Micro-Frontends Under The Microscope

Last year, I was interviewing managers...

“What **trends** do you see in web development?”

Every single one mentioned **micro-frontends** as a trend to watch

Why all the hype?

Micro-Frontends promise to make building complex applications **easier** and your team **more efficient**

Do they live up to that **promise**?

Micro-Frontends Under The Microscope

Trent Willis

Staff Software Engineer, Netflix

Worked on several micro-frontend apps of varying complexities and scale

What are micro-frontends, **really**?

“An architectural style where independently deliverable frontend applications are composed into a great whole.”

- Cam Jackson, martinfowler.com

"An architectural style where independently deliverable frontend applications are composed into a great whole."

- Cam Jackson, martinfowler.com

"A design approach in which a front-end app is decomposed into individual, semi-independent "microapps" working loosely together."

- Bob Myers, Toptal

"A type of architecture where a web application is divided into different modules or individual functions, implemented autonomously."

- Aplyca

"An architectural and organizational style (NOT a specific technology!!!) in which the front-end of the app is decomposed into individual, loosely coupled "micro apps" that can be built, tested, and deployed independently."

- AltexSoft

"An approach to building applications where the front-end is broken down into smaller, independent parts, each with its own user interface and functionality. These independent parts are then integrated to form a complete application."

architecture composed independent

What does it mean for apps to be independent?

"...scaling frontend development so that many teams can work simultaneously on a large and complex product is even harder...[micro-frontends] can increase the effectiveness and efficiency of teams working on frontend code."

- Cam Jackson, martinfowler.com

"...scaling frontend development so that many teams can work simultaneously on a large and complex product is even harder...[micro-frontends] can increase the effectiveness and efficiency of teams working on frontend code."

- Cam Jackson, martinfowler.com

"[Micro-Frontends] are owned by independent teams. Each team has a distinct area of business or mission it cares about and specialises in."

- Michael Geers, micro-frontends.org

"Its purpose is to eliminate the dependency between work teams, which slows down development and increases the complexity of the digital product."

- Aplyca

"Using this type of architecture, the monolith team gets split to separate independent teams, which helps improve scalability, code complexity, etc., as each team works on a specific feature of the application separately."

- XenonStack

"A microfrontend is made of components — owned by different teams — that can be deployed independently...no single team owns the UI in its entirety."

What does it mean for apps to be independent?

They enable teams developing the apps to work independently

What are micro-frontends, **really**?

Any application architecture where an app is composed from sub-apps so that teams can work independently

If your teams are **not independent**, micro-frontends are **failing to deliver on their promises**

But, aren’t they just like **microservices**?

No.
Micro-Frontends are **hyphenated**.
Microservices are not.

They sound quite similar...

An architectural style that structures an application as a collection of services that are:

- Independently deployable
- Loosely coupled
- Organized around business capabilities
- Owned by a small team

There are 2 important differences

Microservices do not share a **runtime**.

Micro-Frontends do.

*“**Isolate Team Code** - Don’t share a runtime, even if all teams use the same framework. Build independent apps that are self contained. Don’t rely on shared state or global variables.”*

- Michael Geers, micro-frontends.org

Sharing a runtime makes isolation and independence more **difficult**

Microservices do not need to deliver a **seamless user experience**.

Micro-Frontends do.

Micro-Frontends have important differences from microservices that make it **harder to be independent**

Micro-Frontends have important differences from microservices that make it **harder to live up to the hype**

What happens in reality?

First, you decide on a **technology**...

What technology?

- Webpack Module Federation
- Emerging Frameworks (single-spa, qiankun, piral, luigi, etc.)
- Framework Specific (Ember Engines, etc.)
- Custom (Dynamic Imports, Iframes, etc.)

What technology?

- Webpack Module Federation
- Emerging Frameworks (single-spa, qiankun, piral, luigi, etc.)
- Framework Specific (Ember Engines, etc.)
- Custom (Dynamic Imports, Iframes, etc.)

There is **no** **“standard”** micro-frontends technology, **yet**

Lack of **standardization** means
lack of **best practices**

The **pioneer tax** is still high today

TypeScript Integration

- How do you share types for the connection points between micro-frontends?
- Not a problem if truly isolated, but most apps are not

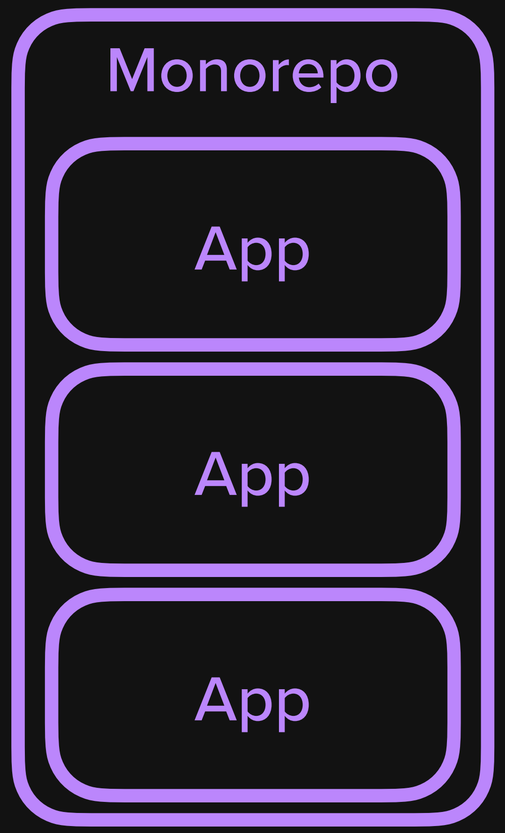
A “**shared**” (type definition) package can quickly become a monolithic dumping ground

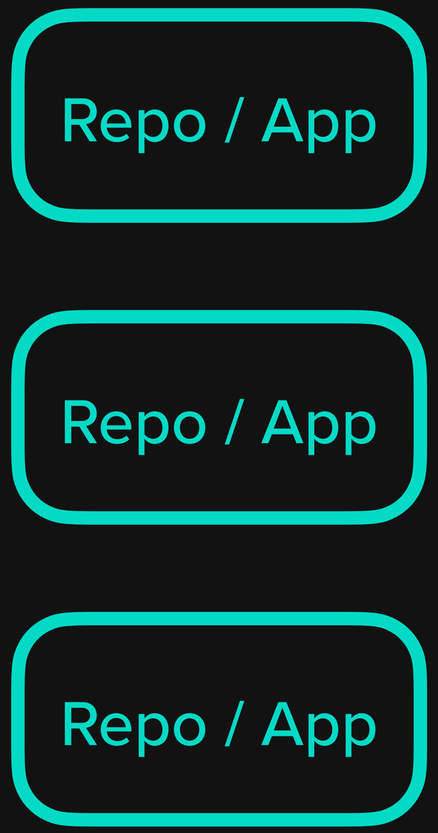
@module-federation/typescript

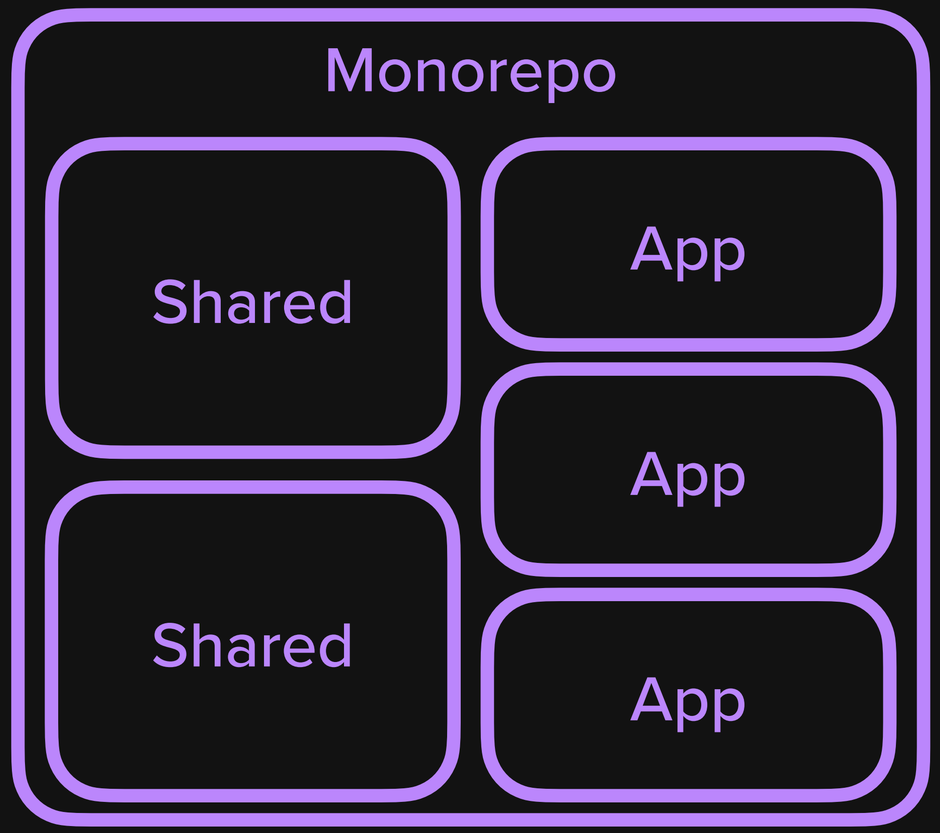
Complex apps could **benefit the most** from micro-frontends, yet they are also the most likely to run into **edge cases**

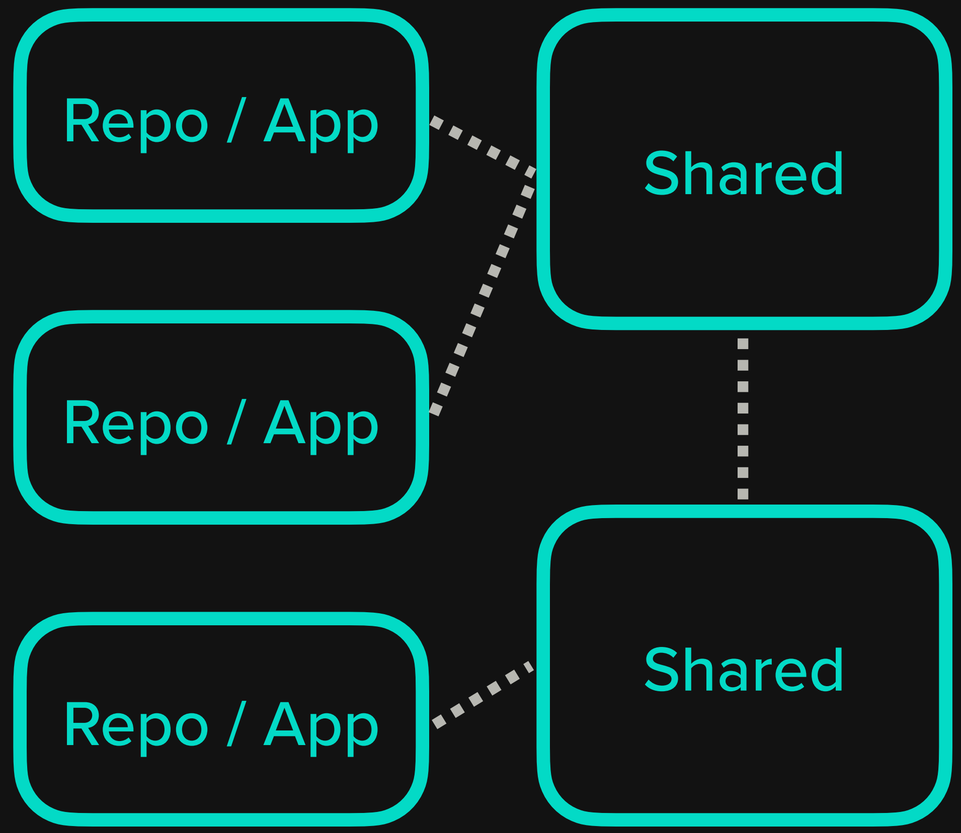
What repository strategy?

- Separate repos (multirepo) vs. monorepo
- A highly consequential decision we often make implicitly


Monorepo
App
App
App


Repo / App
Repo / App
Repo / App


Monorepo
Shared
App
App
Shared
App


Repo / App
Repo / App
Repo / App
Shared
Shared

How much code do you intend to **share** between apps?

Sharing code and micro-frontends both intend to make development more efficient, but together they can actually make it less efficient

Monorepo

- Less overhead to share code
- Requires investment in tooling/processes to keep teams independent
- More likely for apps to be tightly coupled, but less slow down when that happens

Multirepo

- More overhead to share code
- Tools and processes for each repo are independent
- Less likely for apps to be tightly coupled, but more slow down when that happens

If you can’t have **dedicated support** for your architecture, you probably don’t need micro-frontends

Each strategy has **trade-offs**. Choosing the wrong one may negate the intended benefits.

Isolated **Runtime Dependencies**

- Isolation leads to duplicate dependencies
- Will multiple versions running side-by-side work?

The browser has **non-isolated state** no matter how well your micro-frontends are isolated

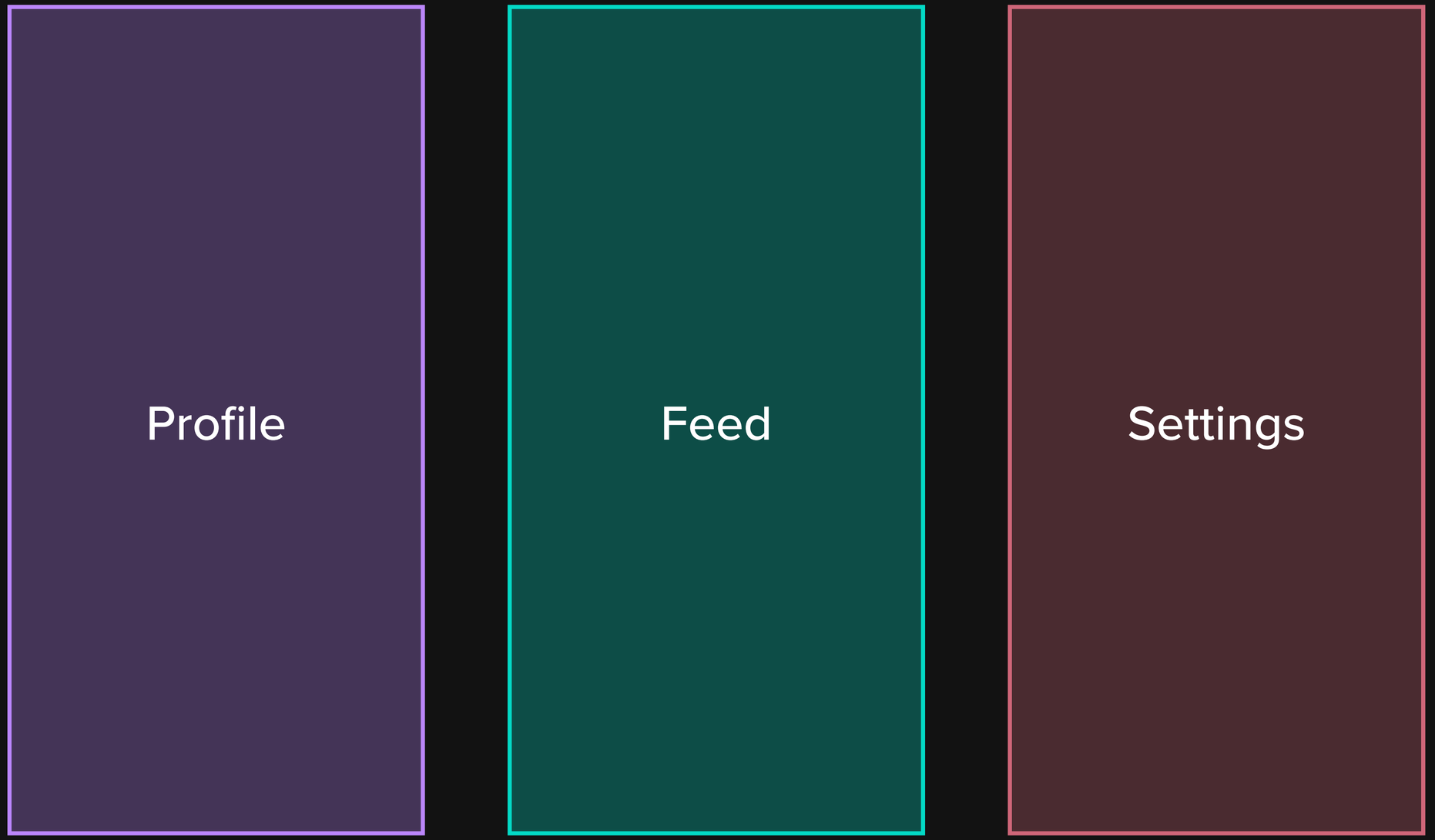
Shared Runtime Dependencies

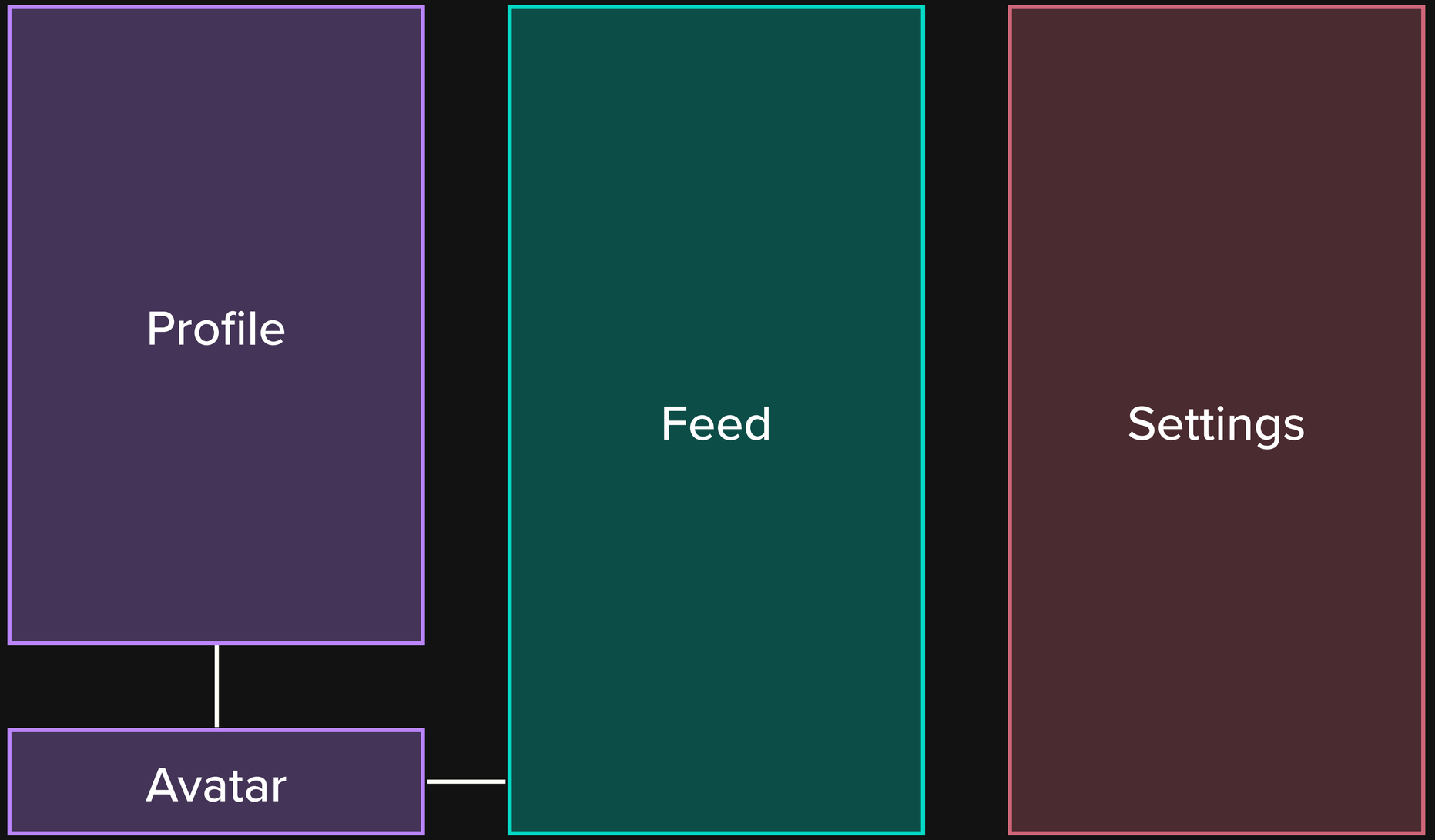
- How will you handle (major) versions?
- Shared dependencies require coordinated upgrades

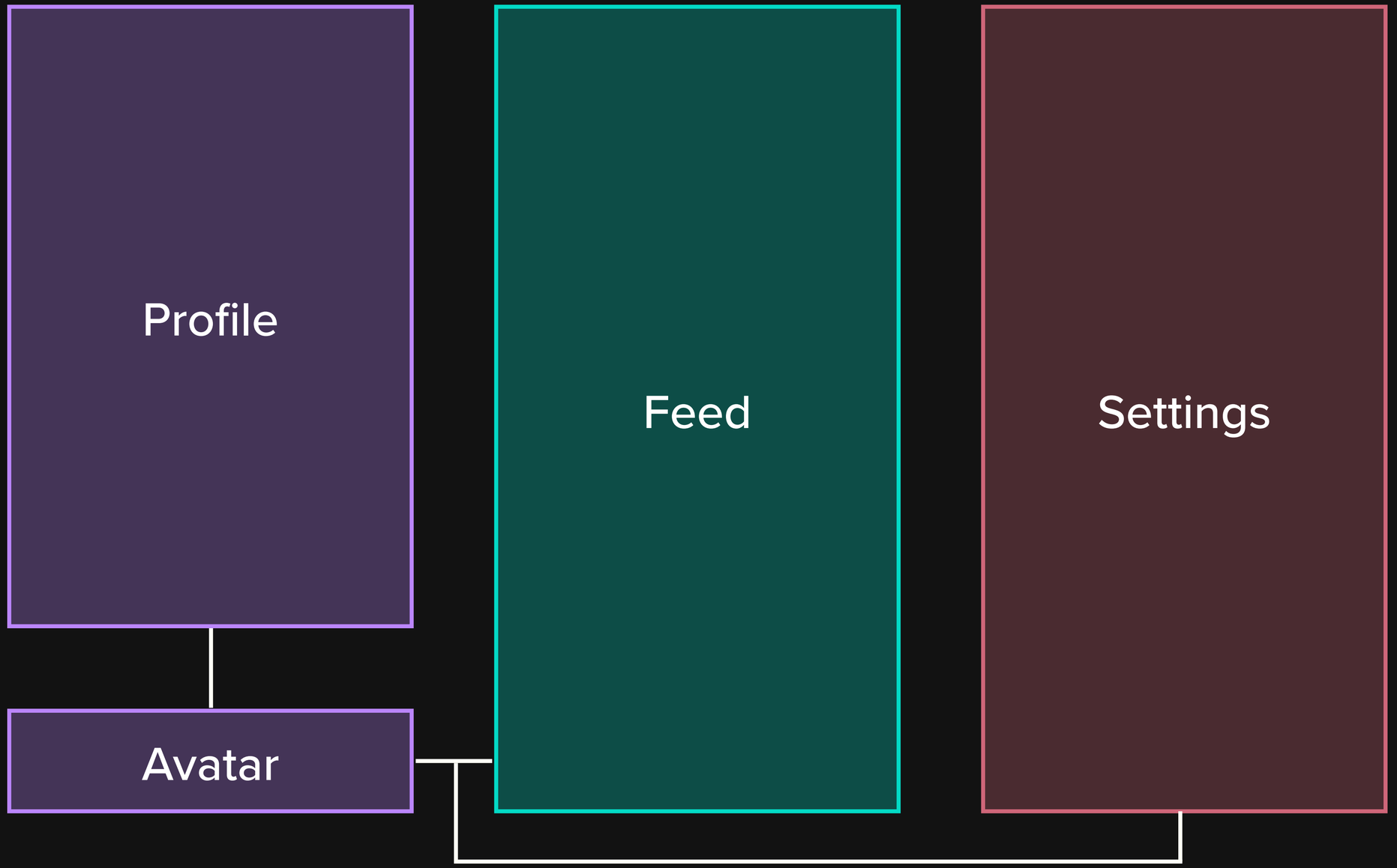
Isolated dependencies = Duplication
Shared dependencies = Coordination

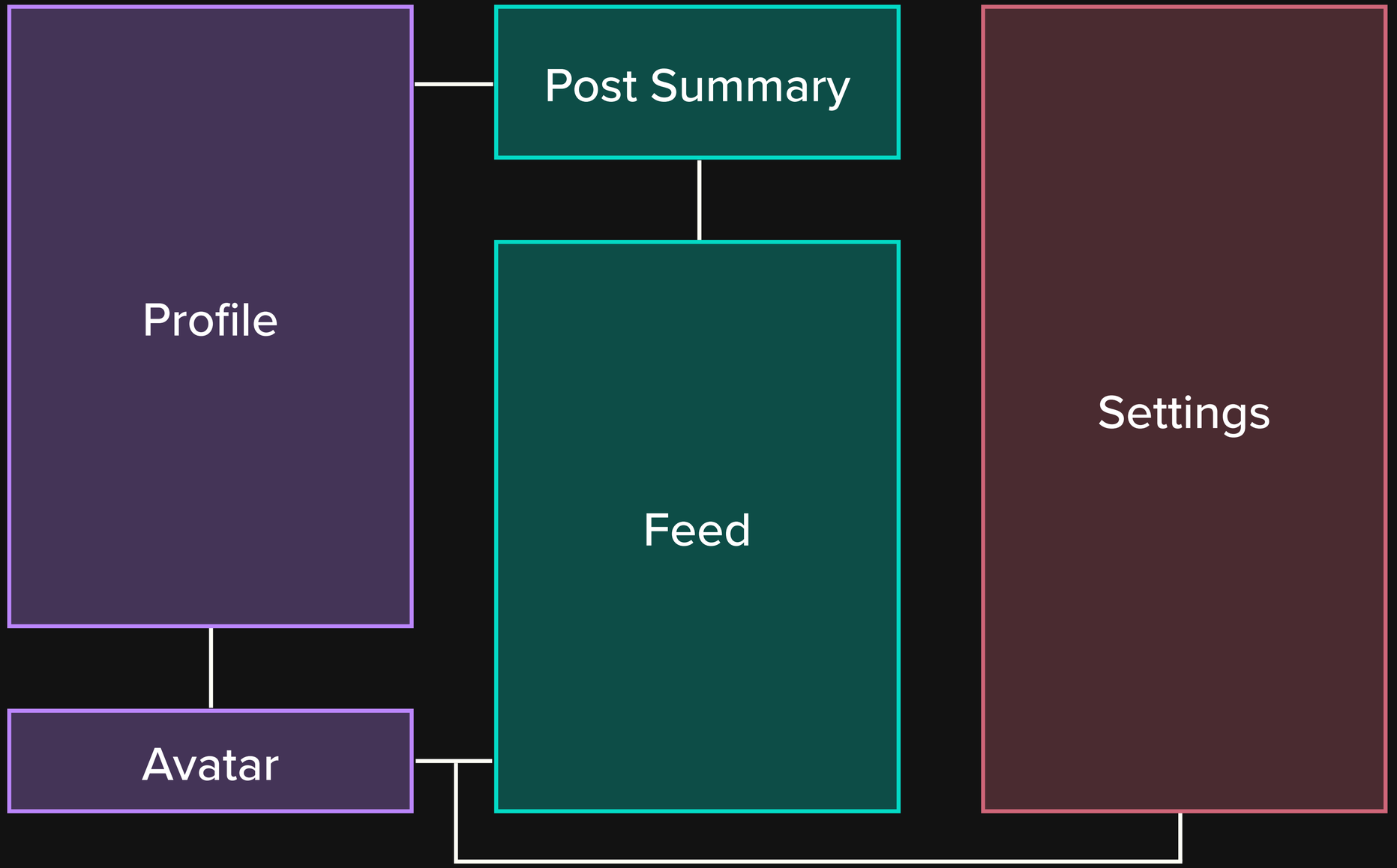
How do you **divide** up the app?

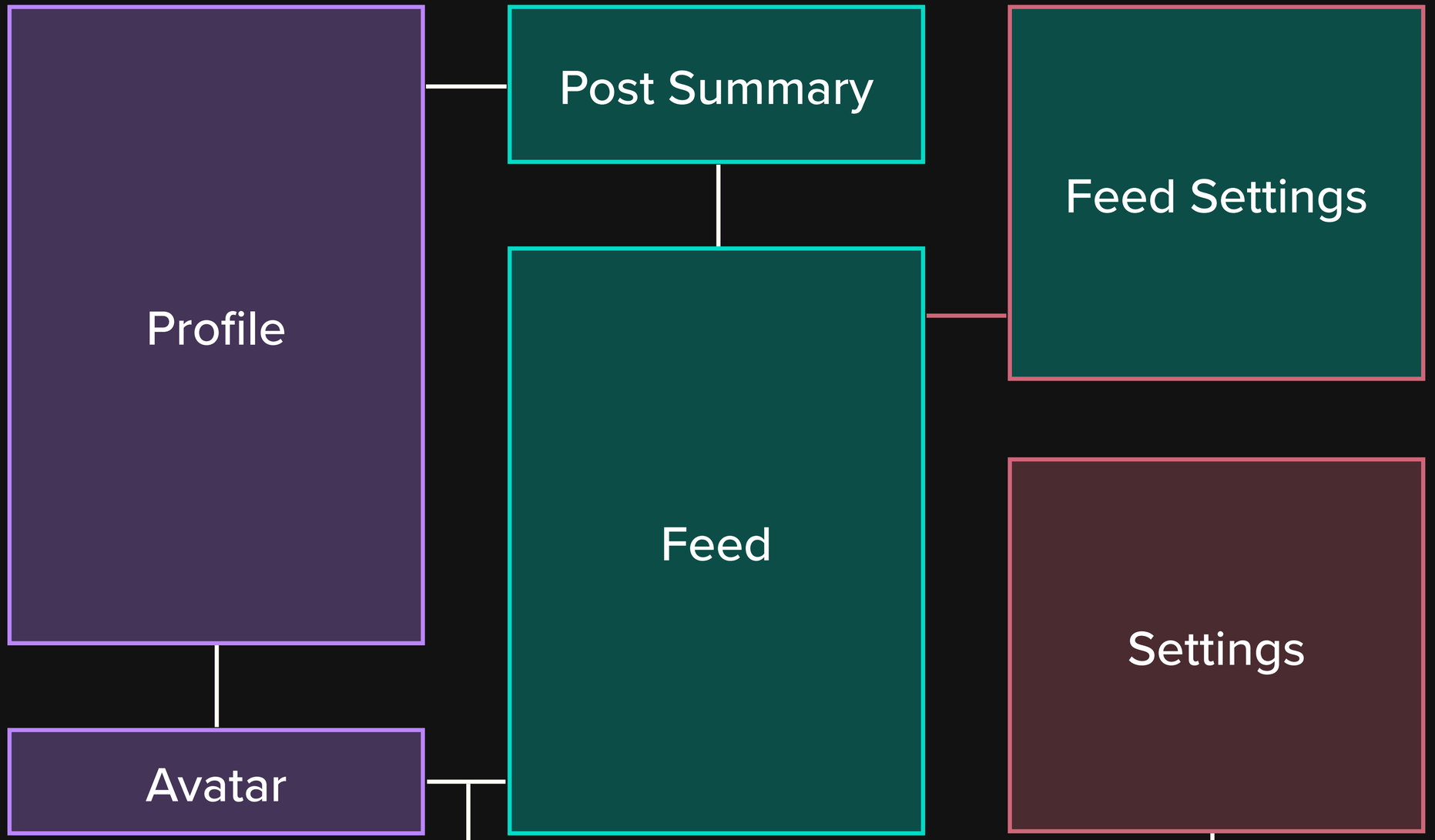
- By route, by feature, by screen section?
- How large/small are the divisions?
- Will the divisions survive organizational change?


Profile
Feed
Settings


Profile
Feed
Settings
Avatar


Profile
Feed
Settings
Avatar


Profile
Post Summary
Feed
Settings
Avatar


Profile
Post Summary
Feed Settings
Feed
Settings
Avatar

Deciding how to divide an app so that teams can operate independently while maintaining consistency is

a hard problem

No technical strategy/architecture will fix **organizational dysfunction**

Handling **troubleshooting**

- Which app is responsible for handling uncaught errors?
- How does the correct team get notified about errors?

Handling errors **consistently** and “**near**” where they occurred requires more than just error boundaries

Dealing with optimization

- Whose responsibility is it to monitor the performance of the composed app?
- Who owns optimizing the app when things get slow?

Separate builds and separate repos make it difficult to analyze performance characteristics **holistically**

Every **division** in an app can add complexity

Every **decision** in an app can add complexity

Every **decision** in an app can add complexity, be **intentional** and consider the **long-term impact**

So, do **micro-frontends**
live up to the hype?

They can **with** the right strategy

Testing (especially end-to-end)
Authentication & Authorization
Privacy
Availability & Resiliency
Analytics
Accessibility
And more...

We are still figuring out “**best practices**” for micro-frontends

My recommendation...

- Use Module Federation
- In a monorepo
- With as few divisions as possible
- With as little shared code as possible
- Supported by an infrastructure team

Keep **your goals** in mind when solving problems

Micro-Frontends promise to make building complex applications **easier** and your team **more efficient**

Micro-Frontends *can* make
building complex applications **easier**
and your team **more efficient**

Micro-Frontends *can* make
building complex applications **easier**
and your team **more efficient**
but **require thoughtful investment**