Der perfekte Microservice

Size does(n’t) matter!

Lars **Röwekamp**
CIO New Technologies

#**wissen**teilen

@**mobile**Larson
@_**open**Knowledge

Microservices

Monolithic Design

Microservice Design

Surviving Service Design


„Monolithic **Design**”

Was ist das **Problem**?

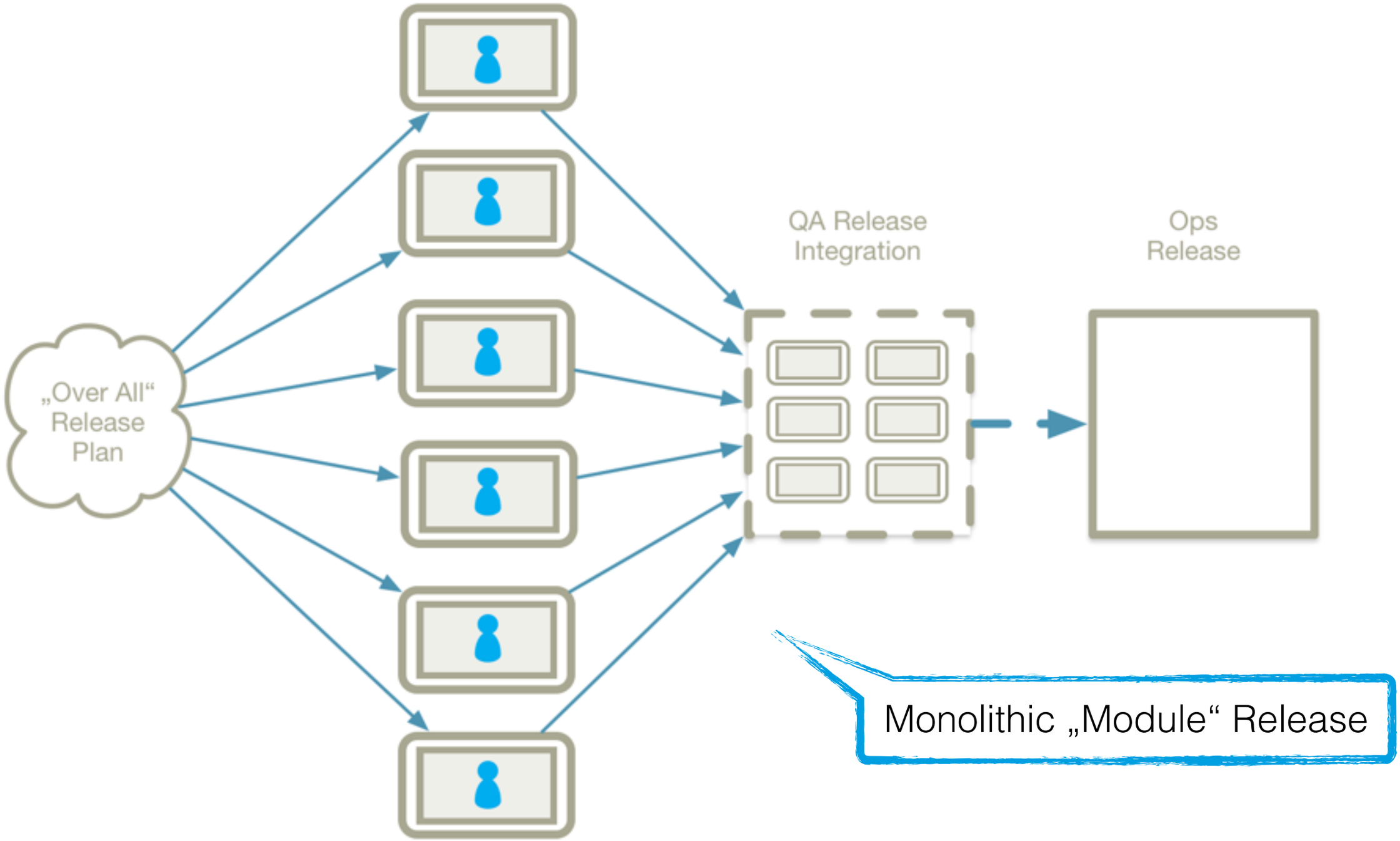
- Umsetzung von Features dauert zu lange
- Architektur-Qualität nimmt ab (an Bedeutung)
- Technologische „Schulden“ sind bekannt
- „-ility“ Probleme wohin man schaut
- Replacement/Refactoring ist zu teuer

- Deployment ist kompliziert und dauert lang
- Skalierung hat Limit erreicht

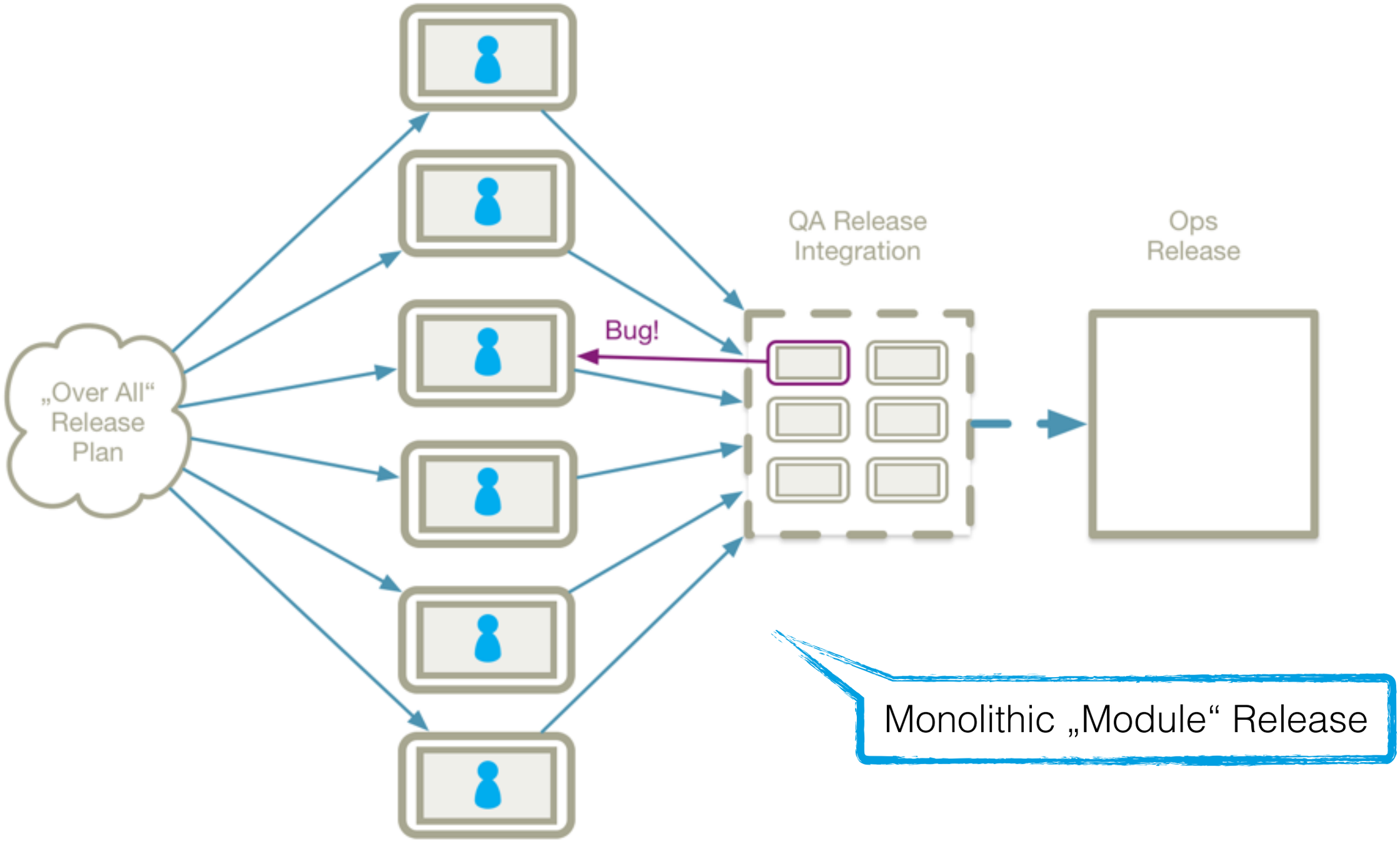
Was ist das **Problem**?

„For a monolithic to change
all must agree on each
change.
Each change has
unanticipated effects
requiring testing beforehand.“

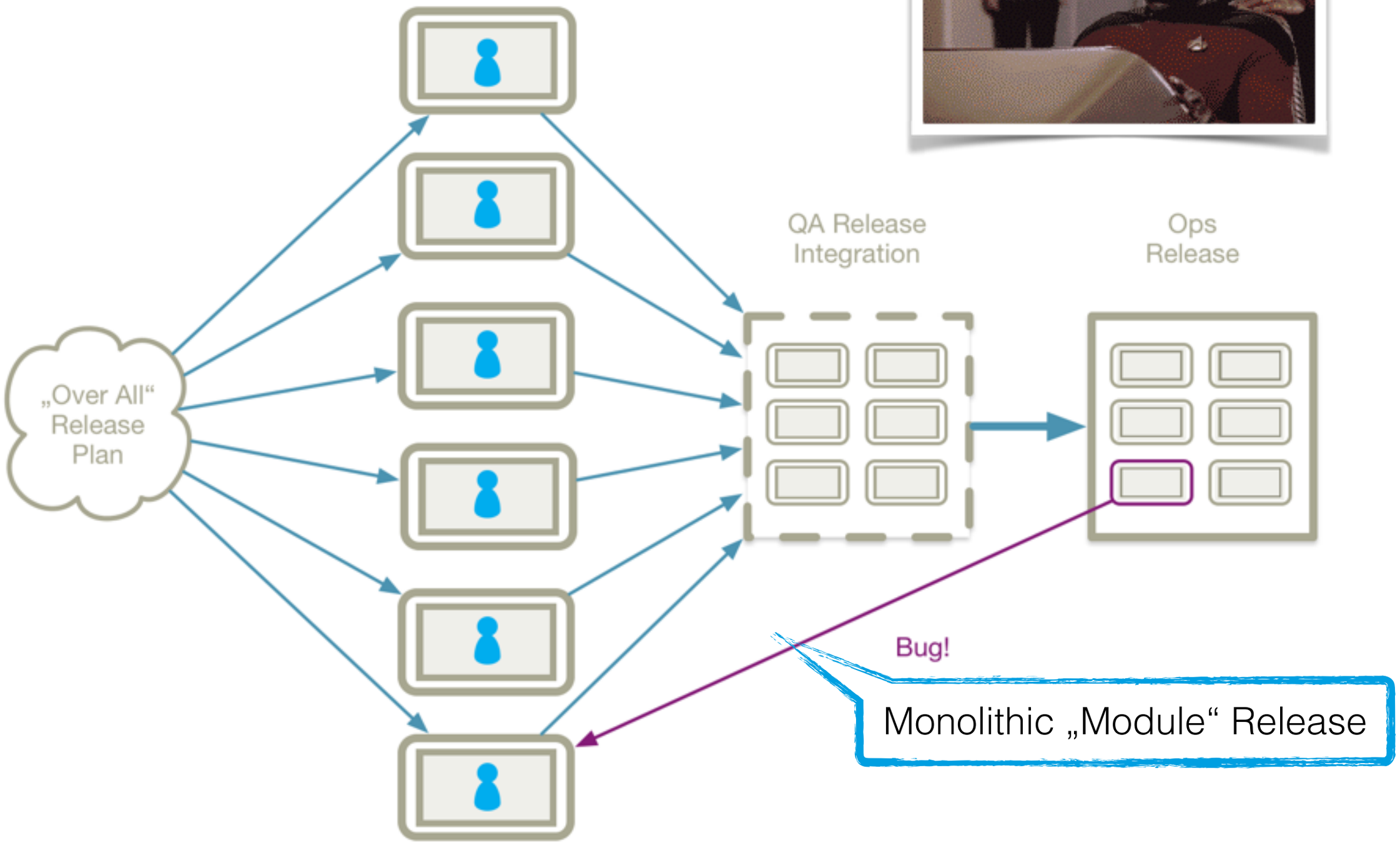
Was ist das **Problem**?



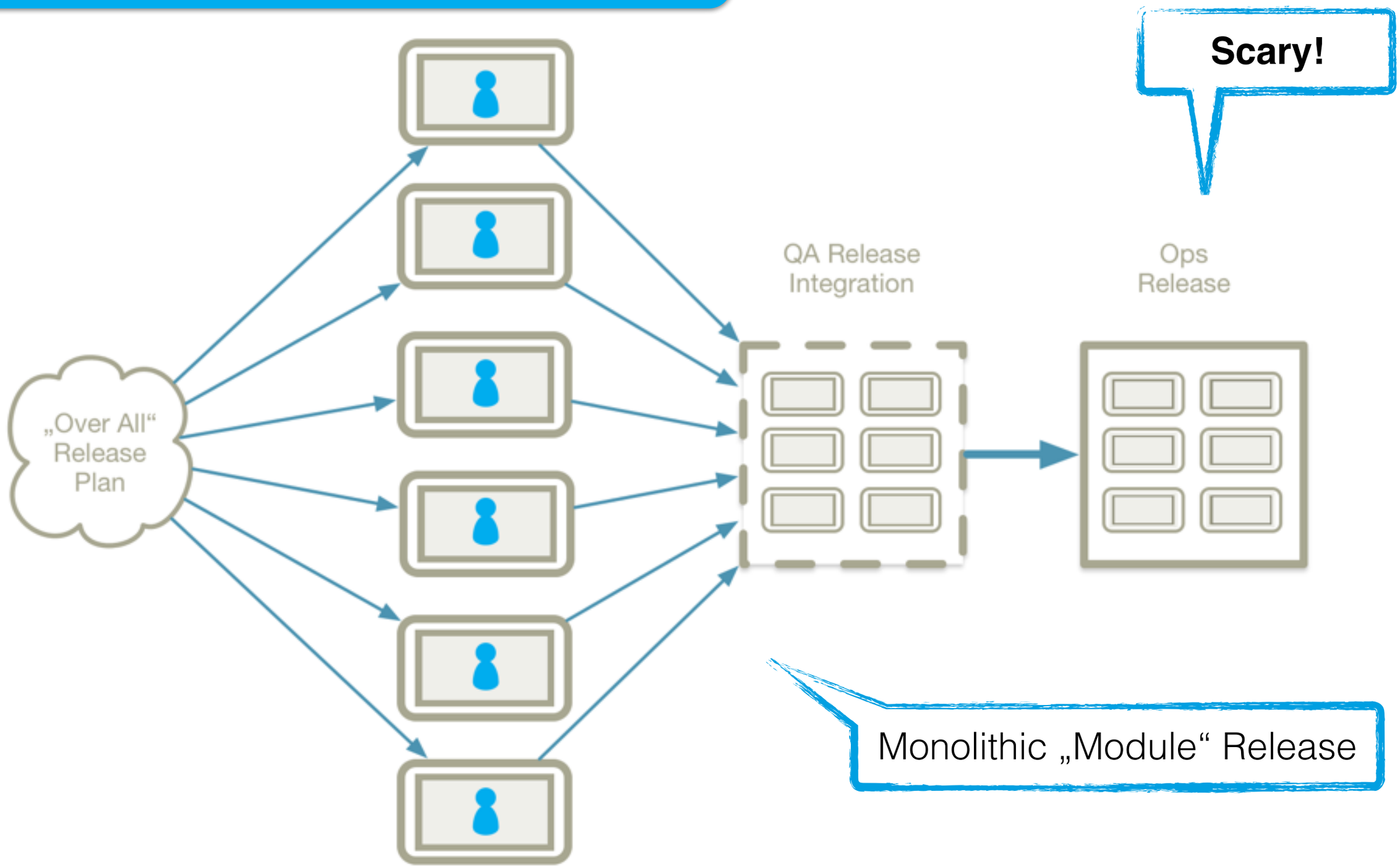
Was ist das **Problem**?



Was ist das **Problem**?

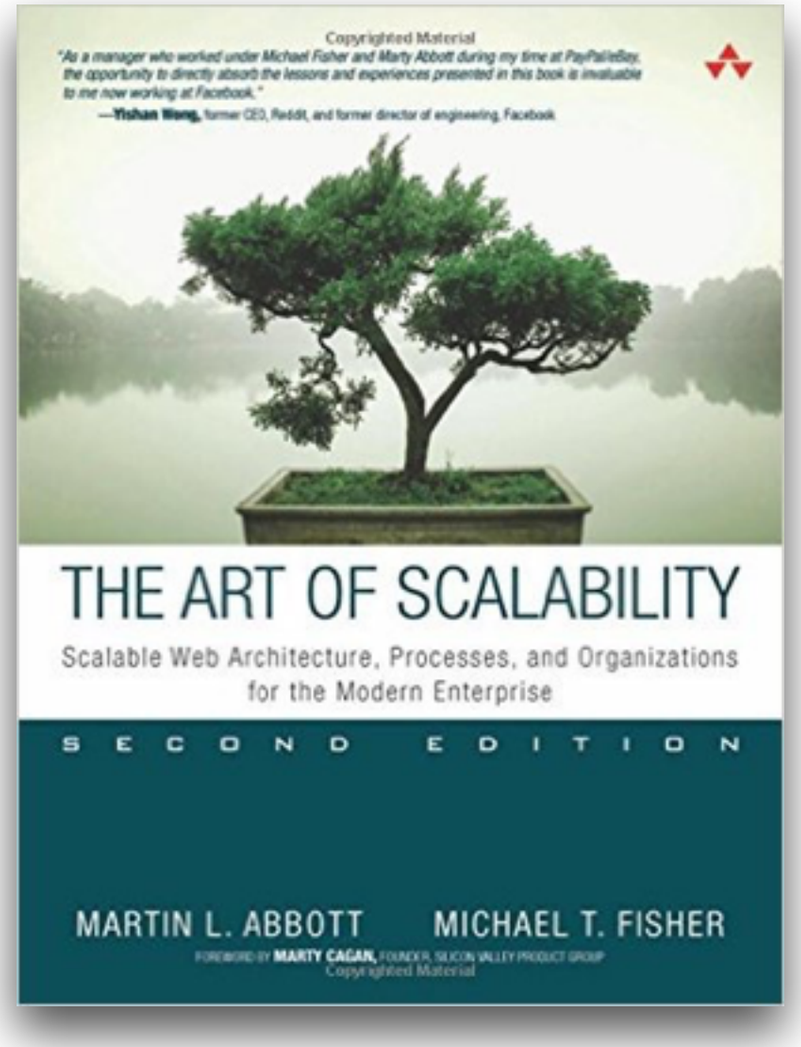


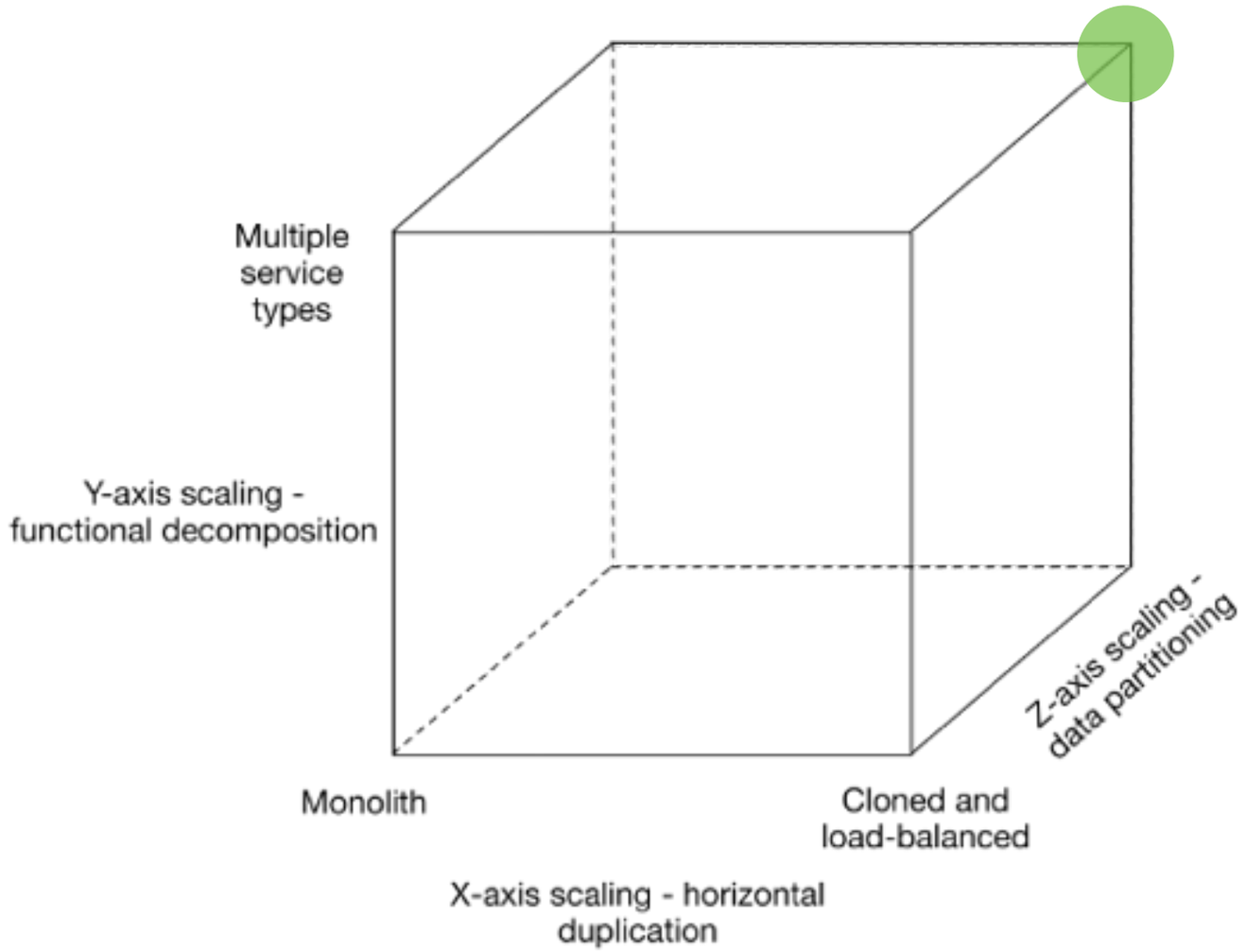
Was ist das **Problem**?



Was ist das **Problem**?

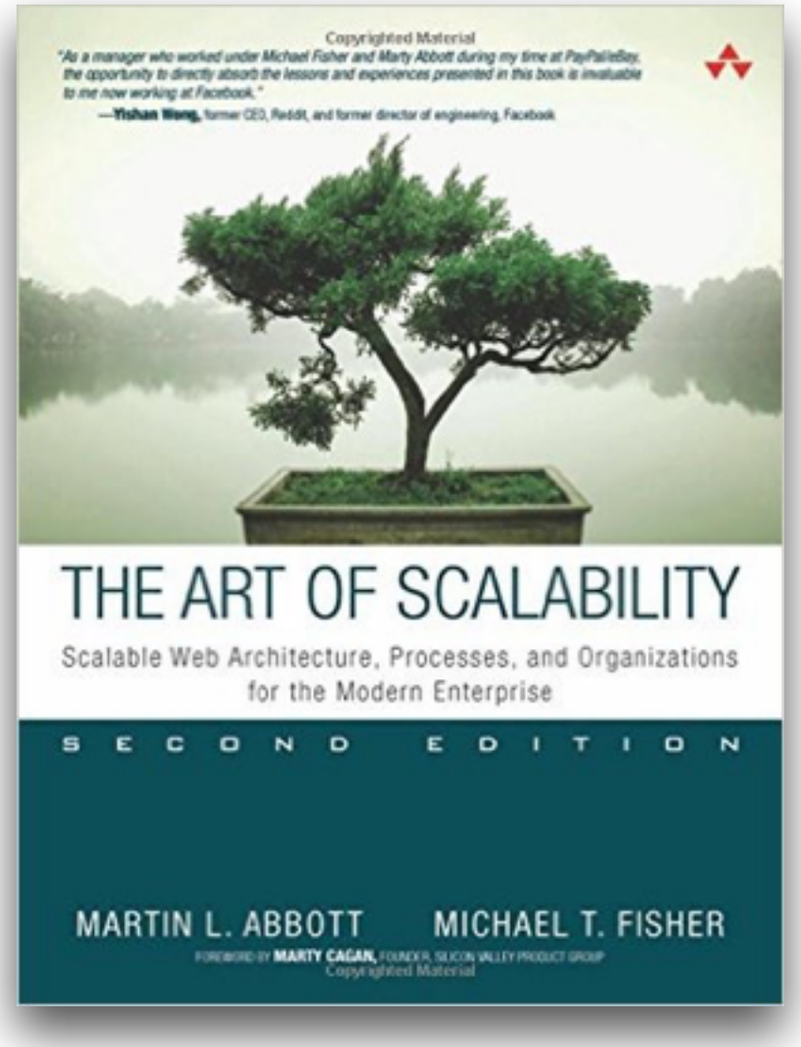
„You want to be **here**.“

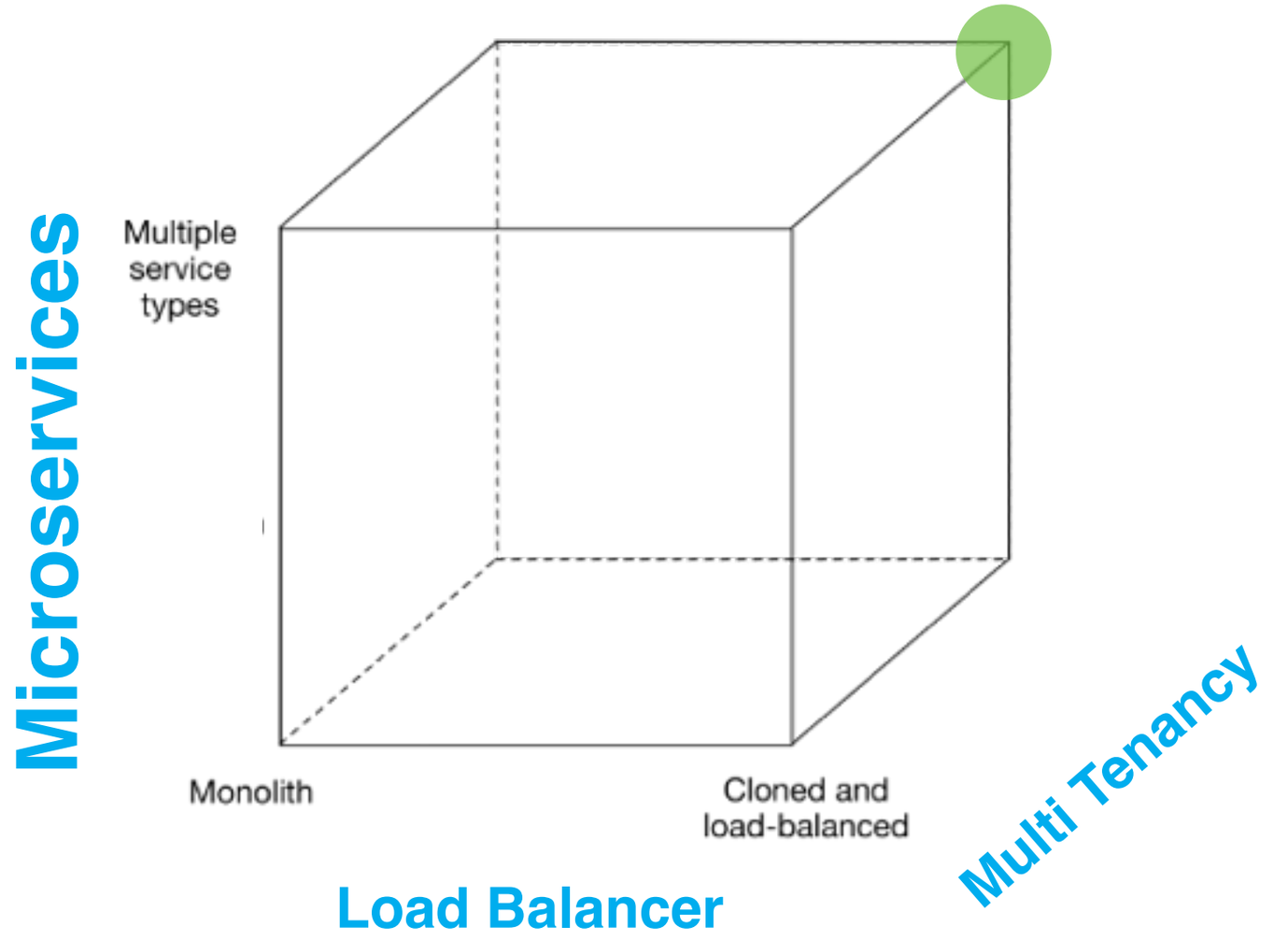




Was ist das **Problem**?

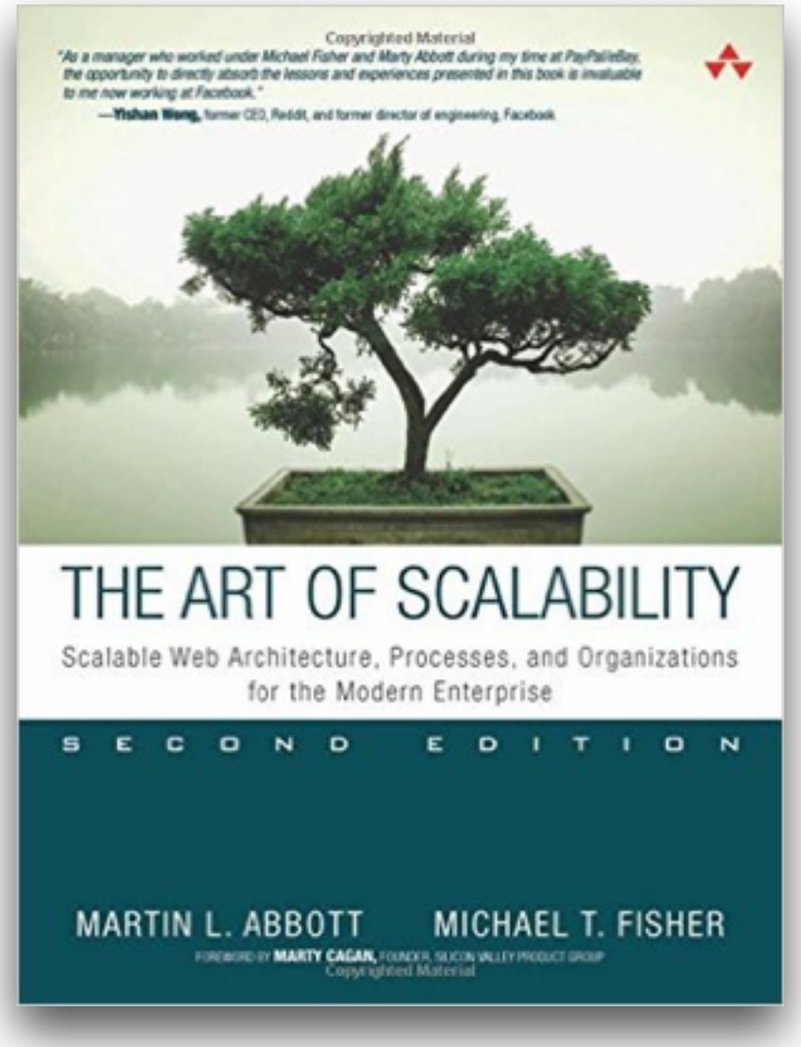
„You want to be **here**.“



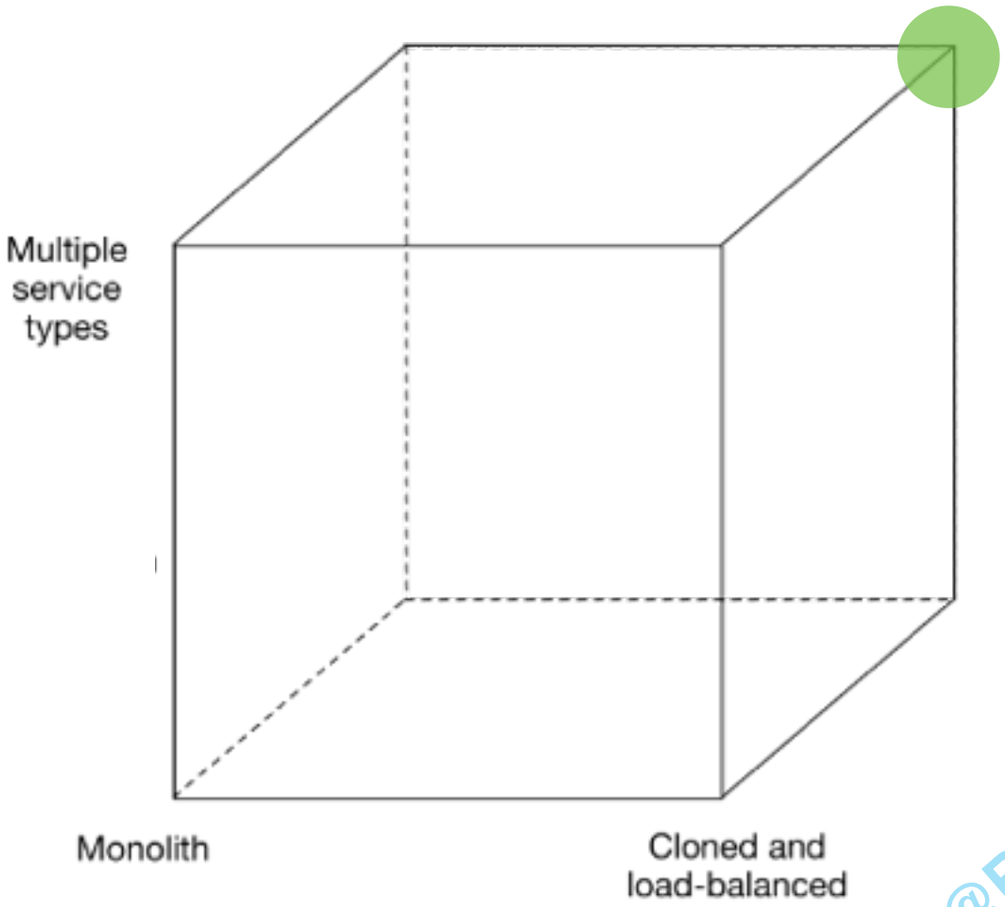


Was ist das **Problem**?

„You want to be **here**.“



a.k.a. „Time to Market“



@Runtime

@Runtime

„**Help**; needed it is.”

Microservices

Monolithic Design

Microservice Design

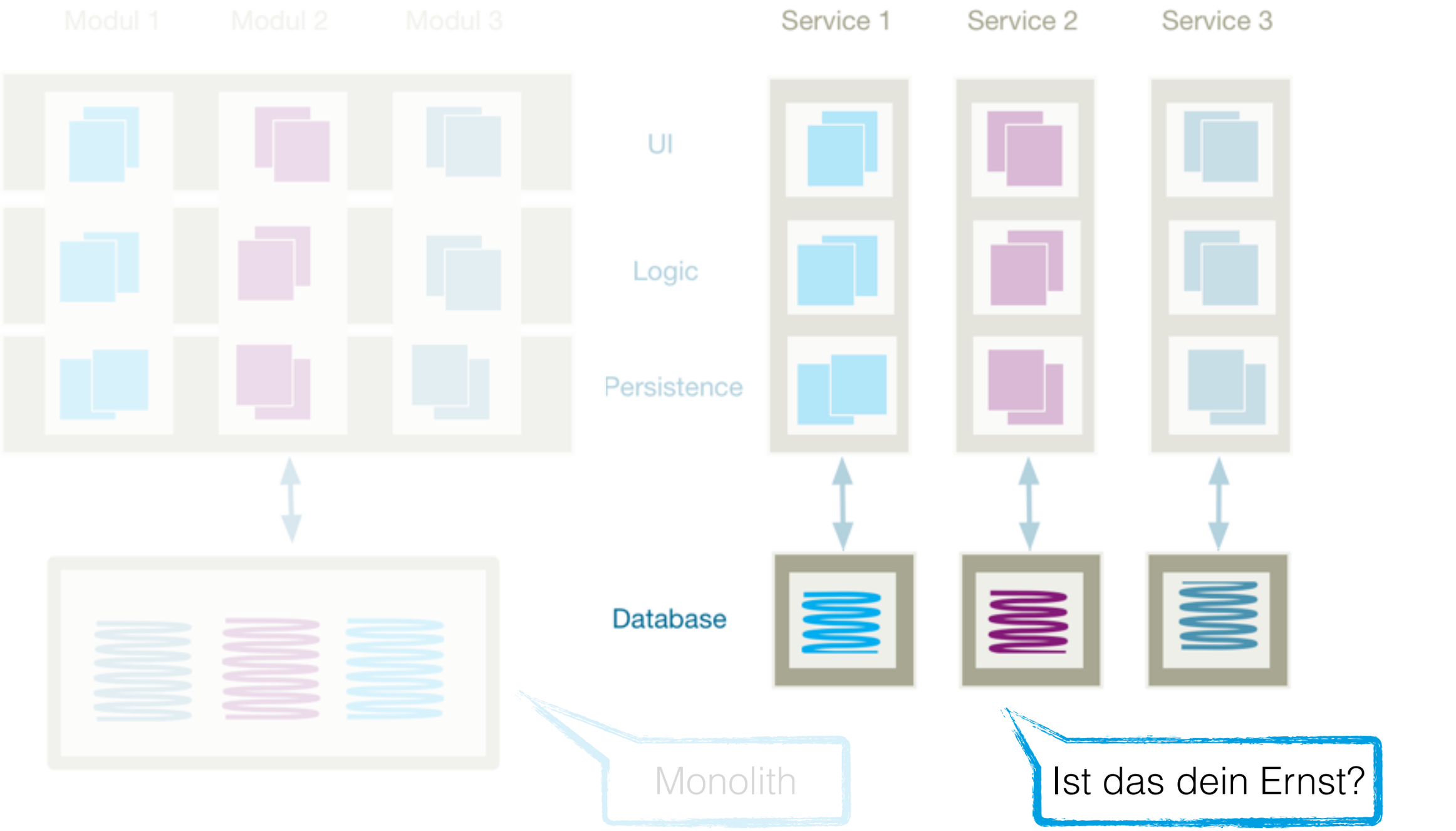
Surviving Microservice Design

„Kind of Definition“

„In short, the microservice architectural style is **an approach to developing a single application as a suite of small services**, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.”

„As well as the fact that services are **independently deployable and scalable**, each service also provides a **firm module boundary**, even allowing for different services to be written in **different programming languages**. They can also be **managed by different teams**.”

Microservice Design



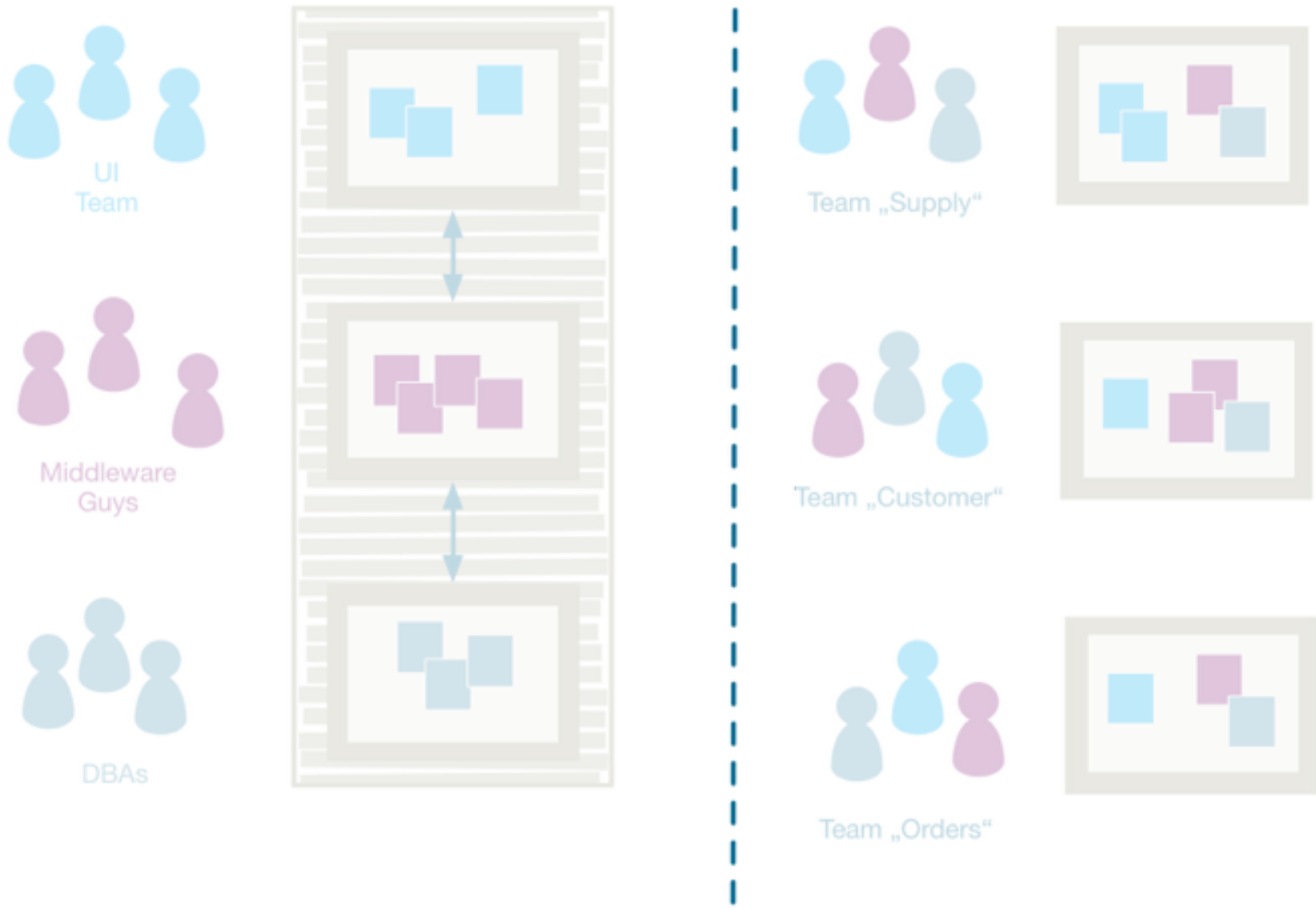
Microservice Design

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

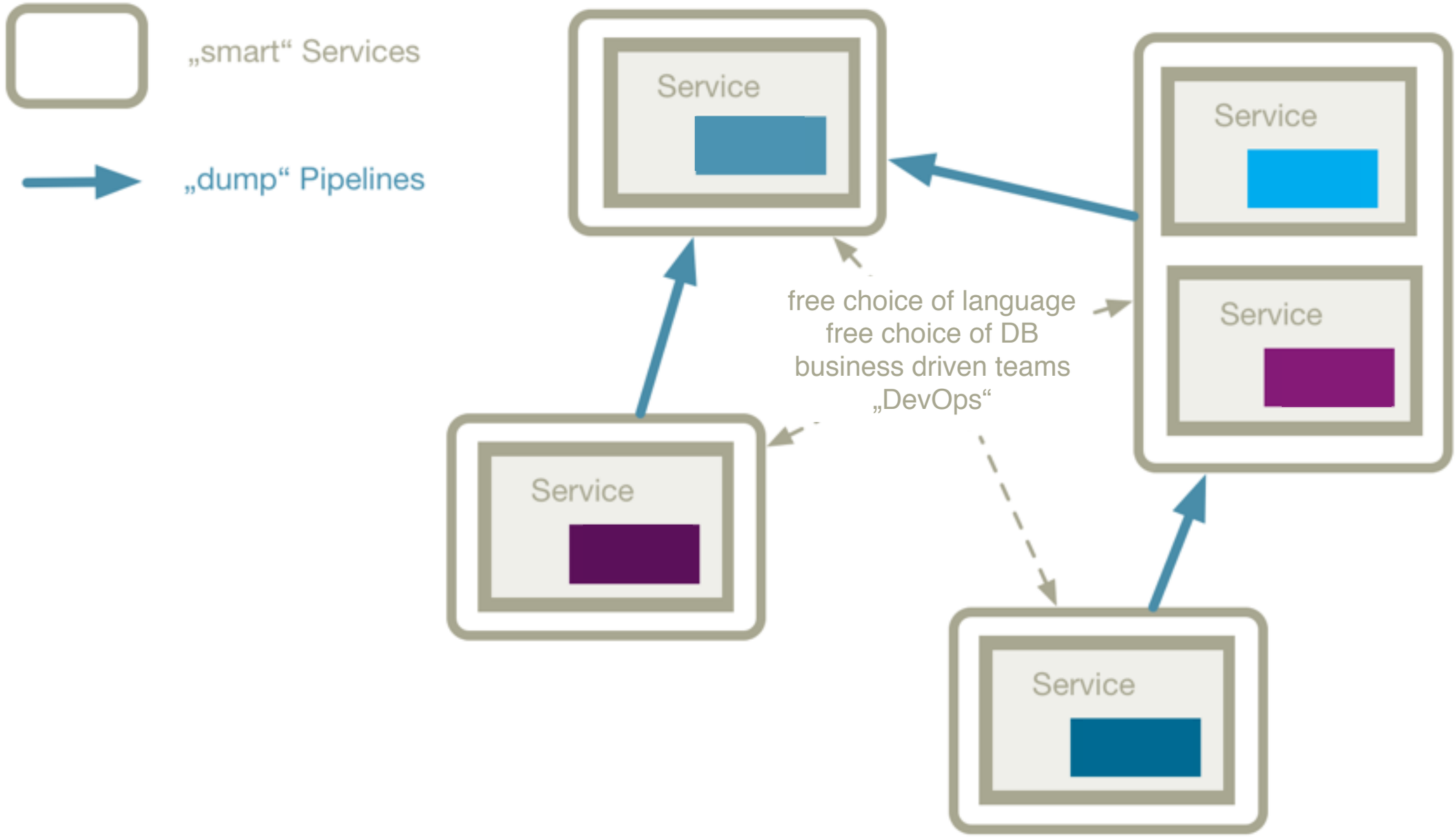
Conway’s Law (1967)

Microservice Design

Conway's Law



Microservice Design



Microservice Design

- Wartbarkeit
- Erweiterbarkeit
- Austauschbarkeit
- Skalierbarkeit
- Zuverlässigkeit & Ausfallsicherheit
- Fehlertoleranz
- „Time to Market“

Klingt gut, oder?

Microservices

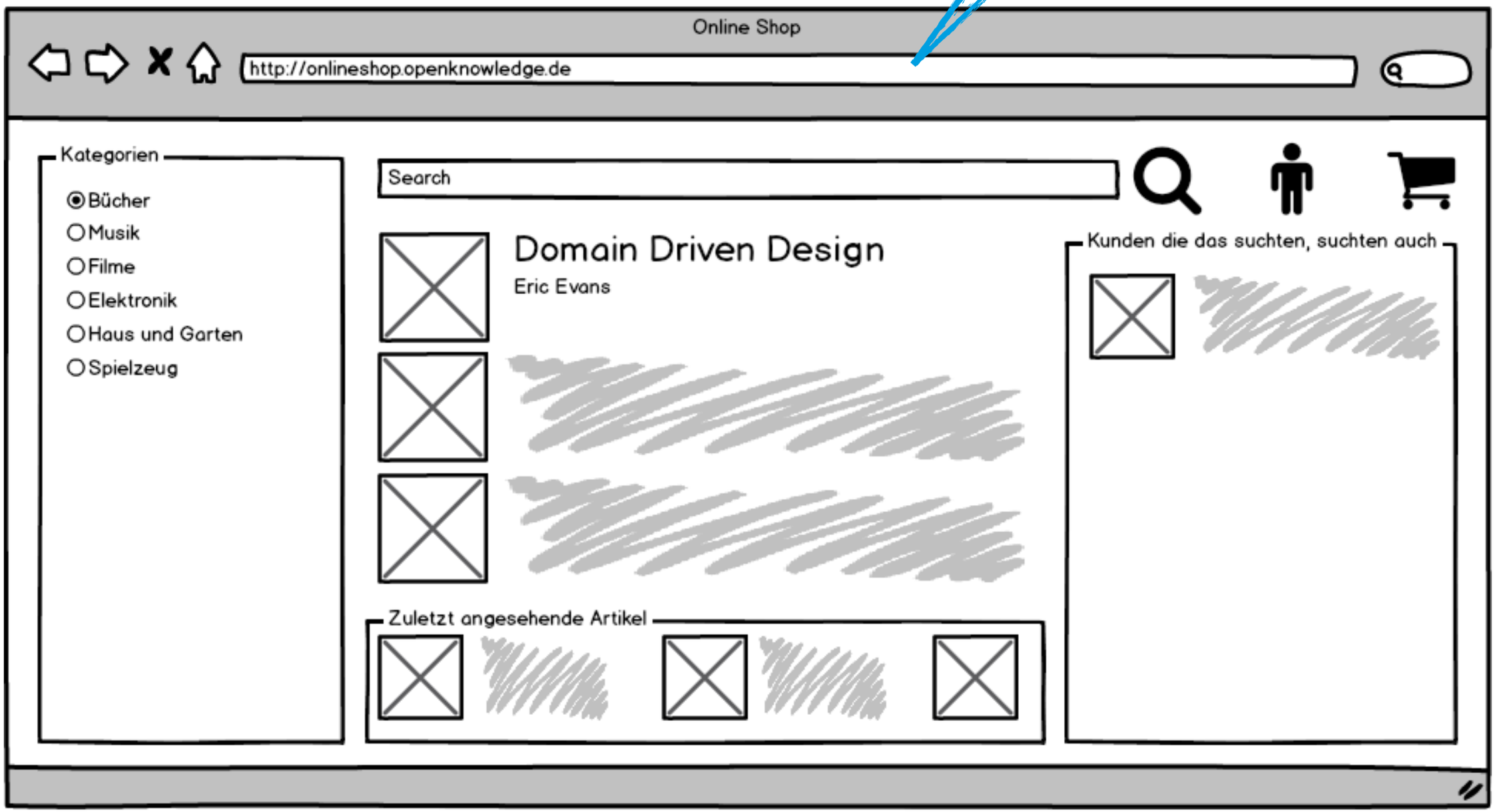
Monolithic Design

Service Design

Surviving Service Design

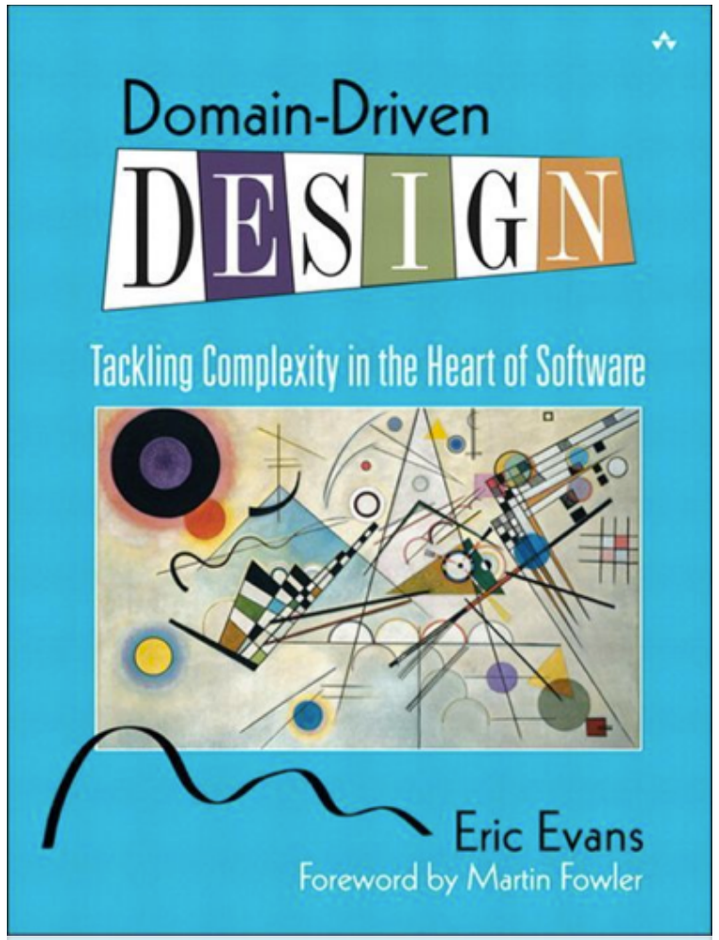
Surviving Service Design

Microservices?



Surviving Service Design

Surviving Service Design

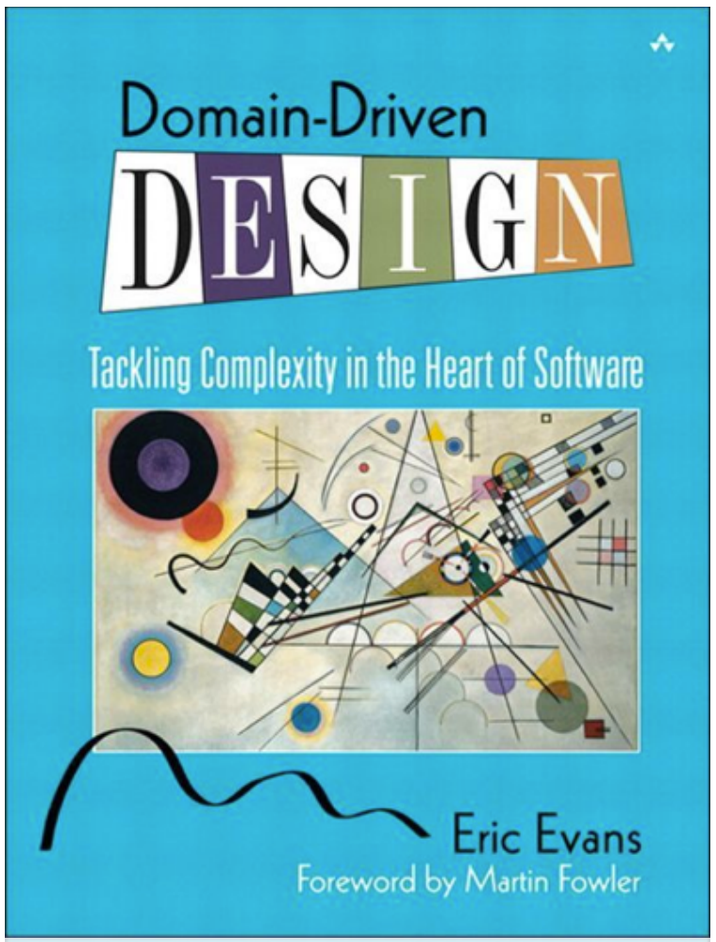


„For most software projects, the primary **focus** should be on the **domain** and **domain logic**.“

Eric Evans, 2004

Surviving Service Design

Abstraktion der realen Welt!

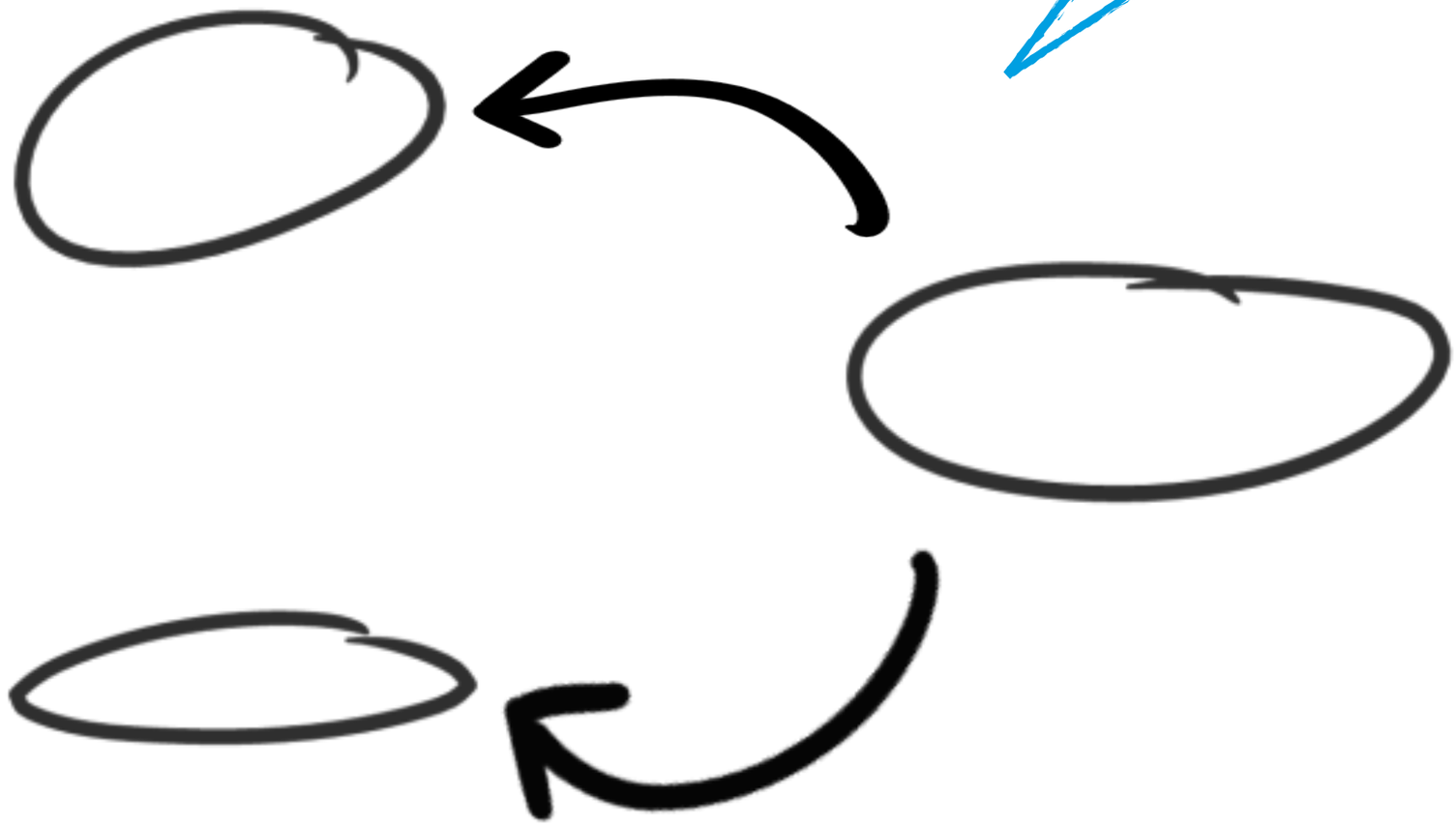


„Complex domain design should be based on a **model**.“

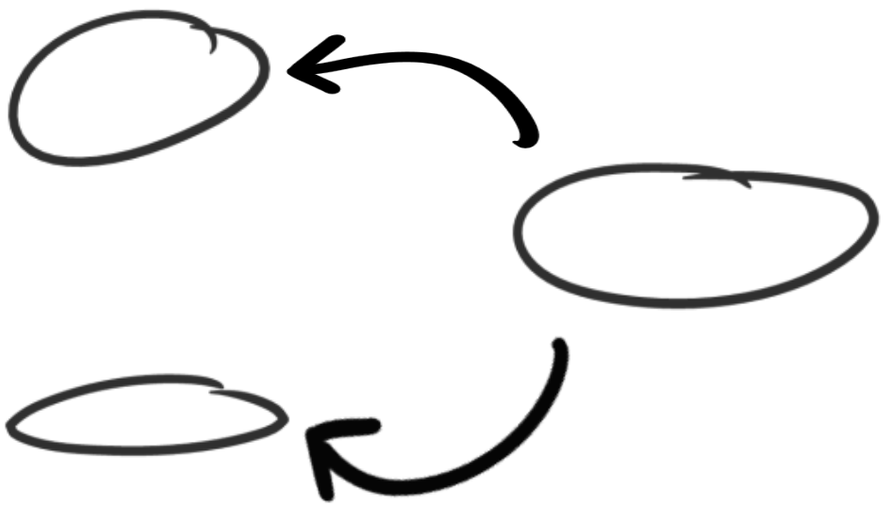
Eric Evans, 2004

Surviving Service Design

Und was ist **unser** Domänenmodell?



Surviving Service Design

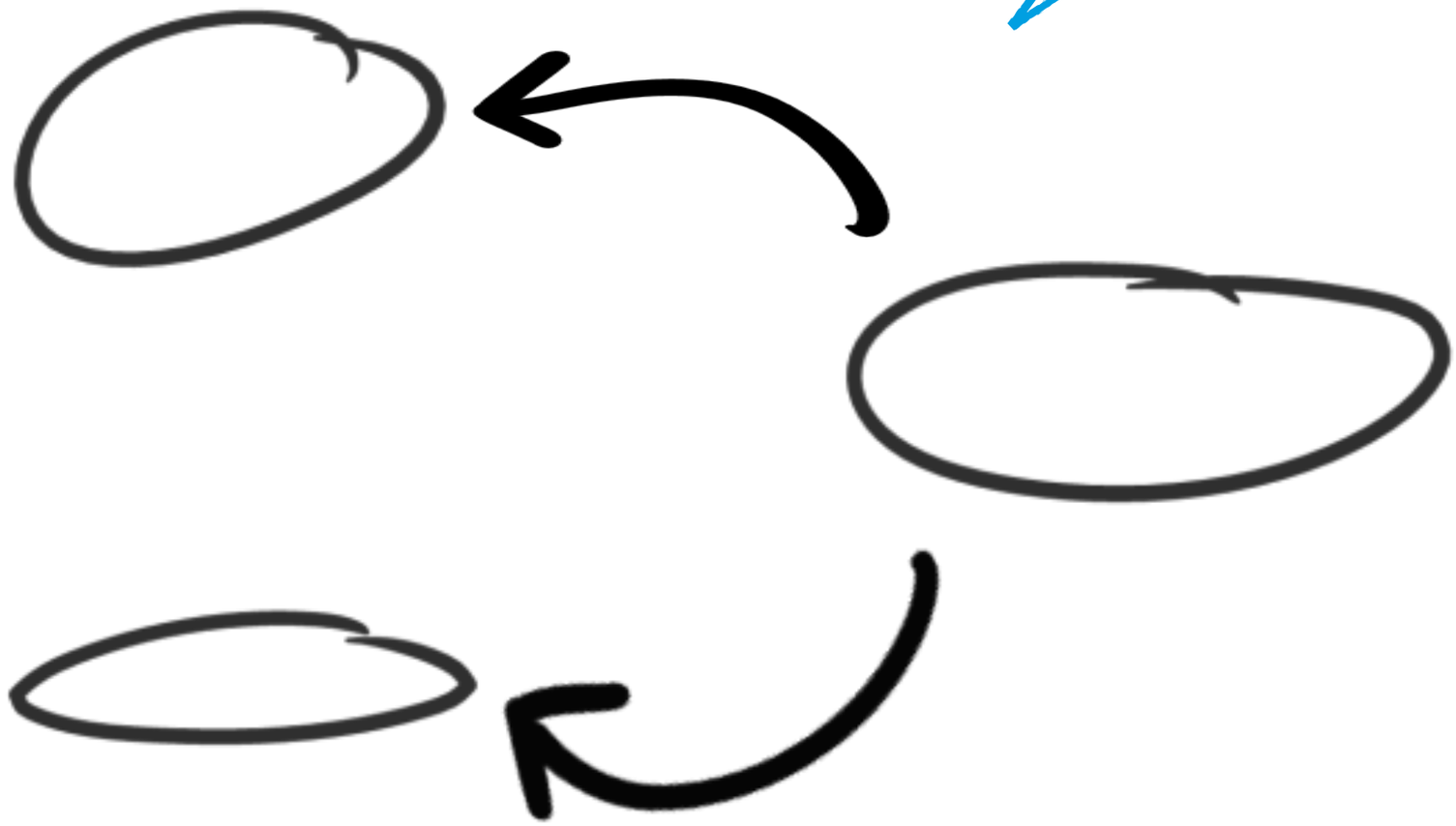


Domänenmodell

- **Ableiten** der Objekt(namen) aus gemeinsamer Sprache
- **Evolution** des Modells durch alle Beteiligten
- Änderungen in der **Sprache** sind Änderungen im **Modell!**

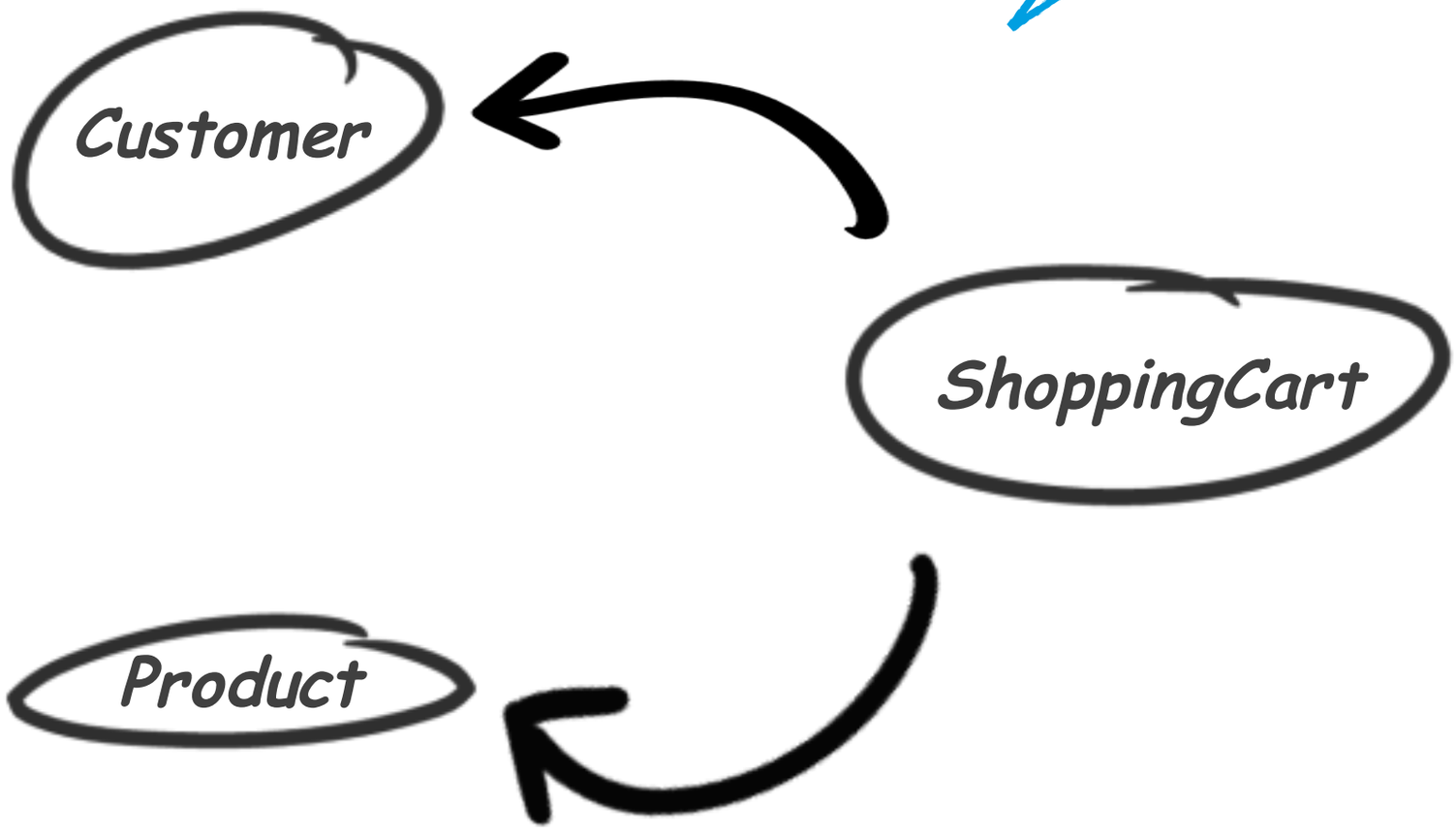
Surviving Service Design

Ein Modell
für alle?



Surviving Service Design

Ganz schön groß, oder?

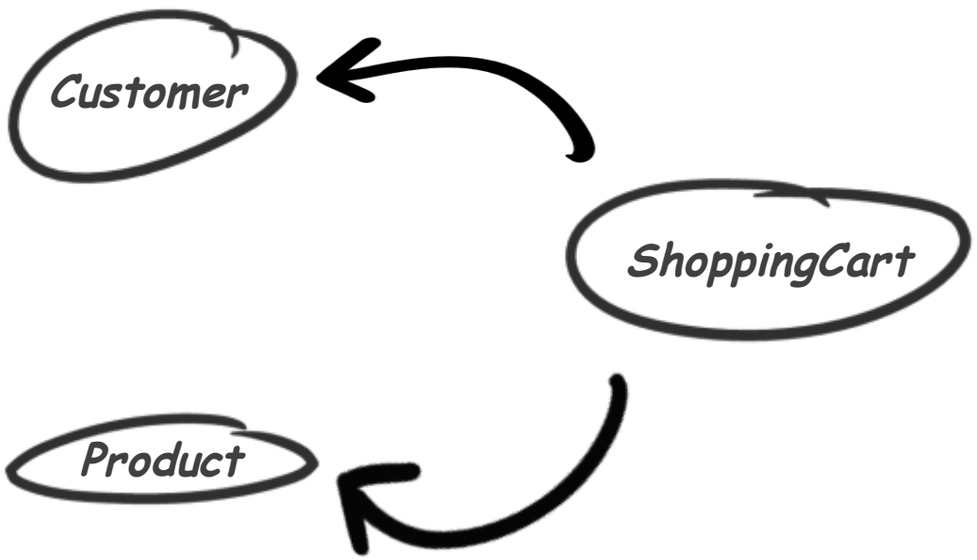


„Einheit, die von einem kleinen Team - **fachlich** und **technisch** - beherrscht werden kann.”

Oliver Wegner (OTTO)

Surviving Service Design

Surviving Service Design



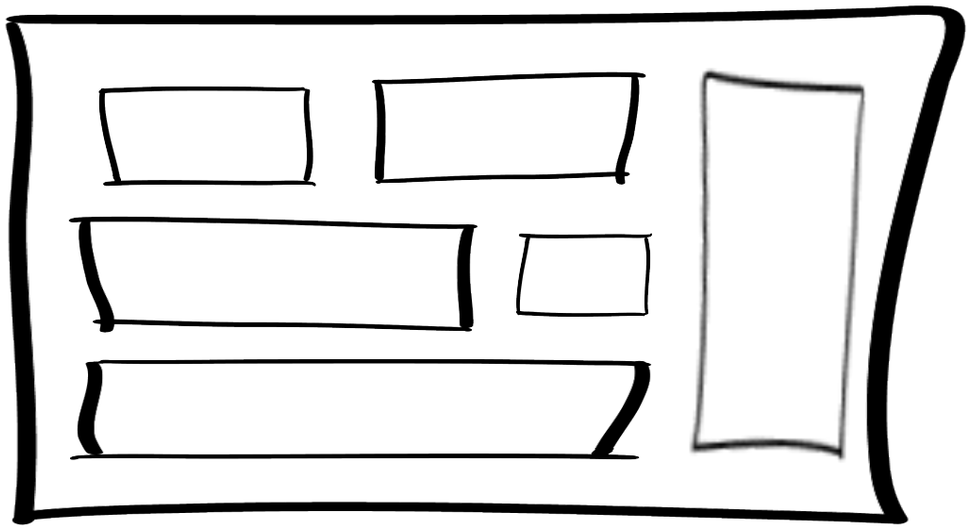
Die **Herausforderung**:

- Auftrennen ohne Verlust der Vorteile von DDD

Mögliche **Konflikte**:

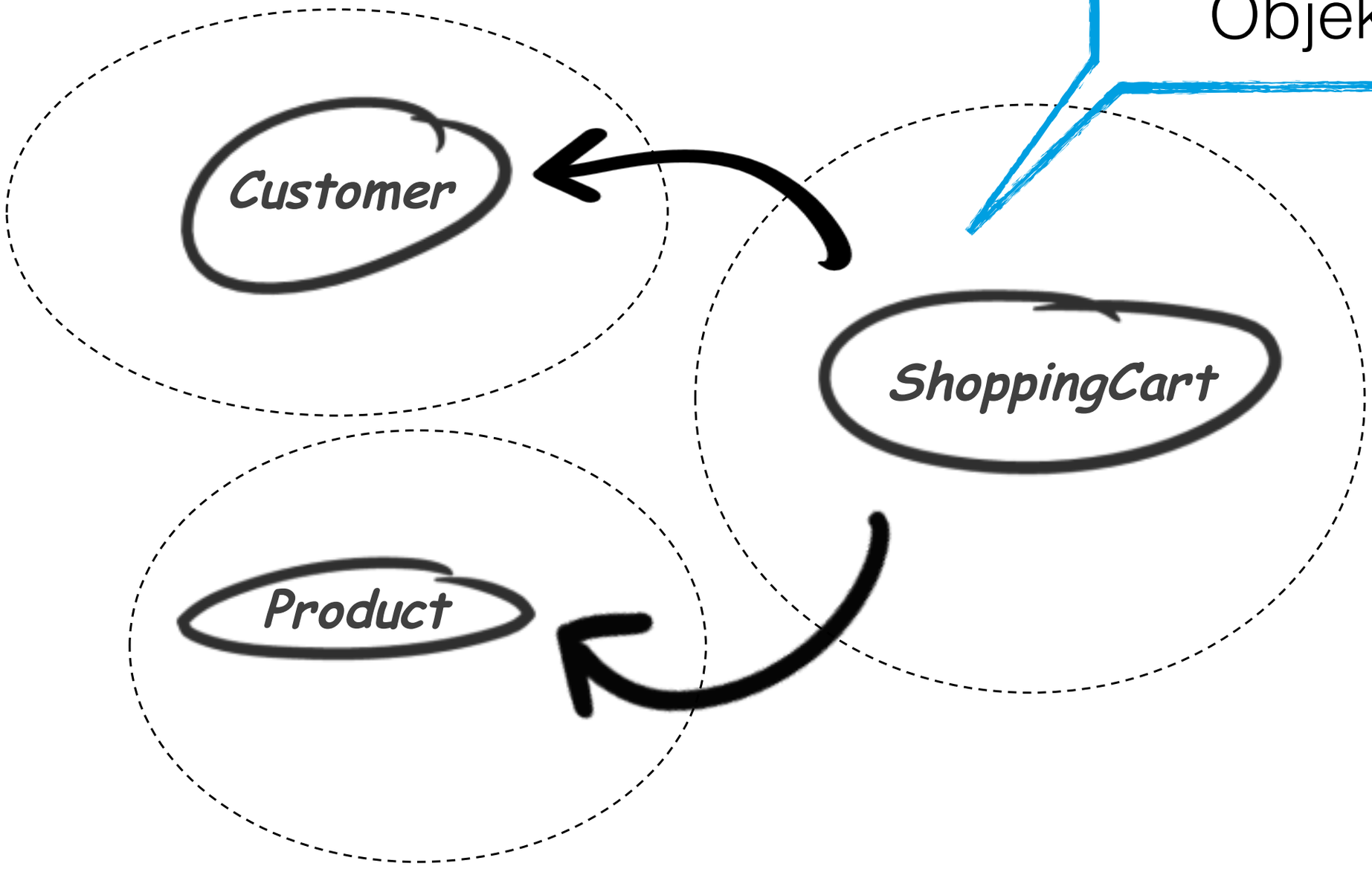
- „verschiedene“ Modelle
- „verschiedenes“ Verständnis

Surviving Service Design



Surviving Service Design

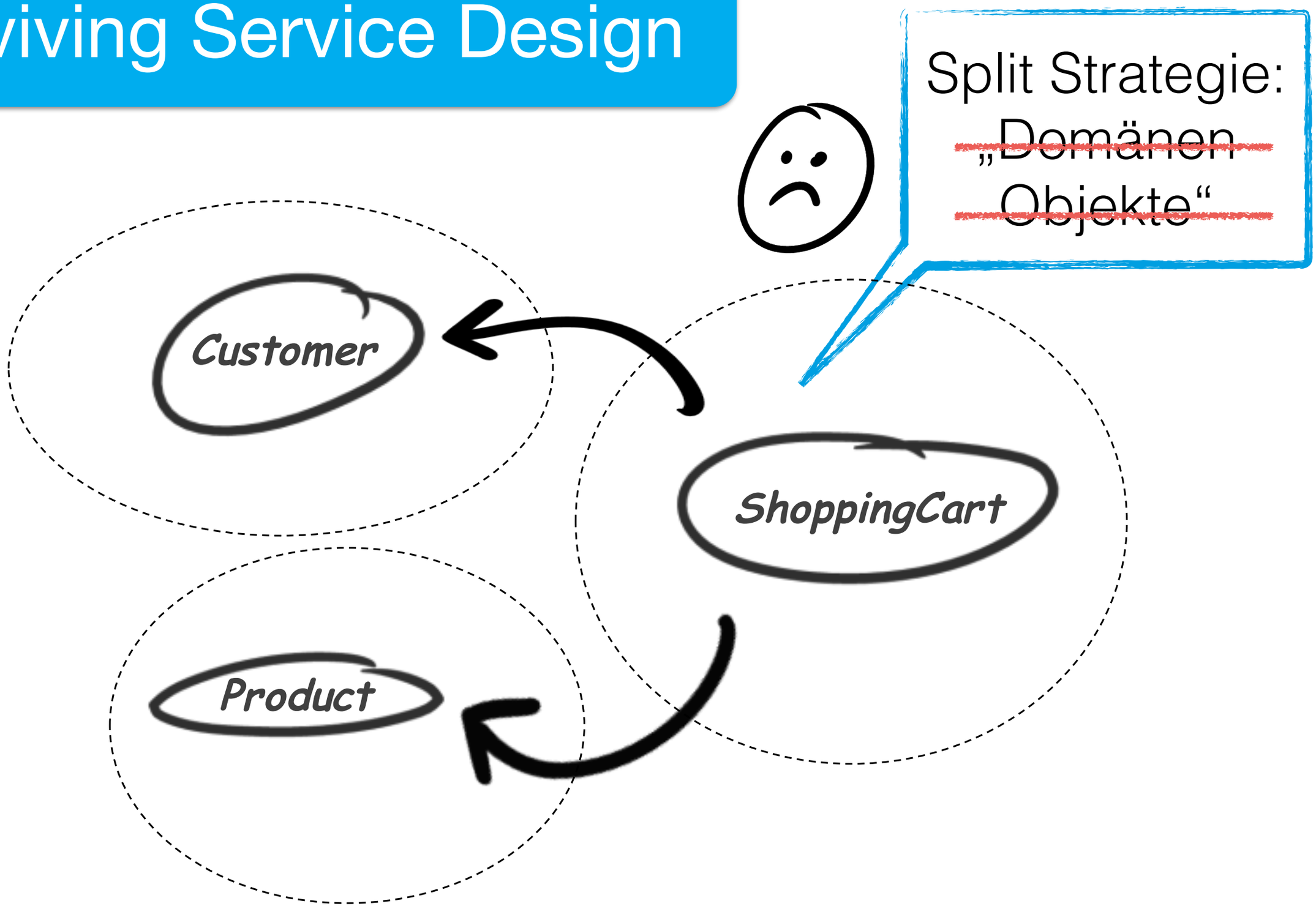
Split Strategie: „Domänen Objekte“




„If every Service has to be updated at the same time, it’s **not loosely coupled**!”
by Adrian Cockcroft (ehemals Netflix)

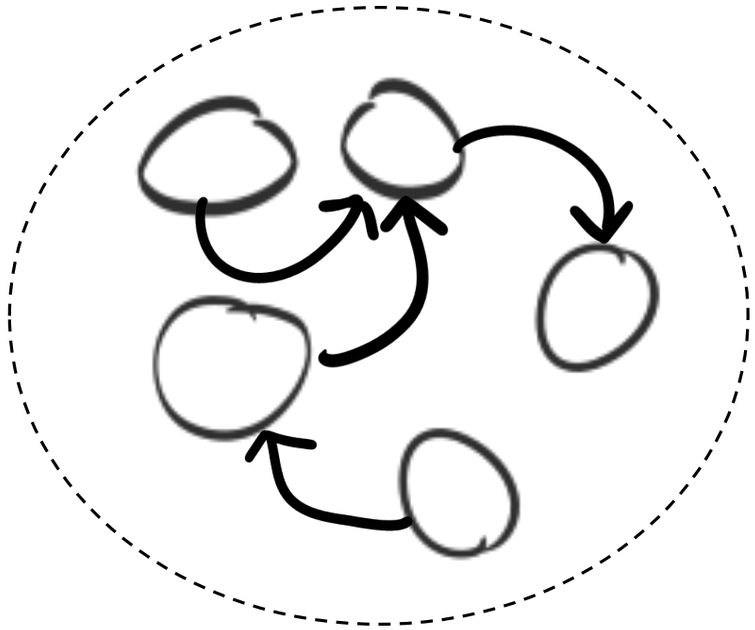

„If you have to know too much about the surrounding services you don’t have a **bounded context**!”
by Adrian Cockcroft (ehemals Netflix)

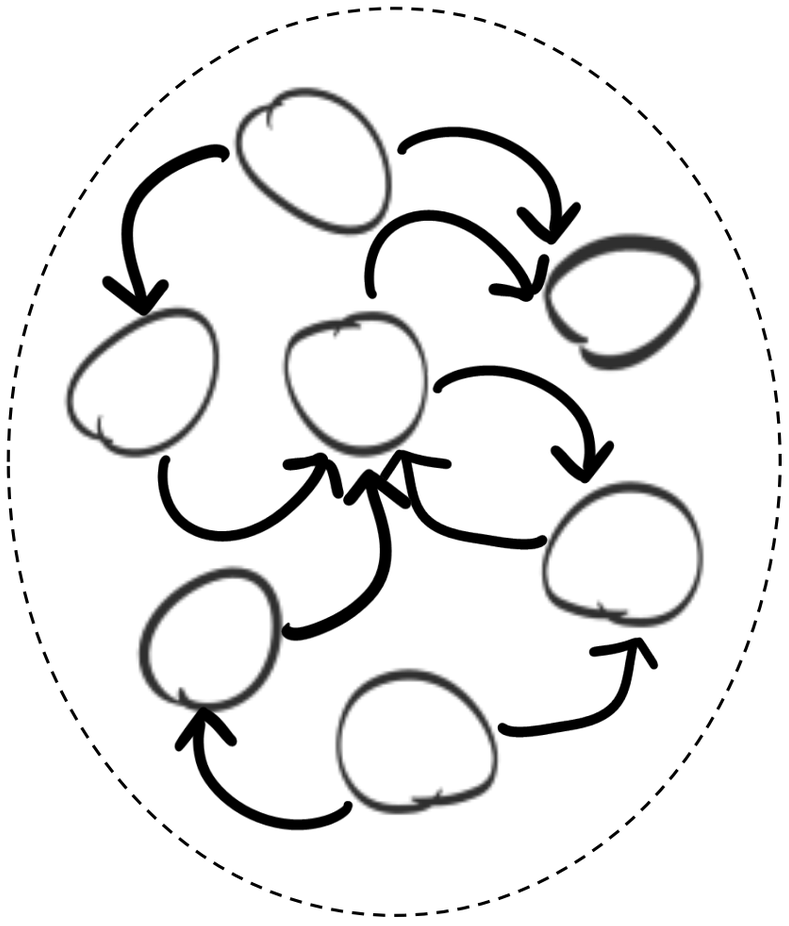
Surviving Service Design

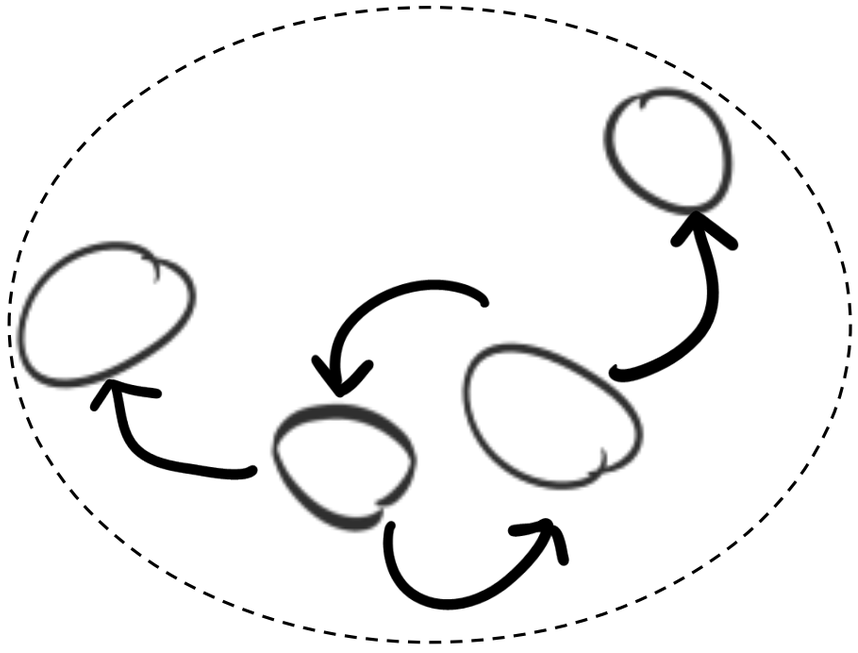


Surviving Service Design

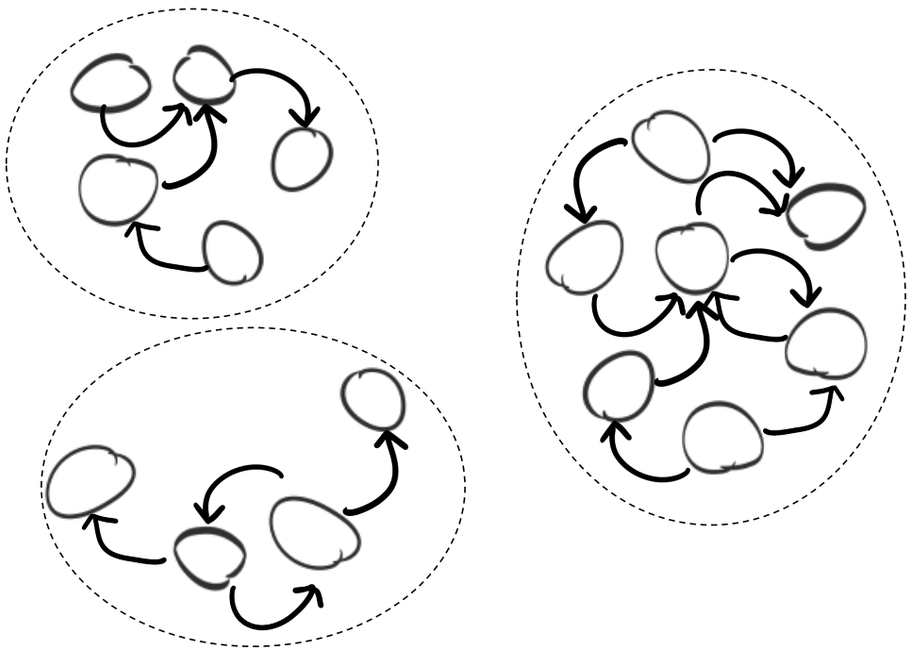
Split Strategie: „Bounded Context“







Surviving Service Design

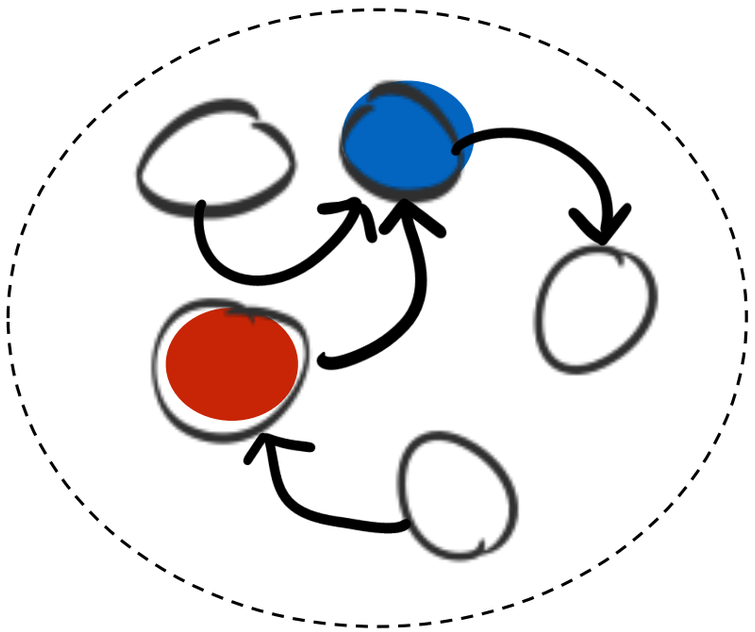


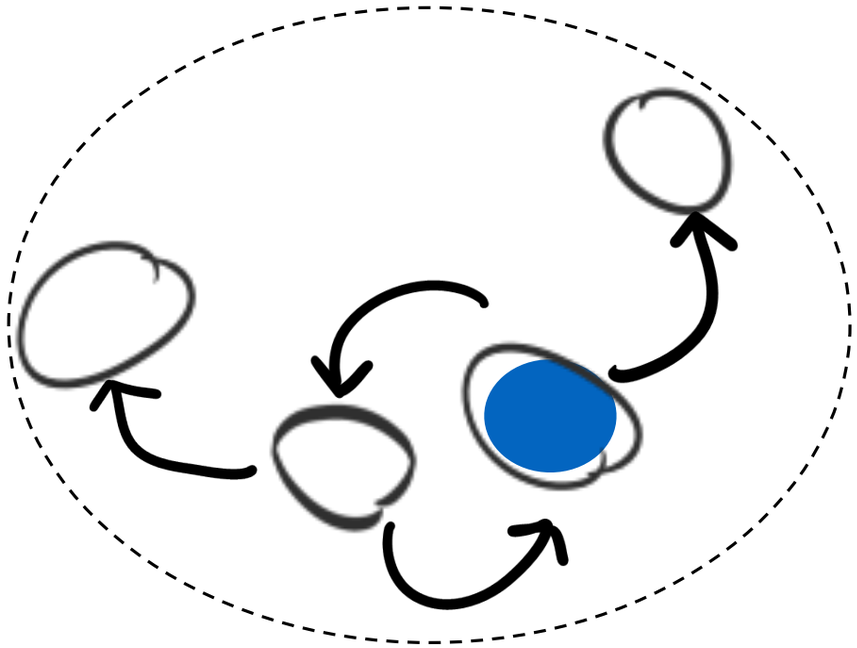
Bounded Context, aber wie?

- Klare Grenzen
- Business-Prozesse
- Inkonsistenzen vermeiden

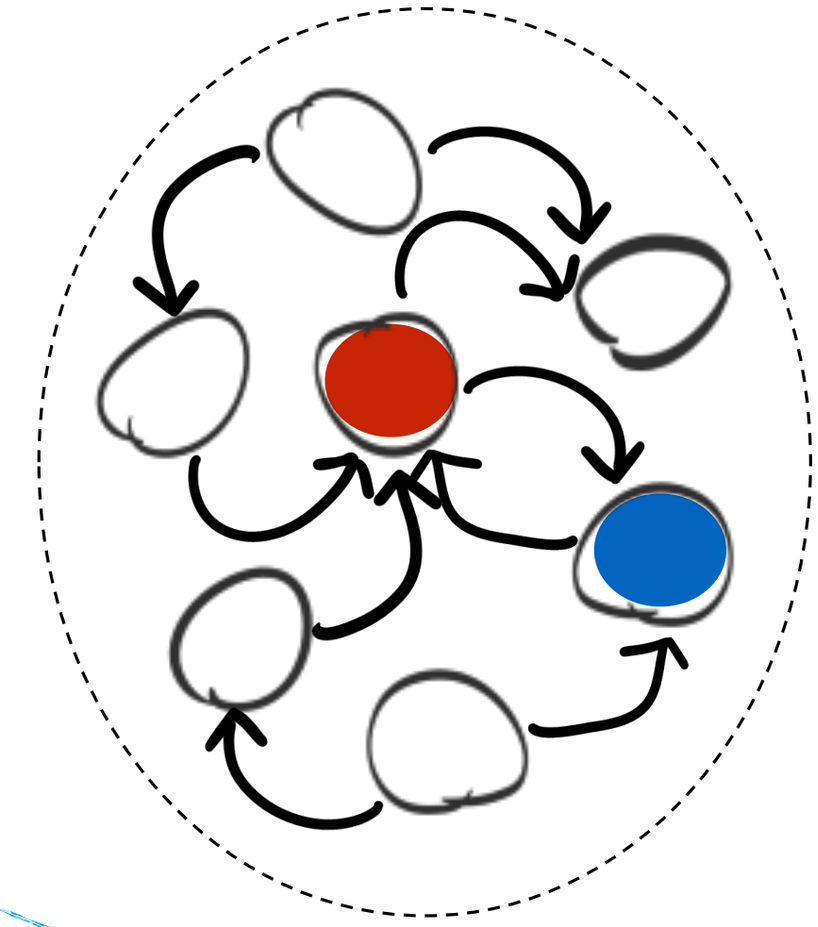
- e.g. „check out“
- e.g. „present product“
- e.g. „my recommendations“

Surviving Service Design





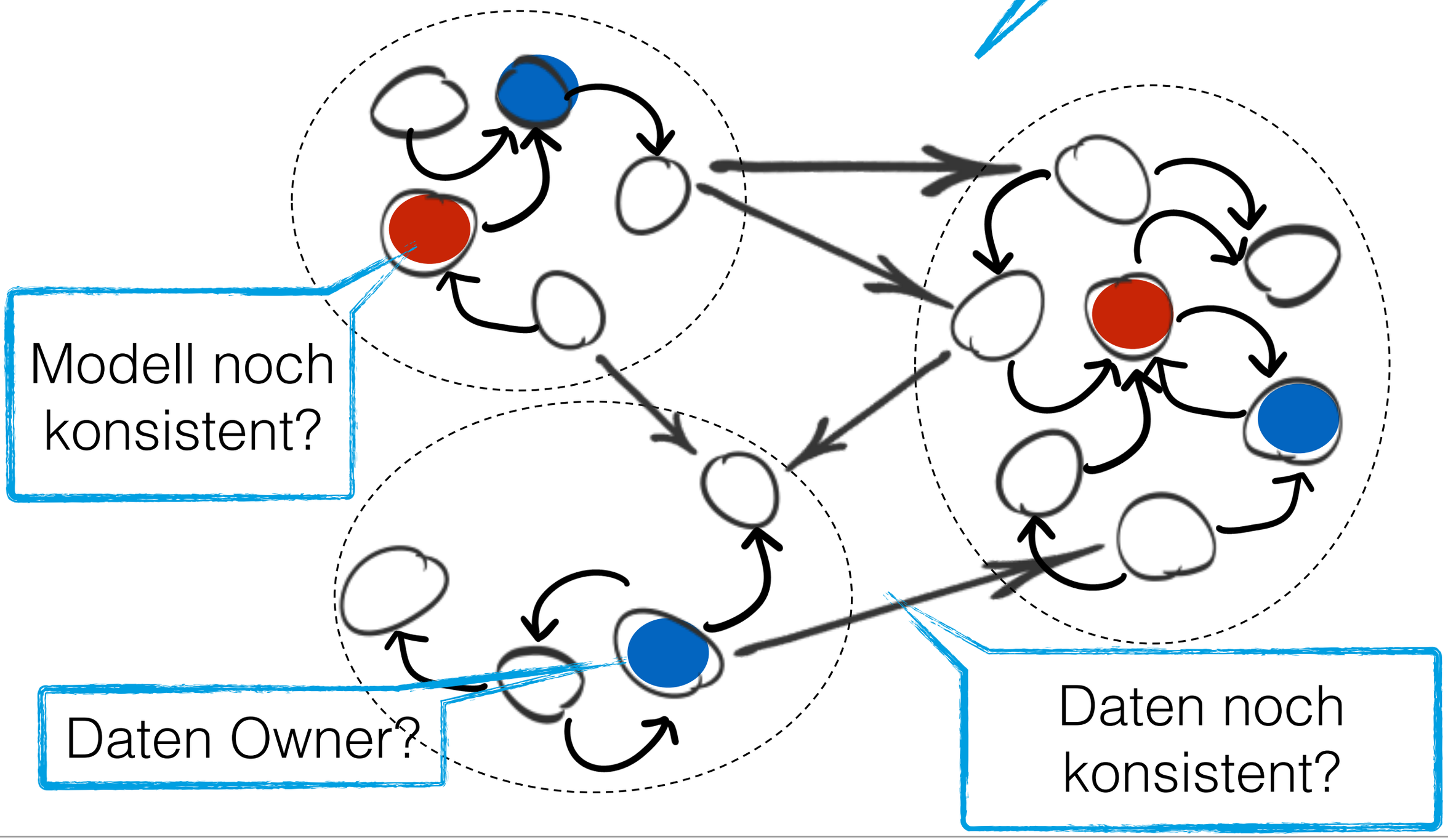
Abhängigkeiten?



„Know your Neighbors!“

Surviving Service Design

Context Map



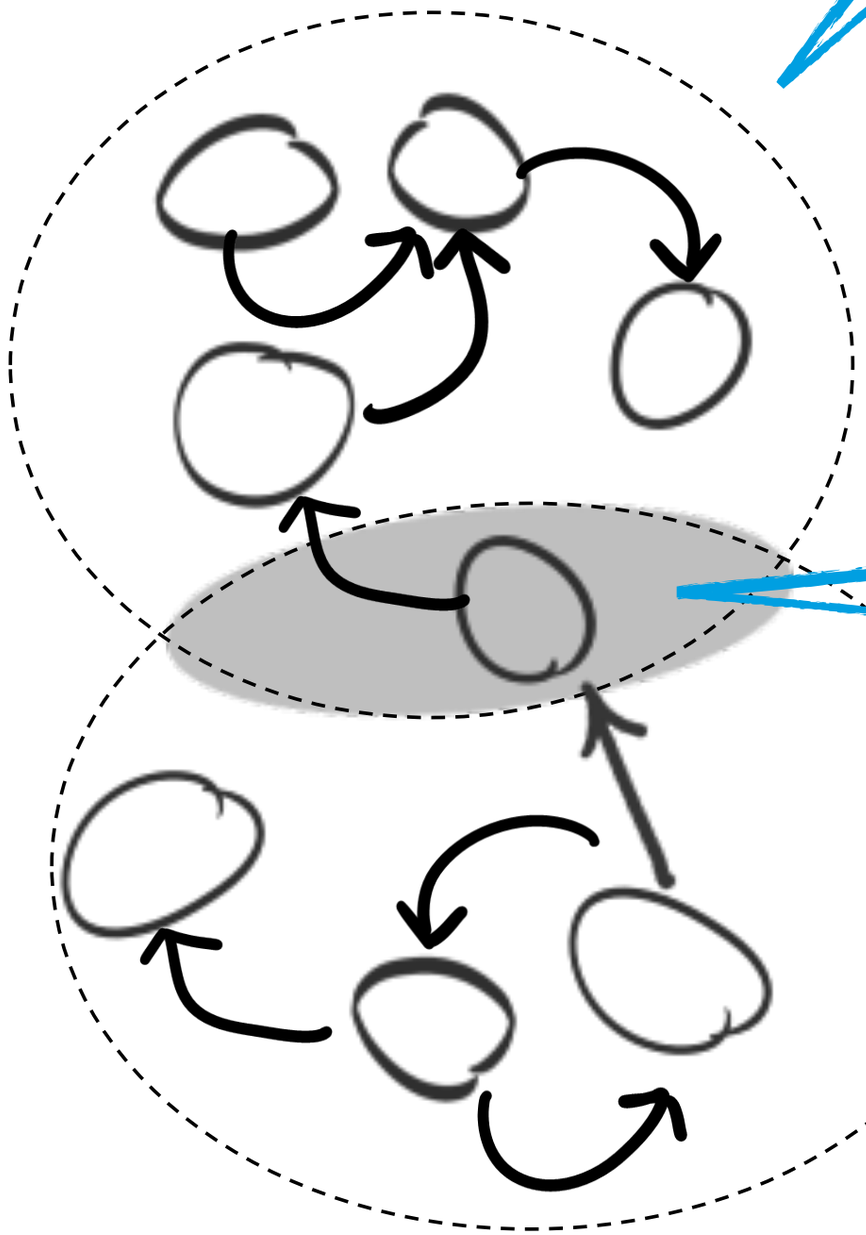
„Starbucks does not use

Two-Phase Commit“

http://www.enterpriseintegrationpatterns.com/ramblings/18_starbucks.html

Surviving Service Design

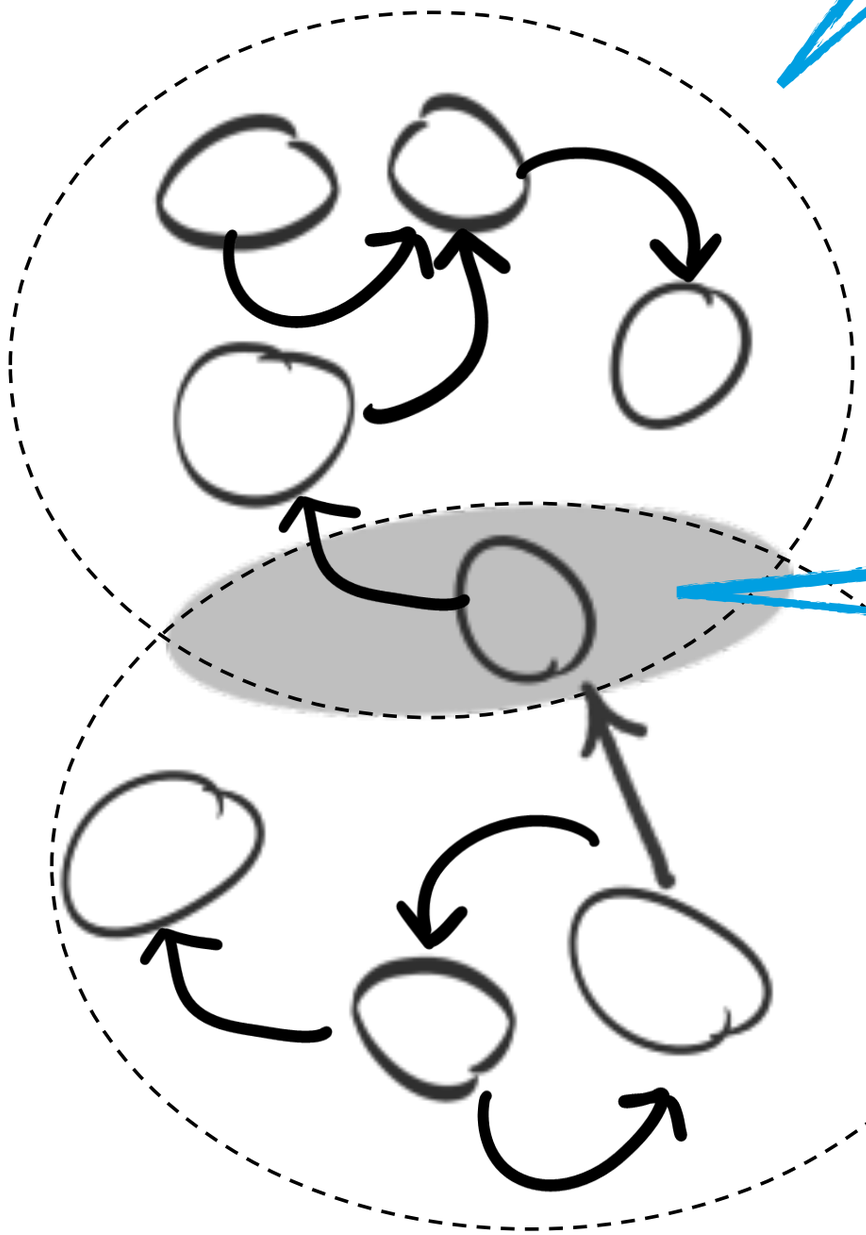
Shared Kernel



Meins? Deins?

Surviving Service Design

Shared Kernel



Versionierung?

Surviving Service Design

Versionierung?

„Be **conservative** in what you do, be **liberal** in what you expect.“

Postel’s Law (a.k.a. robustness principle)

Surviving Service Design

Versionierung?

Best Practices: **strategic**

- nicht forcieren
- „closely coupling“ vermeiden
- Consumer-Driven Contracts

- Semantic Versioning (Major.Minor.Patch)
- Koexistenz mehrerer Endpoints
- Koexistenz mehrerer Service-Versionen

Surviving Service Design

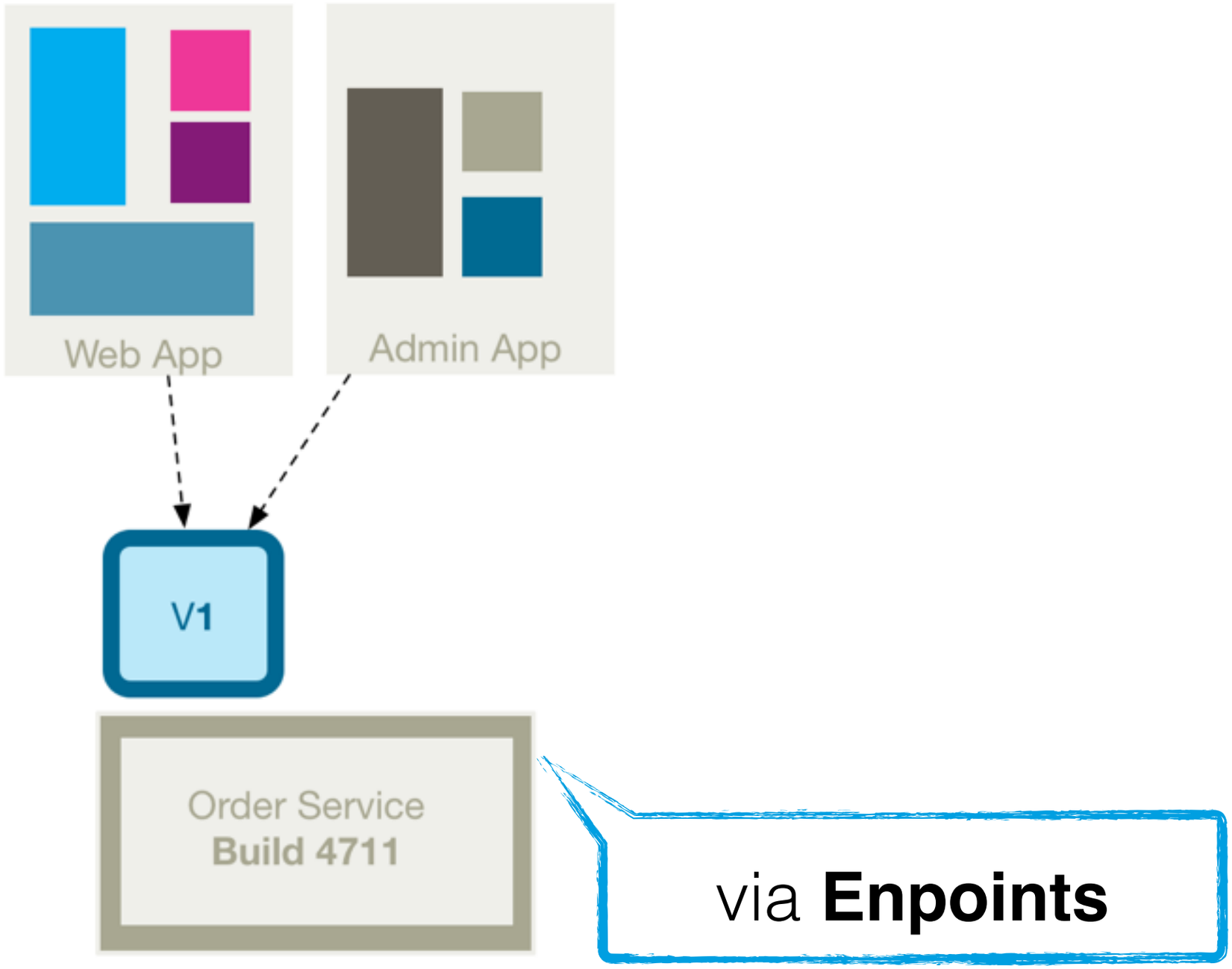
Versionierung?

Best Practices: **technical**

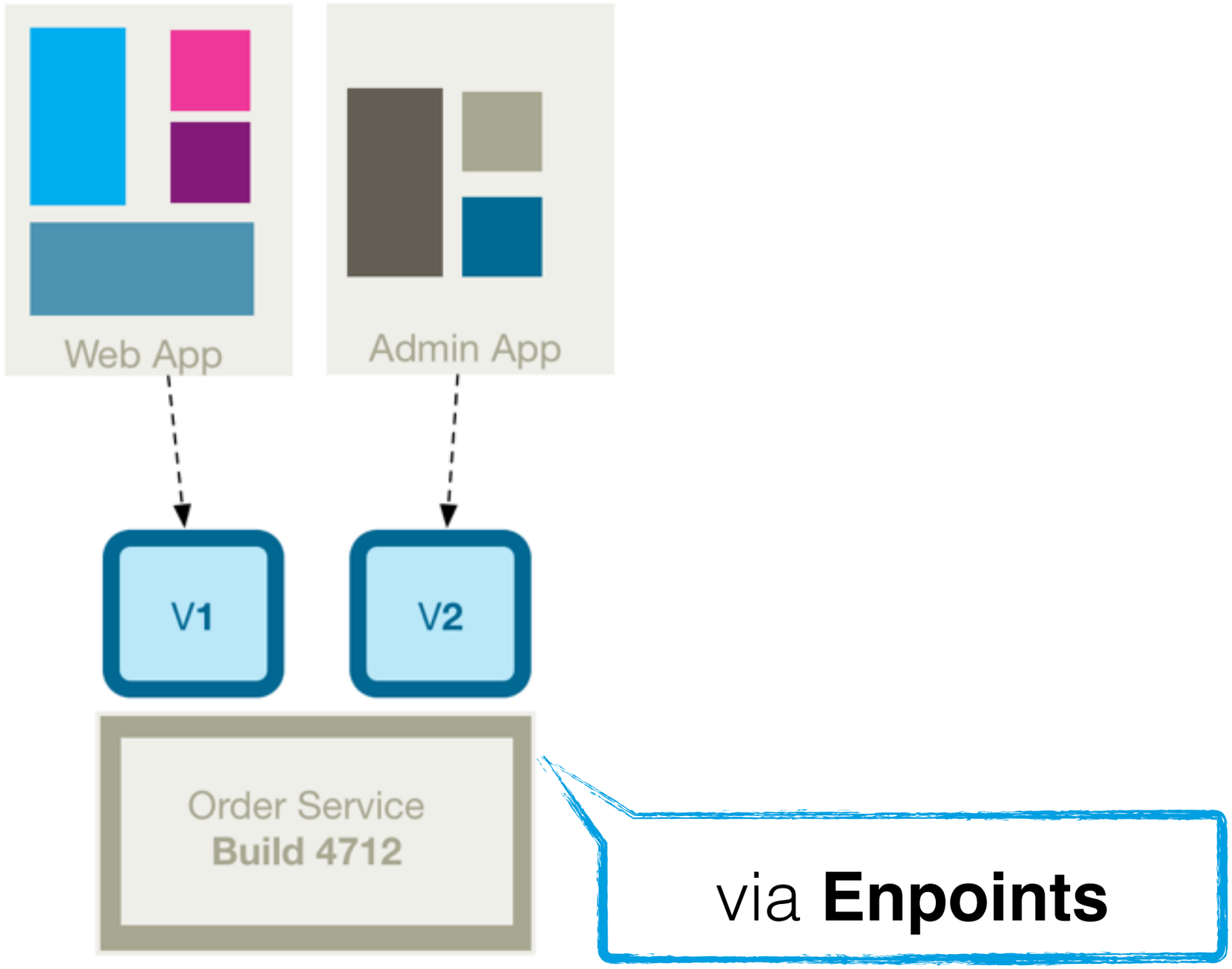
- gar nicht (via „ignorieren“)
- gar nicht (via „zusätzliche Ressourcen“)
- gar nicht (via „erweiterbare Datenformate“)

- Versionsnummer in der URL
- Versionsnummer im Header (via Custom-Header)
- Content Negotiation (via Accept-Header)

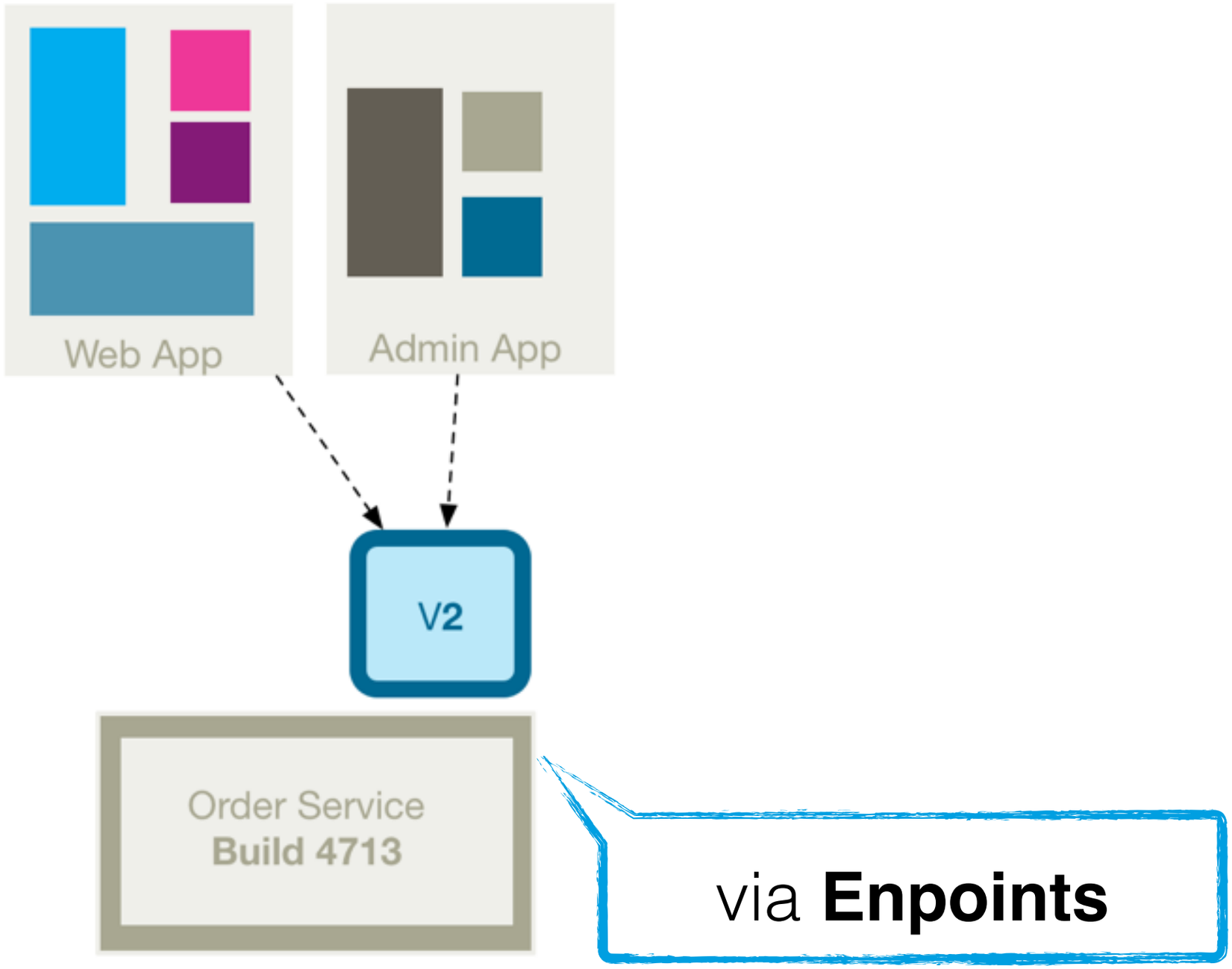
Surviving Service Design



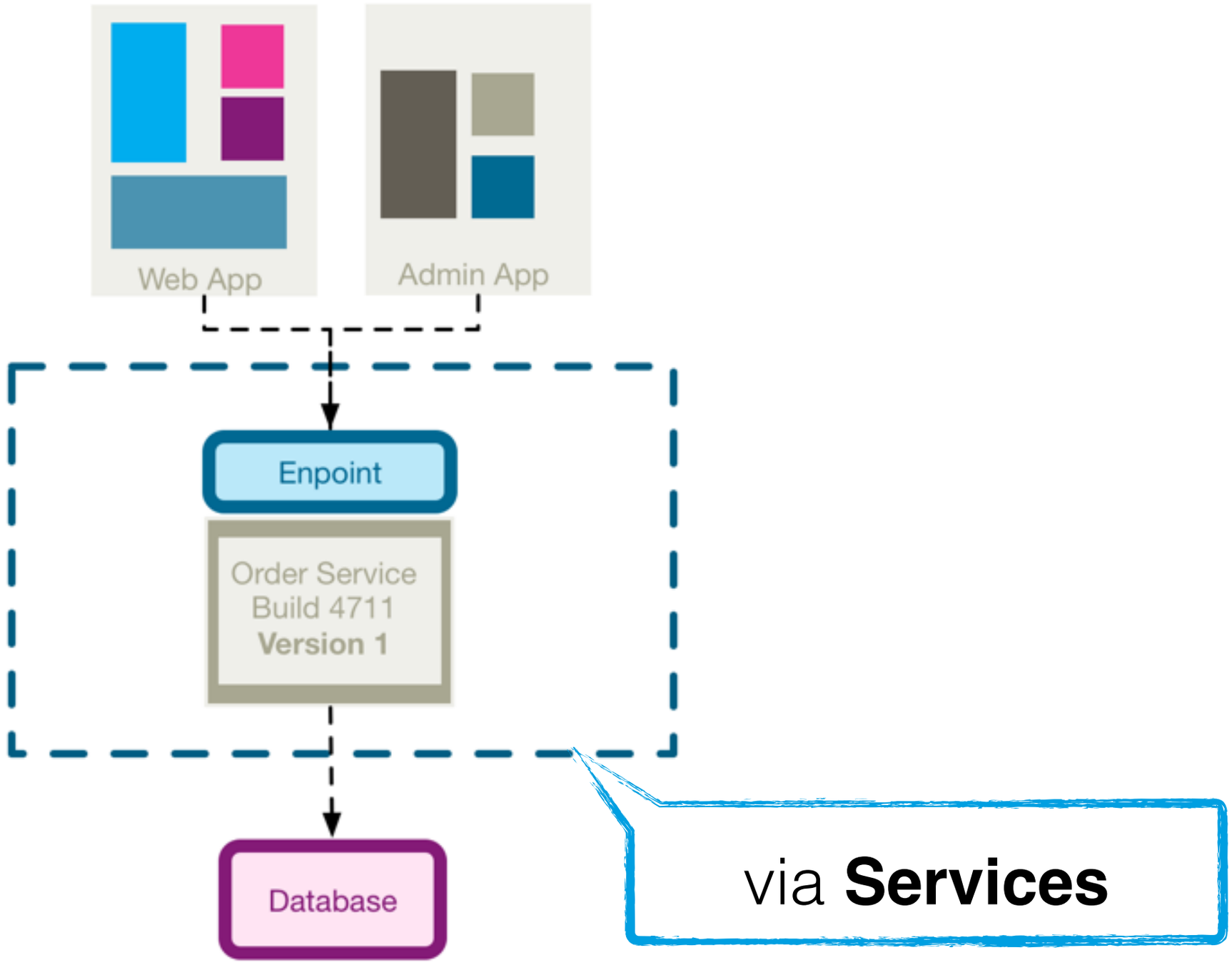
Surviving Service Design



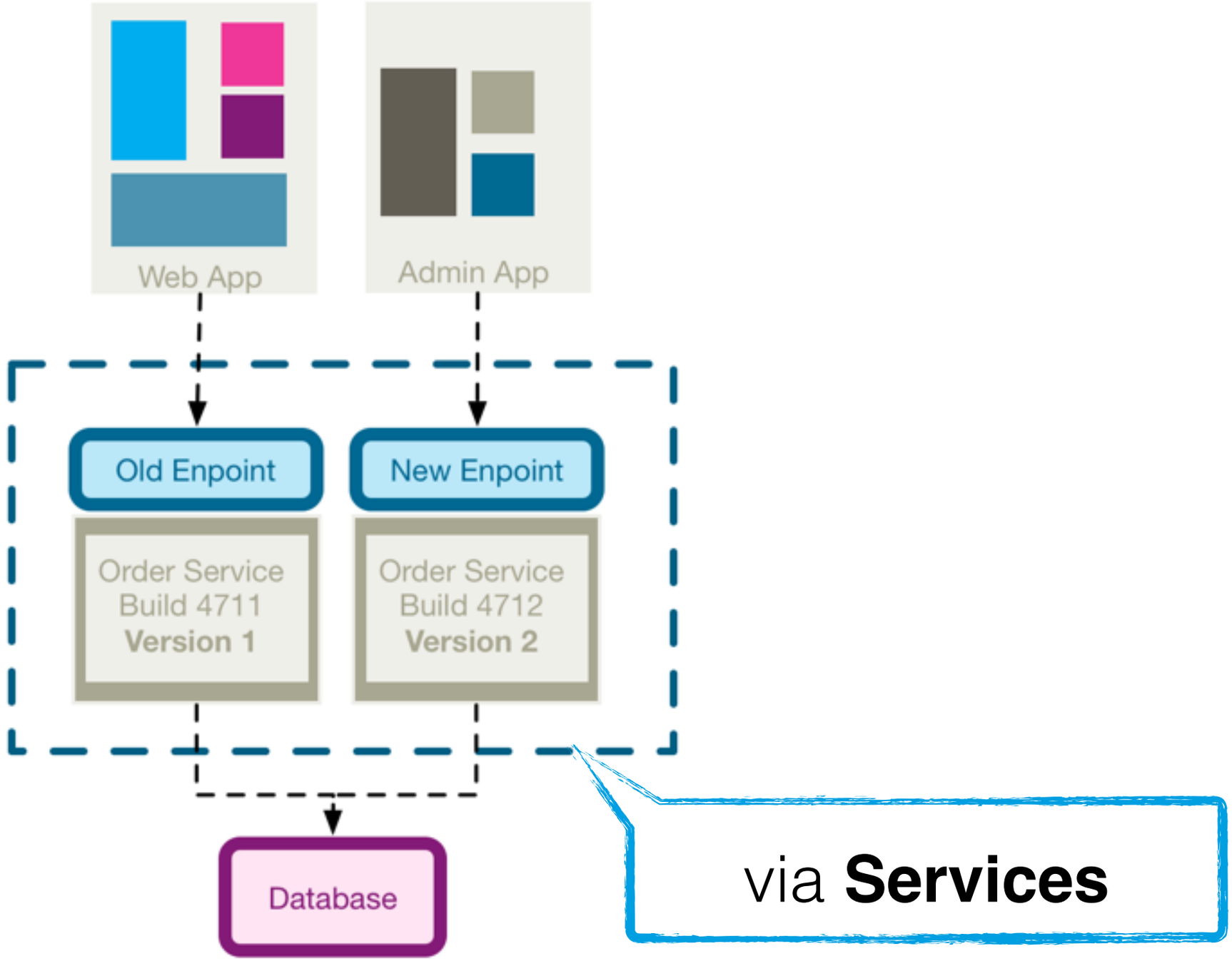
Surviving Service Design



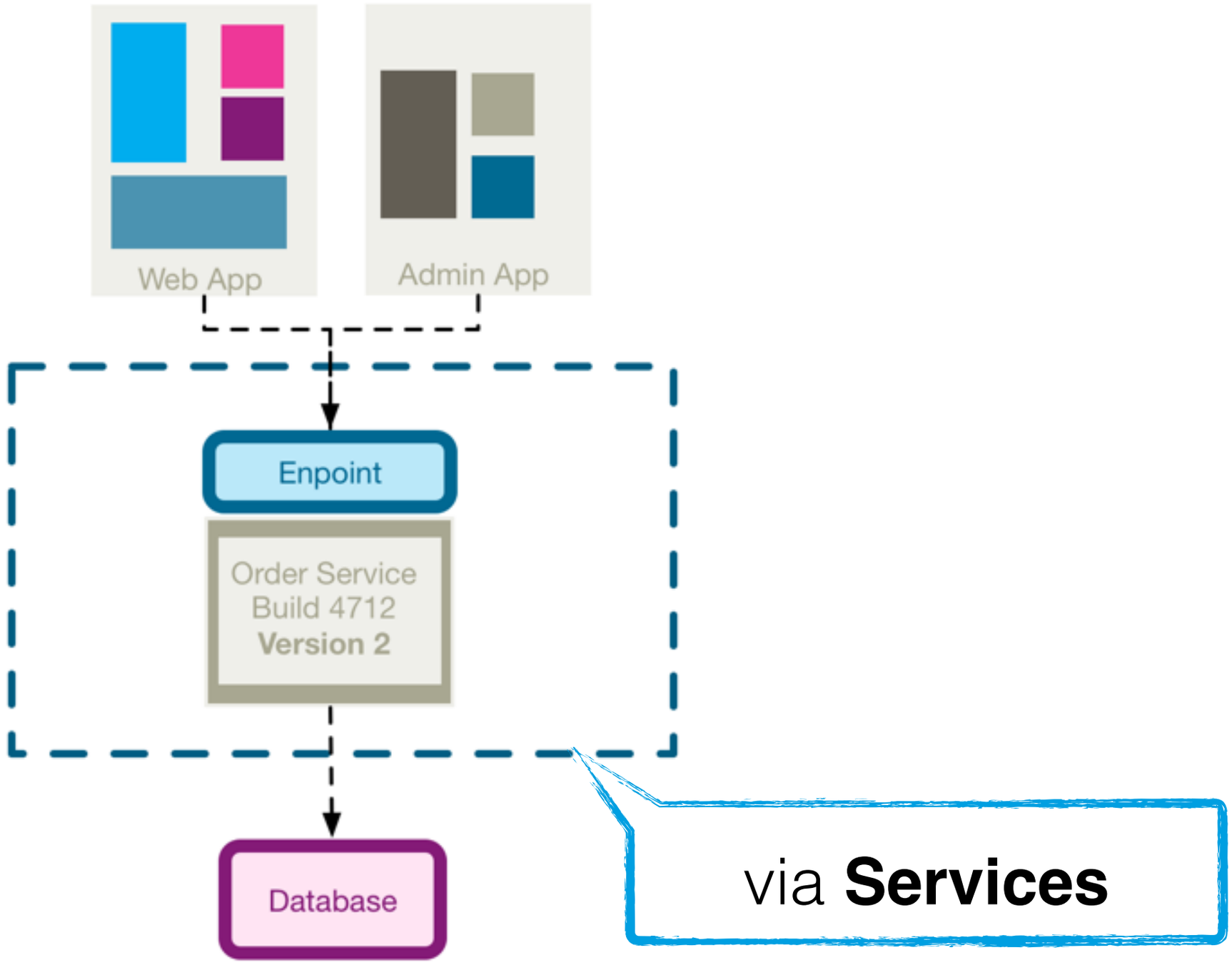
Surviving Service Design



Surviving Service Design

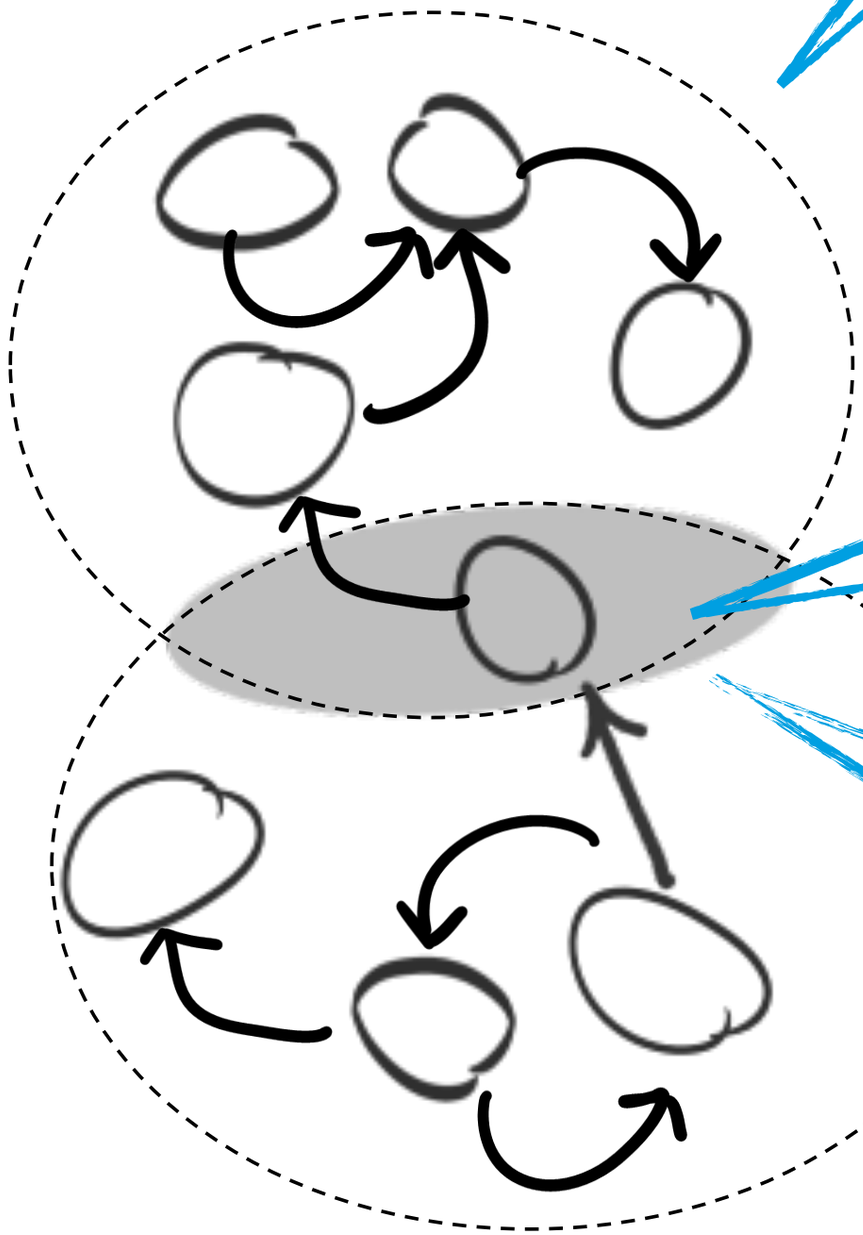


Surviving Service Design



Surviving Service Design

Shared Kernel

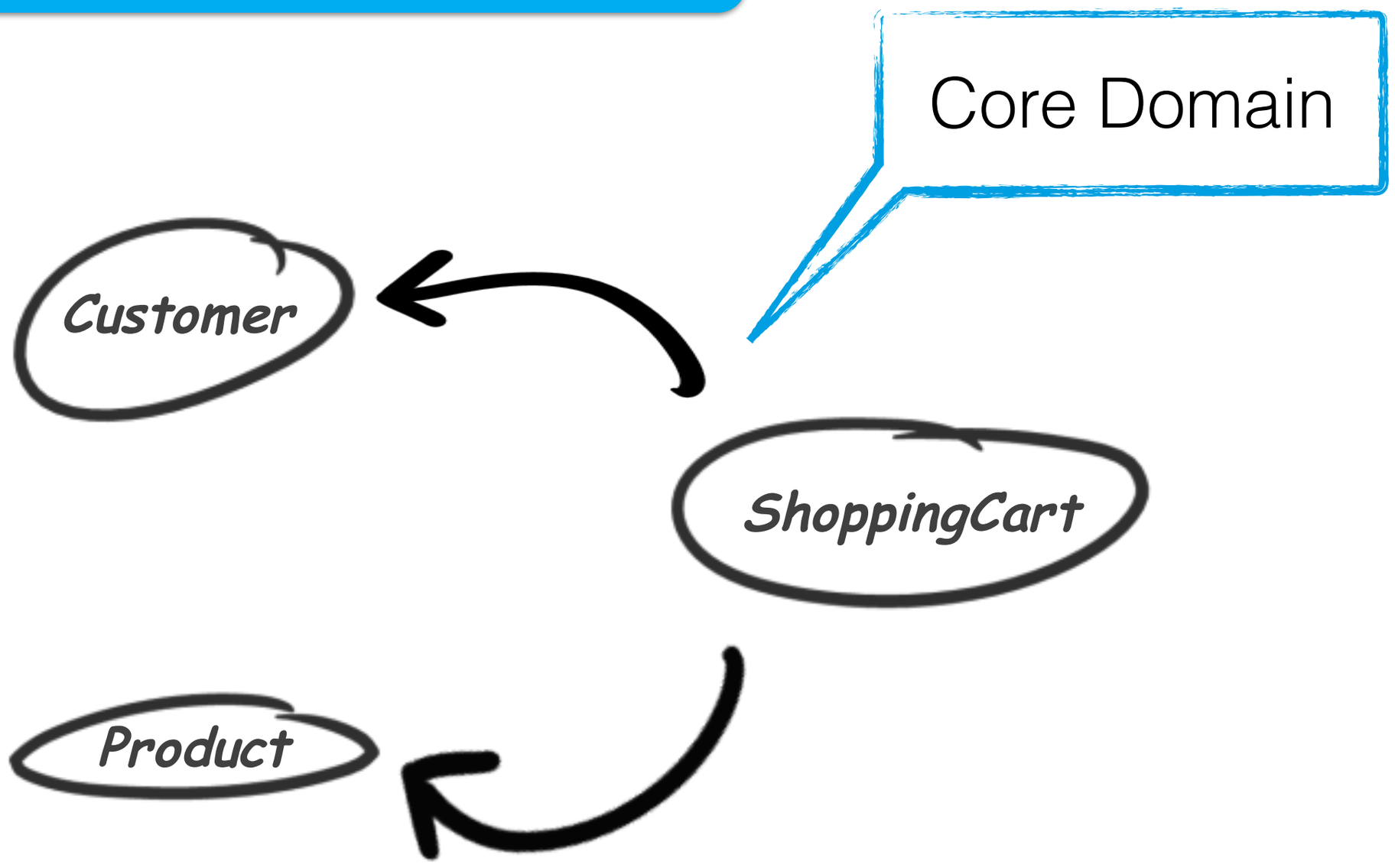


Ich auch!

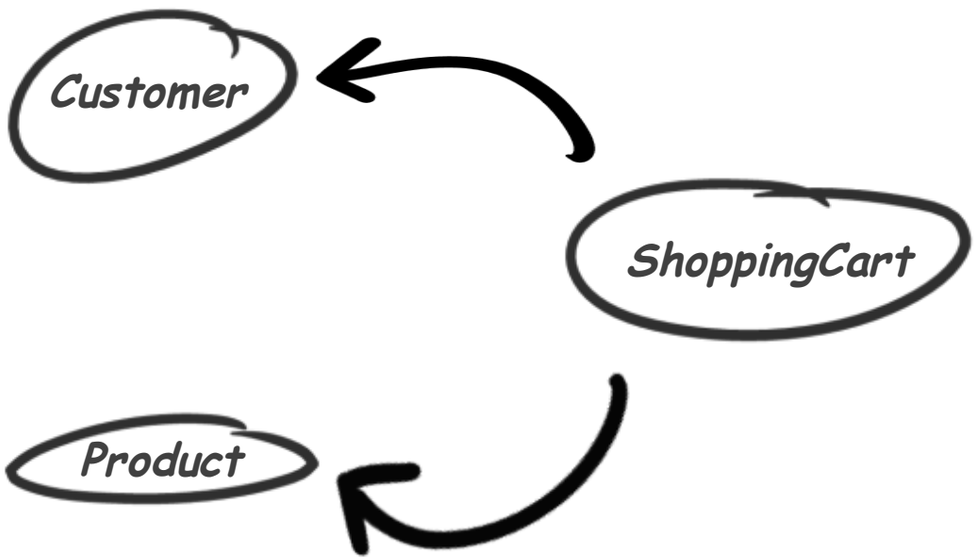
Ich brauche das!

Und ich auch!

Surviving Service Design



Surviving Service Design



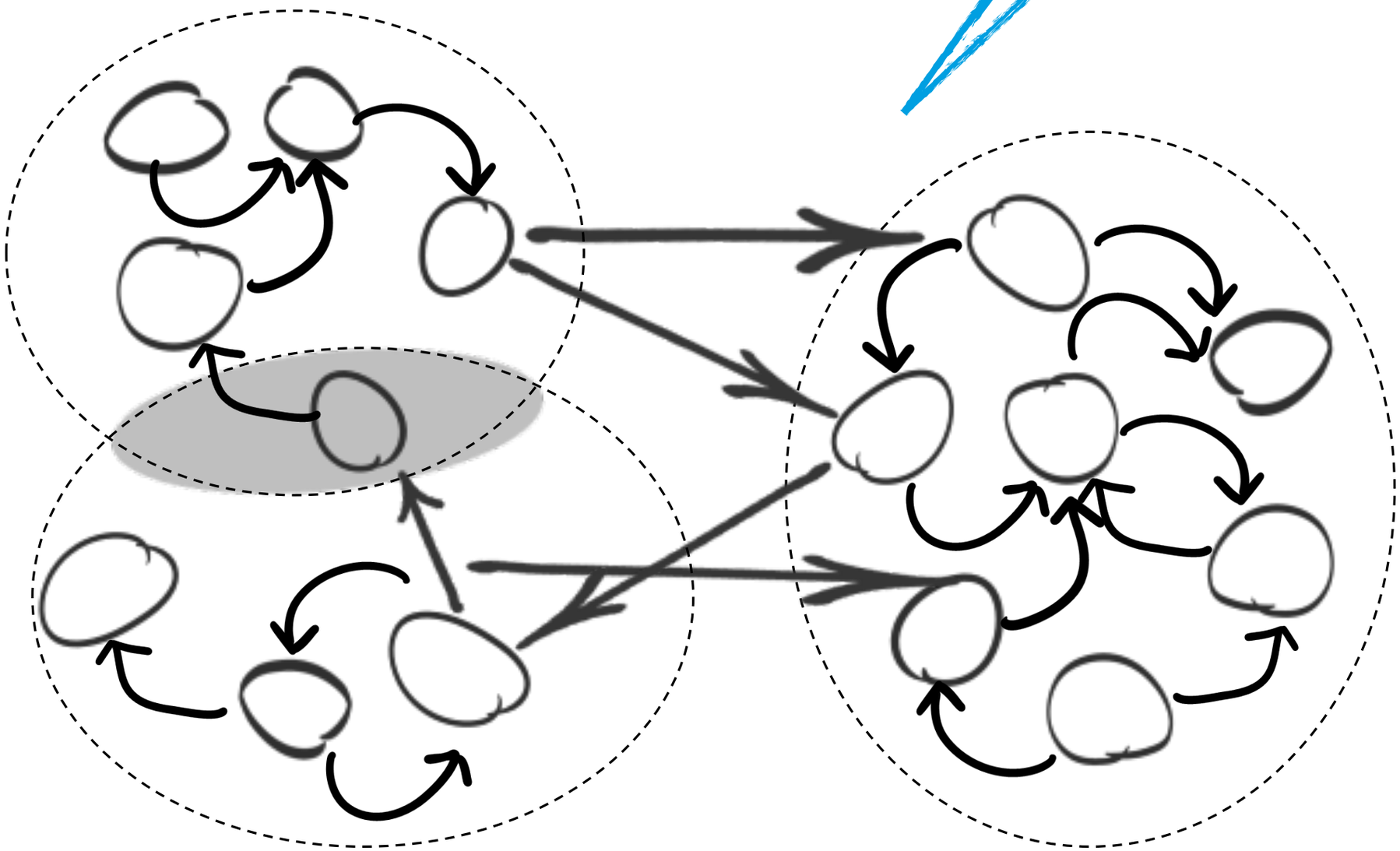
Core Domain

- fachliche Basis
- saubere Modellierung
- div. Repräsentationen
- zur Kommunikation
- i.d.R. nur wenige Attribute

- Trennen von „common“

Surviving Service Design

Evolution?



Surviving Service Design

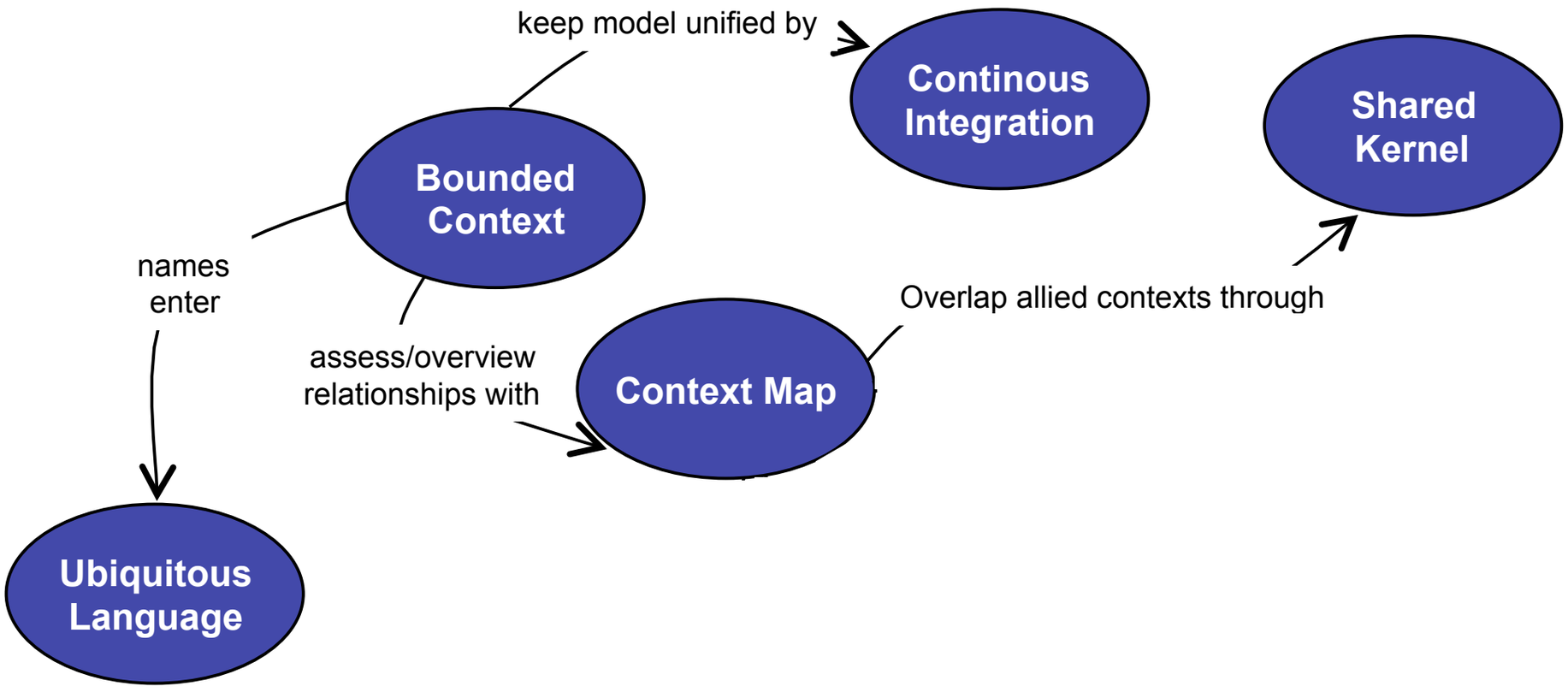
http://techblog.netflix.com/2013/01/announcing-ribbon-tying-netflix-mid.html

Surviving Service Design

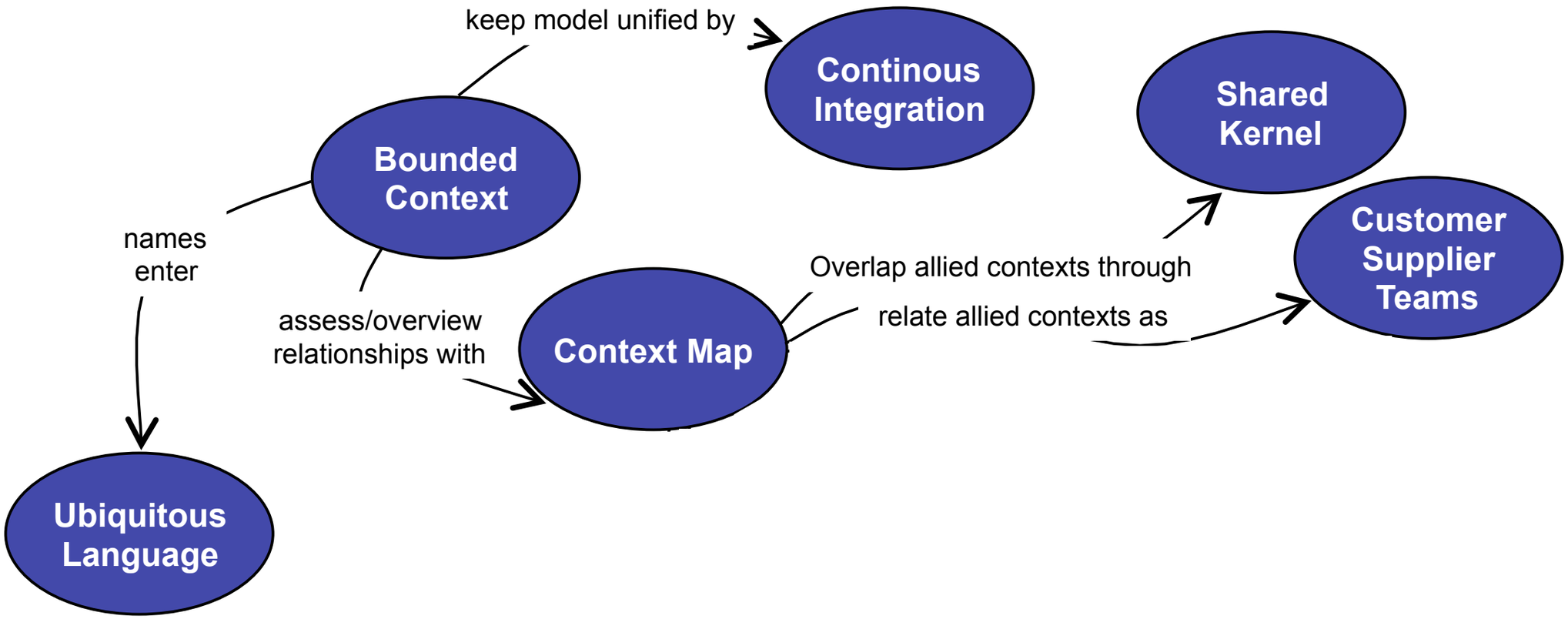
„A Microservice never walks alone!“

me, myself and I (2015)

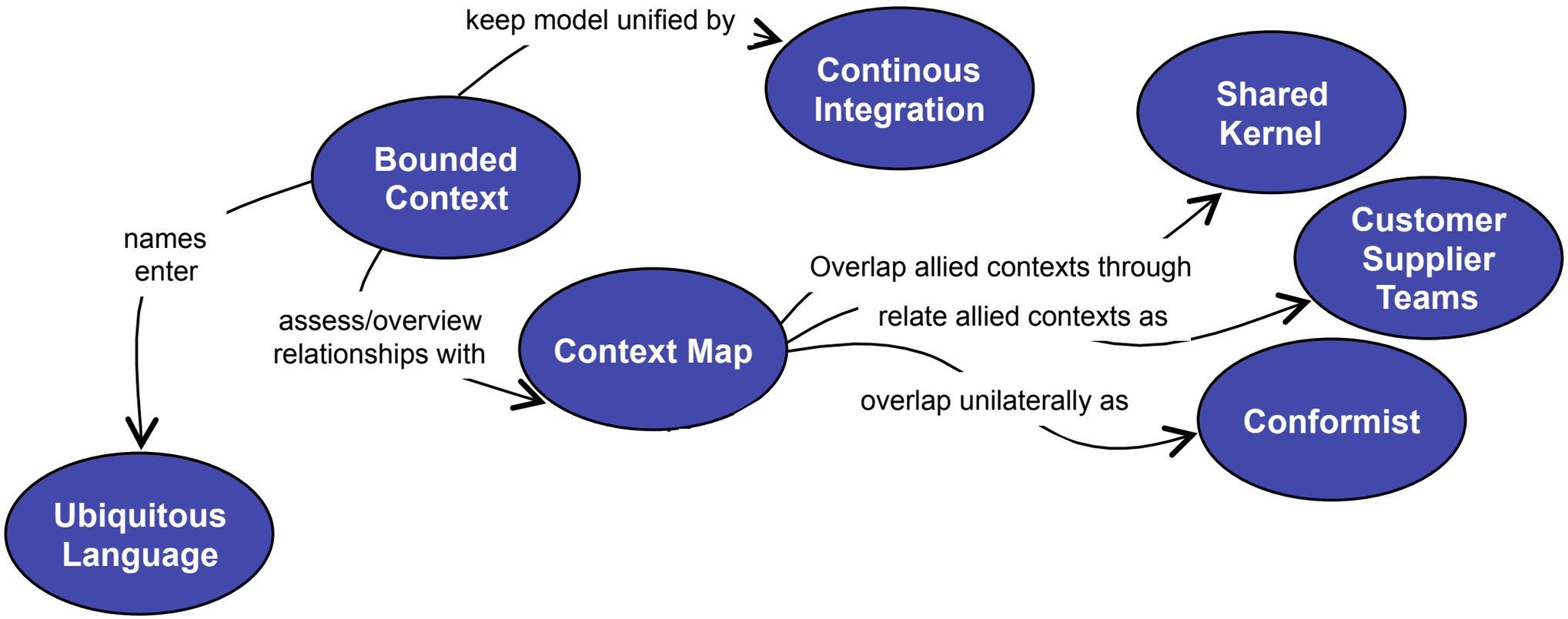
Surviving Service Design



Surviving Service Design

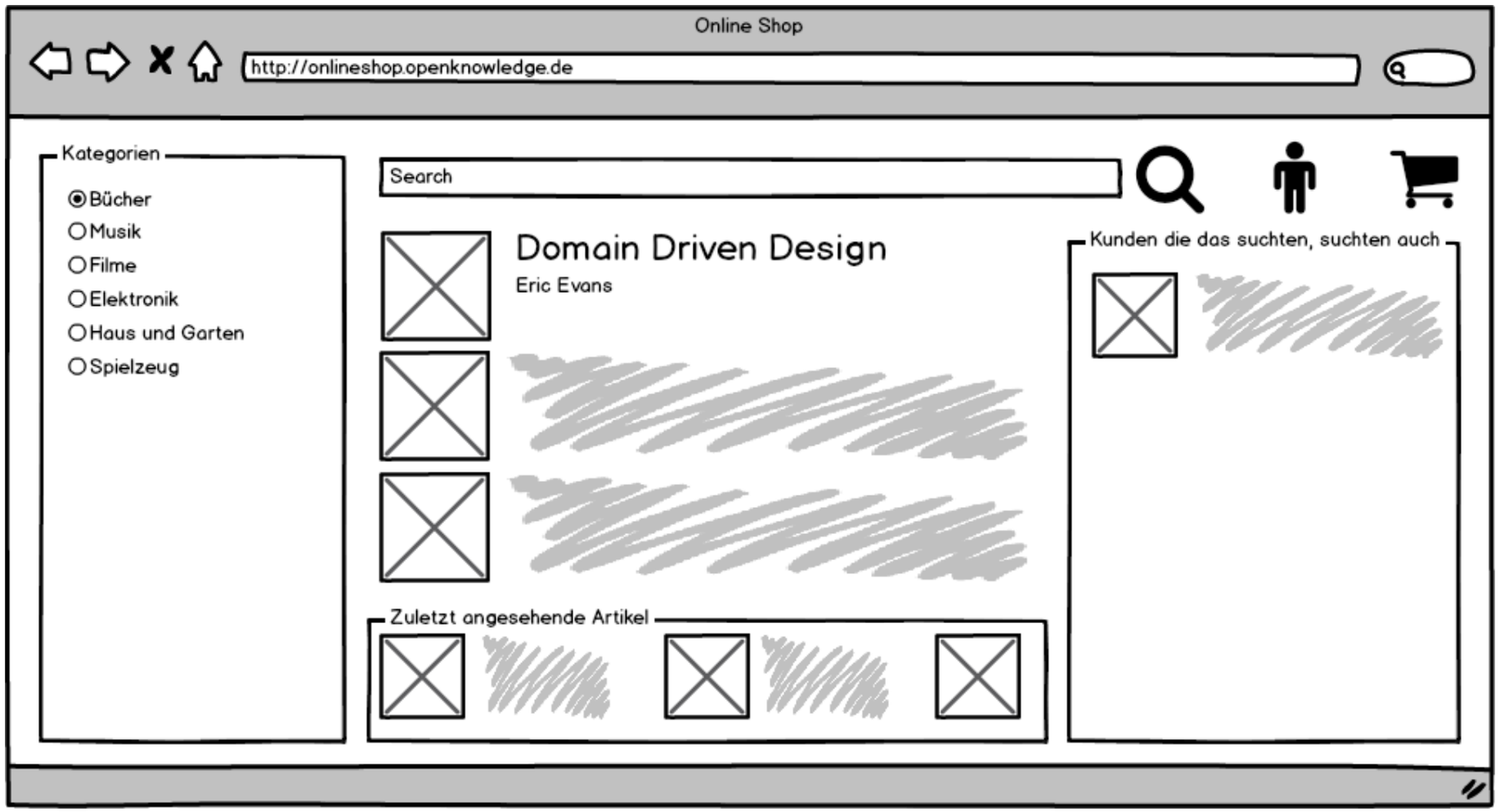


Surviving Service Design

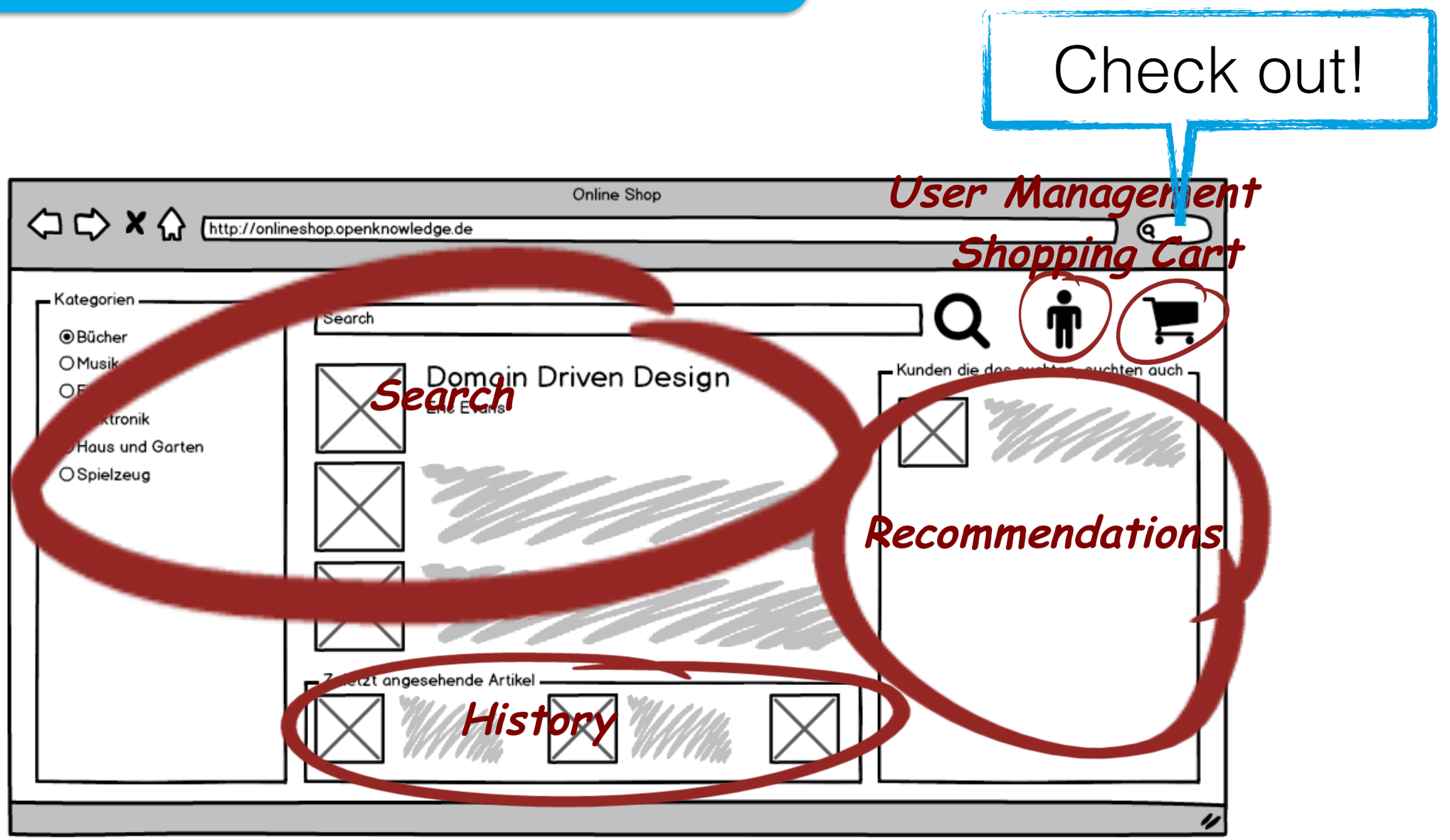


Surviving Service Design

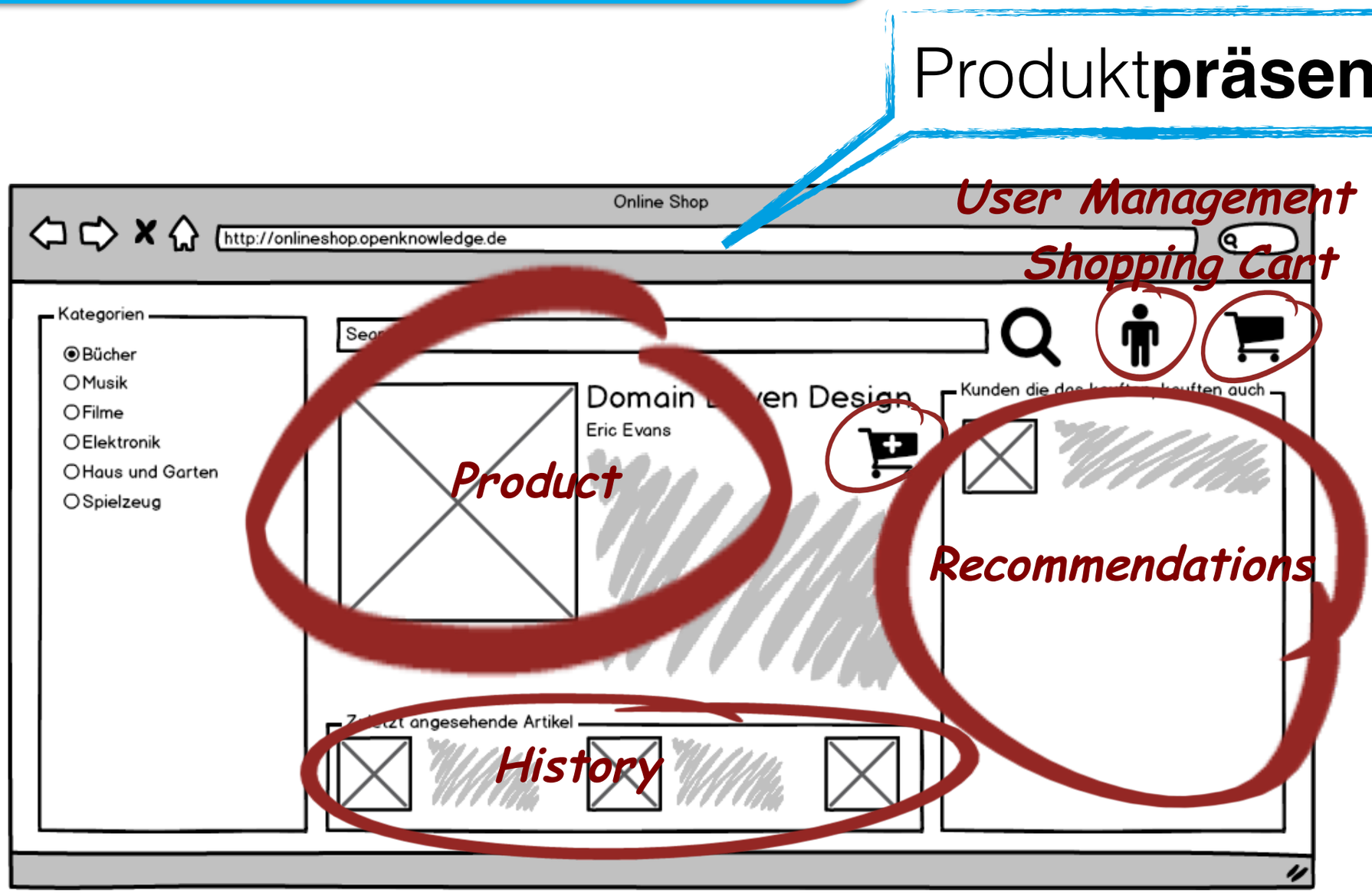
Microservices?



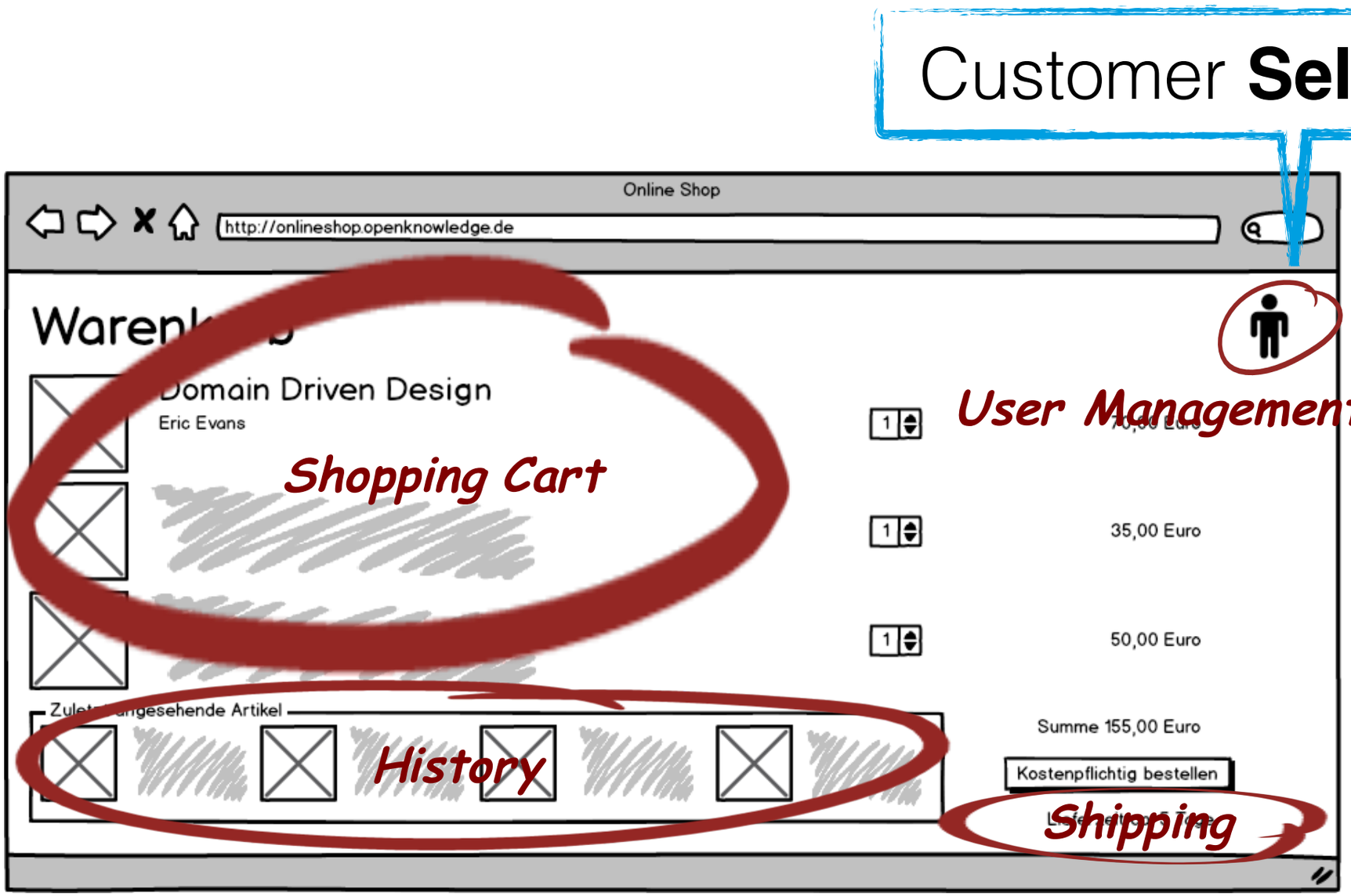
Surviving Service Design



Surviving Service Design



Surviving Service Design



Size does(n’t) matter

Eine subjektive Einschätzung

Meine Sicht der Dinge

- „Verwende eine **gemeinsame Sprache**“
- „Kenne dein **Domain Model**“
- „Kenne deine **Bounded Contexts**“
- „Kenne deine **Nachbarn**“
- „Denke in **Prozessen**, nicht in **Objekten**“
- „Sei **toleranter Reader** und **strikter Supplier**“
- BTW: „**Miteinander reden** ist weiterhin erlaubt!“

Bildnachweise

- #1: iStock.com / Andrew Rich (000029536120)
- #3: © Daniel Steger for openphoto.net
- #10: The Art of Scalability / theartofscalability.com
- #13: iStock.com / jpgfactory (000059091620)
- #15: iStock.com / VisualField (3096596)
- #16: Wikimedia / Webysther 20150414193208
- #25: iStock.com / PeopleImages (62387488)
- #26: dddcommunity.org/uncategorized/about_the_cover
- #32: twitter.com/olliwegner
- #36: aws.amazon.com/de/heroes/usa/adrian-cockcroft/
- #44: pixabay.com
- #51: techblog.netflix.com/2013/01/announcing-ribbon-tying-netflix-mid.html