# Grundlagen

Zwei Hauptgebiebte für verteilte Datenbanksysteme.

1. **Dezentralisierung** (, Verfügbarkeit, Durchsatz, Skalierung, Antwortzeiten, Hardwarekosten)
2. **Integration** (Produktions-, andere Kostensenken, Datenbereitstellung, unterschiedliche Systeme nutzen) → Kooperation autonomer Systeme

Verfügbarkeit wird durch Redundanz erhöht

Durchsatz durch parallele Ausführung der Abfragen → Serialisierbarkeit

Antwortzeiten durch lokale Systeme / Lastverteilung / Redundante Daten

Skalierung zentral → vertikale Skalierung (Limit), verteilt → horizontale Skalierung

**Besonderheiten Anfragebearbeitung VDBMS** – lokale und globale Optimierung wichtig, Lastverteilung, Perfomanceverbesserung durch parallele Abfragen, Ziel *Reduzierung Kommunikationsaufwand, → Join Optimierung (kostenintensive Operationen)*

## *Verteilung von Relationen*

**Allokation –** physische Speicherung der Daten auf verschiedenen Knoten, Aspekte nach Effizienz Anfragebearbeitung / Redundanz für hohe Verfügbarkeit

**Partitionierung** – Aufteilung von Relationen auf verschiedene Knoten, horizontal (zeilenweise) oder vertikal (spaltenweise)

**Horizontale Partitionierung** – Rges = R1 UN R2 UN R3 (Vereinigung)

**Vertikale Partitionierung** – Mit Hilfe PJ (Projektion) realisiert

**Optimale Partitionierung** – Metainformationen nötig, Welche Anwendung wie, Häufigkeit, mit welchen Auswahlprädikaten, worauf zugreift. Min Term Prädikate (konjuktion einfacher Prädikate) bilden p1+- UND p2+- … Prädikate einsetzen, und prüfen ob erfüllbar und sinnhaft

**Verlustfreiheit der Zerlegung** – Wenn gilt R = R1 NJN R2 NJN Rn

**Single System Image** – Nach außen soll ein VDBMS wie ein System wirken

## *Problem der Redundanz*

**Was kann redundant gespeichert werden** → globale Relation oder Partitionen

R*edundanz schafft Probleme bei der konsistenzhaltung der Daten* (Kopien) → Replikationsverfahren, Kopie Update Verfahren (vorherbestimmte Kopien werden upgedatet, Abstimmung bestimmt Update) – Performance Verlust bei Ausfall soll vertretbar sein

*Redundanz schafft Probleme beim Inkonsistenten Lesen* – Es muss sichergestellt werden das Konsistente Daten gelesen werden

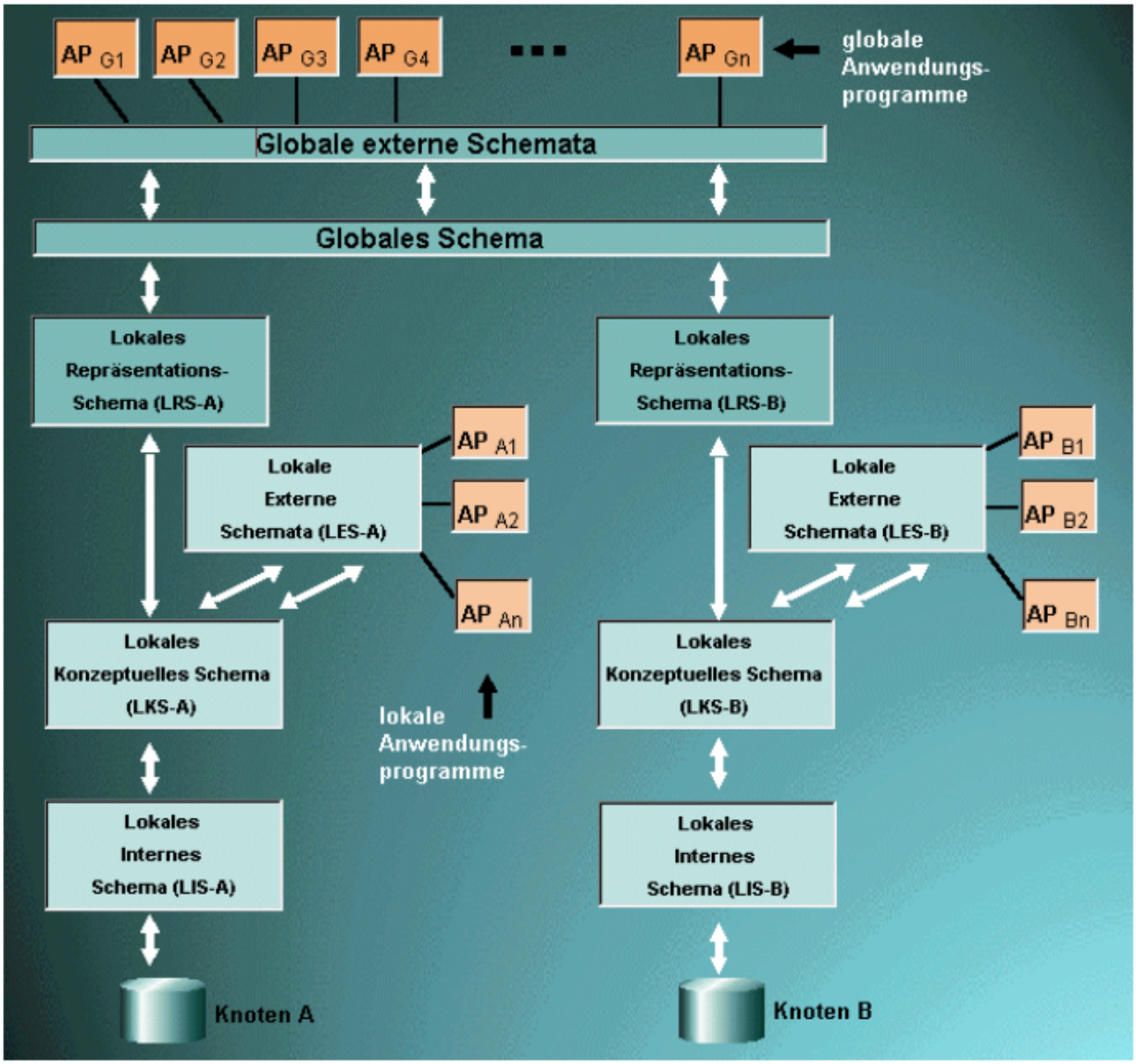
# Integration

**Probleme bei der Integration:** Konflikte über Systemgrenzen hinweg Datenmodell, Abfragedialekte, Hersteller, heterogene Schemata

Konflikte lokalen Schemata in der Semantik, Typen, der Struktur, der Wertebereiche, Schlüsselkonflikte, Beziehungskonflikte, komplett andere Systemarchitektur.

**Lösung – globales Schema als Layer** → Top Down bei Prä Integrieten Systeme (Google), Bottom Up Approach bei Post Integration, Vorgehensweise Binär (step by step) oder n stellig.

**Lokales Repräsentationsschema** als Layer zwischen zu integrierendem System und globalem Schema. *Verbirgt Heterogenität des lokalen Schemas*. Durch Transformation stellt es die Daten in einheitlicher Struktur und Semantik zur Verfügung. Auch *Exportschema* genannt. Das jeweils existierende lokale konzeptuelle Schema wird nicht verändert, ebensowenig die existierenden lokalen externen Schemata und deren Abbildungen auf das lokale konzeptuelle Schema. Die lokalen Repräsentationsschemata führen also eine **Homogenisierung der (Darstellung der) lokalen Schemata** durch, sofern diese global verfügbar gemacht werden sollen.



# Schemaarchitektur

**Aufbau Schema lokal**

- **Externes Schema** – In den ***externen Schemata*** (üblicherweise gibt es mehrere davon) wird festgelegt, welche Daten bzw. Datentypen und welche Beziehungen zwischen den Daten in welcher Form dem Anwender sichtbar gemacht bzw. zur Verfügung gestellt werden (Vergleichbar mit Views)
- **Konzeptuelles Schema -** Im ***konzeptuellen Schema*** sind alle in der Datenbank vorhandenen Entity-Typen sowie alle von der Realwelt abgeleiteten und für die Datenbank relevanten Beziehungen zwischen den Entity-Typen in einer geeigneten Form beschrieben
- **Internes Schema** Im ***internen Schema*** wird festgelegt, welche ***physischen Speicherungsstrukturen*** für die Speicherung der Primär- und Sekundärdaten zur Verfügung stehen. Unter *Primärdaten* verstehen wir die eigentlichen Benutzerdaten, während wir mit *Sekundärdaten* alle Arten von Hilfsdaten

**Aufbau globales Schema**

- **globales konzeptuelles Schema (GKS)**, das die globale "Außensicht" realisiert
- **globales Partitionierungsschema (GPS)**, das die Partitionierung der Relationen des GKS (nur intern sichtbar) beschreibt
- **globales Allokationsschema (GAS)**, das die physische Plazierung der Partionen (die Allokation) beschreibt.

# Schemaintegration Vorgehensmodell

- *Prä-Integrationsphase*: Festlegung der Vorgehensweise, Ermittlung der Entities und Beziehungen.
- *Vergleichsphase*: Ermittlung von Namens- und Strukturkonflikten.
- *Vereinheitlichungsphase*: Festlegung des Zielschemas
- *Restrukturierungs- und Zusammenfassungsphase*: Festlegung der lokalen Repräsentationsschemata und der erforderlichen Abbildungen.

Vorgehensweise Binär (step by step) oder n stellig.

# Aufgaben bei globalen Transaktionen

**Query processor (Anfragebearbeiter)** – Zerlegung der Anfrage in lokal ausführbare Teilanfragen → bnötigt Daten aus globalem Katalog (Partitionierungsinfos, Allokationsinfos)

**Transaktionsmanager** – Startet Primärtransaktion, Aufgaben: Ablaufsteuerung, 2PC – Protokoll, Fehlerbehandlung, Protokollierung, Schick Resultate an Anfrager

**Primär Transaktion** startet **Subtransaktionen** an den Transaktionsmanager der beteiligten Knoten

**Globaler Katalog -** Zentralisierter Katalog, Voll redundanter Katalog, Mehrfach-Katalog ("Cluster-Katalog"), Lokale Kataloge (kein globaler Katalog)

# Serialisierbarkeit

**Bedeutung:** Serialisierbarkeit ist dann gegeben wenn die Wirkung (Ergebnis) von parallelen Transaktionen dem entspricht wenn man sie nacheinander ausführen würde.

**Aus Vorlesung:** Die Forderung nach Serialisierbarkeit bedeutet, daß das bei paralleler (überlappter) Ausführung einer Menge von Transaktionen erzeugte Resultat auch durch mindestens eine serielle (also nicht-überlappte) Ausführungsreihenfolge dieser Transaktionen erzeugbar sein muß. Serialisierbarkeit gewährleistet somit den *logischen Einbenutzerbetrieb*

**Stichworte:** ACID → Isolation

**Anmerkung -** Wir halten also fest, daß im zentralen Fall die Konsistenz der Datenbank dadurch gewährleistet wird, daß die Synchronisationskomponente *nur solche Schedules* erzeugt bzw. zuläßt, *die serialisierbar* sind. Würde die weitere Ausführung einer Transaktion T dazu führen, daß gegen dieses Prinzip verstoßen würde, so wird T abgebrochen und zurückgesetzt.

**Globale Serialisierbarkeit** –Eine überlappte Ausführung einer Menge $T_g$ globaler Transaktionen, die an den Knoten $K_1$, $K_2$, ... $K_n$ ausgeführt werden, ist dann und nur dann korrekt, wenn

1. die sich an $K_1$, $K_2$, ..., $K_n$ ergebenden lokalen Schedules jeweils serialisierbar sind
2. sich aus der Menge der äquivalenten seriellen Schedules an $K_1$, $K_2$, ..., $K_n$ eine widerspruchsfreie serielle Ausführungsreihenfolge
   für die in $T_g$ enthaltenen globalen Transaktionen ableiten läßt.

**2 Phasen Commit und globale Serialisierbarkeit –** 2PC garantiert globale Serialisierbarkeit in dem erst nach dem globalen Commit alle Sperren freigegeben werden, globaler Commitzeitpunkt bestimmt Reihenfolge der Serialisierbarkeit

**Problem Commitzeitpunkt –** lokale Zeitunterschiede → Zähler oder Synchronisierte Zeit

Geben zwei globale Transaktionen zum gleichen Zeitpunkt (Commit Zeitpunkt) alle Sperren frei sind sie untereinander beliebig serialiserbar.

## *Allgemeine Fragen*

Wieso kann man sich trotz geringerer Übertragungskosten dazu entscheiden eine Anfrage durch einen Knoten nicht ausführen zu lassen → **Load Balancing**

Wie werden Transaktionen in verteilten DBs ausgeführt → **2 PC Protokoll**

**Auch bei Serialisierbarkeit lokaler Teiltransaktionen muss globale Serialisierbarkeit gegeben sein.** → Problem globale Dead Locks → globale Transaktionen bleiben hängen (keine Ausführung möglich) → Dead Lock Analyse zentral vs. verteilt (DDD) Warte-auf Graph Analyse nach Zyklen

**Bei 2PC-Protokoll bestimmt der globale Commitzeitpunkt die Reihenfolge der Serialisierbarkeit,** Sperren werden bei ready to commit gehalten.

**Relationsprofil** – Enthält Angaben über die Relation (Anzahl Tupel, Attribute, Attributgröße, Werausprägungen)

**Optimierung von Ausführungsplänen** – Durch Daten des Relationsprofils können Ausführungspläne Optimiert werden – Möglichst Relationen vor Joins reduzieren → Spart Übertragungs und Rechenkosten

# Anfragebearbeitung – Allgemein

**Globale Relationen können virtuell sein** → keine Basis Relation dazu vorhanden – Lösung Relation physisch erzeugen oder Anfrage transformieren in lokale Anfragen

**Optimierung globaler Anfragen** – entfernen Teilanfragen die leeres Ergebnis liefer, Join Optimierung

# Anfragebearbeitung – Join Optimierung

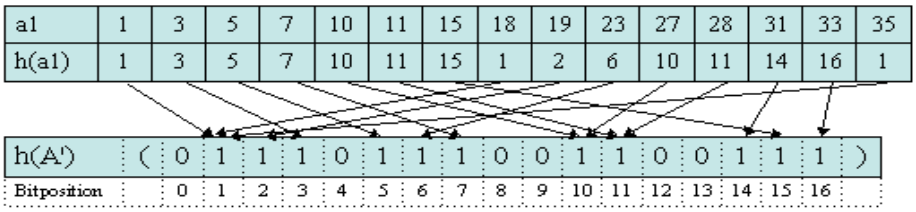
**Nested Loop Join –** Doppelt geschachtelte Loop Schleife, Ohne Index RelationScan, Es wird viel Übertragen da Äußere Schleife alles an die Innere Übergibt und diese zurück, Besonders bei NJN (Natural Join) Problematisch, Verteilten Fall selten wegen Kommunikationsaufwand, Vorgehen kleine Relation komplett übertragen, Alternative Semi Join

**Sort Merge Join –** Kann genutzt werden wenn Index oder Sortierng nach Join Attribut vorliegt, Beide Relationen werden Paralell durchsucht nach Kanidaten für die Vereinigung,

**Semi Join –** Es wird eine Projekt des Joints Attribut an den nächsten Knoten geschickt, damit die Relation reduziert, und diese reduzierte Trefferemenge zurück an den Knoten A und dort wird der Join berechnet, Vorteil bei wenig Ausprägungen des Join Attributes, Nachteil hoher Kommunikationsaufwand bei großen Projektionen

**Hashfilter Join -** Gleicher Ansatz wie Semi Join, Es wird statt echter Join Attribute ein Bitvektor übertragen, Attribute werden darauf mittels Hashfunktion abgebildet, Hashverfahren z.b. Modulo Verfahren, Pseudo Treffer können auftreten werden aber bei Join Berechnung eliminiert,

**Wieso Semi oder Hashfilter Joins** → Übertragungskosten reduzieren im verteilten Fall




**Was ist naive verteilte Join Verarbeitung** – kleine Relation wird an den Knoten der größeren übertragen und dort findet Join Berechnung statt → üblich aber suboptimal

# Globale Transaktionen

Transaktionen auf mehreren Knoten nennt man globale Transaktionen / lokale Transaktionen nur auf einem Knoten.

Verteilungstransparenz gegenüber Anwendungsprogrammen

# 2 Phase Commit Protokoll

Das 2PC-Protokoll basiert auf dem Grundgedanken, daß alle an der Ausführung einer globalen Transaktion T beteiligten Knoten darüber abstimmen, ob T global "committed" oder "aborted" wird. Hierzu dienen die folgenden beiden Phasen (daher der Name) und die in ihnen durchgeführten Aktionen:

**Phase 1**: "*Prepare to Commit*":
Der Koordinator fordert die anderen Teilnehmer auf, das Commit von T vorzubereiten (d. h. die gemachten Änderungen zu sichern), und fordert das Abstimmungsergebnis an.

**Phase 2**: "*Commit/Abort*":
Falls der Koordinator von allen Knoten Zustimmung erhält ("Commit-Fall"), dann meldet der Koordinator "Commit". Falls jedoch mind. ein Knoten seine Zustimmung verweigert, dann meldet der Koordinator "Abort" an alle Beteiligten. Anschließend geben der Koordinator und die anderen Teilnehmer die Sperren auf den Objekten von T frei.

Gerät eine Subtransaktion noch vor Eintreffen der Prepare-to-Commit-Nachricht (PREAPARE in Abb. 7-13) in Schwierigkeiten, so wird sie abgebrochen und dies dem Koordinator (entweder sofort oder bei Eintreffen der PREPARE-Nachricht) mitgeteilt (STIMME-ABORT).

- Kann die STIMME-COMMIT-Nachricht dem Koordinator nicht zugestellt werden, so kann sich der Knoten auf Abbruch entscheiden und dies dem Koordinator bei Wiederherstellung des Kontakts mitteilen (STIMME-ABORT).
- Hat der lokale Knoten seine Zustimmung abgegeben, so muß er die globale Entscheidung abwarten (WAIT). Er kann diese Entscheidung nicht aus eigener Entscheidung wieder rückgängig machen, ohne die Konsistenz der Datenbank zu gefährden (siehe nachfolgende Anmerkungen).
- Kann die Bestätigung des Commit oder Aborts (ACK in Abb. 7-13) nicht zugestellt werden, so kann die Subtransaktion dennoch beendet werden. Allerdings muß sichergestellt sein, daß der Koordinator bei Anfrage die korrekte Antwort erhält (siehe auch hierzu wieder Abschnitt 7.6.3).

Zusammenfassend können wir festhalten, daß nach Absetzen der STIMME-COMMIT-Nachricht bei Ausfall des Koordinators die betroffene Subtransaktion (zunächst einmal) dauerhaft blockiert ist. Dies ist der einzige Punkt, wo das 2PC-Protokoll zu einem globalen Blockierungszustand (da alle beteiligten Subtransaktionen in gleicher Weise davon betroffen sein können) führen kann.

**Anmerkungen:**
Durch "Rundfrage" bei den anderen an der globalen Transaktion beteiligten Subtransaktionen (sofern bekannt oder z. B. über eine globale Transaktions-ID identifizierbar) kann evtl. etwas über den Zustand der globalen Transaktion in Erfahrung gebracht werden:

**Bedeutung RtC Zustand für Teiltransaktionen** – ist Bereit dem globalen wunsch nach Commit / Abort folge zu leisten, Vor RtC müssen Logfiles für Sperren / Redo Infos / Undo Infos geschrieben werden.

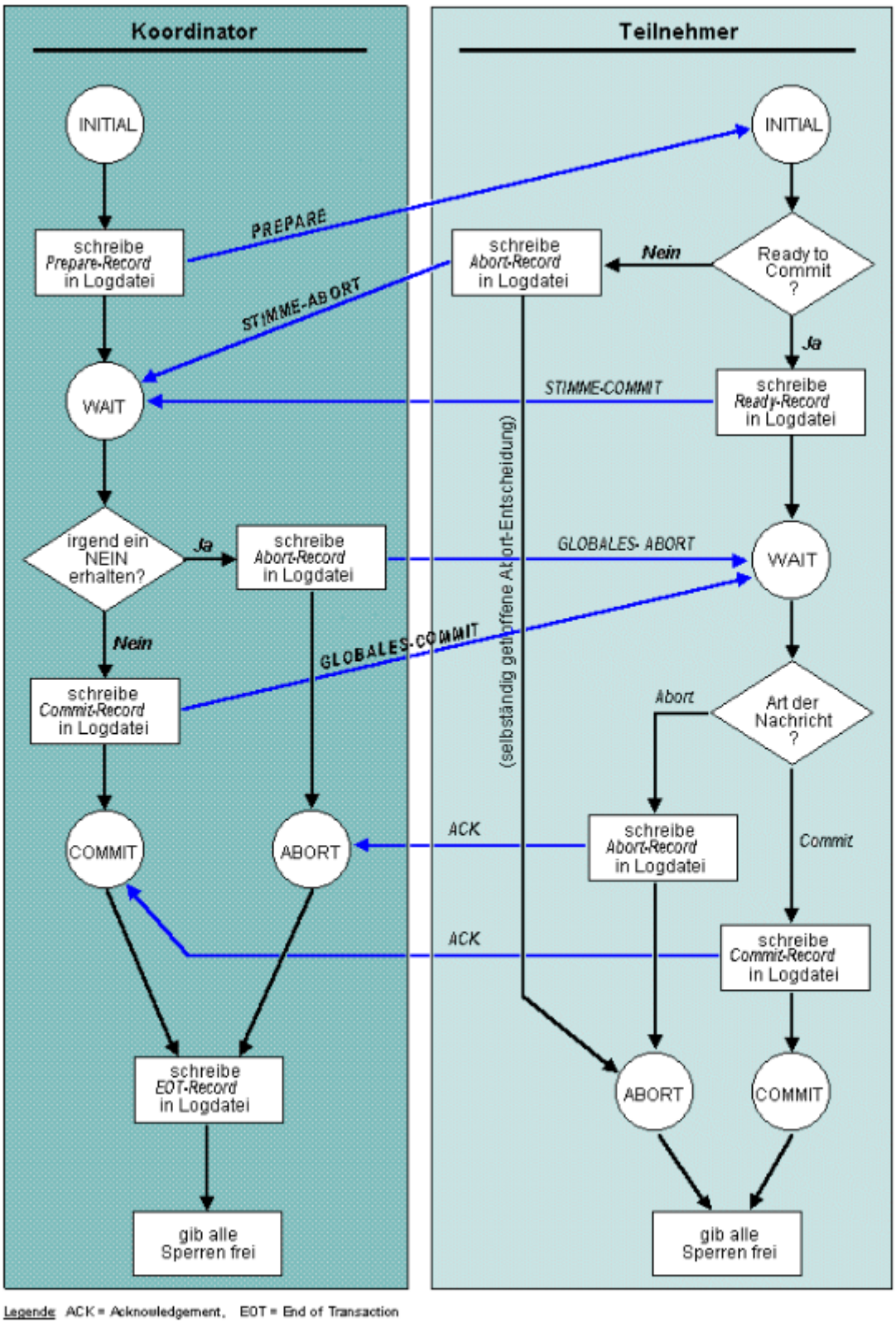



**Abb 7-13: Ablauf des 2PC-Protokolls**

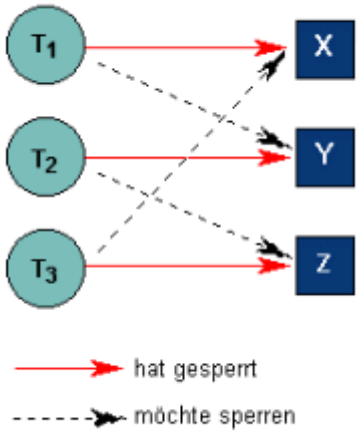
# Zwei-Phasen-Sperrprotokoll (2PL-Protokoll)

Eine Transaktion beachtet das 2PL-Protokoll genau dann, wenn sie

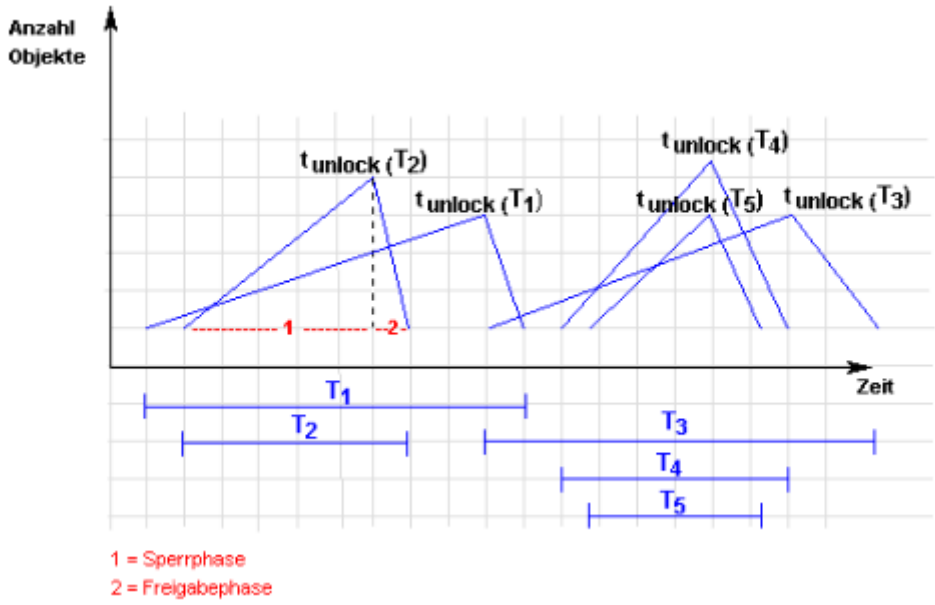
1. jedes Datenbankobjekt, das sie zur erfolgreichen Ausführung benötigt, vor dem Zugriff mit einer (geeignete) Sperre belegt.
2. keine Sperren mehr anfordert, sobald sie einmal eine Sperre freigegeben hat.

**Dead Lock / Verklemmung**

Typischerweise (da nicht im voraus bekannt), werden die Sperren sukzessive angefordert. Hierdurch kann die bereits oben erwähnte zyklische "Wartet-auf"-Situation (siehe Abb. 8-1), *Verklemmung* oder *Deadlock* genannt, auftreten. Deshalb muß bei Einsatz dieser Synchronisationsmethode auch eine Behandlung von Verklemmungen vorgesehen werden




Beispiel Serialisierbarkeit beim 2PL-Protokoll: T4 und T5 sind untereinander beliebige serialisierbar. Da gleichzeitig die Sperren wieder freigegeben werden.




Implementiert wird normalerweise über **Sperrtabellen, Schreibsperre (exklusiv), Lesesperre (shared)**

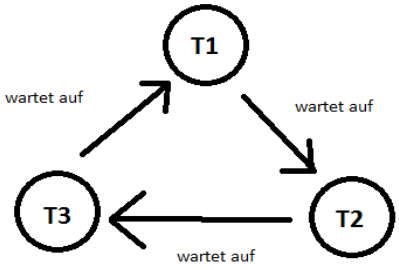
# Erkennung von Dead Locks / Verklemmungen

Möglichkeiten der Erkennung

**Time outs** – Nach dem Ablauf einer Zeitvorgabe wird abgebrochen, Problem hohe Last Time Outs, Zu hohe Time outs → langer Stillstand aller beteiligten Transaktionen

**Dead Lock Erkennung –** in regelmäßigen Abständen, bei Blockierung, bei Time out

**Dead Lock Erkennung in zentralen Systemen –** Über Warte-auf Graph, Grundlage Sperrtabelle, liegt ein Zyklus vor → Dead Lock. In der Sperrtabelle steht was Transaktion gesperrt hat, und Transaktion fordert Sperre an die von anderer Transaktion schon gesperrt ist




**Dead Lock Erkennung in verteilten Systeme** Keine lokaen Verklemmungen bedeutet nicht das keine globale Verklemmung vorliegt. Daher müssen globale warte auf informationen verfügbar gemacht werden für die Analyse

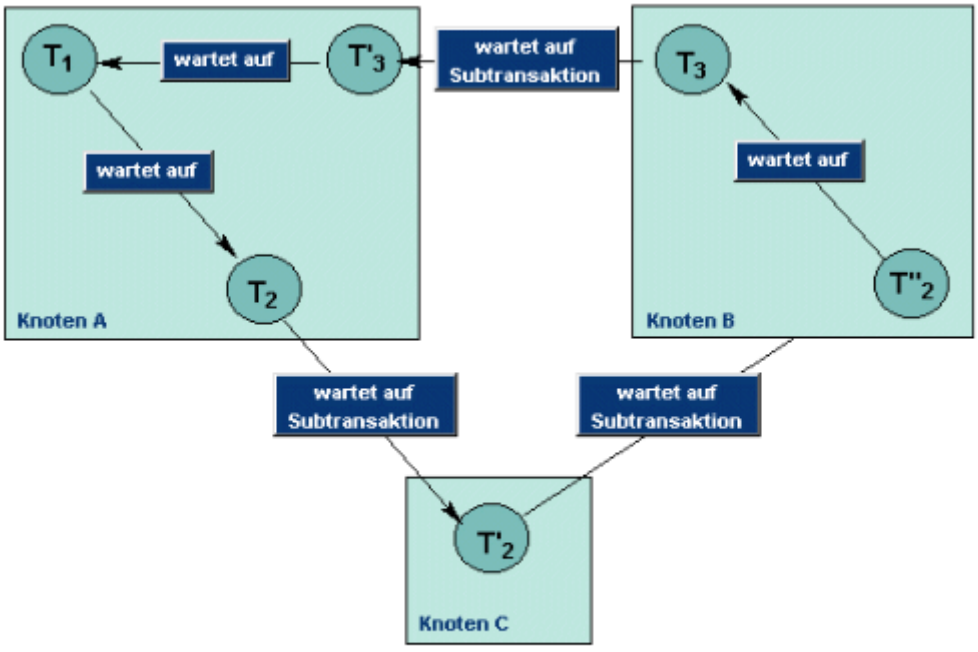



**Abb. 8-7: Gobale Verklemmungen**

# Erkennung von Dead Locks / Verklemmungen

**Zentralisierte Suche nach Dead Locks** – Ein zentraler Knoten übernimmt Analyse, benötigt lokale und globale „warte auf“ Informationen. Entweder zentrale Sperrtabelle oder Informationen an Analyseknoten schicken. Informationen können verdichtet werden auf globale Transaktionen.

**Dezentrale Suche nach Dead Locks** – Jeder Knoten nimmt an der Analyse Teil, Erstellt eigenen „Warte Auf“ Graphen und sucht nach Zyklen.

Jeder Knoten einhält genau einen (globalen) Ein-/Ausgang. Zyklen von EX → EX deuten auf globale Verklemmungen hin.

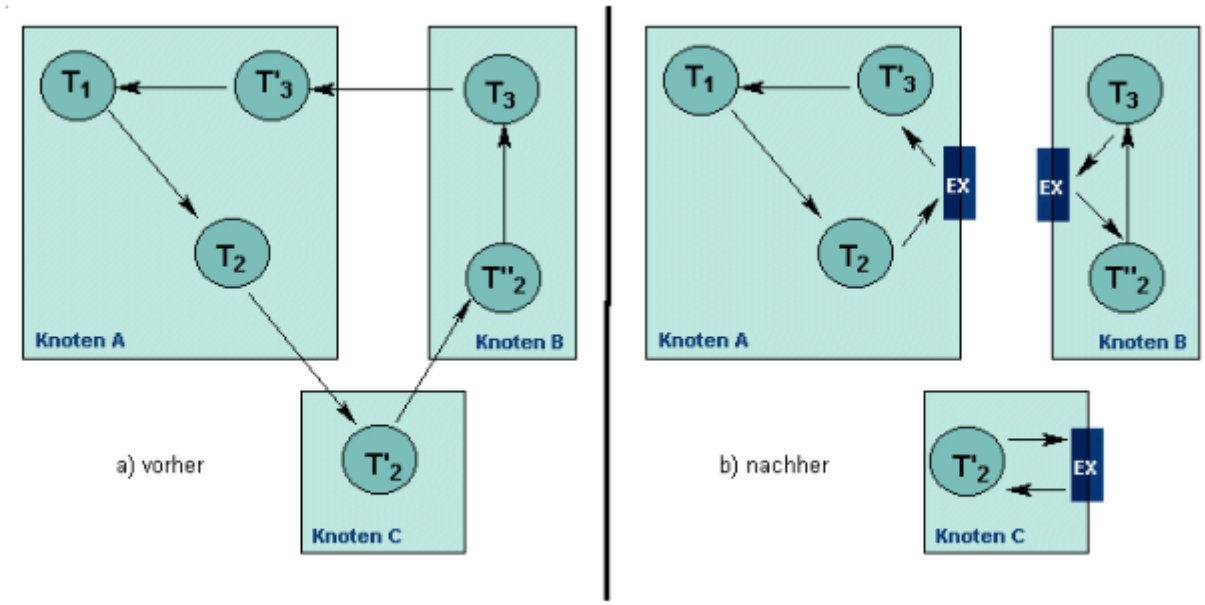



**Abb. 8-8: Lokale WA-Graphen**

**Algorithmus 8-1: "Distributed Deadlock Detection" (DDD)**

1. Konstruiere den lokalen WAG (mit EX-Knoten, wie eben beschrieben).
   Gehe zu Schritt 5.
2. Falls Strings und Deadlock-Information von anderen Knoten eintreffen, so werden sie im lokalen WAG wie folgt berücksichtigt:

   a. Alle Knoten und Kanten von Deadlock-Opfern werden aus dem WAG entfernt (siehe hierzu Anmerkung 3).

   b. Alle Strings, die ein bereits bekanntes Deadlock-Opfer enthalten, werden ignoriert.

   c. Noch nicht bekannte Transaktionen werden eingefügt.

   d. Für die im String enthaltenen Nachfolger-Transaktionen werden im WAG ggf. Pfeile hinzugefügt.

3. Füge für jede Transaktion T im WAG, auf deren Nachricht eine nicht-lokale Transaktion wartet, dem WAG eine EX $\rightarrow$ T - Kante hinzu (sofern noch nicht vorhanden).
4. Füge für jede Transaktion T' im WAG, die auf eine Nachricht einer nicht-lokalen Transaktion wartet, dem WAG eine T' $\rightarrow$ EX - Kante hinzu (sofern noch nicht vorhanden).
5. Analysiere den WAG und erstelle eine Liste aller *elementaren Zyklen* (im folgenden kurz *Zyklenliste* genannt).

   Die nachfolgenden Schritte beziehen sich nur noch auf die *Zyklenliste*:
6. Ermittle alle Zyklen in der Zyklenliste, die *nicht* den Knoten EX enthalten, und wähle daraus jeweils eine Transaktion $T_v$ als Deadlock-Opfer aus.
7. Entferne $T_v$ (falls vorhanden) aus dem WAG (siehe Anmerkung 3) sowie alle Kanten, die von $T_v$ ausgehen oder in $T_v$ einmünden. Entferne alle Strings (und die darauf basierenden Zyklen), die $T_v$ enthalten.
8. Informiere (falls $T_v$ eine globale Transaktion war) die anderen Knoten über das Zurücksetzen von $T_v$.

   In der Zyklenliste sind jetzt nur noch Zyklen mit EX enthalten:
9. Ermittle die Zyklen EX $\rightarrow T_i \rightarrow ... \rightarrow T_j \rightarrow$ EX in der Zyklenliste für die gilt: Zykluslänge > 2 und TransID($T_i$) > TransID($T_j$)

   a. Transformiere diese Zyklen in einen *String* (wie oben beschrieben),

   b. Sende den String zu dem Knoten, auf dessen Nachricht die *letzte* Transaktion im String wartet.

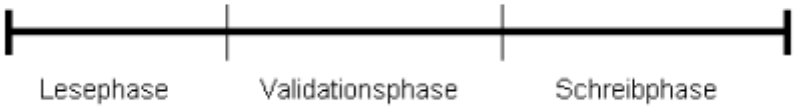
   Gehe zu Schritt 2.

# Alternative: Optimistische Synchronisationsverfahren

**Annahmen**

- Zugriffskonflikte selten sind,
- Transaktionen relativ kurz sind
- eine Wiederholung der Transaktion deshalb im Zweifelsfall billiger ist, als eine Blockierung (wie bei den Sperrverfahren) und eine eventuelle Analyse auf Verklemmungen.

**Aufbau:**




Validationsphase prüft ob Änderungen von anderen Transaktionen durchgeführt wurden. Falls Ja Abbruch. Ansonsten Schreiben

Zentraler Fall: Der erfolgreiche Austritt aus der Validationsphasedefiniert definiert die *äquivalente serielle Ausführungsreihenfolge* der vollständig ausgeführten Transaktionen.

Im *verteilten Fall* muß die Freigabe der lokalen Änderungen, also der Eintritt in die Schreibphase, mit dem Zwei-Phasen-Commit koordiniert werden, sonst besteht die Gefahr nicht-serialisierbarer globaler Schedules. Analog zu den Sperrverfahren muß auch hier sichergestellt werden, daß die Teiltransaktionen keine Änderungen vorzeitig anderen Transaktionen sichtbar machen. Dieses "Sichtbarmachen" geschieht bei optimistischen Verfahren durch Verlassen der Validationsphase, da dann die Schreibphase einsetzt, in der die Änderungen in die Datenbank eingebracht und damit anderen Transaktionen sichtbar gemacht werden.-> globaler Zähler / Zeitstempel

# Replikationsverfahren – Lese-Strategien

Drei Fälle möglich, **Input aktuell und konsistent, Input (etwas) veraltet und konsistent, Input aktuell und (etwas) inkonsistent.**

**Fall 1 aktuell und konsistent** bedeutet das Lese und Update Operationen prinzipiell gleich behandelt werden. (Unterschiede beim Quorum Anzahl Stimme oder unterschiedlichen Sperrmodi) → Nachteil Durchsatz (Performance)

**Fall 2 veraltet aber konsistent (Snap Shot Verfahren)** kombinierbar mit Versionskonzept, wird ein konsistenter Stand gelesen kann aber ältere Version sein. Wir praktisch von allen Replikationsverfahren unterstützt. Lese-Operationen brauchen nicht mit update Transaktionen synchronisiert werden.

**Fall 3 etwas Inkonsistent ist ok** – Differenz wird definiert (Epsilon) in wie weit die Leseoperation Daten lesen darf die abweichen vom aktuellen Datenbankstand. Lesetransaktion gilt als seriealisierbar wenn die Abweichung nicht überschritten wird. **Begriff epsilon Serialisierbarkeit.** Bestimmung der Differenz (Epsilon) in Zeit oder Anzahl Updates. Nur für Anwendungen möglich wo Inkonsistenzen ok sind. Börsenticker, Statistische Life Anwendungen usw.

# Replikationsverfahren – Kopie -Update Strategien

Grund: Hohe Verfügbarkeit der Daten → Redundanz → Verfahren zur Sicherstellung der Konsistenz

Nachteile: Kommunikationsaufwand wird erhöht zwischen den Knoten, Konsistenzhaltung schwieriger

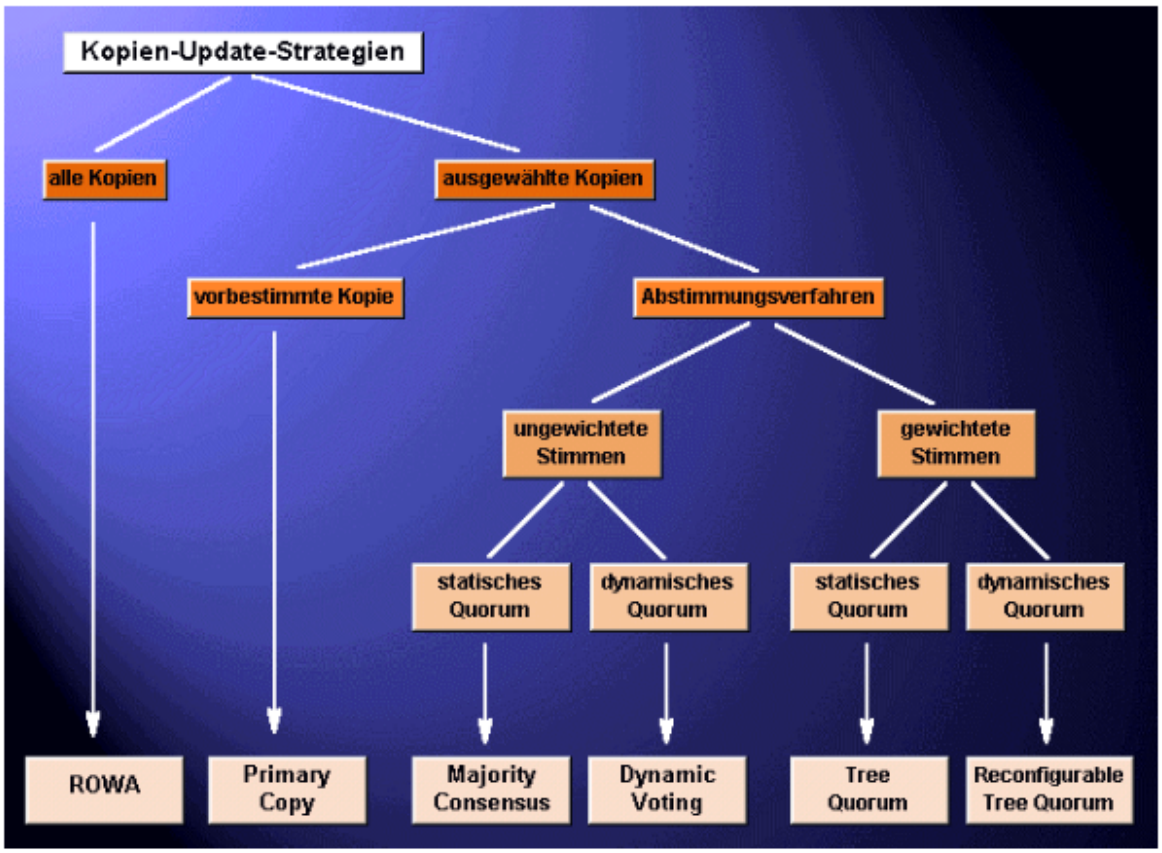



**Abb. 9 -1: Kopien-Update-Strategien im Überblick**

**Update aller Kopien - ROWA-Verfahren:** Lesen von belieber Kopie, Alle Kopien in der gleichen Transaktion geupdated, gut fürs Lesen, sher schlecht beim Update → ein Knoten fehlt komplett kein Update mehr möglich, praktischer Nutzen = 0.

**Update vorbestimmter Kopie - Primary Copy:** Eine bestimmte Kopie ist Master (Originalversion), Updateverfahren zweistufig, erst Sperren der Master Kopie und dann Update der Kopien asynchron, *Problematisch Konsistentes Lesen,* Konsitenz bei bei Ausfall der Masterkopie

**Quorum basiert:** Abstimmung durch Kopien, Anzahl der Stimmen kann geteilt werden z.B. 1/3 Lesen 2/3 Schreiben. Um Serialisierbarkeit zu gewährleisten muss Ql + Qs > Qges. Z.b. Jeweisl 50% + 1 Stimme.

**Dynamisches Quorum** Anzahl der Kopien wird zur Laufzeit bestimmt (Verfügbarkeit der Knoten), erhöhter Kommunikationsaufwand, erhöhter Buchvührungsaufwand.

**Statisches Quorum** wird beim Systemstart festgelegt, Problem beim Ausfall vieler Knoten keine Mehrheit mehr erreichbar

**Ungewichtete vs. gewichtete Stimmen –** Entweder alle gleich oder Hauptkopie mit Gewichtung z.b. Hauptkopie 5 Stimmen + 6 Kopien 1 Stimme . Gesamt 11 Stimmberechtigt, Mehrheit von 6 kann durch Hauptkopie + 1 einzelne oder alle einzelne gewonnen werden. Vorteil weniger Anfragen

## *Replikationsverfahren - Fehlerhandling für Kopien-Update-Strategien*

Unterscheidung **Ausfall einzelner Kopien** / **Netzpartitionierung** durch Trennung der Kommunikationsverbindung

**Ausfall einzelner Knoten Vorbestimmte Kopie (Primary Copy)**
Wird beim Update mit einer vorbestimmten Kopie wie beim Primary-Copy-Verfahren gearbeitet, so ist diese Hauptkopie dafür zuständig, ausgefallene Kopien beim Wiederanlauf mit der aktuellen Version zu versorgen. Fällt die Primärkopie selbst aus, so ist es im Prinzip möglich, eine der anderen Kopien zur Primärkopie zu "küren", wobei unter Umständen allerdings ein gewisser Aktualitätsverlust, je nach Update-Propagations-Strategie der Primärkopie, in Kauf genommen werden muß. Bei der Erzeugung einer neuen Primärkopie muß sichergestellt sein, daß die Primärkopie tatsächlich ausgefallen ist und nicht etwa nur durch eine Netzpartitionierung für einige Knoten unerreichbar geworden ist.,

**Ausfall einzelner Knoten Abstimmungsverfahren**
Bei den Abstimmungsverfahren führt - je nach Verfahren - ohnehin nur ein Teil der Knoten den Update synchron durch, die anderen Knoten werden ggf. asynchron aktualisiert. Der Ausfall einzelner Knoten ist deshalb hinsichtlich der Behandlung der verbliebenen Knoten relativ ähnlich zum Normalfall (verzögerte Aktualisierung), nur daß in diesem Fall die Updateinformation länger aufbewahrt werden muß.

**Netzpartitionierung**

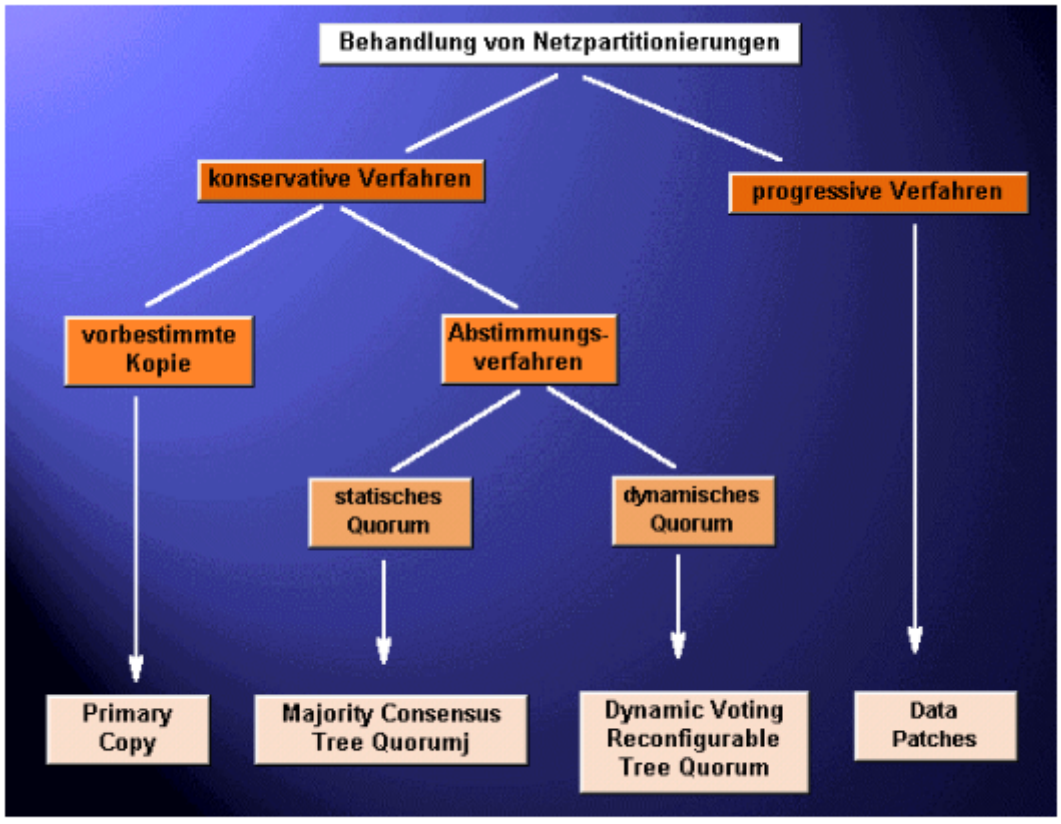



**Abb. 9-2: Verhalten bei Netzpatitionierung**

**Progressive Verfahren -** Verfügbarkeit des Gesamtsystems die höchste Priorität, Inkonsistenzen werden in Kauf genommen, Bei Netzverfügbarkeit Zusammenführung

**Konservative Verfahren** – Konsitenz höchste Prio, Performance Einbusen oder Nicht Verfügbarkeit wird in Kauf genommen, Es wird eine Hauptpartition bestimmt nur dort darf geschrieben werden, Bei Vorbestimmter Master Kopie ist auch die Netzpartion Master, Bei Quorum wird abgestimmt welche Partion der Master ist. Auf die anderen darf nur lesend zugegriffen werden

## *Replikationsverfahren - Synchronisation von Updatetransaktionen*

Das verteilte System soll sich nach außen hin wie ein System ohne Kopien verhalten. Aus Anwendungssicht 1-Kopie äquivalent. Serialisierbarkeit wird um den Begriff 1-Kopie-Serialisierbarkeit erweitert.

**1-Kopie-Serialisierbarkeit -** Eine Schedule S von (abgeschlossenen) Transaktionen, die auf einer repliziert gespeicherten verteilten Datenbank ausgeführt wurden, heißt dann und nur dann *1-Kopie-serialisierbar*, wenn es mindestens eine serielle Ausführung der Transaktionen aus S auf einer verteilten Datenbank ohne Replikate gibt, welche, angewandt auf denselben Ausgangszustand, die gleiche Ausgabe sowie denselben Endzustand erzeugt.

Es existiert eine Schedule von Transaktionen für die auch eine serielle Ausführung mit gleicher Wirkung auf einem VDBMS ohne Replikate möglich ist.

**Konventionelle Verfahren** – Bei ROWA oder Primary Kopie kann jedes konventionelle Sperrverfahren verwendet werden (2PL / Optimistisches Sperrverfahren) um 1 Kopie Serialisierbarkeit zu erhalten.

**Zeitstempel Verfahren –** Bei allen anderen Kopie Update Strategien müssen Zeitstempel oder Versionsnummern verwendet werden da die restlichen Kopien asynchron updated werden. Ziel ist Updateentscheidungen rein lokal zu treffen, falsche Transaktionsabbrüche werden in kauf genommen.

**Einfaches Zeitstempel Verfahren** – Zwei Zeitstempel für Lesen/Schreiben, Zeitstempel der Transaktion. Wird geprüft ob Objekt geändert seit dem Start der Transaktion $Tt < Tw \rightarrow$ Abbruch genauso beim schreiben. Problem dirty reads da Zeitstempel beim schreiben erneuert wird ohne das die komplette Transaktion erfolgreich war. Kaskadierendes Zurücksetzen nötig von Transaktionen die falsch gelesen haben. In der Praxis eher unbrauchbar

**Erweitertes Zeitstempel Verfahren** – Drei Phase ähnlich optimistischen Verfahren Lese-, Sperr-, Schreibphase. Nutzt Queue und exklusive Sperren. Wenn Object gesperrt kann Anfrage gequeued werden.

Zur Read-Aktion

- Der Transaktionszeitstempel fürs Lesen > Schreibzeitstempel des Objekts (analog einfachem Verfahren)
- Ob die Read-Aktion sofort oder verzögert ausgeführt wird, hängt davon ab, ob für dieses Objekt bereits eine Sperranforderung mit niedrigerem Zeitstempel vorliegt. Ist dies der Fall, so wird das Lesen verzögert (und die Leseanforderung in die read_queue eingefügt), andernfalls wird das Lesen sofort ausgeführt.

Zur Lock-Aktion

- Einer Write-Aktion geht stehts eine entsprechende Lock-Aktion voraus.
- Wie der Schreibzeitstempel beim einfachen Verfahren, muß der Transaktionszeitstempel der Lock-Aktion in bezug auf beide Objektzeitstempel aktuell sein.
- Analog zu einer echten Sperre führt die Lock-Aktion keine Objektveränderungen durch, sondern wird lediglich beim Objekt eingetragen. Die lock_queue entspricht damit einer Sperrwarteschlange beim Zwei-Phasen-Sperren.

Zur Write-Aktion

- Da - analog zum exklusiven Sperren - die entsprechende Write-Aktion vom Scheduler stets akzeptiert wird, sofern der entsprechende Lock-Request akzeptiert wurde, geht es hier nur noch darum, ob das Schreiben sofort ausgeführt werden kann oder verzögert werden muß. Das Schreiben wird stets dann verzögert, wenn noch locks oder reads mit kleinerem Zeitstempel vorliegen, die zuvor noch abgearbeitet werden müssen.
- Nach Ausführung der Write-Aktion wird die exklusive Sperre durch Entfernung aus der lock_queue aufgehoben. Hierdurch werden evtl. andere, bislang blockierte Aktionen ausführbar.

**Sematische Verfahren zur Synchronisation von Kopie Updates**

Potentiell höhrer Durchsatz, lediglich Operationen erhöhe /vermindere, kurzeitige Inkonsistenzen durch out of order → Ergebnis aber gleich. Objekte werden sofort freigeben → Rollback (Kompensations) Transaktionen nötig

# Replikationsverfahren – Ausgewählte Verfahren

**Primary Copy –** Ausgewählte Kopie ist das Original, Aktuallisierung von Kopien 2 stufig, Master wird gesperrt, Rest asynchron aktualisiert (Kopien über FIFO Queue angebunden an Master). Vorteil Lesezugriffe auf lokale Kopien möglich, jedoch kein konsistentes Lesen. Lokale Kopie weiß nie wann konsistenter Zustand erreicht ist. Probleme auch bei Ausfall / Netzpartitionierung, Ist die Primärkopie wirklich weg oder nur Netzpartitionierung? Ist die auserwählte Kopie auf dem neusten Stand und konsistent?

**Majority Consensus** – Erstes Abstimmungsverfahren, Ausgangsbasis vollständig replizierte verteilte Datenbank, Vorteil Verfügbarkeit auch bei Knotenausfall, Nachteil: statische Anzahl stimmberechtigter Knoten → Probleme bei Netzpartitionierung

*Verfahren/Vorgehensweise:*

Jede Transaktion ist mit einem global eindeutigen Zeitstempel versehen.

Jedes Datenbankobjekt ist mit einem Zeitstempel versehen, welcher den Zeitpunkt der letzten erfolgreich durchgeführten Änderung (® Commit) wiedergibt.

Die Knoten sind untereinander durch einen logischen Ring verbunden, entlang dessen die Entscheidung vorangetrieben wird. Jeder Knoten ist damit über die Entscheidung seiner Vorgängerknoten informiert.

*Jede Updatetransaktion*

- führt alle Änderungen zunächst rein lokal durch, macht diese aber anderen Transaktionen noch nicht sichtbar
- erstellt eine Liste aller Ein- und Ausgabeobjekte mit den jeweiligen Zeitstempeln
- schickt diese Liste zusammen mit ihrem Transaktionszeitstempel entlang des Rings an alle anderen Knoten
- darf die Änderungen permanent machen, wenn die Mehrheit der Knoten (mind. also n ¸ 2 + 1) dem Update zustimmt.

*Abstimmung:*Jeder Knoten stimmt über eingehende Änderungsaufträge wie folgt ab und reicht sein Votum zusammen mit den anderen Voten an den nächsten Knoten weiter:

- Er stimmt mit ABGELEHNT, wenn einer der übermittelten Objektzeitstempel veraltet ist.
- Er stimmt mit OK und markiert den Auftrag als schwebend (pending), wenn alle übermittelten Objektzeitstempel aktuell sind und der Auftrag nicht in Konflikt mit einem anderen Auftrag steht.
- Er stimmt mit PASSIERE, wenn alle Objektzeitstempel zwar aktuell sind, der Antrag aber mit einem anderen schwebenden Antrag mit höherem Zeitstempel in Konflikt steht. Falls durch PASSIERE keine Mehrheit mehr zustandekommen kann, so stimmt er mit ABGELEHNT.
- Er verzögert seine Abstimmung über den Antrag, wenn der Antrag in Konflikt mit einem Antrag mit niedrigerem Zeitstempel steht oder wenn einer der übermittelten Objektzeitstempel einen aktuelleren Wert als das korrespondierende lokale Objekt aufweist.

Annahme / Ablehnung:

- Der Knoten, dessen Zustimmung (OK) dem Antrag die Mehrheit verschafft hat, erzeugt die globale Commit-Meldung für diese Updatetransaktion.
- Jeder Knoten, der mit ABGELEHNT stimmt, löst ein globales Abort dieser Updatetransaktion aus.
- Wird eine Updatetransaktion abgelehnt, so werden die "verzögerten Abstimmungen" je Knoten daraufhin überprüft, ob jetzt eine Entscheidung möglich ist.
- Bei Ablehnung muß die Transaktion komplett wiederholt werden, einschließlich des Lesens der Objekte.

*Anmerkungen:*

- Lesetransaktionen müssen bei diesem Verfahren für konsistentes Lesen wie (Pseudo-)Updatetransaktionen behandelt werden.
- Das Verfahren läßt zu, daß sich genehmigte Updates unterwegs "überholen". Beim Eintreffen eines Updates werden deshalb zunächst die Objektzeitstempel gelesen und veraltete Updates ggf. ignoriert.
- Die Knoten dürfen ein einmal getroffenes Votum nicht mehr ändern.
- Die Regeln 5c und 5d dienen dazu, mögliche Verklemmungen zu vermeiden:

**Dynamic Voting**

Vorteil: Quorumgröße wird dynamisch an Verfügbare Knoten angeasst, Es handelt sich also um ein *Majority-Consensus-Verfahren mit dynamischer Quorumeinteilung*. Die *Kernidee* dieses Ansatzes ist, daß nur noch diejenigen Knoten ein Stimmrecht besitzen, die beim letzten Update mit beteiligt waren.

Es werden Verwaltungsinfos benötigt (Versionsnummer und am letzten Update beteiligte Knoten), Es wird immer Versionsnummer und SK (Anzahl Knoten seit dem letzten Update). Aktuelle Teilnehmer ergibt sich aus dem SK Wert des Knoten → Immer der größere Teil des Netztes hat Recht und darf arbeiten → Der andere hat durch die dynamische Anzahl Stimmberechtigter keine Handlungsmöglichkeit. Bei Auflösung werden die neuen Knoten aktualisiert.

Immer nur Knoten mit gleicher Versionsnummer sind Stimmberechtigt, Knoten kann sich selbst aktualisieren.

# Replikationsverfahren – Ausgewählte Verfahren

**Tree Quorum –** Versuch Anzahl der Nachrichten zu reduzieren, Quorumbildung entlang logischer (Baum) Struktur, Mehrheit auf Ebenen bezogen reicht, Innhalb einer Ebene durch Abstimmung,

Das jeweilige Lese- und Schreibquorum bei diesem Protokoll ist also durch zwei Parameter Q(e,a) definiert:

- die Anzahl der erforderlichen Ebenen (e) und
- die Anzahl (a) von erforderlichen Zustimmungen je Teilebene (Teilbaum).

e und a sind abhängig von der Höhe (h) und dem Verzweigungsgrad (v) des Abstimmungsbaumes.

Die Parameter von Schreibquorum $Q_w(e_w,a_w)$ und Lesequorum $Q_r(e_r,a_r)$ müssen den folgenden Nebenbedingungen genügen:

$Q_w$: $2 * e_w > h$, $2 * a_w > v$ (Behandlung Schreib-Schreib-Konflikte)

$Q_r$: $e_r + e_w > h$, $a_r + a_w > v$ (Behandlung Lese-Schreib-Konflikte)

Mehrheit:

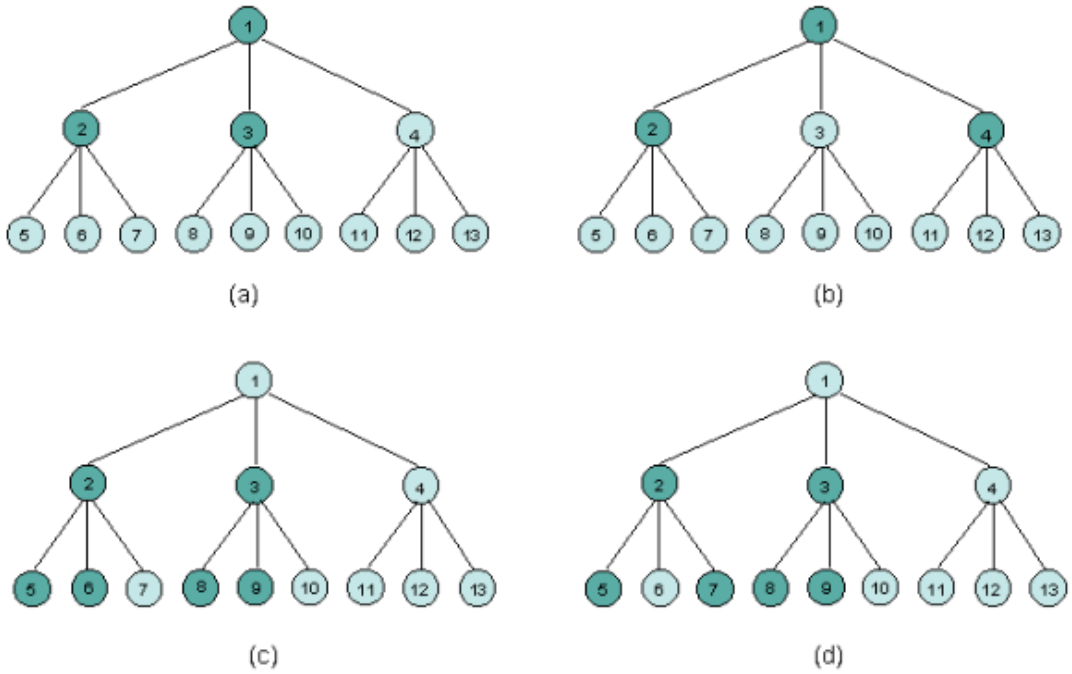



**Abb. 9-7: Auswahl möglicher Konstellationen für ein Schreibquorum**

Vorteil weniger Nachrichten insgesamt, hohe Last bei Knoten oben im Abstimmungsbaum

**Reconfigurable Tree Quorum –** Analog Tree Quorum jedoch mit möglichkeit der Rekonfiiguration

**Data Patches** – Verfügbarkeit des Gesamtsystems inkl Updates höchste Priorität, Konsistentzprobleme werden in Kauf genommen, Nach Wiedervereinigung müssen diese Probleme gelöst werden.

Es wird unterschieden zwischen Einfügen von Tupeln und Ändern von Tupeln während einer Netzpartitionierung. → Tupel Einfügeregelen (KEEP einfügen, REMOVE wieder entfernen, PROGAM Logik, NOTIFY Admin anrufen) → Tupel Integrationsregel ( LATEST, PRIMARY k Master k hat das Wort, ARITHMETIK Mittelwert, PROGRAM, NOTIFY)

Vorgehen bei Wiedervereinigung:

1. Bestimme alle Tupel, die nur in einer Netzpartition eingefügt wurden:
   ⇒ wende *Tupel-Einfügeregel* an
2. Bestimme alle Tupel, die in beiden Partitionen vorkommen und geändert wurden:
   ⇒ wende *Tupel-Integrationsregel* an.

# Recovery Strategien

**Kurzzeit Recovery (Crash Recovery) –** Für das Einhalten der Atomarität von Transaktionen, nur vollständig ausgeführte Transaktionen dürfen auf die Datenbank. Recovery bei Dead Lock, Systemabbruch. Lösung Logfiles mit BOT Zeitstempel, Ende Zeitstempel (Commit Abort), Before und After Image Einträge (Änderungen der Daten für Rollback oder Redo). Im verteilten fall Ready to commit Zustand inkl. gesperrter Objekte. Atomarität bei Einhaltung des 2Phasen Commi gegeben

**Langzeit Recovery (Media Recovery) –** Bei Systemabstürzen / Hardwaredefekten., Redundante Hardware, Fallback Cluster, Sicherungen an Sicherungspunkten stellen veraltete aber konsitente Datensätze da.

**Strikt synchronisierte lokale Sicherungspunkte** – alle Knoten zusammen backup, gemeinsame Wiederherstellung. Problem hohe Last während Backup

**Lose synchronisierte Sicherungspunkte** – unterscheidung lokaler Sicherungspunkte → globaler sicherungspunkt, Lokale Sicherungspunkte treiggern globalen Sicherungspunkt an. Knoten sichern sobald sie bereit sind

**Nicht synchronisierte lokale Sicherungspunkte –** Globale Zusammenhänge werden nicht betrachtet, Verschiebt Aufwand auf globale Recovery Phase, Wird davon ausgegangen das festgestellt werden kann welche lokalen Transaktionen seit dem Backup fehlen und daher Global zurückgewickelt werden müssen.

# Client/Server Anwendungen

Prinziep der Softwarearchitektur

Schichten PRÄSENTATION | APPLIKATION | DATENVERWALTUNG

Je Nach dem wo der Schnittgesetzt wird von dünnem Client (X-Window Server) Bis reiner Fileserver

Optimierung – Reduzierung entfernter DB Zugriffe durch Trigger / Stored Procedures usw

Vorgehen

- Die Initiative zu einer Interaktion geht stets vom Klienten (*Client*) aus.
- Der Client formuliert Aufträge und schickt diese zur Ausführung an einen Dienstanbieter (*Server*).
- Für jeden Server ist festgelegt bzw. bekannt, welche Arten von Diensten (*Services*) er anbietet.

## Client Server Basis Technologien

- Middleware
- RPC
- Message Broker
- Entfernte Datenbankaufrufe

## Normalisierung

Ziel **Redundanzvermeidung.**

Eine Relation ist in *erster Normalform* genau dann, wenn alle ihre Attributwerte atomar sind

*zweite Normalform* - Jedes nicht-primäre Attribut (nicht Teil eines Schlüssels) ist jeweils von allen ganzen Schlüsseln abhängig, nicht nur von einem Teil eines Schlüssels. Wichtig ist hierbei, dass die Nichtschlüsselattribute wirklich von allen Schlüsseln vollständig abhängen

*Die dritte Normalform* ist erreicht, wenn sich das Relationenschema in 2NF befindet, und kein Nichtschlüsselattribut (hellgraue Zellen in der Tabelle) von einem anderen Nichtschlüsselattribut funktional abhängig ist.

*Boyce-Codd-Normalform* - Eine Relation ist in BCNF, wenn jede Determinante ein Superschlüssel ist. Ein Schlüsselkandidat ist eine Menge von Attributen, von der alle Attribute der Relation voll funktional abhängig sind. Ein Superschlüssel ist eine Menge von Attributen, deren Teilmenge ein Schlüsselkandidat ist.