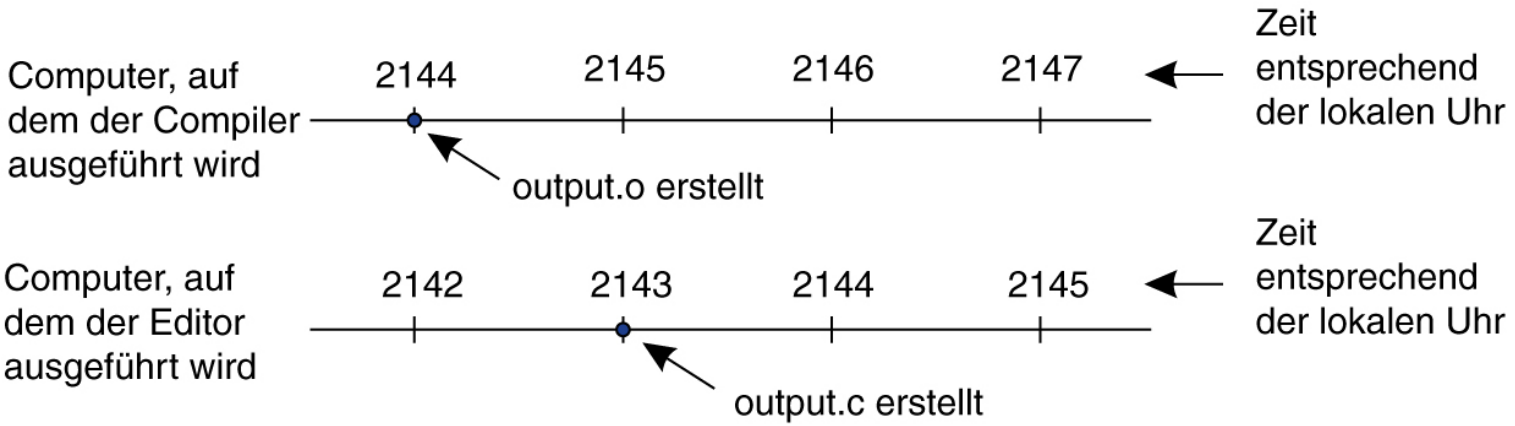
Struktur von Kapitel IV – Architekturen & Algorithmen

A Voraussetzungen, Definitionen
B Verfahren für Physische Uhren
C Verfahren für Logische Uhren

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Grundlagen der Synchronisierung



- **Motivation:** Häufig gibt es Abhängigkeiten zwischen Ereignisse bzw. Aufgaben – Z.B. bei "make".

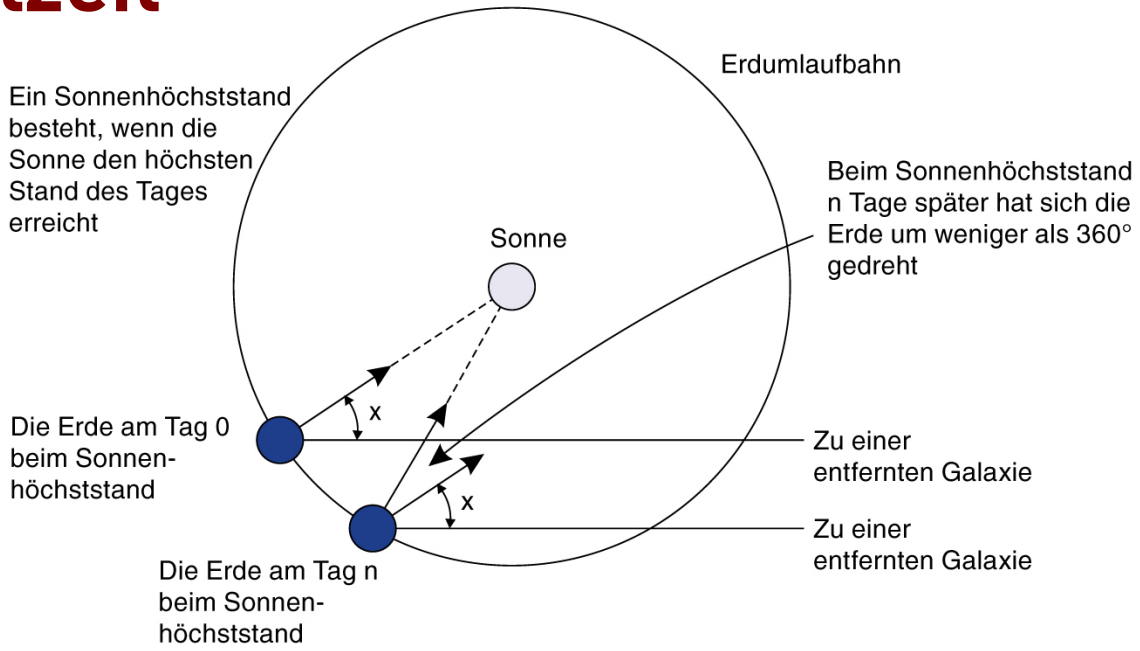
  Die Kommunikation bestimmt nicht immer die Synchronisierung!

- **Logische Uhr:** Wenn die interne Konsistenz der Uhr ausreicht und die Differenz zur absoluten Zeit unerheblich ist.
- **Physiche Uhr:** Wenn zusätzlich zur logischen Uhr dazukommt, dass die Zeit nur um einen bestimmten Betrag von der absoluten Zeit abweichen darf.

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Koordinierte Universalzeit

- Die Erdrotation verlangsamt sich jedoch stetig, also ist die Zeitangabe auf astronomischer Basis ungenau.
- Seit 1958 gibt es die Internationale Atomzeit (kurz TAI):
  1 Sekunde = 9.192.631.770 Übergangsperioden in Cäsium-133 Atomen.
- Seit 1967 gibt es die Koordinierte Universalzeit (UTC): berücksichtigt Schaltsekunden.
- UTC wird per Kurzwelle bzw. per Satellit ausgestrahlt.
- Die Übertragung mittels GEOS (Geostationary Environmental Operational Satellites) hat eine Genauigkeit von ca. 0,1 Millisekunden...
- ...über das GPS (Global Positioning System) eine Genauigkeit von ca. 1 Millisekunde.



- **Denkbare Lösung des Zeitproblems:** jeden Rechner wird mit Satellitenempfänger für UTC ausgerüstet....
- ...leider aus verschiedenen Gründen nicht machbar (welche?)...
- ...aber *ein Rechner* aus einer Gruppe könnte einen Satellitenempfänger haben (würde das helfen? Ausreichen?)

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

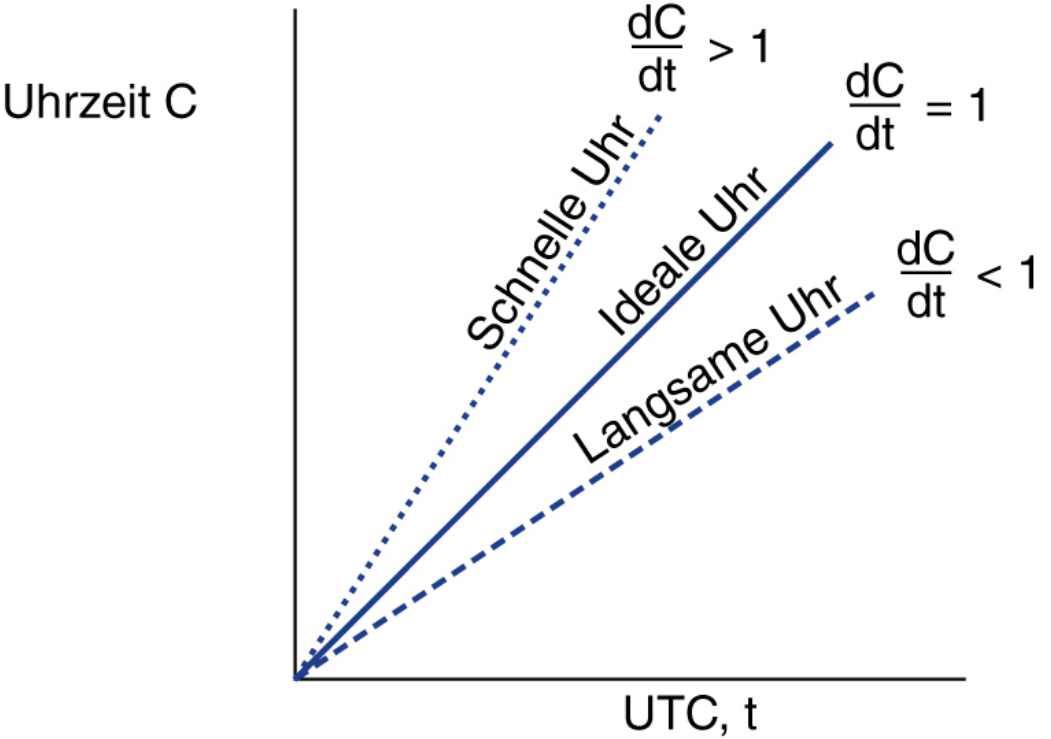
Zeitdrift und Synchronization

- Null oder mehr Rechner werden mit Satellitenempfänger ausgestattet,
- *Alle* Rechner sind auf eine gemeinsame physikalische Zeit zu synchronisieren.
- Sei C = Zeit nach der Uhr eines Rechners; sei t = UTC.

  Also soll dC/dt = 1 idealerweise.
- Wenn dC/dt ≠ 1 gibt es **Zeitdrift**.
- Sei Konstante ρ = die maximale Abweichung (z.B. laut Hersteller)

$$1-\rho \le dC/dt \le 1+\rho$$

- Falls zwei Uhren im schlimmsten Fall entgegengesetzt driften, können sie Δt nach ihrer Synchronisation maximal um 2ρΔt auseinander liegen.



- Wenn also sichergestellt sein muss, dass zwei Uhren nicht mehr als δ auseinander laufen, so müssen sie mindestens alle δ/2ρ Sekunden synchronisiert werden.

Hinweis:
ρ = „rho“;
Δ = „delta“,
δ = „sigma“

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Wie korrigiert man eine System-Uhr?

Q: Was sollen wir machen, wenn wir entdecken, dass die Uhr zu langsam gewesen ist?

A: Vorstellen.

Q: Was sollen wir machen, wenn wir entdecken, dass die Uhr zu schnell gewesen ist?

A: *Nicht nachstellen!* (warum?)

Q: Stattdessen?

A: Stetig verlangsamen.

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Struktur von Kapitel IV – Architekturale Themen

I Einleitung

II Grundlegende Kommunikationsdienste

III Middleware

IV Architekturen & Algorithmen

- A Synchronisierung
- B Konsistenz und Replikation
- C Fehlertoleranz

V Beispiele bzw. Dienste

- A Verteilte Dateisysteme
- B Namensdienste

VI Sicherheit & Sicherheitsdienste

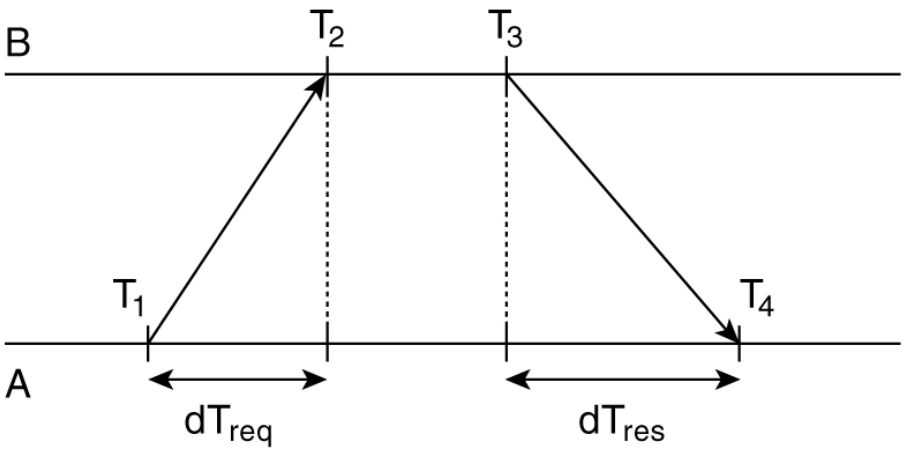
VII Zusammenfassung

- A Voraussetzungen, Definitionen
- B Verfahren für Physische Uhren
- C Verfahren für Logische Uhren

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Verfahren von Christian

- Es gibt genau einen Rechner mit Satellitenempfänger,
- *Alle* Rechner synchronisieren sich mit dem Zeit-Server – wie folgt:
  - regelmäßig ($t \leq \delta/2\rho$ Sekunden) wird Nachricht mit C_{Uhr} an Zeitserver gesendet.
  - der Zeitserver antwortet mit C_{UTC}.
  - Der Sender setzt in **erster Näherung** seine Uhr auf C_{UTC}.
  - Ist $C_{UTC} > C_{Uhr}$, dann wird $C_{Uhr} = C_{UTC}$ übernommen.
  - Ist $C_{UTC} < C_{Uhr}$, dann wird C_{Uhr} stetig verlangsamt.



- Danach wird die eigene Uhr um die Hälfte der Zeit angepasst, die die Nachricht zum und vom Zeitserver gebraucht hat (Roudtrip-Delay): $(T_4 - T_1)/2$.

Problem: Der ***zentrale Zeitserver***.
Bei Ausfall kann keine Synchronisation stattfinden.

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

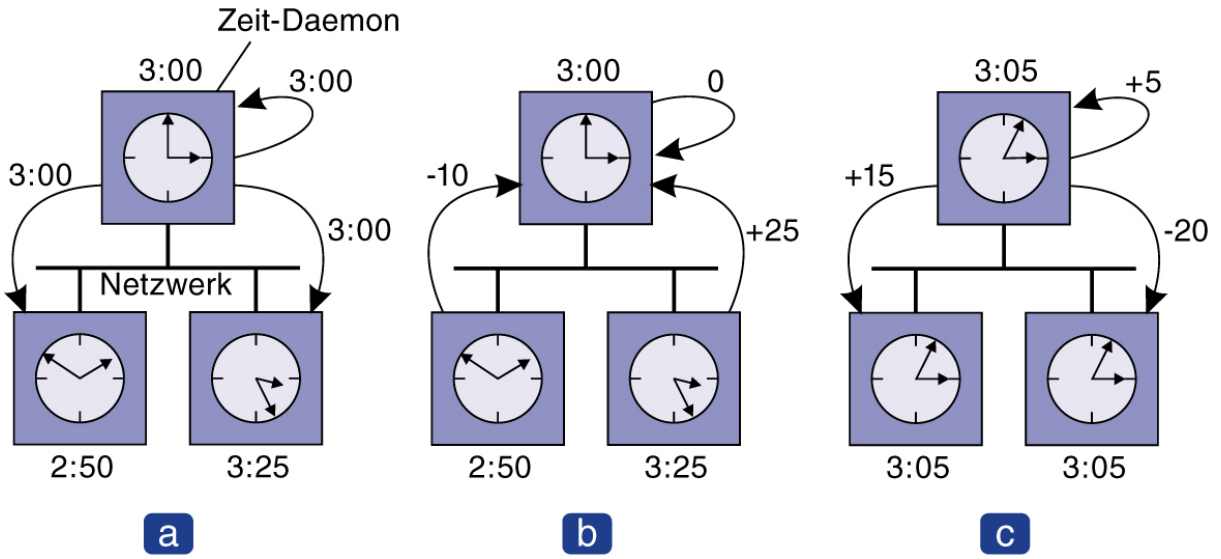
Der Berkley-Algorithmus

- Es muss keinen Rechner mit Satellitenempfänger verbunden sein,
- *Alle* Rechner synchronisieren sich mit einem Zeit-Dämon – wie folgt:

  a) Der Dämon fragt periodisch alle Rechner nach ihrer Zeit.

  b) Aus allen Antworten wird die durchschnittliche Zeit ermittelt.

  c) Diese Durchschnittszeit wird allen Rechnern mitgeteilt. Alle Rechner (auch der Zeitdämon) stellen dann ihre Uhr: sie erhöhen die Zeit oder verlangsamen die Uhr bis eine Angleichung erfolgt ist.

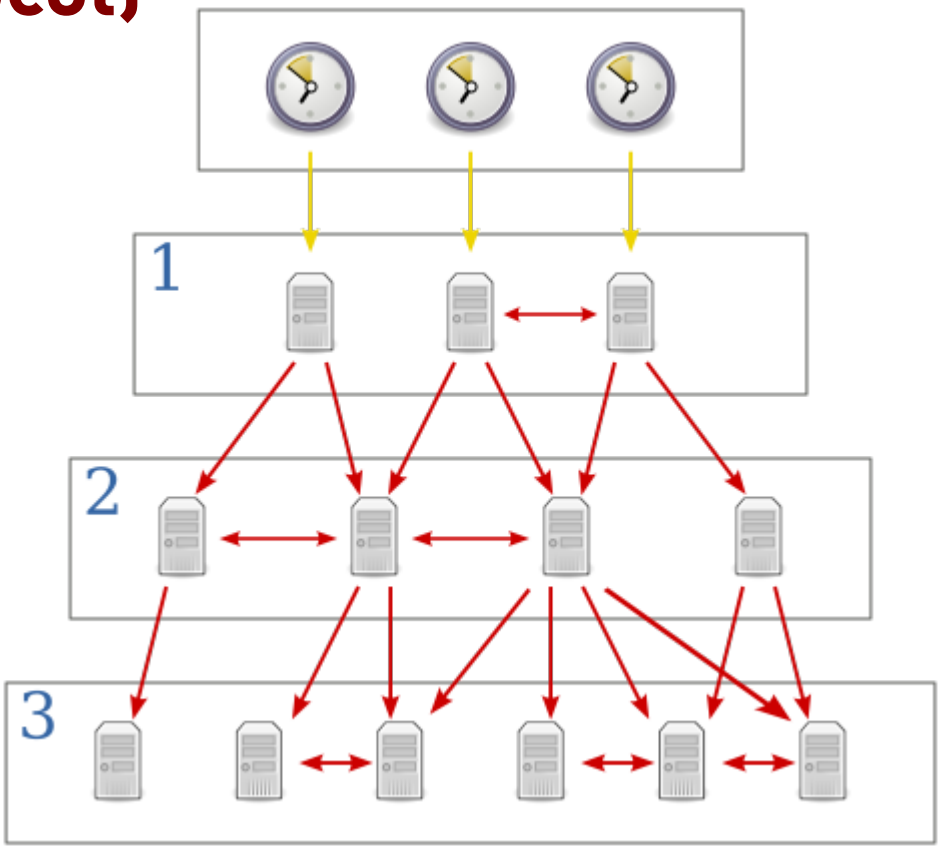


Problem: Der ***zentrale Zeit-Daemon***. Bei Ausfall kann keine Synchronisation stattfinden (allerdings könnte ein anderen Rechner einspringen).

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

NTP (Network Time Protocol)

- Manche Rechner sind mit Satellitenempfänger verbunden, andere nicht...
- *Jeder* Rechner hat einen **Zeit-Dämon** – der entweder *Server, Client* oder *Peer* sein kann (gegenüber andere Dämonen).
- Zeit-Dämonen sind hierarchisch organisiert.
- Ein Stratum = Hierarchie-ebene. Stratum *n* korrigiert Stratum *n*+1.
- Jeder NTP-Daemon kann so konfiguriert werden, dass er mehrere unabhängige Referenzzeitquellen kennt.
- Von allen möglichen Zeitquellen sucht er sich dann die beste aus.



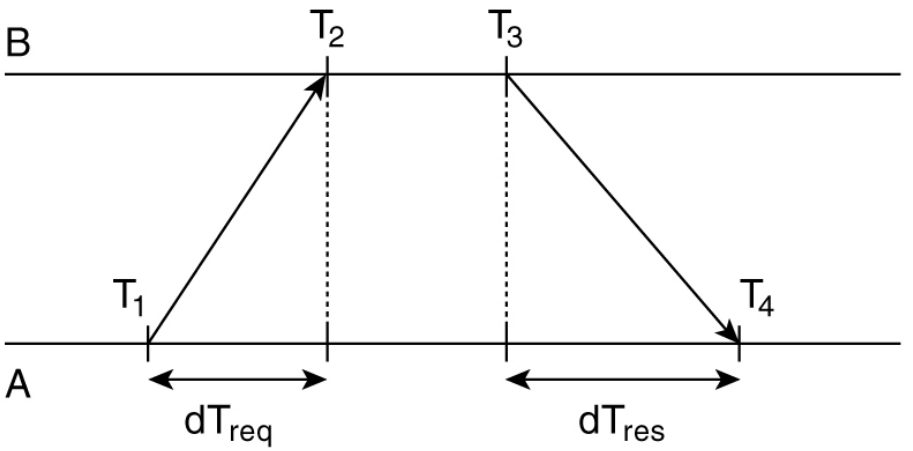
- Kriterien sind Erreichbarkeit, beste Stratum-Wert, sowie die Antwortzeiten (Delay) und die Schwankungsbreite der Antwortzeiten (Jitter).
- Wenn die ausgewählte Referenzzeitquelle nicht mehr erreichbar sein sollte, wählt sich der NTP-Daemon die nächst beste Referenzzeitquelle aus (und ggf. korrigiert seinen eigenen Stratum-Wert).

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

NTP-Korrektur

- Delay $= \delta$
  $= ((T_4 - T_1) - (T_3 - T_2))$
- Korrektur $= \Theta$
  $= ((T_4 - T_3) + (T_2 - T_1)) / 2$

Hinweis:
δ = „delta“,
Θ = „theta“



- Die letzte 8 Paaren (Θ, δ) werden gespeichert.
- Die Paar mit der kleinste δ wird verwendet – wobei eine Paar nur einmal verwendet werden darf, und die nächste Paar, die verwendet wird, neuer sein muss, als die letzte Paar, die verwendet wurde

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Struktur von Kapitel IV – Architekturale Themen

I Einleitung

II Grundlegende Kommunikationsdienste

III Middleware

IV Architekturen & Algorithmen

- A Synchronisierung
- B Konsistenz und Replikation
- C Fehlertoleranz

V Beispiele bzw. Dienste

- A Verteilte Dateisysteme
- B Namensdienste

VI Sicherheit & Sicherheitsdienste

VII Zusammenfassung

- A Voraussetzungen, Definitionen
- B Verfahren für Physische Uhren
- C Verfahren für Logische Uhren

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Logische Uhren

- Häufig brauchen wir nicht die UTC, wir brauchen nur Ereignisse in einer Reihenfolge zu bringen.
- D.h. die Uhren der Rechner können ungleich laufen, sowohl bzgl. der Zeit als auch der Geschwindigkeit.
- Dazu wird die Relation „happened before“ verwendet: der Ausdruck

  $a \rightarrow b$

  – „a tritt vor b ein“ – bedeutet, dass alle Prozesse darin übereinstimmen, dass zuerst das Ereignis a eintritt und danach das Ereignis b.

Definition („happened before“, „→“)

- Sind a und b Ereignisse im **gleichen Prozess** und a ereignet sich vor b, dann gilt:

  $a \rightarrow b$ (a happened before b).

- Ist a ein **Sendeereignis** in einem Prozess und b das zugehörige **Empfangsereignis** in einem anderen Prozess, dann gilt:

  $a \rightarrow b$ (a happened before b).

- (Sonst besteht keine Relation zwischen a und b).
- Damit entspricht die Relation → der **kausalen Ordnung** der Ereignisse.

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Logische Uhren (2)

Die Relation → besitzt folgende Eigenschaften:

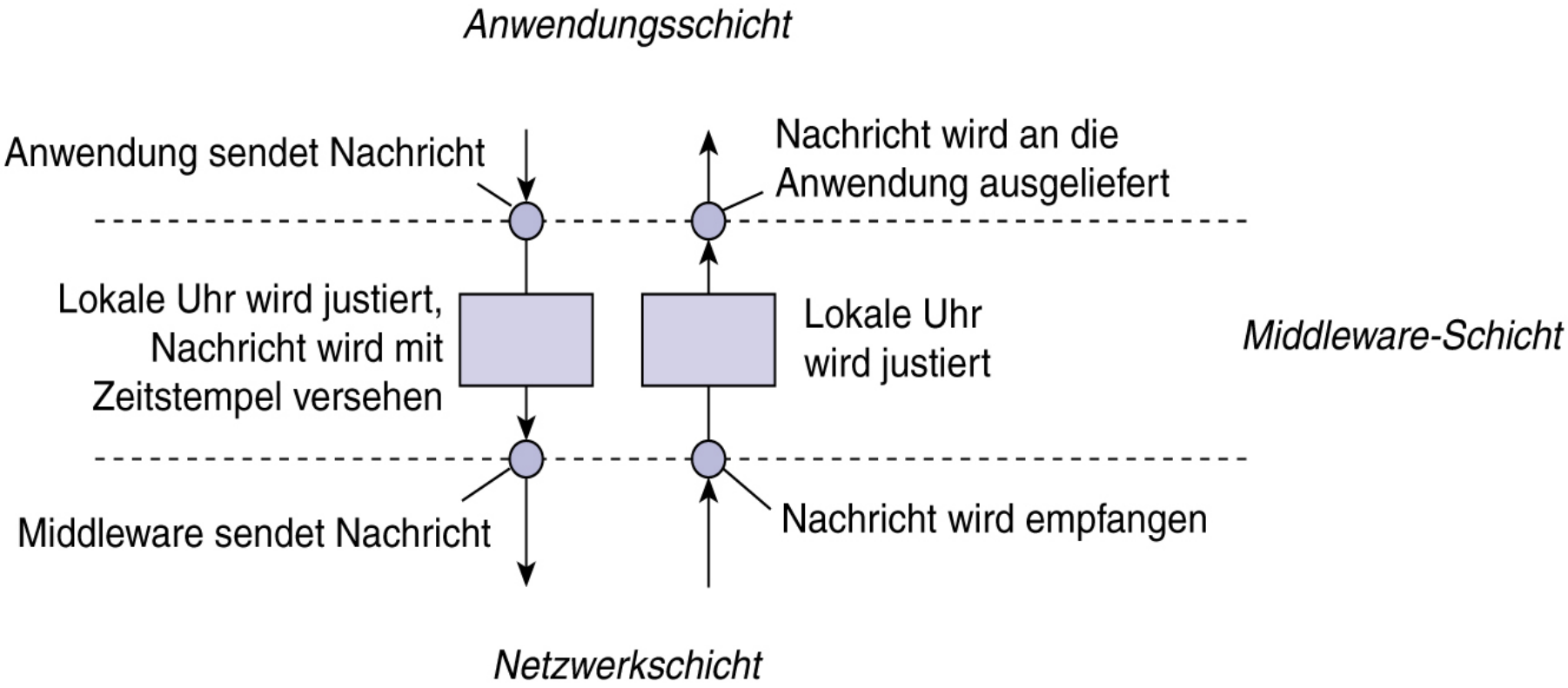
- **Transitivität**, d.h. wenn a→ b und b→ c gilt, dann gilt auch a→ c
- unabhängige Ereignisse (es gilt weder a → b noch b→ a) heißen **nebenläufig** (*concurrent*) und werden als a||b geschrieben
- Happened-before definiert eine **partielle Ordnung** auf den Ereignissen

Wir wollen jedem Ereignis *a* eine Zeit C(*a*) zuweisen, mit der alle Prozesse einverstanden sind.

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

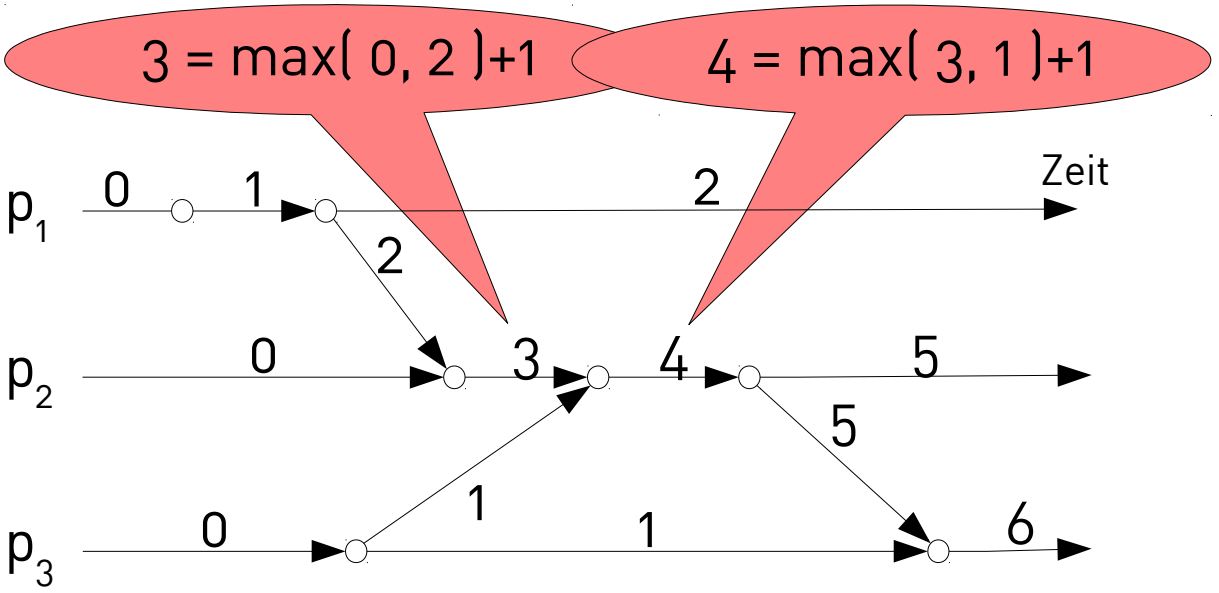
Algorithmus von Lamport

Die Standorte der logischen Uhren von Lamport in verteilten Systemen



h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Algorithmus von Lamport (2)



a) Drei Prozesse, von denen jeder eine eigene Uhr hat.

(b) Der Algorithmus von Lamport definiert die logische Zeit:

Warnung: Es gibt auch andere Versionen des Algorithmus in der Literatur – inkl. alle vorherige Versionen meines Skripts (bis Summer 2016).

Verfahren aus Sicht Prozess *p*

- Lokale Ereignis:
  C_P := C_P + 1;
- Sendeereignis:
  C_P := C_P + 1;
  send(msg, C_P);
  Das heißt: Sende C_P (Zeitstempel) mit!
- Empfangsereignis:
  (msg, C_Q):= receive();
  C_P := max(C_P,C_Q)+1;
  Das heißt:
  Empfange Zeitstempel C_Q;
  Setze Uhr großer als sowohl lokale Zeit als auch Zeitstempel (C_Q).

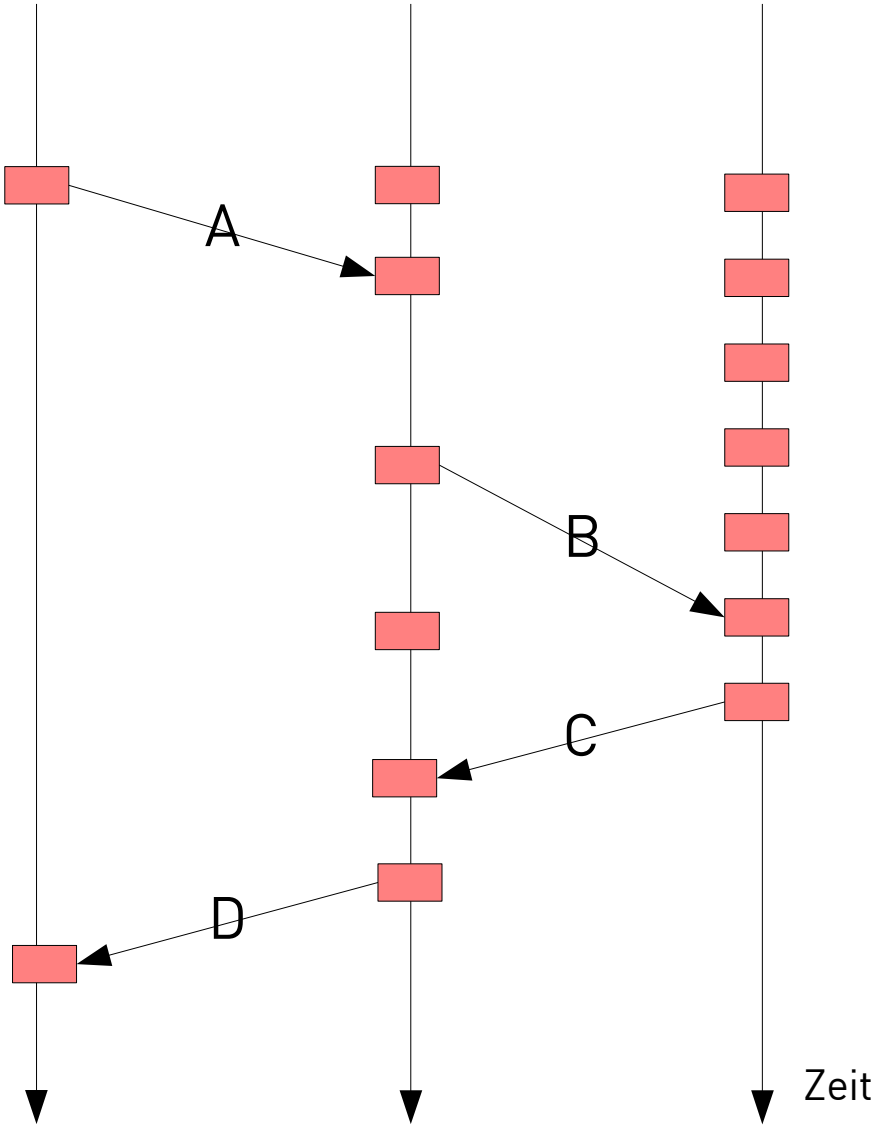
h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Algorithmus von Lamport - Übung

Gegeben seien 3 Programme, die jeweils eine eigene logische Uhr gemäß des Algorithmus von Lamport implementiert haben.

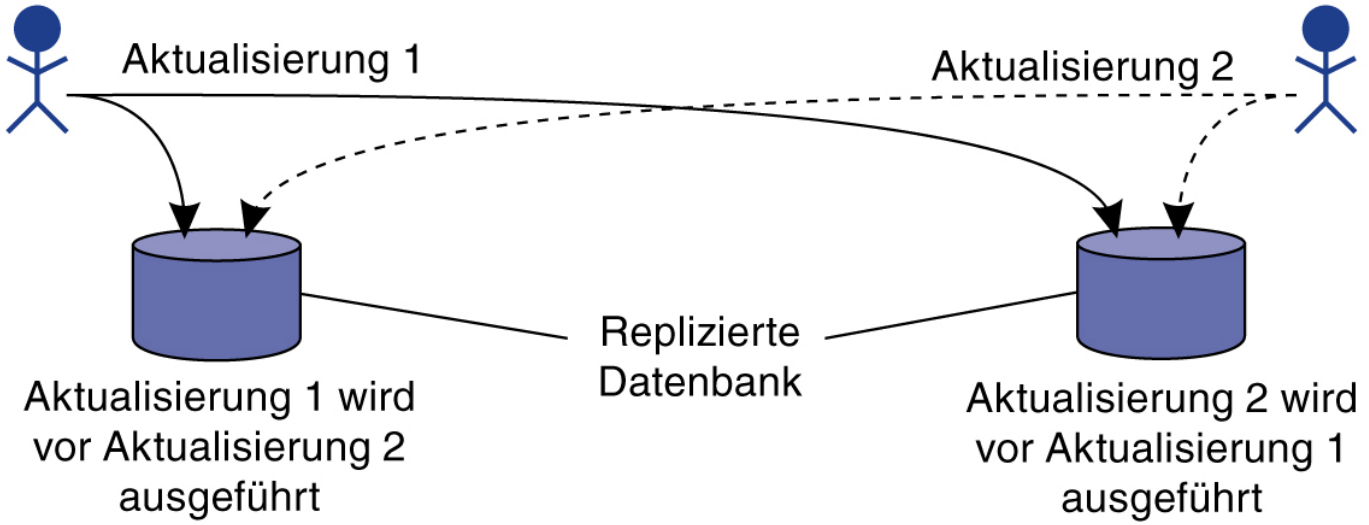
Die Programme tauschen Nachrichten (A-D) gemäß der Abbildung aus.

Zeigen Sie in der Abbildung, wie sich die Zeit der logischen Uhr pro Programm verändert.



h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Lamport Uhren und vollständig geordnetes Multicasting



Herausforderung;

- Die Reihenfolge, in der Nachrichten an die Anwendung geliefert wird, ist wichtig – und nicht vom Netzwerk sicher gestellt.
- Middleware muss Nachrichten überall in *einer* Reihenfolge an die Anwendungen ausliefern.

Lösungsskizze: Nachrichten werden in einer Warteschlange **gehalten**, nach Zeitstempel sortiert.

Problemen mit der Lösung:

- Es kann zwei Nachrichten mit demselben Zeitstempel geben. Ein *Tie-Breaker* muss her: (z.B. Uhr´ := „Uhr**,**Prozess-ID“)
- Wie lange soll (darf, muss) ein Nachrichten zurück gehalten werden?

Hier gibt es mehrere „richtige“ Antworten – es hängt von Ihren **Annahmen** ab. Machen Sie Ihre Annahmen **explizit.**

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Logische Uhren und kausal geordnetes Multicasting

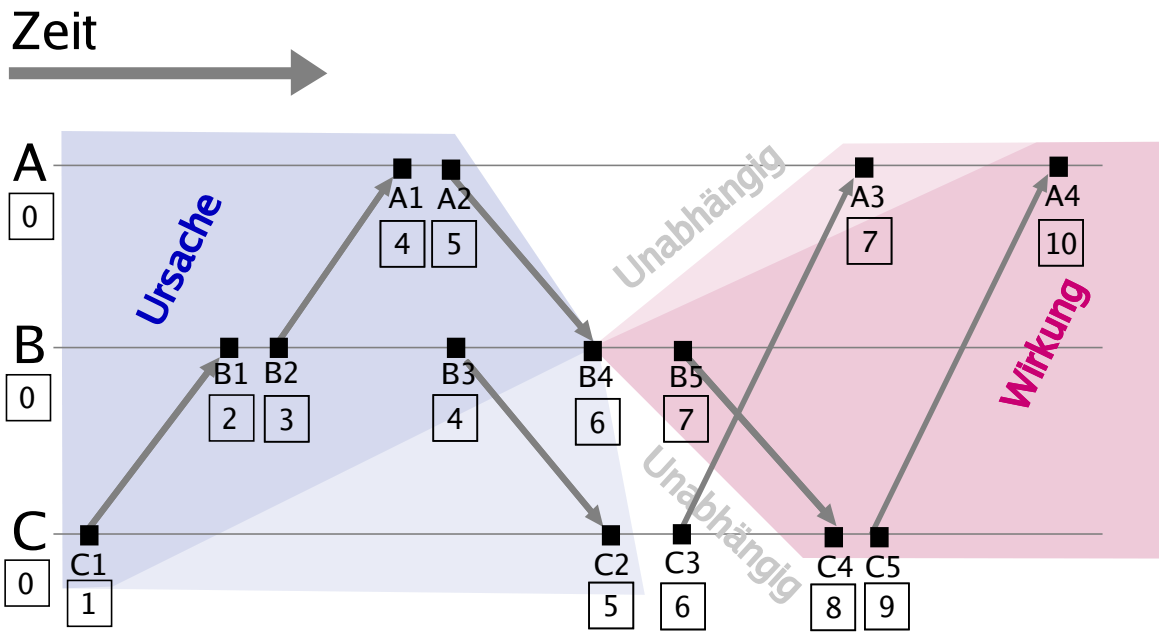
Was sagt uns Lamport'sche Uhren?

Wenn a → b, ist C(a) < C(b).
Wenn C(a) = C(b), ist a || b.

Wenn C(a) < C(b), ist entweder a → b oder a || b.

Also: C(x) gibt uns mehr Informationen, als die Beziehung → !

Für vollständig geordnetes Multicasting OK, aber manchmal brauchen wir „nur" *kausal* geordnetes Multicasting.

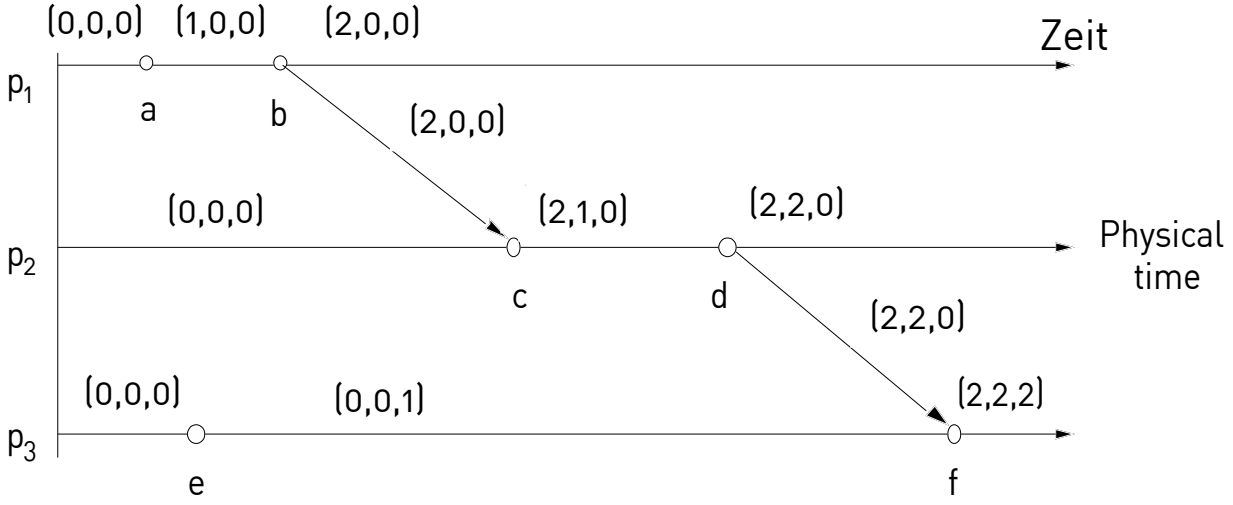


Bildquelle: https://commons.wikimedia.org/wiki/File:Lamport-Uhr.svg
Für alle Ereignisse *x* in dem hellblauen Bereich, ist C(x) < C(B4), aber *x* und B4 sind nebenläufig; *x* || B4. Ähnliches gilt für *y* in dem hellroten Bereich: C(B4) < C(*y*) aber B4 || *y*.

Anders gefragt:
Gibt es logische Uhren, die uns sagen, ob a → b oder b → a oder a || b ?

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Vektor-Uhren (1)



a) Jeder der *n* Prozessen hat eine Uhr, die *n* Elemente hat (ein Vektor der Länge *n*).

b) Gegeben zwei Vektoren *A* & *B*, gilt:

- $A = B$ gdw $A[p] = B[p]$ für alle p
- $A < B$ gdw $A[p] < B[p]$ für alle p
- $A > B$ gdw $A[p] > B[p]$ für alle p

Verfahren aus Sicht Prozess *p*

- *Für alle* Ereignisse, zuerst:
  $VC_p[p] := VC_P[p] + 1;$
- **Danach**:

  Sendeereignis?
  Sende Zeitstempel VC_p mit

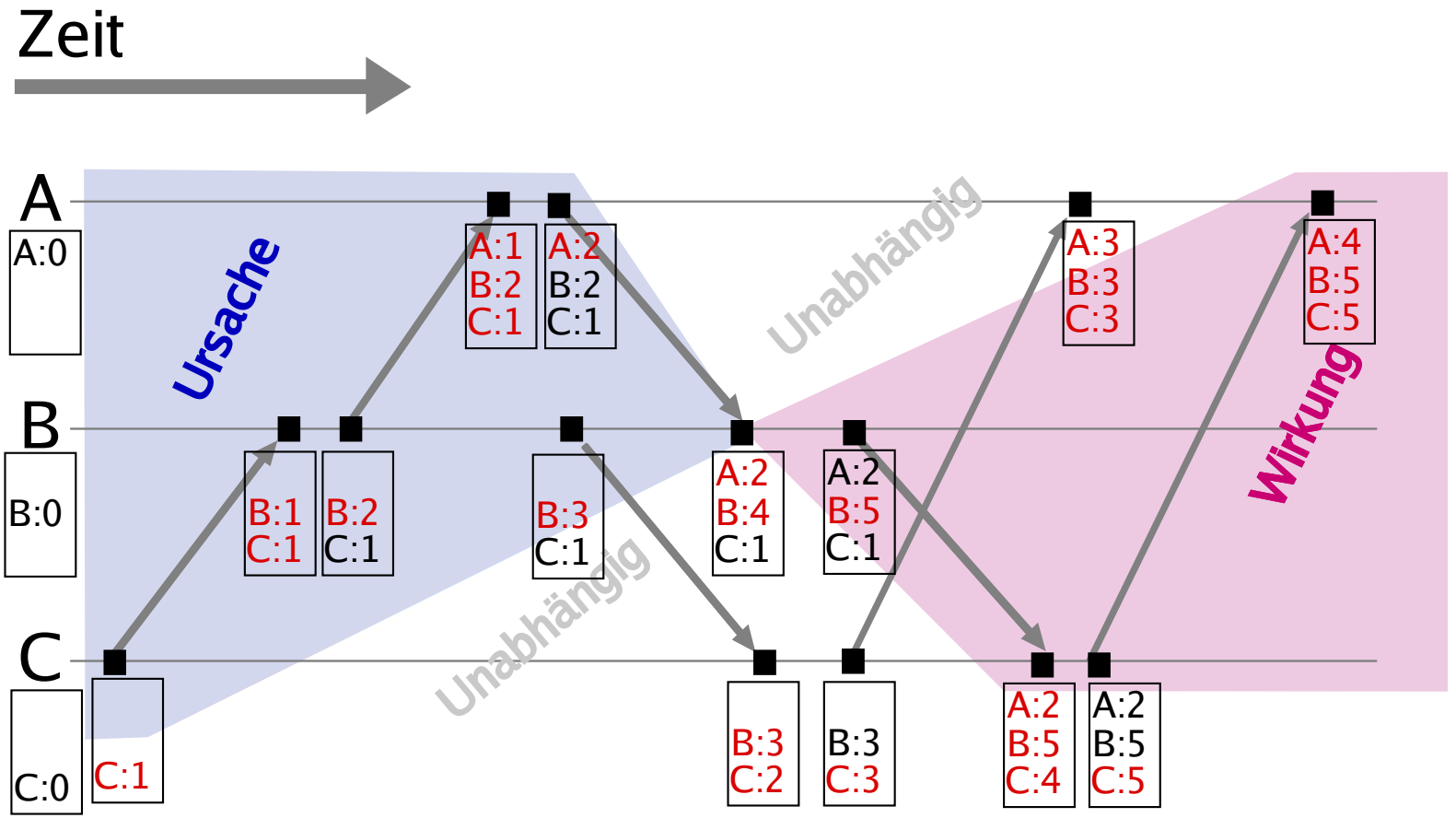
  Empfangsereignis?
  receive(msg, Vcq);
  $VC_p[q] := \max(VC_p[q], VC_{msg}[q])$

  für alle q!

Warnung: Es gibt auch andere Versionen des Algorithmus in der Literatur – inkl. alle vorherige Versionen meines Skripts (bis Summer 2016).

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

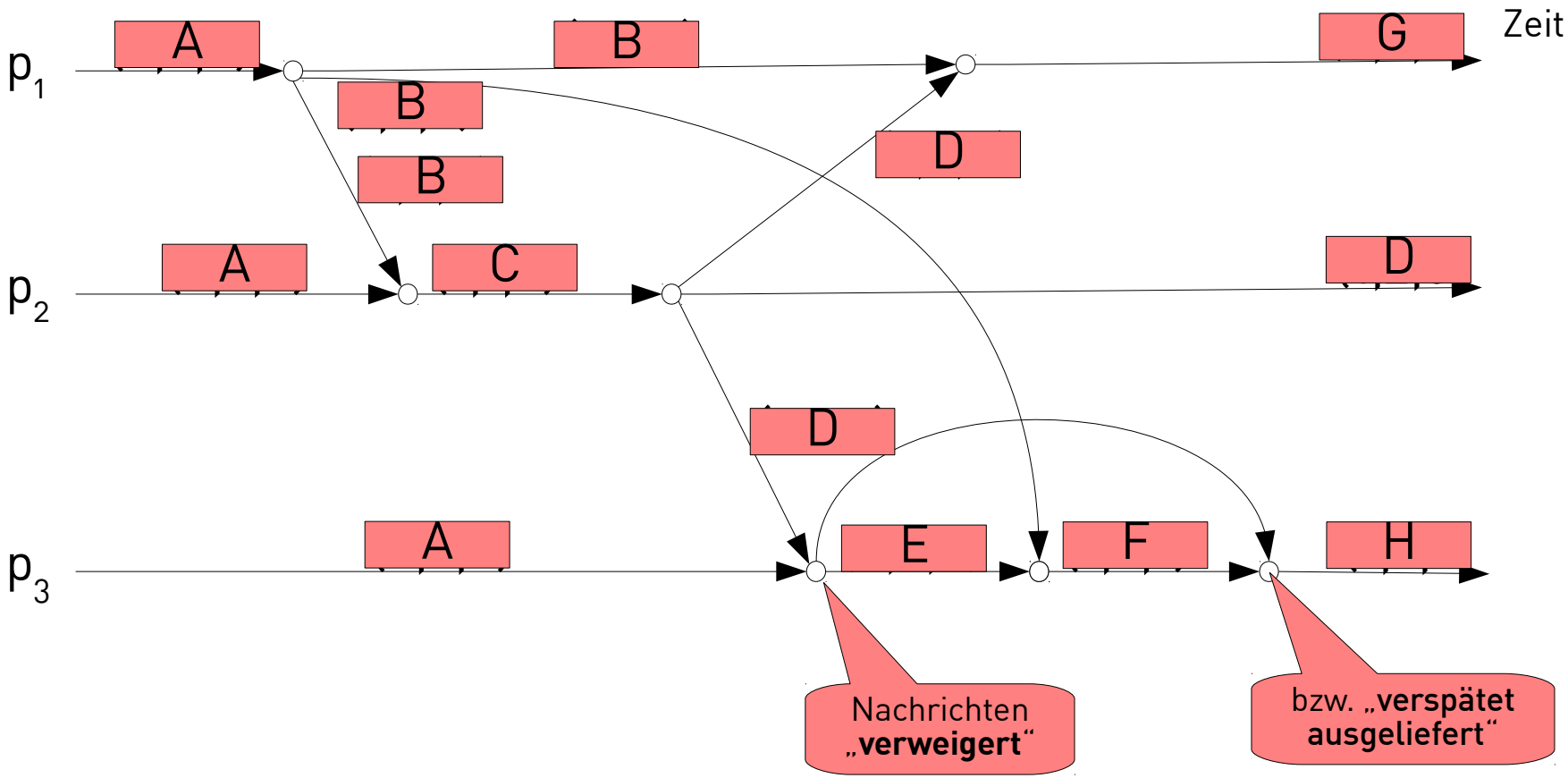
Vektor-Uhren und Kausalität



Bildquelle: https://de.wikipedia.org/wiki/Datei:Vektoruhren.svg

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Vektor-Uhren -Übung



a) Welche Werte nehmen die Vektor-Uhren an? („*Fill in the blanks*“!)

b) Nach welcher Logik kann P2 entscheiden, ob/wann der Nachricht von P0 angenommen werden kann?

Die Logik hat zwei wichtige Voraussetzungen:
1) Es gibt keine lokale Ereignisse
2) Alle Nachrichten „wollen“ Broadcasts sein (müssen allerdings geholfen werden).

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Struktur von Kapitel IV – Architekturale Themen

I Einleitung

II Grundlegende Kommunikationsdienste

III Middleware

IV Architekturen & Algorithmen

- A Synchronisierung
- B Konsistenz und Replikation
- C Fehlertoleranz

V Beispiele bzw. Dienste

- A Verteilte Dateisysteme
- B Namensdienste

VI Sicherheit & Sicherheitsdienste

VII Zusammenfassung

- A Warum Replikation auf Kosten Konsistenz?
- B Architektur & Replikation
- C Konsistenzmodelle
- D Konsistenzprotokolle
- E Verteilte Transaktionen

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Replikation → Konsistenz-Probleme

- **Konsistenz** = (ungefähr)
  Synchronisierung bzw. *Aktualisierung* von **verschiedene Kopien** von Daten.
- Also, gäbe es nur eine Kopie,
  gäbe es keine Problemen mit Konsistenz.
- **Replikation → Konsistenz-Probleme**
- *Also, warum baut man (manchmal) verteilte Systeme mit Replikation?*

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Why replicate? Grunde für Replikation

- **Last-Verteilung:**
  Wenn verschiedene Server Kopien der Daten haben, können Sie den Last verteilen.

- **Latenz-Zeit Optimierung:**
  Lokale Kopien können schneller übermittelt werden als eine weit entfernte Kopie.

  Beide Grunde setzten voraus, dass mehr Leistung dadurch gewonnen werden, als die Aktualisierung kostet.

- **Fehlertoleranz bzw. Verfügbarkeit**
  Fällt eine Kopie aus, haben wir noch *n-1* Kopien.

  Allerdings: bringt mit sich zwei „neue“ *Fehlerquellen*:

  1) Manche Kopien können veraltet sein
  2) Kopien können divergieren.

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Architektur und Replikation

- **Client-Server**
  1 Server, *n* Client – kann es Probleme geben?
  Hinweis: Ja, wenn Clients lesen und dann *schreiben.*

  Beispiel:
  Client *a* ließt Kontozustand *z*, berechnet $z_a = z + \Delta_a$.
  Client *b* ließt Kontozustand *z*, berechnet $z_b = z + \Delta_b$.
  ...Was kann hier schief gehen?

- **Cluster**
  Cluster replizieren Software und Hardware.
  Für den Client tritt ein Cluster als eine Instanz auf.

  Man unterscheidet verschiedene Arten von Clustern:

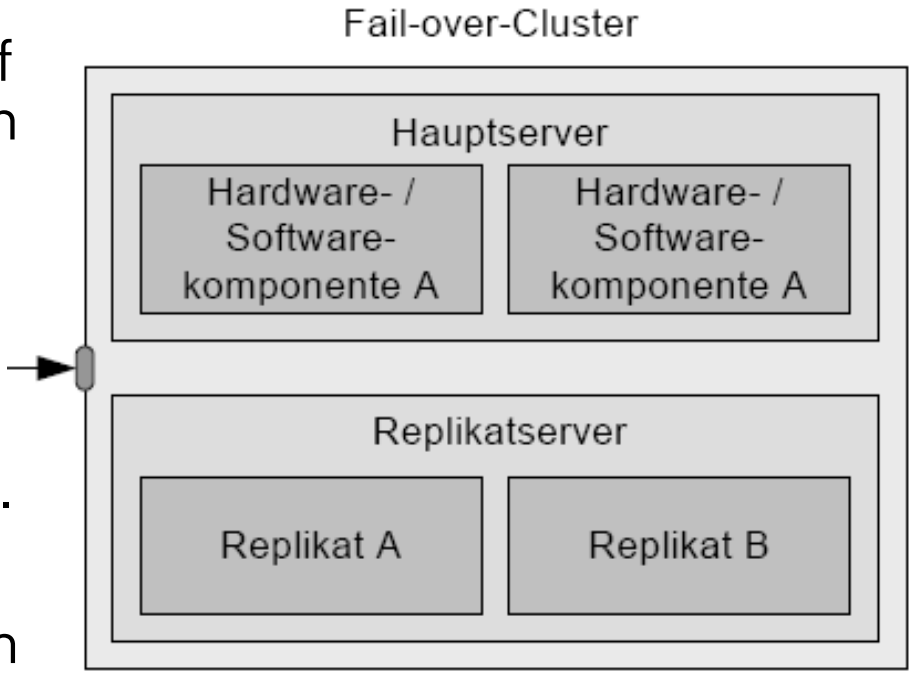
  1) Fail-over Cluster

  2) Load-balancing Cluster

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Fail-over Cluster

- Haupt- und Replikatserver sind vollständig identisch, liegen jedoch auf unterschiedlichen Knoten im verteilten System.
- Im Normalfall arbeitet nur der Hauptserver.
- Tritt ein Fehler auf, werden alle Aufrufe an den Replikatserver geleitet.
- **Hot-Stand-by:** Der Replikatserver steht immer bereit und kann bei einem Wechsel sofort Aufrufe bearbeiten.
- **Cold-Stand-by:** Der Replikatserver muss bei einem Wechsel explizit bereitgestellt (hochgefahren) werden.

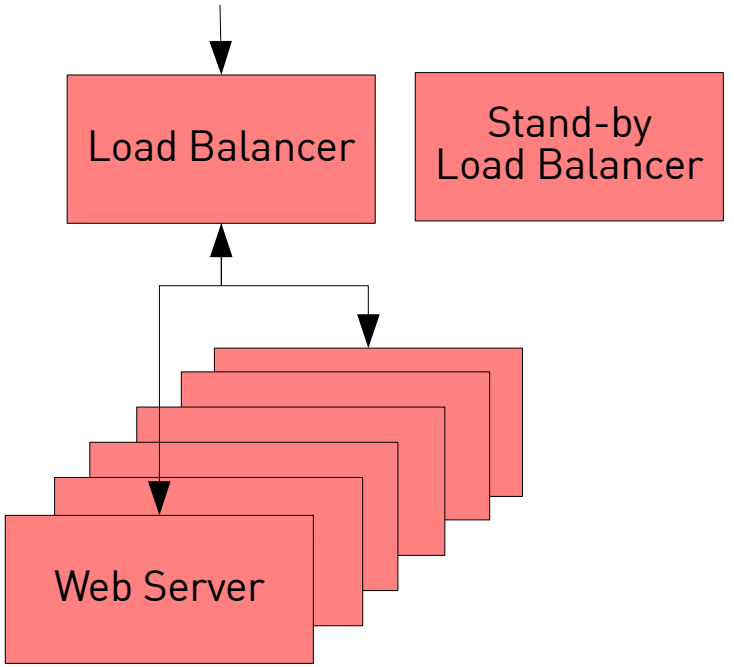


Frage: Hot-Stand-by ist offensichtlich besser. Warum gibt es überhaupt Cold-Stand-by?

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Load-Balancing Cluster

- Bei einem Load-balancing Cluster arbeiten alle Instanzen parallel.
- Client-Aufrufe werden nach einem vorgegebenen Algorithmus (z.B. Round Robin) an alle Instanzen im Cluster verteilt.
- **Beispiel: Webserver-Cluster**
  Fast alle Zugriffe sind Lese-Zugriffe.



Beispiel: Web-Server Cluster (vgl. Kapitel I).
Hinweis: Die Load-Balancer stellen ein Fail-Over Cluster innerhalb der Load-Balancing Cluster dar!

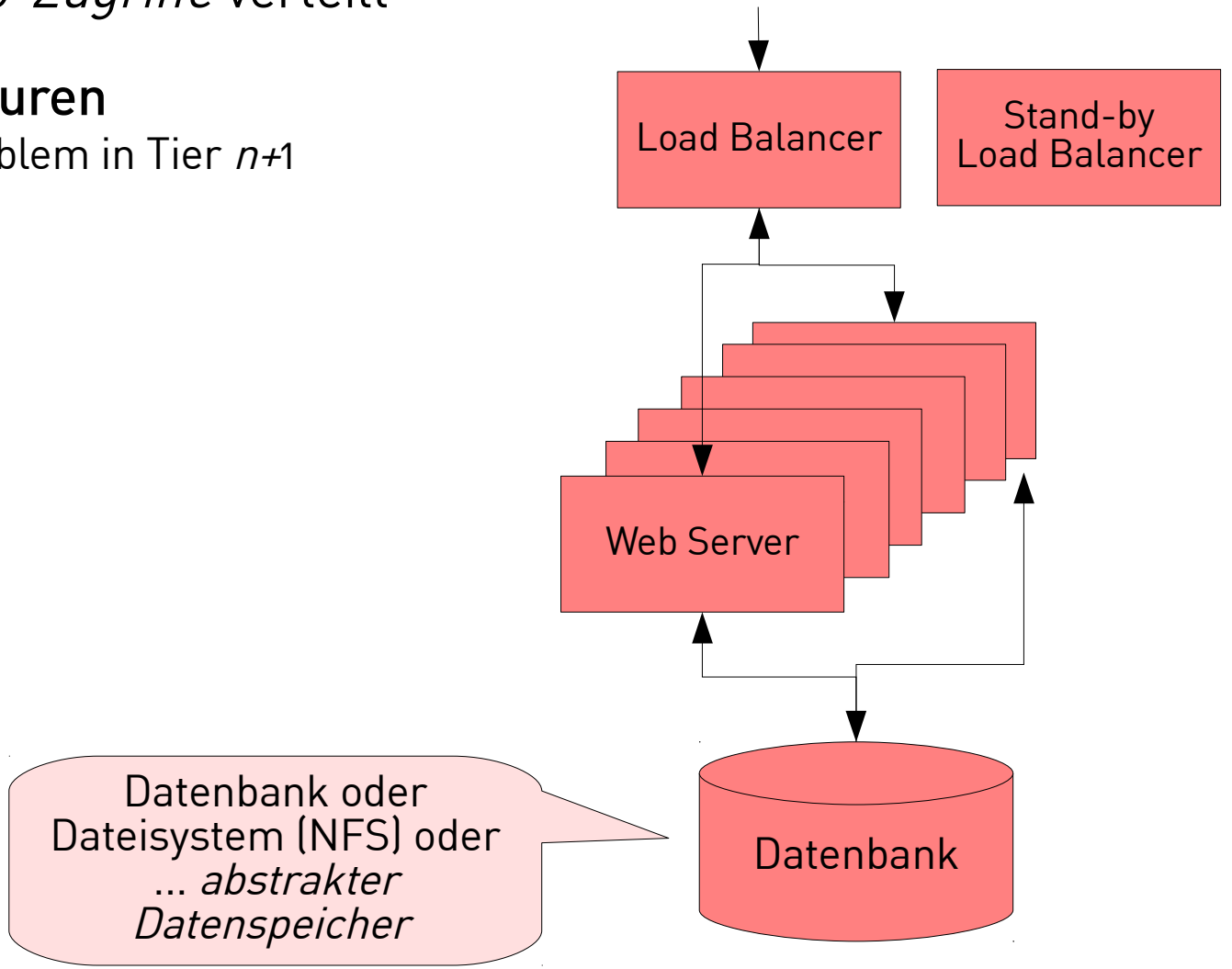
Frage: Kann es Konsistenz-Probleme in Webserver-Cluster geben? - bzw. -
Wie viel Konsistenz braucht ein Web-Surfer (Use-Case)?

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Load-Balancing Cluster Alternativen (1)

Wie sollten *Schreib-Zugriffe* verteilt werden?

1) ***n*-Tier Architekturen**
   Versteckt das Problem in Tier *n+1*

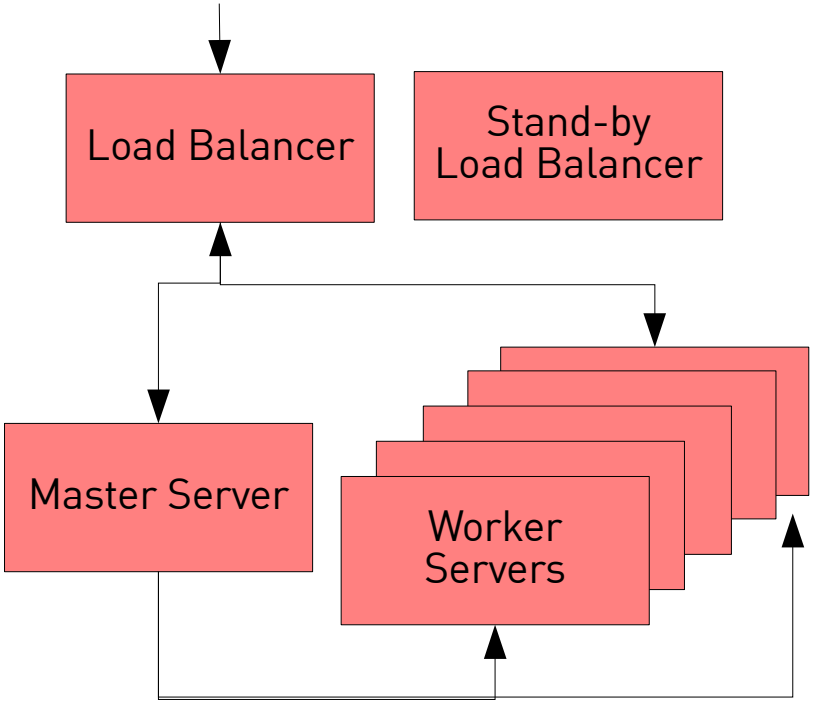


h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Load-Balancing Cluster Alternativen (2)

Wie sollten *Schreib-Zugriffe* verteilt werden?

1) *n*-Tier Architekturen
   Versteckt das Problem in Tier *n+1*
2) **Master / Worker**
   Von jedem kann *gelesen* werden;
   Geschrieben wird nur an den Meister (der danach an den Arbeiter schreibt).
3) **Verteilte Master/Worker**
   Jeder ist Master für einen Teil der Daten.

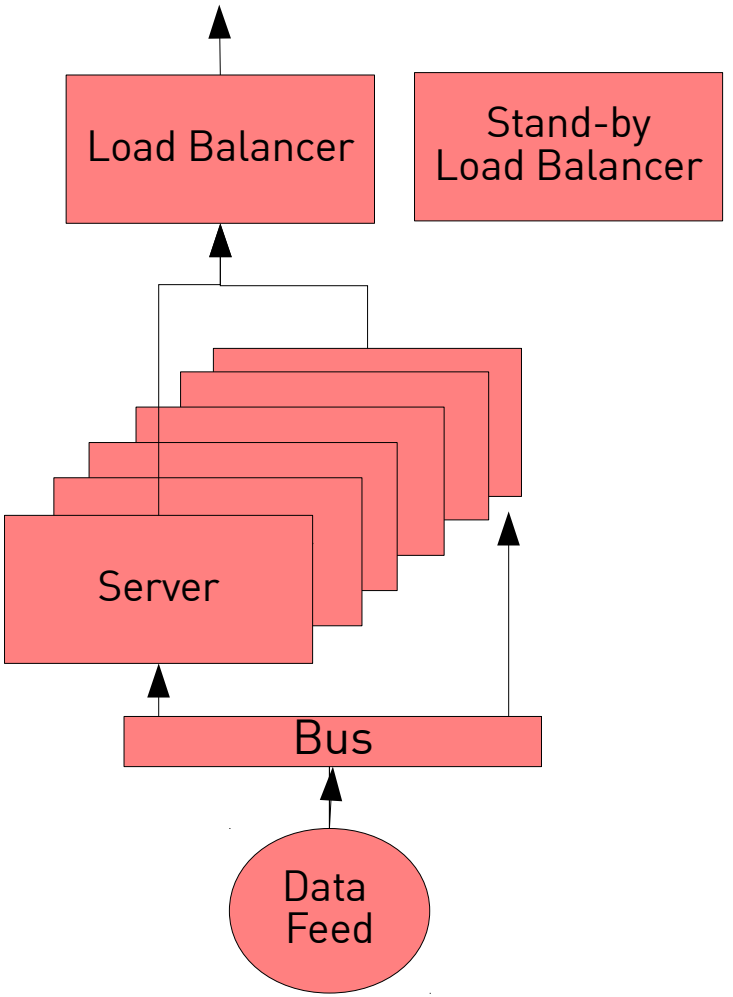


h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Load-Balancing Cluster Alternativen (3)

Wie sollten *Schreib-Zugriffe* verteilt werden?

1) ***n*-Tier Architekturen**
   Versteckt das Problem in Tier *n*+1
2) **Master / Worker**
   Von jedem kann *gelesen* werden;
   Geschrieben wird nur an den Meister (der danach an den Arbeiter schreibt).
3) **Verteilte Master/Worker**
   Jeder ist Master für einen Teil der Daten.
4) **Multicasting**
   Wenn Eingabe-Clients ≠ Ausgabe-Clients, können Schreibe-Zugriffe an alle geschickt werden.
5) **Anwendungs- spezifische Architekturen...**



h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Struktur von Kapitel IV – Architekturale Themen

I Einleitung

II Grundlegende Kommunikationsdienste

III **Middleware**

IV Architekturen & Algorithmen

- A Synchronisierung
- B Konsistenz und Replikation
- C Fehlertoleranz

V Beispiele bzw. Dienste

- A Verteilte Dateisysteme
- B Namensdienste

VI Sicherheit & Sicherheitsdienste

VII Zusammenfassung

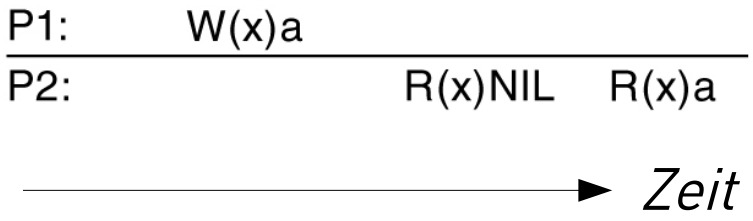
- A Warum Replikation auf Kosten Konsistenz?
- B Architektur & Replikation
- C Konsistenzmodelle
- D Konsistenzprotokolle
- E Verteilte Transaktionen

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Konsistenz-Modelle

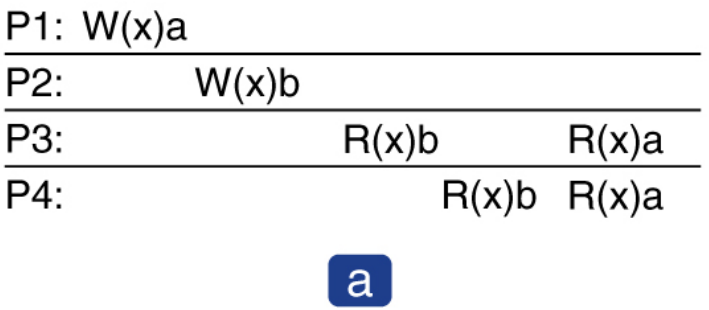
- Können wir Konsistenz ohne spezielle Architektur schaffen?
- Zuerst: Anforderungsanalyse!
- **Konsistenzmodell** ≈ Vertrag (SLA), der sagt, wie viel Konsistenz Clients erwarten dürfen.
- **Komponenten** eines Konsistenzmodells:
  - Abstrakter **Datenspeicher**
  - Schreibe- & Lese-**Operationen**
- **Hintergrund:** Es gibt kein optimales Konsistenzmodell. Verschiedene Anwendungen haben verschiedene Anforderungen (*Requirements*).

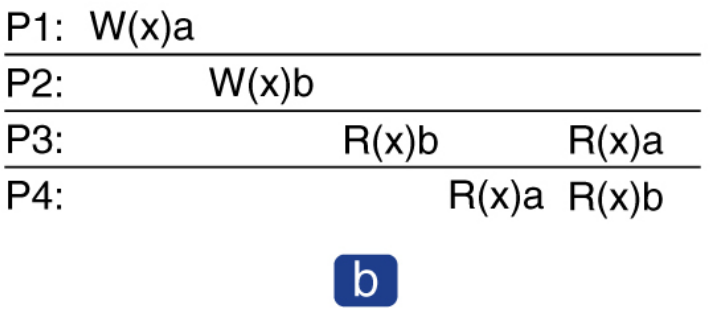
- **Schreibweise:**
  - $R_i(x)a$ heißt: Prozess *i* liest (reads) von Datenelement *x* den Wert *a*.
  - $W_i(x)a$ heißt: Prozess *i* schreibt (writes) den Wert *a* zu Datenelement *x*.
- **Zeichnung:** „Swim-Lanes“



h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Sequnziell konsistenter Datenspeicher





(a) Ein sequenziell konsistenter Datenspeicher; (b) ein Datenspeicher, der nicht sequenziell konsistent ist

Definition

- Es gibt *eine Reihenfolge*, in der die Schreib-Operationen hätten stattfinden können.
- Diese Reihenfolge widerspricht die Resultate der Lese-Operationen keines einzelnen Prozesses.
- Diese Reihenfolge widerspricht die Logik keines einzelnen Programms

D.h. jede Zeile, jedes Swim-Lane, *an sich* ist wie man es erwarten würde.

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Kasual konsistenter Datenspeicher (1)

P1: W(x)a				
P2:	R(x)a	W(x)b		
P3:			R(x)b	R(x)a
P4:			R(x)a	R(x)b

a

(a) Ein Regelverstoß in einem kausal Speicher;

P1: W(x)a				
P2:		W(x)b		
P3:			R(x)b	R(x)a
P4:			R(x)a	R(x)b

b

(b) eine korrekte Ereignissequenz in konsistenten einem kausal konsistenten Speicher

Definition

- Schreibzugänge, die möglicherweise in kausaler Beziehung zueinander stehen, müssen von allen Prozessen in derselben Reihenfolge wahrgenommen werden.
- Parallel Schreibvorgänge können auf unterschiedlichen Rechnern in unterschiedlicher Reihenfolge gesehen werden.

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Kasual konsistenter Datenspeicher (2)

P1:	W(x)a			W(x)c	
P2:		R(x)a	W(x)b		
P3:		R(x)a		R(x)c	R(x)b
P4:		R(x)a		R(x)b	R(x)c

Diese Sequenz ist bei einem kausal konsistenten Speicher zulässig, nicht aber bei einem sequenziell konsistenten.

Definition

- Schreibzugänge, die möglicherweise in kausaler Beziehung zueinander stehen, müssen von allen Prozessen in derselben Reihenfolge wahrgenommen werden.
- Parallel Schreibvorgänge können auf unterschiedlichen Rechnern in unterschiedlicher Reihenfolge gesehen werden.

Kasuale Konsistenz ist schwacher als sequentielle Konsistenz.

(Es gibt auch andere Modelle).

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Hörsaal Übung (alte Klausurfrage!)

Ist folgende Sequenz *sequenziell konsistent*? Ist sie *kausal konsistent*? Begründen Sie Ihre Antwort (5 Punkte).

P1:			R(x) d	W(x) c
P2:	W(x) a		R(x) b	
P3:		R(x) a	W(x) d	
P4:			R(x) c	W(x) b

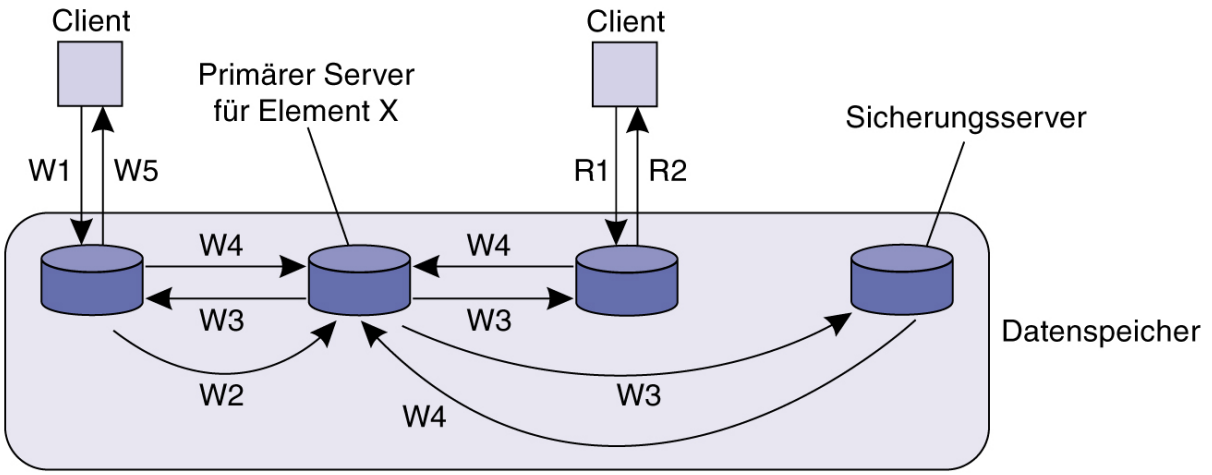
h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Konsistenzprotokolle

- Jetzt wissen wir, *ob* die Daten konsistent sind; wir wissen immer noch nicht, *wie* sie konsistent zu halten sind.
- **Konsistenzprotokoll = Implementierung eines Konsistenzmodells.**
- **Urbildbasierte Protokolle (Primary-Based Protocols):**
  - implementieren sequentielle Konsistenz.
  - setzen voraus, dass *eine Kopie* jedes Datenelements als *die Urkopie* gekennzeichnet ist.
  - Vgl. Verteilter Master/Slave Architektur (s.o.).

Urbildbasierte Protokolle (1)

- **Urbildbasierte Protokolle für entfernte (remote) Schreibvorgänge**
  - Urkopien bleiben, wo sie sind.
  - Der Besitzer der Urkopie bekommt alle Schreib-Zugriffe, gibt sie danach weiter
  - Besitzer der Urkopie blockiert schreibende Klienten, bis die Schreibvorgänge „fertig“ sind.
  - Es gibt unterschiedliche Definitionen von „fertig“... z.B. wie gezeigt oder mit W5 vor W4.



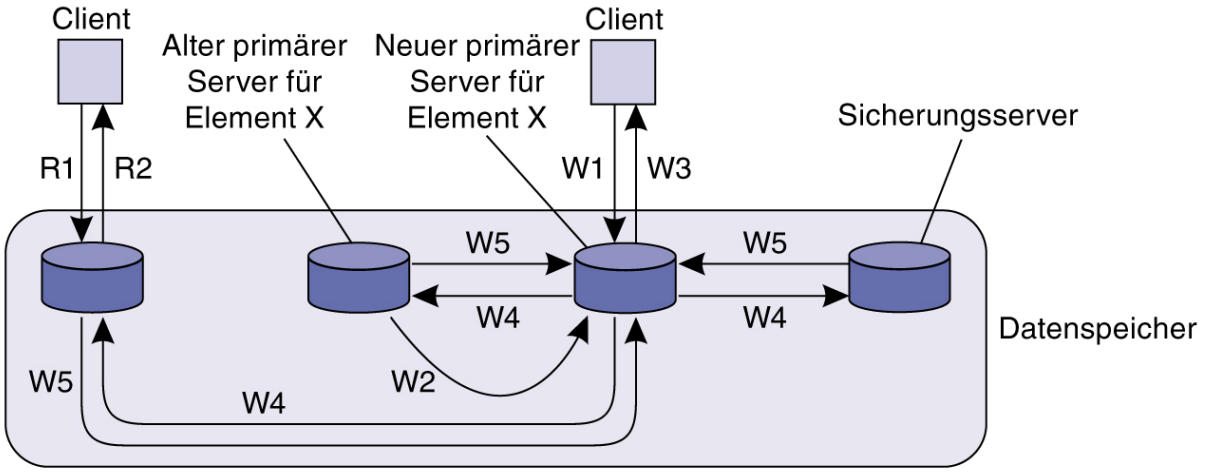
W1 – Schreibanforderung
W2 – Weiterleitung der Anforderung an den primären Server
W3 – Anweisung an die Sicherungen zur Aktualisierung
W4 – Aktualisierungsbestätigung
W5 – Bestätigung des abgeschlossenen Schreibvorgangs

R1 – Leseanforderung
R2 – Leserückmeldung

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Urbildbasierte Protokolle (2)

- **Urbildbasierte Protokolle für lokale Schreibvorgänge**
  - Urkopien migrieren dorthin, wo geschrien wird.
  - Optimiert Reihen von Schreiboperationen.
  - Reduziert Netzwerk.Traffik.
  - Ähnlich wie Cache-Kohärenz Protokolle.
  - Hat Probleme beim Absturtz (wenn Migration nicht fertig sind).
  - Hat Probleme *während* der Migration (Schreiboperationen stauen u.U.).



W1 – Schreibanforderung
W2 – Verschieben von Element X zum neuen primären Server
W3 – Bestätigung des abgeschlossenen Schreibvorgangs
W4 – Anweisung an die Sicherungen zur Aktualisierung
W5 – Aktualisierungsbestätigung

R1 – Leseanforderung
R2 – Leserückmeldung

Rechnerarchitekten nennen das ein *COMA – Cache Only Memory Architecture*.
Vgl. R. Moore, SDAARC: A Self Distributing Associative Architecture, Shaker Verlag, 2001.
;-)

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Struktur von Kapitel IV – Architekturale Themen

I Einleitung

II Grundlegende Kommunikationsdienste

III **Middleware**

IV Architekturen & Algorithmen

- A Synchronisierung
- B Konsistenz und Replication
- C Fehlertoleranz

V Beispiele bzw. Dienste

- A Verteilte Dateisysteme
- B Namensdienste

VI Sicherheits & Sicherheitsdienste

VII Zusammenfassung

- A Warum Replikation auf Kosten Konsistenz?
- B Architektur & Replikation
- C Konsistenzmodelle
- D Konsistenzprotokolle
- E Verteilte Transaktionen

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

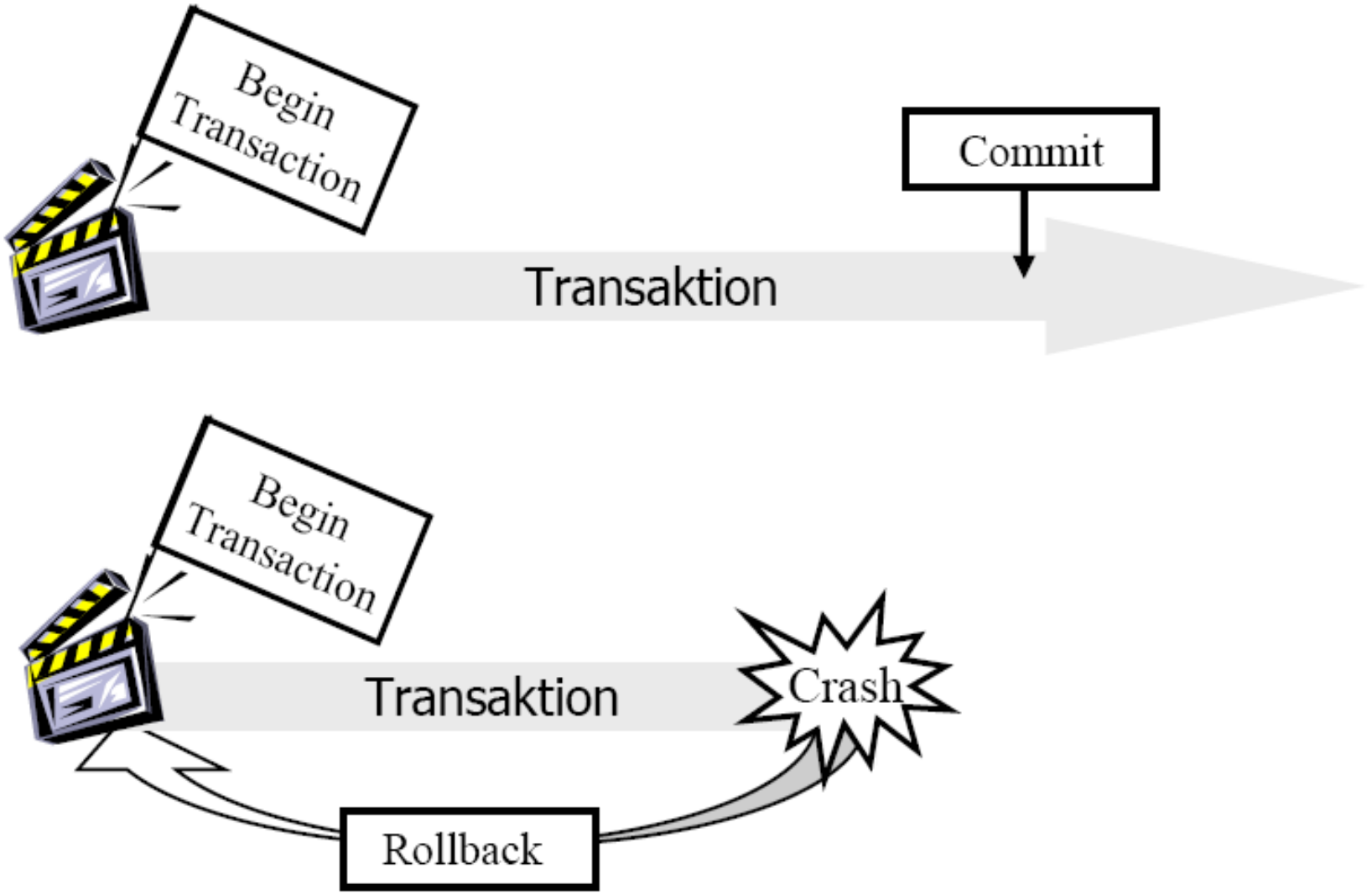
Alternativ-Konsistenz: Transaktionen

- **Es muss nicht immer nur Lese- und Schreiboperationen geben.**
- Manchmal will die Anwendungen Operationen (evtl. auf verschiedene Ressourcen) zusammenfassen...
- ...und unter Unstände abbrechen!
- 3 neue Operationen:
  - Begin
  - Commit
  - Abort

- Die Transaktionsverwaltung erfolgt
  - durch **Ressourcenmanager**: können nur *einfache* Transaktionen verwalten.
  - durch einen **Transaktionsdienst**: verwalten *komplexe* (verteilte, verschachtelte) Transaktionen auf Anwendungsebene.

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Transaktionen – Alles oder Nichts



h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Transaktionen – ACID

- **Atomarität:** Prinzip Alles oder Nichts
  - *Beispiel Karte kaufen: Entweder werden die Karte gedruckt und der Betrag abgebucht oder keines von beidem.*
- **Consistency (Konsistenz):** Ein konsistenter Zustand wird in einen neuen konsistenten Zustand gebracht.
  - *Der Datenbestand ist vor und nach dem Verkauf konsistent (z.B. Sitzplatz belegt, Geld abgebucht).*
- **Isolation:** Transaktionen dürfen sich nicht gegenseitig beeinflussen.
  - *Zwei parallel arbeitende Ticketverkäufer beeinflussen sich nicht während des Verkaufs.*
- **Dauerhaftigkeit:** Am Ende der Transaktion sind die vorgenommenen Änderungen gespeichert, beispielsweise in der Datenbank.
  - *Die Karte ist ausgedruckt, der Platz reserviert und das Geld abgebucht.*

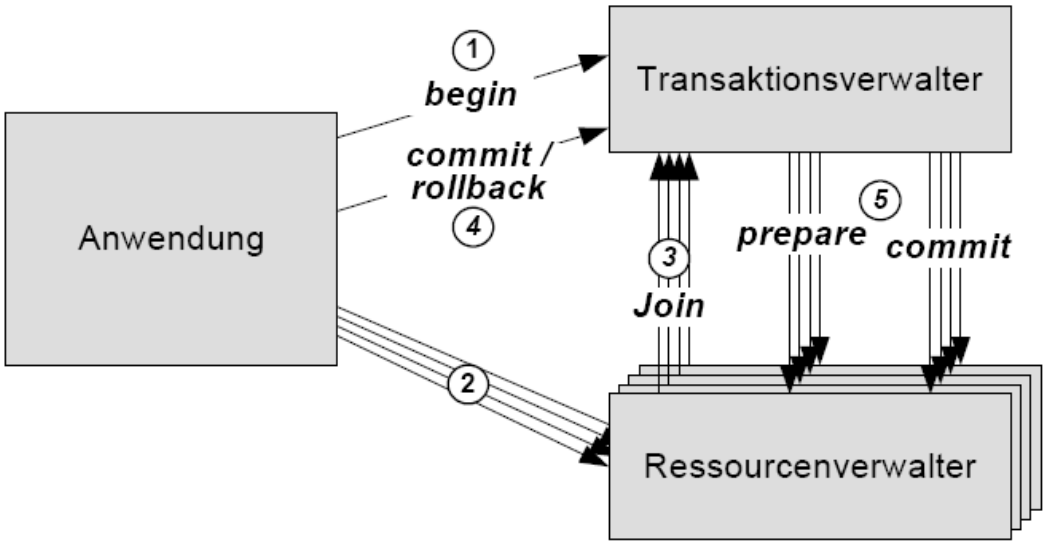
h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Transaktionen – 2 Phasen Commit

Strategie für verteilte Transaktionen

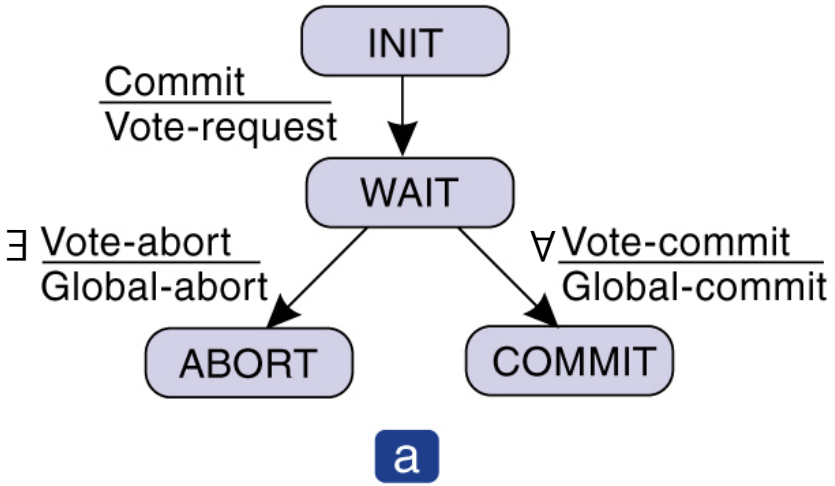
1. Phase: alle beteiligten Ressourcen-manager prüfen, ob die Transaktion bei ihnen erfolgreich ablaufen würde.

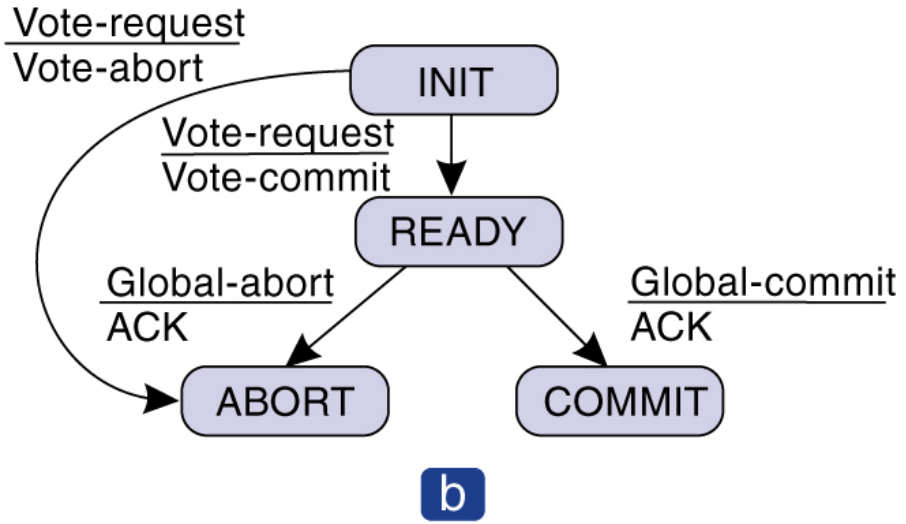
2. Phase: falls alle Ressourcenmanager die lokale Transaktion durchführen können, gibt der Transaktionsverwalter den Befehl zum endgültigen Festschreiben.
   – Tritt bei einem Ressourcenmanager ein Konflikt auf, wird die gesamte Transaktion abgebrochen.



h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

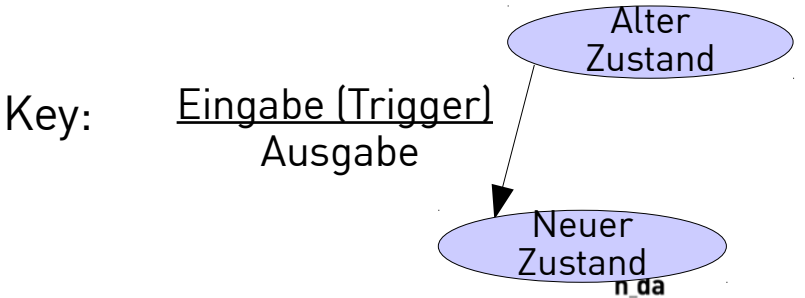
2 Phasen Commit = 2 Zustandsautomen





(a) Der finite Zustandsautomat für den Transaktionsverwalter im 2PC

(b) der finite Zustandsautomat für einen Ressourcenmanager



n_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Struktur von Kapitel IV – Architekturale Themen

I Einleitung

II Grundlegende Kommunikationsdienste

III Middleware

IV Architekturale Themen

- A Synchronisierung
- B Konsistenz und Replication
- C Fehlertoleranz

V Beispiele bzw. Dienste

- A Verteilte Dateisysteme
- B Namensdienste

VI Sicherheit & Sicherheitsdienste

VII Zusammenfassung

- A Verlässliche Systeme
- B Fehlersemantik bei RPC & RMI
- C Fehlerbehandlung bei RPC

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Fehler-Toleranz - Definition

- Der Ziel: Verlässliche Systeme:
  - **Verfügbarkeit (Availability)**
    = Das System kann *jetzt* verwendet werden.
  - **Zuverlässigkeit (Reliability)**
    = Das System ist *oft* Verfügbar (z.B. 99,9% der letzten 6 Monaten).
  - **Funktionssicherhiet (Safety)**
    = Keine katastrophe eintritt, wenn ein system vorübergehend nicht korrekt arbeitet (z.B. im Krankenhaus).
  - **Wartbarkeit (Maintainability)**
    = Ein ausgefallenes System kann leicht repariert werden.

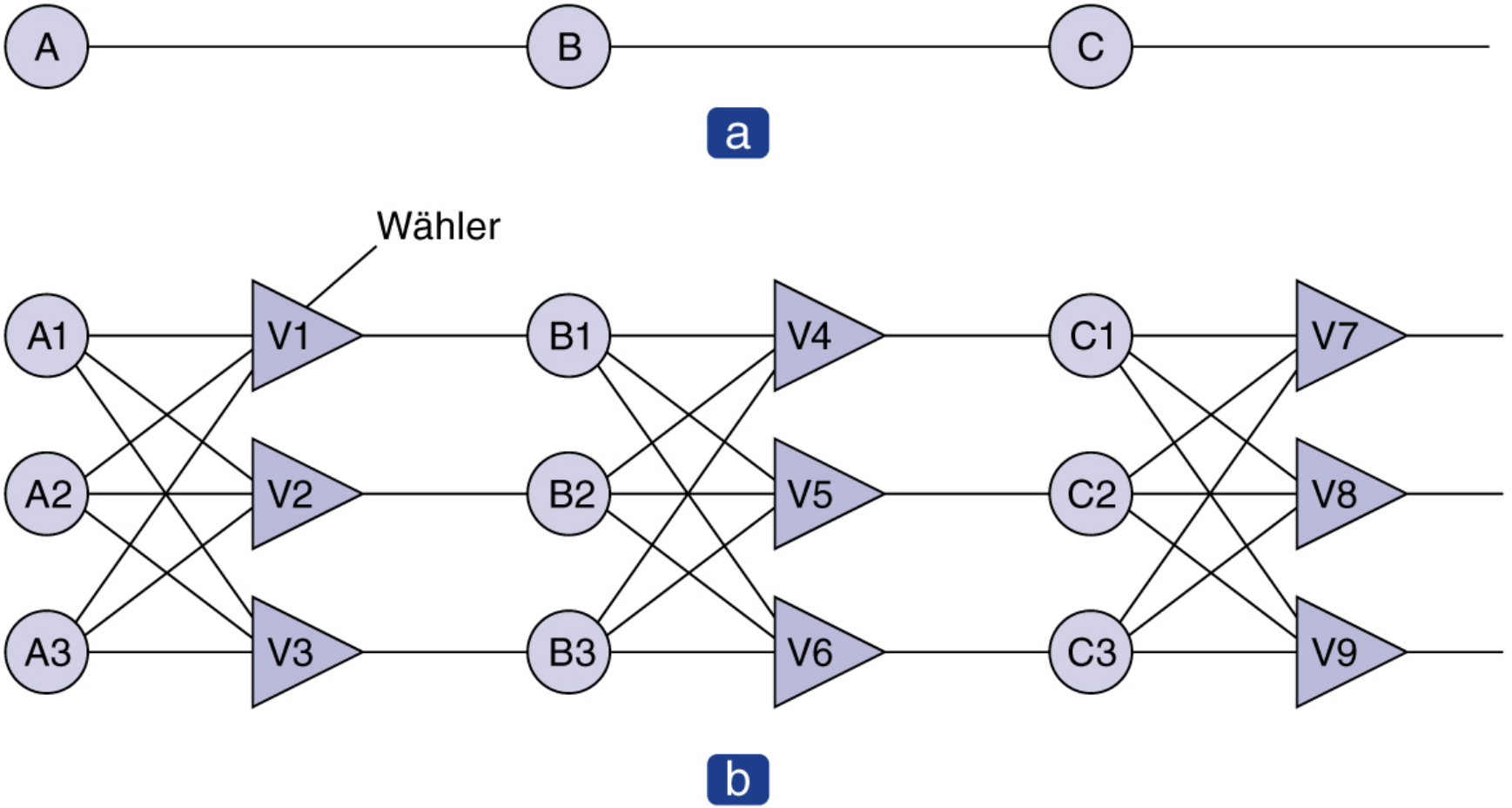
Fehler-Modelle

Ausfallart	Beschreibung
Absturzausfall (Crash Failure)	Ein Server steht, hat aber bis dahin richtig gearbeitet. Der angebotene Dienst bleibt beständig aus (ständiger Dienstausfall).
Dienstausfall (Omission Failure) *Empfangsauslassung* *Sendeauslassung*	Ein Server antwortet nicht auf eingehende Anforderungen. Ein Server erhält keine eingehenden Anforderungen. Ein Server sendet keine Nachrichten.
Zeitbedingter Ausfall (Timing Failure)	Die Antwortzeit eines Servers liegt außerhalb des festgelegten Zeitintervalls.
Ausfall korrekter Antwort (Response Failure) Ausfall durch *Wertfehler (Value Failure)* Ausfall durch *Zustandsübergangsfehler (State Transition Failure)*	Die Antwort eines Servers ist falsch. Dieser Ausfall wird oft auch kurz Antwortfehler genannt. Der Wert der Antwort ist falsch. Der Server weicht vom richtigen Programmablauf ab.
Byzantinischer oder zufälliger Ausfall (Arbitrary oder Byzantine Failure)	Ein Server erstellt zufällige Antworten zu zufälligen Zeiten.

ADT UNIVERSITY OF APPLIED SCIENCES
fbi FACHBEREICH INFORMATIK

Fehler-Toleranz & Redundanz

- TMR = Triple Modular Redundancy



(a) Ohne und (b) Mit dreifache modulare Redundanz

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Struktur von Kapitel IV – Architekturale Themen

I Einleitung

II Grundlegende Kommunikationsdienste

III Middleware

IV Architekturen & Algorithmen

- A Synchronisierung
- B Konsistenz und Replikation
- C Fehlertoleranz

V Beispiele bzw. Dienste

- A Verteilte Dateisysteme
- B Namensdienste

VI Sicherheit & Sicherheitsdienste

VII Zusammenfassung

- A Verlässliche Systeme
- B Fehlersemantik bei RPC & RMI
- C Fehlerbehandlung bei RPC

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Request/Reply Fehler Semantik (1)

Was soll passieren, wenn RPC bzw. RMI nicht klappt?

Man unterscheidet folgende Fehlersemantik:

(1) Maybe
- es erfolgen keine Fehlerbehandlungsmaßnahmen
- der Dienst wird gar nicht oder höchstens einmal durchgeführt
- im Fehlerfall hat man keine Hinweise, ob der Dienst tatsächlich ausgeführt wurde oder nicht

Die Maybe-Semantik ist u.U. ausreichend für Auskunftsdienste.

Wenn nach einer gewissen Zeit keine Auskunft erteilt wurde, so probiert man es eben noch einmal.

Request/Reply Fehler Semantik (2)

Was soll passieren, wenn RPC bzw. RMI nicht klappt?

Man unterscheidet folgende Fehlersemantik:

(1) Maybe
(2) At-Least-Once
- der Auftrag wird mindestens einmal ausgeführt
- bei Nachrichtenverlusten wird der Auftrag bis zum Erfolg wiederholt
- da keine Filterung von Duplikaten stattfindet, kann es vorkommen, dass ein Auftrag mehrfach ausgeführt wird

Bei idempotenten Operationen (d.h. mehrfache Ausführung verändert das Ergebnis nicht) ist diese Semantik ausreichend.

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Request/Reply Fehler Semantik (3)

Was soll passieren, wenn RPC bzw. RMI nicht klappt?

Man unterscheidet folgende Fehlersemantik:

(1) Maybe
(2) At-Least-Once
(3) At-Most-Once
- die Fehlerbehandlung erfolgt durch Wiederholen des Auftrags
- zusätzlich werden alle Duplikate ausgefiltert
- auch bei vermeintlichen Nachrichtenverlusten wird somit der Auftrag höchstens einmal ausgeführt
- sollte der Server jedoch abgestürzt sein, so wird kein Ergebnis mehr erwartet

Ein Auftrag wird also komplett durchgeführt, oder ein Fehler an den Klienten gemeldet.

In RPC-Implementierungen, wie Corba, DCOM, Java RMI wird diese Fehlersemantik verwendet.

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Request/Reply Fehler Semantik (4)

Was soll passieren, wenn RPC bzw. RMI nicht klappt?

Man unterscheidet folgende Fehlersemantik:

(1) Maybe
(2) At-Least-Once
(3) At-Most-Once
(4) Exactly Once
- schließt den Absturz und den Wideranlauf von Komponenten ein
- Konsistente Rücksetzungsmaßnahmen garantieren, dass die Operation genau einmal durchgeführt wird
- persistente Datenhaltung und verteilte Transaktionsmechanismen sind notwendig

Die Exactly-Once-Semantik ist durch den großen Aufwand nur bei Anwendungen mit hohen Sicherheitsanforderungen sinnvoll.

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Struktur von Kapitel IV – Architekturale Themen

I Einleitung

II Grundlegende Kommunikationsdienste

III Middleware

IV Architekturen & Algorithmen

- A Synchronisierung
- B Konsistenz und Replikation
- C Fehlertoleranz

V Beispiele bzw. Dienste

- A Verteilte Dateisysteme
- B Namensdienste

VI Sicherheit & Sicherheitsdienste

VII Zusammenfassung

A Verlässliche Systeme
B Fehlersemantik bei RPC & RMI
C Fehlerbehandlung bei RPC

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Request/Reply Fehlerbehandlung (0)

Annahme:
alle zu einem Auftrag gehörenden Protokollnachrichten sind identifizierbar, z.B. durch eine fortlaufende Nummer.

Ausgangslage:
Der Client erhält nach dem Abschicken des Auftrags längere Zeit keine Antwort vom Server und reagiert mit einem Timeout.

Hörsaal Übung:

Was kann alles schief gehen bzw. welche **Ursachen** kann ein Timeout haben?

Auflisten!

Auf Vollständigkeit aber auch auf Äquivalenz achten: Sie wollen eine kurze jedoch vollständige Liste.

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Request/Reply Fehlerbehandlung (1)

Mögliche Ursachen:
(1) Die Auftragsnachricht ging verloren.

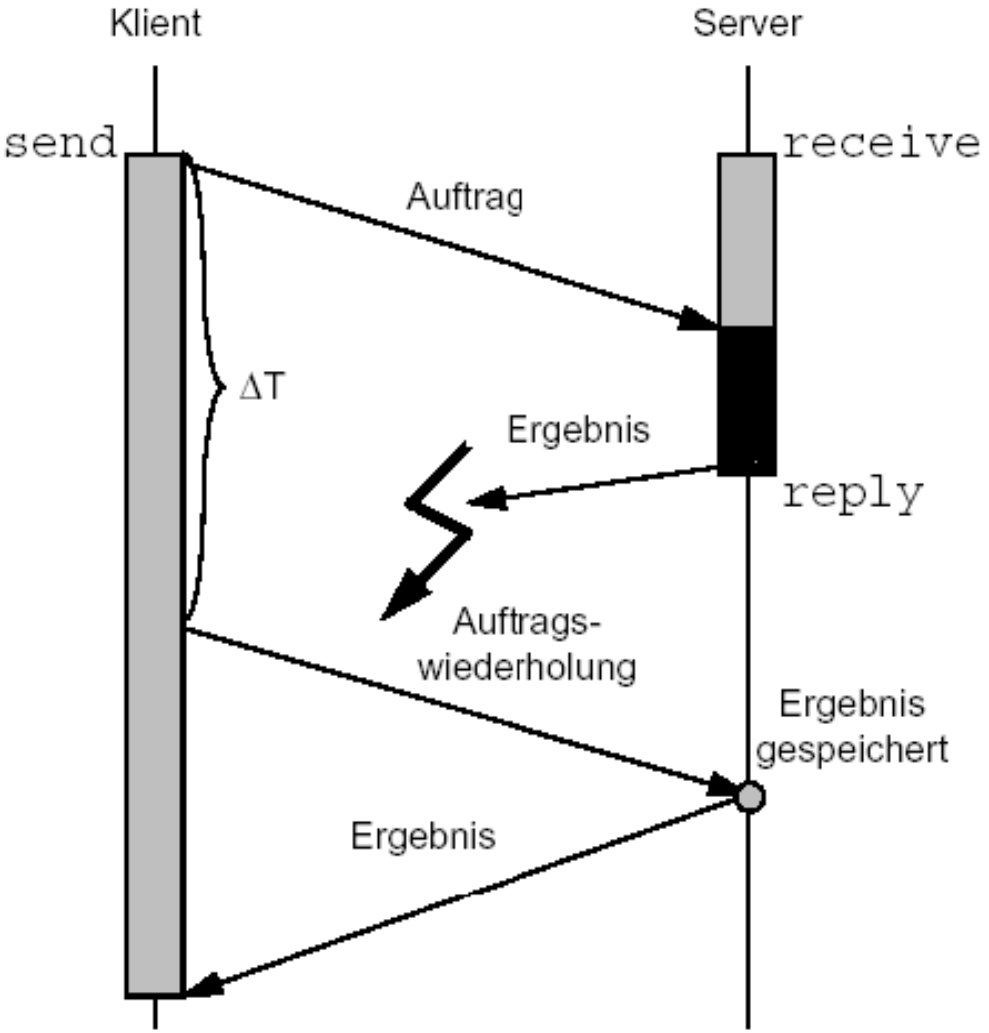
Reaktion:
Den Auftrag wiederholen. Diese Maßnahme ist in allen anderen noch folgenden Fällen ebenfalls sinnvoll.

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Request/Reply Fehlerbehandlung (2)

Mögliche Ursachen:
(1) Die Auftragsnachricht ging verloren.
(2) Die Antwort des Servers ging verloren oder ist noch nicht angekommen.

Reaktion:
Neben der Auftrags-wiederholung kann man auf Serverseite die Antwort-Nachrichten zwischen-speichern und bei erneuter Anforderung nochmals **ohne wiederholte Berechnung** schicken.



h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Request/Reply Fehlerbehandlung (2)

Mögliche Ursachen:
(1) Die Auftragsnachricht ging verloren.
(2) Die Antwort des Servers ging verloren oder ist noch nicht angekommen.
(3) Der Server ist abgestürzt.

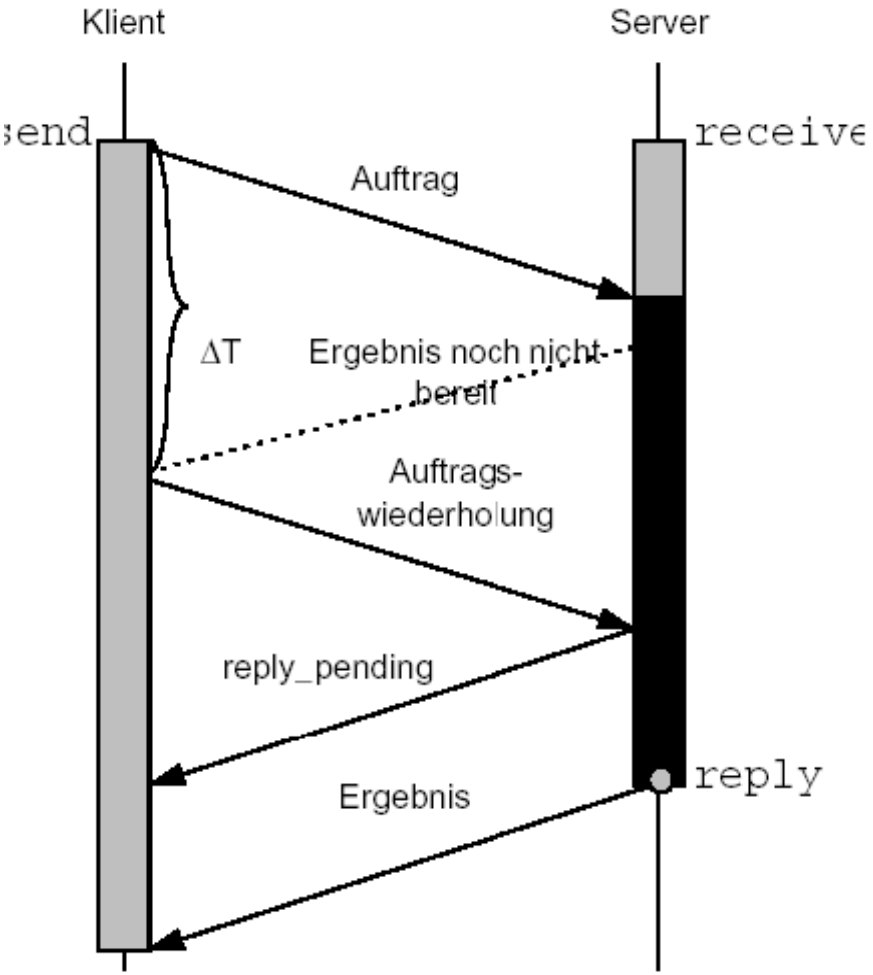
Reaktion:
Server wird neu gestartet,
Clients geben Anfrage nach *n* Time-Outs auf (Fehler wird der Anwendung gemeldet).

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Request/Reply Fehlerbehandlung (3)

Mögliche Ursachen:
(1) Die Auftragsnachricht ging verloren.
(2) Die Antwort des Servers ging verloren oder ist noch nicht angekommen.
(3) Der Server ist abgestürzt.
(4) Der Server braucht lange Zeit für die Auftragsausführung.

Reaktion:
Der Server kann ein reply_pending schicken - Bittet den Client um etwas Geduld.



h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Zusammenfassung Kapitel IV

- **Synchronisierung:**
  Zeit ist überall anders, Komponenten können vor oder nach laufen und *Race-Conditions* entstehen
  - Physische Uhren – vg. NTP
  - Lamport's logische Uhren – können Skalare oder Vektoren sein
- **Fehlertoleranz:**
  - Verlässliche Systeme
    - Was bedeutet das?
    - Architekturelle Lösung
  - 4 verschiedene Request/ Reply Fehlersemantik
  - Fehlerbehandlung bei RPC:
    - Schritt für Schritt, bis alle Ursachen bedacht sind.
- **Replikation und Konsistenz**
  - Wo es Replikation gibt, wird es Konsistenz-Probleme geben (und es muss Replikation geben).
  - Konsistenz kann man manchmal mittels Architektur kriegen.
  - Ein Konsistenzmodell definiert, wie viel Konsistenz wir anbieten; Konsistenz Protokolle definieren, wie.
  - Beispiel-Modelle: Sequentielle Konsistenz vs. Kausale Konsistenz.
  - Beispiel-Protokolle: Urbildbasierte Protokolle für entfernte bzw. lokale Schreibvorgänge
  - Auch wichtig: 2-Phase-Commits für verteilte Transaktionen

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK