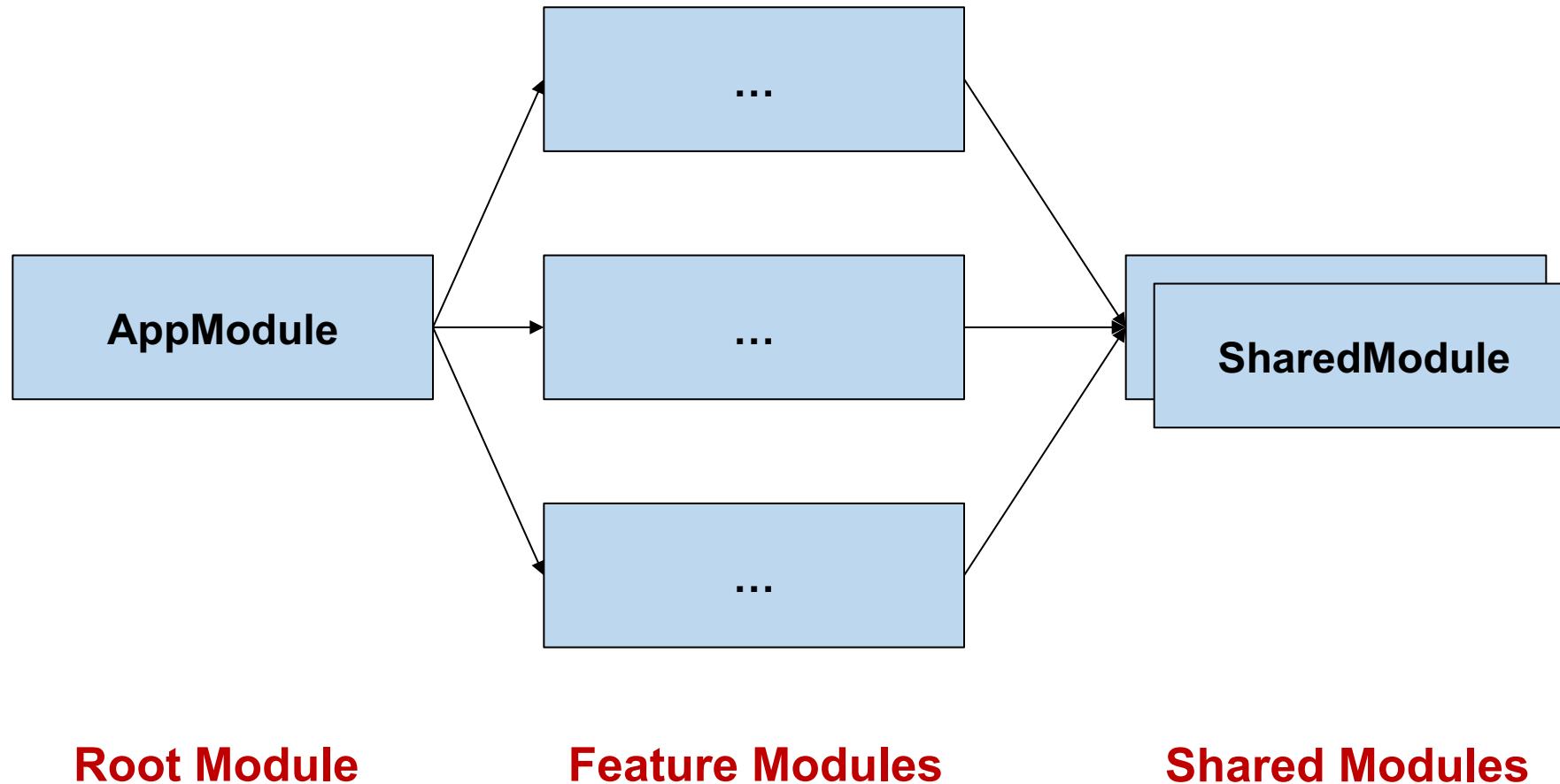




Architecture for Enterprise Applications

Alex Thalhammer

Typical Module Structure



Outline

- Nx Monorepos
- Strategic Design and DDD
- Micro Frontends
- Module Federation



Monorepos

Monorepo Structure

- ▶  node_modules
- ◀  projects
 - ▶  flight-admin
 - ▶  flight-api
 - ▶  flight-app
 - ▶  validation
- ▶  .gitignore
- ▶  angular.json
- ▶  package-lock.json
- ▶  package.json



Advantages

Everyone uses the latest versions

No version conflicts

No burden with distributing libs

Creating new libs: Adding folder

Experience: Successfully used at Google, Facebook, ...

Two Flavors

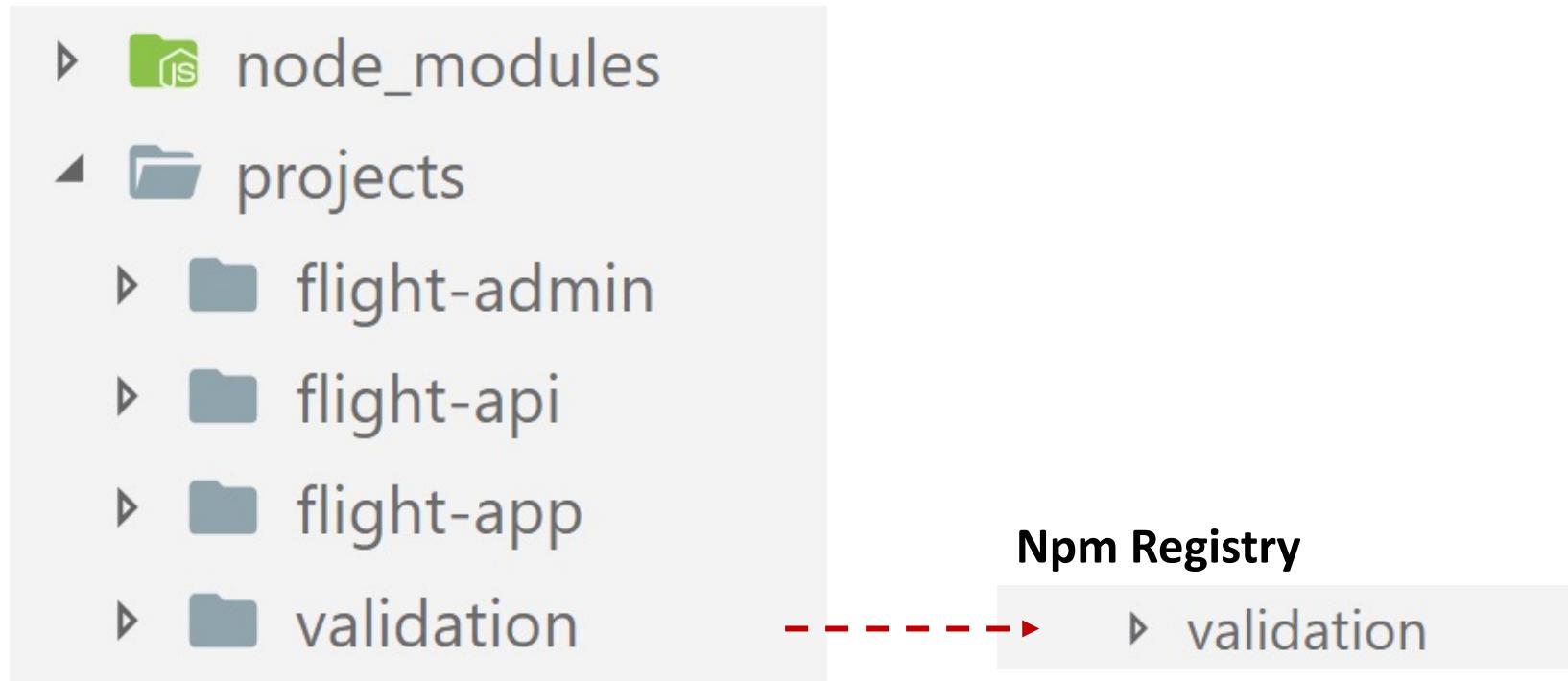
Project Monorepo

- Like Workspaces/Solutions in different IDEs

Company-wide Monorepo

- E. g. used at Google or Facebook

Moving back and forth



Tooling & Generator

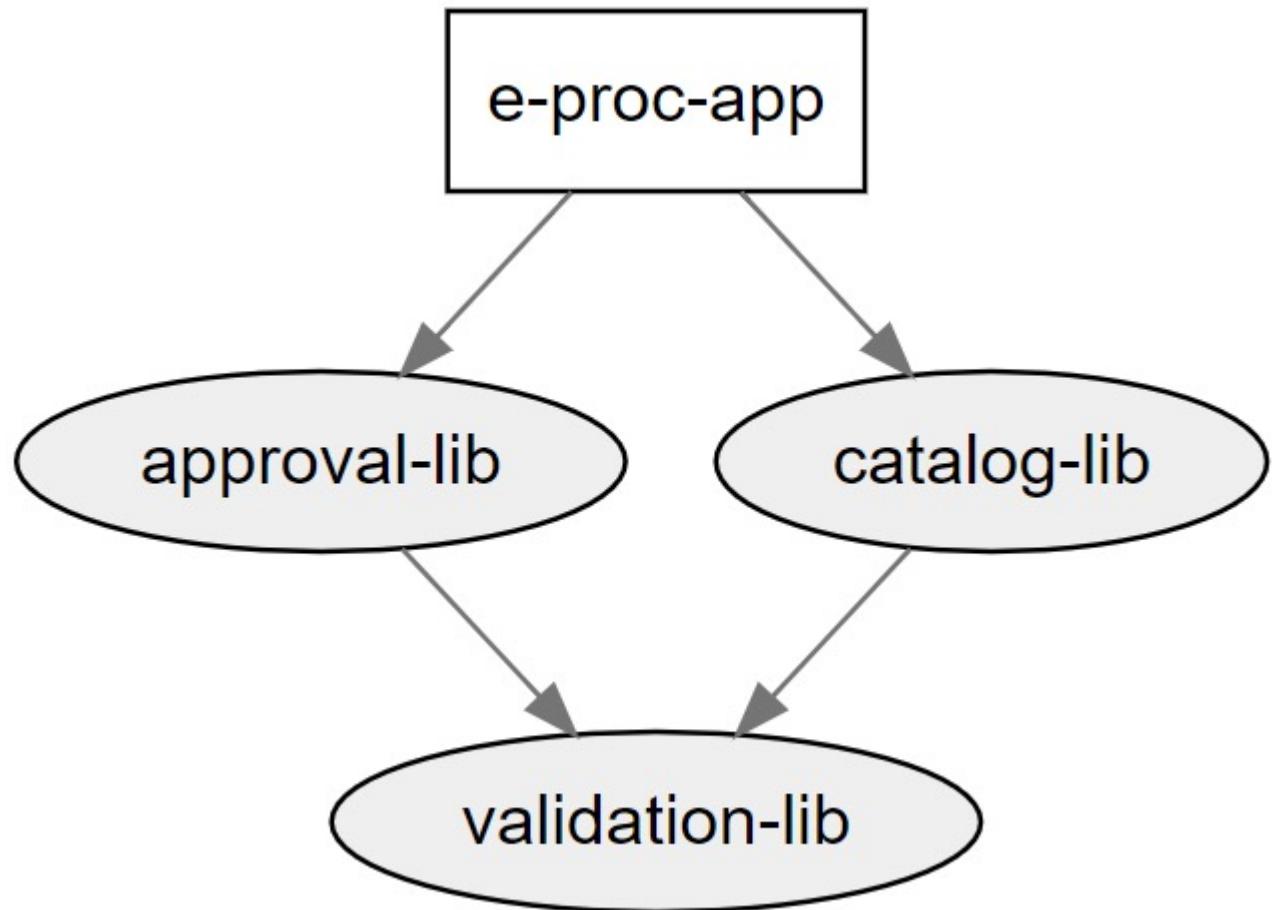
<https://nrwl.io/nx>



Nrwl Extensions for Angular

An open source toolkit for enterprise Angular applications.

Visualize Module Structure



Further Features of Nx

- Define folders for libraries
- Restrict which apps/libs can access which other libs
- Just recompile changed apps
- Visualize module structure and dependencies
- Scaffold Boilerplate

Creating a Workspace

```
npm install -g @angular/cli
```

```
ng new workspace
```

```
cd workspace
```

```
ng generate app my-app
```

```
ng generate lib my-lib
```

```
ng serve --project my-app
```

```
ng build --project my-app
```

Creating a Workspace

```
npm install -g @angular/cli
```

```
npm init nx-workspace workspace
```

```
cd workspace
```

```
ng generate app my-app
```

```
ng generate lib my-lib --buildable
```

```
ng serve --project my-app
```

```
ng build --project my-app
```

Lab

Libraries and Monorepo



DDD

in a nutshell

Domain-Driven DESIGN

Tackling Complexity in the Heart of Software



Eric Evans
Foreword by Martin Fowler

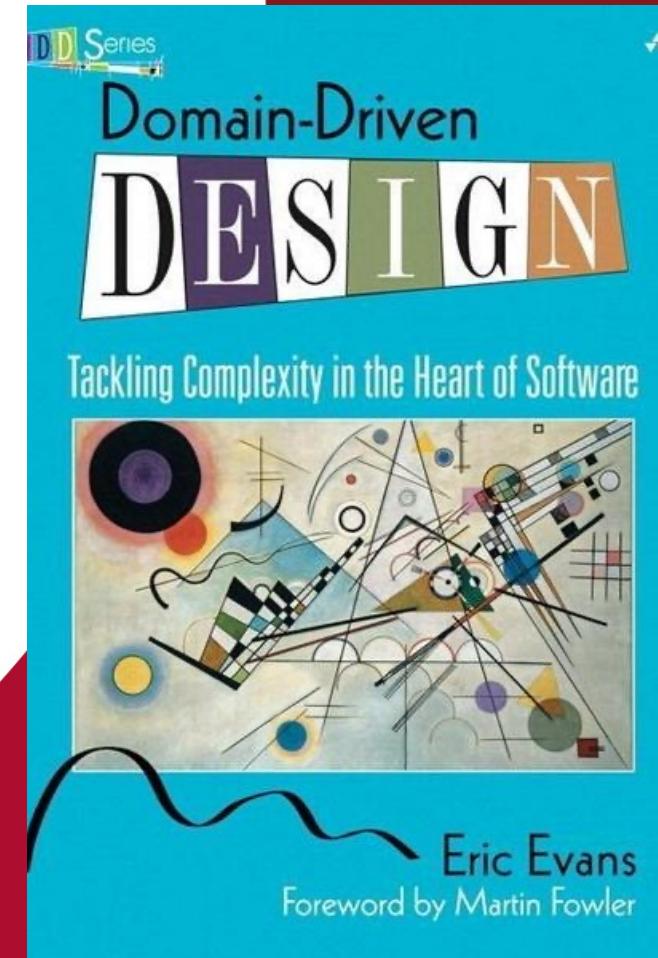
Methodology for
bridging the gap b/w
requirements and
architecture/ design



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

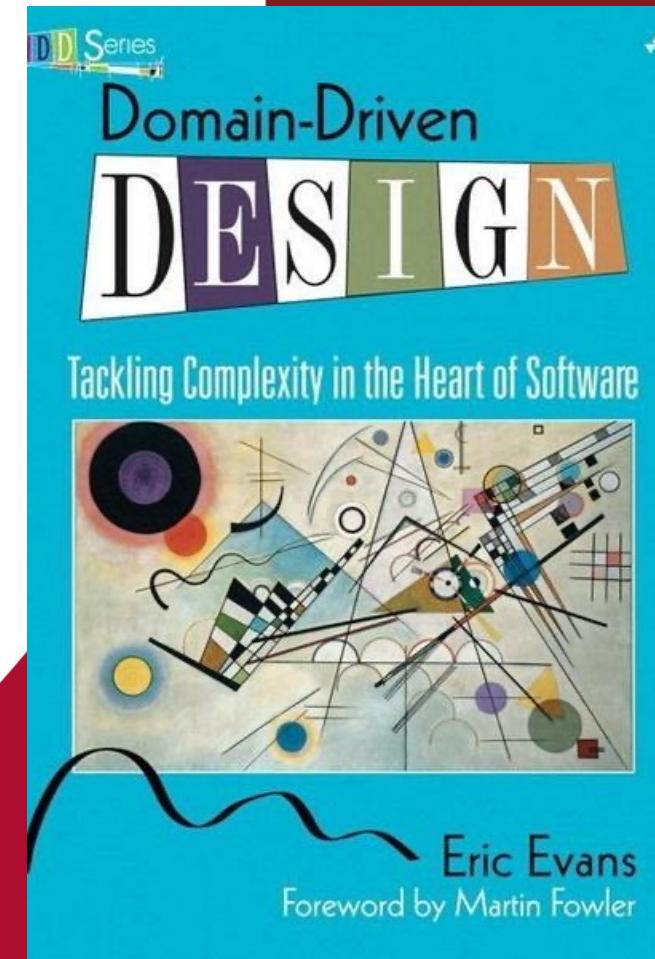


How to create sustainable frontend architectures with ideas from DDD?



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

How to create **sustainable** frontend architectures with **ideas from DDD?**



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

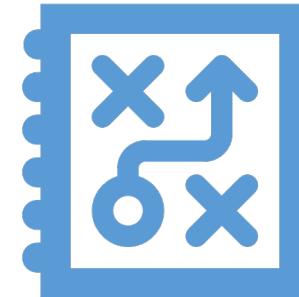
Domain Driven Design

Decomposing a System



Strategic Design

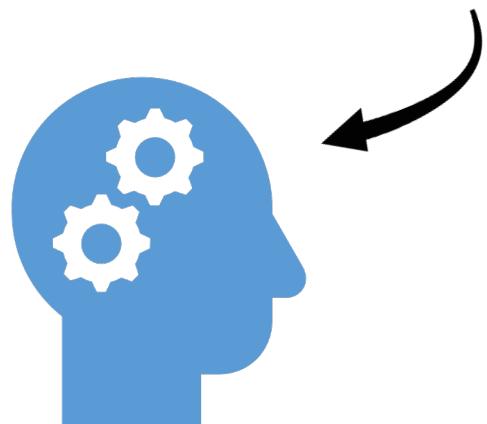
Design Patterns
& Practices



Tactical Design

Domain Driven Design

Decomposing a System



Design Patterns
& Practices



Strategic Design

Tactical Design

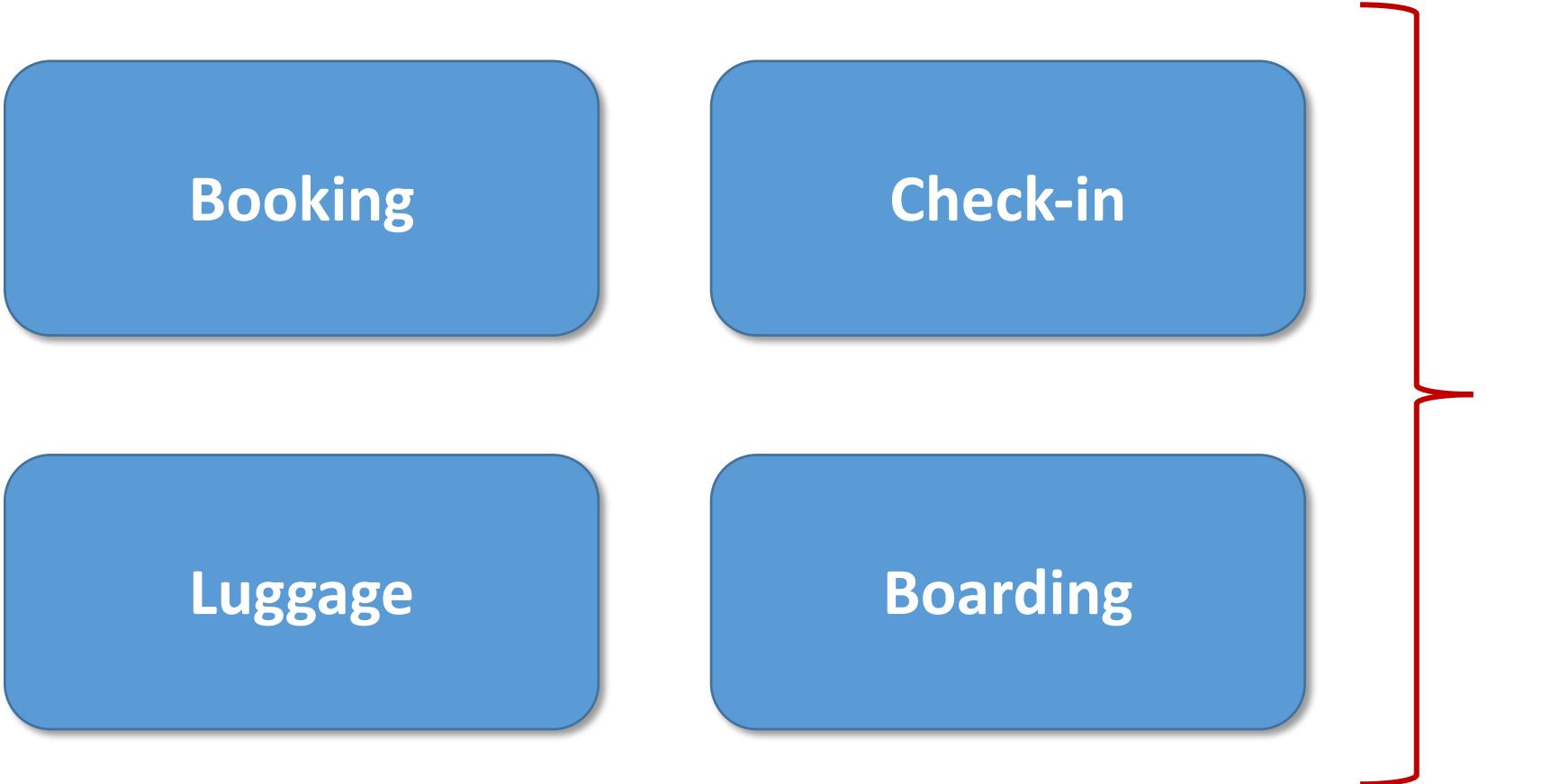


This is what Strategic DDD prevents

Example

Flight System

Example

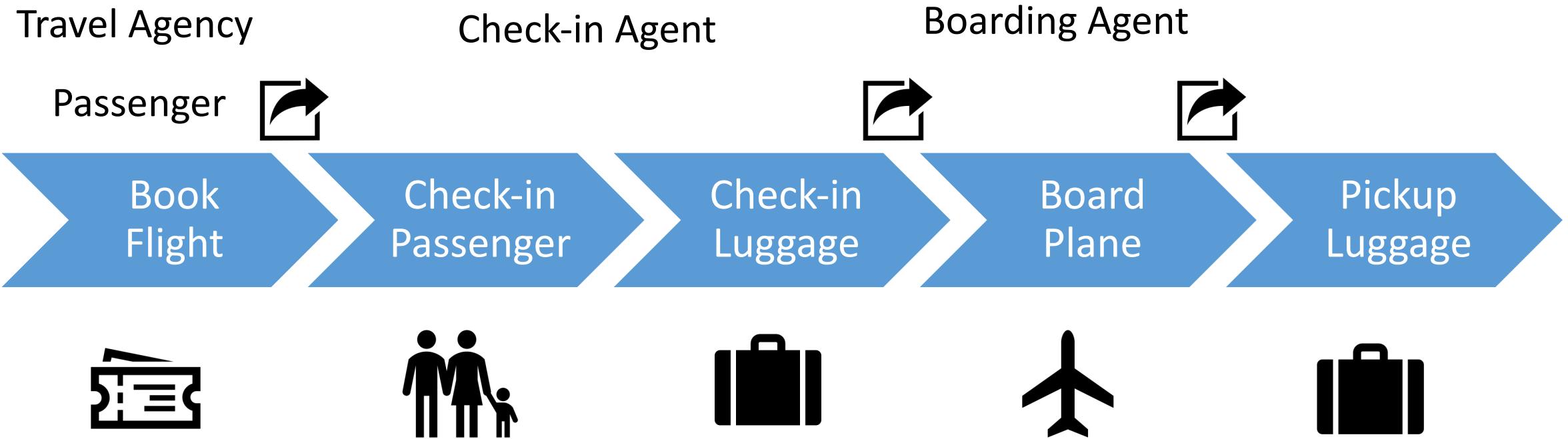


ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



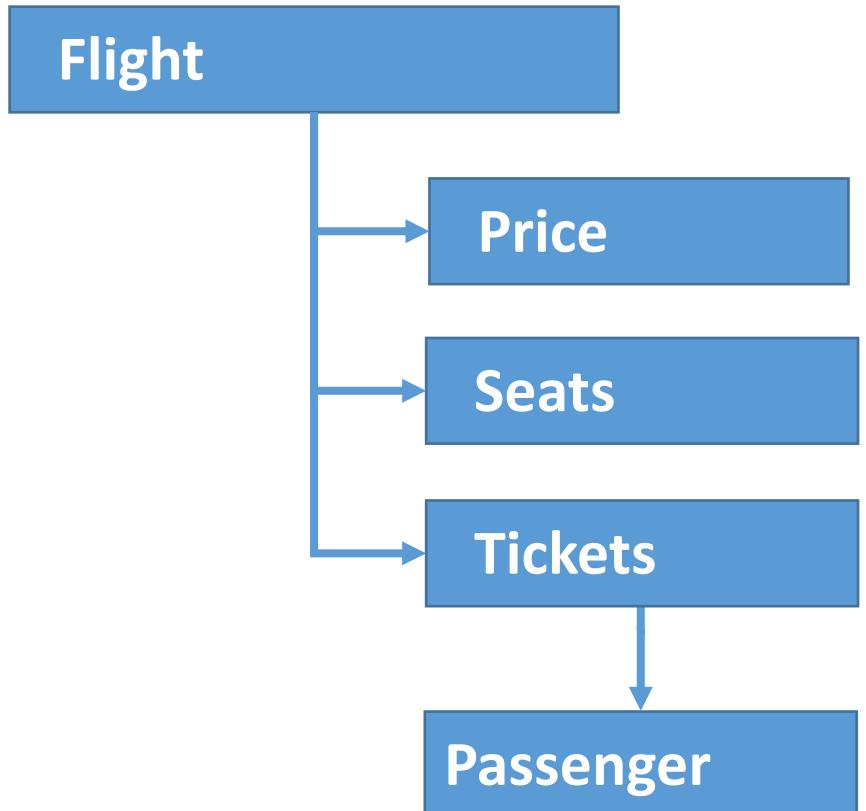
SOFTWARE
ARCHITECT

Finding Sub-Domains



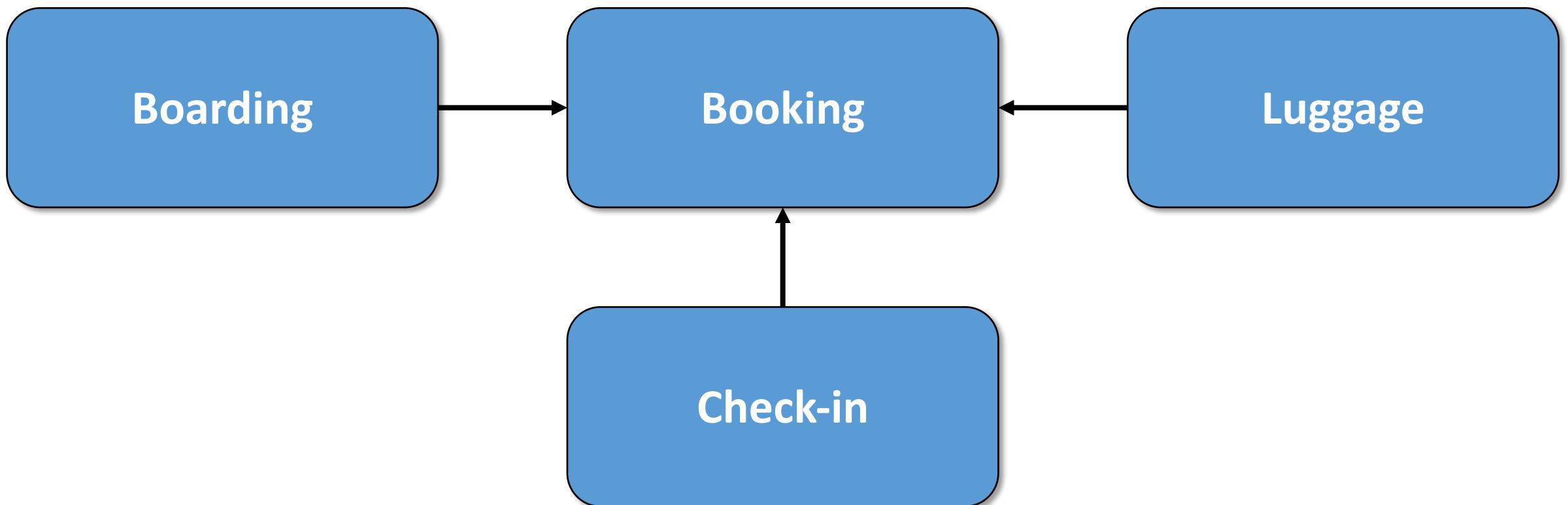
Booking

Boarding



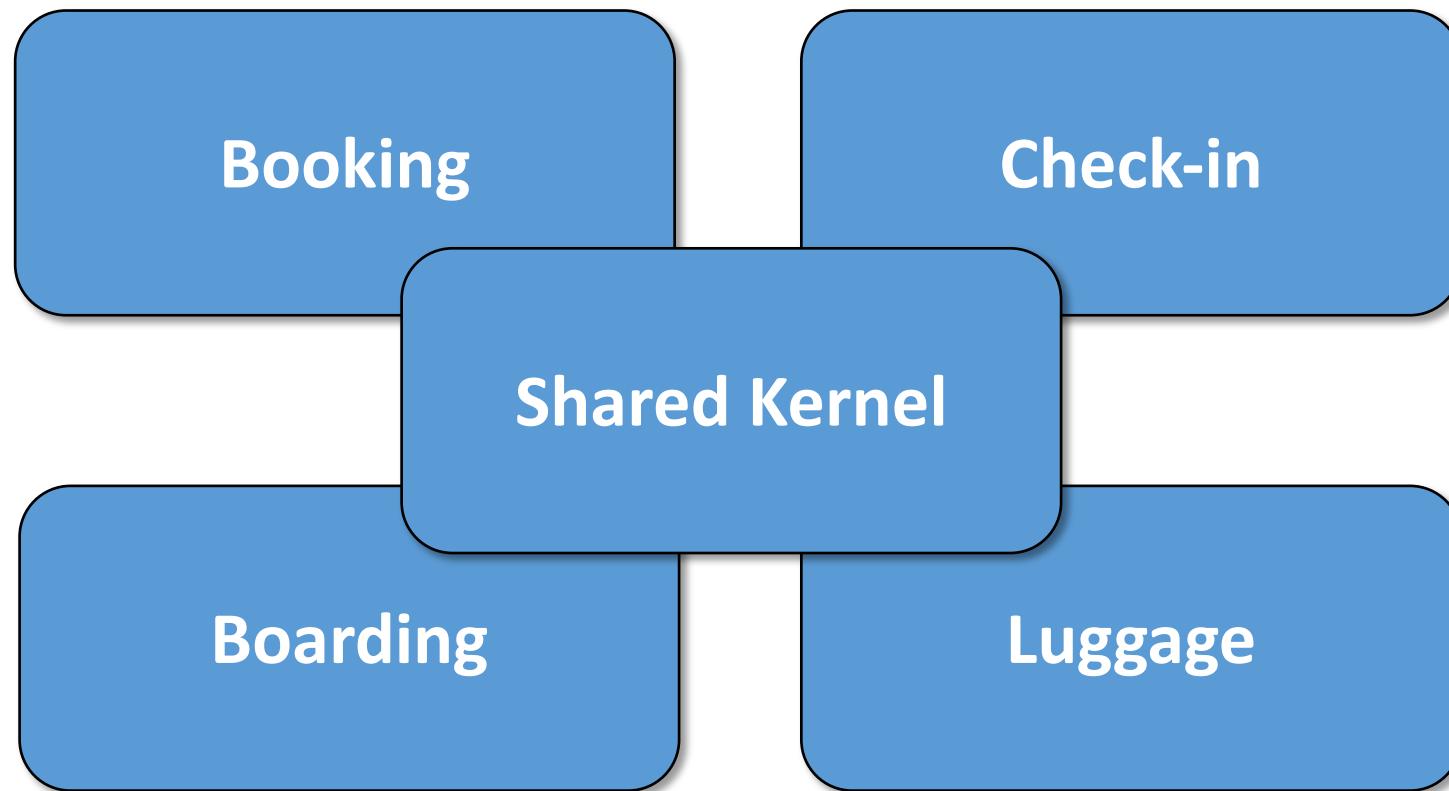
Bounded Context

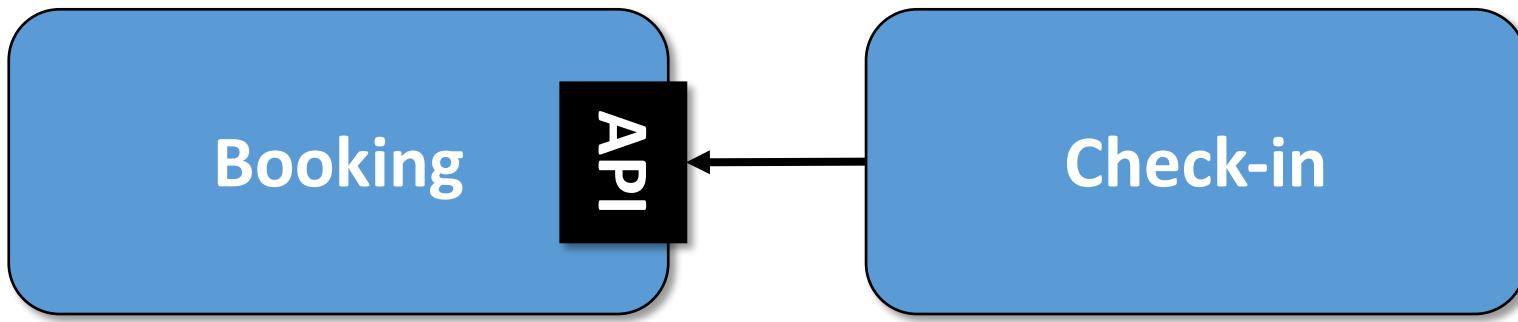
Context Map



Context Map

Responsibilities?
Breaking Changes?





Open-/Host-Service

Domain-Driven DESIGN

Tackling Complexity in the Heart of Software



Eric Evans
Foreword by Martin Fowler

Lots of approaches
for cross-domain
communication and
more ...

Shared Kernel (if really needed) & other libs

Smart
Comp.

Booking

Boarding

Shared

Feature

Feature

Feature

Feature

Feature

UI

UI

UI

UI

UI

UI

UI

UI

UI

Domain

Domain

Domain

Domain

Domain

Domain

Util

Util

Util

Util

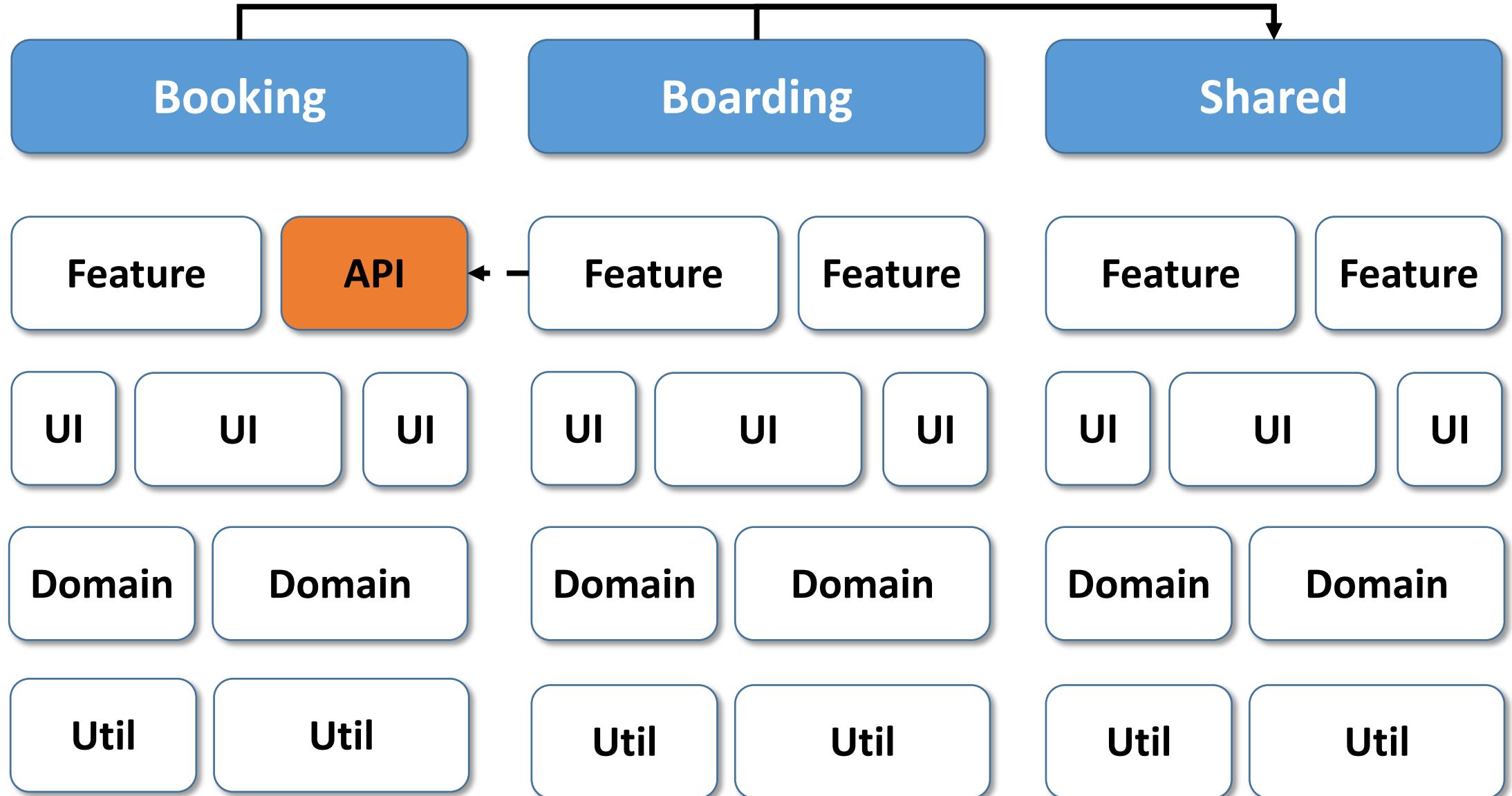
Util

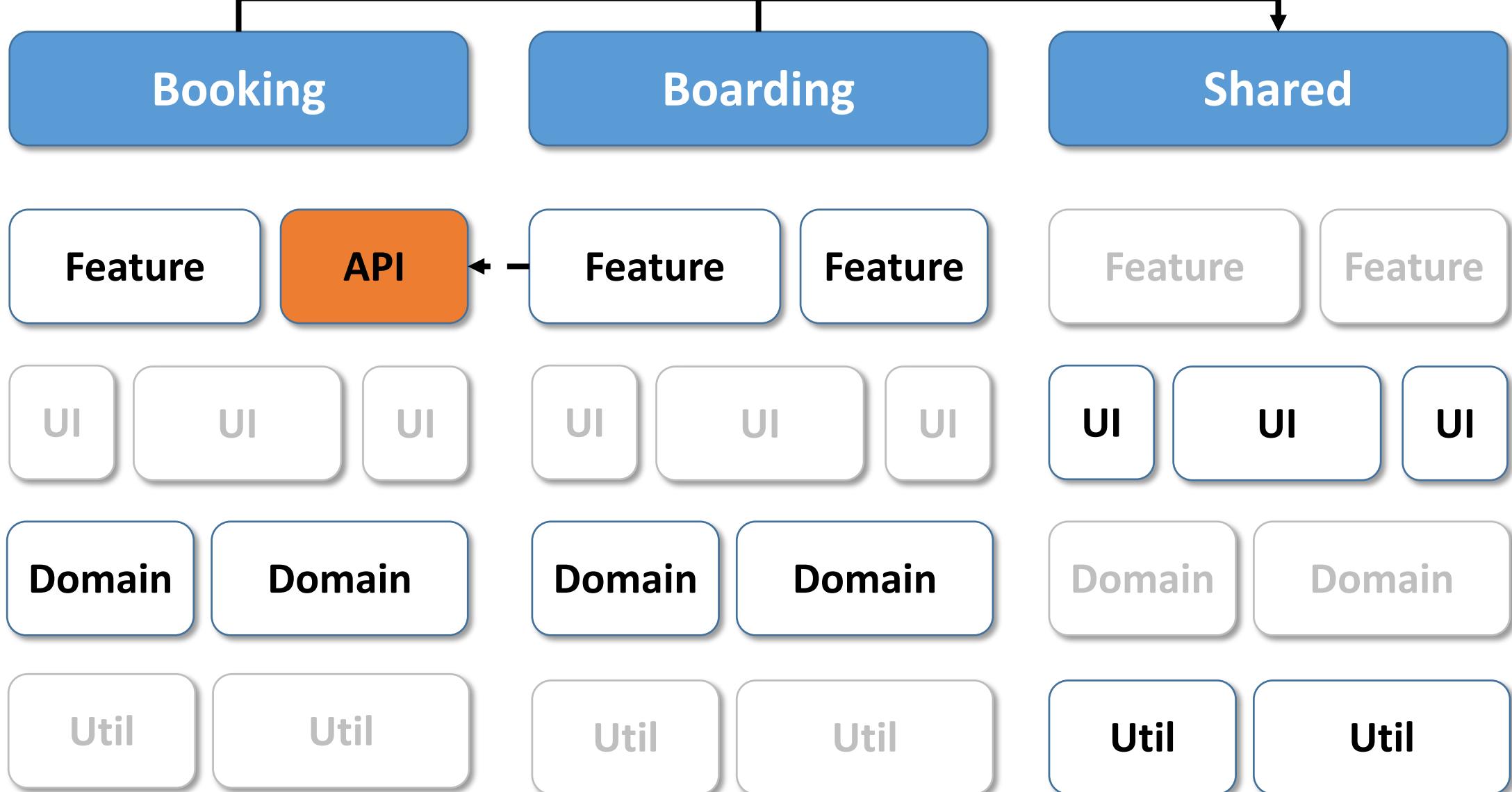
Util



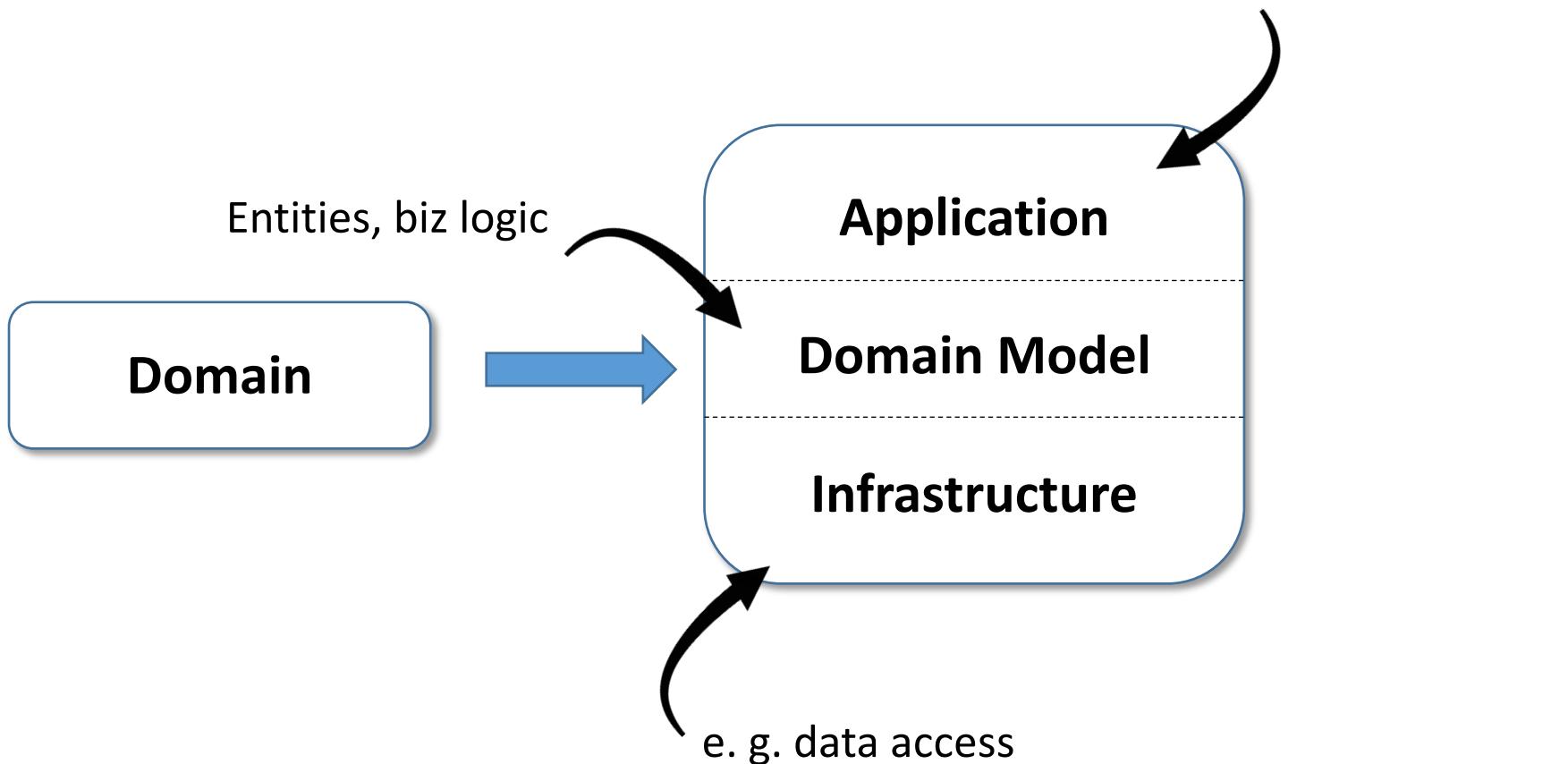
@ManfredSteyer

Enterprise Monorepo Patterns, Nrwl 2018: <https://tinyurl.com/y2jjxld7>





Isolate your domain!





Alternatives to layering

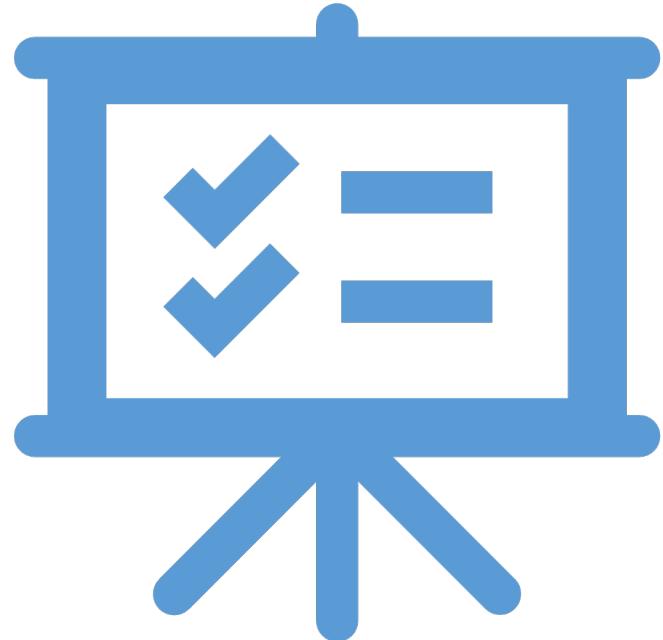
- e. g. Hexagonal Architecture, Clean Architecture
- Anyway: We need to **restrict access** b/w libraries



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT



Finegrained Libraries

- Unit of recompilation
- Unit of retesting
- Access restrictions
- Information Hiding
- Easy: Just `ng g lib ...`
- Future replacement for NgModules?

Categories for Libraries with Nx - I

From the [free e-book about Monorepo Patterns](#)

- **feature:** Implements use case controller (smart) components
- **ui:** Provides use case agnostic, thus reusable (dumb) presentational components
- **data-access:** Contains REST (or GraphQL) services that function as client-side delegate layers to server tier APIs
- **util:** Provides common utilities/services/helper functions

Categories for Libraries with Nx - II

NX access restrictions between libs

E.g. a “util” lib is not able to depend on a “feature” lib (or even an app)

Categories for Libraries with DDD

In addition, we're also using the following categories:

- **shell**: For an application that contains multiple domains, a shell provides the entry point for a domain
- **api**: Provides functionalities exposed for other domains
- **domain**: Domain logic like calculating additional expanses

Nx type:tags for Libraries incl. DDD

e2e

app / shell

feature / api

ui

data-access

domain(-logic)

util

Lab

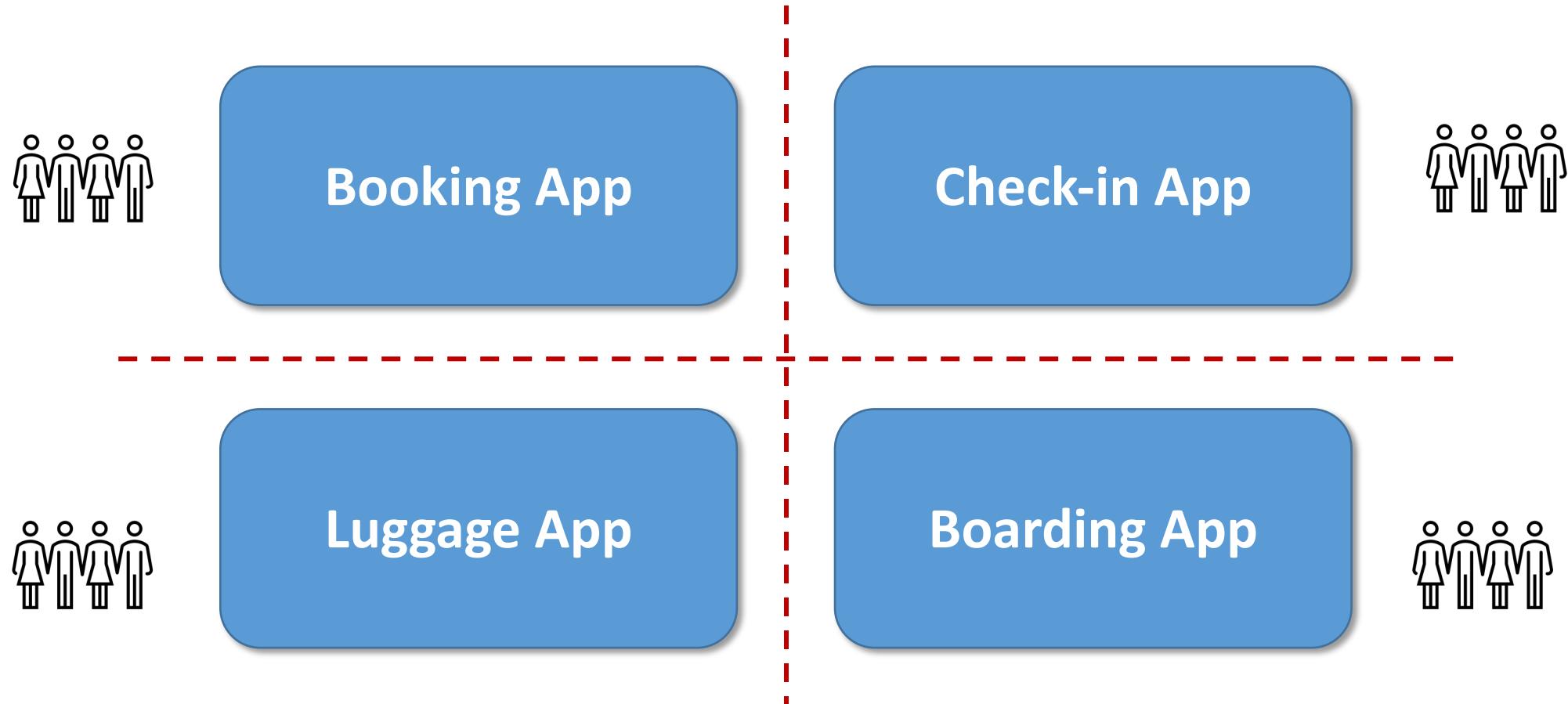
Nx DDD



Micro Frontends?

Short outlook

Microfrontends



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

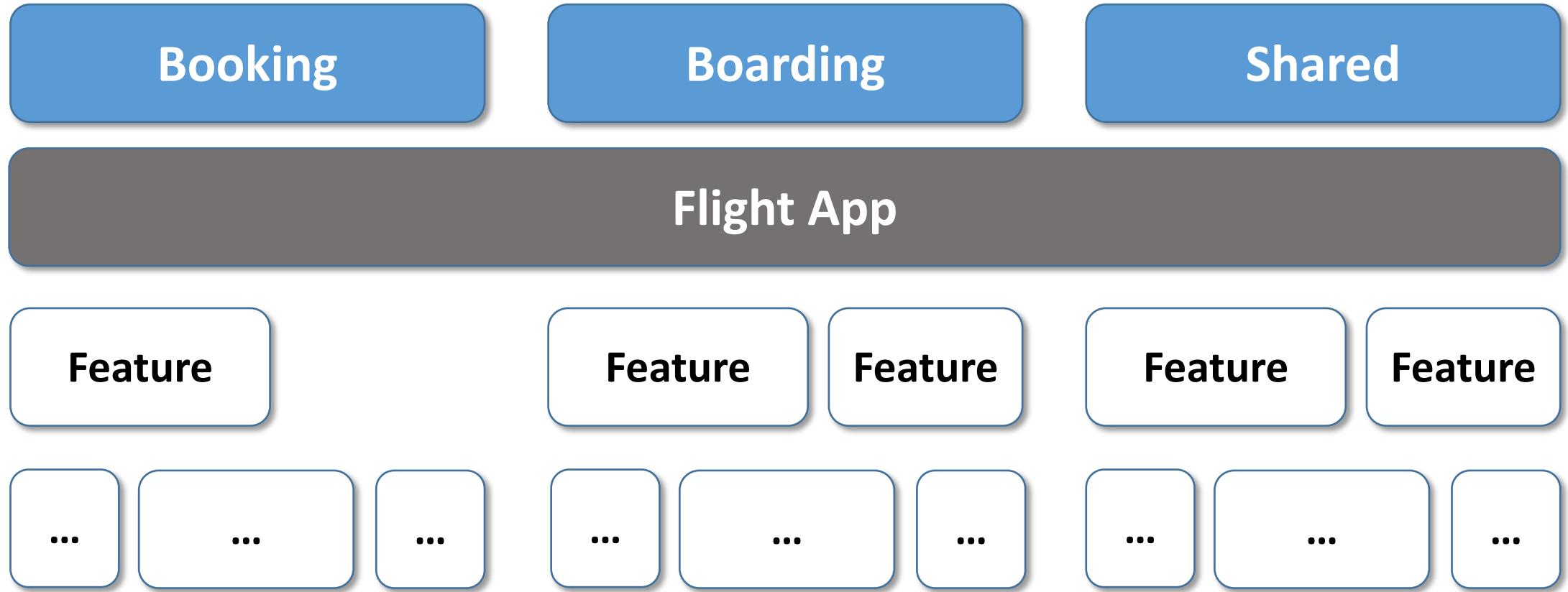


SOFTWARE
ARCHITECT

Microfrontends
are first and foremost
about **scaling teams!**



Deployment Monolith



Microfrontends

Booking

Boarding

Shared

Booking App

Boarding App

Feature

Feature

Feature

Feature

Feature

...

...

...

...

...

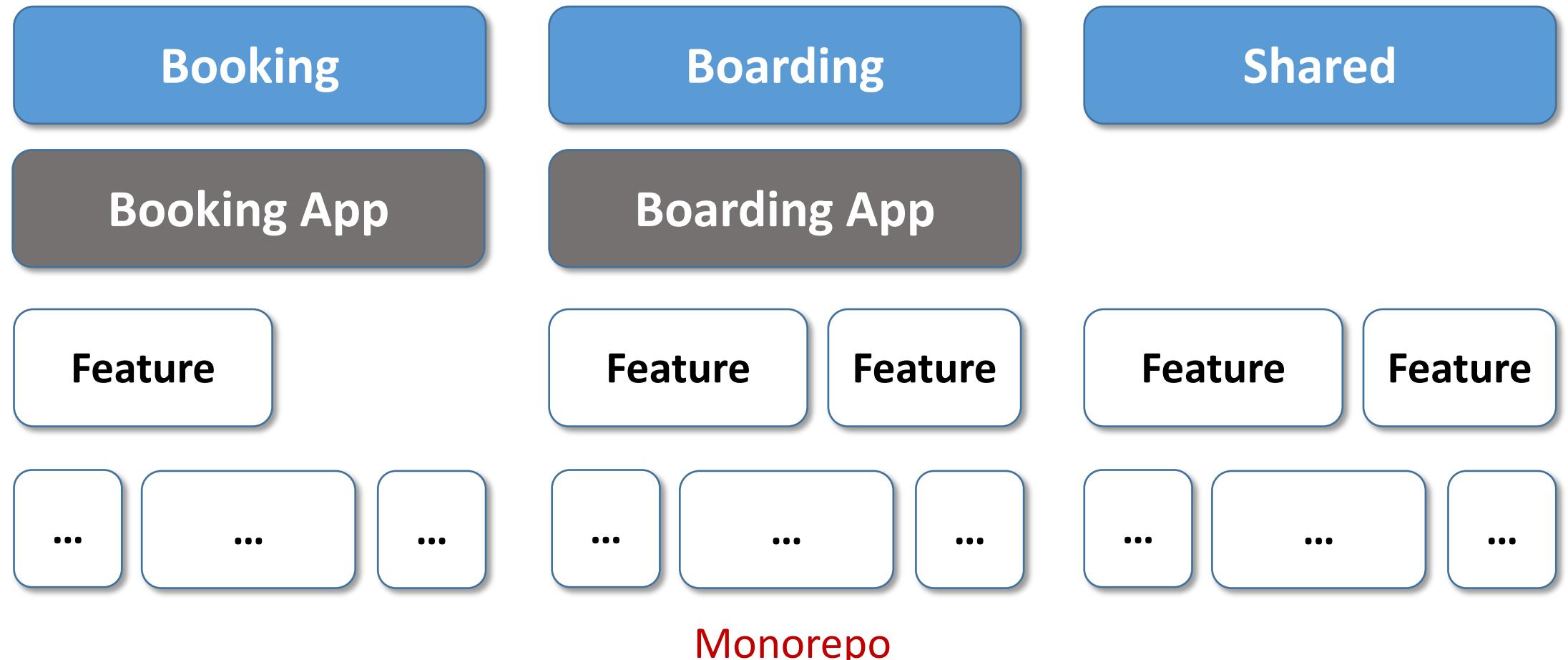
...

...

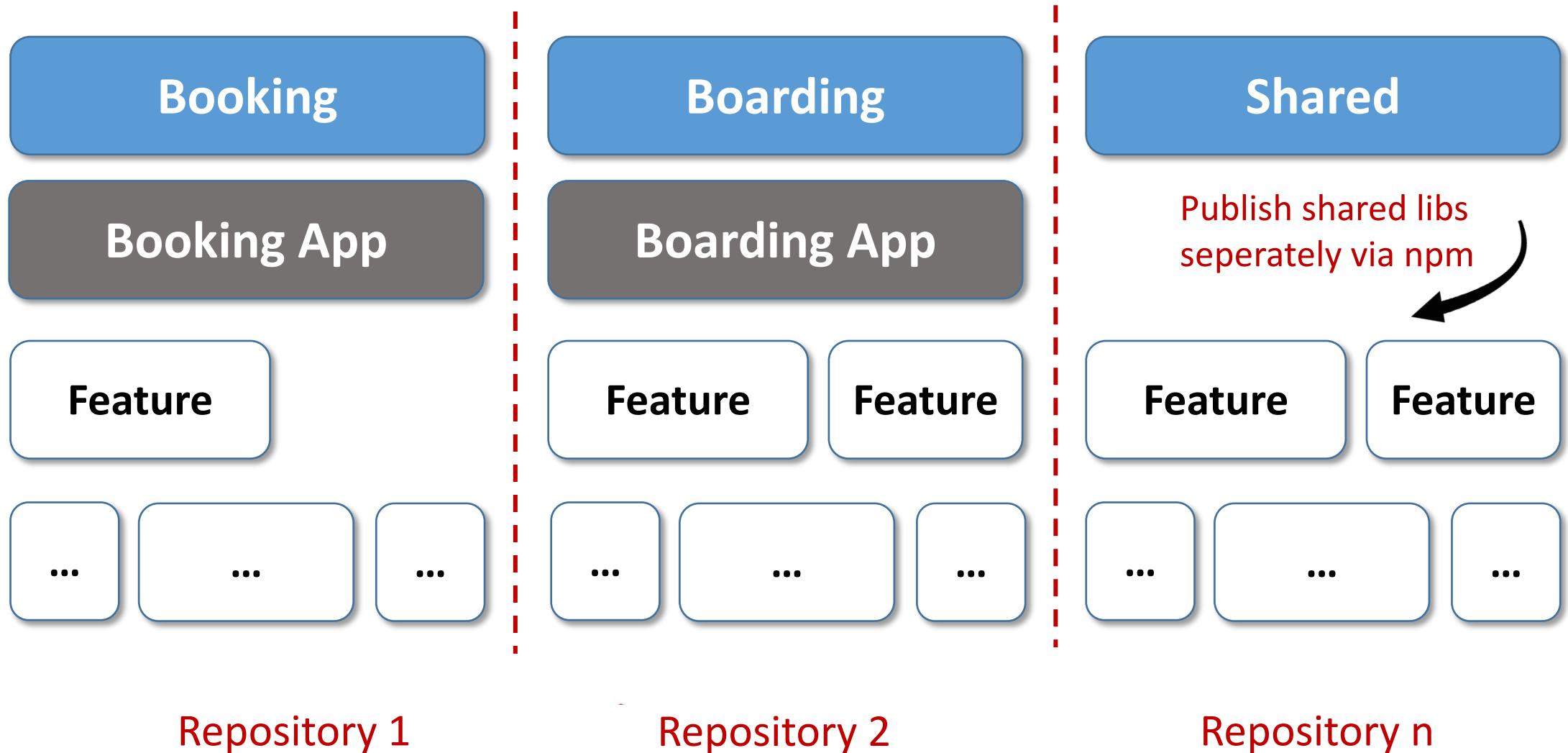
...

...

Option 1: One App per Domain



Option 2: One Monorepo per Domain



Benefits

Autonomous Teams

Separate Development

Separate Deployment

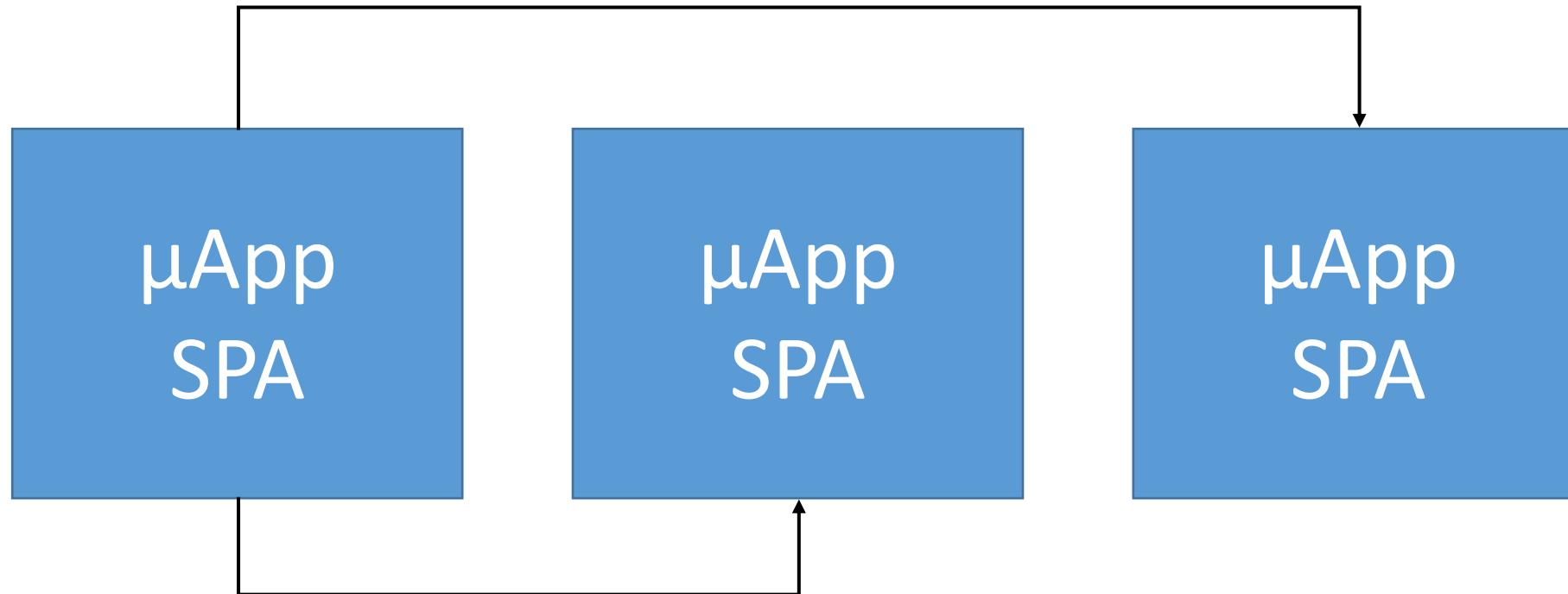
Own architecture decisions

Own technology decisions



Integration via
Hyperlinks

UI Composition w/ Hyperlinks



Word Online

Document - Auf OneDrive gespeichert.

Manfred S... Abmelden

DATEI START EINFÜGEN SEITENLAYOUT ÜBERPRÜFEN ANSICHT Was möchten Sie tun? IN WORD BEARBEITEN

Rückgängig Einfügen Zwischenablage

Calibri (Textkörper) 16 AaBbCc Standard AaBbCc Kein Leerra... Überschrift 1

Schriftart Absatz Formatvorlagen Bearbeiten

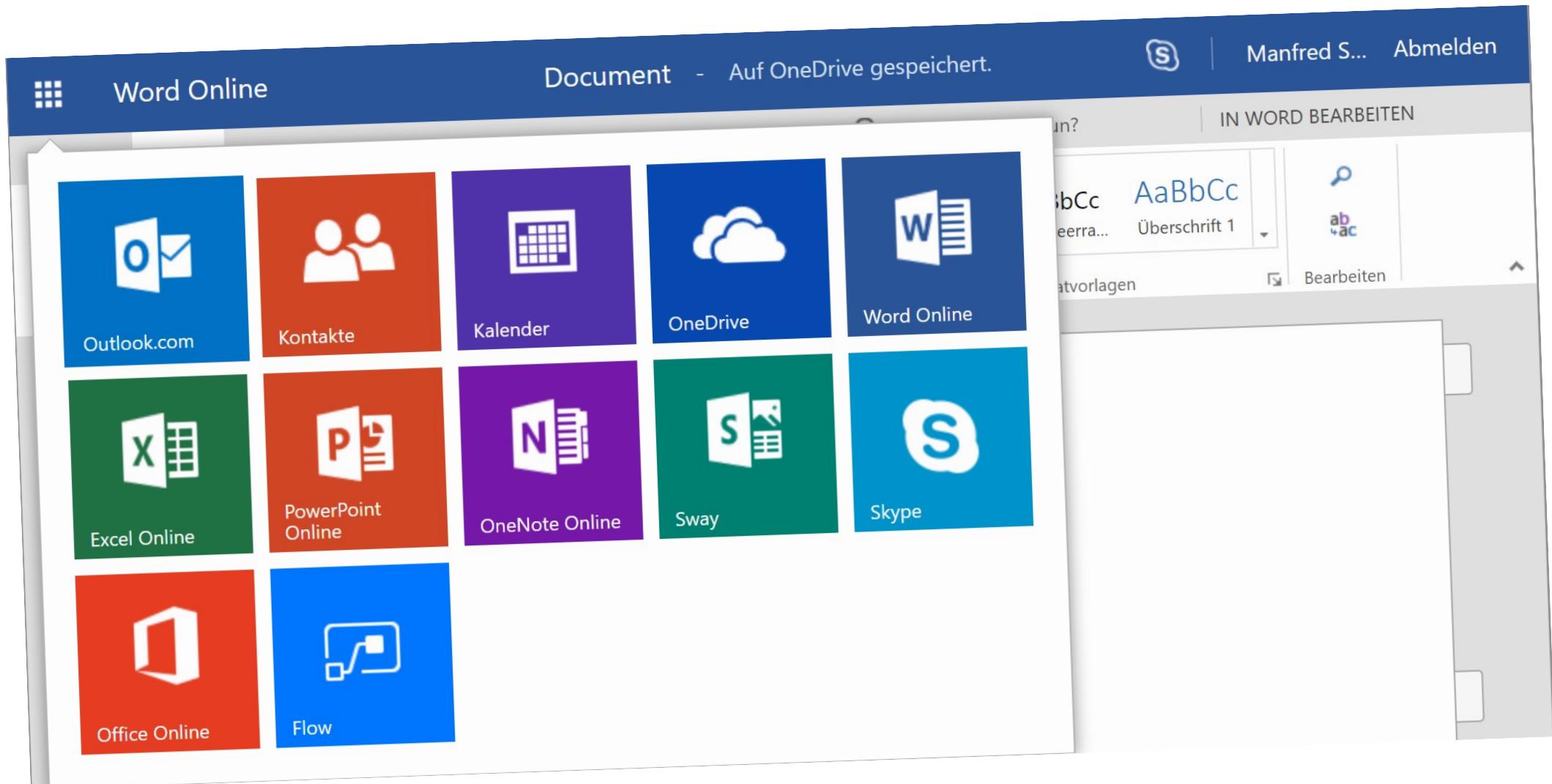
Hello World!



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT



ANGULAR
ARCHITECTS

INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

A large, light-colored conical shell, resembling a giant snail shell, sits on a sandy beach. The shell has a textured, spiral pattern and a pointed apex. In the background, the ocean with white-capped waves meets a clear blue sky.

Integration via
Shell

Providing a (SPA based) Shell (aka Host)

Shell / Host

μApp

μApp

μApp

Lazy Loading?

- This is more than lazy loading
- This is about loading part **NOT KNOWN** at runtime

Providing a (SPA based) Shell

- iframes
- Bootstrapping several SPAs
 - + Lazy Loading



Loading MicroApps

```
<script src="micro-app.bundle.js"></script>
```

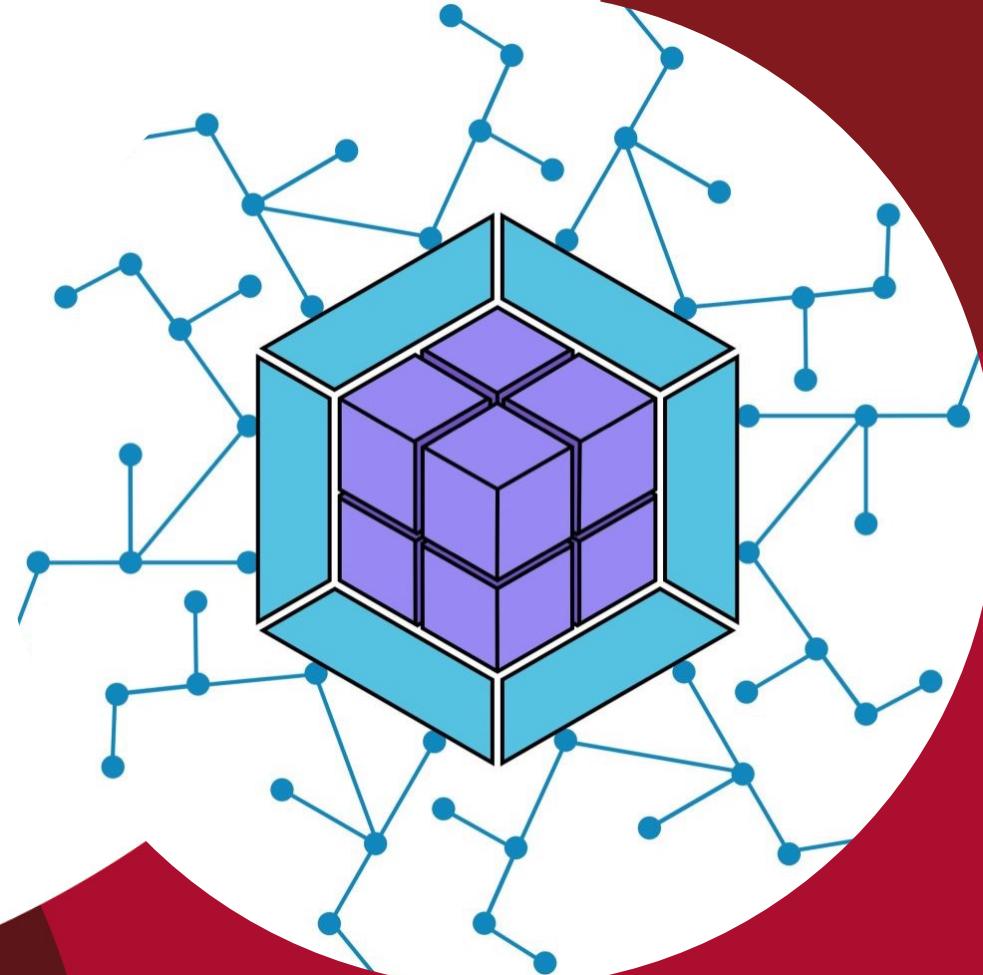
```
<micro-app></micro-app>
```

Loading Separately Compiled SPAs

```
const script = document.createElement('script');
script.src = 'assets/micro-app.bundle.js';
document.body.appendChild(script);

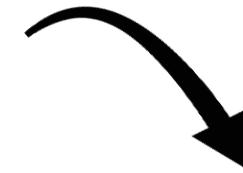
const clientA = document.createElement('client-a');
clientA['visible'] = true;
document.body.appendChild(clientA);
```

Webpack 5 Module Federation



Idea

Does not work with
webpack/ Angular CLI



```
const Component = import('http://other-app/xyz')
```

Even lazy parts must be
known at compile time!



Webpack 5 Module Federation

Shell (Host)

```
import('mfe1/Cmp')  
  
// MapsUrls in  
// webpack config  
remotes: {  
  mfe1: "mfe1"  
}
```

Microfrontend (Remote)

```
// Expose files in  
// webpack config  
exposes: {  
  Cmp: './my.cmp.ts'  
}
```

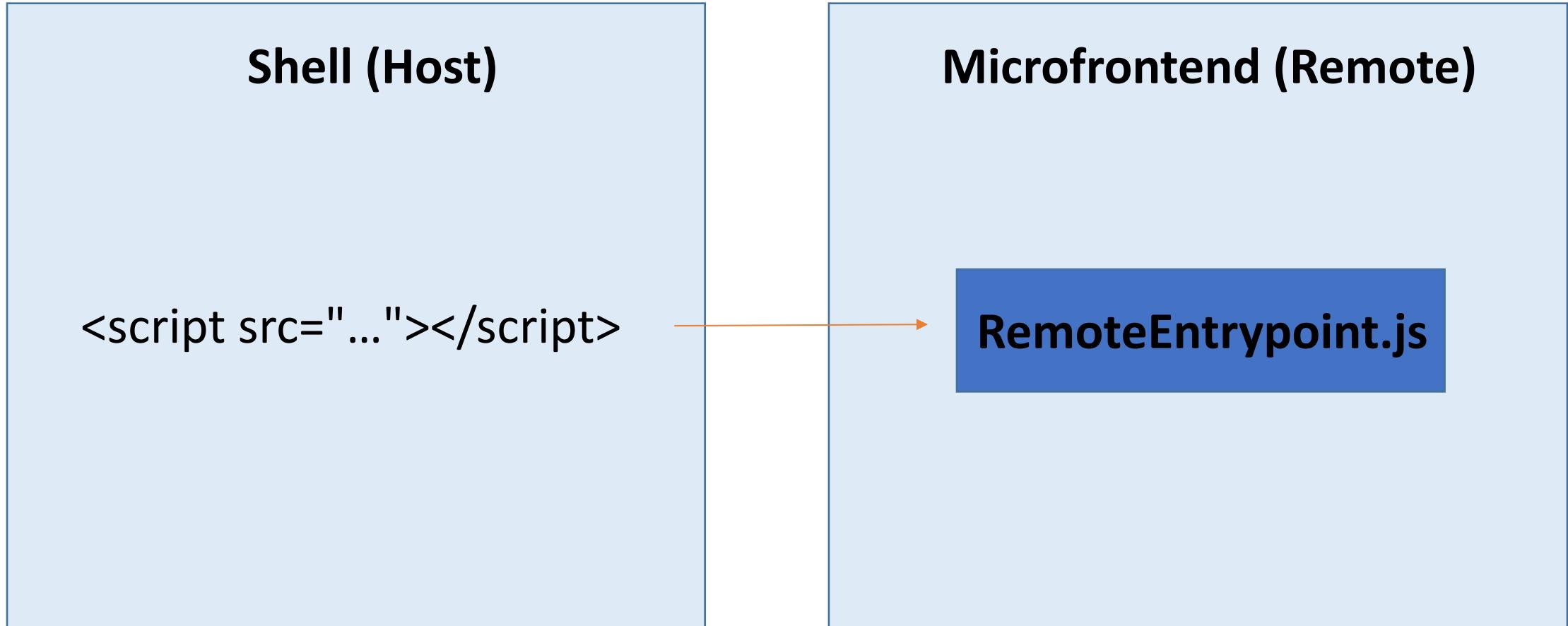


ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

How to Get the Microfrontend's URL?



How to Share Libs?

Shell (Host)

```
shared: [  
  "@angular/core", "..."  
]
```

Microfrontend (Remote)

```
shared: [  
  "@angular/core", "..."  
]
```



ANGULAR
ARCHITECTS

INSIDE KNOWLEDGE



Dealing with Version Mismatches



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Default Behavior

Selecting the highest compatible version

10.0 

10.1 

Default Behavior

Conflict: No highest compatible version

11.0 ✓

10.1 ✓

Example

- Shell: my-lib: ^10.0
- MFE1: my-lib: ^10.1
- MFE2: my-lib: ^9.0
- MFE3: my-lib: ^9.1

Result:

- Shell and MFE1 share ^10.1
- MFE2 and MFE3 share ^9.1

Configuring Singletons

```
shared: {  
  "my-lib": {  
    singleton: true  
  }  
}
```

11.0  10.1 

Configuring Singletons

```
shared: {  
  "my-lib": {  
    singleton: true,  
    strictVersion: true // Error instead of warning!  
  }  
}
```

11.0  10.1 

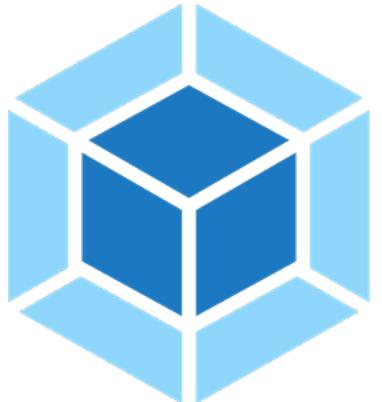
Relaxing Version Requirements

```
shared: {  
  "my-lib": {  
    requiredVersion: ">=1.0.1 <11.1.1"  
  }  
}
```

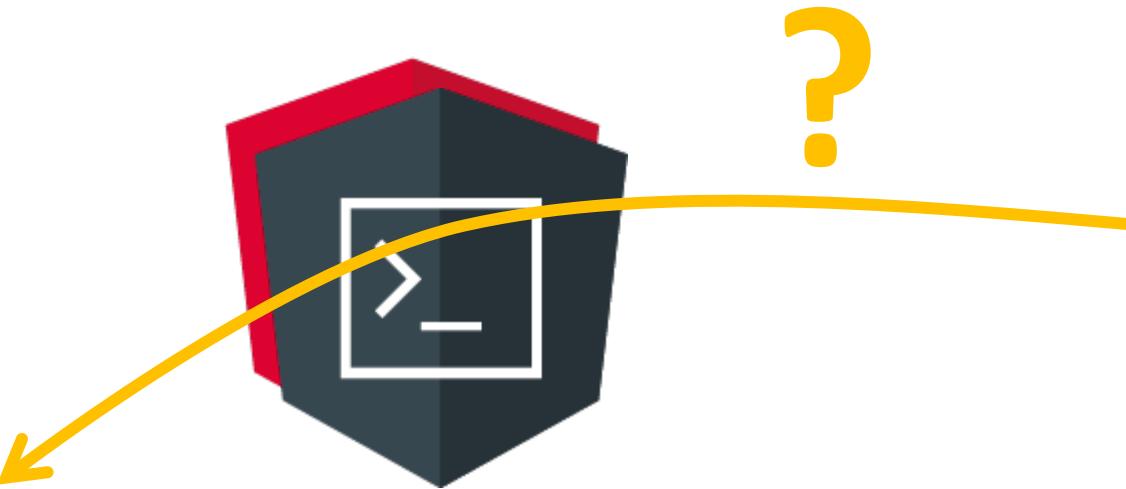
Federated Angular: Angular, CLI, & Module Federation



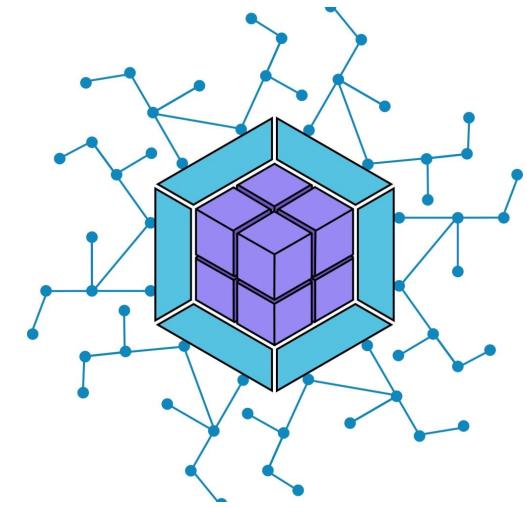
ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



webpack

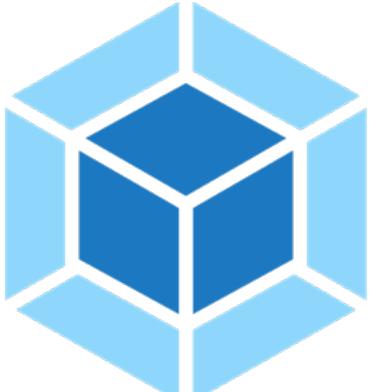


Angular CLI



Module Federation
Configuration

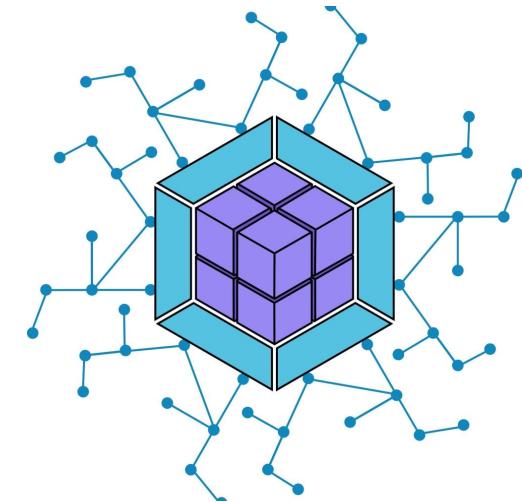
Custom Builder



webpack



Angular CLI



Module Federation
Configuration

@angular-architects/module-federation

1.0.2 • Public • Published 18 hours ago

 Readme

 Explore BETA

 3 Dependencies

Features 🔥

- Generates the skeleton for a Module Federation config.
- Installs a custom builder to enable Module Federation.
- Assigning a new port to serve (`ng serve`) several projects at once.



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

Usage

- 1) ng add @angular-architects/module-federation
- 2) Adjust generated configuration
- 3) ng serve

Module Federation Demo

Lab

Module Federation

Possible Roadmap



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Well ...

Webpack 5: final

@angular-architects/module-federation: final

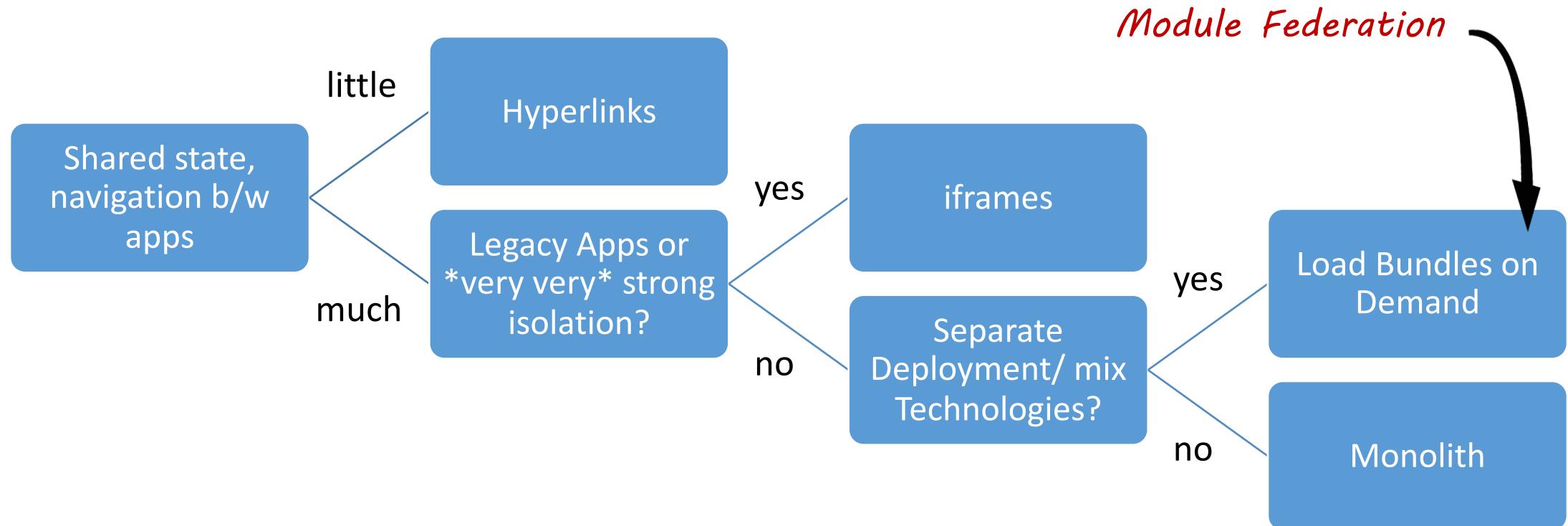
CLI 11: Experimental webpack 5 support (it works!)

CLI 12: Official webpack 5 support (just released!)

The background of the image is a dark, atmospheric room. The wall is covered in a repeating pattern of dark, ornate, floral or damask motifs. In the center of the frame, there is a row of several closed doors, all of which appear to be made of a light-colored wood or similar material. The doors are evenly spaced and extend from the foreground towards the background. The floor is made of dark, polished wooden planks that lead the eye towards the doors.

Choosing a Solution

Some General Advice



Summary

Libs: Subdividing big solution into tiny parts

Monorepo: No burden with distributing libs

Strategic Design: Cut system into loosely coupled domains

Nx: Access restrictions, visualization, incremental compilation/ testing, etc.

Microfrontends: Autarkic Teams, separate deployment, has challenges

Like this topic?

Read Manfred's book at

<https://www.angulararchitects.io/en/book/>

That's it for architecture

- Any questions left over? 😊