Microfrontends Implementation Patterns

Thessaloniki 2025

About me

- Alexandros Tsichouridis
- Senior Software engineer
- Full Stack (Java, Spring, Angular, React ...)
- University of Macedonia, BSc & MSc
    - Clean code and design,
    - CS Education

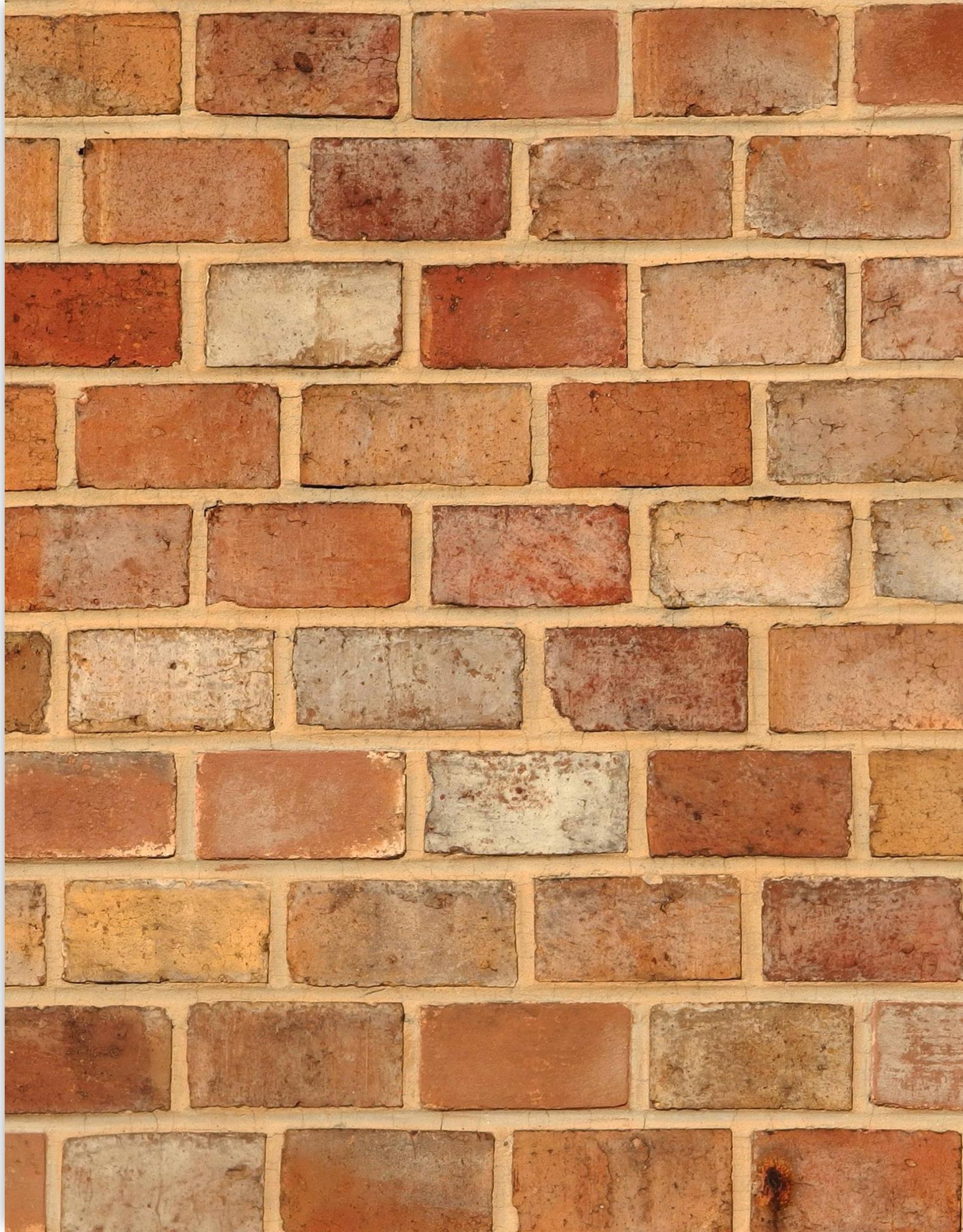
Agenda

01

Why Microftontends

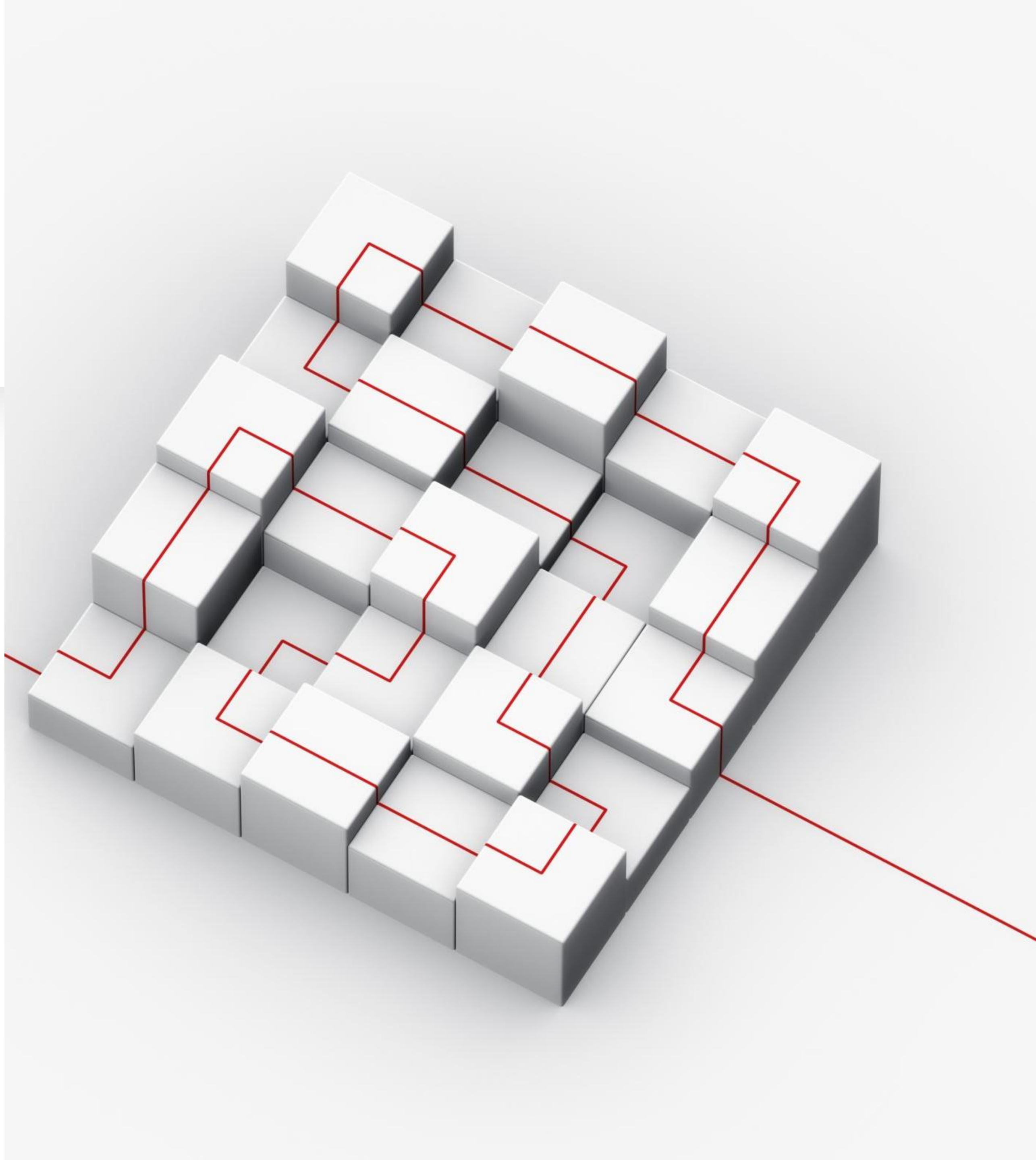
Typical Frontend application - Monolithic Approach challenges

- Very large codebase
- Multiple teams
- Time consuming deployment
- Tightly coupling
- Technology stack restriction



What are microfrontends ?

- Architectural approach for building modern web applications
- Splits a frontend app into smaller, semi-independent modules
- All modules are integrated to form a single user experience
- Inspired by microservices in backend development



Micro frontends - Benefits

- Smaller codebase and loose coupling
- Team independent
- Development flexibility
- Improved maintainability
- Independent deployments
- Technology diversity

Challenges

- Cross application communication
- Complex deployment
- Performance and latency
- Shared packages, library dependencies
- Versioning management
- Routing and application module mounting

02

Main Approaches

Architecture Approaches

- Build-time Integration
- Server-side Composition
- Runtime (Client-side) Integration
  - Edge-side includes

Build Time Integration

- Single bundle
- Monorepo or different artifacts
- Release Strategy

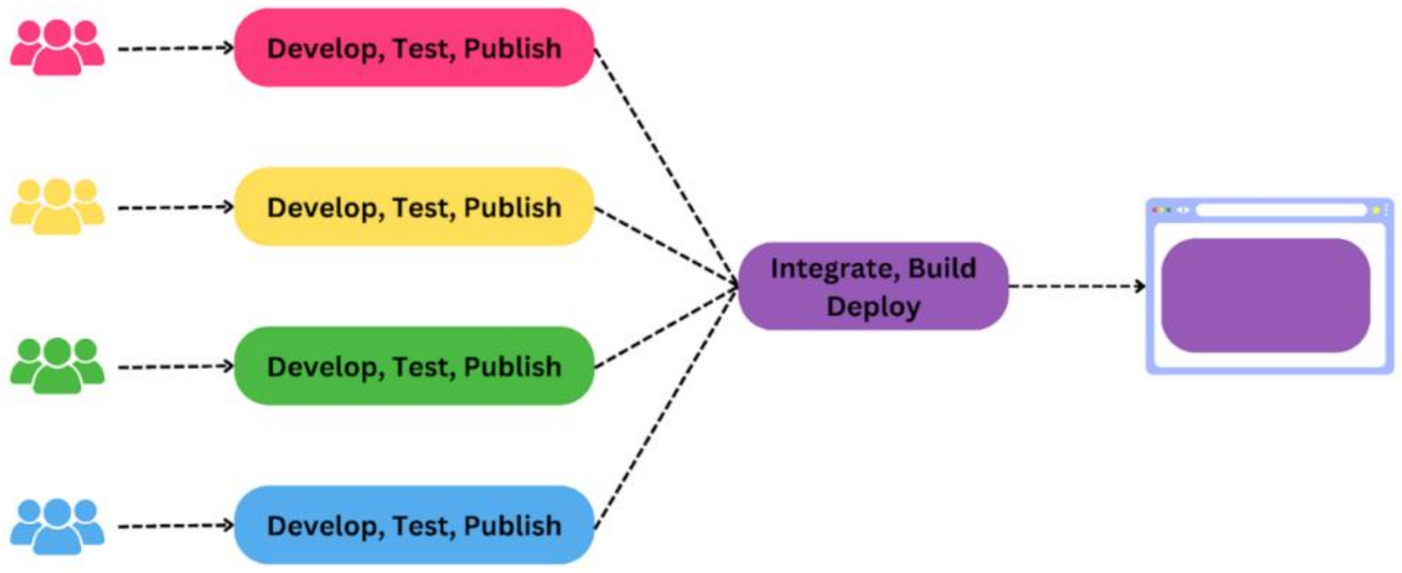


Diagram from https://www.syncfusion.com/blogs/post/micro-frontend-run-time-vs-build-time

Server-side Integration

- Mutliple bundles into one
- Server effort
- SEO

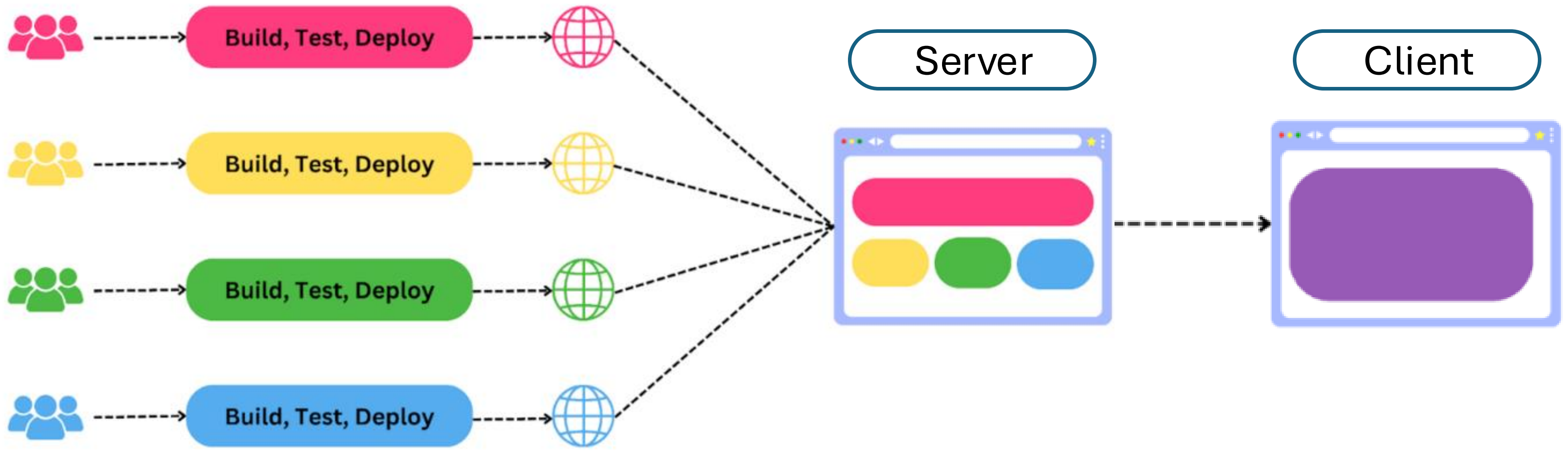


Diagram from https://www.syncfusion.com/blogs/post/micro-frontend-run-time-vs-build-time

Client-side Integration

- Independently deployed bundles
- Monorepo or Multirepo
- Edge side includes
- Client-side logic

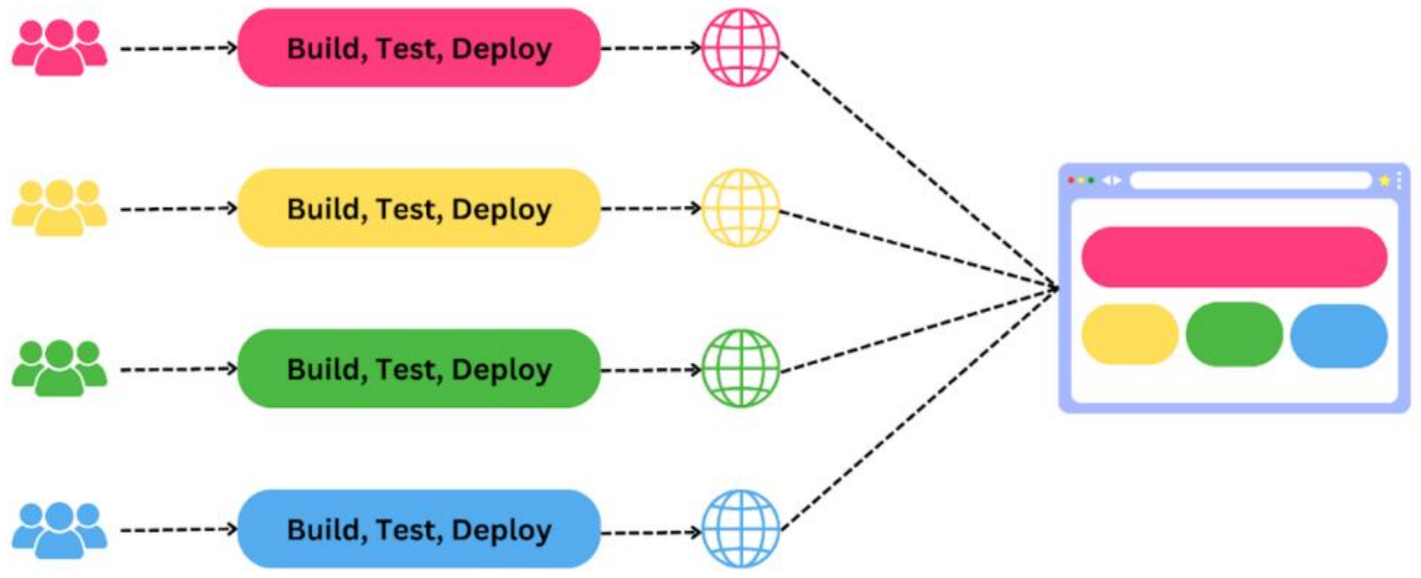


Diagram from https://www.syncfusion.com/blogs/post/micro-frontend-run-time-vs-build-time

03

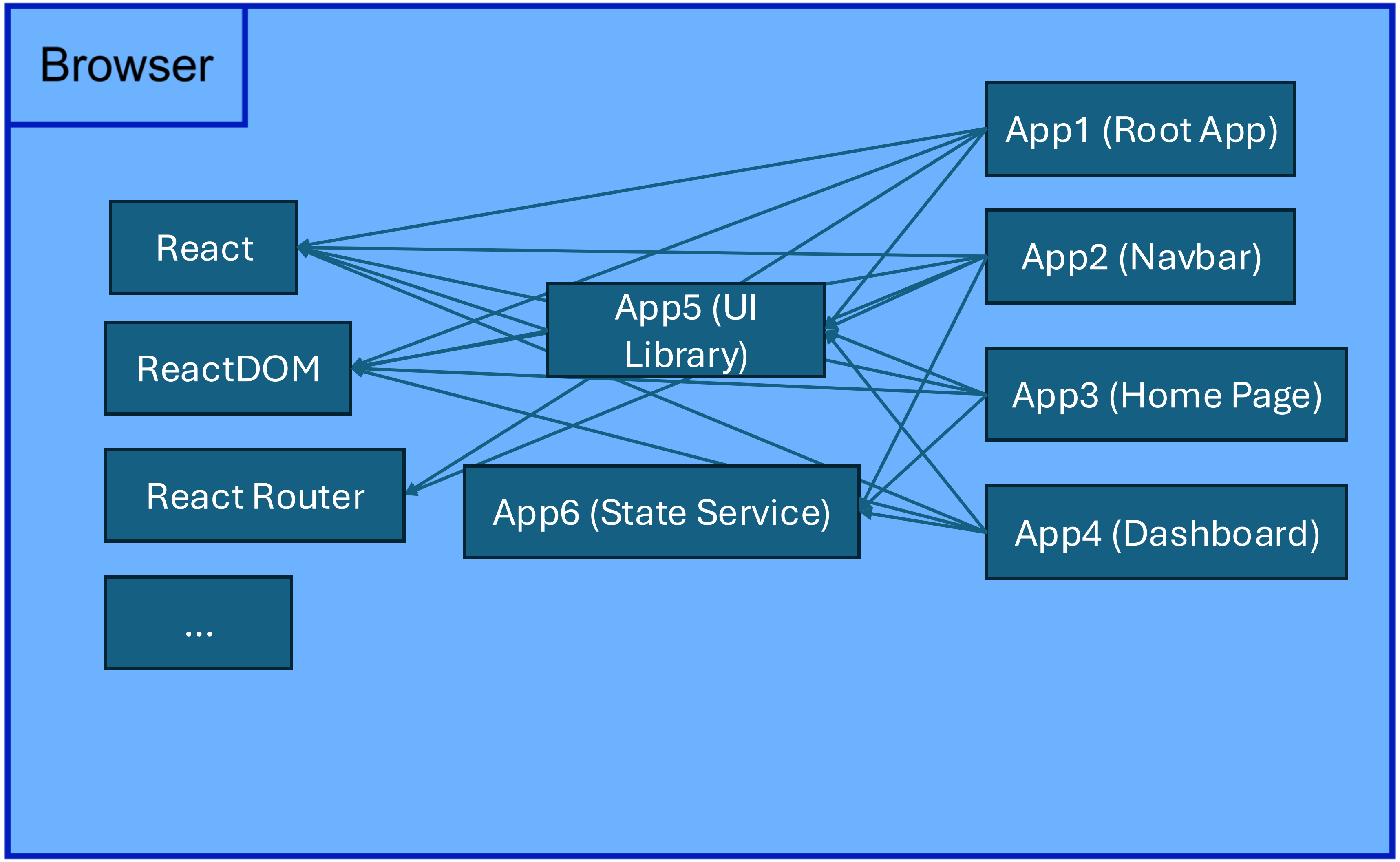
Microfrontends in Action

Micro frontends – Client-side integration in action!

- Webpack Module Federation
- Single-spa
- Web Components

How external / common dependencies work

- Packages are loaded once, used by many
- We have to define which dependencies are external or common
- When sharing dependencies, we can share their state too
- Some packages must be shared across applications to work properly (ReactDOM)
- Plugins do this automatically in many cases



Hands on!

Webpack Module Federation

- Industry Standard
- Included in Webpack 5
- Strong community
- Used by other tools
  - module-federation.io
  - NX

Example Scenario

- E-Commerce application (React)
- Different parts must be developed by different teams
- Consists of 5 parts
  - Navbar
  - Product
  - Product page
  - Checkout page
  - Order history

Single-Spa

- Framework agnostic
  - Angular, React, Vue, Vanilla JS, AngularJS and many others
- Supports legacy technologies before Webpack 5
- Leverages importmaps
- Single-SPA applications or Parcels

Single-Spa Example scenario

- Old Angular 9 company CRM app developed in 2020
- Modernized incrementally
- Different teams work using different technologies
- Modern frameworks
  - Angular 19
  - React 19
- Follow microfrontends approach using single-spa

Web Components

WEB COMPONENTS

- APIs for creating custom elements
- Follows Web Specifications
- Reusable elements / components
  - Design systems and component libraries
- React, Angular, Vue support creating web components
- Lit, Stencil
- Can be combined with Single-Spa or Module Federation or just SPAs

Microfrontend approaches comparison

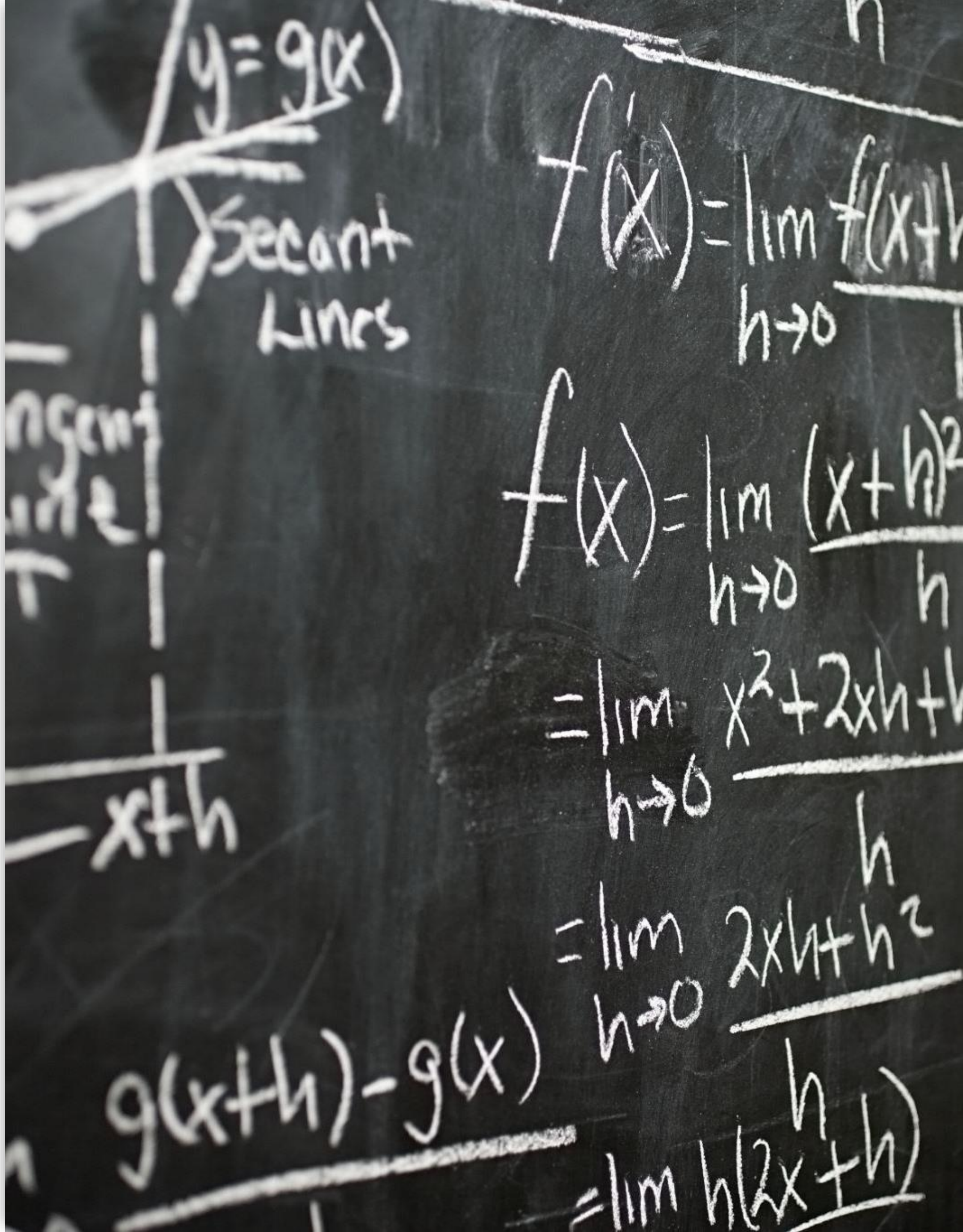
- Module Federation
  - Strong community
  - For webpack, avaible to other bundlers through community plugins
  - Good for same framework
- Single spa
  - Less active community
  - Framework Agnostic
  - Good support for Legacy systems
- Web Components
  - Specification for web applications, Browser native feature
  - Framework Agnostic

Beyond the Basics

- Dynamic module discovery
- Module metadata – manifest
- Assets management
- Styles mounting/unmounting and isolation
- Routing
- Version management – multiple package versioning
- Framework agnostic
- Legacy system modernization

Conclusion

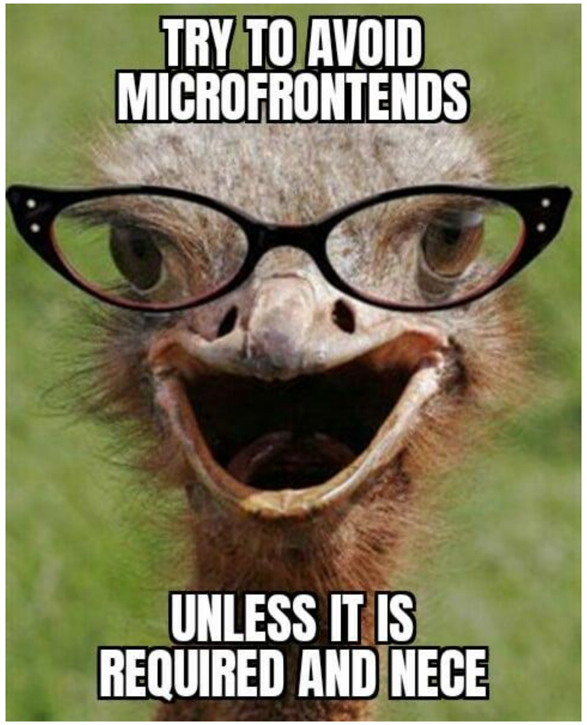
- Examine the actual needs and requirements
- Not ideal for small projects
    - Unnecessary complexity
- Steep learning curve

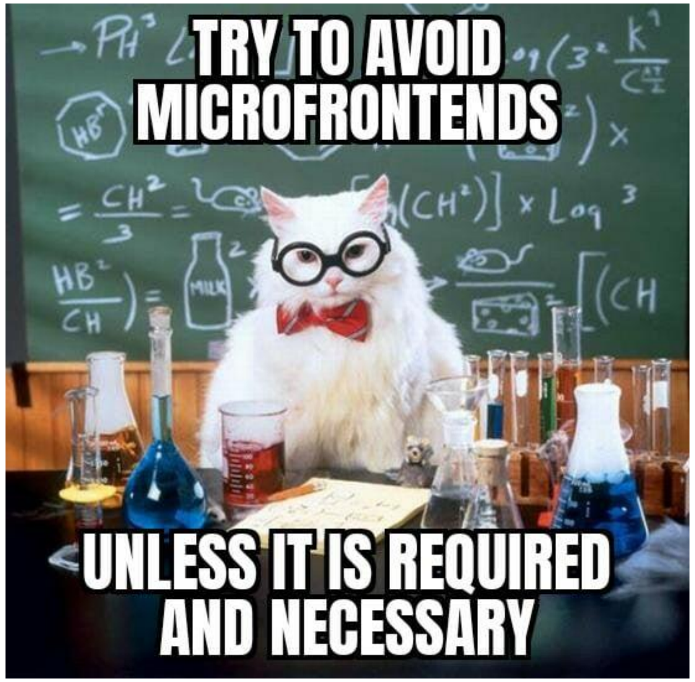


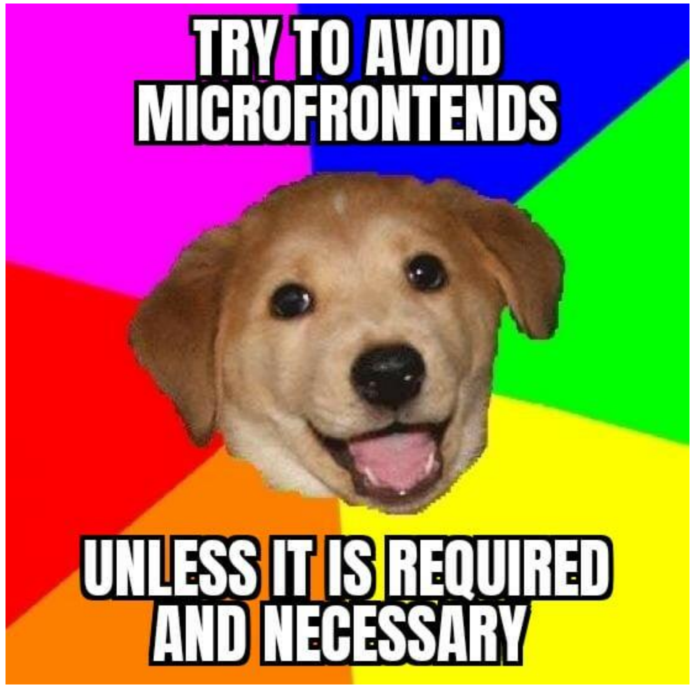
$ Try to avoid microfrontends, unless it is **required** and **necessary**

Got any fatherly advice for me?
try to avoid microfrontends son,
unless it is required and necessary

TRY TO AVOID MICROFRONTENDS
UNLESS IT IS REQUIRED AND NECESSARY


TRY TO AVOID MICROFRONTENDS
UNLESS IT IS REQUIRED AND NECE


TRY TO AVOID MICROFRONTENDS
UNLESS IT IS REQUIRED AND NECESSARY


TRY TO AVOID MICROFRONTENDS
UNLESS IT IS REQUIRED AND NECESSARY

THE END
THANK YOU FOR YOUR ATTENTION

GitHub Repo

LinkedIn