutines (Haynes et al., 1984)

Future Directions

Continuations with suspend functions

```
val view = HtmlFlow.view { page -> page
  .html()
    .body()
      .h3().dynamic<Artist> {
        h3, m -> h3.text(m.artistName)
      }
      .__() // h3
      .h3().text("MusicBrainz info:").__()
      .ul().await<Artist> { ul, m, cb -> m.musicBrainz.doOnNext { mb ->
        ...
        cb.finish()
      }}
      .__() // ul
      .b().text("Spotify popular tracks:").__()
      .div().await<Artist> { div, m, cb -> m.spotify.doOnNext { sp ->
        ...
        cb.finish()
      }
      .__() // div
    .__() // body
  .__() // html
}
```

Continuations with suspend functions

```
val view = HtmlFlow.view { page -> page
  .html()
    .body()
      .h3().dynamic<Artist> {
        h3, m -> h3.text(m.artistName)
      }
      .__() // h3
      .h3().text("MusicBrainz info:").__()
      .ul().await<Artist> { ul, m, cb -> m.musicBrainz.doOnNext { mb ->
        ...
        cb.finish()
      }}
      .__() // ul
      .b().text("Spotify popular tracks:").__()
      .div().await<Artist> { div, m, cb -> m.spotify.doOnNext { sp ->
        ...
        cb.finish()
      }
      .__() // div
    .__() // body
  .__() // html
}
```

Avoid nesting lambdas!

Continuations with suspend functions

```
val view = HtmlFlow.view { page -> page
  .html()
    .body()
      .h3().dynamic<Artist> {
        h3, m -> h3.text(m.artistName)
      }
      .__() // h3
      .h3().text("MusicBrainz info:").__()
      .ul().await<Artist> { ul, m ->
        val mb = m.musicBrainz.await()
        ...

      }}
      .__() // ul
      .b().text("Spotify popular tracks:").__()
      .div().await<Artist> { div, m ->
         val sp = m.spotify.await()

      }
      .__() // div
    .__() // body
  .__() // html
}
```

Imperative approach

return

return

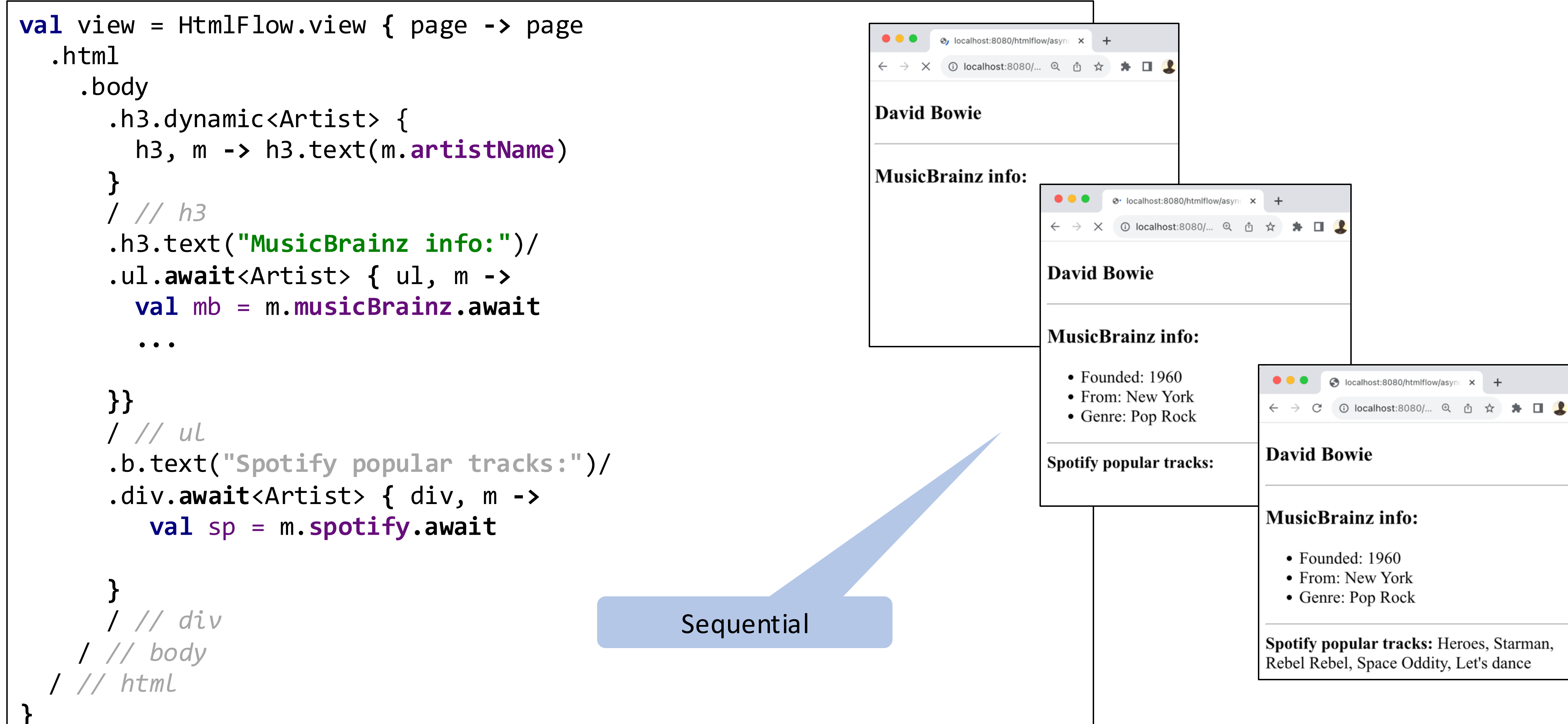
Kotlin idiomatic conventions

```
val view = HtmlFlow.view { page -> page
  .html
    .body
      .h3.dynamic<Artist> {
        h3, m -> h3.text(m.artistName)
      }
      / // h3
      .h3.text("MusicBrainz info:")/
      .ul.await<Artist> { ul, m ->
        val mb = m.musicBrainz.await
        ...

      }}
      / // ul
      .b.text("Spotify popular tracks:")/
      .div.await<Artist> { div, m ->
         val sp = m.spotify.await

      }
      o // div
    l // body
  o // html
}
```

Extension properties

Operator overloading

HtmlFlow Continuations await => Sequential



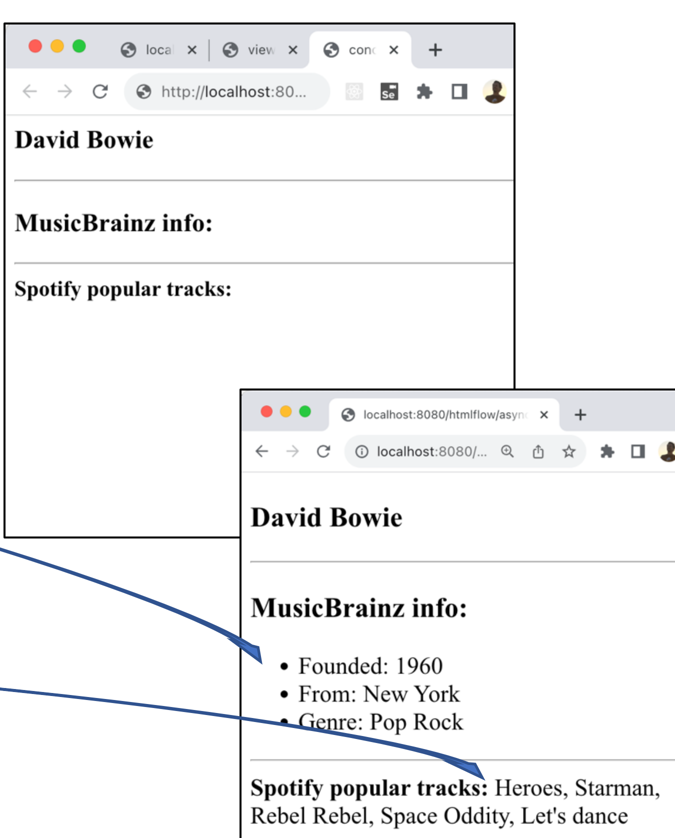
HtmlFlow Continuations async => Concurrent

```
val view = HtmlFlow.view { page -> page
  .html
    .body
      .h3.dynamic<Artist> {
        h3, m -> h3.text(m.artistName)
      }
      / // h3
      .h3.text("MusicBrainz info:")/
      .ul.async<Artist> { ul, m ->
        val mb = m.musicBrainz.await
        ...

      }}
      / // ul
      .b.text("Spotify popular tracks:")/
      .div.async<Artist> { div, m ->
         val sp = m.spotify.await

      }
      / // div
    / // body
  / // html
}
```

push on completion

push on completion

Concurrent

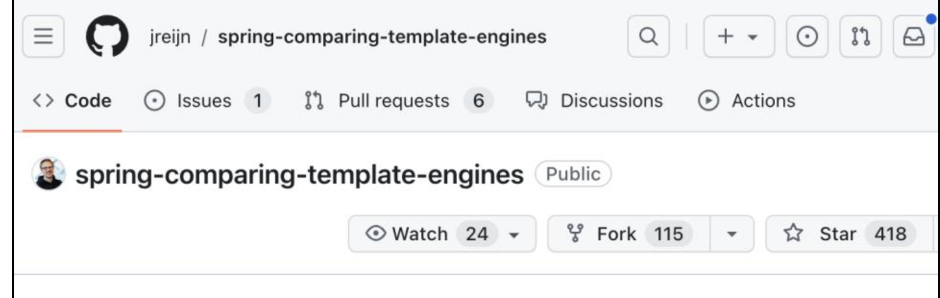


THANK YOU

Turn in Continuations – Async - Evaluation

Spring Comparing Template Engines

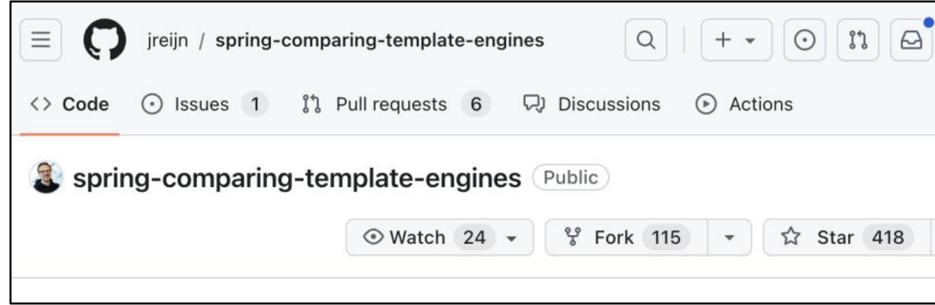
- JMH - Java Microbenchmark Harness
- Replaced Spring MVC by Spring WebFlux



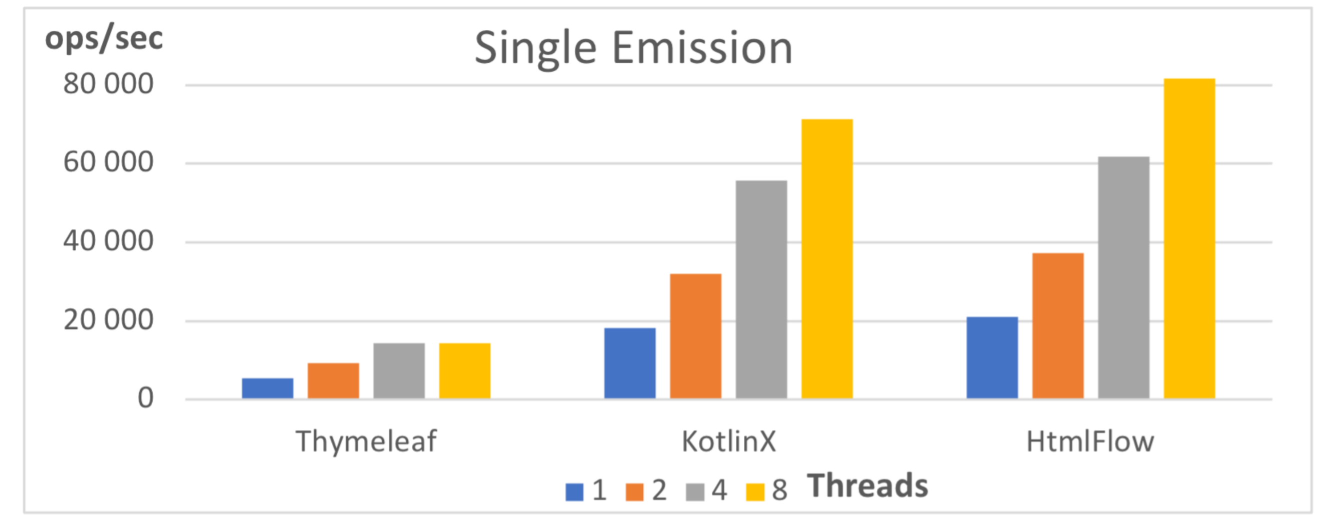
Turn in Continuations – Async - Evaluation

Spring Comparing Template Engines

- JMH - Java Microbenchmark Harness
- Replaced Spring MVC by Spring WebFlux



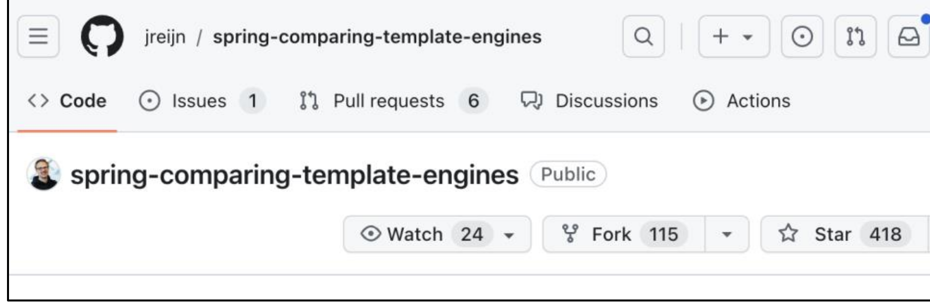
Single Emission (non-progressive) – requests ops/sec



Turn in Continuations – Async - Evaluation

Spring Comparing Template Engines

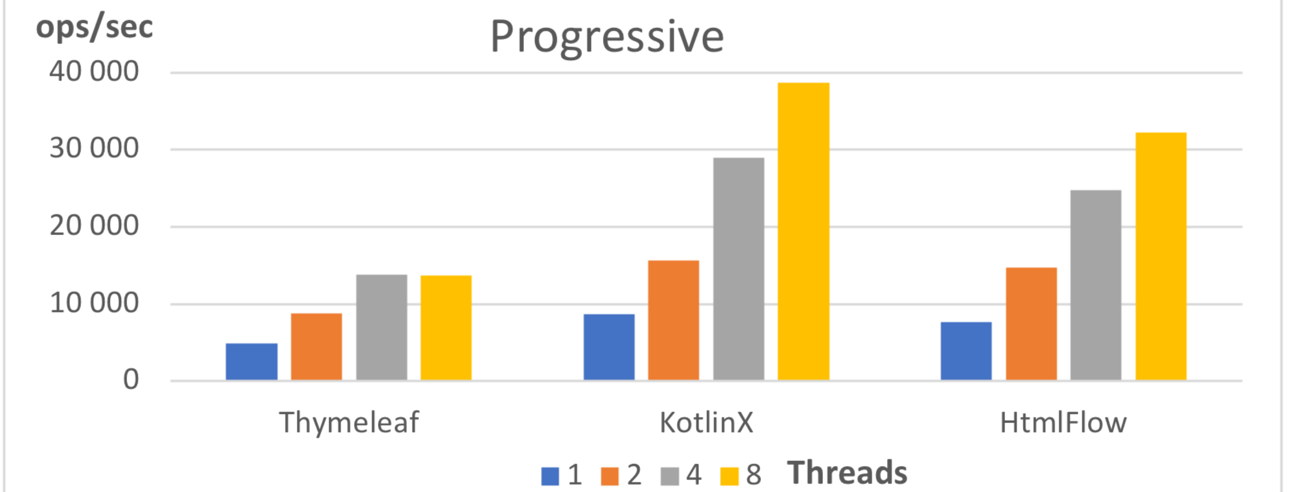
- JMH - Java Microbenchmark Harness
- Replaced Spring MVC by Spring WebFlux



Progressive – requests ops/sec

```java
public interface AwaitConsumer<T extends Element, U> {
    void accept(T parent, U model, OnCompletion cb);
}
```

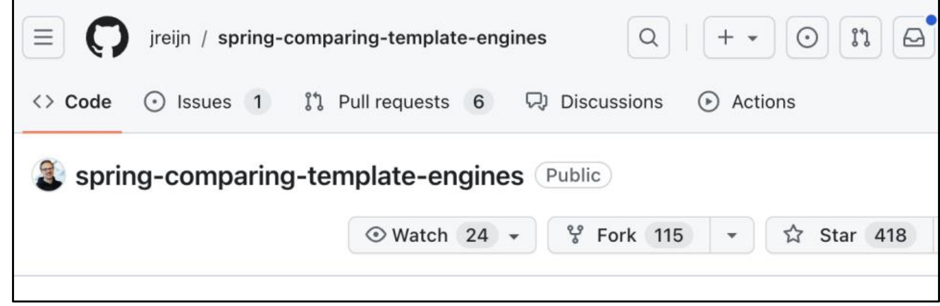
Overhead !



Turn in Continuations – Async - Evaluation

Spring Comparing Template Engines

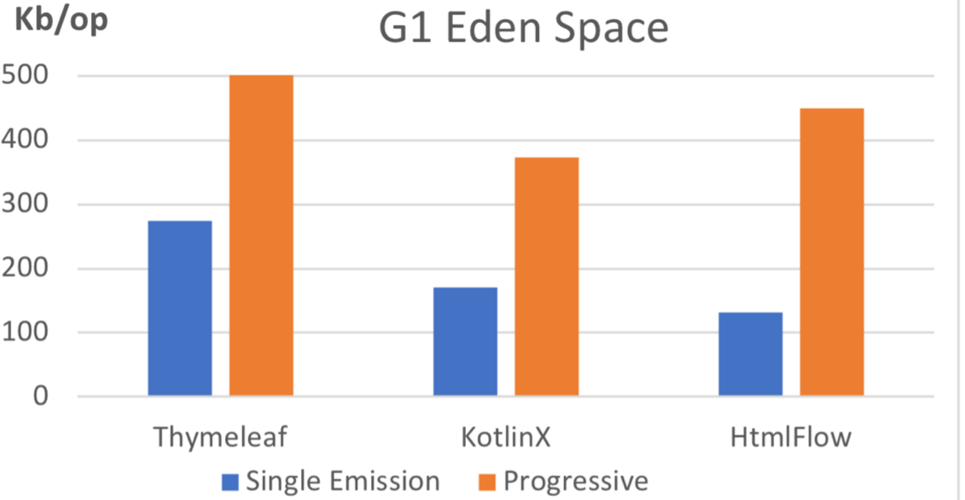
- JMH - Java Microbenchmark Harness
- Replaced Spring MVC by Spring WebFlux



Memory pressure

```java
public interface AwaitConsumer<T extends Element, U> {
    void accept(T parent, U model, OnCompletion cb);
}
```

Overhead !



```kotlin
val view = HtmlFlow.view { page -> page
  .html()
    .body()
      .h3().dynamic<Artist> {
        h3, m -> h3.text(m.artistName)
      }
      .__() // h3
      .h3().text("MusicBrainz info:").__()
      .ul().dynamic<Artist> { ul, m -> m.musicBrainz
        ...
      }
      .__() // ul
      .b().text("Spotify popular tracks:").__()
      .div().dynamic<Artist> { div, m -> m.spotify
        ...
      }
      .__() // div
    .__() // body
  .__() // html
}
```

```
model -> { visitor.write("<html><body><h3>"); next() }
model -> { consumer.accept(element, model); next() }
model -> { visitor.write("</h3><h3>Music…"); next() }
model -> { consumer.accept(element, model); next() }
model -> { visitor.write("</ul><b>Spotify…"); next() }
model -> { consumer.accept(element, model); next() }
model -> { visitor.write("</div></body></html>"); next() }
```