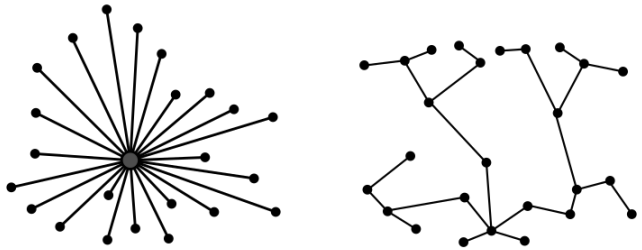
SYTD 2022/23

Florian Stanek – 4AHIT

Definition „Dezentrale Systeme“

- **System** (altgr. *sýstēma*)
  → „aus mehreren Einzelteilen zusammengesetztes Ganzes“

- **Dezentral** (oder auch **verteilt**)
  → Zusammenschluss voneinander **unabhängiger** Komponenten

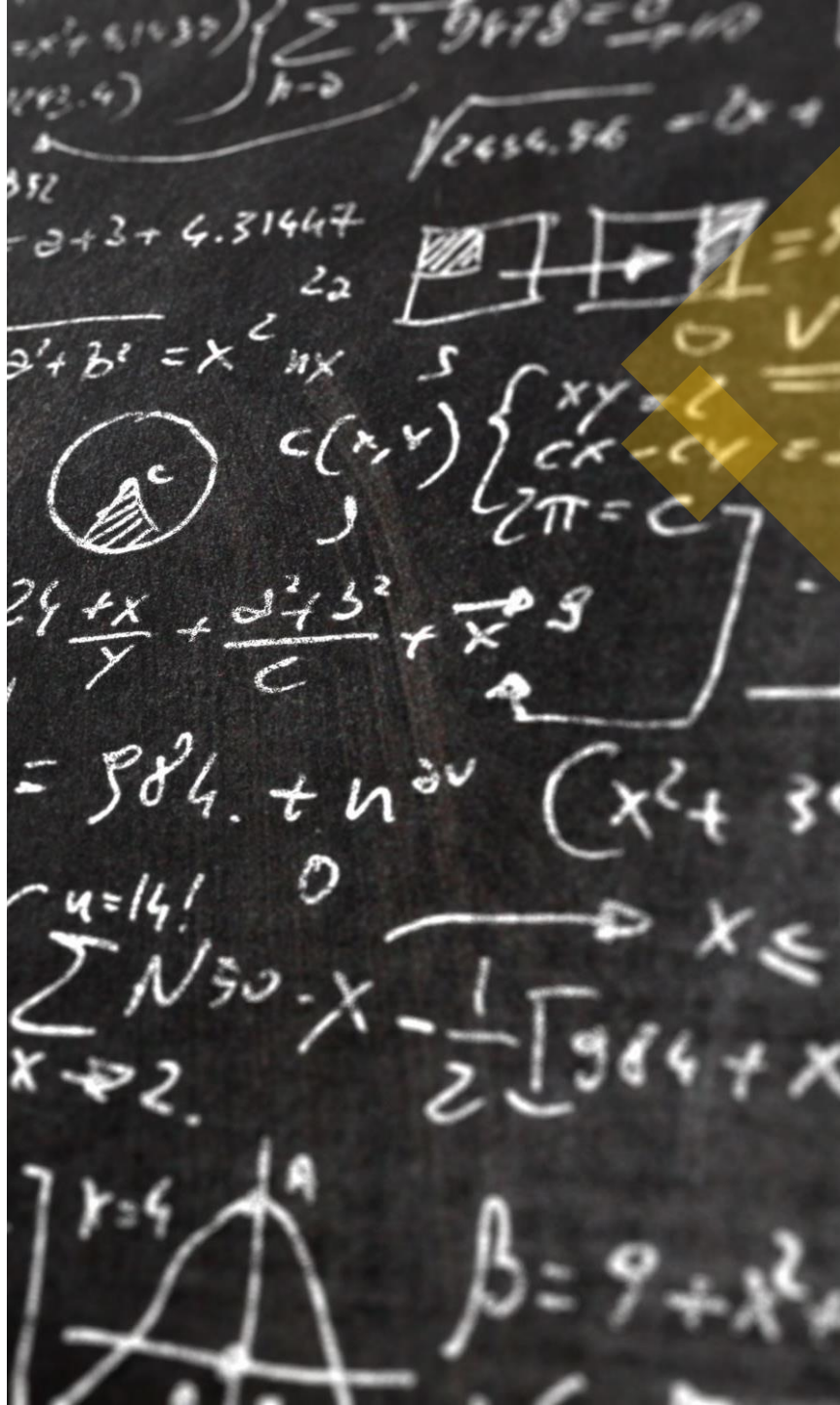


Vorausgesetztes Wissen

- Programmieren in C#
  - **OOP**: Klassen und Properties, Attribute, Interfaces, Vererbung, ...
  - **LINQ**: IEnumerable, List, Select, Where, OrderBy, ...

- Netzwerktechnik
  - **TCP/IP:** Ports, Sockets, ...
  - **URLs**

- Relationale Datenbanken
  - **Primär- und Fremdschlüssel, Relationen, Constraints, ...**
  - **SQL und Transaktionen**



Dezentrale/verteilte Systeme

- Verbund **mehrerer** Computer
  - im Gegensatz zu "*monolithic architecture*"
- Kommunikation über **Netzwerk**
  - oft mittels HTTP und JSON (oder XML)
- erscheinen dem Benutzer **wie einzelnes System**
  - „Web-Applikation“

→ **Service-Oriented Architecture** (SOA)

→ besteht aus einzelnen **Webservices/Web-Applikationen**

Webservice vs Web-Applikation

- **Webservice**
  - Für die Kommunikation von **Maschine zu Maschine**
  - Benutzt standardisierte **Protokolle**

→ **Backend**

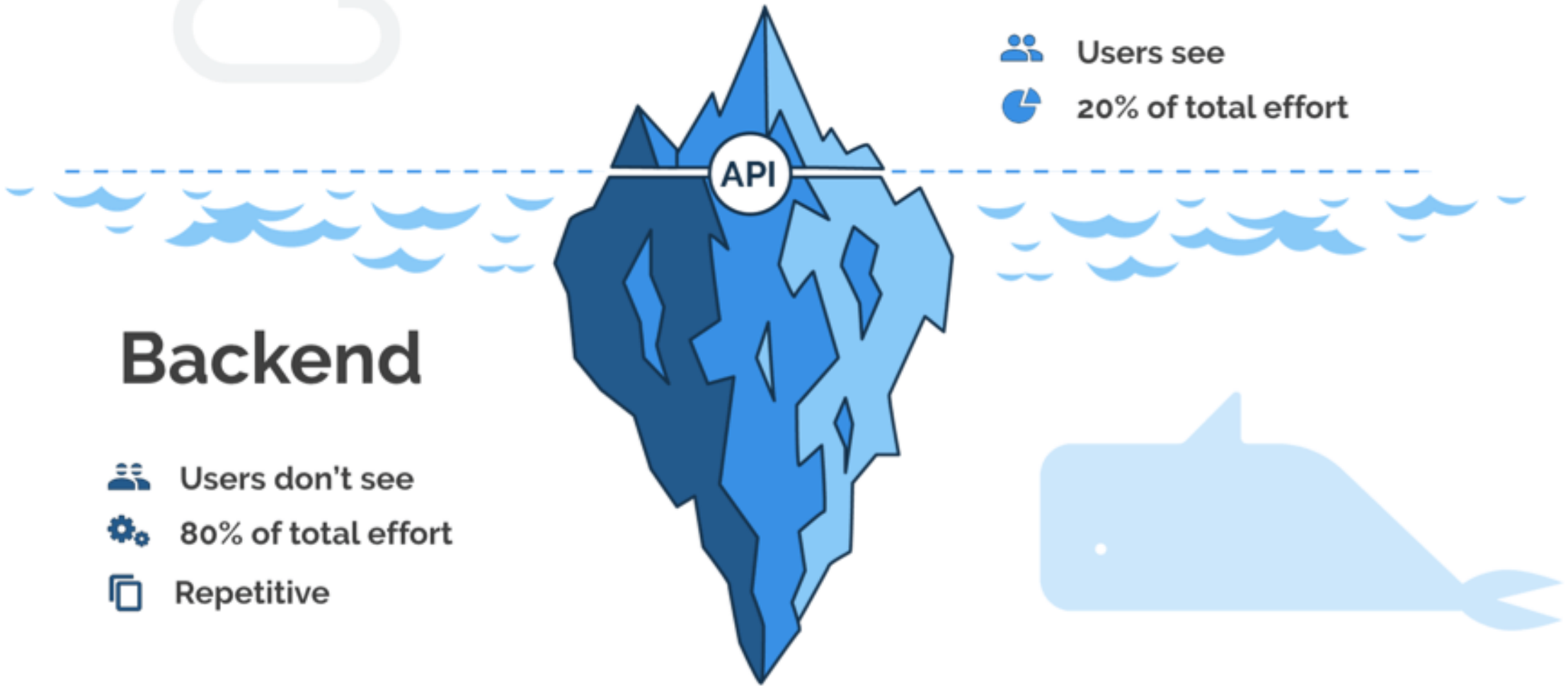
- **Web-Applikation**
  = Webservice **zusätzlich** mit **grafischer Oberfläche (GUI)**
  = Schnittstelle zwischen Benutzer und dezentralem System

→ **Frontend**

Frontend

Users see

20% of total effort



Backend

Users don't see

80% of total effort

Repetitive

Lernziele

- Technologien
  - ASP.NET Core
  - Entity Framework Core

- Konzepte
  - **Service-oriented Architecture**
    - auf Netzwerkebene („grob“) und Programmebene („fein“)
    - 3-Schichten-Modell: Data-Layer, Domain-Layer, Service-Layer
    - Dependency Injection
  - **Request-Response:** HTTP, REST, Routing
  - **Object-Relational Mapping**

Hello ASP.NET Core!

- Anlegen eines Projekts in der Praxis

- Benötigte Werkzeuge:
  - Visual Studio 2022 / Rider
  - Browser / Postman

Warum Verteilte Systeme?

Notwendigkeit:

- Ressourcen verteilt im Netzwerk nutzbar machen.

Motivation für verteilte Systeme

- **Parallele Verarbeitung** von Benutzer und Serviceanfragen
- **Shared Access:** Gemeinsame Nutzung von Betriebsmitteln z.B. Daten bzw. Dateien.
- **Modularisierung:** Die Modularisierung der Anwendung in einzelne Dienste erlaubt es Dienste in anderen Anwendungen wiederzuverwenden.
- **Replikation/Redundanz:** Speicherung von Daten auf mehreren Knoten im Netzwerk zur Erhöhung der Ausfallsicherhit.

Service-oriented Architecture

= anpassbares, flexibles Architekturmuster der Informationstechnik

- speziell für Verteilte Systeme

- **Modularisierung** von Software
  - Aufteilung von Softwareanwendungen in einzelne Module:
    - **einfach**
    - **verständlich**
    - **wartbar**

→ Module können unabhängig von anderen Teilen der Anwendung entwickelt werden

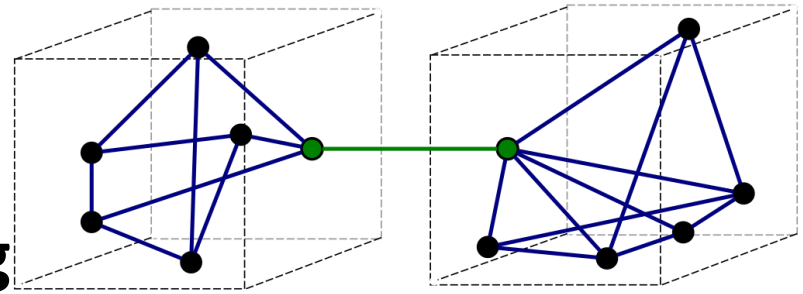
Kohäsion & Koppelung

- **Kohäsion**:
  - Ein Modul erfüllt einen spezifischen Aufgabenbereich der Anwendung.

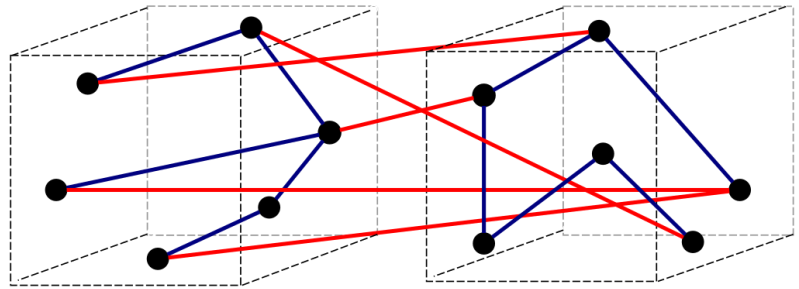
- **Koppelung**:
  - Module tauschen untereinander Nachrichten aus.

- **Module sind voneinander möglichst unabhängig**

  Projektmanagement ♥



a) Good (loose coupling, high cohesion)



b) Bad (high coupling, low cohesion)

Softwareservices

Module mit bestimmten Aufgaben werden auch Softwareservice genannt

- In C#: als Klasse realisiert

Service für seine eigenen Daten verantwortlich, verwaltet eigene Ressourcen

- Dependency Injection

ABER: Services einer Softwareanwendung können in unterschiedlichen Technologien implementiert sein.

- Keine Einschränkung auf eine bestimmte Programmiersprache oder Plattform.

Eigenschaften von verteilten System

- Transparenz
- Offenheit
- Skalierbarkeit

Transparenz

= Durchsichtigkeit

- Zugriffstransparenz
- Orts-/Positionstransparenz
- Replikationstransparenz
- Nebenläufigkeitstransparenz

Zugriffstransparenz

→ der Zugriff auf eine Ressource ist immer gleich

- Unterschiede in der
  - Art des Zugriffs auf Ressourcen
    - am selben Rechner, LAN, WAN, etc.
  - Datendarstellung
    - z.B. Datumsformate verschiedener Länder

Ortstransparenz/Positionstransparenz

- Benutzer können nicht erkennen, wo sich eine Ressource **physisch** innerhalb des Systems befindet

- Ressourcen werden durch virtuelle Name identifiziert
  - z.B. Verwendung von URLs → keine Information über Speicherort http://serviceurl/employee/205

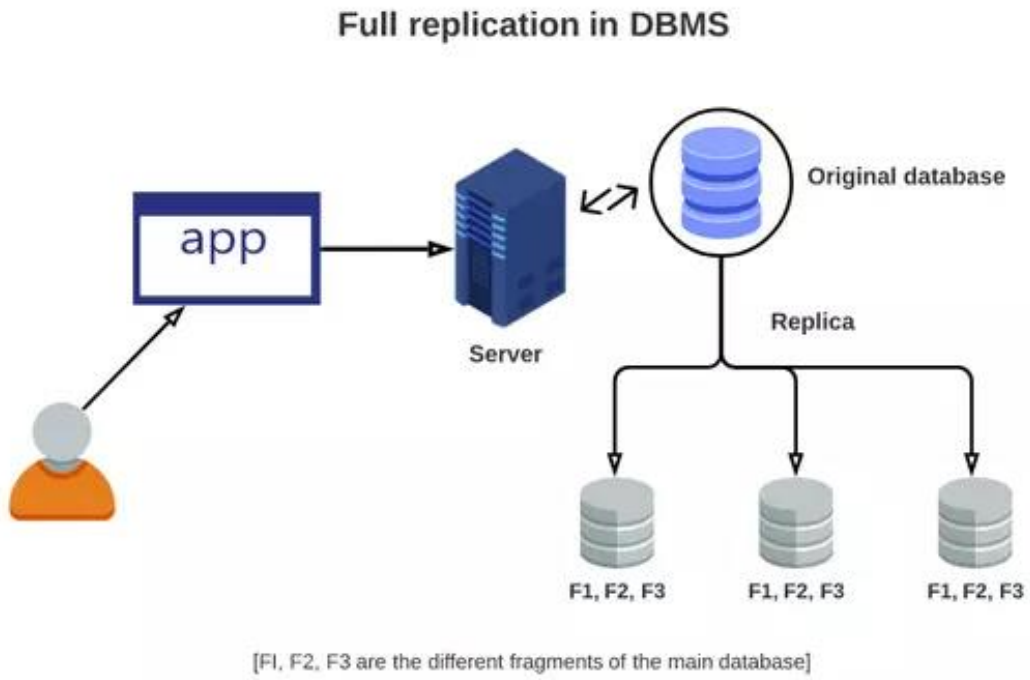
- Herkunft der Daten bleibt verschleiert
  - z.B. aus Datenbank, von Filesystem, von anderem Microservice, usw.

→Ressourcen können beliebig **verschoben/ausgetauscht** werden!

Replikationstransparenz

- Verbirgt, dass mehrere Kopien einer Ressource existieren
- Alle Kopien müssen den selben Namen haben

→ setzt Ortstransparenz vorraus!



Nebenläufigkeitstransparenz

→Zugriff muss problemlos funktionieren, egal wieviele Benutzer zur selben Zeit zugreifen

- Verteilte Systeme ermöglichen gemeinsame Nutzung von Ressourcen und Diensten
  - z.B. Datenbanken

- Herausforderungen:
  - Synchronisierung
  - Datenkonsistenz

Offenheit

- Offenheit bedeutet, dass ein verteiltes System einfach erweitert und verändert werden kann

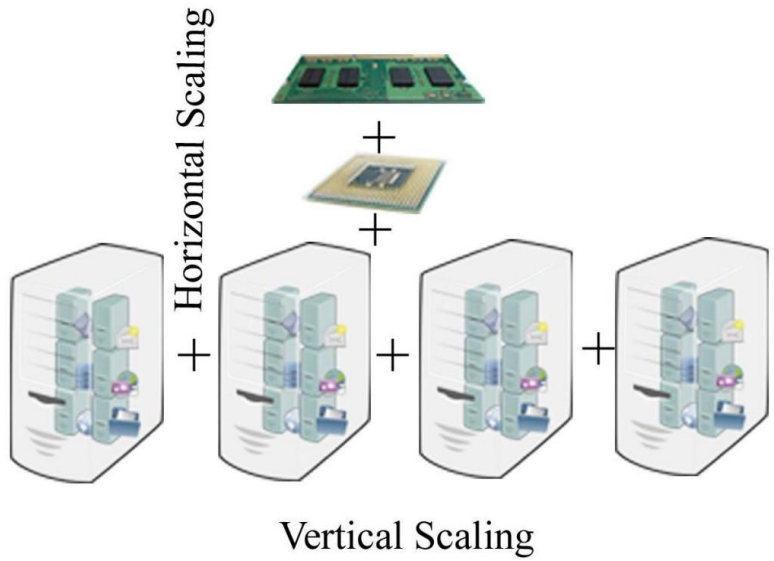
- Vorraussetzung:
  - Gliederung eines Systems in voneinander unabhängige Komponenten (→ SOA)

- Offenheit wird gewährleistet durch:
  - Einheitlicher Kommunikationsmechanismus (z.B. HTTP)
  - Veröffentlichte Schnittstellen (z.B. URLs)

Skalierbarkeit

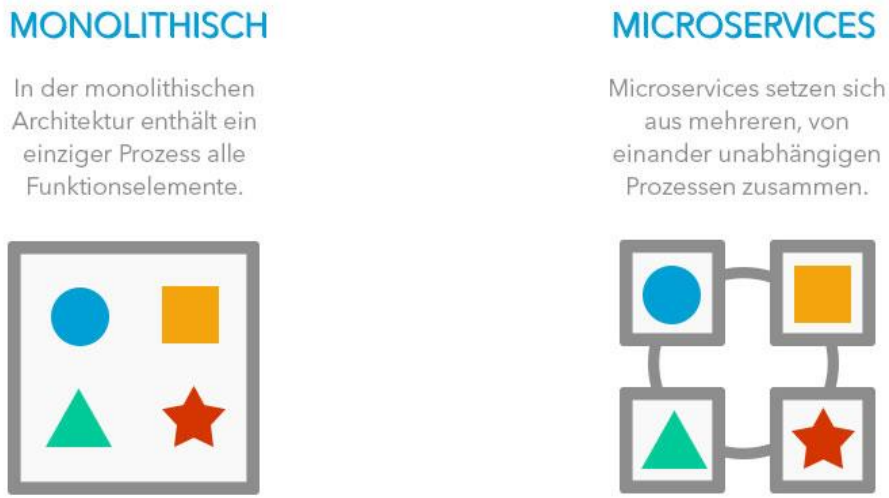
= Fähigkeit, bei steigender Last (Zugriffen) die Funktionalität beizubehalten

- Vertikale Skalierung:
  - Hinzufügen von Ressourcen innerhalb einer logischen Einheit
  - z.B. mehr Arbeitsspeicher, mehr Prozessorkerne
- Horizontale Skalierung:
  - Hinzufügen von weiteren logischen Einheiten
  - z.B. mehr Rechner



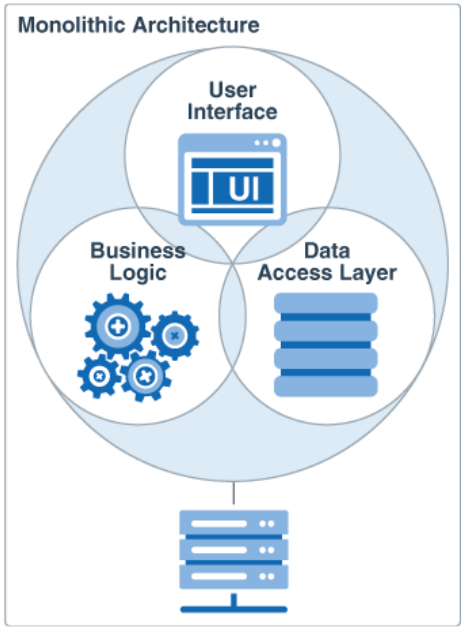
Architekturmuster: Monolith vs Microservices

- Architekturmuster beschreiben die grundlegende Organisation und Interaktion zwischen den Modulen einer Softwareanwendung.
- Für Verteilte Systeme wählt man je nach geforderter Komplexität eines der folgenden drei Architekturmuster:
  - Monolithische Architektur
  - Client-Server-Architektur
  - Microservice-Architektur



Monolithische Architektur

- für Entwicklung einfacher Anwendungen
- keine explizite Gliederung in Teilsysteme oder Komponenten



Vorteile der monolithischen Architektur

- Einfachere Entwicklung für simple Geschäftsprozesse
- IDEs/Entwicklungswerkzeuge optimiert für die Entwicklung monol. Architekturen.
- Einfaches Testen

→ ABER: Die monol. Architektur hat erhebliche Einschränkungen

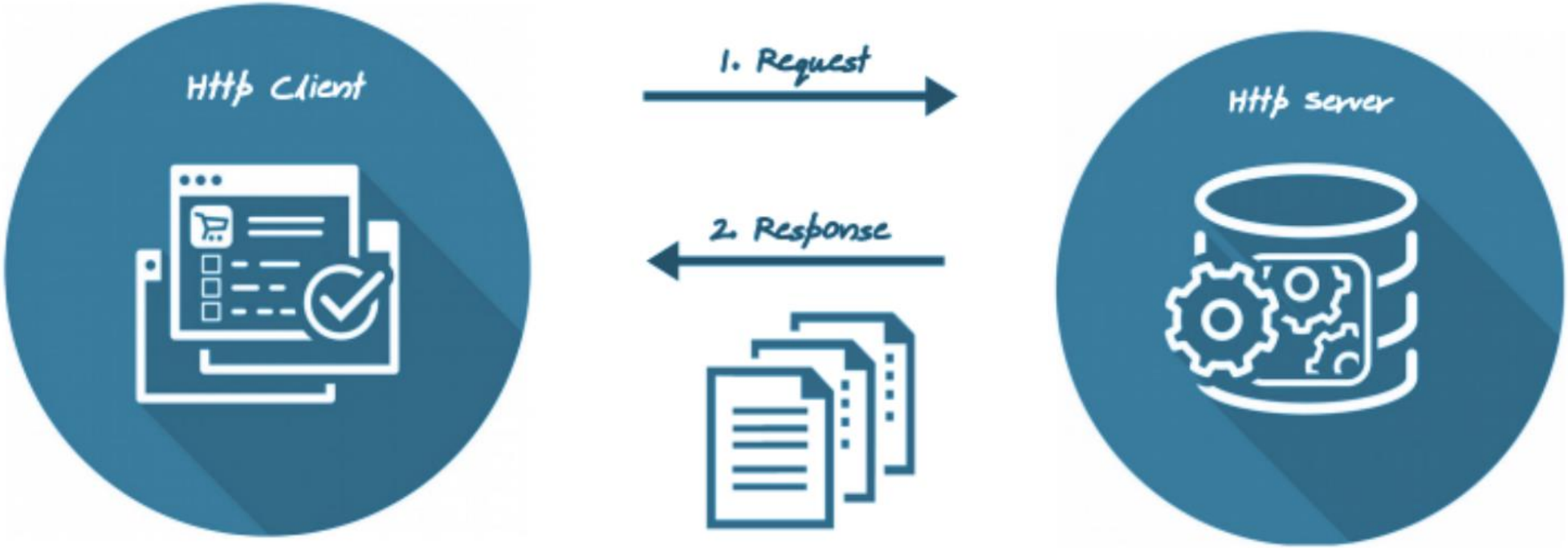
Nachteile der monolithischen Architektur

- Evolution von Webanwendungen
  - Stetige Weiterentwicklung (CI/CD)
  - erfolgreiche Webanwendungen bieten dem Benutzer mit der Zeit zusätzliche Geschäftsfelder an

  → Komplexität ursprünglich einfacher Geschäftsprozesse nimmt mit der Zeit immer mehr zu
- zunehmende Codekomplexität bei Erweiterungen
  - Je größer die Anzahl an Geschäftsprozessen, desto stärker steigt die Codekomplexität

  → Entwickler müssen immer mehr Code „durchforsten“
- Zuverlässigkeit des Gesamtsystems bei komplexen Anwendungen
  - Die Module der Geschäftslogik werden alle im selben Betriebssystemprozess ausgeführt.
- schnelle (agile) Entwicklung durch hohe Komplexität ist erschwert
  - sinnvolle agile Entwicklung der Anwendung erschwert bei monol. Architektur

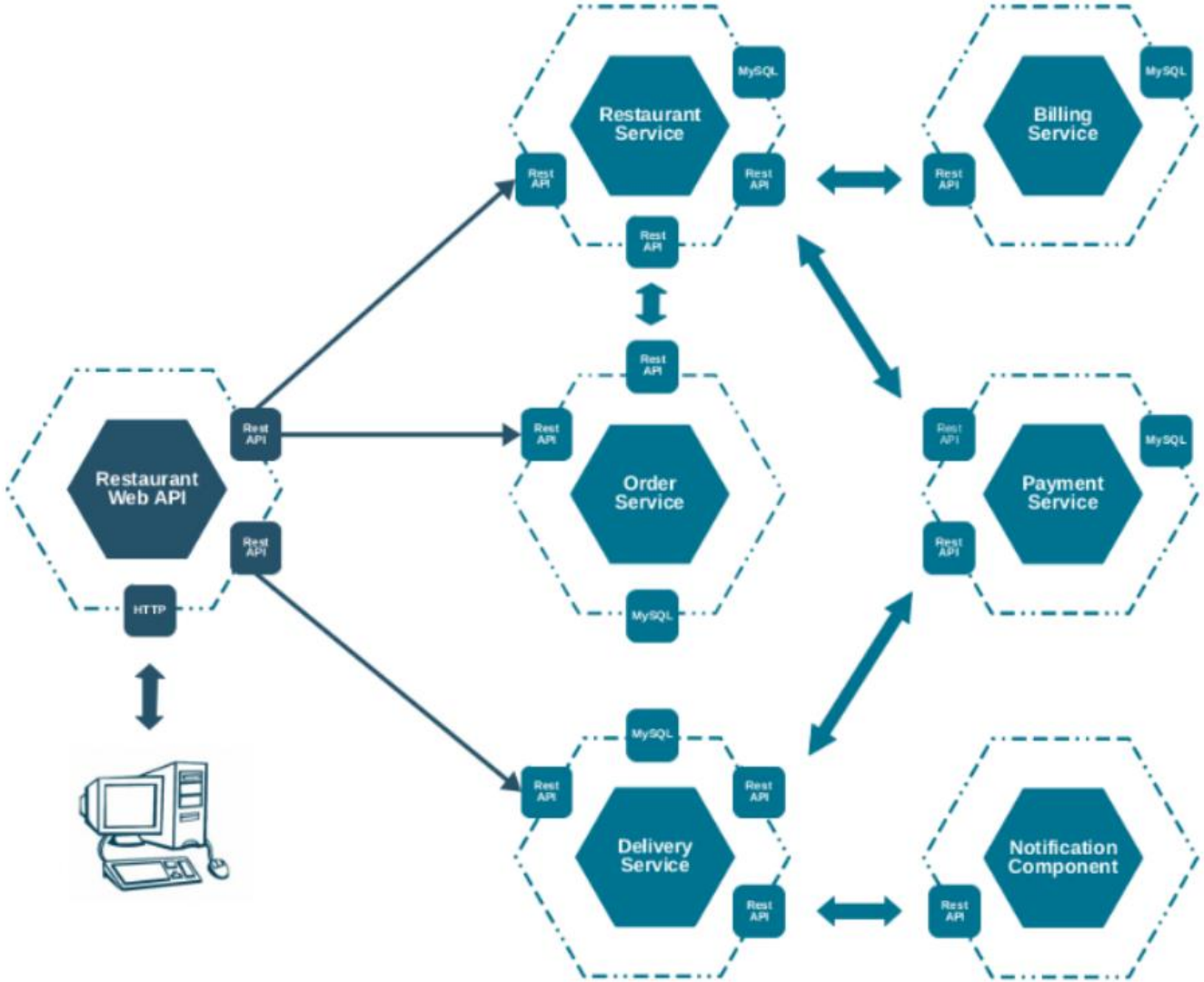
Client-Server-Architektur



Microservice-Architektur (SOA)

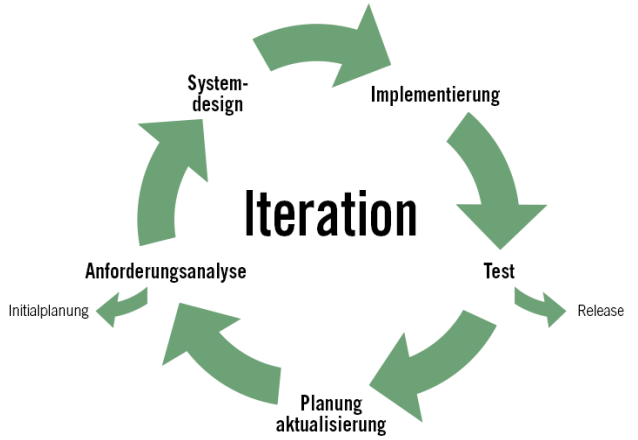
- Das SOA Entwurfsmuster beschreibt ein System von Services die untereinander Nachrichten austauschen.
- (Micro-)Service = **self-contained system**

  → es enthält alle Abhängigkeiten für eine autonome Ausführung



Vorteile der Microservice-Architektur 1/3

- Agile Softwareentwicklung
  - Microservices begünstigen den Einsatz iterativer Projektmanagementmodelle



- Hohe Wartbarkeit:
  - Jedes Service besitzt eine klar definierte Grenze mittels Schnittstellen
  - Wenn nicht mehr wartbar ist: Ersatz durch einen neues Service

  → Voraussetzung: Schnittstellen von Services müssen unverändert bleiben.

Vorteile der Microservice-Architektur 2/3

- Offenes System
  - Service können in beliebigen Programmiersprachen implementiert werden.

  → Auswahl der besten Technologie für die Implementierung eines Services

  → erleichterte Schritt-für-Schritt-Umstellung auf neue Technologie

- Isoliertheit: Hoher Grad der Entkoppelung für zu
  - Unabhängige Skalierung
  - Technologiefreiheit
  - Schutz vor Ausfällen

Vorteile der Microservice-Architektur 3/3

- Robustheit
  - Speicherleak in Microservice
    → nur dieses Service stürzt ab
    → alle anderen Services stellen weiterhin verfügbar

- Skalierbarkeit
  - Jedes Microservice kann unabhängig von den anderen Microservices skaliert werden