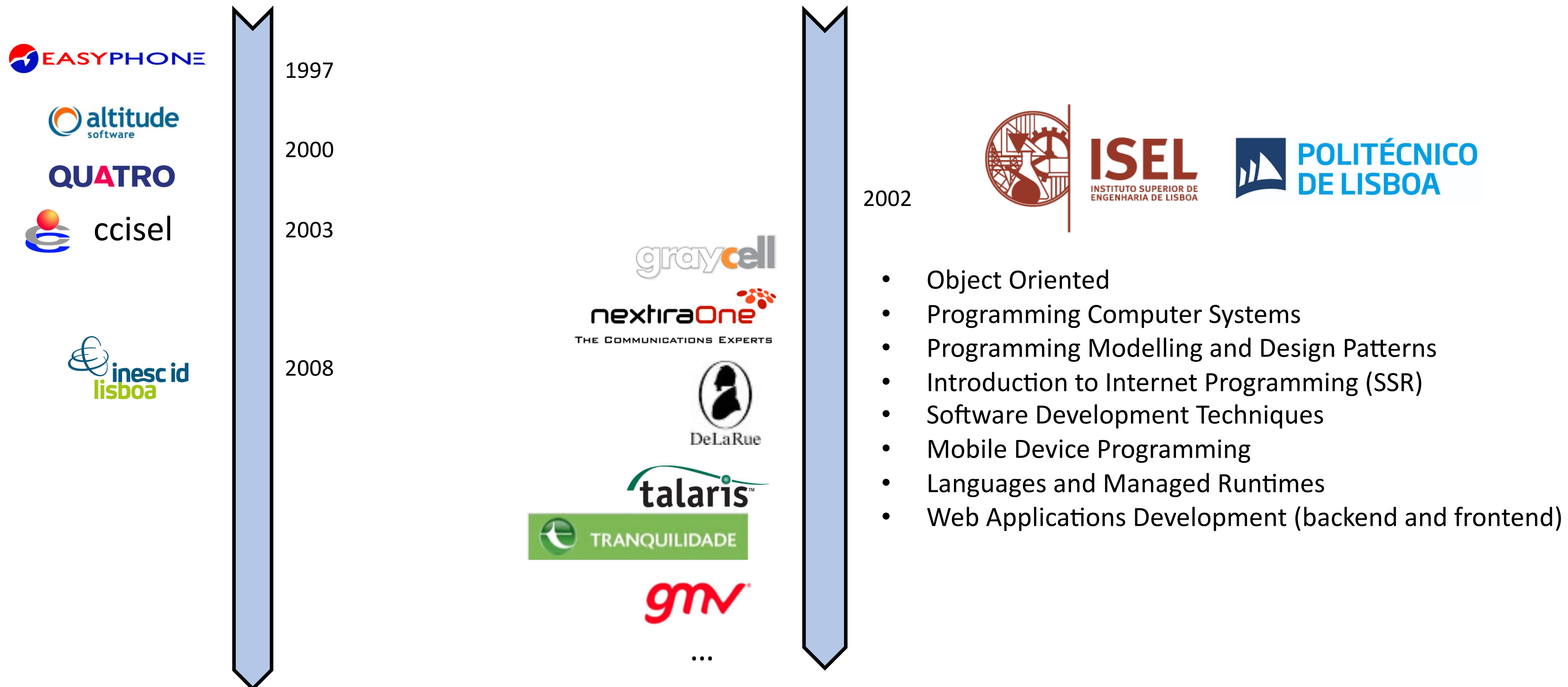
WISE 2024

Progressive Server-Side Rendering with Suspendable Web Templates

Fernando Miguel Carvalho (ISEL)

December 2024

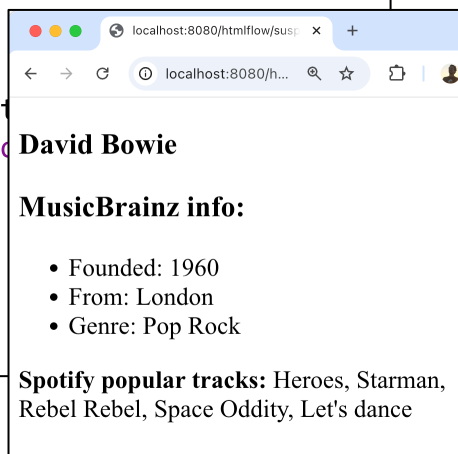
Software Engineering



- Object Oriented
- Programming Computer Systems
- Programming Modelling and Design Patterns
- Introduction to Internet Programming (SSR)
- Software Development Techniques
- Mobile Device Programming
- Languages and Managed Runtimes
- Web Applications Development (backend and frontend)

2019 HtmlFlow – Java DSL to write typesafe HTML

```
val artistView: HtmlView<Artist> = view<Artist> {
  html {
    body {
      h3 {
        dyn { m: Artist ->  text(m.name) }
      }
      h3 { +"MusicBrainz info:" }
      ul {
        dyn { m: Artist -> with(m.musicBrainz) {
            li { +"Founded: $year" }
            li { +"From: $from" }
            li { +"Genre: $genres" }
        }} // dyn
      } // ul
      p {
        b { +"Spotify popular
        dyn { m: Artist -> wi
            text(join(", ", po
        }} // dyn
      } // p
    } // body
  } // html
}
```



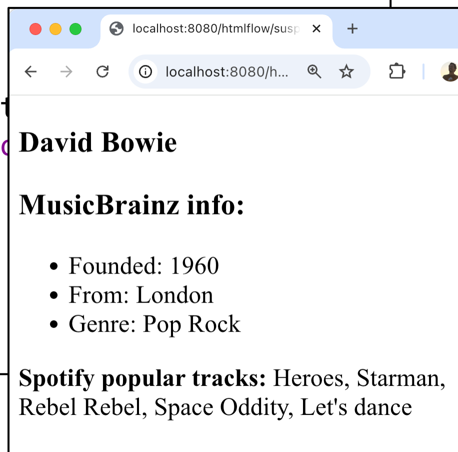
2002

POLITÉCNICO DE LISBOA

- Object Oriented
- Programming Computer Systems
- Programming Modelling and Design Patterns
- Introduction to Internet Programming (SSR)
- Software Development Techniques
- Mobile Device Programming
- Languages and Managed Runtimes
- Web Applications Development (backend and frontend)

2019 HtmlFlow – Java DSL to write typesafe HTML

```
val artistView: HtmlView<Artist> = view<Artist> {
  html {
    body {
      h3 {
        dyn { m: Artist ->  text(m.name) }
      }
      h3 { +"MusicBrainz info:" }
      ul {
        dyn { m: Artist -> with(m.musicBrainz) {
            li { +"Founded: $year" }
            li { +"From: $from" }
            li { +"Genre: $genres" }
        }} // dyn
      } // ul
      p {
        b { +"Spotify popular
        dyn { m: Artist -> wi
            text(join(", ", po
        }} // dyn
      } // p
    } // body
  } // html
}
```



Dangerous Techniques Limited

kicktipp

HtmlFlow – Java DSL to write typesafe HTML

- 2018 – MsC “*Domain Specific Language generation based on an XML Schema*”, Luis Duarte
- 2019 WebIst ”*HoT: Unleash Web Views with Higher-order Templates*”, Luís Duarte and F.M. Carvalho
- 2020 LNBIP, volume 399, “*Text Web Templates Considered Harmful”*, F.M. Carvalho, Luis Duarte and Julien Gouesse
- 2020 IDICA IPL “*Reactive Web streams for Big Data*”, Luís Albuquerque and F.M. Carvalho
- 2022 – MsC “*Reactive Web Templates*”, Pedro Fialho
- 2023 – MsC “*Automatic DSL extension for idiomatic Kotlin*”, André Vizinha
- 2023 WebIst “*Enhancing SSR in Low-Thread Web Servers: A Comprehensive Approach for Progressive Server-Side Rendering with Any Asynchronous API and Multiple Data Models*”, Pedro Fialho and F.M. Carvalho
- 2024 WISE “Progressive Server-Side Rendering with Suspendable Web Templates”, F.M. Carvalho

Progressive Server-Side Rendering with Suspendable Web Templates

Progressive Rendering

localhost:8080/htmlflow/susp

localhost:8080/h...

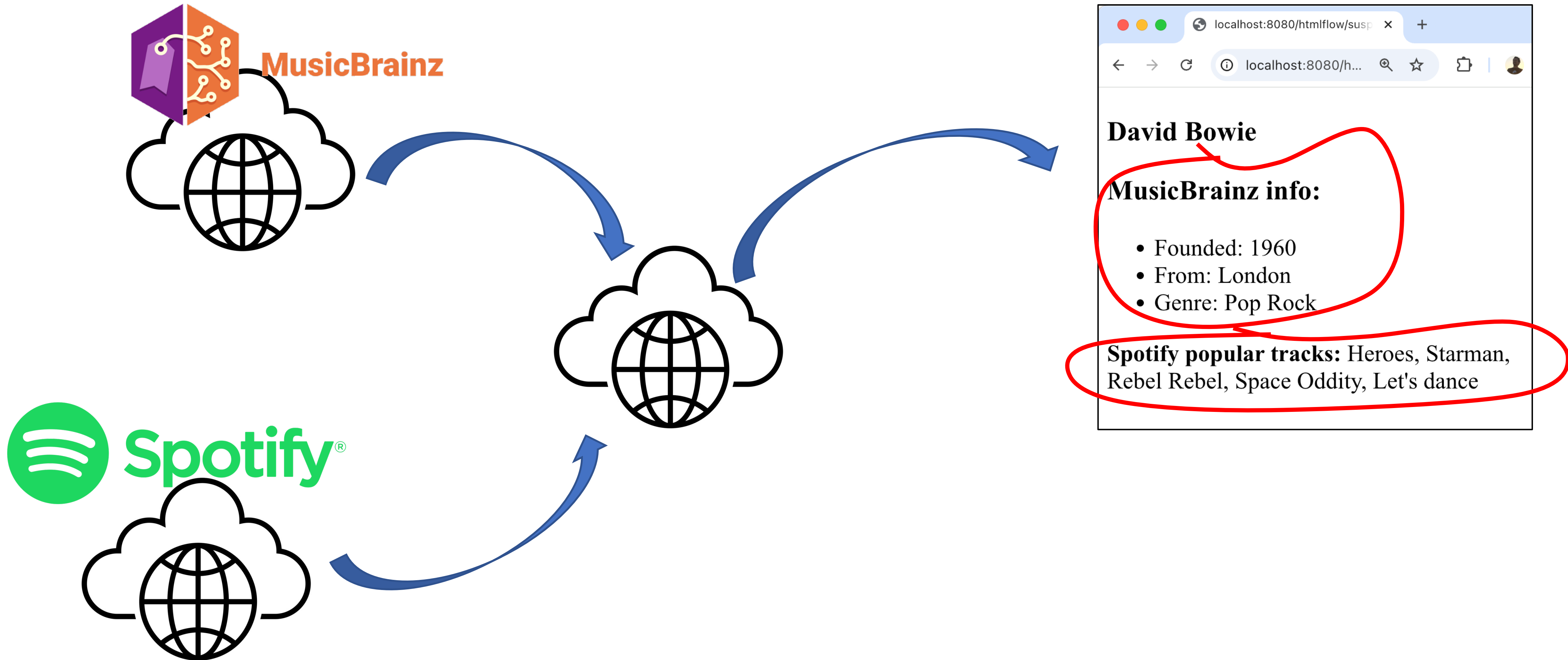
David Bowie

MusicBrainz info:

- Founded: 1960
- From: London
- Genre: Pop Rock

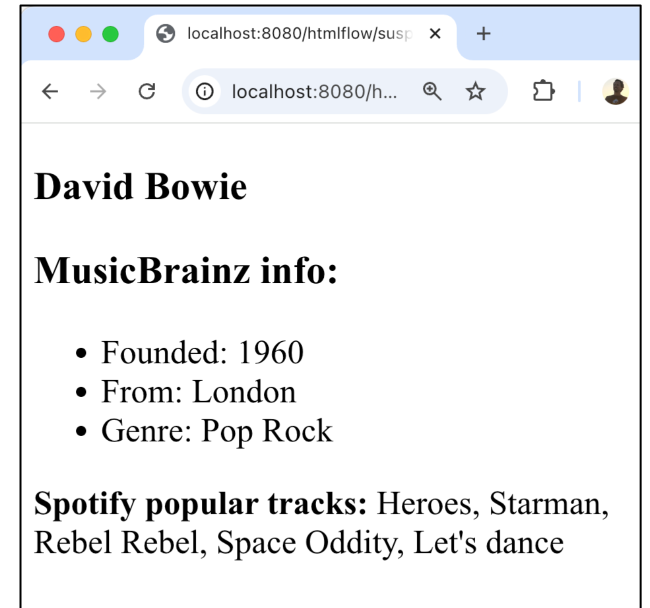
Spotify popular tracks: Heroes, Starman, Rebel Rebel, Space Oddity, Let's dance

Progressive Rendering



Progressive Rendering

```
<!DOCTYPE html>
<html>
    <body>
        <h3>David Bowie0</h3>
        <h3>MusicBrainz info:</h3>
        <ul>
            <li>Founded: 1960</li>
            <li>From: London</li>
            <li>Genre: Pop Rock</li>
        </ul>
        <p>
            <b>Spotify popular tracks:</b>
            Heroes, Starman, Rebel Rebel, …
        </p>
    </body>
</html>
```

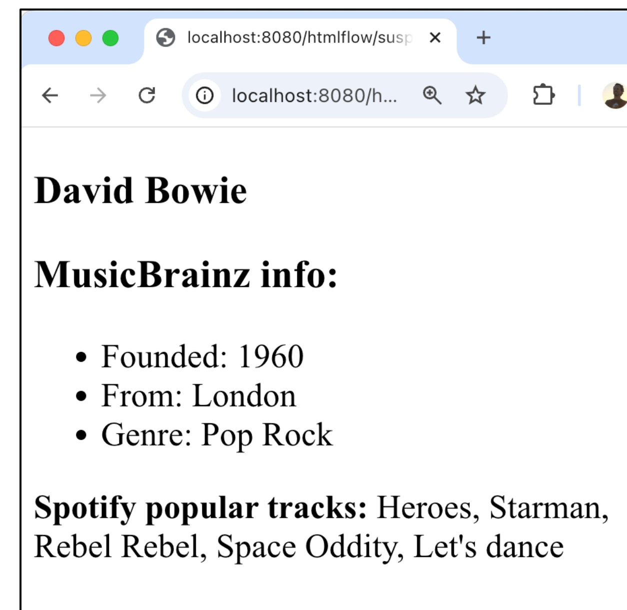


Model

```kotlin
class Artist(
    val name: String,
    val musicBrainz: MusicBrainzInfo,
    val spotify: SpotifyArtistInfo,
)
```

View

```kotlin
val artistView: HtmlView<Artist> = view<Artist> {
  html {
    body {
      h3 {
        dyn { m: Artist ->  text(m.name) }
      }
      h3 { +"MusicBrainz info:" }
      ul {
        dyn { m: Artist -> with(m.musicBrainz) {
            li { +"Founded: $year" }
            li { +"From: $from" }
            li { +"Genre: $genres" }
        }} // dyn
      } // ul
      p {
        b { +"Spotify popular tracks:" }
        dyn { m: Artist -> with(m.spotify) {
            text(join(", ", popularSongs))
        }} // dyn
      } // p
    } // body
  } // html
}
```



Model

```
class Artist(
    val name: String,
    val musicBrainz: MusicBrainzInfo,
    val spotify: SpotifyArtistInfo,
)
```

View

```
val artistView: HtmlView<Artist> = view<Artist> {
  html {
    body {
      h3 {
        dyn { m: Artist ->  text(m.name) }
      }
      h3 { +"MusicBrainz info:" }
      ul {
        dyn { m: Artist -> with(m.musicBrainz) {
            li { +"Founded: $year" }
            li { +"From: $from" }
            li { +"Genre: $genres" }
        }} // dyn
      } // ul
      p {
        b { +"Spotify popular tracks:" }
        dyn { m: Artist -> with(m.spotify) {
            text(join(", ", popularSongs))
        }} // dyn
      } // p
    } // body
  } // html
}
```

localhost:8080/htmlflow/susp

localhost:8080/h...

David Bowie

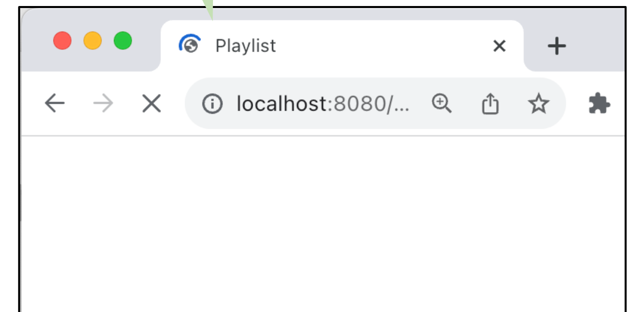
MusicBrainz info:

- Founded: 1960
- From: London
- Genre: Pop Rock

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

```
{
    "name": "David Bowie",
    "musicBrainz": {
        "Founded": 1960,
        "From": "London",
        "Genre": "Pop Rock"
    },
    "spotify": {
        "popularSongs": [ "Heroes",  "Starman", …]
    }
}
```

```
val artistView: HtmlView<Artist> = view<Artist> {
  html {
    body {
      h3 {
        dyn { m: Artist ->  text(m.name) }
      }
      h3 { +"MusicBrainz info:" }
      ul {
        dyn { m: Artist -> with(m.musicBrainz) {
            li { +"Founded: $year" }
            li { +"From: $from" }
            li { +"Genre: $genres" }
        }} // dyn
      } // ul
      p {
        b { +"Spotify popular tracks:" }
        dyn { m: Artist -> with(m.spotify) {
            text(join(", ", popularSongs))
        }} // dyn
      } // p
    } // body
  } // html
}
```



Model

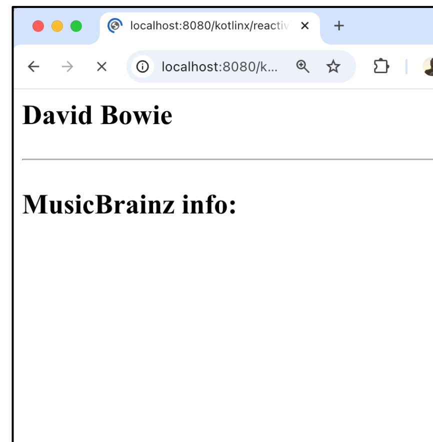
```kotlin
class Artist(
  val name: String,
  val musicBrainz: CompletableFuture<MusicBrainz>,
  val spotify: CompletableFuture<SpotifyArtist>,
)
```

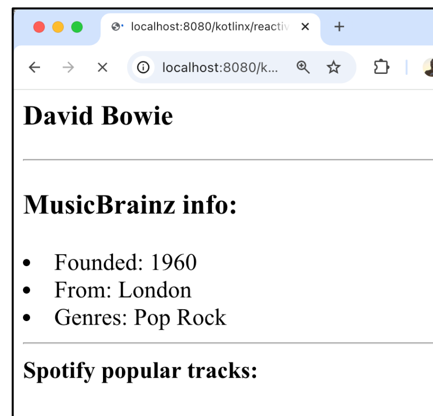
Use case example

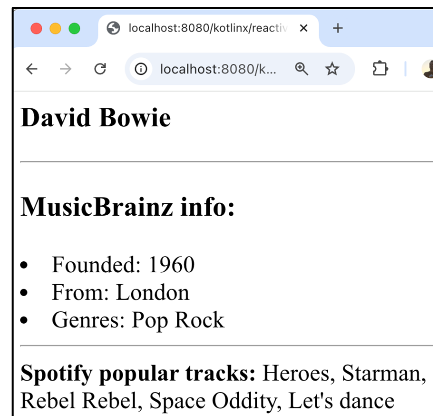
```kotlin
artistView
    .setOut(appendableResp)
    .write(davidBowieInfo)
```

View

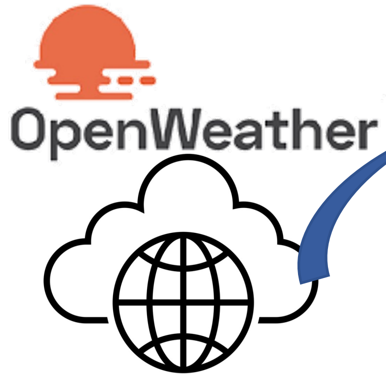
```kotlin
val artistView: HtmlView<Artist> = view<Artist> {
  html {
    body {
      h3 {
        dyn { m: Artist ->  text(m.name) }
      }
      h3 { +"MusicBrainz info:" }
      ul {
        dyn { m: Artist -> m.musicBrainz.thenAccept {
          li { +"Founded: $it.year" }
          li { +"From: $it.from" }
          li { +"Genre: $it.genres" }
        }} // dyn
      } // ul
      p {
        b { +"Spotify popular tracks:" }
        dyn { m: Artist -> m.spotify.thenAccept {
          text(join(", ", it.popularSongs))
        }} // dyn
      } // p
    } // body
  } // html
}
```







Progressive Rendering



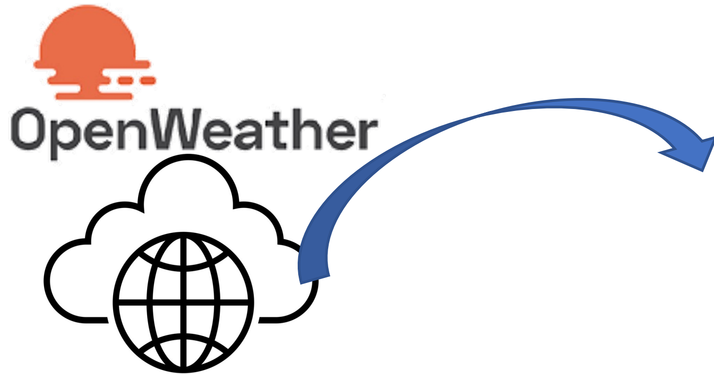

```html
<html>
  <head><title>Australia</title></head>
  <body >
    <table border="1">
      <tr>
        <th>City</th>
        <th>Celsius </th>
      </tr>
      <tr>
        <td>Adelaide </td><td>9</td>
      </tr>
      <tr>
        <td>Darwin</td><td>31</td>
      </tr>
      <tr>
        <td>Perth</td><td>16</td>
      </tr>
    </table>
  </body>
</html>
```

Australia

localhost:8080/htmlflow/

City	Celsius
Adelaide	9
Darwin	31
Perth	16

Progressive Rendering




```
body {
  table { attrBorder(_1)
    tr {
      th { text("City") }
      th { text("Celsius") }
    }
    dyn { m: Weather ->
      m.cities.forEach { it ->
        tr {
          td { text(it.city) }
          td { text(it.celsius) }
        } // tr
      } // forEach
    } // dyn
  } // table
} // body
```

```
Weather("Australia", listOf(
  Location("Adelaide", "Light rain", 9),
  Location("Darwin", "Sunny day", 31),
  Location("Perth", "Sunny day", 16),
))
```

Australia
localhost:8080/htmlflow/

City	Celsius

Australia
localhost:8080/htmlflow/

City	Celsius
Adelaide	9

Australia
localhost:8080/htmlflow/

City	Celsius
Adelaide	9
Darwin	31

Australia
localhost:8080/htmlflow/

City	Celsius
Adelaide	9
Darwin	31
Perth	16

Progressive Rendering

```
body {
  table { attrBorder(_1)
    tr {
      th { text("City") }
      th { text("Celsius") }
    }
    dyn { m: Weather ->
      m.cities.forEach { it ->
        tr {
          td { text(it.city) }
          td { text(it.celsius) }
        } // tr
      } // forEach
    } // dyn
  } // table
} // body
```

```
Weather("Australia", listOf(
  Location("Adelaide", "Light rain", 9),
  Location("Darwin", "Sunny day", 31),
  Location("Perth", "Sunny day", 16),
))
```

Australia

localhost:8080/htmlflow/

City	Celsius
Adelaide	9
Darwin	31
Perth	16

Progressive Rendering Alternatives

- CSR JavaScript Frontend frameworks:
  - React
  - Angular
  - Vue
  - ...

Progressive Rendering Alternatives

- CSR JavaScript Frontend frameworks:
  - React
  - Angular
  - Vue
  - ...
- CSR + SSR:
  - 2014 – Marko web framework (*component-based level*)
  - 2020 – Hotwired Turbo framework (*based on web sockets*)
  - 2022 – ICWE - Progressively streamed web pages, Vogel

Progressive Rendering Alternatives

- CSR JavaScript Frontend frameworks:
  - React
  - Angular
  - Vue
  - ...
- CSR + SSR:
  - 2014 – Marko web framework (*component-based level*)
  - 2020 – Hotwired Turbo framework (*based on web sockets*)
  - 2022 – ICWE - Progressively streamed web pages, Vogel

All these alternatives depend on auxiliary JavaScript

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.

Progressive Server-Side Rendering (NO JavaScript)

SSR Web Template Engines on JVM supporting PSSR:

- ✓ Rocker
- ✓ JStachio
- ✓ Pebble
- ✓ Freemarker
- ✓ Trimou
- ✓ Velocity
- ✓ Thymeleaf
- ✓ KoltinX.HTML
- ✓ HtmlFlow

Asynchronous Data Models?

SSR Web Template Engines on JVM supporting PSSR:

- Rocker
- JStachio
- Pebble
- Freemarker
- Trimou
- Velocity
- Thymeleaf
- HtmlFlow
- KoltinX.HTML

Australia
localhost:8080/htmlflow/

City	Celsius

Australia
localhost:8080/htmlflow/

City	Celsius
Adelaide	9

Australia
localhost:8080/htmlflow/

City	Celsius
Adelaide	9
Darwin	31

Australia
localhost:8080/htmlflow/

City	Celsius
Adelaide	9
Darwin	31
Perth	16

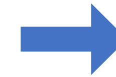
Synchronous binding

Model

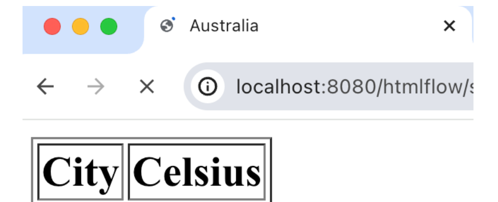
```
Weather("Australia", listOf(
  Location("Adelaide", "Light rain", 9),
  Location("Darwin", "Sunny day", 31),
  Location("Perth", "Sunny day", 16),
))
```

View

```
body {
  table { attrBorder(_1)
    tr {
      th { text("City") }
      th { text("Celsius") }
    }
    dyn { m: Weather ->
      m.cities.forEach {
        tr {
          td { text(it.city) }
          td { text(it.celsius) }
        } // tr
      } // forEach
    } // dyn
  } // table
} // body
```



```
<html>
  <head>
    <title>Australia</title>
  </head>
  <body >
    <table>
      <tr>
        <th>City</th>
        <th>Celsius </th>
      </tr>
```

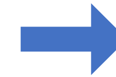


Synchronous binding

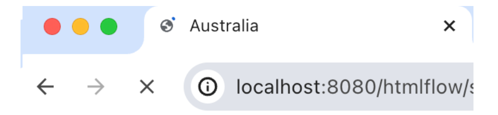
```
Weather("Australia", listOf(
  Location("Adelaide", "Light rain", 9),
  Location("Darwin", "Sunny day", 31),
  Location("Perth", "Sunny day", 16),
))
```

View

```
body {
  table { attrBorder(_1)
    tr {
      th { text("City") }
      th { text("Celsius") }
    }
    dyn { m: Weather ->
      m.cities.forEach {
        tr {
          td { text(it.city) }
          td { text(it.celsius) }
        } // tr
      } // forEach
    } // dyn
  } // table
} // body
```



```
<html>
  <head>
    <title>Australia</title>
  </head>
  <body >
    <table>
      <tr>
        <th>City</th>
        <th>Celsius </th>
      </tr>
      <tr>
        <td>Adelaide</td><td>9</td
      </tr>
      <tr>
        <td>Darwin</td><td>31</td>
      </tr>
      <tr>
        <td>Perth</td><td>16</td>
      </tr>
```



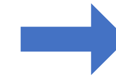
City	Celsius
Adelaide	9
Darwin	31
Perth	16

Synchronous binding

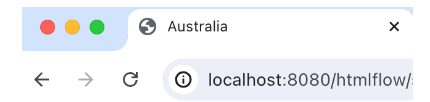
```
Weather("Australia", listOf(
  Location("Adelaide", "Light rain", 9),
  Location("Darwin", "Sunny day", 31),
  Location("Perth", "Sunny day", 16),
))
```

View

```
body {
  table { attrBorder(_1)
    tr {
      th { text("City") }
      th { text("Celsius") }
    }
    dyn { m: Weather ->
      m.cities.forEach {
        tr {
          td { text(it.city) }
          td { text(it.celsius) }
        } // tr
      } // forEach
    } // dyn
  } // table
} // body
```



```
<html>
  <head>
    <title>Australia</title>
  </head>
  <body >
    <table>
      <tr>
        <th>City</th>
        <th>Celsius </th>
      </tr>
      <tr>
        <td>Adelaide</td><td>9</td
      </tr>
      <tr>
       <td>Darwin</td><td>31</td>
      </tr>
      <tr>
        <td>Perth</td><td>16</td>
      </tr>
    </table>
  </body>
</html>
```



City	Celsius
Adelaide	9
Darwin	31
Perth	16

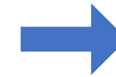
Asynchronous binding

Model

```
WeatherRx("Australia", Observable.from(
  Location("Adelaide", "Light rain", 9),
  Location("Darwin", "Sunny day", 31),
  Location("Perth", "Sunny day", 16)
).concatMap {
  just(it).delay(1000, MILLISECONDS)
}
```

View

```
body {
  table { attrBorder(_1)
    tr {
      th { text("City") }
      th { text("Celsius") }
    }
    dyn { m: Weather ->
      m.cities.forEach {
        tr {
          td { text(it.city) }
          td { text(it.celsius) }
        } // tr
      } // forEach
    } // dyn
  } // table
} // body
```



```
<html>
  <head><title>Australia</title></head>
  <body >
    <table border="1"> <tr>
      <th>City</th>
      <th>Celsius </th>
    </tr>
      <tr>
        <td>Adelaide </td><td>9</td>
      </tr>
      <tr>
        <td>Darwin</td><td>31</td>
      </tr>
      <tr>
        <td>Perth</td><td>16</td>
      </tr>
    </table>
  </body>
</html>
```

Australia

localhost:8080/htmlflow/

City	Celsius
Adelaide	9
Darwin	31
Perth	16

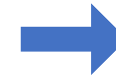
Asynchronous binding

Model

```
WeatherRx("Australia", Observable.from(
  Location("Adelaide", "Light rain", 9),
  Location("Darwin", "Sunny day", 31),
  Location("Perth", "Sunny day", 16)
).concatMap {
  just(it).delay(1000, MILLISECONDS)
}
```

View

```
body {
  table { attrBorder(_1)
    tr {
      th { text("City") }
      th { text("Celsius") }
    }
    dyn { m: WeatherRx ->
      m.cities.forEach {
        tr {
          td { text(it.city) }
          td { text(it.celsius) }
        } // tr
      } // forEach
    } // dyn
  } // table
} // body
```



Australia

localhost:8080/htmlflow/

City	Celsius
Adelaide	9
Darwin	31
Perth	16

Asynchronous binding

Model

```
WeatherRx("Australia", Observable.from(
  Location("Adelaide", "Light rain", 9),
  Location("Darwin", "Sunny day", 31),
  Location("Perth", "Sunny day", 16)
).concatMap {
  just(it).delay(1000, MILLISECONDS)
}
```

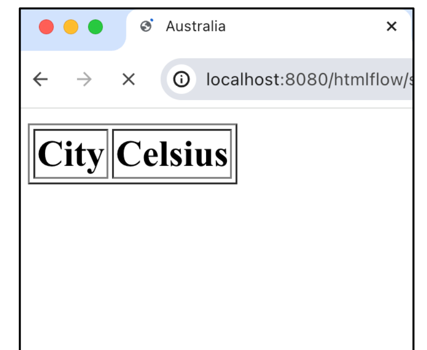
+

View

```
body {
  table { attrBorder(_1)
    tr {
      th { text("City") }
      th { text("Celsius") }
    }
    dyn { m: WeatherRx ->
      m.cities.forEach {
        tr {
          td { text(it.city) }
          td { text(it.celsius) }
        } // tr
      } // forEach
    } // dyn
  } // table
} // body
```

1st →

```
<html>
  <head>
    <title>Australia</title>
  </head>
  <body >
    <table>
      <tr>
        <th>City</th>
        <th>Celsius </th>
      </tr>
    </table>
  </body>
</html>
```



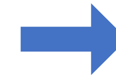
Asynchronous binding

Model

```
WeatherRx("Australia", Observable.from(
  Location("Adelaide", "Light rain", 9),
  Location("Darwin", "Sunny day", 31),
  Location("Perth", "Sunny day", 16)
).concatMap {
  just(it).delay(1000, MILLISECONDS)
}
```

View

```
body {
  table { attrBorder(_1)
    tr {
      th { text("City") }
      th { text("Celsius") }
    }
    dyn { m: WeatherRx ->
      m.cities.forEach {
        tr {
          td { text(it.city) }
          td { text(it.celsius) }
        } // tr
      } // forEach
    } // dyn
  } // table
} // body
```



2nd

```
<html>
  <head>
    <title>Australia</title>
  </head>
  <body >
    <table>
      <tr>
        <th>City</th>
        <th>Celsius </th>
      </tr>
    </table>
  </body>
</html>
<tr>
  <td>Adelaide</td><td>9</td>
</tr>
<tr>
  <td>Darwin</td><td>31</td>
</tr>
<tr>
  <td>Perth</td><td>16</td>
</tr>
```

Asynchronous binding

Model

```
WeatherRx("Australia", Observable.from(
  Location("Adelaide", "Light rain", 9),
  Location("Darwin", "Sunny day", 31),
  Location("Perth", "Sunny day", 16)
).concatMap {
  just(it).delay(1000, MILLISECONDS)
}
```

View

```
body {
  table { attrBorder(_1)
    tr {
      th { text("City") }
      th { text("Celsius") }
    }
    dyn { m: WeatherRx ->
      m.cities.forEach {
        tr {
          td { text(it.city) }
          td { text(it.celsius) }
        } // tr
      } // forEach
    } // dyn
  } // table
} // body
```

2nd

```
<html>
  <head>
    <title>Australia</title>
  </head>
  <body >
    <table>
      <tr>
        <th>City</th>
        <th>Celsius </th>
      </tr>
    </table>
  </body>
</html>
<tr>
  <td>Adelaide</td><td>9</td>
</tr>
<tr>
  <td>Darwin</td><td>31</td>
</tr>
<tr>
  <td>Perth</td><td>16</td>
</tr>
```

Thymeleaf was the first proposal to solve this problem for SSR. Yet:

1. It is limited to the the **Publisher** API (reactive streams standard)
2. Supports only a **single** Asynchronous data source

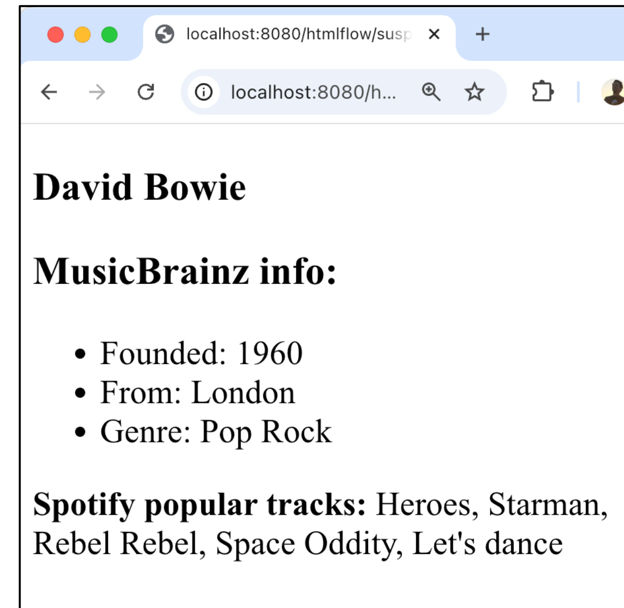
Asynchronous binding

View

```
val artistView: HtmlView<Artist> = view<Artist> {
  html {
    body {
      h3 {
        dyn { m: Artist ->  text(m.name) }
      }
      h3 { +"MusicBrainz info:" }
      ul {
        dyn { m: Artist -> m.musicBrainz.thenAccept {
            li { +"Founded: $it.year" }
            li { +"From: $it.from" }
            li { +"Genre: $it.genres" }
        }} // dyn
      } // ul
      p {
        b { +"Spotify popular tracks:" }
        dyn { m: Artist -> m.spotify.thenAccept {
            text(join(", ", it.popularSongs))
        }} // dyn
      } // p
    } // body
  } // html
}
```

Model

```
class Artist(
  val name: String,
  val musicBrainz: CompletableFuture<MusicBrainz>,
  val spotify: CompletableFuture<SpotifyArtist>,
)
```



Thymeleaf cannot deal with more than 1 Reactive Data Source

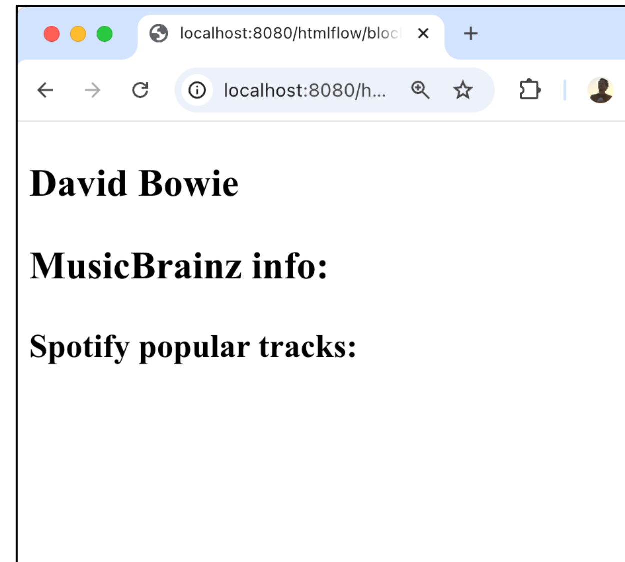
Asynchronous binding

View

```
val artistView: HtmlView<Artist> = view<Artist> {
 html {
  body {
   h3 {
    dyn { m: Artist ->  text(m.name) }
   }
   h3 { +"MusicBrainz info:" }
   ul {
    dyn { m: Artist -> m.musicBrainz.thenAccept {
        li { +"Founded: $it.year" }
        li { +"From: $it.from" }
        li { +"Genre: $it.genres" }
    }} // dyn
   } // ul
   p {
    b { +"Spotify popular tracks:" }
    dyn { m: Artist -> m.spotify.thenAccept {
        text(join(", ", it.popularSongs))
    }} // dyn
   } // p
  } // body
 } // html
}
```

Model

```
class Artist(
  val name: String,
  val musicBrainz: CompletableFuture<MusicBrainz>,
  val spotify: CompletableFuture<SpotifyArtist>,
)
```



Asynchronous binding + async/await

View

```
val artistView: HtmlView<Artist> = view<Artist> {
  html {
    body {
      h3 {
        dyn { m: Artist ->  text(m.name) }
      }
      h3 { +"MusicBrainz info:" }
      ul {
        dyn { m: Artist -> m.musicBrainz.thenAccept {
          val it = m.musicBrainz.await()
          li { +"Founded: $it.year" }
          li { +"From: $it.from" }
          li { +"Genre: $it.genres" }
        }} // dyn
      } // ul
      p {
        b { +"Spotify popular tracks:" }
        dyn { m: Artist -> m.spotify.thenAccept {
          val it = m.spotify.await()
          text(join(", ", it.popularSongs))
        }} // dyn
      } // p
    } // body
  } // html
}
```

Suspension Point

Suspension Point

Model

```
class Artist(
  val name: String,
  val musicBrainz: CompletableFuture<MusicBrainz>,
  val spotify: CompletableFuture<SpotifyArtist>,
)
```

Asynchronous binding + async/await

Model

```
class Artist(
  val name: String,
  val musicBrainz: CompletableFuture<MusicBrainz>,
  val spotify: CompletableFuture<SpotifyArtist>,
)
```

View

```
val artistView: HtmlView<Artist> = view<Artist> {
  html {
    body {
      h3 {
        dyn { m: Artist ->  text(m.name) }
      }
      h3 { +"MusicBrainz info:" }
      ul {
        suspending { m: Artist ->
          val it = m.musicBrainz.await()
          li { +"Founded: $it.year" }
          li { +"From: $it.from" }
          li { +"Genre: $it.genres" }
        }} // dyn
      } // ul
      p {
        b { +"Spotify popular tracks:" }
        suspending { m: Artist ->
          val it = m.spotify.await()
          text(join(", ", it.popularSongs))
        }} // dyn
      } // p
    } // body
  } // html
}
```

Suspension Point

Suspension Point

Kotlin suspending functions

New HtmlFlow builder:

```
public fun <E : Element, M> E.suspending(block: suspend E.(M) -> Unit): E {
    /* code */
}
```

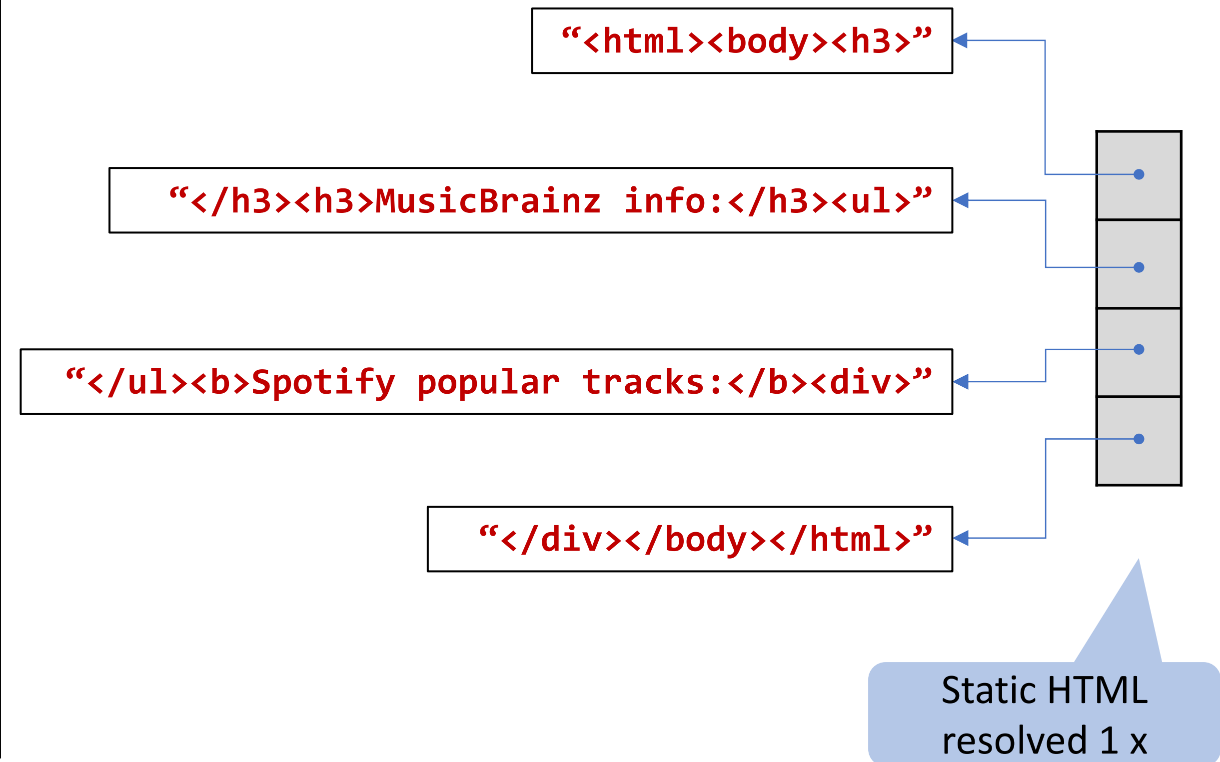
```
      ...
      suspending { m: Artist ->
          val it = m.musicBrainz.await()
          li { +"Founded: $it.year" }
          li { +"From: $it.from" }
          li { +"Genre: $it.genres" }
      }} // dyn
    } // ul
    p {
      b { +"Spotify popular tracks:" }
      suspending { m: Artist ->
          val it = m.spotify.await()
          text(join(", ", it.popularSongs))
      }} // dyn
    } // p
  } // body
} // html
}
```

HtmlFlow pre-encoding

```
val artistView: HtmlView<Artist> = view<Artist> {
  html {
    body {
      h3 {
        dyn { m: Artist ->  text(m.name) }
      }
      h3 { +"MusicBrainz info:" }
      ul {
        dyn { m: Artist -> m.musicBrainz.thenAccept {
            li { +"Founded: $it.year" }
            li { +"From: $it.from" }
            li { +"Genre: $it.genres" }
        }} // dyn
      } // ul
      p {
        b { +"Spotify popular tracks:" }
        dyn { m: Artist -> m.spotify.thenAccept {
            text(join(", ", it.popularSongs))
        }} // dyn
      } // p
    } // body
  } // html
}
```

HtmlFlow pre-encoding

```
val artistView: HtmlView<Artist> = view<Artist> {
  html {
    body {
      h3 {
        dyn { m: Artist ->  text(m.name) }
      }
      h3 { +"MusicBrainz info:" }
      ul {
        dyn { m: Artist -> m.musicBrainz.thenAccept {
            li { +"Founded: $it.year" }
            li { +"From: $it.from" }
            li { +"Genre: $it.genres" }
        }} // dyn
      } // ul
      p {
        b { +"Spotify popular tracks:" }
        dyn { m: Artist -> m.spotify.thenAccept {
            text(join(", ", it.popularSongs))
        }} // dyn
      } // p
    } // body
  } // html
}
```



Turn in Continuations

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

HtmlFlow 4.0

```
val artistView: HtmlView<Artist> = view<Artist> {
  html {
    body {
      h3 {
        dyn { m: Artist ->  text(m.name) }
      }
      h3 { +"MusicBrainz info:" }
      ul {
        dyn { m: Artist -> m.musicBrainz.thenAccept {
            li { +"Founded: $it.year" }
            li { +"From: $it.from" }
            li { +"Genre: $it.genres" }
        }} // dyn
      } // ul
      p {
        b { +"Spotify popular tracks:" }
        dyn { m: Artist -> m.spotify.thenAccept {
            text(join(", ", it.popularSongs))
        }} // dyn
      } // p
    } // 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

Turn in Continuations

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

HtmlFlow 4.0

```kotlin
val artistView: HtmlView<Artist> = view<Artist> {
  html {
    body {
      h3 {
        dyn { m: Artist ->  text(m.name) }
      }
      h3 { +"MusicBrainz info:" }
      ul {
        dyn { m: Artist -> m.musicBrainz.thenAccept {
            li { +"Founded: $it.year" }
            li { +"From: $it.from" }
            li { +"Genre: $it.genres" }
        }} // dyn
      } // ul
      p {
        b { +"Spotify popular tracks:" }
        dyn { m: Artist -> m.spotify.thenAccept {
            text(join(", ", it.popularSongs))
        }} // dyn
      } // p
    } // body
  } // html
}
```

```java
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() }
```

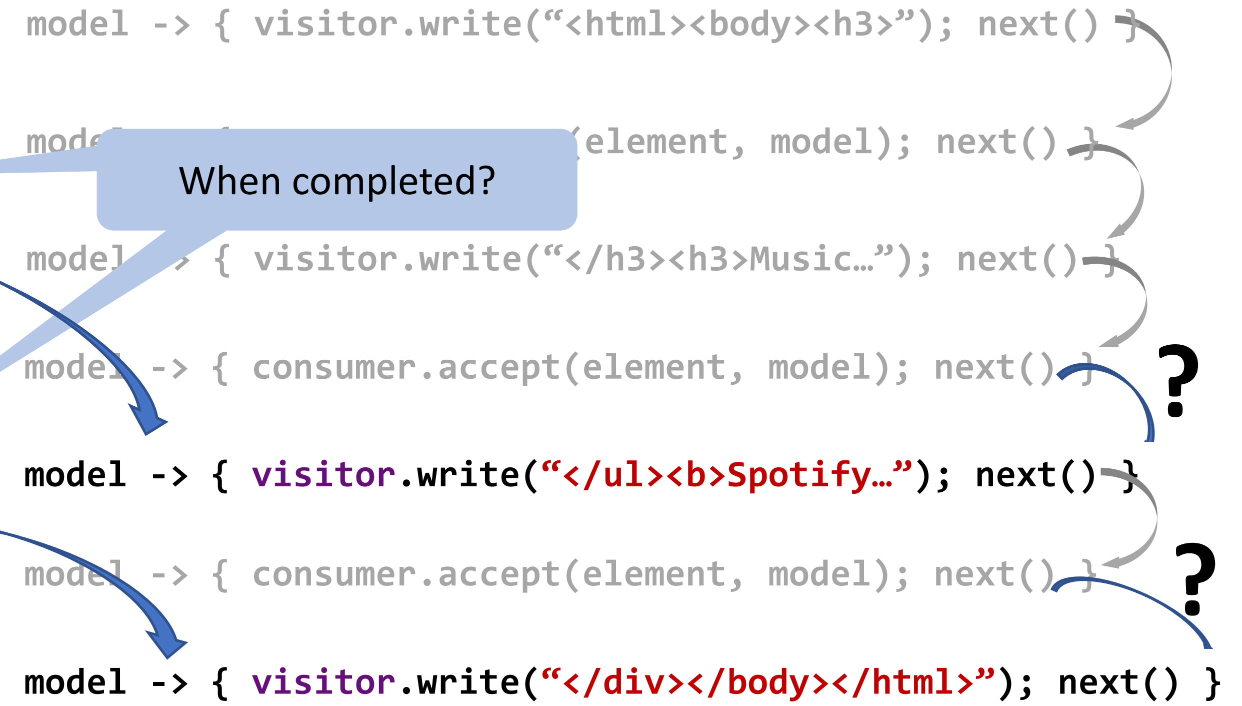
Chain-of-responsibility pattern

Turn in Continuations - Async

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

```
val artistView: HtmlView<Artist> = view<Artist> {
  html {
    body {
      h3 {
        dyn { m: Artist ->  text(m.name) }
      }
      h3 { +"MusicBrainz info:" }
      ul {
        dyn { m: Artist -> m.musicBrainz.thenAccept {
            li { +"Founded: $it.year" }
            li { +"From: $it.from" }
            li { +"Genre: $it.genres" }
        }} // dyn
      p {
        b { +"Spotify popular tracks:" }
        dyn { m: Artist -> m.spotify.thenAccept {
            text(join(", ", it.popularSongs))
        }} // dyn
      } // p
    } // body
  } // html
}
```

When to call the next continuation?



Turn in Continuations - Async

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

```
val artistView: HtmlView<Artist> = view<Artist> {
  html {
    body {
      h3 {
        dyn { m: Artist ->  text(m.name) }
      }
      h3 { +"MusicBrainz info:" }
      ul {
        dyn { m: Artist, cb -> m.musicBrainz.thenAccept {
            li { +"Founded: $it.year" }
            li { +"From: $it.from" }
            li { +"Genre: $it.genres" }
        }} // dyn
      } // ul
      p {
        b { +"Spotify popular tracks:" }
        dyn { m: Artist, cb -> m.spotify.thenAccept {
            text(join(", ", it.popularSongs))
        }} // dyn
      } // p
    } // 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() }
```

AKA Continuation-passing style (**CPS**)

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

Turn in Continuations - Async

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

```
val artistView: HtmlView<Artist> = view<Artist> {
  html {
    body {
      h3 {
        dyn { m: Artist ->  text(m.name) }
      }
      h3 { +"MusicBrainz info:" }
      ul {
        suspending { m: Artist ->
            val it = m.musicBrainz.await()
            ...
        }} // dyn
      } // ul
      p {
        b { +"Spotify popular tracks:" }
        suspending { m: Artist ->
            val it = m.spotify.await()
            text(joi(", ", it.popularSongs))
        }} // dyn
      } // p
    } // 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() }
```

Callback will be hidden by the Kotlin suspend function

Kotlin suspending functions

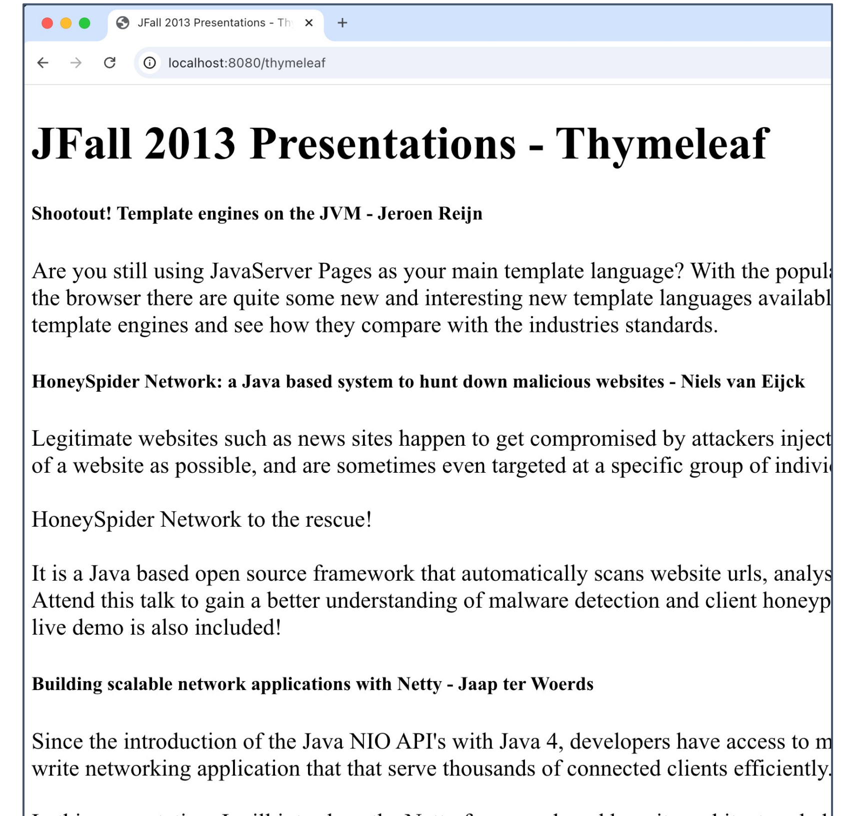
New HtmlFlow builder:

```kotlin
public fun <E : Element, M> E.suspending(block: suspend E.(M) -> Unit): E {
    /* code */
}
```

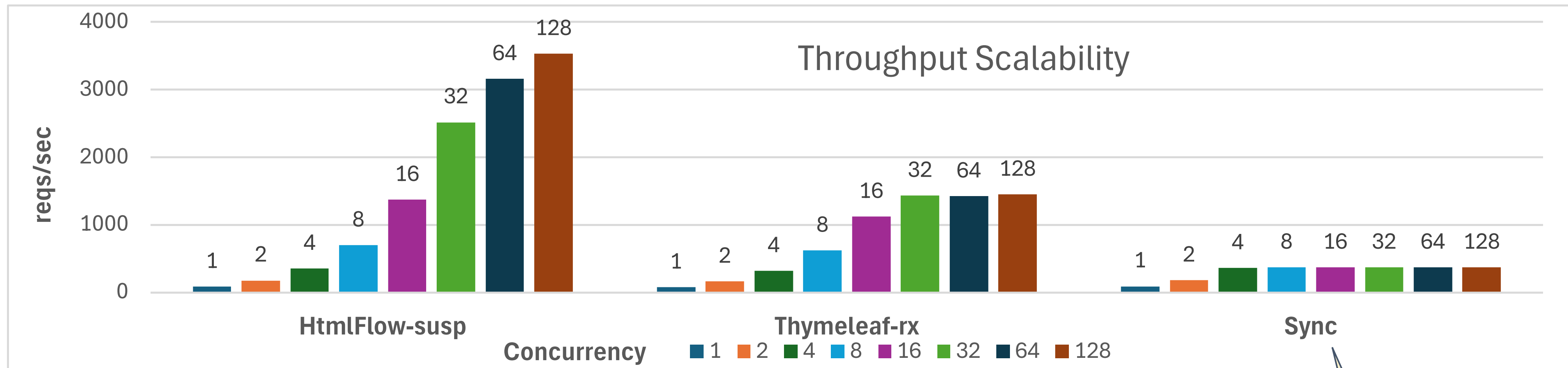
Comparing Template Engines for Spring

- Replaced Spring MVC by Spring WebFlux, the **event-driven** server version of Spring (**low-thread**).
- **Scalability** with concurrent requests ranging from 1 to 128.
- **Repository** contains 10 instances of Presentation
- **Web Template** generates an HTML document featuring a table with 10 rows
- The resulting HTML code spans approximately 80 lines and occupies a size of around 9 KB.

```java
public record Presentation(
    long id,
    String title,
    String speakerName,
    String summary
) {}
```



Comparing Template Engines for Spring



- Rocker
- JStachio
- Pebble
- Freemarker
- Trimou
- Velocity
- Thymeleaf
- KoltinX.HTML

GitHub-hosted virtual machine under GitHub Actions,
running Ubuntu 22.04 with 4 processors and 16 GB of RAM

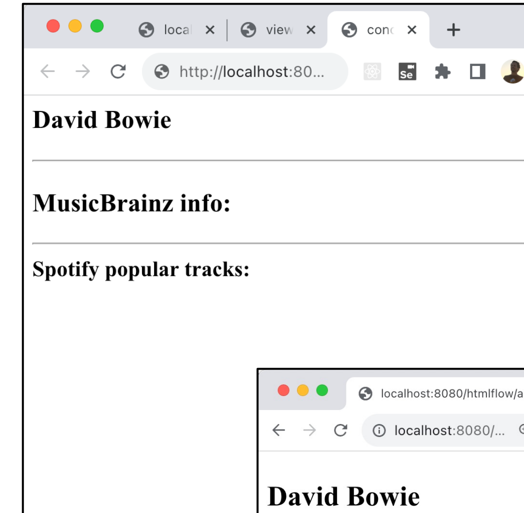
Future Directions

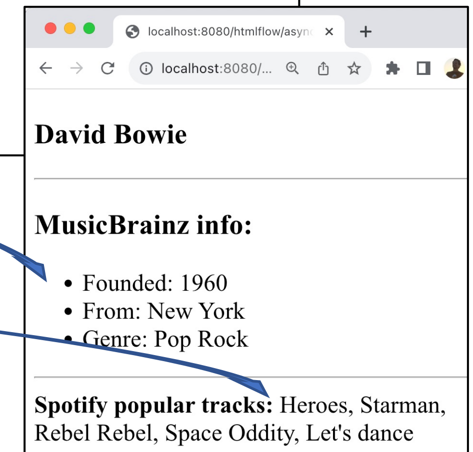
HtmlFlow Continuations => Concurrent

```
val artistView: HtmlView<Artist> = view<Artist> {
  html {
    body {
      h3 {
        dyn { m: Artist ->  text(m.name) }
      }
      h3 { +"MusicBrainz info:" }
      ul {
        suspending { m: Artist ->
            val it = m.musicBrainz.await()
            ...
        }}
      } // ul
      p {
        b { +"Spotify popular tracks:" }
        suspending { m: Artist ->
            val it = m.spotify.await()
            text(join(", ", it.popularSongs))
        }}
      } // p
    } // body
  } // html
}
```

push on completion

push on completion





Concurrent

THANK YOU

Challenges of depending on client-side JavaScript

2023 - ICWE - International Conference on Web Engineering

Vepsäläinen, J., Hellas, A., Vuorimaa, P.:

“*The rise of disappearing frameworks in web development*”. In:. pp. 319–326.

Challenges of depending on client-side JavaScript

2005 "*The Lost Art of Progressive HTML Rendering*", Jeff Atwood

*"One thing I dislike about ASP.NET is that **it renders the entire web page in memory before sending one single byte of that page to the browser."***

*"even more galling is that HTML was originally designed to **render progressively** as content is received."*

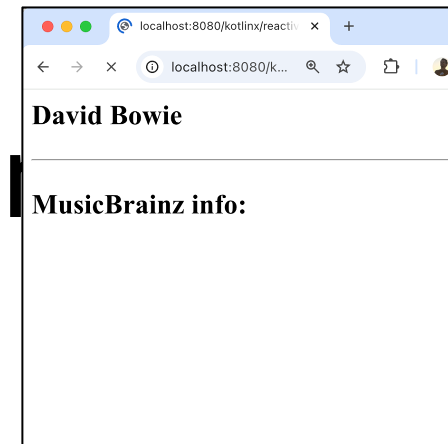
*The first version of Netscape appeared in **October 1994** under the code name "Mozilla." **Netscape 1.0's** early beta versions **introduced the "progressive rendering" of pages and images**, meaning that the page begins to appear and the text can be read even before all of the text and/or images have been completely downloaded.*

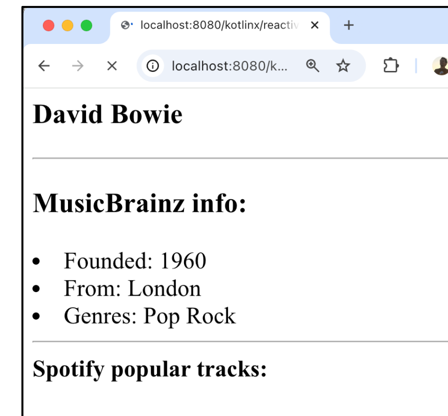
Progressive Server-Side Renderi ript)

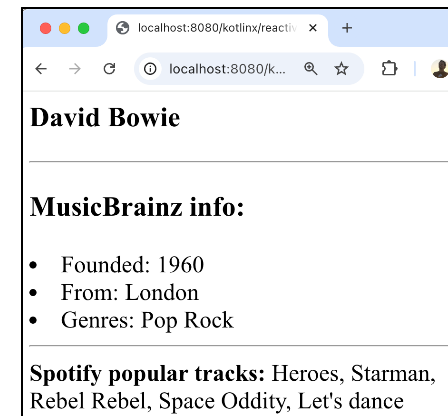
SSR Web Template Engines on JVM supporting PSSR:

- Rocker
- JStachio
- Pebble
- Freemarker
- Trimou
- Velocity
- Thymeleaf
- HtmlFlow
- KoltinX.HTML

Progressive but NOT concurrent!
=> Sequential







Progressive Server-Side Rendering (NO JavaScript)

SSR Web Template Engines on JVM supporting PSSR:

- ✓ Rocker
- ✓ JStachio
- ✓ Pebble
- ✓ Freemarker
- ✓ Trimou
- ✓ Velocity
- ✓ Thymeleaf
- ✓ KoltinX.HTML
- ✓ HtmlFlow

Progressive but NOT concurrent!