AI-B.41: Verteilte Systeme

Übungseinheit 2

Übungseinheit 2: Lernziele

- Vertiefung von grundlegenden Aspekten verteilter Systeme (Grundlagen für die Bearbeitung des Übungsblattes):
  - Skalierbarkeit
  - Verteilung
  - n-tier-Architekturen
  - Zeitkonzepte
- Beginn der Bearbeitung von Übungsblatt 2

Skalierung & Verteilung

„Skalierbarkeit“: Begriffseingrenzung

Definition	Quelle(n)
„Ein System, dass als skalierbar bezeichnet wird, bleibt auch dann effektiv, wenn die Anzahl der Ressourcen und die Anzahl der Benutzer wesentlich steigt.“	Coulouris et al. [2002:38]
„Die Skalierbarkeit eines Systems lässt sich in mindestens drei unterschiedlichen Dimensionen messen (Neumann, 1994). Erstens kann ein System skalierbar in Hinblick auf seine Größe sein, was bedeutet, dass wir dem System ganz einfach weitere Benutzer und Ressourcen hinzufügen können. Zweitens ist ein System geografisch skalierbar, wenn die Benutzer und die Ressourcen weit auseinanderliegen können. Drittens kann ein System administrativ skalierbar, also auch dann noch einfach zu verwalten sein, wenn es sich über viele unabhängige administrative Organisationen erstreckt.“	Tanenbaum & van Steen [2008:26]
„Skalierbarkeit bezeichnet die Fähigkeit eines Systems, wachsende quantitative Anforderungen durch Hinzufügen von Ressourcen auszugleichen, ohne dass eine Änderung von Systemkomponenten notwendig wird.“	Schill & Springer [2012:6]
„Das Verhalten der Leistung eines parallelen Programmes bei steigender Prozessoranzahl wird durch die Skalierbarkeit (engl. scalability) erfasst. Die Skalierbarkeit eines parallelen Programmes auf einem gegebenen Parallelrechner ist ein Maß für die Eigenschaft, einen Leistungsgewinn proportional zur Anzahl p der verwendeten Prozessoren zu erreichen.“	Rauber & Rünger[2012:179]

Skalierbarkeit: Einflussfaktoren

vgl. Coulouris et al. [2002:34ff.]

- Kosten für physische Ressourcen
- Art der genutzten Algorithmen (zentral vs. dezentral[1], linear vs. hierarchisch[2])
- Architektur: Aufteilung von Ressourcen auf verschiedene Komponenten (Caching, Replikation)
- Erschöpfung von Ressourcen
- Qualität der Evaluation/Antizipation der aktuellen/zukünftigen Bedarfssituation der Ressourcen eines VS

[1] Dezentrale Algorithmen sind in verteilten Systemen gemäß Tanenbaum & van Steen [2008: 28] im Vergleich zu zentralen Algorithmen besser skalierbar und zu präferieren. Eigenschaften dezentraler Algorithmen umfassen, dass 1.) kein Computer eine vollständige Information über den Systemstatus besitzt, 2.) Computer nur aufgrund lokaler Informationen entscheiden, 3.) der Ausfall eines Computers nicht den Algorithmus schädigt und 4.) nicht angenommen wird, dass es eine globale Uhr gibt (zu Zeitkonzepten und Problemen vgl. Abschnitt „Zeit“).

[2] Coulouris et al. [2002: 38] konstatieren, dass Algorithmen besser skalierbar sind, wenn sie hierarchische Strukturen anstelle von linearen Strukturen verwenden (vgl. „Beispiel: Verteilung“)

Skalierungstechniken

Tanenbaum & van Steen [2008:26]

1. Verbergen der Latenzzeiten der Kommunikation
2. Verteilung
3. Replikation (& Caching)

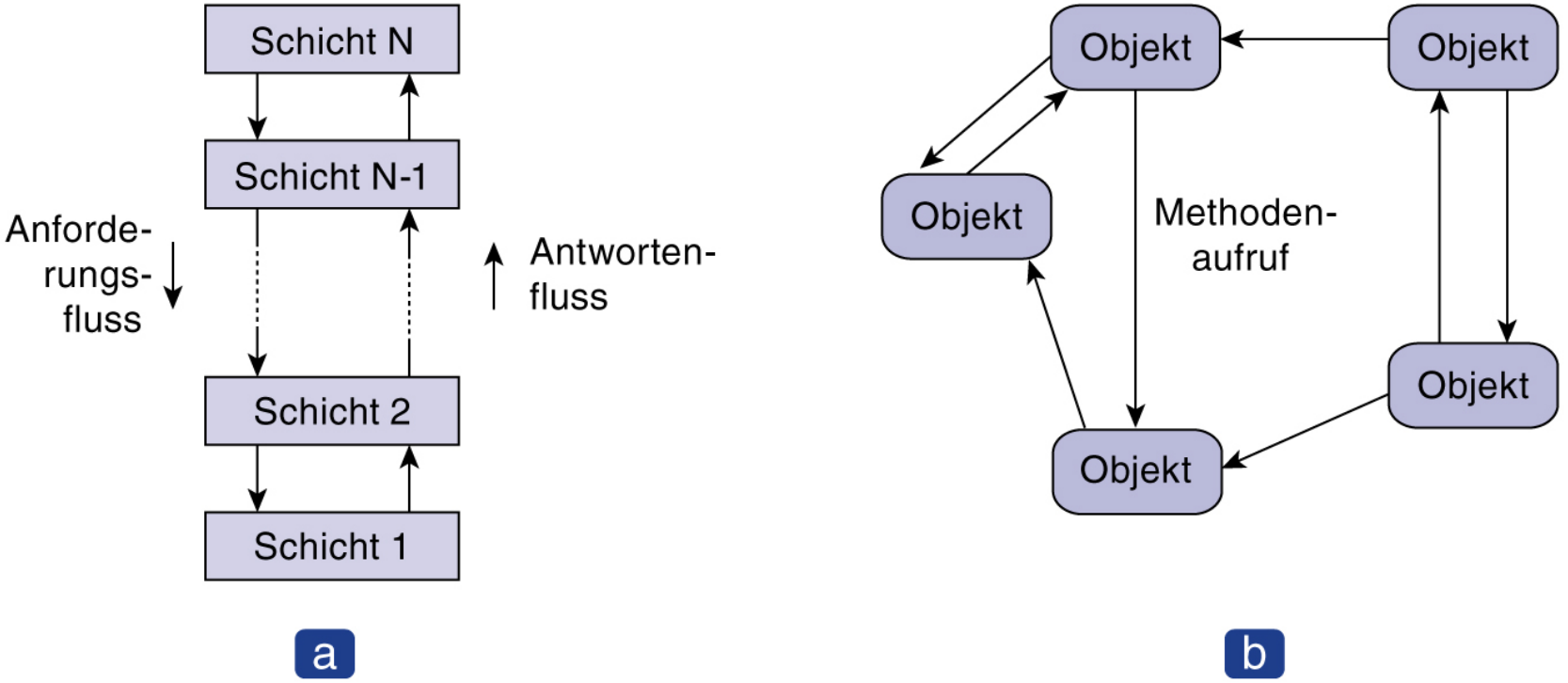
Verteilung

- Vertikal:
  - » Verteilung unterschiedlicher Funktionalitäten einer mehrstufigen Architektur auf mehrere Rechner (vgl. Abschnitt „III. Verteilte Architekturen (A))
  - » Jede Stufe ist auf einem anderen Rechner implementiert (Multi-Tier-System“)

- Horizontal:
  - » Verteilung von Funktionalität in logische Schichten auf einem Rechner
  - » Beispiel: IP-Implementierung auf verschiedenen Schichten (vgl. Abschnitt „IV. Netzwerke (B))

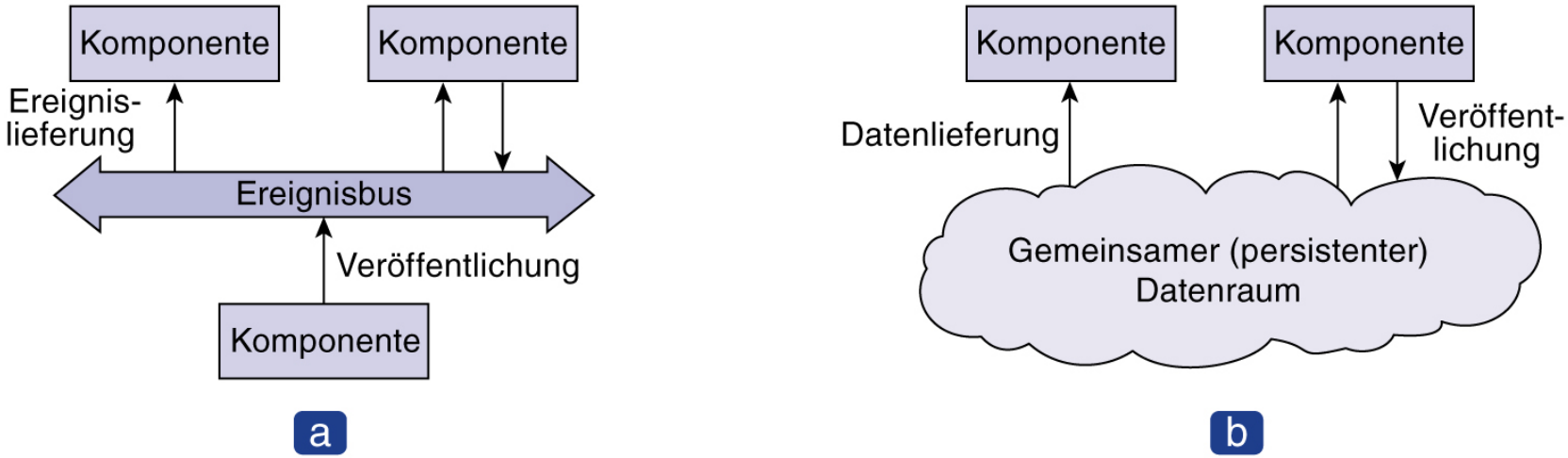
n-tier Architekturen

Architekturstil: geschichtet (a) vs. objektbasiert (b)



Quelle: Tanenbaum & van Steen [2008:54]

Architekturstil: Ereignisbasiert (a) vs. gemeinsamer Datenraum (b)



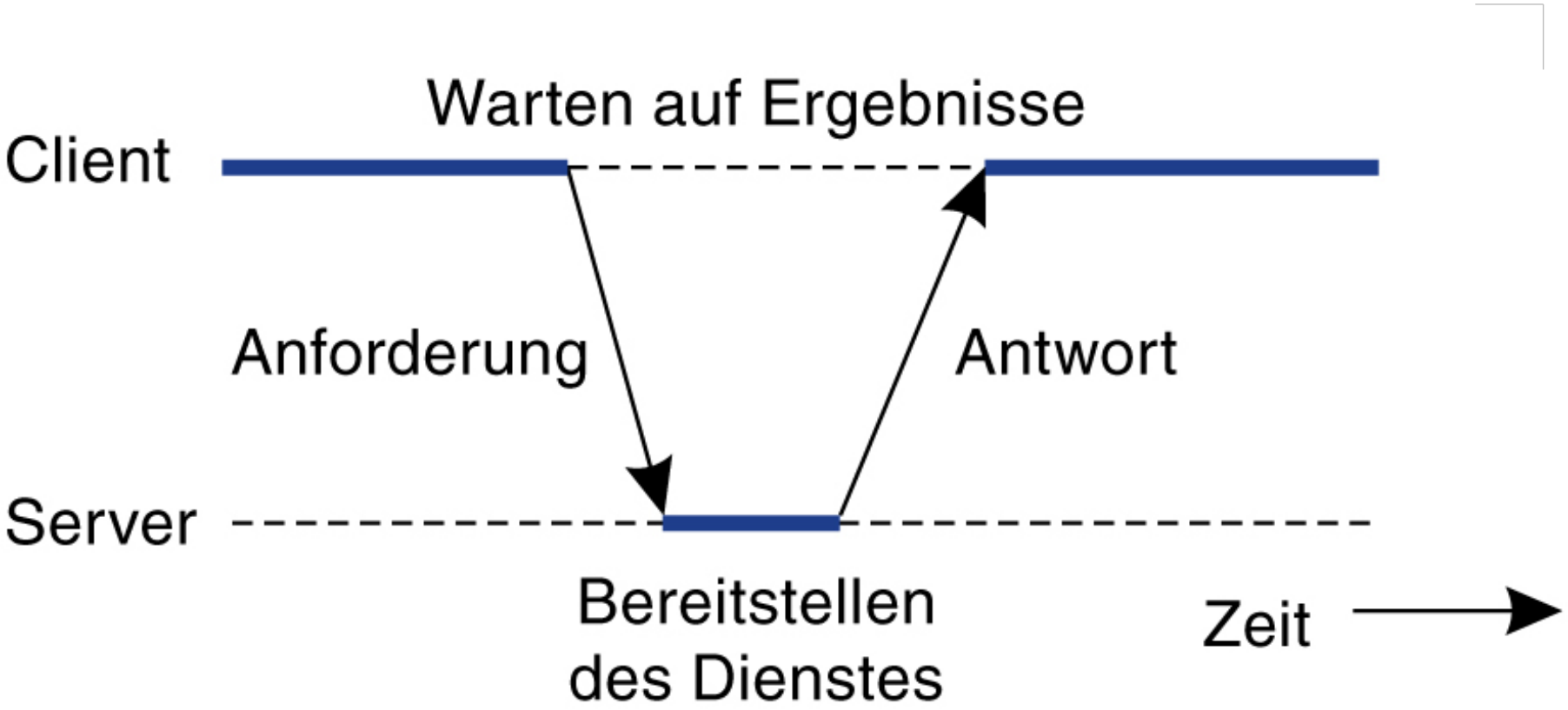
Quelle: Tanenbaum & van Steen [2008:54]

Fokus: geschichteter Architekturstil

Unterscheidung nach Schichten / Stufen („tiers“):

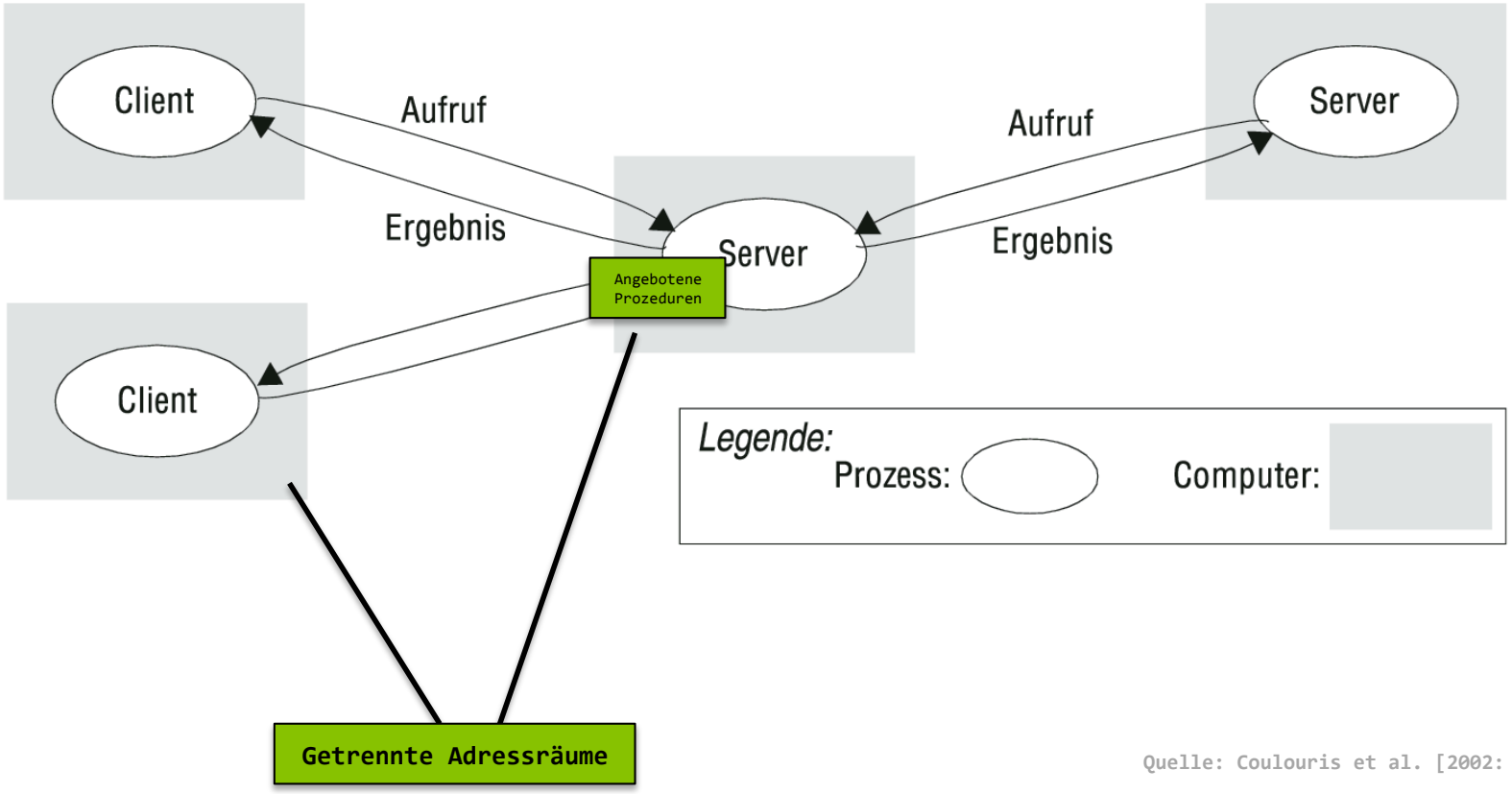
- Zweistufig: nur EIN Client und EIN Server
- Dreistufig (C*SS*; vgl. Abschnitt „IV. Entwurfsaspekte“)
  - Client: Benutzerschnittstelle (ggf. Vorverarbeitungsfunktionen) / Präsentationsschicht
  - Server I: Vorverarbeitung (Anwendungslogik/ Verarbeitungsschicht)
  - Server II: Datenverwaltung/Persistenzschicht

Allgemeine Zusammenarbeit zwischen einem Client und einem Server (2-tier)



Quelle: Tanenbaum & van Steen [2008:55]

3-tier-Architektur



Quelle: Coulouris et al. [2002: 52]

Klassifikation von C^+SS^+-Systemen (nach Rolle des Servers in mehrstufigen Architekturen)

[vgl. Bengel 2004: 68ff.; Bengel et al. 2008: 118ff.; Vertiefung in Abschnitt „IV. Entwurfsaspekte“]

Akürzung	Rolle Server I	Beschreibung
$C^+S_PS^+$	Proxy	Server kann mehrere Server vertreten
$C^+S_{Br}S^+$	Broker	Server kann zwischen Clients und Servern in 3. vermitteln
$C^+S_TS^+$	Trader	Server kann den am besten geeigneten Server in 3. 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 3.
$C^+S_KS^+$	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 3.

Zeitkonzepte

Eigenschaften von verteilten Systemen (im Vergleich zu zentralisierten Systemen)

- Keine gemeinsame Zeit (-> erschwerte logische Ordnung von Ereignissen)
- Unvorhersehbare Abläufe (nicht deterministisches Verhalten)
- Kein gemeinsamer Zustand
  - » Knoten in einem VS haben Zugriff auf Ihren eigenen Zustand, nicht jedoch auf den Zustand des gesamten Systems
  - » Konsequenz: Entscheidungen auf Basis des globalen Zustands eines Gesamtsystems
    - Lassen sich nicht hinreichend treffen
    - Versuche, dies zu realisieren basieren immer auf alten (ungültigen) Daten

➢ Problem: Unüberschaubarkeit führt bei Programmierung von verteilten Systemen zu fehlerhaften Anwendungen, z.B.: fehlerhafter Inhalt der transformierten Information, Vergessen nötiger Synchronisationspunkte, falsche Reihenfolgen

Logische Ordnung von Ereignissen [I]

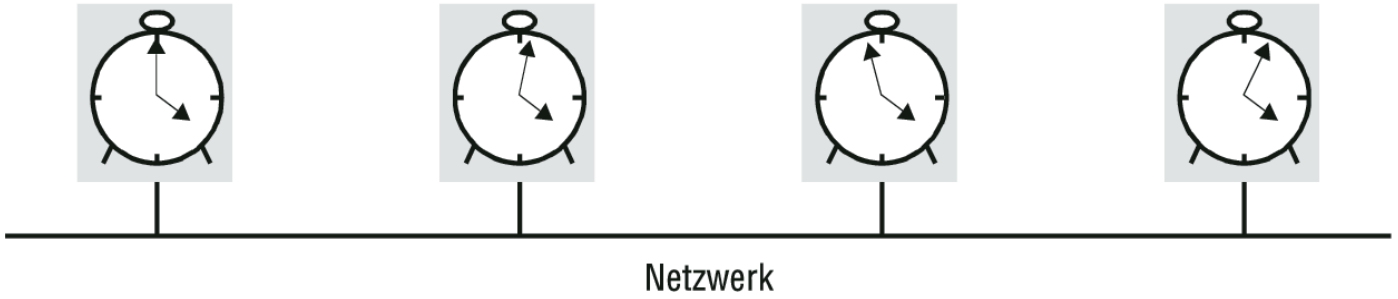
- Ereignis: Auftreten einer einzelnen Aktion, die ein Prozess während seiner Ausführung durchführt, z.B.:
  - Senden/Empfangen-Operationen von Nachrichten
  - Änderung des Zustands (state s) eines Prozesses P (Zustand beinhaltet Werte aller enthaltenen Variablen)
  - ✓ Der Verlauf („history“) eines Prozesses ist die Abfolge seiner Ereignisse, die darin stattfinden

- ✓ **Wie können Ereignisse logisch geordnet werden?**

  Antwort: Alle Maschinen einigen sich auf eine gemeinsame Zeit. Diese Zeit muss nicht mit der „realen“ Zeit übereinstimmen

Logische Ordnung von Ereignissen [II]
Physische Uhren

- Computer enthalten physische Uhren
- Uhr: elektronisches Gerät, welches die Schwingungen in einem Quarz bei einer festgelegten Frequenz zählt
  - » Teilung des Zählers
  - » Ablegen des Ergebnisses in Zählerregister



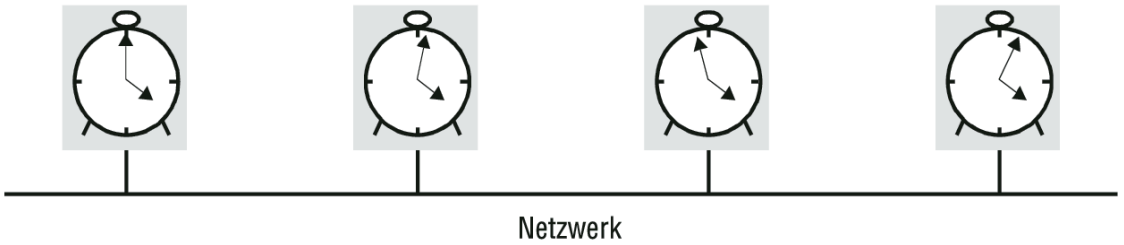
Quelle: Coulouris et al. [2002:452]

  - » Problem: Abweichungen

Logische Ordnung von Ereignissen [III]
Beispiel: Lokale Echtzeituhren

- Die Uhrzeit ergibt sich aus folgenden Uhren:
  - Hardwareuhr
    - Zähler
    - Taktgeber
    - Energiespeicher (Lithiumzelle): Uhr bleibt nicht stehen, wenn Gerät ausgeschaltet ist
  - Softwareuhr:
    - Auslesen des Wertes der Hardwareuhr $H_i(t)$ + Offset = $C_i(t)$
- ✓ $H_i(t) \neq C_i(t)$
- ✓ Bei hoher Uhrgenauigkeit kann $C_i(t)$ dazu verwendet werden, ein beliebiges Ereignis p_i mit einem Zeitstempel zu versehen
- ✓ Achtung: aufeinander folgende Ereignisse haben nur dann einen unterschiedlichen Zeitstempel, wenn das Intervall zwischen den Aktualisierungen des Uhrwerts (Uhrauflösung) kleiner ist als das Intervall zwischen den aufeinander folgenden Ereignissen

Logische Ordnung von Ereignissen [IV]
Uhren: Synchronisierungsfehler/Abweichung



Quelle: Coulouris et al. [2002:452]

- Uhren stimmen nicht immer überein
  - » Differenz = Synchronisierungsfehler
  - » Abweichgeschwindigkeit: Änderung des Offsets zwischen Uhr und einer Referenzuhr R pro Zeitintervall gemessen mit R

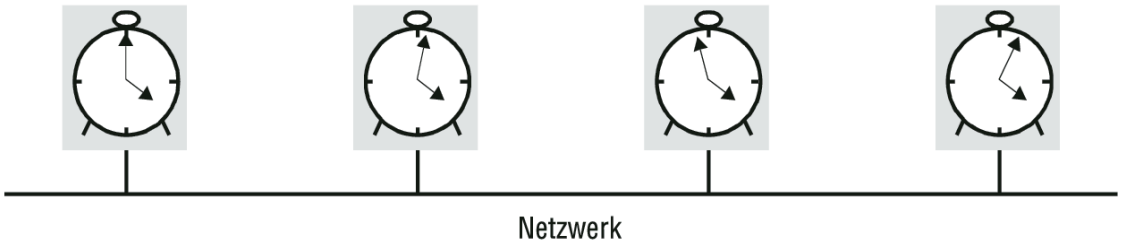
    Beispiel: Abweichgeschwindigkeit von Quarzuhren beträgt ca. 10^{-8} Sekunden (Differenz von einer Sekunde alle 100.000.000 Sekunden)

- Einflüsse:
  - » Divergierende Zähl-/Oszillationsgeschwindigkeiten
  - » Temperatur
- Notwendigkeit zur Synchronisierung von Uhren mit Standardreferenz, z.B.: Coordinated Universal Time (UTC), welche von Stationen empfangen/abgerufen werden kann

Coordinated Universal Time

- Synonym: Universal Temps Coordiné (UTC)
- Gültige Weltzeit seit 1972
- Keine Berücksichtigung von Zeitschwankungen im Zusammenhang mit der Erdrotation, sondern Einfügung von Schaltsekunden (leap seconds)
- Zusammenführung von astronomischer Zeit und Atomzeit
- Basiert auf atomaren Oszillatoren (Abweichgeschwindigkeiten um 10^{-13} Sekunden)
- UTC kann von Stationen empfangen/abgerufen werden

Synchronisierung [I]

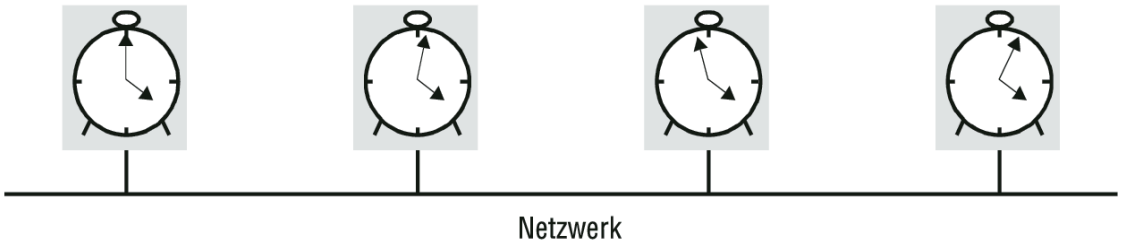


Quelle: Coulouris et al. [2002:452]

Wie können Ereignisse in zwei Prozessen innerhalb eines VS unter Verwendung physischer Uhren synchronisiert werden?

- Möglichkeiten:
  - » Interne Synchronisierung:
    - Verwendung der lokalen Uhr ohne Synchronisation mit externer autoritativer Zeitquelle
    - Zeit kann von externer Zeit abweichen (Uhren müssen nicht genau sein, um korrekt zu sein)
  - » Externe Synchronisierung

Synchronisierung [II]



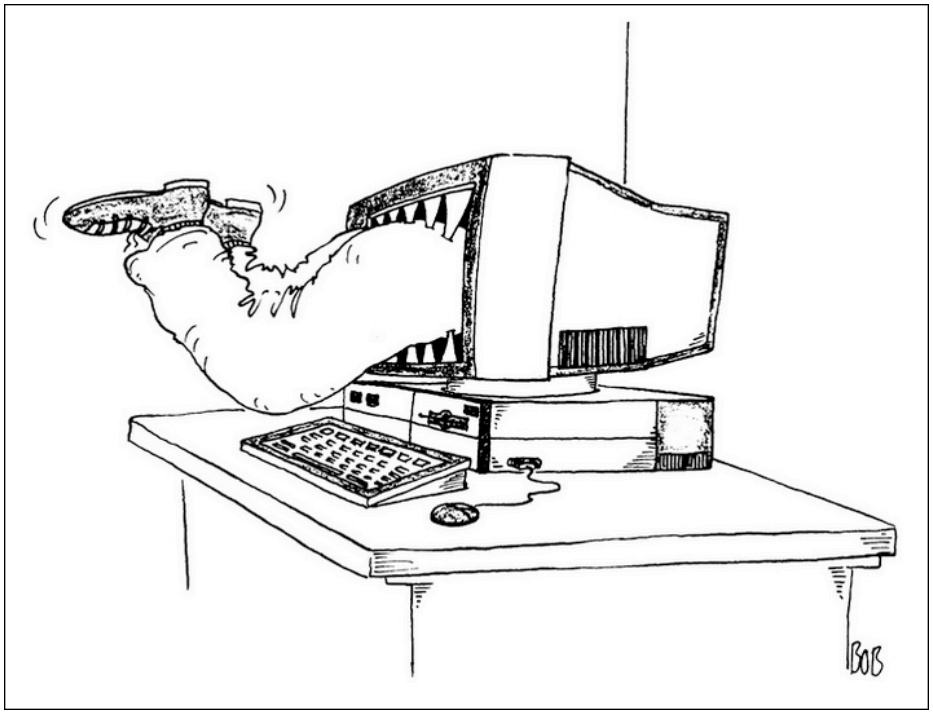
Quelle: Coulouris et al. [2002:452]

- Korrektheitsbedingungen:
  - Eine Uhr muss monoton laufen, d.h. sie darf nicht rückwärts laufen
  - Ihre Abweichgeschwindigkeit muss innerhalb eines Synchronisierungsintervalls begrenzt sein
  - An Synchronisierungspunkten muss es möglich sein, den Uhrwert springen zu lassen
- Fehlerhafte Uhren: erfüllen die Korrektheitsbedingungen nicht
  - Absturz: Uhr tickt nicht mehr
  - Zufälliger Fehler:
    - Monotonie-Problem (Beispiel: Y2K-Bug – 31.12.1999 + 1 Tag = 01.01.1900)
    - Hohe Abweichungsgeschwindigkeiten durch schwache Batterie
  - Aber: Uhren müssen nicht genau sein, um korrekt zu sein

Übungseinheit 2: Lernziele

- ✓ Vertiefung von grundlegenden Aspekten verteilter Systeme (Grundlagen für die Bearbeitung des Übungsblattes):
  - ✓ Skalierbarkeit
  - ✓ Verteilung
  - ✓ n-tier-Architekturen
  - ✓ Zeitkonzepte
- ✓ Beginn der Bearbeitung von Übungsblatt 2

Übungsblatt 2



Source: https://baubeccav.files.wordpress.com/2011/03/christian-born_cartoon_computer21.jpg [last accessed: 9 III 15]

Lösungsvorschläge: vgl. Teil B

Viel Erfolg!

Literatur

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

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

Melzer, I. (2010) Service-orientierte Architekturen mit Web Services; 4. Auflage; Heidelberg: Spektrum.

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.