UI In A Microservice Architecture

MICROSERVICES

Dozent - Anton Spöck

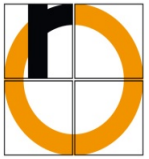
- Co-Founder & CTO innFactory GmbH

- Wirtschaftsinformatiker
- Fachinformatiker

- Cross Platform UIs
- React, React-Native
- Conversational UIs (Sprach- u. Chatbots)
- JavaFX, JavaEE, Android

Für weitere Fragen/Anregungen/Beschimpfungen:

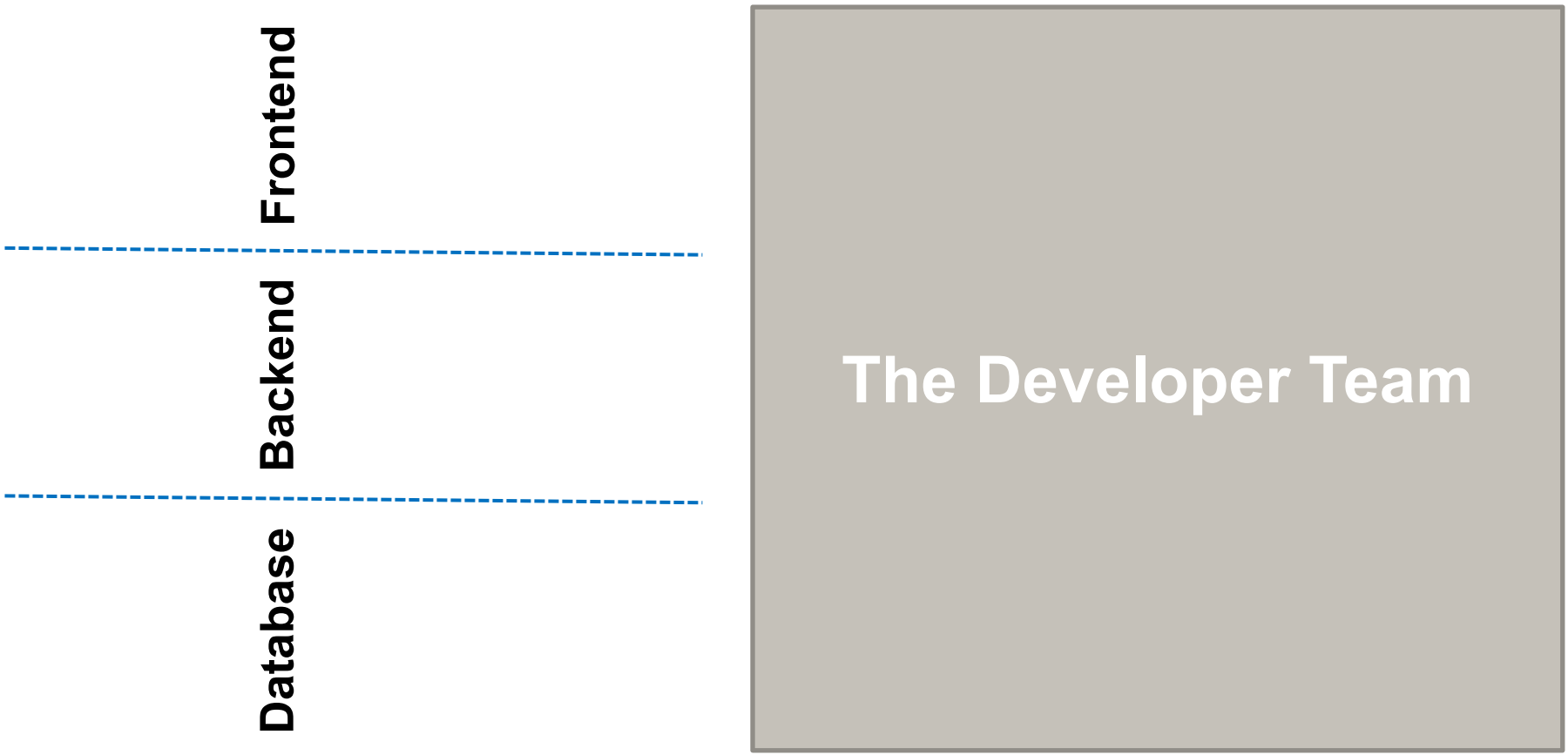
a.spoeck@innFactory.de



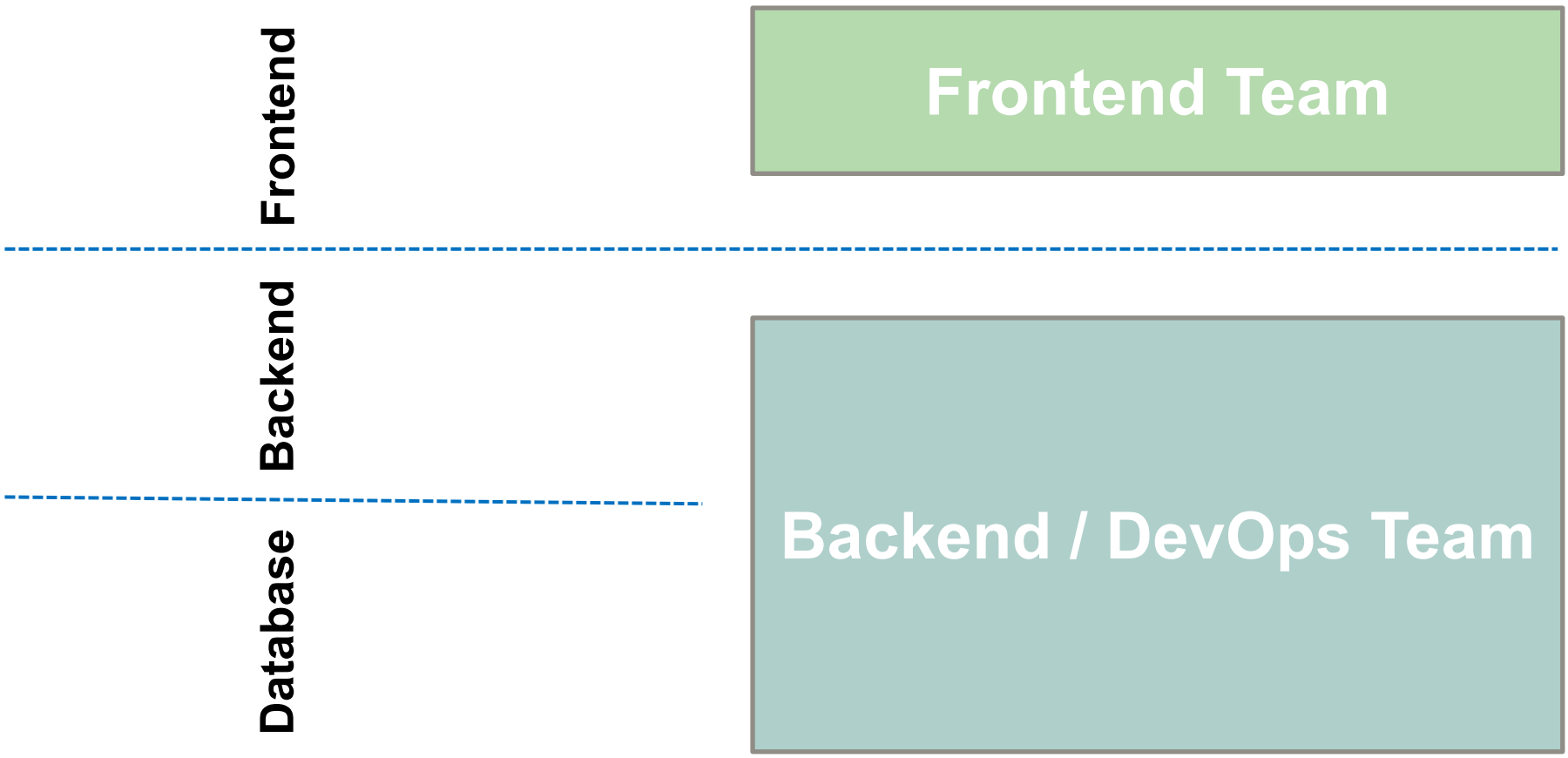
Content

- The classical approach
- The fullstack approach
- Comparison
- Problems with a Microservice Architecture in a GUI
- Styleguides
- Latest UI trends

The Monolith



Front & Back


Frontend
Backend
Database
Frontend Team
Backend / DevOps Team

The classical approach to split UI and Backend

Without Microservice Architecture:

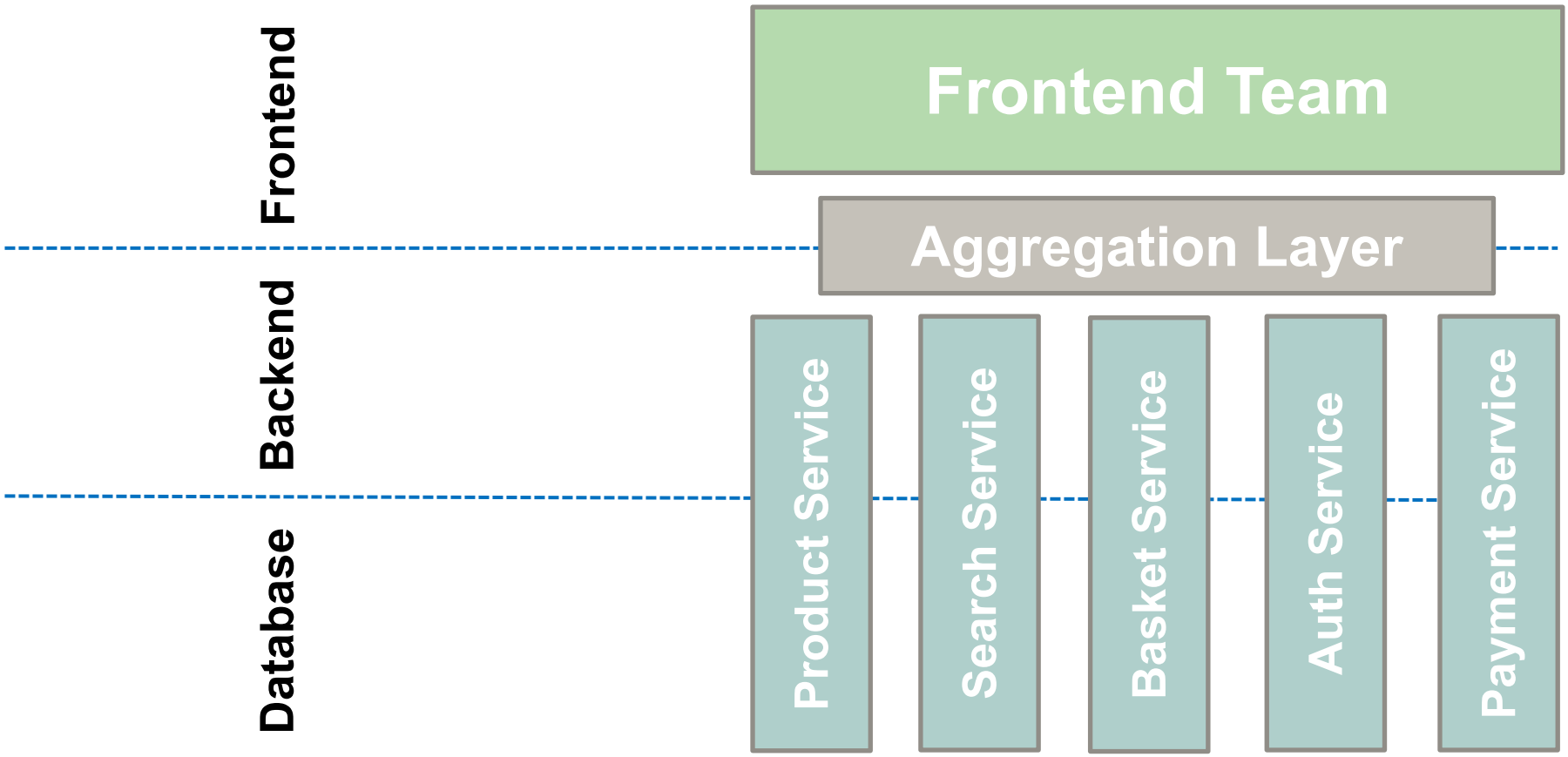
Split the Frontend Developers and the Backend Developers (+ DevOps Engineers) into separate Team

Together with a Microservice Backend:

- Also split the team in front- and backend
- Horizontal separation of each service in the backend
- Specific aggregation for each frontend

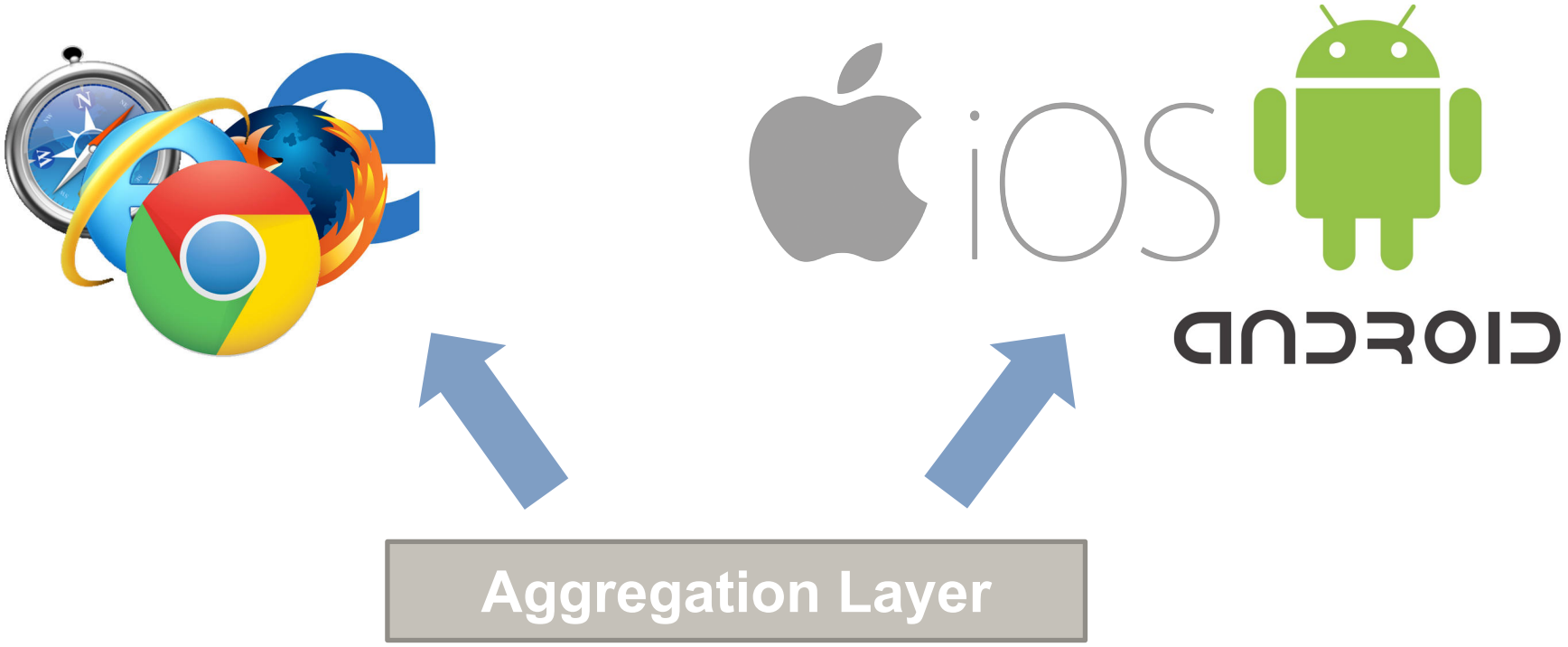
→ Developers are specialized and separated for each technology

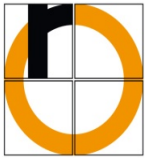
Microservices



Aggregation Layer

- Supports an API for each frontend → Backend for Frontend Pattern (BFF)
- e.g. REST, GraphQL



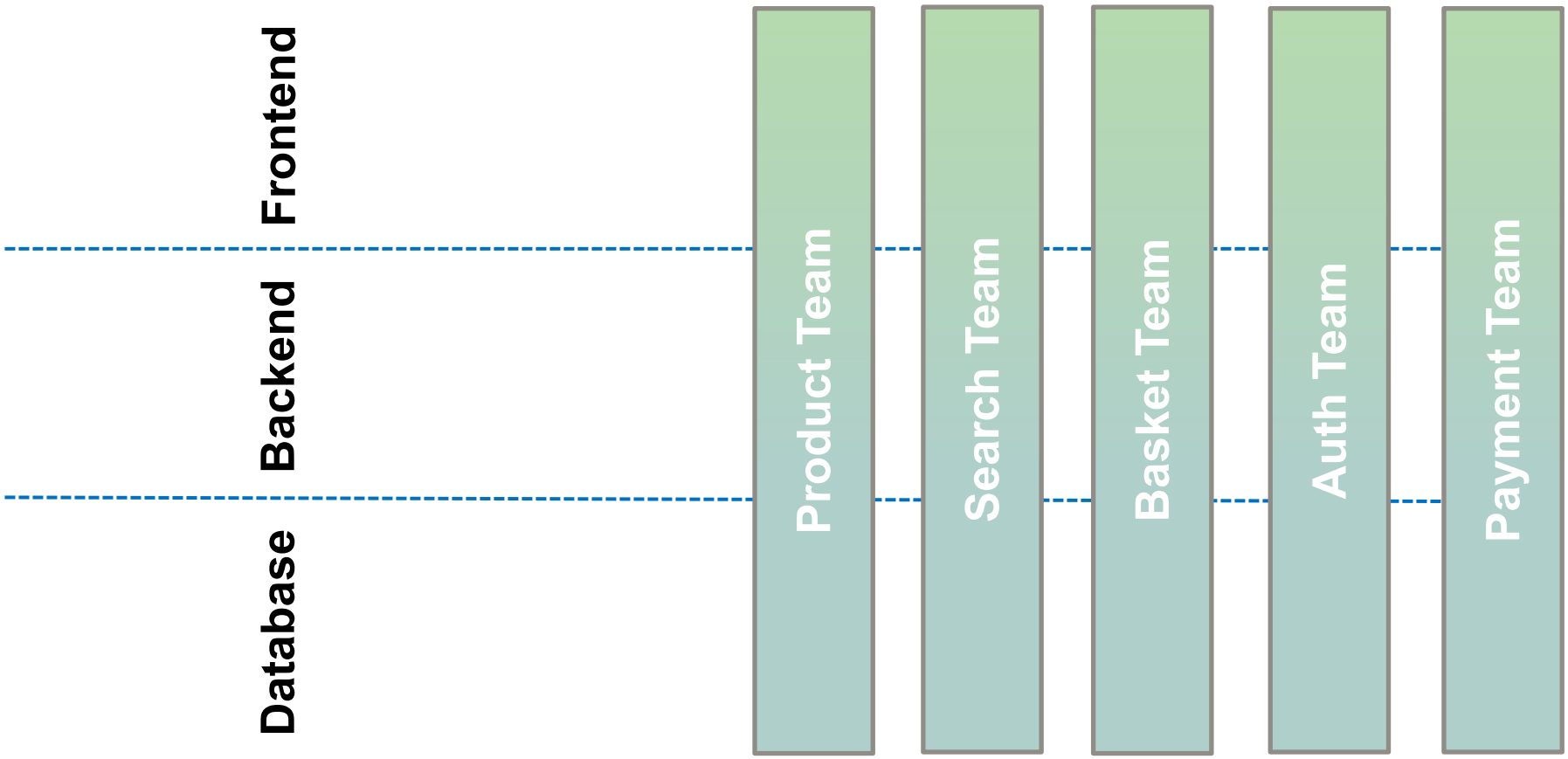


The fullstack approach

- Full vertical separation

→ Developers are specialized and separated by business cases

FullStack Microservices



Note: Each service is still splittet into backend and frontend. This graphic should illustrate the team organization.

Comparison

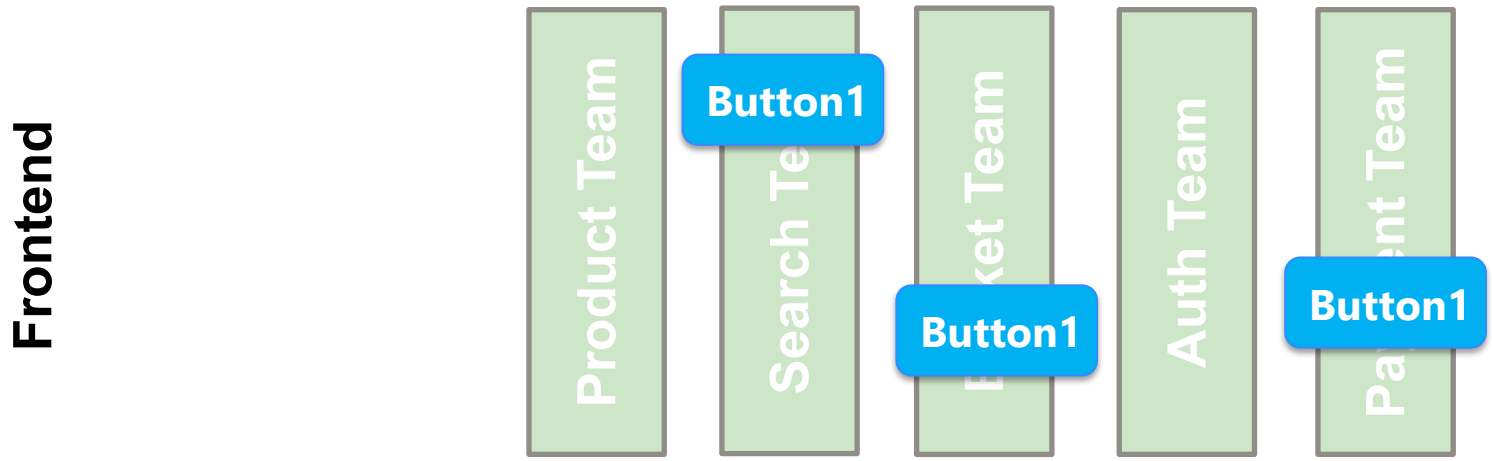
The classical or horizontal approach

- State of the art and less risky (from a technological perspective)
- Easier to find specialized developers
- One team, one user experience

The fullstack or vertical approach

- Knowledge transfer in your team (technological)
- Flexible developers (+ working fullstack is often engaging for developers)
- Pure focus on one service

Problems with a Microservice Architecture in a GUI

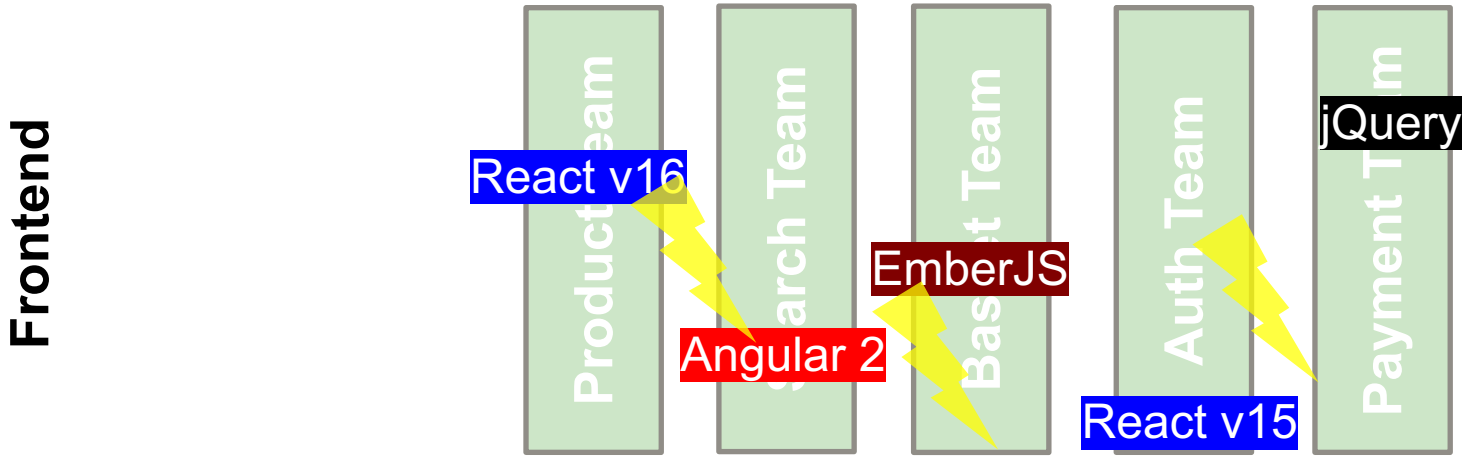


→ The same component is used by several teams.

- How to share it?
- Versioning?

WebComponents could be the answer for the future!

Problems with a Microservice Architecture in a GUI



→ The teams can't make their own choice for a technology stack or are at least very limited.

- In a productively scale it's almost impossible to combine different frontend frameworks and even different versions of the same framework (even for server side rendering)
- The point: In a dynamic gui the pages and components are depending on each other. They're highly interacting and need context data/states of other parts. But each framework has it's own solution for flowing state changes.

Styleguides

Styleguides

„A **style guide** (or **manual of style**) is a set of standards for the writing and design of documents, either for general use or for a specific publication, organization, or field. (It is often called a **style sheet**, though that term has other meanings.)" *Wikipedia*

Benefits

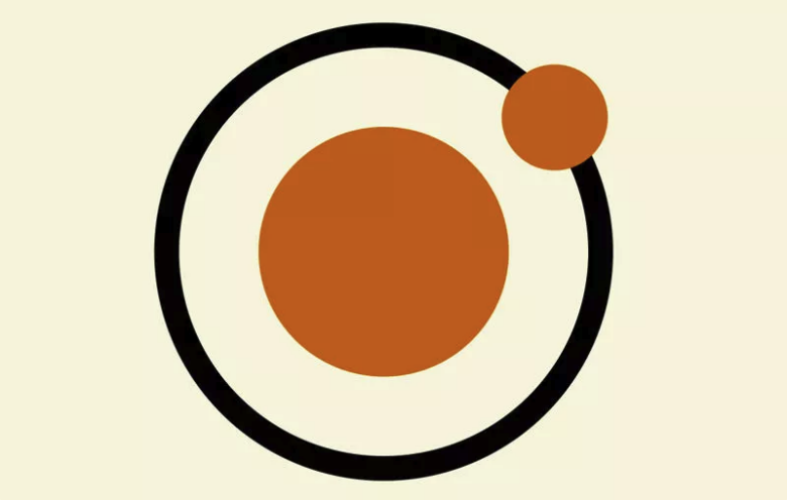
- Expandability
- Consistency
- Collaboration

Examples:

- Google Material Design: https://material.io/guidelines/
- Starbucks: https://www.starbucks.com/static/reference/styleguide

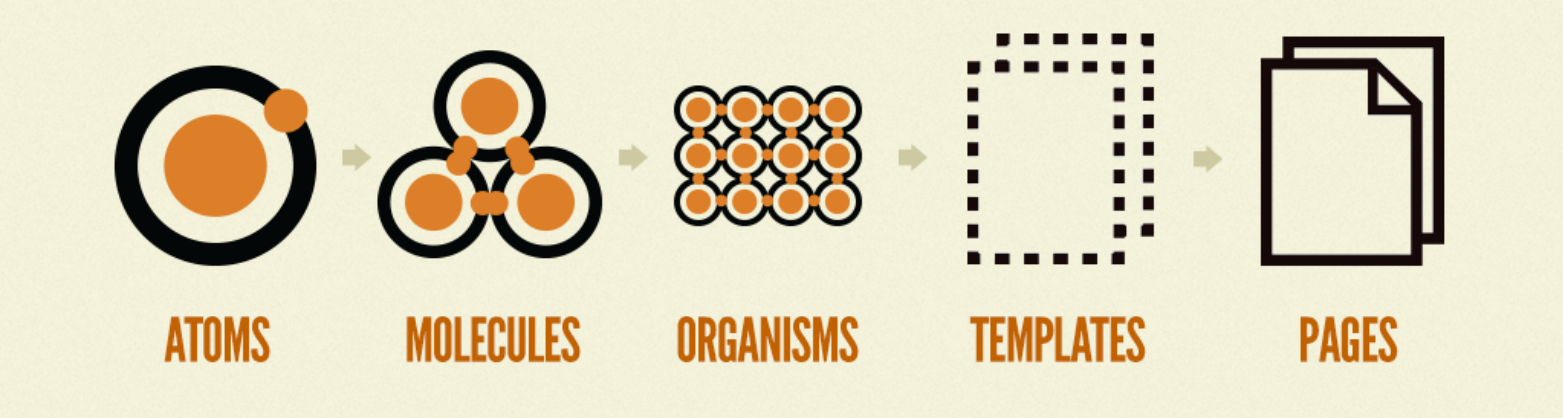
Atomic Design System

„A mental model to help us think of our user interfaces“ ~ Brad Frost



Styleguides

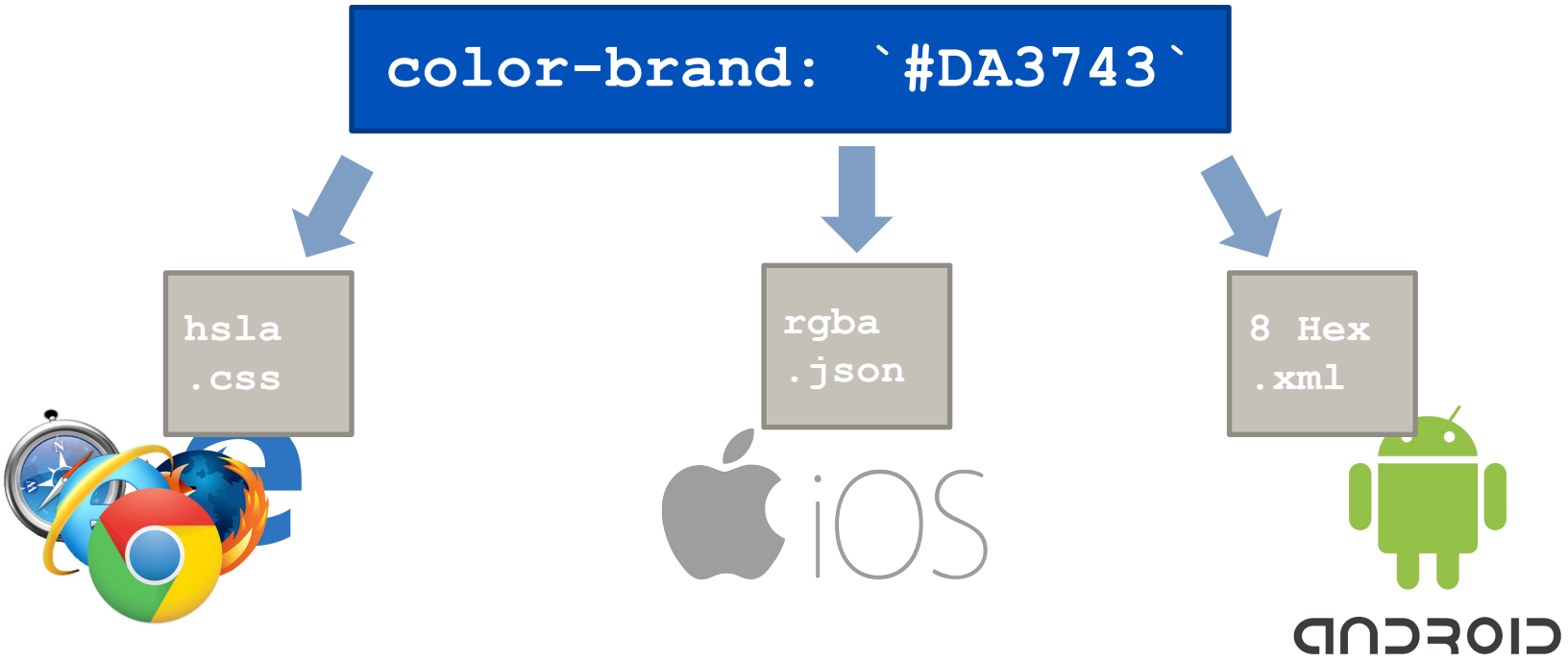
Atomic Design System



Styleguides

Atomic Design System

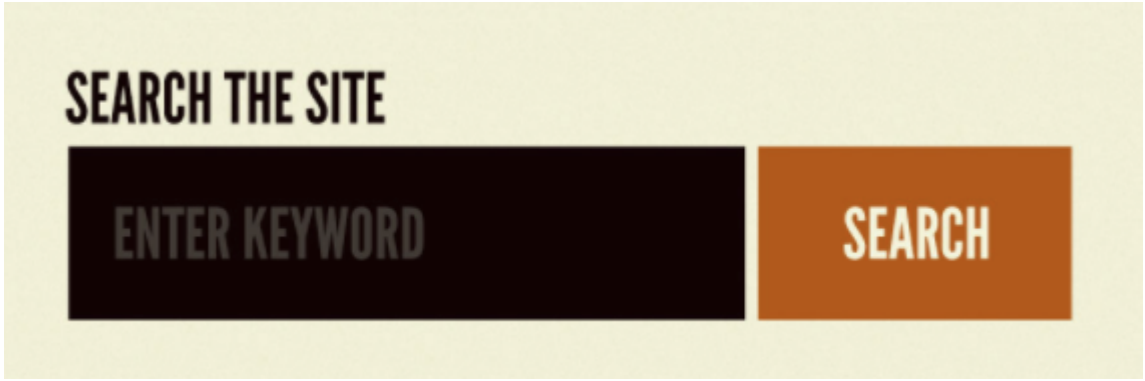
→ Atoms e.g. Colors, Spacings, Typography…



Styleguides

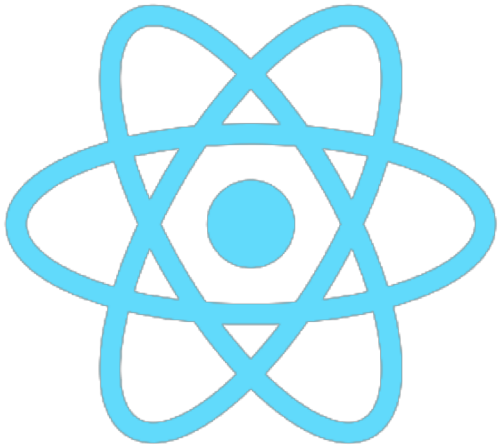
Atomic Design System

→ for example a molecule



Latest UI trends

- Web: The biggest players



React

ANGULAR

Latest UI trends

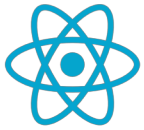
- Web: The rising star

Vue.js

Latest UI trends

- Web: Other frameworks and technologies you should keep in mind

polymer



React Native

ELECTRON

Latest UI trends

- Web:
  - WebComponents (https://www.webcomponents.org/)
  - Progressive Web Apps (PWA)
    → „Installable“ Websites