Yandex Ads

Glorious Monolith

Scaling an Application Without Microfrontends

Maksim Zemskov
Lead Frontend Engineer at Yandex

Contents

01 The Scalability Issue

02 Monoliths and Microfrontends

03 What is Glorious Monolith

04 Principles of Building a Scalable Monolith

05 Summary

speaker

maxaz74@gmail.com

linkedin.com/in/maxim-zemskov

Maksim Zemskov

Lead Frontend Engineer at Yandex

- Professional web developer since 2010
- Skilled in Frontend and Fullstack
- Worked at Yandex for 9 years
- Proficient in building new products and maintaining large-scale existing codebases
- Currently leading an infrastructure team at Yandex Direct
- Successfully solved scalability issues for Yandex Direct

Frontend of Yandex Direct

22 Years of Intensive Development

4 Monoliths from Different Ages

50 Frontend Engineers

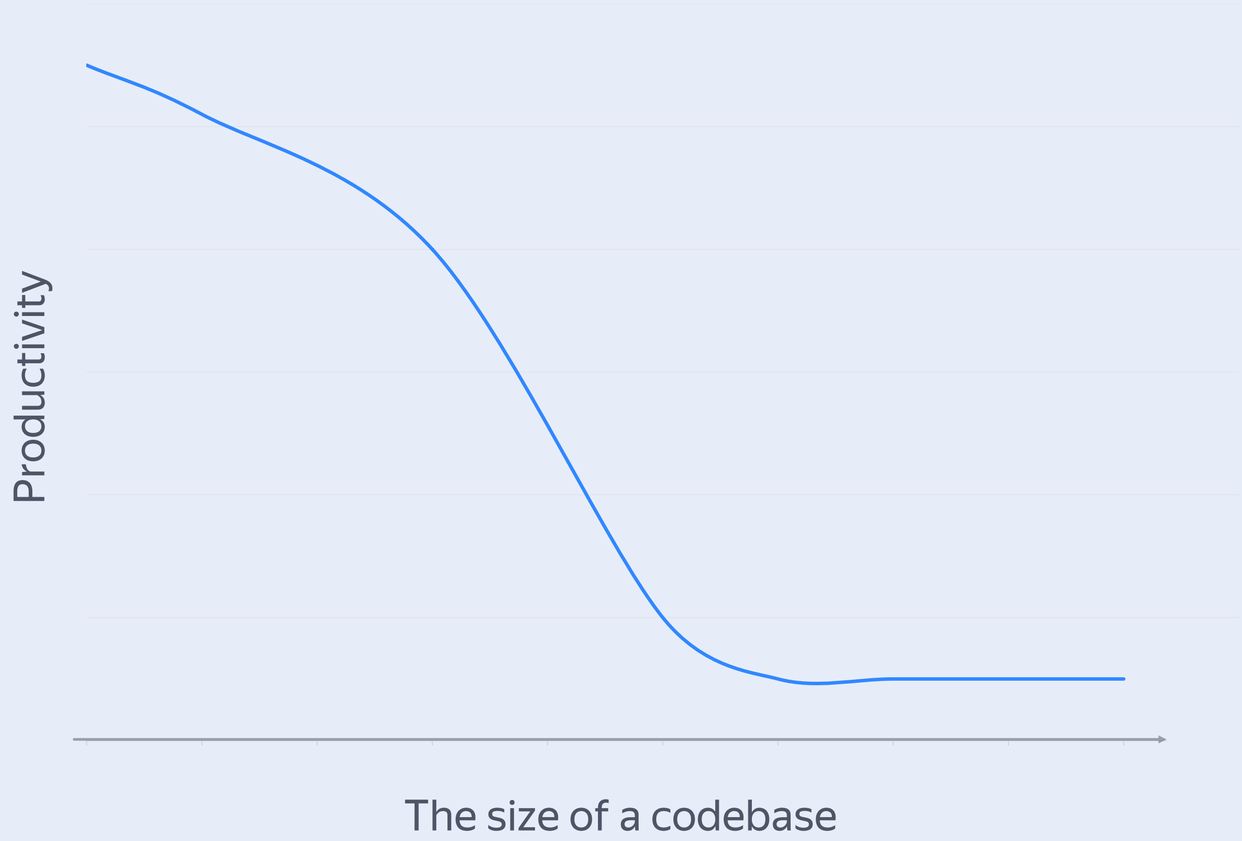
2 500 000
Lines of Code

>100
Complex Pages

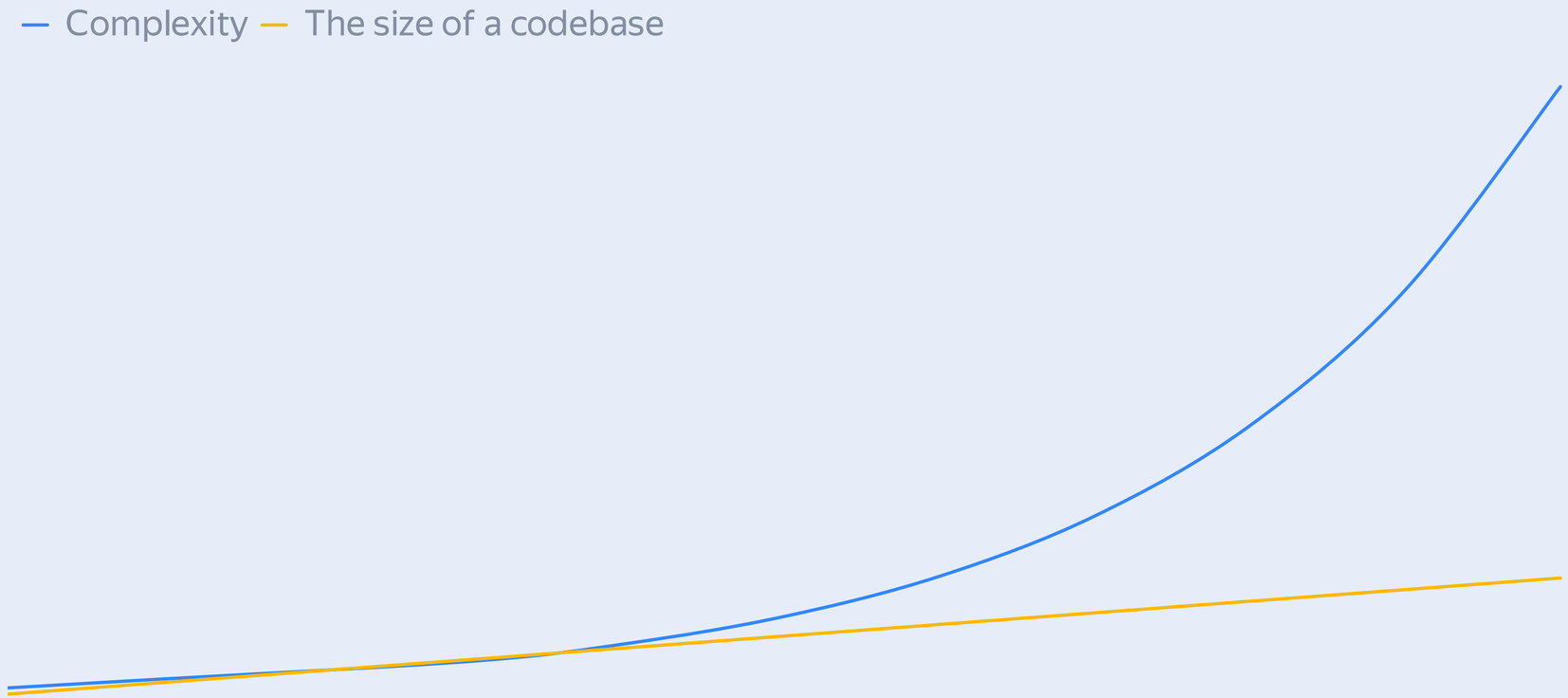
The Scalability Issue

The Scalability Issue

- The project becomes too complex to fully grasp
- Unexpected side-effects and bugs
- Writing new code becomes difficult
- Multiple solutions to the same problems appear
- Any task can only be completed by project experts

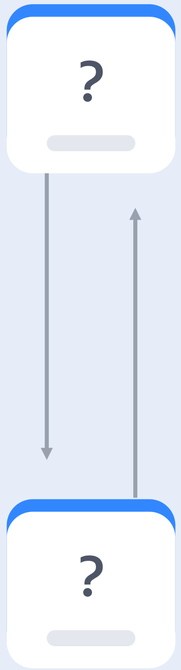


Complexity in a Regular Monolith



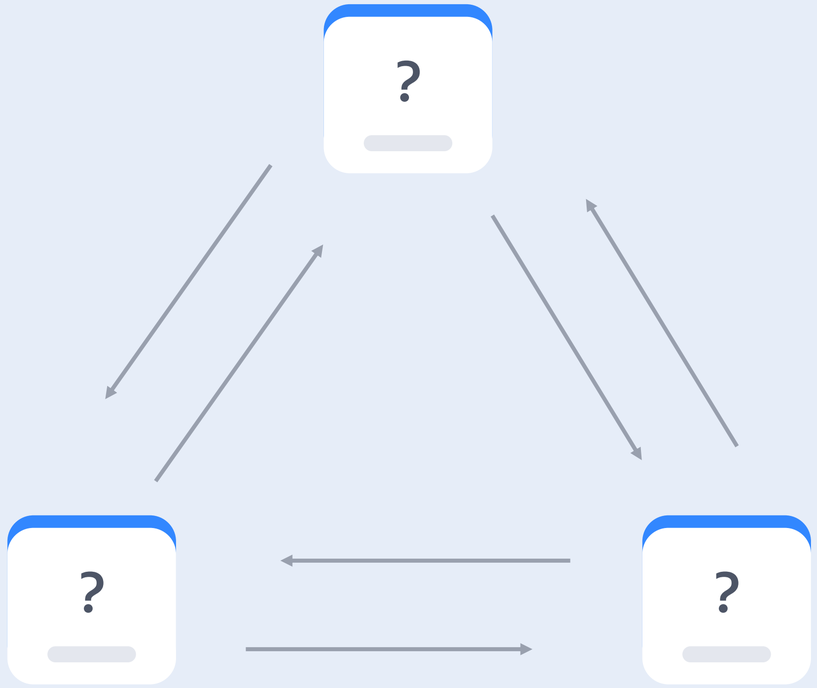
The Source of Complexity

Codebase Size × High Coupling



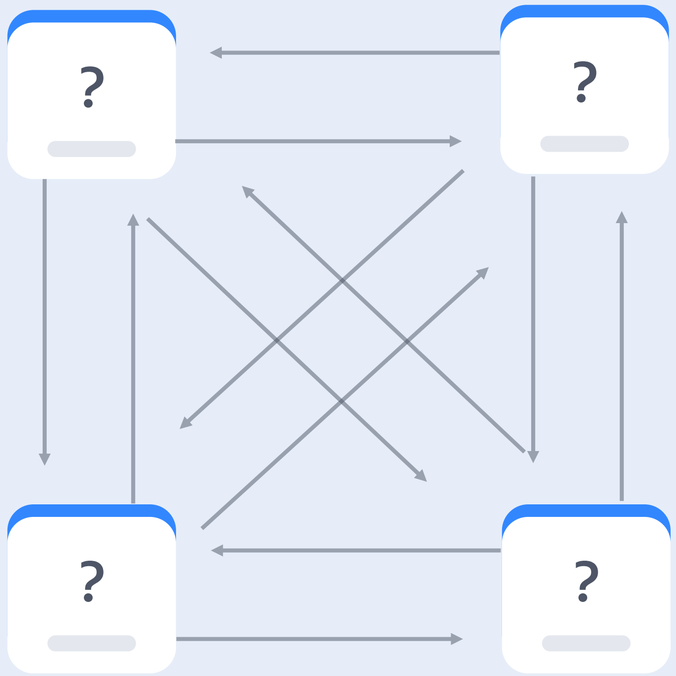
The Source of Complexity

Codebase Size × High Coupling



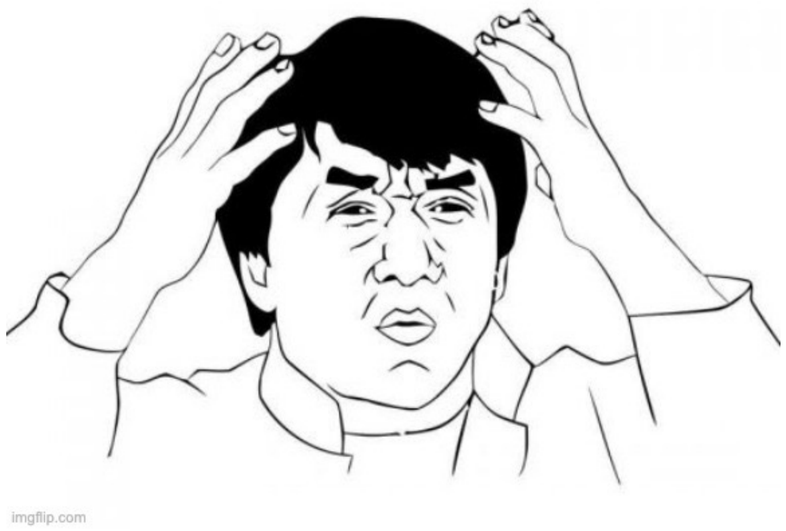
The Source of Complexity

Codebase Size × High Coupling



The Source of Complexity

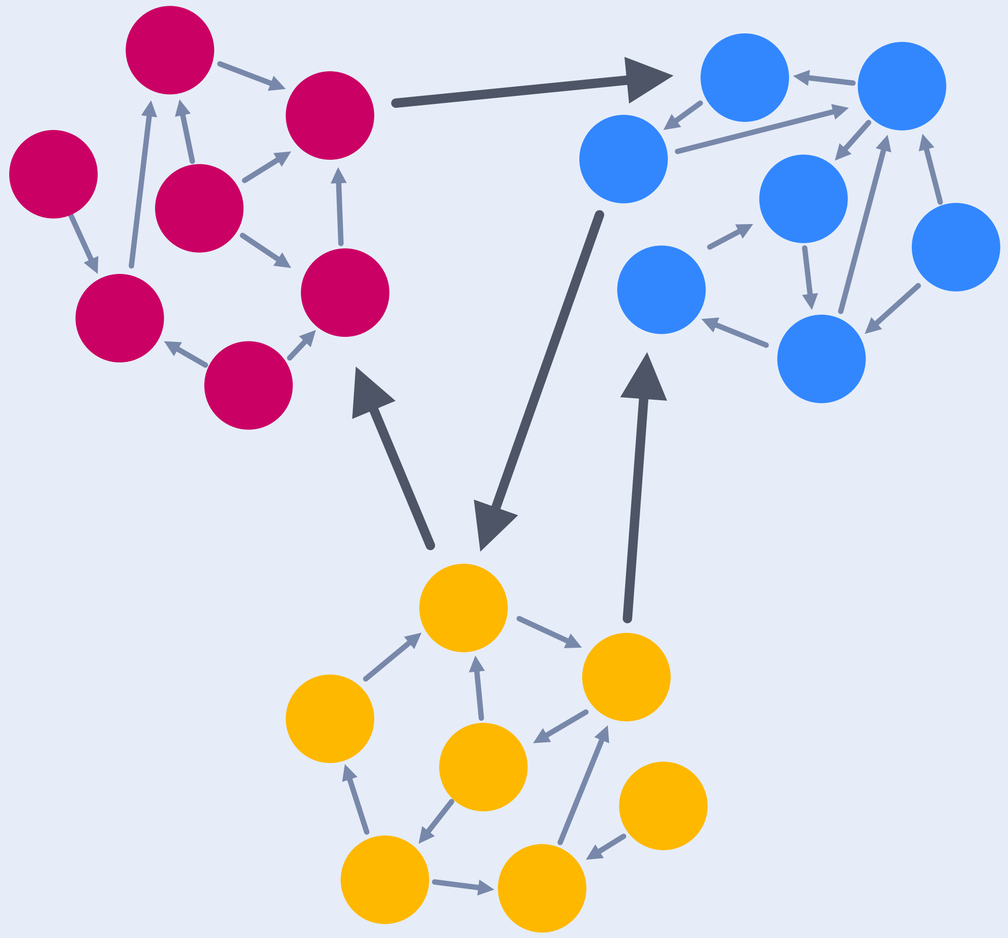
Codebase Size × High Coupling



How About Adding a Microfrontend?

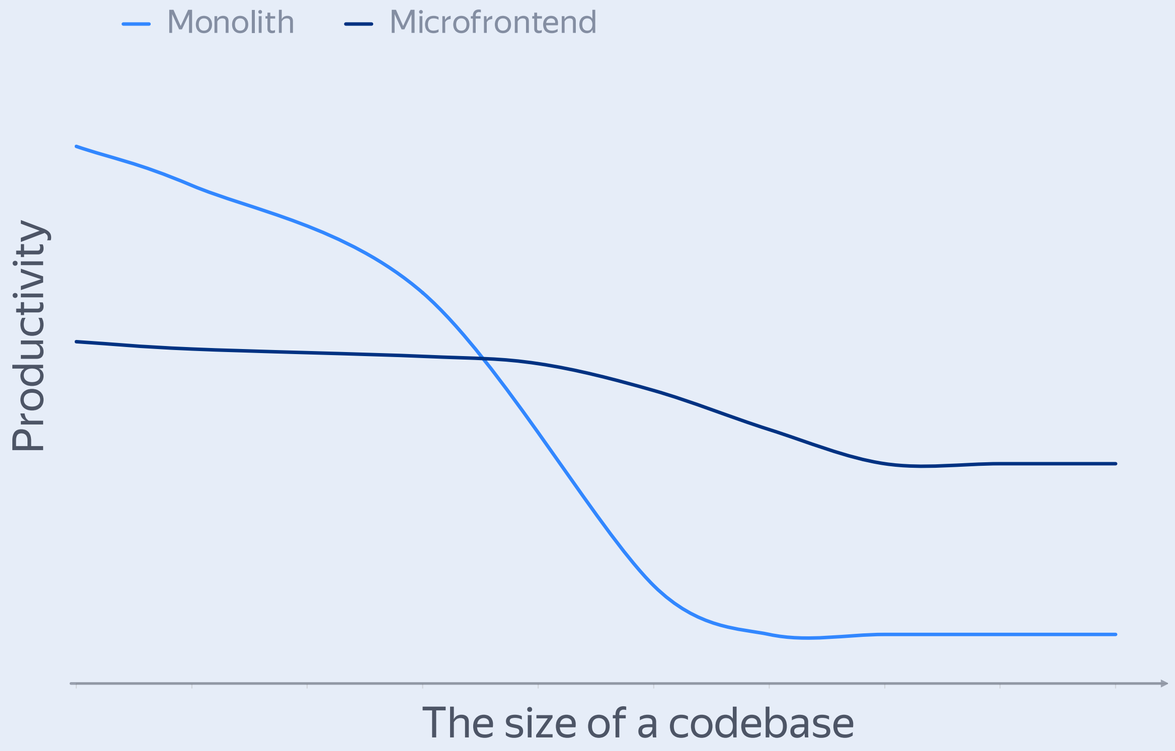
Advantages of Microfrontends

- Low coupling



Advantages of Microfrontends

- Low coupling
- Better scalability



Advantages of Microfrontends

- Low coupling
- Better scalability
- Distribute responsibilities among different teams

Separate technologies and processes for each team

Separate release cycle

— Is Microfrontends Make Every Project Better?

— Not really ☺

The Cost of Microfrontends

- More complex infrastructure
- Harder to reuse code

monorepositiries

package publishing

npm

rollup

webpack

module federation

turbopack

vite

The Cost of Microfrontends

- More complex infrastructure
- Harder to reuse code
- SSR leads to microservices

The Cost of Microfrontends

- More complex infrastructure
- Harder to reuse code
- SSR leads to microservices
- Complex release processes

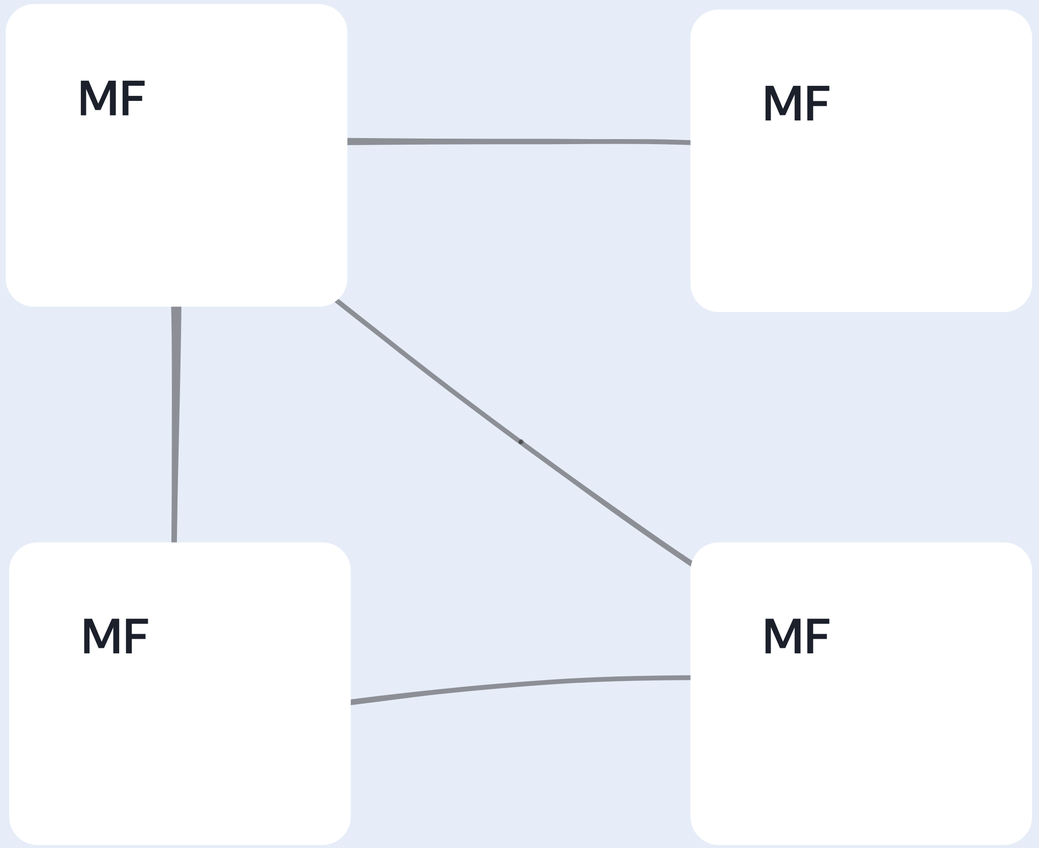
The Cost of Microfrontends

- More complex infrastructure
- Harder to reuse code
- SSR leads to microservices
- Complex release processes
- Duplicated technologies and solutions
- Solving the same tasks by different teams multiple times

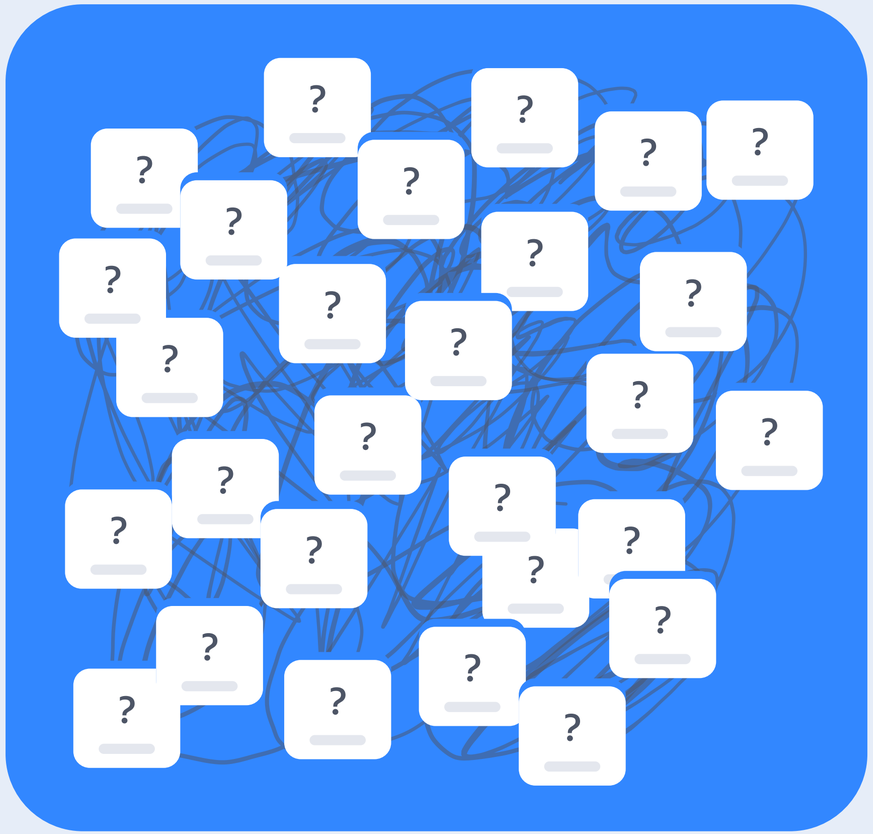
monorepositiries
versions management
package publishing
npm
operational costs
rollup
monitoring
webpack
module federation
tracing
turbopack
vite
microservices
application performance
forward compatibility
load balancing
backward compatibility
bundle size
UI consistency
switching between projects

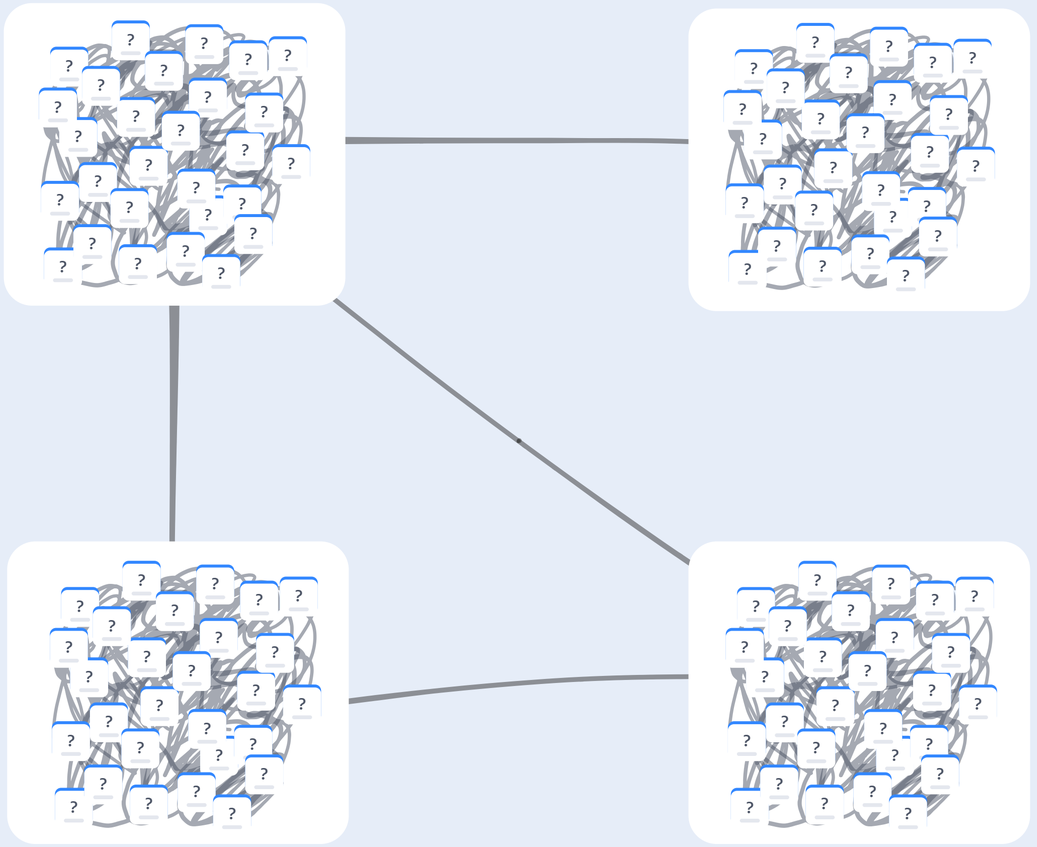
Microfrontends are not always micro

Monolith Application



Microfrontends are not always micro

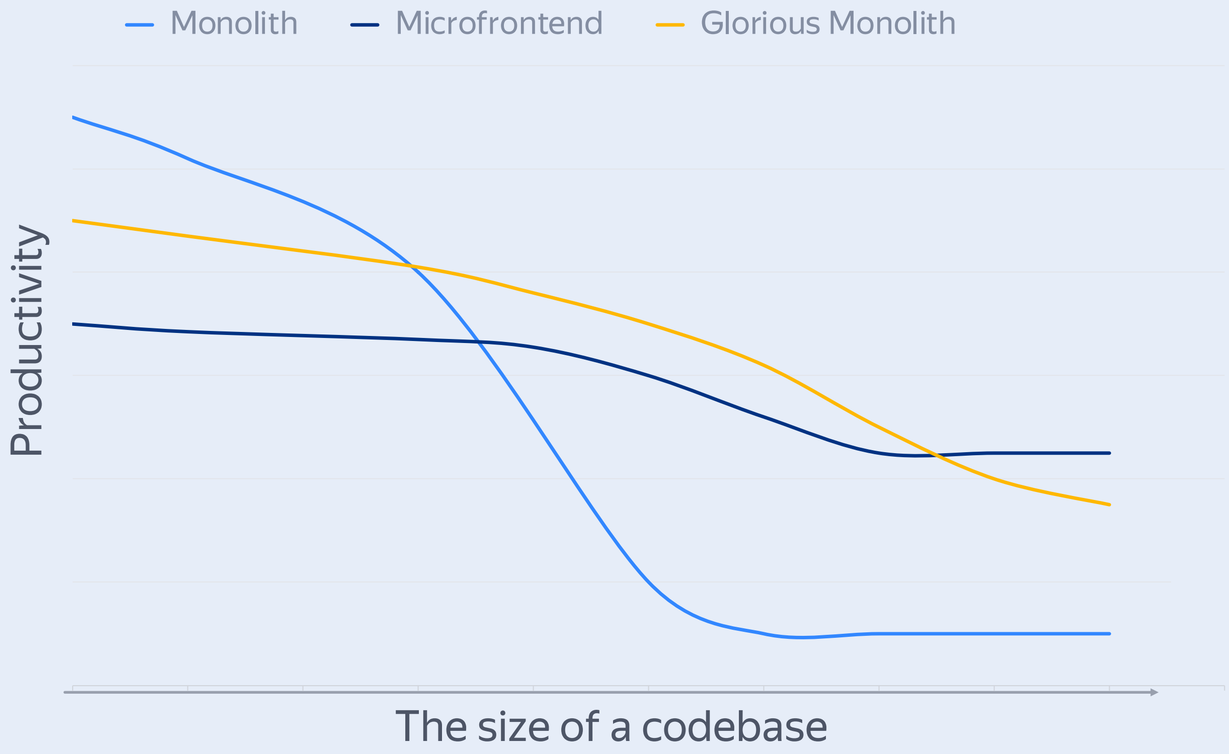




Glorious Monolith

Glorious Monolith

- Relatively easy to implement
- Has excellent scalability



Glorious Monolith

Physical Architecture

Monolith

Logical Architecture

Microfrontend

— How to build
Glorious Monolith?

— Follow the rules
and constraints

The Path to the Glorious Monolith

01

Modules

- Structure
- Isolation
- Public API

02

Runtime for Modules

03

Breaking Code Into Modules

Direct

Add

Max Zemskov

$ 3 926,77

Top up

Overview β

Campaigns

Recommendations 1

Conversions

Statistics

Library

Tools

Information

Search

Administration

Summary of ad performance in Yandex.Direct

Attribution: Automatic

Customize order of widgets

End-to-end analytics from Yandex.Metrica

Data over the past 30 days

Clicks | Conversions | Revenues

12345678

Source: Yandex.Metrica

Other sources	22,873
Clicks from search systems	17,782
Yandex Direct	10,588
Test site 1	377
Test site 2	64
Other ad: identified by tags	52
instagram.com	43
Telegram	32

Direct

Add

Max Zemskov

$ 3 926,77

Top up

Overview β

Campaigns

Recommendations 1

Conversions

Statistics

Library

Tools

Information

Search

Administration

Summary of ad performance in Yandex.Direct

Attribution: Automatic

Customize order of widgets

End-to-end analytics from Yandex.Metrica

Data over the past 30 days

Clicks | Conversions | Revenues

12345678

Source: Yandex.Metrica

Other sources	22,873
Clicks from search systems	17,782
Yandex Direct	10,588
Test site 1	377
Test site 2	64
Other ad: identified by tags	52
instagram.com	43
Telegram	32

Direct

Add

Max Zemskov

$ 3 926,77

Top up

Overview β

Campaigns

Recommendations 1

Conversions

Statistics

Library

Tools

Information

Search

Administration

Summary of ad performance in Yandex.Direct

Attribution: Automatic

Customize order of widgets

End-to-end analytics from Yandex.Metrica

Data over the past 30 days

Clicks | Conversions | Revenues

12345678

Source: Yandex.Metrica

Other sources	22,873
Clicks from search systems	17,782
Yandex Direct	10,588
Test site 1	377
Test site 2	64
Other ad: identified by tags	52
instagram.com	43
Telegram	32

Direct

Add

Max Zemskov

$ 3 926,77

Top up

- Overview β
- Campaigns
- Recommendations 1
- Conversions
- Statistics
- Library
- Tools
- Information
- Search
- Administration

Summary of ad performance in Yandex.Direct

Attribution: Automatic

Customize order of widgets

End-to-end analytics from Yandex.Metrica

Data over the past 30 days

Clicks | Conversions | Revenues

12345678

Source: Yandex.Metrica

Other sources	22,873
Clicks from search systems	17,782
Yandex Direct	10,588
Test site 1	377
Test site 2	64
Other ad: identified by tags	52
instagram.com	43
Telegram	32

Direct

Add

Max Zemskov

$ 3 926,77

Top up

- Overview β
- Campaigns
- Recommendations 1
- Conversions
- Statistics
- Library
- Tools
- Information
- Search
- Administration

Summary of ad performance in Yandex.Direct

Attribution: Automatic

Customize order of widgets

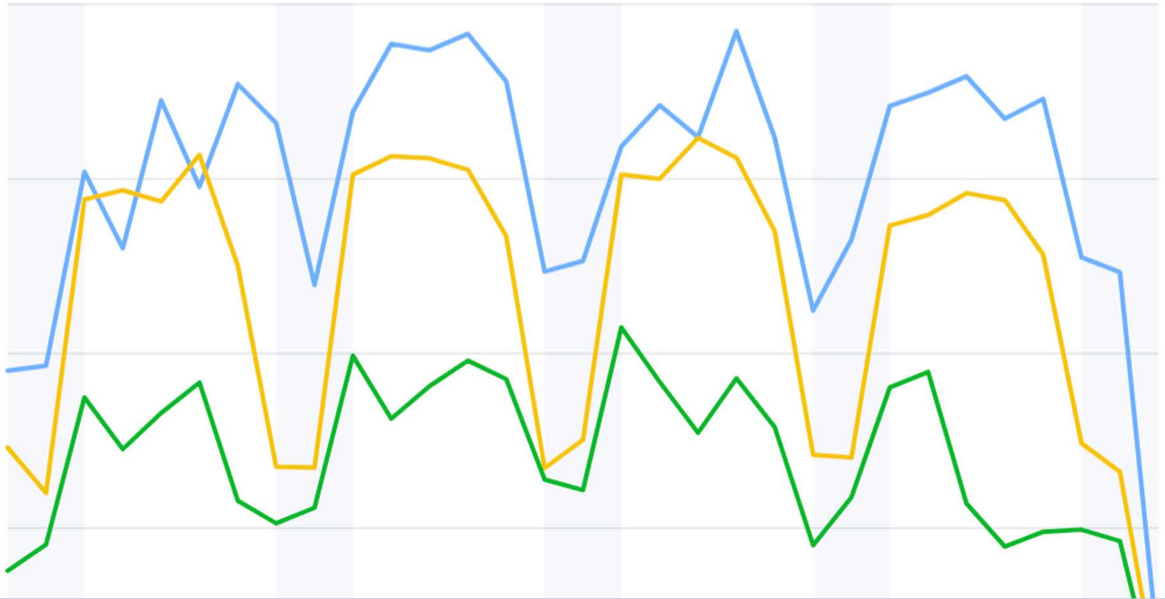
End-to-end analytics from Yandex.Metrica

Data over the past 30 days

Clicks | Conversions | Revenues

12345678

Source: Yandex.Metrica



Other sources	22,873
Clicks from search systems	17,782
Yandex Direct	10,588
Test site 1	377
Test site 2	64
Other ad: identified by tags	52
instagram.com	43
Telegram	32

Requirements for Modules

Module Structure 01

Module Isolation 02

Communication between modules 03

01

Module Structure

What is Inside a Module?

All the code should be contained within one of the modules

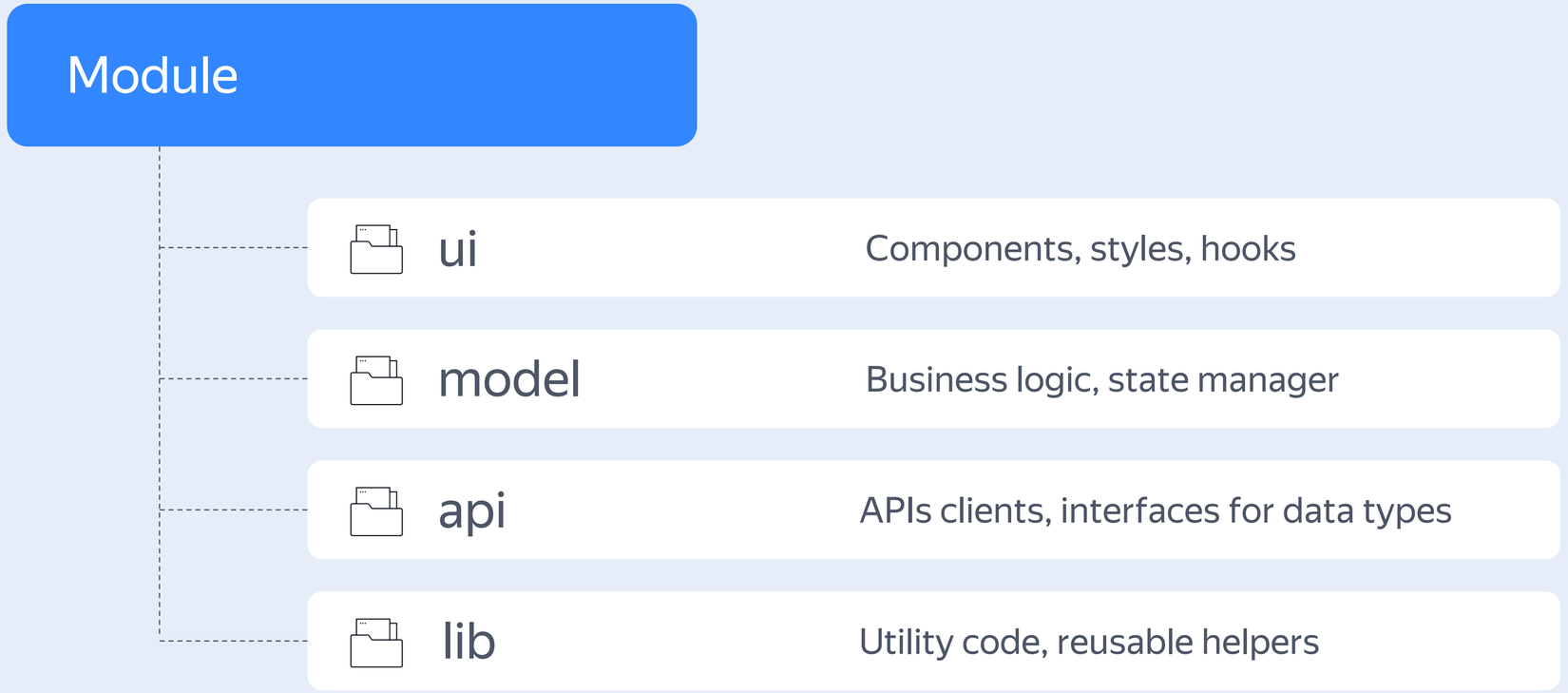
Domain

- UI Components
- Business Logic
- Interaction with APIs
- CSS
- Fonts, Images
- Server-side Code

Infrastructure

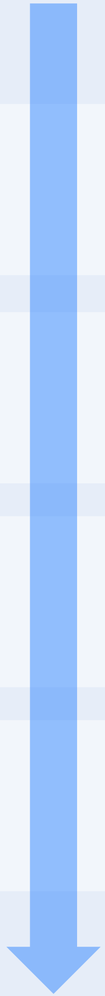
- Any Type of Tests
- Documentation
- Utility Code
- Framework
- Libraries

Module Structure



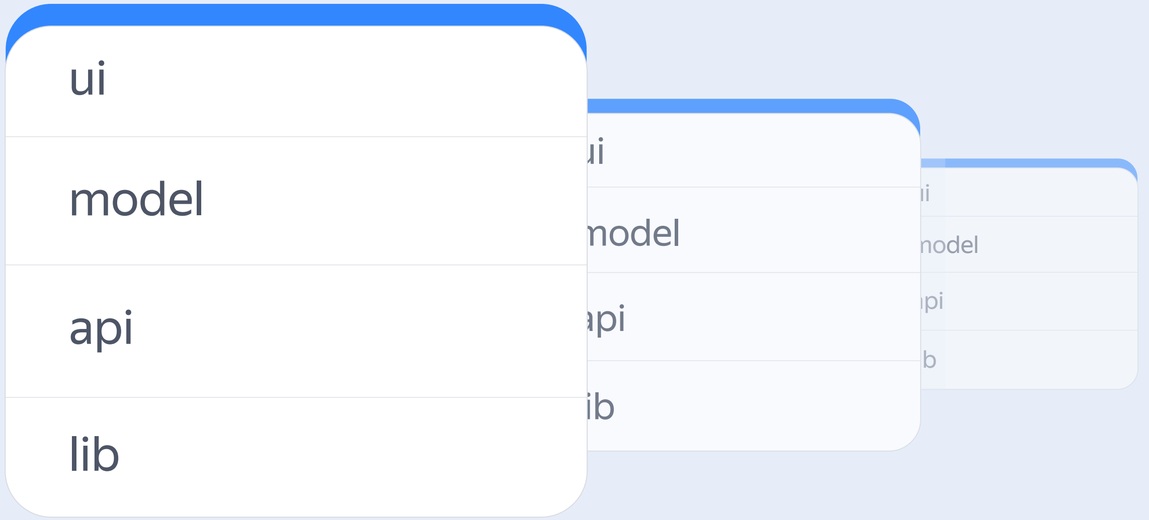
One-way Imports for Segments

Segment	Can Use	Can Be Used By
UI	ui, model, api, lib	ui
MODEL	model, api, lib	ui, model
API	api, lib	ui, model, api
LIB	lib	ui, model, api, lib



The Benefits of Segments

- Unifying the codebase
- Making it easy to decide where to place new code
- Easily switch between different module development
- Collocating related code
- Protecting against mixing UI and business logic



02

Module Isolation

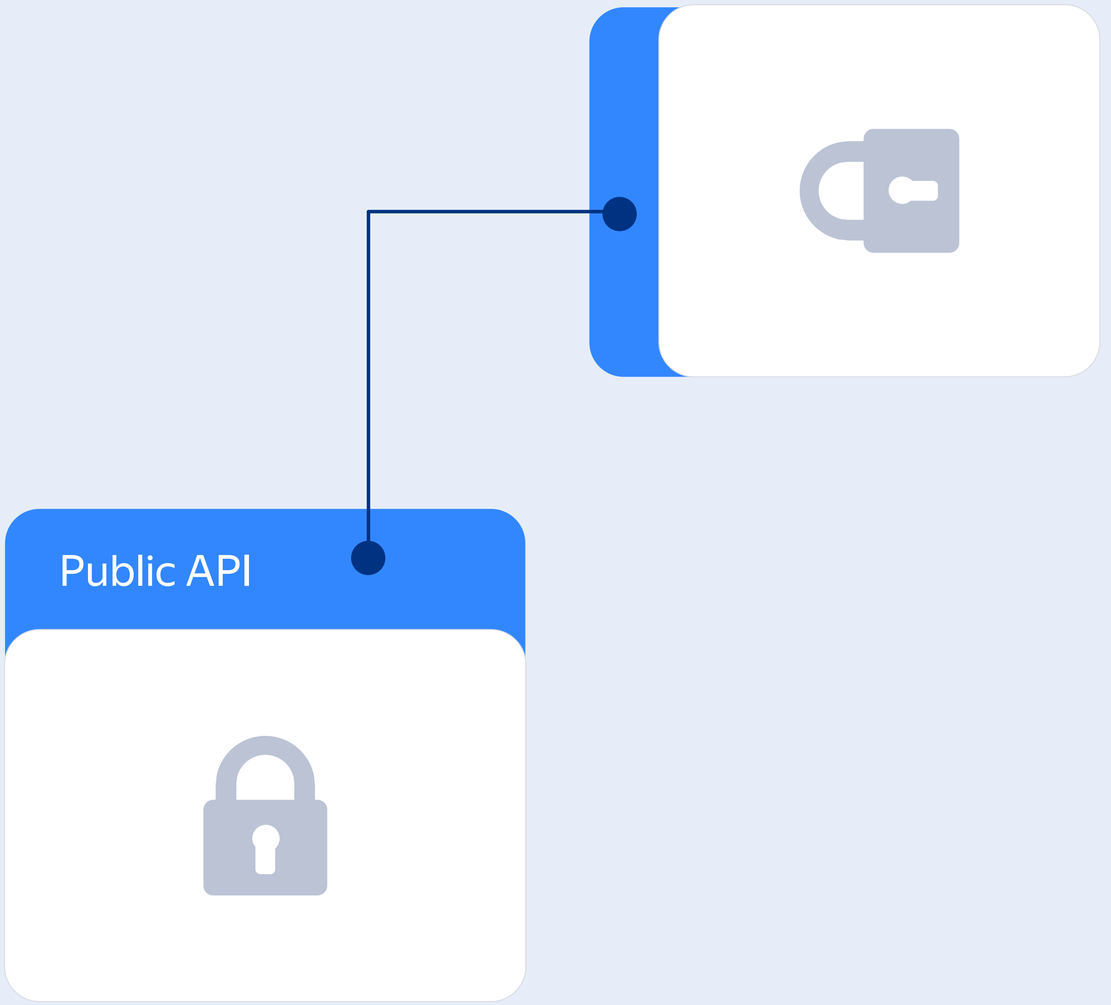
Module Isolation

Why isolate modules?

- Low coupling
- Make changes safely

What should be isolated?

- Code
- Stylesheets
- Data



Code Isolation

Avoiding Global Side Effects

- Mutation of global objects
- Loading polyfills
- DOM mutation

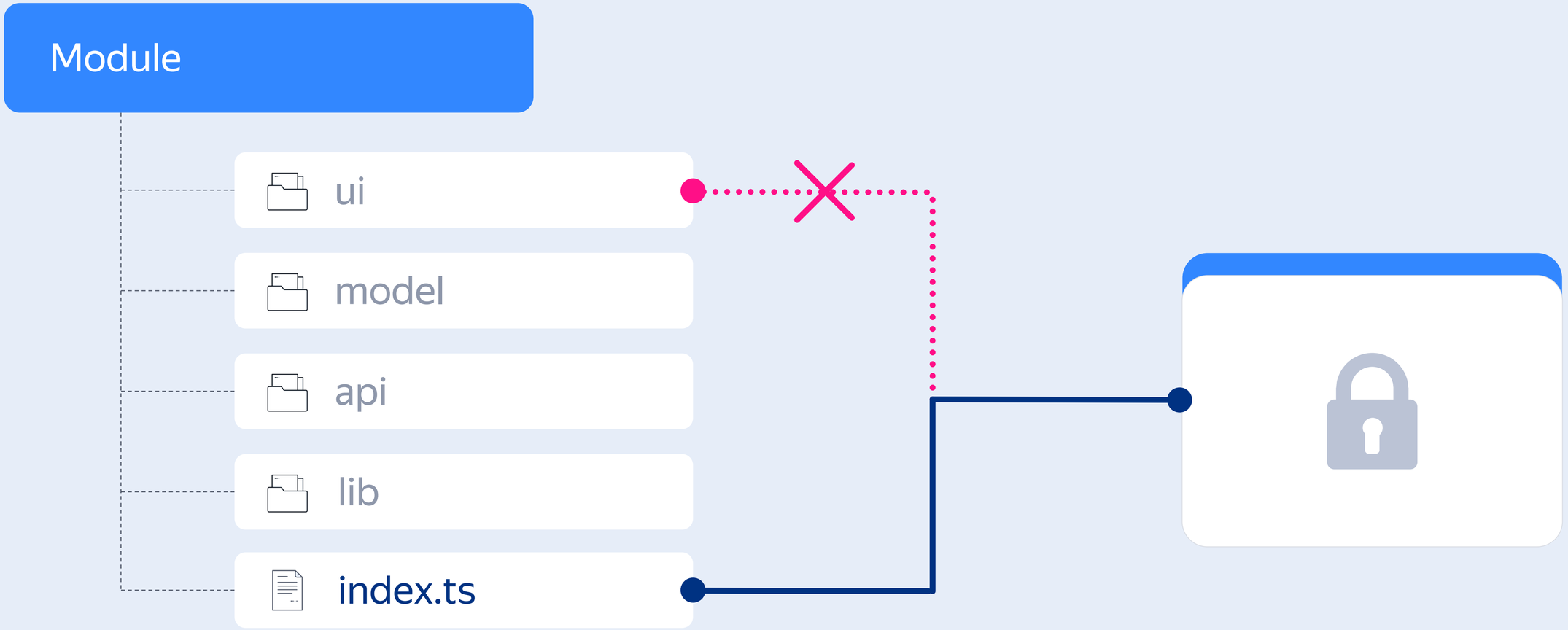
01

Restricting Access to Module's Functionality

- Importing files from a module (code, styles, assets)
- Direct access to the data storage (idb, local storage, etc.)

02

Entrypoint



Public API

ES Modules

- ✓ The simplest option
- ✓ Works well with all tools
- ✗ Stronger dependencies between modules

Dependency Injection

- ✓ Less tight dependencies between modules
- ✗ Requires additional infrastructure

Event-Based Architecture

- ✓ Easier transition to microfrontends
- ✗ Need to manually control module loading sequence

Public API Example

index.ts

```ts
// React Components
export { AddToCart } from './ui/AddToCart';
export { LazyCartView } from './ui/CartView.lazy';

// Actions to modify data in the store
export { clearCart } from './model/actions';

// Data selectors from the store
export { useProductCount, getProductCount } from './model/selectors';

// Event Emitter
export { cartChangedSignal } from './model/events';
```

The Size of Public API

Number of exports

More exports →
more coupling in a system

The number of parameters in exported methods

More parameters →
harder to use the module

Complexity of exported data structures

More data exposed →
harder to change the module

03

Styles Isolation

Styles Isolation

Module A

```
.button {
  margin: 16px 8px 0;
  --button-color: rgb(163, 172, 19
}
```

Module B

```
.button {
  margin: 0;
  --button-color: rgb(67, 94, 202);
}
```

Options to Isolate Styles

01

Shadow DOM

02

CSS Modules

03

CSS-in-JS

04

Tailwind CSS *

* If it is considered as part of the runtime for modules

Example Rules for CSS Modules

- Use only class selectors, pseudo-classes, and pseudo-elements
- Mix styles through the className property in components
- Try to avoid overriding styles for other modules
- Utilize Stylelint to check for rules and constraints

- Avoid resetting browser styles
- Avoid using CSS custom properties
- Avoid importing styles between modules
- Avoid using styles from third-party libraries

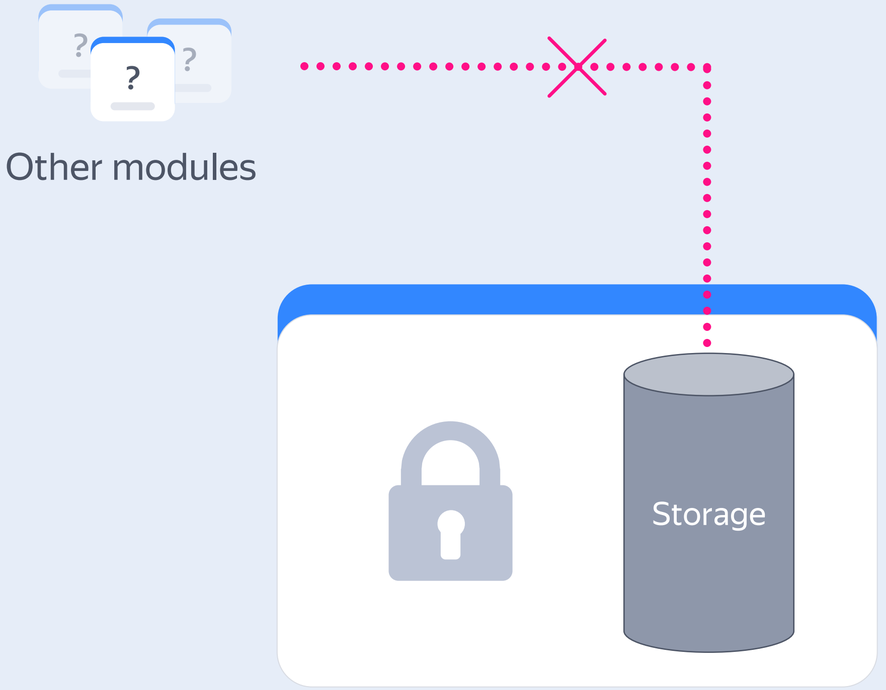
04

Data Isolation

Data Isolation

One Global Storage

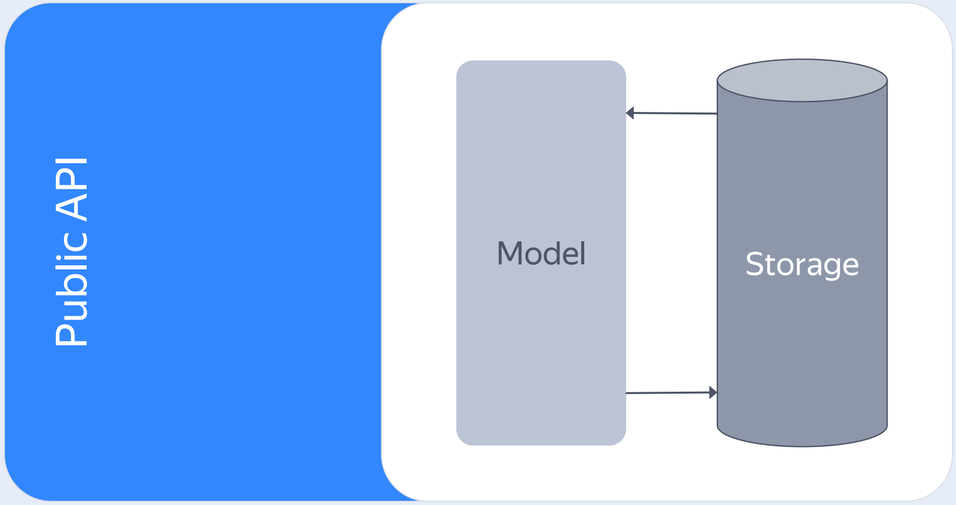
Too Permissive Public API



How to Isolate Data

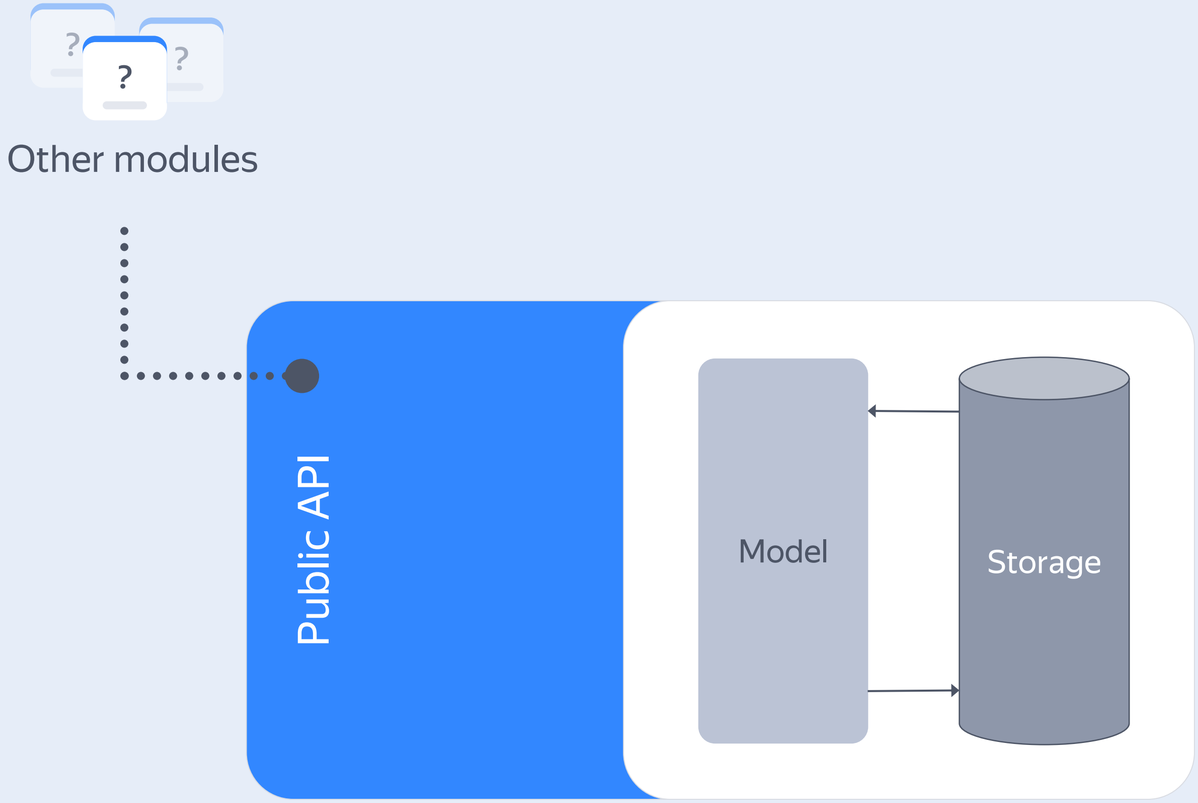
- Data storage within each module

Other modules



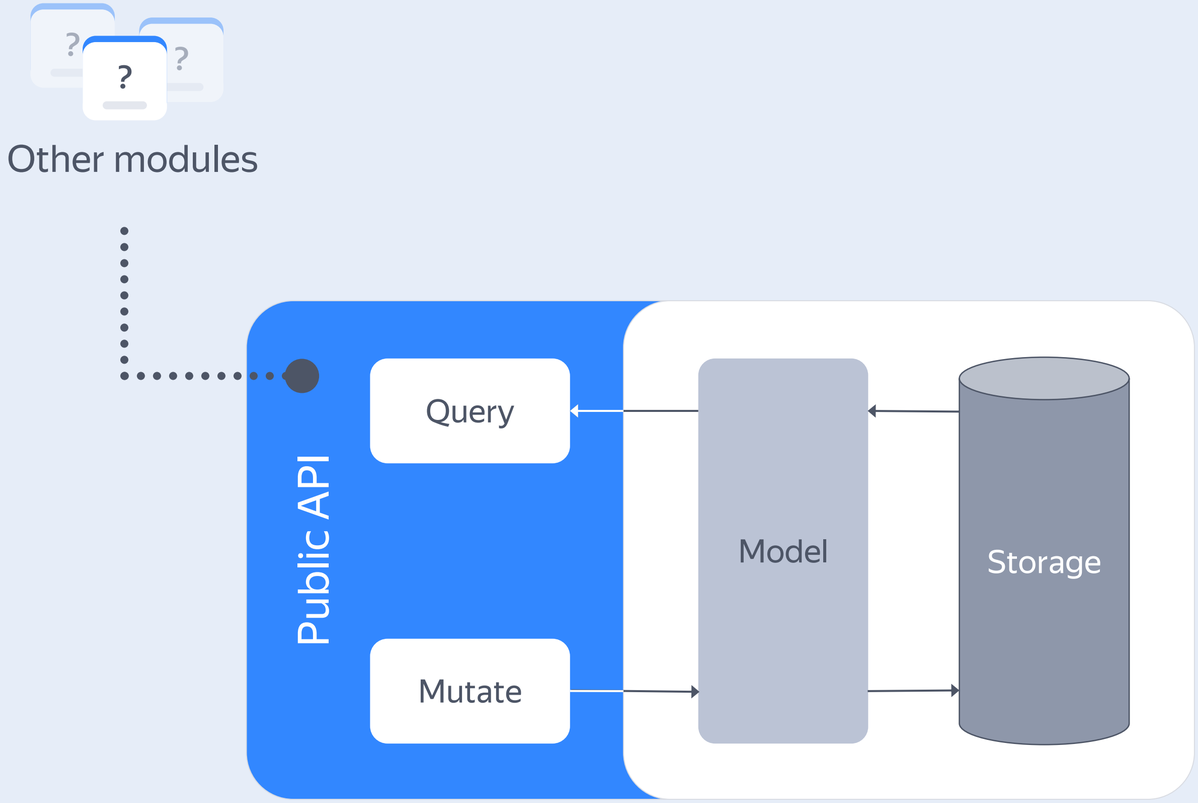
How to Isolate Data

- Data storage within each module
- Data access only through the Public API



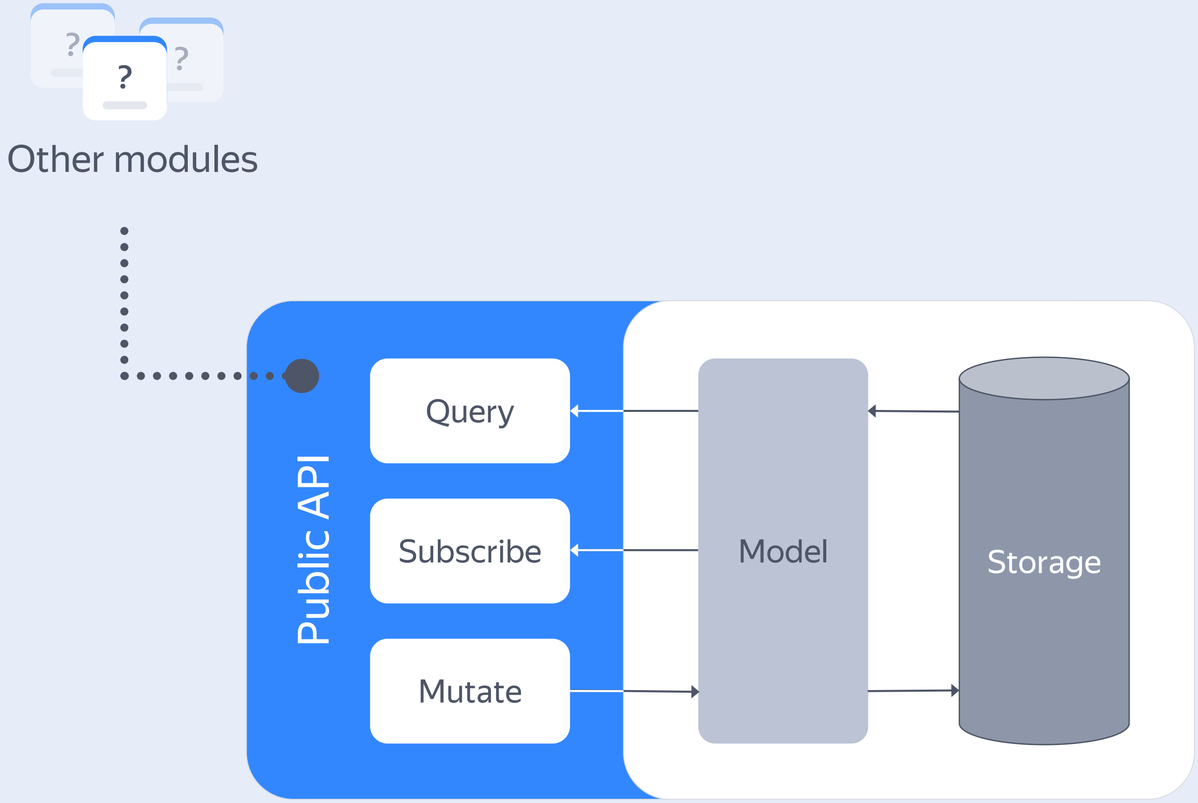
How to Isolate Data

- Data storage within each module
- Data access only through the Public API
- Commands and Queries for getting and mutating data (CQRS)



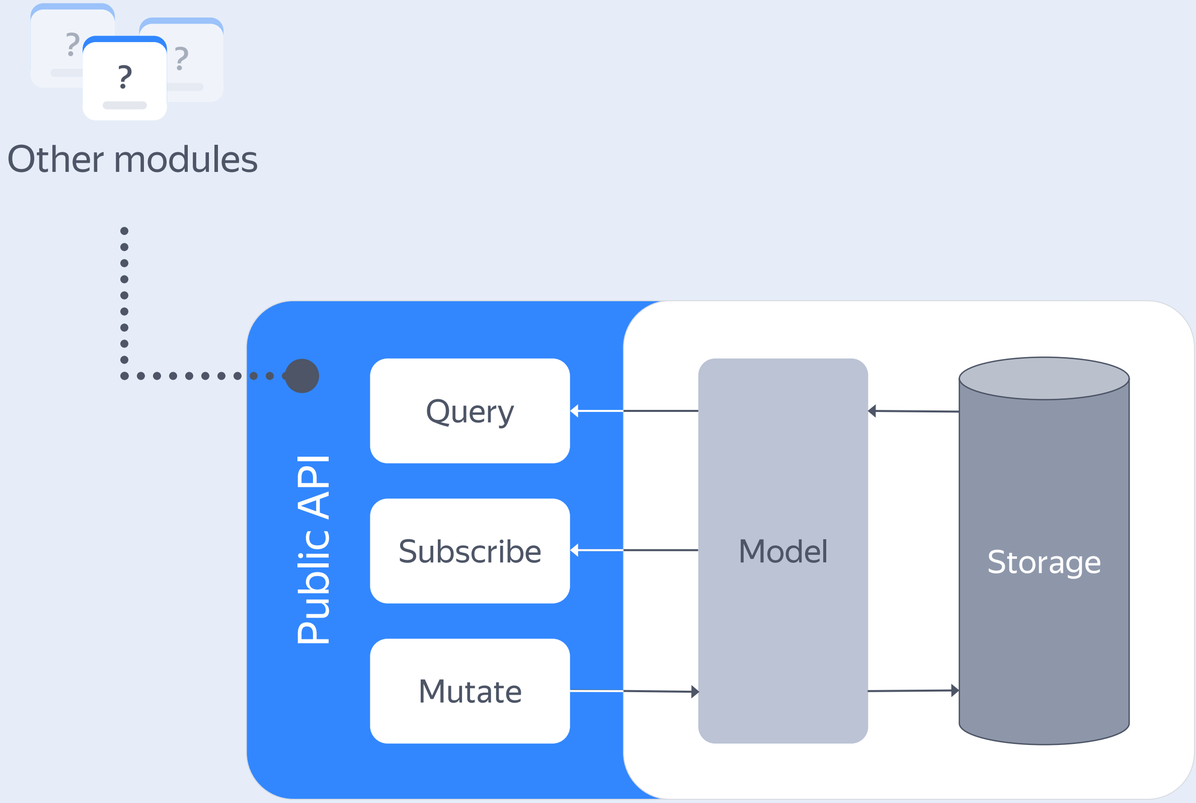
How to Isolate Data

- Data storage within each module
- Data access only through the Public API
- Commands and Queries for getting and mutating data (CQRS)
- Ability to subscribe to data changes



How to Isolate Data

- Data storage within each module
- Data access only through the Public API
- Commands and Queries for getting and mutating data (CQRS)
- Ability to subscribe to data changes
- Protection against modifications: TS Readonly, Object.freeze



The Path to the Glorious Monolith

01

Modules

- Structure
- Isolation
- Public API

02

Runtime for Modules

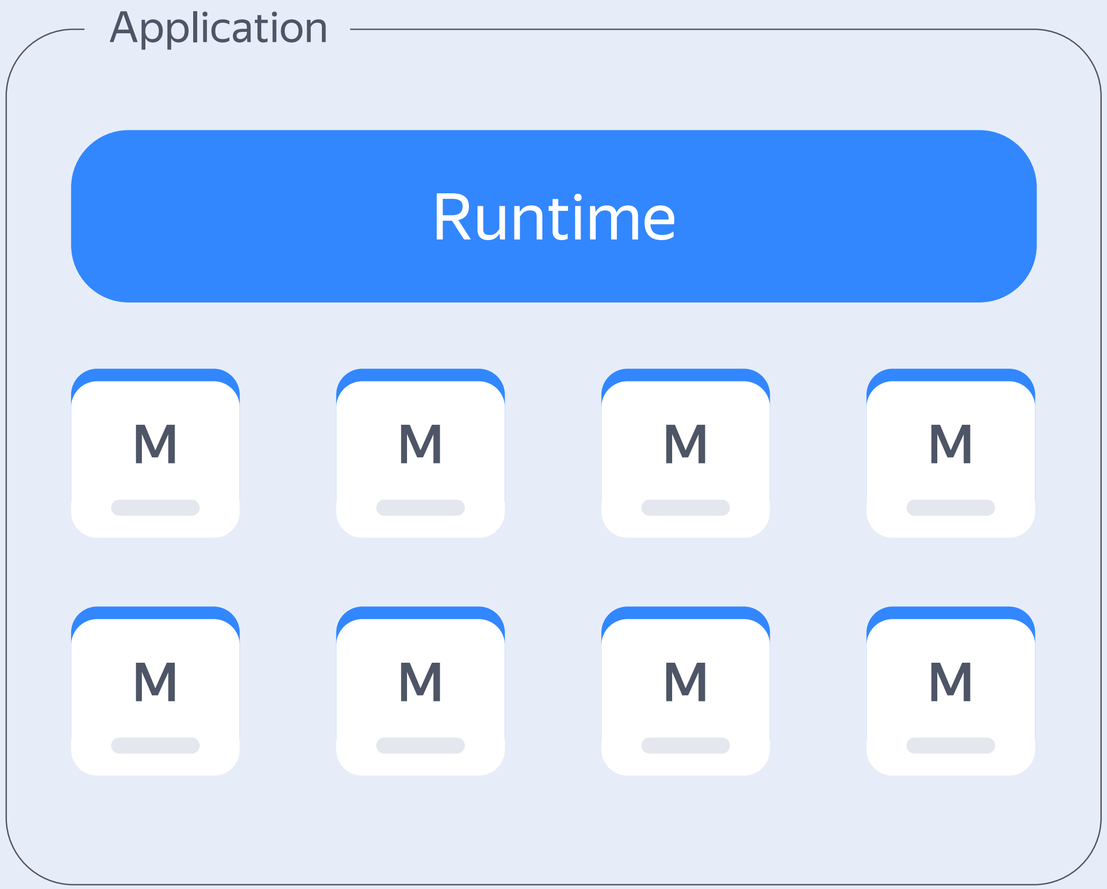
- Environment
- Available dependencies
- Ready-made solutions

03

Breaking Code Into Modules

Runtime for Modules

- One runtime for all modules
- Guarantees on bundling capabilities
- Shared libraries
- Target platforms
- Methods of inter-module communication
- ...



Runtime Example

- TypeScript
- React
- State Manager
- HTTP client
- Router
- Feature Flags
- UI Components Library
- CSS Custom Properties
- Node.js Version for SSR
- Browser Versions
- Bundler and Env Variables
- Error Handling
- Web Vitals
- Libraries from package.json
- Polyfills

The Size of Runtime

Smaller Runtime

- ✓ Less restrictions
- ✗ Challenging module development

Larger Runtime

- ✓ Unified tech stack
- ✗ Difficult to maintain

The Path to the Glorious Monolith

01

Modules

- Structure
- Isolation
- Public API

02

Runtime for Modules

- Environment
- Available dependencies
- Ready-made solutions

03

Breaking Code Into Modules

- Responsibility segregation
- Dependency restrictions
- High cohesion & low coupling

Breaking Code Into Modules

Single Responsibility

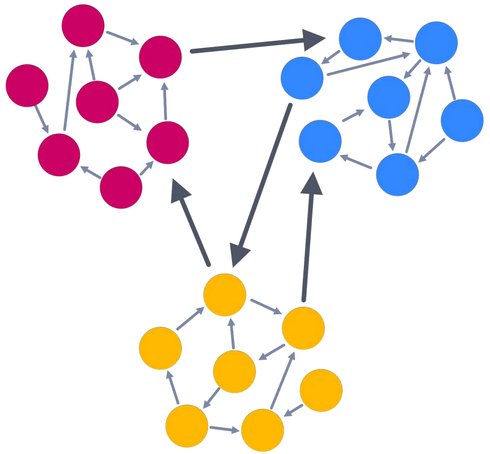
- One clear responsibility
- Related to a specific domain or technical task

Breaking Code Into Modules

Single Responsibility

- One clear responsibility
- Related to a specific domain or technical task

High Cohesion & Low Coupling

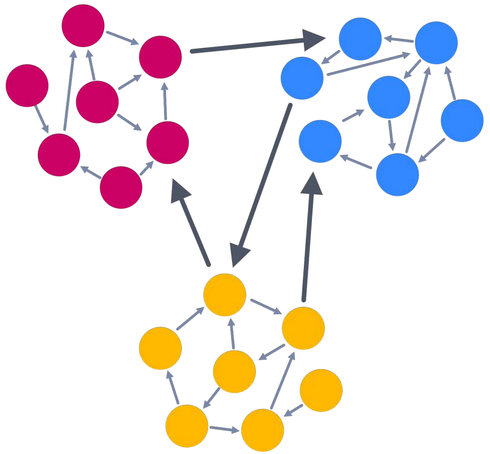


Breaking Code Into Modules

Single Responsibility

- One clear responsibility
- Related to a specific domain or technical task

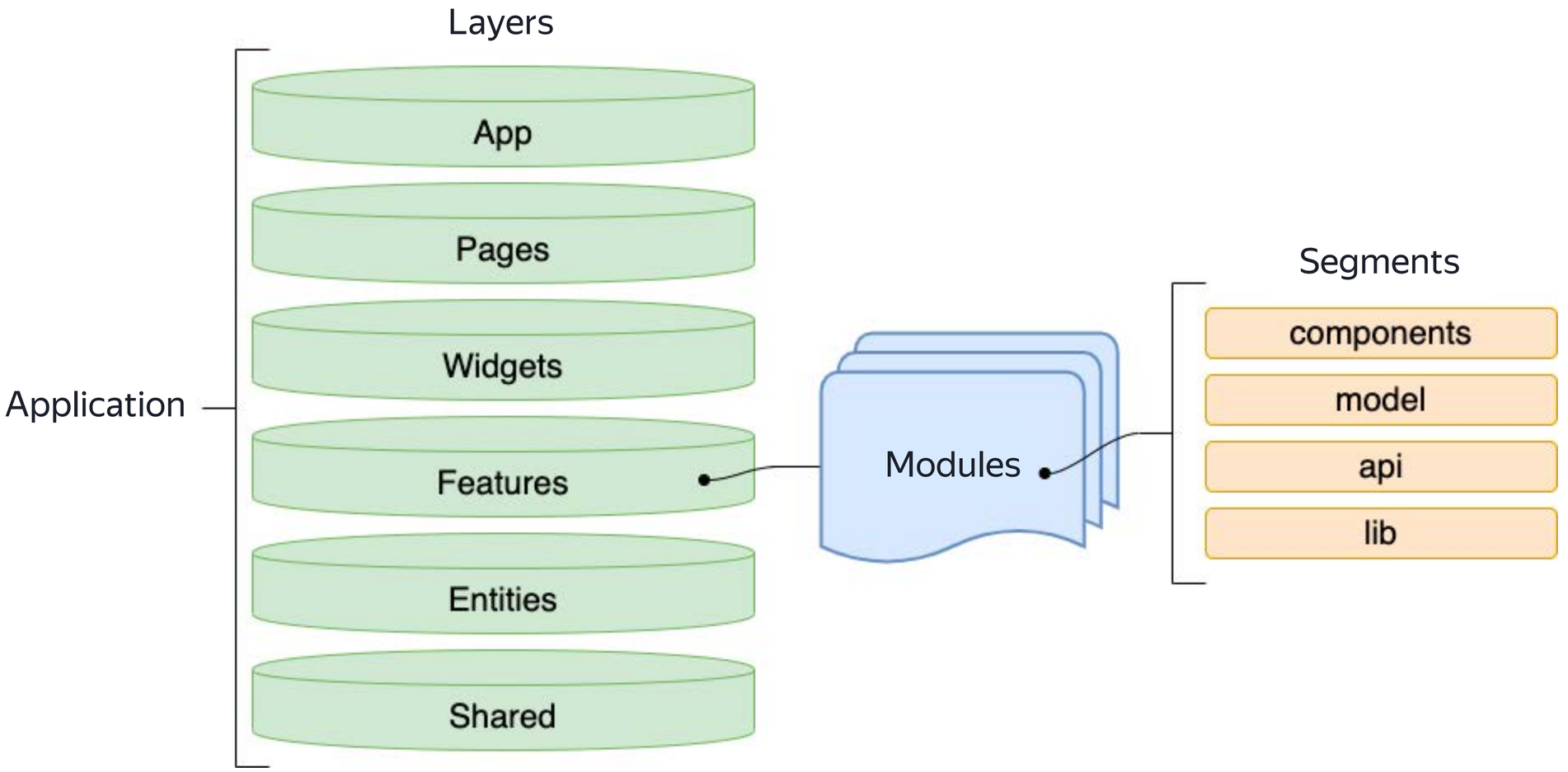
High Cohesion & Low Coupling



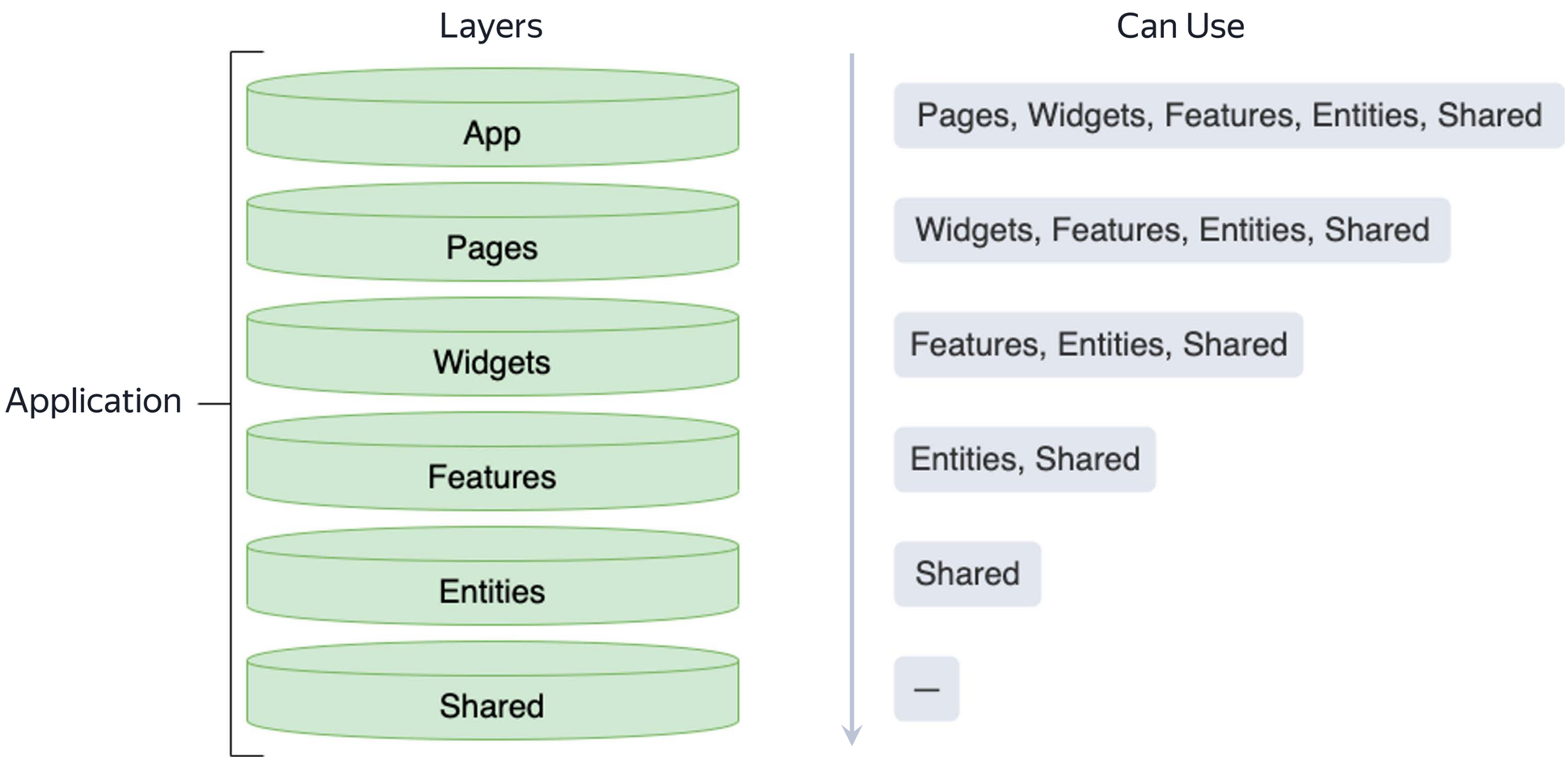
Dependency restrictions

- Division into a set of meaningful groups
- Rules for imports between modules from different groups

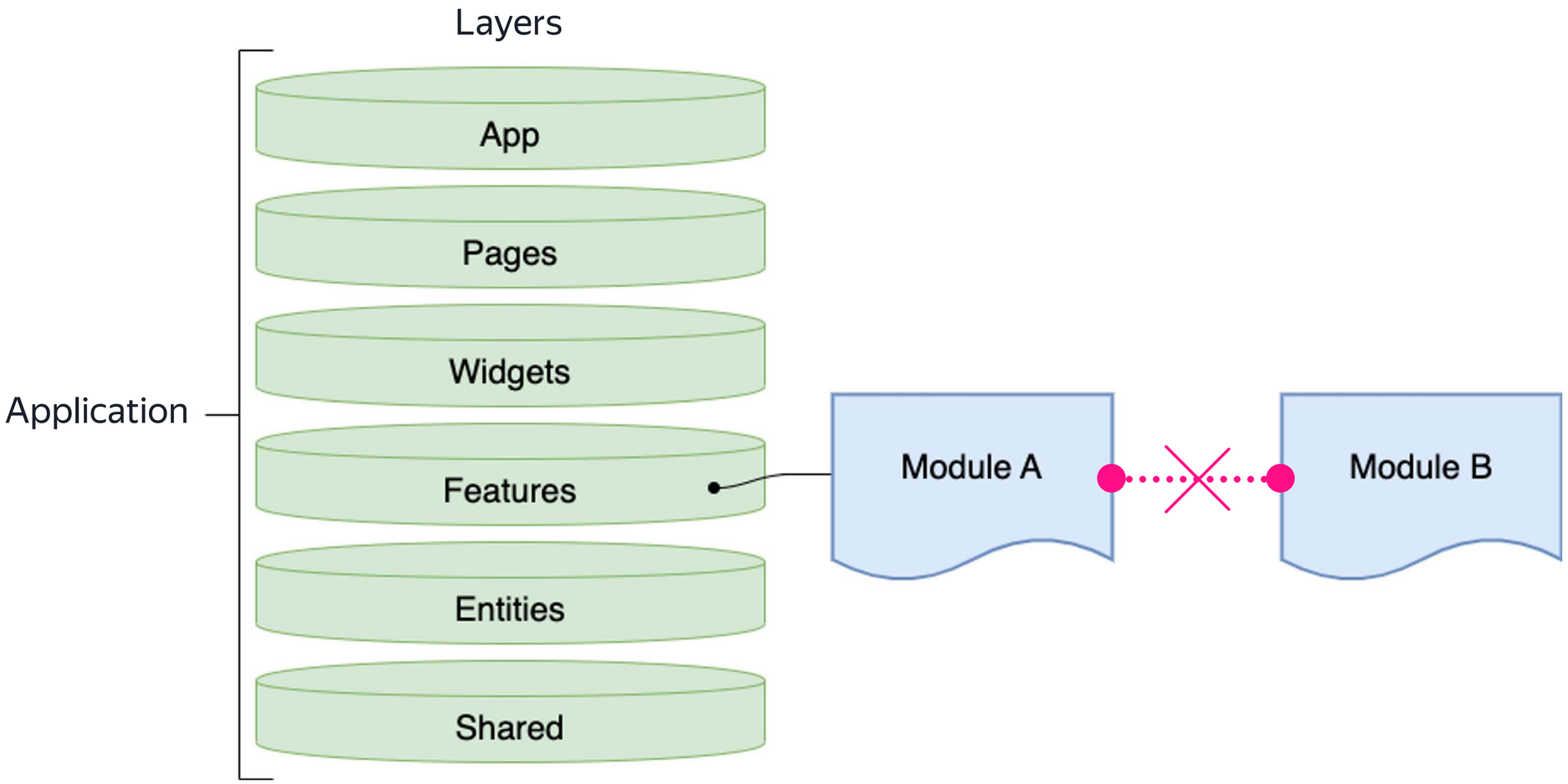
Feature Sliced Design



Направление импортов



Запрет импортов внутри слоя



Summary

Key Principles of the Glorious Monolith

01

Modules

- Structure
- Isolation
- Public API
- Linters

02

Runtime for Modules

- Environment
- Available dependencies
- Ready-made solutions
- Linters

03

Breaking Code Into Modules

- Responsibility segregation
- Dependency restrictions
- High cohesion & low coupling
- Linters

Glorious Monolith

Advantages

- Control of codebase complexity
- Improved Developer Experience
- Relatively simple implementation
- Suitable for projects of any size
- Suitable for starting a new project

Primary Expenses

- Time for getting familiar with rules and constraints
- Time for setting up linters
- Fine-tuning devtools performance * (bundler, TypeScript, etc)

  * only for exceptionally large projects

You may not need to adopt microfrontend

What's Next?

Learn

- Modular Monolith
- Domain Driven Design
- Clean Architecture
- Feature Sliced Design
- CQRS

Use

- TypeScript
- eslint + custom rules
- eslint-plugin-boundaries
- eslint-plugin-import
- stylelint + custom rules
- dependency-cruiser

Yandex Ads

Thank you!

Maksim Zemskov
Lead Frontend Engineer at Yandex

maxaz74@gmail.com
linkedin.com/in/maxim-zemskov