01

iu INTERNATIONALE HOCHSCHULE

PROGRAMMIERUNG VON WEB-AWENDUNGEN

EINFÜRHUNG

I. INTERNET UND WEB-ANWENDUNGEN

1. Einführung: Bedeutung und Einfluss des Internets

- Internet als größtes globales Netzwerk
- Verbindet alle Kontinente miteinander
- Nicht nur Sprachverbindungen, sondern auch Breitband-Datenanbindungen
- Revolutioniert:
  - Soziale Netzwerke
  - Organisation von Gruppen
  - Zugang zu Informationen, Nachrichten & Bildung
- Vorteile: Schneller Informationsaustausch weltweit, Aufbau neuer sozialer Strukturen , Demokratisierung von Wissen und Information
- Nachteile: Rückgang persönlicher Kontakte (z. B. Vereinsamung), Anonymität ermöglicht kriminelle Strukturen (Datenklau, Viren, Cyberangriffe)

2. Historie und Aufbau des Internets

2.1 Definition & Grundkonzept

- „Internet“ = „Interconnected Network“
- Netzwerk, das viele Computer miteinander verbindet
- Erweiterung des klassischen Heimnetzwerks:
  - Heimnetz: lokal, nur in Wohnung/Unternehmen
  - Internet: global, länder- und kontinentübergreifend
- Ermöglicht weltweiten Datenaustausch

2.2 Lokale Netzwerke & globale Anbindung

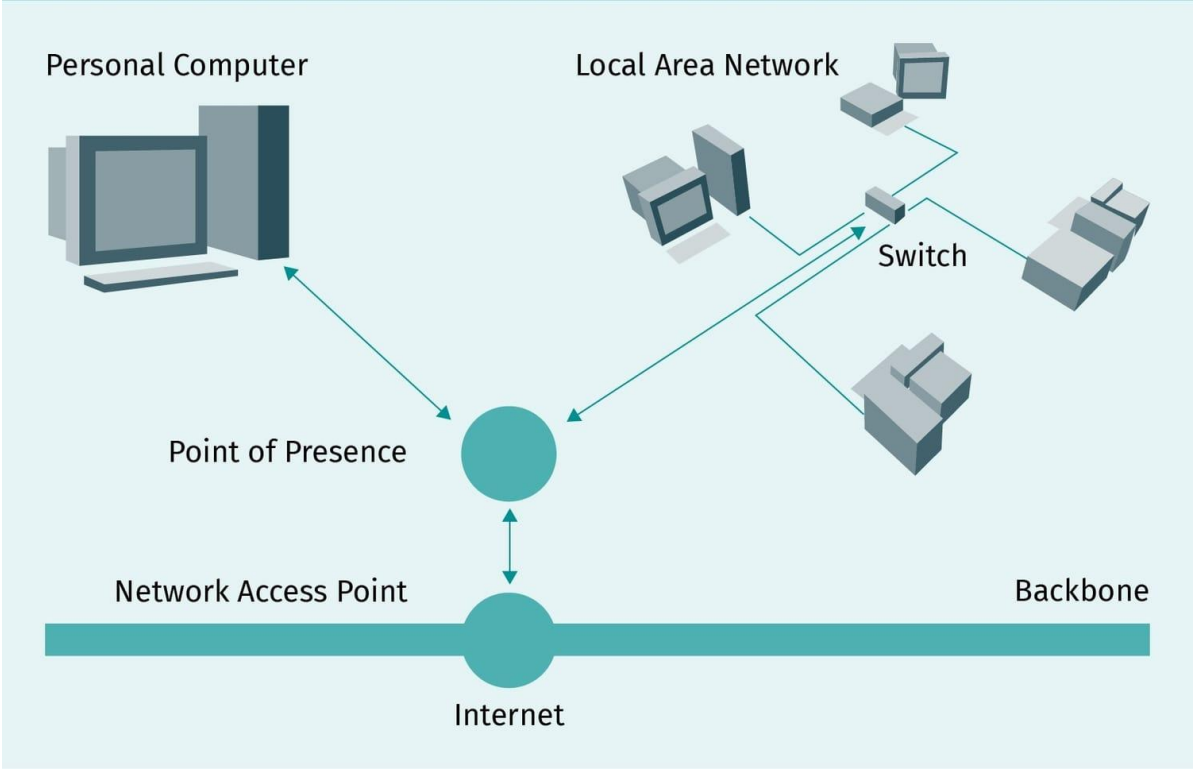
- Lokale Netzwerke: Computer über „Switches“ verbunden
- Internet erfordert organisierten Zugang:
  - Internet Service Provider (ISP) = „Provider“
  - Aufgaben des Providers:
    - Bereitstellung von Hardware (z. B. Router)
    - Software für Zugang und Konfiguration
- Repeater:
  - Signalverstärkung in lokalen, kabellosen Netzwerken
  - Erhöhung der Reichweite des WLAN-Signals

2.3 IP-Adressen – Adressierung im Internet

- IP-Adresse = „Internet-Protokoll-Adresse“
  - Eindeutige Adresse jedes Geräts im Netzwerk
  - Vergleichbar mit einer Telefonnummer
- Jede Kommunikation benötigt eindeutige IP-Adressen
- IP-Adressen ermöglichen gezielte Adressierung & Kommunikation

2.4 Aufbau des Internetzugangs

- Provider stellt Zugang über:
  - „Point of Presence“ (POP) = Einwahlknoten
  - POPs sind mit dem Backbone-Netz des Providers verbunden
  - Verbindung über „Network Access Point“ (NAP)
- Backbone-Netz = Hauptnetzwerkstruktur eines Providers
- Verbindung zu Endgeräten meist über:
  - Kupferkabel (ältere Technologie)
  - Glasfaserverbindungen (heutiger Standard, höhere Bandbreiten)



2.5 NAT (Network Address Translation)

- Heimnetzwerk: private IP-Adressen (lokal)
- Router übersetzt lokale IP-Adressen in öffentliche IP-Adresse
- Anfrage ins Internet wird mit öffentlicher IP gesendet
- Serverantwort wird an öffentliche IP-Adresse zurückgeschickt
- Router übersetzt wieder in lokale IP-Adresse → Weiterleitung an das Endgerät

2.6 Historische Entwicklung: ARPAnet und militärischer Ursprung

- Ursprung in den 1950er-Jahren: militärischer Kontext
- „ARPAnet“ (Advanced Research Projects Agency Network)
  - Ziel: Großrechner miteinander vernetzen
  - Datenkommunikation über weite Distanzen
- Motivation:
  - Ausfallsicherheit: Netzwerke mit mehreren Knoten
  - Bei Ausfall eines Knotens können andere übernehmen
- Anfangs nur für staatliche/militärische Zwecke
- Spätere Entwicklung zum heutigen öffentlichen Internet

2.7 Meilensteine der Internetentwicklung

1969: Start von ARPAnet

- Beginn der praktischen Vernetzung: **nur 4 Großrechneranlagen**
- Verbindungsaufbau über **IMP** (Interface Message Processor) – Spezialcomputer zur Paketvermittlung
- Ziel: Sicherer Datenaustausch zwischen militärischen Einrichtungen

1971: Vorstellung des ARPAnet

- ARPAnet wird der **Öffentlichkeit vorgestellt**
- Zu diesem Zeitpunkt: **ca. 15 Netzknoten**
- **Einsatzbereiche:** Militärisch, Wissenschaftlich
- **MILnet**: Militärisches Netzwerk – vom zivilen Teil **abgekoppelt** aus Sicherheitsgründen

2.7 Meilensteine der Internetentwicklung

1989: Wachstum & Übergang

- ARPAnet umfasst ca. **100.000 angebundene Host-Computer**
- Umstellung auf das **NSFnet** (National Science Foundation Network)
    - Modernere Struktur
    - **Kommerzielle Nutzung ab 1990** → Beginn des Internets als globales Kommunikationsmittel

Frühe technische Grenzen

- Sehr geringe Datenübertragungsraten: **9.600 bps (bit/s)**
- Beispiel: Upload eines **3 MB-Fotos** = ca. **43 Sekunden**
- Vergleich: Moderne Glasfaser-Netze erreichen heute Millionen bit/s

2.8 Wissenschaftliche Entwicklungen – Basis für das WWW

Tim Berners-Lee & CERN

- **Entwicklung von HTML (Hypertext Markup Language)**
  - Durch Tim Berners-Lee am CERN
  - Sprache zur Strukturierung von Inhalten im Internet

1993: Einführung des World Wide Web

- **WWW (World Wide Web)** = revolutionärer Internetdienst
  - Präsentation und Navigation von Hypertext-Dokumenten
  - Machte das Internet **grafisch zugänglich** für breite Öffentlichkeit
- Erster Browser: **Mosaic**
  - Unterstützt Text, Bilder, Links → Grundstein moderner Webbrowser

2.9 Das Internet im 21. Jahrhundert

Exponentielles Wachstum

- Seit **Anfang der 2000er**: rasanter Ausbau des Internets
- **2018**:
  - Über **1 Milliarde** Host-Computer am Netz
  - Rund **20 Milliarden internetfähige Geräte**
  - Im Vergleich: ca. **7 Milliarden Menschen weltweit**

Datenflut & Big Data

- **Tägliches Datenaufkommen 2018**: ca. **5 Exabyte** (= 5 Milliarden GB)
- Vergleich: Das entspricht der **12.500-fachen Menge** aller jemals geschriebenen Bücher
- Herausforderung: Speicherung, Analyse, Datenschutz

3. Architektur von Web-Anwendungen

3.1 Grundlagen einer Web-Anwendung

- Eine Website ist vereinfacht eine Datei auf einem Server
- Server = z. B. **Apache**, **nginx**
- Website-Inhalte liegen typischerweise im **HTML-Format** vor
- HTML (Hypertext Markup Language):
  - Beschreibt Struktur & Inhalt einer Webseite
  - Wird vom **Browser interpretiert und angezeigt**

3.2 Trennung von Inhalt und Darstellung

- **CSS** (Cascading Style Sheets):
  - Definiert das Aussehen (Design) der HTML-Elemente
  - Beispiele: Farben, Schriftarten, Abstände, Layouts
- Vorteile:
  - Einheitliches Design über mehrere Seiten hinweg
  - Trennung von Struktur (HTML) und Stil (CSS)

3.3 XML und XSL

- **XML** (Extensible Markup Language):
  - Metasprache zur Definition strukturierter Daten
  - Flexibel & hierarchisch (z. B. Lagerbestand, Artikellisten)
  - Bedeutung der XML-Tags wird über DTD (Document Type Definition) geregelt

- **XSL** (Extensible Stylesheet Language):
  - Wandelt XML-Daten in HTML-Dokumente um
  - Ermöglicht visuelle Aufbereitung strukturierter Daten

4. Client- und Serverseitige Architektur

4.1 Statische Webseiten

- Inhalte sind fix hinterlegt (z. B. einfache HTML-Datei)
- Änderungen nur durch direkte Änderung am Code („**Hardcoding**“)
- Immer dieselbe Ausgabe bei jedem Seitenaufruf

4.2 Client-seitige Dynamik

- Ergänzung statischer Webseiten durch **JavaScript**
- Ausführung direkt im **Browser des Nutzers**
- Mögliche Funktionen: Benutzerinteraktionen (Formulare, Klicks) ,Dynamische DOM-Manipulation (Ändern der Seite ohne Neuladen), Einfache Berechnungen
- JavaScript wird direkt in HTML eingebettet oder als separate Datei geladen

4.3 Server-seitige Dynamik

- Server erzeugt die Webseite dynamisch je nach Anfrage
- Technologien:
  - **CGI (Common Gateway Interface)**: Schnittstelle zwischen Server & Skript
  - **PHP**: weit verbreitete serverseitige Sprache
- Funktionsweise:
  - Anfrage vom Browser → Server führt Skript aus → HTML-Antwort an Browser
- Vorteile:
  - Zugriff auf Datenbanken
  - Dynamischer, kontextabhängiger Inhalt

4.4 Hybride Ansätze

- Kombination von client- und serverseitiger Dynamik
- z. B. SPA (Single Page Applications):
  - Initiale HTML-Seite vom Server
  - Danach Kommunikation über **AJAX/REST-APIs**

5. Internetprotokolle & Kommunikation

5.1 Grundlagen

- Standardisierte Kommunikation = **Protokolle**
- Wichtigste Internetprotokolle:
  - **IP (Internet Protocol)**: Adressierung & Routing
  - **TCP (Transmission Control Protocol)**: Zuverlässiger Datentransfer

5.2 Datenübertragung mit TCP/IP

- Daten werden in **Pakete** (max. 1.500 Bytes) zerlegt
- Header (je 20 Byte für TCP & IP) → **1.460 Bytes** Nutzdaten
- Pakete enthalten Zieladresse (**IP-Adresse**)
- Empfänger setzt Pakete zur ursprünglichen Nachricht zusammen
- Vorgang wird als **Routing** bezeichnet

5.3 IPv4 vs. IPv6

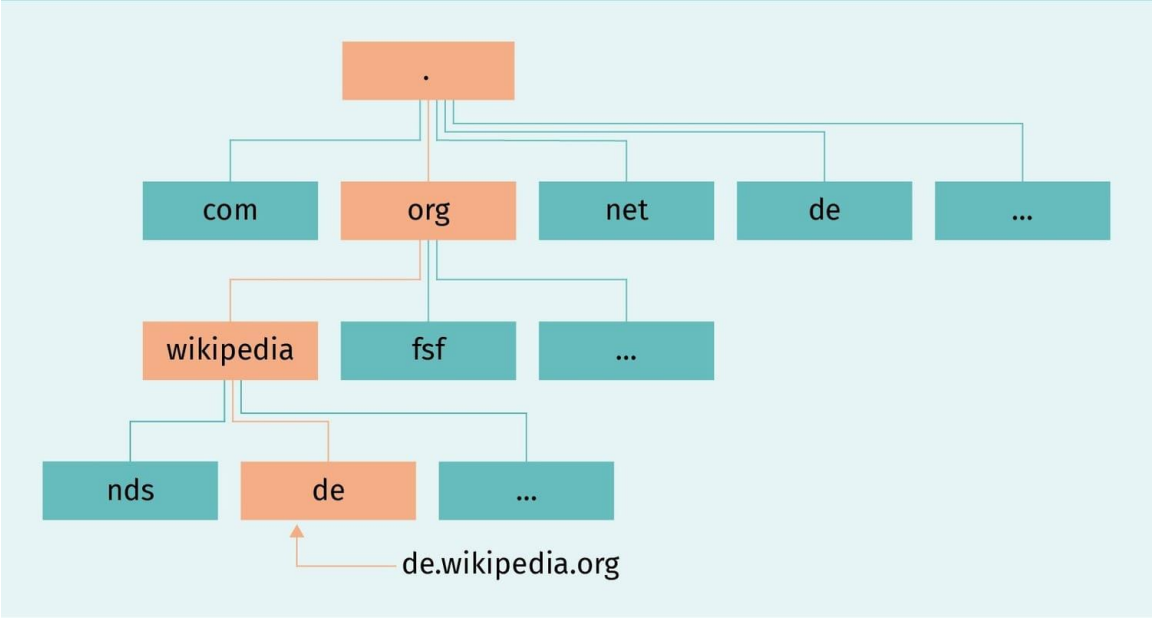
Merkmal	IPv4	IPv6
Adressgröße	32 Bit (4 Dezimalblöcke)	128 Bit (8 Hexadezimalblöcke)
Adressraum	ca. 4,3 Mrd. Adressen	2^{128} Adressen (~3,4 x 10^{38})
Darstellung	z. B. 192.168.0.1	z. B. 2001:0db8:85a3::8a2e:0370:7334
Motivation	Erschöpfung durch IoT etc.	Zukünftige Geräteflut (IoT, Industrie 4.0)

6. Domain Name System (DNS)

6.1 DNS – Name statt Nummer

- Jeder Server hat eine IP-Adresse – aber schwer zu merken
- **DNS** wandelt Domainnamen in IP-Adressen um
- Beispiel: www.iu-fernstudium.de → 212.77.238.122

- Hierarchischer Aufbau:
- TLD (Top-Level-Domain): .de, .com, .org
- SLD (Second-Level-Domain): iu-fernstudium
- Subdomains: de.wikipedia.org



6.2 Dynamisches DNS (DDNS)

- Problem: dynamisch vergebene IP-Adressen (Privathaushalte)
- Lösung: DDNS aktualisiert Zuordnung regelmäßig
- Beispiel: Heimserver bleibt unter konstanter URL erreichbar
- Risiko: leichtere Auffindbarkeit durch Angreifer

7. URI, URL, URN – Adressierung von Ressourcen

7.1 Begriffe und Unterschiede

Begriff	Beschreibung
URI (Uniform Resource Identifier)	Allgemeiner Bezeichner
URL (Uniform Resource Locator)	Gibt exakte Adresse & Pfad an
URN (Uniform Resource Name)	Dauerhafte Namensgebung (z. B. ISBN)

7.2 URL-Schema – Beispiel

Struktur einer URL:

Protokoll: //Server.Domain-Name/Ordner/Datei

z. B. https://www.iu-fernstudium.de/bachelor/bachelorstudiengaenge

- https: Protokoll (Hypertext Transfer Protocol Secure)
- www.iu-fernstudium.de: Server & Domain
- /bachelor/bachelorstudiengaenge: Pfadangabe auf dem Server
- Automatischer Fallback auf index.html, wenn keine Datei angegeben

8. Qualität von Web-Anwendungen

8.1 Bedeutung von Qualität

- Qualität ist **subjektiv geprägt**: Entwickler und Nutzer haben unterschiedliche Vorstellungen
- Ziel: **Objektivierbare Kriterien** zur Bewertung finden → Qualitätskriterien einer Web-Anwendung nach ISO 25010

Qualitätskriterium	Beschreibung
Effectiveness	Vollständigkeit und Genauigkeit, mit der ein Benutzer spezifische Ziele erreichen kann.
Efficiency	Resourcenaufwand, mittels dessen ein Benutzer die Vollständigkeit und Genauigkeit gesetzter Ziele erreichen kann.
Satisfaction	Grad der Erfüllung der Benutzerbedürfnisse.
Freedom from risk	Grad, zu dem eine Anwendung das Risiko bzgl. einer Gefährdung des ökonomischen Status, des menschlichen Lebens, der Gesundheit oder der Umwelt verringert.
Context coverage	Grad, zu dem eine Anwendung oder ein System mit Effizienz, Effektivität, der Abwesenheit von Risiken und Zufriedenstellung in dem primär angedachten Kontext und in einem leicht davon abweichenden Kontext genutzt werden kann.

8.2 ISO 25010 – Standardisierte Qualitätskriterien

- Internationale Norm zur Softwarequalität
- Unterteilt in **funktionale** und **nicht-funktionale** Kriterien
- Fokus: **Hauptkategorien** der Qualitätsattribute

8.3 Funktionale vs. nicht-funktionale Qualität

Typ	Beschreibung
Funktionale Kriterien	Entspricht die App den spezifizierten Funktionen?
Nicht-funktionale Kriterien	Antwortzeit, Verfügbarkeit, Stabilität, Kompatibilität, Usability etc.

8.3 Konformität & Fehlerfreiheit

Ziel: möglichst **fehlerfreie Darstellung** & Funktion

- Unterstützung durch Standards & Tools:
  - **W3C (World Wide Web Consortium)**:
    - Internationale Organisation zur Webstandardisierung
    - Standards: HTML, CSS, XML etc.
  - **W3C Validatoren**:
    - Prüfen HTML-/CSS-Code auf Konformität mit W3C-Standards

8.4 Kompatibilität

- Unterschiedliche **Browser**: Chrome, Firefox, Safari, Edge, Opera etc.
- Unterschiedliche **Betriebssysteme**: Windows, macOS, Linux, iOS, Android
- Unterschiedliche **Endgeräte**: Desktop-PCs, Tablets, Smartphones, Smart-TVs, Sprachassistenten

8.5 Responsive Design & Adaptivität

- **Webseiten müssen sich verschiedenen Bildschirmgrößen anpassen**
- **Ziel: Einheitliche User Experience auf allen Geräten**
- Techniken: Media Queries (CSS), Mobile First Design, Adaptive Layouts

8.6. Usability – Benutzerfreundlichkeit

8.6.1 Definition nach ISO

- Usability = **Zielerreichung durch Benutzer im Anwendungskontext**
- Wichtig bei alltäglichen Anwendungen & Business-Webtools

8.6.2 Faktoren guter Usability z.B.

- Intuitive Navigation, Verständliche Inhalte & klare Sprache, Konsistentes Layout
- Schnelle Ladezeiten, Feedbacksysteme (z. B. bei Fehlern)
- Zugänglichkeit (Accessibility, z. B. für Menschen mit Behinderungen)

8.7 Herausforderungen bei der Qualitätssicherung

8.7.1 Technische Vielfalt

- **Plattformunabhängigkeit** erforderlich
- Browser & Geräte verhalten sich oft unterschiedlich
- Beispiel: Unterschiedliches Rendering von CSS in Safari vs. Chrome

8.7.2 Integration in Entwicklung

- Qualitätssicherung muss:
  - Frühzeitig im Entwicklungsprozess starten (Test-Driven Development)
  - Automatisiert werden (z. B. Unit Tests, CI/CD Pipelines)
  - Regelmäßig mit echten Nutzern geprüft werden (Usability-Tests)

Fazit : Qualität von Web-Anwendungen

- Qualität von Web-Anwendungen ist **mehrdimensional**
- ISO 25010 bietet verlässliche, objektive Kriterien
- Besondere Herausforderungen: **Vielfalt von Geräten, Systemen, Browsern**
- Tools & Standards (W3C, Validatoren) helfen, Fehler zu vermeiden
- Gute Usability = Schlüsselfaktor für Akzeptanz & Erfolg

9. Client-Server, 3-Schichten-Architektur, Model-View-Controller

9.1 Softwarearchitekturen im Wandel

- Frühe Software: einfache Programme (z. B. Zinsberechnung, Buchhaltung)
- Keine Modularisierung, monolithische Struktur
- Änderung einer Funktion = Änderung des gesamten Programms
- Wartung aufwendig, keine Trennung von Darstellung, Logik, Daten

→ *Software mit einfacher Architektur*

Nachteile

- Keine klare Aufgabentrennung (z. B. Berechnung, Darstellung, Speicherung)
- Änderungen in einem Bereich erfordern oft vollständige Neukompilation
- Beispiel: Alle Funktionen in einer einzigen Klasse im Online-Shop
- Hohe **Wartungskosten** und **Fehleranfälligkeit**

9. Client-Server, 3-Schichten-Architektur, Model-View-Controller

9.1 Softwarearchitekturen im Wandel

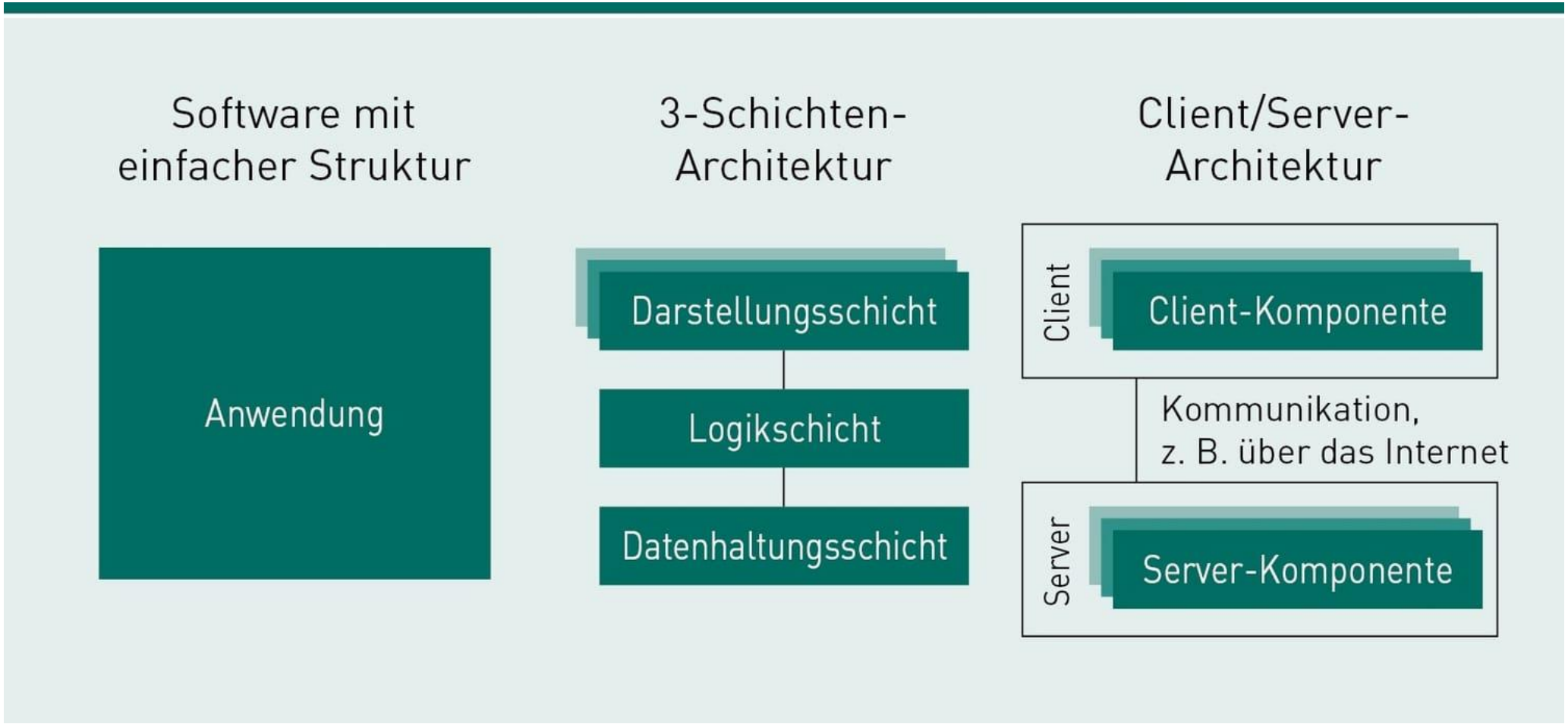
- Frühe Software: einfache Programme (z. B. Zinsberechnung, Buchhaltung)
- Keine Modularisierung, monolithische Struktur
- Änderung einer Funktion = Änderung des gesamten Programms
- Wartung aufwendig, keine Trennung von Darstellung, Logik, Daten

→ *Software mit einfacher Architektur*

Nachteile

- Keine klare Aufgabentrennung (z. B. Berechnung, Darstellung, Speicherung)
- Änderungen in einem Bereich erfordern oft vollständige Neukompilation
- Beispiel: Alle Funktionen in einer einzigen Klasse im Online-Shop
- Hohe **Wartungskosten** und **Fehleranfälligkeit**

9. Client-Server, 3-Schichten-Architektur, Model-View-Controller



9. Client-Server, 3-Schichten-Architektur, Model-View-Controller

9.2 Lösung: 3-Schichten-Architektur (Three-Tier Architecture)

- Ziel: Trennung von Verantwortlichkeiten
- Besser wartbar & flexibel erweiterbar
- **Drei klar getrennte Schichten:**

— **1. Darstellungsschicht (Presentation Layer)**

- Benutzeroberfläche
- Entgegennahme von Eingaben, Ausgabe von Ergebnissen

— **2. Logikschicht (Business Logic Layer)**

- Geschäftsregeln, Prozesse
- Beispiel: Bestellabwicklung, Preisberechnung

— **3. Datenhaltungsschicht (Data Access Layer)**

- Datenzugriff: Erstellen, Lesen, Aktualisieren, Löschen (CRUD)
- Verbindung zu Datenbanken

9. Client-Server, 3-Schichten-Architektur, Model-View-Controller

9.3 Schichtenkommunikation in der Architektur

- Nur Kommunikation **zwischen benachbarten Schichten** erlaubt
- Darstellung ↔ Logik ↔ Datenhaltung
- Kein direkter Zugriff z. B. von Darstellung auf Datenbank

— **Vorteile der 3-Schichten-Architektur**

- **Austauschbarkeit von Komponenten**
- Leichte Anpassung der Benutzeroberfläche für verschiedene Geräte
    - z. B. Smartphone vs. Desktop
- Unabhängigkeit der Schichten → **flexiblere Weiterentwicklung**

9. Client-Server, 3-Schichten-Architektur, Model-View-Controller

9.4 Client-Server-Architektur

- Trennung zwischen:
  - **Client**: fordert Dienste an (z. B. Browser)
  - **Server**: bietet Dienste an (z. B. Google-Suche)
- Kommunikation über Netzwerke (z. B. Internet)
- Komponenten meist auf **unterschiedlichen Geräten**

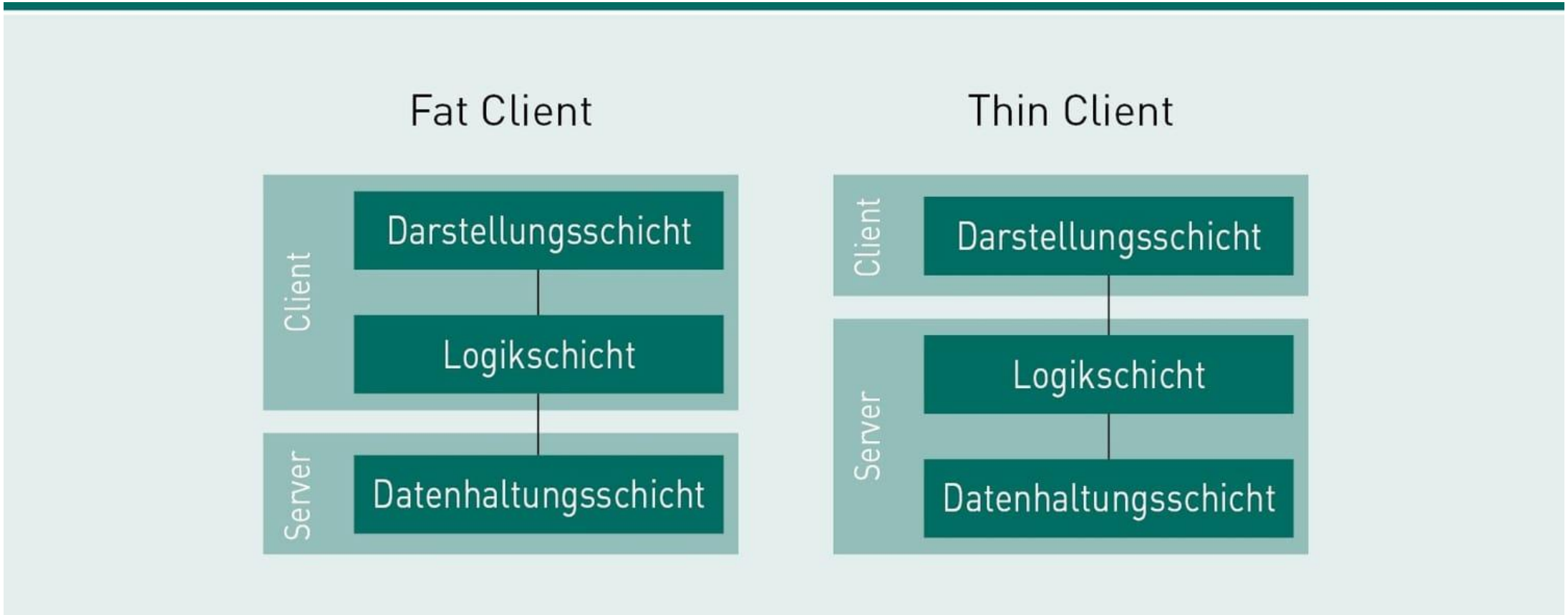
— **Vorteile:**

- Verteilung über mehrere Geräte
- Zentrale Wartung am Server genügt

— **Nachteile:**

- **Abhängigkeit vom Server**
- Serverausfall → Ausfall der Funktionalität
- Lösung: Cloud-Technologien, Virtualisierung (z. B. IaaS)

9. Client-Server, 3-Schichten-Architektur, Model-View-Controller



9. Client-Server, 3-Schichten-Architektur, Model-View-Controller

9.5 Kombination von Architekturen: Thin vs. Fat Client

- Trennung der Schichten auf Client & Server möglich
- Je nach Lastverteilung spricht man von:

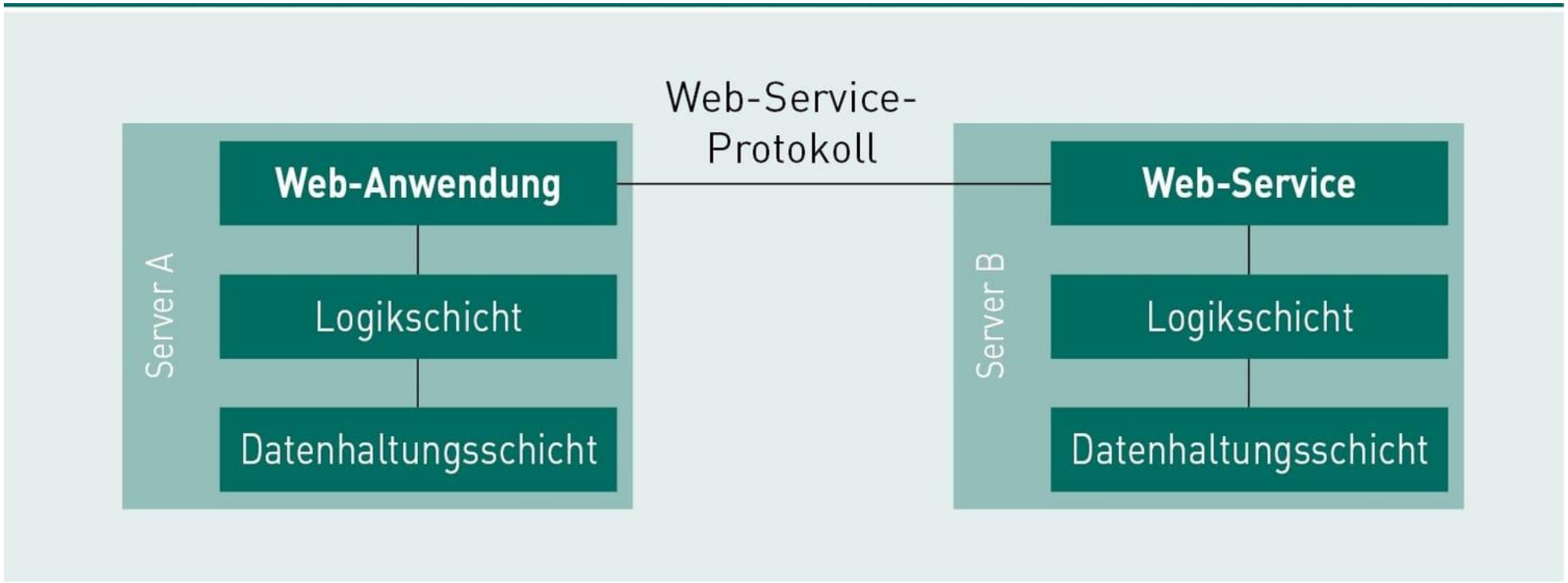
— **Thin Client:**

- Nur Darstellungsschicht im Client
- Logik & Datenhaltung auf dem Server
- Vorteil: geringe Hardwareanforderungen beim Client

— **Fat Client:**

- Darstellung + Teile der Logik lokal beim Client
- Nur Datenhaltung auf dem Server
- Vorteil: bessere Performance, offline-fähig

9. Client-Server, 3-Schichten-Architektur, Model-View-Controller
Exkurs



9. Client-Server, 3-Schichten-Architektur, Model-View-Controller

9.6 Model-View-Controller (MVC) Architektur

- Entwickelt bereits **1979**
- Ähnelt der 3-Schichten-Architektur
- Besonders geeignet für **Web-Anwendungen**

Grundprinzip: Trennung von Zuständigkeiten

MVC-Komponenten:

— **1. View (Ansicht)**

- Präsentiert die Daten in der Benutzeroberfläche
- Interagiert direkt mit dem Nutzer
- Holt Informationen ggf. direkt vom Modell

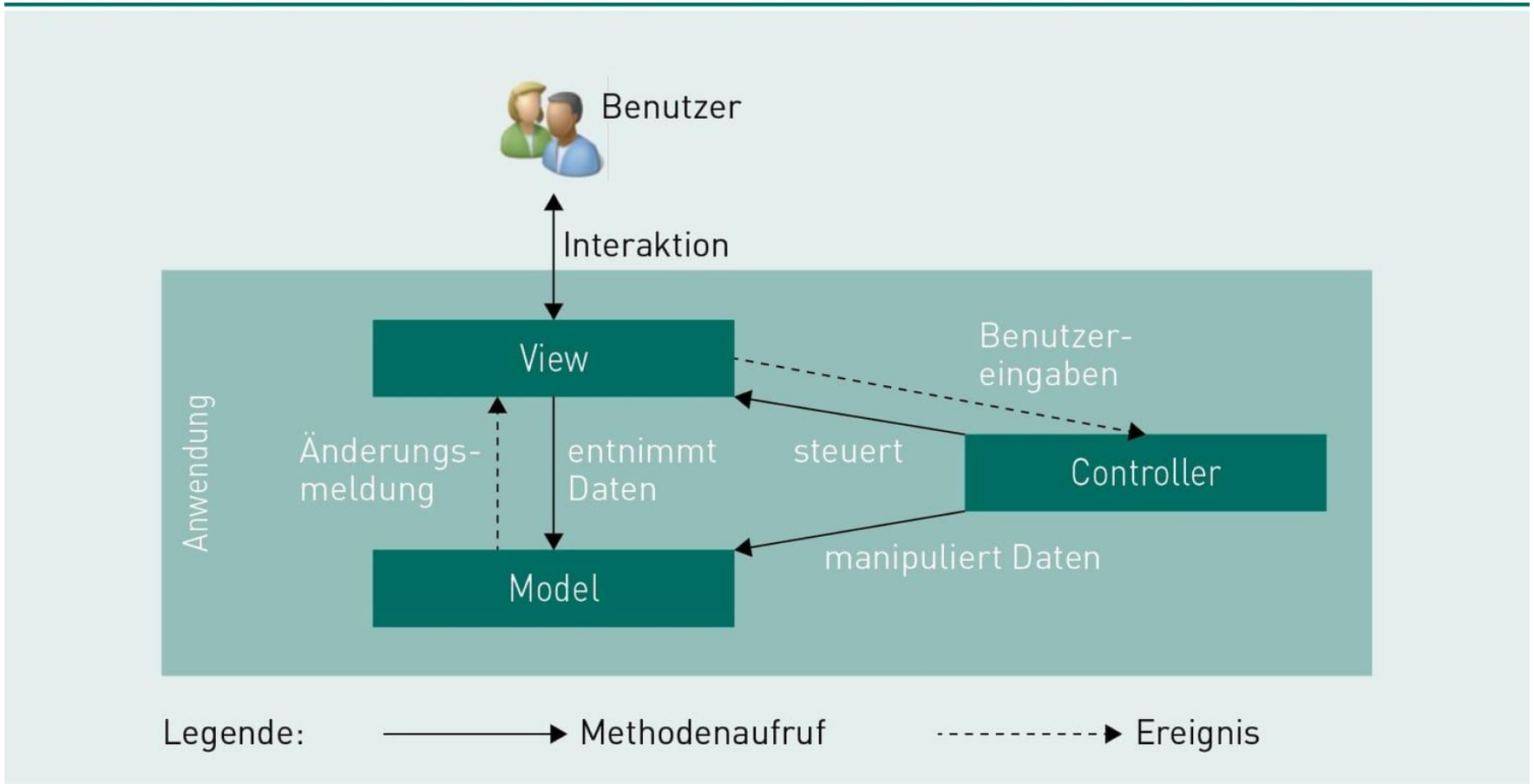
— **2. Model (Modell)**

- Enthält **Daten** & **fachliche Logik**
- Verantwortlich für Geschäftsobjekte (z. B. Artikel, Kunde)
- Kann View bei Datenänderung benachrichtigen

— **3. Controller**

- Vermittelt zwischen View und Model
- Verarbeitet **Benutzereingaben**
- Führt Validierungen & Aktionen aus
- Steuert Navigation & Feedback an Nutzer

9. Client-Server, 3-Schichten-Architektur, Model-View-Controller



9. Client-Server, 3-Schichten-Architektur, Model-View-Controller

9.6 Model-View-Controller (MVC) Architektur

Interaktionsfluss im MVC-Modell

1. Benutzer interagiert mit der **View**
2. **Controller** verarbeitet Eingabe
3. **Model** wird aktualisiert
4. **View** zeigt aktualisierte Daten an

Vergleich zu 3-Schichten-Architektur	
3-Schichten	**MVC**
Darstellung	View
Logik + Datenhaltung	Model + Controller
Klare Trennung	Objektorientierte Integration

9. Client-Server, 3-Schichten-Architektur, Model-View-Controller

9.6 Model-View-Controller (MVC) Architektur

9.6.1 Vorteile der MVC-Architektur

- Besser geeignet für **objektorientiertes Design**
- Fachkonzepte (z. B. Artikel, Kunde) als Klassen mit Daten & Methoden
- Keine Trennung zwischen Daten und Logik im Modell
- Erleichtert Wartung & Erweiterung
- **Framework-Support** (z. B. Spring MVC, Angular, React mit Redux etc.)

Fazit: Bedeutung von MVC für Web-Anwendungen

- MVC spielt eine zentrale Rolle in der **Modellierung & Umsetzung** von Web-Apps
- Wird im Kurs weiterhin eine wichtige **Grundstruktur** zur Erklärung sein
- Unterstützt saubere Architektur, **hohe Wiederverwendbarkeit** & klare Rollenverteilung

9. Client-Server, 3-Schichten-Architektur, Model-View-Controller

9.6 Model-View-Controller (MVC) Architektur

9.6.2 MVC im modernen Web-Framework-Kontext

- Viele moderne Frameworks basieren auf dem MVC-Prinzip
  - z. B. Java Spring MVC, Ruby on Rails, ASP.NET MVC
- Softwareentwickler integrieren primär das **Model (Fachkonzept)**
- View und Controller oft durch Framework vorgegeben
- Ergebnis: **schnelle Entwicklung**, **modularer Code**, **Wartbarkeit**