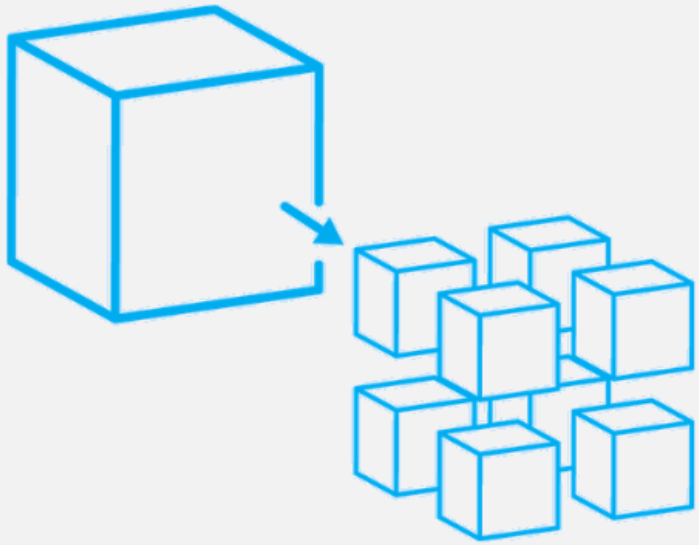
Microservices

Sanbercode Super Bootcamp 2021

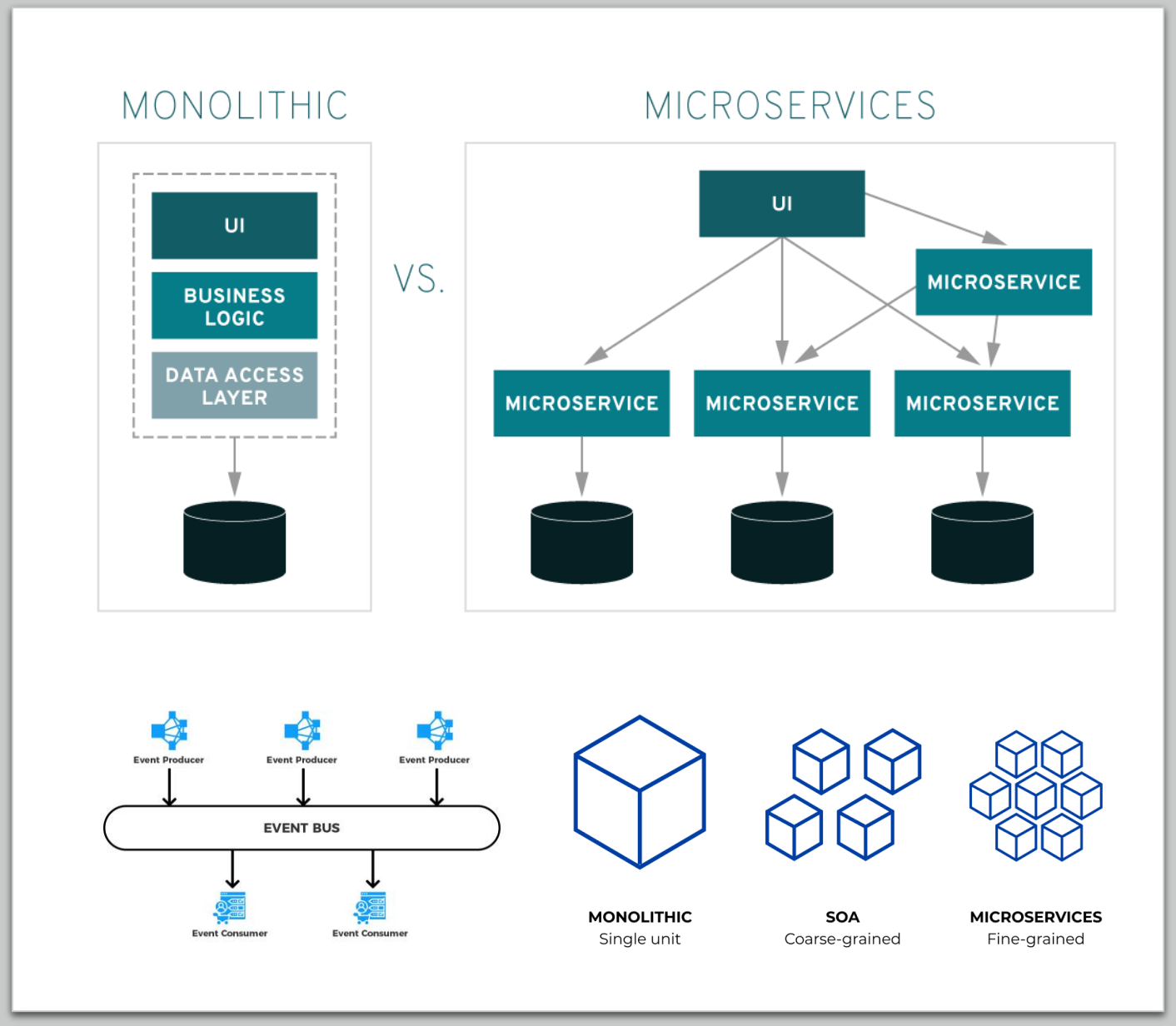
Ardhi Rofi . Miftahul Arifin

2021-02-24

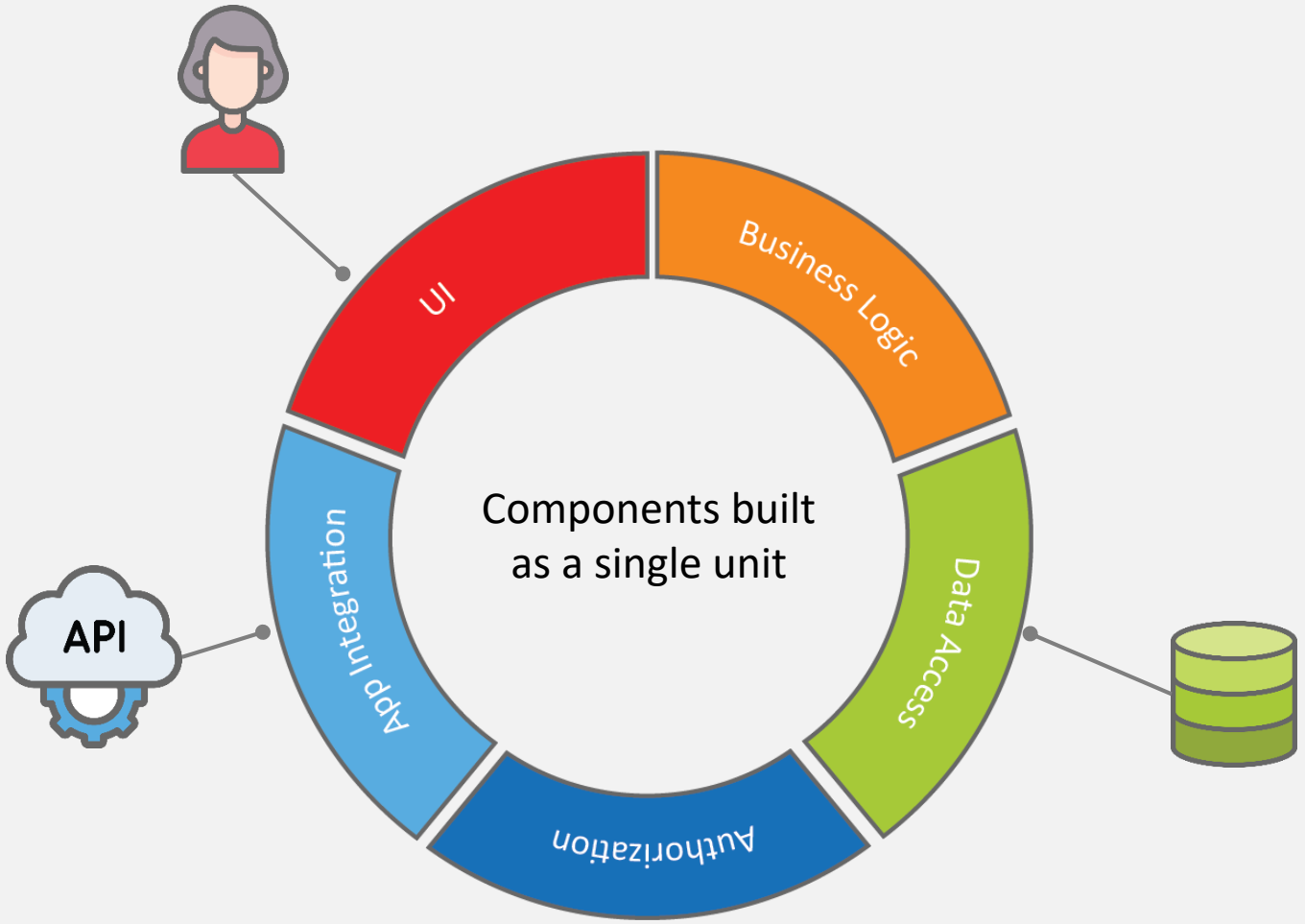


Application architecture

- Monolith architecture
- Microservice architecture
- Event-driven architecture
- Service-oriented architecture (SOA)



Monolith architecture



Monolith architecture

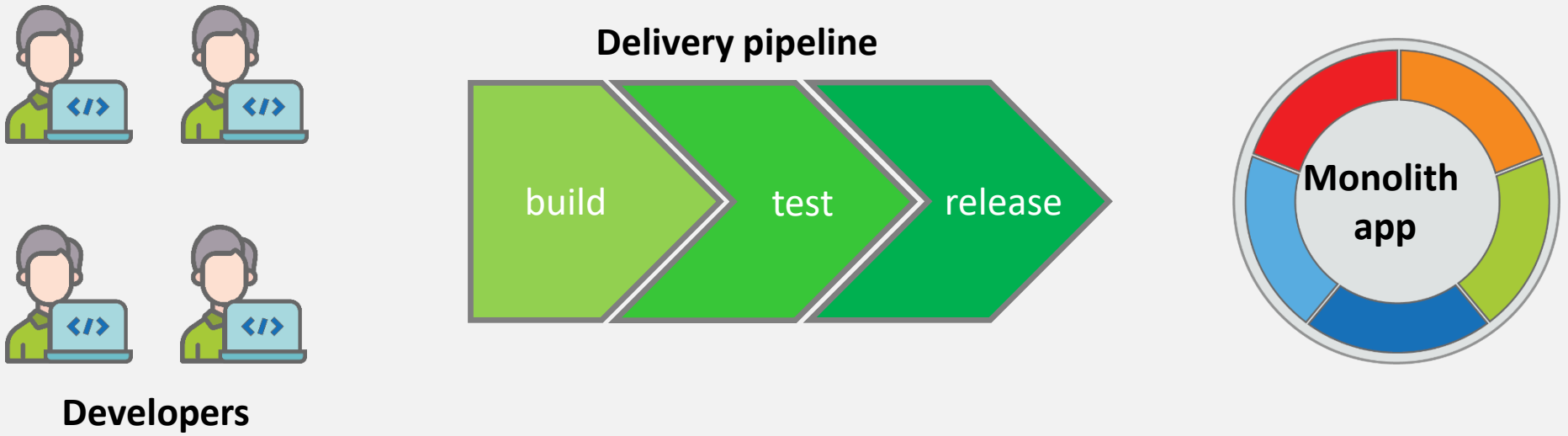
Strengths

- Can be good for starting out development
- Simple to develop
- Simple to deploy
- Easier debugging and testing

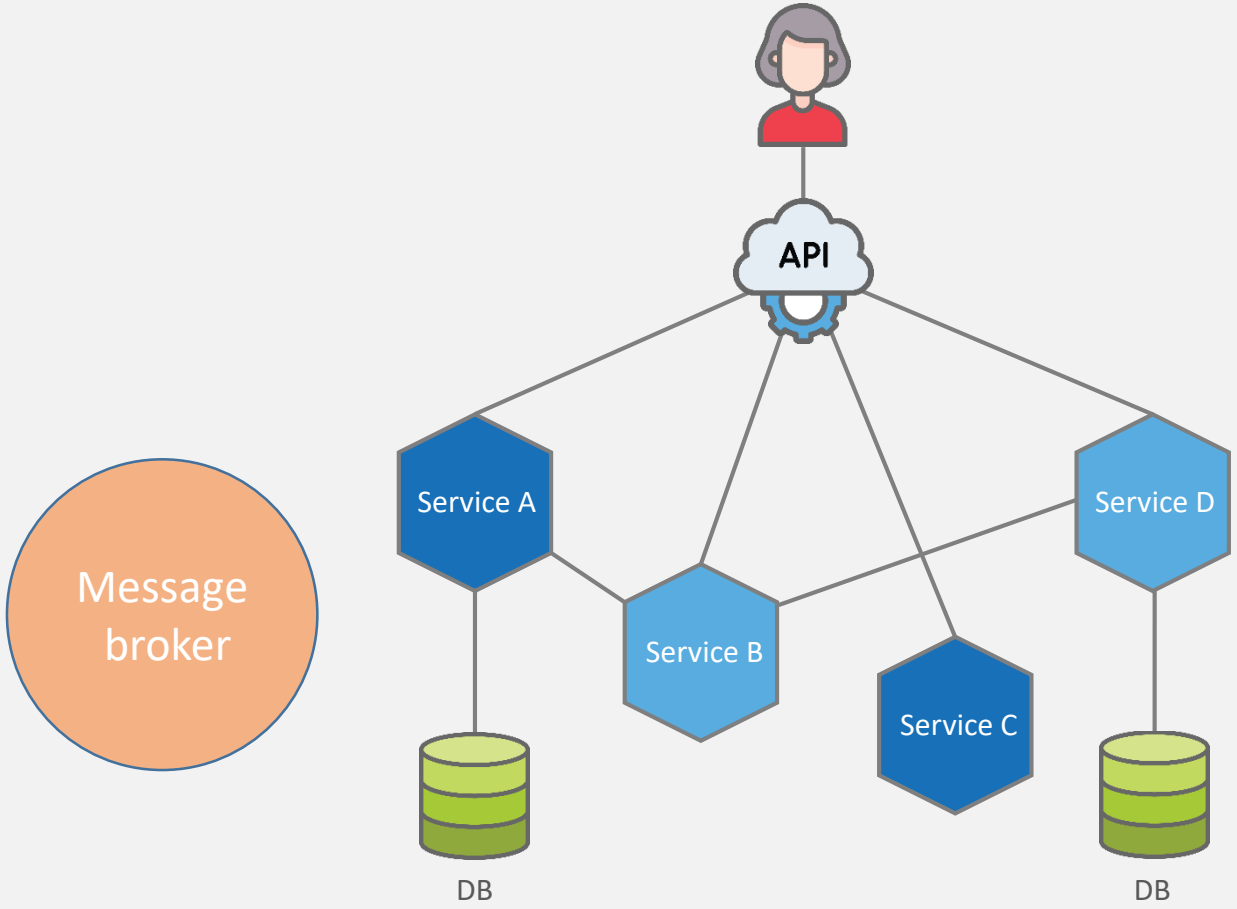
Challenges

- Difficult to scale
- Long build/test/release, need to redeploy entire app
- Difficult to maintain a large and complex project
- Not reliable, any faulty could impact the entire process
- Lack of innovation, releasing new features could take a long time
- New technology barriers
- Large app tends to slow down

Monolith development lifecycle



Microservice architecture



Definition of microservice

“**Service-oriented architecture**

composed of

loosely couples elements

that have

bounded contexts”.

- **Service-oriented architecture** → Service communicate each other over network
- **loosely couples elements** → Service can be update independently
- **bounded contexts** → Self-contained. Code can be updated without knowing anything about the internals of other services

Adrian Cockcroft – VP Cloud Architecture Strategy AWS

Microservice architecture

Strengths

- Continuous development & deployment
- Better testability & deployability
- Team organizing
- Improve performance & fault isolation
- No long-term commitment to technology
- Clear ownership and accountability

Challenges

- Complexity of distributed system
- Inter-service communication mechanism
- Team coordination
- Use cases that span multiple services
- Deployment complexity
- Automate everything (use of DevOps)
- Service Monitoring
- Decentralized data management

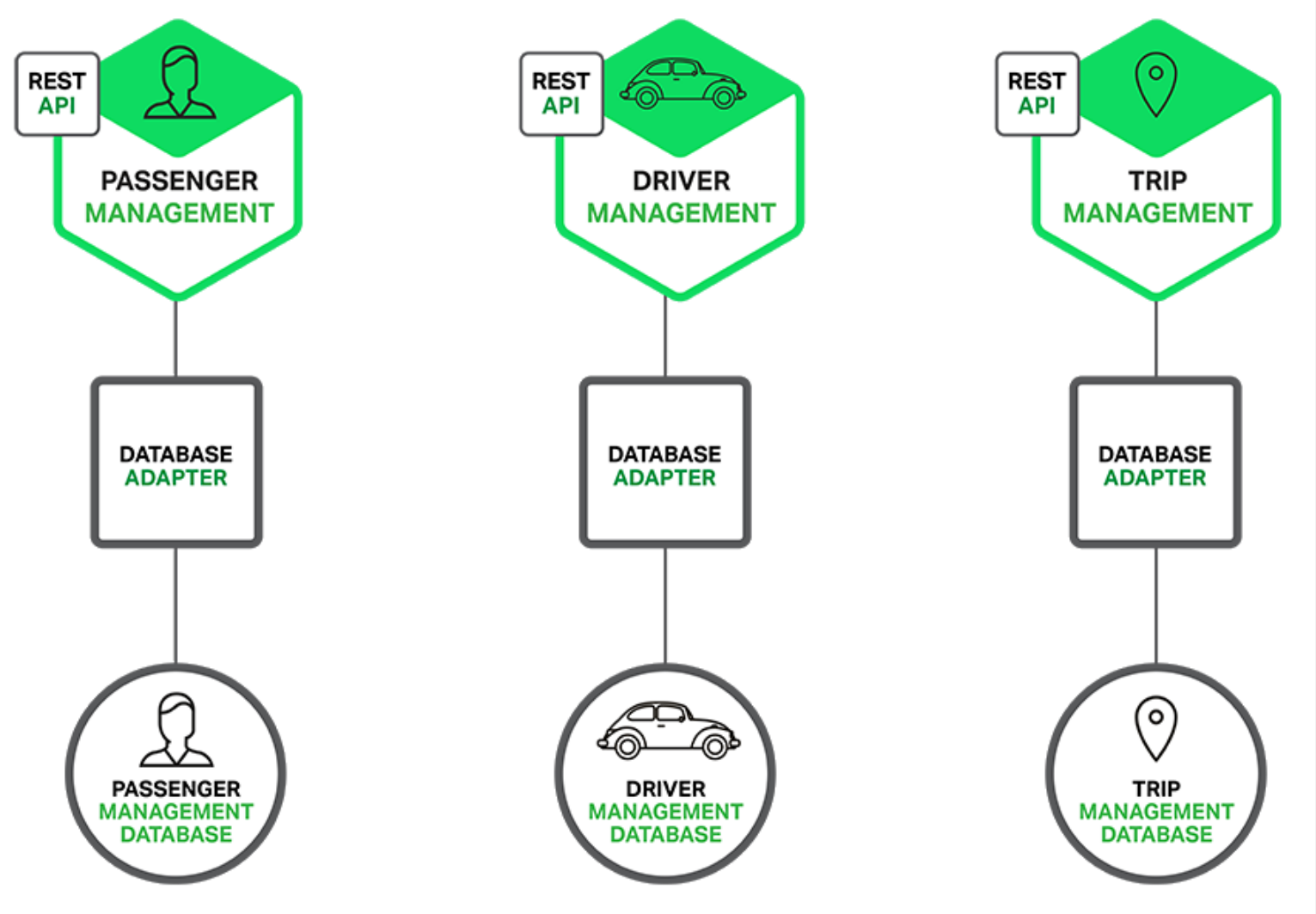
Microservice principles

- Highly maintainable and testable, Isolate failure, Observable
- Loosely coupled, Decentralise, Hide details
- Independently deployable
- Business domain-based model
- Single responsibility, owned by a small team
- Automation culture
- Event driven

Service communication

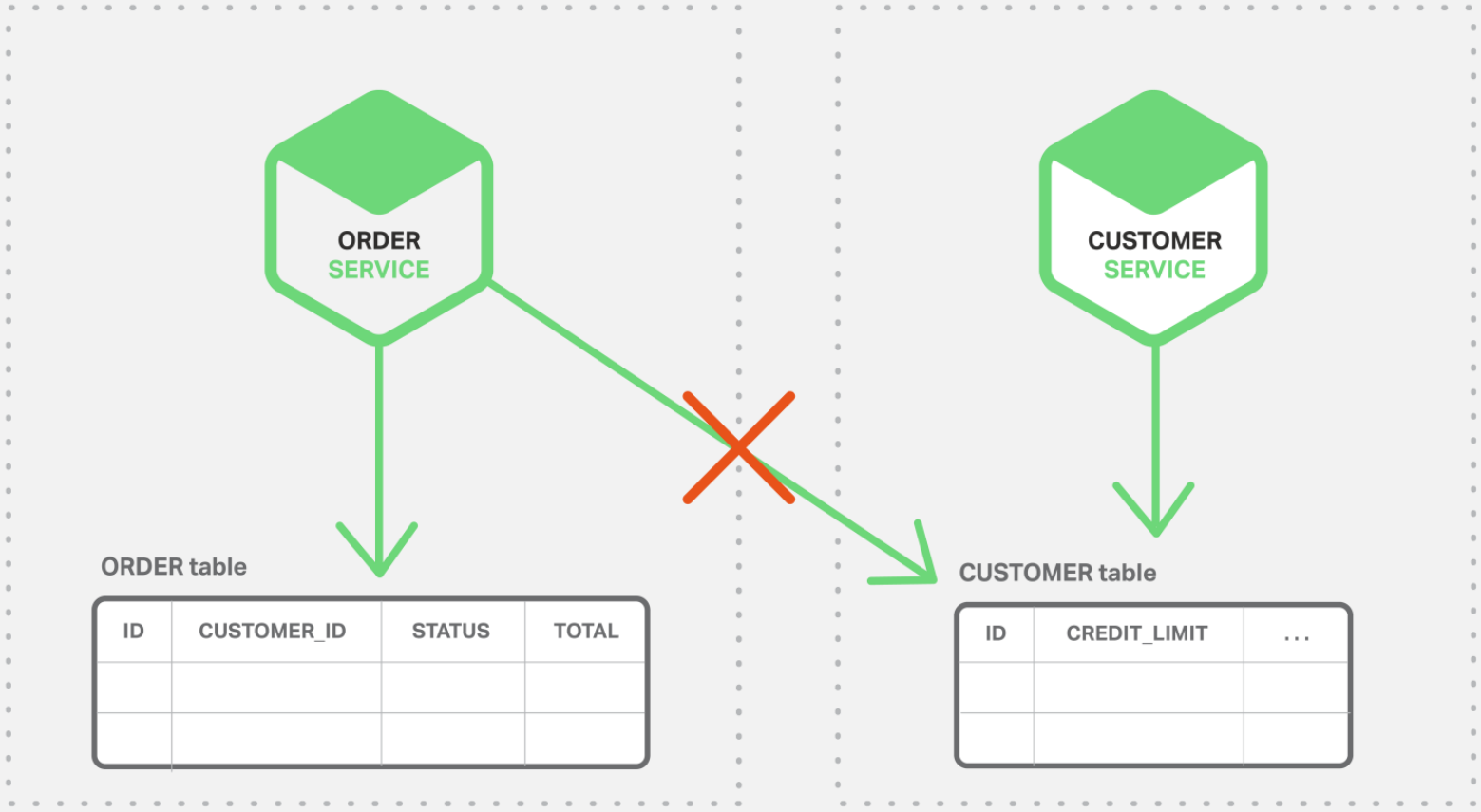
- HTTP call (to a REST service)
- RPC-like, such as gRPC or GraphQL
- Message Queuing (NATS, MQTT, AMQP)

Data Decentralization



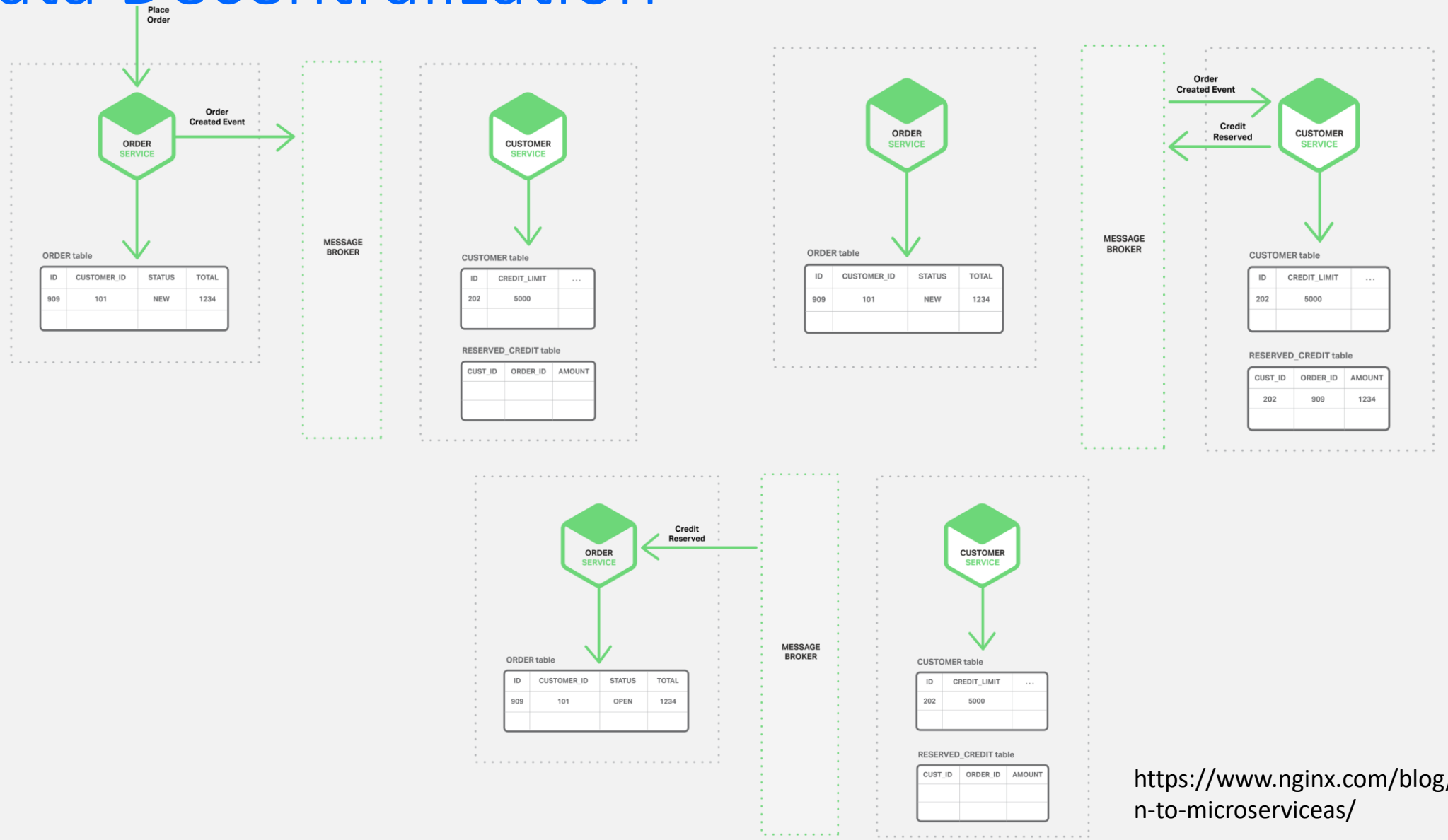
https://www.nginx.com/blog/introduction-to-microserviceas/

Data Decentralization



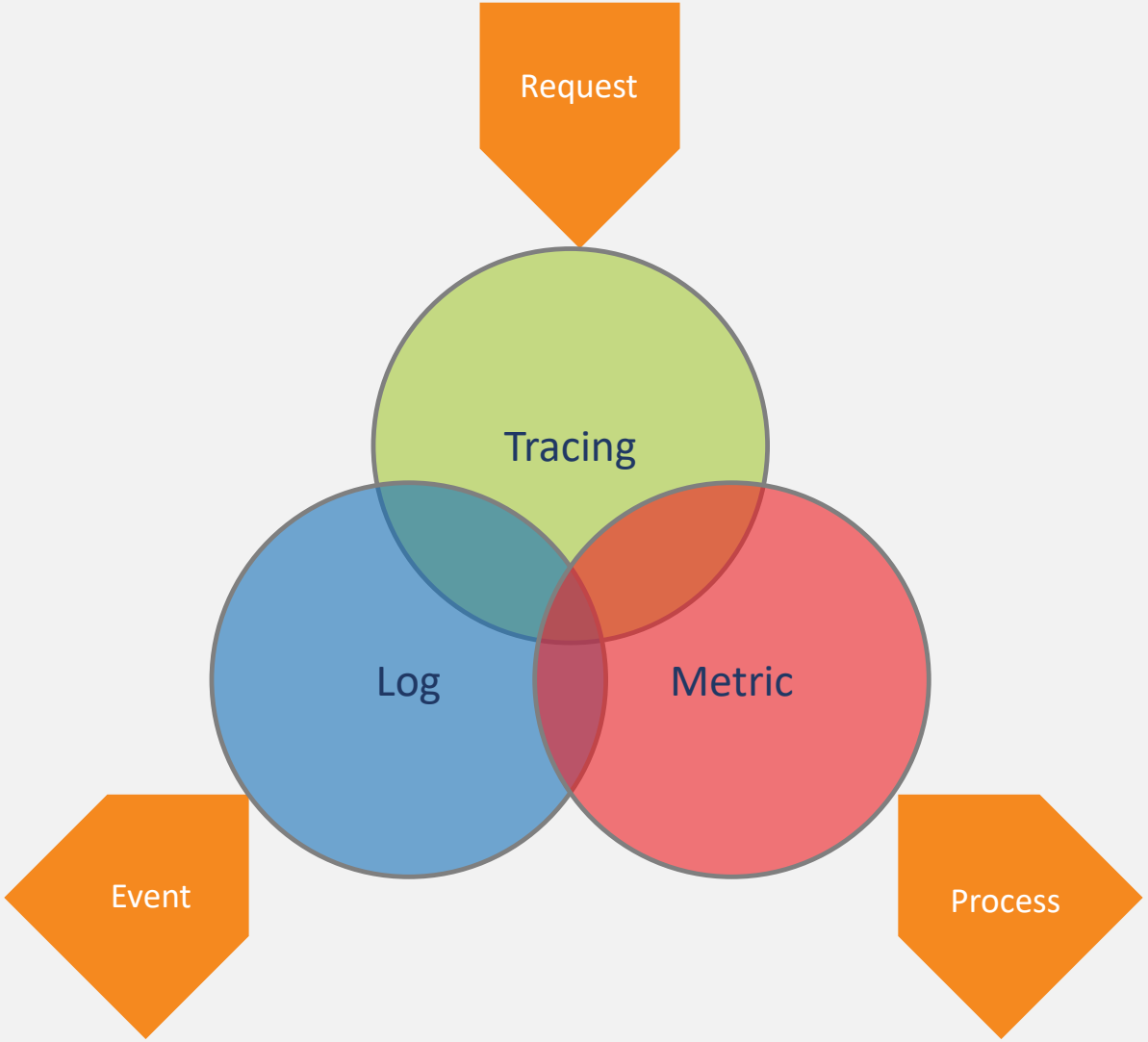
https://www.nginx.com/blog/event-driven-data-management-microservices

Data Decentralization



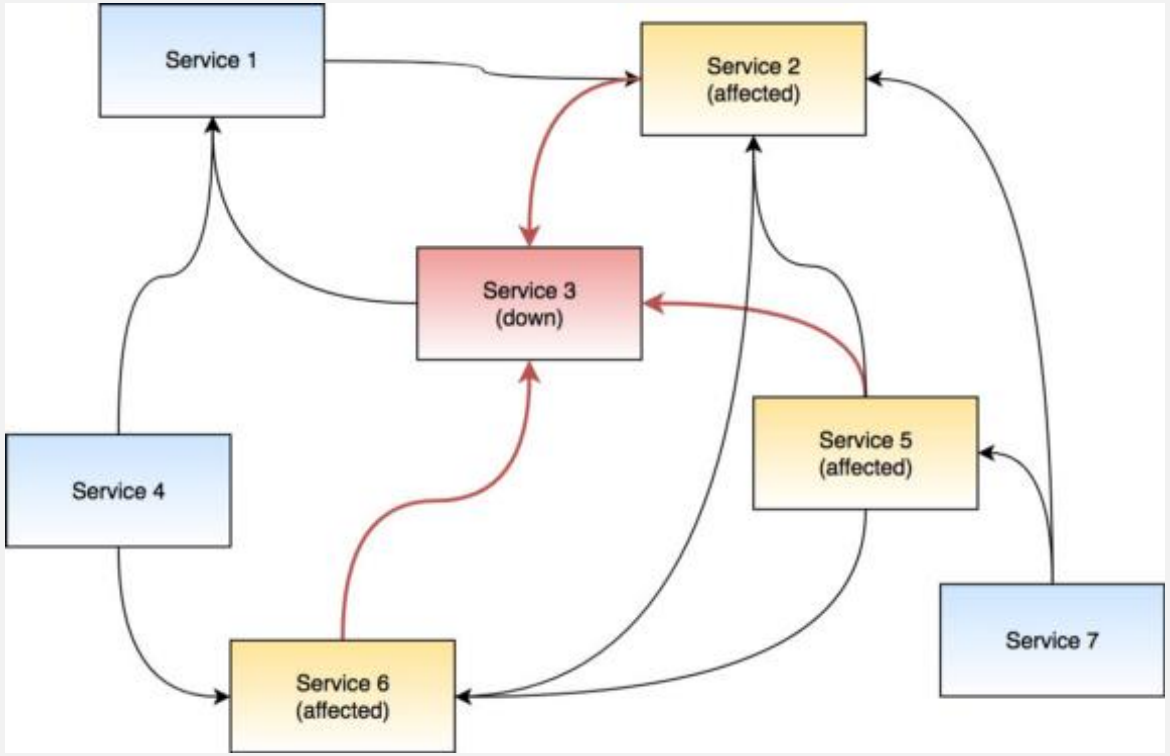
https://www.nginx.com/blog/introduction-to-microserviceas/

Observability

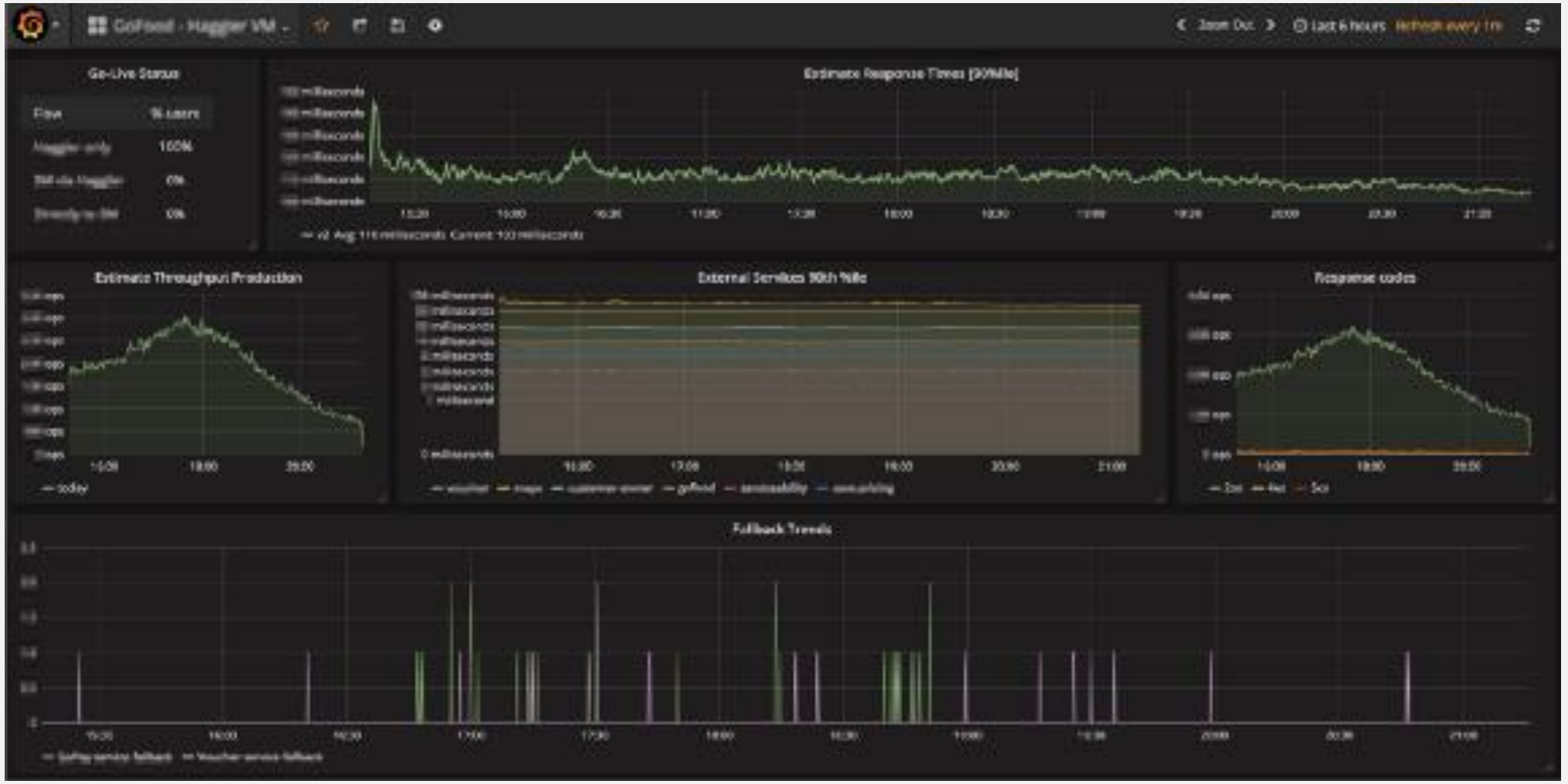


Monitoring

- Each services generate information about their information such as warning, errors, or debug messages.
- Detects and track errors, monitor service health, gather service metrics, failure alert
- It's a good practice to use centralized monitoring
- Jaeger tracing, Zipkin, Open Tracing, AWS Cloud watch



Monitoring



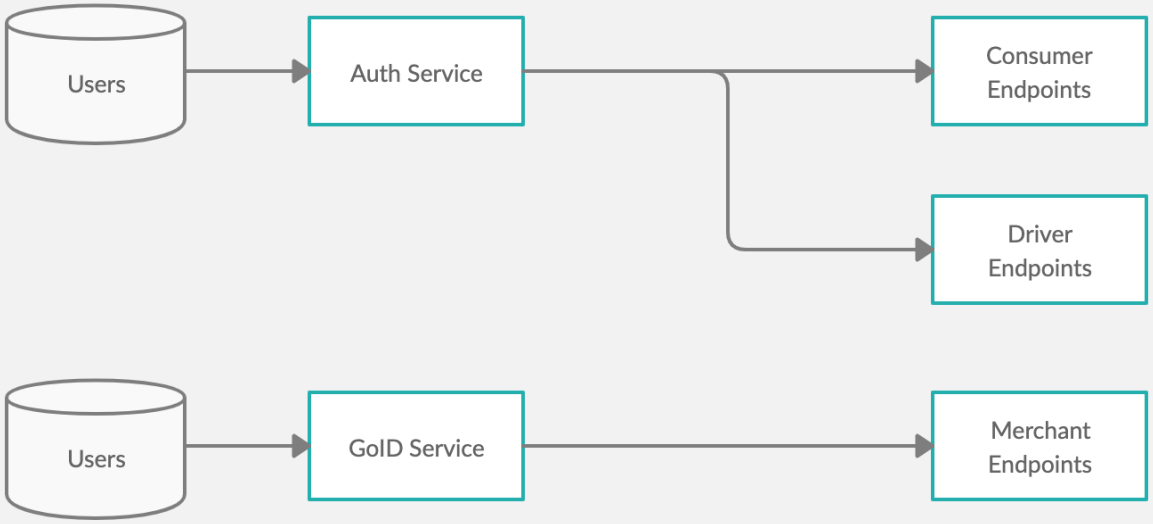
https://miro.medium.com/max/576/1*Oi3PxdKsSZpxhXLwlQ2eVg.png

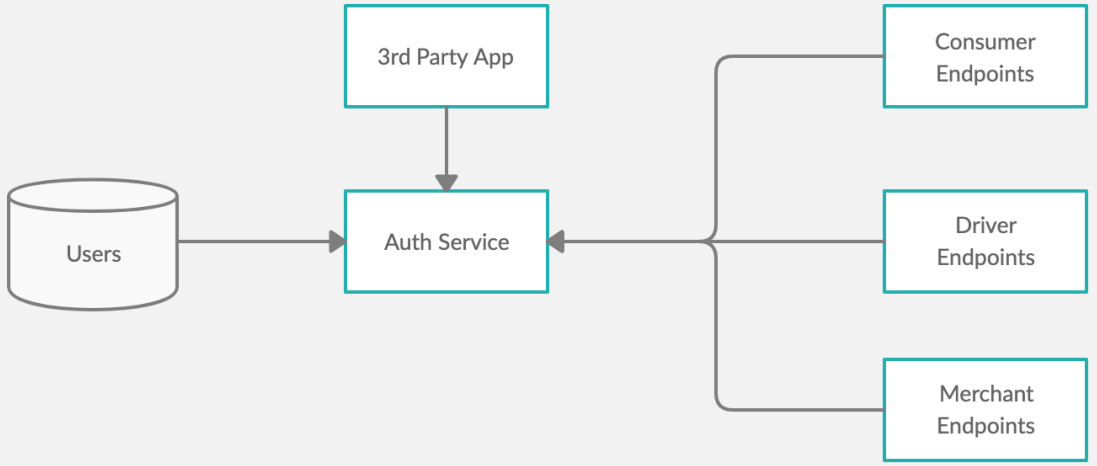
When to use microservice?

- Don’t start with microservices.
- Only when you feel the pain and complexity of the monolith begin to creep in is it worth considering how you might refactor that application into smaller services.
- Build and test services on a large scale
- When we need to scale-up separately (DevOps or cloud services)
- Keep in mind that it's all for maximize your apps development, performance and availability.
- See https://microservices.io/ for more use case

Real Implementation

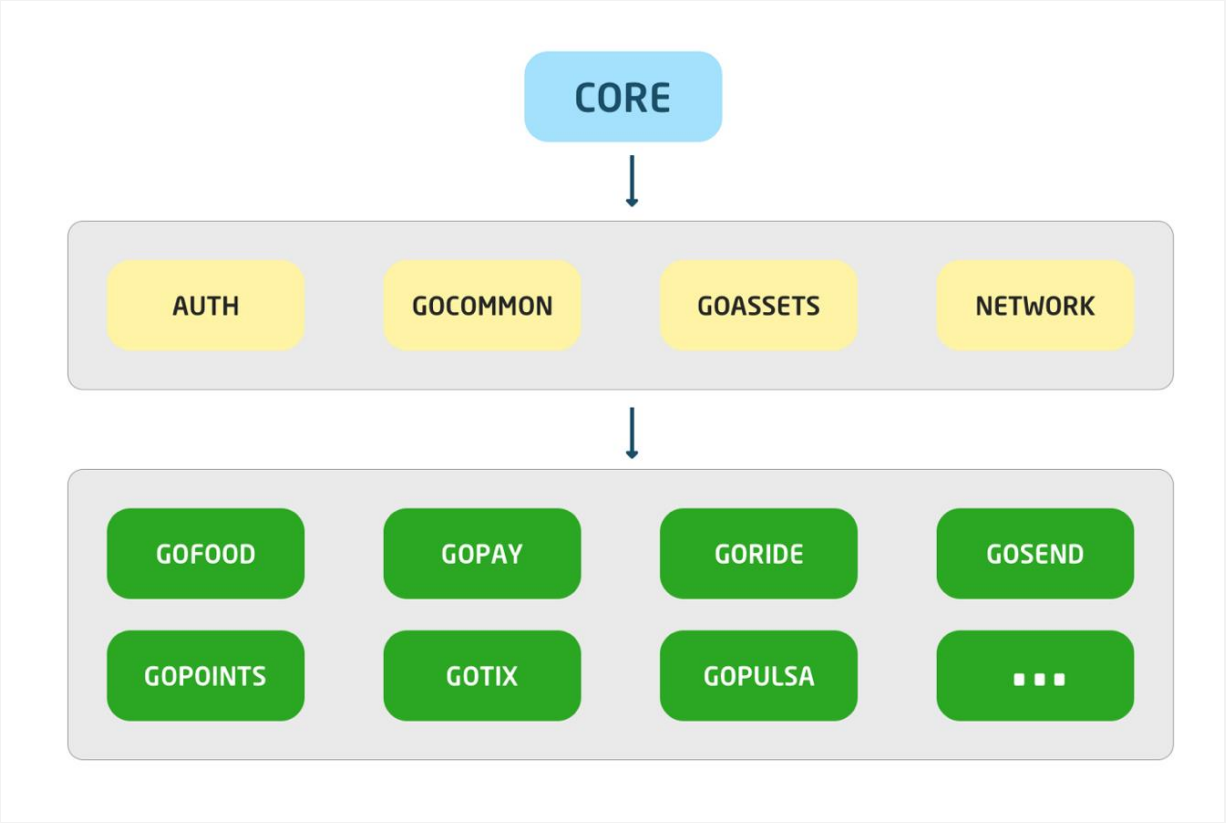
Previous Gojek Auth architecture (left) is having difficulties while trying to standardize user properties and perform operations due to heterogenous user account. With the new GoID (right), all Auth data is consistent among different user types.





Real Implementation

First Gojek Arch changing from monolithic to microservices on Gojek's iOS codebase. It sandbox all product codebases into their own repositories



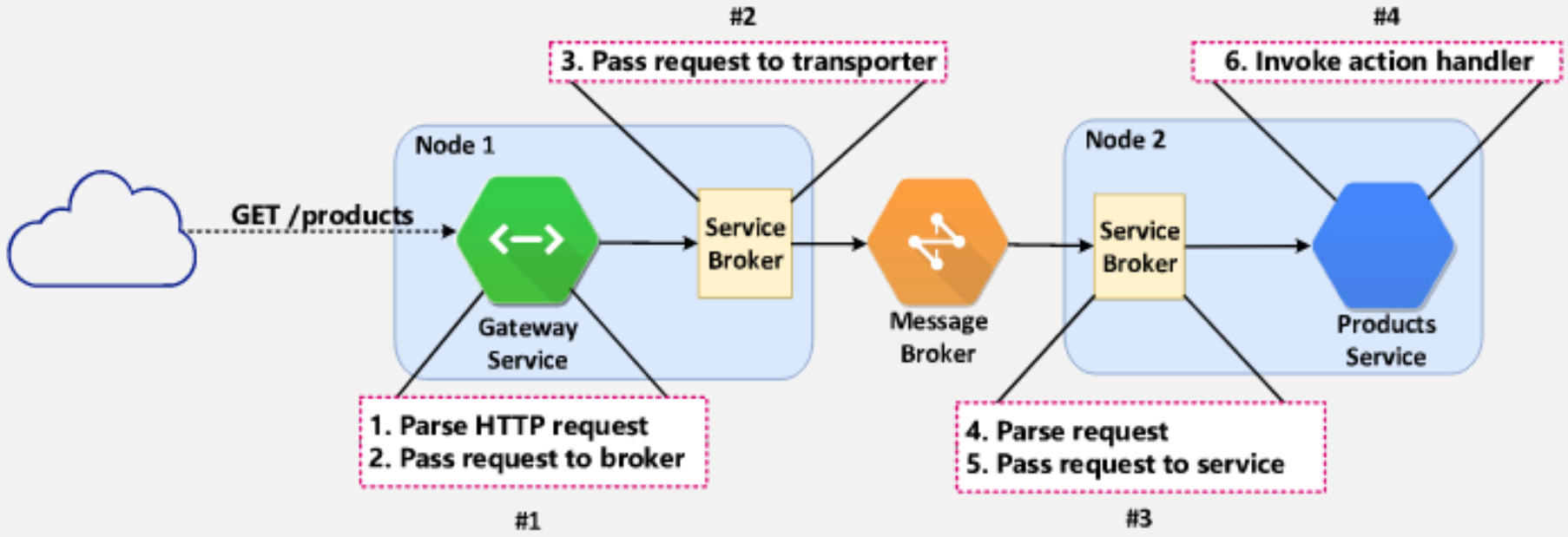
HANDS ON TIME

Moleculer

- Progressive microservices framework
- Built in modules (caching, serializer, transporter)
- Fault tolerance (built in load balancer, circuit breaker, retries, timeout and bulkhead features)

Moleculer

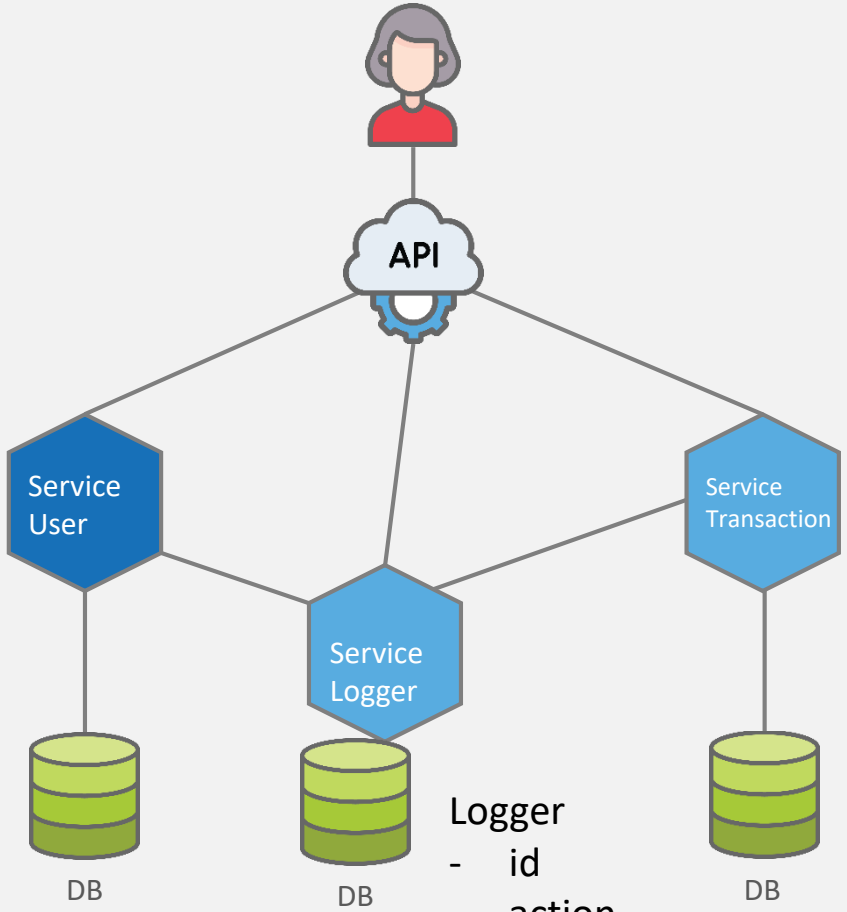
- Node
- Service : local and remote
- Service broker
- Transporter
- API Gateway



It’s your turn

E-Wallet

User
- id
- name
- email
- address



Transaction
- id
- to
- from
- value

Logger
- id
- action
- date