Lehrveranstaltung
Erweiterte Datenbanktechnologien / Medienarchive
(Datenbanksysteme III)

Gliederung

5. „Big Data“-Datenbanktechnologien

5.1. Einführung
Was versteht man unter „Big Data“?

Wikipedia:

„Als Big Data werden **besonders große Datenmengen** bezeichnet, die mit Hilfe von Standard-Datenbanken und Daten-Management-Tools nicht oder nur unzureichend verarbeitet werden können.

Problematisch sind hierbei vor allem
- die Erfassung,
- die Speicherung,
- die Suche,
- Verteilung,
- Analyse und
- Visualisierung

von großen Datenmengen.

Das Volumen dieser Datenmengen geht in die Terabytes (10^{12} Byte), Petabytes (10^{15} Byte), Exabytes (10^{18} Byte) und Zettabytes (10^{21} Byte) .“

Anwendungsszenarien von Big Data

Beispiele für große Datenbanken

Wal Mart Data Warehouse

- Produktinfos (Verkäufe etc.) von 60.000 Märkten
- ca. 312.000 Produkte , 4,2 Bill. Kauftransaktionen/Jahr
- 2004: ca. 480 TB → 2014: > 8.192 TB = 8 PB

SAP ERP-Installation der Deutschen Telekom AG (2012)

- Financial Accounting: Rechnungen, Zahlungsaufforderungen, Lastschriften, Mahnungen etc.
- 15 SAP R/3-Systeme; jedes verarbeitet 200.000 Rechnungen, 12.000 Mahnungen, 18.000 Änderungen von Kundendaten pro Tag
- bis zu jeweils 10.000 Nutzer gleichzeitig, > 13.000 Datenbanktabellen

NASA Earth Observing System (EOS)

- 1,9 TeraByte Datenvolumen pro Tag (10 PetaByte Gesamtvolumen)
- Erfassungszeitraum: 15 Jahre
- nur 10% des Datenmaterials wird analysiert
- Echtzeit-Übernahme des Messdatenstroms (51 MegaBit/sec bzw. 553 GigaByte pro Tag)
- jährlich ca. 100.000 Benutzer der EOS-Datenbank
- mittlere Objektgröße als Resultat einer Anfrage: 10 MegaByte

Anforderungen an BigData-Technologien

Gartner-Group: Wenn Standard-Technologien die Datenflut nicht mehr beherrschen, sind Big-Data-Ansätze gefragt

→ „3Vs“ („Big Data Merkmale“):

Volume: Sehr ***große Datenmengen*** im Tera-, Peta- und Exabytebereich werden gespeichert und analysiert.

Variety: Mehr noch als die reine Größe stellt die ***Komplexität*** starke Anforderungen an die Analyse von Big Data.

Beisp.:
- Qualitätsschwankungen, die vom Wetter abhängen,
- Verkaufszahlen, die mit Nachrichtenmeldungen korrelieren,
- Betrugsmuster, die sich erst durch hochstrukturiertes, technisches Wissen erkennen lassen

→ erst die Kombination der einzelnen Puzzlestücke liefert das ganze Bild!

Velocity: oft ist ***Geschwindigkeit*** alles !

z.B. bei Datenströmen, der Reaktion auf Kreditkarten-Betrug, beim Ausliefern von Online-Werbung oder bei der Optimierung von Produktionsprozessen in Near-Realtime

+

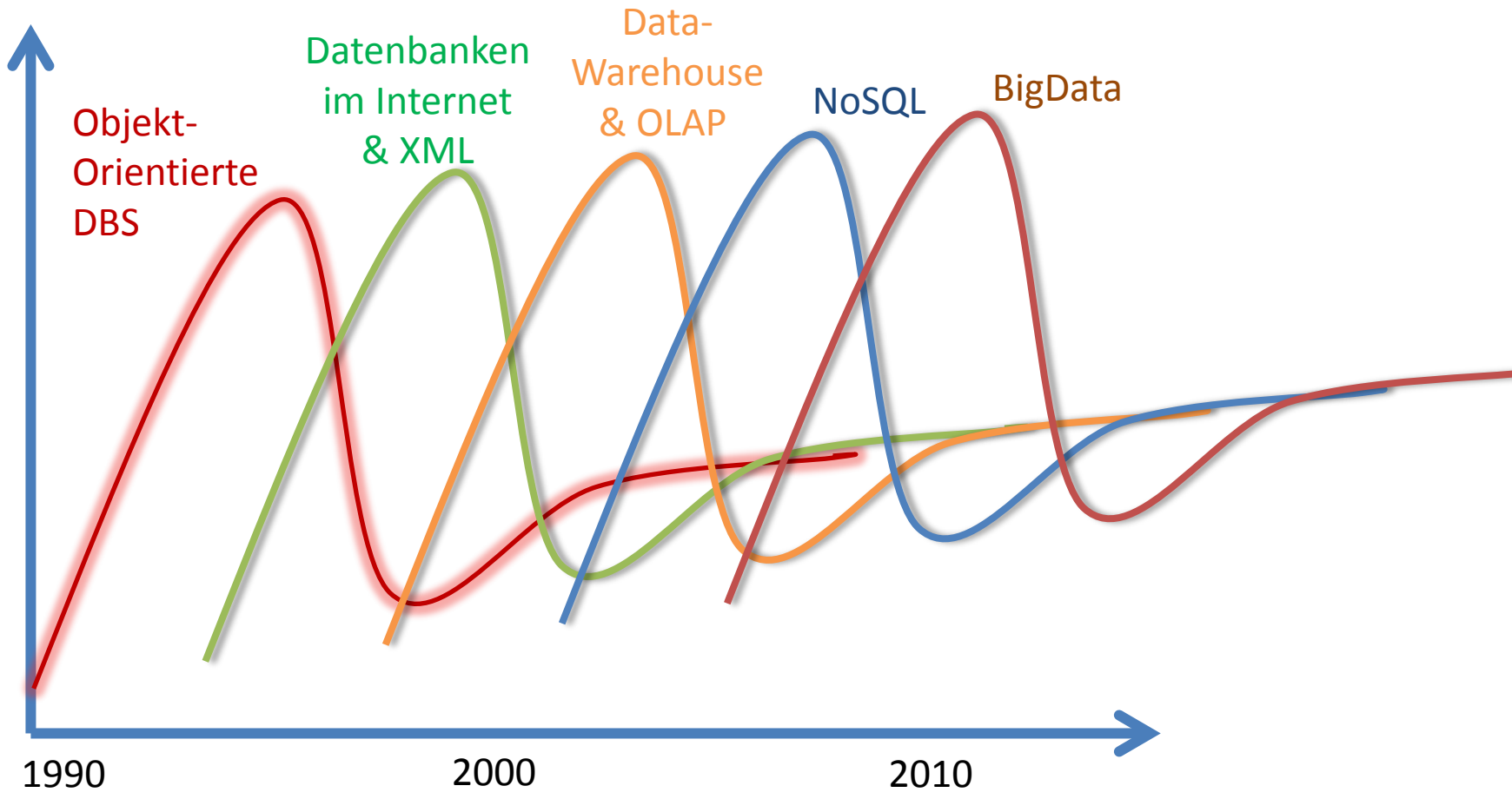
Veracity: ***Unsicherheit*** der Datenquellen, fehlende Qualitätsstandards

Gartner: Big Data →„Extreme Information Processing“

nicht nur pure Speicherung von Daten, sondern insbesondere schnelle und flexible Verarbeitung von Massendaten

„Hype“-Zyklus von Datenbanktechnogien

Der **Hype-Zyklus** stellt dar, welche Phasen der öffentlichen Aufmerksamkeit eine **neue Technologie bei deren Einführung** durchläuft.

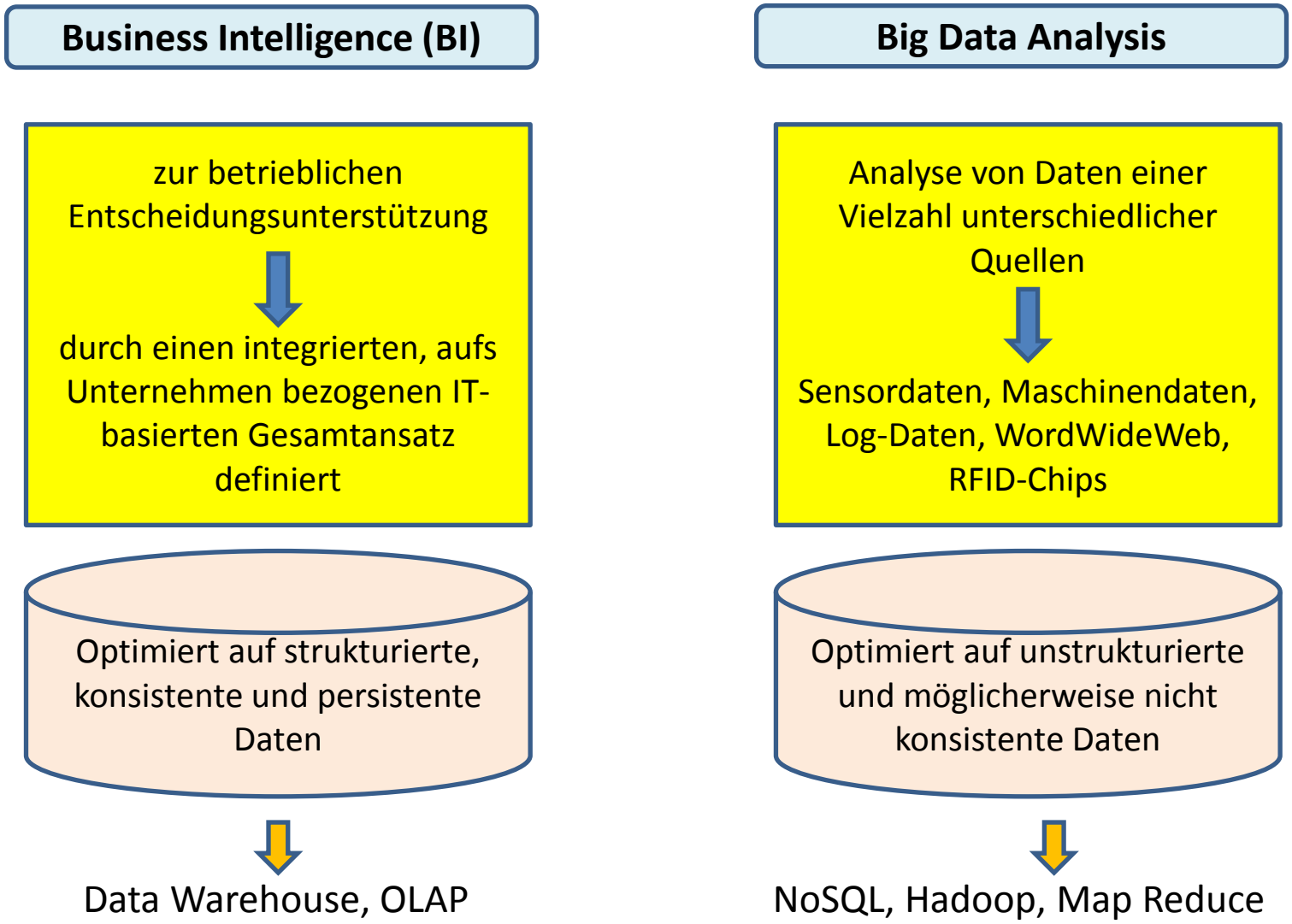


Datenbank-Technologien für BigData

- **Data Warehousing** – Speicher- und Analysesysteme zur Verwaltung strukturierter (aktueller und historischer) Datenbestände
  Beispiele: MS OLAP Server, SAP HANA, Oracle DW, Sybase IQ
- **Massively Parallel Processing** (**MPP**) or **parallel DBMS** – Systeme zur Verwaltung von massiven Datenbankzugriffen, die konkurrierend und schreibend auf große Datenbestände zugreifen.
  Beispiele: EBay DW, Yahoo! Everest Architecture, Greenplum, AsterData
- **Column-oriented database** and **In-Memory-Technology** – Spaltenorientierte Speicherung von Datenbankinhalten mit der Zielrichtung eines schnellen lesenden Zugriffs
  Beispiele: SAP HANA, Vertica, Sybase IQ, MonetDB
- **Streaming processing** (**ESP** or **CEP**) – Systeme zur Verwaltung konstanter (oder ereignisabhängiger) Datenströme, die über einen bestimmten Zeitraum gespeichert und ausgewertet werden müssen.
  Beispiele: Truviso, DataGRID
- **Key-value storage** (with MapReduce model) or **other NoSQL-DBS** – Systeme zur Verwaltung von Datenmassen, bei denen der Fokus auf einen lesenden Zugriff mit höchster Performance liegt
  Beispiele: viele der NoSQL DBS (Google, Amazon, Facebook)

Siehe Park Kieun: Database Technology for Large Scale Data

Big Data Analytics



allgemeine Zielsetzungen:

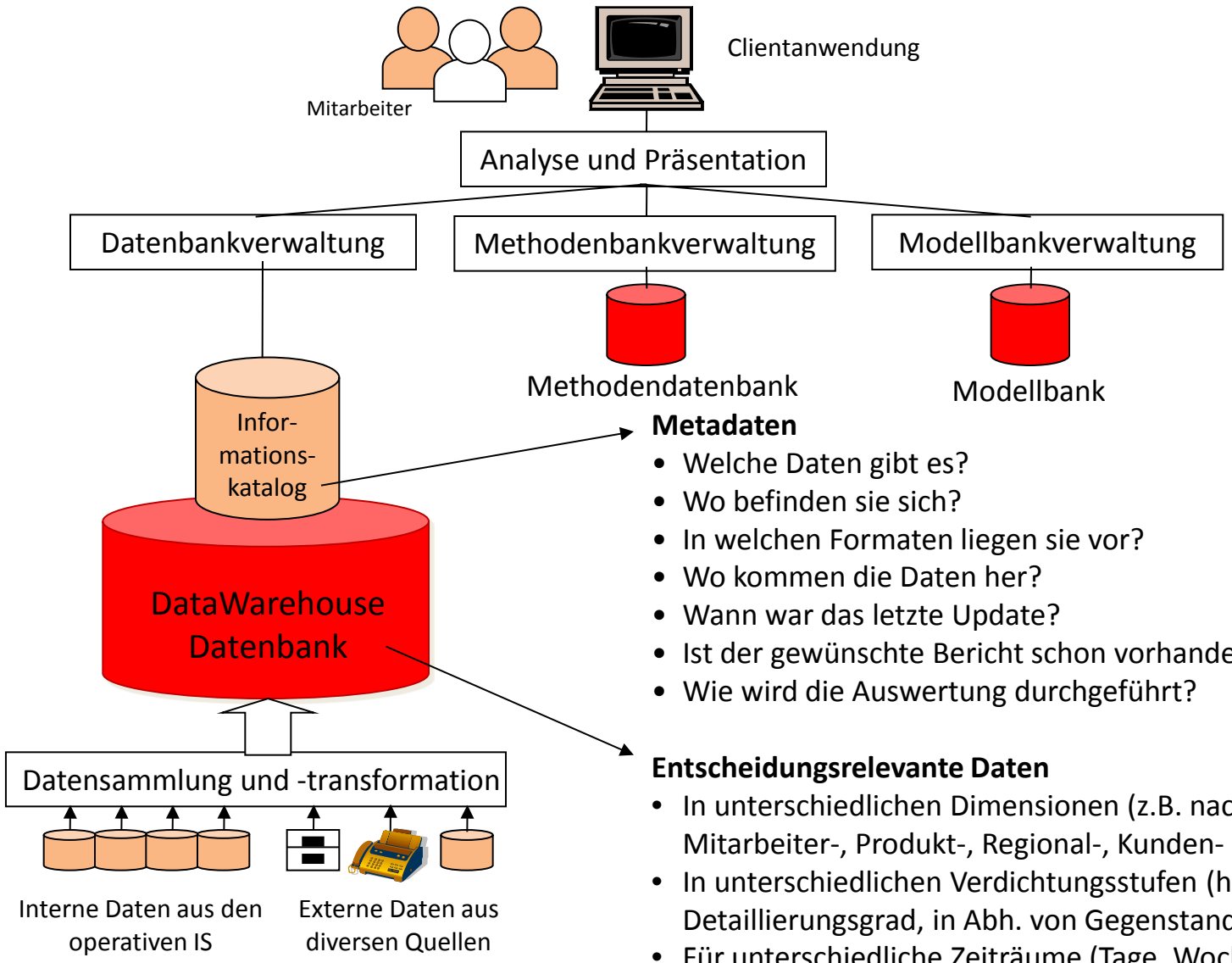
- Verwaltung großer Datenbestände (mit Historisierung, aber keine Archivierung !)
- Aufbau einer zentralen und konsistenten Datenbasis
- losgelöst betrieben von den operativen Datenbanken
- zur Unterstützung von analytischen Aufgaben (Data Mining, Business Intelligence)

Ein **Data Warehouse** ist eine aus einer oder mehreren operativen Datenbanken extrahierte Datenbank, die alle für den Analyseprozess relevanten Daten (eines Unternehmens) zusammenfasst, aufbereitet und aggregiert.

Ein Data Warehouse umfasst die **Meta-, die Dimensions- und die Aggregationsdaten** sowie die **Fakten**, um Informationen verfügbar und informationsgestützte Entscheidungen möglich zu machen.

Darüber hinaus beinhaltet es die notwendigen **Verwaltungsprozesse** wie **Einfügen, Warehousing und Abfrage**.

Beispiel für eine Data Warehouse - Architektur



Metadaten

- Welche Daten gibt es?
- Wo befinden sie sich?
- In welchen Formaten liegen sie vor?
- Wo kommen die Daten her?
- Wann war das letzte Update?
- Ist der gewünschte Bericht schon vorhanden?
- Wie wird die Auswertung durchgeführt?

Entscheidungsrelevante Daten

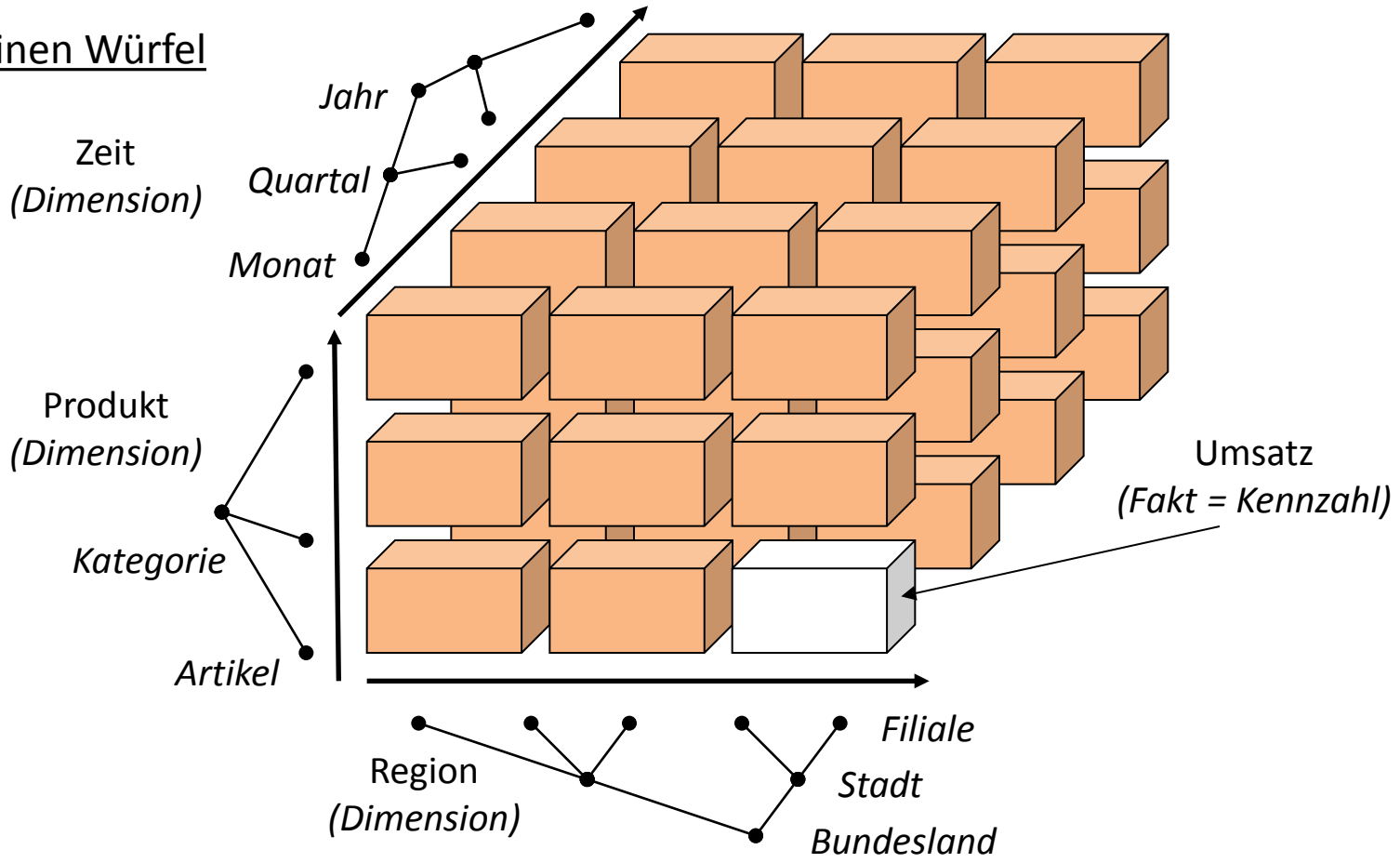
- In unterschiedlichen Dimensionen (z.B. nach Organisations-, Mitarbeiter-, Produkt-, Regional-, Kunden- und Zeitstrukturen ..)
- In unterschiedlichen Verdichtungsstufen (hoher oder geringer Detaillierungsgrad, in Abh. von Gegenstand und Alter der Daten)
- Für unterschiedliche Zeiträume (Tage, Wochen, Monate, Quartale, Jahre)

Veranschaulichung der Multidimensionalität

- Dimensionalität
  Anzahl der Dimensionen
- Kanten = Dimensionen
- Zellen = Fakten

- Visualisierung
  - 2 Dimensionen: Tabelle
  - 3 Dimensionen: Würfel
  - >3 Dimensionen: Multidim. Domänenstruktur

Bsp. für einen Würfel



Strategien zur Speicherung multidimensionaler Daten

relational

- Datenspeicherung im relationalen DBMS
- Modellierung der Cubestruktur mittels Relationen (Tabellen)
- Verwendung des SQL-Standards zur Datenabfrage und -manipulation
- Beispiele:
  - Star Schema
  - Snowflake Schema

ROLAP

(Relational Online Analytical Processing)

nicht relational

- Datenspeicherung nicht in relationaler Art
- Speicherung des Cube mit klassischen Methoden der Informatik
- Fehlende Standards für Datenabfrage und -manipulation
- Beispiele:
  - arrays
  - hash-tables
  - bitmap-indices

MOLAP

(Multidimensional Online Analytical Processing)

Star - Schema

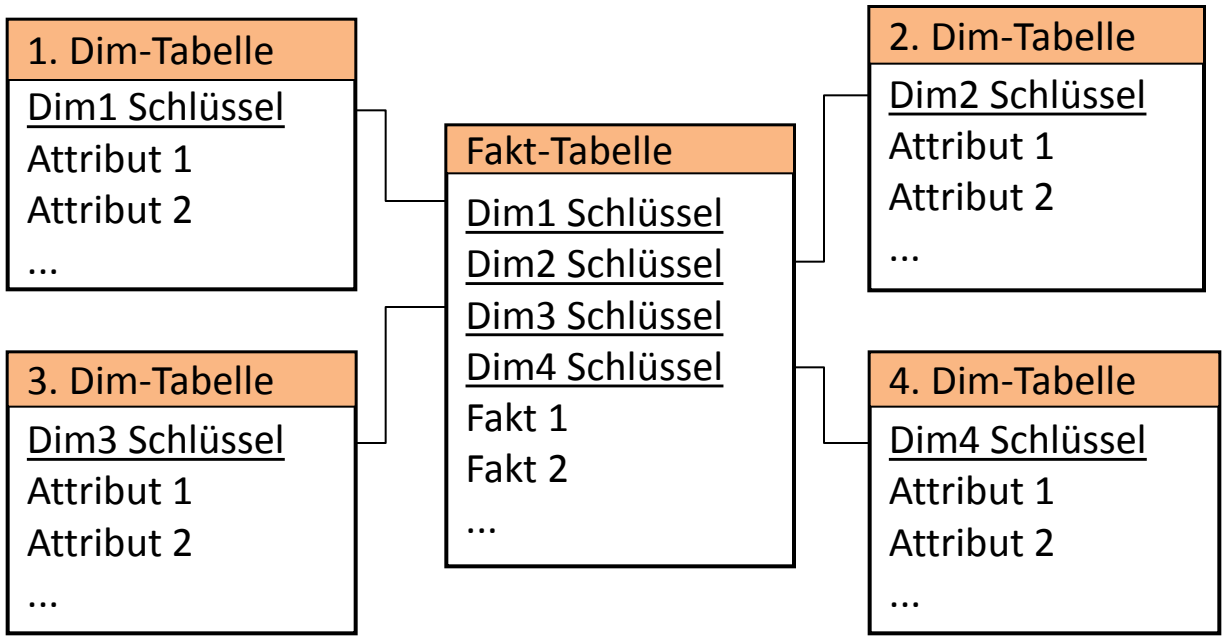
Ein Star-Schema ist ein Datenbankschema, welches so strukturiert ist, dass eine typische Abfrage zur Entscheidungsfindung unterstützt wird.

Im Zentrum des Schemas befindet sich die Fakttabelle, um welche die Dimensionstabellen angeordnet sind.

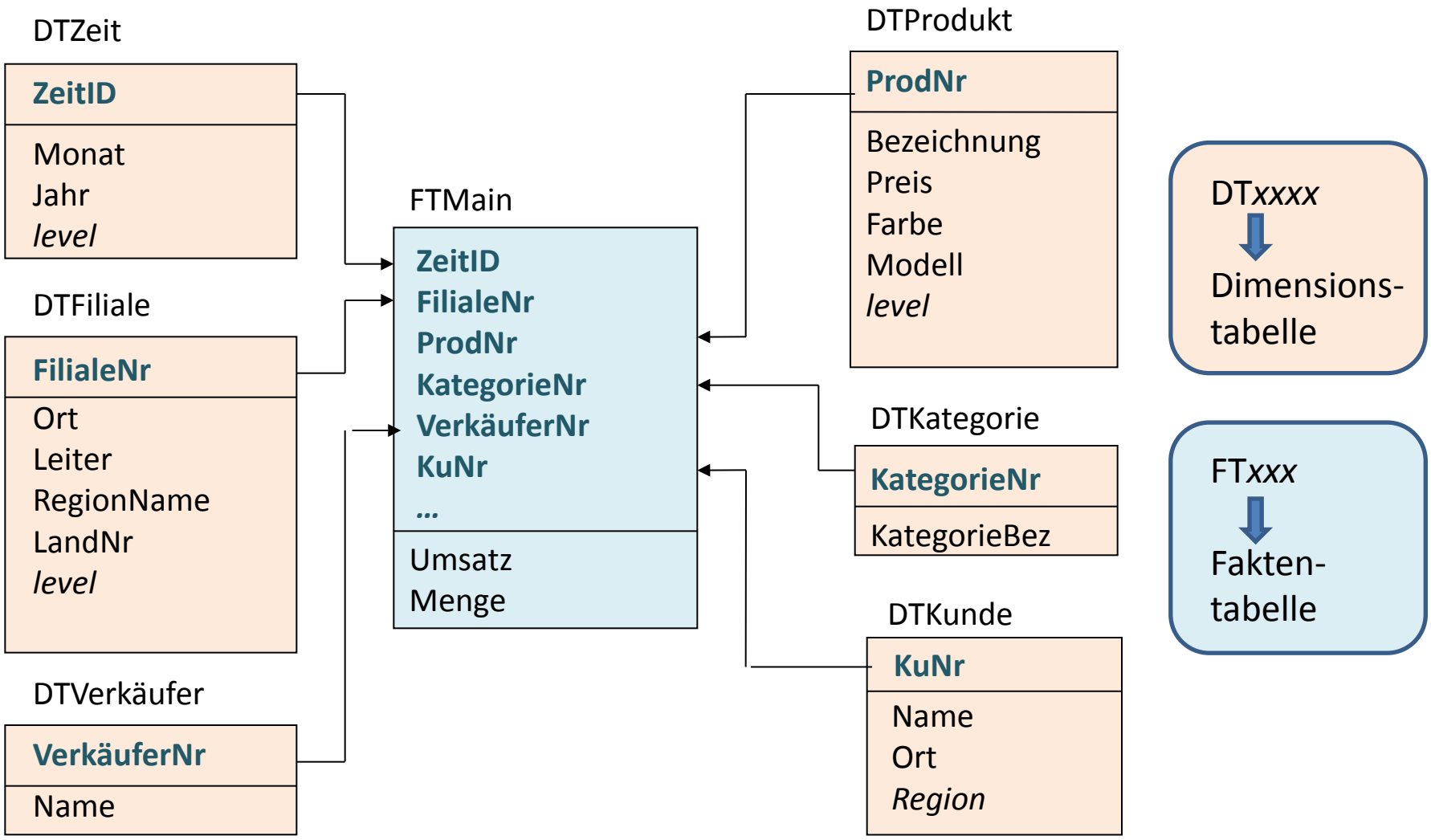
Die Verbindung zwischen den Dimensions-und der Fakttabellen wird über Fremdschlüssel-Primärschlüssel-Beziehungen realisiert.

- Verwendung des relationalen Datenmodells zur Abbildung multidimensionaler Strukturen

Aufbau eines Starschemas:

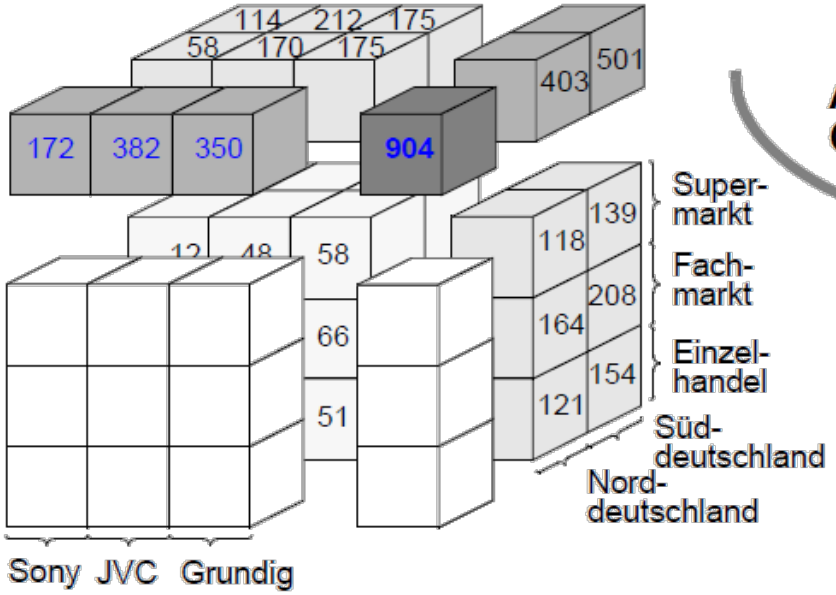


Beispiel für ein Star Schema



Multidimensionale Analyse

Verkäufe(	Region	Geschäftstyp	Marke	Verkäufe)
	Norddeutschland	Supermarkt	Sony	12
	Norddeutschland	Supermarkt	JVC	48
	Norddeutschland	Supermarkt	Grundig	58
	Norddeutschland	Fachmarkt	Sony	31
	Norddeutschland	Fachmarkt	JVC	67
	Norddeutschland	Fachmarkt	Grundig	66
	Norddeutschland	Einzelhandel	Sony	15
	Norddeutschland	Einzelhandel	JVC	55
	Norddeutschland	Einzelhandel	Grundig	51
	Süddeutschland	Supermarkt	Sony	22
	Süddeutschland	Supermarkt	JVC	50
	Süddeutschland	Supermarkt	Grundig	67
	Süddeutschland	Fachmarkt	Sony	51
	...			

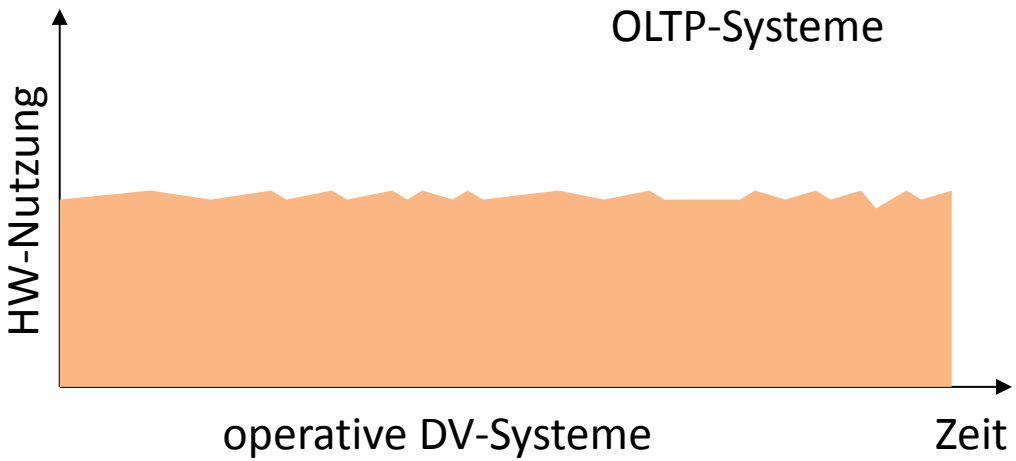


Anwendung des CUBE-Operators

Verkäufe(	Region	Geschäftstyp	Marke	Verkäufe)
	Norddeutschland	Supermarkt	Sony	12
	Norddeutschland	Supermarkt	JVC	48
	Norddeutschland	Supermarkt	Grundig	58
	Norddeutschland	Supermarkt	ALL	118
	Norddeutschland	Fachmarkt	Sony	31
	Norddeutschland	Fachmarkt	JVC	67
	Norddeutschland	Fachmarkt	Grundig	66
	Norddeutschland	Fachmarkt	ALL	164
	Norddeutschland	Einzelhandel	Sony	15
	Norddeutschland	Einzelhandel	JVC	55
	Norddeutschland	Einzelhandel	Grundig	51
	Norddeutschland	Einzelhandel	ALL	121
	Norddeutschland	ALL	ALL	403
	Süddeutschland	Supermarkt	Sony	22
	...			
	Süddeutschland	ALL	ALL	501
	ALL	Supermarkt	Sony	34
	ALL	Supermarkt	JVC	98
	ALL	Supermarkt	Grundig	155
	ALL	Supermarkt	ALL	257
	ALL	Fachmarkt	Sony	82
	...			
	ALL	ALL	Sony	172
	ALL	ALL	JVC	382
	ALL	ALL	Grundig	350
	ALL	ALL	ALL	904

[Quelle: Lehner – DBST 2003]

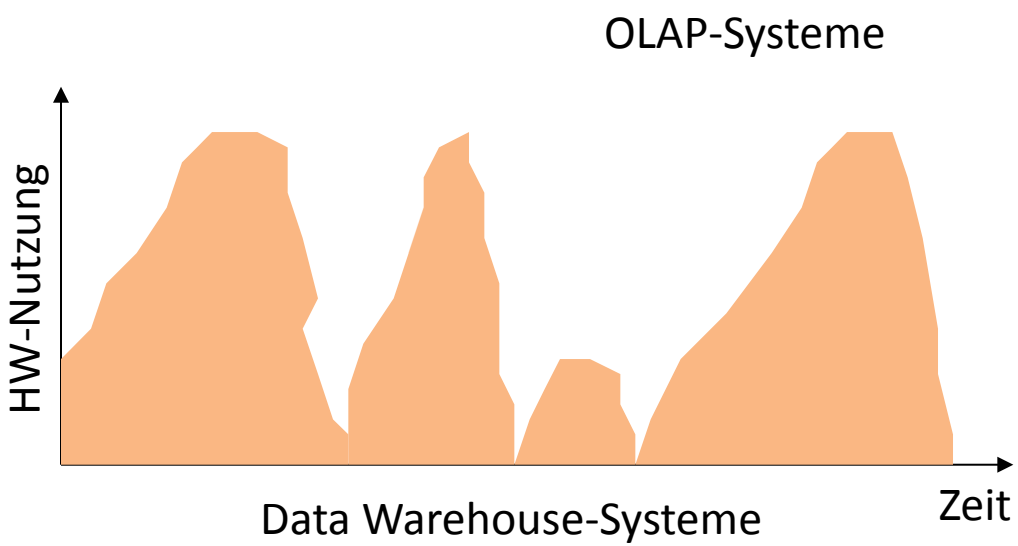
Vergleich OLTP - OLAP



Klassische operative DBS

→ **Online Transactional Processing (OLTP)**

– Erfassung und Verwaltung von Daten im Mittelpunkt
– Transaktionale Verarbeitung
– kurze Lese-/ Schreibzugriffe auf
– wenige Datensätze



Data Warehouse Systeme

→ **Online Analytical Processing (OLAP)**

– Analyse im Mittelpunkt
– lange Lesetransaktionen auf vielen Datensätzen
– Integration, Konsolidierung und Aggregation der Daten

Vergleich der Systemauslastung

Zugriffs- und Speicheroperationen von DBMS

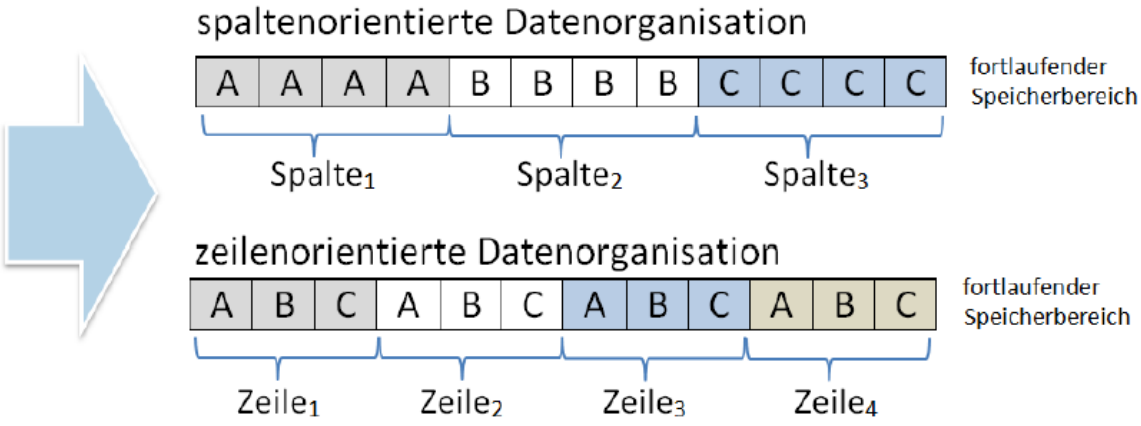
Charakteristika von Operationen in verschiedenen Anwendungsszenarien:

- **OLTP: Lese-** und **Schreib**operationen auf einzelnen Datensätzen
  - beim kompletten Lesen einer Tabelle werden viele nicht benötigte Datensätze gelesen
  - Optimierung des gezielten Zugriffs auf Datensätze mit Indexen
- **OLAP: Lese**operationen auf vielen Datensätzen (wobei i.a. nur einzelne Attribute relevant sind)
  - beim kompletten Lesen einer Tabelle werden viele nicht benötigte Attributwerte gelesen

    → Optimierung durch viele Indexe (ggf. Index auf jedem Attribut)
    → Alternative: vertikale Partitionierung
    →Neuer Ansatz:

Column-Oriented Database Systems (Spaltenorientierte DBMS)

	$Spalte_1$	$Spalte_2$	$Spalte_3$
$Zeile_1$	A	B	C
$Zeile_2$	A	B	C
$Zeile_3$	A	B	C
$Zeile_4$	A	B	C



Vergleich zeilen- und spaltenorientierter Speicherung

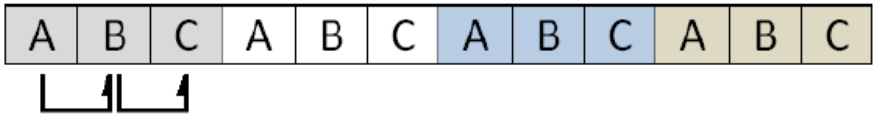
Kriterium	Zeilenorientierte Speicherung im RDBMS	Spaltenorientierte Speicherung im IDBMS
Ort der Datenhaltung	Datenbank liegt nicht im Hauptspeicher, daher Verarbeitung zwischen Arbeitsspeicher und Festplatte notwendig.	Datenbank liegt im Hauptspeicher somit schnellere Verarbeitung von Daten.
Daten-Organisation/ -Management	Effiziente Schreibprozesse, da Daten als gesamt Konstrukt eingefügt/aktuallisiert werden können.	Schnelle Verarbeitung beim Lesen von Daten, da u.a. nur relevante Spalten gelesen werden müssen.
Darstellungs-möglichkeiten	Für zweidimensionale Darstellungen sehr gut geeignet, stößt jedoch an Grenzen bei Darstellung von komplexerer Szenarien.	Sehr gut für große Datenmengen, komplexe Sachverhalte können gut dargestellt werden.
Echtzeitverarbeitung	Nicht geeignet, aufgrund schlechter Performance insbesodere für große Datenmengen.	Daten können in Echtzeit verarbeitet werden.
INSERT/ UPDATE/ DELETE	Änderungen sowie das Einfügen von Datensätzen kann schnell umgesetzt werden. Bildung von Aggregaten ist zeitintensiv.	Bei Änderungen bzw. Anfügen von Daten erhöhter Aufwand, da mehrere Schreiboperationen notwendig sind. Dafür können Aggregate schneller gebildet werden.

Vergleich zeilen- und spaltenorientierter Verarbeitung

Zeilenorientierte Datenorganisation

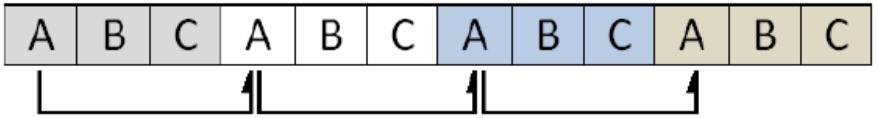
Zeilenoperationen

SELECT * FROM ORDER WHERE ...



Spaltenoperationen

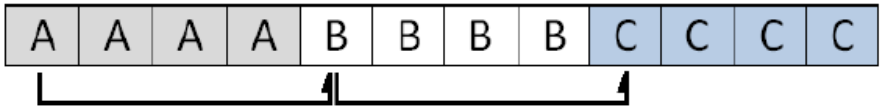
SELECT SUM(VERKAUFSPREIS) FROM ORDER



Spaltenorientierte Datenorganisation

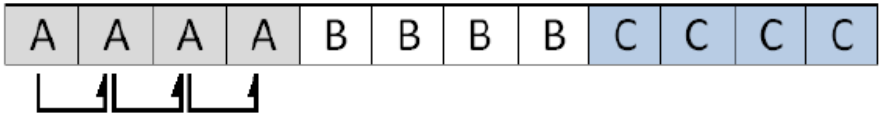
Zeilenoperationen

SELECT * FROM ORDER WHERE ...

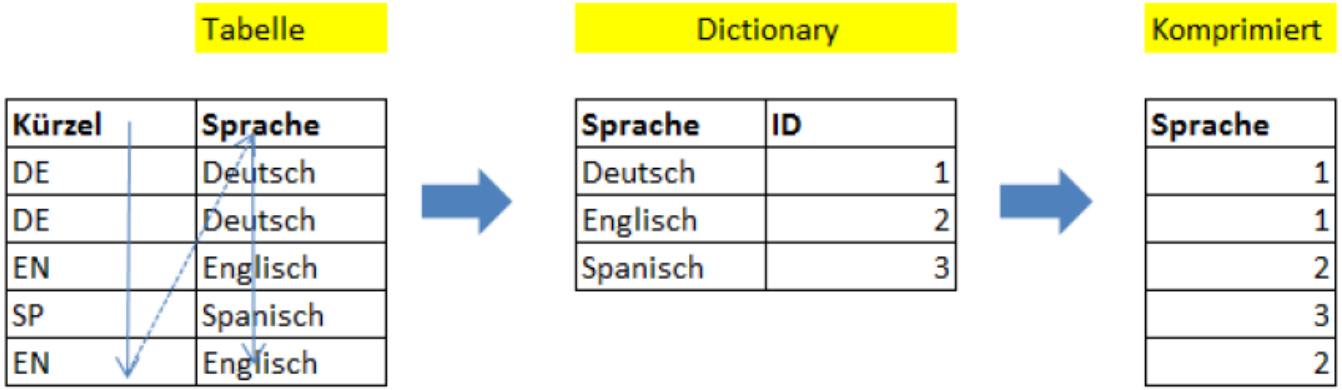


Spaltenoperationen

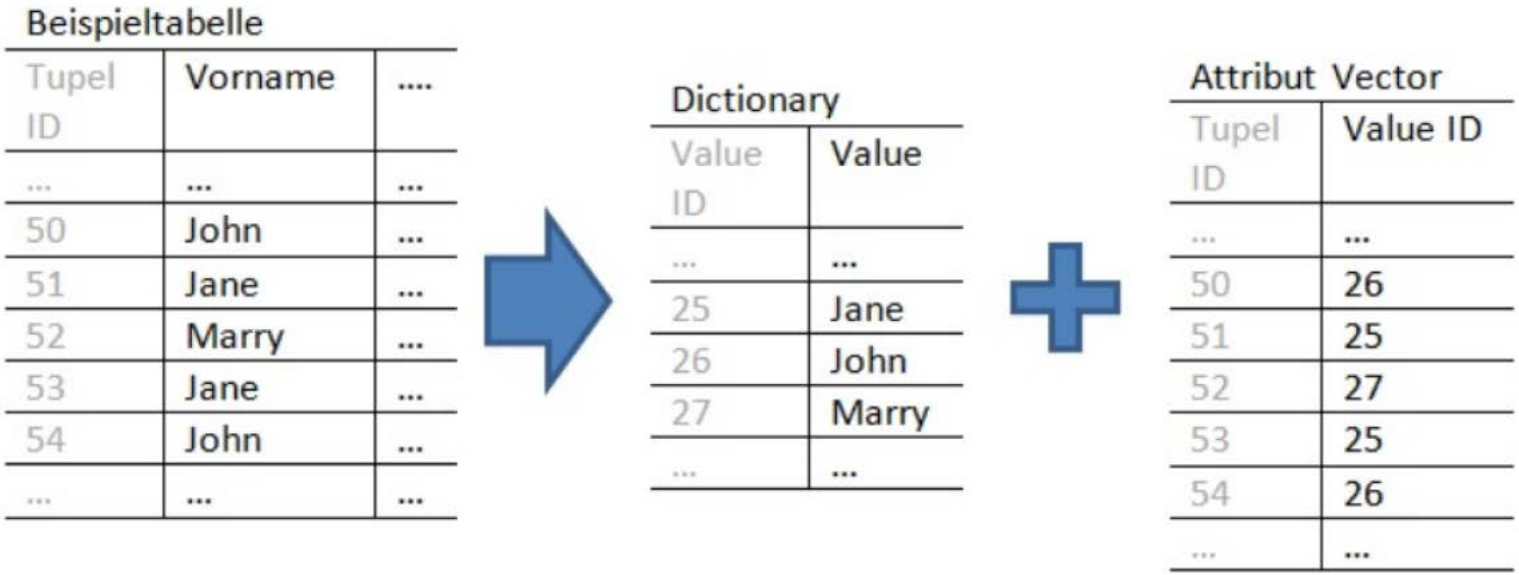
SELECT SUM(VERKAUFSPREIS) FROM ORDER



Komprimierung bei spaltenorientierter Speicherung



In SAP HANA wird eine Tabellenspalte in ein **Dictionary** mit einzigartigen Spaltenattributen und ein **Attribute Vektor** zerlegt:



Vertikale Partitionierung und Bitmap Index

vertikale Partitionierung

Einr	Typ	Land	Prod	Abs
32	G	SA	Werne	6
36	G	MV	Becks	9
38	G	SA	Radeb	5
41	K	NS	Jever	11
43	G	SA	Radeb	9
46	G	BY	Paula	3
47	M	NW	Dortm	7
49	K	SA	Lands	12

Bitmapindex am Beispiel der Spalte Land

Bitmap Index für Land

row-id		BY	MV	NS	NW	SA
1		0	0	0	0	1
2		0	1	0	0	0
3		0	0	0	0	1
4		0	0	1	0	0
5		0	0	0	0	1
6		1	0	0	0	0
7		0	0	0	1	0
8		0	0	0	0	1

[Quelle: Bittner – 160. Datenbank-Stammtisch]

Vergleich zeilenorientiertes und spaltenorientierte DB

	spaltenorientierte Datenorganisation	zeilenorientierte Datenorganisation
Vorteil	*effizienter beim Aggregieren von Daten*	*effizienter bei den Abfragen mit großer Anzahl an Datenspalten*
	geeignet für umfangreiche Datenkomprimierung	*effizienter beim Schreiben vieler Datensätze*
Nachteile	*schlechtere Performance bei den Schreiboperationen*	*schlechtere Performance bei Aggregation von Daten*
	schlechtere Performance bei den Abfragen mit großer Anzahl an Datenspalten	*schlechtere Möglichkeiten bei der Datenkomprimierung*

Anwendungen:

- Sybase Adaptive Server IQ seit 1996
  IQ Multiplex
- SAP HANA
- In Memory DB von MS SQL Server und Oracle
- ParAccel, Vertica seit 2008

Vergleich RAM- und HD-Zugriff

GAP - Zugriffs- und Lesezeiten von Festplatte und Hauptspeicher:

Aktion	**Zeit**
Main Memory Access	100 ns
Read 1 MB Sequentially from Memory	250,000 ns
Disk Seek	5,000,000 ns
Read 1 MB Sequentially from Disk	30,000,000 ns

storage hierarchy		**What does it mean in "our dimensions"?**	
storage type	rel. access time	location	access time
register	1	my head	1 min
cache (on chip)	2	this room	2 min
cache (on board)	10	this building	10 min
main memory	100	Leipzig (by car)	~2 h
magnetic disk	10^6-10^7	Pluto ($5910 * 10^6$ km)	>2 years
tape/opt. storage (automat. loading)	10^9-10^{10}	Andromeda	>2000 years

[Quelle: Härder – 150. Datenbank-Stammtisch]

Scale up vs. Scale out

Scale up:
wenige, große Server

Scale out:
viele, kleinere
(Commodity-)Server

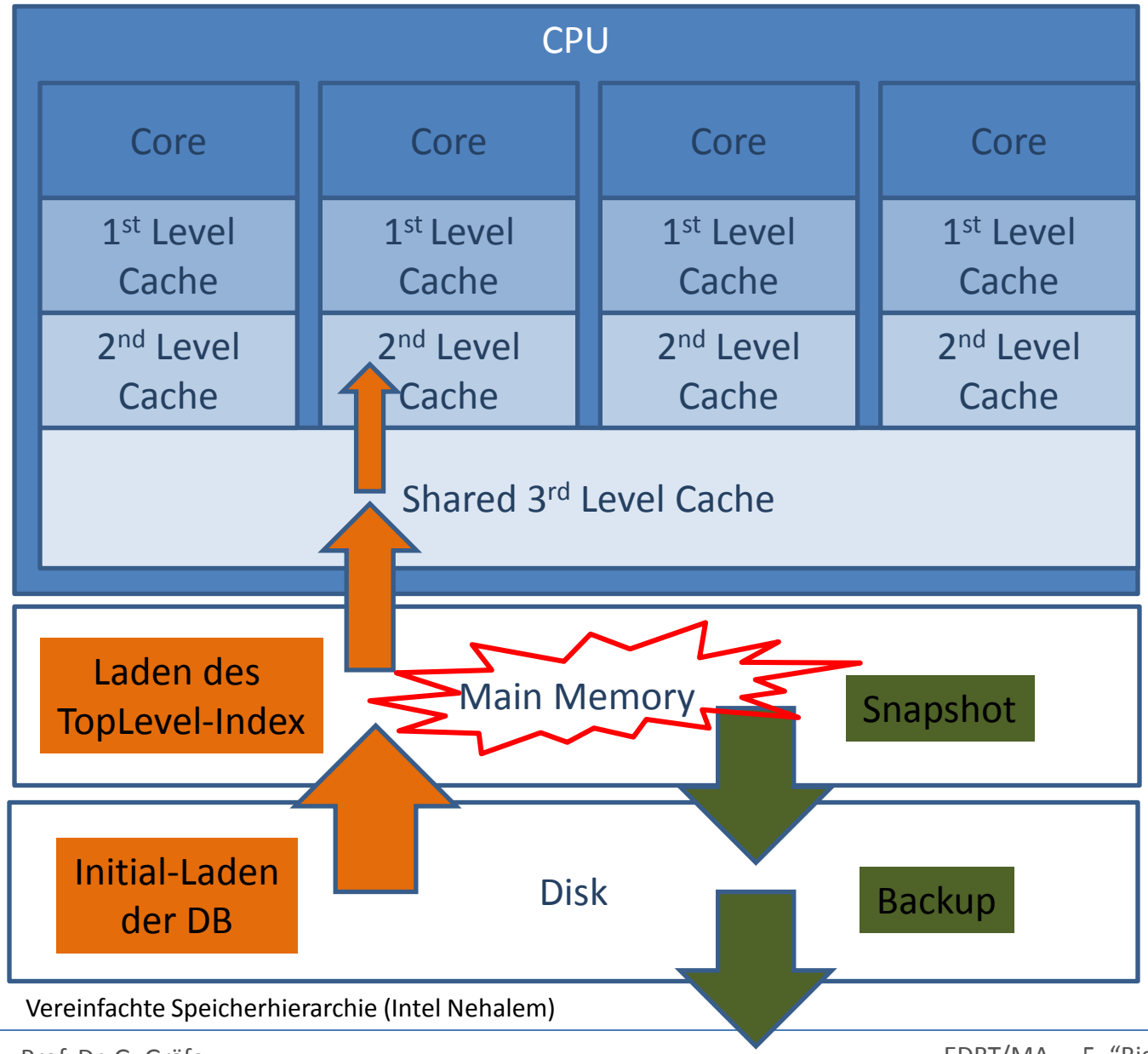
Scale up vs. Scale out

	Scale up	Scale out
Vorteile	• transparent für DBMS • Administrationsaufwand konstant	• Kostengünstigere Hardware • Skalierung in kleineren Stufen möglich
Nachteile	• Hardware-Kosten • Skalierung nur in größeren Stufen möglich ⇨ höhere Kosten und ungenutzte Leistung	• Last- und Datenverteilung notwendig • Ggf. verteilte Protokolle (2PC, Replikation) • Erhöhte Fehlerrate (mehr und einfachere Hardware) • Erhöhter Administrationsaufwand

Microsoft Massive Data Center Northlake, Illinois (US)

[Quelle: Lehner – 155.DBST]

Speicherhierarchie bei Main-Memory-DBMS

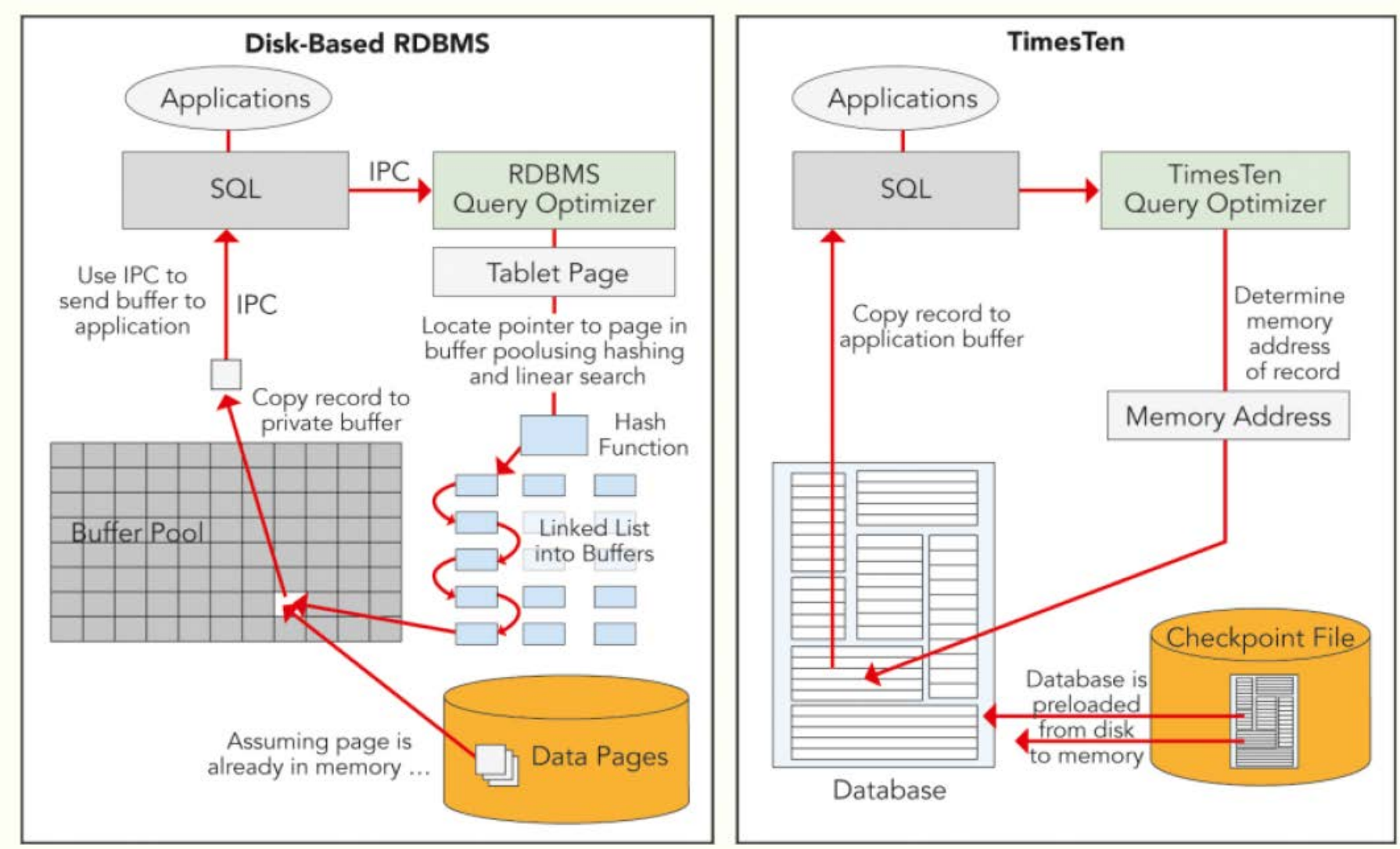


Size	Latency
64KB	~4 cycles 2 ns
256KB	~10 cycles 5 ns
8MB	35-40+ cycles 20 ns
several GBs up to TBs	~ 200 cycles ~ 100ns
several TB	several million cycles several ms

Vereinfachte Speicherhierarchie (Intel Nehalem)

Quelle: Färber – SAP AG

Vergleich relationaler und Main-Memory-DBMS



(Quelle: Ott und Stirnimann 2012]

Merkmale von InMemory-DBS

InMemory- bzw. MainMemory-DBS → Caching der Datenbank im Hauptspeicher

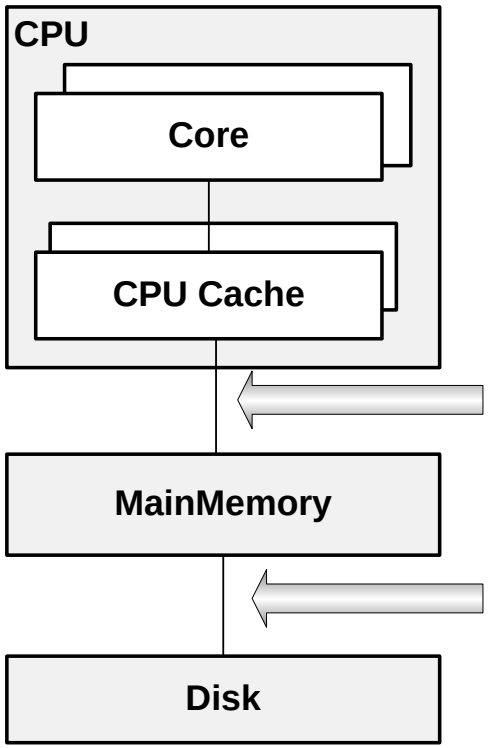
Möglichkeiten:
- Preis/MB RAM fällt immer weiter
- 64-Bit-Architekturen ermöglichen Nutzung von 2^x GB Hauptspeicher

„Disk is Tape“

- Festplattenzugriff 105 mal langsamer als Hauptspeicherzugriff
- Kein Festplattenzugriff während der normalen Operationen
- Nutzung der Festplatte für Backup bzw. Archiv

Problem:
- ➢ Flaschenhals Hauptspeicher - Cache

Main Memory Bottleneck



Performance bottleneck today: CPU waiting for data to be loaded from memory into cache

Performance bottleneck in the past: Disk I/O

Quelle: Färber – SAP AG

Oracle Exadata

Vorkonfigurierte RAC Database Machine

- DBS: 64 Nehalem Cores, 576 GB RAM
- Netzwerk: InfiniBand 40 Gbit/s
- Exadata Storage Server:
- 112 Nehalem Cores, 336 GB RAM
- 100 TB SAS HDD
- 5 TB Flash Cache (SSD)

Flash Cache

- automatischer Write-through Cache
- von Hand:

```
create table emp( ... )
storage (flash_cache keep)
```

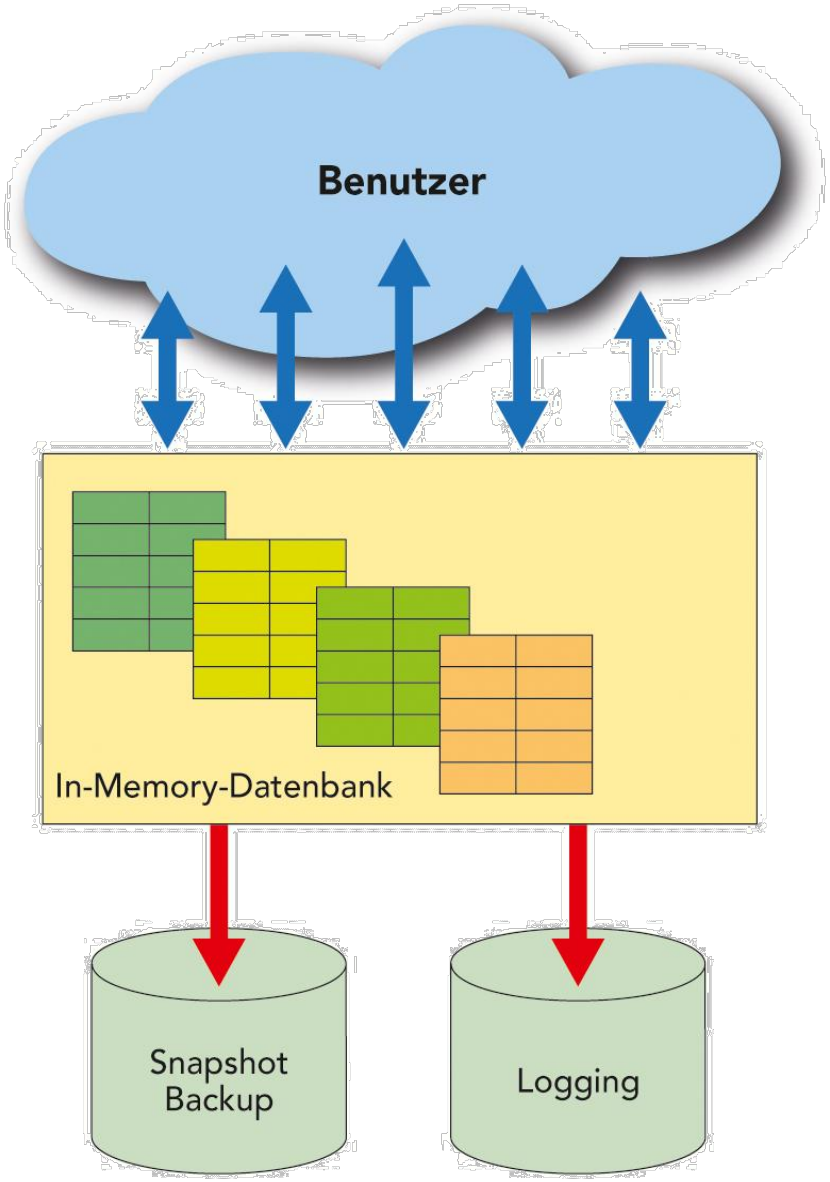
[Quelle: Oracle]

InMemory-Computing und ACID

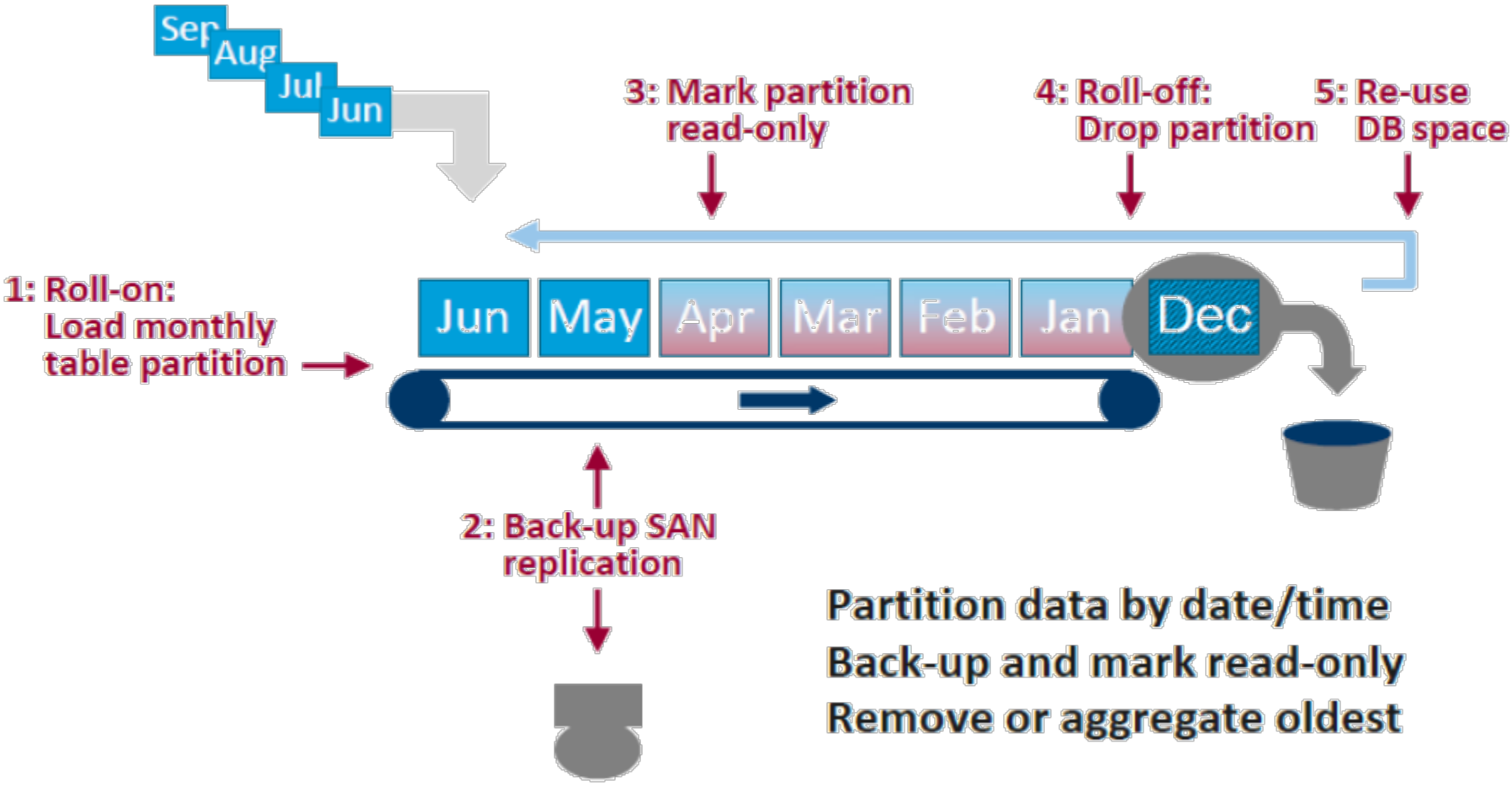
In-Memory-Datenbanken speichern Daten in flüchtigem Arbeitsspeicher, dieser kann erfolgreich abgeschlossene Transaktionen aber beispielsweise bei Systemabstürzen **nicht dauerhaft** halten.
→ ACID-Prinzip wird verletzt

Strategien:

- **Snapshot Dateien** speichern den Zustand der Datenbank zu bestimmten Zeitpunkten (zeitgesteuert oder beim kontrollierten Abschalten der Datenbank)
  → neueste Änderungen gehen ggf. verloren
- **Transaction Logs** speichern Änderungen auf Dateien und ermöglichen somit die automatische Wiederherstellung der Datenbank.
- **Non-Volatile Random-Access Memory** (Kombination eines herkömmlichen flüchtigen RAM-Speichers mit einem Energiespeicher)
- **2PC mit Replikation und Failover** auf eine identische herkömmliche Datenbank
  → Problem Performance



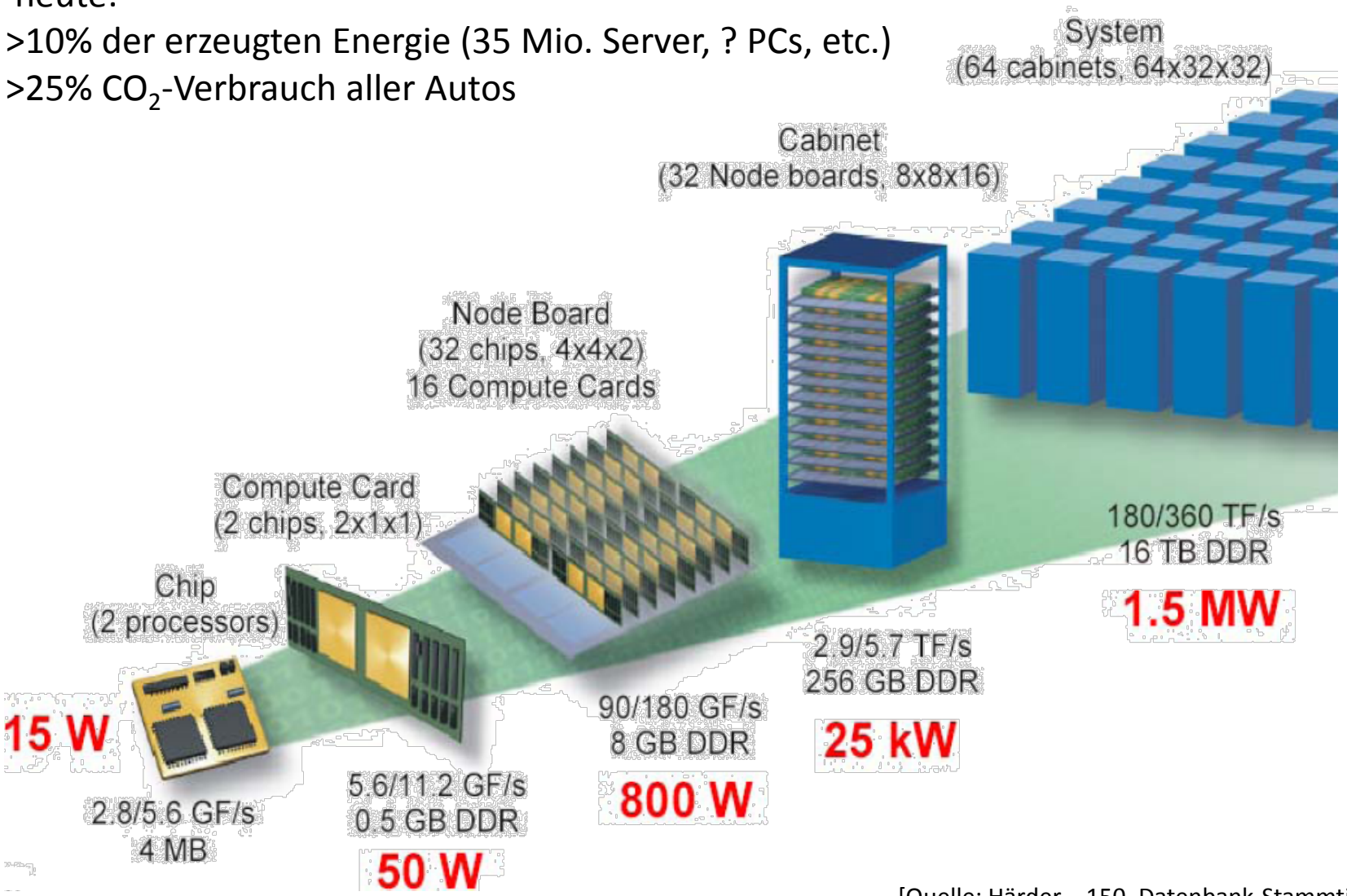
Information LifeCircle Management bei InMemory-DBS



Energieverbrauch von Main-Memory-DBMS

IKT heute:

- >10% der erzeugten Energie (35 Mio. Server, ? PCs, etc.)
- >25% CO_2-Verbrauch aller Autos



[Quelle: Härder – 150. Datenbank-Stammtisch]

5.4. NoSQL - Kategorisierung

NoSQL-Systeme: Schlanke, effiziente DBS, die auf eine performante und kostengünstige Verwaltung einfach strukturierter, sehr sehr großer Datenmengen (*Big Data*) spezialisiert sind

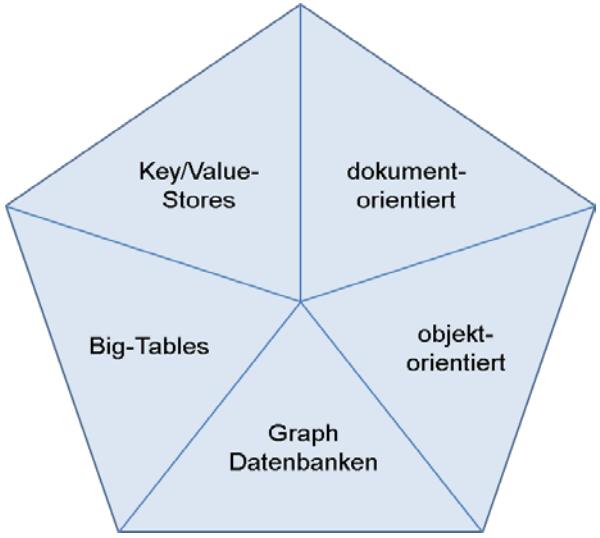
Merkmale:
- im Wesentlichen Select- und Insert-Operationen
- geringe Anforderungen an Konsistenz der DB

(Noch) keine einheitliche **Klassifikation**

→ eine mögliche und relativ verbreitete Kategorisierung in Anlehnung an ***http://nosql-database.org/***

Core NoSQL Systems:
- Key Value / Tuple Store
- Wide Column Store / Column Families
- Document Store
- Graph Databases

Soft NoSQL Systems:
- Object Databases
- Grid & Cloud Database Solutions
- XML Databases
- Multivalue Databases
- …



NoSQL-Datenbanksysteme: Merkmale

Eine mögliche Definition …

Unter **NoSQL** wird eine neue Generation von Datenbanksystemen verstanden, die meistens einige der nachfolgenden Punkte berücksichtigen:

- Das zugrundeliegende Datenmodell ist **nicht relational**.
- Das System ist **schemafrei** oder hat nur schwächere Schemarestriktionen. (das heisst: man kann ein neues Feld einfügen, ohne erst die Datenstruktur aller anderen Datensätze um ein solches zu erweitern.)
- Das System bietet eine **einfache API**.
- Die Systeme sind i.d.R. von Anbeginn an auf eine **horizontale Skalierbarkeit** ausgerichtet.
- Dem System liegt meistens auch ein **anderes Konsistenzmodell** zugrunde: *Eventually Consistent* und *BASE*, aber nicht *ACID*
- Das NoSQL-System ist **Open Source**.

[Edlich et al: 2011 bzw. http://nosql-database.org/]

Transaktionsbegriff in NoSQL-Systemen

- Viele NoSQL-Systeme garantierten nur **Atomarität einer einzelnen Operation** – wenige Systeme bieten operationsübergreifende Konsistenzmodelle
- Häufig **BASE** mit Eventual Consistency
  - Ermittlung der korrekten Version: häufig Verwendung von **Vector Clocks**

Key-Value Databases

- **Datenmodell :**
  - ein sehr einfaches Datenmodell: unter einem Schlüssel wird ein Wert gespeichert
  - Key-Value-Paare mit eindeutigem Key („the big hash table“)
  - Key und Value enthalten Byte-Arrays = beliebige, serialisierte Datentypen (für value auch beliebig komplex)
  - Typische Grundoperationen:
    - set (key, value)
    - value = get (key)
    - delete (key)

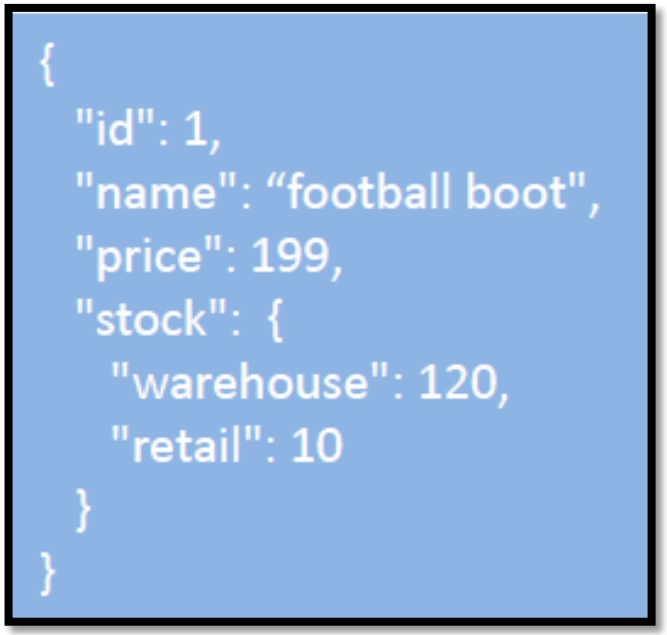
key	value
key	value
key	value
key	value
key	value

- **Indexstrukturen**:
  - Hash-Maps, B*-Bäume auf key

- **Systeme:**
  Amazon Dynamo / S3, Redis, Riak, Voldemort, …

Document Databases

- **Datenmodell**
  - haben ein flexibles Datenmodell
  - Kleinste logische Einheit: „Dokument“ identifiziert über documentID
  - Format i.a. **JSON, BSON**, YAML, RDF
  - Schemafrei, d.h. Anwendung übernimmt Schema-Verantwortung
  - die Datensätze können in einer Datenbank beliebige Strukturen haben



- **Indexstrukturen**
  - B-Baum-Index für documentID
  - Teilweise auch B-Baum-Indexe für Datenfelder in Dokumenten

- **Systeme**

  MongoDB, CouchDB, …

Gegenüberstellung Relationale DBMS / SQL - MongoDB

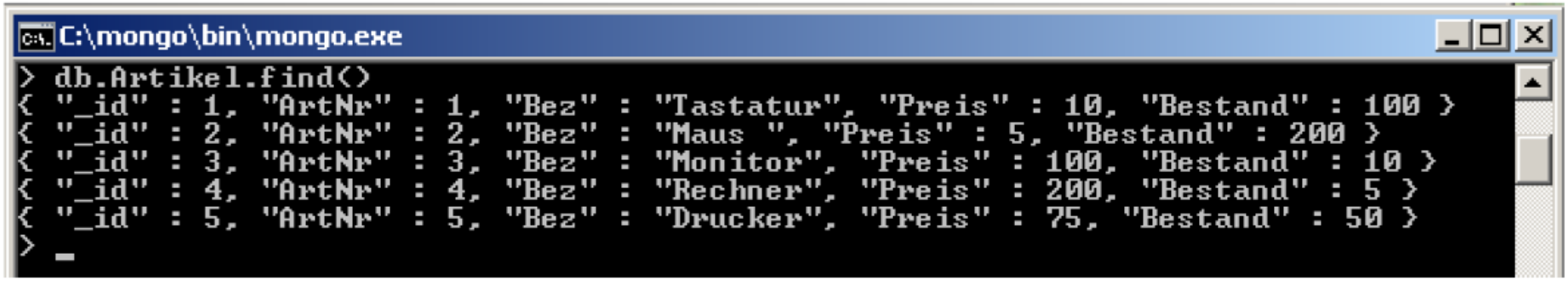
SQL:		MongoDB:
• Create	⟷	• Erfolgt explizit
• Insert	⟷	• db. Artikel.insert({a:1,b:1})
• Select	⟷	• db. Artikel.find({}, {a:1,b:1 })
• Delete	⟷	• db. Artikel.remove({z:'abc'});
• Create Index	⟷	• db. Artikel.ensureIndex({xy:1})
• Update	⟷	• db. Artikel.update({b:'q'}, {$set:{a:1}}, false, true)

Nachteil gegen über RDBMs:
Transaktionen nicht möglich!

Beispiel Datenstruktur in MongoDB

Relationale Struktur und MongoDB-Collection von „Artikel“

ARTNR	BEZ	PREIS	BESTAND
1	Tastatur	10.00	100
2	Maus	5.00	200
3	Monitor	100.00	10
4	Rechner	200	5
5	Drucker	75	50



Wide Column Store / Column Families

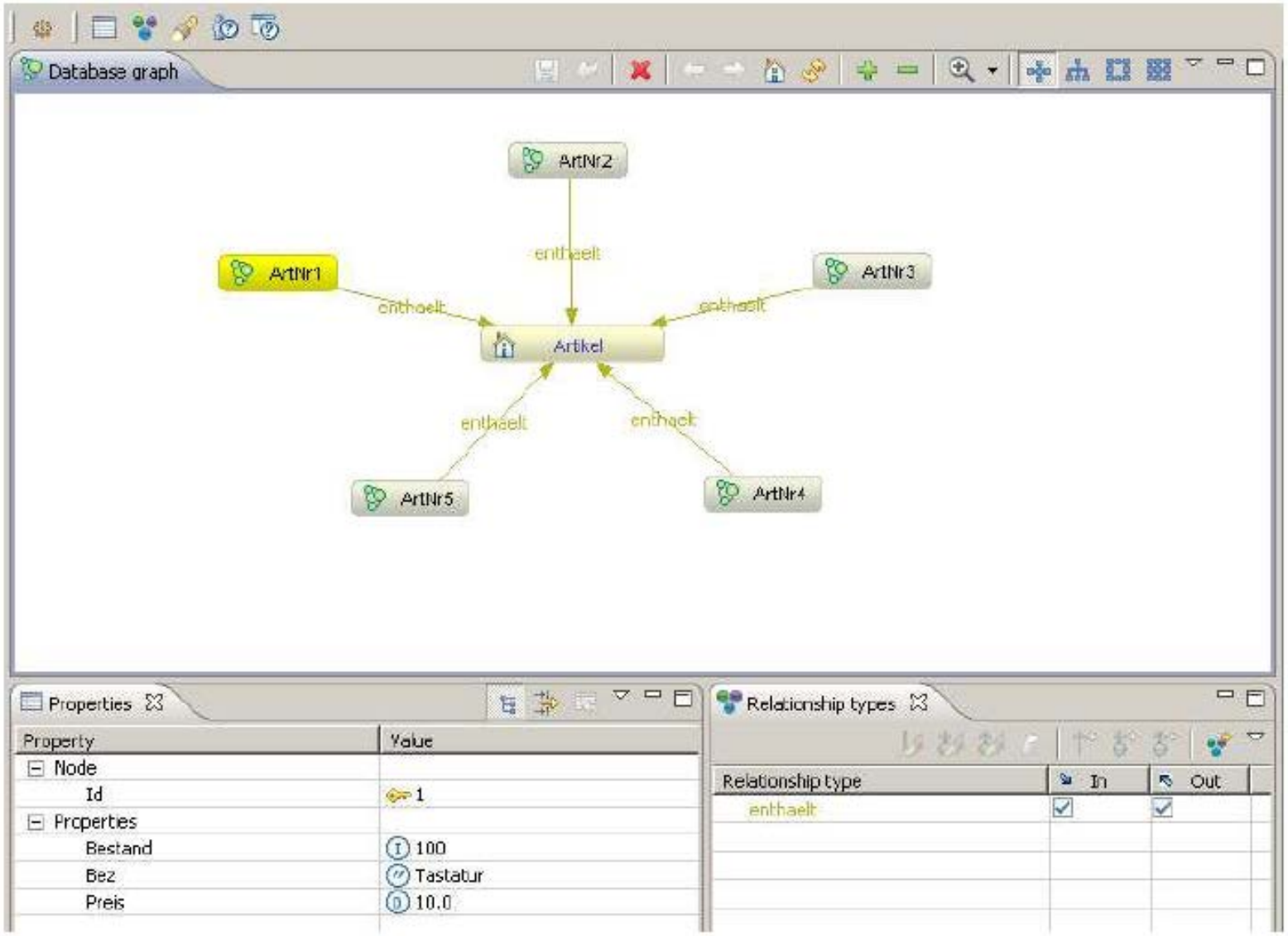
- **Datenmodell**
  - Datenmodell ist mächtiger und dem relationalen Modell ähnlich, das heißt, es basiert auf einzelnen Tabellen
  - spaltenorientierte Speicherung → mehr Performanz für bestimmte Abfragen
  - z.B. Aggregieren innerhalb einer Spalte
  - flexibleres Schema
  - keine ’echten’ Beziehungen
- **Systeme**

  Cassandra, BigTable,. . .

```
cassandra> get Keyspace1.Standard1['Artikel']
=> (column=4172744e7235, value=Drucker, 75, 50, timestamp=1280820767646000)
=> (column=4172744e7234, value=Rechner, 200, 5, timestamp=1280820739760000)
=> (column=4172744e7233, value=Monitor, 100, 10, timestamp=1280820660276000)
=> (column=4172744e7232, value=Maus, 5, 200, timestamp=1280820638150000)
=> (column=4172744e7231, value=Tastatur, 10, 100, timestamp=1280820605545000)
Returned 5 results.
cassandra> get Keyspace1.Standard1['Artikel']['ArtNr1']
=> (column=4172744e7231, value=Tastatur, 10, 100, timestamp=1280820605545000)
cassandra>
```

Die „Artikel“-Relation als SuperColumn in Cassandra.
Darunter die einzelne SubColumn ArtNr1

Graphdatenbanken



Graphendatenbank – die SQL-Relation „Artikel“ in Neoclipse

Parallelisierung der Anfrageverarbeitung

- **Scale out** – Skalierung auf viele (kleinere) Server
- Aufgrund der verteilten Architektur unterstützt das System eine einfache Datenreplikation.
- Parallele Verarbeitung sehr großer Datenmenge erfordert **neue Algorithmen** und Frameworks
- (alte) Idee aus funktionaler Programmierung (LISP, ML etc.)
  - Operationen ändern die Daten nicht, sondern arbeiten immer auf neu erstellten Kopien (→ MVCC - **Multi Version Concurrency Control**)
  - Unterschiedliche Operationen auf den gleichen Daten beeinflussen sich nicht (keine *Concurrency*-Konflikte, keine *Deadlocks*, keine *RaceConditions*)

MapReduce

Idee neu angewandt und mit komfortablem Framework vorstellt:

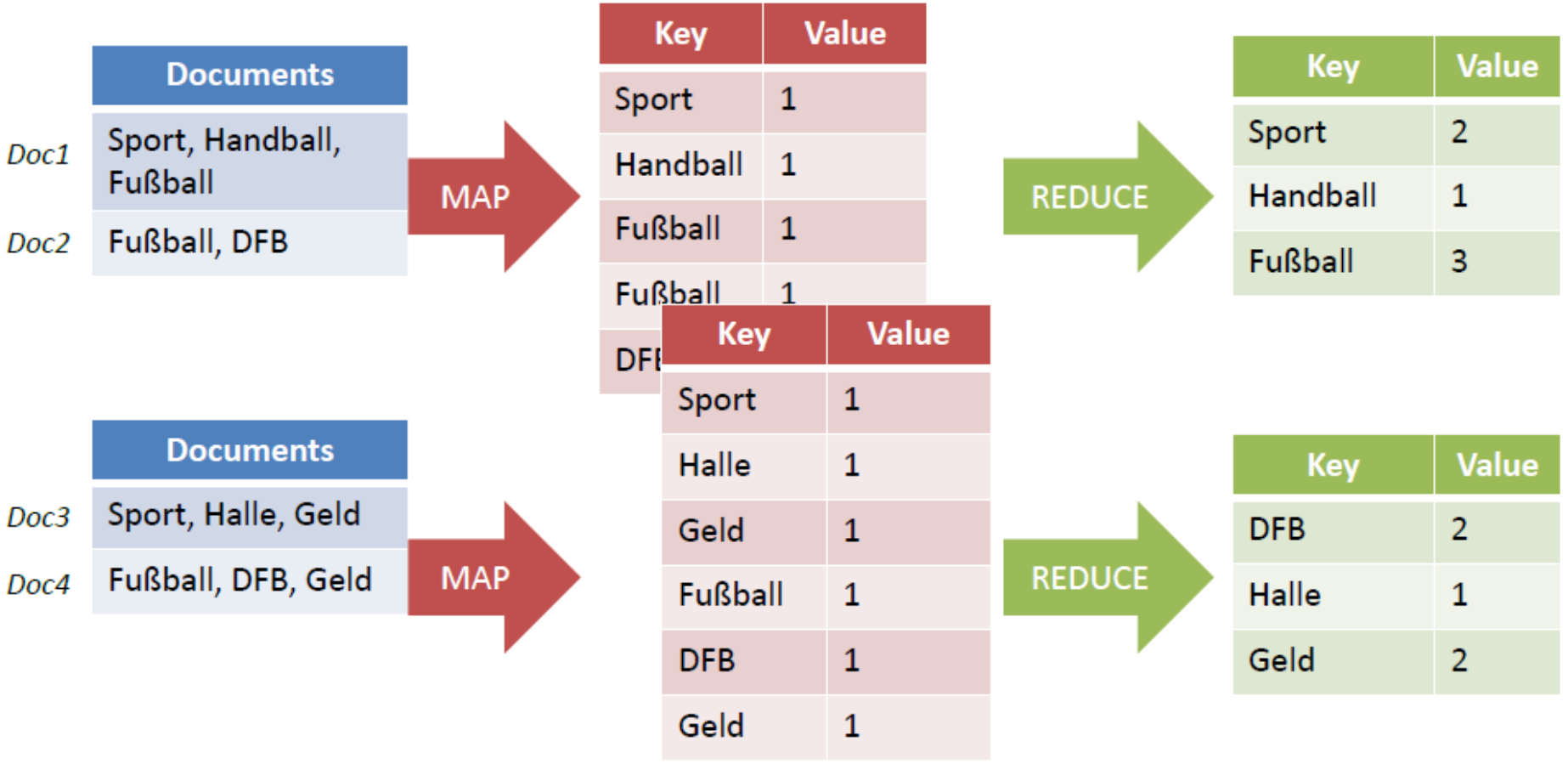
J. Dean and S.Gehmawat. *MapReduce: Simplified Data Processing on Large Clusters.* OSDI'04. 2004 (http://labs.google.com/papers/mapreduce.html)

Map Reduce

- **map** und **reduce** Funktionen: Konzept aus der funktionalen Programmierung parallele Verarbeitung großer Datenmengen
- MapReduce: framework zur verteilten Verarbeitung großer Datenmengen (freie Implementierung: Hadoop)
  - **map** verarbeitet Dokumente, erzeugt Schlüssel-Wert-Paare
  - optionales **reduce** erzeugt aggregierte (Zwischen)Werte, verarbeitet Ergebnisse von map oder rekursiv Zwischenergebnisse von reduce
  - **group**: anwenden auf Objekte mit gleichem Schlüssel, Beispiel: nicht alle Blogposts zählen, sondern Blogposts pro Tag
- Map-Reduce-Funktionen gespeichert in Dokumenten (Designdokumente)

MapReduce: Grundprinzip & WordCount Bsp.

- Entwickler muss zwei primäre Methoden implementieren
  - **Map**: (key1, val1) → [(key2, val2)]
  - **Reduce**: (key2, [val2]) → [(key3, val3)]



[Quelle: Störl - NoSQL-Datenbanksysteme]

Beispiel MapReduce in CouchDB

View ohne `reduce`

Key	Grouping: none	Value
[2010, 4, 12] ID: 17cf8a2934231a6cedc9e30fd3005311		1
[2010, 3, 31] ID: 17cf8a2934231a6cedc9e30fd30045a7		1
[2010, 3, 13] ID: 17cf8a2934231a6cedc9e30fd30041a8		1

View mit `reduce`

Key	Grouping: none	Value
null		3

View mit `reduce` und `group_level=2`

Key	Grouping: level 2	Value
[2010, 4]		1
[2010, 3]		2

Performance

Was „kostet“ Scale out?

- Flexibilität führt zu Storage-Overhead

  HBase: Faktor 2.1 ohne Replikation – Faktor 6.3 mit 3 Replikaten
  (Quelle: Schindler, NoSQL I/O, VDLB2012)

- Verteilungsarchitektur impliziert per se Overhead
- Außerdem: NoSQL-Systeme häufig in Release 0.* oder 1.*

  → noch viel Optimierungspotential

- Benchmarks für NoSQL-Systeme?

  Was ist ein „typisches“ NoSQL-Szenario ? Facebook? Log-Analyse? ?

  Metriken: → Bisher kaum generalisierende Benchmarks

  - Verhalten bei wachsender Datenmenge
  - Verhalten bei veränderter Server-Anzahl
    (dynamisches Hinzufügen und Entfernen)

Beobachtung in der Praxis:

- Die Daten- und Knotenmenge muss groß genug sein,
  damit ein wirklicher Performance-Vorteil zum Tragen kommt

NoSQL-Systeme: Zusammenfassung

Vorteile	Nachteile
• Flexible und kostengünstige horizontale Skalierung (scale out) • Verarbeitung riesiger Datenmengen mit kostengünstiger Software • Hochgradig parallelisierbare Anfrageverarbeitung mit MapReduce • Schemaflexibilität (falls benötigt)	• I.d.R. Abstriche bei Konsistenz • Erhöhter Aufwand für Entwicklung – Proprietäre, wenig mächtige APIs / „Anfragesprachen“ – OR-Mapper bislang nur rudimentär unterstützt • Bisher kaum Tools für Performance-Analyse und DB-Administration

Wichtige offene Punkte :

- Keine Standards / Standardisierte Anfragesprache(n) und APIs
- Keine Benchmarks
- Communities statt kommerzielle Software-Entwickler

«Man denkt bei NoSQL besser an eine soziale Bewegung als an eine Technologie.»
Fowler

". . . era of "one-size-fits-all database" seems to be over. Instead of squeezing all your data into tables, we believe the future is about choosing a data store that best matches your data set and operational requirements. It's a future of heterogeneous data backends, polyglot persistence and choosing Not Only SQL but sometimes also a document database, a key-value store or a graph database."

[Ankündigung no:sql(eu) conference, April 2010]

Oliver Bussmann, CIO von SAP:

"Relationale Datenbanken spielen in zehn Jahren im Enterprise-Umfeld keine große Rolle mehr."

[Bussmann im CIO-Jahrbuch 2012]

Argumente gegen relationale DBMS?

- „Striktes“ Datenbankschema

  - aufwändig zu entwerfen: Normalisierung,. . .
  - nachträglich schwierig zu ändern

- Performance-Probleme

  → Scale up, DBMS-Update
  → Änderung im Datenmodell
  → Anfrageformulierung und -optimierung bei JOIN, SELECT * …
  → Tuning, Nutzung Index …

- „Impedence Mismatch“

  → OR-Mapper effektiv nutzen (z.B. Hibernate)

- Fehlende Mechanismen zur Verwaltung unstrukturierter Informationen

  → Objektrelationale Erweiterungen (UDT und Methoden)
  → Integration neuer Funktionen (Information Retrival, FileTable …)

Anforderungen an eine klassische (relationale) Datenbank

- Die Daten sollen viele Generationen von Hardware, Programmen und Programmiersprachen überleben.
- Verschiedenste Programme in verschiedenen Programmiersprachen sollen auf verschiedene Ansichten desselben Datenstamms zugreifen können.
- Es gibt keine ausgezeichnete Hierarchie des Zugriffs, sondern alle logischen Abfragen sollen gleichermaßen möglich sein.
- Der Aufbau der Datenbank sowie die Manipulation der Daten soll nach vereinheitlichten Standards erfolgen.

Koexistenz von SQL- und NoSQL-DBMS?

Muss es immer eine entweder-oder-Auswahl sein?

→ Auswahl des geeigneten Systems für die jeweilige Aufgabe statt *„One Size Fits All“*?

Beispiel: Amadeus Log Service

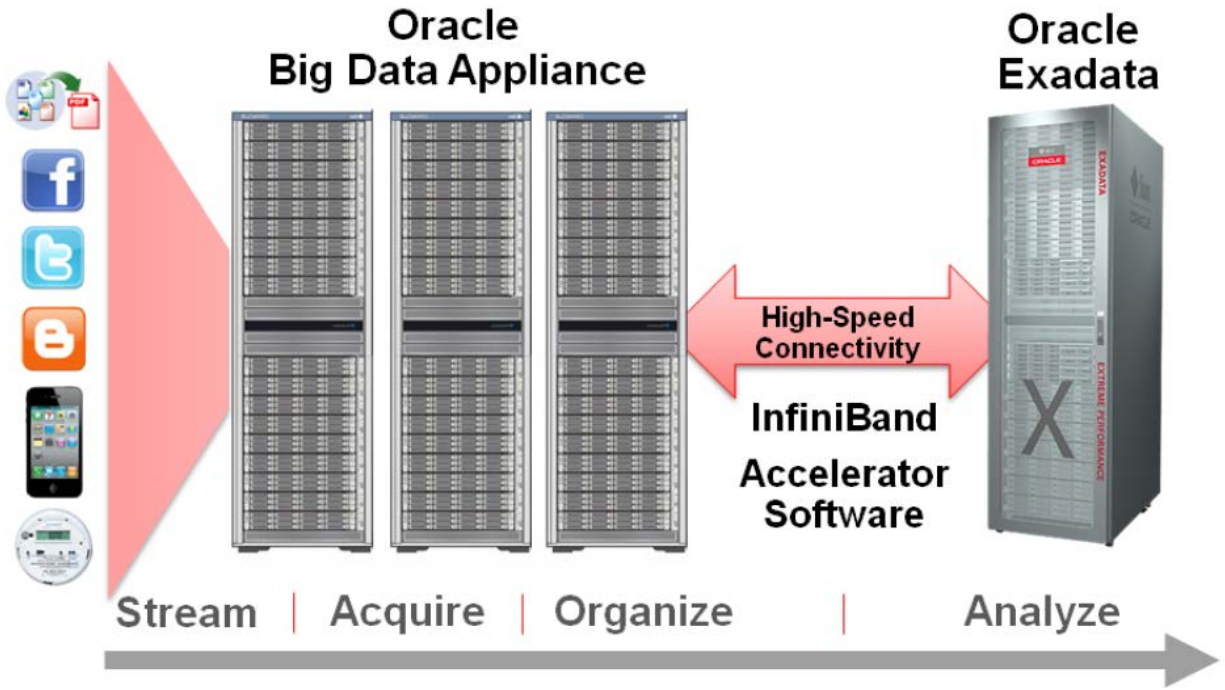
Wöchentlich mehrere hundert Terabyte Log-Daten von verschiedenen Servern einer SOA-Architektur

Architektur:

- Verteiltes Dateisystem (HDFS) für komprimierte Log-Daten
- NoSQL-System (HBase) zur Indexierung nach Timestamp und SessionID
- Full Text Search Engine (SOLR) für Volltextsuche
- MapReduce-Framework (Hadoop) zur Analyse (Nutzerstatistiken und Fehleranalyse)
- Relationales DBMS (Oracle) für Meta-Daten (Benutzer-Infos etc.)

[Quelle: Kossmann: http://wp.sigmod.org/?p=559]

Anwendungsbeispiel für Oracle Big Data Appliance und Exadata



[Quelle: Oracle]

Trend: **MapReduce (Hadoop) Integration in relationale DBMS und Data Warehouse Systeme**

Entwicklungen:

- Oracle BigData-Appliance
- Oracle NoSQL 2.0 (Key-Value-Store)
- IBM Infosphere mit Hadoop-Support
- Microsoft SQL Server 2012/2014 mit Hadoop-Support
- Integration von Hadoop (Cloudera-Distribution) in SAPs BigData-Angebot
  (SAP HANA, SAP Sybase IQ, SAP Data Integrator, SAP Business Objects)

NoSQL - Revolution oder Evolution?

„… Relationale Datenbanken werden in zehn Jahren vielleicht anders aussehen, verschwinden werden sie aber nicht."

Andreas Bitterer, Gartner-Analyst

→ „Kleine" Revolution die zur Evolution der etablierten relationalen Systeme führt

- Integration von MapReduce-Ansätzen in relationale DBMS insbesondere DataWarehouse-Systeme
- (Noch) stärkere Entwicklung hin zu nicht-sperrenden Concurrency Control Verfahren, um scale out zu ermöglichen
- Mittelfristig: Integration von „konfigurierbarer" Konsistenz in relationalen DBMS

→ Nutzung von NoSQL-DBMS als „Applikationsdatenbank" – nicht als „Integrationsdatenbank" ⇨ Koexistenz

NoSQL-Systeme vs. Relationale DBS: Zusammenfassung

Entscheidung für oder gegen ein System:

- Datenanalyse
  - Wie groß ist die erwartete Datenmenge?
  - Komplexität der Daten und Schemaflexibilität?
  - Art der Navigation zwischen den Daten?
- Konsistenzanforderungen der Anwendungen?
- Anfrageanforderungen der Anwendungen?
- Performanceanforderungen (Latenz, Skalierbarkeit, Concurrency)?
- Nicht-funktionale Anforderungen (Lizenz, Firmenpolitik, Sicherheit, Dokumentation etc.)
- Kosten (inkl. Entwicklung und Administration!)
- Standardisierung!

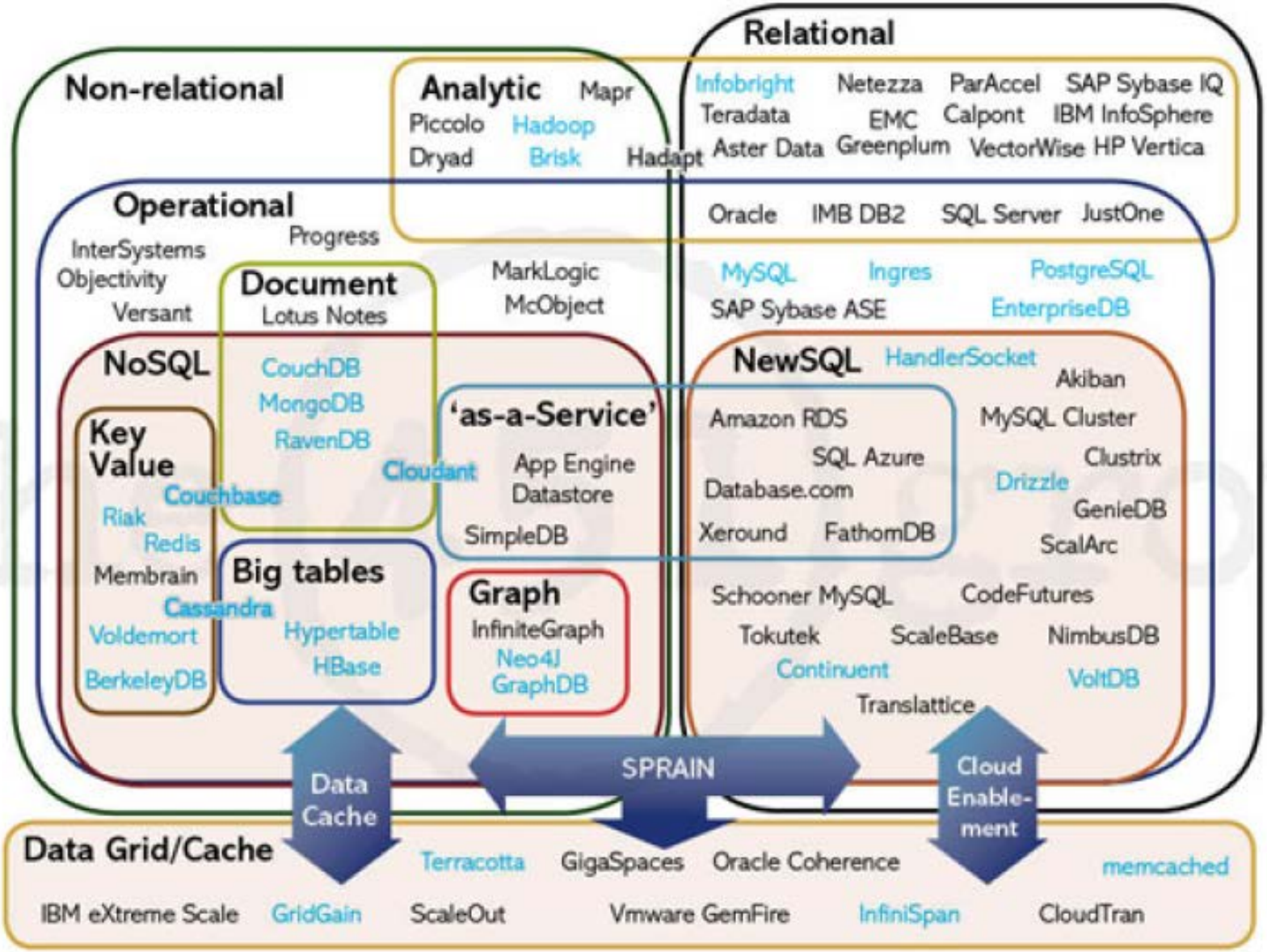
DB-Engines Ranking 6/2017

330 Systeme im Ranking, Juni 2017

Rang Jun 2017	Rang Mai 2017	Rang Jun 2016	DBMS	Datenbankmodell	Punkte Jun 2017	Punkte Mai 2017	Punkte Jun 2016
1.	1.	1.	Oracle	Relational DBMS	1351,76	-2,55	-97,49
2.	2.	2.	MySQL	Relational DBMS	1345,31	+5,28	-24,83
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1198,97	-14,84	+33,16
4.	4.	↑ 5.	PostgreSQL	Relational DBMS	368,54	+2,63	+61,94
5.	5.	↓ 4.	MongoDB	Document Store	335,00	+3,42	+20,38
6.	6.	6.	DB2	Relational DBMS	187,50	-1,34	-1,07
7.	7.	↑ 8.	Microsoft Access	Relational DBMS	126,55	-3,33	+0,32
8.	8.	↓ 7.	Cassandra	Wide Column Store	124,12	+1,01	-7,00
9.	9.	↑ 10.	Redis	Key-Value Store	118,89	+1,44	+14,39
10.	10.	↓ 9.	SQLite	Relational DBMS	116,71	+0,64	+9,92
11.	11.	11.	Elasticsearch	Suchmaschine	111,56	+2,74	+24,14
12.	12.	12.	Teradata	Relational DBMS	77,33	+1,00	+3,49
13.	13.	13.	SAP Adaptive Server	Relational DBMS	67,52	-0,23	-4,16
14.	14.	14.	Solr	Suchmaschine	63,61	-0,16	-0,46
15.	15.	15.	HBase	Wide Column Store	61,87	+2,37	+8,88
16.	16.	↑ 18.	Splunk	Suchmaschine	57,52	+0,82	+12,29
17.	17.	↓ 16.	FileMaker	Relational DBMS	57,08	+0,60	+8,39
18.	18.	↑ 20.	MariaDB	Relational DBMS	52,89	+1,91	+18,24
19.	19.	19.	SAP HANA	Relational DBMS	47,50	-1,55	+6,23
20.	20.	↓ 17.	Hive	Relational DBMS	44,38	+0,91	-2,62
21.	21.	21.	Neo4j	Graph DBMS	37,87	+1,73	+3,84
22.	22.	↑ 25.	Amazon DynamoDB	Document Store	34,02	+0,82	+9,68

[Quelle: http://db-engines.com/de/ranking]

Klassifizierung von Datenbanksystemen



[Quelle: Matthew Aslett 451 Group]

NewSQL-DBMS

Idee: → Das Beste aus beiden Welten vereinen

- SQL
- ACID
- Nicht-sperrende Concurrency Control
- Hohe per-node Performance
- Scale out, shared nothing Architektur

Erweiterungen bestehender Systeme:

- MySQL Cluster
- PostgreSQL: nativer JSON-Support und hstore (key-value) Datentyp

Entwicklung komplett neuer Systeme:

- VoltDB (Michael Stonebraker)
- Drizzle
- …