AI-B.41: Verteilte Systeme

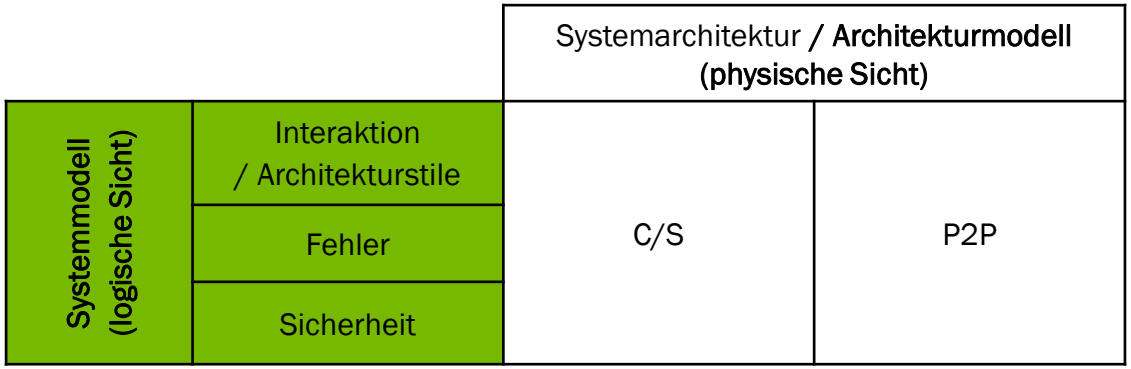
- Lecture Notes [SL] -

III. Systemarchitekturen [I]
- C/S (Client-Server-Architekturen und Variationen) -

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

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

Rückblick letzte Vorlesung [I]



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…

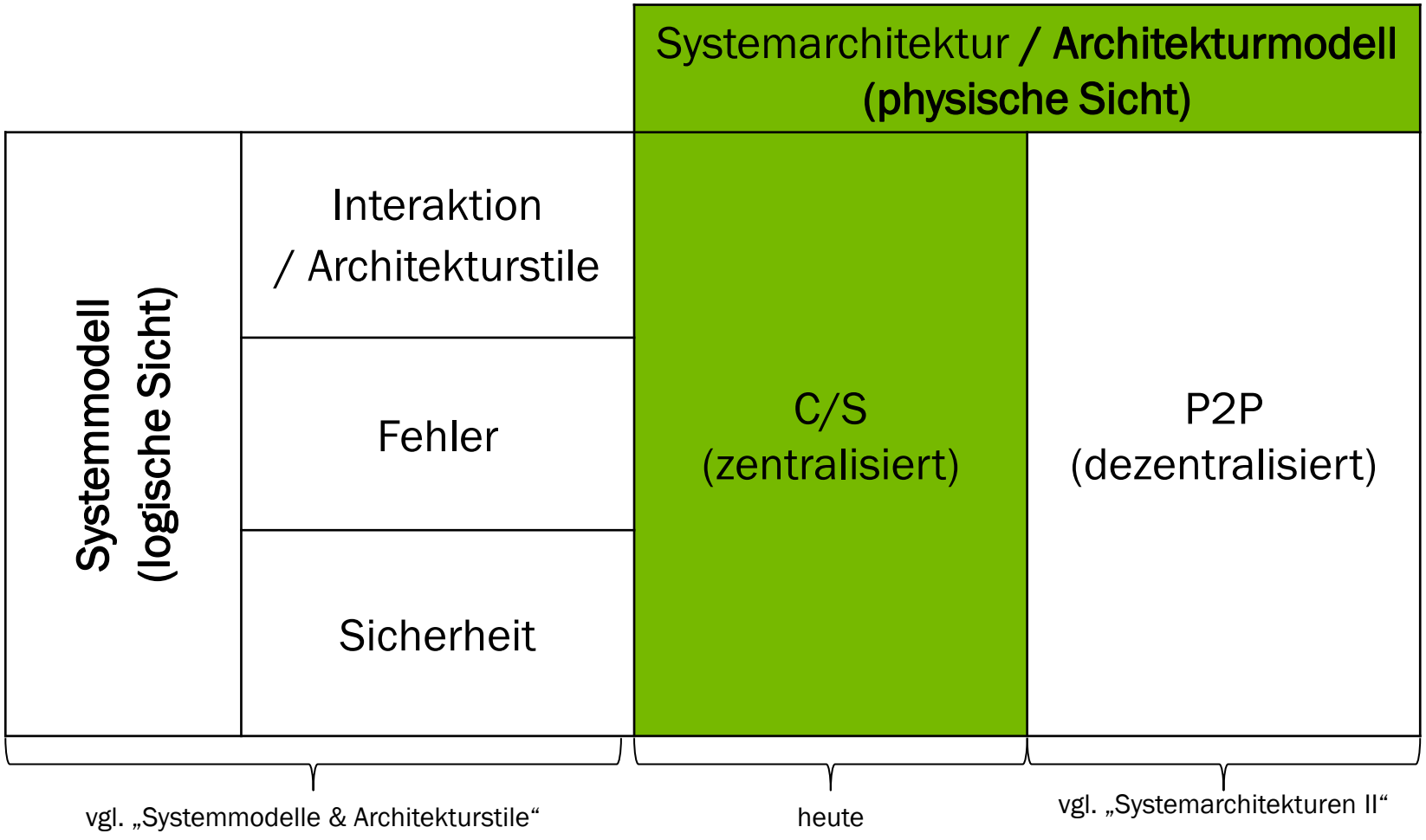
Rückblick letzte Vorlesung [II]

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

Modellperspektiven verteilter Systeme



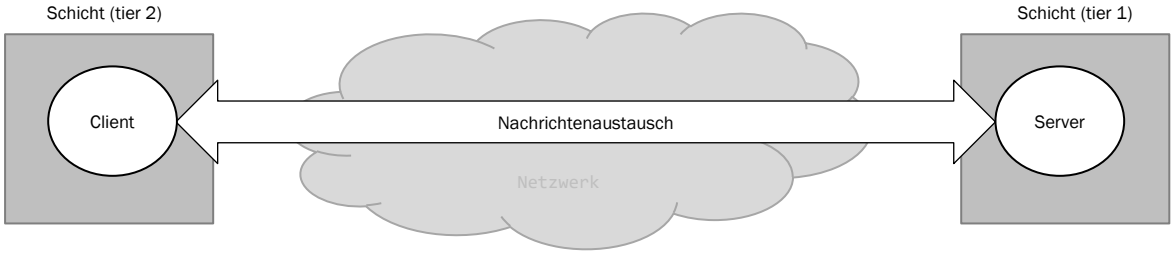
Lernziele

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.

Systemarchitektur: Client-Server (C/S)

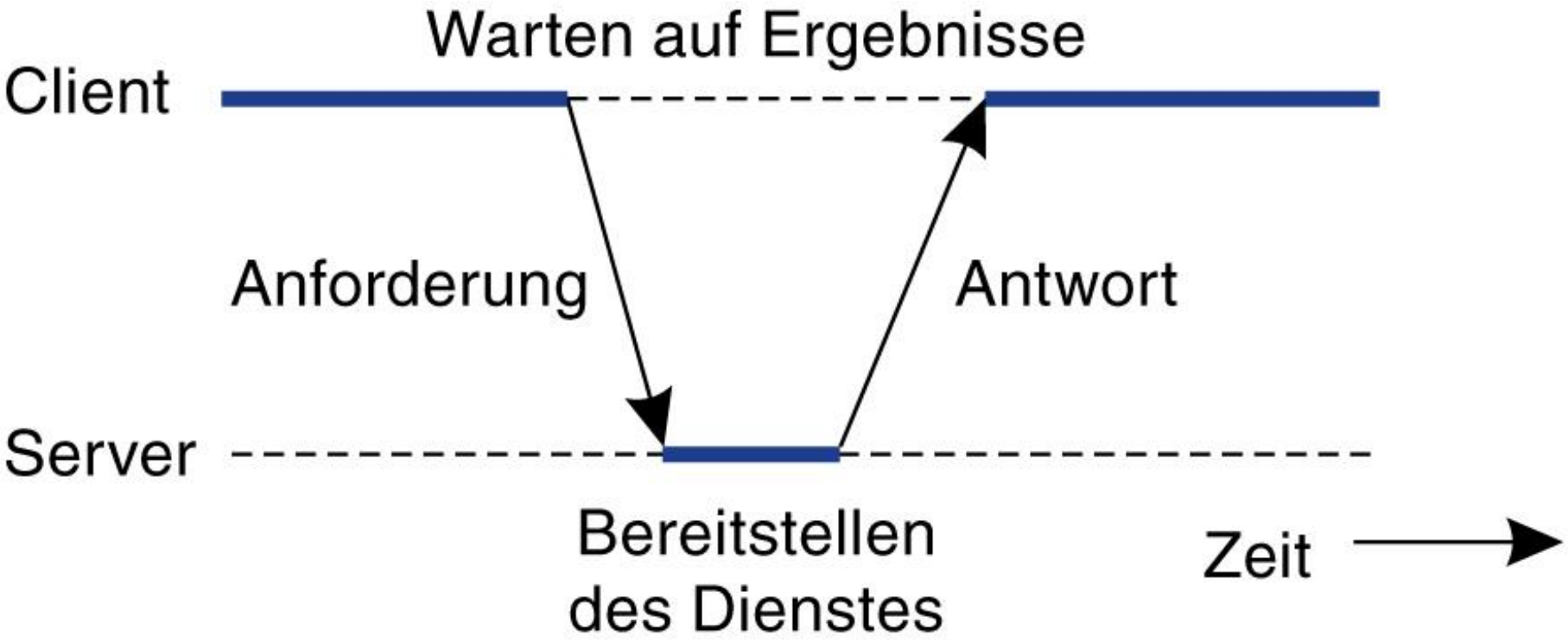


Zur Erinnerung (vgl. Lecture Notes „Einführung“, „Systemmodelle & Architekturen):

- Historisch etablierte Architektur verteilter Systeme
- Client und Server sind Prozesse
- Ein Prozess kann gleichzeitig Client und Server sein (je nach Rolle innerhalb einer Anforderung)
- Clients sind aktiv, Server sind passiv
- Clients werden nur für die Dauer der Applikation ausgeführt, deren Bestandteil sie sind
- Server
  - » werden kontinuierlich ausgeführt
  - » können Clients anderer Server sein, z.B.:
    - Webserver -> Dateiserver
    - Webserver -> DNS, Suchmaschinen
      - – Antwort auf Client-Anfragen
      - – Ausführung eigener Clients zur Anfrage an andere Webserver

2-tier

Zusammenarbeit zwischen einem Client und einem Server:



Quelle: Tanenbaum & van Steen [2008:55]

Welche Auswirkungen hat diese Form der Zusammenarbeit?
Gibt es noch denkbare Alternativen?

Interaktionssemantik [I]

zu Grunde liegende Frage:

„Wie koordinieren sich Client und Server beim Ablauf einer Interaktion?“

Antwort:

- **Alternative 1 (wie zuvor dargestellt):** Client wartet nach Absenden einer Anforderung an Server auf dessen Rückantwort und blockiert bis zum Erhalt Ressourcen (Aktives Warten des Clients)

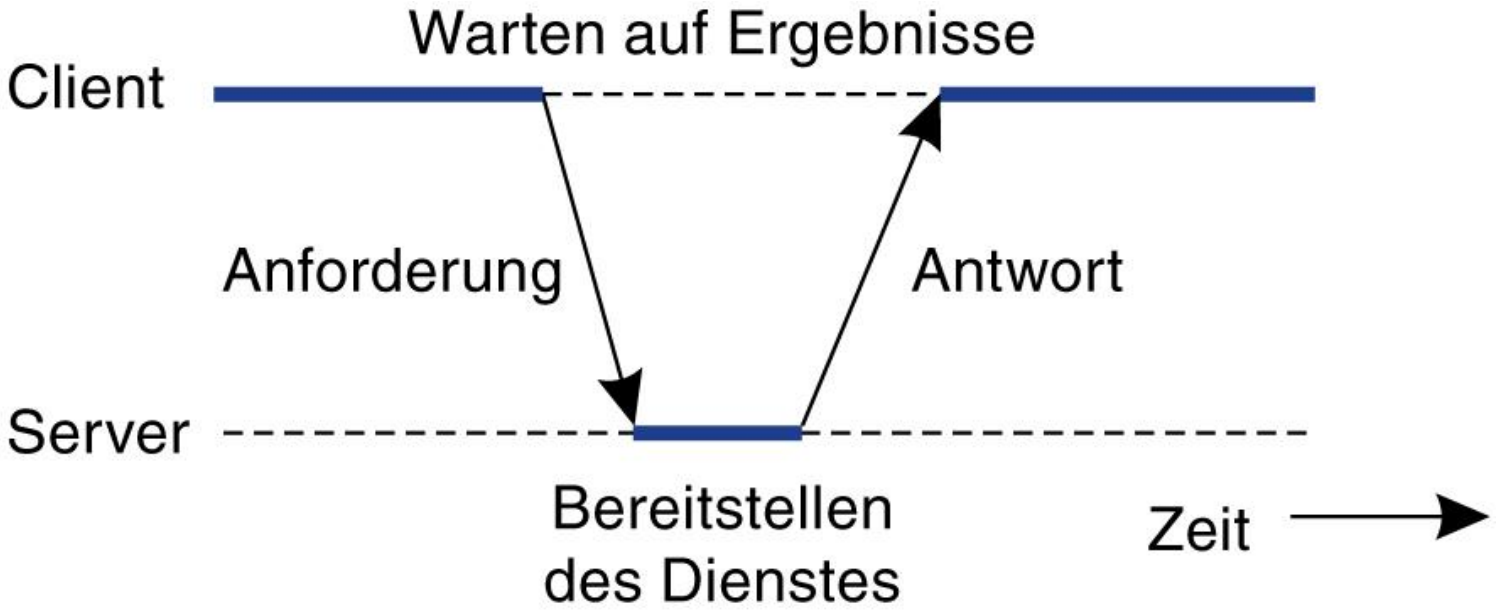
- **Alternative 2**: Client sendet Anforderung an Server und arbeitet danach sofort weiter (nach dem Senden kann Client unterschiedlich interagieren)

Interaktionssemantik [II]

zu Grunde liegende Frage: *„Wie koordinieren sich Client und Server beim Ablauf einer Interaktion?“*

1. Alternative: Blockierend (synchron)

- Client wartet nach Absenden einer Anforderung an Server auf dessen Rückantwort und blockiert bis zum Erhalt Ressourcen
- Aktives Warten des Clients



Quelle: Tanenbaum & van Steen [2008:55]

Interaktionssemantik [III]

zu Grunde liegende Frage: *„Wie koordinieren sich Client und Server beim Ablauf einer Interaktion?“*

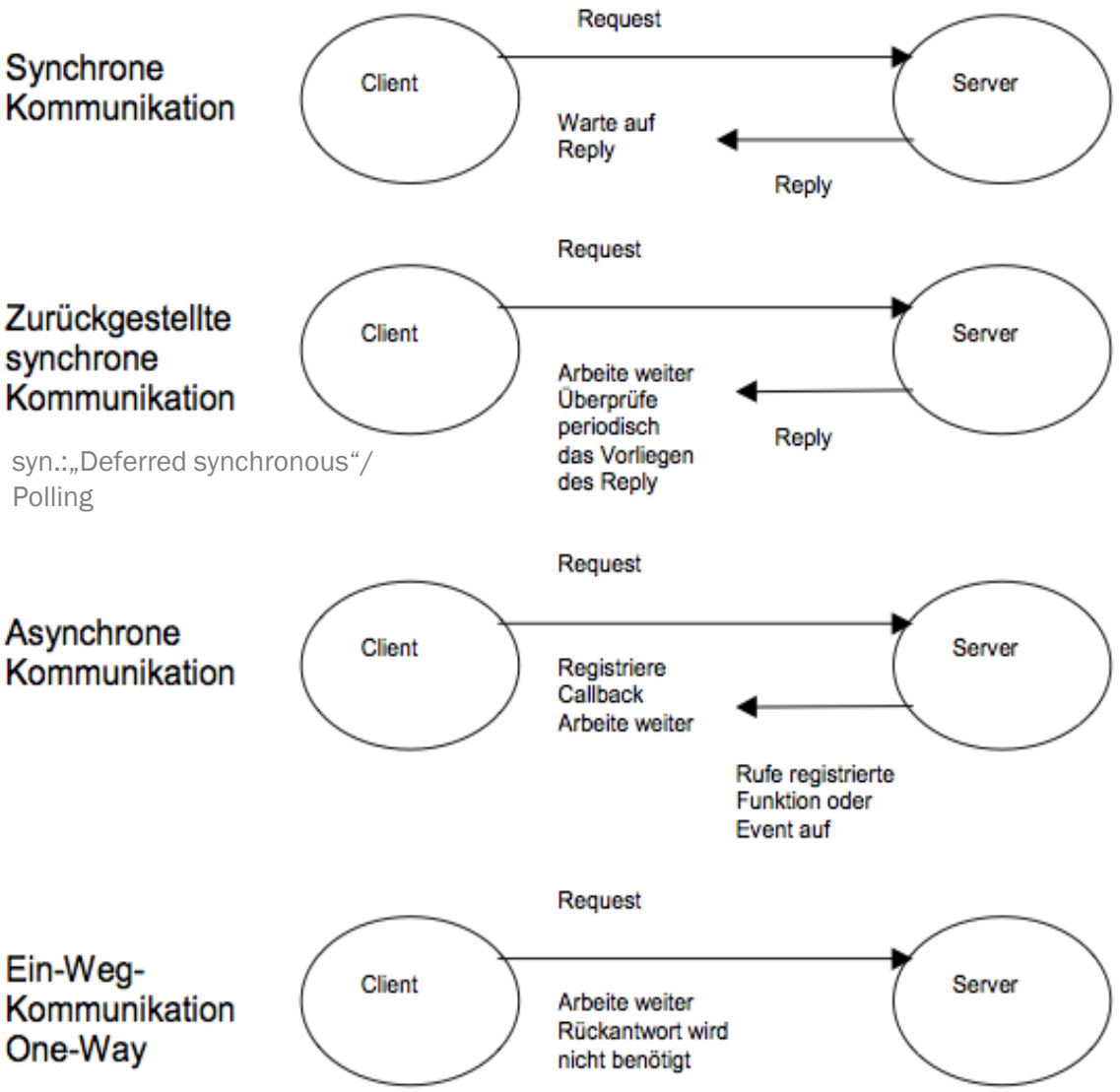
2. Alternative: Nicht blockierend (asynchron)

- Client sendet Anforderung an Server und arbeitet danach sofort weiter
- Nach dem Senden kann Client unterschiedlich interagieren:
  - „deferred synchronous“: Client überprüft sukzessive Eingangswarteschleife auf Rückantwort des Servers
  - „callback“: Registrierung von Rückrufen des Servers (Funktionseingangspunkte, Ereignisse), die nach Eintreffen beim Client aktiviert werden
  - „one-way“: Client kümmert sich nicht mehr um Rückantwort

✓ Vorteil: Ressourcennutzung auf Client-Seite ist verbessert, da kein Prozess durch Warten blockiert wird.

✓ Nachteil: Kontrollkomplexität ist erhöht

Interaktionssemantik [IV]
Überblick



syn.:„Deferred synchronous“/ Polling

Quelle: Bengel [2004:32]

Interaktionssemantik: Fehler [I]

zu Grunde liegende Frage:
Welche Fehler können während der Interaktion auftreten?

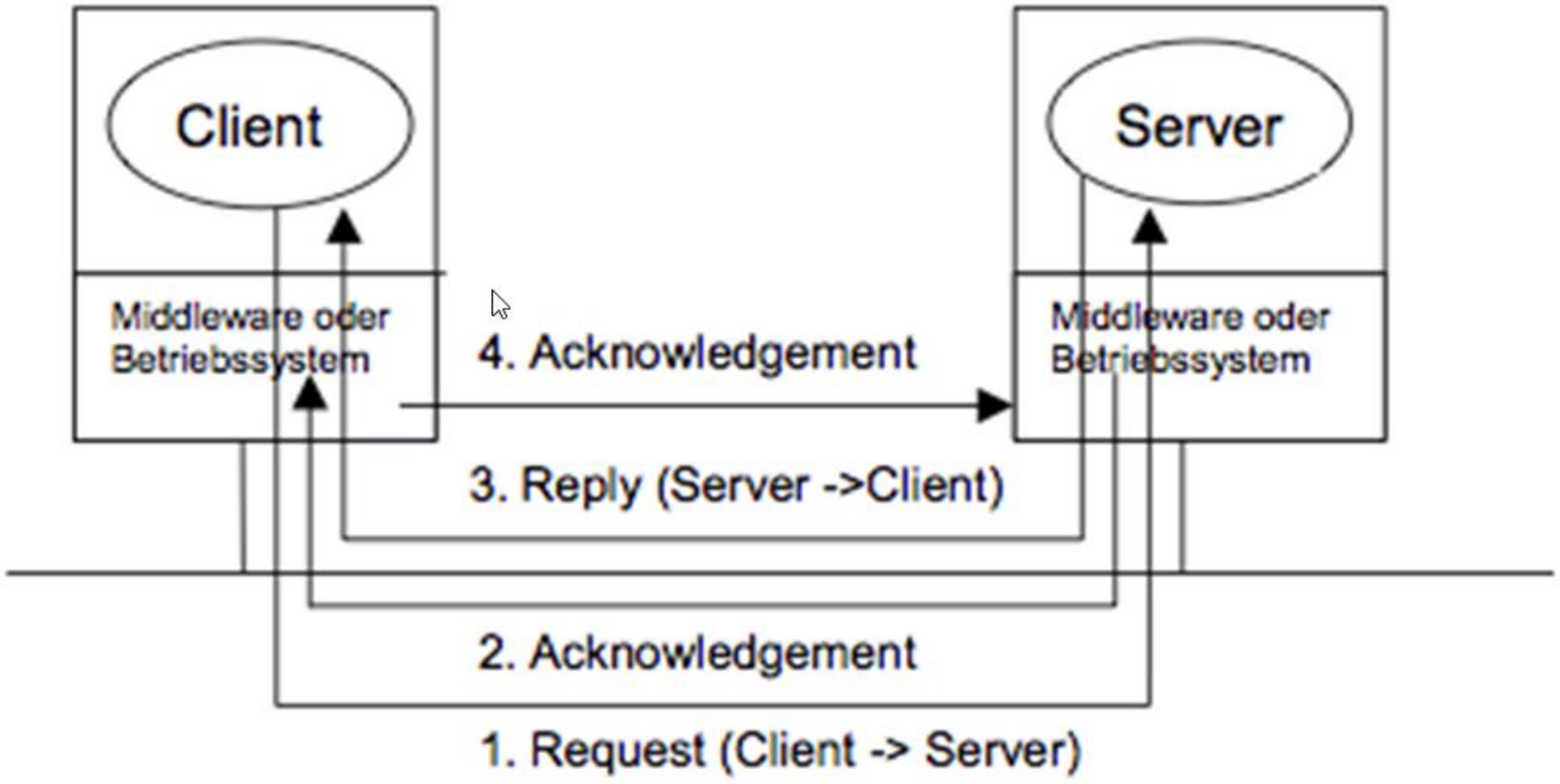
- Ausgangspunkt: Fehler in der Interaktion sind nicht auszuschließen
  - » Anforderung geht verloren/verzögert sich
  - » Rückantwort geht verloren/verzögert sich
  - » Absturz (Client oder Server) während der Interaktion

- Erhöhung der Zuverlässigkeit einer Interaktion durch:
  - » Blockierung von Sendeprozessen und
  - » Quittierung von Nachrichten

Interaktionssemantik: Fehler [II]

zu Grunde liegende Frage: *„Wie kann die Nachrichtenübertragung zuverlässiger gestaltet werden?“*

Individuell quittierte Nachrichten

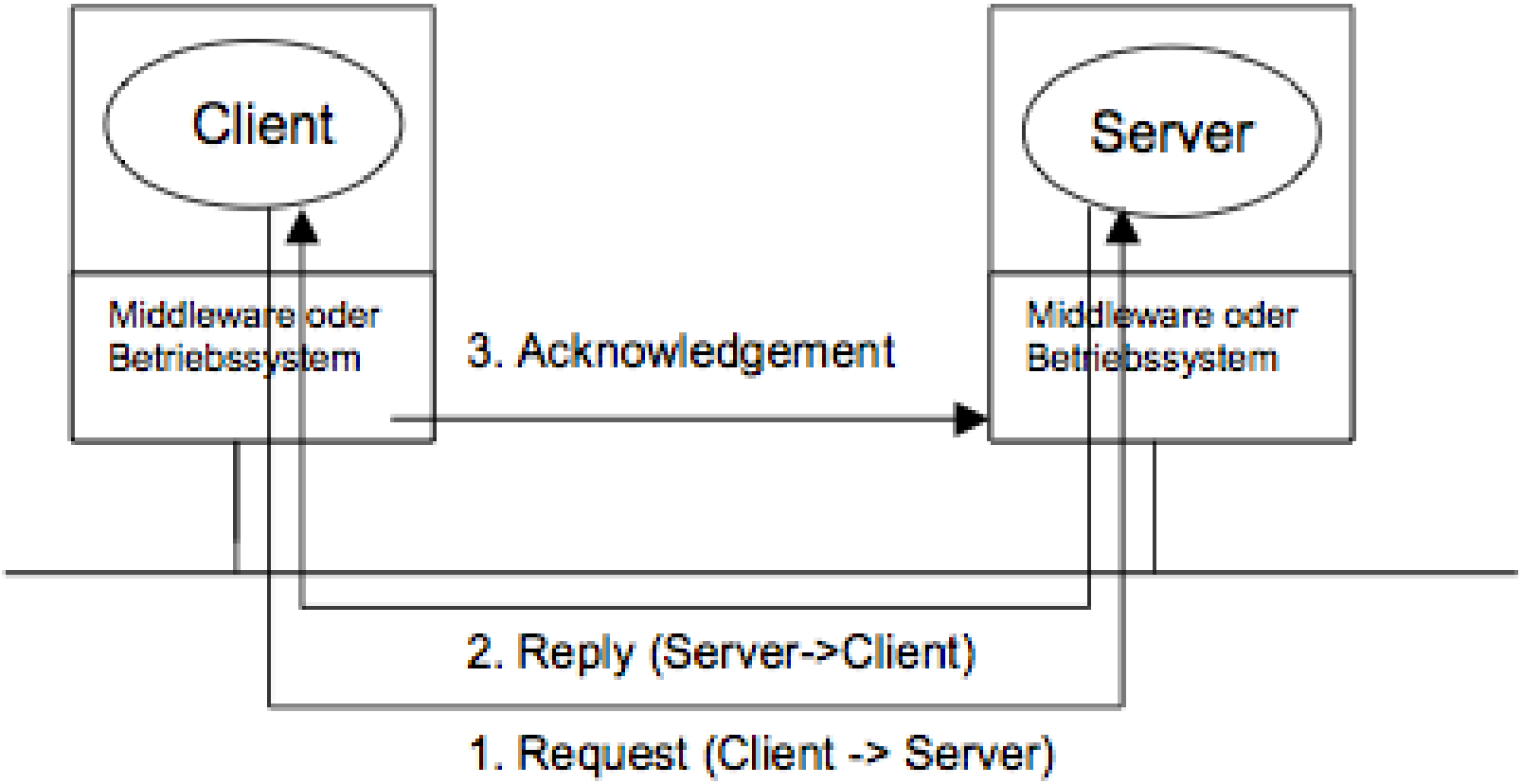


Quelle: Bengel [2004:32]

Interaktionssemantik: Fehler [II]

zu Grunde liegende Frage: *„Wie kann die Nachrichtenübertragung zuverlässiger gestaltet werden?“*

Quittierung eines Request und Reply

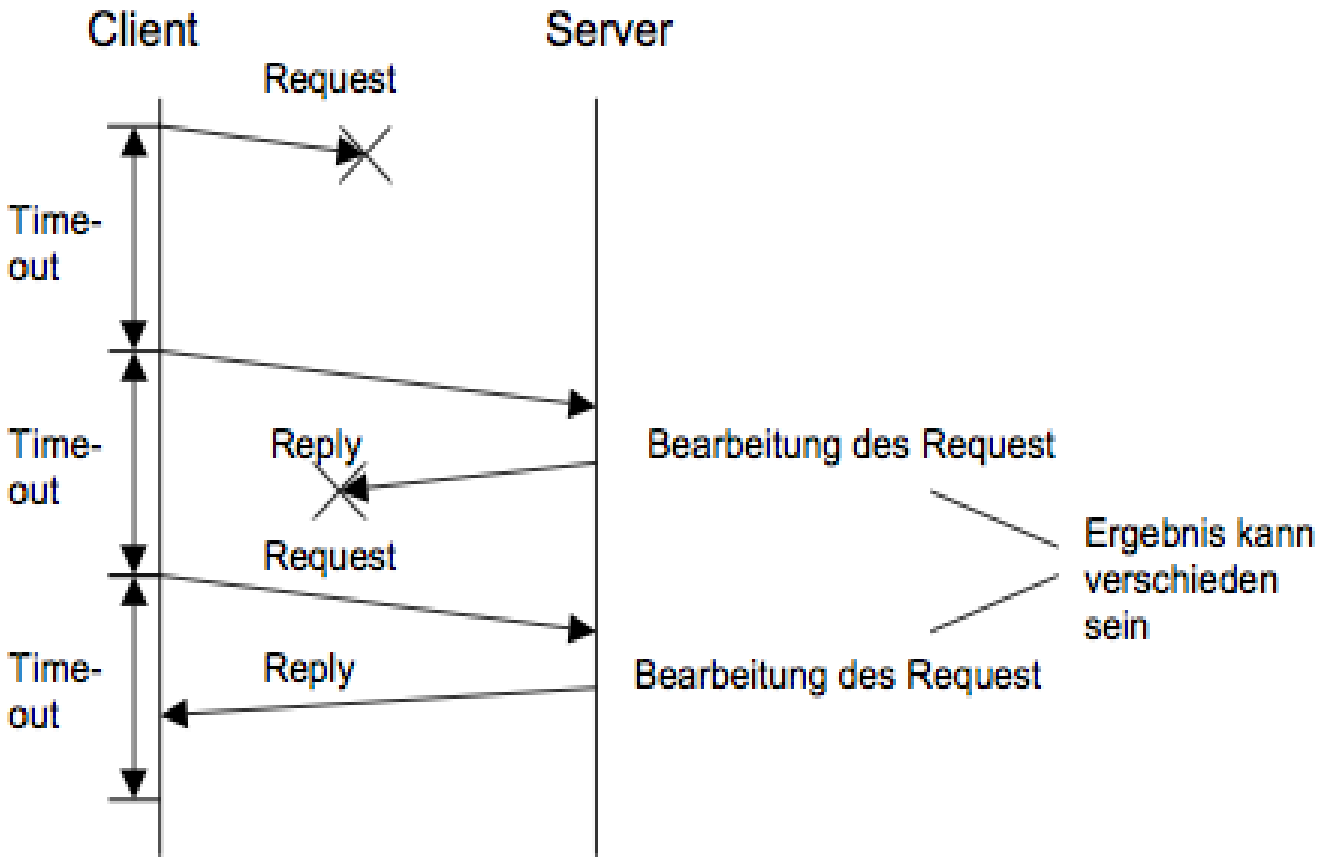


Quelle: Bengel [2004:34]

Interaktionssemantik [V]

zu Grunde liegende Frage: *„Wie koordinieren sich Client und Server beim Ablauf einer Interaktion?“*

„At-least-once“: eingehende Client-Anforderungen werden mindestens einmal bearbeitet, können aber auch mehrfach bearbeitet werden (oder überhaupt nicht)

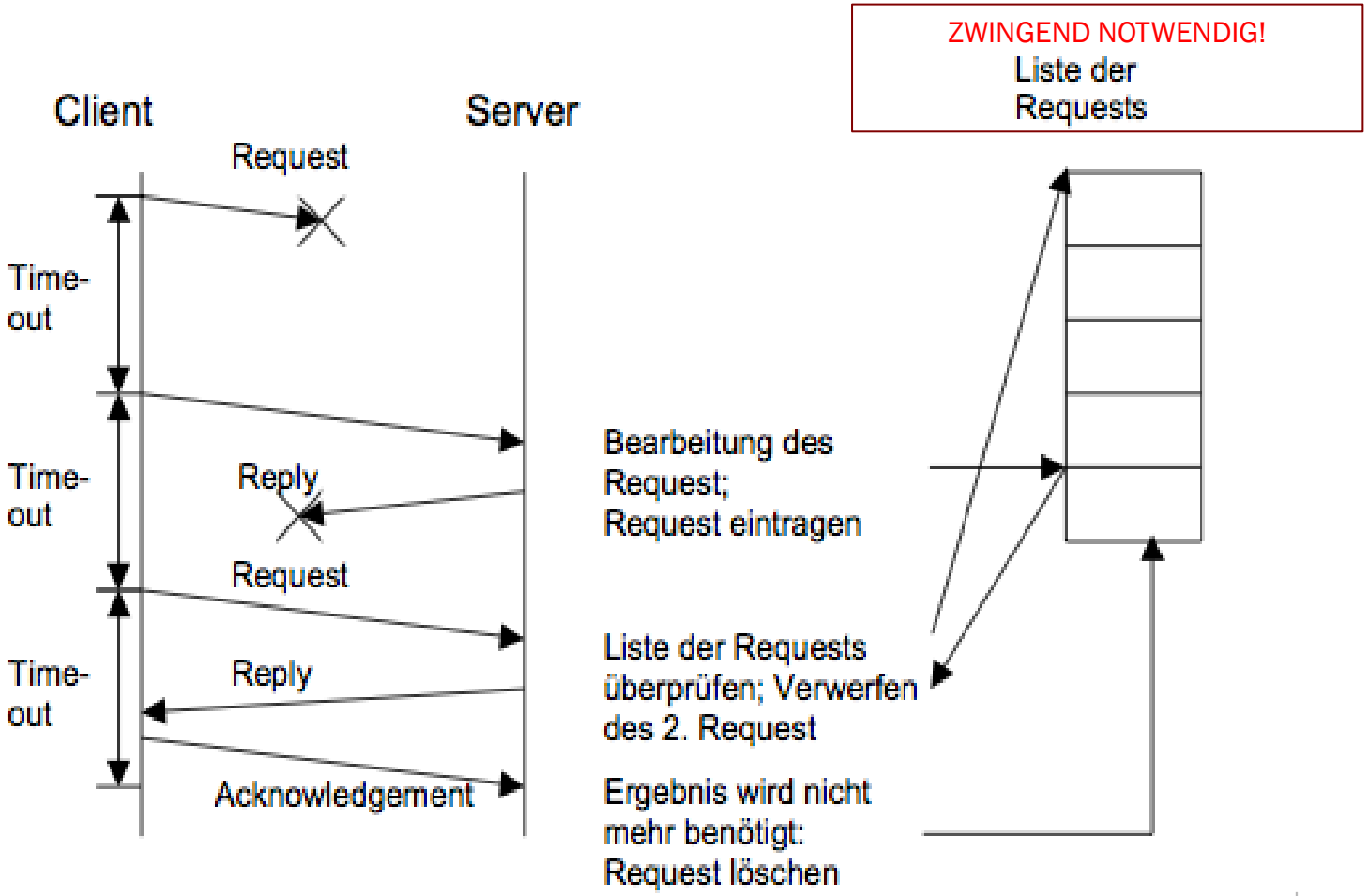


Quelle: Bengel [2004:36]

Interaktionssemantik [VI]

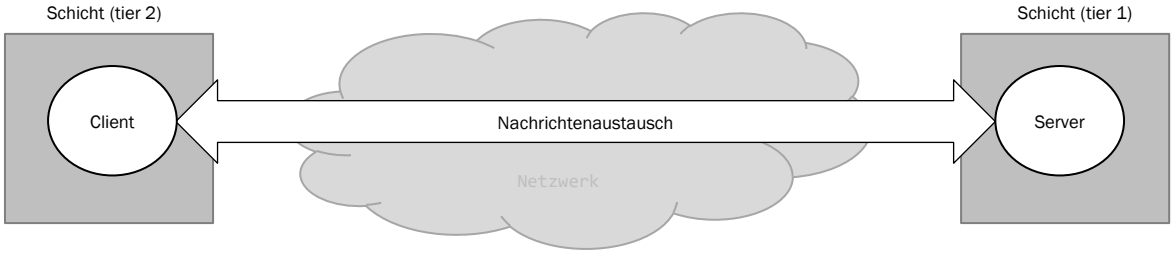
zu Grunde liegende Frage: *„Wie koordinieren sich Client und Server beim Ablauf einer Interaktion?“*

„At-most-once“: eingehende Client-Anforderungen werden höchstens einmal bearbeitet (oder überhaupt nicht)



Quelle: Bengel [2004:36]

Ausfälle und deren Auswirkung in C/S Systemen



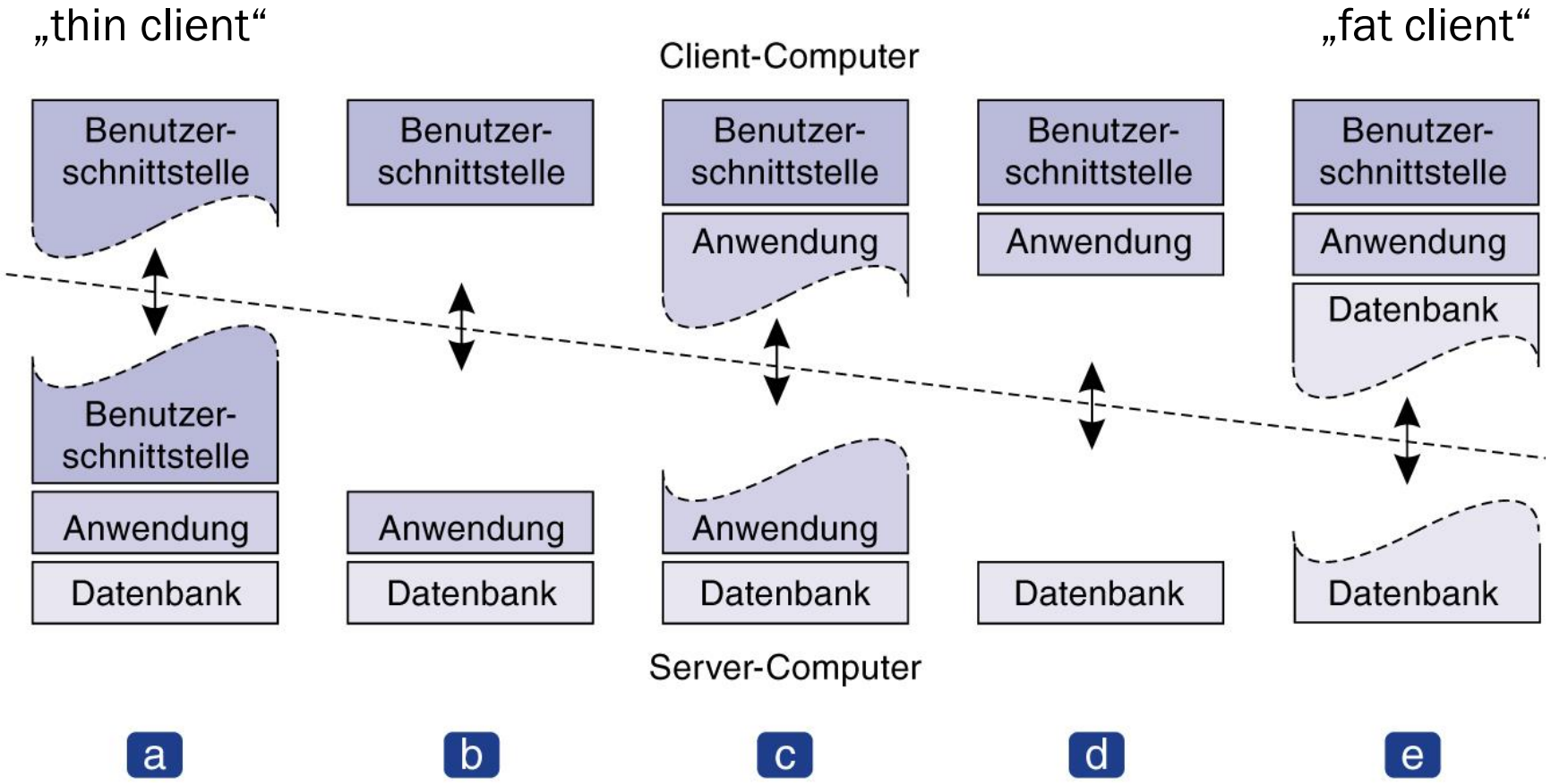
- Server: Fällt ein Server nach Erhalt einer Client-Anforderung aus, gibt es
  - » für den Server keine Möglichkeit, dies dem Client mitzuteilen
  - » für den Client keine Möglichkeit, dies herauszufinden

- Client(s): Fällt ein Client nach Senden einer Anforderung im Wartezustand aus, verwaisen Rückantworten

- Interaktionssemantiken steuern diesen Umständen nur begrenzt entgegen
- Weitere Ansätze / Klassifikationen mit Fokus auf Client und/oder Server:
  - » Z.B.: Unterschiedliche Verteilung von Funktionalität auf Client & Server, Zustand des Servers (vgl. ff.)

Variationen von Client-Server-Anordnungen i.A. des Anteils der Verteilung von Funktionalität [I]

Klassifikationsansatz für Clients



Quelle: Tanenbaum & van Steen [2008:60]

Klassifikation von Servern: nach Zustand [I]

- ✓ Der Zustand eines Servers ist die Menge der Zustände seiner Objekte

- **Zustandsinvariante Server**:
  - » Client-Anforderungen (idempotent) führen nicht zu neuen Zuständen des Servers
  - » Reihenfolge von Client-Anfragen spielt keine Rolle
  - » Beispiele: Web-Server, FTP-Server, Zeit-Server, Namens-/Directory-Server

- **Zustandsändernde Server („stateless“ oder „stateful“)**:
  - » Client-Anforderungen (nicht-idempotent) führen (möglicherweise) zu neuen Zuständen des Servers
  - » Reihenfolge von Client-Anfragen spielt große Rolle
  - » Beispiel: File-Server
    - 1. Anforderung (Client A): Delete File x
    - 2. Anforderung (Client B): Read File x (nicht mehr möglich)

Klassifikation von Servern: nach Zustand [II]

Zustandsändernde Server: **Zustandsspeichernd (stateful)**

- Server speichert seinen Zustand in seinen Zustandstabellen (Gedächtnis des Clients)
  - » Verbindungs-Identifier
  - » Zustandsinformation
- Vorteile:
  - » Reduzierung der Netzlast durch geringere Länge von Anforderungsnachrichten der Clients
  - » Performanz

Zustandsändernde Server: **Zustandslos (stateless)**

- Server besitzt keine Information über Zustand seiner Objekte, d.h. Clients senden diese Informationen innerhalb Ihrer Anforderungsnachrichten
- Vorteile:
  - » Fehlertoleranz: Absturz des Servers führt nicht zu Verlust von Zustandsdaten
  - » Skalierbarkeit

Klassifikation von Servern:

... nach Nachrichtenbearbeitung:

- Iterativ/Sequentiell: Server bearbeitet Client-Anfragen nach deren Eintreffen nacheinander
- (Pseudo-)Parallel: Hauptprozess nimmt Client-Anforderungen an und startet Unterprozesse (Threads) für jede Anfrage, welche diese bearbeiten
  - ✓ Vorteil: geringere Antwortzeiten

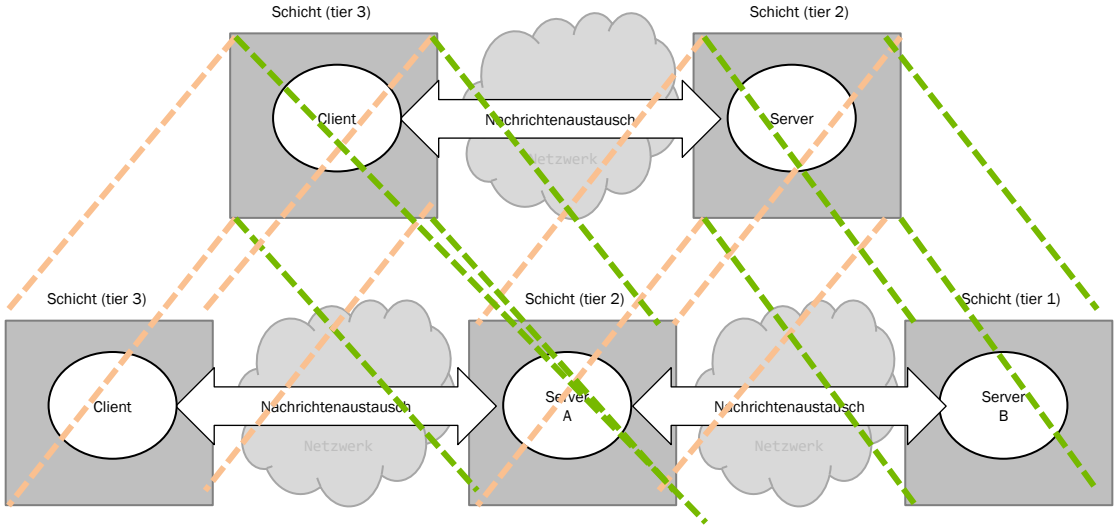
... nach Aktivierung:

- **Shared Server**: mehrere Dienste (Services) werden von einem Server angeboten
  - » Serverprozess aktiv (**persistent**): parallele oder iterative/sequentielle Bearbeitung einer Anfrage
  - » Serverprozess inaktiv (**per request**):
    - Starten des Prozesses
    - Entgegennahme der Anforderung: parallele oder iterative/sequentielle Bearbeitung einer Anfrage
- **Unshared Server**: jeder Dienst liegt in einem eigenen Server (persistent oder per request)

Client-Server (C/S): Geschichtete Architekturstile

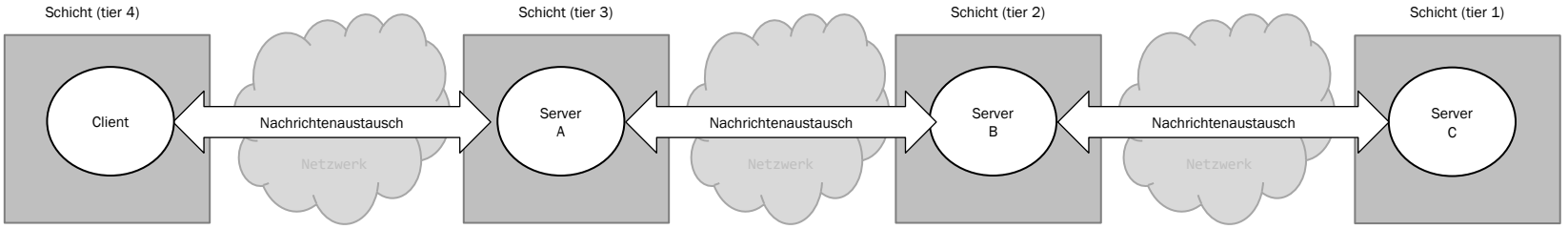
- ✓ Interaktionssemantiken, … lassen sich bilateral auch auf n-/multitier-Architekturen übertragen
- ✓ Viele Variationen / Kombinationen möglich (vgl. ff.)

2-tier:



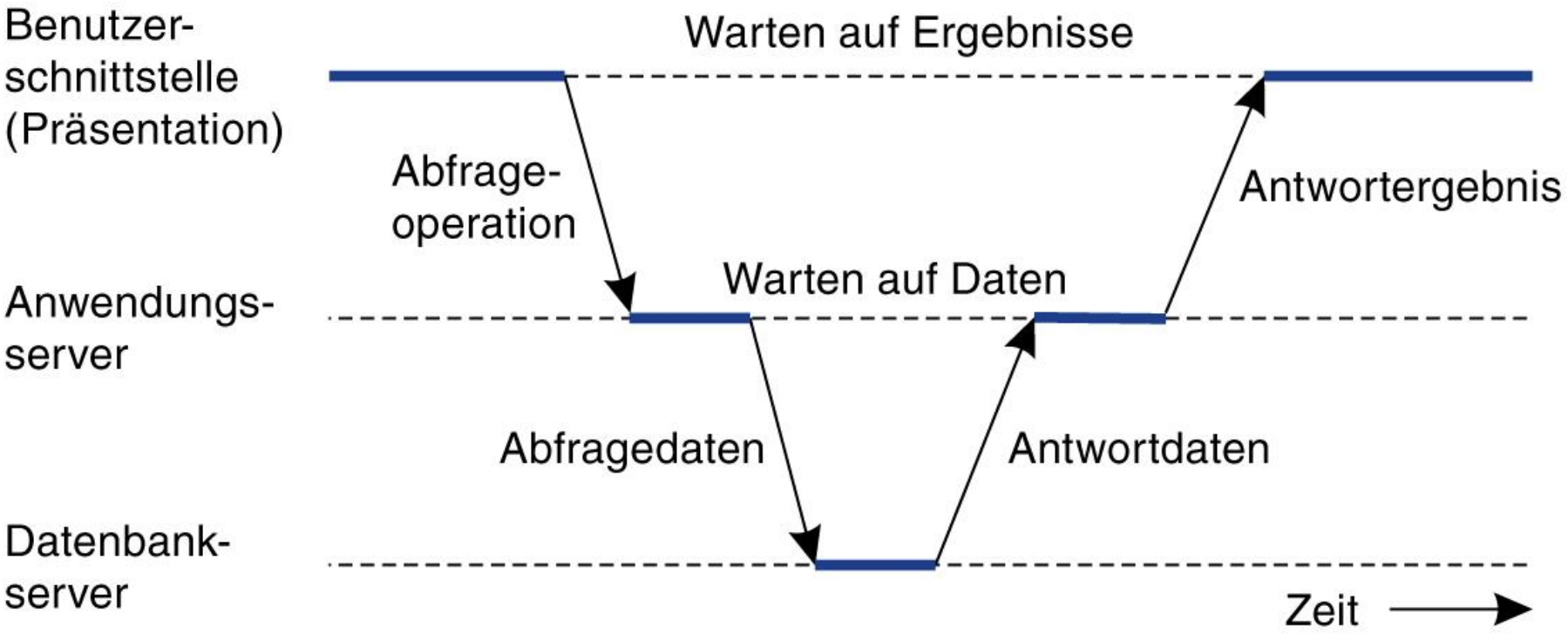
3-tier:

4-tier:



Beispiel: Variation (3-tier) [I]

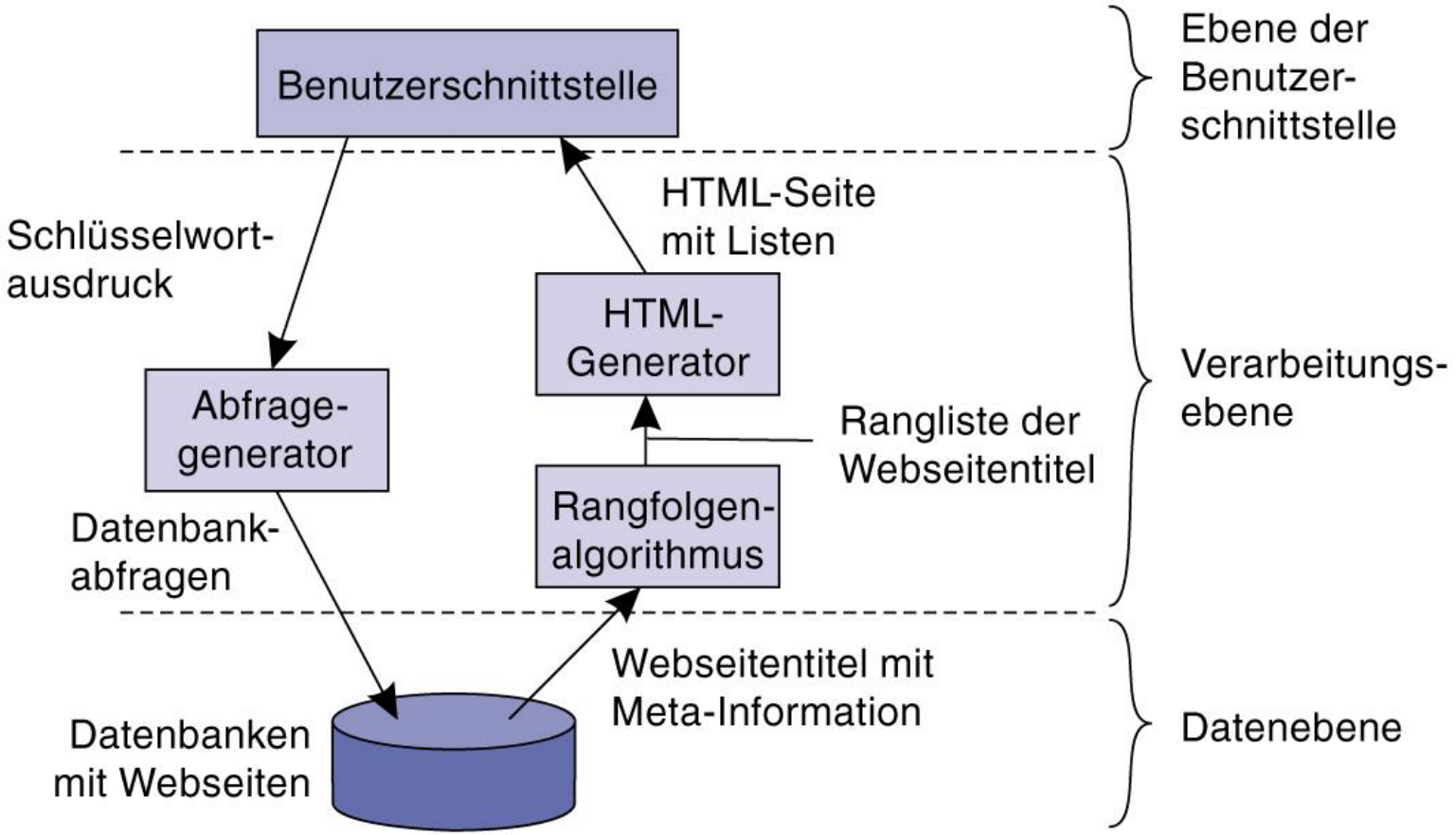
Synchrone Zusammenarbeit zwischen einem Client und zwei Servern (ein Server fungiert auch als Client):



Quelle: Tanenbaum & van Steen [2008:61]

Beispiel: Variation (3-tier) [II]

Aufbau einer Suchmaschine im Internet in drei unterschiedlichen Ebenen:

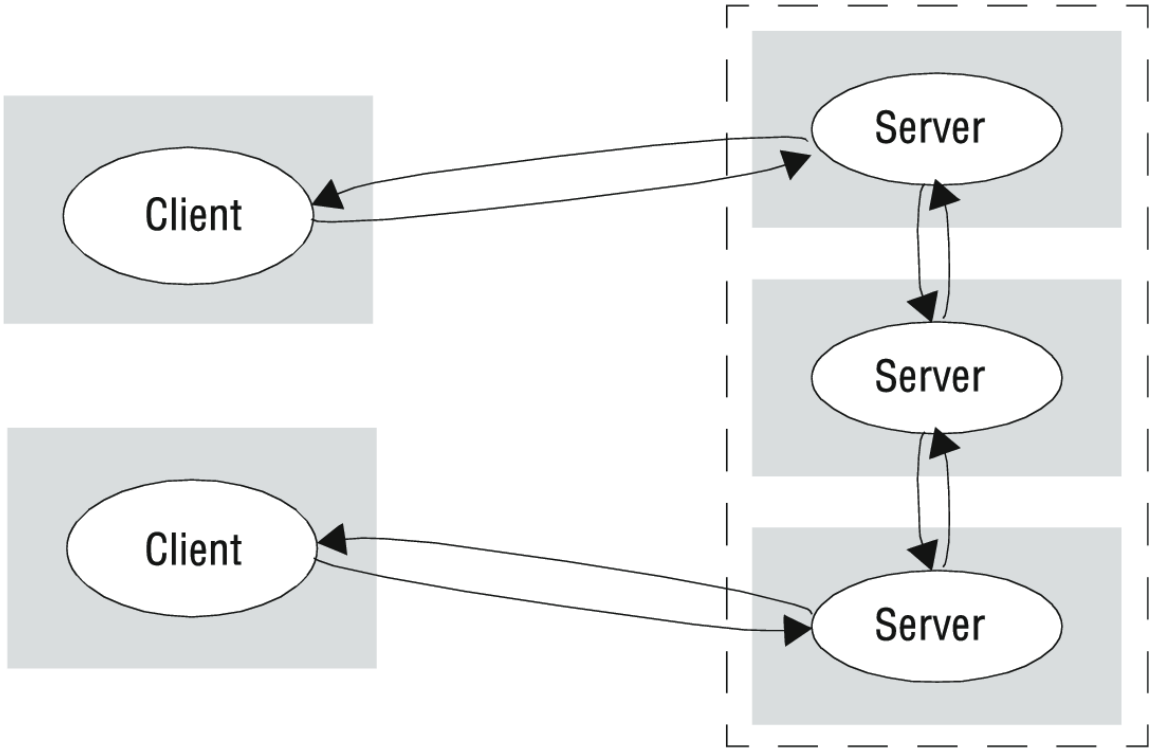


Quelle: Tanenbaum & van Steen [2008:58]

Beispiel: Variation [II]
Dienstbereitstellung durch mehrere Server

Implementierung mehrerer Server-Prozesse in separaten Host-Computern:

- Zerlegung und Verteilung der Objektmenge eines Dienstes auf mehrere Hosts (z.B.: Partitionierung von Daten und Verwaltung eigener Ressourcenmengen durch Webserver im Web)
- Replikation: Verwaltung von konsistenten Kopien eines Dienstes durch mehrere Hosts
- Die Rolle(n) verschiedener Server beleuchten wir später noch…



Quelle: Coulouris et al. [2002:54]

*/

Klassifikation von C^+SS^+-Systemen: nach Rolle von Servern in geschichteten Architekturen [I]

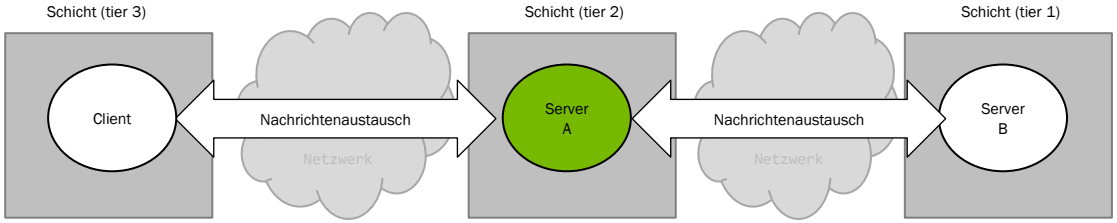
vgl. Bengel [2004: 68ff.] & Bengel et al. [2008: 118ff.]

Client-Server-Server-Systeme ($C^+SS^{+)}$ bezeichnen verteilte Systeme mit drei logischen Teilen:

- Schicht N: Client(s)
- Schicht N-1: Server, der zusammen mit den unter N-2, ... , N-i genannten Servern eine bestimmte Aufgabe erfüllt
- Schicht N-2: weitere_r Server

Klassifikation von C^+SS^+-Systemen: nach Rolle von Servern in geschichteten Architekturen [II]

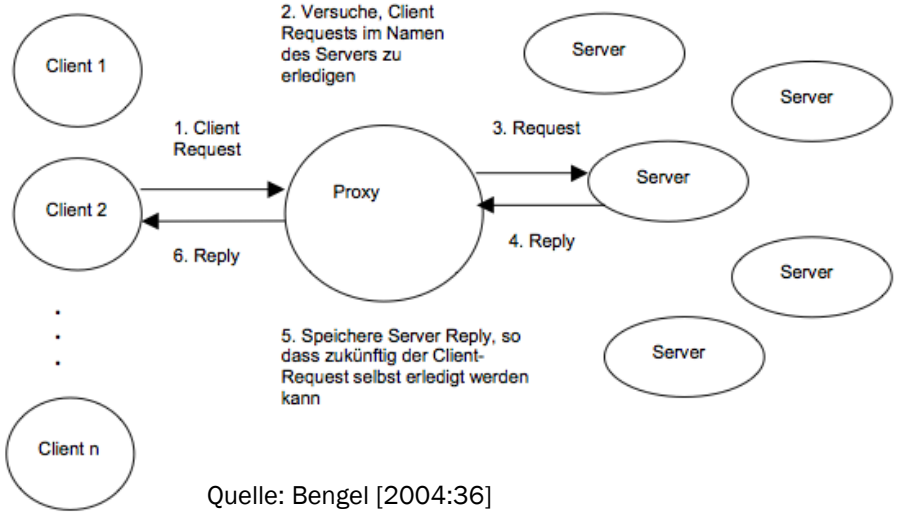
vgl. Bengel [2004: 68ff.] & Bengel et al. [2008: 118ff.]



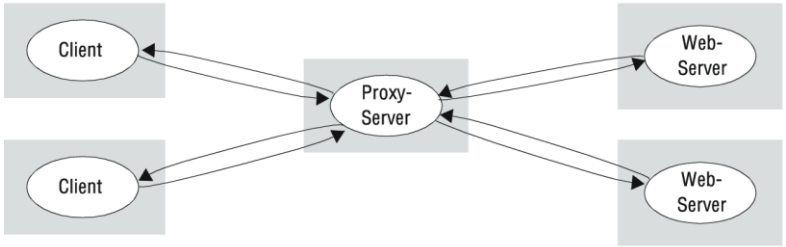
Abkürzung	Rolle Server A	Beschreibung
$C^+S_PS^+$	*Proxy*	Server kann mehrere Server vertreten
$C^+S_{Br}S^+$	*Broker*	Server kann zwischen Clients und Servern in tier-1 vermitteln
$C^+S_TS^+$	*Trader*	Server kann den am besten geeigneten Server tier-1 in Bezug auf Charakteristiken und Eigenschaften für diese Aufgabe aus einer Menge von Servern heraussuchen
$C^+S_FS^+$	*Filter*	Server filtert Anfragen / Antworten
$C^+S_{Ba}S^+$	*Balancer*	Server verteilt Arbeitslast auf mehrere Server in tier-1
$C^+S_KS^+$	*Koordinator*	Server koordiniert Sequenz von Teilleistungen eines Dienstes
$C^+S_AS^+$	*Agent*	Server bestimmt (möglicherweise mit Künstlicher Intelligenz) aus Client-Anforderung weitere Anfragen an Server in tier-1.

$C^+S_PS^+$-System: Proxy

- Server kann mehrere Server vertreten
- Fungiert als „Cache“: Speicher für Datenobjekte, die vor kurzem verwendet wurden
- Implementierung:
  - Im Client (z.B. Webbrowser)
  - Auf einem Server, der durch mehrere Clients genutzt wird (Proxy-Server)
- Aufruf eines eines Objekts erfolgt immer über den Cache
  - Verfügbarkeitsprüfung
  - Aktualitätsprüfung
  - Ausführung / Anzeige



Quelle: Bengel [2004:36]

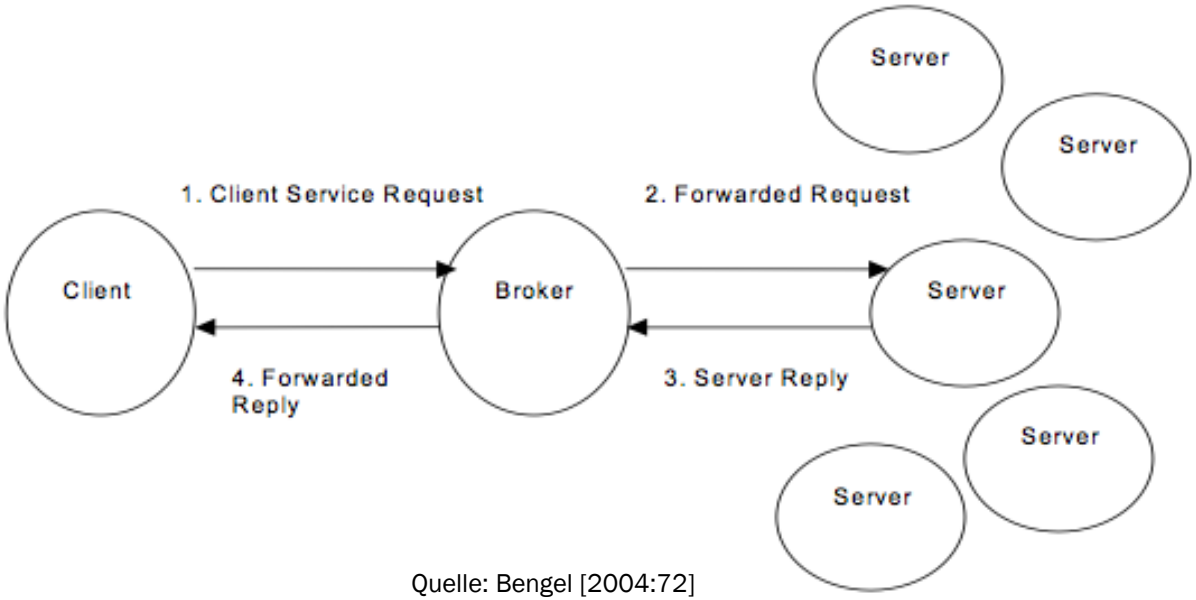


Quelle: Coulouris et al. [2002:54]

Vorteil_e	Nachteil_e
✓ Reduzierung ✓ Antwortzeiten ✓ Last hinsichtlich Server in niedrigerer Schicht / im gesamten Netz (z.B.: Wide Area Network) – Erhöhung der Verfügbarkeit/Leistung von Diensten – Ausübung weiterer Rollen möglich (z.B.: Firewall)	✓ Limitation auf zustandsinvariante Server (z.B. im WWW) ✓ Konsistenzrisiken

$C^+S_{Br}S^+$-System: Broker (forwarding)

- ✓ Server kann zwischen Client(s) und Servern in darunter / dahinter liegender Schicht vermitteln

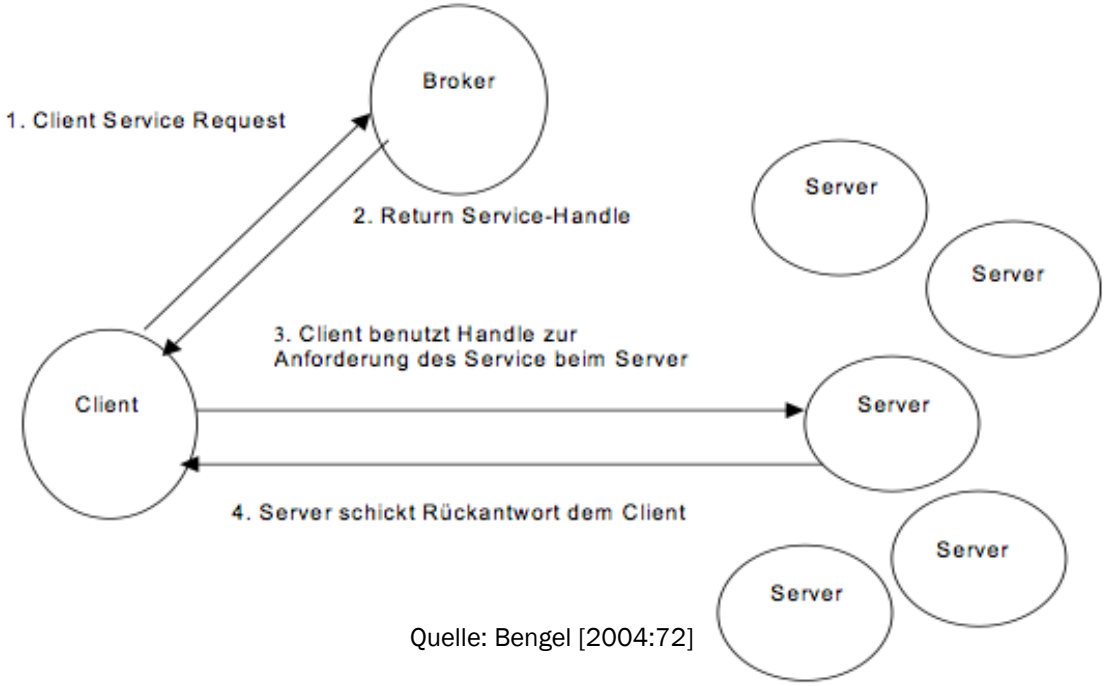


Quelle: Bengel [2004:72]

Vorteil_e	Nachteil_e
– Client muss genaue Adresse eines Servers nicht kennen (Ortstransparenz) – leichte Erweiterbarkeit des Brokers zur Adressverwaltung – Replikationstransparenz durch Möglichkeit, beim Broker weitere Eigenschaften eines Servers (replizierte Server) anzugeben – Migrationstransparenz: einfache Änderung von Adressen migrierter Server beim Broker	– S_{Br} Single-Point-of-Failure / Single-Point-of-Attack (Empfohlen: Replikation des Brokers) – Bei Replikation: Konsistenz der Datenbasis (Adressen aller Server im System)

$C^+S_{Br}S^+$-System: Broker (handle driven)

- ✓ Server kann zwischen Client(s) und Servern in darunter / dahinter liegender Schicht vermitteln

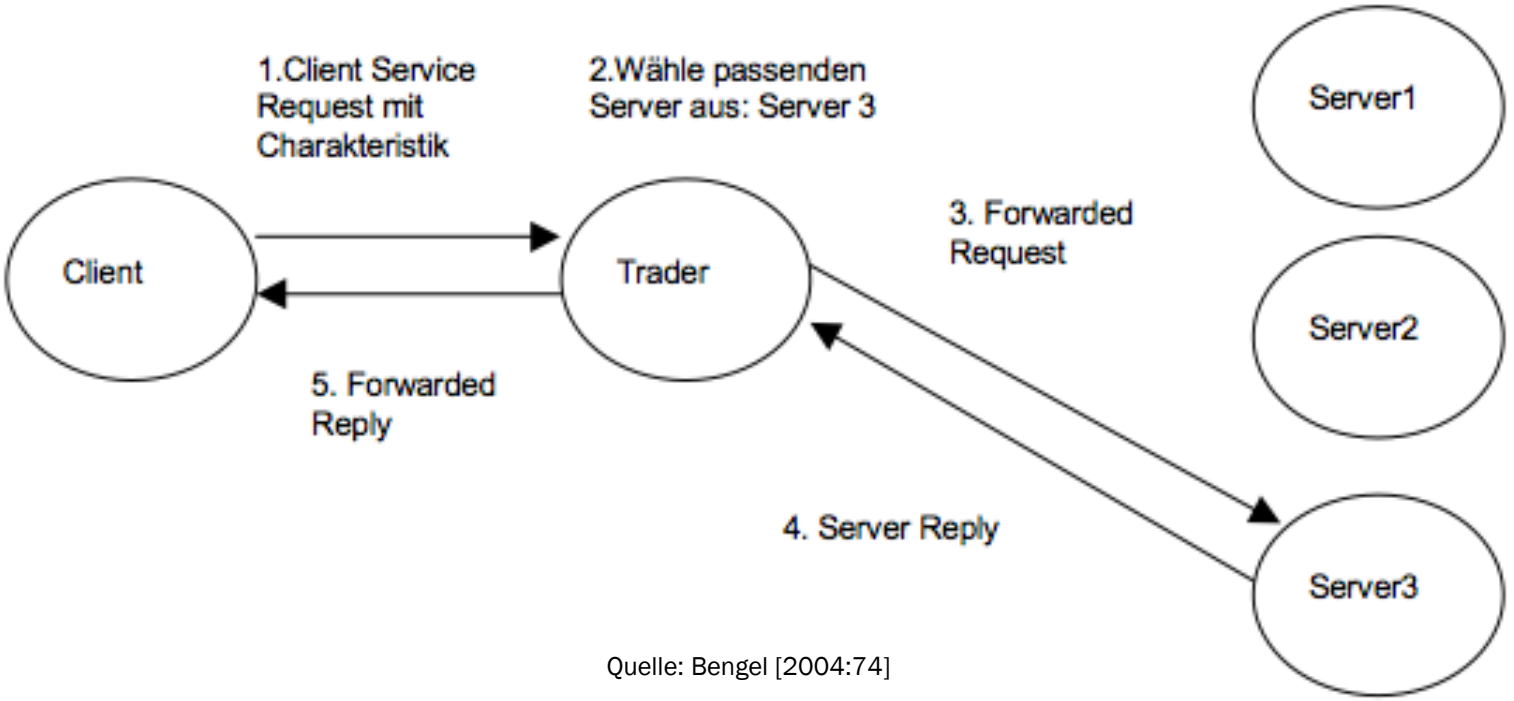


Quelle: Bengel [2004:72]

Vorteil_e	Nachteil_e
– Client muss genaue Adresse eines Servers nicht kennen (Ortstransparenz) – leichte Erweiterbarkeit des Brokers zur Adressverwaltung – Replikationstransparenz durch Möglichkeit, beim Broker weitere Eigenschaften eines Servers (replizierte Server) anzugeben – Migrationstransparenz: einfache Änderung von Adressen migrierter Server beim Broker	– S_{Br} Single-Point-of-Failure / Single-Point-of-Attack (Empfohlen: Replikation des Brokers) – Bei Replikation: Konsistenz der Datenbasis (Adressen aller Server im System)

$C^+S_TS^+$-System: Trader

- Server kann den am besten geeigneten Server in darunter / dahinter liegender Schicht in Bezug auf Charakteristiken und Eigenschaften für diese Aufgabe aus einer Menge von Servern aussuchen
  - » Erweiterter Broker, da Trader bei Servern, die den gleichen Service anbieten, geeigneten Diensterbringer aussucht (Einsatzbeispiel: Reporting-Anfragen in unterschiedlicher Qualität)
  - » Vor- und Nachteile vgl. „$C^+S_{Br}S^+$-System: Broker“

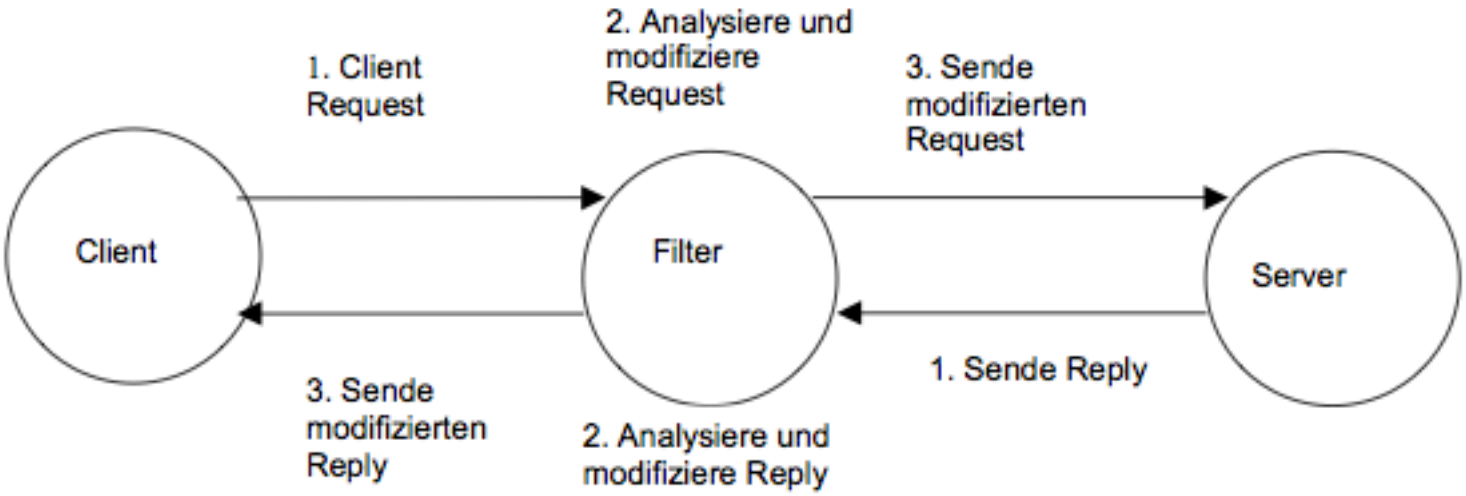


Quelle: Bengel [2004:74]

*/

$C^+S_FS^+$-System: Filter

- ✓ Server filtert Anfragen / Antworten, d.h. er analysiert/modifiziert diese Nachrichten (Einsatzbeispiele: Verschlüsselung, Komprimierung, Logging)

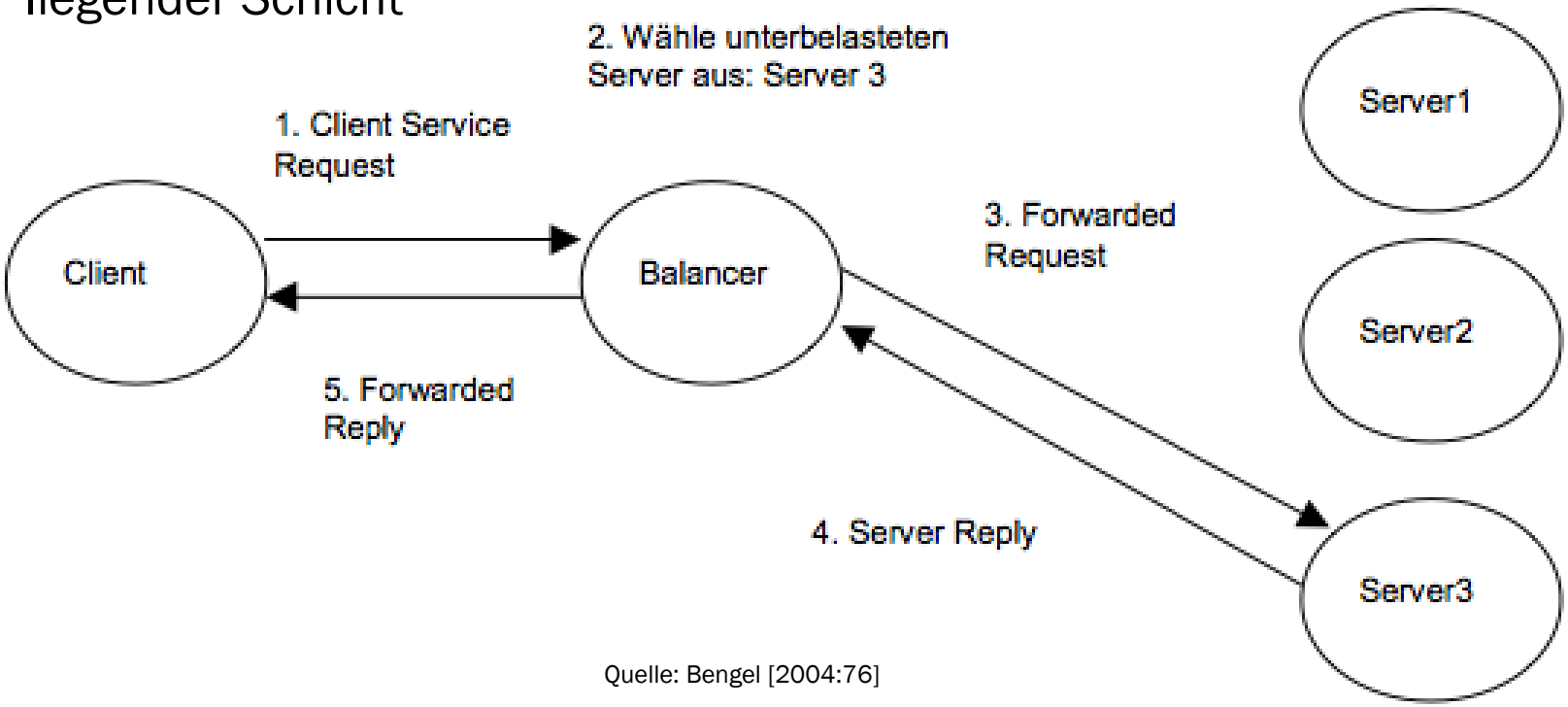


Quelle: Bengel [2004:75]

Vorteil_e	Nachteil_e
– Auslagerungsmöglichkeit repetitiver Tätigkeiten eines nachgelagerten Servers (Vorwärts-Filter + Rückwärts-Filter)	– S_F Single-Point-of-Failure / Single-Point-of-Attack (Empfohlen: Replikation des Filters) – Bei Replikation: Konsistenz der Analyse- und Modifikationsoptionen

$C^+S_{Ba}S^+$-System: Balancer

- ✓ Server verteilt Arbeitslast auf mehrere Server in darunter / dahinter liegender Schicht

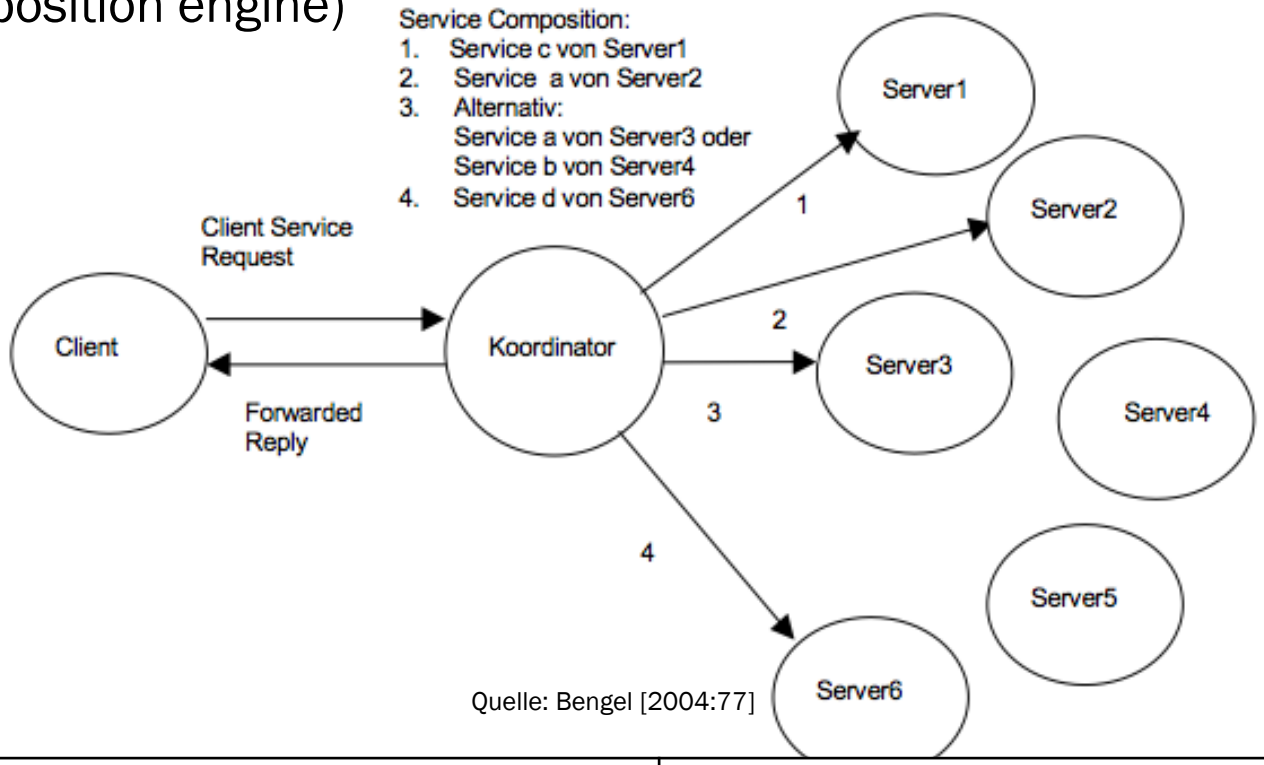


Quelle: Bengel [2004:76]

Vorteil_e	Nachteil_e
– Gleichmäßige Auslastung der nachgelagerten Server	– nur bei replizierten nachgelagerten Servern einsetzbar

$C^+S_KS^+$-System: Koordinator

- ✓ Server koordiniert Sequenz von Teilleistungen eines Dienstes (composition engine)

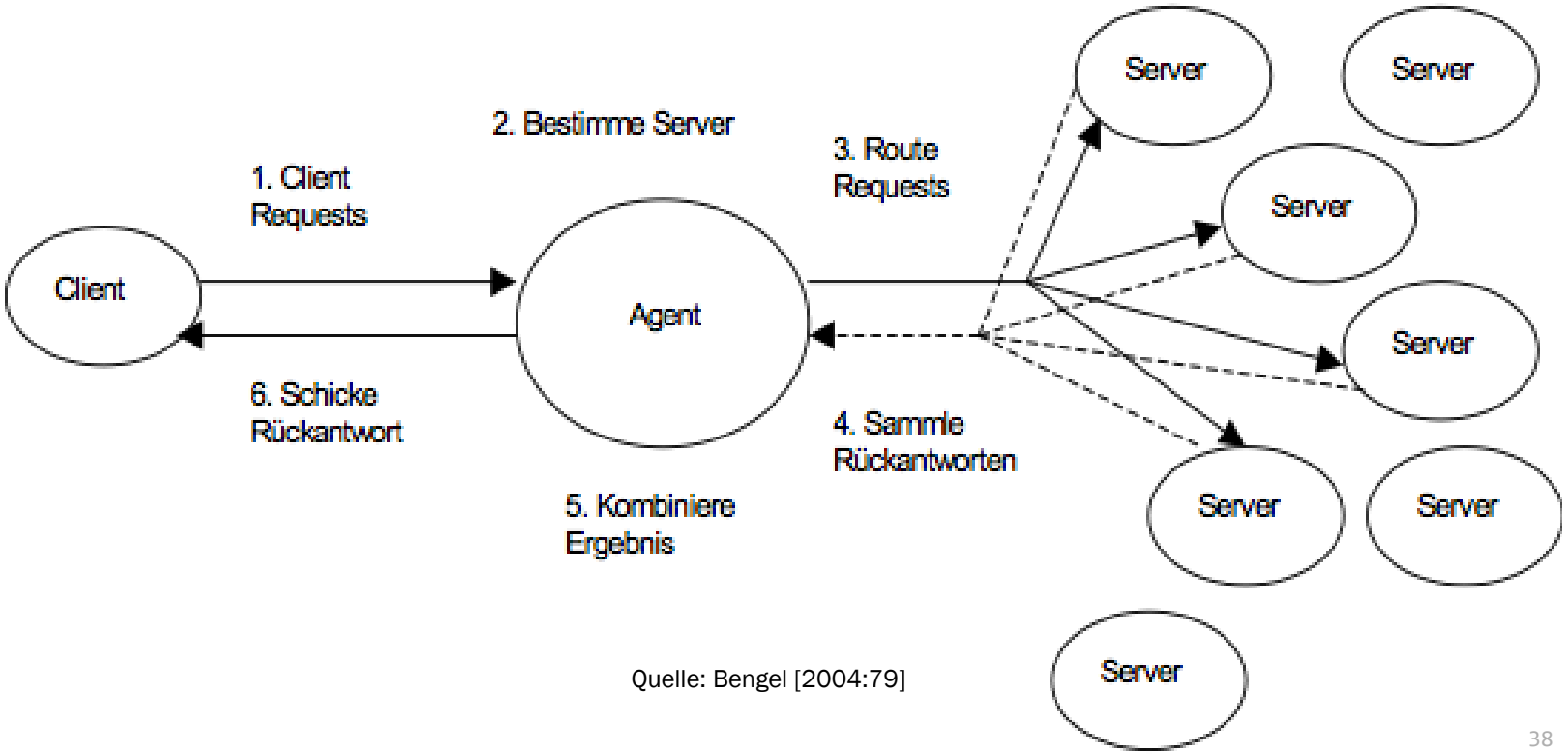


Quelle: Bengel [2004:77]

Vorteil_e	Nachteil_e
– Verfügbarkeit von Diensten – Teilleistungen sind transparent für Client – Ermöglichung „schlanker Clients“ bei angeforderten Services, die sich aus sequentiell zu durchlaufenden (alternativen) Einzelservices zusammen setzen („Service Composition“) – Performanz bei Möglichkeit zur Parallelisierung (Divide and Conquer, Master-Slave)	– S_K Single-Point-of-Failure / Single-Point-of-Attack (Empfohlen: Replikation des Filters) – Bei Replikation: Konsistenz der Service Composition

$C^+S_AS^+$-System: Agent [I]

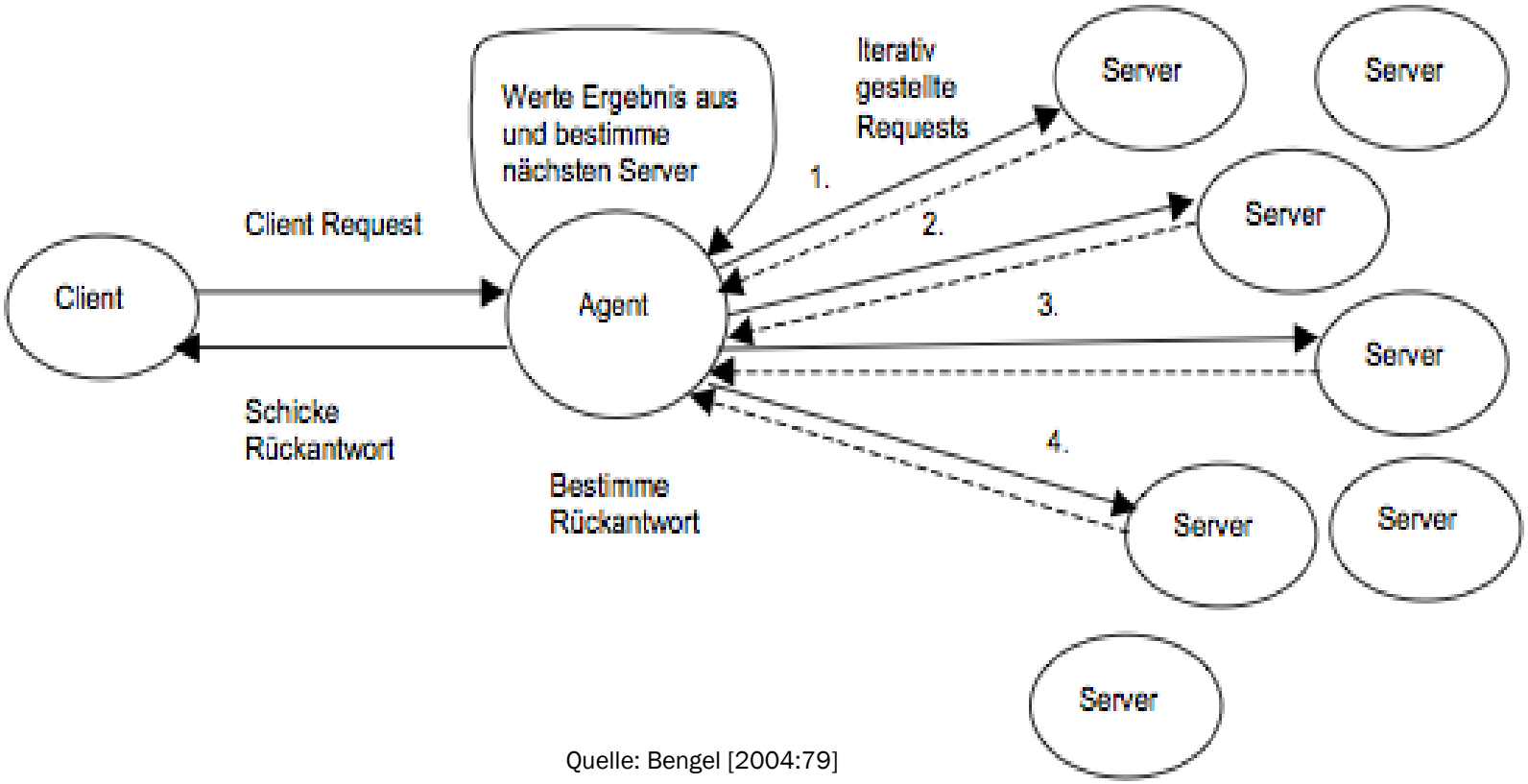
- ✓ Server bestimmt als Koordinator ohne vorherige Festlegung der Service Composition (möglicherweise mittels Künstlicher Intelligenz) aus Client-Anforderung weitere Anfragen an Server in darunter / dahinter liegender Schicht
- ✓ Protokoll zwischen Agent und Servern: iterativ oder parallel



Quelle: Bengel [2004:79]

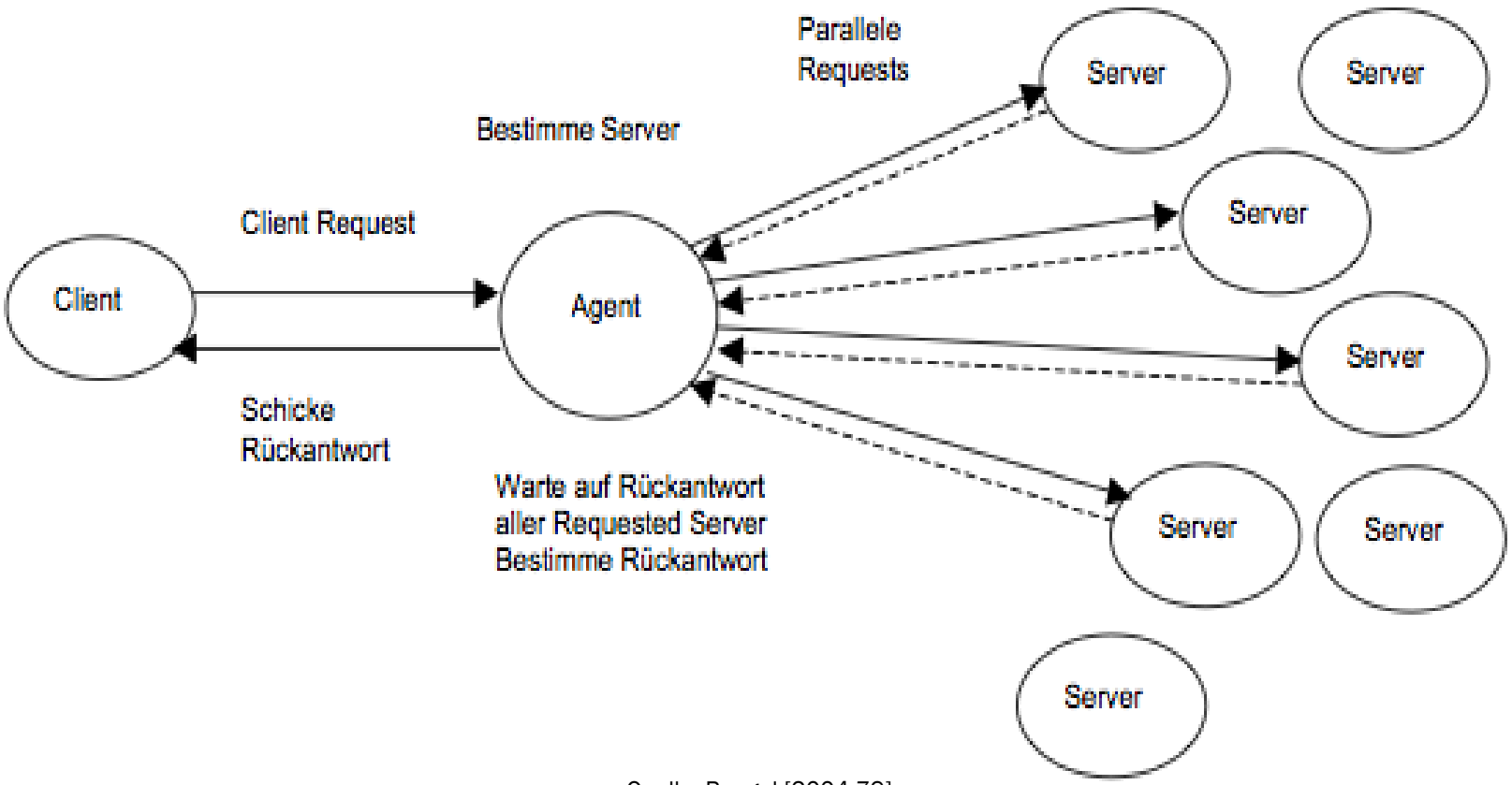
*/

$C^+S_AS^+$-System: Agent [II]
Iterative Serveranfragen



Quelle: Bengel [2004:79]

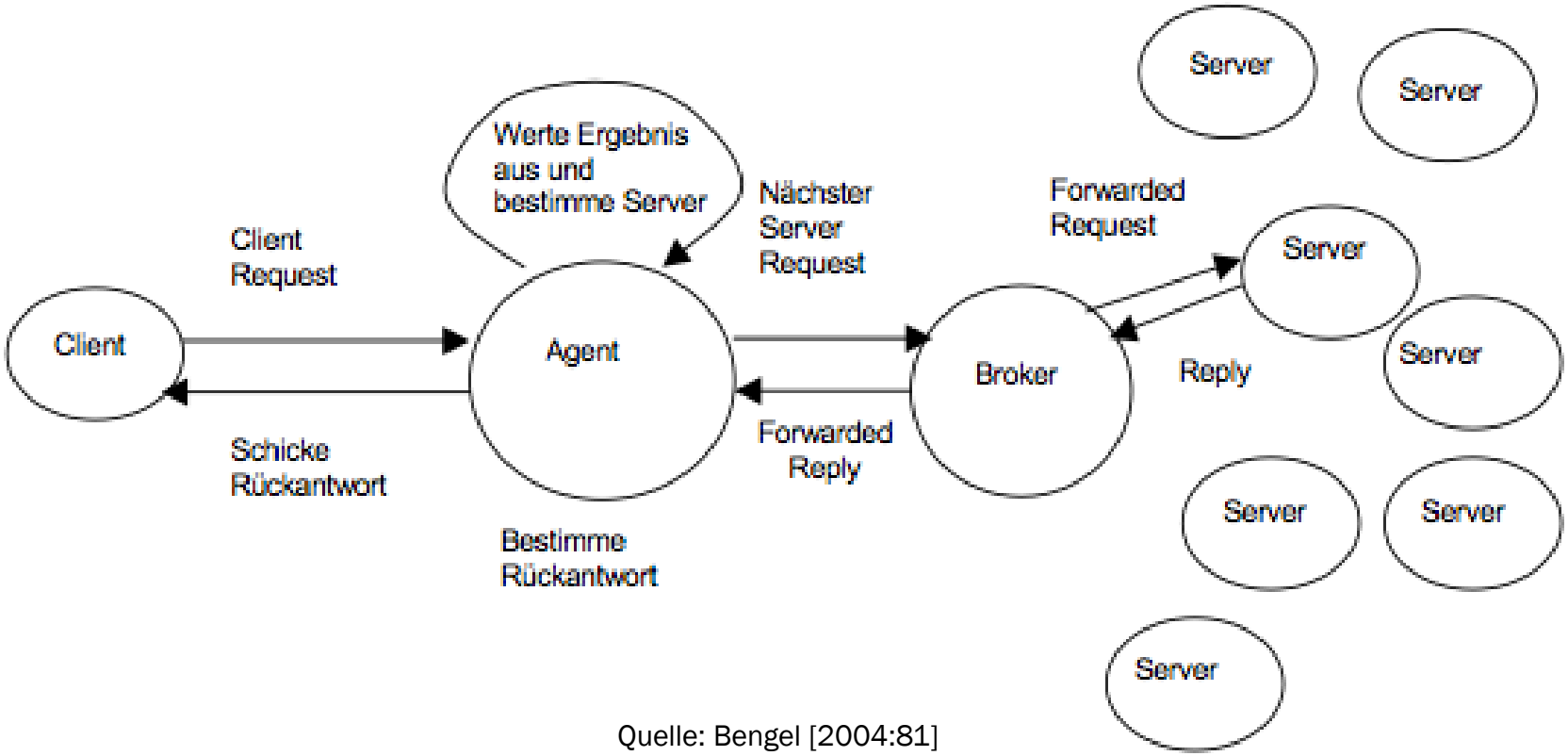
$C^+S_AS^+$-System: Agent [III]
Parallele Serveranfragen



Quelle: Bengel [2004:79]

Beispiel: Kombinationsmöglichkeiten

- ✓ Adressermittlung des Agenten bei einem $C^+S_AS_{Br}S^+$-System mit einem forwarding Broker



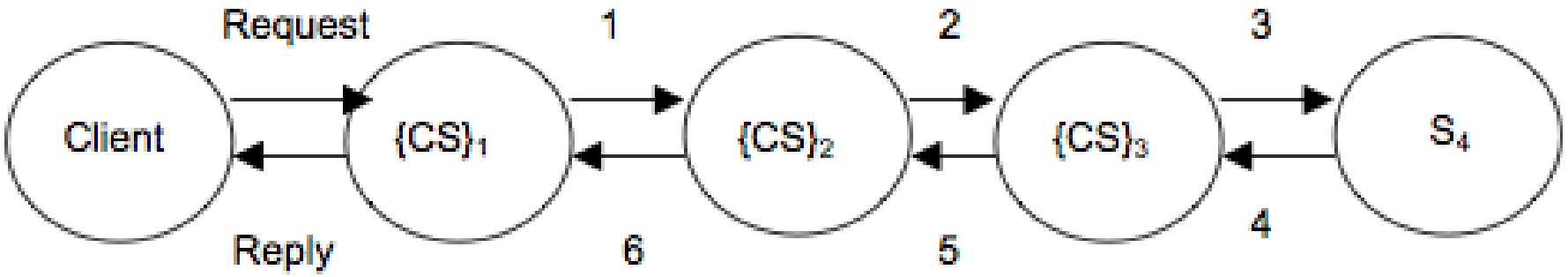
Quelle: Bengel [2004:81]

Klassifikation von C⁺SS⁺-Systemen: nach Verkettung [I]

vgl. Bengel [2004: 68ff.] & Bengel et al. 2008: [118ff.]

Rekursive Client-Server-Kette:

- ohne Koordinator
- Server bestimmen selbst, welcher nachfolgende Server die Weiterbearbeitung der Client-Anforderung übernimmt
- Client erhält nach rekursivem Durchlauf der gesamten Serverkette die Antwort desjenigen Servers, der die initiale Anforderung entgegengenommen hat



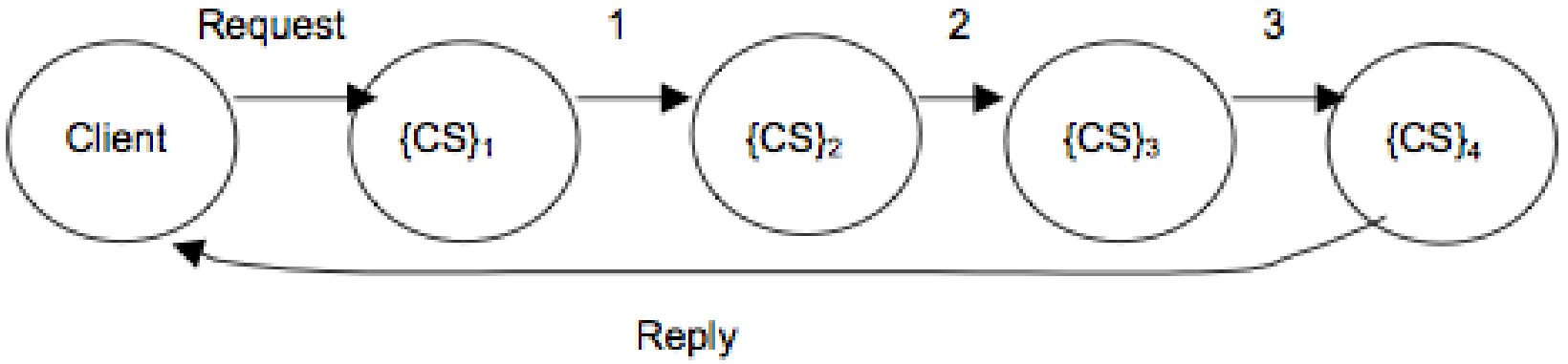
Quelle: Bengel [2004:82]

Klassifikation von C+SS+-Systemen: nach Verkettung [II]

vgl. Bengel [2004: 68ff.] & Bengel et al. 2008: [118ff.]

Transitive Client-Server-Kette:

- ohne Koordinator
- Server bestimmen selbst, welcher nachfolgende Server die Weiterbearbeitung der Client-Anforderung übernimmt
- Client erhält nach sequentiellem Durchlauf der an Bearbeitung der Anforderungsnachricht beteiligter Server die Antwort desjenigen Servers, der den letzten Bearbeitungsschritt vorgenommen hat entgegengenommen hat

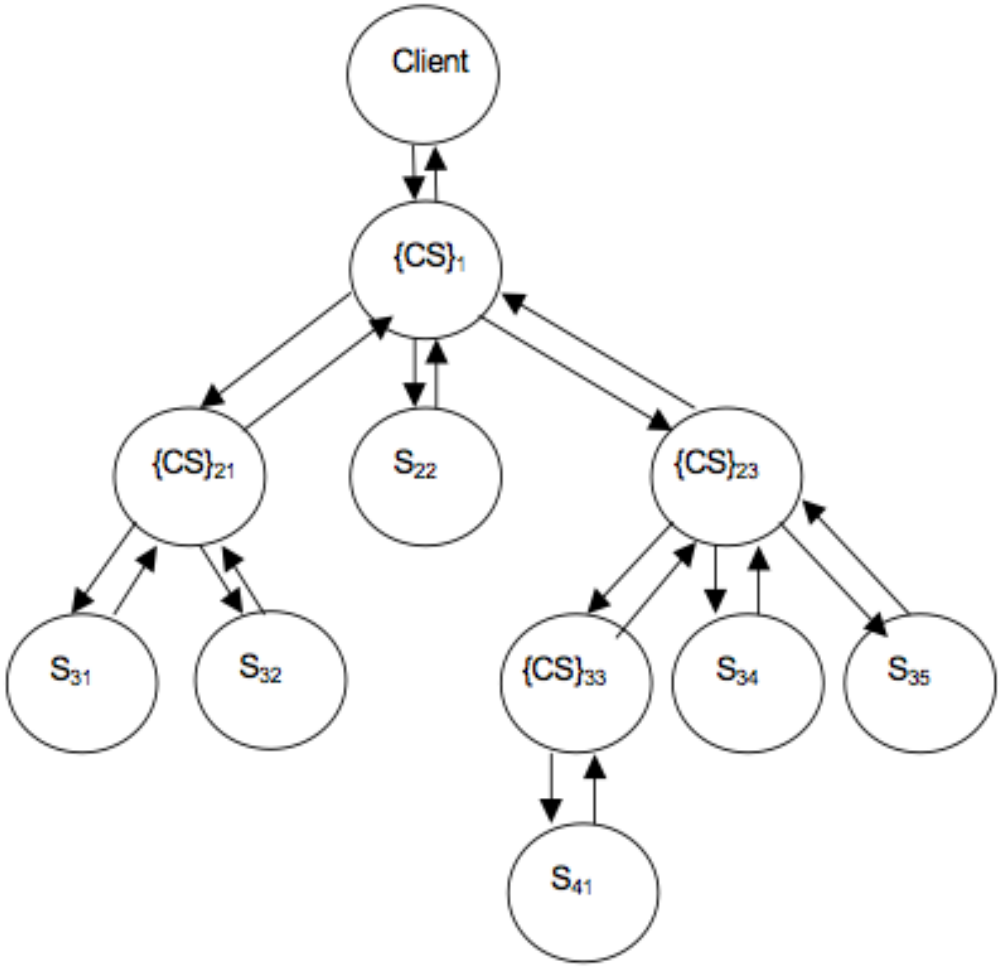


Quelle: Bengel [2004:82]

Klassifikation von C+SS+-Systemen: nach Verkettung [III]

vgl. Bengel [2004: 68ff.] & Bengel et al. 2008: [118ff.]

Bäume: Verkettungen, bei denen die Anzahl der Nachfolgeprozesse größer als 1 ist



Quelle: Bengel [2004:83]

Quelle: Bengel [2004:83]

*/

C+S-Paradigma: Bewertung

vgl. Bengel et al. [2008: 118ff.]

Vorteile:

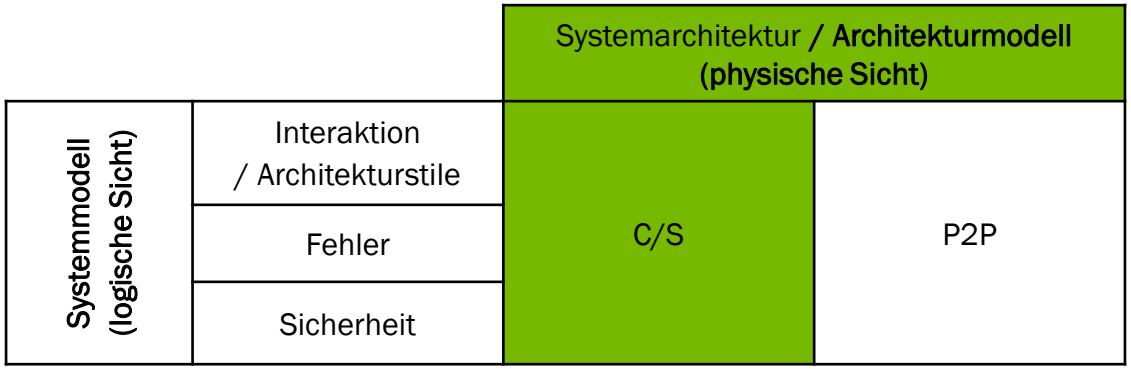
- Administration/Koordination
- „Einfache“ Abstraktion:
  - » Aufteilen von Teilen einer Anwendung in Client- und Server-Teile
  - » Abbildung der Interaktion (request/reply) zwischen Client und Server auf Nachrichtenverkehr (send/reply)

Nachteile:

- Skalierbarkeit
- Server ist „Performance bottleneck“ bei dramatischer Zunahme von Client-Anfragen, Single-Point-of-Failure und Single-Point-of-Attack
- Kommunikationskanal zum Server: „Channel bottleneck“
- Schlechte Nutzung des Client-Potenzials: Rechenleistung erfolgt meist durch Server
- Programmierparadigma ist prozedural (entfernter Methoden oder Prozeduraufruf), d.h. Ausschluss deklarativer oder funktionaler Programmierparadigmen

✓ Alternative(n):
  - ✓ P2P
  - ✓ Hybride Architekturen

Danke. Lernziele erreicht?



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.

Quellen

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

Bengel, G.; Baun, C.; Kunze, M.; Stucky, K.-U. (2008) *Masterkurs Parallele und Verteilte Systeme*; Wiesbaden: Vieweg & Teubner.

Bengel, G (2004) *Grundkurs Verteilte Systeme*, 3. Auflage; Wiesbaden: Springer / Vieweg.

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.