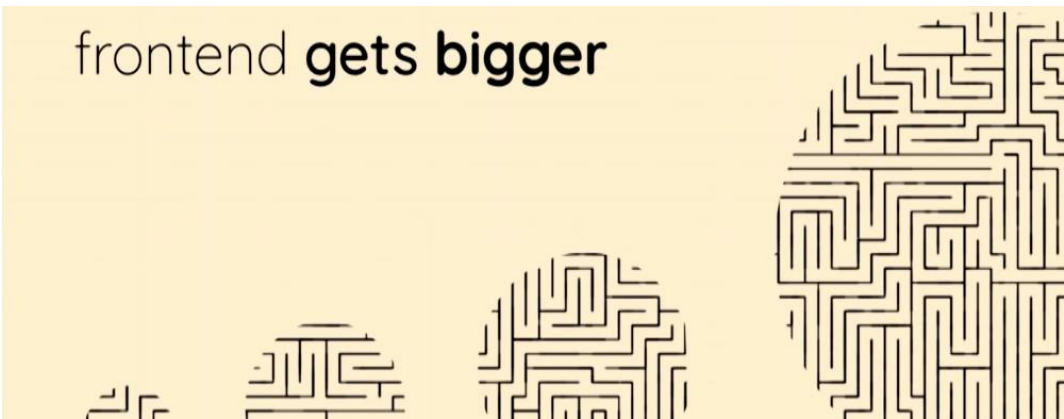





JavaScript frameworks are on the rise. More and more companies employ specialized frontend teams which are building client-side web applications. As projects grow, things start to get more complicated. Building a feature requires coordination between frontend and backend teams. In my talk I present an alternative organizational model where teams have end-to-end ownership. Each team focusses on one specific use-case and not a technology. Building a site with independent cross-functional teams has major advantages. It reduces communication and make shipping features fun again.





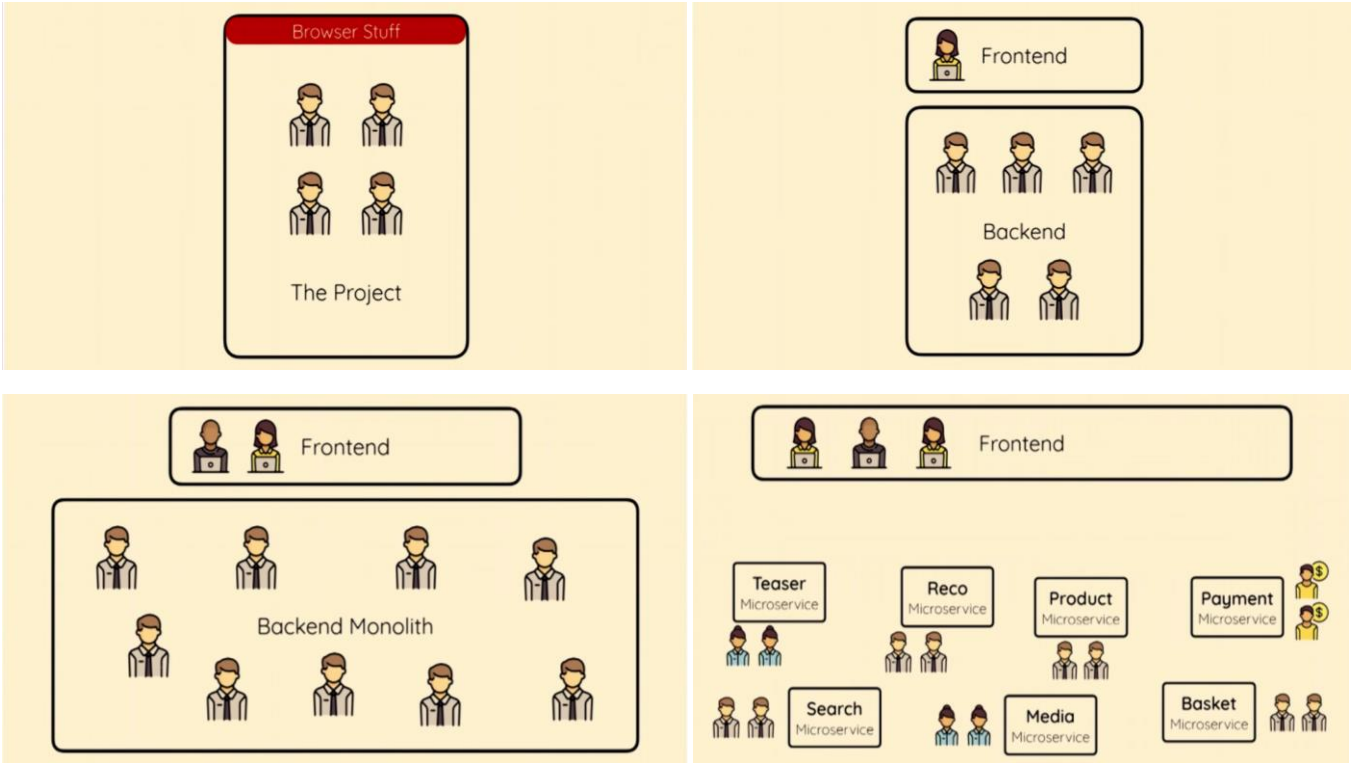
Michael Geers
Frontend Engineer

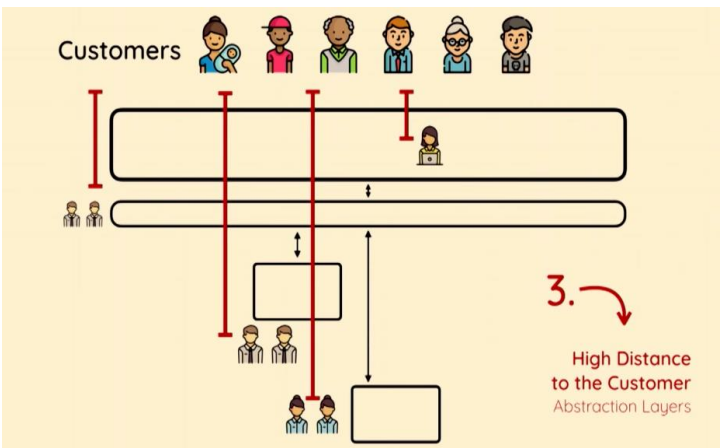
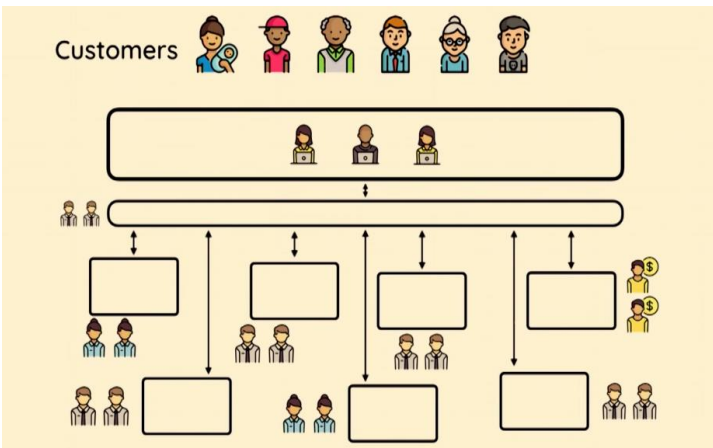
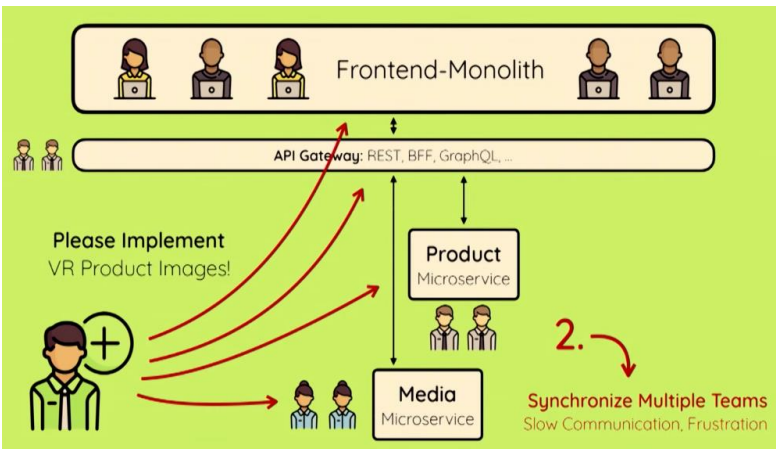
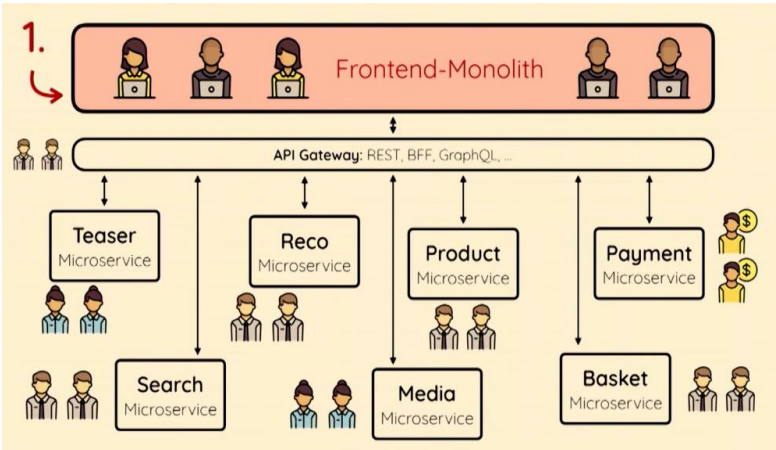
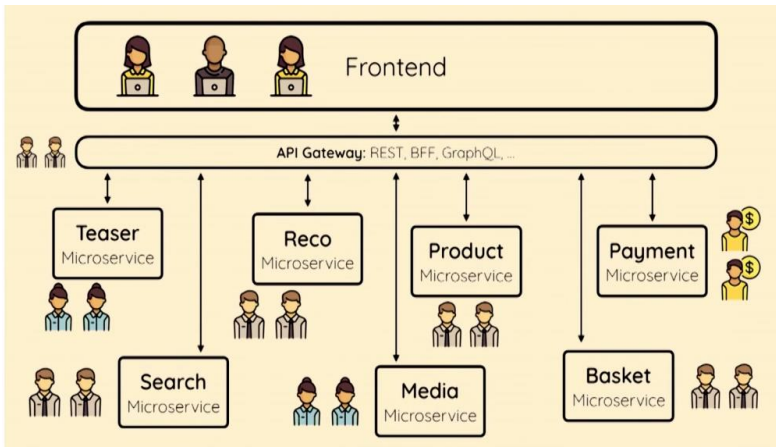
naltatis
on Twitter & GitHub

Micro Frontends

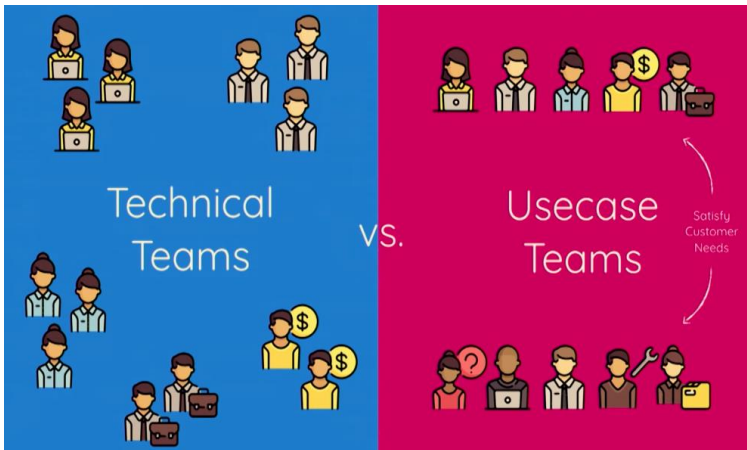
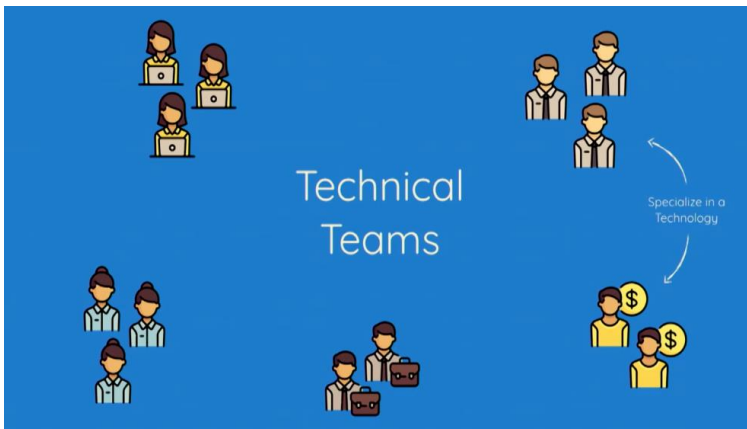
- Verticalized Teams
- Self Contained Systems
- Vertical Decomposition
- UI Composition

Architecture

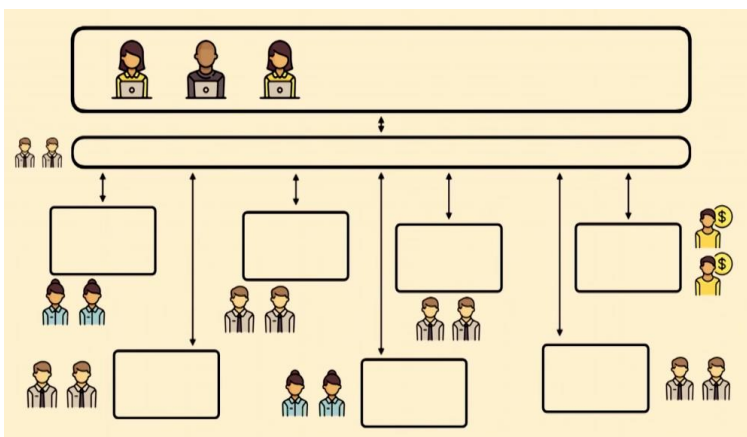
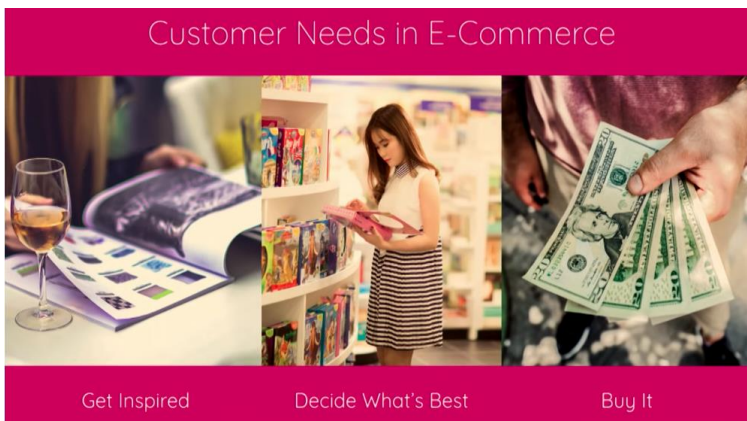




The distance to customer needs increases as we move down the stack

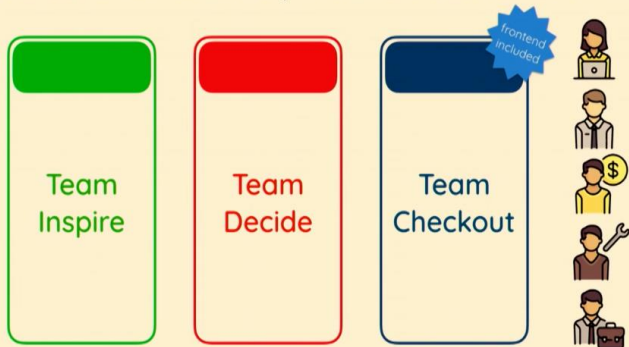


Cross functional teams are much better at being able to build what customers need quickly

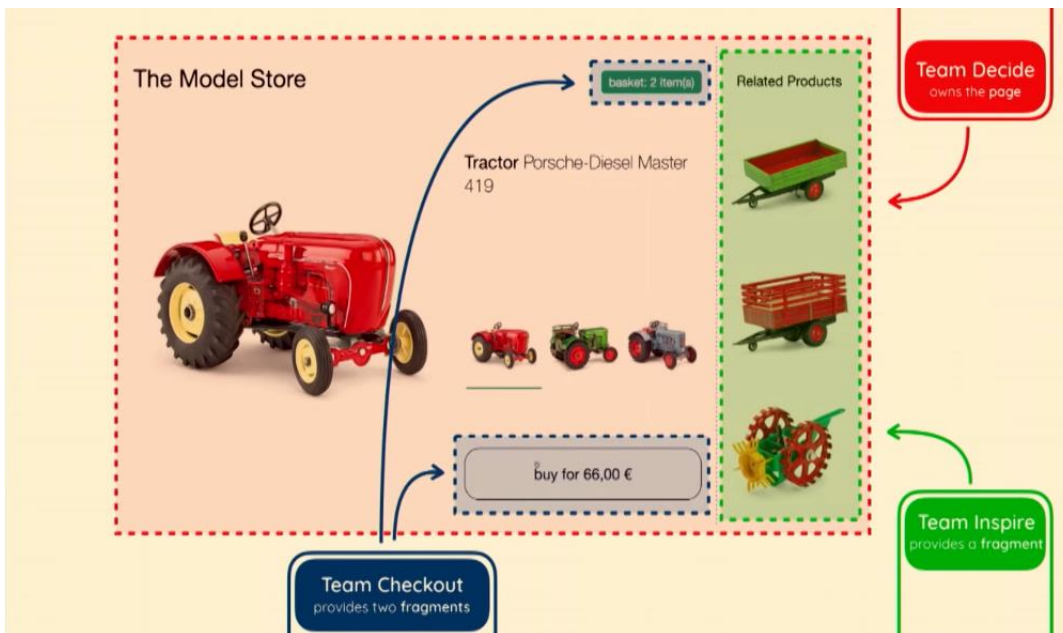
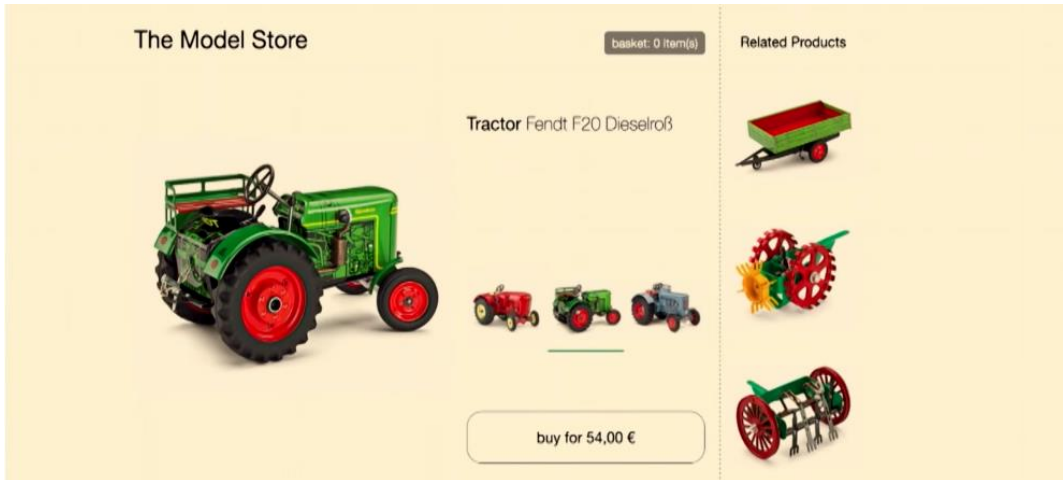
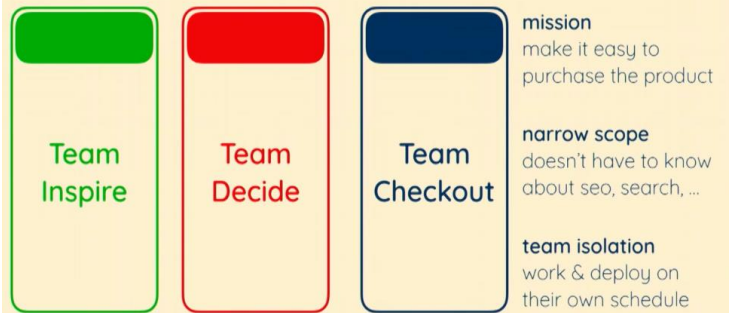


This is the classical microservice architecture that we need to improve on

End-to-End Teams / Micro Frontends



End-to-End Teams / Micro Frontends



How to integrate?

We need to have a mutual component model so that teams can be technology agnostic and use what they want

HTML
Imports

HTML
Template

Web Components

Custom
Elements

Shadow
DOM



Custom Elements

```
<checkout-basket></checkout-basket>
```



<https://developers.google.com/web/fundamentals/getting-started/primers/customelements>



Custom Elements

```
class CheckoutBasket extends HTMLElement {  
  connectedCallback() {  
    this.innerHTML = 'mini basket ...';  
  }  
}
```



```
customElements.define('checkout-basket', CheckoutBasket);
```

Element Lifecycle

```
class CheckoutBasket extends HTMLElement {  
  
  constructor() {...} is created  
  
  connectedCallback() {...} attached to DOM  
  
  attributeChangedCallback(attr, oldVal, newVal) {...} someone change an attribute  
  
  disconnectedCallback() {...} removed from DOM, cleanup!  
}
```

Browser Support

Custom Elements v1
API stabilized end of last year

Browser Support

with Polyfill (5KB)

<https://github.com/WebReflection/document-register-element>

`<checkout-basket>`
`</checkout-basket>`

basket: 0 item(s)

Tractor Porsche-Diesel Master
419

buy for 66,00 €

Related Products

`<inspire-reco sku="t_red">`
`</inspire-reco>`

`<checkout-buy sku="t_red">`
`</checkout-buy>`

These 3 fragment blocks are custom elements that the Red team that owns the page has to include in their code

`<checkout-basket>`
`</checkout-basket>`

basket: 0 item(s)

Tractor Porsche-Diesel Master
419

buy for 66,00 €

Related Products

`<inspire-reco sku="t_green">`
`</inspire-reco>`

`<checkout-buy sku="t_green">`
`</checkout-buy>`

When the user clicks on a product, the red team only has to toggle the SKU property from red to green and the custom elements on the page will react as needed. Each fragment gets a call and can re-render itself

the **DOM** is the **API**

Teams publish their Custom Elements Documentation
Element-Name, Attributes, Events

Define the tag name of your custom element, specify the attributes that can be used with it, and the events that it triggers/dispatches to the DOM and the event that it listens to on the DOM.

Framework Support



Interoperability Test Suite
for Frameworks


custom-elements-everywhere.com

But Progressive Enhancement?

No **Universal** Web Components

The custom element specification is only concerned with what goes on inside the browser and not for the server-side

Related Products



Server Initial Render

```
/inspire-reco?sku=t_red
```

```
res.send("<h3>Related Products</h3><ul>...")
```

Browser Rehydration & User Interaction

```
<inspire-reco sku="t_red"></inspire-reco>
```

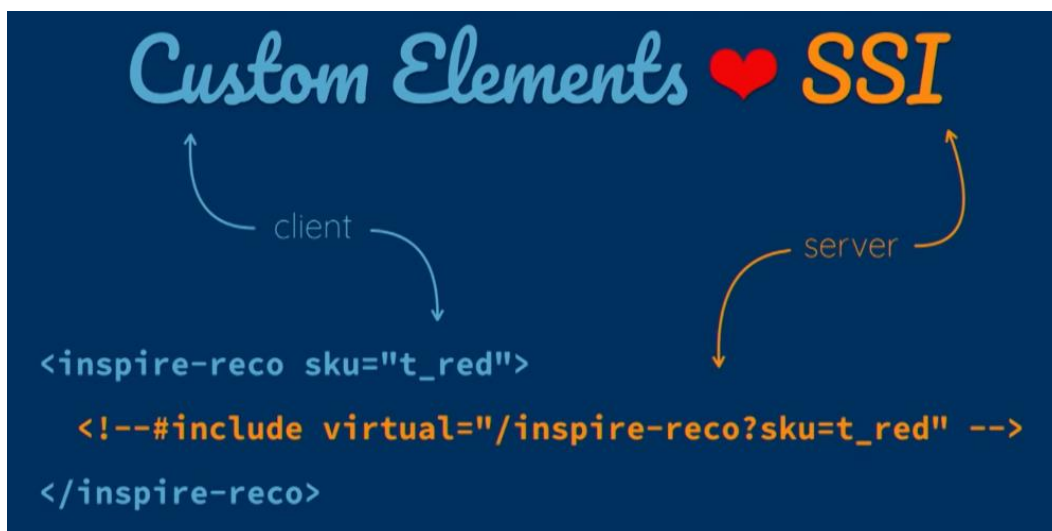
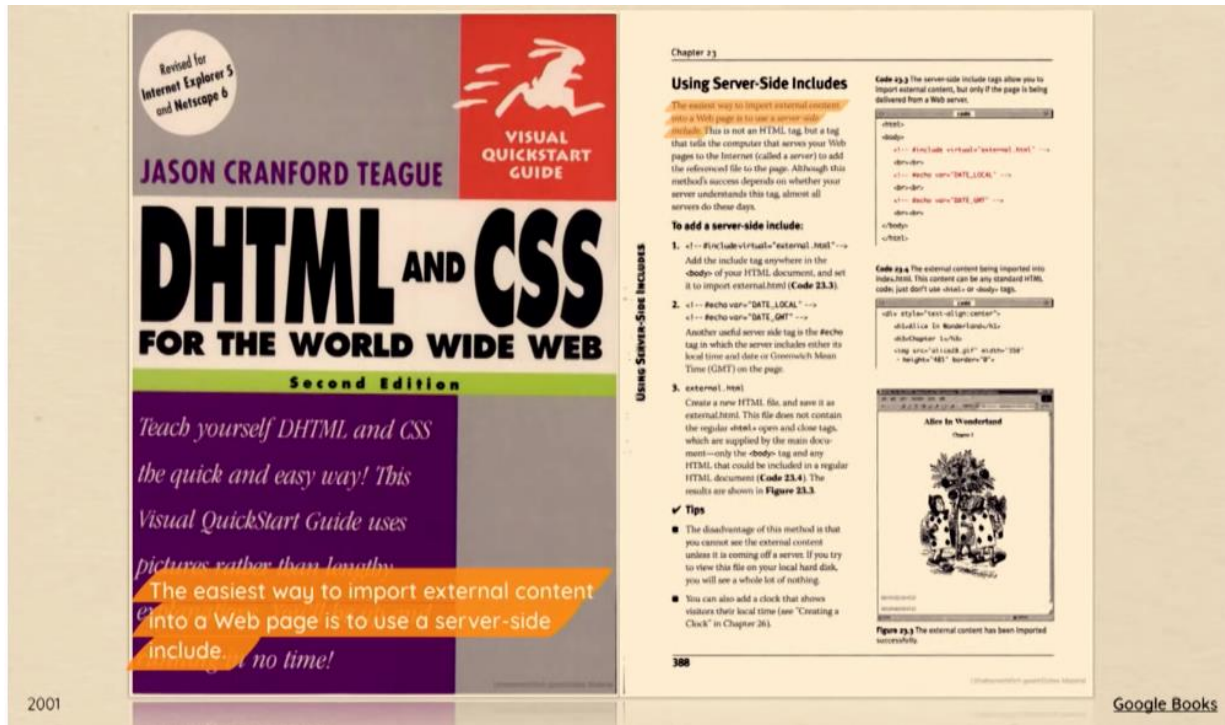
```
el.innerHTML = "<h3>Related Products</h3><ul>...")
```

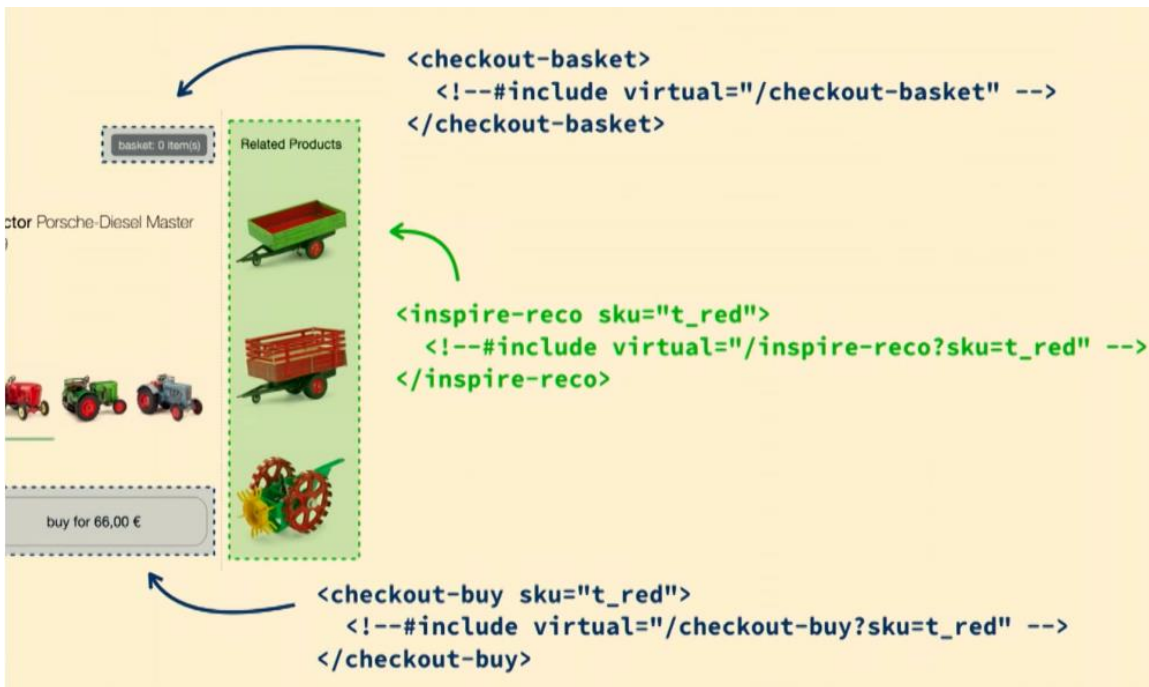
But we can make URLs on the server that delivers some HTML fragments to the browser when called

SSI Server Side Includes

```
<!--#include virtual="/some-url" -->
```

For the initial page load, we need to integrate how to get the fragments and render them out on the first load/initial render using SSI with `<!--#include>` tags to import external HTML into the DOM.



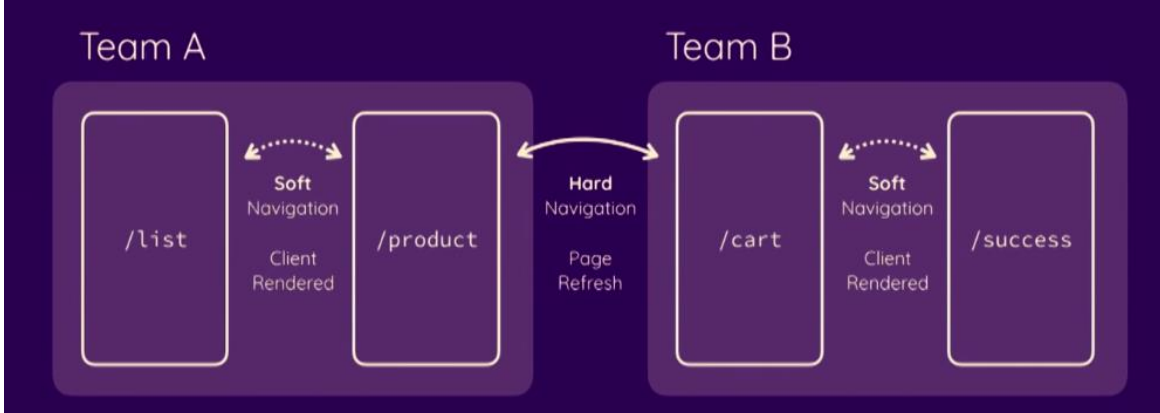


The red team just needs to include the correct custom elements on the page using SSI includes as above

Page Transitions

Only Inside a Team

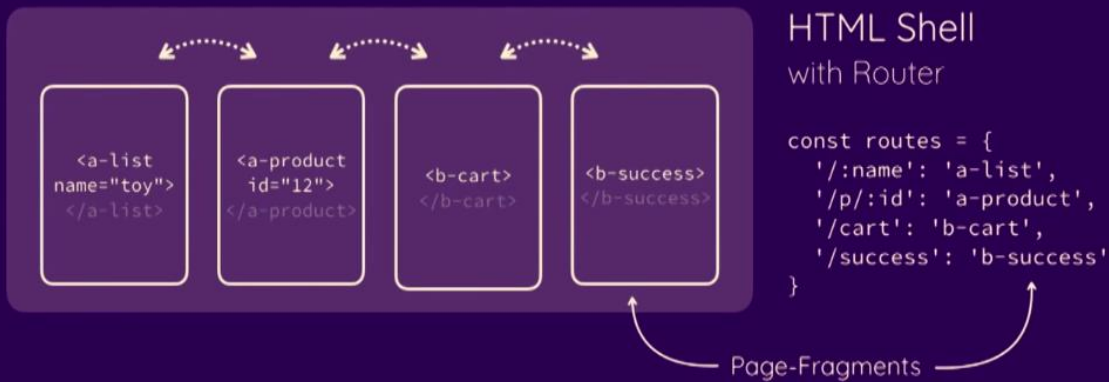
One Router per Team



This is an easy solution where each team delivers a complete HTML per page with head and body tags, the drawback is when you have an increasing number of teams and the hard navigations start increasing with degrading user experience

Top Level Router

Shared Universal Router & Page Fragments



A top-level router requires that each team does not deliver a complete HTML page with head and body tags, they only deliver the snippets of HTML for render in a larger parent component. We will have a common page HTML that is blank and contains client-side and server-side router with a centralized mapping list. The drawback is that we have shared code between all the teams and changing code brings some coupling.

Things we've learned

Use Browser API

don't build a meta framework
avoid shared code

Isolate Teams

don't share runtime, state or globals

Talk to your Neighbors

share best practices

Pack light

register custom elements immediately
download code when needed

Lazy load components or do code splitting to only load components when needed

Measure Performance

HTTP/2 is great, optimize but don't overoptimize

HTTP/2 allows you to lazy load multiple JS files in parallel when needed

Ownership is important

use team prefixes when needed

Always prefix each team's JS, CSS classes as well as prefix all the events with the team name.

Have a Design System

build a [Style Guide](#) or [Pattern Library](#) everybody can use

Thanks for Listening

micro-frontends.org @naltatis