BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
DRESDEN
UNIVERSITY OF COOPERATIVE EDUCATION

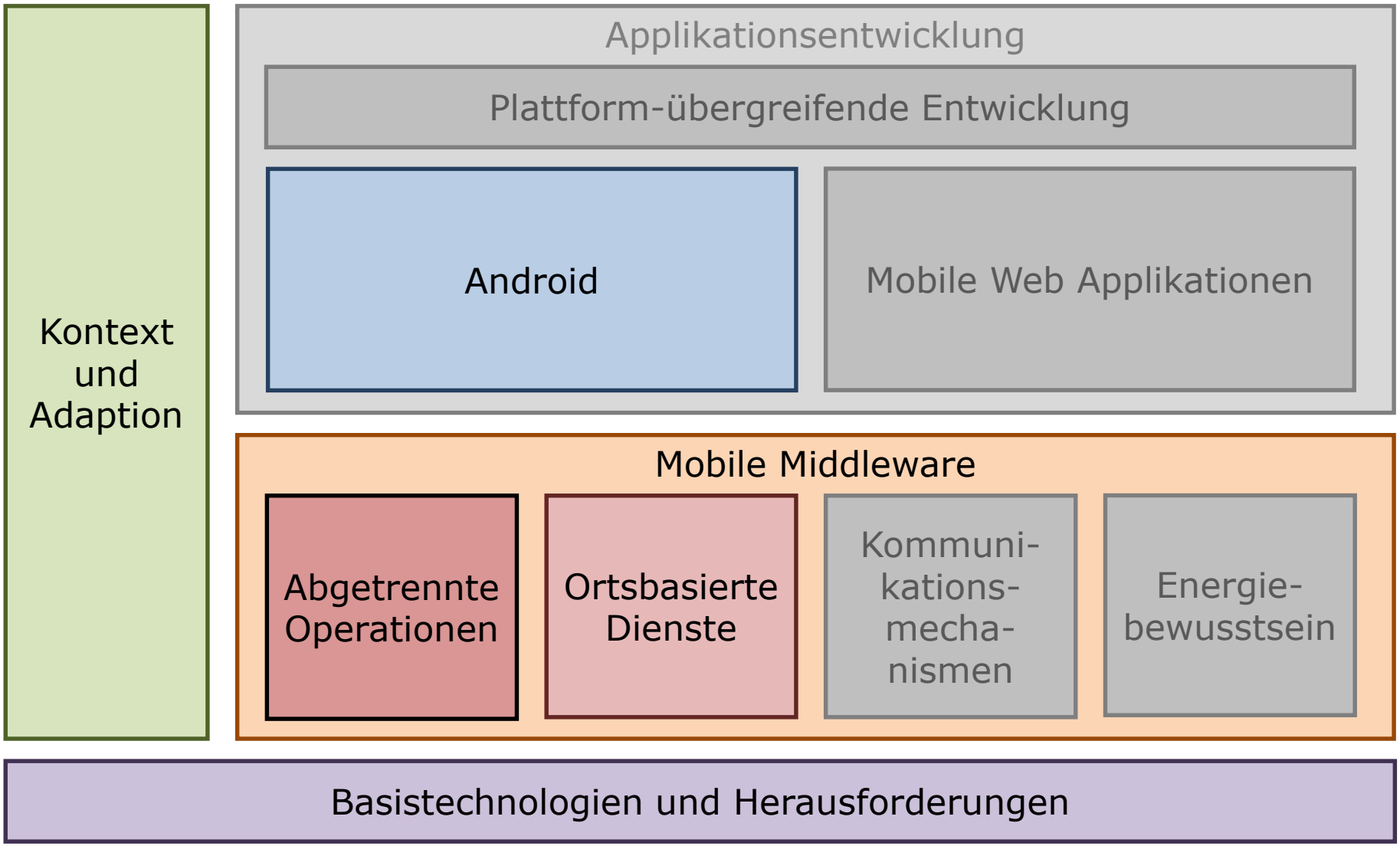
Web- und App-Programmierung

# Abgetrennte Operationen

mit Skriptmaterial von Dr.-Ing. T. Springer

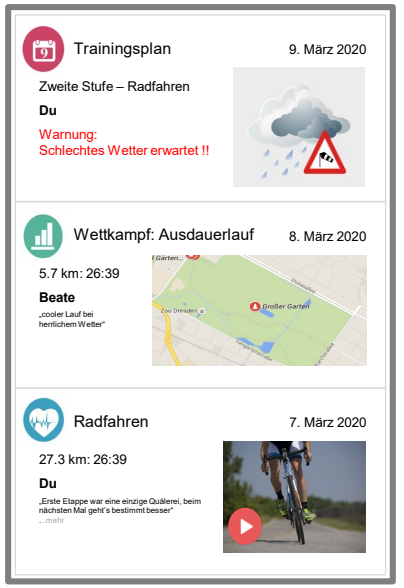
**Prof. Dr.-Ing. Tenshi Hara**
tenshi.hara@ba-sachsen.de

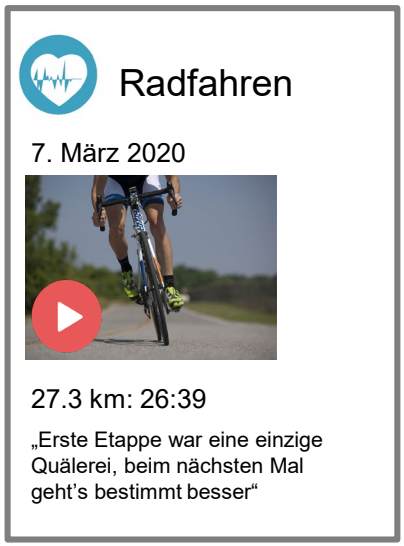
# Aufbau der Lehrveranstaltung



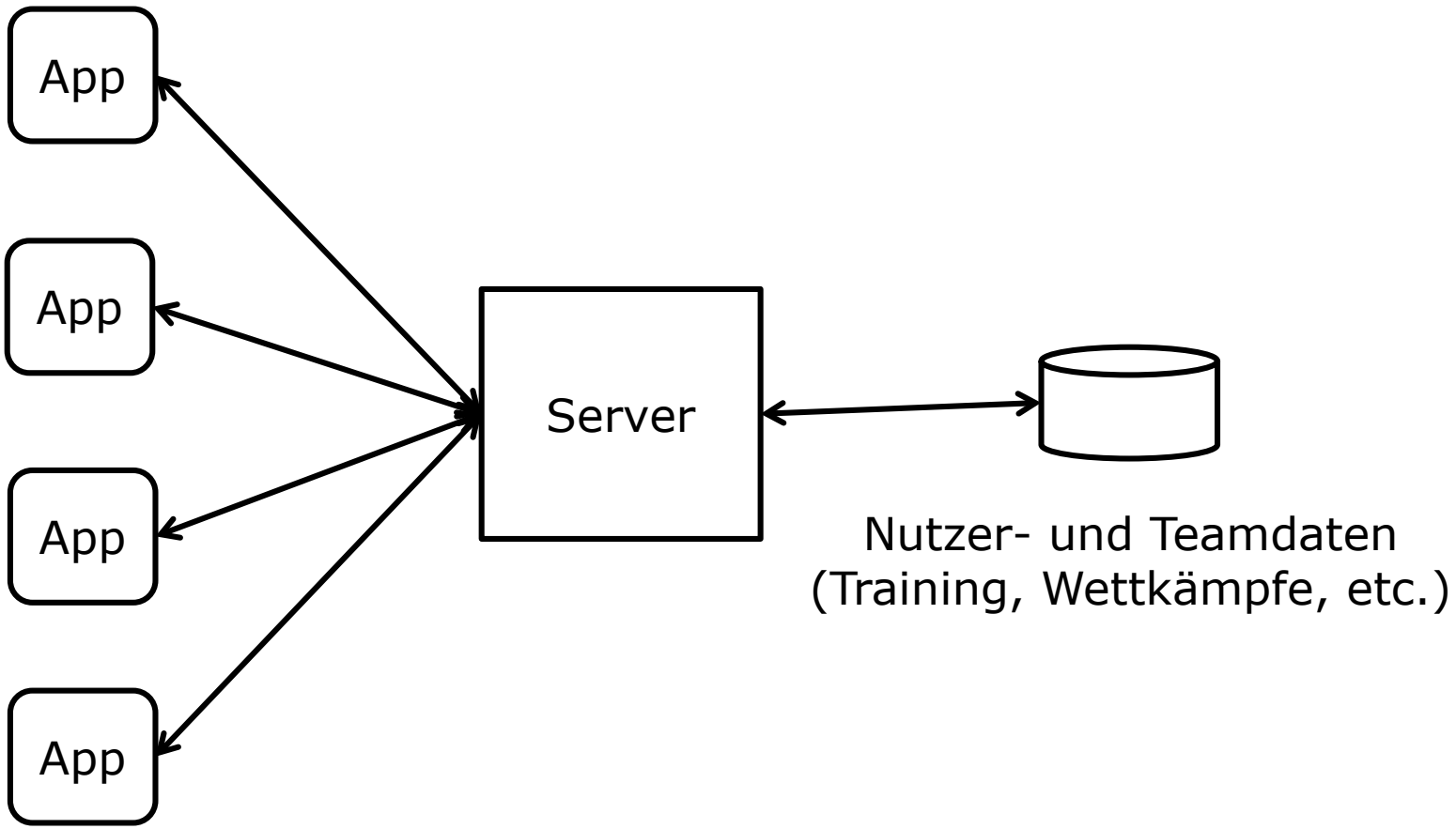
# Rückblick: motivierendes Beispiel

- Social-Fitness-App
  - Smart Watch + Smartphone/Tablet-App
  - Server-Komponente für Datenhaltung und Nutzerverwaltung
- Funktionalität
  - automatische Erkennung und Aufzeichnung von Sportaktivitäten
  - Aufzeichnung der Aktivitätsdaten und -inhalte (Bilder, Videos, Strecke, Pulskurve, etc.)
  - Aktivitätszeitlinie mit integrierten Medien: eigene Aktivitäten teilen und die anderer Nutzer sehen
  - Verwaltung des Trainingsplans und von Trainingsaktivitäten
  - Verwaltung von Wettkämpfen und Rangliste



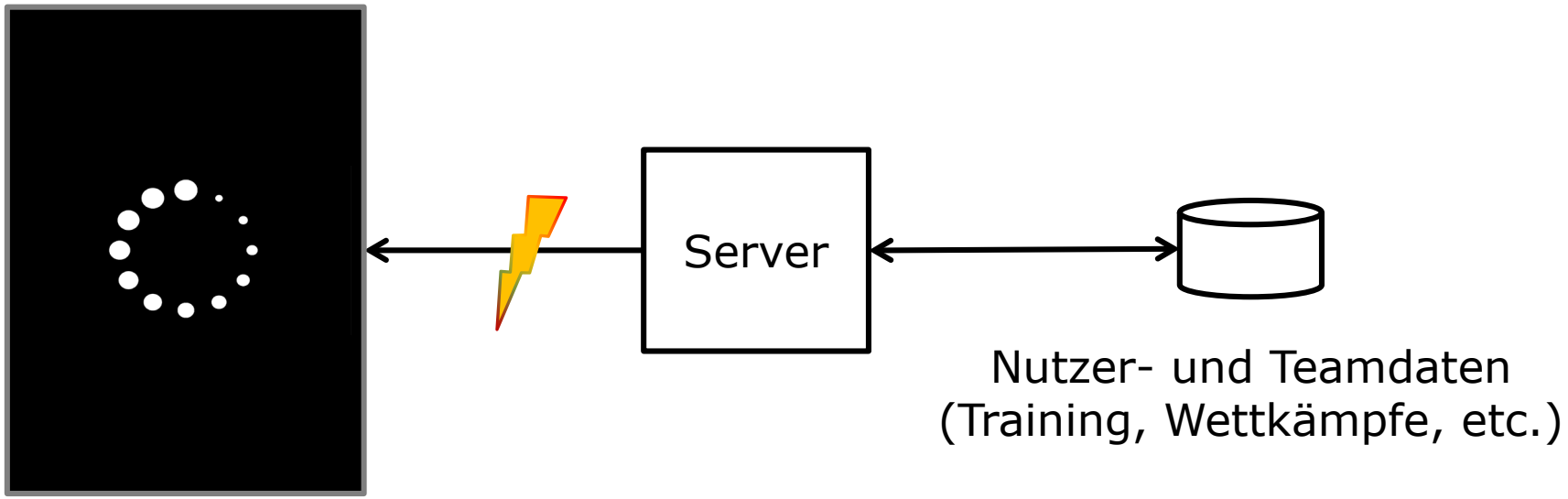



# Architektur der Beispiel-App



# Offline-Challenge der Social-Fitness-App – Teil 1

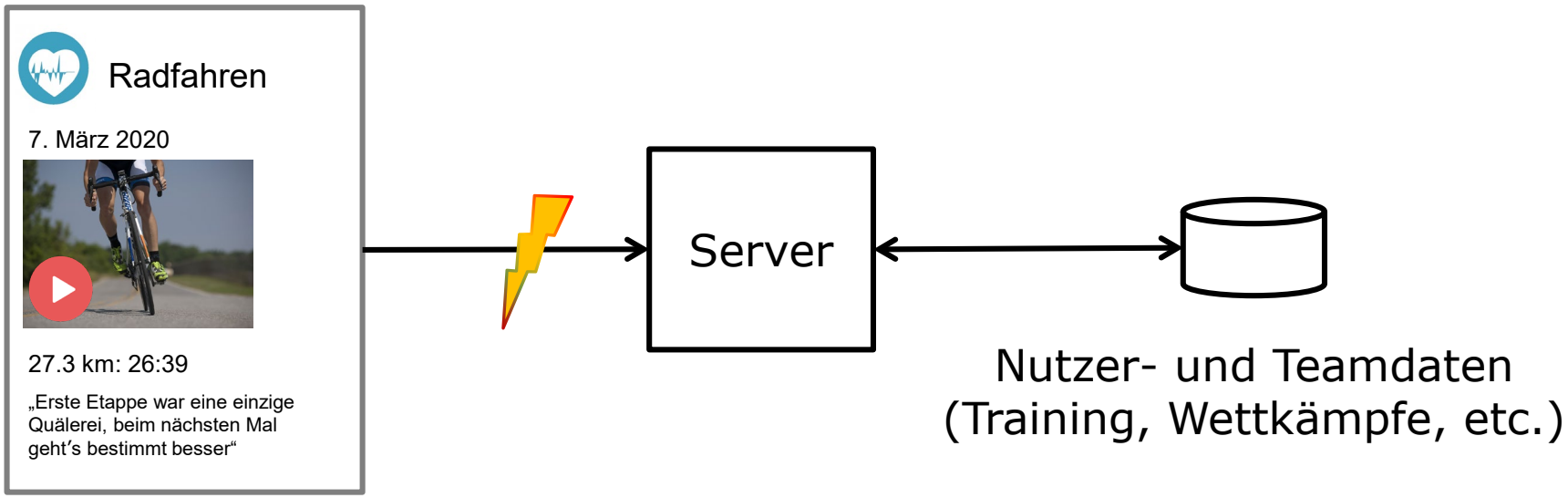
- Holen von Zeitlinieneinträgen




- App blockiert
- Anzeige veralteter Einträge in der Zeitlinie (aus dem Cache)

# Offline-Challenge der Social-Fitness-App – Teil 2

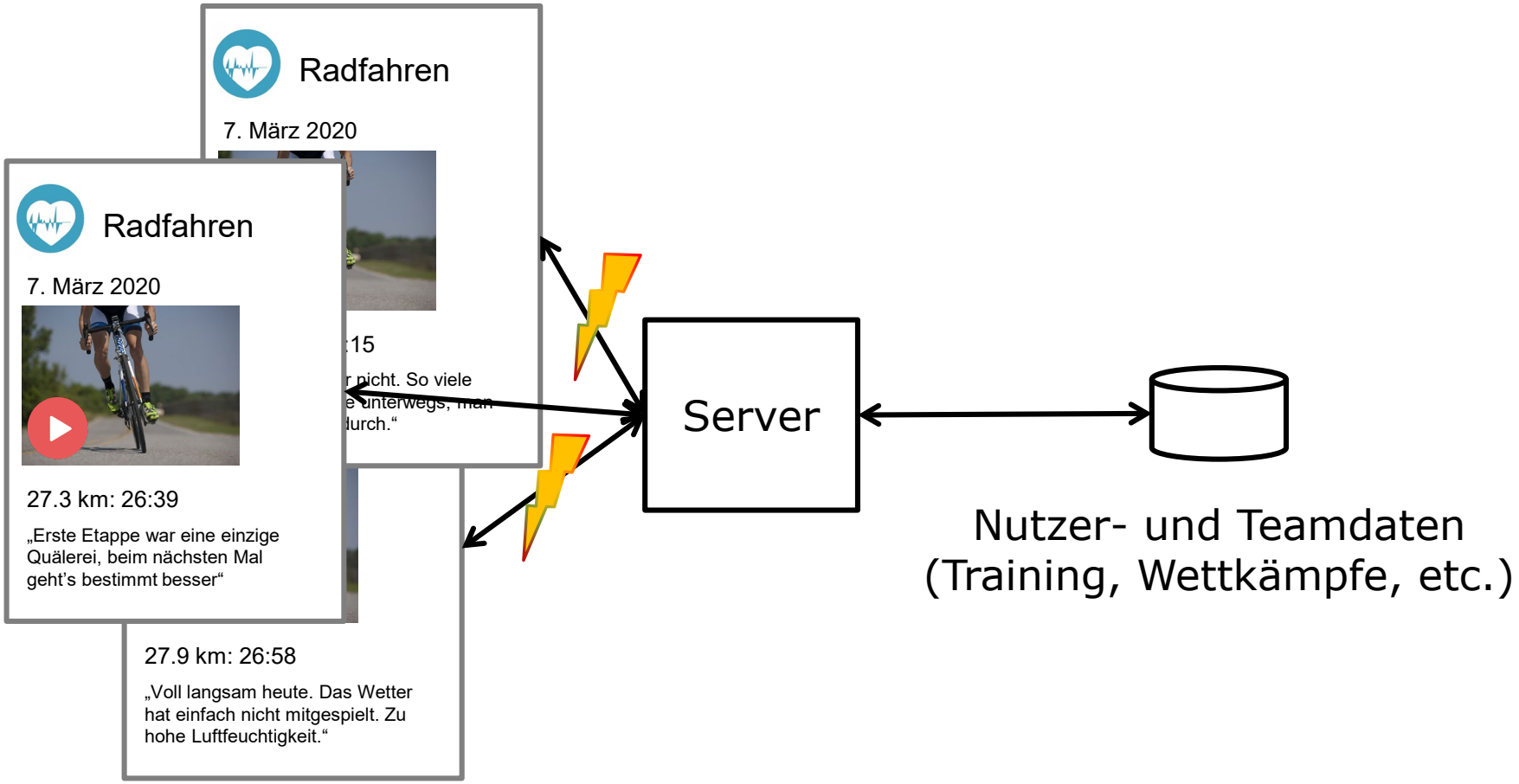
- Hochladen von Trainingsdaten und Medien




- lokales Zwischenspeichern bis Verbindung verfügbar wird
- Nutzer könnte im Glauben sein, dass Daten gesendet werden oder Kommunikationsaktivitäten stattfinden

# Offline-Challenge der Social-Fitness-App – Teil 3

- sich widersprechende Informationen für Team-Aktivitäten




- lokale Änderungen an veralteten Kopien
- Aktualisierungskonflikte wegen konkurrierender Änderungen

# Motivation

- Herausforderung: Abtrennung, d.h. längere Phasen ohne Netzzugriff
  - vorhersehbar (Energie, Kommunikationskosten, Ortsänderung, …)
  - unerwartet (regionale Abdeckung, Server-Ausfall, Netzwerküberlast)
- Ziele
  - Verfügbarkeit der Anwendungsfunktionalität und Daten während der Abtrennungsphasen → Logik und Daten müssen lokal verfügbar sein
  - Transparenz aus Nutzer-/Entwicklersicht
    - Nutzersicht: Anwendungsnutzung ohne Unterbrechungen
    - Entwicklersicht: Verbergen der Abtrennung vor der Anwendung

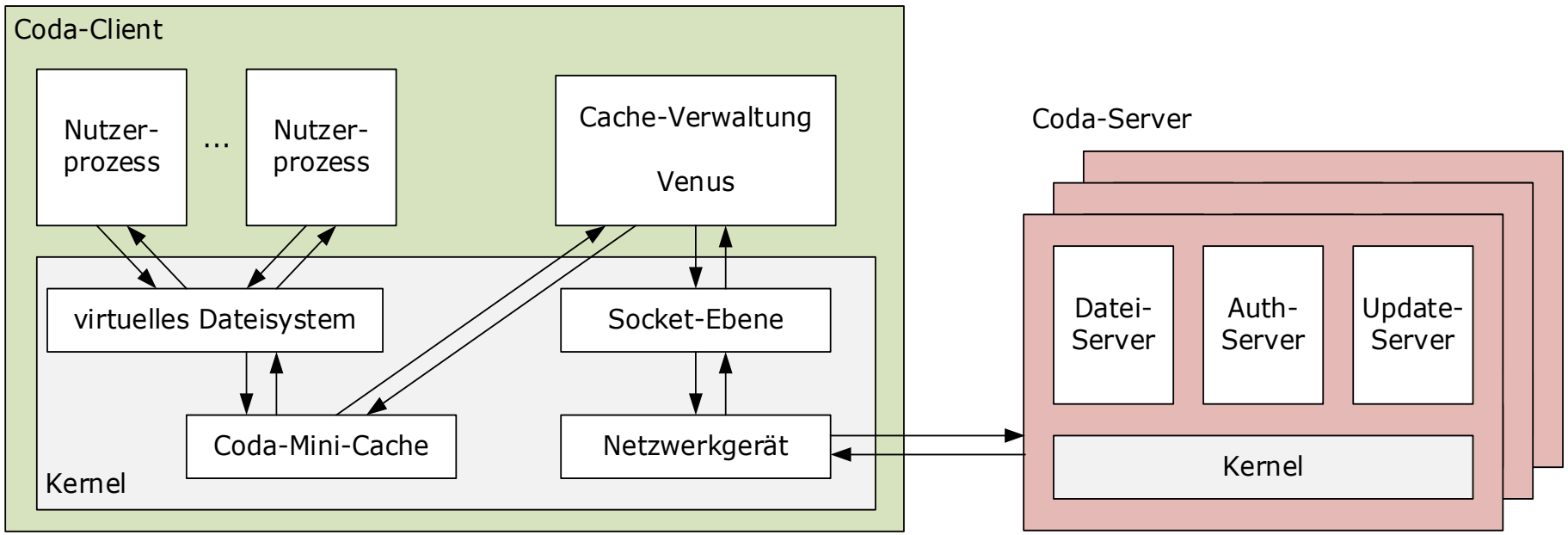
# Übersicht

- abgetrennte Operationen
  - Coda-Dateisystem
  - mobile Datenbanksysteme
  - Verallgemeinerung der Konzepte
- schwach korrelierte Operationen
  - Konzepte
  - schwach korrelierte Verbindungen in Coda
- Offline-First-Prinzip

# abgetrennte Operationen

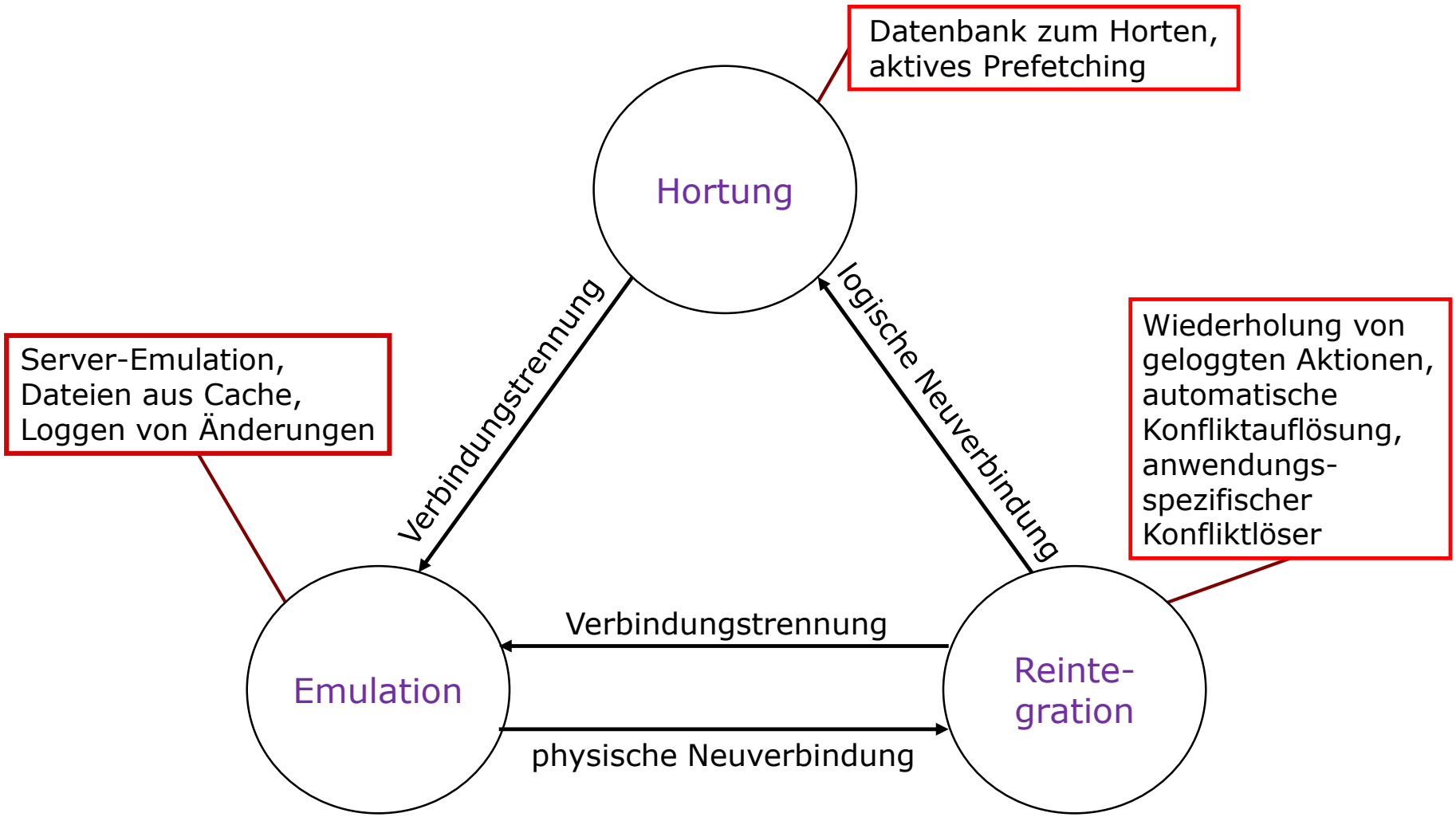
# CODA

- verteiltes Dateisystem mit Unterstützung von abgetrennten Operationen (Disconnection Support)
- Ziele: hohe Verfügbarkeit und Skalierbarkeit



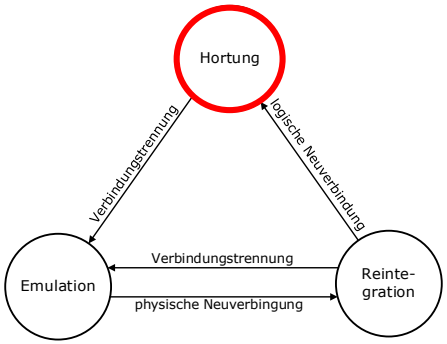

[Noble, Satyanarayanan; Carnegie Mellon University]

# Behandlung von Verbindungstrennungen im Coda Cache Manager




[Noble, Satyanarayanan; Carnegie Mellon University]

# CODA – VORBEREITUNGSPHASE



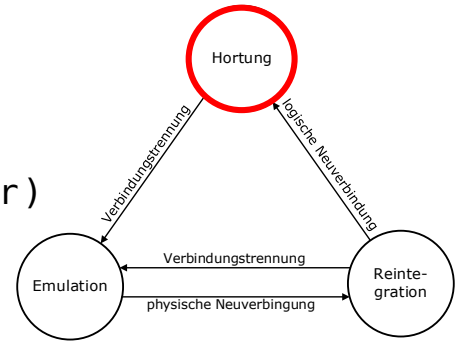

- Operationen während initialer Verbindung
  - direkter Zugriff auf entfernte Dateien (auf Server)
  - clientseitiges **Cachen ganzer Dateien**
  - **Durchschreiben von Änderungen**
- Horten
  - Prefetchen von Dateien für spätere Offline-Verfügbarkeit
  - Hoard Profile: Skript zur Festlegung einer zu cachenden Dateimenge
    - Festlegung der Wichtigkeit anhand einer Hoard Priority
    - werkzeuggestützte Definition von Hoard-Profilen

# CODA – BEISPIELHAFTES HOARD PROFILE

```
# persönliche Dateien
a /coda/usr/springer d+
a /coda/usr/springer/papers 100:d+
a /coda/usr/springer/papers/mobisys 1000:d+

# Systemdateien
a /usr/bin 100:d+
a /usr/etc 100:d+
a /usr/include 100:d+
a /usr/lib 100:d+
a /usr/local/gnu d+
a /usr/local/rcs d+
a /usr/ucb d+
```

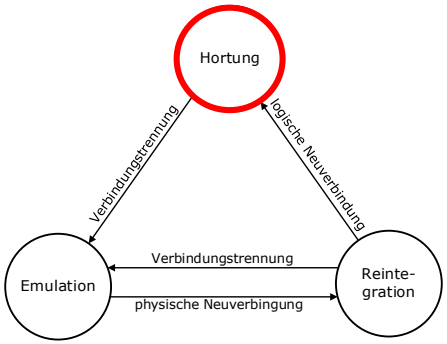
```
# X11-Dateien
# (vom X11-Maintainer)
a /usr/X11/bin/X
a /usr/X11/bin/Xvga
a /usr/X11/bin/mwm
a /usr/X11/bin/startx
a /usr/X11/bin/xclock
a /usr/X11/bin/xinit
a /usr/X11/bin/xterm
a /usr/X11/include/X11/bitmaps c+
a /usr/X11/lib/app-defaults d+
a /usr/X11/lib/fonts/misc c+
a /usr/X11/lib/system.mwmrc
```




```
# Venus-Quelldateien (geteilt unter Coda-Entwicklern)
a /coda/project/coda/src/venus 100:c+
a /coda/project/coda/include 100:c+
a /coda/project/coda/lib c+

\d{0,4}:? .. Priorität (umso größer der Wert, umso höher die Priorität)
c         .. direkte Kinder
d         .. alle Kinder
\+        .. aktuelle und zukünftige Dateien
```

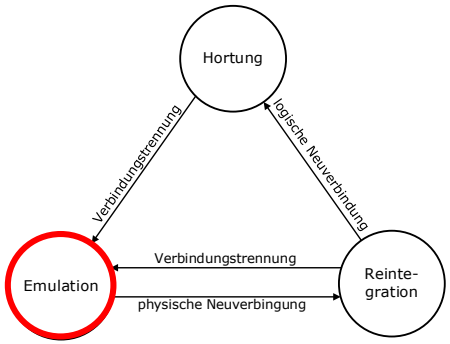
# CODA – VORBEREITUNGSPHASE




- priorisierter Caching-Algorithmus
  - Balance zwischen
    - kurzfristigem Bedarf für Performanz und
    - langfristigem Bedarf für Verfügbarkeit
  - Cache-Verwaltungsstrategie ist Least Recently Used (LRU) in Kombination mit geräteindividuellen Dateiprioritätslisten (Hoard Database)
  - Verwaltung von Prioritäten (Hoard Priority, Zugriffsstatistiken)

- Hortbegehung (Hoard Walking)
  - periodische Cache-Aktualisierungen oder bei Bedarf
  - Phase 1: Dateimengen aktualisieren (bspw. neue Dateien in von anderen Clients angelegten Ordnern erkennen)
  - Phase 2: Reevaluation aller Prioritäten im Cache und in der Hoard Database, dann Aktualisierung aller gecacheten Datei, die entfernt verändert wurden

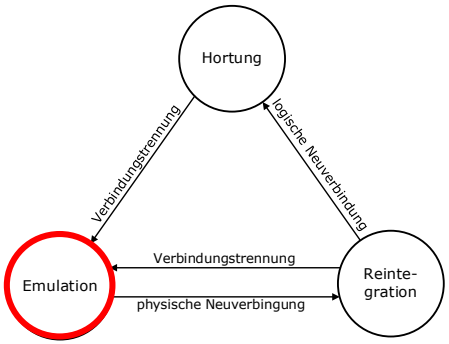
# CODA – ABTRENNUNGSPHASE




- Operationen während einer Verbindungstrennung
  - kein Zugriff auf entfernten Datei-Server
  - Operationen nur auf **lokal gecacheten Dateien**
  - Fehlermeldungen beim Zugriff auf nicht gecachete Dateien

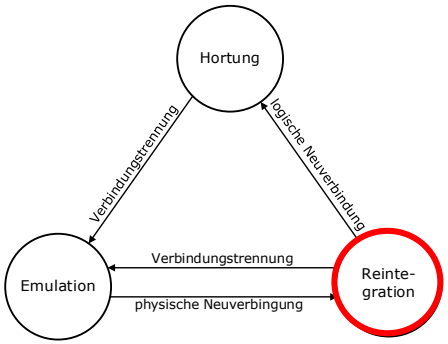
- Emulation
  - virtuelles Dateisystem emuliert Datei-Server-Operationen
  - holt Datei vom lokalen Venus-Cache-Verwalter
  - Änderungen werden als lokale Kopie geschrieben
  - lokale Dateiänderungen werden lokal sofort sichtbar
  - Versionsverwaltung für spätere Konflikterkennung
  - Loggen aller Änderungen um diese später zum entfernten Server übertragen und dort wiederholen zu können
  - lokales Änderungs-Log (Change Modify Log, CML) enthält Kopie aller durchgeführten Operationen und Versionszustand beteiligter Dateien

# CODA – TRENNUNGSPHASE



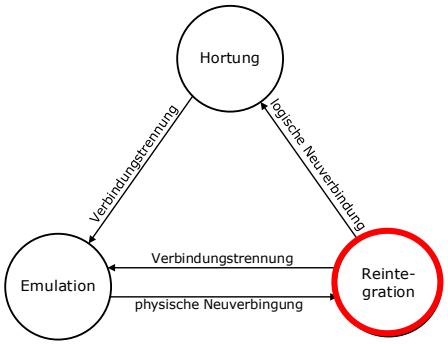

- Change Modify Log
  - enthält Liste von Änderungssätzen, insbesondere lokal durchgeführte Änderungsoperationen
  - jeder Änderungssatz enthält eine Kopie der durchgeführten Dateioperationen sowie den eingehenden und den ausgehenden Dateiversionszustand jeder beteiligten Datei
- Optimierung des Änderungs-Logs
  - Operationen, welche die Effekte vorheriger Operationen überschreiben verdrängen die entsprechenden Log-Einträge (bspw. verdrängt eine Löschoperation alle vorherigen Anlege- und Schreiboperationen auf diese Datei)
  - reduziert Speicherbedarf des CML
  - erhöht Reintegrationsperformanz

# CODA – REINTEGRATIONSPHASE




- Operationen bei Neuverbindung
  - direkter Zugriff auf entfernten Datei-Server wieder möglich
  - **Log der lokalen Änderungen abgleichen**
  - **geänderte und veraltete Dateien im Cache erkennen**
- Reintegration
  - geloggte Operationen an Server übertragen und dort wiederholen
  - Konflikterkennung durch Versionsvergleich
  - mögliche Konflikte
    - parallele Dateiänderung auf Client und Server
    - gleicher Name für neu erstellte Dateien/Ordner
    - parallele Änderung von Datei/Ordner-Attribut auf Client und Server

# CODA – REINTEGRATIONSPHASE




- Reintegration
  - automatische Konfliktauflösung
    - generische Konfliktauflösung
    - anwendungsspezifische Konfliktauflösung (durch Anwendungsentwickler bereitzustellen; Wissen über Bedeutung von Dateiinhalten notwendig)
  - Werkzeuge zur manuellen Konfliktauflösung
    - falls automatische Auflösung fehlschlägt
    - falls im einzelnen so spezifiziert oder gewünscht

# Mobile Databanksysteme

- eingebettete mobile Datenbanken (Embedded Mobile Database)
  - lokale, leichtgewichtige Datenbank wird in Anwendung eingebettet
  - Beispiel: SQLite, Couchbase Mobile
- mobile Client/Server-Datenbanken
  - leichtgewichtige Datenbank synchronisiert mit Server-Datenbank
  - Beispiele: DB2 Everyplace, Oracle 10g lite, Tamino Mobile Suite

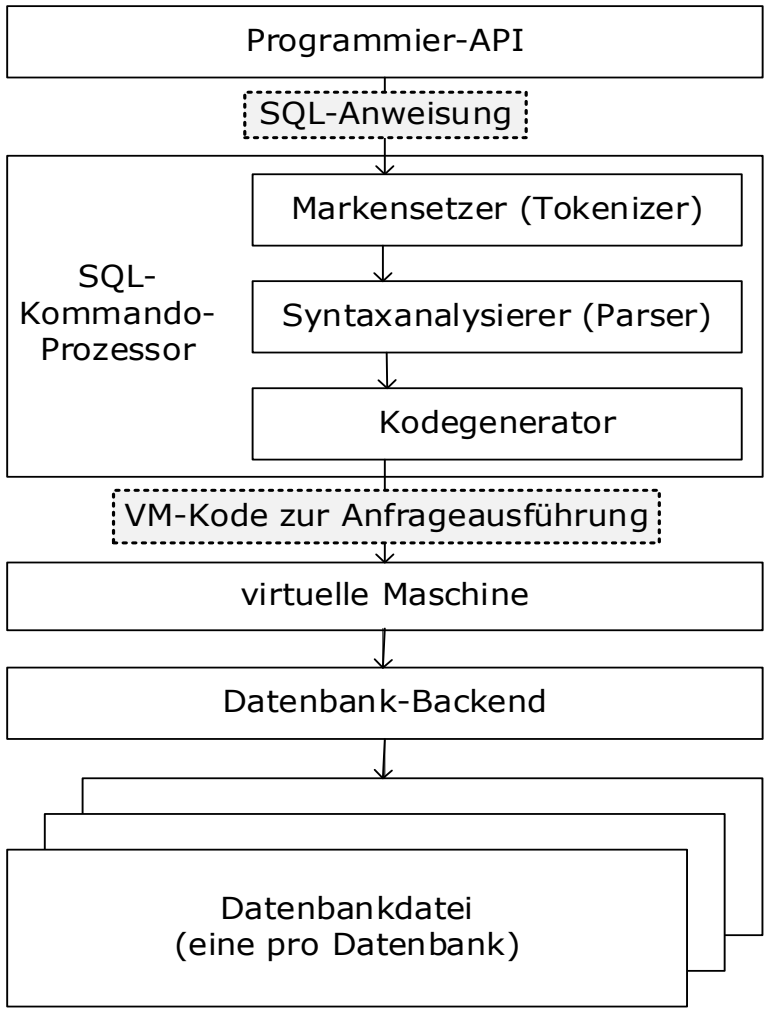
# EINGEBETTETE MOBILE DATABANKSYSTEME

lokaler Datenzugriff

+ Persistenz für Applikationsdaten
+ simple Programmier-API
+ läuft auf lokalem Gerät; in Anwendung eingebettet
+ hängt nicht von bestehender, permanenter Netzwerkverbindung ab
+ kleiner Fußabdruck
+ gute Performanz
– Datenbank in Zuständigkeit einer Anwendung
  → kein gemeinsamer Datenzugriff zwischen Anwendungen
  → keine Synchronisierung mit Backend

# BEISPIEL: SQLITE

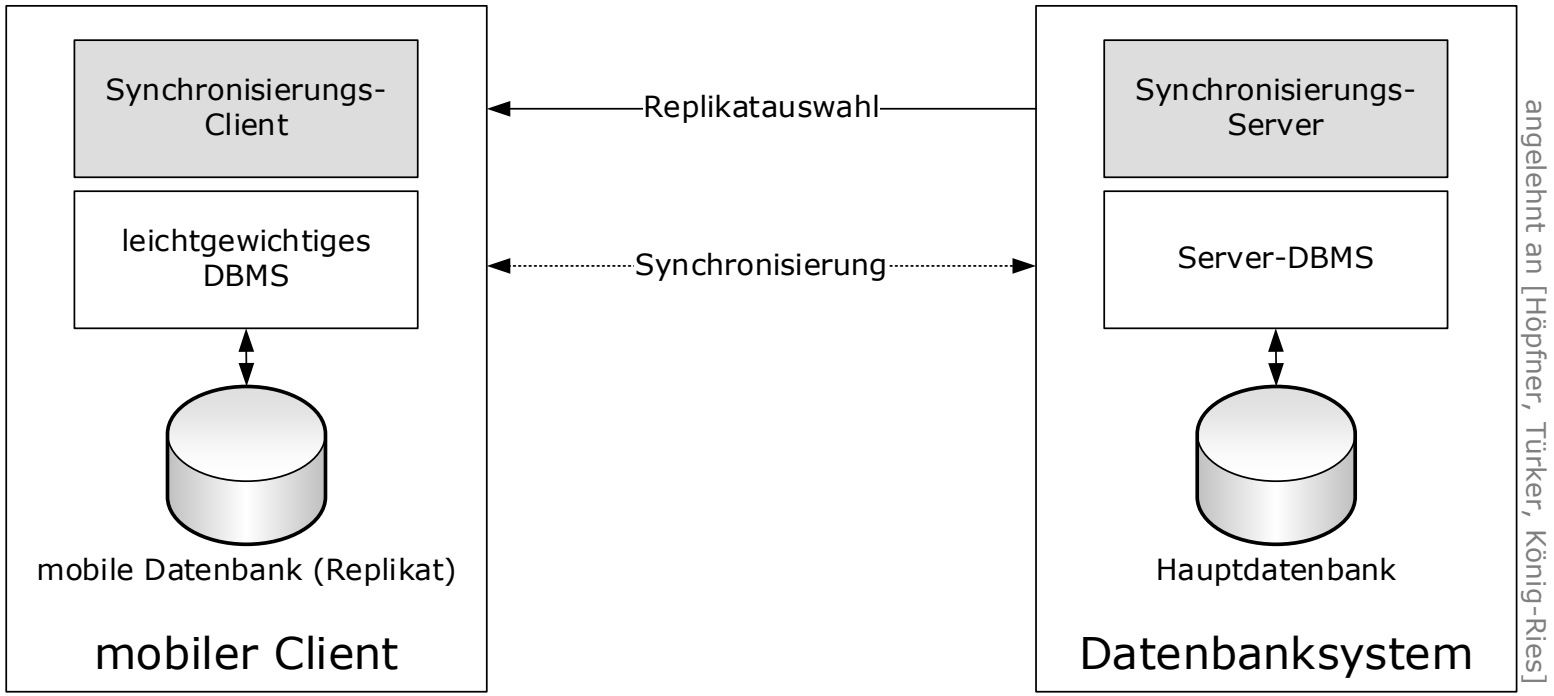
- eingebettete SQL-Datenbankumgebung
- serverlos; schreibt direkt auf Dateien im Persistenzspeicher
- Eigenschaften
  - unterstützt lokale ACID-Transaktionen, auch geschachtelte
  - setzt Großteil von SQL92 um
  - abgeschlossen (ohne externe Abhängigkeiten)
  - plattformübergreifend (DB-Datei beliebig verschiebbar)
  - kleiner Fußabdruck
    - voll konfiguriert: <300KB
    - nur Notwendigstes: <180KB



# Mobile Client/Server-Databanken

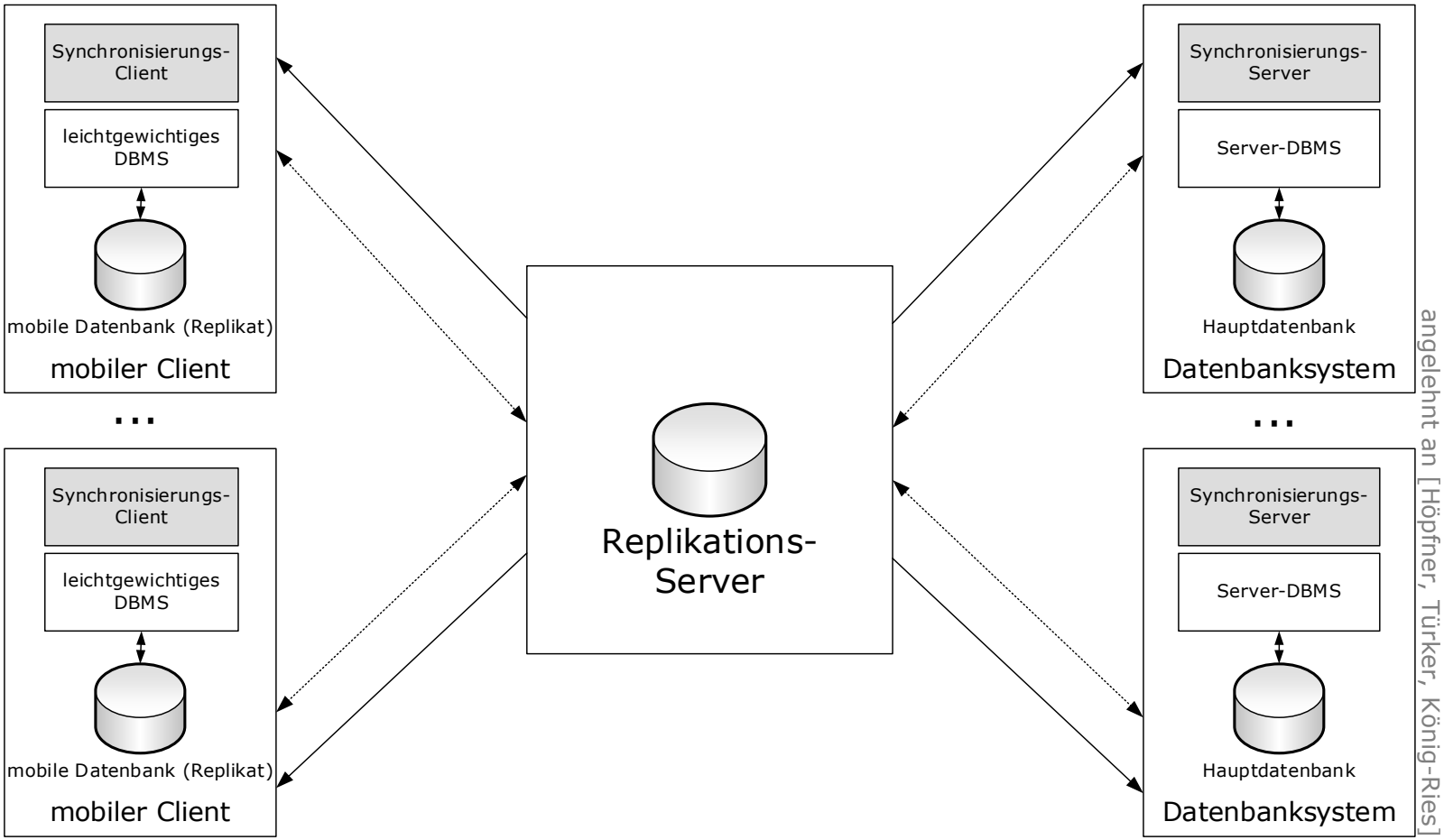
- Datenbanksystem mit über mobile und stationäre Geräte verteilten Komponenten
- zwei Architekturkonzepte
  1. erweitertes Client/Server-System
  2. Middleware-System mit Replikations-Server

# ERWEITERTES CLIENT/SERVER-SYSTEM




- auf Client leichtgewichtiges DB-Replikat mit Untermenge der Hauptdatenbank
- DB-Replikat verwaltet bei Netzwerktrennung alle Operationen lokal (Server-Emulation)
- keine ständige Neusynchronisierung, stattdessen
  - durch Client ausgelöst, oder
  - periodisch vom System angestoßen

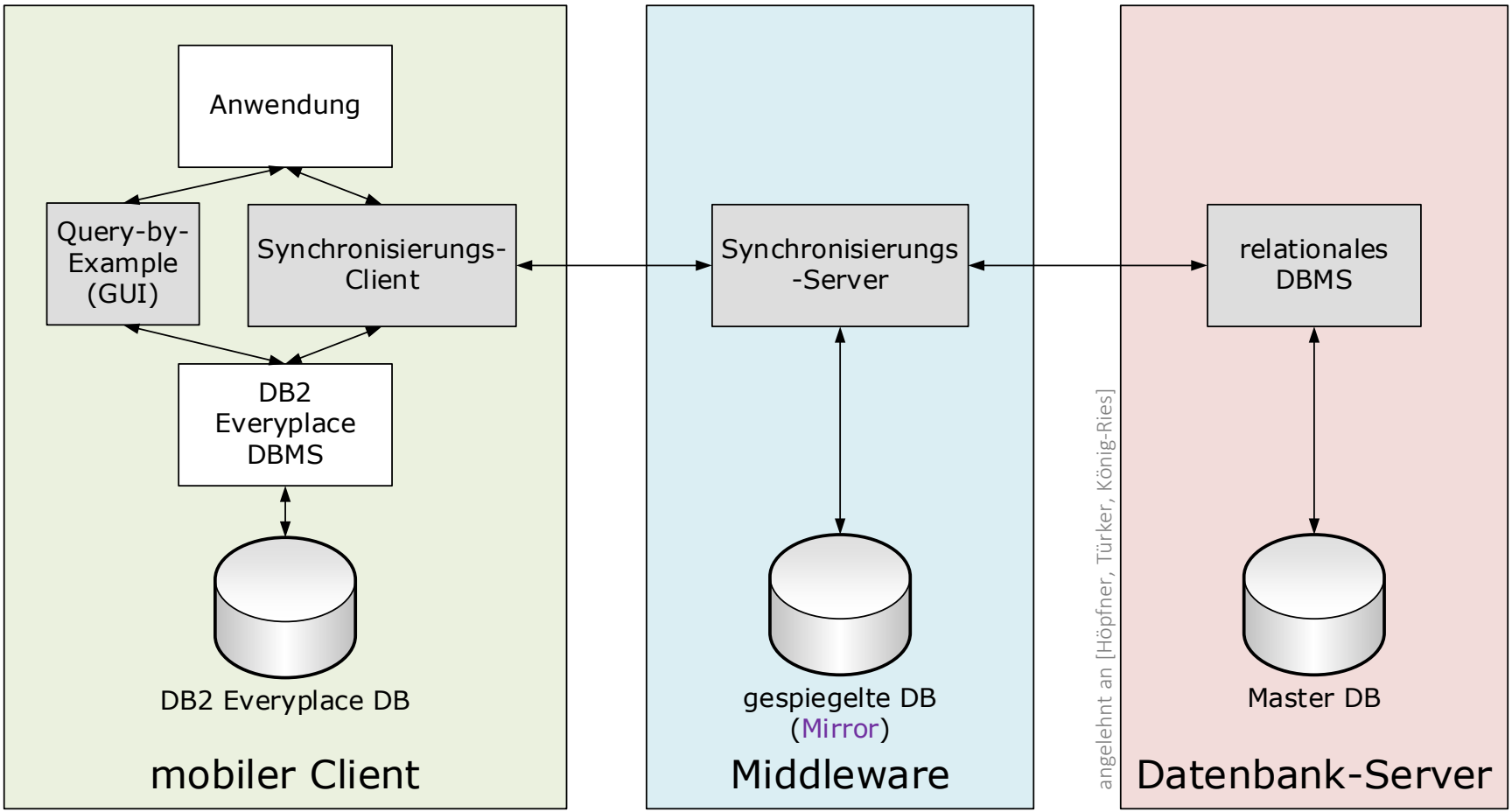
# Middleware-System mit Replikations-Server




vermittelnder Middleware-Server für Replikationsverfügbarkeit

- entkoppelt mobile DB und Haupt-DB – multiple Quellen werden transparent integriert
- Entlastung der einzelnen Datenquellen
- Zugriffskontrolle

# DB2 Everyplace

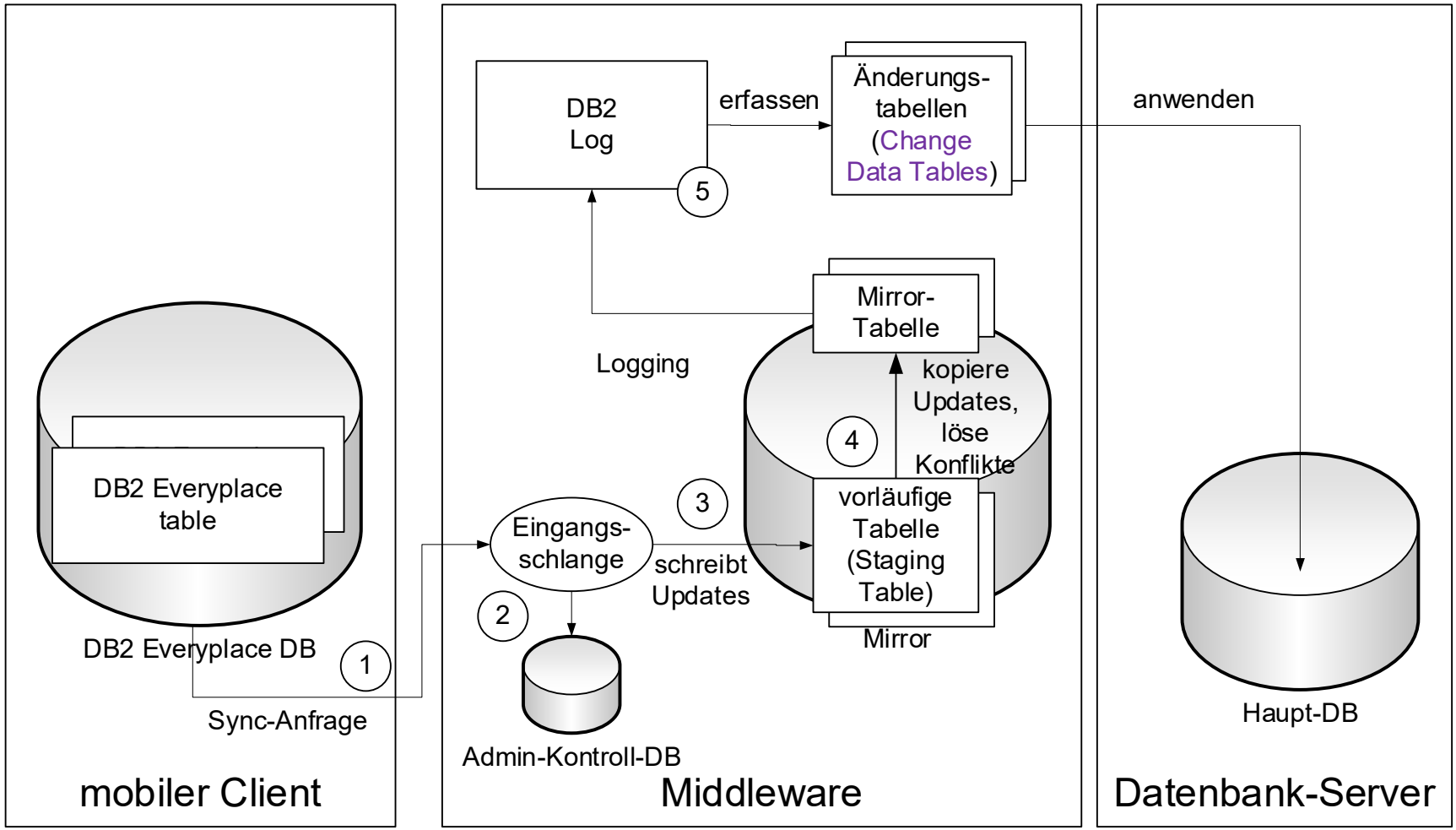



Architektur: Middleware-System mit Replikations-Server

# REPLIKATION

- Werkzeuge
  - Mobile Device Administration Center (MDAC)
  - XML Scripting Tool
- ermöglicht Festlegung von zu synchronisierenden Teilen der Haupt-DB
  - auf Tabellenebene
  - Spezifikation horizontaler oder vertikaler Filter zur spezifischen Spalten- und Reihenselektion
  - Zieltabelle kann aus Werten multipler Quelltabellen gebildet werden (Join, Merge) und eigene Berechtigungen besitzen (R, W oder RW)
  - Beschränkungen bei RW-Berechtigung
    - keine Aggregation bei Zeilen/Spalten-Selektion
    - Quelltabelle darf selbst nicht Ergebnis eines Joins/Merges sein
    - reine Tabellenreplikation (weder Integrität noch Trigger)

# Synchronisierung lokaler Änderungen mit der Haupt-DB



# KONFLIKTBEHANDLUNG

- Erkennung von Konflikten anhand Versionsnummern
  - Sync-Server und Sync-Client verwalten für jede Tabellenzeile ihre Versionsnummern
  - unterschiedliche Versionsnummern dienen als Konfliktindikator
- Konfliktbehandlung
  - Konflikt ignorieren → schlecht!
  - simple Behandlung: Änderungen ablehnen, Transaktionen rückgängig machen, Loggen des Konfliktes, Datenstand der mobilen DB als Konfliktversion markieren, mobilen Datenbestand mit Daten des Mirrors überschreiben, ggf. Nutzer über Konflikt benachrichtigen
  - nutzergesteuerte Konfliktbeseitigung: Anzeigen von Unterschieden, ggf. Vorschläge für Zurückrollen oder Vereinigen, Auswahl des gültigen Versionsstandes durch Nutzer, Überschreiben der verworfenen Version mit der ausgewählten

# Abgetrenntes Arbeiten mit mobilen Datenbanken

- Horten
  - Replikation ausgewählter/aller Daten der Haupt-DB
  - explizite Auswahl durch Administrator in einem Werkzeug
  - Prefetchen von Daten anhand definierten Schemas
- Emulation
  - Anwendung nutzt lokale Datenbank zur Datenmanipulation
  - auch dann, wenn eine Verbindung besteht
- Reintegration
  - Synchronisierung der lokalen DB mit der Haupt-DB
  - explizit oder periodisch ausgelöst

# COUCHBASE

- NoSQL-Datenbank mit Synchronisierungsunterstützung
- Couchbase Lite
  - eingebettete Datenbank mit CRUD- und Abfrageunterstützung
  - JSON und binäre Datenrepräsentationen
  - Abfragen basierend auf N1QL
- Synchronisierungs-Gateway
  - Synchronisierung zwischen Couchbase Lite und Couchbase Server
  - Peer-to-Peer-Synchronisierung zwischen Couchbase-Lite-Instanzen
  - läuft auf Server oder in Cloud
- Datenzugriff
  - ReST und ereignisbasierte APIs
  - JSON-Ein- und -Ausgabe

Couchbase
www.couchbase.com

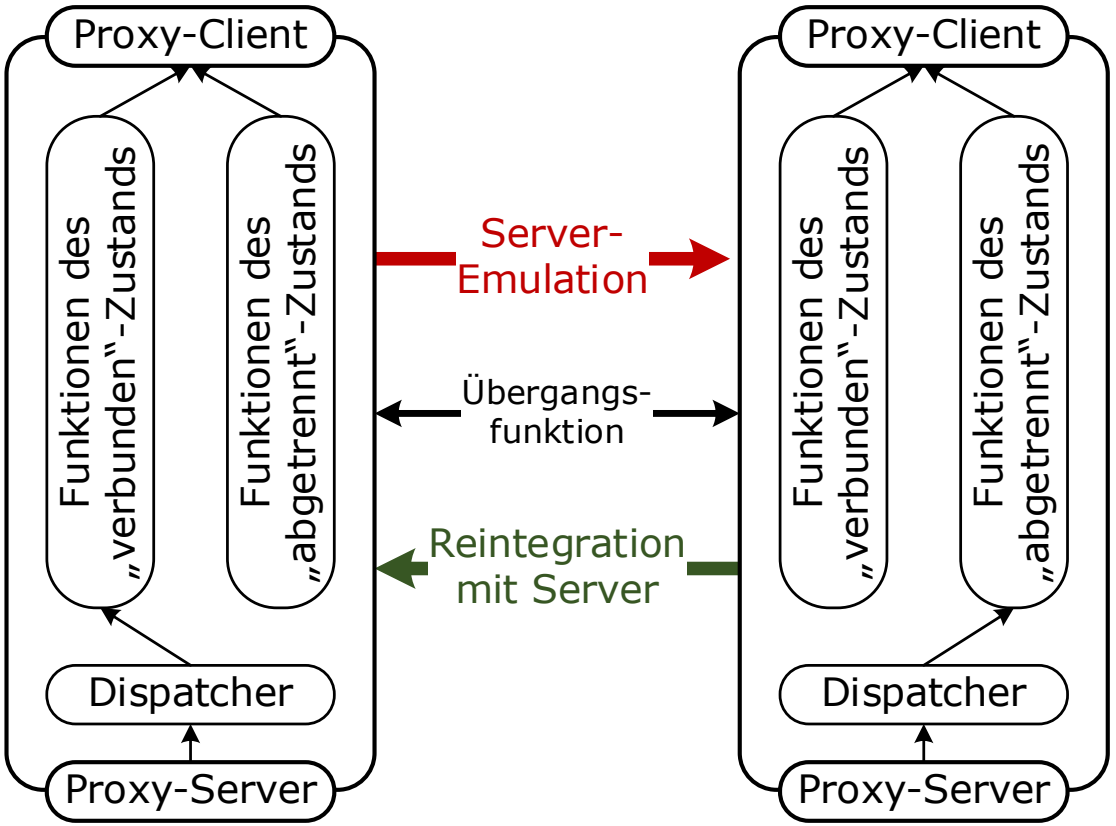
# REALM

- Kernabstraktion: leichtgewichtiger Objektcontainer (Realm)
- Realm-Plattform synchronisiert Realme innerhalb der Plattform
- Realm-Plattform besteht aus
    - Reals Object Server
    - Client-SDK für native Applikationen

realm
realm.io

# ABGETRENNTE OPERATIONEN – PRINZIPIEN

- Aktions-überwachung
- Caching, Prefetching

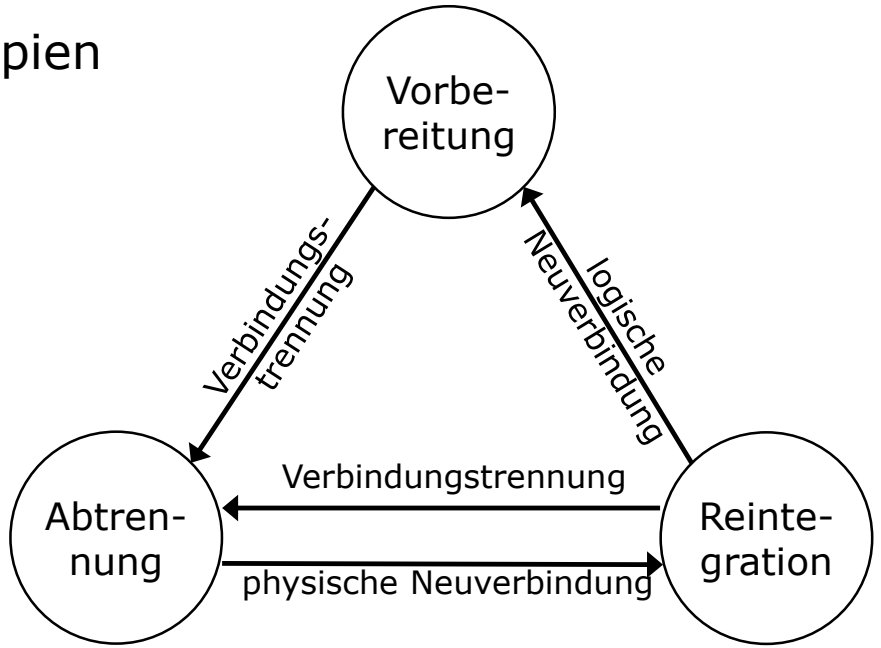



- Änderungen an Datenkopien
- Aufzeichnung von Aktionen

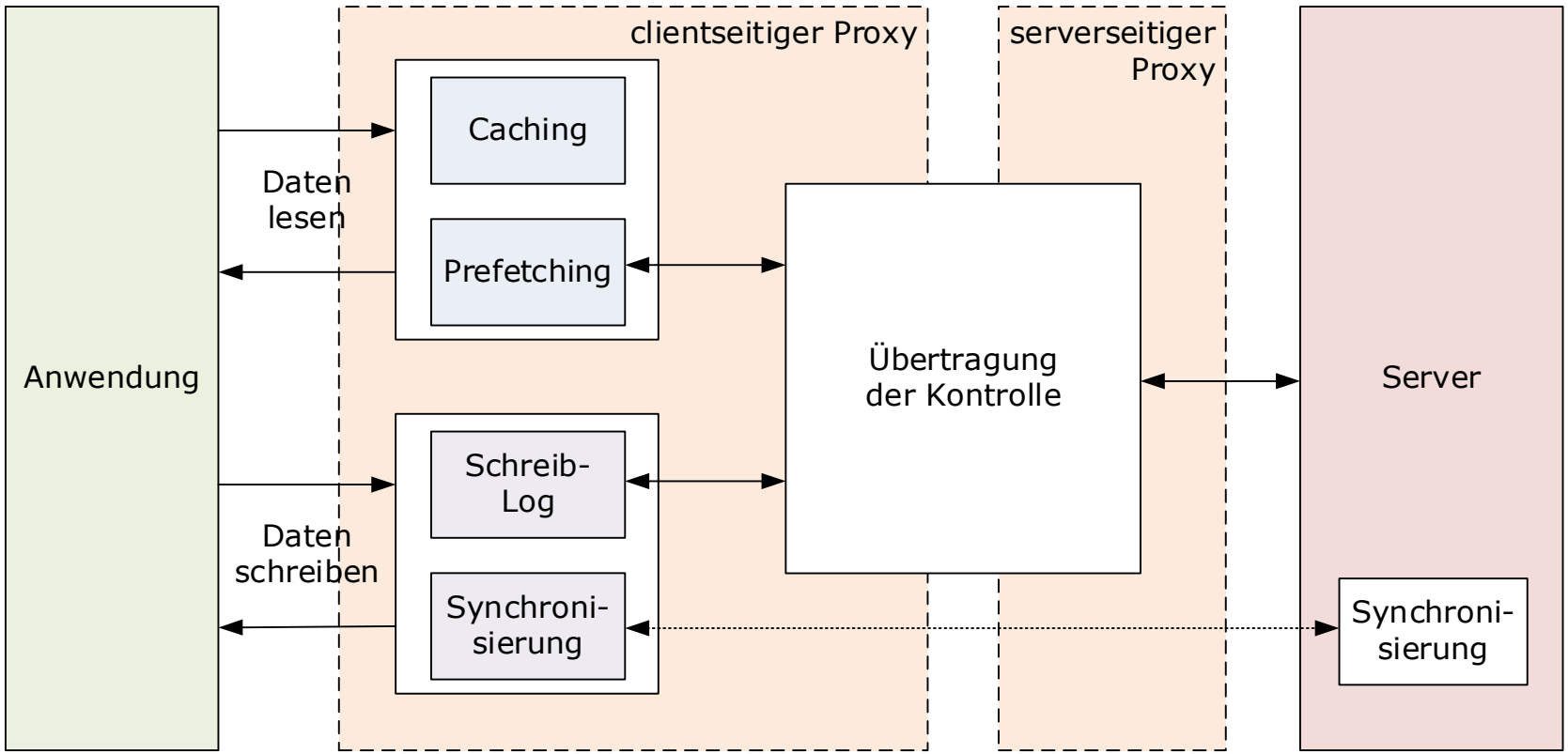
- Einführung eines Doppel-Proxys
- Funktionsimplementierungen für verbundenen und abgetrennten Zustand (`f_connected()`, `f_disconnected()`)
- Übergangsfunktion für Zustandwechsel zwischen verbunden und abgetrennt

# ABGETRENNTE OPERATIONEN – GENERALISIERUNG

- Vorbereitungsphase → lokale Verfügbarkeit von Dateien/Kode sicherstellen
  - Datenreplikation und Caching
  - Nutzeraktivitäten überwachen und Prefetching
  - statische oder dynamische Installation emulierten Kodes auf Client
- Abtrennungsphase → lokale Server-Emulation
  - Änderungen auf lokalen Datenkopien
  - Aufzeichnen von Aktionen
- Reintegrationsphase
  - lokale Änderungen an Server vermitteln
  - Erkennung und Behebung von Konflikten



# GENERALISIERTE ARCHITEKTUR



# schwach korrelierte Operationen

# Grenzen abgetrennter Operationen

abgetrennte Operationen haben Grenzen

- begrenzte Aufgabenfortführung mangels Server-Zugriff
- lokale Änderungen bleiben entfernten Partnern bis zur Synchronisierung verborgen
- Konfliktwahrscheinlichkeit steigt mit zunehmender Abtrennungszeit
- lokale Ressourcen für Caching, Loggen und Server-Emulation nötig
- schlechte Performanz und Antwortzeit auf schmalbandiger Verbindung

# SCHWACH KORRELIERTE OPERATIONEN

Verbesserung zur abgetrennten Operation

- Operationsmodus zum Verschleiern von Netzwerklatenz
- Minimierung von Abtrennungszeiten
- geringere Ressourcenanforderungen auf dem Client
- erhöhte Konsistenz und Aufgabenfortführung
- geringere Konfliktwahrscheinlichkeit

⇒ schmalbandige Verbindungen benötigen aber explizite Unterstützung!

# Unterstützung schwach korrelierter Operationen

- Mechanismus zur adaptiven und dynamischen Verbindungsressourcennutzung
  - Laden neuer Daten und Cache-Aktualisierungen
  - Reintegration lokaler Änderungen
- Laden angeforderter Daten hat höchste Priorität
  - Unterstützung der Aufgabenfortführung
  - kurze Antwortzeiten
- Cache-Aktualisierungen und Reintegration unabhängig von Antwortzeit
  - schnelle Reintegration verkleinert lokales Operations-Log und Konfliktwahrscheinlichkeit
  - langsamere Reintegration ermöglicht Optimierung des lokalen Operations-Logs (bspw. Entfernen überlagerter Operationen)
    → weniger über das Netzwerk zu transferierende Daten

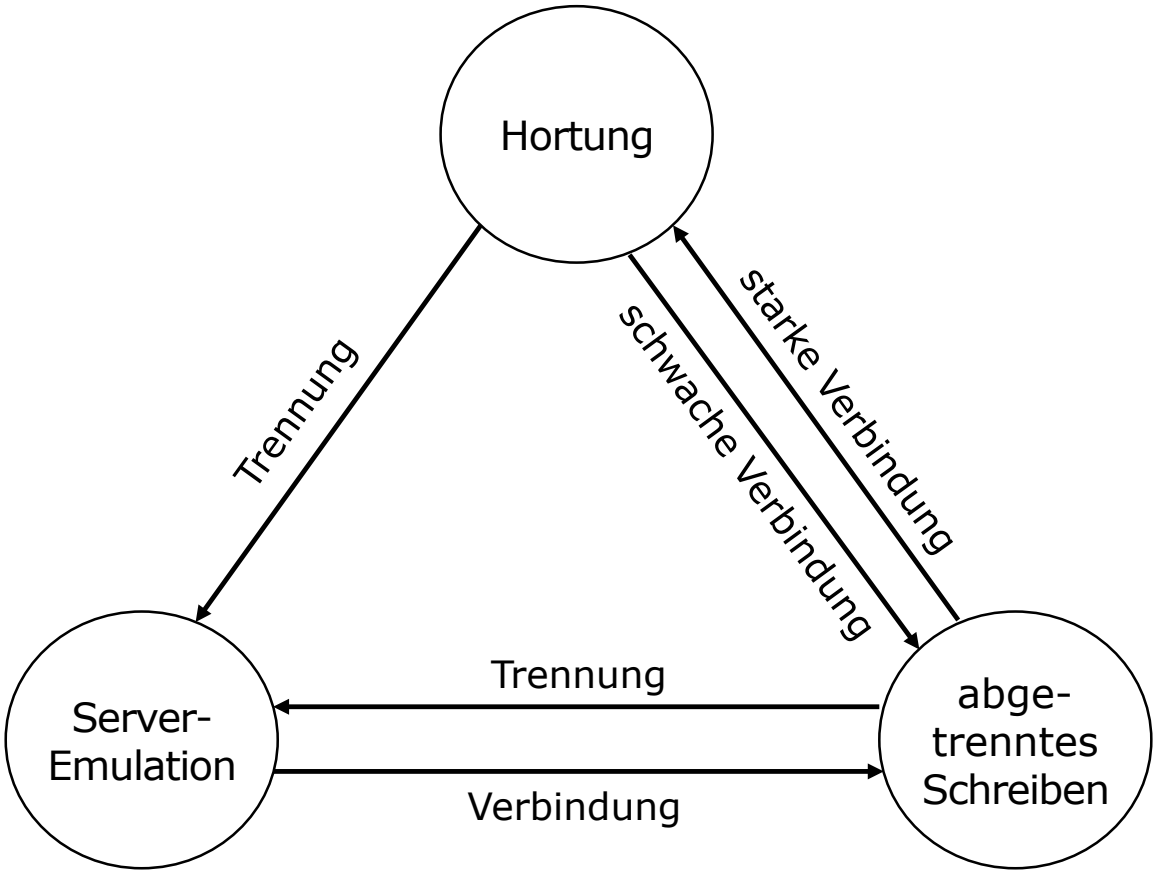
# PARTIELL VERBUNDENE OPERATIONEN

- Erweiterung des Andrew File System (AFS)
- schreibende Dateisystemzugriffe werden lokal durchgeführt und geloggt
- drei priorisierte Datenübertragungsklassen für effiziente Netzwerknutzung
  1. interaktive Kommunikation
  2. Lesen/Datenholung
  3. Schreiben/Datenplatzierung für Reintegration
- Klassen haben unterschiedliche Prioritäten
- eine Warteschlange pro Klasse
- spezifische Strategie zur Priorisierung interaktiver Kommunikation und hohe Datenverfügbarkeit im Cache

# Coda – schwach korrelierte Operationen

- Erweiterung abgetrennter Operationen in Coda
- zwei implizite Klassen der Datenübertragung
  1. Leseoperationen für fehlende oder veraltete Dateien hoher Priorität
  2. wenn keine Leseoperationen, dann Bruchteilreintegration im Hintergrund mit niedriger Priorität
- Berechnung eines Grenzwertes der maximalen Dateigröße beim Lesen neuer Dateien → Fehlermeldung bei größeren Dateien

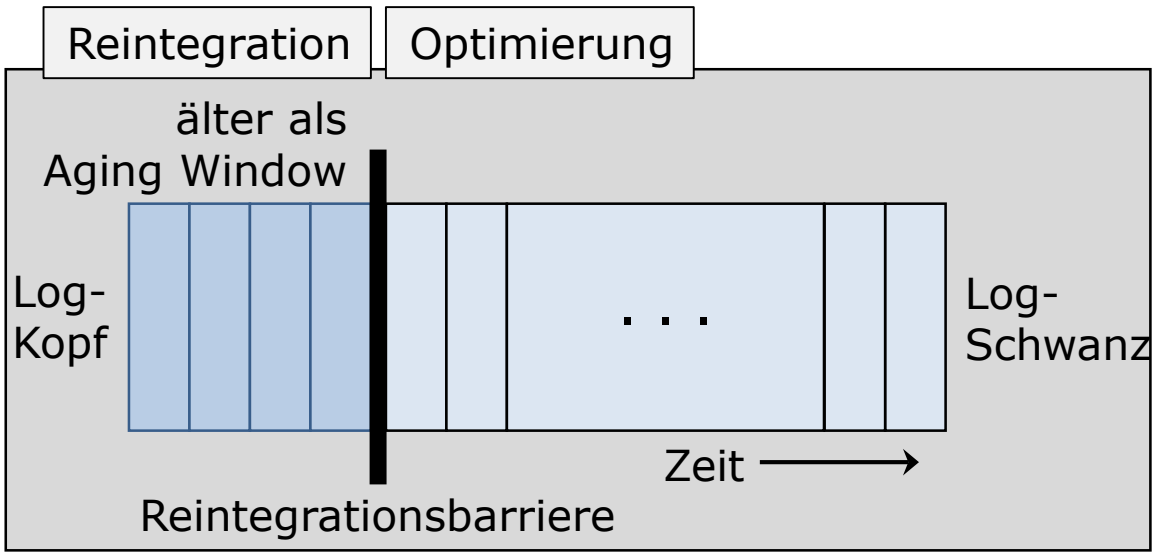
# CODA – SCHWACH KORRELIERTE OPERATIONEN




- löst strikte Trennung zwischen „verbunden“ und „abgetrennt“
- falls Netzwerkverbindung geringer Bandbreite verfügbar
  - Arbeit im getrennten Zustand beim Schreiben
  - Performanz der Reintegrationsfunktionalität bei Cache-Aktualisierungen

# CODA – REINTEGRATION VON BRUCHTEILEN

- kontinuierliche Reintegration im Hintergrund
- minimale Beeinflussung von Vordergrundaktivitäten
- Balance zwischen Effektivität der Log-Optimierung und Konfliktwahrscheinlichkeit
- minimale Verzögerung (an aktuelle Verbindung angepasst)
- Wiederholungsoperation verbleibt nur begrenzte Zeit im Änderungs-Log (Aging Window) (Best Practice: ca. 10min)



# Offline-First-Prinzip

- Idee: progressive Verbesserung
  - ausgehend von Annahme grundlegender Fähigkeiten
  - dann Nutzung von Fähigkeiten sobald diese verfügbar werden
- Offline-First
  - Grundannahme: keine Verbindung
  - progressives Verbessern sobald Verbindung verfügbar wird
  - Umdenken: mangelnde Verbindung ist kein Fehler; App so bauen, dass sie ohne Verbindung funktioniert

# Zusammenfassung und Aufgaben

# Zusammenfassung

- Ziel: für Programmierer/Anwender transparente Verbindung(sverluste)
- Kombination der Mechanismen
  - Caching/Prefetching
  - Emulation des Servers auf dem Client
  - Delayed Write-back
  - ausgewogener Netzwerkzugriff
- abgetrennte oder schwach korrelierte Operationen als Blaupause für anwendungsspezifische Lösungen
- Anwendung des Offline-First-Prinzips

# AUFGABEN

- Diskutieren Sie mit Ihren Kommilitonen Beispiele, wie die von Ihnen regelmäßig benutzten Apps Hortung, Emulation und Reintegration verwenden.
- Erstellen Sie mit Ihren Kommilitonen eine Liste von Applikationen nach dem Offline-First-Prinzip. Was sind erkennbare Gemeinsamkeiten und Unterschiede zwischen diesen Applikationen?
- Diskutieren Sie mit Ihren Kommilitonen, ob eine serverseitige Client-Emulation ebenfalls sinnvoll sein könnte.
- Überlegen Sie mit Ihren Kommilitonen, wie verhindert wird, dass unterschiedliche Clients in Phasen der Abtrennung die gleichen Versionsnummern für unterschiedliche Änderungen vergeben.
- Diskutieren Sie mit Ihren Kommilitonen, wann und wie Sie abgetrennte Operationen und/oder schwach korrelierte Verbindungen umsetzen würden. Geben Sie geeignete Beispielanwendungen!

# Referenzen

Mutschler, Bela; Specht, Günther: Mobile Datenbanksysteme

Höpfner, Hagen; Türker, Can; König-Ries, Birgitta: Mobile Datenbanken und Informationssysteme

Fanghänel, Thomas; Karlsson, Jonas S.; Leung, Cliff: DB2 Everyplace Database Release 8.1: Architecture and Key Features

The Ultra-Compact Database for Enterprise Mobility; Product Datasheet, http://www.pointbase.com/resourcecenter/pdfs/micro.pdf

https://www.slideshare.net/ChristianMelchior/introduction-to-realm-mobile-platform

UltraLite Database Management and Reference Sybase iAnywhere

https://techbeacon.com/offline-first-web-mobile-apps-top-frameworks-components