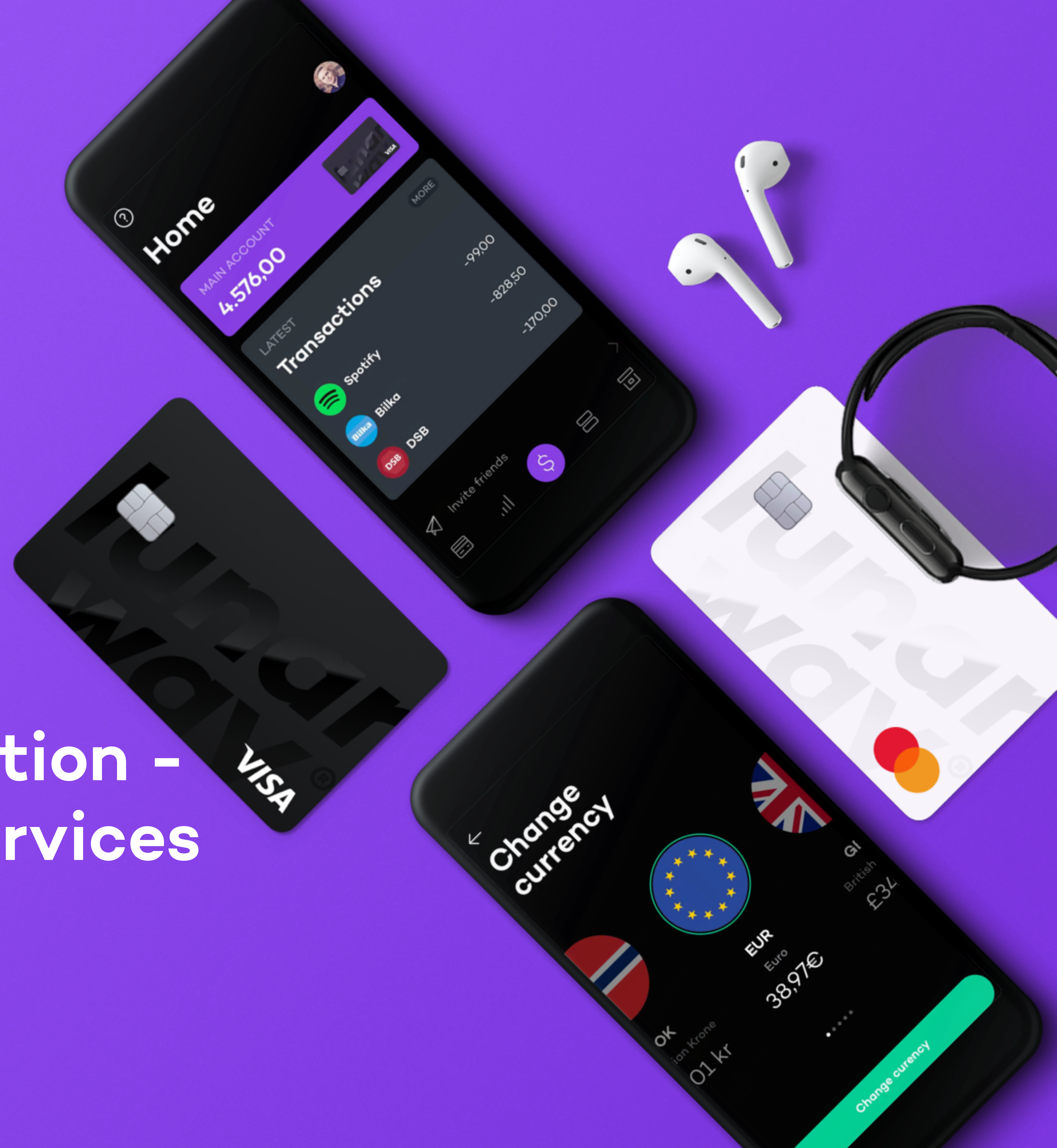
lunar way®



Cloud Native Transformation - From Monolith to Microservices

GOTO Nights CPH August 2018

Thomas Bøgh Fangel

Who?

Thomas Bøgh Fangel (@tbfangel)

- Web Architect @lunarway since June 2016
- M.Sc. in Maths
- Professional software developer since 2004
- Previously: J2EE, JSE, Scala, Akka, Spring Boot
- Now primarily working in Go and Typescript

AGENDA

- Cloud Native Software Architecture
- The Scene
- Transforming the Monolith
- The Road Ahead

Cloud Native Software Architecture

https://pixabay.com/en/skyline-new-york-600001/

Cloud Native, the CNCF definition

*Cloud native technologies empower organizations to build and run **scalable applications** in modern, **dynamic environments** such as public, private, and hybrid clouds. **Containers, service meshes, microservices, immutable infrastructure,** and **declarative APIs** exemplify this approach.*

*These techniques enable **loosely coupled** systems that are **resilient, manageable,** and **observable**. Combined with **robust automation**, they allow engineers to make **high-impact changes frequently** and **predictably** with **minimal toil**.*

*The Cloud Native Computing Foundation seeks to drive adoption of this **paradigm** by fostering and sustaining an ecosystem of **open source, vendor-neutral** projects. We democratize state-of-the-art patterns to make these innovations accessible for everyone.*

CLOUD NATIVE
COMPUTING FOUNDATION

https://twitter.com/cra/status/1006228498892181504

Key characteristics

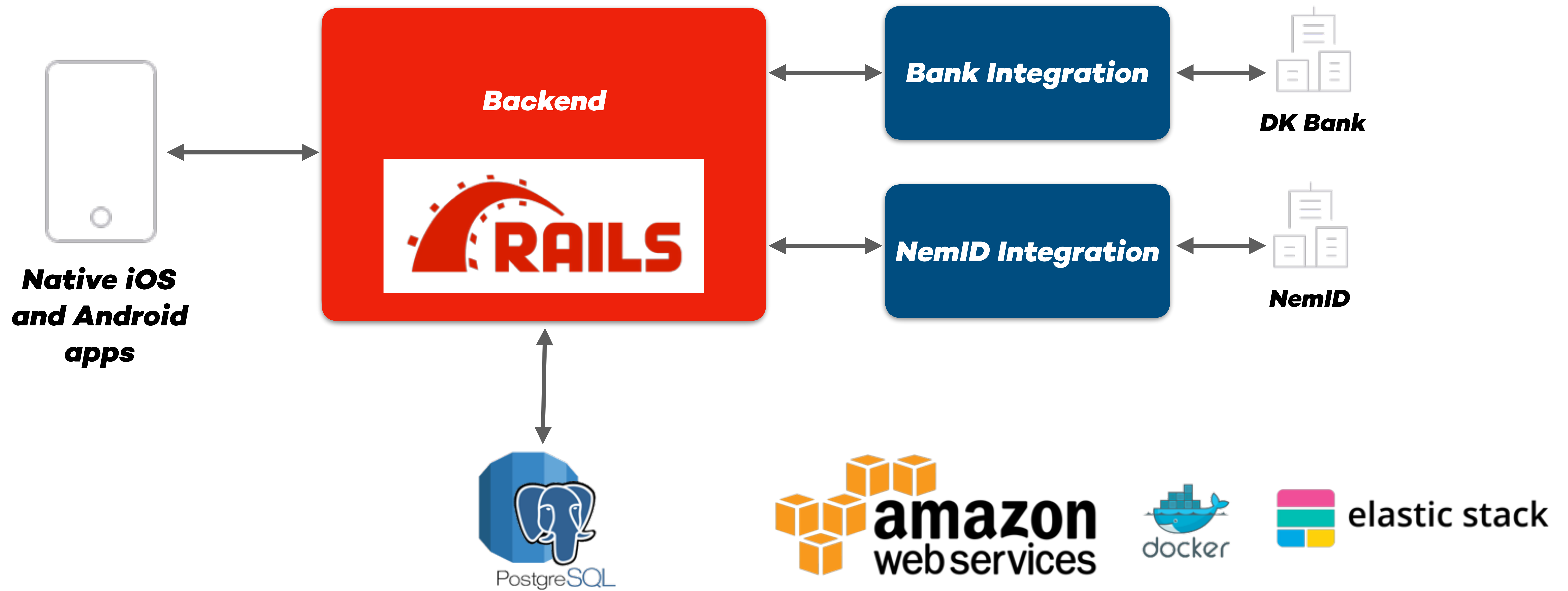
- Microservices
- Decoupling
- Resilience
- Frequent and predictable deployments
- Automation
- Observability and manageability

But why?

- Development speed
- Autonomous teams
- Scalability
- Fault tolerance

The Scene

Lunar Way's Platform of 2016


Backend
RAILS
Native iOS and Android apps
Bank Integration
DK Bank
NemID Integration
NemID
PostgreSQL
amazon web services
docker
elastic stack

Cloud Native Evaluation

- Microservices
- Decoupling
- Resilience
- Frequent and predictable deployments
- Automation
- Observability and manageability

https://pixabay.com/en/architecture-sky-building-3173357/

So, what is a microservice?

Microservices are small, autonomous services that work together.

– **Sam Newman**

…a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms…

…services are built around business capabilities and independently deployable by fully automated deployment machinery

– **Martin Fowler**

Why microservices – or why not?

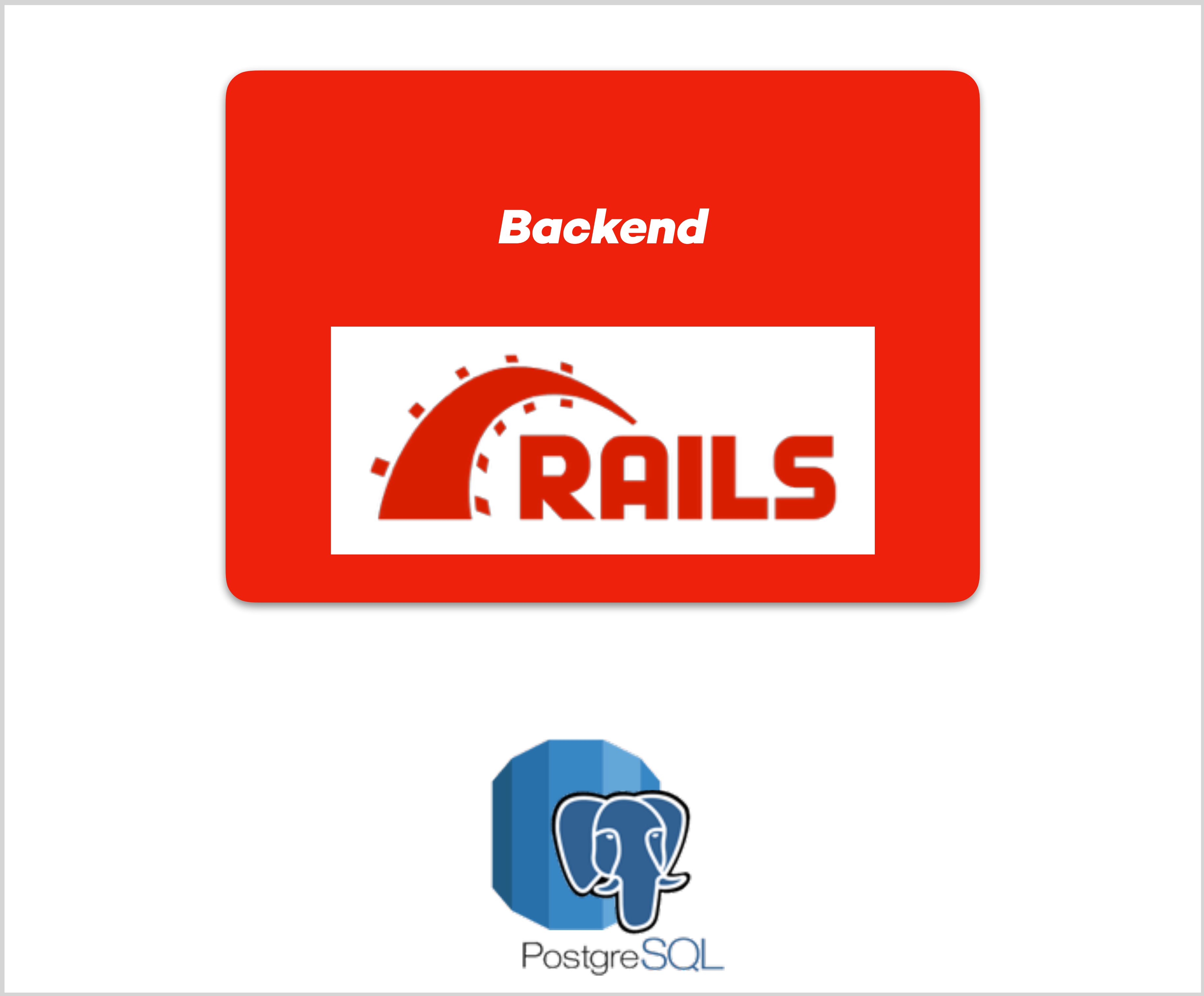
Benefits

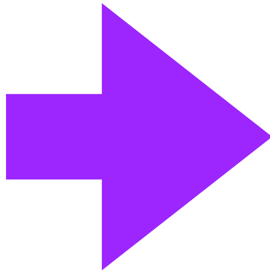
- Modularity
- Coherence and low coupling
- Fault tolerance
- Fast development
- Autonomy
- Independent deployment

Challenges

- Sharing data across services
- Debugging and tracing
- Orchestration
- Deployment
- Increased overall complexity
- Insight across service boundaries

Microservice 1, 2, 3 and 4





Learning 1

Never allow microservice A to access the data of microservice B directly

Learning 2

Reduce the number of new technologies introduced in one go

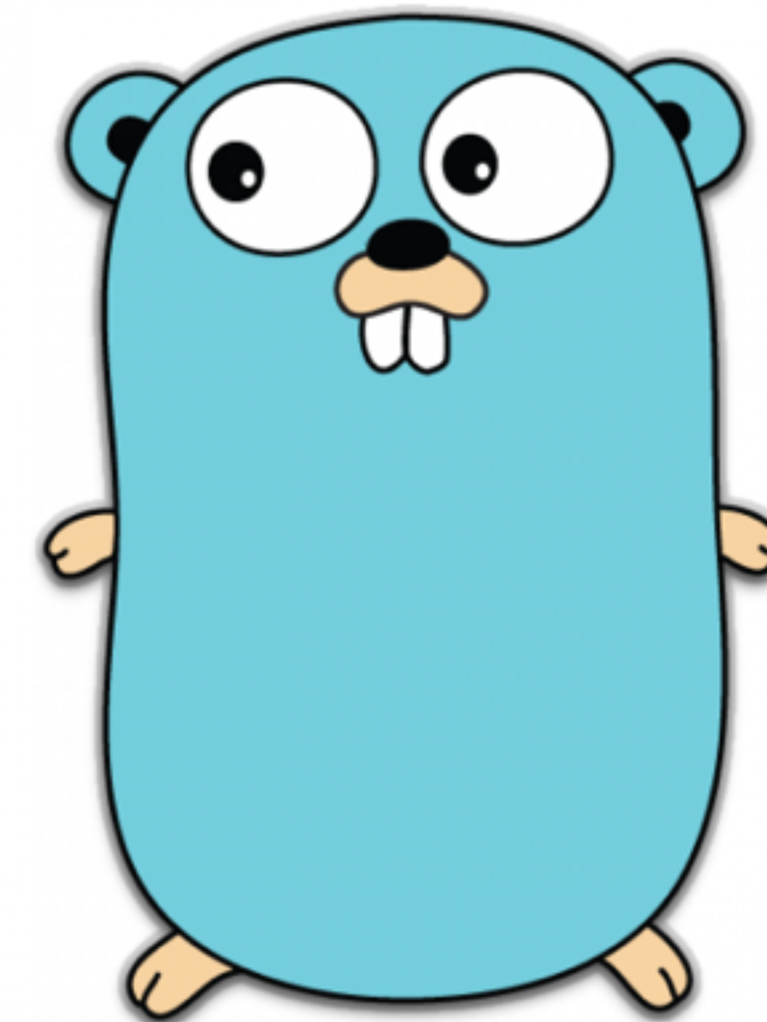
http://mcfunley.com/choose-boring-technology

Learning 3

DRY up your services – factor out common functionality into new services

Microservice X, Y and Z

VS



Learning 4

Insist on paying off technical debt

Learning 5

Choose your toolbox wisely

Microservice demo!

Deployment
lunar way
GOTO Nights CPH 2018
https://pixabay.com/en/site-concrete-construction-pillar-1892917/

Learning 6

Prioritise deployment pipeline and runtime platform from day 0

Learning 7

Automate!

Deployment demo!

Inter Service Communication

https://pixabay.com/en/team-motivation-teamwork-together-386673/

Inter service communication

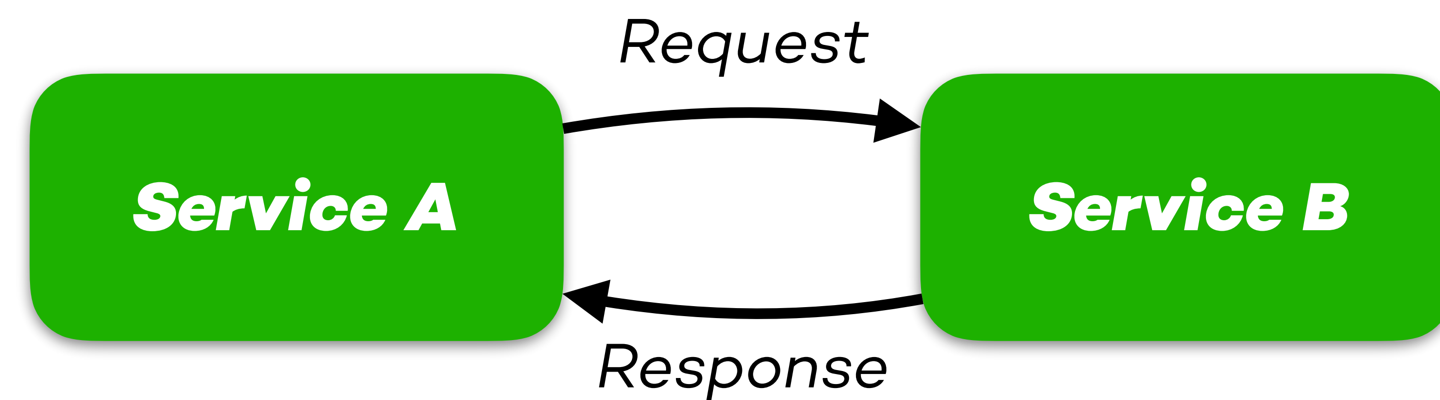
Synchronous req/resp

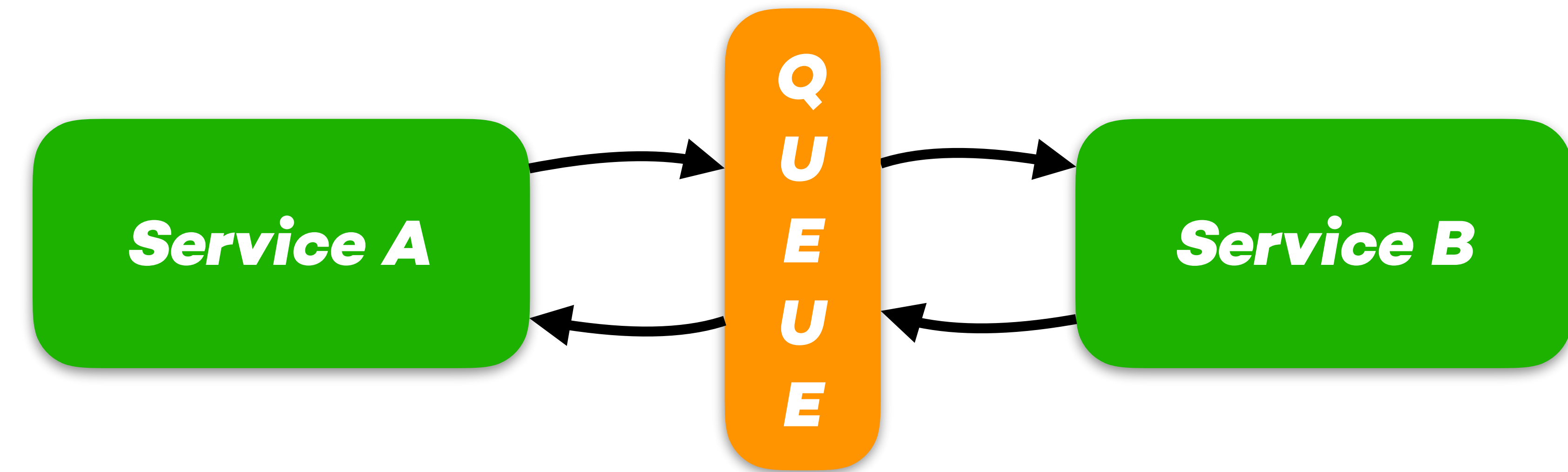
- Closed communication
- Trusted/ “secure”
- High coupling (code/space/time)
- Works good when synchronous app request involved

Async messages

- Open ended communication (pub/sub)
- Low coupling (data only)
- “Insecure” from a dev perspective
- Flow orchestration is complex
- Works bad when synchronous app request involved

Communication shapes coupling





Synchronous req/resp

Temporal coupling

Spatial coupling

Behavioural coupling - Service A commands the behaviour of Service B

Example: Signup commands user service to CreateUser

Async messages

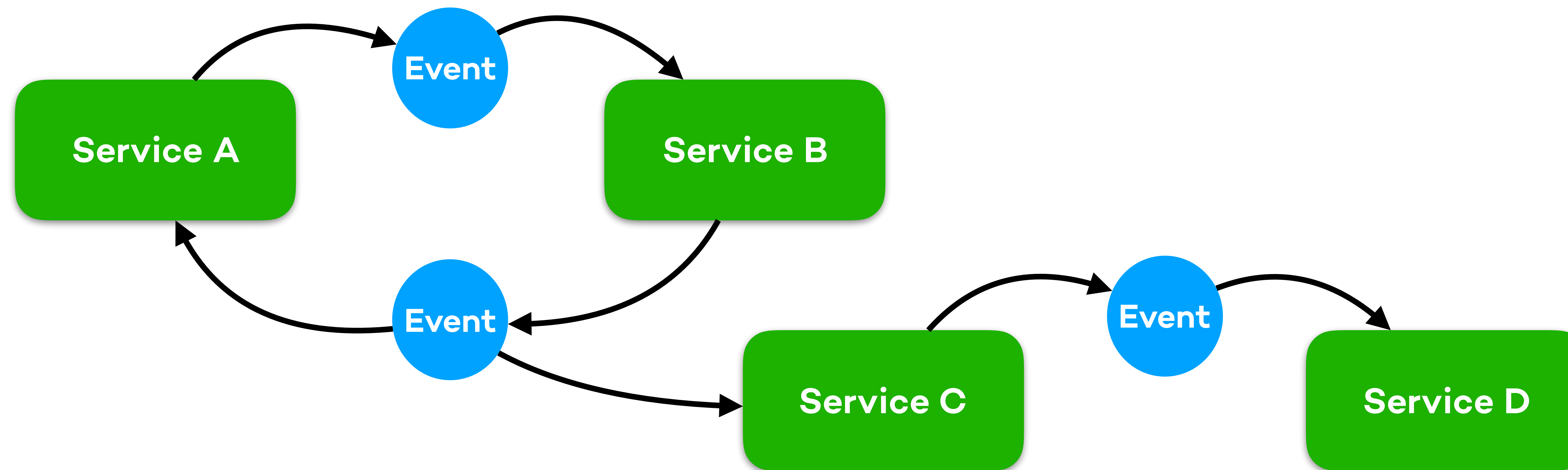
No spatial and temporal coupling

No behavioural coupling - Service B determines its own behaviour based on the behaviour of Service A

Example: Signup publishes UserApplied. User service consumes event, creates user and publishes UserCreated. Signup service consumes and changes state

https://iansrobinson.com/2009/04/27/temporal-and-behavioural-coupling/

Event driven systems



- All changes published as events
- Events drive behaviour
- Traditional system design focus on only the state changes - events disappear after they happen
- Event driven systems complete the picture

https://hackernoon.com/events-as-first-class-citizens-8633e8479493

Learning 8

Appreciate the tremendous value of an event driven system

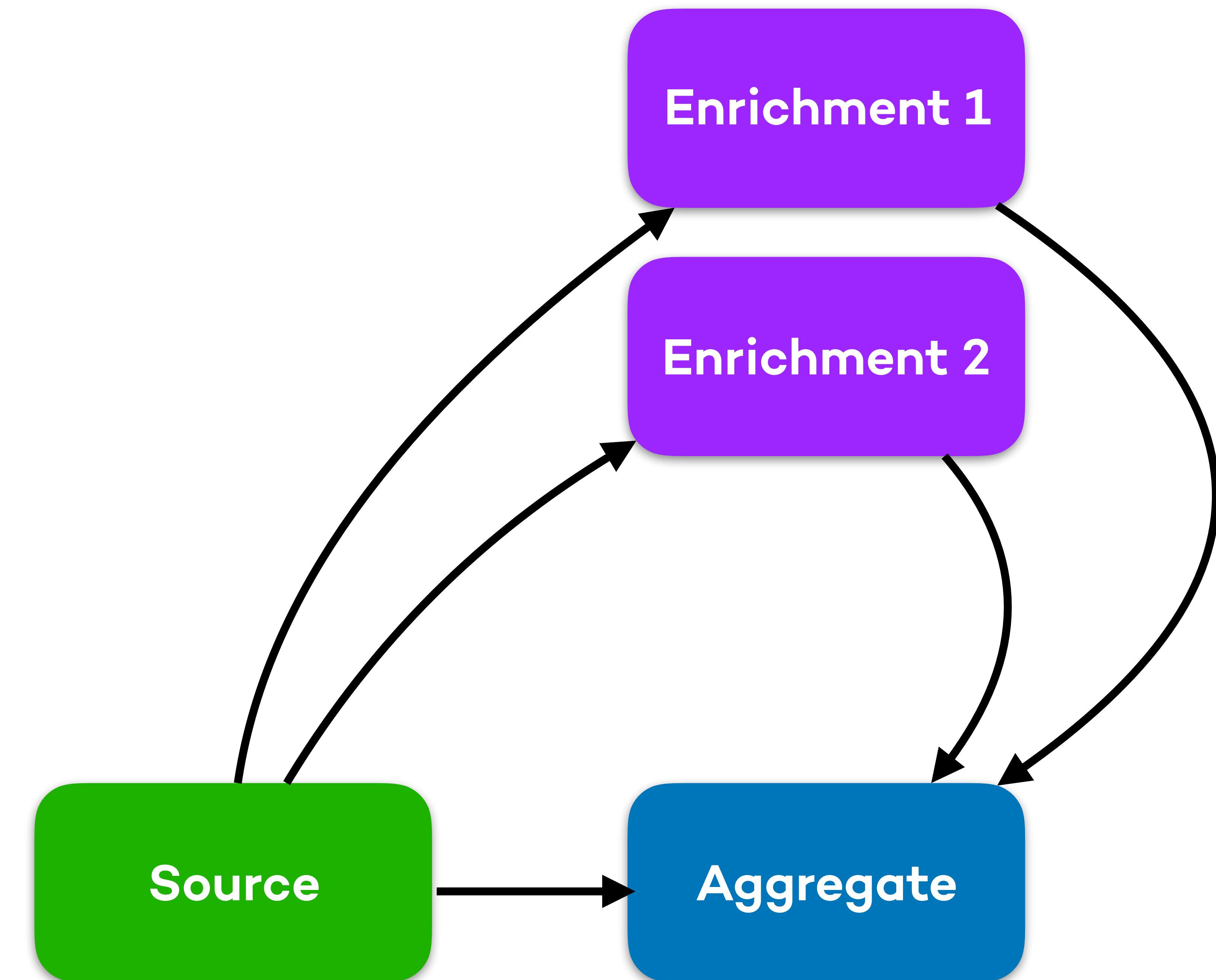
Async events demo!

Use Cases of Async Message Passing

Data enrichment

One source of data, multiple enrichments

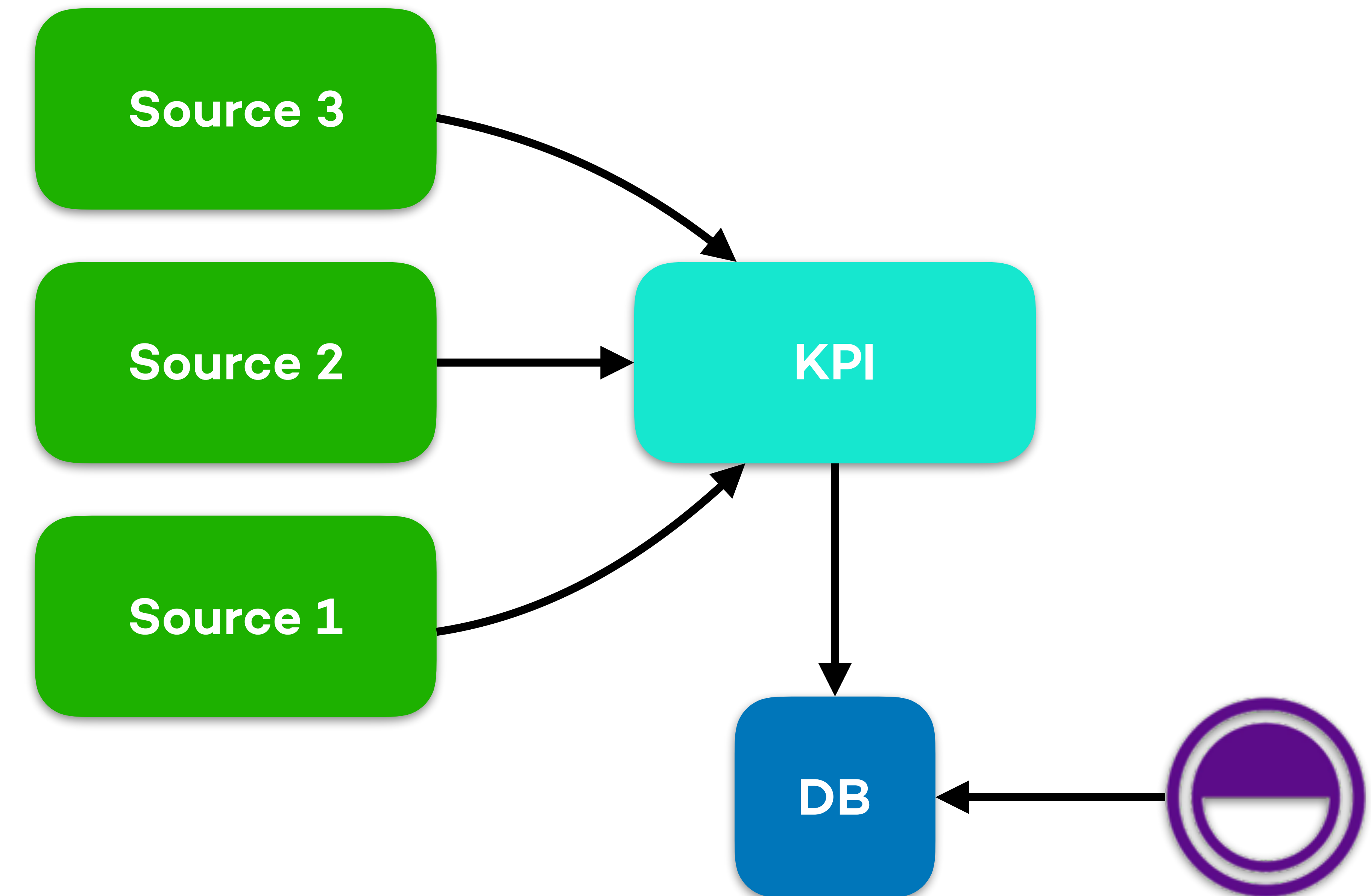
1. Source publishes event
2. Aggregate stores entity
3. Enrichments runs and publishes event
4. Aggregate updates aggregate with enrichment



Business insight

Business requires insight across services

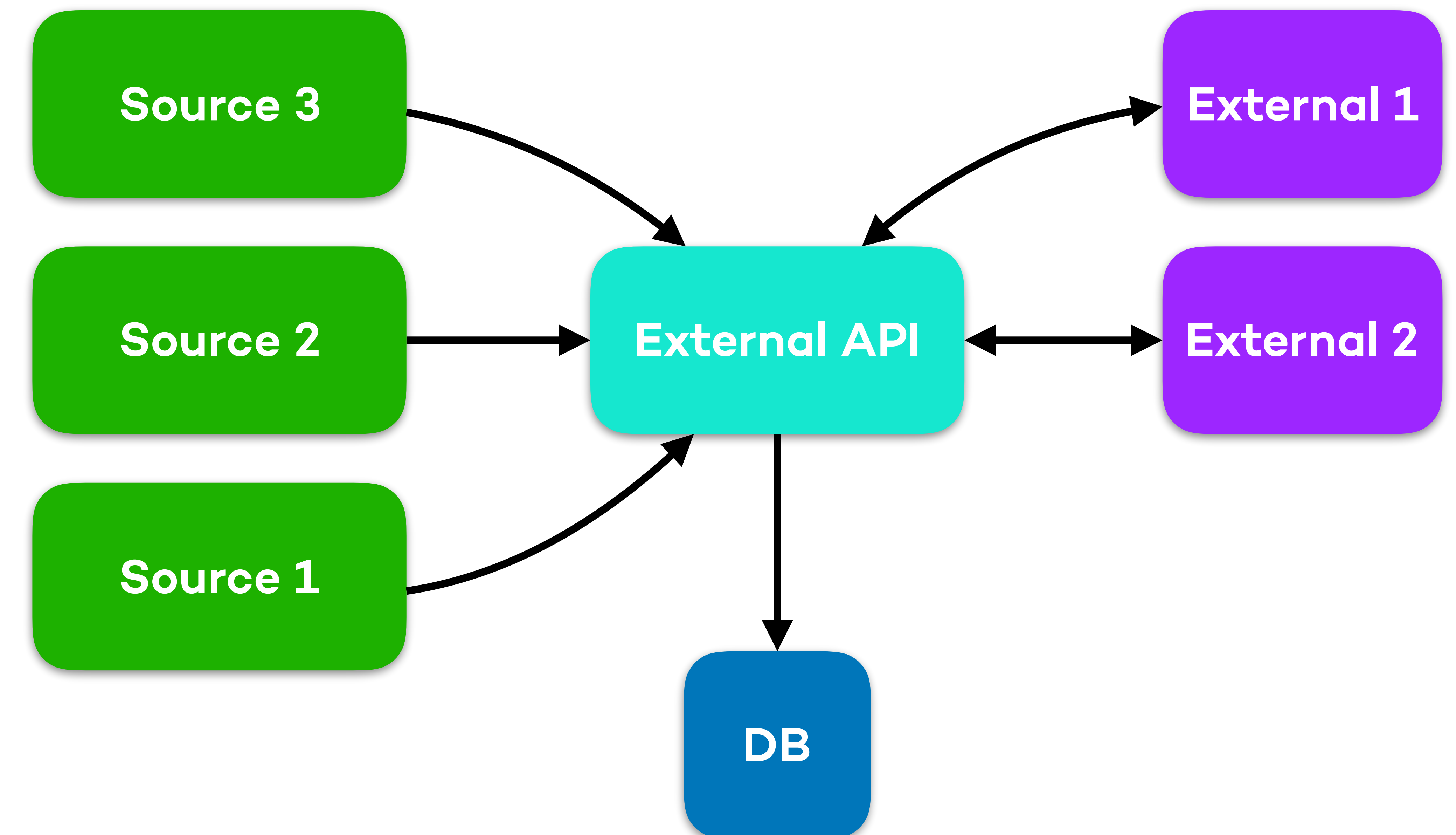
1. Sources publish events
2. KPI subscribes on events and converts to own model
3. Tooling on top to provide insight



External APIs and web hooks

3rd parties require access to your data

1. Sources publish events
2. External API subscribes on events and converts to own model
3. 3rd parties access external API and may register web hooks



Logging

https://pixabay.com/en/logs-wood-timber-lumber-woodpile-498538/

Microservice N,
N+1, N+2...

Beyond
counting...

Learning 8

Be systematic!

Logging demo!

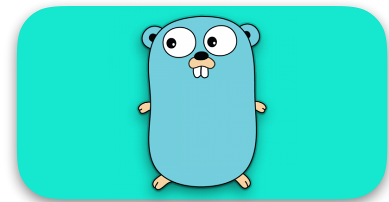
Observability

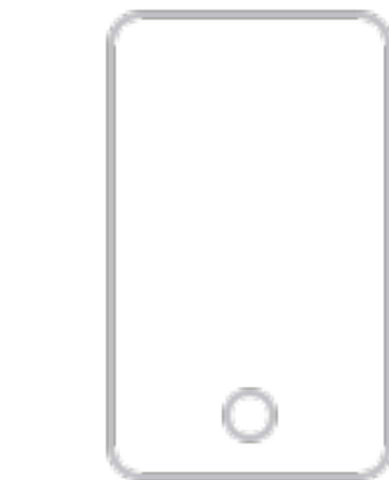
https://pixabay.com/en/logs-wood-timber-lumber-woodpile-498538/

Service metrics

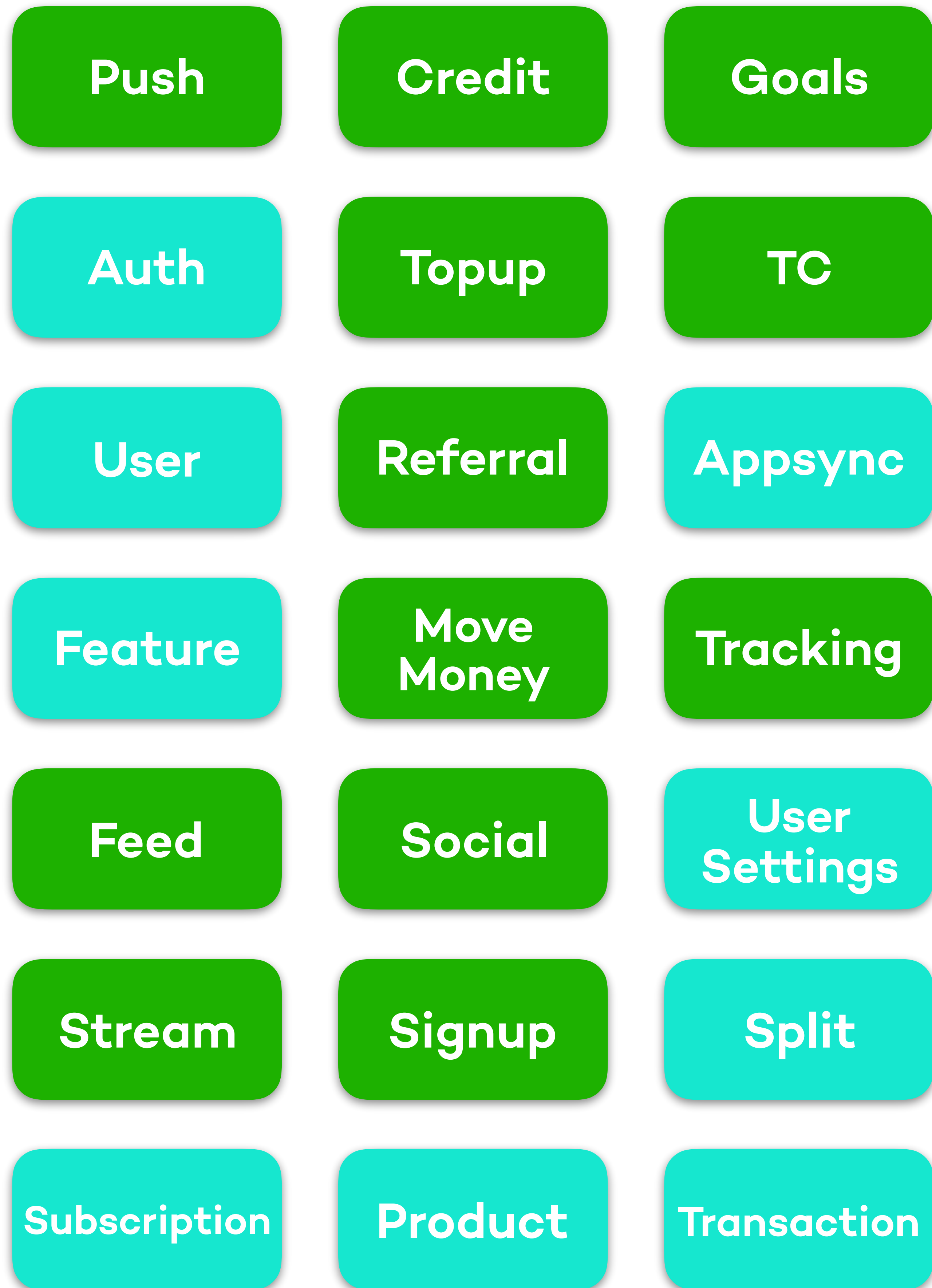
- K8s metrics
- API metrics (HTTP, gRPC)
- Event metrics
- Runtime metrics
- Service specific metrics

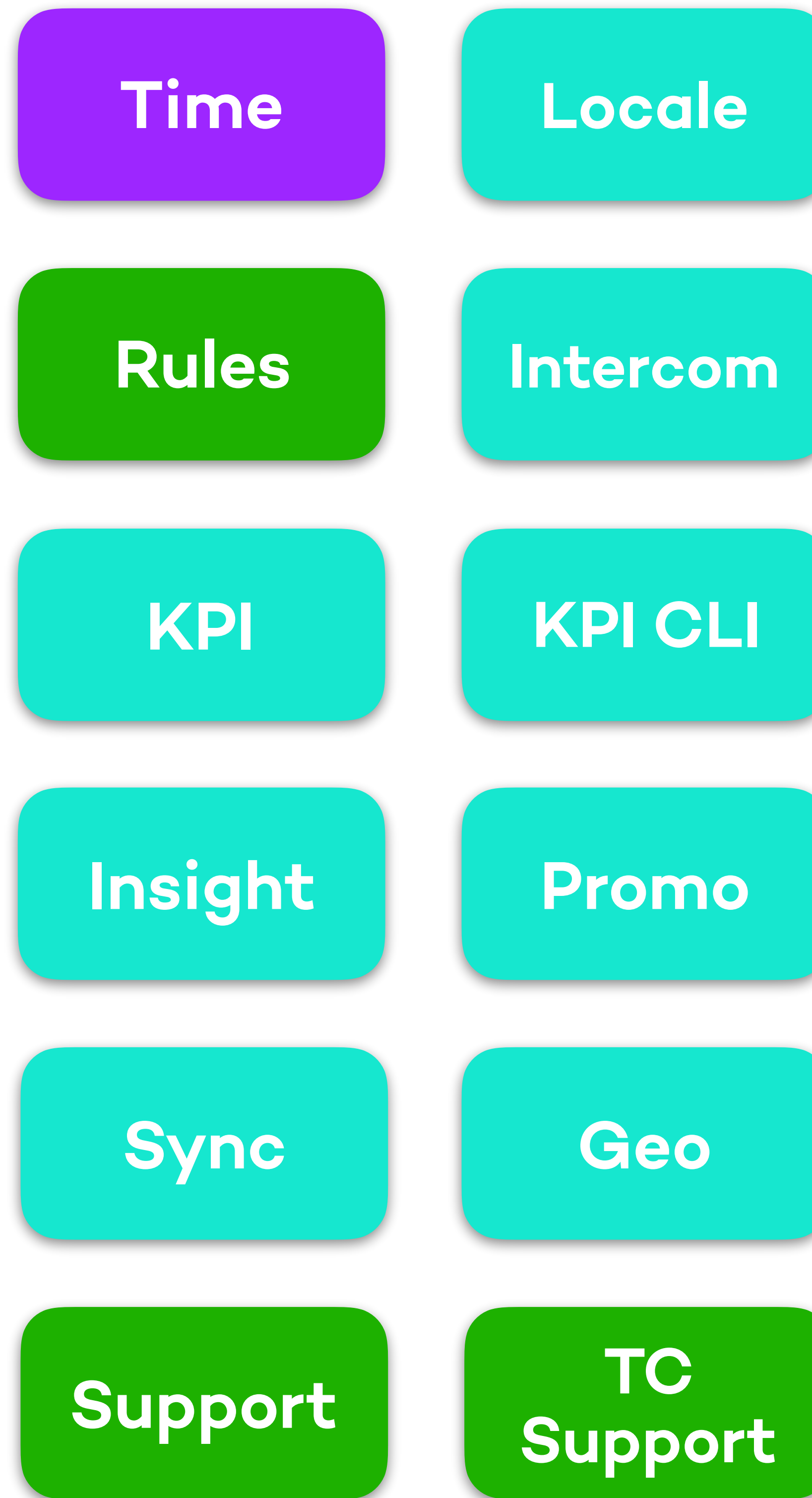
Metrics demo!

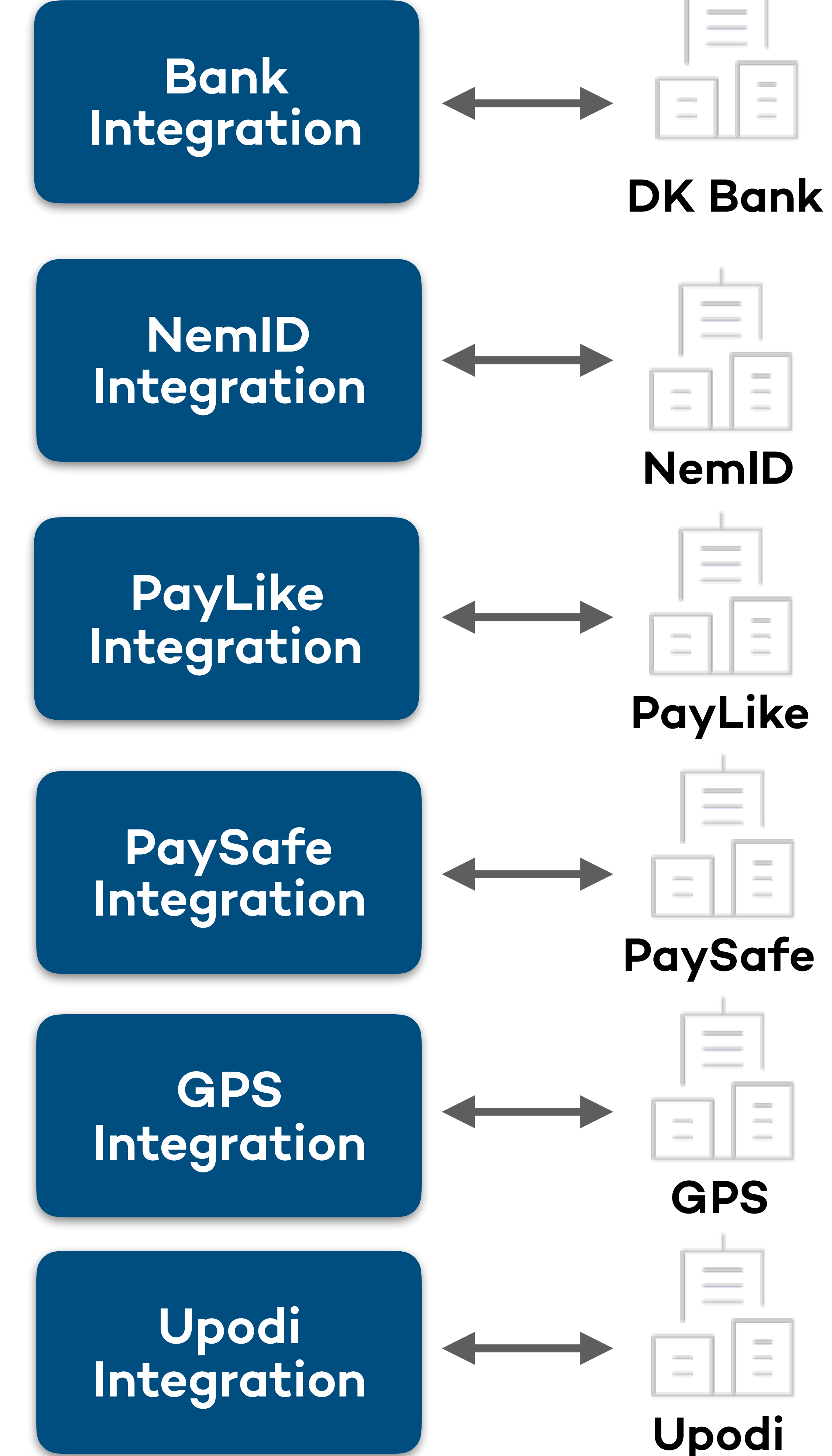



Native iOS and Android apps

Houston (Support)







The Road Ahead

https://pixabay.com/en/road-winding-street-bridge-1030789/

What’s left?

Event system

- Event before state change
- Replay of events

Platform

- Ease of service setup
- Scalable DB
- Replace RabbitMQ

Going fully async

Current model

- App requests are completed synchronously
- Major pain in the backend
- Downstream failures result in error

Possible future model

- App requests are "intentions" acknowledged synchronously
- Result pushed to the app
- Failures downstream handled by queuing execution of the intent

Wrapping up

Key takeaways if entering microservice land

Adapt to the size of your team

Use asynchronous communication between services… preferably event driven

Prioritise your deployment pipeline and runtime platform from the start

Be systematic and automate!

lunar way®