亚马逊云科技

Micro-Frontends anti-patterns

distributed architecture for the user interfaces

Luca Mezzalira

Principal Serverless Specialist Solutions Architect
亚马逊云科技

"There is no compression algorithm for experience"

Andy Jassy – CEO of Amazon

Luca Mezzalira

Principal Serverless Specialist at AWS

International Speaker

Author

Micro-Frontends A₵Ö ɟ Ö *technical representation of a business subdomain* ɟ Ö⁷ Aﬃ̄ ʸ *independent implementations* ʸ Pi ɟ Ö *same or different technology*

~ṔĆAﬀ ɟ Ö⁷ Ņi Ī ꭗﬀĐ *minimize the code shared* ʸ Pi Ī ɟ Ö₵

ŅꭗÓĐĪ ÛAṔĆŅAĆĐ ɟ Ö⁷ A₵Ö *own by a single team*

Micro-Frontends AÇÖ i Ö technical representation of a business subdomain i Ö7 Affy independent implementations y Pi i Ö same or different technology

~RÉAff i Ö7 Ni Ī ÐF minimize the code shared y Pi Ī i ÖÇ NÓÐĪ ÛARÉN AĆÐ i Ö7 AÇÖ own by a single team

Micro-Frontends A₵Ö ⫞ Ö technical representation of a business subdomain ⫞ Ö7 Aﬃ y ***independent implementations*** y Pi ⫞ Ö same or different technology

~ṔĆAﬀ ⫞ Ö7 Ņi Ī ⅍Đ minimize the code shared y Pi Ī ⫞ Ö₵ Ņ⅍ÓĐĪ ÛAṔĆŅ AĆĐ ⫞ Ö7 A₵Ö own by a single team

Micro-Frontends A₵Ö ⫞ Ö technical representation of a business subdomain ⫞ Ö⁷ Aﬃ̄ ʸ independent implementations ʸ Pi ⫞ Ö **same or different technology**

~ṔĆA ff ⫞ Ö⁷ Ņi Ī ꭗfĐ minimize the code shared ʸ Pi Ī ⫞ Ö₵ ŅꭗÓĐĪ ÛAṔĆŅA ĆĐ ⫞ Ö⁷ A₵Ö own by a single team

Micro-Frontends are a technical representation of a business subdomain with dedicated independent implementations using the same or different technology

~Ideally there should minimize the code shared by the components owned by a single team

Micro-Frontends [illegible] technical representation of a business subdomain [illegible] independent implementations [illegible] same or different technology

[illegible] minimize the code shared [illegible] own by a single team

Micro-Frontends benefits

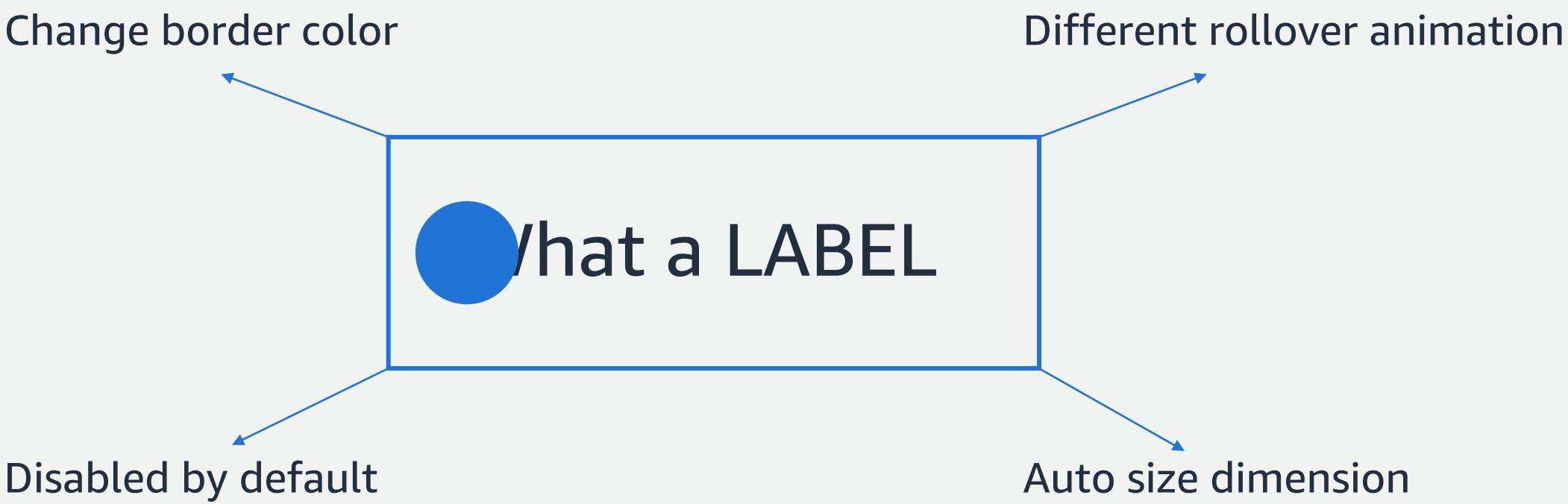
o

X

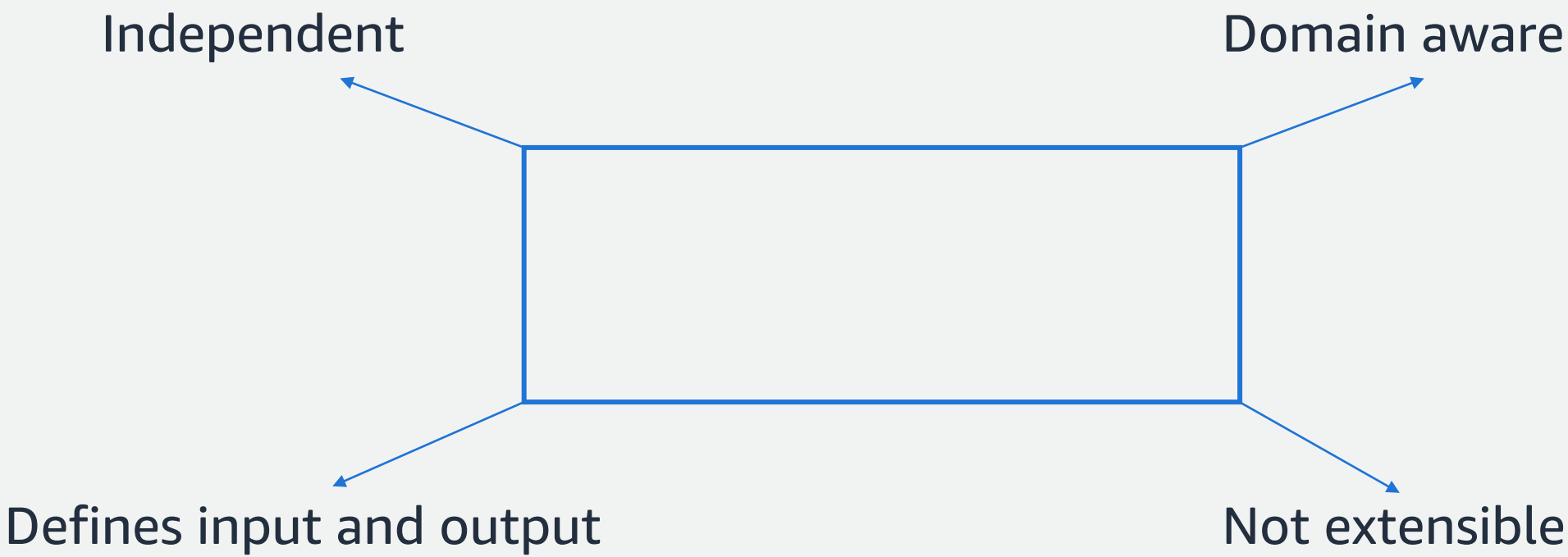
“*Yin and Yang*”

(Micro-Frontends and Components)

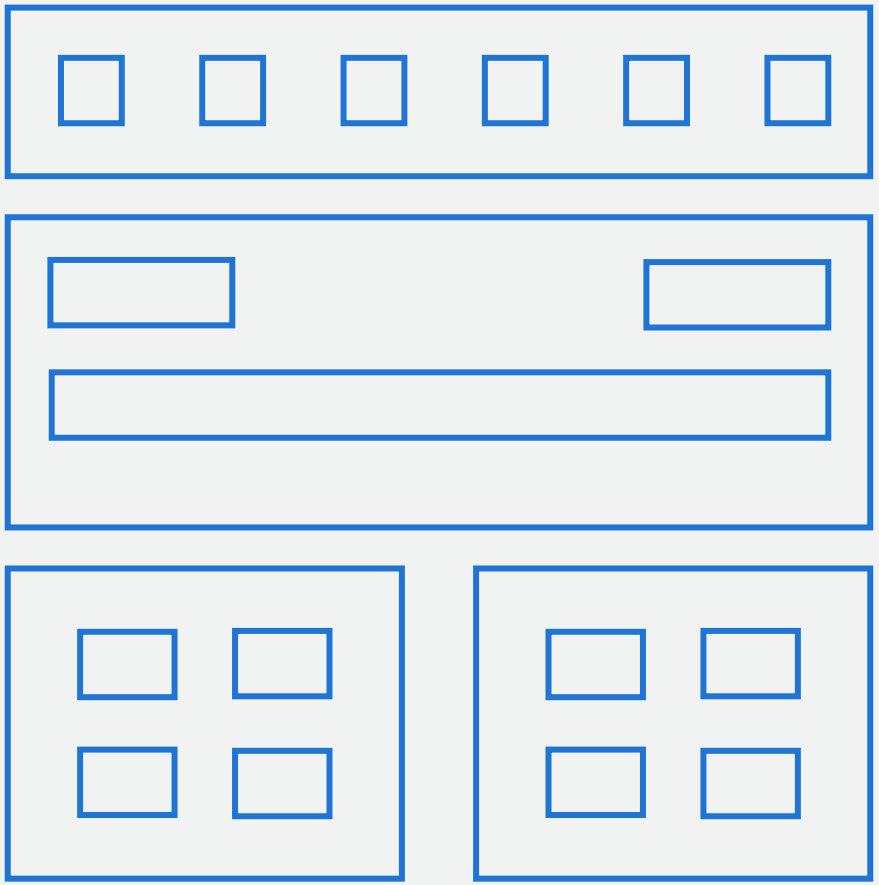
A component



A micro-frontend



Too many… components?

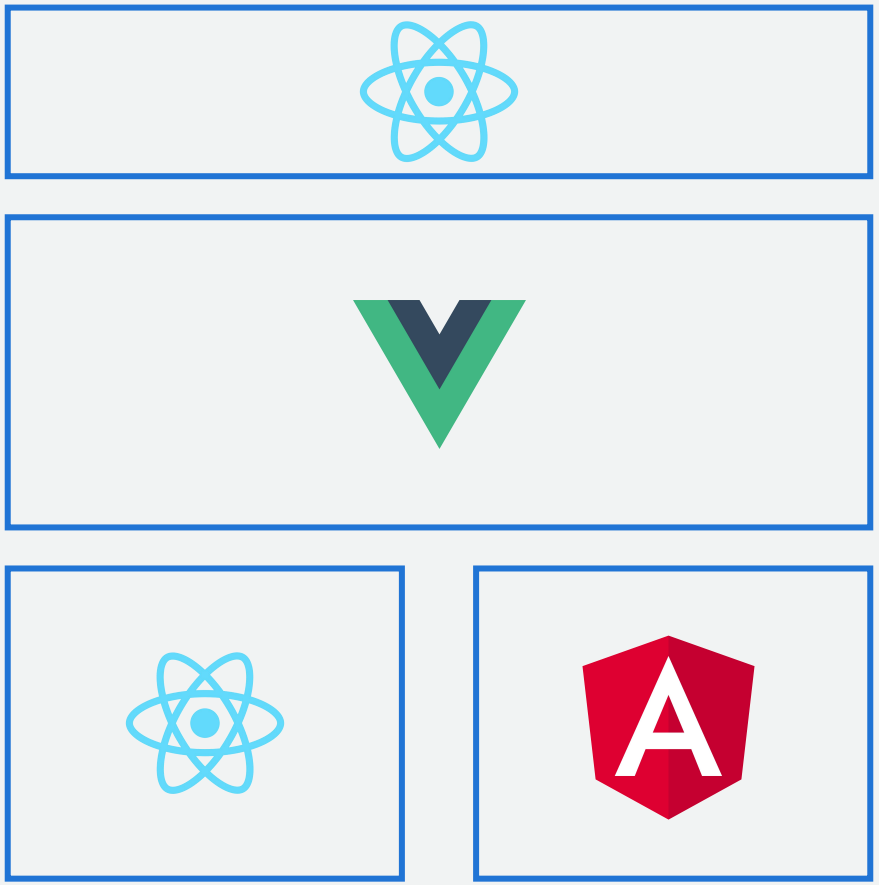


Are you designing a micro-frontend or a component?

"Hydra of Lerna"

(Multi frameworks approach)

Frameworks, frameworks everywhere!



How many
UI libraries or frameworks
would you use in a SPA?

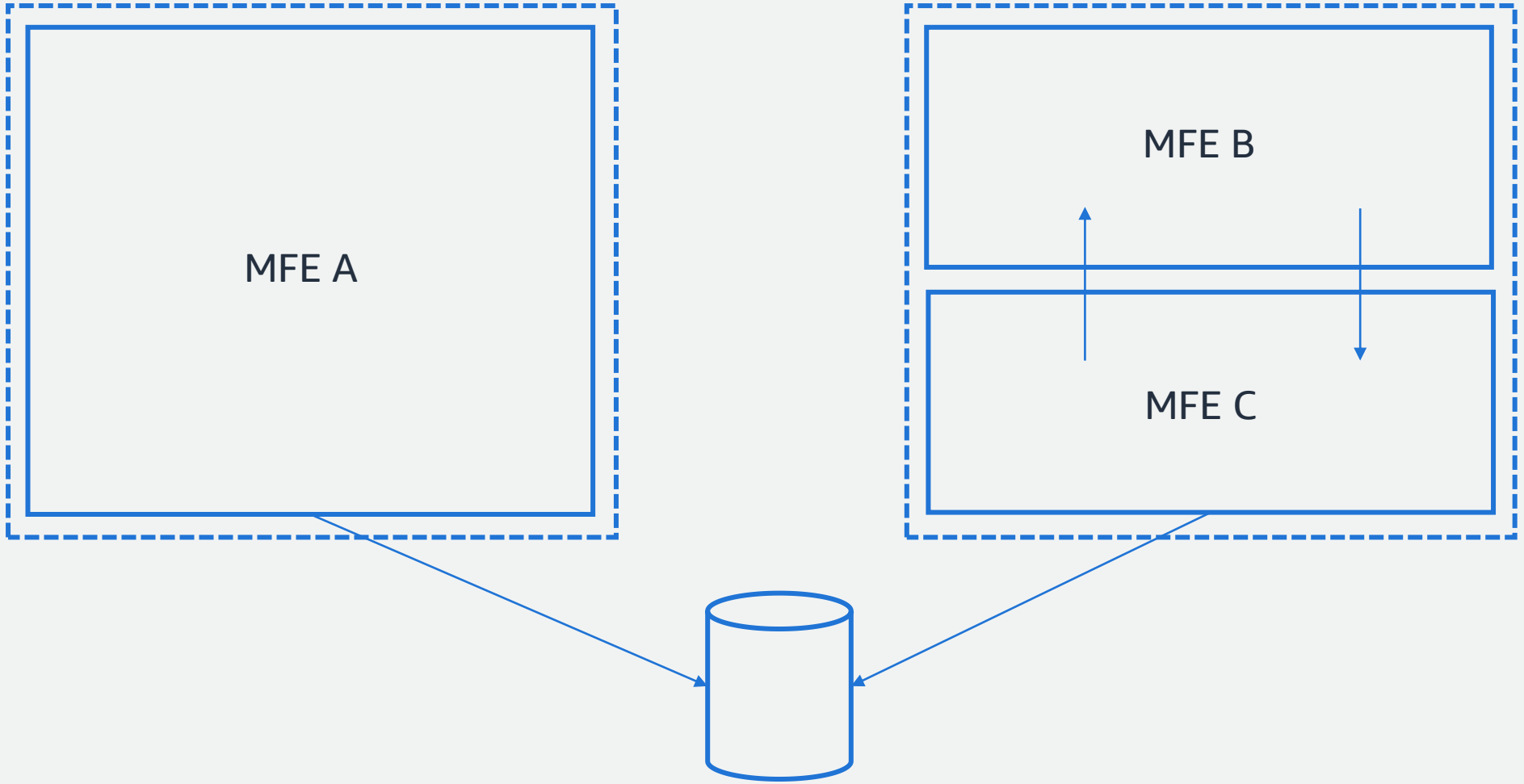
Multi-framework approach

Use multi-frameworks when appropriate, don’t optimize your architecture for them

"The swiss army knife"

(Write programs that do one thing and do it well)

The greenfield project...



The legacy editor

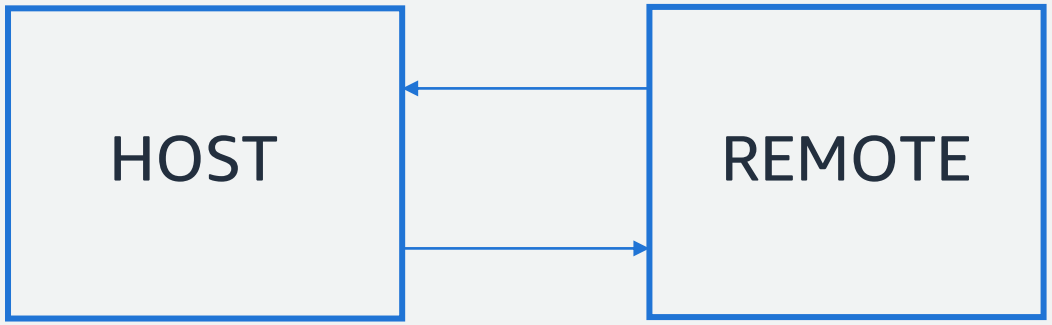
- [illegible]
- [illegible]
- [illegible]

Spare the application shell codebase, use an anti-corruption layer for the legacy system

"A return ticket, please"

(Unidirectional data flow at the rescue)

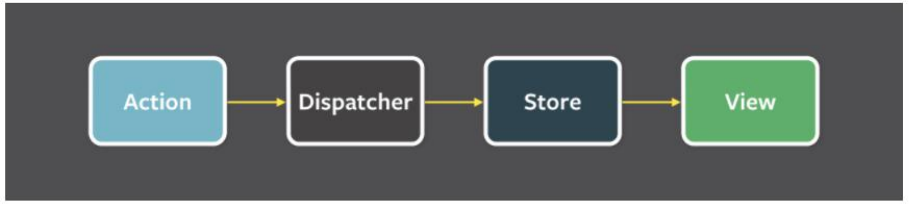
Sharing



Unidirectional data flow at the rescue

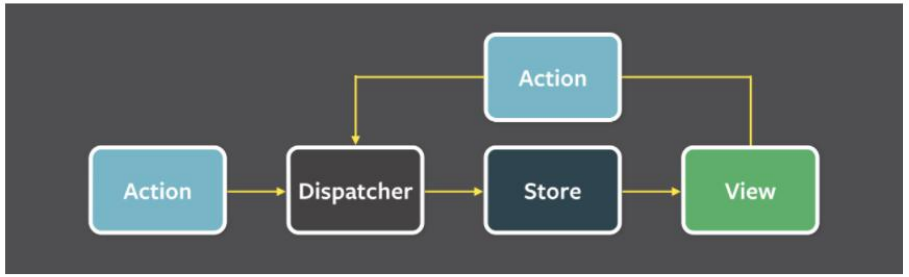
Structure and Data Flow

Data in a Flux application flows in a single direction:



A unidirectional data flow is central to the Flux pattern, and the above diagram should be **the primary mental model for the Flux programmer**. The dispatcher, stores and views are independent nodes with distinct inputs and outputs. The actions are simple objects containing the new data and an identifying *type* property.

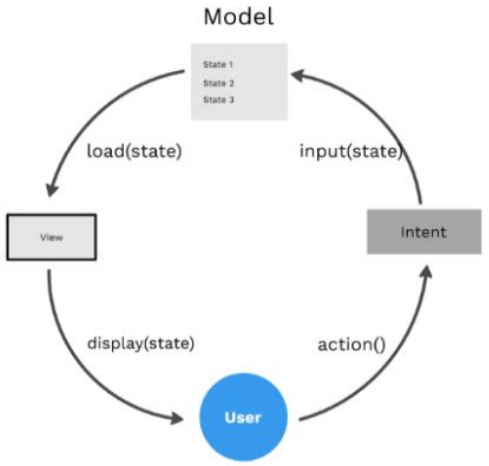
The views may cause a new action to be propagated through the system in response to user interactions:



All data flows through the dispatcher as a central hub. Actions are provided to the dispatcher in an *action creator* method, and most often originate from user interactions with the views. The dispatcher then invokes the callbacks that the stores have registered with it, dispatching actions to all stores. Within their registered callbacks, stores respond to whichever actions are relevant to the state they maintain. The stores then emit a *change* event to alert the controller-views that a change to the data layer has occurred. Controller-views listen for these events and retrieve data from the stores in an event handler. The controller-views call their own `setState()` method, causing a re-rendering of themselves and all of their descendants in the component tree.

How does the MVI work?

User does an action which will be an Intent → Intent is a state which is an input to model → Model stores state and send the requested state to the View → View Loads the state from Model → Displays to the user. If we observe, the data will always flow from the user and end with the user through intent. It cannot be the other way, Hence its called Unidirectional architecture. If the user does one more action the same cycle is repeated, hence it is Cyclic.



MVI cyclic flow representation

Unidirectional data flow learnings

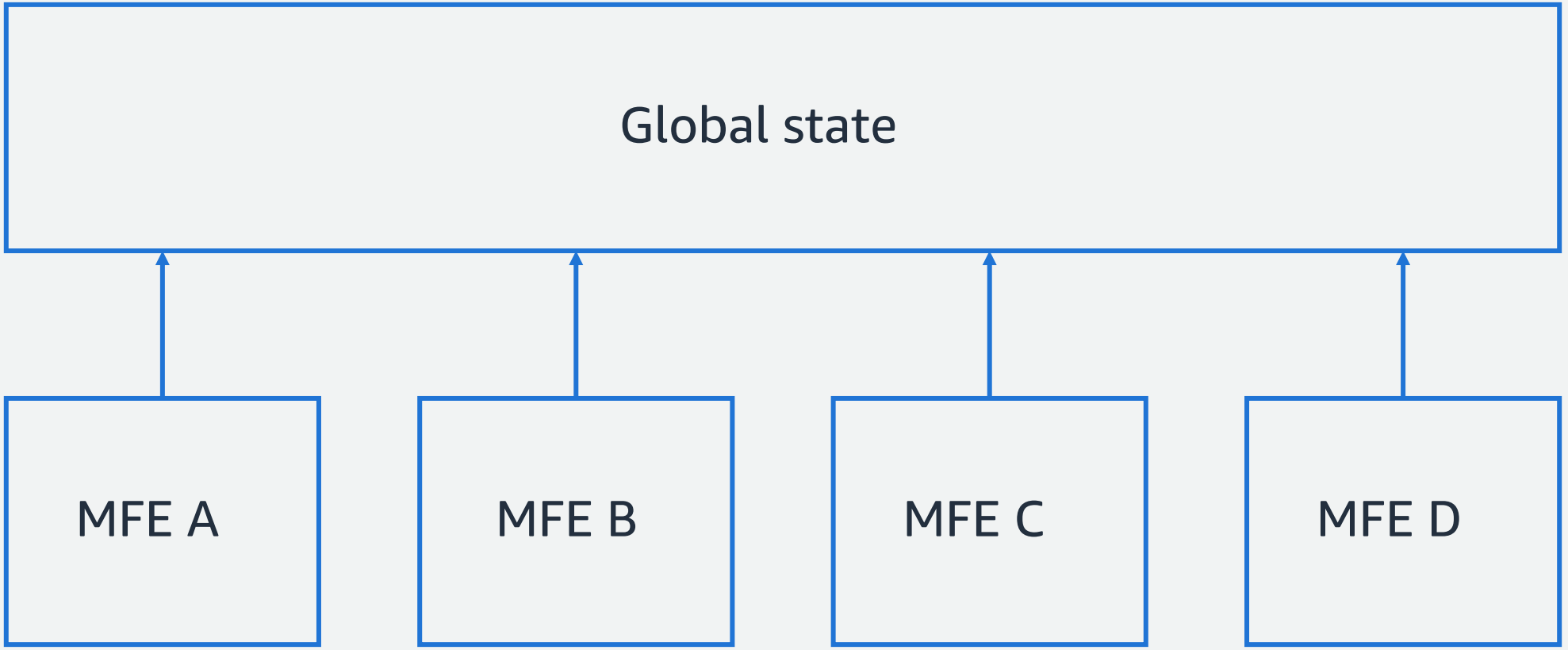
6

Avoid bi-directional sharing unless strictly needed

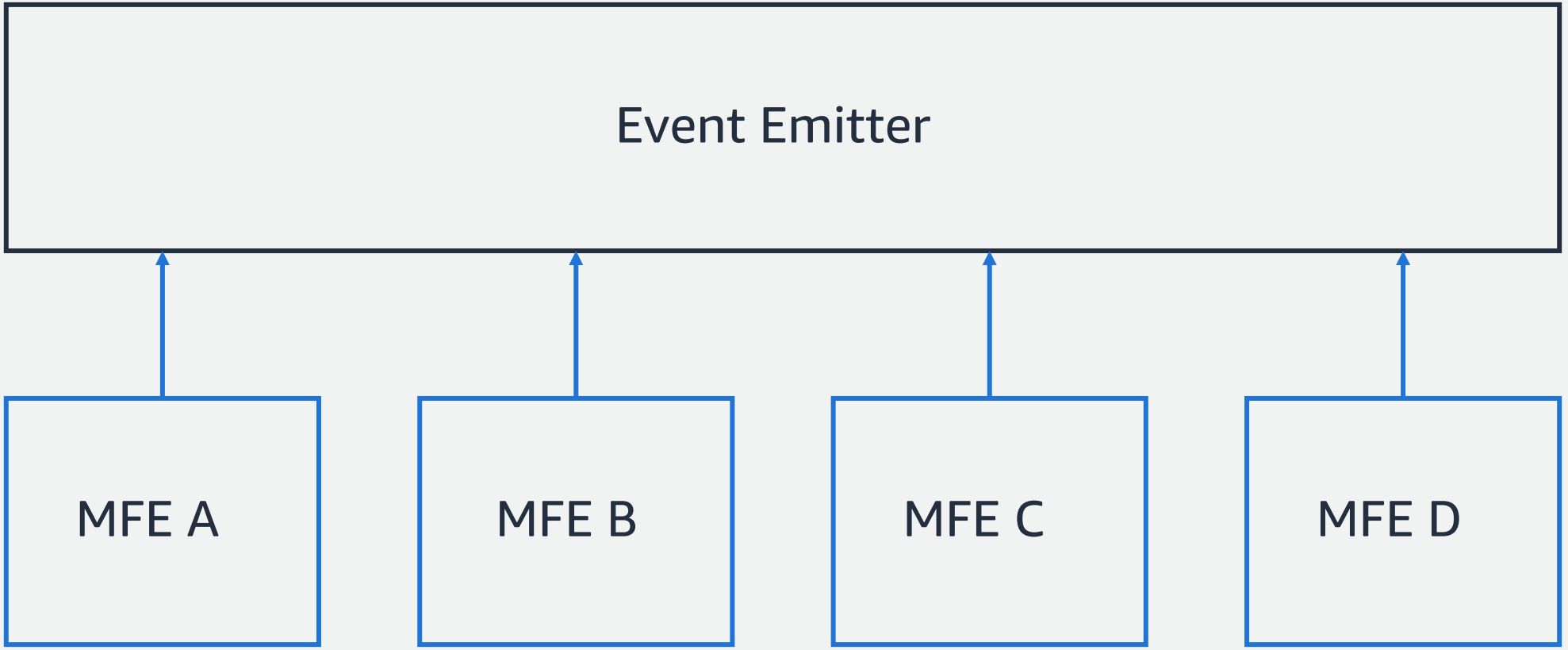
"Relax, it's just code"

(Avoid organizational coupling)

Design-time coupling



Loosely coupled entities

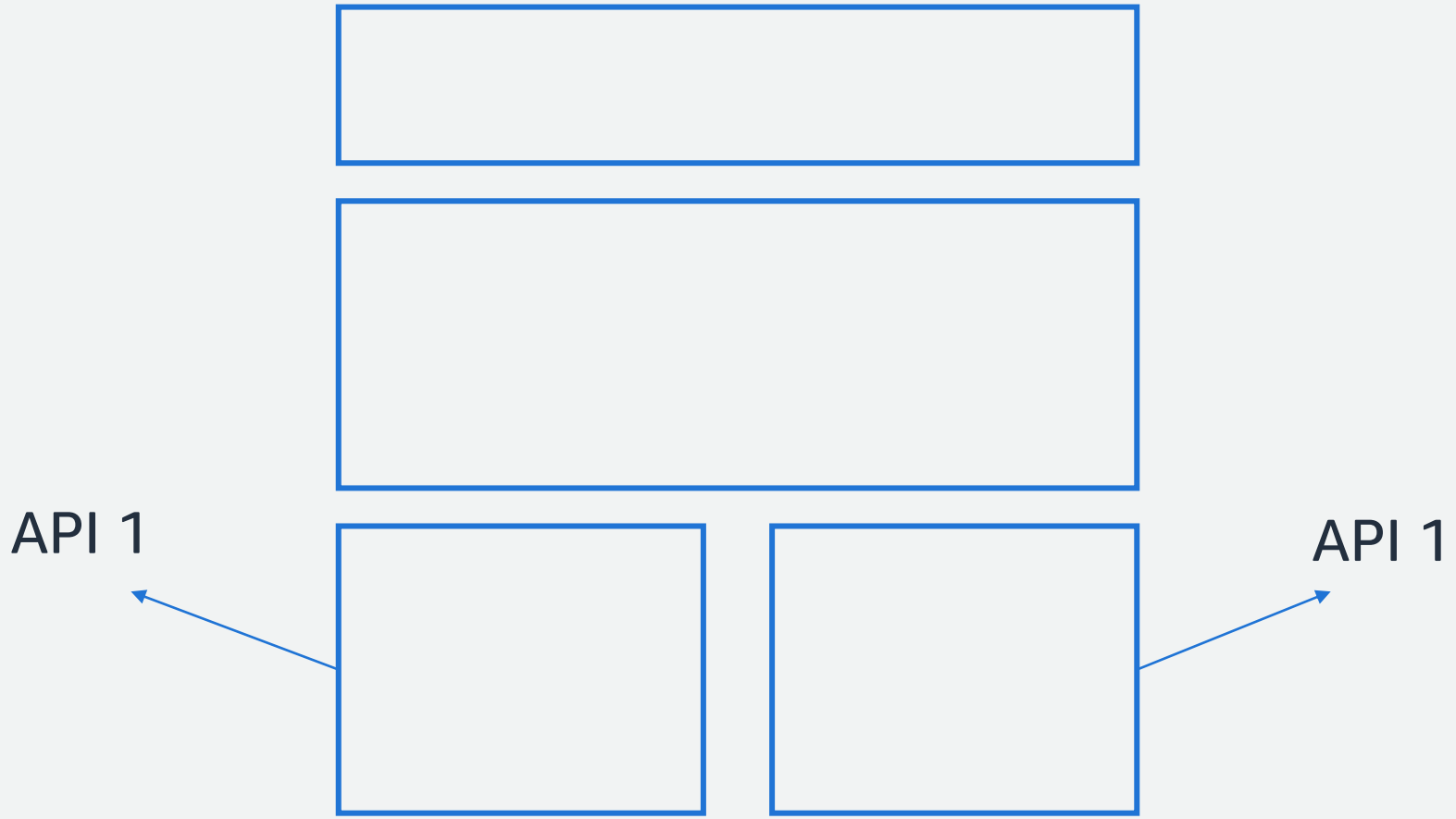


Embrace loosely coupled but highly aligned MFEs

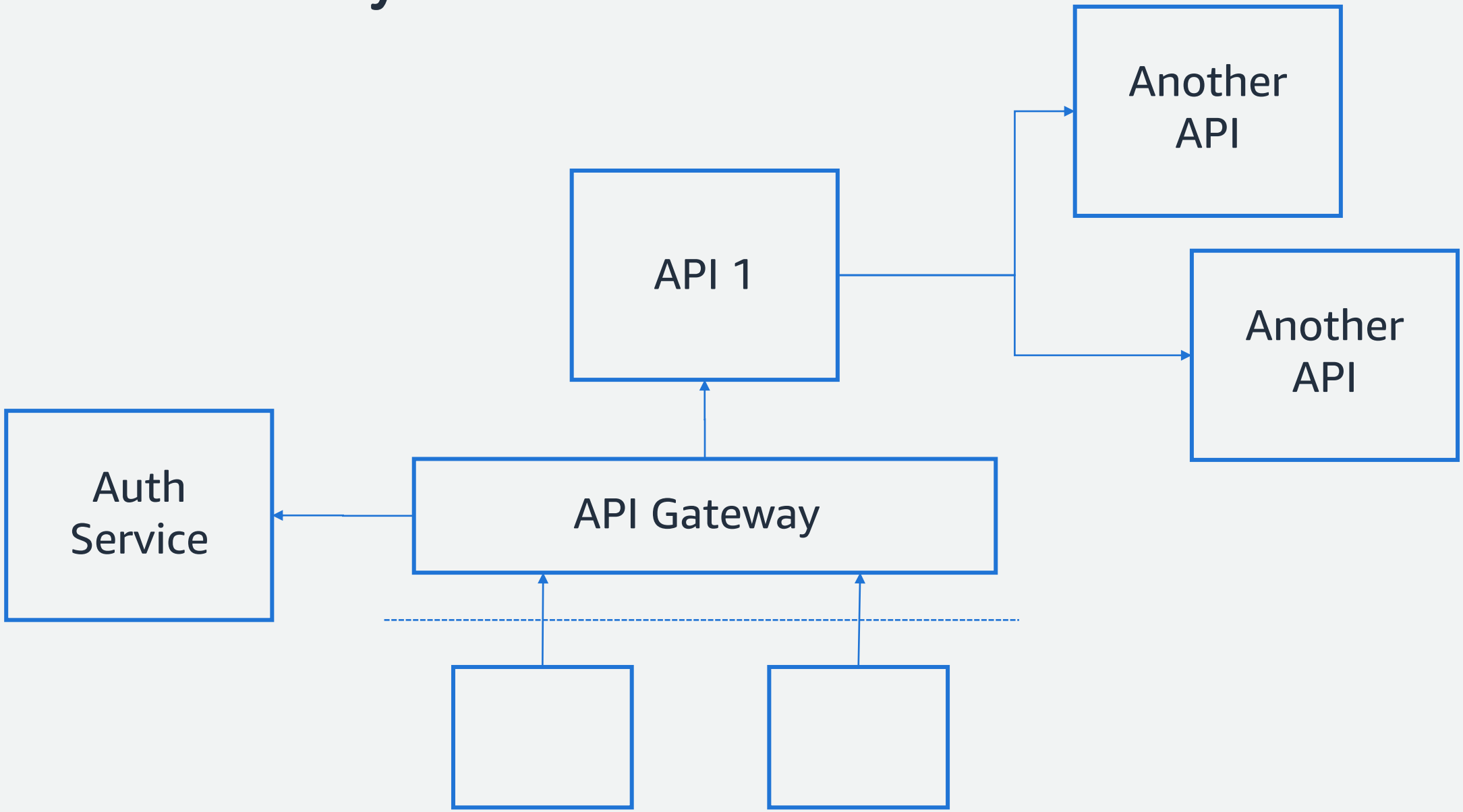
"Let's hammer the APIs"

(Multiple MFEs calling the same endpoint)

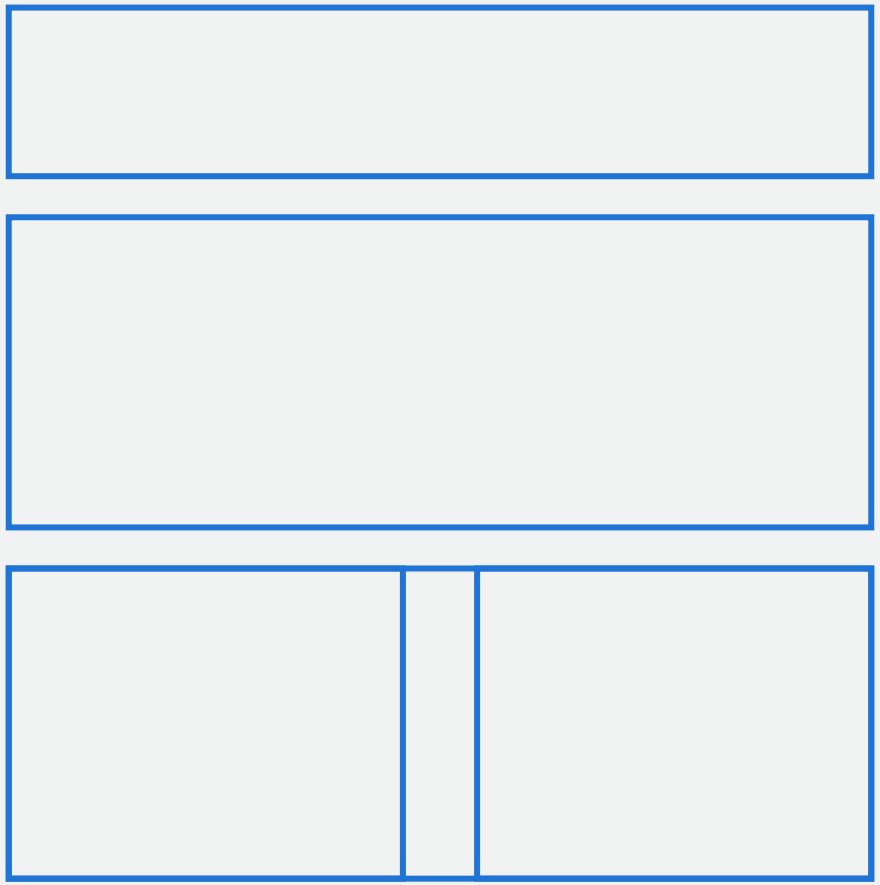
Multiple MFEs in the same view



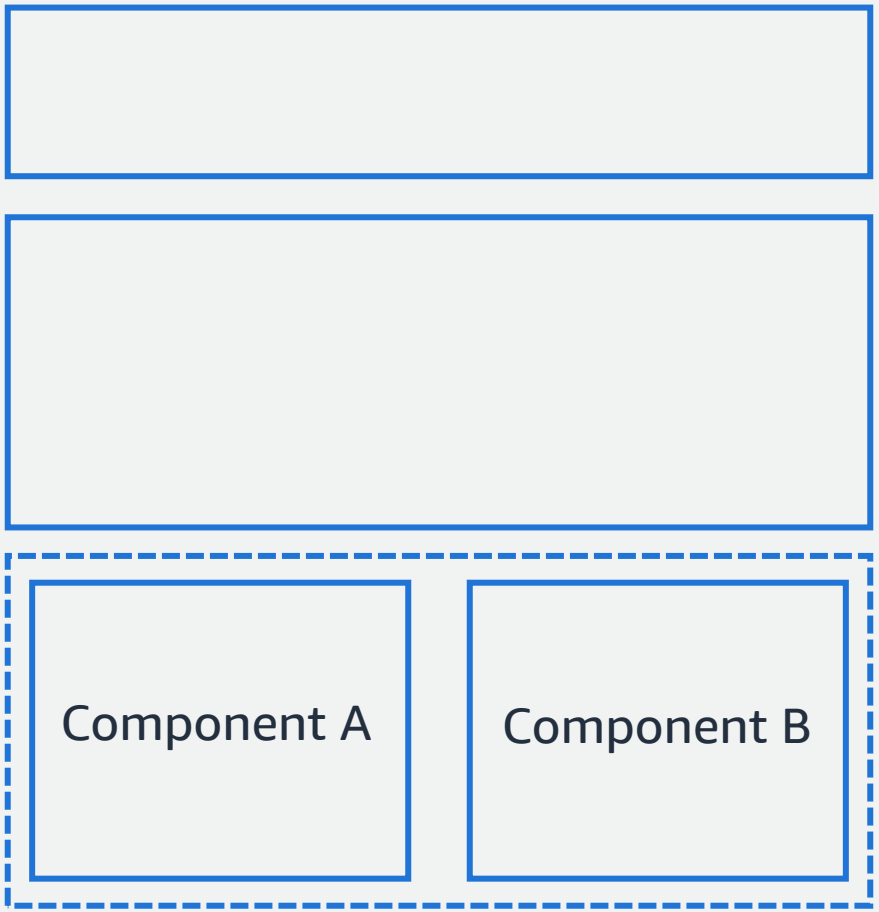
Your distributed system



Possible solutions



Possible solutions

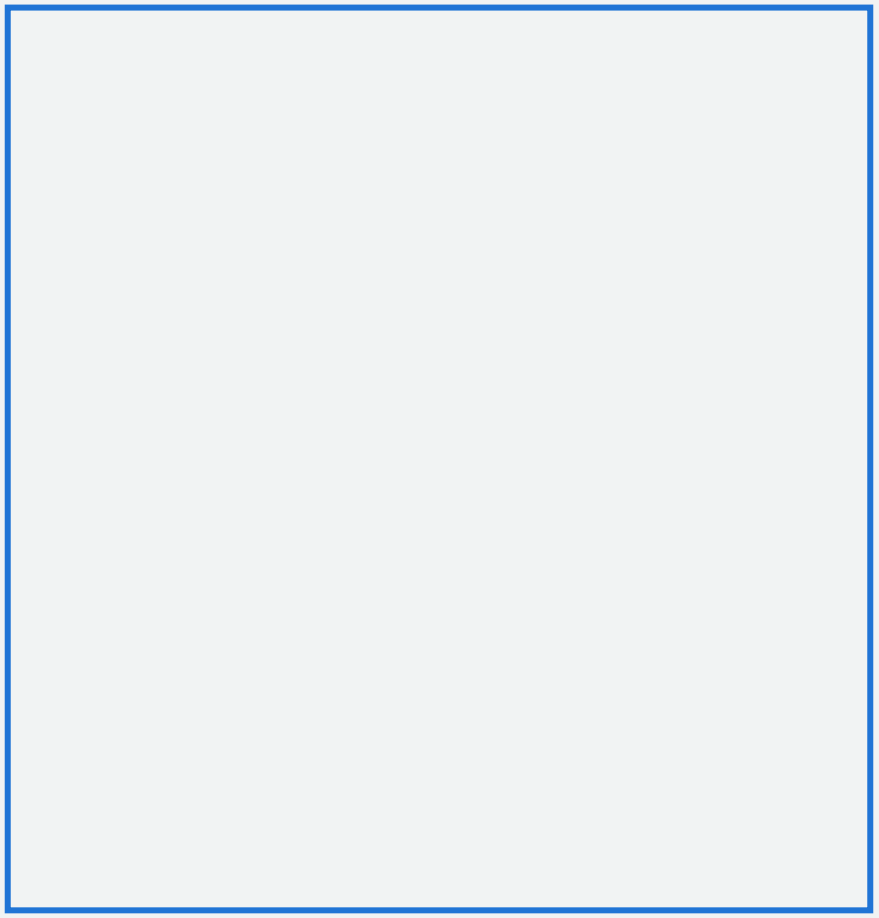


Understand the end-to-end impact of your decisions

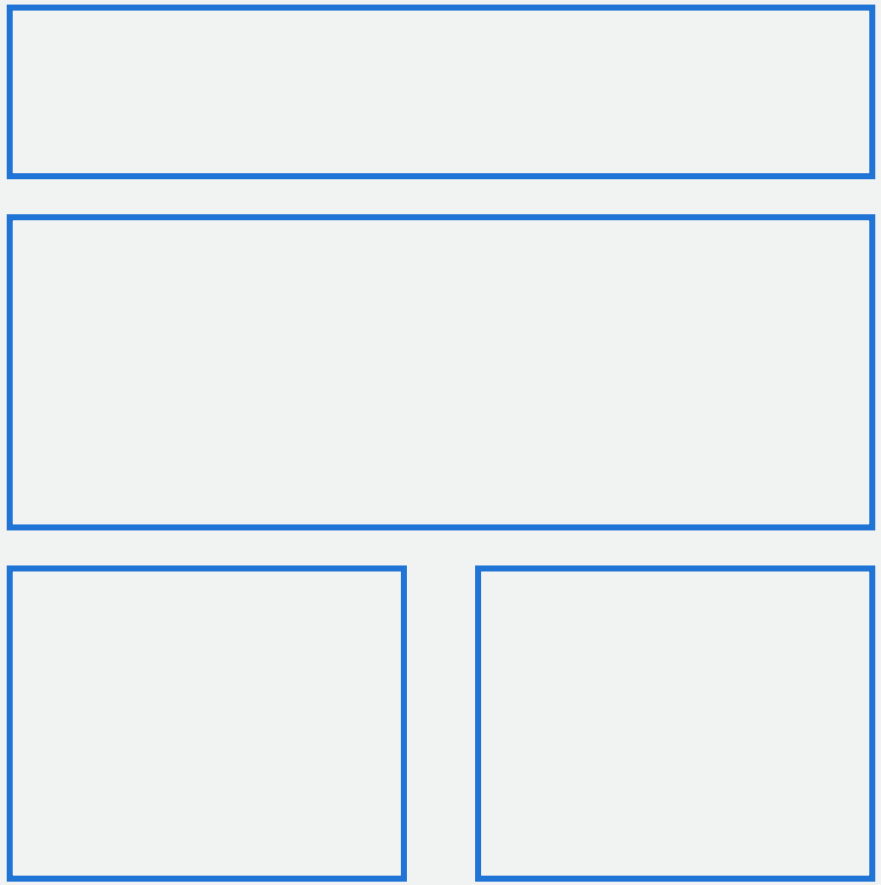
"Bye bye big-bang"

(Iterative deployment)

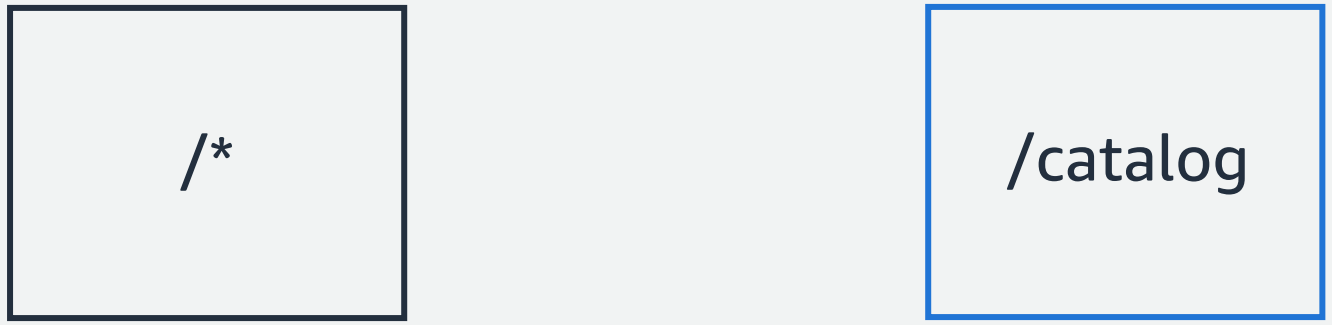
Migration strategies



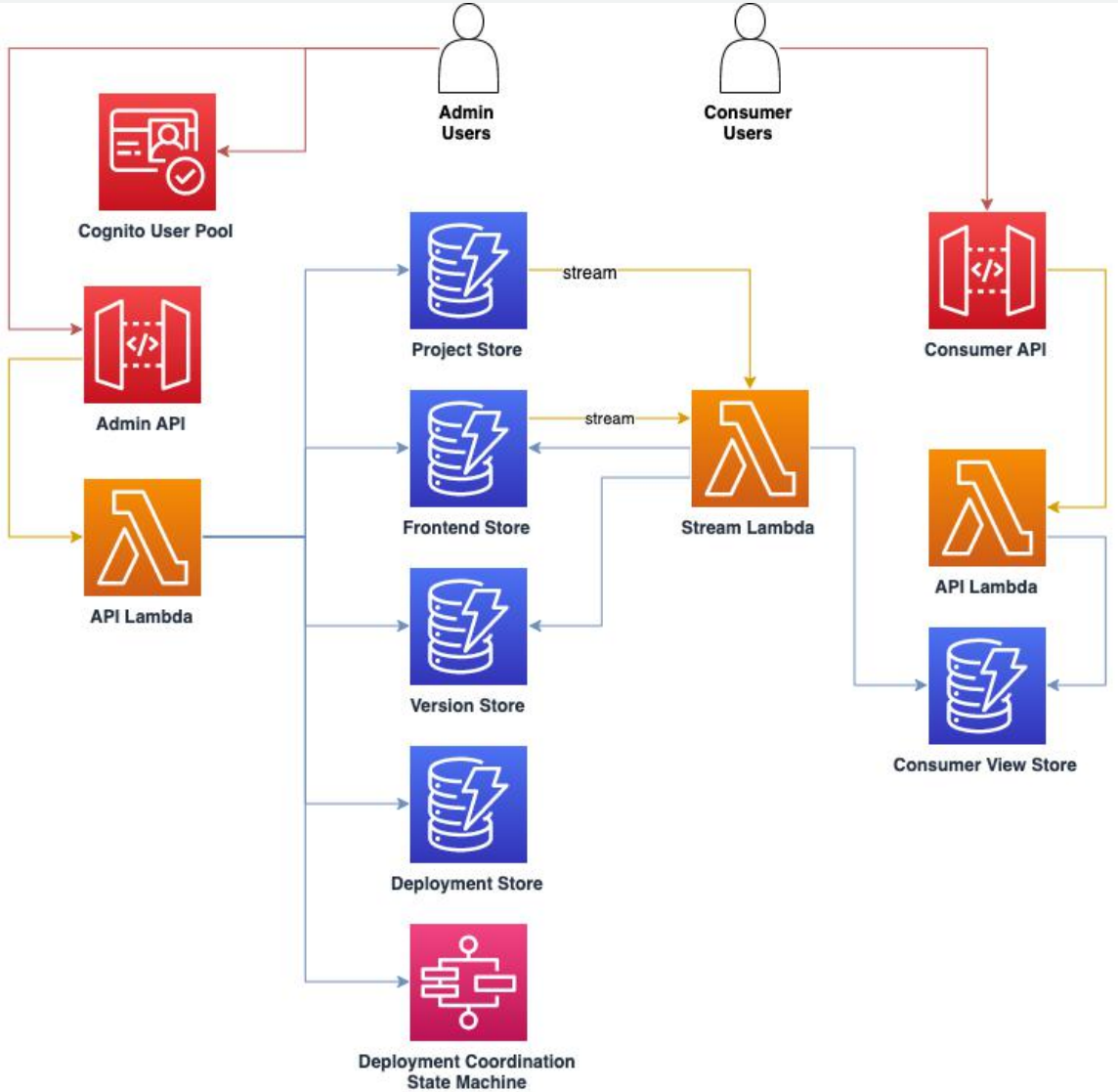
Migration strategies



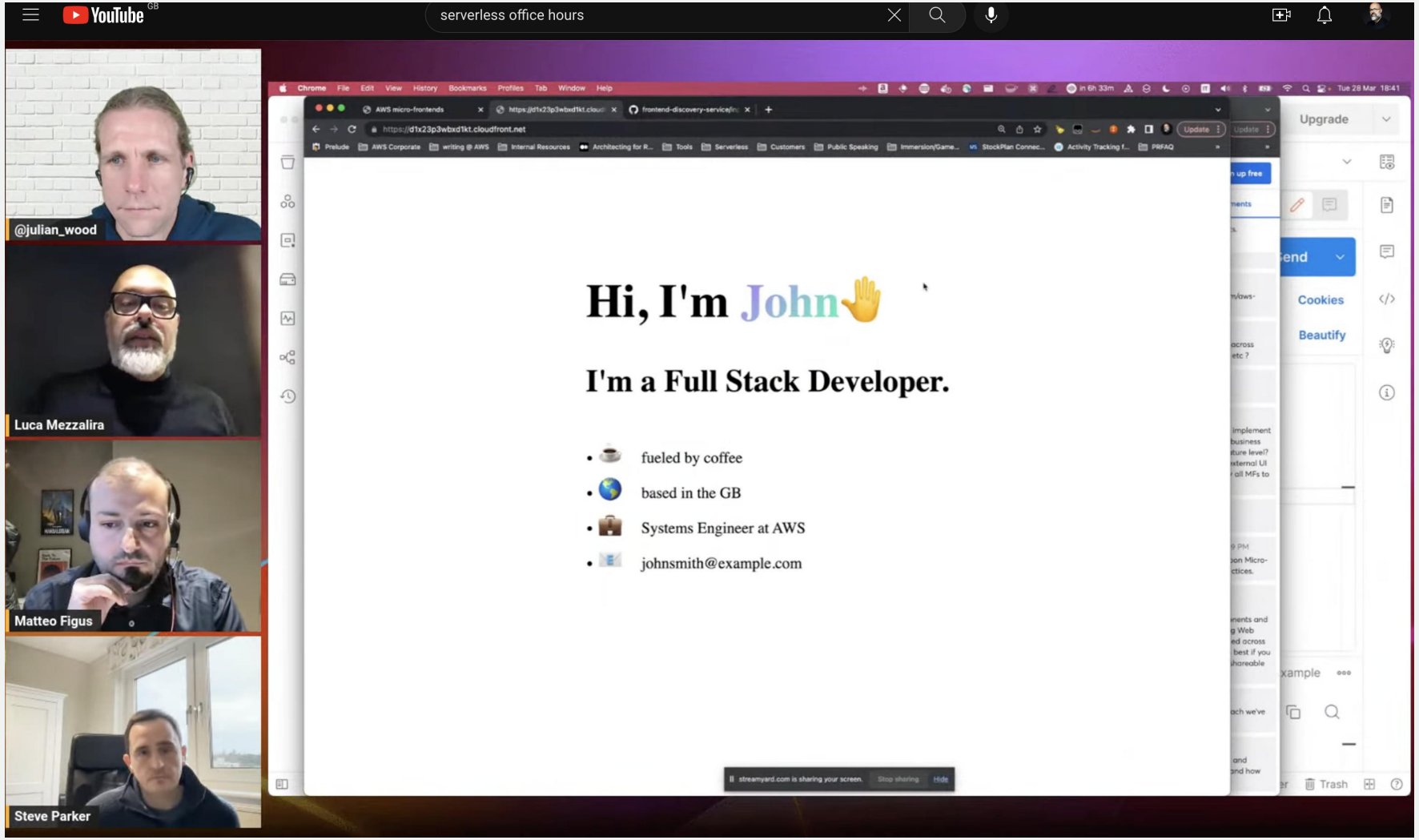
Migration strategies



Migration strategies



https://bit.ly/3G0xnNR



https://www.youtube.com/watch?v=Vm9jlRSlKVQ

Iteratively deploy micro-frontends helps increasing the developers confidence as well as adding value for users

Architecture is always a trade-off, *just find a balanced approach for your context*

谢谢

Luca Mezzalira

lmezza@amazon.com

亚马逊云科技