



Masterstudiengang Wirtschaftsinformatik Big Data Engineering

FH Münster
Master Wirtschaftsinformatik
Wintersemester 2016
Dozent: Lars George



Einheit 1

- Übersicht
- Was bedeutet Big Data?
- Einführung in Hadoop
- **Hauptziel:** HDFS Kenntnisse erlangen
- **Übung 1:**
 - Hadoop installieren
 - Dateien in HDFS schreiben und lesen
 - Funktionstest (Block Fehler emulieren)



Einheit 1

- **Übersicht**
- Was bedeutet Big Data?
- Einführung in Hadoop
- Einführung in HDFS

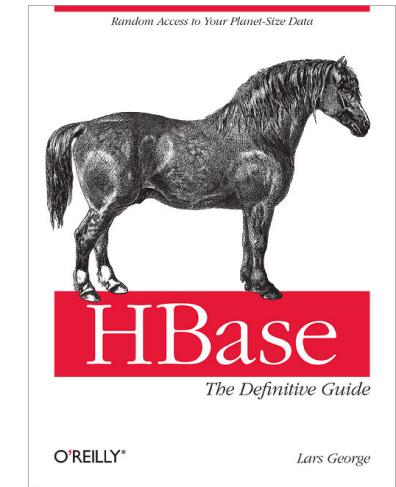


Übersicht

- Vorstellungsrunde
- Logistik
- Themen und Inhalte
- Prüfungsform

Über Mich

- OpenCore Partner & Cofounder
 - Vorher EMEA Chief Architect @ Cloudera (4+ Jahre)
 - Beratungsdienstleistungen und Lösungsentwicklung europaweit rund um Big Data
- Apache Committer & Member
 - HBase (und PMC)
- O'Reilly Autor
 - HBase – The Definitive Guide
 - Ein weiteres...
- Kontakt
 - lars@larsgeorge.com
 - [@larsgeorge](https://twitter.com/larsgeorge)





Frage an Studenten

- Welche Erwartungen sind vorhanden?
- Welche Vorkenntnisse sind vorhanden?
- Welche Fragen sind vorhanden?



Logistik

- Wann, wie oft, was?



Themen und Inhalte

Dieser Kurs vermittelt alle nötigen **Grundkenntnisse** für die Sammlung, Speicherung und Verarbeitung von „großen“ Daten.

Größe ist dabei nur ein Merkmal, andere sind **Komplexität** oder **Rate**.

Verteilte Systeme werden anhand des **Hadoop** Ökosystems erklärt und **praktisch** umgesetzt.



Prüfungsform

Der Kurs **vermittelt** in den Vorlesungen die **nötigen** Konzepte, welche in Übungen **praktisch** umgesetzt werden.

Die abschließende **Prüfung** setzt sich aus **80% Projektarbeit** und **20% mündlicher Prüfung** zusammen.

Wer die Übungen mitmacht sollte wenig Probleme bei der Arbeit haben. Die mündliche Prüfung ist ein **Vortrag** der gewählten Lösung.



Einheit 1

- Übersicht
- **Was bedeutet Big Data?**
- Einführung in Hadoop
- Einführung in HDFS



Technology disruption will continue to come.....

85%



plan to use **cloud** tools in
the next 3 years

79%



experienced **security** incidents in
past year that caused financial
and/or reputational impact

26 Billion



of devices managed by
Internet of things (by 2020, MIT)

35ZB



Annual **data** generation by 2020
(35 trillion terabytes)

**Creating new demands
on IT customers...
and IT providers**

90%



% of world's **data** created
in last 2 years (Microsoft)

63%



User-owned **mobile devices**
used for work

78%



say **digital transformation**
will become critical in 2 years
(MIT)

79%



Use a smart phone
when **shopping**



Was ist Big Data?

McKinsey sagt:

Box 1. What do we mean by "big data"?

"Big data" refers to datasets whose size is beyond the ability of typical database software tools to capture, store, manage, and analyze. This definition is intentionally subjective and incorporates a moving definition of how big a dataset needs to be in order to be considered big data—i.e., we don't define big data in terms of being larger than a certain number of terabytes (thousands of gigabytes). We assume that, as technology advances over time, the size of datasets that qualify as big data will also increase. Also note that the definition can vary by sector, depending on what kinds of software tools are commonly available and what sizes of datasets are common in a particular industry. With those caveats, big data in many sectors today will range from a few dozen terabytes to multiple petabytes (thousands of terabytes).



Was ist Big Data?

Big Data ist nicht alleine bestimmt von der Größe der Datenmenge, sondern auch dessen Format und Geschwindigkeit. Diese werden oft als die drei V's des Big Data beschrieben:

Volume	Die Datenmenge in Bytes
Variety	Die möglichen Formate und Schemata
Velocity	Die Geschwindigkeit mehr der Daten auftreten/erzeugt werden



Volume

Kurze Wiederholung:

Ein **Megabyte** ist 10^6 oder 1.000.000 Bytes oder 1000 Kilobytes oder ungefähr 6 Sekunden Audiosignal in unkomprimierter CD-Qualität.

Ein **Gigabyte** ist 10^9 oder 1.000.000.000 Bytes oder 1000 Megabytes. Eine DVD fast circa 4.7GB an Informationen.



Volume

Ein **Terabyte** 10^{12} oder 1.000.000.000.000 Bytes oder 1000 Gigabytes. Im April 2011, fasste die US Library of Congress digitale Data von circa 235 Terabytes; jeden Monat kommen ungefähr 5TB dazu.

Ein **Petabyte** ist 10^{15} oder 1000 Terabytes oder 1 Million Gigabytes. Google verarbeitet rund 24PB an Daten jeden Tag und seit August 2011 existiert ein von IBM gebautes 120PB Speichersystem (dem größten aller Zeiten).



Volume

Ein **Exabyte** ist 10^{18} oder 1 Milliarde Gigabytes oder 1000 Petabytes. Nach Angaben von Cisco im Juni 2009 Visual Networking Index, erreicht der weltweite IP Verkehr 667 Exabytes in 2013, mit 18 Exabytes nur für Internet Video pro Monat.

Ein **Zettabyte** ist 10^{21} oder 1.000.000.000.000.000.000 Bytes, 1 Milliarde Gigabytes oder 1 Billiarde Megabytes. In 2003 berechnete Mark Liberman, ein Linguist der University of Pennsylvania, dass die Menge aller jemals gesprochener menschlicher Sprache 42 Zettabytes beträgt.



Was ist Big Data?

Nicht viele Menschen haben mit solchen **großen Datenmengen** zu tun. Deshalb ist man aber **nicht** unbedingt von den Möglichkeiten der Big Data Lösungen **befreit**. Wenn Datenmenge nicht ein Problem ist („Ich habe nur 900GB an Daten!“) dann kann immer noch die **Natur** der Daten oder dessen **Generierungsrate** ein **wichtiger** Entscheidungsfaktor sein.



Variety

Über die letzten Jahre sind immer **mehr**, nicht verbundene Datensysteme entstanden, deren **Inhalte** aber übergreifend eine **große** Rolle spielen, zum Beispiel soziale Netzwerke (Facebook, LinkedIn, Instagram, Twitter). Oder auch Informationsdienste welche nun ihre **Daten** zum Verkauf **anbieten** (Wetterdienst, Börsendaten, Geldwechselraten).



Variety

Variety bedeutet für uns: **Nicht mein Schema!**

Die Anzahl **externer** Datenquellen steigt **zunehmend** und diese Daten mit den eigenen zu **verbinden** macht sehr viel Sinn.

Frage: „Unser mit Sensoren ausgestatteten Schiffsmotoren sollen möglichst kosteneffizient gewartet werden.“



Variety

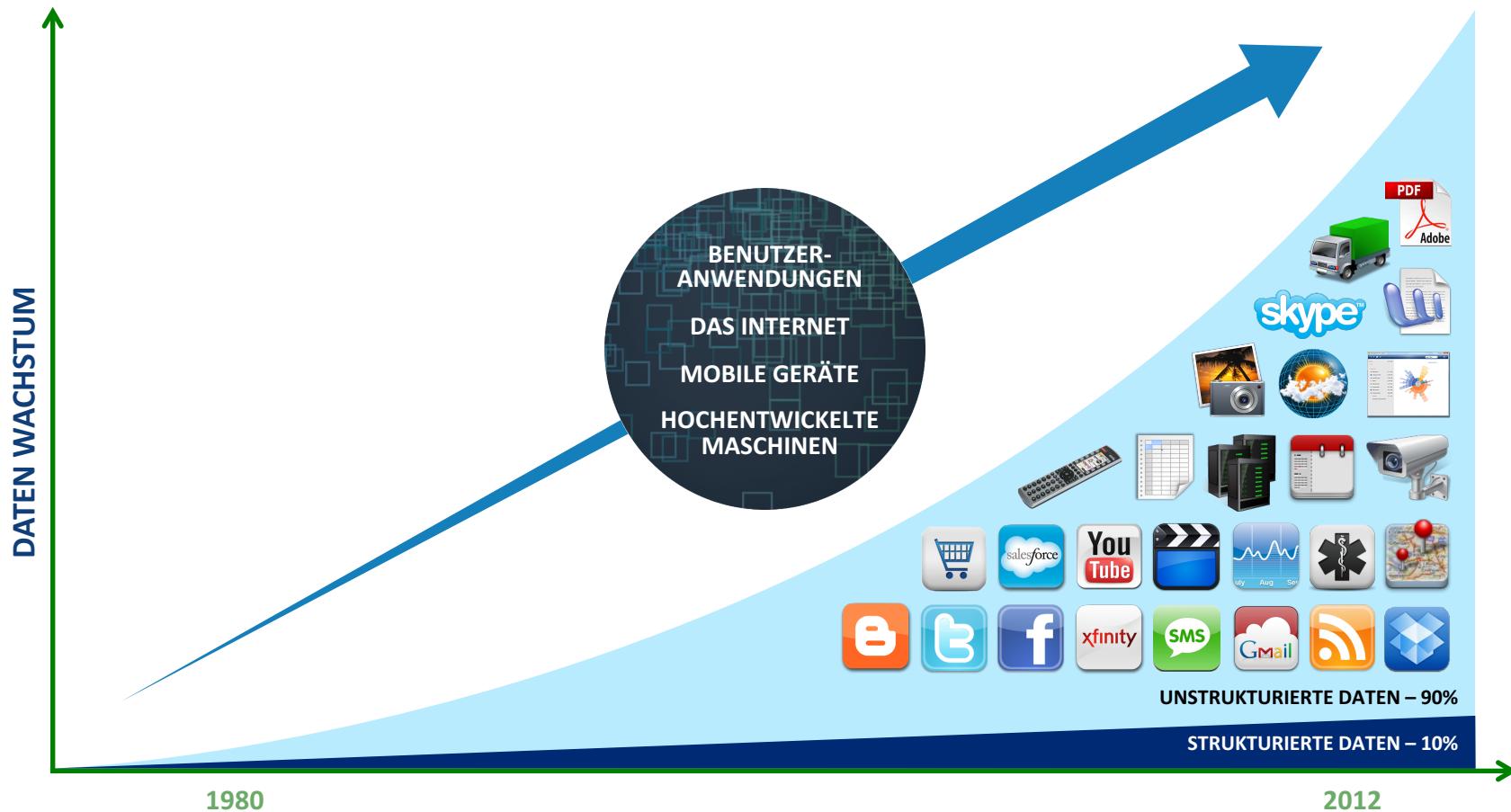
Dazu können wir Umweltdaten mit den Sensorendaten **verbinden**. Diese liegen aber in einem **anderen** Format vor.

Sollen diese **umgewandelt** (das sogenannte ETL) oder direkt **eingebunden** werden? Kann dies aber zeitnah und flexibel passieren? Was passiert wenn noch 10 oder 100 weitere Quellen dazukommen?



Variety & Volume

Wandel der Daten in den letzten 30 Jahren:





Velocity

Das dritte V steht für Velocity und beschreibt wie **schnell** wir heute Daten generieren.

Beispiel: Twitter im August 2013



New Tweets per second (TPS) record:
143,199 TPS. Typical day: more than 500
million Tweets sent; average 5,700 TPS.



Velocity

Beispiel: Facebook in 2013

Benutzerstatistik:

- Anzahl Benutzer: 1,26 Milliarden
- Täglich aktive Nutzer: 699 Millionen
 - USA: 128 Millionen
 - UK: 24 Millionen
- Nutzer in China: 87 Millionen
- Meistbenutzung: Kanada



Velocity

Beispiel: Facebook in 2013

Aktivitätsstatistik:

- Verbindungen zwischen Freunden: 150 Milliarden
- Tägliche Likes: 4,5 Milliarden
- Fotos gesamt/täglich: 250 Milliarden/350 Millionen
- Nachrichten pro Tag: 10 Milliarden



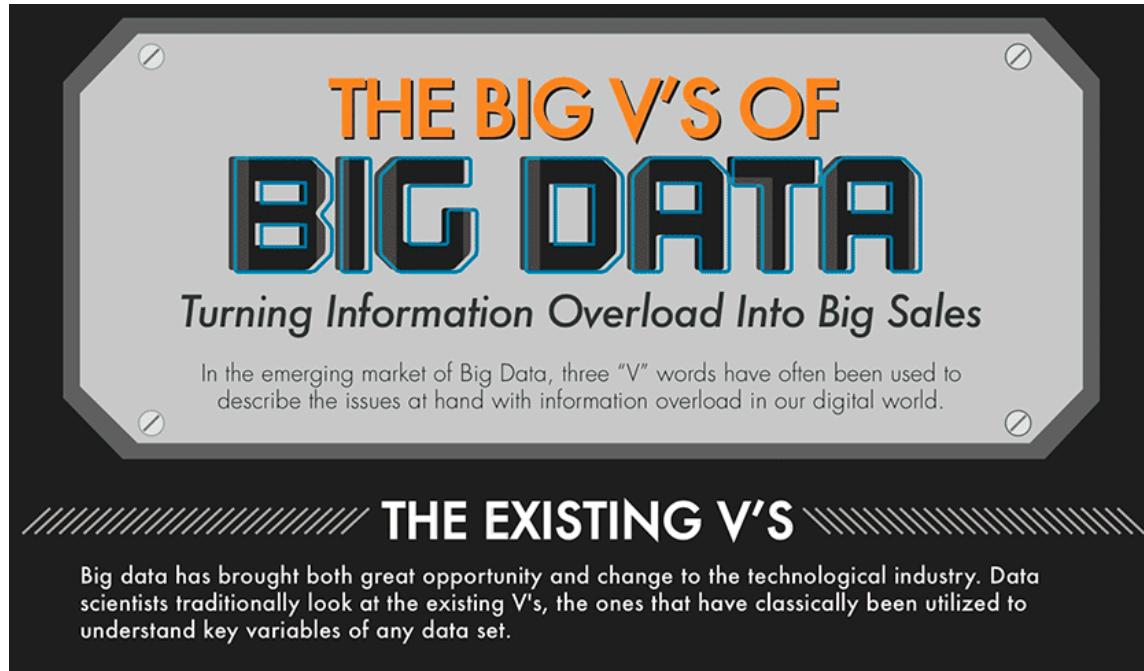
Velocity

Dieses sind offensichtlich **extreme** Beispiele.
Aber sie verdeutlichen doch wie sehr wir heute
„**Online**“ sind – und damit Daten **produzieren**!
Ein weiterer Trend ist das mit vielen Daten noch
mehr Daten **erzeugt** werden, zum Beispiel durch
die **Analyse**.

Auch im kleinen ist wiederum interessant sich
mit der Thematik zu beschäftigen, denn die
eingesetzte **Technik** ist relativ **universell**.

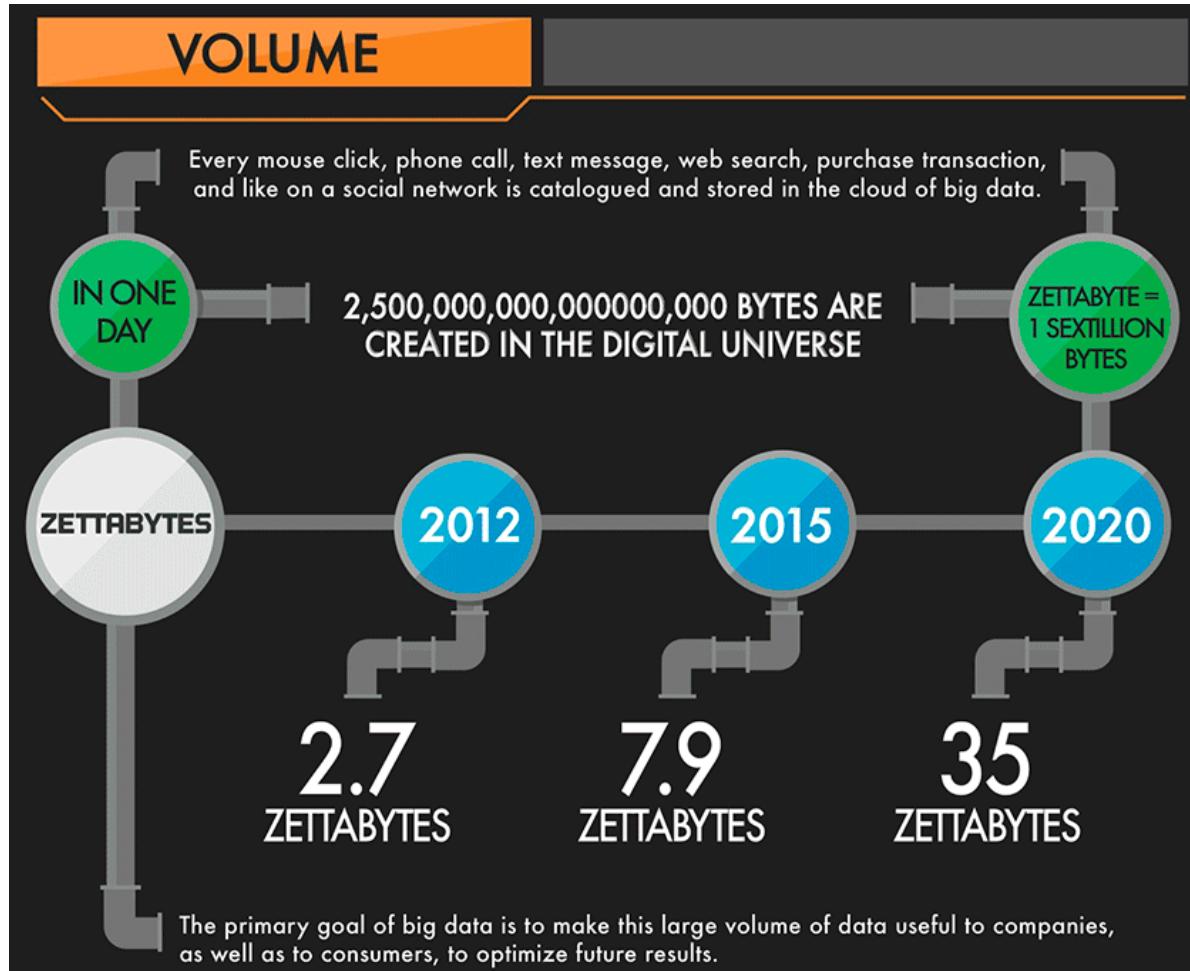
Die drei V's in Bildern

Es gibt viele Quellen welche Big Data beschreiben. Interessant sind Infographiken, z. B.:



Quellen: <http://whatsthebigdata.com/>, <http://visual.ly/big-vs-big-data>, <http://pros.com>,

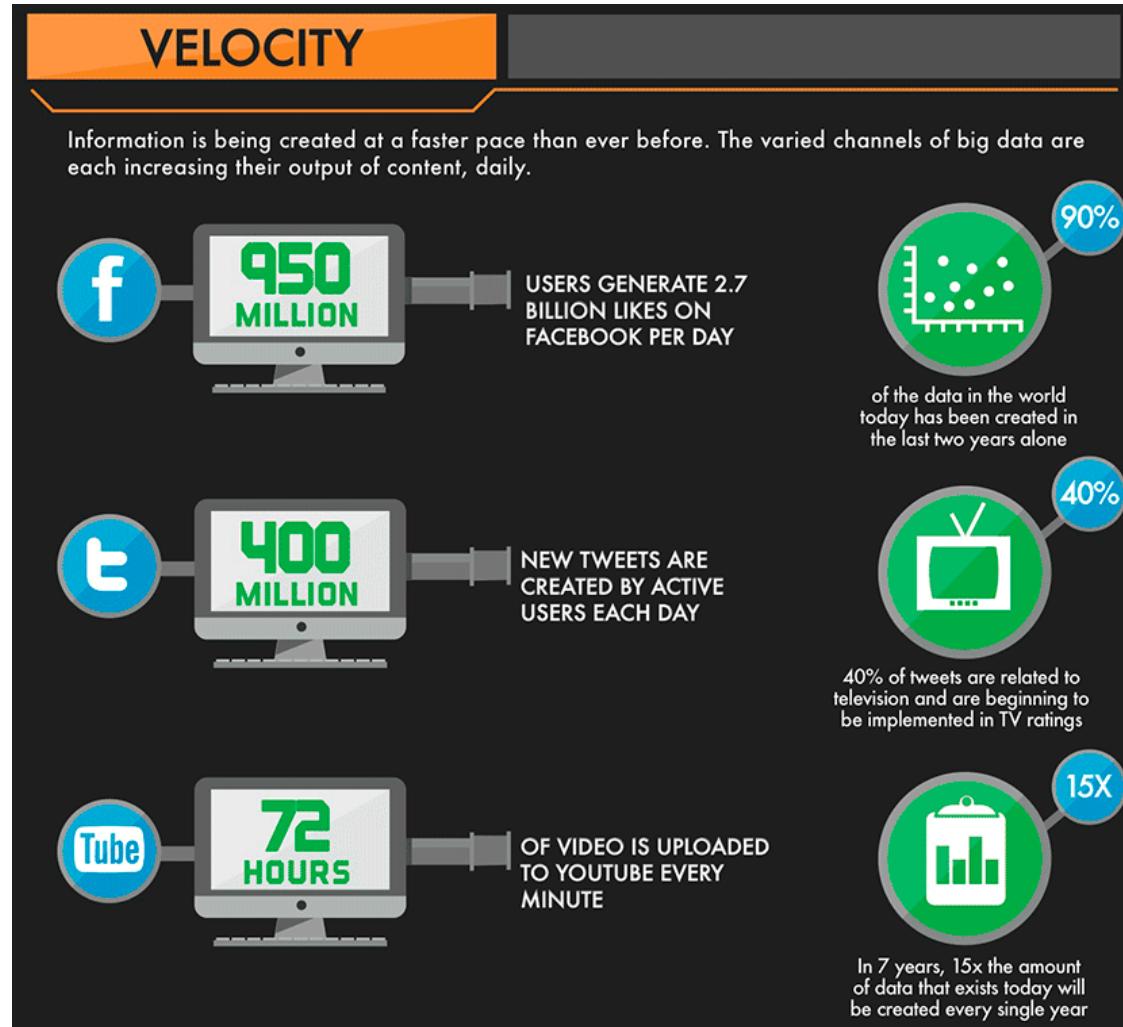
Die drei V's in Bildern



Die drei V's in Bildern



Die drei V's in Bildern



Die drei V's in Bildern



THE MISSING V

This new influx of data requires a re-examination and addition to the classic 3 V's concept.

VIABILITY

It is necessary to filter through this information and carefully select the attributes and factors that are most likely to predict outcomes that matter most to businesses. The secret to success is uncovering the latent, hidden relationships among these variables.

QUESTIONS TO CONSIDER

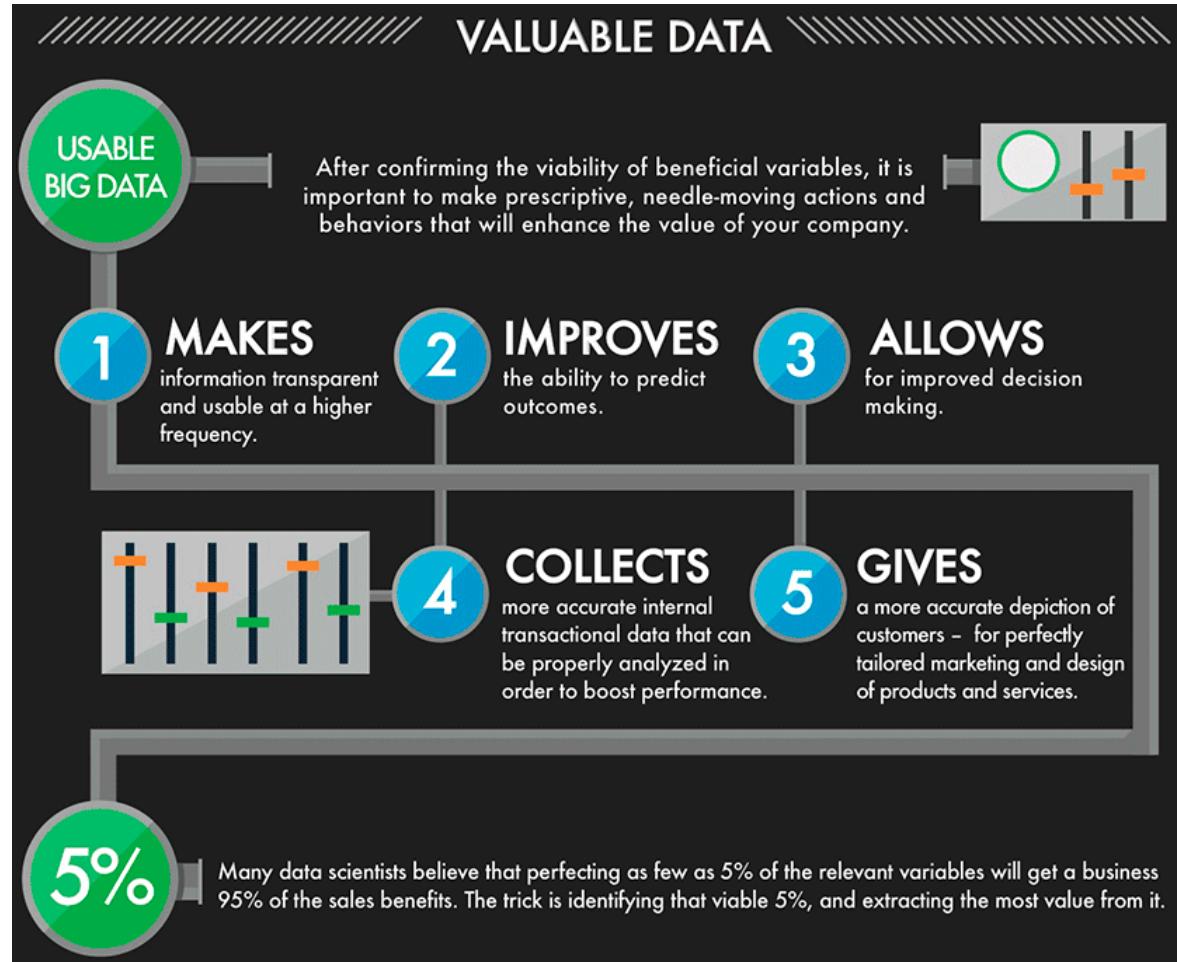
-  What effect does time of day or day of week have on buying behavior?
-  How do age, family size, credit limit, and vehicle type all converge to predict a consumer's propensity to buy?
-  How do geo-location, product availability, and purchasing history predict a consumer's propensity to buy?
-  Does a surge in Twitter or Facebook mentions presage an increase or decrease in consumer purchases?

THE NECESSARY STEPS

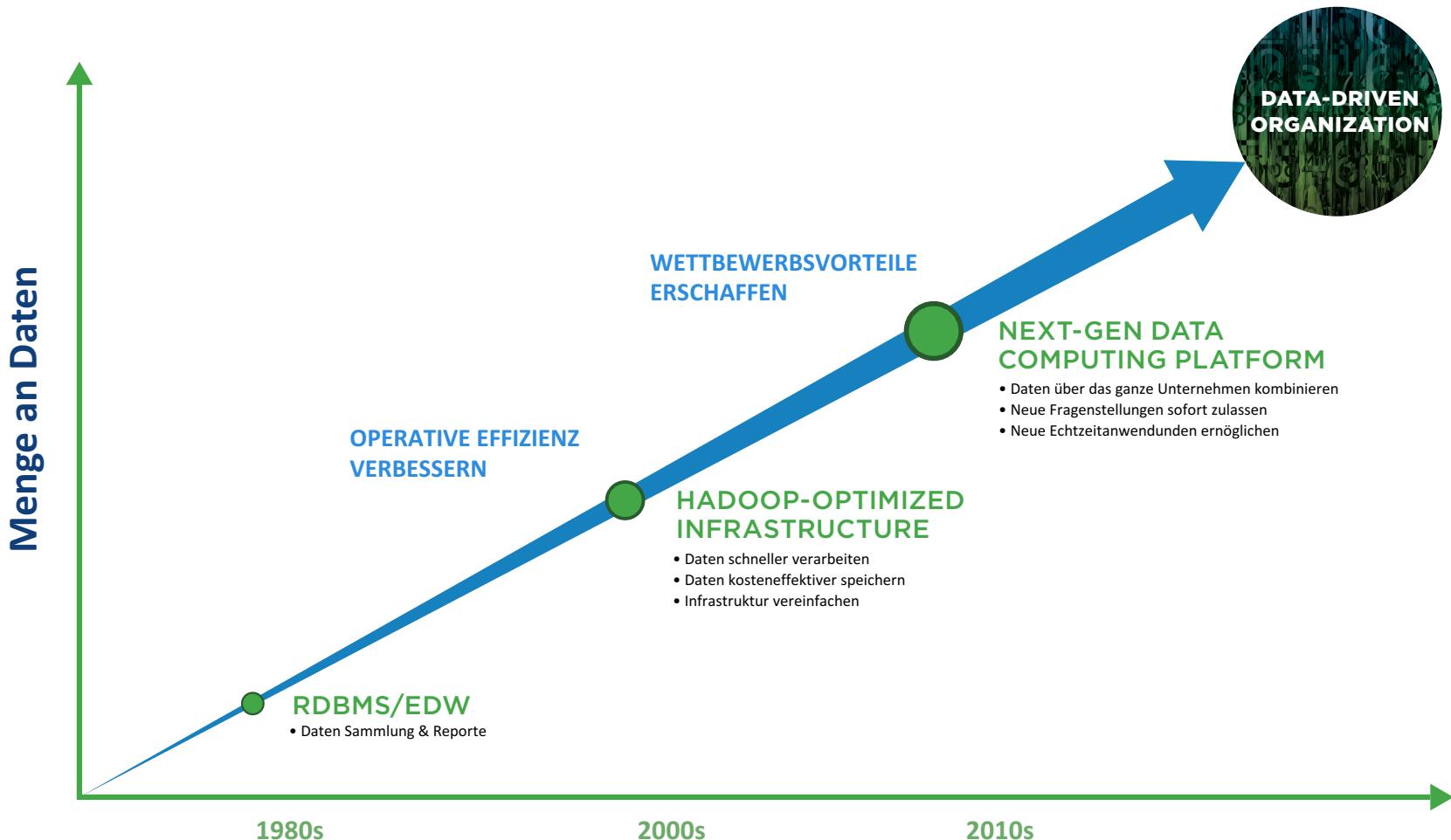
Test and Conclude

-  Establishing a method to assess the viability of information, regardless of field type or size of data.
-  Establishing a method that is quick and cost-effective.
-  Confirming a variable's relevance before investing in the creation of a fully formed model.

Die drei V's in Bildern



Evolution eines Unternehmens



Gründe für Big Data

Big Data wird primär durch die Hardware als Konsumware angetrieben. Es ist heute möglich für einen Bruchteil der Kosten einen Rechnerverbund (Cluster) in Petabyte Größe aufzubauen.

Beispiel:

- 2 x Quadcore, 96GB RAM, 12 x 2TB HDD \approx 5000€
- 1024TB / 24TB pro Server \approx 43
- Replikationsfaktor 3 * 43 = 129 = 645.000€



Ein Beispiel: Data Warehouse

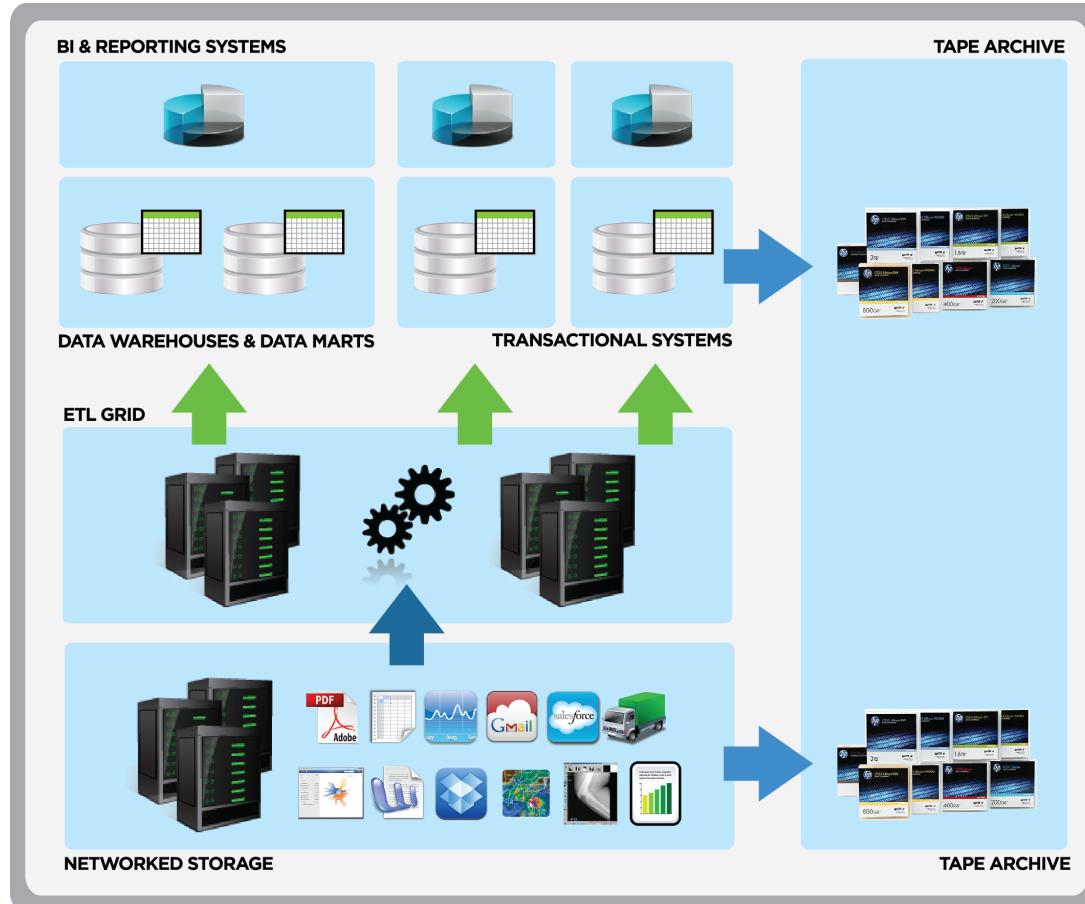
Im folgenden wird anhand des **traditionellen** Enterprise Data Warehouse (EDW) gezeigt in welchen Bereichen **Big Data Probleme** verursacht.

Dies ist nur ein **Beispiel**, ist aber **symptomatisch** für viele andere **IT-Systeme**.

Data Warehouse



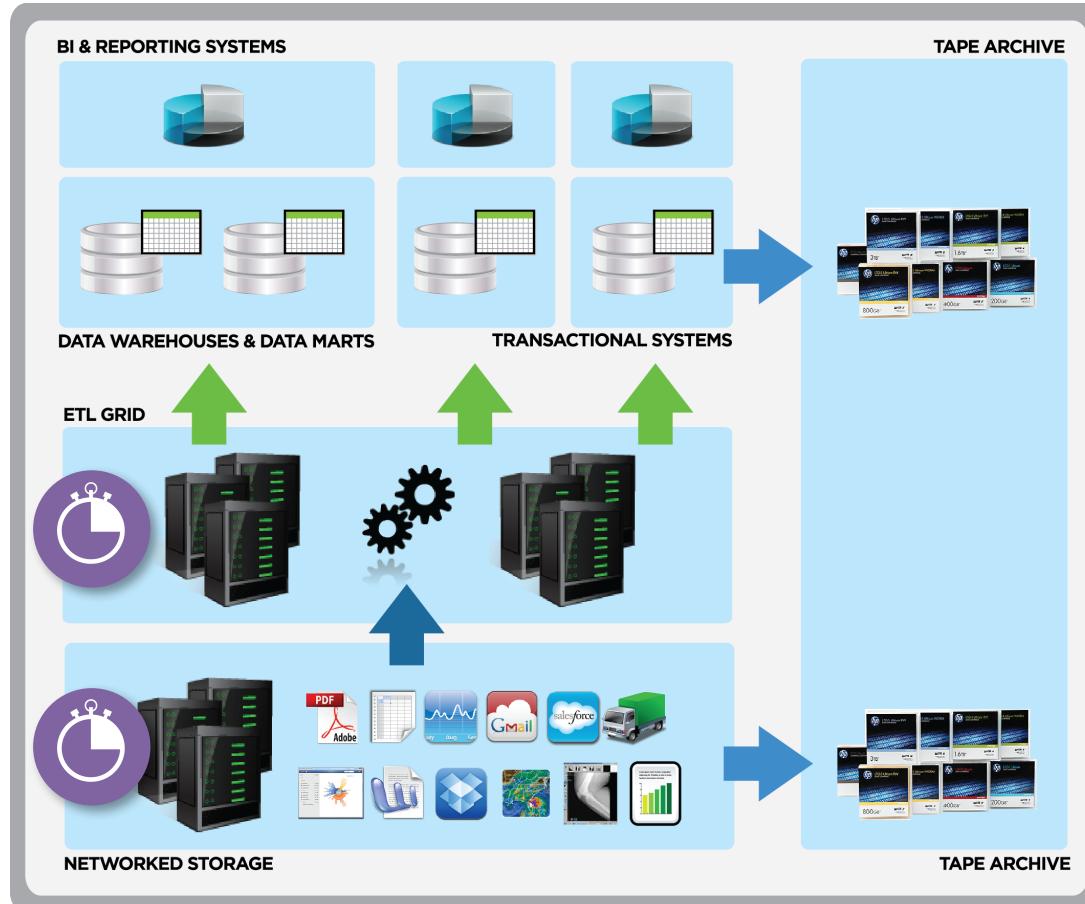
Datenmanagement Strategien haben sich nicht geändert.



- Rohdaten auf SAN, NAS oder Tape
- Daten werden aus dem Speichersystem zum Rechensystem bewegt
- Relationale Modelle mit vorgefertigten Schemata

Data Warehouse

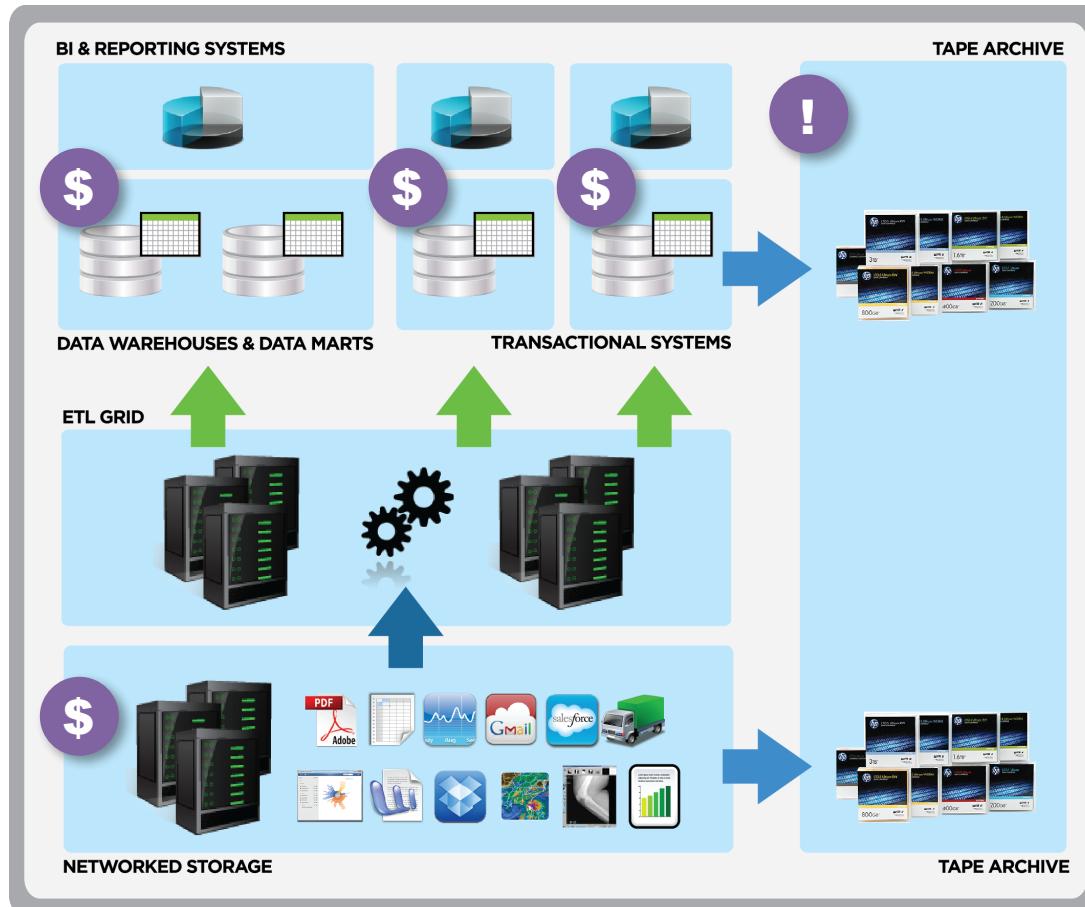
Zu viele Daten, zu viele Quellen.



- Daten können nicht schnell genug aufgenommen werden

Data Warehouse

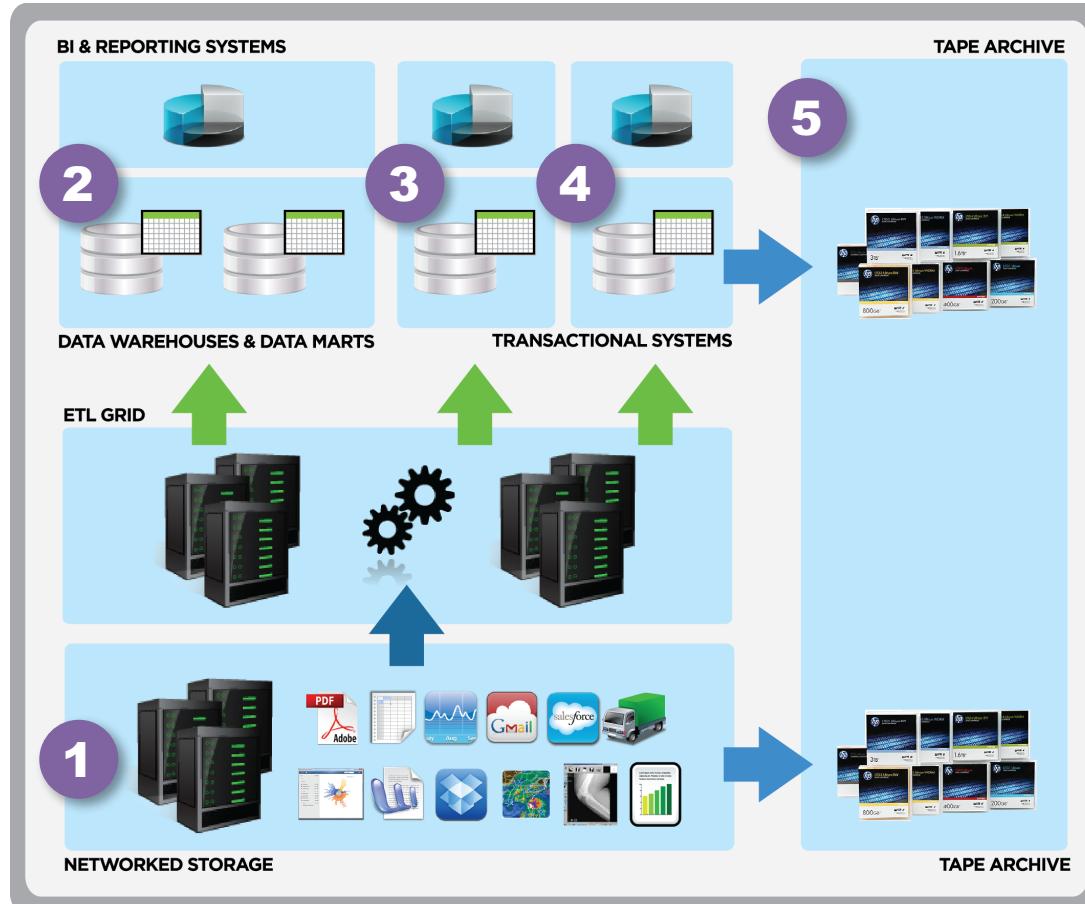
Zu viele Daten, zu viele Quellen.



- Daten können nicht schnell genug aufgenommen werden
- Kosten verbieten alle Daten zu speichern

Data Warehouse

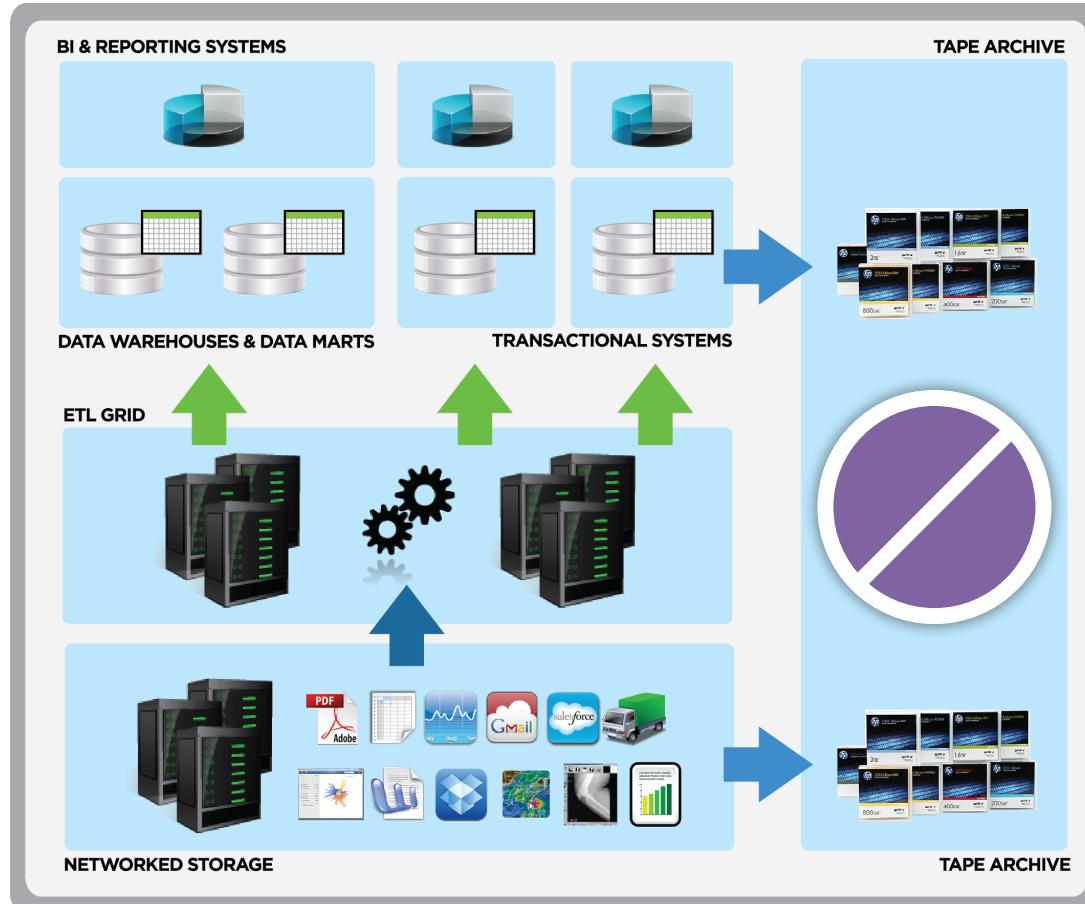
Zu viele Daten, zu viele Quellen.



- Daten können nicht schnell genug aufgenommen werden
- Kosten verbieten alle Daten zu speichern
- Daten existieren in mehreren Orten

Data Warehouse

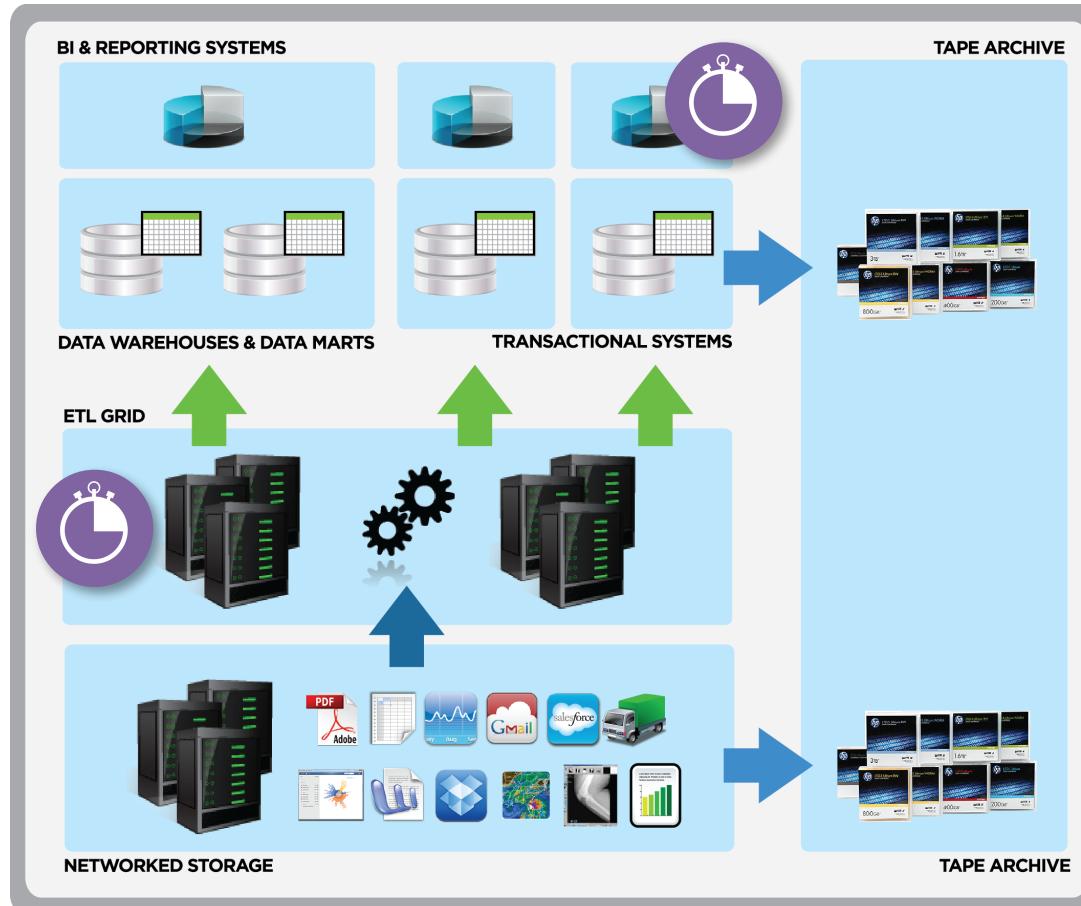
Zu viele Daten, zu viele Quellen.



- Daten können nicht schnell genug aufgenommen werden
- Kosten verbieten alle Daten zu speichern
- Daten existieren in mehreren Orten
- Archivierte Daten sind nicht zugänglich

Data Warehouse

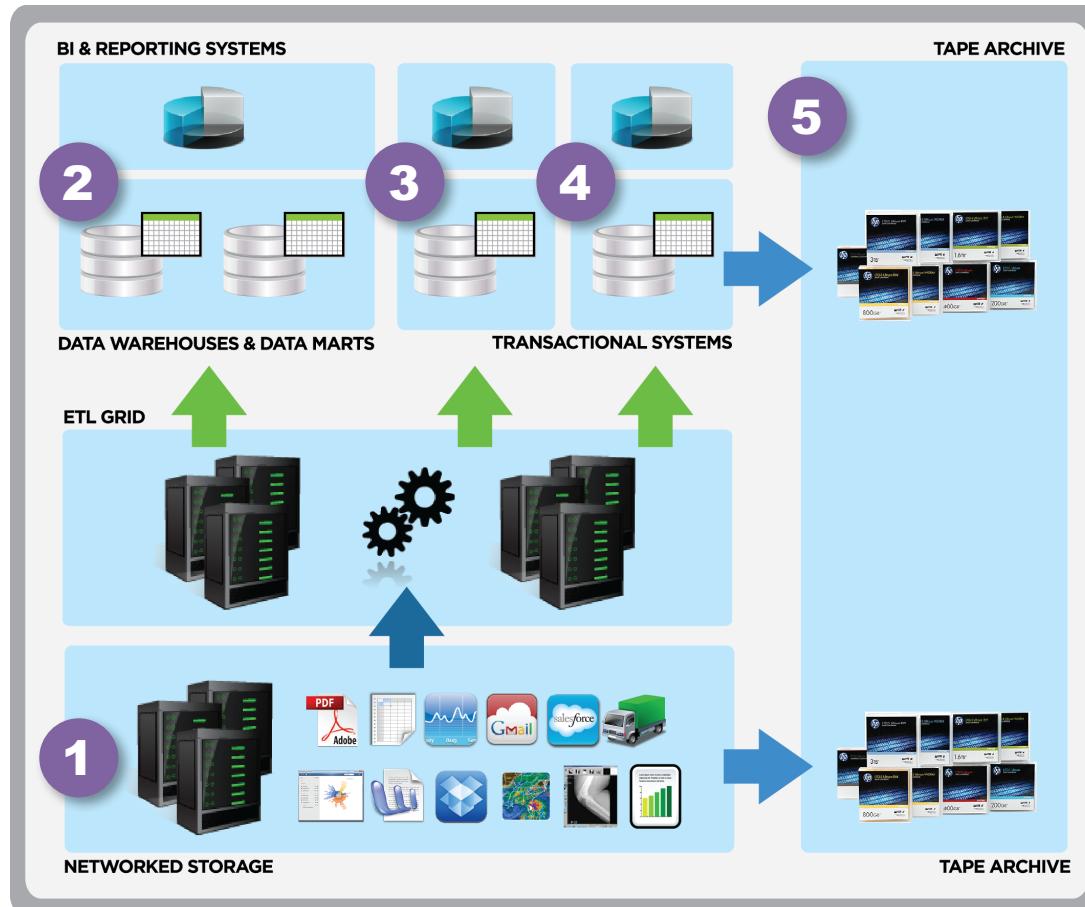
Daten können nicht wie benötigt verarbeitet werden



- Analyse und Verarbeitung dauert zu lange

Data Warehouse

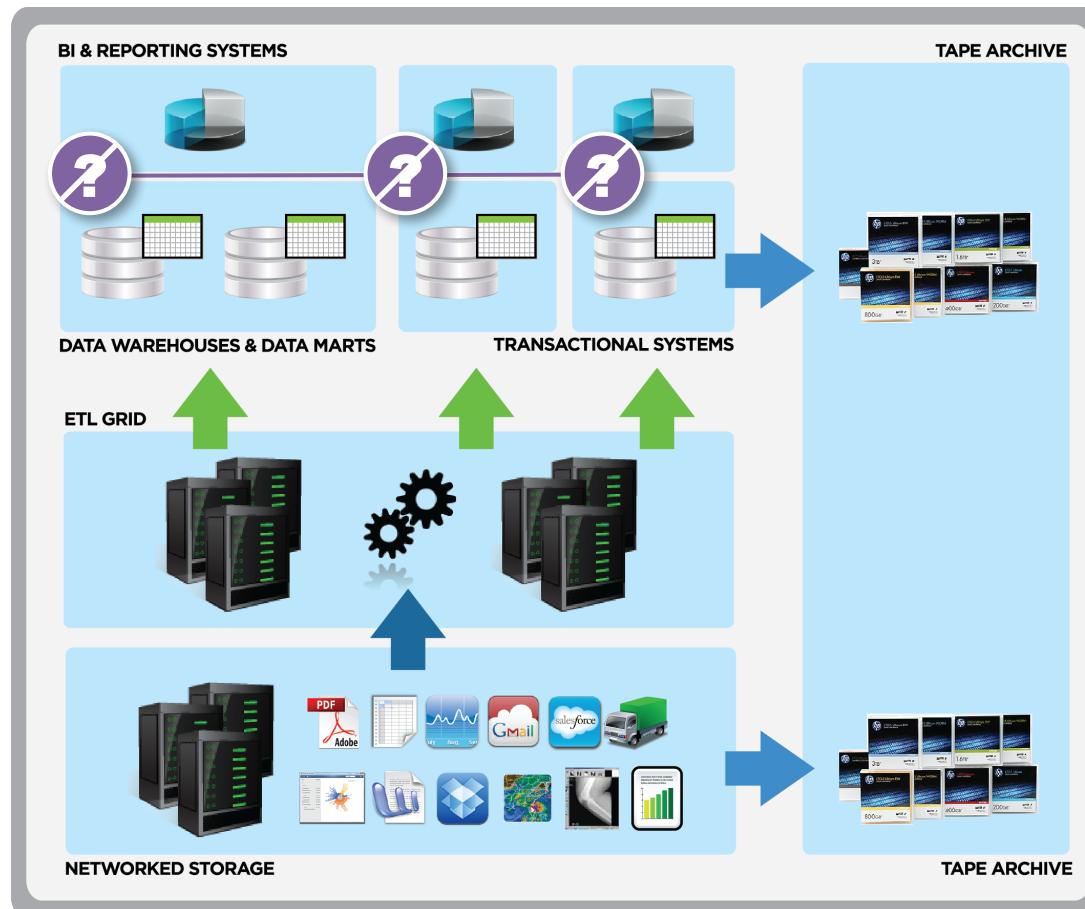
Daten können nicht wie benötigt verarbeitet werden



- Analyse und Verarbeitung dauert zu lange
- Daten existieren in Silos

Data Warehouse

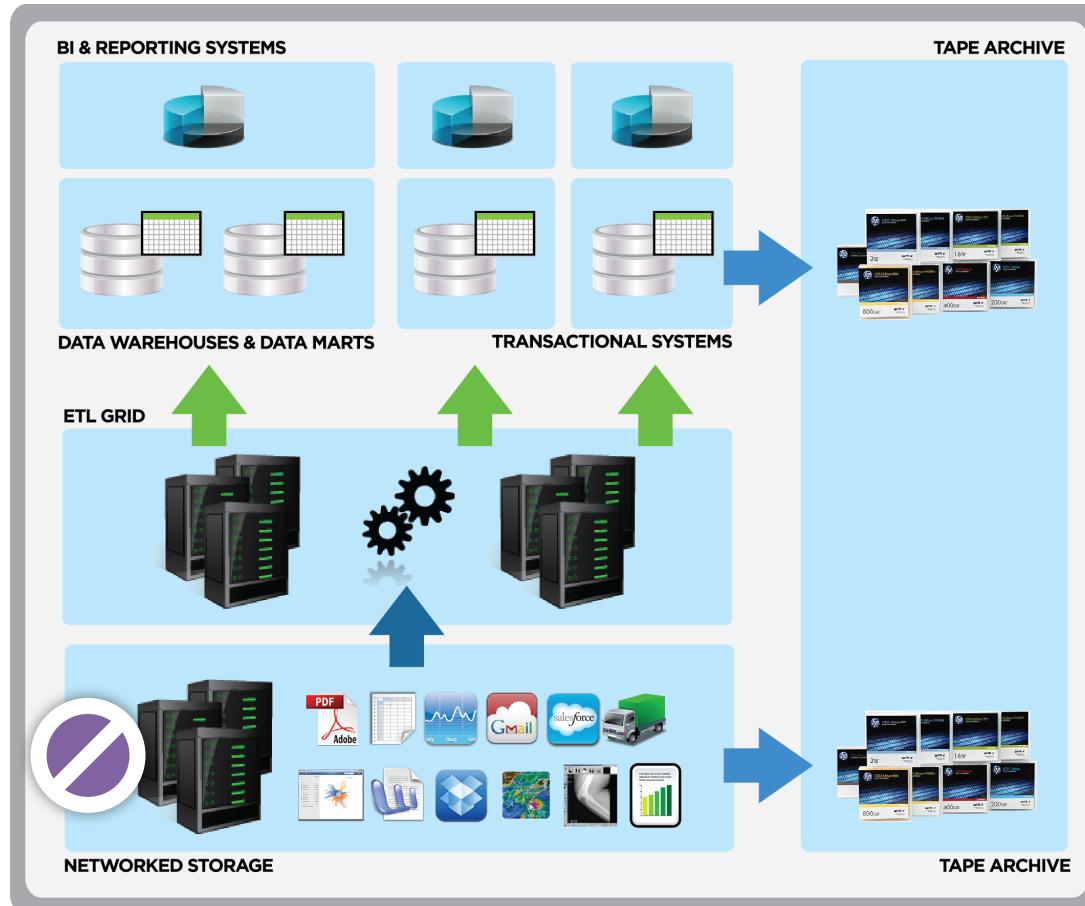
Daten können nicht wie benötigt verarbeitet werden



- Analyse und Verarbeitung dauert zu lange
- Daten existieren in Silos
- Neue Fragen können nicht gestellt werden

Data Warehouse

Daten können nicht wie benötigt verarbeitet werden



- Analyse und Verarbeitung dauert zu lange
- Daten existieren in Silos
- Neue Fragen können nicht gestellt werden
- Unstrukturierte Daten können nicht analysiert werden

Daten in Big Data



Daten liegen in **stetig** zunehmender Menge und in hauptsächlich **unstrukturierter** Form vor. Es Bedarf nicht nur der **Technology** sondern auch der **Personen** welche mit diesen umgehen können. Es entstehen neue, übergreifende Stellenbeschreibungen, wie **DevOps** oder **Data Scientist**.

Neue Stellenbeschreibungen



On Data Janitors, Engineers, and Statistics

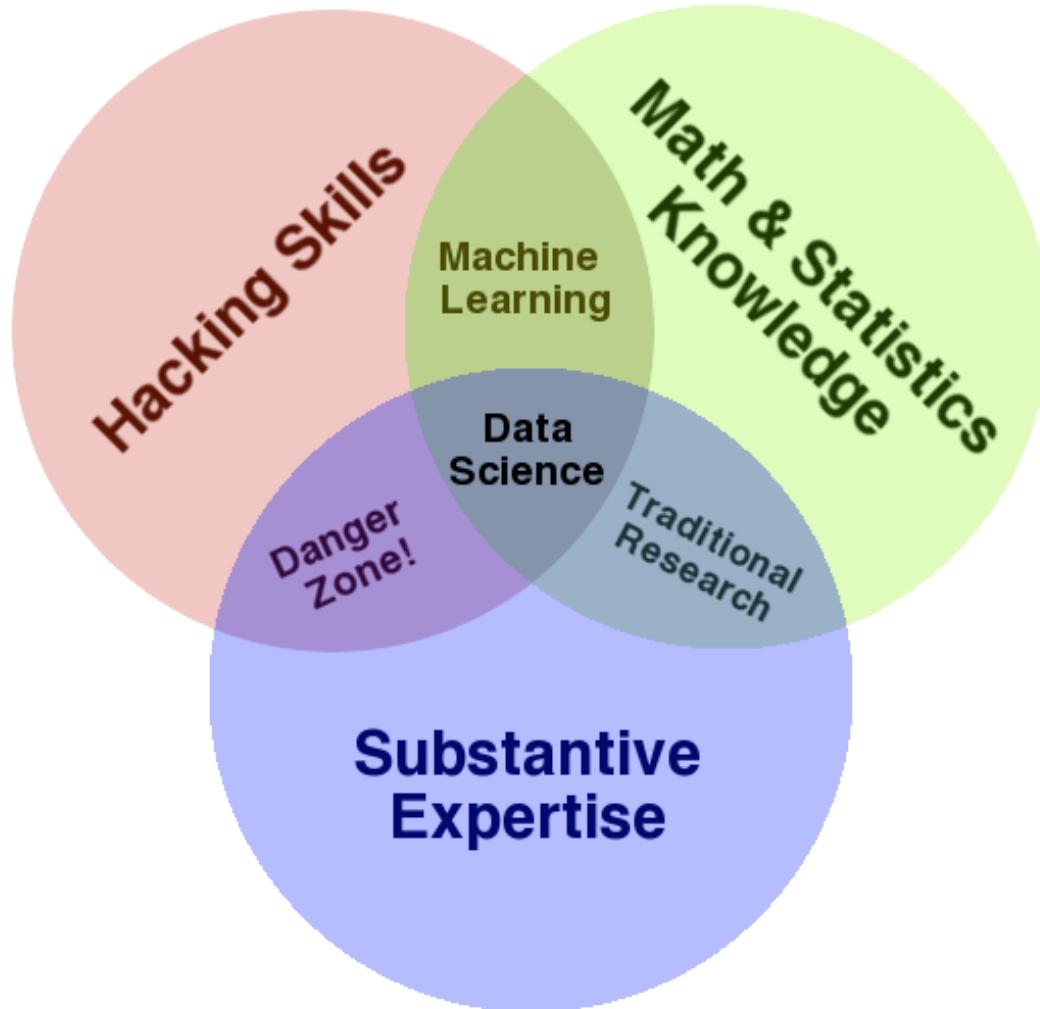
Posted on June 20, 2013



[Big Data Borat](#) tweeted recently that “Data Science is 99% preparation, 1% misinterpretation.” Commenting on the 99% part, Cloudera’s [Josh Wills says](#): “I’m a data janitor. That’s the sexiest job of the 21st century. It’s very flattering, but it’s also a little baffling.” [Kaggle](#), the data-science-as-sport startup, takes care of the “1% misinterpretation” part by providing a matchmaking service between the sexiest of the sexy data janitors and the organizations requiring their hard-to-find skills. It charges \$300 per hour for the service, of which \$200 go to the data janitor (at least in the case



Data {Scientist | Hausmeister}





Was ist Big Data?

Abschließende Anmerkungen:

Big Data ist eine **neue Disziplin** welche sich aus **bestehenden** weiterentwickelt hat.

Technologien und Personal müssen sich dabei **umstellen und anpassen**. Dies ist ein **längerer** Prozess und braucht Zeit, jedoch werden oft frühe **Hoffnungen** nur Stück für Stück erfüllt.



Einheit 1

- Übersicht
- Was bedeutet Big Data?
- Einführung in Hadoop
- Einführung in HDFS

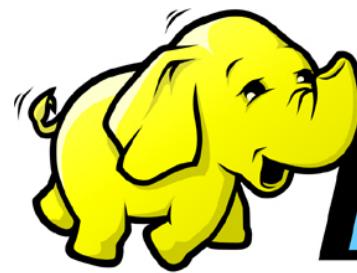


Einführung in Hadoop

Ein paar Fakten:

- Quelloffen und Apache lizenziert
- Viele Entwickler von namhaften Firmen
 - Cloudera, Yahoo!, Facebook, Apple etc.
- Hunderte Freiwillige welche Funktionen hinzufügen, Fehler beheben
- Viele zusammengehörige Projekte, Anwendungen, Werkzeuge, etc.

Das Hadoop Ökosystem





Wer benutzt Hadoop?



The New York Times

facebook



Autodesk

contextWeb

twitter

amazon.com



JPMORGAN CHASE & CO.

NAVTEQ





Wer benutzt Hadoop?

Die Frage ist eher, **wer nicht?** Oder was sind die **Alternativen?**

Viele Firmen machen Hadoop, wissen aber **nichts** damit anzufangen (siehe **Danger Zone** in Venn Diagramm).

Hadoop bietet viele Möglichkeiten, dabei gute wie auch weniger gute.



Wie bekomme ich Hadoop?

Es gibt heute mehrere Möglichkeiten, welche wir im einzelnen durchgehen:

- Apache Hadoop
- Quelloffene Distribution
- Kommerzielle Variante



Wie bekomme ich Hadoop?

Apache Hadoop:

Dies entspricht in etwa Linux und kernel.org, d. h. man muss alles selber übersetzen und packen – den Hadoop Core und alle abhängigen Projekte. Welche Version von A läuft dabei mit welcher Version von B zusammen?

Insgesamt ist dies die umständlichste Variante!



Wie bekomme ich Hadoop?

Quelloffene Distribution:

Diese kann man mit RedHat oder SUSE vergleichen, denn es gibt die kostenlose, quelloffenen Distribution und dann weitere Dienstleistungen und Softwarekomponenten welche nur mit einer Lizenz (und damit Kosten) zu haben sind.

Die kostenlose Distribution ist ideal für den Einsteiger, denn alles ist aus einem Guss.



Wie bekomme ich Hadoop?

Kommerzielle Distribution:

Hier sind die Teile des Hadoop Kerns durch geschlossene, proprietäre Eigenentwicklungen ausgetauscht. Damit steht auch die Kompatibilität zu nativen Hadoop Distribution in Frage. Außerdem sind von Anfang an Lizenzkosten zu erwarten.



Wie bekomme ich Hadoop?

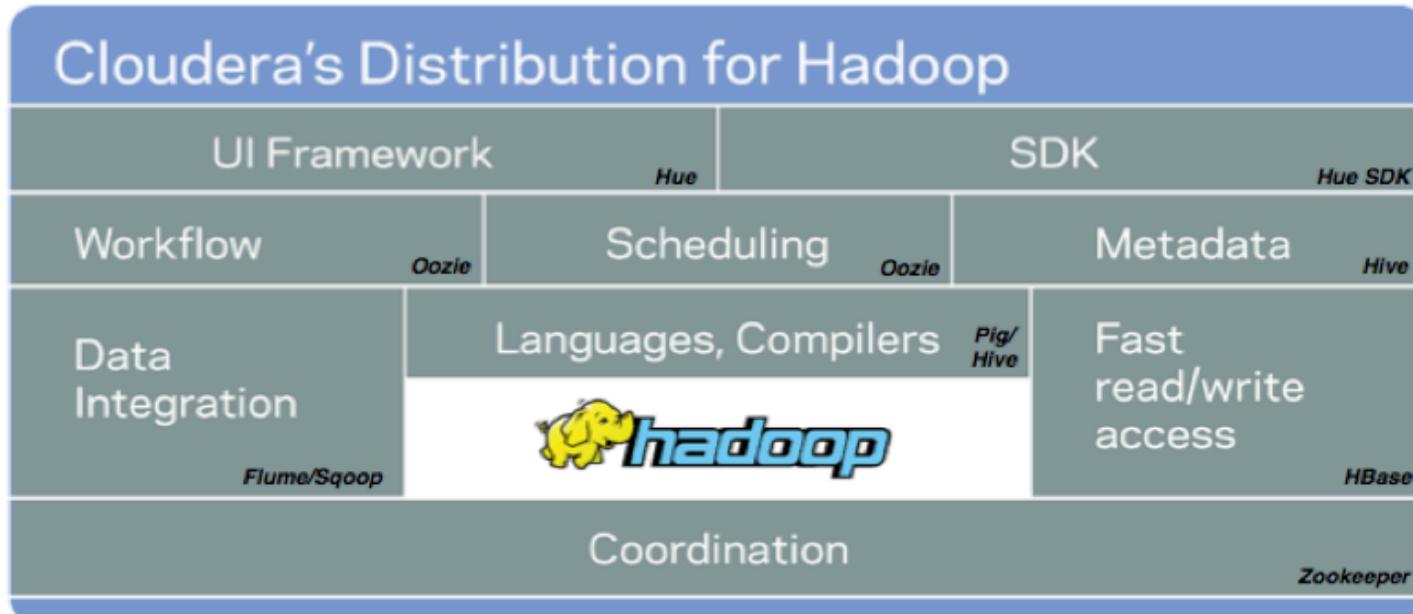
Beispiele:

- Apache Hadoop
 - <http://hadoop.apache.org>
- Quelloffene Distribution
 - Cloudera
 - HortonWorks
- Kommerzielle Variante
 - MapR
 - Pivotal
 - IBM



Wie bekomme ich Hadoop?

Beispiel: Cloudera





Warum Hadoop?

Im Folgenden schauen wir uns an **warum** Hadoop überhaupt benötigt wird. Dies schließt an die **vorherige** Diskussion „Warum Big Data“ und „Gründe für Big Data“ an und **erweitert** die Gesichtspunkte aus.

Traditionelle Großrechner

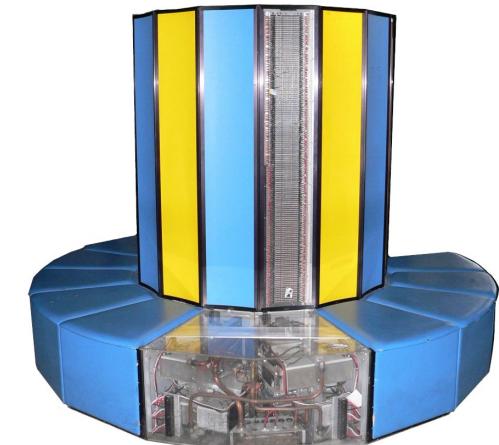


Traditionell war die Berechnung an die Leistung des Prozessors gebunden

- Relativ kleine Mengen an Daten
- Signifikanter Anteil von komplexen Berechnungen mit diesen Daten

Für Jahrzehnte war das Hauptziel die Leistungsfähigkeit eines einzigen Rechners zu erhöhen

- Schneller Prozessor, mehr Speicher





Verteilte Systeme

Moore's Gesetz sagt grob, dass sich die Rechenleistung alle zwei Jahre verdoppelt. Aber auch damit sind die Großrechner an ihre Grenzen gestoßen wenn es um CPU-intensive Berechnungen geht.

Verteilte Systeme sind aus diesem Grund weiterentwickelt worden, um Entwicklern mehrere Rechner für eine Berechnung (Job) zur Verfügung zu stellen (MPI, PVM, HTCondor).

Verteilte Systeme: Probleme



Die Programmierung der traditionellen verteilten System ist aber komplex

- Datenaustausch benötigt Synchronisierung
- Bandbreite ist begrenzt
- Zeitliche Abhängigkeiten sind kompliziert
- Teilausfall des Systems ist schwer zu handhaben

Von Ken Arnold, CORBA Designer:

„Ein Ausfall ist der definierende Unterschied zwischen verteilter und lokaler Programmierung, deshalb muss man ein verteiltes System mit der Erwartung eines Ausfalls entwerfen.“ ← Mehraufwand!!!

Verteilte Systeme: Speicherung



Typischerweise werden die Daten in einem **verteilten** System einem Storage Area Network (**SAN**) gespeichert.

Bei der Ausführung der Berechnung werden die **Daten** dann zum Rechnerknoten **kopiert**.

Das funktioniert gut mit relativ **begrenzter** Menge an **Daten**.



Die datengetriebene Welt

Moderne Systeme müssen sich mit weitaus mehr Daten auseinander setzen als in der Vergangenheit

- Organisationen generieren riesige Mengen an Daten
- Diese Daten haben einen (inhärenten) Wert und dürfen nicht verworfen werden

Beispiele wurden bereits genannt mit Facebook und Twitter. eBay hält 24PB vor – zweimal! Viele Firmen generieren Terabytes an Daten jeden Tag.



Datenflaschenhals

Das Transportieren der Daten zum Prozessor wird zum **Flaschenhals**.

Eine (einfache) Rechnung:

- Typische Festplatten Transferrate: 100MB/s
- Zeit benötigt **100GB** an Daten an den Prozessor zu liefern: **16 Minuten!**

Dies setzt eine feste Transferrate voraus. Die wirkliche Zeit sieht **schlechter** aus den wenigen Rechner können auch 100GB im Speicher halten.

Es ist zwingend einen **neuen** Ansatz zu finden.



Anforderungen

Im folgenden schauen wir uns an, was ein solches, neues System alles an Anforderungen mitbringen sollte. Diese reflektieren direkt die gerade angesprochenen Probleme der traditionellen System.

Unterstützung des Teilausfalls



Das System muss **Teilausfälle** aushalten können.

Wenn Komponenten ausfallen sollte die Leistung nur **leicht** prozentual zurück gehen, aber das ganze System läuft **weiter** und fällt **nicht** ganz aus.



Datenwiederherstellung

Wenn eine Komponente des Systems ausfällt
übernimmt eine noch funktionierende Einheit
dessen **Aufgaben**.

Ausfall darf **niemals** Datenverlust bedeuten.

Komponentenwiederherstellung

Wenn eine **ausgefallene** Komponente des Systems wieder zum Laufen gebracht wurde, dann kann sie dem System **wieder** beitreten.

Dies erfordert **keinen** Neustart des ganzen Systems.



Konsistenz

Wenn eine Komponente des Systems **während** der Laufzeit eines Berechnungsauftrags **ausfällt**, dann hat dies **keinerlei** Einfluss auf das Ergebnis der Berechnung.



Skalierbarkeit

Wenn die **Last** auf dem System sich **erhöht**, dann drückt sich das durch eine **leichte** Verschlechterung aller **individuellen** Berechnungsaufträge aus. Auf **keinen** Fall fällt das **ganze** System aus.

Wenn die Ressourcen **erhöht** werden, dann drückt sich dies in einer anteiligen Erhöhung der **Lastkapazität** aus.



Hadoop

Kommen wir nun zu dem **Kern** des heutigen Tages: **Hadoop**. Wir werden im Anschluss die Geschichte, Konzepte und Architektur von Hadoop besprechen.

Die **Übung** im Anschluss wird dann eine erste Möglichkeit darstellen Hadoop selbst **kennenzulernen**.



Hadoop's Geschichte

Hadoop basiert auf der Arbeit welche Google am Anfang der 2000er investiert hat:

- Speziell sind die Fachveröffentlichungen über das Google File System (GFS) in 2003 und MapReduce in 2004 zu nennen.

Diese Arbeit wählte einen radikal neuen Ansatz des Problems bei der Entwicklung verteilter Systeme: sie sollte allen genannten Anforderungen genüge tun.



Hadoop's Geschichte

Kernkonzept:

Verteile die Daten während diese im System ursprünglich gespeichert werden.

Individuelle Rechner (Knoten) können dann die **lokalen** Daten verarbeiten – es werden **keine** Daten über das Netzwerk oder auch anderweitig außerhalb des Rechners **transferiert**.



Hadoop's Hauptkonzepte

Anwendungen werden in einer **höheren** Programmiersprache geschrieben

- Entwickler können die Probleme z. B. der Netzwerkprogrammierung, oder zeitlicher Abhängigkeiten komplett **ignorieren**

Knoten kommunizieren so **wenig** wie möglich

- Entwickler sollten es nicht nötig haben Code zu schreiben der sich mit der Kommunikation zwischen Knoten beschäftigt
- „**Shared Nothing**“ Architektur

Daten werden vor der Benutzung über Knoten **verteilt**

- Berechnung erfolgt dort, wo die Daten gespeichert sind, solange dies möglich ist.
- Daten werden mehrfach **repliziert** im System für erhöhte Verfügbarkeit und Ausfallsicherheit



Hadoop: Einfache Übersicht

Wenn Daten im System gespeichert werden, dann werden diese automatisch in **Blöcke** aufgeteilt: typischerweise 64MB oder 128MB.

Map Tasks (der erste Teil des MapReduce Systems) arbeiten auf **kleineren** Teilen der Daten: typischerweise ein einzelner Block.

Ein **Master** Programm **verteilt** Arbeit an Knoten mit Rücksicht auf **Datenlokalität**: viele Knoten arbeiten **parallel**, jeden auf dessen Teilbereich.



Fehler-/Ausfalltoleranz

Wenn ein Knoten **ausfällt**, dann **erkennt** dies der Master und verteilt die **gleiche** Arbeit an einen **anderen** Knoten im System.

Der **Neustart** eines Teilauftrags (Task) erfordert **keine** Kommunikation mit Knoten welche an **anderen** Teilaufträgen arbeiten.

Wenn ein **ausgefallenen** Knoten neu startet wird er dem System **automatisch** wieder hinzugefügt und bekommt Arbeit **zugeteilt**.

Wenn ein Knoten augenscheinlich sehr **langsam** arbeitet, kann der Master **redundant** die gleiche Teilaufgabe einem **anderen** Knoten zuordnen: wer **zuerst** fertig wird gewinnt.



Hadoop: Einfache Übersicht

Hiermit haben wir die Konzepte und Anforderungen an ein modernes verteiltes System anhand von Hadoop vorgestellt.

Wir haben einleitend untersucht, warum traditionelle Systeme großen Datenmengen nicht gewachsen sind, da deren Architektur andere Ziele verfolgen.

Schauen wir uns nun an wie sich Hadoop zusammensetzt.



Hadoop: Komponenten

Hadoop besteht aus **zwei** Kernkomponenten:
dem Hadoop Distributed File System (**HDFS**) und
dem **MapReduce** Software Framework bzw.
YARN.

Es gibt viele **weitere** Projekt rund um diesen
Hadoop Kern, umgänglich als **Hadoop**
Ökosystem bezeichnet: Hive, HBase, Flume,...



Hadoop: Komponenten

Eine **Verbund** von Rechnern welche HDFS und MapReduce bzw. YARN ausführen werden als **Hadoop Cluster** bezeichnet.

Individuelle Rechner werden also Knoten bezeichnet.

Ein Cluster kann von ein paar bis mehrere tausende Knoten enthalten.

Grundregel: Mehr Knoten = höhere Leistung



Hadoop: HDFS

HDFS, das Hadoop Distributed File System, ist für die Speicherung der Daten im Cluster zuständig.

Datendateien werden in Blöcke aufgeteilt und über mehrere Knoten im Cluster verteilt.

Jeder Block wird mehrfach repliziert, wobei die Vorgabe 3-mal ist. Replizierte Blöcke werden auf unterschiedlichen Knoten gespeichert. Dies erhöht die Ausfallsicherheit und Verfügbarkeit.



Hadoop: MapReduce

MapReduce ist ein System zum **Verarbeiten** von Daten innerhalb des Hadoop Clusters.

Es besteht aus **zwei** Phasen: **Map** und dann **Reduce**.

Jede Map Teilaufgabe (Task) arbeitet auf eine **diskreten** Teilmenge des **gesamten** Datensatzes (normalerweise ein HDFS Datenblock).

Nachdem alle Map Tasks fertig sind, **transferiert** das MapReduce System die Daten zu **wenigen** Knoten, um die Reduce Phase durchzuführen.



Hadoop: YARN

YARN steht für Yet Another Resource Negotiator und bricht das bestehende **monolithische** MapReduce System in **zwei** Teile: **Ressourcen Verwaltung und Anwendungen.**

Dies ist ähnlich dem Linux Kernel und User-Land Anwendungen. YARN kann **beliebige** verteilte Verarbeitungssystem **gleichzeitig** ausführen.

MapReduce wurde hier API-kompatibel als Anwendung **neu** implementiert.



Hadoop: Komponenten

Dies schließt die einleitende Übersicht ab. Wir haben gesehen warum Hadoop Sinn macht und aus welchen Komponenten dessen Kern besteht.

Wir schauen uns nun HDFS im Detail an, welches auch den Abschluss der heutigen Übungen darstellt. Jeder Student sollte HDFS am laufen haben und Dateien lesen uns speichern können.



Einheit 1

- Übersicht
- Was bedeutet Big Data?
- Einführung in Hadoop
- **Einführung in HDFS**



Einführung in HDFS

Grundlegende **Konzepte** des HDFS:

Es ist ein Dateisystem welches in **Java** geschrieben ist. Es basiert auf den Ideen des Google File System (**GFS**).

HDFS sitzt auf einem **nativen** Dateisystem auf, z. B. ext3, ext4, xfs etc.

Es bietet **redundanten** Speicher für **große** Datenmengen, unter Benutzung von kosteneffektiven, **unsicheren** Rechner.



Einführung in HDFS

Fortsetzung:

HDFS ist am leistungsfähigsten mit einer „vernünftigen“ Anzahl sehr großer Dateien: **Millionen** anstatt Milliarden von Dateien und jede dieser Dateien typischerweise **1GB** und größer.

Alle Dateien in HDFS sind „**einmal schreiben**“ (write once), d. h. es gibt keine freien Schreibzugriffe.

Einführung in HDFS



Fortsetzung:

HDFS ist **optimiert** für große, **fließende** (streaming) Leseoperation auf den Dateien. Also **sequentielle** Zugriffe auf **große** Blöcke, welche den Datendurchsatz der Hardware **voll** ausreizen. **Zufällige** Lesezugriffe laufen der aktuellen Hardwarearchitektur **entgegen** (Disk Seek, also die Schreib-/Lesearmbewegungen sind sehr **langsam** im Vergleich).

Speicherung der Dateien

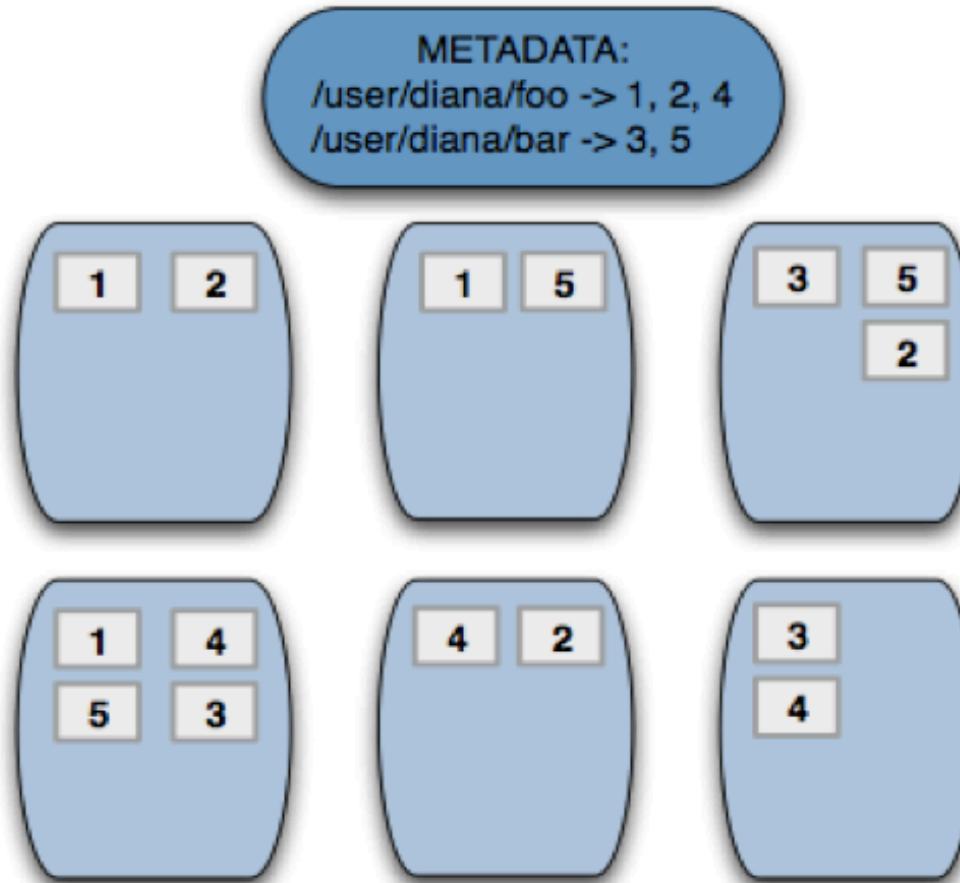


Die Dateien sind in Blöcke **aufgeteilt**.

Die Daten werden verteilt auf viele Rechner während der Speicherung. Verschiedene Blöcke derselben Datei werden auf **verschiedenen** Rechnern abgelegt. Dies **hilft** auch bei dem verteilten Bearbeiten der Daten mit MapReduce (in Einheit 2 besprochen) um dessen Effizienz zu **erhöhen**.

Speicherung der Dateien

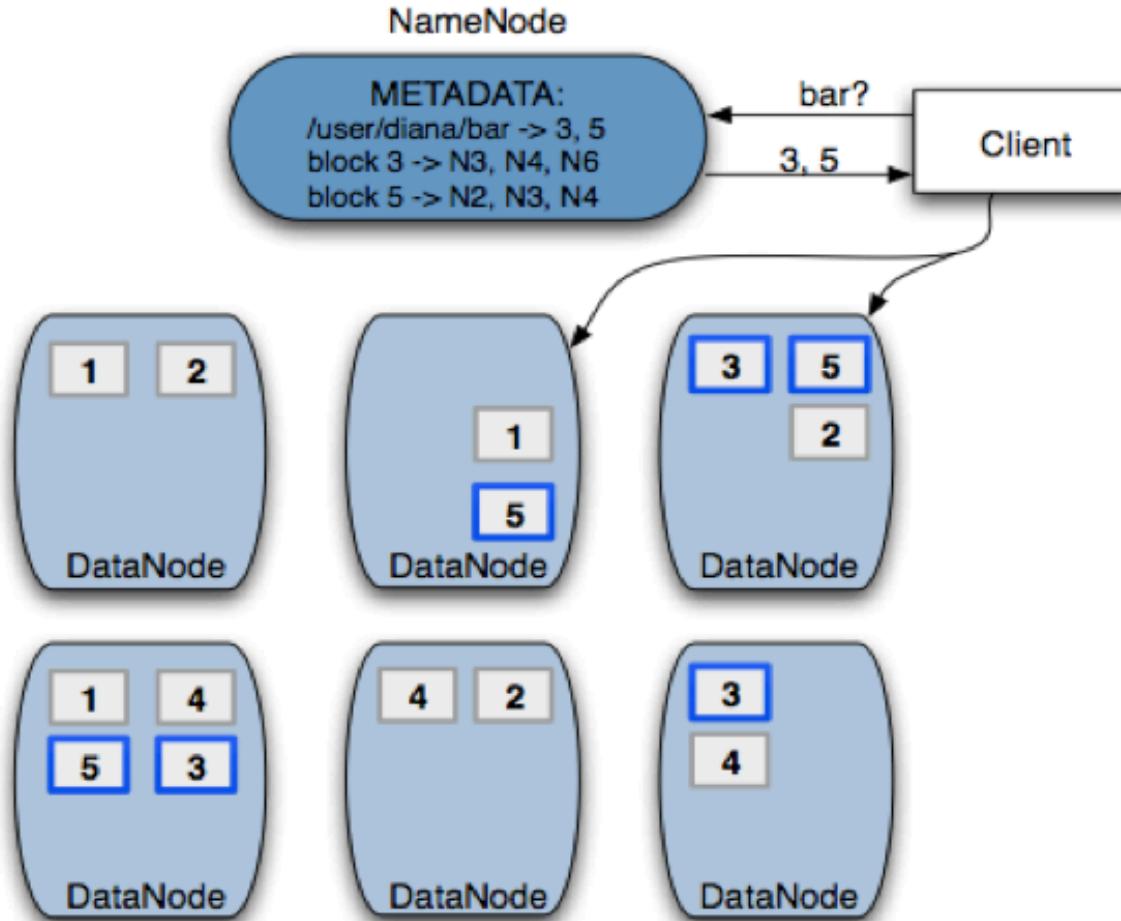
NameNode: Speichert nur Metadaten



DataNode: Speichert die Daten

- **NameNode** verwaltet die Metadaten für die Datendateien
- **DataNodes** speichern die eigentlichen Daten
 - Jeder Block wird dreimal repliziert abgelegt im Cluster

Lesen der Dateien



Lesezugriff durch Anwendung:

- Die Anwendung kommuniziert mit dem NameNode, um die Datenblöcke der gefragten Datei zu bestimmen und wo diese abgelegt sind.
- Anschließend kommuniziert die Anwendung direkt mit dem DataNodes um die Daten zu lesen.

Zugriff auf Dateien



Hadoop Kommandozeile:

```
$ hadoop fs -copyFromLocal <local_dir> <hdfs_dir>
$ hadoop fs -copyToLocal <hdfs_dir> <local_dir>
```

Hadoop Projekte:

- Flume – Sammelt Daten aus Logquellen (z. B. Webserver, syslog, STDOUT)
- Sqoop – Transportiert Daten aus und/oder nach HDFS aus relationalen Datenbanken

Business Intelligence Werkzeuge



Anatomie eines Clusters

Bis hierhin haben wir nun gesehen wie HDFS die Dateien ablegt, die Daten dabei zerlegt werden, und wie eine Anwendungen über NameNode und DataNodes die Datei wieder lesen kann.

Nachfolgend wird betrachtet wie ein Hadoop Cluster im einzelnen aufgebaut ist.



Installation eines Clusters

Die Installation wird normalerweise durch einen Systemadministrator durchgeführt. Diese werden durch entsprechende Schulungen vorbereitet.

Wir werden die Cloudera Distribution für Hadoop benutzen und in einer virtuellen Maschine (VM) installieren. Dieses ist mehr als ausreichend, um Hadoop komplett zu benutzen – nur nicht als verteiltes System.



Hadoop Prozesse

Hadoop setzt sich aus **5** verschiedenen Prozessen zusammen:

1. **NameNode**

Verwaltet die Metadaten für HDFS.

2. Secondary oder Standby NameNode

a. **Secondary** NameNode

Übernimmt Hintergrundarbeiten für den NameNode, ist aber kein Backup oder Hot-Standby für den NameNode.

a. **Standby** NameNode für HA Modus

Im Hochverfügbarkeitsmodus ist dieser NameNode gleichzeitig Hot-Standby als auch Hintergrundarbeiter.



Hadoop Prozesse

Fortsetzung:

3. DataNode

Speichert die eigentlichen HDFS Datenblöcke.

4. JobTracker

Verwaltet MapReduce Aufträge und verteilt Teilaufträge ab TaskTracker.

5. TaskTracker

Verantwortlich für das Erstellen und Überwachen der individuellen Map und Reduce Tasks.



Risiko Verminderung

Single Point of Failure (**SPoF**), also „einzelne Stelle des Scheiterns“:

- NameNode
- Secondary NameNode
- JobTracker

Diese sollten im Produktivcluster im **HA** Modus aufgesetzt werden. Außerdem sind die Rechner besser ausgestattet als die Arbeiterknoten.

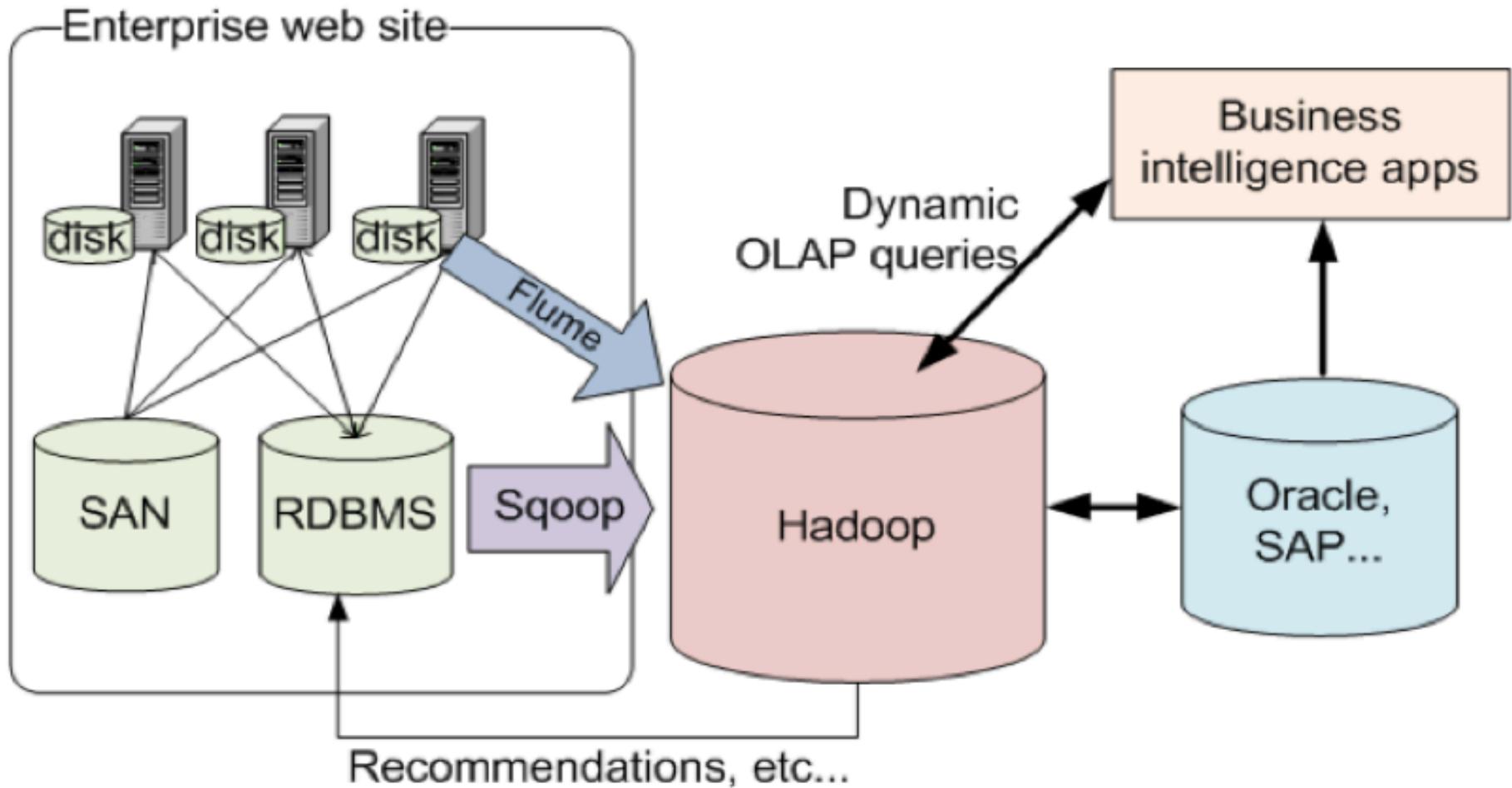
Integration im Rechenzentrum



Hadoop bietet sich an als **Erweiterung** bestehender IT-Systeme eingesetzt zu werden. Dazu werden Daten aus den existierenden Systemen in Hadoop **langzeitgespeichert**. Durch die Verarbeitungskomponente MapReduce können **zusätzlich** weitere Aufgaben in Hadoop nach und nach abgebildet werden. Dieser **inkrementeller** Wandel ist einer der großen Stärken von Hadoop.



Integration im Rechenzentrum





Das Rechenzentrum

Ein Rechenzentrum hat normalerweise:

- Relationale Datenbanken
- Data Warehouses
- Dateiserver (SAN, NAS)
- Backup Systeme

Das Rechenzentrum

Jedes dieser Systeme hat Vorteile:

Technologie	Stärken	Schwächen
Relationale Datenbank	Komplexe Transaktionen, tausende Abfragen pro Sekunde, SQL	Daten müssen in Zeilen und Spalten passen, Schema ist nur umständlich änderbar, durchlesen ganzer Tabellen ist langsam und löscht den Cache, SQL
Data Warehouses	Reportmöglichkeiten, bis zu hunderten von Terabytes an Daten	Dimensionen benötigen vorzeitige Materialisierung
Dateiserver	Zugriff auf einzelne Dateien, Schreibcaches	Kosten, lesen von großen Teilen der Daten füllt das Netzwerk aus
Backup Systeme (Tape)	Billig	Teuer auf Daten zuzugreifen



Hadoop's Stärken

Hadoop kann sehr große Datenmengen importieren **ohne** das ein Schema definiert werden muss (Schema on Read).

Werkzeuge für den Export und Import **existieren** mit Flume, Sqoop und kommerziellen Lösungen.

Speicher und Verarbeitung **skalieren** auf Petabyte Größe.



Hadoop's Schwächen

Hadoop ist weniger geeignet

- für kleine, strukturierte Datensätze, welche interaktive Abfragen benötigen (Latenzzeiten im Millisekunden Bereich), z. B. um einen Webservice zu betreiben
- wenn Transaktionen benötigt werden
- für strukturierte Analyse



Einheit 1

An dieser Stelle endet die erste Einheit mit einer Einführung in Big Data Engineering. In der nächsten Einheit werden wir auf weitere Merkmale eines Hadoop Clusters eingehen und MapReduce kennenlernen.

Bis bald!