



Microservices Workshop





Somkiat Puisungnoen

Search

Somkiat | Home

Update Info 1 View Activity Log 10+ ...

Timeline About Friends 3,138 Photos More

When did you work at Opendream? X

... 22 Pending Items

Post Photo/Video Live Video Life Event

What's on your mind?

Public Post

Intro

Software Craftsmanship

Software Practitioner at สยามชานาญกิจ พ.ศ. 2556

Agile Practitioner and Technical at SPRINT3r

Somkiat Puisungnoen 15 mins · Bangkok · ...

Java and Bigdata

 Microservices

© 2020 - 2023 Siam Chamnankit Company Limited. All rights reserved.

3

Microservice



Module 1 : Design

Cloud Native Application

Evolution of architecture

Microservice architecture

How to decompose app to Microservice

Communication between service

Data consistency

Workshop



Module 2 : Develop + Testing

Recap Microservice

Properties of Microservice

Microservice 1.0 - 4.0

How to develop Microservice ?

How to test Microservice ?

12-factors app

Observable services



Module 3 : Deploy

How to deploy Microservice ?
Continuous Integration and Delivery
Practices of Continuous Integration
Deployment strategies
Working with containerization (Docker)
Workshop



Design

Develop

Deploy

Observability

Continuous Integration/Delivery



“ Don’t use Microservice ”

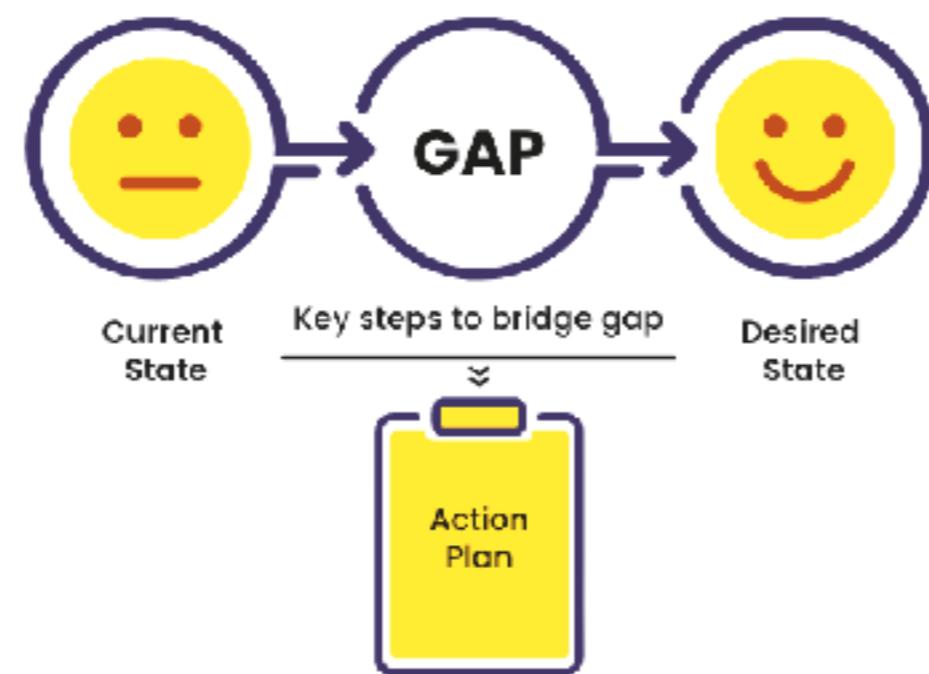
Start with your problems ?



Software Architecture



Pain Points ?





Performance

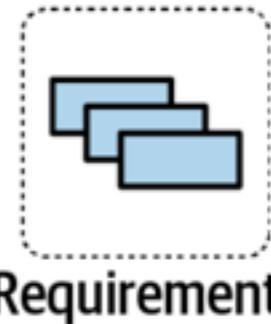


Scalability



Security

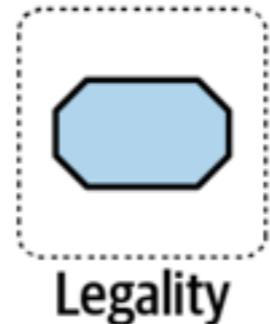
Software Architecture



Requirements



Auditability

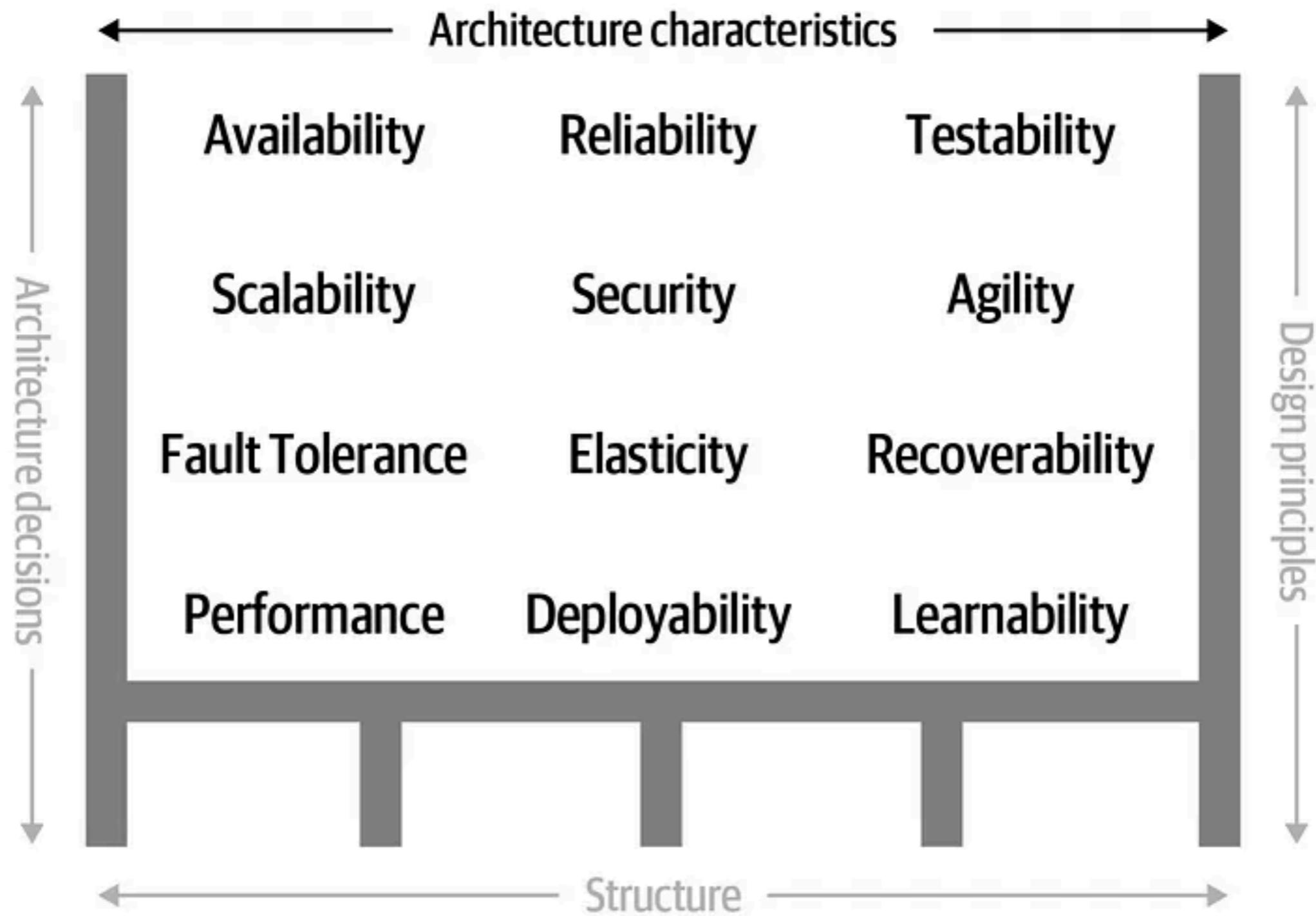


Legality



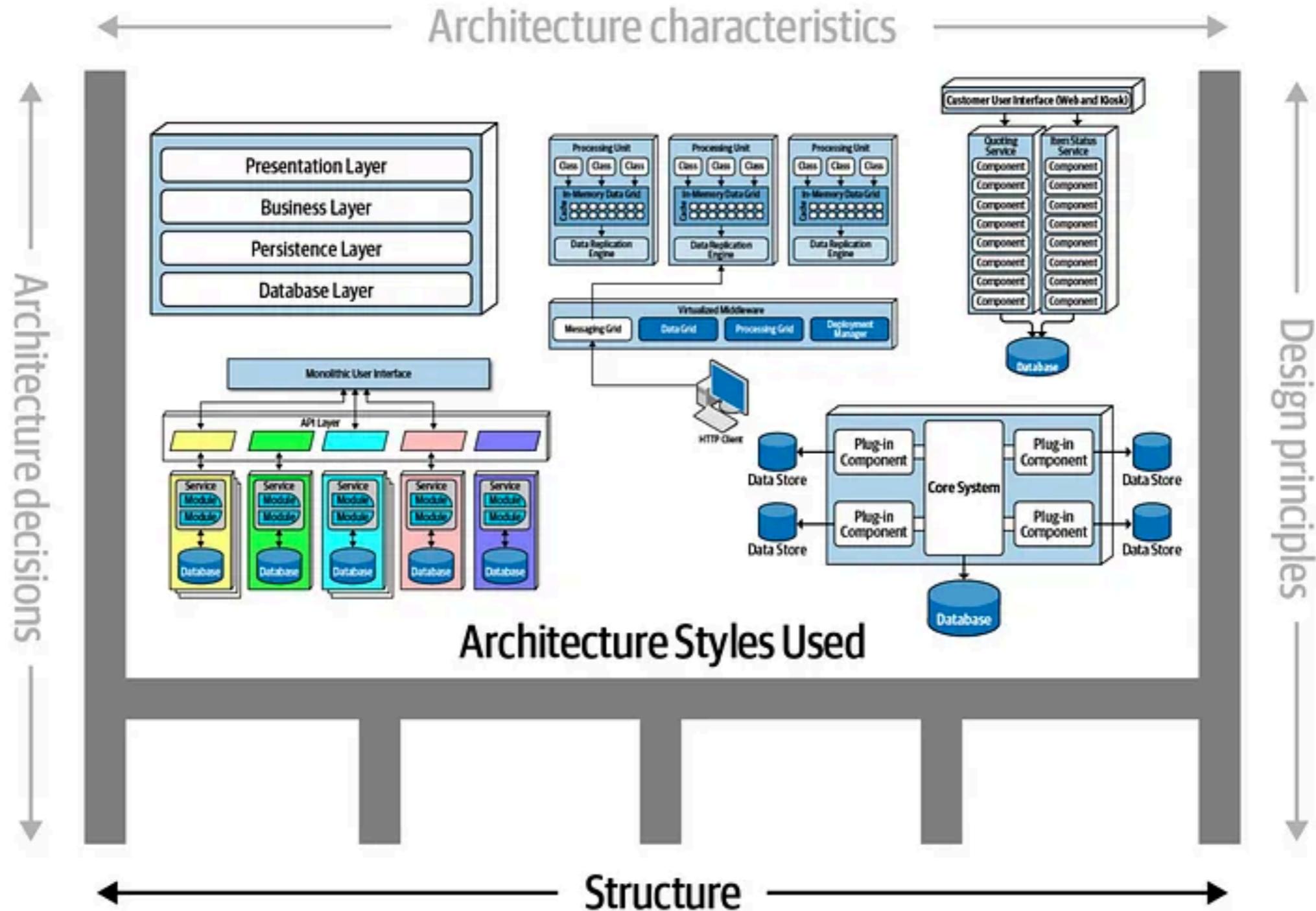
Data

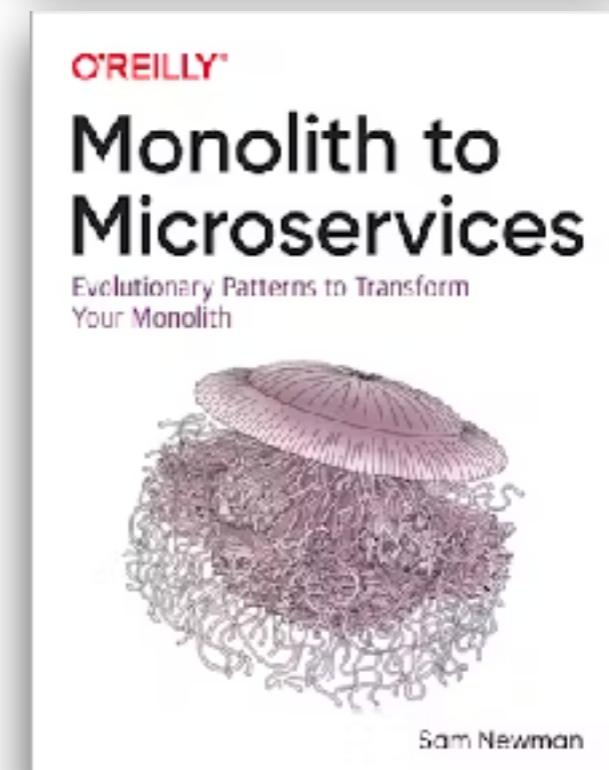
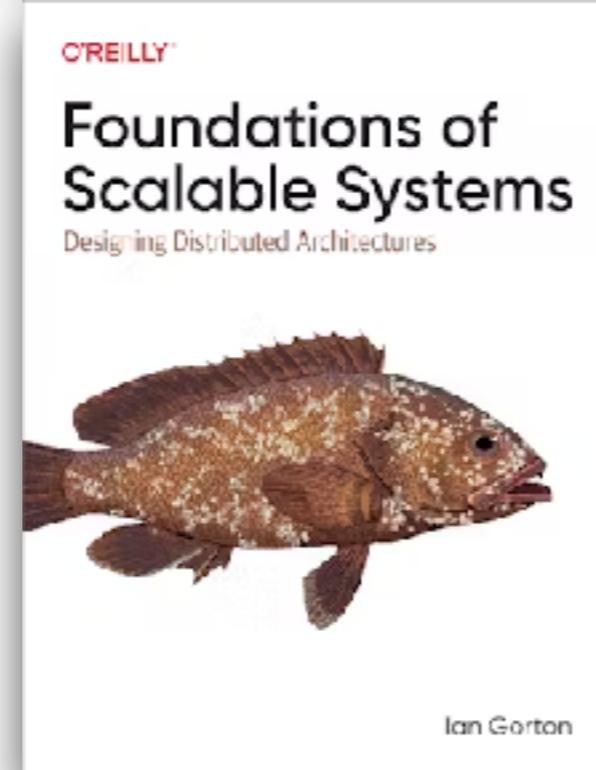
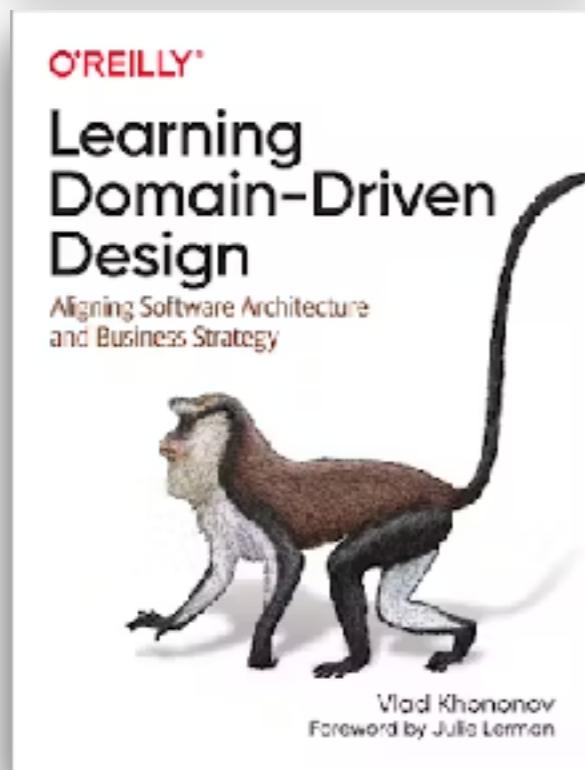
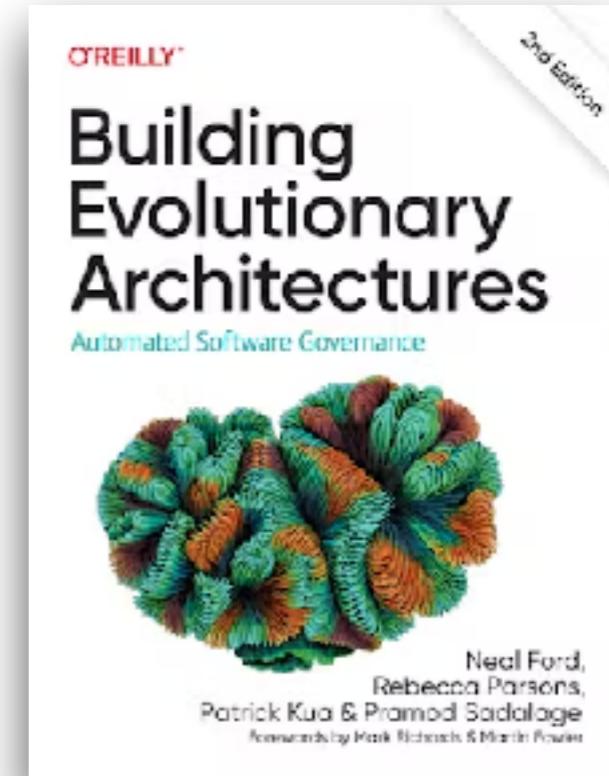
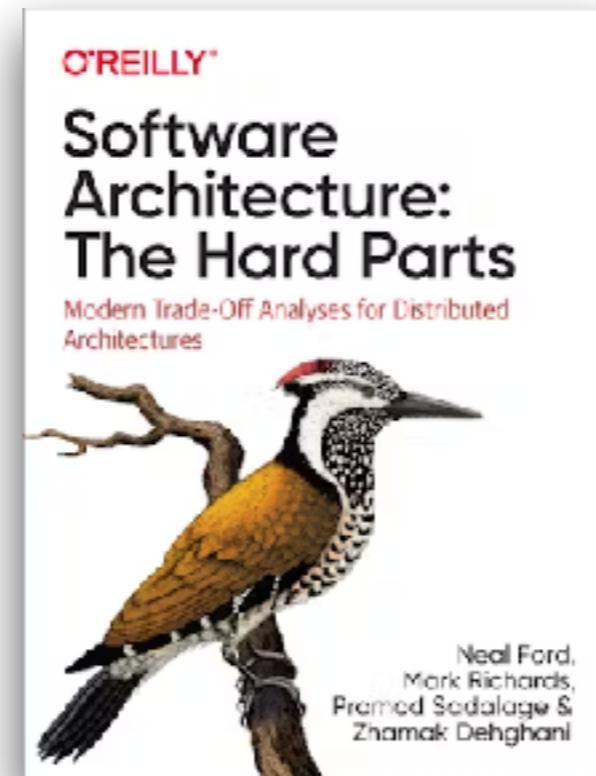
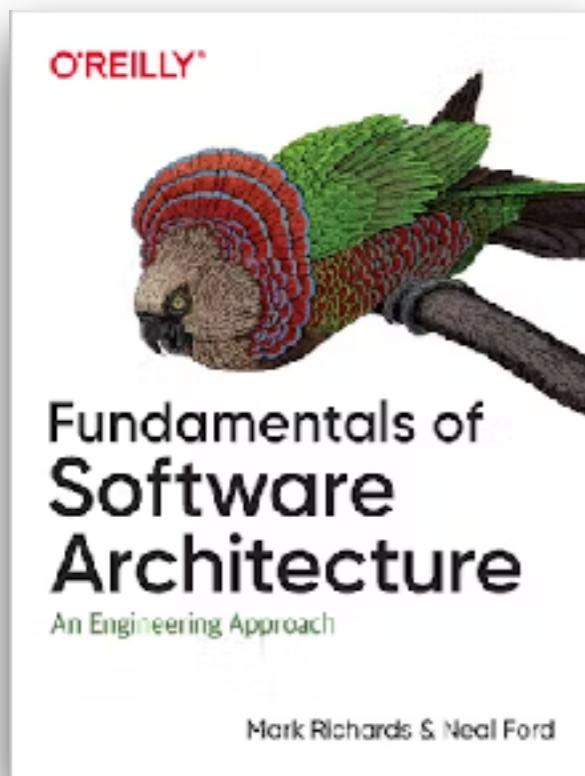




<https://www.amazon.com/Fundamentals-Software-Architecture-Comprehensive-Characteristics/dp/1492043451>







<https://www.amazon.com/Fundamentals-Software-Architecture-Comprehensive-Characteristics/dp/1492043451>

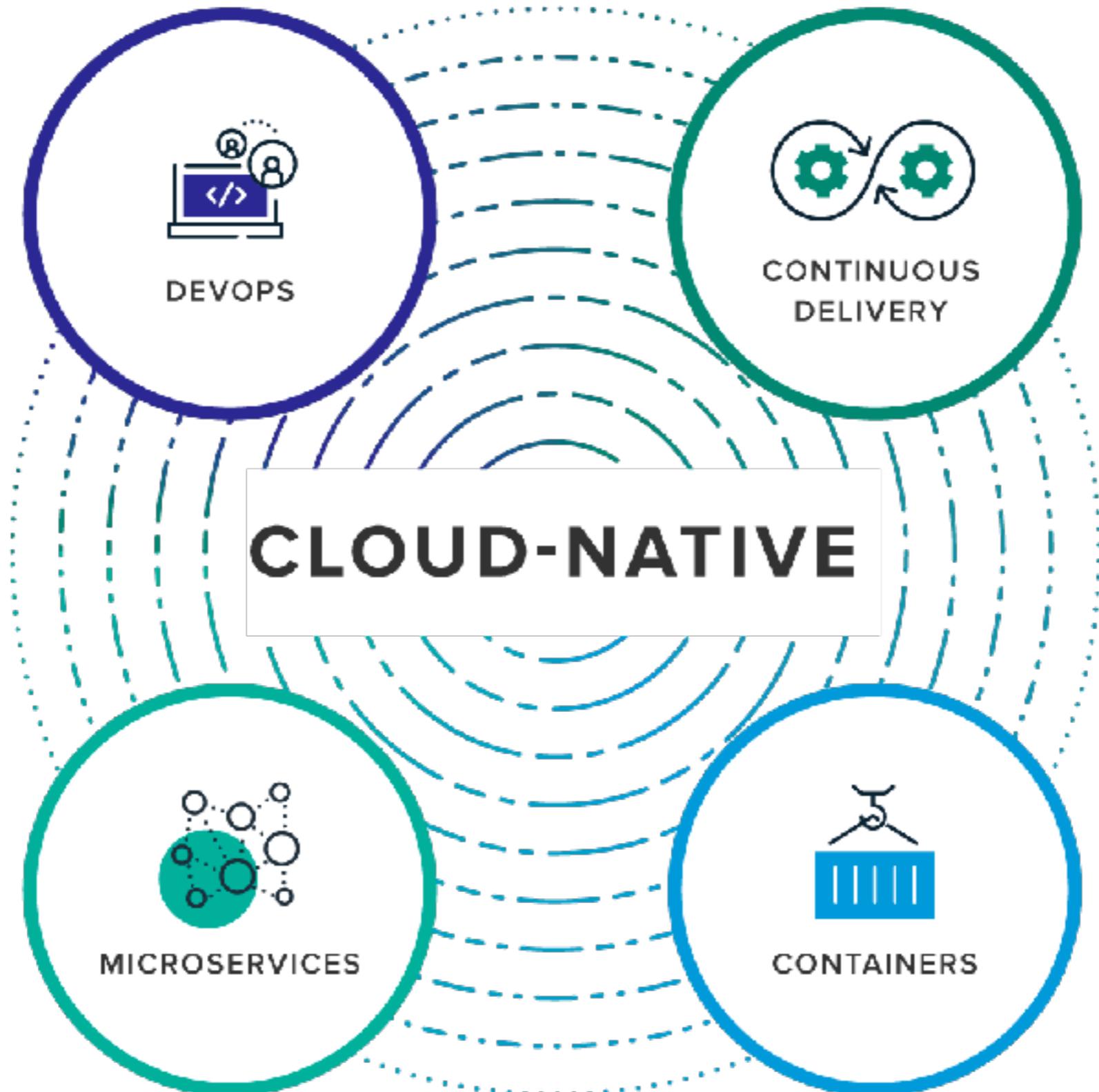


Microservices

© 2020 - 2023 Siam Chamnankit Company Limited. All rights reserved.

Module 1 : Design





<https://pivotal.io/cloud-native>



Cloud Native ?

Build efficiency

Scalability

Portability

On-premise and On-cloud

Loosely
Coupled

Resilience

Manageable

Observable

<https://www.somkiat.cc/cloud-native-go-part-1/>

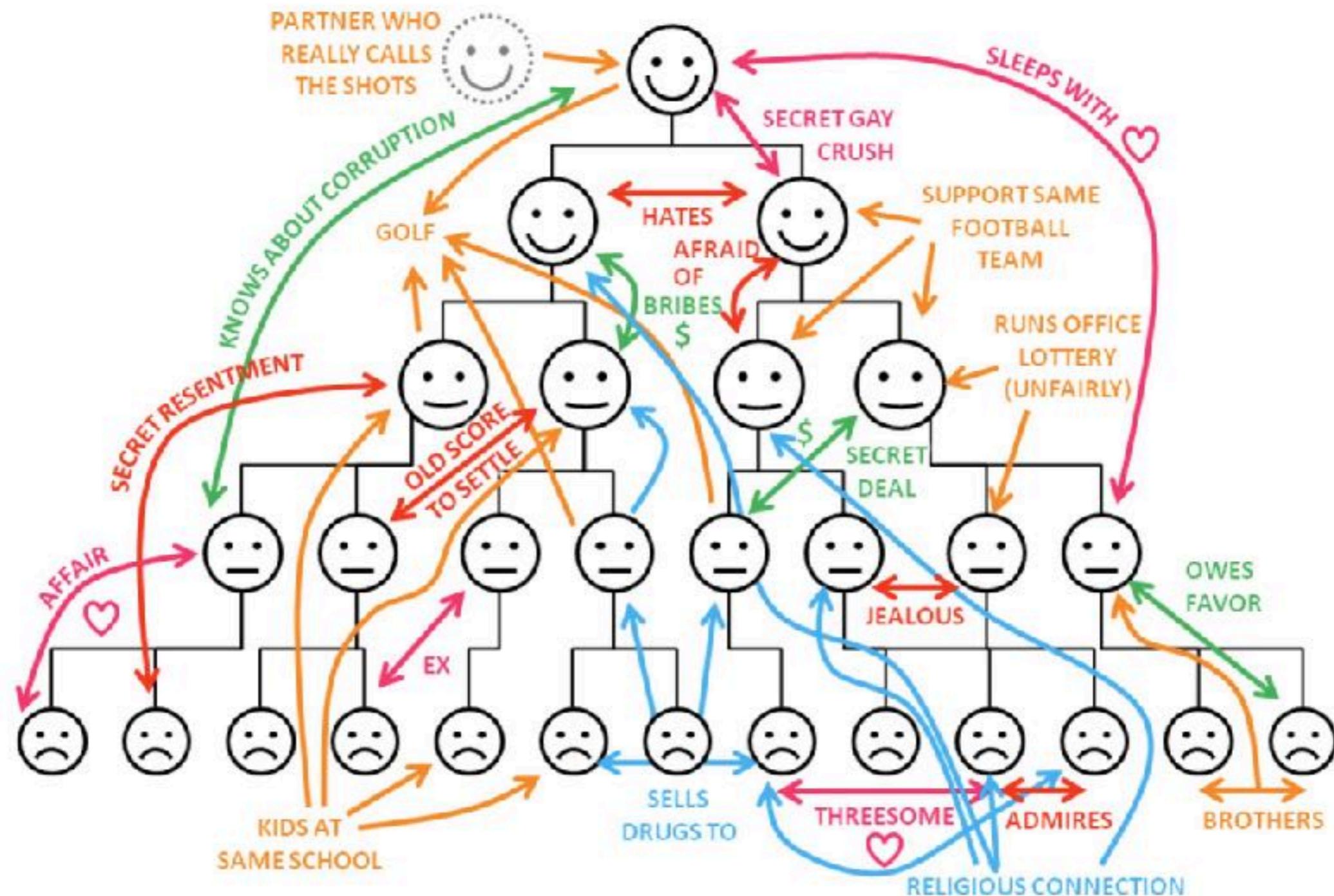


**Any organization that designs a system
will produce a design whose structure
is a copy of the organization's
communication structure.**

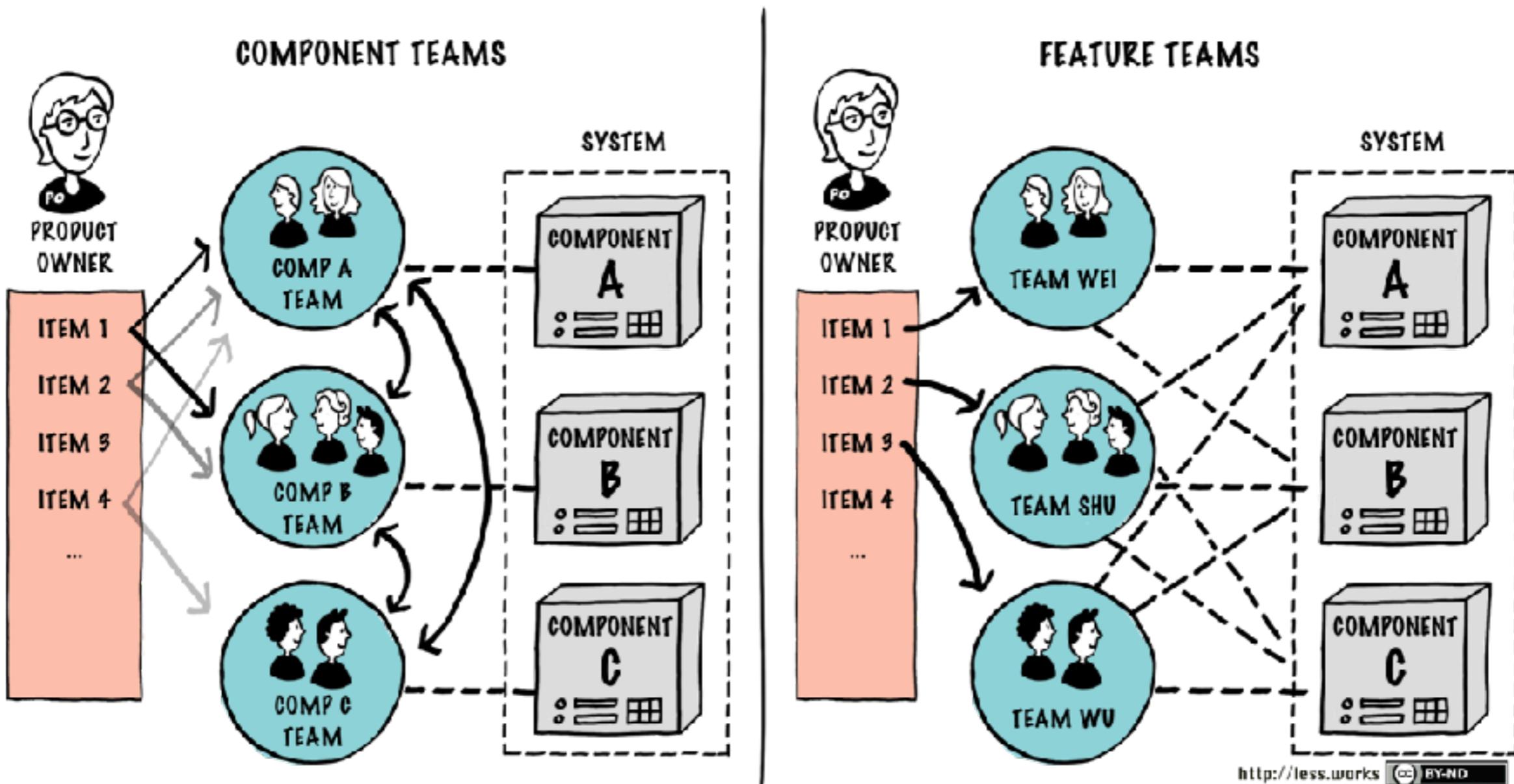
-- Melvin Conway



Organization Structure



Component vs Feature teams



<https://less.works/less/structure/feature-teams>



Evolution of Architecture



WANT TO KNOW THE BEST SOFTWARE ARCHITECTURE?



Pangaea
300M years ago

LOOK AT THE EVOLUTION OF THE EARTH.



nowadays

Daniel Storri {turnoff.us}



3 Pillars of Evolution Architecture

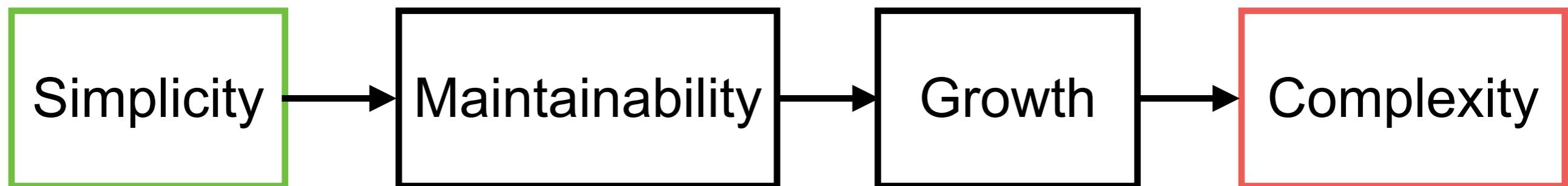
Solution
Structure

Infrastructure

Deployment
strategy



Steps of Evolutions



Let's Start with solution and structure

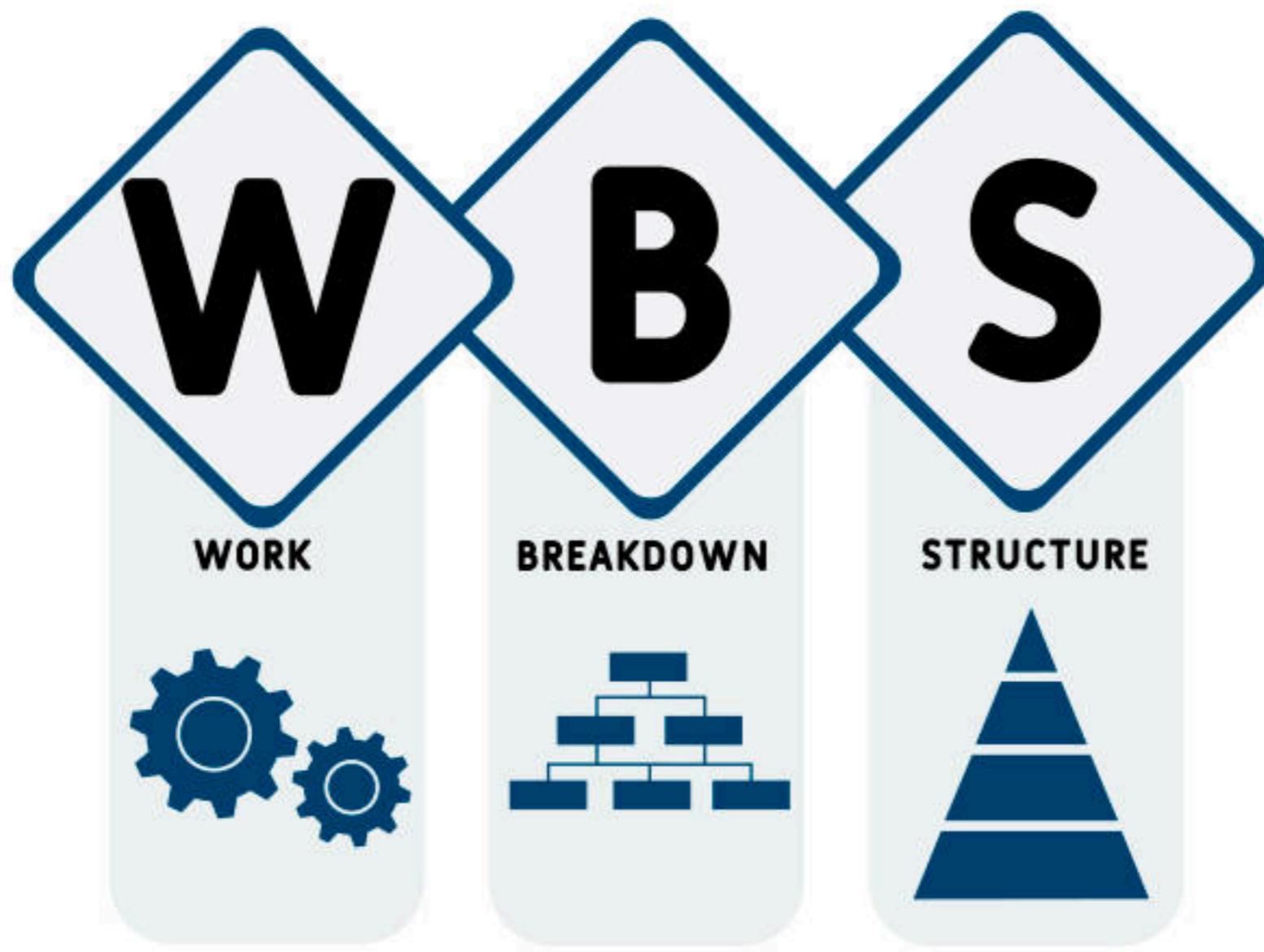


Monolith

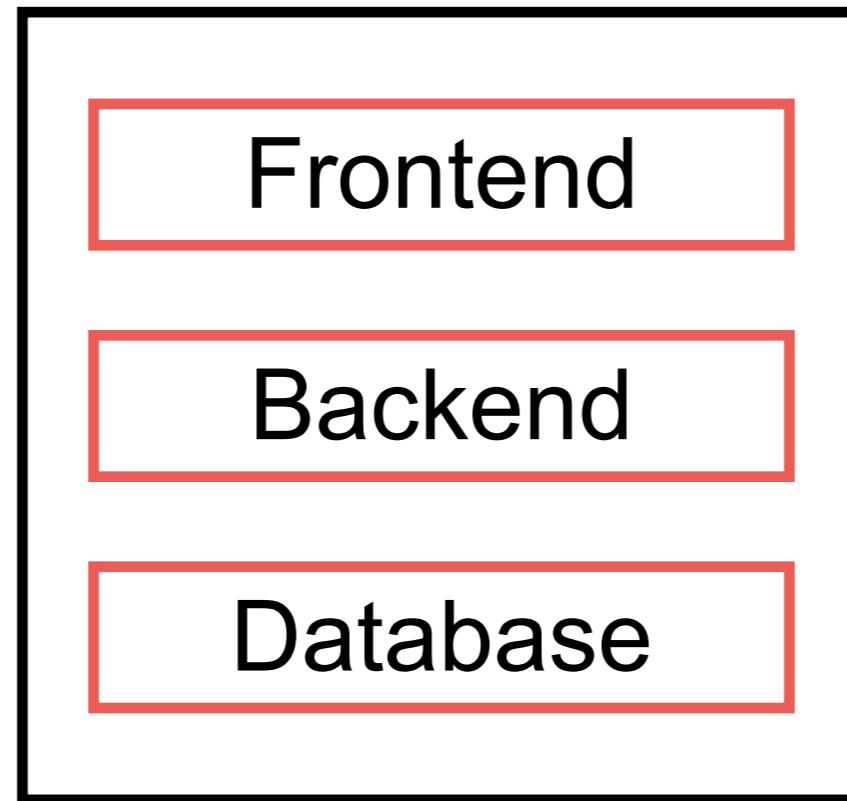
App

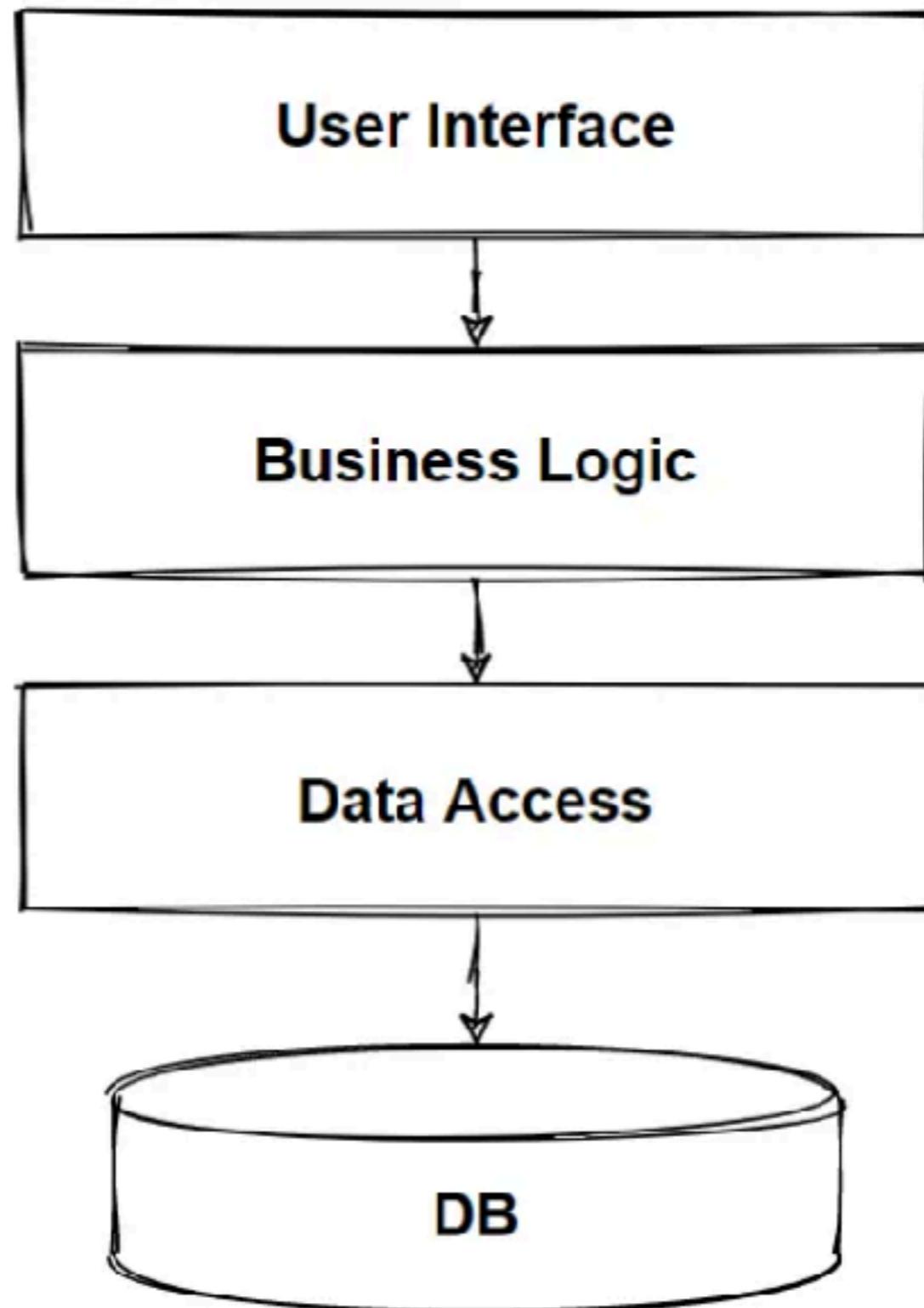


Work Breakdown Structure

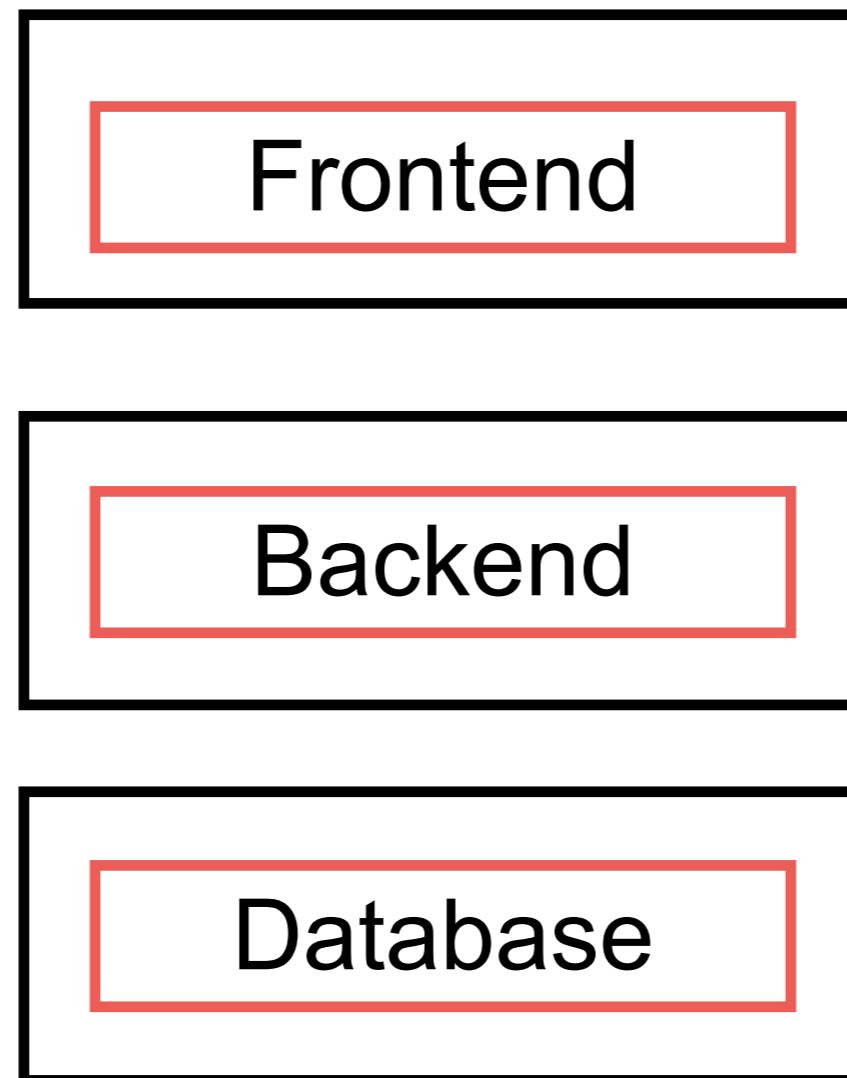


Layers (inter-process)

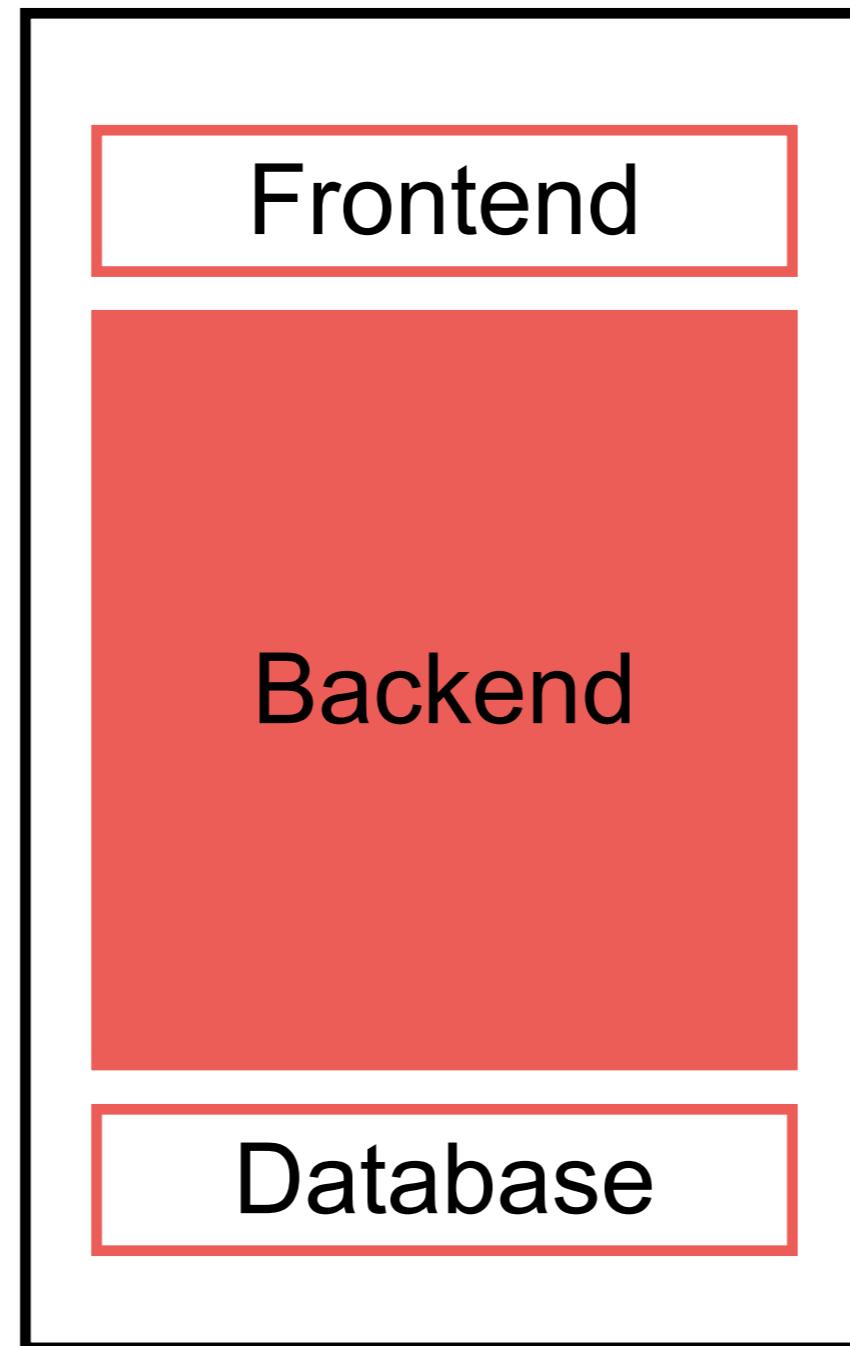




Tiers (out-of-process)



More features ...



More features ...

Frontend

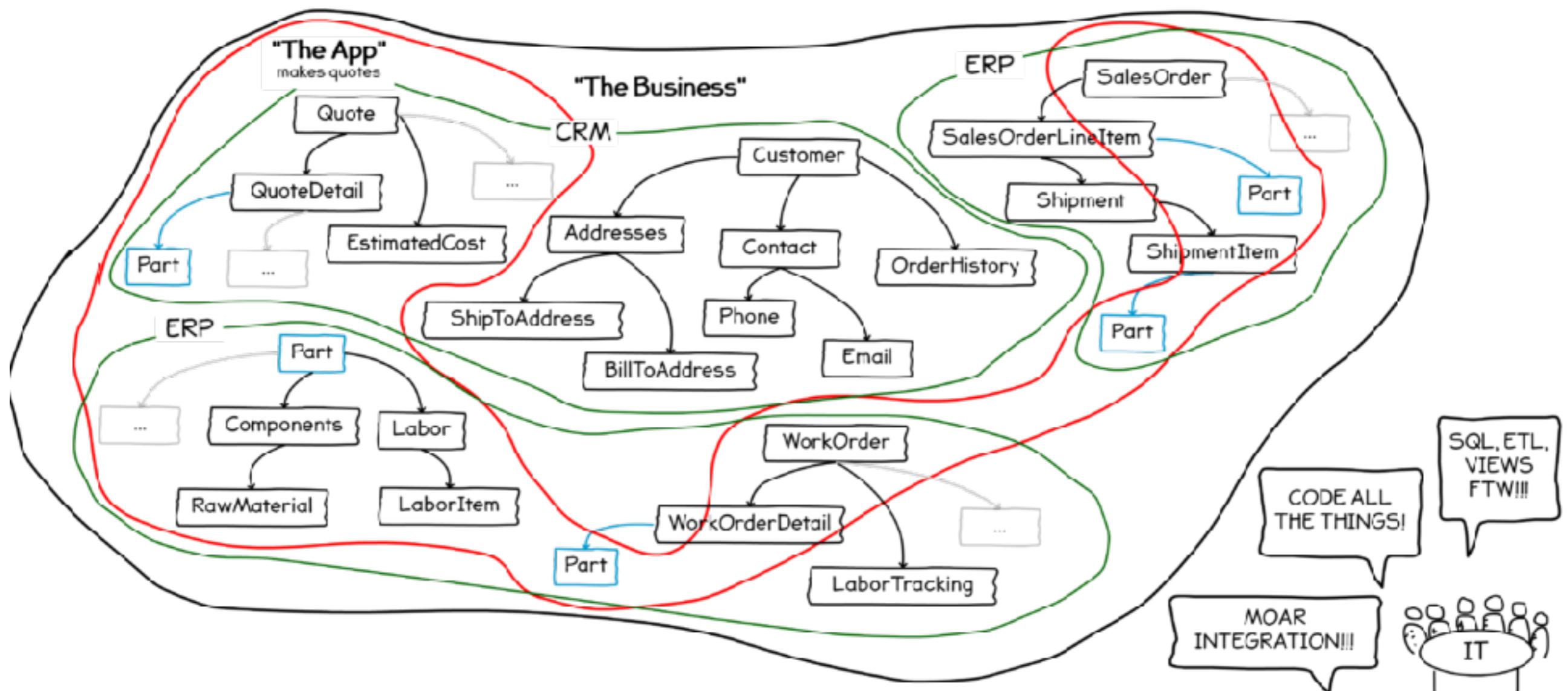


In spaghetti code, the relations between the pieces of code are so tangled that it is nearly impossible to add or change something without unpredictably breaking something somewhere else.

Database



Monolith Hell

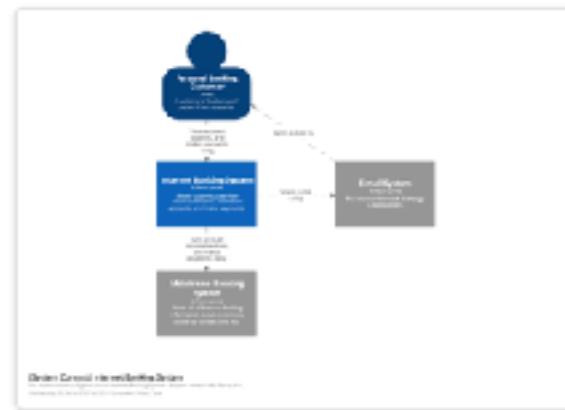
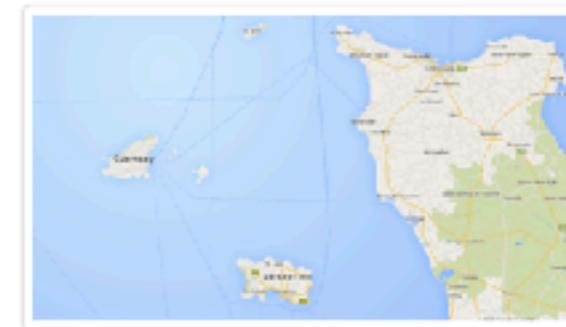


Overall architecture of system ?

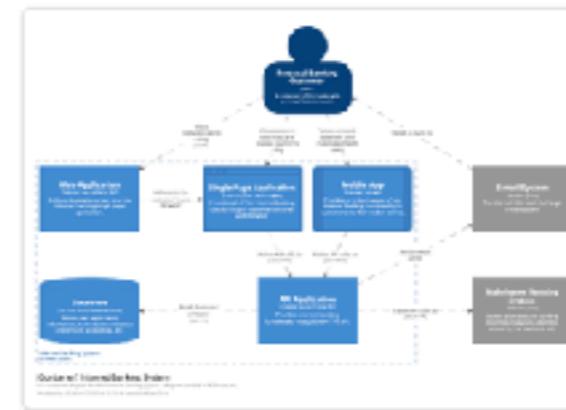


C4 Model

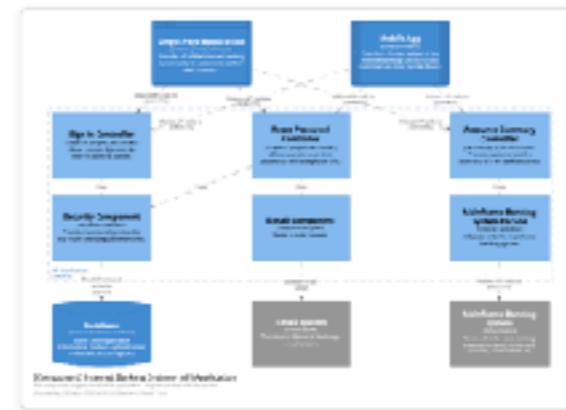
Visualizing software architecture



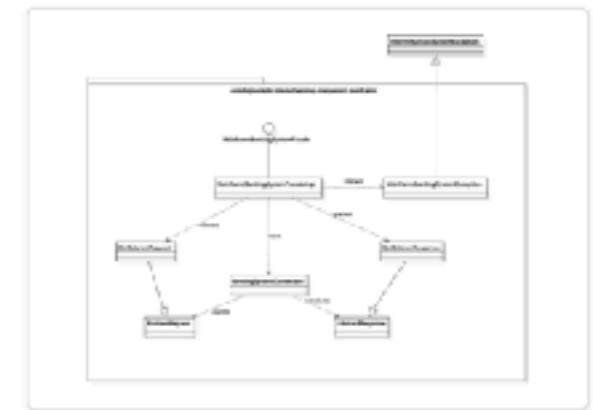
Level 1: A **System Context** diagram provides a starting point, showing how the software system in scope fits into the world around it.



Level 2: A **Container** diagram zooms into the software system in scope, showing the high-level technical building blocks.



Level 3: A **Component** diagram zooms into an individual container, showing the components inside it.

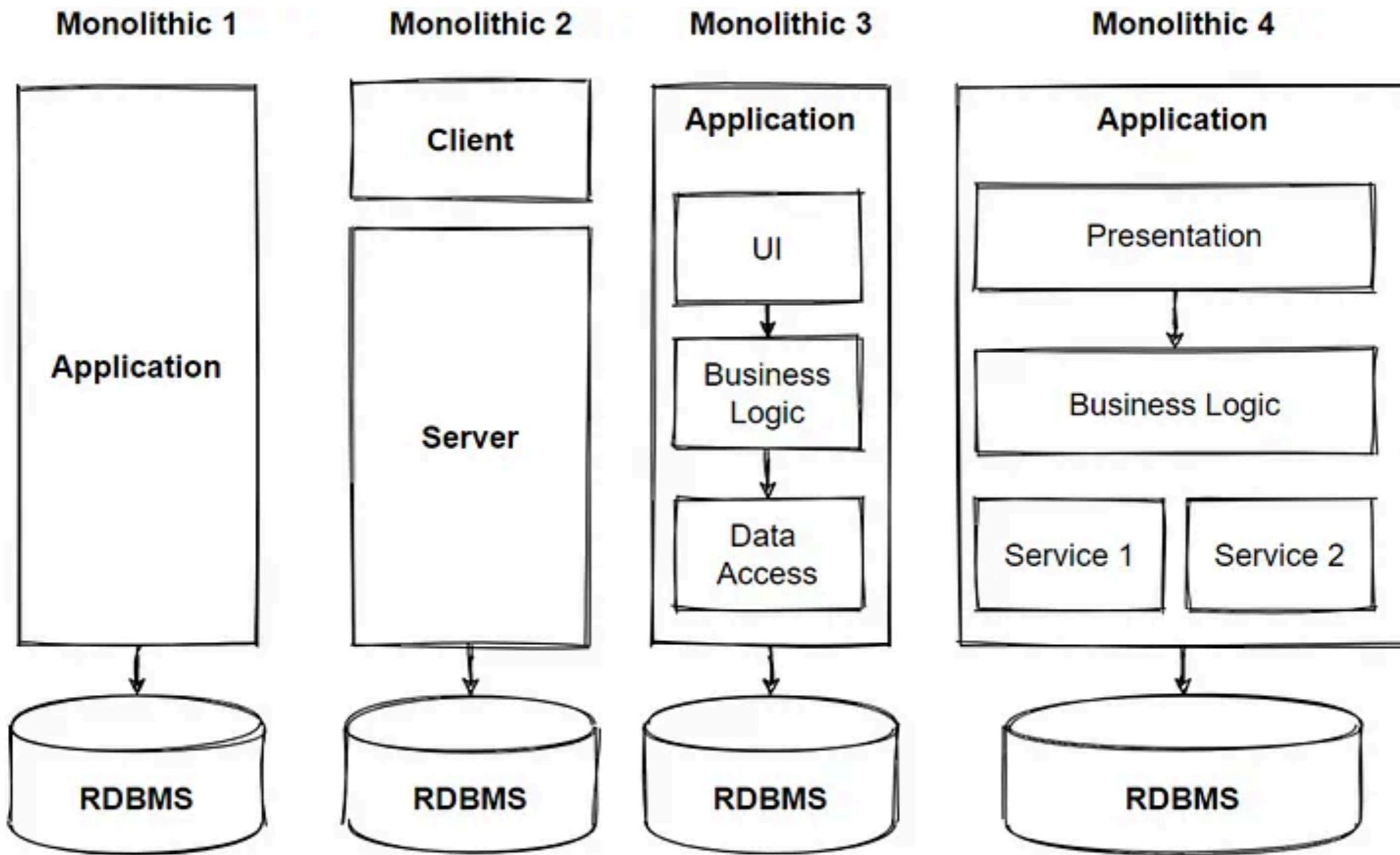


Level 4: A **code** (e.g. UML class) diagram can be used to zoom into an individual component, showing how that component is implemented.

<https://c4model.com/>



Evolution of Monolith ...



Monolith problems ?

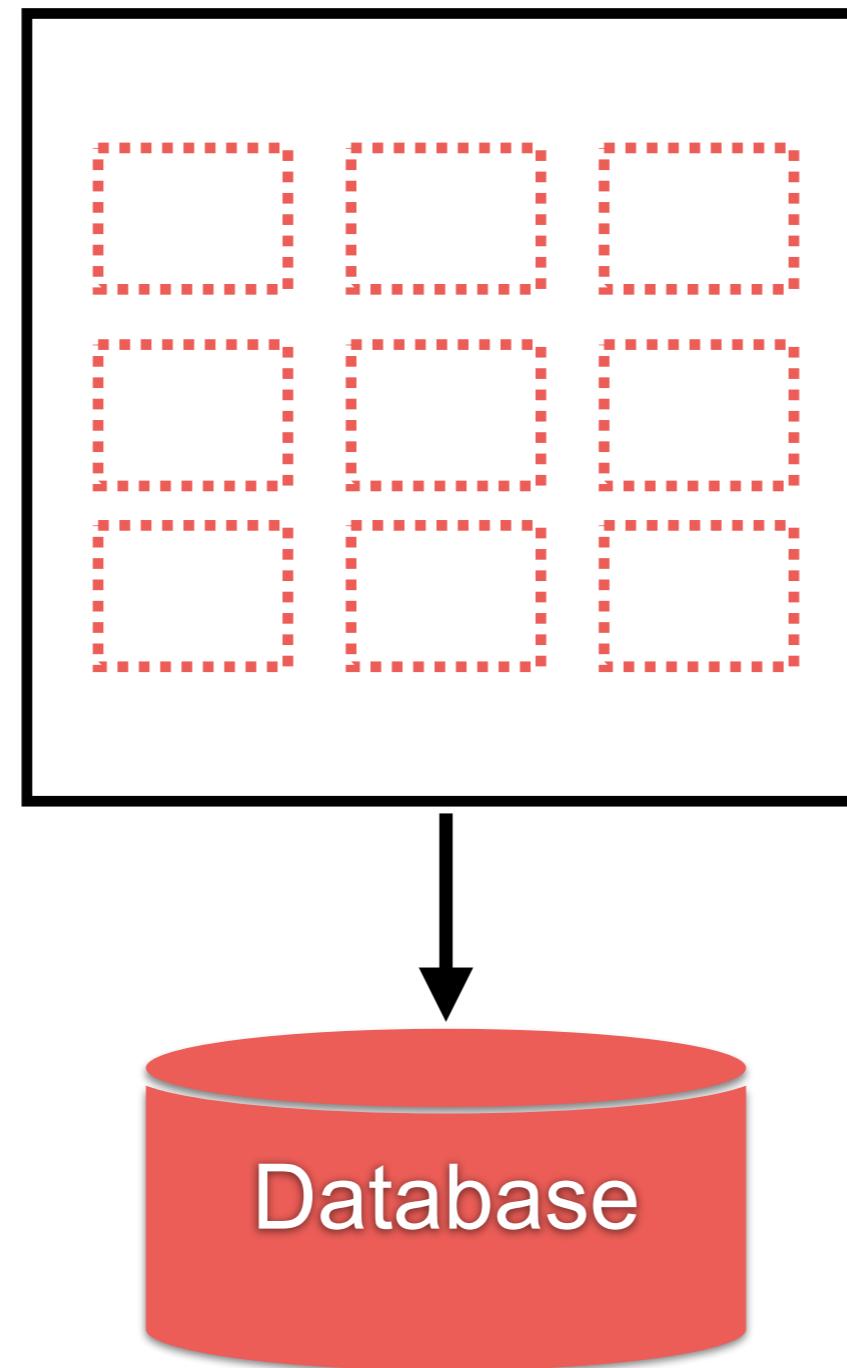
Build
Test
Deploy
Troubleshooting
Scale out (horizontal)

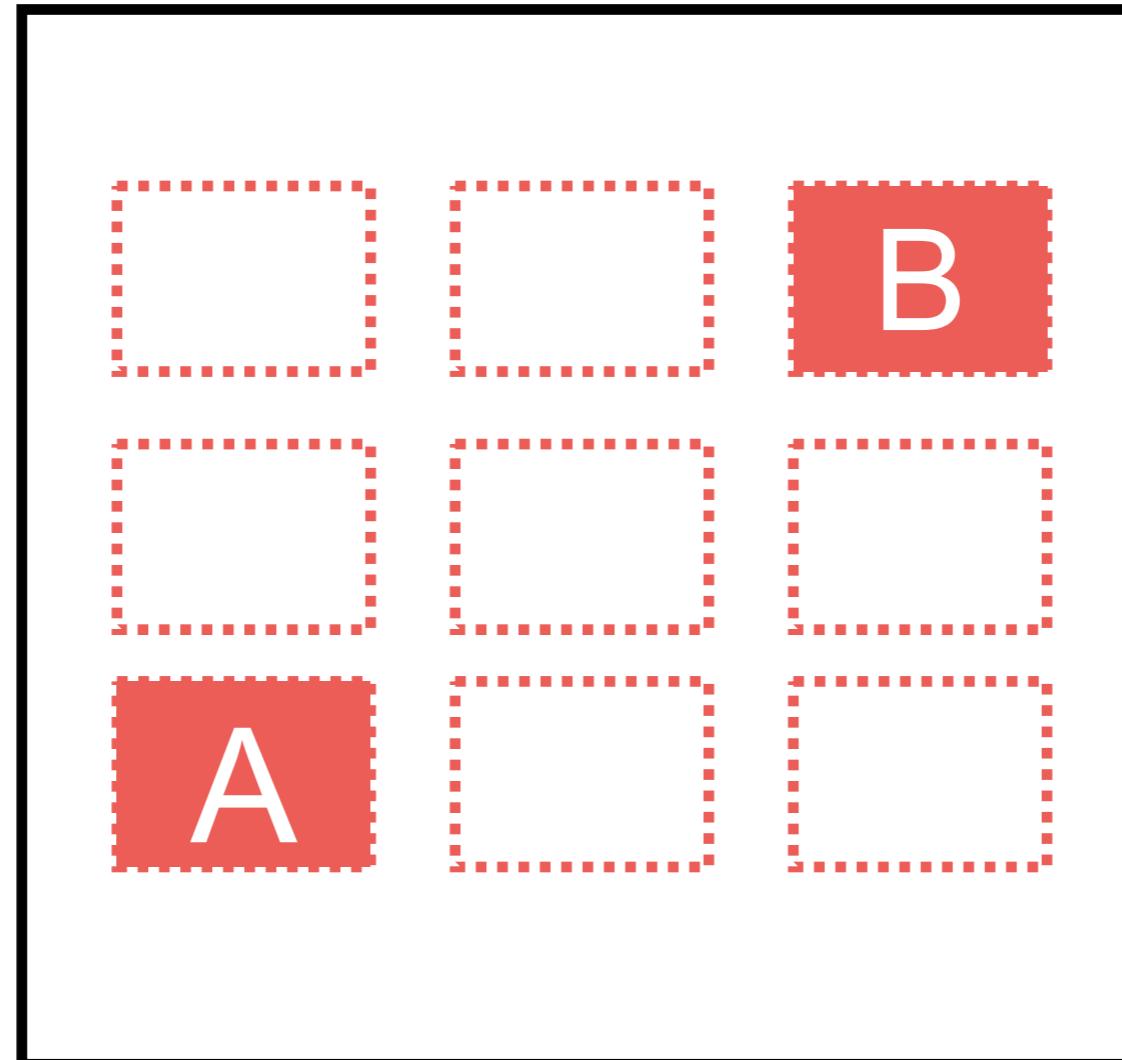


How to scale system ?



Service with Modular





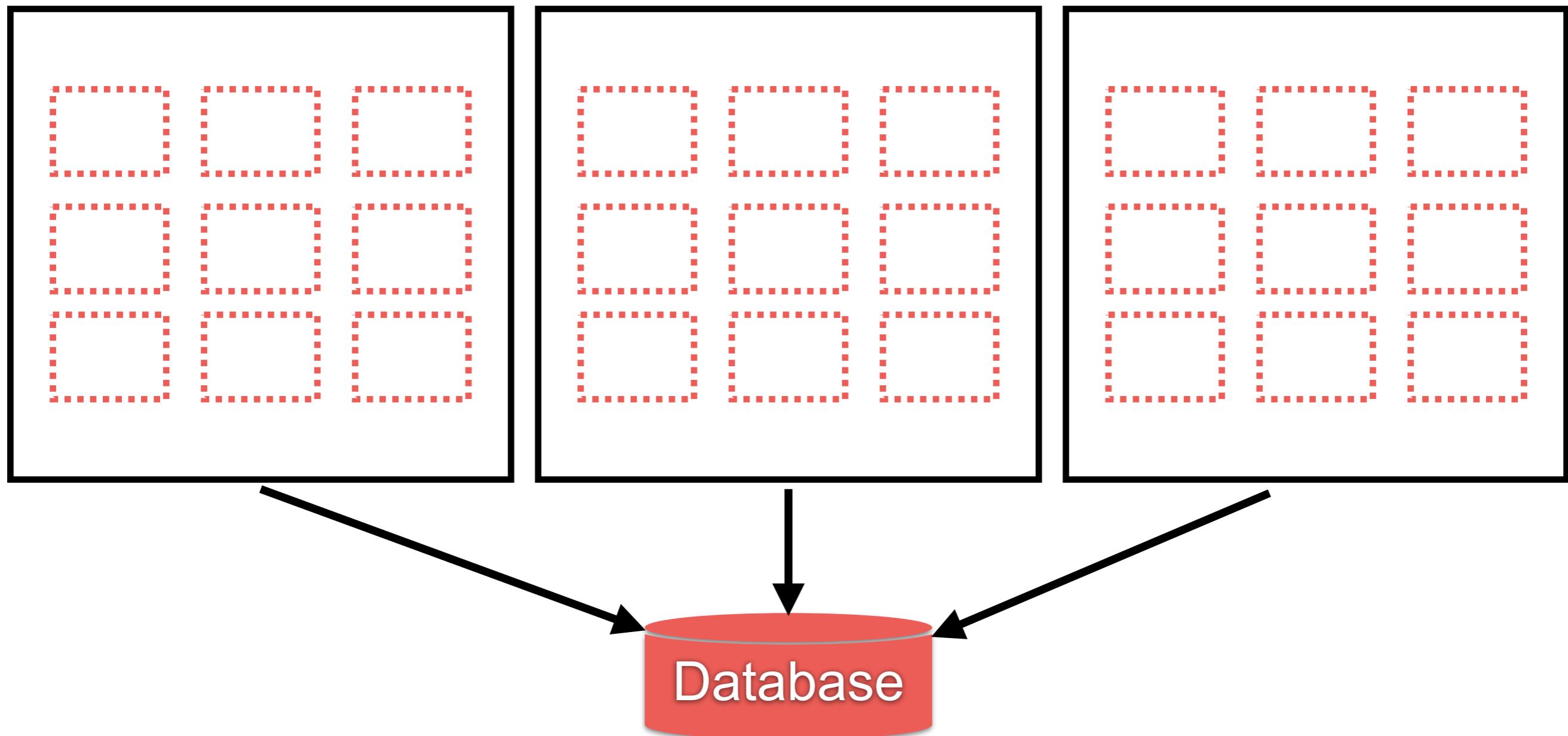
How to scale ?



Scale Up (Vertical)



Scale Out (Horizontal)



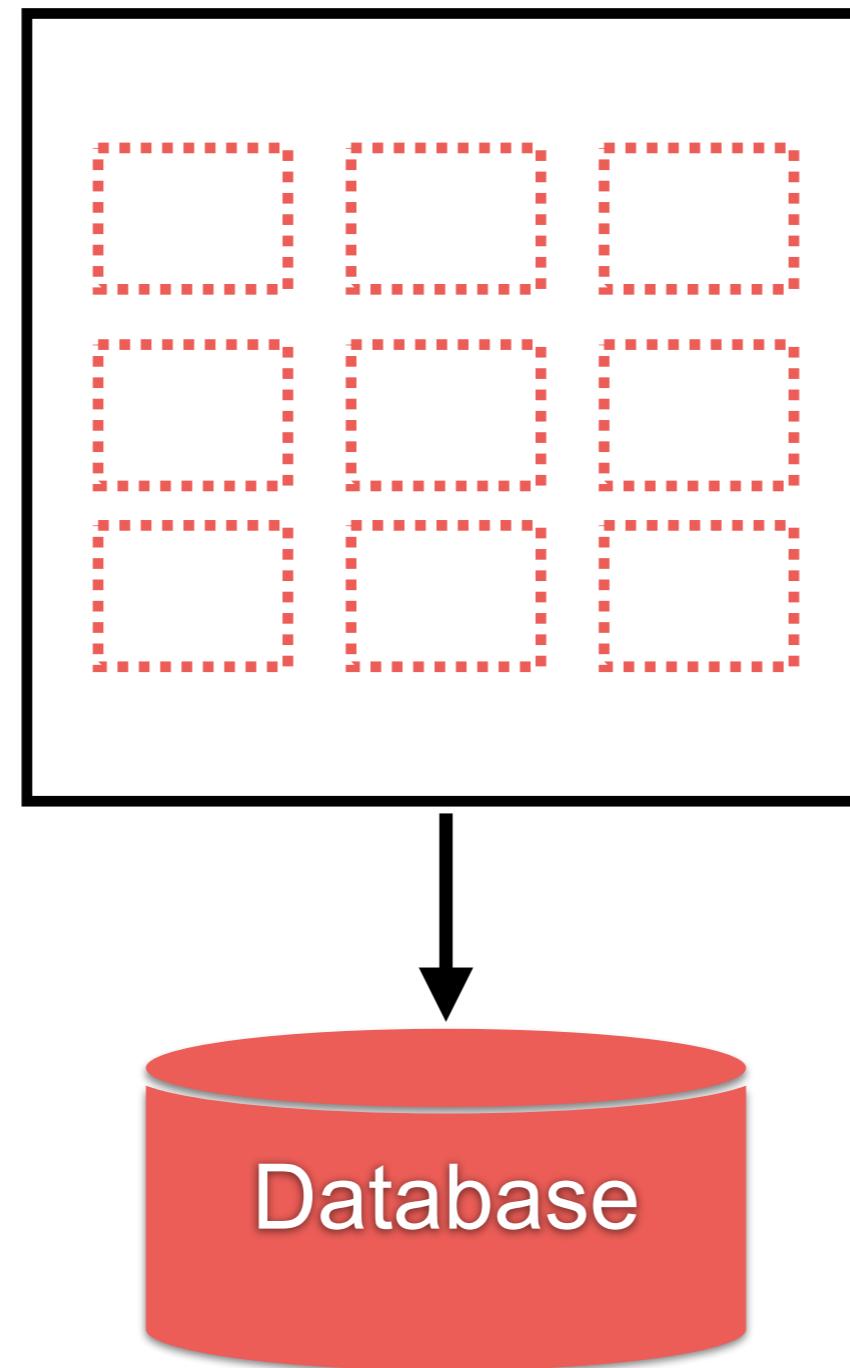
What happens when we need more servers ?



What if we don't use all modules equally ?



Database ?



How can we update individual models/database ?



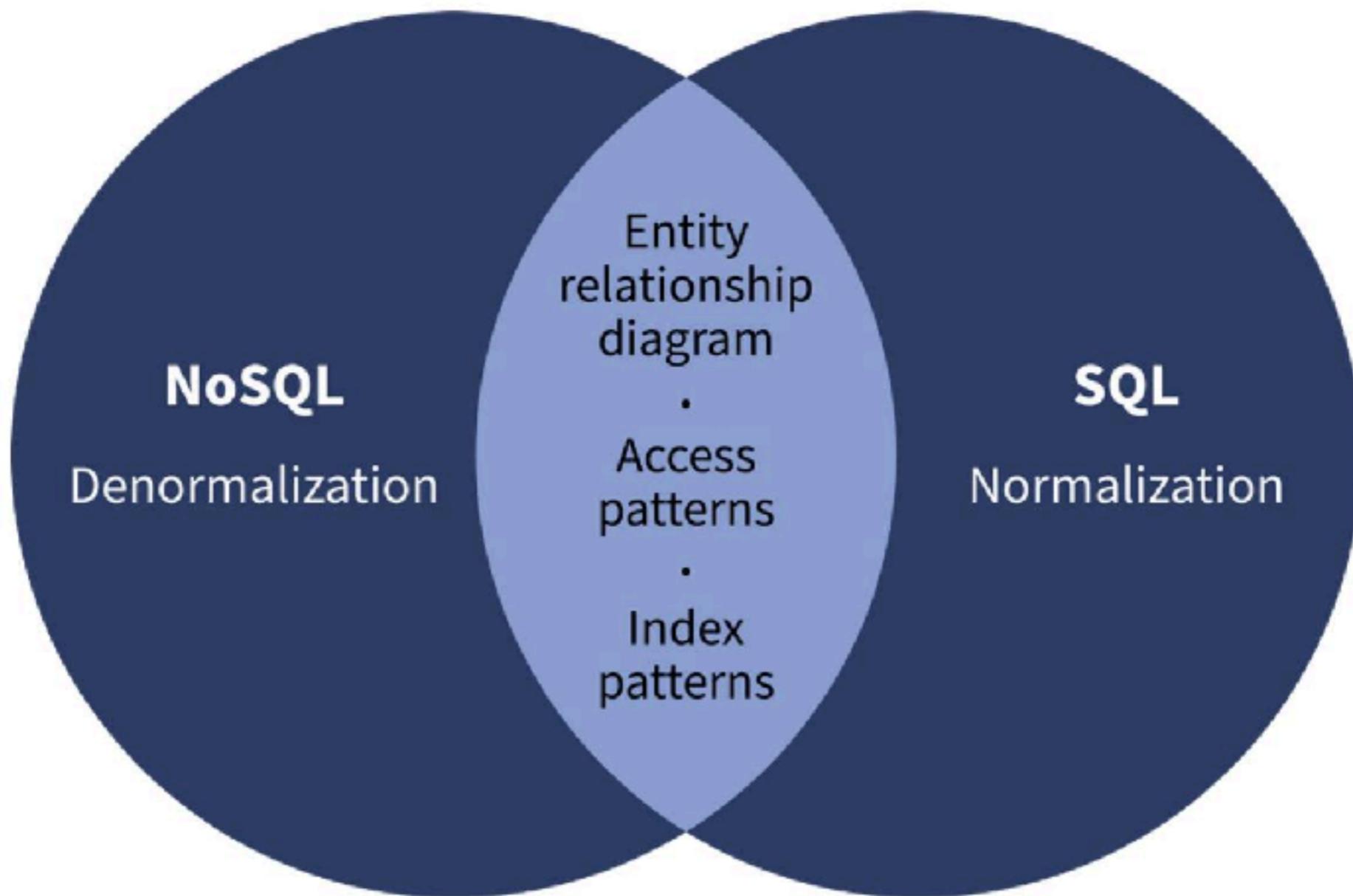
Scale up or Scale out ?



**Do all modules need to use the
same Database, Language and
Runtime ... ?**



Database ?



Database Models

422 systems in ranking, September 2023

Rank			DBMS	Database Model	Score		
Sep 2023	Aug 2023	Sep 2022			Sep 2023	Aug 2023	Sep 2022
1.	1.	1.	Oracle 	Relational, Multi-model 	1240.88	-1.22	+2.62
2.	2.	2.	MySQL 	Relational, Multi-model 	1111.49	-18.97	-100.98
3.	3.	3.	Microsoft SQL Server 	Relational, Multi-model 	902.22	-18.60	-24.08
4.	4.	4.	PostgreSQL 	Relational, Multi-model 	620.75	+0.37	+0.29
5.	5.	5.	MongoDB 	Document, Multi-model 	439.42	+4.93	-50.21
6.	6.	6.	Redis 	Key-value, Multi-model 	163.68	+0.72	-17.79
7.	7.	7.	Elasticsearch	Search engine, Multi-model 	138.98	-0.94	-12.46
8.	8.	8.	IBM Db2	Relational, Multi-model 	136.72	-2.52	-14.67
9.	↑ 10.	↑ 10.	SQLite 	Relational	129.20	-0.72	-9.62
10.	↓ 9.	↓ 9.	Microsoft Access	Relational	128.56	-1.78	-11.47
11.	11.	↑ 13.	Snowflake 	Relational	120.89	+0.27	+17.39
12.	12.	↓ 11.	Cassandra 	Wide column, Multi-model 	110.06	+2.67	-9.06
13.	13.	↓ 12.	MariaDB 	Relational, Multi-model 	100.45	+1.80	-9.70

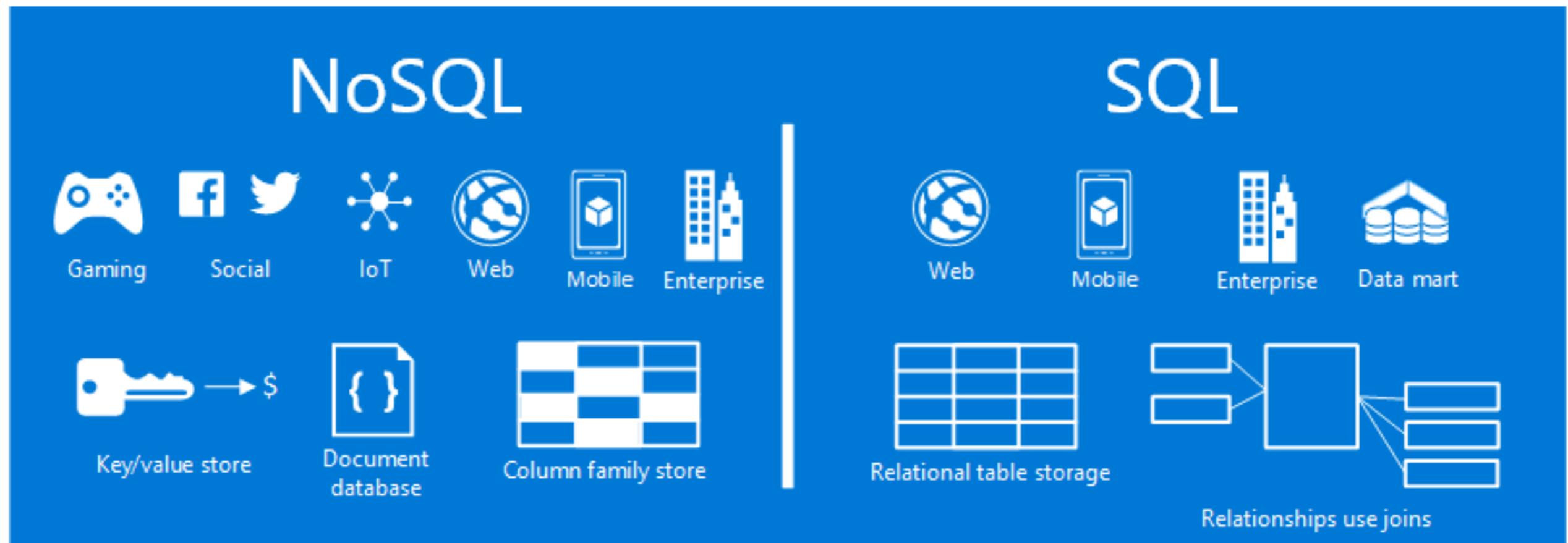
<https://db-engines.com/en/ranking>



Microservices

© 2020 - 2023 Siam Chamnankit Company Limited. All rights reserved.

Database Models



Database design

Design for read

Design for write

Pre-join tables or pre-aggregate data

Manage data size

Caching data



We need to improve

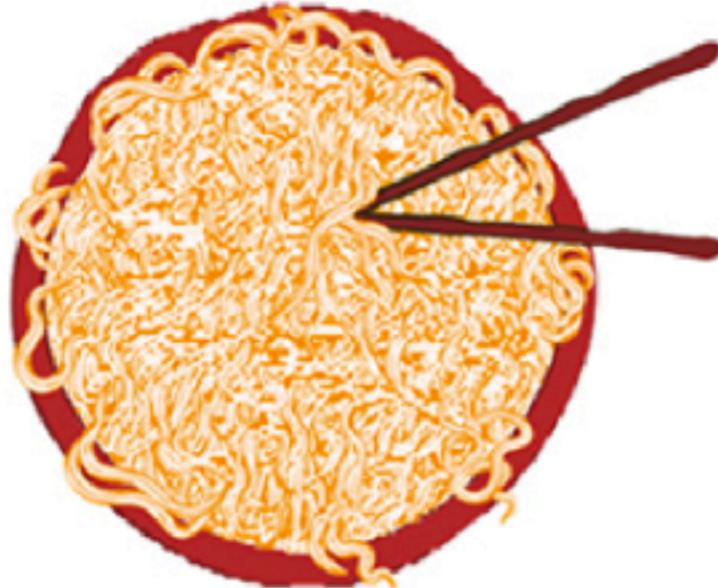
We need to get better



SOA

1990s and earlier

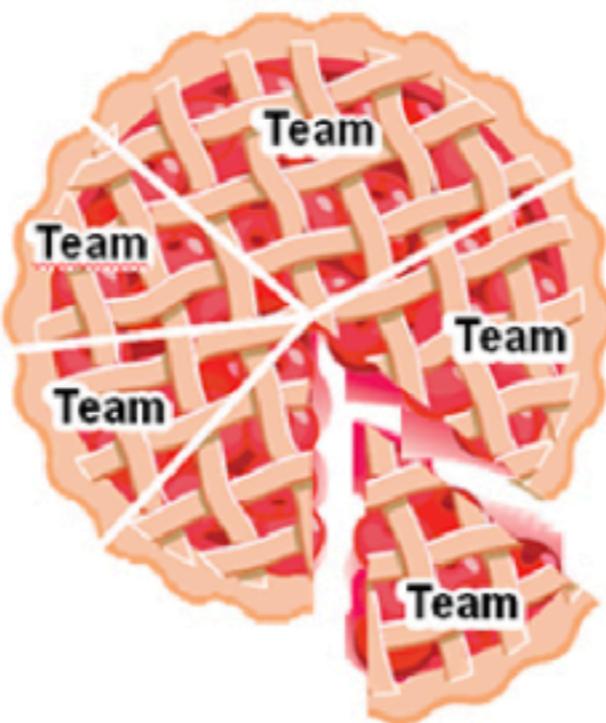
Pre-SOA (monolithic)
Tight coupling



For a monolith to change, all must agree on each change. Each change has unanticipated effects requiring careful testing beforehand.

2000s

Traditional SOA
Looser coupling



Elements in SOA are developed more autonomously but must be coordinated with others to fit into the overall design.

2010s

Microservices
Decoupled



Developers can create and activate new microservices without prior coordination with others. Their adherence to MSA principles makes continuous delivery of new or modified services possible.



Service-Oriented Architecture



What is service ?

Standalone and loosely couple

Independently deployable software component

Implement some useful functionality



Service-Oriented Architecture

SOA Manifesto

Service orientation is a paradigm that frames what you do.
Service-oriented architecture (SOA) is a type of architecture
that results from applying service orientation.

We have been applying service orientation to help organizations
consistently deliver sustainable business value, with increased agility
and cost effectiveness, in line with changing business needs.

<http://www.soa-manifesto.org/>



Service-Oriented Architecture

Through our work we have come to prioritize:

Business value over technical strategy

Strategic goals over project-specific benefits

Intrinsic interoperability over custom integration

Shared services over specific-purpose implementations

Flexibility over optimization

Evolutionary refinement over pursuit of initial perfection

<http://www.soa-manifesto.org/>



Microservices

© 2020 - 2023 Siam Chamnankit Company Limited. All rights reserved.

Service-Oriented Architecture

Through our work we have come to prioritize:

Business value over technical strategy

Strategic goals over project-specific benefits

Intrinsic interoperability over custom integration

Shared services over specific-purpose implementations

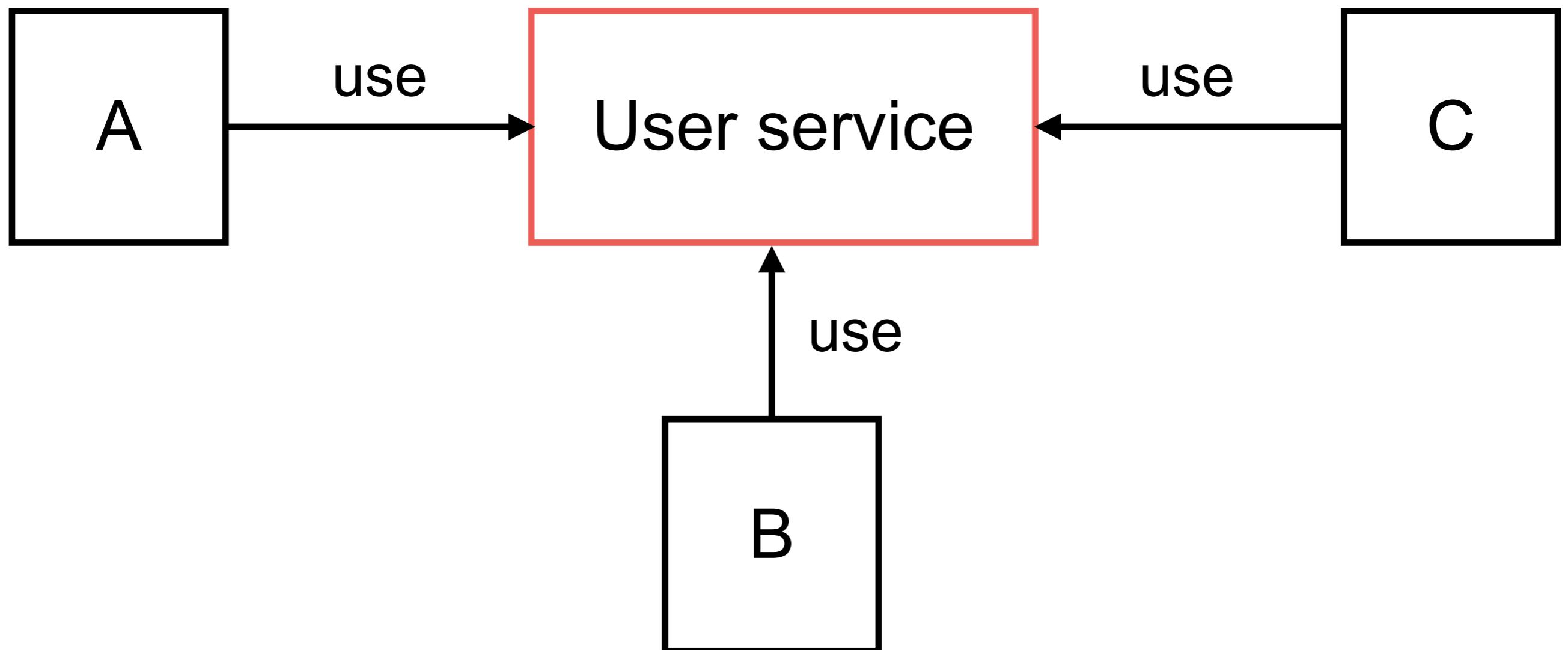
Flexibility over optimization

Evolutionary refinement over pursuit of initial perfection

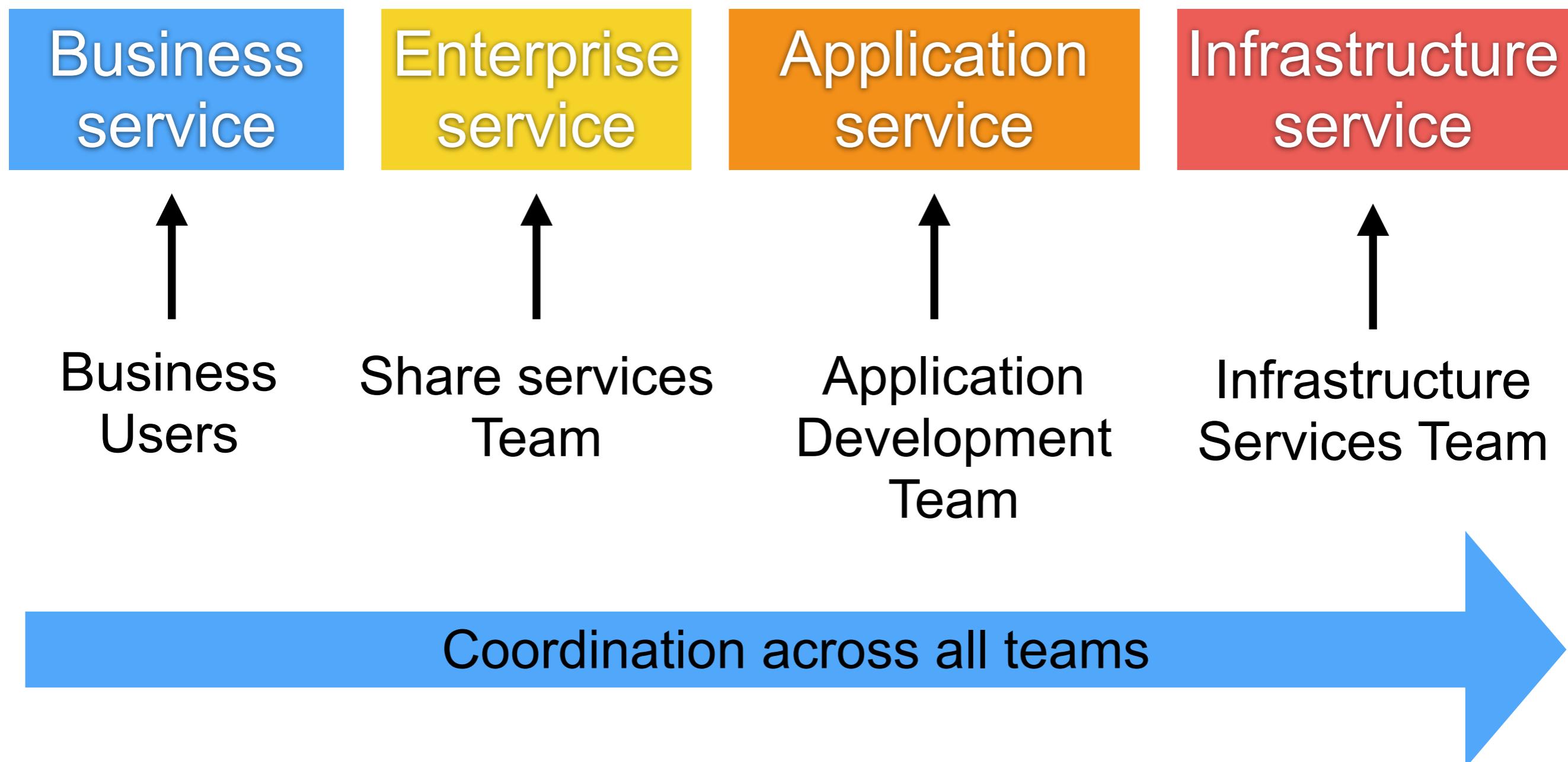
<http://www.soa-manifesto.org/>



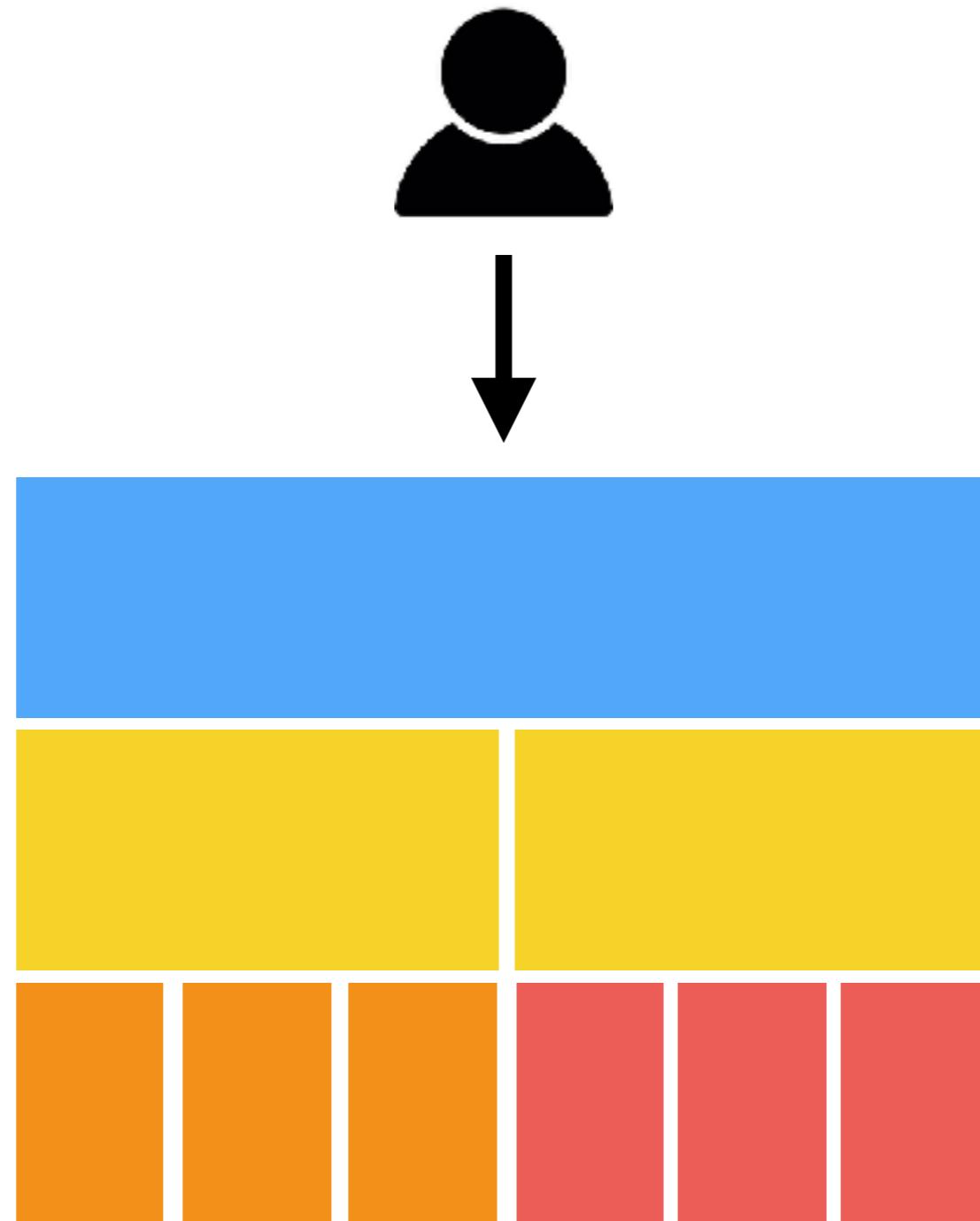
Shared service ?



Service-Oriented Architecture



Service-Oriented Architecture



We need to improve

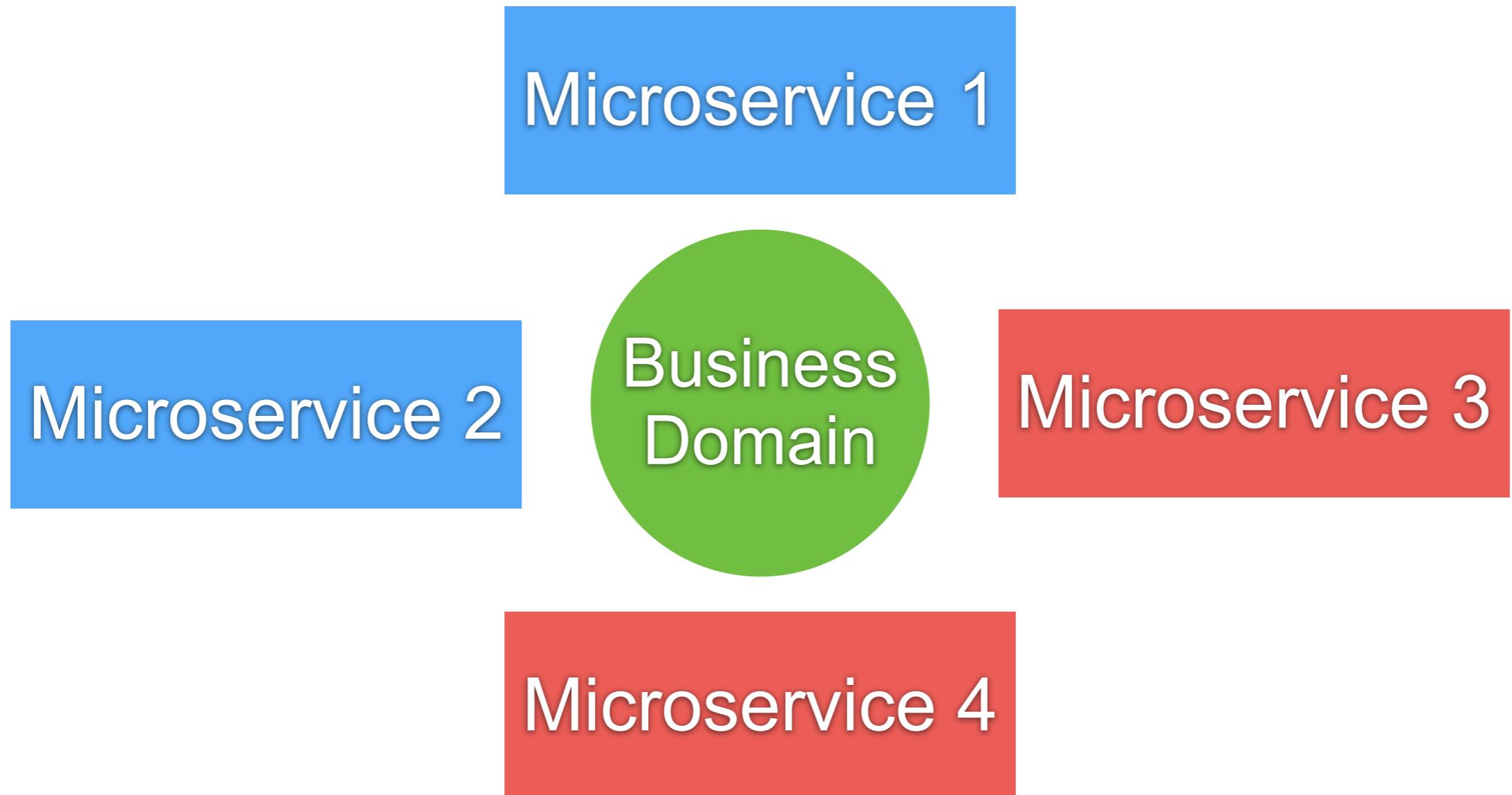
We need to get better



Microservice



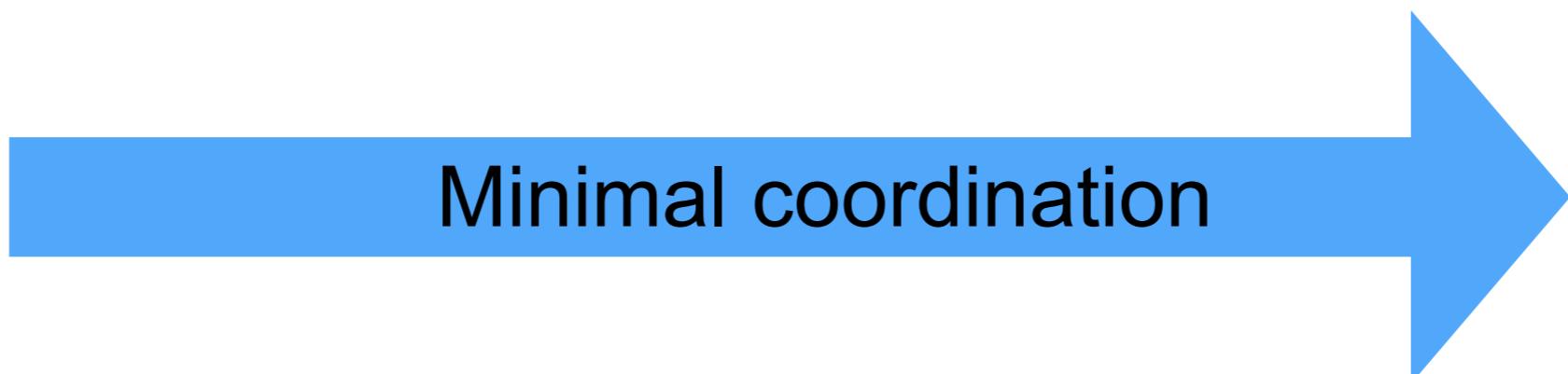
Microservice



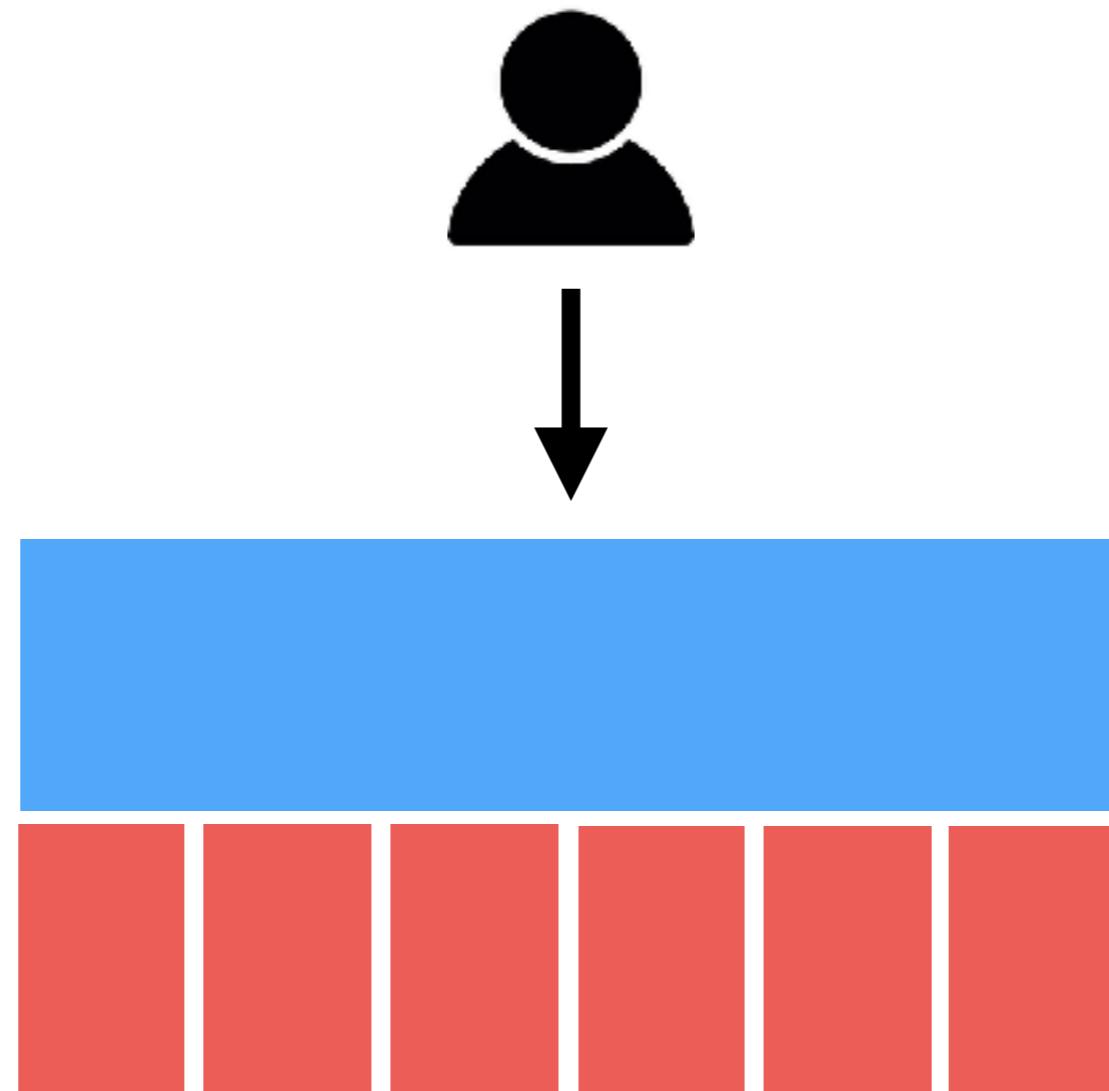
Microservice



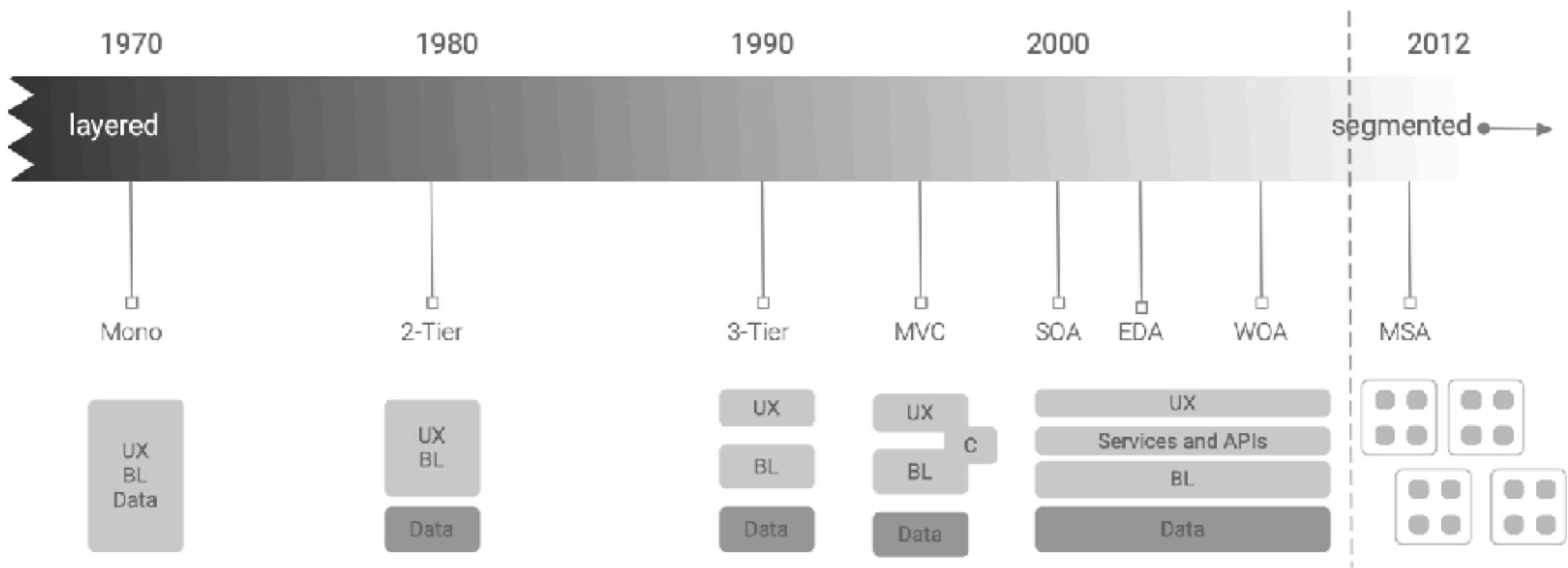
Application Development Team



Microservice



Summary Evolution of Architecture



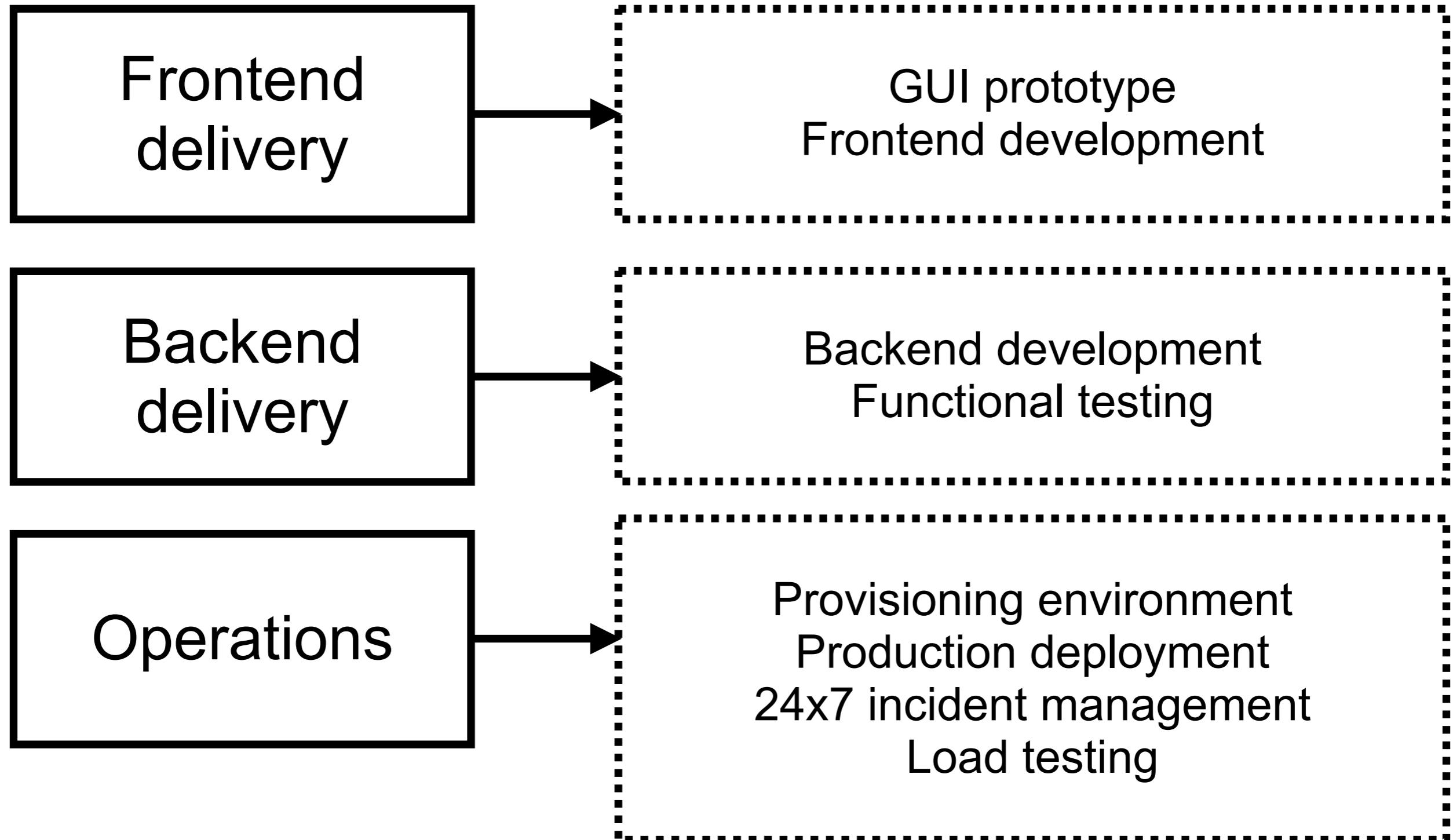
<https://github.com/wso2/reference-architecture/blob/master/reference-architecture-layered-segmented.md>



Delivery Problems



Delivery Responsibilities



Delivery Improvement ?



Better ?

Delivery Team

Frontend
delivery

Backend
delivery

Operations

GUI prototype
Frontend development
Backend development

Functional testing
Provisioning environment
Incident management (office hours)

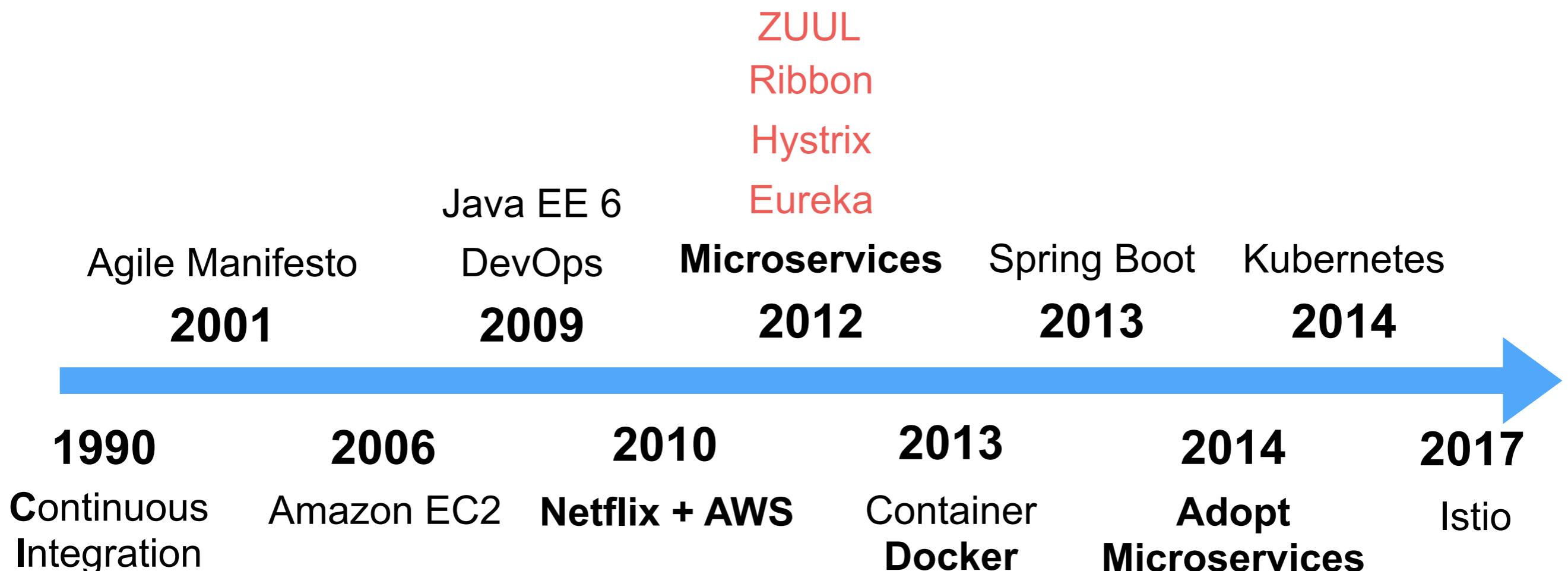
Incident management (office hours)
Load testing
Production deployment
Self-service platform



Microservice History



Microservice History



Microservice

Small, Do one thing (Single Responsibility)

Modular

Easy to understand

Easy to develop

Easy to deploy

Easy to maintain

Scale independently



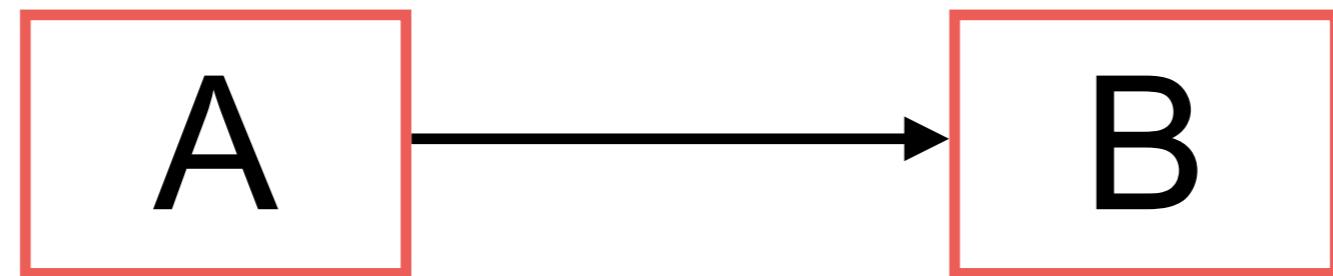
Goals

Increase the **velocity** of application release
By decomposing app to **small services**
Autonomous services
Deploy independently



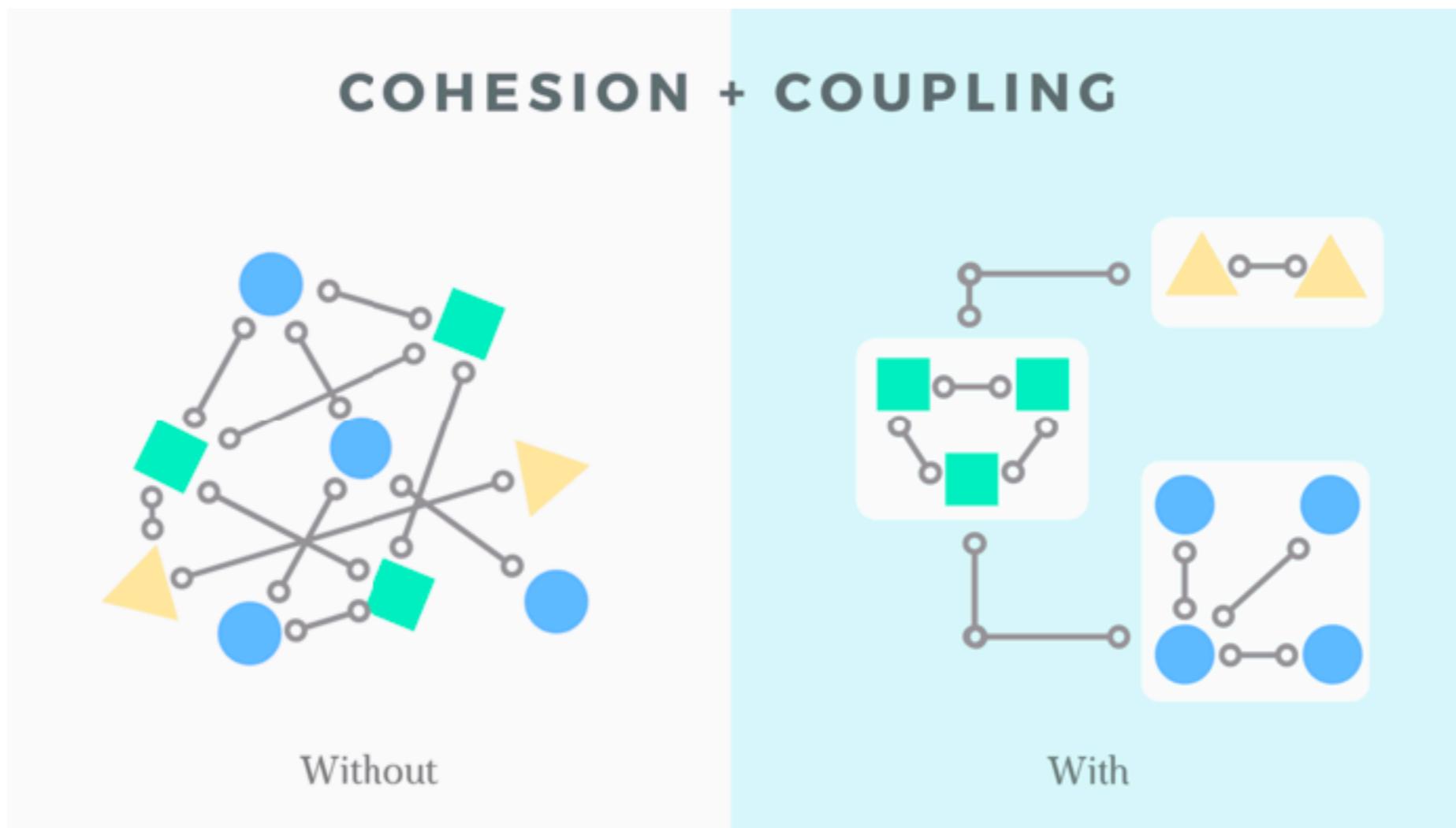
Independent

Replicable Updatable Scalable Deployable

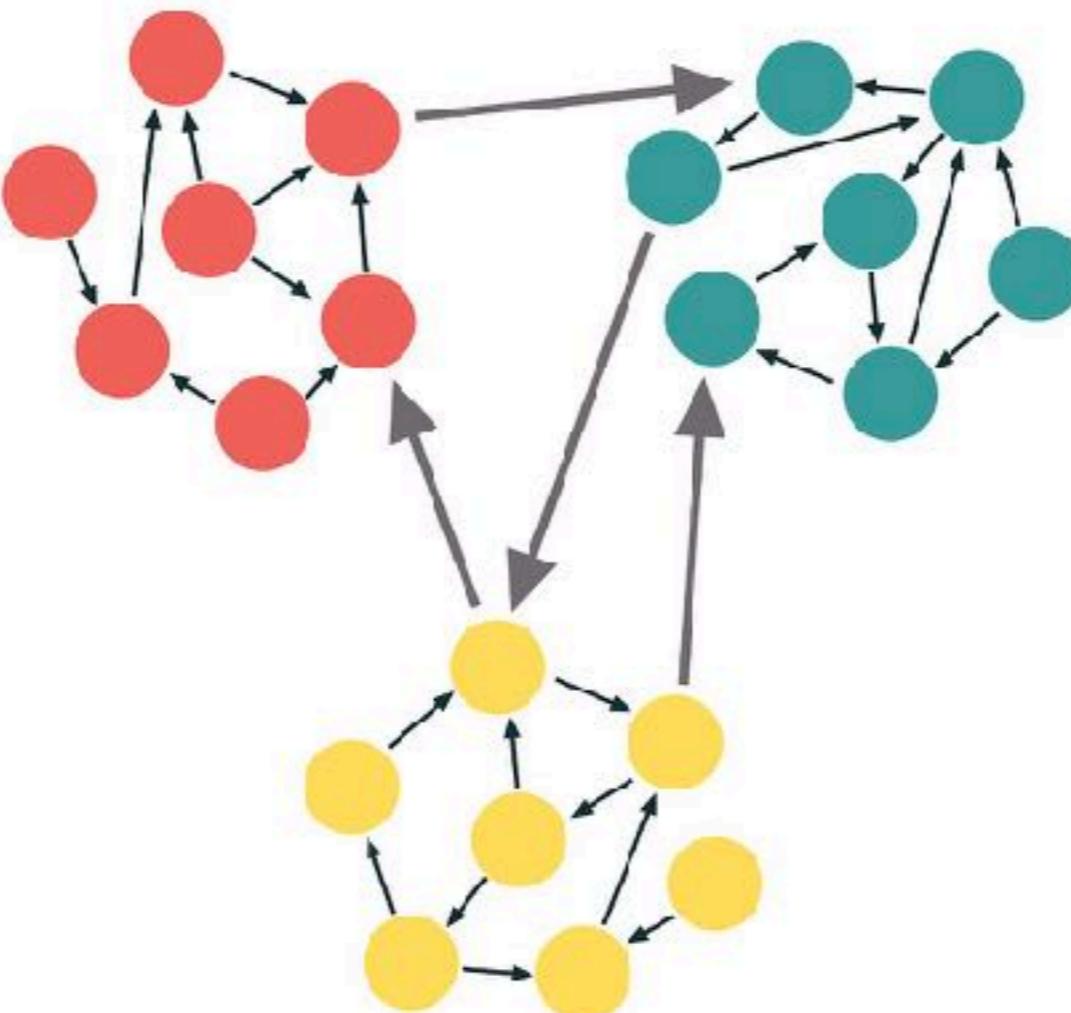


Better Architecture

Loose coupling
High cohesion



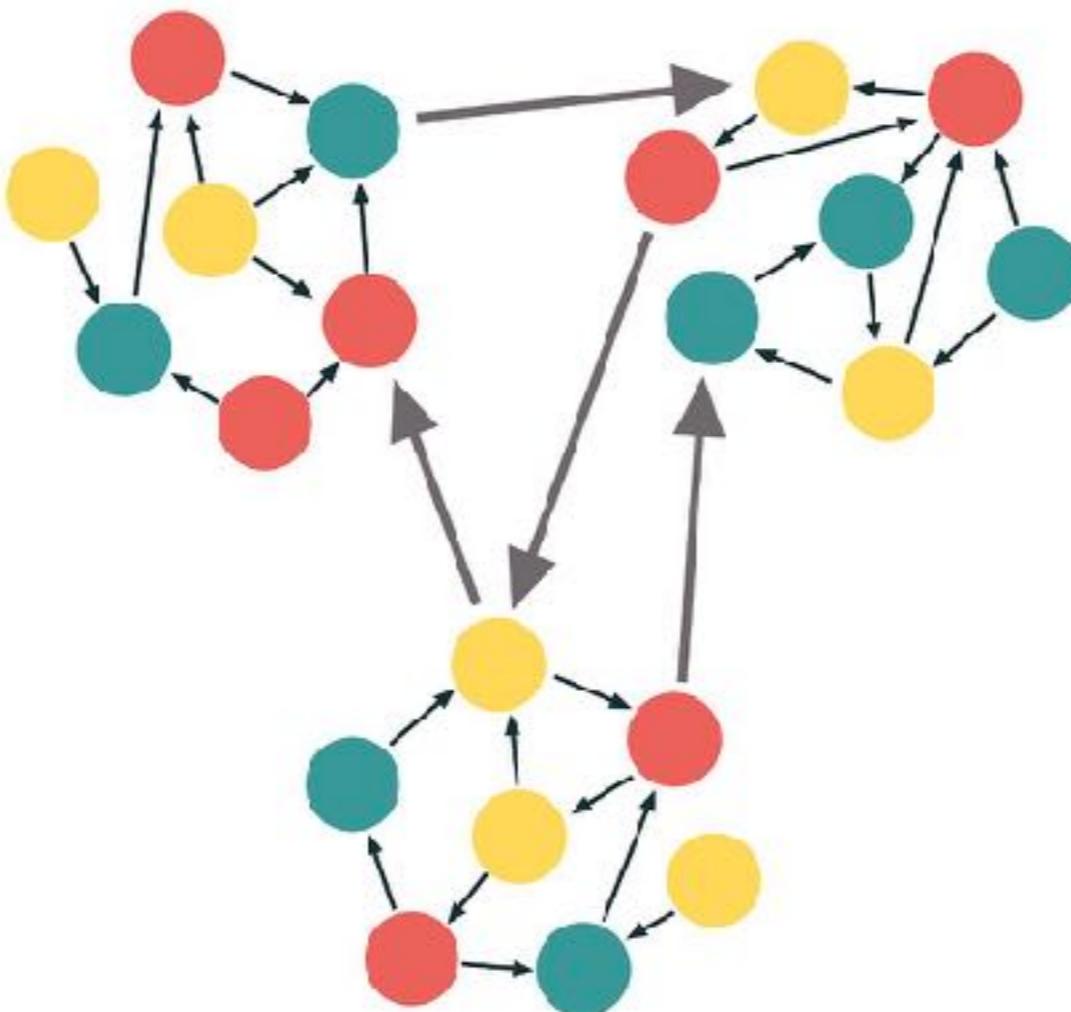
Coupling (Idea)



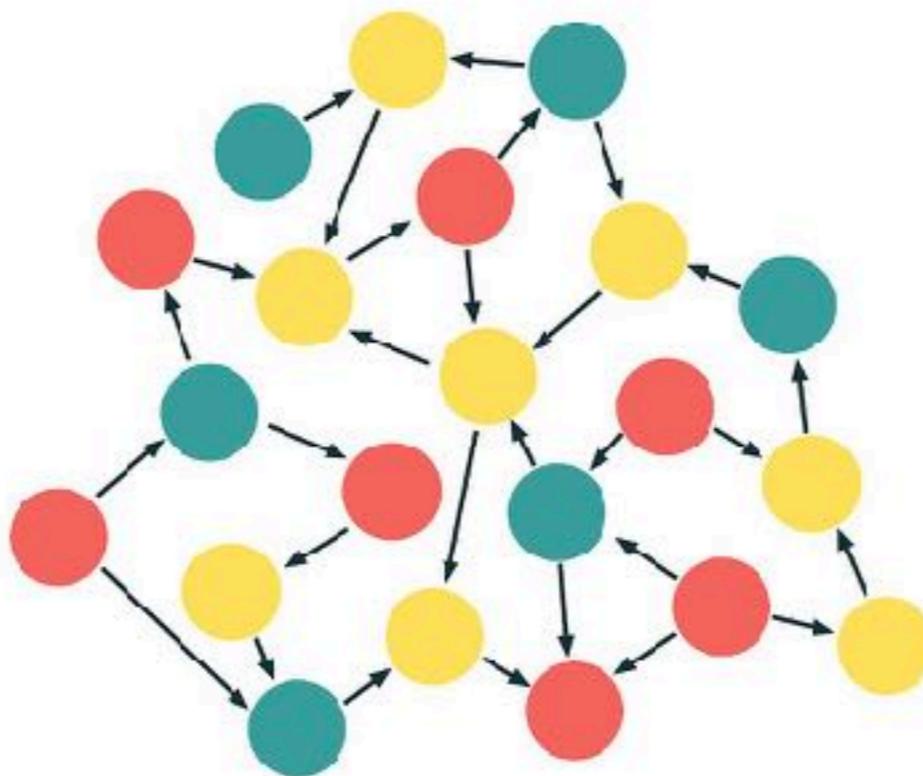
<https://enterprisecraftsmanship.com/posts/cohesion-coupling-difference/>



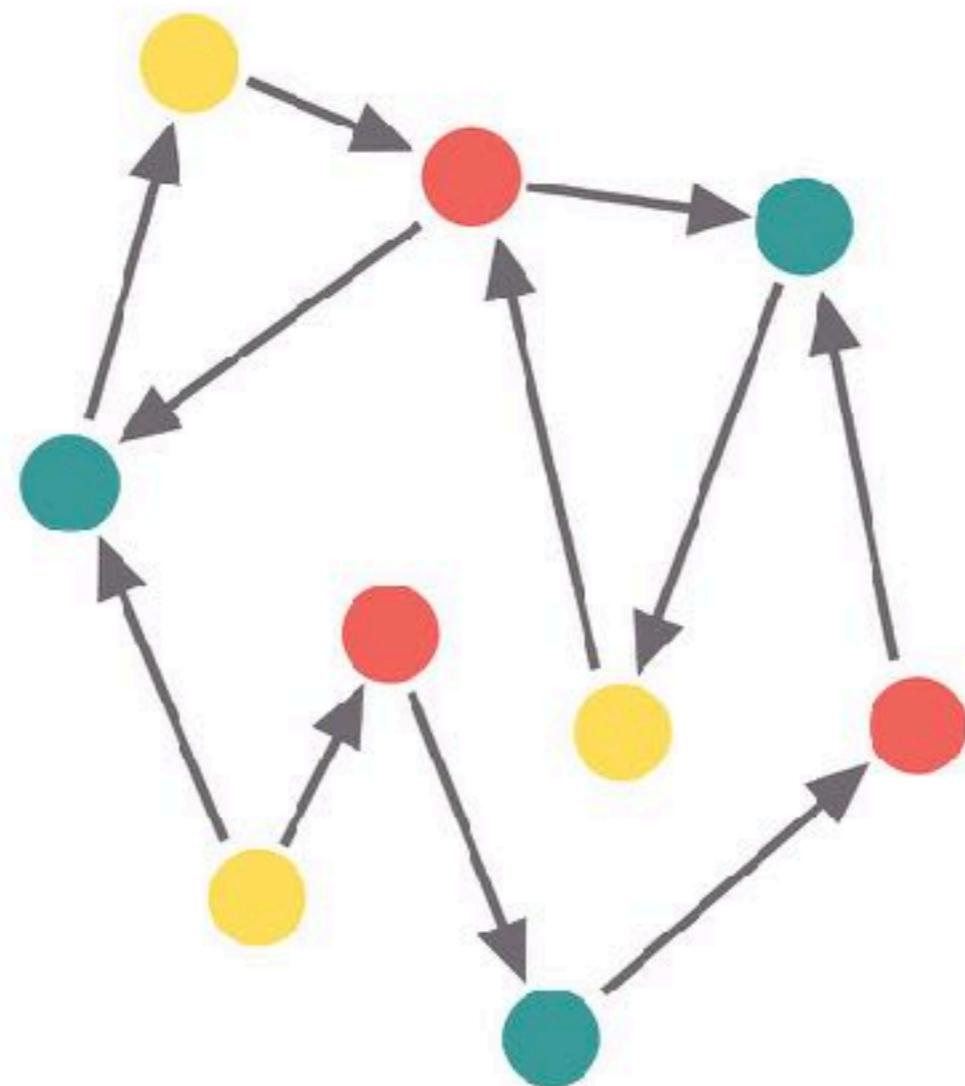
Coupling (Poorly boundary)



Coupling (God)

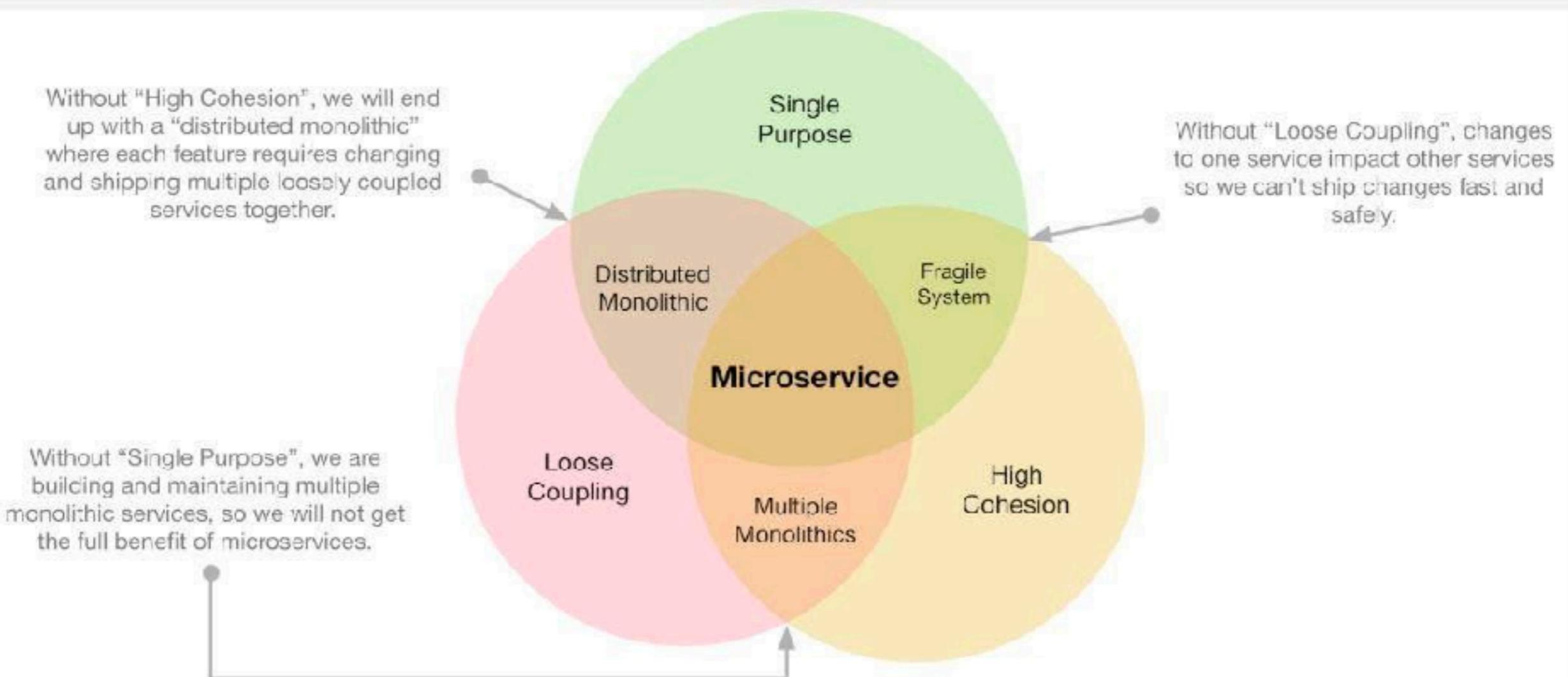


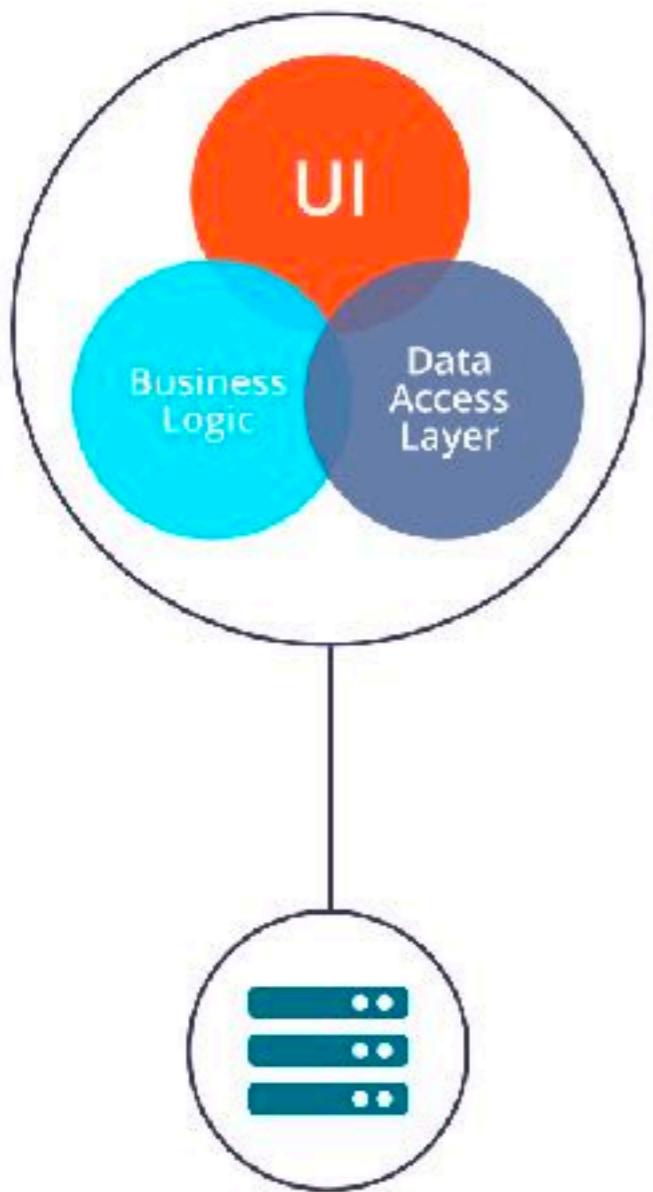
Coupling (Destructive decouple)



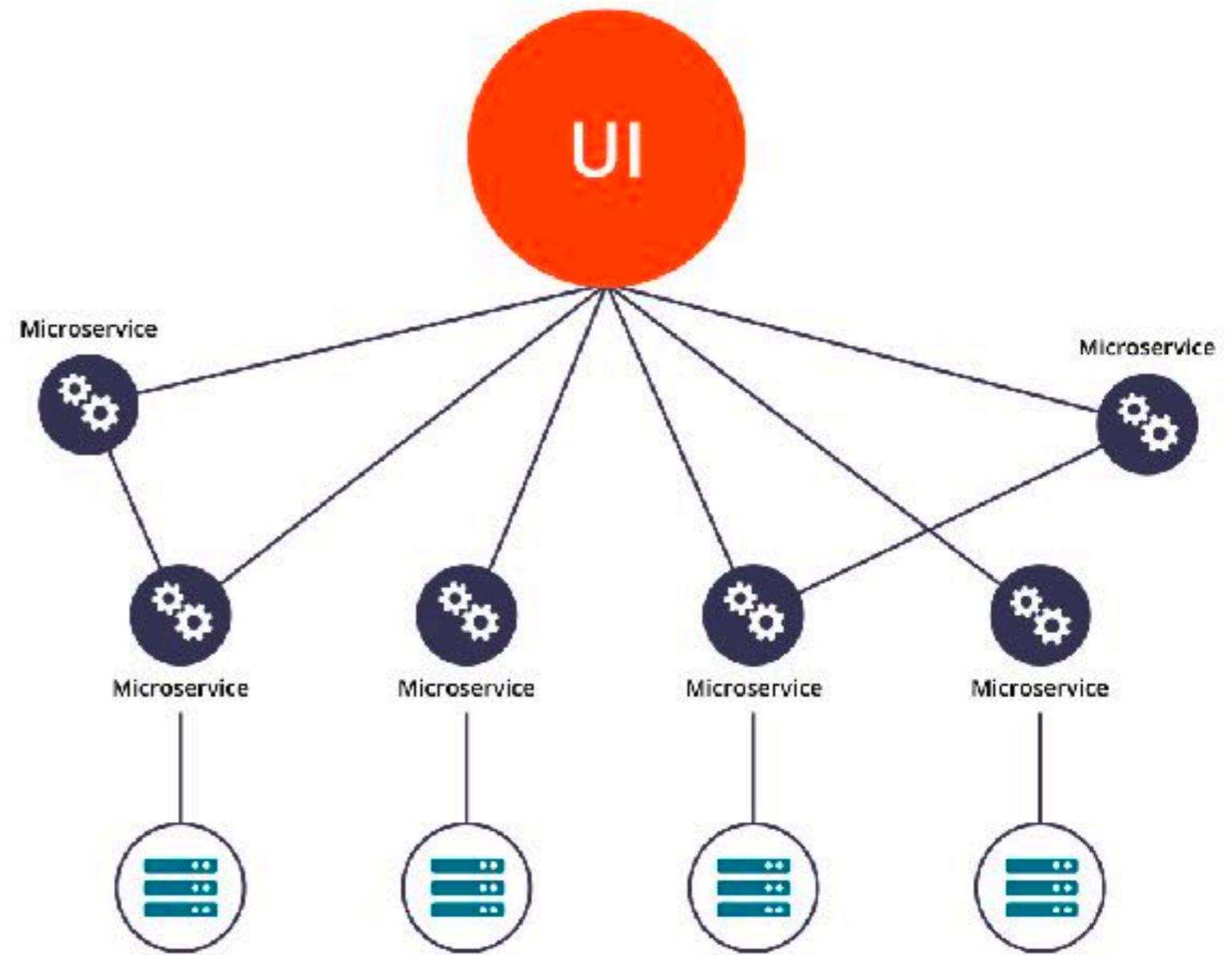
Better Architecture

Three principles for well-architected microservices. Source: Ma 2018.*





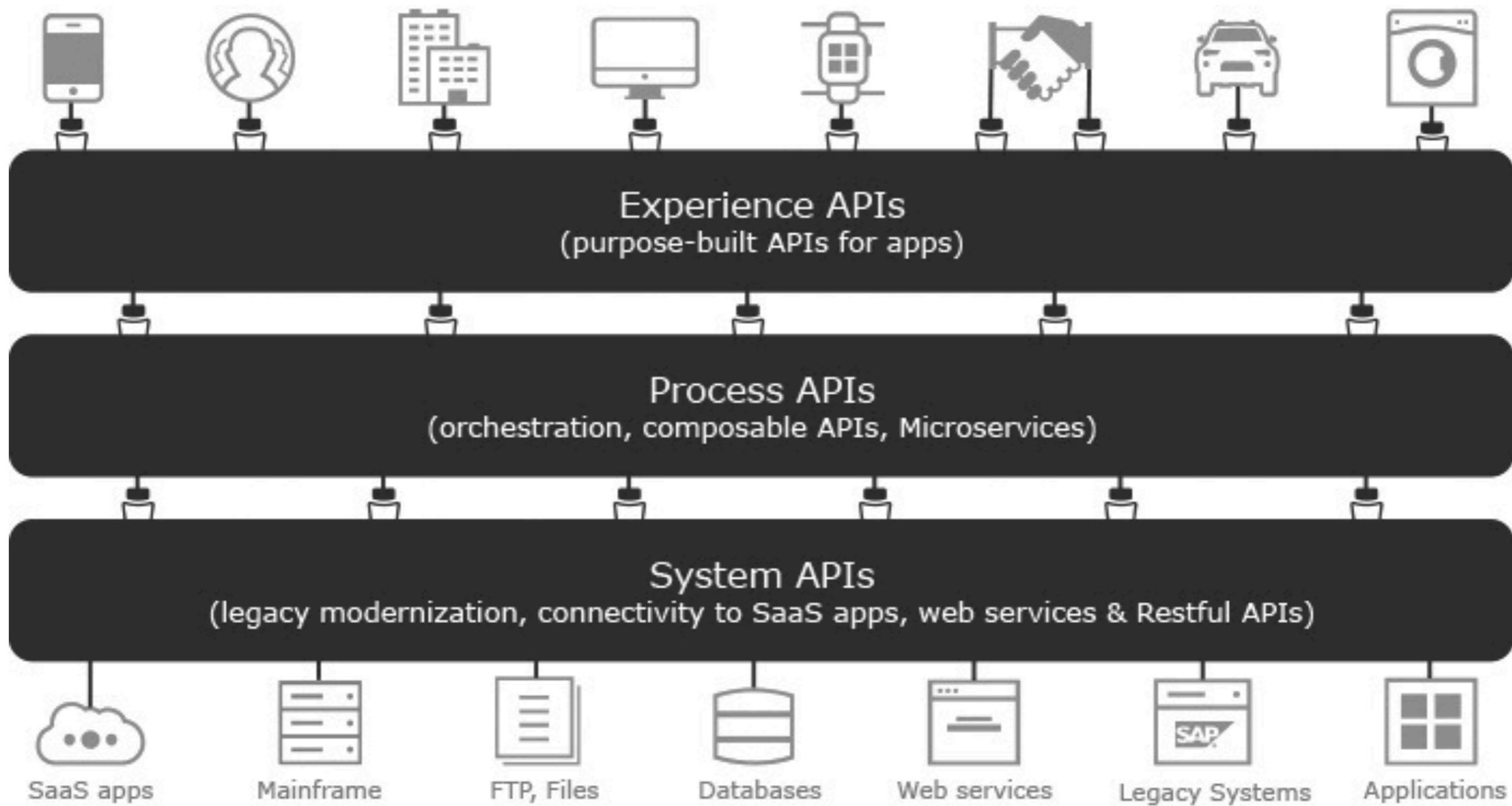
Monolithic Architecture



Microservice Architecture



Real World Architecture

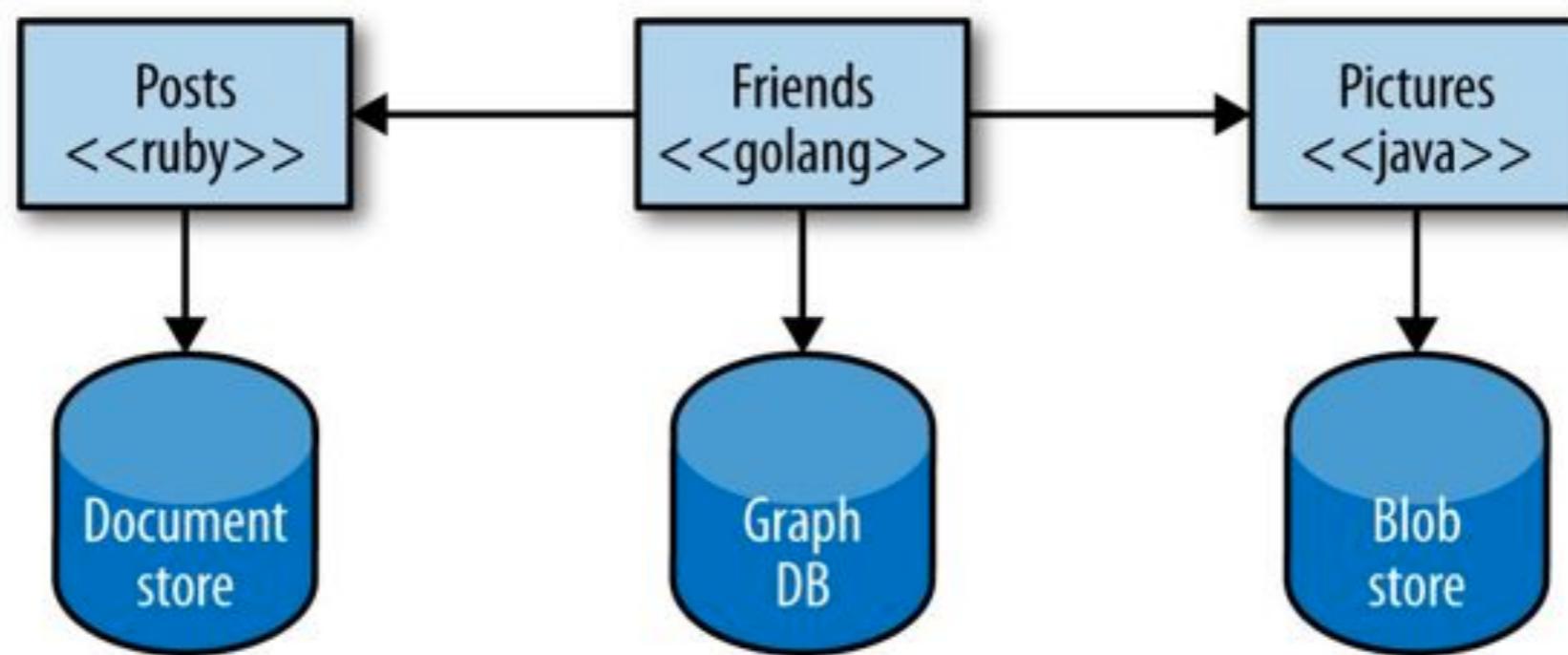


Key Benefits

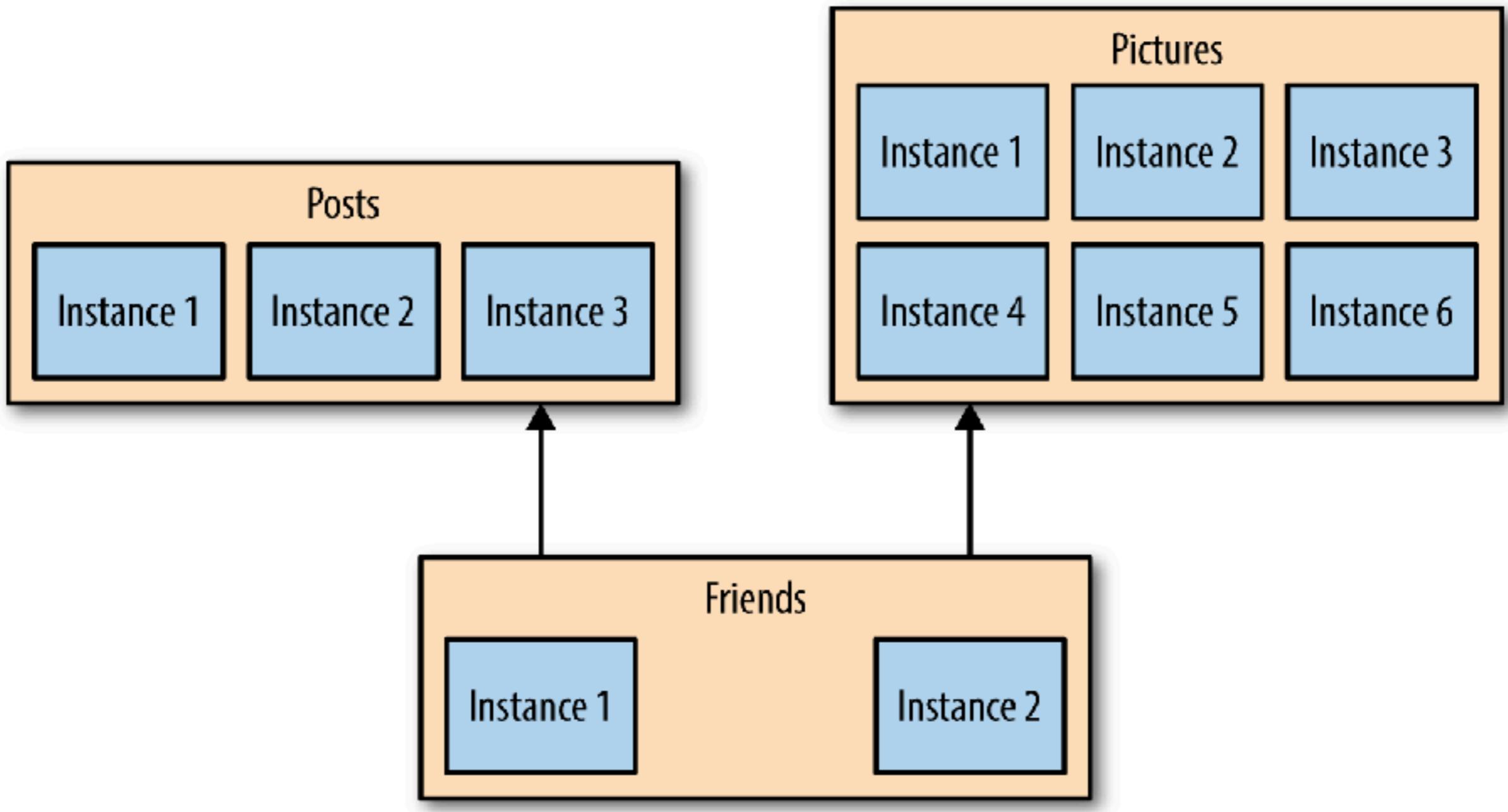


1. Technology heterogeneous

The right tool for each job
Allow easy experimenting and adoption of new technology

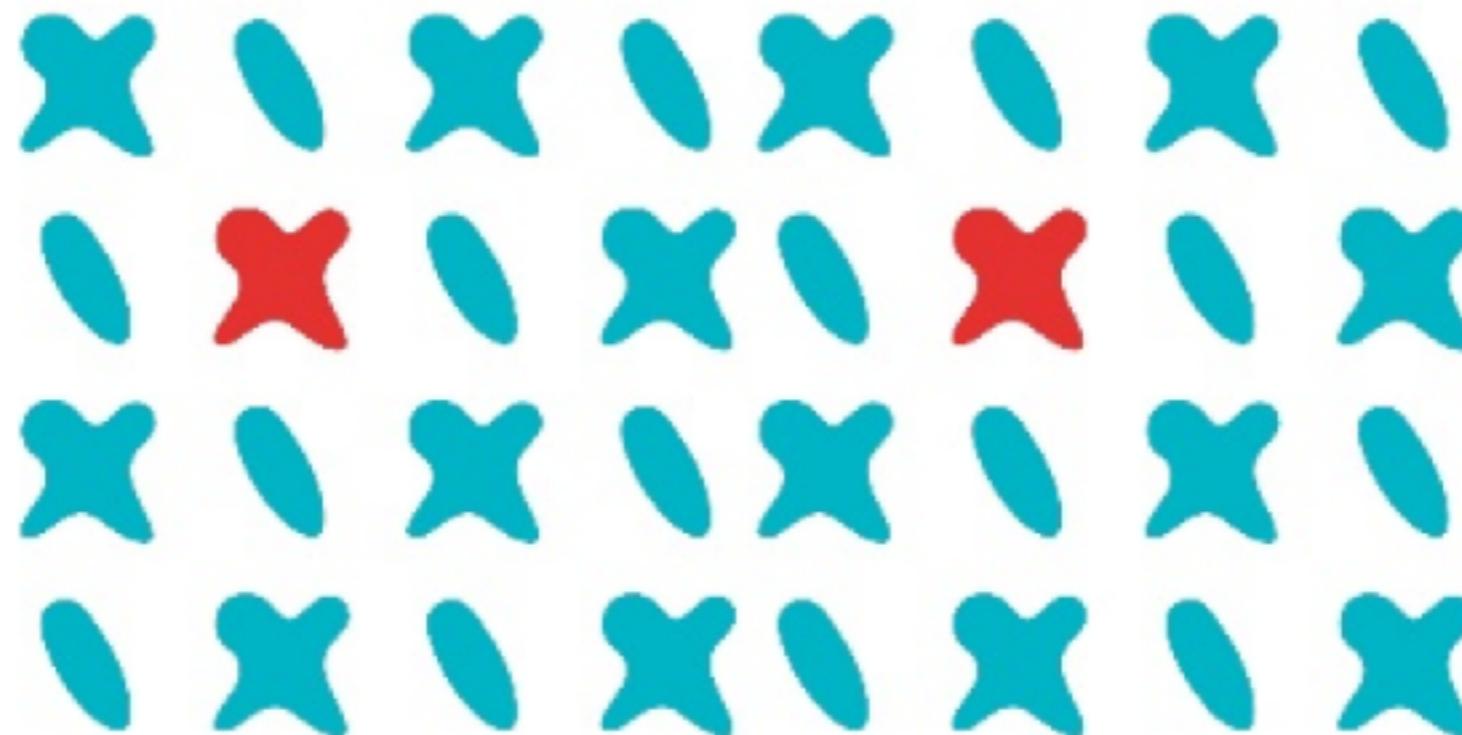


2. Scaling

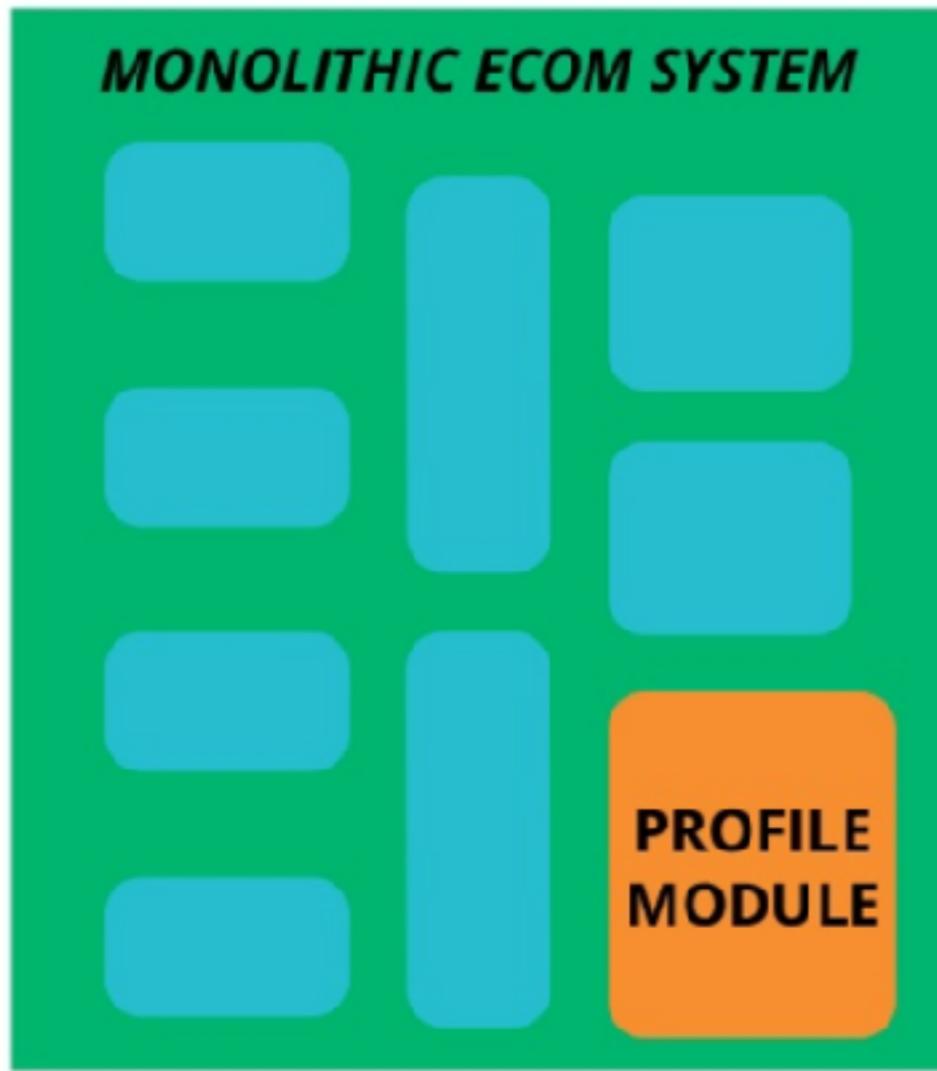


3. Ease of deployment

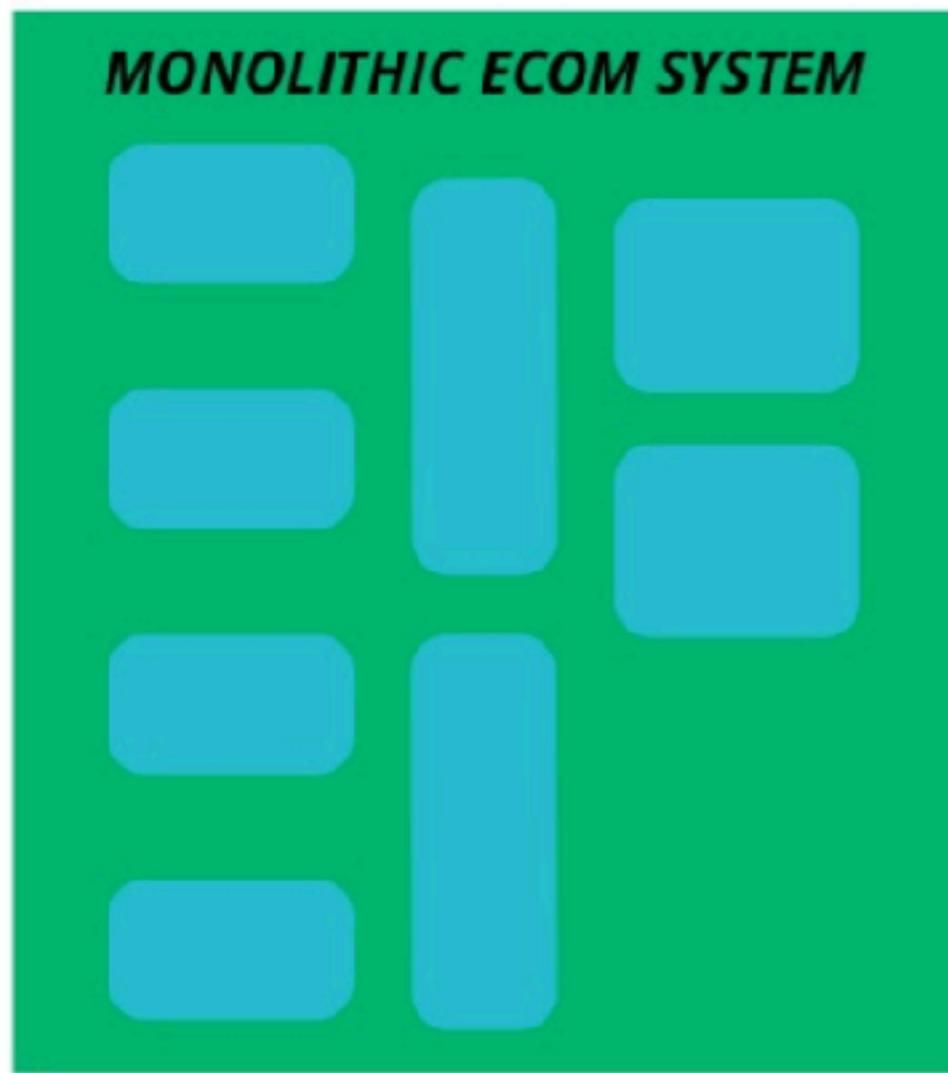
Deploys are faster, independent and problems can be isolated more easily



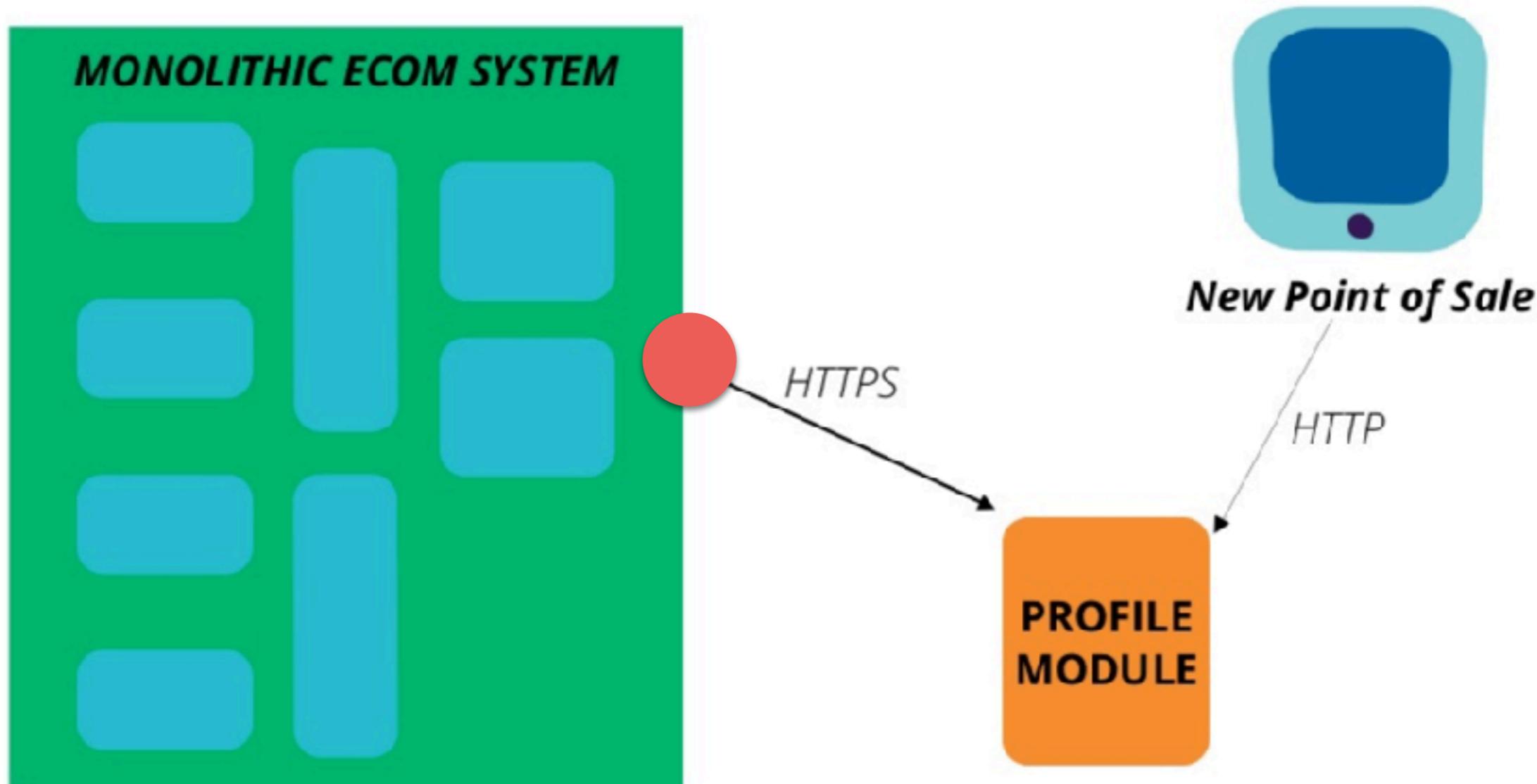
4. Composability and replaceability



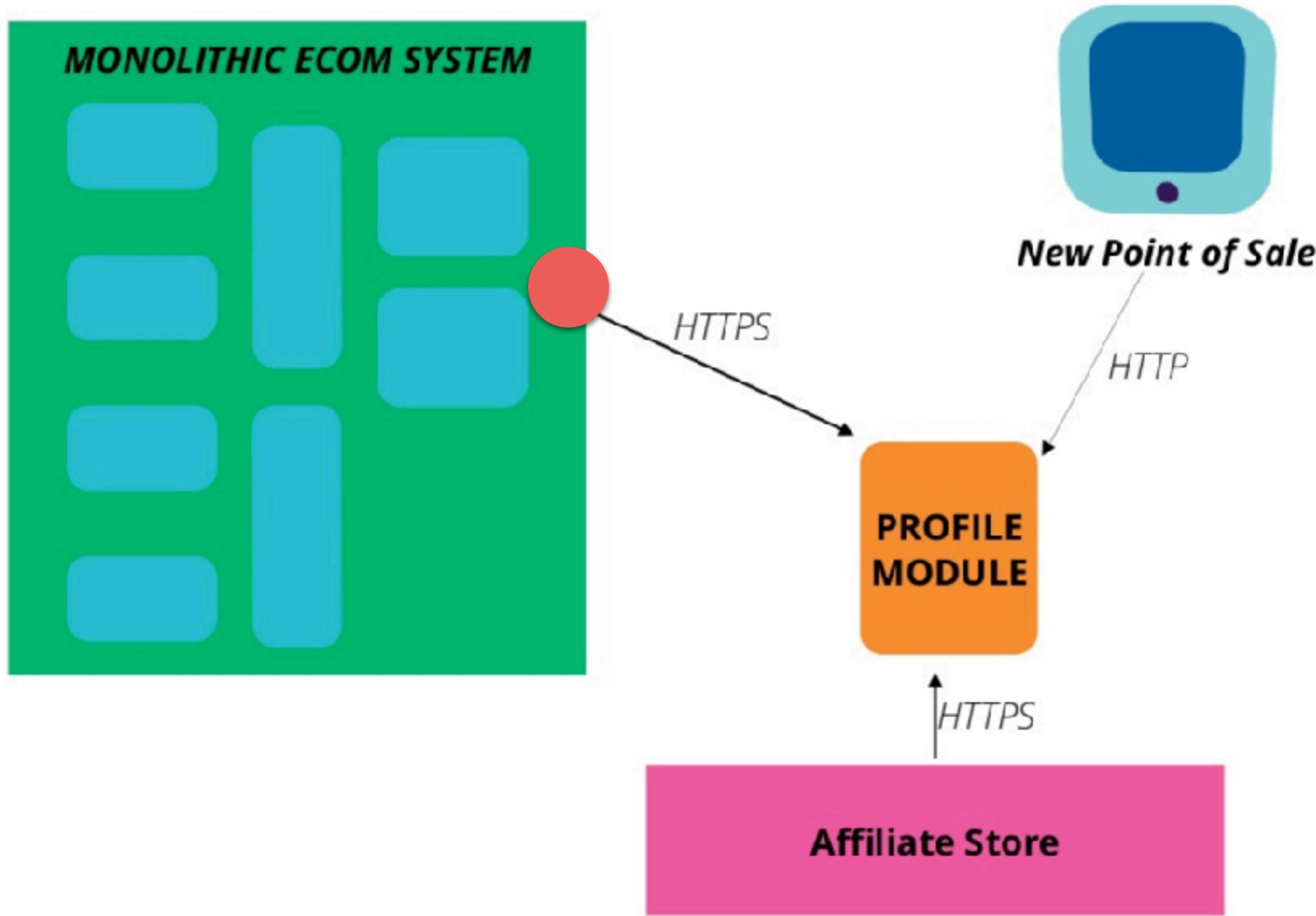
4. Composability and replaceability



4. Composability and replaceability



4. Composability and replaceability



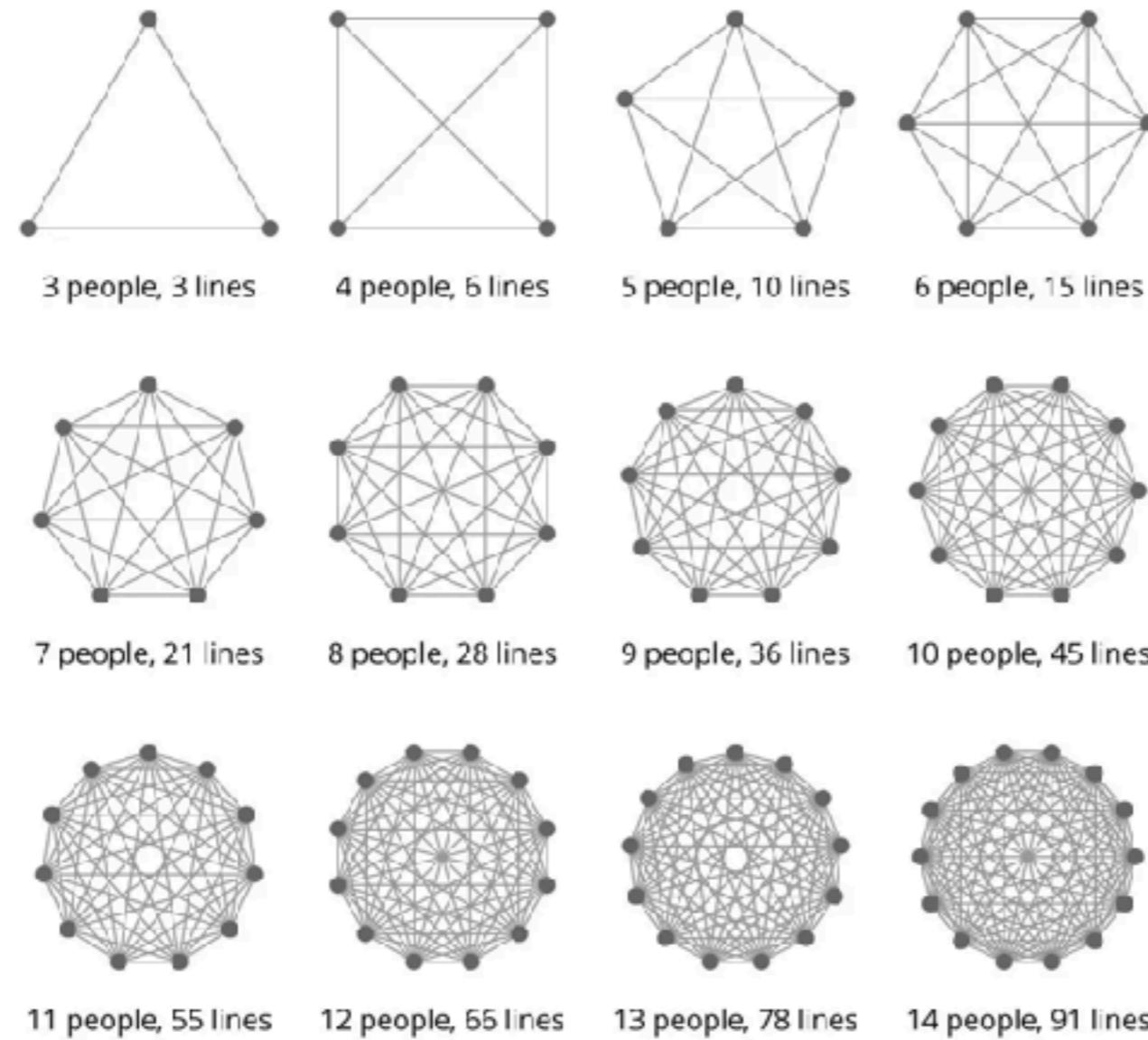
5. Organization alignment

Small teams and smaller codebases



Small team !!

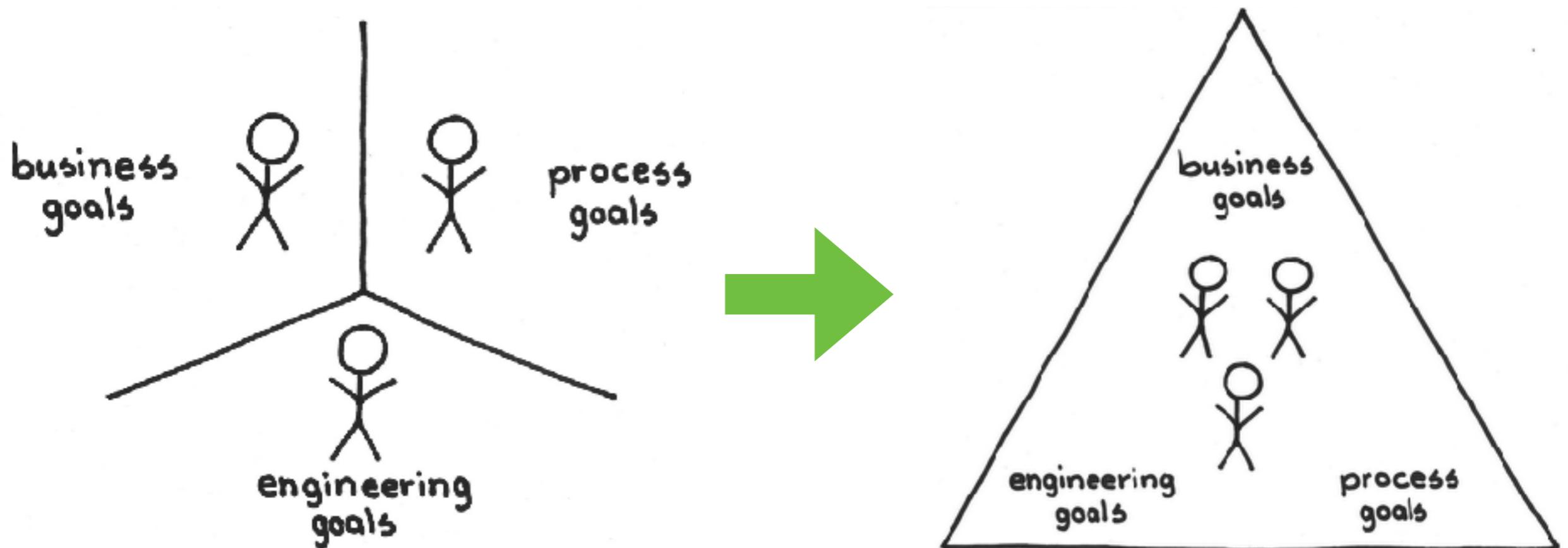
Line of communication and team size



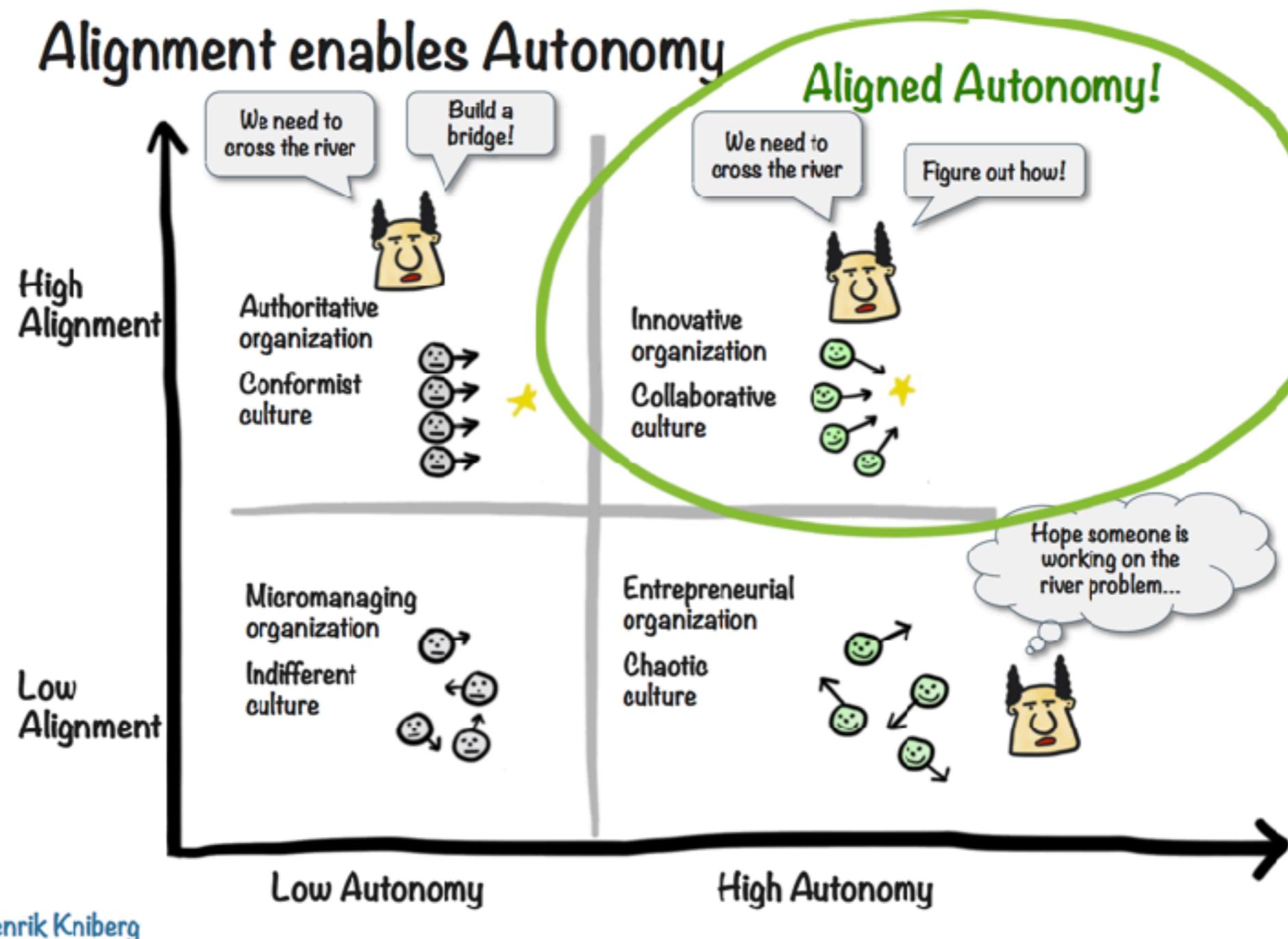
<https://www.leadingagile.com/2018/02/lines-of-communication-team-size-applying-brooks-law>



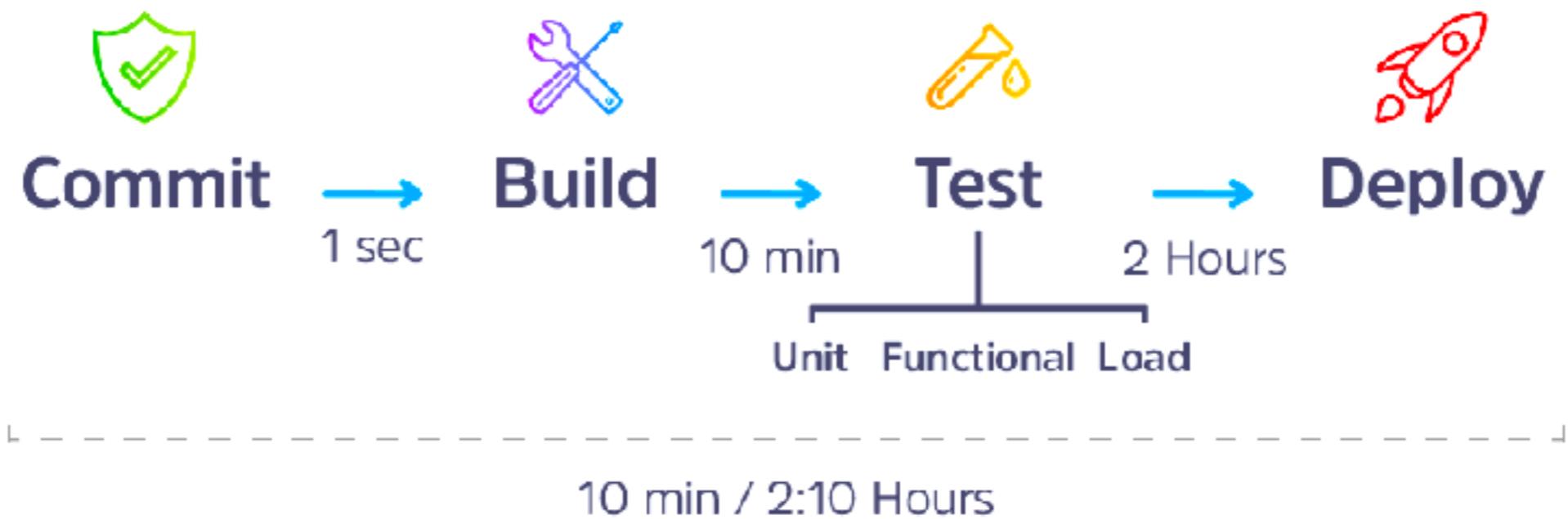
Autonomous team



Autonomous team



Autonomous team



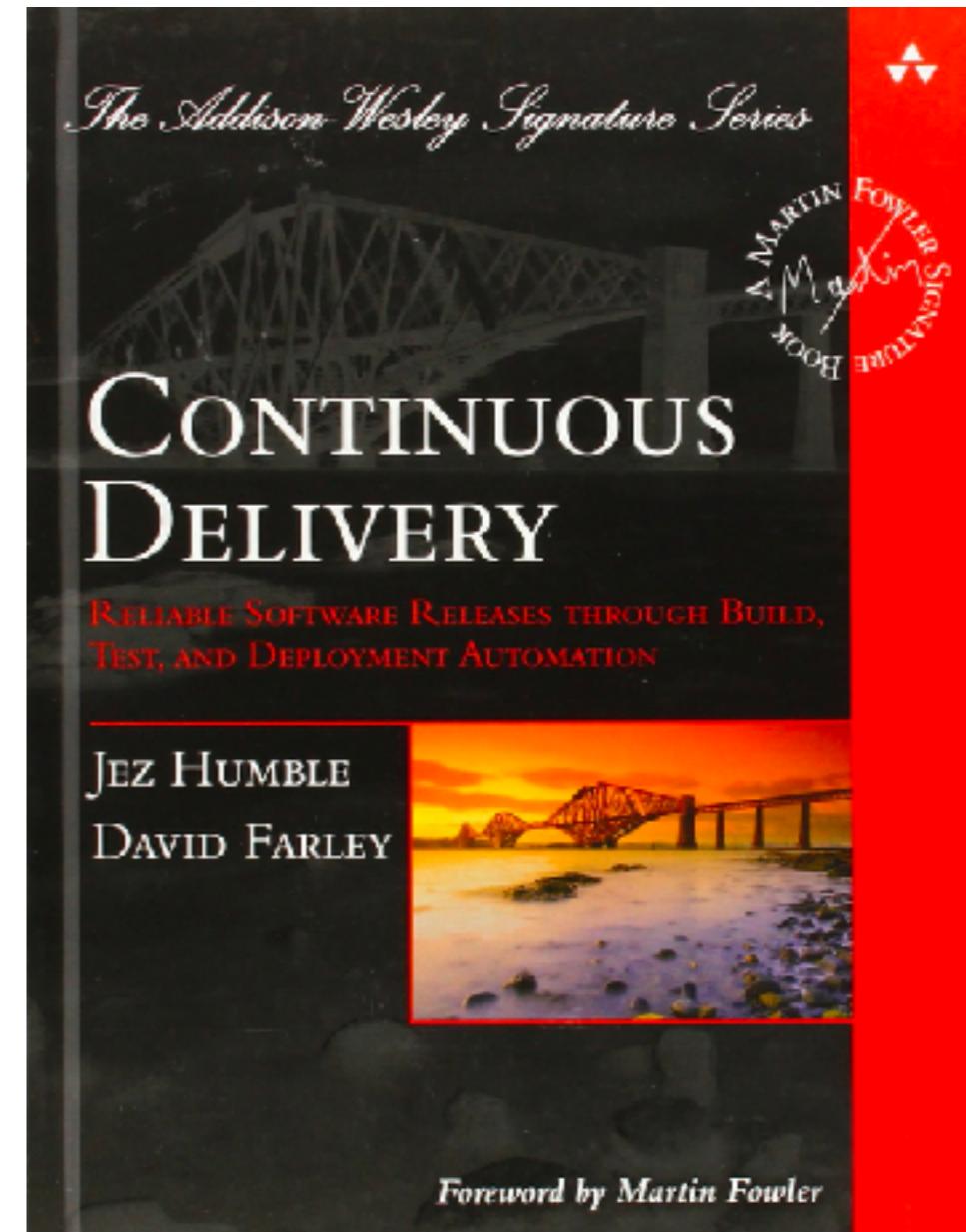
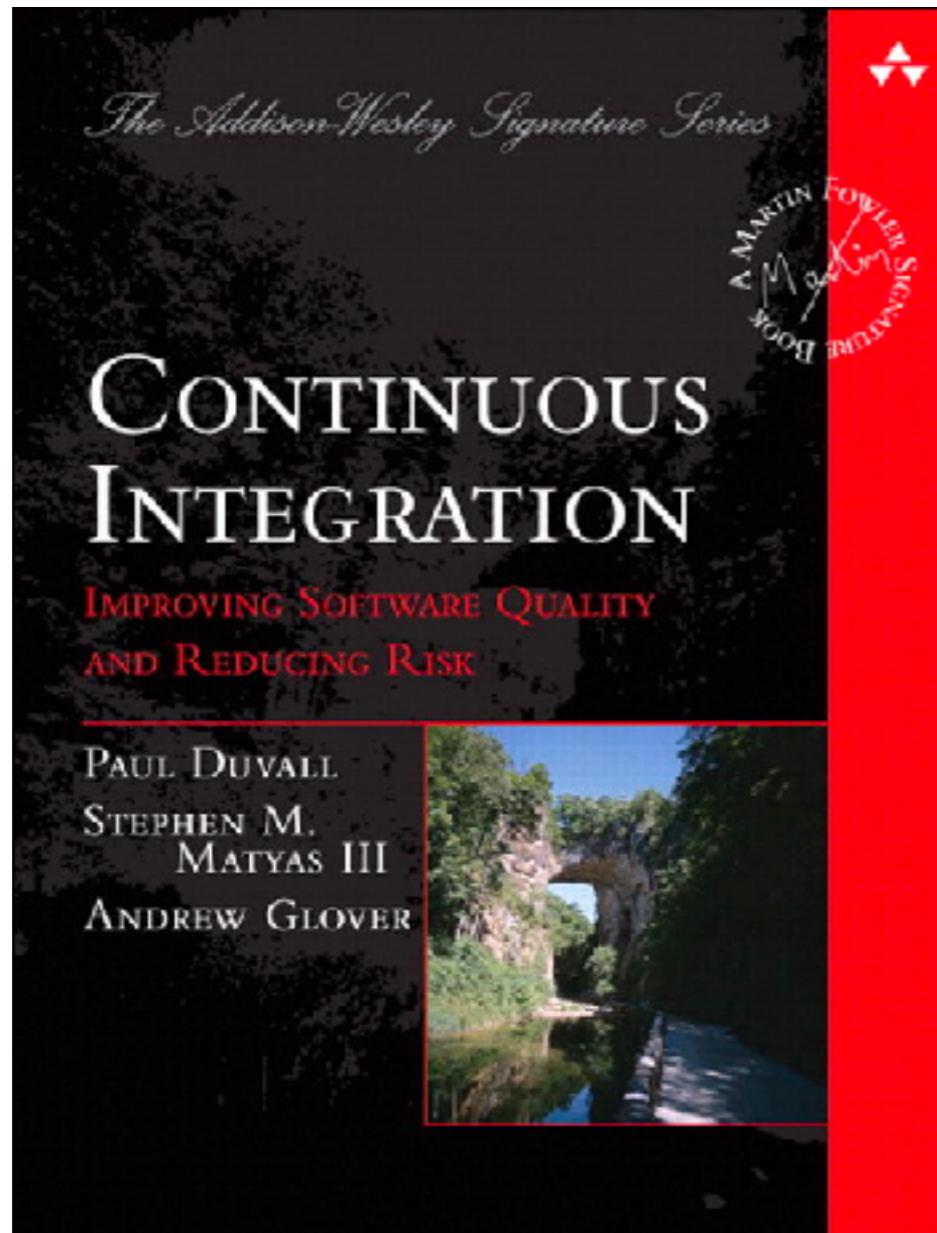
6. Enable Continuous Delivery

A part of DevOps

Set of practices for the **rapid, frequent and reliable delivery** software



Continuous Delivery/Deployment



Microservices

© 2020 - 2023 Siam Chamnankit Company Limited. All rights reserved.

Continuous Delivery/Deployment

Testability
Deployability
Autonomous team and loose coupling



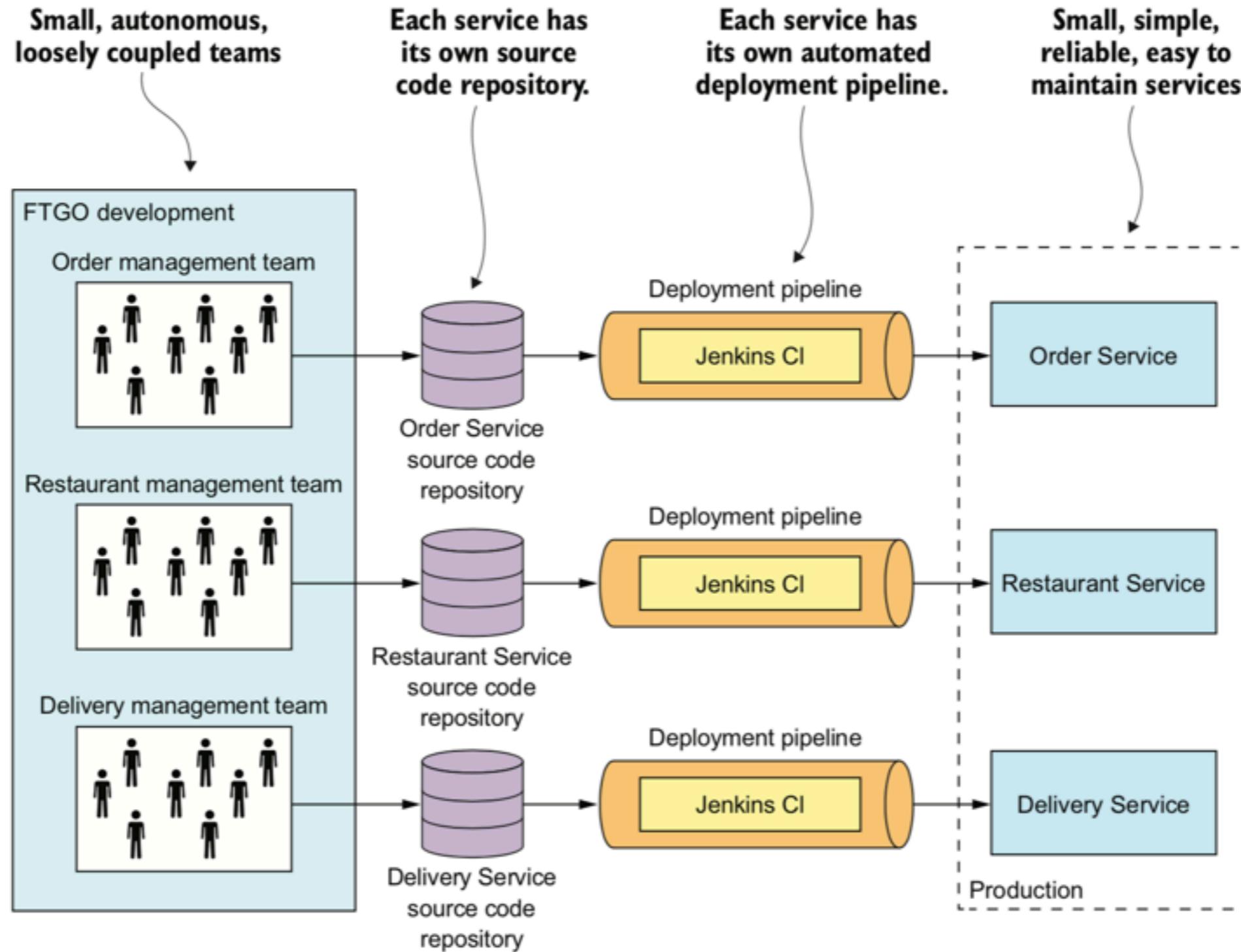
Benefits of Continuous Delivery

Reduce time to market

Enable **business** to provide reliable service
Employee satisfaction is higher



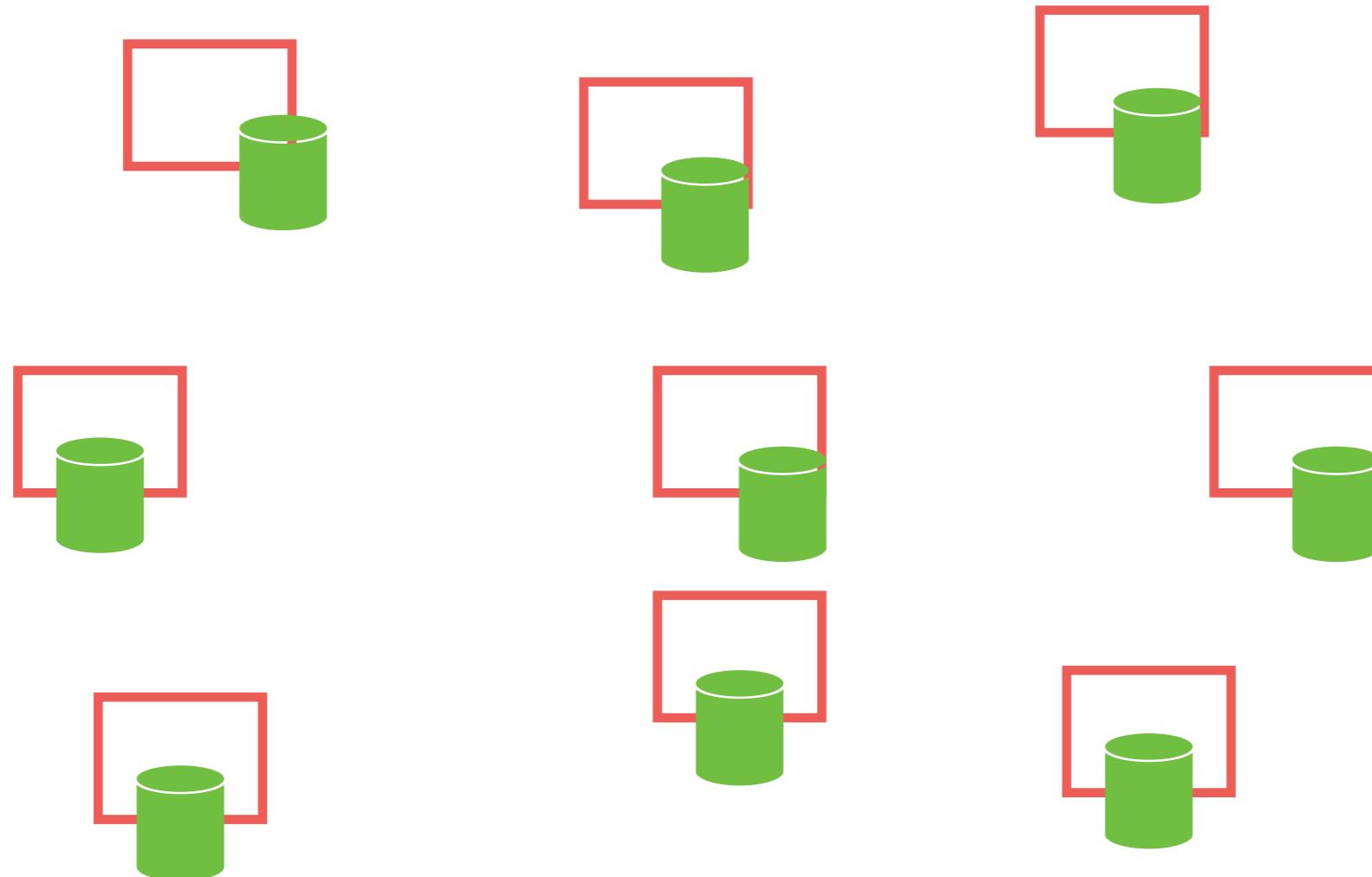
Continuous Delivery for service



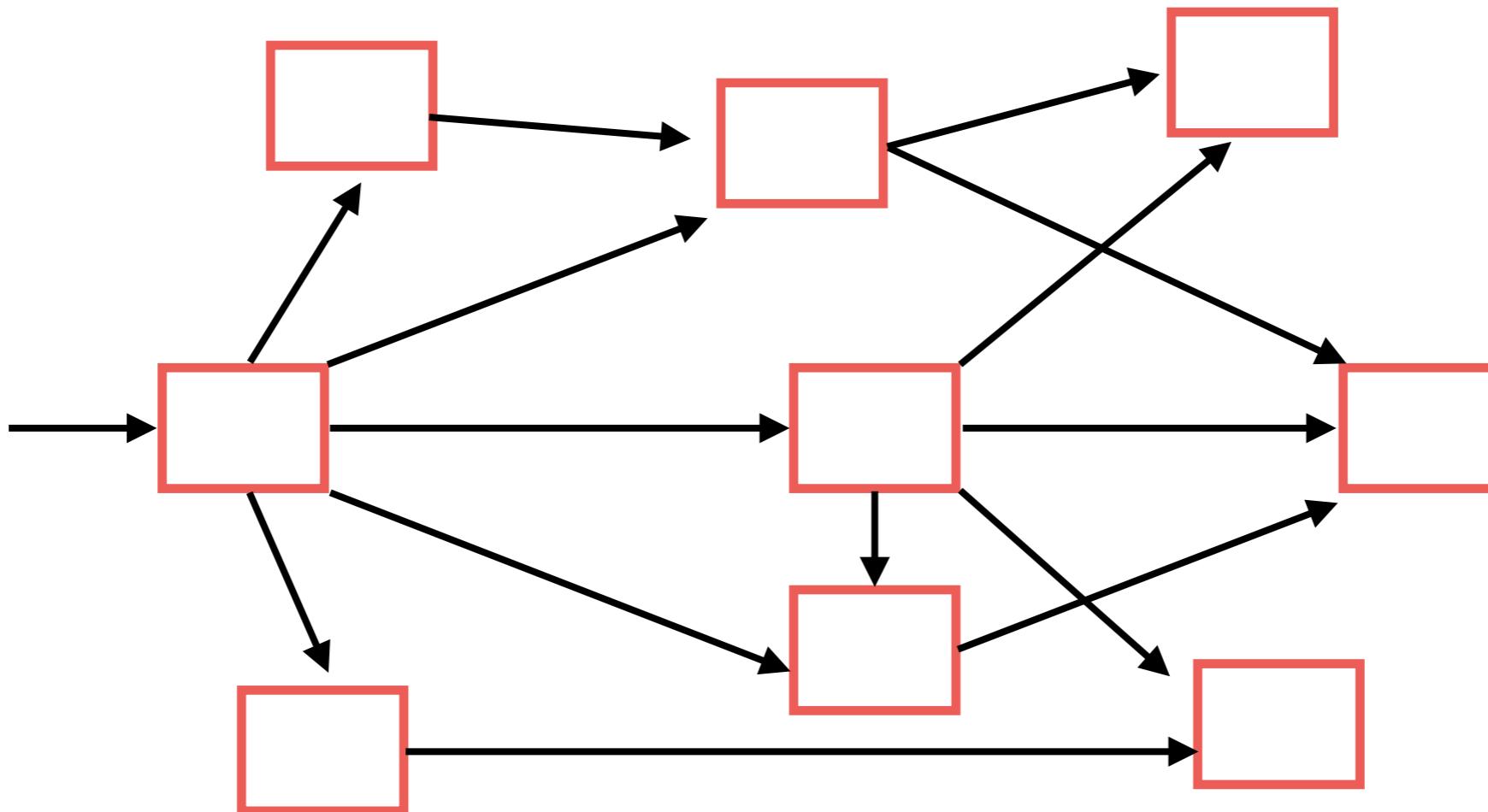
Drawbacks of Microservice



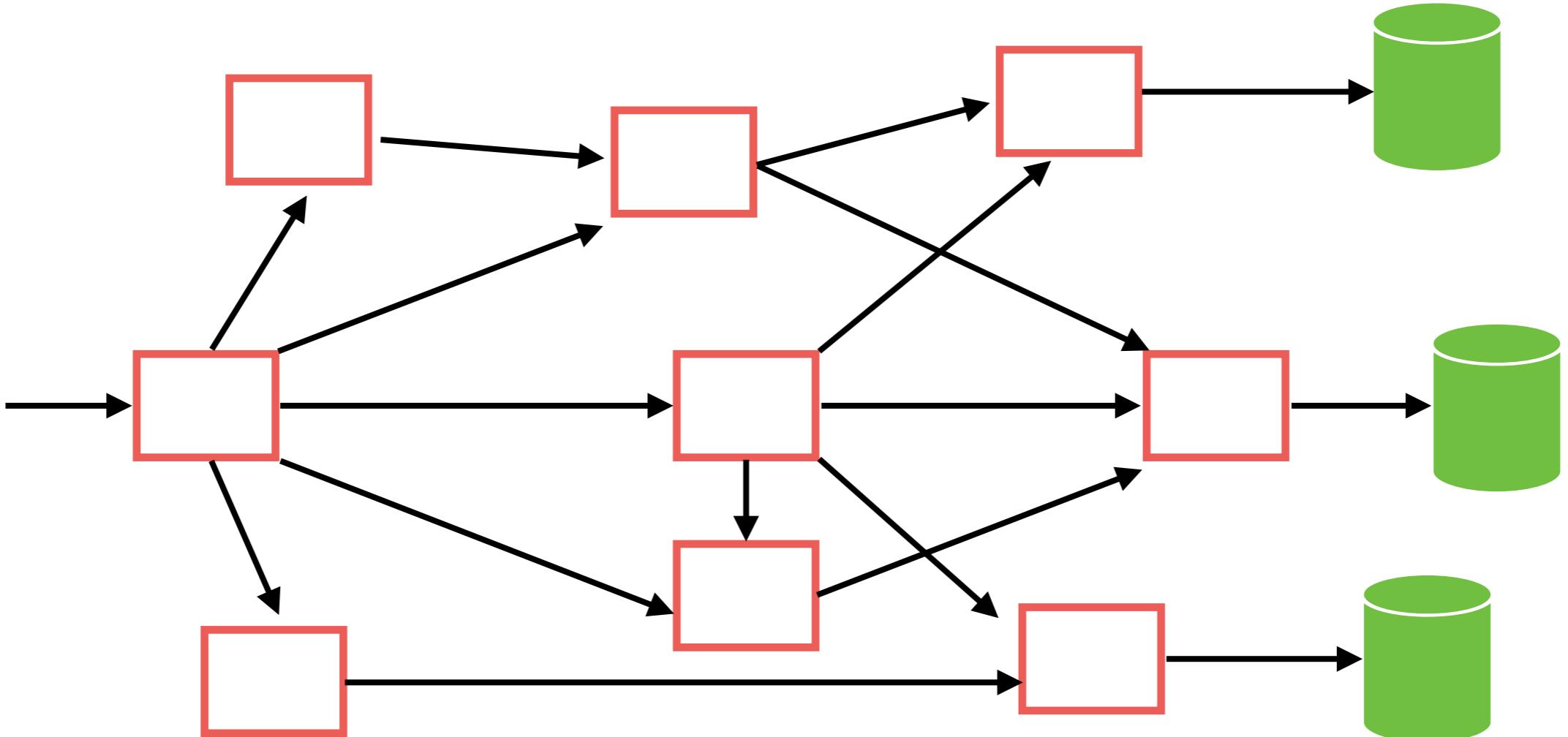
Microservice (Loosely coupled)



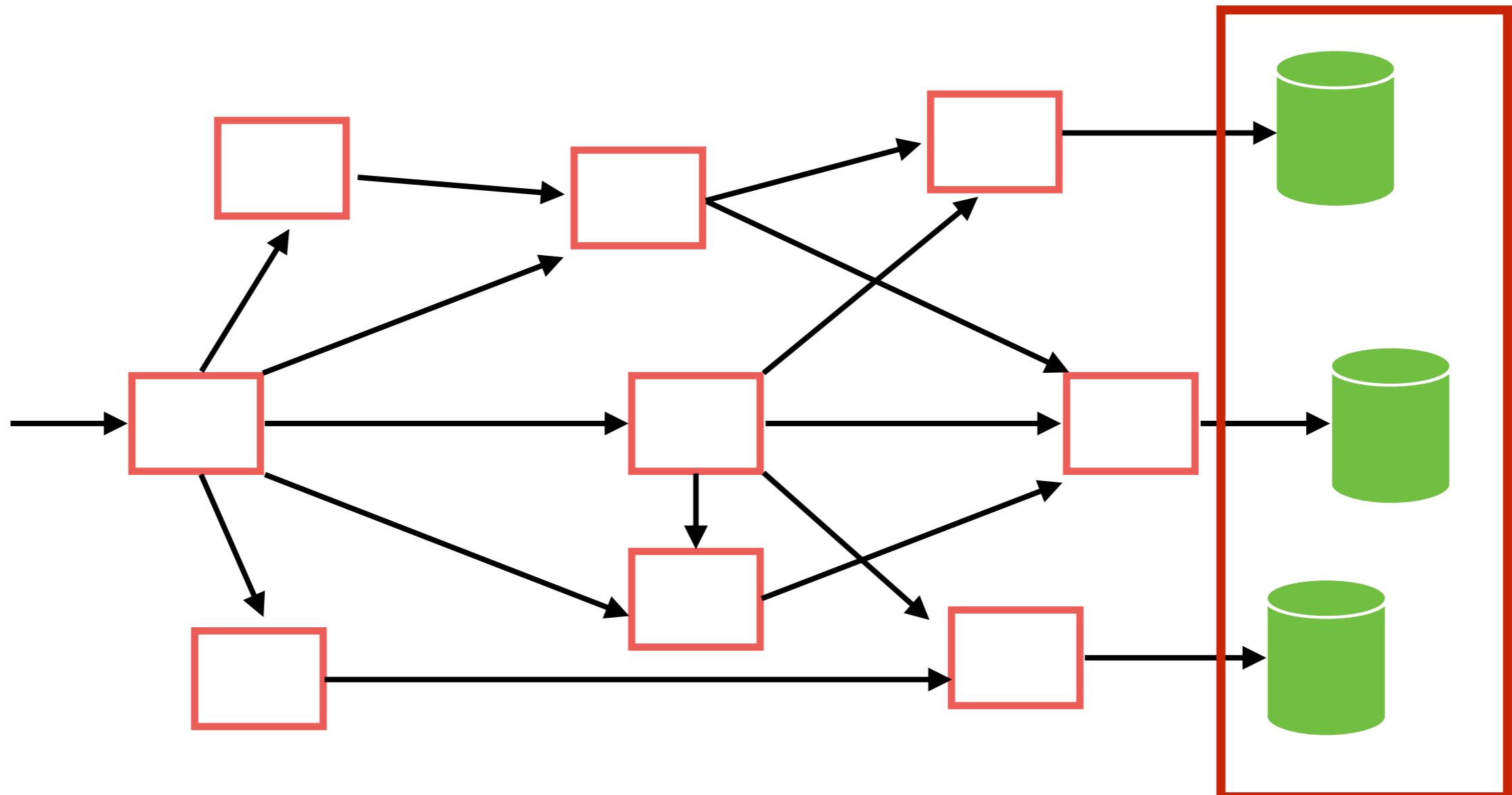
Microservice (Tightly coupled)



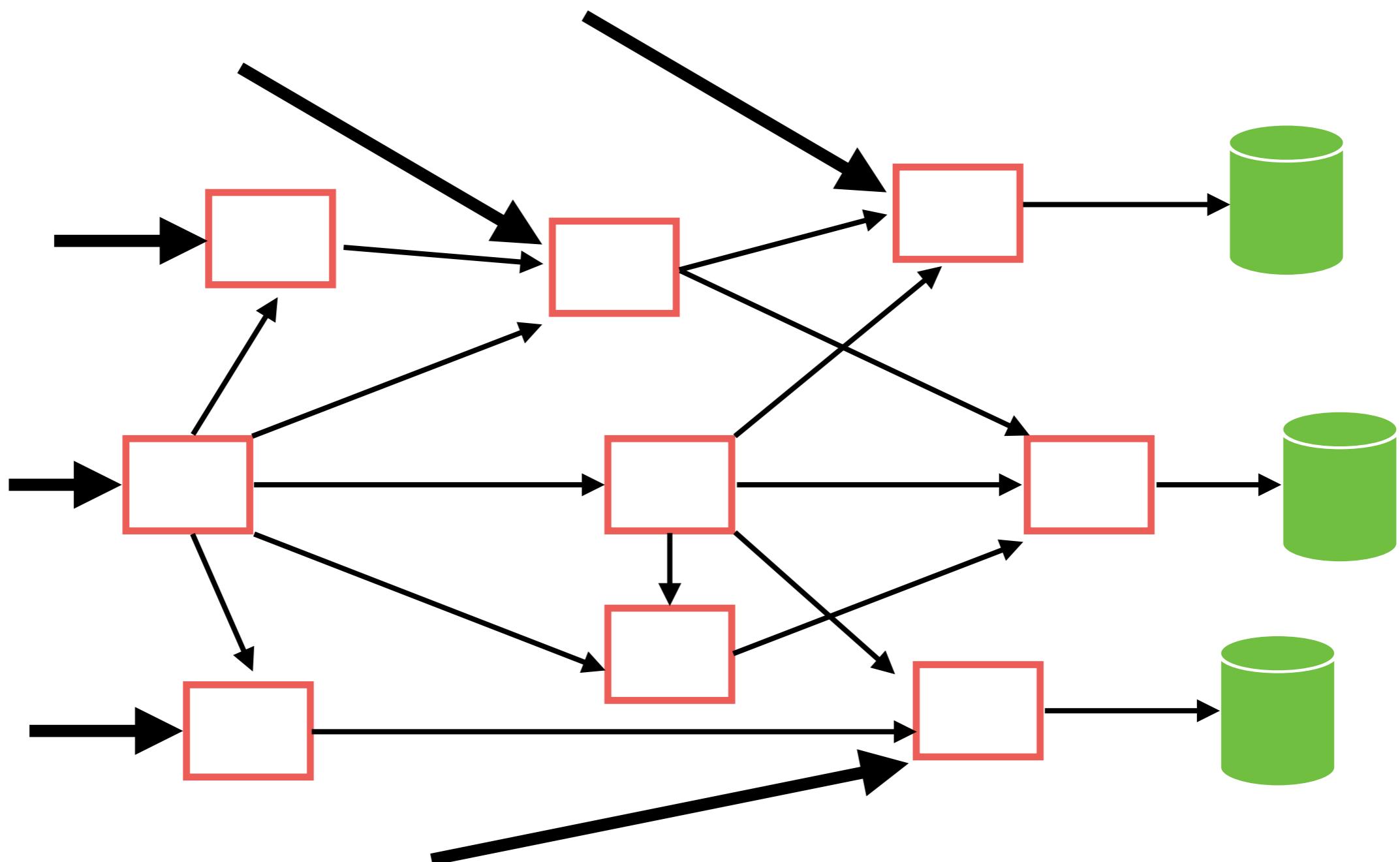
Own data



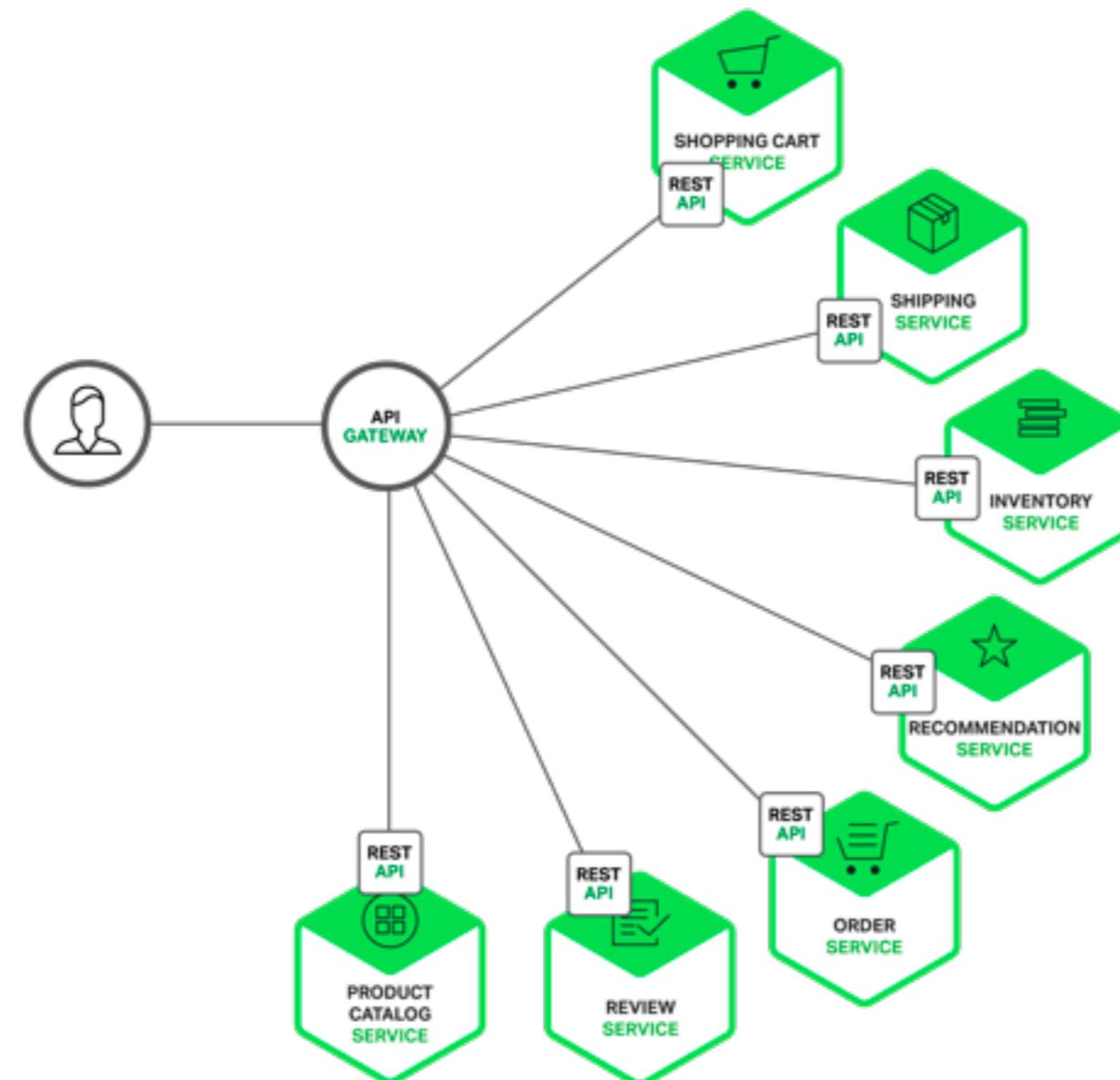
Data consistency !!



Multiple entry points



Working with API gateway



<https://www.nginx.com>



Functions of API gateway

Authentication

Authorization

Rate limiting

Caching

Metrics collection

Request logging

Load balancing

Circuit breaking



Benefits

Encapsulation interface structure
Client talk to service via gateway
Reduce round trip between service

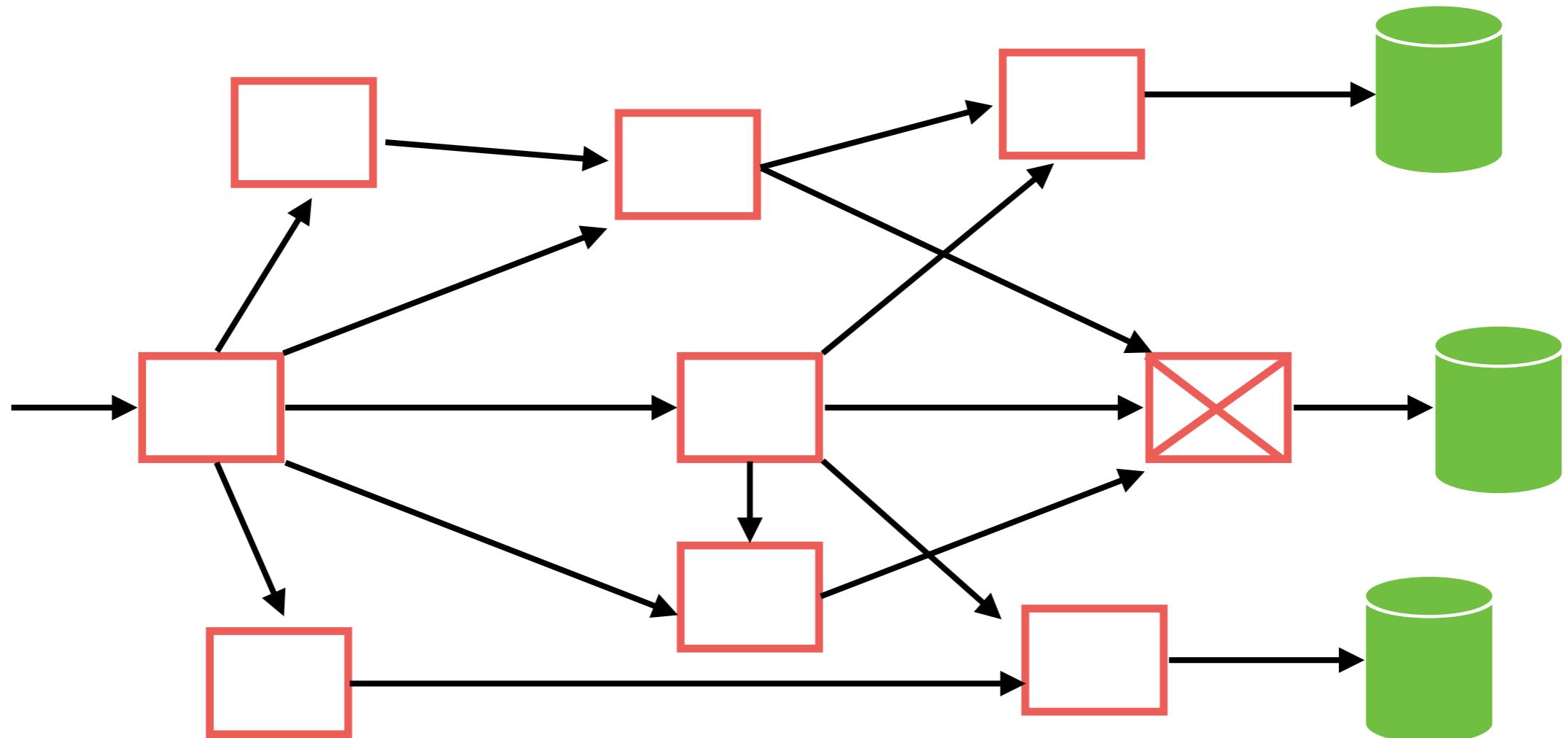


Drawbacks

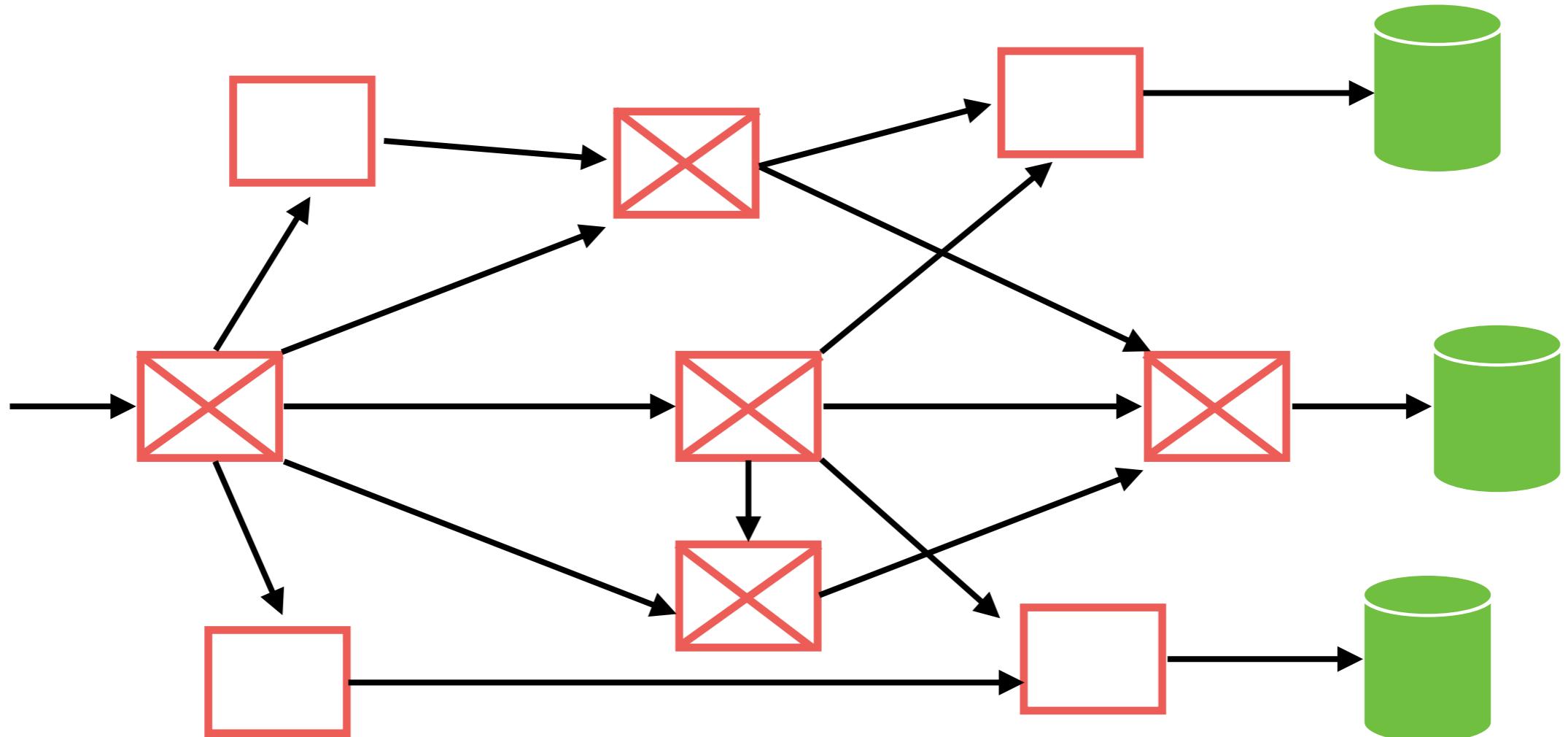
Development/Performance bottleneck
Single point of failure
Waiting for update data in gateway



Failure !!



Cascading Failure !!



Solutions for Reliability ?

Timeout

Retry

Rate Limit

Queue

Circuit breaker

Bulkhead

Auto-scale

<https://learn.microsoft.com/en-us/azure/well-architected/resiliency/reliability-patterns>



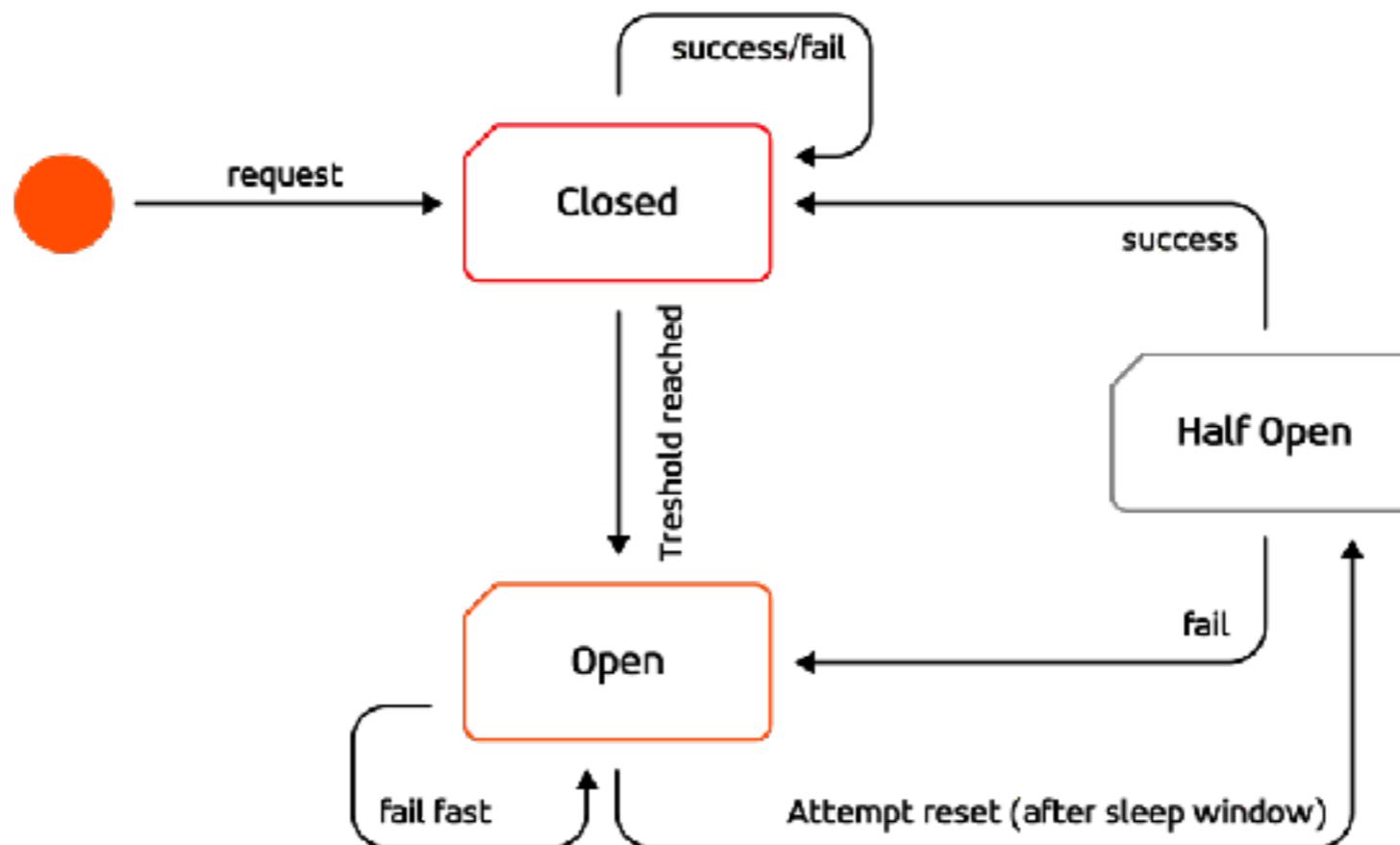
Microservices

© 2020 - 2023 Siam Chamnankit Company Limited. All rights reserved.

Circuit Breaker pattern

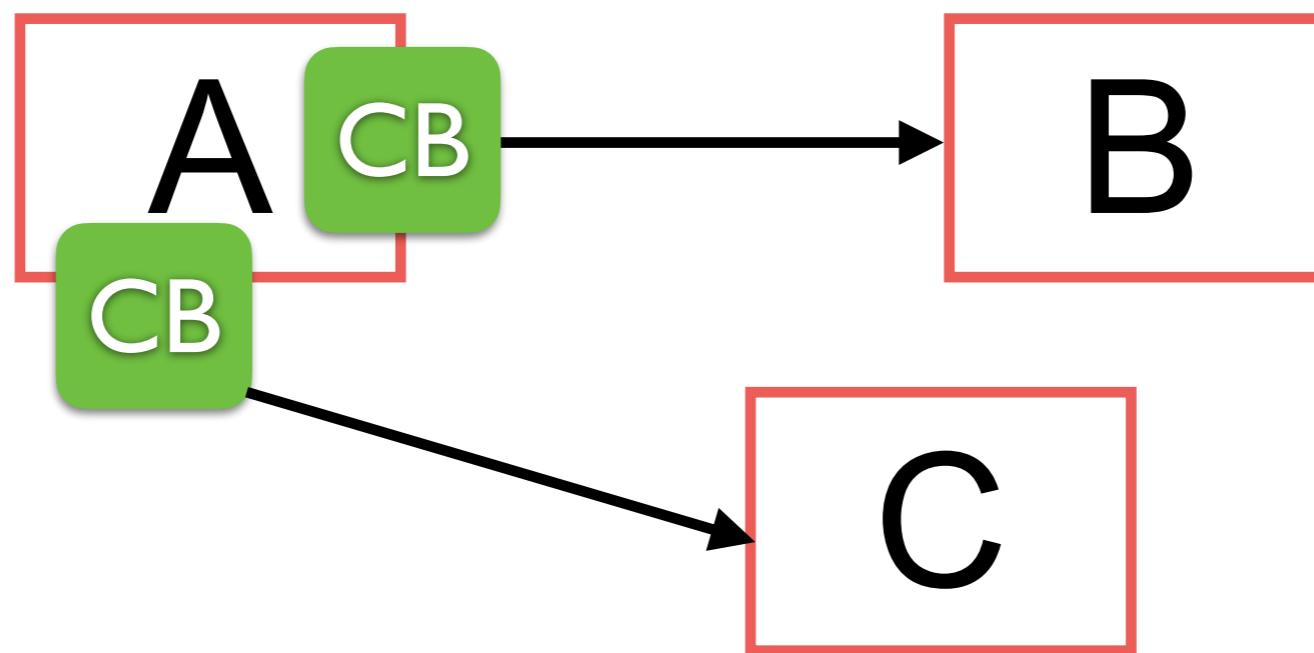
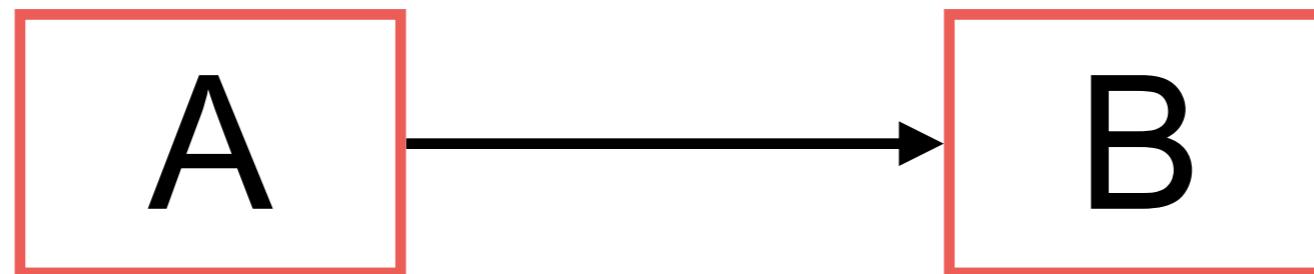
Track the number of success and failure

***If error rate exceed some threshold
then enable circuits breaker***



Circuit Breaker pattern

How to handling partial failure ?



Drawbacks of Microservice



Drawbacks of Microservice

Find the right set of services
Distributed systems are complex
How to develop, testing and deploy ?



Service Principles

Creation

Interface

Testing

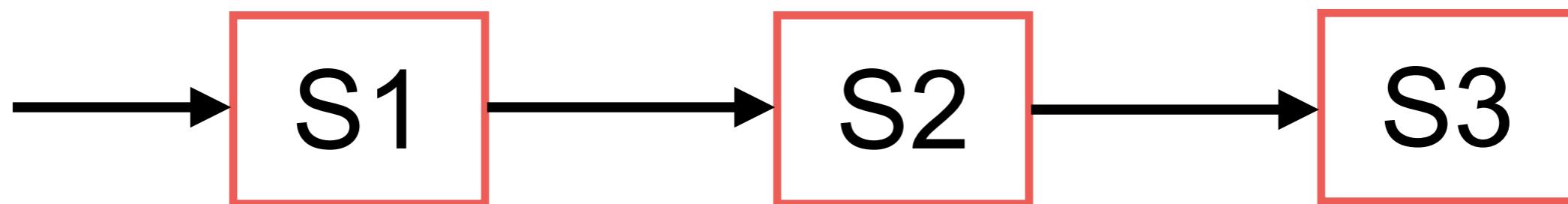
Operation

<https://github.com/Yelp/service-principles>



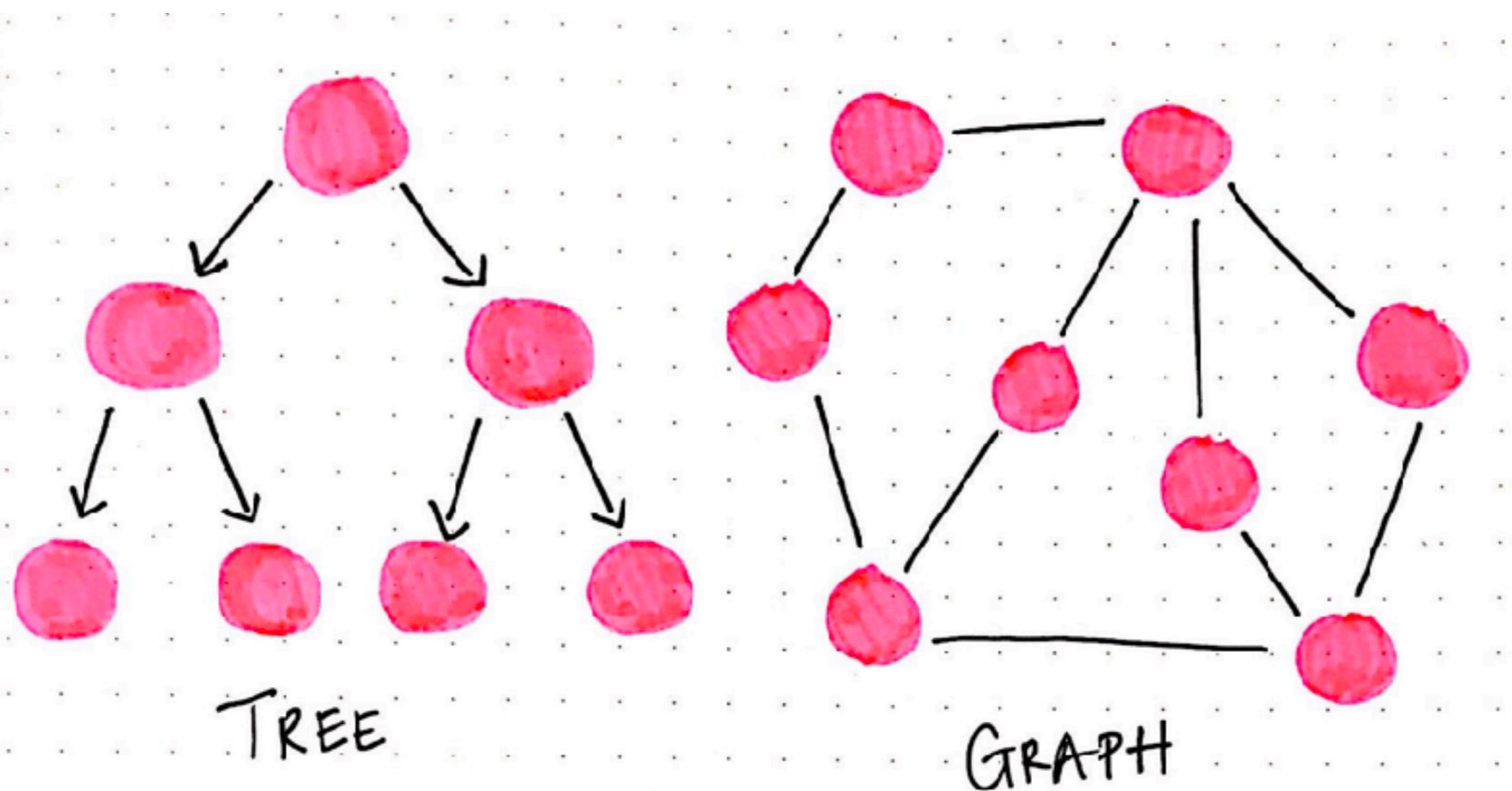
Drawbacks of Microservice

Deploy feature that required multiple service ?



Service Principles

Minimize the depth of the service call-graph



<https://github.com/Yelp/service-principles>

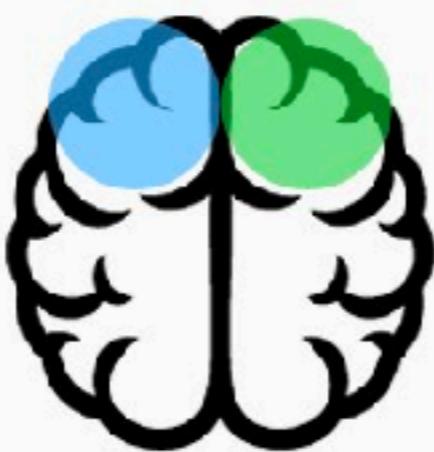


Service Principles

Minimize the number of services owned by your team

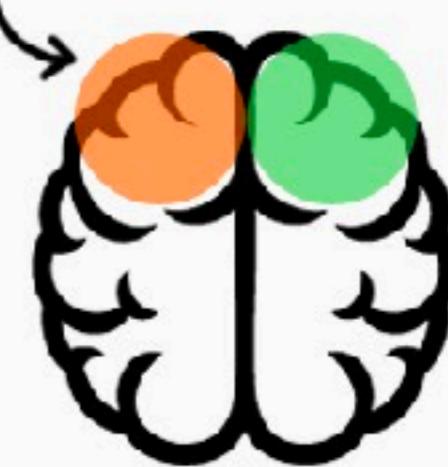


SINGLE TASK



TWO TASKS

THE 3RD TASK IS REPLACING
ONE OF THE 3 TASKS



THREE TASKS

<https://github.com/Yelp/service-principles>



Service Interfaces

APIs => HTTP, Messaging
Backward compatibility
Versioning



Testing strategies for Services

Unit testing

Integration testing

Component testing

Contract testing

End-to-End testing

Chaos testing

Performance
testing

Security
testing

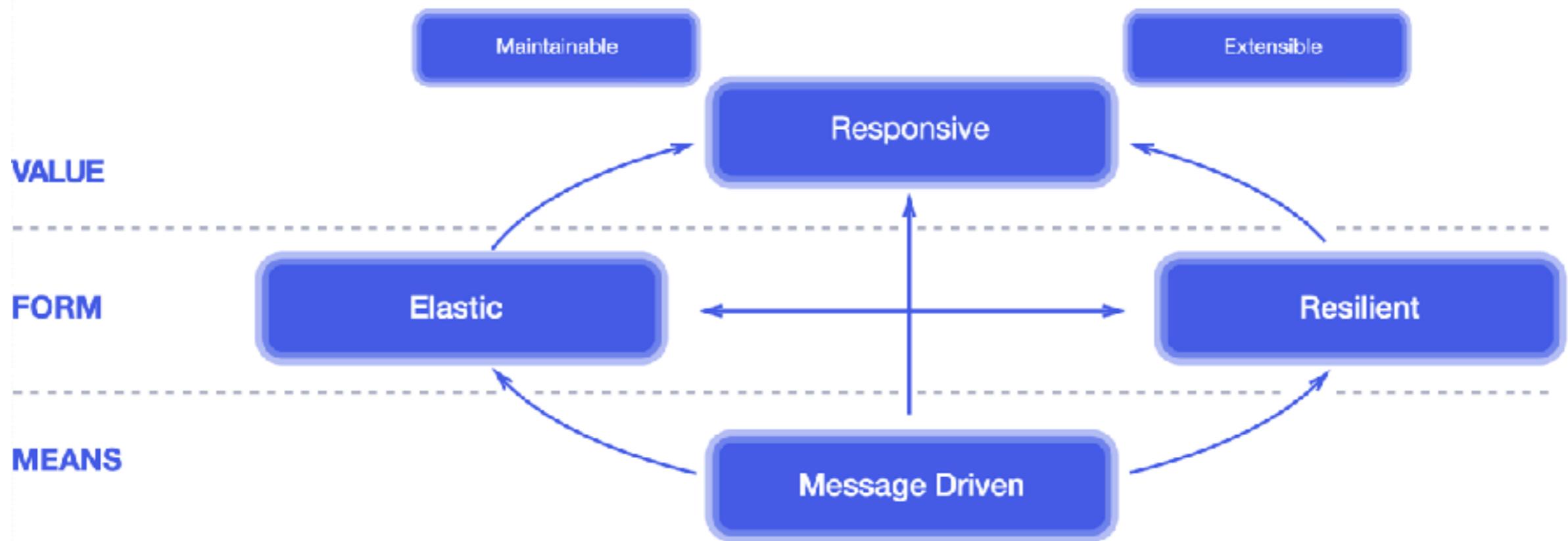


Operation for Services

How to deploy ?
Observable servies
Plan for failures !!



Reactive Architecture

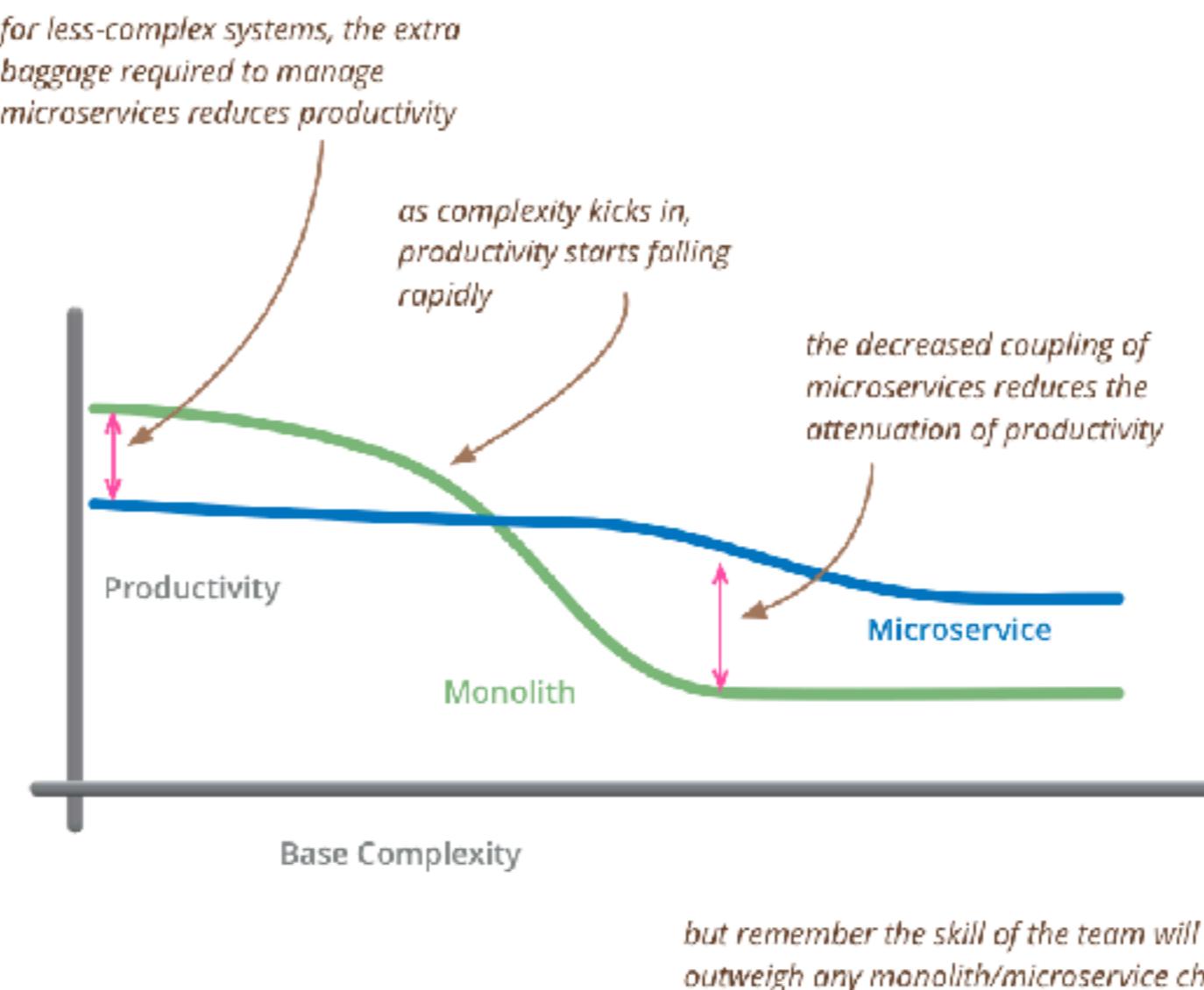


<https://www.reactivemanifesto.org/>



Drawbacks of Microservice

Decide to use when adopt is difficult !!



Microservice architecture patterns

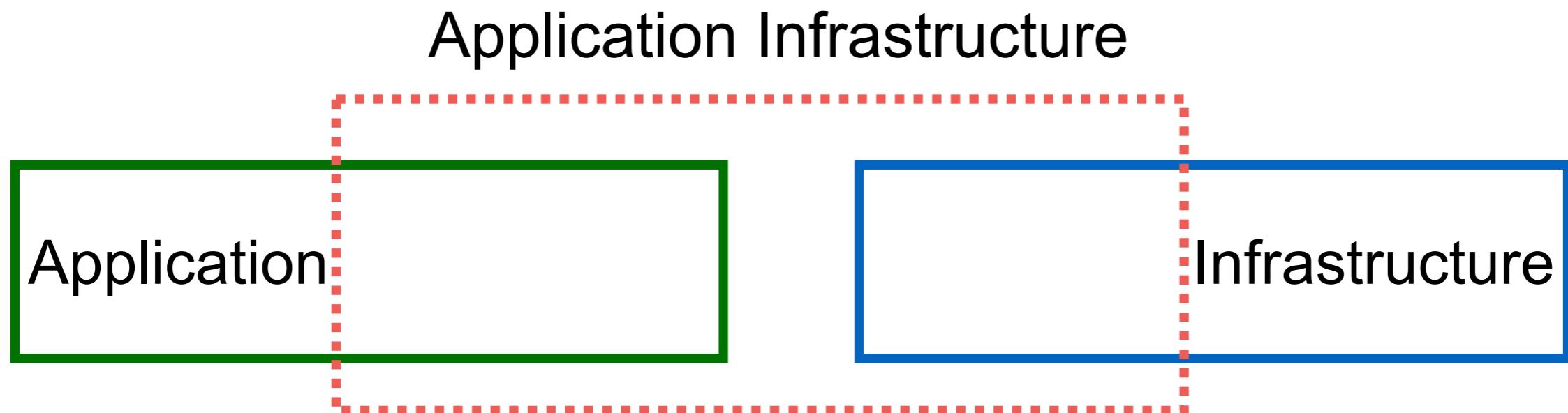


3 Layers of service patterns

Application patterns

Application infrastructure pattern

Infrastructure pattern



<https://microservices.io/>



Decompose application into small services !!



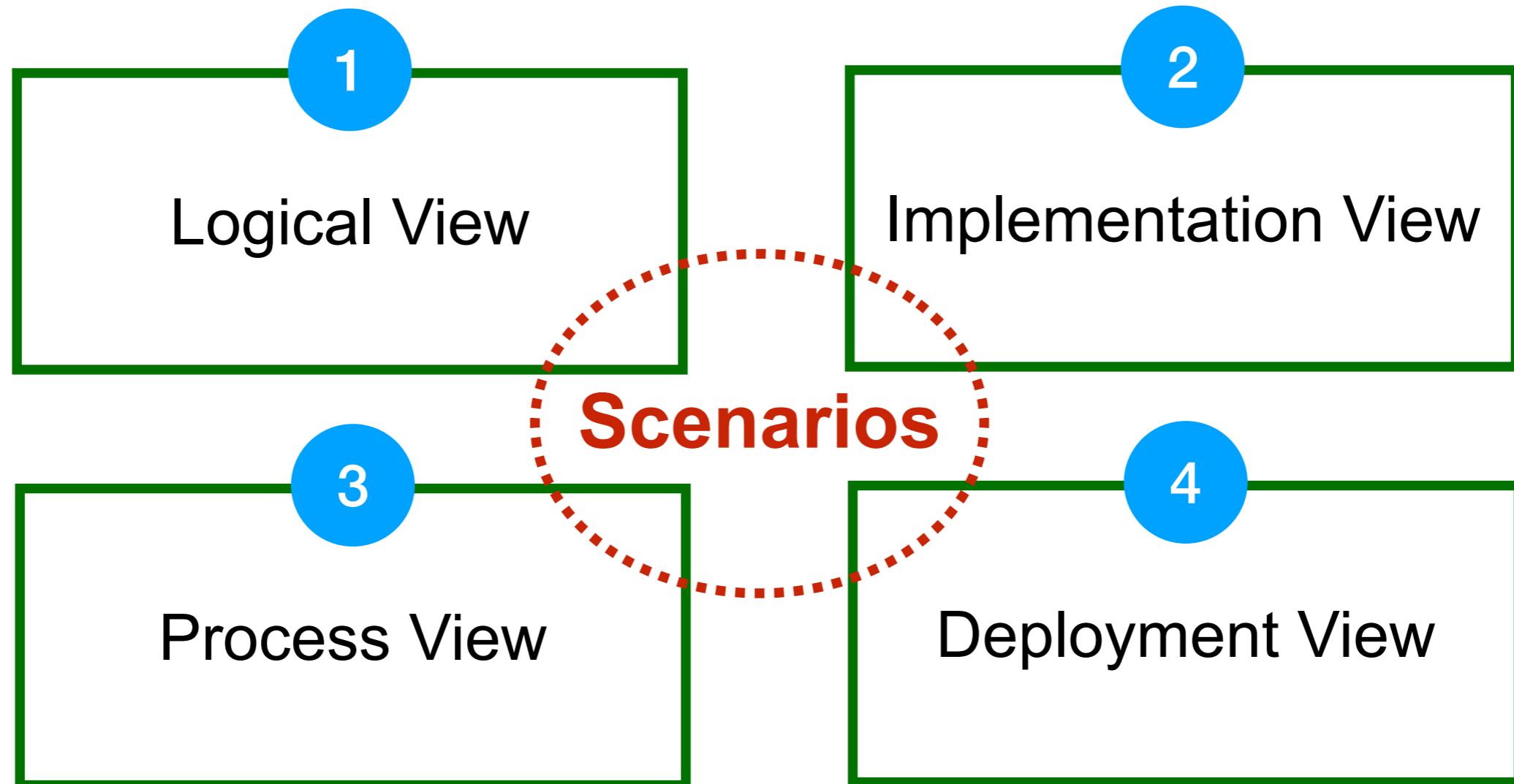
Premature splitting is the root of
all evil.



Every time you make the decision
to split out a new microservice,
there's a **risk** of ending up with a
bloated app.



4 View model of Software Architecture



Decompose application into services

By business capability ?

By subdomain/technical ?

By team ?



Step to define services

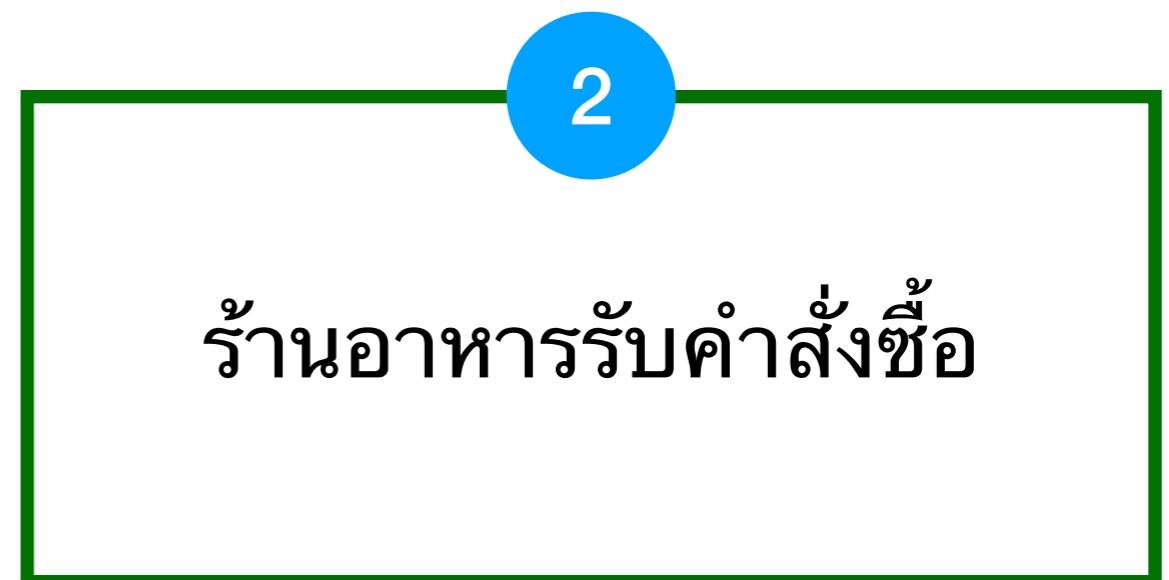
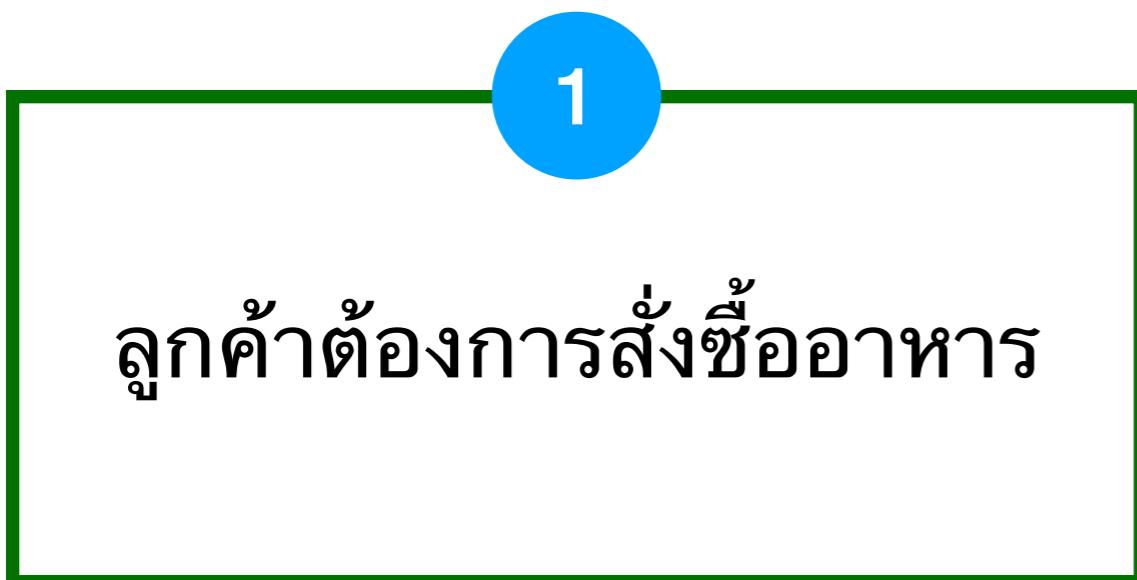
1. Identify system operations
2. Identify services
3. Define service APIs and collaboration



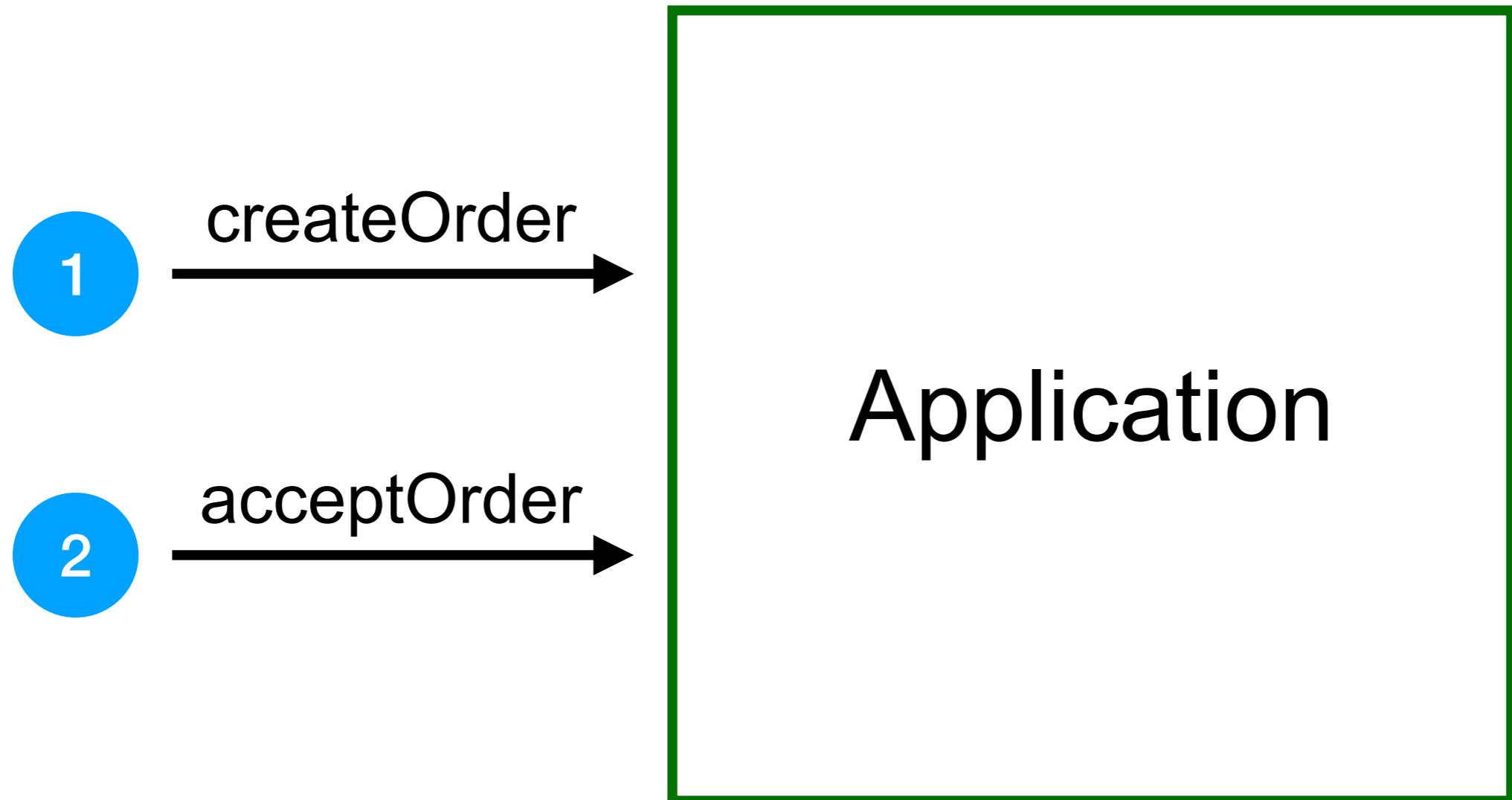
1. Identify system operations

Start with functional requirements

User story



1. Identify system operations



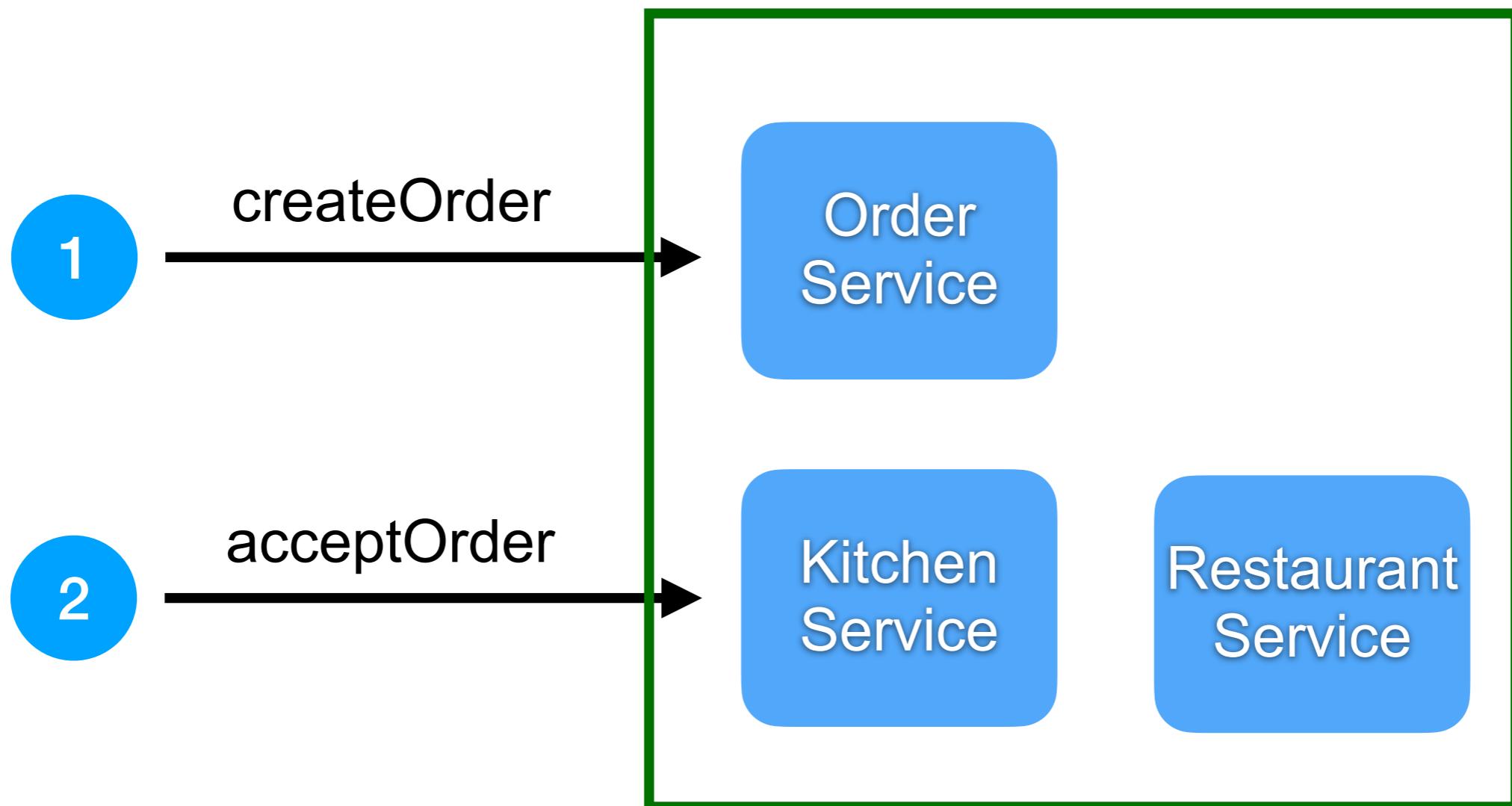
2. Identify services

Try to decomposition into services

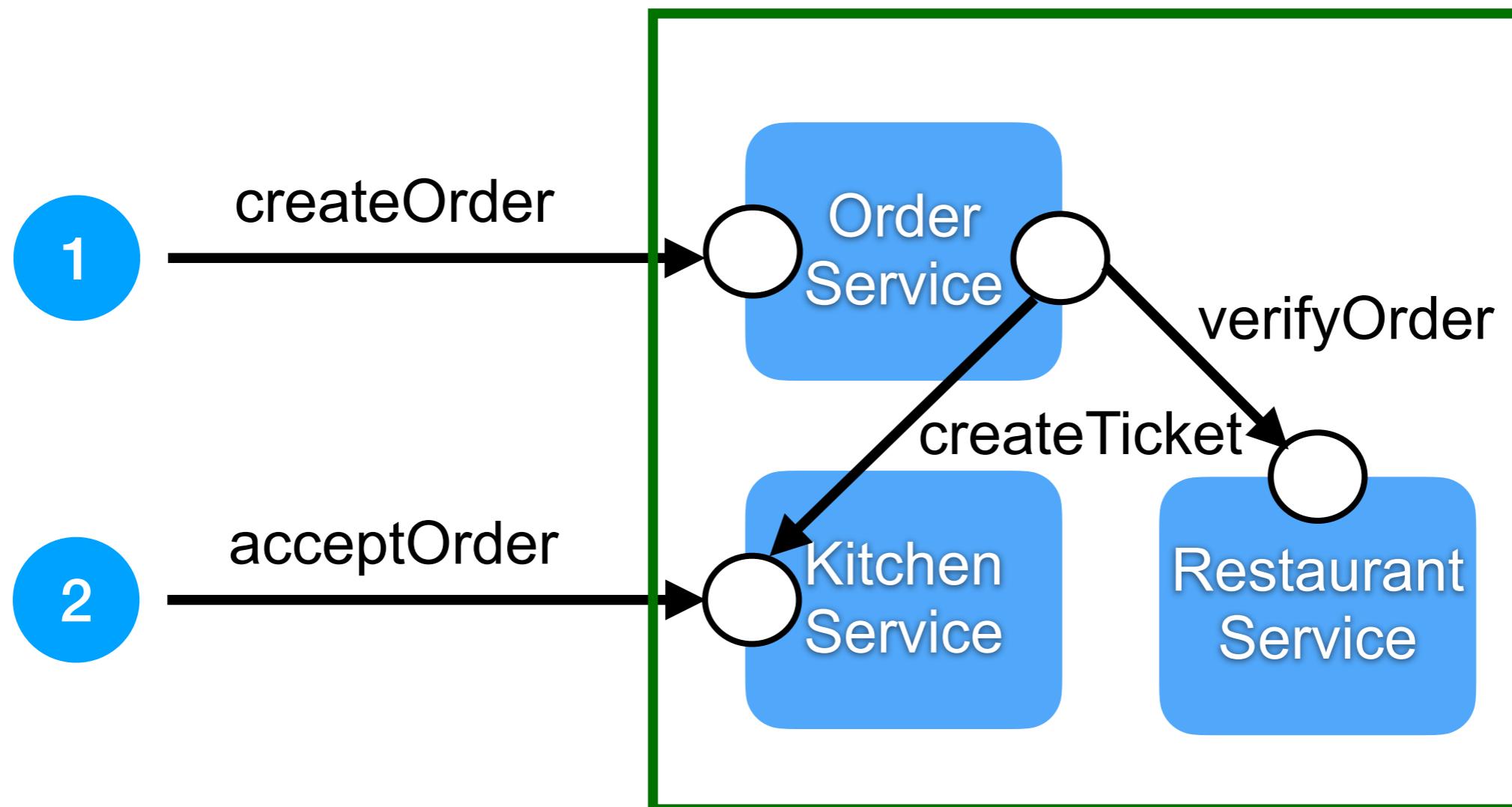
Business > Technical concept
What > How



2. Identify services



3. Define service APIs and collaborations



Problems when decompose services ?

Network latency

Reliability of communication

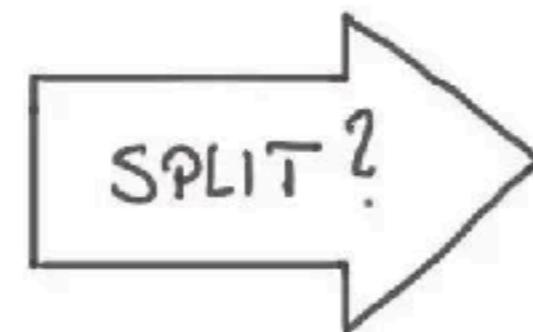
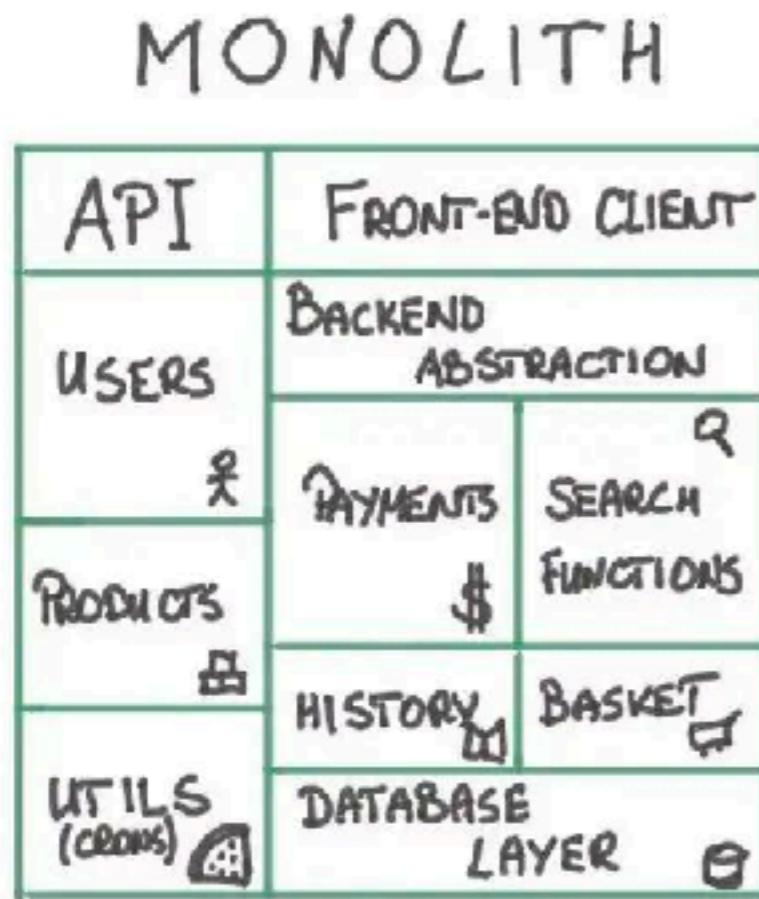
Maintain data consistency

God services !!

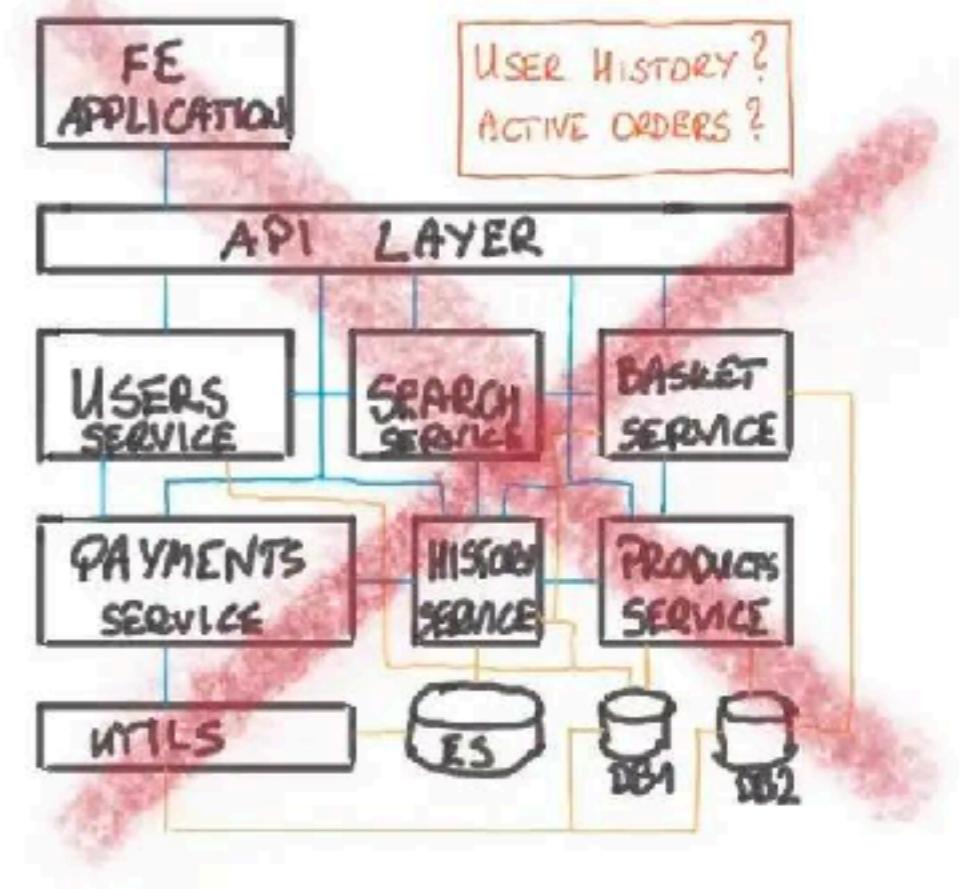
Microlith !!



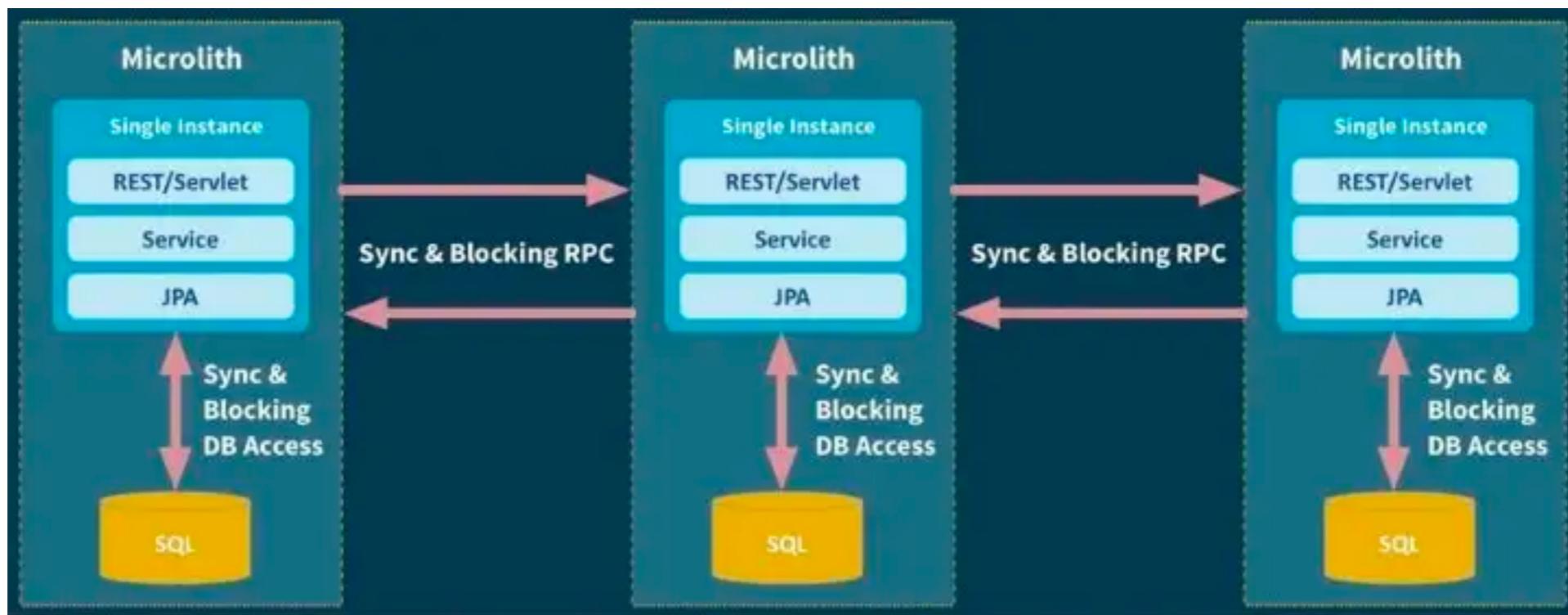
Microservices !!



MICRO SERVICES

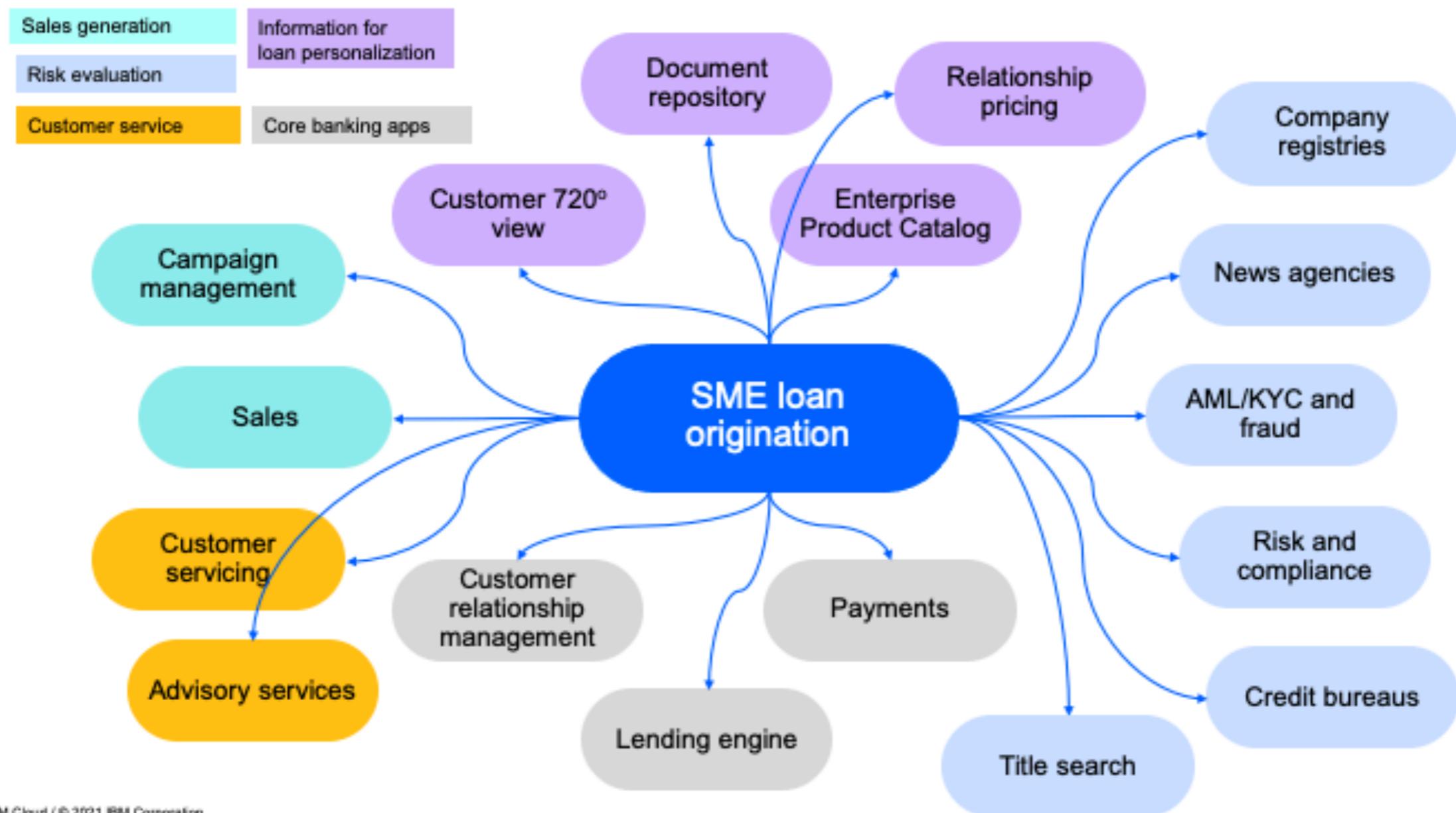


Microlith !!



Example of System Context

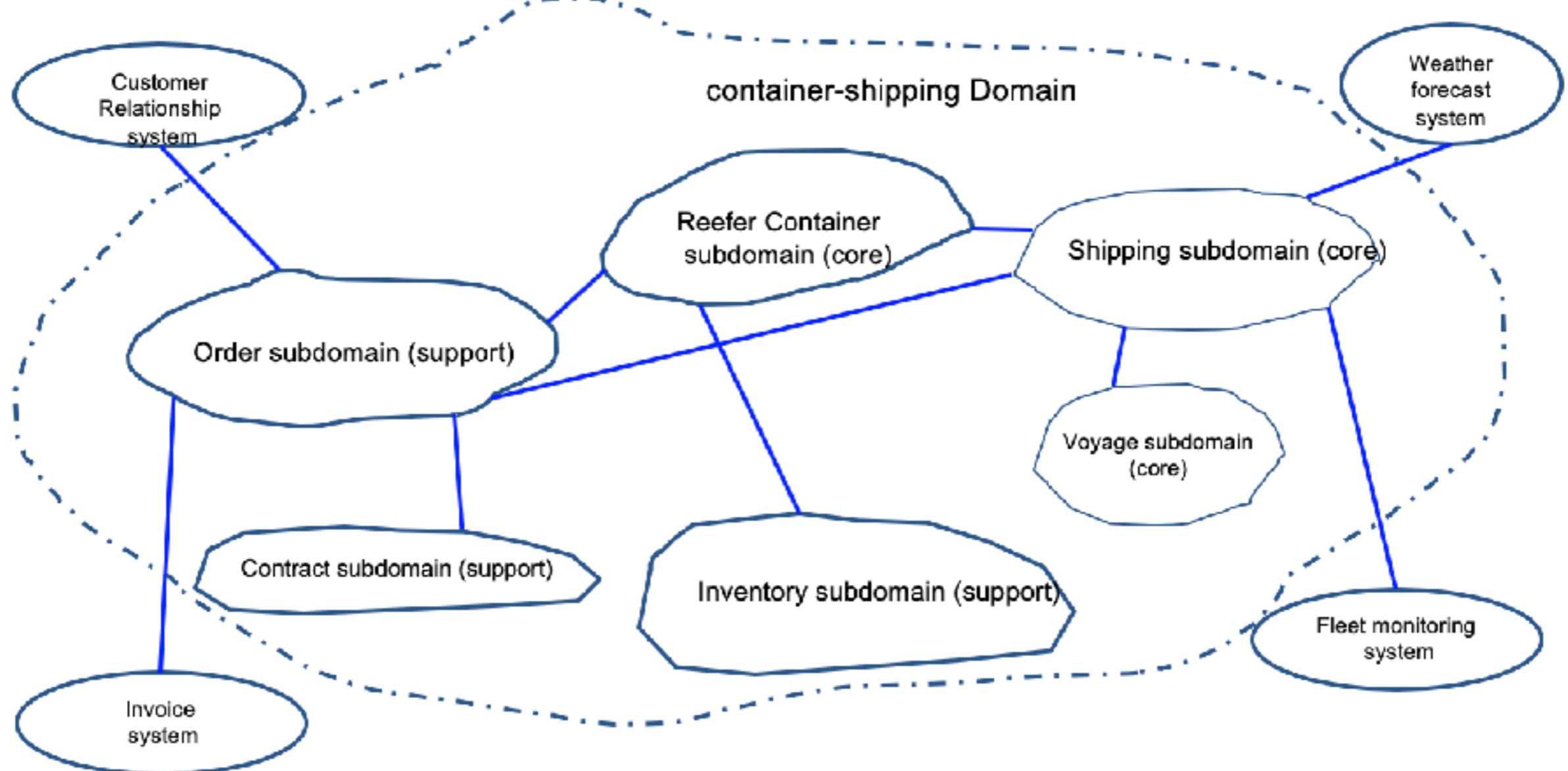
System context



IBM Cloud / © 2021 IBM Corporation



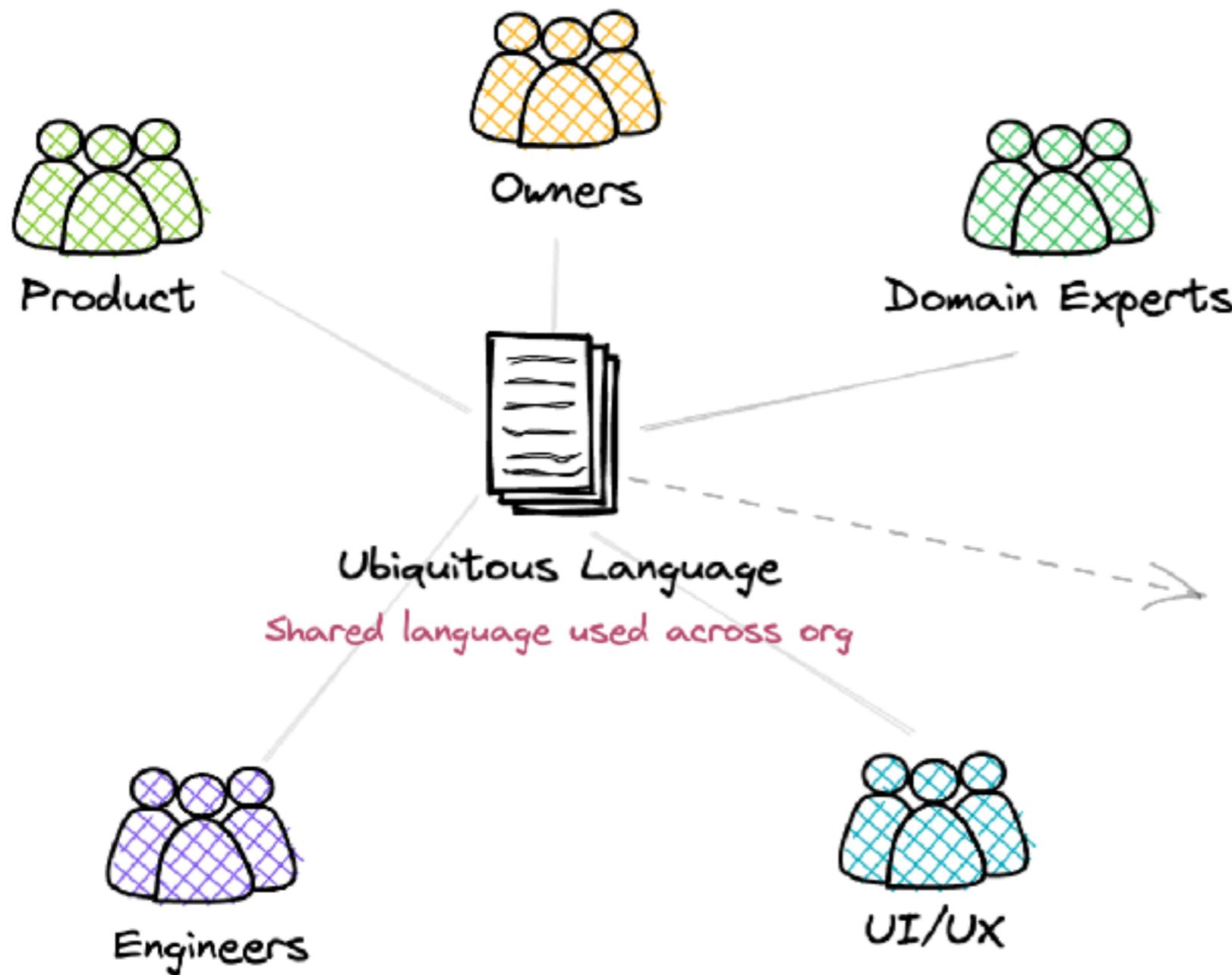
Assess domains and sub-domains



Define the ubiquitous language



Define the ubiquitous language



What is Ubiquitous Language

1. Language of the business
2. Reduces domain translations
3. Improve communication
4. Single Language
5. Terms easy to understand
6. Precise and consistent
7. Eliminate the need for assumptions
8. Single meaning for each term
9. Part of DDD
10. Use when describing business domain

@boyney123

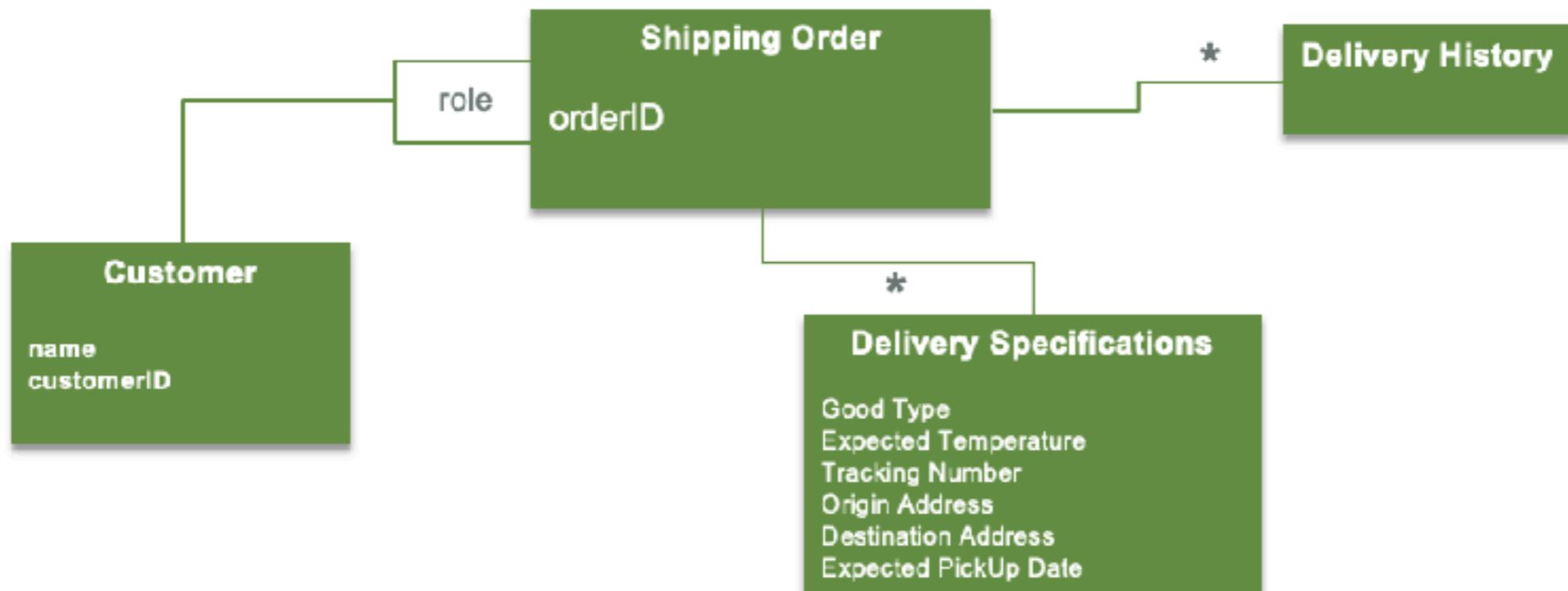
<https://serverlessland.com/event-driven-architecture/visuals/ubiquitous-language>



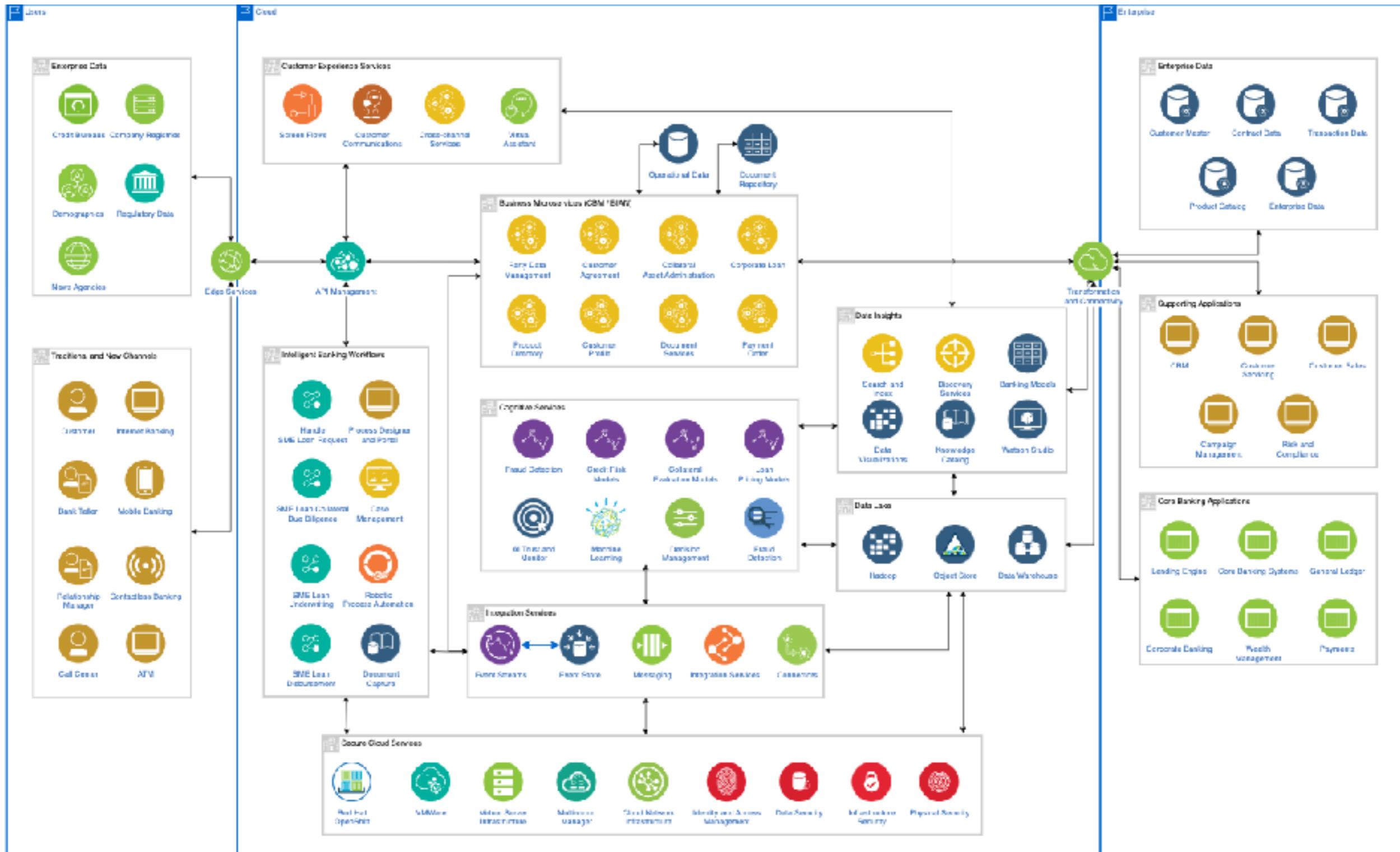
Microservices

© 2020 - 2023 Siam Chamnkit Company Limited. All rights reserved.

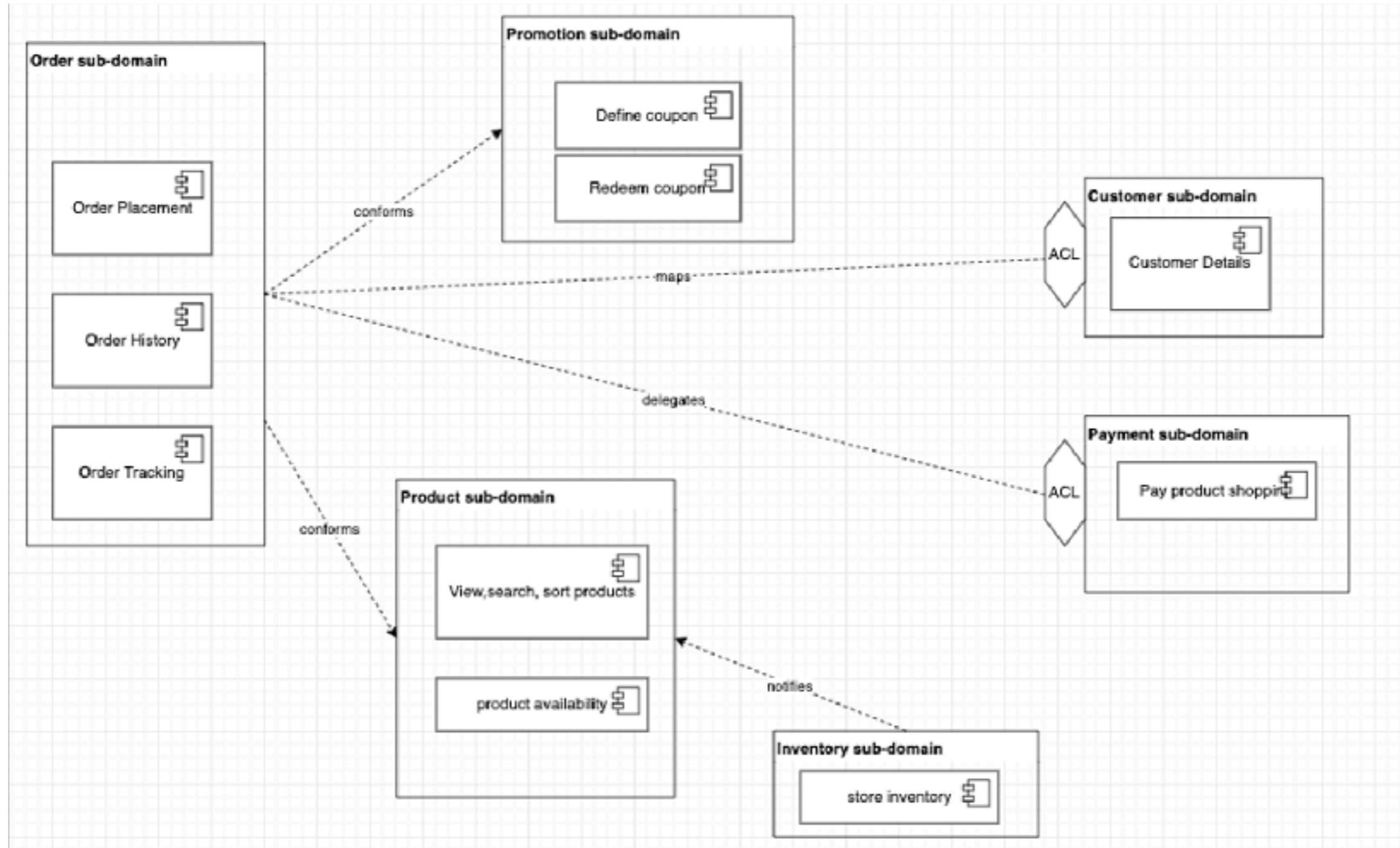
Structure of data



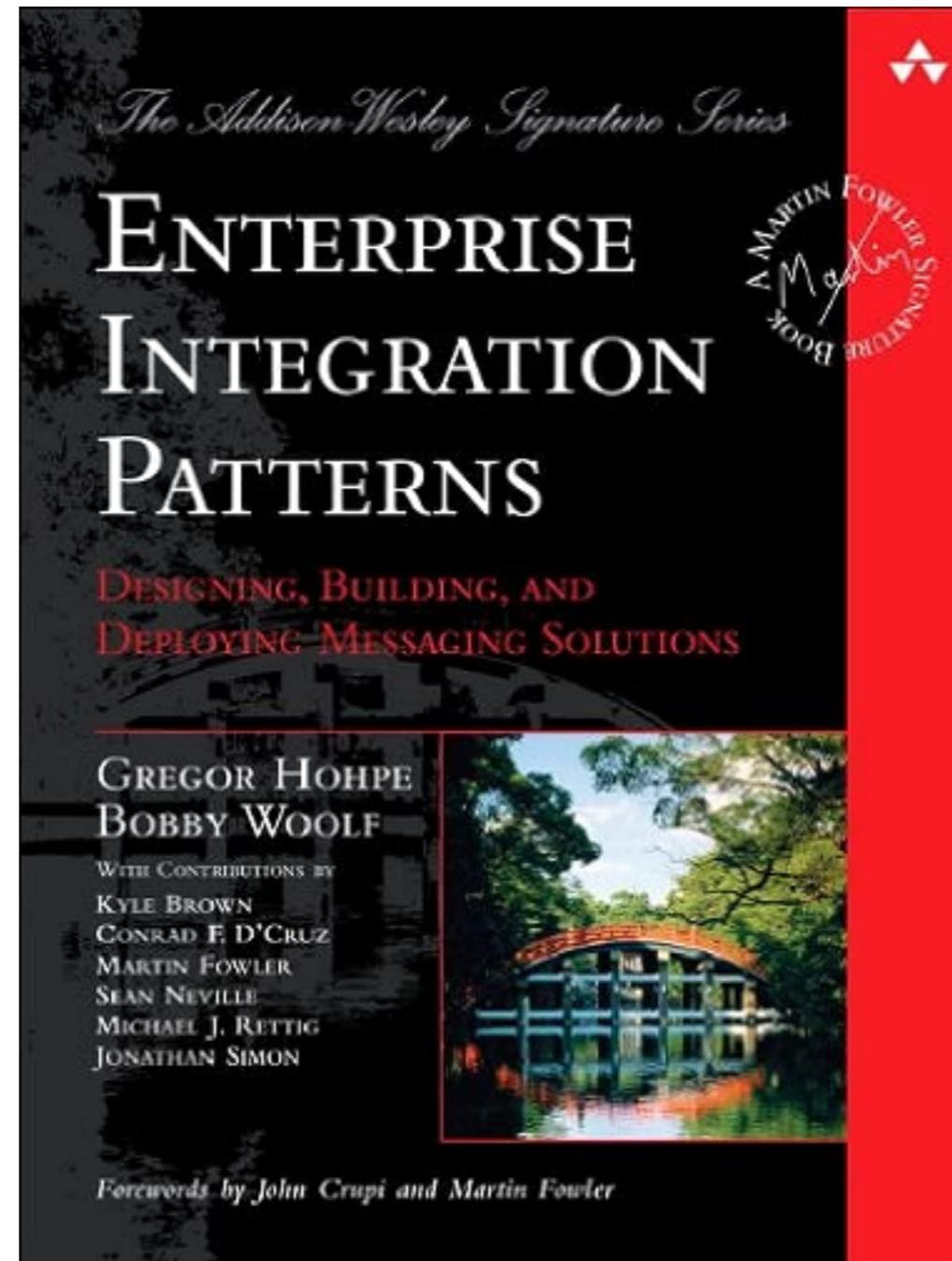
Grouping services



Communication types ?



Enterprise Integration Patterns



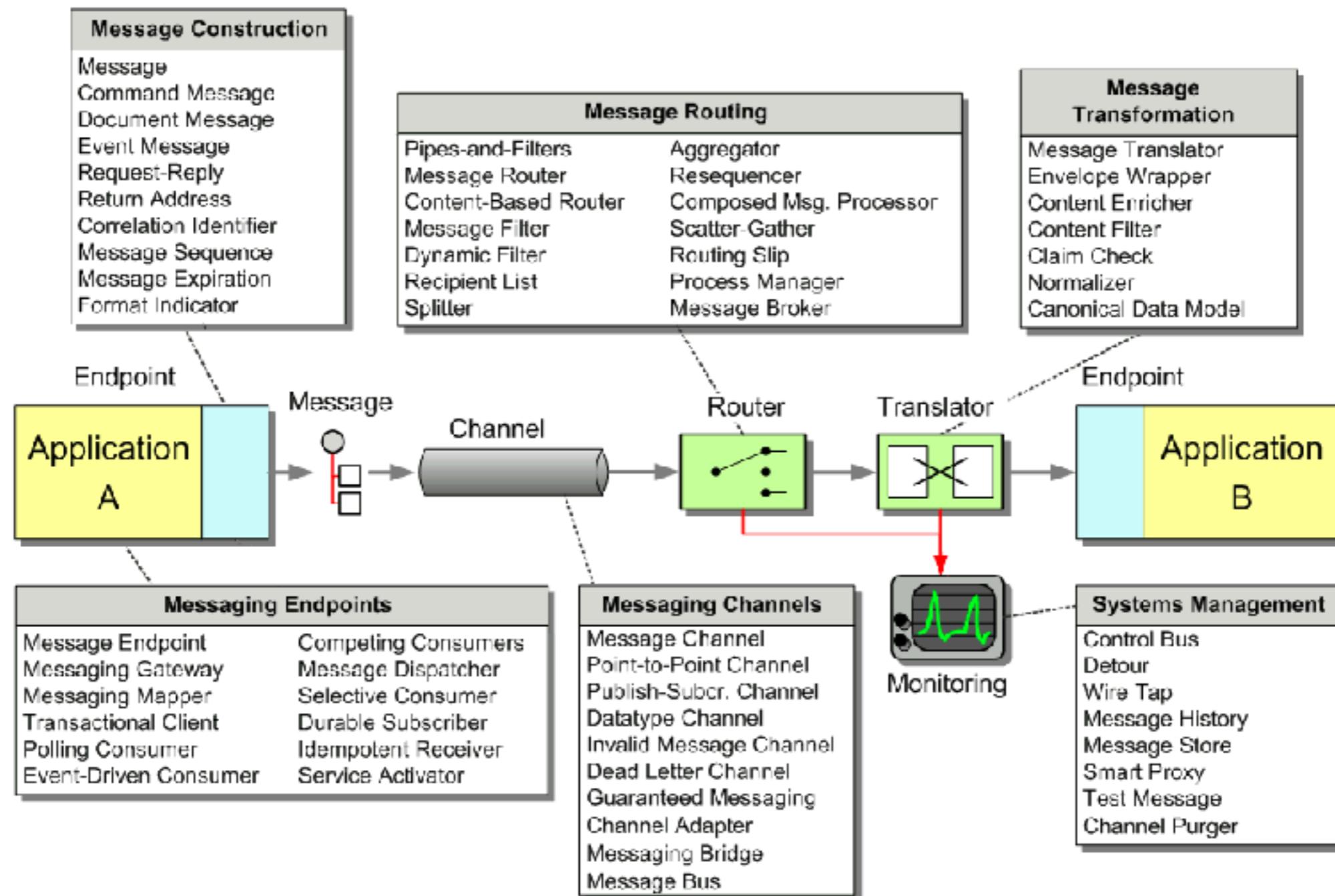
<https://www.amazon.com/Enterprise-Integration-Patterns-Designing-Deploying/dp/0321200683>



Microservices

© 2020 - 2023 Siam Chamnankit Company Limited. All rights reserved.

Integration Pattern Language



<https://www.enterpriseintegrationpatterns.com/patterns/messaging/>



Microservices

© 2020 - 2023 Siam Chamnkit Company Limited. All rights reserved.

Communication between services



Interaction Styles

	One-to-one	One-to-many
Synchronous	Request/response	-
Asynchronous	Async request/response	Publish/subscribe
	Notification	Publish/async response



Synchronous Communication

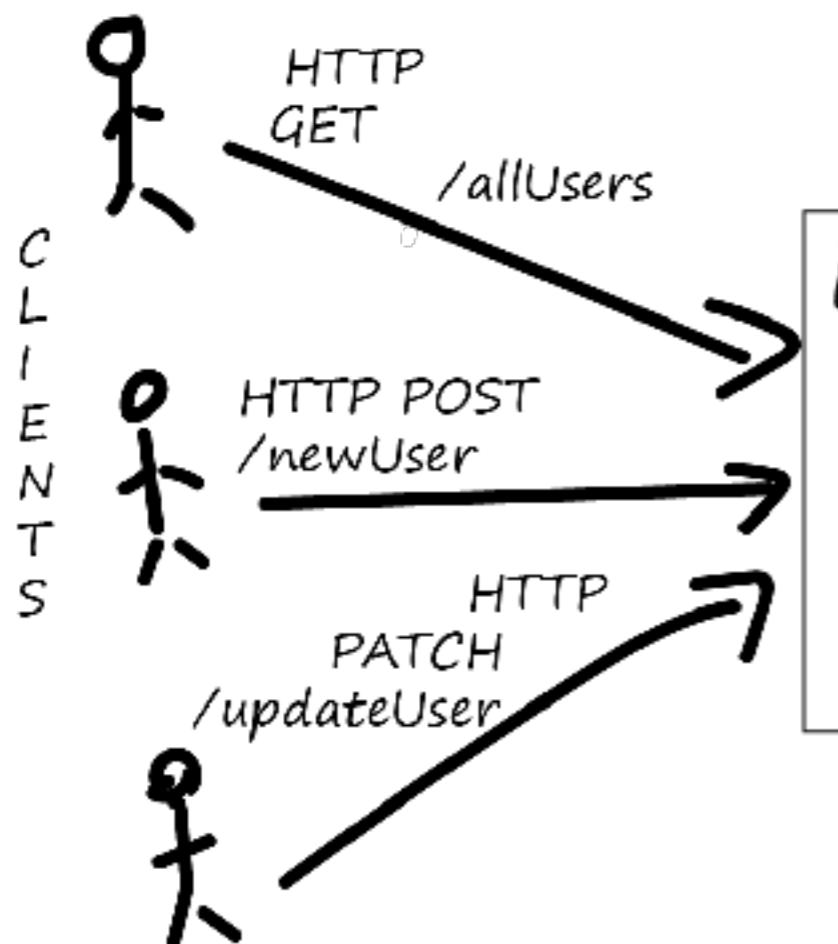
REST
gRPC

Handling failure with Circuit breaker
Service discovery

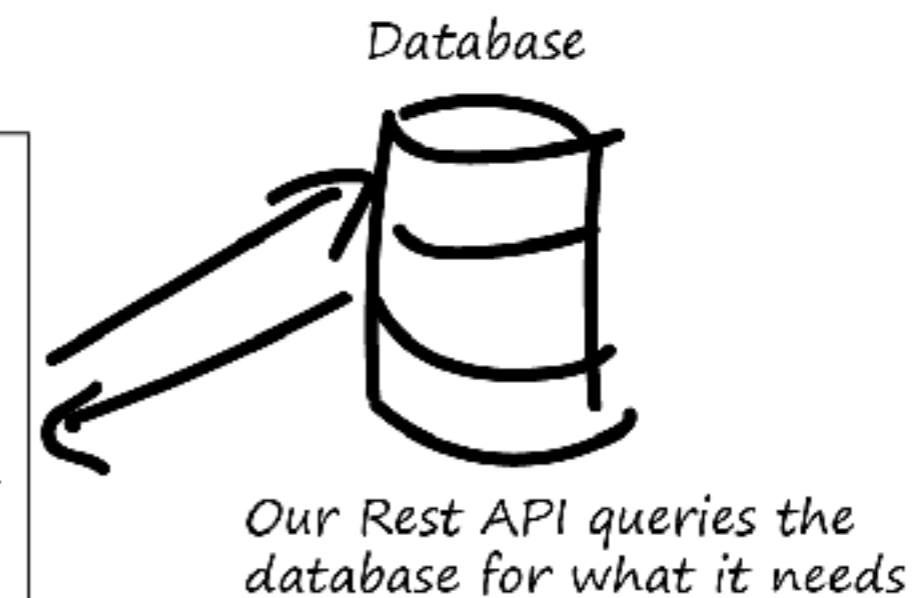


REpresentational State Transfer

Rest API Basics



Typical HTTP Verbs:
GET → Read from Database
PUT → Update/Replace row in Database
PATCH → Update/Modify row in Database
POST → Create a new record in the database
DELETE → Delete from the database



Response: When the Rest API has what it needs, it sends back a response to the clients. This would typically be in JSON or XML format.

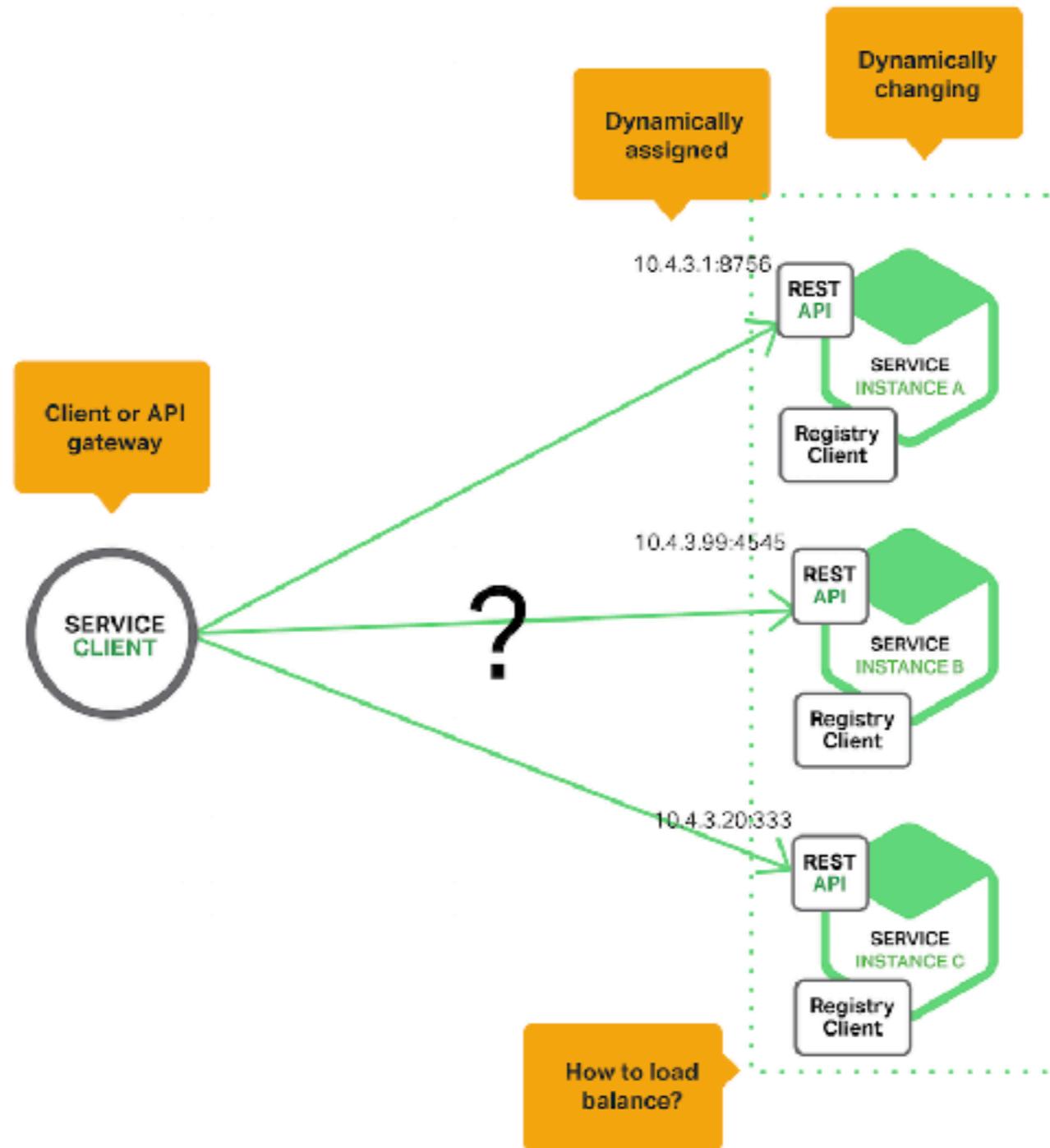


REST :: mapping with HTTP verbs

Activity	HTTP Method
Retrieve data	GET
Create new data	POST
Update data	PUT
Delete data	DELETE



Service Discovery ?



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

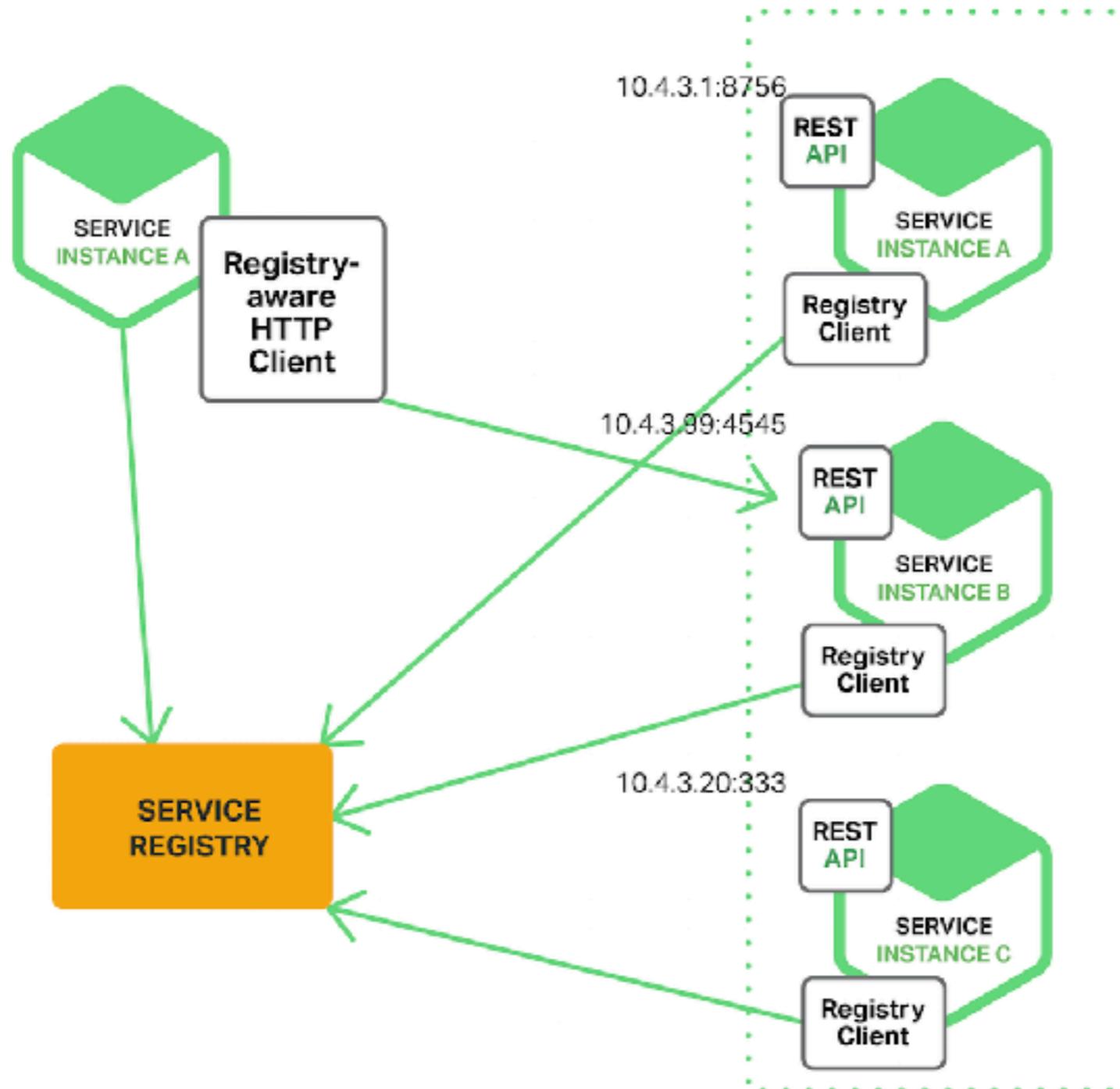


Service Discovery ?

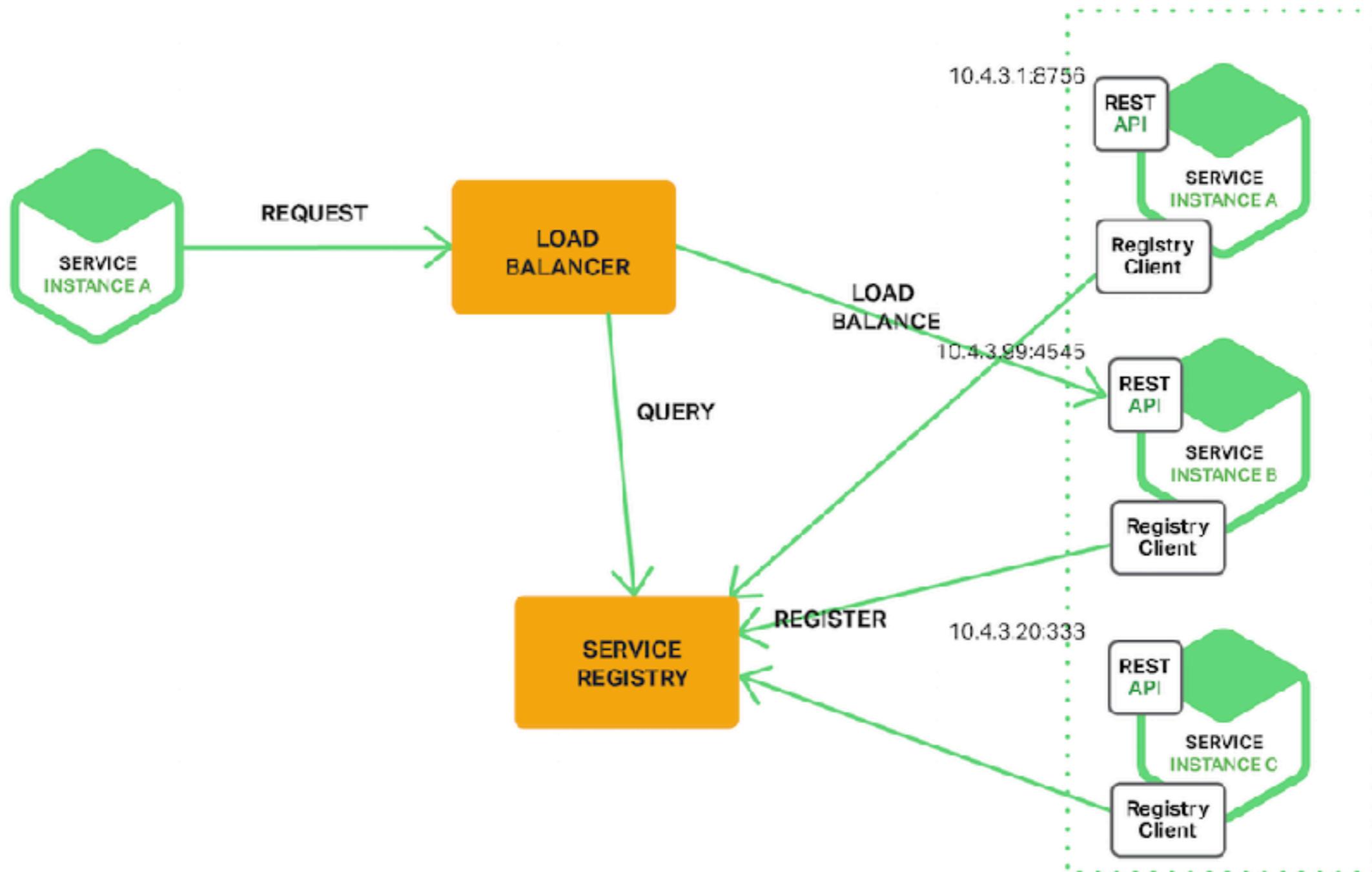
Client-side
Server-side



Client-side Discovery



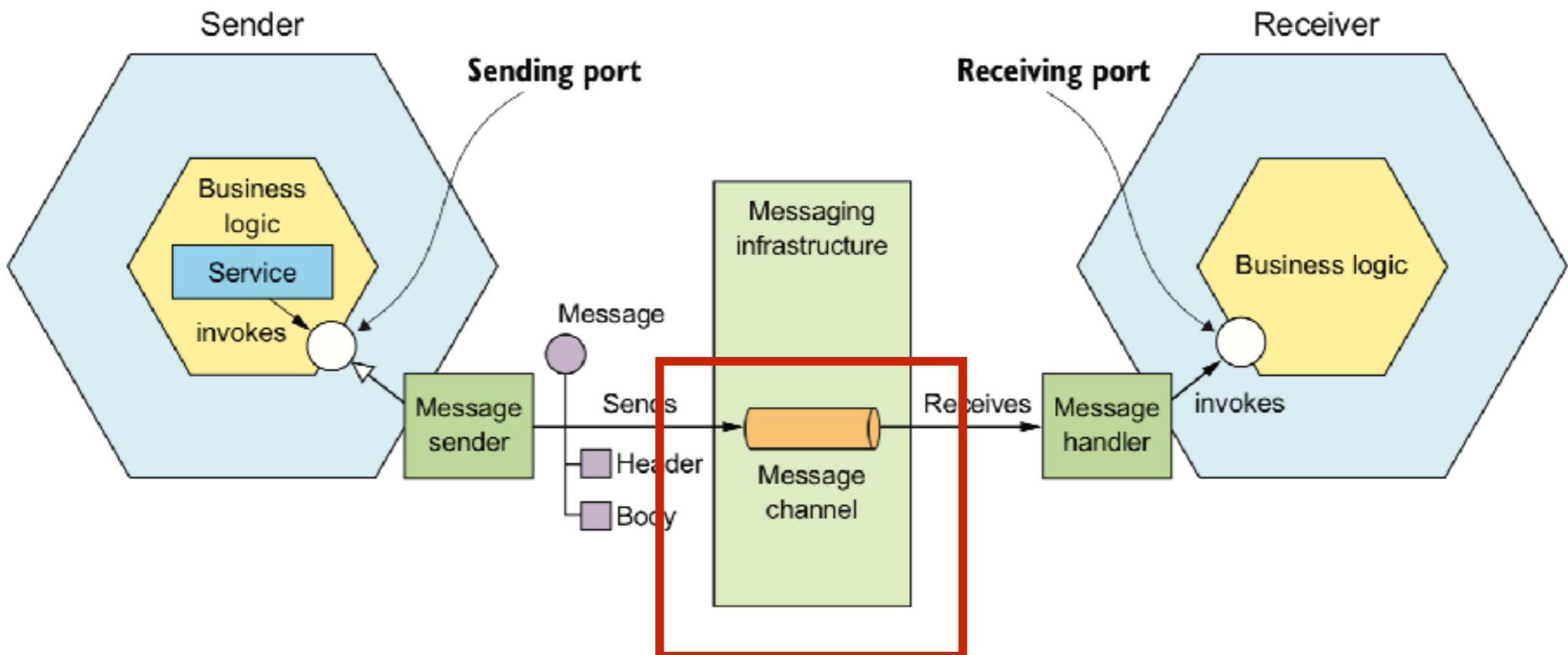
Server-side Discovery



Asynchronous communication



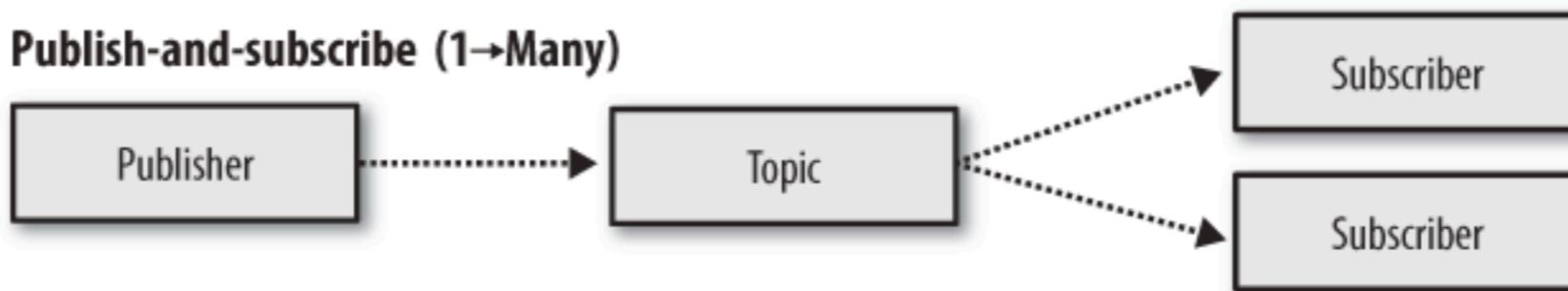
Asynchronous messaging pattern



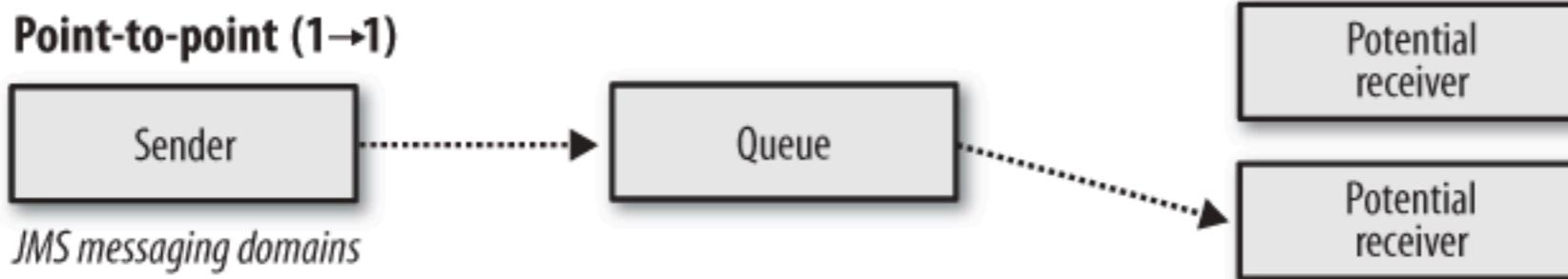
Messaging channels

Publish-subscribe Point-to-point

Publish-and-subscribe (1→Many)

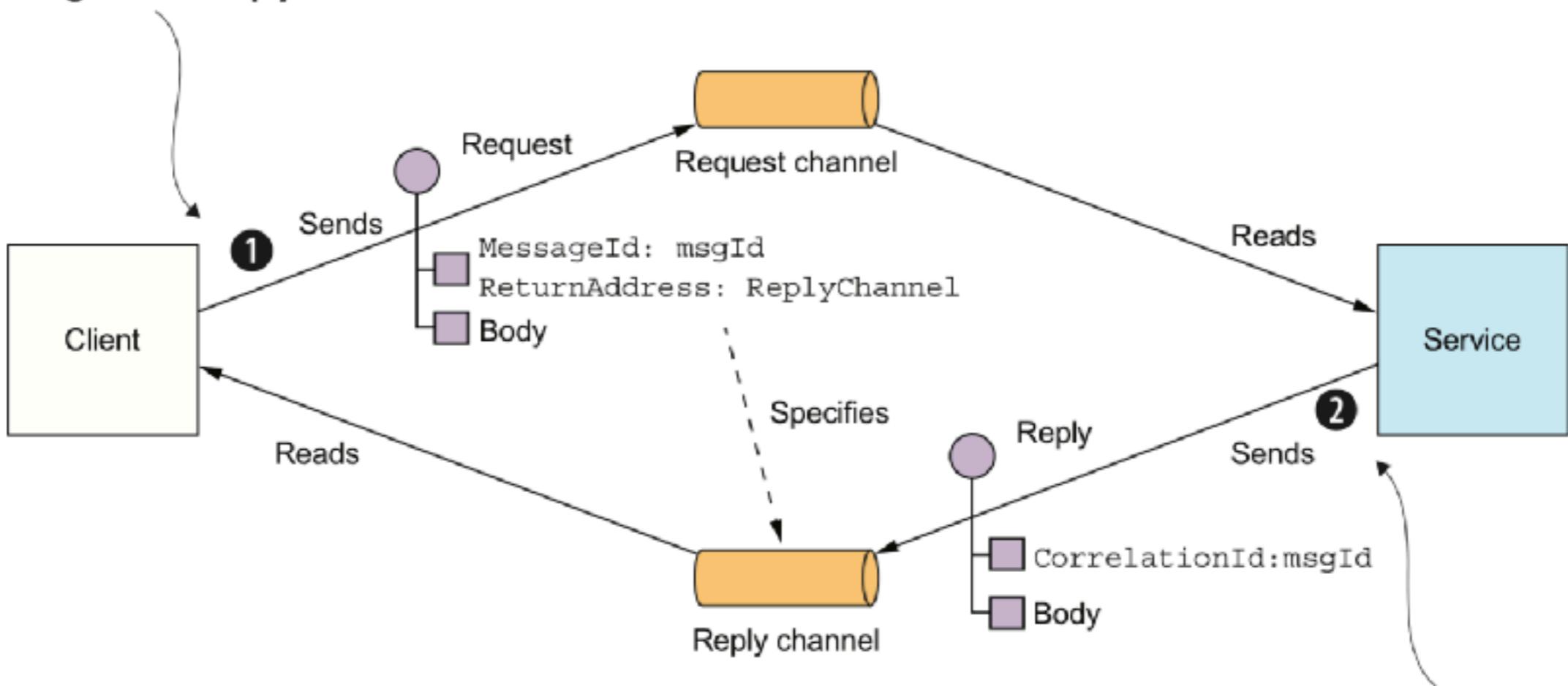


Point-to-point (1→1)



Async request/response (1)

Client sends message containing msgId and a reply channel.



Service sends reply to the specified reply channel. The reply contains a correlationId, which is the request's msgId.



Message brokers



Benefits

Loose-coupling
Message buffer
Flexible communication



Drawbacks

Performance bottleneck
Single point of failure
Operational complexity

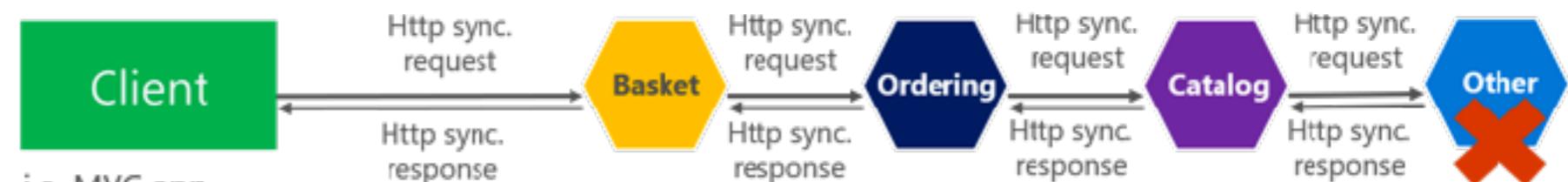


Communication !!

Synchronous vs. async communication across microservices

Anti-pattern

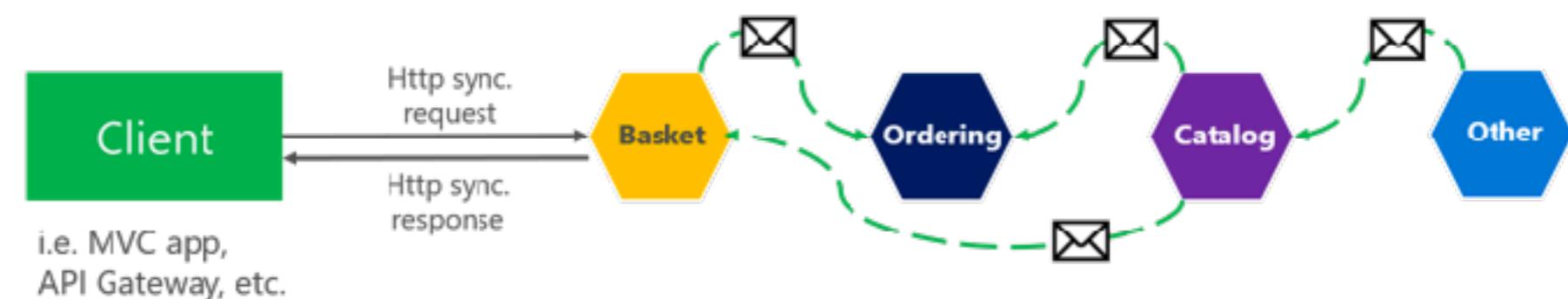
Synchronous
all req./resp. cycle



Same http request/response cycle!

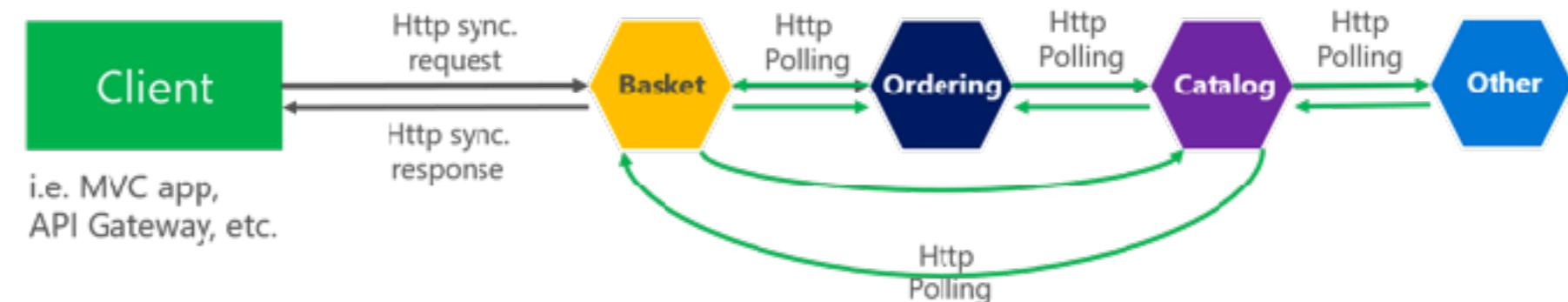
Asynchronous

Comm. across
internal microservices
(EventBus: i.e. **AMQP**)

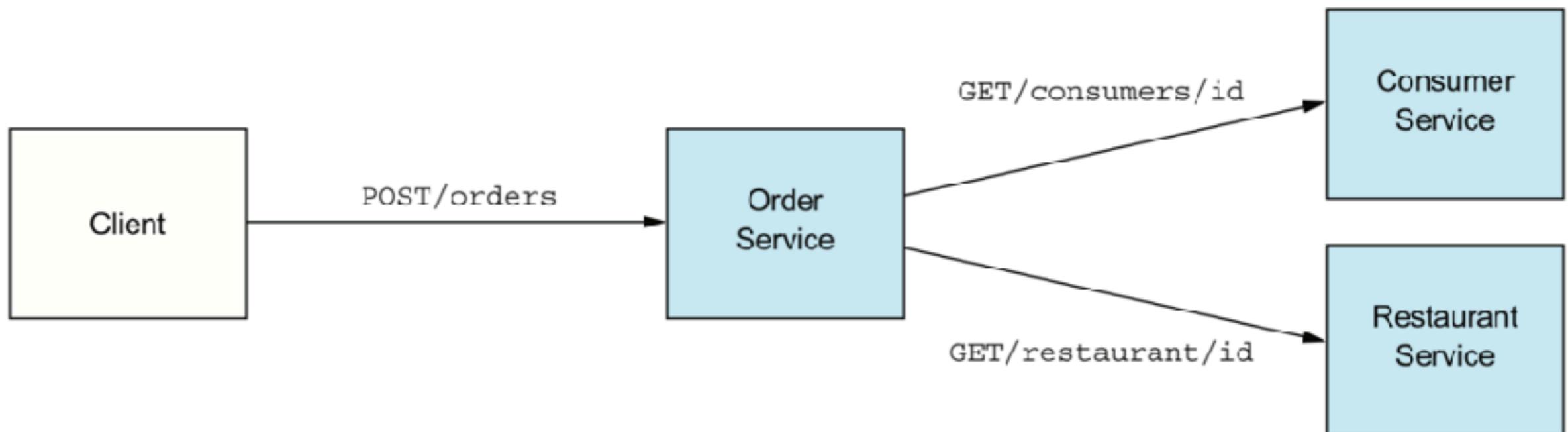


"Asynchronous"

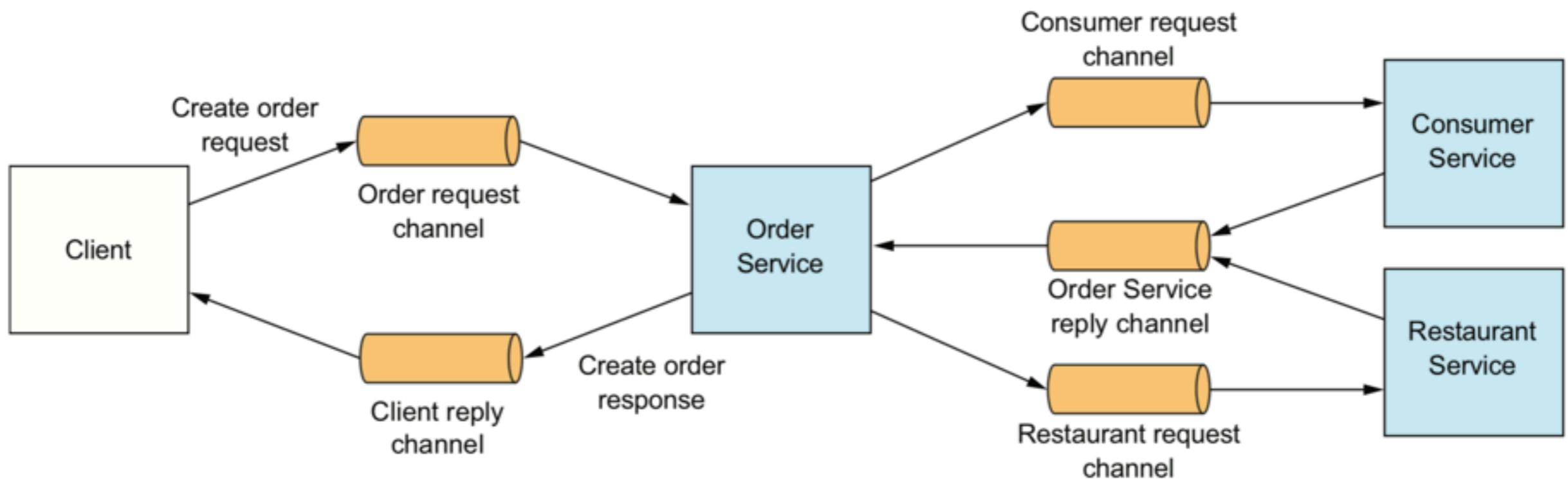
Comm. across
internal microservices
(Polling: **Http**)



Synchronous reduce availability



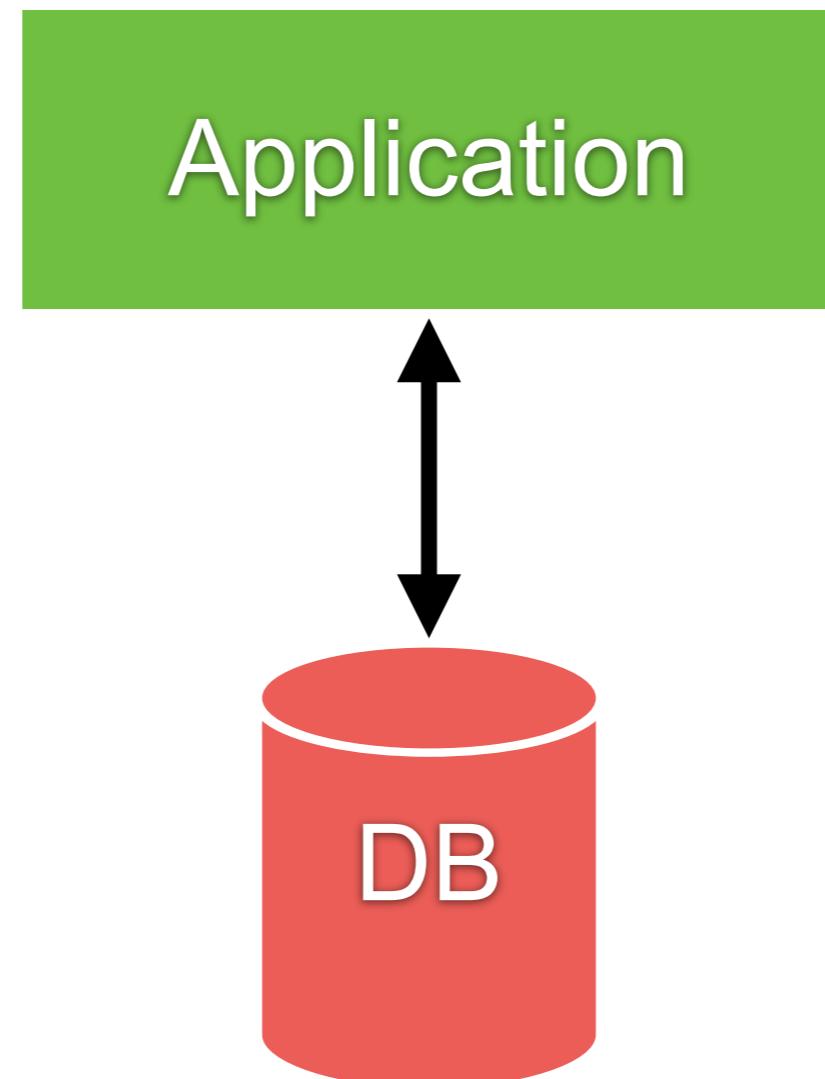
Replace with asynchronous (1)



Manage Data Consistency

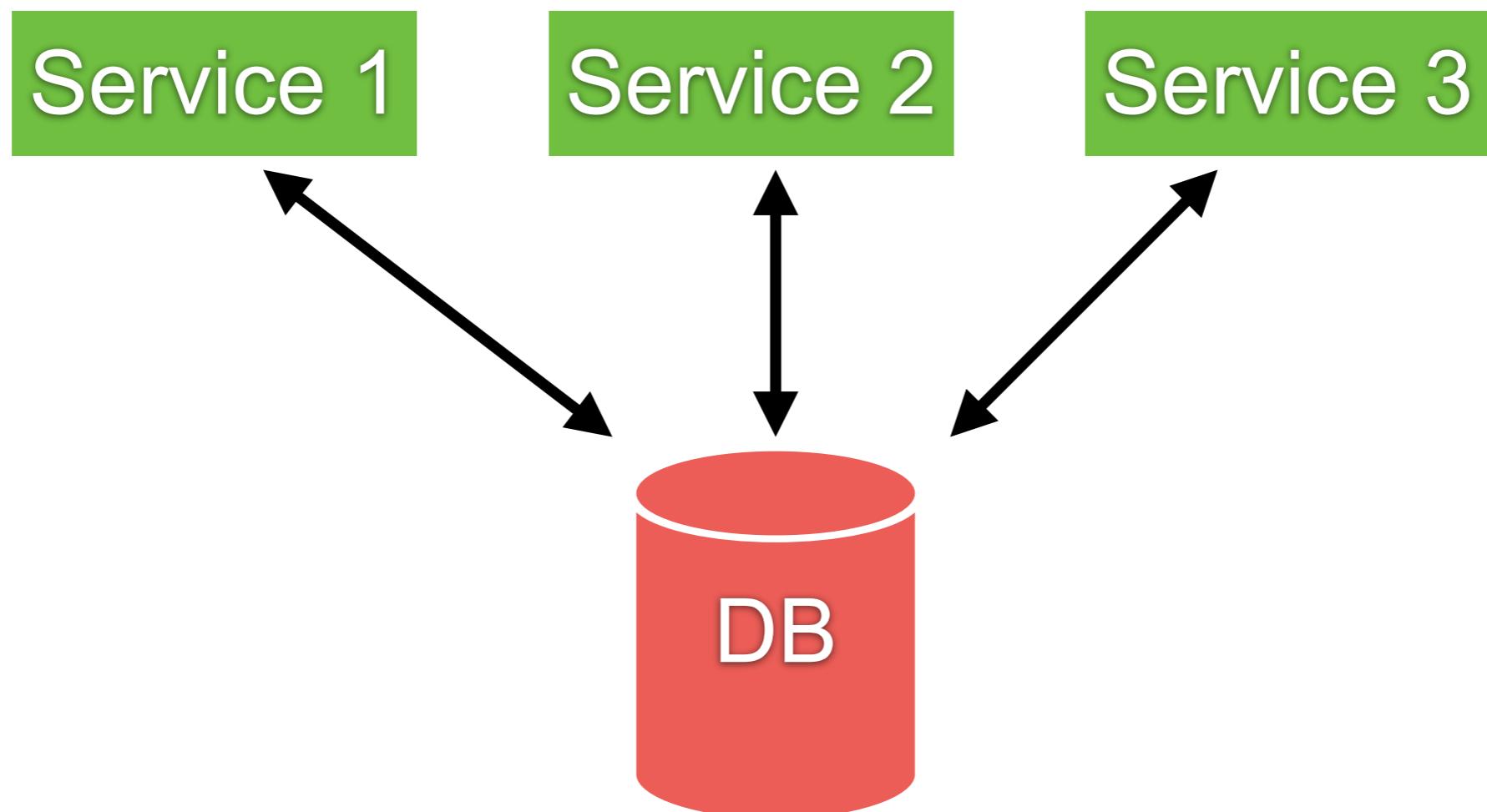


Monolithic



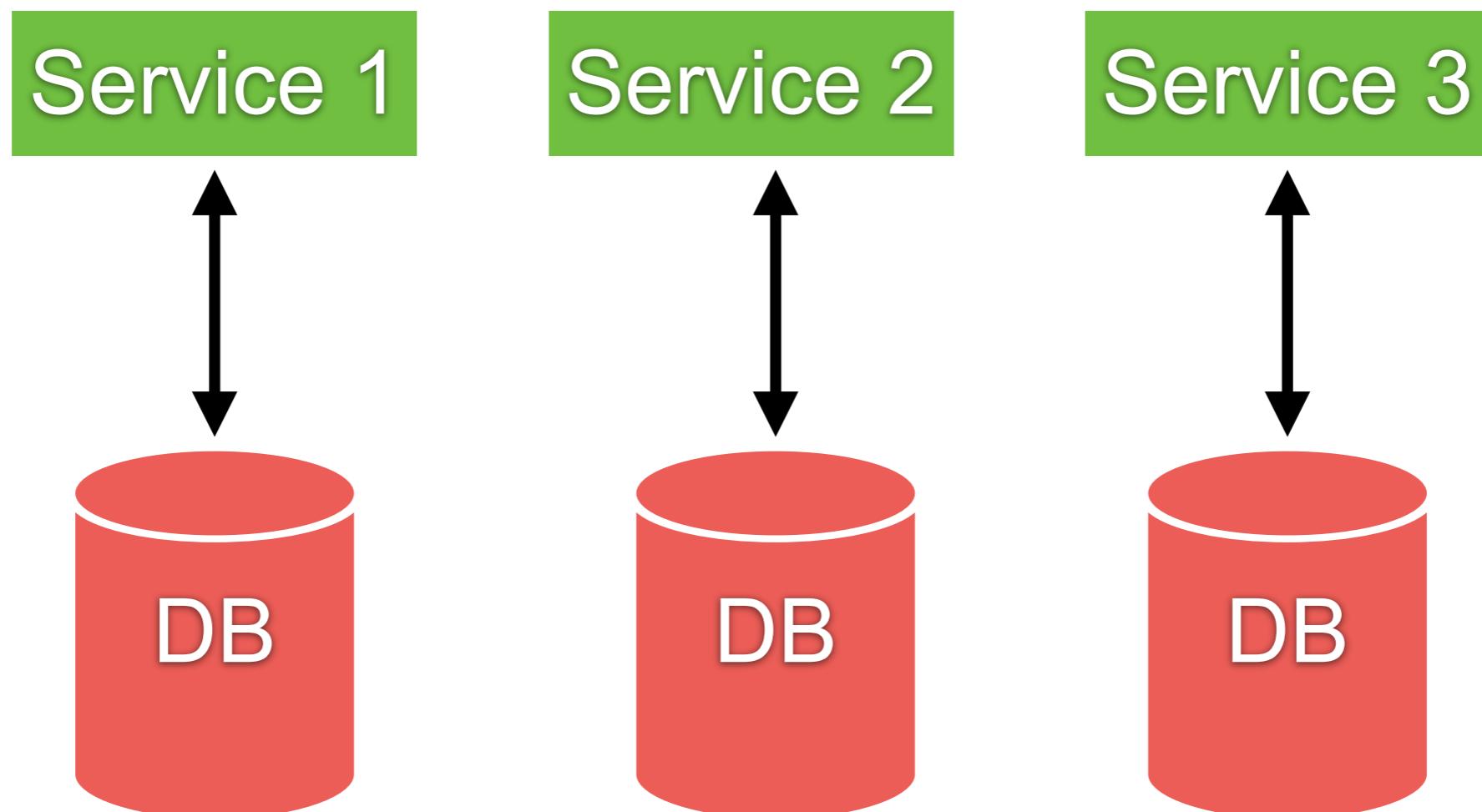
Microservices

Shared database

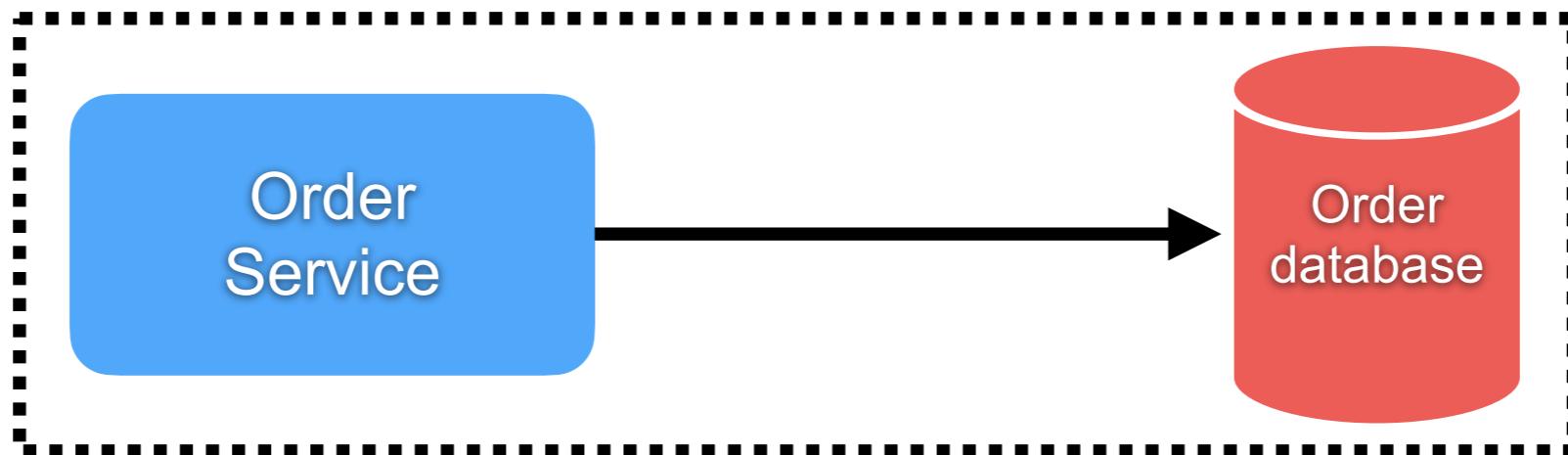


Microservices

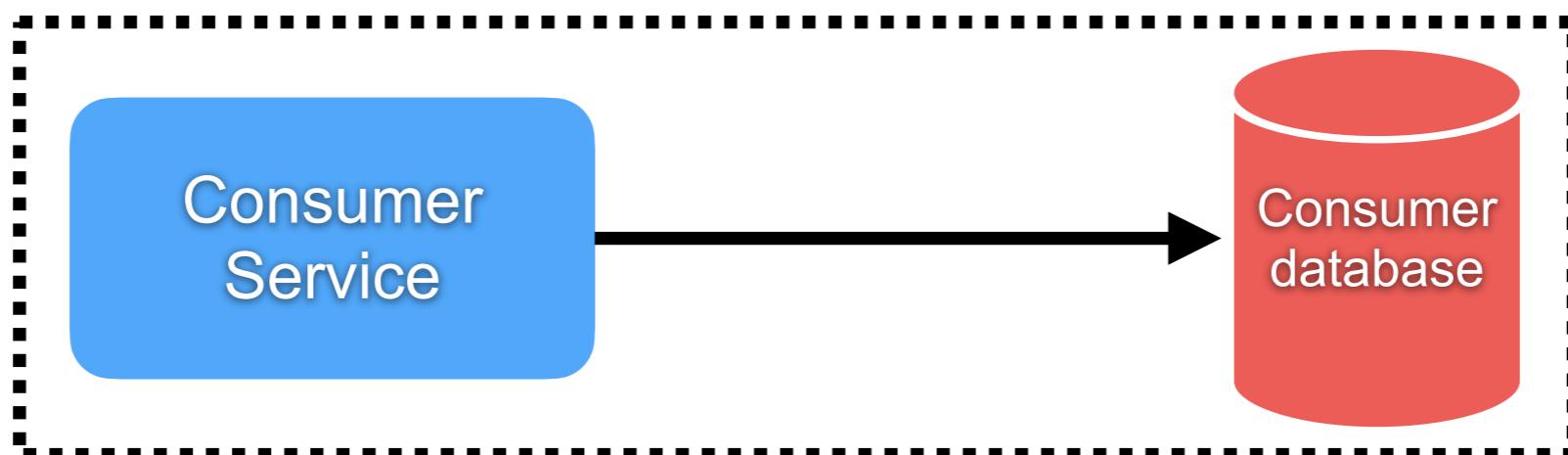
Database per service



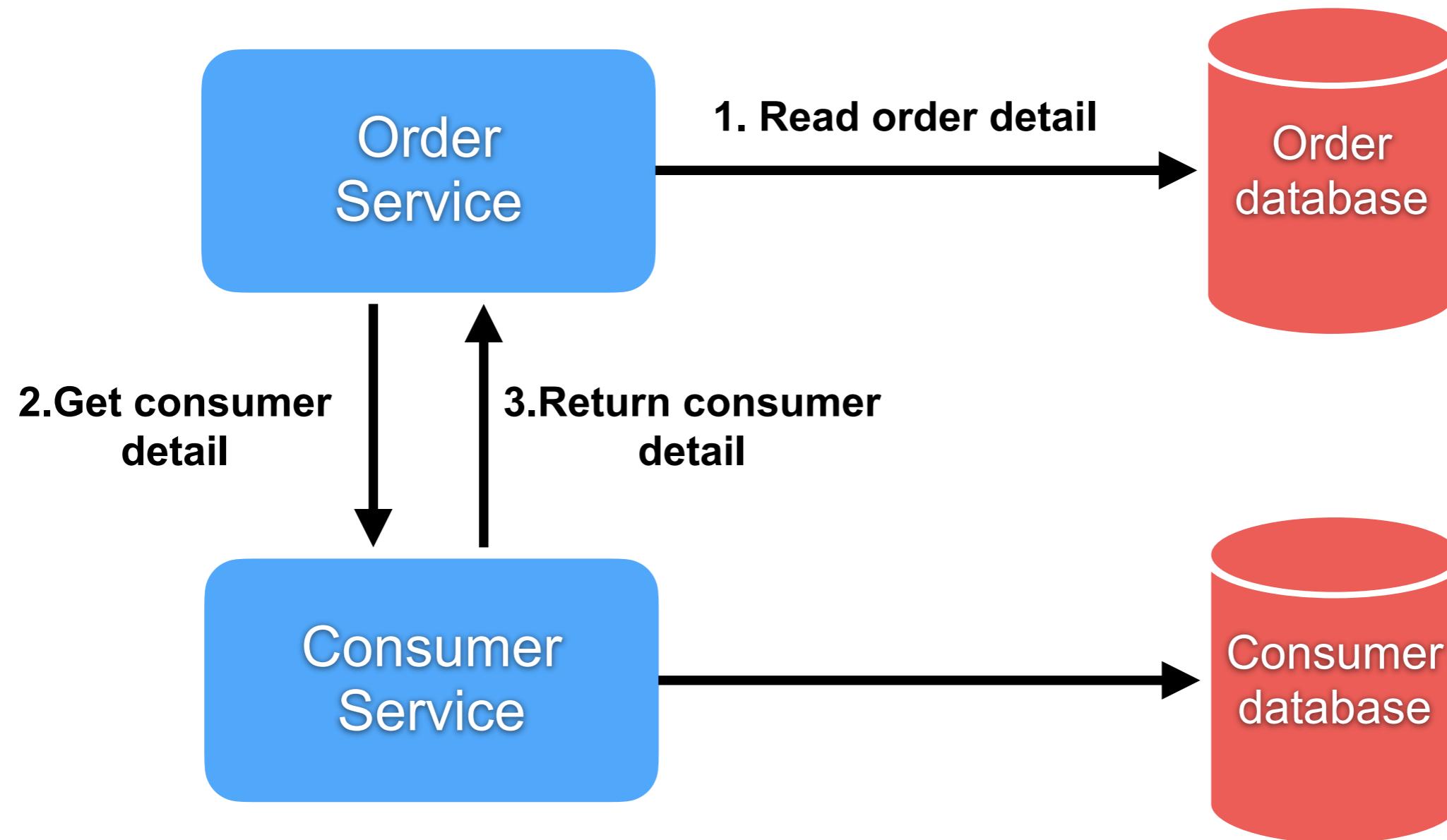
Database per service



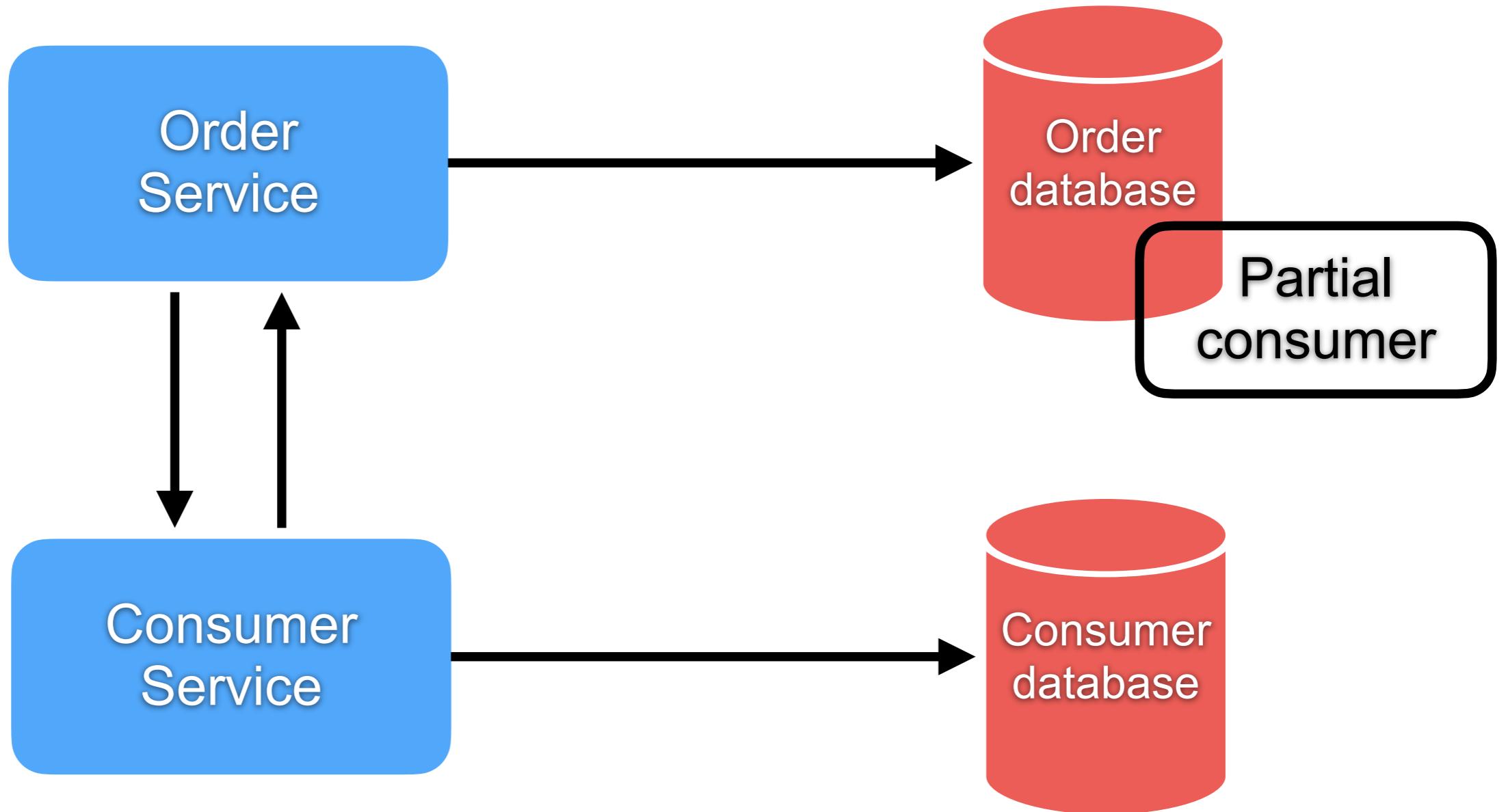
Strong Boundary



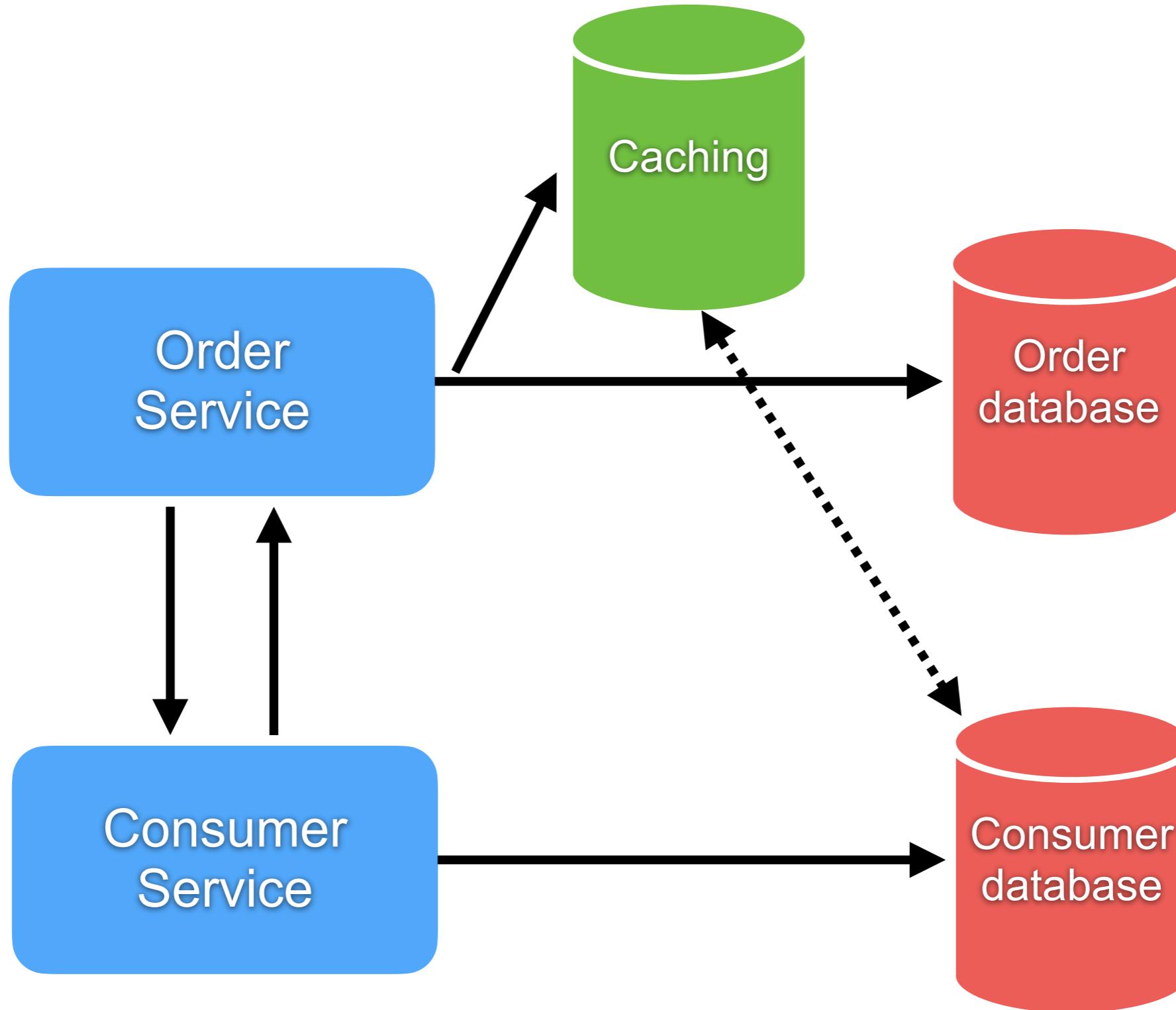
Loose coupling = encapsulation data



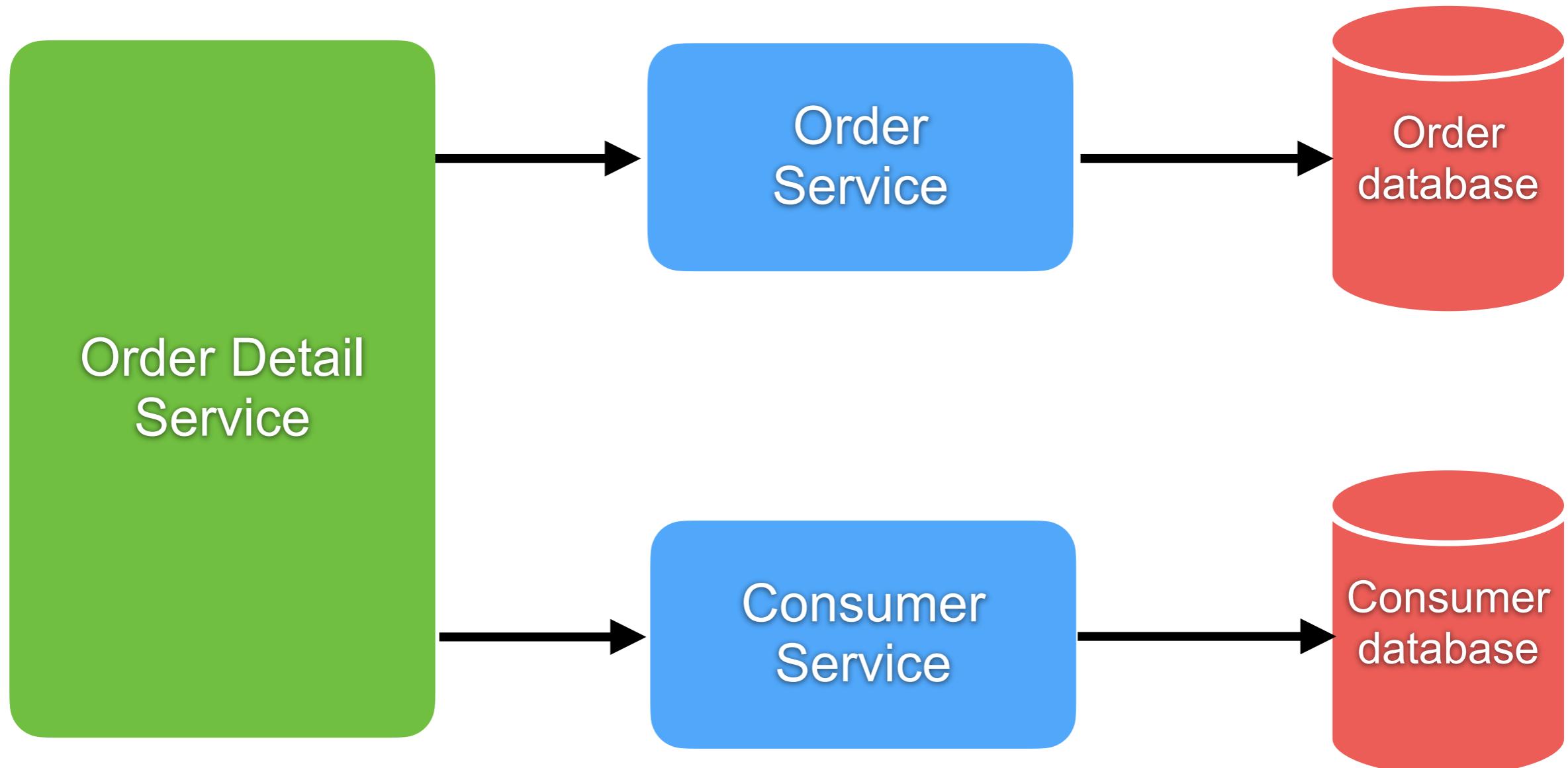
Store partial data



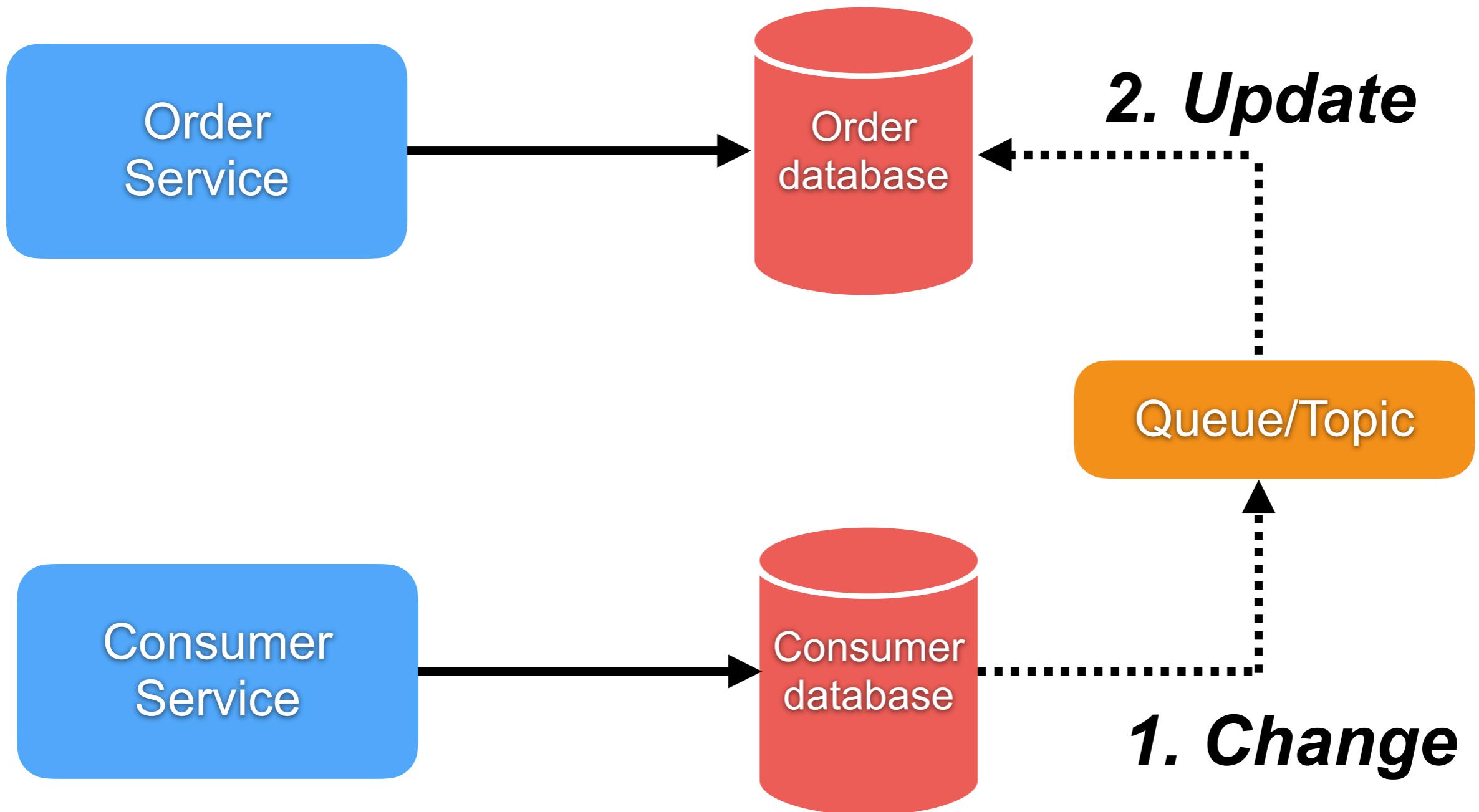
Caching data



Order detail orchestration service



Use messaging to shared data



Database per service

High complexity

Query data across multiple databases

Challenge “How to join data ?”

Maintain database consistency

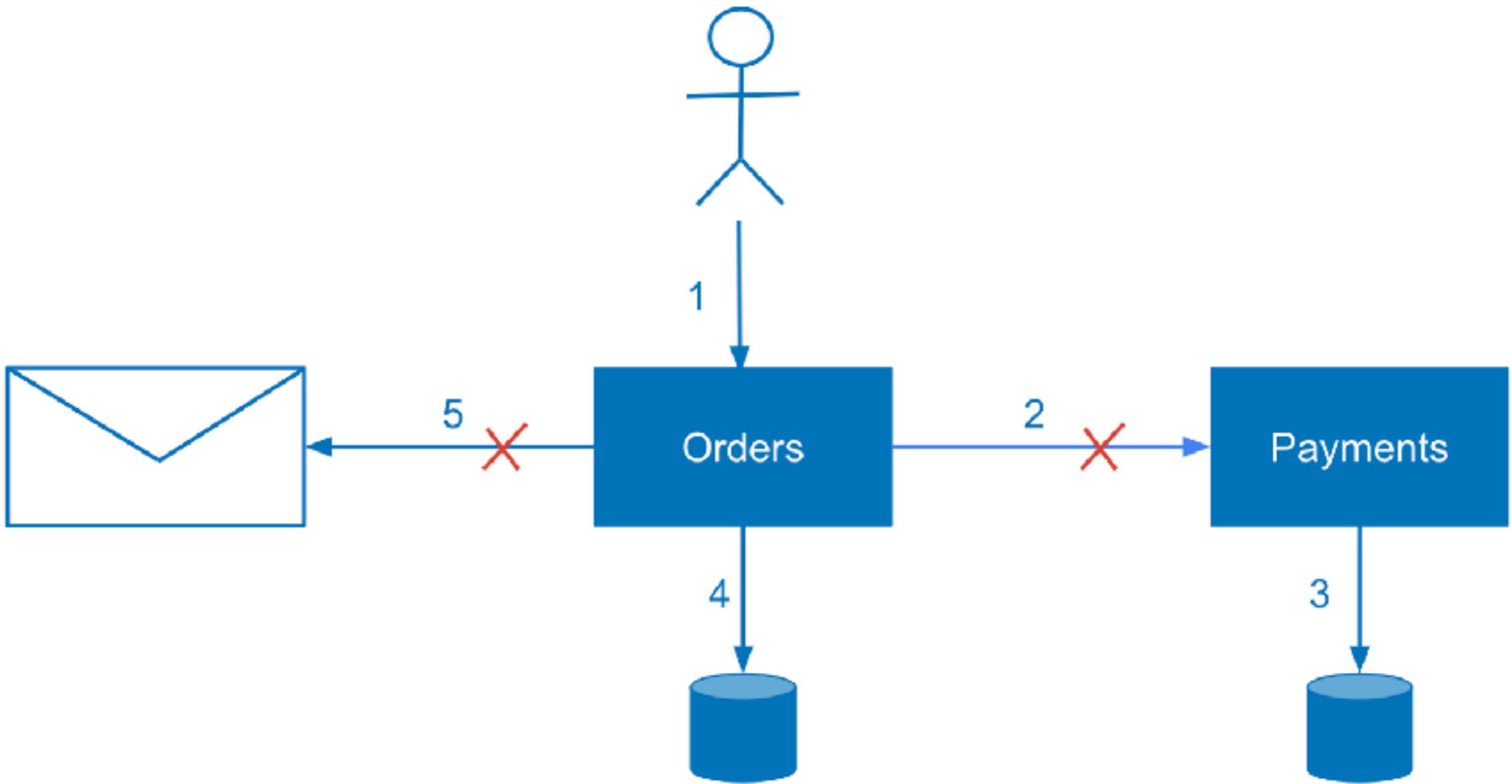


Manage transaction ?

Manage data consistency ?

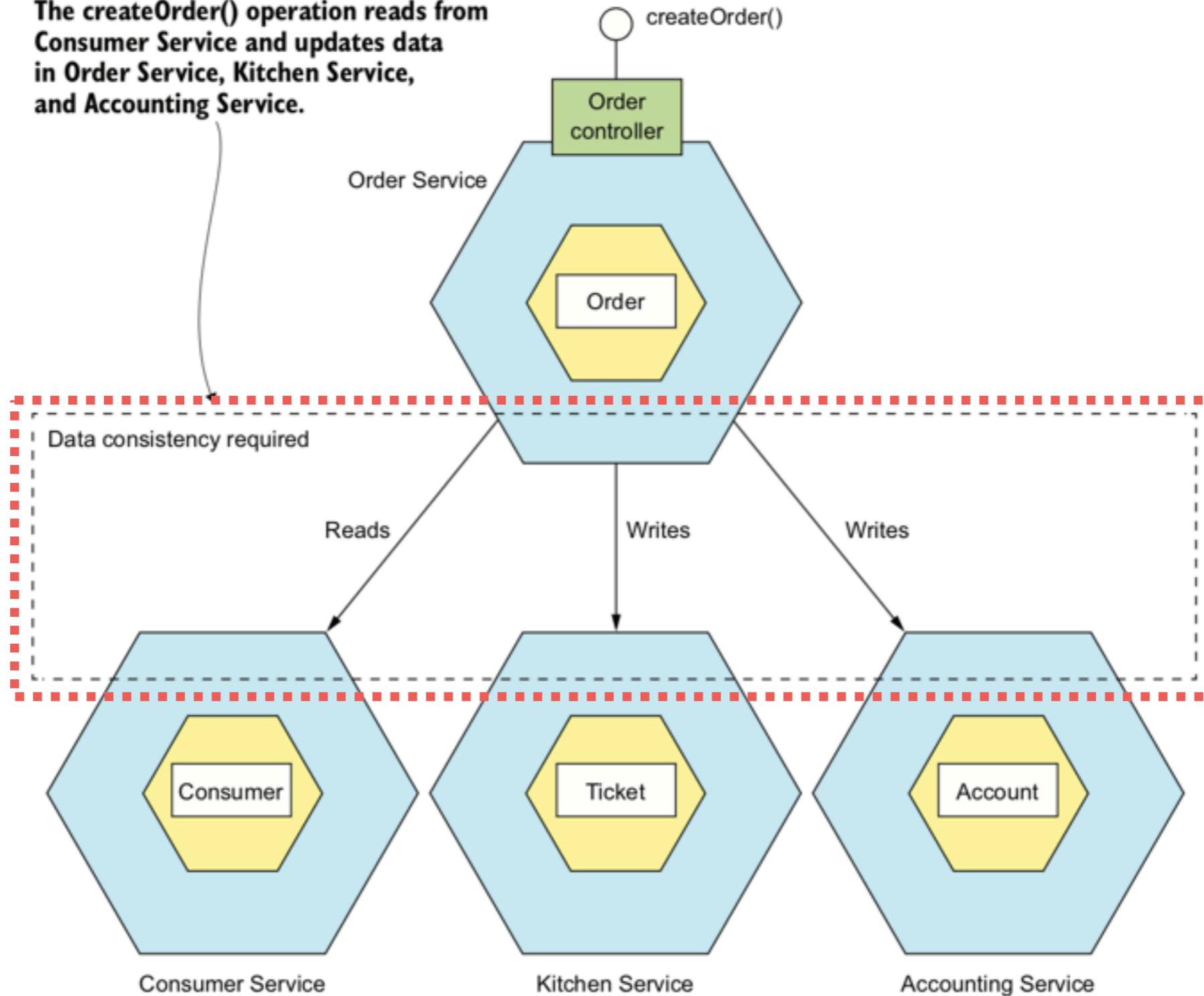


Distributed process failure !!



Problem ?

The **createOrder()** operation reads from Consumer Service and updates data in Order Service, Kitchen Service, and Accounting Service.



Can't use **ACID** transaction !!



A - Atomicity

All or Nothing Transactions

C - Consistency

Guarantees Committed Transaction State

I - Isolation

Transactions are Independent

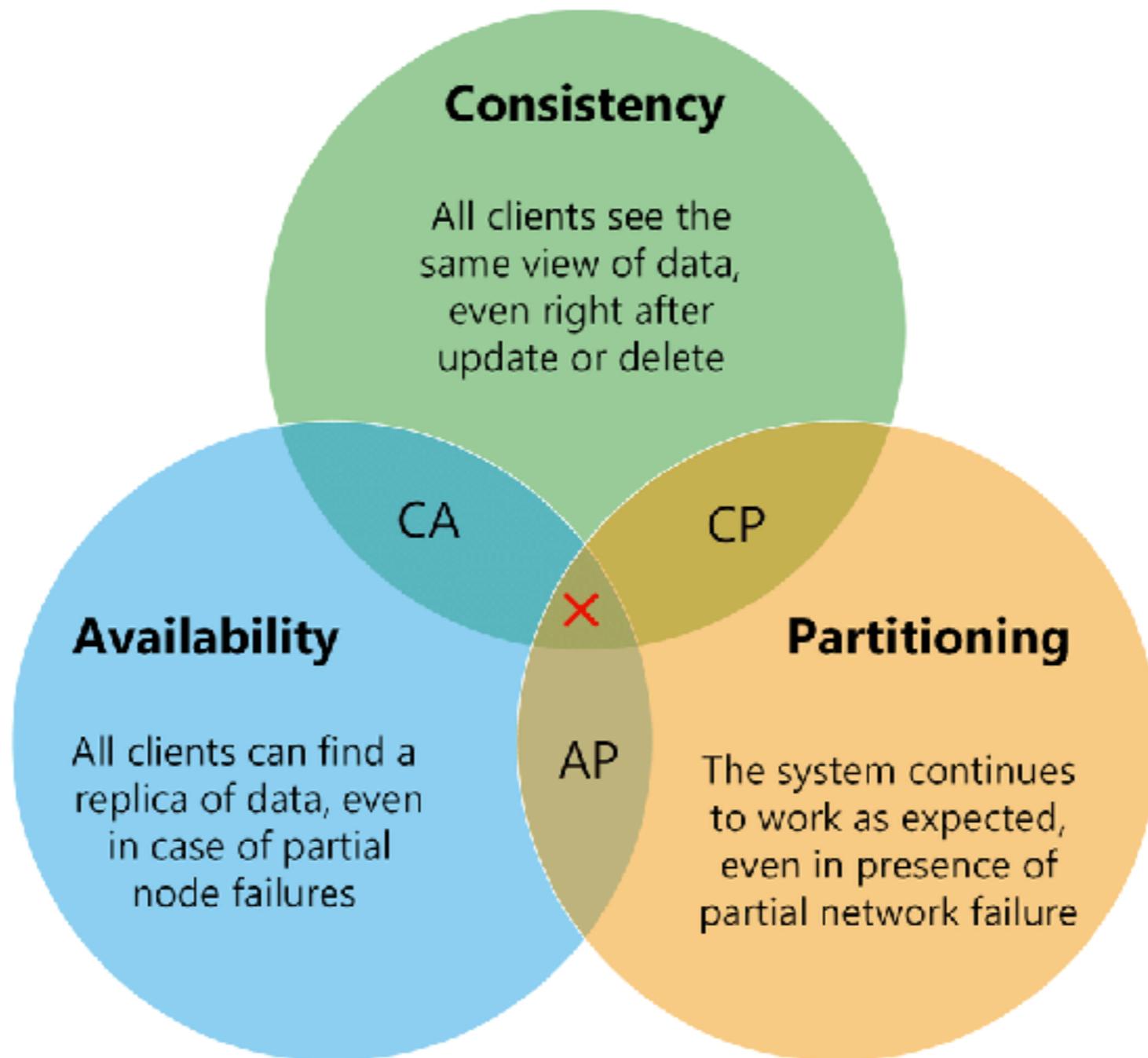
D – Durability

Committed Data is Never Lost

(c) <http://blog.sqlauthority.com>



CAP Theorem



BASE

Basic Available
Soft state
Eventually consistent



ACID vs BASE

ACID	BASE
Scale up (vertical)	Scale out (horizontal)
Strong consistency	Weak/Eventual consistency
Isolation	Last write win, availability first
Transaction	Application transaction
Fix schema	Flexible schema
Expensive joins and relations	Free



Distributed transaction

Common approach is Two-phase commit(2PC)

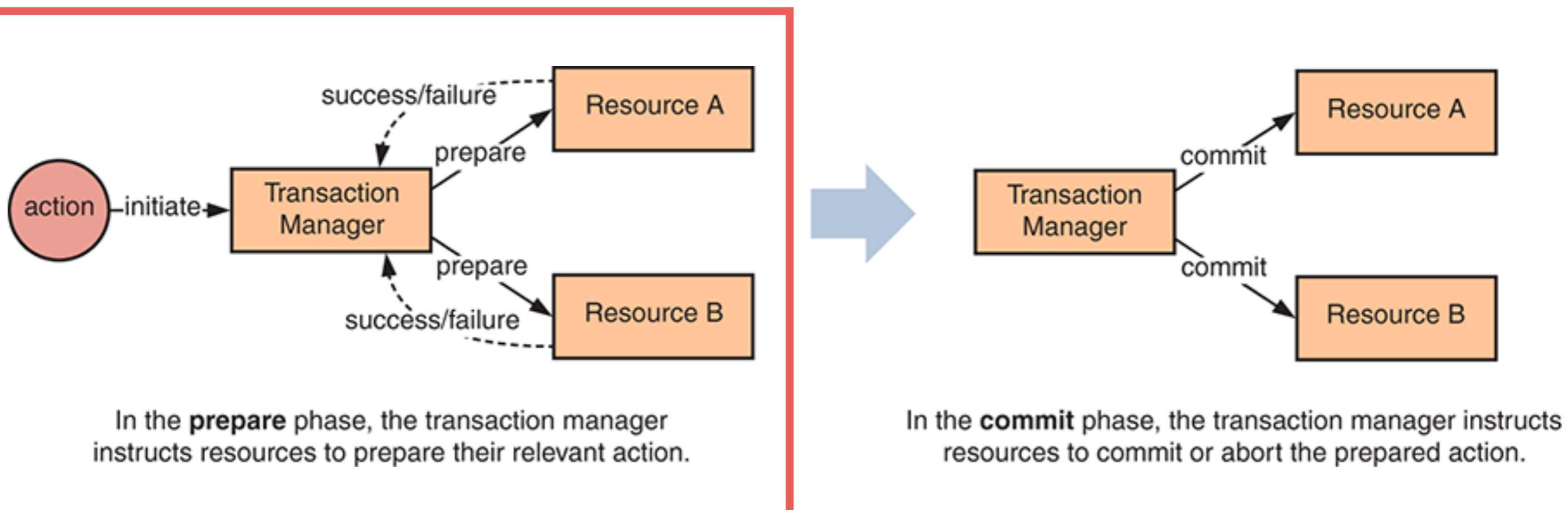
Use transaction manager to split operations across multiple resources in 2 phases

1. *Prepare*
2. *Commit*



Two-phase commit

Phase 1: prepare

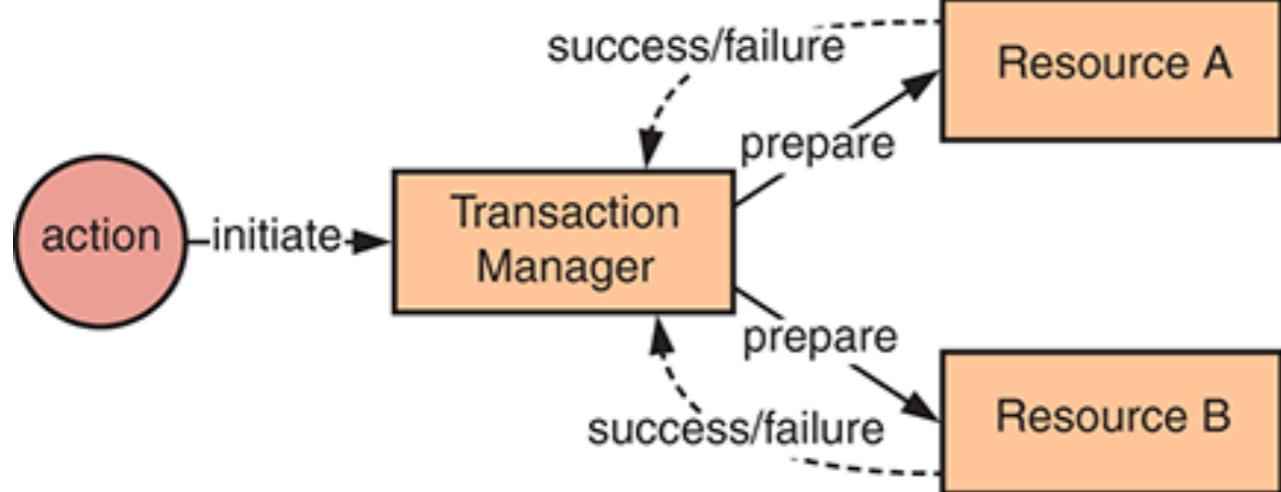


https://en.wikipedia.org/wiki/Two-phase_commit_protocol

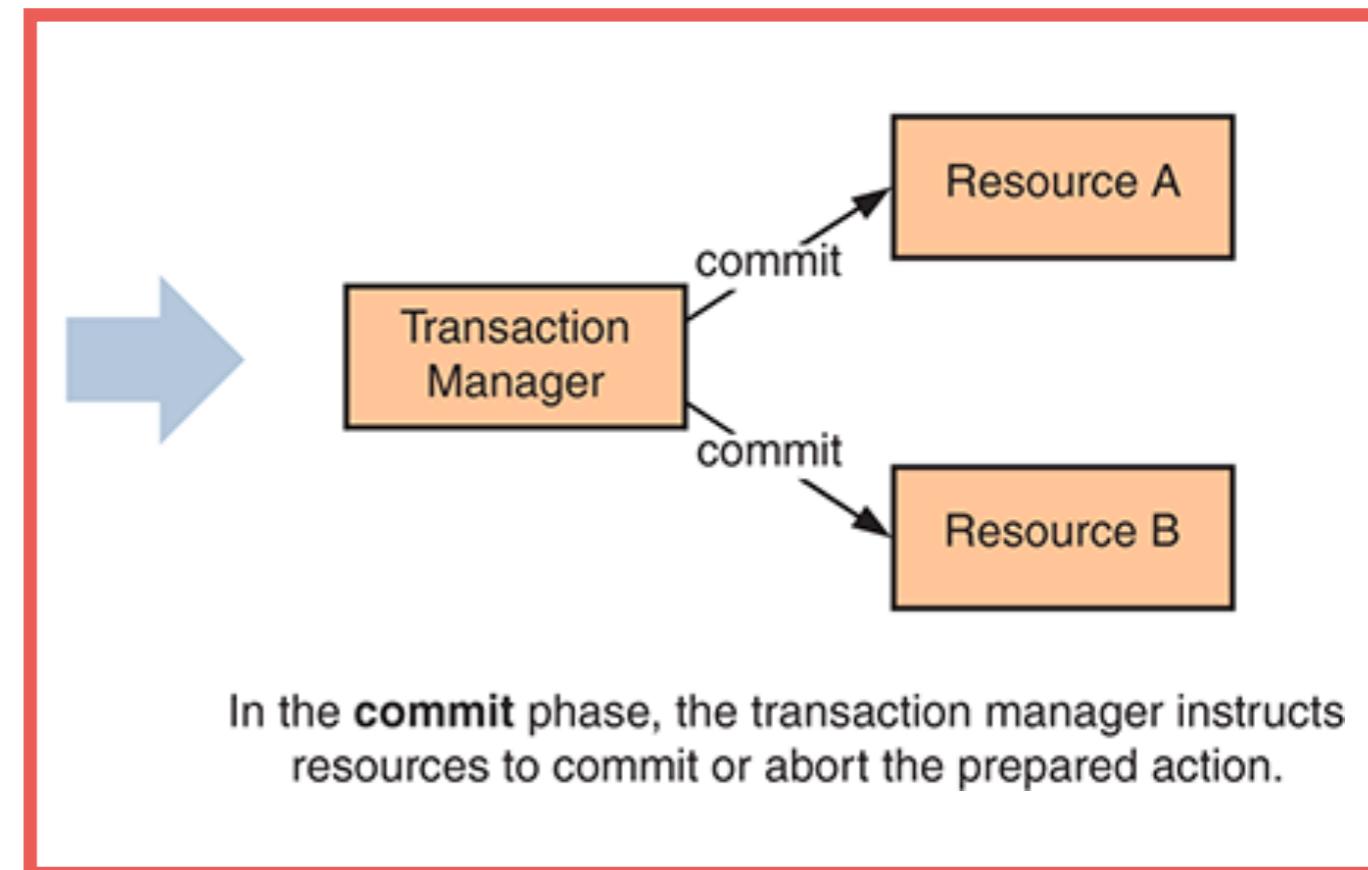


Two-phase commit

Phase 2: commit



In the **prepare** phase, the transaction manager instructs resources to prepare their relevant action.



In the **commit** phase, the transaction manager instructs resources to commit or abort the prepared action.

https://en.wikipedia.org/wiki/Two-phase_commit_protocol



**Distributed transaction places
a lock on resources under
transaction to ensure isolation**



Inappropriate for long-running operations



Increase risk of contention and deadlock

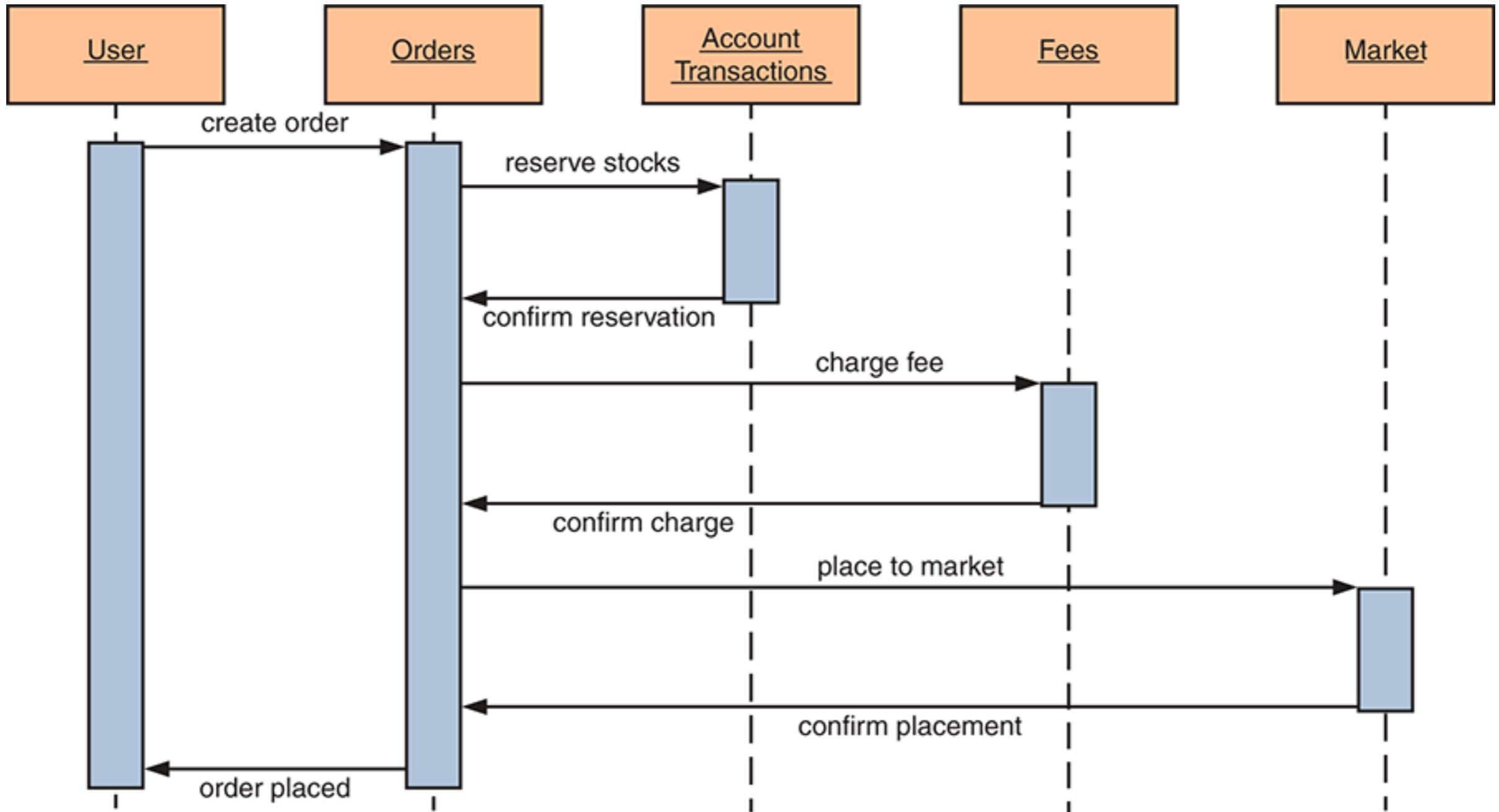


We need ...

Don't need distributed transaction
Reduce complexity of code
Reliable



Synchronous process

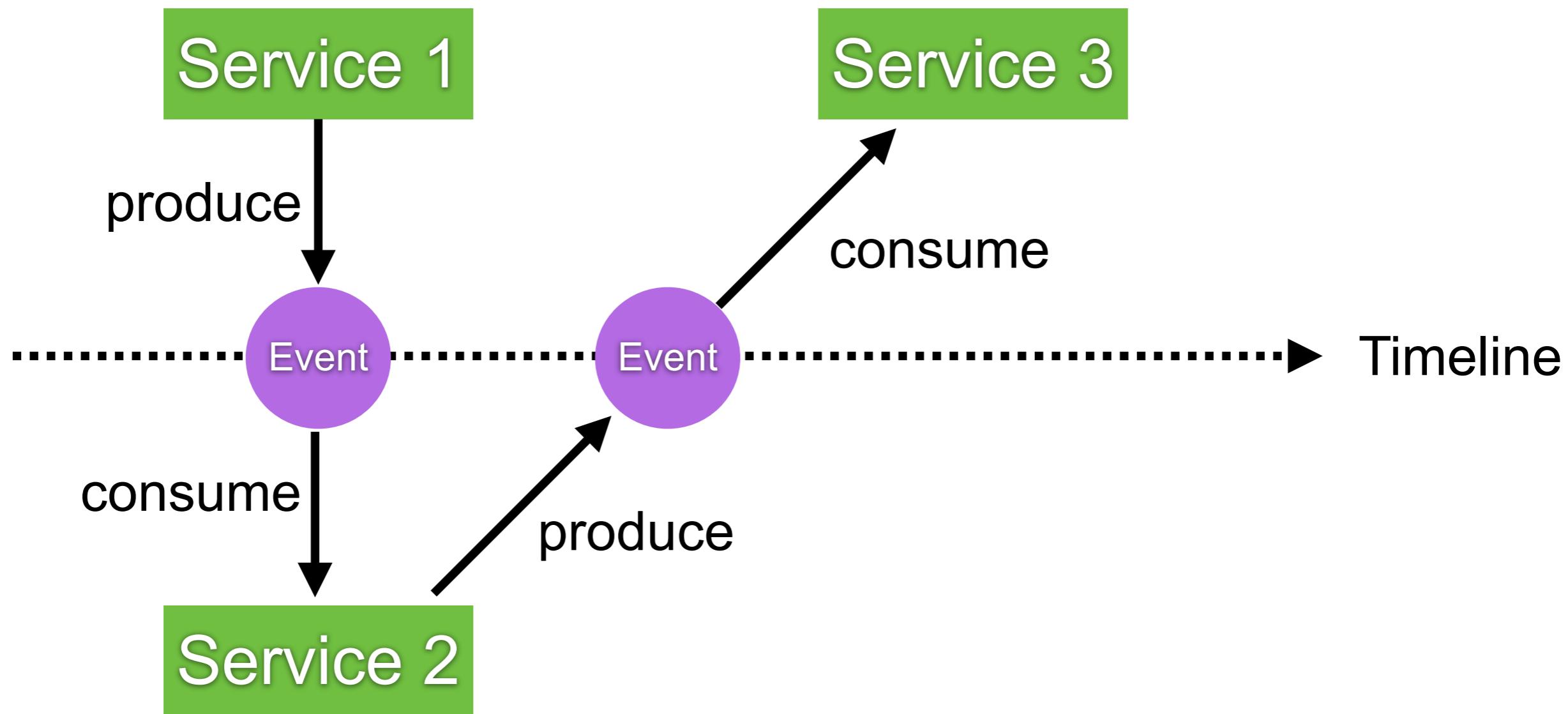


Event-based communication



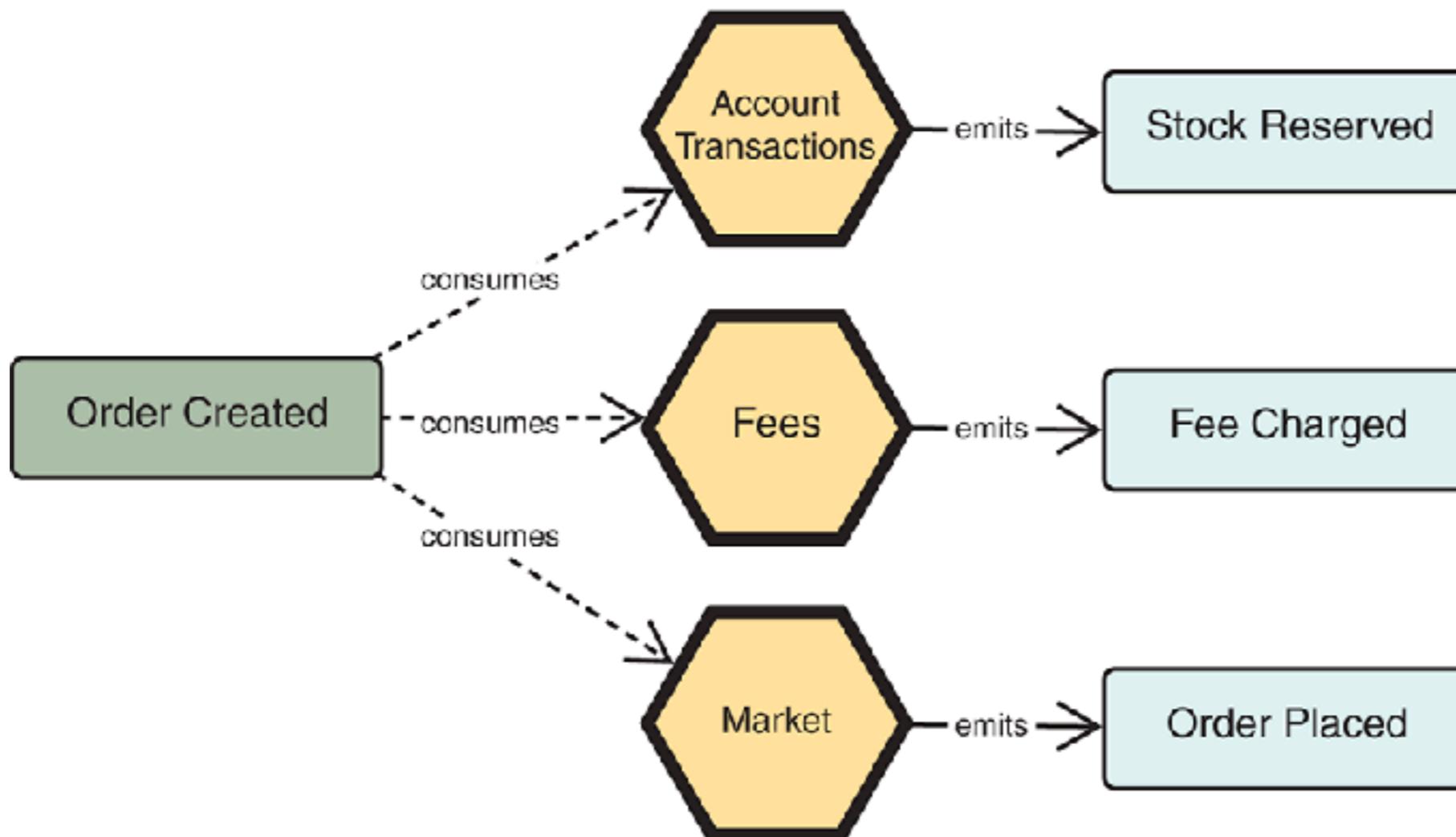
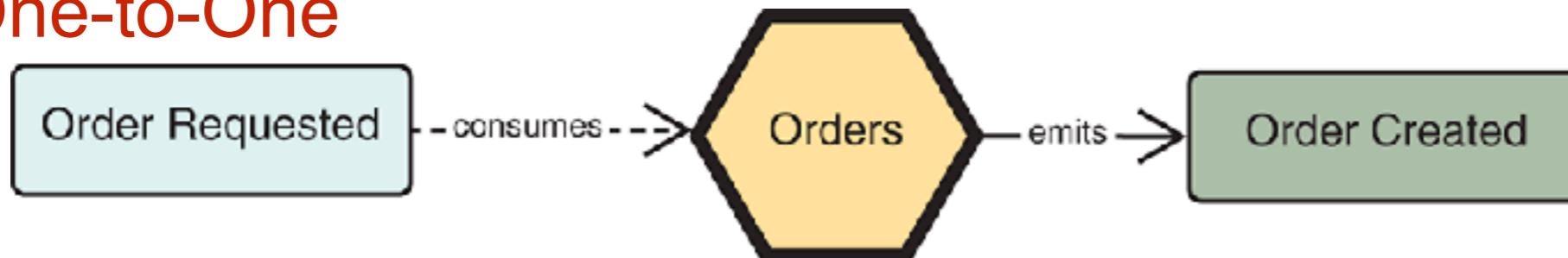
Choreography pattern

Sequence of Tx and emit **event** or **message** that trigger the next process in Tx

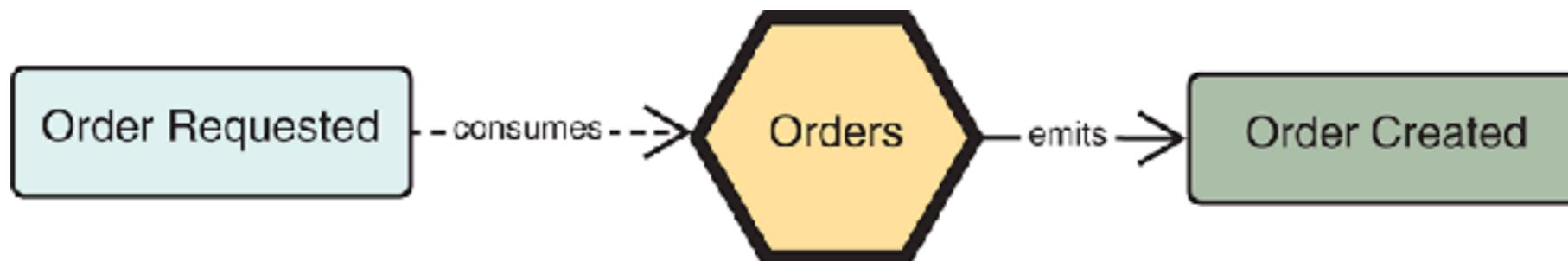


Service consume and emit event

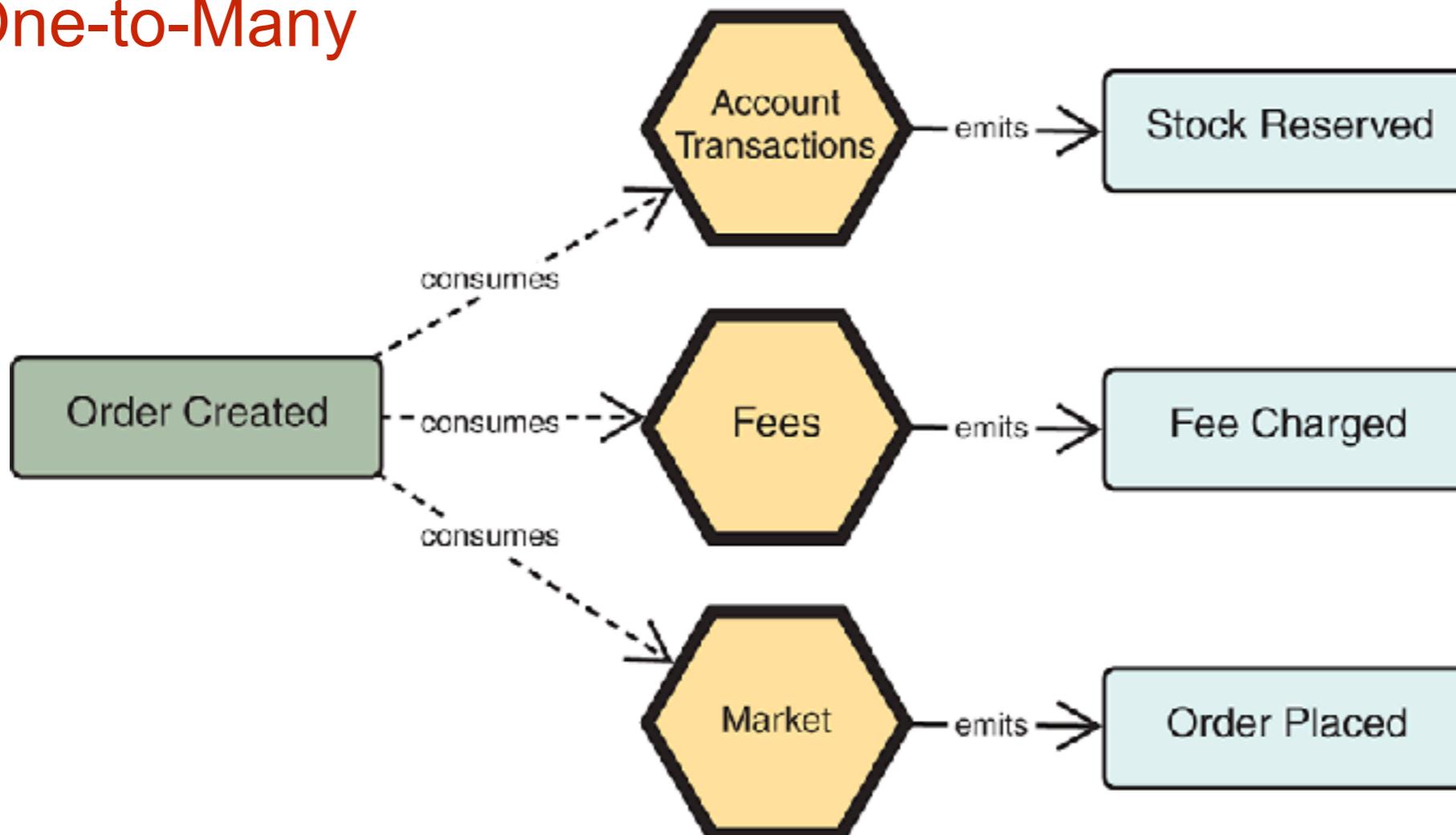
One-to-One



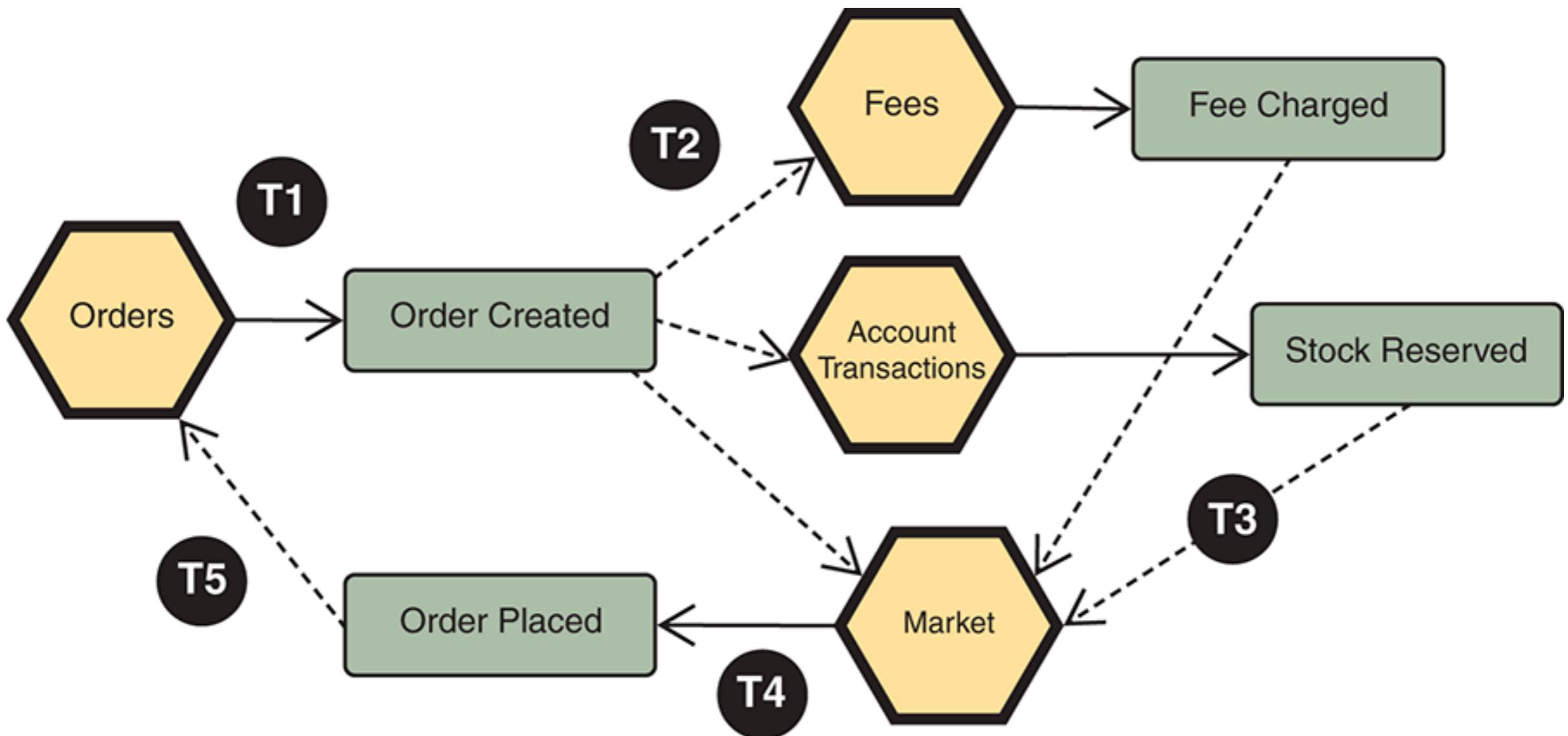
Service consume and emit event



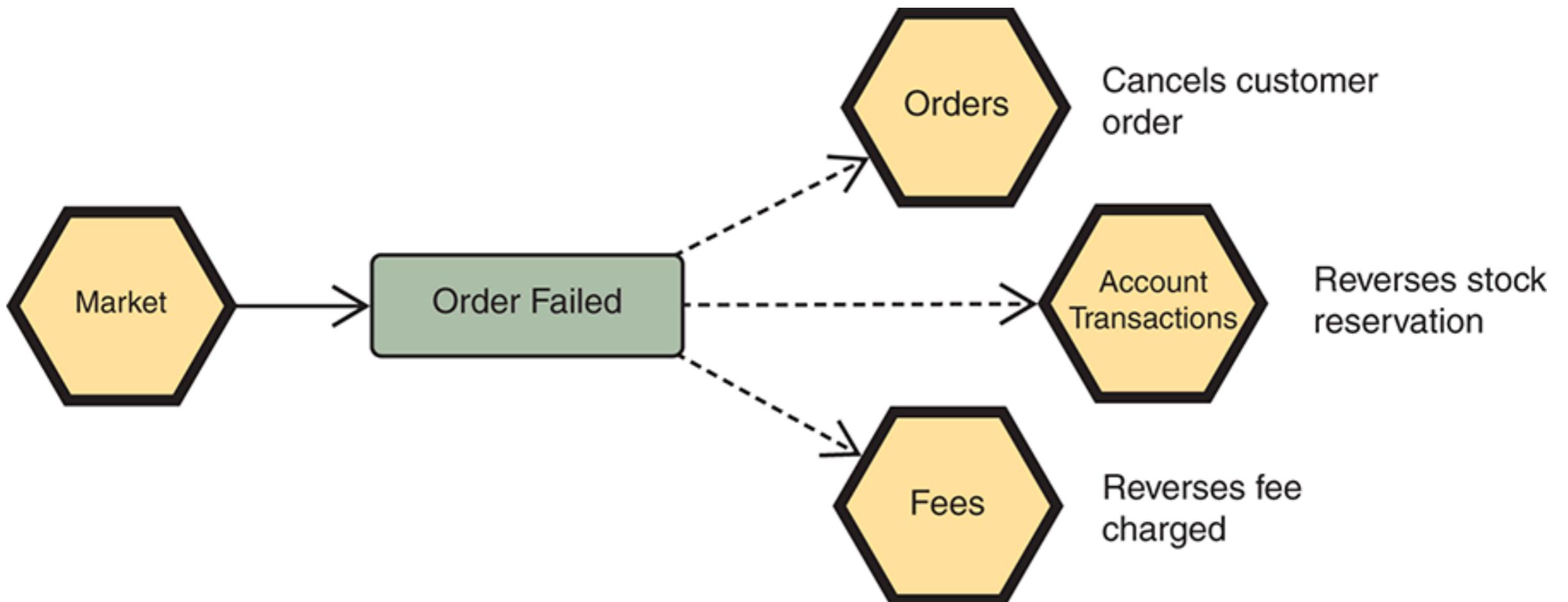
One-to-Many



Example (Success)

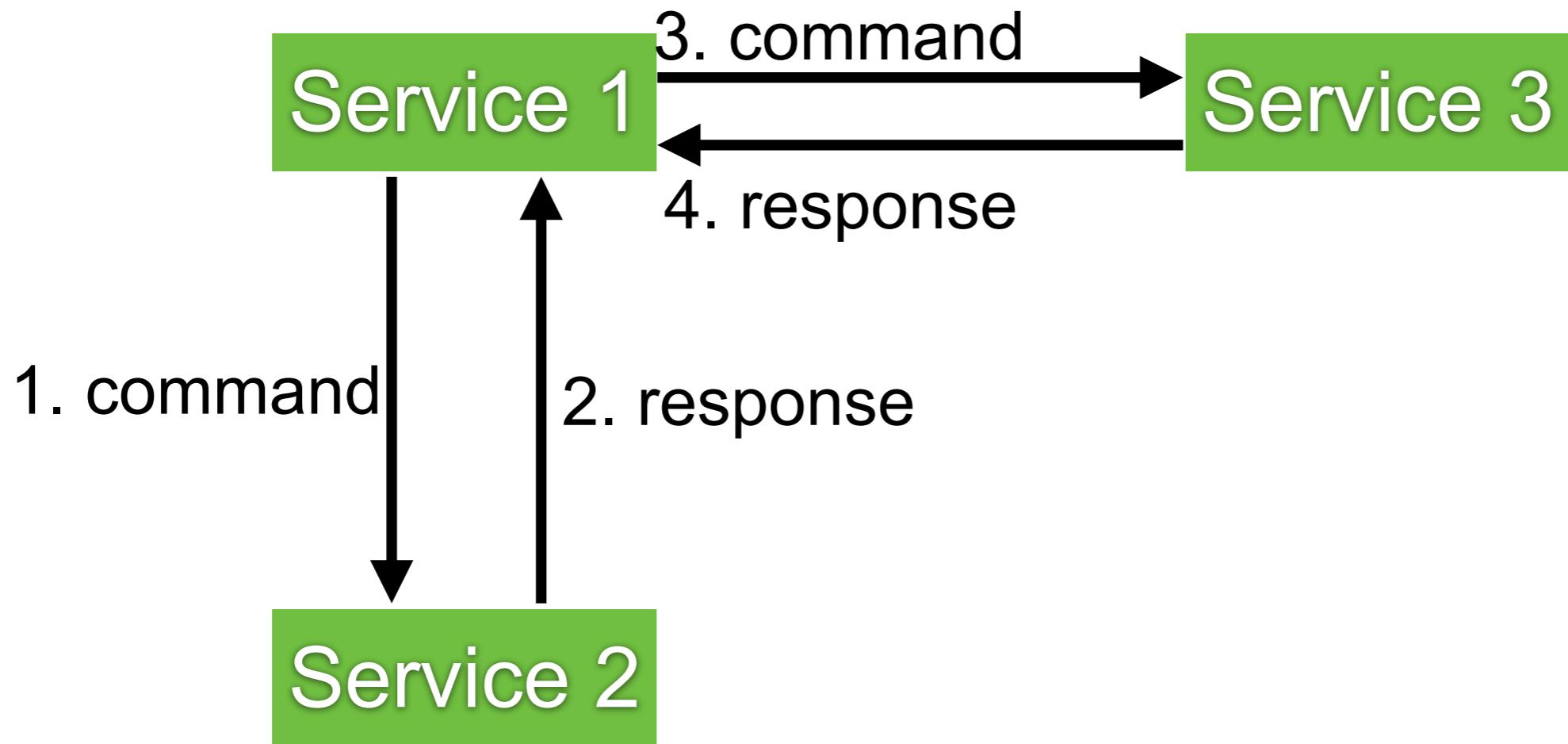


Example (Fail)

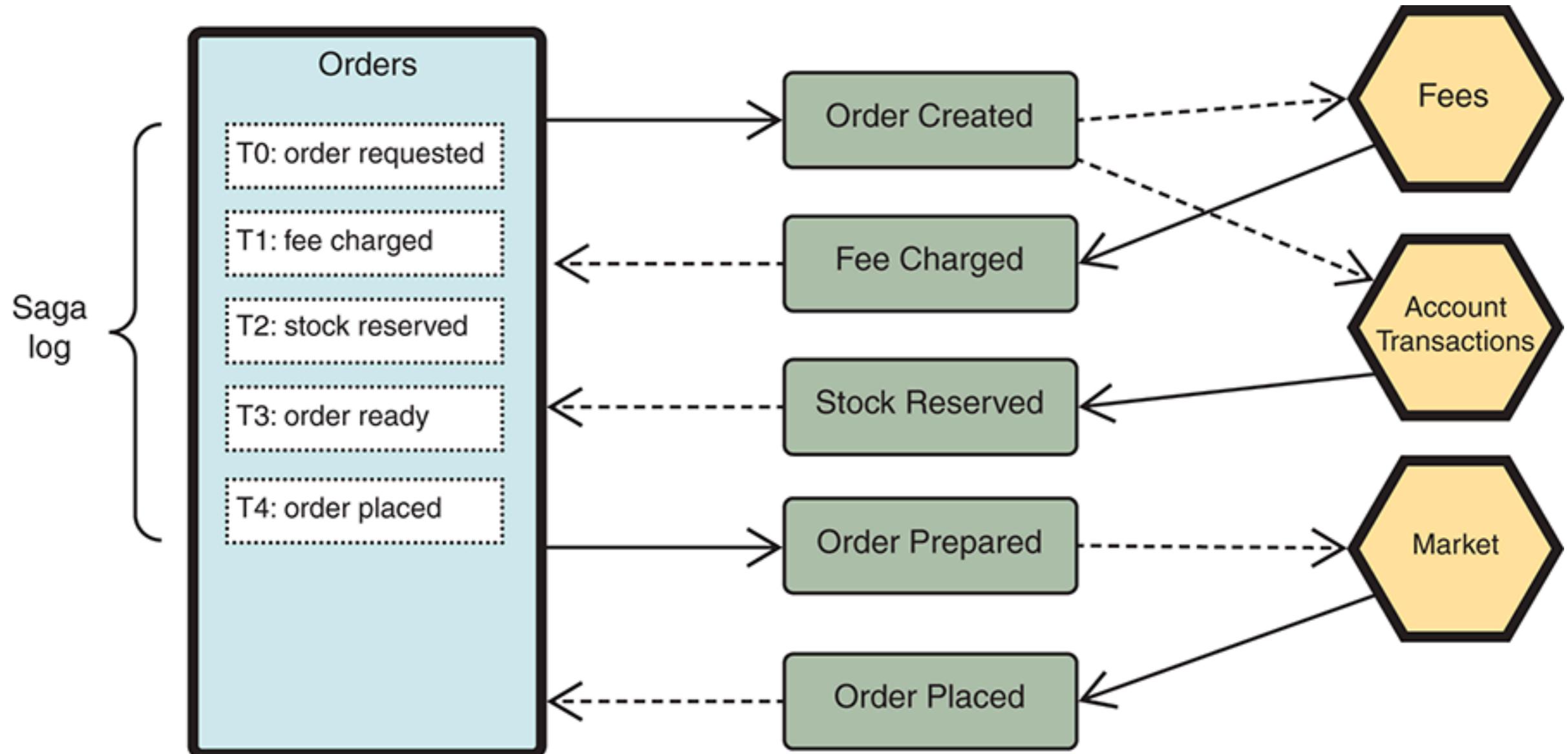


Orchestrate pattern

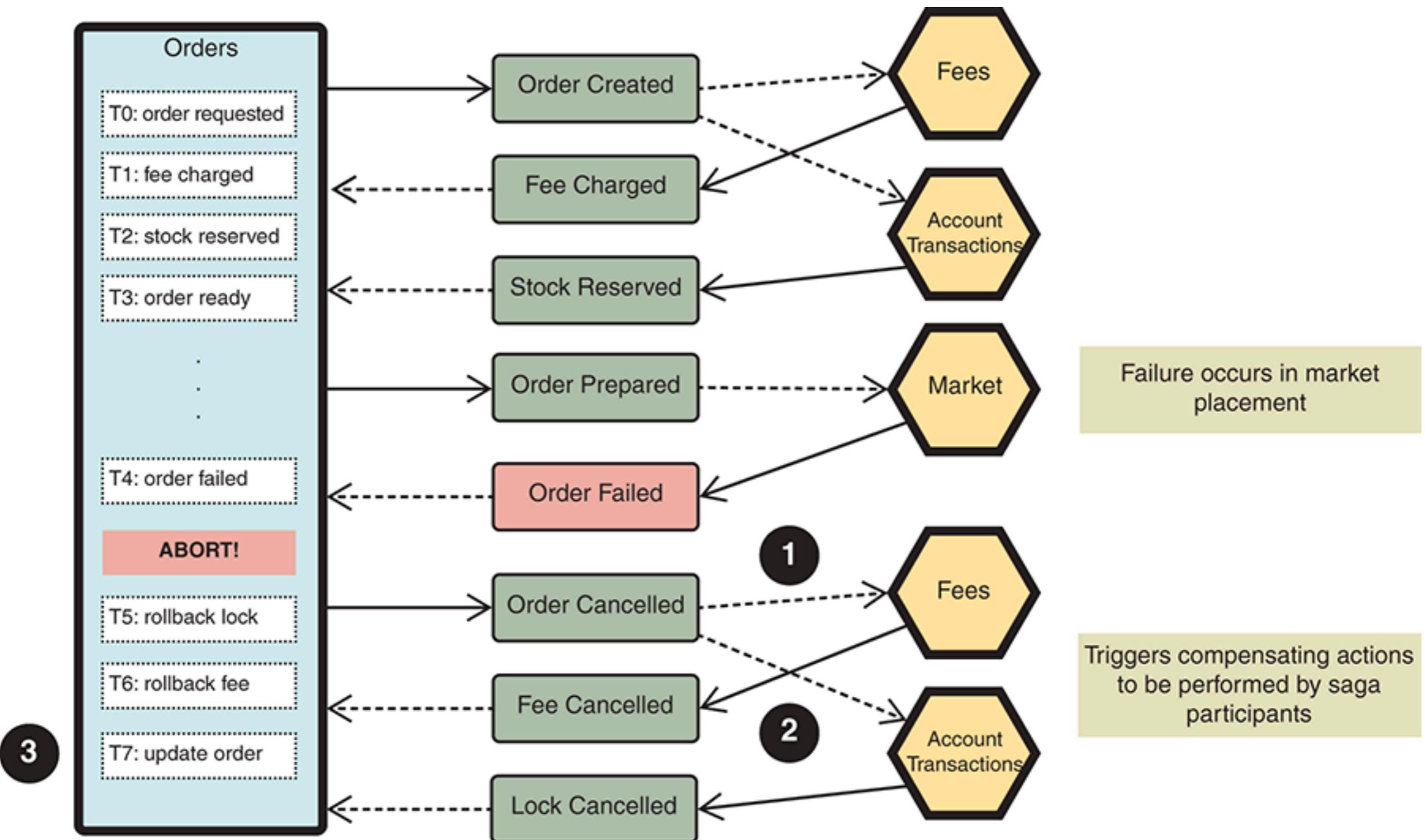
Sequence of Tx and emit **event** or **message** that trigger the next process in Tx



Example (Success)



Example (Fail)



Consistency patterns

Name	Strategy
Compensating action	Perform action that undo prior action(s)
Retry	Retry until success or timeout
Ignore	Do nothing in the event of errors
Restart	Reset to the original state and start again
Tentative operation	Perform a tentative operation and confirm (or cancel) later



“Saga”



Message vs Event

Message is addressed to someone

Event is something that happen and someone can react to that



Event sourcing

Maintain source of truth of business
Immutable sequence of events

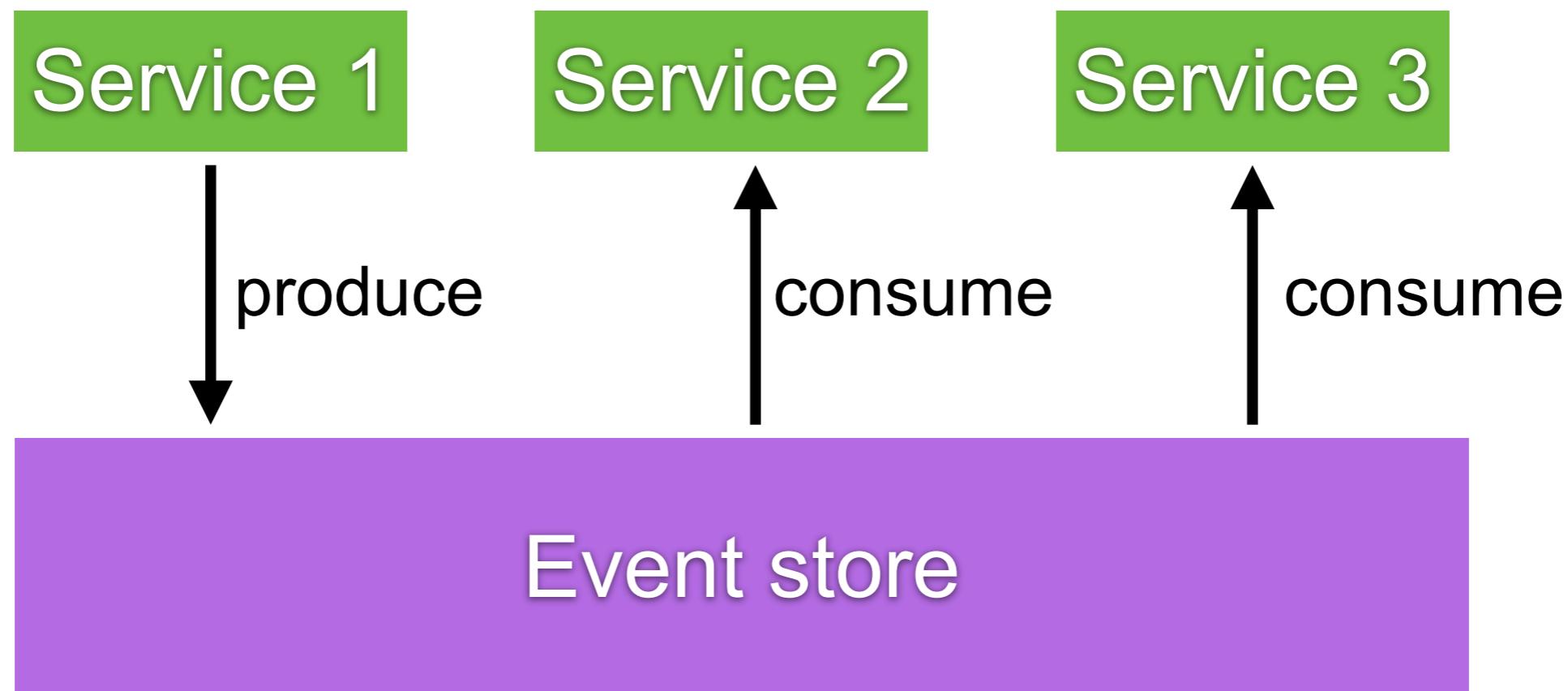


Sequence of event is keep in **Event store**



Event store

Keep events in order
Broadcast new event

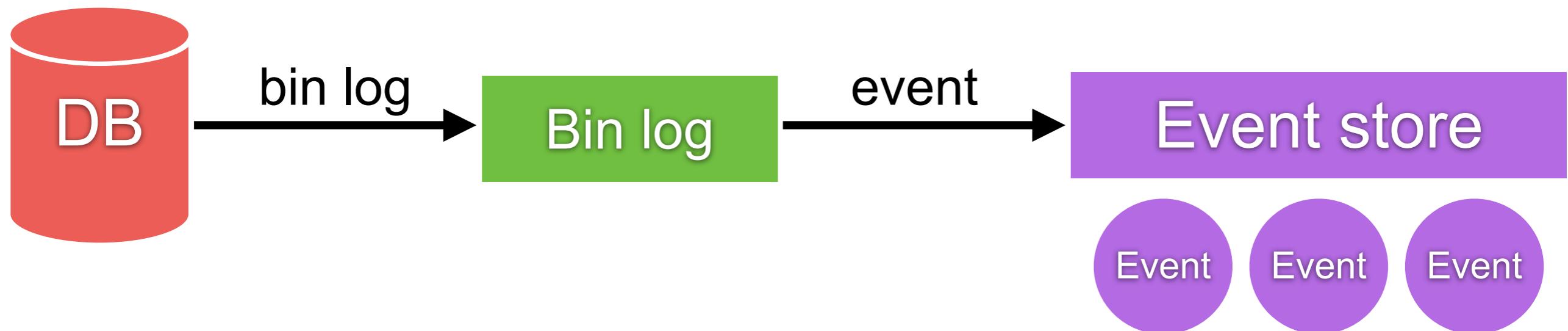


**With Event sourcing, we can
decouple services (query and
command)**



Event sourcing with database

Working with database



Binlog or Binary log event is information about data modification made to database



How to implement queries data ?



Query data from multiple services

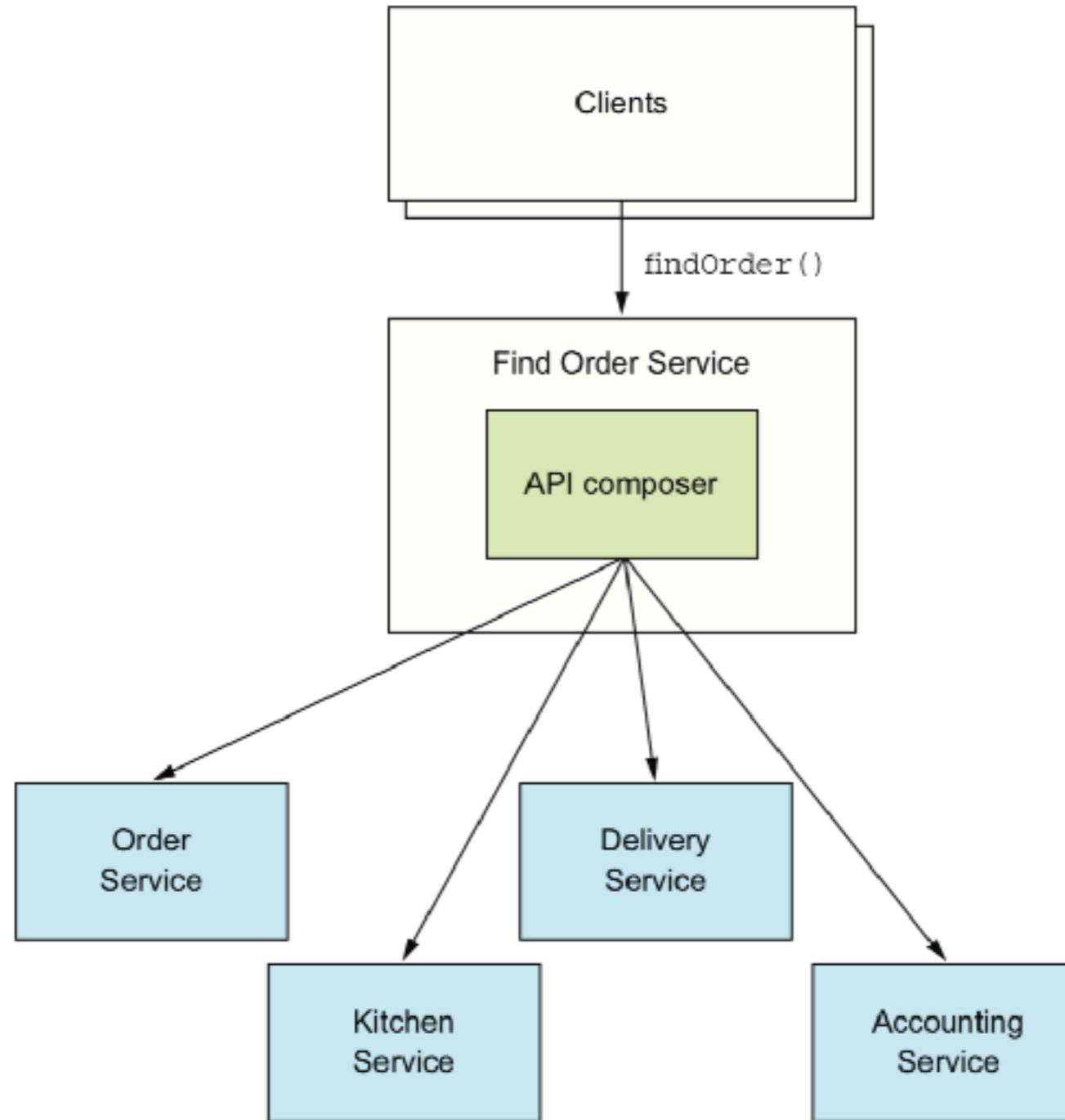
API composition

Cold data from services

CQRS (Command Query Responsibility Segregation)



API composition pattern



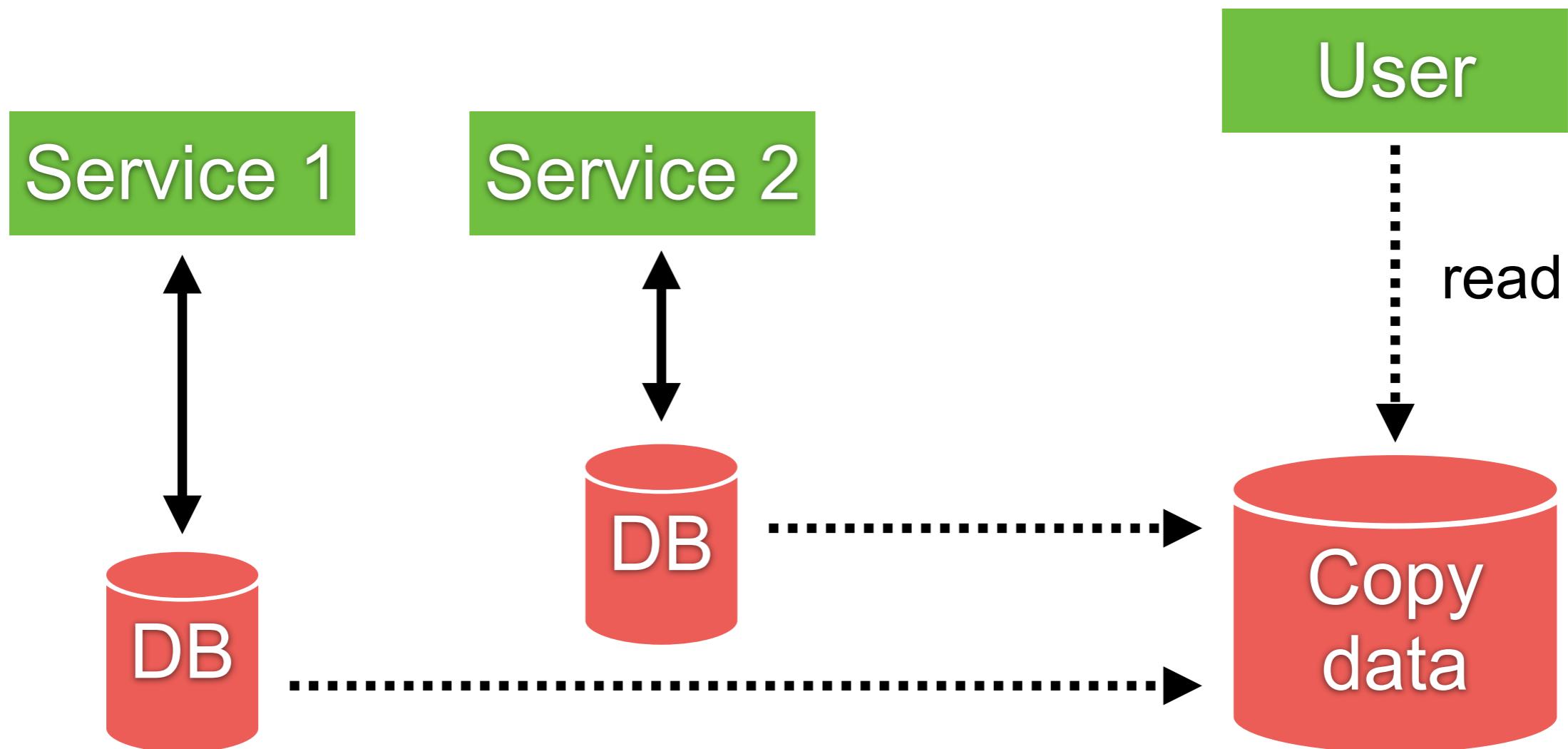
Drawbacks

Increase overhead
Lack of transactional data consistency
Reduce availability



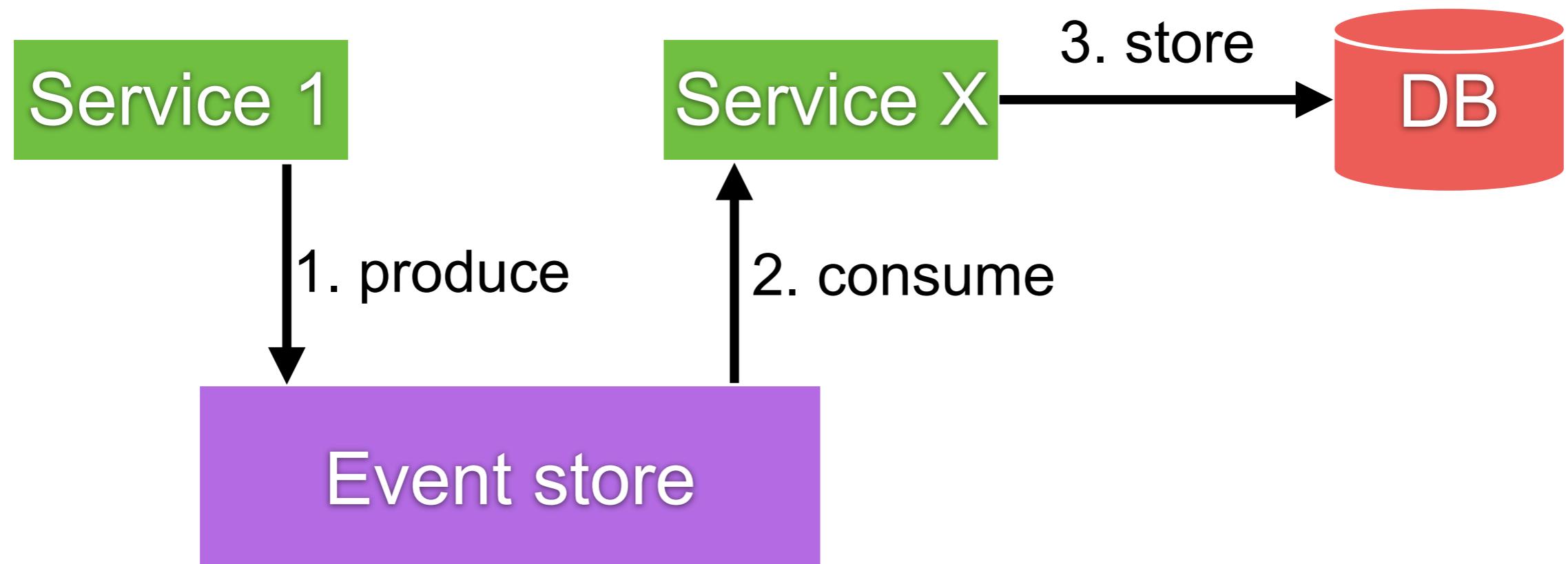
Storing copy data

Copy or caching data to other database



Working with Event-based

Publish event to store copy data

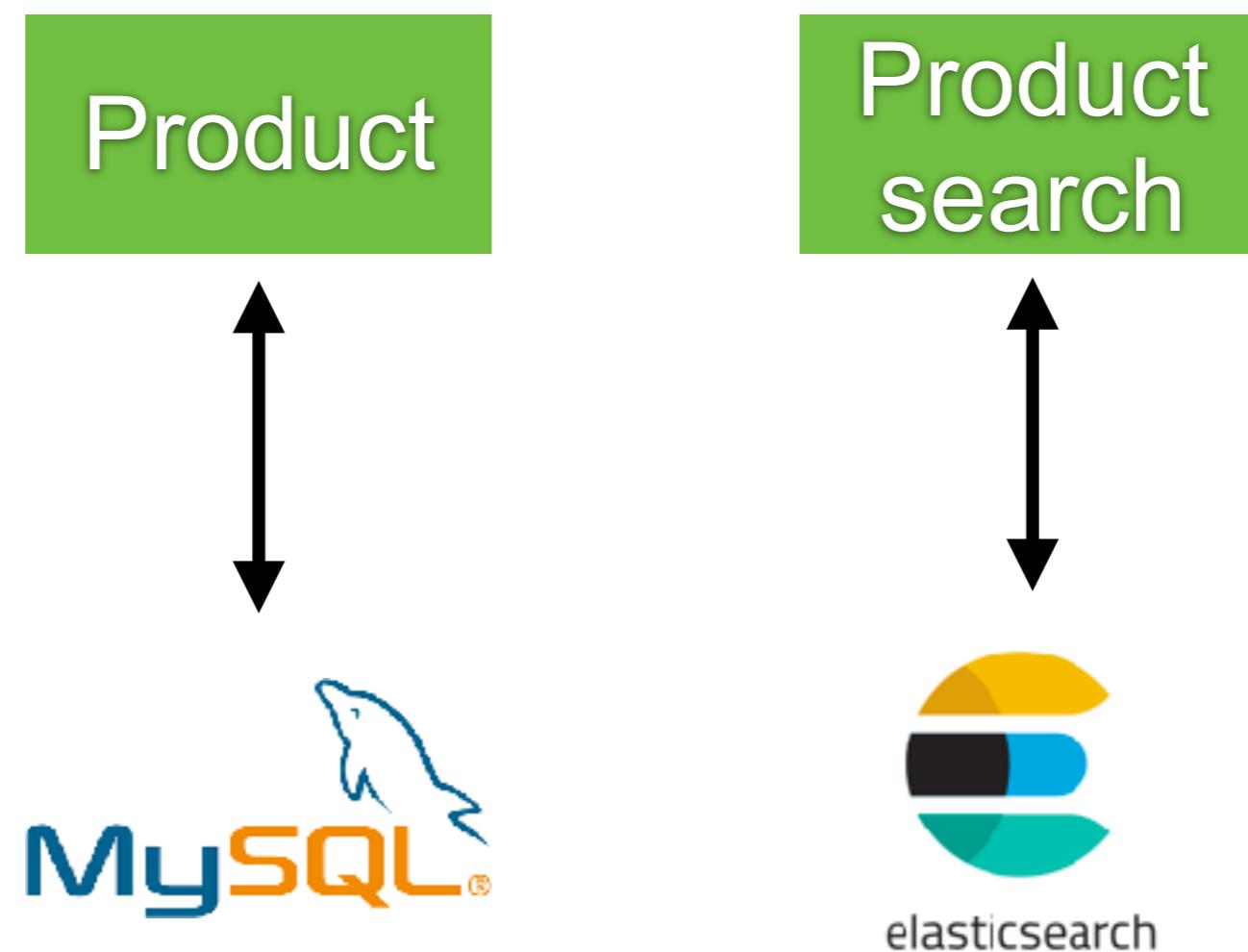


Separate query and command



Separate data for read and write

For example MySQL to write, Elasticsearch to search

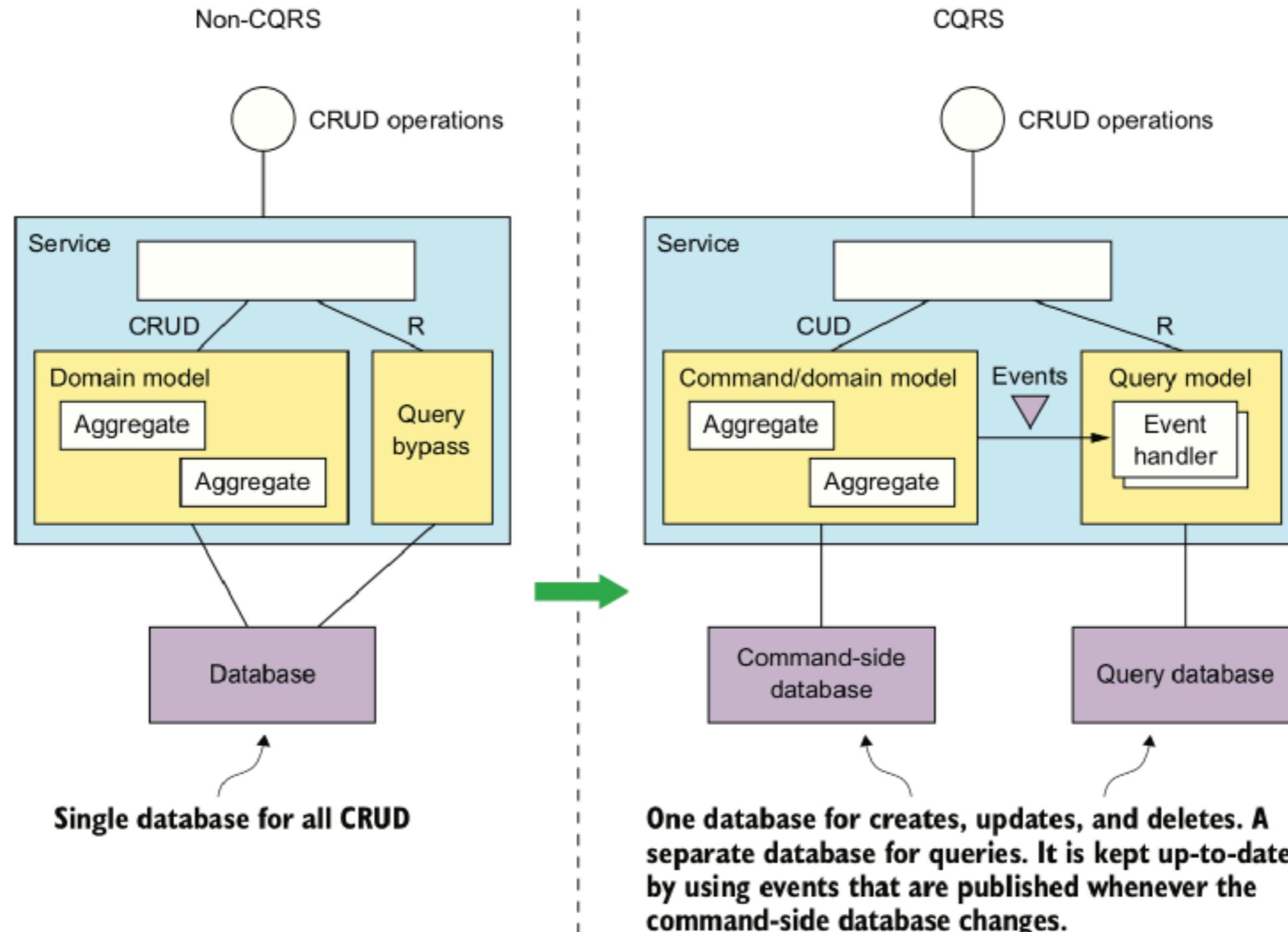


CQRS

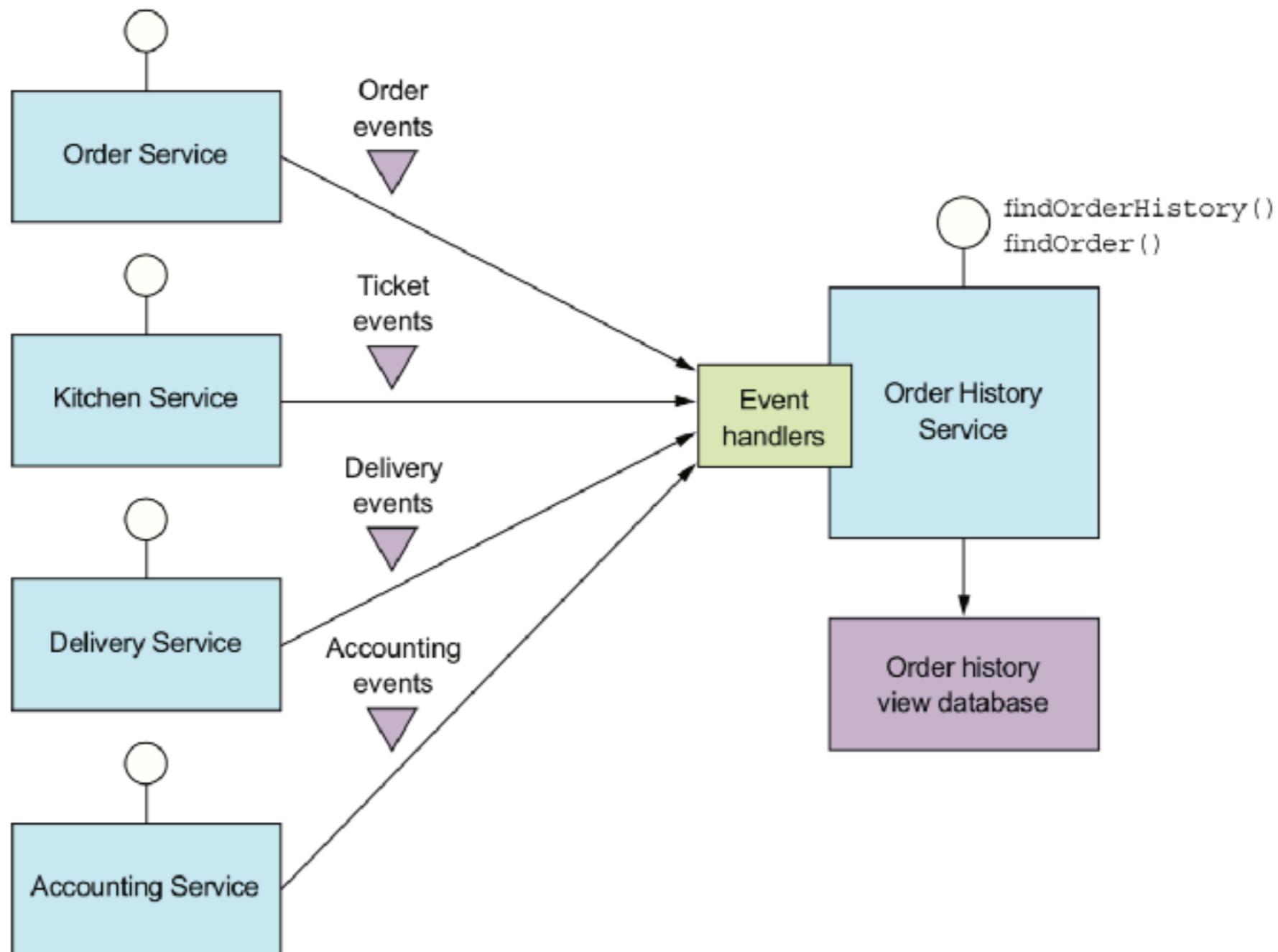
Command Query Responsibility Segregation



CQRS = Separate command from query



CQRS = Query-side service



Benefits

Improve separation of concern
Efficiency query



Drawbacks

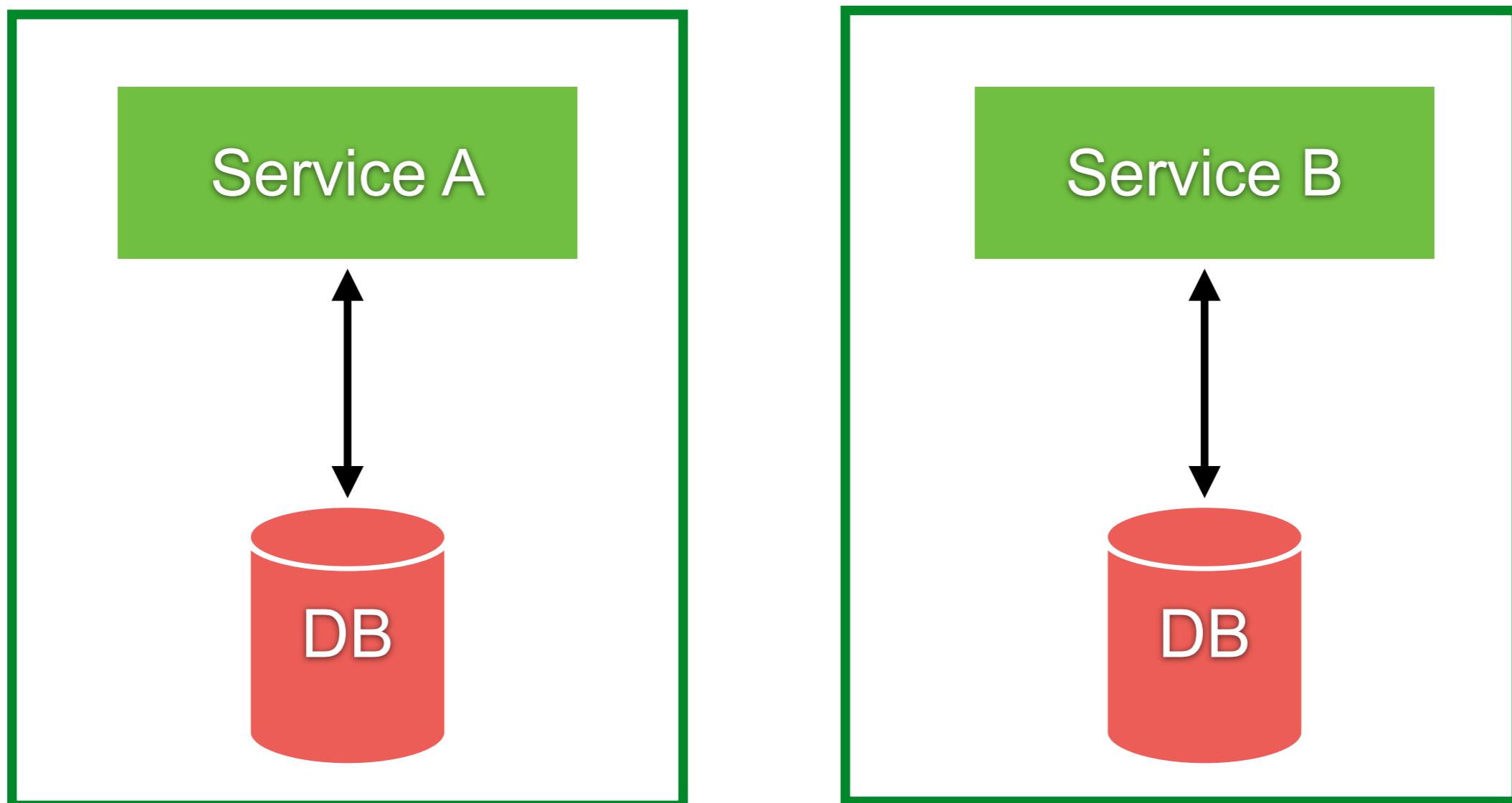
More complex

Lag between **command** and **query** side

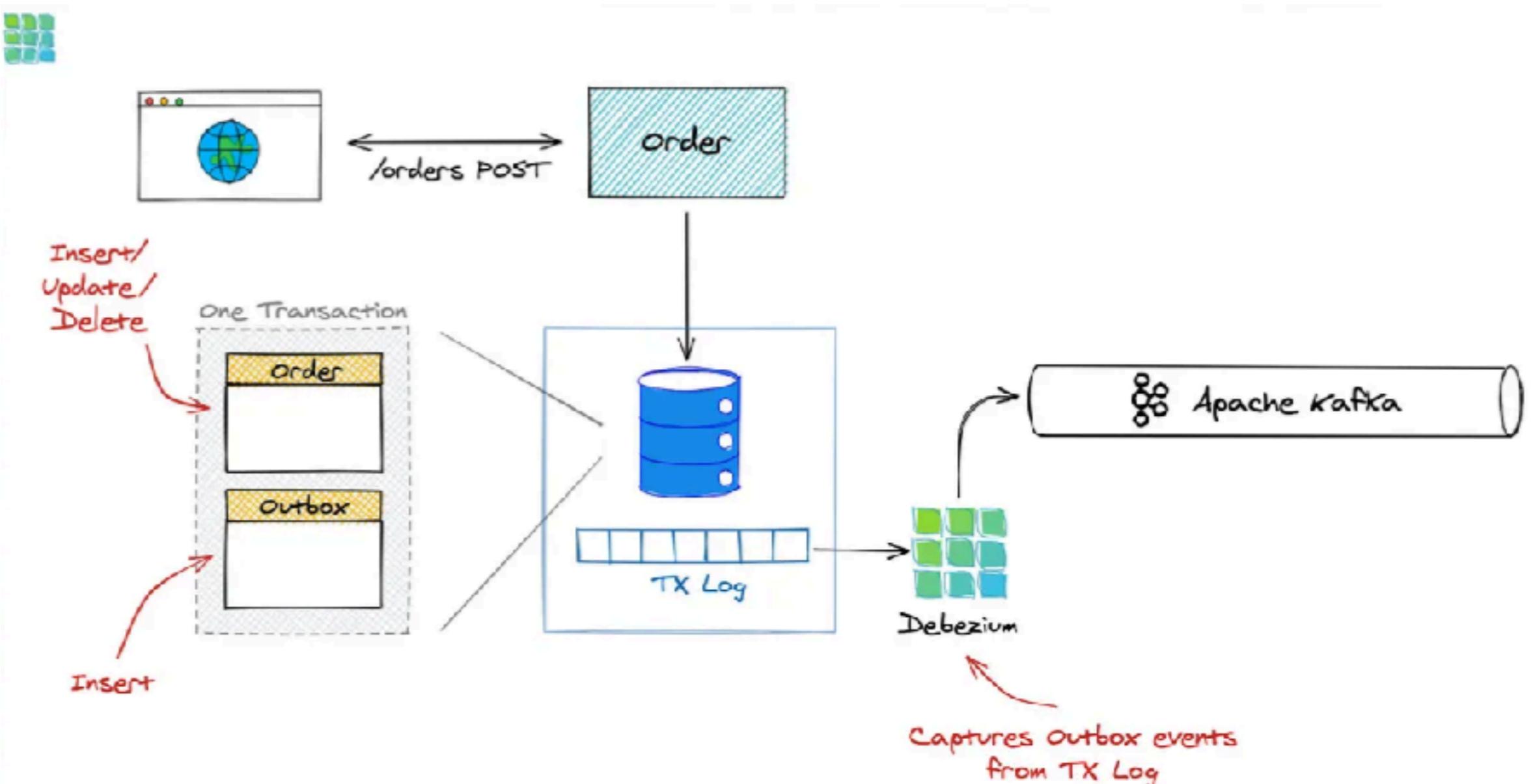


Outbox pattern

Decouple services using Pub/Sub
to exchange data between services



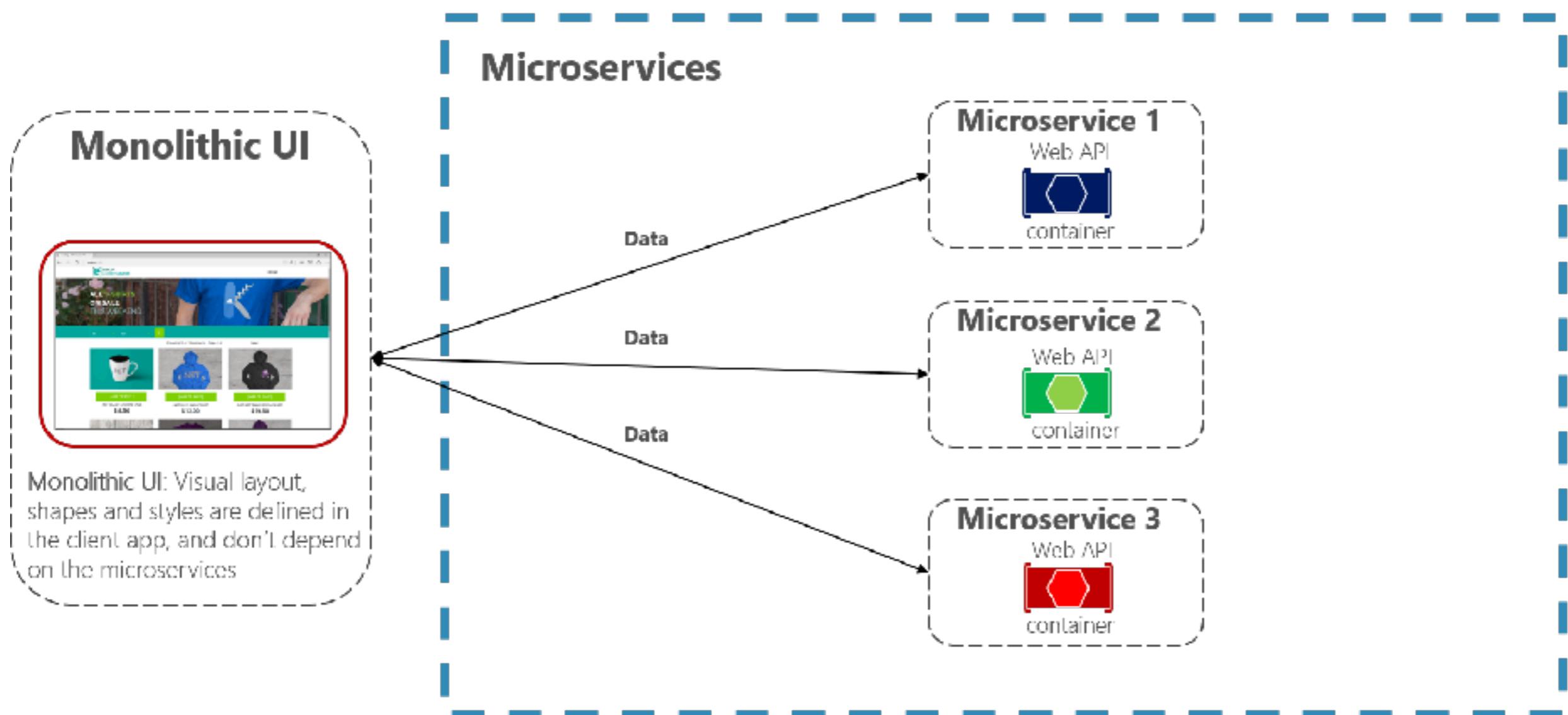
Example of Outbox pattern



Integrate services with User Interface



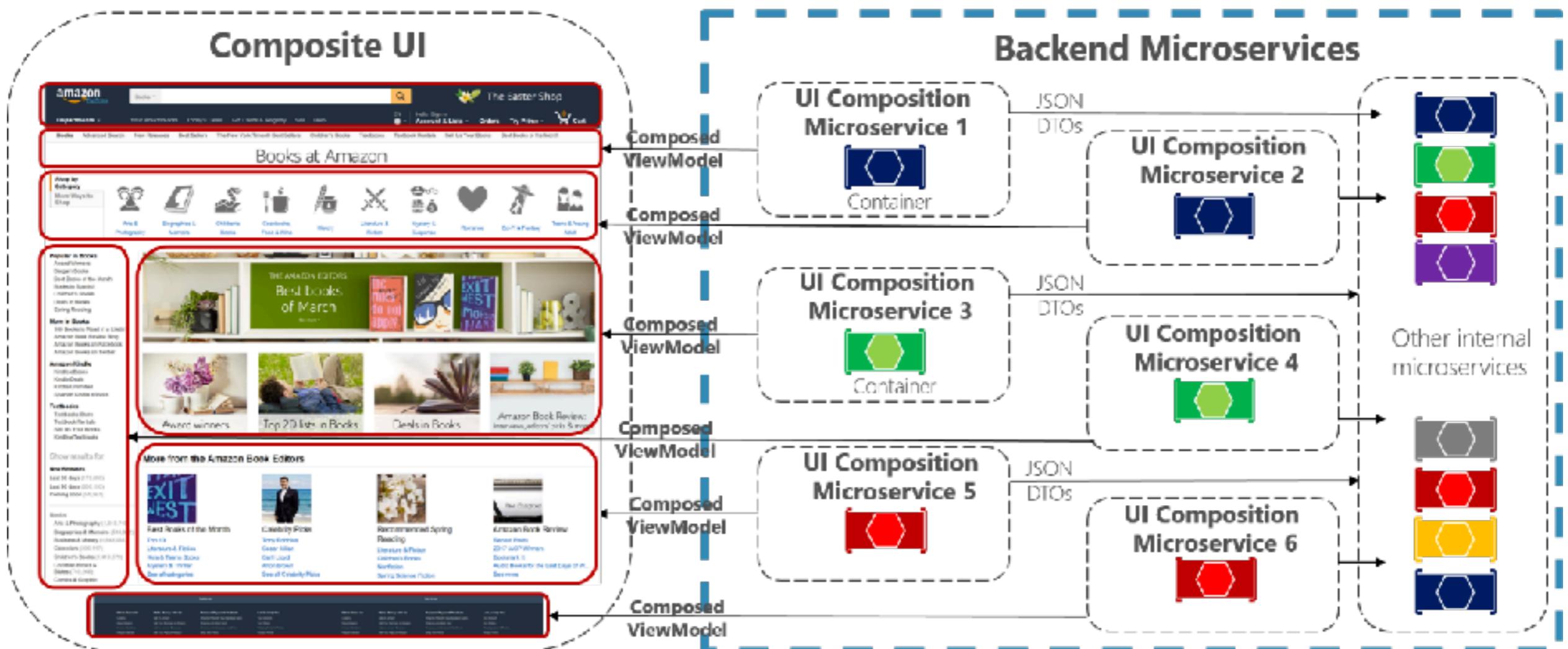
Monolithic UI consuming microservices



<https://docs.microsoft.com>



Composite UI generated by microservices



<https://docs.microsoft.com>

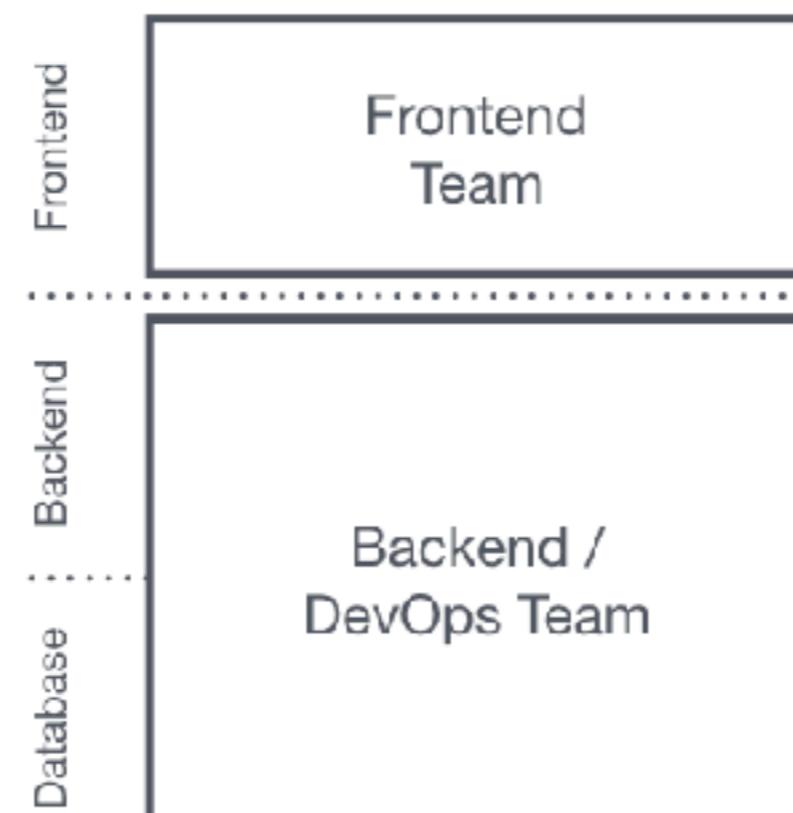


Monolith frontend

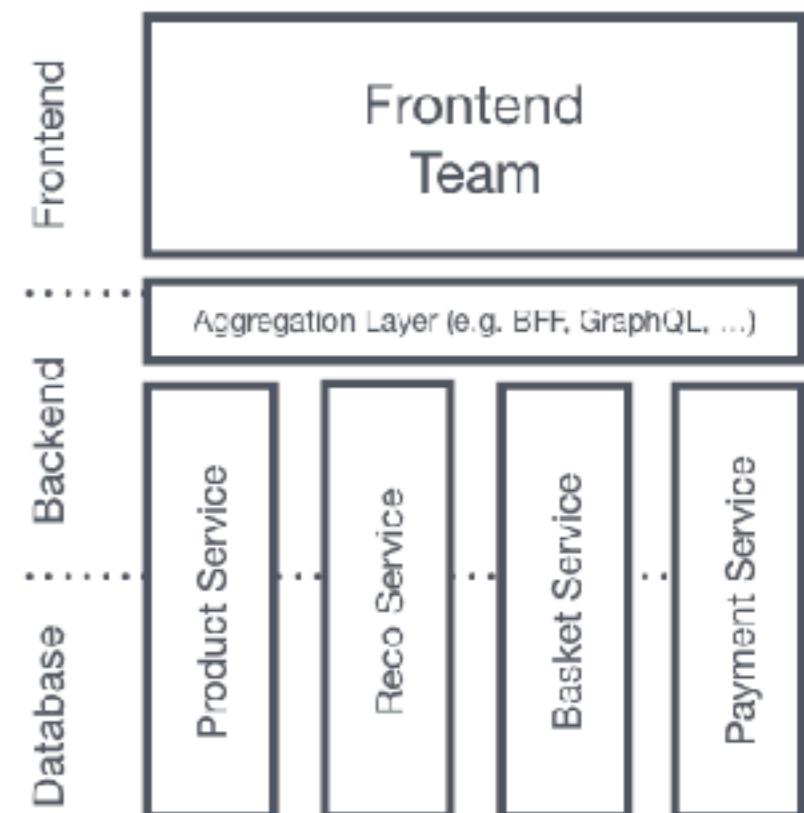
The Monolith



Front & Back



Microservices

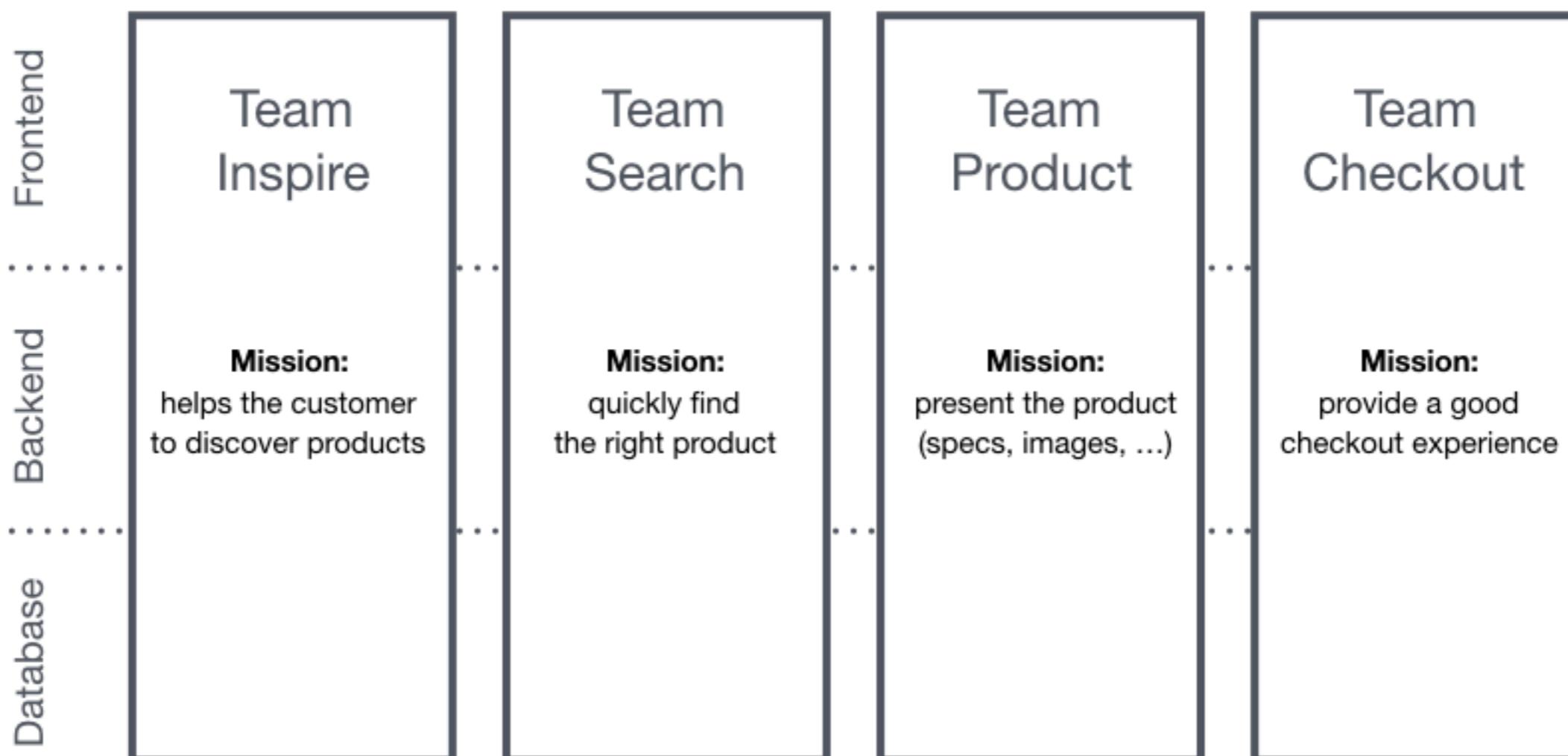


<https://micro-frontends.org/>



Micro frontend

End-to-End Teams with Micro Frontends



<https://micro-frontends.org/>



Observability of services

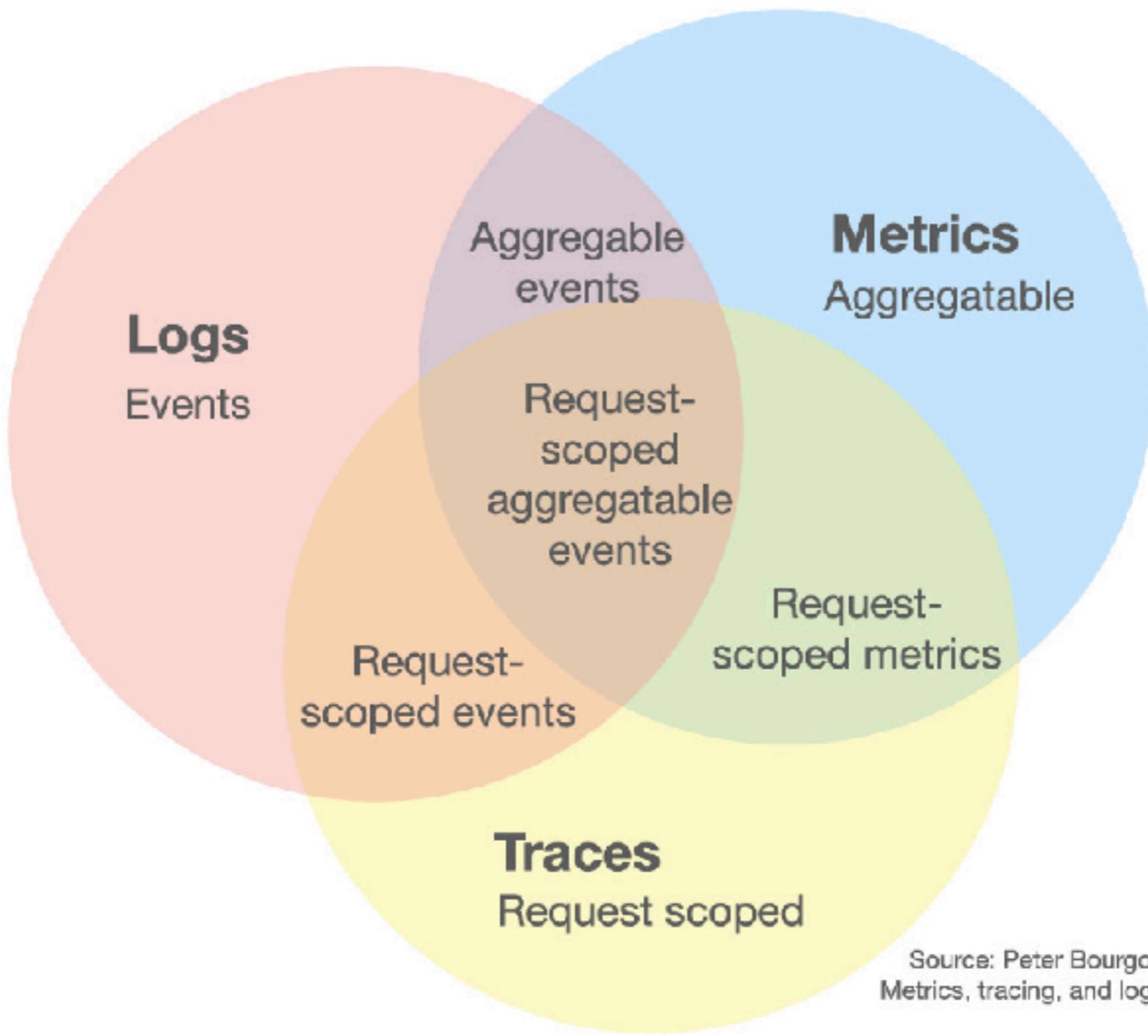


Observability of services

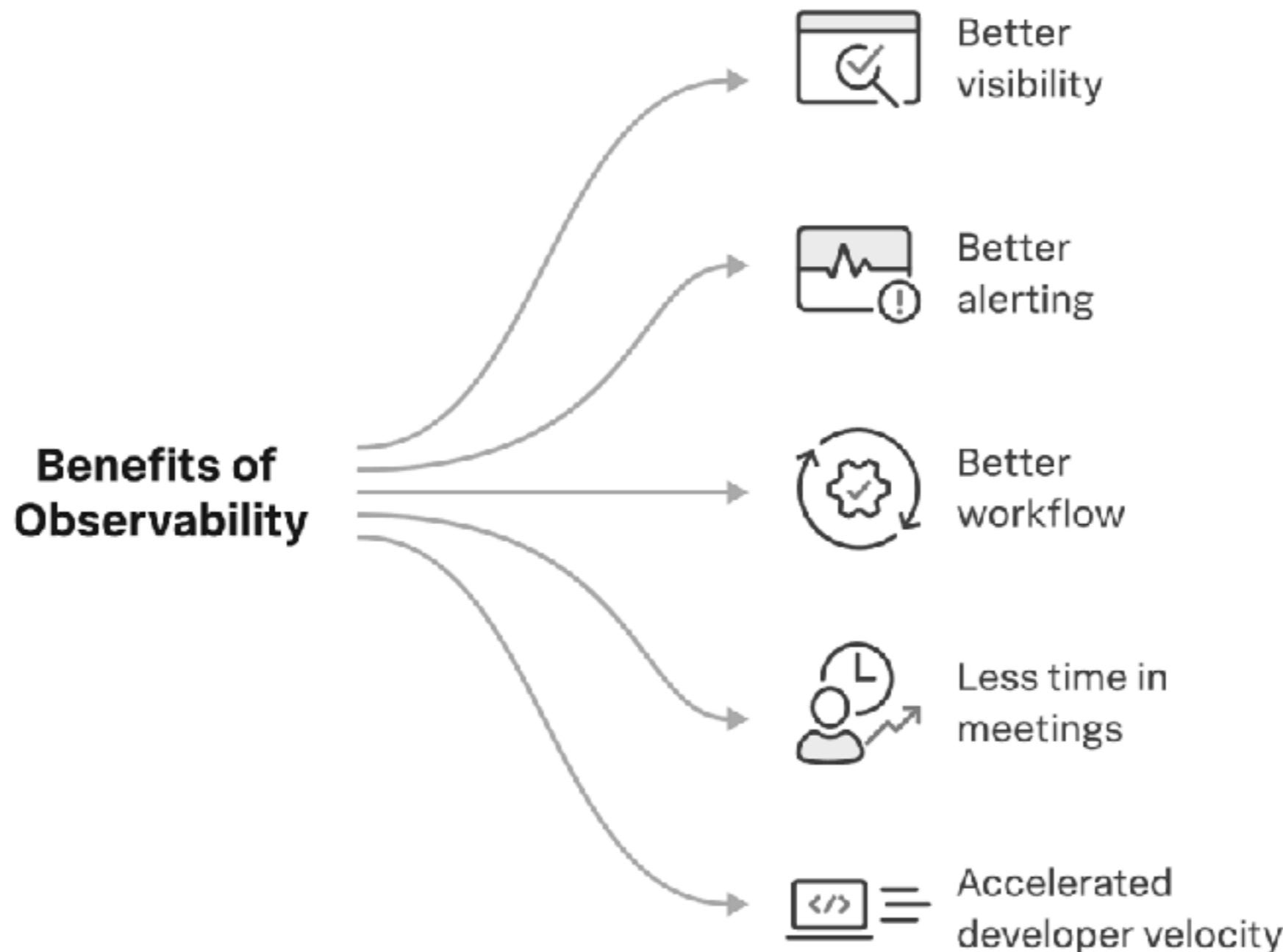
Application metrics
Distributed tracing
Log aggregation
Alert system
Exception tracking



Observability of services



Observability of services



Application metrics

Metrics represent logs numerically over intervals of time

Request rate

Error rate

Health

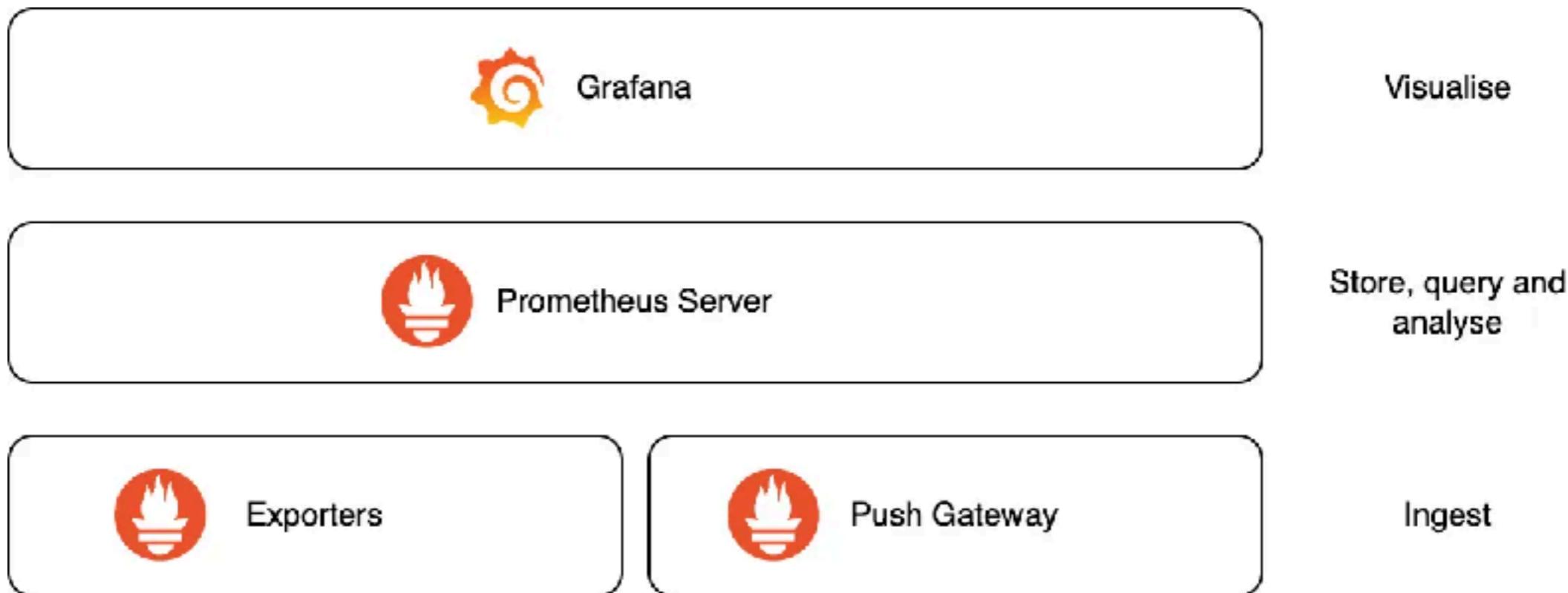
Latency

Utilization

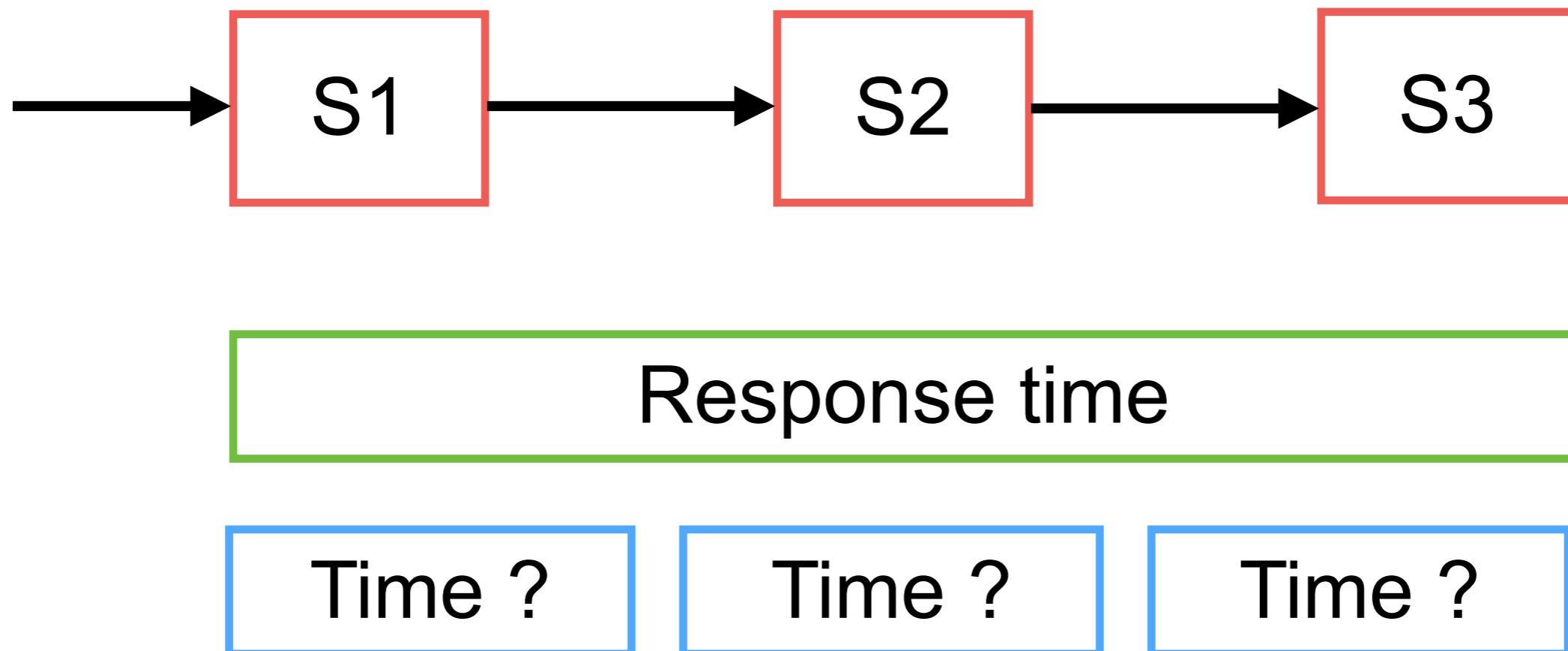
Availability



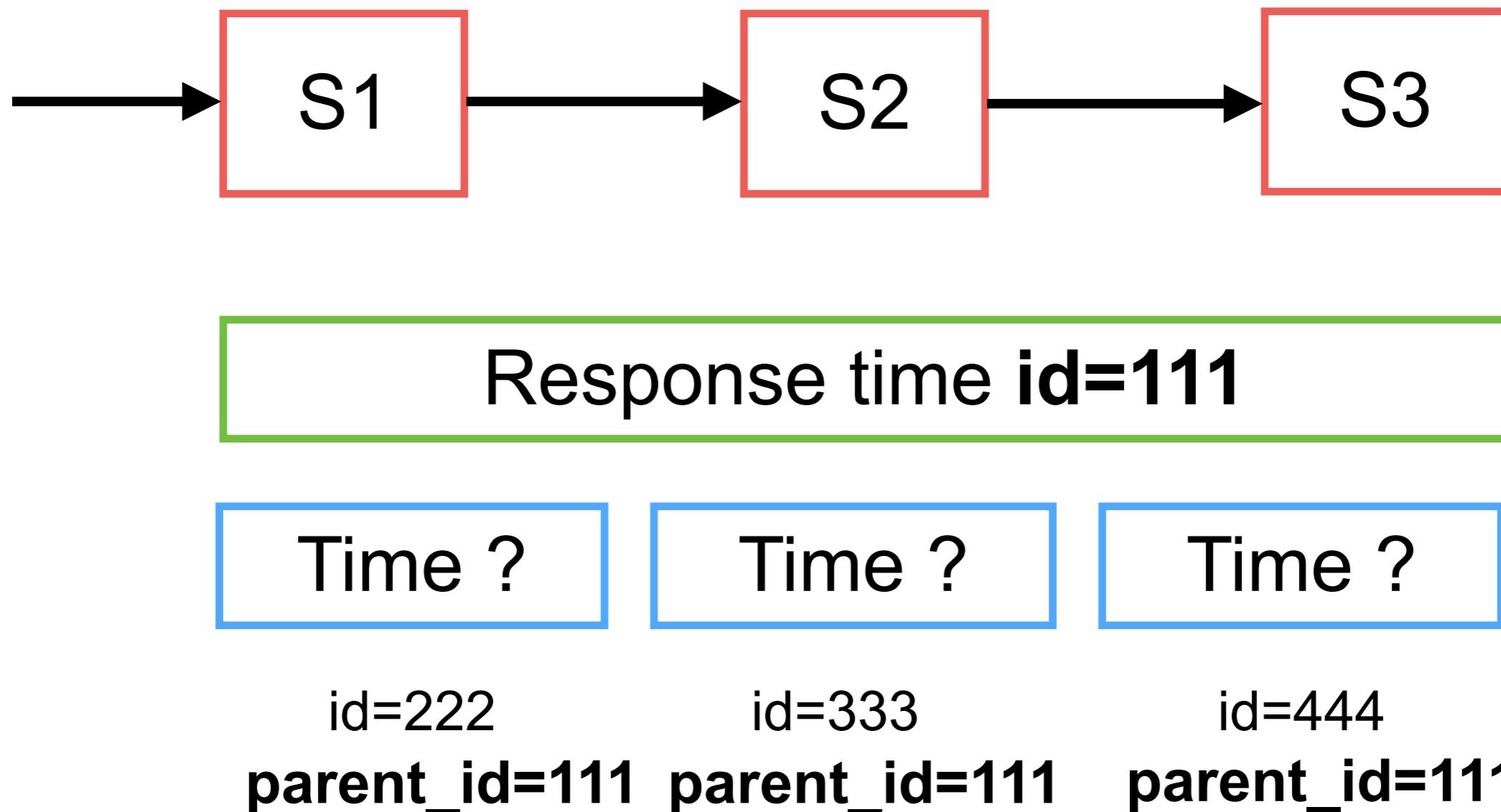
Application metrics



Distributed tracing



Correlation IDs





OpenTelemetry

High-quality, ubiquitous, and portable telemetry to enable effective observability

[Learn more](#)[Try the demo](#)

Get started based on your role

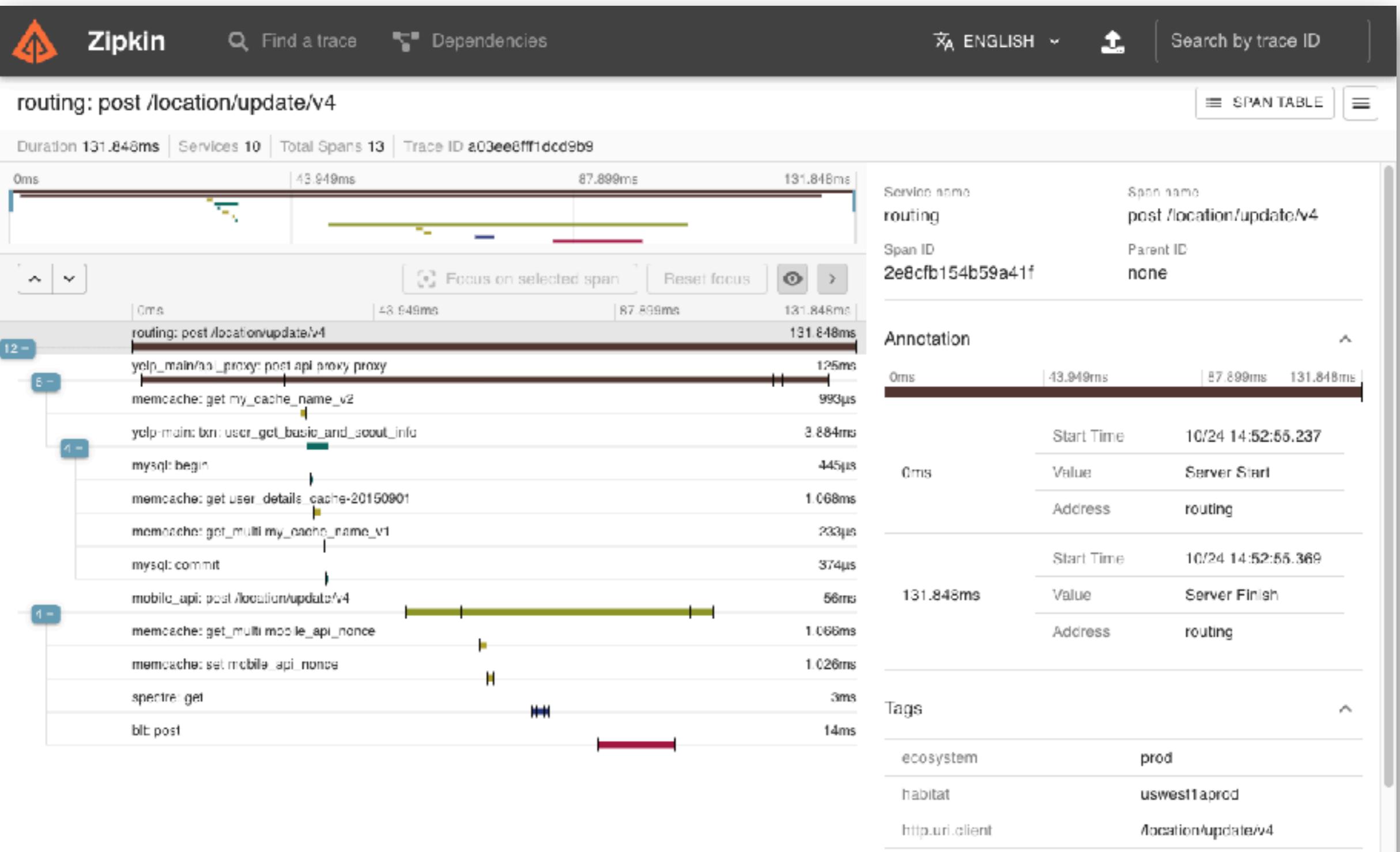
[Dev](#)[Ops](#)

<https://opentelemetry.io/>



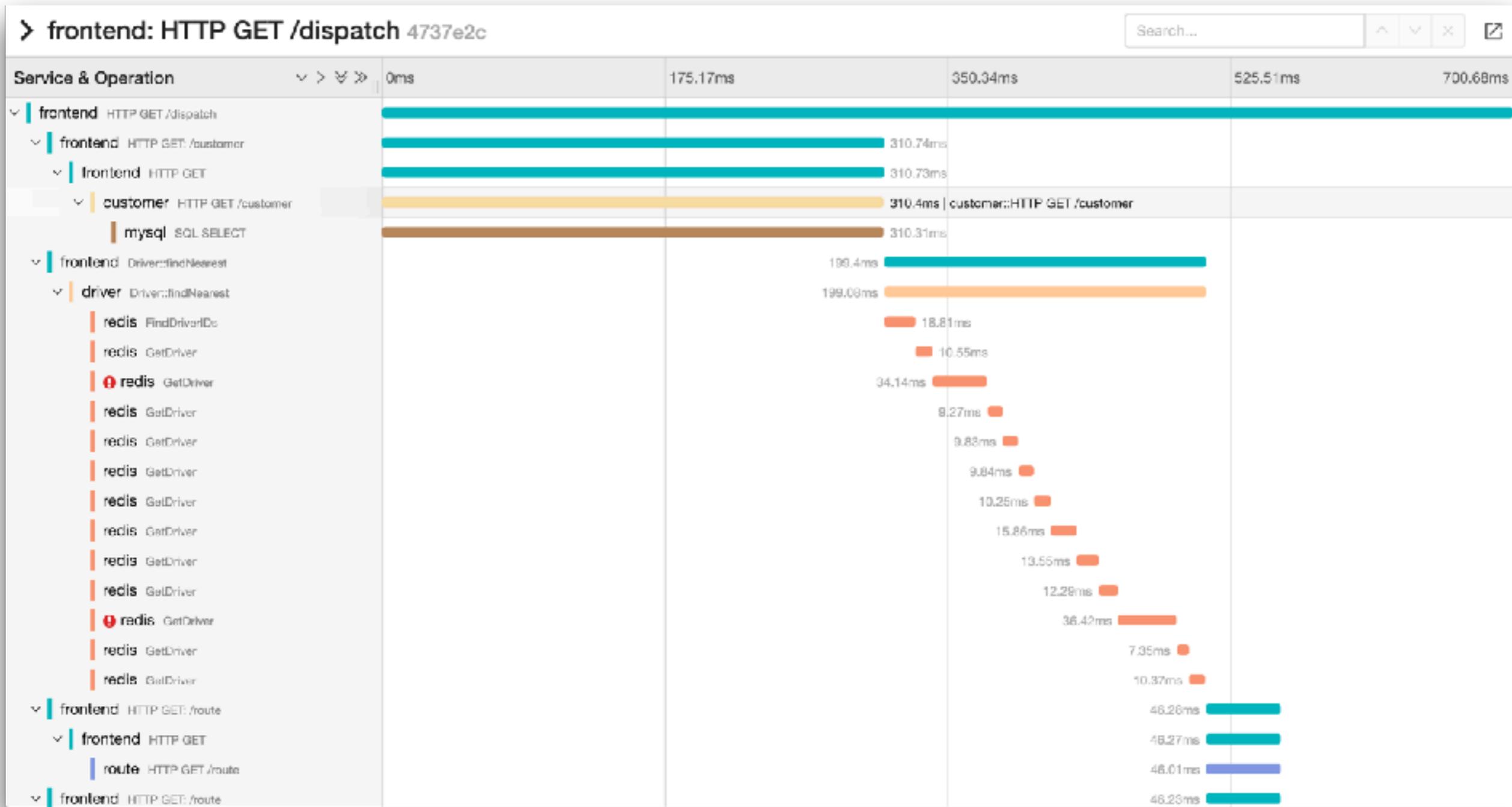
Microservices

© 2020 - 2023 Siam Chamnankit Company Limited. All rights reserved.



<https://zipkin.io/>

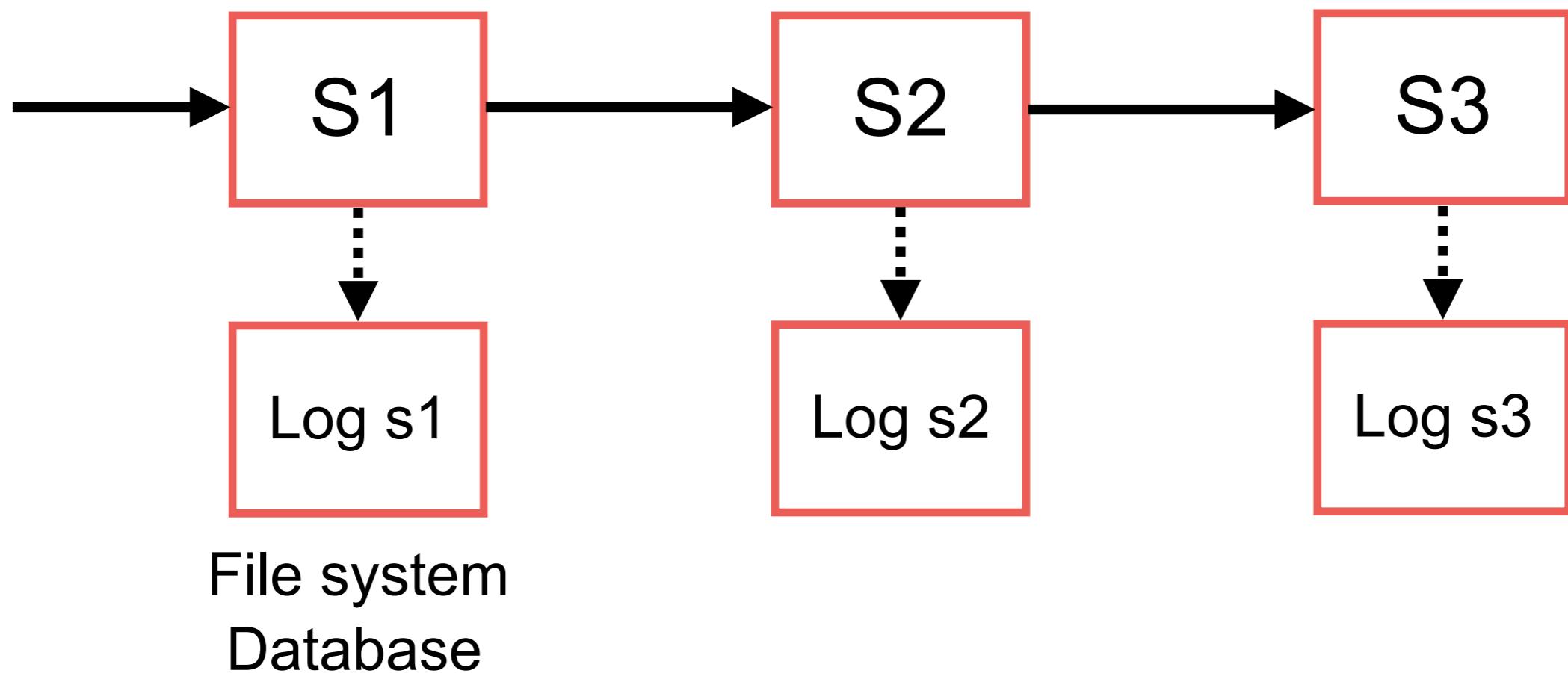




<https://www.jaegertracing.io/>



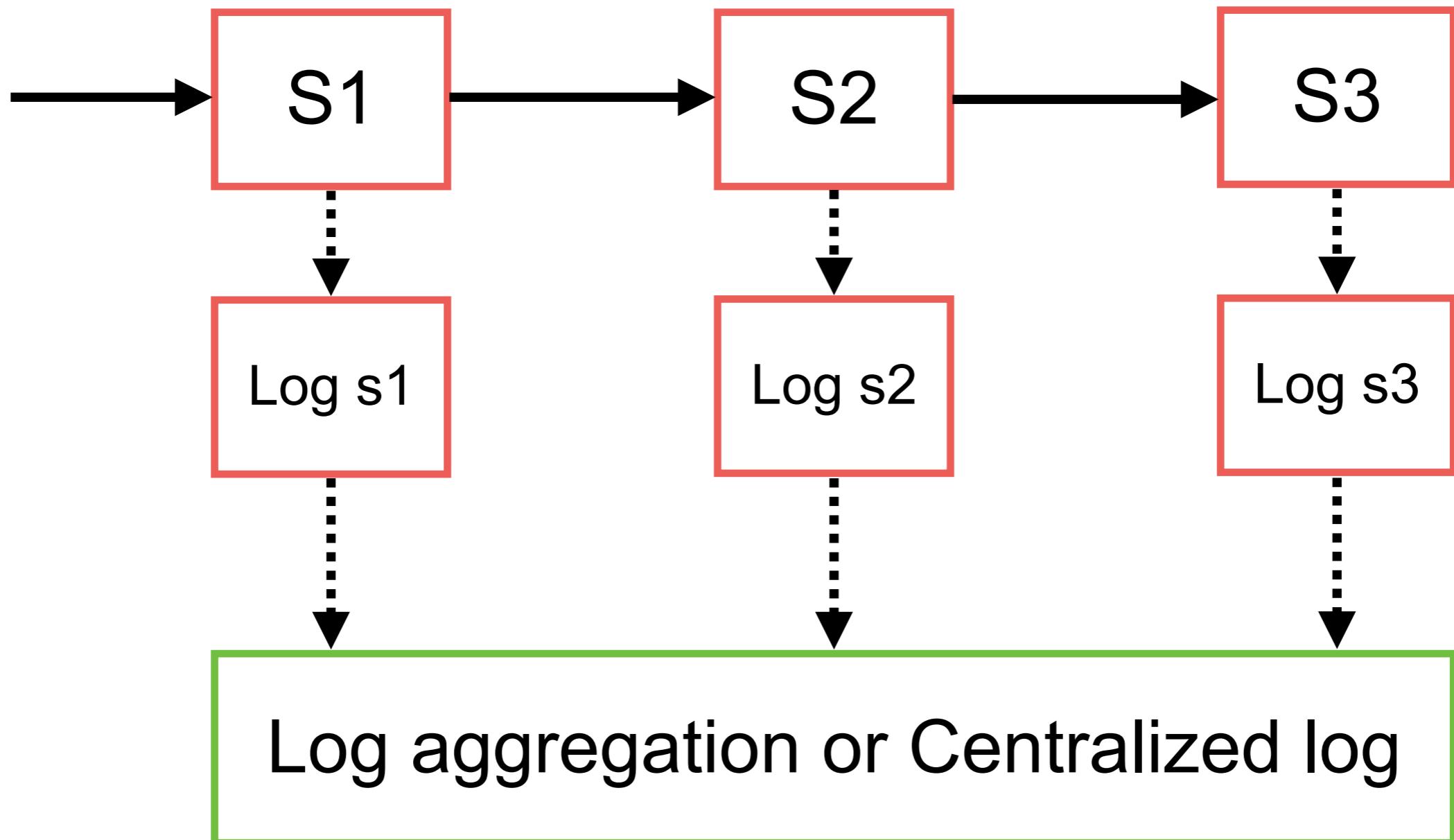
Log aggregation



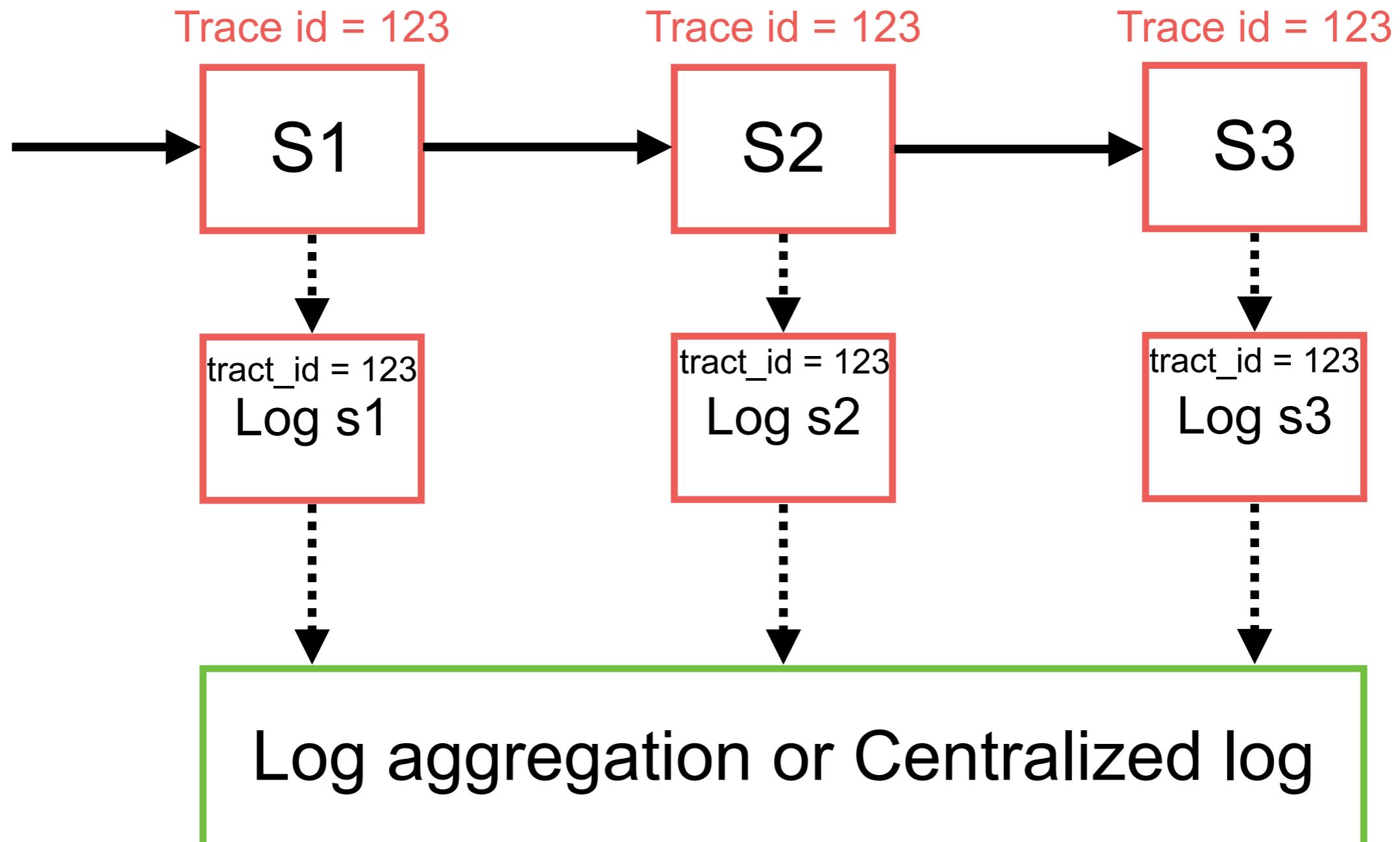
How to find errors from log ?



Log aggregation



Log aggregation + Correlation IDs



Consistent Structure Logs

Property	Description	Example
Timestamp	Date and time of the log	2023-07-01
Log level	DEBUG, INFO, ERROR	
Trace Id or Correlation Id	Unique identifier that refer to other logs from all services	
Event/Action Name	Identify to event or action of log	Authentication fail
Service ID/ Name	Identify to service	
Request path	Path for the request	/api/products

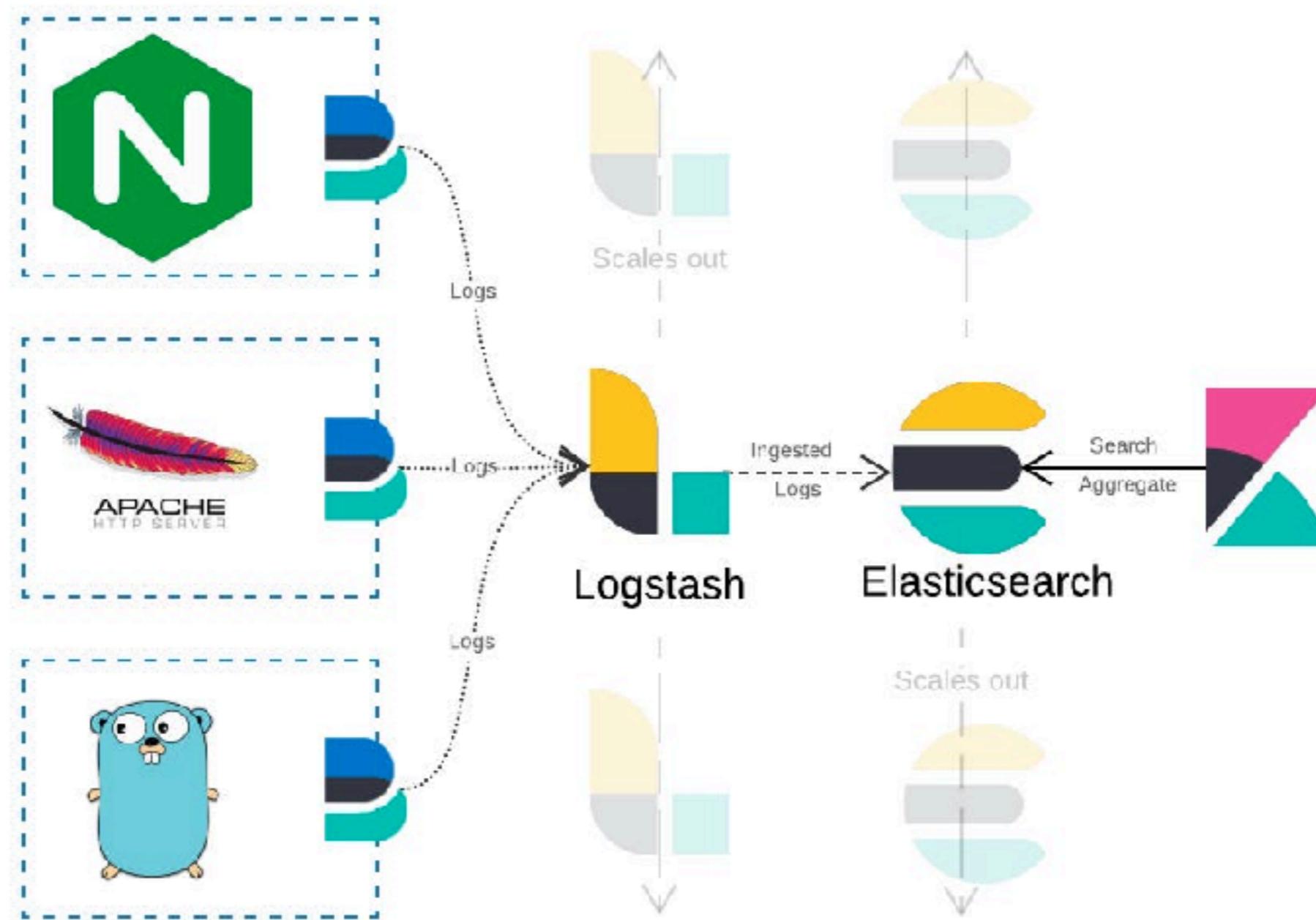


Don't keep !!

```
com.framework.FrameworkException: Error in web request
  at com.framework.ApplicationStarter.lambda$start$0(ApplicationStarter.java:15)
  at spark.RouteImpl$1.handle(RouteImpl.java:72)
  at spark.http.matching.Routes.execute(Routes.java:61)
  at spark.http.matching.MatcherFilter.doFilter(MatcherFilter.java:134)
  at spark.embeddedserver.jetty.JettyHandler.doHandle(JettyHandler.java:50)
  at org.eclipse.jetty.server.session.SessionHandler.doScope(SessionHandler.java:1568)
  at org.eclipse.jetty.server.handler.ScopedHandler.handle(ScopedHandler.java:144)
  at org.eclipse.jetty.server.handler.HandlerWrapper.handle(HandlerWrapper.java:132)
  at org.eclipse.jetty.server.Server.handle(Server.java:503)
  at org.eclipse.jetty.server.HttpChannel.handle(HttpChannel.java:364)
  at org.eclipse.jetty.server.HttpConnection.onFillable(HttpConnection.java:260)
  at org.eclipse.jetty.io.AbstractConnection$ReadCallback.succeeded(AbstractConnection.java:305)
  at org.eclipse.jetty.io.FillInterest.fillable(FillInterest.java:103)
  at org.eclipse.jetty.io.ChannelEndPoint$2.run(ChannelEndPoint.java:118)
  at org.eclipse.jetty.util.thread.QueuedThreadPool.runJob(QueuedThreadPool.java:765)
  at org.eclipse.jetty.util.thread.QueuedThreadPool$2.run(QueuedThreadPool.java:683)
  at java.base/java.lang.Thread.run(Thread.java:834)
Caused by: com.project.module.MyProjectFooBarException: The number of FooBars cannot be zero
  at com.project.module.MyProject.anotherMethod(MyProject.java:20)
  at com.project.module.MyProject.someMethod(MyProject.java:12)
  at com.framework.ApplicationStarter.lambda$start$0(ApplicationStarter.java:13)
  ... 16 more
Caused by: java.lang.ArithmaticException: The denominator must not be zero
  at org.apache.commons.lang3.math.Fraction.getFraction(Fraction.java:143)
  at com.project.module.MyProject.anotherMethod(MyProject.java:18)
  ... 18 more
```



ELK stack



More ...



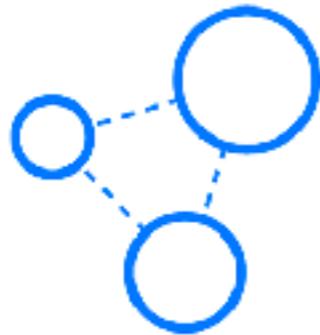
How to develop ?
How to test ?
How to deploy ?



Summary



Beyond Microservice Process and Organization



Flexible organizational structure

With clear roles and accountabilities



Efficient meeting formats

Geared toward action and eliminating over-analysis



More autonomy to teams and individuals

Individuals solve issues directly without bureaucracy



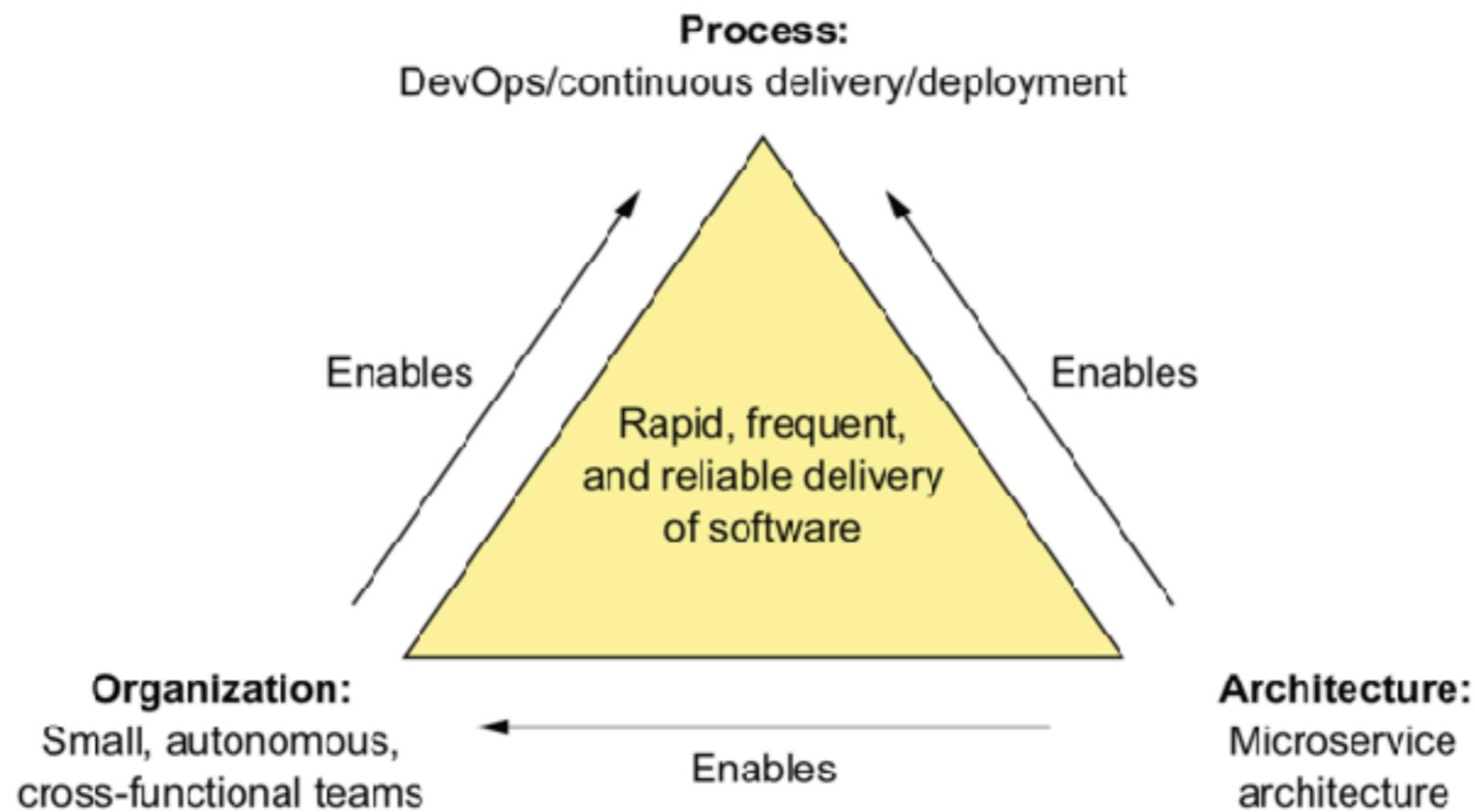
Unique decision-making process

To continuously evolve the organization's structure.

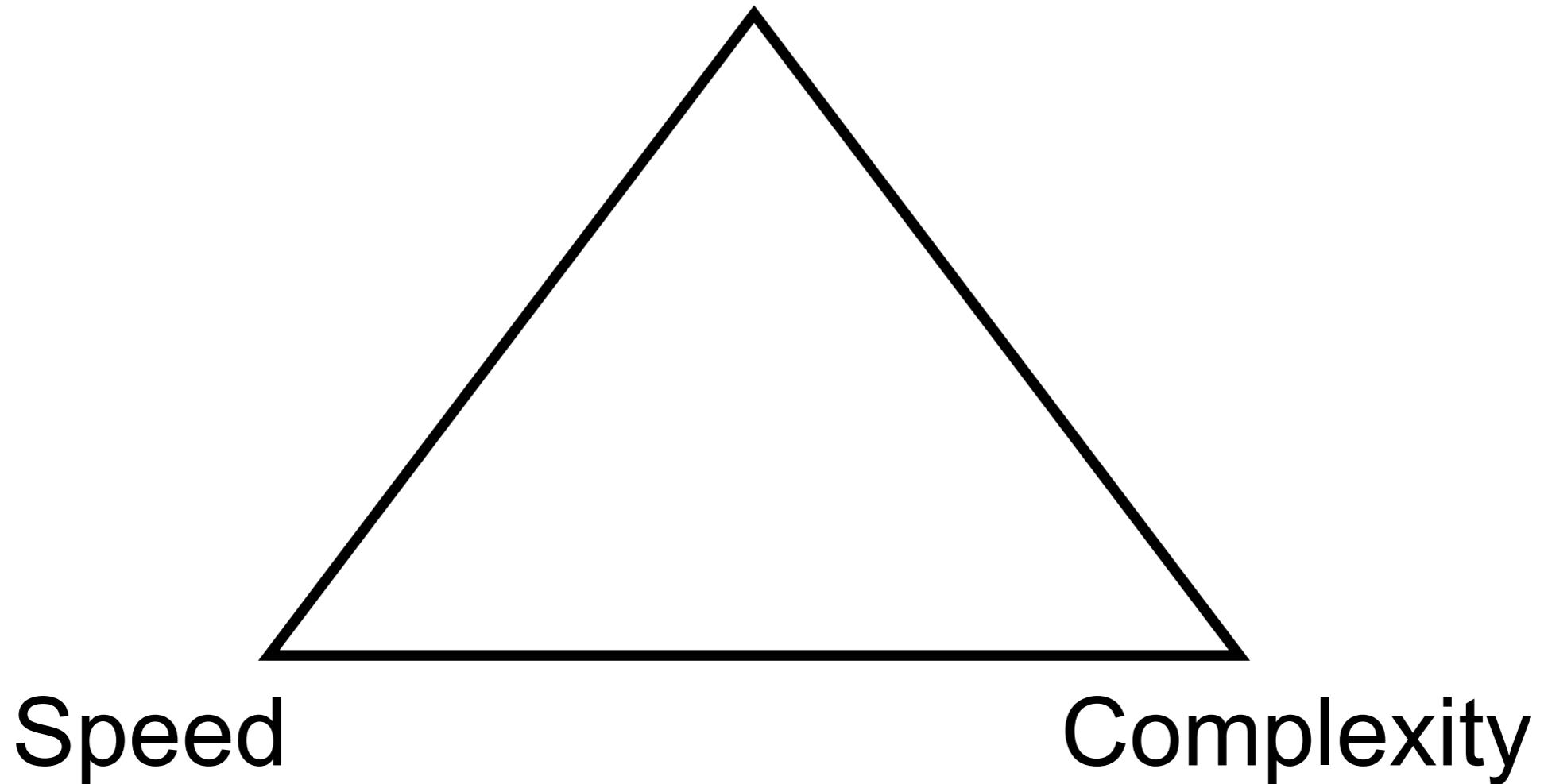


Success project needs ...

Organization process
Development process
Delivery process



Customer value



Don't forget about the human side when adopt Microservice



Let's start with good monolith



Module 1

Module 2

Module 3

Module 4

Module 5

Module 6



Find your problem



Module 1

Module 2

Module 3

Module 4

Module 5

Module 6



Module 1

Module 2

Module 3

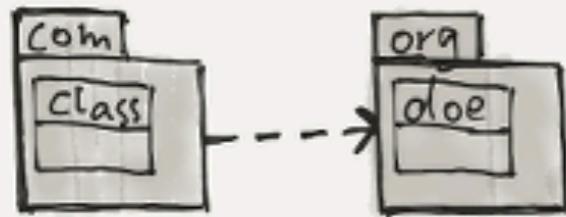
Module 5

Module 6

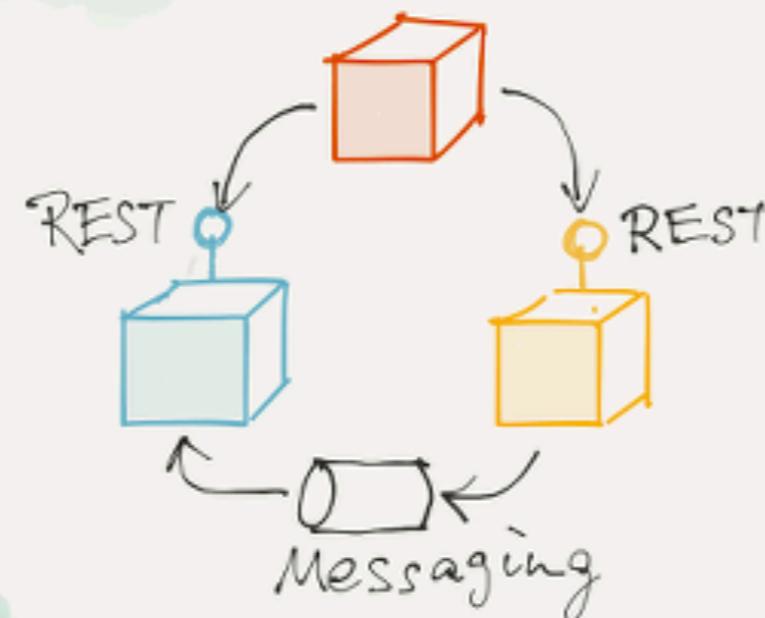
Module 4



Architecture



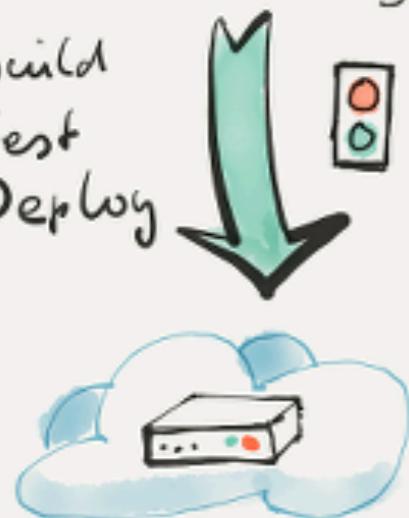
Microservices



Deployment

Continuous Delivery

`{ var i=1; }`
Build
Test
Deploy



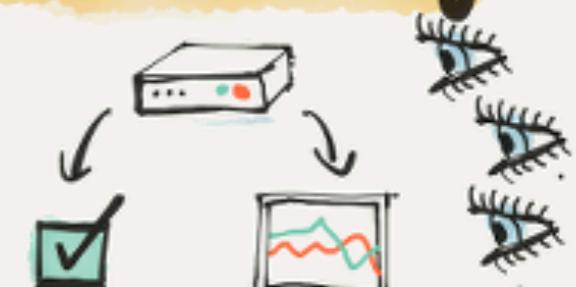
Infrastructure



People & Teams



Monitoring

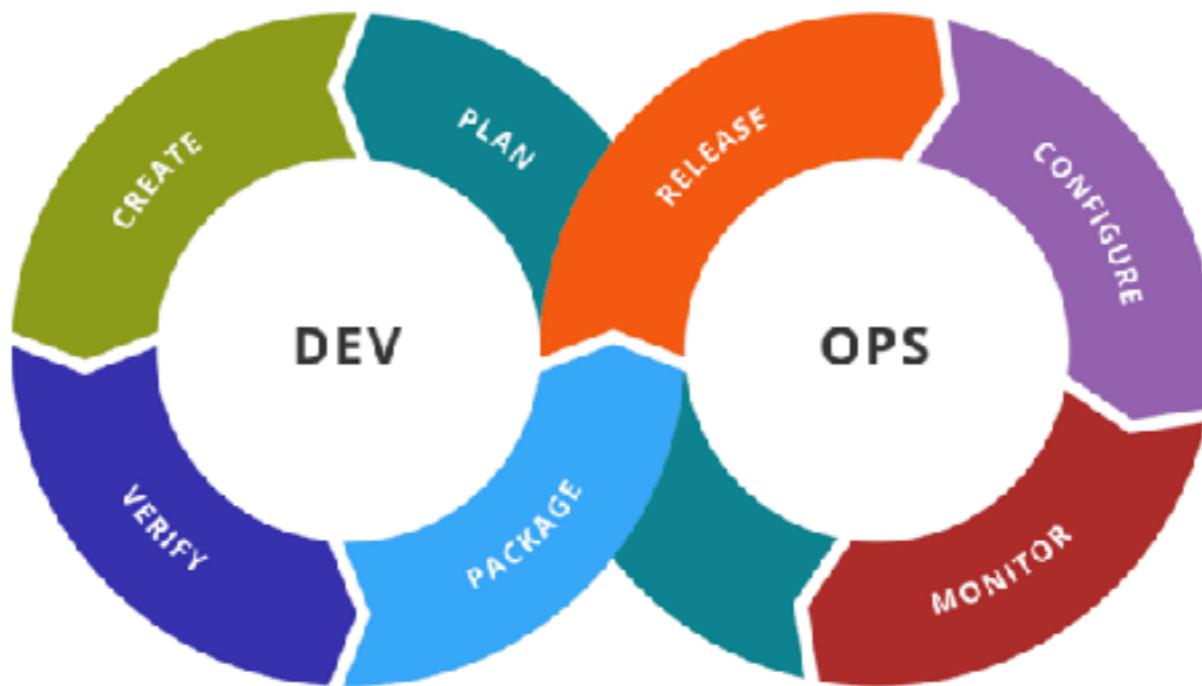


Features & Technology



Microservice prerequisites

Rapid provisioning
Basic monitoring
Rapid Application Deployment
DevOps culture



Let's start your journey !!

