Micro Frontends

Sven Kölpin - open knowledge GmbH
Lars Kölpin - open knowledge GmbH

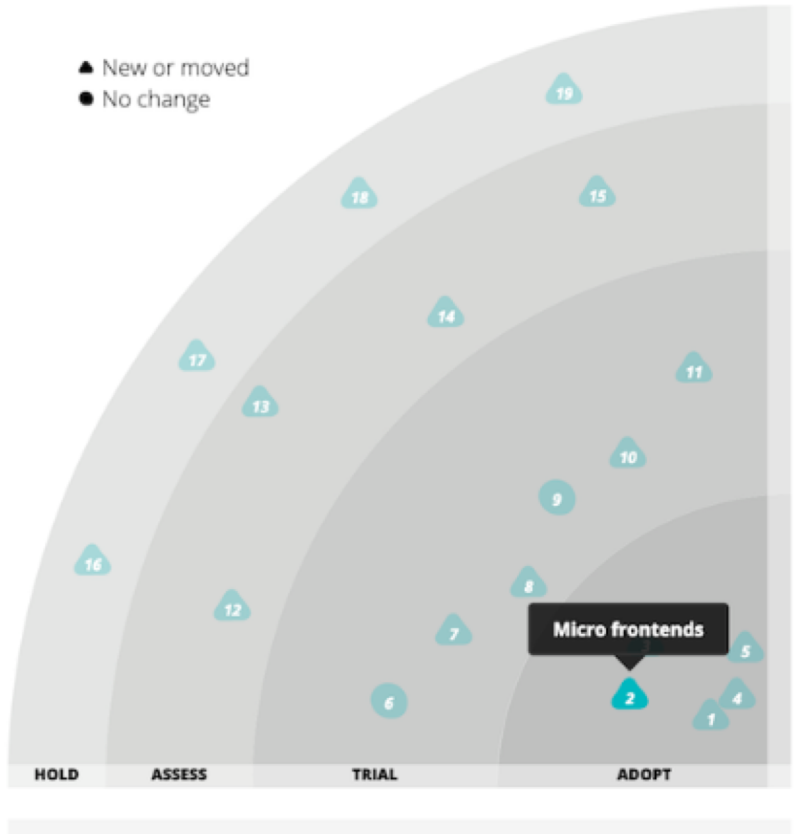
@dskgry @LarsKoelpin @_openknowledge #wissenteilen

ADOPT

1. Four key metrics

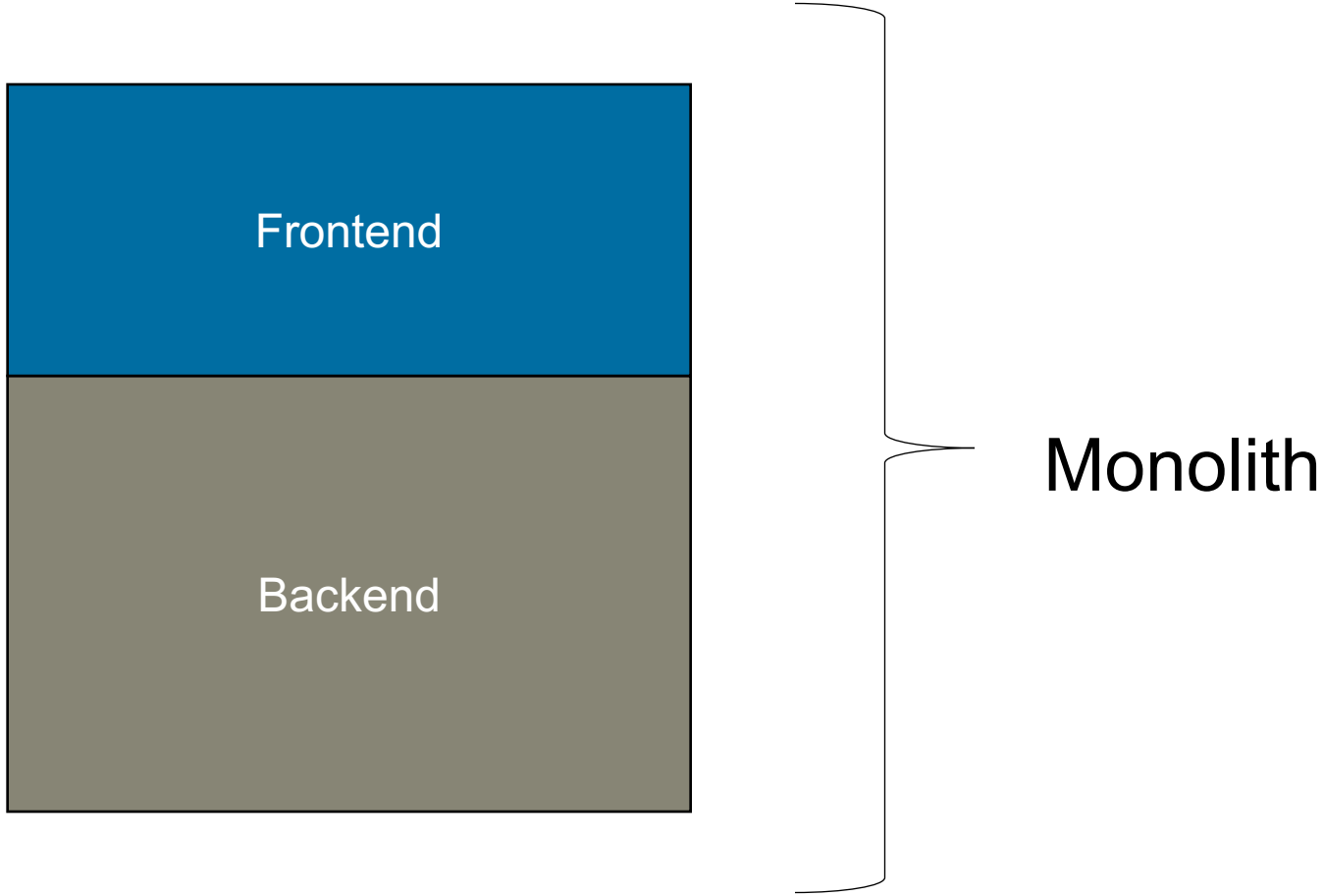
2. Micro frontends

We've seen significant benefits from introducing microservices, which have allowed teams to scale the delivery of independently deployed and maintained services. Unfortunately, we've also seen many teams create a frontend monolith — a large, entangled browser application that sits on top of the backend services — largely neutralizing the benefits of microservices. Since we first described **micro frontends** as a technique to address this issue, we've had almost universally positive experiences with the approach and have found a number of patterns to use micro frontends even as more and more code shifts from the server to the web browser. So far, web components have been elusive in this field, though.

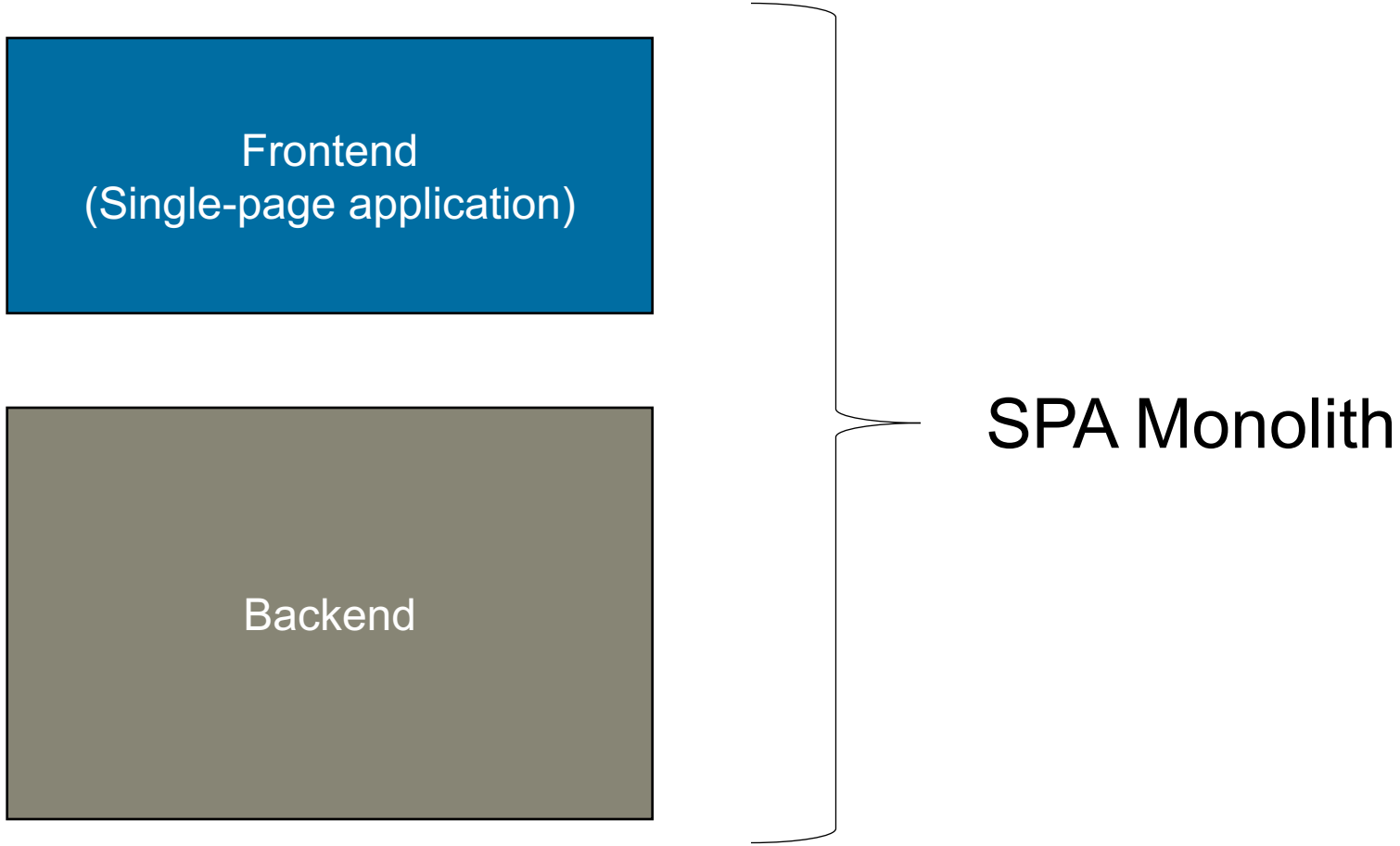


Micro Frontends: Conceptual level

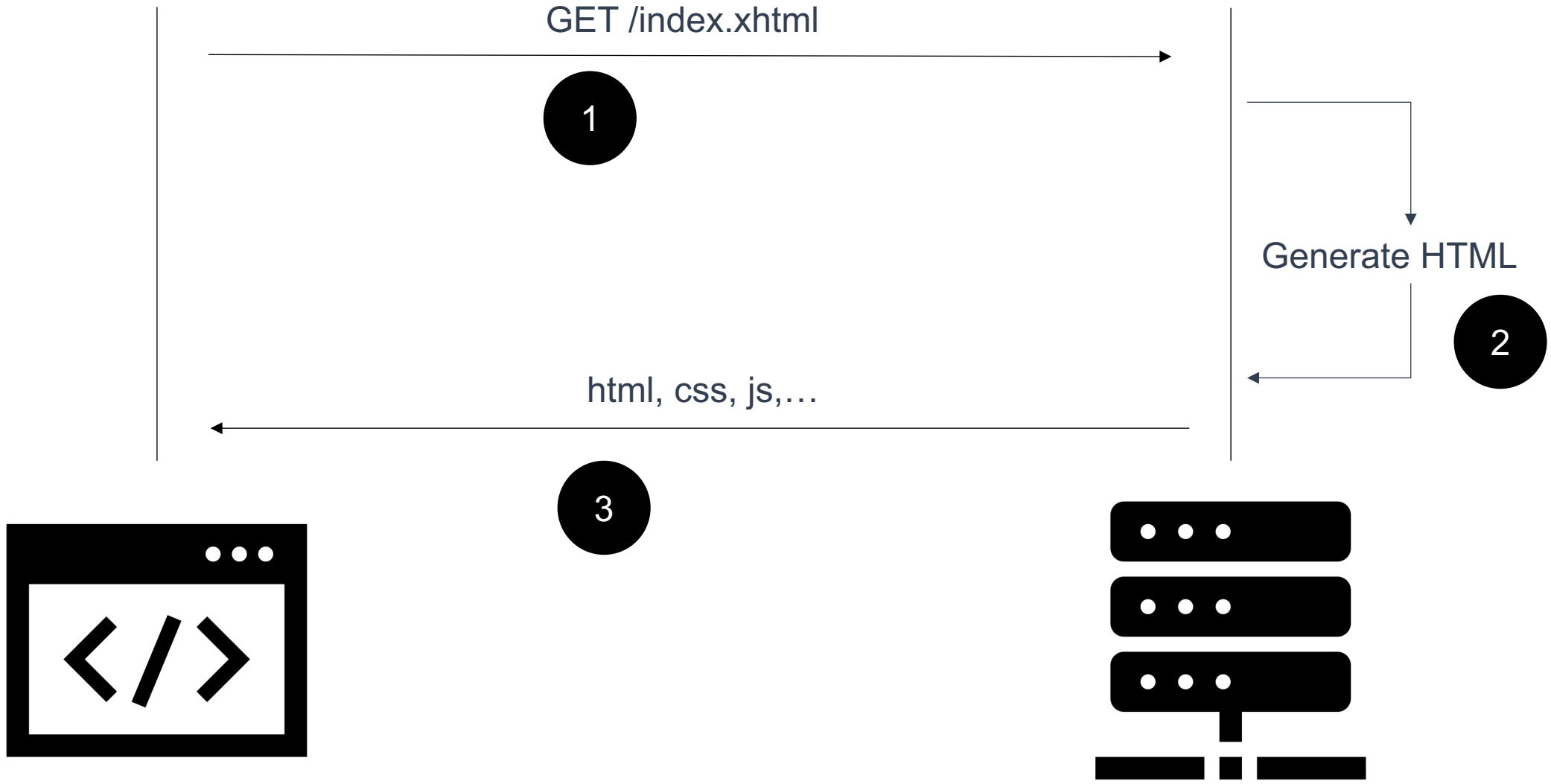
Web Architecture Evolution



Web Architecture Evolution



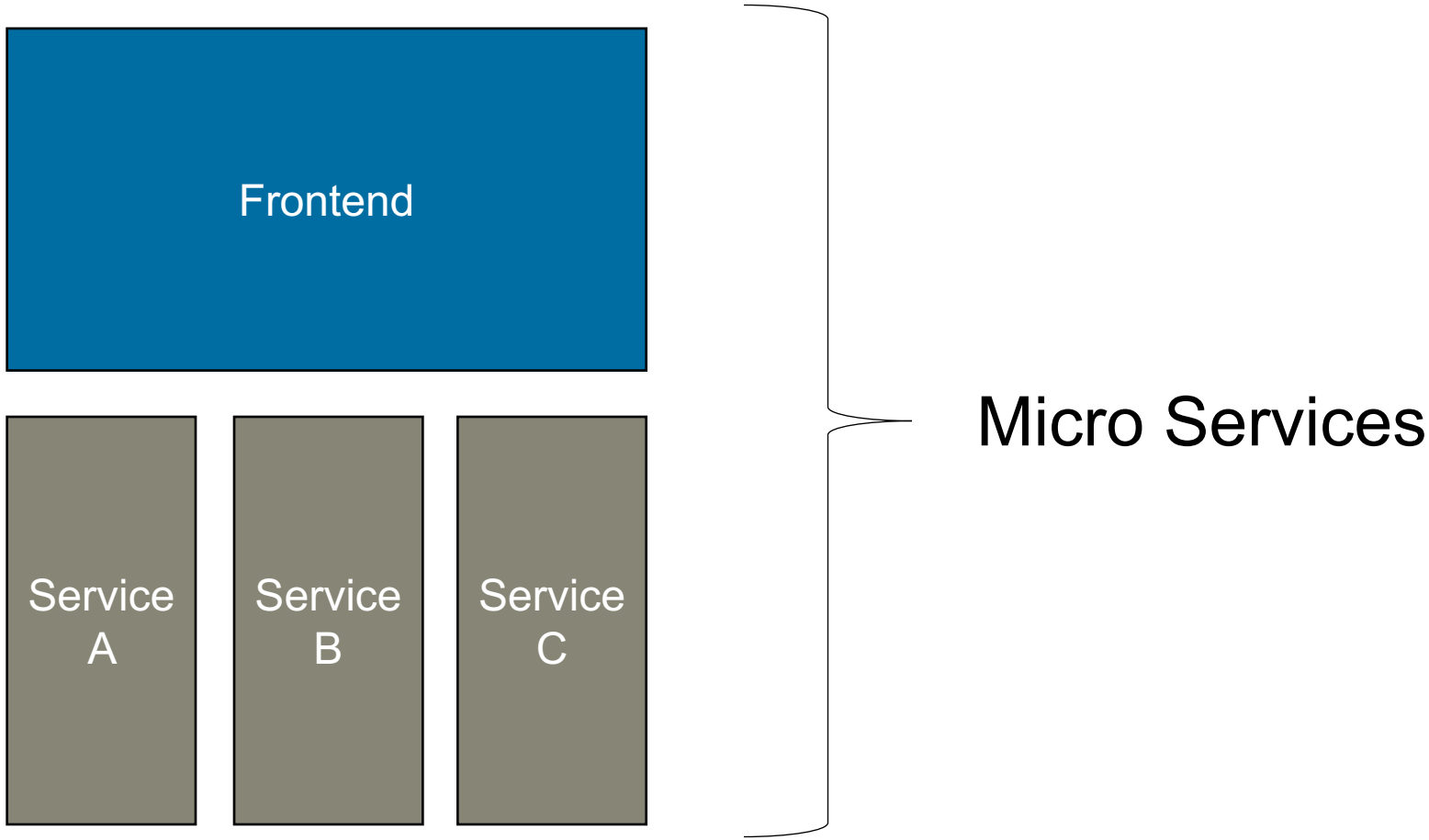
Server-side rendering



Client-side rendering

Demo Time: 00

(Web) Architecture Evolution



Frontend

Service A

Service B

Service C

Autonomous Teams

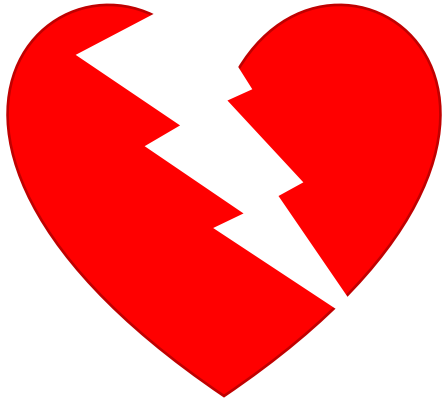
Independent Deployments

Scalability

Flexibility

Frontend

?



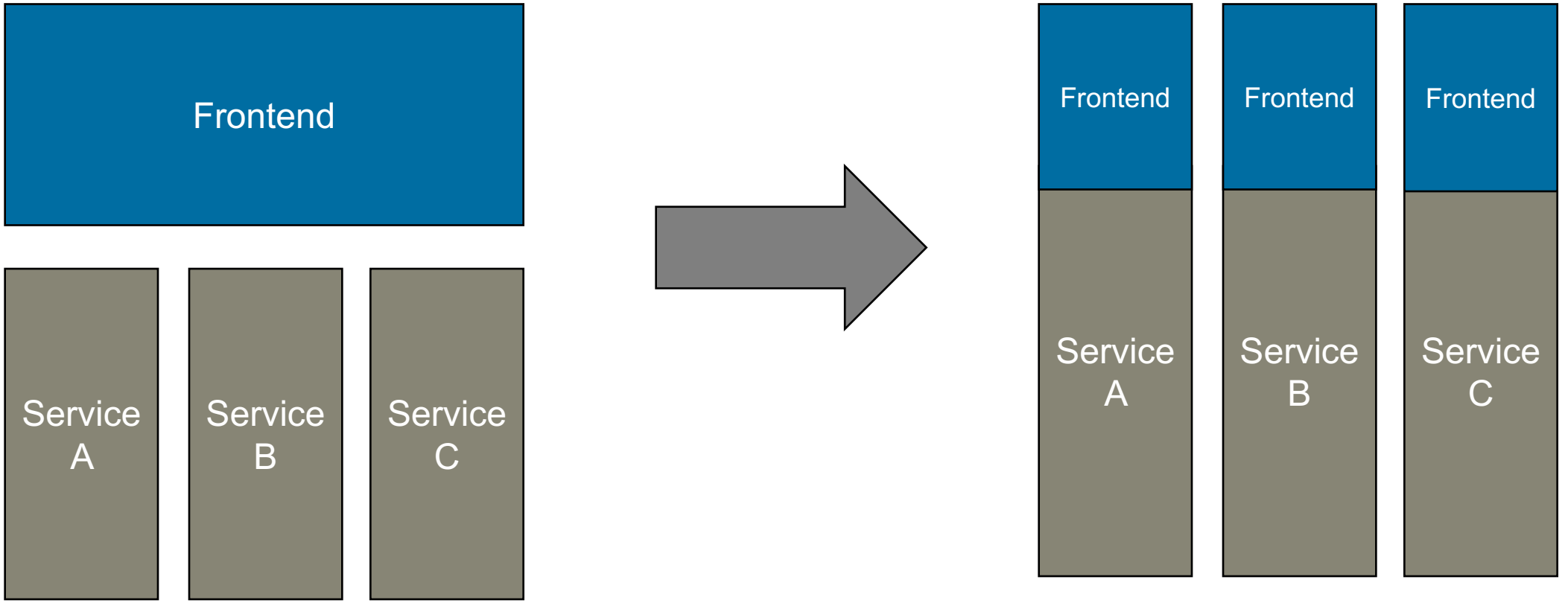
Service A

Service B

Service C

Micro Frontends

„Bringing the idea of Micro Services to the frontend“

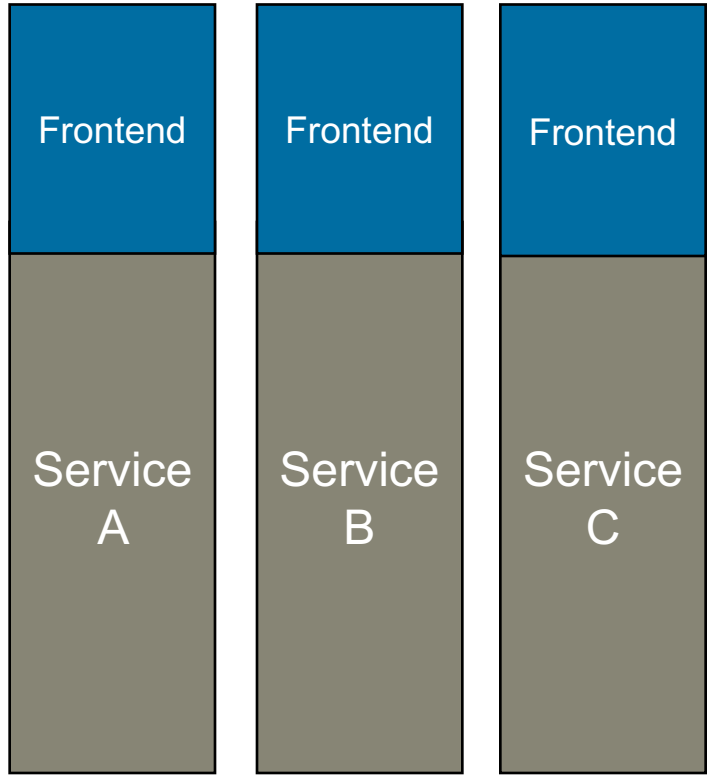

Frontend
Service A
Service B
Service C
Frontend
Frontend
Frontend
Service A
Service B
Service C

Independently deliverable

End-to-End

Local Decisions

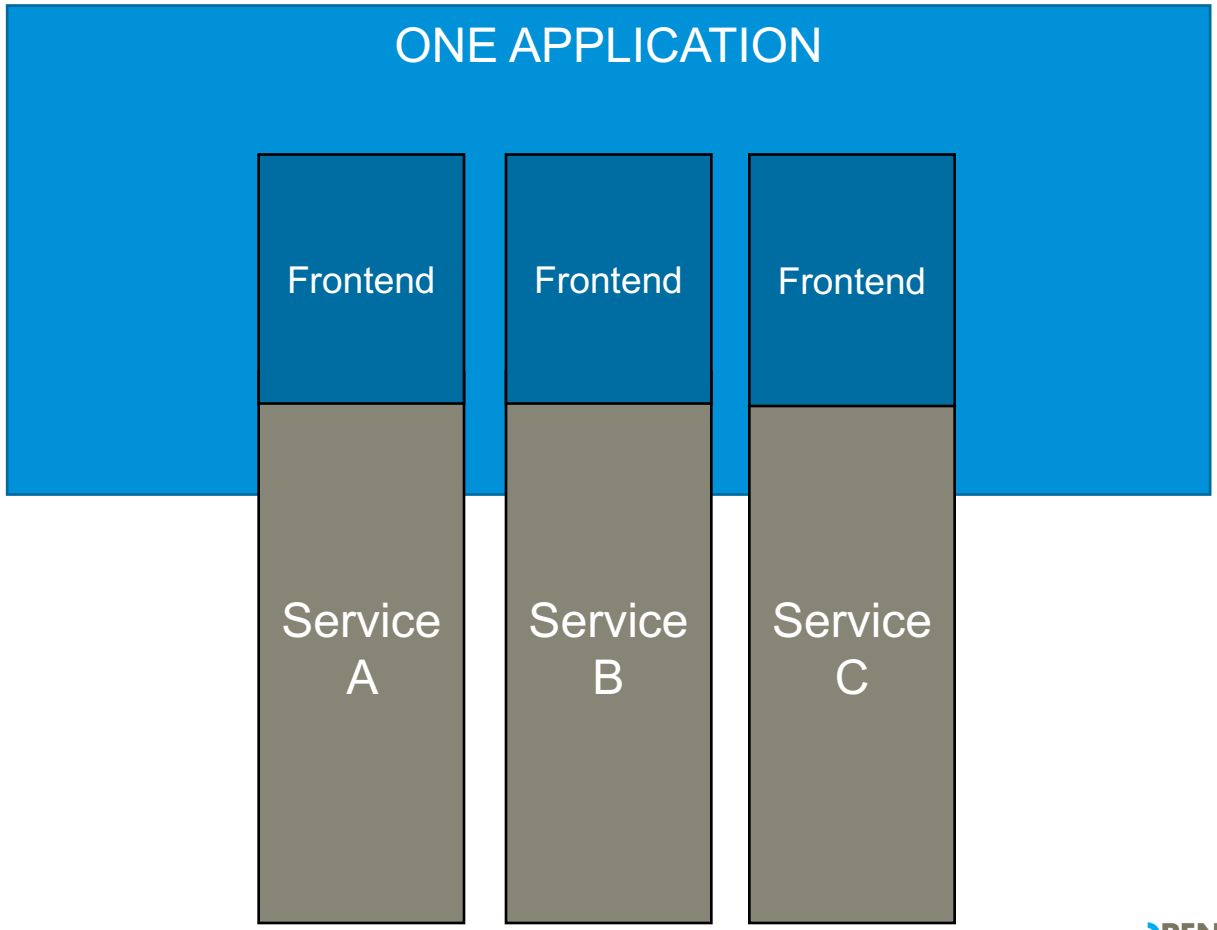
Better products

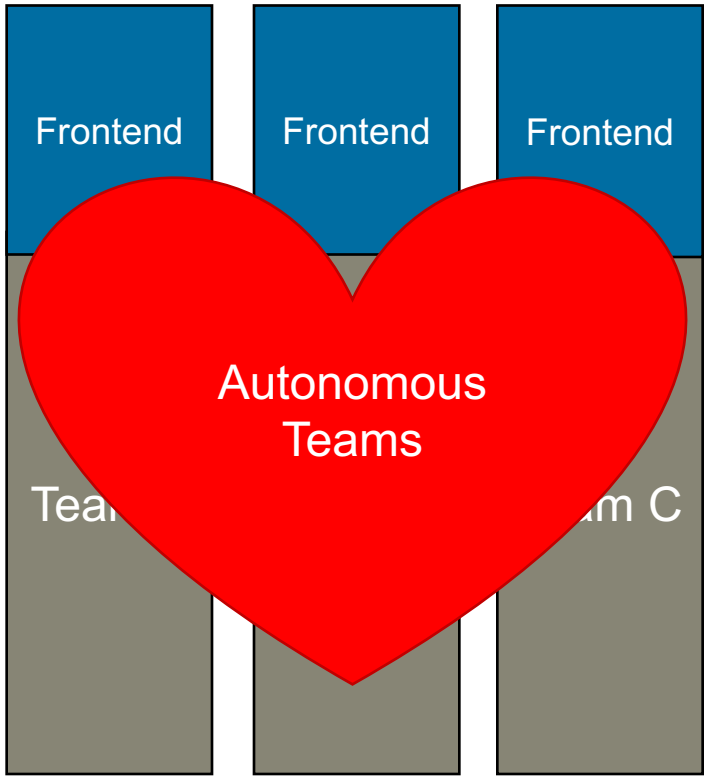


Composition

UI === Composition

How to compose?





Focus on products

Cross functional teams

Technology agnostic

How Micro?

DDD Subdomains

Core

Supporting

Generic

Subdomain === Micro Frontend

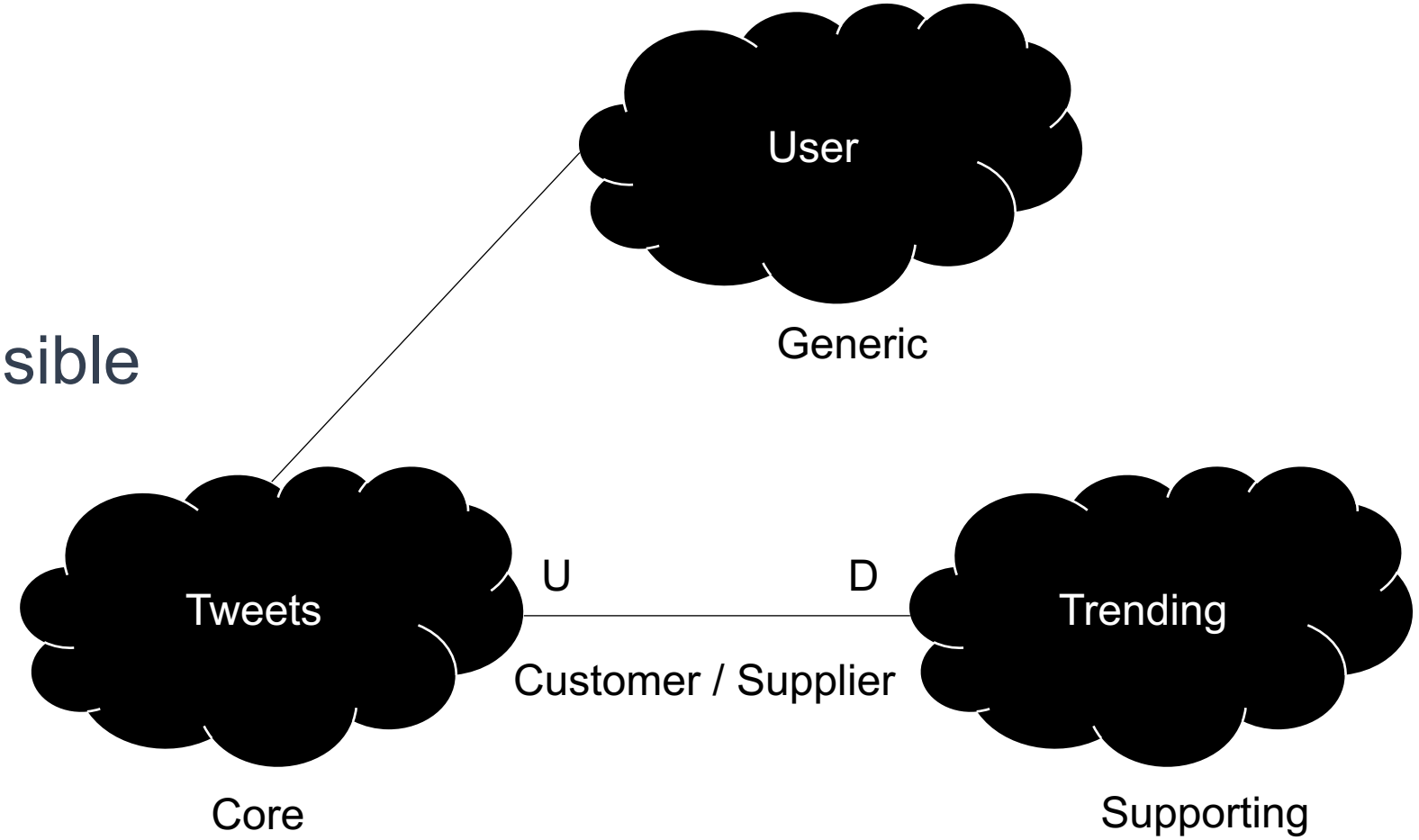
- Micro Frontend !== UI Component
- Often **match** with **user behavior**
- Little dependencies

Dependencies

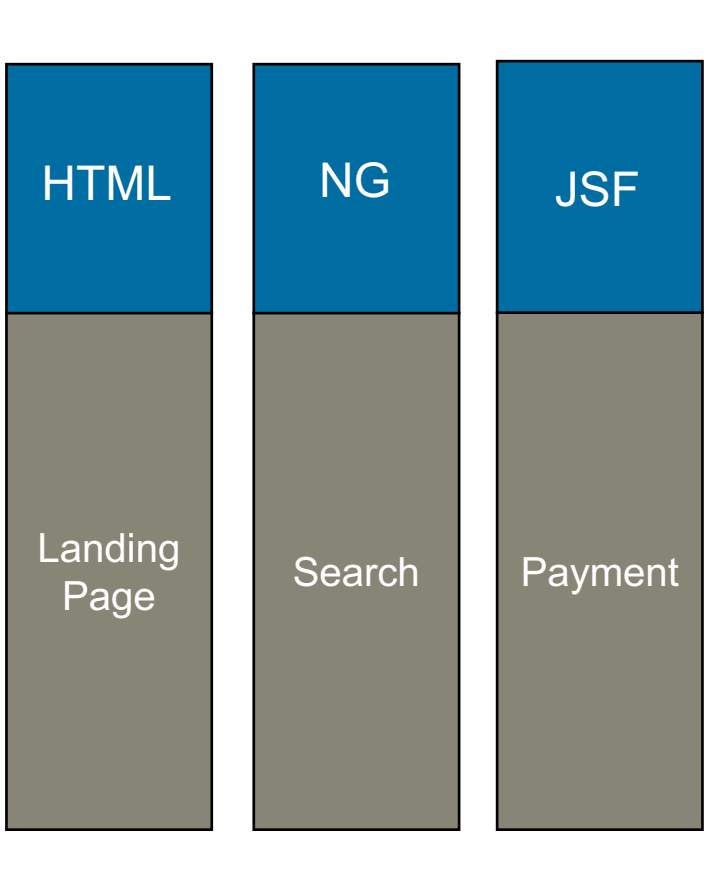
As **Self-Contained** as possible

Explicit dependencies

Define Contracts



Product-based technology decisions



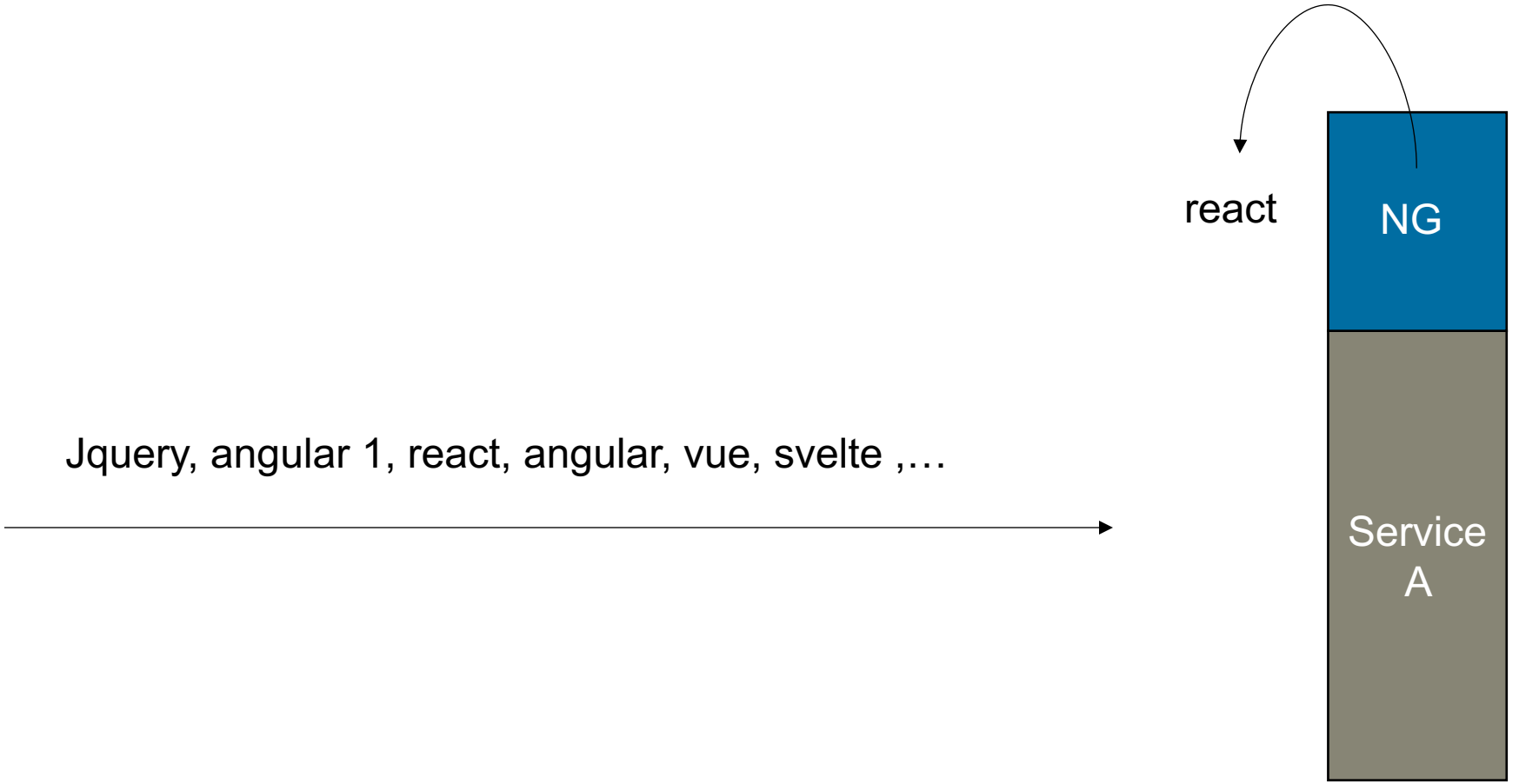
What fits best to product?

Team skills

Legacy integration

Future proof development

Jquery, angular 1, react, angular, vue, svelte ,…



Autonomous !== Isolation

Fully **autonomous** teams

- Knowledge sharing
- Verification of decisions

How to compose?

No Composition

Server Side Composition

Client Side Composition

Mixture of all

… more to come soon

How to integrate?

No direct communication

No shared data

Achieve **data consistency**

High coupling means wrong domains

How to achieve UX consistency?

Independent !== Isolated

Micro Frontends != Framework wild west

Shared code / libs?

Global rules?

What about rules?

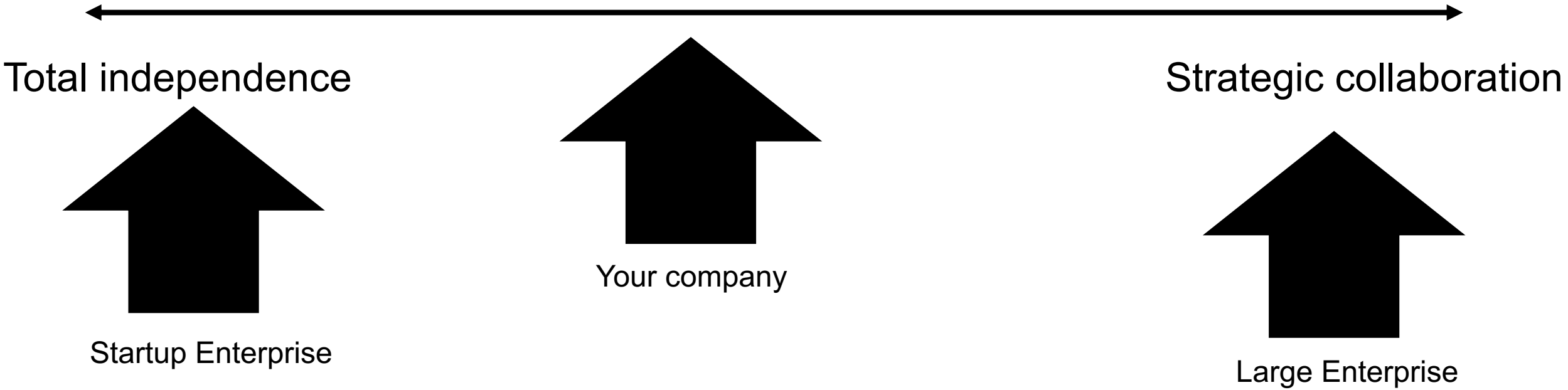
38 Frameworks
Everything looks different

Centralized decisions
→ OOOPS monolith

Total independence

Strategic collaboration

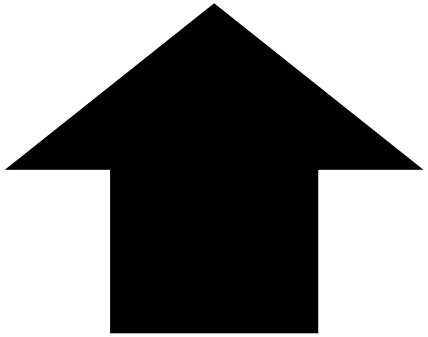
Find the right **balance** for your company **culture**



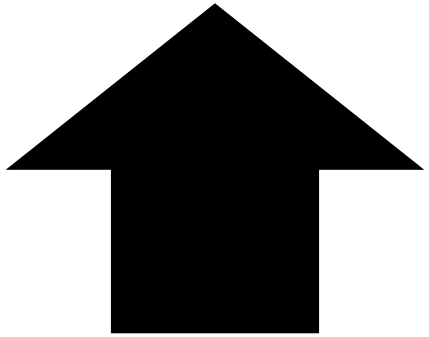
Find the right **balance** for your product

Total independence

Strategic collaboration



Internal Apps



Public Apps

How to share?

Re-use is **not essential**

Who is responsible?

How fast can things be changed?

Which technology?

At most share **primitives**

„This does not scale“

„How can this be consistent?“

„This is nothing new!“

„This is way too complex!“

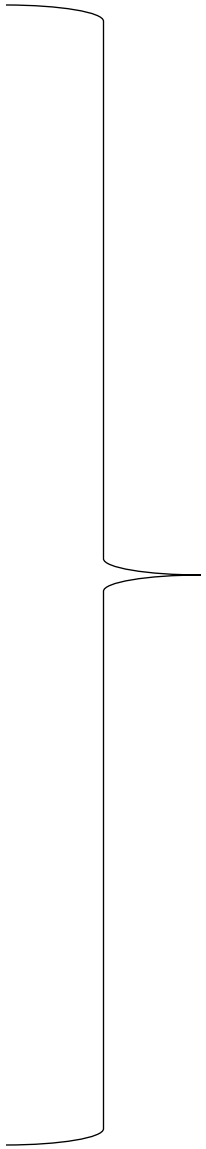
IKEA

DAZN

ZALANDO

SPOTIFY

…



Large organizations

Distributed teams

Massive growth

Wrap up: Conceptual level

Independent deployments is the key

Should you use Micro Frontends?

- Do we have to scale N people working on the same product?
- Problems with frontend techn. stack?

Decisions: contracts & sharing

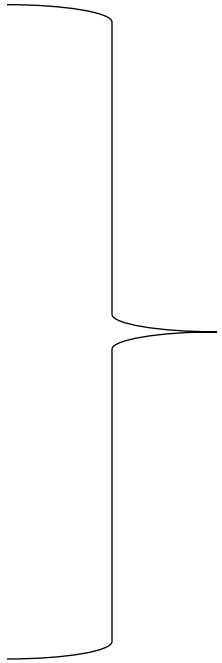
Micro Frontends: Implementation level

Challenges

How to **compose**

How to **integrate**

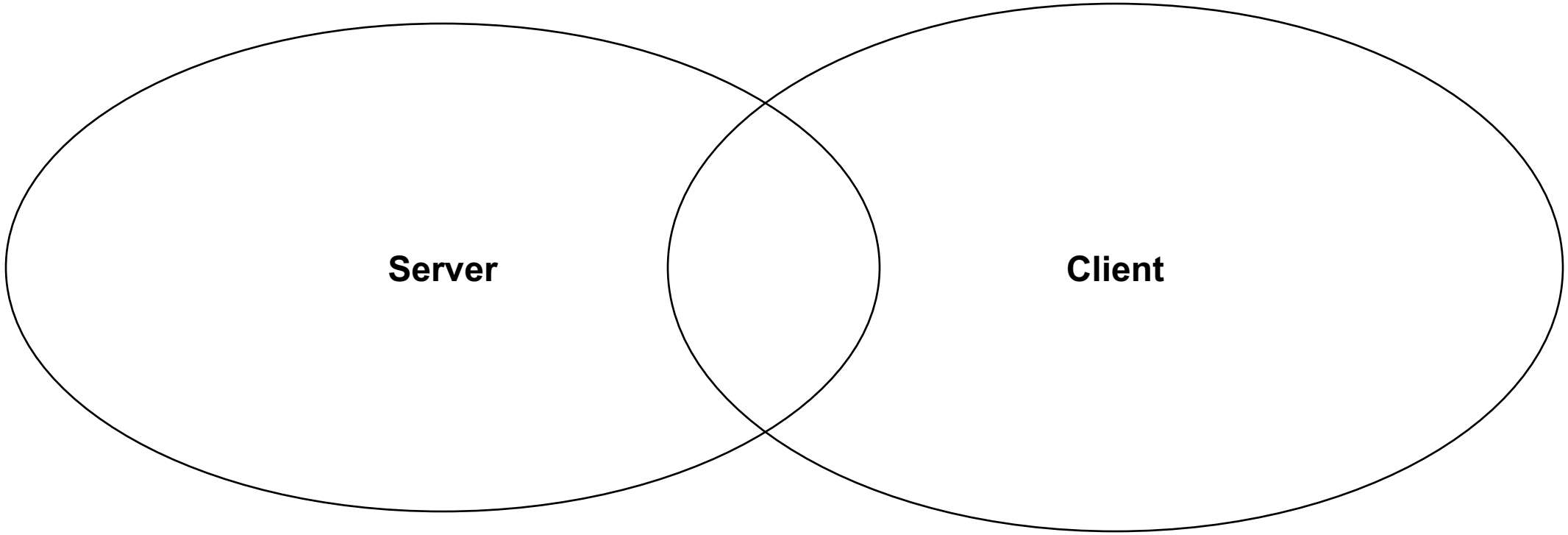
How to **share**



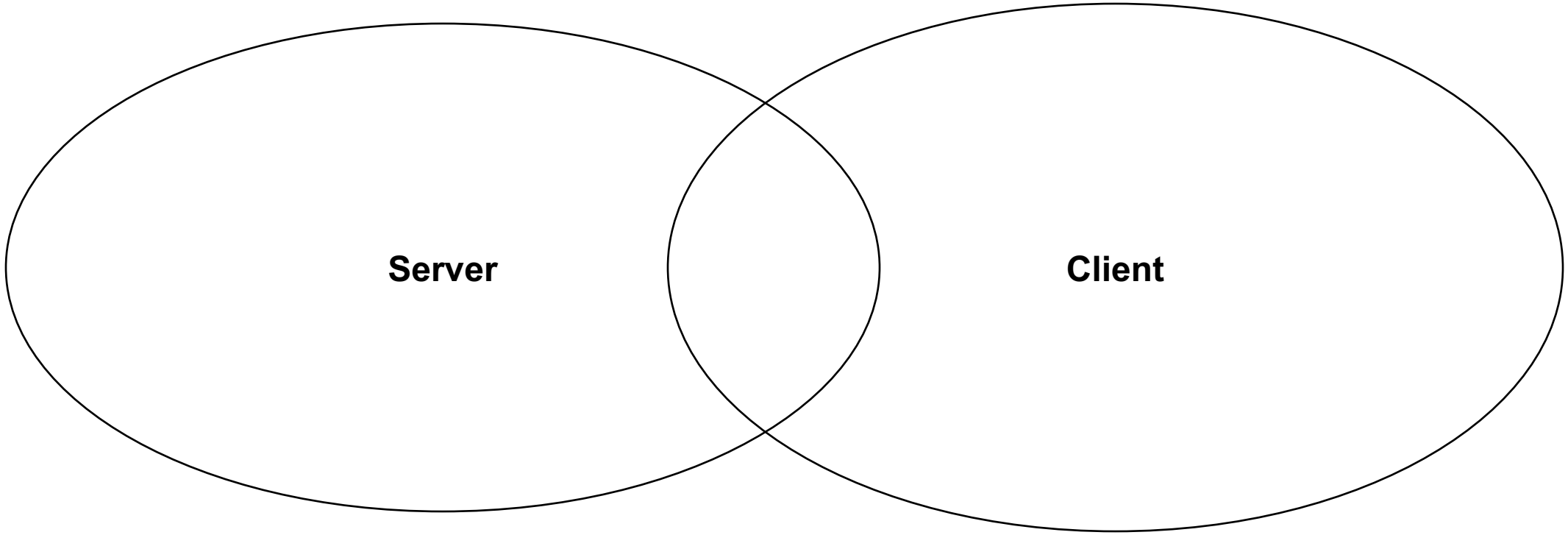
The Web

There is no single right way!

Where to compose?

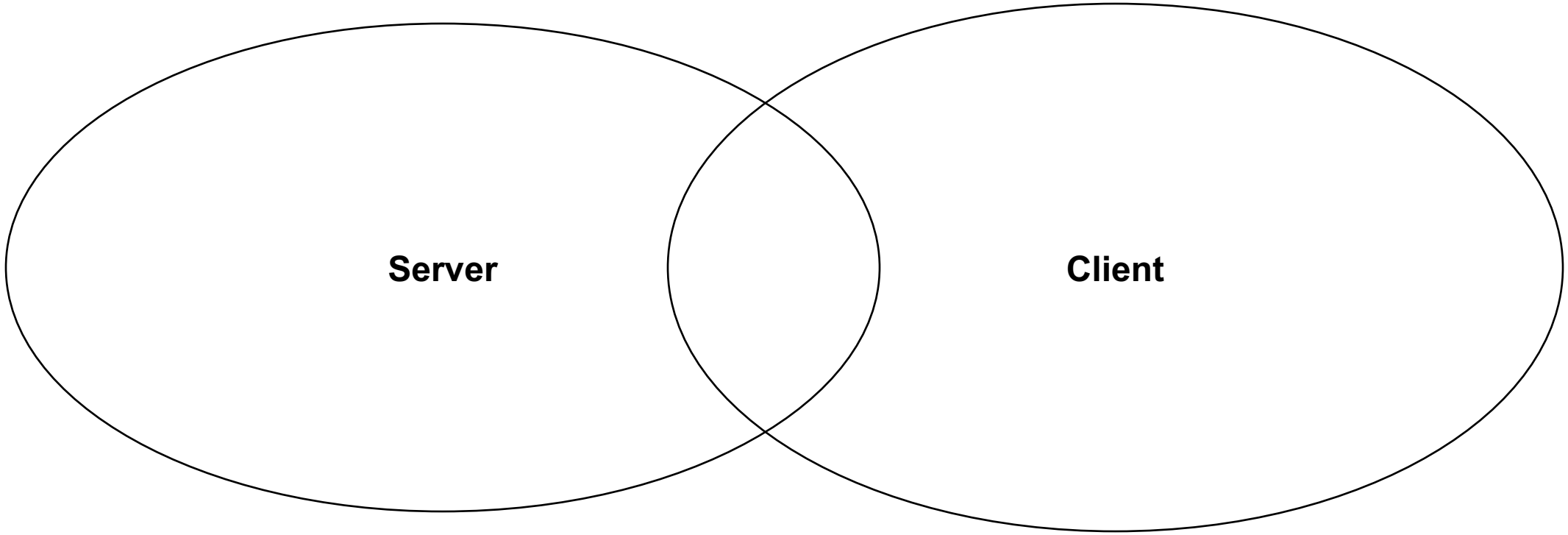


Where to compose?



… at build time!?

Where to compose?



… or not at all

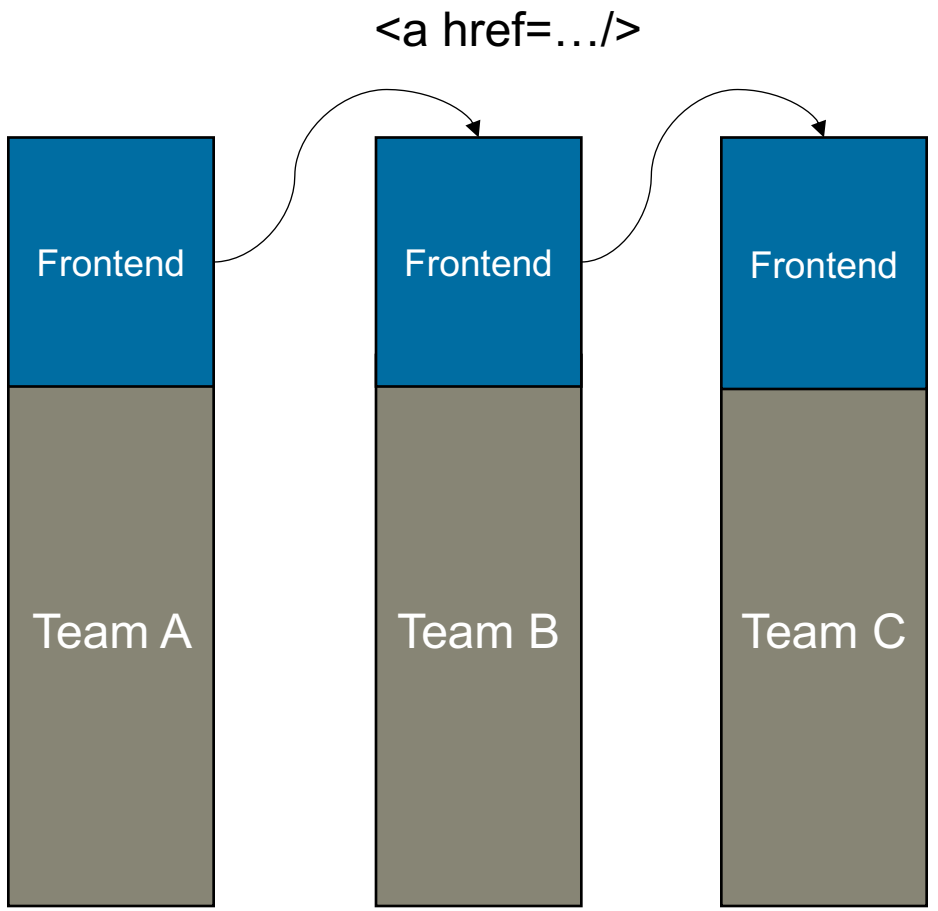
No Composition

App per **route**

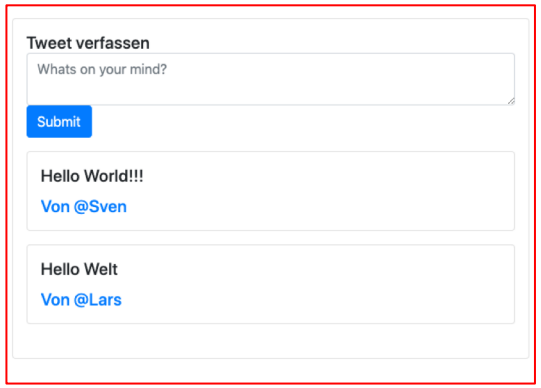
Integration via **links**

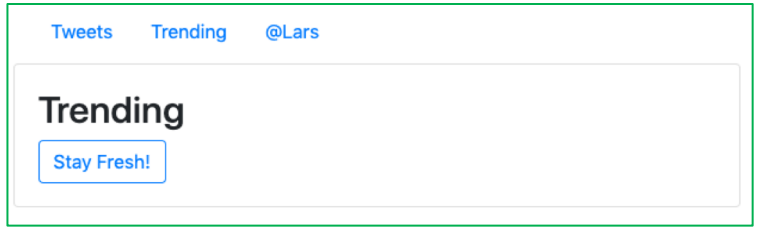
Complete **isolation**

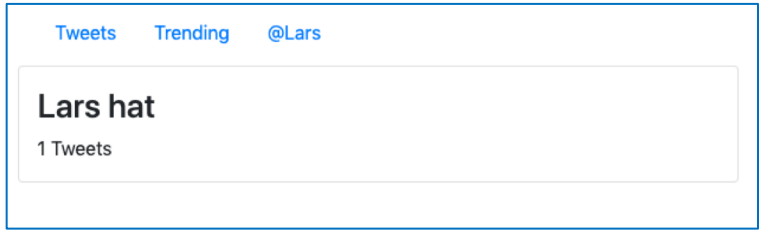
SSR or **CSR**

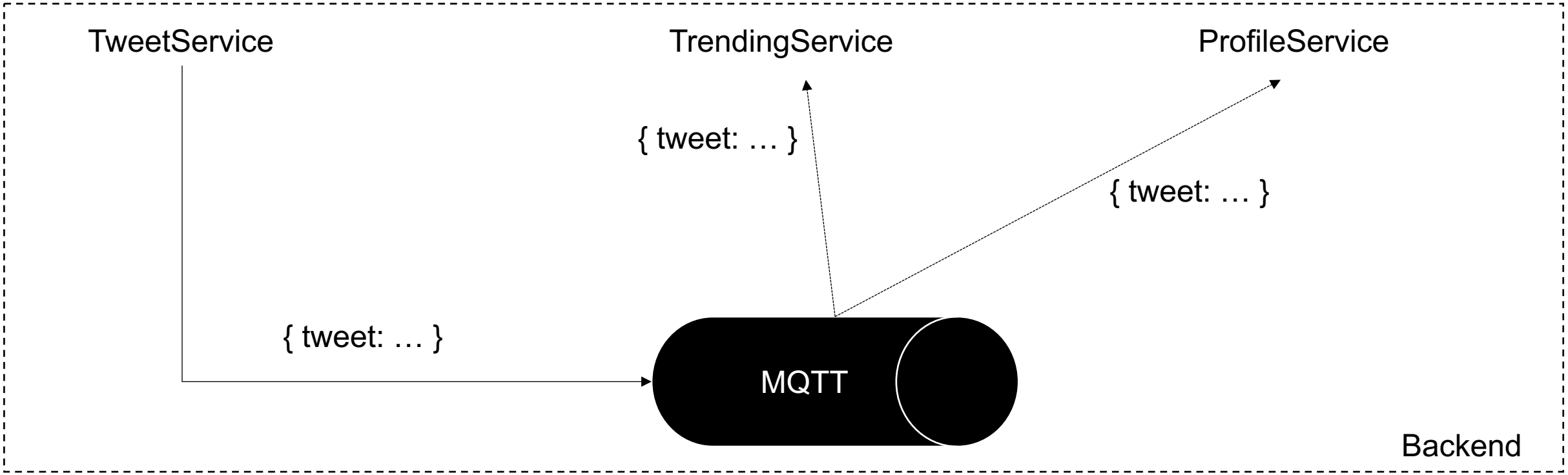


Demo Time: 01a


Tweet verfassen
Whats on your mind?
Submit
Hello World!!!
Von @Sven
Hello Welt
Von @Lars


Tweets
Trending
@Lars
Trending
Stay Fresh!


Tweets
Trending
@Lars
Lars hat
1 Tweets


TweetService
TrendingService
ProfileService
{ tweet: … }
{ tweet: … }
{ tweet: … }
MQTT
Backend

No Composition

Works well when…

(almost) **no shared state**

Independent products

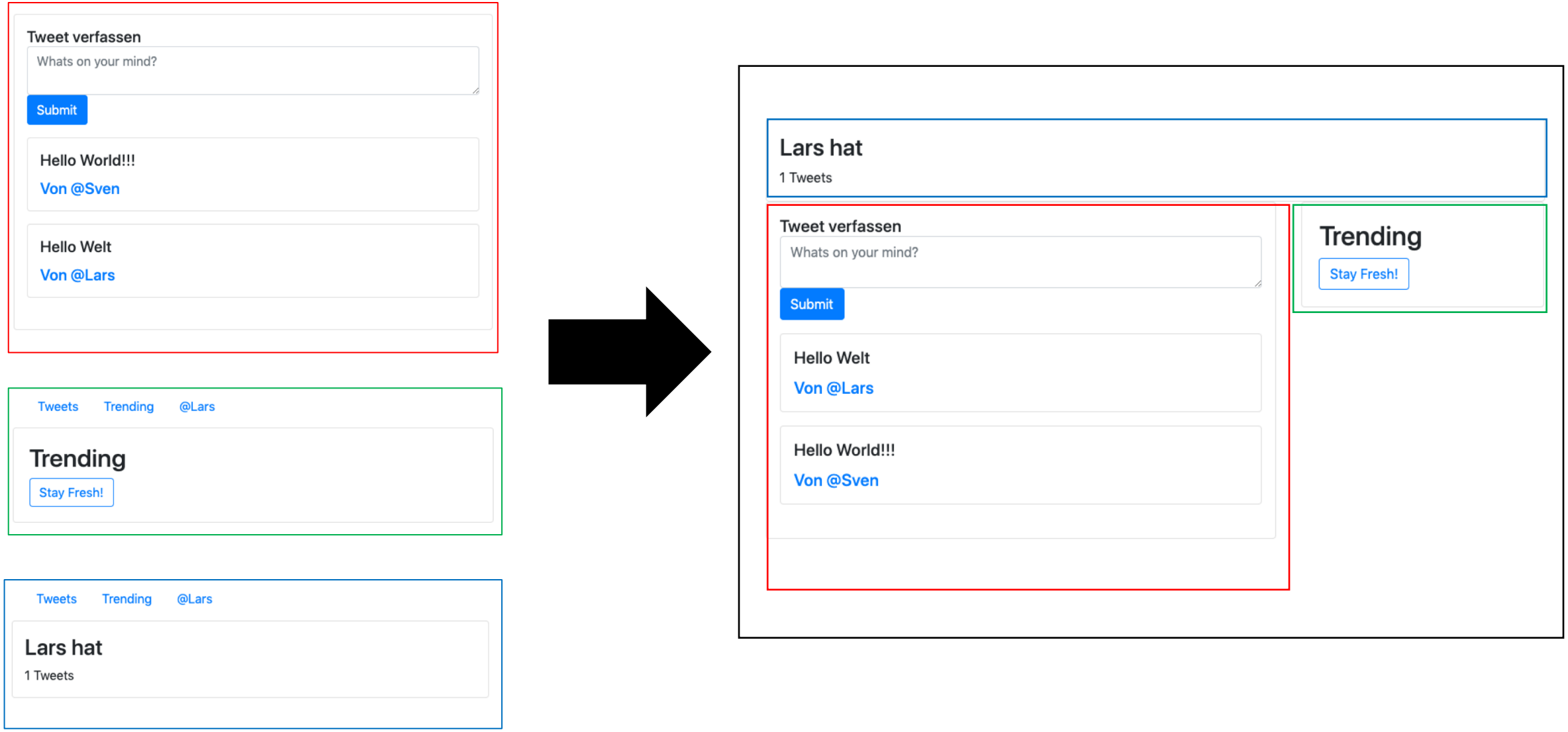
High **isolation** needed

Say hello to portals <3

Seamless page **transitions**

Experimental…

Demo Time: 01b


Tweet verfassen
Whats on your mind?
Submit
Hello World!!!
Von @Sven
Hello Welt
Von @Lars
Tweets
Trending
@Lars
Trending
Stay Fresh!
Tweets
Trending
@Lars
Lars hat
1 Tweets
Lars hat
1 Tweets
Tweet verfassen
Whats on your mind?
Submit
Hello Welt
Von @Lars
Hello World!!!
Von @Sven
Trending
Stay Fresh!

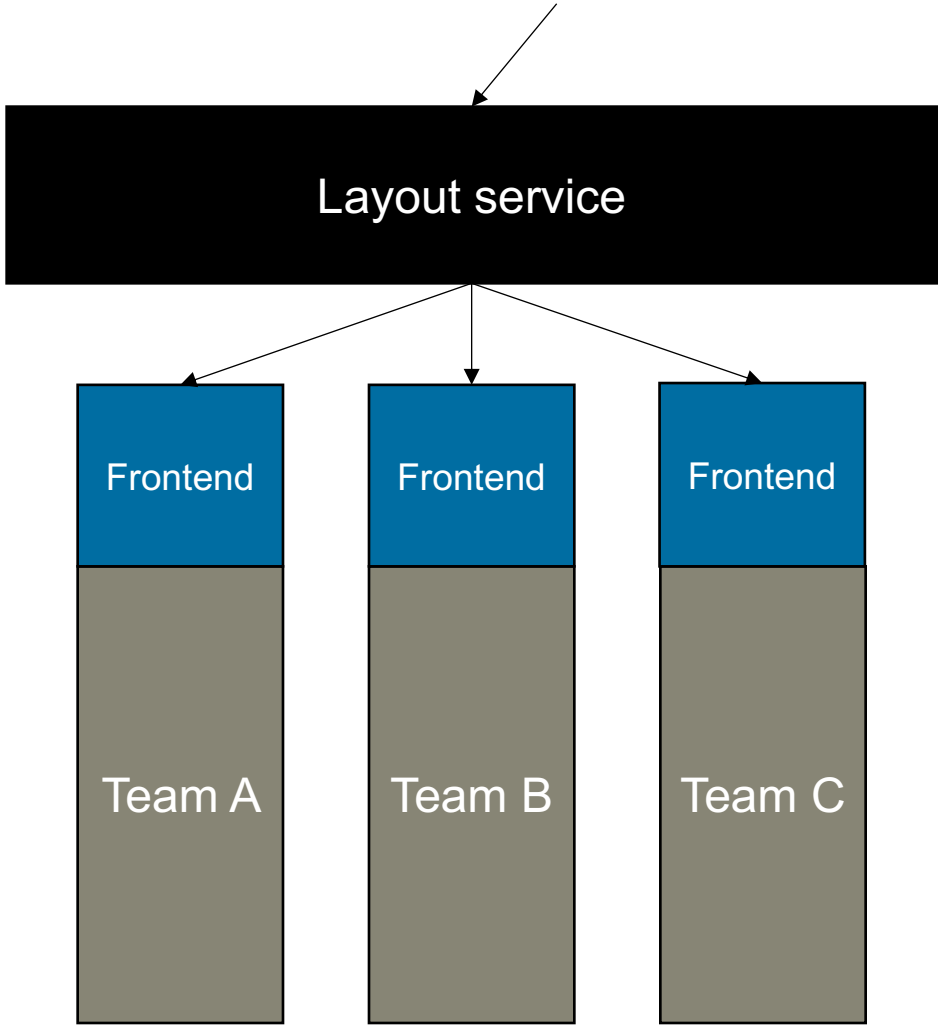
Server-Side Composition

Integration via **Layout service**

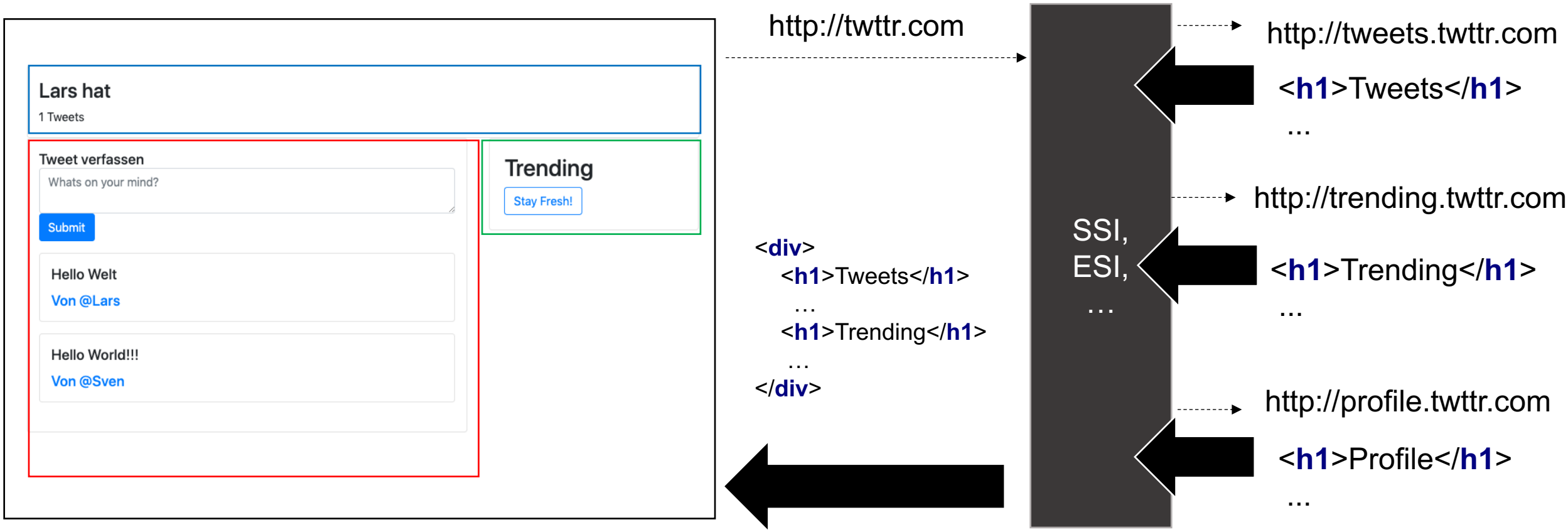
One app for the user

Extra service vs. **Part of** service

SSR & CSR



Server-Side Composition



Server-Side Includes (SSI)

Server-side **transclusion**

Good support

- **ngnix**, apache, IIS, …

Use Cases

- Re-use code on server side
- Simple dynamic content creation

```
<html>
…
<!--#include virtual="menu.cgi" -->

<!--#include file="footer.html" -->

<!--#exec cgi="/cgi-bin/foo.cgi" -->
…
</html>
```

Edge-Side Includes (ESI)

Declarative server-side **transclusion**

More **features** than SSI

- Error instructions
- Alternatives

Focused on **caching**

- CDN-Level

```
<html>
…
<esi:include
    src="http://tweets.trending.com"
    alt="http://tweets.ads.com"
    onerror="continue"/>
…
</html>
```

Demo Time: 02

Tailor

Streaming **layout** service

Project Mosaic

Parallel requests & response streaming

Resilience

Can also be used for **CSR** Apps

Tailor

Node.js based

Simple, powerful API

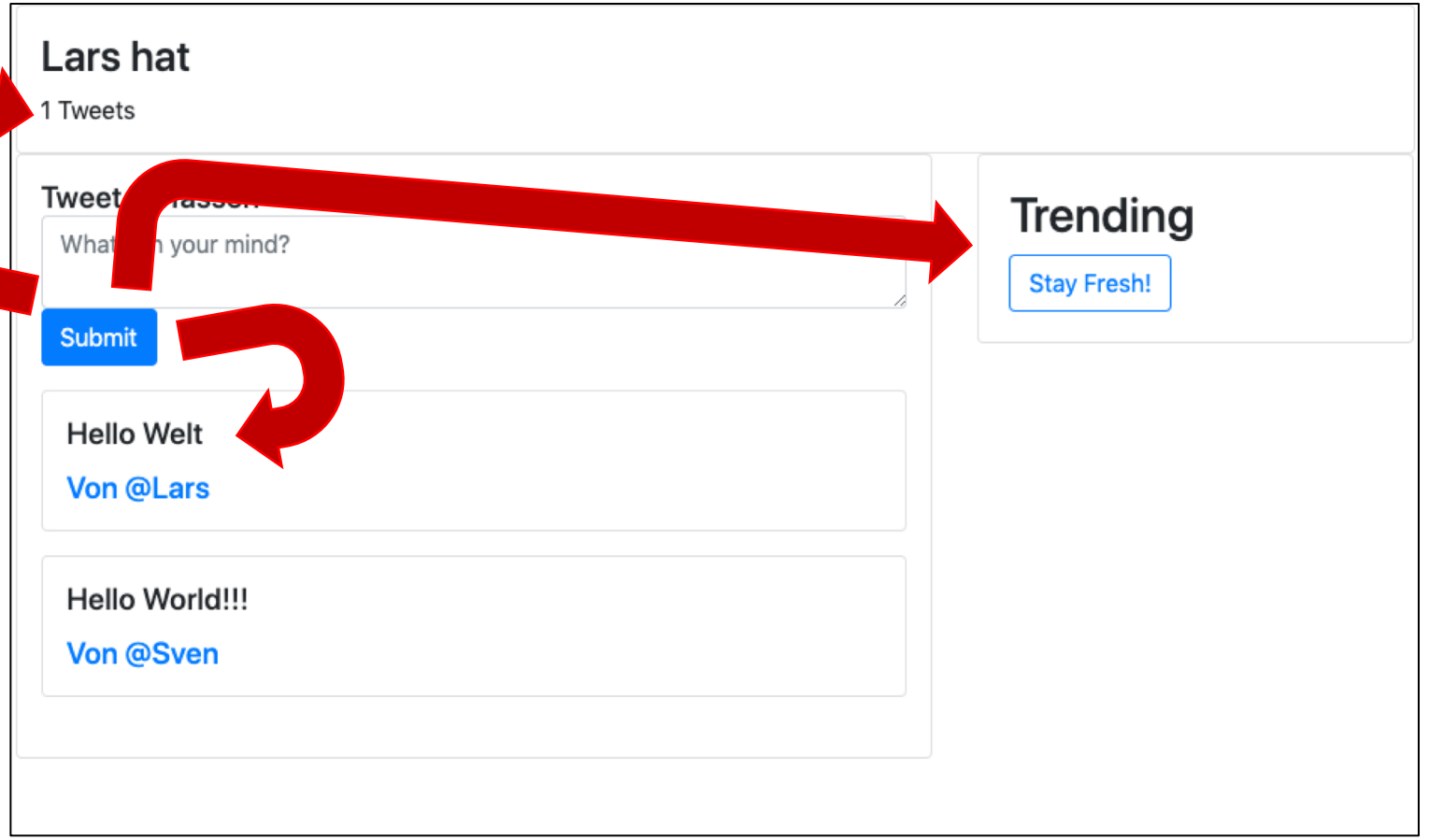
No built-in caching

```
<html>
<body>
    <fragment src="http://header.domain.com"></fragment>
    <fragment src="http://content.domain.com" primary></fragment>
    <fragment src="http://footer.domain.com" async></fragment>
</body>
</html>
```

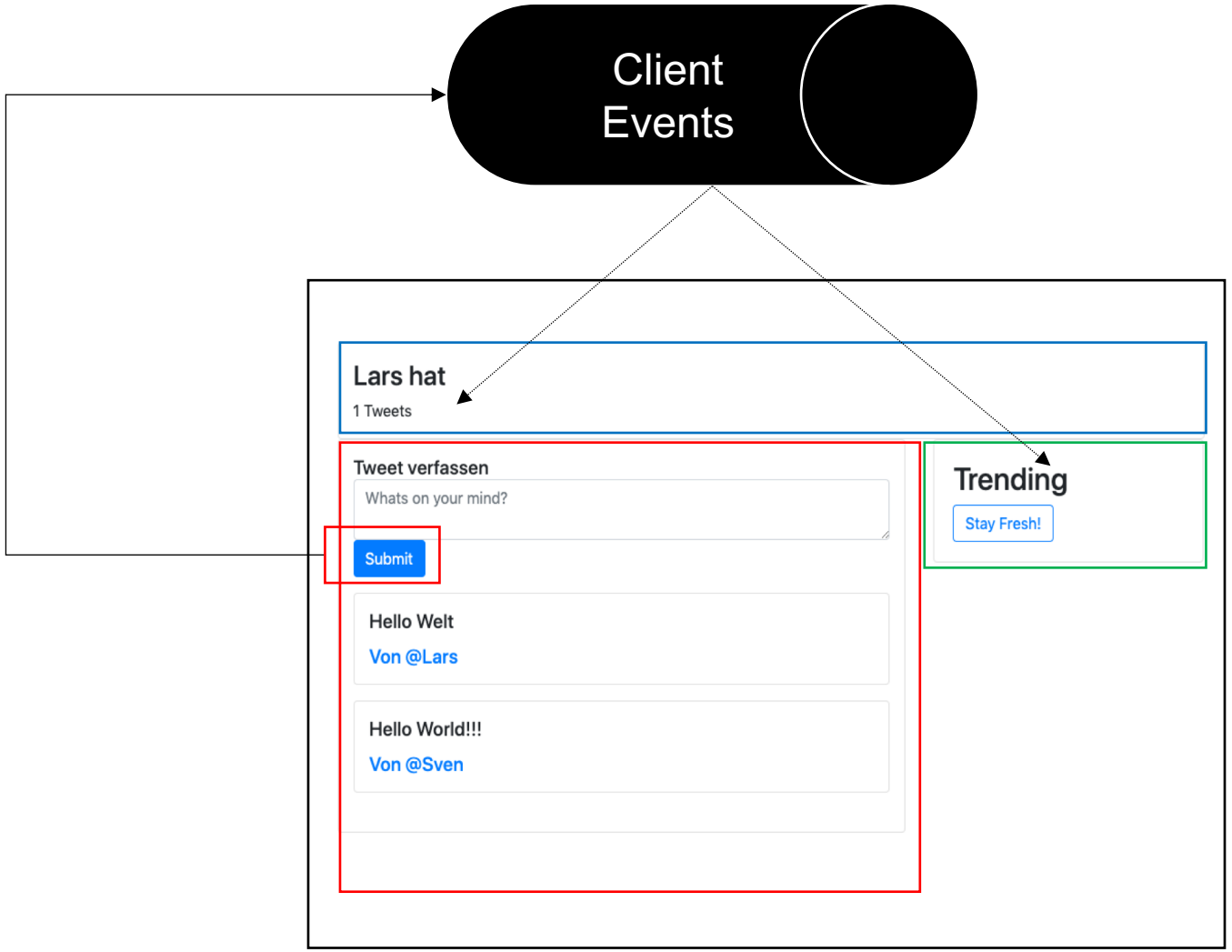
Will be **replaced** by Interface Framework

Demo Time: 03

Integration



Events

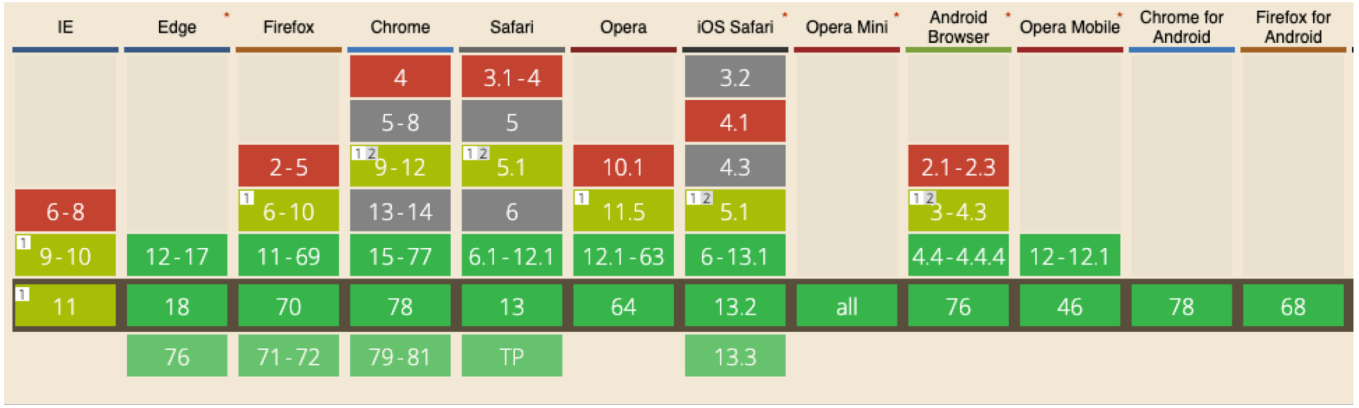


Loose coupling

Contracts!

Events How?

Browser has event system



IE	Edge	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Opera Mobile	Chrome for Android	Firefox for Android
			4	3.1-4		3.2					
			5-8	5		4.1					
		2-5	9-12	5.1	10.1	4.3		2.1-2.3			
6-8		6-10	13-14	6	11.5	5.1		3-4.3			
9-10	12-17	11-69	15-77	6.1-12.1	12.1-63	6-13.1		4.4-4.4.4	12-12.1		
11	18	70	78	13	64	13.2	all	76	46	78	68
	76	71-72	79-81	TP		13.3					

Events / Custom Events

```
window.addEventListener('myEvent', e => {
    console.log(e.type);
    throw new Error();
});
window.addEventListener('myEvent', e => console.log(e.detail));


window.dispatchEvent(new CustomEvent('myEvent', {detail: 'My data'}));
```

How to update?

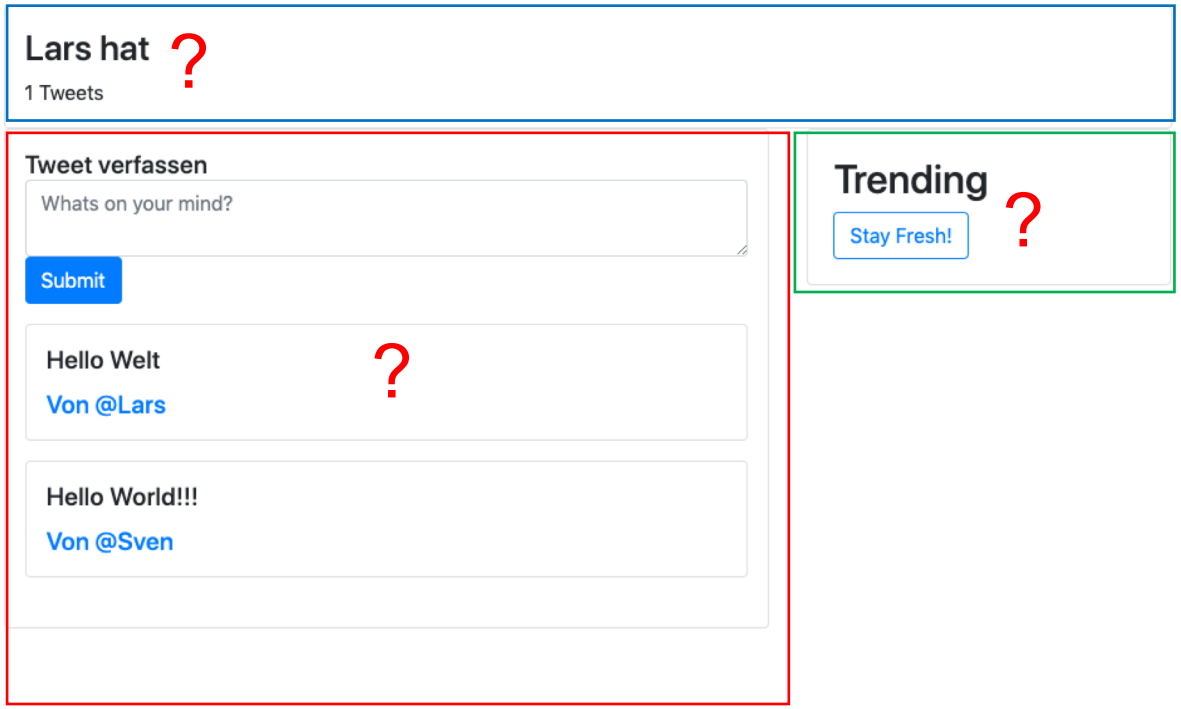
Every MF **refreshes itself**

Self contained!

Re-**fetch**

Optimistic update

Server **push**



Trade-offs

Re-**fetch**

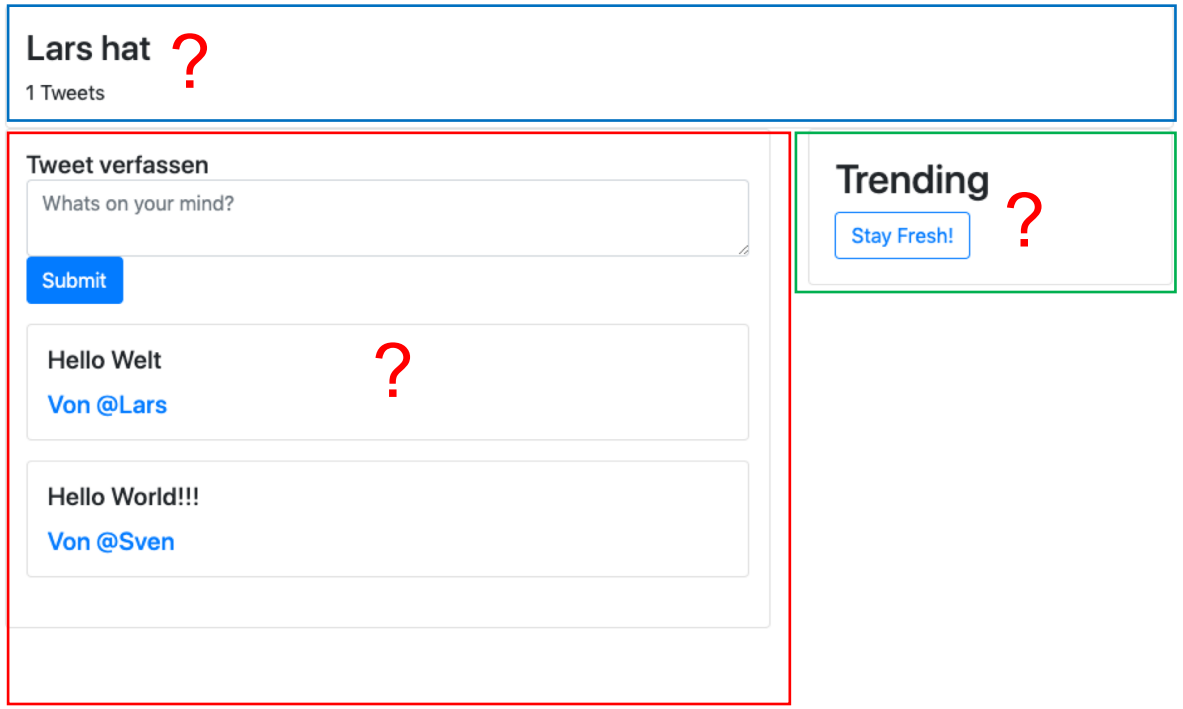
What if data not ready yet?

Optimistic update

What if something goes wrong?

Server **Push**

Scaling?



Demo Time: 05

Server-Side Composition

Works well when…

Lots of **SSR** is used

Caching

Well defined **contracts** & **responsibilties & events**

Server-Side Composition

Works not so well when

High coupling between MFs

Try to **avoid**

Maybe **wrong domains**

Client-Side Composition

Integration via **app shell**

One app for the user

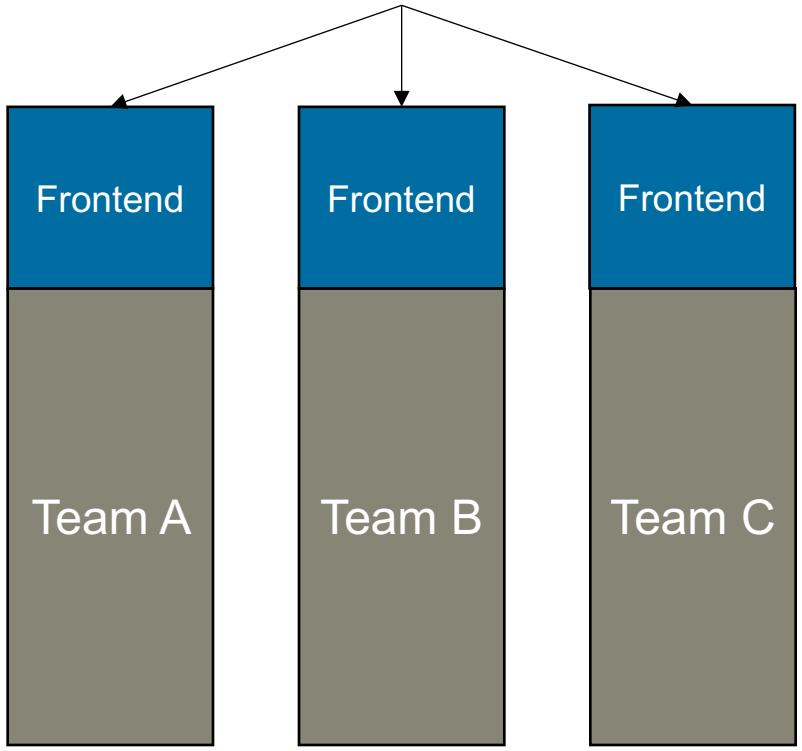
N-Requests

SSR

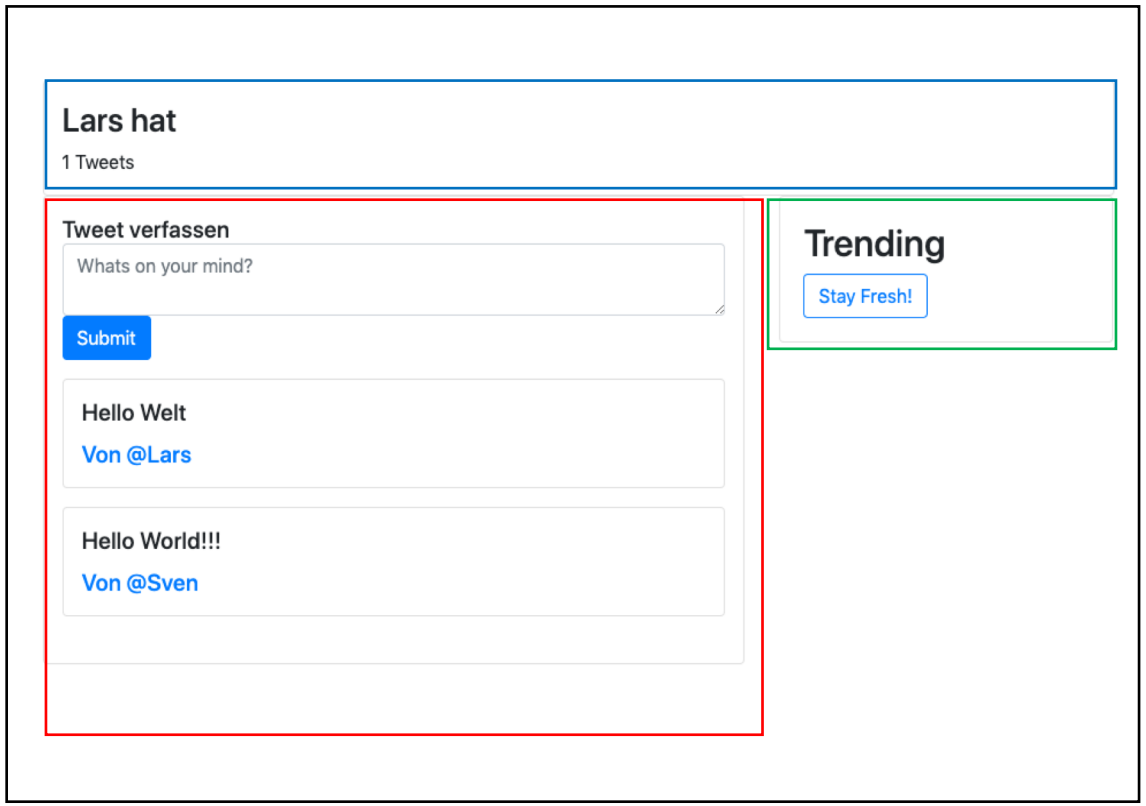
MF == Fragment

CSR

MF == JS-Bundle



Client-Side Composition



http://tweets.twttr.com

function(){…}

http://trending.twttr.com

function(){…}

http://profile.twttr.com

<**h1**>Profile</**h1**>

...

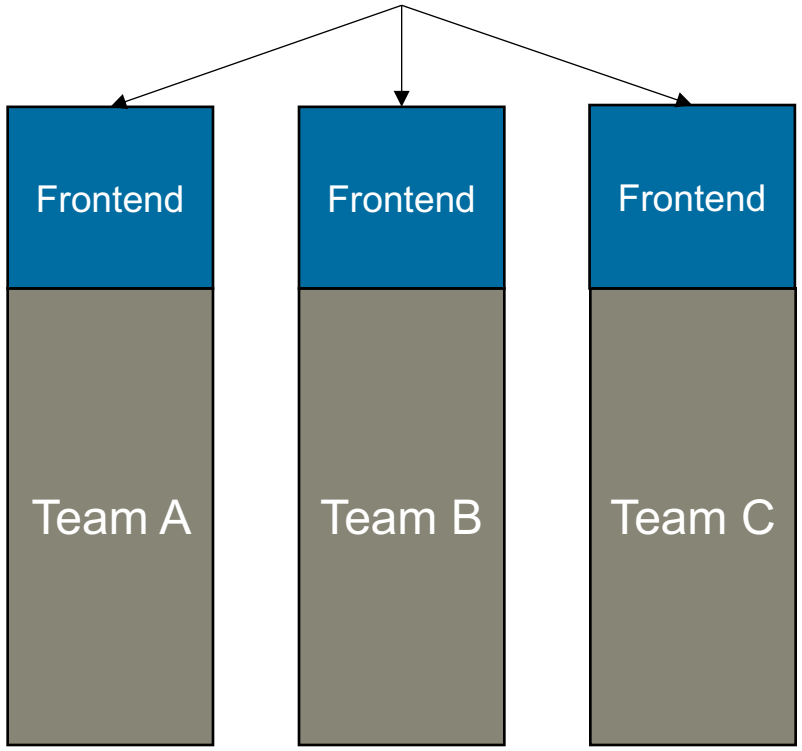
Client-Side Composition

Traffic?

Performance?

Resilience?

Routing?

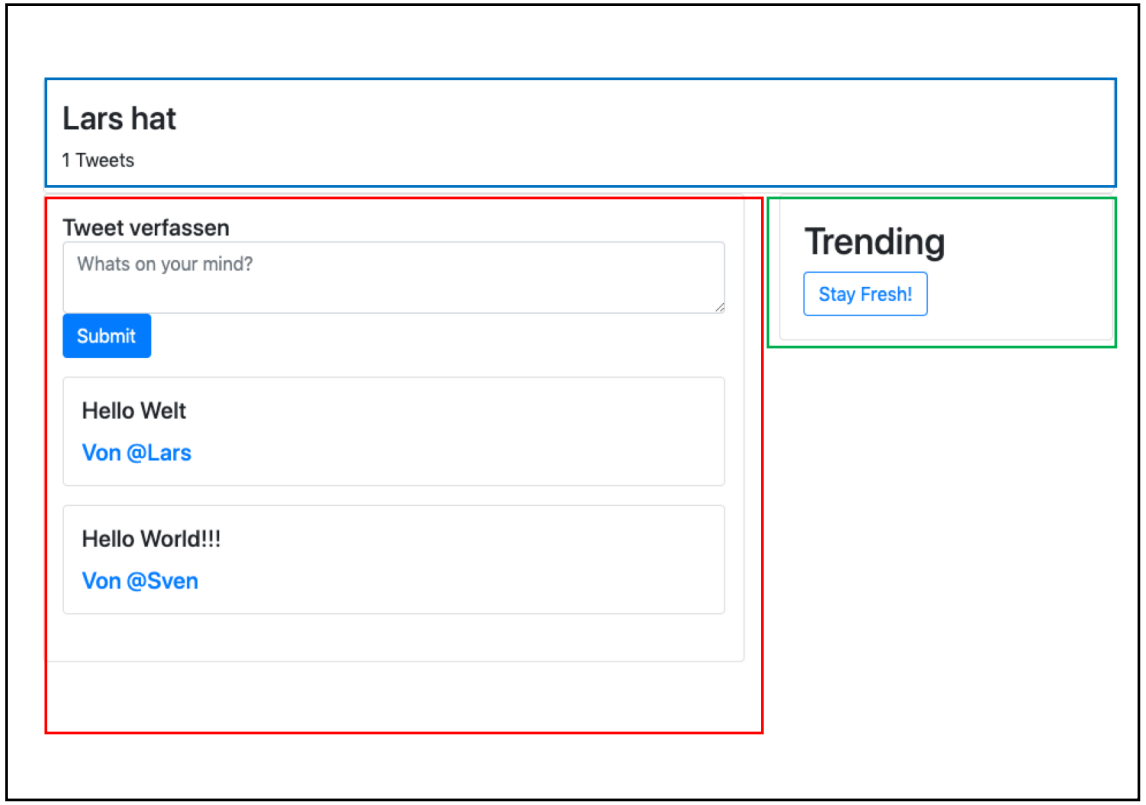


iFrames

„Browser in Browser“

```
<iframe src="http://localhost:4001/profile" ></iframe>
```

```
<iframe src="http://localhost:4002/tweets" ></iframe>
```



iFrames

Total **isloation**

- Legacy integration
- Communication
- Routing

A bit **old fashioned**

- Hard to style
- UX

```
<iframe src="http://localhost:4001/profile" ></iframe>

<iframe src="http://localhost:4002/tweets" ></iframe>
```

Demo Time: 06

SPA-Composition

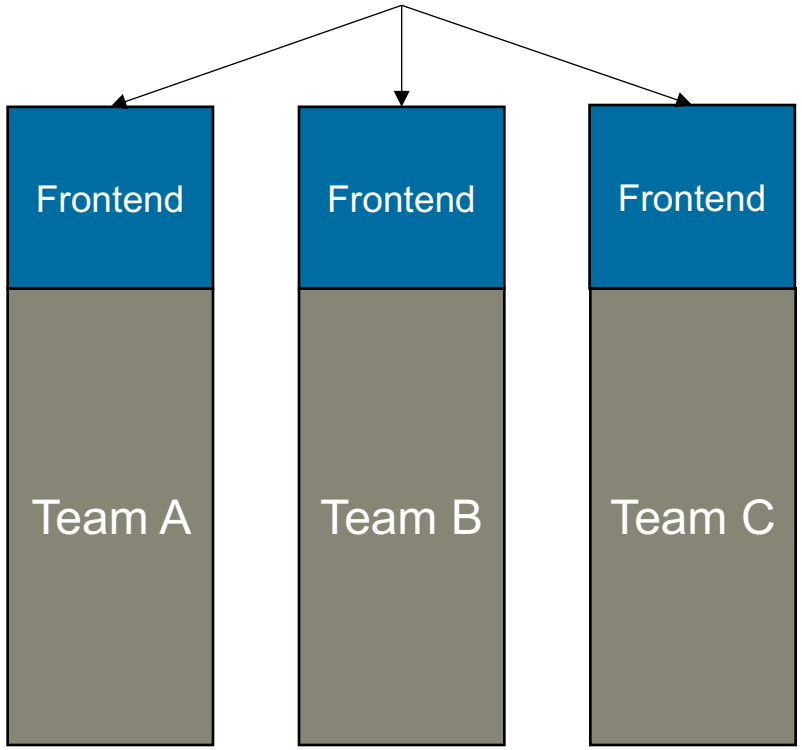
Combine various **SPA**-Frameworks

- Single App
- Multi App

Mostly **CSR**

Requires **Meta-Framework**

- Lifecycle
- Lazy-loading
- Routing



single-spa

Meta-Framework

App shell

Configuration, Routing, Lazy-loading,
Error handling, Collision handling

Adapters for Angular, React & Co

Bootstrap, Mount, Unmount
Custom Adapters

Demo Time: 07a

Just because you **can** does **not** mean you **should**...

The DAZN way

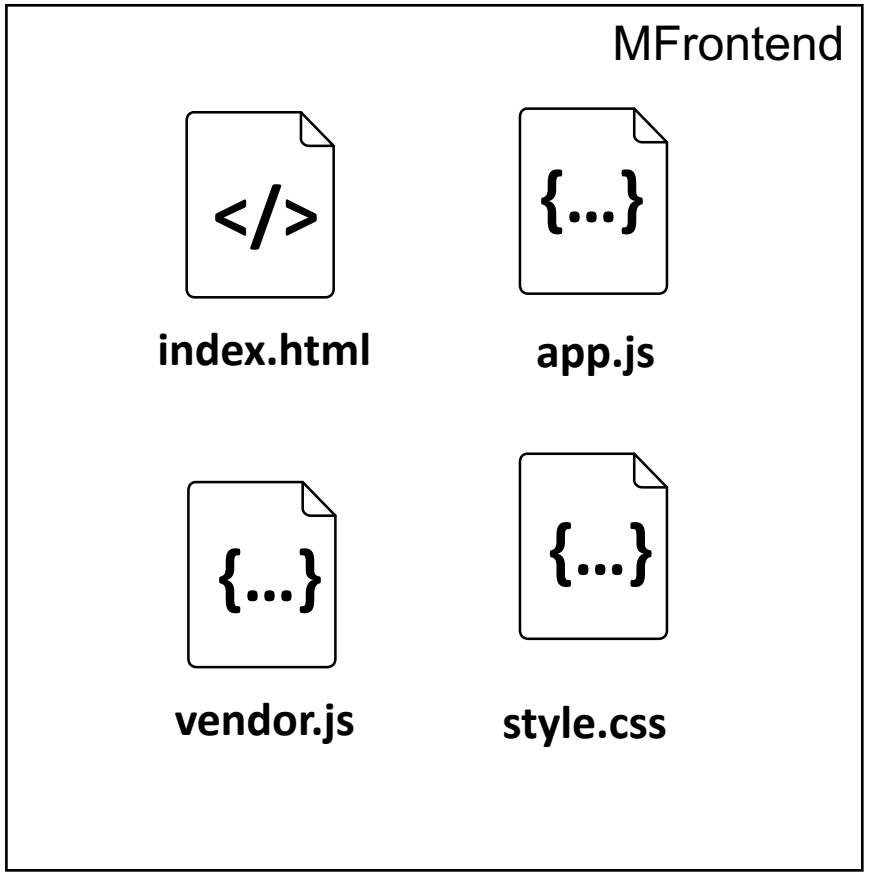
Every **subdomain** is a **SPA**

SPAs **never** run in **parallel**

Load & Unload

DAZN-Bootstrap

Routing, Loading, Device Abstraction



Demo Time: 07b

Client-Side Composition

No server necessarry

But **app shell**

„Modern“ alternative

Does **not** mean **100 frameworks** on one page!

open components

By OpenTable

Registry

SSI & CSI

- Framework agnostic

Node.js & various adapters

Demo Time: 08

CSS

Style **overrides**

- Namespaces
- CSS-Modules
- Shadow DOM

Common look & feel

- CSS-Frameworks
- CSS-Properties
- Web Components

Web Components

Web Components

Set of **standardized** APIs

Denominator for frameworks

Chrome since 2014 (v0)

Refinement 2016 / 2017 (**v1**)

Building blocks

Custom Elements

Create your own tags

HTML Templates

Reusable HTML

Efficient rendering

Building blocks

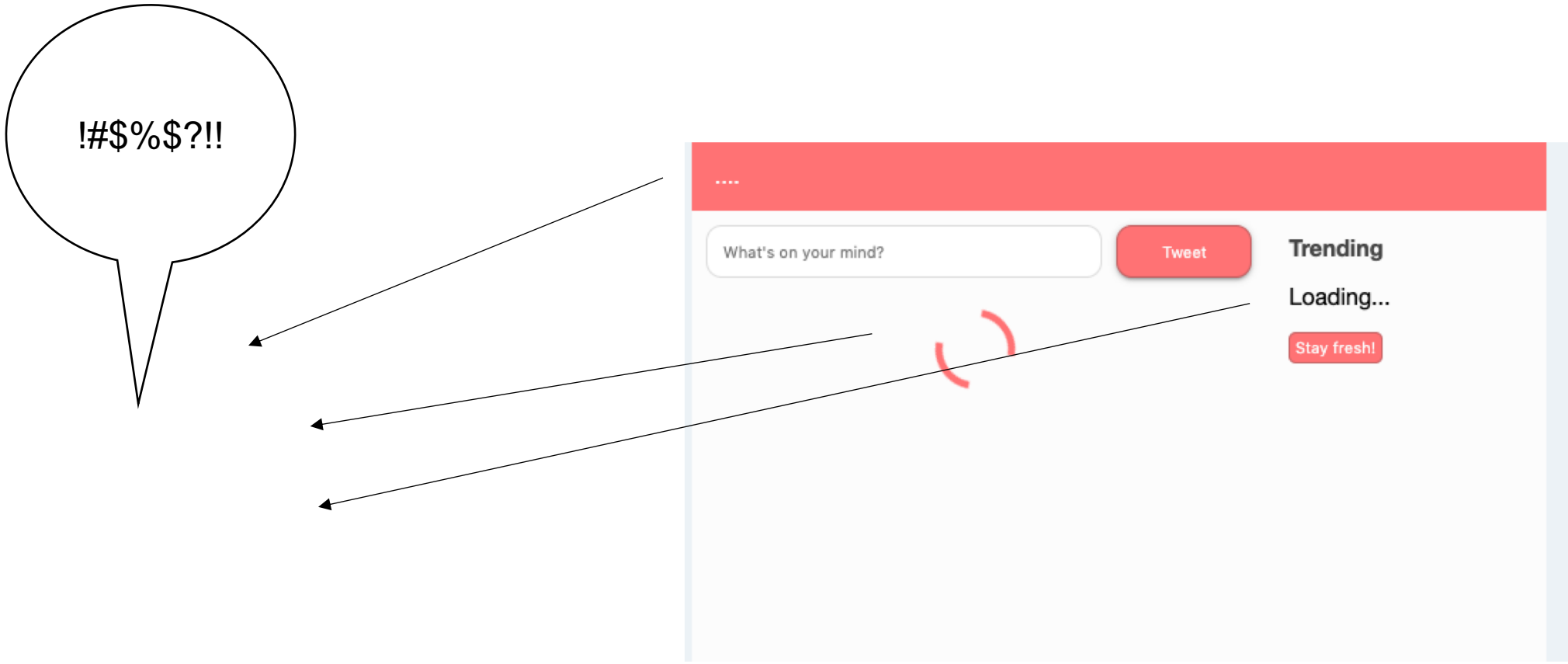
Shadow DOM

Encapsulation

ES Modules

Bye bye HTML imports

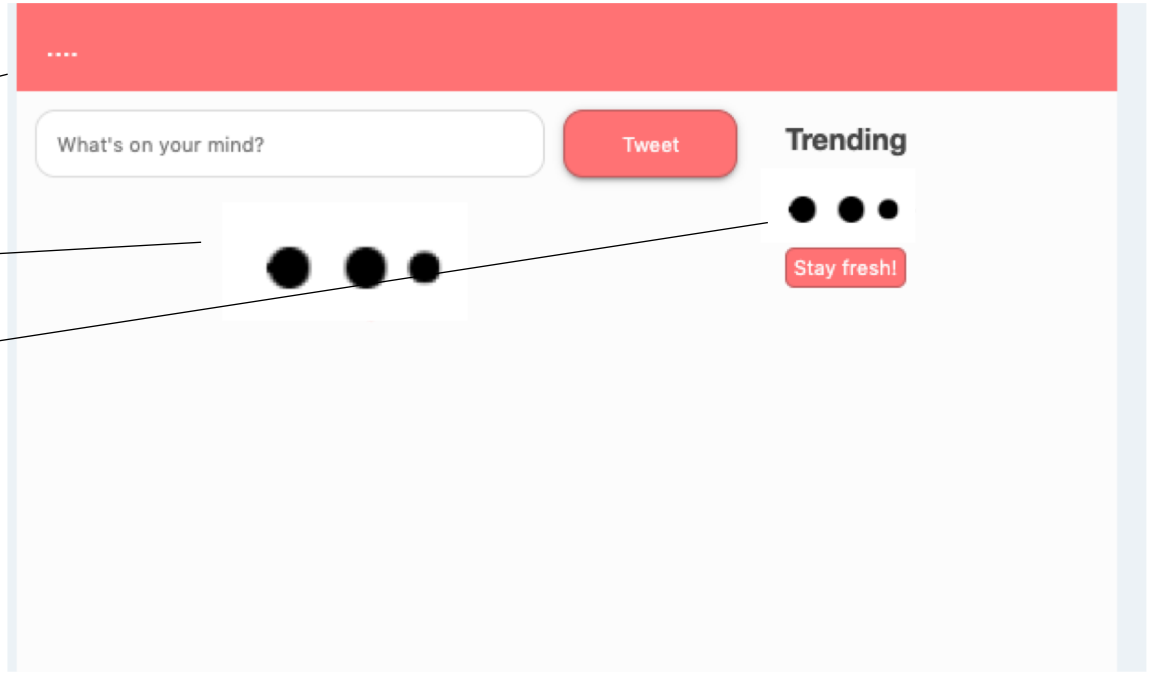
Shared components



Shared components

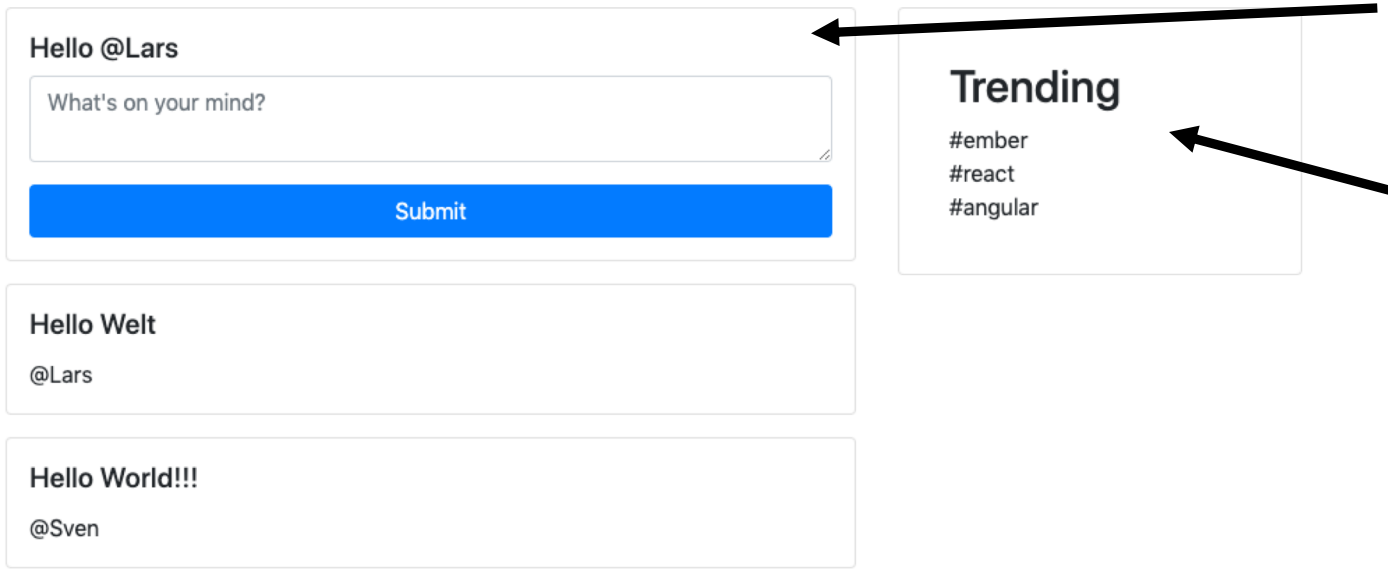
<loader-component>

</loader-component>



Demo Time: 12

Web Components: Integration Layer




```
<html lang="en">
…
<div id="page">
…

<tweets-view user="@Lars"></tweets-view>

…

<trending-view>

</trending-view>

<script src="http://localhost:4000/tweets.js"
        type="module"></script>
<script src=http://localhost:4002/trending.js
        type="module"></script>
</body>
</html>
```

Demo Time: 13

Wrap up: Implementation

No Composition

Server-Side Composition

Client-Side Composition

Web Components

All images without branding are taken from pixabay.com