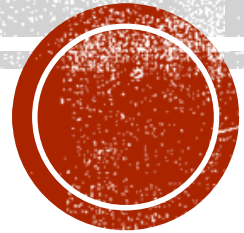
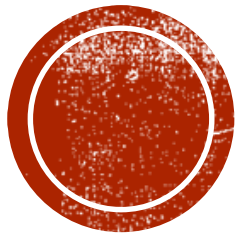
РАЗВОЈ СОФТВЕРА 2

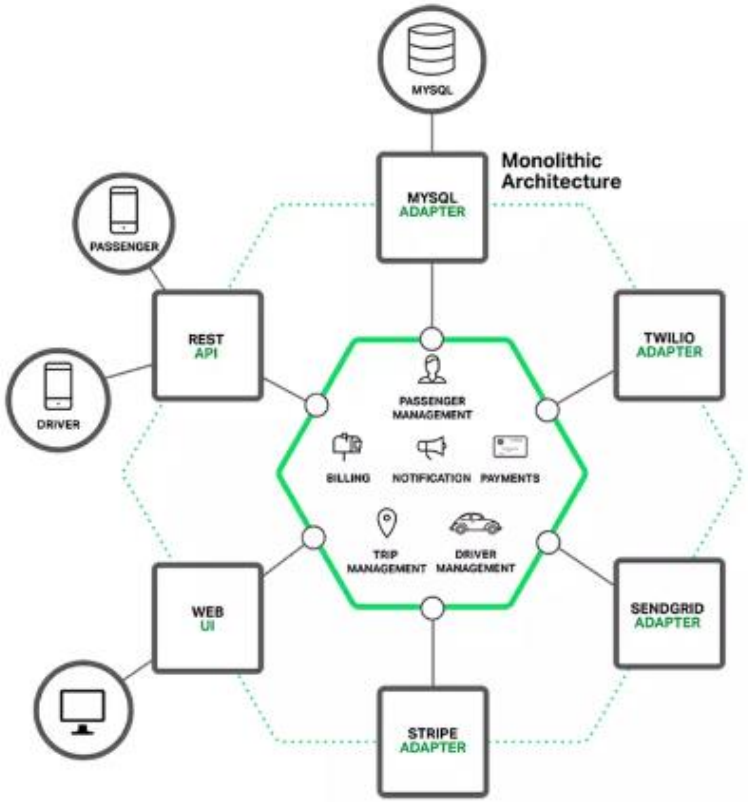
Микросервиси





ДЕФИНИЦИЈА МИКРОСЕРВИСА

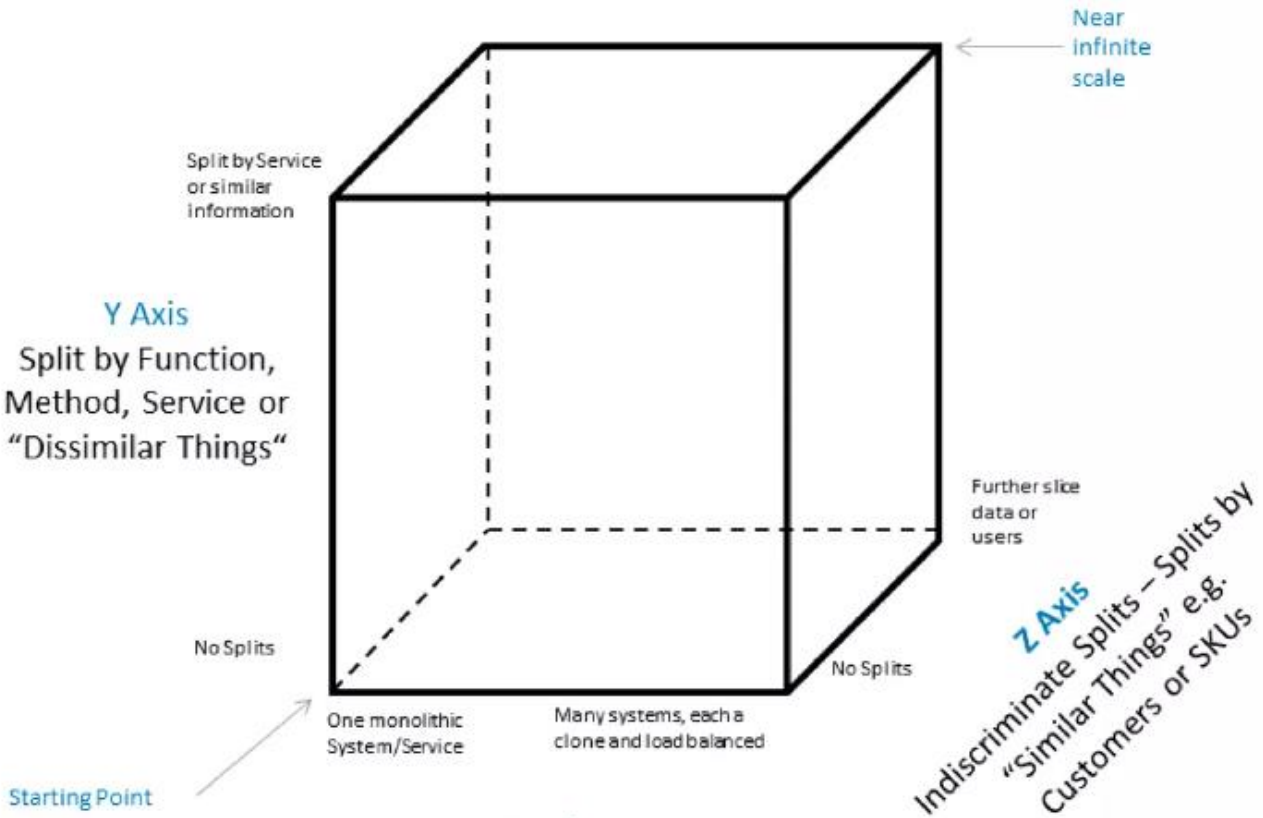
The monolith

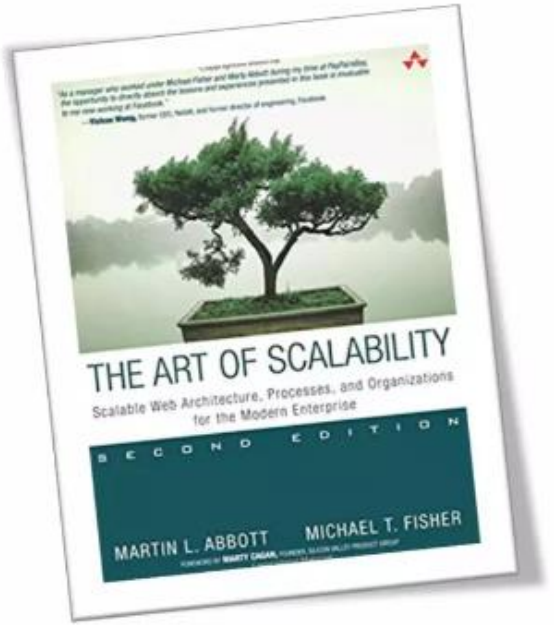


- Too big to understand
- Changes in one area require the full build & delivery
- Long build, deploy & startup times
- Changes in one area have to wait for other areas to be ready in order to be available
- Typically highly coupled modules
- Difficult to scale

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

AKF Scale Cube

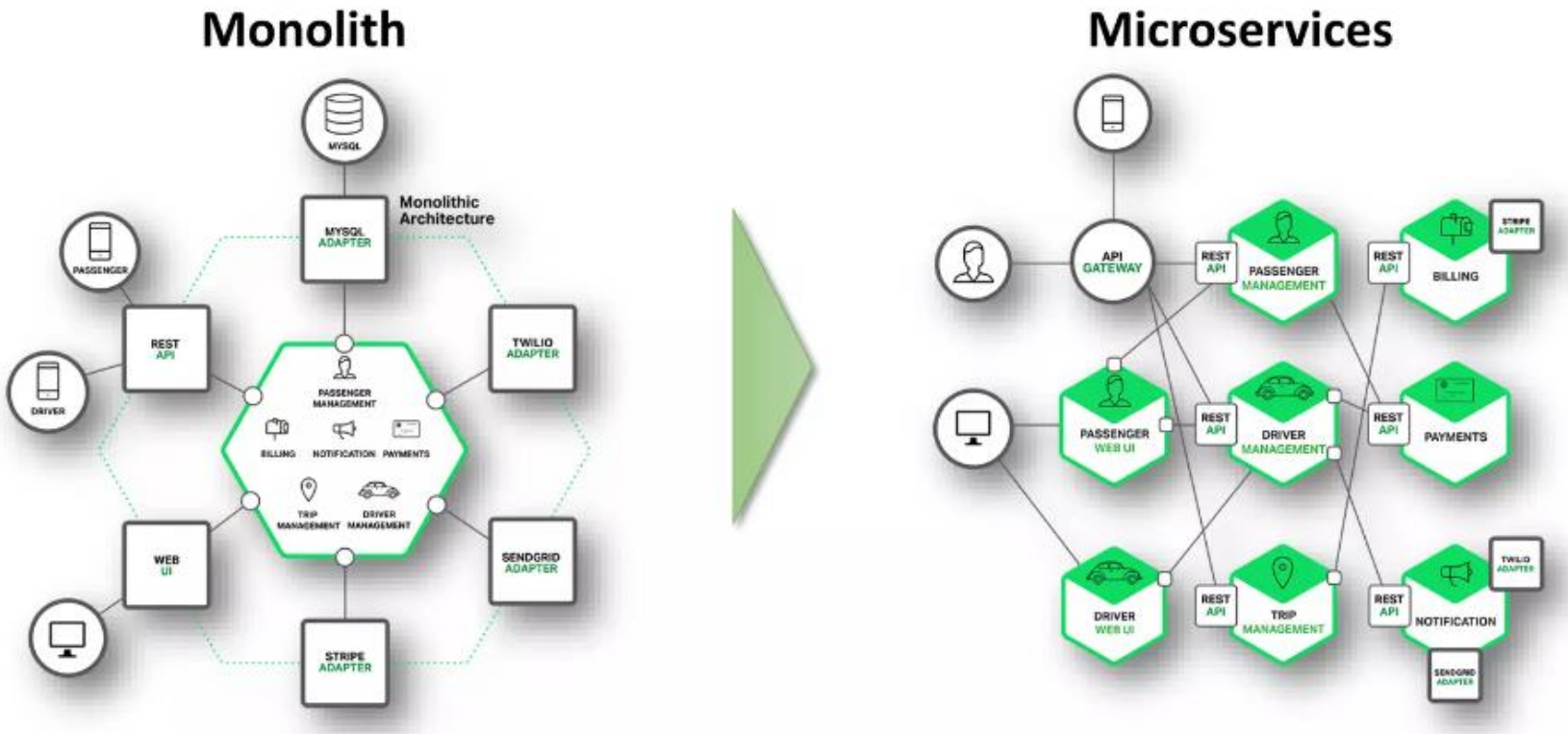




Martin Abbott, Michael Fisher (2015) **The Art of Scalability, The: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise** 2nd Edition. Addison-Wesley Professional

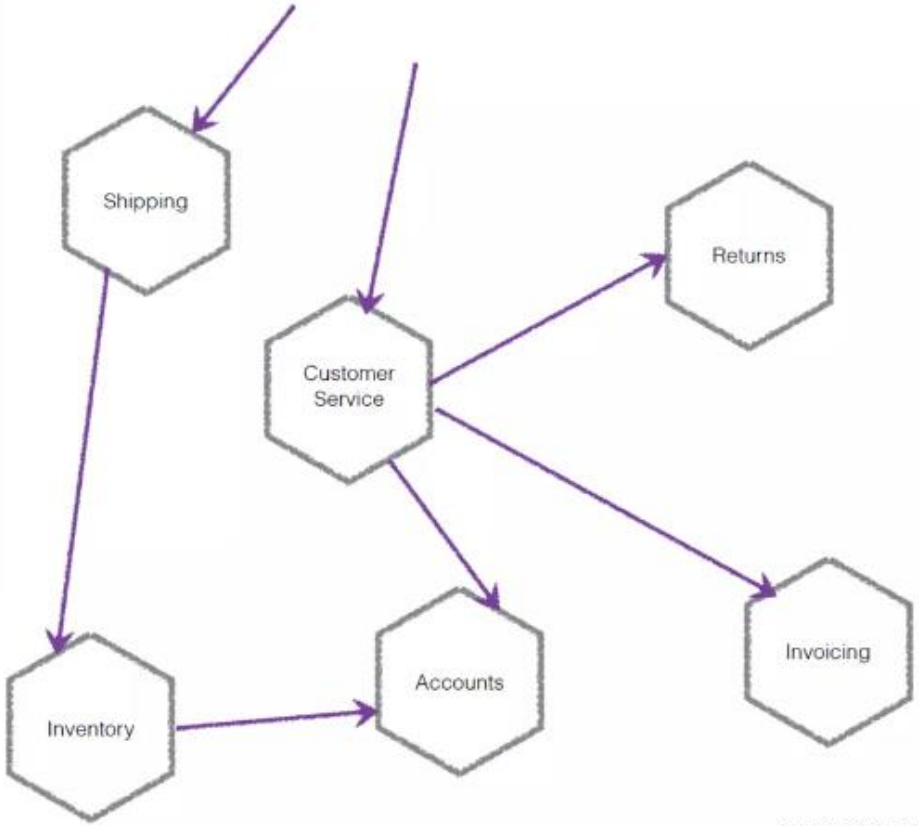
https://akfpartners.com/techblog/2008/05/08/splitting-applications-or-services-for-scale/

Tackling the complexity



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

Small ***independently deployable*** *services that* ***work together****, modelled around a* ***business domain***



SMALL

Size is not the actual point!

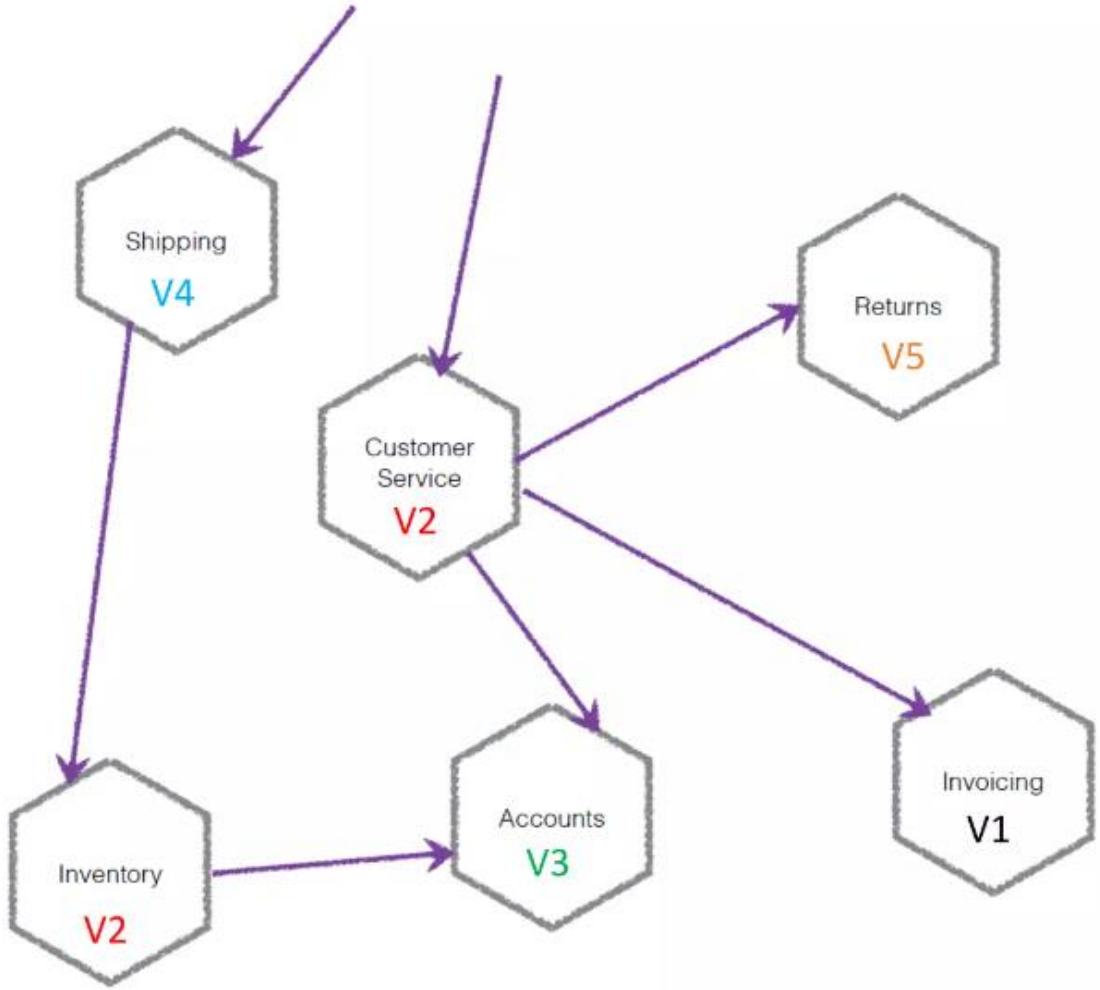
not as big as a server app that needs to be built and deployed as a single block

Manageable units of functionality and deployability

INDEPENDENTLY DEPLOYABLE

***No lock-step** build and deployment*

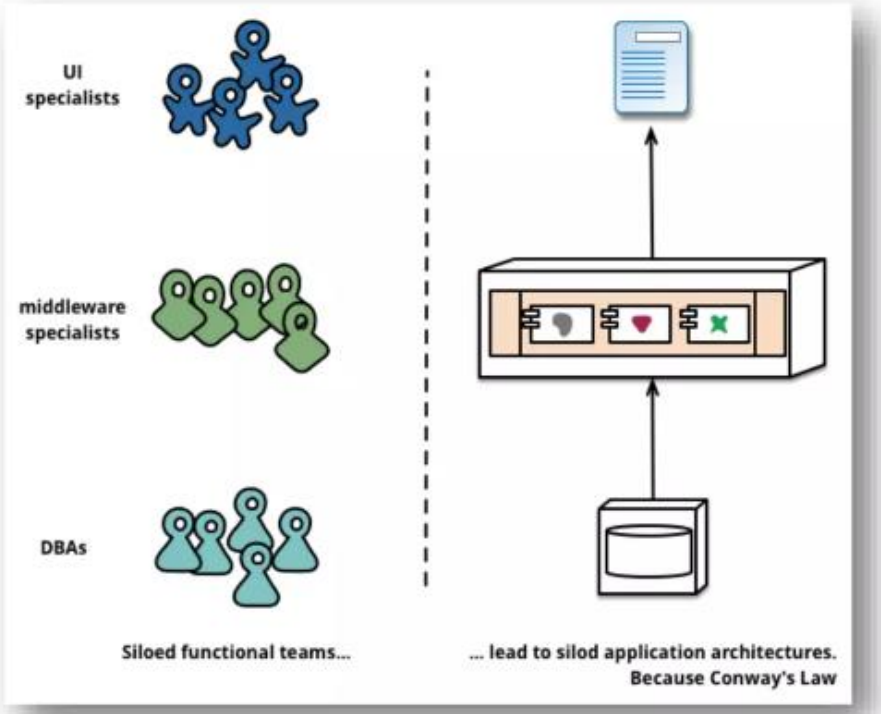
Avoiding the "Distributed Monolith"

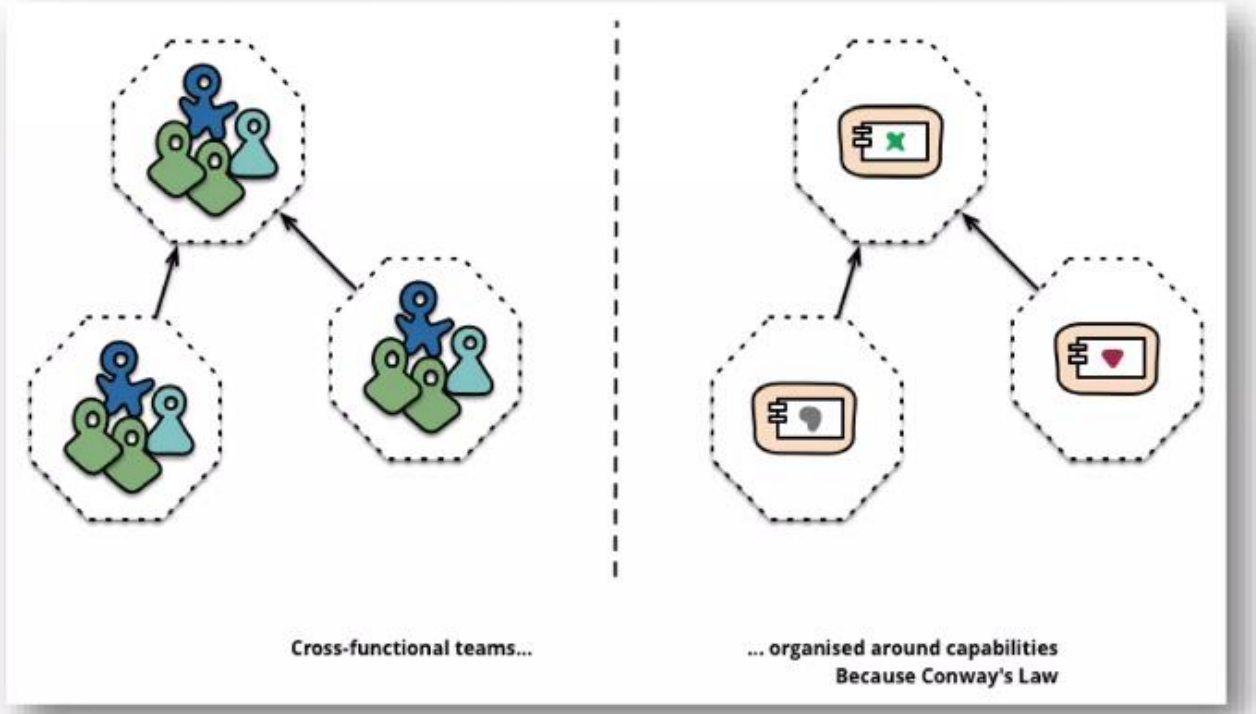


WORK TOGETHER

Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.

-- Melvyn Conway, 1967

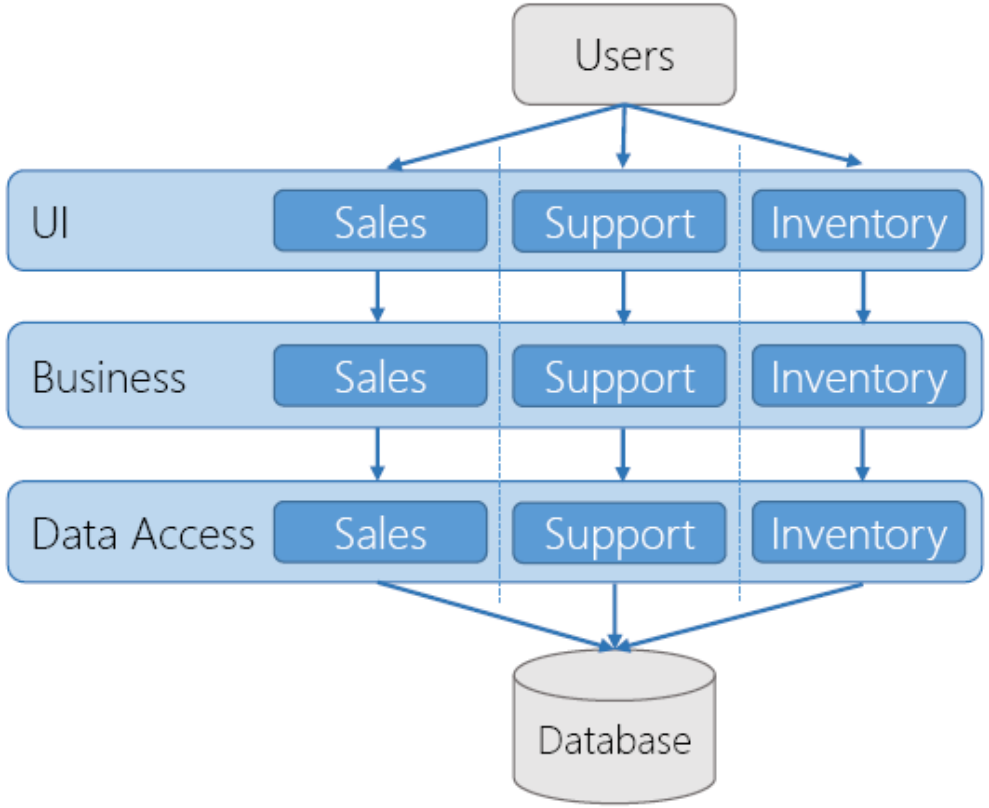




https://martinfowler.com/articles/microservices.html

ПРИМЕР ПОСЛОВНОГ ДОМЕНА

Components


Users
UI
Sales
Support
Inventory
Business
Sales
Support
Inventory
Data Access
Sales
Support
Inventory
Database

Problem Domain

Sales

Sales Opportunity

Contact

Sales Person

Product

Sales Territory

Support

Support Ticket

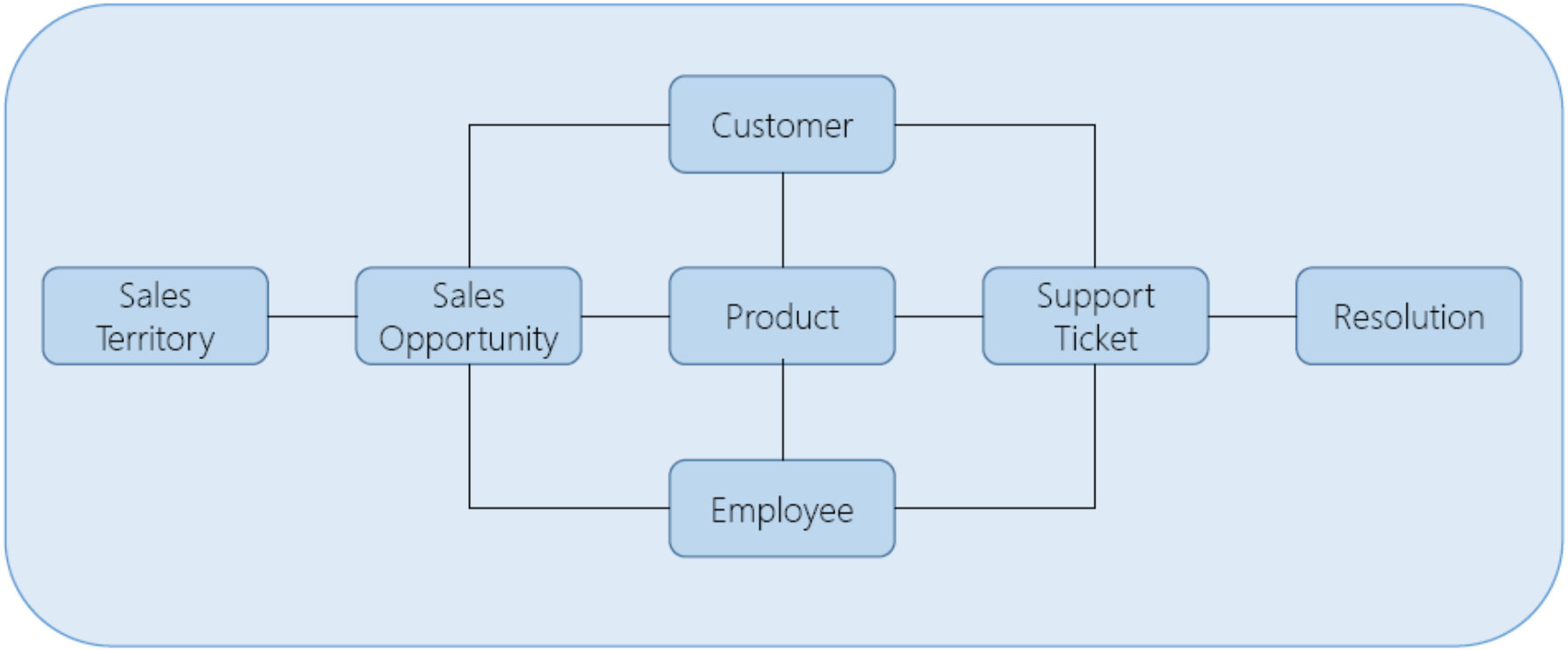
Customer

Support Person

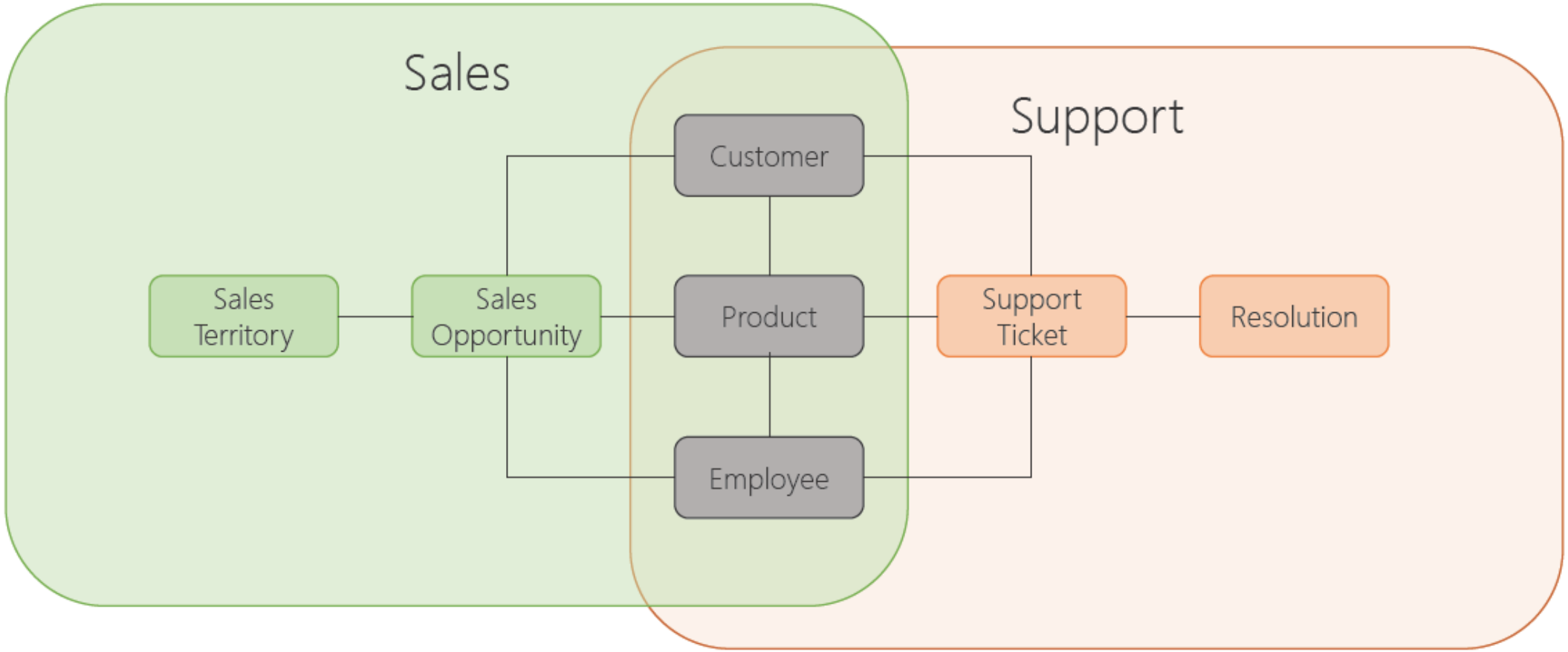
Product

Resolution

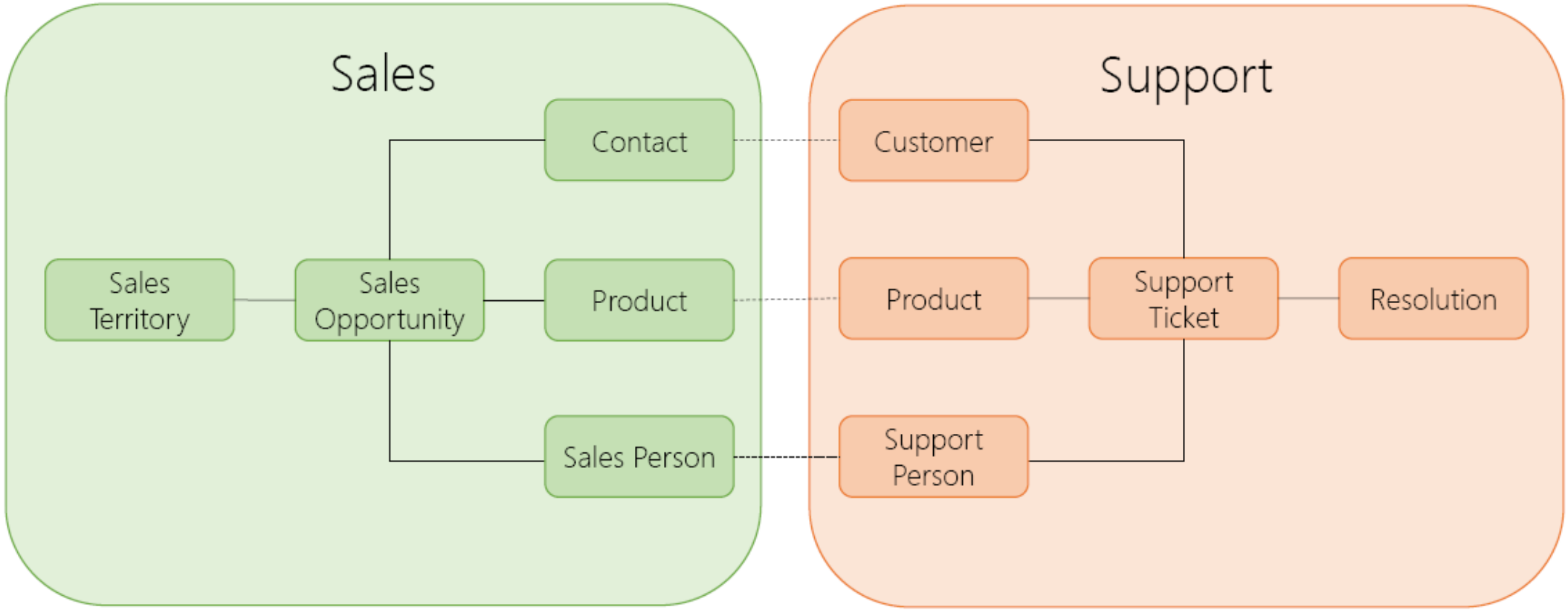
Single Domain Model

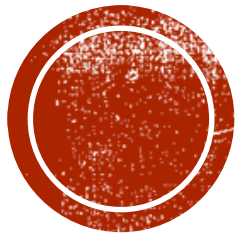


Overlapping Contexts



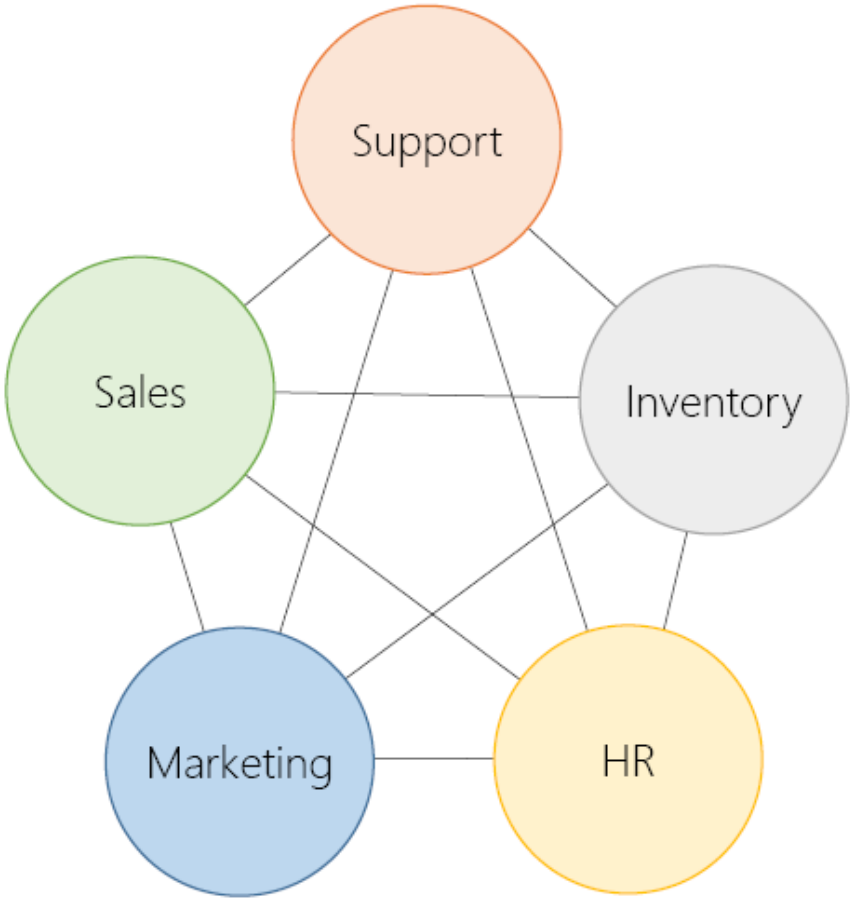
Bounded Contexts



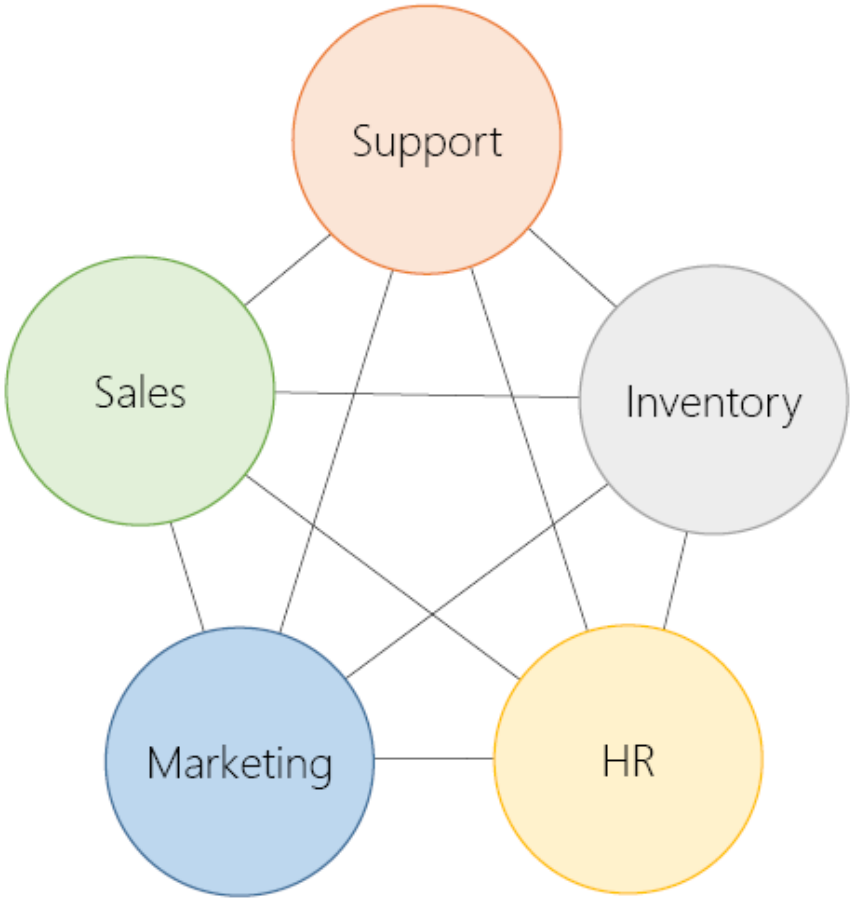


МИКРОСЕРВИСНА АРХИТЕКТУРА ЗА ДАТИ ПОСЛОВНИ ДОМЕН

Microservice Architectures



Microservice Architectures

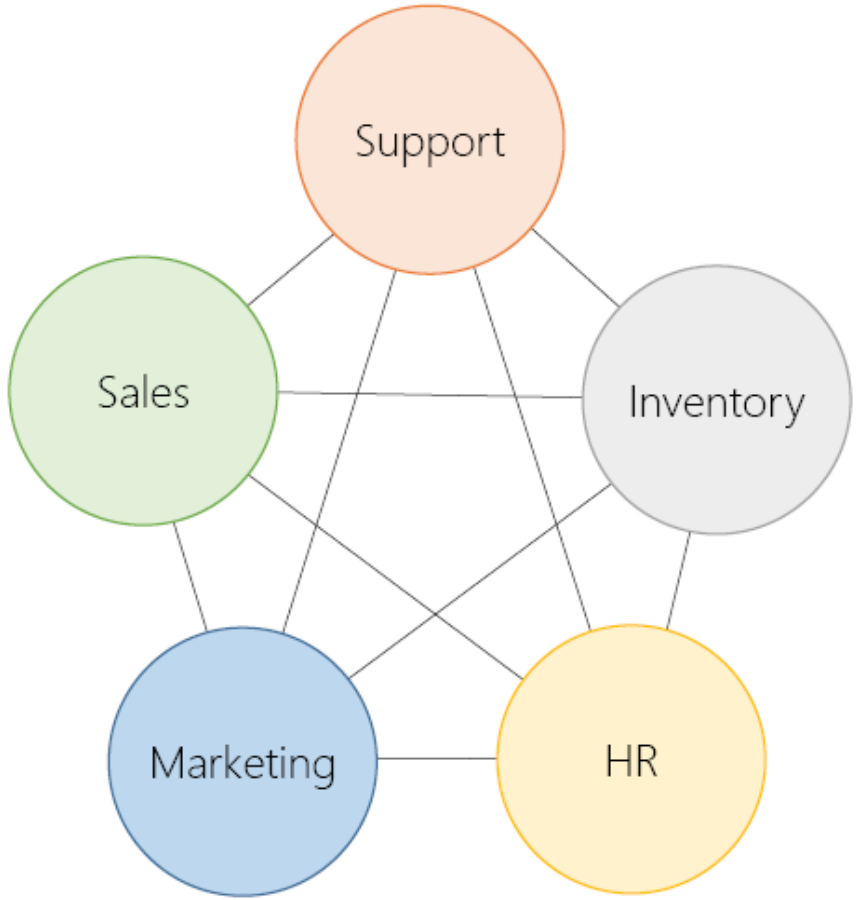


Microservice Architectures

Subdivide system

Light-weight APIs

Small teams

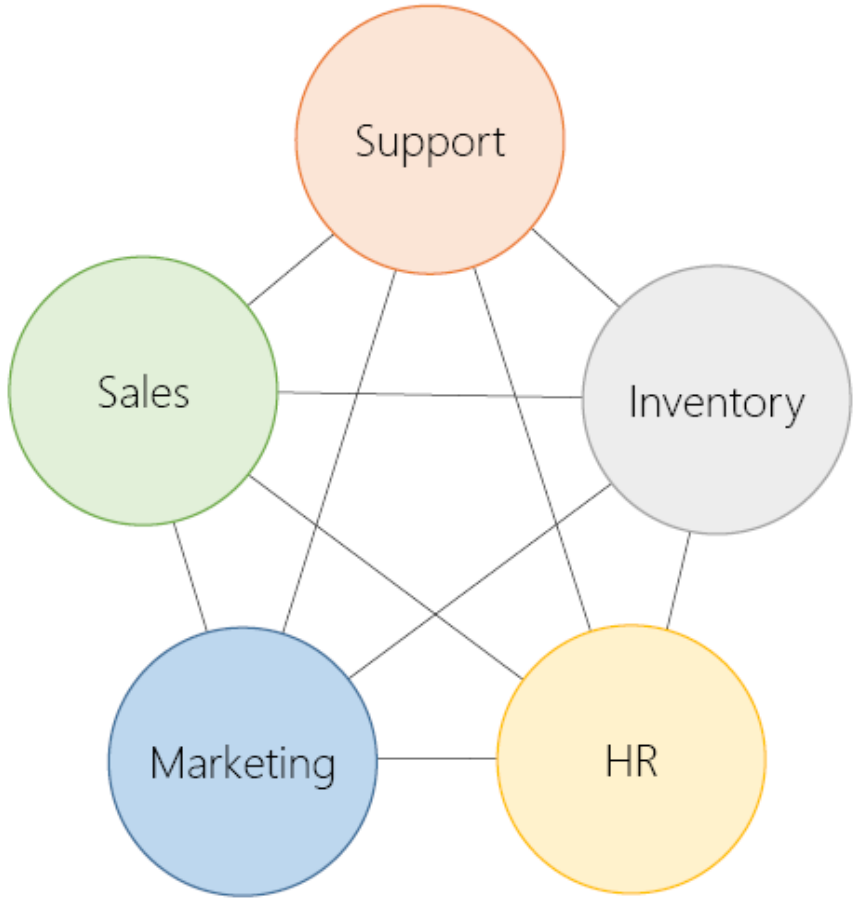


Microservice Architectures

Independent

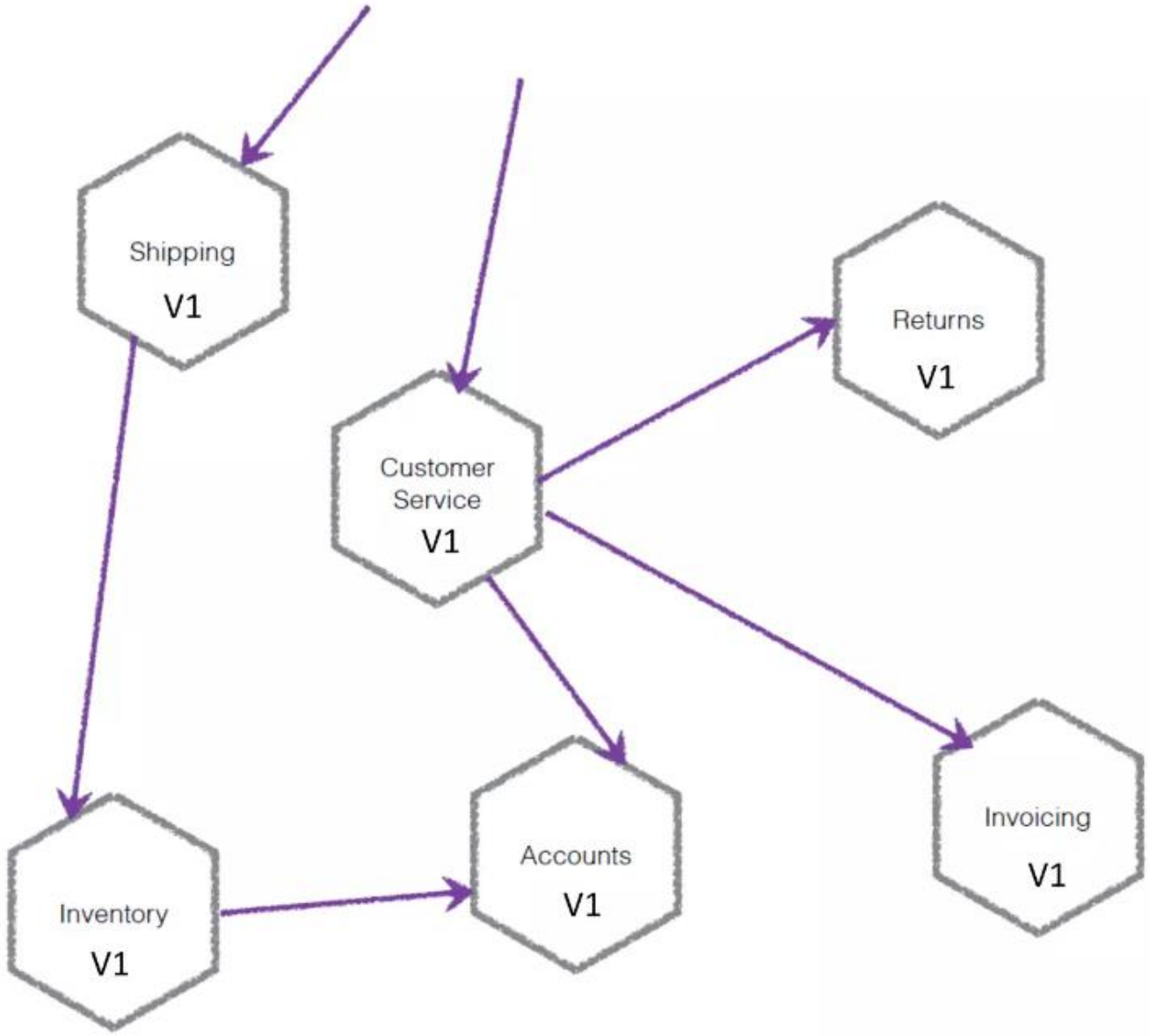
Similar to SOA

Size matters



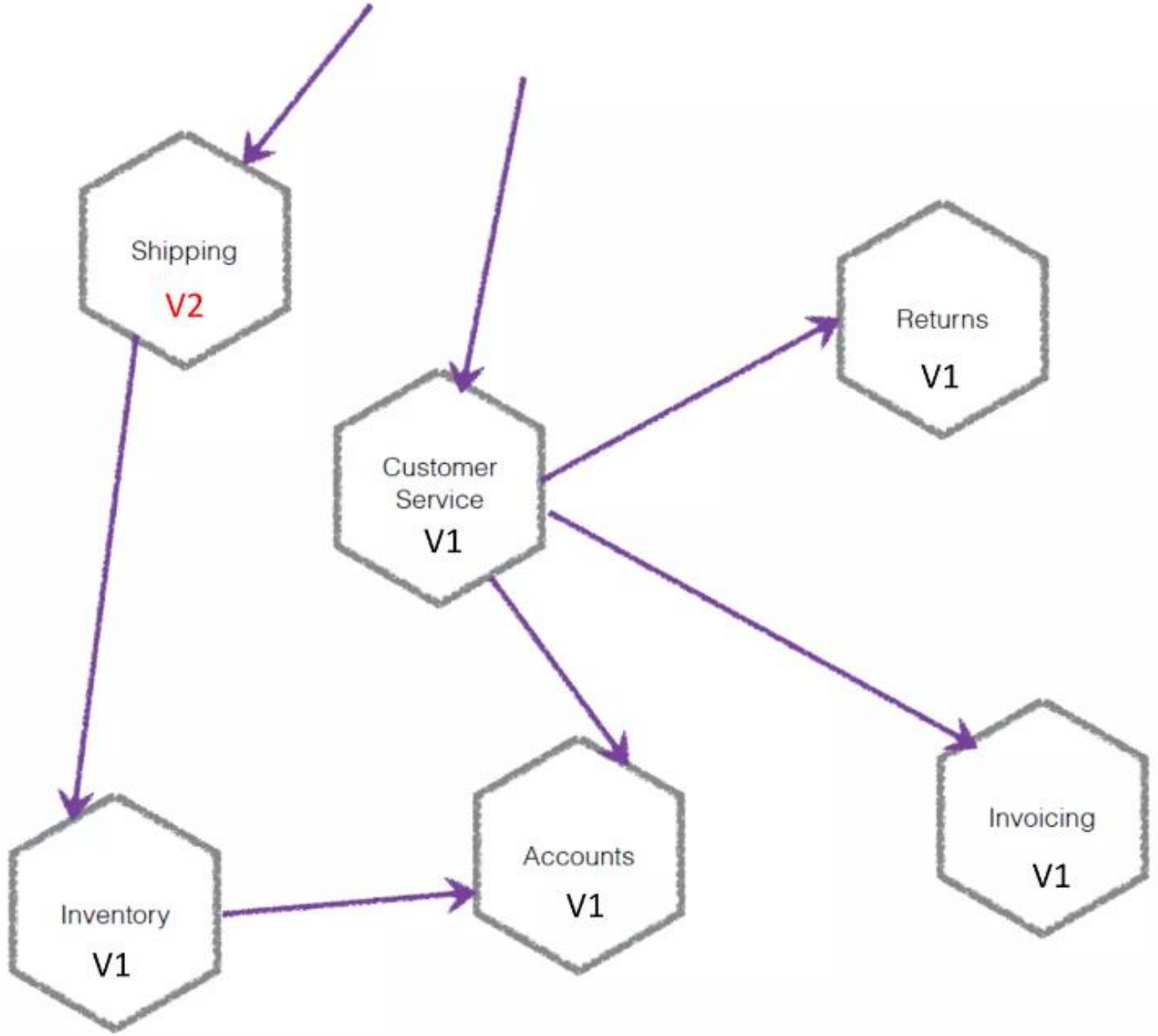
ПРЕДНОСТИ И МАНЕ МИКРОСЕРВИСНЕ АРХИТЕКТУРЕ

In the begining everything works fine

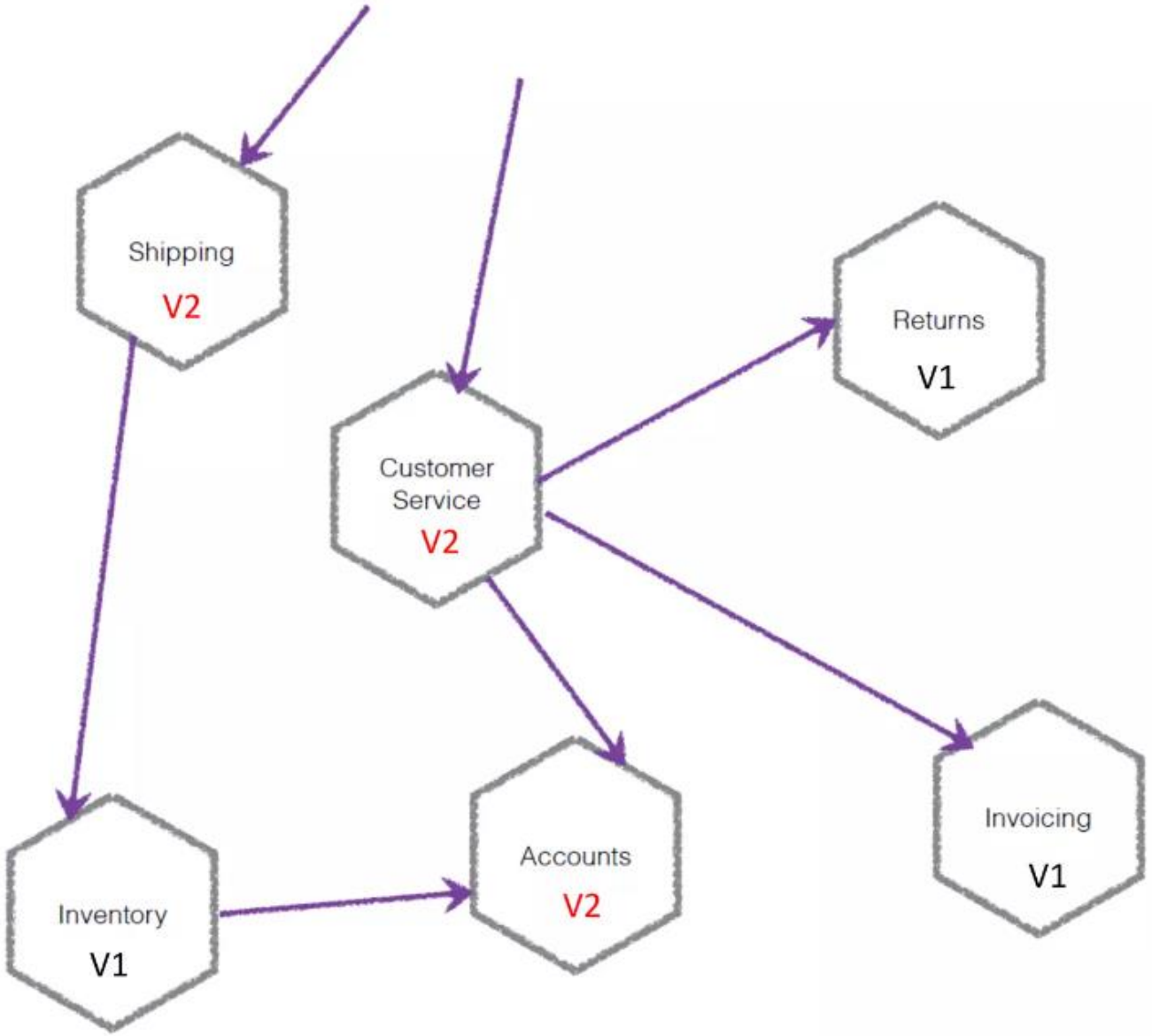


But 💩 happens.

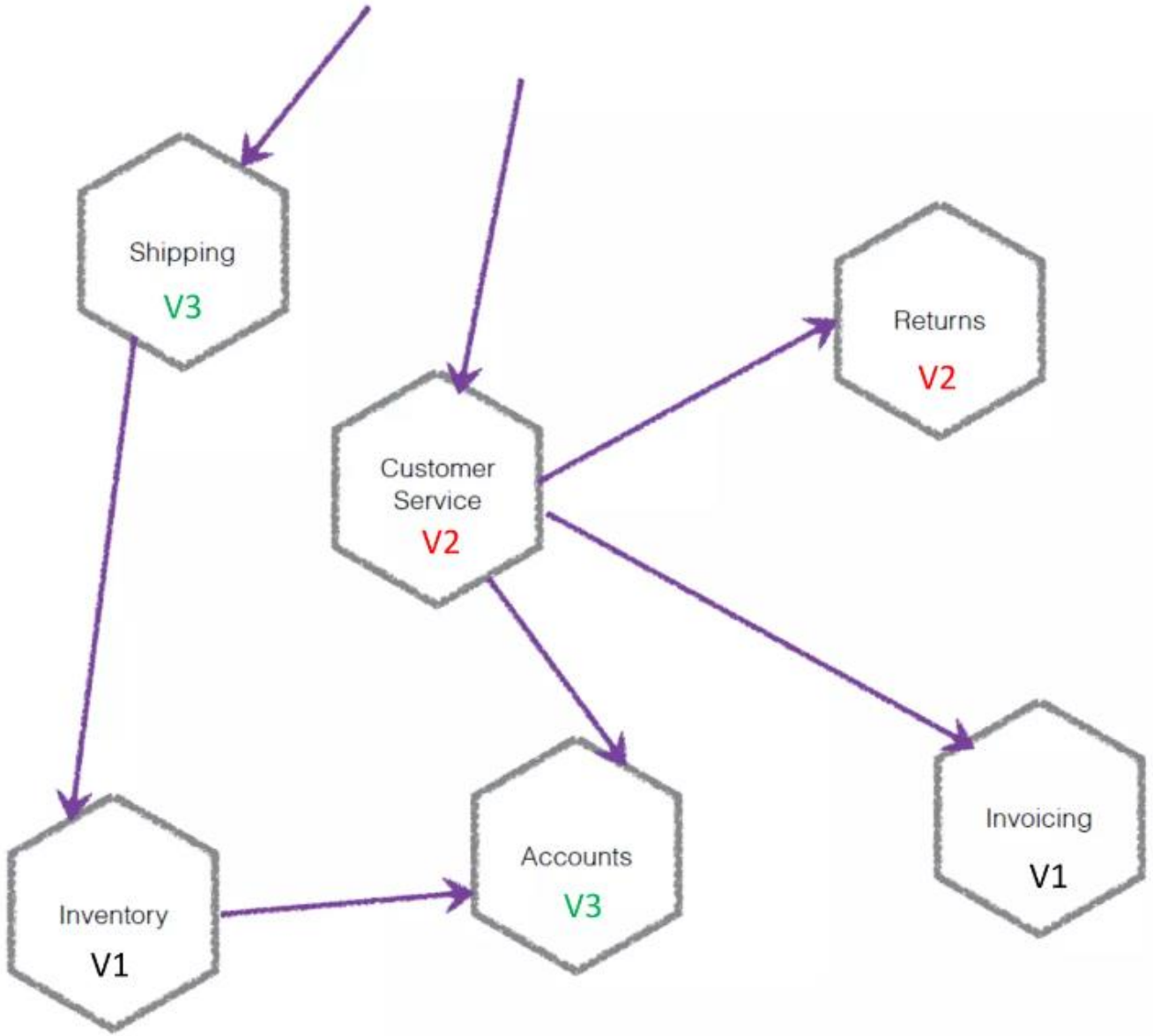
Fortunatelly, no one else needs to know about it.



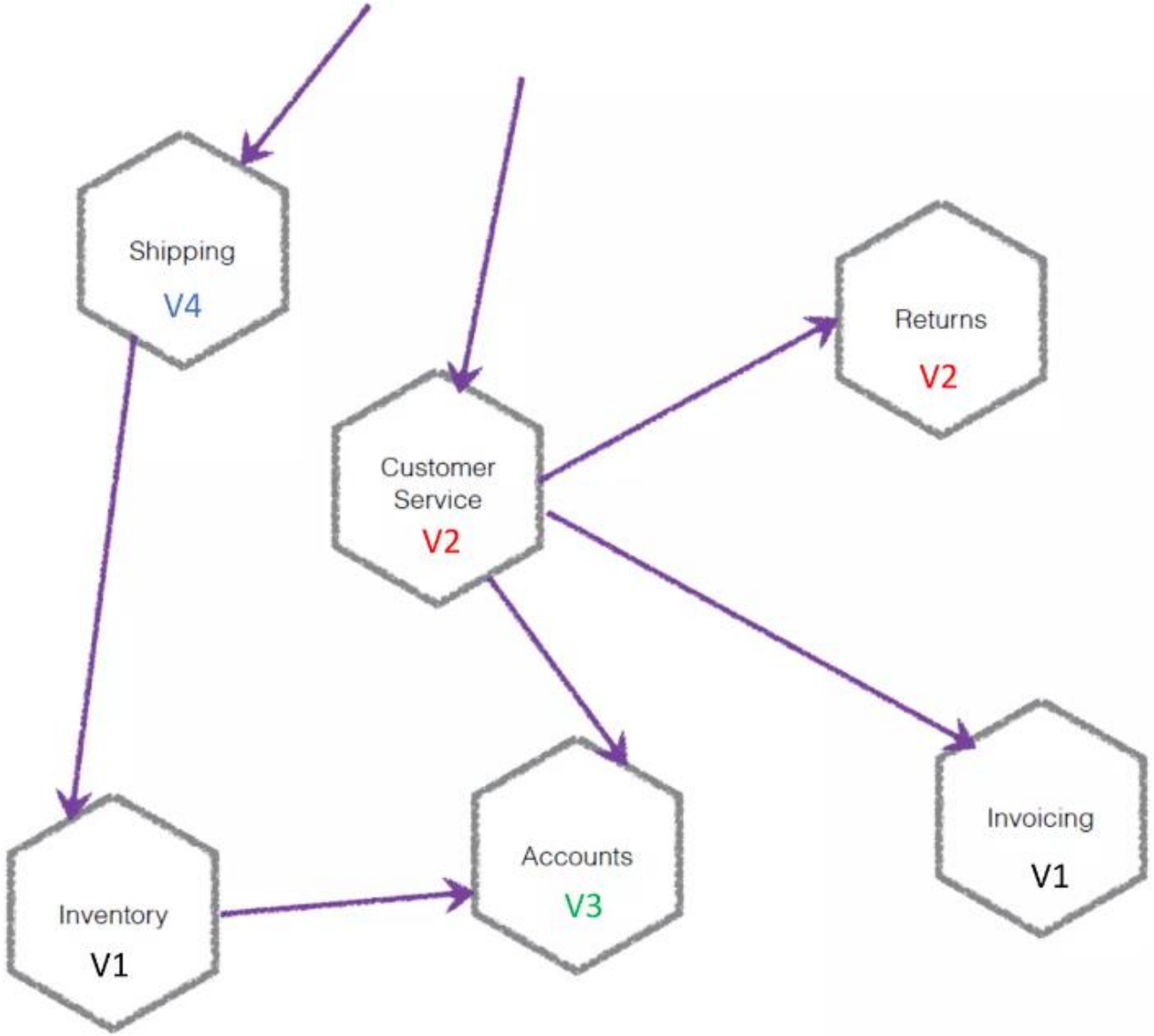
And it happens again



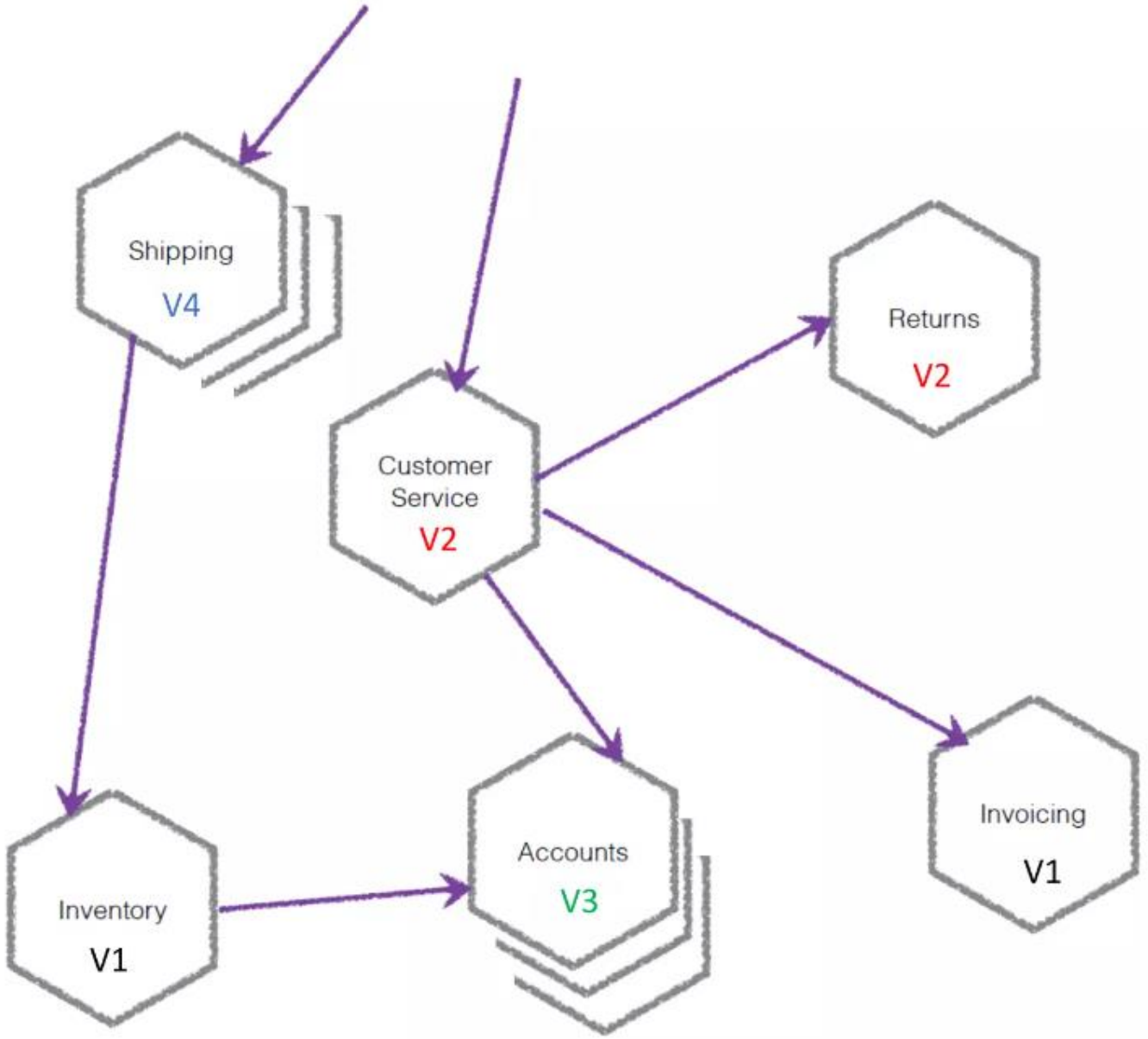
And we also want to refactor and update the technology stack



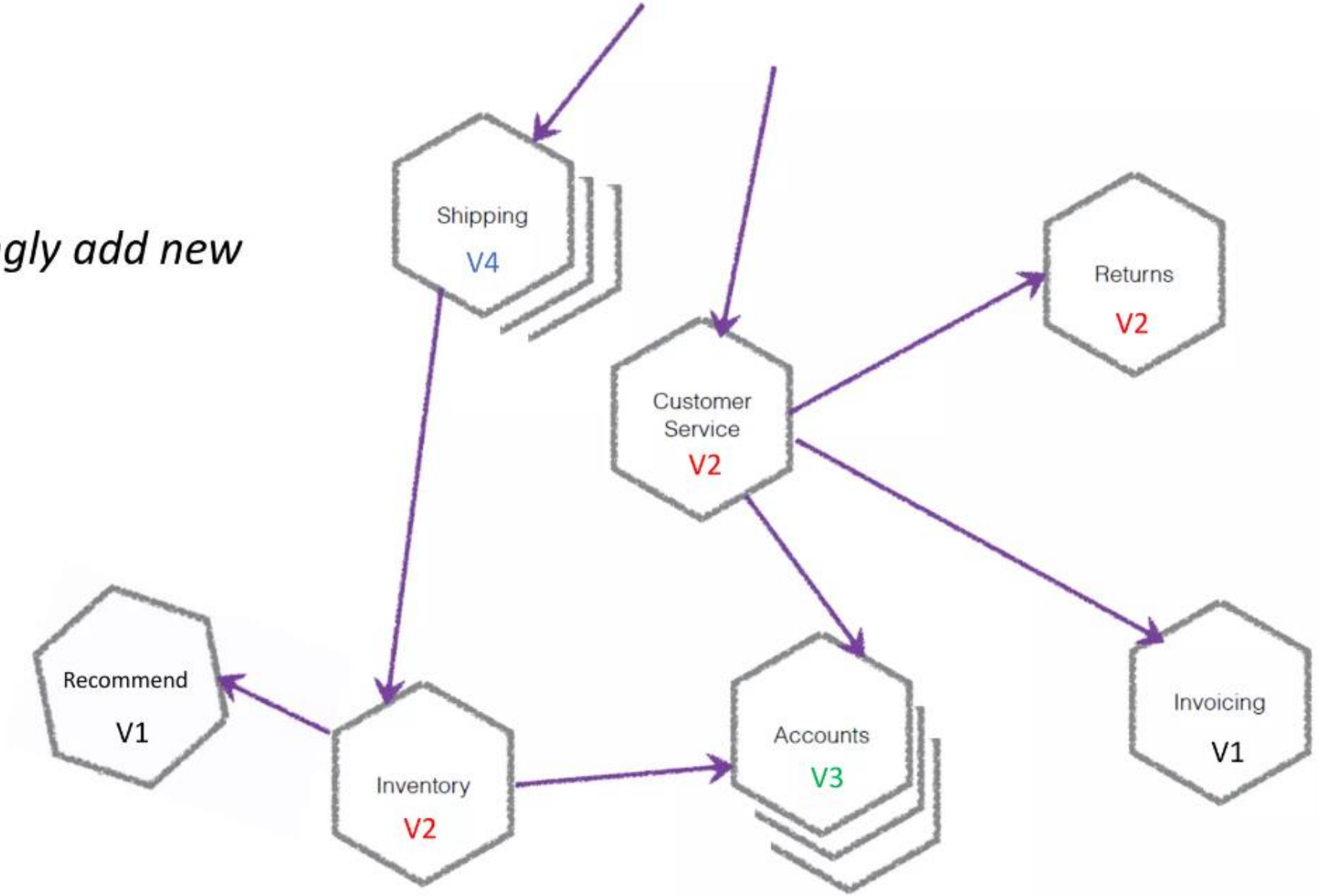
And enhance the service



And scale the parts of the system that really need extra power

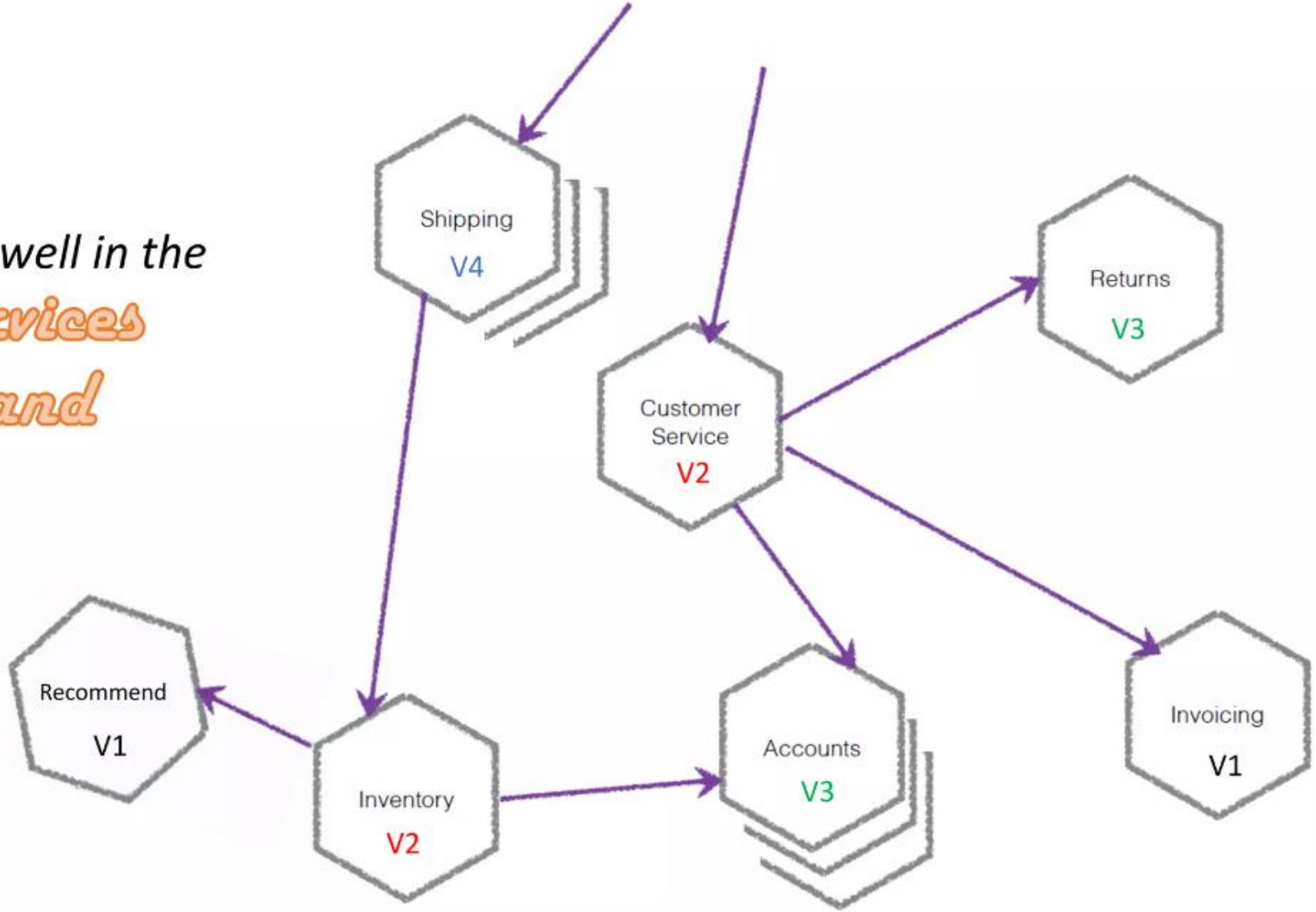


And seamingly add new features



And all goes well in the

microservices wonderland

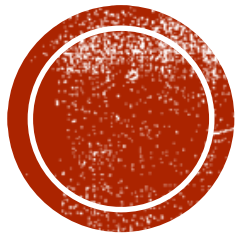


BENEFITS

- Focus on one thing and do it right
- Organizational alignment
- Release functionality faster
- Independent scaling
- Technology diversity; Adopt technologies faster
- Enable security concern segregation
- Enable resiliency by designing for failure

DOWNSIDES

- Cognitive overloading (many tooling options)
- Cognitive overloading (system understanding)
- Testing is more complicated
- Monitoring is more complex
- Operational overhead
- Resiliency isn’t free

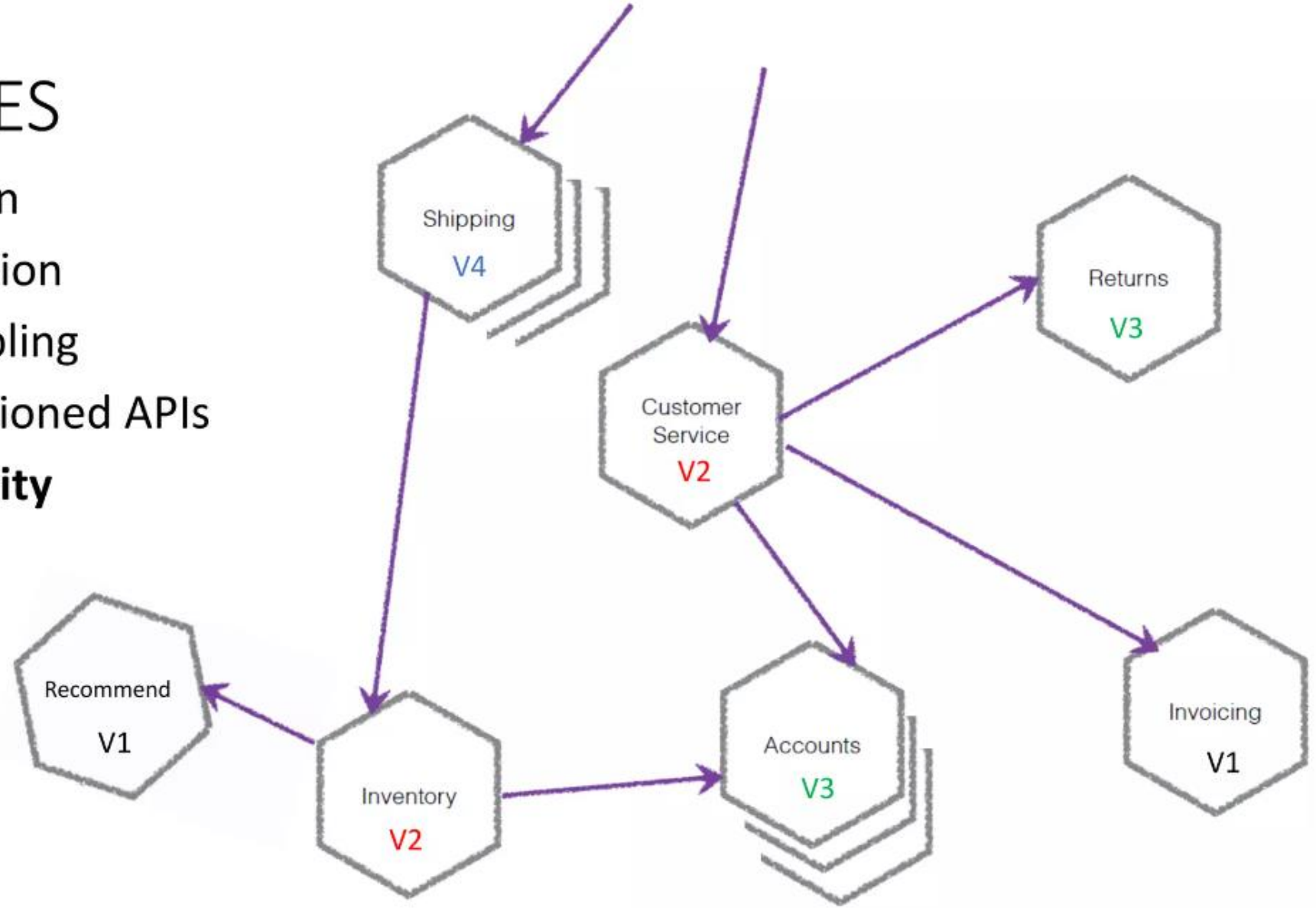


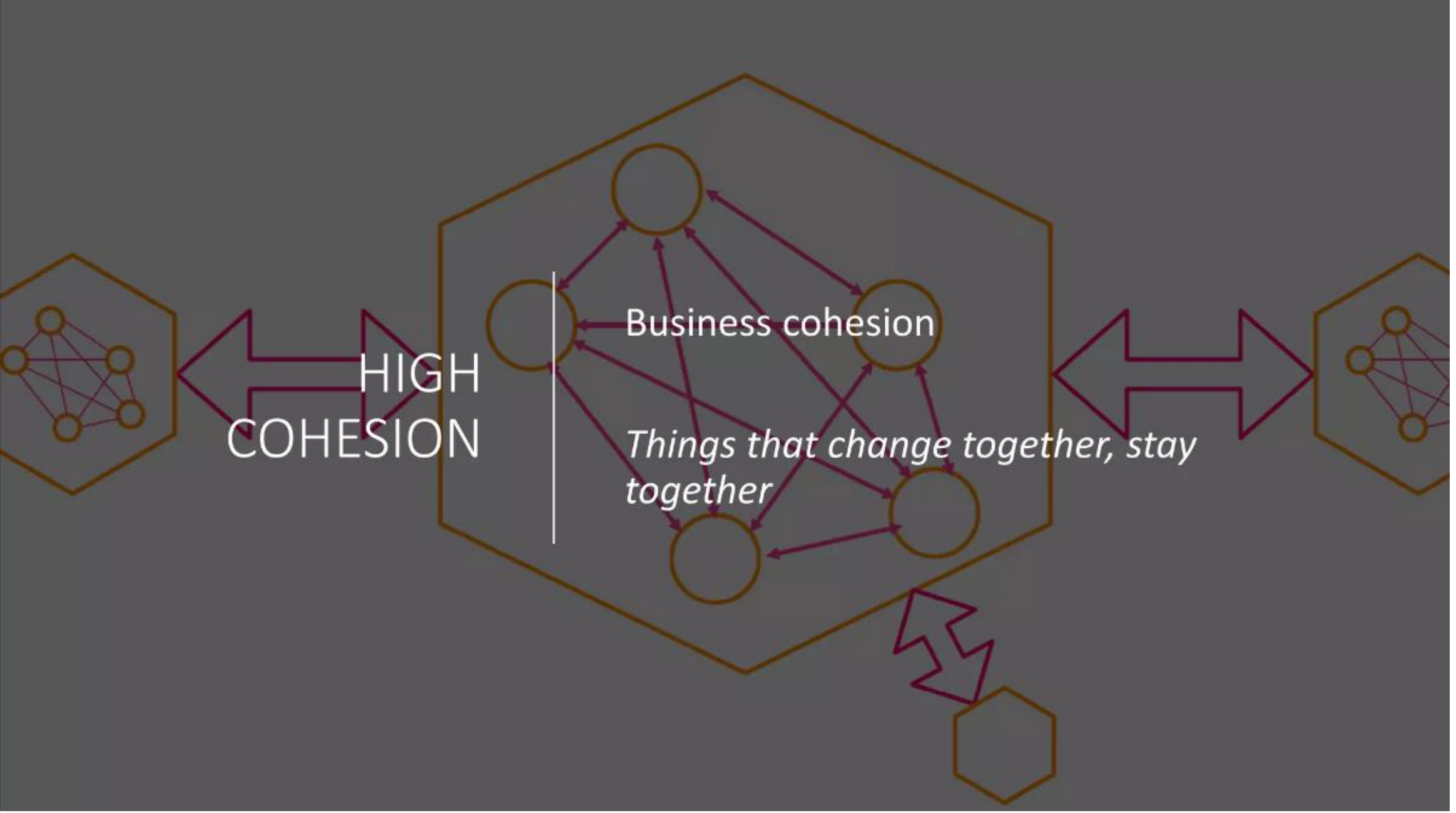
ЗАХТЕВИ ЗА МИКРОСЕРВИСНУ АРХИТЕКТУРУ

REQUIRES

- Automation
- High cohesion
- Loose coupling
- Stable versioned APIs

a.k.a. **Maturity**




HIGH
COHESION
Business cohesion
Things that change together, stay together

LOOSE COUPLING

Implementation coupling

Domain coupling

Temporal coupling

ПОТРЕБА ЗА МИКРОСЕРВИСНОМ АРХИТЕКТУРОМ

Microservices are about ***problems at scale***

- *# of developers*
- *# of features*
- *# of users*

*IT IS **A PATH** TO ACHIEVE*
*A **BUSINESS OBJECTIVE***

EMBRACE

Eventual consistency

Data redundancy & caching

No single cannonical model

Long running transactions

Truth is,

Most problem-contexts don’t (**usually**) have a “scale problem”

However, there are several potential *seams*


Calculations


Partners


Users & Authorizations

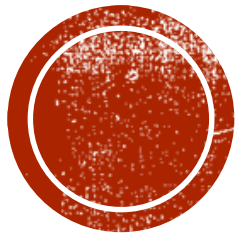

Customers


Business processes


Document management & printing


Widgets


...

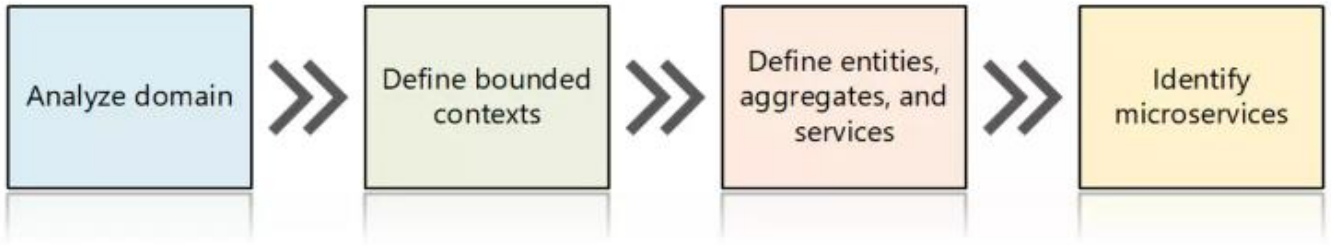


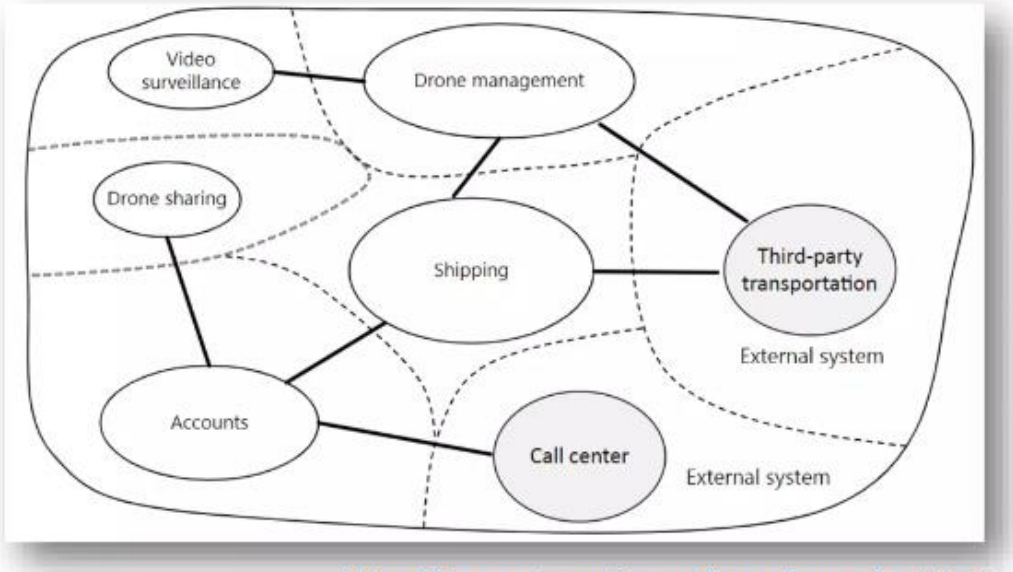
РЕАЛИЗАЦИЈА МИКРОСЕРВИСНЕ АРХИТЕКТУРЕ

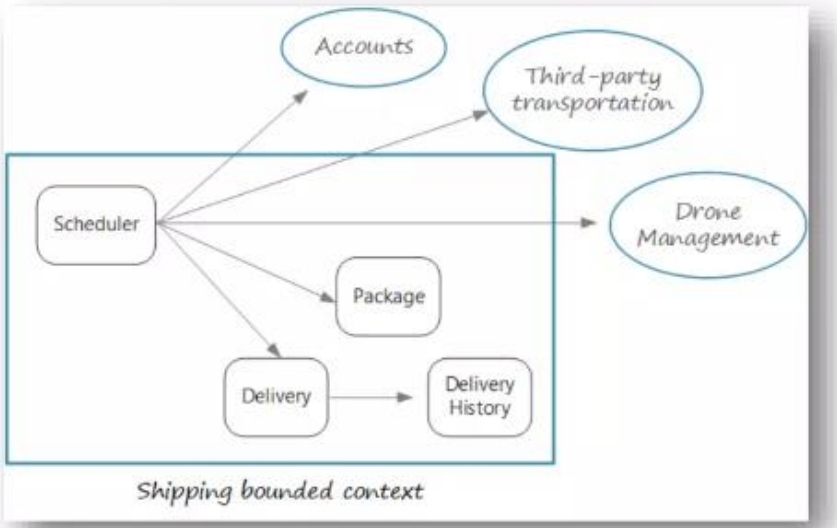
If we decide to build it

DESIGNING AND ARCHITECTING

IDENTIFYING SERVICE BOUNDARIES

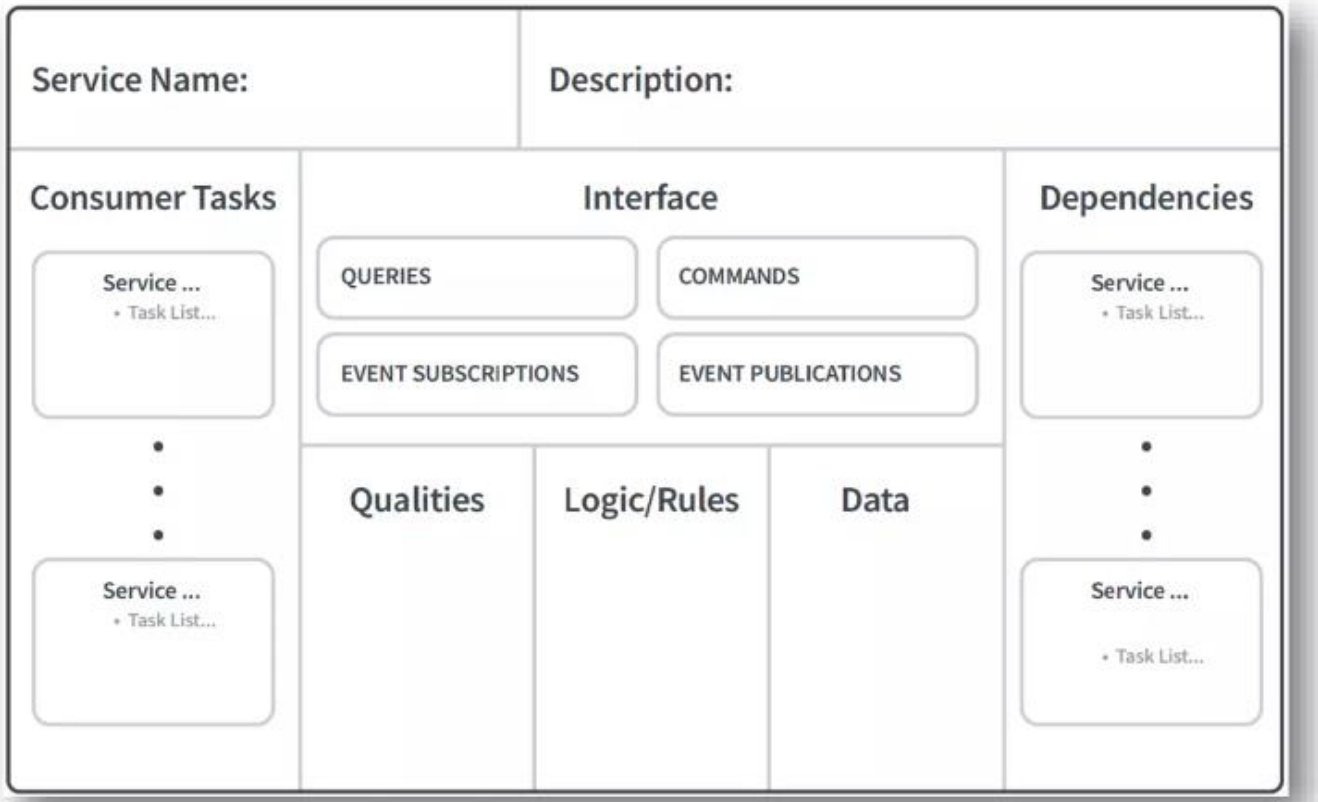






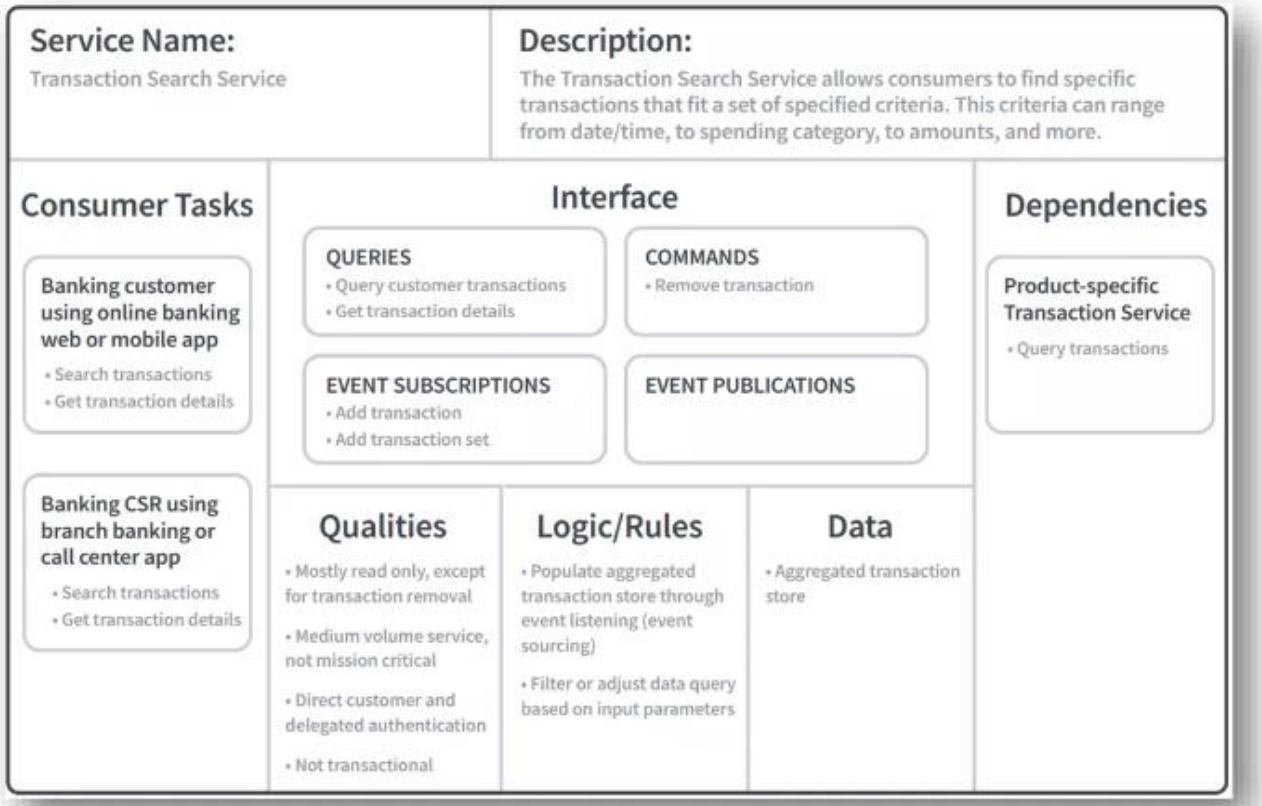
https://docs.microsoft.com/en-us/azure/architecture/microservices/model/domain-analysis

THE MICROSERVICES DESIGN CANVAS



Matt Mclarty, Irakli Nadareishvili (2017)

THE MICROSERVICES DESIGN CANVAS



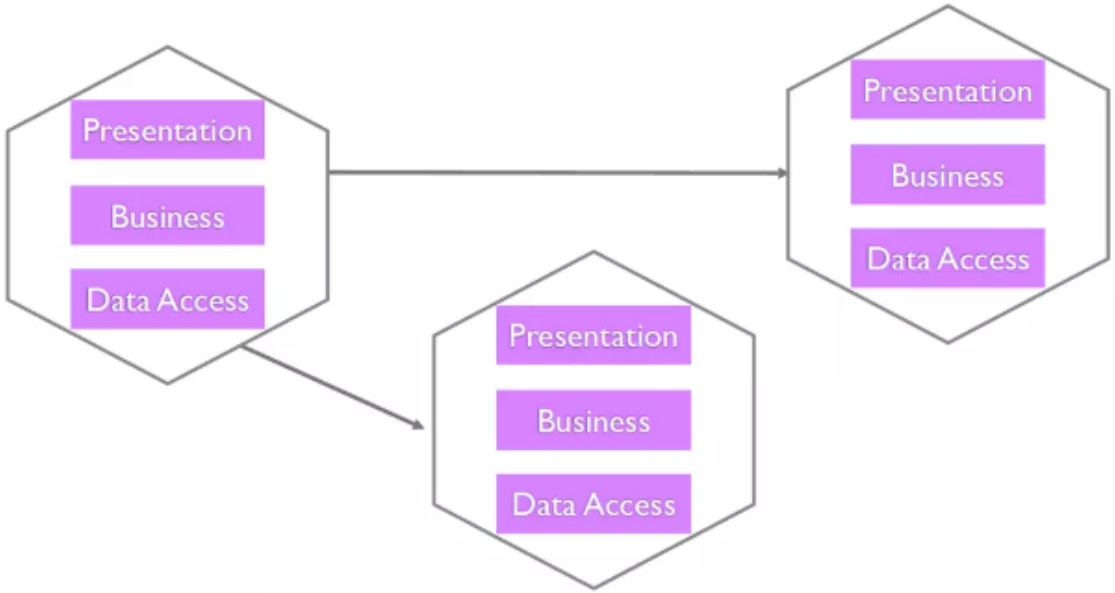
Matt Mclarty, Irakli Nadareishvili (2017)

Vertical slices

Instead of focusing on the nouns in your system (Orders, Customers, Products), you instead should **focus on capabilities** (Catalog, Checkout).

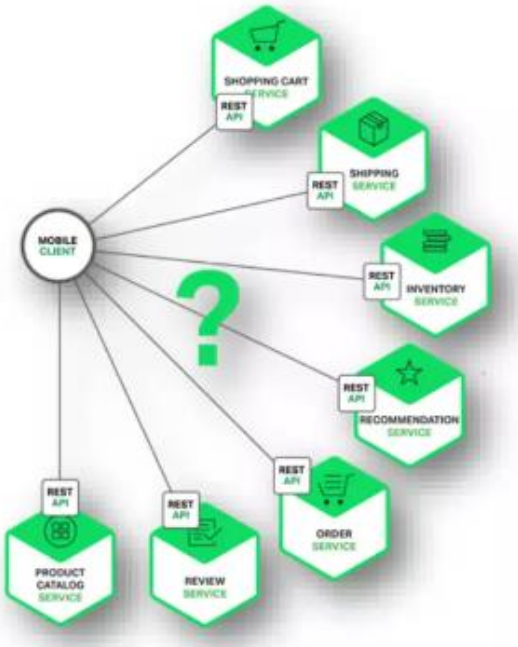
Each microservice owns the end-to-end

Even the UI & data store!

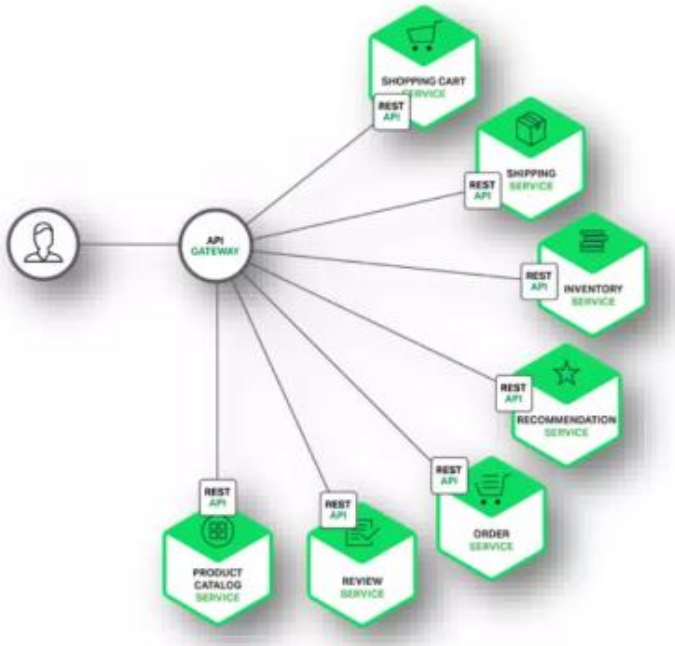


Interacting with the system

Direct connection



Use a Façade

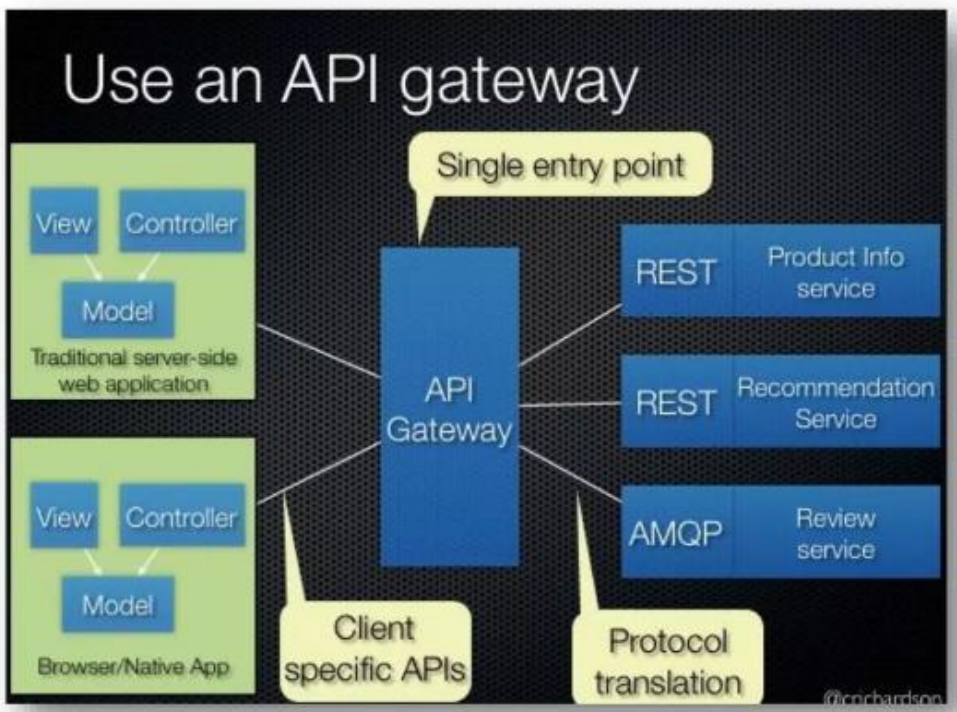


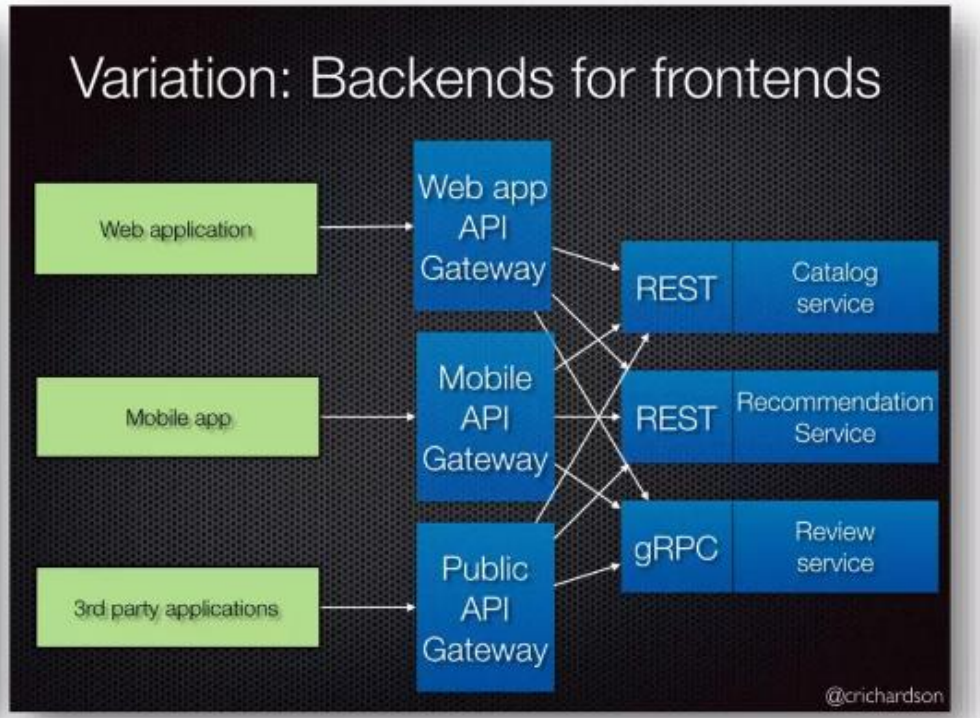
https://www.nginx.com/blog/building-microservices-using-an-api-gateway/

API Gateway

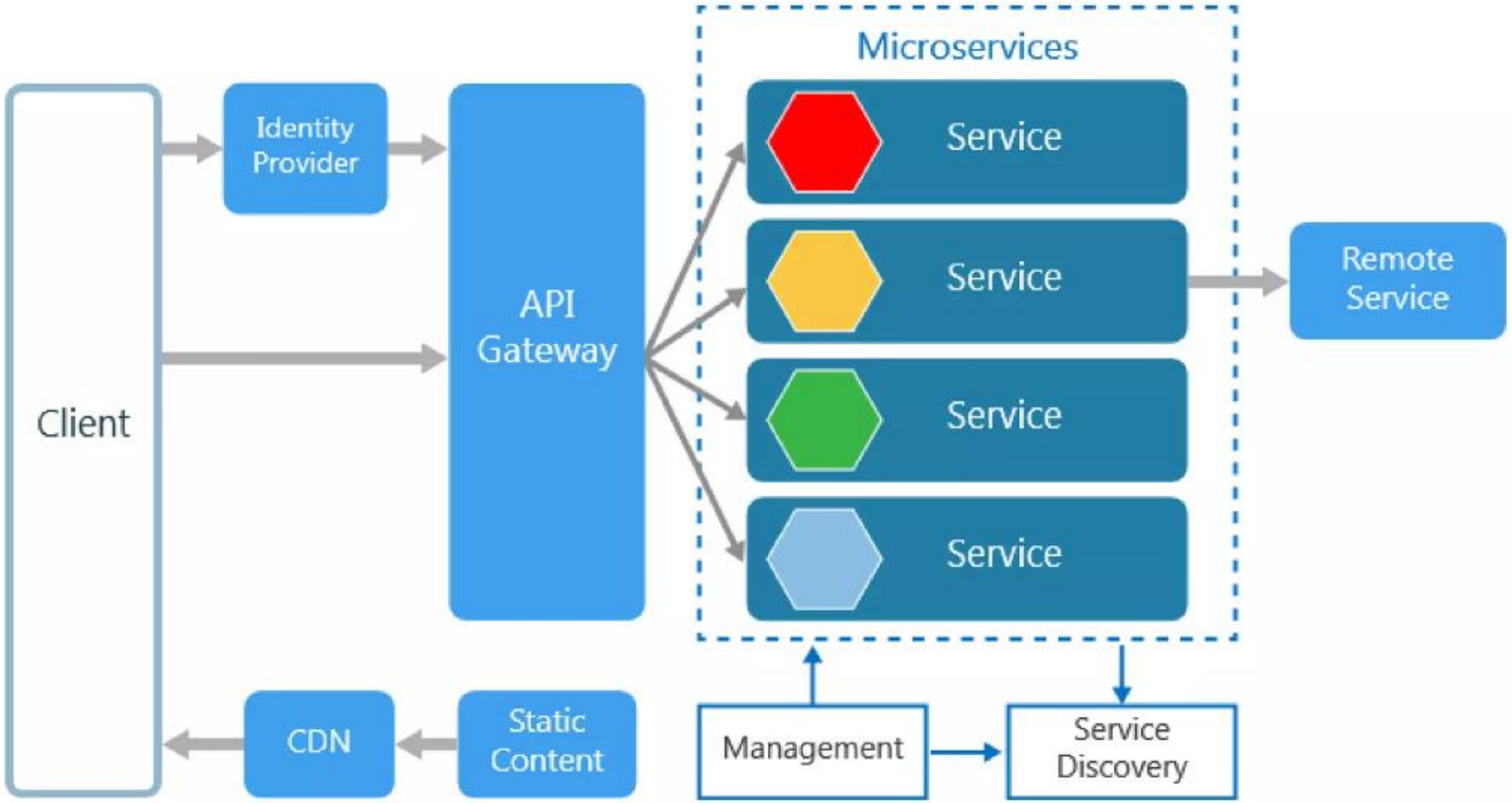
- Authentication
- Throttling
- Translation
- Monitoring & logging usage
- Monetization
- Routing
- Composition
- Hiding implementation details

API Gateway





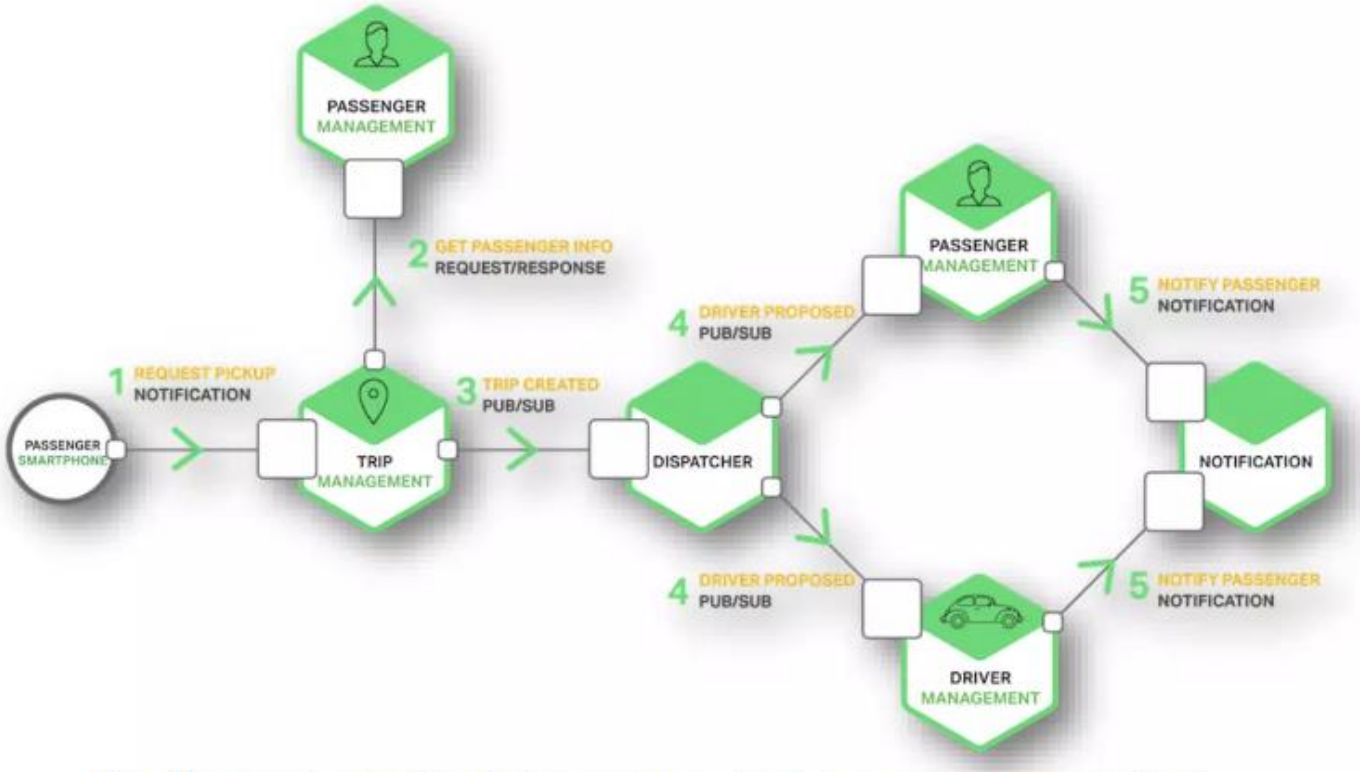
https://microservices.io/patterns/apigateway.html



https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices

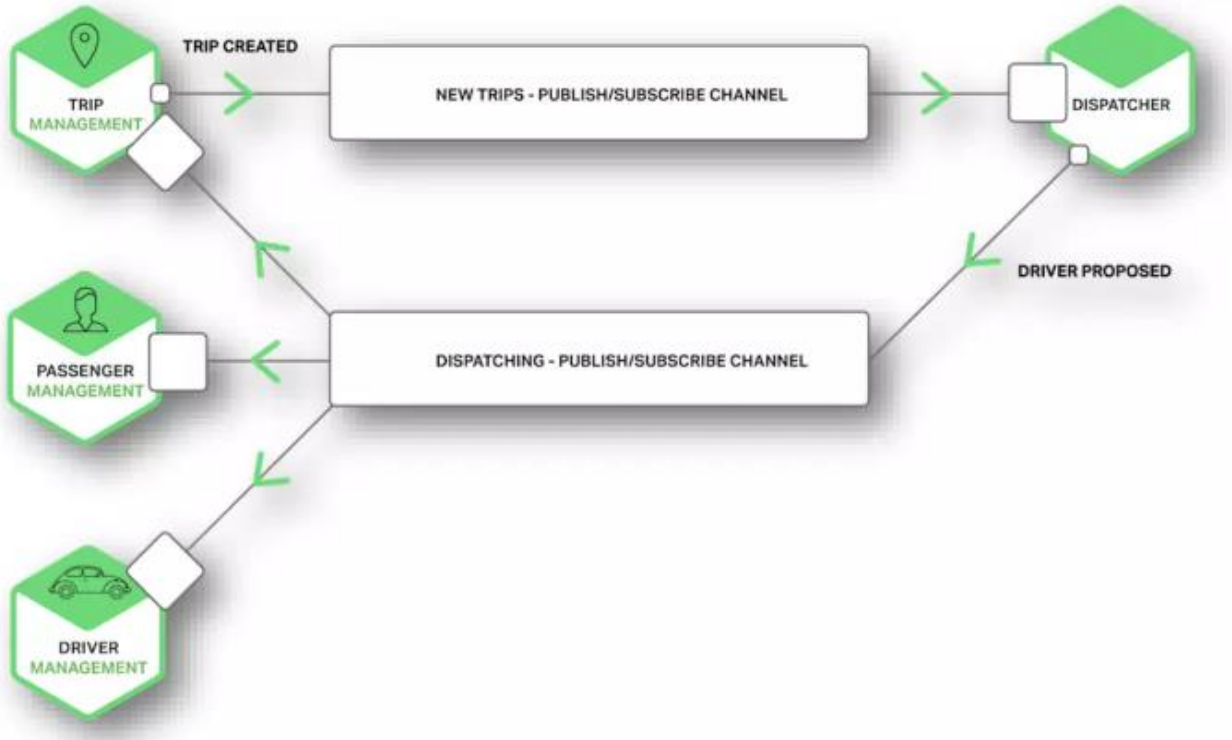
IPC

	One-to-One	One-to-Many
Synchronous	Request/response	—
Asynchronous	Notification	Publish/subscribe
	Request/async response	Publish/async responses

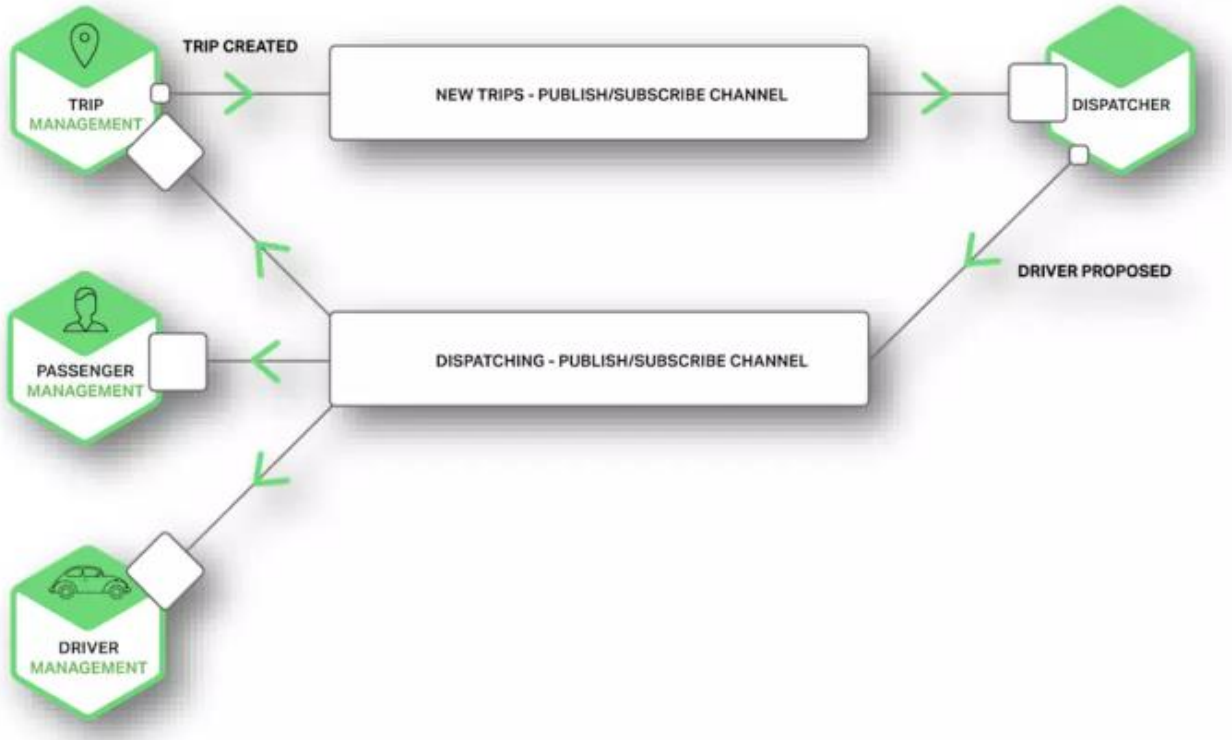


https://www.nginx.com/blog/building-microservices-inter-process-communication/

Asynchronous IPC with asynchronous response

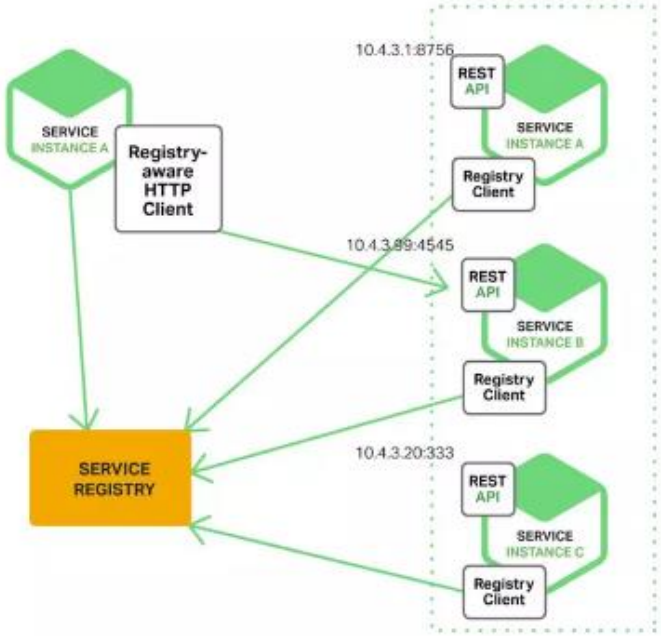


Asynchronous IPC with asynchronous response

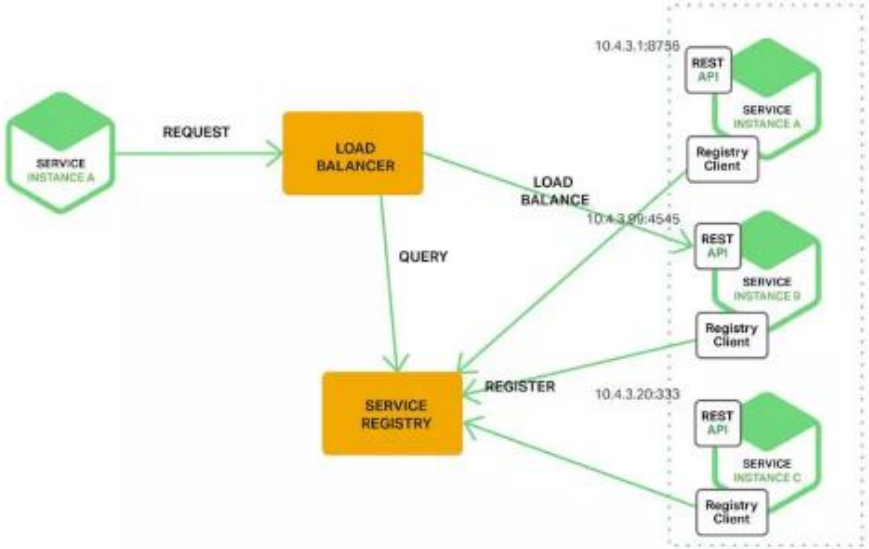


Service Discovery

Client-side discovery

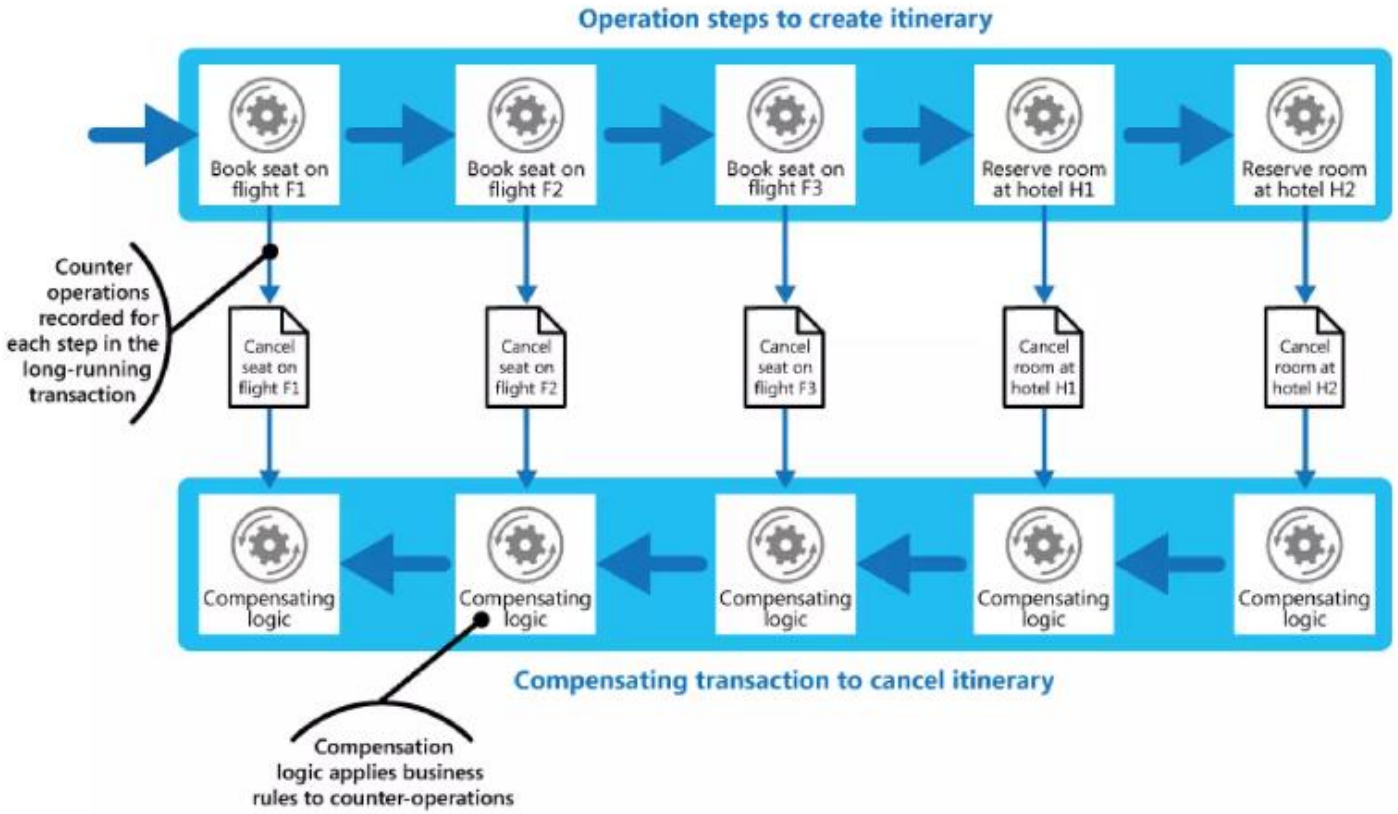


Server-side discovery



https://www.nginx.com/blog/service-discovery-in-a-microservices-architecture/

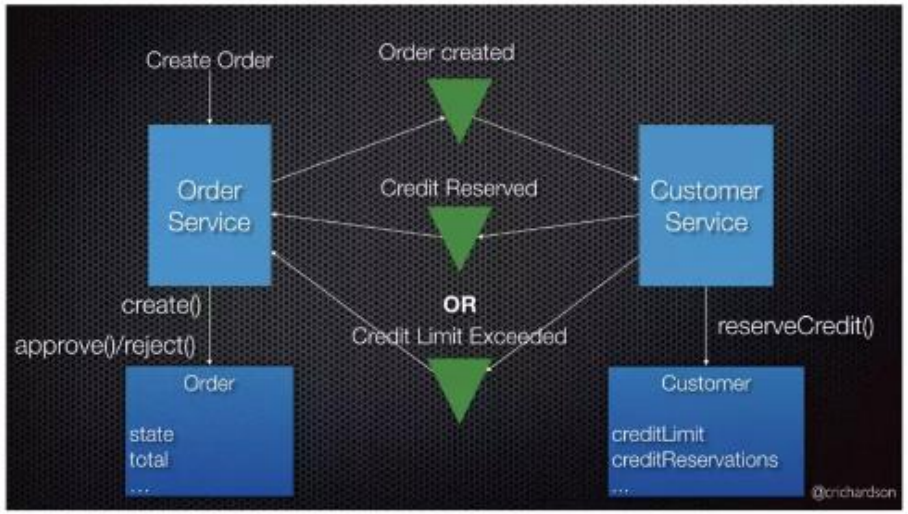
SAGAS, COMPENSATING TRANSACTIONS



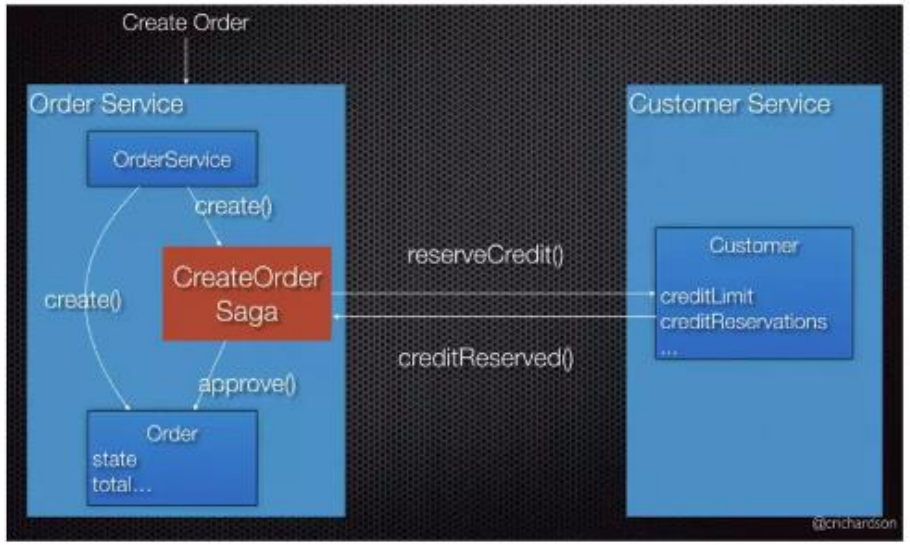
https://docs.microsoft.com/en-us/azure/architecture/patterns/compensating-transaction

SAGAS, COMPENSATING TRANSACTIONS

CHOREOGRAPHY



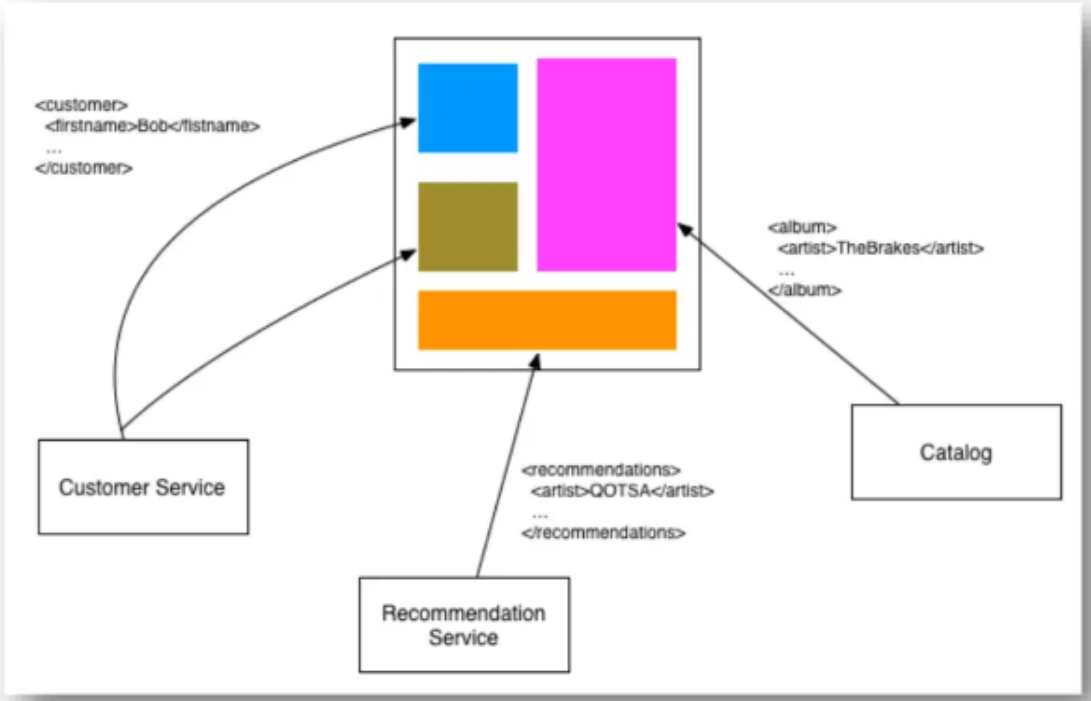
ORCHESTRATION



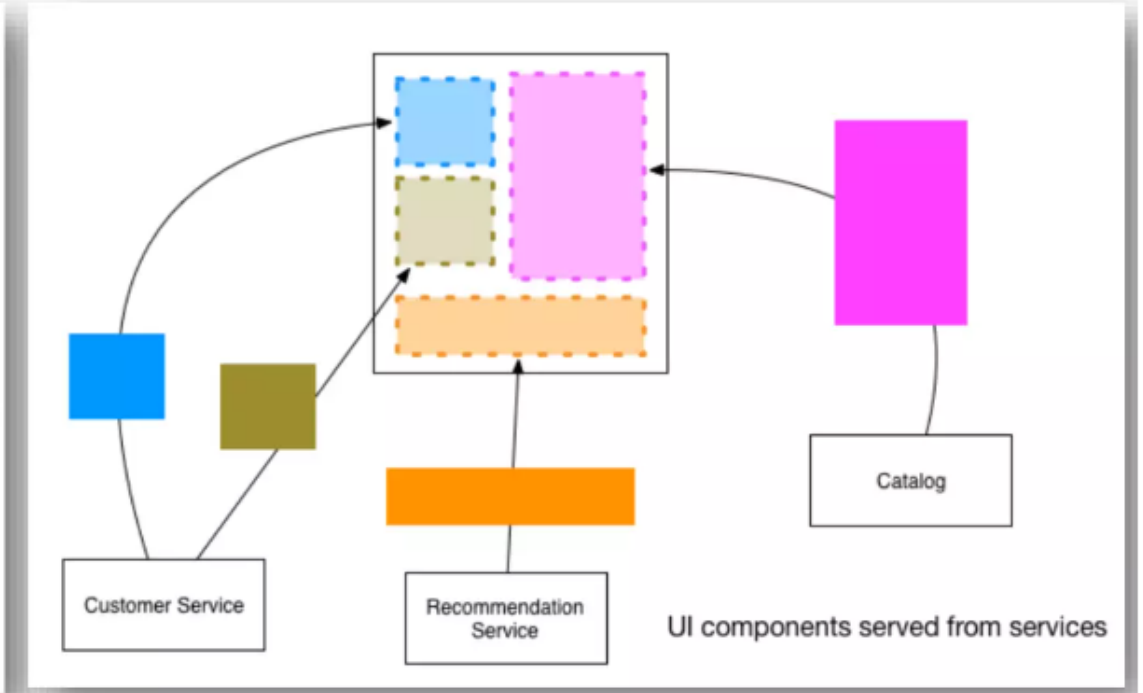
https://microservices.io/patterns/data/saga.html

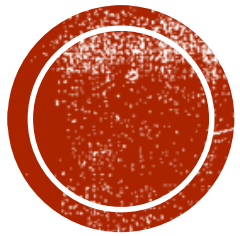
MICROSERVICES AND THE UI

SINGLE PAGE APPLICATIONS



COMPOSITE UI ASSEMBLY

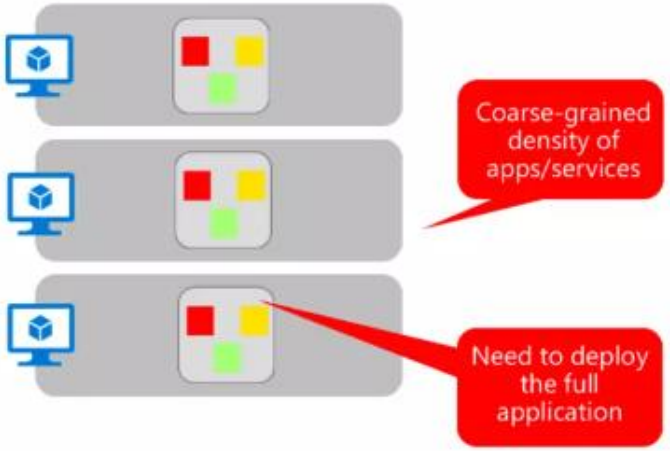




ОД МОНОЛИТНЕ ДО МИКРОСЕРВИСНЕ АРХИТЕКТУРЕ

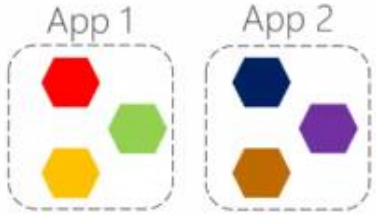
Monolithic deployment approach

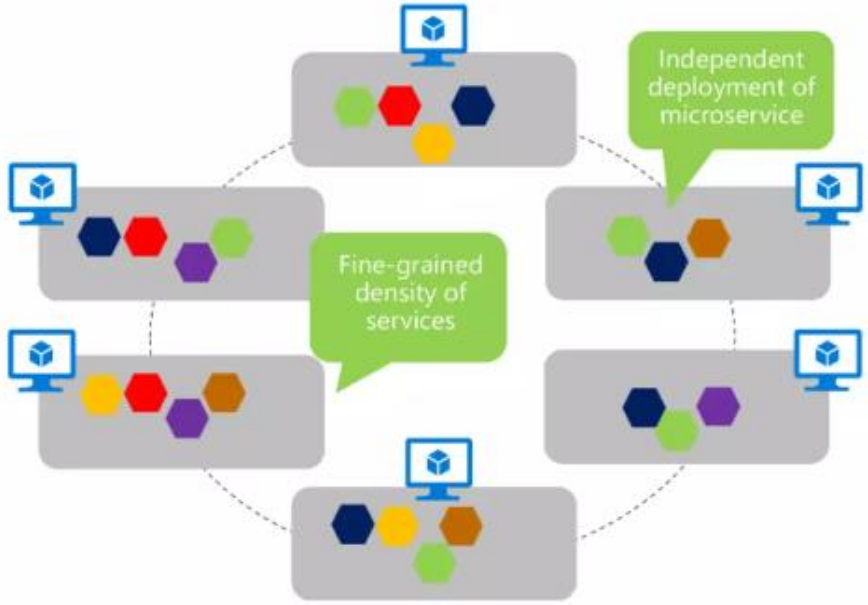
- A traditional application has most of its functionality within a few processes that are componentized with layers and libraries.
- Scales by cloning the app on multiple servers/VMs



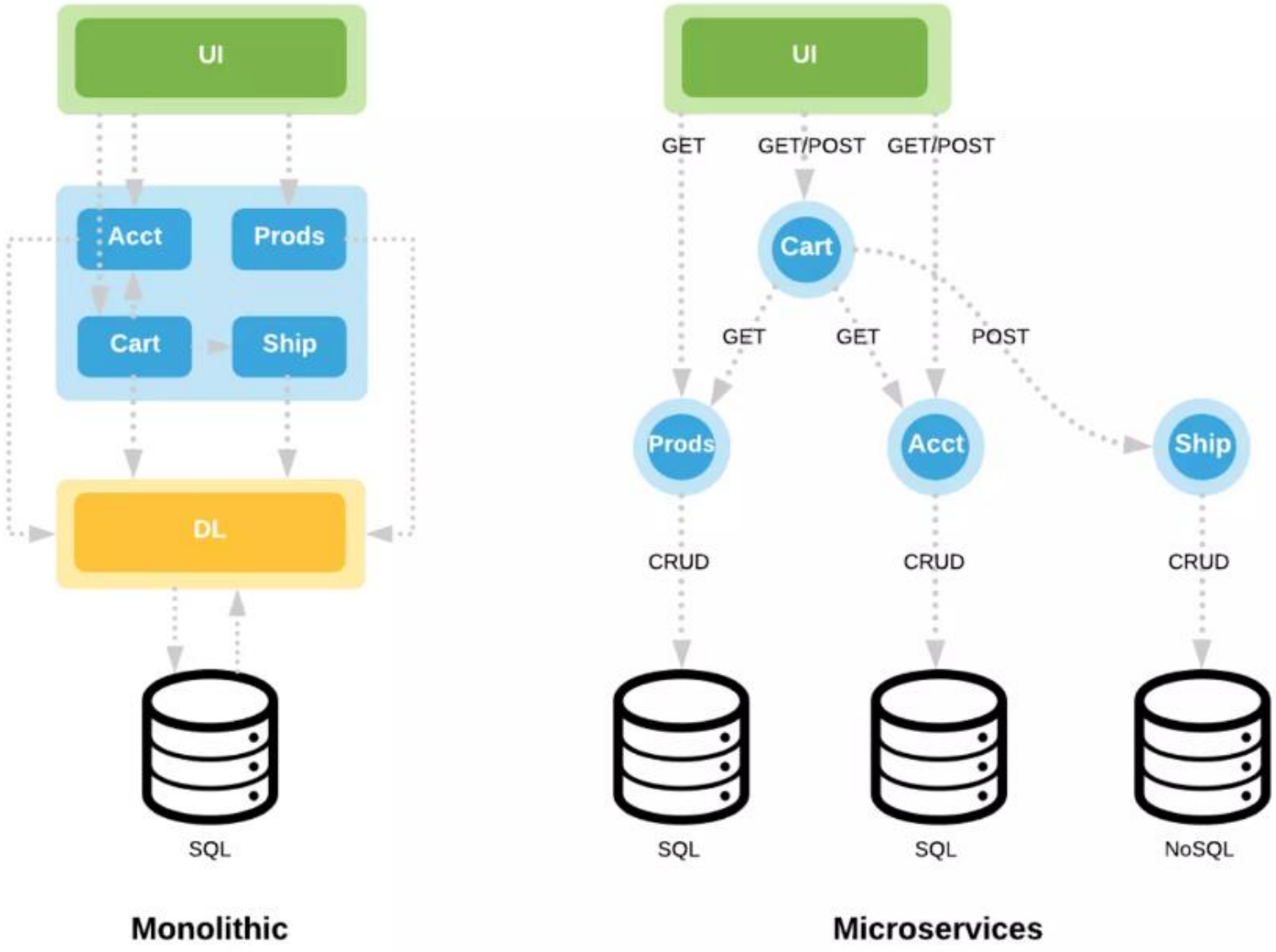
Microservices application approach

- A microservice application segregates functionality into separate smaller services.
- Scales out by **deploying each service independently** with multiple instances across servers/VMs



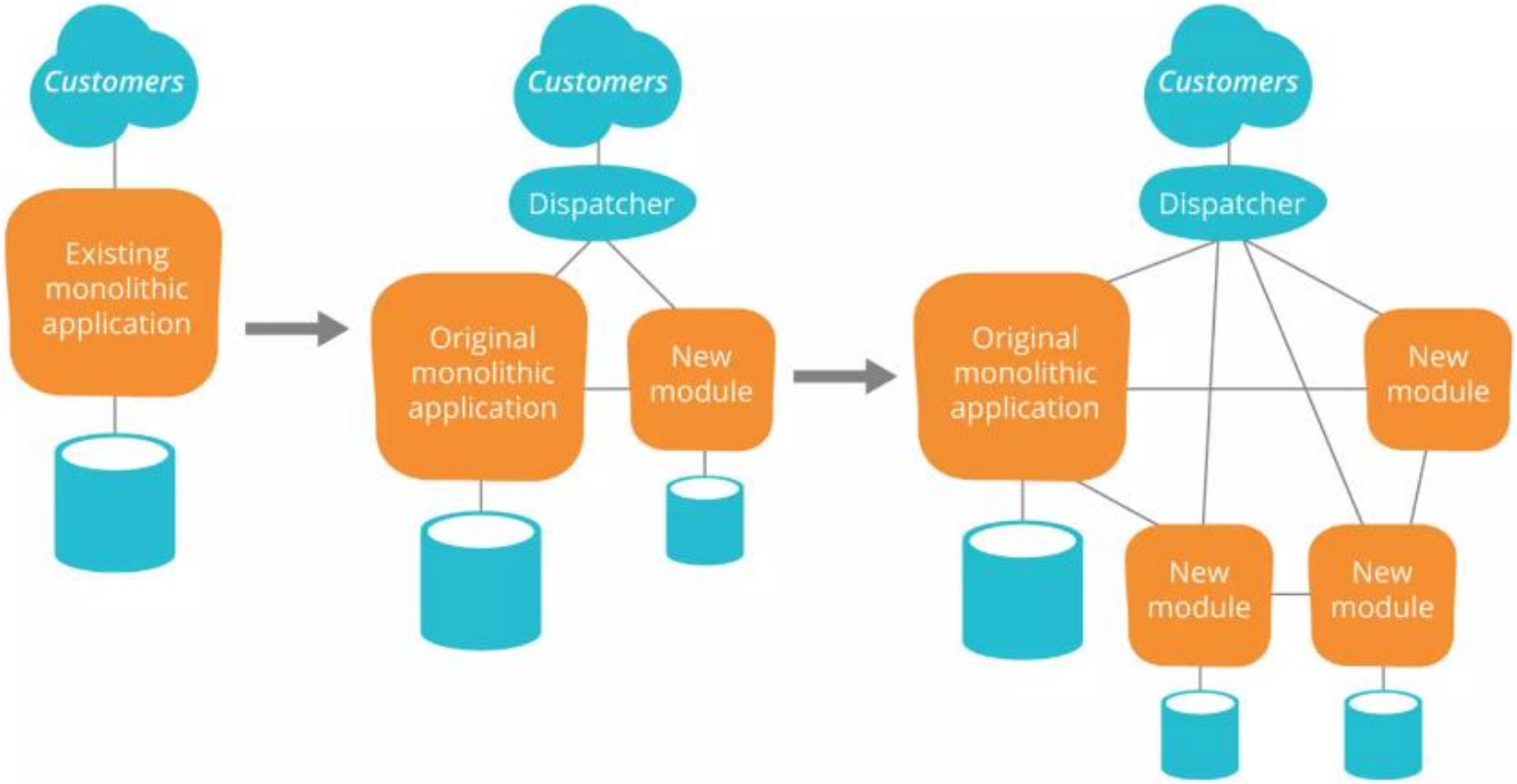


https://docs.microsoft.com/en-us/dotnet/standard/microservices-architecture/architect-microservice-container-applications/microservices-architecture



https://cloudacademy.com/learning-paths/dotnet-monolithic-to-microservices-migration-284/

THE STRANGLER PATTERN



http://mycloudcomputing2017.blogspot.com/2017/02/what-is-strangler-application-pattern.html

ЗАКЉУЧАК

microservices

- Are not the goal
- Are about problems at scale
- Are manageable units of functionality and deployability
- Own the end-to-end
- Require automation and maturity