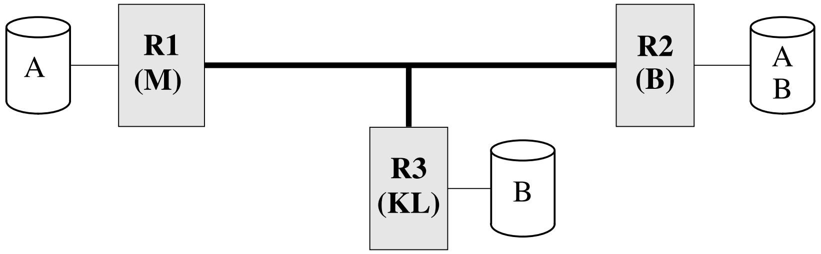
7. Replizierte Datenbanken

- **Replikation**
  - Anwendungsmöglichkeiten
  - Replikationsstrategien
  - Serialisierbarkeit
- **Aktualisierungsstrategien bei strenger Konsistenz**
  - Replikationskontrolle
  - Primary-Copy-Verfahren
  - ROWA-Verfahren / 2PC
  - Voting-Verfahren
- **Schwächere Formen der Datenreplikation**
  - Refresh-Alternativen
  - Schnappschuß-Replikation
  - Katastrophen-Recovery
  - Merge-Replikation
- **Replikation in kommerziellen DBS**
- **Datenreplikation in Parallelen DBS**
  - Spiegelplatten
  - Verstreute Replikation
  - Verkettete Replikation

Replikate in verteilten DBS



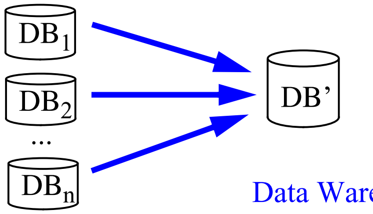
- **Vorteile**
  - \+ **erhöhte Verfügbarkeit der Daten**
    (Maskierung und Tolerierung von Fehlern im Netz)
  - \+ **Beschleunigung von Lesezugriffen**
    (bessere Antwortzeiten, Kommunikationseinsparungen)
  - \+ **verbesserte Möglichkeiten zur Lastbalancierung und Anfrageoptimierung**
- **Nachteile**
  - \- **hoher Aktualisierungsaufwand**
  - \- **größerer Speicherplatzbedarf**
  - \- **gestiegene Systemkomplexität**
    - Aktualisierungsalgorithmus
    - erweiterte Synchronisationsanforderungen (1-Kopien-Serialisierbarkeit)
    - Recovery-Probleme v.a. bei Netzwerk-Partitionierungen
    - Anfrageoptimierung

Replikation: Anwendungsmöglichkeiten

- **Replizierte Datenbankfragmente** in Verteilten und Parallelen DBS
- **Replizierte Katalogdaten** (Metadaten)
- **Katastrophen-Recovery**

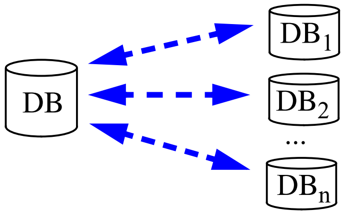
Katastrophen-Recovery

- **Data Warehousing:**
  Übernahme transformierter Daten in eigene Datenbank für Entscheidungsunterstützung



Data Warehouse

- **Mobile Computing:**
  Datenbank-Ausschnitte, Dateien ... auf Notebook, PDA etc.



Mobile Computing

- **Internet:**
  Replizierte Dateien, Datenbanken und Metadaten für schnelleren Zugriff und höhere Verfügbarkeit (Mirror Sites, Suchmaschinen, Portale)

Replikationsstrategien

- **Korrektheitskriterium / Synchronisation**
  - 1-Kopien-Äquivalenz:
    - vollständige Replikationstransparenz
    - Serialisierung von Änderungen
    - alle Replikate sind wechselseitig konsistent
    - jeder Zugriff liefert jüngsten transaktionskonsistenten Objektzustand
  - geringere Konsistenzanforderungen
    - Zugriff auf ältere Objektversionen
    - evtl. keine strikte Serialisierung von Änderungen, sondern parallele Änderungen mit nachträglicher Konflikt-Behebung

- **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 einem Knoten (Master, Publisher, Primärkopie)
    - Lesen an n Knoten (Slave, Subscriber, Sekundärkopien)

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

Serialisierbarkeit

■ **Definition der Serialisierbarkeit bei einer replikatfreien DB**

Ein Schedule S heißt serialisierbar, wenn es mindestens einen seriellen Schedule mit denselben TA gibt, der den gleichen DB-Zustand sowie die gleiche Ausgabe erzeugt wie S.

■ **Definition: 1-Kopien-Serialisierbarkeit**

Ein Schedule S einer replizierten DB heißt 1-Kopien-serialisierbar, wenn es mindestens eine serielle Ausführung der TA dieses Schedule auf einer replikatfreien DB gibt, der die gleiche Ausgabe sowie den gleichen DB-Zustand erzeugt wie S auf der replizierten DB.

➥ **Synchrone Aktualisierung** aller Kopien

■ **Abgeschwächtes Konsistenzkriterium**

- Manchmal ist eine Abschwächung des Konsistenzkriteriums erwünscht, um höhere Verfügbarkeit/höheren Durchsatz zu erzielen.
- Es müssen nicht alle Kopien gleichzeitig (synchron) aktualisiert werden.
- DB muß aber wieder zu einem konsistenten Zustand konvergieren.

➥ **Epsilon-Serialisierbarkeit** (ε-Serialisierbarkeit)

Serialisierbarkeit (2)

■ **Epsilon-Serialisierbarkeit**

Ein Schedule S einer replizierten DB heißt ε-serialisierbar, wenn gilt:

1. Der Schedule $S_U = S \setminus \{\text{Lese-TA}\}$ ist serialisierbar.
2. Für jede Lese-TA gilt:
   Solange die gelesenen Werte weniger als ε von den aktuellen (konsistenten) Werten abweichen, dürfen sich Aktionen einer Lese-TA beliebig mit den Aktionen anderer mit dieser in Konflikt stehenden TA überschneiden.

Manchmal wird die Überlappungsgrenze nicht über ein Wertintervall (als gewichteter Überlappungsparameter ε), sondern über die Anzahl der erfolgten Aktualisierungen definiert.

■ **ε-Serialisierbarkeit erlaubt begrenzte Kompatibilität (0)**
zwischen dem Zugriff einer Lese-TA (R_Q) und einer Schreib-TA W_U:

	R_Q^j	R_U^j	W_U^j
R_Q^i	+	+	0
R_U^i	+	+	-
W_U^i	0	-	-

Standard-kompatibilität

0: bis zum Erreichen der ε-Grenze werden die Konflikte mit Schreib-TA ignoriert

Serialisierbarkeit (3)

- **Ausgangsituation mit Konten k_1 und k_2**

A: $k_1=1000$, $k_2=1000$

B: $k_1=1000$

- **Transaktionen**

1. T_U überweist 200 DM von k_1 nach k_2:

$w^A_{T_U}(k_1, -200)$, $w^B_{T_U}(k_1, -200)$, $w^A_{T_U}(k_2, +200)$

2. T_Q wird gleichzeitig auf Knoten B gestartet, um den Kontostand aller Konten (für statistische Zwecke) zu bestimmen:

$r^B_{T_Q}(k_1)$, $r^A_{T_Q}(k_2)$

- **Mögliche Schedules auf Knoten A und B**

$S_A = (w^A_{T_U}(k_1, -200), w^A_{T_U}(k_2, +200), r^A_{T_Q}(k_2))$

$S_B = (r^B_{T_Q}(k_1), w^B_{T_U}(k_1, -200))$

➥ Der globale Schedule $S = (S_A, S_B)$ ist nicht 1-Kopien-serialisierbar!

- **Endzustand**

A: $k_1= 800$, $k_2=1200$

B: $k_1=800$

➥ Falls ein gewichteter ε-Überlappungsparameter größer 200 DM gewählt wurde (oder eine ε-Überlappung von mehr als 1 TA), dann ist $S = (S_A, S_B)$ ein korrekter ε-serialisierbarer Schedule

Replikationskontrolle

- **Zielkonflikte beim Einsatz von Replikaten**

Erhöhung der Verfügbarkeit und des effizienten Zugriffs

→ große Kopienanzahl

→ Zugriff über beliebige und möglichst wenige Kopien

Minimierung des Änderungsaufwandes

→ kleine Kopienanzahl

→ möglichst wenige Kopien synchron aktualisieren

Erhaltung der Datenkonsistenz

→ kleine Kopienanzahl

→ möglichst viele (alle) Kopien wechselseitig konsistent halten (synchron aktualisieren)

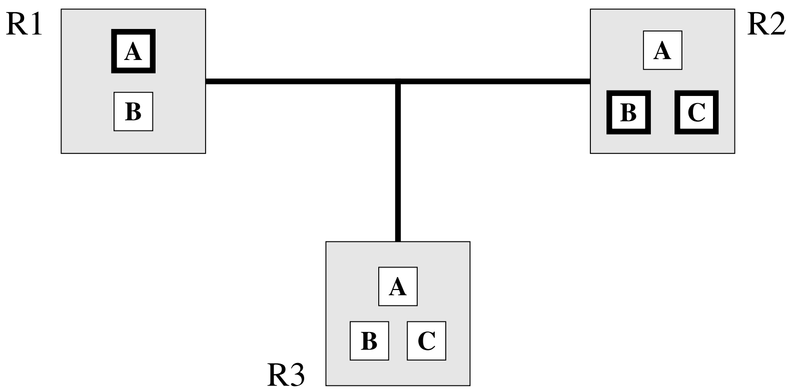
Ziel: 1-Kopien-Äquivalenz

Grobklassifikation Replikationsstrategien

Korrektheits-ziel	strenge Konsistenz (1-Kopien-Serialisierbarkeit)		schwache Konsistenz (weak consistency)	
Refresh-Zeitpunkt	asynchron (lazy)	synchron (eager)	asynchron (lazy)	
#Änderer (Masters) pro Objekt	1	n	n	1
Beispiel-Strategien	*- Primary Copy (mit Lesen aktueller Daten)*	*- ROWA (2PC)* *- Voting*	*- Merge-Replikation („Update Anywhere“)*	*- Primary Copy mit Lesen veralteter Daten* *- Schnappschuß-Replikation* *- Standby-Replikation*

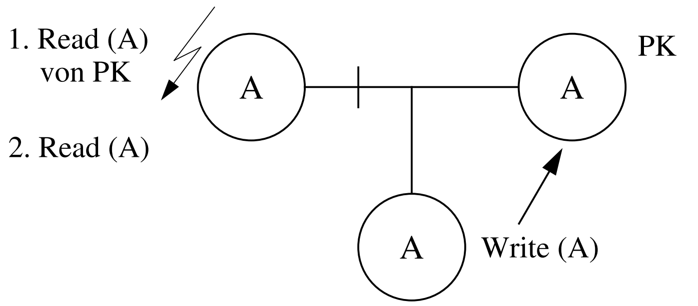
Primärkopien-Verfahren

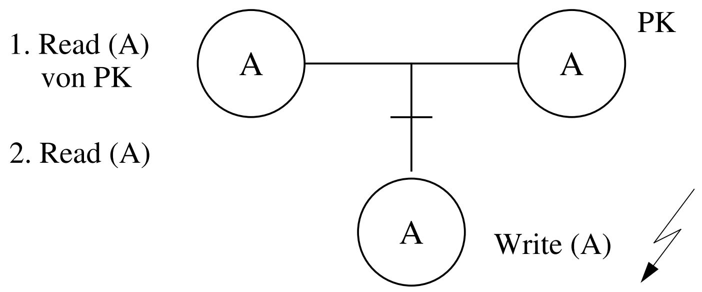
- **Asymmetrisches Verfahren**
  - 1 Primärkopie, n Sekundärkopien pro Objekt
  - Objekt besitzt Versionsnummer
- **Synchrone Änderung nur für Primärkopie** (primary copy)
  - Bearbeitung von Schreib- und Lesesperren beim Primärkopien-Rechner (Publisher)
  - verzögerte/asynchrone Aktualisierung der Sekundärkopien (Subscriber) durch Primärkopien-Rechner
  - Lesen: Primärkopien-Rechner teilt Versionsnummer der aktuellen Kopie bei Gewährung der Lesesperre mit; Lesen einer lokalen (aktuellen) Kopie
- **Alternativen für Lesezugriffe**
  - Lesen der lokalen Kopie ohne Sperre beim Primärkopien-Rechner (mglw. veraltet)
  - Lesen der Primärkopie (Replikation wird nicht genutzt!)



Primärkopien-Verfahren (2)

■ **Netzwerk-Partitionierung:**
nur in der Partition mit der Primärkopie können weiterhin Änderungen erfolgen





■ **Ausfall des Primärkopien-Rechners:**

- **statischer Ansatz:** keine weiteren Schreibzugriffe möglich bis Primärkopien-Rechner wieder aktiv
- **dynamischer Ansatz:**
  Bestimmung eines neuen Primärkopien-Rechners
  - Bei Netzwerk-Partitionierungen darf höchstens in einer Partition Verarbeitung fortgesetzt werden (z.B. wo Mehrzahl der Kopien vorliegt)
  - Neuer PK-Rechner muß ggf. zunächst seine Kopie auf den neuesten Stand bringen ("pending updates")

Read-one-Copy

■ **Synchrone Aktualisierung**

- Leser sehen immer den aktuellen Objektzustand
- Verfahren dieser Klasse können als Spezialfall der Voting-Verfahren angesehen werden (R=1)

1. Write-All / Read-Any-Methode

(Read-One / Write-All, ROWA)

■ **Bevorzugte Behandlung von Lesezugriffen**

- Lesen des nächstgelegenen (lokalen) Replikats
- erhöhte Verfügbarkeit für Leser

■ **Sehr hohe Kosten für Änderungstransaktionen**

- Schreibsperren bei allen Rechnern anfordern
- Propagierung der Änderungen im Rahmen des 2PC-Protokolls

■ **Verfügbarkeitsproblem**

- Änderungen nur bei Verfügbarkeit aller kopienhaltenden Knoten
- Knotenausfall/Partitionierung kann Schreibverfügbarkeit betreffen

2. Write-All-Available-Variante

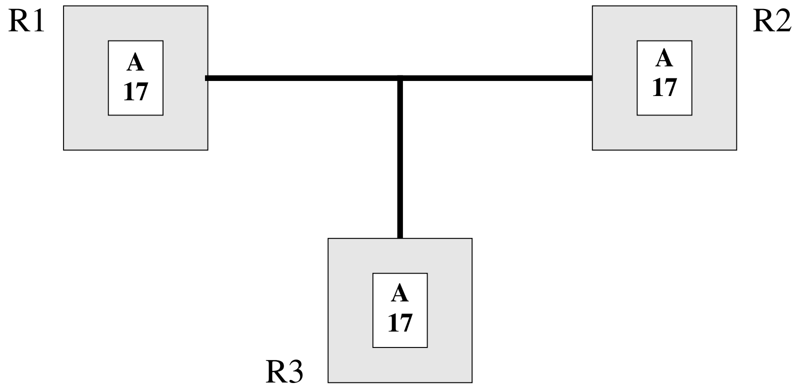
■ Nur verfügbare Replikate werden geändert

- für ausgefallene Rechner werden Änderungen bei Restart nachgefahren
- funktioniert nicht bei Netzwerk-Partitionierungen

3. Write-One / Read-All ?

Voting-Verfahren

- **„Demokratischer" Ansatz**
  - Synchronisation der Zugriffe erfolgt durch Abstimmung
  - Grundform: Majority Consensus
- **Eigenschaften**
  - Lesen oder Schreiben eines Objektes verlangt Zugriff auf Mehrheit der Replikate
  - Jedes Replikat kann gleichzeitig von mehreren TA gelesen, jedoch nur von einer TA geändert werden
  - Bestimmung der aktuellen Objektzustandes über Versionsnummer



- **Fehlerfall:**
  Fortsetzung der Verarbeitung möglich, solange die Mehrheit der Replikate noch erreichbar ist
- **Probleme:**
  - hohe Kommunikationskosten für Lese- und Schreibzugriffe
  - ungeeignet für N=2 („Read All, Write All")

Voting-Verfahren (2)
(weighted voting, quorum consensus)

- **Jede Kopie erhält bestimmte Anzahl von Stimmen (votes)**
- **Protokoll:**
  - Lesen erfordert R Stimmen (Read-Quorum)
  - Schreiben erfordert W Stimmen (Write-Quorum)
  - Schreib/Lese-Überschneidungsregel:
    $R + W > V$ (= Summe aller Stimmen)
  - Schreib/Schreib-Überschneidungsregel:
    $W > V / 2$
- **Eigenschaften:**
  - gleichzeitiges Lesen und Schreiben nicht möglich
  - jeder Zugriff auf R (W) Kopien enthält wenigstens eine aktuelle Kopie
  - Festlegung von V, R und W erlaubt Trade-off zwischen Lese- und Schreibkosten sowie zwischen Leistung und Verfügbarkeit
  - andere Verfahren ergeben sich als Spezialfälle
- **Beispiel 1:**
  - 5 Kopien mit jeweils 1 Stimme
  - R=1, W=5 ➥ Write All, Read Any
  - R=3, W=3 ➥ Majority Consensus
- **Beispiel 2:**
  - 5 Kopien
  - 4 Kopien ohne Stimme, 1 Kopie mit 1 Stimme
  - R=1, W=1 ➥ Primärkopien-Verfahren

Schwächere Replikationsformen

- **Bisherige Prämissen**
  - vollständige Replikationstransparenz
  - jede Kopie ist immer auf dem aktuellen Stand (strenge Konsistenz)
  - jede Kopie kann gelesen und geändert werden (außer bei PK)

- **Gravierende Probleme**
  - hohe Änderungskosten, da jede Änderung synchron propagiert wird
  - Verfügbarkeitsprobleme (besonders bei geographischer Verteilung)
  - geringe Skalierbarkeit
  - nicht anwendbar für mobile Replikate (meist offline)

- **Schwächere Konsistenzformen**
  durch asynchrone Aktualisierung der Kopien
  - Schnappschuß-Replikation
  - „fast” aktuelle Kopien, z. B. für Unterstützung einer schnellen Katastrophen-Recovery

- **Bei n Änderungsknoten pro Objekt** („Update Anywhere”)
  - asynchrones Refresh führt zu Konsistenzproblemen
  - anwendungsspezifische Konfliktbehebung nötig (Merge-Replikation)

- Unterstützung durch die meisten DBS-Hersteller:
  Oracle, IBM DataPropagator, MS SQL-Server, Sybase, Informix, CA-Ingres, ...

Refresh-Alternativen

- **Zwischenpufferung weiterzuleitender Änderungen**
  in persistenten Warteschlangen

- **Verzögerungsdauer: Aktualität vs. Overhead**
  - baldmögliche Weiterleitung nach Commit der Änderungstransaktion (ASAP)
  - spätere Weiterleitung:
    - feste Zeitabstände
    - Anzahl gemeinsam übertragbarer Änderungen ...

- **Push- vs. Pull-Replikation**
  - Push: Notifikation von Änderungen durch Master / Publisher
  - Pull: Probing durch Slave / Subscriber (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

- **sequentielle vs. parallele Aktualisierung**
  (1 vs. n Änderungsströme)

Schnappschuß-Replikation

- **Merkmale**
  - Erzeugung einer materialisierten Anfrage / Sicht
  - i. allg. nicht auf dem neuesten Stand
  - typischerweise nur lesender Zugriff
  - Replikation für Benutzer in der Regel nicht transparent
  -

Beispiel:

CREATE SNAPSHOT DS_Gehalt (Abt, Geh) **AS**
SELECT Anr, AVG (Gehalt)
FROM Pers
GROUP BY Anr
REFRESHED EVERY MONTH;

- **Aktualisierung der Kopie (Refresh)**
  - periodisch (täglich, monatlich etc.)
  - explizit auf Anforderung:

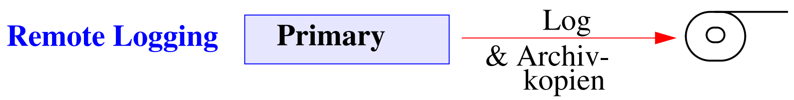
    REFRESH SNAPSHOT DS_Gehalt

- **Bewertung**
  - geringer Änderungsaufwand
  - keine Synchronisationskonflikte auf der Kopie
  - zahlreiche Anwendungsmöglichkeiten: Warehousing, Produktkataloge, ...

Katastrophen-Recovery

- **Traditionell: Zurückgehen auf Archivkopie**
  - Verlust vieler Transaktionen, falls aktuelle Log-Daten seit Erstellung der Archivkopie verlorengehen
  - zeitaufwendiges Einspielen der Archivkopie

- **Remote Logging: Übertragung der Log-Daten** an entferntes Rechenzentrum



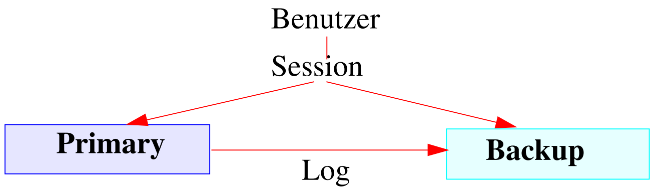
- **Alternative für hohe Verfügbarkeit:**

  zwei räumlich entfernte Rechenzentren mit replizierter Datenbank

- **Zwei gleichberechtigte (aktive) Rechenzentren**
  - Aufteilung der Last im Normalbetrieb
  - gegenseitiger Austausch der Änderungen (z.B. gemäß Primary-Copy-Schema)
  - hoher Realisierungsaufwand

Katastrophen-Recovery (2)

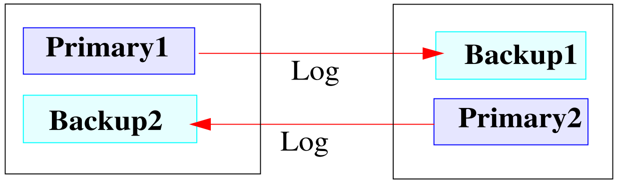
■ **Passives Stand-By-System (Hot Stand-By)**



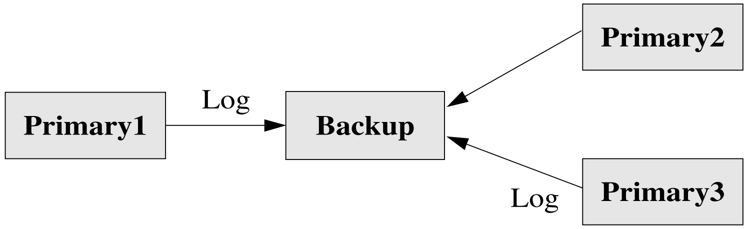
Grundmodell

- 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)
- Kopie kann weitergehend genutzt werden, z.B. für komplexe Anfragen (Entlastung des OLTP-Betriebes) und Administrationsaufgaben

■ **Konfigurationsvarianten**



Wechselseitiges Backup (zwei Systempaare)



Zentraler Backup-Knoten

Katastrophen-Recovery (3)

■ **Unterschiedliche Sicherheitsstufen**

- 1-sicher (1-safe)
  lokales Commit am Primary, asynchrone Übertragung der Änderungen an Backup
- 2-sicher (2-safe)
  synchrone Aktualisierung des Backup;
  Kommunikation in Commit-Phase 1
- "sehr sicher"
  2PC zwischen Primary und Backup;
  beide Knoten müssen Commit zustimmen

➥ keine Verarbeitung bei Backup-Fehler !

■ Gemeinsame Unterstützung von 1-sicheren und 2-sicheren Transaktionen wünschenswert
(Abhängigkeiten bei DB-Rekonstruktion zu beachten)

■ Probleme bei mehreren Log-Strömen zwischen Primary und Backup

Merge-Replikation

- **Unsynchronisierte Änderung eines replizierten Objekts** an mehreren Knoten mit asynchroner Propagierung der Änderungen
  - Performance- und Verfügbarkeitsvorteile gegenüber synchroner Aktualisierung
  - unabdingbar für mobile DB-Nutzung mit Änderungsbedarf (z.B. Außendienst-Mitarbeiter)
  - Problem: nachträgliche Konflikt-Behandlung, Mischen paralleler Änderungen (Merge-Replikation)

- **Konfliktmöglichkeiten** für Update, Insert (z.B. Eindeutigkeit) und Delete

- **Konflikterkennung**
  - Objektänderungen enthalten Zeitstempel v der Vorgängerversion und neuen Wert
  - **Konflikt:** falls lokaler Zeitstempel einer zu aktualisierenden Objektversion von v abweicht
  - Konfliktwahrscheinlichkeit wächst mit Anzahl der änderbaren Replikate und Aktualisierungsverzögerung

- **Anwendungsspezifische Konflikt-Behandlung** (reconciliation)
  - vordefinierte Strategien in existierenden DBS: „last timestamp wins“, „site priority“, „average“ ...
  - benutzerdefinierte Konfliktauflösungsroutinen oder manuelle Konfliktbehebung
  - Gefahr von „lost updates“ und anderen Anomalien

Replikation in MS SQL-Server

- Drei unterschiedliche Publish/Subscribe-Replikationsmodelle: **Snapshot, Transactional und Merge Replication**

- Replikation von Tabellen, vertikalen/horizontalen Fragmenten sowie DB-Prozeduren/Operationen

- **Zwei Master/Slave-Fälle mit lesenden Abonnenten**
  - Snapshot Replication: Daten werden zu bestimmten Zeitpunkten an Abonnenten verteilt (Push- vs. Pull-Abonnenten)
  - Transactional Replication: Log-Daten zu Änderungen an replizierten Daten werden transaktionsweise weitergeleitet und auf den Kopien angewandt
  - Spezialfälle:
    - 1 Publisher / n Subscriber
    - n Publisher / 1 Subscriber (Außendienst-Mitarbeiter - Zentrale)

- **Merge Replication: mehrere Änderer pro Replikat**
  - Einsatz von Triggern zur Propagierung von Änderungen
  - Konflikterkennung auf Satz- oder Attributebene (Merge Agent)
  - benutzerdefinierte Konfliktauflösungsstrategien möglich

Replikation in weiteren DBS

■ **Oracle**

- "Table snapshots" und "Row replication"
- Aktualisierung replizierter Tabellen über Trigger: synchrone Aktualisierung oder Erzeugen asynchroner Update-Aufträge (RPCs)
- Update Anywhere: benutzergesteuerte Behebung von Update-Konflikten
- Standby-DB für Katastrophen-Recovery (Transfer der Redo-Log-Daten für komplette DB)

■ **IBM Data Propagator**

- flexible SQL-basierte Auswahl und Transformation der Quell-Daten (Aggregation, Joins, Verwendung von stored procedures)
- Trigger- und Log-basierte Änderungspropagierung; vollständiges oder inkrementelles Refresh
- Update Anywhere mit mehreren Strategien zur Konfliktbehebung

■ **Sybase Replication Server**

- 1 Primärkopie und n Kopien (Zeilen-Granulat)
- asynchrone Aktualisierung (transaktionskonsistent) durch Übertragung der Log-Daten

■ **Informix**

- vollständige DB-Kopie an entferntem Ort
- synchrone oder asynchrone Aktualisierung
- nur Lesezugriff auf Kopie

Replikation in Parallelen DBS

■ **Replizierte Zuordnung von Fragmenten**
im Rahmen der Datenallokation

■ **Ziele einer Replikation**

- Fehlertoleranz gegenüber Externspeicherfehlern
- Fehlertoleranz gegenüber Rechnerausfällen
- günstige Lastbalancierung im Normalbetrieb und Fehlerfall

■ **Eigenschaften**

- Lokalität und Knotenautonomie weniger/nicht relevant
- i.allg. synchrone (parallele) Aktualisierung der Replikate

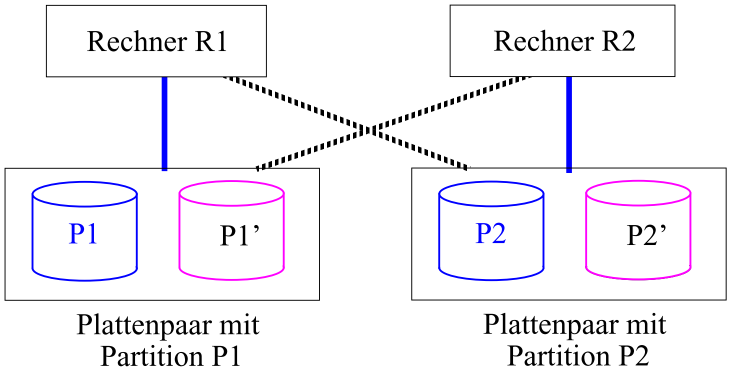
■ **Drei Replikationsansätze mit doppelter Speicherung der Daten**

- Spiegelplatten
- Verstreute Replikation (Teradata)
- Verkettete Replikation (Gamma-Prototyp)

Spiegelplatten

- **Eigenschaften**
  - Sie bieten effektiven Schutz gegen Gerätefehler (weitverbreitet zur Maskierung von Plattenfehlern)
  - Sie sollten unabhängige Fehlermodi besitzen: verschiedene Plattenkontroller, unabhängige Stromversorgung usw.
  - Sie sind aber als Schutzmaßnahme gegen SW-Fehler (fehlerhafter Inhalt eines Blockes) und Umgebungsfehler nicht geeignet
  - simultanes Schreiben aller Änderungen auf zwei (oder mehr) Platten
  - **verbesserte Lese-Performanz**
    - Zugriffszeitoptimierung
    - gleichmäßige Aufteilung der Leseanforderungen auf alle Platten (Minimierung der Warteschlangenverzögerungen, Lastbalancierung)

- **Shared Nothing: Replikation auf einen Knoten beschränkt**

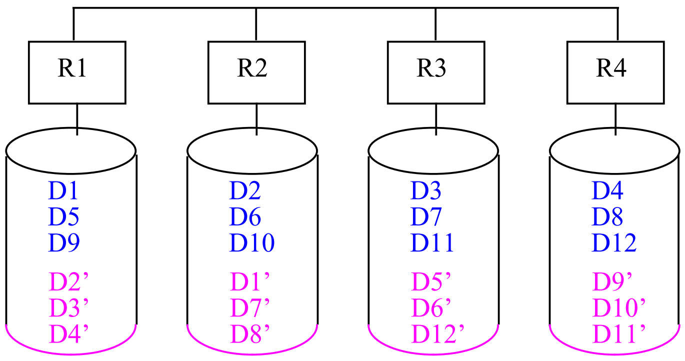


- **Rechnerausfall** erfordert Übernahme der kompletten Partition durch zweiten Rechner
  ➥ **sehr ungünstige Lastbalancierung im Fehlerfall**

Verstreute Replikation (Interleaved Declustering)

- **Ziel: bessere Lastbalancierung im Fehlerfall**
  - Datenknoten einer Relation werden in Gruppen von je G Rechnern unterteilt (node groups)
  - Replikate zu Daten eines Rechners werden gleichmäßig unter den G-1 anderen Rechnern der Gruppe verteilt
  - nach Crash erhöht sich Last jedes überlebenden Rechners der Gruppe gleichmäßig um Faktor G/G-1

Beispiel (D=G=4)

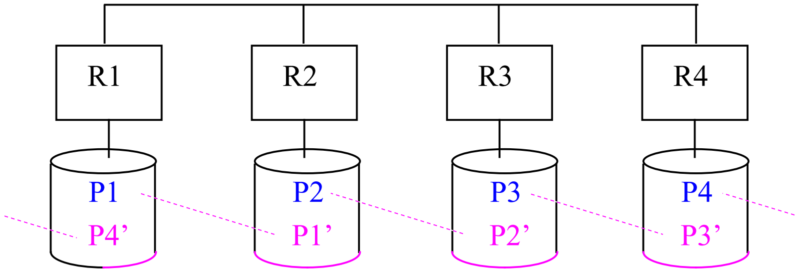


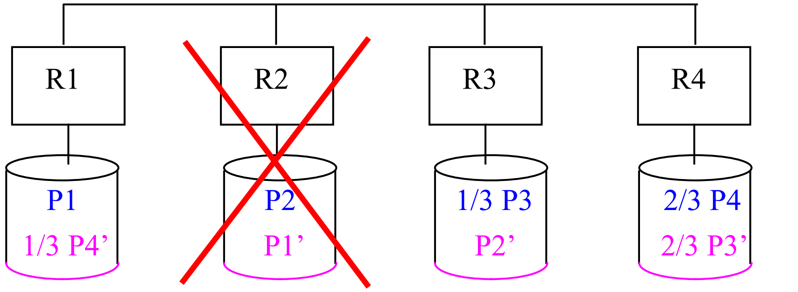
- **Gruppenbildung**
  - Ausfall mehrerer Rechner beeinträchtigt Datenverfügbarkeit nicht, sofern jeweils verschiedene Gruppen betroffen sind
  - Wahl von G erlaubt Kompromiß zwischen Verfügbarkeit und Lastbalancierung
  - Extremfälle: G=D (= Verteilgrad der Relation) und G=2

- **Nutzung der Replikate** zur Lastbalancierung im Normalbetrieb aufwendig

Verkettete Replikation (Chained Declustering)

- **Ziel:** hohe Verfügbarkeit wie für Spiegelplatten + günstige Lastbalancierung im Fehlerfall wie für verstreute Replikation
  - Kopie der Partition von Rechner R_i wird vollständig am "nächsten" Rechner $R_{(i \text{ MOD } G) + 1}$ der Gruppe gehalten





  - selbst Mehrfachausfälle in einer Gruppe erhalten die Verfügbarkeit aller Daten, so lange nicht zwei benachbarte Rechner ausfallen
- **Lastbalancierung im Fehlerfall**
  - Zugriffe auf betroffener Partition sind vollständig vom "nächsten" Rechner der Gruppe auszuführen
  - Zugriffe auf den anderen Partitionen sind unter den beiden Kopien umzuverteilen, so daß Lastbalancierung erreicht wird
  - keine Umverteilung von Daten, sondern nur Anpassung der Verteilungsinformationen

Zusammenfassung

- **Zielkonflikte:**
  Verfügbarkeit und effizienter Zugriff, Minimierung des Änderungsaufwands, Erhaltung der Datenkonsistenz
- **Synchronisationsansätze:** ROWA, Voting, Primary Copy
  - sehr große Anzahl an Synchronisationsverfahren
  - Primary-Copy-Verfahren ist konzeptionell einfach, aber anfällig gegen Knotenausfällen und Netzpartitionierungen
  - Voting-Verfahren sind wesentlich komplexer und teurer, erlauben aber viele Variationen und Optimierungen; sie tolerieren außerdem Knotenausfälle und einfache Netzpartitionierungen
- **Probleme bei strikter Synchronisation** (1-Kopien-Serialisierbarkeit)
  - synchrone Aktualisierung erfordert sehr hohen Aufwand (2PC)
  - geringe Skalierbarkeit
  - Viele Anwendungen erfordern nur Kopien für Lesezugriff, die nicht immer auf aktuellstem Stand sein müssen
- **Kommerzielle DBS** unterstützen unterschiedliche Replikationsmodelle
  - Publish/Subscribe-Ansätze
  - Primärkopien vs. „Update Anywhere“
  - synchrone oder verzögerte Übertragung von Änderungen
  - einfache Kopien und abgeleitete (transformierte) Replikate
  - benutzergesteuerte Konfliktbehandlung paralleler Änderungen (Merge-Replikation)
  - Parallele DBS: Datenduplizierung mit paralleler Aktualisierung
- **Zahlreiche Anwendungsmöglichkeiten, u.a.**
  - Katastrophen-Recovery (mit verschiedenen Sicherheitsstufen)
  - Nutzung von Replikaten in mobilen Endgeräten
  - Data Warehousing