
Lecture 6

Architectures and Patterns

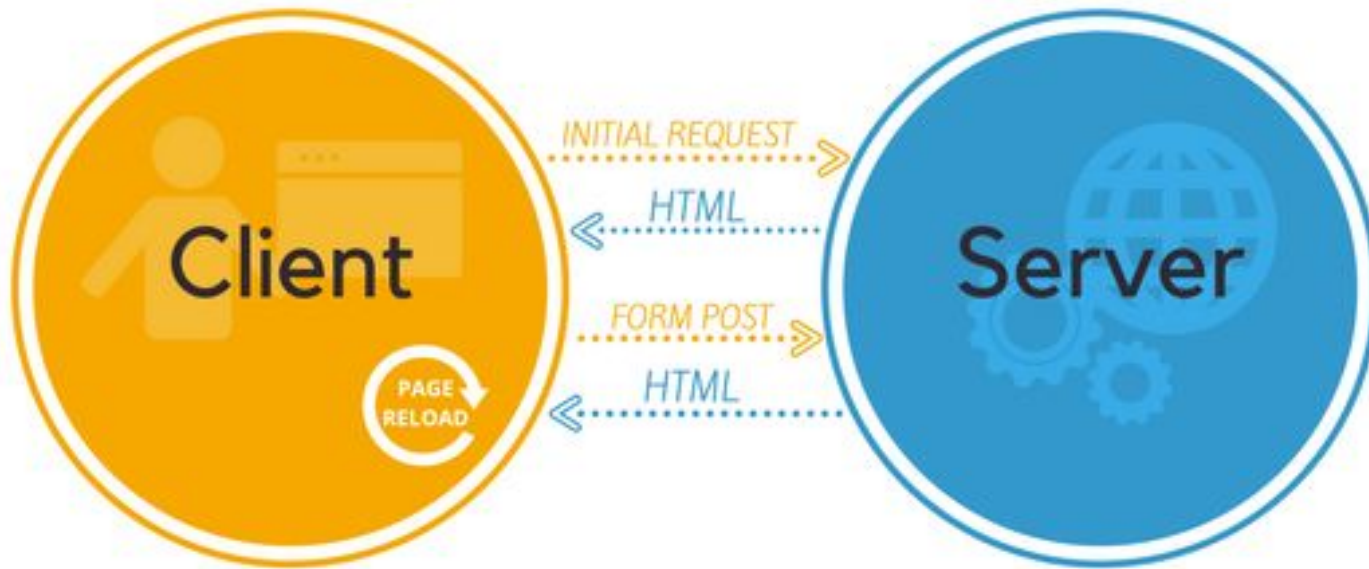
— Frontend Web Development —

What will we learn today?

- What are the options of actually running a modern front-end application?
- What's the fastest way to develop a complete website?
- How to make backend developers obsolete?
- How to make machines deploy your website for you?
- What is SSR and why is everyone talking about it in front-end circles?

Application lifecycle

Traditional page lifecycle



Single Page Applications

JavaScript together with AJAX technology turned browser into a platform that can run fully featured client.

That led to appearance of frameworks that helped to create Single Page Applications. Once they are downloaded and launched in browser, there are no page reloads in case of any state change.

SPA lifecycle



Advantages of SPA

- Fast and smooth UX
- Local caching and offline functionality
- API reuse
- Back-end/front-end separation
- Less server load

Disadvantages of SPA


- Bad SEO
- Easy to make it insecure
- Slow launch
- Memory leaks
- Depends on JavaScript

Rendering Techniques

Rendering Techniques

There are 4 main strategies:

- Server-Side Rendering (SSR)
- Static Site Generation (SSG)
- Client-Side Rendering (CSR)
- SSR/SSG + Hydration (a.k.a. Universal Rendering)



Rendering on
the Web

Client-Side Rendering

The server only sends a minimal HTML shell and a script tag. The script handles creating all the HTML on the client.

Benefits:

- Less work for the server and simpler to implement

Drawbacks:

- The website won't render with JS disabled (or not loaded yet)
- Primitive search engines cannot crawl the content of the website
- Puts the burden of rendering on less powerful client devices
- Can result in a flash of incomplete content



Knockout

plain

JS

Google



Bing



YAHOO!



Aol.



DuckDuckGo



Yandex



Not indexed

Server-Side Rendering

HTML file generated on the fly in the server

Benefits:

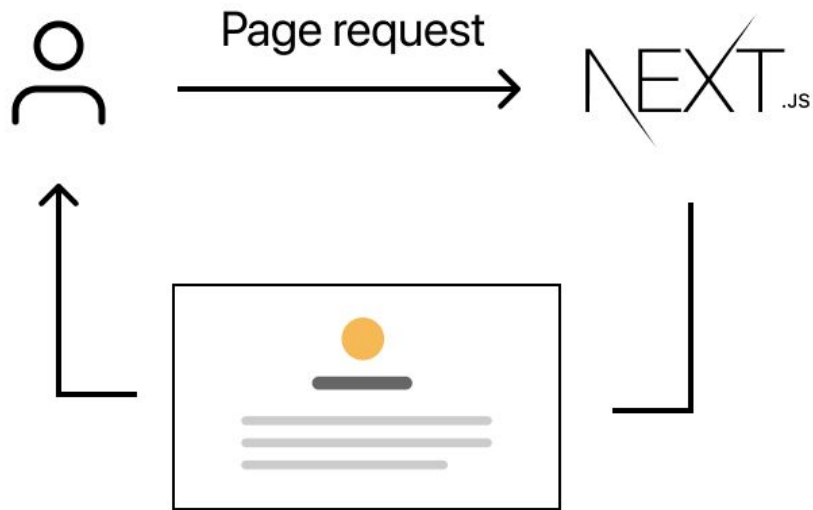
- Can send different content to different users
- Improved SEO over CSR (search engine bots **weak**)
- Improved performance over CSR (client **weak**)
- Prevents a flash of incomplete content
- When JavaScript doesn't load, your website still works (network **weak**)

Drawbacks:

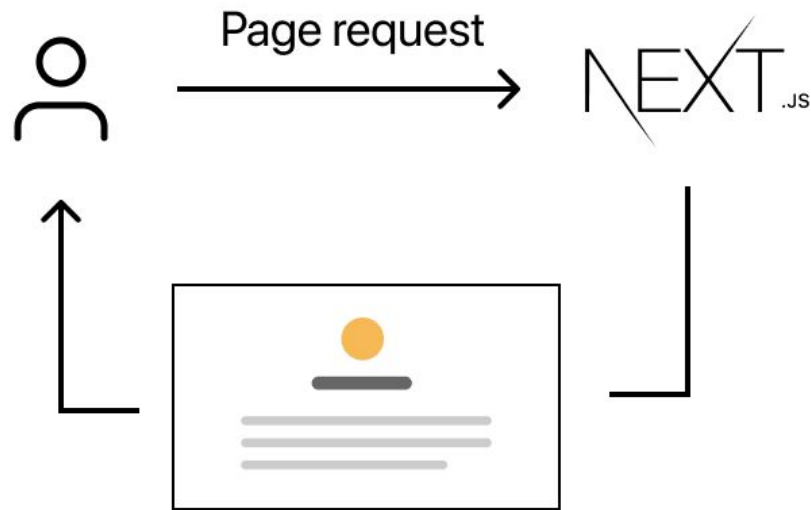
- Requires a server that is capable of SSR
- Increases load on the server

Server-side Rendering

The HTML is generated on **each request**.



The HTML is generated



The HTML is generated

Static Site Generation

HTML files are generated ahead of time and served statically. Works best when cached at CDNs.

Benefits:

- Improved SEO over CSR (again)
- Improved performance over CSR (again)
- When JavaScript didn't load... (you get the point)
- Can be cached really well

Drawbacks:

- Sends the same page to all users (but can still introduce dynamic content through hydration)
- No dynamic URLs (without server support)

BUILD TIME



Data / Content / Templates



Static Site Generator



Web Site

REQUEST TIME

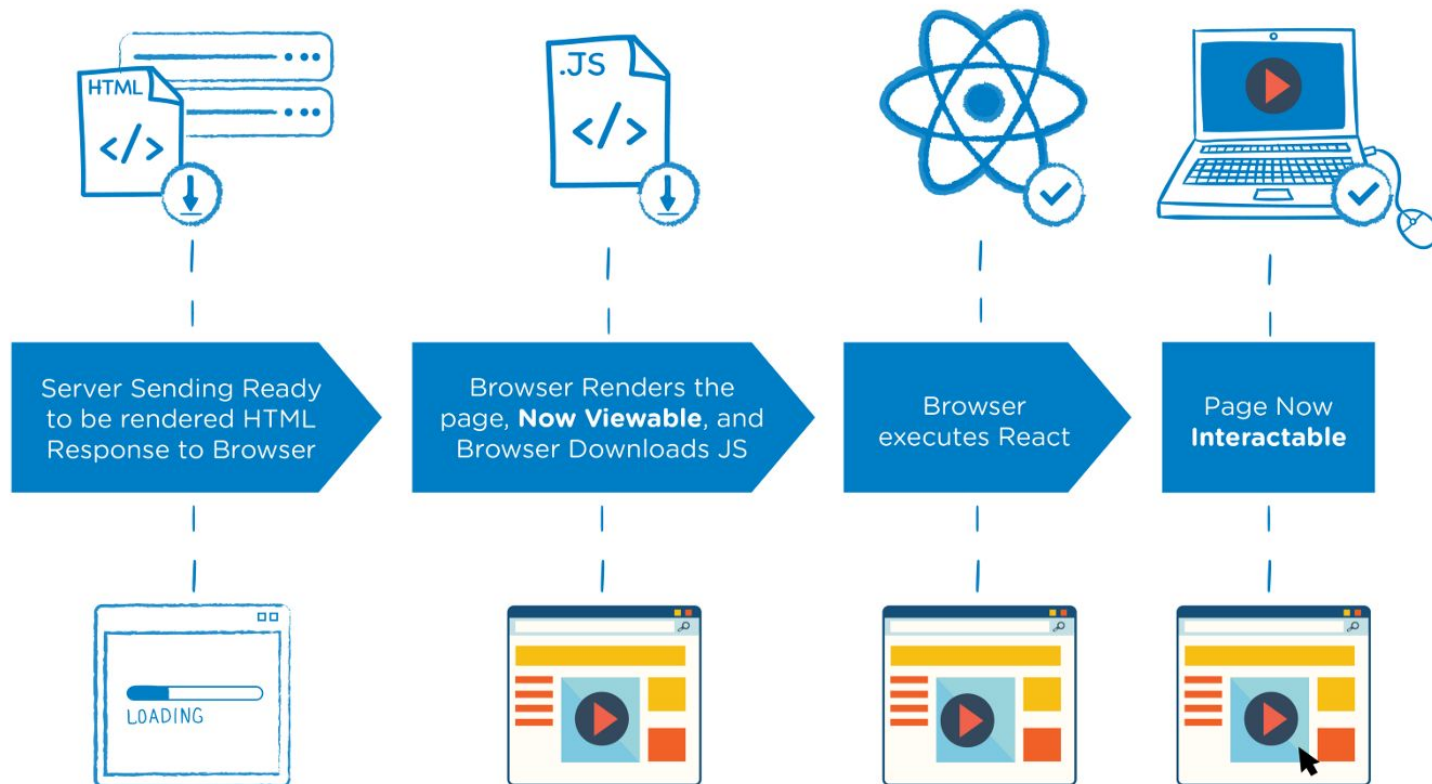


Users



Hosting

Hydration in SSG and SSR




When to use each approach?

- **Static Site Generation** — websites with a large amount of static content
 - Blogs
 - Documentation websites
 - News websites
- **Server-Side Rendering** — whenever you can afford it 🏰
- **Client-Side Rendering** — for mostly user-specific websites
 - Auth-protected control panels
 - When only the client can make requests to the backend for data (e.g.: CORS policy)

Performance Comparison Metrics

- **FP:** First Paint
- **FCP:** First Contentful Paint
- **TTI:** Time to Interactive
- **TTFB:** Time to First Byte



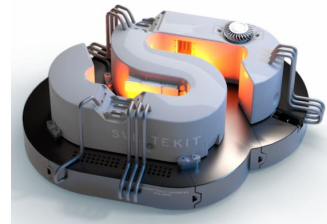
User-centric
Performance
Metrics

Rendering Techniques Comparison

	FP	FCP	TTI	TTFB
SSR	✓	✓	✓	✗
SSG	✓	✓	✓	✓
CSR	✗	✗	✗	✓

Meta-frameworks

SvelteKit / Next.js / Nuxt



A complete package for developing production-ready websites with a lot of cool features and optimizations out of the box

Highlights:

- Code splitting
- Built-in routing solution
- SSR out of the box
- Prefetching
- State management solutions



Architectural Patterns

Serverless

Don't have a long-running server process at all.

Umbrella-term for:

- No-code backends (Firebase, Supabase)
- Serverless functions

Static Hosting

Server only serves static files

- Works only with SSG
- CSR requires server support for it (redirect all to /)
- Cheapest and simplest option
- Most performant (uses CDNs)



Firebase Hosting

Edge Rendering

Combines SSR with Serverless, using CDNs

- Many small servers, distributed across the globe
- Only run on demand (per request)
- Perform server-side rendering

Backend For Frontend (BFF - or front-back)

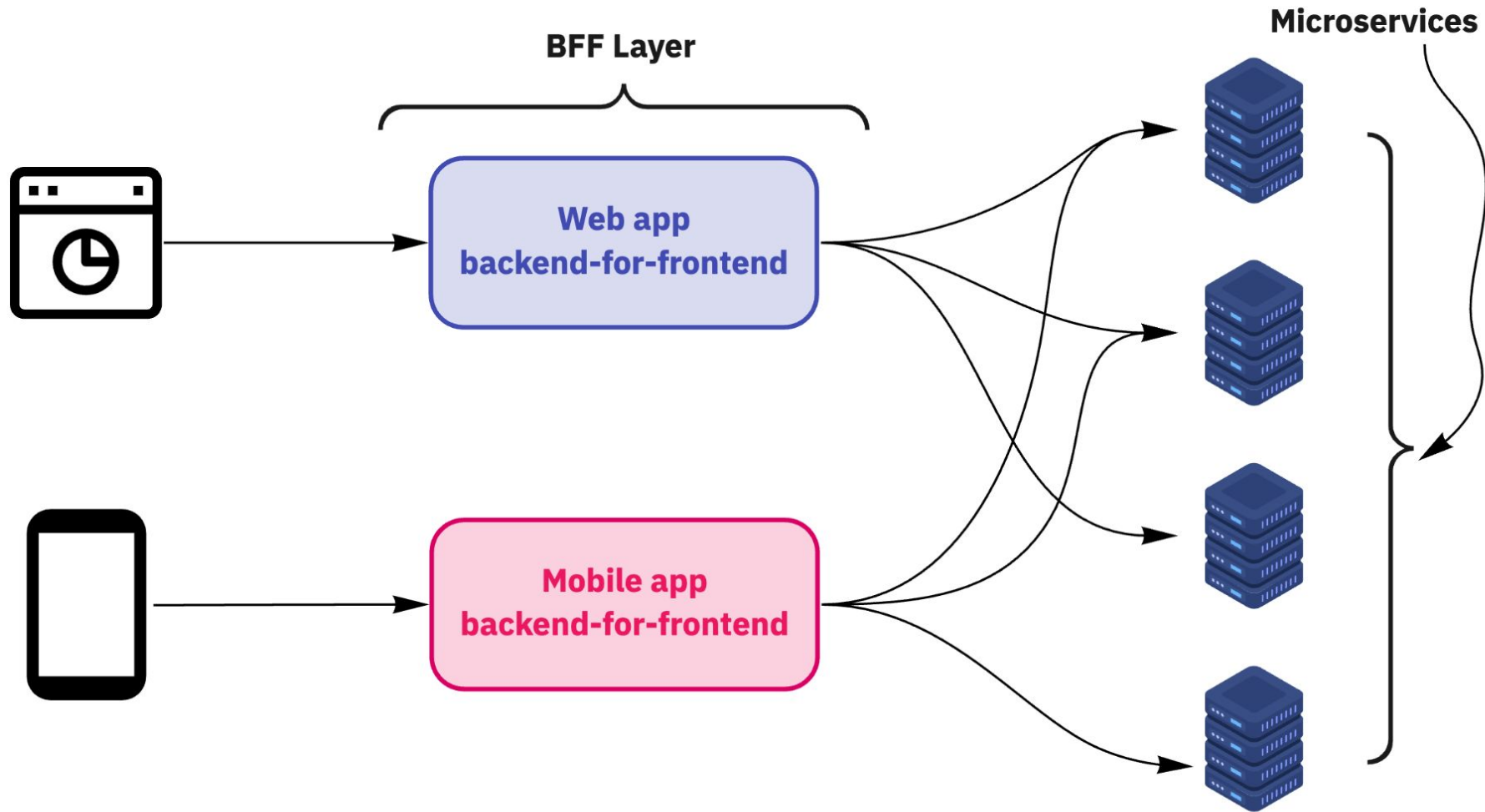
In addition to an API, you have a server in JavaScript that handles SSR and backend communication.

Benefits:

- Your API stays secure
- JS servers can do SSR

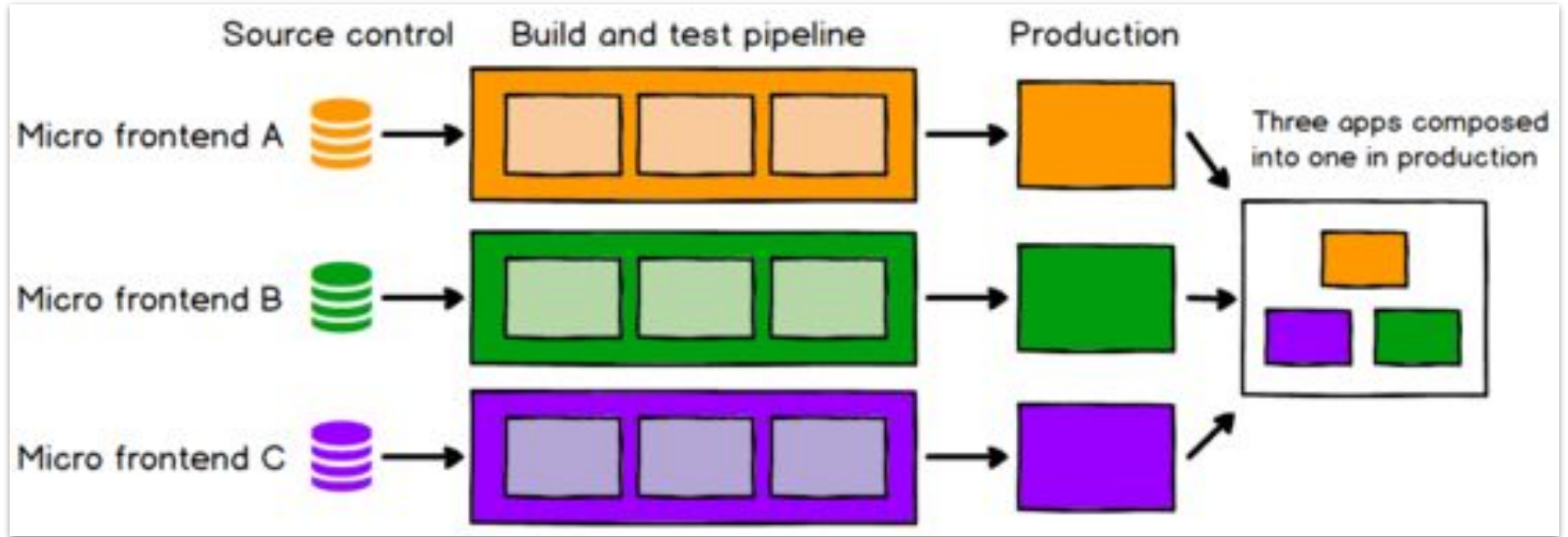
Drawbacks:

- You have to run another server (can still be done on a single machine)

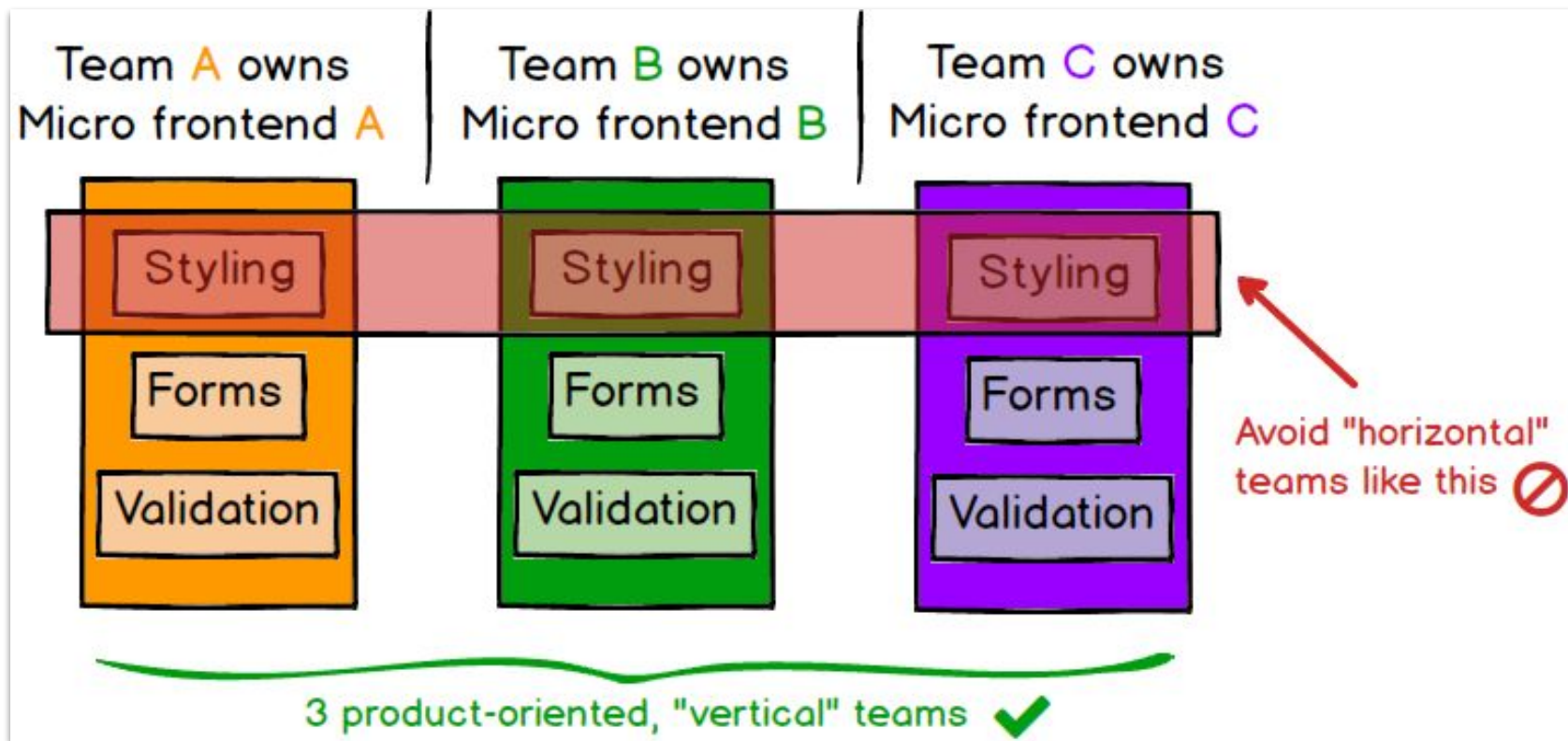


Micro-frontends

Microservices but for the frontend



Micro-frontends



Deployment

CI/CD

Pipelines to do anything you want on events such as pushes to the main branch. Usually used for:

- Linting code automatically
- Running tests automatically
- Deploying your website automatically
- ~~Writing offensive comments to first time contributors automatically~~

GitHub Actions

- Works best with GitHub (duh)
- YAML configuration files that you push to the `.github/workflows/` directory in your repository
- Most common tasks have already been published as *actions*

Learn here: <https://docs.github.com/en/actions/learn-github-actions>

Steal from here: <https://github.com/illright/attractions>

Acknowledgment

Thanks to **Lev Chelyadinov** for his help with preparing the content of this lecture.

Learn more

- <https://web.dev/rendering-on-the-web/>
- <https://dev.to/coderedjack/critical-rendering-path-web-performance-23ij>
- <https://www.netlify.com/blog/edge-functions-explained/>
- <https://kit.svelte.dev/docs/adapters/cloudflare>
- <https://www.patterns.dev/posts/progressive-hydration>