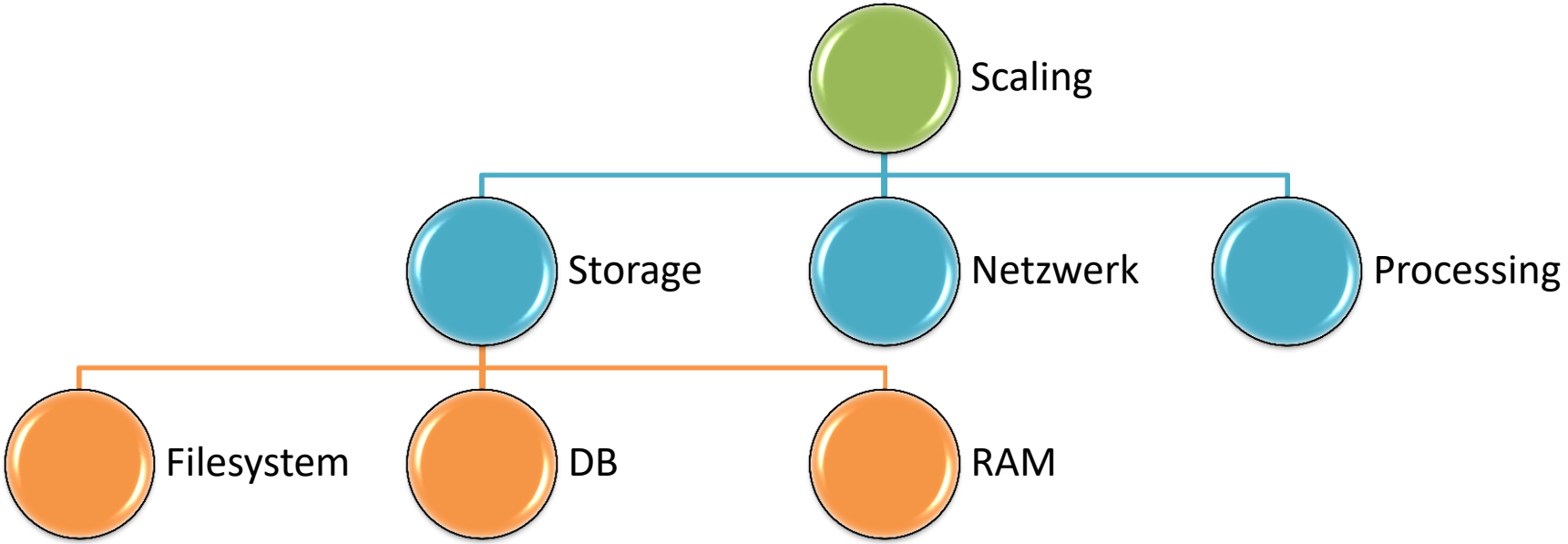
Wegweiser

Scaling:

wie kriegen wir die Cloud groß genug?

Scaling Dimensionen



Wegweiser

DB Scaling

Umgang mit Relationalen DBMSs

- Siehe Einführung
  - Scale-Up
  - Scale-Out
- Normalisierung + De-Normalisierung!
- Sharding
- Dimensionen:
  - Storage
  - Zugriffe/Performance
  - technische DB-Optimierung (z.B. Indizes, Caching)
- …

Zoo an Alternativen aka NoSQL

- In Memory DBs: SAP HANA; memcached, eXtremeDB
- Caches: memcached, redis
- Key-Value Stores: redis, Amazon Dynamo, Apache Cassandra
- Dokumenten DBs: CouchDB, MongoDB
- Graph DBs: InfoGrid, Neo4j
- Object DBs: ZopeDB, Gemstone
- …

NoSQL: Häufige Eigenschaften

- Nomen est Omen – keine SQL basierte Abfrage => aber bewegt sich in die Richtung
- Verzicht auf striktes ACID
- Eventual Consistent
- Einfache Skalierbarkeit durch Daten Verteilung / Sharding => siehe Consistent Hashing.
- Spezialisierter Use-Case
- Keine oder weniger „strikte“ Schemata

Beispiel:

Apache Cassandra

- Open Source, ursprünglich entwickelt von Facebook
- Konzepte übernommen von
  - Amazons Dynamo DB (der Core Entwickler von Cassandra hat vorher an Dynamo mitentwickelt)
  - Google BigTable
- Kommerzieller Support von DataStax
  - (inkl. AWS Deployment, falls gewünscht)

Cassandra Konzepte (1)

- Peer-to-Peer Modell - kein Master => Gossip Protokoll
- Automatische Replikation / Verteilung => Consistent Hashing
- Multi-Datacenter Support
- Tuneable Consistency Modell (per Operation) – read repair Konflikt Auflösung
- Hohe Performance => r/w skaliert weitestgehend linear mit Anzahl der Knoten
- Hohe Verfügbarkeit => definierbarer Replikationsfaktor, Hinted Handoff

Cassandra Konzepte (2)

- Key-Value++ / Column based
- Cassandra Query Language (CQL)
- Lightweight Transactions => PAXOS
- Map/Reduce support => Integration in Hadoop

Cassandra Tuneable Consistency

- Replikationsfaktor wird festgelegt
- Quorum = GanzZahligAbgerundet(Replikationsfaktor / 2 + 1)
- Verschiedene Konsistenzlevel (pro Operation wählbar)
- Konflikt Auflösung via Timestamp – neuster gewinnt
- Lesend (Auswahl):
  - ONE: Antwort des nächsten Knotens
  - Quorum: Antwort mit aktuellstem Zeitstempel aus Quorum Knoten
  - Local Quorum: wie Quorum aber nur aus einem Rechenzentrum
  - ALL: Antwort mit aktuellstem Zeitstempel nach Abfrage aller Replika-Knoten

Cassandra Tuneable Consistency

- Schreibend (Auswahl):
  - Any: Der Schreibzugriff muss persistiert sein (eventuell Lesen nicht möglich wenn die zuständigen Knoten nicht verfügbar => Hinted Handoff)
  - ONE: Der Schreibzugriff muss einem zuständigen Knoten persistiert sein (ist danach lesbar)
  - Quorum: der Schreibzugriff muss bei Quorum Knoten erfolgreich sein
  - Local Quorum: wie Quorum nur 1 RZ
  - All: der Schreibzugriff muss bei allen Replika-Knoten erfolgreich sein

Publikumsfrage – Beispiele für verschiedene Lese-/ Schreibszenarien

Cassandra Datenmodell

Keyspace *(=>Datenbank)*

Bestimmt den zuständigen Knoten => Consistent Hashing

Column Family *(=>Tabelle)*

Row

Column

Row Key: Name:Value Name:Value Name:Value

Row Key: ▪ ▪ ▪

▪ ▪ ▪

Der Inhalt der Row liegt „linear“ auf der Platte

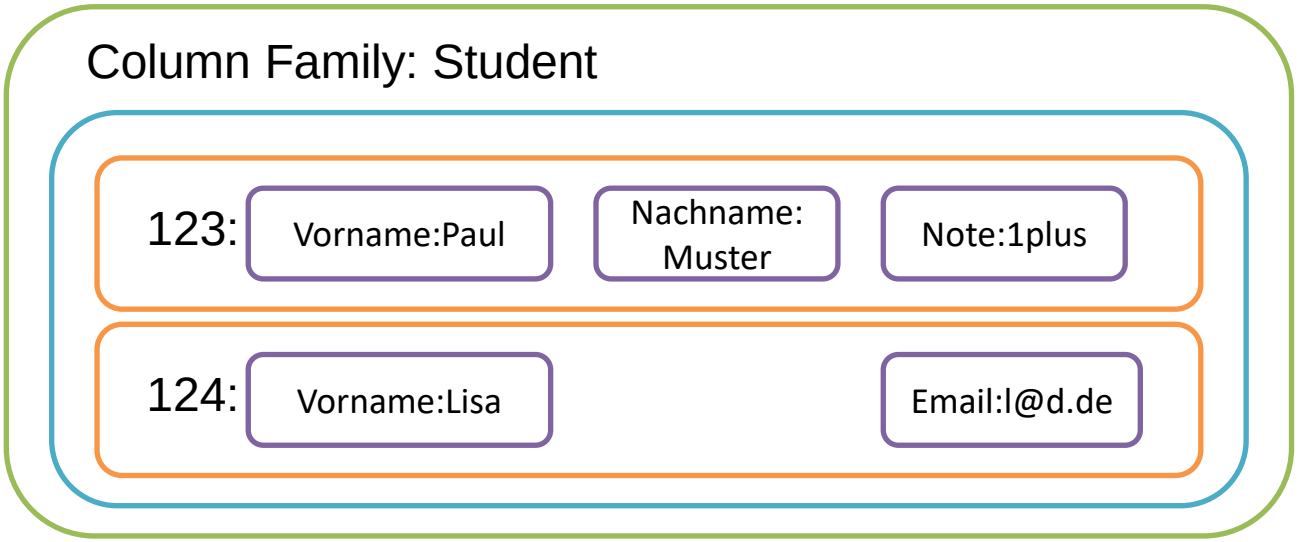
Row

▪ ▪ ▪

▪ ▪ ▪

Cassandra Datenmodell Beispiele

Keyspace: StudiDB




```
create keyspace StudiDB;
use StudiDB;
create column family Student;
set Student [‚123‘][‚Vorname‘] = ‚paul‘
set Student [‚123‘][‚Nachname‘] = ‚Muster‘
set Student [‚123‘][‚Note‘] = ‚1plus‘
set Student [‚124‘][‚Vorname‘] = ‚Lisa‘
…
```

```
…
get Student[‚123‘]
=>(colum-Vorname, value=Paul, timestamp=34324)
=>(column-Nachname=Muster, timestamp=132423)
=>(column…

get Student where note=‚1plus‘
```

Secondary Index muss gesetzt sein

Cassandra Randbedingungen

- Colums können hinten oder vorne hinzugefügt werden => implizite zeitliche Sortierreihenfolge möglich
- Löschen von Colums erzeugen „Tombstones“
  - => z.B: Queues sind ein Anti-Pattern
- Schreibzugriffe schneller als Lesen (disk commit log => Memtable => DataFile (Sorted Strings Table) & SSTableIndex & Bloomfilter)
- Vorsicht bei secondary Indexen auf Columns
  - Die Rows sind verteilt gespeichert
  - Niedrige Kardinalität (viele Duplikate) von Vorteil
  - Bei Hoher Kardinalität (und sehr kleine ausgeprägte Selektivität) zu hoher overhead => unnötige Anfragen an alle Knoten im Cluster

Level:ANY

Level:ONE