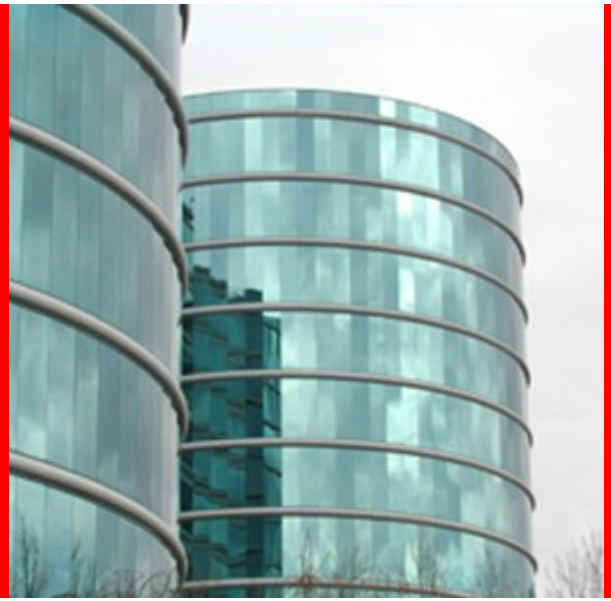

ORACLE®



MySQL

ORACLE

# MySQL Replikationstechnologien – eine Übersicht

Lenz Grimmer
MySQL Community Relations Specialist

# $ whoami


1998


2002


2008


2010

# Agenda

- Replikation: Definition und Klassifizierung
- Anwendungsgebiete
- Replikation in MySQL Server
- MySQL Cluster
- Andere Replikationstechologien

# Definition

*“Replikation: Die mehrfache Speicherung derselben Daten an meist mehreren verschiedenen Standorten und die Synchronisation dieser Datenquellen”*

(Quelle: http://de.wikipedia.org/wiki/Replikation_(Datenverarbeitung)

# Physikalische vs. logische Replikation

- Physikalisch: bit-identische Kopie durch blockweise oder zeilenweise Übertragung
- Logisch: Repliziere SQL-Anweisungen um den Datenbestand zu transformieren

# Anweisungsbasierte Replikation

- Übertragung aller datenverändernden Anweisungen (z.B. `INSERT`, `UPDATE`, `DELETE`)
- Empfänger wendet Änderungen auf lokale Kopie der Daten an
- Kompakt – geringe zu übertragende Datenmenge
- Gute Auditing-Möglichkeiten
- Nachteil: höhere Systembelastung
- Nicht-deterministische und system-spezifische Anweisungen problematisch

# Anweisungsbasierte Replikation

- Pro
  - Bewährt (seit MySQL 3.23 verfügbar)
  - Kleinere Logdateien
  - Auditing der tatsächlichen SQL-Anweisungen
  - Kein Primärschlüssel bei replizierten Tabellen erforderlich
- Kontra
  - Nicht-deterministische Funktionen und UDFs
  - LOAD_FILE(), UUID(), USER(), FOUND_ROWS() (RAND() and NOW() gehen)

# Zeilenbasierte Replikation

- Überträgt die tatsächlichen Änderungen am Datenbestand
- Empfänger übernimmt geänderte Werte direkt
- Keine Einschränkungen
- Geringerer Aufwand auf Empfängerseite
- Höheres Übertragungsvolumen

# Zeilenbasierte Replikation

- Pro
  - Alle Veränderungen können repliziert werden
  - Verfahren ähnlich zu anderen DBMSen
  - Erfordert weniger Locks für bestimmte INSERT, UPDATE oder DELETE Anweisungen
- Kontra
  - Mehr Daten müssen gelogged werden
  - Logfile-Größe (Auswirkungen auf Backup/Restore)
  - Replizierte Tabellen benötigen expliziten Primärschlüssel
  - Mögliche Ergebnis-Differenzen bei Bulk-INSERTs

# Asynchrone Replikation

- Verzögerung zwischen Erstellung und Replikation / Festschreibung der Daten auf Empfängerseite
- Replikation erfolgt nach Rückmeldung an Anwendung
- Robust gegenüber Ausfällen
- Geeignet bei Netzwerken mit höheren Latenzzeiten
- Risiko: Datenverlust bei Ausfall des Hauptsystems

# Synchrone Replikation

- Abschluß der Transaktion erst nach erfolgreicher Replikation an alle Teilnehmer
- Daten sind garantiert auf alle Knoten repliziert
- Erhöhter Kommunikationsaufwand zwischen den Teilnehmern
- Latenzzeit zwischen Anwendung und Datenbank steigt

# Semi-synchrone Replikation

- Sonderform als Kompromiss
- Transaktion erfolgreich, wenn mindestens ein Teilnehmer erfolgreich repliziert hat
- Stellt sicher, dass zumindest eine vollständige Kopie der Daten existiert

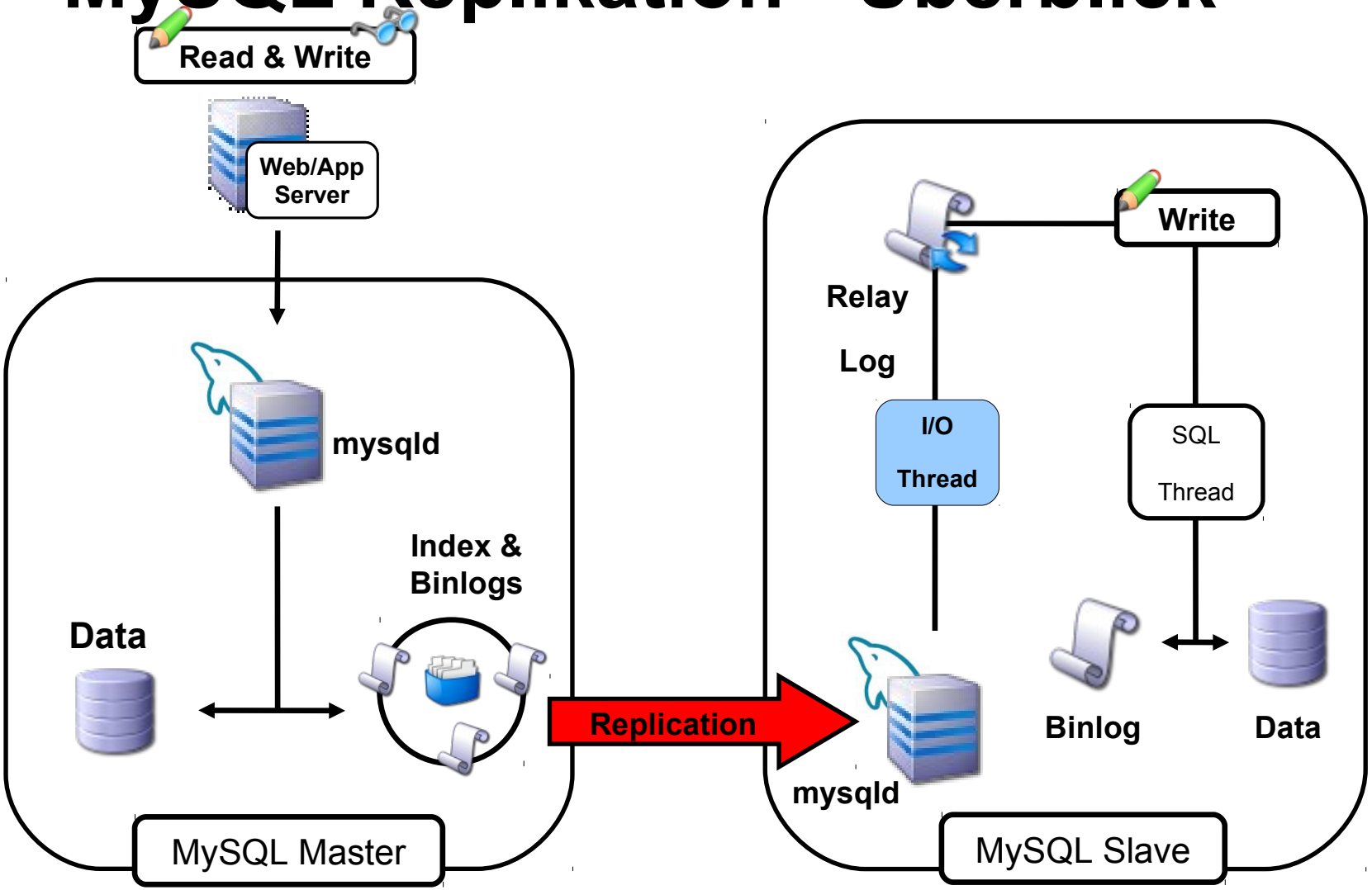
# Anwendungsgebiete für Datenbank-Replikation

- Hochverfügbarkeit durch Redundanz
  - Bei Ausfall des Primärsystems übernimmt eine Replika
- Skalierung
  - Scale-Out anstelle Scale-Up
  - Lese-Last per Load Balancer auf mehrere Server verteilen
- Backup
  - Reduziert I/O-Last auf Master
  - Offline Backups
- Wartung
  - Schema-Änderungen
  - Updates

# Replikation in MySQL

- Anweisungsbasiert (seit MySQL 3.23)
- Master verwaltet Binärlogs
- Unidirektional
- Asynchron
- Slave arbeitet single-threaded
- Seit MySQL 5.1: zeilenbasierte Replikation
- Ab MySQL 5.5: semisynchrone Replikation
- Ab MySQL 5.6: Verzögerte Replikation

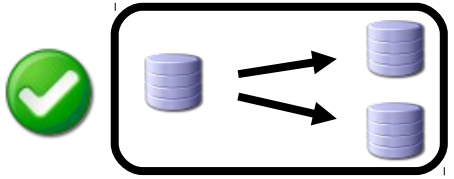
# MySQL Replikation - Überblick



# Replikationstopologien

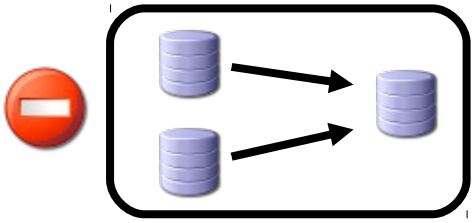
Master > Slave

Master > Slaves



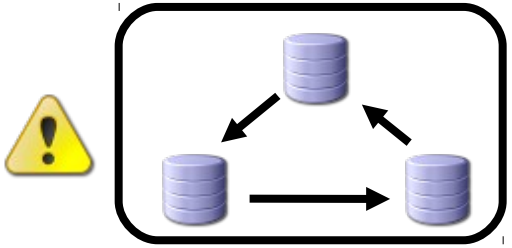
Master > Slave > Slaves

Masters > Slave (Multi-Source)



Master < > Master (Multi-Master)

Ring (Multi-Master)



# Master-Master-Replikation

- Zwei Server: sowohl Master als auch Slave zueinander
- Vereinfacht Failover
- **Nicht geeignet** um Schreiblast zu verteilen
- **Nicht** auf beide Master schreiben!
- Sharding oder Partitionierung (z.B. MySQL Proxy) eignen sich besser

# MySQL Cluster

- HV-Cluster für MySQL Server
- Speicher-Engine
- Auch unabhängig von MySQL Server verwendbar
- Flexible APIs (SQL, NDB, Java, LDAP...)
- Skalierung mit off-the-shelf hardware

# MySQL Cluster

- “Shared-nothing”-Architektur
- 99.999% Verfügbarkeit möglich
- Selbstheilend
- Rolling updates
- Synchrone Replikation mittels 2-phasen Commit-Protokoll
- Cluster-Cluster Replikation mittels asynchroner MySQL-Replikation (RBR)

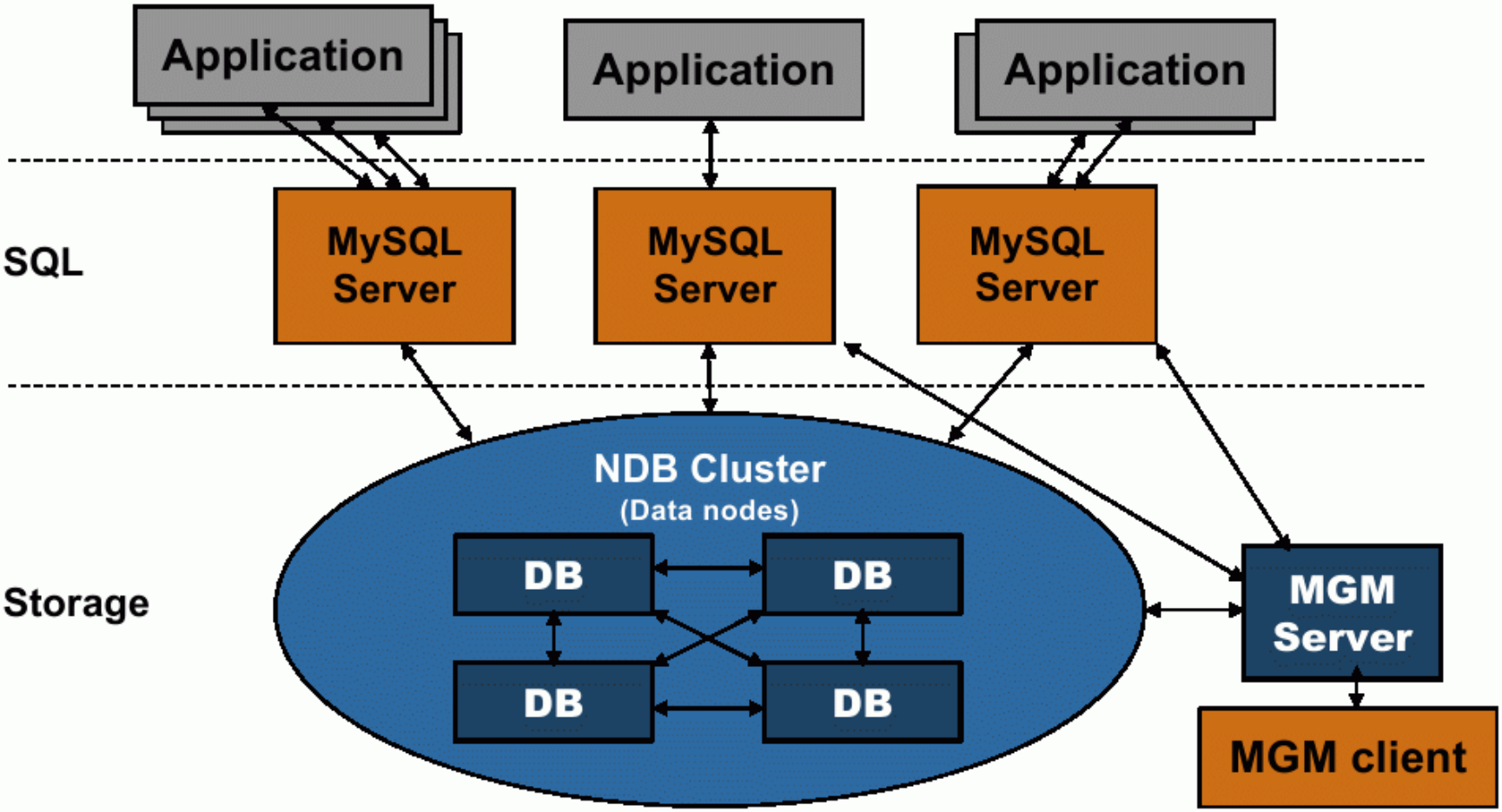
# MySQL Cluster

- Verteilte parallele Architektur begünstigt Skalierung
- Komplexe Abfragen (Komplexe JOINs oder full-table Scans) teuer
- Erfordert dediziertes Netzwerk mit geringer Latenz
- Gigabit-Ethernet oder Dolphin SCI

# MySQL Cluster

- In-memory indexes
- Keine Unterstützung für Fremdschlüssel
- Ungeeignet für lang laufende Transaktionen
- http://mysql.com/products/database/cluster/

# Cluster Components

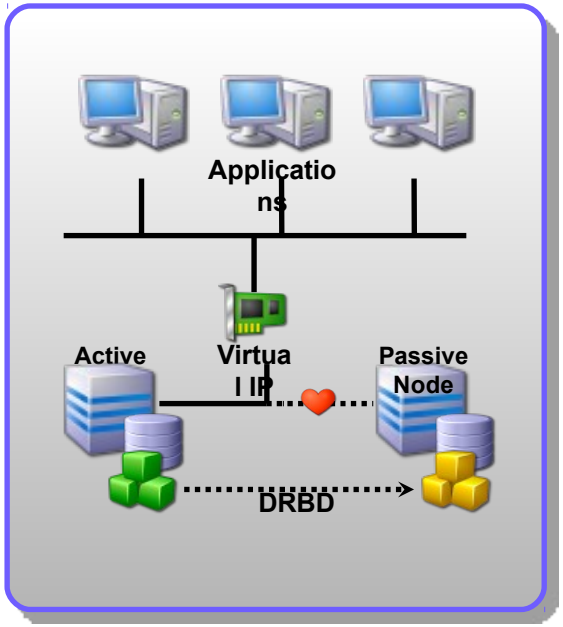


# DRBD

- Distributed Replicated Block Device
- “RAID-1 über das Netzwerk”
- Synchrone/asynchrone Block-Replizierung
- Automatische Resynchronisierung
- Applikations-agnostisch
- Kann lokale I/O-Fehler maskieren
- Aktiv/passiv-Konfiguration vorgegeben
- Dual-primary Modus (benötigt ein Cluster Dateisystem wie GFS or OCFS2)
- http://drbd.org/

# DRBD im Detail

- DRBD repliziert Datenblöcke zwischen zwei Plattenpartitionen
- DRBD kann mit Linux-HA und anderen HV-Lösungen gekoppelt
  werden
- MySQL läuft normal auf dem Primärknoten
- MySQL ist nicht aktiv auf dem Sekundärknoten
- DRBD ist nur für Linux verfügbar



# Galera Replication

- Patch für InnoDB plus externe Bibliothek
- Synchrone Replikation
- Single- oder Multi-Master
- Multicast-Replikation
- HA plus Lastverteilung möglich
- Zertifikat-basierte Replikationsmethode (anstatt 2PC)
- http://codership.com/products/mysql_galera

# PBXT Replikation

- PBXT Speicher-Engine
- MVCC, Transaktionen, Fremdschlüssel
- Log-basiert
- Asynchrone Replikation
- Ein Master, mehrere Slaves (Fan-out)
- Zeilenbasiert (PBXT-internes Format)
- http://www.primebase.org/

# Continuent Tungsten Replicator

- Datenbank-extern
- Asynchron, Master-Slave, Fan-out & Fan-in
- Java
- Log-basiert
- Ereignisse werden in Transaction History Log (THL) abgelegt
- Modulare Architektur (Pipelines, Stages)

# Fragen / Diskussion

# Vielen Dank!

Lenz Grimmer <lenz.grimmer@oracle.com>
http://www.lenzg.net/
@lenzgr


ORACLE®