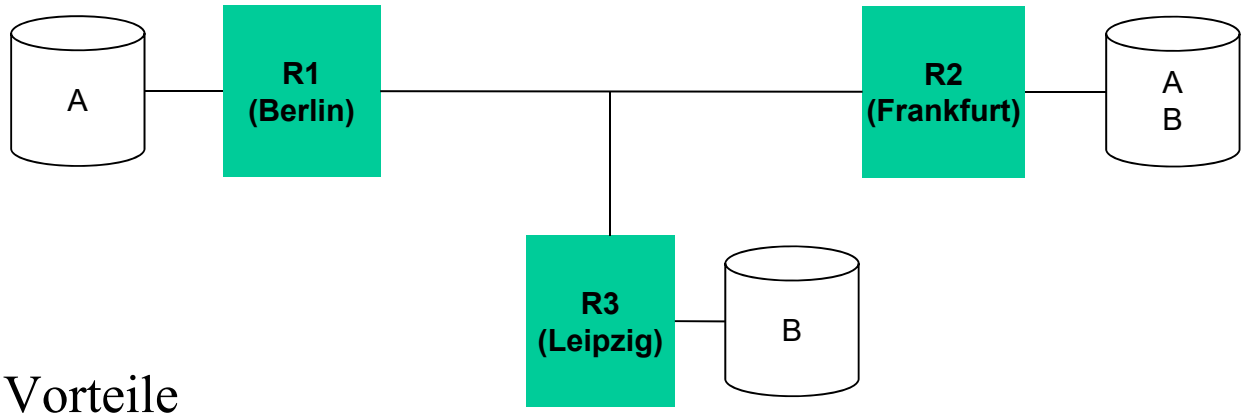
7. Replizierte Datenbanken

- Einführung (Replikationsstrategien)
- Update-Strategien bei strenger Konsistenz
  - ROWA-Verfahren / 2PC
  - Voting-Verfahren
- schwächere Formen der Datenreplikation
  - Refresh-Alternativen
  - Primary-Copy-Verfahren
  - Merge-Replikation
- Katastrophen-Recovery / Geo-Replikation
- Blockchain / distributed ledger
  - Einleitung
  - Kryptowährung Bitcoin

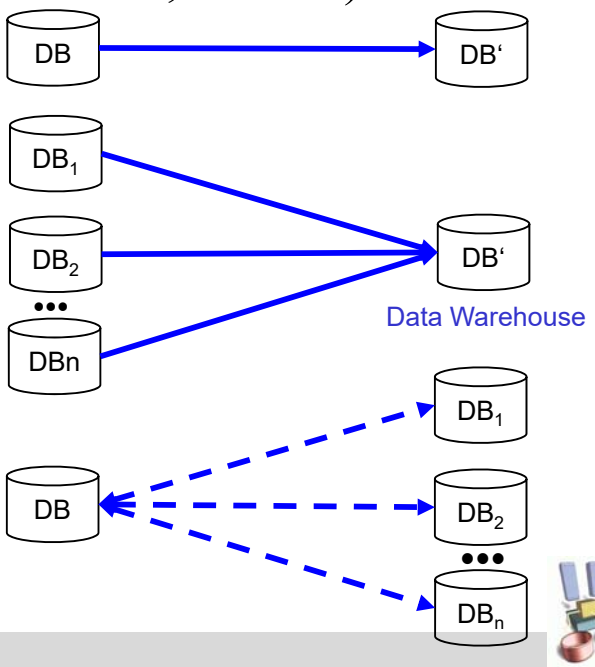
Replikate in Verteilten DBS



- Vorteile
  - erhöhte Verfügbarkeit der Daten
  - Beschleunigung von Lesezugriffen (bessere Antwortzeiten, Kommunikationseinsparungen)
  - erhöhte Möglichkeiten zur Lastbalancierung / Query-Optimierung
- Nachteile
  - hoher Update-Aufwand
  - erhöhter Speicherplatzbedarf
  - erhöhte Systemkomplexität

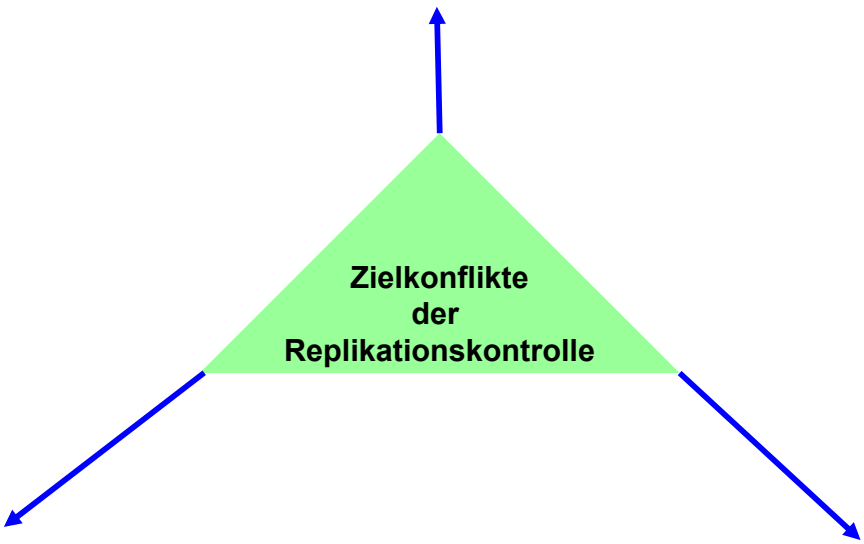
Replikation: Anwendungsmöglichkeiten

- Verteilte und Parallele DBS
  - replizierte DB-Tabellen / -Fragmente
  - replizierte Katalogdaten (Metadaten)
- Web: replizierte Daten und Metadaten für schnelles Lesen und Verfügbarkeit (Mirror Sites, Suchmaschinen, Portale)
- Katastrophen-Recovery
- Data Warehousing
  - Übernahme transformierter Daten in eigene Datenbank für Entscheidungsunterstützung
- Mobile Computing
  - Datenbank-Ausschnitte, Dateien ... auf Notebook, PDA etc.



Zielkonflikte der Replikationskontrolle

Erhaltung der Datenkonsistenz
- Kopien wechselseitig konsistent zu halten: *1-Kopien-Äquivalenz*
- kleine Kopienzahl



Erhöhung der Verfügbarkeit, effizienter Lesezugriff
- große Kopienzahl
- Zugriff auf beliebige und möglichst wenige Kopien

Minimierung des Änderungsaufwands
- kleine Kopienzahl
- möglichst wenige Kopien synchron aktualisieren

Replikationsstrategien (1)

- Korrektheitskriterium / Synchronisation

- 1-Kopien-Äquivalenz:
  - vollständige Replikationstransparenz: jeder Zugriff liefert jüngsten transaktionskonsistenten Objektzustand
  - alle Replikate sind wechselseitig konsistent
  - Serialisierung von Änderungen

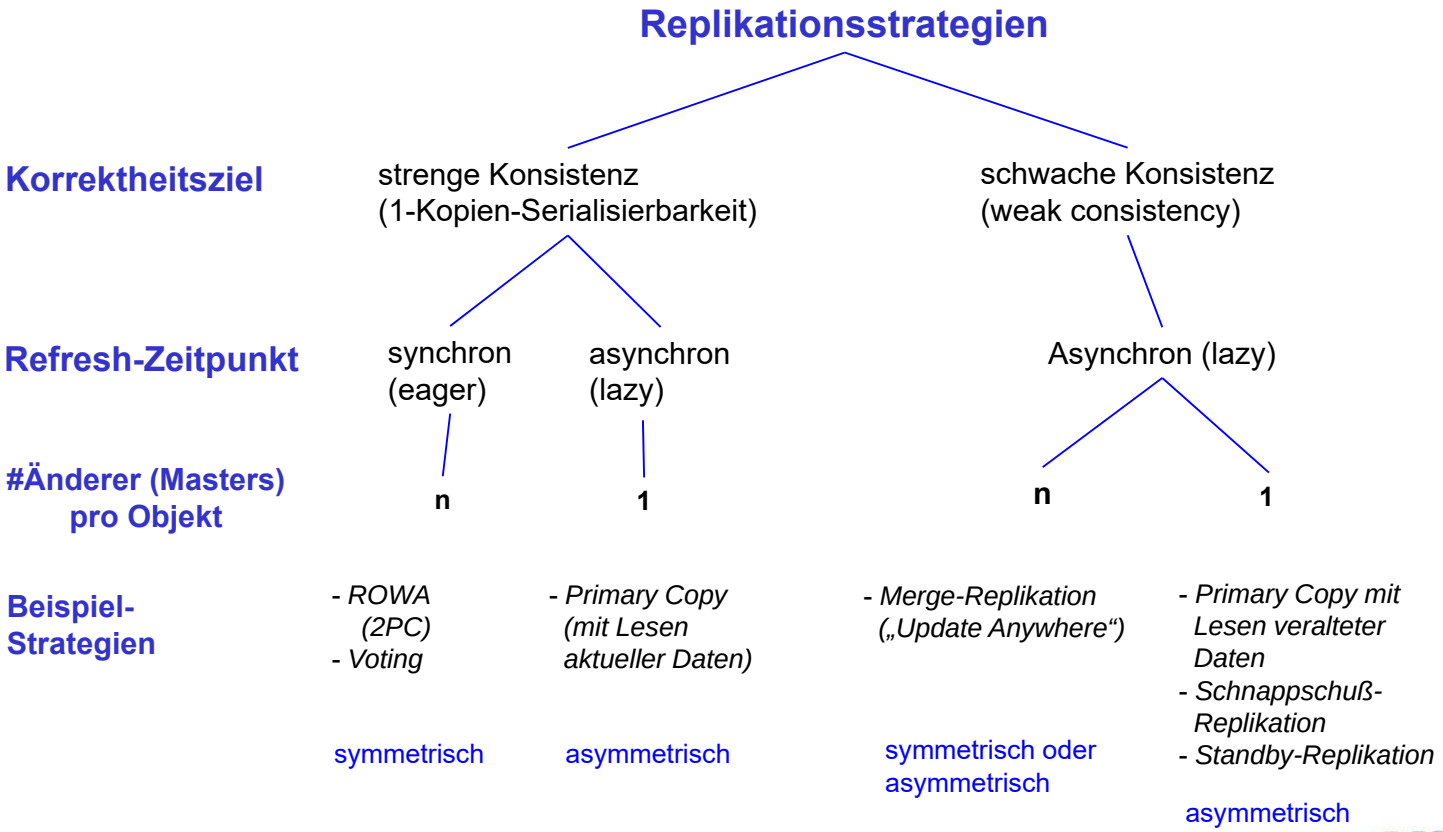
- geringere Konsistenzanforderungen
  - Zugriff auf ältere Objektversionen
  - evtl. keine strikte Serialisierung von Änderungen, sondern parallele Änderungen mit nachträglicher Konflikt-Behebung

Replikationsstrategien (2)

- symmetrische vs. asymmetrische Konfigurationen
  - symmetrisch: jedes Replikat gleichartig bezüglich Lese- und Schreibzugriffen; n Knoten können Objekt ändern
  - asymmetrisch (Master/Slave, Publish/Subscribe-Ansätze)
    - Änderungen eines Objekts an 1 Knoten (Master, Publisher, Primärkopie)
    - Lesen an n Knoten (Slave, Subscriber, Sekundärkopien)
  - Kombinationen (z.B. Multi-Master-Ansätze)

- Zeitpunkt der Aktualisierung: synchron oder asynchron
  - synchron (eager): alle Kopien werden durch Änderungstransaktion aktualisiert
  - asynchron (lazy): verzögertes Aktualisieren (Refresh)

Grobklassifikation



Write-All/Read-Any-Strategie (Read-One/Write-All, ROWA)

- bevorzugte Behandlung von Lesezugriffen
  - Lesen eines beliebigen (lokalen) Replikats
  - hohe Verfügbarkeit für Leser
- sehr hohe Kosten für Änderungstransaktionen
  - Schreibsperren bei allen Replikaten anfordern
  - Propagierung der Änderungen im Rahmen des Commit-Protokolls (2PC)
- Verfügbarkeitsproblem: Änderungen von Verfügbarkeit aller kopienhaltenden Knoten abhängig
  - sei p die mittlere Wahrscheinlichkeit, daß ein Knoten verfügbar ist
  - bei N Kopien beträgt die Wahrscheinlichkeit, daß *geschrieben* werden kann: p^N
  - Wahrscheinlichkeit, daß *gelesen* werden kann: $1 - (1 - p)^N$

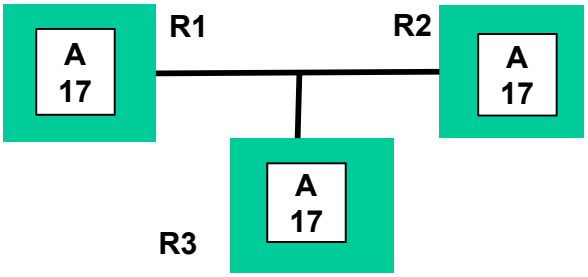
Beispiel p=0.9, N=2: Schreibverfügbarkeit:
Leseverfügbarkeit:

Write All Available

- ROWA-Variante, bei der nur verfügbare Replikate geändert werden
  - für ausgefallene Rechner werden Änderungen bei Wiederanlauf nachgefahren
  - funktioniert nicht bei Netzwerk-Partitionierungen

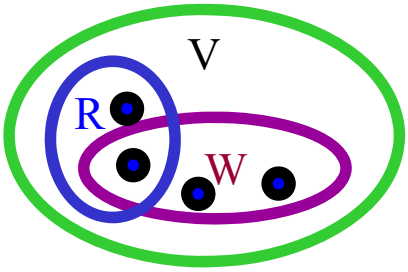
Mehrheits-Votieren (Majority Voting)

- Lesen oder Schreiben eines Objektes verlangt Zugriff auf Mehrheit der Replikate
- jedes Replikat kann gleichzeitig von mehreren Transaktionen gelesen, jedoch nur von einer Transaktion geändert werden
- Bestimmung der aktuellsten Version z. B. über Änderungszähler
- Fehlerfall: Fortsetzung der Verarbeitung möglich, solange Mehrheit der Replikate noch erreichbar
- hohe Kommunikationskosten für Lese- und Schreibzugriffe
  - ungeeignet für N=2 („Read All, Write All“)



Gewichtetes Votieren (Quorum Consensus)

- jede Kopie erhält bestimmte Anzahl von Stimmen (votes)
- Protokoll:
  - Lesen erfordert R Stimmen (Read-Quorum)
  - Schreiben erfordert W Stimmen (Write-Quorum)
  - R + W > V (= Summe aller Stimmen)
  - W > V/2

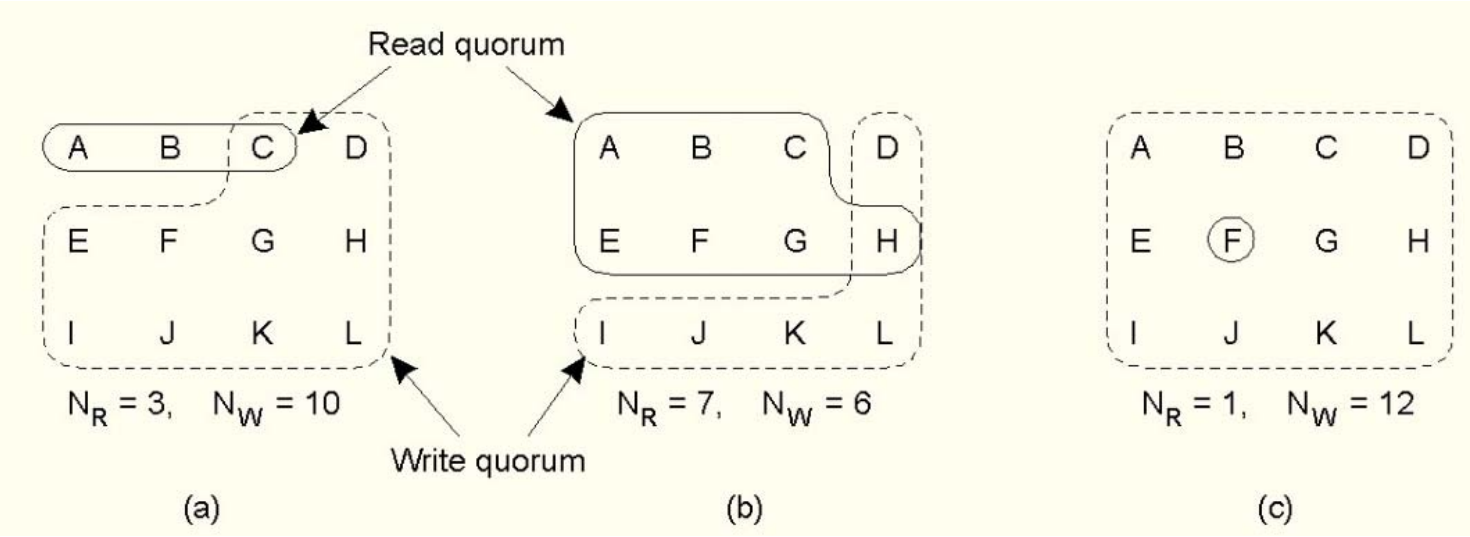


- Eigenschaften:
  - gleichzeitiges Lesen und Schreiben nicht möglich
  - jeder Zugriff auf R (W) Kopien enthält wenigstens 1 aktuelle Kopie
  - Festlegung von V, R und W erlaubt Trade-Off zwischen Lese- und Schreibkosten sowie zwischen Leistung und Verfügbarkeit

Gewichtetes Votieren (2)

- andere Verfahren ergeben sich als Spezialfälle
  - Beispiel : 3 Kopien mit jeweils 1 Stimme (V=3)
  - ROWA:
  - Majority Voting (Consensus):
  - 4 Kopien mit folgender Stimmenverteilung <2,2,2,1>

Gewichtetes Votieren (3)



source: Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms, Prentice-Hall, Inc. 2002

7. Replizierte Datenbanken

- Einführung (Replikationsstrategien)
- Update-Strategien bei strenger Konsistenz
  - ROWA-Verfahren / 2PC
  - Voting-Verfahren
- schwächere Formen der Datenreplikation
  - Refresh-Alternativen
  - Primary-Copy-Verfahren
  - Merge-Replikation
- Katastrophen-Recovery / Geo-Replikation
- Blockchain / distributed ledger
  - Einleitung
  - Kryptowährung Bitcoin

Schwächere Replikationsformen

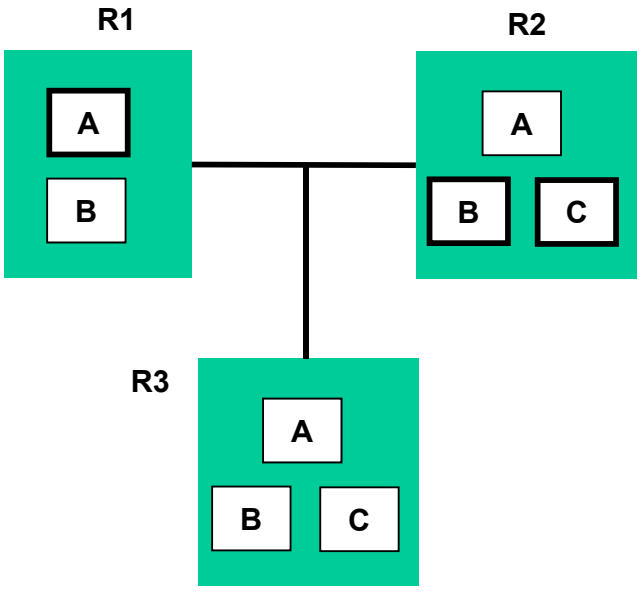
- gravierende Probleme bei vollständiger Replikationstransparenz
  - hohe Änderungskosten bei synchroner Propagierung von Änderungen
  - Verfügbarkeitsprobleme (besonders bei geographischer Verteilung)
  - geringe Skalierbarkeit!
  - nicht anwendbar für mobile Replikate (meist offline)
- DBS-Hersteller (Oracle, IBM, MS) unterstützen schwächere Konsistenzformen mit asynchroner Aktualisierung der Kopien
  - Schnappschuß-Replikation (materialisierte Sichten für Lesezugriffe)
  - "fast" aktuelle Kopien, z.B. für Unterstützung einer schnellen Katastrophen-Recovery
- bei n Änderungsknoten pro Objekt („Update Anywhere“):
  - asynchrones Refresh führt zu Konsistenzproblemen
  - anwendungsspezifische Konflikt-Behebung (Merge-Replikation)

Refresh-Alternativen

- baldmögliche Weiterleitung (ASAP) nach Commit der Änderungstransaktion oder verzögert (Timer, #Änderungen)
  - Verzögerungsdauer: Aktualität vs. Overhead
- Push- vs. Pull-Replikation
  - Push: Notifikation von Änderungen durch Master / Publisher
  - Pull: Probing durch Slave/Subsriber (z. B. mobiles Endgerät)
- einfache Replikate (Kopien) vs. abgeleitete Replikation (Ergebnisse einer SQL-Operation)
- vollständige vs. inkrementelle Aktualisierung von Replikaten
  - Übertragung der Objektkopien / Werte
  - Verwendung von Triggern
  - Übertragung der Log-Daten

Primärkopien-Verfahren

- asymmetrisches Verfahren
  - 1 Primärkopie (primary copy): *publisher*
  - n Sekundärkopien pro Objekt: *subscriber*
- Bearbeitung von Schreib- und Lesesperren beim Primärkopien-Rechner
- synchrone Änderungen nur für Primärkopie
- verzögerte/asynchrone Aktualisierungen der Sekundärkopien
  - in der selben Commit-Reihenfolge wie bei Primärkopien-Rechner
  - Verwendung von Sequenz-Nummern



Die Primärkopien verschiedener Objekte können auf verschiedenen Rechnern liegen

Primärkopien-Verfahren: Lesezugriffe

mehrere Alternativen

- Sperren und Lesen der Primärkopie
  - aktuelle Daten
  - Replikation wird nicht genutzt!
- lokale Kopie lesen ohne Sperre
  - effizient
  - lokale Kopie kann leicht veraltet sein (keine 1-Kopien-Serialisierbarkeit)
- Lesesperre beim Primärkopien-Rechner + Lesen einer aktuellen (lokalen) Kopie
  - z.B. Test über Versionsnummer bei Primärkopien-Rechner
  - strenge Konsistenz, jedoch geringe Kommunikationseinsparung

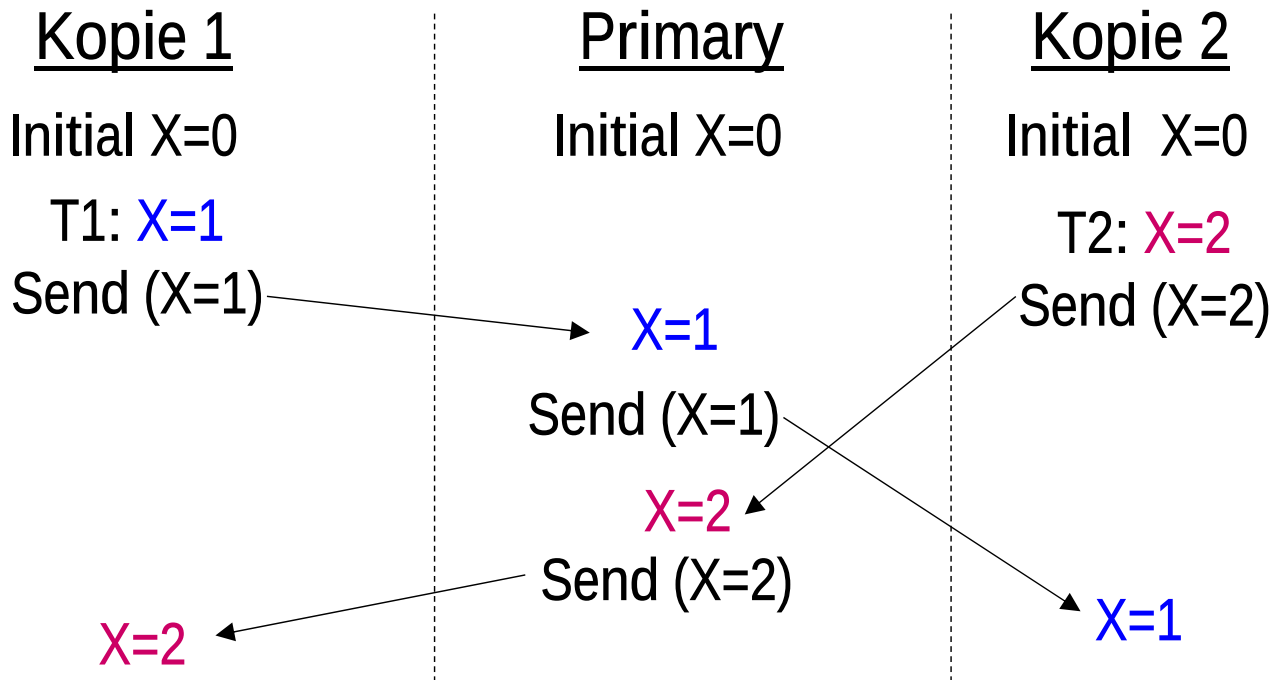
Primärkopien-Verfahren: Fehlerbehandlung

- Netzwerk-Partitionierung: nur in Partition mit Primärkopie können weiterhin Änderungen erfolgen
- Ausfall des Primary-Copy-Rechners verhindert weitere Schreibzugriffe bis zu dessen Recovery
- Alternative: Bestimmung eines neuen Primary-Copy-Rechners, z.B. mit Paxos-Protokoll
  - jede Änderung erfordert, dass Mehrheit der Kopien nicht nur Primärkopie geändert wird
  - auch nach Ausfall der Primärkopie entsteht kein Datenverlust und Verarbeitung kann mit aktuellen Daten fortgesetzt werden

Merge-Replikation

- unsynchronisierte Änderung eines replizierten Objekts an mehreren Knoten (Multi-Master) mit asynchroner Propagierung der Änderungen
- Performance- und Verfügbarkeitsvorteile gegenüber synchroner Aktualisierung
- unabdingbar für mobile DB-Nutzung mit Änderungsbedarf (z.B. Außendienst-Mitarbeiter)
  - Eintragung neuer Kunden / Aufträge (geringe Konfliktgefahr)
- Probleme:
  - nachträgliche Konflikt-Behandlung
  - Mischen paralleler Änderungen (Merge-Replikation)

Beispiel für Änderungskonflikt bei unabhängiger Änderung

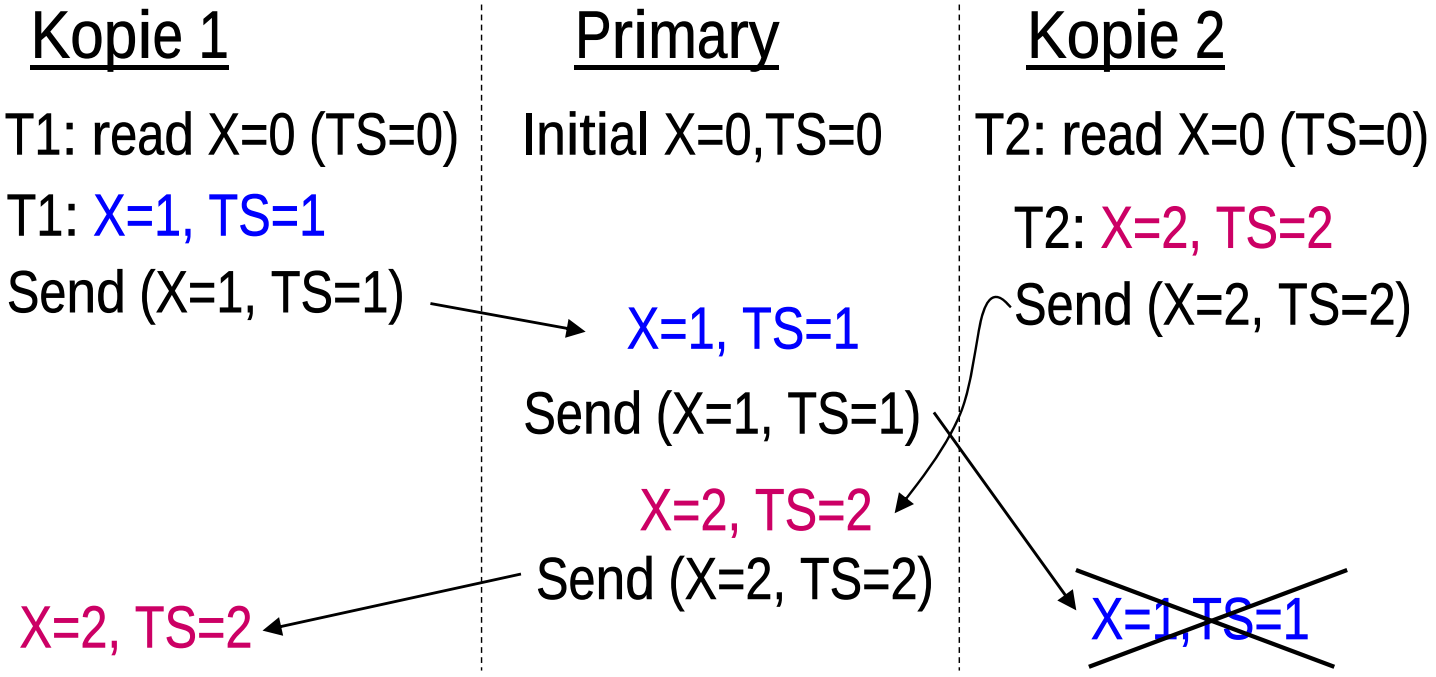


- Kopien enden in unterschiedlichen Zuständen

Merge-Replikation (2)

- Konfliktmöglichkeiten für
  - Update
  - Insert (z. B. Eindeutigkeit)
  - Delete
- Konflikterkennung
  - Objektänderungen enthalten Zeitstempel v der Vorgängerversion und neuen Wert
  - Konflikt: falls lokaler Zeitstempel einer zu aktualisierenden Objektversion abweicht von v
  - Konfliktwahrscheinlichkeit wächst mit Anzahl der änderbaren Replikate und Aktualisierungsverzögerung
- anwendungsspezifische Konflikt-Behandlung (reconciliation)
  - vordefinierte Strategien in ex. DBS: „last timestamp wins“, „site priority“, „average“ ...
  - benutzerdefinierte Konfliktauflösungsroutinen oder manuelle Konfliktbehebung
  - Gefahr von „lost updates“ und anderen Anomalien (keine Serialisierbarkeit)

Ablauf für “last timestamp wins”



Kopien enden in gleichem Zustand, jedoch hat weder T1 noch T2 das Ergebnis der anderen Transaktion gesehen (nicht serialisierbar)

Katastrophen-Recovery

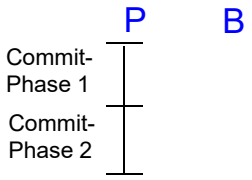
- traditionell: Zurückgehen auf Archivkopie
  - Verlust vieler Transaktionen falls aktuelle Log-Daten seit Erstellung der Archivkopie verloren gehen
  - zeitaufwendiges Einspielen der Archivkopie
- Remote Logging: Übertragung der Log-Daten an entferntes Rechenzentrum
- Alternative für hohe Verfügbarkeit: räumlich entfernte Rechenzentren mit replizierter Datenbank
  - Replikationswartung gemäß Primary-Copy-Verfahren (Primär- und Sekundärkopien)
  - kommerzielle DBS: meist zwei Rechenzentren, davon nur in einem Änderungsbetrieb
  - Web Data (Google, Yahoo, Ebay, Amazon): mehrere (<10) aktive Data Center mit replizierten Daten

Katastrophen-Recovery (2)

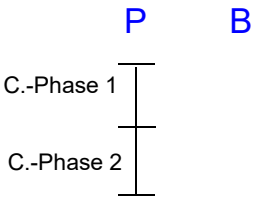
- passives Stand-By-System (Hot Stand-By)
  - Änderungen erfolgen nur im Primärsystem
  - Kopie dient als Stand-By bzw. wird nur für Queries genutzt (Entlastung des Primärsystems)
  - Redo-Log-Daten werden ständig zum Backup-System übertragen und DB-Kopie wird nachgefahren
  - bei asynchroner Übertragung gehen höchstens einige wenige Transaktionen im Kastastrophenfall verloren
  - für "wichtige" Transaktionen: synchrone Übertragung der Log-Daten (Commit-Verzögerung bis entfernte DB aktualisiert ist)

Katastrophen-Recovery: Sicherheitsstufen

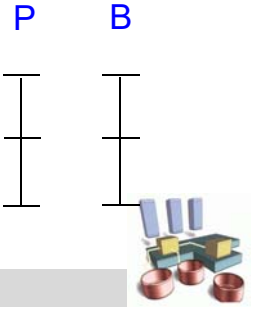
- 1-sicher (1-safe)
  - lokales Commit am Primary (P)
  - asynchrone Übertragung der Änderungen an Backups (B)



- 2-sicher (2-safe)
  - synchrone Aktualisierung der Backups
  - Kommunikation in Commit-Phase 1



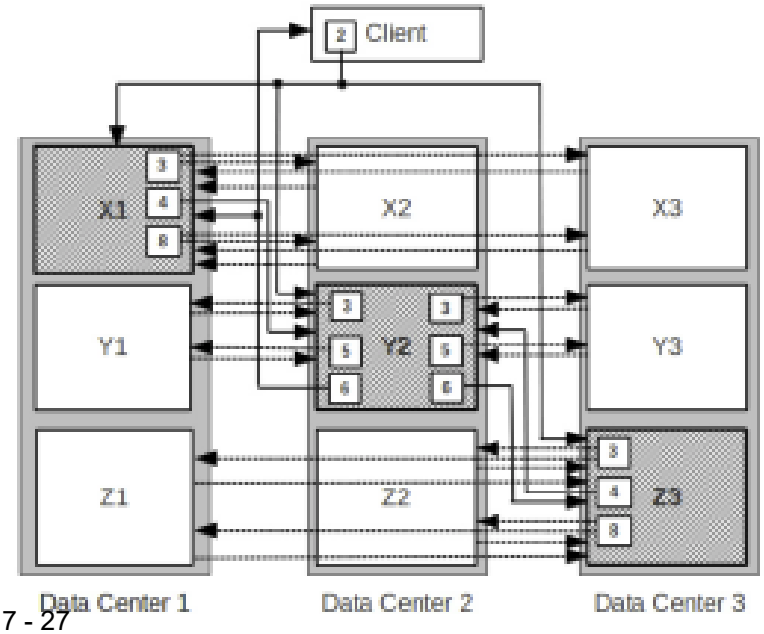
- "sehr sicher"
  - 2PC bzw. Paxos-Verfahren zwischen Primary u. Backups
  - alle Knoten (bzw. Mehrheit) müssen Commit zustimmen
  - nur sinnvoll bei gleichberechtigten Data Centers



Geo-Replikation für Cloud-Daten

- mehrere geographisch verteilte Data Center mit replizierter Datenhaltung (meist 3-5 Replikate pro Objekt)
  - schnelle Zugriffe auf räumlich nahe Replikate
  - Gewährleistug einer sehr hohen Verfügbarkeit gegenüber Ausfällen innerhalb Data Center sowie Katastrophen
- Beisipielansätze: Cassandra, Yahoo Pnuts, Google MegaStore / Spanner
- meist synchrone Replikation („sehr sicher“) für Änderungen, um Datenverlust zu vermeiden
  - Google nutzt dazu Paxos-Verfahren zur konsistenten Aktualisierung der Replikate, eingebettet in 2PC
  - Beispiel für Client-initiiertes 2PC für Änderung von dreifach replizierten Objekten X,Y,Z

(dunkel: Paxos Leader pro Replikatgruppe)



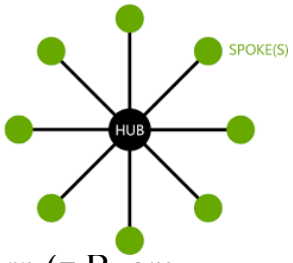
Quelle: D. Agrawal et al.: Managing Geo-replicated Data in Multi-Datacenters. LNCS 7813, 2013

7. Replizierte Datenbanken

- Einführung (Replikationsstrategien)
- Update-Strategien bei strenger Konsistenz
  - ROWA-Verfahren / 2PC
  - Voting-Verfahren
- schwächere Formen der Datenreplikation
  - Refresh-Alternativen
  - Primary-Copy-Verfahren
  - Merge-Replikation
- Katastrophen-Recovery / Geo-Replikation
- Blockchain / distributed ledger
  - Einleitung
  - Kryptowährung Bitcoin

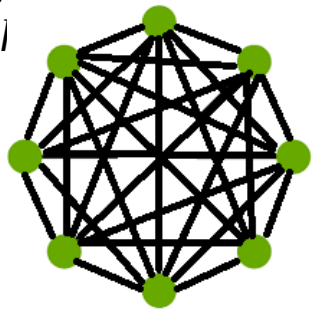
Generelle DB-Lösungen

- Lösung 1: zentralisierte Kontrolle (z.B. Bank) mit kompletter Kontrolle über Zustände und Zustandswechsel



  - Nutzung eines zentralisierten DBS mit Transaktionskontrolle
  - auch bei Einsatz von parallelen/verteilten DBS bleibt zentrale Kontrolle bzw. begrenzte Knotenautonomie (Verteilungstransparenz)
  - Nutzer müssen Unternehmen (z.B. Bank) und anderen Transaktionspartnern (z.B. am Geldverkehr) vertrauen

- Lösung 2: Blockchain-Systeme / verteilte Ledger-Systeme
  (*DLT: Distributed Ledger Tec*



  - „Ledger“ (Buchführungssystem = Datenbank) zur Repräsentation des Zustandes
  - verteilte Kopien mit Append-Only Änderungen
  - globale Identität (Signatur) von Akteuren
  - Transaktion für Zustandsübergänge (Synchronisation gegen Double Spending)

Blockchain: Begriff

- Blockchain = verteiltes System zur Verwaltung von Datensätzen mit dem Ziel, Konsens über den Zustand zu erzielen
- Eigenschaften / Ziele
  - keine zentrale Instanz
  - Teilnehmer …
    - müssen andere Teilnehmer nicht kennen
    - müssen anderen Teilnehmern nicht vertrauen (keine „trusted third parties“)
    - können sich dennoch über einen Zustand einig sein
  - besserer Schutz gegenüber Angriffen / Datenänderungen
- Prinzip
  - Konsens über den initialen Zustand (z.B. leerer Zustand)
  - P2P-Netz aus Teilnehmern (Netzwerkknoten)
  - Transaktionen werden im Netzwerk angezeigt und weitergeleitet
  - Verhindern der Manipulation von Existenz oder Inhalt bereits ausgeführter Transaktionen

Blockchain/DLT-Typen

- öffentliche vs. private Blockchains
- öffentliche Blockchains (z.B. für Kryptowährungen)
  - lassen beliebige Teilnehmer zu
  - maximale Transparenz, gesamter Ledger öffentlich
  - Pseudonymität der Nutzer suggeriert Privacy, aber oft Tracking-Angriffe möglich
  - sehr hoher Ressourcenbedarf
- private Blockchains (z.B. für Unternehmensanwendungen)
  - durch Eigentümer oder Konsortium kontrollierter Teilnehmerkreis (*permissioned* blockchains)
  - effizientere und kostengünstigere Realisierung von Transaktionen
  - typische Anwendung: Prozesse zwischen großen Organisationen abbilden, z.B für Lieferketten (supply chains)
  - Beispielrealisierung: Hyperledger (www.hyperledger.org)

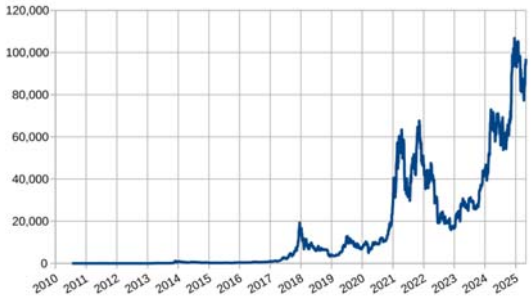
DLT: Anwendungen

- Krypto-Währungen (Bitcoin, Ether etc.)
- viele dezentrale, mit Smart Contracts realisierte, Anwendungen
- E-Voting-Systeme
- virtuelle Organisationen
- Crowdfunding
- Auditing: Aufzeichnung sicherheitskritischer Operationen
  - Zugriff auf bzw. Veränderung von Ressourcen (z.B. Daten, Dokumente)
  - Zugriff auf Gesundheitsakten, …
- dezentrale Energieversorgung und –Abrechnung
- Lieferketten: Dokumentation der Teilschritte

  …
- generell: Regeln eines gemeinsamen Prozesses automatisch Durchsetzen ohne auf zentrale Instanz zu vertrauen

Krypto-Währungen

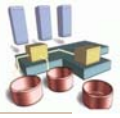
- Historie: Bitcoin
  - 2008: Artikel von „Satoshi Nakamoto" *Bitcoin – A Peer-to-Peer Electronic Cash System* https://bitcoin.org/bitcoin.pdf
  - 2009 Open-Source-Software, eigentlicher Start
  - starke Kursschwankungen
    - all-time high (Dez. 2024): 100 TE pro BTC (bitcoin), März 2023 < 20 TE
  - seit Jan. 2024 auch Börsenfonds (ETFs) in Bitcoin möglich



- aktuell weit über 1000 Währungen
  - fast keine staatliche Anerkennung (Ausnahme: El Salvador)
  - oft geringe Verbreitung / betrügerischer Hintergrund / kurze Lebensdauer
  - Akzeptanz erfordert ausreichendes Vertrauen (durch sich gegenseitig kontrollierende Teilnehmer statt Zentralbank bzw. Staat)

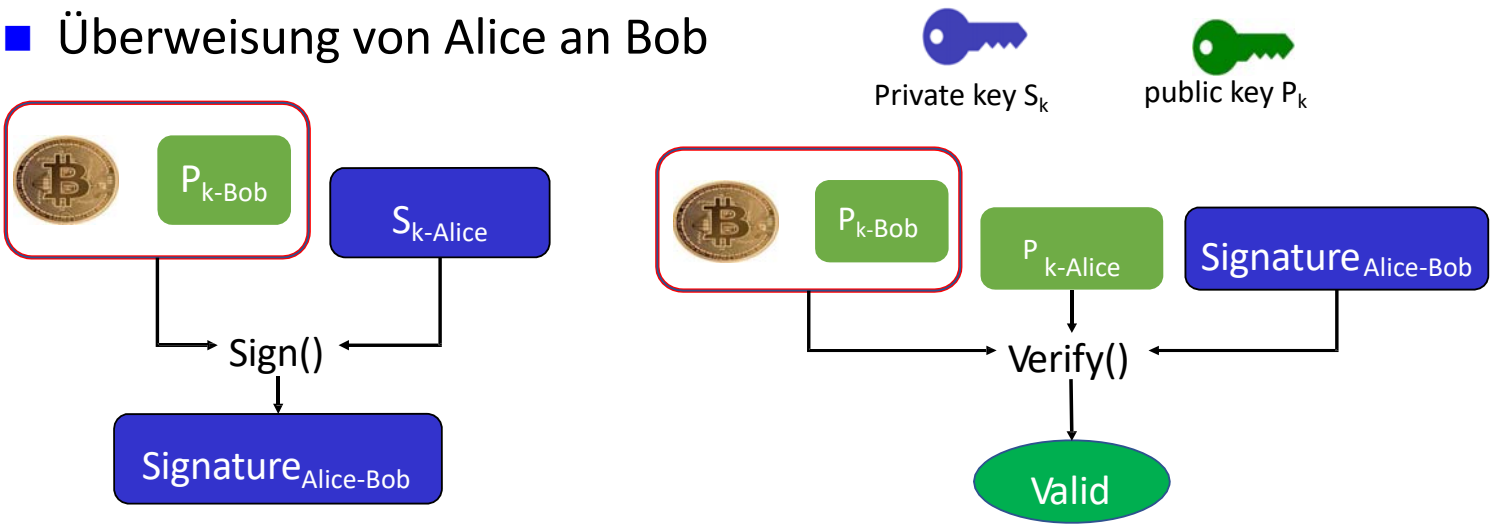
Krypto-Währungen: Bausteine

- Vernetzung der Teilnehmer: P2P-Netz statt zentrale Instanz
  - mehr als 21.000 Bitcoin-Knoten nach https://bitnodes.io (>1.200 in D)
  - größter (Mining-)Pool hat 25% aller Rechner
- kryptographische Signaturen: Public-Key-Kryptosystem
  - öffentlicher Schlüssel = Kontonummer
  - privater Schlüssel = Verfügungsgewalt über Konto
  - Überweisung: Betrag + öffentlicher Schlüssel des Empfängerkontos, signiert mit privatem Schlüssel des Senders (=Transaktion)
  - Überweisung wird im Netz verteilt und kann von allen überprüft werden
- Buchführung: Transaktionen werden im Ledger voll repliziert auf allen Knoten verwaltet
- Bitcoin-Transaktionen
  - keine expliziten Konten: Guthaben = eingegangene Gutschriften, die noch nicht weiter überwiesen wurden

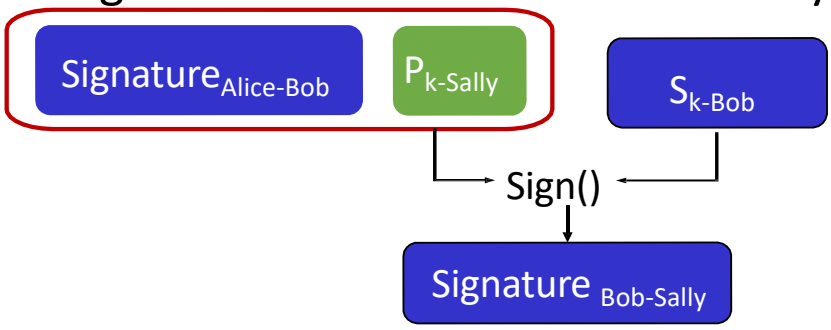


Digitale Signaturen und Bitcoin

- Überweisung von Alice an Bob



- Weitergabe der Bitcoins von Bob an Sally



Hashing H(x)

- Kombination von Signaturen und Public Keys über Hashing
  - Eingabe: String beliebiger Länge
  - Ausgabe fester Länge (z.B. 256 Bits)
  - effizient berechenbar
- Bitcoin nutzt SHA-256 (**S**ecure **H**ash **A**lgorithm)

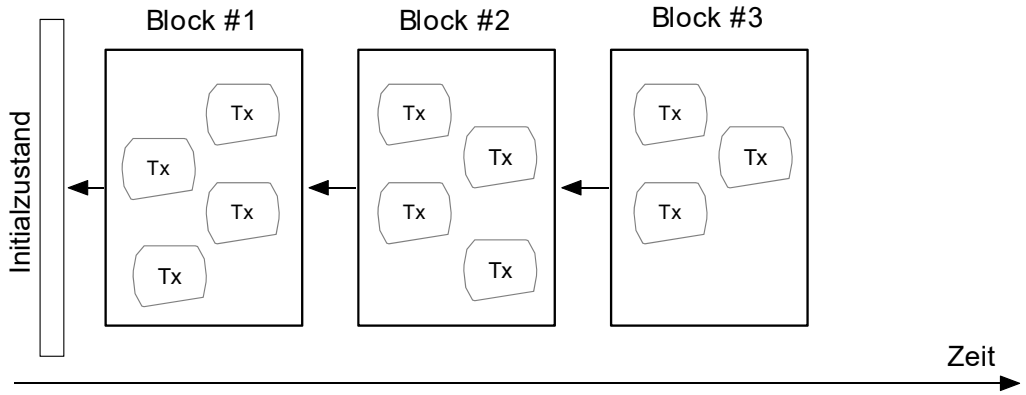
SHA256 ($Signature_{Alice\text{-}Bob}$ || $P_{k\text{-}Sally}$) =

256-Bit (32-Byte) eindeutiger String

- Eigenschaften:
  - *kollisionsfrei*: keine zwei x, y so dass H(x) = H(y)
  - *sicher:* unmöglich x aus H(x) abzuleiten (one-way hash function)

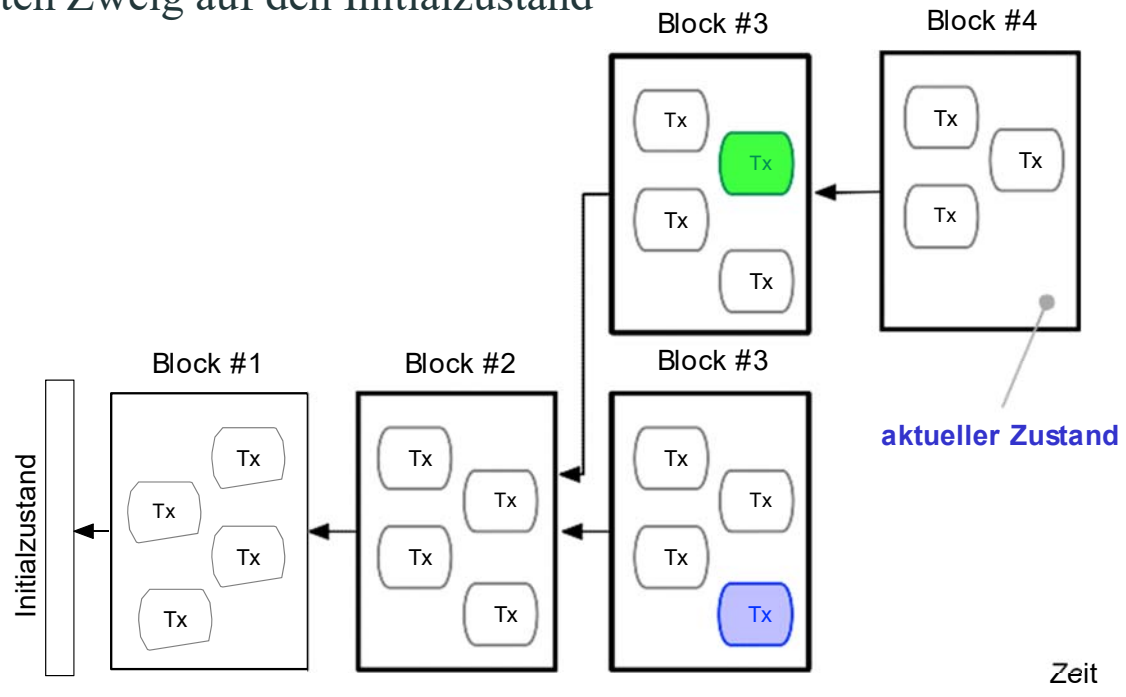
Blockchain-Elemente: Blöcke

- Ansatz:
  - Zusammenfassung von Transaktionen zu Blöcken (Block-)
  - Blöcke werden verkettet (-chain), d.h. ein Block basiert auf seinen Vorgänger
    - Block enthält kryptographisch sicheren Hashwert seines Vorgängerblocks
- Blockinhalt: Transaktionsdaten, Zeitstempel, Hash des Vorgängerblocks (unveränderlich)
- jeder Teilnehmer kann jederzeit neuen Block erstellen



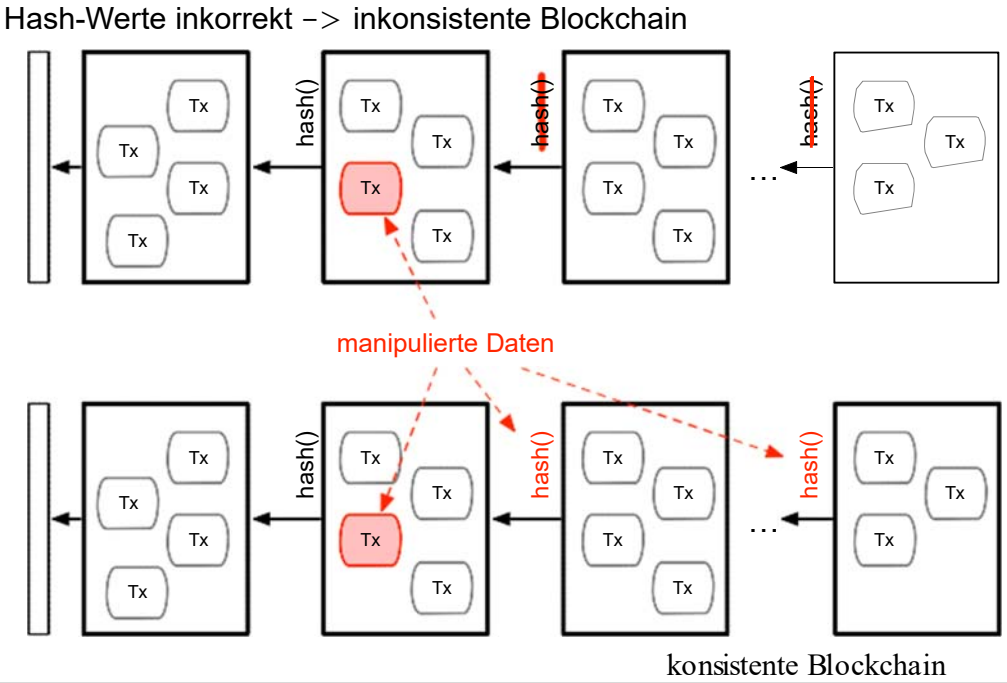
Blockchain-Elemente: Ledger

- Ledger = Blockkette (enthält Transaktions-Log)
  - jeder Knoten im Netzwerk verwaltet eigene Kopie des Ledgers
  - aktueller Zustand = Anwendung aller Transaktionen der Blöcke im längsten Zweig auf den Initialzustand



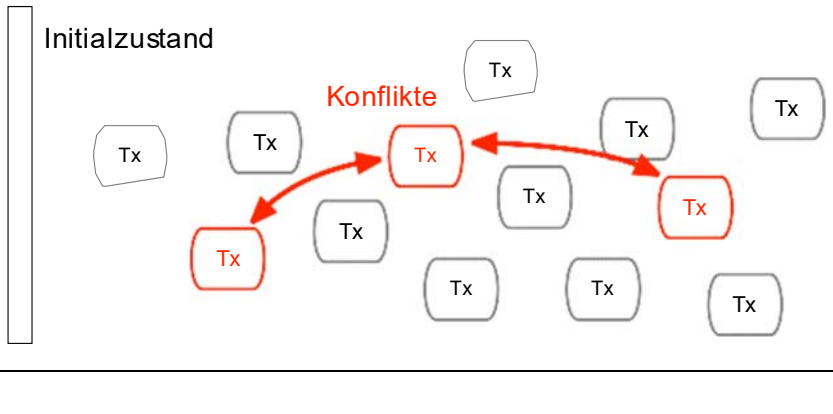
Blockchain: Manipulationssicherheit

- Verkettung der Blöcke durch Hash-Zeiger
  - Manipulation eines Blockinhaltes soll erkannt werden können
  - alleine nicht ausreichend: Ersetzen einer Teilkette durch eine manipulierte Teilkette muss extrem schwer gemacht werden



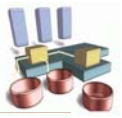
Probleme verteilter Transaktionen

- Konsistenzprobleme
  - (kurzzeitig) unterschiedliche Zustände der Knoten
  - Reihenfolge der Transaktionen
  - Versuche doppelter Ausgaben
  - Konflikte / Abhängigkeiten zwischen Transaktionen
- Notwendigkeit der Konsensusfindung



Mining / Konsens-Findung

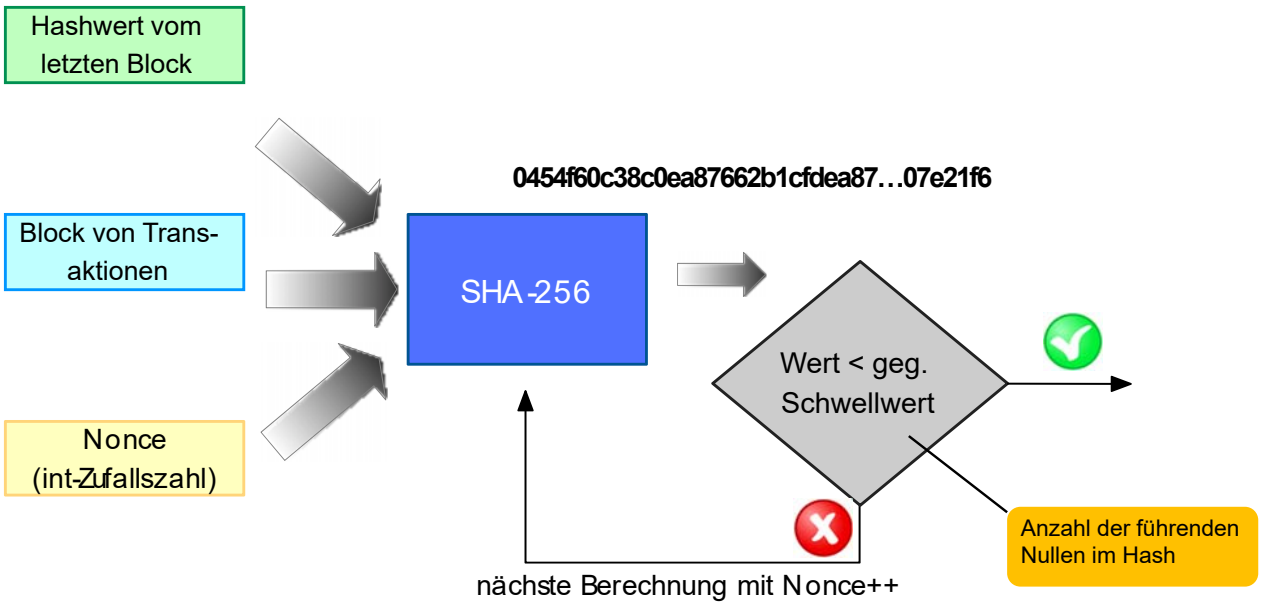
- Miner sind Nodes, die das Konsens-Protokoll ausführen.
  - werden dafür belohnt, da dies Dienstleistung für die Nutzer (Peers) ist
  - Miner empfangen neue Transaktionen von Nutzern und bündeln sie in einem neuen Block. Neue Blöcke werden per Broadcast im Netz verteilt.
  - da Miner parallel (konkurrent) arbeiten, können gleichzeitig verschiedene neue Blöcke im Netz kursieren (temporäre Inkonsistenz)
- Eignung bekannter Konsensverfahren wie Paxos …?
  - keine Behandlung byzantinischer Fehler (bösartiges Verhalten von Teilnehmern/Knoten, z.B. Austausch gefälschter Nachrichten)
  - Knoten müssen bekannt und immer verfügbar sein
- anderer Ansatz notwendig ->Proof of Work (PoW)



Proof of Work

- um einen neuen Block zu signieren, muss eine absichtlich sehr rechenaufwändige Aufgabe gelöst werden.
- Großteil des Netzwerks muss an der längsten Block-Kette mitgearbeitet haben → Peers übernehmen die längste Kette
- PoW Anforderungen:
  - aufwändige Berechnung (nur mit Brute Force) aber einfache und schnelle Validierung
  - muss abhängig vom zu erzeugenden Block sein (Vermeidung von Vorabberechnungsangriffen)
  - variabler Schwierigkeitsgrad
  - Anreizsystem: Mining selbst sollte lohnenswert sein: Belohnungstransaktion (Reward Transaction)
  - Teil des neuen Blocks (*Coinbase* in Bitcoin)

PoW: Hashcash von Bitcoin

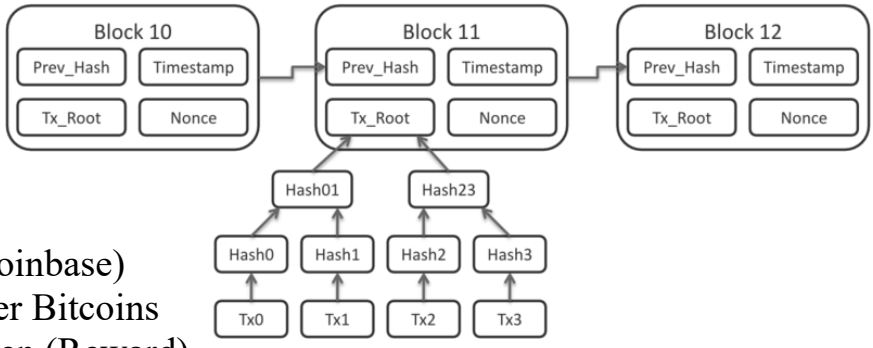


PoW: Ablauf

- wenn Knoten Aufgabe gelöst hat (Mining abgeschlossen):
  - füge Block von Transaktionen der Blockchain hinzu
  - Multicast (Flooding) der Lösung an andere Netzwerkknoten
  - Netzwerkknoten validieren und akzeptieren Lösung
- eingehende Blöcke werden nur akzeptiert, wenn Sie längste Kette korrekt erweitern
- in welchem Zweig der Blockchain sollte ein Miner arbeiten?
  - für Belohnung: Zweig muss Teil des aktuellen Zustands sein (=längster Zweig)
  - keine Koordination notwendig!
  - für von Mehrheit akzeptierte Blöcke erhält Miner geschürfte Bitcoins + Gebühren der enthaltenen Transaktionen

Bitcoin: Transaktionen und Blöcke

- Transaktionen
  - Inhalt: Senderadresse, Empfängeradresse, Betrag, Signatur
  - selbstgewählte Transaktionsgebühren
  - mit privatem Schlüssel des Senders signiert
  - im Netzwerk validiert und verbreitet
- Blöcke
  - feste Größe (z.B. 1 MB)
  - 1. Block = Genesis-Block
  - neue Blöcke durch Mining erzeugt
  - erste Transaktion eines Blocks (coinbase) enthält Überweisung neu erzeugter Bitcoins für Mining + Transaktionsgebühren (Reward)
  - Hashwert = paarweises Hashing der Transaktionen in *Merkle-Baum*, Hashwert des Wurzelknotens (Root-Hash) als Prüfsumme des Blocks



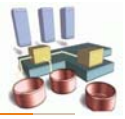
- Mining: PoW wie beschrieben (Nonce-Variation, Flooding)

Netzwerkangriffe: 51%-Angriff

- Mehrheitsangriff: Angreifer kontrolliert über die Hälfte der Rechenleistung des Netzwerks
  - ermöglicht Double Spends, Rückgängigmachen von Transaktionen
  - Prinzip: eigene Blöcke schneller anlegen als der Rest des Netzes und nachträglich gültig machen
- nach Wikipedia:
  - 2014: Mining Pool GHash überschreitet kurzzeitig 50%-Marke
  - Attacken auf Bitcoin Gold (2018) und Ethereum Classic (2019)
- Gegenmaßnahmen
  - 51%-Angriff ist ein auffälliges Ereignis -> z.B. Hard Forks in Bitcoin
    - ersten Block einer verdächtigen Kette ungültig erklären
  - eingebaute Anreizmechanismen:
    - hohe Miningkosten im Falle einer Abwehr verloren
    - Senden anderer Transaktionen kann nicht verhindert werden

Bitcoin: Fakten (Wikipedia)

- Größe Blockchain 665 GB (Juni 2025), Nov. 2020: 310 GB
  - seit April 2024: 3,25 neu erzeugte Bitcoins pro Block; halbiert sich alle 4 Jahre
  - Transaktionskosten: 1.000 Satoshi (= 10 µBTC)
- max. 7 Transaktionen pro Sekunde (schlechte Skalierbarkeit)
  - ca. 10 Minuten pro Block / Transaktion
- extremer und wachsender Ressourcen/Energie-Bedarf
  - Schätzung: 172 Terawattstunden in 2024 (0,5 % des Weltenergiebedarfs)
  - pro Transaktion: 1200 kWh (2021); Kreditkartentransaktion: 1,5 Wh
- energieeffizientere Konsensus-Ansätze existieren
  - z.B. “Proof of Stake” (seit 2022 in Ethereum System)



Zusammenfassung

- Zielkonflikte: Leistung vs. Verfügbarkeit für Leser vs. Schreiber / Konsistenzanforderungen
- Synchronisationsansätze: ROWA, Voting, Primary Copy
- Probleme bei 1-Kopien-Serialisierbarkeit
  - geringe Skalierbarkeit für synchrone Aktualisierung
- kommerzielle DBS unterstützen unterschiedl.Replikationsmodelle
  - Publish/Subscribe-Ansätze , Primärkopien vs. „Update Anywhere“
  - synchrone oder verzögerte Übertragung von Änderungen
  - parallele Änderungen mit Nutzerkontrolle (Merge-Replikation)
- Cloud Data: Geo-Replikation über mehrere Data Center
  - „billige“ Rechner mit hoher Ausfallwahrscheinlichkeit erfordern sichere Protokolle gegen Datenverlust, z.B. mit 2PC und Paxos
- Blockchain / Bitcoin
  - voll replizierte Datenbank ohne zentralisierte Kontrolle / Institution
  - aufwändige Konsensbildung durch Mining (Proof of Work)
  - private Blockchains für Unternehmensanwendungen mit geringerem Ressourcenbedarf