

# CONWAY'S LAW AND DEVOPS

Abdelkader Mazan, Senior Consultant  
20. März 2018



# CGI

Experience the commitment®

# AGENDA

- Conway's Law
- The Monolith as an example
- The Scale Cube
- Bounded Context
- Self-Contained Systems
- Integrating Self-Contained Systems
- Chaos engineering in Netflix



# AGENDA

- **Conway's Law**
  - The Monolith as an example
  - The Scale Cube
  - Bounded Context
  - Self-Contained Systems
  - Integrating Self-Contained Systems
  - Chaos engineering in Netflix



# CONWAY'S LAW

„...**Organizations** which **design systems**...are constrained to produce designs which are copies of the communication structures of these organizations...”

Melvyn Conway, 1968



# Exploring the Duality between Product and Organizational Architectures: A Test of the “Mirroring” Hypothesis

The mirroring phenomenon is consistent with two rival causal mechanisms. First, designs may evolve to reflect their development environments. **In tightly-coupled organizations**, dedicated teams employed by a single firm and located at a single site develop the design. Problems are solved by face-to-face interaction, and performance “tweaked” by taking advantage of the access that module developers have to information and solutions developed in other modules. **Even if not an explicit managerial choice, the design naturally becomes more tightly-coupled. By contrast, in loosely-coupled organizations**, a large, distributed team of volunteers develops the design. Face-to-face communications are rare given most developers never meet. Hence fewer connections between modules are established. **The architecture that evolves is more modular** as a result of the limitations on communication between developers

Harvard Business School and MIT Sloan School of Management

Working Paper Publication Date: March 2008



# AGENDA

- Conway's Law
- **The Monolith as an example**
- The Scale Cube
- Bounded Context
- Self-Contained Systems
- Integrating Self-Contained Systems
- Chaos engineering in Netflix



# THE MONOLITH AS AN EXAMPLE



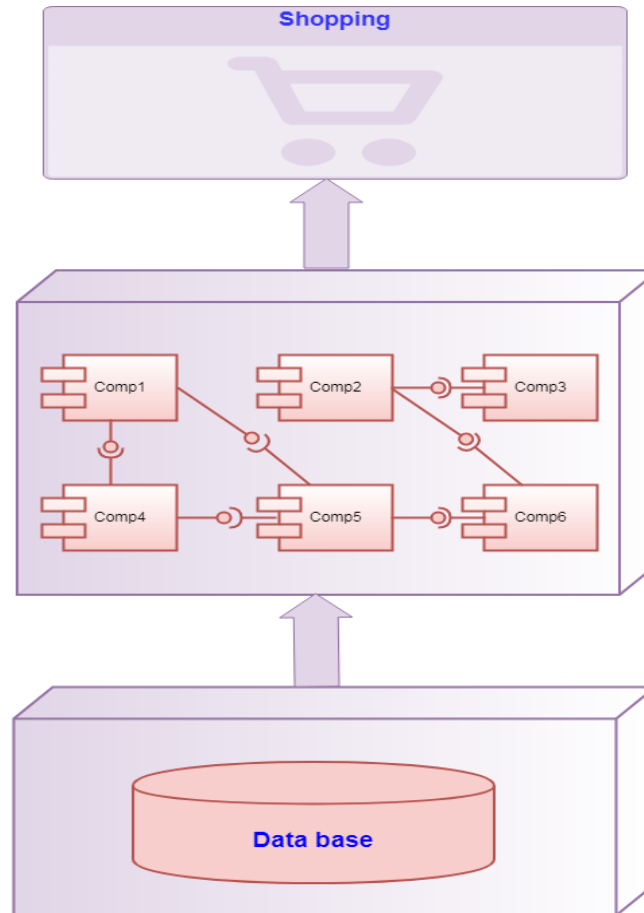
**UI**



**business logic**



**database**



Scale Up (CPU, Memory)

Scale out (horizontal duplication using Load balancer)

# PROBLEMS

- starke Abhängigkeiten und ineffiziente Zusammenarbeit zwischen den Teams
- starke Abhängigkeiten zwischen den Modulen
- sehr langsames Feedback
- kein Verantwortlichkeitsgefühl (**Accountability**) für das ganze System
- die Entwicklung von neuen Features wird mit der Zeit immer schwerer
- die Wartbarkeit wird mit der Zeit immer schwerer
- Skalierbarkeit nur durch Scale Up und Scale Out möglich
- Umfangreiche Regressionstests
- großes und langsames Deployment





# DESIGN WITH CONWAY'S LAW IN MIND

- **Loosely coupled Applikationen:**

- haben ihre eigene UI, business Logic und Datenbank
- sollen unabhängig deploybar sein
- gehören jeweils einem einzigen Team

- **Cross-functional Teams mit **end-to-end ownership** Mentalität**

**X Project thinking:** Ein Team liefert am Ende eine Stück Software und wird dann aufgelöst

**✓ Product thinking:** Ein Team besitzt ein Produkt während seiner gesamten Lebensdauer  
(Amazon culture: „**you build it, you run it**“)

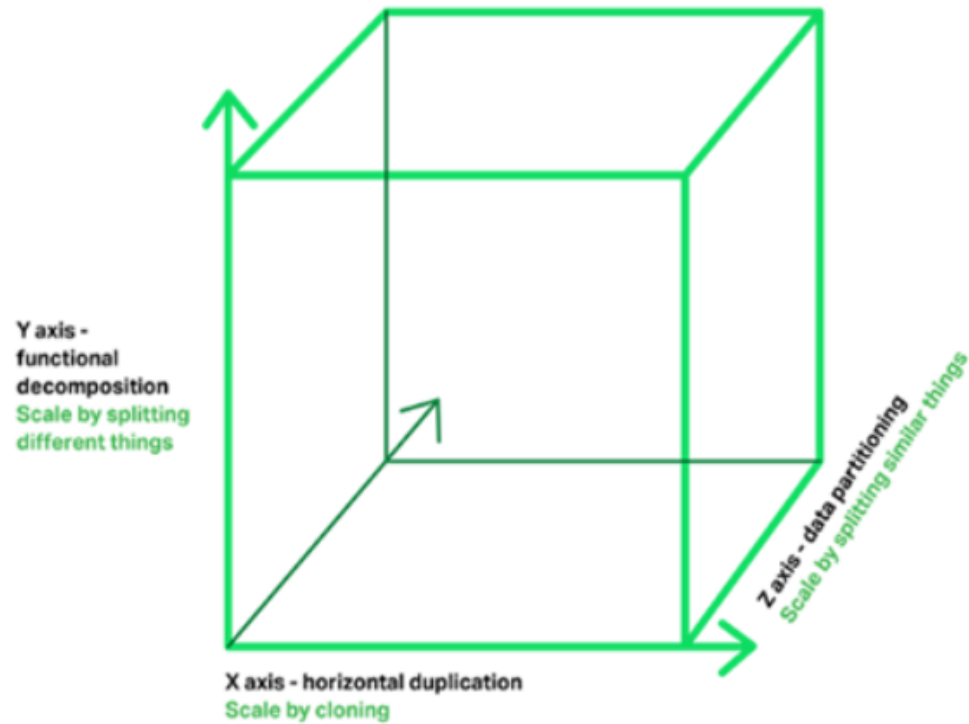


# AGENDA

- Conway's Law
- The Monolith as an example
- **The Scale Cube**
- Bounded Context
- Self-Contained Systems
- Integrating Self-Contained Systems
- Chaos engineering in Netflix



# THE SCALE CUBE



Quelle: Distributed Computing in Microservices: CAP Theorem 

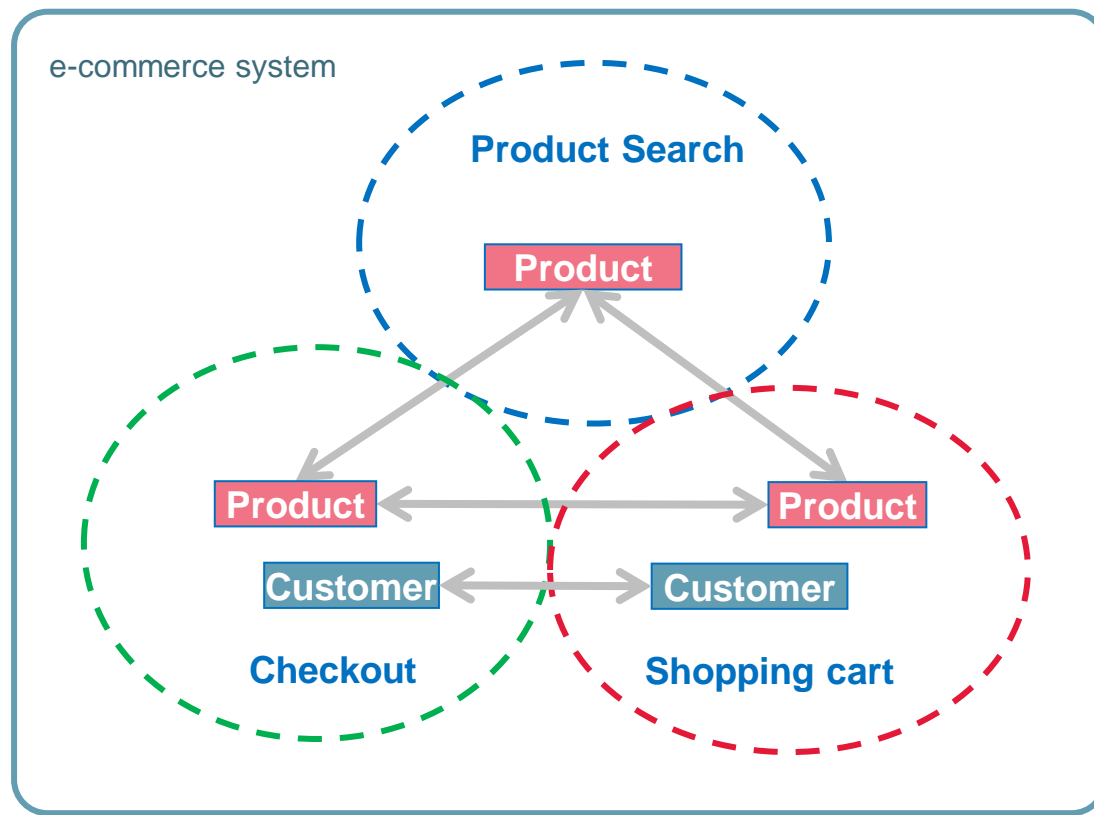
# AGENDA

- Conway's Law
- The Monolith as an example
- The Scale Cube
- **Bounded Context**
- Self-Contained Systems
- Integrating Self-Contained Systems
- Chaos engineering in Netflix



# BOUNDED CONTEXT(Y-AXIS SCALING)

- **Bounded context (DDD):** Ein **Domain Model** kann sich im **Kontext** von mehreren **business domains** befinden. Oft finden hand-overs zwischen diesen Kontexten statt.



# CANDIDATES FOR BC?

- **Non-critical and fairly loosely coupled features:** Diese Features können dupliziert werden und mit Hilfe einer **feature-toggle** neben den anderen Features aus dem Monolith koexistieren
  - **New features**
- Die loosely coupled Features plus die neuen sollen in einem neuen System integriert werden, das parallel zum alten System weiterentwickelt wird.



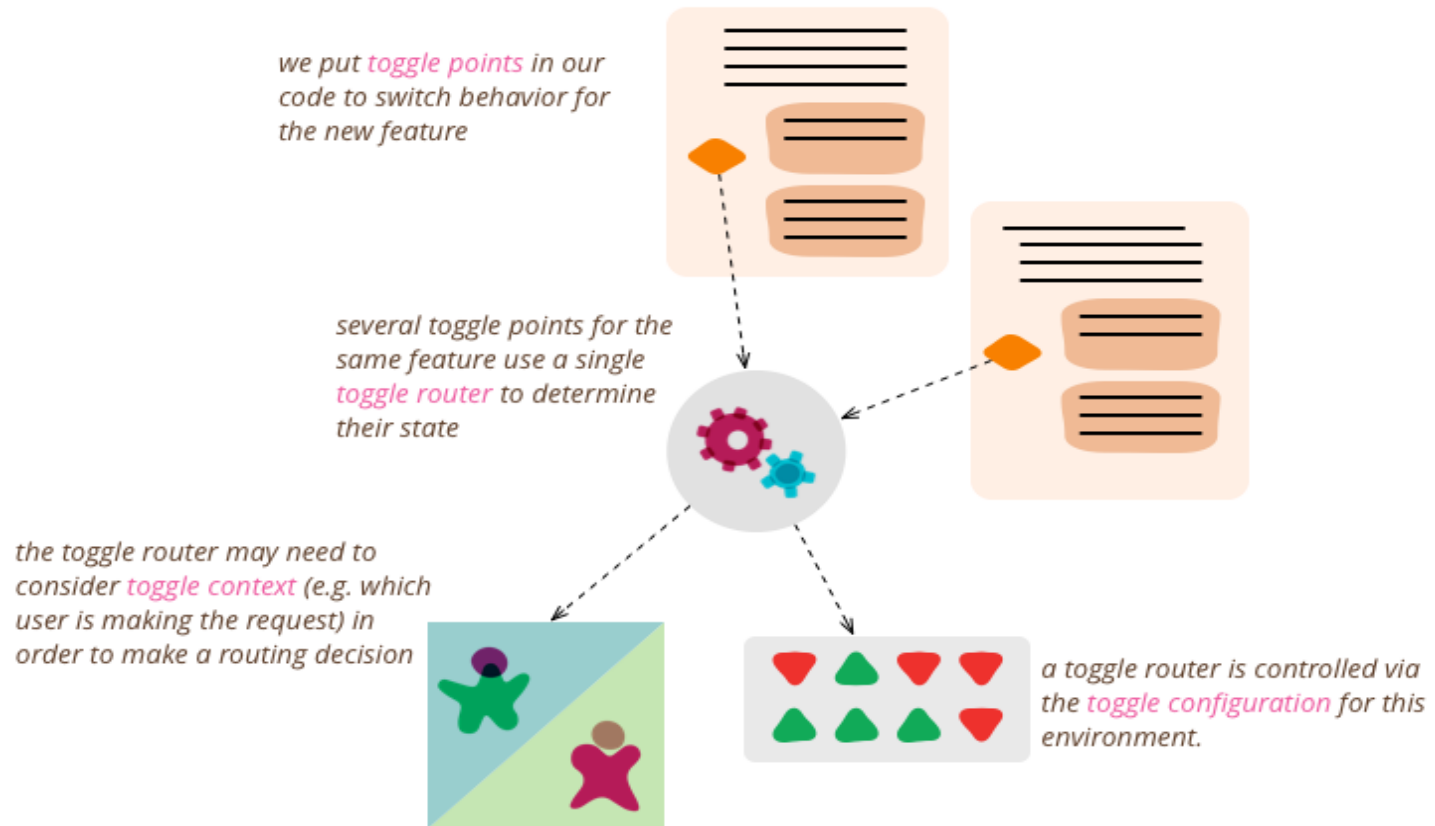
# FEATURE TOGGLES

**Feature Toggles** (often also referred to as Feature Flags) are a powerful technique, **allowing teams to modify system behavior without changing code**. They fall into **various** usage **categories**, and it's important to take that categorization into account when implementing and **managing toggles**. Toggles **introduce complexity**. We can keep that complexity in check by using smart toggle implementation practices and appropriate **tools to manage our toggle configuration**, but we should also aim to **constrain the number of toggles in our system**.

Quelle: Pete Hodgson: Feature Toggles (aka Feature Flags) 



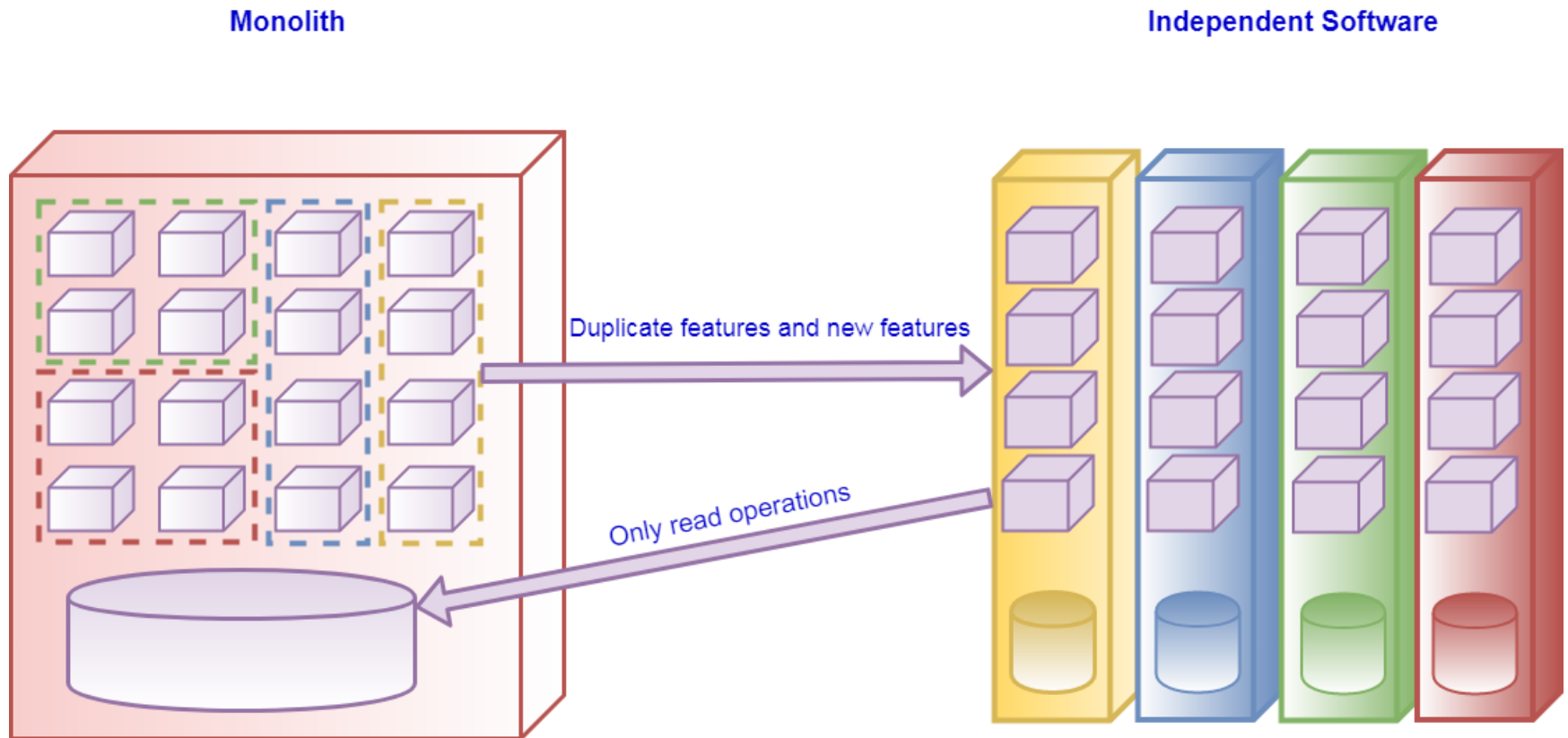
# FEATURE TOGGLES



Quelle: Pete Hodgson: Feature Toggles (aka Feature Flags) 

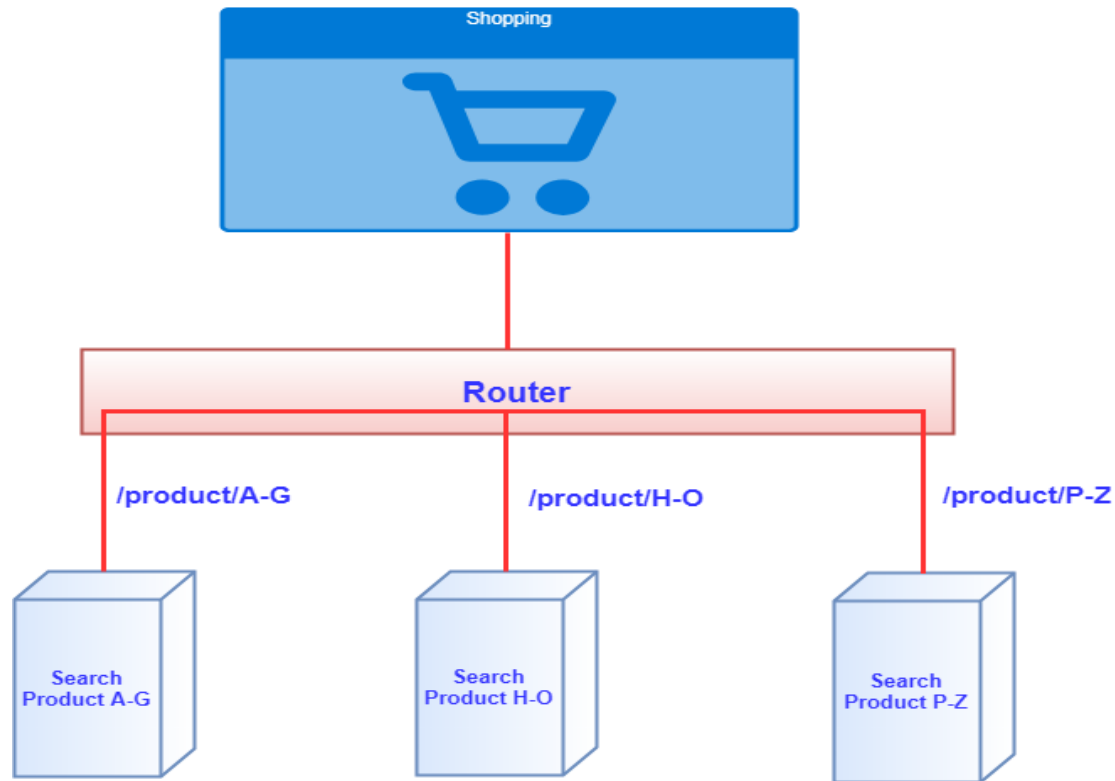


# WELL DEFINED BOUNDED CONTEXTS



# Z-AXIS SCALING

- **Sharding Database or Data Partitioning:** Wird benutzt falls ein Service z.B. „Product Search“ eine große Datenmenge bearbeiten muss.



# AGENDA

- Conway's Law
- The Monolith as an example
- The Scale Cube
- Bounded Context
- **Self-Contained Systems**
- Integrating Self-Contained Systems
- Chaos engineering in Netflix



# SELF-CONTAINED SYSTEMS (SCS)

- sind **autonome Webapplikationen** und haben eigene UI, business logic und datenbank
- Ein SCS gehört einem einzigen Team (**Amazon's Two-Pizza Team rule**)
- No shared UI
- No shared business code
- Integration über UI Schicht
- Änderungen können lokal in einem SCS erfolgen
- Können optional ein API anbieten
- Keine Abhängigkeiten zwischen den Teams und den SCS

➤ **Klare Definition**



# AGENDA

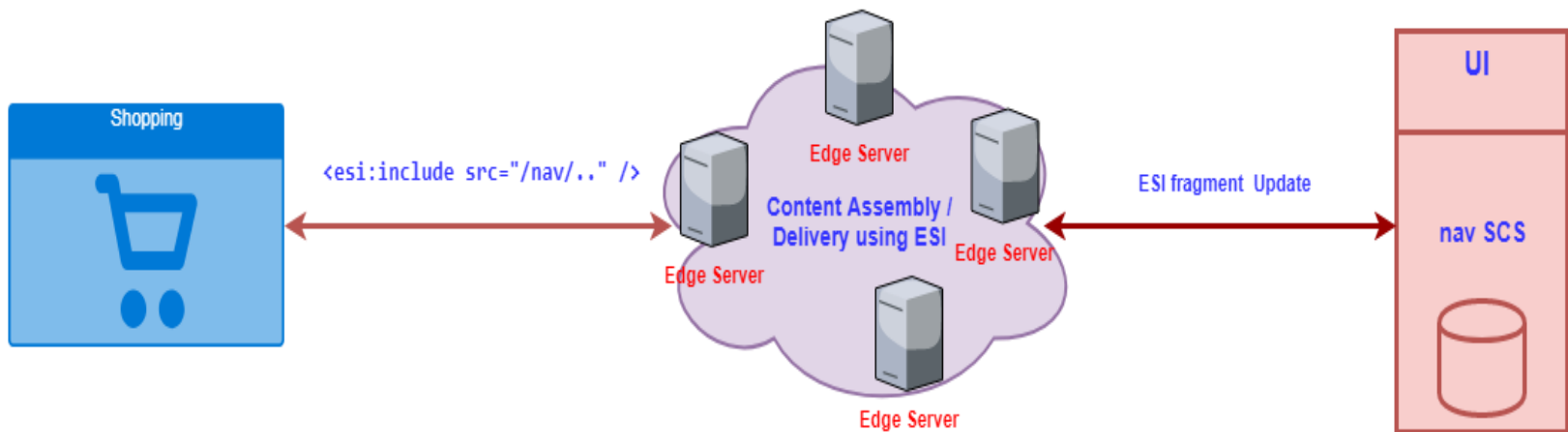
- Conway's Law
- The Monolith as an example
- The Scale Cube
- Bounded Context
- Self-Contained Systems
- **Integrating Self-Contained Systems**
- Chaos engineering in Netflix



# INTEGRATION ON THE UI LEVEL

## ■ Transclusion über Edge-Side Include(ESI):

- z.B. die „**Navigation**“ und die „**Shopping cart**“ in allen Seiten integrieren
- Edge Server: Ein Reverse Proxy oder ein Webserver
- Laden von JS und CSS in response ist möglich
- Cross-Domain ist möglich
- bietet die Möglichkeit zum **failover**, falls der aufgerufene SCS nicht erreichbar ist



# INTEGRATION ON THE UI LEVEL

- **Client Side Transclusion:**

- z.B. AJAX für personalized data
- Laden von JS und CSS möglich? Durch extra Frameworks z.B. [little-loader](#) für JS and [loadCSS](#) ?
- Cross-Domain nur falls CORS möglich

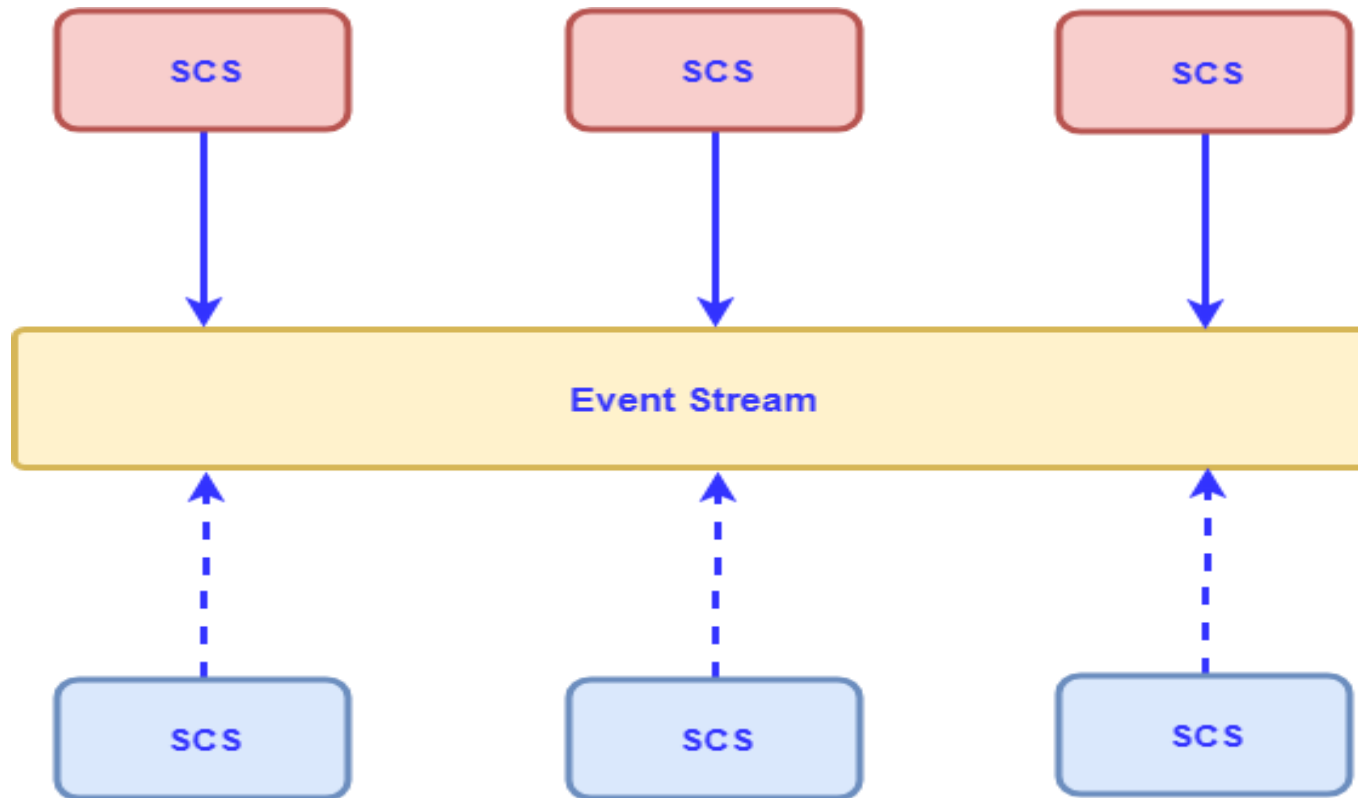


- SCS müssen in bestimmten Situationen miteinander Kommunizieren
- **BESSER** asynchrone Kommunikation





# EVENT-DRIVEN ARCHITECTURE



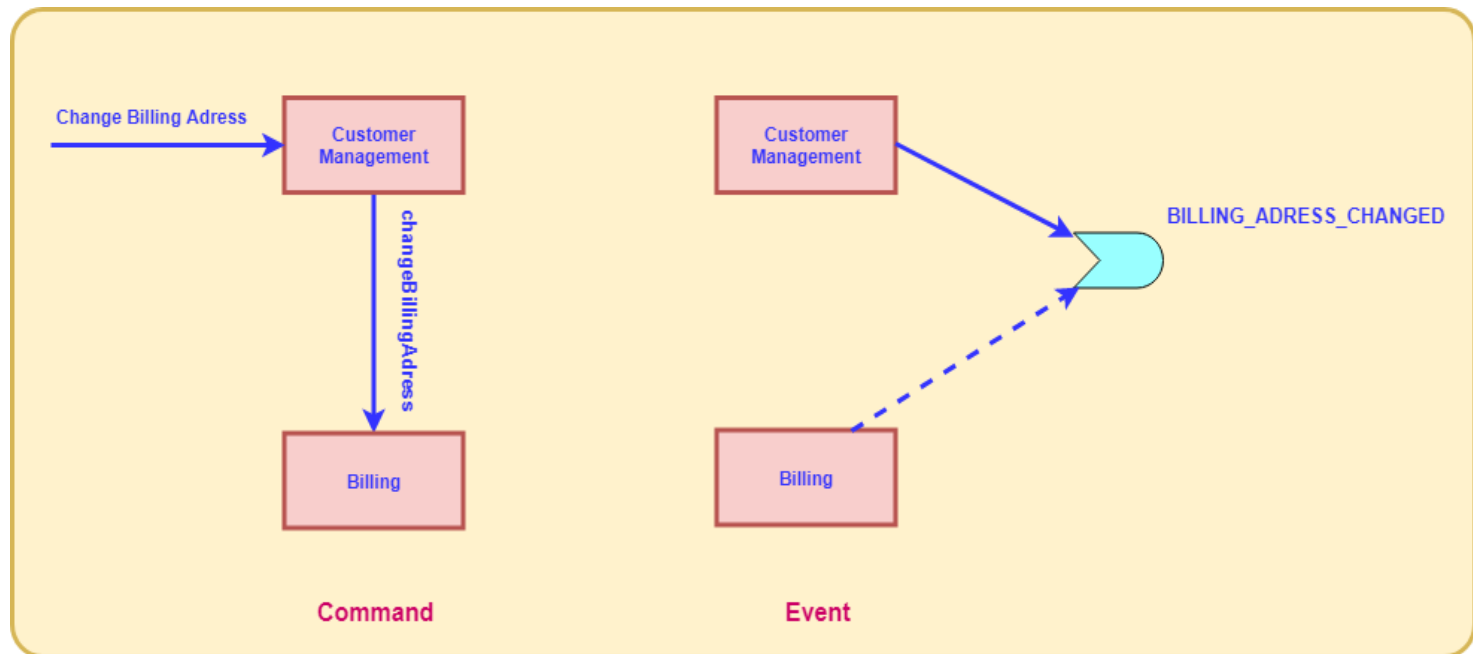
# EVENT-DRIVEN ARCHITECTURE

- **Asynchron:** Der Publisher wartet nicht, bis der Consumer das Event bearbeitet
- **Self-contained Event:** Event enthält alle relevanten Daten, die von den Consumer benötigt werden, um das Event zu bearbeiten.
- **Consumer responsibility:** verwalten die Daten, die sie brauchen und erfragen sie nicht beim anderen Service.
- **Idempotent Consumer:** Falls ein Event mehrfach gelesen wird, wird es nicht zweimal berücksichtigt.
- **Immutable Event data:** Die Daten in einem Event werden nicht geändert, bei jeder Statusänderung wird ein neues Event erzeugt.





# EVENT NOT COMMAND

- ✗ Command:** Mit einem Command erwartet der Sender, dass **ETWAS** passiert. Das hat den Nachteil, dass der Sender und (die) Empfänger **tightly coupled** sind.
- ✓ Event:** Fire and forget Princip. Der Sender erzeugt ein Event und ist nicht daran interessiert, ob **ETWAS** passiert



# WHO IS USING IT?

- **Otto**

- Otto-Kennzahlen 
- 11 SCS: Product, Order, Backoffice usw.(Quelle: On Monoliths and Microservices ) 


- **Zalando:**

„ ...We aim to develop **autonomous isolated services** that can be independently deployed and that are **centered around defined business capabilities**...” 

- **KUHN+NAGEL**

- Starring as "the monolith" 

- **IKEA**

- **SCS:** Editorial Content, Search, Products, Checkout, Whishlist 




# AGENDA

- Conway's Law
- The Monolith as an example
- The Scale Cube
- Bounded Context
- Self-Contained Systems
- Integrating Self-Contained Systems
- **Chaos engineering in Netflix**



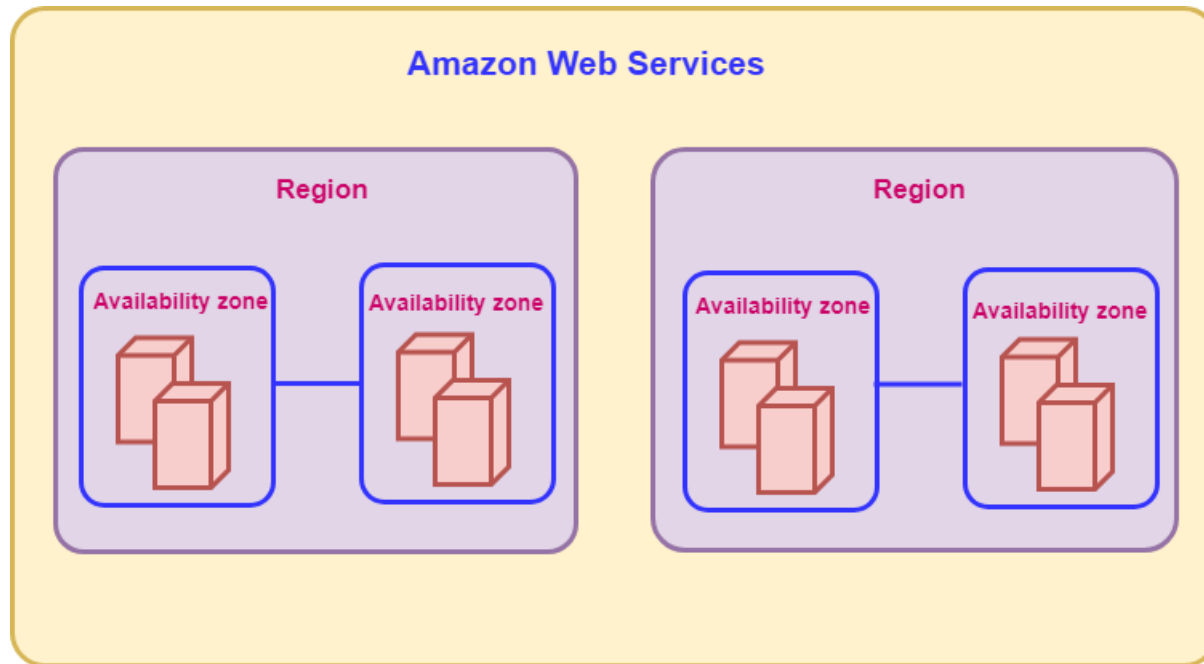
# NETFLIX SIMIAN ARMY

- **Chaos Engineering:** Is the discipline of experimenting on a distributed system in order to build confidence in the system's capability to withstand turbulent conditions in production.
- **Simian Army:** Consists of services (Monkeys) in the cloud for generating various kinds of failures, detecting abnormal conditions, and testing our ability to survive them. The goal is to keep our cloud safe, secure, and highly available.
- Simian Army is part of the Netflix open Source cloud platform 



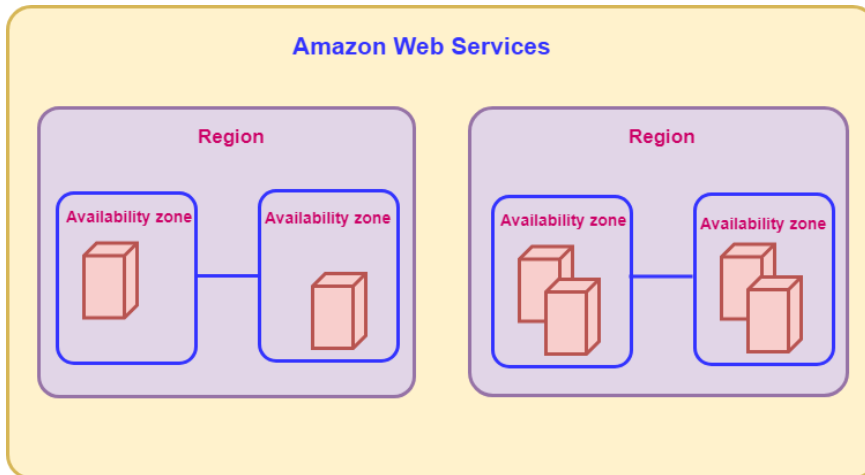
Quelle: <https://github.com/Netflix/SimianArmy>

# REGION AND AVAILABILITY ZONE CONCEPT



# CHAOS MONKEY

- is a service which identifies groups of systems and randomly terminates one of the systems in a group
- only operates during business hours
- in a carefully monitored environment with engineers standing by to address any problems

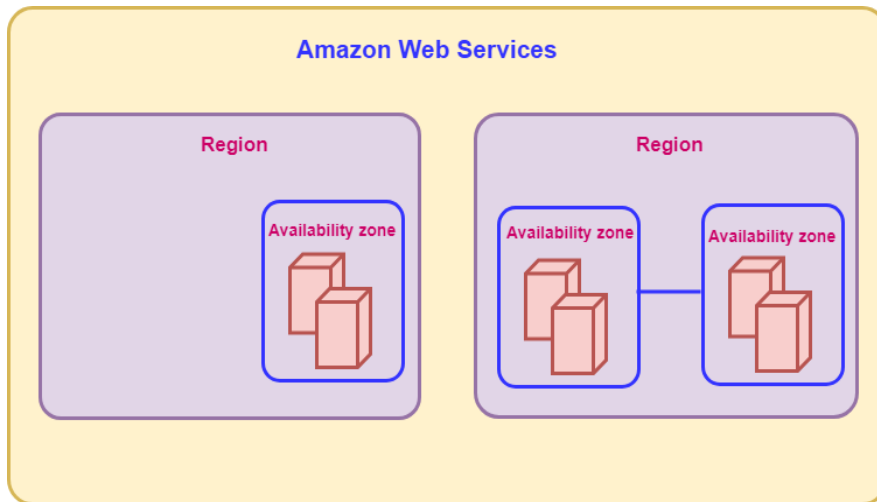


Quelle: <https://github.com/Netflix/SimianArmy>



# CHAOS GORILLA

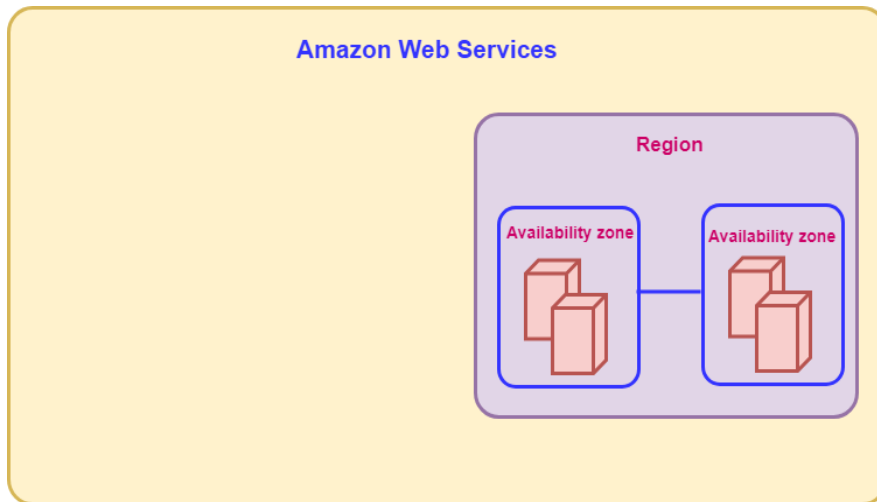
- is a service which simulates an outage of an **entire Amazon availability zone**
- only operates during business hours
- 3-zone configuration
- ensure that other zone can handle the load without any impact to user experience



Quelle: <https://github.com/Netflix/SimianArmy>

# CHAOS KONG

- is a service which simulates an outage of an **entire Amazon Region**
- only operates during business hours
- Ensure that other Region can handle the load without any impact to user experience



Quelle: <https://github.com/Netflix/SimianArmy>

# REFERENZEN

- On Monoliths and Microservices
- Self Contained Systems (SCS): Microservices Done Right
- From Monoliths to Microservices: An Architectural Strategy
- Self-contained Systems: A Different Approach to Microservices
- Self Contained Systems
- Microservices, Martin Fowler
- Event-driven Microservices & Event Processing
- Zalando rules of play
- Feature Toggles, Pete Hodgson
- Transclusion

