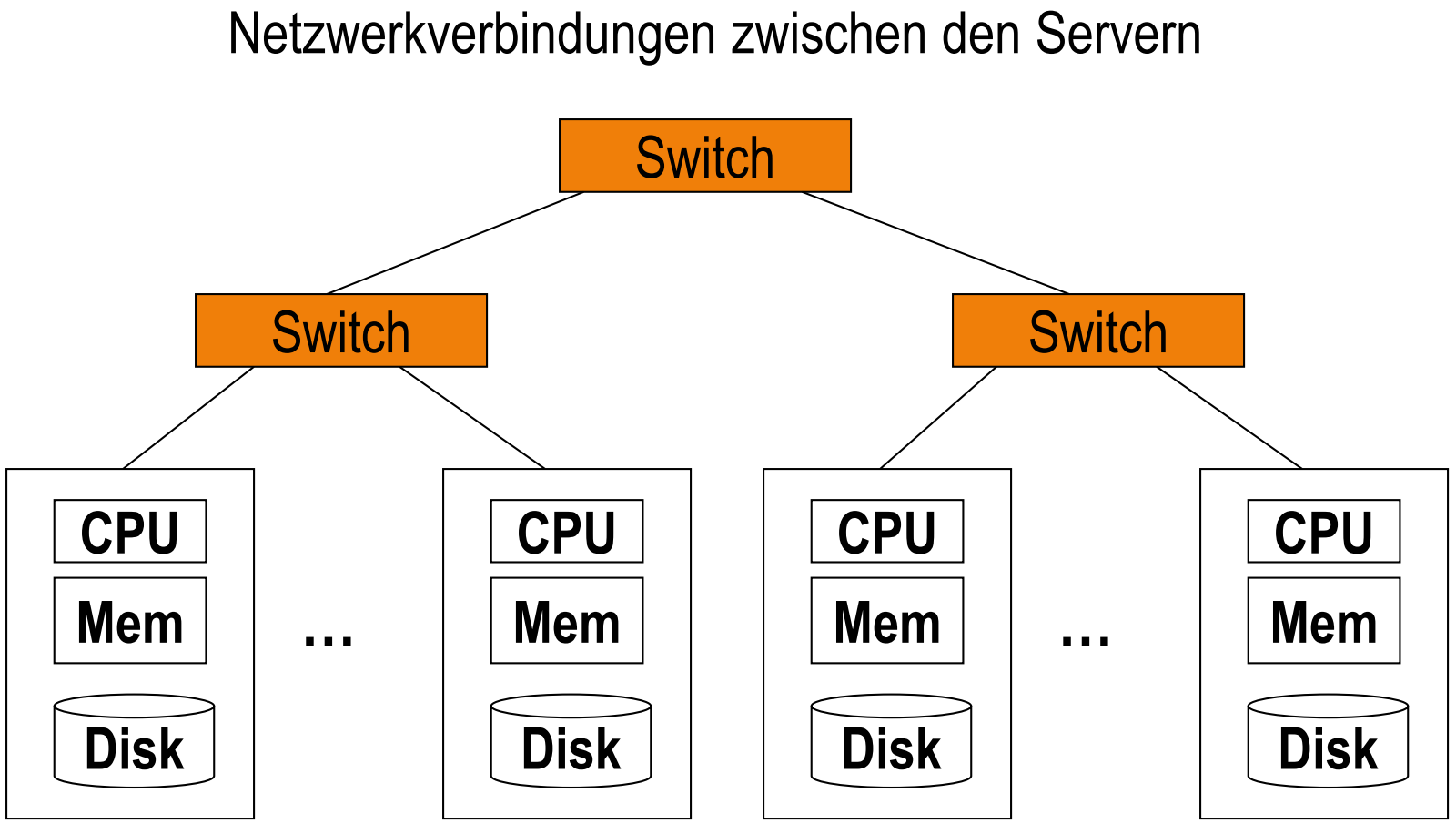
Inhaltsverzeichnis

Verteilte Systeme

- Single-Server-Architektur ist begrenzt bzgl. Speicher und Rechenzeit
- Verteilte Architektur/Horizontale Skalierung (Beispiel):



Partitionierung und Replikation

Verteilte Systeme ermöglichen **Skalierbarkeit, Fehlertoleranz** und **geringe Latenzzeiten** über **Partitionierung** und **Replikation**

Skalierbarkeit: Erweiterbarkeit auf wachsende/schrumpfende Datenmengen oder Anforderungen bzgl. Rechenzeit

Fehlertoleranz: Toleranz gegenüber Server-/Netzwerkfehler

Geringe **Latenzzeiten**: Kurze Antwortzeiten des Servers

Partitionierung: Daten werden über mehrere Rechner verteilt

Skalierbarkeit: Daten A-D auf Server 1, E-H auf Server 2, …
Latenzzeiten: Gleichzeitige Bearbeitung unterschiedlicher Anfragen

Replikation: Datenkopien werden auf mehreren Rechnern gespeichert

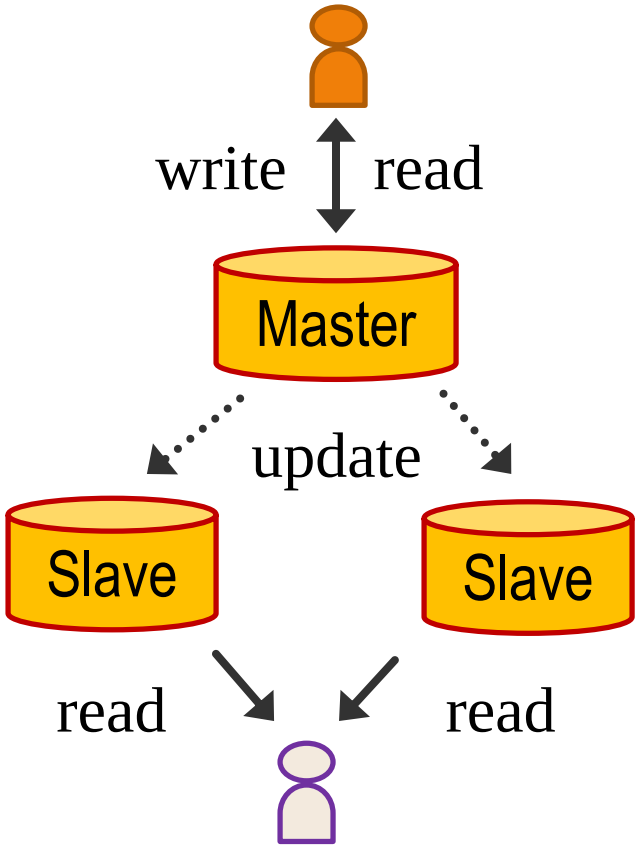
Fehlertoleranz: Kein „Single-Point-of-Failure“ (Übernahme der Aufgaben)
Latenzzeiten: Gleichzeitige Bearbeitung mehrerer Anfragen

Partitionierung

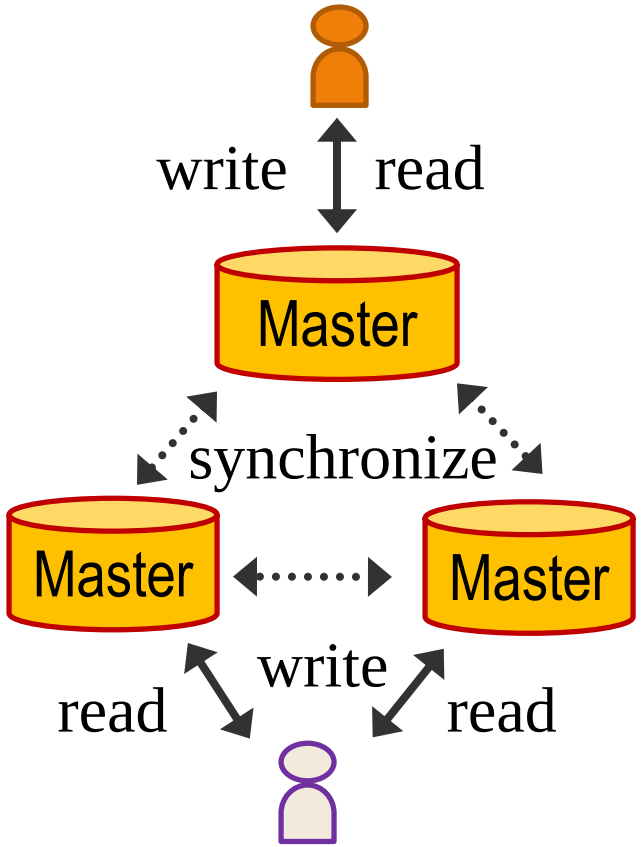
- Horizontale Aufteilung der Daten (*Sharding*)
- Kriterien
  - Art des Zugriffs (überwiegend lesend oder schreibend)
  - Muster des Zugriffs (welche Daten besonders häufig)
  - Häufigkeit des Zugriffs
  - Dauer des Zugriffs
- Eine gute Partitionierung impliziert …
  - Nutzung von Lokalität
  - Minimierung der Kommunikation
  - Lastenausgleich
- Vorgehen: handmade, random, structure-based, value-based, hash-based, affinity-based, cost-based, workload-aware, …

Replikation

- Master-Slave



- Multi-Master

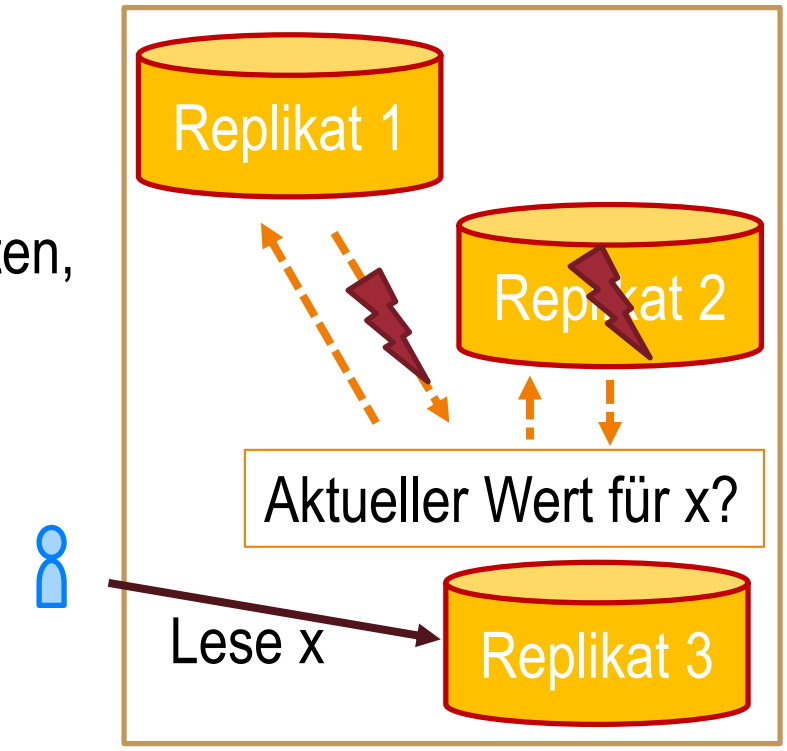


Inhaltsverzeichnis

- **Organisation**
- **Motivation**
  - Relationale Datenbanken
  - NoSQL-Datenbanken
- **Übersicht zur Vorlesung**
- **Verteilte Datenbanksysteme**
  - Verfügbarkeit und Serialisierbarkeit
  - Eventual Consistency, Sitzungsgarantien und kausale Konsistenz
  - ACID-Garantien
  - CAP-Theorem

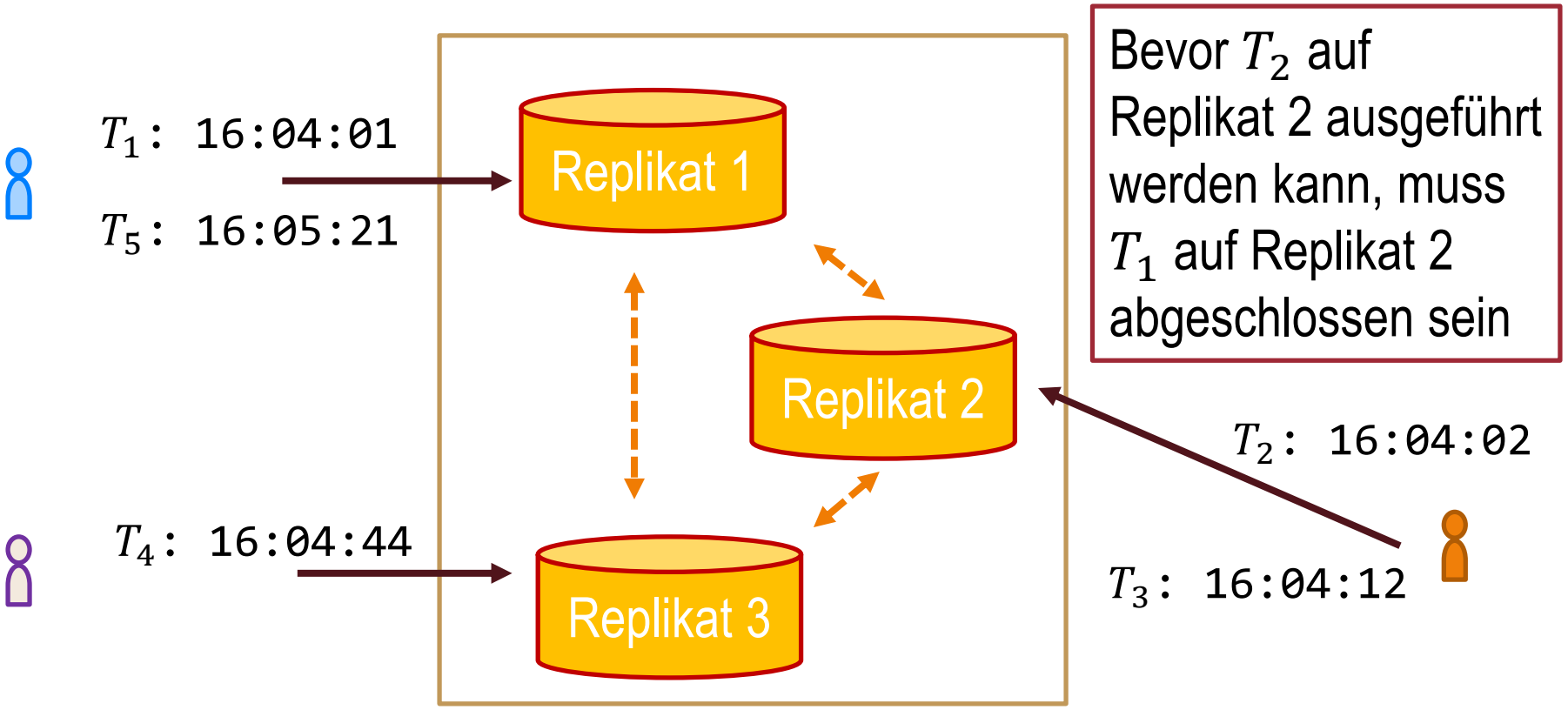
Verfügbarkeit

- **Verfügbarkeit** = Jede Anfrage an einen funktionierenden Server wird garantiert beantwortet
- In verteilten Systemen kann eine Aufrechterhaltung hoher Verfügbarkeit problematisch sein, wenn weitere **semantische Garantien bzgl der Inhalte der Antworten** einzuhalten sind, z.B.
  - Lesen des aktuellsten Werts
  - Änderungen gehen nicht verloren
- Replikation erfordert **Kommunikation/ Synchronisation** zwischen Rechnerknoten, um bestimmte Garantien einzuhalten
  - Bei Rechner-/Netzwerkfehlern können beliebig langen Wartezeiten entstehen (geringe Verfügbarkeit)
  - **Hohe Verfügbarkeit**: Nur ein Replikat ist erforderlich, um Anfrage zu beantworten



Goldstandard

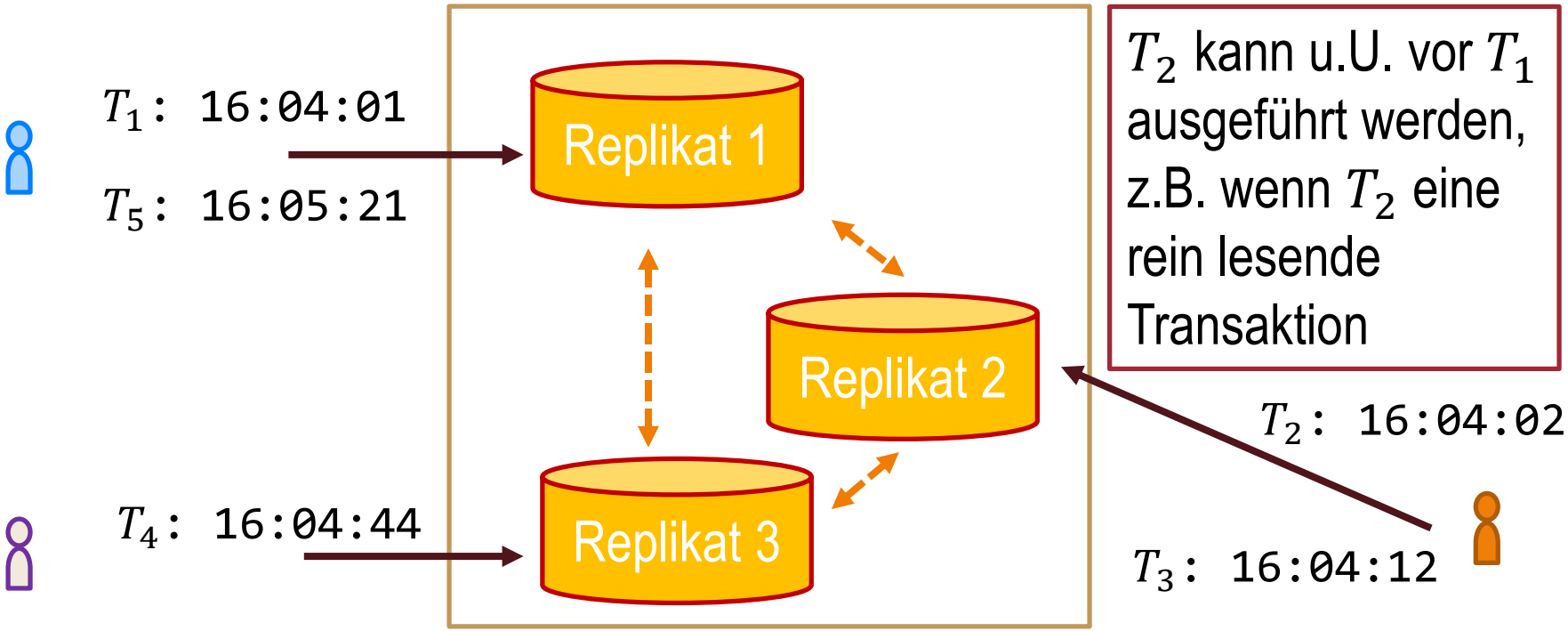
Strikte Serialisierbarkeit = alle Transaktionen erscheinen für alle Nutzer in der, *durch die reale Zeit gegebenen*, Reihenfolge



- Benötigt Sperrprotokolle für alle Transaktionen
- Blockierung bei Server-/Netzwerkfehler (geringe Verfügbarkeit)

Abschwächung

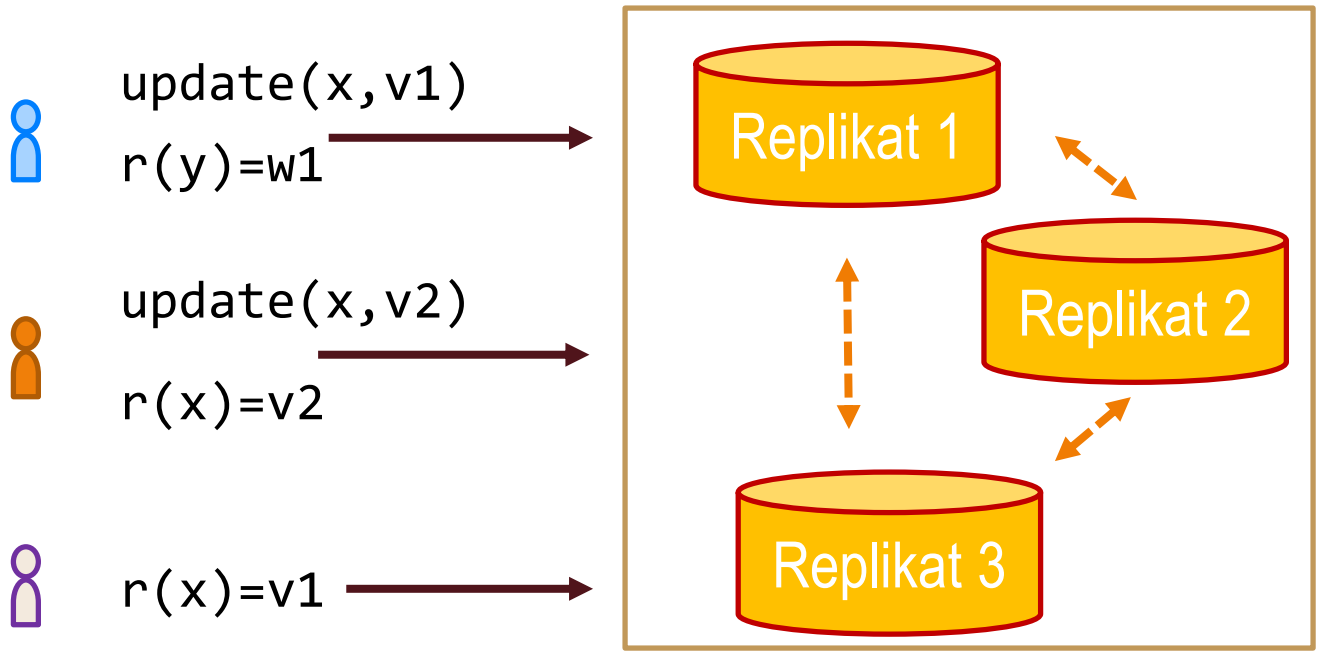
Serialisierbarkeit = alle Transaktionen erscheinen für alle Nutzer in derselben globalen Reihenfolge, z.B. $T_2 \rightarrow T_1 \rightarrow T_5 \rightarrow T_3 \rightarrow T_4$



- Mehrversionen-Synchronisation für rein schreibende Transaktionen
- Benötigt Sperrprotokolle für alle schreibenden Transaktionen
- Blockierung bei Server-/Netzwerkfehlern (geringe Verfügbarkeit)

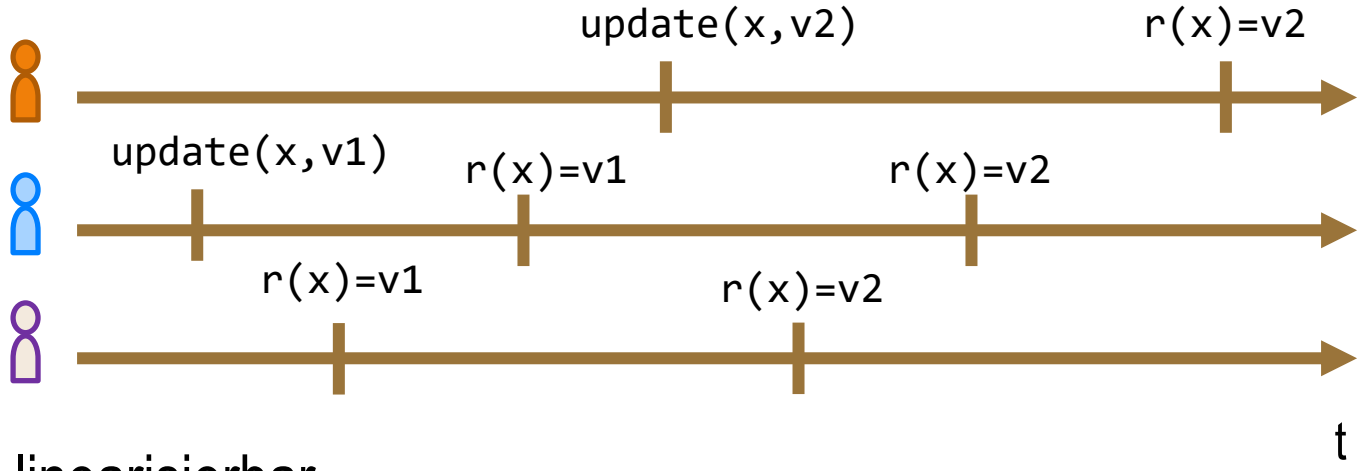
Konsistenzmodelle

- Weitere Abschwächungen der **semantischen Anforderungen** zur Erhöhung der Verfügbarkeit
- **Einschränkung**: Transaktionen mit ***einem Operator und einem Objekt***
  - Lesen einer Variable x: `r(x)` oder `r(y)`
  - Schreiben einer Version v1 auf Variable x: `update(x,v1)`
- Unter dieser Einschränkung: Strikt Serialisierbar = **Linearisierbar**
- Beispiel [Li10]

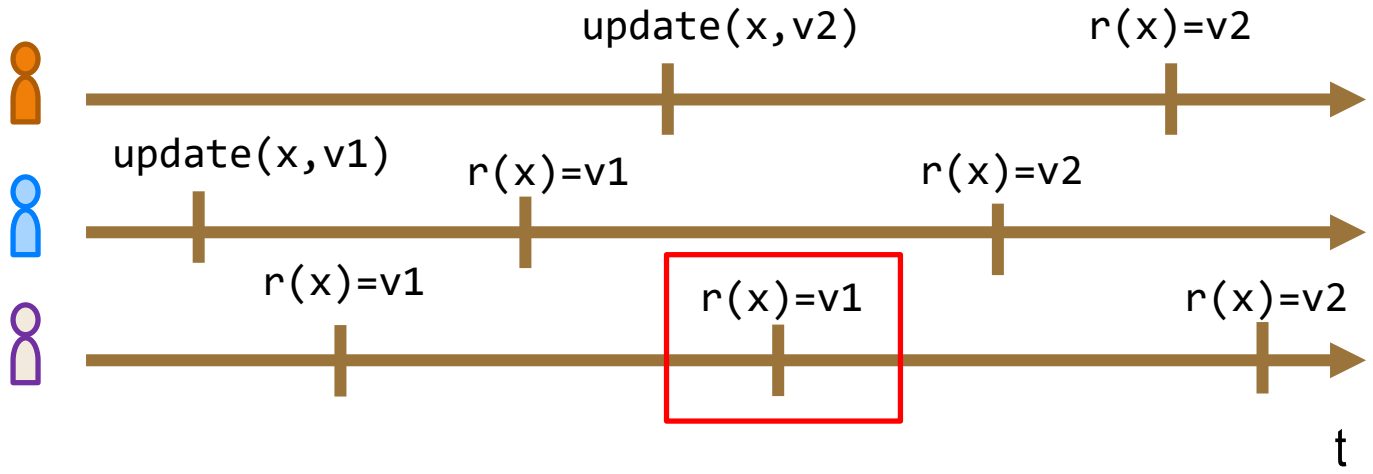


Konsistenzmodelle

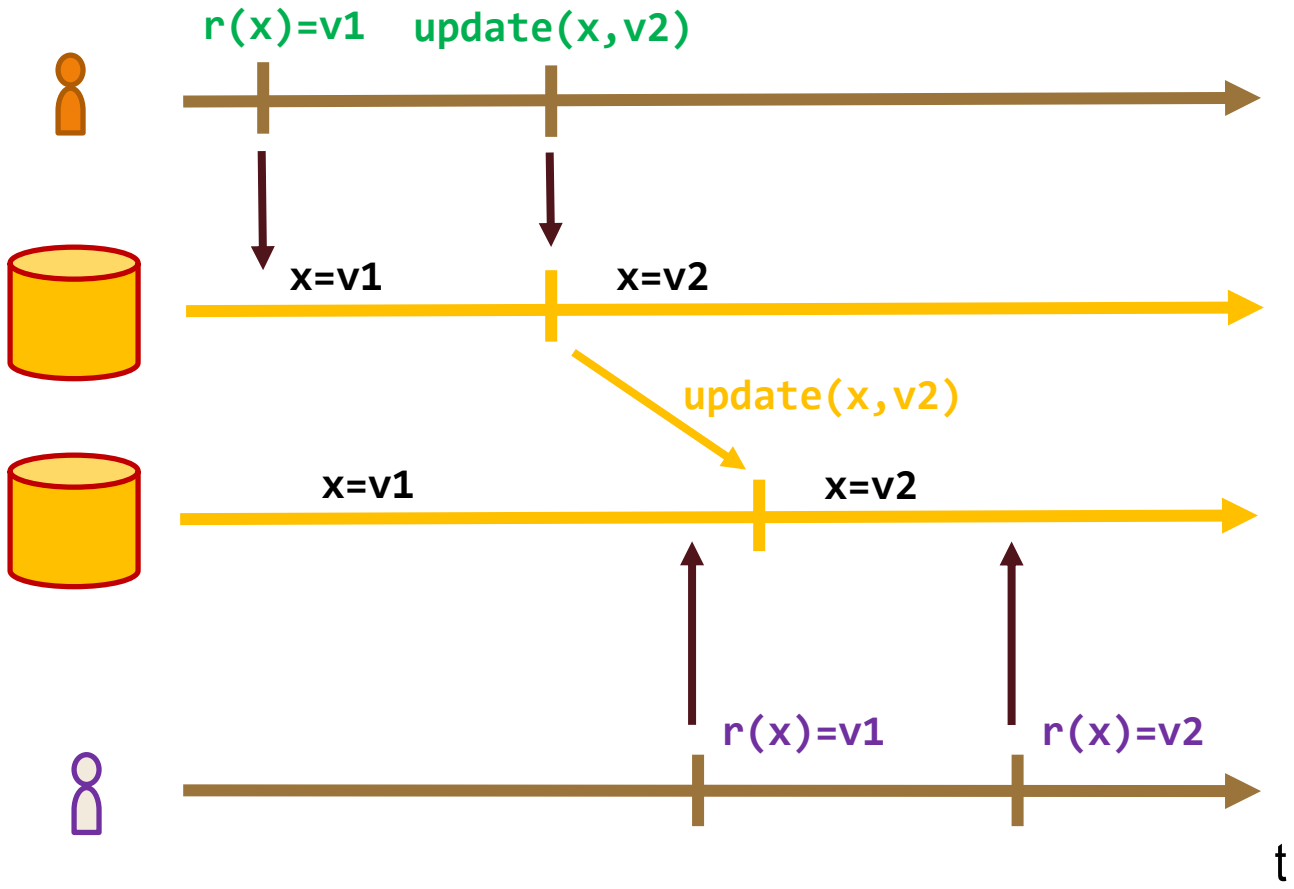
- Linearisierbar



- Nicht linearisierbar

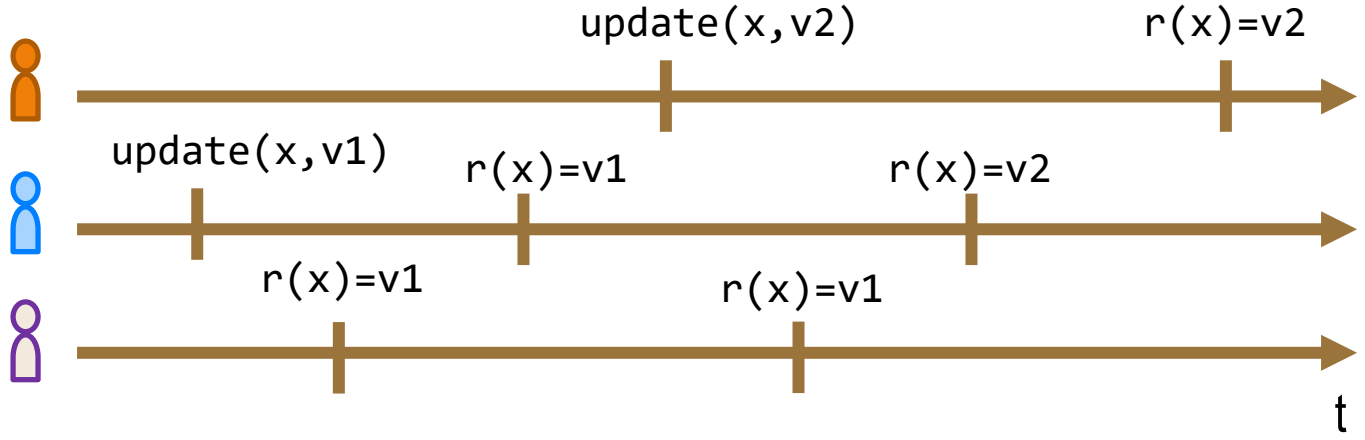


Probleme in verteilten Systemen

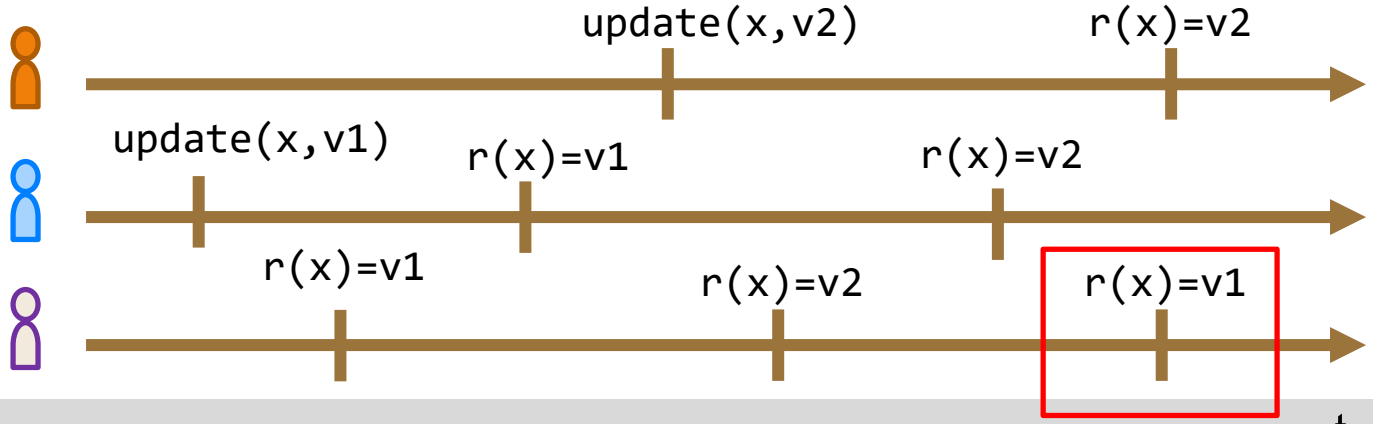


Konsistenzmodelle

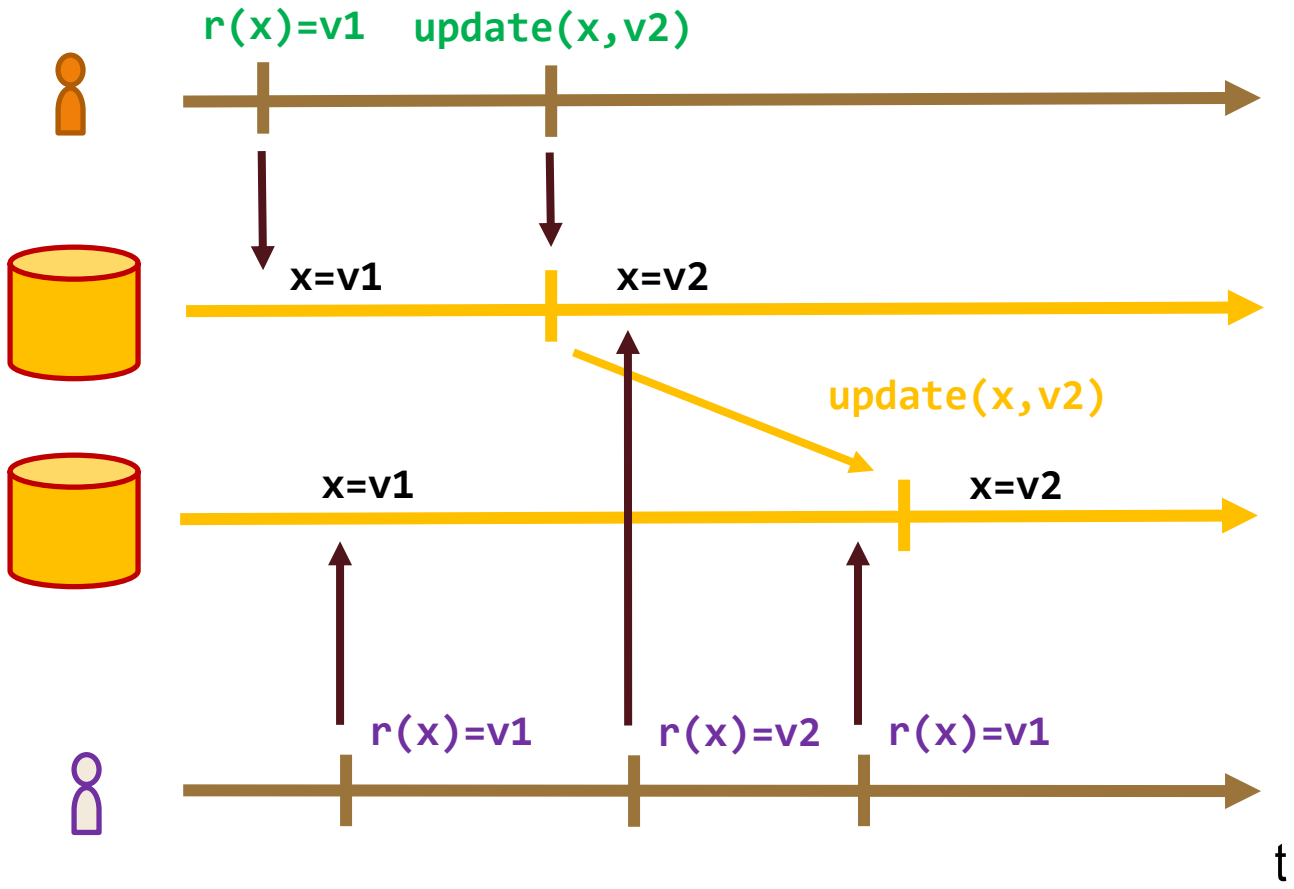
- **Sequentielle Konsistenz**: alle Operationen erscheinen für alle Nutzer in derselben globalen Reihenfolge *und diese Reihenfolge ist konsistent mit den einzelnen Nutzersitzungen* (sessions)



- Keine sequentielle Konsistenz

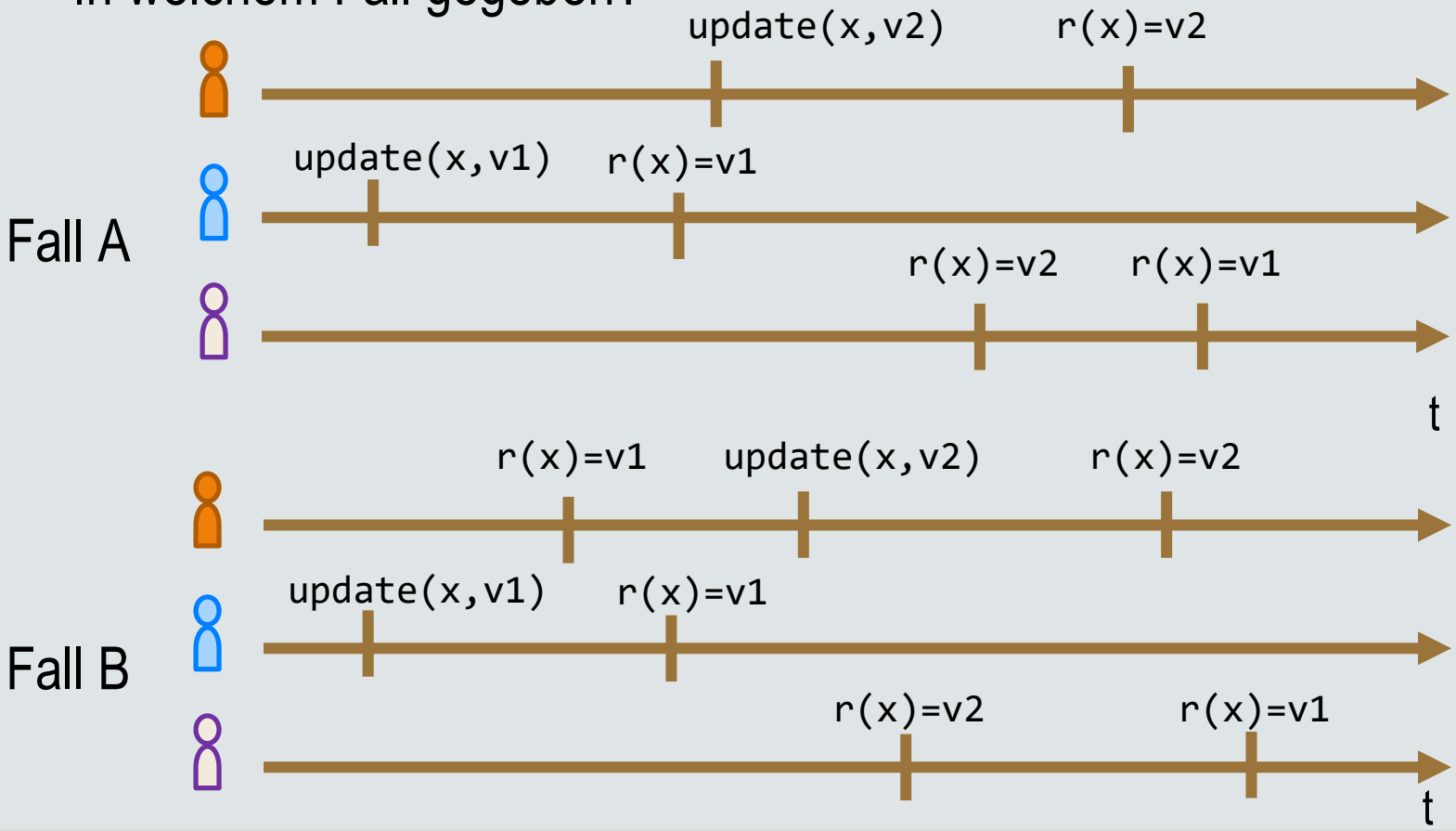


Probleme in verteilten Systemen



Übung: arsnova.rz.uni-leipzig.de 81 60 01 90

- Sequentielle Konsistenz: alle Operationen erscheinen für alle Nutzer in derselben globalen Reihenfolge und diese Reihenfolge ist konsistent mit den einzelnen Nutzersitzungen
- In welchem Fall gegeben?

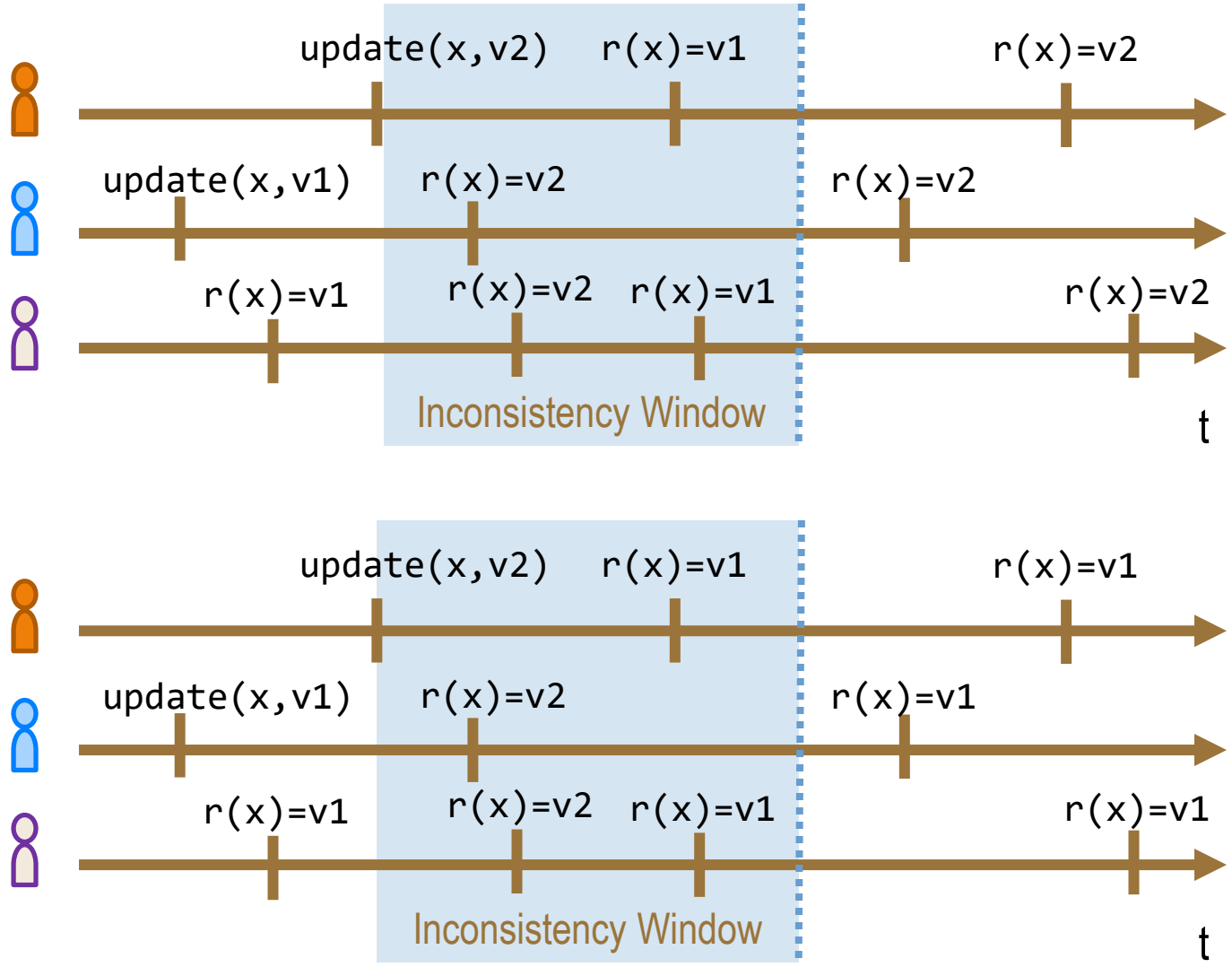


Inhaltsverzeichnis

BASE

- Sequentielle Konsistenz erfordert eine relativ aufwendige Synchronisation der Operationen zwischen Replikaten
- Dies vermindert die Verfügbarkeit und erhöht die Latenz
- Im Gegensatz dazu steht das **BASE-Prinzip**
- BA - Basically Available = Verfügbarkeit
- S - Soft State:
  - Werte sollten immer als vorläufig angesehen werden
  - Einträge werden letztlich mit aktuelleren Werten überschrieben
  - Lesen alter Werte ist möglich (stale reads)
- **E - Eventually Consistent**
  - DB kann in *inkonsistenten Zustand* kommen: Kopien der Daten auf verschiedenen Replikaten können für einen kurzen Zeitraum unterschiedliche Werte enthalten
  - *Letztendlich* enthalten alle Kopien die gleichen Werte
  - Kann zu einem gegebenen Zeitpunkt nicht ausgeschlossen werden

Eventual Consistency

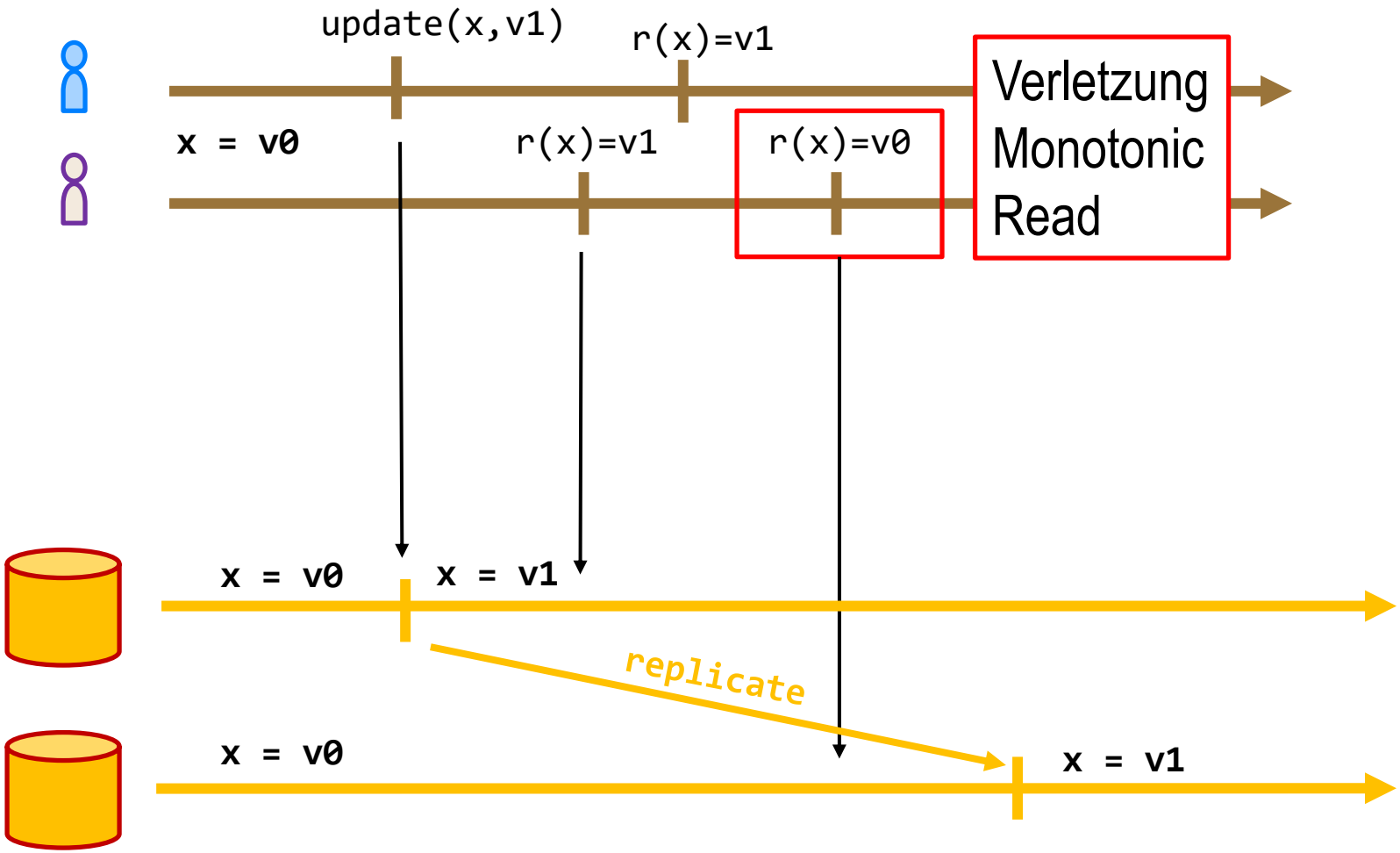


Eventual Consistency

- Priorität auf **Verfügbarkeit & Geschwindigkeit** anstatt einer (sequentiell) konsistenten Sicht auf die Daten
- Anfragen werden von nur einem Replikat bearbeitet (*hohe Verfügbarkeit)* und danach (lazy/asynchron) an andere Replikate weitergeleitet
- In manchen Kontexten ist Verfügbarkeit wichtiger als Konsistenz:
  - Messaging/E-Mailing
  - Statusaktualisierungen in Sozialen Medien
  - Aktualisierung des Warenkorbs im Online-Einkauf
- Nachteile:
  - Inkonsistenzen, Konflikte, Datenverlust
  - Anwendung kann sich nicht auf Datenqualität verlassen und muss mit verschiedenen Versionen umgehen
- Optimistischer Ansatz: Probleme entstehen *nur selten* und können leicht behoben werden

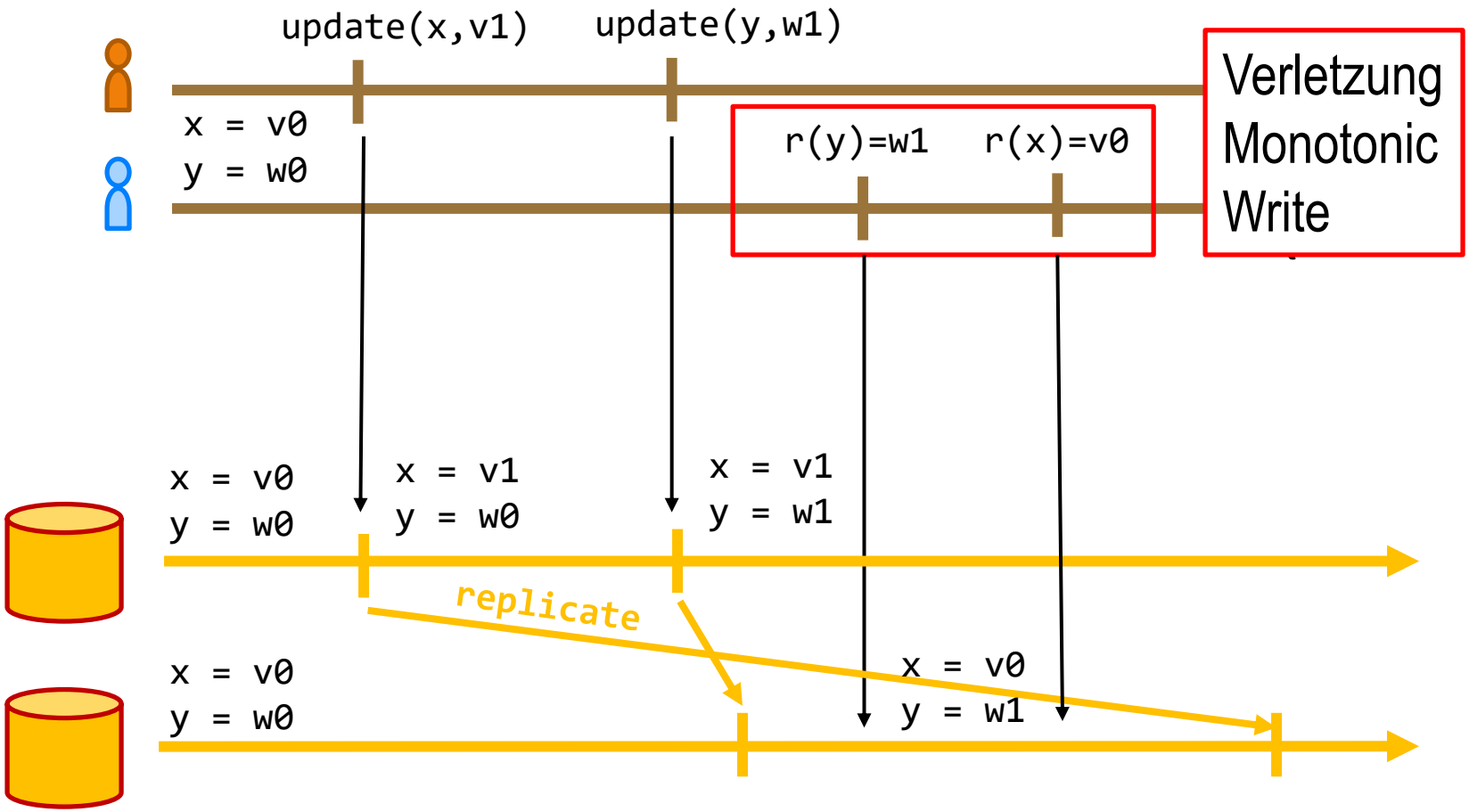
Sitzungsgarantien (session guarantees)

- **Monotonic Reads (MR)**: Ausschluss des Lesens älterer Werte **nachdem** ein aktuellerer Wert gelesen wurde



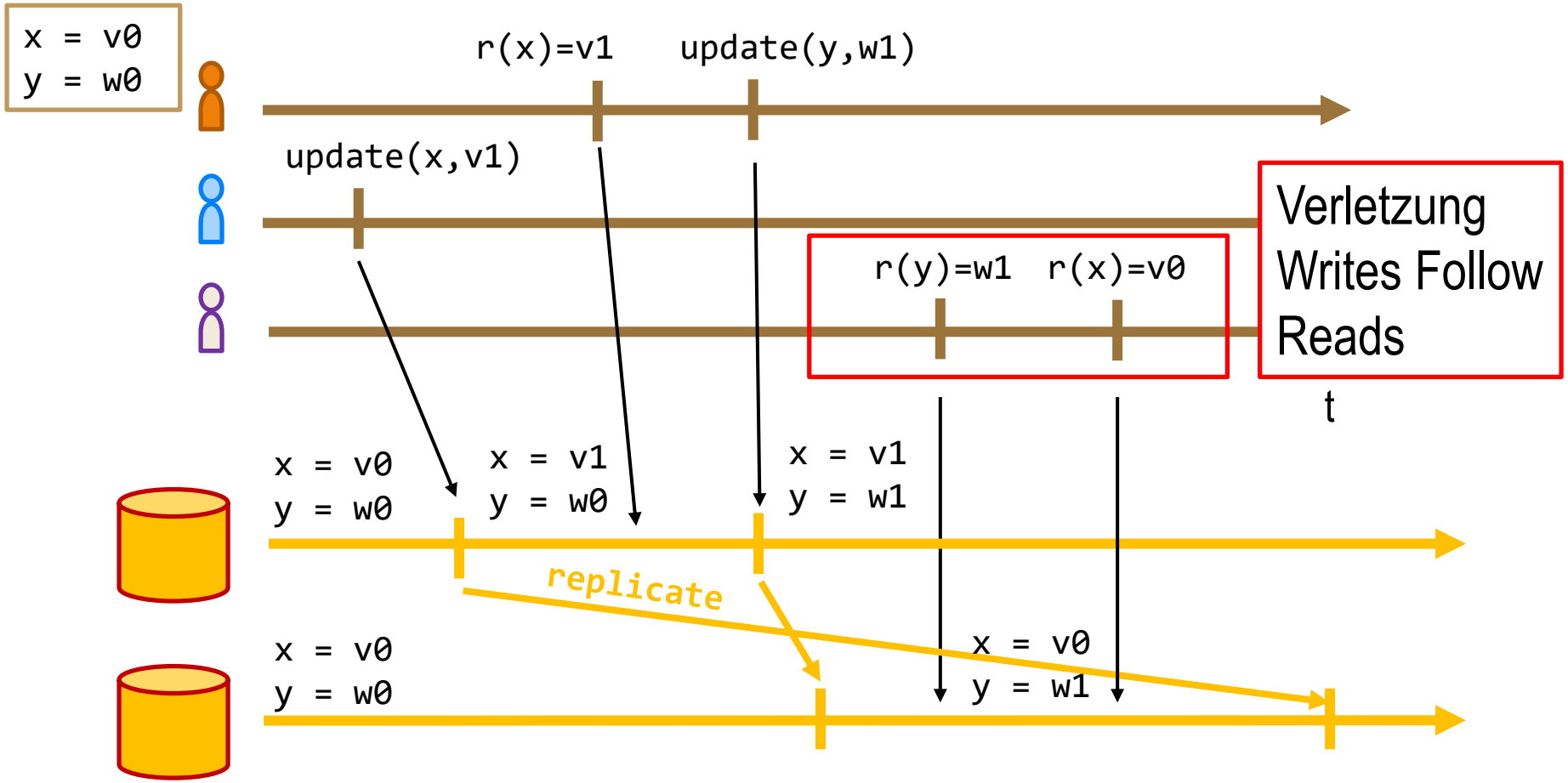
Sitzungsgarantien (session guarantees)

- **Montonic Write (MW)**: Aufeinanderfolgende Schreibbefehle des selben Nutzers werden von allen Nutzern in der gleichen Reihenfolge wahrgenommen



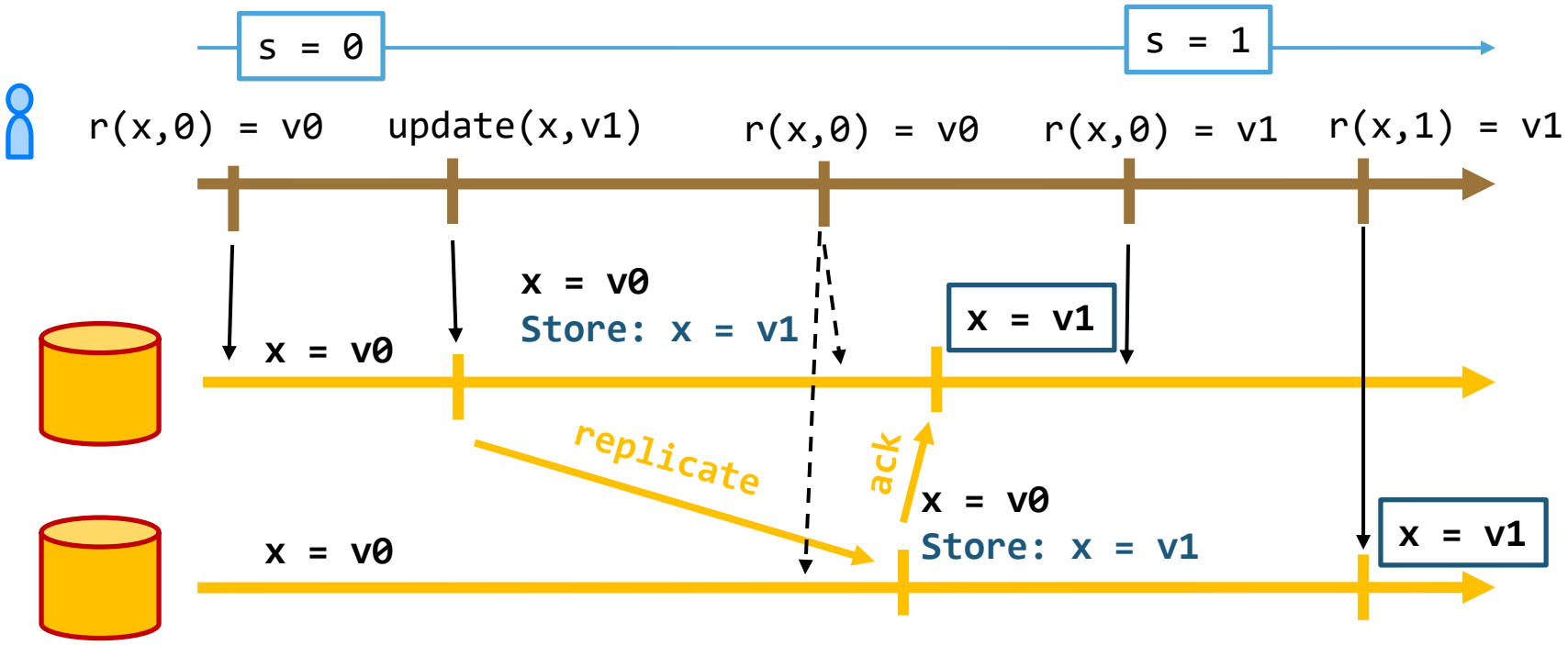
Sitzungsgarantien (session guarantees)

- **Writes Follow Reads (WFR)**: Wenn ein Nutzer einen Wert v liest, der aus einem Schreibvorgang u1 stammt, und später den Schreibvorgang u2 durchführt, muss u2 bei allen Nutzern nach u1 sichtbar sein



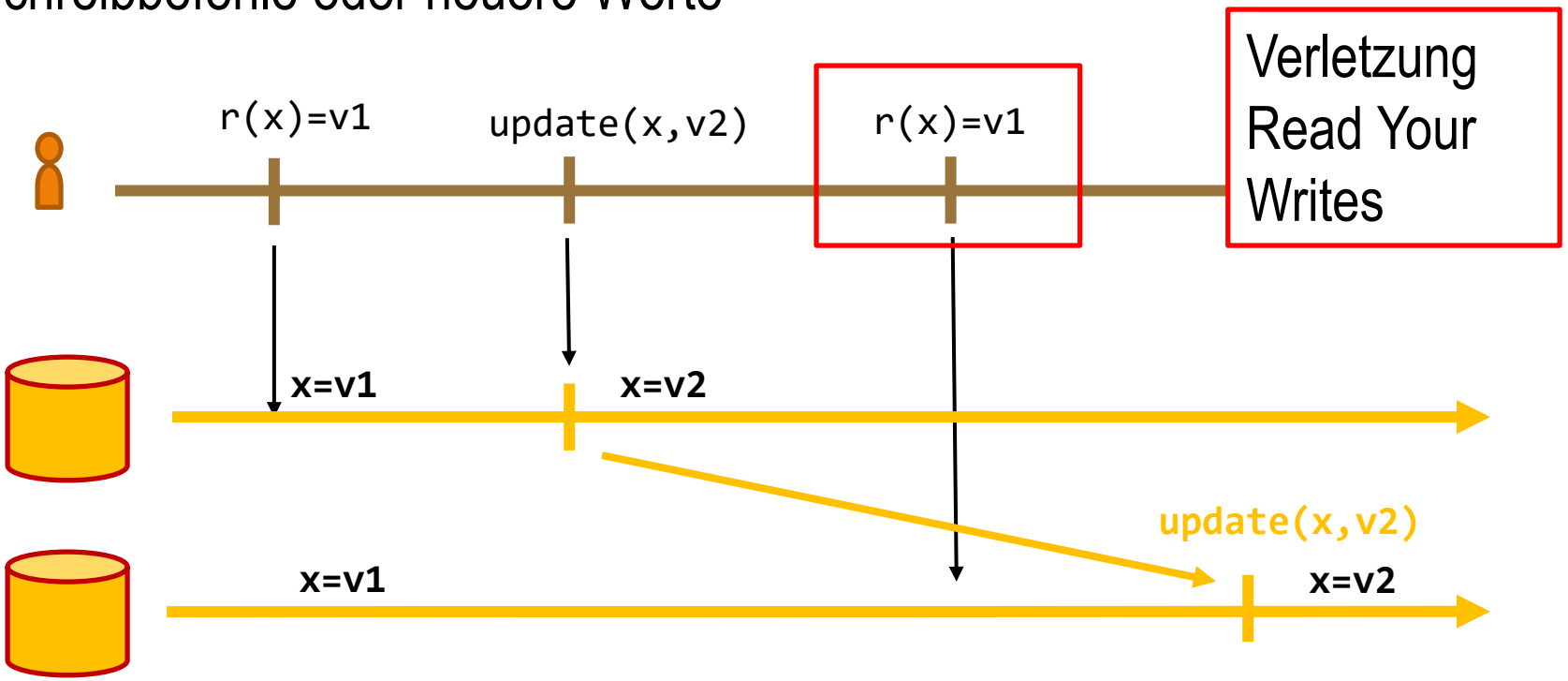
Sitzungsgarantien (session guarantees)

- [BDFGHS13]: **MR**, **MW**, und **WFR** sind möglich trotz **hoher Verfügbarkeit** (Anfragen werden von einem Replikat beantwortet, ohne Kommunikation mit andern Replikaten)
- z.B. über *Sitzungsversionsnummer* *s* (letzte bekannte Version) und indem die Effekte der Schreibbefehle erst dann für Nutzer **sichtbar werden**, wenn die jeweiligen Abhängigkeiten auf allen Replikaten erfüllt sind



Sitzungsgarantien (session guarantees)

- **Read Your Writes:** Jeder Nutzer liest die Werte seiner eigenen Schreibbefehle oder neuere Werte

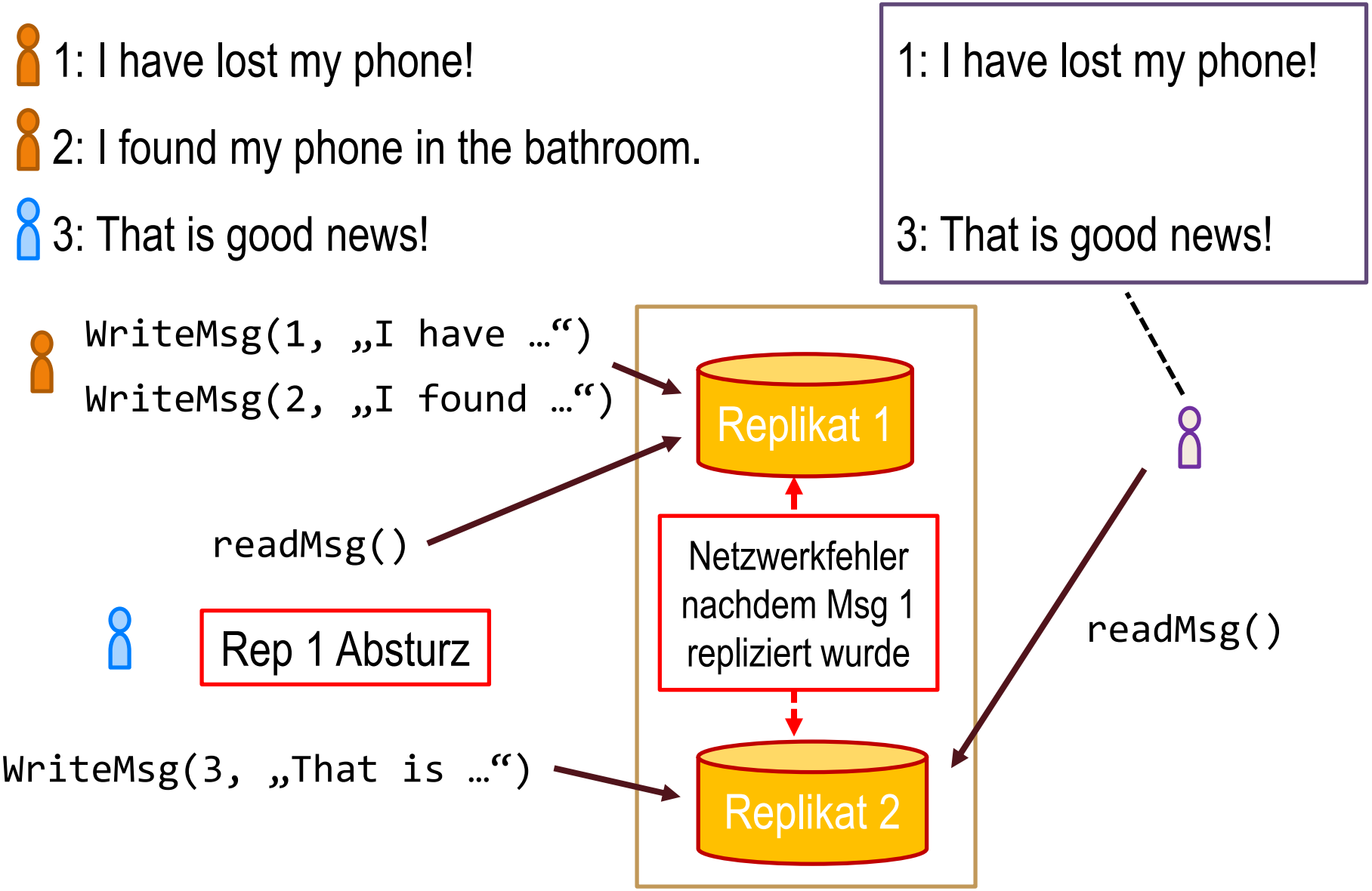


- Read Your Writes ist generell *nicht* mit hoher Verfügbarkeit vereinbar
  - z.B. bei Verzögerungen aufgrund von Netzwerkfehlern
  - Benötigt Zugehörigkeit zu einem festen Replikat („Stickiness“) [BDFGHS13] oder lokalen Cache [BGHS13]

Kausale Konsistenz

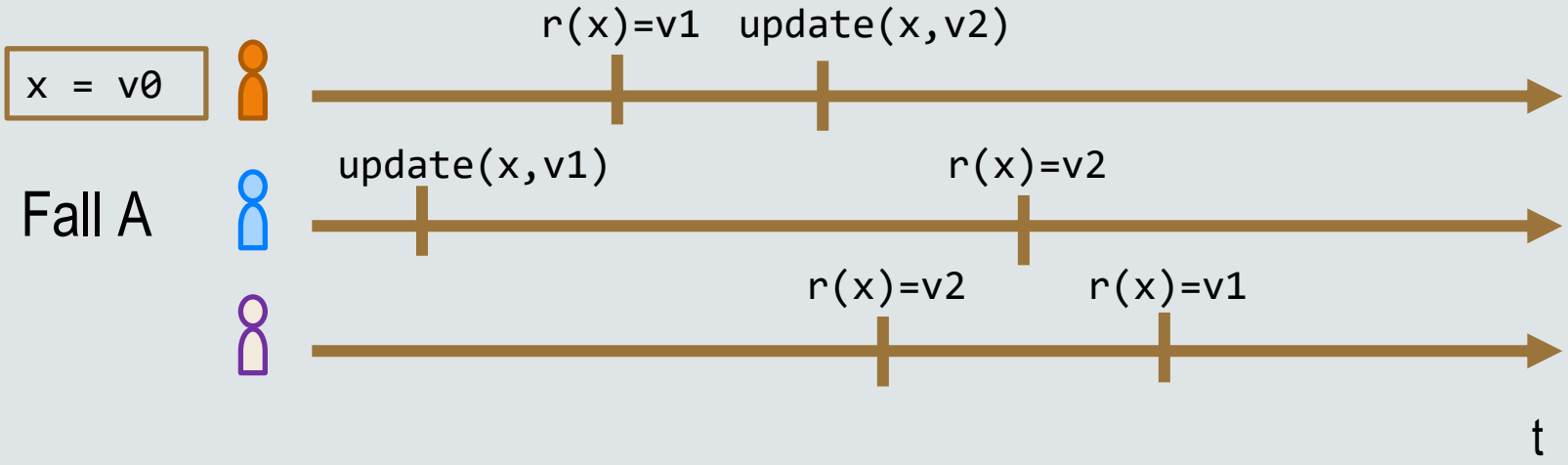
- Kompromiss zwischen seq. Konsistenz und eventual Consistency
  - *Kausal abhängige* Operationen sollten allen Nutzern in der gleichen Reihenfolge angezeigt werden
  - Die Reihenfolge kausal unabhängiger Operationen kann variieren
- Eine Operation T_2 ist ***kausal abhängig*** von T_1 ($T_1 \rightarrow T_2$), falls
  - der selbe Nutzer führte T_1 vor T_2 aus,
  - T_2 liest Werte, die durch T_1 geschrieben wurden, oder
  - es gibt eine Operation T_3 mit $T_1 \rightarrow T_3$ und $T_3 \rightarrow T_2$ (Transitivität).
- Eigenschaften:
  - Kausale Konsistenz ist mit **hoher Verfügbarkeit** vereinbar, falls Nutzer mit einem festen Replikat verbunden ist (oder lokaler Cache)
  - Kausale Konsistenz ist genau dann erfüllt, wenn die vier obigen Sitzungsgarantien erfüllt sind [BSW04]: MR, MW, WFR, RYW
  - Sequentielle Konsistenz impliziert kausale Konsistenz
  - Eventual Consistency wird *nicht* impliziert, aber generell zusätzlich angenommen

Beispiel: Keine kausale Konsistenz



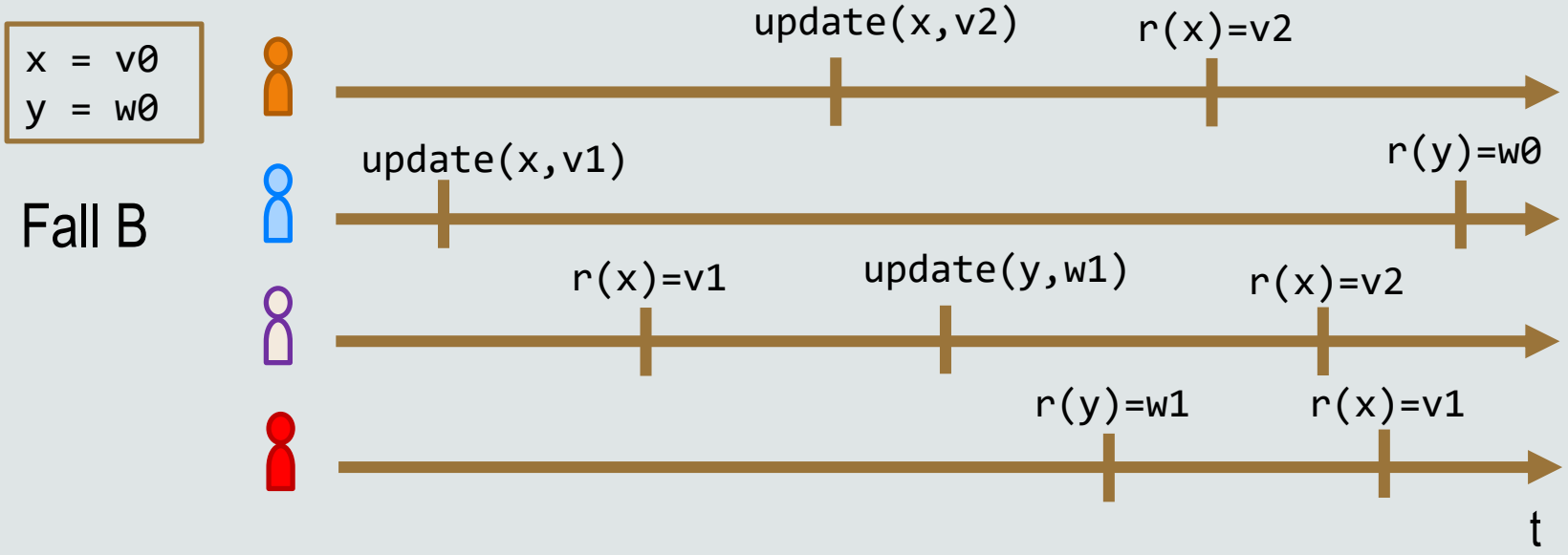
Übung: arsnova.rz.uni-leipzig.de 81 60 01 90

- Sequentielle Konsistenz?
- Monotonic Reads?
- Monotonic Writes?
- Read Your Writes?
- Writes Follow Reads?



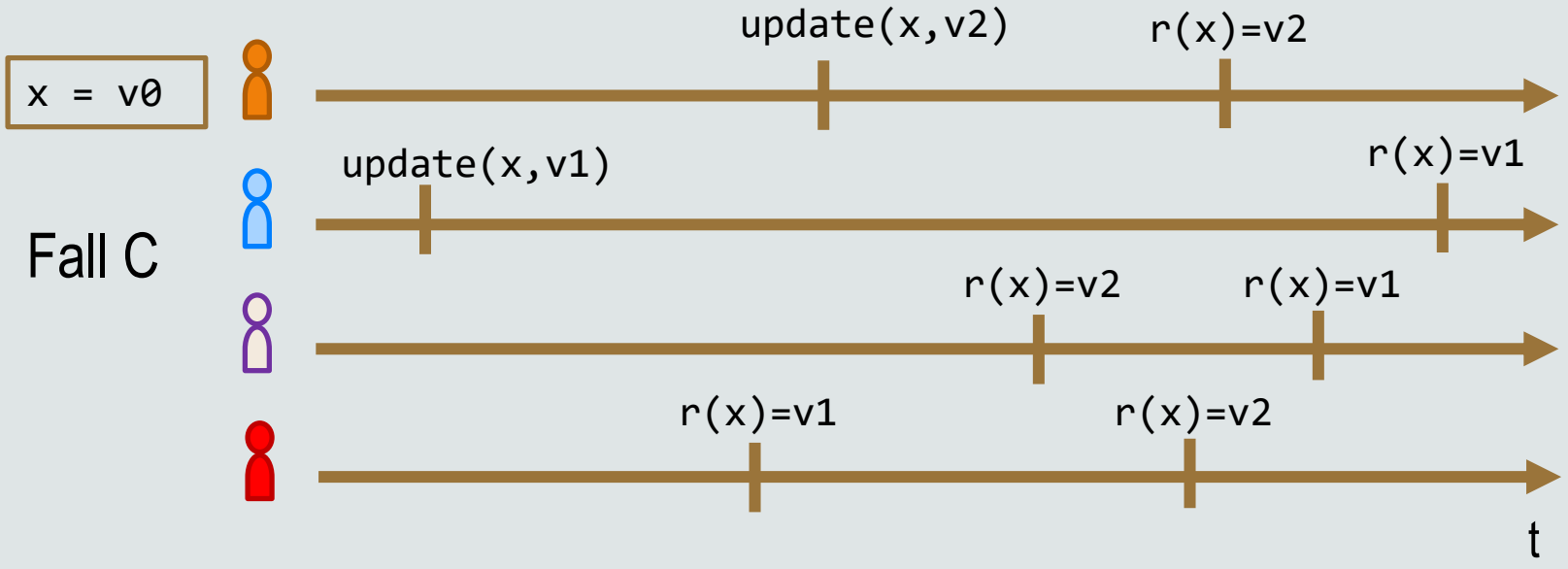
Übung: arsnova.rz.uni-leipzig.de 81 60 01 90

- Sequentielle Konsistenz?
- Monotonic Reads?
- Monotonic Writes?
- Read Your Writes?
- Writes Follow Reads?



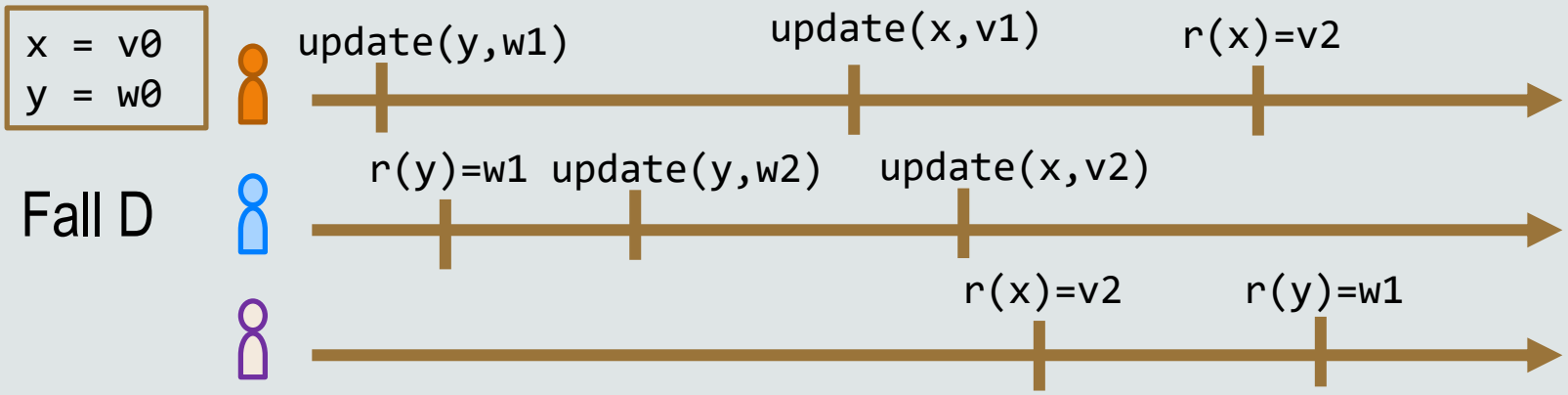
Übung: arsnova.rz.uni-leipzig.de 81 60 01 90

- Sequentielle Konsistenz?
- Monotonic Reads?
- Monotonic Writes?
- Read Your Writes?
- Writes Follow Reads?



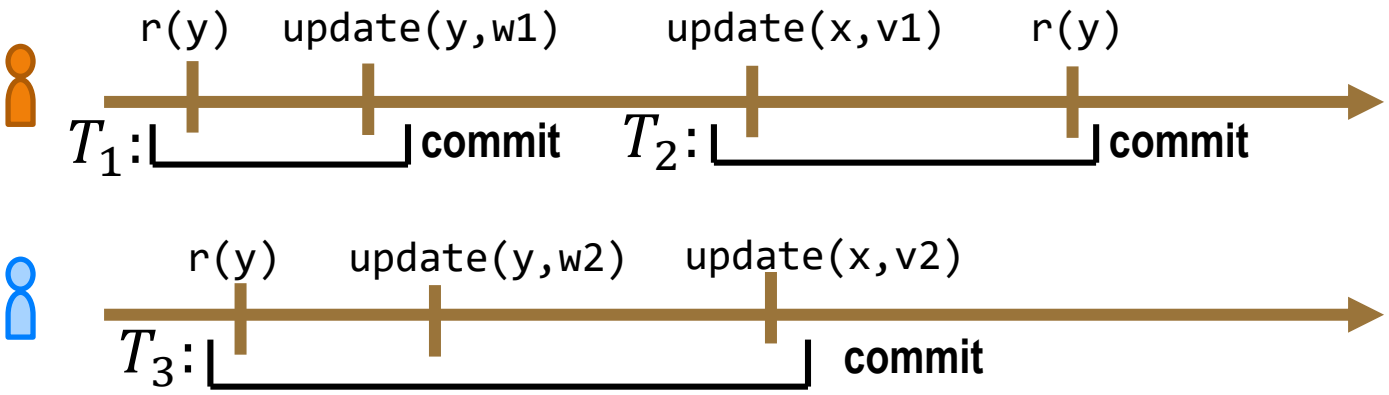
Übung: arsnova.rz.uni-leipzig.de 81 60 01 90

- Sequentielle Konsistenz?
- Monotonic Reads?
- Monotonic Writes?
- Read Your Writes?
- Writes Follow Reads?



Inhaltsverzeichnis

Transaktionen aus mehreren Operationen/Objekten



- **ACID**-Eigenschaften können teilweise unter *hoher Verfügbarkeit (**HA**)* garantiert werden
  - **Atomicity**: „Alles oder Nichts“ → **HA**
  - **Correctness**: eine erfolgreiche Transaktion erhält das DB-Schema (insb. Gewährleistung der definierten Integritätsbedingungen) → **nicht HA**
  - **Isolation**: alle Operationen innerhalb einer Transaktion müssen vor parallel ablaufenden Transaktionen verborgen werden → **teilweise HA**
  - **Durability**: Überleben von Änderungen erfolgreich beendeter Transaktionen trotz beliebiger (erwarteter) Fehler → **nicht HA**

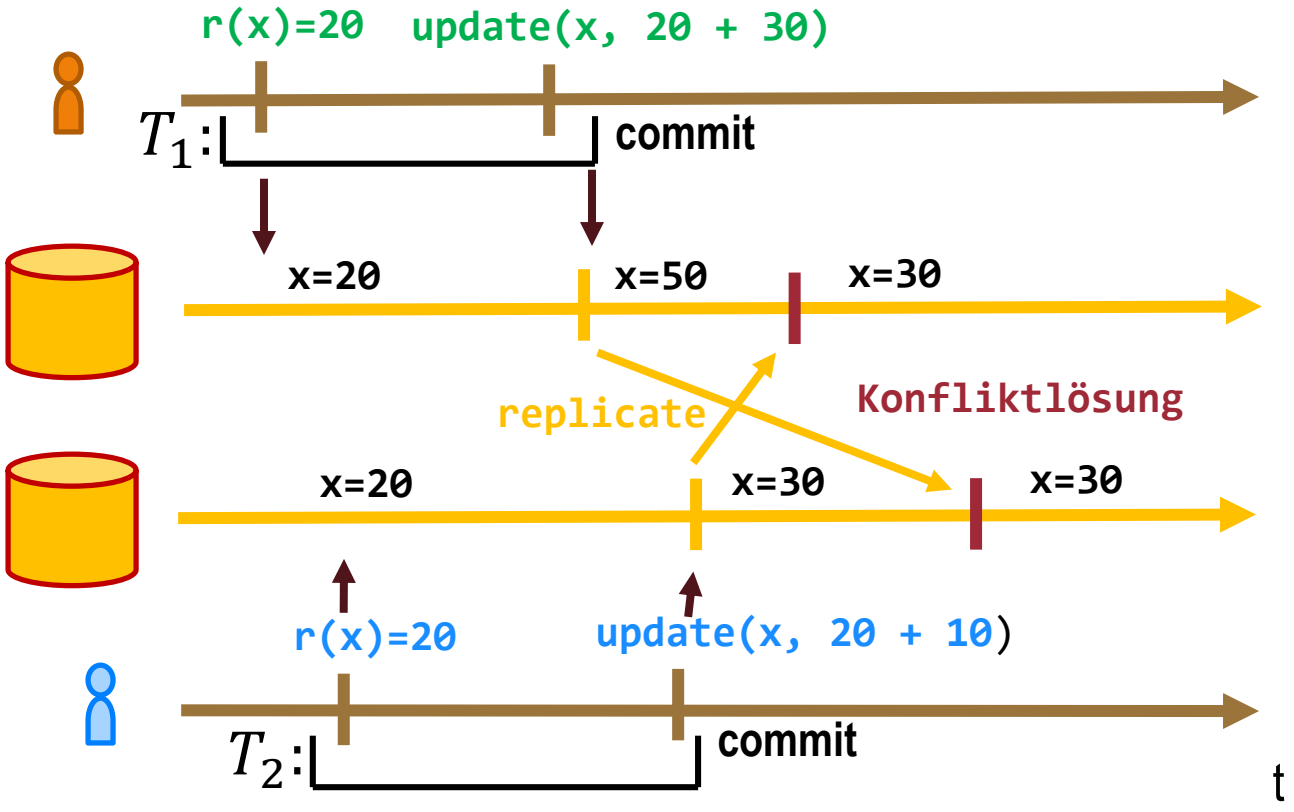
Eigenschaften ohne HA

- **Durability**: Überleben von Änderungen erfolgreich beendeter Transaktionen trotz beliebiger (erwarteter) Fehler
  - **Nicht garantierbar unter hoher Verfügbarkeit**
  - Damit eine Änderung den Ausfall von F Replikaten überlebt, müssen mindestens F+1 Replikate vor Commit kontaktiert werden
  - HA nur bei F = 0

- **Correctness**: eine erfolgreiche Transaktion erhält das DB-Schema
  - **Nicht garantierbar unter hoher Verfügbarkeit**
  - Lost Updates und Write Skews sind nicht unter HA vermeidbar
  - Für Correctness sollten beide Fehler nicht auftreten

- Die Garantie von Durability oder Correctness in verteilten Systemen benötigt (u.a.) ein Kommunikation bei Commit, wie z.B. durch das **2-Phasen-Commit Protokoll** gegeben

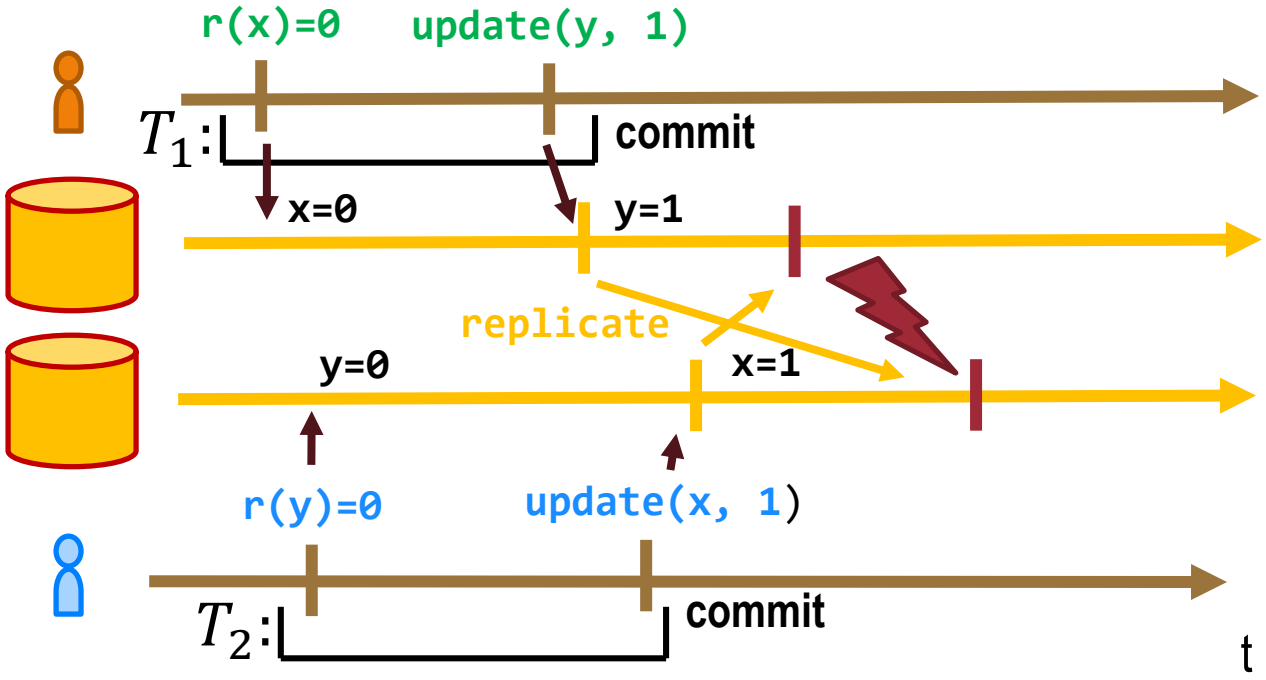
Lost Update



- Konfliktlösung: Last-Write-Wins (über Zeitstempel)
- Correctness (x = 20 + 30 + 10) erfordert Abbruch von T_2, da x inzwischen durch T_1 aktualisiert wurde
- Nur über Kommunikation zwischen Replikaten bei Commit möglich

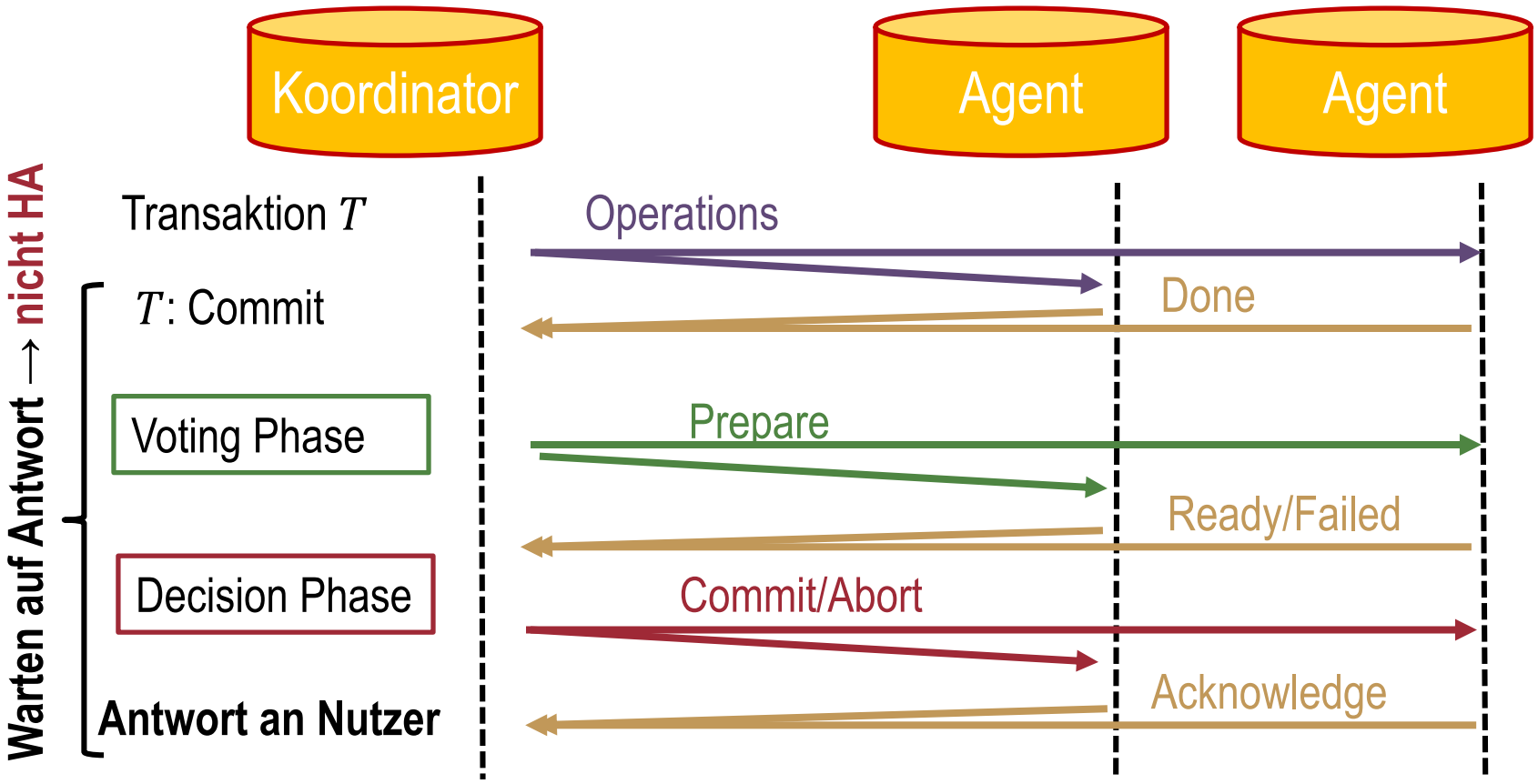
Write Skew

- Generalisierung des Lost Update auf verschiedene Objekte



- Konflikt unter *Integritätsbedingung*: not(x = 1 and y = 1)
- Correctness erfordert Abbruch von T_2, da y inzwischen durch T_1 aktualisiert wurde
- Nur über Kommunikation zwischen Replikaten bei Commit möglich

Kommunikation: 2-Phasen-Commit Protokoll



Weiteres Problem: Ausfall von Koordinator + Agent führt zur Blockierung

Fehlertolerante Alternativen: **Raft** und **Paxos Protokoll** (spätere Kapitel)

Eigenschaften mit HA

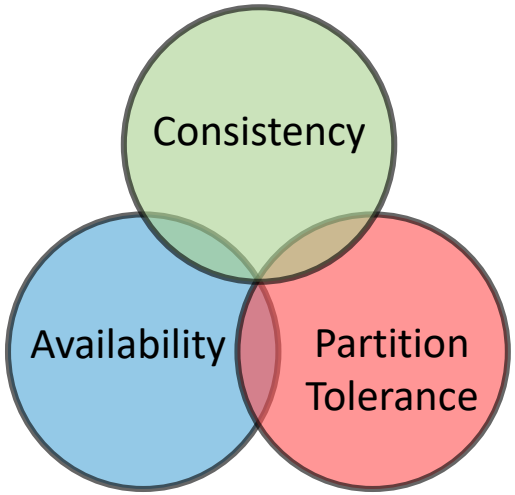
- **Atomicity**: garantierbar unter hoher Verfügbarkeit, z.B. über ähnliches Verfahren, wie bei Sitzungsgarantien [BDFGHS13]:
  - Metadaten: Zeitstempel und Liste der aktualisierten Werte
  - Zurückhalten der Aktualisierungen bis aktuelle Version aller Werte bei allen Replikaten angekommen

- **Isolation**: alle Operationen innerhalb einer Transaktion müssen vor parallel ablaufenden Transaktionen verborgen werden
  - „logischer Einbenutzerbetrieb“ = Serialisierbarkeit → **nicht HA**
  - Schwächere Anforderungen an Isolation sind mit HA möglich [BDFGHS13]:
    - **Read Committed**: Transaktionen sollten keine Objekte lesen oder schreiben, die in einer anderen Transaktion aktualisiert werden und deren Commit noch aussteht (Garantie indem Daten erst nach dem Commit repliziert werden)
    - **Item Cut Isolation:** das wiederholte Lesen des gleichen Objektes innerhalb einer Transaktion liefert stets den gleichen Wert (Garantie über Cache bei Nutzer)

Inhaltsverzeichnis

Trade-off: Verfügbarkeit vs. Datenkonsistenz

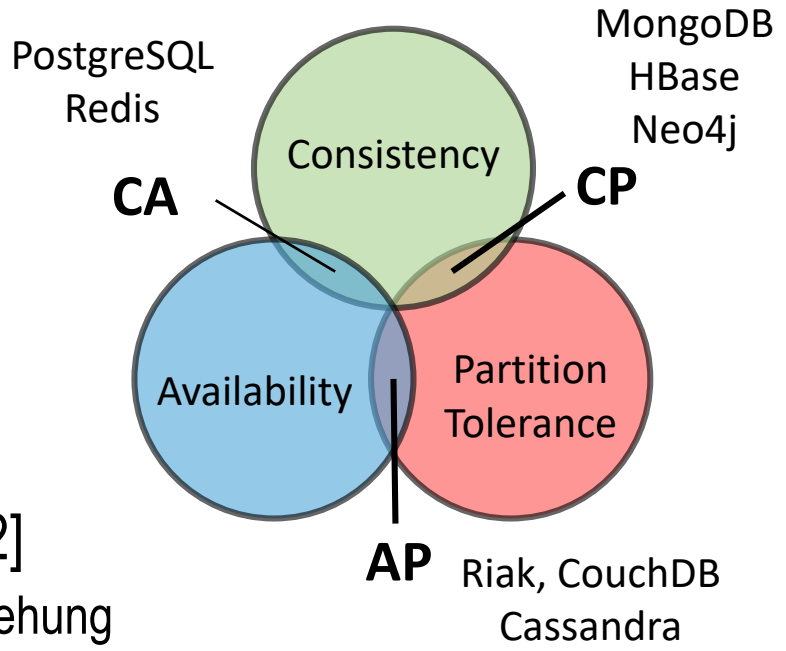
- Konsistenzprobleme durch parallele Zugriffe auf verschiedene Replikate und Netzwerkausfälle/-verzögerungen
- Linearisierbarkeit und andere ACID-Garantien sind in verteilten System nur durch ein hohes Ausmaß an Koordination und evtl. **Blockierungen** möglich
- CAP-Theorem [Bre00] [GL02]
  - Transaktionen mit nur einem Operator und Objekt
  - 3 Eigenschaften einer verteilten DB:
    - **C**onsistency = Linearisierbarkeit
    - **A**vailability = Verfügbarkeit
    - **P**artition Tolerance**:** System funktioniert trotz Netzwerkpartitionierung (Knoten aus einer Partition können nicht mehr mit Knoten aus anderer Partition kommunizieren)



Theorem: Ein verteiltes System kann maximal 2 der 3 Eigenschaften erfüllen.

CAP Theorem – Fälle und Kontroverse

- CA: in nicht-verteilten Systemen
- CP: konsistent aber nicht verfügbar bei Netzwerkpartitionierung
- AP: Verfügbar aber nicht konsistent bei Netzwerkpartitionierung



- Kontroverse: “2 of 3” ist irreführend [Bre12]
  1. Netzwerkpartitionen sind selten: Wie ist Beziehung zwischen Konsistenz und Latenzzeiten falls keine Netzwerkpartitionierung vorliegt (PACELC)?
  2. Wahl zwischen C und A auf verschiedenen Ebenen möglich
     - NoSQL-DB lassen sich nicht eindeutig in eine der drei Ecken einordnen
     - z.B. Riak, MongoDB, Cassandra
  3. C und A beschreiben zwei Enden eines Intervalls → *schwaches CAP*: falls hohe Garantien für zwei Eigenschaften erwünscht, dann Garantien der dritten Eigenschaft schwächer

Literatur

- [Bre00] Brewer. Towards robust distributed systems. Proceedings of the Annual ACM Symposium on Principles of Distributed Computing (2000) http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf
- [GL02] Gilbert and Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. ACM SIGACT News (2002)
- [Bre12] Brewer. CAP Twelve Years Later: How the "Rules" Have Changed, https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed
- [Li10] Jinyang Li: Distributed Systems, Lec 12: Consistency Models - Sequential, Causal, and Eventual Consistency, https://www.cs.columbia.edu/~du/ds/assets/lectures/lecture12.pdf
- [BSW04] J. Brzezinski, C. Sobaniec, and D. Wawrzyniak: From session causality to causal consistency. In PDP 2004
- [BGHS13] P. Bailis, A. Ghodsi, J. M. Hellerstein, I. Stoica: Bolt-on Causal Consistency, http://www.bailis.org/papers/bolton-sigmod2013.pdf
- [BDFGHS13] P. Bailis, A. Davidson, A. Fekete, A. Ghodsi, J. M. Hellerstein, Ion Stoica: Highly Available Transactions: Virtues and Limitations, http://www.vldb.org/pvldb/vol7/p181-bailis.pdf