AI-B.41: Verteilte Systeme

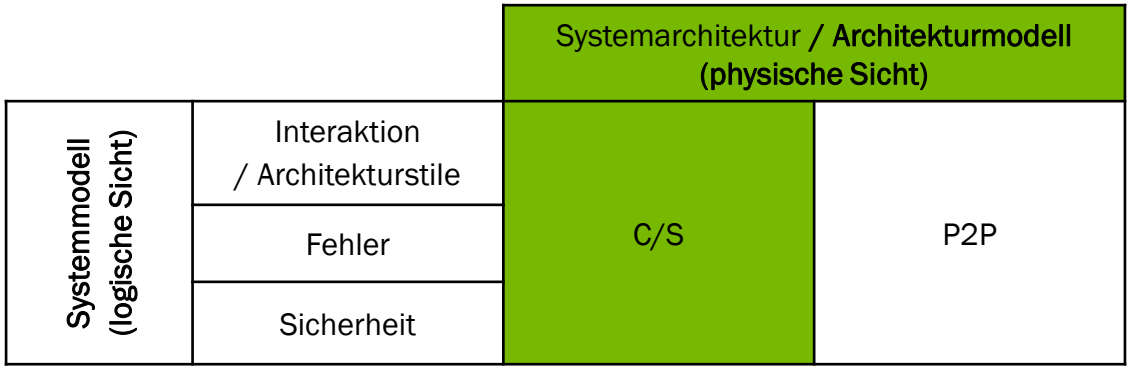
- Lecture Notes [SL] -

IV. Systemarchitekturen [II]
- P2P (Peer-to-Peer) -

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

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

Rückblick letzte Vorlesung [I]



Nach dieser Lehrveranstaltung kennen Studierende idealerweise:

- Kernmerkmale und Unterschiede grundsätzlicher Systemarchitekturen verteilter Systeme
- Aspekte einer zentralisierten C/S-Systemarchitektur im Kontext eines geschichteten Architekturstils
- Merkmale verschiedener Interaktionssemantiken in Client-Server-Systemen
- Klassifikationsansätze von Servern nach Zustand, Nachrichtenbearbeitung, Aktivierung
- Variationen von C/S-Systemen innerhalb mehrschichtiger Architekturen
- Ansätze zur Klassifikation von C^+SS^+-Systemen nach Rolle sowie Verkettung von Servern in geschichteten Architekturen
- Vergleichskriterien zur Architekturauswahl

✓ Studierende lernen vertiefende Aspekte zur Bewertung von Systemarchitekturen verteilter Systeme, welche das zu Grunde liegende logische Systemmodell und den Architekturstil physisch unterschiedlich implementieren können.

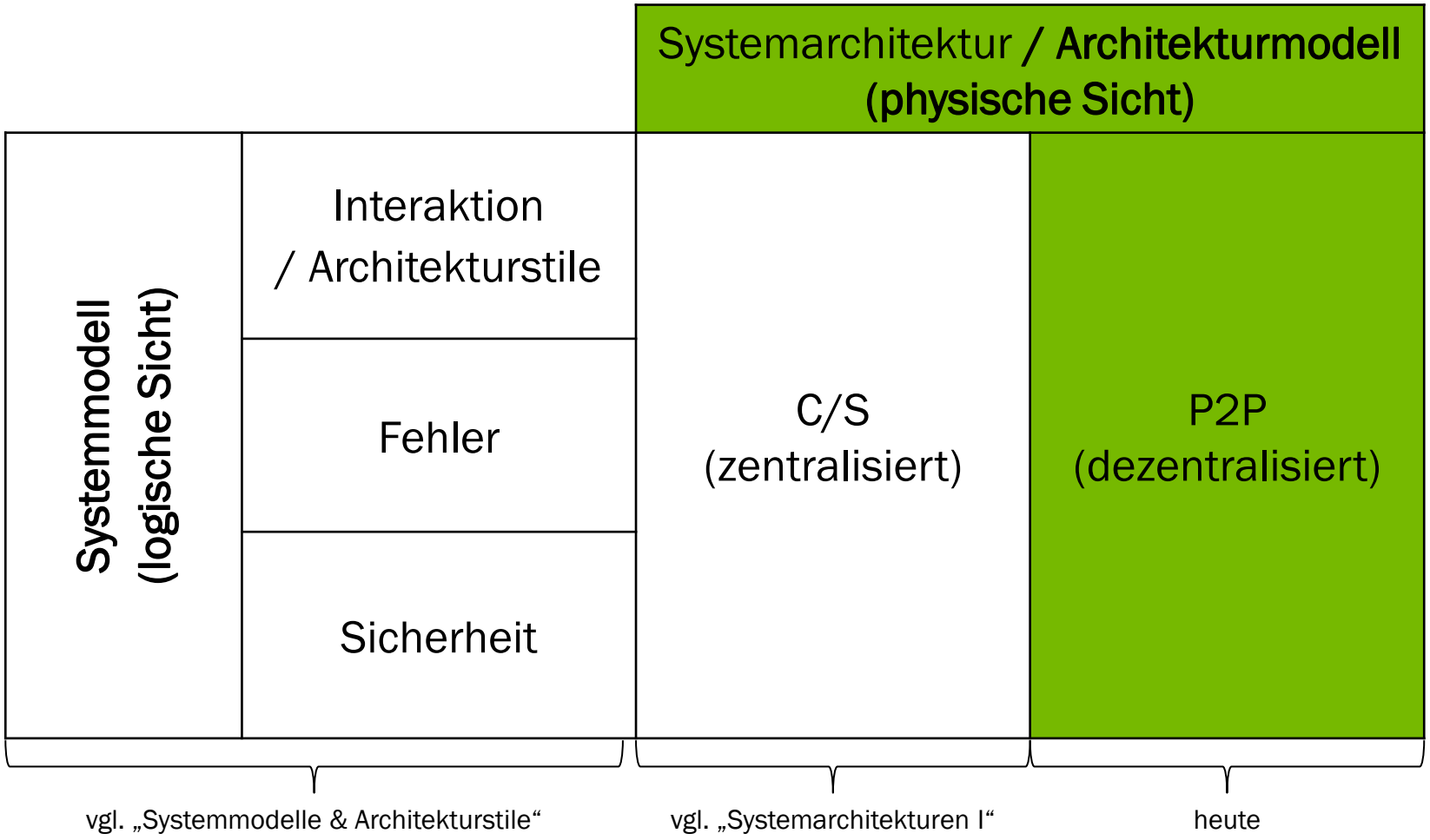
Rückblick letzte Vorlesung [II]

Wir erinnern uns:

- Ein Architekturstil entspricht der logischen Anordnung der Softwarekomponenten eines verteilten Systems (syn.: Softwarearchitektur)
- Nicht zu verwechseln mit der Systemarchitektur
  - » physische Realisierung der Verteilung,
  - » Instanz einer Softwarearchitektur

- Systemarchitekturen:
  - ✓ Client-Server
    - ✓ Nicht gleichrangig
    - ✓ Zentralisiert
  - » Peer Process, Peer-to-Peer (P2P)
    - Gleichrangig
    - Dezentralisiert
  - » Hybride Architekturen als Mischform zwischen C/S und P2P (nicht Teil dieses Moduls)

Modellperspektiven verteilter Systeme



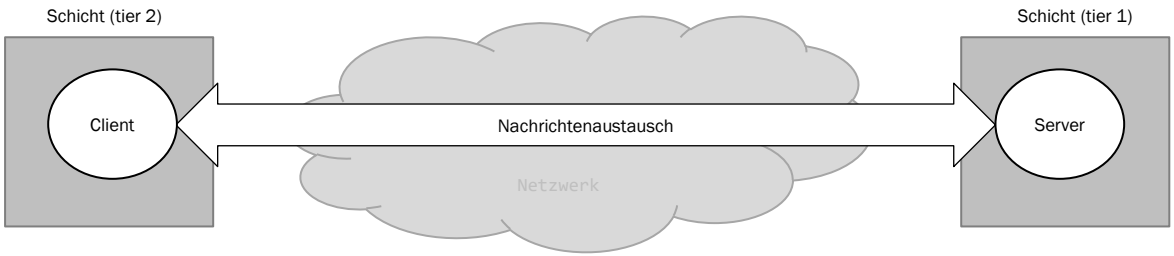
Lernziele

Nach dieser Lehrveranstaltung kennen Studierende idealerweise:

- Aspekte einer dezentralisierten Systemarchitektur
- Merkmale, Eigenschaften von P2P-Architekturen
- Möglichkeiten zur Abgrenzung von P2P zu C/S-Architekturen
- Klassifikationsansätze von P2P-Architekturen nach Zentralisierung, Rollenverteilung, Strukturierung
- Charakteristika einer logischen Netzstruktur (Overlay)
- Ausgewählte Topologien und Algorithmen im Kontext strukturierter P2P Overlay-Netze
- Eine Möglichkeit zur Anordnung von Prozessen mittels Distributed Hash Tables (DHT)
- Entwurfsaspekte und Vergleichskriterien zur Architekturauswahl und –bewertung
- Brewer‘s Theorem (CAP-Theorem)
- Ausgewählte Beispiele zu den o.g. Themen / Inhalten

✓ Studierende lernen vertiefende Aspekte zur Bewertung von Systemarchitekturen verteilter Systeme, welche das zu Grunde liegende logische Systemmodell und den Architekturstil physisch unterschiedlich implementieren können.

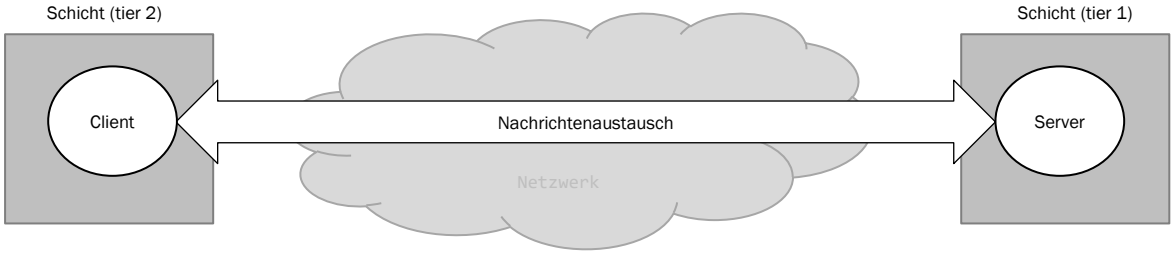
Rückblick: Client-Server (C/S) [I]



- ✓ Zentralisierte Architektur
- ✓ Netzteilnehmer haben unterschiedliche Funktionen:
  - ✓ Server:
    - ✓ Empfänger von Dienstanfragen des Clients
    - ✓ Sender, Erbringer von Diensten für Client
  - ✓ Client:
    - ✓ Sender von Dienstanfragen an Server
    - ✓ Empfänger, Nutzer von Diensten eines Servers
- ✓ Unterschiedliche Ausprägungen, z.B. „n-tier“ (ein-, zwei-, drei- oder multi-tier) i. A. der Darstellung, Verarbeitung, Speicherung von Daten

Vorteil_e	Nachteil_e
– Administration – Koordination	– Risiko leistungsbezogener Engpässe durch Bündelung von Anfragen – Hohe Tragweite von Fehlern / Ausfällen (Single Point of Failure) [Stein 2004:482 f.]

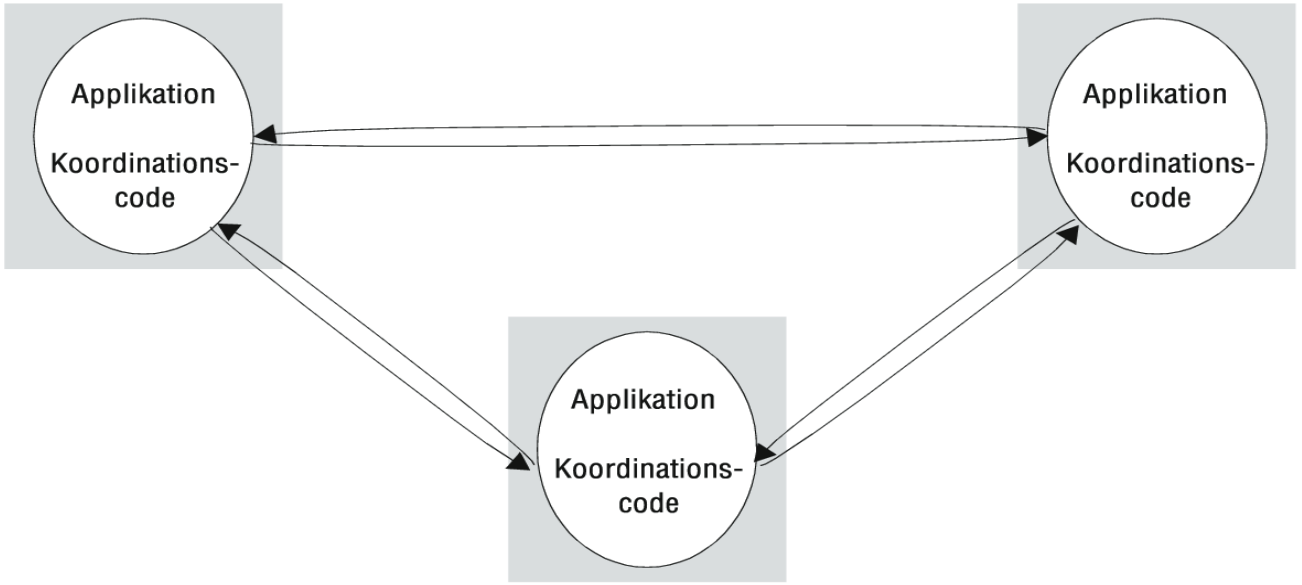
Rückblick: Client-Server (C/S) [I]



Vorteil_e	Nachteil_e
– Administration – Koordination – Daten und Dienste lassen sich leicht suchen / finden	– Die Architektur ist schlecht skalierbar: Spitzenlasten durch zu hohe Anfragelast bei Servern bilden bottlenecks, welche es nicht mehr zulässt, Clients im Rahmen akzeptabler Antwortzeiten zu bedienen (Risiko leistungsbezogener Engpässe durch Bündelung von Anfragen) [Dunkel et al. 2008:142] – Hohe Tragweite von Fehlern / Ausfällen (Single Point of Failure) [Stein 2004:482 f.] – Zur Verfügung stehende Rechenleistung von Clients werden nicht gut genutzt [Dunkel et al. 2008:142]

Ausblick: Gleichrangige Prozesse / Peer-to-Peer (P2P)

- Form eines Rechnernetzes, welches aus Sicht der Anwendungen gleichberechtigte Teilnehmer („Peers“) miteinander verbindet [Stein 2004]
- System mit vollständig dezentraler Selbstorganisation und Ressourcennutzung [Steinmetz/Wehrle 2004]
- Verwaltung / Synchronisation von Ressourcen erfolgt auf Applikationsebene



Quelle: Coulouris et al. [2002:56]

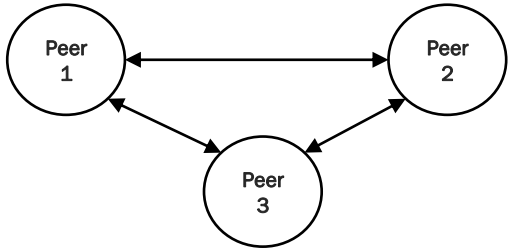
P2P: Beispiele

vgl. Dunkel et al. [2008:142]

- Bekanntheitsgrad durch Datei-Tauschbörsen: Musik- und Filme
  - » Negative Schlagzeilen durch Verletzung von Urheberrechten durch illegalen Download von Dateien
  - » Beispiele: Gnutella, Napster, eDonkey, BitTorrent
- P2P-Anwendungen produzieren mehr als die Hälfte des im Internet anfallenden Datenvolumens
- Bedeutung von P2P:
  - » Quantitativ: hoch (durch Anteil am Datenvolumen)
  - » Qualitativ (gemessen an Durchdringung in ausgereiften Technologien / Produkten): gering (vorrangig: Filesharing)
- Es gibt weitere sinnvolle Anwendungen:
  - » Internet Relay Chat (IRC)
  - » Internet-Telefonie (z.B. Skype)
  - » Ad-hoc Netze
  - » Mobile Computing (vgl. Module AI)

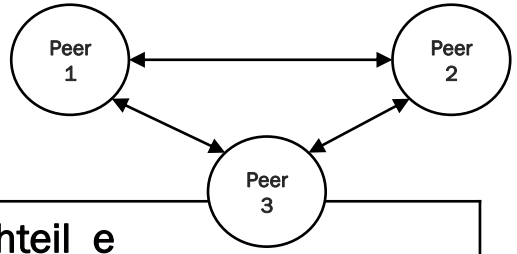
✓ Was genau ist nun P2P?

P2P: Eigenschaften



- Dezentralität:
  - » Keine zentrale Autorität/Steuerung
  - » Keine zentrale Datenakkumulation
- Autonomie: Peer entscheidet, ob und in welchem Ausmaß Ressourcen anderen Peers zur Verfügung gestellt werden
- Variable Konnektivität der Peers
- Rollensymmetrie („Servent“-Prinzip): Peers können gleichzeitig Server und Client sein
- Nutzung der Internet-Infrastruktur („Overlay“)
- Ausprägungen: rein / dezentral, zentralisiert, hybrid

P2P: Vor- und Nachteile

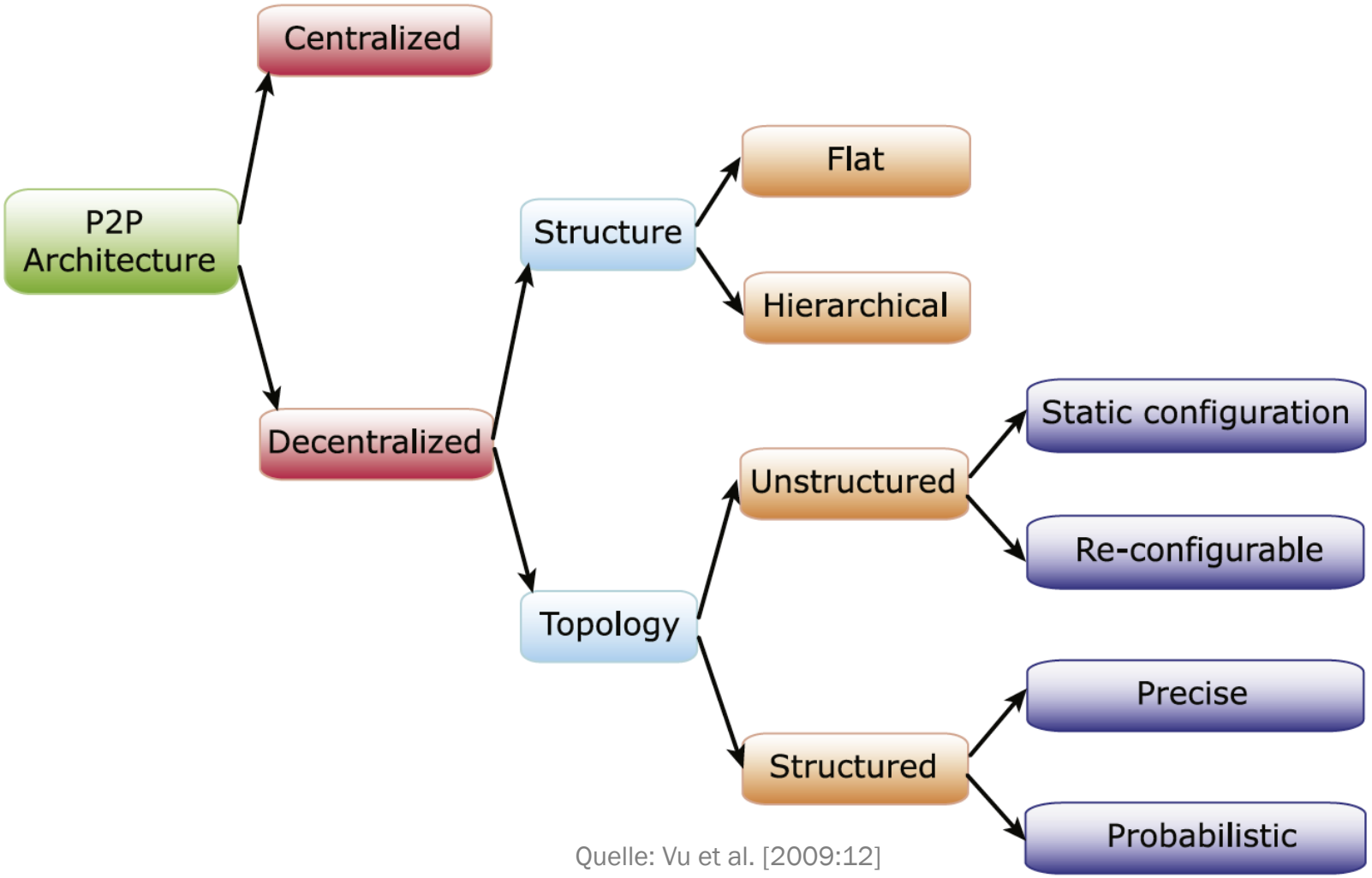


Vorteil_e	Nachteil_e

– Autonomie: Peers entscheiden, wann und in welchem Umfang Sie in Verbindung zum Gesamtsystem treten (-> Dynamik)
– Selbstorganisation:
  – kein vorgegebenes Modell zur Topologie
  – Kein vorgegebener Kontrollmechanismus
– Geringe Tragweite von Fehlern / Ausfällen von Peers
– Skalierbarkeit im Vergleich zu C/S-Systemen höher
– Geringer Administrationsaufwand durch Wegfall zentraler Instanzen (Server)
	– Autonomie / Selbstorganisation: Kein verlässlicher Grundnutzen
	– Unzuverlässigkeit der Partnerschaft zwischen den Peers, z.B. Free-Riding / Free-Loading: Geringe Motivation, Leistungen / Daten bereitzustellen [Adar/Hubermann 2000]
	– Bildung von Knoten und Bottlenecks (Paradoxon: P2P-Idee ist Dezentralisierung)
	– Gefährdung der (erhofften) Skalierbarkeit
	– Sicherheitsrisiken

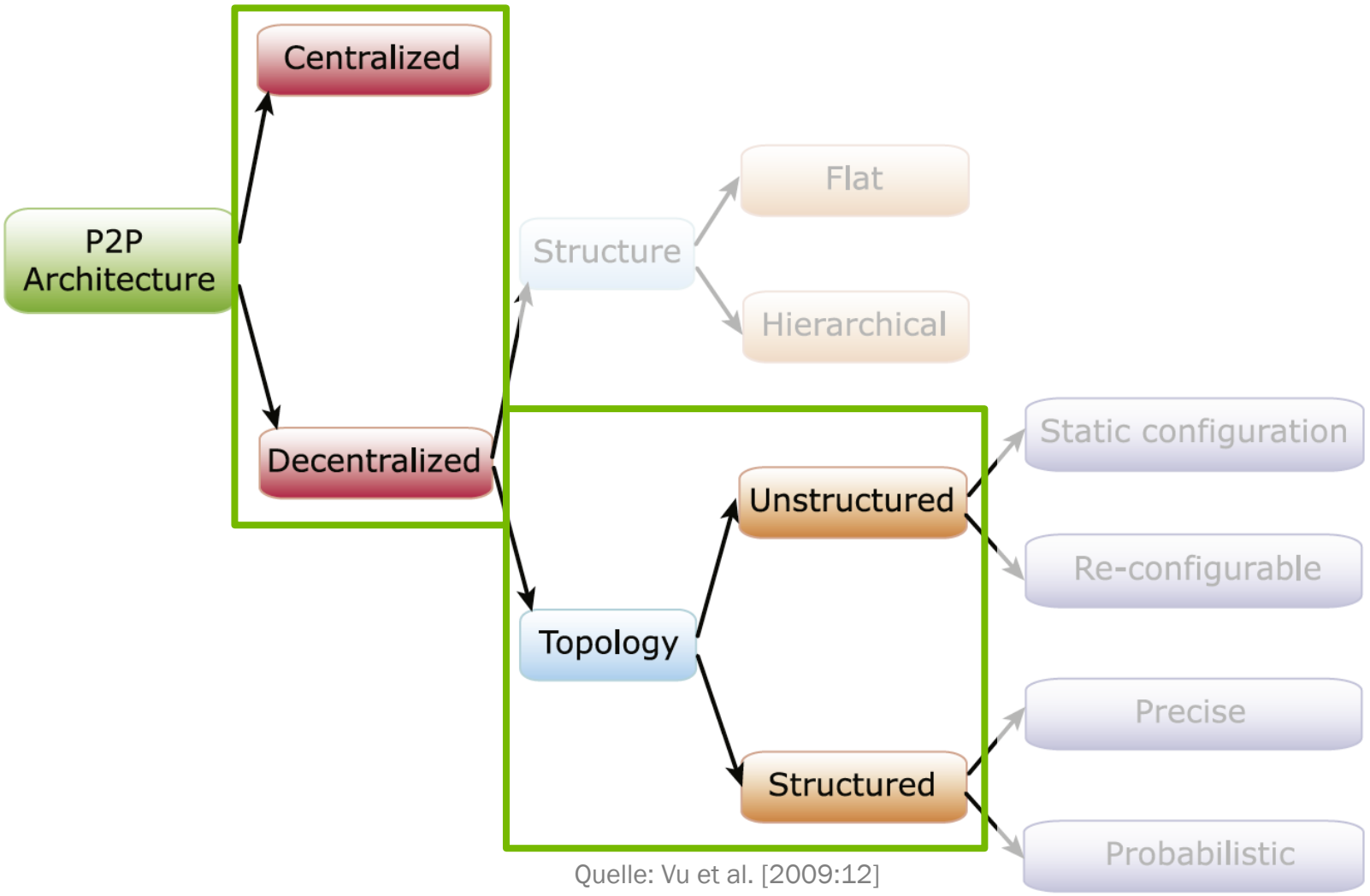
P2P-Systeme: Taxonomie

vgl. Vu et al. [2009]



Quelle: Vu et al. [2009:12]

P2P-Systeme: Typisierungen im Fokus (vgl. ff.)

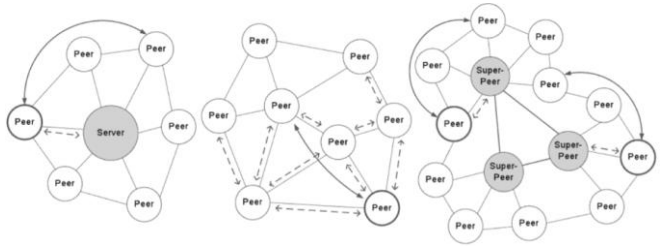


Quelle: Vu et al. [2009:12]

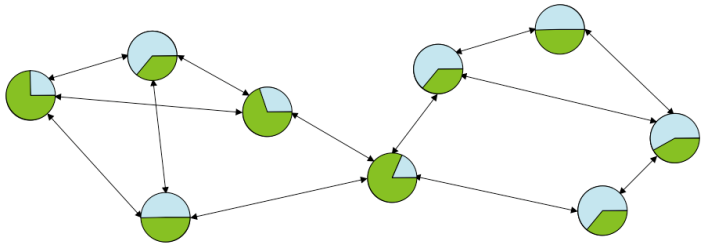
P2P-Systeme: Typisierung

Nach Grad der

- Zentralisierung



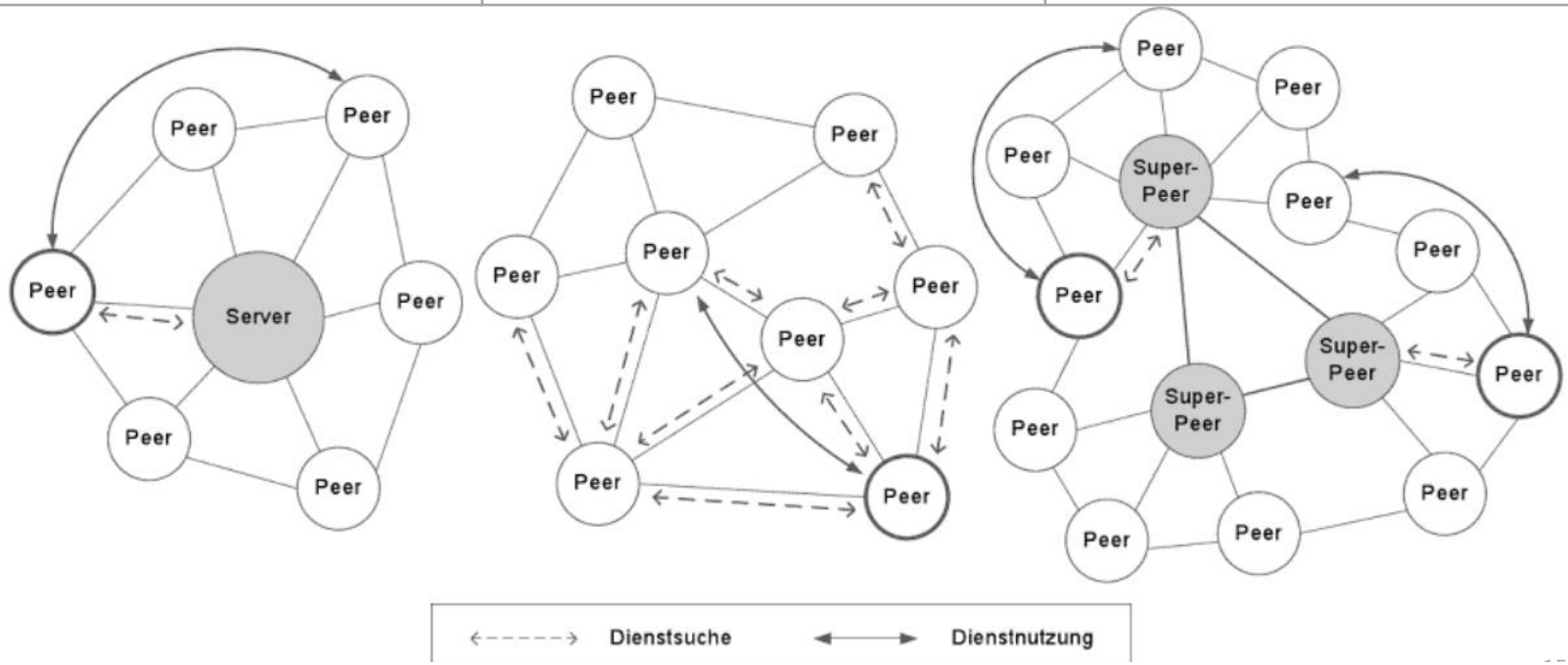
- Rollenverteilung / Interaktion zwischen den Peers



- Strukturierung

Typisierung nach Grad der Zentralisierung

Zentralisiert (links)	Pur / rein (mitte)	Hybrid (rechts)
– Server als Dienstvermittler zwischen dienste-anbietenden Peers (registrieren bei Server) und dienste-anfragenden Peers – Nachteile: vgl. C/S – Beispiel: Napster	– Kein zentraler Server – Dienstsuche mit Flooded Requests per Broadcast an Nachbarpeers – Nachbarpeers reichen die Anfrage an ihre direkte Nachbarn weiter, bis: – Entsprechender Anbieter gefunden wurde – Die maximale Anzahl von Flutungsschritten (Tim-to-live (TTL)) erreicht wurde – Beispiel: Gnutella, Bitcoin	– Verwendung von Superpeers – Enthalten / verwalten Teilmenge der Informationen / Dienste im gesamten Netz, welche nicht von allen Peers angeboten werden, z.B. Dienstangebot von Nachbarknoten – Dynamische Rolle von Superpeers – Beispiel: KaZaa, Skype

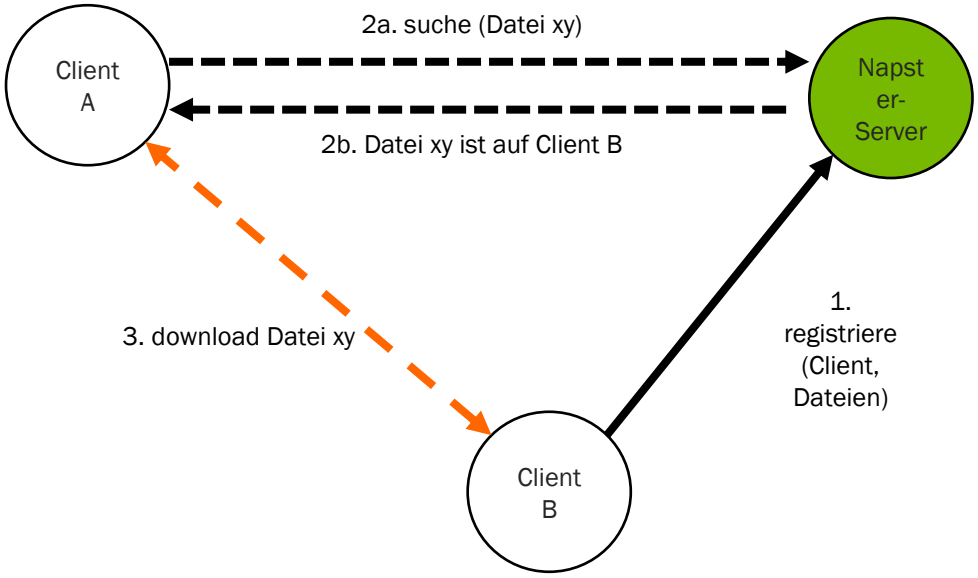


Bildquelle: Schill & Springer [2012:38]

Beispiel (zentralisierte Architektur): Napster

Musiktauschbörse / Filesharing (~1999):

- Clients
  - » Registrieren sich bei Server mit fester IP-Adresse
  - » Veröffentlichen Index Ihrer Dateien
- Server:
  - » Kennt Angebot im P2P-Netz
  - » Sendet nach Anfrage IP-Adressen verfügbarer Dienstanbieter
- Danach: P2P Kommunikation zwischen Clients (z.B. Download)



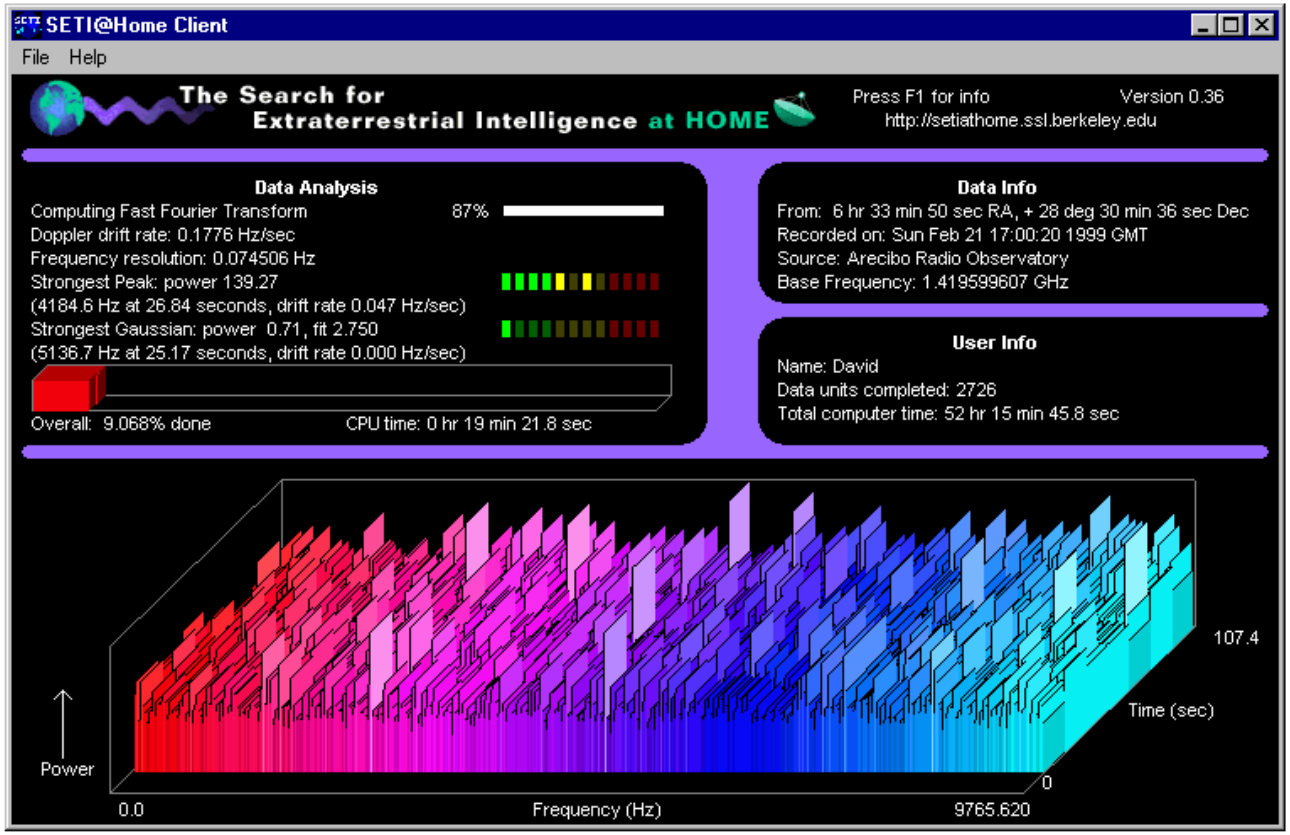
Bildquelle: Dunkel et al. [2008:145]

Beispiel (zentralisierte Architektur): SETI@home

vgl. http://setiathome.berkeley.edu [letzter Zugriff: 20 XII 17]

„Search for Extraterrestrial Intelligence“ (SETI):

- Suche nach außerirdischer Intelligenz mittels Radioteleskopie (seit 1999)
- „Boinc“: Peer-Client als Bildschirmschoner (erbringen Rechenleistung)
- Zentrale Kontrollinstanzen



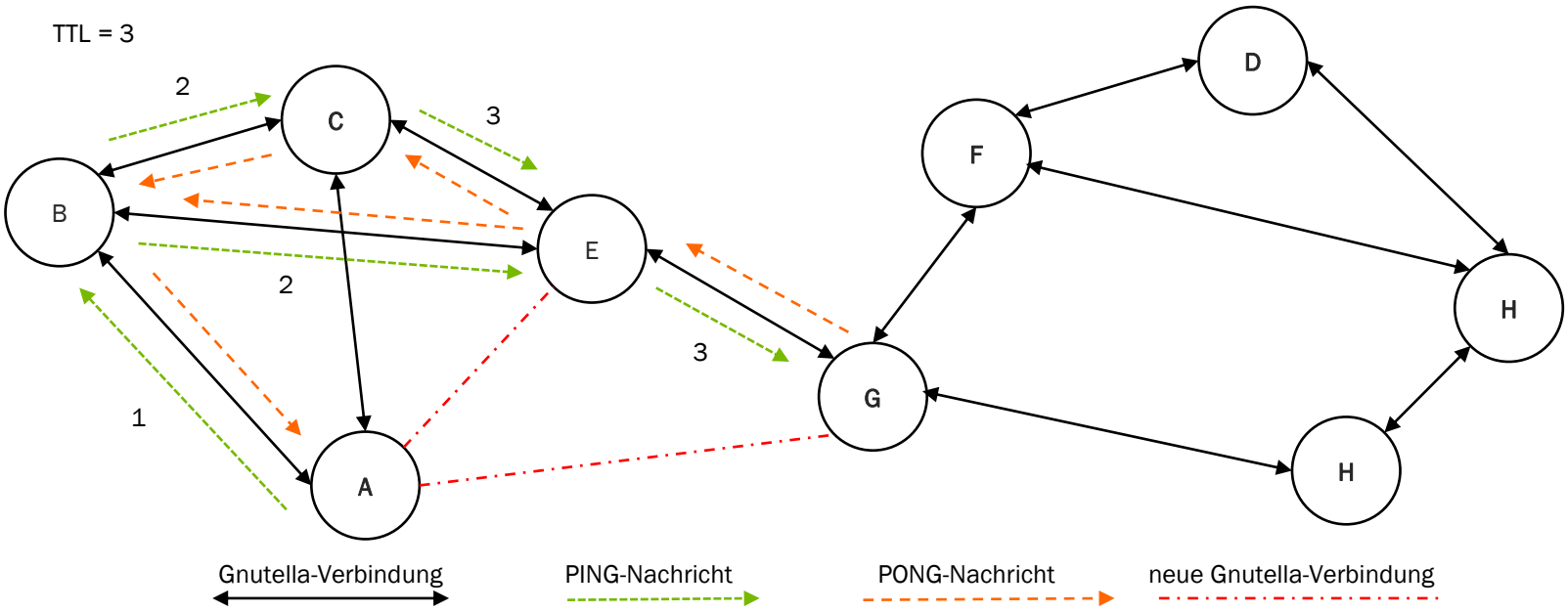
Bildquelle: http://science.nasa.gov/media/medialibrary/1999/05/17/ast23may99_1_resources/screen-detail.gif [Zugriff: 09.10.2010]

*/

Beispiel (dezentrale, unstrukturierte Architektur): Gnutella in frühen Versionen [I]

Filesharing (~2000)

- Beitritt eines Peers: Ping-Pong-Mechanismus
  - » Ping: Bei Eintritt eines Peers erfolgt Nachrichtenpropagation durch Flooding
  - » Pong: Alle Peers, welche Ping empfangen, senden einen Pong mit Ihrer IP-Adresse und weiteren Angaben
  - » TTL (time-to-live): Wert, der bei jedem Weiterleiten heruntergezählt wird (0=Ende der Weiterleitung)

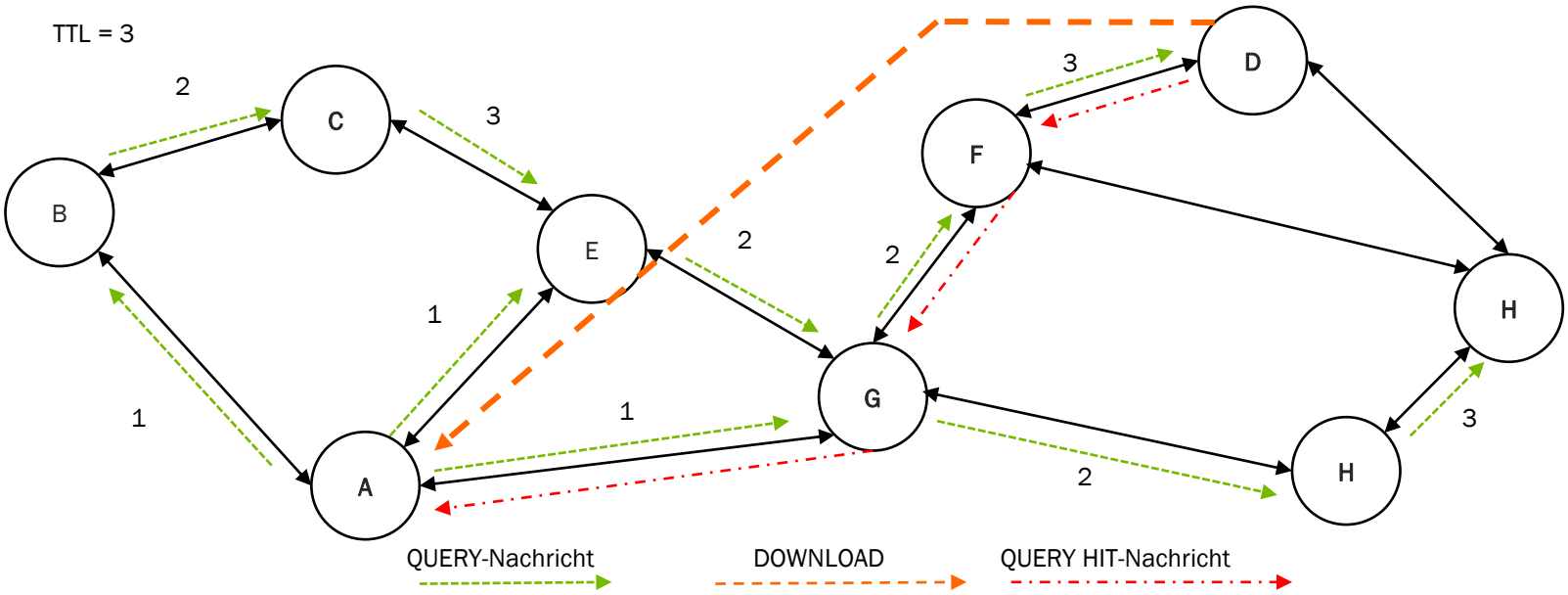


Bildquelle: i.A. an Anthony [2016:325] und Dunkel et al. [2008:146]

Beispiel (dezentrale, unstrukturierte Architektur): Gnutella in frühen Versionen [II]

Suche:

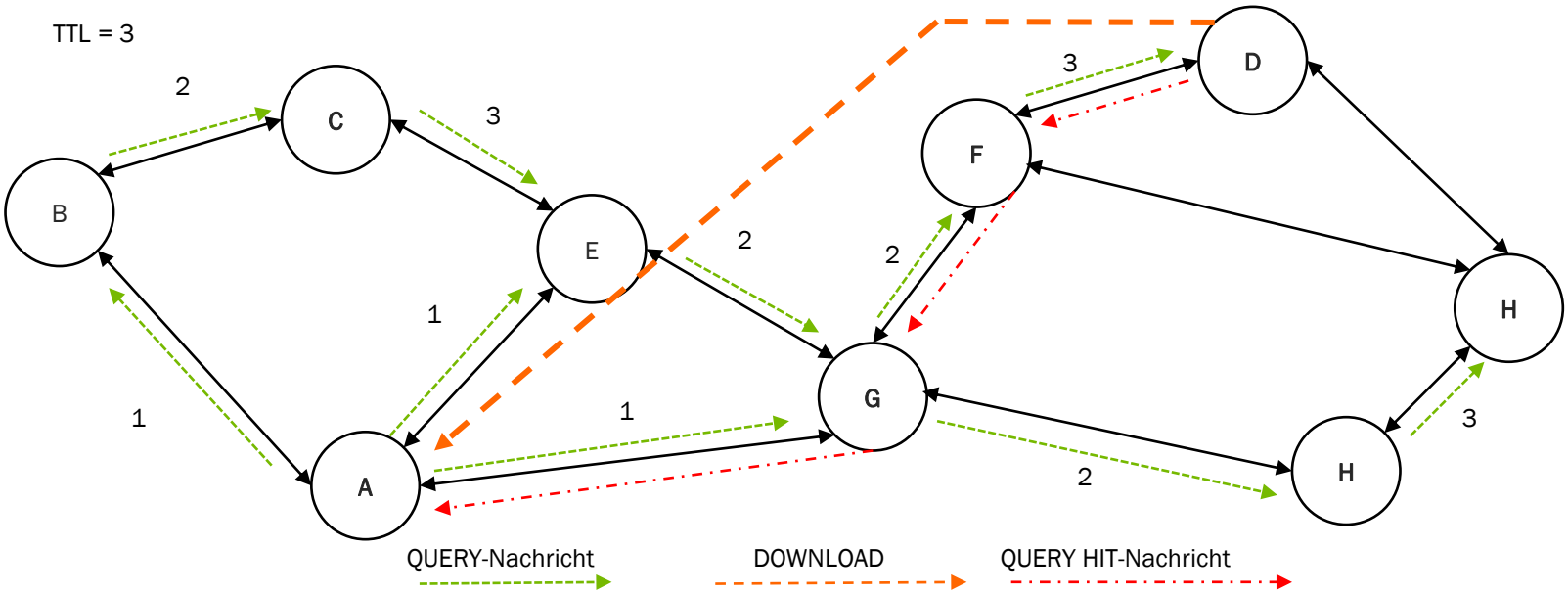
- Query: kaskadierende Anfrage (A) an benachbarte Peers mit TTL
- Query-Hit: umgekehrter Nachrichtenfluss im Falle, dass ein Peer die Query bedienen kann
- Danach: Datentransfer zwischen Peers (A, D)



Bildquelle: i.A. an Anthony [2016:325] und Dunkel et al. [2008:148]

Beispiel (dezentrale, unstrukturierte Architektur): Gnutella in frühen Versionen [III]

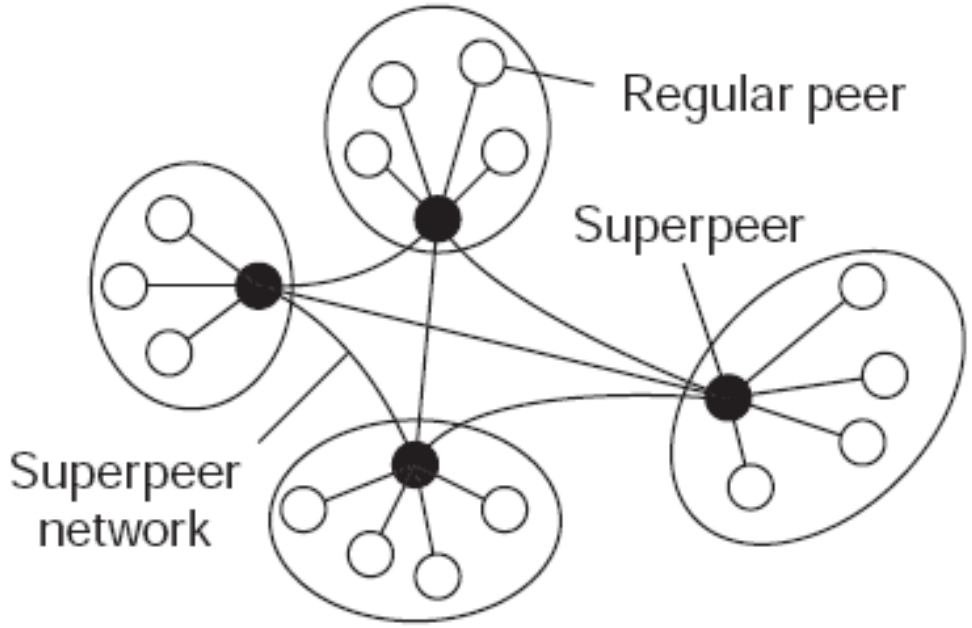
- Ohne TTL-Wert würde jeder Peer immer alle Nachrichten / Anfragen aller Knoten erhalten
- Free-Riding: wenige Peers bieten an, viele fragen nach
- Grundproblem: Suche ist unstrukturiert



Bildquelle: i.A. an Anthony [2016:325] und Dunkel et al. [2008:148]

Hybride Architektur / Superpeer-Konzept

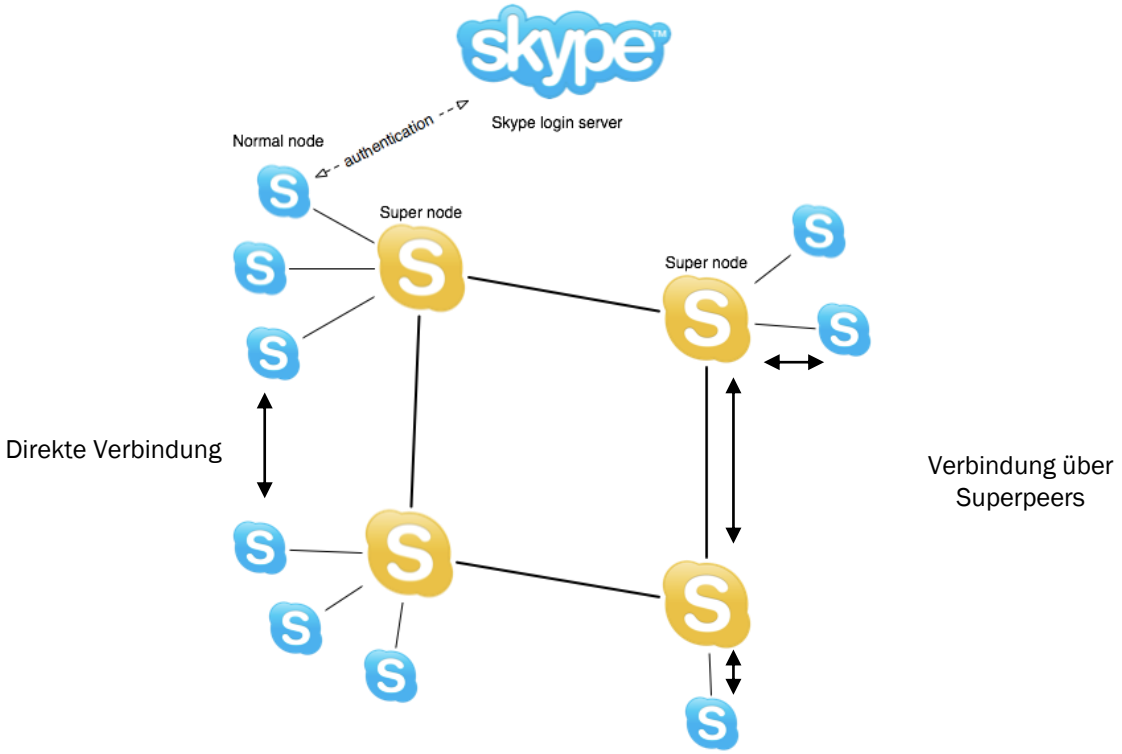
- Verwendung von Superpeers: verwalten Teilmenge der Informationen / Dienste im gesamten Netz
- Verschiedene dynamische Rollen von Superpeers
  - » Suchindex (Dienstangebot von Nachbarknoten)
  - » Monitoring
  - » Herstellung von Verbindungen



Bildquelle: http://cse.csusb.edu/tongyu/courses/cs660/images/2/superpeer.png [letzter Zugriff: 20 XII 17]; vgl. auch Tanenbaum & van Steen [2008:69]

Beispiel (hybride Architektur / Superpeer-Konzept): Skype

- VoIP-Applikation
- Knoten treten dem Overlay-Netz nach erfolgreicher Authentifizierung bei einem Skype-Login-Server bei
- Knoten sind entweder normal oder super



Bildquelle: i.A. an https://crypto.stanford.edu/cs294s/projects/skype-network-2.png [letzter Zugriff: 20 XII 17]

Typisierung nach Grad der Rollenverteilung / Interaktion zwischen den Peers

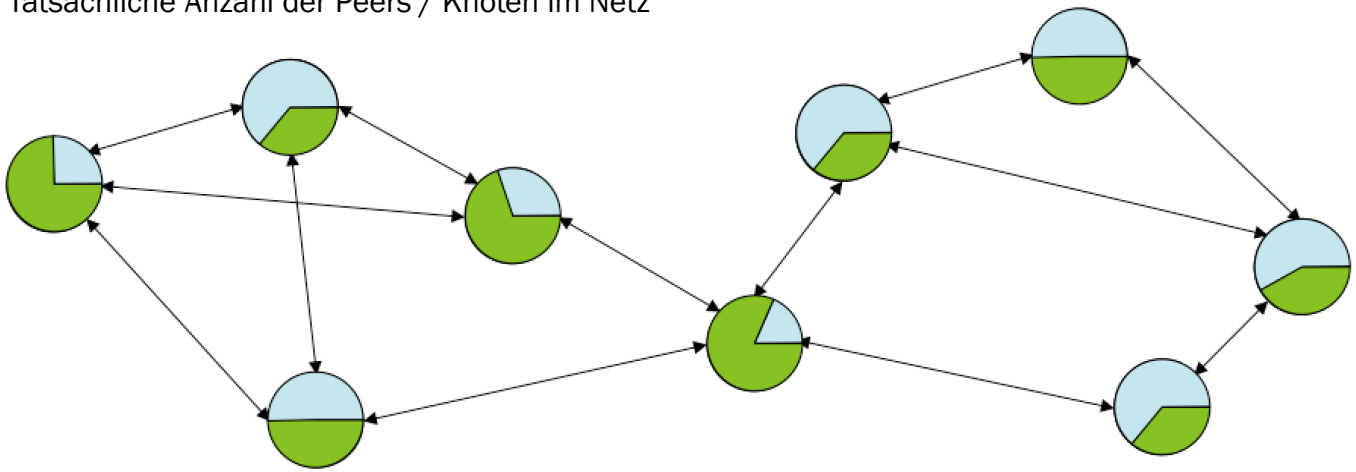
vgl. Anthony [2016:324ff.]

- ✓ Rollenverteilung bei den „servents“ (Anteil service requests / service replies)

- ✓ Interaktion: Anteil vorhandener Verbindungen an Verbindungsmaximum

  Formel: C = P(P-1) / 2

  C Maximale Anzahl der Verbindungen (Kanten)
  P Tatsächliche Anzahl der Peers / Knoten im Netz

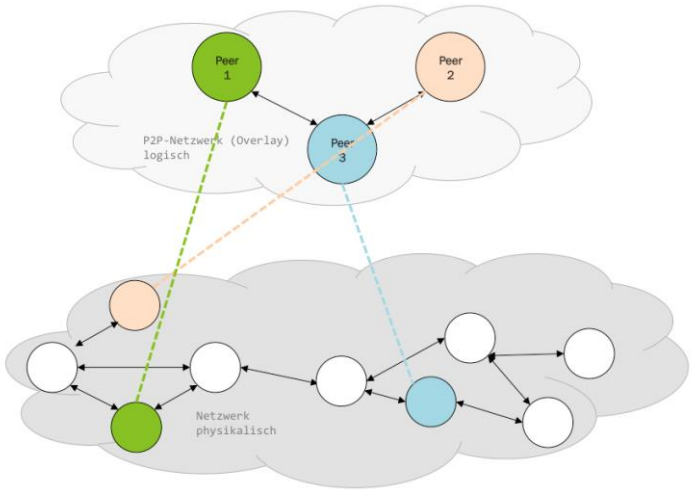


Bildquelle: eigene Darstellung i.A. an Anthony [2016:325]

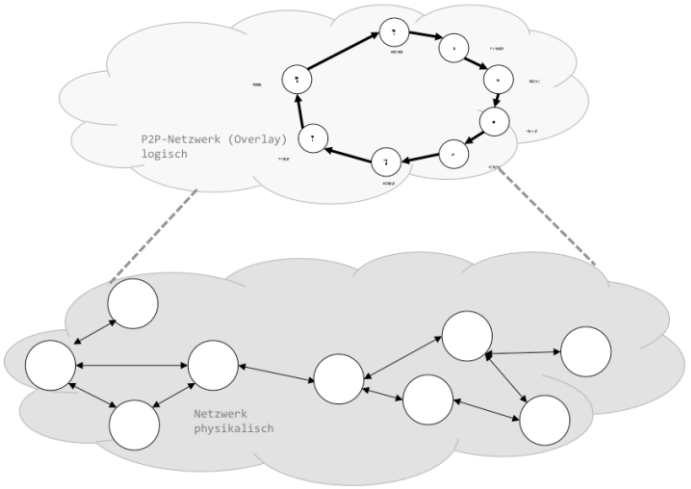
Typisierung nach Grad der Strukturierung

vgl. Steinmetz/Wehrle [2004]; Tanenbaum & van Steen [2008:62f.]; Tirado et al. [2010]

Unstrukturiert:
Organisationsstruktur nicht vorgegeben
(unstrukturiertes Overlay-Netz)

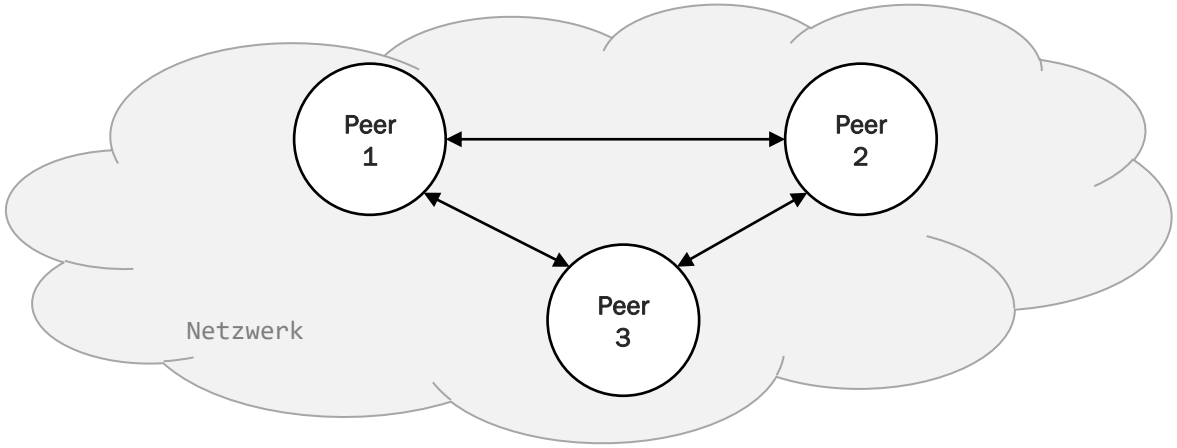


Strukturiert:
Topologie des Systems (Overlay-Netz) ist
durch ein Modell determiniert



Overlay [I]

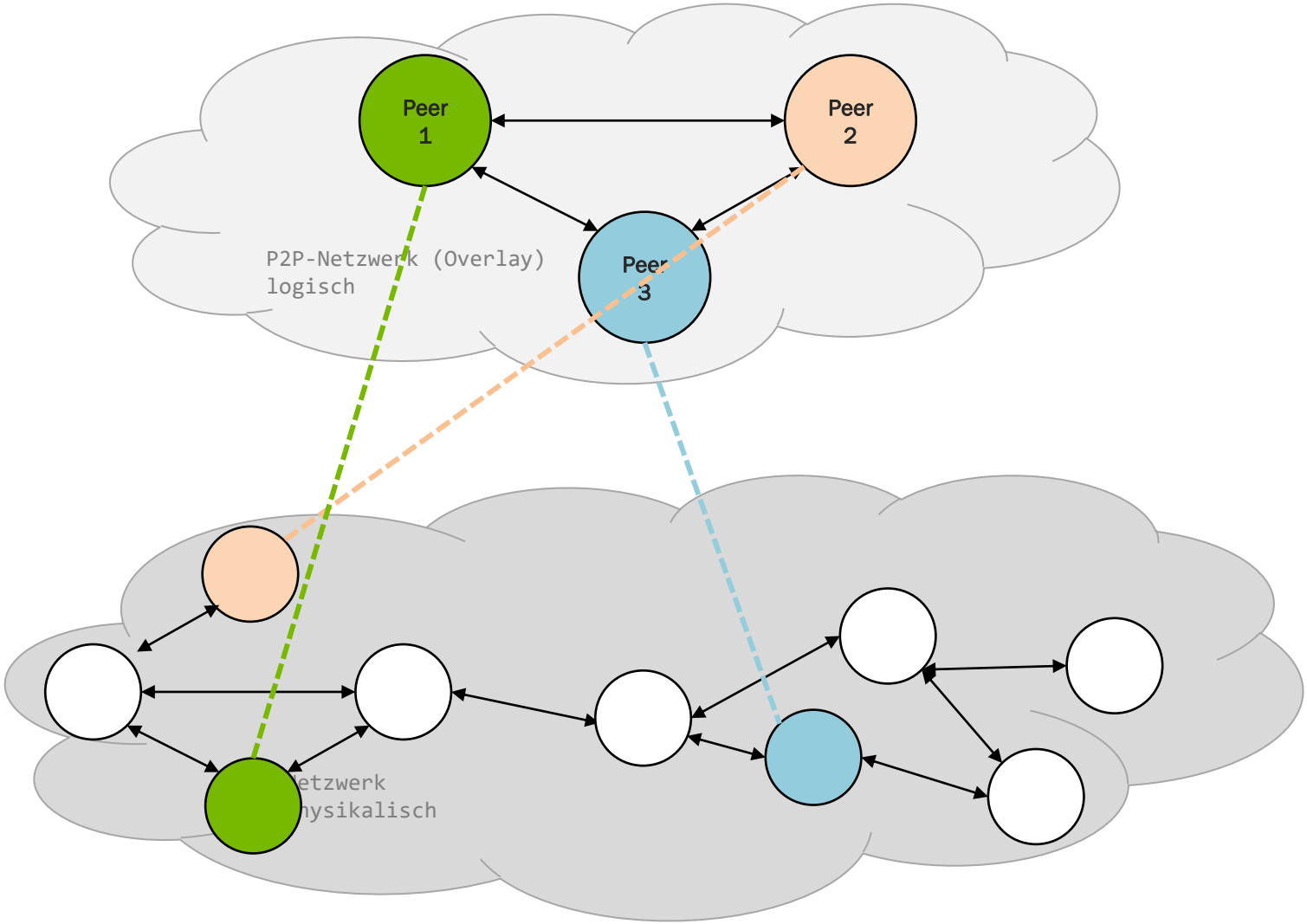
vgl. Dunkel et al. [2008:142ff.]



- Ein P2P-System ist applikationsbasiert.
- Damit die Knoten miteinander kommunizieren können, müssen sie über ein bestehendes Netzwerk miteinander verbunden werden,
- Diese Verbindung der Peers untereinander wird Overlay-Netzwerk genannt.
- Overlay-Netzwerk bezeichnet ein logisches Netz auf Basis eines physikalischen Netzwerks
- Das logische Netz (Overlay) kann eine andere Struktur als das physikalische Netz besitzen (vgl. f.).

Overlay [II]

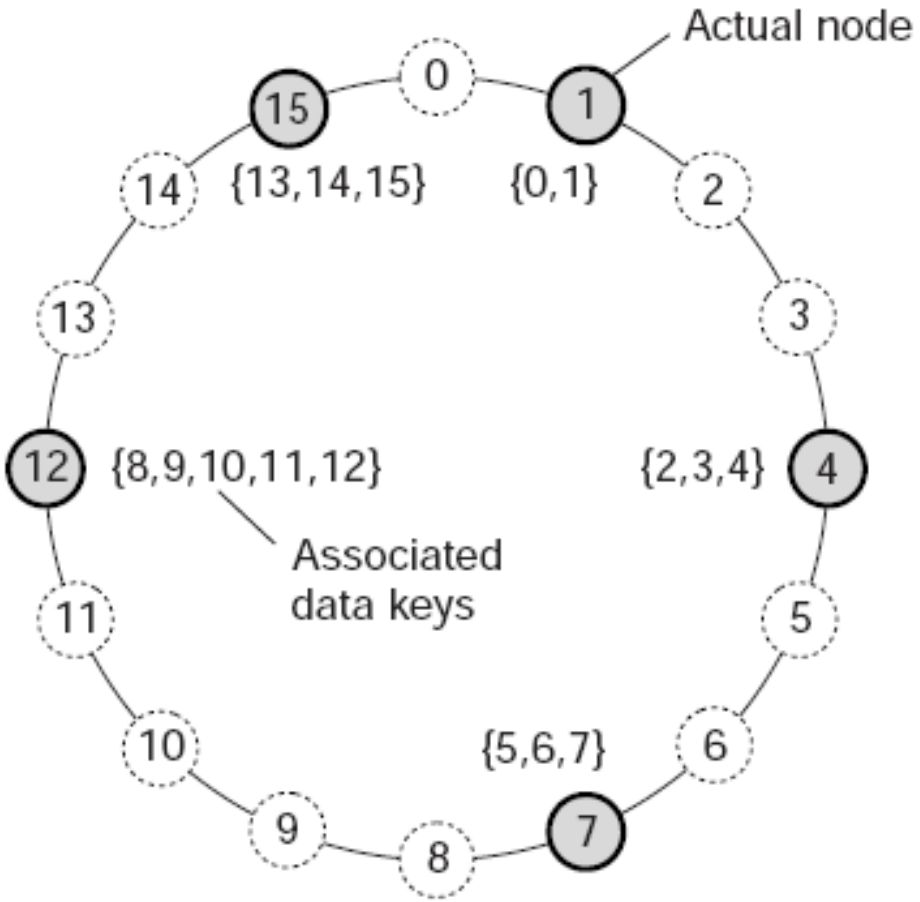
vgl. Dunkel et al. [2008:145]



Strukturierte P2P-Architekturen

vgl. Tanenbaum & van Steen [2008:62ff.]

- ✓ Topologie des Systems ist durch ein Modell determiniert
- ✓ Beispiel: Distributed Hash Tables (vgl. ff.)

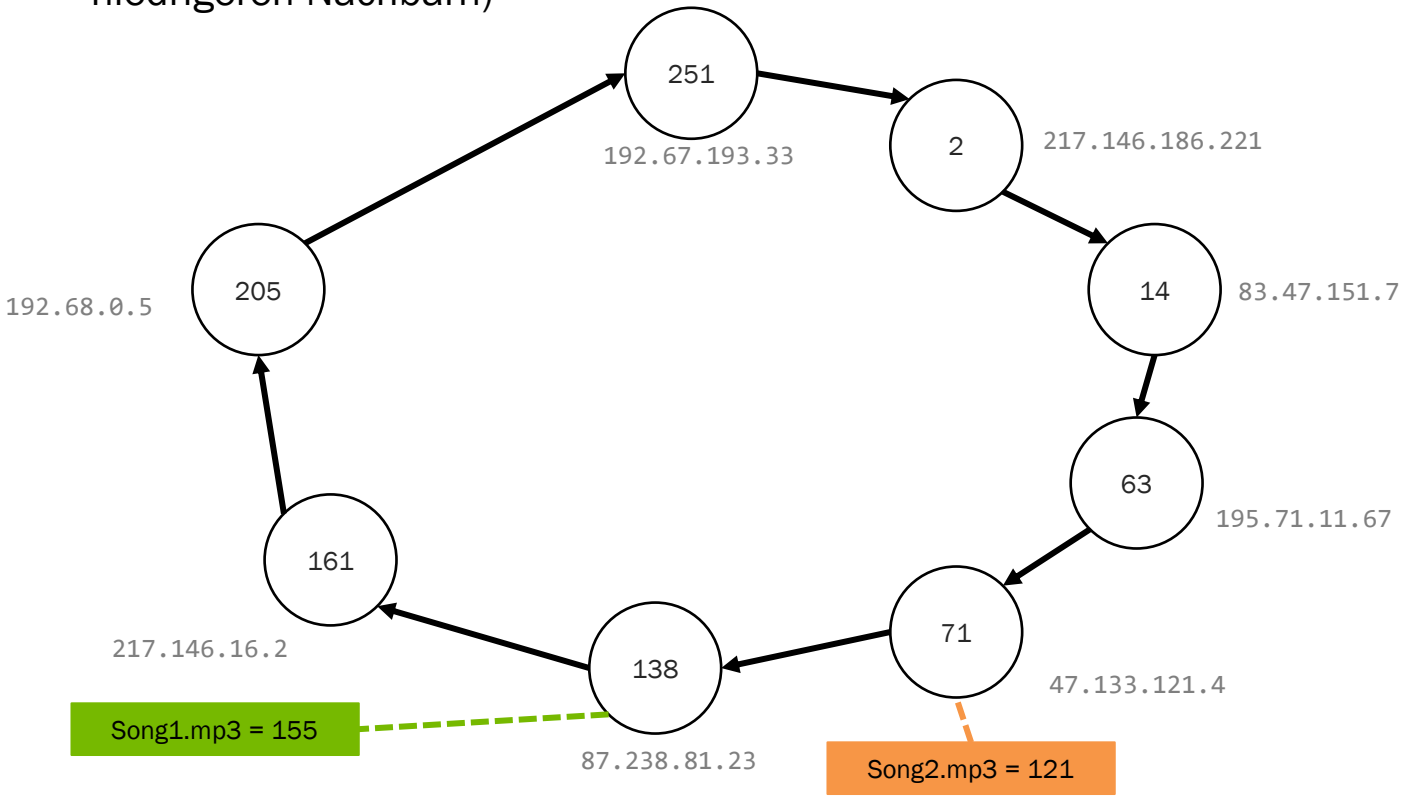


Bildquelle: Die Zuordnung von Datenelementen zu Knoten in Chord; Tanenbaum & van Steen. [2008:63]

Distributed Hash Tables [I]

vgl. Dunkel et al. [2008:149f.]

- Hashwerte von IP-Adressen der Peers, abgebildet auf einem abstrakten Ring (hier: aufsteigend im Intervall (0, 256))
- Suchschlüssel:
  - » Hashwerte der Dateien werden gebildet und den Peers nach deren Hash-ID zugeteilt
  - » Peers sind für Intervalle zuständig: (eigener Hashwert, Hash des nächst höheren / niedrigeren Nachbarn)

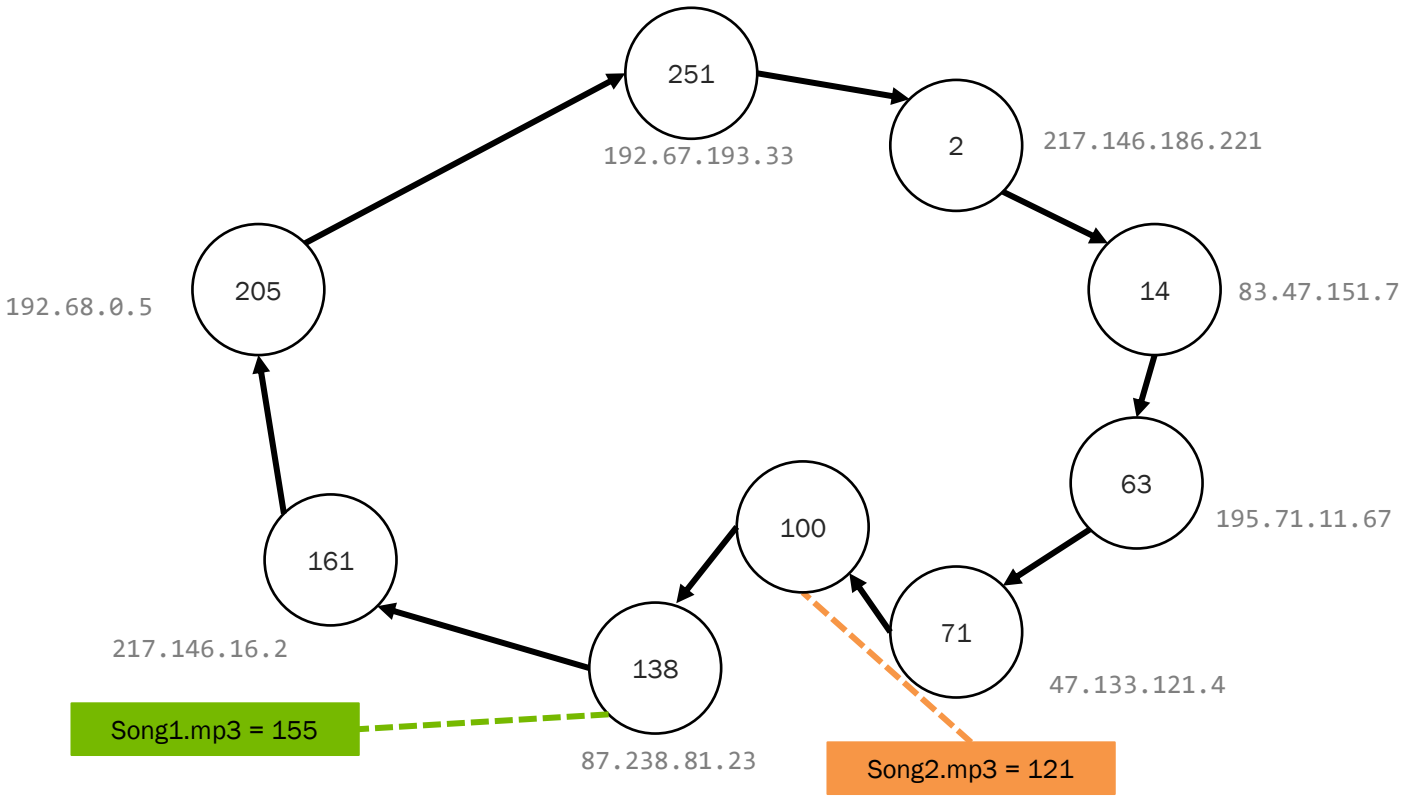


Bildquelle: i.A. an Dunkel et al. [2008:150]

Distributed Hash Tables [II]

vgl. Dunkel et al. [2008:149f.]

Eintritt eines Knotes (Hash-ID: 100)

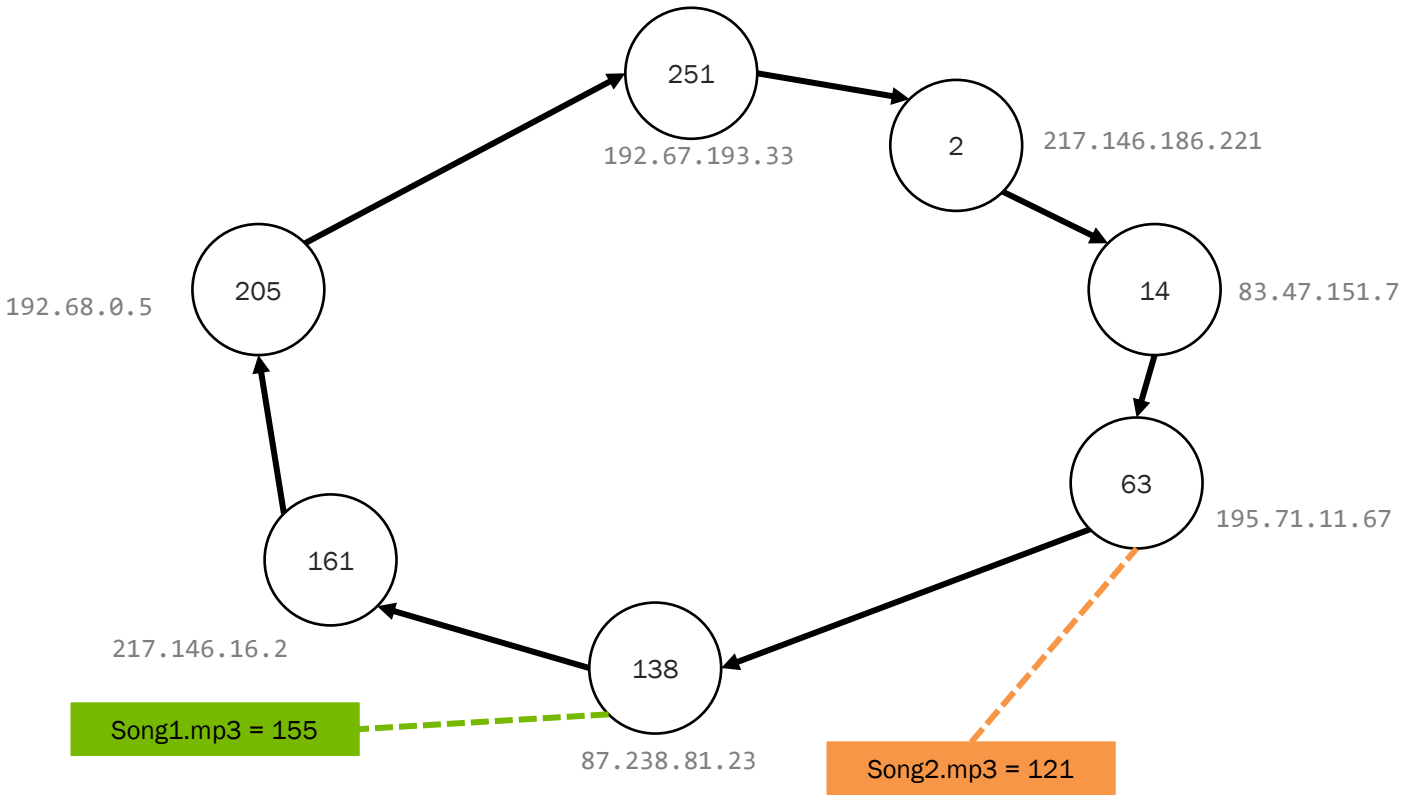


Bildquelle: i.A. an Dunkel et al. [2008:151]

Distributed Hash Tables [II]

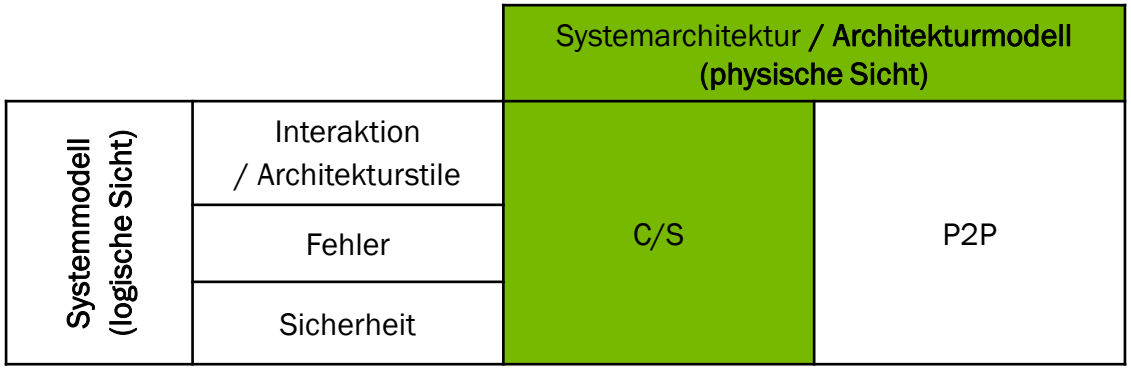
vgl. Dunkel et al. [2008:149f.]

Austritt von Knoten (Nachbarknoten übernimmt und erweitert sein Zuständigkeitsintervall)



Bildquelle: i.A. an Dunkel et al. [2008:151]

Rückblick letzte Vorlesung [I]



Nach dieser Lehrveranstaltung kennen Studierende idealerweise:

- Kernmerkmale und Unterschiede grundsätzlicher Systemarchitekturen verteilter Systeme
- Aspekte einer zentralisierten C/S-Systemarchitektur im Kontext eines geschichteten Architekturstils
- Merkmale verschiedener Interaktionssemantiken in Client-Server-Systemen
- Klassifikationsansätze von Servern nach Zustand, Nachrichtenbearbeitung, Aktivierung
- Variationen von C/S-Systemen innerhalb mehrschichtiger Architekturen
- Ansätze zur Klassifikation von C^+SS^+-Systemen nach Rolle sowie Verkettung von Servern in geschichteten Architekturen
- Vergleichskriterien zur Architekturauswahl

✓ Studierende lernen vertiefende Aspekte zur Bewertung von Systemarchitekturen verteilter Systeme, welche das zu Grunde liegende logische Systemmodell und den Architekturstil physisch unterschiedlich implementieren können.

Entwurfsaspekte: Kriterien zur Architekturauswahl und –bewertung [I]

vgl. Dunkel et al. [2008: 213ff.]

		Systemarchitektur / **Architekturmodell (physische Sicht)**	
Systemmodell (logische Sicht)	Interaktion / Architekturstile	C/S	P2P
	Fehler		
	Sicherheit		

- Ausgangspunkt: Vielfalt an existenten Architekturen
- Potenzielle Fragen: Welche ist wann einzusetzen?
- Problem: Entwickler entscheiden oft
  - » Im Rahmen des Bekannten
  - » Zu subjektiv

- Notwendigkeit von „objektiven“ Vergleichskriterien als Entscheidungsgrundlage
- Einflussfaktoren:
  - » Anforderungen aus dem Softwarelebenszyklus
  - » Anforderungen der Anwendung

Entwurfsaspekte: Kriterien zur Architekturauswahl und –bewertung [II]

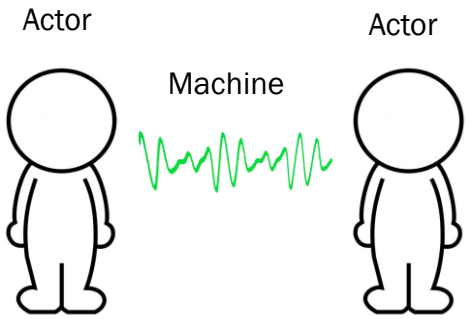
vgl. Dunkel et al. [2008: 213ff.]

- Anforderungen aus dem Softwarelebenszyklus:
  - » Analyse: Identifikation relevanter Objekte der Problemwelt sowie der dort ablaufenden Prozesse
  - » Design: Lösungsmodell
    - Beschreibung der Struktur des zu realisierenden Systems
    - Formale Darstellung
  - » Implementierung: Übersetzung des Lösungsmodells in Code (modular, lose gekoppelt)
  - » Test: Unit, Modul, Integration, …
  - » Betrieb / Wartung
  - » Übergreifend: Art/Umfeld
    - Software-Engineering
    - (Projekt-)management

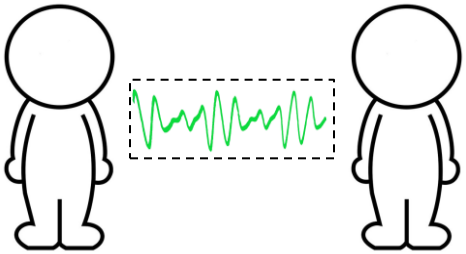
✓ Immer zu berücksichtigen: ethische Aspekte (vgl. ff.; value-sensitive Design)

Ebenen des ethischen Diskurses („Computer-Ethik“)

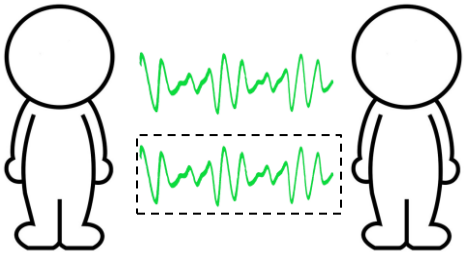
vgl. Spiekermann [2016:166]



1. Actor‘s value
Fokussiert die ethischen Konsequenzen einer gegebenen Technologie für die beteiligten Akteure



2. Value-sensitive design
Einbettung von ethischen Aspekten und Werten in den gesamten (SW-)Entwicklungsprozess



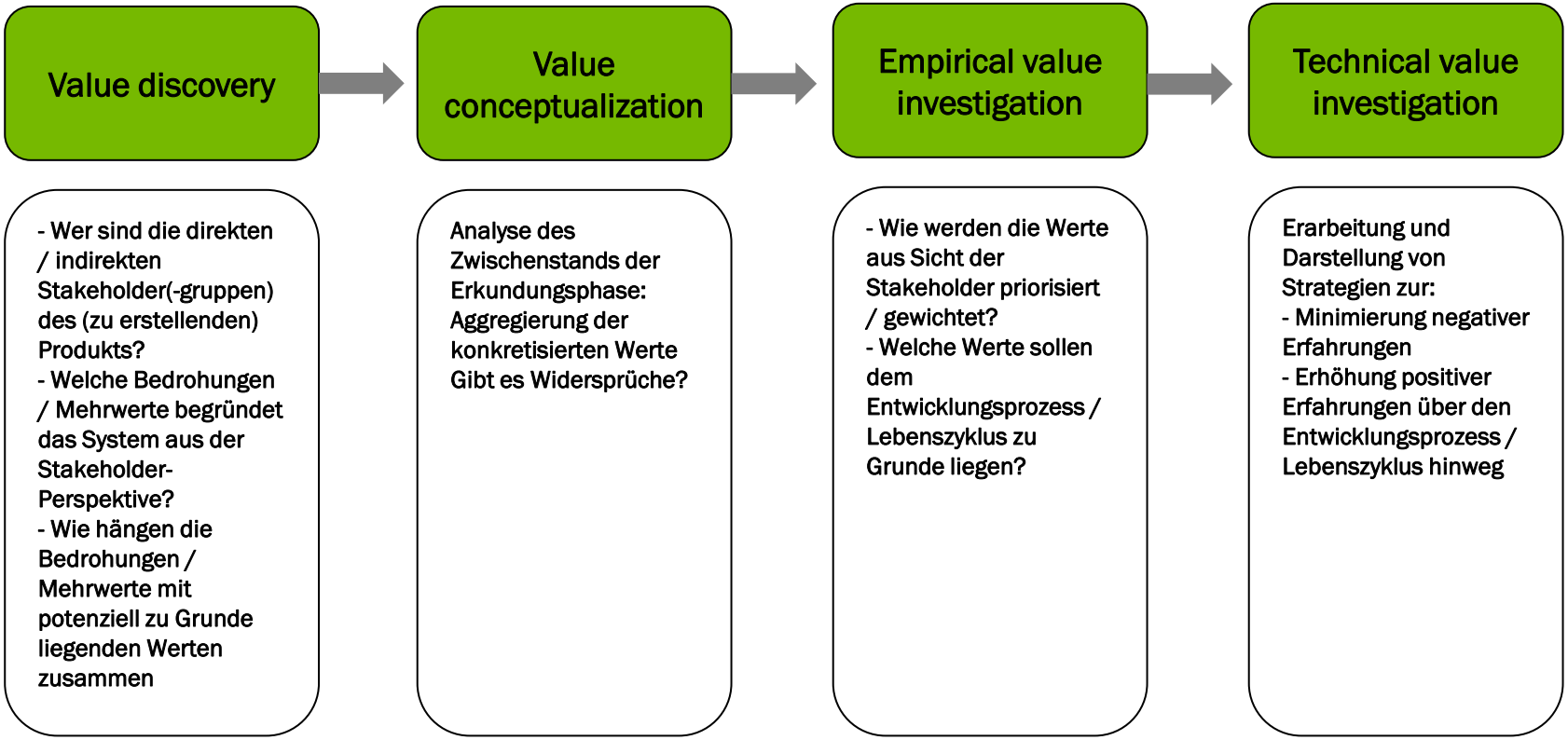
3. Value-sensitive alternative:
Konstante Suche / Entwicklung nach besseren Alternativen zu 2)

Bildquellen [letzter Zugriff: 4 XII 17]:
- Männchen: https://www.schulbilder.org/malvorlage-maennchen-leeres-gesicht-dm25691.jpg
- Signal: http://weclipart.com/gimg/69D028FFD933E018/signal-clipart-angreen.jpg

2.) Value-sensitive Design

vgl. Spiekermann [2016:166ff.]

Methodologie:



Bildquelle: eigene Darstellung in Anlehnung an Spiekermann [2016:168]

Beispiel: Ganzkörperscanner [I]

vgl. Spiekermann [2016:166ff.]

- Ausgangspunkt: Veränderte Sicherheitslage bei Passagierkontrollen an Flughäfen
- Idee: Einführung von Ganzkörperscannern bei Passagierkontrolle

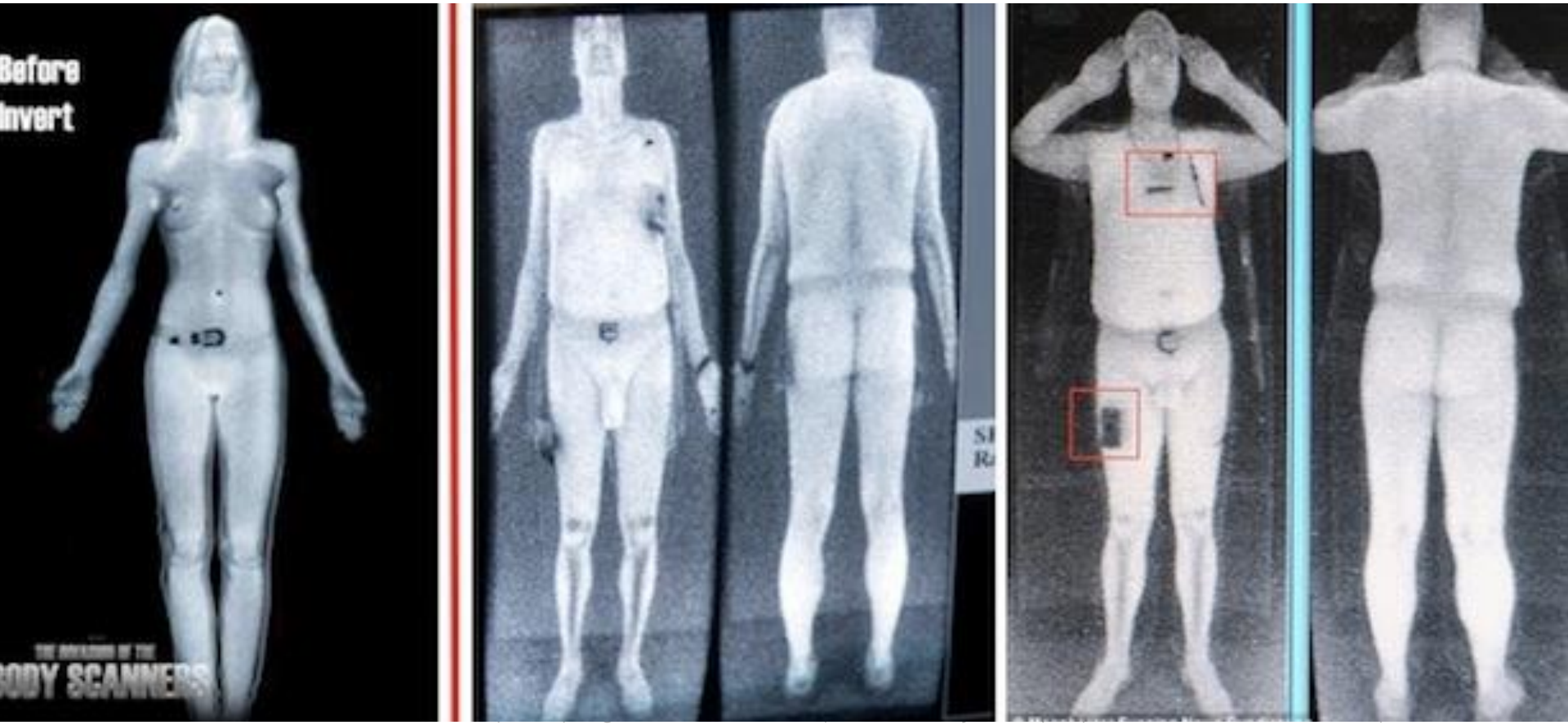
- Actor‘s value:
  - Alle: Erhöhung der Sicherheit
  - Flughafen:
    - Erhöhung der Trefferquote
    - Einsparung Personal
  - Passagiere:
    - Datenschutzbedenken, aber
    - Haben eigentlich keine Mitbestimmung
  - Hersteller der Bodyscanner (IT-System:
    - Potenzieller Absatzmarkt i.A. der entwickelten Technologie

Beispiel: Ganzkörperscanner [II]

vgl. Spiekermann [2016:166ff.]

Ohne value-sensitive design:

- ✓ Risiko der Bedrohung der Privatsphäre
- ✓ Unangenehme Situation in Arena „Passagierkontrolle“
- ✓ Variation bei IT-System: Abstraktion der Repräsentation von Menschen

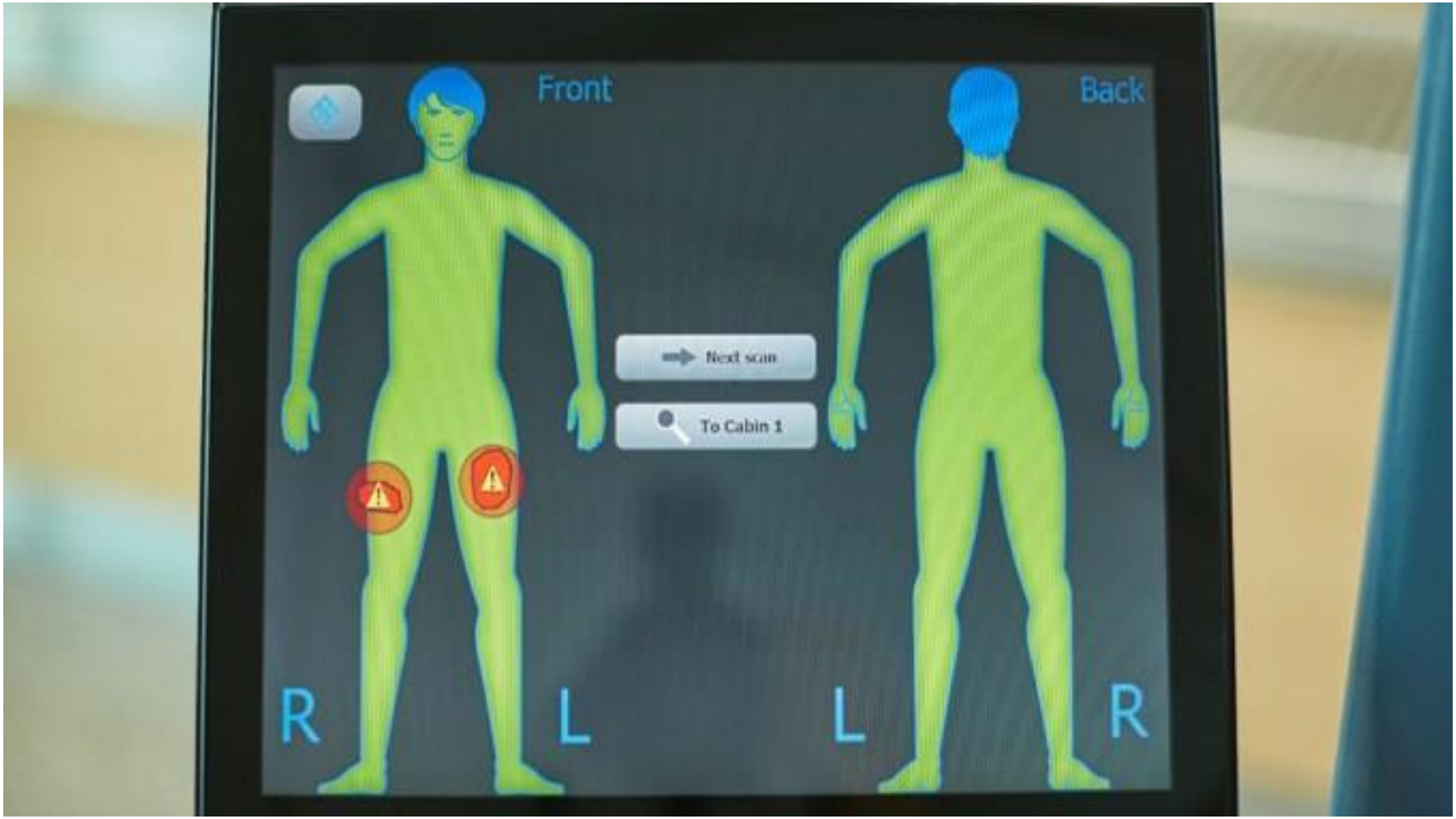


Bildquelle: https://images.fastcompany.com/upload/bodyscans.jpg [letzter Zugriff: 4 XII 17]

Beispiel: Ganzkörperscanner [III]

vgl. Spiekermann [2016:166ff.]

Value-sensitive design: Schutz der Privatsphäre

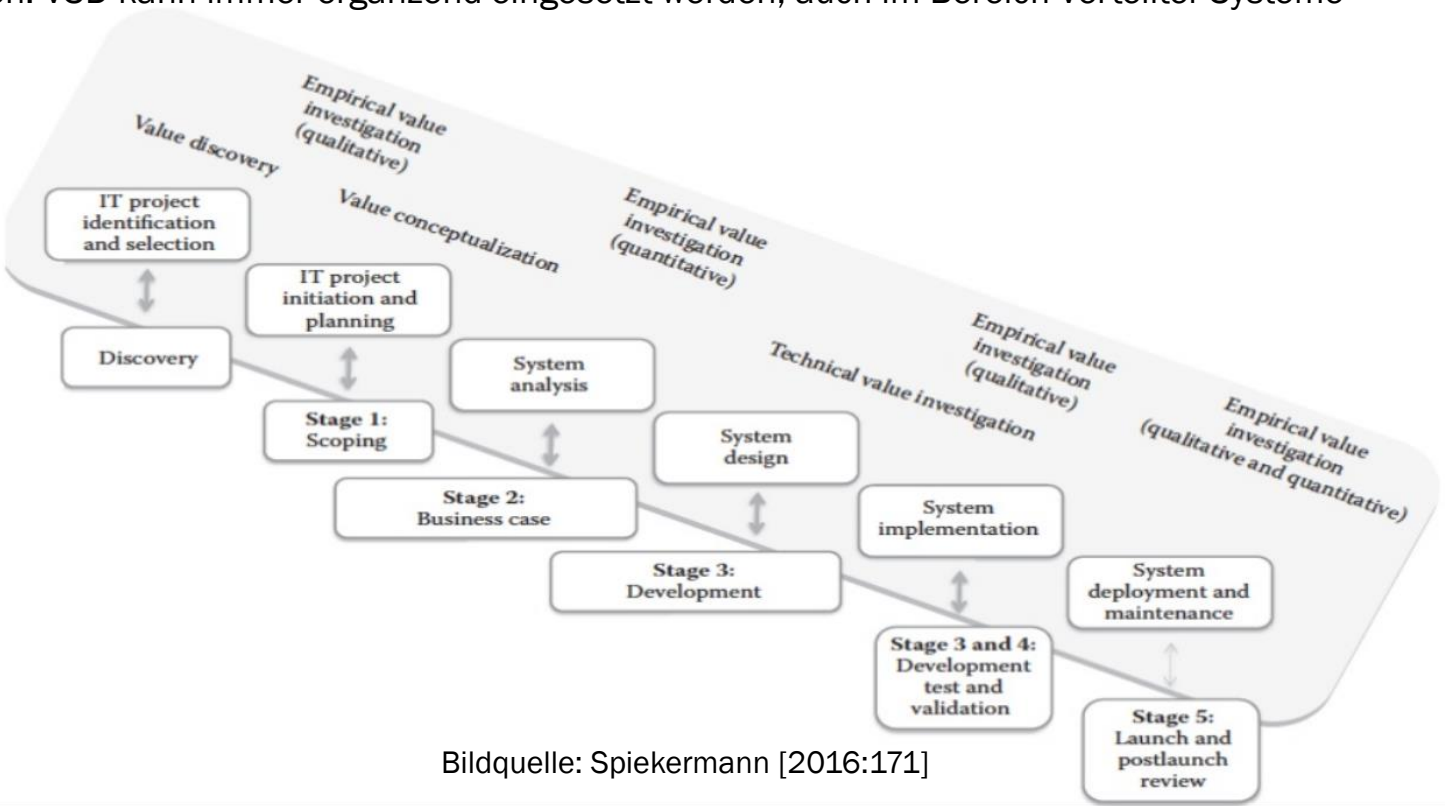


Bildquelle: https://i1.wp.com/www.passengerselfservice.com/wp-content/uploads/Helsinki-body-scanner.jpg?resize=678%2C381 [letzter Zugriff: 4 XII 17]

Value-sensitive Design (VSD) in Kombination mit einem Vorgehensmodell im Software Development Life Cycle (SDLC)

- Kontinuierliche Ergänzung des SDLC durch Meta-Prozesse des VSD
  - » Empirische Investigation auf Basis der vorherigen Phase:
    - Value damns: Funktionalitäten, Praktiken, die zu Werten im Widerspruch stehen
    - Value flows: Kontextfaktoren, welche die Entfaltung von Werten ermöglichen
  - » Technische Perspektive: Einfluss einer ethischen werte-basierten Metrik bei Entscheidungen (z.B. bei der Wahl einer Design-Alternative)

✓ Sie werden im Laufe Ihres Studiums mehrere Vorgehensmodelle im Software-Engineering kennen lernen. VSD kann immer ergänzend eingesetzt werden, auch im Bereich Verteilter Systeme



Bildquelle: Spiekermann [2016:171]

*/

Mapping: von Bedrohungen / Vorteilen zu Werten

Beispiel: Ganzkörperscanner

	Negativ (Werte)	Positiv (Werte)
Effekte (gesellschaftlich): Stakeholder: Kund(inn)en	- Arbeitslosigkeit durch überflüssige MA (soziale Stabilität, Sicherheit) - Servicequalität	Schnellerer Check-In (Komfort)
Effekte (intern): Unternehmen / Aktionäre	Schlechte Arbeitseinstellung (Identität, Vertrauen, Zugehörigkeitsgefühl)	Mehr Profit (finanzieller Wert)

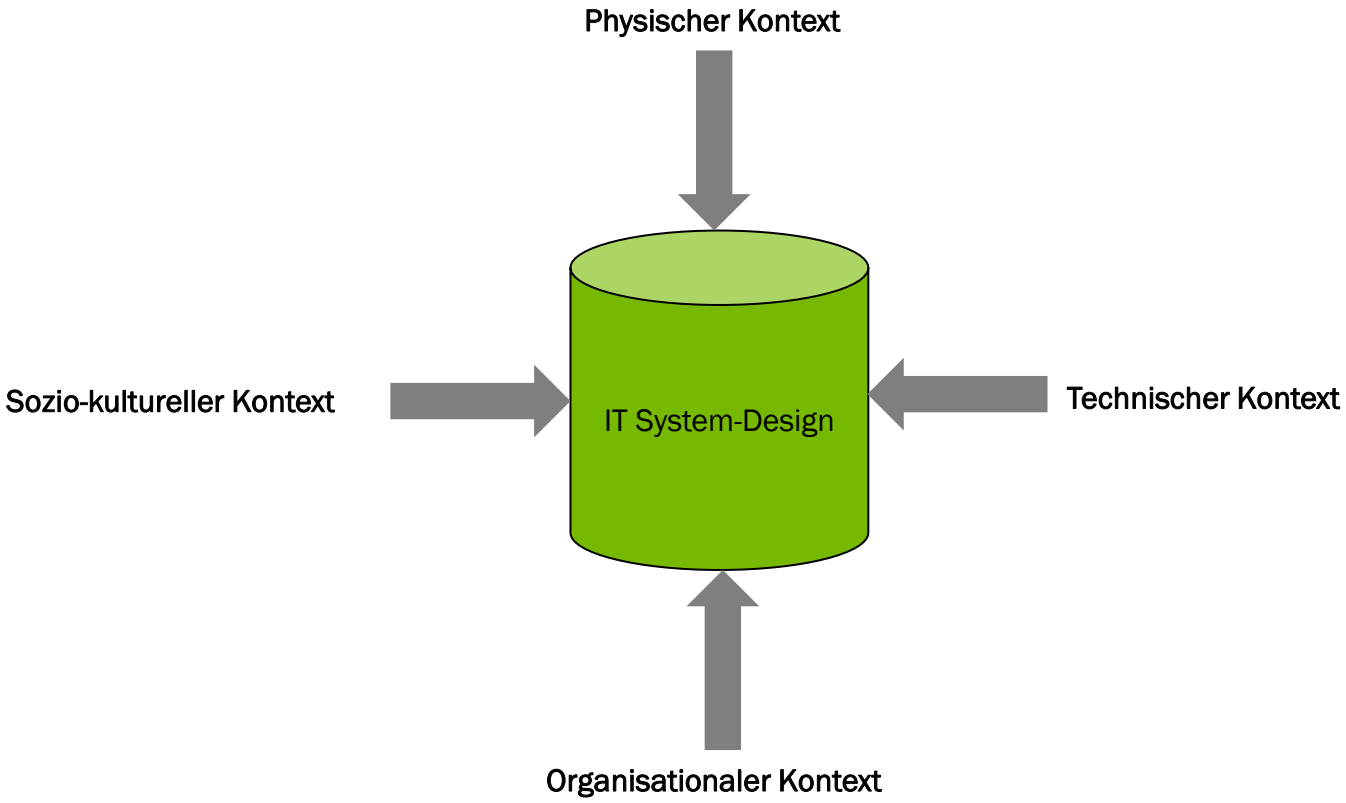
- ✓ Im Anschluss können Werte gewichtet werden und in Folgeaktivitäten als werte-basierte Einflussgrößen münden.

Kontext im IT-System-Design [I]

vgl. Spiekermann [2016:214ff.]

Traditionelle Sicht auf Kontext-Analyse:

- Der Humane / physische Kontext beeinflusst die Anforderungen des IT-Systems
- Es kann ein optimaler „fit“ zwischen IT-System und Umwelt erreicht werden

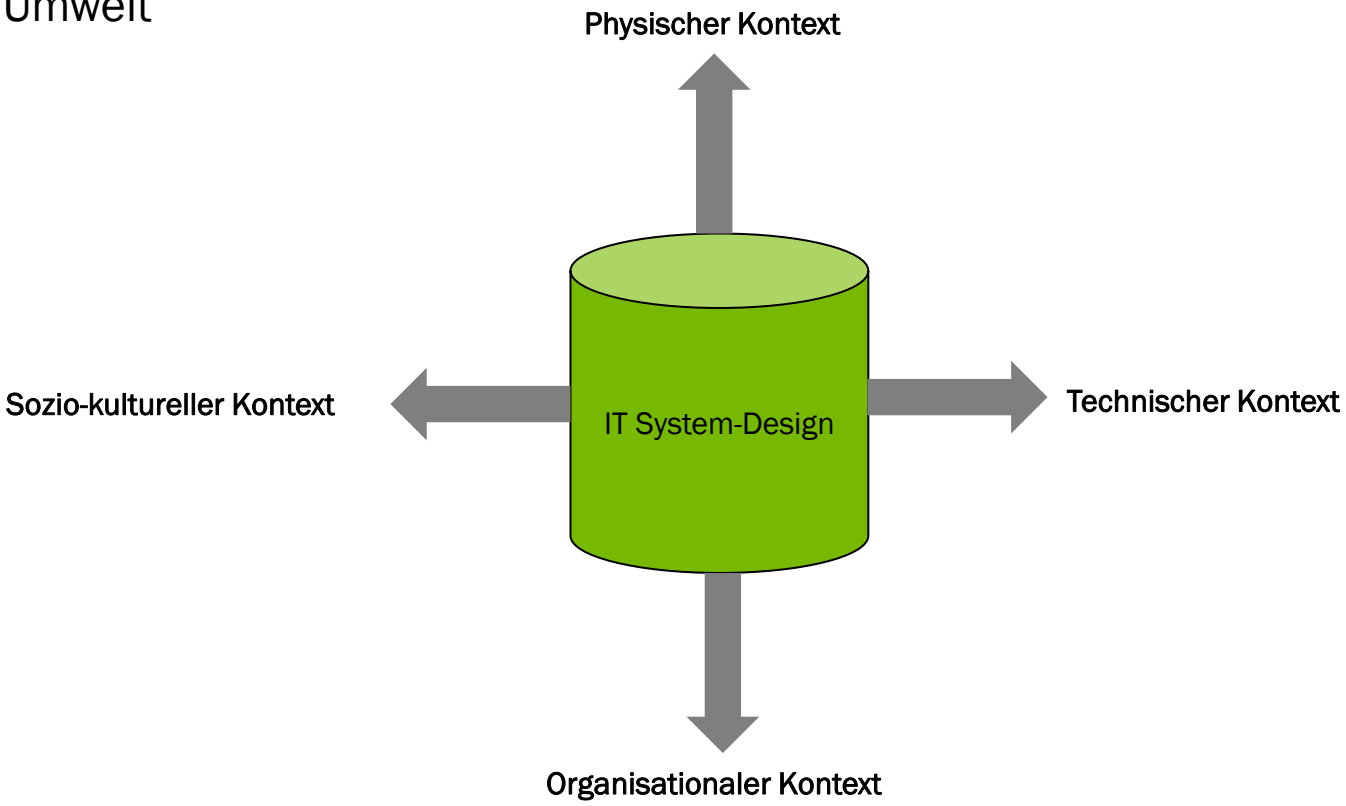


Kontext im IT-System-Design [II]

vgl. Spiekermann [2016:214ff.]

Ethische Sicht auf Kontext-Analyse:

- Das IT-System hat einen Einfluss auf seinen Kontext
- Es erzeugt Nutzer-Gefühle und -Erfahrungen im Kontext
- Repräsentiert die Präsenz von Werten zur langfristigen Beeinflussung der Umwelt



Entwurfsaspekte: Kriterien zur Architekturauswahl und –bewertung [III]

vgl. Dunkel et al. [2008: 213ff.]

- Anforderungen der Anwendung:
  - Grad an Interaktivität: Hohe Interaktivität durch kürzere Reaktionszeiten, z.B. durch Ausführung von clientseitigem Code
  - Zahl der Teilnehmer:
    - B2C: hoch (erfordert hoch skalierbare Architekturen)
    - B2B: niedrig (geringer Bedarf an mehrschichtigen Architekturen)
  - Ressourcenbedarf
    - Hoch (Rechenzeit, Speicher): Grid, P2P -> mit der Teilnehmerzahl steigt die Ressourcenanzahl an, was bei C/S-Architekturen nicht der Fall ist
  - Dynamik (Antizipation von Umweltbedingungen)
  - Robustheit (Fehler, Ausfallsicherheit)
  - Anwendungsgebiet

Weitere Entwurfsaspekte verteilter Systeme

- Leistung
- Quality of Service
- Replikation / Caching
- Zuverlässigkeit

Entwurfsaspekt: Leistung [I]

`Geringe Antwortzeiten`

- Einflussfaktoren:
  - Serverleistung
  - Serverauslastung
  - Verzögerungen in den Softwarekomponenten (OS, Middleware)
- Maßnahmen:
  - Reduzierung der Softwareschichten
  - Reduzierung der Datenmenge

Entwurfsaspekt: Leistung [II]

`Hoher Durchsatz:`
`Geschwindigkeit mit der Rechenarbeit erledigt wird`

- Einflussfaktoren
  - » Verarbeitungsgeschwindigkeit (Client, Server)
  - » Datenübertragungs-geschwindigkeiten
    - zwischen Softwareschichten
    - im Netzwerk
- Maßnahmen:
  - » Reduzierung der Softwareschichten
  - » Reduzierung der Datenmenge

Entwurfsaspekt: Leistung [III]

`Ausgleich von Rechenauslastung`

- Ziel:
  - Prozesse sollten gleichzeitig ausgeführt werden können, ohne um Rechenressourcen (Prozessor-, Speicher-, Netzwerkkapazität) konkurrieren zu müssen
- Maßnahmen:
  - Verwendung von mobilem Code/Applets zur Lastreduzierung auf dem Server
  - Bereitstellung eines Dienstes durch mehrere Server (Mehrere Hosts in einer Domäne)
  - Programme, die laufende Prozesse auf weniger ausgelastete Hosts verschieben

Entwurfsaspekt: Quality of Service

- Dienstgüte:
  - Zuverlässigkeit
  - Sicherheit
  - Leistung
  - Anpassbarkeit in wechselnden Systemkonfigurationen
  - Ressourcenverfügbarkeit

- Dimensionen:
  - Betriebssysteme
  - Netzwerke

Entwurfsaspekt: Caching/Replikation

- Implementierung von Maßnahmen zur Aktualisierung der Cache-Dateien bei sich verändernder Ressource auf dem Server
  - Web-Caching-Protokoll
    - Antworten eines Servers werden im Cache des Browsers/Proxy-Servers gespeichert und mit Zeitdaten versehen
      - Serverzeit
      - Ablaufdatum
    - Bei Client-Anforderung sucht Browser/Proxy nach potenzieller Antwort und deren Haltbarkeitsdatum
    - Neue Serveranfrage (Validierung, Caching) wenn Ablaufdatum überschritten

Entwurfsaspekt: Zuverlässigkeit [I]

- Dimensionen der Zuverlässigkeit:
  - » Korrektheit
  - » Sicherheit: sensible Ressourcen sollen nur auf effektiv vor Angriffen geschützten Computern untergebracht sein
  - » Fehlertoleranz: Applikationen sollen trotz des Auftretens von Fehlern korrekt (weiter)arbeiten

Entwurfsaspekt: Zuverlässigkeit [II]

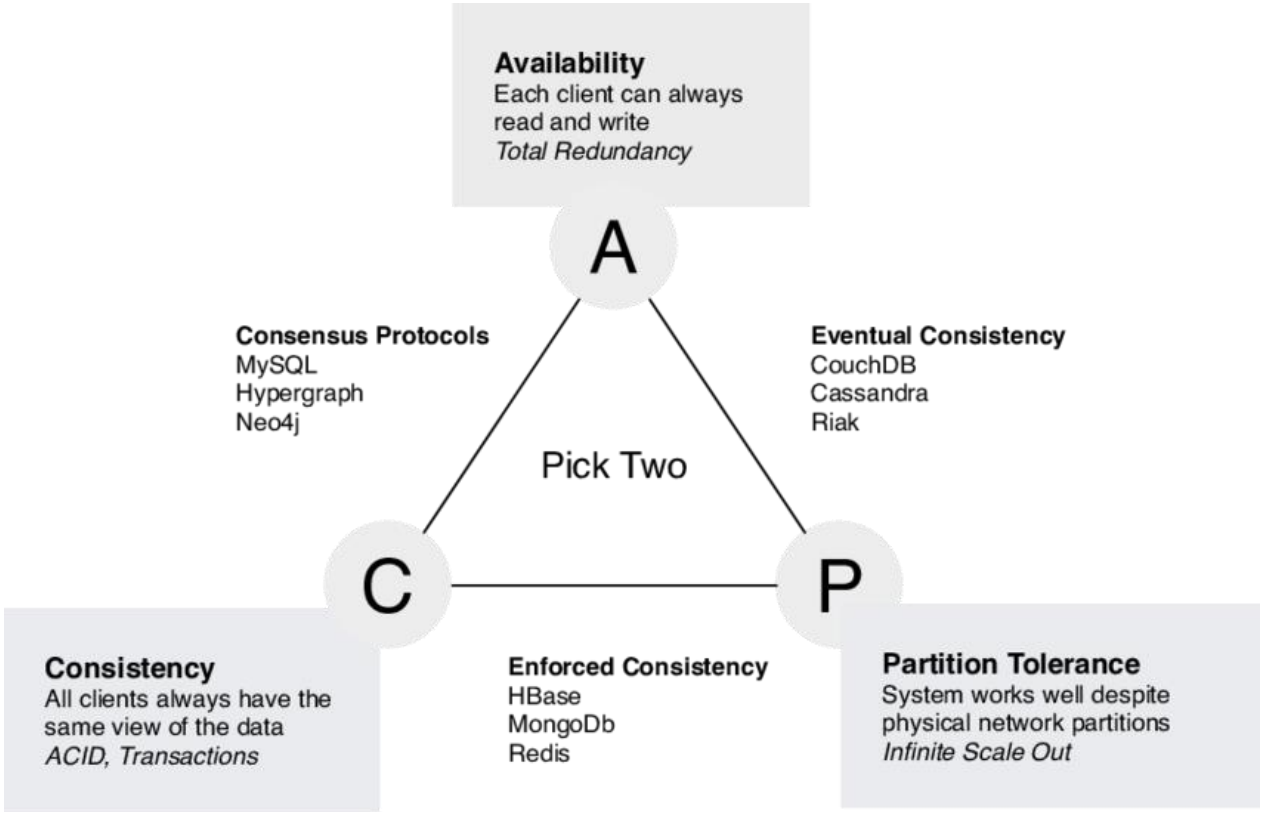
`Fehlertoleranz`

- Maßnahmen: Redundanz
  - » Replikation: Bereitstellung mehrerer Ressourcen/Repliken von Datenelementen auf unterschiedlichen Computern
  - » Wiederholung der Nachrichtenübermittlung
- Konsequenzen: Kosten- und Zeitaufwand durch Verwendung mehrerer Computer/Kommunikationspfade zur Ausführung

✓ Ziel kann lediglich eingeschränkt erreicht werden!

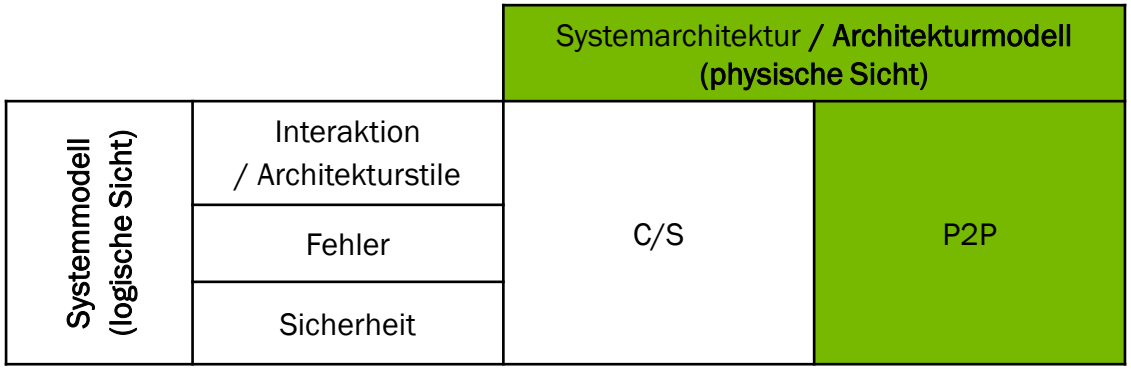
CAP-Theorem / Brewers Theorem

In einem VSYS ist es unmöglich, gleichzeitig die drei Eigenschaften **C**onsistency (Konsistenz), **A**vailability (Verfügbarkeit) und **P**artition Tolerance (Ausfalltoleranz) zu garantieren.



https://www.researchgate.net/profile/Sven_Rizzotti/publication/221462089/figure/fig4/AS:393977489575939@1470942961359/Figure-4-CAP-Theorem.png

Danke. Lernziele erreicht?



Nach dieser Lehrveranstaltung kennen Studierende idealerweise:

- Aspekte einer dezentralisierten Systemarchitektur
- Merkmale, Eigenschaften von P2P-Architekturen
- Möglichkeiten zur Abgrenzung von P2P zu C/S-Architekturen
- Klassifikationsansätze von P2P-Architekturen nach Zentralisierung, Rollenverteilung, Strukturierung
- Charakteristika einer logischen Netzstruktur (Overlay)
- Ausgewählte Topologien und Algorithmen im Kontext strukturierter P2P Overlay-Netze
- Eine Möglichkeit zur Anordnung von Prozessen mittels Distributed Hash Tables (DHT)
- Entwurfsaspekte und Vergleichskriterien zur Architekturauswahl und –bewertung
- Brewer‘ Theorem (CAP-Theorem)
- Ausgewählte Beispiele zu den o.g. Themen / Inhalten

✓ Studierende lernen vertiefende Aspekte zur Bewertung von Systemarchitekturen verteilter Systeme, welche das zu Grunde liegende logische Systemmodell und den Architekturstil physisch unterschiedlich implementieren können.

Quellen

Adar, E.; Hubermann, B. A. (2000) *Free Riding on Gnutella*, First Monday, Vol. 5(10)

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

Dunkel, J; Eberhart, A.; Fischer, S.; Kleiner, C. ; Koschel, A. (2008) *Systemarchitekturen für Verteilte Anwendungen*; München: Hanser.

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.

Spiekermann, S. (2016) *Ethical IT Innovation – A Value-Based System Design Approach*. Boca Raton: CRC Press / Taylor & Francis Group.

Stein, E. (2004) *Taschenbuch Rechnernetze und Internet*. München, Wien: Fachbuchverlag Leipzig im Carl Hanser Verlag

Steinmetz, R.; Wehrle, K. (2004) *Peer-to-Peer- Networking and Computing*. Informatik-Spektrum, Vol. 27(1), pp. 51-54

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

Tirado, J. M.; Higuero, D.; Isaila, F.; Carretero, J.; Iamnitchi, A. (2010) *Affinity P2P: A self-organizing content-based locality-aware collaborative peer-to-peer network*, Computer Networks, Vol. 54(12), pp. 2056-2070

Vu, Q. H.; Lupu, M.; Ooi, B. C. (2009) *Peer-to-Peer Computing: Principles and Applications*; Berlin, Heidelberg: Springer