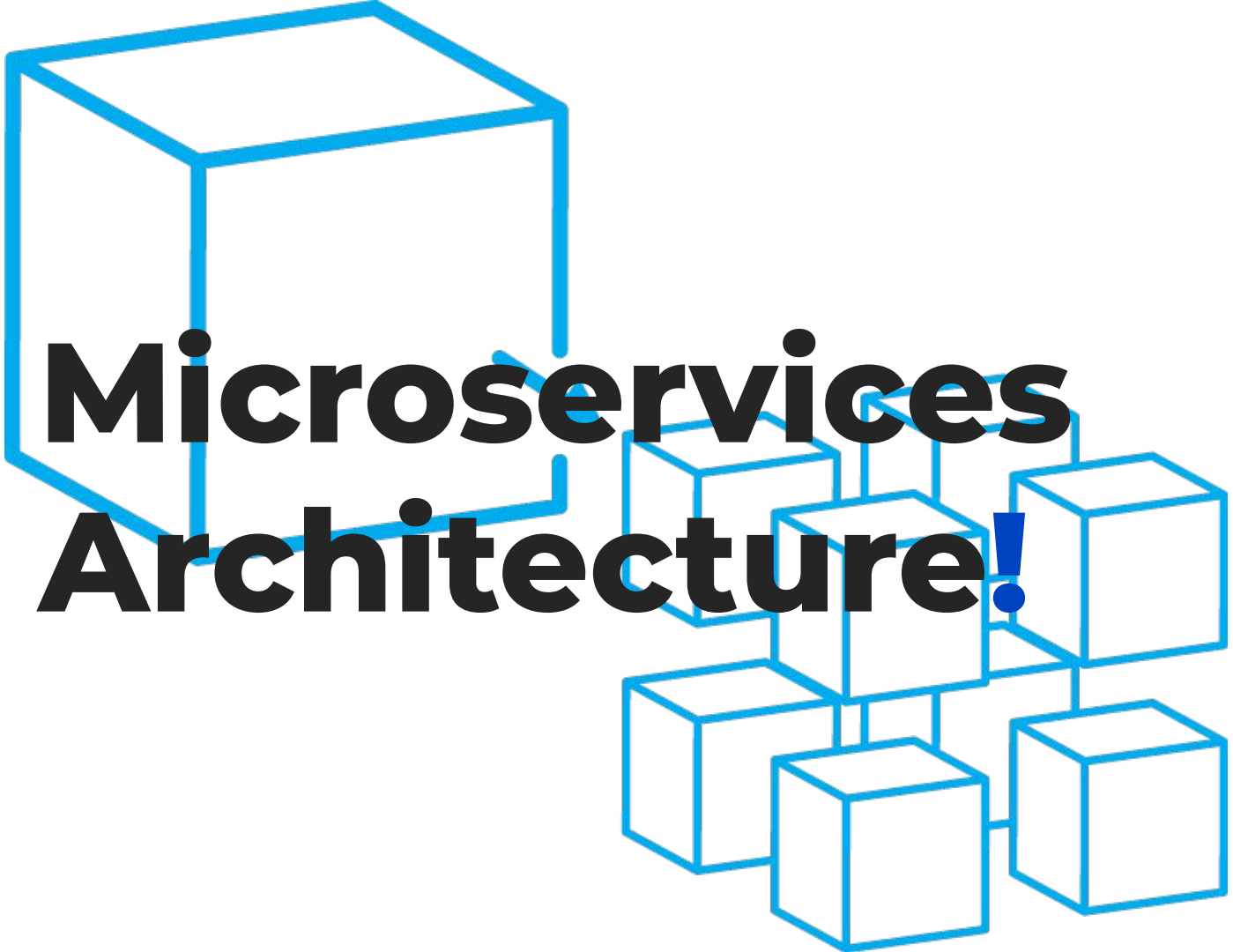
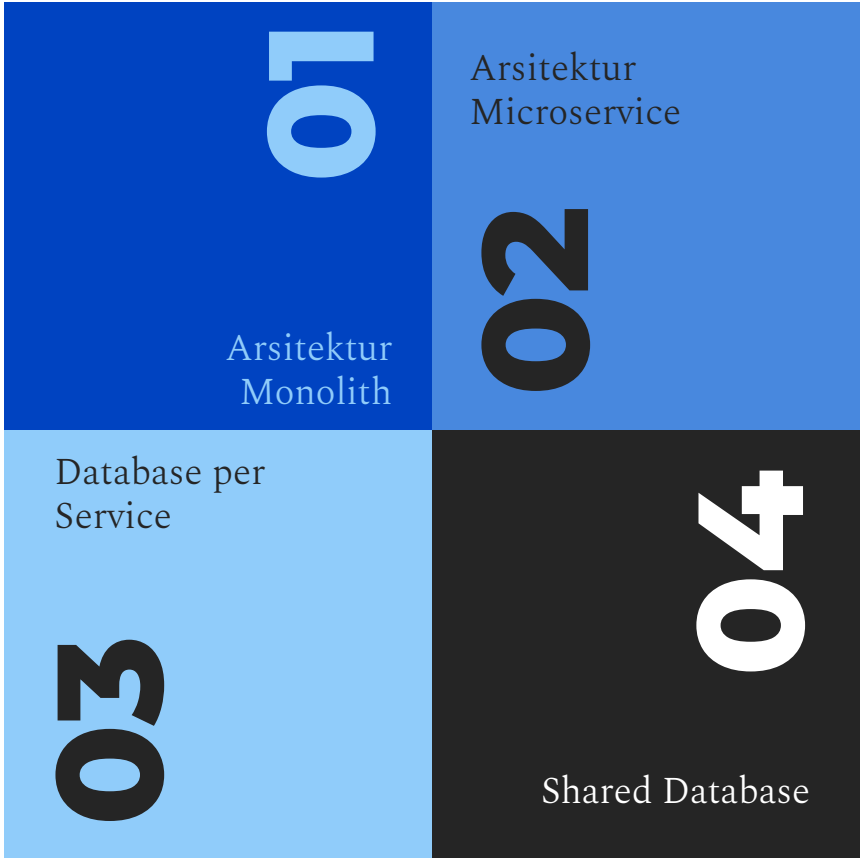


Microservices Architecture!

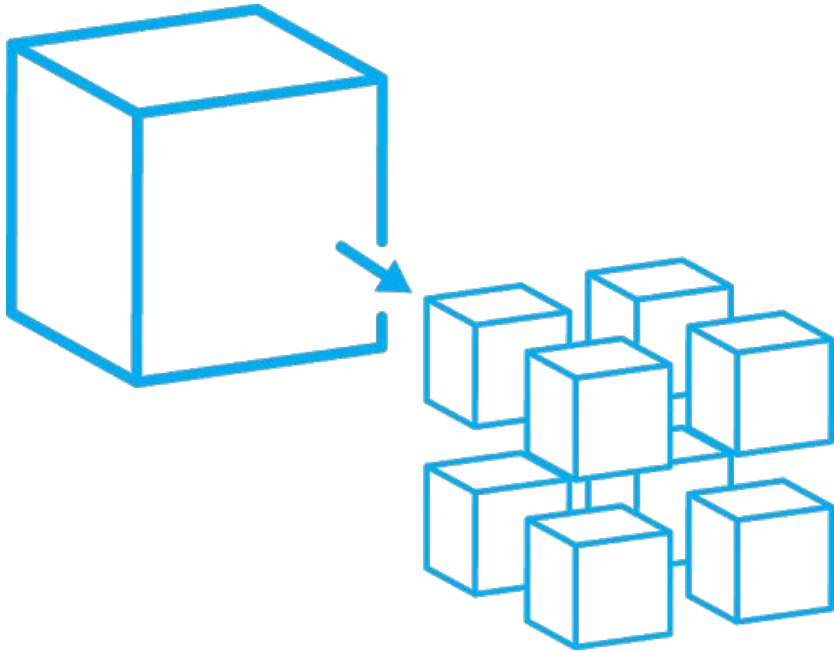
Robert C. Martin coined the term **single responsibility principle** which states “gather together those things that change for the same reason, and separate those things that change for different reasons.”


gojek
UBER
tokopedia
NETFLIX

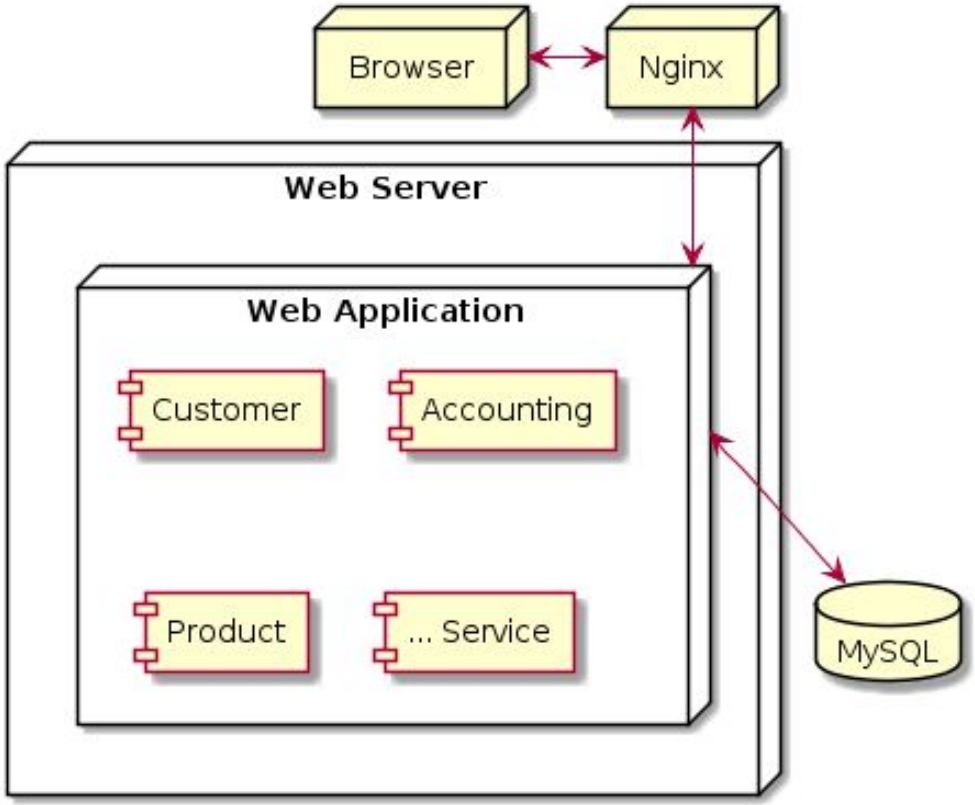

01
Arsitektur
Monolith
Arsitektur
Microservice
02
Database per
Service
03
04
Shared Database

Agenda

1. Arsitektur Monolith



Arsitektur Monolith



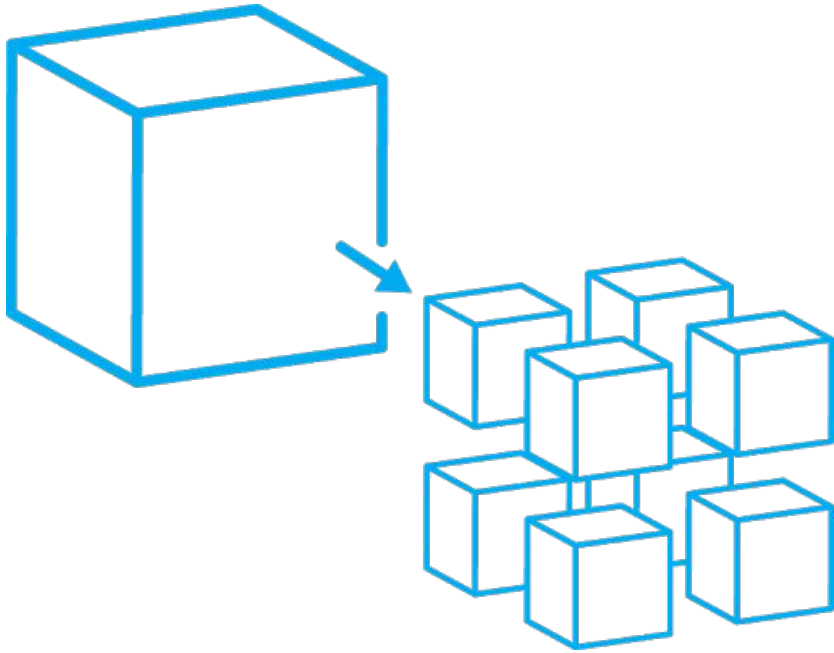
Kelebihan Arsitektur Monolith

- Mudah di Develop
- Mudah di Deploy
- Mudah di Test
- Mudah di Scale

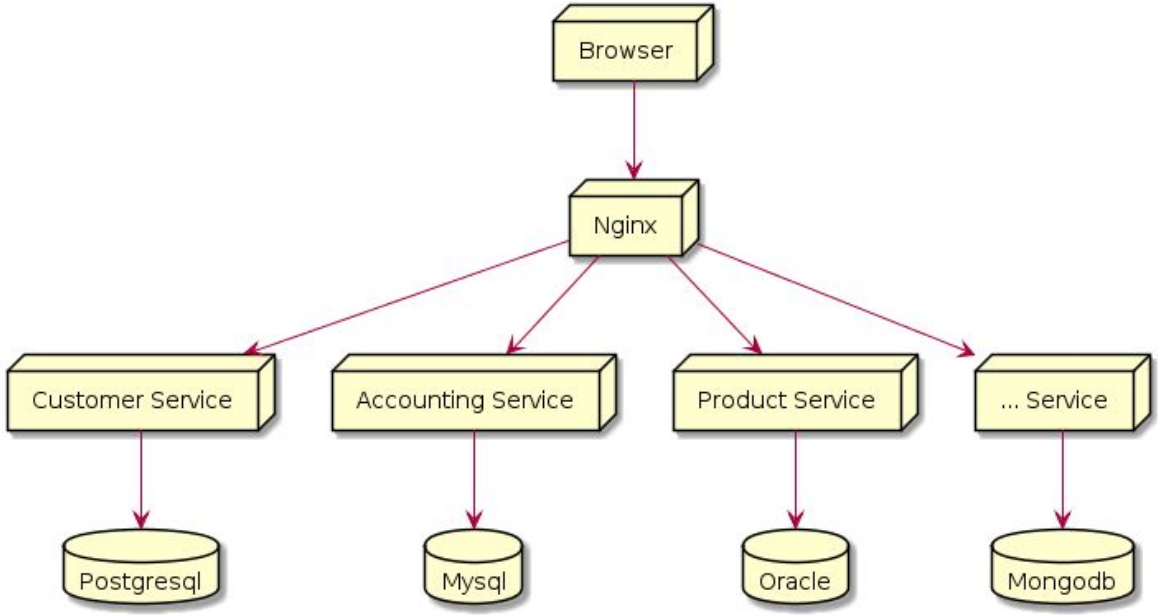
Masalah di Arsitektur Monolith

- Mengintimidasi Developer yang baru bergabung
- Scaling development dengan banyak Developer agak menyulitkan
- Butuh kontrak panjang dengan teknologi yang digunakan (bahasa pemrograman, database, dan lain-lain)
- Scaling pada bagian tertentu tidak bisa dilakukan
- Running app Monolith sangat berat

2. Arsitektur Microservice



Arsitektur Microservices



Apa itu Arsitektur Microservices

- Aplikasi-aplikasi kecil yang saling bekerja sama.
- Fokus mengerjakan satu pekerjaan dengan baik
- Independent, dapat di deploy dan diubah tanpa tergantung dengan aplikasi lain
- Setiap komponen pada sistem dibuat dalam service
- Komunikasi antar service biasanya melalui network-call

Kelebihan Arsitektur Microservices

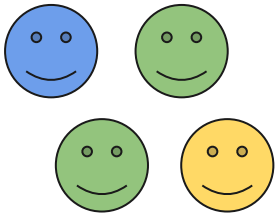
- Mudah dimengerti, karena relative kecil ukuran service nya
- Lebih mudah di develop, di maintain, di test dan di deploy
- Lebih mudah bergonta-ganti teknologi
- Mudah di scale sesuai kebutuhan
- Bisa dikerjakan dalam tim-tim kecil

Masalah di Arsitektur Microservices

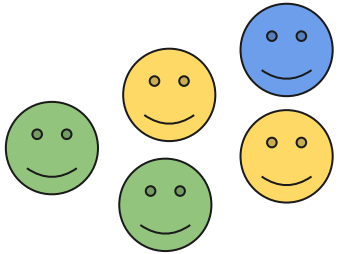
- Distributed system
- Komunikasi antar service yang rawan error
- Testing interaksi antar service lebih sulit

Pembagian Aplikasi Microservices

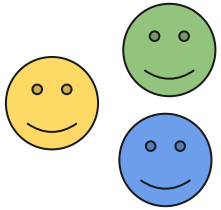
Merchant



Product



Shipping



Seberapa Kecil Aplikasi Microservices?

- Single responsibility
- Sekecil mungkin sehingga bisa dimengerti oleh satu orang
- Bisa di kerjakan sejumlah X developer

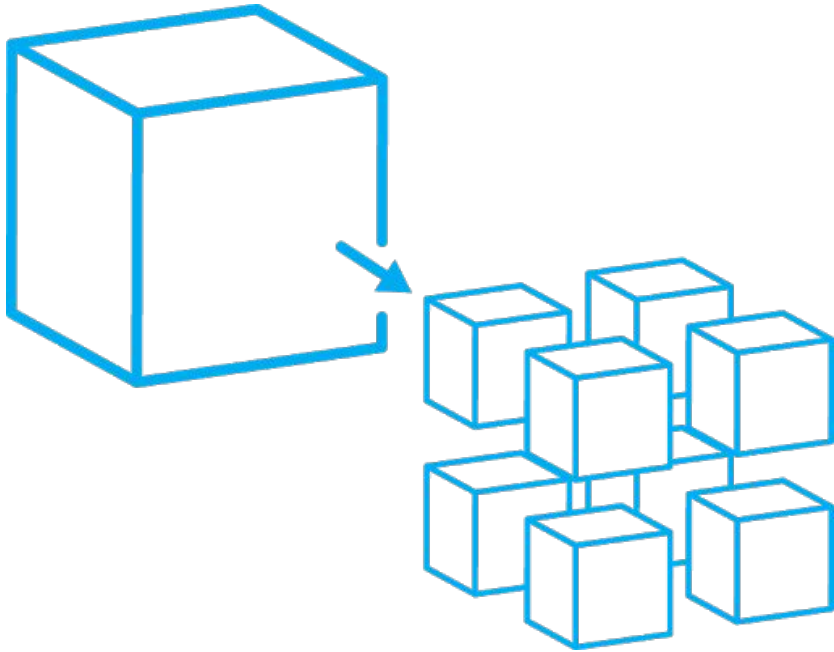
Monolith

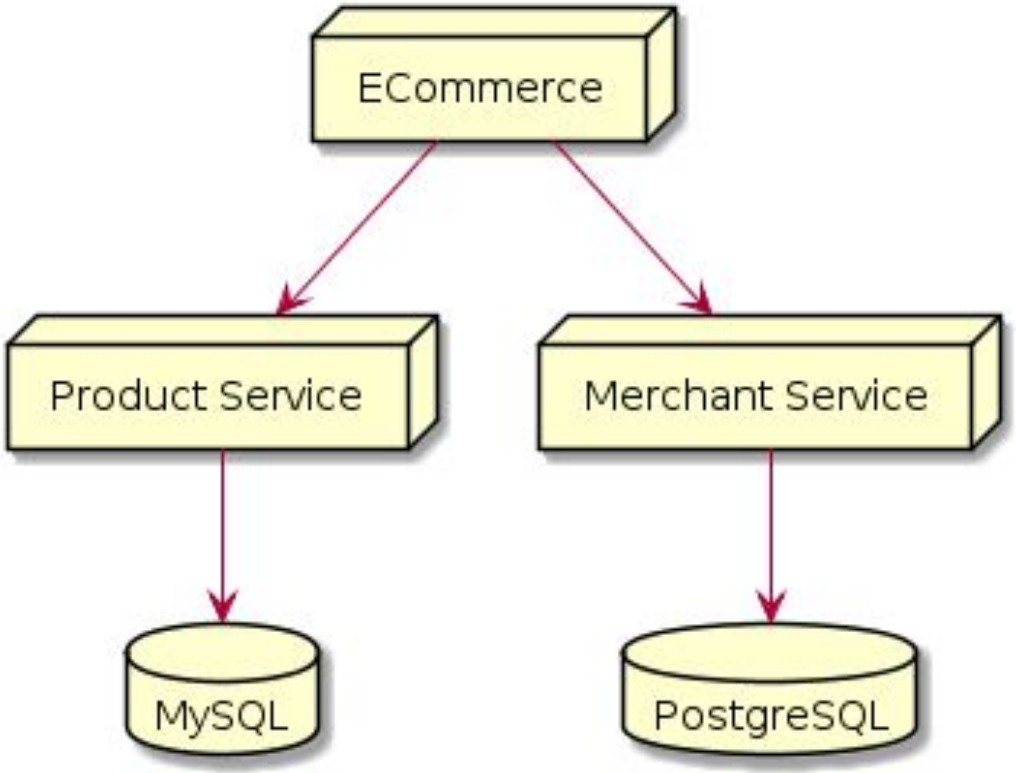
- Simplicity
- Consistency
- Easy to Refactor

Microservices

- Partial Deployment
- Availability
- Multiple Platform
- Easy to Scale

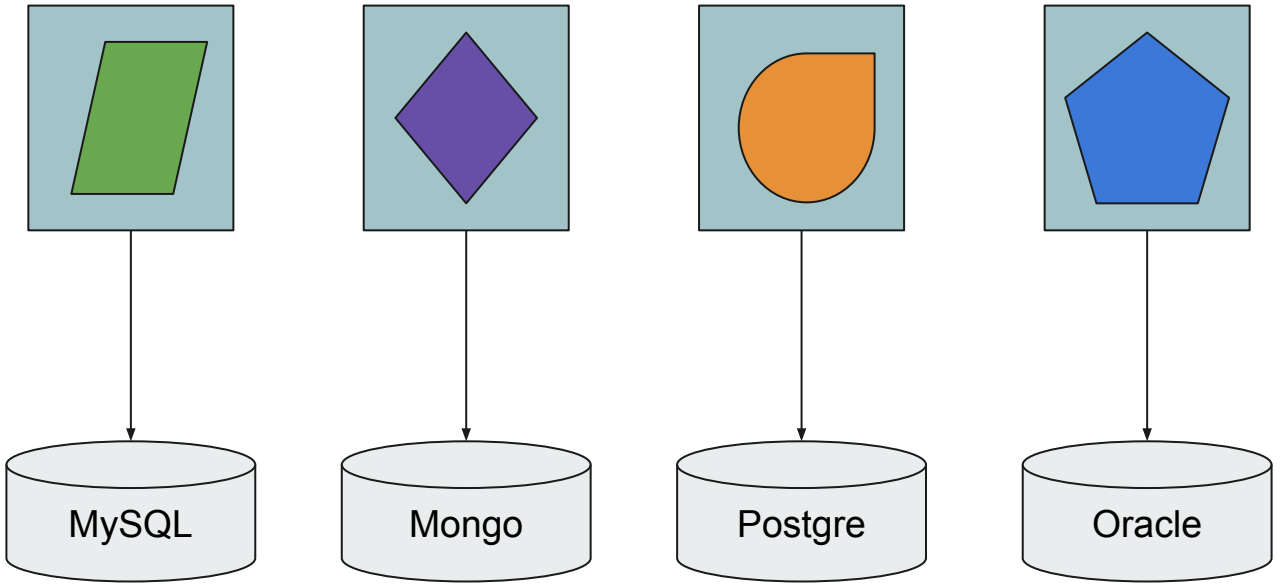
3. Database per Service





Contoh Database per Service

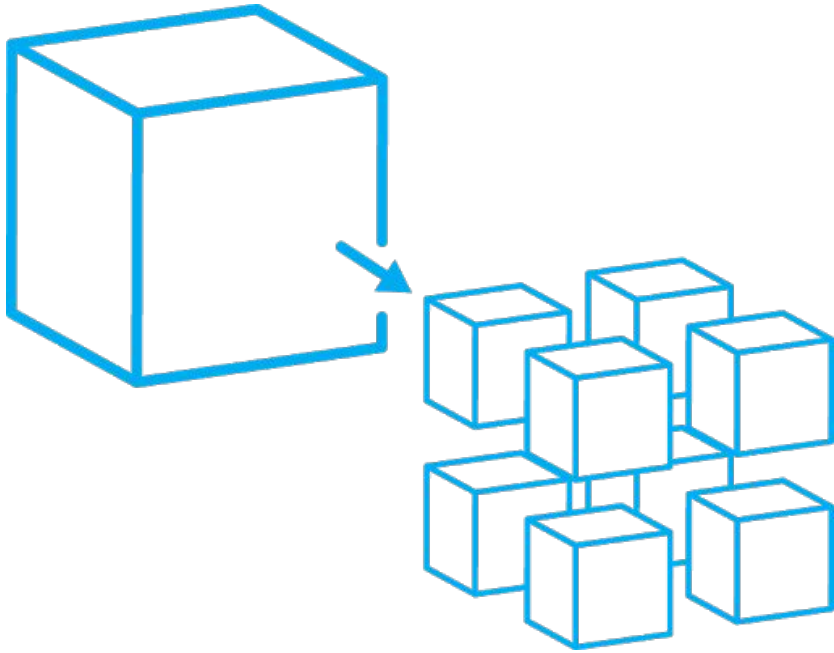
Decentralized Database



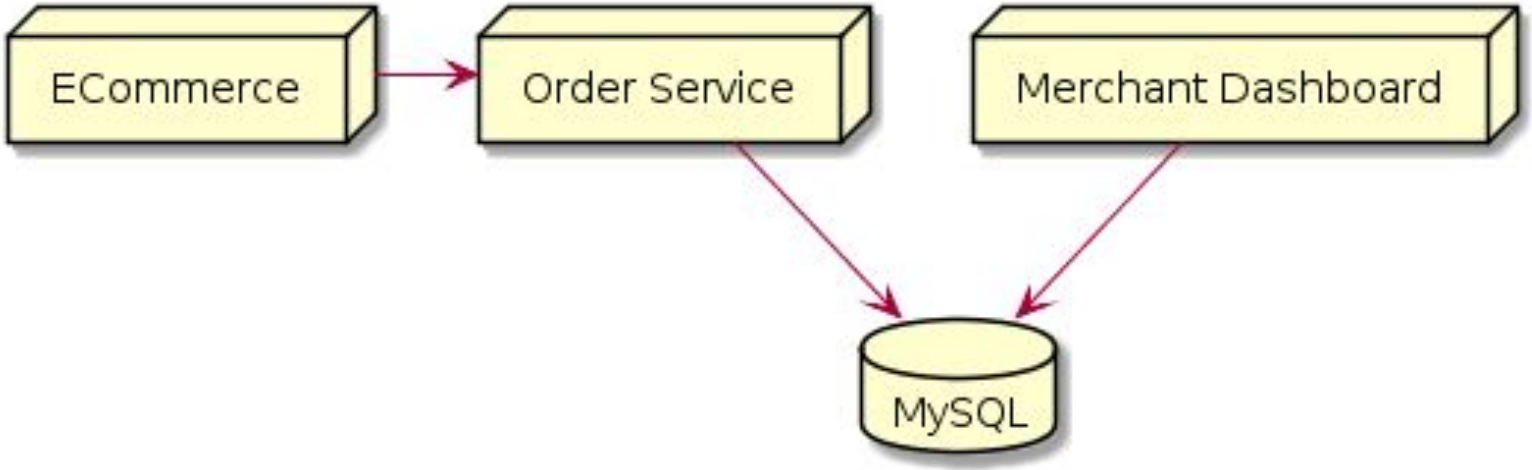
Kenapa Harus Database per Service?

- Memastikan bahwa antar service tidak ketergantungan
- Tiap service bisa menggunakan aplikasi database sesuai dengan kebutuhan
- Service tidak perlu tahu kompleksitas internal database service lain

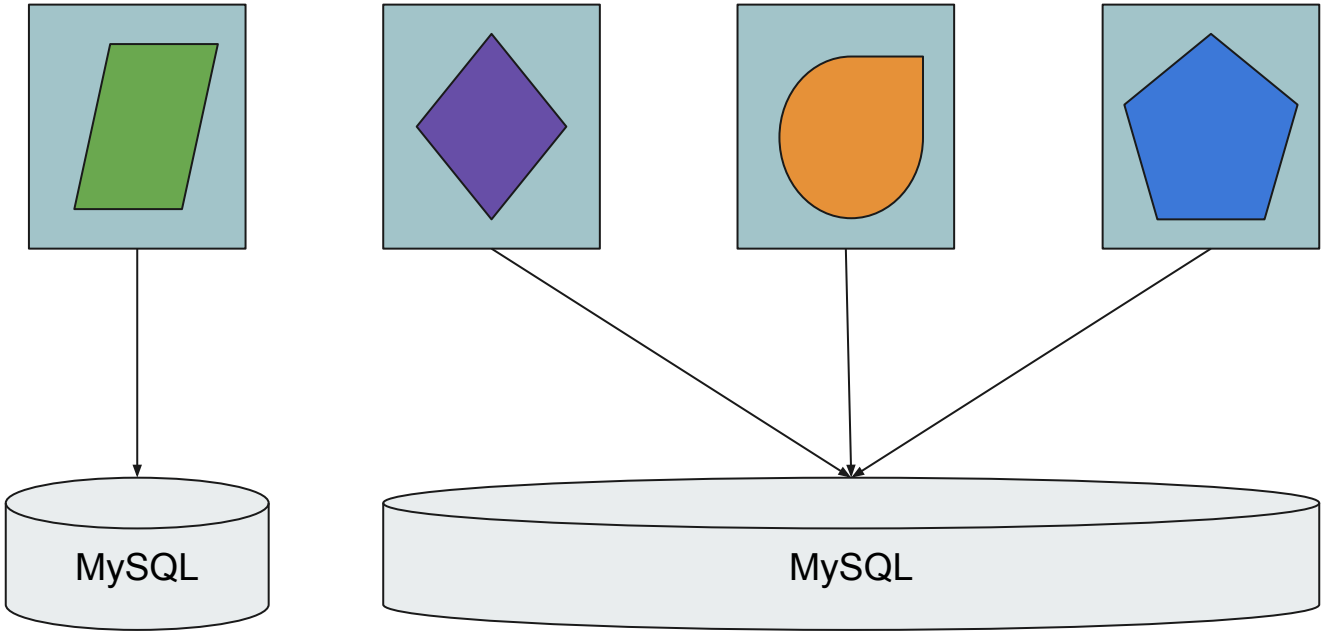
4. Shared Database



Contoh Shared Database

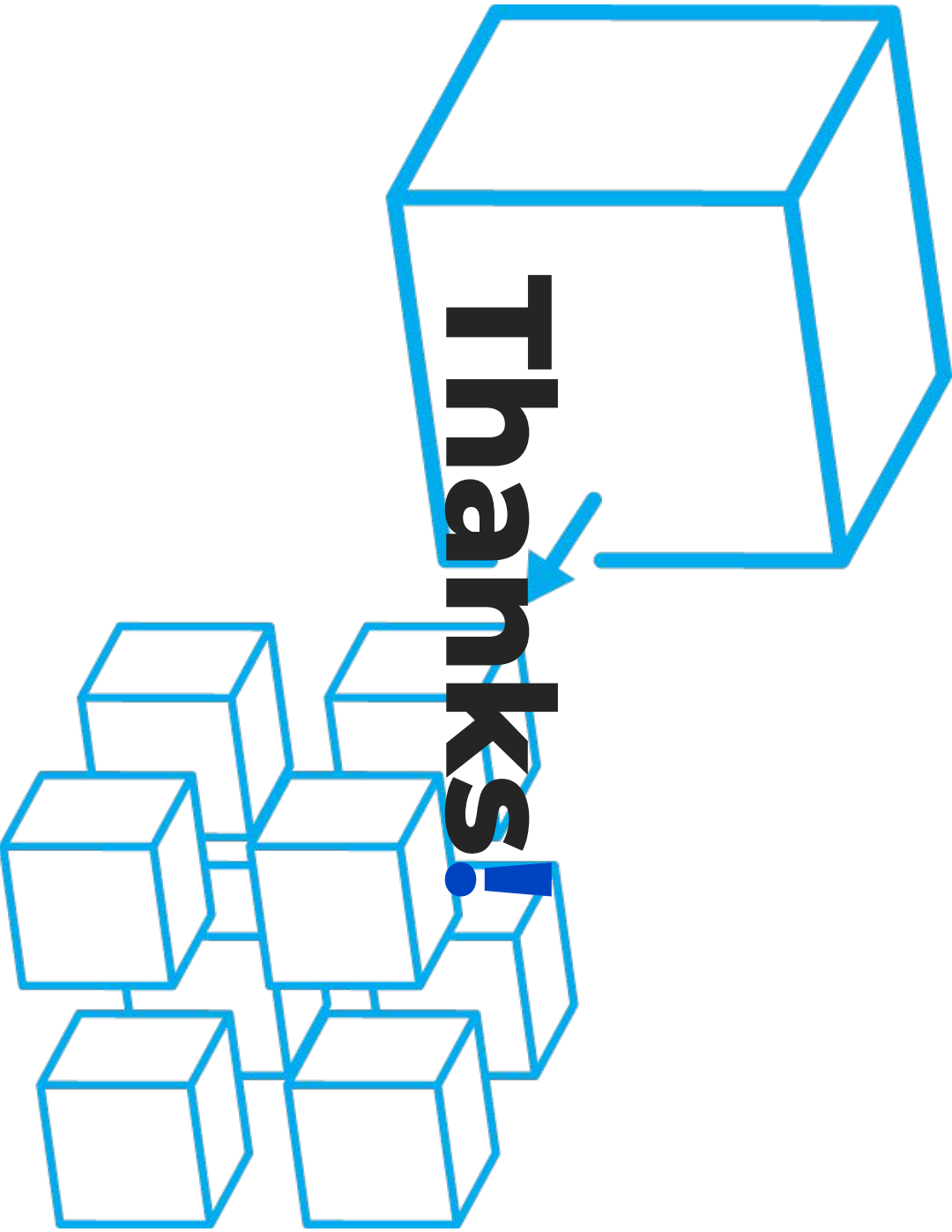


Shared Database



Kapan Harus Shared Database?

- Ketika melakukan transisi dari aplikasi Monolith ke Microservices
- Ketika bingung memecahkan data antar Service
- Ketika dikejar waktu, sehingga tidak ada waktu untuk bikin API


Thanks!