Einsatz von Replikation

Florian Munz

Agenda

- Wofür braucht man Replikation?
  - Anwendungsszenarien
- Wie funktioniert Replikation?
  - Konsistenzbegriff und Strategien
- Wie implementiert man Replikation?
  - Lösungen

Einführung

- Durch die vollständige oder teilweise Kopie einer Datenbank-Instanz erzeugt die Replikation absichtlich Datenredundanz
- Dadurch entstehen autonome Datenbank-Instanzen mit identischen Daten
- Solange diese Daten statisch sind, entstehen keine weiteren Probleme
- Ändern sich diese Daten, sind Strategien erforderlich, wie Änderungen synchronisiert oder propagiert werden

Unterschiede zwischen Caching und Replikation

- Caching betrifft redundante Daten, die
  - volatil sind (im Hauptspeicher stehen)
  - dynamisch erzeugt und gelöscht werden (z. B. durch LRU-Strategie)
  - für die Administration unsichtbar sind
- Replikation betrifft redundante Daten, die
  - persistent sind (auf der Platte stehen)
  - autonom existieren
  - für die Administration sichtbar sind
- Caching und Replikation haben keine Auswirkungen auf die Anwendungsprogrammierung

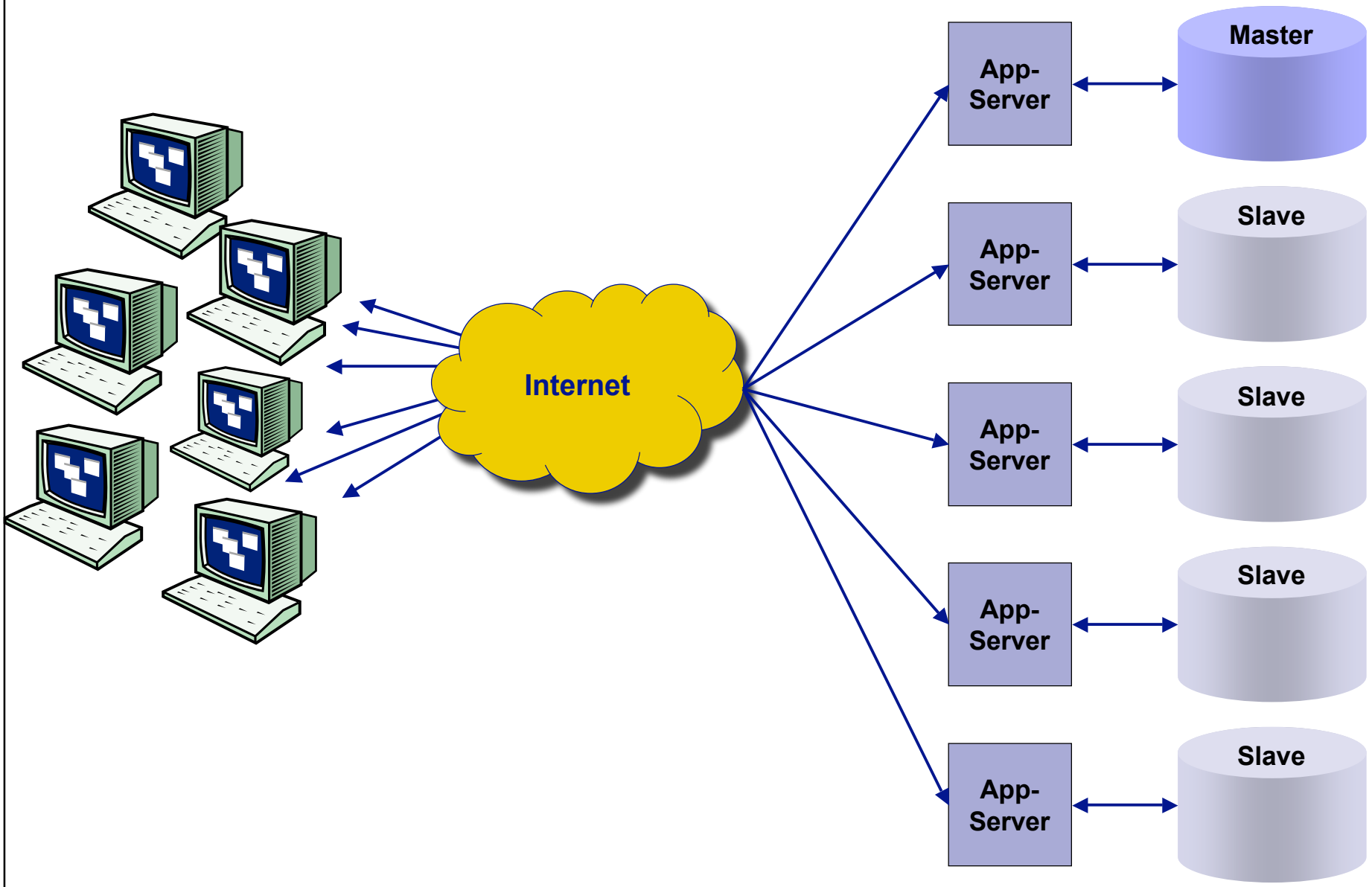
Typische Anwendungsszenarien für Replikation

- Anfrage-Last auf mehrere Datenbank-Instanzen verteilen
  - Populäre Web Sites wie Ebay, Yahoo, Google etc.
- Ausfallsicherheit
  - Fail-over Konfigurationen zum Schutz vor HW-Ausfällen
- Mobile Anwendungen
  - Autonome Anwendungen auf Laptops oder PDAs

Anfrage-Last verteilen

- Populäre Web Sites mit täglich Hunderttausenden von Besuchern, die überwiegend in einem Katalog browsen
- Typisches Profil: 97% Lesezugriffe, 3% Bestellungen
- Produktkataloge sind nahezu statisch
- Anfrage-Last wird auf mehrere identische Datenbank-Instanzen verteilt
- Ausgezeichnete Datenbank-Instanz (Master) für Änderungen
- Änderungen werden vom Master zu den anderen Datenbank-Instanzen (Slaves) propagiert
- Ausfall eines Slaves unkritisch
- Beim Ausfall des Masters übernimmt ein Slave die Master-Rolle

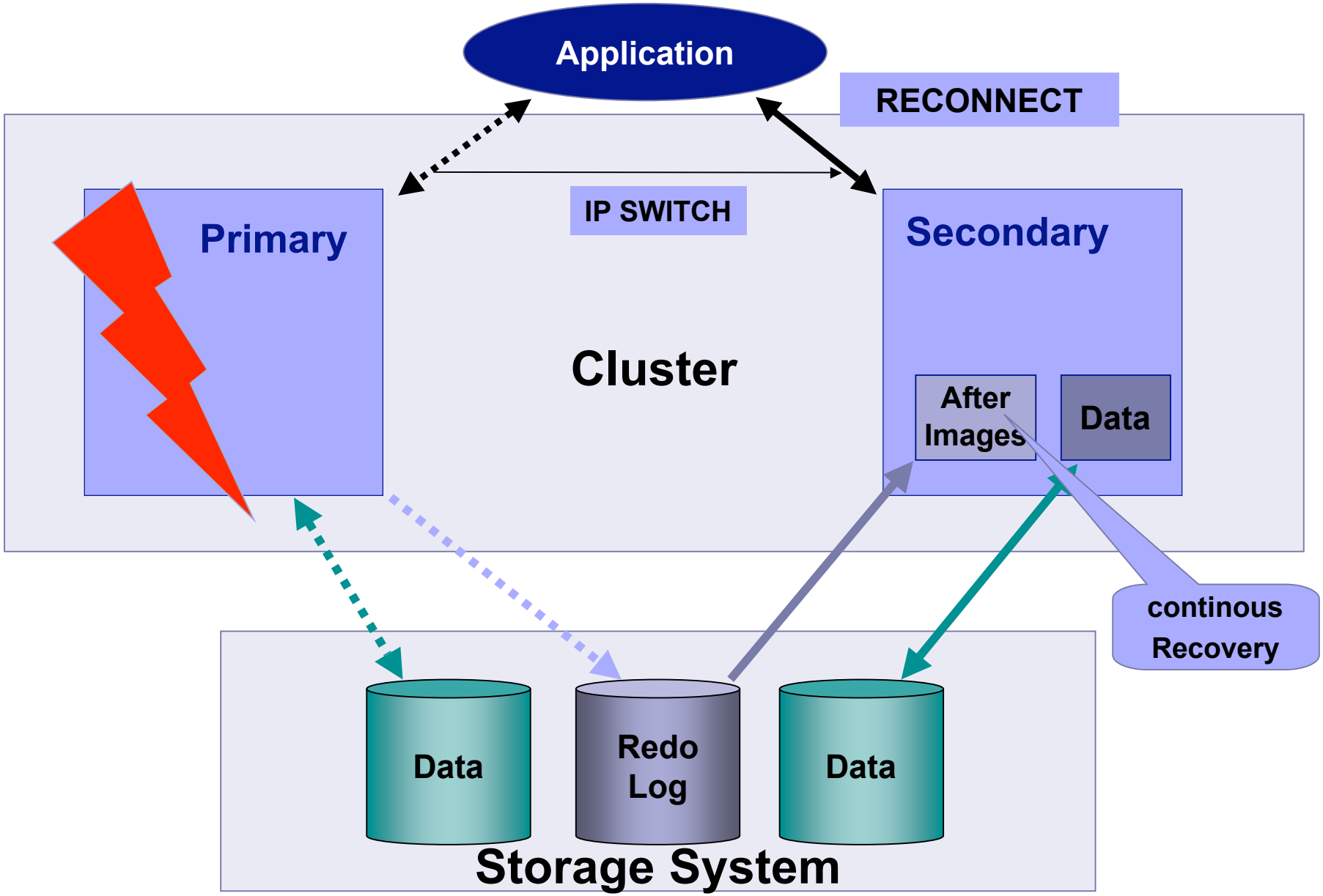
Konfiguration einer Lastverteilung



Ausfallsicherheit

- Um HW-Fehler abzufedern, wird ein redundantes System (Standby) bereitgestellt (Betriebssystem-Cluster)
- Primary/Secondary-Knoten bzw. Master/Slave
- Der Datenbestand des Masters wird zum Slave repliziert (vollständige Datenbank-Kopie)
- Änderungen des Masters werden zum Slave propagiert
- Hohe Änderungslast (OLTP-Profil) muss verkraftet werden
- Beim Ausfall des Masters (Heartbeat der Cluster-Logik) übernimmt der Slave die Funktion des Masters und alle Anwendungs-Sessions

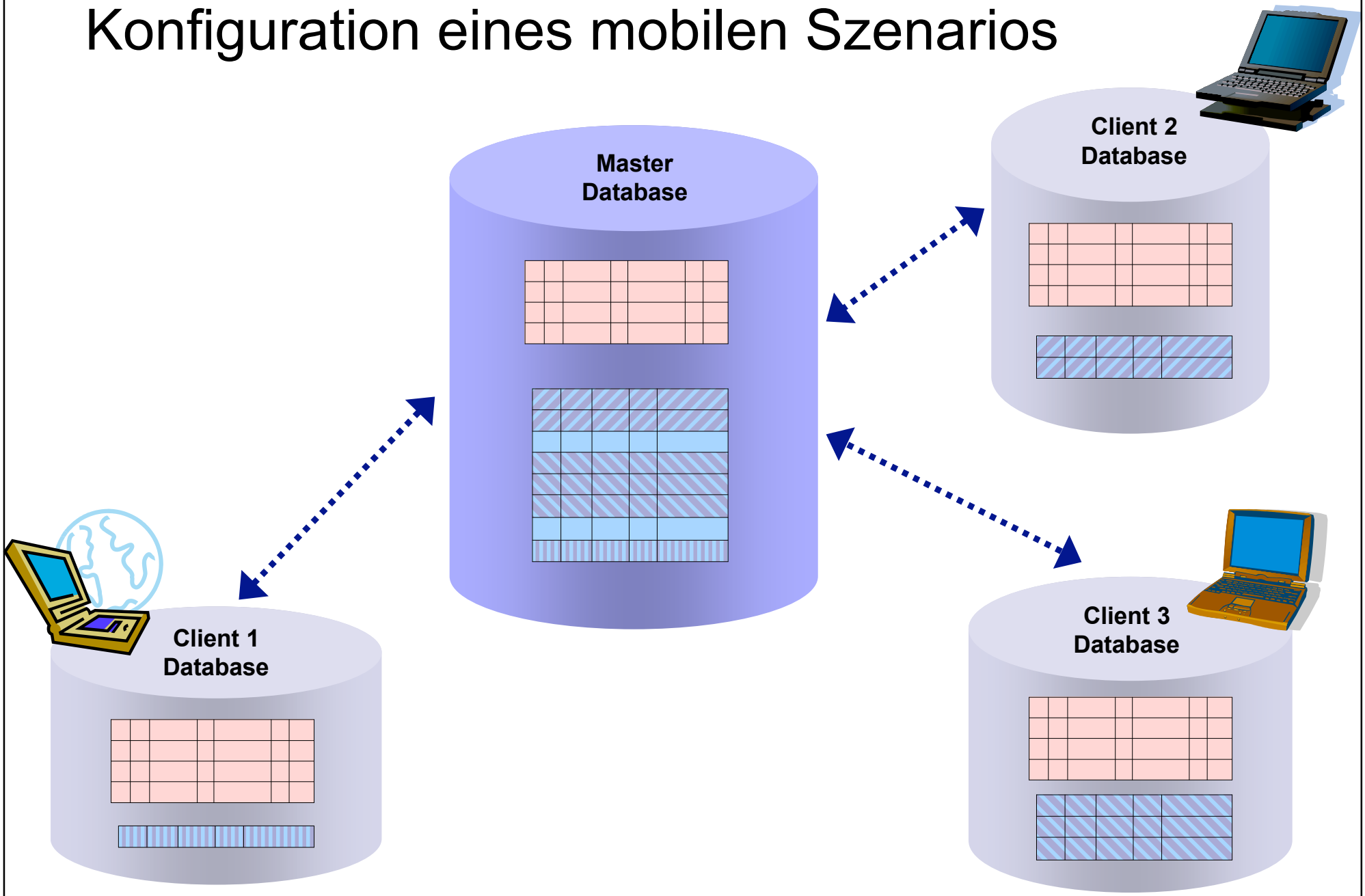
Ausfallsichere Konfiguration



Mobile Anwendungsszenarien

- Mobile Anwendungen auf Laptops oder PDAs für Vertriebs- oder Service-Mitarbeiter
- Offline-Anwendungen mit lokalem Datenbestand
- Keine ständige Netz- bzw. Funkverbindung
- Firmenweit interessante Daten (Produktkataloge, Preislisten, Lieferzeiten) werden in alle Endgeräte publiziert
- Regionale Daten (Kundenadressen, Einsatzpläne) werden spezifisch für einzelne Endgeräte aufbereitet
- Zentral enthält eine konsolidierte Datenbank eine Kopie sämtlicher Daten in sämtlichen Endgeräten
- Unidirektionale Replikation: Zentrale → Endgerät
- Bidirektionale Replikation: Zentrale ←→ Endgerät

Konfiguration eines mobilen Szenarios



Agenda

- Wofür braucht man Replikation?
  - Anwendungsszenarien
- Wie funktioniert Replikation?
  - Konsistenzbegriff und Strategien
- Wie implementiert man Replikation?
  - Lösungen

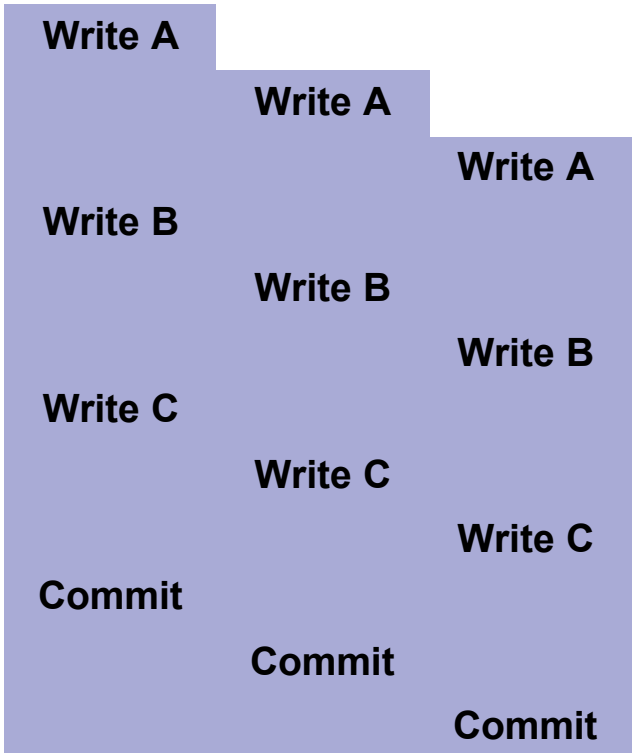
Transaktionale Konsistenz

- 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 **gleichen Ausgaben** sowie den **gleichen DB-Zustand** erzeugt **wie S auf der replizierten DB**.

Synchrone Änderungen

- Alle Replikate werden in derselben verteilten Transaktion aktualisiert
- Alle Sperrkonflikte werden vor dem Commit erkannt
- 2 Phasen Commit erforderlich

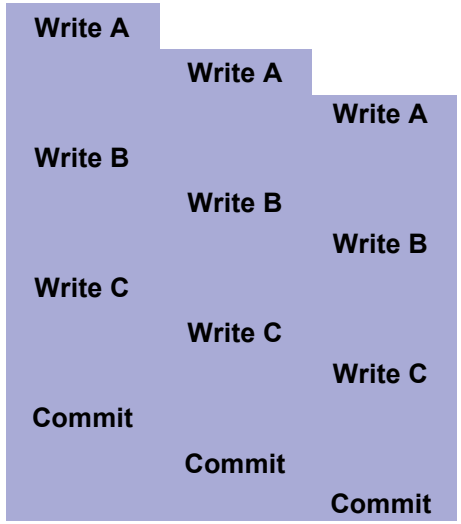
- Probleme:
  - Fehlende Skalierbarkeit
  - Keine Unterstützung mobiler Szenarien



Skalierungsprobleme

- Untersuchung der Skalierbarkeit synchroner Änderungen durch J. Gray:

- Jeder Knoten hält eine vollständige Kopie der DB, führt eine feste Anzahl von Transaktionen pro Sekunde durch
- Eine synchrone Transaktion ist *N* mal größer
  - Muss die Änderungen zu N-1 Knoten senden
  - Erzeugt N² mehr Aufwand (z.B. Deadlocks)
  - Nichtlineare Skalierung

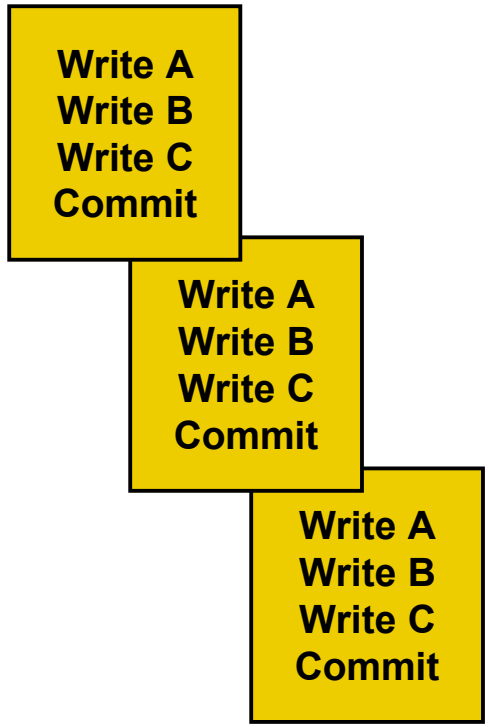


Schwache Konsistenz

- Konvergenz:
  - Nach einer änderungsfreien Zeit konvergieren alle Replikate in denselben Endzustand

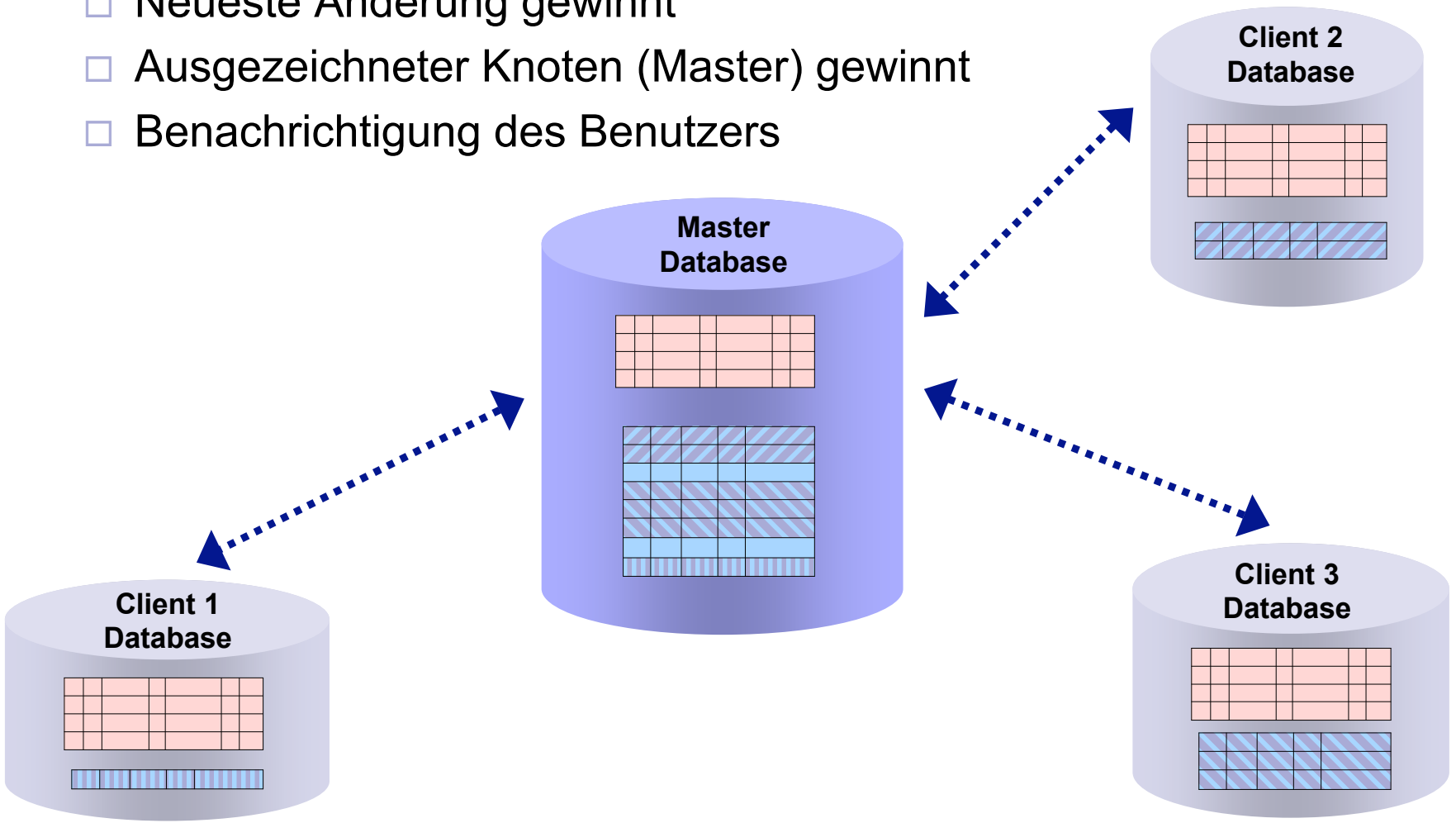
Asynchrone Änderungen

- Änderungen nur auf dem Master
  - Unidirektionale Replikation
  - Änderungen werden zeitversetzt zu den Slaves propagiert
- Änderungen auf allen Knoten (Update Everywhere)
  - Bidirektionale Replikation
  - Jede Kopie kann auf jedem Knoten aktualisiert werden
  - Änderungen werden zeitversetzt propagiert
  - Änderungskonflikte sind möglich



Bidirektionale Replikation

- Strategien zur Auflösung von Änderungskonflikten:
  - Neueste Änderung gewinnt
  - Ausgezeichneter Knoten (Master) gewinnt
  - Benachrichtigung des Benutzers



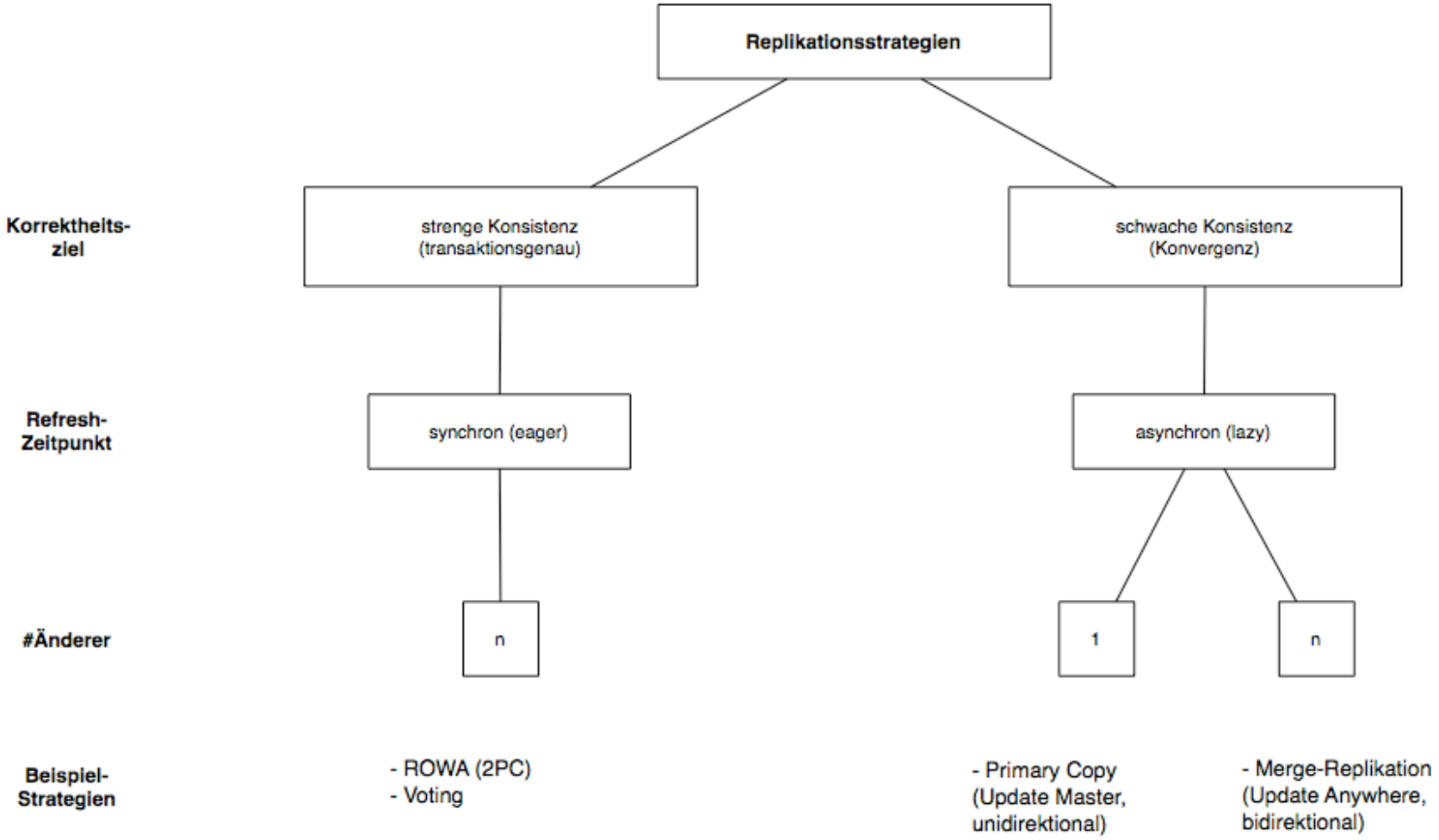
Klassifikation der Replikationsstrategien

- Wann werden Änderungen propagiert (eager/lazy)
- Wo sind Änderungen möglich (master/everywhere)

Zeitpunkt vs. Ort	**Eager**	**Lazy**
Master	1 Änderer 1 verteilte TA	1 Änderer M Transaktionen
Everywhere	N Änderer N verteilte TAs	N Änderer M Transaktionen

M = Anzahl der Replikate

Replikationsstrategien



Konsequenzen

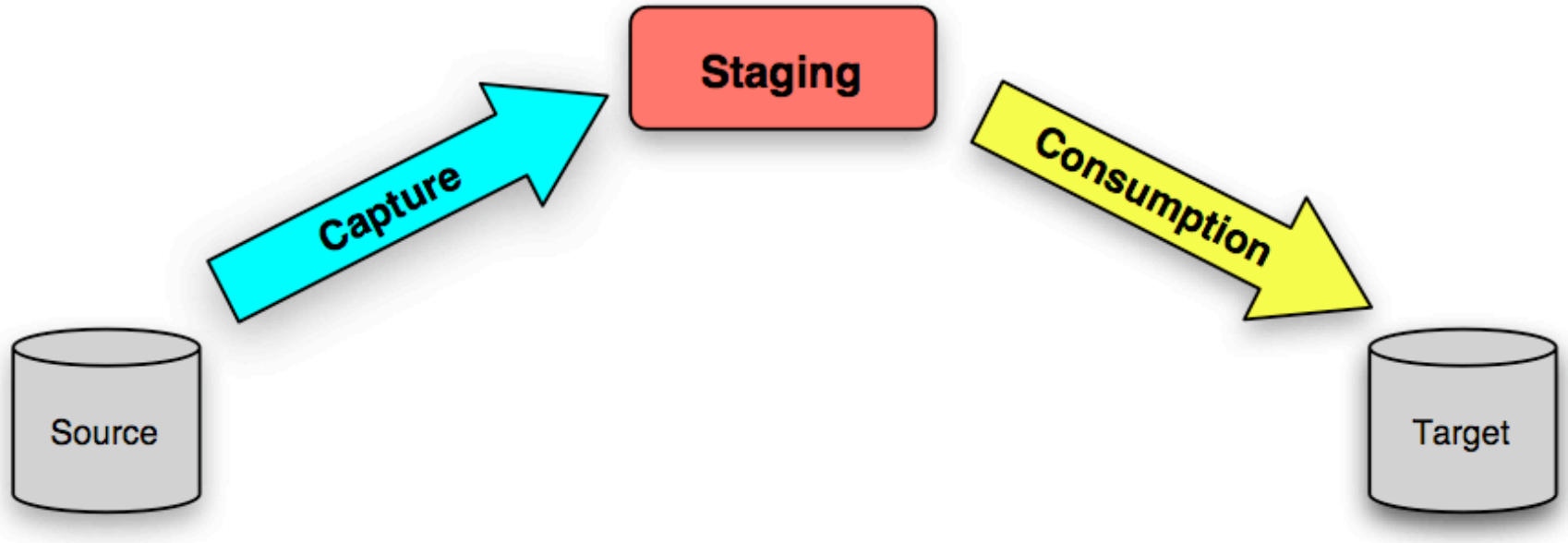
- Synchrone Verfahren aus dem Umfeld verteilter DBMS:
  - Enge Kopplung der DBMS-Instanzen (Verfügbarkeit)
  - Verteilte Transaktionen (2-Phasen Commit)
  - Schlechte Skalierbarkeit
- Asynchrone Verfahren
  - Lose Kopplung der DBMS-Instanzen
  - Zeitlich versetzte, lokale Transaktionen
  - Gute Skalierbarkeit bei niedrigen und hohen Änderungslasten
  - Schwächerer Konsistenzbegriff der Konvergenz
  - Dominieren im praktischen Einsatz

Agenda

- Wofür braucht man Replikation?
  - Anwendungsszenarien
- Wie funktioniert Replikation?
  - Konsistenzbegriff und Strategien
- Wie implementiert man Replikation?
  - Lösungen

Implementierungsprinzip

- Capture: Abgriff der Änderungen
- Staging: Aufbewahrung der Änderungen
- Consumption: Nachvollziehen der Änderungen



Anfrage-Last verteilen

- Beispiel: E-Applikation mit MySQL
- Initialisierung der Slaves durch Backup
- Delta-Übertragung der Änderungen am Master durch ein Kommando-Log
- Kommando-Log enthält alle Werte als Literale
- Asynchrone Abarbeitung der SQL-Statements im Kommando-Log durch die Slaves
- SQL-Statements im Kommando-Log sind durchnummeriert, um ein Wiederaufsetzen zu ermöglichen

MySQL-Szenario bei Sabre

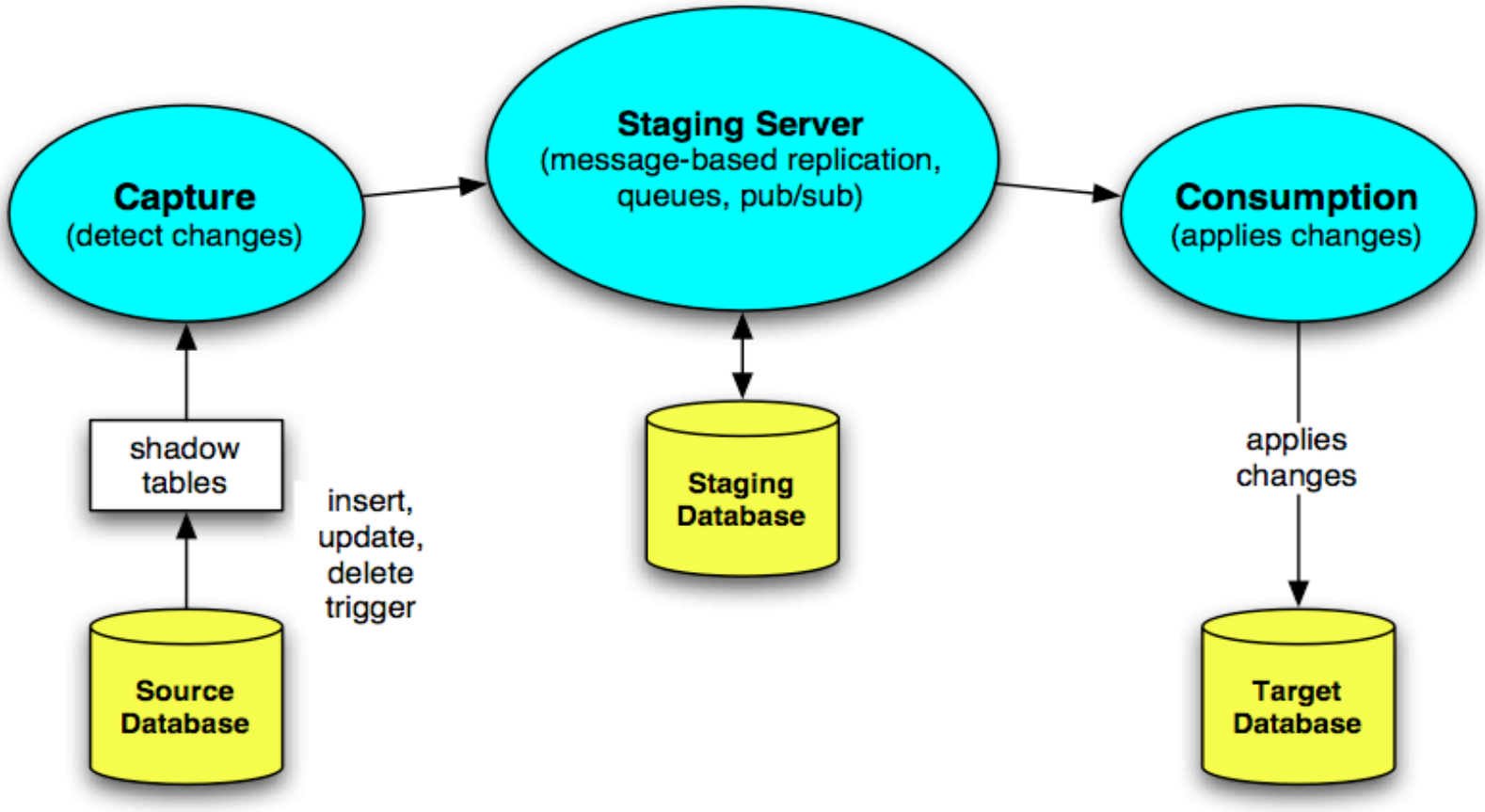
- User → Shop → Price → Buy
- Flugreservierung für American Airlines und Travelocity.com
- 250 Preis-Abfragen pro Sekunde
- 45 4-CPU Linux-Server, 1.5 GHz, 32 GB RAM für die Anfrage-Last (Shopping)
- Replikation von ca. 60 GB Daten auf 45 Knoten
- 17 weitere Knoten für die Abwicklung der Buchungen

Ausfallsicherheit

- Performance des Master-Knotens mit hoher Änderungs-last darf nicht beeinträchtigt werden
- Abgriff der Änderungen über das Redo-Log
- Asynchrones Nacharbeiten des Redo-Logs durch den Slave-Knoten
- Beim Ausfall des Master-Knotens
  - Arbeitet der Slave die ausstehenden Transaktionen nach
  - Rollt die offenen Transaktionen zurück
  - Anwendungs-Sessions werden per IP-Switch vom Slave übernommen
  - Offenen Transaktionen wird ein Transaktionsabbruch signalisiert

Mobile Anwendungsszenarien

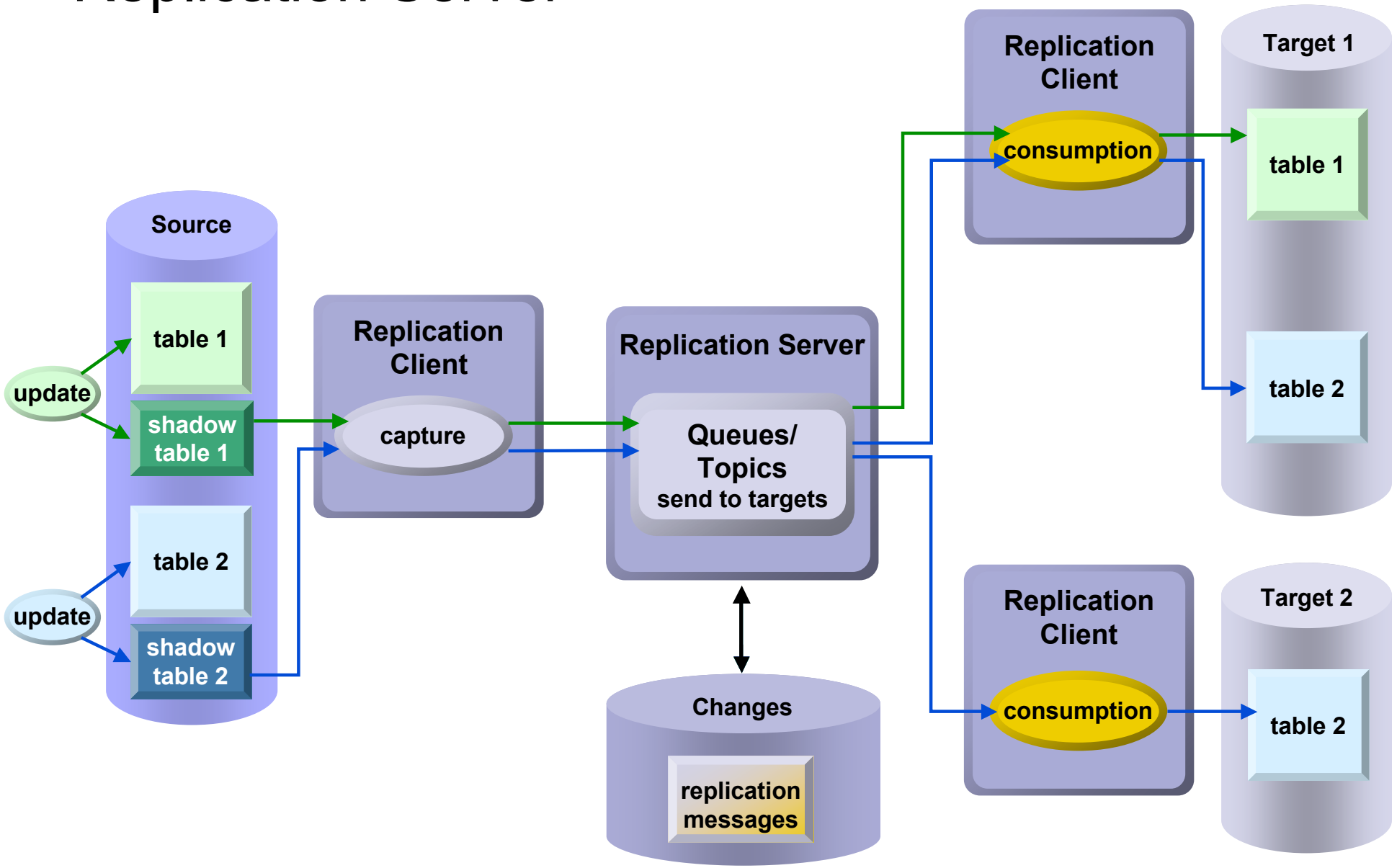
- Replication Server (eigene DB-Instanz zum Zwischenspeichern der Änderungen) entkoppeln Quelle und Ziele der Replikation bezüglich ihrer Verfügbarkeit



Queues vs. Publisher/Subscriber

- Queue-Prinzip:
  - 1 Sender, 1 Empfänger
  - Enge Kopplung, Sender kennt Empfänger
  - Änderungen werden abgeholt, verarbeitet und gelöscht
  - Änderungen erreichen nur einen einzigen Empfänger
  - Anwendungsbeispiel: Übermittlung von Bestellungen
- Publish/Subscribe-Prinzip:
  - 1 Publisher, N Subscriber
  - Lose Kopplung, Subscriber sind dem Publisher nicht bekannt
  - Publisher stellt Informationen bereit (Topics, Channels, Folder)
  - Subscriber holen sich diese Informationen ab
  - Der Publisher aktualisiert die Informationen von Zeit zu Zeit
  - Die Subscriber holen sich dann nur noch die Änderungen ab
  - Anwendungsbeispiel: Produktinformationen, Preisliste

Replication Server



Implementierungsvarianten

Leselast-Skalierung, geringe Änderungslast, unidirektional	
Capture	Kommando-Log
Staging	Kommando-Log-Datei
Consumption	Weitere DB-Instanz zur Skalierung
Hochverfügbarkeit, hohe Änderungslast, unidirektional	
Capture	Redo-Log
Staging	Redo-Log-Datei
Consumption	Standby-System
Mobile Computing, geringe Änderungslast, bidirektional	
Capture	Insert-, Update-, Delete-Trigger mit Schattentabelle
Staging	Dedizierter Replication Server
Consumption	Mobile Endgeräte

Zusammenfassung

- Replikation bringt Daten „vor Ort“
- Erzeugt dadurch Redundanz, die kontrolliert werden muß
- Unidirektionale Replikation (Master hat Änderungsrecht)
- Bidirektionale Replikation (alle dürfen ändern)
- Konvergenzbegriff statt transaktionaler Konsistenz
- Änderungen werden in der Regel asynchron übermittelt
- Capture-Staging-Consumption-Prinzip bei den unterschiedlichen Impementierungen

Vielen Dank für Ihre Aufmerksamkeit