

# Frameworks, Daten und Dienste im Web

**Web Development**  
Vertiefungsmodul im Medieninformatik Bachelor

19.05.2022, Dirk Breuer

# HELO

- Dirk Breuer
- Medieninformatik Alumni (Master)
- Über 12 Jahre Erfahrung als Software Entwickler in unterschiedlichen Projekten
- Disclaimer: Verdiene mein Geld mit synchronen Anwendungen 😅

# Voraussetzungen

- Die Inhalte von AP1, AP2 und GDW
- Die Kompetenz zur Entwicklung von webbasierten Anwendungen in JavaScript
- Die Bereitschaft und Fähigkeit zum selbstständigen Wissenserwerb

# Was sind Ihre Erwartungen?



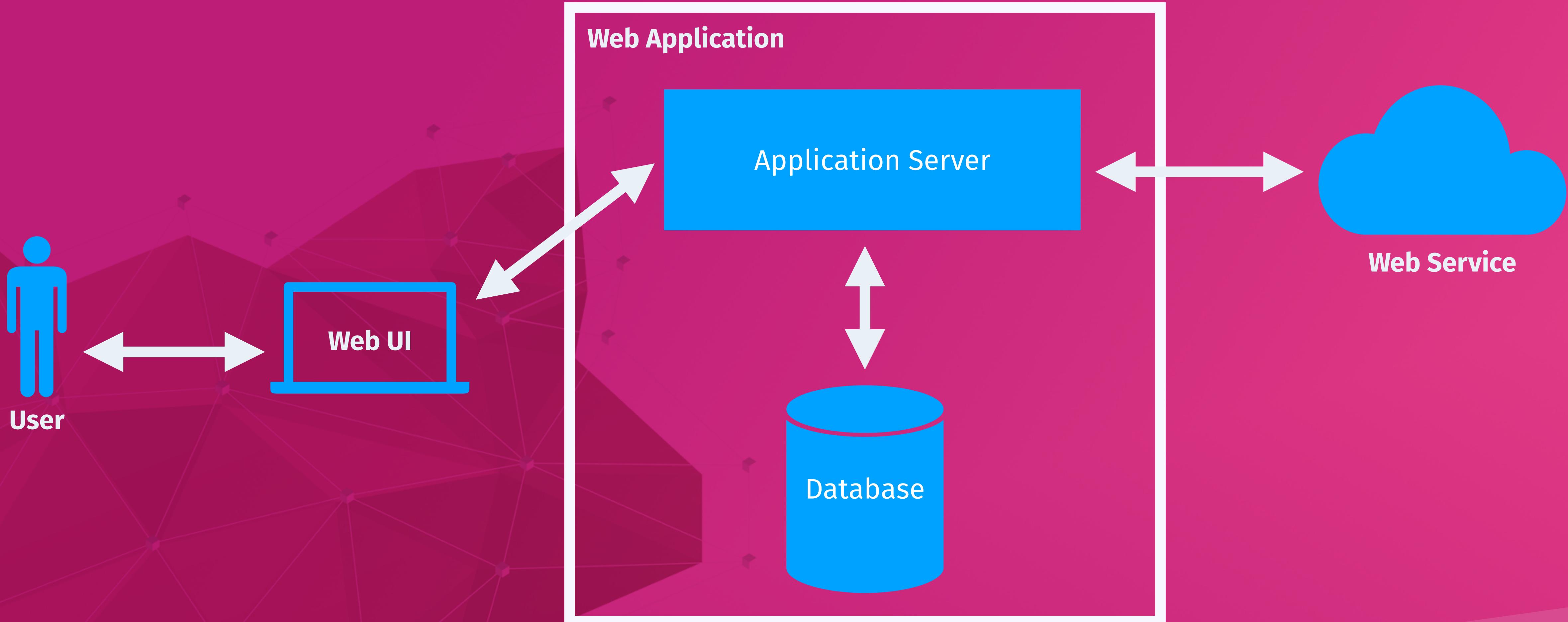
## miro Board

<https://shorturl.at/eiG67>

# Ziele

- Den Grundgedanken der asynchronen, serverseitigen Kommunikation und die zugehörigen Standards, Frameworks und Protokolle erklären und im Fachdiskurs vertreten können
- Projektbezogen den Einsatz von Frameworks für die asynchrone, serverseitige Kommunikation recherchieren, bewerten und programmieren können
- Den Grundgedanken und die Standards, Frameworks und Protokolle für offene Daten erklären und im Fachdiskurs vertreten können
- Projektbezogen die Nutzung von offenen Daten und Services im Web recherchieren, bewerten und den Zugang programmieren können
- Zugang zum gezielten, selbstständigen Wissenserwerb durch Literatur, Webinars und Recherchen im Web gewonnen haben
- Für die Implementierung von serverseitige Anwendungen Frameworks auswählen, installieren, programmieren können; dabei Schwierigkeiten überwinden können
- Deployment von serverseitigen Anwendungen durchgeführt haben und Grundlegende Konzepte des Betriebs benennen können

# Motivation



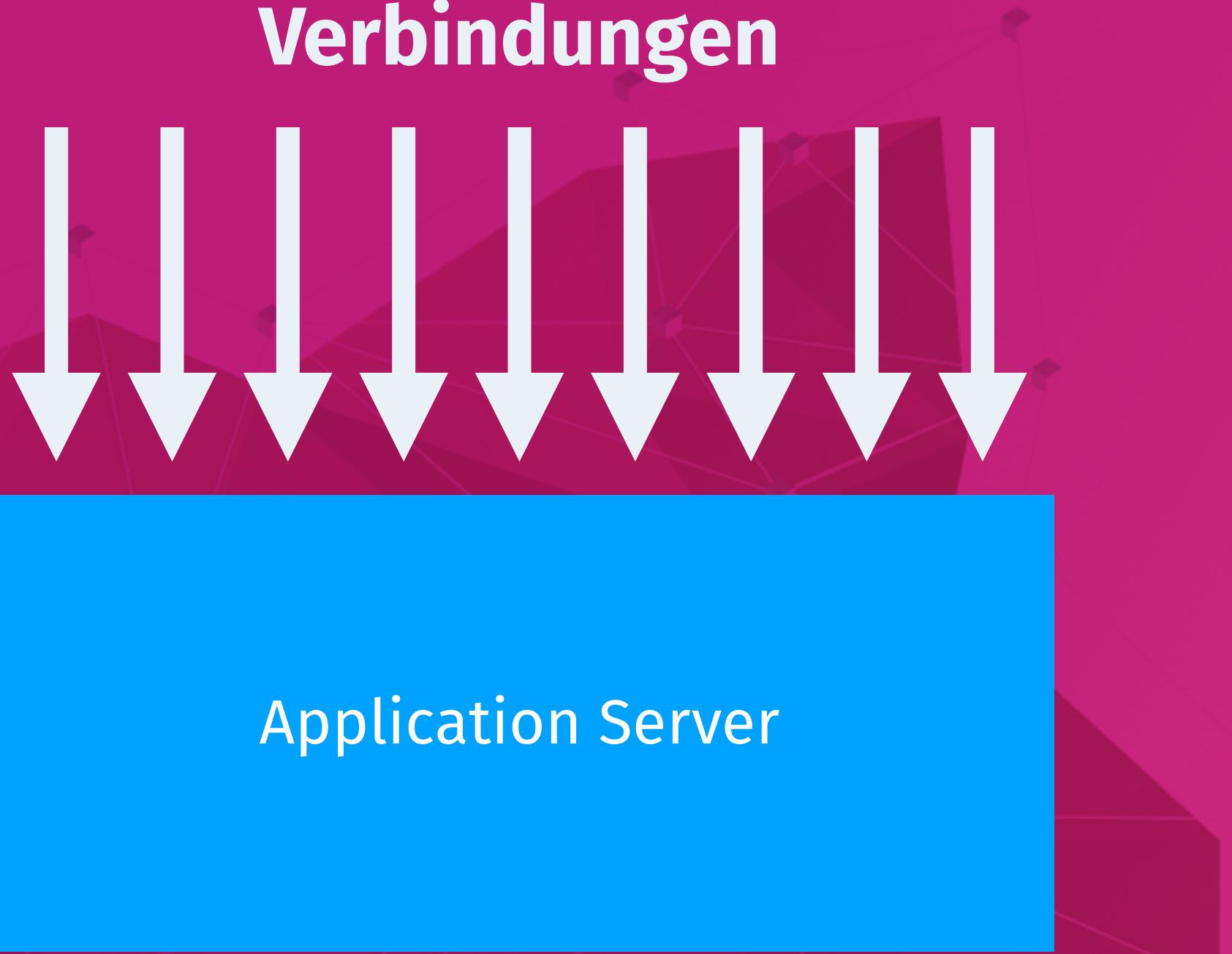
# Synchrone Kommunikation



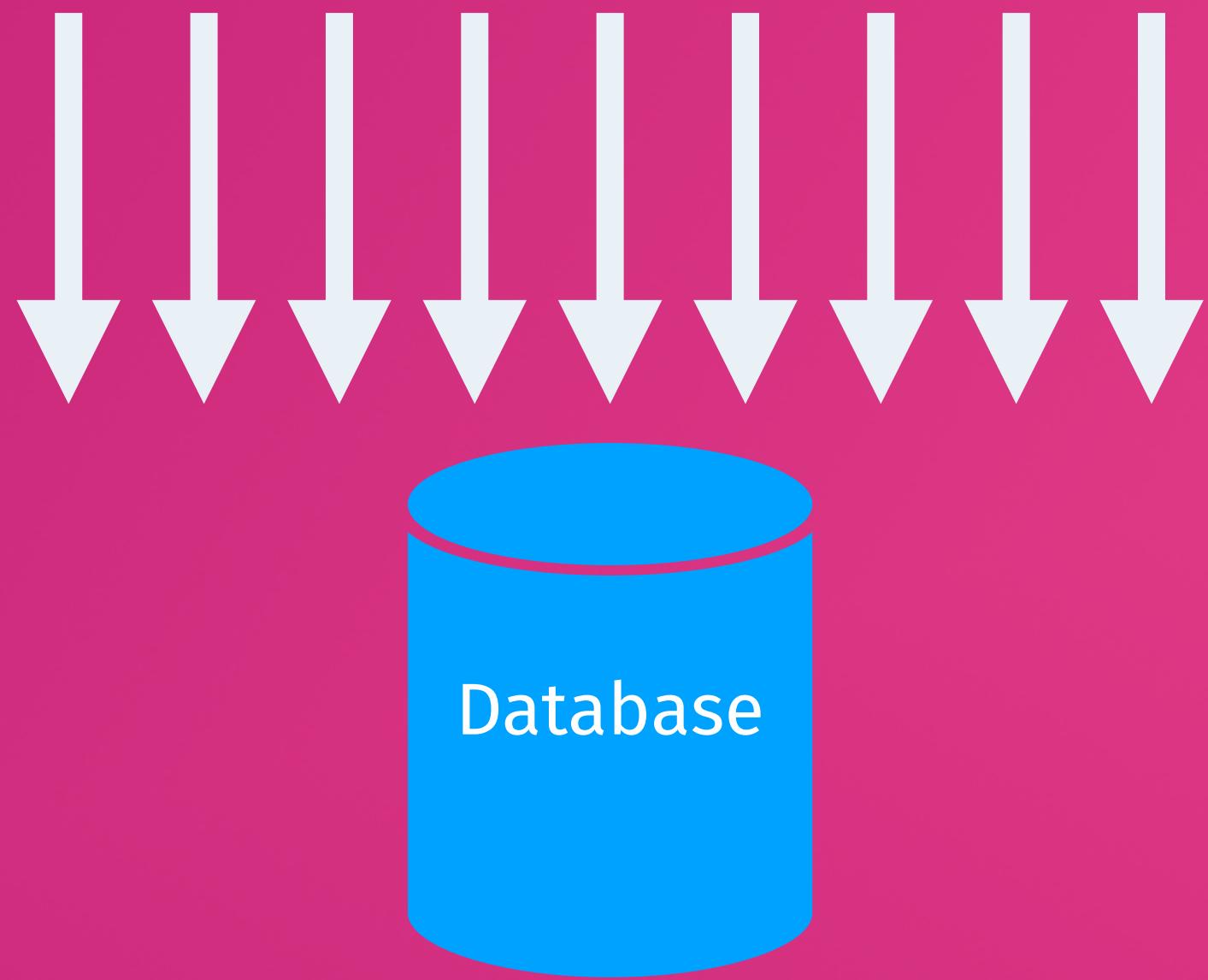
“Unter **synchroner Kommunikation** versteht man in der Informatik und Netzwerktechnik einen Modus der Kommunikation, bei dem die Kommunikationspartner (Prozesse) beim Senden oder beim Empfangen von Daten immer synchronisieren, also **warten (blockieren)**, bis die **Kommunikation abgeschlossen** ist.”

–Wikipedia, [https://de.wikipedia.org/wiki/Synchrone\\_Kommunikation](https://de.wikipedia.org/wiki/Synchrone_Kommunikation), abgerufen 27.05.2021

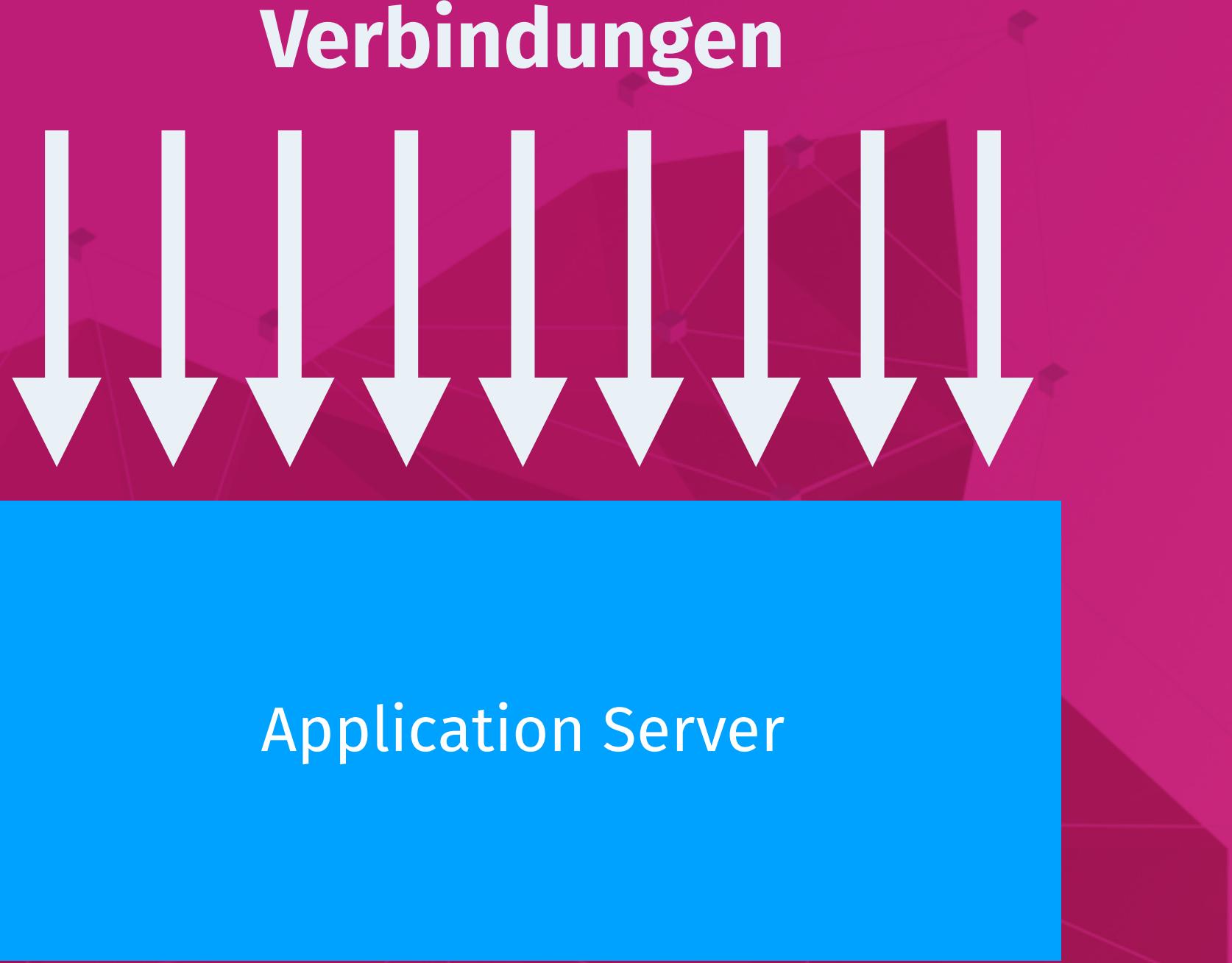
**Verbindungen**



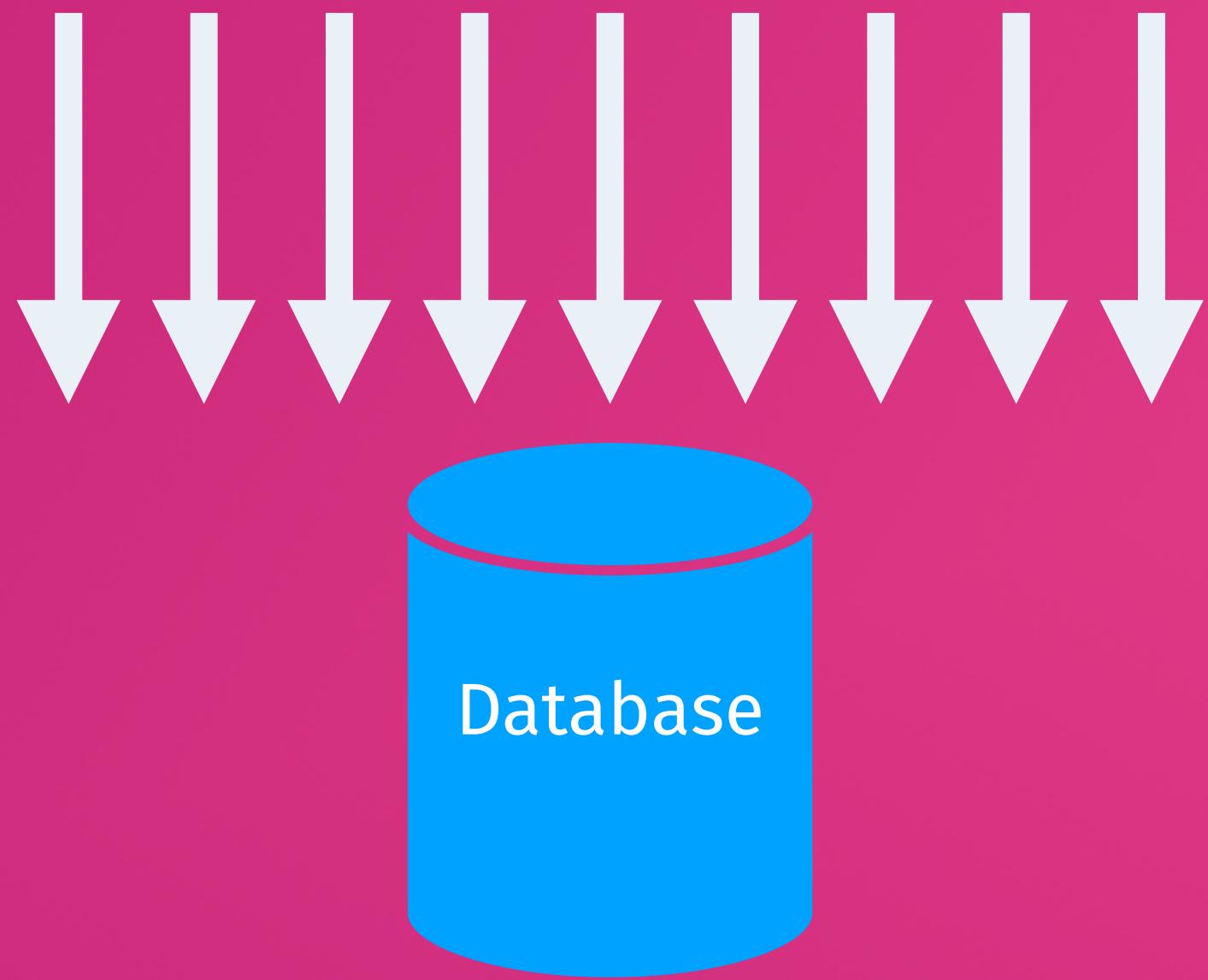
**Verbindungen**



**Verbindungen**



**Verbindungen**



Verbindungen



Verbindungen



Verbindungen (Connections)  
sind endlich!

# Endliche Ressourcen

- Verbindungen
- Rechenleistung
- Speicherkapazität
- Netzwerkbandbreite



*“So viele Nutzer:innen habe ich gar nicht!”*

*-Max Mustermann*

# Software Lebensdauer

- < 100.000 LOC: 6-8 Jahre
- < 1.000.000 LOC: 8-12 Jahre
- > 1.000.000 LOC: 12-14+ Jahre

```
const apiUrl = 'https://my-awesome-web-shop.com/api/cart';  
  
document.querySelector("a").addEventListener("click", loadContent);  
  
let loadContent = function (event) {  
    fetch(apiUrl)  
        .then(response => response.text())  
        .then(text => {  
            this.outerHTML = text;  
  
            return null;  
        })  
        .catch((err) => {  
            console.error(`Failed to embed link ${apiUrl}`, err);  
        });  
}
```



# E-Mail Versand

# Asynchrone Prozesse in synchronen Systemen

- JavaScript in Web-UI
- E-Mail Versand
- Langlaufende Prozess
  - Import von Daten
  - Export von Daten
  - ...

# Asynchrone Kommunikation



“Unter **asynchroner Kommunikation** versteht man in der Informatik und Netzwerktechnik einen Modus der Kommunikation, bei dem das Senden und Empfangen von Daten **zeitlich versetzt** und **ohne Blockieren** des Prozesses durch bspw. Warten auf die Antwort des Empfängers ”

–Wikipedia, [https://de.wikipedia.org/wiki/Asynchrone\\_Kommunikation](https://de.wikipedia.org/wiki/Asynchrone_Kommunikation), abgerufen 27.05.2021



Here be Dragons

Los dos hermanos

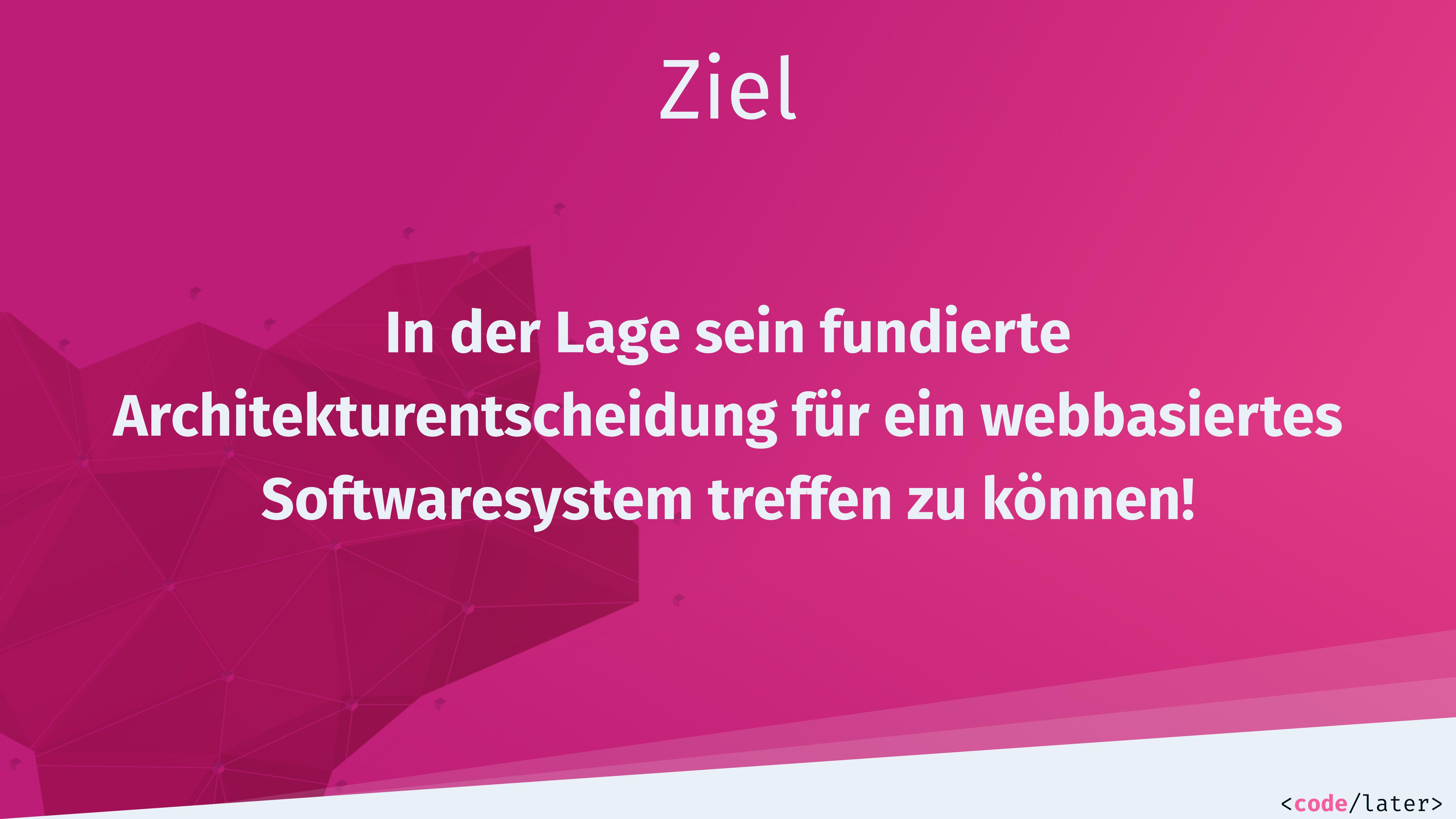
Los Bolcanes.

Tropicus Cancri

Lalabriga

- Fokus auf Anwendungslogik "hinter" Endgeräten (bspw. Browser, App)
- Alternativen zu "klassischer" Client-Server-Architektur
- Nutzung von "Reactive Systems"
- Nutzung "Offener Daten", um Unabhängigkeit von "Daten-Monopolisten" zu erreichen

# Ziel



In der Lage sein fundierte  
Architekturentscheidung für ein webbasiertes  
Softwaresystem treffen zu können!

# Softwarearchitektur

*“[...] eine strukturierte oder hierarchische Anordnung der Systemkomponenten sowie Beschreibung ihrer Beziehungen [...]”*

*-Helmut Balzert: Lehrbuch der Softwaretechnik. Bd. 2: Entwurf, Implementierung, Installation und Betrieb,  
Spektrum Akademischer Verlag, 2011*

# Softwarearchitektur



*“Architektur ist die Menge der wichtigen Entscheidungen [...], deren Änderungen [sehr] teuer sind.”*

-Stefan Tilkov, <https://geekstammtisch.de/8-gst031-corba-in-cool>



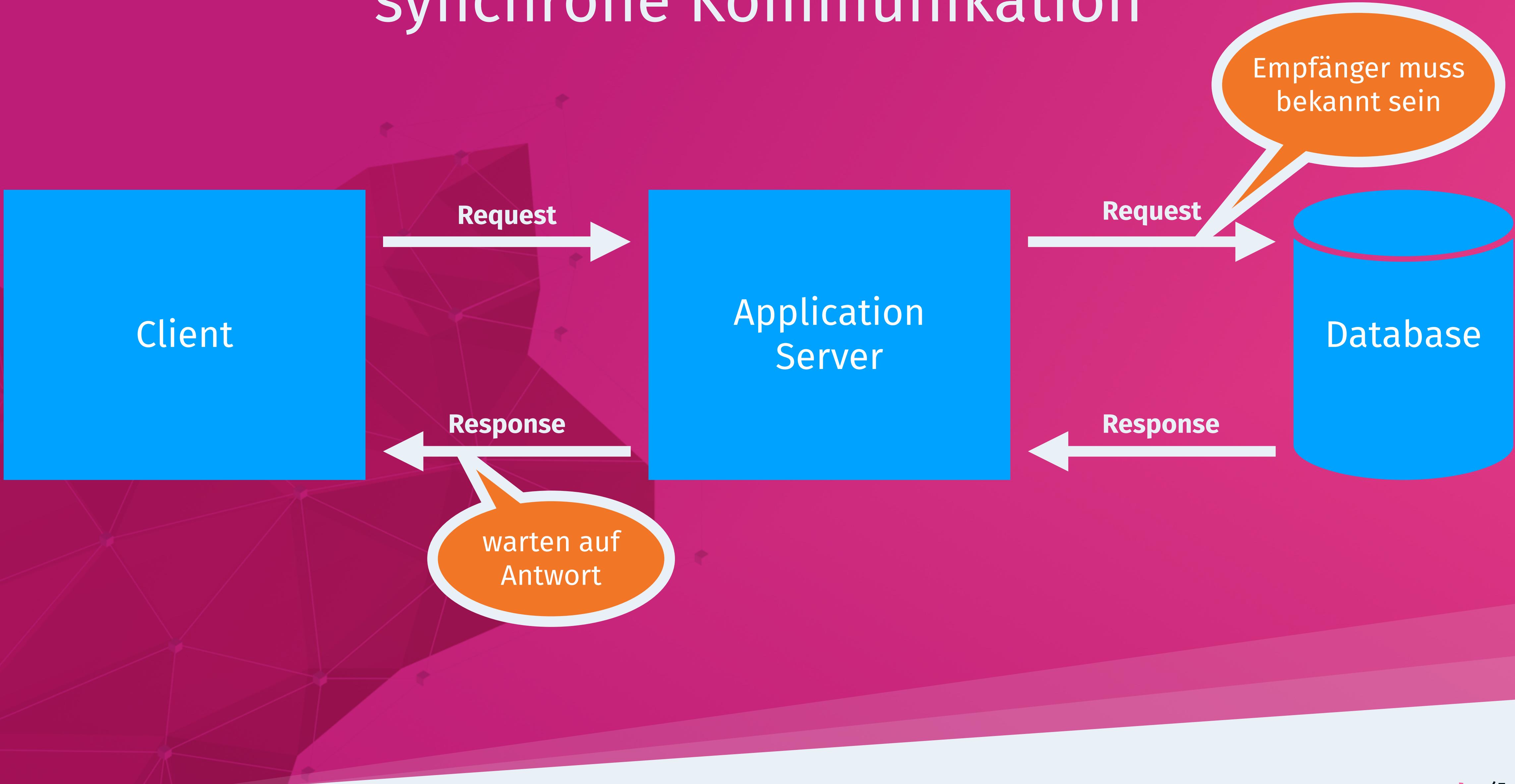
# Werkzeugkasten

Oder: Wenn das einzige Werkzeug ein Hammer ist, dann ist jedes Problem ein Nagel.

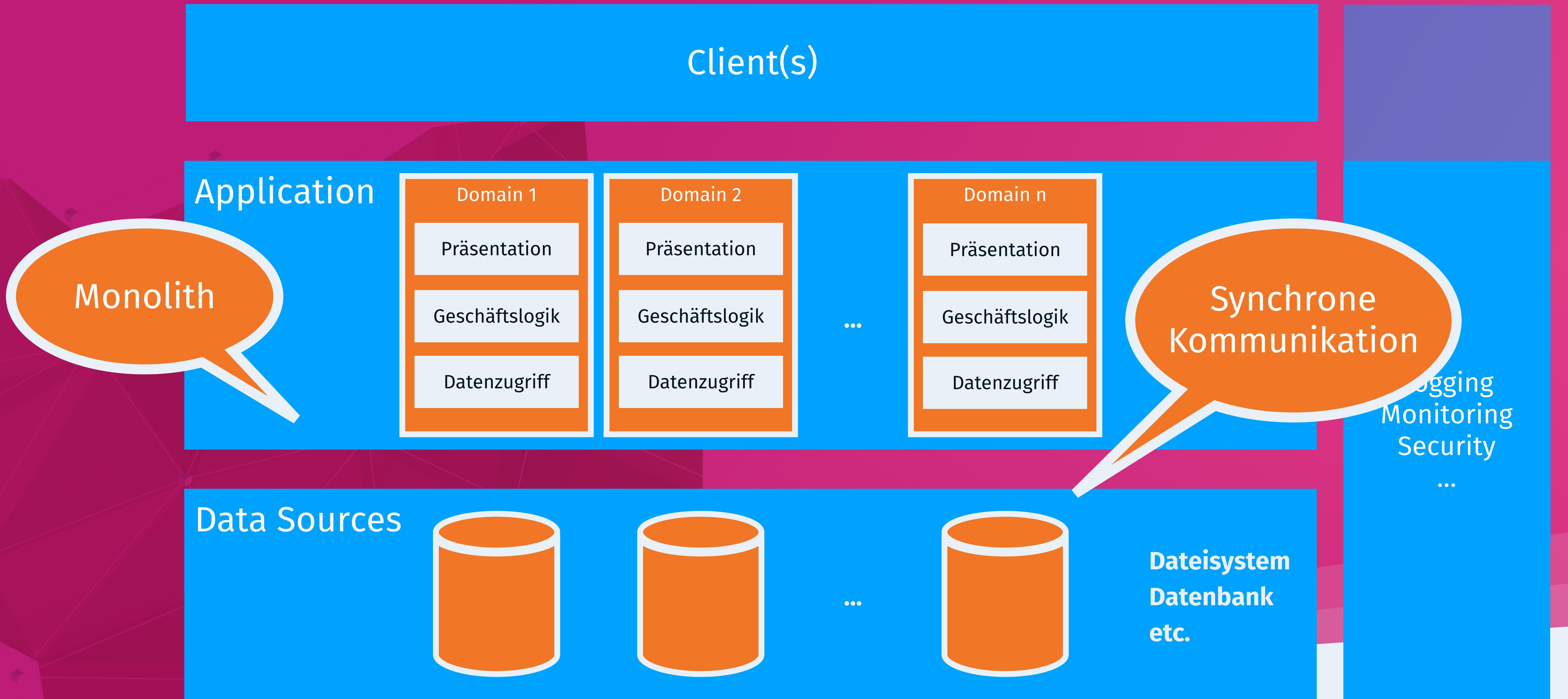


# Kurze Einführung

# Enge Kopplung durch synchrone Kommunikation



# Multi-Tier Architekturen



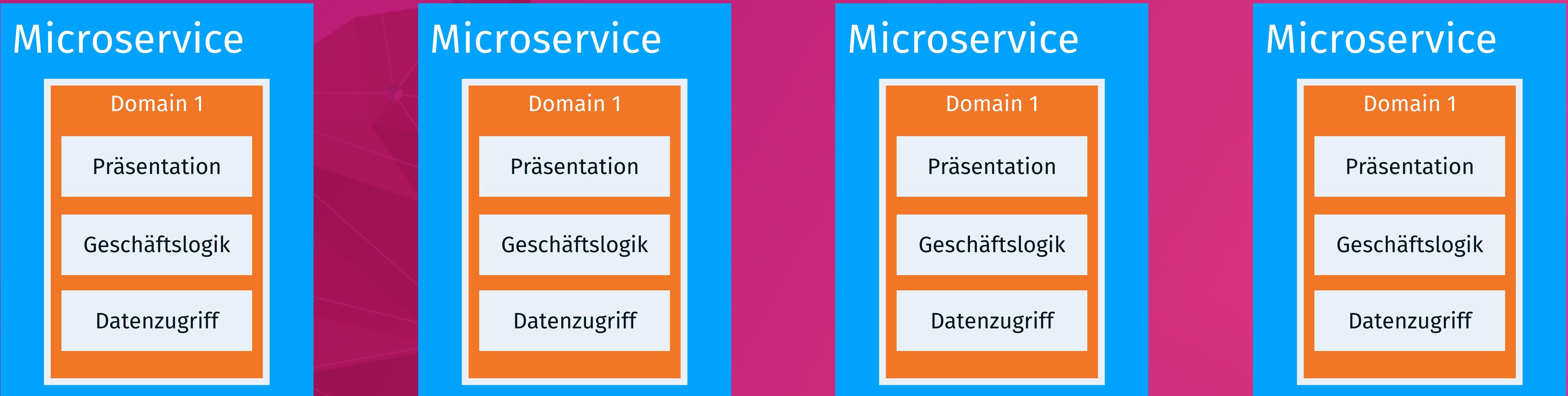
# Monolithen

- Enthält ganz alte und ganz neue Teile
- (Oft) zu groß, um vollständig erfasst zu werden
- Neue Features (oft) schwer umsetzbar
- All-or-nothing: Muss immer vollständig
  - getestet und
  - ausgerollt werden
- (Oft) Geringe Fehlertoleranz
- Modernisierung riskant und kostspielig
- Skalierung bei wachsenden Anforderungen sehr aufwendig

# Microservice Architektur



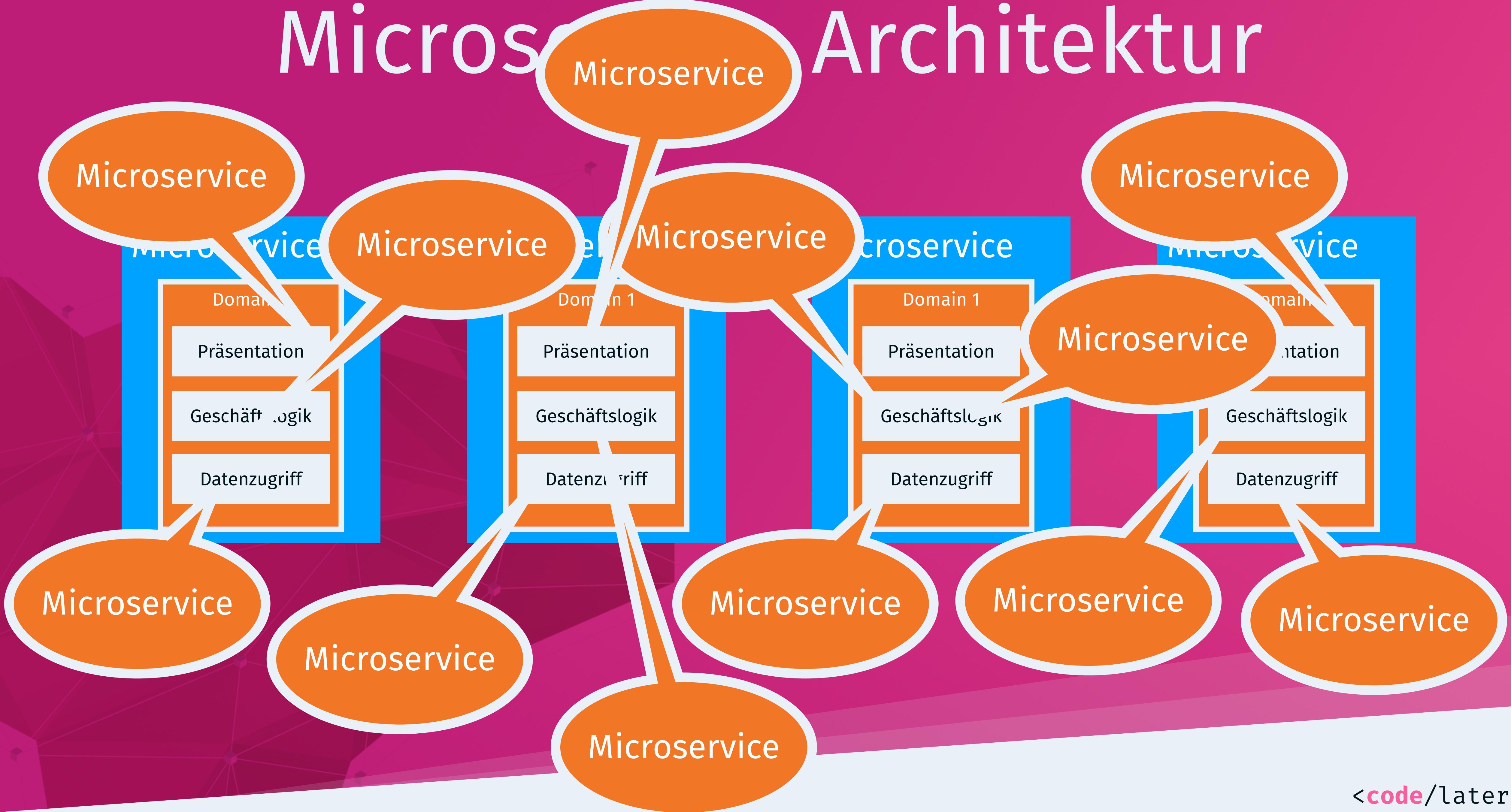
# Microservice Architektur



# Microservice Architektur

- Klein genug, um vollständig überblickt zu werden
- Technologisch vollständig unabhängig (theoretisch)
- Von einander Isoliert
- Können (müssen) unabhängig getestet und deployed werden können
- Leicht zu skalieren
- Ausfall einzelner Microservices lokal und mit wenig Auswirkungen auf das Gesamtsystem

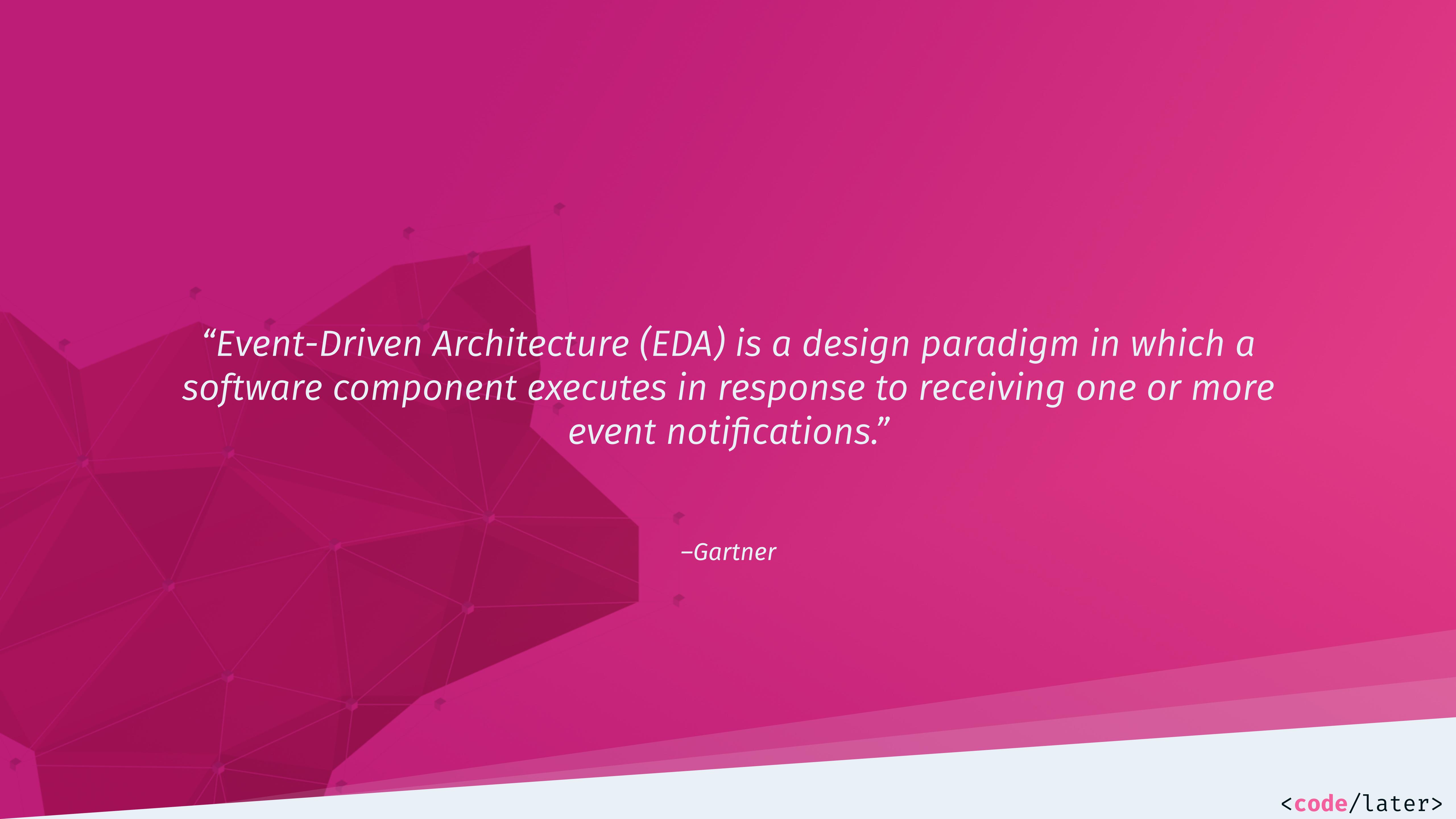
# Microservice Architektur



# Microservice Architektur

- Synchrone Kommunikation nicht vertretbar
- Ständige Konsistenz von Daten nicht möglich
- Keine Transaktionen möglich (nur innerhalb eines Microservice)
- Netzwerk ist unzuverlässig
- Umgang mit unterschiedlichen Versionen
- Umgang mit Ausfall / Überlast einzelner Microservices
- Korrekter "Schnitt" einzelner Verantwortlichkeiten nicht trivial

# Formalisierung des Datenaustauschs



*“Event-Driven Architecture (EDA) is a design paradigm in which a software component executes in response to receiving one or more event notifications.”*

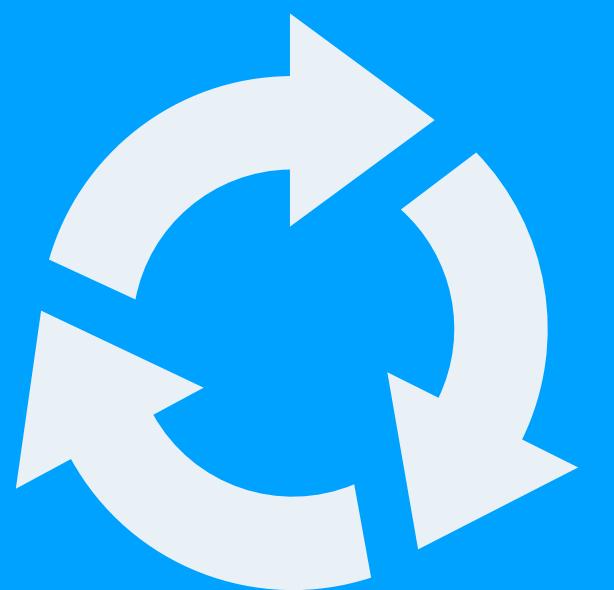
*-Gartner*

**Konsumiert  
Events**



Asynchron

**Microservice**



**Veröffentlicht  
Events**



# Events == Fakten

- Events repräsentieren unverändertere Fakten
- Sie akkumulieren, d.h. "überschreiben" sich nicht
- Können nicht "zurückgezogen", aber vom empfangenden System (Microservice) ignoriert werden
- Können nicht gelöscht werden
- Neue Events (Fakten) können vorherige falsifizieren

# Event

Zeitstempel

Daten

- Events finden in zeitlicher Abfolge ab
- Zeit ist in verteilten Systemen nicht absolut
- Konflikte sind keine Ausnahme, sondern sind zu erwarten

# Events

- Können in einem Log persistiert werden
- Die Datenbank ist ein Cache des Event-Logs
- Das Log kann jederzeit wieder abgespielt werden
- Beispiele: Apache Kafka

# Fehler- und Überlastsituationen

- In verteilten System können (werden) einzelne Komponenten unabhängig von einander ausfallen bzw. überlastet sein
- Schnelle Komponenten dürfen langsame nicht überlasten
- Langsame Komponenten dürfen das System nicht beeinträchtigen
- Daten können inkonsistent sein
- Daten können in unterschiedlichen Versionen vorliegen
- **Die Anwendungslogik muss mit diesen Situationen umgehen können!**

# CAP-Theorem

- Ein verteiltes System kann immer nur zwei der folgenden Eigenschaften gleichzeitig erfüllen:
  - **Konsistenz (C consistency)**: Die Konsistenz der gespeicherten Daten.
  - **Verfügbarkeit (A availability)**: Die Verfügbarkeit im Sinne akzeptabler Antwortzeiten.
  - **Partitionstoleranz (P partition tolerance)**: Das System arbeitet auch weiter, wenn es partitioniert wird, d. h., wenn Knoten nicht mehr miteinander kommunizieren können.

# Reactive Manifesto

- **Responsive:** The system responds in a timely manner if at all possible.
- **Resilient:** The system stays responsive in the face of failure.
- **Elastic:** The system stays responsive under varying workload.
- **Message Driven:** Reactive Systems rely on asynchronous message-passing to establish a boundary between components that ensures loose coupling, isolation and location transparency.

# Übergänge Async/Sync

- Wenn eine Systemarchitektur auf asynchroner, event-basierter Kommunikation beruht müssen in der Regel an den Schnittstellen zur Umwelt Übergänge zu synchronen Interaktionen realisiert werden
  - Umwandlung synchroner Datenabrufen (etwa von REST-basierten Diensten) in Events (bspw. durch Polling der Daten)
  - Zusammenführung aller asynchronen Events in einen synchronen Zustand zur Anzeige auf bspw. einer Web-UI

# Prüfungsleistung

# Ermittlung Note

Insgesamt können 100 Punkte erreicht werden, davon 50 im Projektteil und 50 im Fachgespräch. Zum Bestehen des Moduls müssen mindestens 48 Punkte erzielt werden.

# Mündliche Prüfung

- Termin: 23.09.2022
- Fachdiskurs mit den Projektgruppen
- Vor Ort in Gummersbach
- Zu erreichende Punkte: 50

# Inhalte

- Erarbeitetes Wissen aus der Literatur
- Leitfragen geben Rahmen der Recherche vor
- Vor allem im Selbststudium zu leisten
- Bezug auf Projektarbeit möglich

# Projektarbeit

- Abgabe (via GitHub): 08.09.2022
- Präsentation: 15.09.2022
- Präsentation im Plenum in Gummersbach
- 5-10 Minuten pro Projektgruppe
- Ziel der Präsentation: Ergebnisse und Herausforderungen einem Fachpublikum vorstellen
- Zu erreichende Punkte: 50

# Zielsetzung

- Anwendung die in Microservices / Komponenten aufgeteilt ist
- Asynchrone Kommunikation zwischen Microservices (bspw. durch Events)
- Anbindung mindestens eines externen (offenen) Datendienstes
- Es muss State geben! (Ohne State ist alles möglich)
- Einfache! Web-UI zur Interaktion mit der Anwendung
- Deployment der Anwendung
- Betriebskonzept
- Dokumentation und Nachvollziehbarkeit in GitHub

# Zu beachten

- Fokus auf inhaltliches Arbeiten setzen
- Komplizierte Infrastruktur und/oder Dienste vermeiden
- Aufwendiges UI nicht notwendig
- Umfang im Vorfeld klar definieren
- Aspekte im Vorfeld explizit ausklammern
- Umsetzung mit NodeJS auf dem Server

# Thema

- Frei wählbar
- Sehr gerne Synergien mit IoT und/oder Frontend Development!

- Verteilte Anwendungen mit asynchroner Kommunikation sind schwierig
- Selbststudium herausfordernd und zeitintensiv
- Daher: Vorlesungstermine zum gemeinsamen Arbeiten nutzen

# Beispiel-Projekt

- Wir bauen gemeinsam in den Veranstaltungen eine verteilte Anwendung
- Wird die wesentlichen Aspekte der Anforderungen an die Projektarbeit beinhalten
- Projekt in Ruby
- Quellcode steht via GitHub zur Verfügung
- Wird als Beispiel für die Thematik Deployment genutzt

# Vorgehen

- Bildung von Projektgruppen (02.06.2022)
- Präsentation Konzept (09.06.2022)
- Nachvollziehbare Implementierung in GitHub
- Deployment er Anwendung
- Dokumentation und Betriebskonzept
- Abgabe von Code und Zugang zur Anwendung (08.09.2022)
- Präsentation im Plenum (15.09.2022)

# What's next?

- Literaturrecherche anhand der Leitfragen beginnen
- Gruppen bilden
- Projekt konzipiert haben und im Plenum vorstellen können
  - Welches Ziel wird verfolgt?
  - Welche Herausforderungen sind zu erwarten?
  - Welche Lösungen werden vorgeschlagen?



02. Juni 2022, 13:00 Uhr

# Literatur

[https://th-koeln.github.io/mi-bachelor-webdevelopment/  
lehrveranstaltungen/fddw-01/](https://th-koeln.github.io/mi-bachelor-webdevelopment/lehrveranstaltungen/fddw-01/)

# "Microservice Patterns & Antipatterns" - Stefan Tilkov

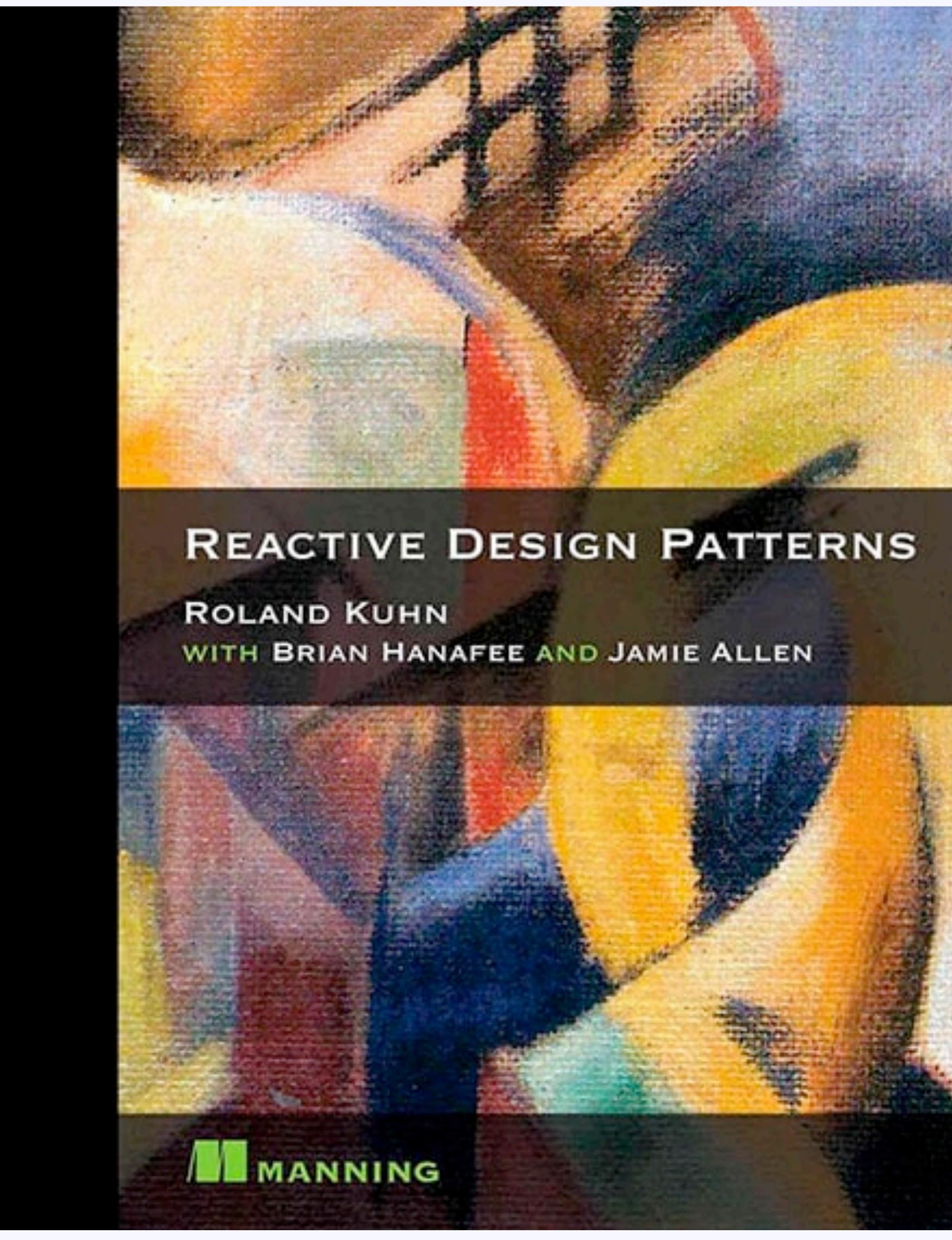


<https://www.youtube.com/watch?v=RsyOkifmaml>

# "Reactive Microsystems" - Jonas Bonér



<https://www.youtube.com/watch?v=3hMtjPcU248>



<https://th-koeln.sciebo.de/s/lL0Qmu5Hq3OzNKB>

# The Pragmatic Programmer



from journeyman  
to master

Andrew Hunt  
David Thomas

Foreword by Ward Cunningham

<https://pragprog.com/book/tpp/the-pragmatic-programmer>



Danke.