



# NoSQL

Datenbanken für Big und Fast Data



Cegos Group

inspire  
qualify  
change



# Inhalts- verzeichnis



Die Problemstellung



NoSQL



Produkte



# Die Problemstellung



- Big & Fast Data
- Das CAP-Theorem
- Analyse von Big Data
- Einige Klarstellungen



# Big & Fast Data



## Was ist „Big“?

- Definitionsversuche:
  - Physikalisch
    - Terabyte
  - System
    - Übersteigt den Bedarf eines normalen Speichermediums
  - Technisch
    - Datenverarbeitung beginnt, auf Grund der Größe Probleme zu bereiten
- Allerdings:
  - Keine wirklich eindeutige Definition erkennbar
  - Grenzen können sich verschieben



## Wo treten Big Datas auf?

- Völlig verschiedene Bereiche:
  - DNS-Server für Internet
  - Indizierung von Web Seiten für Suchmaschinen
    - Google
  - Monitoring von Seitenzugriffen, aber auch Metrik-Informationen in großen Server-Systemen
  - Content Management und Waren-Systeme
    - Amazon
  - Social Media
    - Facebook



## Cluster-Strategien

- Zur Ablage und Auswertung von Daten sind selbst mächtige Server-Maschinen auf Dauer überfordert
- Die Datenhaltung wird deshalb auf mehrere Server verteilt
- Die Lastverteilung übernimmt ein simpler Load Balancer oder ein komplexer "Master"
  - Aus Client-Sicht heraus tritt der Cluster damit immer noch als eine Einheit auf
- Im Optimum ist der Cluster damit eine quasi unendliche Datensenke mit beliebiger skalierbarer Rechnerkapazität
  - Damit eine Vorstufe zur "Cloud"



# Vertikale Skalierbarkeit

- Grenzen sind klar definiert
  - Kosten
  - Aktuelle Hardware-Grenzen
  - Anforderungen an Ausfallsicherheit





# Horizontale Skalierbarkeit

- Grenzen
  - Ausfallsicherheit fordert Replikation über Netzwerk
  - System-Administration und Überwachung

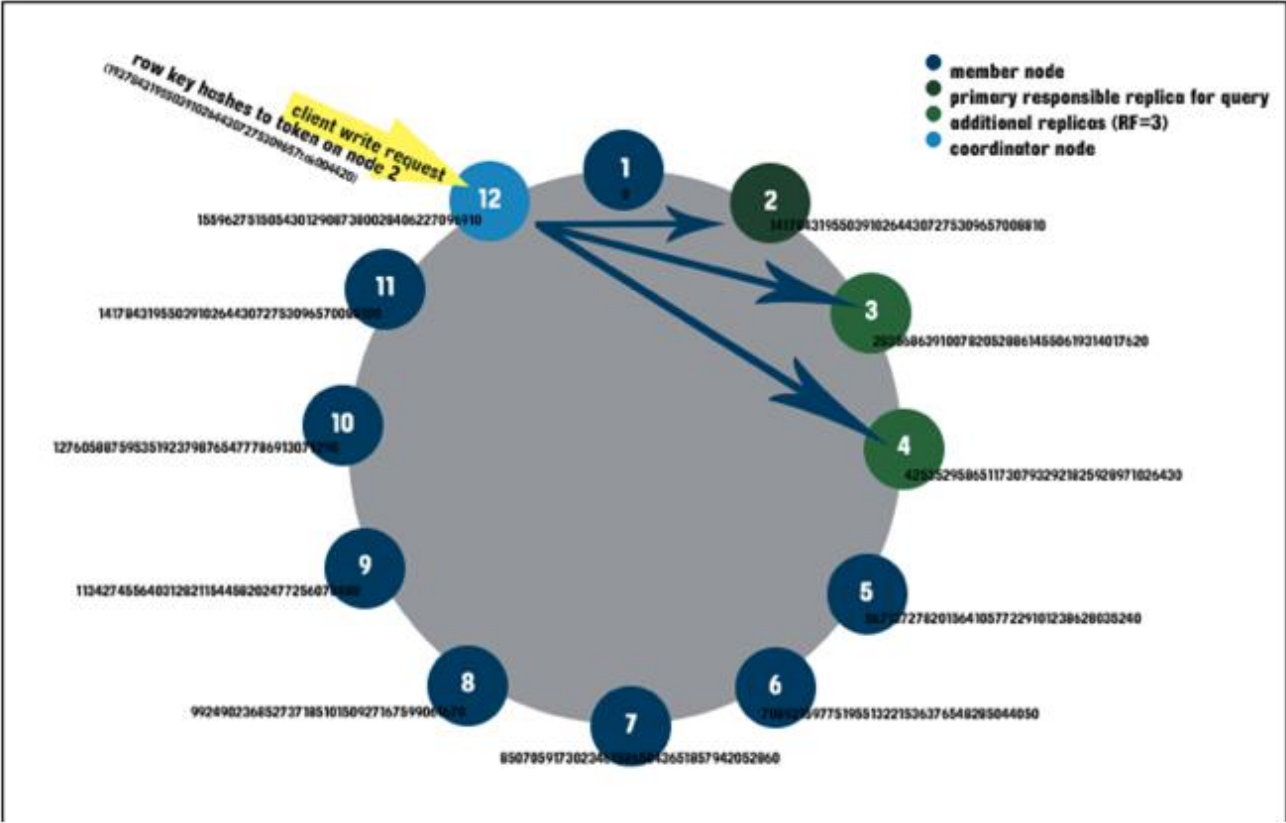


## Der horizontal skalierende Ring- Cluster

- Die Daten werden auf Grund eindeutiger Hashes auf einen "Ring" von einzelnen Servern verteilt
- Es gibt hierbei keinen Master
  - Jeder Knoten kann auf den Server delegieren, der die Daten wirklich hält



# Beispiel: Apache Cassandra Ring

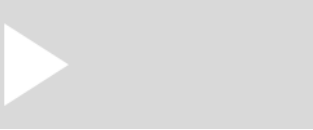


Quelle: <http://www.planetcassandra.org/blog/node-of-the-rings-fellowship-of-the-clusters-or-understanding-how-cassandra-stores-data/>

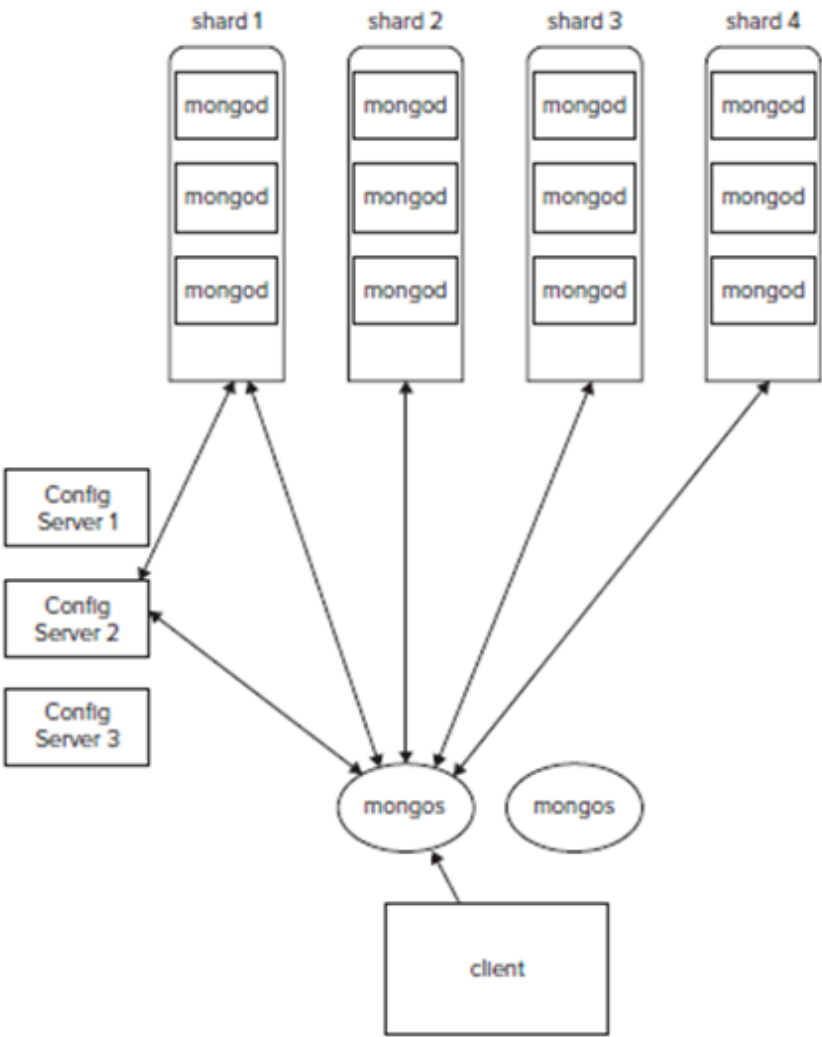


# Sharding

- Daten werden in sinnvoller, wohl-definierter Art und Weise auf mehrere Rechner verteilt
- Eine besondere Herausforderung stellt sich im Cluster-Betrieb
  - Das Hinzufügen eines Knoten kann ein „Resharding“ auslösen
  - Dies kann zu beträchtlicher Last auf dem Server führen
    - Administrativer Vorgang!

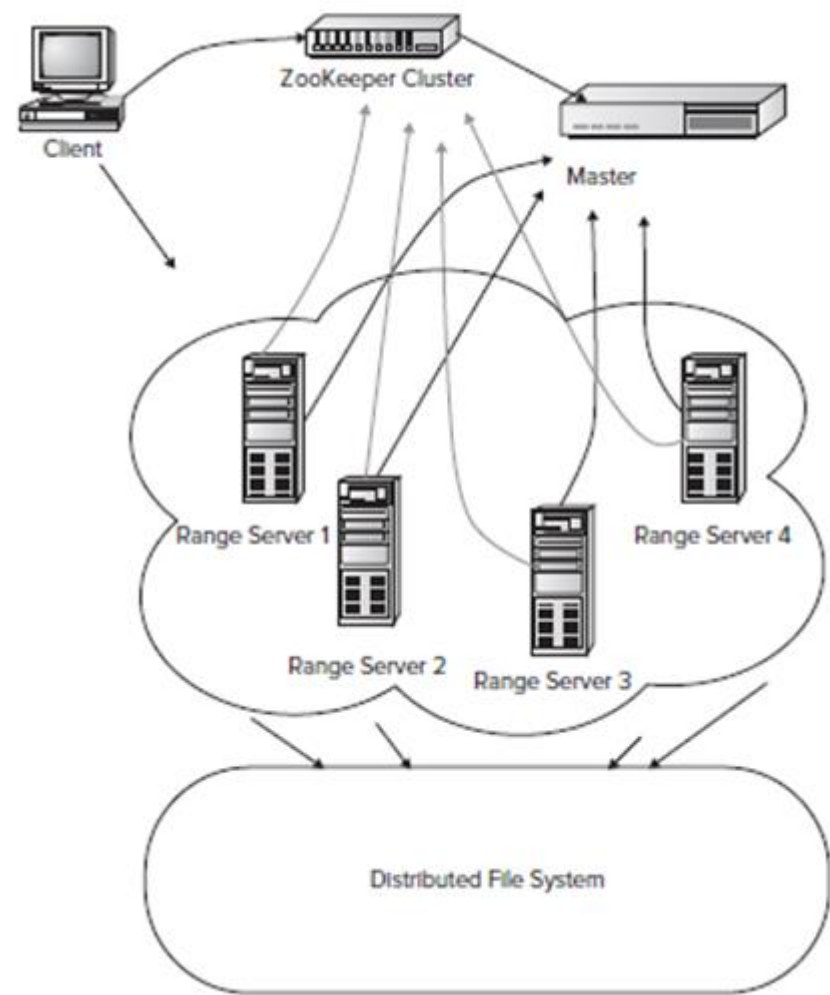


# Beispiel: Sharding und die Mongo DB





# Beispiel: Apache Hadoop





# Das CAP-Theorem



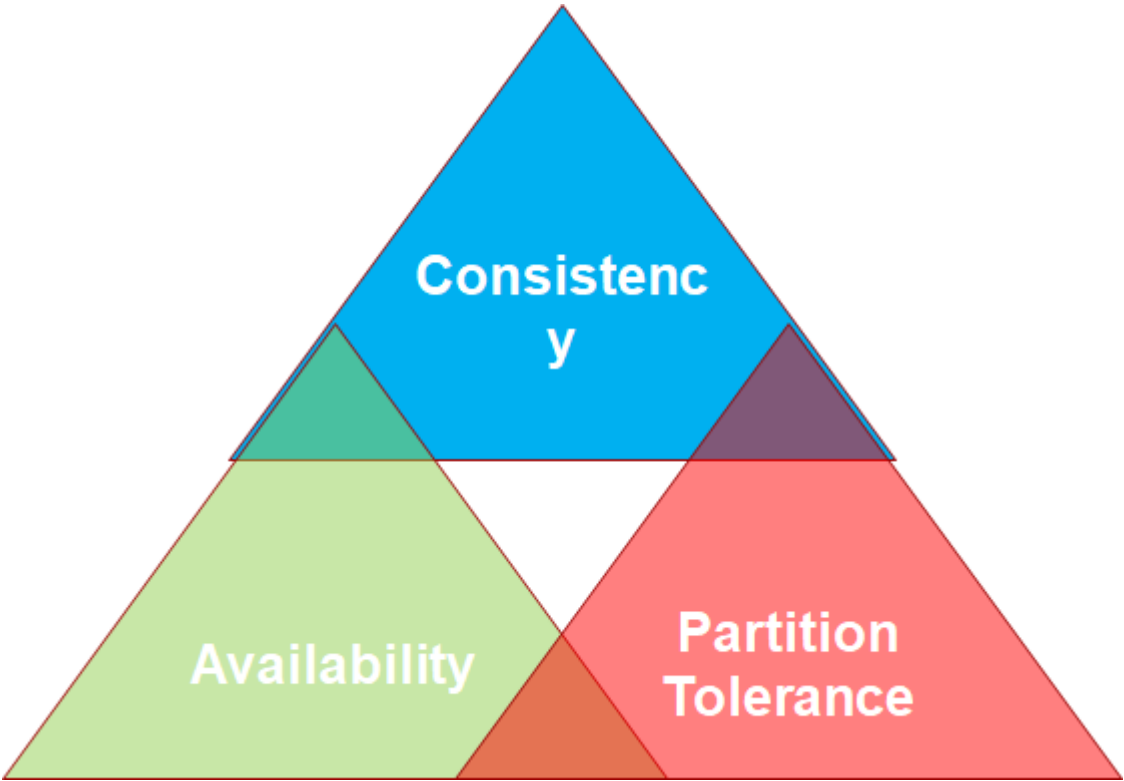
# Begriffe

- Consistency
  - Alle Knoten haben jederzeit den selben Informationsstand
- Availability
  - Jeder Client bekommt garantiert Antwort
    - Solange zumindest ein Knoten im Cluster läuft
- Partition Tolerance
  - Das System toleriert
    - Den Ausfall von Knoten
    - Den Ausfall des Netzwerks zwischen Knoten





Keine gemeinsame  
Schnittmenge!





## BASE: Ein Schlupfloch

- Consistency
  - Alle Knoten haben **jederzeit** den selben Informationsstand
- Eventually Consistent
  - Änderungen des Datenbestandes werden zeitlich versetzt an die anderen Knoten propagiert
- BASE
  - **B**asically **A**vailable
  - **S**oft state
  - **E**ventual consistency



# ACID und BASE

- ACID
  - Starke Konsistenz der Daten
  - Transaktionssteuerung
    - Transaction-Isolation
    - Bei Bedarf Zwei-Phasen-Commit
  - Relativ komplexe Entwicklung
- BASE
  - Schwache Konsistenz
  - Hohe Verfügbarkeit
  - Schnelligkeit
    - Bei Bedarf "Fire-and-forget"
  - Leichtere Entwicklung



# Analyse von Big Data

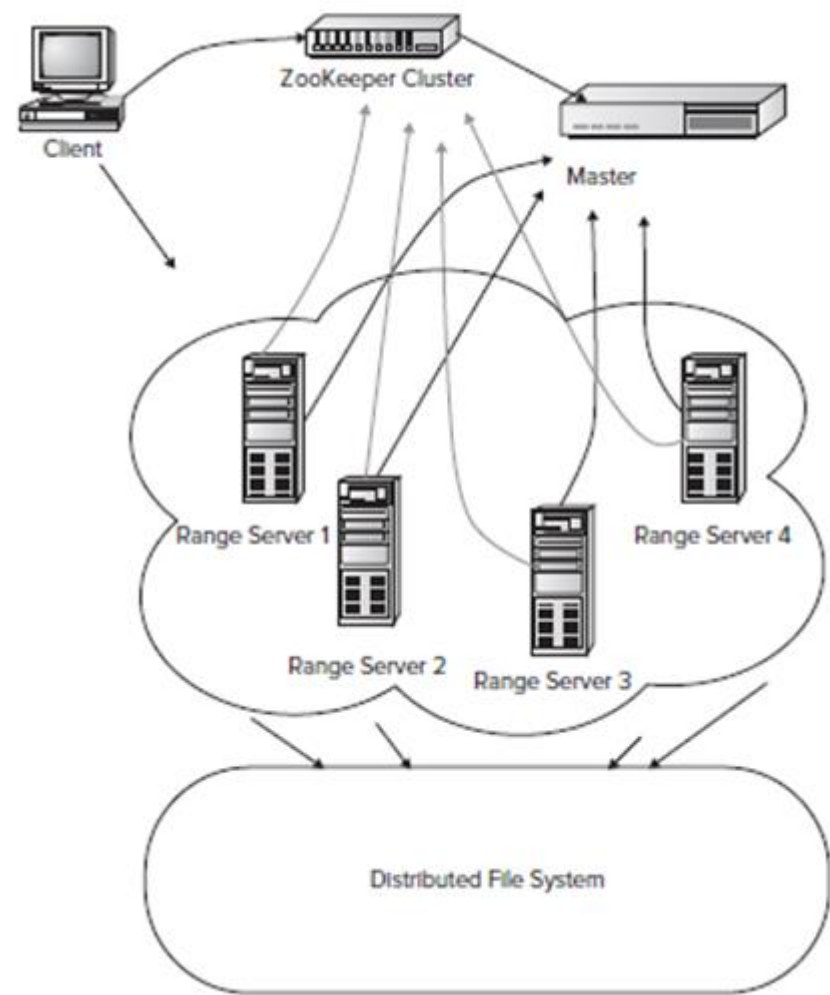


# Analyse von Big Datas

- Verlangt „neue“ Algorithmen
  - Aufteilung in Jobs
  - Diese werden parallelisiert
  - Operieren auf einem verteilten Datenbestand
- Alle genannten Unternehmen haben in den letzten 10 Jahren Algorithmen entwickelt bzw. eingesetzt
  - Und „netterweise“ der Open Source Community zur Verfügung gestellt
- Fast Data
  - Schneller Zugriff auf Daten-Selektionen
  - Schnelle Umsetzung neuer Abfrage-Anforderungen



# Beispiel: Apache Hadoop

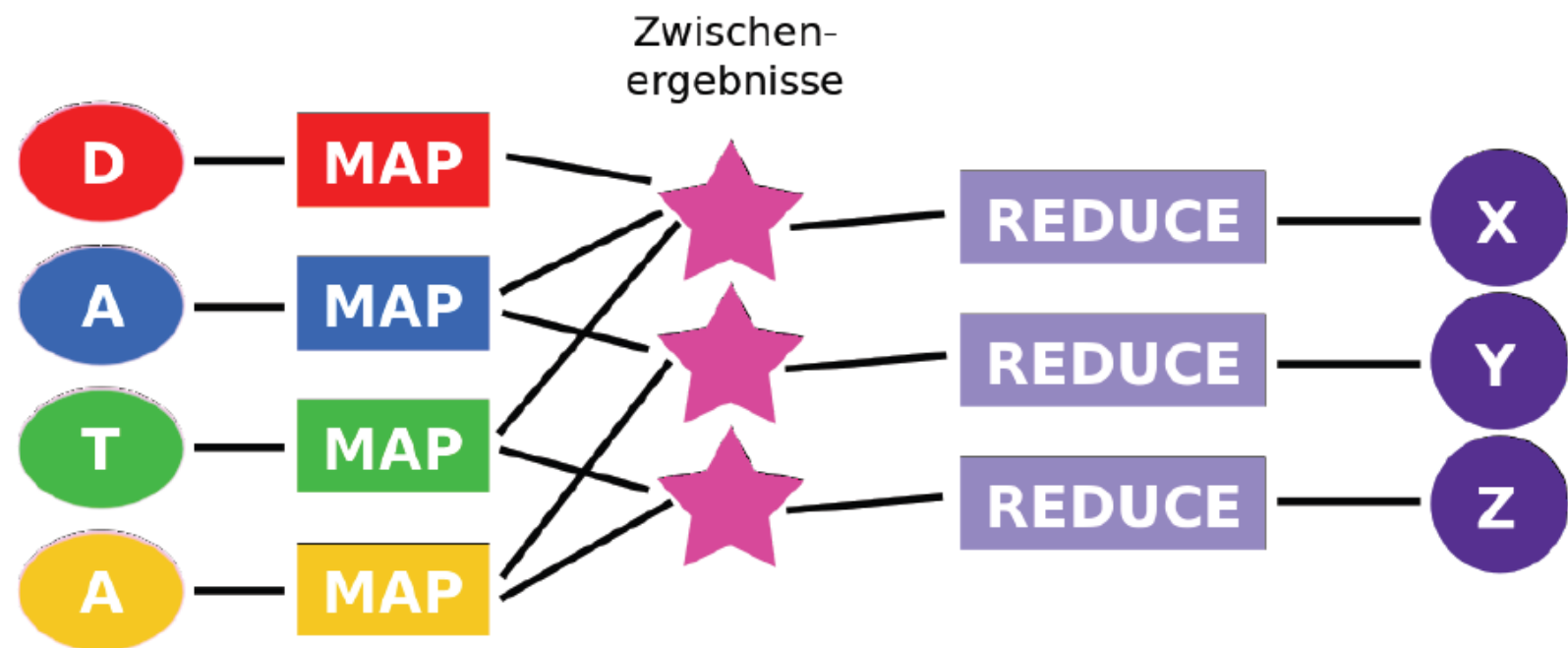




# MapReduce

- Bahnbrechende Entwicklung
  - Idee wird Google zugeschrieben
- Grundsätzliche Arbeitsweise
  - Operiert auf Mengen (Maps)
  - In einem ersten Schritt werden die Daten der Map in einer neuen Map aufbereitet
  - Diese wird in einem oder mehreren Reduce-Schritten zur Ergebnis-Struktur zusammengefasst

# Ablauf



Quelle: <http://de.wikipedia.org>





# Einige Klarstellungen



# NoSQL ist kein Komplettpaket!

- NoSQL-Produkte benötigen
  - Netzwerk-Infrastruktur
    - Netzwerk
    - Router/Load Balancer
  - Dateisystem
  - Überwachung
  - Administration
  - System-Design



# Hadoop ist nicht NoSQL!

- Hadoop wird von einigen NoSQL-Produkten benutzt
  - Oder kann benutzt werden
- Ist Bestandteil der Infrastruktur
  - Job-Manager
  - Distributed File System



# Relevanz von NoSQL außerhalb von Web Anwendungen

- Einige Teile der Historie von NoSQL entstammen aus Problemstellungen von Web Anwendungen
  - Session Management
  - DNS-Server
  - Content Repositories
- Viele andere Aufgabengebiete sind aber ohne Bezug hierzu
  - Content Management Systeme
  - Caches
  - Tagging/Indizierung von Big Data
  - ...



## NoSQL-Datenbanken verlangen den Einsatz von Skript- Sprachen

- Durch das fehlende statische Schema ist ein Mapping von NoSQL-Strukturen in eine statisch kompilierte Programmiersprache tatsächlich schwierig
  - Aber nicht unmöglich
- Skriptsprachen wie Ruby, Python oder auch JavaScript erzeugen deshalb meistens deutlich kompaktere Programme
  - Man muss diese Sprachen aber definitiv nicht benutzen
  - Mappings nach Java, C# etc. können benutzt werden
- Viele NoSQL-Datenbanken ermöglichen es, MapReduce-Algorithmen beispielsweise mit JavaScript zu formulieren
  - Dies wird jedoch stets auf Server-seite optimiert
  - Es existieren auch native Alternativen
- Das aus JavaScript stammende JSON-Format hat sich unabhängig als Fast-Standard für den Datentransfer entwickelt

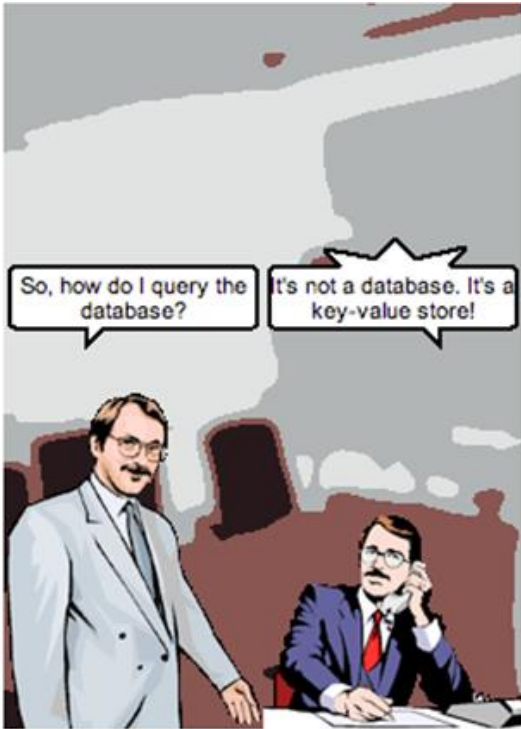


## NoSQL-Daten sind unstrukturiert

- Die Verarbeitung unstrukturierter Daten in einer NoSQL kann eine sehr große Stärke darstellen
  - „Ich muss nicht detailliert wissen, wie etwas strukturiert ist, wenn mich nur bestimmte Teile interessieren“
    - Genaues Format einer Log-Meldung
    - Auswertung von Server-Statistiken verschiedenster Produkte
    - Agile Software-Entwicklung
- Aber auch NoSQL kann Struktur vorgeben
  - Verwendung von XML-Schema
  - Mandatory Properties eines Dokuments



# NoSQL ist simpel





# NoSQL ist auf dem Weg zur Standardisierung

- MapReduce-Funktionen
  - Werden häufig als JavaScript definiert
    - Die Programmlogik einer MapReduce-Funktion ist sehr überschaubar!
- Datenformat
  - Hier hat sich JSON etabliert
- Client-Zugriff
  - RESTful Web Services





# NoSQL



Definition



Klassifizierung



Datenmodellierung



# Definition



# NoSQL

- No SQL!
  - Ursprüngliche Bedeutung
    - Etwas dogmatische Ablehnung relationaler Datenbank-Prinzipien
- Not only SQL
  - Schon deutlich abgeschwächt
  - Reduktion auf geeignete Problemstellungen
- No/ Not only relational
  - Der eigentlich richtige Begriff
  - Fokussierung auf die Daten-Modellierung, nicht auf die Abfragesprache



# Klassifizierung



# Klassifizierung von NoSQL-Produkten

- Key/Value
- Column-orientiert
- Dokumenten-orientiert
- Graphen-orientiert

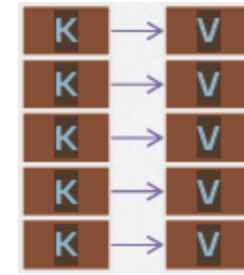


# Key/Value

- Ablage von Inhalten (=Value) unter einem eindeutigen Schlüssel (key)
  - Flache Struktur
  - Values werden als Byte-Array betrachtet
- Teilweise Verzicht auf Persistierung der Daten
  - Cache-Systeme
- Auslegung auf eine Vielzahl konkurrierender Zugriffe



# Key/Value: Arbeitsweise





## Column-orientiert

- Daten werden in Columns abgelegt
  - Eine Zeile besteht aber nicht zwangsläufig aus allen Columns
- Diese Columns können noch in Gruppen unterteilt werden
  - Vorteilhaft bei einer verteilten Datenhaltung
- Ausrichtung auf wirklich große Datenmengen



# Column-orientiert: Arbeitsweise

				1
1			1	1
		1		1
		1		1
		1		
				1
		1		

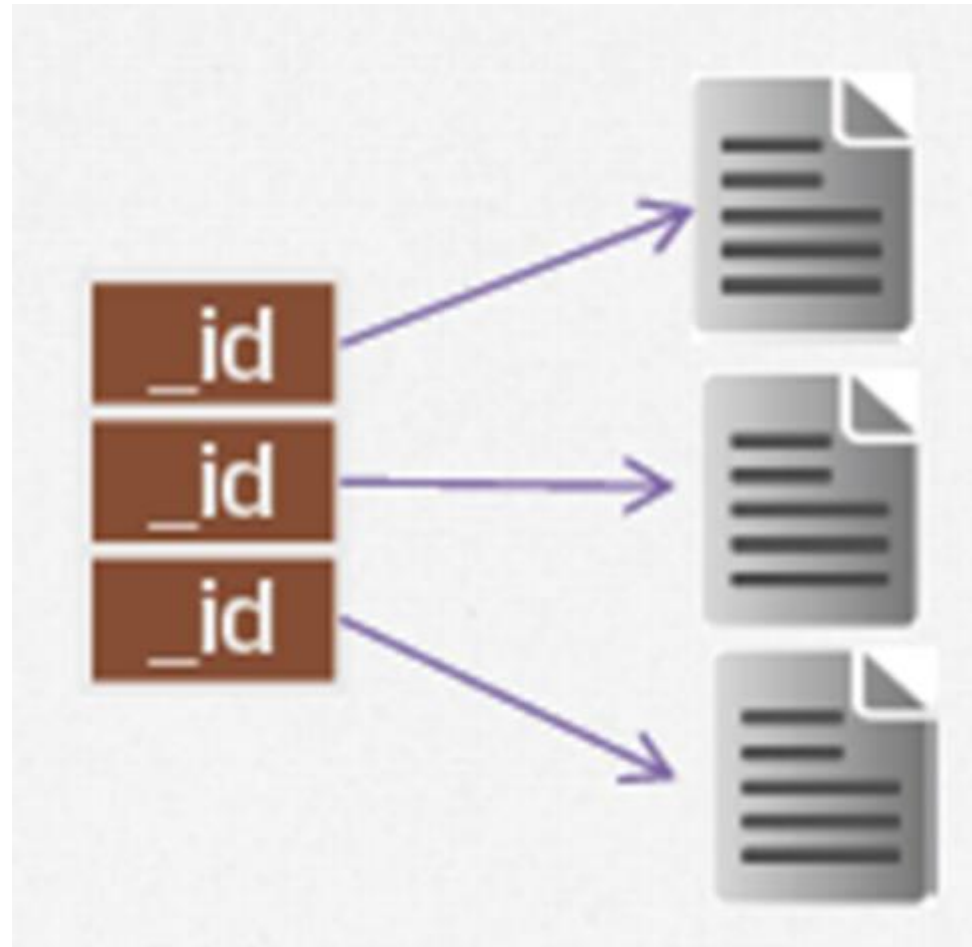


# Dokumenten-orientiert

- Ablage der Daten in strukturierten Dokumenten
  - Nicht unbedingt XML!
  - Wesentlich weiter verbreitet ist JSON
    - Bzw. das binäre BSON



## Dokumenten- orientiert: Arbeitsweise





# Graphen-orientiert

- Ablage der Daten in
  - Knoten
  - Beziehungen
  - Attribute
- Knoten und Beziehungen haben Attribute
- Knoten sind mit Beziehungen verknüpft und umgekehrt
- „Whiteboard-Friendly“
  - „Was Sie auf einem Whiteboard zeichnen können, können Sie in einer Graph-DB ablegen.“

Graphen-orientiert:  
Arbeitsweise





# Datenmodellierung



## Statische Typisierung

- Daten werden abstrahiert modelliert
  - Klassenorientierte Programmierung
    - Java, C++/C#, ...
  - Tabellen-Schemata
  - XML-Schema
- Eine Umwandlung von Datentypen erfordert ein Umkopieren
  - „Migration“
- Damit sind Entwicklungszyklen und die Zeiten für die Einführung geänderter Features relativ lang



## Dynamische Typisierung

- „Duck Typing“: Nur vorhandene Eigenschaften und implementiertes Verhalten zählen
  - „Was gelb ist, quakt und watschelt ist eine Ente“
- Dies ist vorwiegend die Domäne von dynamischen Skript-Sprachen
  - Ruby, Python, JavaScript
- Ein Umkopieren von Daten ist nur noch in Ausnahmefällen notwendig
- Damit wird eine agile Software-Entwicklung unterstützt





# Statisch versus Dynamisch

- Was ist...
  - Besser?
  - Richtig?
  - Komfortabel?
- Leider keine pauschale Antwort möglich
  - „Wer lebend den Raum verlässt hat gewonnen“
- Aber:
  - Der Trend der letzten Jahre geht immer mehr Richtung Dynamik



# Produkte



Übersicht





# Übersicht



# Produkte





## Einige ausgewählte Produkte

- Mongo DB
  - Eine Dokumenten-orientierte Datenbank
- Riak
  - EinKey/Value-Store
- Neo4J
  - Eine Graphenorientierte Datenbank
- HBase
  - Spalten-orientiert
- Die jeweiligen Dokumentationen stehen bei den Herstellern im Web zur Verfügung
  - Technische Dokumentation
  - Tutorials und Beispiele