Hydration, Islands & Resumability

Michele Stieven

Angular GDE, Founder of accademia.dev

@MicheleStieven

www.accademia.dev

RxJS

Masterclass

Functional

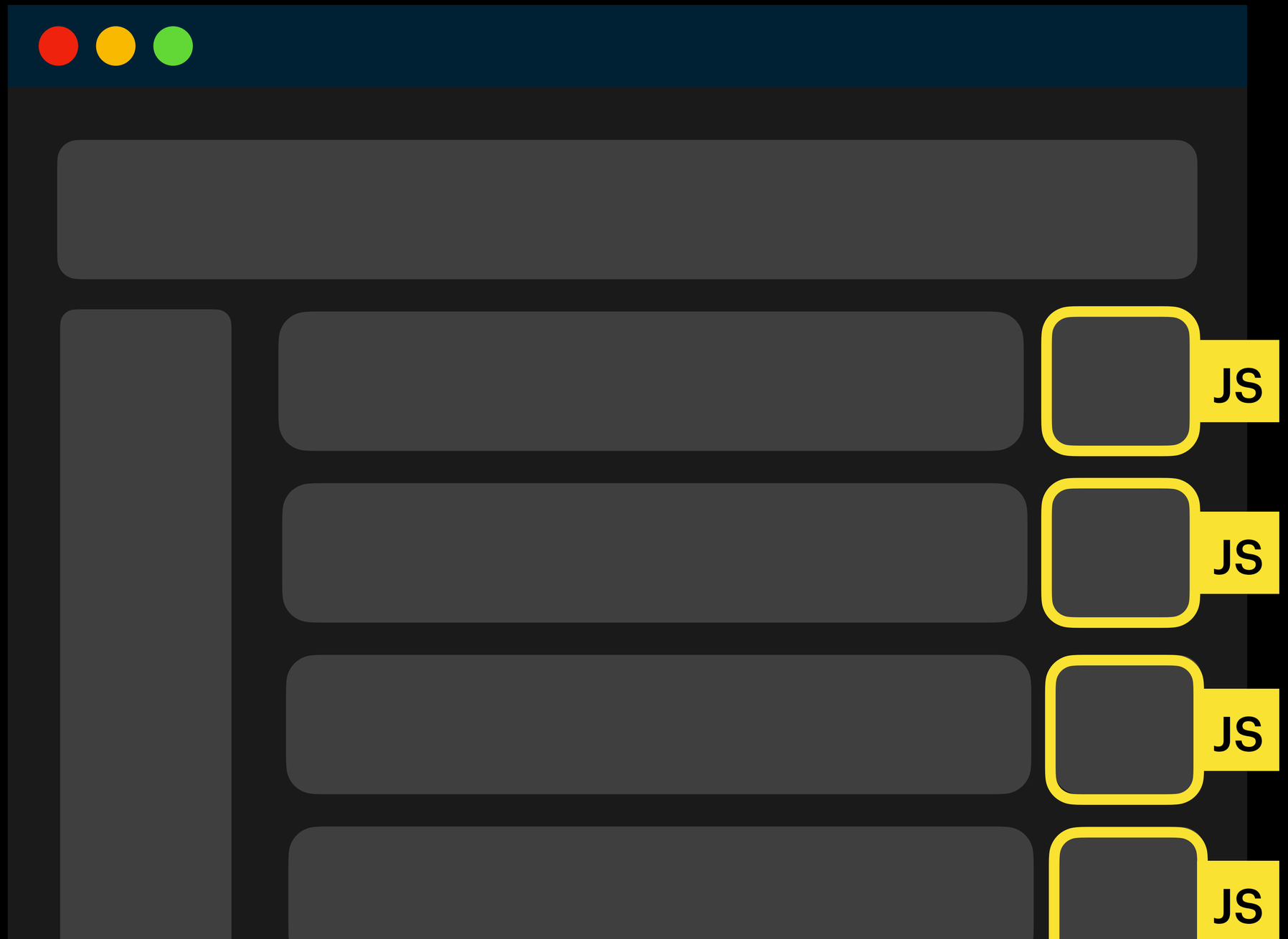
JavaScript

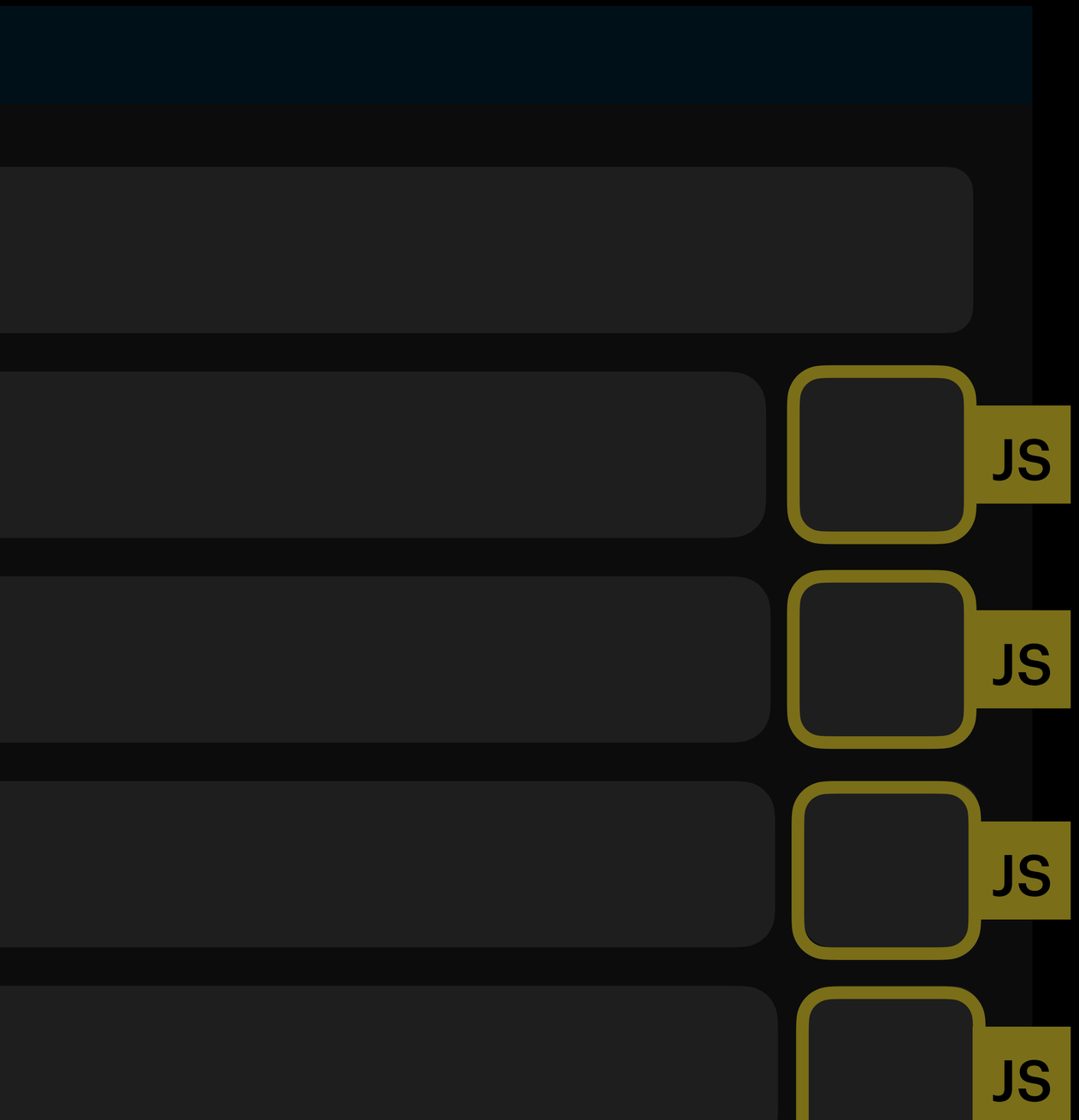
👨‍🍳 Today’s menu

- SSR
- Hydration
- Progressive Hydration
- Partial Hydration
  - Islands
  - Server Components
- Resumability

SSR

SERVER-SIDE RENDERING


JS
JS
JS
JS



PROs

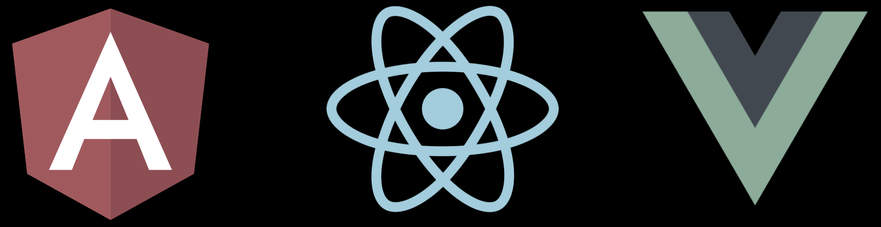
- Initial performance
- Less JavaScript
- SEO

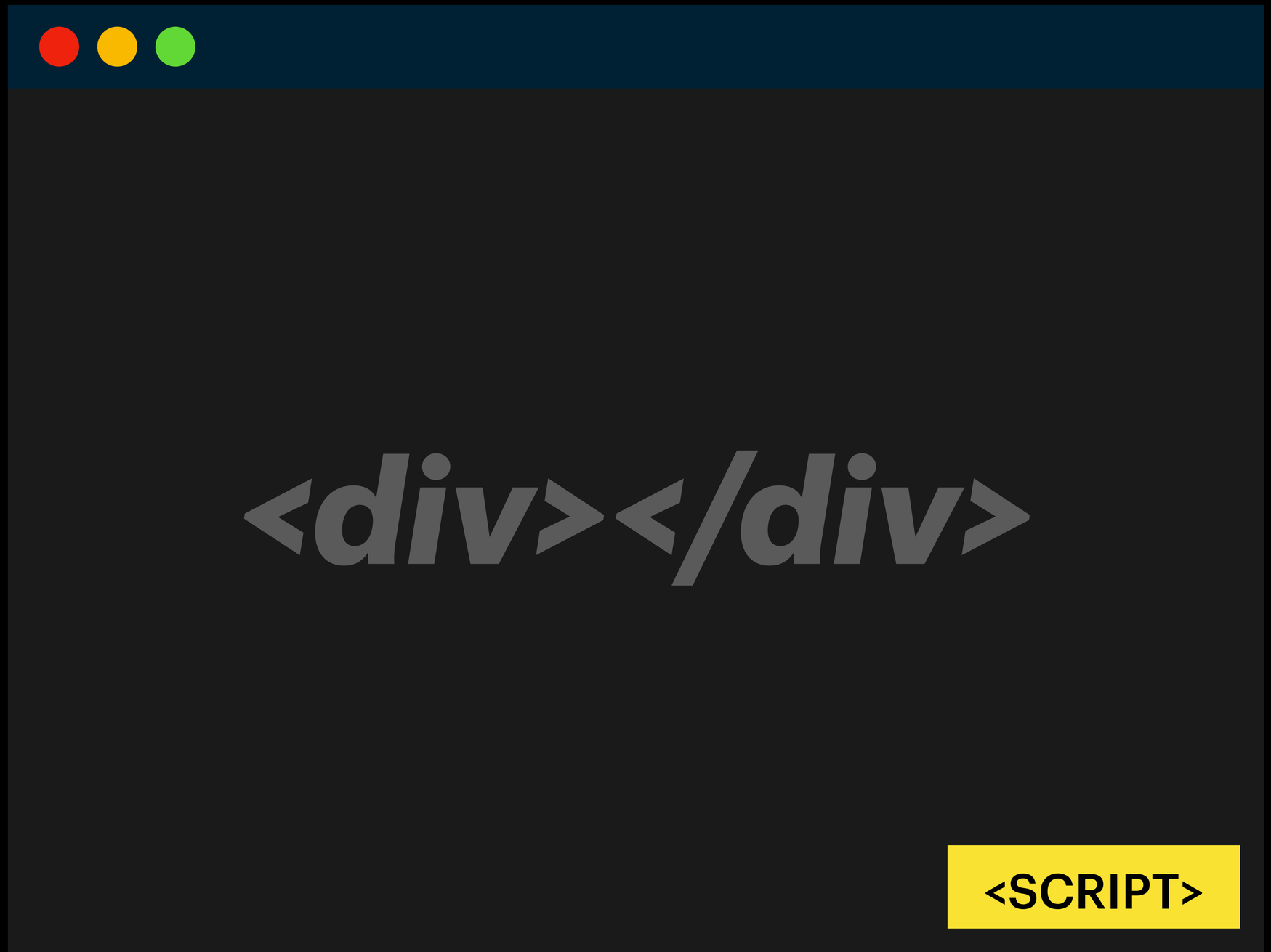
CONs

- Duplicated logic
- Imperative JS
- No client routing

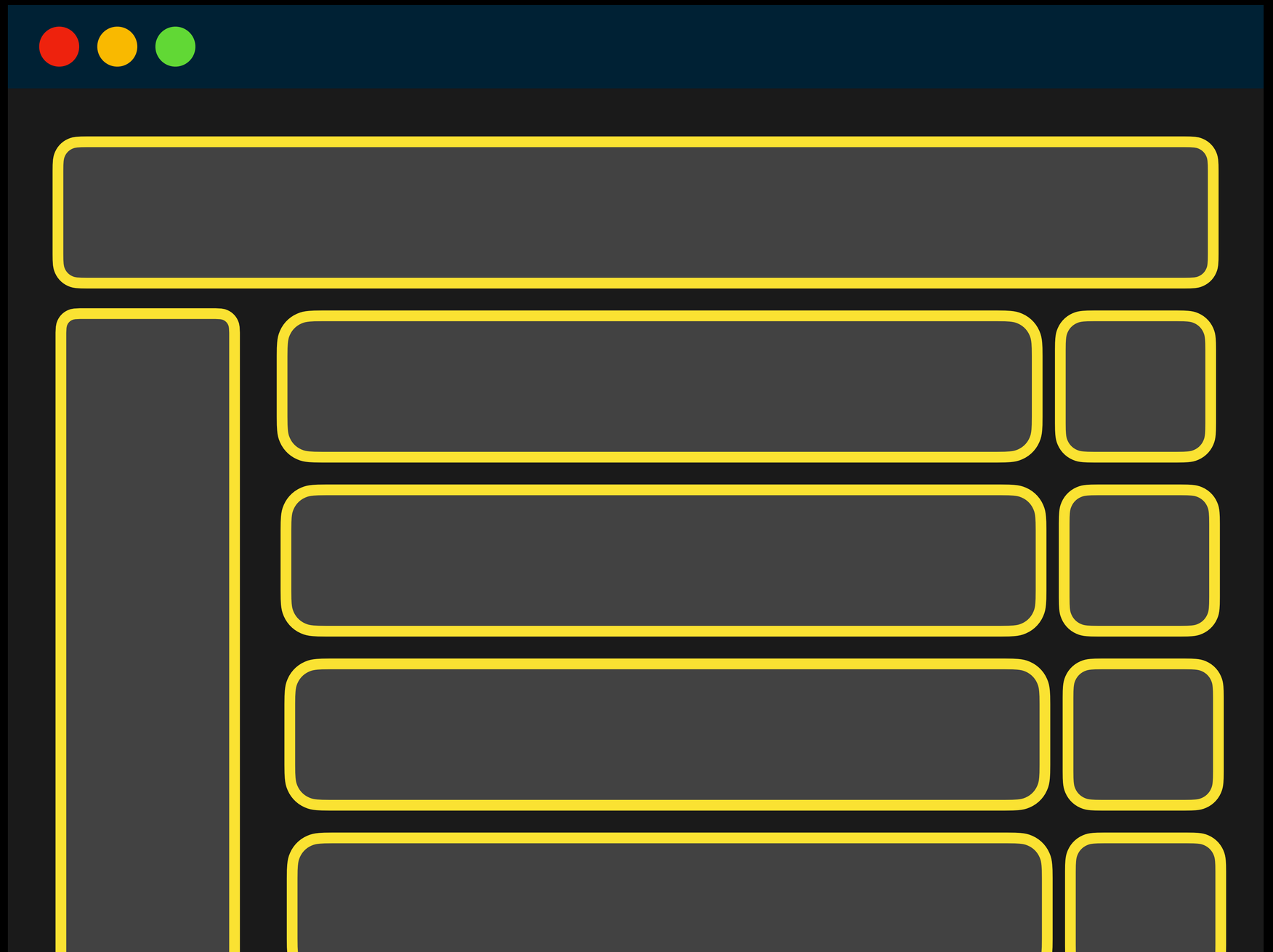
CSR

CLIENT-SIDE RENDERING





<SCRIPT>



PROs

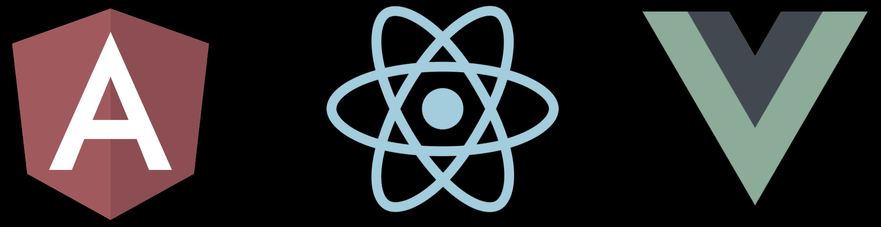
- Unified logic
- Declarative JS
- Client-side routing

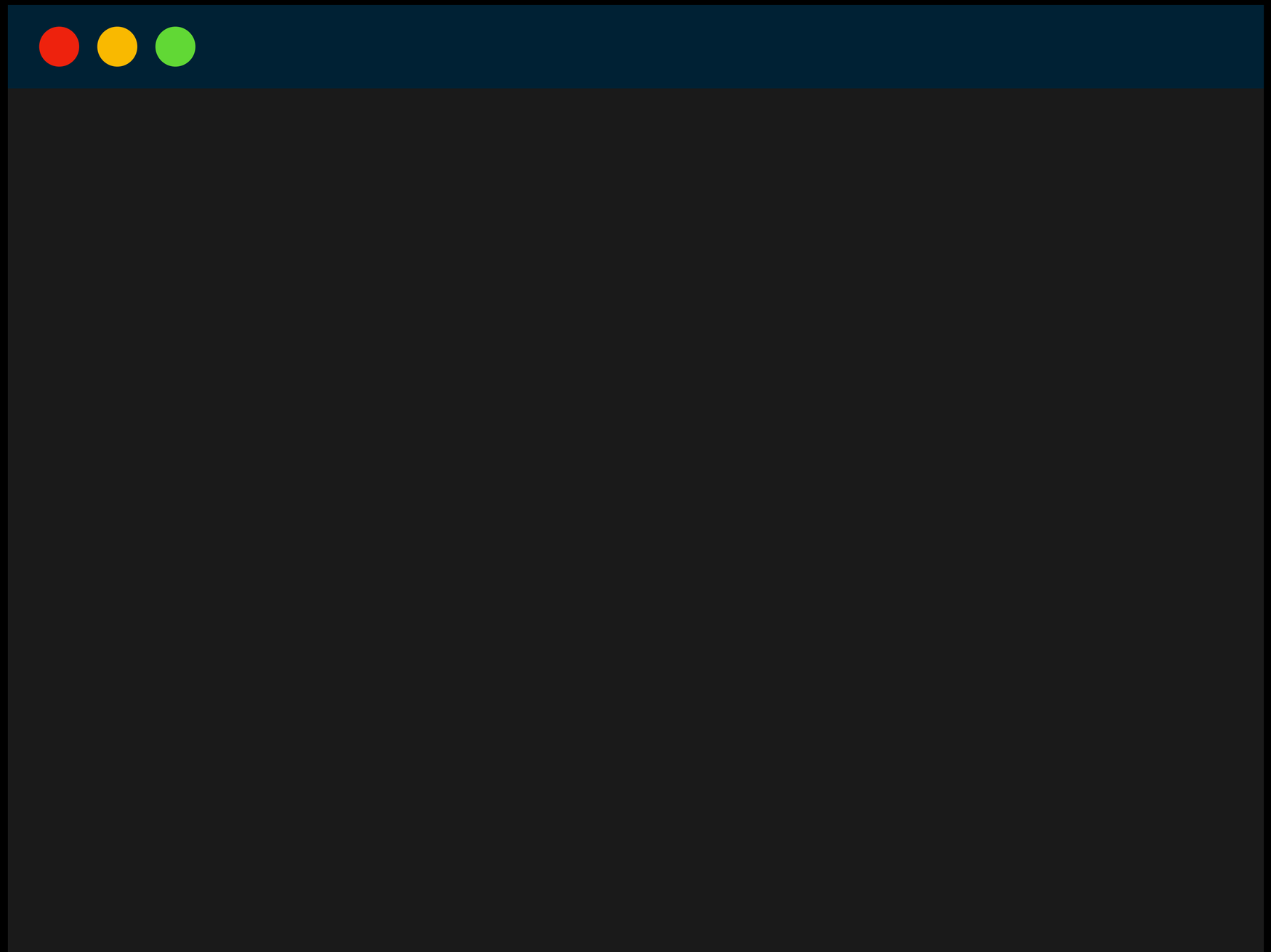
CONs

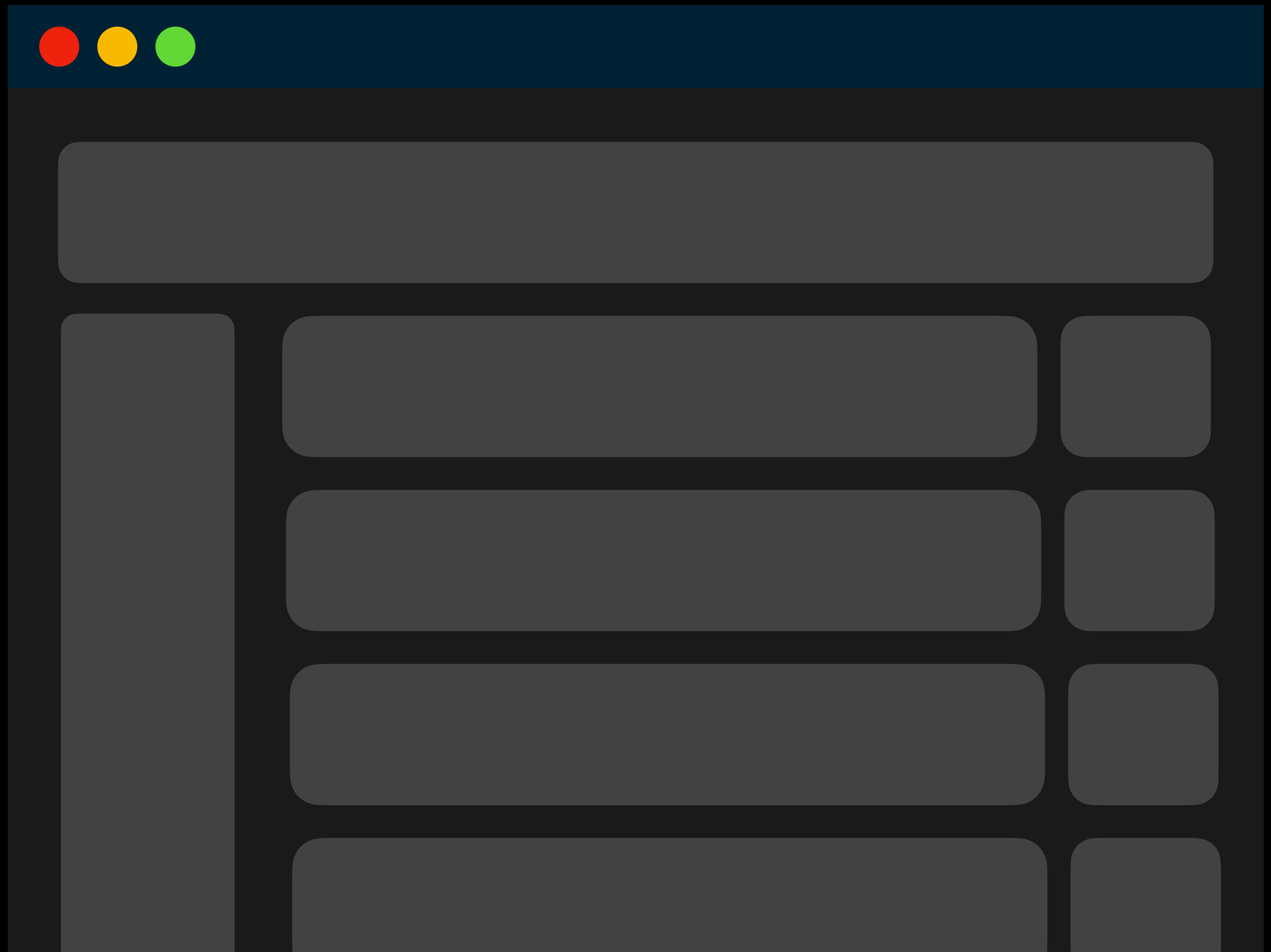
- SEO
- Initial performance
- Bundle size

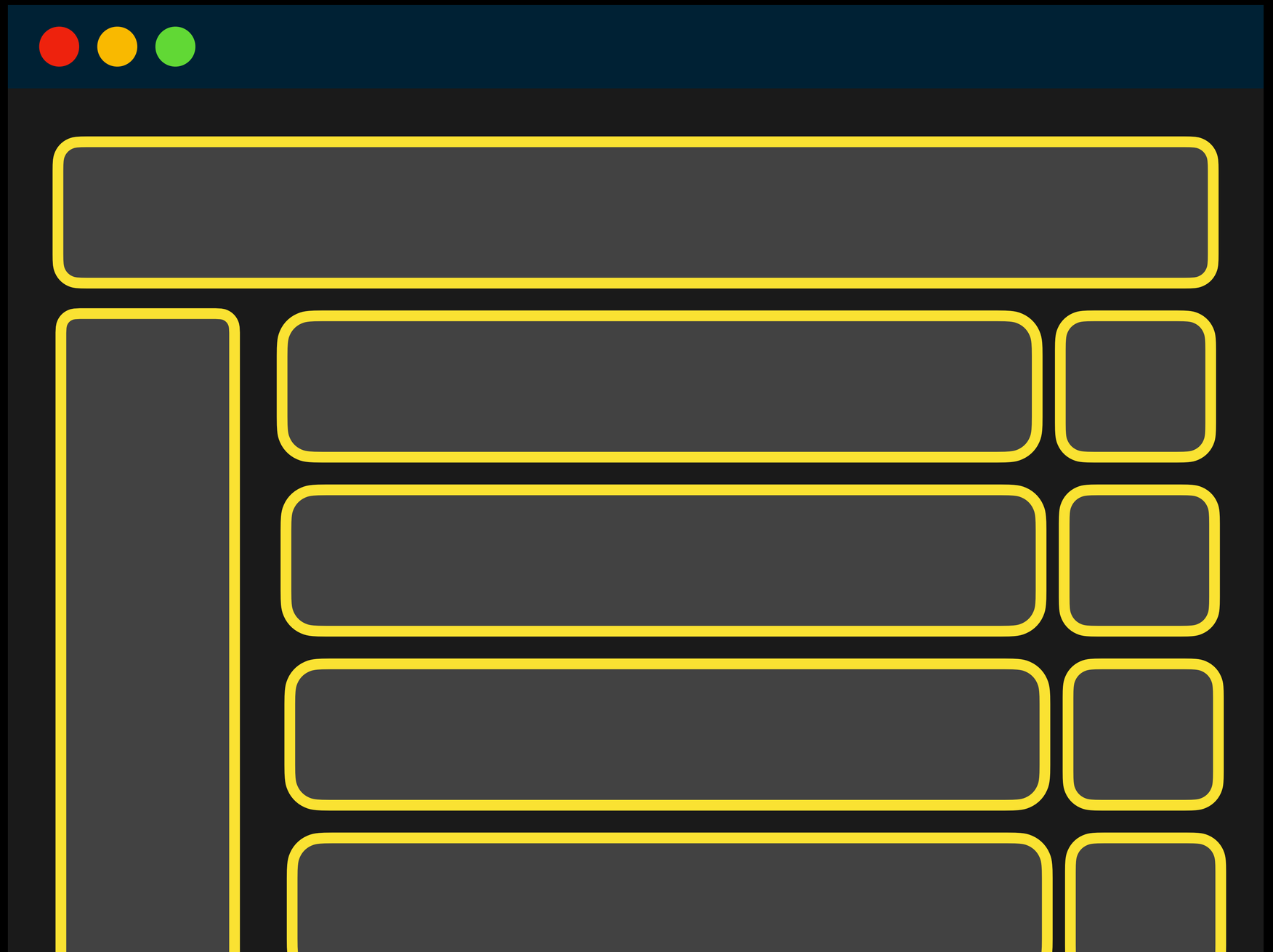
SSR 2.0

SSR FOR JS-BASED FRAMEWORKS





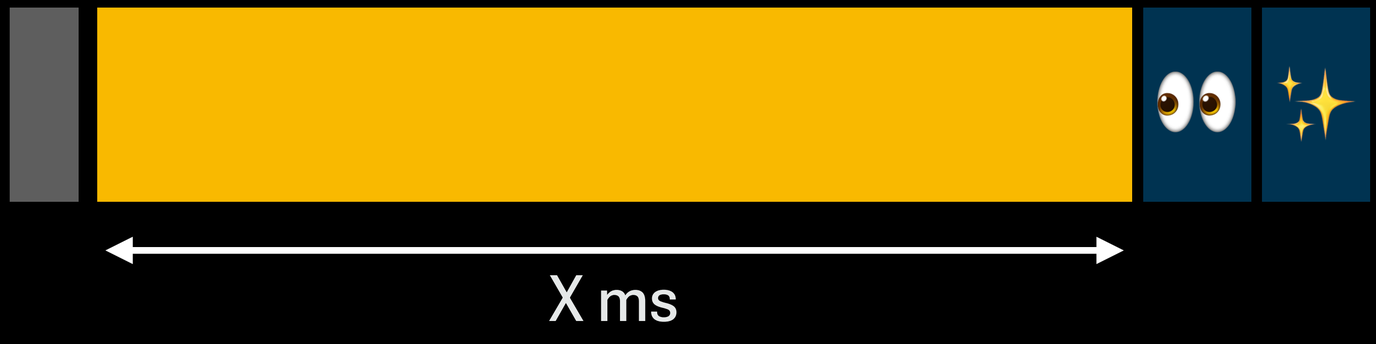


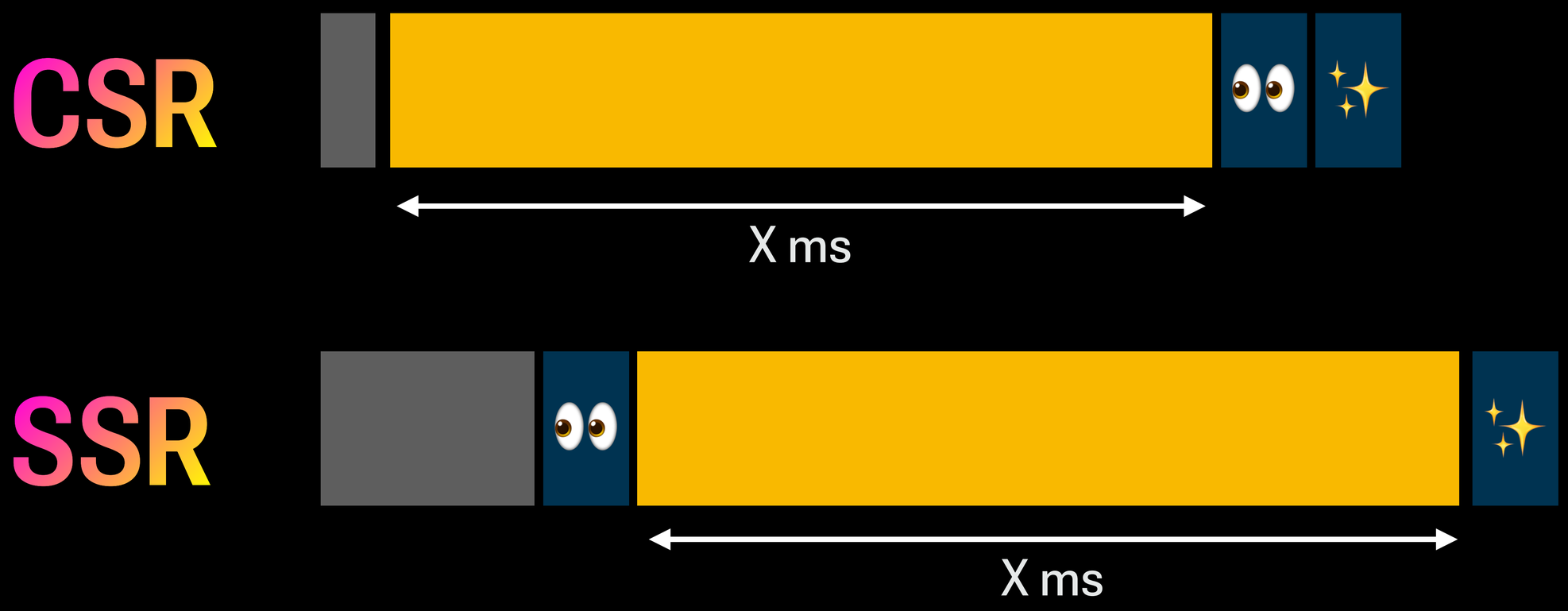


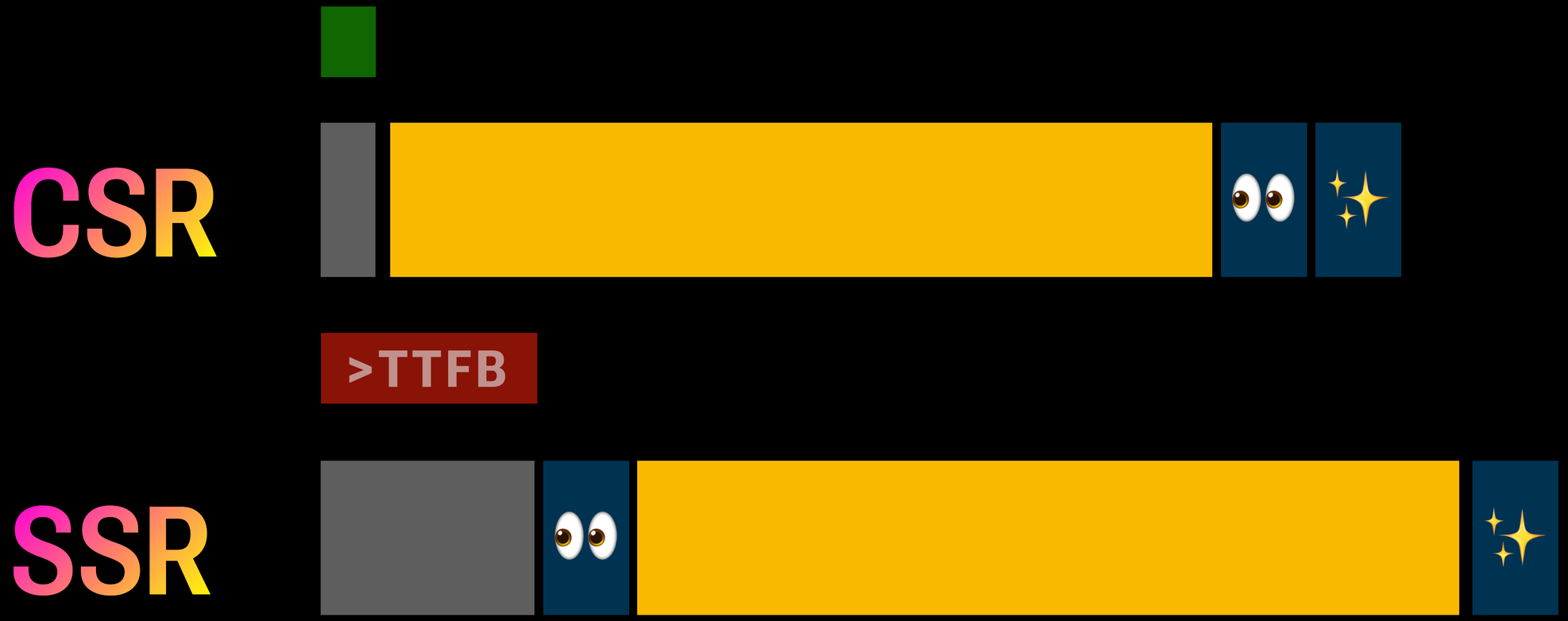
Hydration

✨ The app is re-executed on the browser to make it interactive

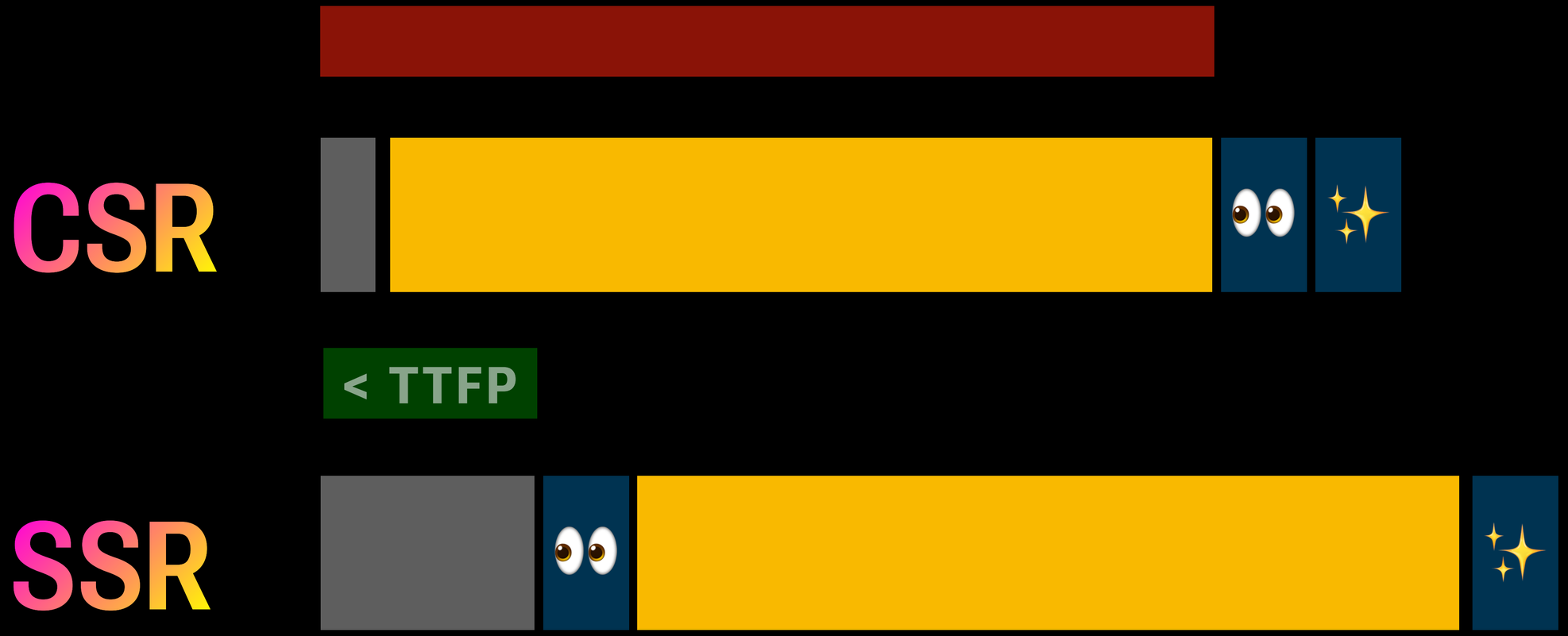
CSR

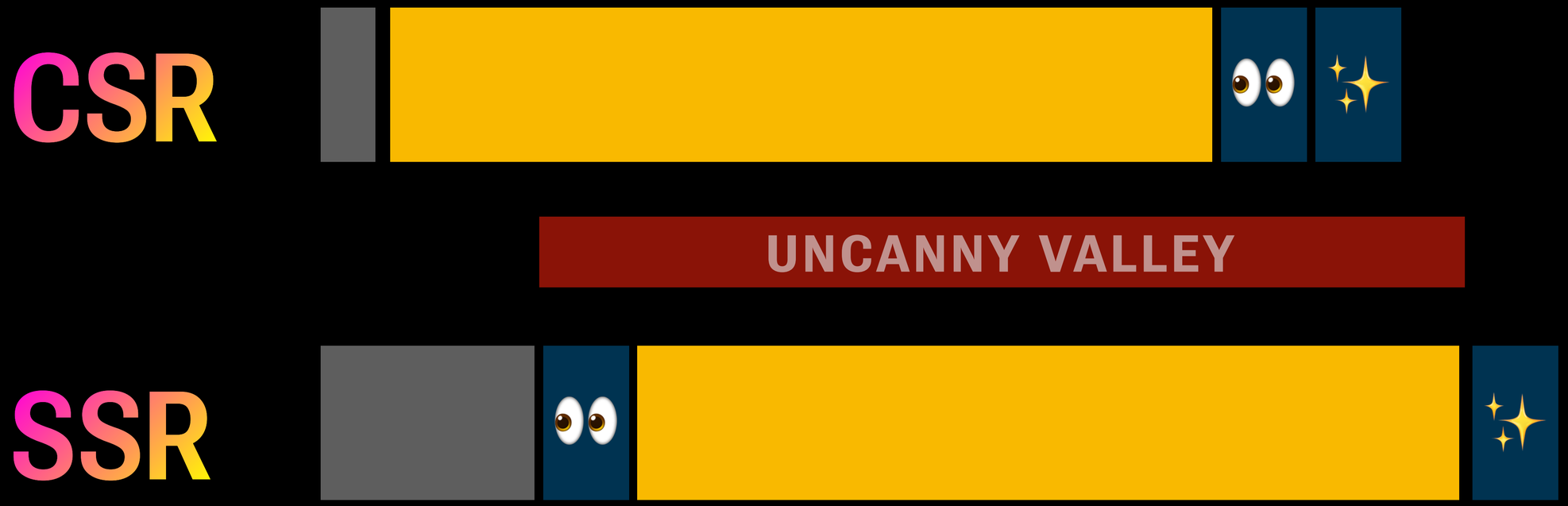



CSR
X ms
SSR
X ms


CSR
👀
✨
>TTFB
SSR
👀
✨


CSR
>TTI
SSR


CSR
👀
✨
< TTFP
SSR
👀
✨


CSR
UNCANNY VALLEY
SSR

Uncanny Valley

😕 When, while hydrating, the app is not interactive

SSR is not free

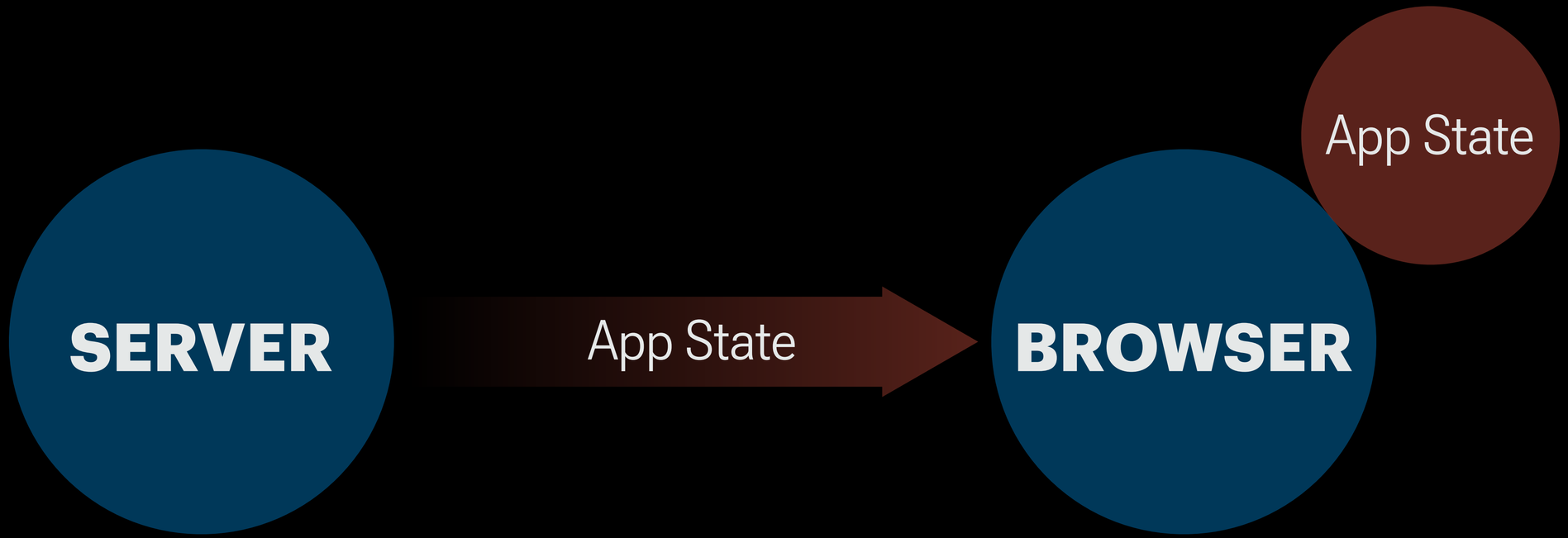
- > Time to interactive, > Time to first byte
- Uncanny Valley (if hydration)
- No ***document.window*** on the server (localStorage, height / width…)
- Partial run on the server (eg. React skips ***useEffect***'s)
- App State transfer
- App State serialization / deserialization

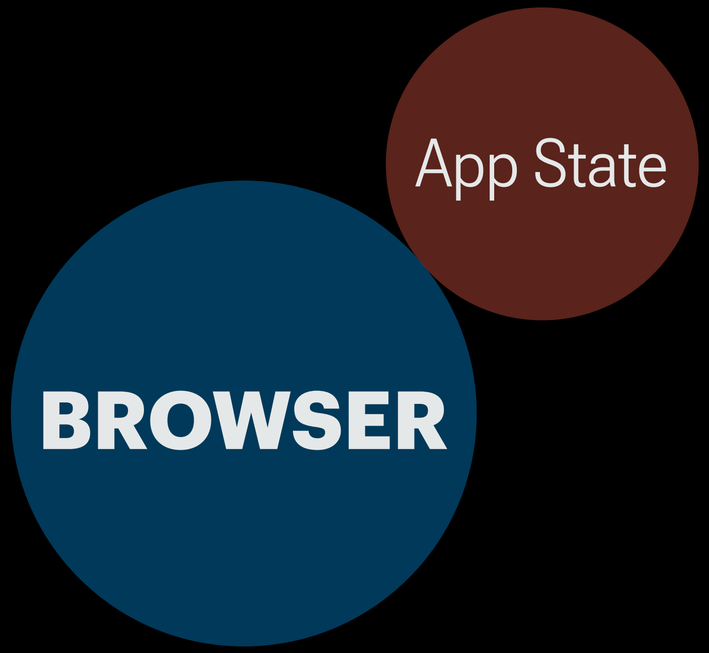
On the server

- Fetch API
- Populate variables
- Generate the HTML

On the client

- Receive the HTML
- …But the variables are not populated


App State
SERVER
App State
BROWSER



Size = JS Framework + JS App + HTML + State

Is SSR worth the trouble?

- “Apps and Sites are two separate things”
  - …but the distinction is becoming blurry
- “Devices are faster every year”
  - …but we’re shipping more and more JavaScript
- “My framework is fast enough with CSR”
  - …but expectations change with time
- “My app is protected by authentication and users have long sessions”
  - Then that’s probably not a big deal
  - …but what if SSR was not an afterthought?

So...

...delay Hydration?

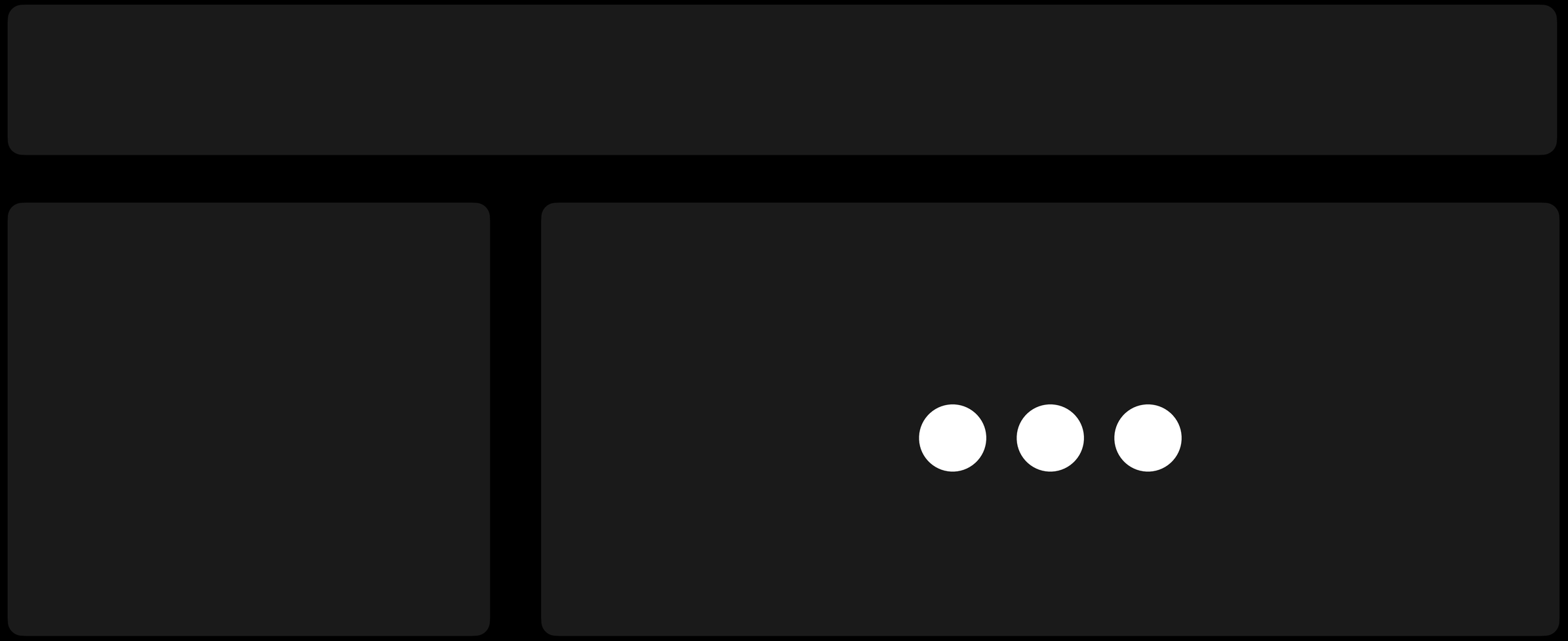
Progressive Hydration

🚀 Don’t hydrate all at once

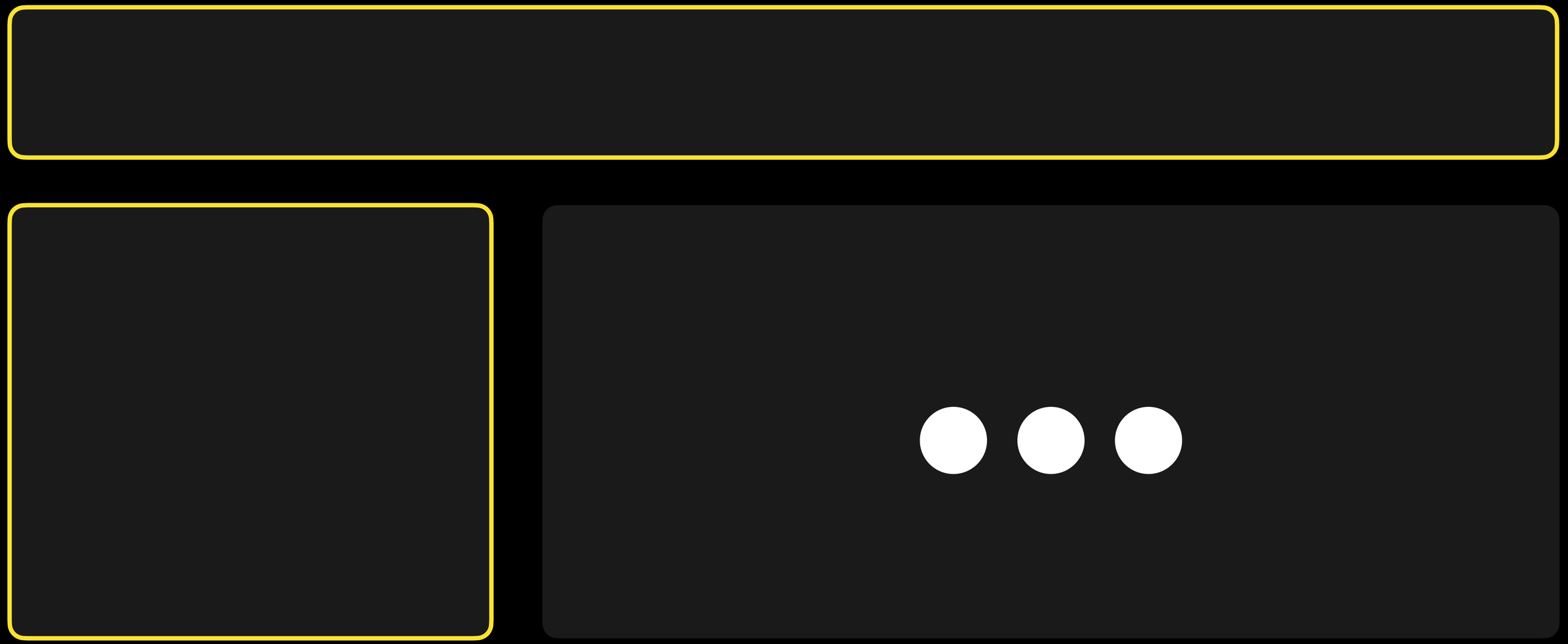
Suspense, Streaming SSR

```
<Layout>
  <Navbar />
  <Sidebar />
  <Suspense fallback={<Spinner />}>
    <Comments />
  </Suspense>
</Layout>
```

Suspense, Streaming SSR



Suspense, Streaming SSR



Suspense, Streaming SSR

SSR

SSR

Progressive
Hydration

Maybe...

...less Hydration?

Partial Hydration

🚀 Only hydrate some parts of the app

Astro

Static Site Generator 🚀 Bring your own Framework 🚀 Ship Less JavaScript

- Island architecture
- Partial Hydration (zero JavaScript default)
- Progressive Hydration (on interaction, on visibility...)
- Mix components: React, Preact, Svelte, Alpine, Vue, Lit, Solid...
- SSR support (personalized requests, login state...)

Islands Architecture (Astro)

```
import MyReactComponent from '…';

<!— 100% HTML, Zero JavaScript —>
<MyReactComponent />

<!— Interactive —>
<MyReactComponent client:load />
```

Directives

- client:load
- client:idle
- client:visible
- client:media
- client:only

Islands Architecture (Rocket)

```
<my-element loading="client"></…>
<my-element loading="server"></…>
<my-element loading="hydrate:*"></…>
```

Hydrate options

- onClientLoad
- onClick
- onMedia
- onVisible
- onHover
- withIdle
- withDelay

<is-land>

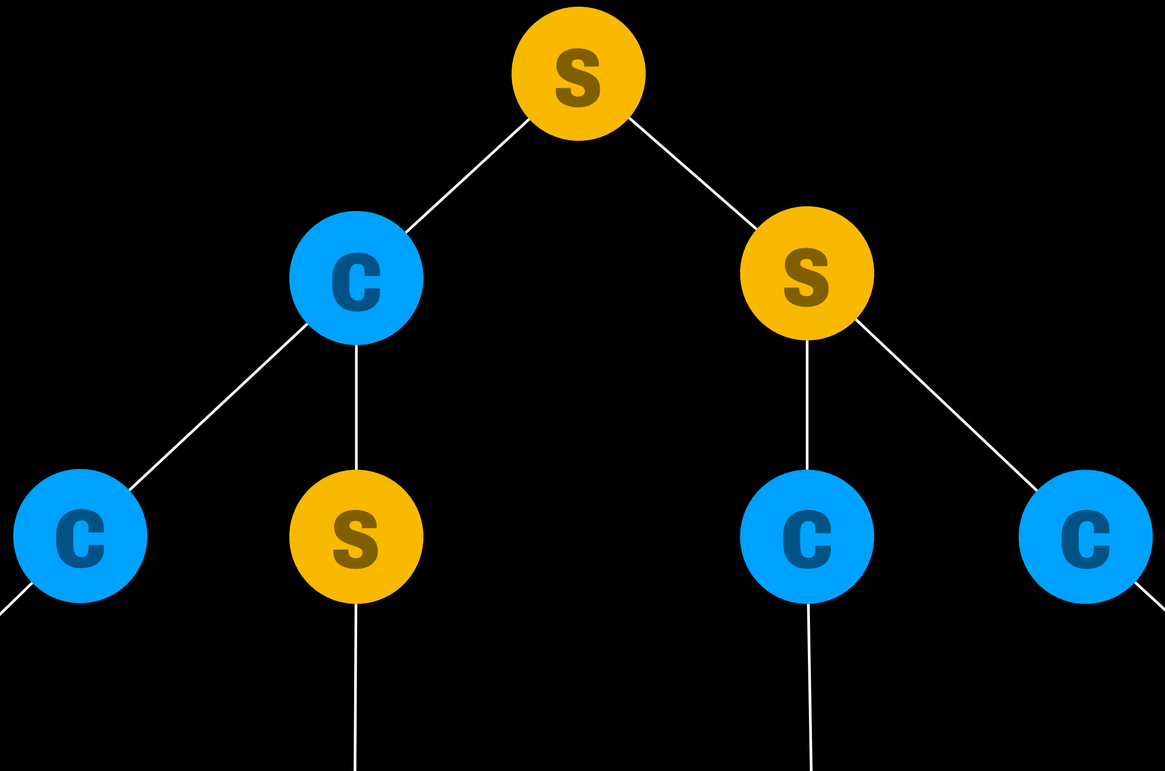
🚀 Framework-agnostic selective hydration

```
<is-land on:visible>
  <my-element></…>
  <my-react-app></…>
  <my-vue-app></…>
</is-land>
```

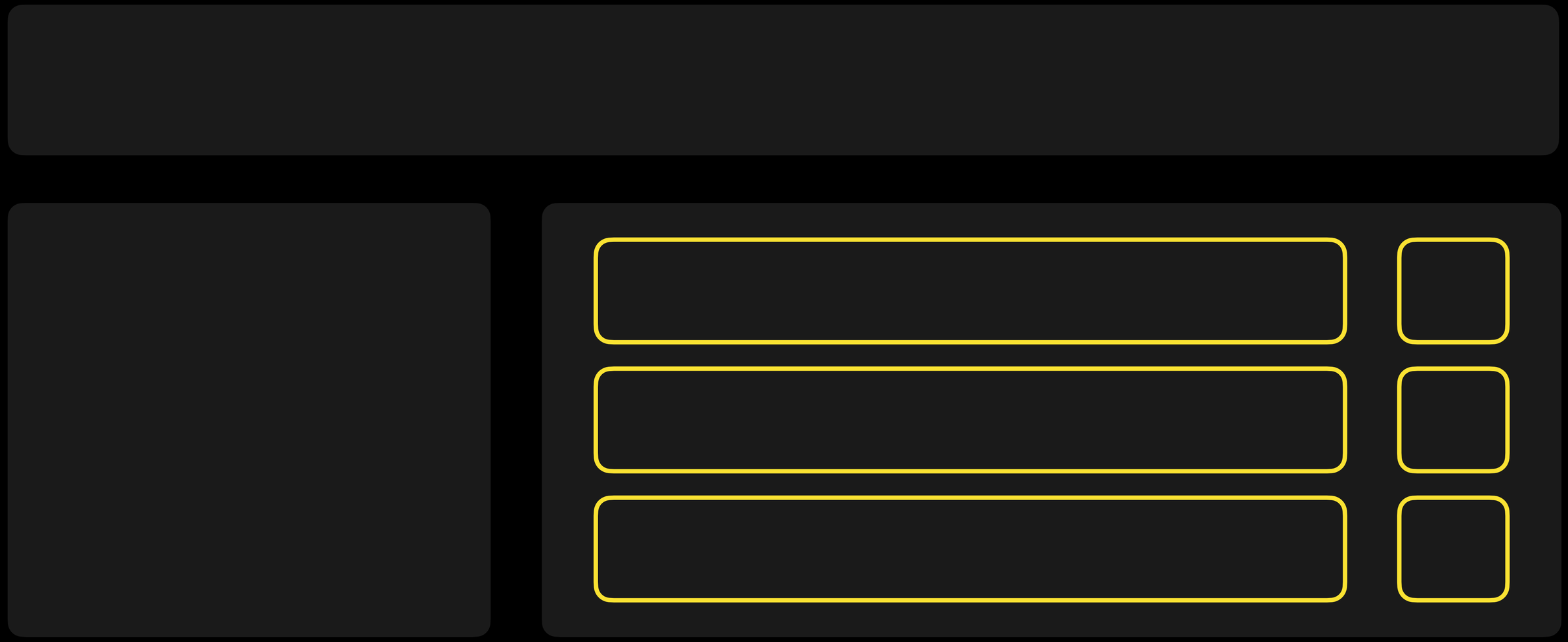
```
<is-land on:visible>
  <my-element>
    <template shadowroot=“open”>
      Count: 0
    </template>
  </my-element>
</is-land>
```

Server Components

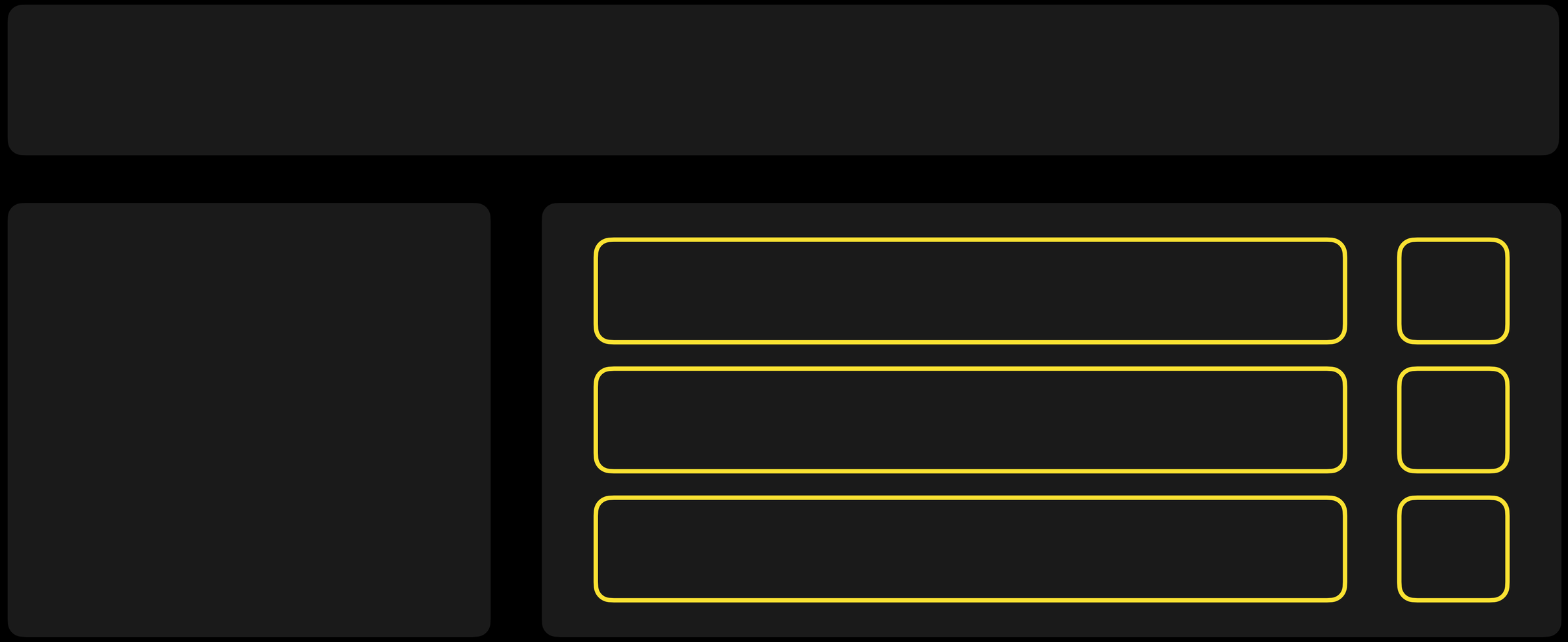
🚀 Components without state, only executed on the server



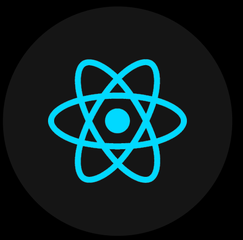
Server Components



Server Components



Server Components



🚀 Components without state, only executed on the server

- Partial Hydration
- SPA (client-side routing)
  - Re-render server components on the server, get the result via AJAX
- Very sophisticated, React exclusive feature
- Manual boundaries (component file names)

Progressive

- Most frameworks hydrate top-down… if a component needs hydration, all its parents do too

Partial (islands)

- Needs manual boundaries
- Needs sophisticated workarounds for SPAs (eg. React Server Components)

What if...

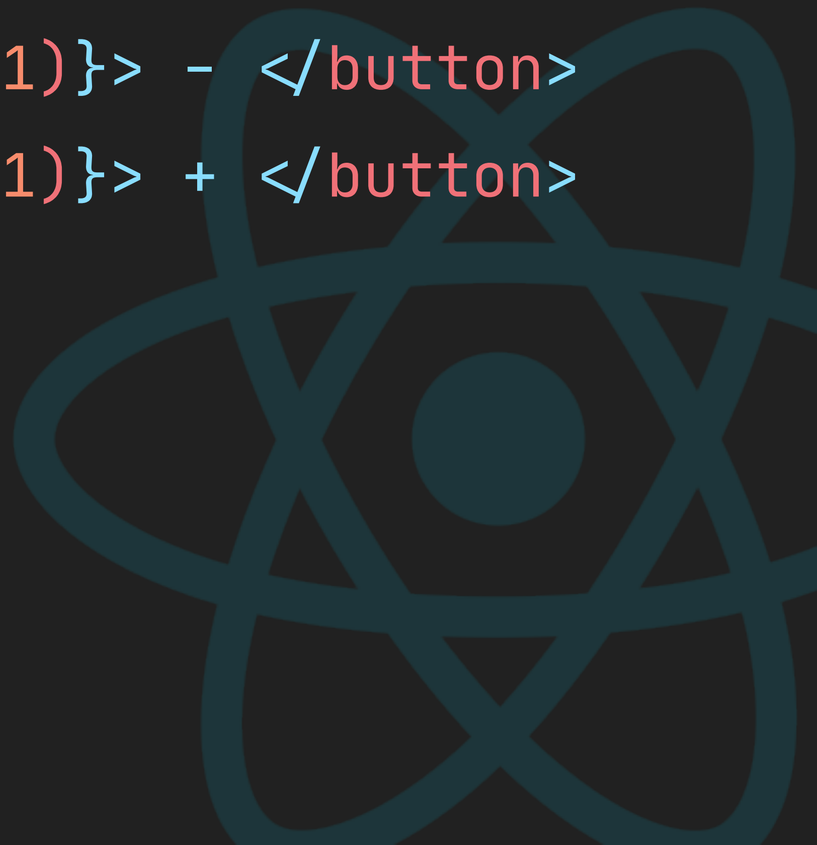
OUT OF ORDER
GRANULAR
SPA READY

HYDRATION

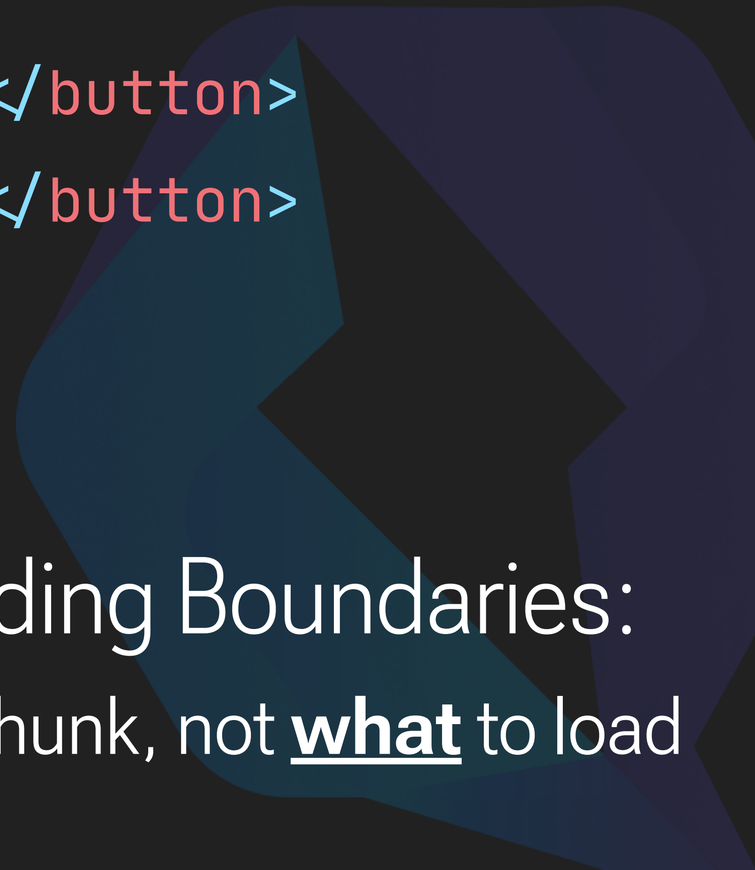
Qwik introduces

RESUMABILITY

```
const Counter = () => {
  const [count, setCount] = useState(0);

  return <div>
    Count: {count}
    <button onClick={() => setCount(c => c - 1)}> - </button>
    <button onClick={() => setCount(c => c + 1)}> + </button>
  </div>
}
```



```
const Counter = component$(() => {
  const store = useStore({ count: 0 });

  return <div>
    Count: {store.count}
    <button onClick$={() => store.count--}> - </button>
    <button onClick$={() => store.count++}> + </button>
  </div>
})
```



Lazy Loading Boundaries:

where to chunk, not **what** to load

Output files

- Component render function
- Increment handler function
- Decrement handler function

Component

```
export const Counter = () => {
  const store = useStore({ count: 0 });

  return <div>
    Count: {store.count}
    <button onClick$={qrl(() => import('a.js'), 'dec', [store])}> - </button>
    <button onClick$={qrl(() => import('b.js'), 'inc', [store])}> + </button>
  </div>
}
```

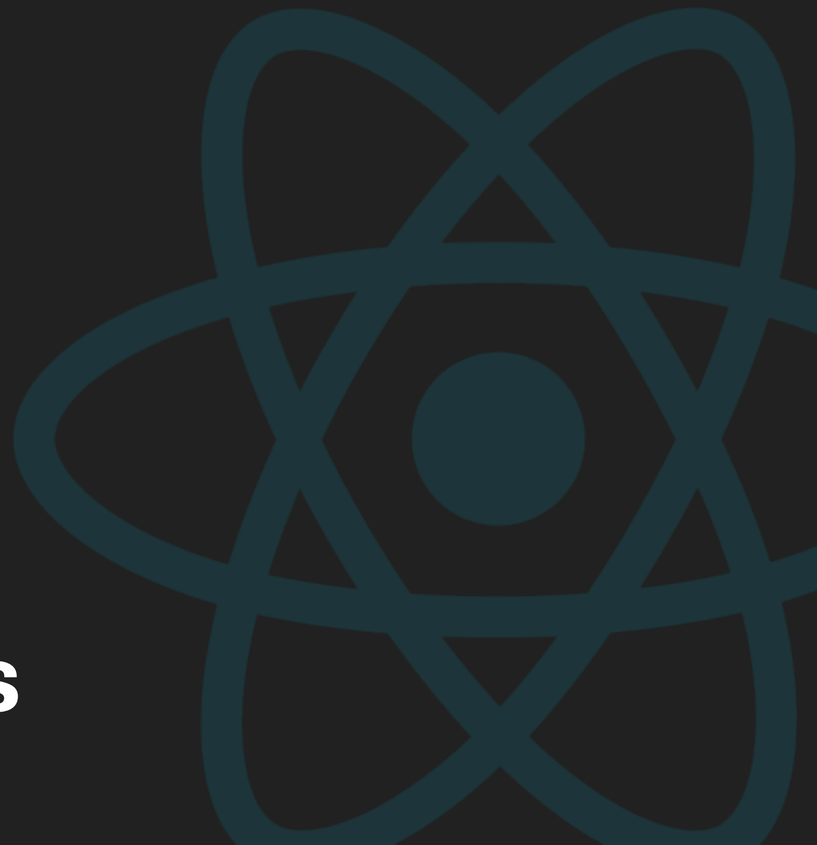
Decrement (a.js)

```
import { useLexicalScope } from ‘@builder.io/qwik’;

export const dec = () ⟹ {
  const [store] = useLexicalScope();
  return store.count--;
}
```

Increment (b.js)

```
import { useLexicalScope } from ‘@builder.io/qwik’;

export const inc = () ⟹ {
  const [store] = useLexicalScope();
  return store.count++;
}
```

```
<div>
  Count: 0
  <button> - </button>
  <button> + </button>
</div>
<!— React, App —>
<script>…</script>
```



No props, no state, no event listeners

```
<div>
  Count: 0
  <button on:click="./a.js#dec"> - </button>
  <button on:click="./b.js#inc"> + </button>
</div>
<!— App State, Framework State —>
<script type="qwik/json">{ … }</script>
<!— Global listener (1kb) —>
<script id="qwikloader">…</script>
```

Resumability in  Qwik

👆 Click

The global listener receives the event

Consume Qwik State

App State and Framework State are de-serialized

Download handler

The appropriate file containing the handler is downloaded

Run handler

The handler is executed, the state is updated

Download component

Qwik detects that the state is involved in the component's template, its file is downloaded

Re-render

The component is re-rendered

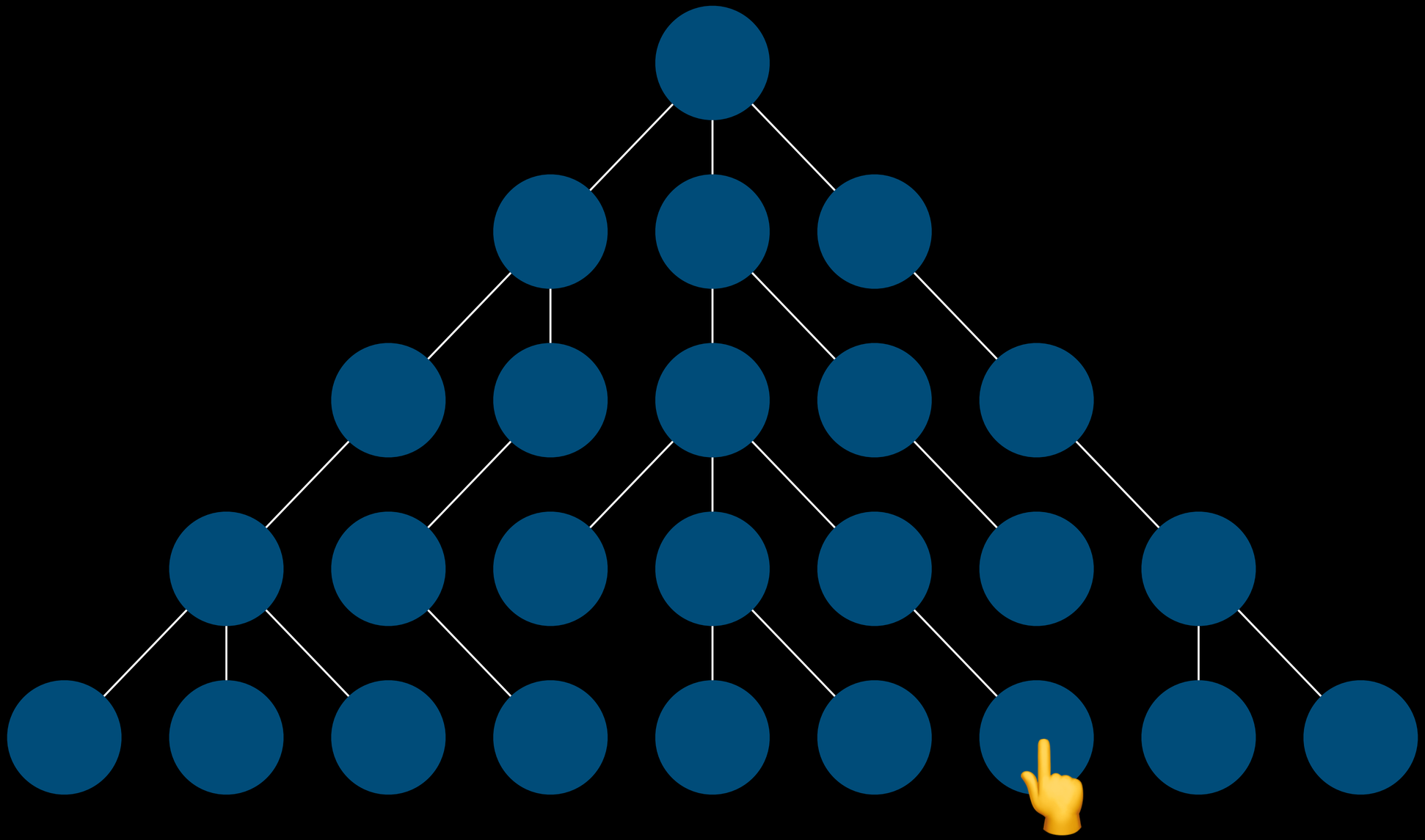
UNCANNY VALLEY

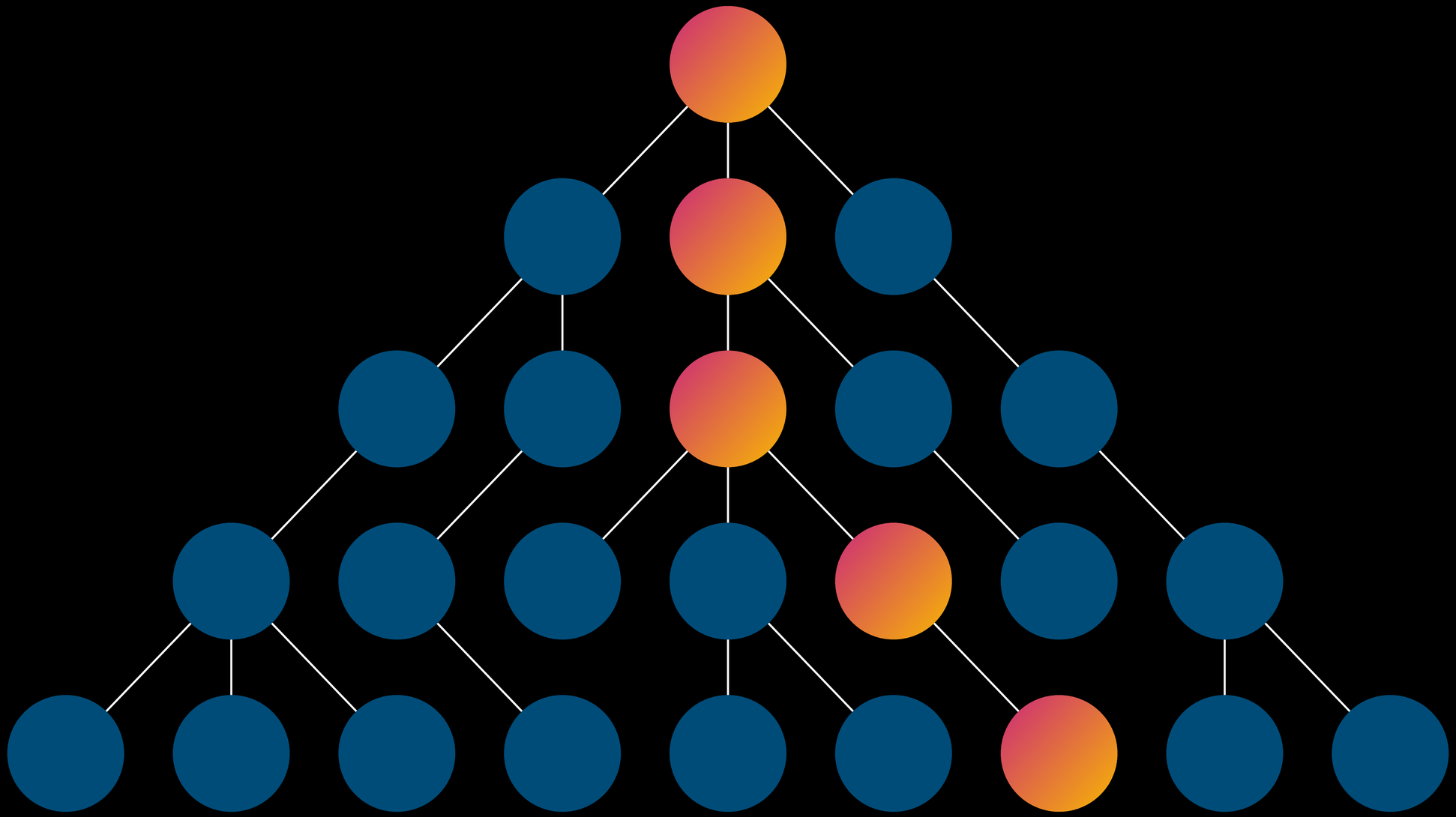
HYD

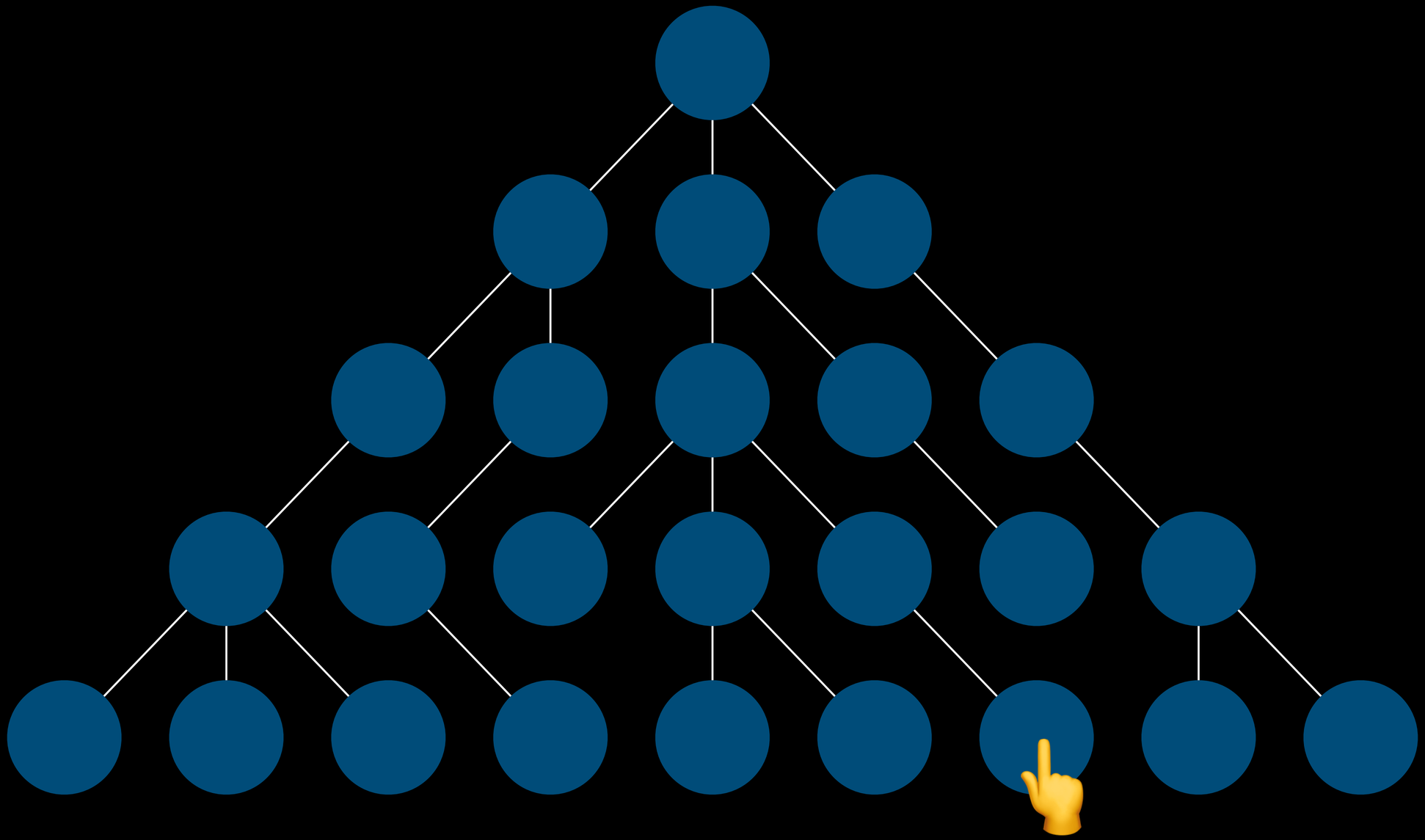
RES

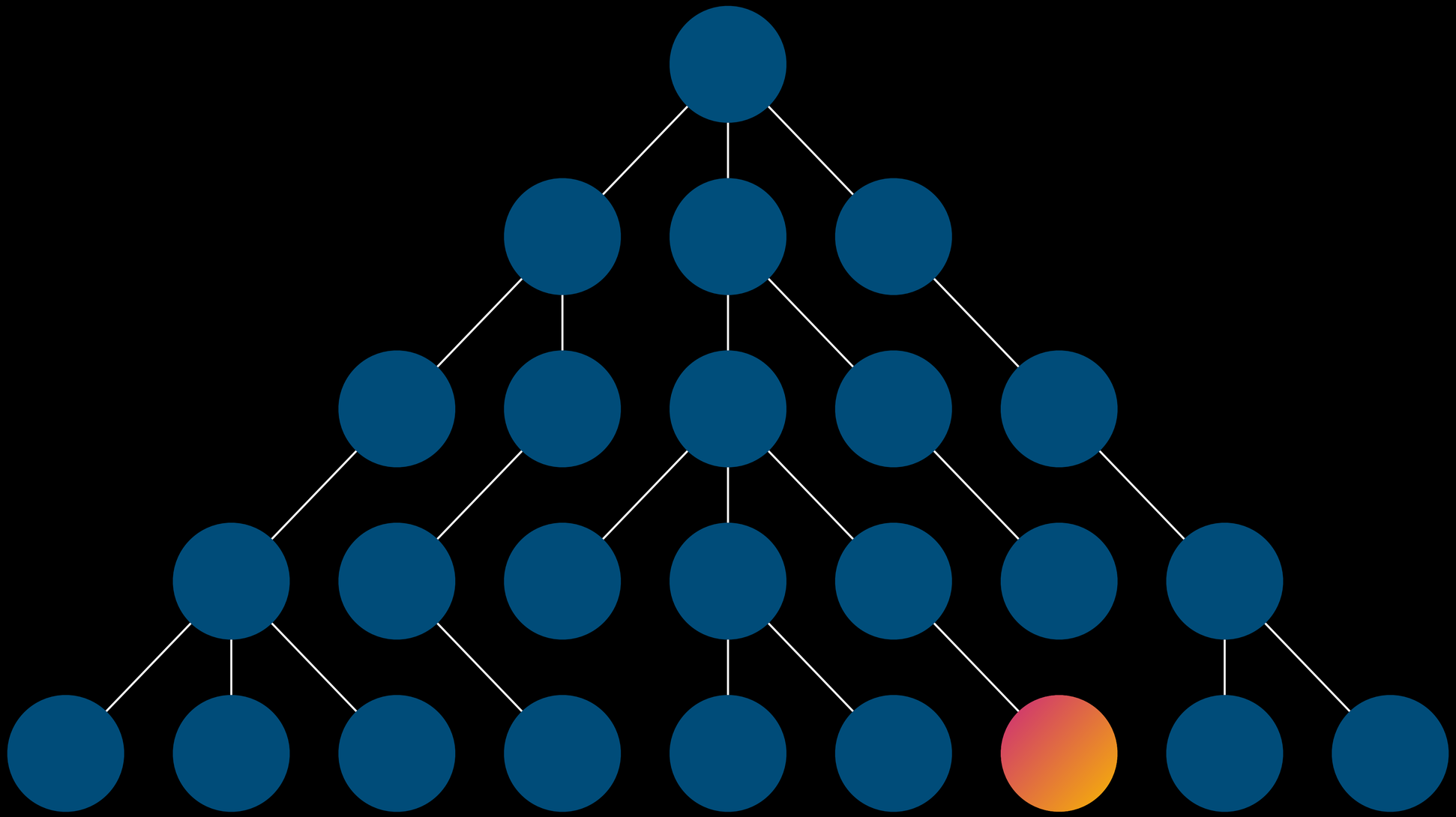
Resumability isn’t easy

- Serialize everything
  - App State, Framework State
  - Props
  - Component boundaries
  - State -> Component relationships
- Setup global event listeners
- Render components in isolation, out of order







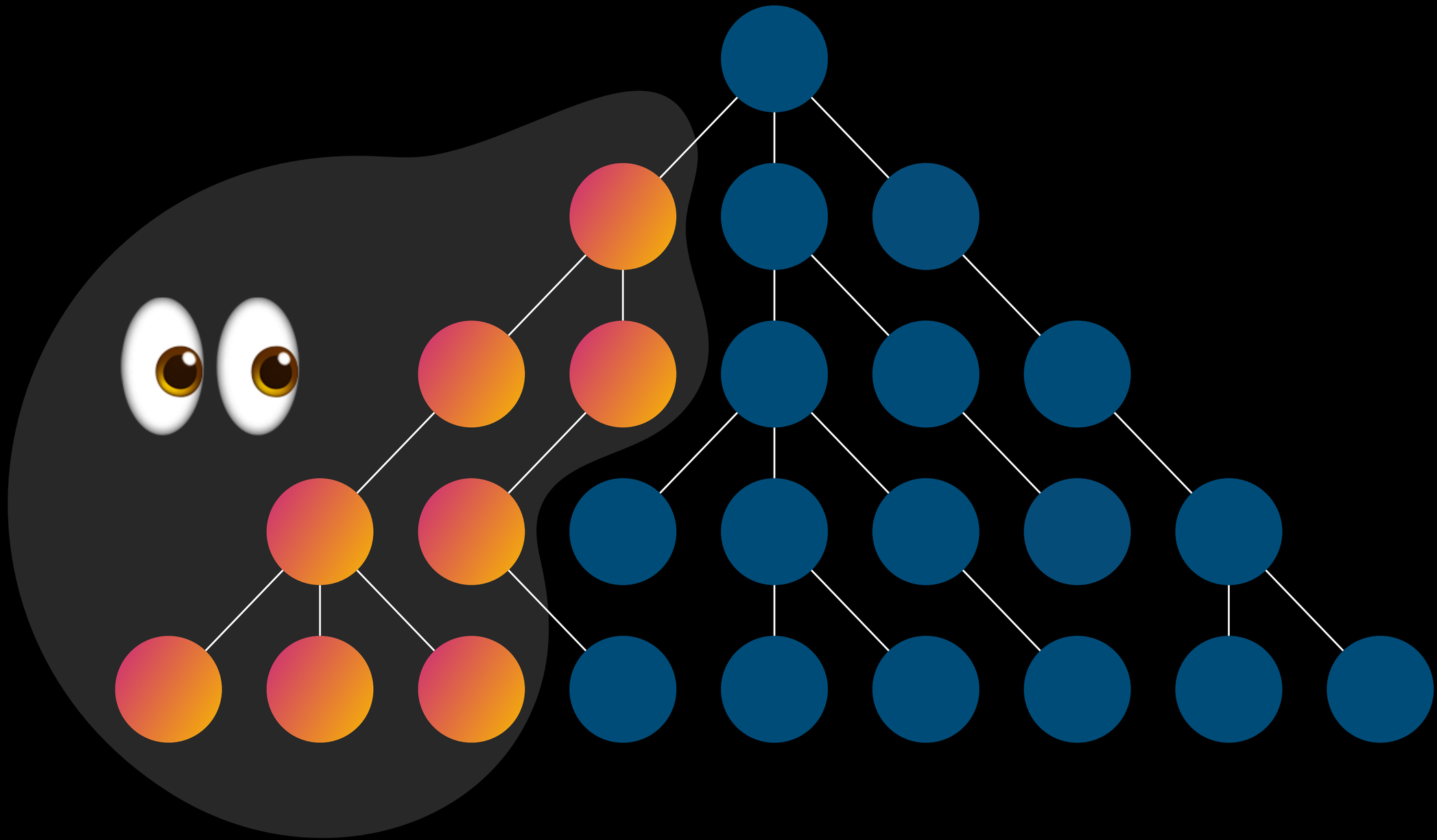


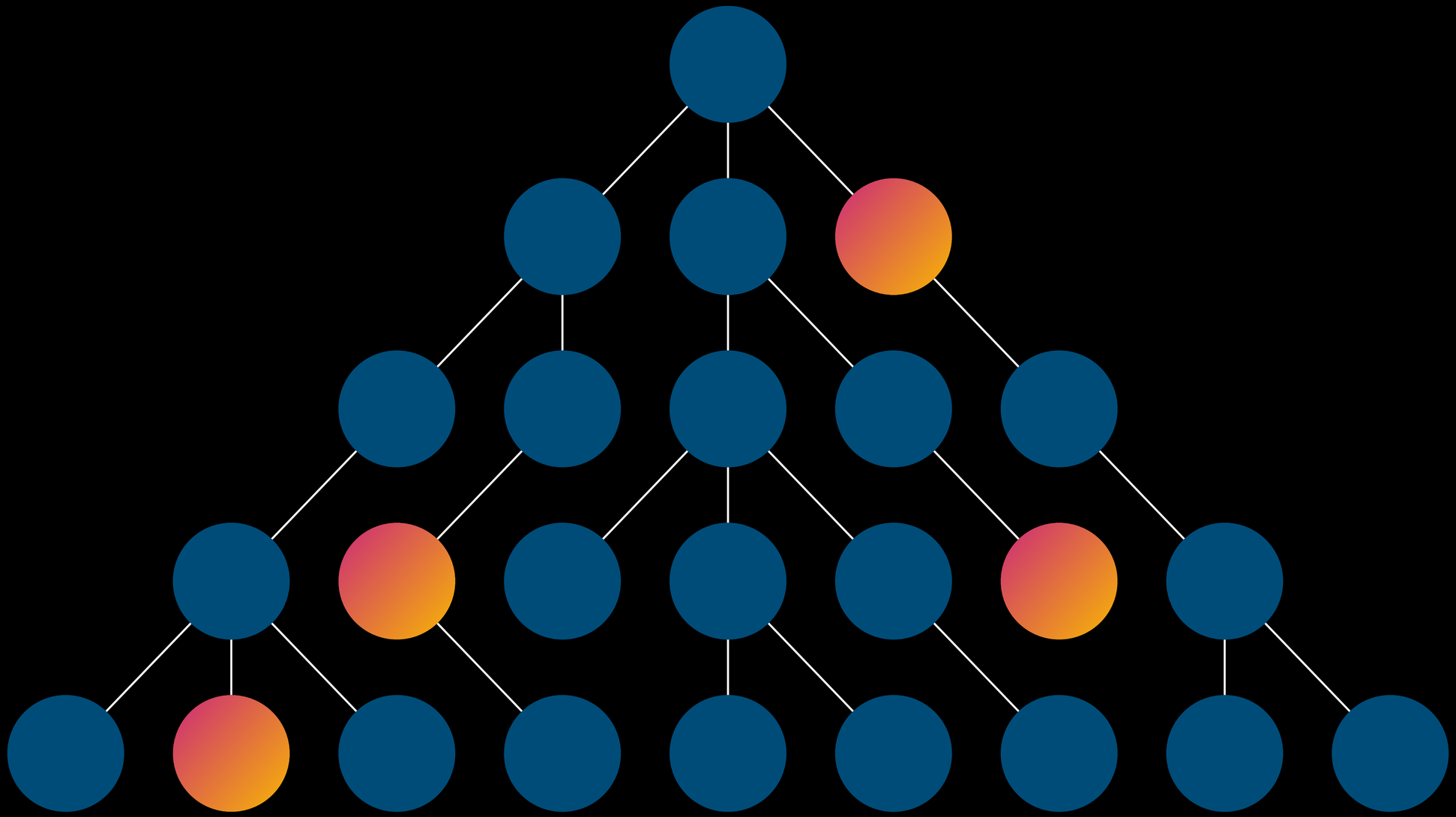
Resumability isn’t free

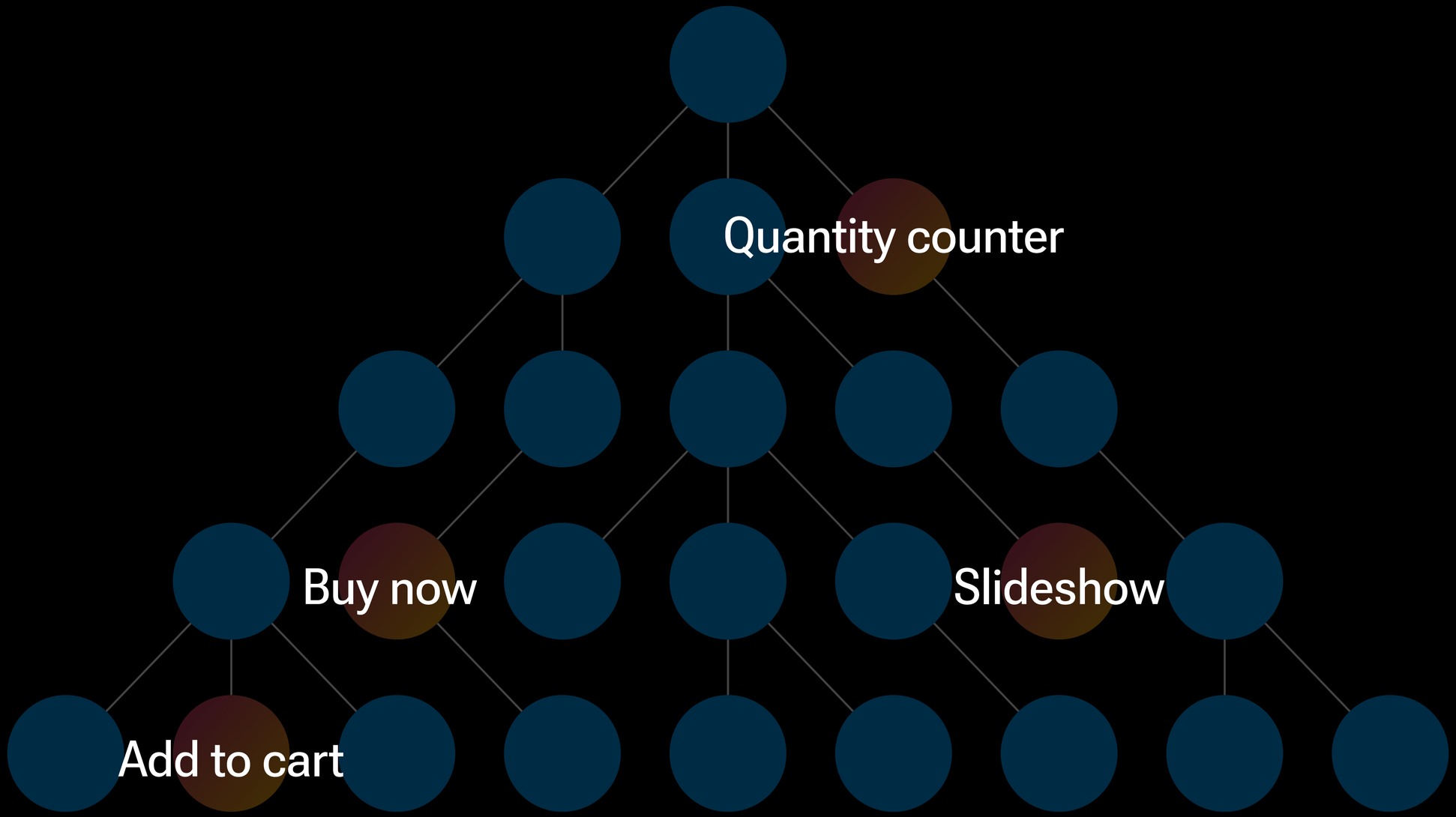
- Many restrictions due to serialization and asynchronous code. Examples:
  - Cannot refer to non-serializable data (eg. Promises) via closure
  - Cannot directly pass functions as props (must be wrapped)
  - Cannot use “event.preventDefault()” (must use ad hoc HTML attribute)
- In Qwik, you have to know what the framework can do, what it can’t do, and when there’s hidden “compiler magic” involved. Example: you can’t serialize Dates in JSONs, but Qwik can.

Main concerns

- Developer Experience
  - Tooling
  - Maturity, Documentation, Ecosystem
  - Restrictions & conventions: it’s not “Just JavaScript”
- Input lag
  - Can be solved with Prefetching
    - Prefetching ❤️ Analytics






Quantity counter
Buy now
Slideshow
Add to cart

Recap

- Hydration is expensive and can be slow. And, a lot of JS could be unnecessary.
- Progressive Hydration ➡️ lazy loading some JS
- Partial Hydration, Islands ➡️ skipping some JS
  - Mainly MPAs
  - React Server Components (SPAs)
- Resumable frameworks ➡️ lazy load all JS
  - Qwik, Marko v6 (eBay), Wiz (Google)

RESUMABILITY:

THE NEXT PARADIGM

OR

PASSING TREND?

Michele Stieven

Angular GDE, Founder of accademia.dev

@MicheleStieven