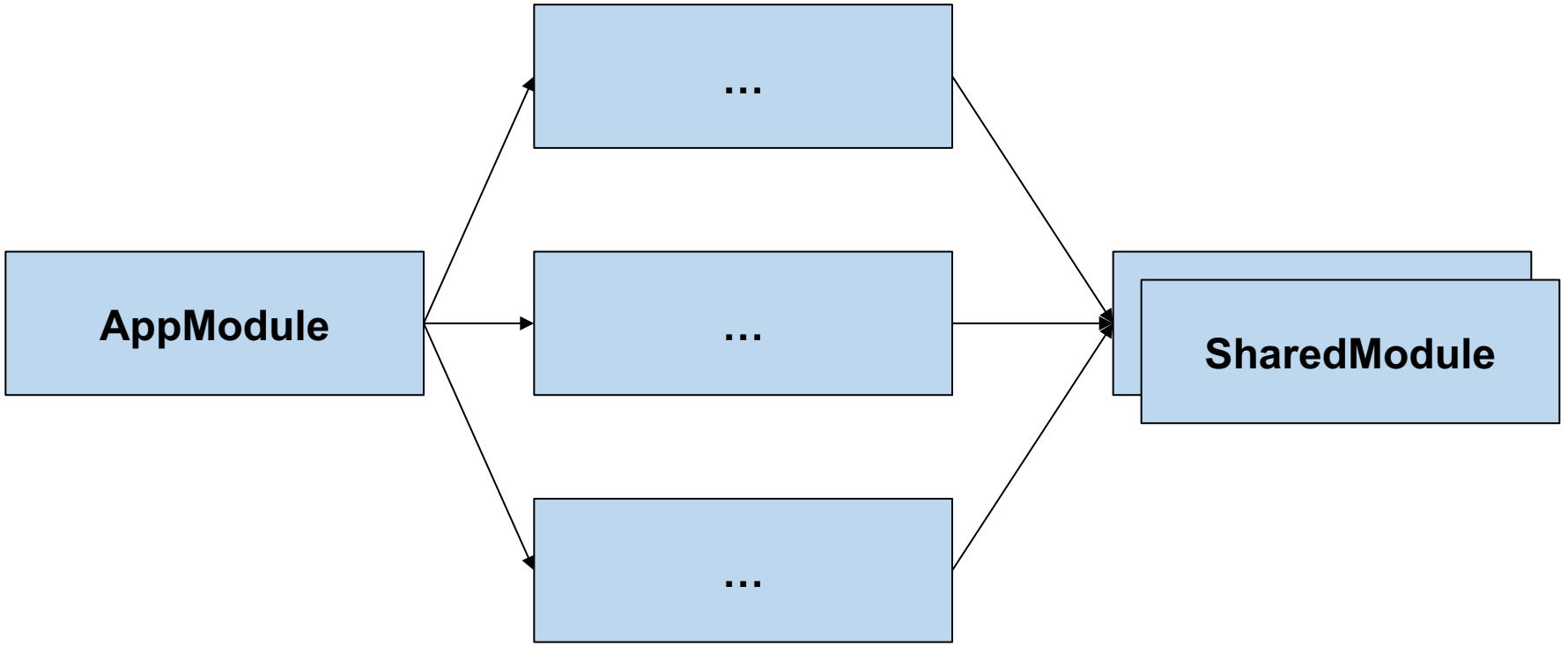
ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Architecture for Angular Enterprise Applications

Alex Thalhammer

Typical Module Structure



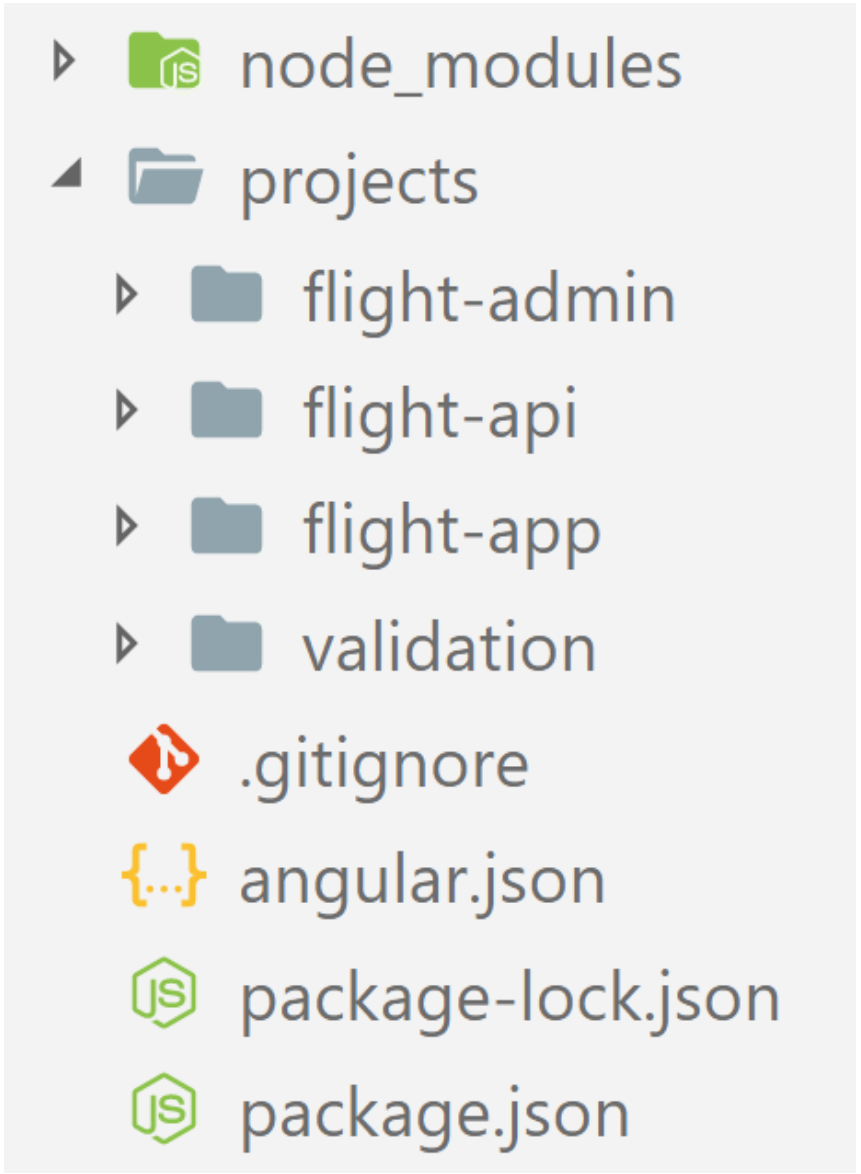
Root Module **Feature Modules** **Shared Modules**

Outline

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


Monorepos

Monorepo Structure



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

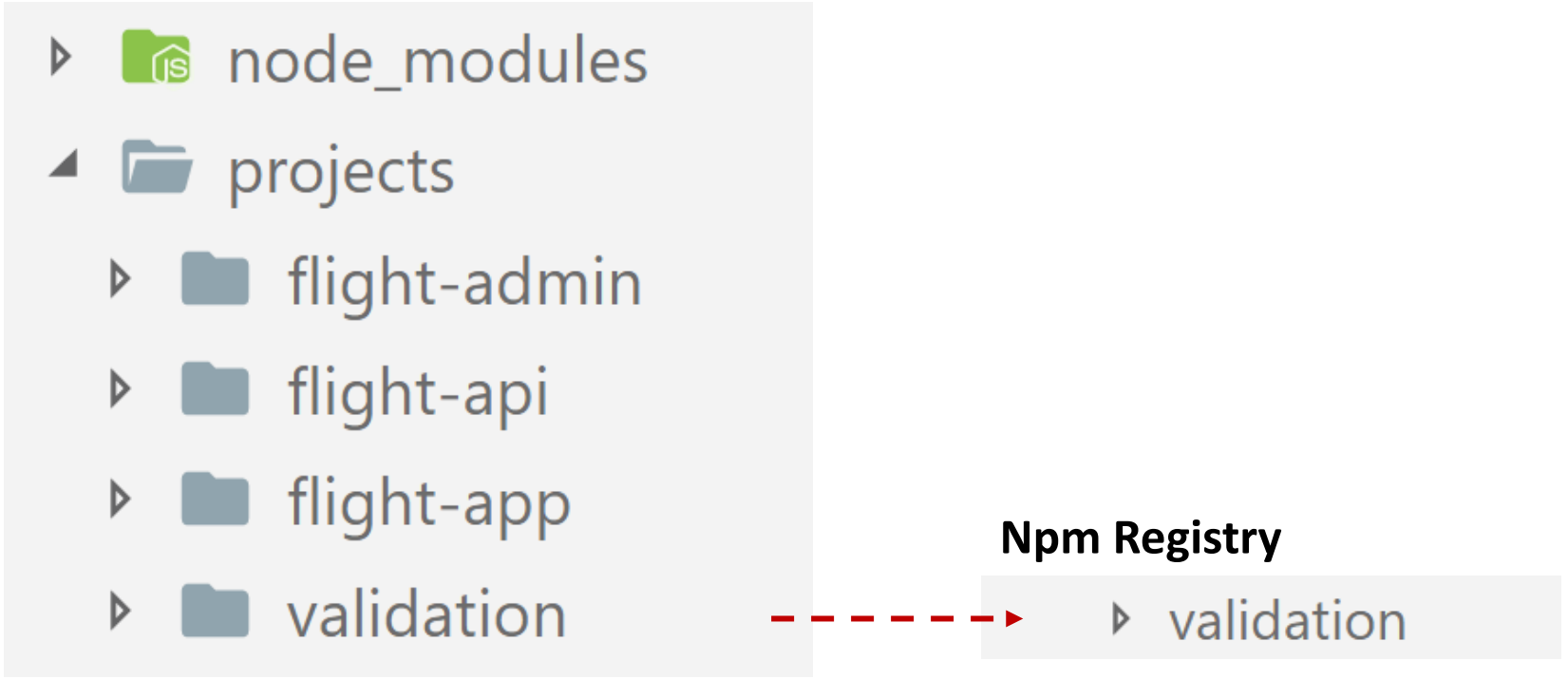
Project Monorepo

- Like Workspaces/Solutions in different IDEs

Company-wide Monorepo

- E. g. used at Google or Facebook

Moving back and forth



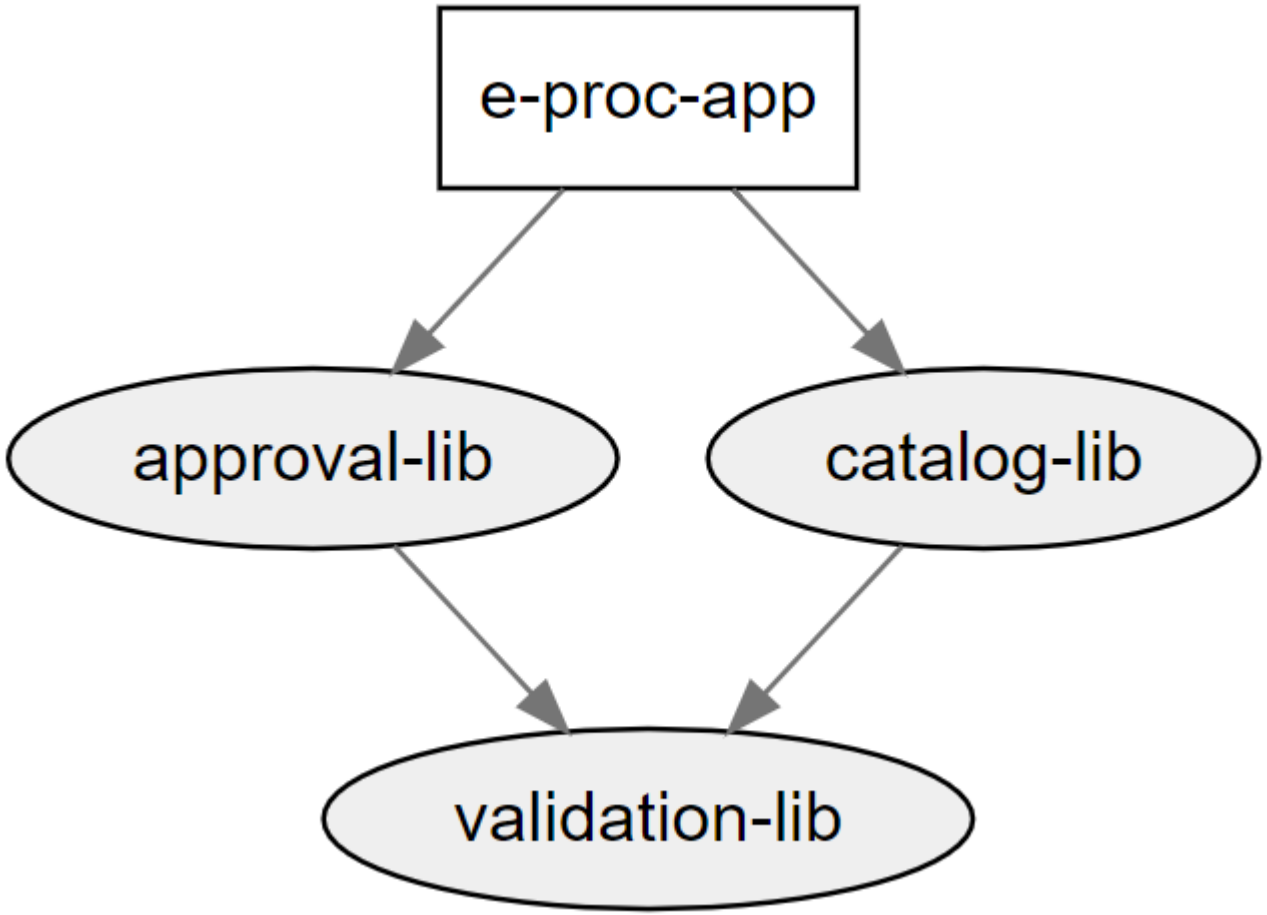
Tooling & Generator

https://nrwl.io/nx

Nrwl Extensions for Angular

An open source toolkit for enterprise Angular applications.

Visualize Module Structure



Features of Nx

- Visualize module structure and dependencies
- Addtional schematics / generators (nx g app/lib/… also for NgRx)
- Access restrictions: which apps/libs can access which libs (via ESLint)
- Just recompile changed apps/libs (works w/o cloud cache!)
- Support for
  - ESLint
  - NgRx
  - Jest, Cypress & Storybook

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

nx generate app my-app
nx generate lib my-lib --buildable

nx serve --project my-app
nx build --project my-app
```

DEMO

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

Lab

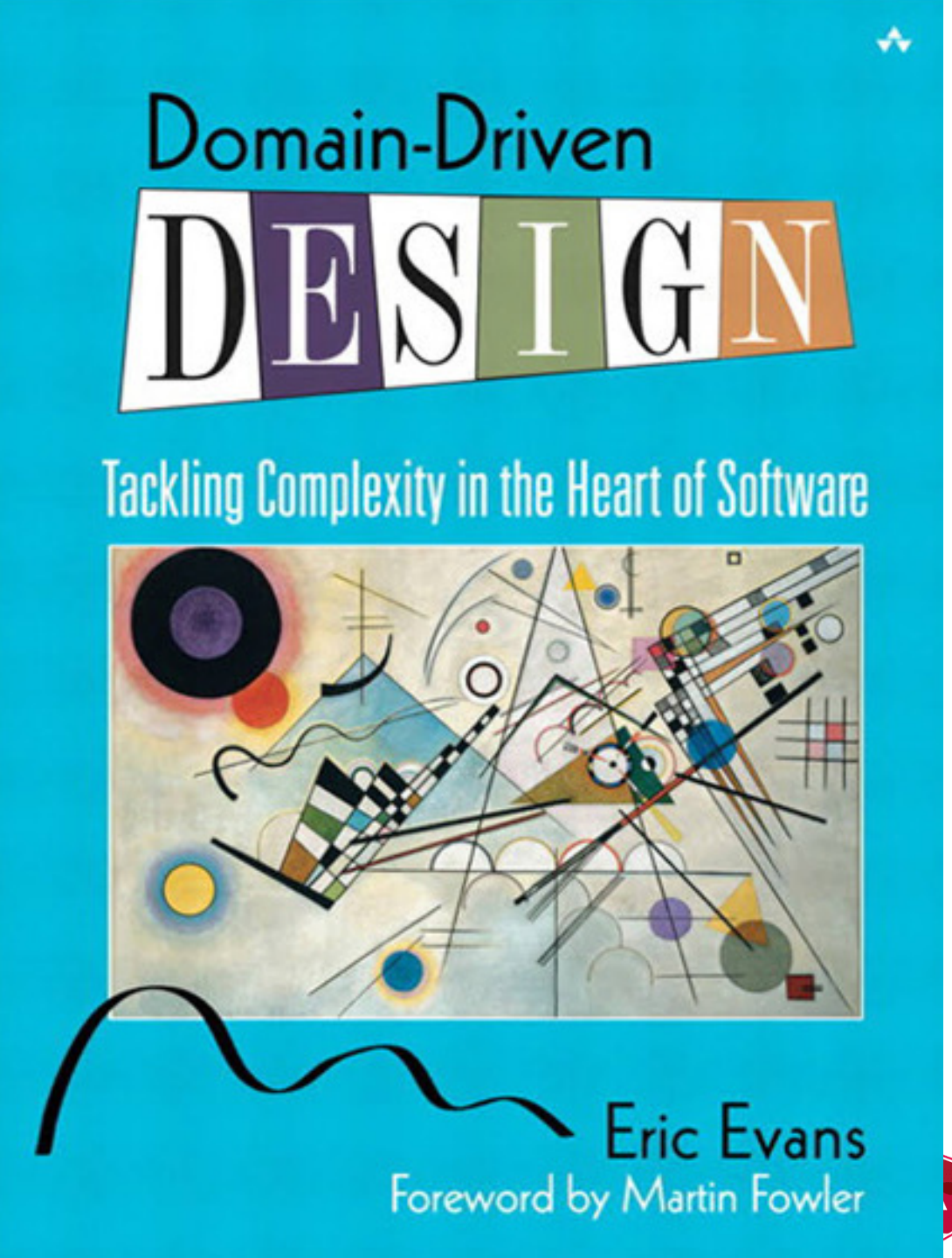
Libraries and Monorepo

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

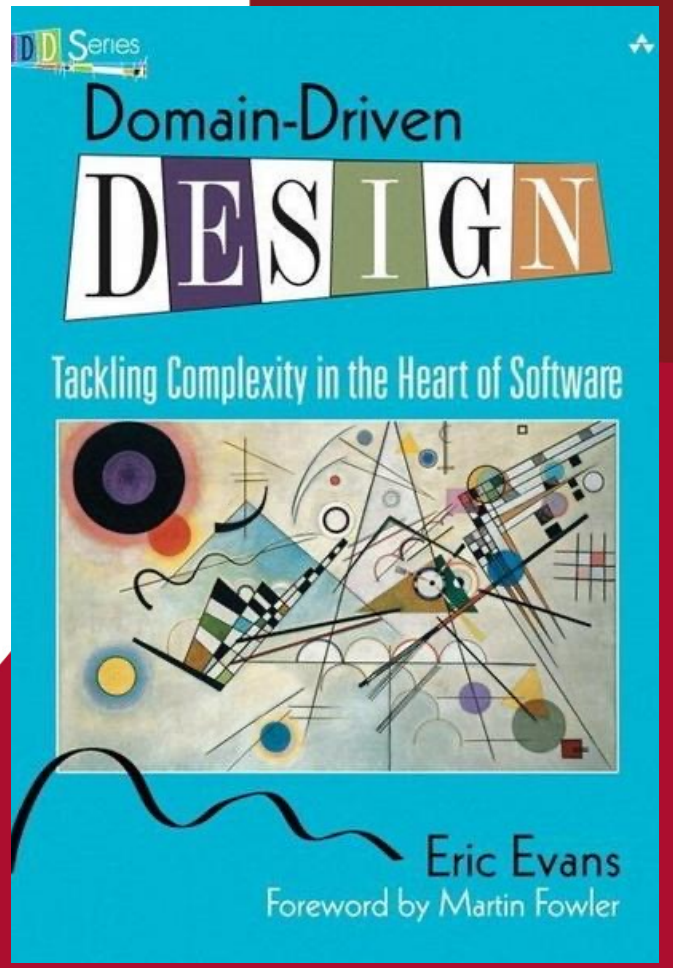
DDD

in a nutshell



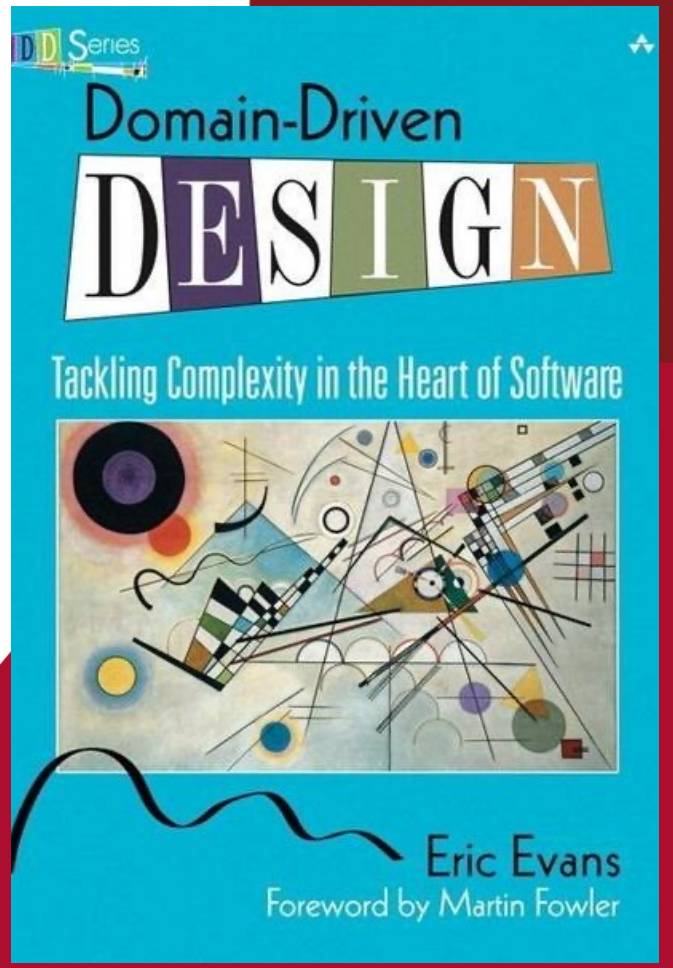
Methodology for bridging the gap b/w requirements and architecture/ design

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

Design Patterns
& Practices

Strategic Design

Tactical Design

Domain Driven Design

Decomposing a System

Design Patterns
& Practices

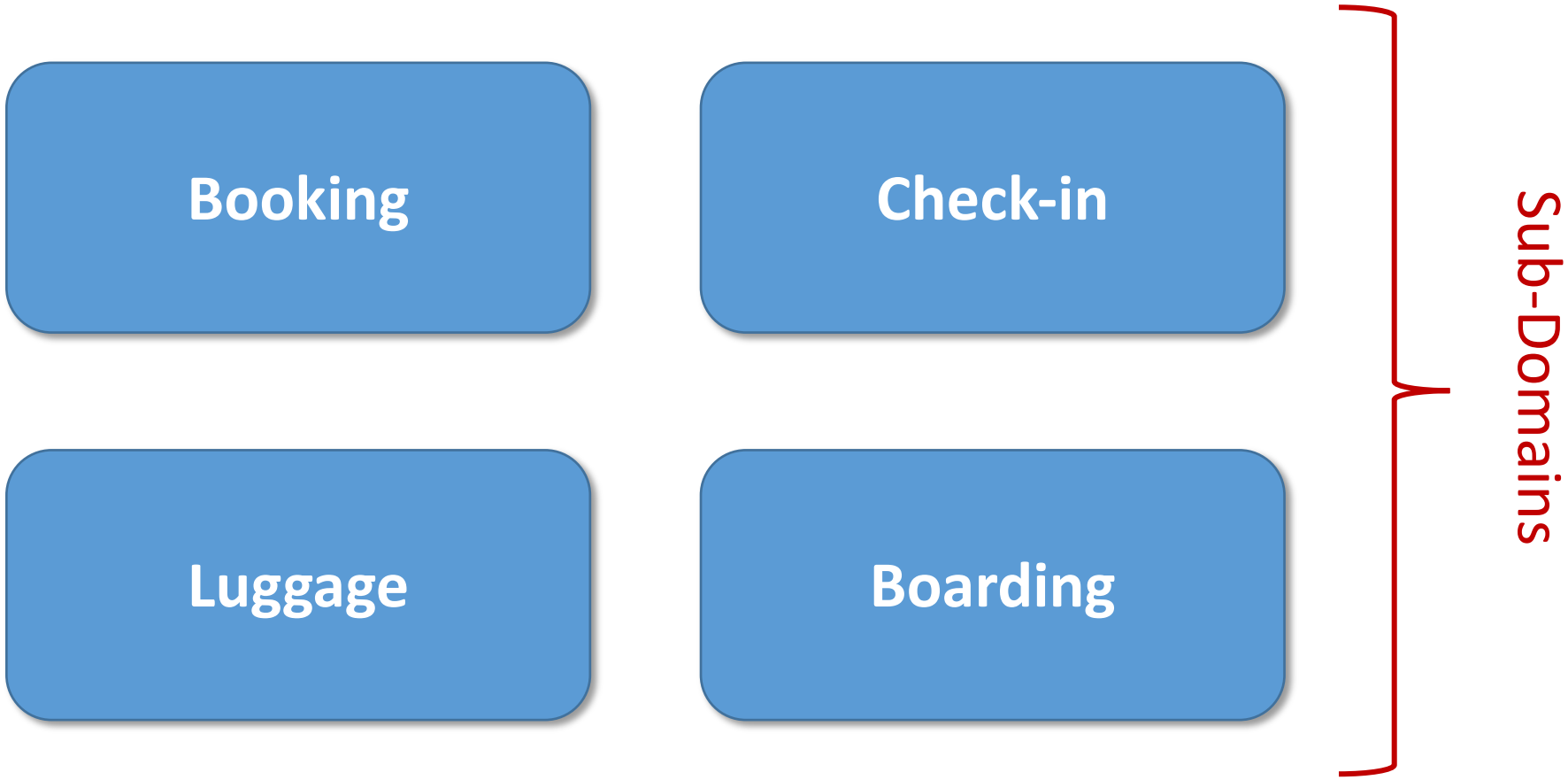
Strategic Design

Tactical Design

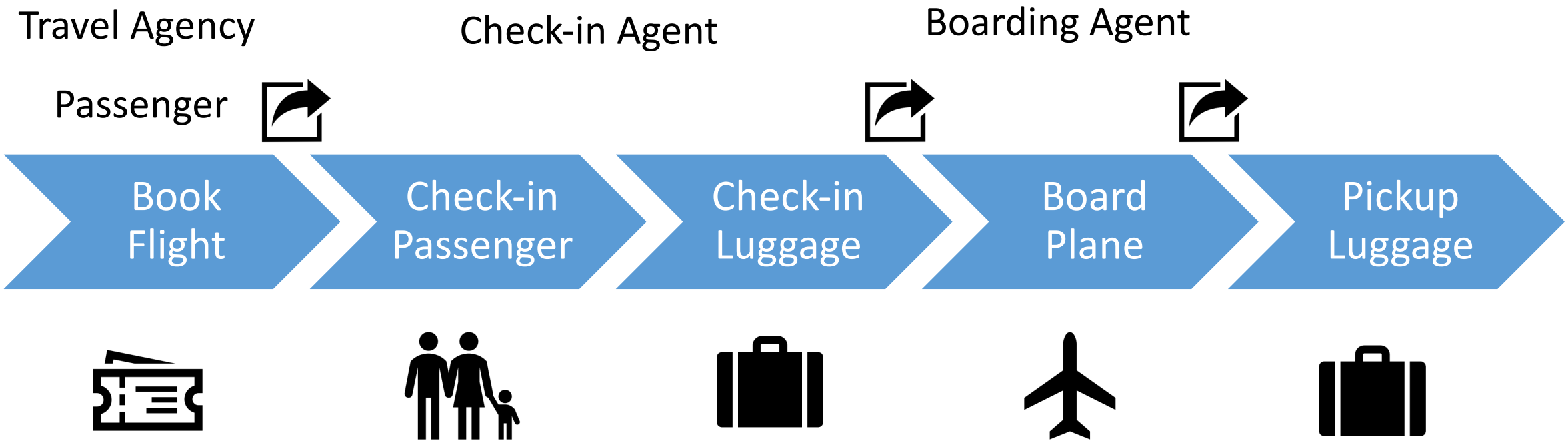

This is what Strategic DDD prevents

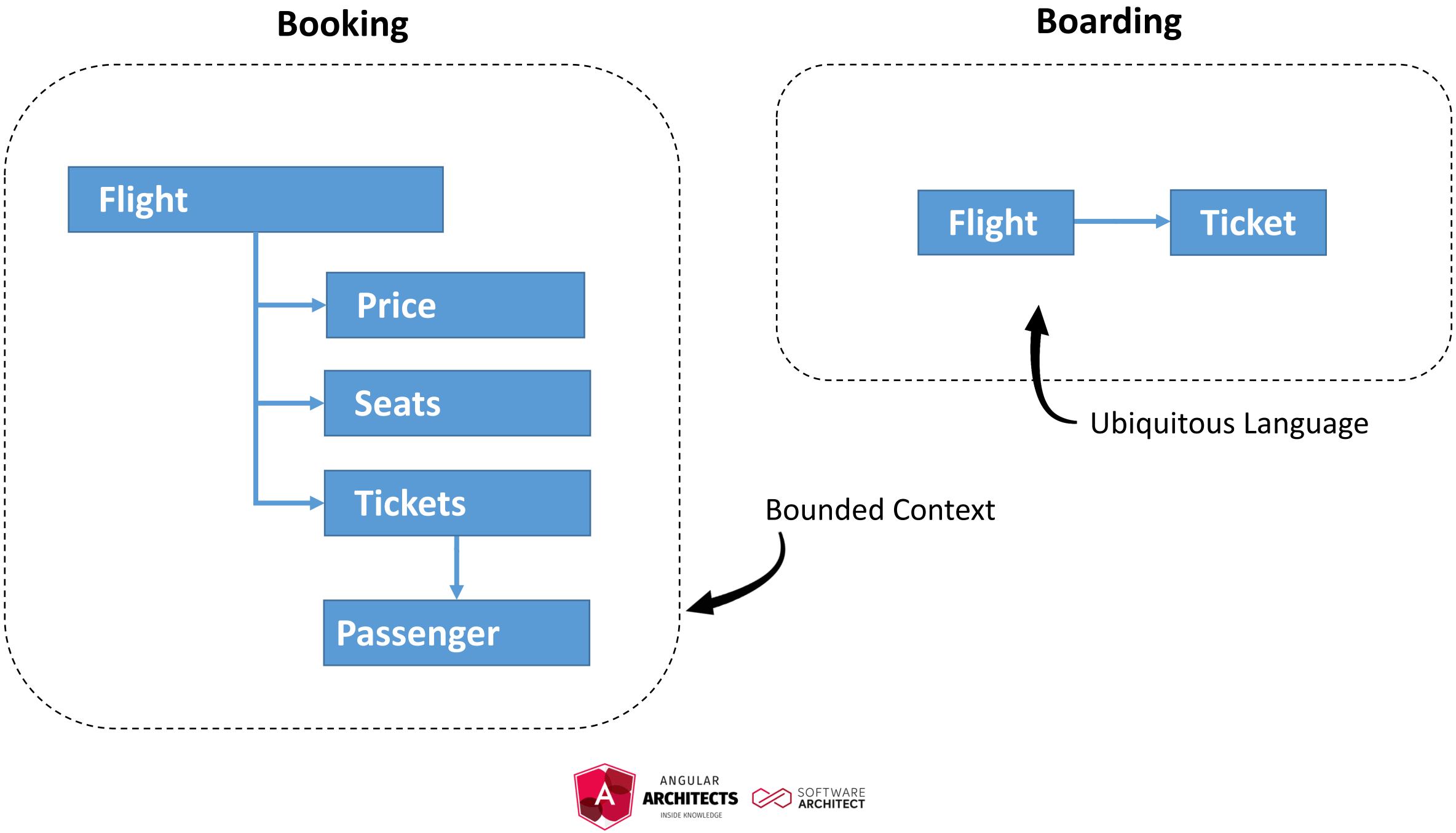
Example

Example

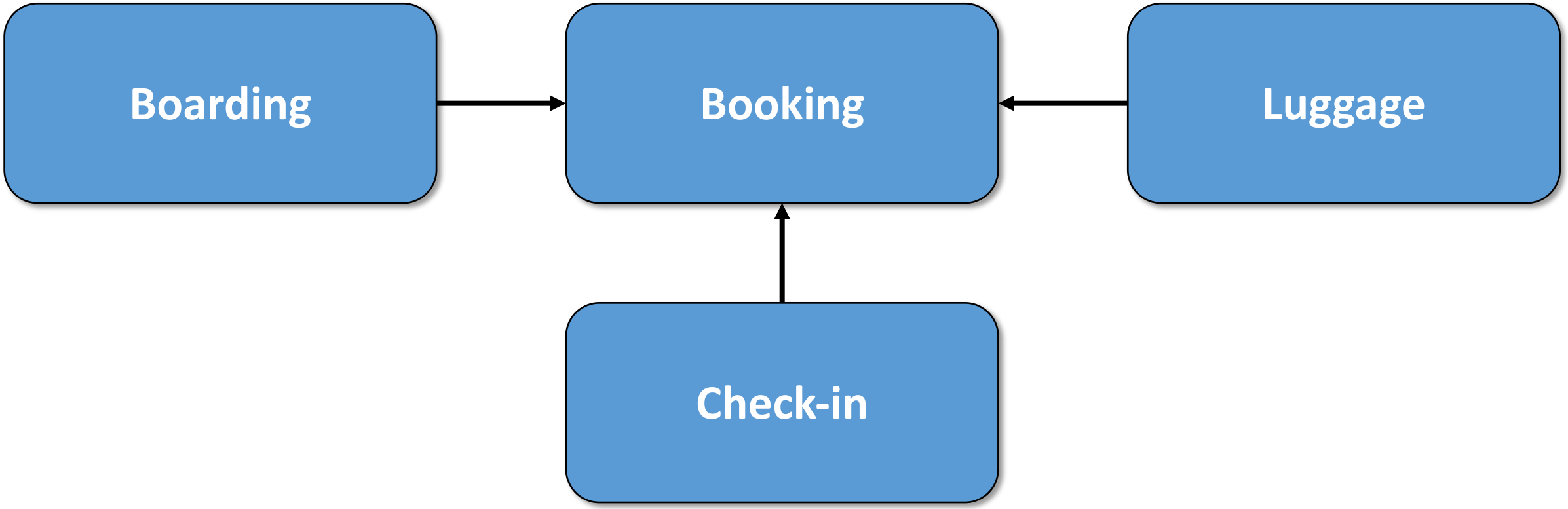


Finding Sub-Domains




Booking
Flight
Price
Seats
Tickets
Passenger
Boarding
Flight
Ticket
Ubiquitous Language
Bounded Context

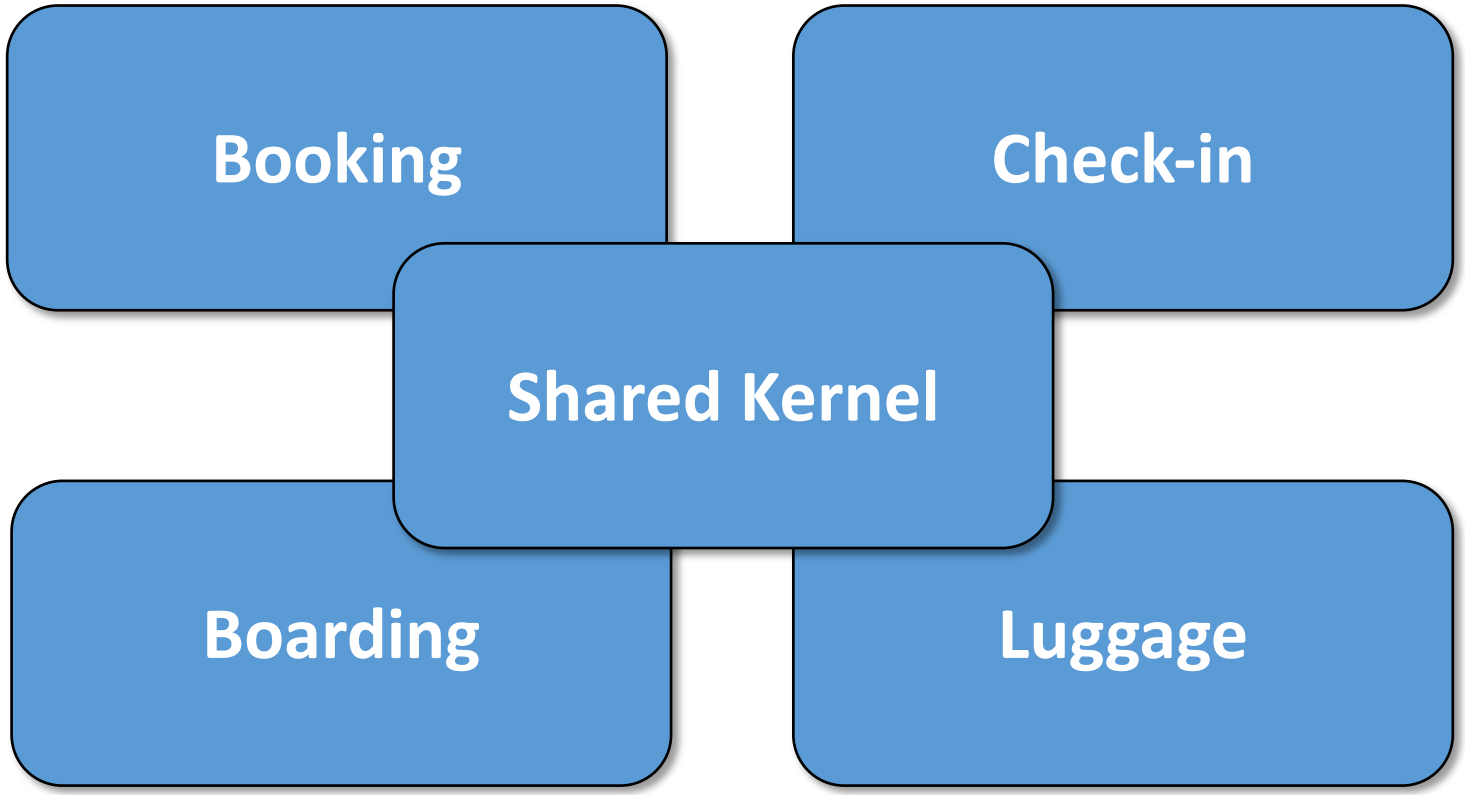
Context Map

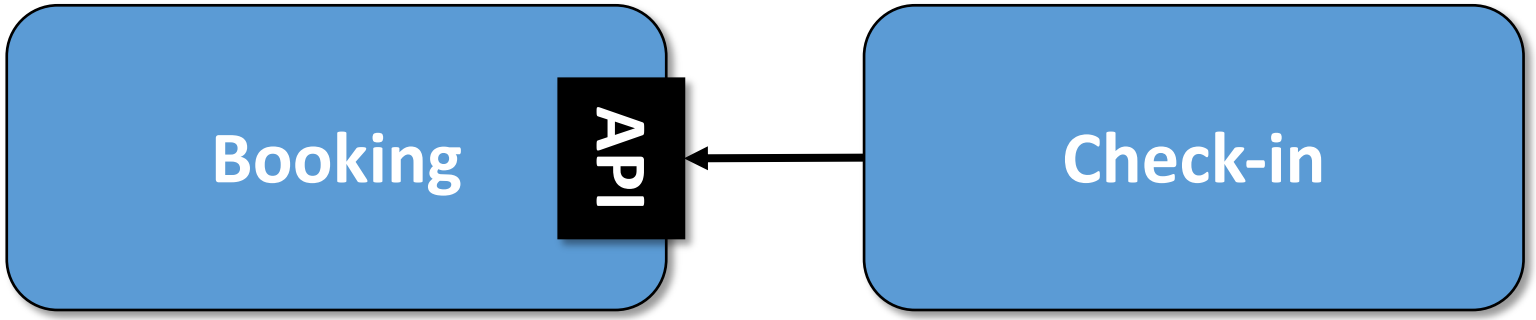


Context Map

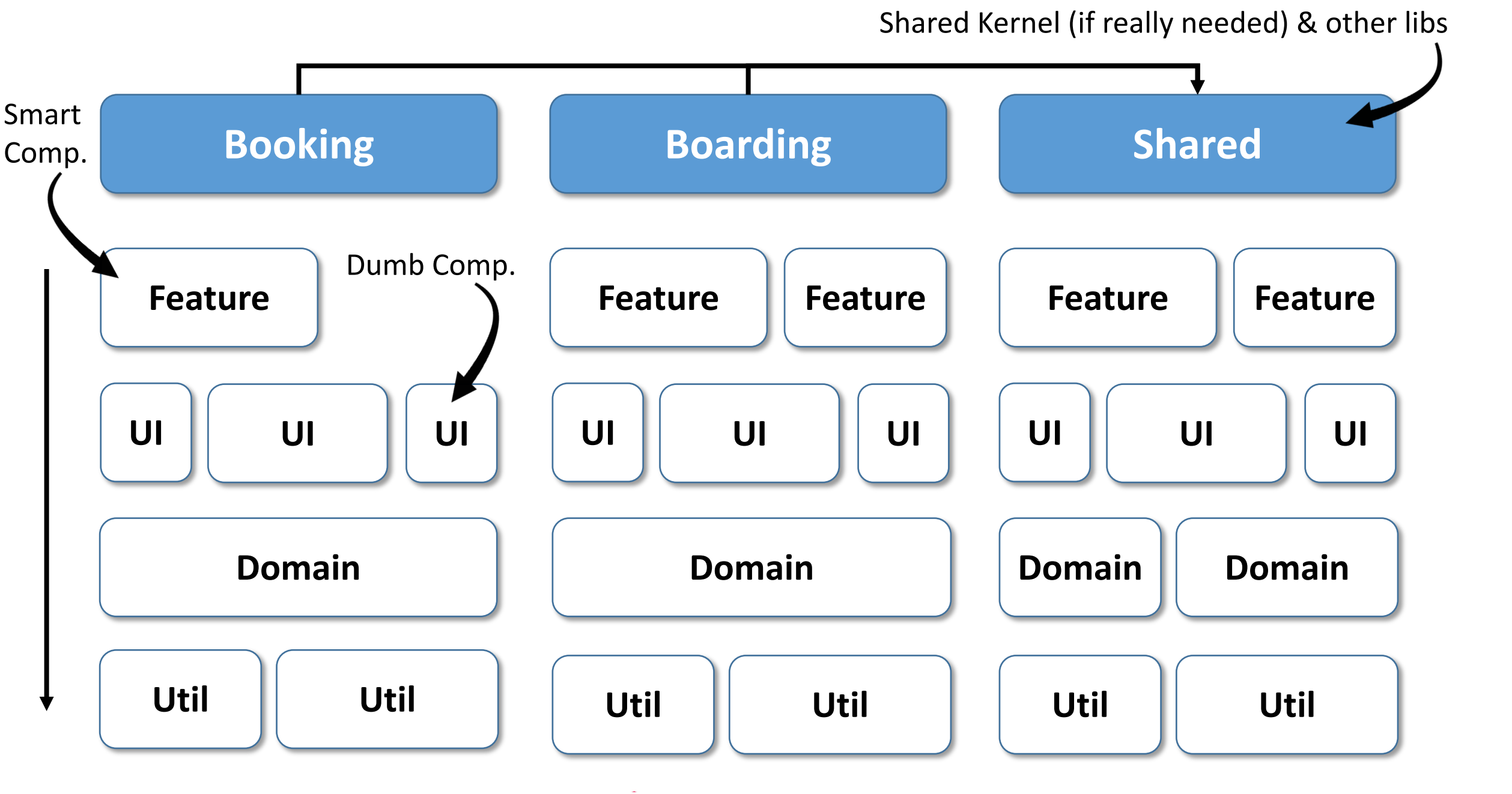
Responsibilities?

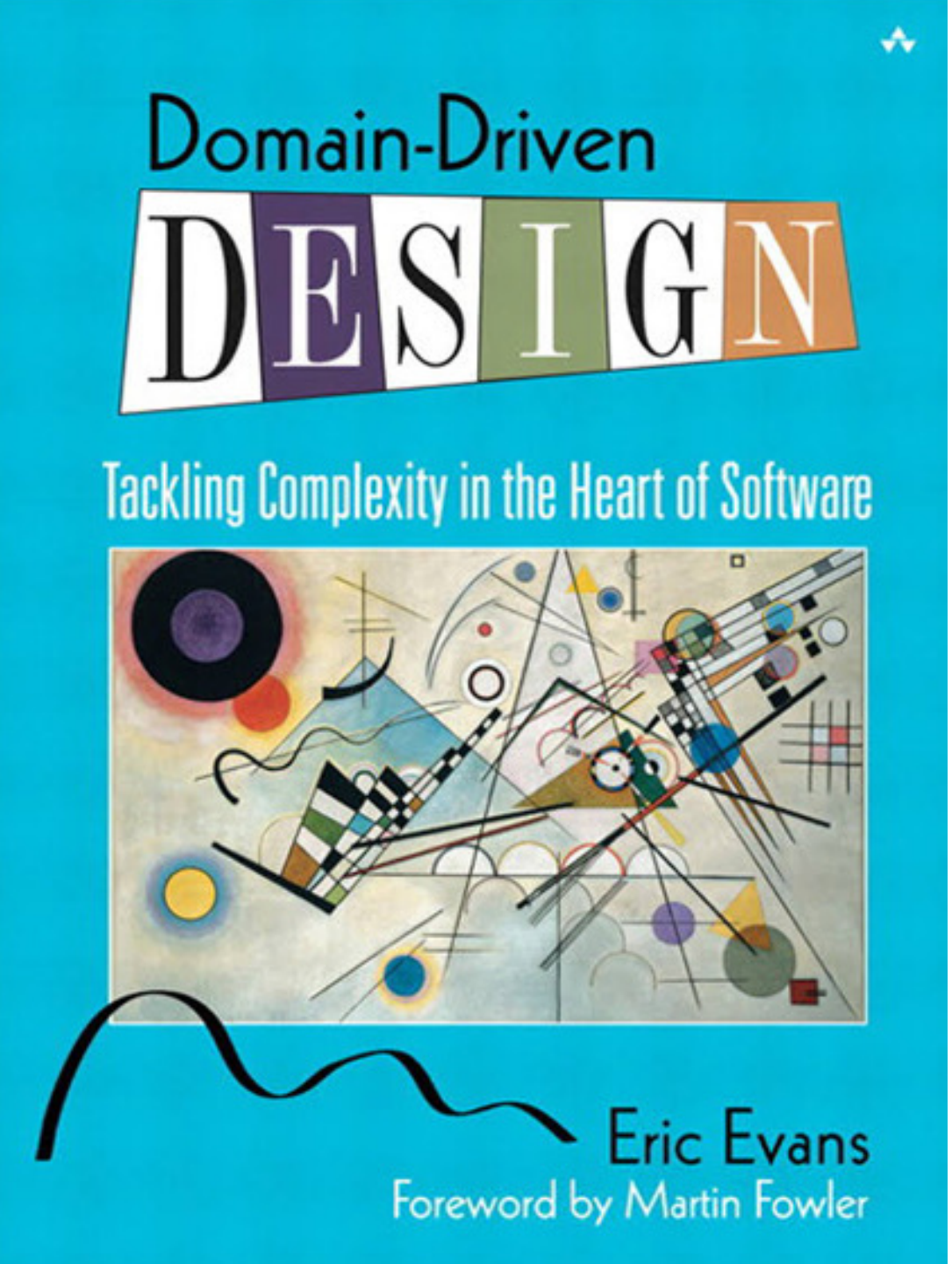
Breaking Changes?



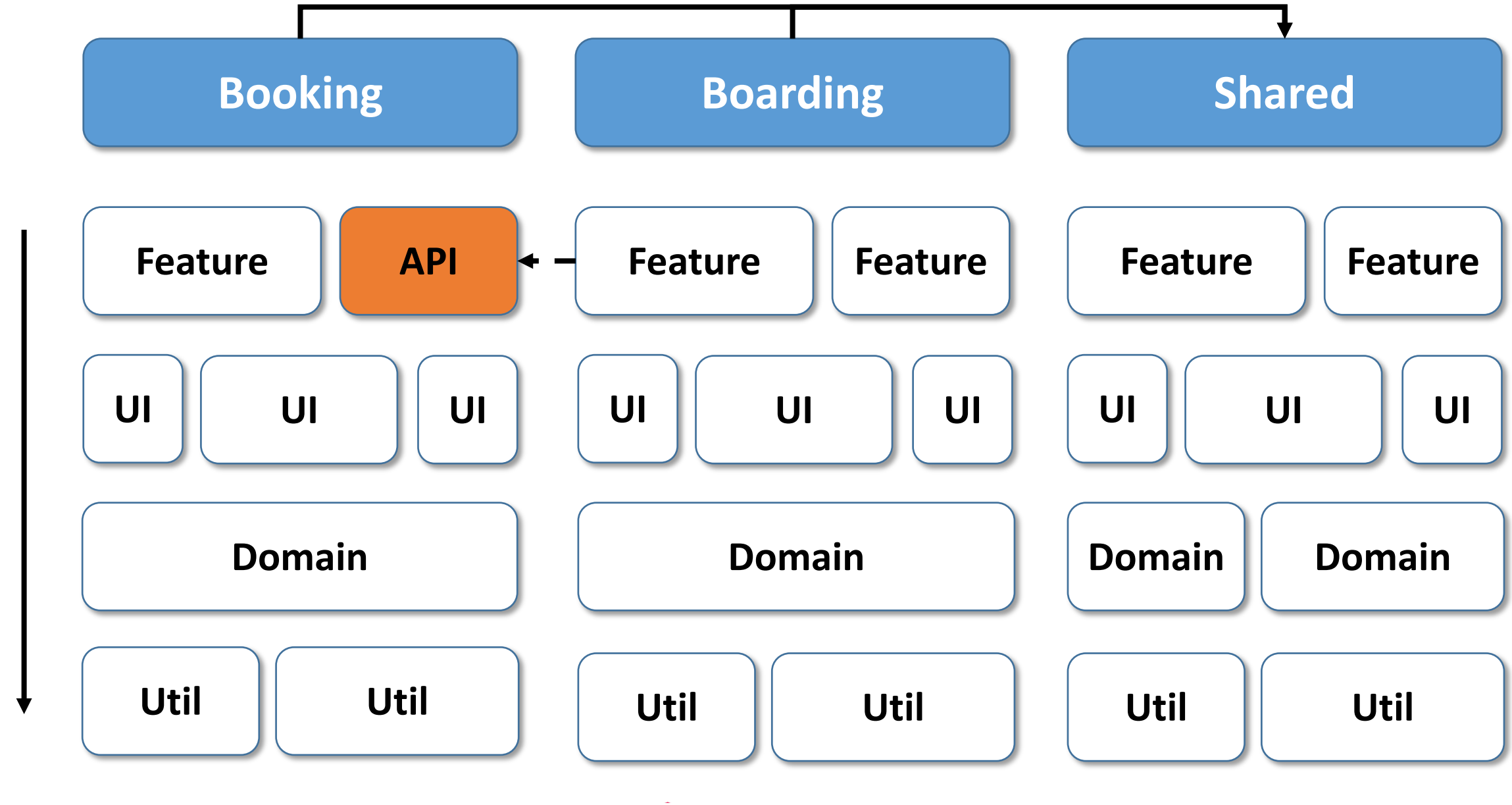


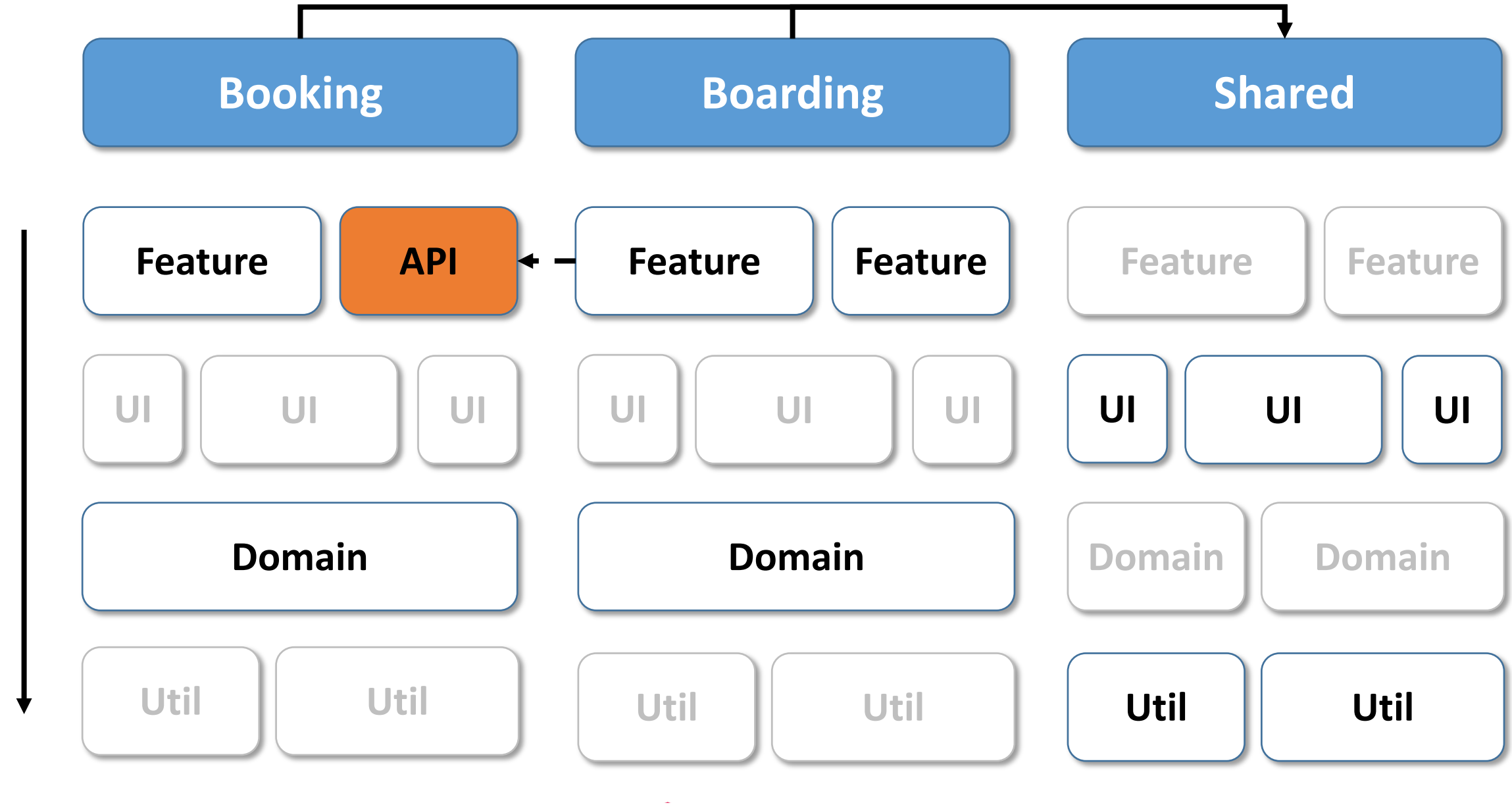
Open-/Host-Service


Shared Kernel (if really needed) & other libs
Smart Comp.
Booking
Boarding
Shared
Feature
Dumb Comp.
Feature
Feature
Feature
Feature
UI
UI
UI
UI
UI
UI
UI
UI
UI
Domain
Domain
Domain
Domain
Util
Util
Util
Util
Util
Util

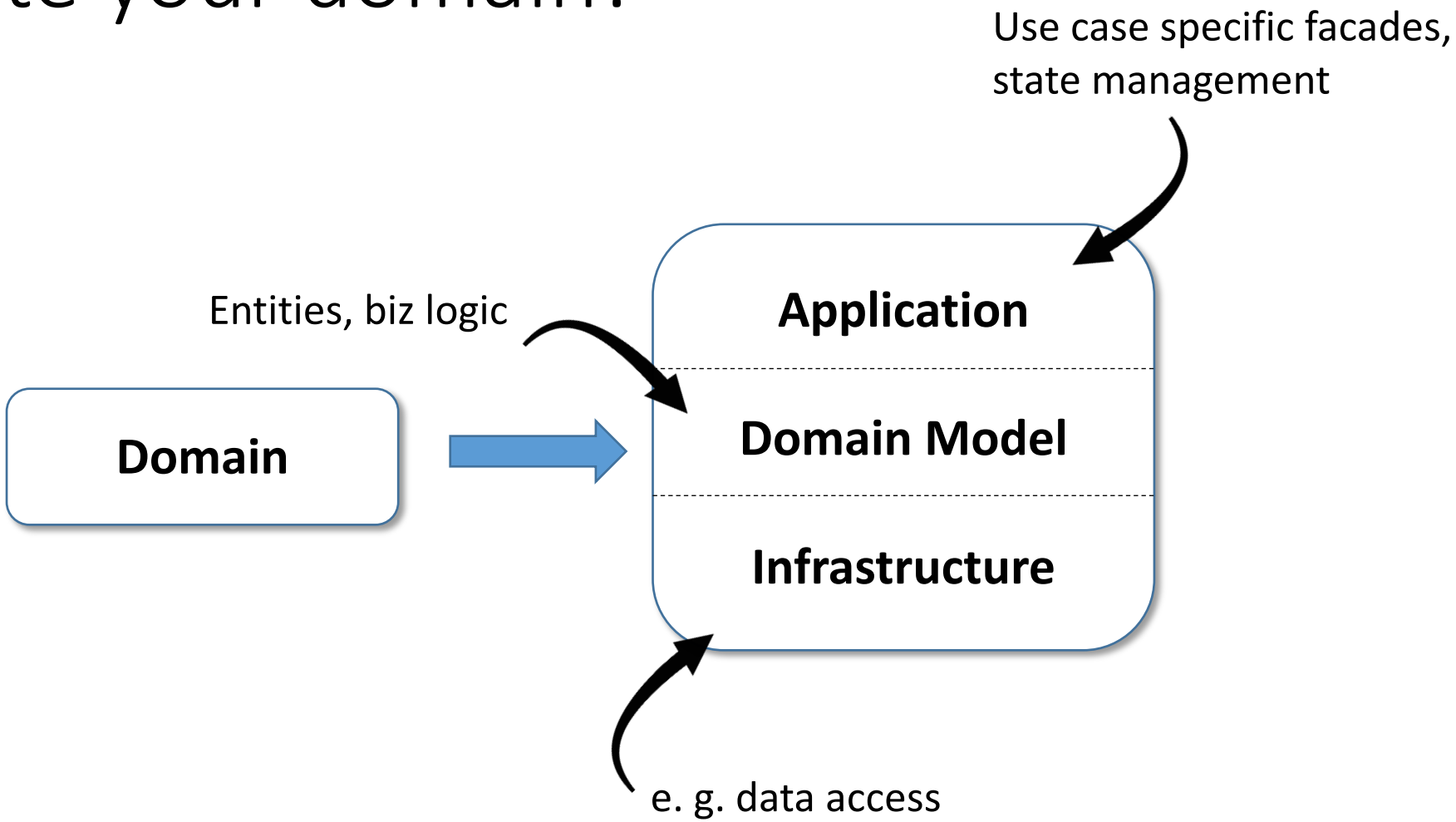


Lots of approaches for cross-domain communication and more …


Booking
Boarding
Shared
Feature
API
Feature
Feature
Feature
Feature
UI
UI
UI
UI
UI
UI
UI
UI
UI
Domain
Domain
Domain
Domain
Util
Util
Util
Util
Util
Util


Booking
Boarding
Shared
Feature
API
Feature
Feature
Feature
Feature
UI
UI
UI
UI
UI
UI
UI
UI
UI
Domain
Domain
Domain
Domain
Util
Util
Util
Util
Util
Util

Isolate your domain!



Alternatives to layering

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

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

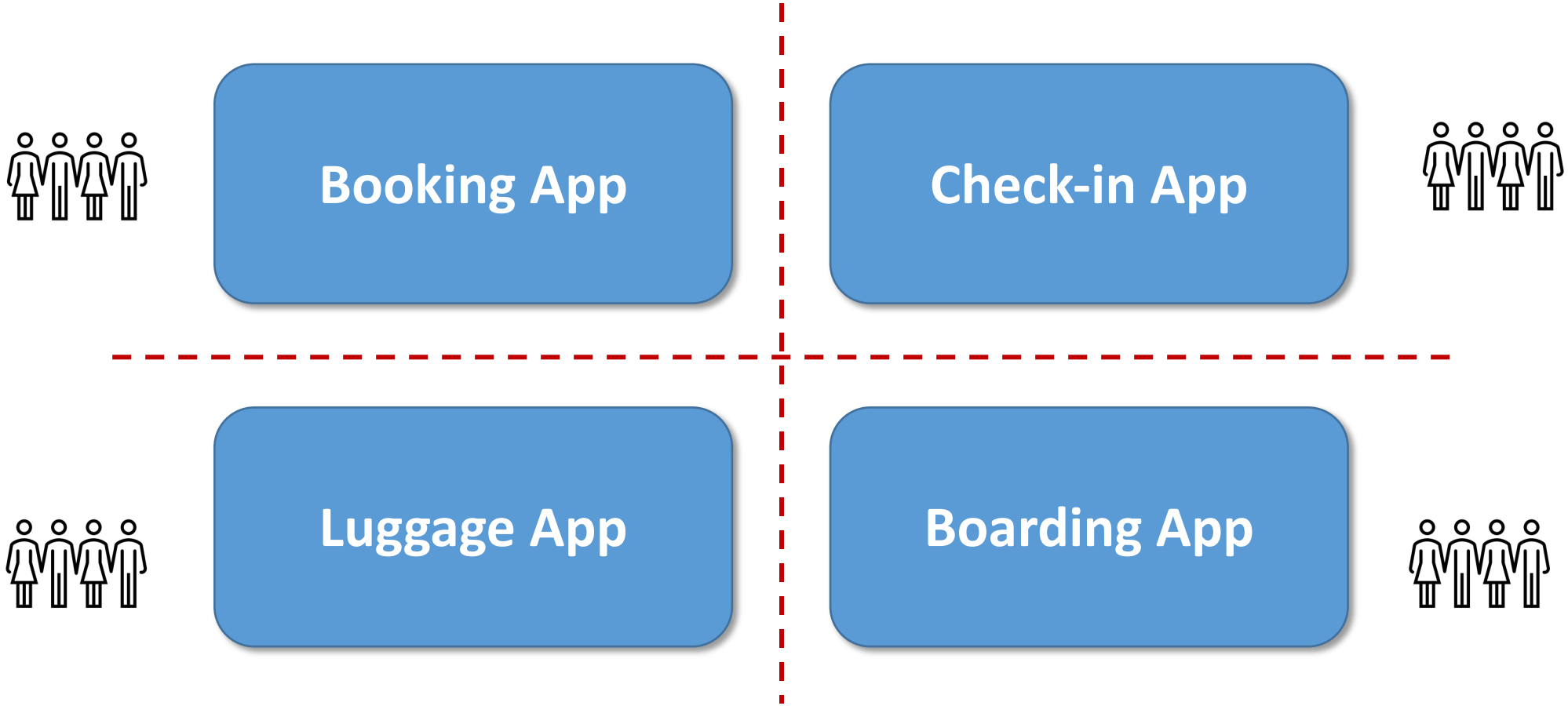
ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE ARCHITECT

Micro Frontends?

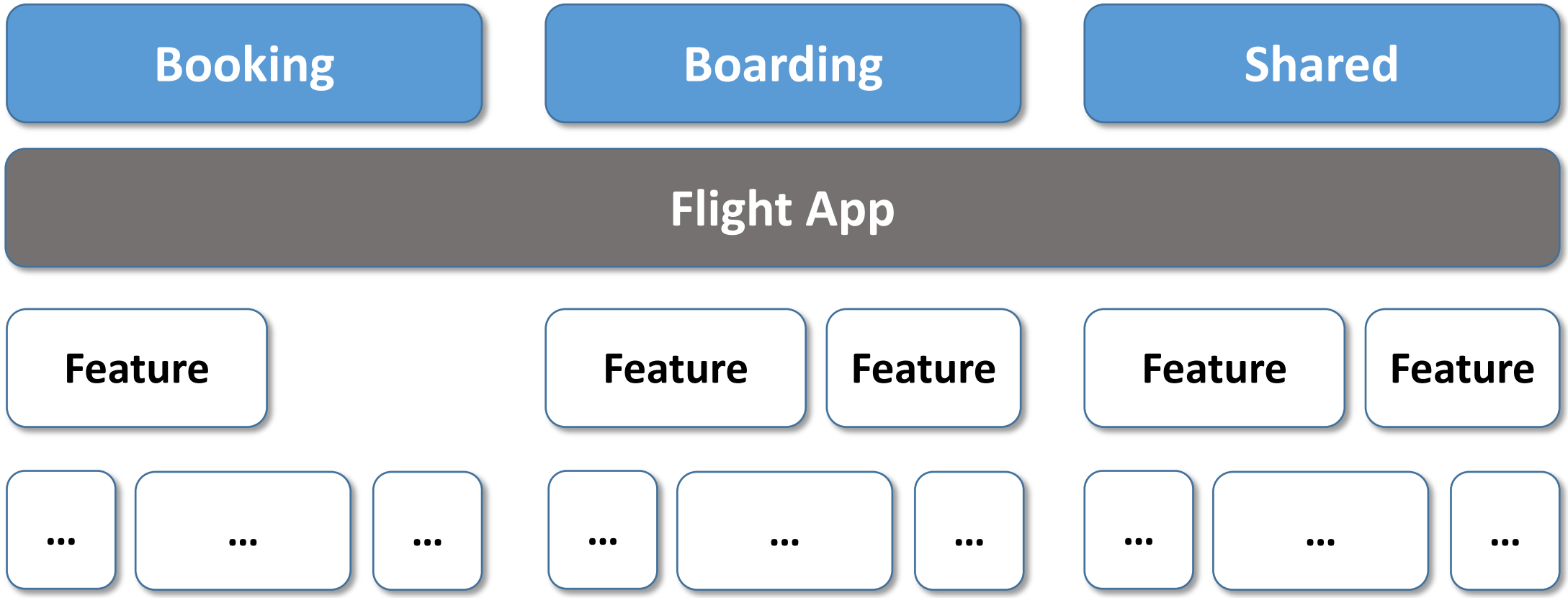
Short outlook

Microfrontends

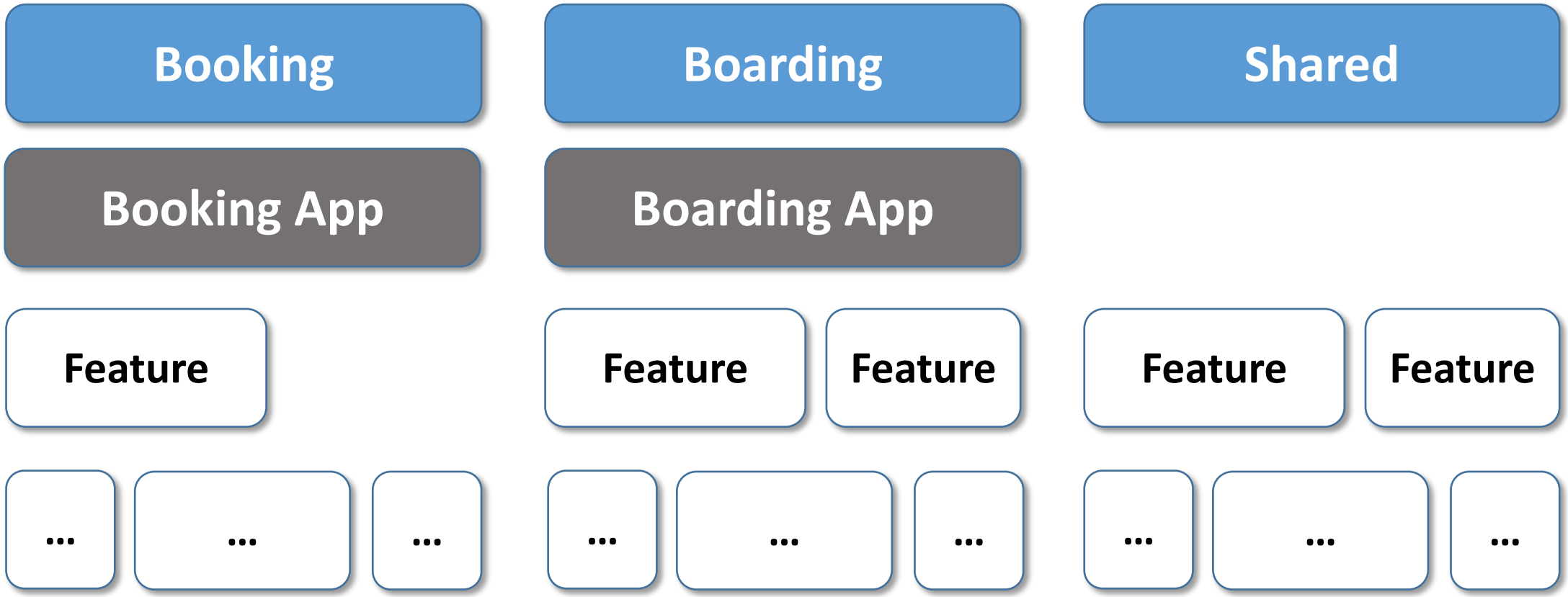


Microfrontends
are first and foremost
about **scaling teams!**

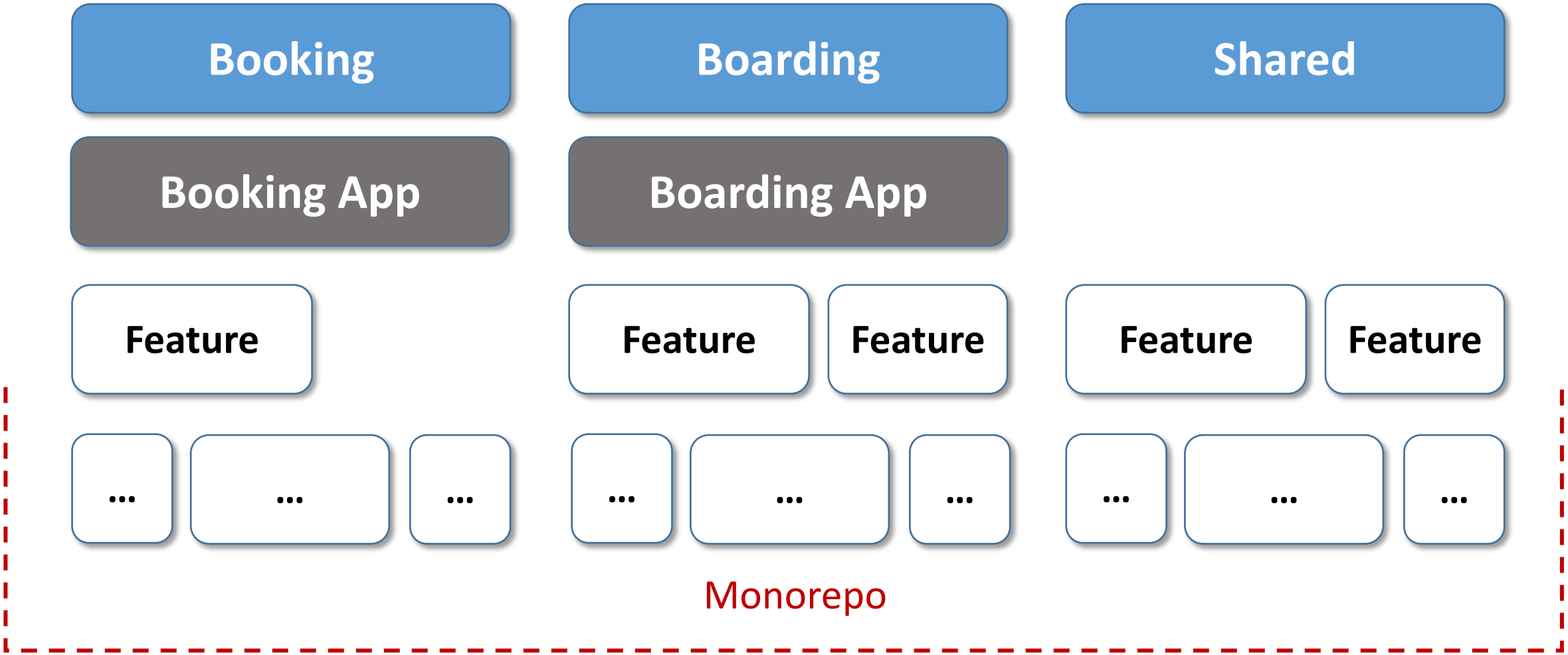
Deployment Monolith



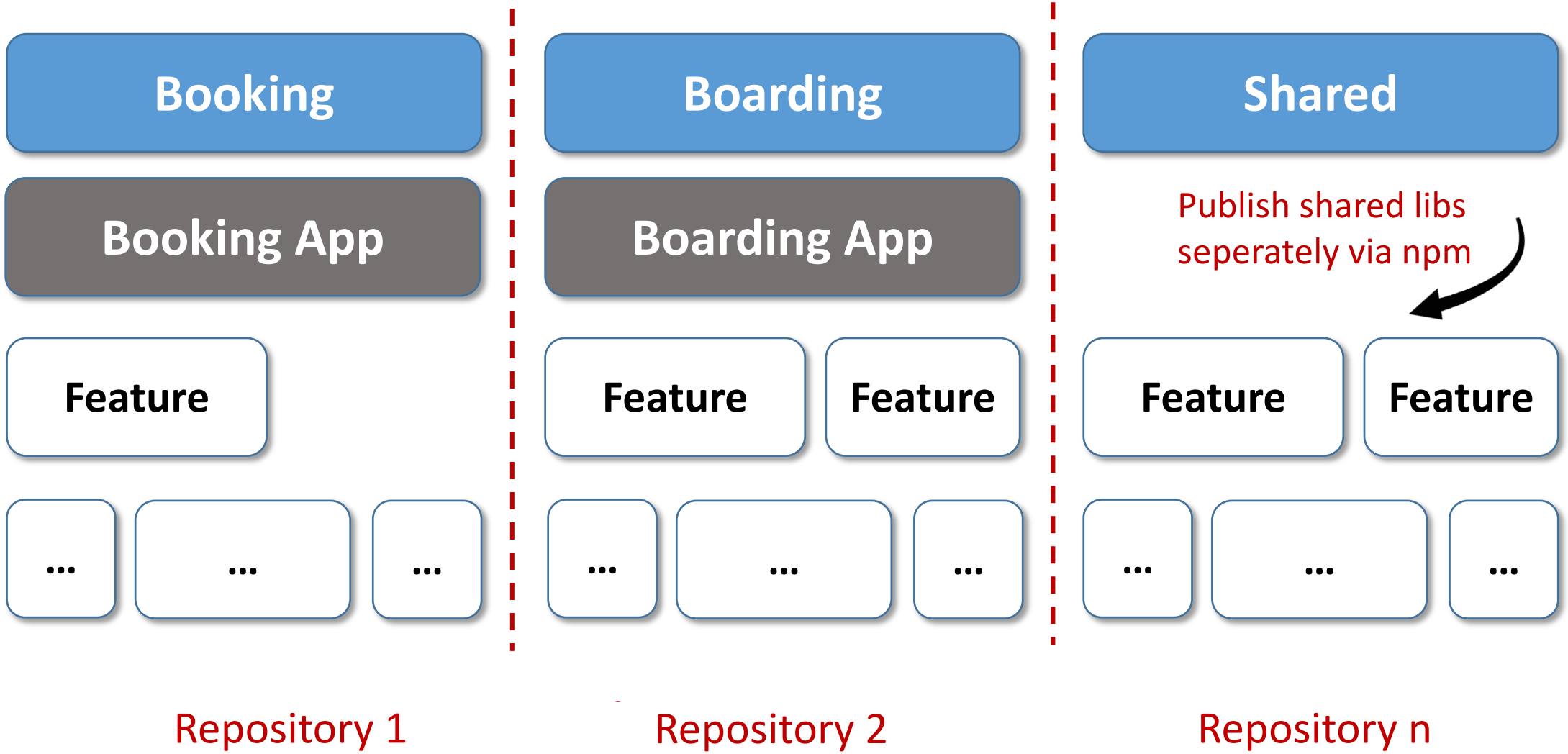
Microfrontends



Option 1: One App per Domain



Option 2: One Monorepo per Domain

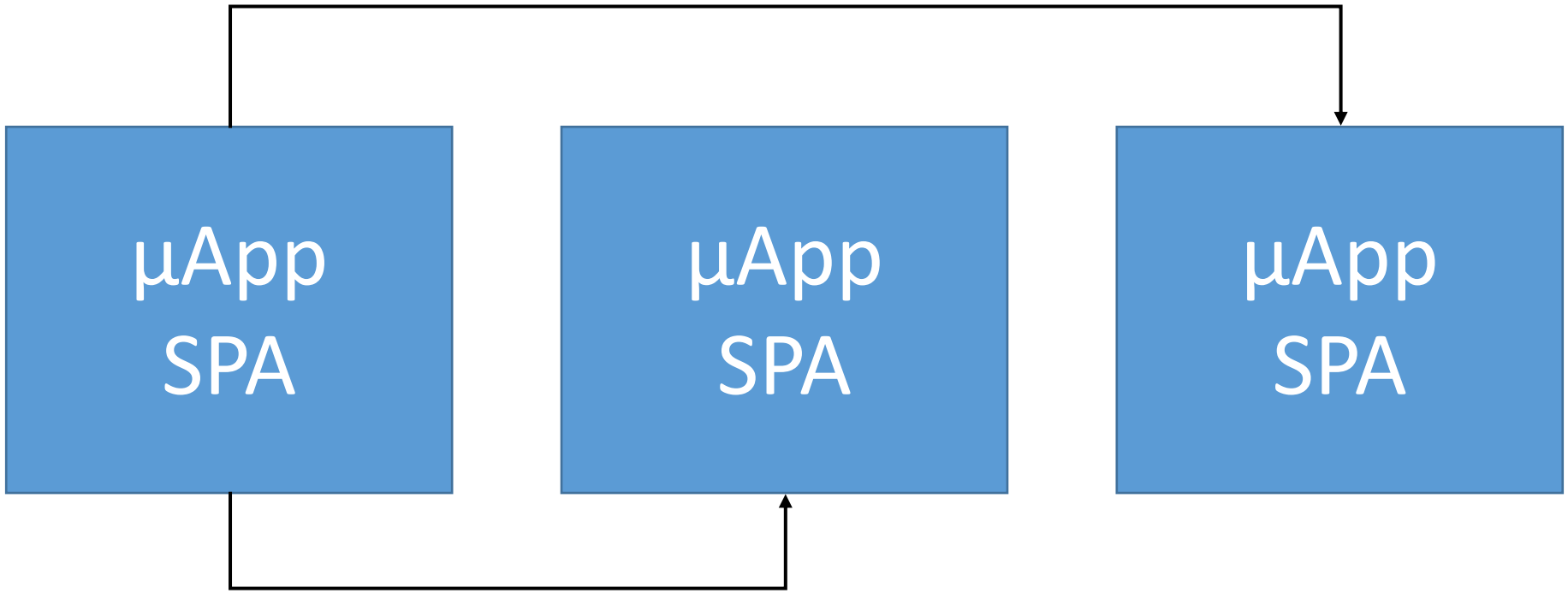


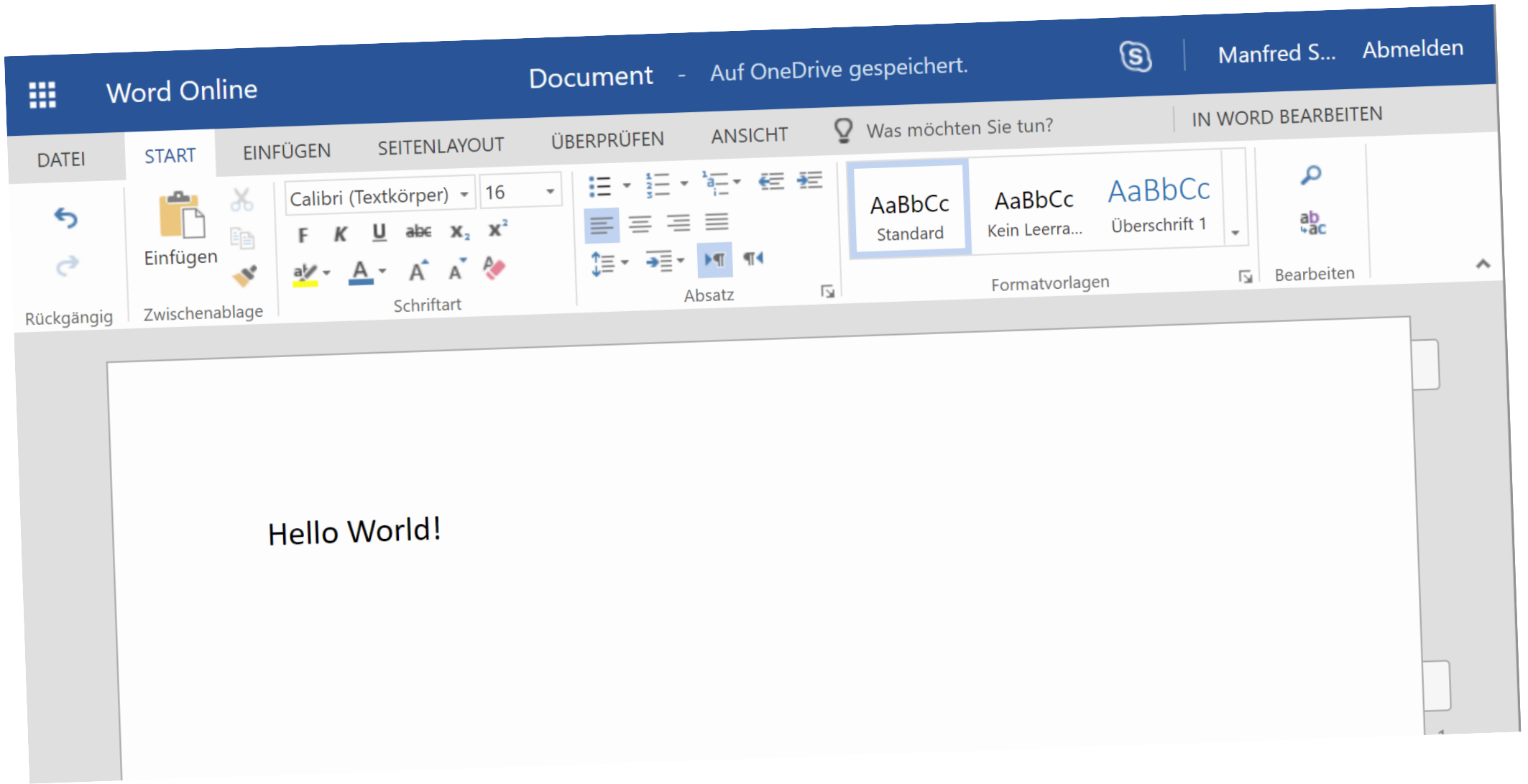
Benefits

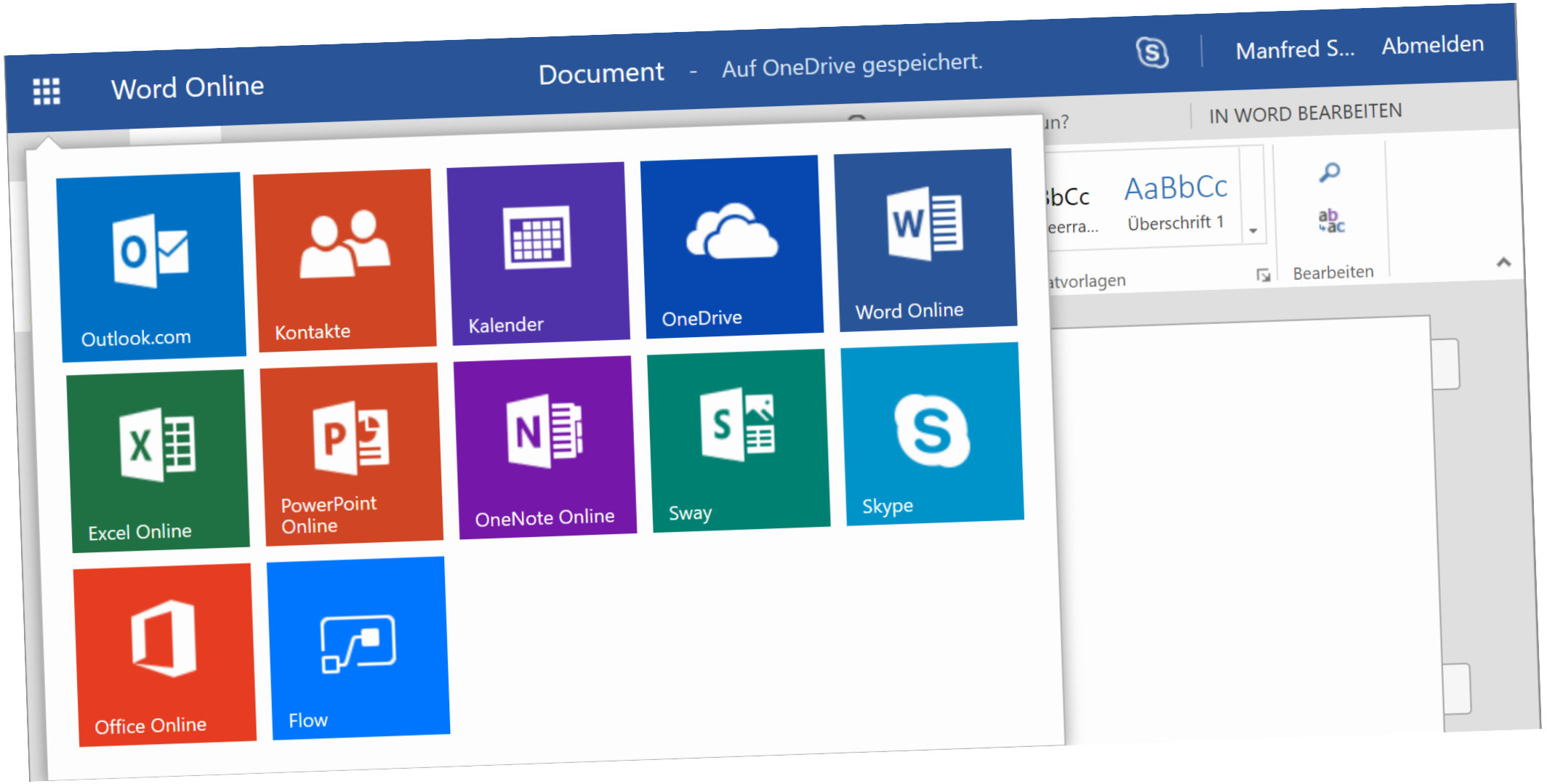
- **Autonomous Teams**
- Separate Development
- Separate Deployment
- Own architecture decisions
- Own technology descisions

Integration via Hyperlinks

UI Composition w/ Hyperlinks








Integration via
Shell

Providing a (SPA based) Shell (aka Host)

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

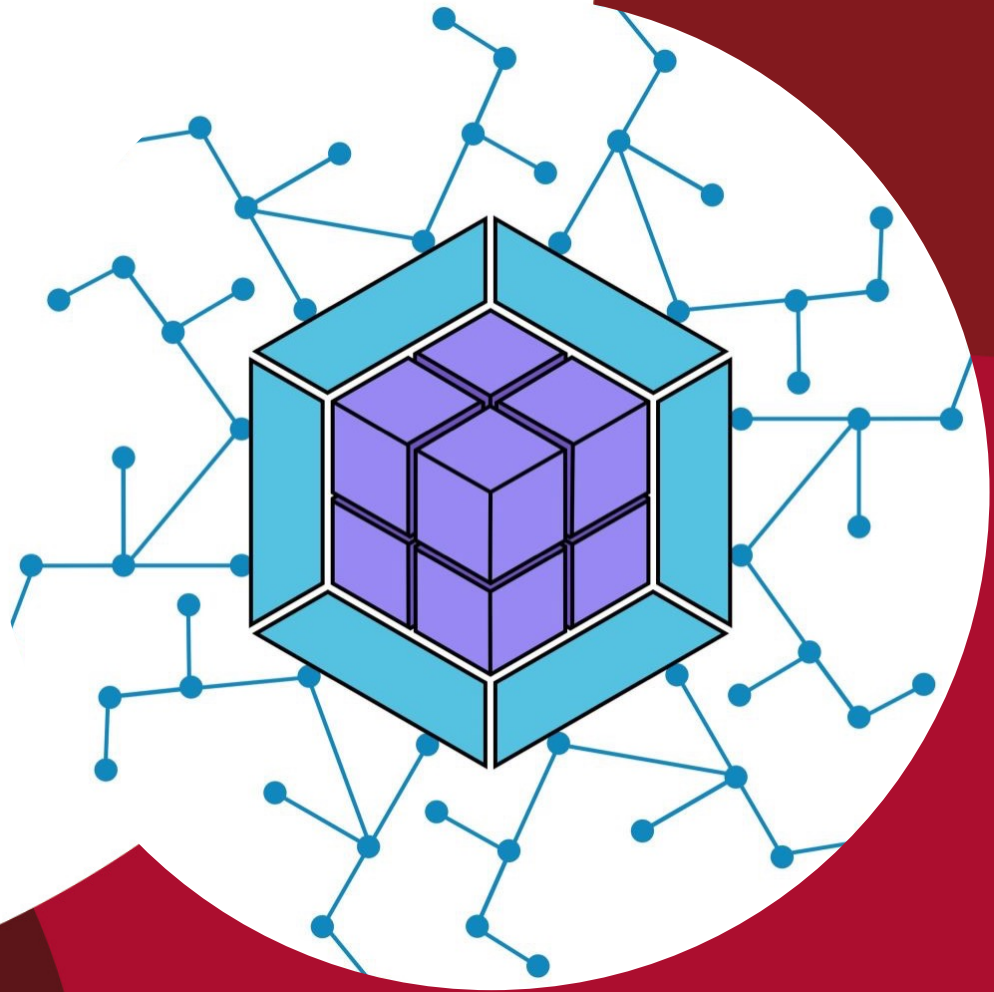
Loading MicroApps

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

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

Loading Separately Compiled SPAs

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

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

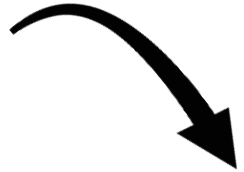
Webpack 5
Module Federation



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Idea

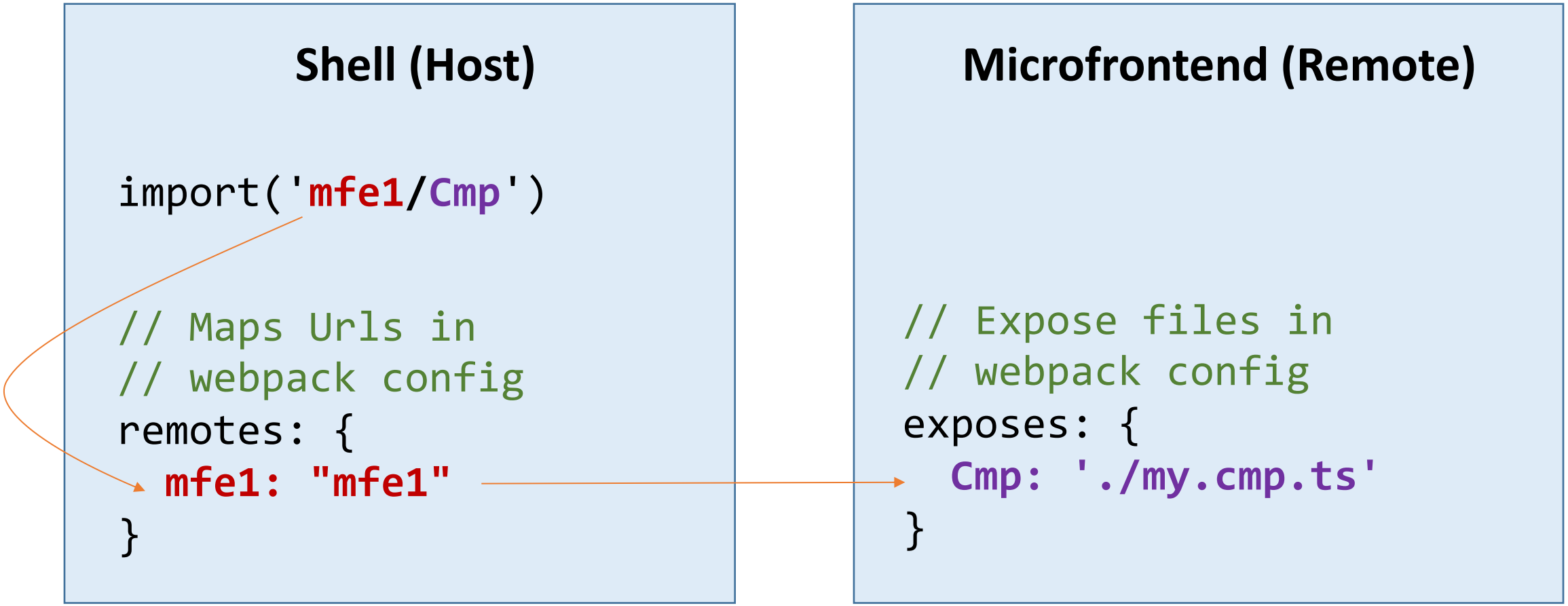
Does not work with
webpack/ Angular CLI



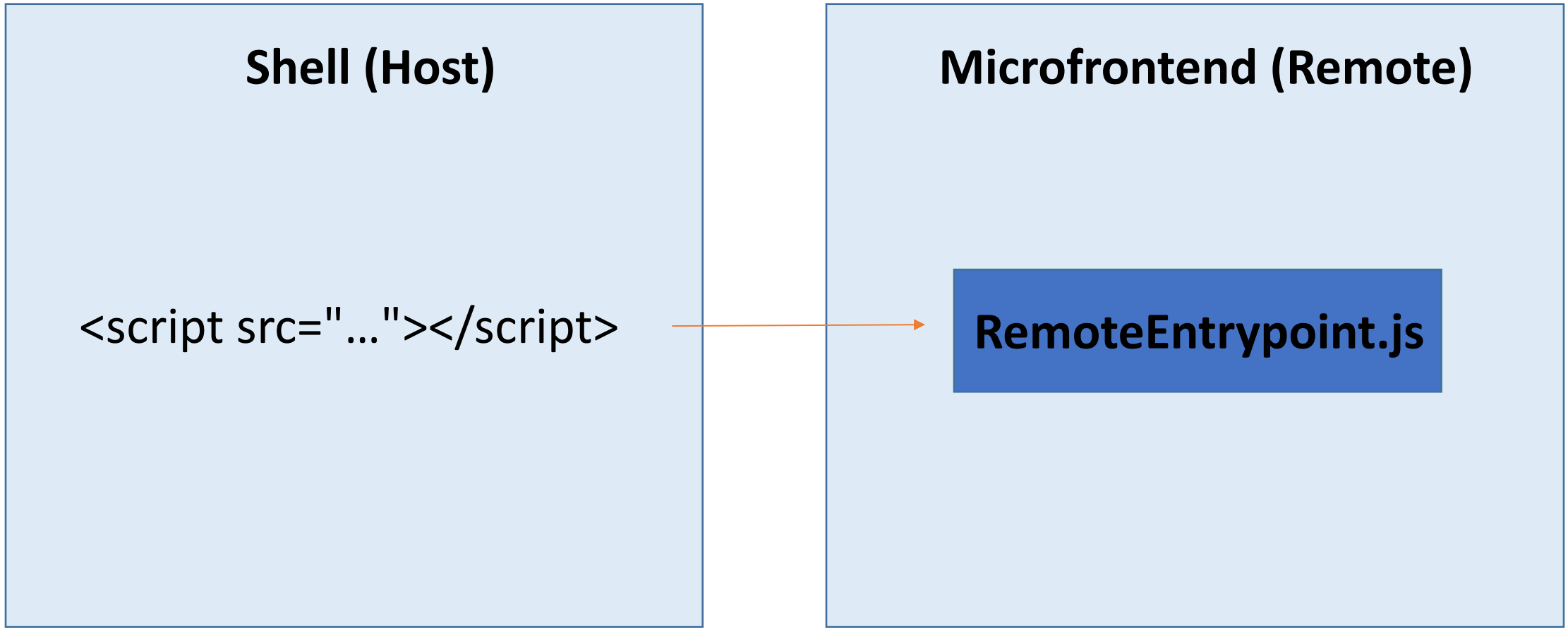
```
const Component = import('http://other-app/xyz')
```

Even lazy parts must be
known at compile time!

Webpack 5 Module Federation



How to Get the Microfrontend's URL?



How to Share Libs?

Shell (Host)

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

Microfrontend (Remote)

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

Dealing with Version Mismatches

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

Default Behavior

Selecting the highest compatible version

Default Behavior

Conflict: No highest compatible version

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"
  }
}
```

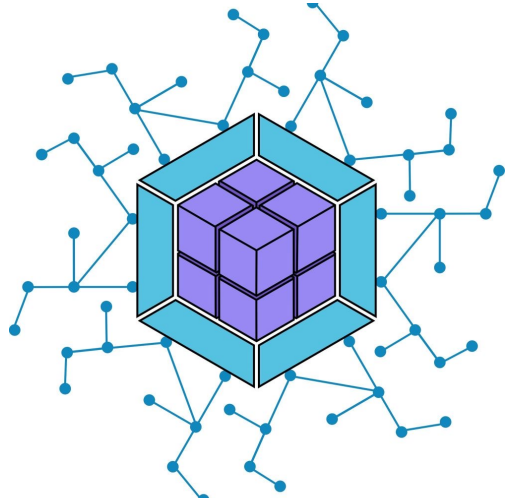
Federated Angular:
Angular, CLI, & Module Federation

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

Angular CLI

?

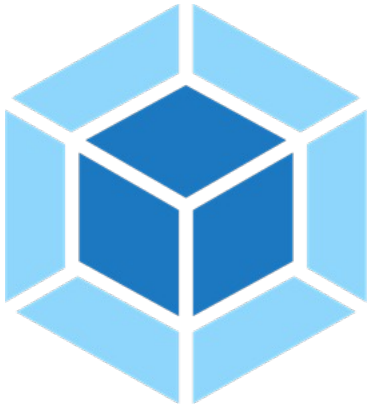
webpack



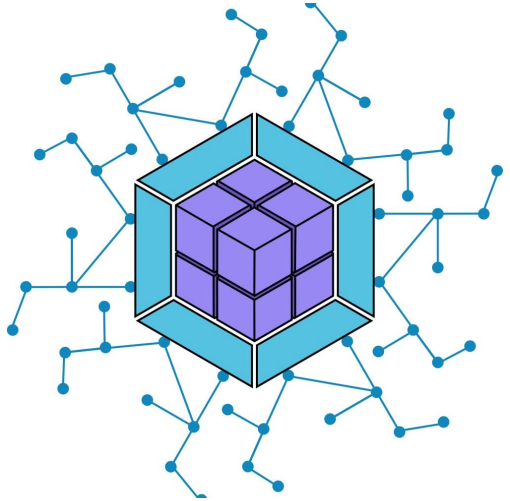
Module Federation
Configuration

Custom Builder

Angular CLI



webpack



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.

Usage

1) ng add @angular-architects/module-federation
2) Use schematic to create configuration
3) Adjust generated configuration
4) ng serve

Module Federation Demo

Lab

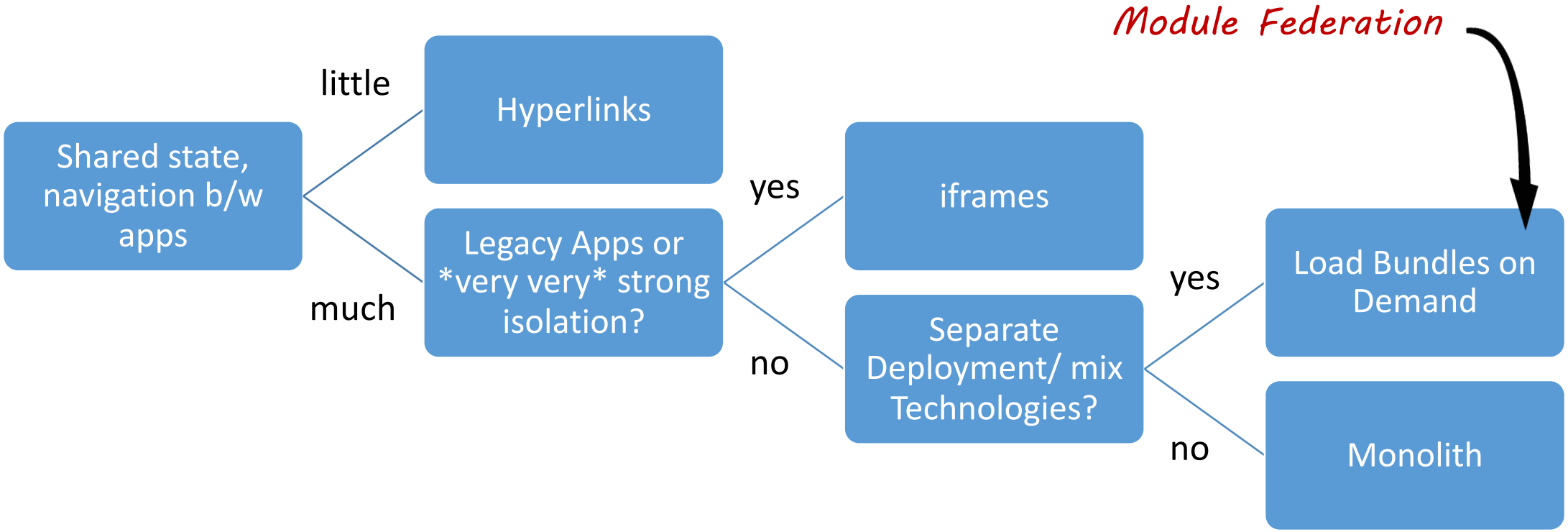
Module Federation

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT


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? ☺