AI-B.41: Verteilte Systeme

- Lecture Notes [SL] -

II. Systemmodelle & Architekturstile

C. Schmidt | SG AI | FB 4 | HTW Berlin

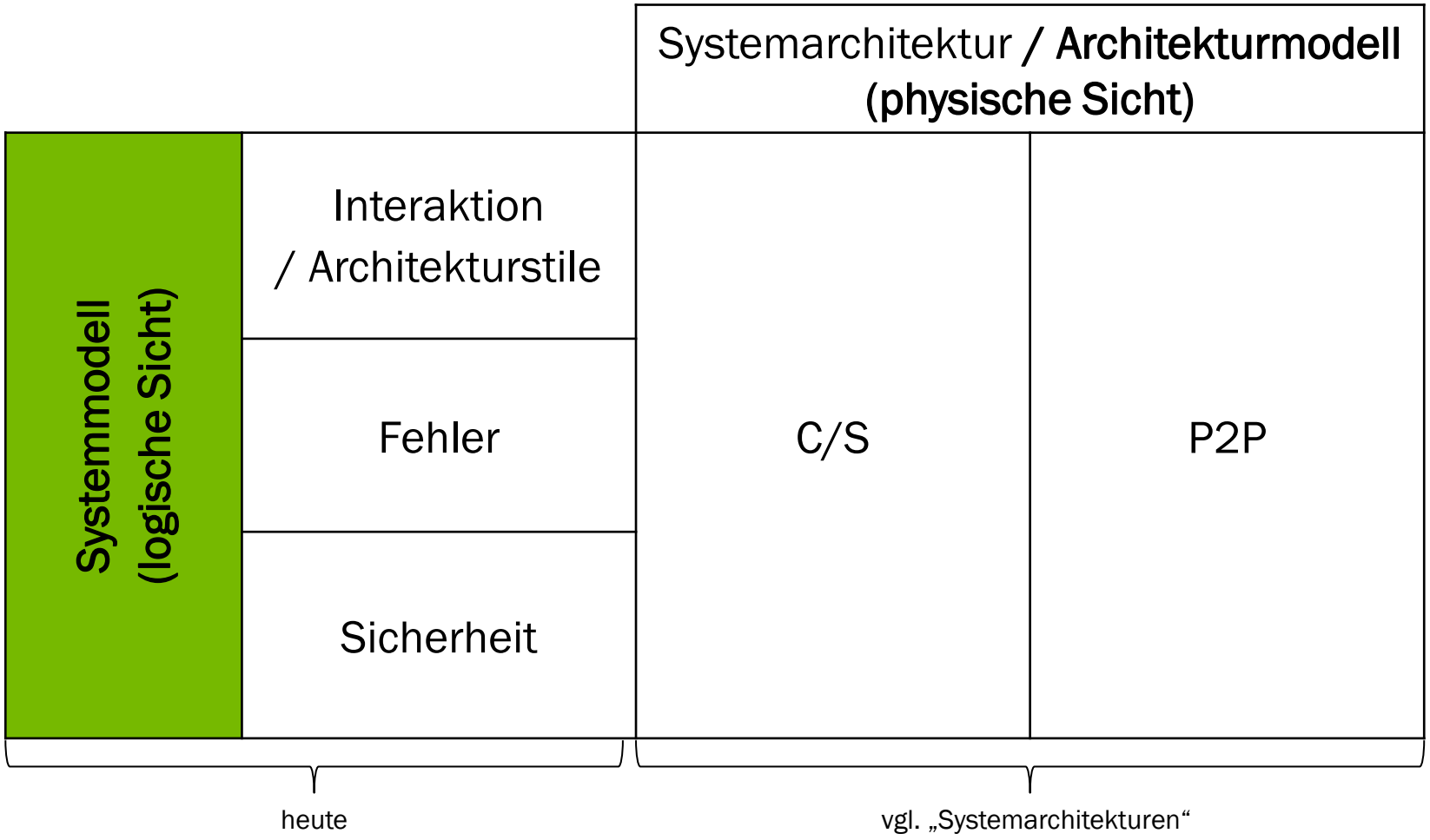
Stand: WiSe 18/19
Urheberin: Prof. Dr. Christin Schmidt

Lernziele

Nach dieser Lehrveranstaltung kennen Studierende idealerweise:

- Grundlegende Aspekte und Perspektiven der logischen Architektur verteilter Systeme (Systemmodell):
  - » Interaktion / Architekturstile:
    - Konzepte zur logischen Anordnung von Softwarekomponenten eines verteilten Systems
    - Unterscheidung und Bewertung der Konzepte in Kategorien- und Varianten (Granularität, Kopplung, Isolation, Schichtung, Objektfokus, Datenfokus, Ereignisfokus)
  - » Fehler: Klassen und Arten
  - » Sicherheit (vgl. hierzu auch Lecture Notes „Sicherheit“)
- Probleme und Maßnahmen zur Risikominimierung in den Dimensionen Interaktion, Fehler und Sicherheit
- Die Notwendigkeit zur Planung / Berücksichtigung von Einflussfaktoren in den jeweiligen Dimensionen

Modellperspektiven verteilter Systeme



Systemmodell [I]

- Vereinfachung/Abstraktion von Entwurfsaspekten verteilter Systeme in den Dimensionen:
  - Interaktion / Architekturstile
  - Fehler
  - Sicherheit

- Konsistente Beschreibung/Definition eines Entwurfsmodells für das VS (-> deskriptiv)
  - Funktional:
    - Funktionen der Komponenten eines VS
    - Kommunikation/Kommunikationsmuster der Komponenten eines VS
    - Abbildung auf das zu Grunde liegende Netzwerk des VS
  - Nicht-funktional (vgl. Parameter in Lecture Notes I)

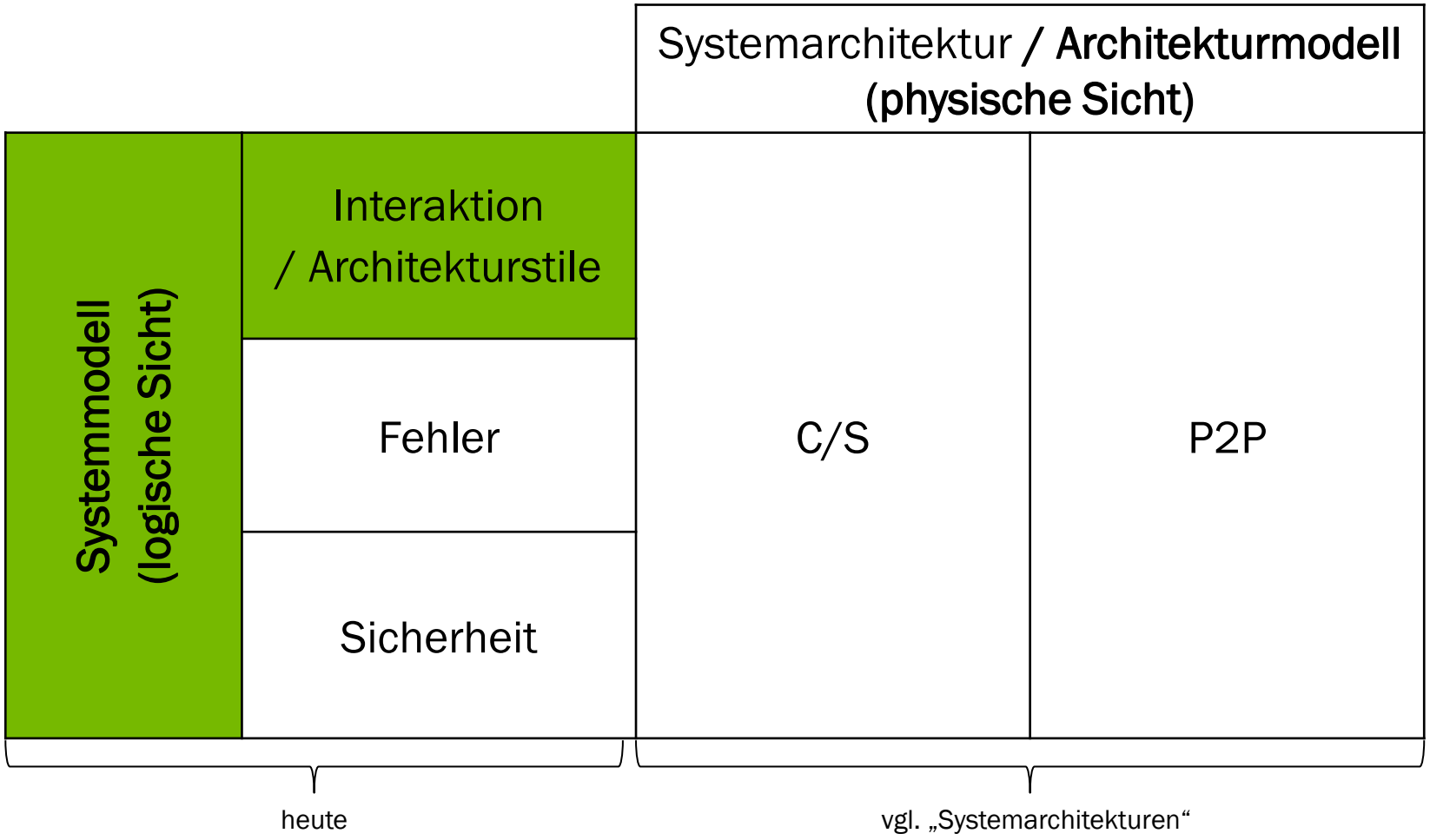
- Ziel: Korrektheit, Zuverlässigkeit, Sicherheit des VS
  - Abbildung aktueller Anforderungen
  - Antizipation zukünftiger Anforderungen

Systemmodell [II]

- Ziel: Beantwortung der folgenden Fragen für den Entwurf
  - » Was sind die wichtigsten Einheiten im System?
  - » Wie arbeiten Sie zusammen?
  - » Welche Eigenschaften beeinflussen ihr eigenes Verhalten oder das Verhalten des Zusammenspiels aller Einheiten?

- Aufgaben:
  - » Aufzeigung aller relevanter Voraussetzungen zu den zu modellierenden Systemen
  - » Aufzeigung von Verallgemeinerungen des
    - Möglichen
    - Unmöglichen

Modellperspektiven verteilter Systeme



Architekturstil [I]

vgl. Tanenbaum & van Steen [2008:53]

- Entspricht der logischen Anordnung der Softwarekomponenten eines verteilten Systems (syn.: Softwarearchitektur)

- Nicht zu verwechseln mit der Systemarchitektur (vgl. Lecture Notes zu Systemarchitektur)
  - » physische Realisierung der Verteilung,
  - » Instanz einer Softwarearchitektur

- Komponente: modulare Einheit mit wohldefinierten erforderlichen und bereitgestellten Schnittstellen (engl.: interfaces), die in ihrer Umgebung ersetzbar ist.
  - ✓ Konnektor: Mechanismus zur Vermittlung der Kommunikation, Koordination oder Kooperation zwischen Komponenten

- ✓ Komponenten (z.B.: Clients, Server, unterstützende Prozesse) in einem VS erbringen zusammen Leistungen, die sie für sich genommen nicht erbringen könnten

- By the way: Die eigentliche Implementierung von Komponenten erlernen Sie in anderen Modulen in der Angewandten Informatik (z.B. Prog III, KBE, ...)

Architekturstil [II]

vgl. Tanenbaum & van Steen [2008:53] & Anthony [2016:298ff.]

- Ein Stil wird formuliert nach
  - den Komponenten,
  - nach der Art, wie die Komponenten miteinander verbunden sind,
  - nach den Daten, die zwischen den Komponenten ausgetauscht werden und
  - wie diese Elemente gemeinsam zu einem System konfiguriert werden.

- Typische Fragestellungen zur Findung eines konkreten Stils:
  - Welche Ressourcen sollen verteilt genutzt werden?
  - Welche Dienste/Prozesse sollen diese Ressourcen verwalten? (Server-Prozesse, Client-Prozesse, Gleichrangige Prozesse?)
  - Welche Dienste/Prozesse sollen wo platziert werden?
  - Welche Komponenten werden wie konnektiert?

✓ „Separation of concerns“: logische Grenze zwischen Komponenten sollte zunächst durch Funktionalitäten, nicht durch abstraktere Kriterien (z.B. gleiche Größe der Komponenten) gezogen werden

✓ Welche verschiedenen Architekturstile gibt es und wie kann man diese unterscheiden? Vgl. ff.

Architekturstile: Kategorien [I]

vgl. Tanenbaum & van Steen [2008: 53]

Unterscheidung nach Granularität
[vgl. hierzu ergänzend Schill & Springer [2012: 14ff.]:

- Grob: wenige Prozesse mit langen Ausführungszeiten / geringes Auftreten von Datensynchronisation
- Fein: viele Prozesse mit kurzen Ausführungszeiten / hohes Auftreten von Datensynchronisation

Unterscheidung nach Kopplung

- » Stärke der Kopplung:
  - Lose: geringe Dichte im Graphen aller dem VS zugehörigen Knoten
  - Stark / eng: hohe Dichte
- » Logische Verbindung der Kopplung [Anthony 2016:297f.; vgl. ff.]
  - Direkte Kopplung
  - Indirekte Kopplung
  - Isolierte Kopplung
- ✓ Ergibt zahlreiche Variationen

Architekturstile: Unterscheidung nach Kopplung [II]

vgl. Anthony [2016:298ff.]

Direkte Kopplung: Kommunikation zwischen zwei Prozessen / Komponenten (vgl. Interprozesskommunikation)

- synchron / asynchron
- Konstanz / Permanenz der physischen Verbindung

Indirekte Kopplung: Kommunikation zwischen Komponenten erfolgt durch einen Intermediär / Vermittler („Middleware“).

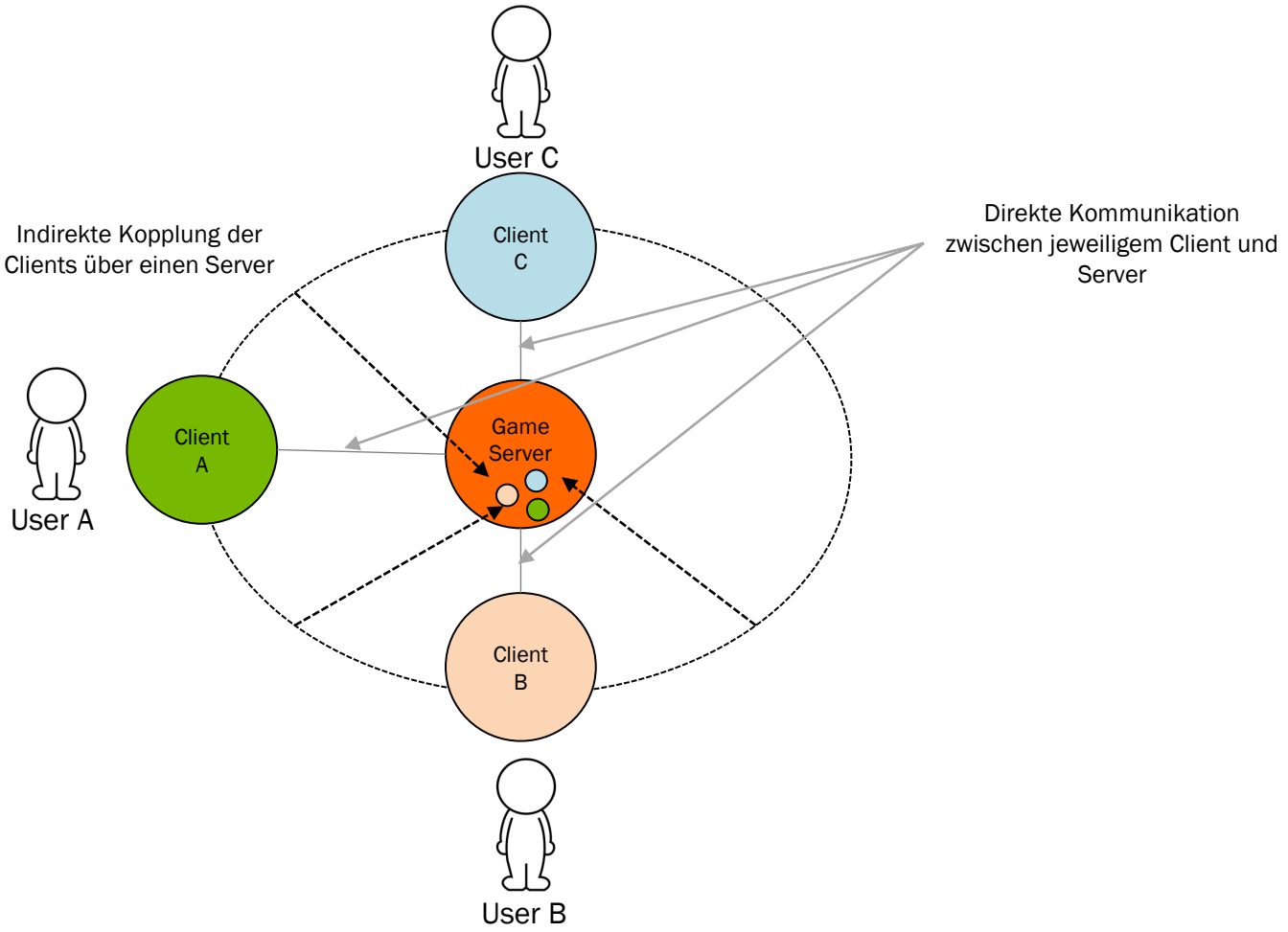
- Nicht isoliert: Clients sind nicht direkt miteinander verbunden (Kommunikation erfolgt über einen Server), wissen aber über den Vermittler voneinander
- Isoliert: Clients sind nicht direkt miteinander verbunden (Kommunikation erfolgt über einen Server) und wissen nicht voneinander

✓ Beispiele: vgl. ff.

Beispiel: Indirekte Kopplung (nicht isoliert) [I]

vgl. Anthony [2016:299f.]

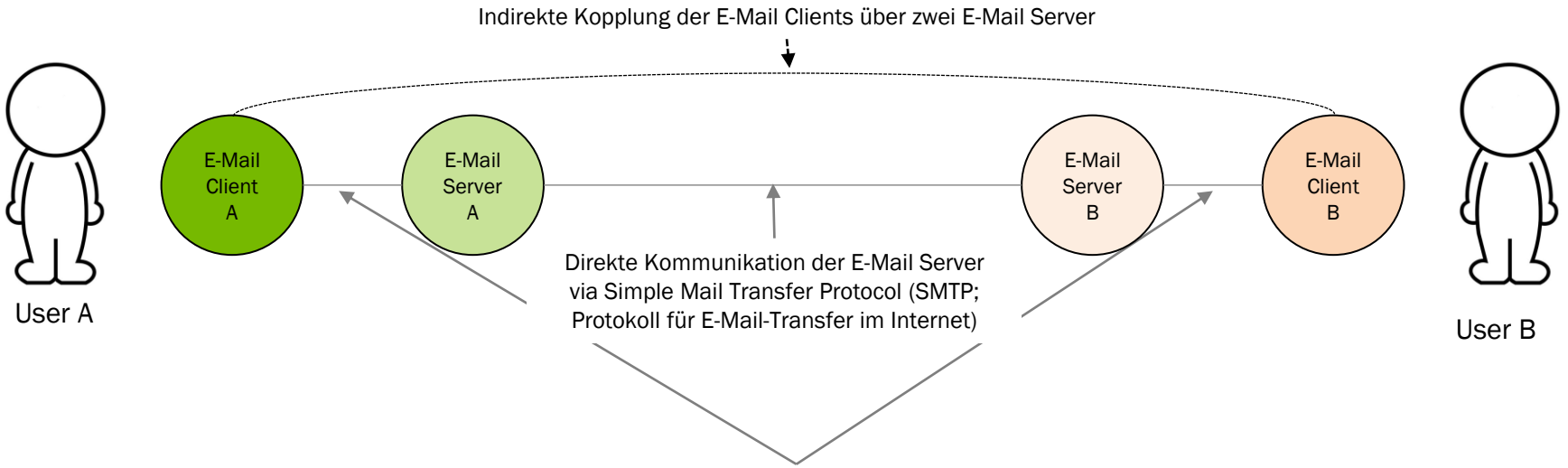
Online Multiplayer Game (logische Verbindung: die User sind Gegner in einem Spiel)



Beispiel: Indirekte Kopplung (isoliert) [II]

vgl. Anthony [2016:299f.]

E-Mail (logische Verbindung: E-Mail Konversation)

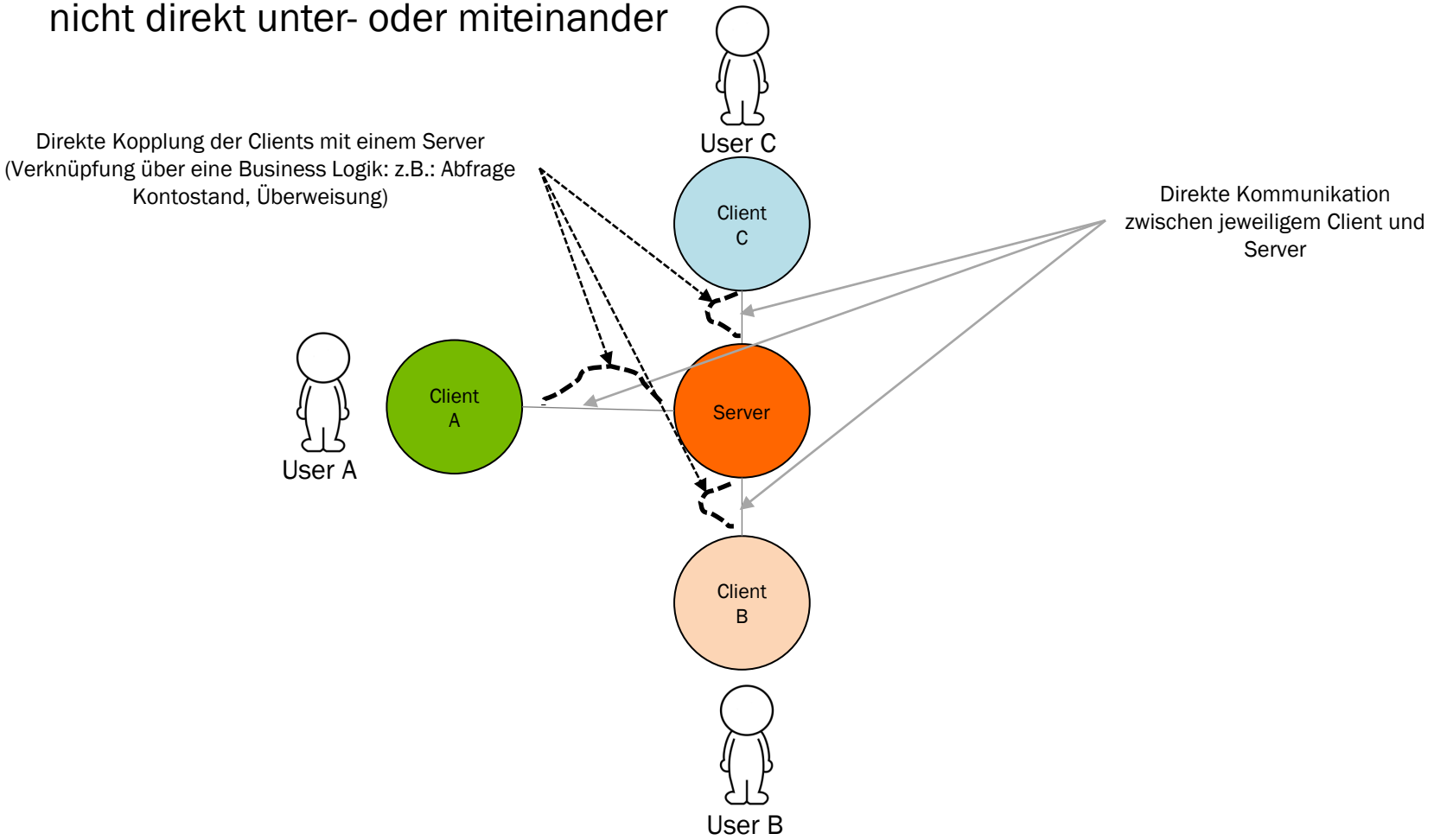


Bildquelle: eigene Darstellung i. A. an Anthony [2016:299] unter Verwendung von Männchen; online: https://www.schulbilder.org/malvorlage-maennchen-leeres-gesicht-dm25691.jpg [letzter Zugriff: 4 XII 17]:

Beispiel: Direkte isolierte Kopplung [II]

vgl. Anthony [2016:300f.]

Online-Banking-System: Komponenten partizipieren an einem System (und sind mit diesem gekoppelt, d.h. es gibt eine logische Verbindung), aber sie kommunizieren nicht direkt unter- oder miteinander



Architekturstile: Kategorien [II]

vgl. Tanenbaum & van Steen [2008: 53]

Unterscheidung nach Konfiguration der Komponenten und Konnektoren:

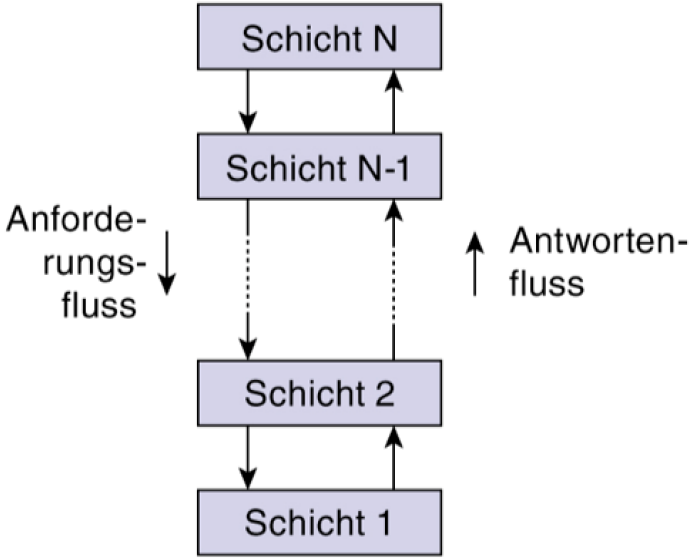
- SWE: erlaubt viele Konfigurationen
- Teilmenge einiger für VS relevanter Konfigurationen:
  - » Geschichtete Architekturen („tier architectures“)
  - » Objektbasierte Architekturen
  - » Datenzentrierte Architekturen
  - » Ereignisbasierte Architekturen

✓ Übrigens: einige Ansätze werden vermischt / abgewandelt und gehen in zahlreichen anderen Architekturstilen und Implementierungen mit eigenen Nomenklaturen und Standards auf, z.B. Service-Orientierte Architekturen (SOA)

Architekturstile: Geschichtete Architekturen [I]

vgl. Tanenbaum & van Steen [2008:53ff.]

- Komponenten werden schichtweise angeordnet
  - » Komponente auf Ni darf Komponente(n) auf Ni-1 aufrufen, aber nicht umgekehrt
  - » Anforderungen durchlaufen die Hierarchie nach unten, Antworten werden nach oben gereicht



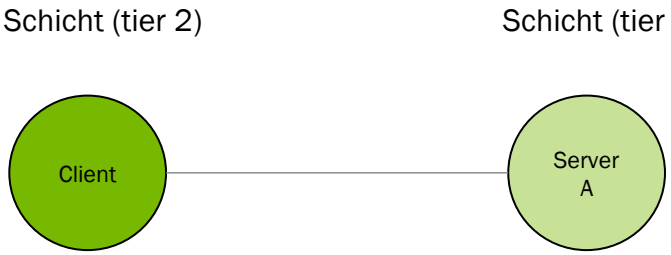
Quelle: Tanenbaum & van Steen [2008:54]

Architekturstile: Geschichtete Architekturen [II]

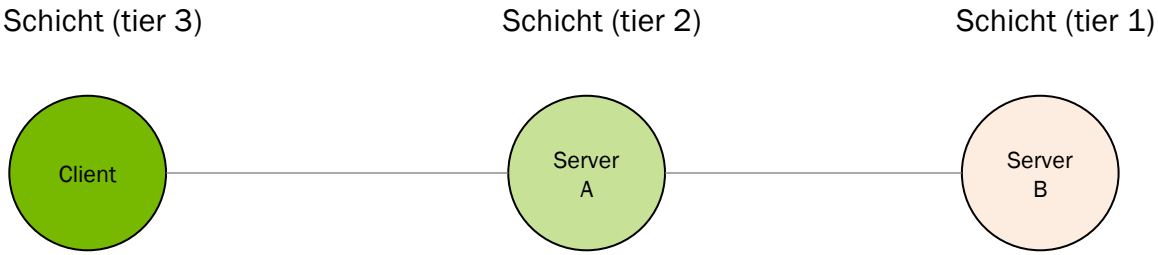
vgl. Tanenbaum & van Steen [2008:53ff.]

Unterscheidung nach Anzahl der Schichten / Stufen („n-tiers“):

- Zweistufig: nur EIN Client und EIN Server



- Dreistufig (C*SS*; vgl. Abschnitt „Systemarchitekturen“)

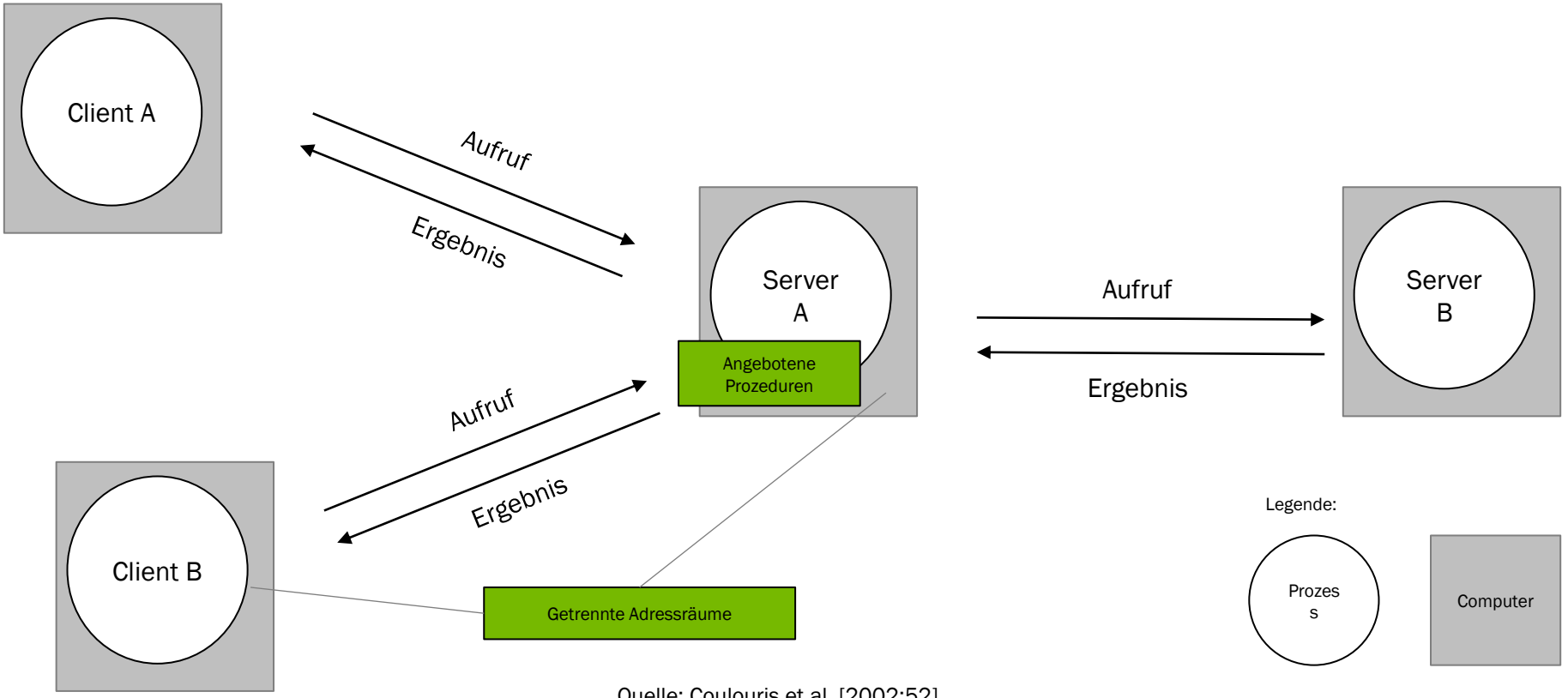


Architekturstile: Geschichtete Architekturen [III]

vgl. Tanenbaum & van Steen [2008:53ff.]

Beispiel: 3-tier-Architektur

- Client: Benutzerschnittstelle (ggf. Vorverarbeitungsfunktionen) / Präsentationsschicht
- Server I: Vorverarbeitung (Anwendungslogik/Verarbeitungsschicht)
- Server II: Datenverwaltung/Persistenzschicht



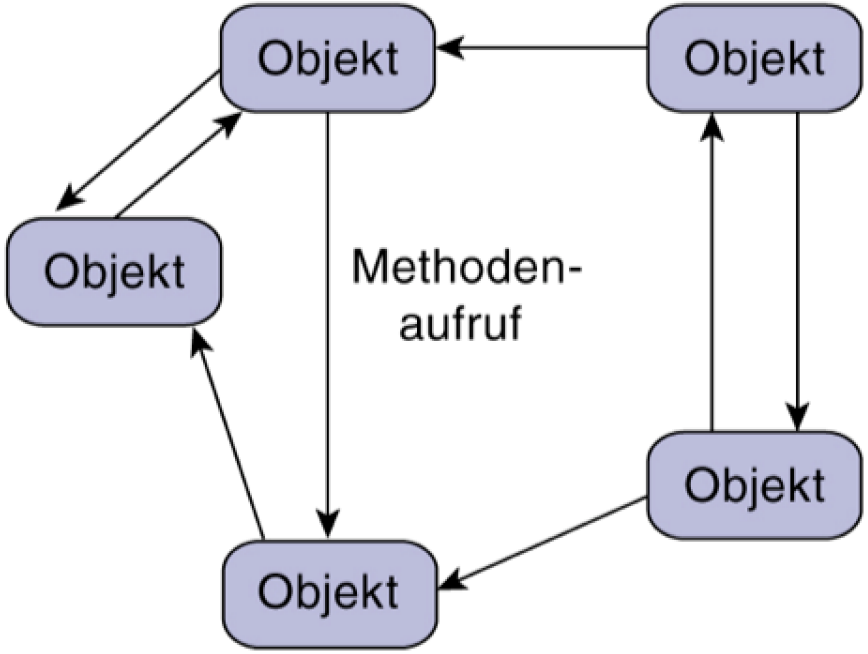
Quelle: Coulouris et al. [2002:52]

Architekturstile: Objektbasierte Architekturen

vgl. Tanenbaum & van Steen [2008:53ff.]

Unterscheidung nach beteiligten Objekten Objekte entsprechen Komponenten / Prozessen)

- ✓ sind ggf. auch geschichtet (vgl. geschichtete Architekturen)
- ✓ Lose Anordnung
- ✓ Mechanismus zur Interaktion: Entfernter Prozedur-/Methodenaufruf (vgl. „Interprozesskommunikation“)

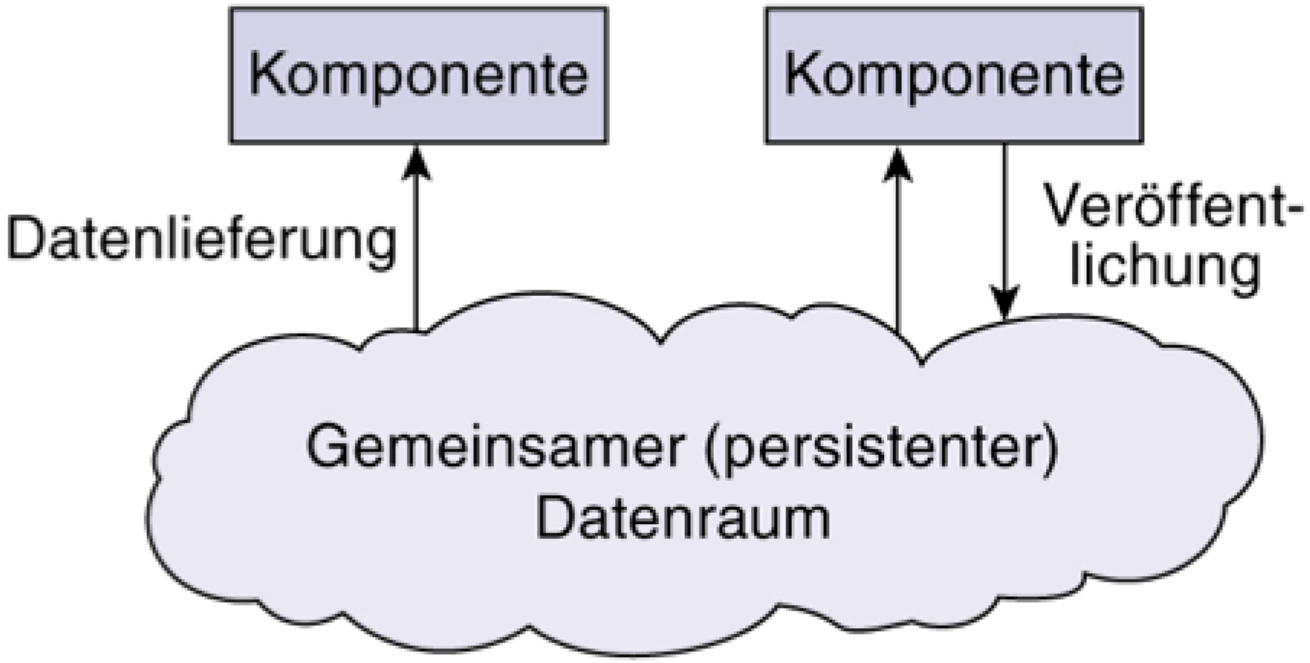


Quelle: Tanenbaum & van Steen [2008:54]

Architekturstile: Gemeinsamer Datenraum [I]

vgl. Tanenbaum & van Steen [2008:53ff.]

Prozesse kommunizieren über ein gemeinsames (aktives oder passives) Repository / Datendienste

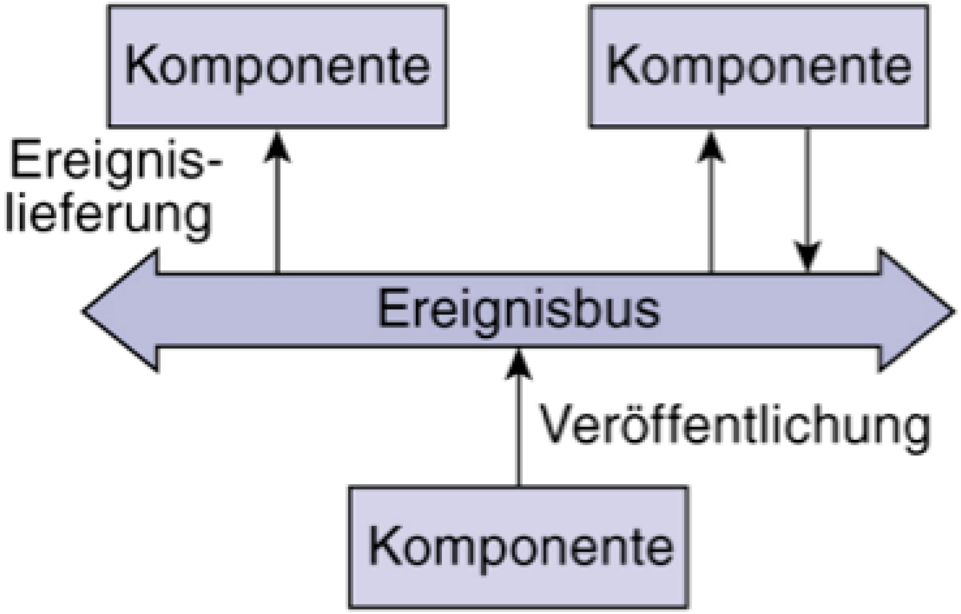


Quelle: Tanenbaum & van Steen [2008:54]

Architekturstile: Ereignisbasierte Architekturen [I]

vgl. Tanenbaum & van Steen [2008:53ff.]

Prozesse kommunizieren über die Weitergabe von Ereignissen



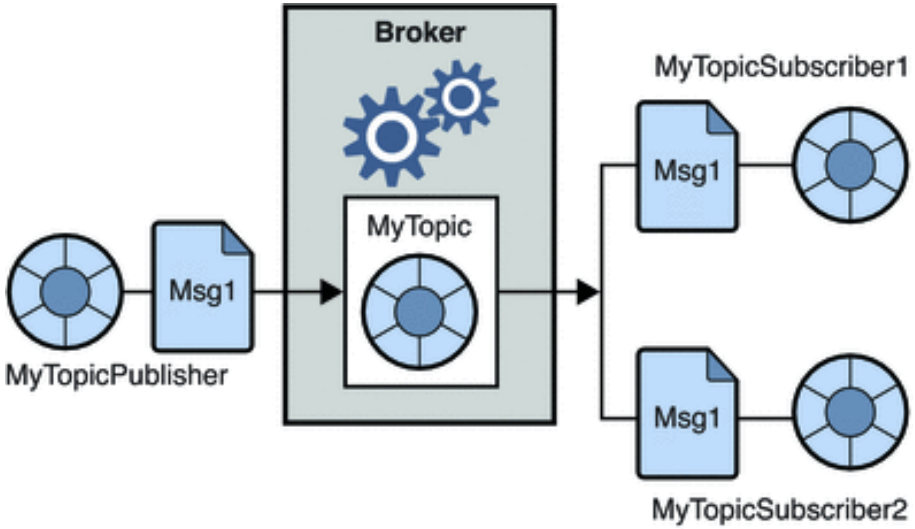
Quelle: Tanenbaum & van Steen [2008:54]

Architekturstile: Ereignisbasierte Architekturen [II]

vgl. Tanenbaum & van Steen [2008:53ff.]

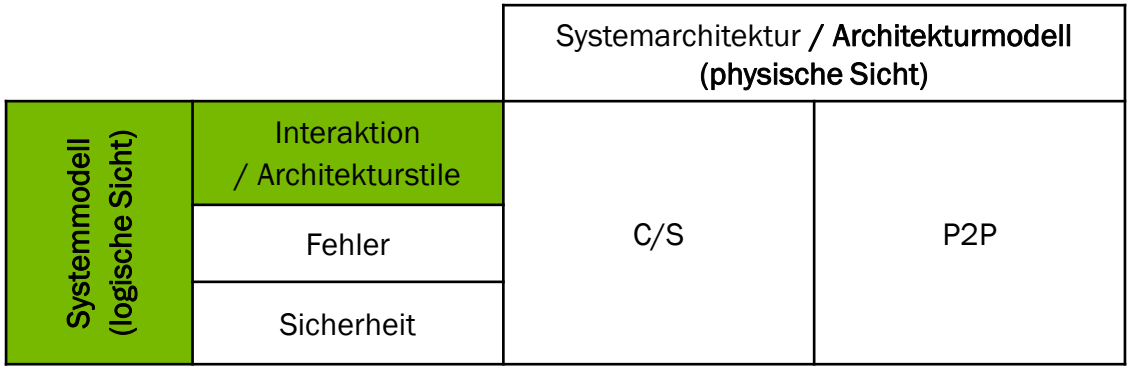
Beispiel: Pattern „Publish/Subscribe“ (pub/sub)

- Sender (publisher) veröffentlichen Ereignisse zum Abruf bei einem Broker
- n Empfänger (subscriber) können Ereignisse zu verschiedenen Themen bei einem Broker abonniert haben
- Publisher kennt n nicht,
- Vorteil: Prozesse müssen nicht direkt aufeinander verweisen (räumliche / referenzielle Entkopplung)
- Prozesse müssen nicht beide gleichzeitig aktiv sein (zeitliche Entkopplung)



Quelle: https://docs.oracle.com/cd/E19879-01/821-0028/images/to_simpleTopic.gif [letzter Zugriff: 8 XII 17]

Limitationen: Interaktion / Architekturstile [I]



Ausgangspunkt: Die gesamte Aktivität eines verteilten Systems wird über interagierende Prozesse realisiert

- Kommunikation sollte zwischen den Prozessen entstehen (Informationsfluss)
- Koordination zwischen den Prozessen muss etabliert werden (z.B.: Synchronisierung, Festlegung der Reihenfolge von Aktivitäten)

Aber:

- Koordination in VS ist limitiert -> Notwendigkeit der Berücksichtigung von Verzögerungen, Zeitkonzepten (reale vs. logische Sicht, vgl. „Zeit“)
- Es gibt schwer abzuschätzende Faktoren (vgl. ff.):
  - Kommunikationsleistung
  - Timing bei der Nachrichtenübertragung zwischen den Prozessen

Limitationen: Interaktion / Architekturstile [II]

Kommunikationsleistung = Geschwindigkeit der Verarbeitung hinsichtlich:

- **Bandbreite:** Gesamtbetrag der Information, die innerhalb einer bestimmten Zeit über ein Computernetzwerk übertragen werden kann

- **Latenz:**
  - » Verzögerung zwischen Senden einer Nachricht durch Prozess A und deren Empfang (erste Bit-Zeichenkette) durch Prozess B
  - » Wird beeinflusst durch Netzwerk- und OS-Auslastung

- **Jitter:** Variation der Zeit, die bei der Auslieferung einer Folge von Nachrichten auftritt (Varianz der Latenz)

Limitationen: Interaktion / Architekturstile [III]

„Timing“

- Unterschiedliche Computer haben unterschiedliche lokale Uhren, die von lokalen Prozessen verwendet
  werden (Zeitstempel)
- Die Zeit lokaler Uhren weicht höchstwahrscheinlich in unterschiedlichen Uhrabweichgeschwindigkeiten[1] von einer Referenzuhr ab

Bildquelle: eigene Darstellung mit Dash array clock: https://publicdomainvectors.org/photos/StrokeDasharrayClock.png [letzter Zugriff: 7 XII 17]

[1] Uhrabweichgeschwindigkeit = relativer Betrag, um den sich eine Computeruhr von einer Referenzuhr unterscheidet

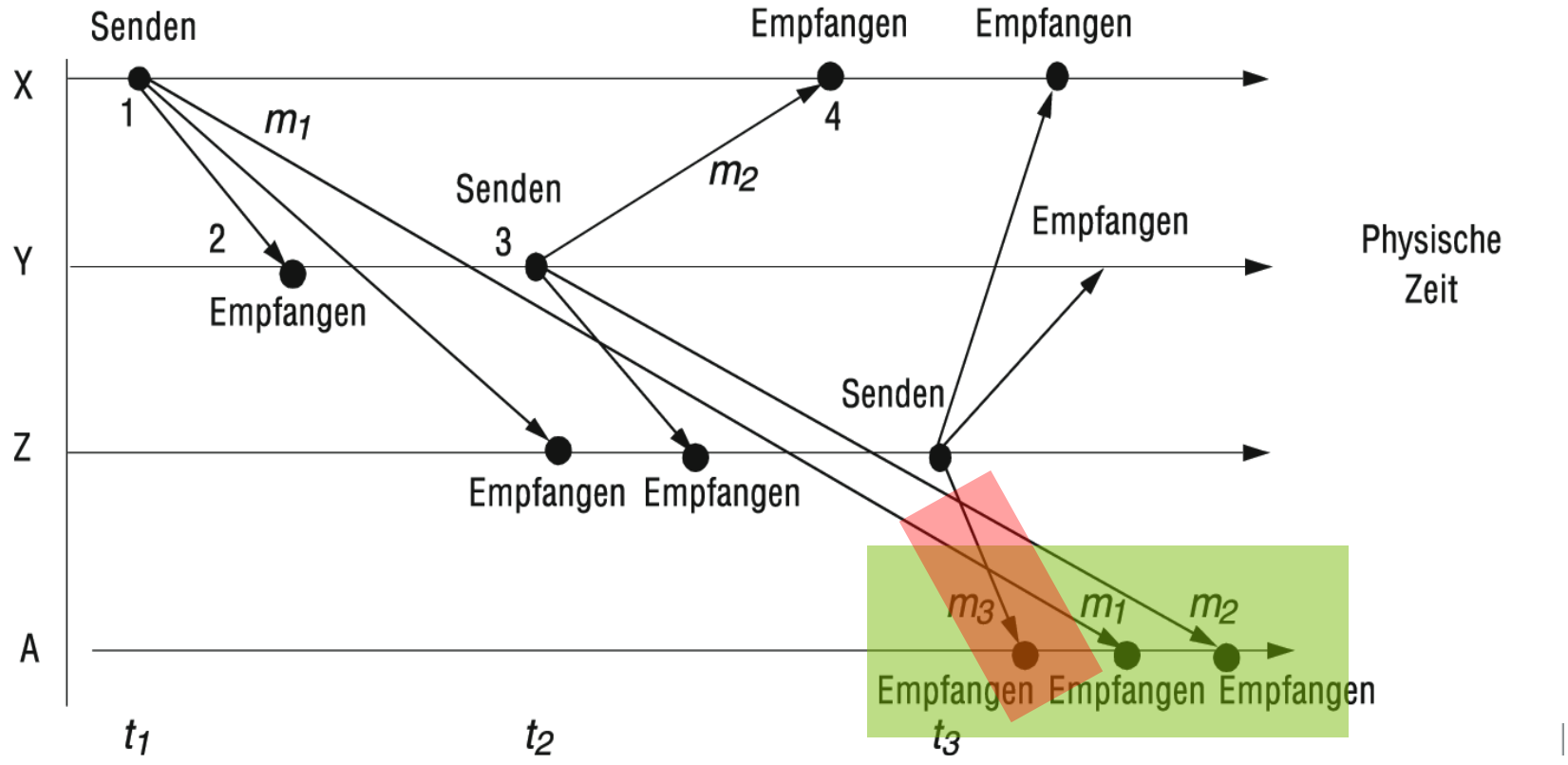
Architekturstile / Interaktion: Zeitgrenzen

Varianten:

- **Synchrones Verteiltes System:**
  - » mit (oberen und unteren) Zeitgrenzen für die Prozessausführungszeit, Nachrichtenauslieferung, Uhrabweichungsgeschwindigkeit
    - Schwierigkeit:
      - Festlegung realistischer Ober- und Unterwerte
      - Garantie
    - Notwendigkeit von Algorithmen bei Fehlerbehandlung (z.B. Timeouts)

- **Asynchrones Verteiltes System:** keine Begrenzung (per definitionem)

Beispiel (Ausblick): logische Zeit vs. „reale“ Zeit [I]



Quelle: Coulouris et al. [2002: 75]

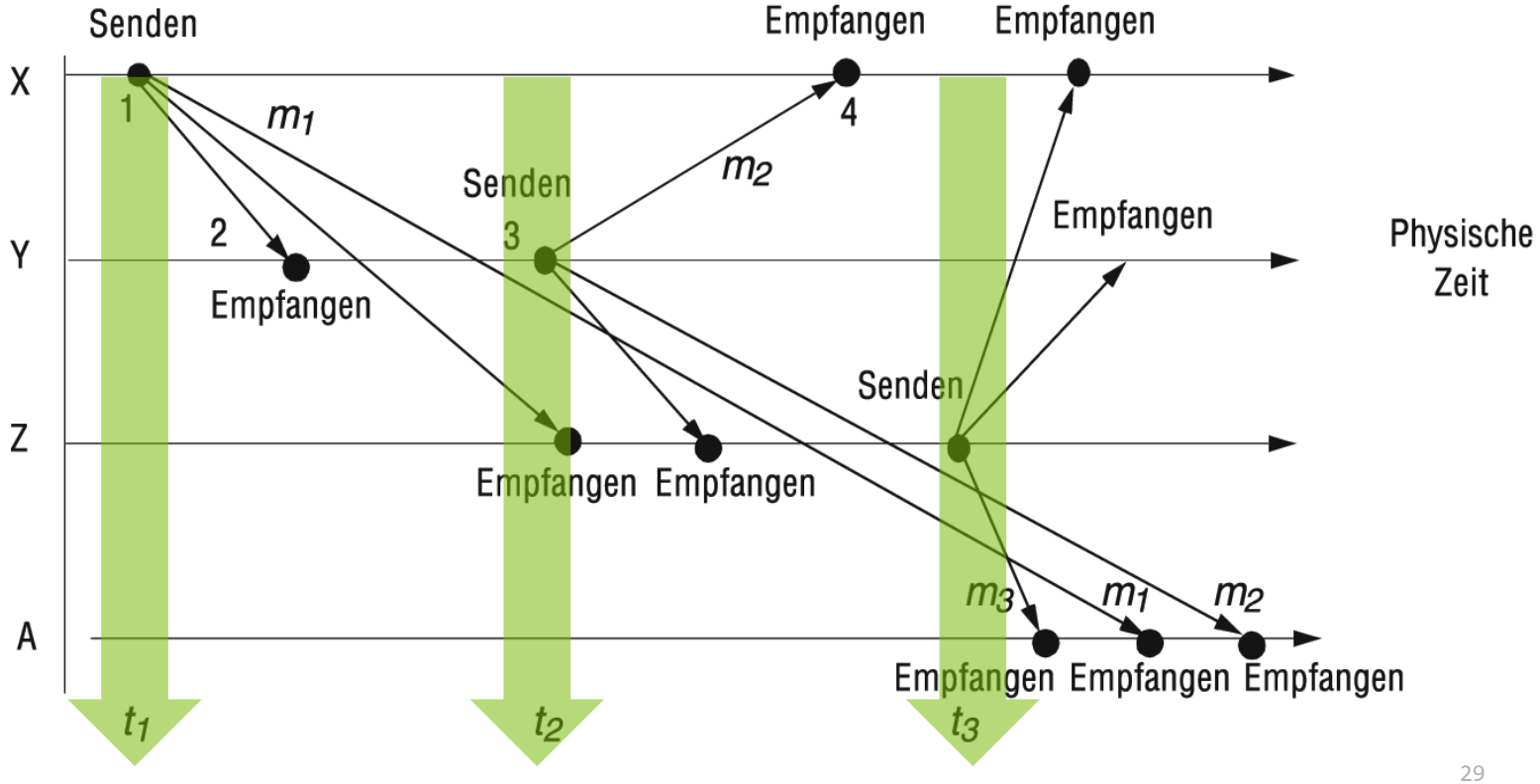
Beispiel (Ausblick): logische Zeit vs. „reale“ Zeit [II]

Probleme der Koordinierung von Ereignisabfolgen mit „realer Zeit“ am Beispiel von E-Mail Nachrichten

Posteingang (A)			
Eintrag	Von	Thema	
23	Y	Re: V-SYS	m_2
24	X	V-SYS	m_1
25	Z	Re: V-SYS	m_3

Beispiel (Ausblick): logische Zeit vs. „reale“ Zeit [III]

- Lösungsansatz: Sortierung nach „realer“, zeitlicher Abfolge mit Zeitstempeln in der Nachricht : $t_1 < t_2 < t_3$
- Problem: Hohe Wahrscheinlichkeit der Asynchronizität lokaler Uhren
- ✓ Alternative: Sortierung nach logischer Zeit/Ereignisabfolge

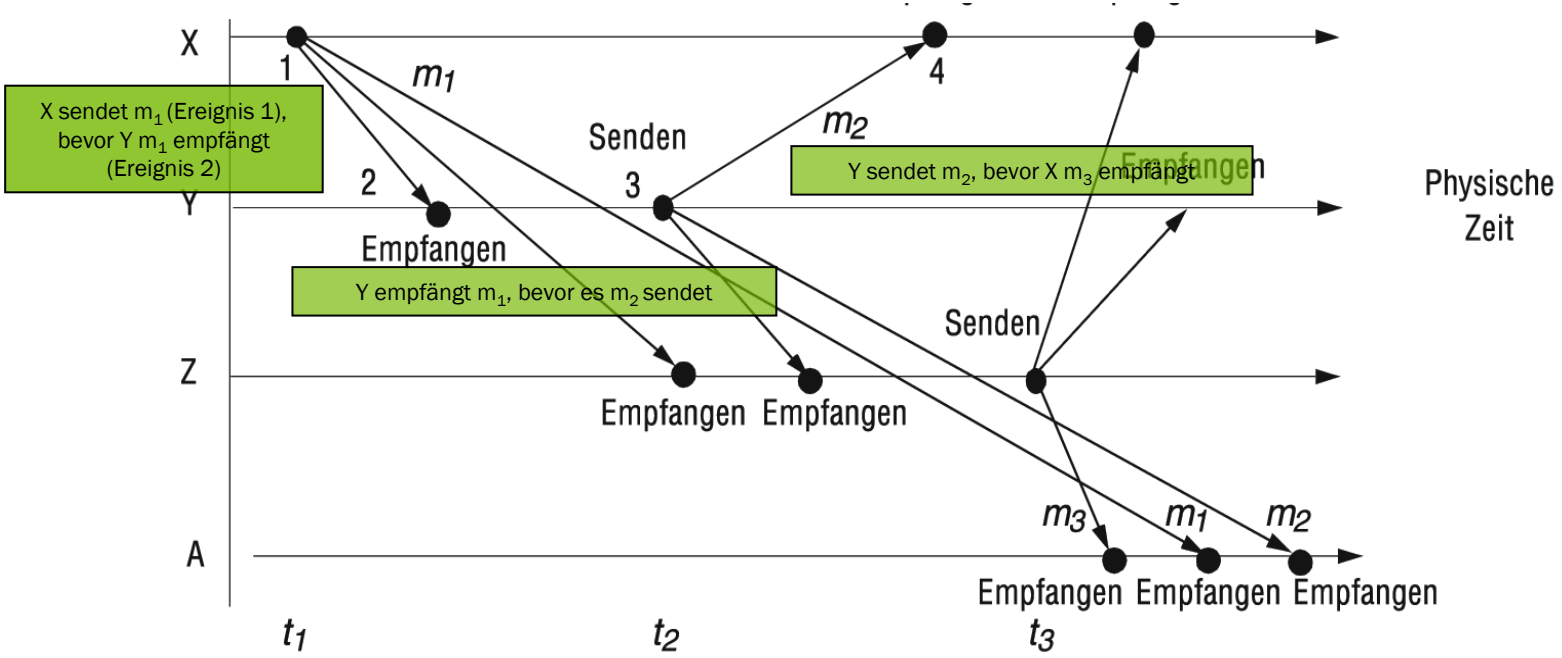


Quelle: Coulouris et al. [2002: 75]

Beispiel (Ausblick): logische Zeit vs. „reale“ Zeit [IV]

Logische Zeit:

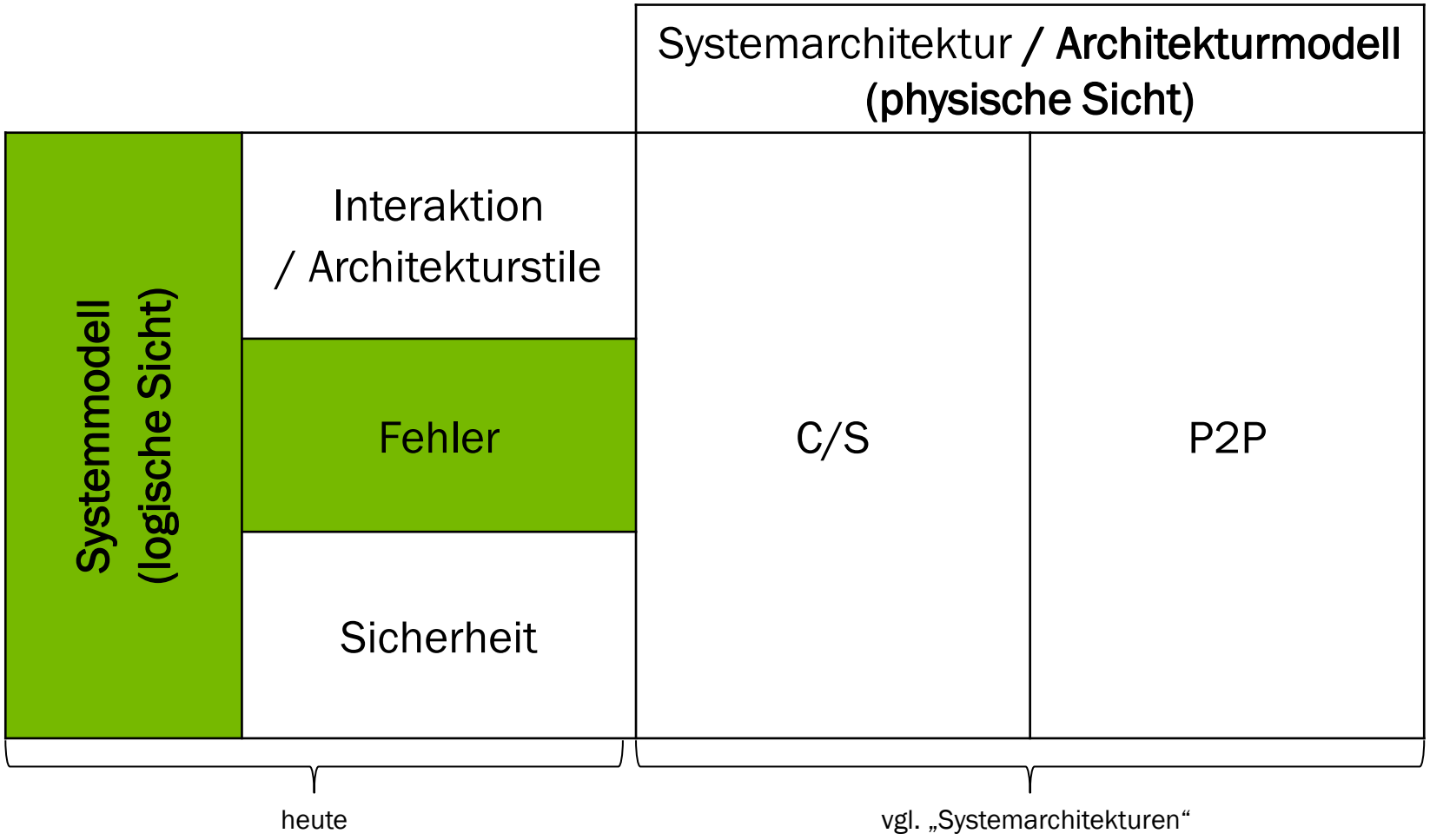
- ✓ Ableitung der Reihenfolge, in der Nachrichten angezeigt werden, ohne Nutzung lokaler Uhren (Bildung von Ereignispaaren)
- ✓ Jedem Ereignis wird gemäß seiner logischen Reihenfolge eine Nummer (1,2,...,n) zugewiesen



Quelle: Coulouris et al. [2002: 75]

*/

Modellperspektiven verteilter Systeme



Systemmodell: Fehler [I]

- Fehler sind Abweichungen von einem als korrekt oder wünschenswert definierten Verhalten
- Fehlerobjekte: Prozesse, Kommunikationskanäle
- Fehlerquellen:
  - » Ein Computer des VS
  - » Zu Grunde liegendes Netzwerk des VS
- Perspektive erfordert Erstellung eines Fehlermodells:
  - » Definition, Klassifikation von Fehlern
  - » Ziel: Antizipation möglicher Auswirkungen auf den Entwurf eines (möglichst) fehlertoleranten VS

Beispiel: Taxonomie nach Hadzilacos und Toueg [in: Coulouris et al. 2002: 75ff.]

Class / Type	Process	Link (Kommunikationskanal)
Ommission (Dienstverweigerung/Auslassung)		
Timing		

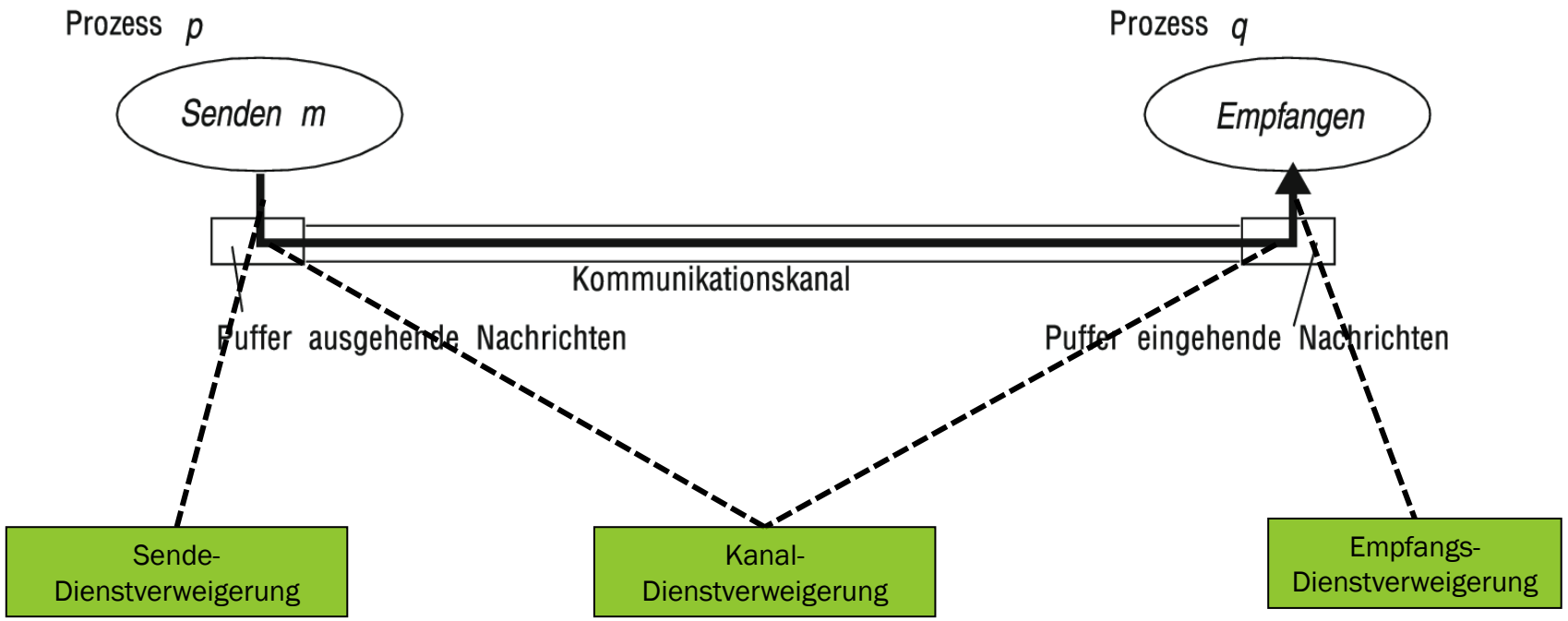
Fehlerarten [I]

- Dienstverweigerung/Auslassungsfehler: Prozess/Kommunikationskanal kann Aktionen nicht ausführen, die benötigt werden
  - » Prozessdienstverweigerung („Absturz“):
    - Abbruch des Prozesses
    - Keine weitere Ausführung von Programmschritten
  - » Kommunikationsdienstverweigerung (vgl. ff.)

- Timing Fehler

- Zufällige Fehler

Fehlerarten [II]
Kommunikationsdienstverweigerung



Quelle: Coulouris et al. [2002:76]

Fehlerarten [III]
Timing-Fehler

- ✓ Timing-Fehler können in synchronen VS spezifiziert werden

- ✓ Fehlerklassen (Timing):

Fehlerklasse	Auswirkung auf	Beschreibung
Uhr	Prozess	Die lokale Uhr des Prozesses überschreitet die Grenzen für ihre Abweichungsrate von der Echtzeit.
Leistung	Prozess	Der Prozess überschreitet die Grenzen des Intervalls zwischen zwei Verarbeitungsschritten.
Leistung	Kanal	Die Übertragung einer Nachricht dauert länger als der angegebene Grenzwert.

Quelle: Coulouris et al. [2002:79]

Fehlerarten [IV]
„gute“ vs. „böse“ Fehler

Fehlerhaftes Abbrechen	Prozess	Der Prozess wird abgebrochen und bleibt abgebrochen. Andere Prozesse können diesen Zustand erkennen.
Absturz	Prozess	Der Prozess wird abgebrochen und bleibt abgebrochen. Andere Prozesse können diesen Zustand möglicherweise nicht erkennen.
Dienstverweigerung/ Auslassung	Kanal	Eine in einen Ausgangsnachrichtenpuffer gestellte Nachricht kommt nie im Eingangsnachrichtenpuffer des anderen Teilnehmers an.
Sende-Dienstverweigerung	Prozess	Ein Prozess vervollständigt das *send*, aber die Nachricht wird nicht in seinen Ausgangsnachrichtenpuffer geschrieben.
Empfangs-Dienstverweigerung	Prozess	Eine Nachricht wird in den Eingangsnachrichtenpuffer eines Prozesses geschrieben, aber dieser Prozess ruft sie nicht ab.
Zufällig (Byzantinisch)	Prozess oder Kanal	Der Prozess/Kanal weist ein zufälliges Verhalten auf: er kann beliebige Nachrichten zu beliebigen Zeiten senden/ übertragen und Dienstverweigerungen begehen; ein Prozess kann beendet werden oder einen fehlerhaften Verarbeitungsschritt vornehmen.

„Gutartige“ Fehler

„Bösartige“ Fehler[1]

[1] vertiefend zu Algorithmen zur Erkennung / Umwandlung byzantinischer Fehler vgl. Fokkink [2013:121ff.]

Quelle: Coulouris et al. [2002:78]

Systemmodell: Fehler [II]

- ✓ Ausgangspunkt: Fehler / Fehlerarten begründen Herausforderungen für den Entwurf:
  - » Prozesse sollen „sauber“ abstürzen
    Logik: Entweder ein Prozess läuft oder er soll abbrechen -> Programmatische Abbruchsklauseln
  - » Abgebrochene Prozesse sollen von anderen Prozessen als abgestürzt identifiziert werden
    - Timeouts: Prozess wartet eine bestimmte Zeit auf Ereignis
    - Asynchrone VS: Keine Information über Ursache der fehlenden Reaktion (mögliche Ursachen: Absturz, langsame Verarbeitung, Nachrichtenübermittlung)

Systemmodell: Fehler [III]

- ✓ Trotz allem: es ist möglich, partiell fehlertolerante VS zu realisieren
- Technik: Fehlermaskierung
  - » Völlige Verbergung
  - » Umwandlung in einen akzeptablen Fehlertyp, z.B. zufälliger / byzantinischer Fehler wird in eine Dienstverweigerung mittels Prüfsummen zur Maskierung fehlerhafter Nachrichten umgewandelt
  - » Ersetzung eines abgestürzten Prozesses durch einen neuen Prozess, dessen Speicher sich aus den gespeicherten Informationen seines Vorgängers speist

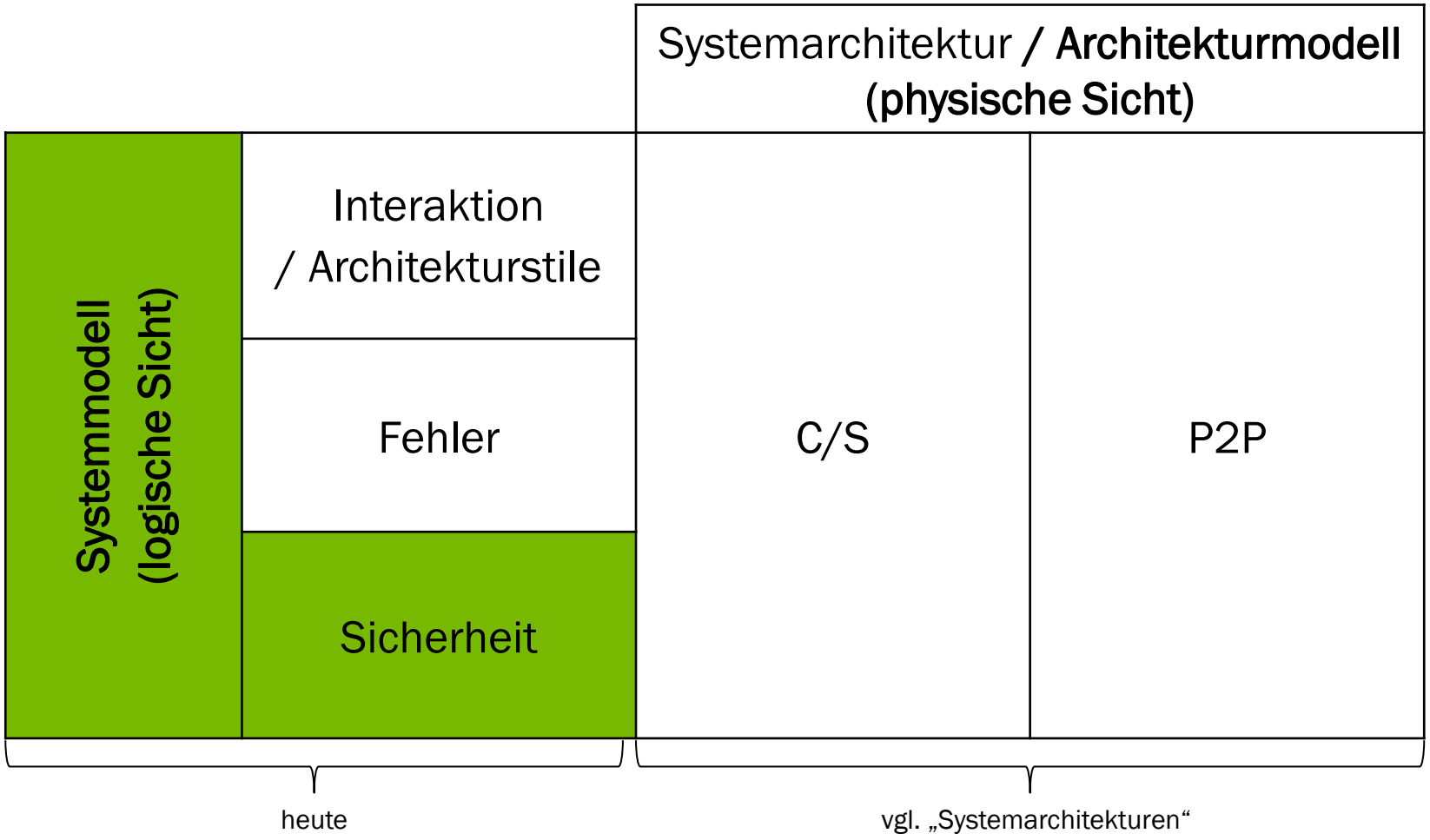
Systemmodell: Fehler [IV]
Zuverlässige 1:1-Kommunikation

Dimensionen der „Zuverlässigkeit“:

- Gültigkeit: Jede Nachricht in einem Ausgangsnachrichtenpuffer wird irgendwann in einen Eingangsnachrichtenpuffer ausgeliefert

- Integrität:
  - » Empfangene Nachricht ist identisch mit gesendeter Nachricht
  - » Keine doppelten Nachrichten
  - » Gefährdungen: Protokolle, die wiederholte Nachrichtenübertragung zulassen (-> Lösung: Protokolle, die Abfolgenummern in Nachrichten zulassen)
  - » Sicherheitsmaßnahmen zur Reaktion auf „Böswillige Nutzer“

Modellperspektiven verteilter Systeme



Systemmodell: Sicherheit [I]

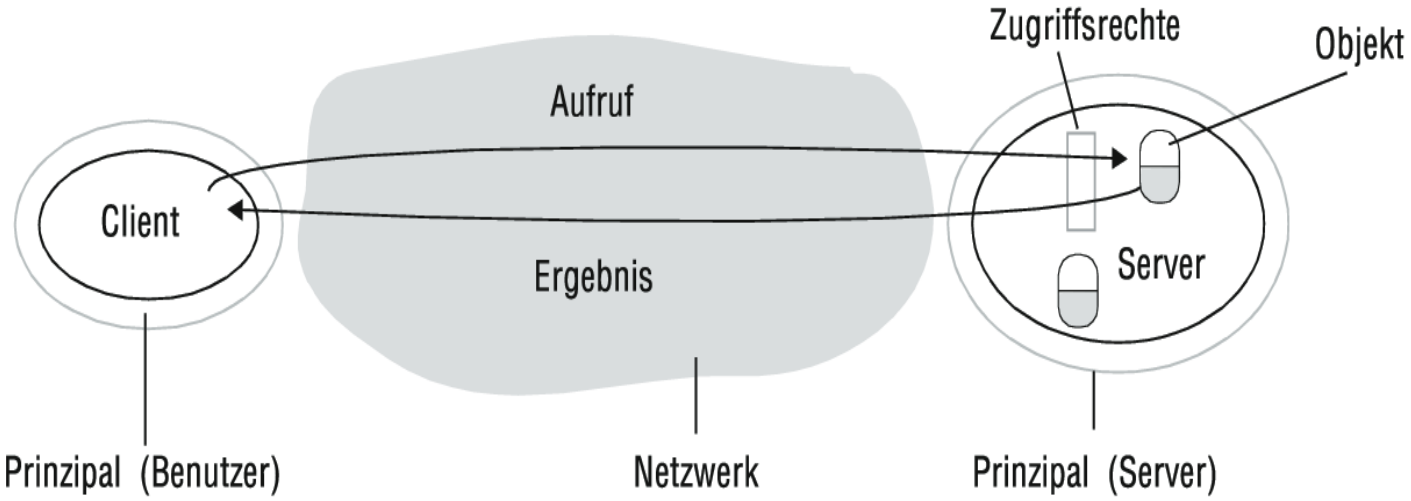
- Ziel: Antizipation möglicher Sicherheitsrisiken/Systemgefährdungen (intern/extern) im Rahmen eines Bedrohungsmodells als Grundlage für Analyse und Entwurf „sicherer“ VS:

- Definition & Klassifikation: was soll gesichert (vor unerlaubtem Zugriff geschützt) werden? genutzte Kanäle / gekapselte Objekte ?

- Auflistung aller potenziellen
  - Bedrohungen
  - Bewertungen der Bedrohungen
  - Auswirkungen
  - Möglichen Gegenmaßnahmen
  - Aufwände/Kosten der Gegenmaßnahmen

Systemmodell: Sicherheit [II]

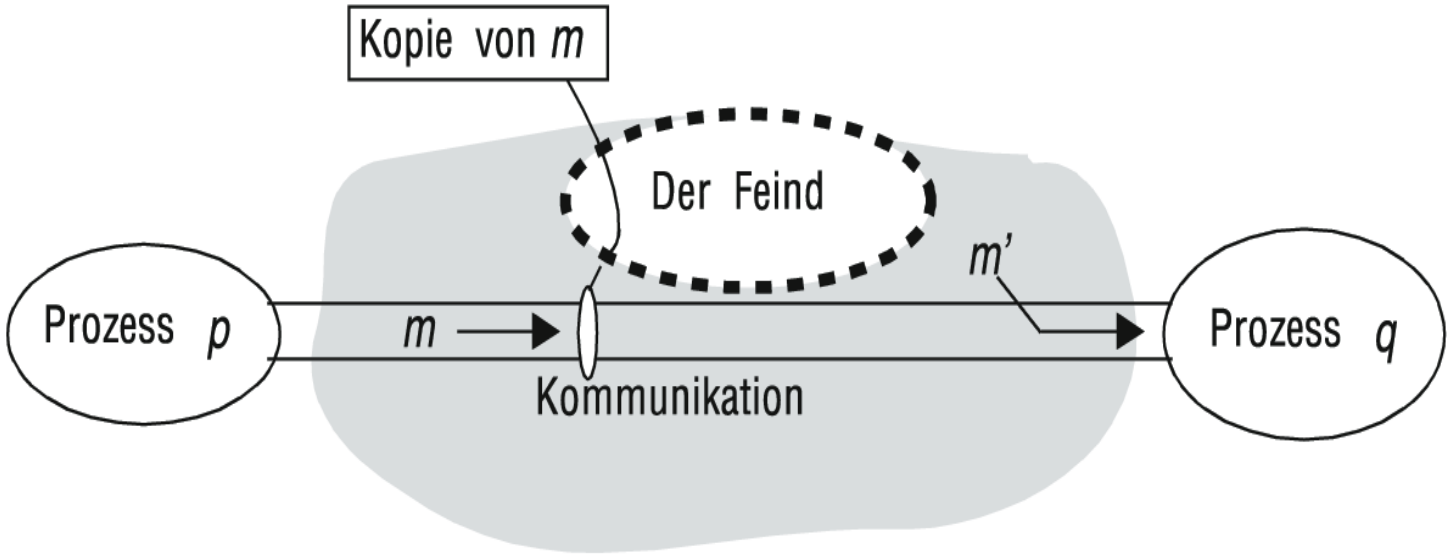
- Fokus: Schutz von Objekten, Prozessen, Interaktionen
  - » Umsetzung mittels Einrichtung von Zugriffsrechten, wer Operationen für ein Objekt ausführen kann (z.B. Zustand lesen, Zustand ändern)
  - » Zugriffsdimensionen: Aufruf der Operation, Ergebnis der Operation
  - » „Prinzipal“: Benutzer/Prozess mit Zugriffsrechten



Quelle: Coulouris et al. [2002: 80]

Systemmodell: Sicherheit [III]

- ✓ Potenzielle Herausforderungen / Risiken:
  - » VS verwenden offene Schnittstellen, Netzwerke und Kommunikationsdienste
  - » Hohe Wahrscheinlichkeit von Angriffen durch böswillige Nutzer



Quelle: Coulouris et al. [2002: 81]

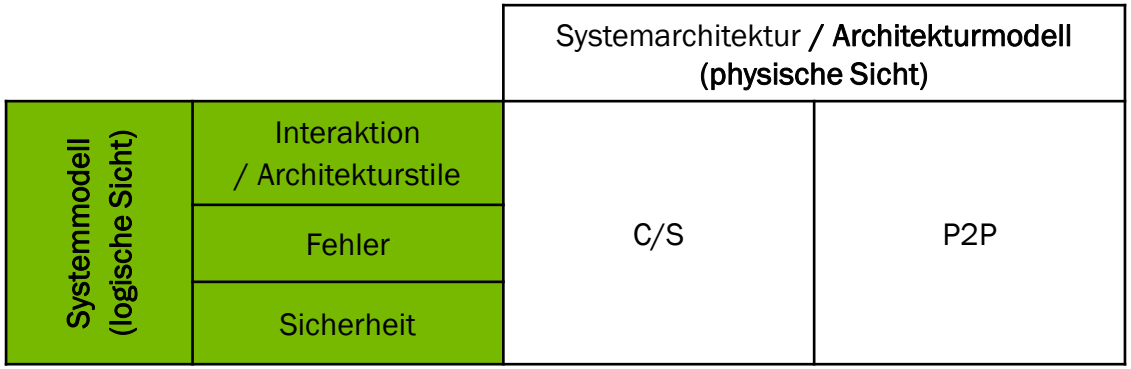
Systemmodell: Sicherheit [IV]

- Annahme: Es gibt einen/mehrere Feind(e)/Gegner
  - » Feind kann beliebige Nachrichten an beliebige Prozesse senden
  - » Feind kann alle Nachrichten zwischen einem Prozesspaar lesen/kopieren
- Ziel: Antizipation feindlicher Gefährdungen
  - » Bedrohungen von Prozessen durch fehlende Identität des Prinzipals:
    - Prozess kennt sendende Quelle nicht
    - Prozess erhält Nachricht mit gefälschter Quelle
  - » Bedrohung von Kommunikationskanälen: Nachrichten können zwischen Sender und Empfänger kopiert, geändert oder eingespeist werden
  - » Bedrohung der Privatsphäre, Informationsintegrität, Systemintegrität

Systemmodell: Sicherheit [V]

- ✓ Gegenmaßnahmen (Auszug; vgl. „Sicherheit“):
  - » Verwenden sicherer Kanäle (z.B.: Virtual Private Network (VPN), Secure Socket Layer (SSL))
  - » Verschlüsselung von Nachrichten
  - » Authentifizierung: Überprüfung der Identität (Dateiinhalt + Prinzipal) durch Vorab-Senden einer verschlüsselten Teilnachricht als ersten Schritt der Anforderung (Identität des Prinzipals, Identität der Datei, Zeit/Datum der Anforderung)

Danke. Lernziele erreicht?



Nach dieser Lehrveranstaltung kennen Studierende idealerweise:

- ✓ Grundlegende Aspekte und Perspektiven der logischen Architektur verteilter Systeme (Systemmodell):
  - ✓ Interaktion / Architekturstile:
    - ✓ Konzepte zur logischen Anordnung von Softwarekomponenten eines verteilten Systems
    - ✓ Unterscheidung und Bewertung der Konzepte in Kategorien- und Varianten (Granularität, Kopplung, Isolation, Schichtung, Objektfokus, Datenfokus, Ereignisfokus)
  - ✓ Fehler: Klassen und Arten
  - ✓ Sicherheit (vgl. hierzu auch Lecture Notes „Sicherheit“)
- ✓ Probleme und Maßnahmen zur Risikominimierung in den Dimensionen Interaktion, Fehler und Sicherheit
- ✓ Die Notwendigkeit zur Planung / Berücksichtigung von Einflussfaktoren in den jeweiligen Dimensionen

Die Frage, die sich nun stellt, wie logische Architekturüberlegungen zu einem System tatsächlich physisch realisiert werden können (als Instanz einer Klasse des spezifischen Systemmodells). Grundsätzlich kann dies zentralisiert, dezentralisiert oder hybrid (beeinflusst durch die Einordnung der Gleichrangigkeit der beteiligten Komponenten) erfolgen. Diese Themen werden in den kommenden Wochen Fokus des Moduls sein…

Quellen

Anthony, R. (2015) *Systems Programming - Designing and Developing Distributed Applications*; Amsterdam et al.: Morgan-Kaufman / Elsevier.

Coulouris, G.; Dollimore, J.; Kindberg, T. (2002) Verteilte Systeme - Konzepte und Design; 3., überarbeitete Auflage; München: Pearson Studium.

Fokkink, W. (2013) *Distributed Algorithms: an intuitive approach*, Cambridge, MA (USA): MIT Press.

Schill, A.; Springer, T. (2012) *Verteilte Systeme*; 2. Auflage; Berlin, Heidelberg: Springer Vieweg.

Tanenbaum, A.; van Steen, M. (2008) Verteilte Systeme – Prinzipien und Paradigmen; 2., überarbeitete Auflage; München: Pearson Studium.