

---

# **Zusammenfassung DDM**

Jurij Maïkoff

2019-06-11

## Inhaltsverzeichnis

<b>1 Sie können verteilte Datenbank systeme entlang der Dimension HEterogenität, Auto- nomie und Verteilung klassifizieren</b>	<b>5</b>
<b>2 Sie kennen die 12 Regeln für verteilte Datenbanksysteme von Chris Date</b>	<b>5</b>
<b>3 Sie können den Begriff Transparenz in verteilten Datenbanken erklären.</b>	<b>5</b>
<b>4 Sie können mit mindestens 2 verschiedenen Methoden Ortstransparenz mit einer ver- teilten Oracle Datenbank realisieren.</b>	<b>6</b>
<b>5 Sie können das Entwurfsproblem für verteilte Datenbanken erläutern und kennen 2 Entwurfsstrategien.</b>	<b>6</b>
<b>6 Sie können eine primäre horizontale Fragmentierung (PHF) durchführen</b>	<b>8</b>
6.1 Simple predicate p . . . . .	8
<b>7 Sie können eine vertikale Fragmentierung durchführen.</b>	<b>9</b>
<b>8 Sie kennen die Korrektheitsregeln für eine Fragmentierung und können sie auf ein Bei- spiel anwenden.</b>	<b>9</b>
<b>9 Sie können die verschiedenen Phasen einer verteilten Anfrageverarbeitung beschreiben.</b>	<b>10</b>
9.1 Zerlegung . . . . .	10
9.2 Lokalisierung . . . . .	11
9.3 Globale Optimierung . . . . .	11
9.4 Lokale Optimierung . . . . .	11
<b>10 Sie können eine globale Anfrage in eine lokalisierte und reduzierte Anfrage überführen.</b>	<b>12</b>
<b>11 Sie verstehen den Begriff Collocated Inline View und können ihn anhand eines Beispiels erklären.</b>	<b>13</b>
<b>12 Sie kennen den Semijoin und können anhand eines Beispiels erklären, wie damit die Performance eines Joins in einer verteilten Datenbank verbessert werden kann.</b>	<b>14</b>
<b>13 Sie kennen die Aspekte der Transaktionsverarbeitung in verteilten Datenbanksystemen.</b>	<b>15</b>
<b>14 Sie verstehen die Erweiterung des Begriffs Serialisierbarkeit in einem verteilten Daten- banksystem</b>	<b>16</b>

<b>15 Sie können 3 verschiedene Varianten der Realisierung des 2-Phasen-Sperrprotokolls in verteilten Datenbanksystemen erklären.</b>	<b>16</b>
15.1 Zentral . . . . .	17
15.2 Primary . . . . .	17
15.3 Verteilt . . . . .	18
<b>16 Sie können das Problem der Erkennung von verteilten Deadlocks beschreiben.</b>	<b>18</b>
<b>17 Sie kennen die Komponenten zur Sicherstellung von Atomicity und Durability verteilter Transaktionen.</b>	<b>19</b>
17.1 Komponenten . . . . .	19
<b>18 Sie können das 2-Phasen-Commit Protokoll beschreiben und kennen 3 verschiedene Kommunikationsvarianten.</b>	<b>20</b>
<b>19 Sie können die 2 Hauptprobleme des 2PC Protokolls erläutern.</b>	<b>22</b>
<b>20 Sie kennen die Vor- und Nachteile von replizierten Daten.</b>	<b>23</b>
<b>21 Sie kennen verschiedenen Konsistenzmodelle für die Replikation.</b>	<b>23</b>
21.1 Strong Consistency . . . . .	23
21.2 Weak Consistency . . . . .	23
21.3 Client-Centric Consistency . . . . .	24
<b>22 Sie können die verschiedenen Update Propagation Strategien beschreiben.</b>	<b>24</b>
22.1 Eager / Lazy Replication . . . . .	24
22.2 Master / Group Replication . . . . .	24
22.3 Eager Master (Synchronous Primary Copy) Replication . . . . .	25
22.4 Eager Group (Synchronous Update Everywhere) Replication . . . . .	26
22.5 Lazy Master (Asynchronous Primary Copy) . . . . .	27
22.6 Lazy Group (Asynchronous Update Everywhere) Replication . . . . .	28
<b>23 Sie können erläutern, wie in Oracle 12c Replikation mit Materialized Views realisiert wird und welcher Replikations Strategie dies entspricht.</b>	<b>31</b>
<b>24 Sie kennen die Kategorisierung der NoSQL Systeme mit den typischen Vertretern.</b>	<b>31</b>
<b>25 Sie können das CAP-Theorem erläutern.</b>	<b>31</b>
<b>26 Sie kennen das Map/Reduce Verfahren und können dafür typische Anwendungen programmieren.</b>	<b>31</b>

<b>27 Sie kennen die Modellierungsprinzipien für Cassandra und können diese anwenden.</b>	<b>31</b>
<b>28 Sie können mit CQL Datendefinitionen realisieren und Anfragen formulieren.</b>	<b>31</b>
<b>29 Sie können Datenmodelle für MongoDB entwerfen und kennen die verschiedenen Darstellungsmöglichkeiten von Beziehungen.</b>	<b>31</b>
<b>30 Sie können mit der Mongo Shell Datendefinitionen realisieren und Anfragen formulieren.</b>	<b>31</b>
<b>31 Sie kennen das Graphdatenmodell von Neo4j.</b>	<b>31</b>
<b>32 Sie können mit der Sprache Cypher Datendefinitionen realisieren und Anfragen formulieren.</b>	<b>31</b>
<b>33 Sie können das Datenmodell des Resource Description Frameworks anhand eines Beispiels erläutern.</b>	<b>31</b>

## 1 Sie können verteilte Datenbank systeme entlang der Dimension Heterogenität, Autonomie und Verteilung klassifizieren

- **Heterogenität:** Hardware, Netzwerkprotokolle, Datenverwaltung, Datenmodell, Abfragesprache, Transaktionsverwaltung
  - 2 Ausprägungen: homogen, heterogen
- **Verteilung:** betrifft die Verteilung der Daten
  - 3 Ausprägungen: zentral, client/server, verteilt
- **Autonomie:** betrifft die Verteilung der Steuerung
  - 3 Ausprägungen: stark integriert, halbautonom, isoliert

## 2 Sie kennen die 12 Regeln für verteilte Datenbanksysteme von Chris Date

1. Lokale Autonomie
2. Unabhängigkeit von zentralen Systemfunktionen
3. Hohe Verfügbarkeit
4. Ortstransparenz
5. Fragmentierungstransparenz
6. Replikationstransparenz
7. Verteilte Anfragebearbeitung
8. Verteilte Transaktionsverarbeitung
9. Hardware Unabhängigkeit
10. Betriebssystem Unabhängigkeit
11. Netzwerkunabhängigkeit
12. Datenbanksystem Unabhängigkeit

## 3 Sie können den Begriff Transparenz in verteilten Datenbanken erklären.

- **Ortstransparenz** (Positionstransparenz) Der Ort, an dem sich ein Dienst oder eine Ressource befindet ist dem Benutzer nicht bekannt. Der Zugriff erfolgt über einen bestimmten Namen, der allerdings keine Ortsinformationen enthält.

- **Zugriffstransparenz** Der Zugriff auf die Ressource erfolgt immer auf die gleiche Art und Weise, egal ob diese sich lokal oder entfernt im Netz befindet.

#### 4 Sie können mit mindestens 2 verschiedenen Methoden Ortstransparenz mit einer verteilten Oracle Datenbank realisieren.

- **Replikationstransparenz** Aus Performancegründen kann es mehrere Kopien derselben Ressource geben. Das System sorgt für die transparente Replikation der darin vorgenommenen Änderungen.
- **Fragmentierungstransparenz** Die Teilbestandteile einer Ressource können an verschiedenen Orten gespeichert sein.

```
1 -- Ortstransparenz
2 -- wird benötigt, dass bei select statement nicht immer der ganze link
  angezeigt werden muss
3 -- so muss der benutzer nicht wissen wo die daten genau liegen, hö
  chstens dass sie an einem anderen ort liegen
4 CREATE OR REPLACE VIEW filme
5 AS SELECT * FROM filme@ganymed.sirius.fhnw.ch;
6 DROP VIEW filme;
7
8 -- Namenstransparenz
9 CREATE SYNONYM filme FOR filme@ganymed.sirius.fhnw.ch;
10
11 --link zu anderem Datenbankserver
12 CREATE DATABASE LINK ganymed.sirius.fhnw.ch
13 CONNECT TO ddm61 IDENTIFIED BY ddm61
14 USING 'ganymed';
```

#### 5 Sie können das Entwurfsproblem für verteilte Datenbanken erläutern und kennen 2 Entwurfsstrategien.

- Entscheid über die Platzierung von Daten auf den Knoten eines Computernetzwerks
- beeinflusst die Performance der DDB und der Anwendungen
- lokaler Zugriff ist günstiger als Zugriff auf entfernte Knoten
- Analyse:
  - welche Anwendungen (Queries)

- auf welchen Knoten
  - benötigen welche Daten
  - mit welcher Häufigkeit
- Resultat:
  - Menge von Fragmenten (Ausschnitte der Daten)
  - zugeteilt auf verschiedene Knoten

### Top-down Entwurf

- beim Entwurf „from scratch“
- in homogenen Systemen
- nachgelagert an den konzeptionellen Entwurf

### Bottom-up Entwurf

- Multidatenbank Anwendungen
- Datenbanken schon auf verschiedenen Knoten vorhanden
- Problem der Datenintegration, Schemaintegration
- Web Services

**Abbildung 1:** Entwurfsproblem - Lösung

## 6 Sie können eine primäre horizontale Fragmentierung (PHF) durchführen

<p><b>BIKES</b></p> <table> <tr> <th>BNr</th><th>BName</th><th>Preis</th><th>Typ</th><th>Bestand</th></tr> <tr><td>B5</td><td>MCD03</td><td>4490.00</td><td>Road</td><td>2</td></tr> <tr><td>B4</td><td>Siena</td><td>2390.00</td><td>Mountain</td><td>4</td></tr> <tr><td>B2</td><td>City Cross</td><td>2190.00</td><td>Trekking</td><td>3</td></tr> <tr><td>B3</td><td>Valiant</td><td>1090.00</td><td>Trekking</td><td>7</td></tr> <tr><td>B1</td><td>Luxor</td><td>980.00</td><td>City</td><td>10</td></tr> <tr><td>B6</td><td>Atlanta</td><td>890.00</td><td>Trekking</td><td>8</td></tr> <tr><td>B7</td><td>Striker</td><td>890.00</td><td>Mountain</td><td>7</td></tr> </table>					BNr	BName	Preis	Typ	Bestand	B5	MCD03	4490.00	Road	2	B4	Siena	2390.00	Mountain	4	B2	City Cross	2190.00	Trekking	3	B3	Valiant	1090.00	Trekking	7	B1	Luxor	980.00	City	10	B6	Atlanta	890.00	Trekking	8	B7	Striker	890.00	Mountain	7
BNr	BName	Preis	Typ	Bestand																																								
B5	MCD03	4490.00	Road	2																																								
B4	Siena	2390.00	Mountain	4																																								
B2	City Cross	2190.00	Trekking	3																																								
B3	Valiant	1090.00	Trekking	7																																								
B1	Luxor	980.00	City	10																																								
B6	Atlanta	890.00	Trekking	8																																								
B7	Striker	890.00	Mountain	7																																								
<p>SELECT * FROM bikes WHERE preis &lt; 2000</p> <p><b>BIKES1</b></p> <table> <tr> <th>BNr</th><th>BName</th><th>Preis</th><th>Typ</th><th>Bestand</th></tr> <tr><td>B3</td><td>Valiant</td><td>1090.00</td><td>Trekking</td><td>7</td></tr> <tr><td>B1</td><td>Luxor</td><td>980.00</td><td>City</td><td>10</td></tr> <tr><td>B6</td><td>Atlanta</td><td>890.00</td><td>Trekking</td><td>8</td></tr> <tr><td>B7</td><td>Striker</td><td>890.00</td><td>Mountain</td><td>7</td></tr> </table>					BNr	BName	Preis	Typ	Bestand	B3	Valiant	1090.00	Trekking	7	B1	Luxor	980.00	City	10	B6	Atlanta	890.00	Trekking	8	B7	Striker	890.00	Mountain	7															
BNr	BName	Preis	Typ	Bestand																																								
B3	Valiant	1090.00	Trekking	7																																								
B1	Luxor	980.00	City	10																																								
B6	Atlanta	890.00	Trekking	8																																								
B7	Striker	890.00	Mountain	7																																								
<p>SELECT * FROM bikes WHERE preis &gt;= 2000</p> <p><b>BIKES2</b></p> <table> <tr> <th>BNr</th><th>BName</th><th>Preis</th><th>Typ</th><th>Bestand</th></tr> <tr><td>B5</td><td>MCD03</td><td>4490.00</td><td>Road</td><td>2</td></tr> <tr><td>B4</td><td>Siena</td><td>2390.00</td><td>Mountain</td><td>4</td></tr> <tr><td>B2</td><td>City Cross</td><td>2190.00</td><td>Trekking</td><td>3</td></tr> </table>					BNr	BName	Preis	Typ	Bestand	B5	MCD03	4490.00	Road	2	B4	Siena	2390.00	Mountain	4	B2	City Cross	2190.00	Trekking	3																				
BNr	BName	Preis	Typ	Bestand																																								
B5	MCD03	4490.00	Road	2																																								
B4	Siena	2390.00	Mountain	4																																								
B2	City Cross	2190.00	Trekking	3																																								

**Abbildung 2:** Horizontale Fragmentierung

### 6.1 Simple predicate p

Vergleich von einem Attribut mit einem Wert - bname = „Siena“, preis < 2000



## 7 Sie können eine vertikale Fragmentierung durchführen.

BIKES				
BNr	BName	Preis	Typ	Bestand
B5	MCD03	4490.00	Road	2
B4	Siena	2390.00	Mountain	4
B2	City Cross	2190.00	Trekking	3
B3	Valiant	1090.00	Trekking	7
B1	Luxor	980.00	City	10
B6	Atlanta	890.00	Trekking	8
B7	Striker	890.00	Mountain	7

```
SELECT bnr, bname, preis
FROM bikes
```

BIKES1		
BNr	BName	Preis
B5	MCD03	4490.00
B4	Siena	2390.00
B2	City Cross	2190.00
B3	Valiant	1090.00
B1	Luxor	980.00
B6	Atlanta	890.00
B7	Striker	890.00

```
SELECT bnr, typ, bestand
FROM bikes
```

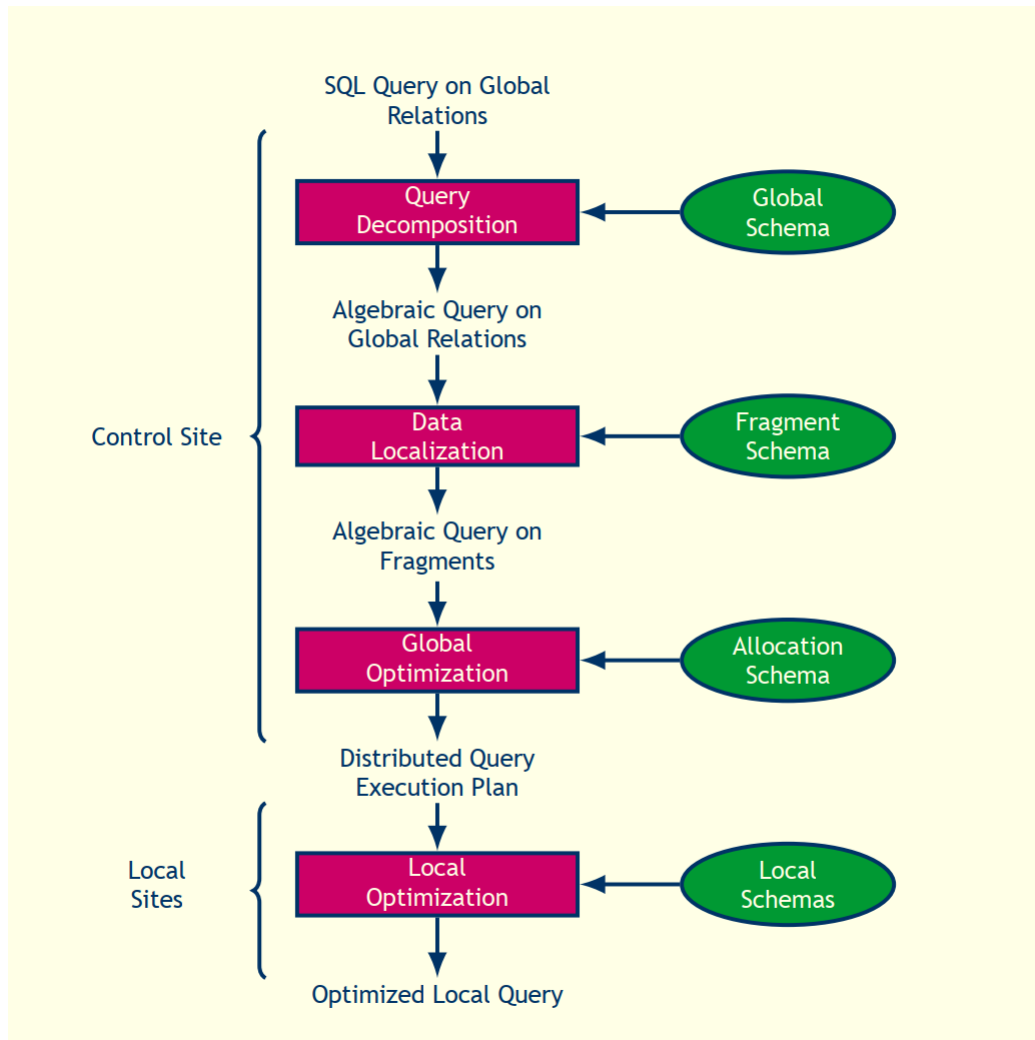
BIKES2		
BNr	Typ	Bestand
B5	Road	2
B4	Mountain	4
B2	Trekking	3
B3	Trekking	7
B1	City	10
B6	Trekking	8
B7	Mountain	7

Abbildung 3: Vertikale Fragmentierung

## 8 Sie kennen die Korrektheitsregeln für eine Fragmentierung und können sie auf ein Beispiel anwenden.

- **vollständigkeit**
  - wenn R zerlegt wird in R1, R2, . . . , Rn, dann muss jedes Datenelement aus R in einem Ri enthalten sein.
- **rekonstruierbar**
  - wenn R zerlegt wird in R1, R2, . . . , Rn, dann muss es relationale Operatoren geben, so dass R wiederhergestellt werden kann.
- **disjunkt**
  - wenn R horizontal zerlegt wird in R1, R2, . . . , Rn, dann müssen die Fragmente paarweise disjunkt sein. wenn R vertikal zerlegt wird in R1, R2, . . . , Rn, dann müssen die Fragmente bezogen auf die nichtprimen Attribute paarweise disjunkt sein.

## 9 Sie können die verschiedenen Phasen einer verteilten Anfrageverarbeitung beschreiben.



**Abbildung 4:** Phasen der Anfragebearbeitung

### 9.1 Zerlegung

SQL Query zerlegen und umformen in relationale Algebra unter Verwendung des globalen Schemas

- Normalisierung (Bedingung in WHERE Klausel) - Analyse, um inkorrekte Queries zurückzuweisen
- Analyse nach Typ - Analyse nach Semantik - Vereinfachung, Redundanz beseitigen
- Umformen in optimalen Ausdruck der relationalen Algebra

## 9.2 Lokalisierung

- verwenden des Fragmentierungsschema
- Verteilte Anfrage mit globalen Relationen abbilden in Anfragen mit Fragmenten
  - Ersetzen der globalen Relation mit den Fragmenten
  - U für horizontale Fragmentierung
  - (kravatten-symbol) für vertikale Fragmentierung
- Optimieren der lokalisierten Anfrage durch Reduktion
  - Reduktion mit Selektion
  - Reduktion mit Join

## 9.3 Globale Optimierung

finde den besten globalen Ausführungsplan - Kostenfunktion minimieren - verteilte Join Verarbeitung - Bushy versus Left Deep (Linear) Tree - Welche Relation wohin transferieren - Vollständig oder nach Bedarf transferieren - Einsatz des SemiJoins - Übertragungskosten sparen aber mehr lokale Verarbeitung - Join Methoden - Nested Loop versus Hash Join versus ...

## 9.4 Lokale Optimierung

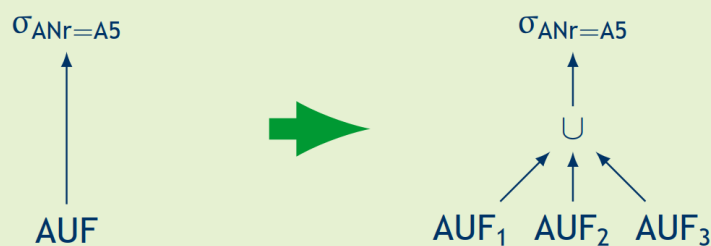
Aus der globalen Optimierung steht der beste globale Ausführungsplan zur Verfügung Jeder Knoten muss für sich - den besten Zugriffspfad bestimmen - Optimierungstechniken verwenden wie in zentralisierten Systemen

## 10 Sie können eine globale Anfrage in eine lokalisierte und reduzierte Anfrage überführen.

### Reduktion mit Selektion für PHF

- Schritt 1: globaler Query Baum erstellen
- Schritt 2: globale Relation mit Fragmenten ersetzen

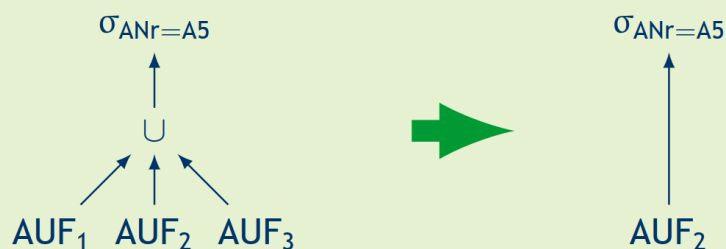
#### Beispiel



- Schritt 3: Reduktion

$ANr = A5$  widerspricht  $ANr \leq A3$  (Fragment  $AUF_1$ ) und  $ANr > A6$  (Fragment  $AUF_3$ )

#### Beispiel



## **11 Sie verstehen den Begriff Collocated Inline View und können ihn anhand eines Beispiels erklären.**

- Ziel ist es, auf einen entfernten Knoten so wenig wie möglich zuzugreifen und nur die nötigen Daten zu verlangen. Mit Hilfe der Collocated Inline Views wird dafür gesorgt, dass auf einen entfernten Knoten nur einmal zugegriffen wird.
  - Collocated: Zwei oder mehr Tabellen sind auf dem gleichen Knoten
  - Inline View: Ein Subquery anstelle einer Tabelle in der FROM-Klausel
  - Collocated Inline View: Das sind Inline Views, die nur auf Tabellen einer einzigen Datenbank zugreifen

## 12 Sie kennen den Semijoin und können anhand eines Beispiels erklären, wie damit die Performance eines Joins in einer verteilten Datenbank verbessert werden kann.

### Definition (SemiJoin)

$X$  sei das gemeinsame Attribut von  $R$  und  $S$ ,  $\text{Attr}(R)$  bezeichnet alle Attribute von  $R$

$$R \triangleright S = \pi_{\text{Attr}(R)}(R \bowtie S)$$

$$R \triangleright S = R \bowtie (\pi_X(S))$$

In SQL:

```
SELECT *  
FROM r  
WHERE EXISTS  
  (SELECT 1  
   FROM s  
   WHERE r.x = s.x)
```

### weitere Eigenschaften

- $R \triangleright S \neq S \triangleright R$

Folgende Alternativen stehen zur Wahl, je nach Kosten

- $R \bowtie S = (R \triangleright S) \bowtie S$
- $R \bowtie S = R \bowtie (S \triangleright R)$
- $R \bowtie S = (R \triangleright S) \bowtie (S \triangleright R)$

## SemiJoin vermindert die zu übertragenden Relationen

sei  $R$  auf Knoten 1,  $S$  auf Knoten 2 und  $\text{size}(R) < \text{size}(S)$

- regulärer Join
  - $R$  nach Knoten 2
  - Knoten 2 berechnet  $R \bowtie S$
- SemiJoin  $(R \triangleright S) \bowtie S$ 
  - Knoten 2 berechnet  $S' = \pi_X(S)$
  - $S'$  nach Knoten 1
  - Knoten 1 berechnet  $R' = R \bowtie S'$  (dh.  $R \triangleright S$ )
  - $R'$  nach Knoten 2
  - Knoten 2 berechnet  $R' \bowtie S$

## 13 Sie kennen die Aspekte der Transaktionsverarbeitung in verteilten Datenbanksystemen.

- Transaktionsstruktur (Transaktionsmodell)
  - fache Transaktion, verschachtelte Transaktion
- Konsistenzerhaltung der Datenbank
- Zuverlässigkeitsprotokolle
  - Unteilbarkeit, Dauerhaftigkeit
  - lokale Wiederherstellung
  - globale Commit Protokolle
- Nebenläufigkeitskontrolle
  - Ausführung nebenläufiger Transaktionen (Korrektheit?)
  - Konsistenzerhaltung zwischen Transaktionen, Isolation
- Kontrolle der Replikate
  - Kontrolle der gegenseitigen Konsistenz von Replikaten
  - One-Copy Equivalence, ROWA

## **14 Sie verstehen die Erweiterung des Begriffs Serialisierbarkeit in einem verteilten Datenbanksystem**

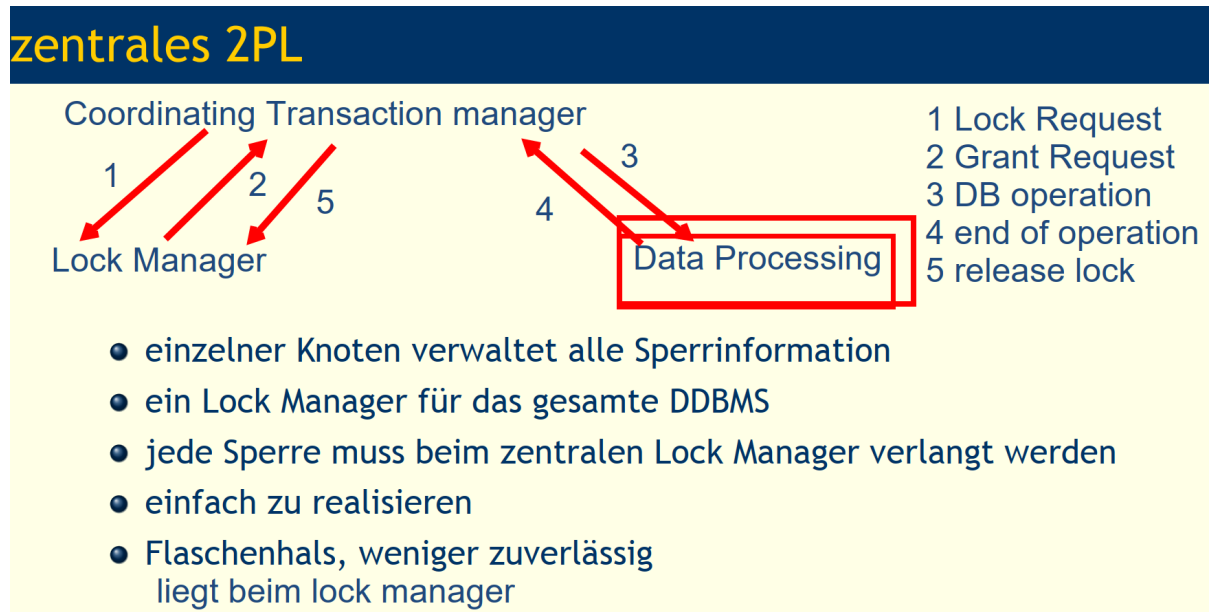
für Serialisierbarkeit des globalen Ablaufplans sind zwei Bedingungen nötig - jeder lokale Ablaufplan muss serialisierbar sein - zwei Konflikt Operationen müssen in der gleichen Reihenfolge auftreten in allen lokalen Ablaufplänen, in denen sie zusammen auftreten

## **15 Sie können 3 verschiedene Varianten der Realisierung des 2-Phasen-Sperrprotokolls in verteilten Datenbanksystemen erklären.**

- basierend auf 2 Phasen Sperrprotokoll
  - zentrales (primary site) 2PL
  - Primary Copy 2PL
  - verteiltes 2PL
- häufig mit Snapshot Verfahren
  - Read Consistency bei Oracle
  - verwenden der SCN als Timestamp
  - SCN Synchronisieren bei verteilten DBS



## 15.1 Zentral

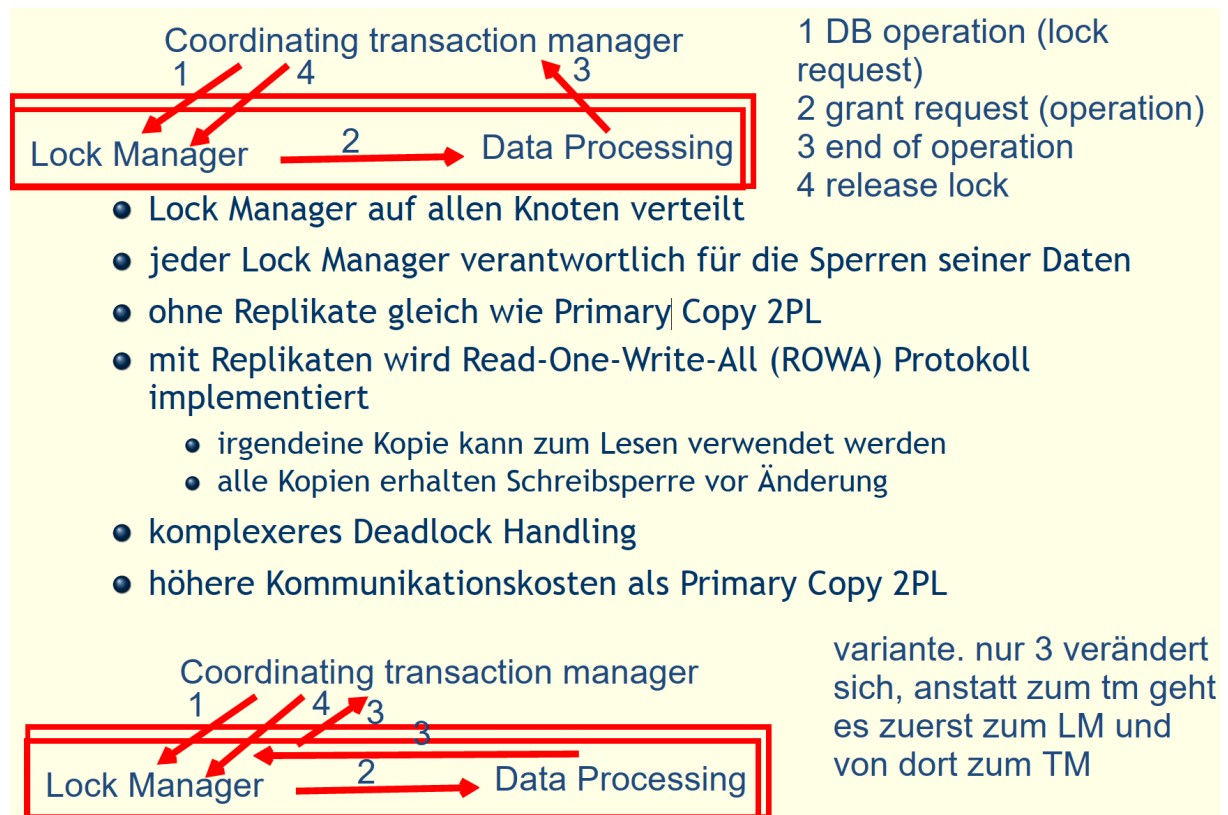


**Abbildung 5:** zentrales 2PL

## 15.2 Primary

- Lock Manager auf einigen Knoten verteilt jeder Lock Manager verantwortlich für die Sperren einer Menge von Daten
- bei Replikaten wird eine Kopie als Primary Copy bestimmt, die anderen sind slave copies
- Schreibsperre nur auf Primary Copy
- nach Änderung der Primary Copy, Änderungen zu den slaves bringen
- komplexeres Deadlock Handling
- weniger Kommunikation, bessere Performance

### 15.3 Verteilt



**Abbildung 6:** Verteiltes 2PL

## 16 Sie können das Problem der Erkennung von verteilten Deadlocks beschreiben.

- Transaktionen sind in einem Deadlock, wenn sie blockiert sind und es bleiben, bis das System eingreift
- Deadlocks treten in 2PL auf zur Vermeidung nicht serialisierbarer Ablaufpläne
- um einen Deadlock aufzulösen, muss eine Transaktion abgebrochen werden
- in einem Wait For Graph (WFG) können Deadlocks erkannt werden
  - WFG ist gerichteter Graph und stellt die Erwartung auf Beziehung
  - zwischen Transaktionen dar
  - Transaktionen sind die Knoten im WFG
  - Kante  $T_i \rightarrow T_j$  im WFG bedeutet,  $T_i$  wartet auf die Freigabe einer
  - Sperre von  $T_j$

- Zyklen im WFG zeigen Deadlocks
- Ansatz: automatisch Deadlocks erkennen und eine betroffene Transaktion abbrechen
- bevorzugter Ansatz
  - höhere Ressourcen Nutzung
  - einfache Verfahren
- Timeout: Transaktion, die zu lange blockiert ist, abbrechen
  - einfach zu realisieren
  - bricht Transaktionen unnötigerweise ab
  - Deadlocks können lange bestehen

## **17 Sie kennen die Komponenten zur Sicherstellung von Atomicity und Durability verteilter Transaktionen.**

- Jeder Knoten kann
  - lokaler Teil der Transaktion zuverlässig verarbeiten
  - nach Bedarf lokales Commit durchführen
  - nach Bedarf lokales Rollback durchführen
  - nach Bedarf lokales Recovery durchführen
- Übergeordnete Protokolle müssen sich mit der Koordination der beteiligten Knoten befassen
  - Start der Transaktion beim Ursprungsknoten verarbeiten
  - Read und Write an den Zielknoten verarbeiten
  - spezielle Vorkehrungen bei Replikation
  - Abort, Commit und Recover spezifisch für DDB

### **17.1 Komponenten**

- unterscheiden in
  - Coordinator Process beim Ursprungsknoten, steuert die Ausführung
  - Participant Process bei den anderen Knoten, die an der Transaktion beteiligt sind
- verschiedene Protokolle (Reliability Protocols)
  - Commit Protocols
  - Termination Protocols
  - Recovery Protocols

## 18 Sie können das 2-Phasen-Commit Protokoll beschreiben und kennen 3 verschiedene Kommunikationsvarianten.

### Phase 1: Wahlphase

Der Koordinator fragt alle Teilnehmer, ob sie bereit sind für ein Commit: **Prepare**.

Jeder Teilnehmer teilt dem Koordinator seine Entscheidung mit: **Vote-Commit** oder **Vote-Abort**.

### Phase 2: Entscheidungsphase

Der Koordinator trifft endgültige Entscheidung:

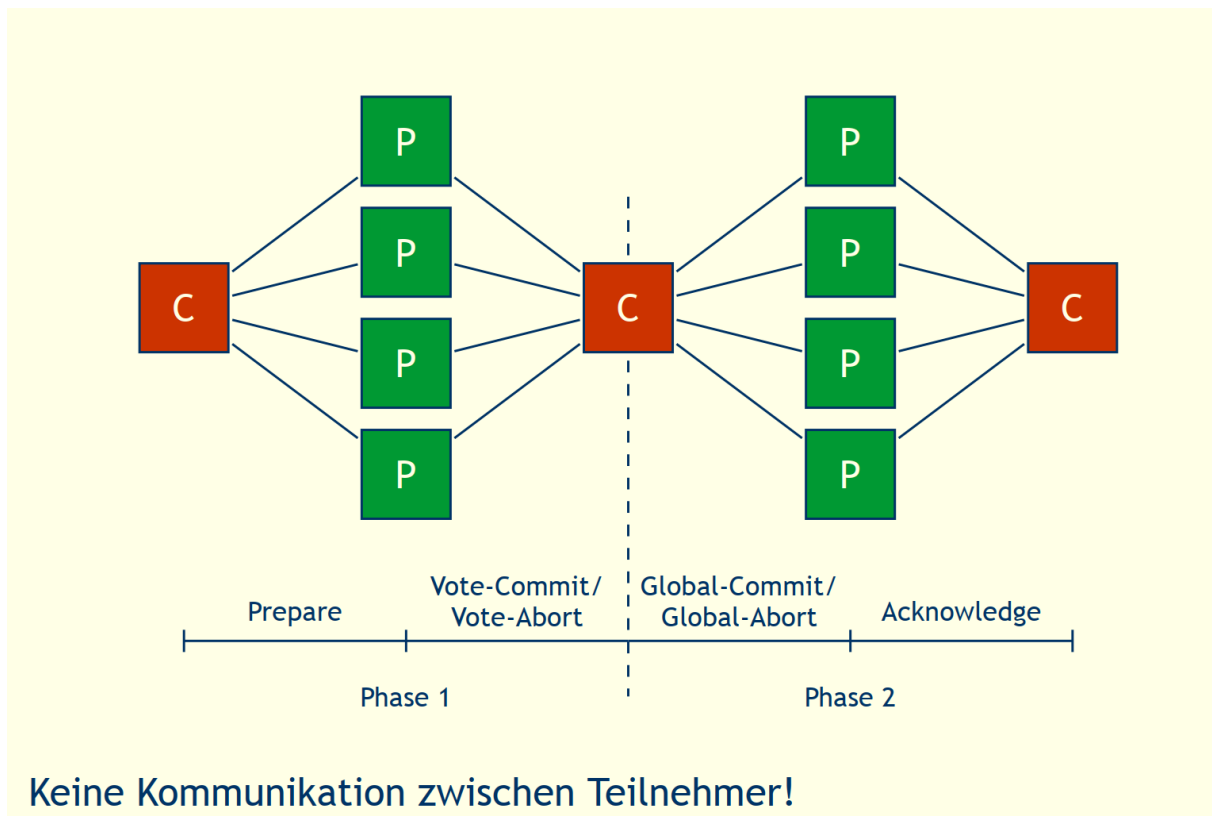
Commit: Falls alle Teilnehmer bereit zum Commit

Abort: In allen anderen Fällen

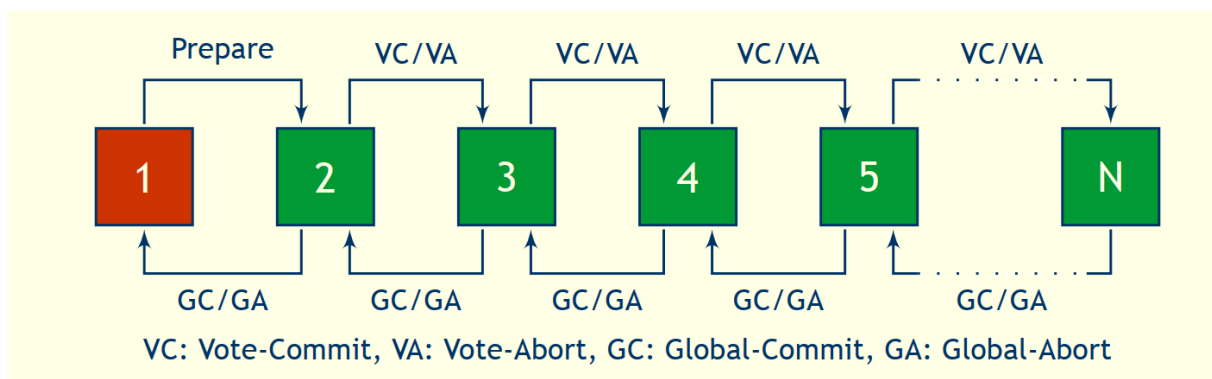
und teilt sie den Teilnehmern mit: **Global-Commit** oder **Global-Abort**.

Diese müssen sich entsprechend verhalten und bestätigen: **Acknowledge**.

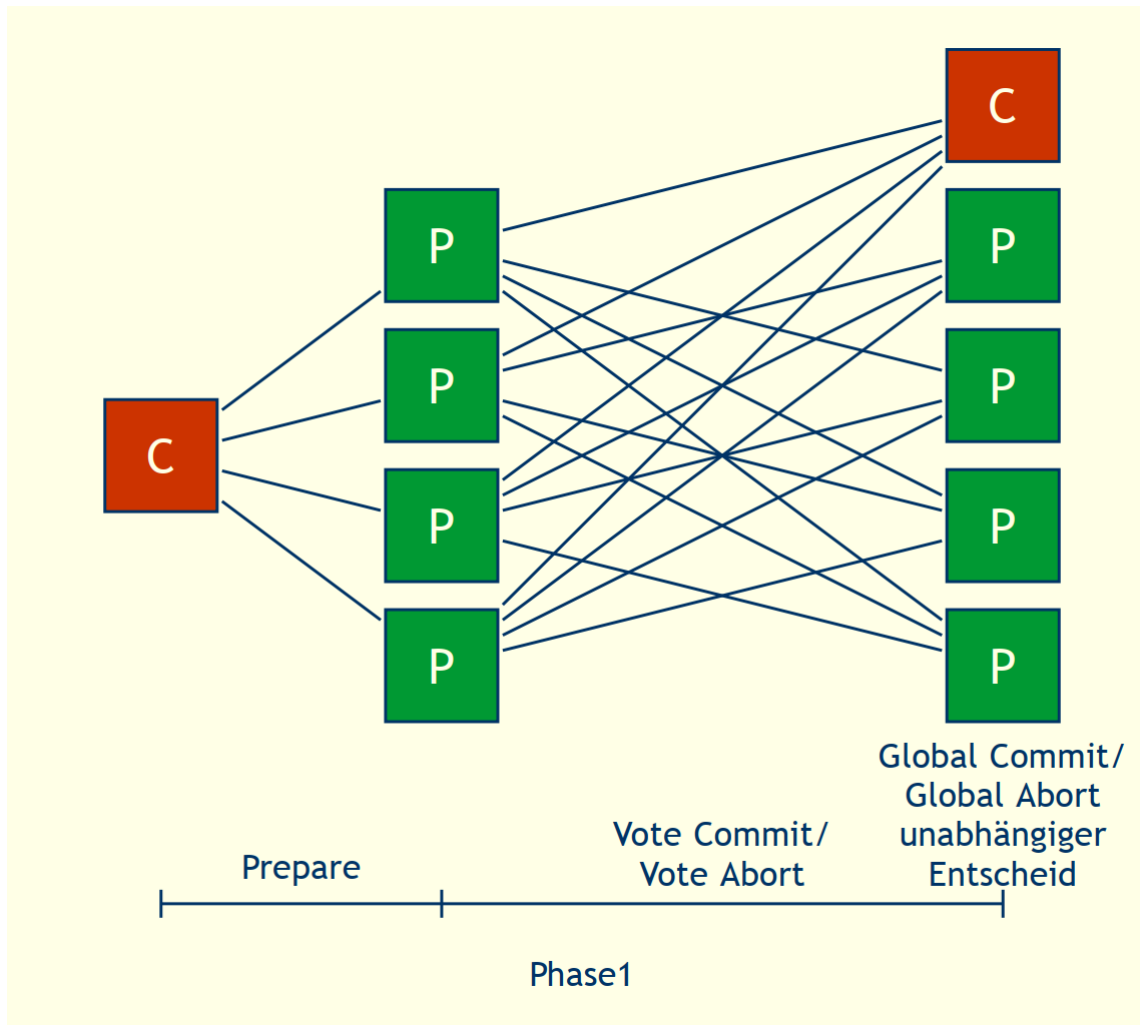
**Abbildung 7:** 2 Phasen Commit Protokoll



**Abbildung 8:** Zentrales 2 Phasen Commit rprotokoll



**Abbildung 9:** Lineares 2 Phasen Commit protokoll



**Abbildung 10:** Verteilt 2 Phasen Commit protokoll

## 19 Sie können die 2 Hauptprobleme des 2PC Protokolls erläutern.

- Blockierend
  - Im Zustand READY muss Teilnehmer auf Entscheid des Koordinators warten.
  - Fällt Koordinator aus, ist Teilnehmer blockiert.
  - Damit wird Verfügbarkeit reduziert.
- Unabhängiges Recovery nicht möglich
  - Recovery eines Teilnehmers nach Ausfall im Zustand READY kann nicht unabhängig erfolgen.
  - Verfügbarkeit des Koordinators oder eines Teilnehmers, der die Rolle des Koordinators übernimmt ist nötig.

## 20 Sie kennen die Vor- und Nachteile von replizierten Daten.

- Gründe für Replikation
  - Zuverlässigkeit
    - \* vermeidet single points of failure
  - Performance
    - \* vermeidet Kommunikationskosten (lokaler Zugriff)
  - Skalierbarkeit
    - \* unterstützt Wachstum des Systems
  - Anforderungen durch Anwendungen
    - \* Spezifikation
- Probleme mit Replikation
  - Transparenz der Replikation
    - \* Abbilden logischer Zugriff in physische Zugriffe auf Kopien
  - Fragen der Konsistenz
    - \* Konsistenzkriterien
    - \* Synchronisierung der Kopien

## 21 Sie kennen verschiedenen Konsistenzmodelle für die Replikation.

### 21.1 Strong Consistency

One-Copy Equivalence - gegenseitige Konsistenz wenn alle Kopien identische Werte haben - Wirkung einer Transaktion auf replizierte Daten ist die gleiche, wie wenn sie auf einer einzelnen Datenmenge operierte.

One-Copy Serializability - Wirkung von Transaktionen auf replizierte Daten ist die gleiche, wie wenn sie eine nach der anderen auf einer einzelnen Datenmenge operierten. - Histories sind äquivalent zu einer seriellen Ausführung auf nicht replizierten Daten

### 21.2 Weak Consistency

- Abgeschwächte Formen der Konsistenz
- Eventual Consistency: Konsistenz wird letztendlich (später) erreicht

- Wenn weitere Updates ausbleiben, konvergieren die Replikate zu identischen Kopien.
- Nur die Ausbreitung der Updates muss garantiert sein
- Kein Problem, solange die Nutzer immer auf das gleiche Replikat zugreifen.
  - Übergang zu Client-Centric Consistency
  - Garantiert einem einzelnen Nutzer konsistenten Zugriff auf Daten

### 21.3 Client-Centric Consistency

**Monotonic Reads (gleichbleibend)** - Liest ein Prozess Datenelement x, dann gibt jedes nachfolgende Lesen von x durch diesen Prozess denselben oder neueren Wert zurück **Monotonic Writes** - Schreiboperation auf Datenelement x ist abgeschlossen, bevor derselbe Prozess nachfolgende Schreiboperationen auf x ausführt (Serialisierung des Schreibens) **Read your Writes** - Die Wirkung von Schreiboperation auf Datenelement x wird immer in nachfolgenden Leseoperationen auf x durch denselben Prozess gesehen **Writes follow Reads** - Schreiboperationen auf Datenelement x, die auf ein Lesen durch denselben Prozess folgen, überschreiben immer denselben oder neueren Wert von x

## 22 Sie können die verschiedenen Update Propagation Strategien beschreiben.

Zeitpunkt der Update Propagation - Eager (Synchronous) - Lazy (Asynchronous) Ort des Updates - Primary Copy (Master) - Update everywhere (Group)

### 22.1 Eager / Lazy Replication

**Eager (Synchronous) Replication** - Jede Änderung wird sofort zu allen Kopien übertragen - Übertragung der Änderungen erfolgt innerhalb der Grenzen der Transaktion. - ACID Eigenschaften gelten für alle Kopien **Lazy (Asynchronous) Replication** - Zuerst werden die lokalen Kopien geändert. Anschliessend werden die Änderungen zu allen anderen Kopien übertragen (push/pull). - Während der Übertragung sind Kopien inkonsistent - Zeitraum der Inkonsistenz kann in Abhängigkeit der Anwendung angepasst werden

### 22.2 Master / Group Replication

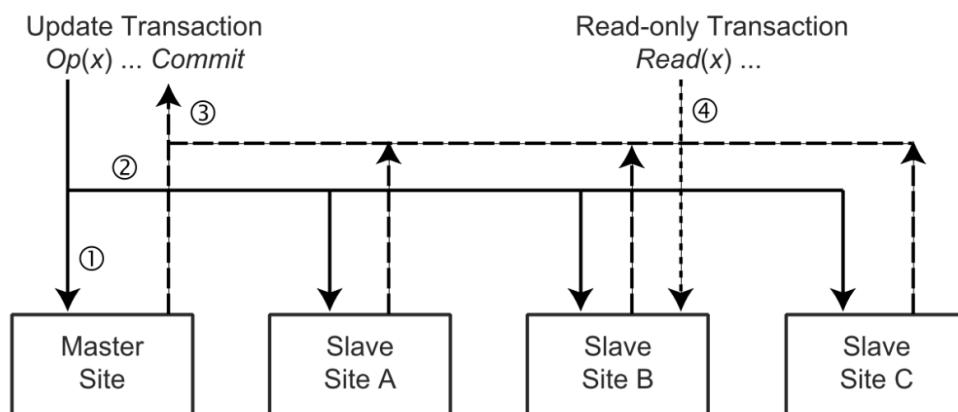
**Master (Primary Copy) Replication** - Es gibt eine einzige Kopie, auf der Änderungen ausgeführt werden können (Master) - Alle anderen Kopien (Slaves) übernehmen die Änderungen vom Master - für



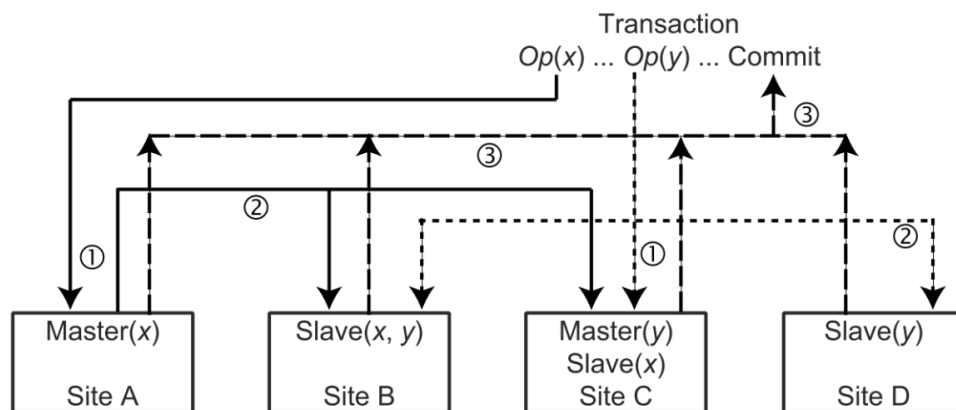
verschiedene Datenelemente können verschiedene Knoten Master sein **Group (Update everywhere)**  
**Replication** - Änderungen können auf jeder Kopie ausgeführt werden. - D.h. jeder Knoten, der eine Kopie besitzt, kann auf ihr Änderungen ausführen.

### 22.3 Eager Master (Synchronous Primary Copy) Replication

**Primary Copy** - Read: lokales Lesen (eigene Kopie), Resultat zurückgeben - Write: lokales Schreiben, Write an alle Slaves weiterleiten (in FIFO Reihenfolge oder mit Timestamps), Kontrolle sofort dem Nutzer zurückgeben - Commit: verwende 2PC als Koordinator - Abort: lokales Abort, Slaves informieren  
**Slave** - Read: lokales Lesen (eigene Kopie), Resultat zurückgeben - Write von Master: ausführen der Writes in richtiger Reihenfolge (FIFO oder Timestamp) - Write von Client: zurückweisen oder an Master weiterleiten - ist Teilnehmer am 2PC der Transaktionen vom Master



**Fig. 13.1** Eager Single Master Replication Protocol Actions. (1) A *Write* is applied on the master copy; (2) *Write* is then propagated to the other replicas; (3) Updates become permanent at commit time; (4) Read-only transaction's *Read* goes to any slave copy.

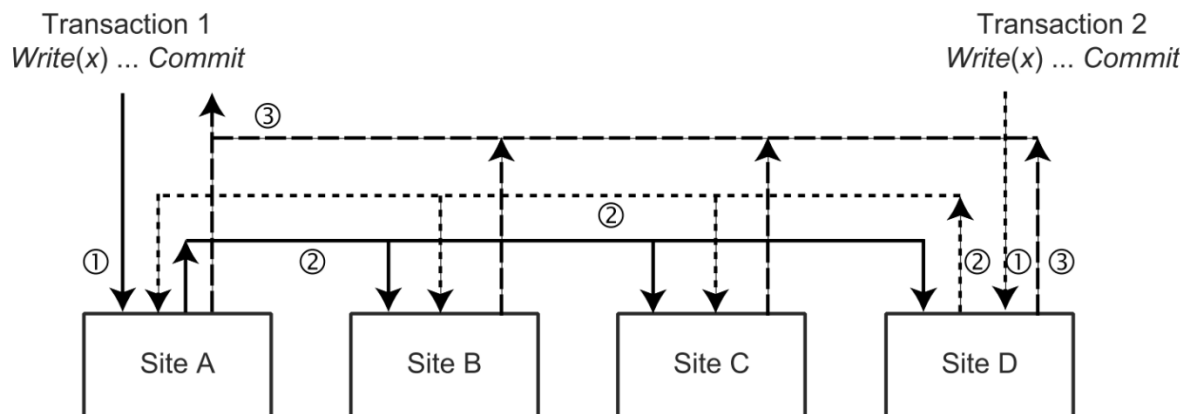


**Fig. 13.2** Eager Primary Copy Replication Protocol Actions. (1) Operations (*Read* or *Write*) for each data item are routed to that data item's master and a *Write* is first applied at the master; (2) *Write* is then propagated to the other replicas; (3) Updates become permanent at commit time.

**Vorteile** - Änderungen müssen nicht koordiniert werden - Keine Inkonsistenzen **Nachteile** - Längste Antwortzeit - Nur bei wenigen Updates sinnvoll (Master ist Flaschenhals) - Lokale Kopien sind beinahe nutzlos - selten eingesetzt

## 22.4 Eager Group (Synchronous Update Everywhere) Replication

Read One Write All (ROWA) - jeder Knoten verwendet 2 Phasen Sperrprotokoll - Leseoperationen werden lokal durchgeführt - Schreiboperationen werden auf allen Knoten ausgeführt mithilfe eines verteilten Sperrprotokolls



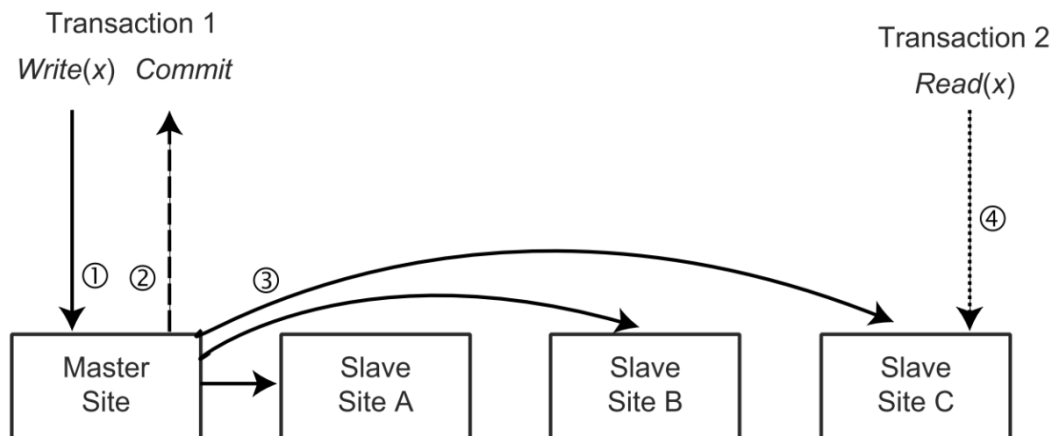
**Fig. 13.3** Eager Distributed Replication Protocol Actions. (1) Two *Write* operations are applied on two local replicas of the same data item; (2) The *Write* operations are independently propagated to the other replicas; (3) Updates become permanent at commit time (shown only for Transaction 1).

#### Abbildung 11: Eager Group

**Vorteile** - Keine Inkonsistenzen - elegante Lösung (symmetrisch) **Nachteile** - Vielzahl von Nachrichten  
 - Antwortzeiten der Transaktionen sind sehr lang - beschränkte Skalierbarkeit (Deadlock Wahrscheinlichkeit wächst mit Anzahl Knoten)

### 22.5 Lazy Master (Asynchronous Primary Copy)

**Primary Copy** - Read: lokales Lesen (eigene Kopie), Resultat zurückgeben - Write: lokales Schreiben  
 Kontrolle dem Nutzer zurückgeben - Commit, Abort: Transaktion lokal beenden - Irgendwann nach dem Commit: an alle Knoten die Änderungen in einer einzigen Nachricht übermitteln (FIFO oder Timestamp Reihenfolge)  
**Slave** - Read: lokales Lesen (eigene Kopie), Resultat zurückgeben - Nachricht von Master: Änderungen in richtiger Reihenfolge (FIFO oder Timestamp) anwenden - Write von Client: zurückweisen oder an Master weiterleiten - Commit, Abort: nur für lokale Read-only Transaktionen



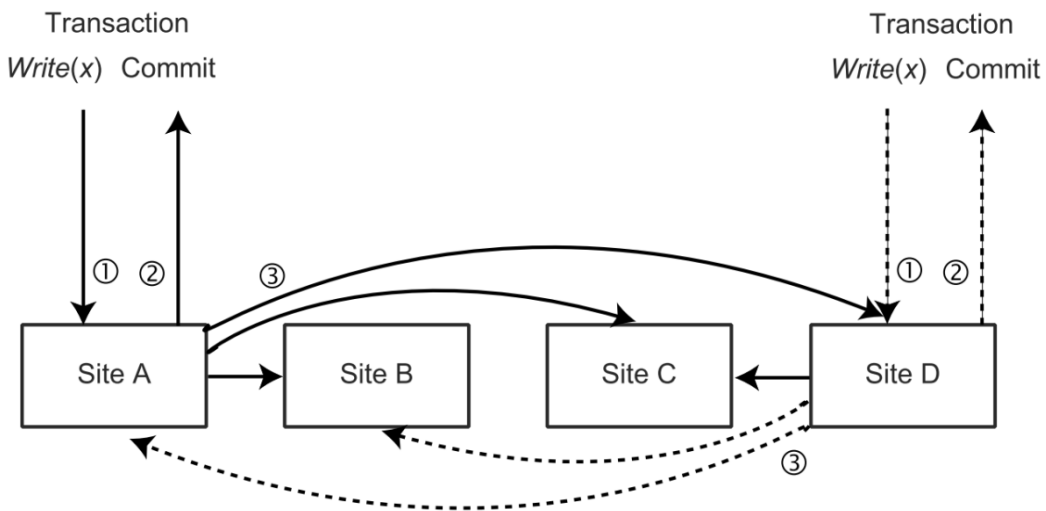
**Fig. 13.4** Lazy Single Master Replication Protocol Actions. (1) Update is applied on the local replica; (2) Transaction commit makes the updates permanent at the master; (3) Update is propagated to the other replicas in refresh transactions; (4) Transaction 2 reads from local copy.

#### Abbildung 12: Lazy Master

**Vorteile** - Keine Koordination nötig - kurze Antwortzeiten (Transaktionen sind lokal) **Nachteile** - lokale Kopien sind nicht aktuell - Inkonsistenzen (verschiedene Knoten haben unterschiedliche Werte für das gleiche Datenelement)

### 22.6 Lazy Group (Asynchronous Update Everywhere) Replication

jeder Knoten - Read: lokales Lesen (eigene Kopie), Resultat zurückgeben - Write: lokales Schreiben Kontrolle dem Nutzer zurückgeben - Commit, Abort: Transaktion lokal beenden - Irgendwann nach dem Commit: an alle Knoten die Änderungen in einer einzigen Nachricht übermitteln (FIFO oder Timestamp Reihenfolge) - Nachricht von anderem Knoten: - Erkennen von Konflikten - Änderungen anwenden - Reconciliation (Abgleichen)



**Fig. 13.6** Lazy Distributed Replication Protocol Actions. (1) Two updates are applied on two local replicas; (2) Transaction commit makes the updates permanent; (3) The updates are independently propagated to the other replicas.

**Abbildung 13:** Lazy Group

**Vorteile** - Keine Koordination nötig - kürzeste Antwortzeiten **Nachteile** - Inkonsistenzen - Änderungen könne verloren gehen (Abgleich)



- 23 Sie können erläutern, wie in Oracle 12c Replikation mit Materialized Views realisiert wird und welcher Replikations Strategie dies entspricht.**
- 24 Sie kennen die Kategorisierung der NoSQL Systeme mit den typischen Vertretern.**
- 25 Sie können das CAP-Theorem erläutern.**
- 26 Sie kennen das Map/Reduce Verfahren und können dafür typische Anwendungen programmieren.**
- 27 Sie kennen die Modellierungsprinzipien für Cassandra und können diese anwenden.**
- 28 Sie können mit CQL Datendefinitionen realisieren und Anfragen formulieren.**
- 29 Sie können Datenmodelle für MongoDB entwerfen und kennen die verschiedenen Darstellungsmöglichkeiten von Beziehungen.**
- 30 Sie können mit der Mongo Shell Datendefinitionen realisieren und Anfragen formulieren.**
- 31 Sie kennen das Graphdatenmodell von Neo4j.**
- 32 Sie können mit der Sprache Cypher Datendefinitionen realisieren und Anfragen formulieren.**
- 33 Sie können das Datenmodell des Resource Description Frameworks anhand eines Beispiels erläutern.**

- Alles, was benannt werden kann, ist eine Ressource

- Global eindeutige Namen durch URI bzw. IRI
- Konstruktion von Aussagen mit Subjekt, Prädikat und Objekt
- Subjekt und Prädikat sind immer Ressourcen.
- Das Objekt kann entweder eine Ressource oder ein Literal sein
- Dargestellt als gerichteter Graph
- Knoten: Subjekt, Objekt
- Kanten: Prädikat

```

1  xml
2  <rdf:RDF
3    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4    xmlns:lit="http://www.SampleLiteratur.rdf#">
5
6    <rdf:Descriptionrdf:about="http://www.SampleLiteratur.rdf#UmbertoEco"
7      >
8      <lit:Autor_von rdf:resource="http://www.SampleLiteratur.rdf#
9        DerNameDerRose"/>
10     <lit:Name>Umberto Eco</lit:Name>
11   </rdf:Description>
12 </rdf:RDF>

```

Property	Description
rdf:type	The subject is an instance of a class.
rdf:type	The subject is an instance of a class.
rdf:Statement	The class of RDF statements.
rdf:Property	The class of RDF properties.
rdf:subject	The subject of the subject RDF statement.
rdf:predicate	The predicate of the subject RDF statement.
rdf:object	The object of the subject RDF statement.
rdfs:Class	The class of classes.
rdfs:domain	A domain of the subject property.
rdfs:range	A range of the subject property.
rdfs:subClassOf	The subject is a subclass of a class.
rdfs:subPropertyOf	The subject is a subproperty of a property.
rdfs:Ressource	The class resource, everything.