Verteilte Systeme

Oktober - November 2023

6. Vorlesung – 08.11.2023

Kurs: TINF21AI1

Dozent: Tobias Schmitt, M.Eng.

Kontakt: d228143@
student.dhbw-mannheim.de

Wiederholungsfragen

- Was unterscheidet die logische Uhr von Lamport von der Vektoruhr?
- Was verstehen Sie unter einem vollständig geordnetem Multicasting?
- Wie lässt sich gegenseitiger Ausschluss in verteilten Systemen realisieren?
- Welchem Zweck dienen Wahlalgorithmen?
- Beschreiben Sie einen möglichen Wahlalgorithmus.

Themenüberblick

- **Konsistenz und Replikation**
  - **Datenzentrierte Konsistenzmodelle**
  - Clientzentrierte Konsistenzmodelle
  - Replikationsverwaltung
  - Konsistenzprotokolle

Konsistenz und Replikation

- Welche Gründe kann es für Replikation geben?
- Was kann gegen das Replizieren von Daten sprechen?
- Was bedeutet Konsistenz? Welche Vorstellung verbinden Sie damit?
- Ist Konsistenz messbar?
- Was könnten Sie sich Vorstellen, was datenzentrierte und clientzentrierte Konsistenz bedeutet?

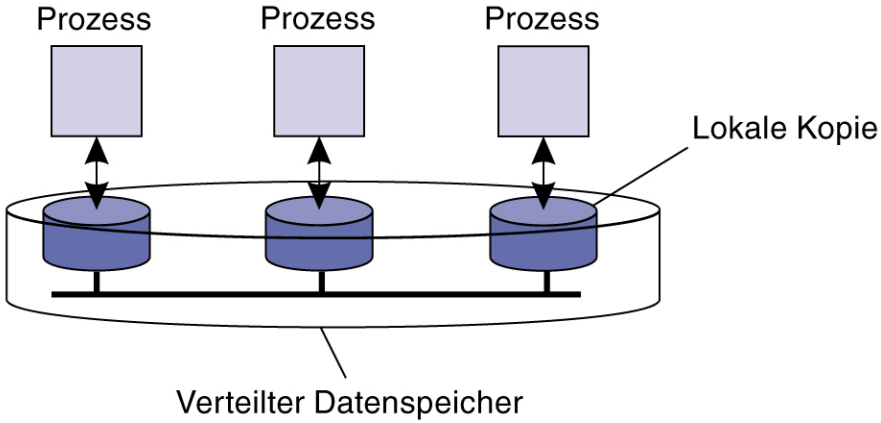
Konsistenz und Replikation

- Gründe für Replikation
  - Verlässlichkeit
    - Im Fehlerfall – Umschalten auf anderes Replikat
    - Schutz vor beschädigten Daten
  - Systemleistung
    - Zahlenmäßige Skalierung → Zugriff von mehr Prozessen
    - Geografische Skalierung → Verkürzung der Datenzugriffszeit

- Gegenindikationen
  - Replikate aktuell zu halten bedarf Bandbreite
  - Konsistenzprobleme

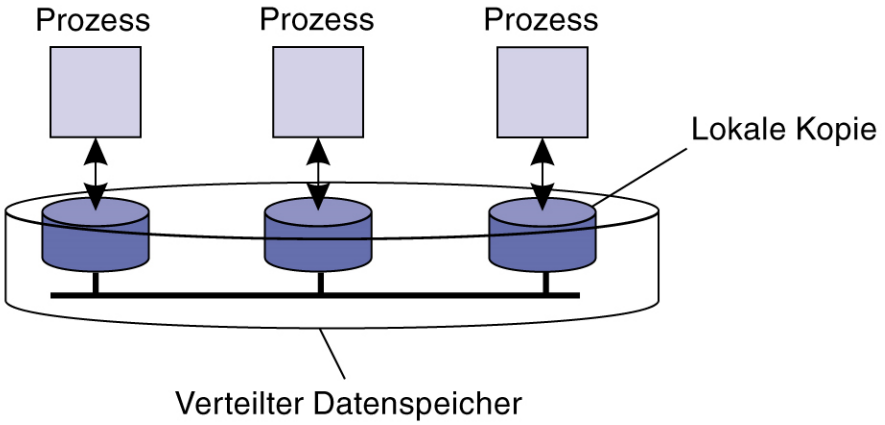
Datenzentrierte Konsistenzmodelle

- Modell des logischen Datenspeichers
- Konsistenzmodell
  - Vertrag zwischen Prozessen und Datenspeicher
  - Korrektheit des Speichers, solange Einhaltung gewisser Regeln
  - Datenspeicher spezifiziert genau, was das Ergebnis von Lese- und Schreiboperationen im Rahmen der parallelen Ausführung sein wird



Datenzentrierte Konsistenzmodelle

- Modell des logischen Datenspeichers
- Idee: Leseoperation liefert Resultat des letzten Schreibvorganges
- Problem: Festlegung schwierig bei Fehlen einer globalen Uhr
  - Modelle schränken Rückgabemöglichkeiten von Leseoperationen ein
- Aber: Je stärker die Konsistenzanforderung, desto höher der Aufwand für die Aktualisierung



Datenzentrierte Konsistenzmodelle

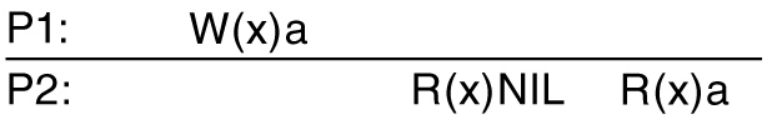
- Definition von Inkonsistenzen im Rahmen stufenloser Konsistenz:
- Abweichung von Replikaten hinsichtlich
  - numerischer Werte
    - z.B. Aktienkurse → Definition einer absoluten oder relativen numerischen Abweichung
    - Oder auch die Anzahl auf ein Replikat angewendete Aktualisierungen
  - Veralterungsgrad
    - Bezug auf den Zeitpunkt der letzten Aktualisierung
    - z.B. Daten vom Wetterbericht veralten nicht innerhalb weniger Stunden
  - Reihenfolge von Aktualisierungsoperationen
    - z.B. nur vorläufige Aktualisierung bis Zustimmung hinsichtlich der Reihenfolge aller Replikate → ggf. Zurücknahme der Aktualisierungen

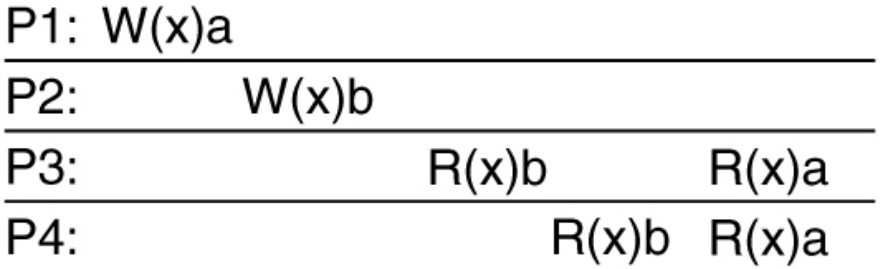
Datenzentrierte Konsistenzmodelle

- Messung von Inkonsistenzen im Rahmen stufenloser Konsistenz
  - Definition einer Konsistenzeinheit (Consistency Unit) – kurz „Conit“
  - Beispiele für fein- und grobkörnige Conits:
    - Aufzeichnung hinsichtlich einer Aktie
    - Einzelner Wetterbericht
    - Komplette Datenbank
  - Conit-Größe
    - Kleinere Conits → mehr Verwaltung
    - Größere Conits → schneller im Inkonsistenz-Zustand
  - Wunsch: Klare Angabe der Konsistenzanforderung

Konsistente Anordnung von Operationen

- Sequentielle Konsistenz
  - Das Ergebnis jeder Ausführung ist dasselbe, als wären die Operationen von allen Prozessen in einer sequentiellen Reihenfolge ausgeführt worden und die Operationen jedes einzelnen Prozesses erscheinen in dieser Sequenz in der von seinem Programm vorgegebenen Reihenfolge.
  - Alle Prozesse beobachten dieselbe Verzahnung und jede Verzahnung von Operationen ist gültig, solange alle Prozesse die selbe Verzahnung sehen.
  - Hinweis: Keine Aussage über die Zeit.
  - Beispiele:





Erklärung:
P1: W(x)a → Prozess 1 schreibt in die Variable x den Wert a.
P2: R(x)b → Prozess 2 liest die Variable x und erhält den Wert b.

Konsistente Anordnung von Operationen

- Sequentielle Konsistenz
  - Weiteres Beispiel:
    - Operationen von 3 gleichzeitig ausgeführten Prozessen (Hinweis: print(a,b) meint Ausgabe von a und b.)

Prozess P1	Prozess P2	Prozess P3
x ← 1;	y ← 1;	z ← 1;
print(y, z);	print(x, z);	print(x, y);

    - Ausführung ergibt 6-Bit-Zeichenkette
    - Nicht alle 6-Bit-Zeichenketten sind im Rahmen der sequenziellen Konsistenz erlaubt! … siehe folgende Beispiele:

x ← 1;	x ← 1;	y ← 1;	y ← 1;
print(y, z);	y ← 1;	z ← 1;	x ← 1;
y ← 1;	print(x, z);	print(x, y);	z ← 1;
print(x, z);	print(y, z);	print(x, z);	print(x, z);
z ← 1;	z ← 1;	x ← 1;	print(y, z);
print(x, y);	print(x, y);	print(y, z);	print(x, y);
Prints: 001011	Prints: 101011	Prints: 010111	Prints: 111111

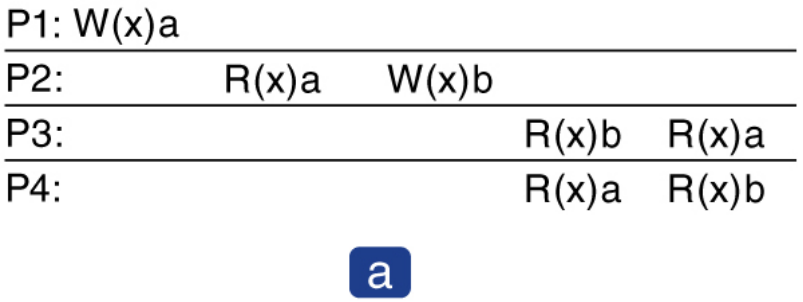
Konsistente Anordnung von Operationen

- Kausale Konsistenz
  - Schreibvorgänge, die möglicherweise in kausaler Beziehung zueinander stehen, müssen von allen Prozessen in derselben Reihenfolge wahrgenommen werden. Parallele Schreibvorgänge können auf unterschiedlichen Rechnern in unterschiedlicher Reihenfolge gesehen werden.
  - Hinweis: Tritt eine Leseoperation gefolgt von einer Schreiboperation ein, dann sind die Ereignisse potentiell kausal verknüpft.
  - Kausale Konsistenz ist schwächer als sequentielle Konsistenz.
- Beispiel:

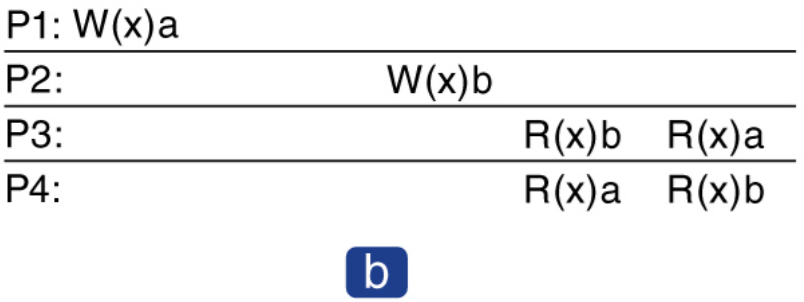
P1:	W(x)a			W(x)c		
P2:		R(x)a	W(x)b			
P3:		R(x)a			R(x)c	R(x)b
P4:		R(x)a			R(x)b	R(x)c

Konsistente Anordnung von Operationen

- Kausale Konsistenz
  - Frage: Welcher der folgenden Abläufe ist aus Sicht der kausalen Konsistenz falsch?



a

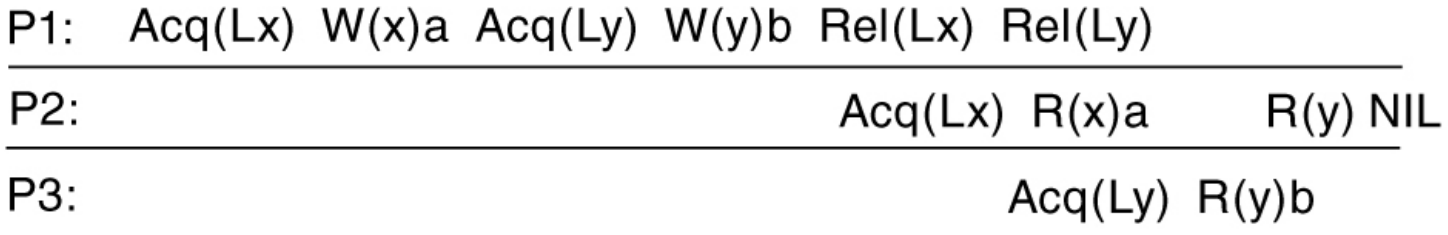


b

Konsistente Anordnung von Operationen

- Gruppieren von Operationen
  - Arbeit mit Synchronisationsvariablen für wechselseitigen Ausschluss
  - Grund: Abfolge von Lese- und Schreiboperationen wird atomar
  - Synchronisationsvariable steht für bestimmen Satz an Daten (einzelne Datenelemente bis hin zur Gesamtheit der gemeinsam genutzten Daten)

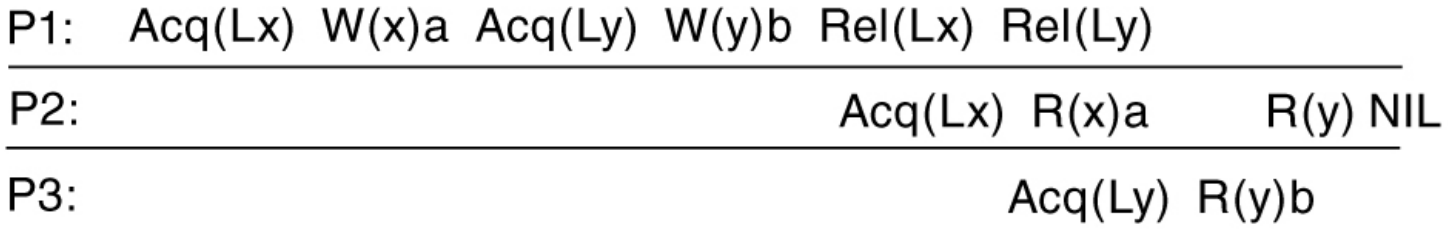
Beispiel:



Konsistente Anordnung von Operationen

- Gruppieren von Operationen
  - Besitzer der Synch.-Variablen: Letzter Nutzer
  - Aneignung über Nachrichten an Besitzer
  - Kriterien an die Synch.-Variablen
    - Zugriff auf Synch.Variablen muss sequentiell konsistent sein
    - Kein Zugriff auf Synch.Variablen ist erlaubt, bis alle vorangegangen Aktualisierungen ausgeführt sind
    - Kein Daten-Zugriff ist erlaubt, bis alle vorangegangenen Zugriffe durch Synch.Variablen erfolgt sind

Beispiel:



Themenüberblick

- **Konsistenz und Replikation**
  - Datenzentrierte Konsistenzmodelle
  - **Clientzentrierte Konsistenzmodelle**
  - Replikationsverwaltung
  - Konsistenzprotokolle

Clientzentrierte Konsistenzmodelle

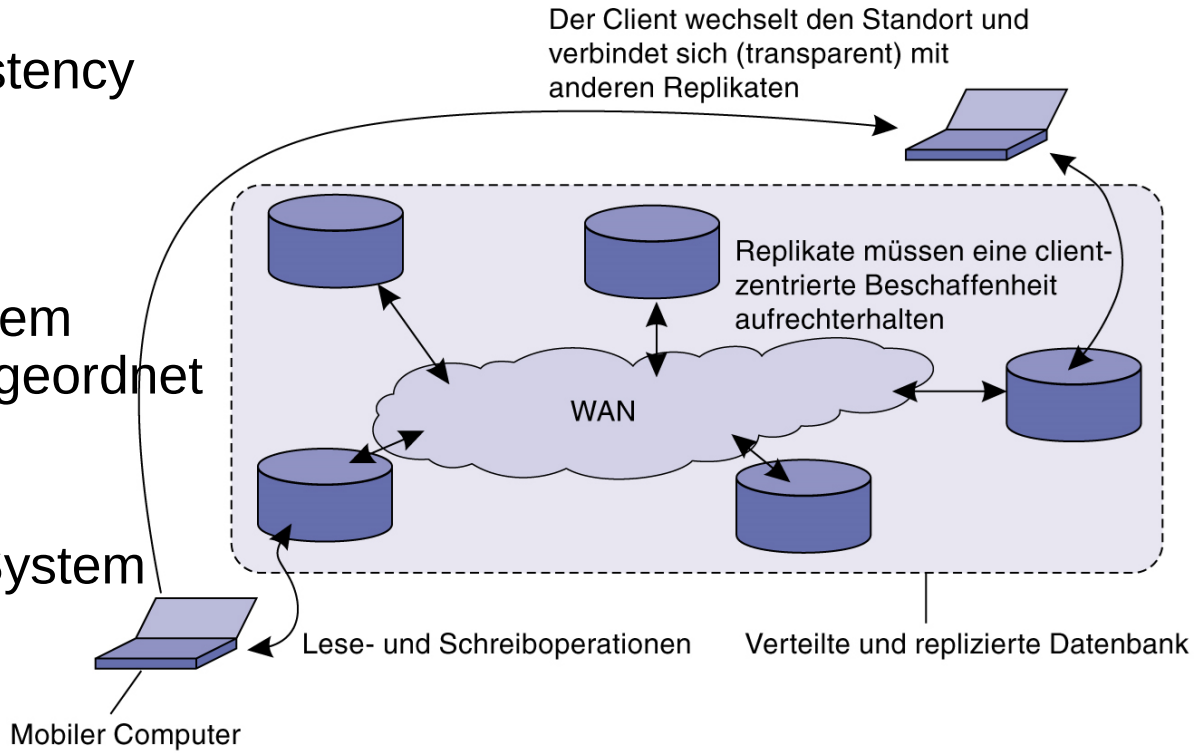
- Eventualy Consistency („Letzendliche“ Konsistenz)
  - Idee:
    - Systeme können recht hohes Maß an Inkonsistenz tollerieren
    - Aber: Im Falle keiner Aktualisierungen – nach einer gewissen Zeit sind alle Replikate konsistent
  - Beispiel:
    - Bennenungssystem DNS – Aktualisierung nur durch jeweiligen Verantwortlichen für einen Teil des Namensraums
    - World Wide Web – Aktualisierungen von Webmaster, aber Browser und Webproxys arbeiten mit Caches

Clientzentrierte Konsistenzmodelle

- Eventualy Consistency („Letzendliche“ Konsistenz)
  - Probleme:
    - Schreib-Schreib-Konflikte – ggf. Einschränkung auf kleinen Gruppe von Prozessen
    - Lese-Schreib-Konflikte – Behebbar solange Zugriff auf gleiche Replikate

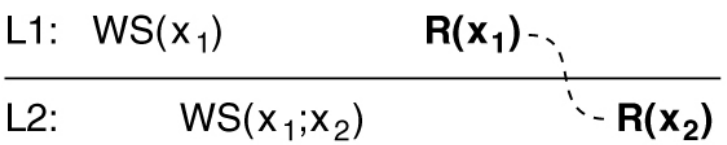
Clientzentrierte Konsistenzmodelle

- Problem des mobilen Clients
  - Zugriff auf verteilten Datenspeicher von verschiedenen Standpunkten
  - Konsistenz innerhalb des verteilten Datenspeichers nicht wichtig
  - Konsistenz aus Sicht des Clients wichtig
  - Beispiel für Eventual Consistency
  - Annahme für folgende Betrachtungen:
    - Datenelemente sind einem Prozess als Besitzer zugeordnet
  - Anwendungsbeispiel:
    - Kalender im verteilten System

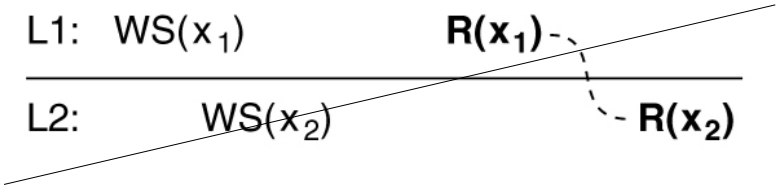


Clientzentrierte Konsistenzmodelle

- Realisierungsmöglichkeiten
  - Monotones Lesen
  - Monotones Schreiben
  - „Read Your Writes“-Konsistenz
  - „Writes Follow Reads“-Konsistenz
- Monotones Lesen:
  - Wenn ein Prozess eines Datenelementes *x* liest, gibt jede anschließende Leseoperation dieses Prozesses auf *x* stets denselben oder einen aktualisierten Wert zurück.
  - Hinweis: Niemals die Rückgabe eines älteren Wertes
  - Beispiel: E-Mail-Postfach
  - Illustrierung:



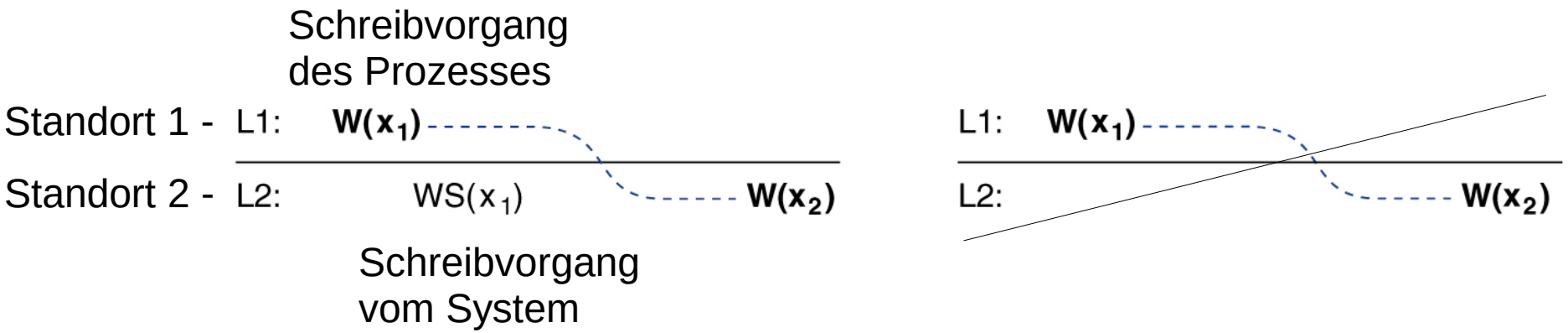
a



b

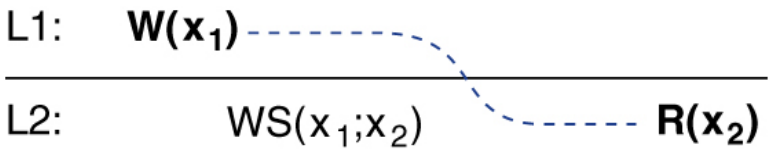
Clientzentrierte Konsistenzmodelle

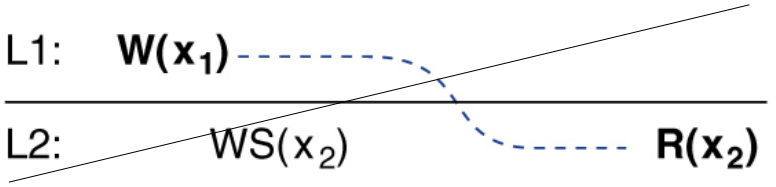
- Monotones Schreiben
  - Eine Schreiboperation eines Prozesses an einem Datenelement *x* wird abgeschlossen, bevor eine folgende Schreiboperation auf *x* durch denselben Prozess erfolgen kann.
  - Illustrierung:



Clientzentrierte Konsistenzmodelle

- „Read Your Writes“-Konsistenz
  - Die Folge einer Schreiboperation eines Prozesses auf das Datenelement *x* wird für eine anschließende Leseoperation auf *x* durch denselben Prozess stets sichtbar sein.
  - Illustrierung:

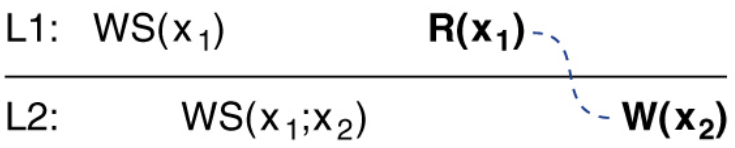


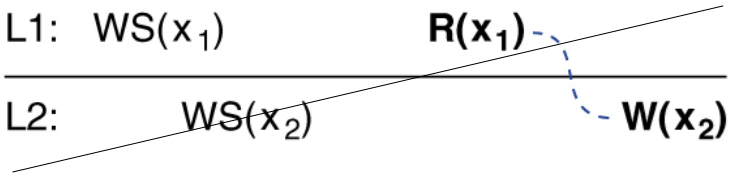


  - Problemstellung:
    - Änderung eines Javascript-Dokumentes einer Webseite direkt auf dem Server.
    - Aufruf der Webseite über Browser
  - Frage: Liegt für die Problemstellung eine „Read Your Writes“-Konsistenz vor?

Clientzentrierte Konsistenzmodelle

- „Writes Follow Reads“-Konsistenz
  - Eine Schreiboperation eines Prozesses auf ein Datenelement x, die auf eine vorherige Leseoperation auf x durch denselben Prozess folgt, wird garantiert, dass sie auf demselben oder einem aktuelleren Wert von x stattfindet.
  - Beispiel: Newsgroup-Posting
    - Reaktion auf einen Artikel erst, wenn Orginalartikel gesehen wurde
  - Illustrierung:





Themenüberblick

- **Konsistenz und Replikation**
  - Datenzentrierte Konsistenzmodelle
  - Clientzentrierte Konsistenzmodelle
  - **Replikationsverwaltung**
  - Konsistenzprotokolle

Replikationsverwaltung

- Um welche Kernfragen könnte sich bei der Replikationsverwaltung handeln?

- Nach welchen Kriterien könnte man Replikate verteilen / anlegen?

- Warum könnte man eine Unterscheidung zwischen serverinitiierter und clientinitiierter Replikation vornehmen? Was bedeutet das?

- Wer initiiert die Aktualisierung eines Replikates?

- Wie werden Inhalte / Aktualisierungen verteilt? Welche Arten sind hier denkbar?

Replikationsverwaltung

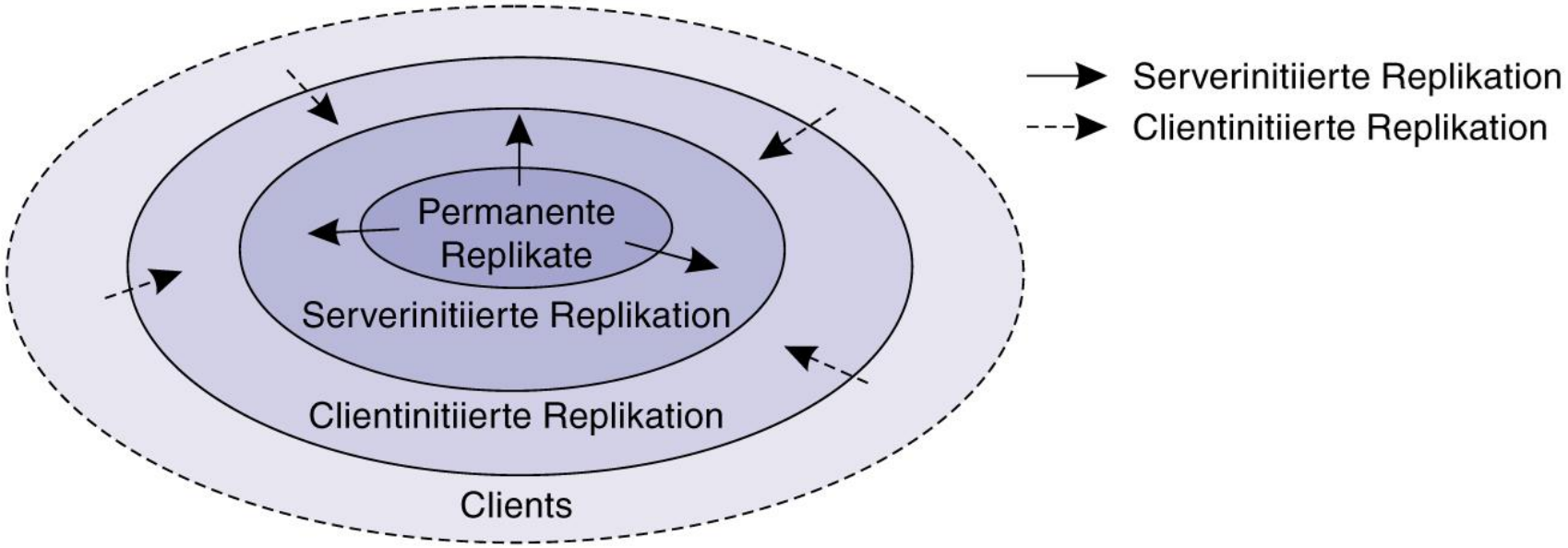
- Kernfragen:
  - Wo, wann und von wem sollen Replikate platziert werden?
- Teilaufgaben:
  - Platzierung von Replikatservern
  - Platzierung von Inhalten

Replikationsverwaltung

- Platzierung von Replikatservern
  - Achtung: Häufig verwaltungstechnische und kommerzielle Frage als Optimierungsproblem
  - Betrachtungsmöglichkeiten:
    - Geografisch (clientorientiert) – Berücksichtigung der räumlichen Entfernung
    - Topologisch – Betrachtung autonomer Systeme (System mit gleichem Routing-Protokoll; Teilnetze)
    - Nutzungsorientiert – Einteilung in Regionen mit Knoten, die auf dieselben Inhalte zurückgreifen

Replikationsverwaltung

- Replikation und Platzierung von Inhalten
  - Unterscheidung
    - Permanente Replikate
    - Serverinitiierte Replikation
    - Clientinitiierte Replikation



Replikationsverwaltung

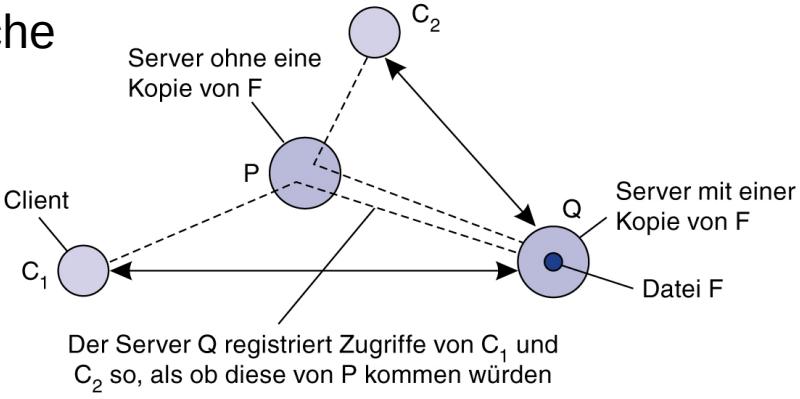
- Permanente Replikate
  - Statische Methode
  - Unterscheidungen
    - Replikate an eine Standort
      - Weiterleitung der Anforderungen nach einer Round-Robin-Strategie
    - Replikate geografische verteilt
      - alias Spiegelung (Mirroring)
      - Beispiel: Clients können aus Liste Mirror-Webseite wählen

Replikationsverwaltung

- Serverinitiierte Replikation
  - Erzeugung temporärer Replikate in Regionen aus denen Anforderungen kommen
    - z.B. Ticketverkauf eines K-Pop-Konzertes in Dtl. über koreanische Webseite
  - Einrichtung auf Initiative des Datenspeichers
  - Grund: Leistungsverbesserung
  - Annahme: Wissen über alle Server Q_i des Webhosting-Dienstes
  - **Frage:** Wie kann man bestimmen, auf welchem Server Q_i die Daten (z.B. einer Webseite) am besten aufgehoben sind?

Replikationsverwaltung

- Serverinitiierte Replikation
  - **Frage:** Wie kann man bestimmen, auf welchem Server Q_i die Daten (z.B. einer Webseite) am besten aufgehoben sind?
  - Lösungsansatz:
    - Ansatz: Überwachung der Zugriffszahlen
    - Zugriffszählung N_{ik} eine Datei F auf Q_i, welche über einen „nächstgelegenen“ Server Q_k hätte laufen können
    - Zugriffszählung N_{ik} > Replikationsgrenzwert
      - Replikation empfehlenswert
    - Zugriffszählung N_{ii} < Löschgrenzwert
      - Löschung, solange noch ein weiteres Replikat
    - Zugriffszählung N_{ii} > 2 * Summe über alle N_{kk} außer Server Q_i
      - Ggf. Migrationsempfehlung



Replikationsverwaltung

- Clientinitiierte Replikation
  - Bezeichnung: (clientseitiger) Cache
  - Hintergrund: Großteil der Zugriffe ist lesend
  - Cache auf selben Computer oder im selben LAN (größer angelegte Caches auch möglich)
  - Vorhaltung:
    - Auf bestimmte Zeit
    - Berücksichtigung von Cache-Treffer (Cache-Hits)
  - Datenspeicher i.A. nicht verantwortlich Daten im Cache konsistent zu halten

Replikationsverwaltung
Verteilen von Inhalten

- Ziel: Betrachtung hinsichtlich der Weiterleitung von (aktualisierten) Inhalten
- Frage: Was soll weitergeleitet werden?
  - Weiterleiten nur einer Benachrichtigung über eine Aktualisierung
  - Übertragen der Daten von einer Kopie zur nächsten
  - Weiterleiten der Aktualisierungsoperationen an andere Kopien

- Erläuterungen zur Benachrichtigungsweiterleitung
  - Benachrichtigung über Invalidierungsprotokolle
    - Nutzung einer geringen Netzwerkbandbreite
    - Nur Infos über Ungültigkeit von Daten
    - Anwendbarkeit: Wenig Leseoperationen im Vergleich zu Aktualisierungen

Replikationsverwaltung
Verteilen von Inhalten

- Erläuterung zur Übertragung der Daten
  - Anwendbarkeit, wenn viel mehr Leseoperationen als Aktualisierungen
  - Bündelungen möglich, zwecks Einsparung von Bandbreite

- Erläuterung zur Weiterleitung der Aktualisierungoperationen
  - Bezeichnung: aktive Replikation
  - Ziel: Einsparung von Bandbreite
  - Achtung: Aktualisierungsoperationen können unterschiedlich komplex ausfallen → ggf. mehr Rechenleistung beim Replikat nötig

Replikationsverwaltung
Verteilen von Inhalten

- Frage: Aktualisierung über Pull und Push?

- Push-basierter Ansatz = serverbasierte Protokolle
  - Einsatz meist bei dauerhaften oder serverinitiierten Replikaten
  - Ziel: hoher Konsistenzgrad
- Pull-basierter Ansatz = clientbasierte Protokolle
  - Einsatz z.B. bei Webcaches
  - Effizient, wenn Verhältnis von Lesezugriffen zu Aktualisierungen gering ist
  - Einsatz von Polling

Replikationsverwaltung
Verteilen von Inhalten

- Vergleich zwischen Push-basierten und Pull-basierten Ansätzen
  - Annahme: 1 Server und mehrere Clients

Thema	Push-basiert	Pull-basiert
Zustand auf dem Server	Auflistung der Client-Replikate und Clientcaches	Keine
Gesendete Nachrichten	Aktualisieren (sowie später möglicherweise Abrufen der Aktualisierung)	Ständiges Abfragen und Aktualisieren
Antwortzeit für den Client	Unmittelbar (oder Zeitaufwand für den Abruf der Aktualisierung)	Zeitaufwand für den Abruf der Aktualisierung

Replikationsverwaltung
Verteilen von Inhalten

- Hybride Form der Aktualisierungsweiterleitung
  - Basis: Leases (Verleihen)
  - Eine Lease ist eine Zusage eines Servers, dass er dem Client eine Zeit lang Aktualisierungen bereitstellt. Danach muss der Client beim Server Aktualisierungen abfragen.
  - Oder kurz: Push-Methode vom Server, wenn Lease vorhanden, ansonsten Pull-Methode vom Client
  - Lease-Laufzeit-Kriterien
    - Alter: längere Leases werden gewährt, wenn Objekt sich über längeren Zeitraum sich nicht verändert hat
    - Zugriffshäufigkeit: Je häufiger die Zugriffe, desto länger die Leases.
    - Zustand und Speicherplatz: Wenn Server überlastet, Verkürzung der Laufzeit.
      - Resultat: Weniger Clients zum Überwachen
      - Effizients abhänig von der Anzahl der Aktualisierungen

Replikationsverwaltung
Verteilen von Inhalten

- Fragestellung hinsichtlich Verwendung von Unicast und Multicast

- Unicast eher bei Pull-basiertem Ansatz
  - Da ein Client bei dem Server anfragt.
- Multicast eher bei Push-basiertem Ansatz

- Hinweis: Multicast häufig billiger.
  - Auf LAN-Ebene sogar Hardwareunterstützung

Themenüberblick

- **Konsistenz und Replikation**
  - Datenzentrierte Konsistenzmodelle
  - Clientzentrierte Konsistenzmodelle
  - Replikationsverwaltung
  - **Konsistenzprotokolle**

Konsistenzprotokolle

- Beschreibung der Implementierung eines Konsistenzmodells
- Arten von Konsistenzprotokollen
  - Urbildbasierte Protokolle
    - Existenz eines ausgezeichneten Replikates als Startpunkt für Aktualisierungen
  - Nicht-Urbildbasierte Protokolle
    - Aktualisierungen können bei beliebigen Replikaten beginnen

Konsistenzprotokolle

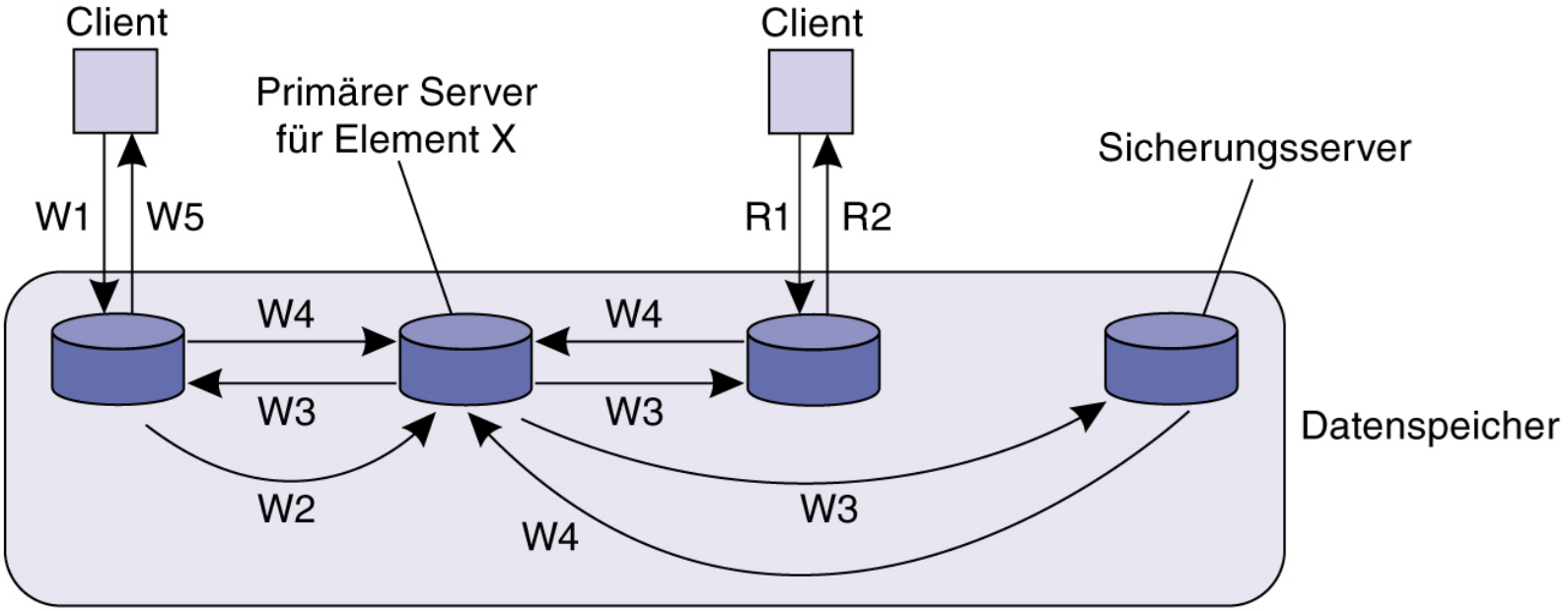
Urbildbasierte Protokolle (Primary-Based Protocols)

- Protokolle für entfernte Schreibvorgänge
  - Verfahren:
    - Datenelemente haben für sie verantwortliche Server
    - Schreiboperation an Datenelement *x* wird an entsprechenden primären Server für *x* weitergeleitet
    - Primärer Server führt Schreiboperation aus und leitet diese anschließend Backupserver weiter
    - Nach Rückmeldung von den Backupservern → Rückmeldung an den Ausgangsprozess
  - Realisierung als sperrende Operation:
    - Vorteil: Client-Prozess weiß, dass Aktualisierungsprozess von mehreren Servern abgesichert ist
    - Nachteil: Aktualisierung braucht mehr Zeit und ggf. Leistungsproblem wegen zu langer Wartzeiten

Konsistenzprotokolle

Urbildbasierte Protokolle (Primary-Based Protocols)

- Protokolle für entfernte Schreibvorgänge



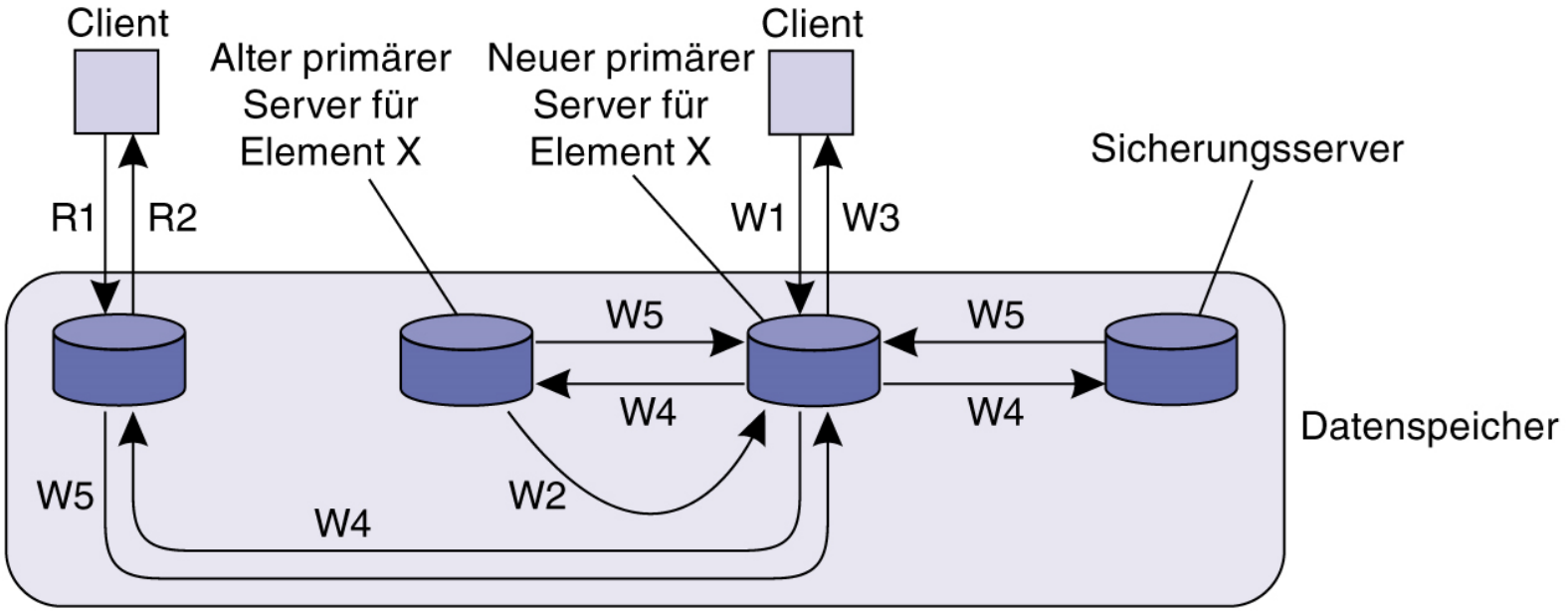
W1 – Schreibanforderung
W2 – Weiterleitung der Anforderung an den primären Server
W3 – Anweisung an die Sicherungen zur Aktualisierung
W4 – Aktualisierungsbestätigung
W5 – Bestätigung des abgeschlossenen Schreibvorgangs

R1 – Leseanforderung
R2 – Leserückmeldung

Konsistenzprotokolle

Urbildbasierte Protokolle (Primary-Based Protocols)

- Protokolle für lokale Schreibvorgänge



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

Konsistenzprotokolle

Urbildbasierte Protokolle (Primary-Based Protocols)

- Protokolle für lokale Schreibvorgänge
  - Idee: primäre Kopie migriert zwischen Prozessen
  - Vorteil: mehrere Schreiboperationen leicht hintereinander ausführbar
  - Achtung: Leistungsgewinn nur bei Einsatz von nicht sperrenden Protokollen

Konsistenzprotokolle
Protokolle für replizierte Schreibvorgänge

- Idee: Schreiboperationen auf mehreren Replikaten möglich (Gegensatz zu Urbildbasierten Protokollen)
- Ansätze:
  - Aktive Replikation
  - Quorumgestützte Protokolle (auf mehrheitliche Abstimmung basierend)

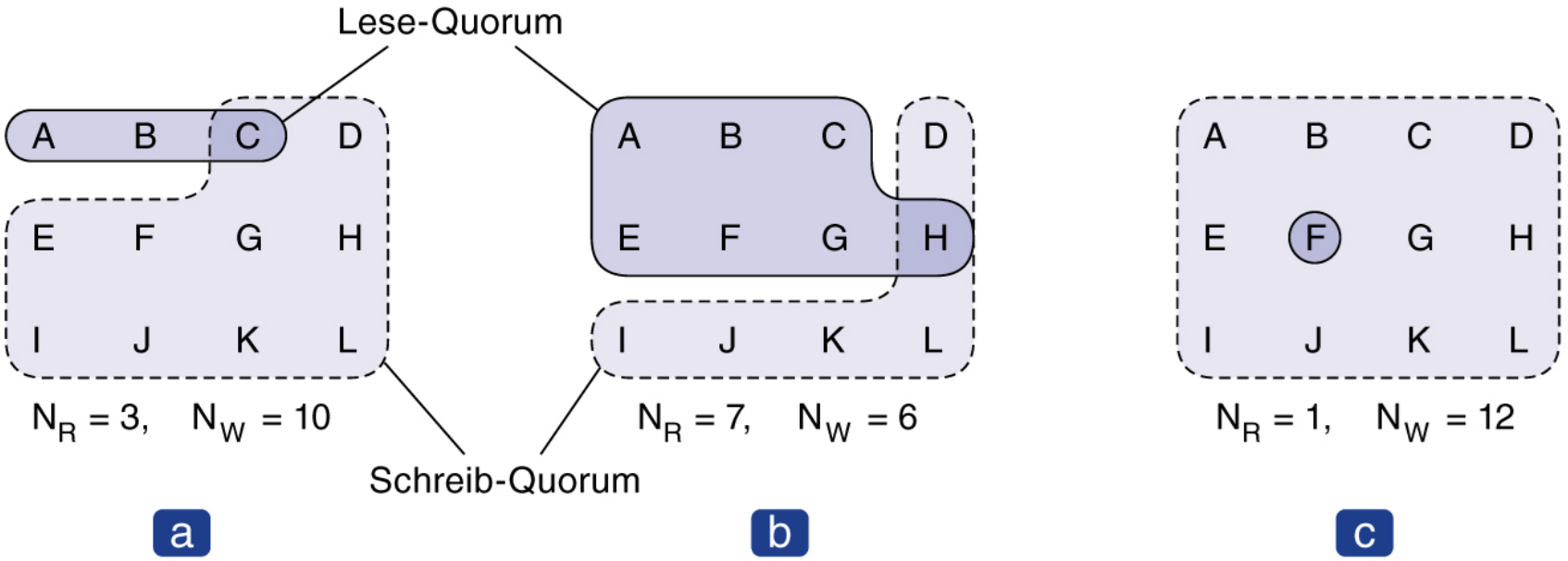
- Aktive Replikation
  - Aktualisierung in Form von Schreiboperationen an alle Replikate gesendet
  - Problem: Gleiche Reihenfolge ist nötig z.B. durch vollständig geordnetes Multicasting
    - Einsatz der logischen Uhr von Lamport oder
    - Einsatz eines Sequenzierers (zentraler Koordinator, der allen Operationen eindeutige fortlaufende Nummer gibt)
  - Problem nun: Skalierbarkeit

Konsistenzprotokolle
Protokolle für replizierte Schreibvorgänge

- Grundgedanke:
  - Clients bedürfen der Erlaubnis mehrerer Server, bevor sie ein repliziertes Datenelement lesen oder schreiben dürfen
- Einfache Realisierung: Es bedarf stets der Mehrheit von mindestens N/2+1
- Verallgemeinerung:
  - N – Anzahl aller beteiligter Server / aller Replikaten
  - N_R – Anzahl an nötigen Servern für ein Lesequorum
  - N_w – Anzahl an nötigen Servern für ein Schreibquorum
  - Bedingungen:
    1) $N_R + N_W > N$
    2) $N_W > N/2$
  - Bedingung 1): Verhinderung von Lese-Schreib-Konflikten
  - Bedingung 2): Verhinderung von Schreib-Schreib-Konflikten

Konsistenzprotokolle

Protokolle für replizierte Schreibvorgänge



Drei Beispiele für den Abstimmungsalgorithmus:
(a) eine korrekte Vorgabe für den Lese- und den Schreibvorgangssatz
(b) eine Wahl, die zu Schreib-Schreib-Konflikten führen kann
(c) eine korrekte Wahl, die als ROWA (Read-One, Write-All) bekannt ist

Quellenhinweis

- Vorlesung „Distributed Systems“ von Maarten van Steen
  - Thema: Consistency & Replication
    https://www.youtube.com/watch?v=pdxGtahoqlY