Why Microservices Fail

INNOQ

EBERHARD WOLFF

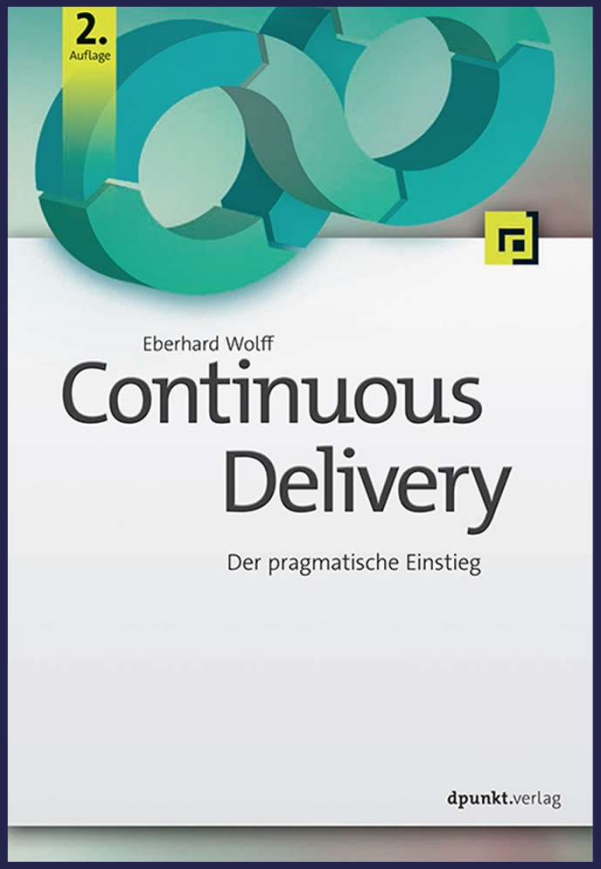
Fellow at INNOQ Deutschland GmbH

@ewolff

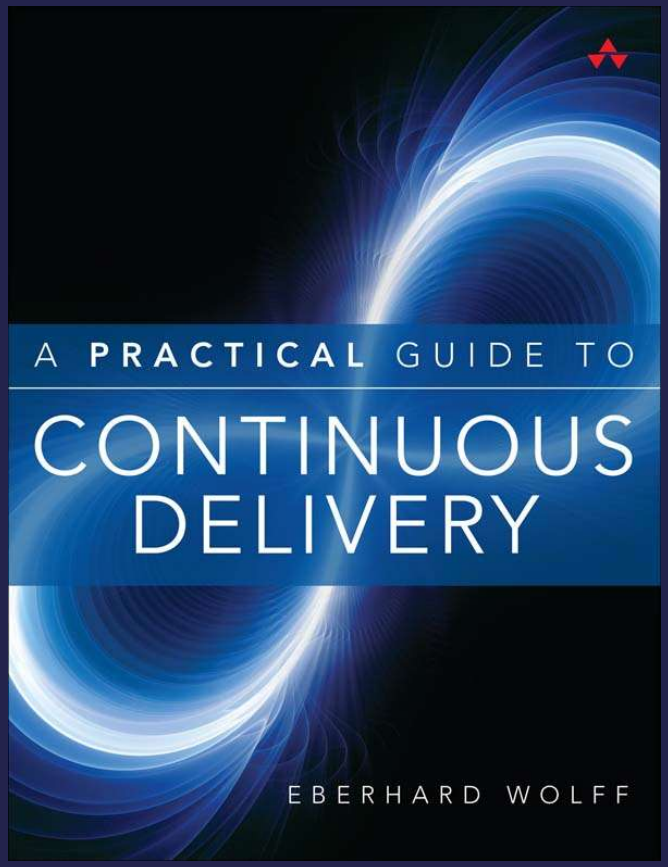
www.ewolff.com

FREE

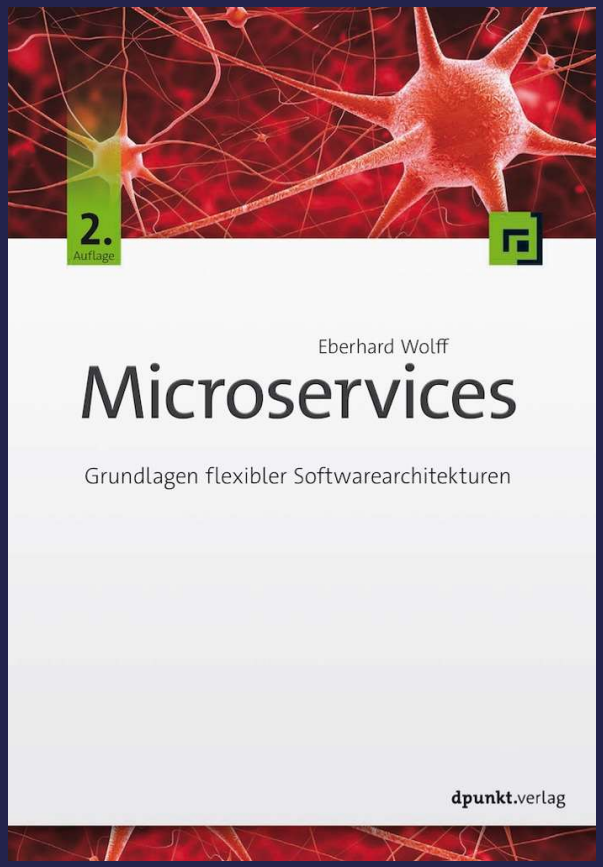
leanpub.com/service-mesh-primer/

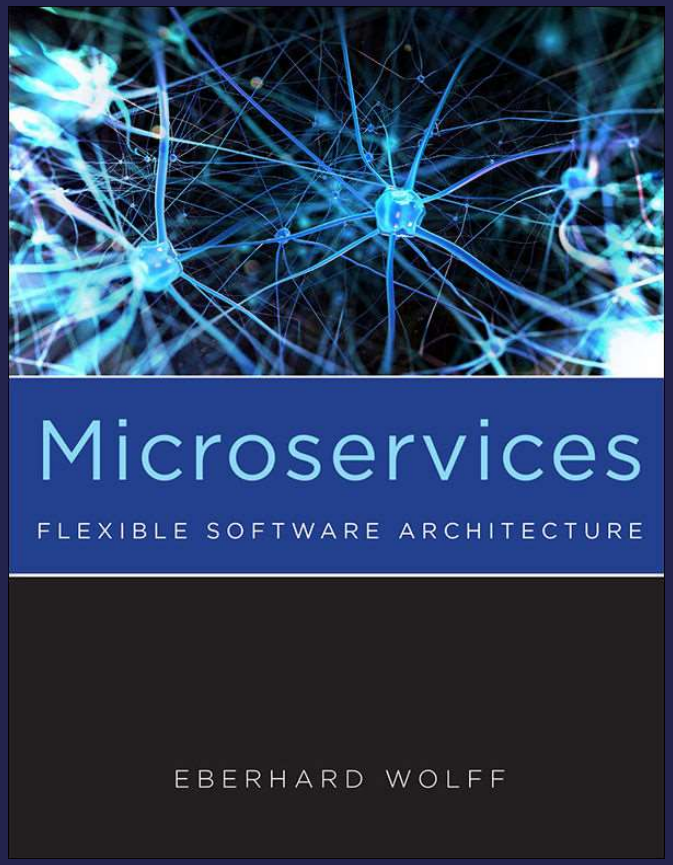


www.continuous-delivery-buch.de



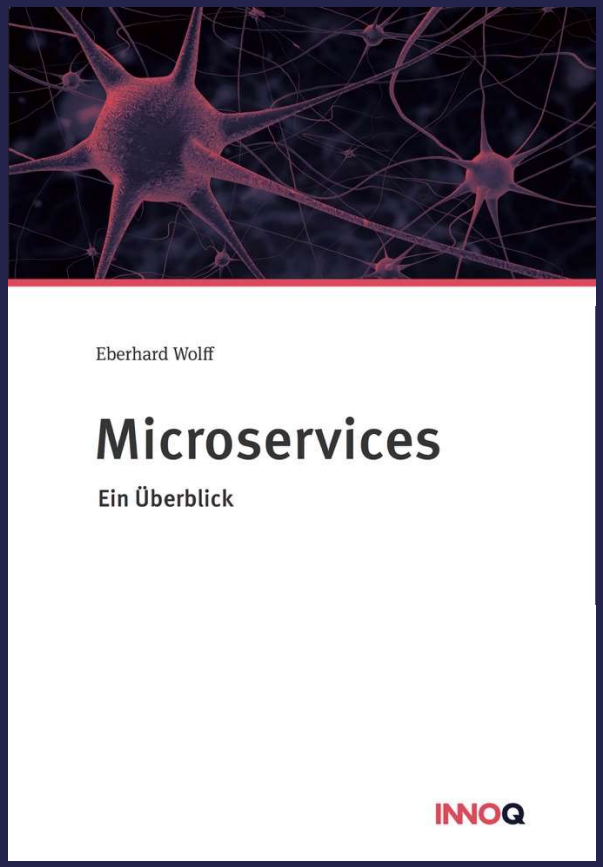
www.continuous-delivery-buch.de



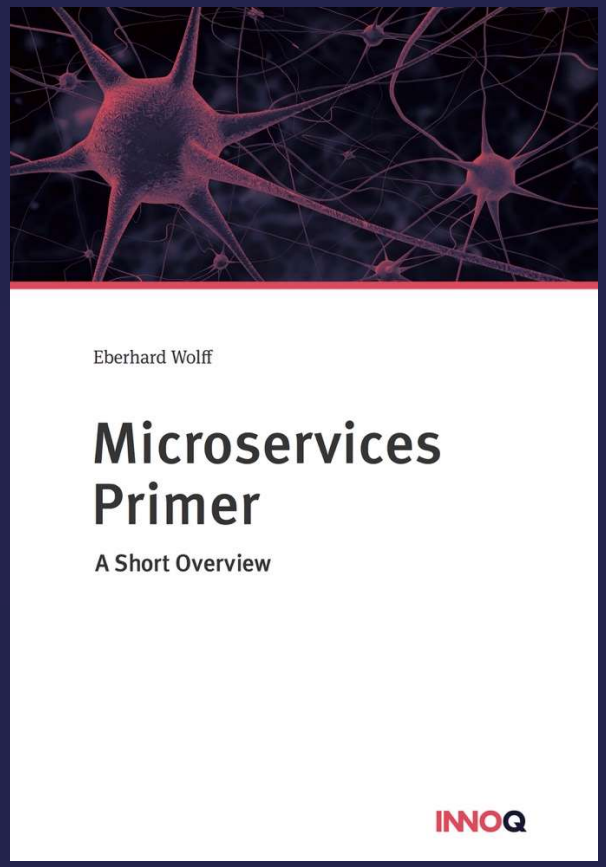


www.microservices-buch.de

www.microservices-book.com

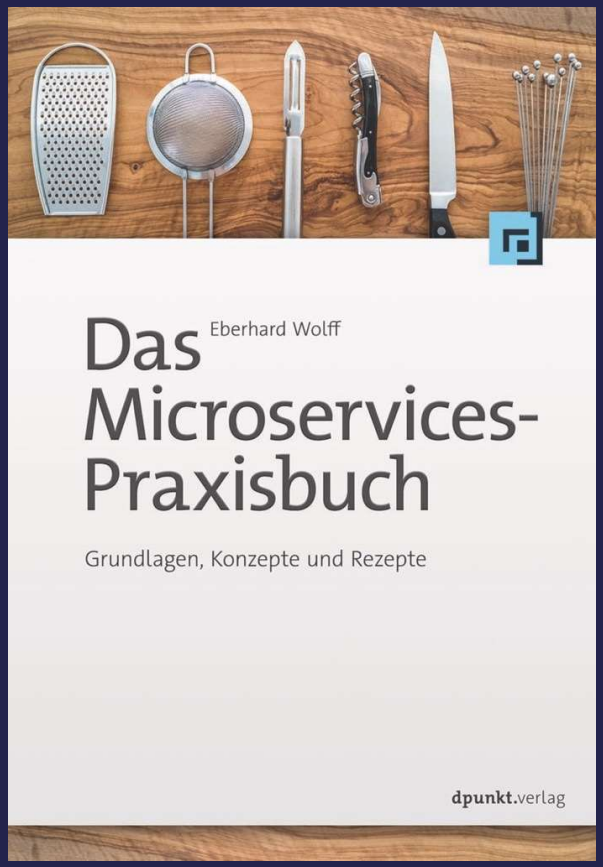


FREE

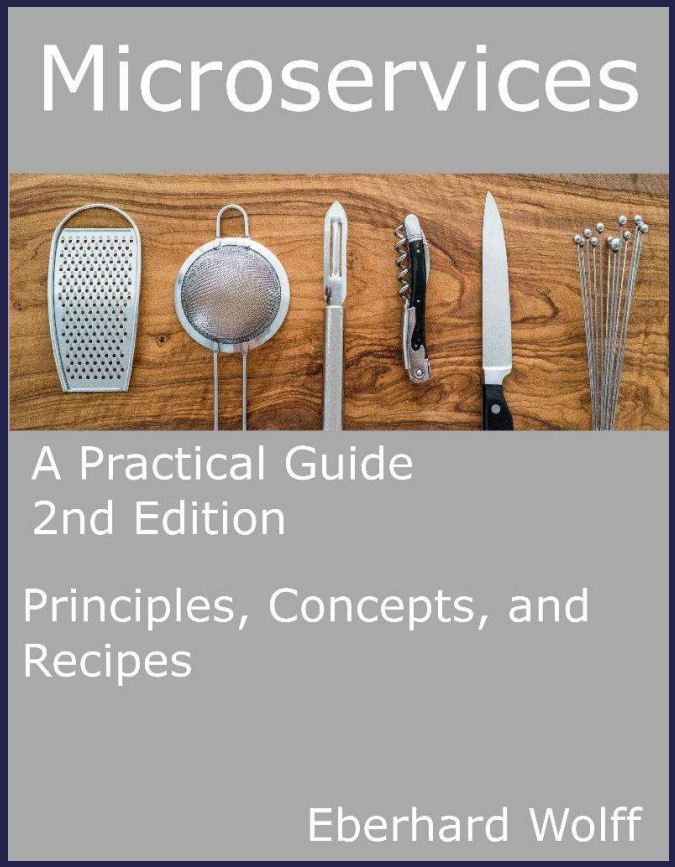


www.microservices-buch.de/ueberblick.html

www.microservices-book.com/primer.html



www.microservices-praxisbuch.de



www.practical-microservices.com

Eberhard Wolff

Microservices Rezepte

Technologien im Überblick

INNOQ

FREE

Eberhard Wolff

Microservices Recipes

Technology Overview

INNOQ

www.microservices-praxisbuch.de/rezepte.html

www.practical-microservices.com/recipes.html

FREE

www.ddd-referenz.de **www.domainlanguage.com/ddd/reference**


What are
Microservices?

Creator: INNOQ | www.isa-principles.org

ISA

Independent Systems Architecture

Creator: INNOQ | www.isa-principles.org

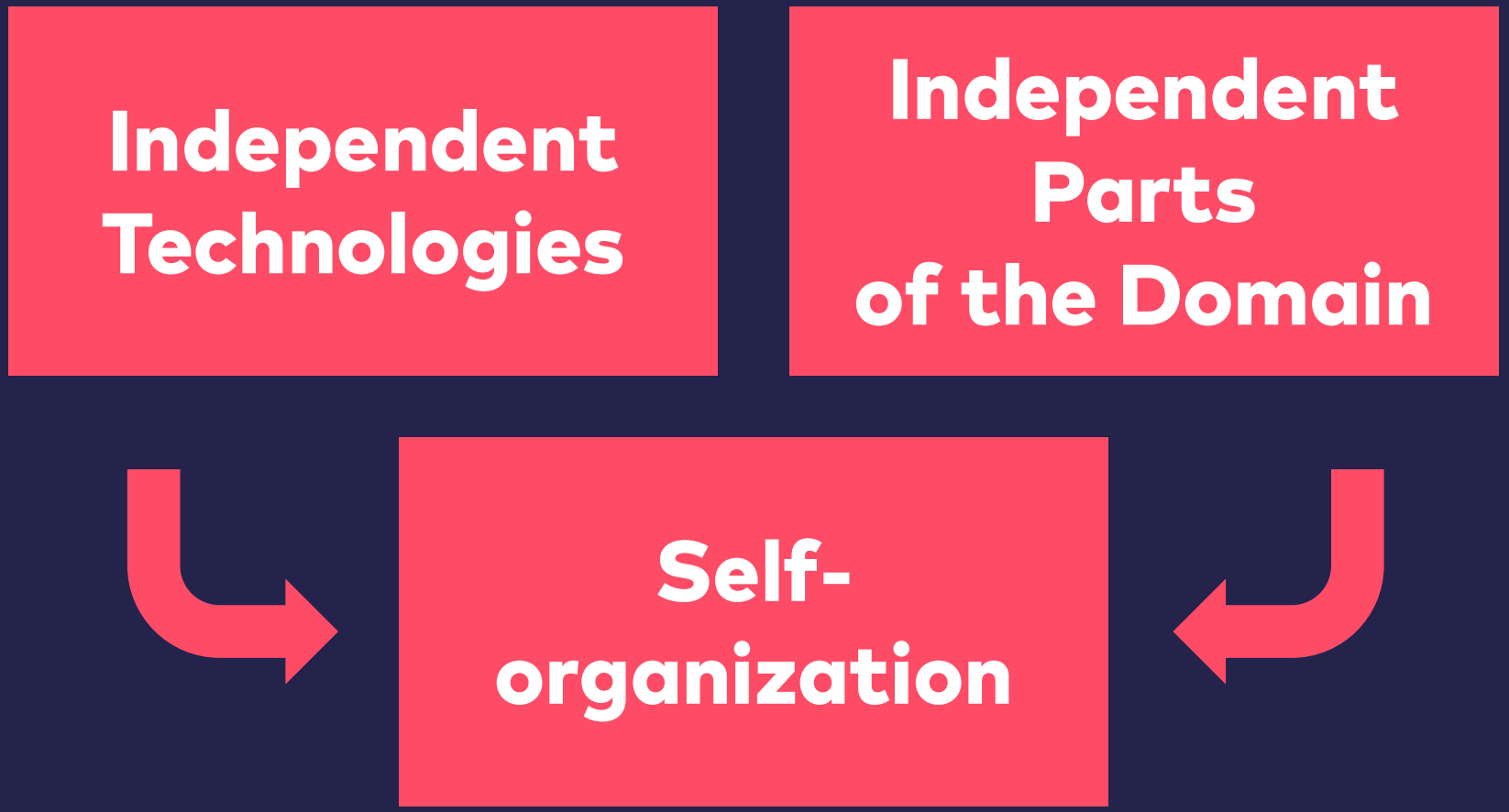
Modules	Macro / Micro Architecture	Container
Integration & Communication	Authentication & Metadata	Independent Continuous Delivery Pipeline
Standardized Operations	Standards: Interface only	Resilience


Why
Microservices?

Technological Benefits

- Decoupled Development
- Decoupled Scalability
- Decoupled Crashes
- Security
- Architecture Firewalls
- Replaceability
- Continuous Delivery

Organizational Benefits



Challenges

- Consistency
- Fail Safeness
- New Technologies
- Operations

Deployment Monolith

- Everything deployed at once
- Opposite of microservices

- You loose extreme decoupling
- ... and the other benefits.

- But no microservices challenges
- Valid trade-off


Why
Microservices
Fail

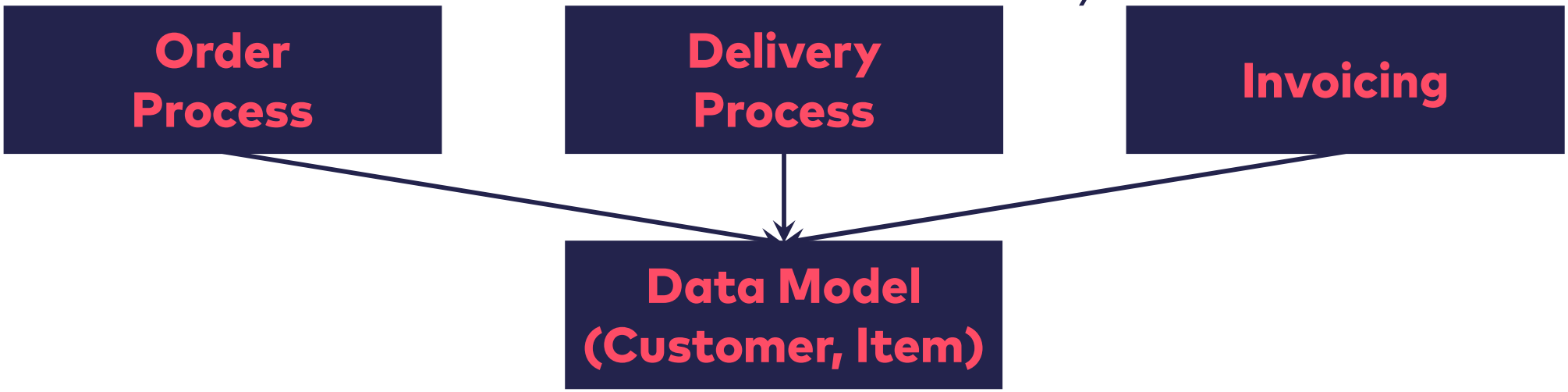
Data Model

Common Data Model

"The services need some common model to communicate!"

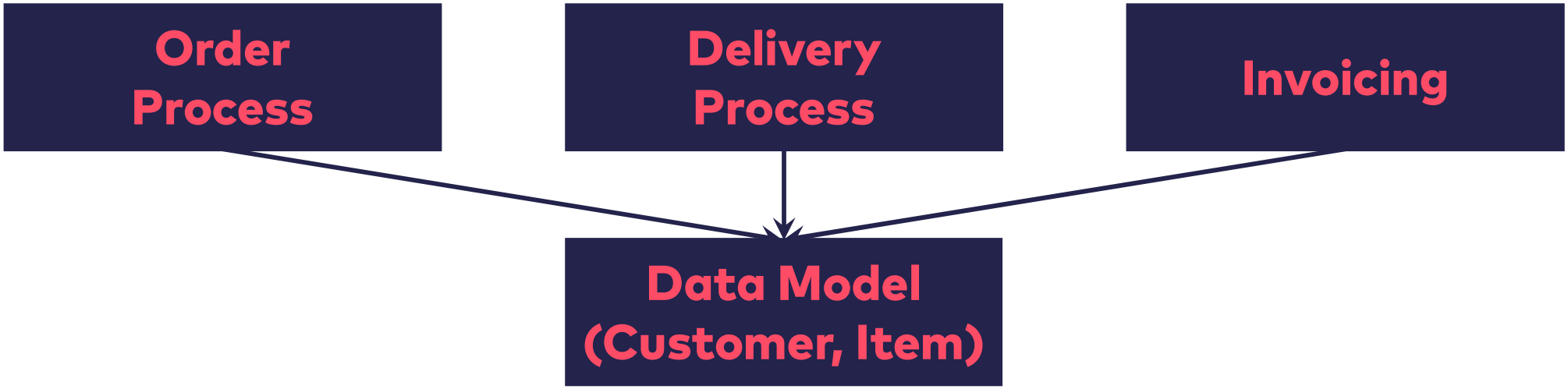
Common Data Model: Communication

- Common data model for communication only
- Might have separate internal model
- Data model = common library
- All services must use latest version of library



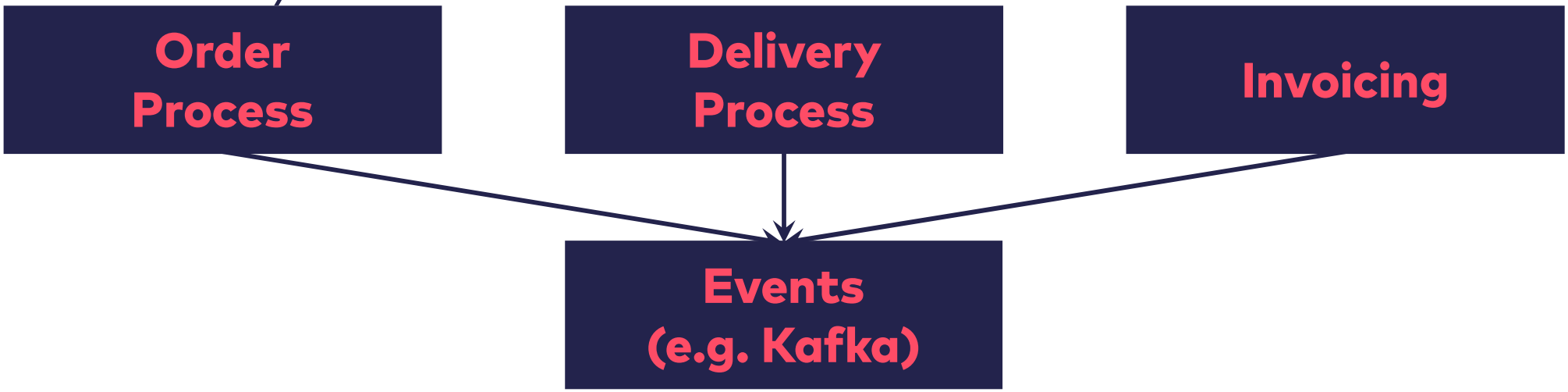
Common Data Model: Communication

- Change -> redeploy all services
- No decoupled deployment
- Deployment monolith with microservices challenges



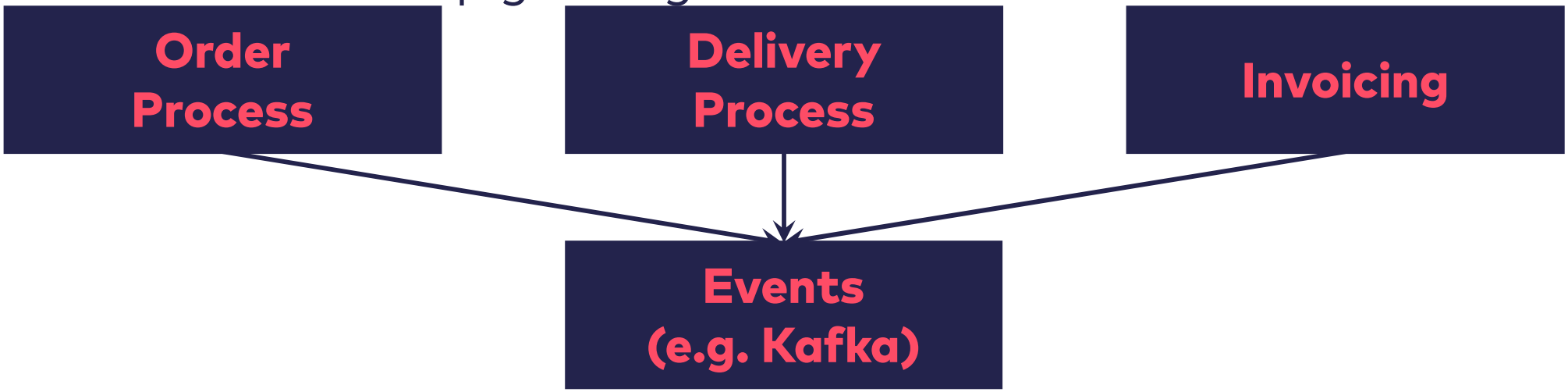
Common Data Model: Communication

- Data model = events stored e.g. in Kafka
- Event sourcing
- Rebuild local state from events
- Essentially a shared database schema



Common Data Model: Communication

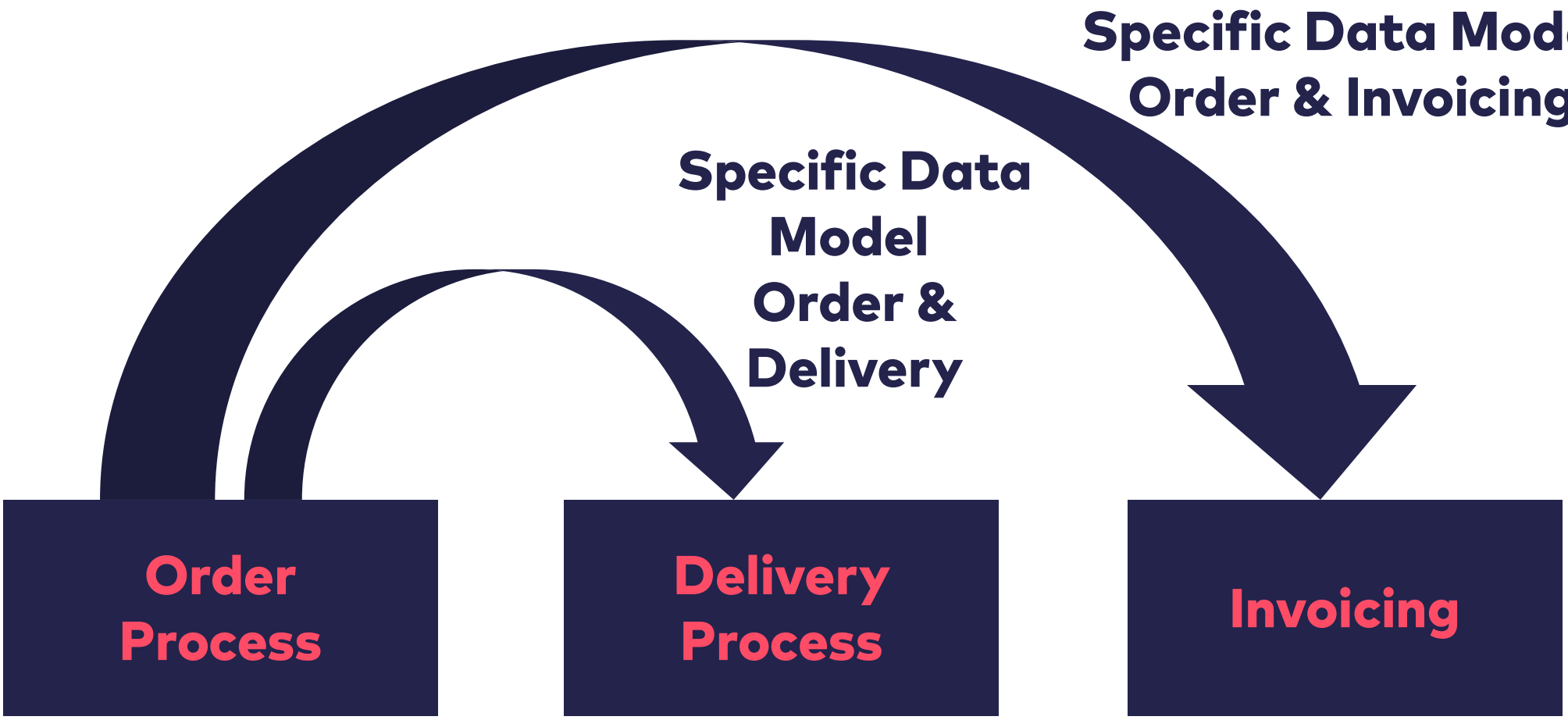
- Many dependencies
- Event data model hard to change
- Particularly hard: remove an attribute
- I.e. model will keep growing



Centralized Data Model: Cure

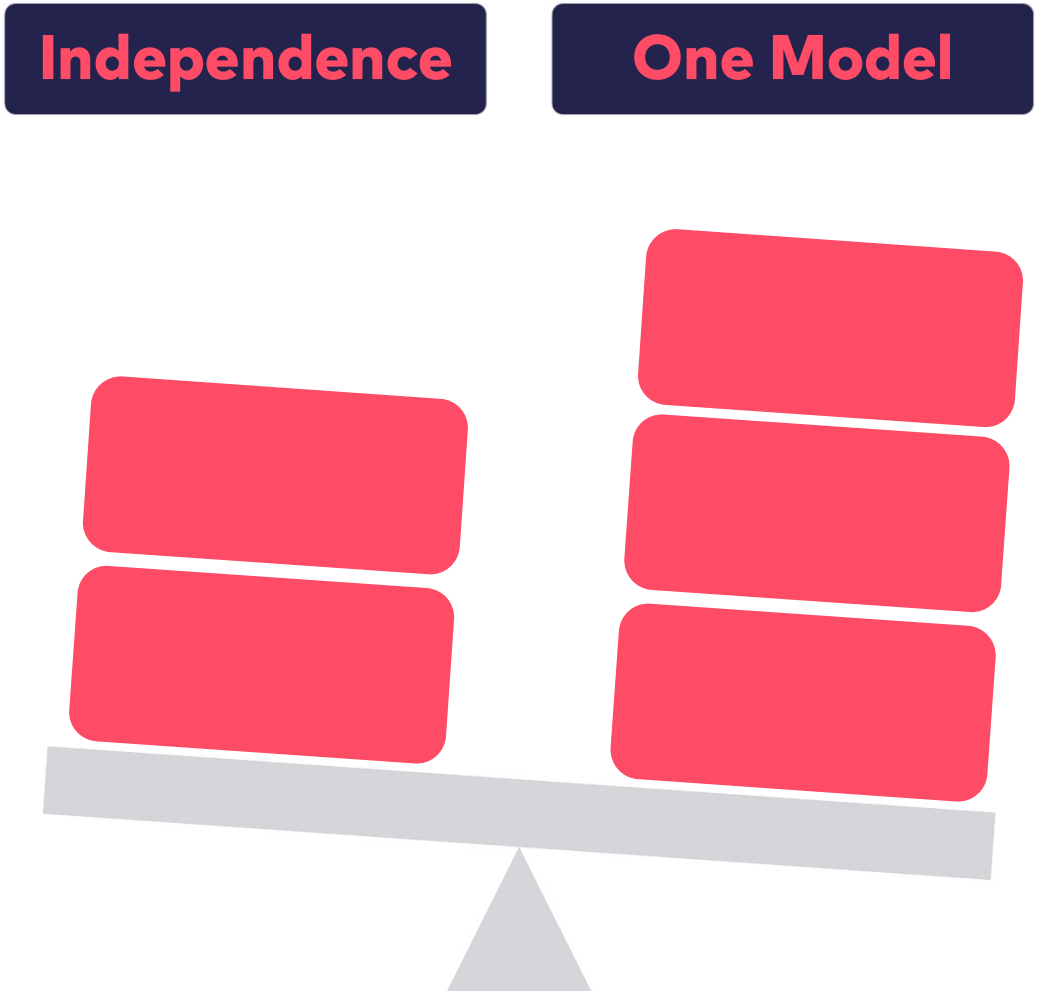
- Use separate local data models
- No global data model for communication!
- No common data model for events!
- Specific model for each interface between microservices!

Centralized Data Model: Cure



Data Model Inflation?

- Independence vs. one model
- Trade-off
- No one single best solution.



Flaky System

“What is resilience?”

Flaky Systems

- A lot more chances for failure
- Many servers
- Network
- Many services

Flaky Systems

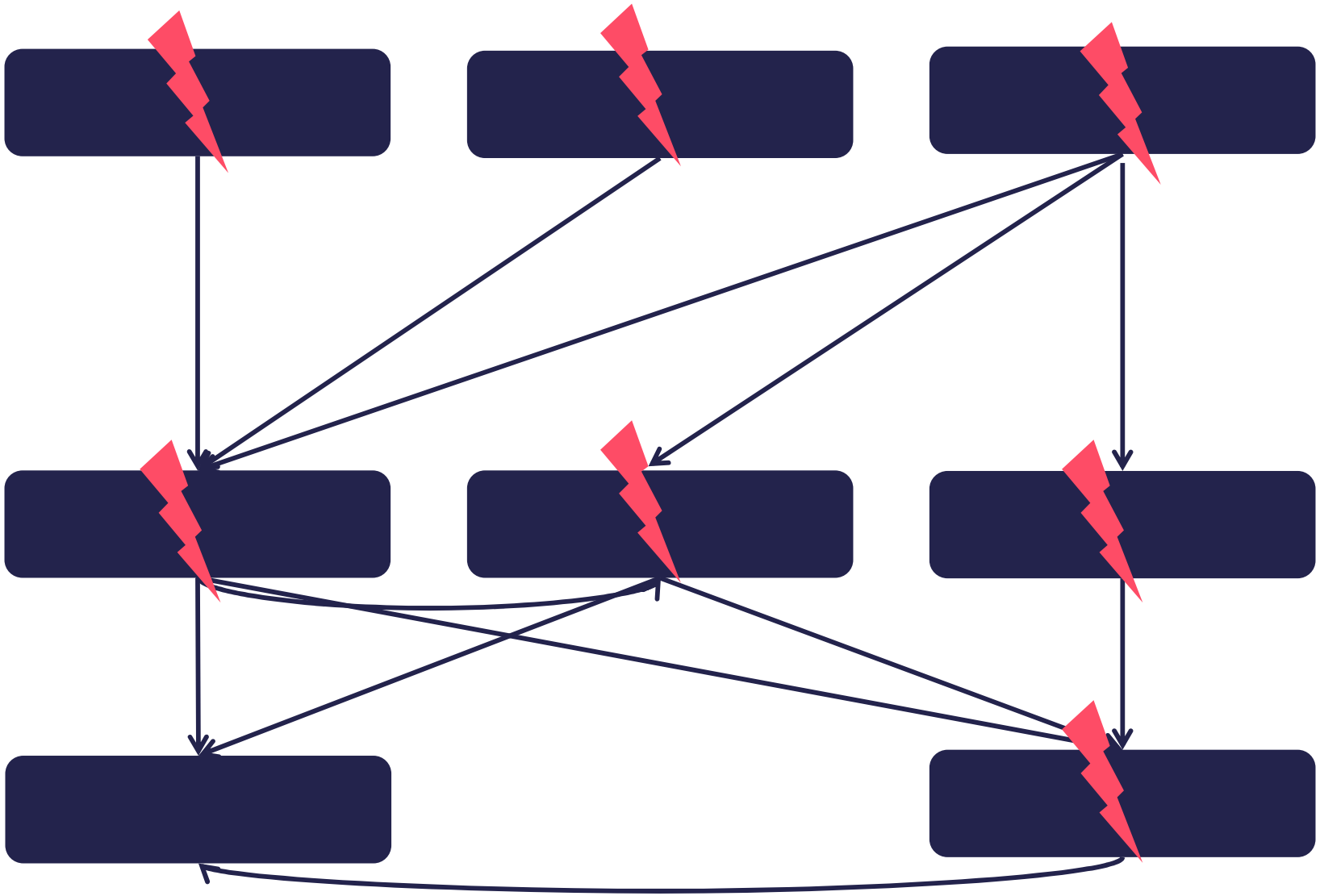
- Microservices depend on each other.
- One failed service might make another service fail.
- … and that makes another fail
- … and so on.
- Just like domino pieces

News

Fly ruins German domino world record attempt

A German domino team was attempting to break a record for miniature dominoes. But a fly triggered a premature chain reaction.

www.dw.com/en/fly-ruins-german-domino-world-record-attempt/a-44955761



Flaky Systems: Cure

- Resilience
- Microservice continues to operate

... even if another microservice fails.

- Probably not everything still work

e.g. process orders up to some limit.

- At least provide a sensible error

... don't make callers wait forever.

Flaky Systems: Cure

- Asynchronous communication = sensible default for failure:

Process messages later.

- What if the security service fails?
- Resilience = unauthenticated access?
- Probably not a good idea
- Resilience is limited

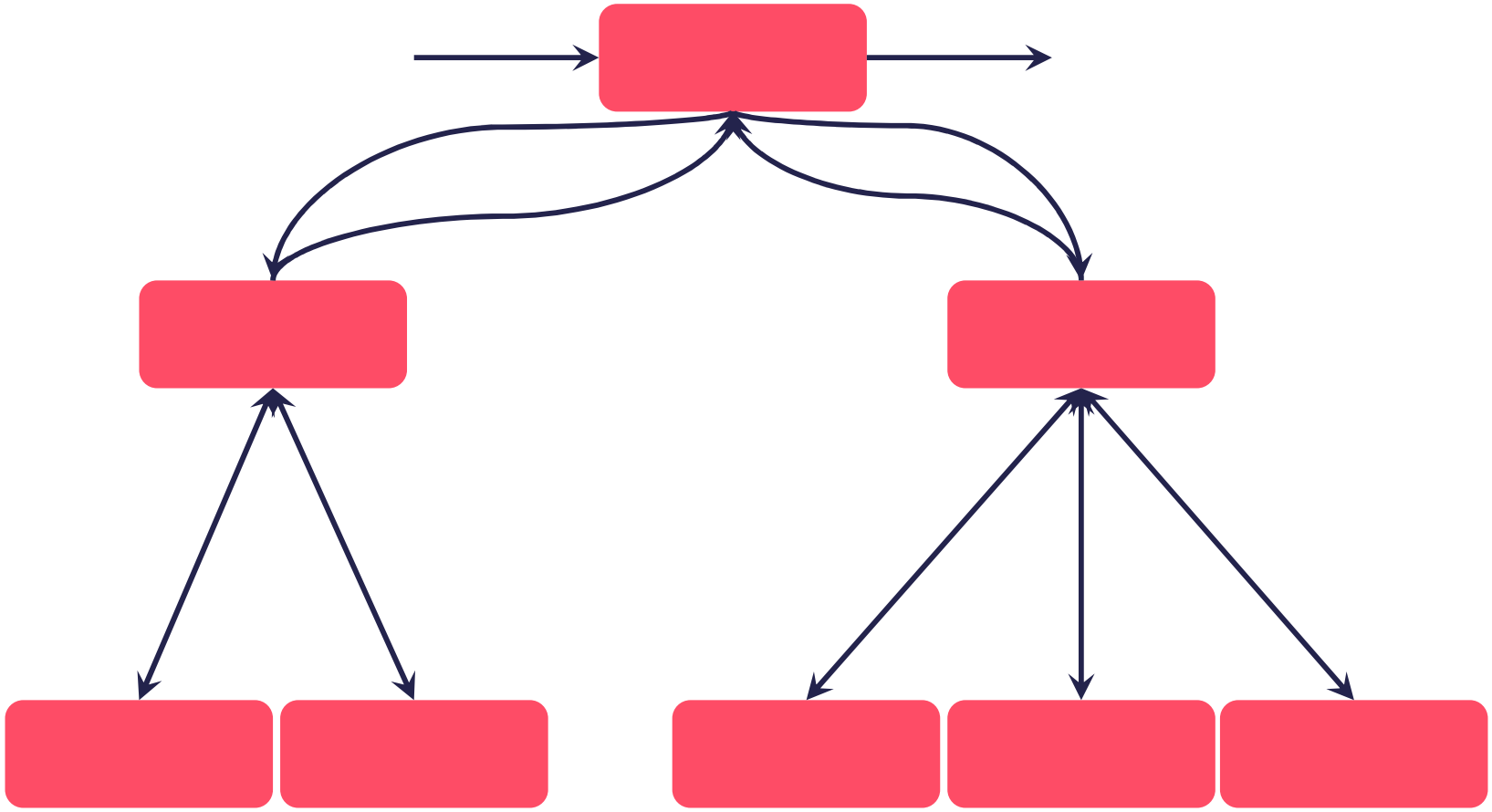
Synchronous Calls

"We do microservices the Netflix way!"

Cascading Synchronous Calls

- Easy to understand
- Similar to non-distributed programing

Cascading Synchronous Calls



Synchronous Calls: Challenge

- Performance issues due to network traffic
- Latencies add up
- … or calls have to be in parallel
- Flaky service: Hard to compensate failures
- Asynchronous resilience: Messages transferred later, inconsistencies

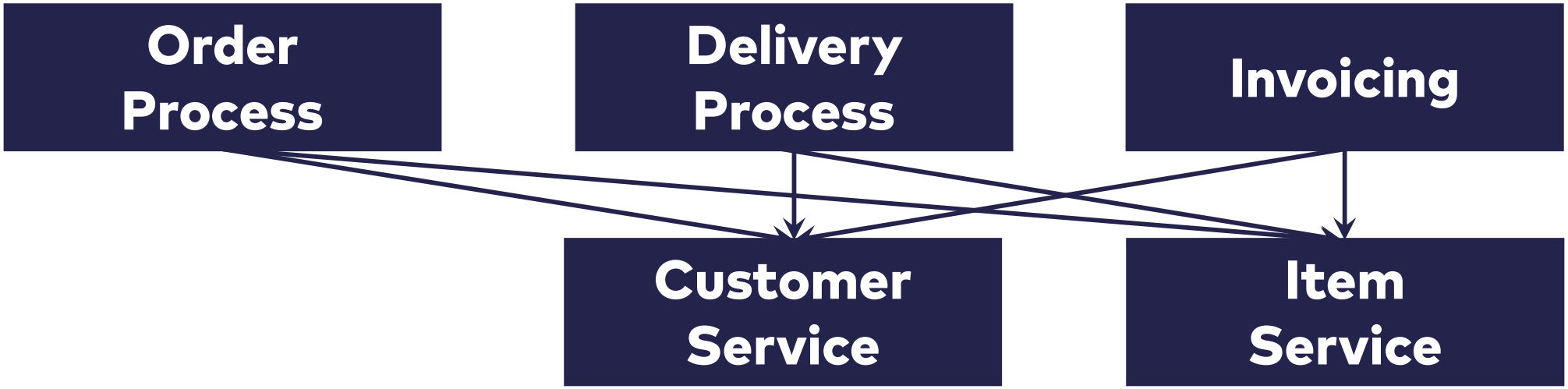
Synchronous Calls: Cure

- Go async
- Quite natural if you do business events.
- Independent parts of the domain mean less communication

Entity Service

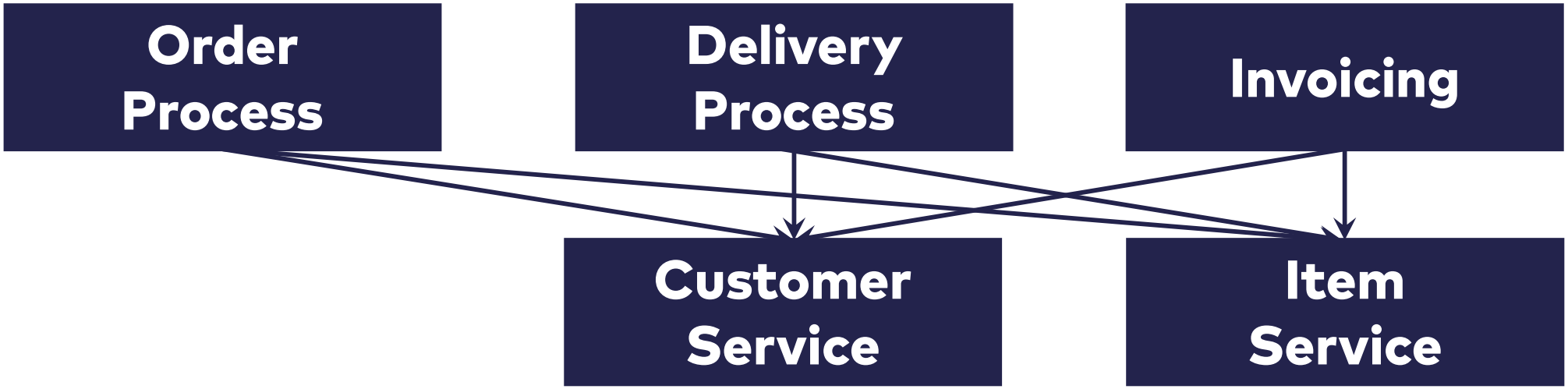
"Model each domain object as a microservice!"

Entity Service



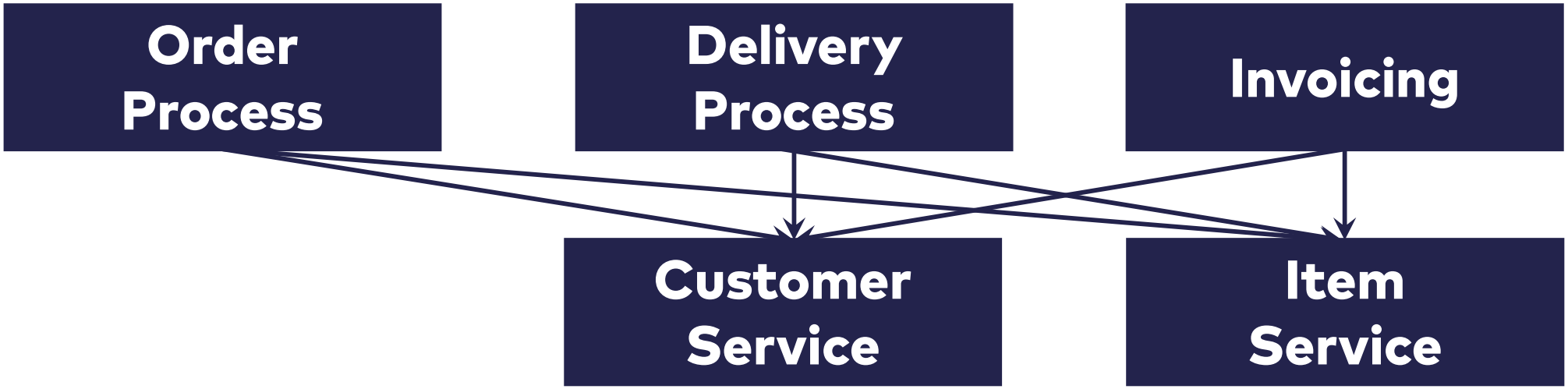
Entity Service

- Can easily become a centralized data model



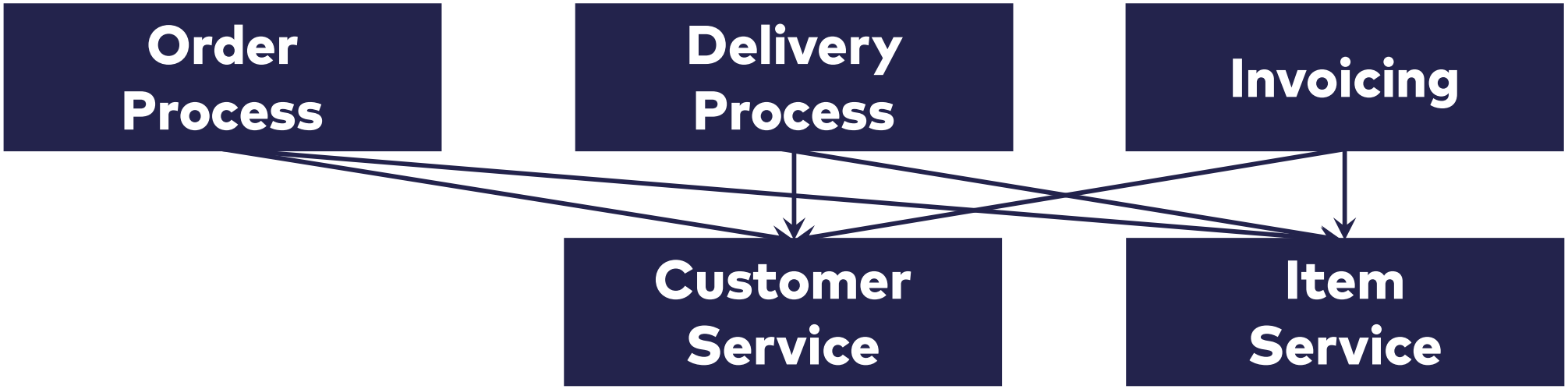
Entity Service

- Synchronous calls



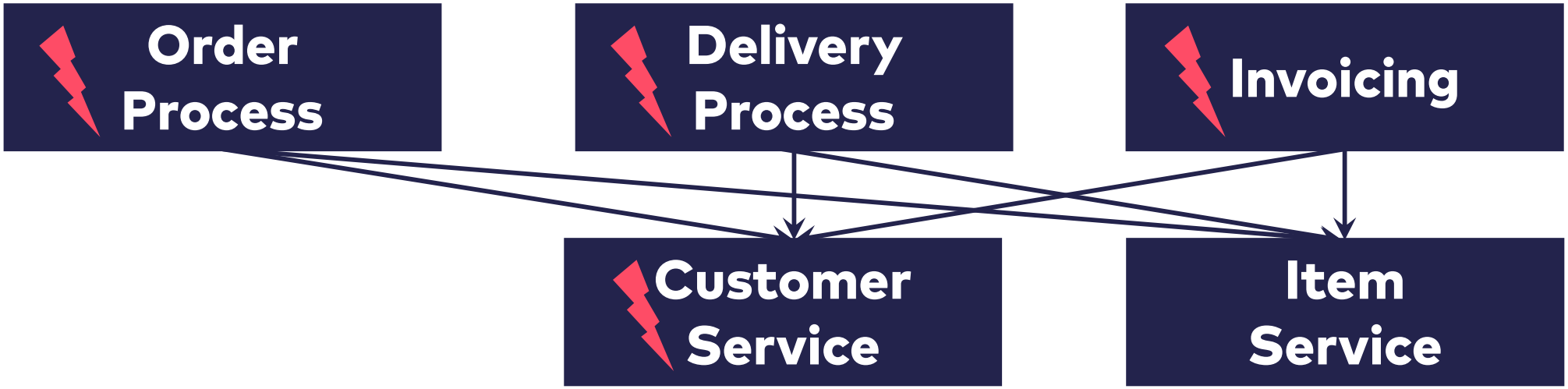
Entity Service

- Every call goes through three services.
- Performance
- Latency

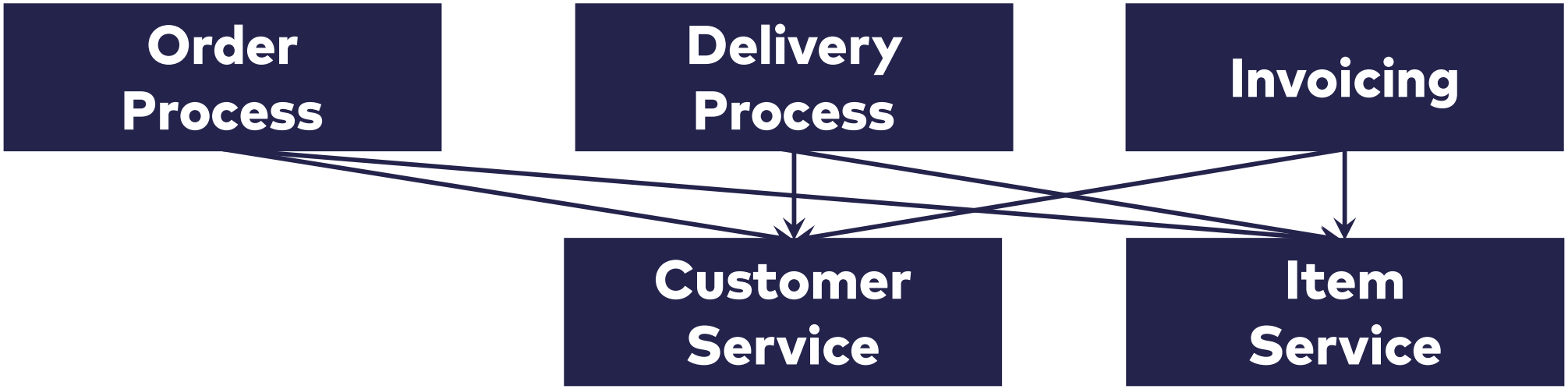


Entity Service

- Failure can easily propagate.
- Flaky services

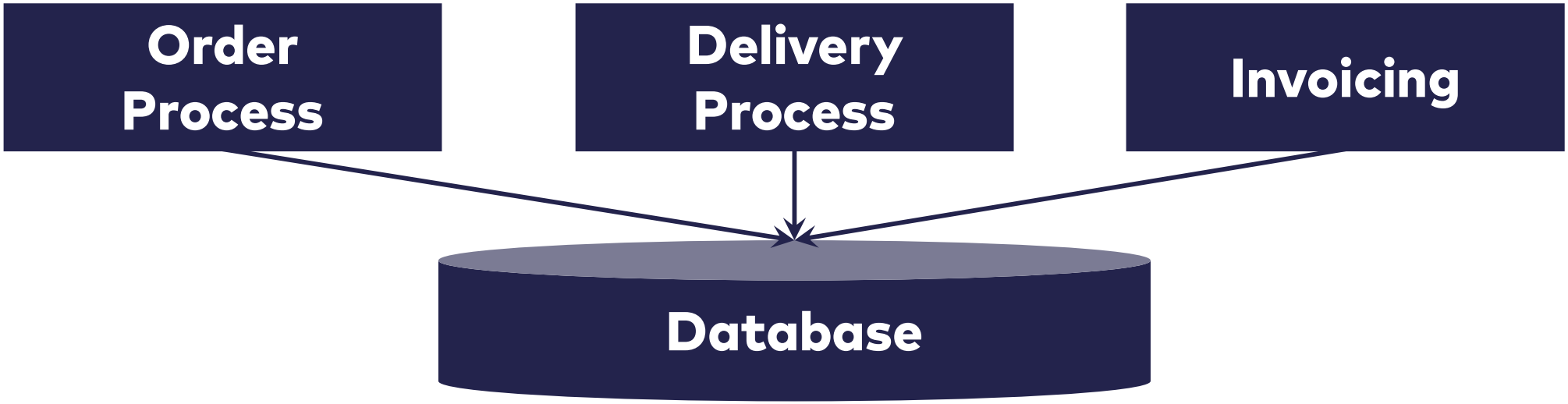


Entity Service



Common Database

- Might be a centralized data model
- Performance / latency not an issue
- Shouldn't be flaky.



Entity Service: Cure

Order Process

- Customer for Order
- Item for Order

Delivery Process

- Customer for Delivery
- Item for Delivery

Invoicing

- Customer for Invoicing
- Item for Invoicing

Entity Service: Cure

- Microservices should have their own data model = Domain-driven Design's Bounded Context
- Might share a database … but with separate schemas

Operations

"Why do you need so many servers? Do you have any clue about software architecture?"

Operations: Challenge

- Must be able to deploy
- … and operate many microservices
- … and other new technologies.
- Existing technologies might not fit
- Processes and people might not support the challenge.

Operations: Cure

- Problem well-known
- Problem obvious up-front
- Don't do microservices
- Might be a valid trade-off

Operations: Cure

- Install and use new technologies

... only if needed.

- No technology fetish, please!

Operations: Cure

- Introduce a PaaS
- Install PaaS once
- Afterwards operations out of the loop
- Marketing strategy for PaaS
- PaaS = standardization
- Kubernetes is better customizable

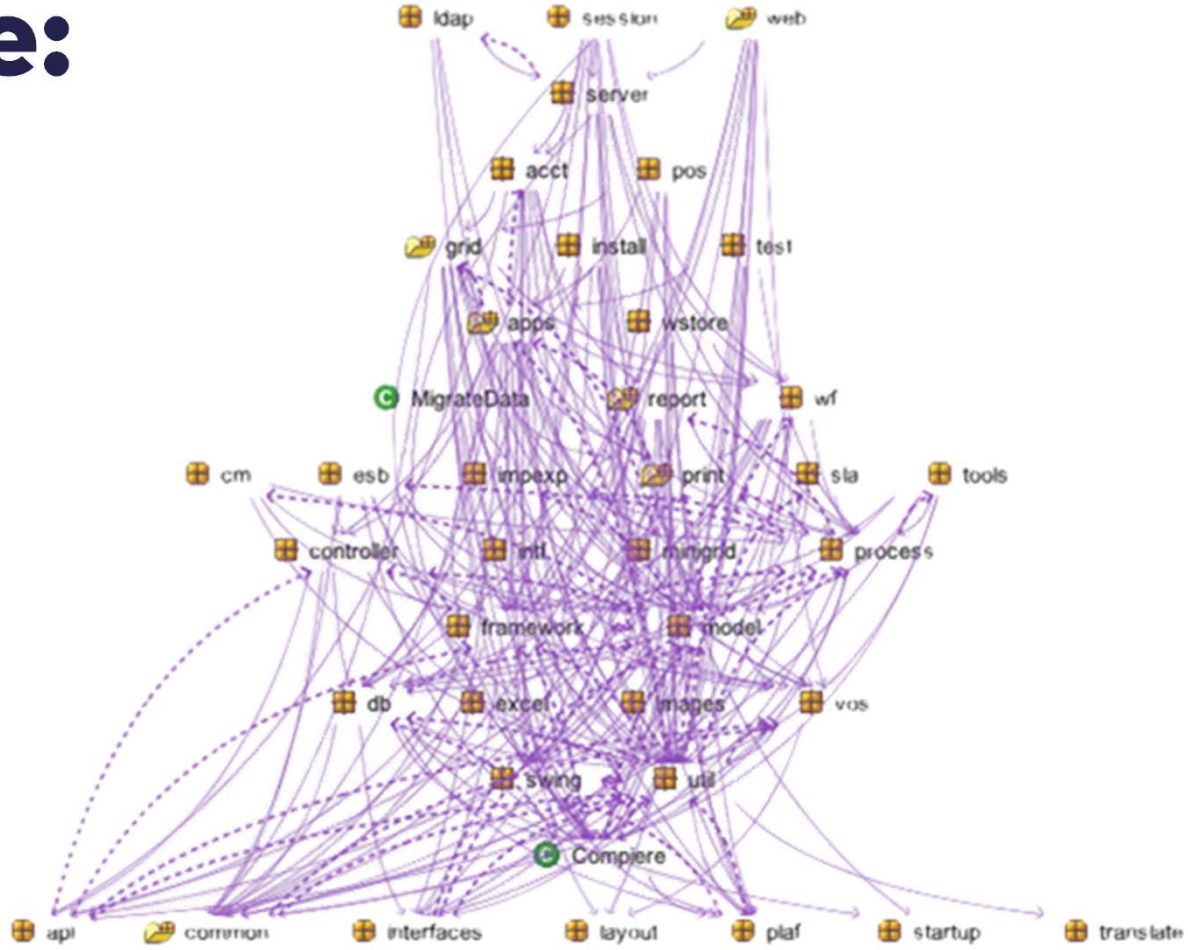
Operations: Cure

- Public Cloud
- Lots of technologies pre-packaged (e.g. Kubernetes)
- Easy to automate (e.g. reboot if machine fails)
- … so easier to support many services
- Operations out of the loop

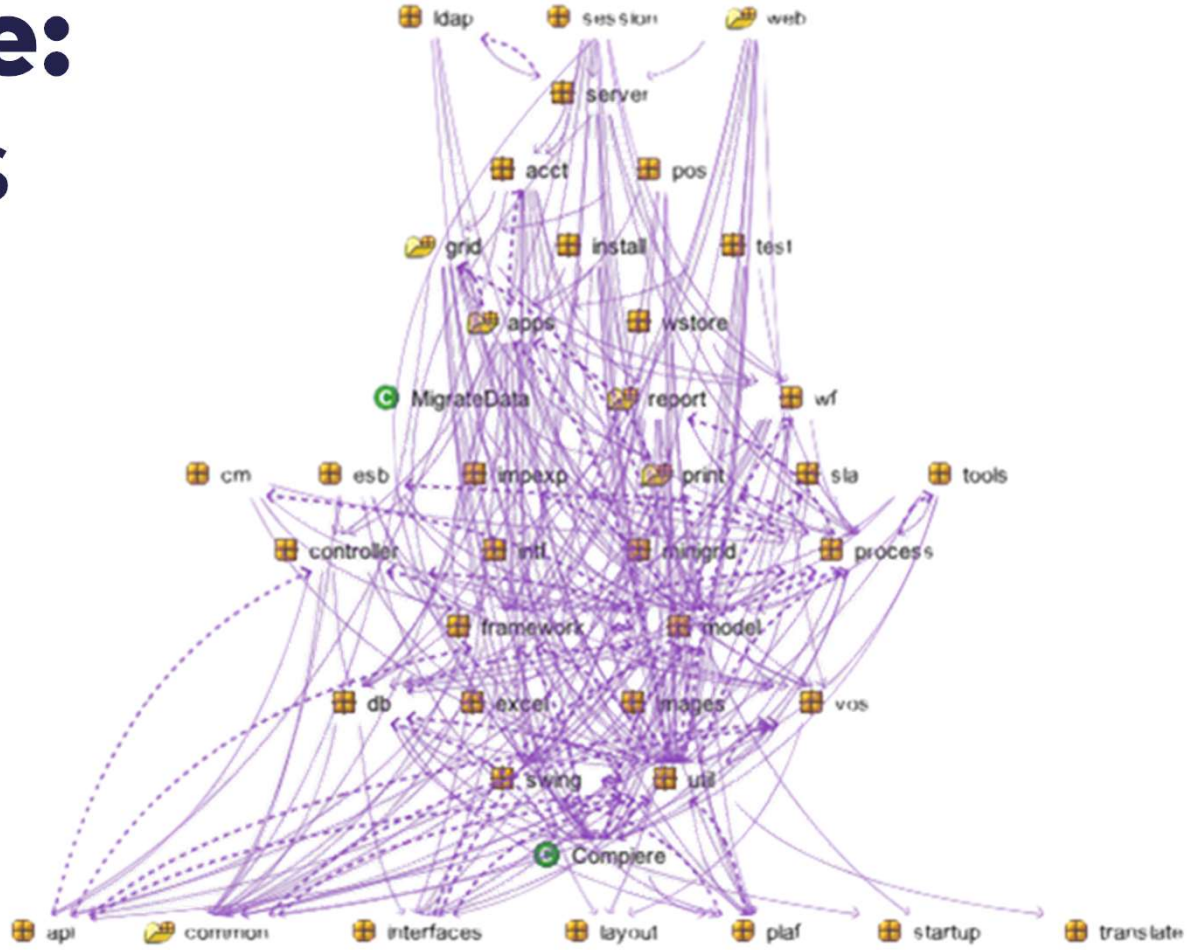
Bad Structure

"The system is flexible and maintainable – because we use microservices!"

Bad Structure: Deployment Monolith



Bad Structure:
Microservices



Bad Structure

- Microservices are just different modules.
- Microservices won't fix modularization
- Distributed Ball of Mud

Bad Structure: Challenge

Microservices' extreme decoupling becomes a problem:

- Multiple coordinated deployments
- Architecture firewalls might make bigger changes hard
- Chatty microservice cause problems for performance

... and latency

... resilience

Bad Structure: Cure

- Decouple logic
- Bounded context: Domain model per microservice
- Less communication
- Migrate by bounded context
- Don‘t reuse the existing structure for migration!

If you want to fix the structure,
microservices won't help.

If you want to fix the structure,
fix the structure.

Organization

"Architects will decide. The teams are just not up to the challenge ☹"

Organization: Challenge

- Leap of faith: Empower teams
- If you trust people, they behave differently.
- Dev works differently if code goes to prod and not QA...

Organization: Cure

- Microservices enable independent teams

... independent technologies

... independent parts of the domain

- Centralized decisions = no independent teams
- Reduces the benefit of microservices

Fashion

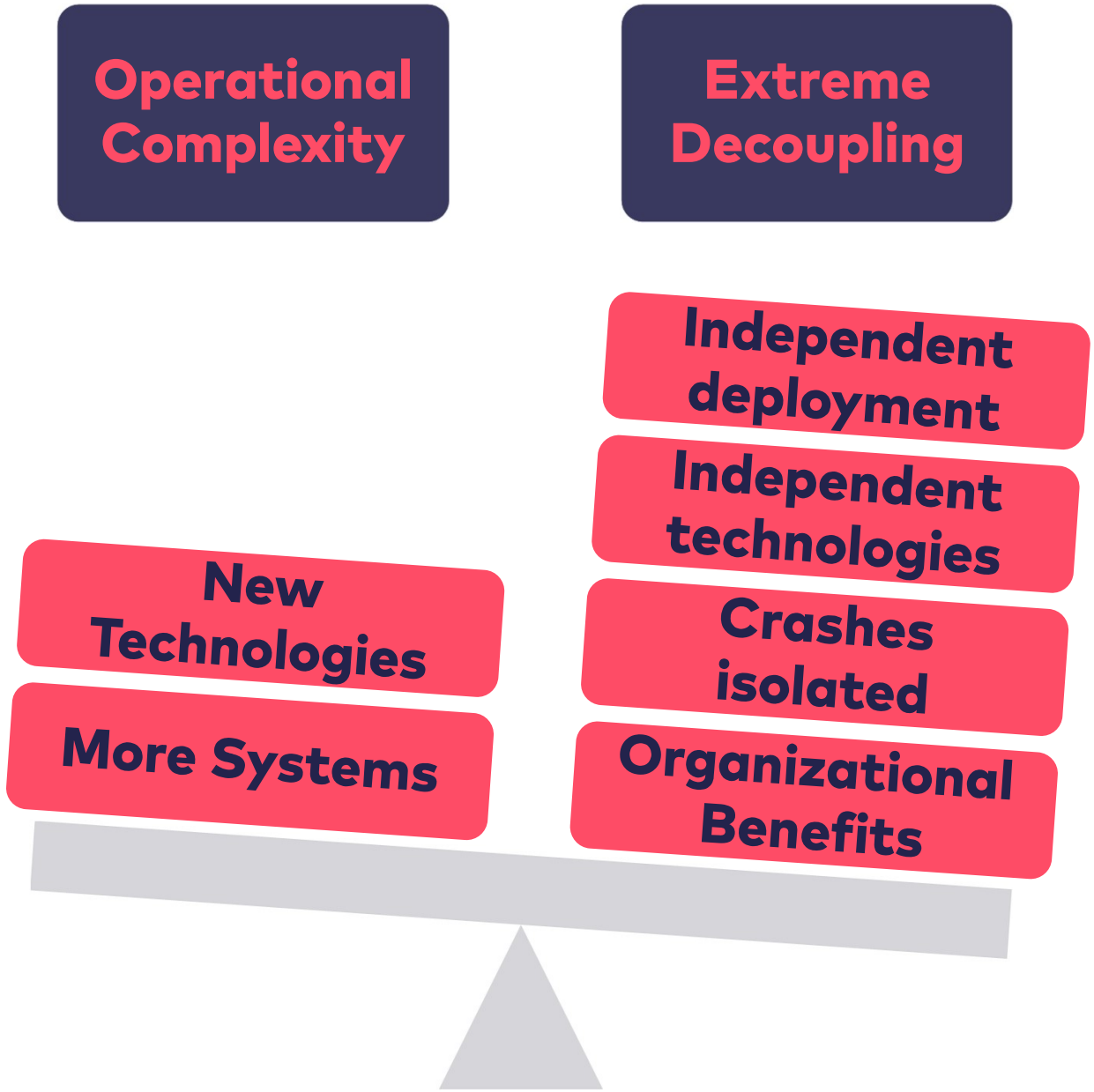
"Microservices is how you build systems nowadays!"

Fashion: Challenge

- Microservices are a trade-off
- If you don't reap the benefits …
  … you still get the challenges
- Many different architecture possible
- Software architecture = find the best trade-off

Fashion: Cure

- Decide about the trade-off!
- Choose other options, if needed.
- Deployment monoliths are still an option.


Operational Complexity
Extreme Decoupling
Independent deployment
Independent technologies
Crashes isolated
Organizational Benefits
New Technologies
More Systems


OMG!
memegenerator.net

OMG

- We do microservices

... but we deploy once each quarter

... all microservices at once

... with a common technology stack

- Why do you do microservices????
- No benefits


Conclusion

The problem is not microservices.
The problem is the right trade-off.