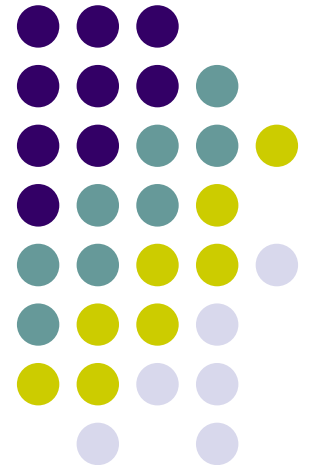


# CouPé: Ein Query Processor für UniStore

---

Martin Richtarsky  
TU Ilmenau





# Gliederung

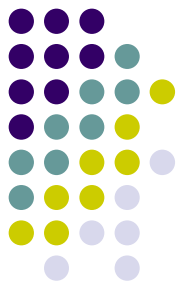
- UniStore
- Grundlagen
  - P2P-Netzwerke
  - Speichern strukturierter Daten, Algebra
- Query Processor
  - Architektur
  - Execution Engine
  - Operatoren
- Evaluierung

# UniStore



- Datenbanken werden immer umfangreicher
  - oft fehlt zentrale Instanz, die DB verwaltet (Nonprofits, Privatnutzer, Verzeichnisdienste...)
  - Kostengründe, organisatorische Gründe
- Lösung: P2P
  - keine zentrale Instanz, gleichberechtigte Hosts
  - “power in numbers“
  - Hat sich bewährt für Distribution von Dateien, Grid Computing, ...
- Anwendung auf Datenbanken
  - Ziel: massiv skalierbare Datenbank

# Grundlagen



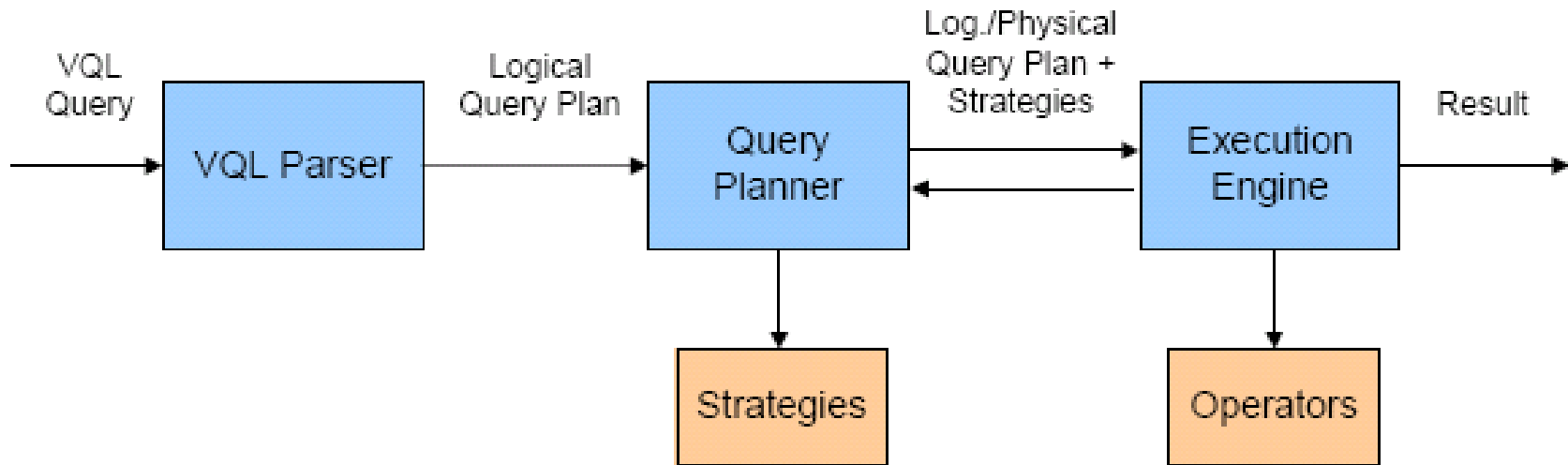
- Benötigt: P2P-Netzwerk
  - hier: Distributed Hash Table (DHT)
- Vertreter: CAN, Chord, P-Grid
- Herausforderung: Ablegen von strukturierten Daten (z.B. relationalen Daten) in einer Hashtabelle, so dass Datenbankoperationen darauf effizient ausgeführt werden können
  - Partitionierung - jede Zelle der Tabelle wird separat in der Hashtabelle abgelegt
    - Ansatz vergleichbar mit RDF
  - mehrfaches Speichern möglich – versch. Indexe

# Algebra

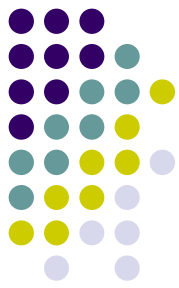


- Algebra
  - vergleichbar mit relationaler Algebra
  - + Operationen zur Rekonstruktion der ursprünglichen Tupelstrukturen aus den Daten der DHT
- VQL: Anfragesprache
  - `SELECT p WHERE { <x; 'price'; p> } ORDER BY p DESC LIMIT 5;`

# Query Processor: Architektur

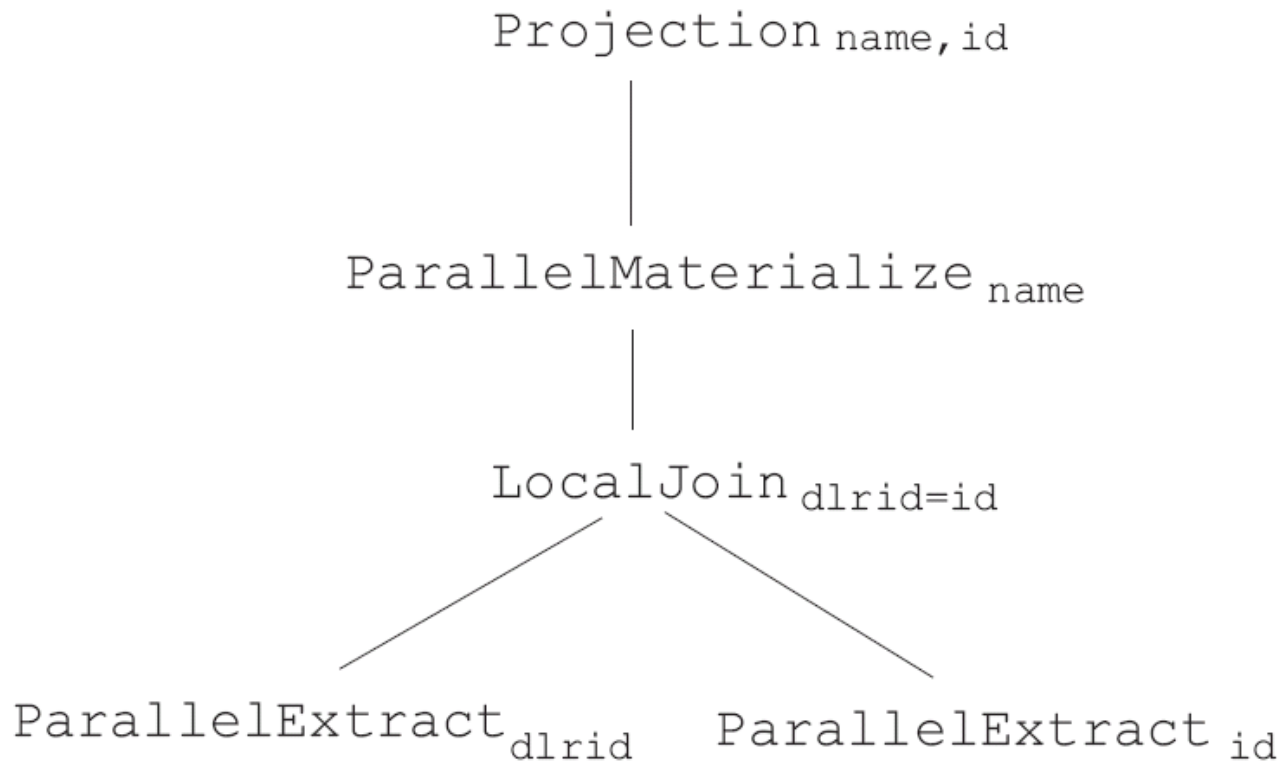


- vorhanden auf jedem Peer
- adaptives Bearbeiten der Anfrage



# Execution Engine

- Ausgangspunkt: physischer Anfrageplan





# Execution Engine

- Plan „wandert“ durch das Netz und nimmt Daten auf (Mutant Query Plan [PM02])
  - jeder Operator speichert die von ihm erzeugten Daten
  - Postorder-Reihenfolge
  - Wurzeloperator beendet? → Plan enthält Ergebnisse, zurück an Initiator
- Vorteil: kompletter Zustand der Anfrage im Plan gespeichert



# Execution Engine

- **Nachteil: relativ langsam, da seriell**
- **Parallelisierung**
  - in Execution Engine und Operatoren
- **Intra-Operator-Parallelität**
  - P-Grid-Feature: Range Queries (vgl. bar multicast)
  - Plan Cloning
- **Inter-Operator-Parallelität**
  - gleichzeitiges Ausführen von Join-Zweigen
- **Nachteil: Synchronisierung notwendig**



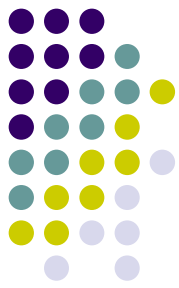
# Operatoren

- Lokale
  - Arbeiten auf Daten der Kindoperatoren im Plan
- DHT-Operatoren
  - Routing zu anderen Peers, Integration benötigter Daten in Plan
    - seriell, parallel
  - logischer Operator *Materialisierung* – 4 Implementierungen (versch. Indexe)
    - `OIDMaterialize`, `OptOIDMaterialize` (seriell)
    - `ParallelAVMaterialize` (parallel, Range Queries)
    - `ParallelOIDMaterialize` (parallel, Plan Cloning)

# Operatoren



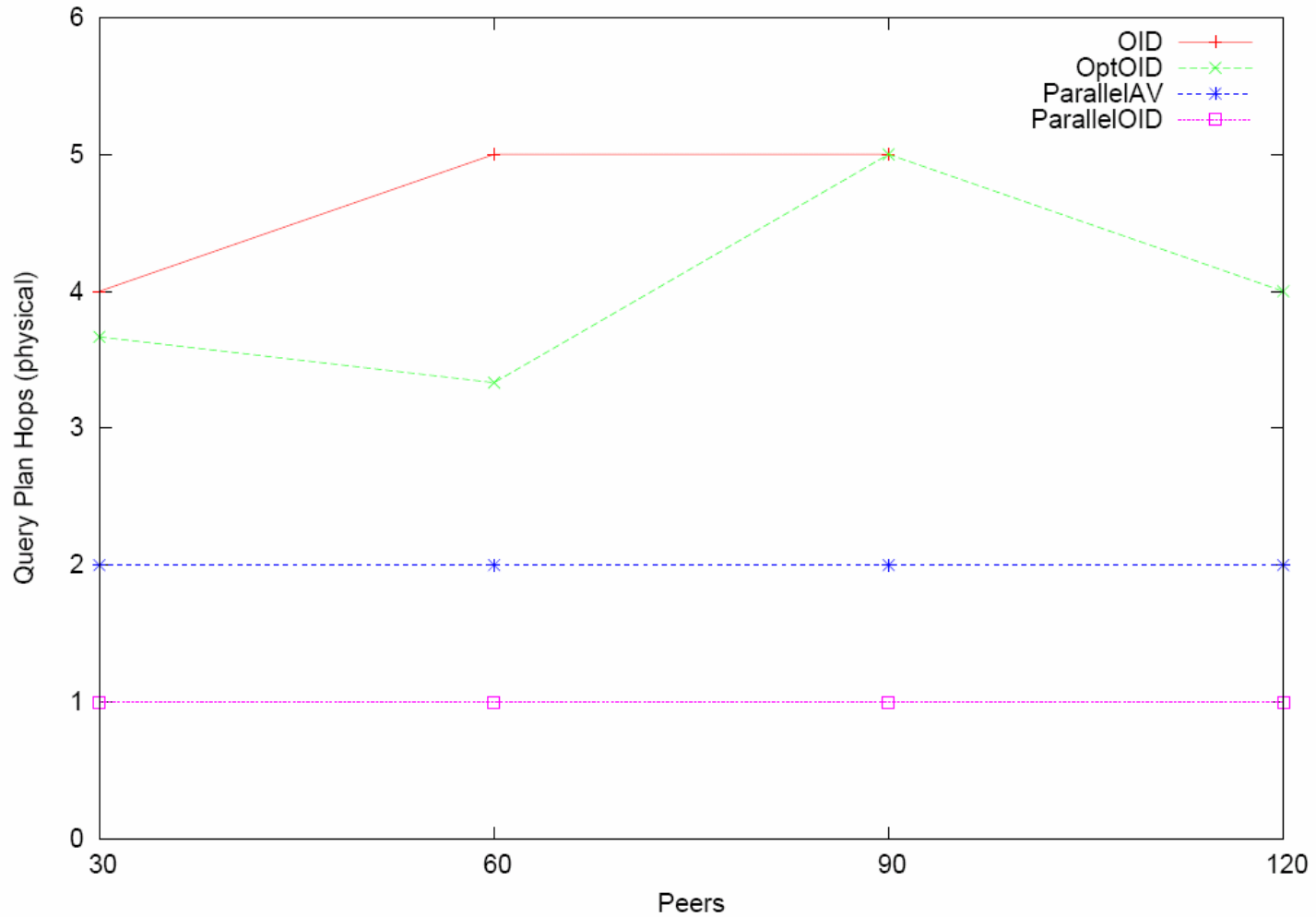
- Ähnlichkeitsoperationen (String, Integer)
  - spezielle Indexe für Ähnlichkeitssuche auf Strings  
→ effiziente Bearbeitung von  
Ähnlichkeitsselektionen, Ähnlichkeitsjoins
- Operatoren arbeiten auch auf Schema-Ebene
  - Anfrage: „alle Daten mit Spaltenname ‘name‘ und max. 1 Zeichen Unterschied“ →  
Daten der Spalten ‘name‘, ‘namen‘, ‘names‘



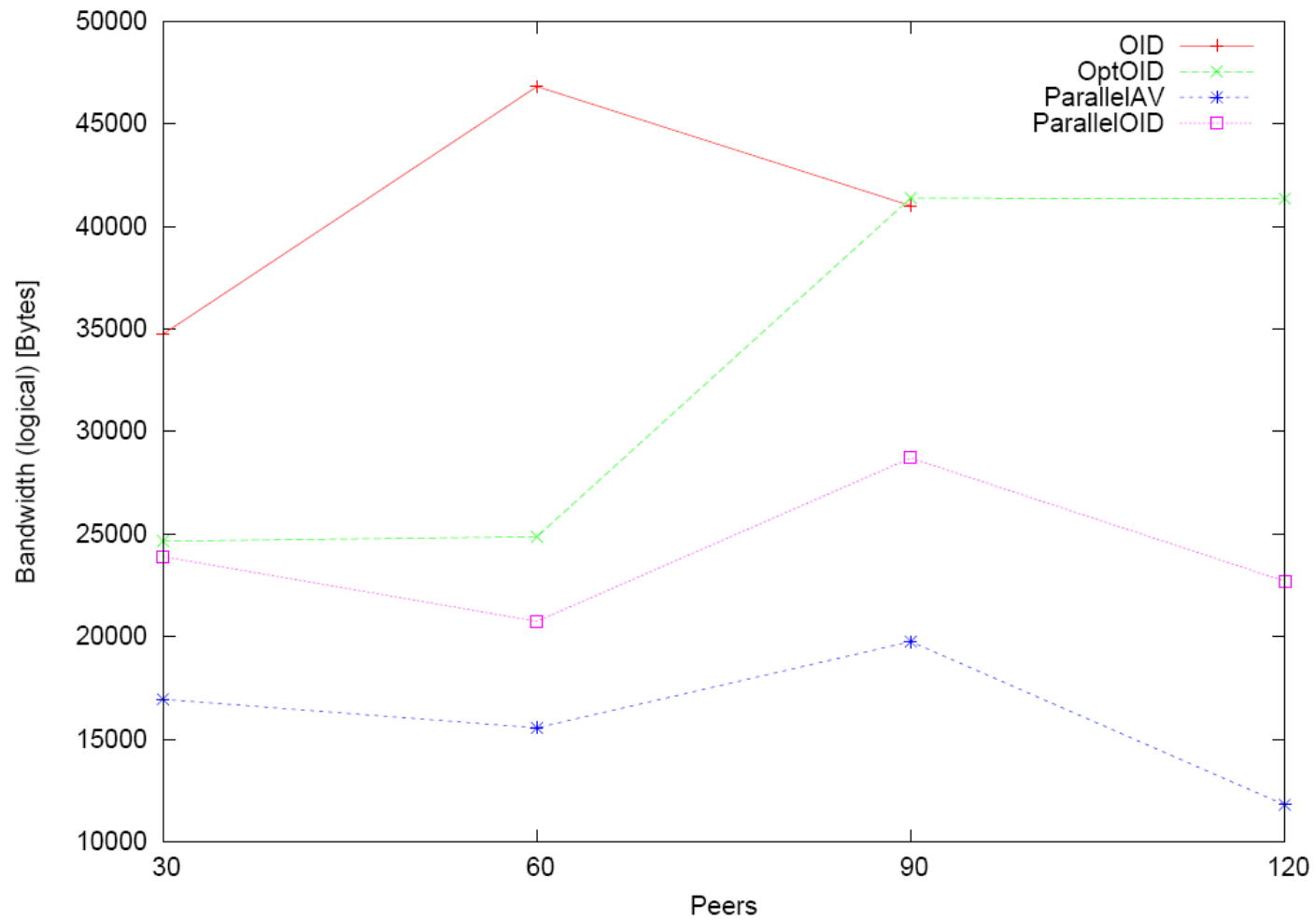
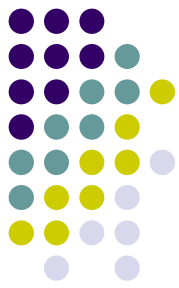
# Evaluierung

- Fragen:
  - Skaliert der Query Processor?
  - Welche Operatoren sind in welchen Situationen besonders geeignet?
- lokale Tests: Bugs ausmerzen ;)
- Planetlab
  - weltweites Netz von Rechnern für Experimente
- 4 Netze: 30, 60, 90, 120 Peers
- ~100 verschiedene Anfragen

# Evaluierung



# Evaluierung



# Evaluierung



- Aussagen über Skalierung sind mit nur 120 Peers nur begrenzt möglich
- Differenzierung zwischen Operatoren wird schon deutlich

**Danke für die Aufmerksamkeit!**



Fragen?