Microservices & Event-getriebene Architektur (MEA)

WASP 1, SS 2025

Prof. Dr.-Ing. Stefan Bente

Motivation: Wofür Microservices?

Technology
Arts Sciences
TH Köln

Warmup

& Motivation

Bild: Jeff Knezovich, https://www.flickr.com/photos/knezovjb/2102314562

Agenda

- Ein Aspekt … Größe & Komplexität
- Eng gekoppelter Architekturstil: Monolith
- Der nächste wichtige Aspekt … Geschwindigkeit
- Lose gekoppelter Architekturstil: Microservices

Ein Aspekt …

Größe & Komplexität

https://ahaslides.com/J3HBK

Wie übersetzt sich „# Klassen“ in „SLoC“

- SLoC = Source Lines of Code
  - Häufig verwendete Metrik zum Messen der Größe von Software
  - Auch gebräuchlich: KLoC (1000 Zeilen Code)

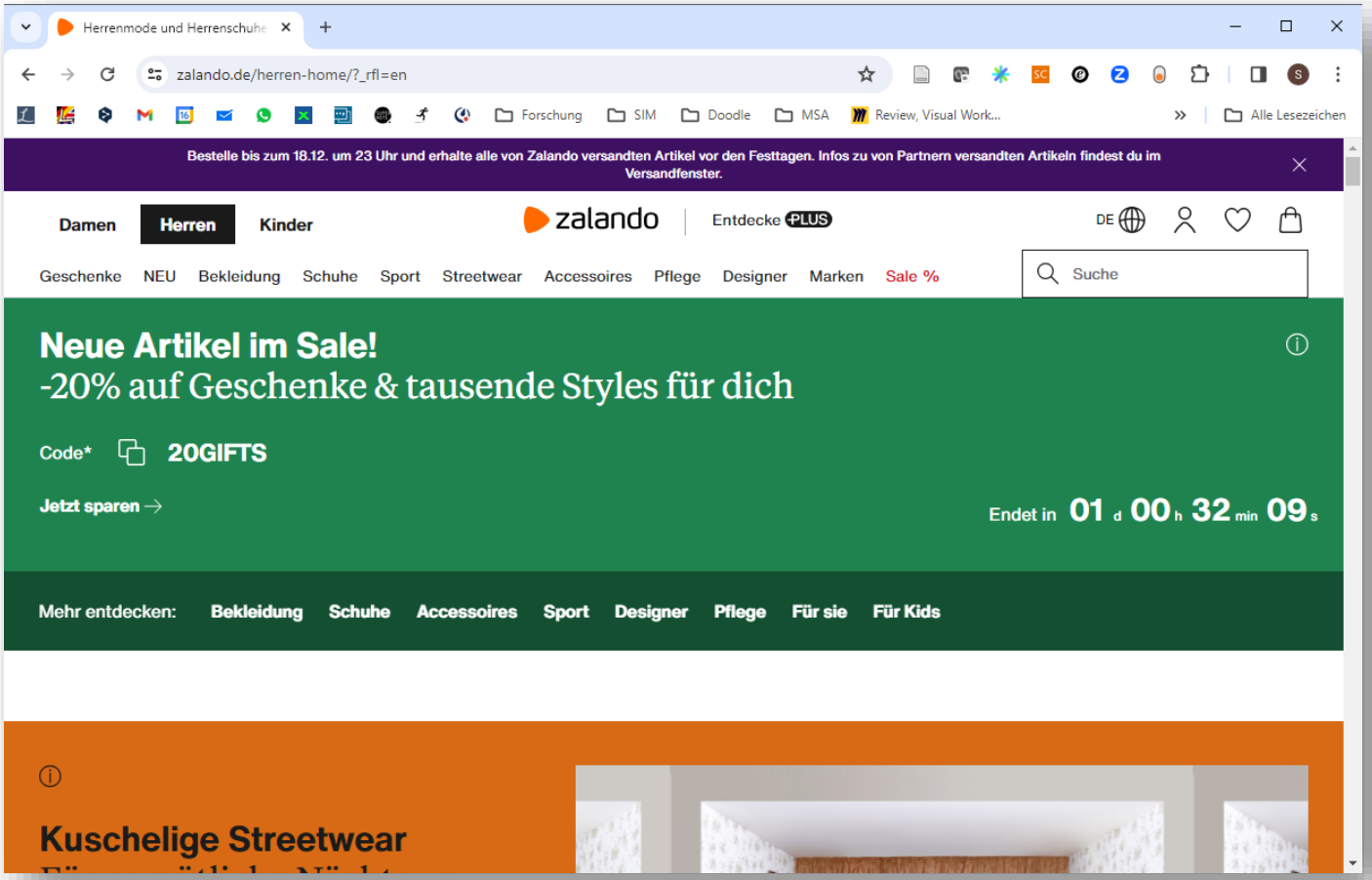
- Zhang & Tan (2007): Mittlere Größe einer (Java-)Klasse =

  114 SLoC

- 5 Klassen ~ 570 SLOC
- 50 Klassen ~ 5700 SLOC
- 500 Klassen ~ 57.000 SLOC

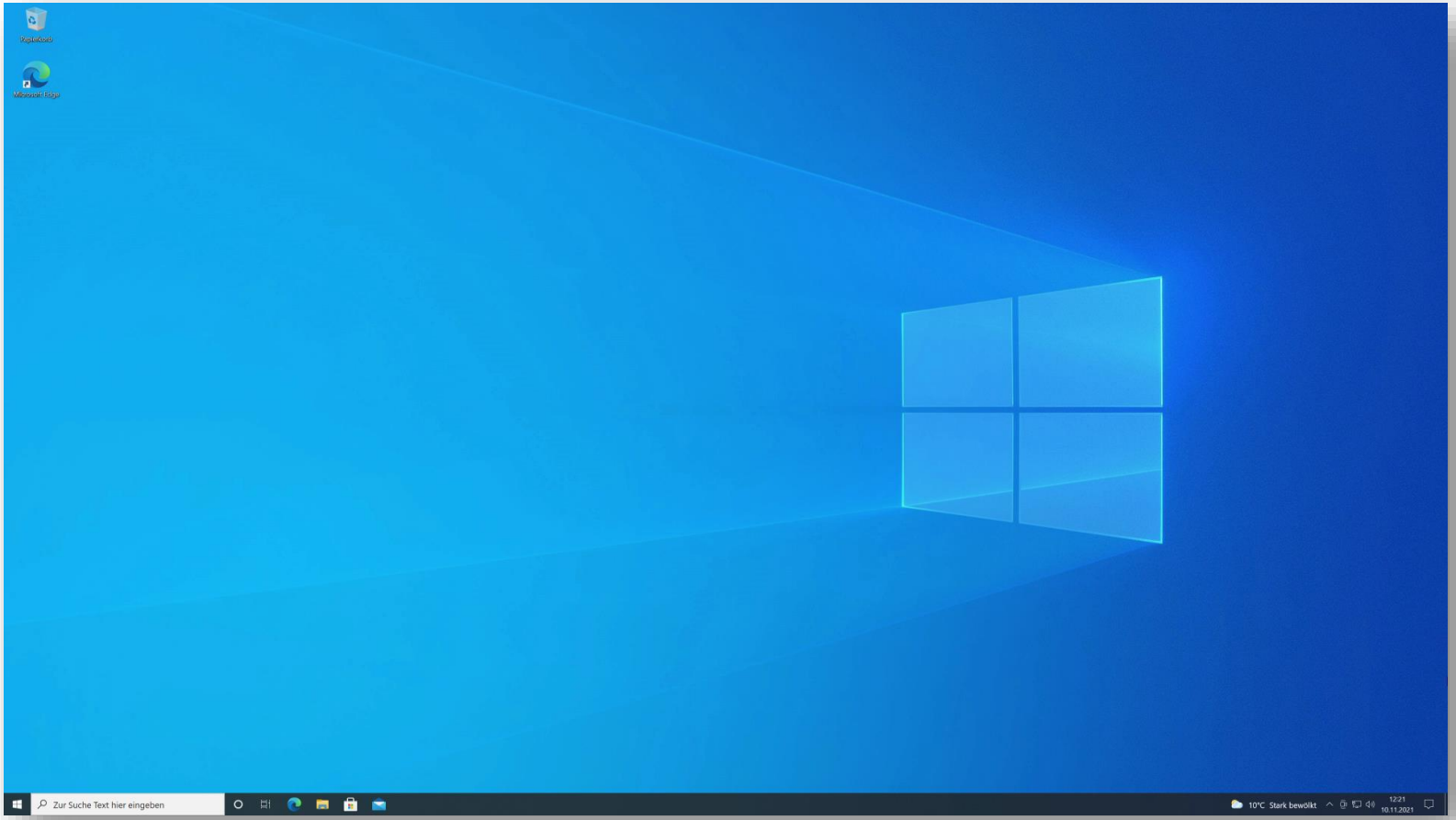
- Source: https://ieeexplore.ieee.org/abstract/document/4425860
  - H. Zhang and H. B. K. Tan, "An Empirical Study of Class Sizes for Large Java Systems," 14th Asia-Pacific Software Engineering Conference (APSEC'07), Nagoya, Japan, 2007, pp. 230-237, doi: 10.1109/ASPEC.2007.64

Beispiel 1: Zalando Logistics System (Zalos)



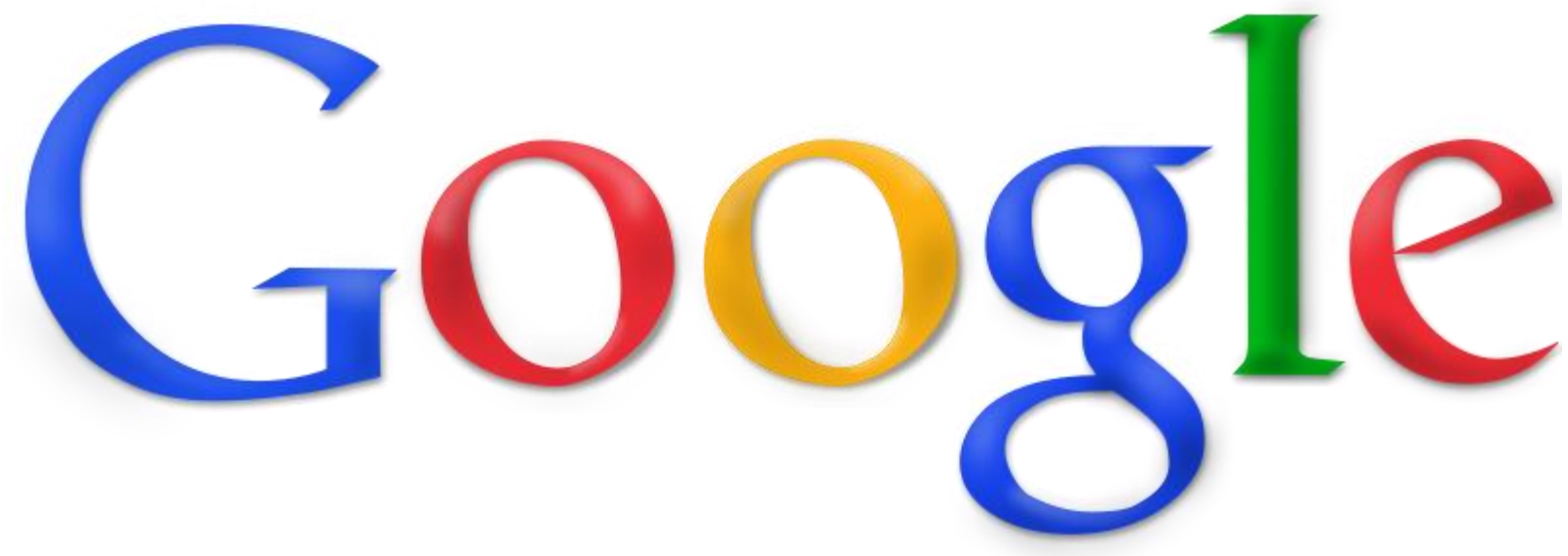
Zalos = Lager- und Logistik-Verwaltung von Zalando

Beispiel 2: Windows 10



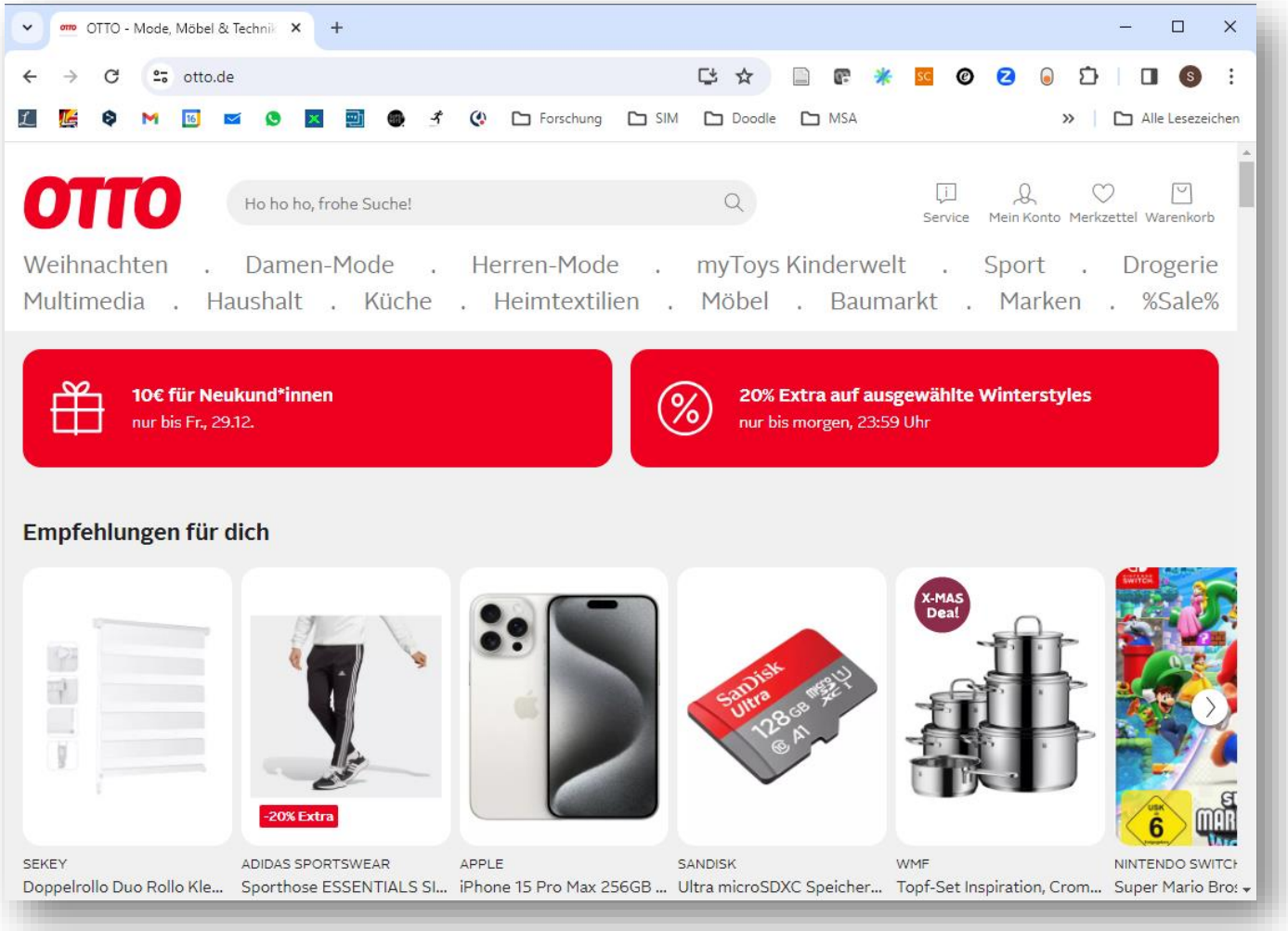
Gesamtheit des Betriebssystem-Codes

Beispiel 3: Google Monorepo



(alle Google Services, Front- & Backend, ohne Google Chrome und Android)

Beispiel 4: Otto eCommerce-Plattform



Open Source Kernplattform

Beispiel 5: Apollo-Mondlandung 1969

Software auf allen Computern
in Rakete und Landemodulen

https://ahaslides.com/J3HBK

Wie viele Zeilen Code …

- 145.000 - **Apollo 11**
  - https://www.synopsys.com/blogs/software-security/apollo-11-software-development.html
- 710.000 - **Otto.de Plattform**
  - Fürstenau, D., Rothe, H., Baiyere, A., Schulte-Althoff, M., Masak, D., Schewina, K., & Anisimova, D. (2019). Growth, Complexity, and Generativity of Digital Platforms: The Case of Otto.de. *Fortieth International Conference on Information Systems*. https://aisel.aisnet.org/wi2019/track07/papers/10
- 1,3 Mio - **Zalando Logistics System (Zalos)**
  - https://engineering.zalando.com/posts/2018/04/migrating-java-8.html
- 50 Mio. - **Windows 10**
  - 3 Mio. - Kernel: 3 Mio
  - 25 Mio. - Applications (Office, Outlook, Store, …)
  - 22 Mio. – 3rd Party (Treiber, …)
  - https://softkeys.uk/blogs/blog/how-many-lines-of-code-in-windows-10
- 2 Mrd. - **Google Monorepo**
  - https://qeunit.com/blog/how-google-does-monorepo

Produktivität von Entwicklern – LoC / day

- Gesamtzahl aggregiert über längeren Zeitraum
  - Inklusive Tests, Doku, Bugfixing, Team-Events, Smalltalk, Training, sinnlose Meetings, …

https://ahaslides.com/J3HBK

Produktivität von Entwicklern – LoC / day

- Gesamtzahl aggregiert über längeren Zeitraum
  - Inklusive Tests, Doku, Bugfixing, Team-Events, Smalltalk, Training, sinnlose Meetings, …

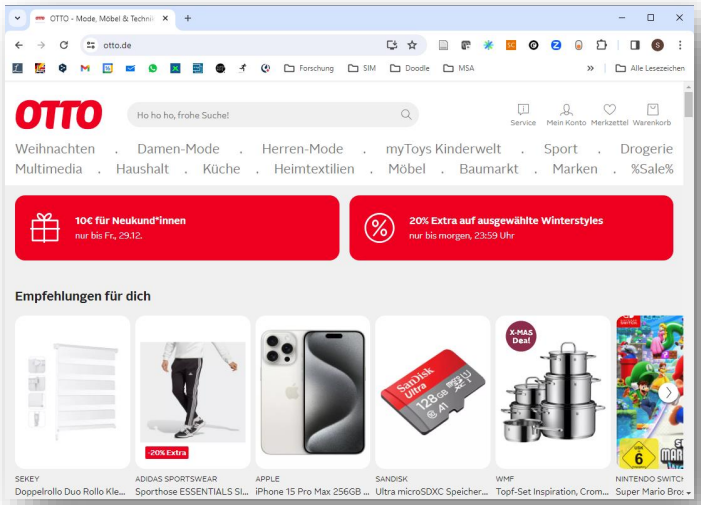
LoC / day	Quelle
10	Fred Brooks, „The Mythical Man Month“
16 … 38	Capers Jones
20 … 125	McConnel, small projects (<= 10.000 LoC)
1,5 … 25	McConnel, large projects (>= 10 Mio LoC)
50	Andy Brice, own data (90.000 – 150.000 LoC systems)

Quelle: Andy Brice (2017), https://successfulsoftware.net/2017/02/10/how-much-code-can-a-coder-code/

- => Wir gehen für die weiteren Überlegungen mal von **50 LoC / day** aus

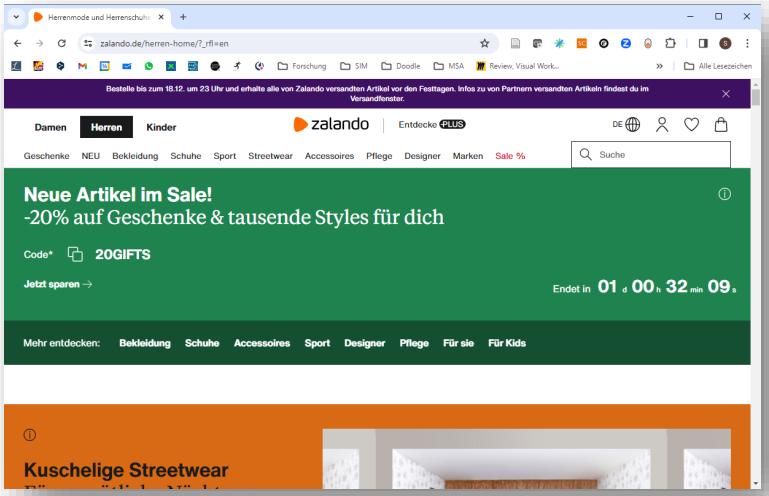
Wie viele Entwickler:innen arbeiten an so etwas?

Otto.de Plattform



- 506 Contributors in 65 repos
- Growth: ~25.000 LoC / month
  - Feb 18 – Feb 19
- => ca. 30 Developer
- => ca. 4-6 Teams

Zalando Zalos



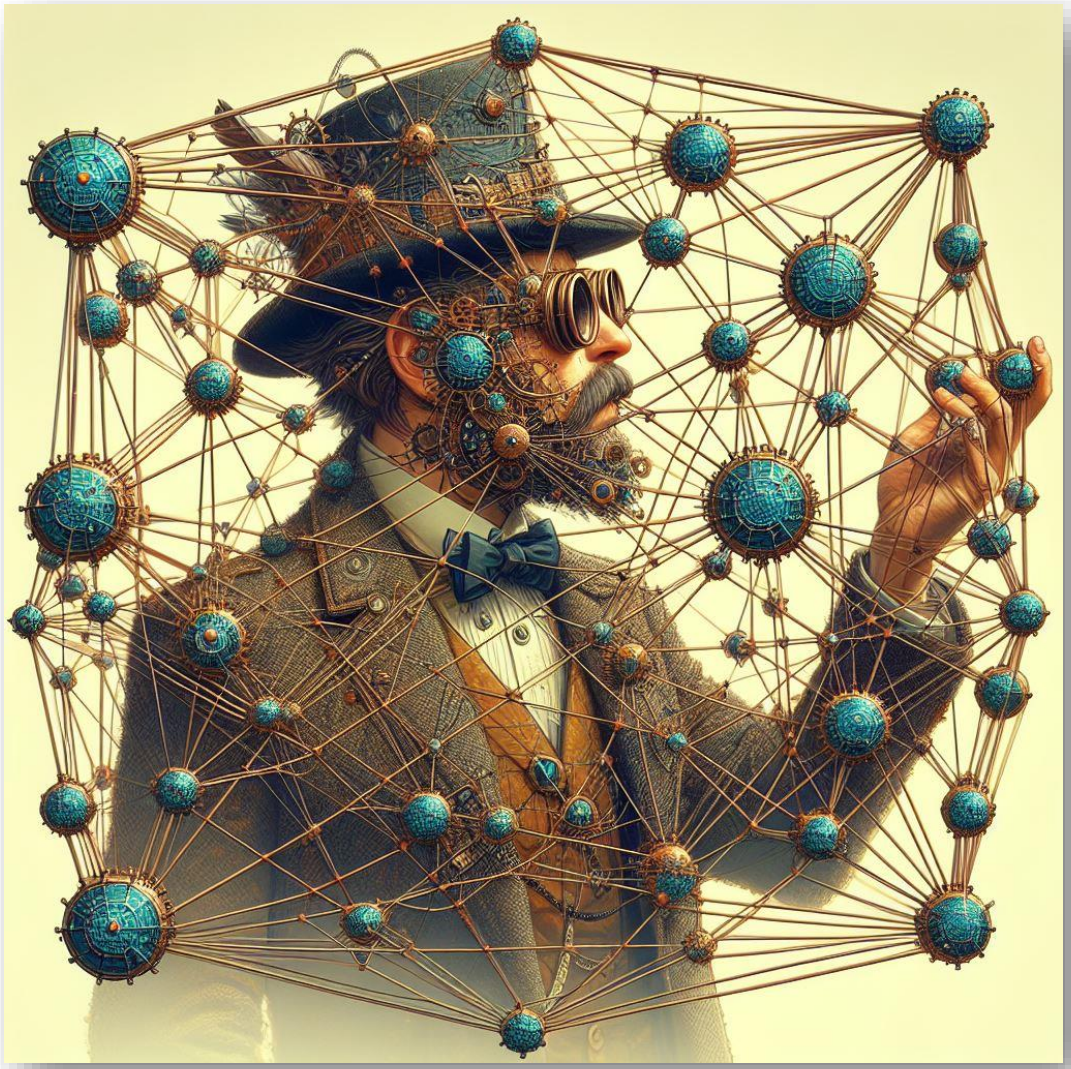
- 70 Developer
  - (Angabe aus der Quelle)
- => ca. 10 Teams

… und wie lange lebt so eine Software?

https://ahaslides.com/J3HBK

- … allein Java ist fast 30 Jahre alt:
  - d.h. die ersten Java-Anwendungen sind schon retired, viele gibt es aber auch noch. Und das sind KEINE Mainframe-Anwendungen.

Abhängigkeiten zwischen den Teams?



Ein praktisches Beispiel

Hochschul-Verwaltungssystem

Ein Portfolio von Hochschul-Software-Systemen

- Nehmen wir einmal an, ein professionelles Software-Team würde ein Portfolio von Hochschul-Software-Systemen entwickeln …
- (Mindestens) die folgenden Systeme würden dabei herauskommen:

Notenverwaltung

Einschreibung

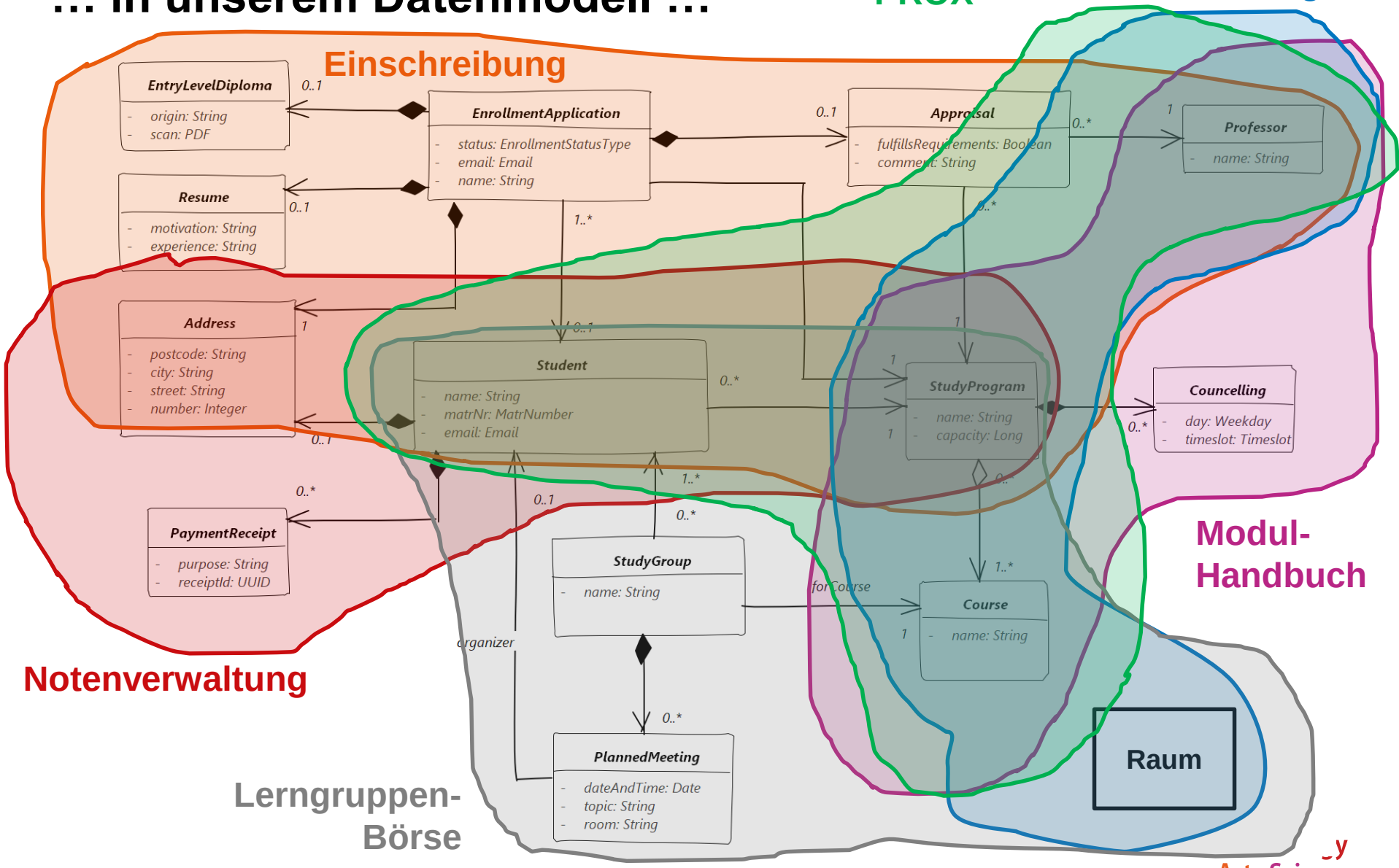
Modul-Handbuch

Stundenplanung

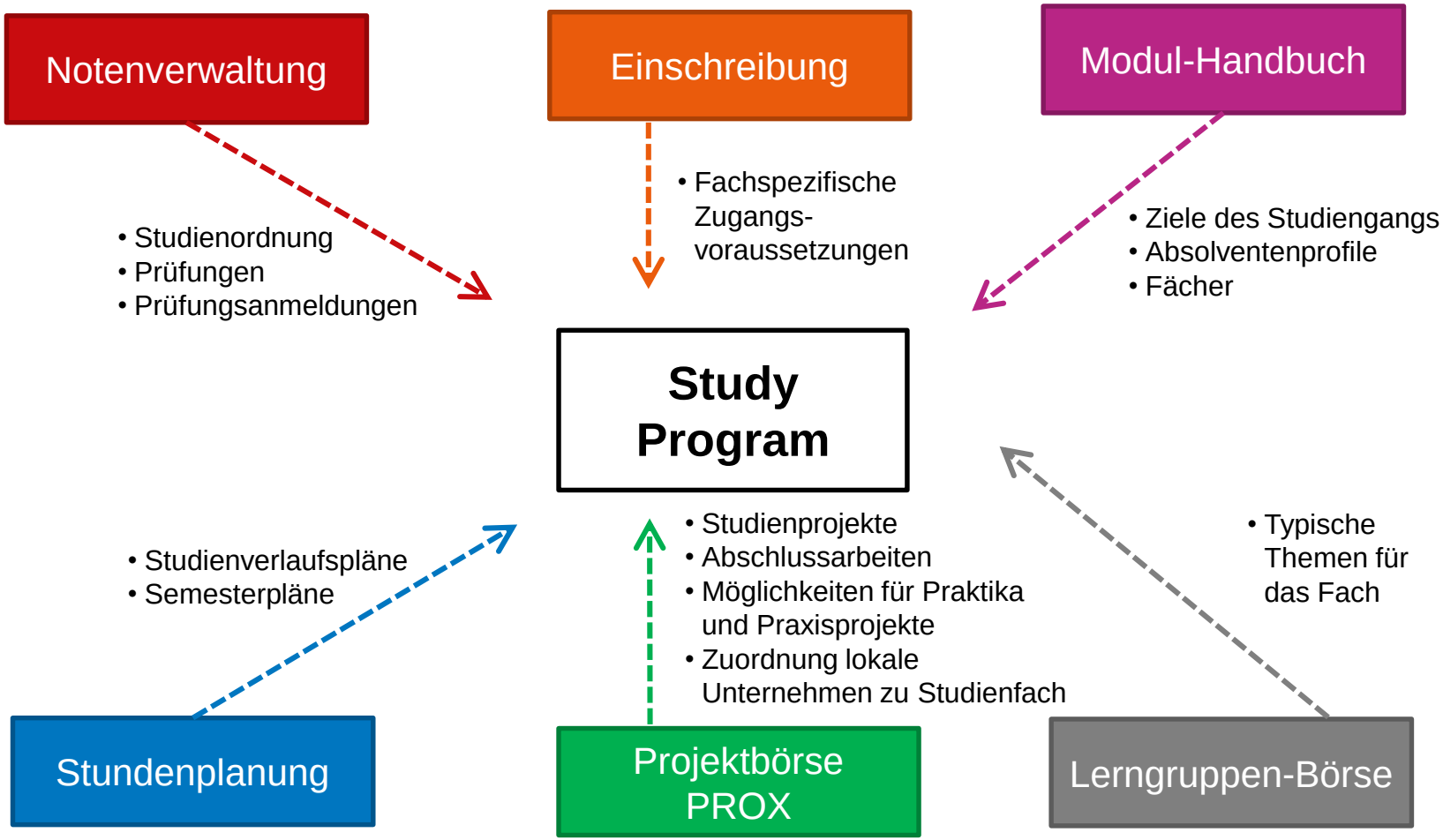
Projektbörse PROX

Lerngruppen-Börse

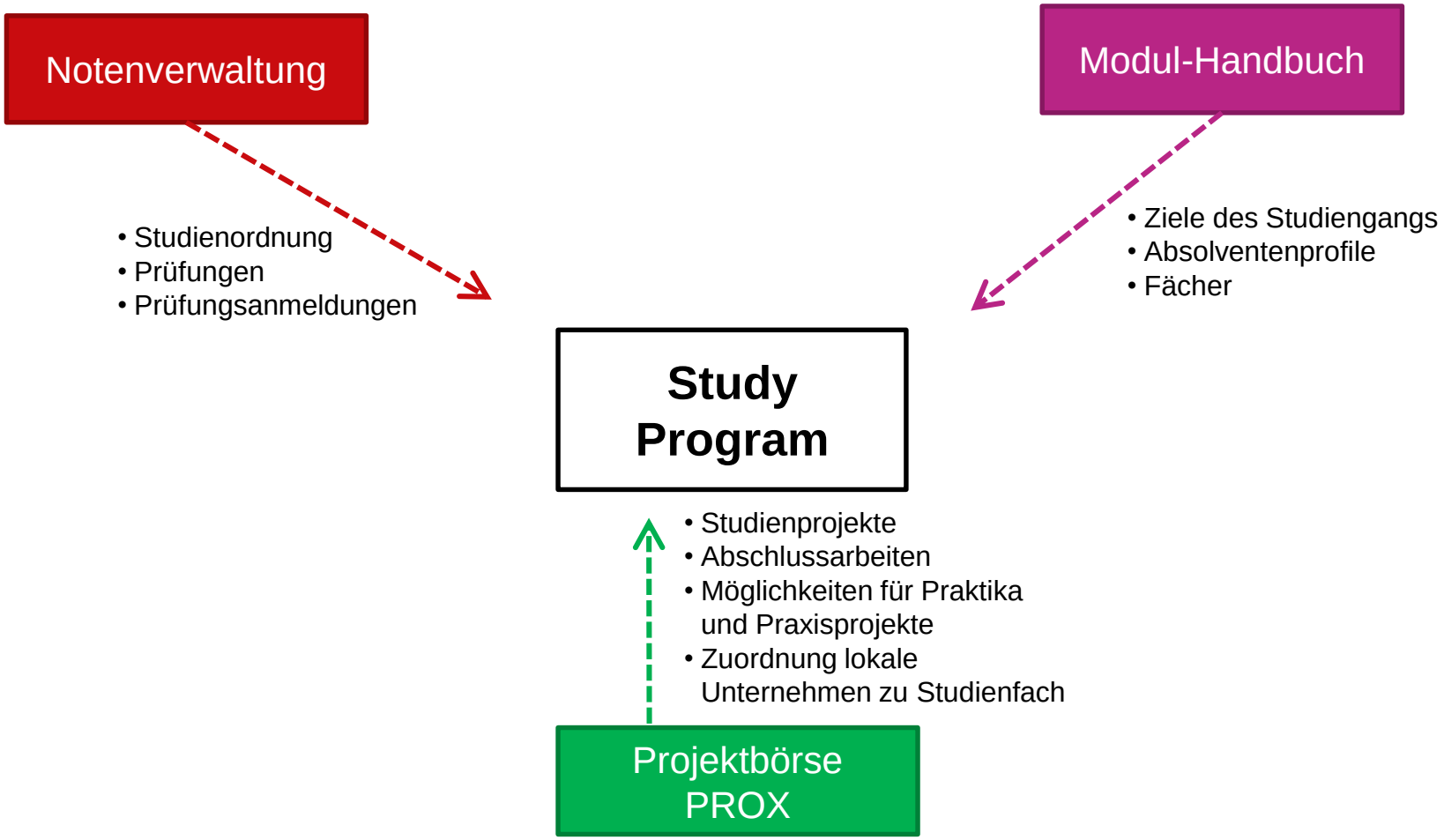
… in unserem Datenmodell …



Bsp: **Jedes System hat eine eigene Sicht auf StudyProgram**



Für Einfachheit: **Wir betrachten nur 3 Systeme weiter …**

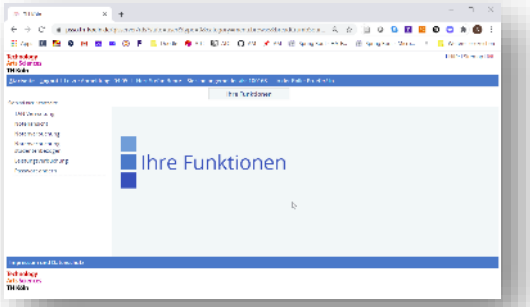


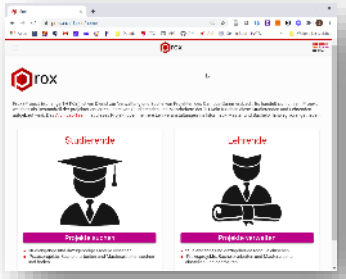
Eng gekoppelter Architekturstil:

Monolith

Könnte man das so implementieren?

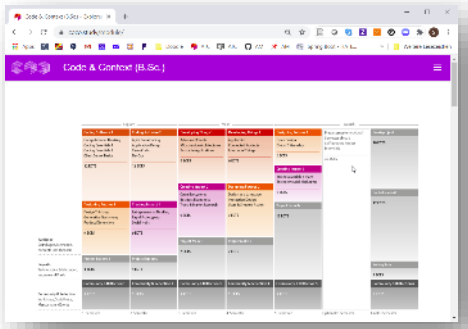
Web-Client für Notenverwaltung



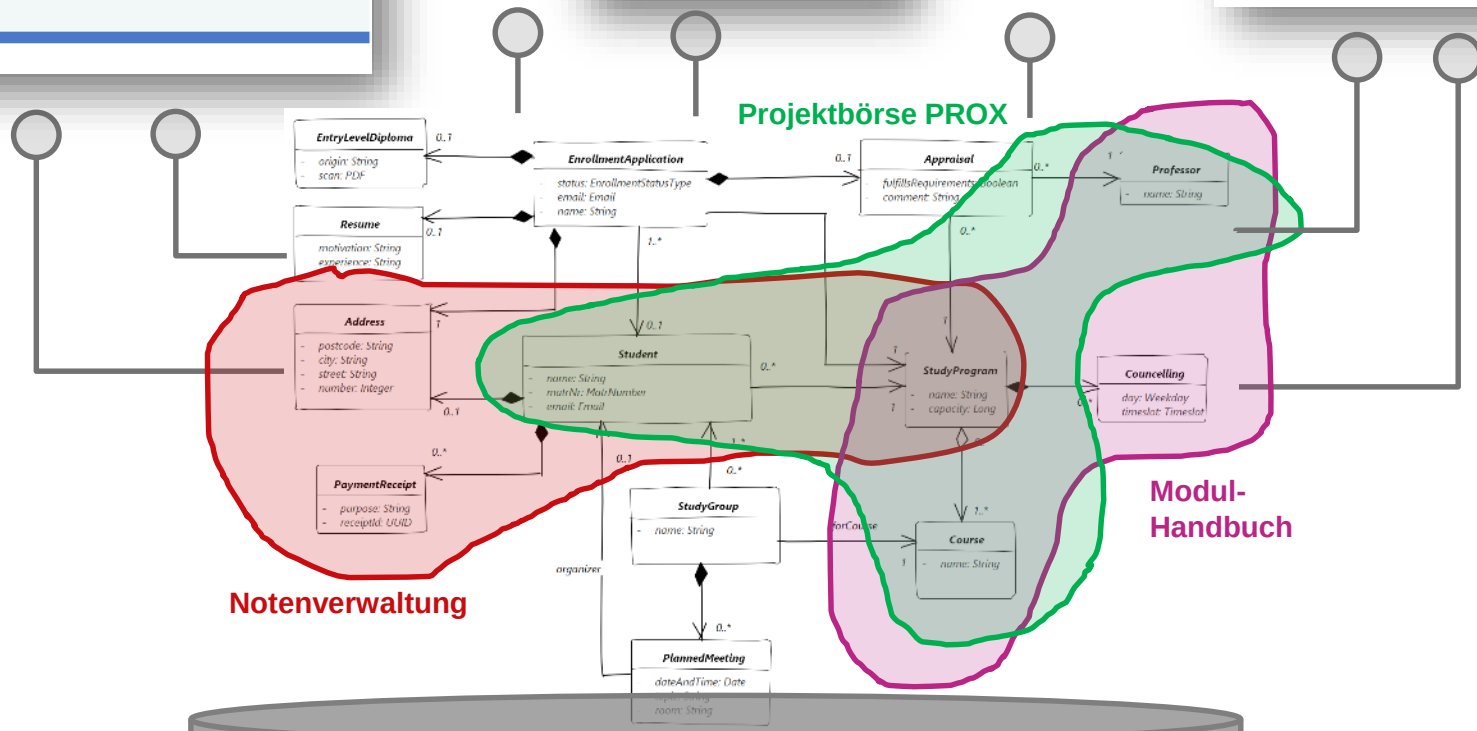


PROX-Client

Web-Client für Modul-Handbuch



Projektbörse PROX



Notenverwaltung

Modul-Handbuch

gemeinsame Datenbank

Technology Arts Sciences TH Köln

Definition: (Software-)Monolith

- Kombination vieler verschiedener fachlicher Teilfunktionen
- Eine große Code-Base
  - üblicherweise in einem Repo
- Nur als Ganzes deployed
- **Nachteile**
- **Vorteile**

https://ahaslides.com/J3HBK

Risiko: „Big Ball of Mud“

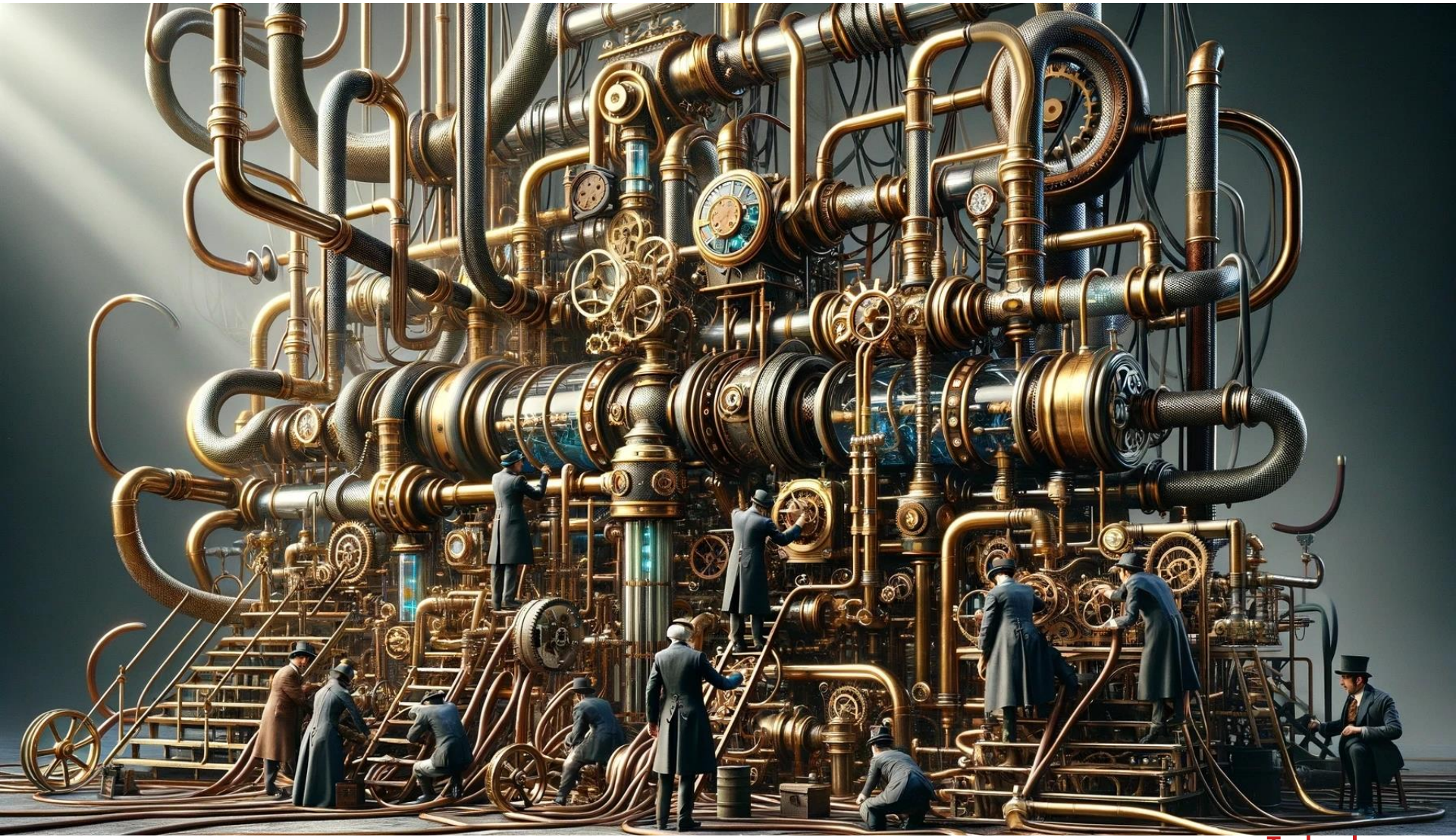


Bild: Dall-E

Risiko: „Big Ball of Mud“ im gemeinsamen Datenmodell

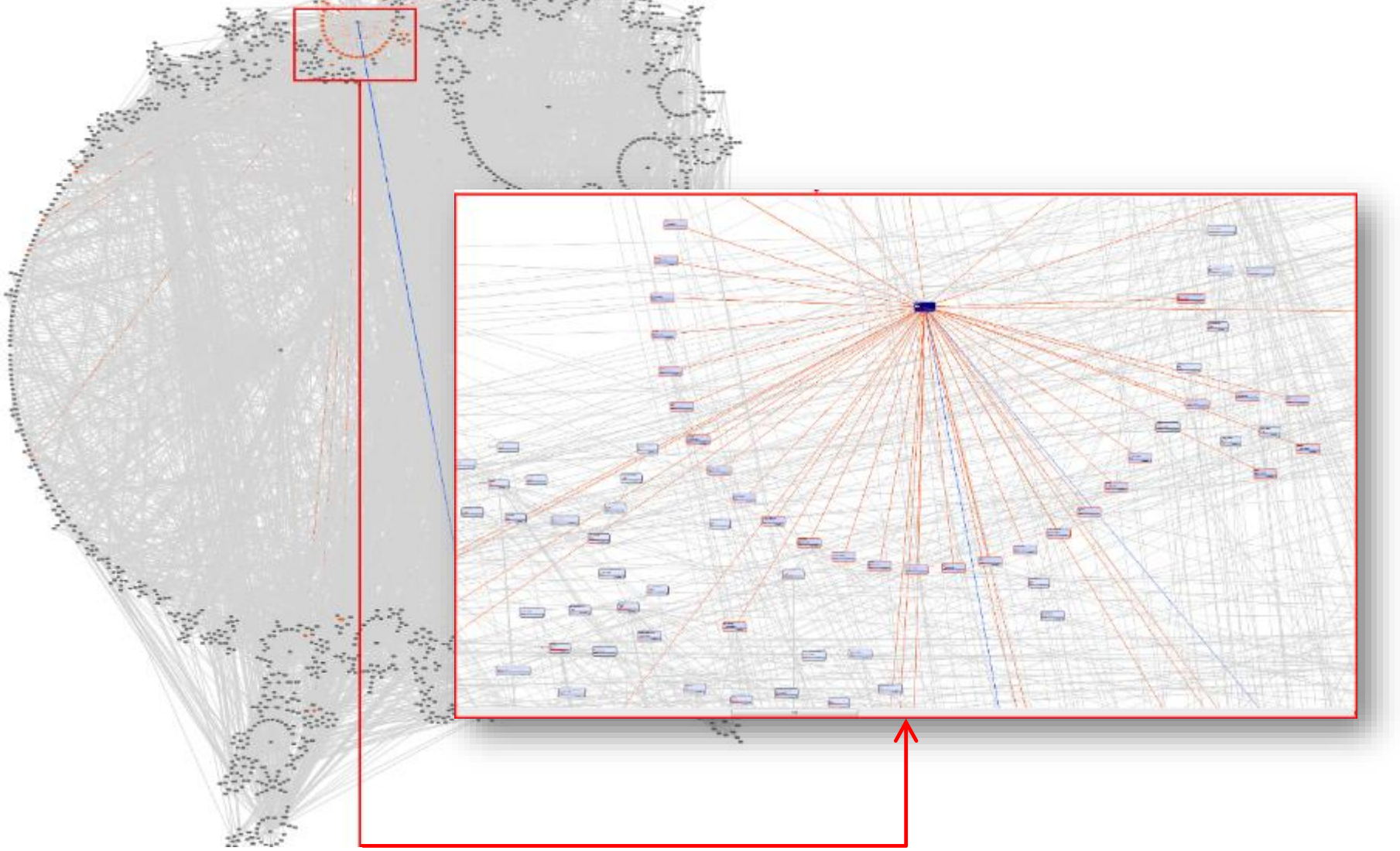


Bild: Remy Porter, https://thedailywtf.com/articles/the-big-balls-of

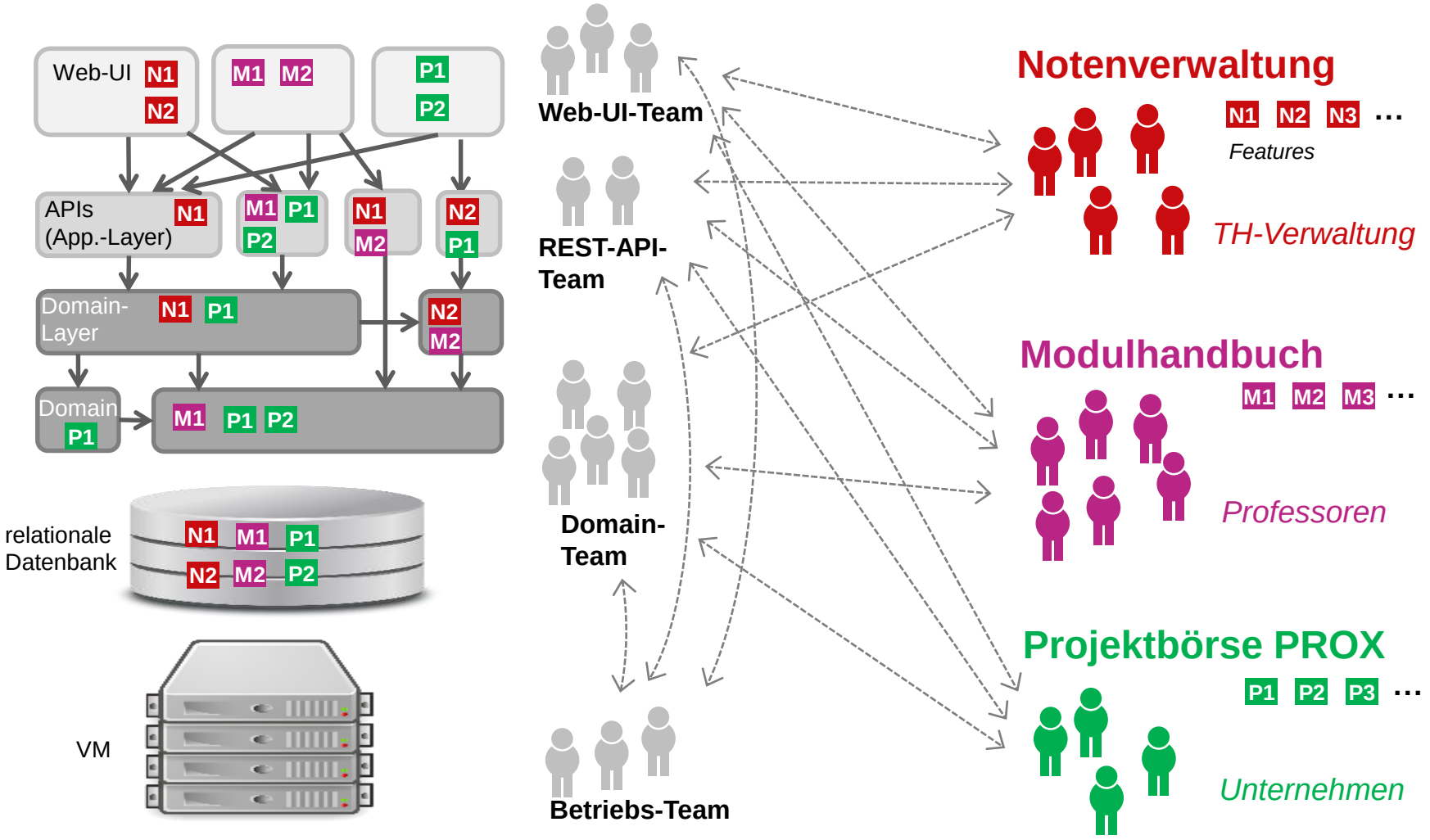
Definition: (Software-)Monolith

- Kombination vieler verschiedener fachlicher Teilfunktionen
  - Eine große Code-Base, üblicherweise in einem Repo
- Nur als Ganzes deployed
- **Nachteile**
  - Kleine Änderungen ziehen oft massive Tests nach sich
  - U.U. zahlreiche Seiteneffekte – man weiß oft nicht, was Änderungen bewirken
    - Daher oft weniger robust
  - Deployment Pipeline ist schwerer zu automatisieren
    - => Komplexes Deployment, oft mit sehr viel manuellem Aufwand (z.B. Testing) verbunden, da immer ALLES deployed werden muss
- **Vorteile**
  - Lokales Dev Env ist einfach(er), Debugging einfach
  - Performance KANN besser sein
  - E2E- und System-Testing relativ gut machbar

Der nächste wichtige Aspekt …

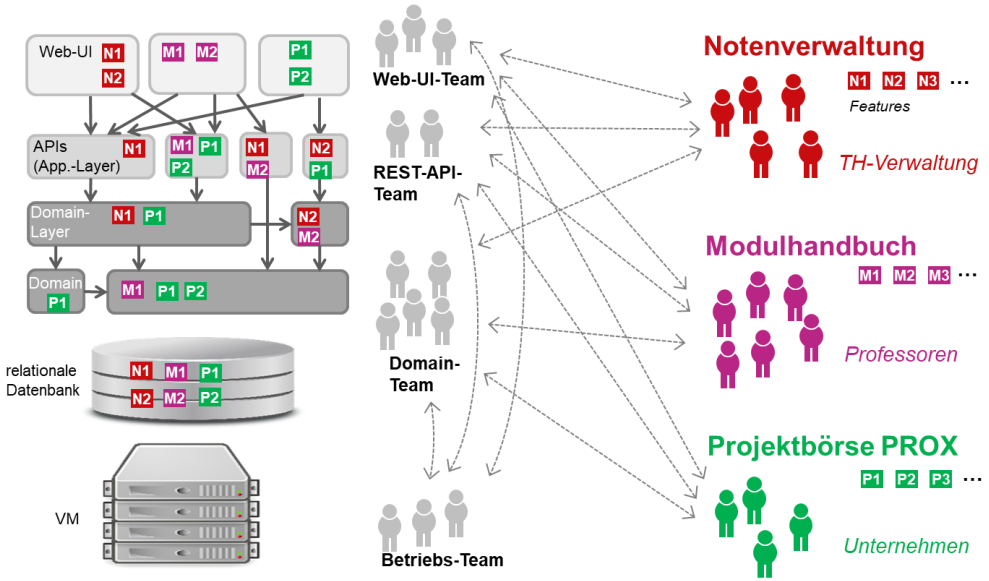
Geschwindigkeit

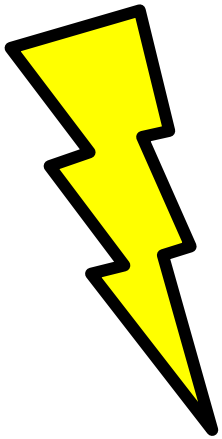
Klassischer (technisch geschnittener) Monolith



https://ahaslides.com/J3HBK

… und die Folgen des Monoliths





Potentielle Innovations-bremse

- Viel Abstimmungsbedarf bei jedem Feature
- Lange Release-Zyklen (i.d.R. <= 4-6 Main Releases pro Jahr)
- Komplexe Integrationstests
- Wenig Flexibilität im Entwicklungs- und Change-Prozess

https://ahaslides.com/J3HBK

Wie oft pro Woche stellt Amazon eine neue Version live?

Bild: Amazon.de, Quelle: Jenkins, Jon. “Velocity Culture - The Unmet Challenge in Ops.” presented at the O’Reilly Velocity Conference 2011, June 16, 2011. http://assets.en.oreilly.com/1/event/60/Velocity%20Culture%20Presentation.pdf.

https://ahaslides.com/J3HBK

Wie oft pro Woche stellt Amazon eine neue Version live?

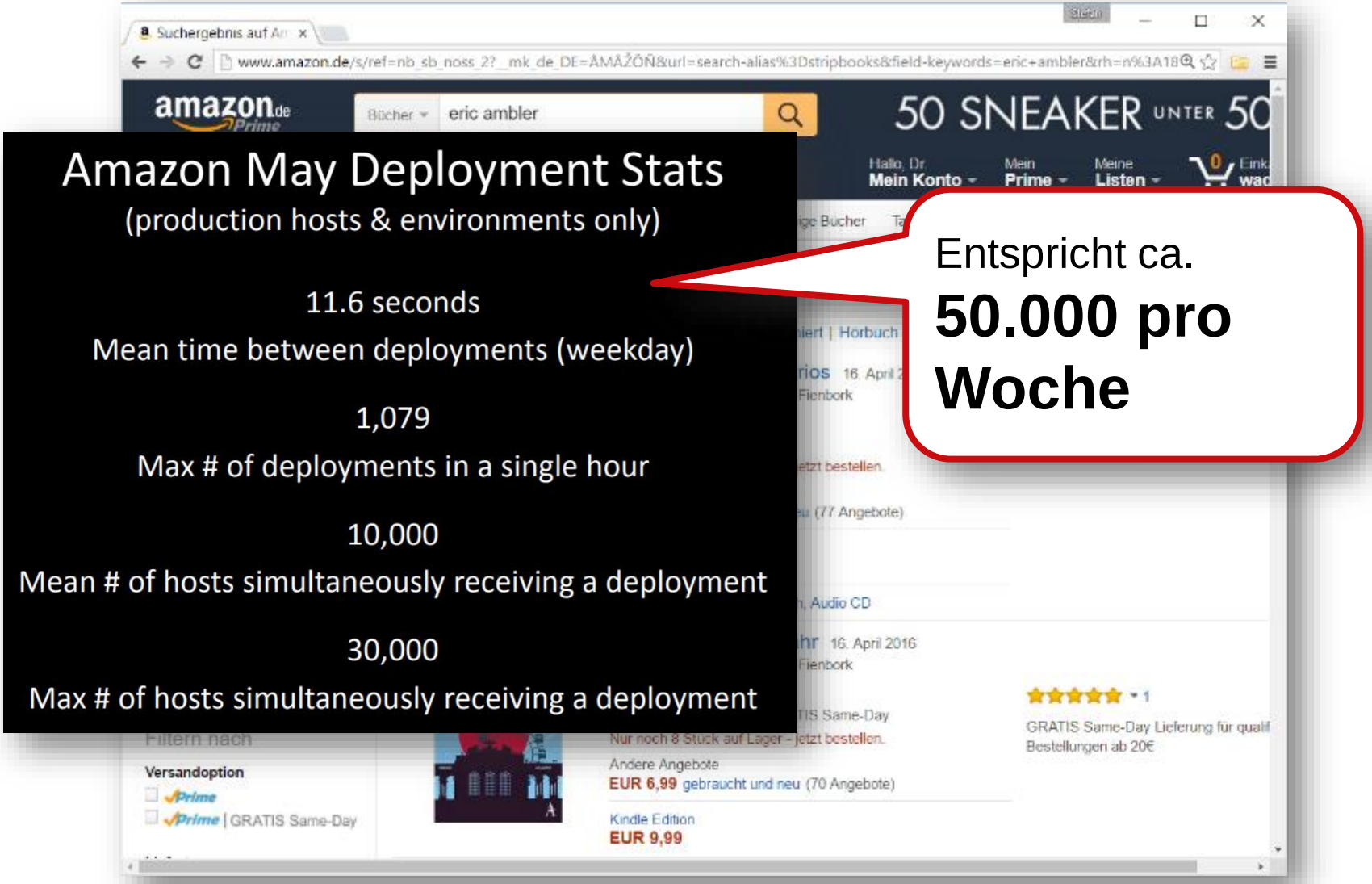
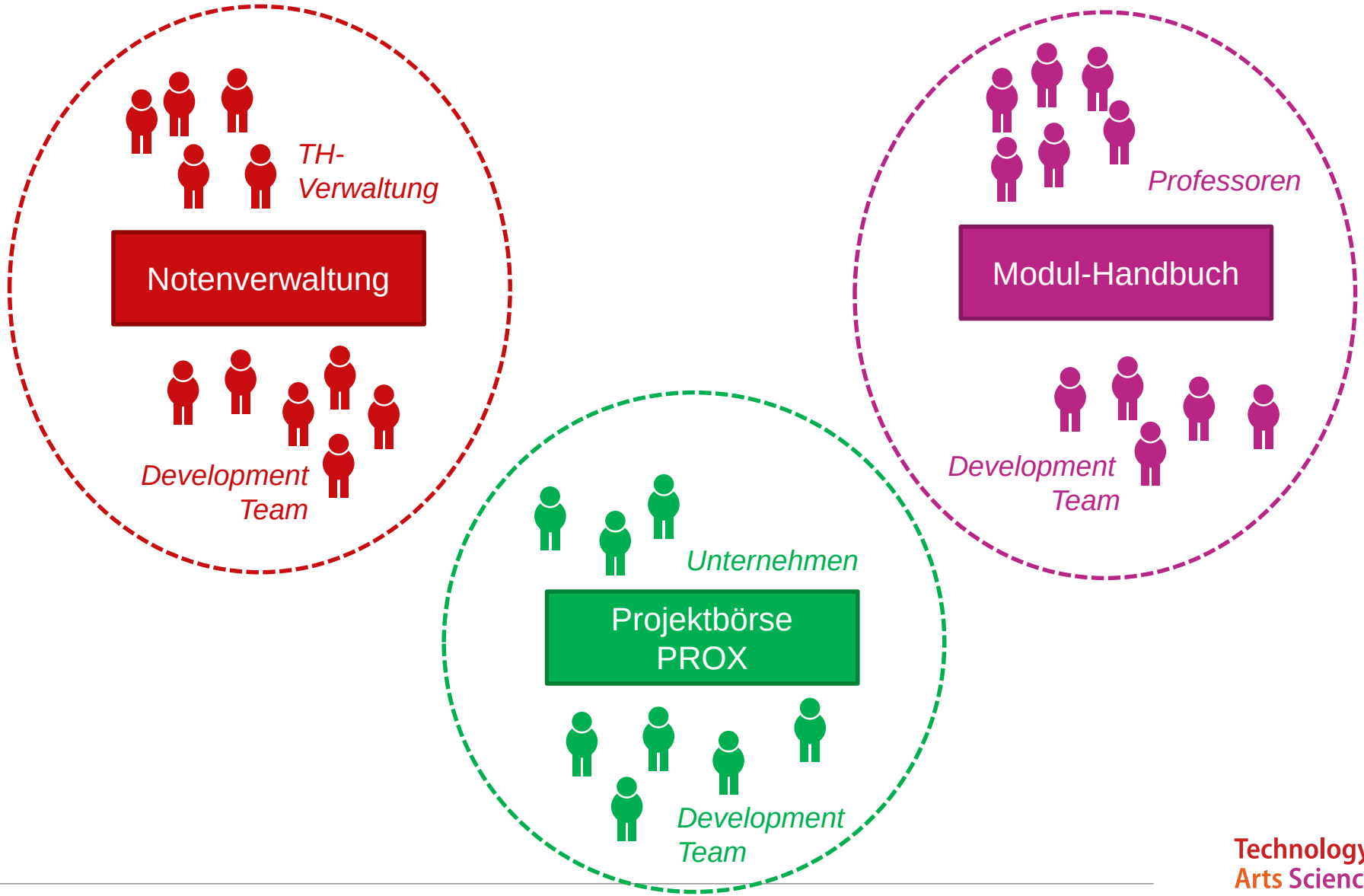


Bild: Amazon.de, Quelle: Jenkins, Jon. “Velocity Culture - The Unmet Challenge in Ops.” presented at the O’Reilly Velocity Conference 2011, June 16, 2011. http://assets.en.oreilly.com/1/event/60/Velocity%20Culture%20Presentation.pdf.

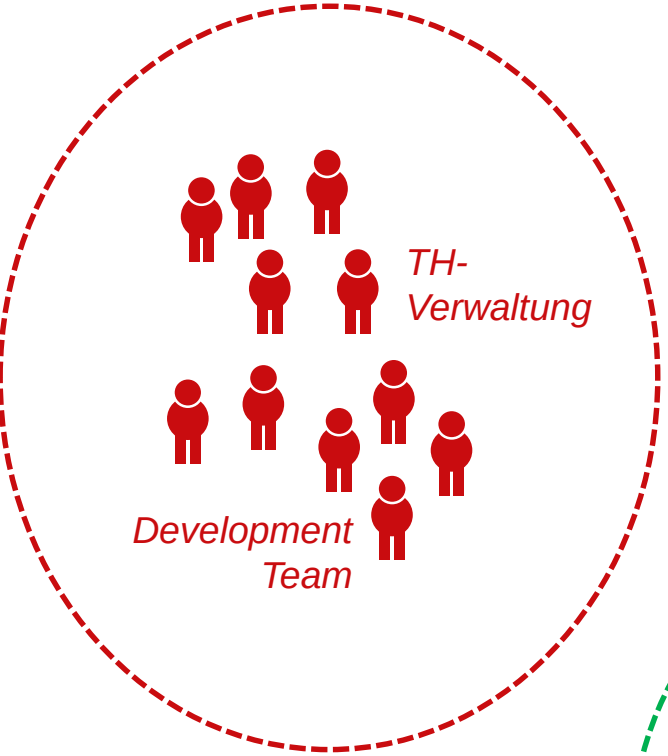
Lose gekoppelter Architekturstil:

Microservices

Ziel: Autarke Teams mit lose gekoppelter Architektur

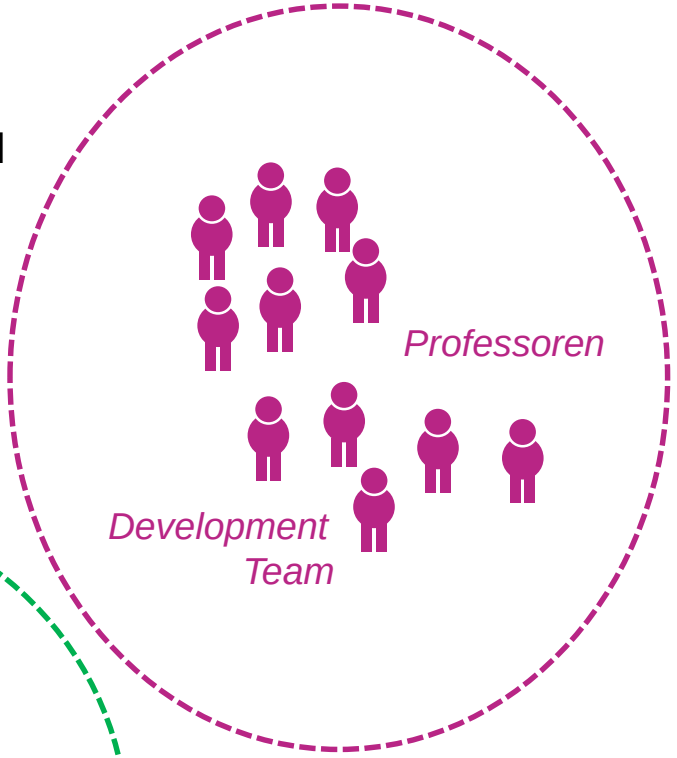


Ziel: Autarke Teams mit lose gekoppelter Architektur

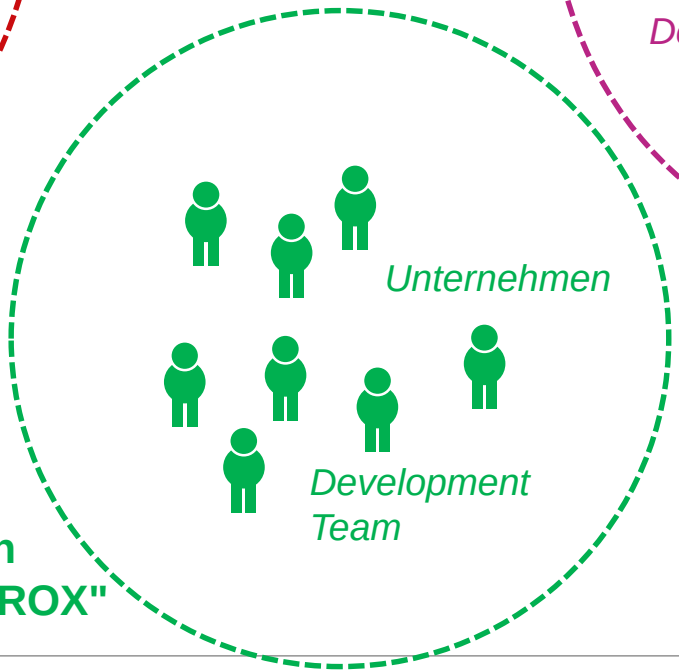


Sub-Domain "Notenverwaltung"

- Jede (Sub-)Domain sollte ihr eigenes Modell definieren dürfen
- … zunächst ohne Notwendigkeit, sich mit anderen abzustimmen

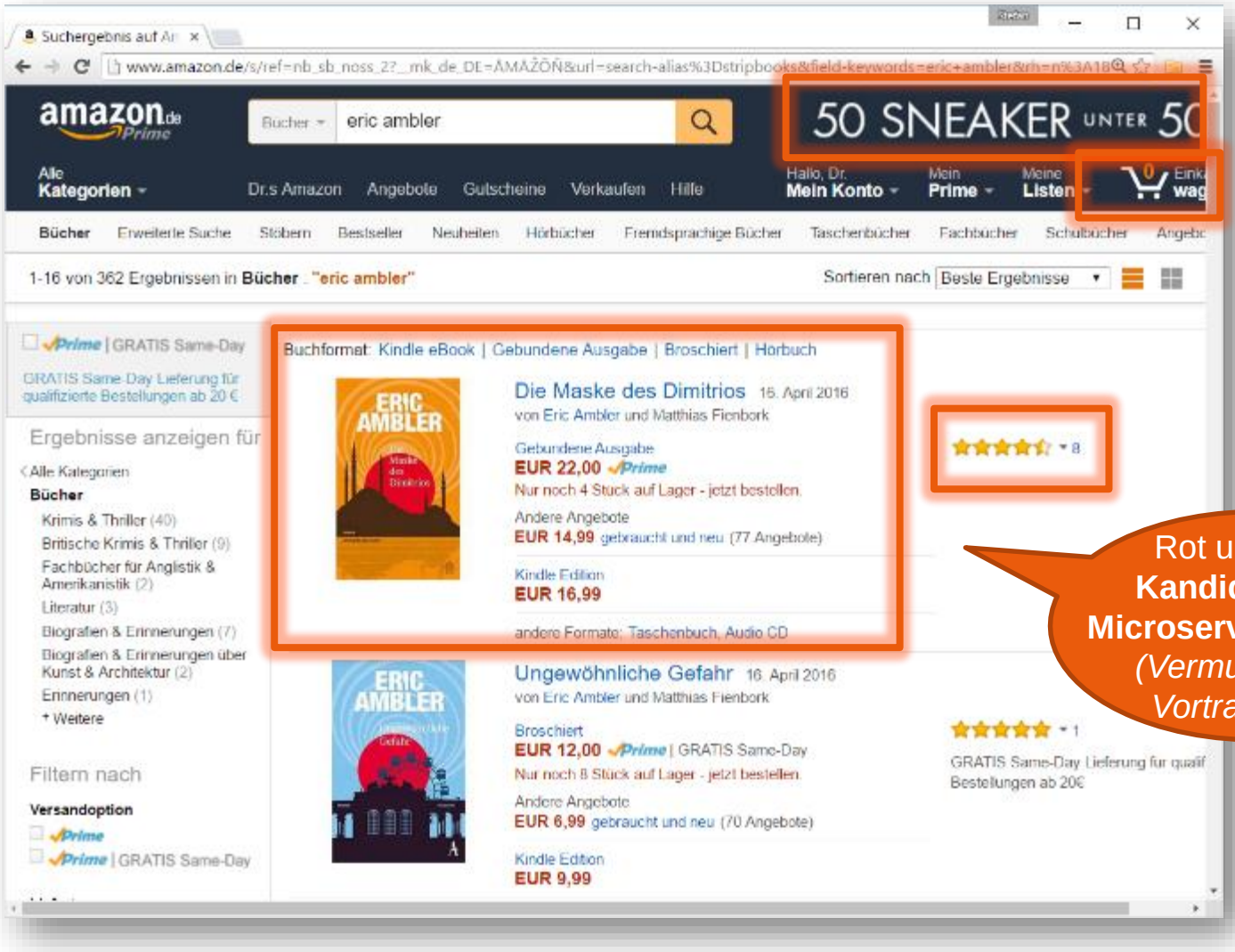


Sub-Domain "Modul-Handbuch"



Sub-Domain "Projektbörse PROX"

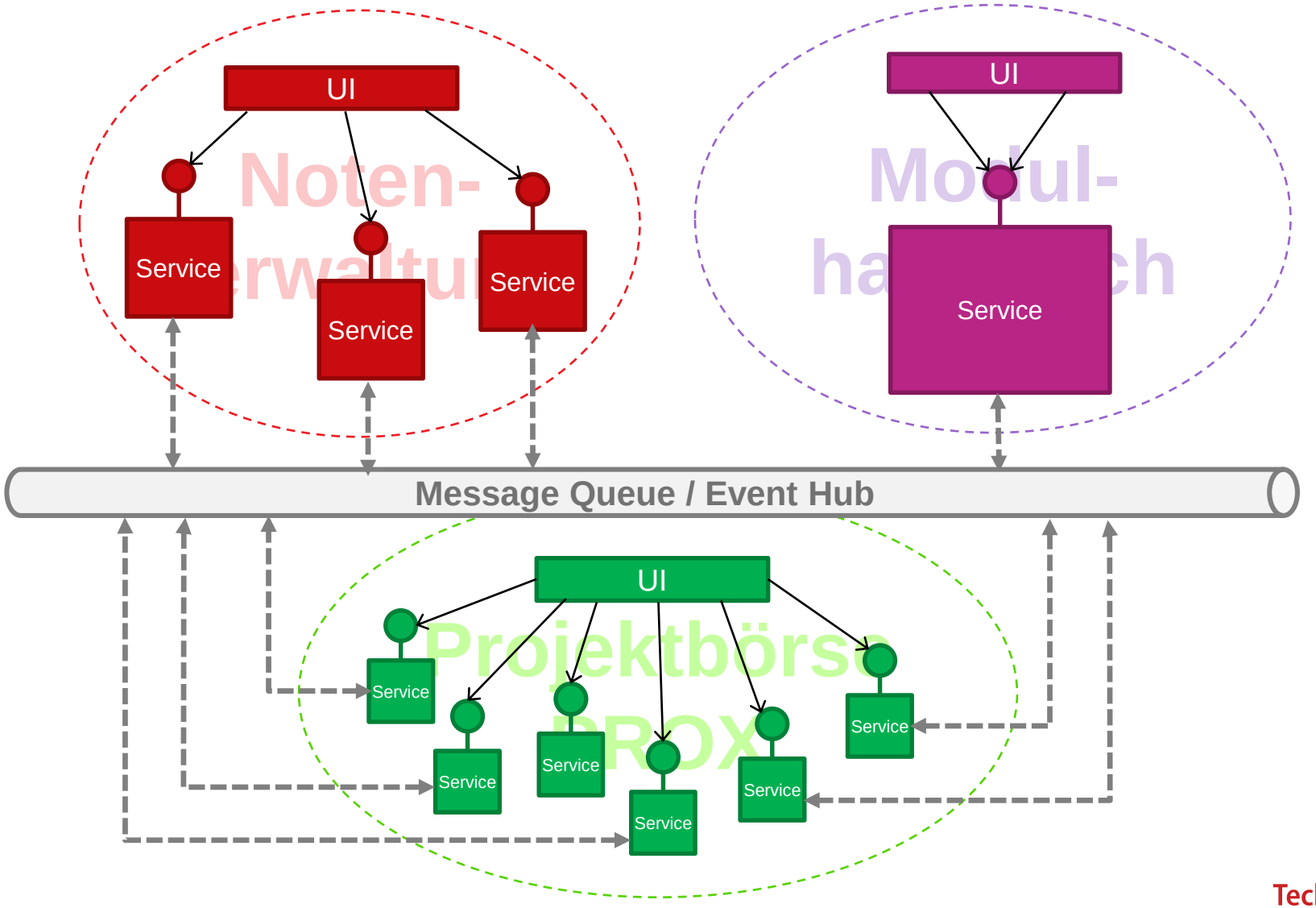
Amazon: Beispiel für dynamische Inklusion *(Vermutung!)*



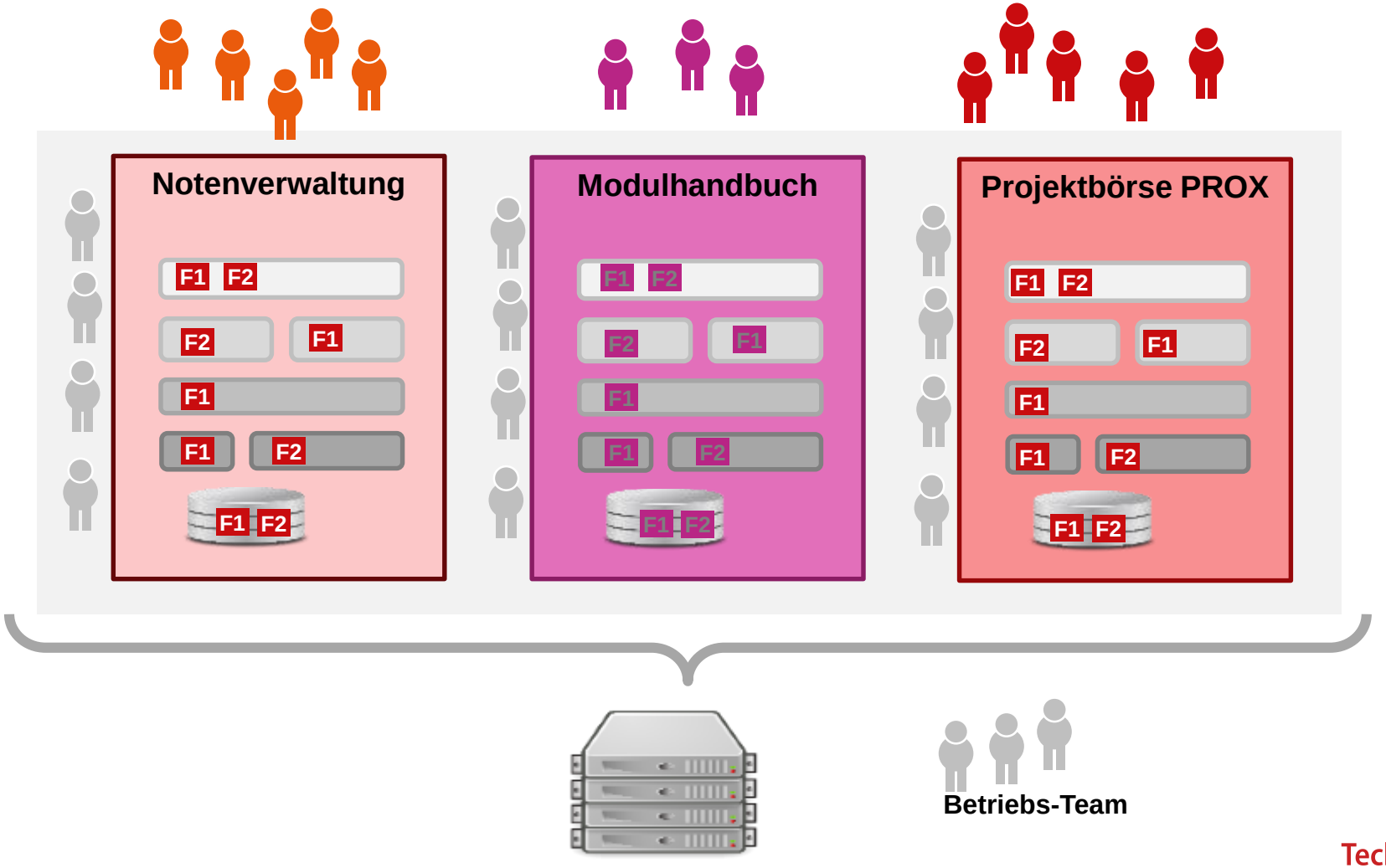
Rot umkreist:
Kandidaten für Microservice-Teil-UIs
(Vermutung des Vortragenden)

Bild: Amazon.de

Entkoppelung über asynchrone Kommunikation



Alternative: "Modulith"
(Deployment-Monolith, aber nach DDD entworfen)



Technology Arts Sciences TH Köln

Backup

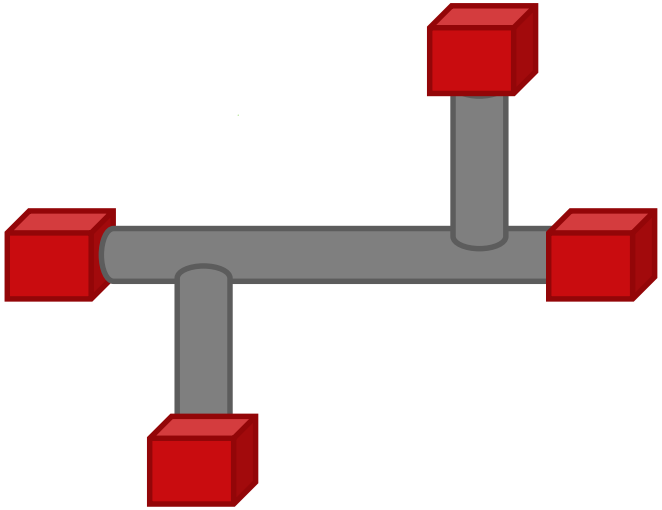
Choreographie (Microservices)

- Einzelne Einheiten agieren weitgehend autark
- Einheiten sind lose gekoppelt
- Systemverhalten entsteht im Zusammenspiel

Pipes vs. Endpoints

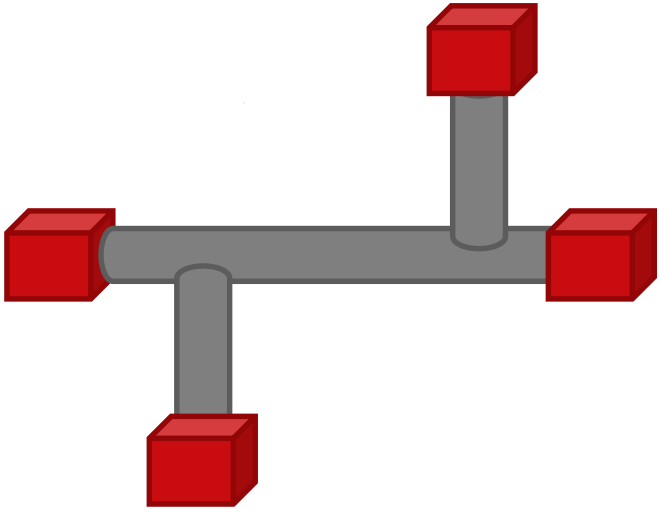
Orchestrierung / SOA

- „Smart pipes, dumb endpoints“



Choreographie / Microservices

- „Smart endpoints, dumb pipes“



Kompromisse und Prinzipien bei Microservices

Kernprinzipien und Vorzüge

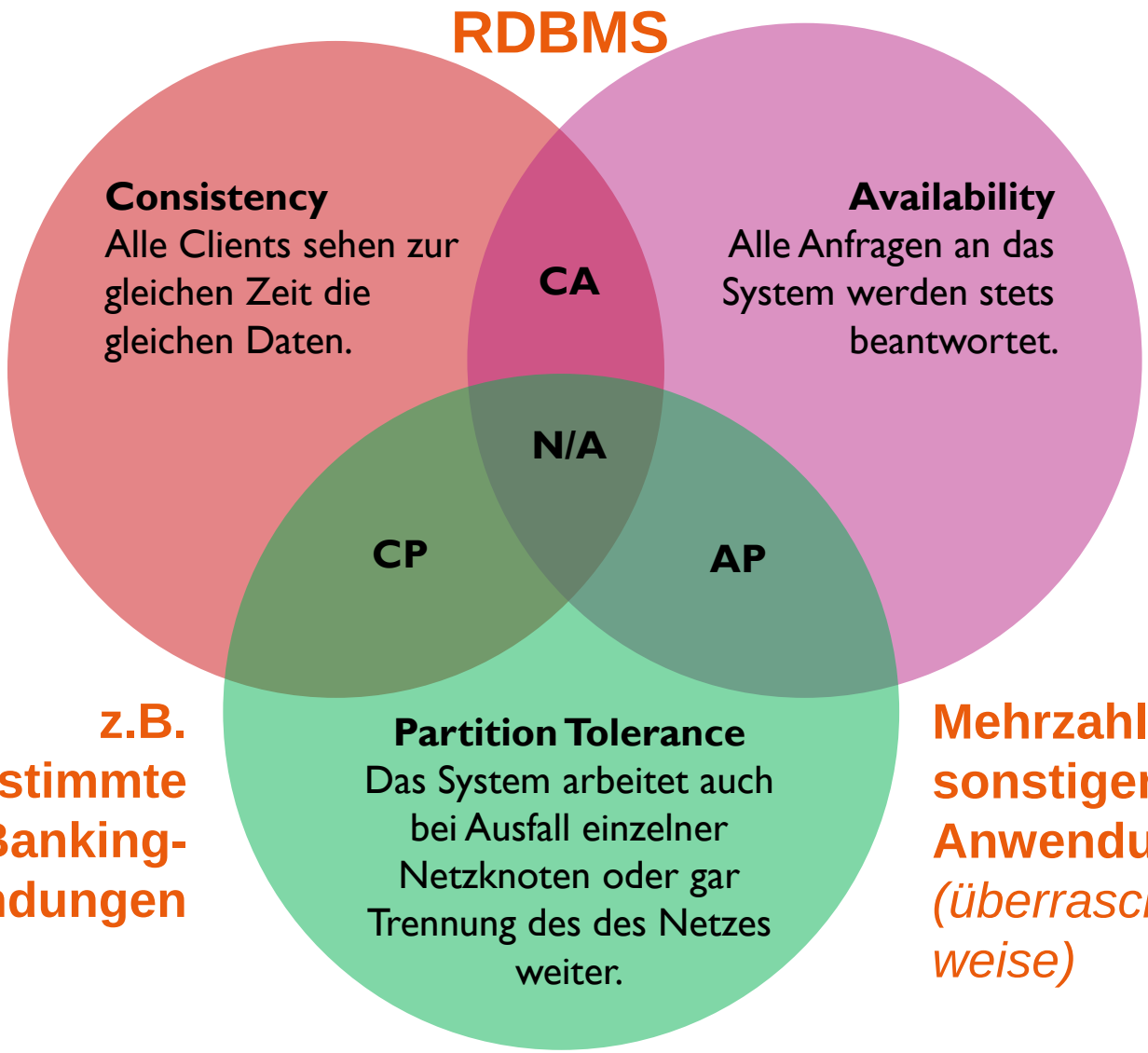
- Separation of Concerns
  - Geschnitten nach geschäftlichen, nicht technischen Gesichtspunkten
- Design for Failure
  - Resilienz
- Lose Kopplung / hohe interne Kohäsion
  - Ermöglicht evolutionäres Design
- Dezentralisierte Governance
  - E2E-Autonomie (You build it, you run it)
  - Technologiefreiheit

Kompromisse

- Redundanz bei Daten und Technologien
- Datenintegrität (Eventual Consistency)

Kompromisse bei Microservices - und Alternativen dazu

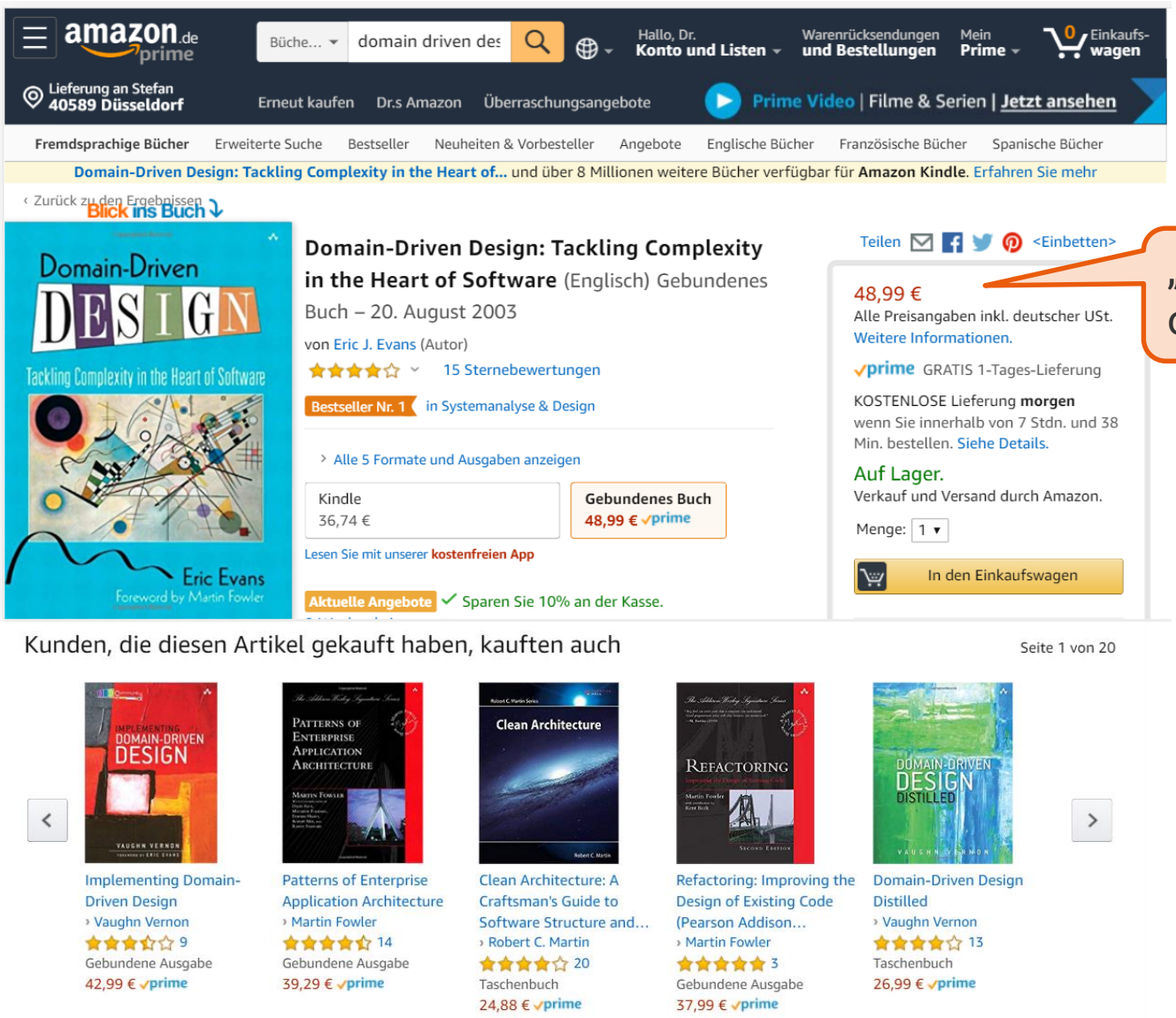
CAP – Theorem: Nur 2 von 3 Eigenschaften erfüllbar



z.B. bestimmte Banking-Anwendungen

Mehrzahl der sonstigen Anwendungen *(überraschenderweise)*

Eventual Consistency: Daten u.U. nicht sofort konsistent



Conway‘s Law: Architektur und Organisation

“Jede Organisation, die [IT-]Systeme entwirft, produziert ein Design, das die Kommunikationsstruktur der Organisation widerspiegelt.” (*)

- ***und umgekehrt!***

Organisationsstruktur
der betreuenden Personen

Architektur der IT

- Technisch segmentierte, hierarchische Organisation
- Klare Trennung Entwicklung/Betrieb

- Technisch geschnittener (Deployment-)Monolith

- Agiler Prozess
- Schneller Reaktion auf Änderungen
- Flexible, autarke Teams
- DevOps (Entwicklung/Betrieb Hand in Hand)

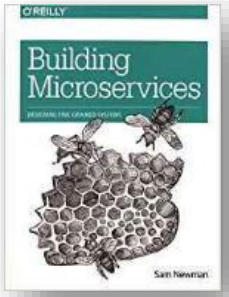
- Radikal lose gekoppelte IT-Komponenten
- Microservice-Architektur

(*) Quelle: Conway, Melvin. “How Do Committees Invent?” *Datamation*, 1968. http://www.melconway.com/Home/Committees_Paper.html. Übersetzung durch den Vortragenden.

Eigenschaften von Microservice-Landschaften

- Unabängiges Deployment
- Unabhängige Teams
- Asynchrone Kommunikation

Literaturempfehlungen für DDD + Microservices



Sam Newman (2015)

Building Microservices, O'Reilly

- Standardwerk aus dem englischen Sprachraum
- Gibt es auch auf Deutsch

Eberhardt Wolff (2018)

Microservices: Grundlagen flexibler Softwarearchitekturen (2. Auflage), dpunkt

- Deutsches „Standardwerk" zu Microservices
- Starker Fokus auf Hosting-Konzepte

Bente, Deterling, Reitano, Schmidt (2020)

Sieben Weggabelungen - Wegweiser im DDD-Dschungel.
JavaSPEKTRUM, *2020*(02), 28–31.
(Sciebo-Link, Passwort = Kürzel des Kurses (Kleinbuchstaben))

- Guter Einstiegs- & Übersichts-Artikel ☺