

Big Data Engineering

Konstruktion datenintensiver Systeme

christian.zirpins@hs-karlsruhe.de

Big Data Speicherung



Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES



BDE-Termine — Winter 2018

#	KW	Termin	Block	Thema	Raum
	40	8.10.			
1	41	15.10.		Big Data Paradigma	Hörsaal
2	42	22.10.	Batch Layer	Big Data Modellierung	Hörsaal
3	43	29.10.		Big Data Speicherung	Hörsaal
4	44	5.11.		<i>Übung: Hadoop DFS/Thrift</i>	Labor
5	45	12.11.		Batch Verarbeitung	Hörsaal
6	46	19.11.		<i>Übung: Hadoop Map Reduce</i>	Labor
7	47	26.11.	Serving Layer	Big Data Bereitstellung	Hörsaal
8	48	3.12.		<i>Übung: Cascalog</i>	Labor
9	49	10.12.	Speed Layer	Big Data in Echtzeit	Hörsaal
10	50	17.12.		Queues und Stream Verarbeitung	Hörsaal
	51	24.12.			
	52	31.12.			
11	1	7.1.		<i>Übung: Apache Kafka/Storm</i>	Labor
12	2	14.1.		Micro-Batch Stream Verarbeitung	Hörsaal
13	3	21.1.		Lambda Architektur	Hörsaal
14	4	28.1.		Outro / Q&A / Abgabe	Labor

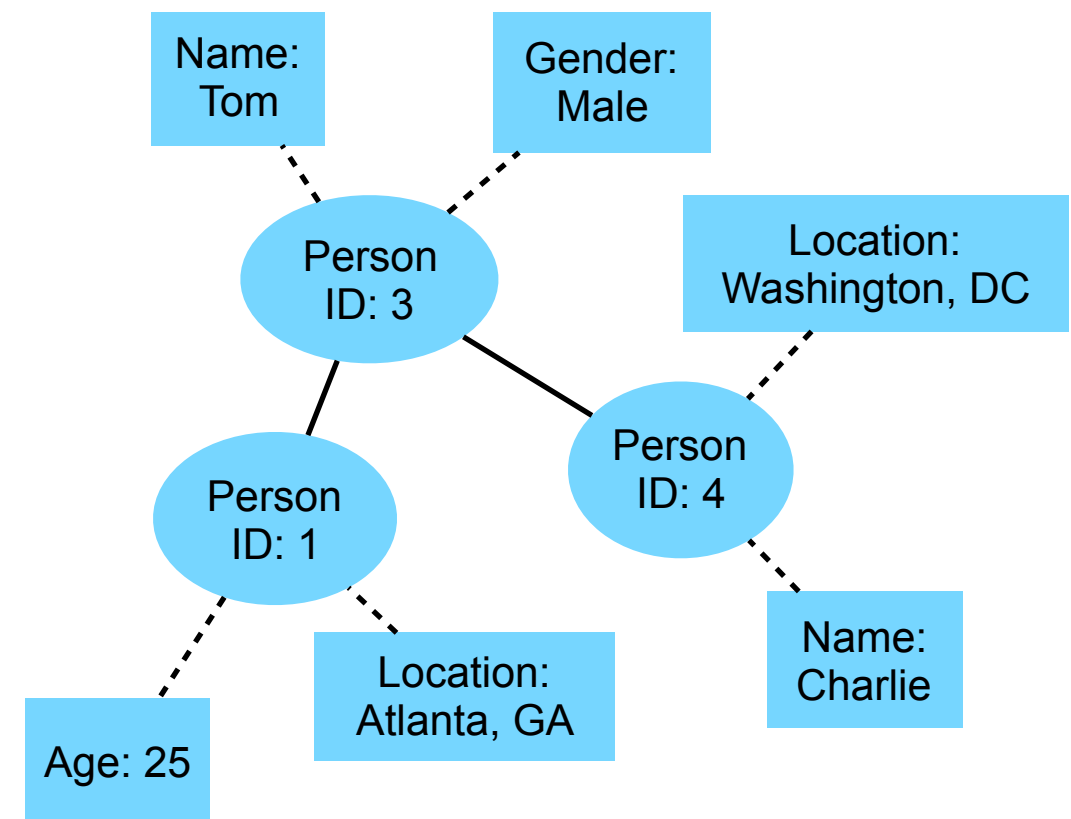
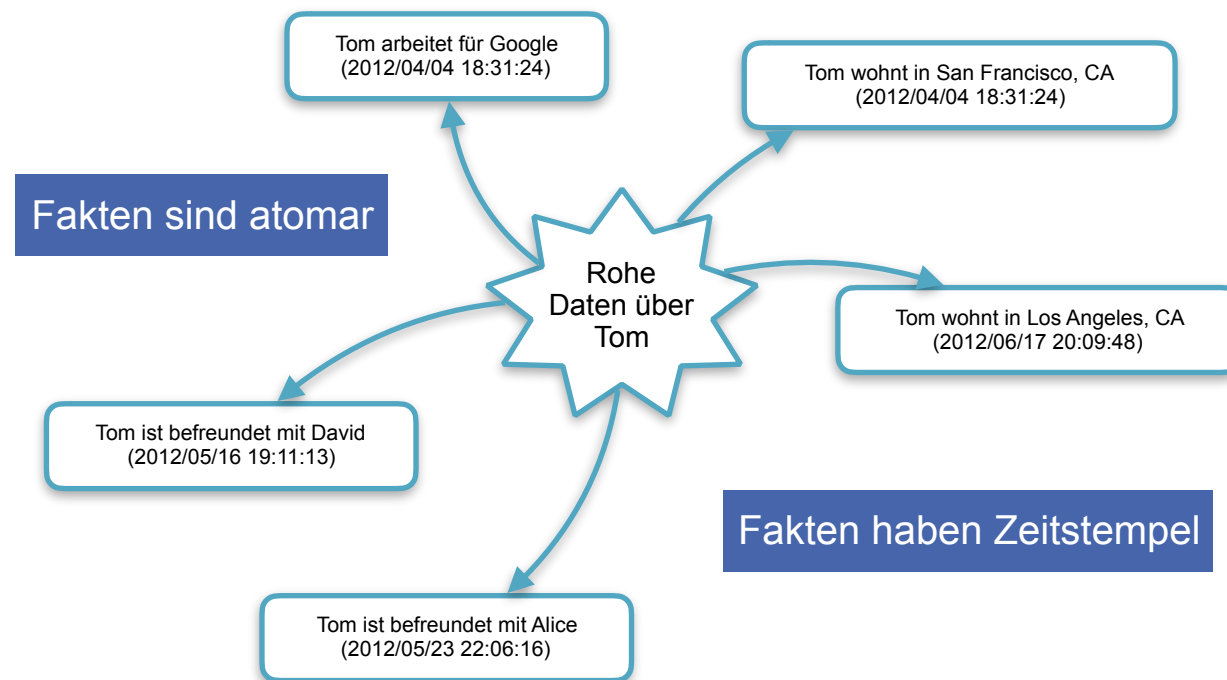
Raumplan

Hörsaal	E 303
Labor	LI137

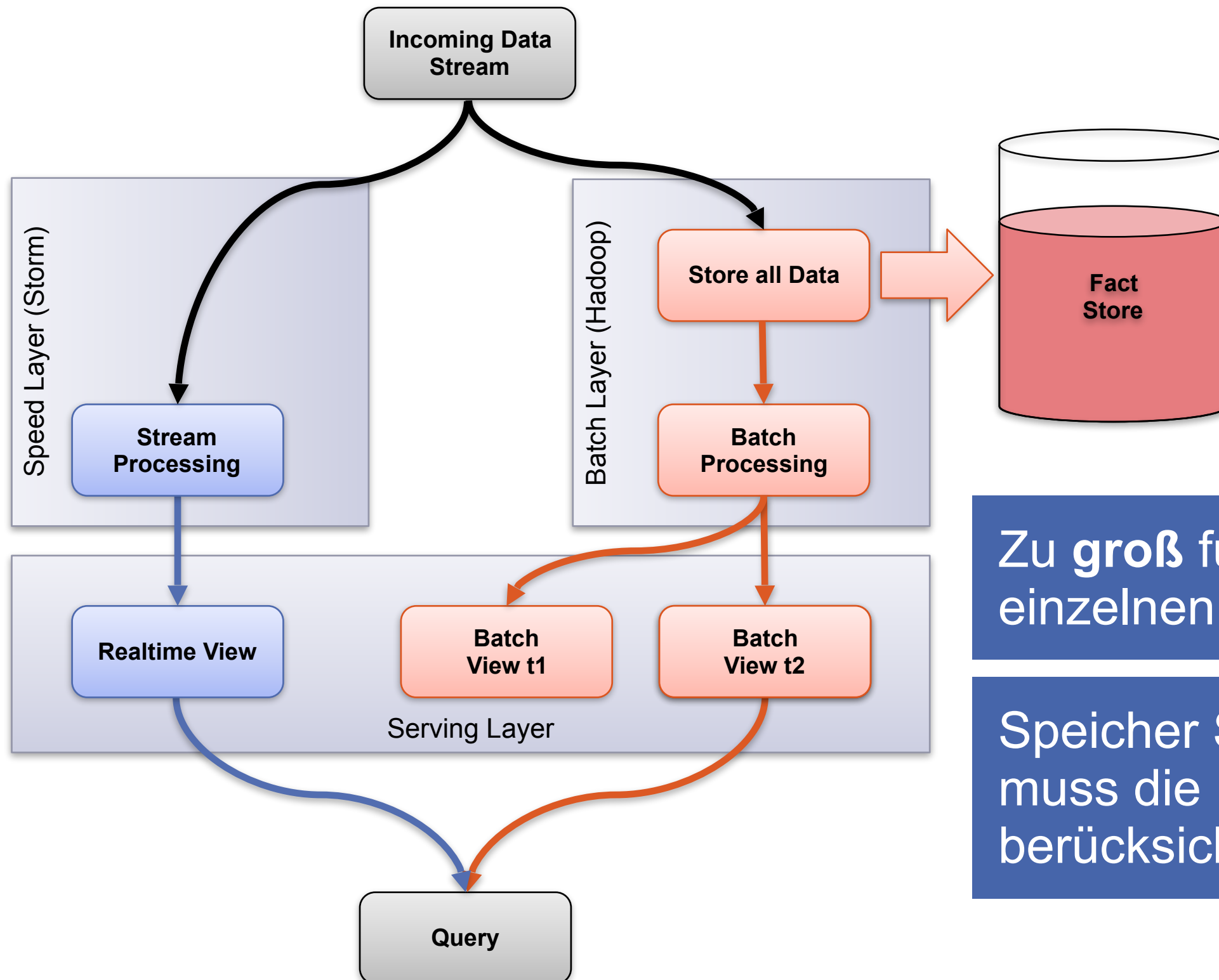
Kurzer Rückblick

Letzte Vorlesung haben wir...

- ... die **Eigenschaften von Daten** untersucht
- ... Daten **faktenbasiert** modelliert
- ... Fakten durch **Graph Schemas** strukturiert
- ... Graph Schemas mit **Thrift** implementiert



Master Datenmenge in der Lambda Architektur



Zu groß für einen einzelnen Server!

Speicher Strategie muss die **Nutzung** berücksichtigen.

Lernziele

Nach dieser Vorlesung können Sie...

- **Speicheranforderungen** für Master Datenmenge **erklären**
- **Verteilte Dateisysteme** **einordnen**
- **Vertikale Partitionierung** zur Effizienzoptimierung **einsetzen**
- Das **Hadoop Distributed File System (HDFS)** **benutzen**
- **Pail** zur Datenmanipulation in HDFS **verwenden**

Speicheranforderungen für die Master Datenmenge

Anforderungen für den Batch Speicher

- Daten sind unveränderlich und ewig wahr: Speicher muss großen, wachsenden Datensatz halten 💬
- `batch view = function(all data)`: Speicher soll viele Daten auf einmal Lesen können



Operation	Anforderung
Write	💬 Effizientes Anhängen neuer Daten
Read	Skalierbarer Speicher
	Unterstützung für parallele Verarbeitung
beide	Optimierung von Speicher- und Verarbeitungskosten
	💬 Durchsetzbare Unveränderbarkeit

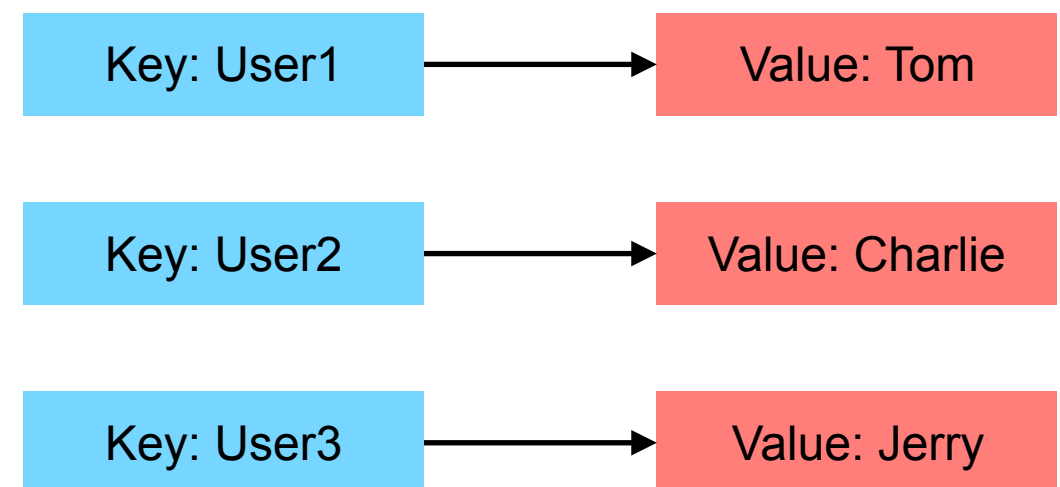
Key/Value-Store als Batch Speicher

Key/Value Store: riesige persistente HashMap, auf viele Maschinen verteilt



Fakten im Key/Value Store

- Es gibt keinen natürlichen **Schlüssel** im faktenbasierten Modell 
- **Bulk Reads** (viele Werte auf einmal) sind nicht vorgesehen 
- Key/Value-Paare werden nicht gemeinsam **komprimiert**
- Key/Value-Paare sind **änderbar**
- Wahlfreier Zugriff **überflüssig**



Dateisystem als Batch Speicher

Ein Dateisystem ist perfekt

- Dateien sind Bytes ...
- ...sequentiell auf Platte geschrieben
- ...einfach darüber zu lesen
- ...alle frei kontrollier-/komprimierbar
- ...mit Zugriffskontrolle (read only)
- ...all you need 😊

Dateisystem als Batch Speicher

Ein Dateisystem ist wäre perfekt

- Dateien sind Bytes ...
- ...sequentiell auf Platte geschrieben
- ...einfach darüber zu lesen
- ...alle frei kontrollier-/komprimierbar
- ...mit Zugriffskontrolle (read only)
- ...all you need 😊
- ...auf einem Rechner 😞

Dateisystem als Batch Speicher

Ein Dateisystem ist wäre perfekt

- Dateien sind Bytes ...
- ...sequentiell auf Platte geschrieben
- ...einfach darüber zu lesen
- ...alle frei kontrollier-/komprimierbar
- ...mit Zugriffskontrolle (read only)
- ...all you need 😊
- ...auf einem Rechner 😞

→ **Verteilte Dateisysteme**
verteilen ihren Speicher
über ein Computer Cluster

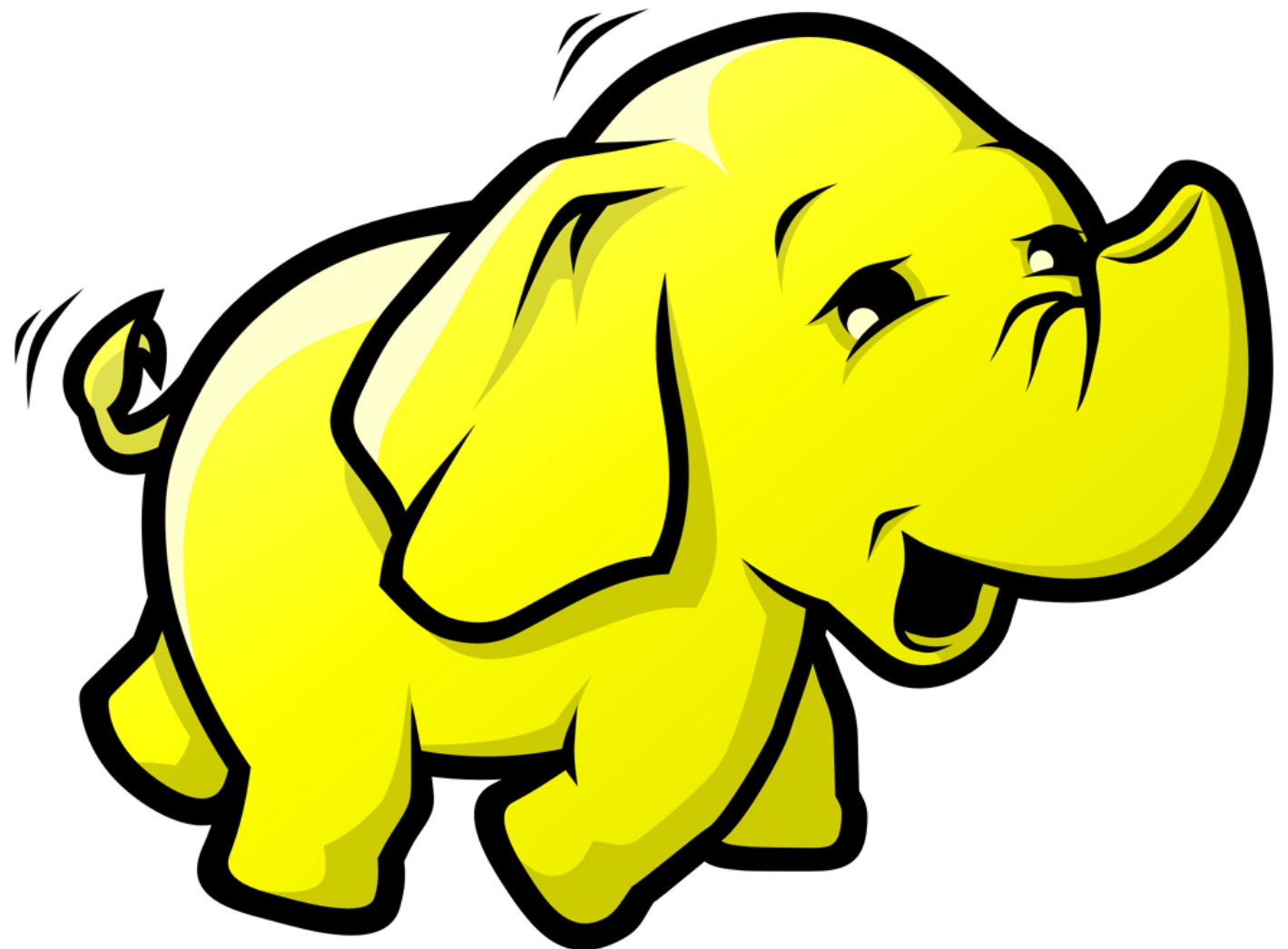
Sie **skalieren**, indem sie
dem Cluster weitere
Computer hinzufügen

Sie sind **fehlertolerant** bei
Verlust eines Computers:
Dateien sind dann immer
noch zugänglich

Verteilte Dateisysteme als Speicher für die Master Datenmenge

Verteilte Dateisysteme am Beispiel von **HDFS**

- Das Hadoop **Distributed File System (HDFS)** ist ein verteiltes, fehlertolerantes **Speichersystem**, das auf Petabyte Daten skaliert
- Hadoop **MapReduce** ist Programmiermodell und horizontal skalierbares Framework für **Batch Verarbeitung**, das *mit HDFS integriert* ist
- Teilprojekte vom **Apache Hadoop-Projekt**
- <http://hadoop.apache.org>



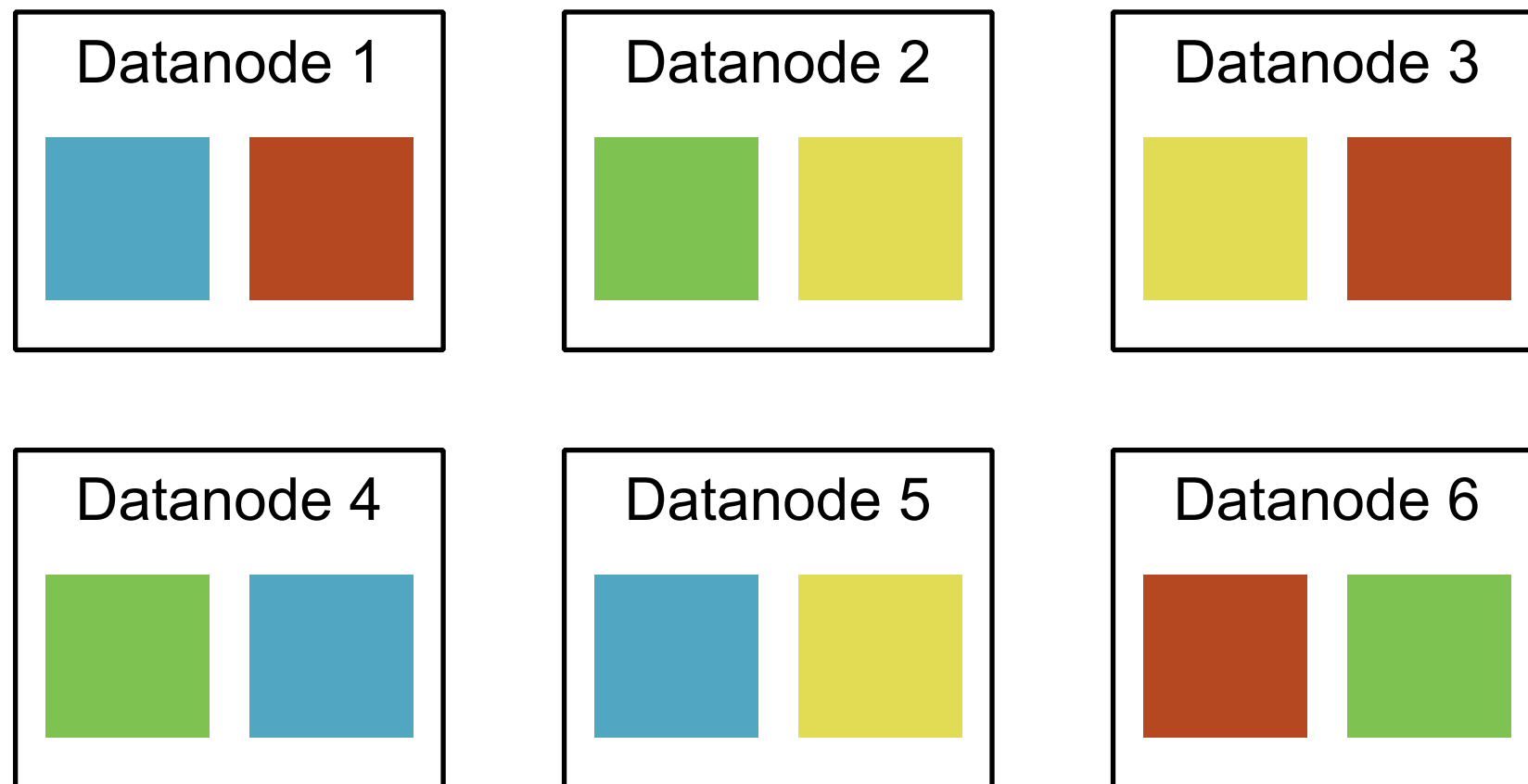
Prinzip von HDFS



Alle (typisch großen) Dateien werden in **Blöcke** zerlegt (meist 64 bis 256 MB)

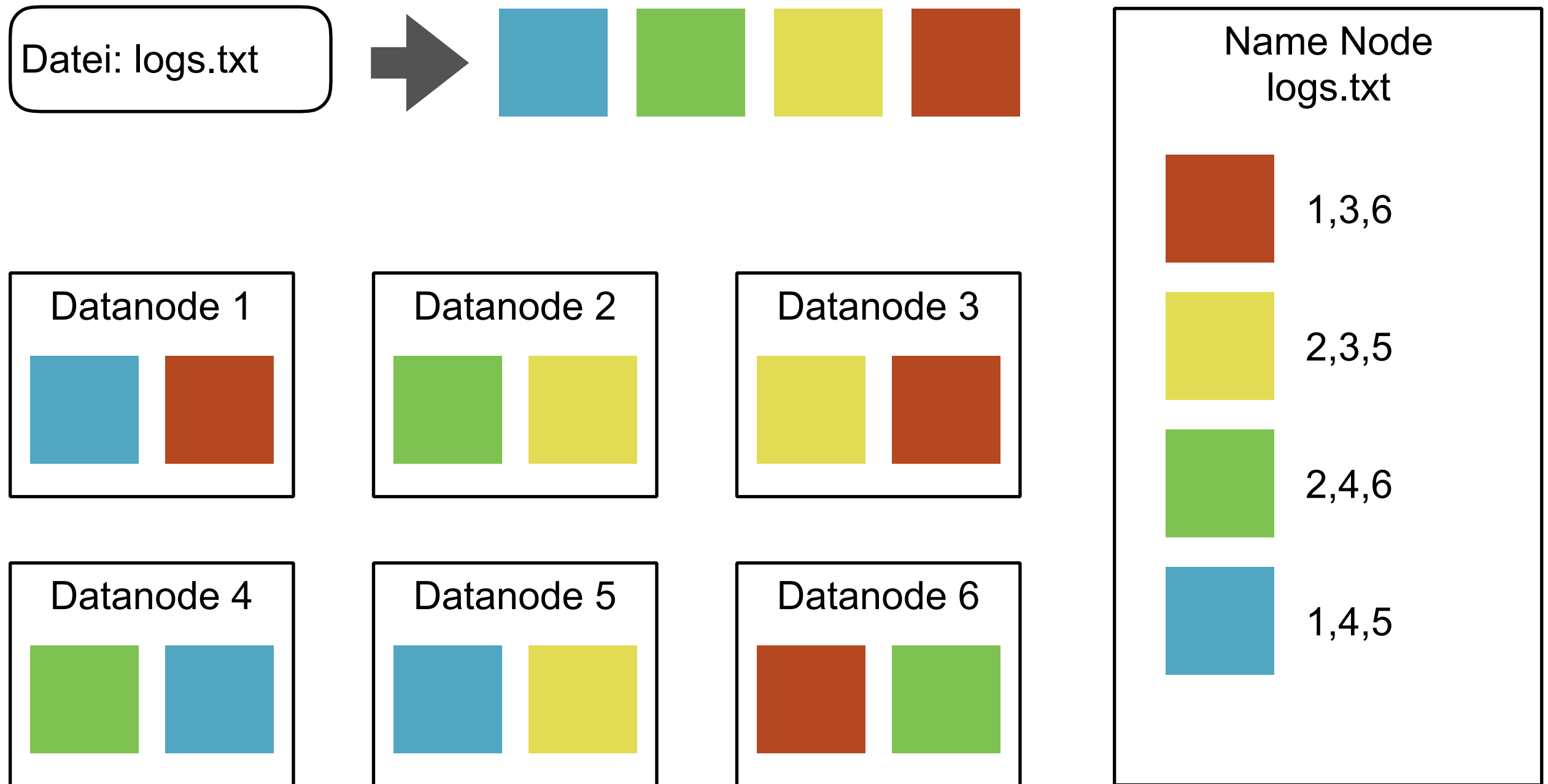


Prinzip von HDFS



Blöcke werden über
Datanodes
repliziert
(meist 3 Kopien)

Prinzip von HDFS

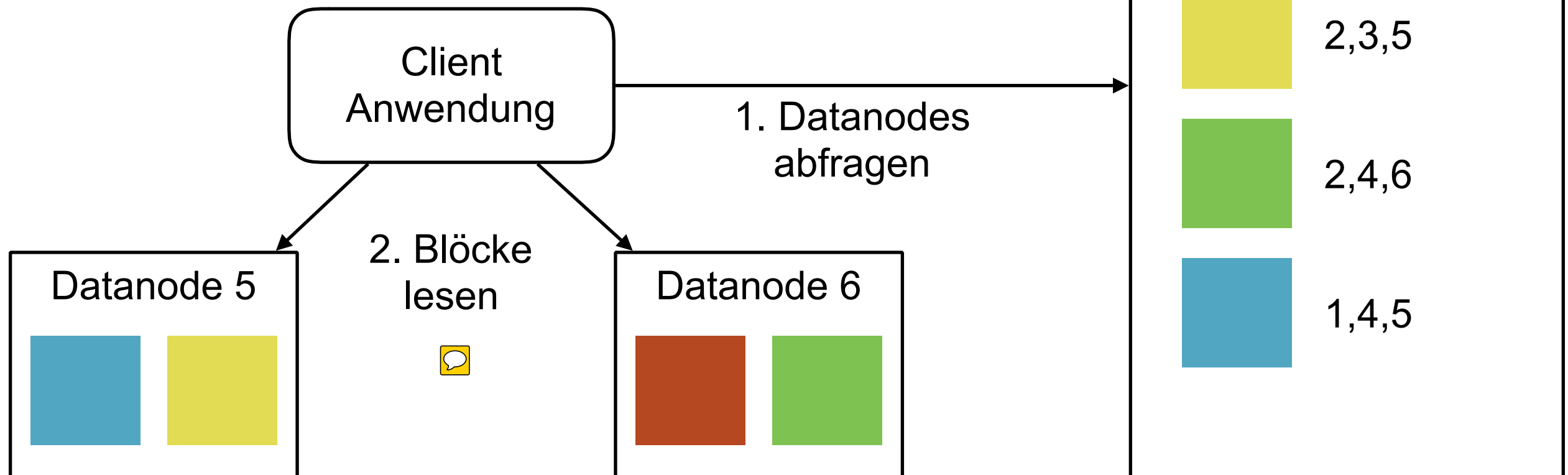


Namenode ist Namensdienst und steuert Replikation der Blöcke

Prinzip von HDFS

Verteilung von Dateien auf mehrere Knoten erlaubt Skalierung und parallele Verarbeitung

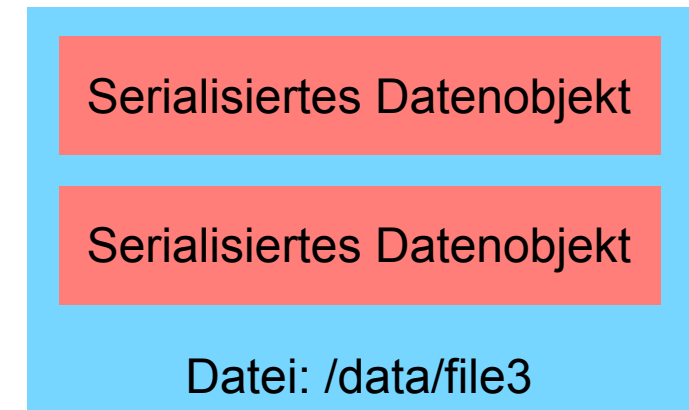
Replikation von Dateiblöcke über mehrere Knoten bietet Fehlertoleranz



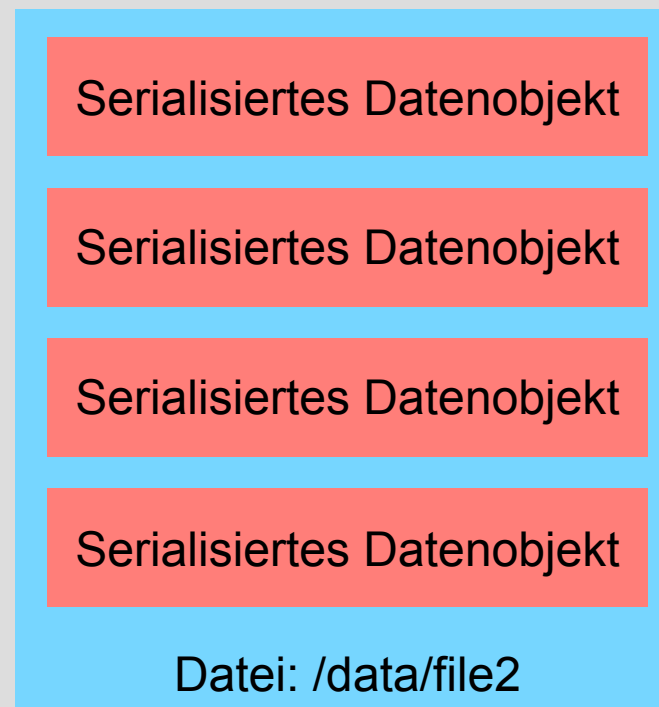
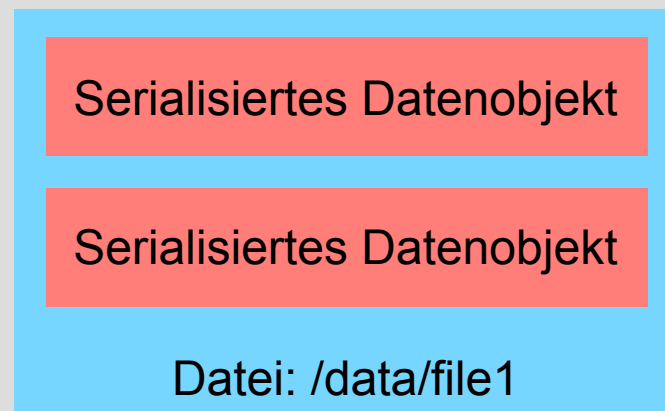
Master Datensatz in verteiltem Dateisystem speichern

Speichern durch schreiben serialisierter Fakten in eine **Datei**

Falls Dateien unveränderbar sind, neue Dateien schreiben um Fakten anzufügen



hochladen



Ordner: /data/



Anforderungen prüfen

	<i>Anforderung</i>	<i>Lösung in HDFS</i>
Write	Effizientes Anhängen neuer Daten	Neue Datei schreiben
Read	Skalierbarer Speicher	Datanodes hinzufügen
	Unterstützung für parallele Verarbeitung	Blöcke auf Datanodes verteilt
beide	Optimierung von Speicher- und Verarbeitungskosten	Dateiformat und -komprimierung frei wählbar
	Durchsetzbare Unveränderbarkeit	Dateirechte (read only)

Vertikale Partitionierung mit verteilten Dateisystemen

Vertikale Partitionierung des Master Datensatzes

- Nicht alle Berechnungen brauchen alle Daten
- **Partitionierung** von Daten kann die Performanz erhöhen
- Funktionen lesen nur relevante Daten

Vertikale Partitionierung von Daten auf verteilten Dateisystemen durch Sortierung von Daten in separate Ordner

Vertikales Partitionierungsschema für Anmeldedaten

Ordner: /logins/

Ordner: /logins/2018-03-25

File: /logins/2018-03-25/logins-2018-03-25.txt

```
alex 192.168.12.125 Sun Mar 25 22:33 - 22:46 (00:12)
bob 192.168.8.251 Sun Mar 25 21:04 - 21:28 (00:24)
...
```

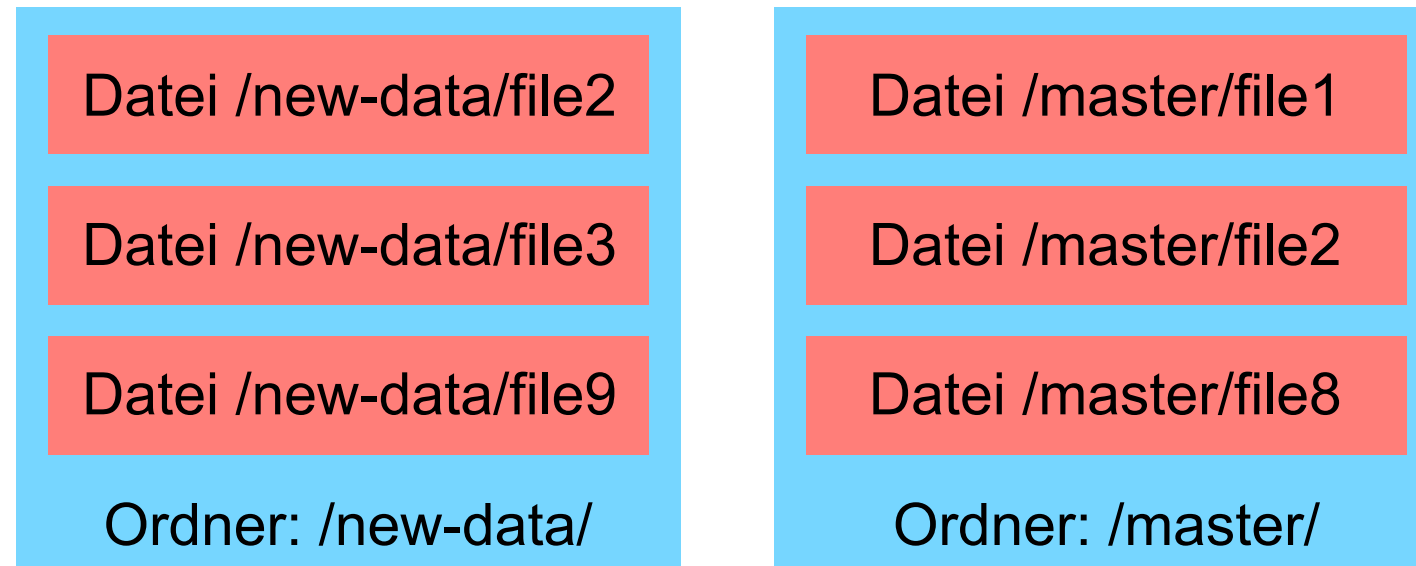
Ordner: /logins/2018-03-26

File: /logins/2018-03-26/logins-2018-03-26-part1.txt

File: /logins/2018-03-26/logins-2018-03-26-part2.txt

Sortierung von Informationen für jedes Datum in separaten Ordnern erlaubt es Funktionen, nur relevante Ordner auszuwählen

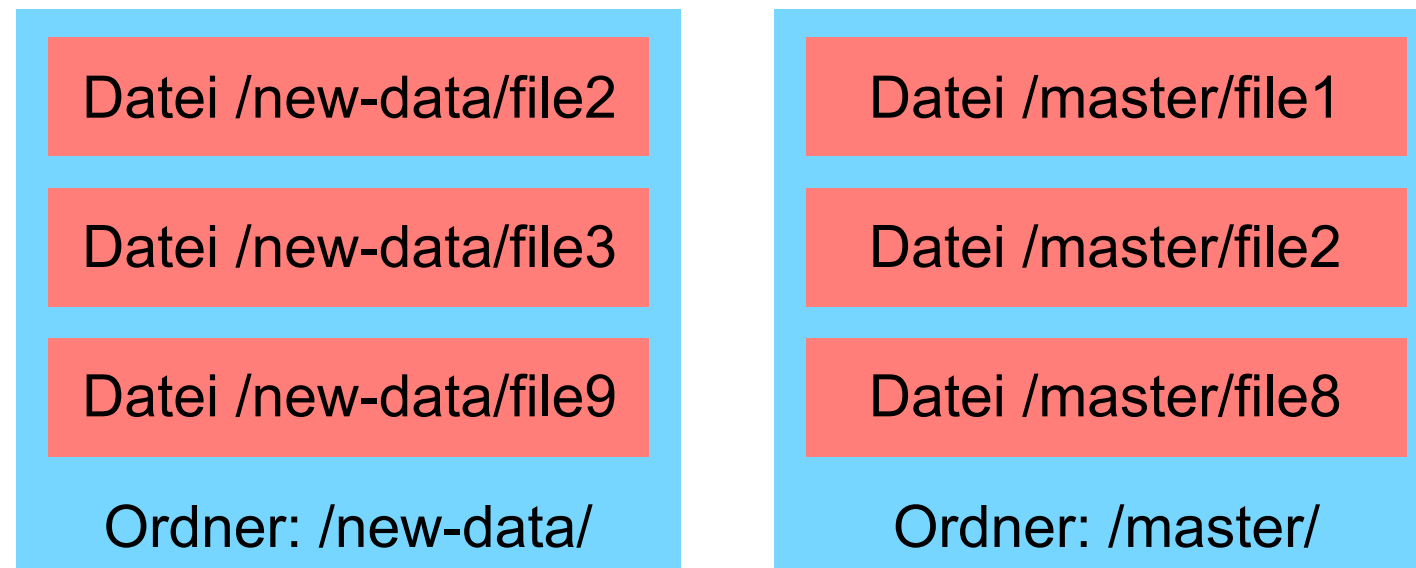
"Usability" der Dateisystem API



```
foreach file : "/new-data"  
  mv file "/master/"
```

Frage: welche Probleme können auftreten?

"Usability" der Dateisystem API

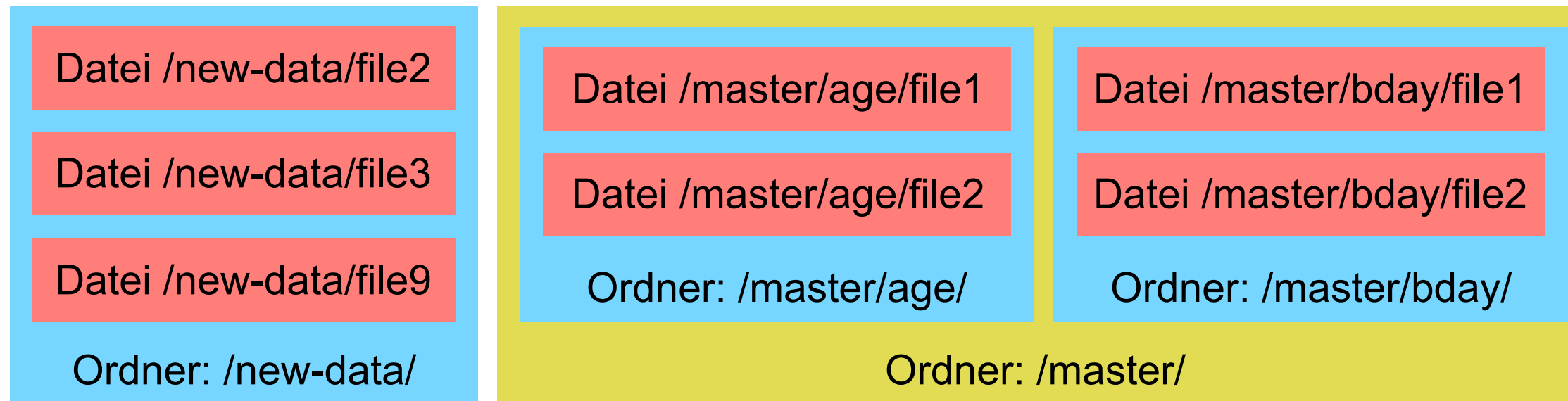


```
foreach file : "/new-data"
  mv file "/master/"
```

<i>Problem</i>	<i>Lösung</i>
<i>Gleiche Dateinamen</i>	<i>Dateien zufällig umbenennen</i>
<i>Unterschiedliche Formate</i>	<i>Daten lesen-deserialisieren-serialisieren-schreiben</i>

Frage: welche Probleme können auftreten?

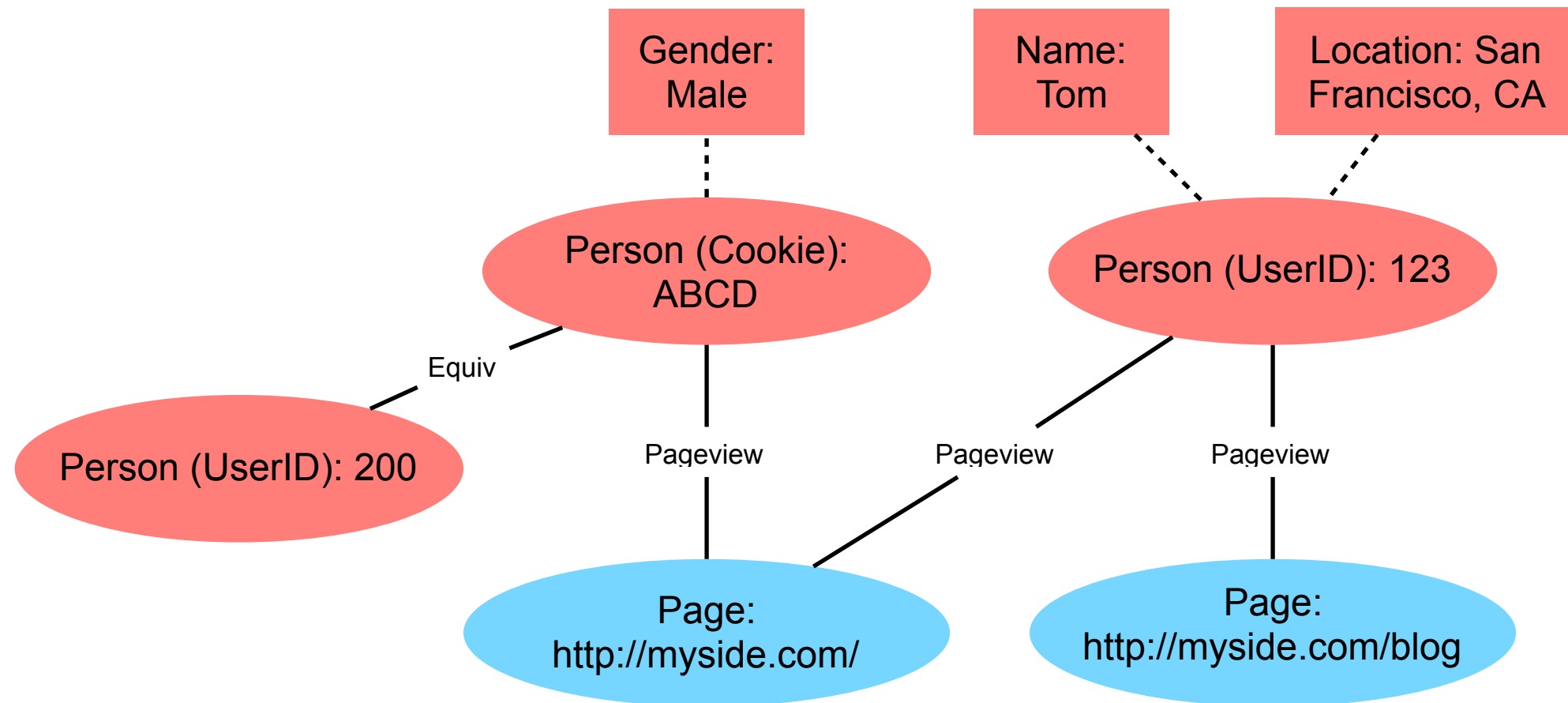
"Usability" der Dateisystem API



<i>Problem</i>	<i>Lösung</i>
<i>Dateien partitioniert</i>	<i>Daten neu partitionieren</i>

Einfache File-and-Folder API unpassend für Speicheraufgaben

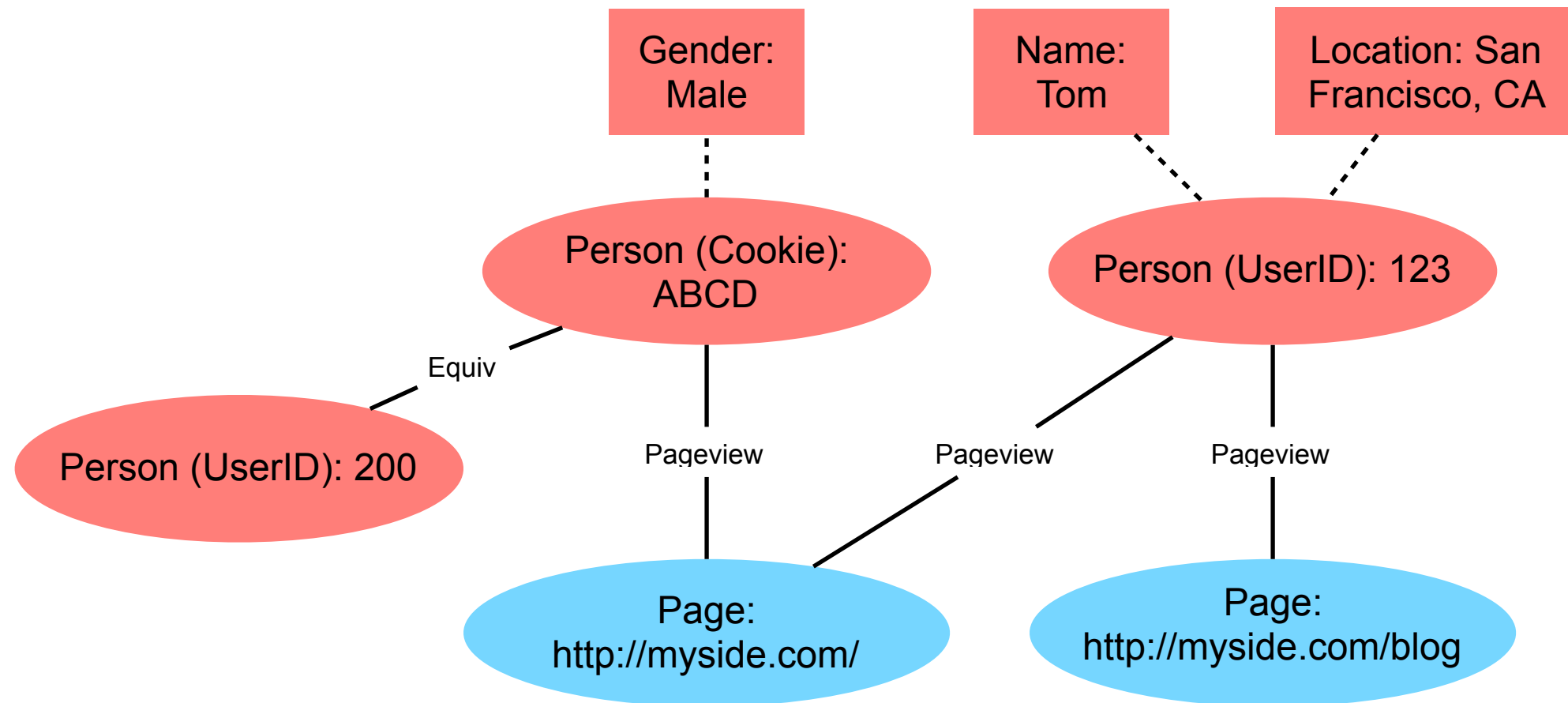
SuperWebAnalytics Graph Schema Beispiel



Jede Kante / Eigenschaft wird durch eigene DataUnit dargestellt

Frage: wie könnte ein Graph Schema partitioniert werden?

SuperWebAnalytics Graph Schema Beispiel



Jede Kante / Eigenschaft wird durch eigene DataUnit dargestellt

Natürliche vertikale Partitionierung im Graph Schema:
alle Kanten- und Eigenschaftstypen in separate Ordner



HDFS

Hadoop Distributed File System

HDFS in der Praxis: Shell Befehle

"**hadoop fs**" Hadoop-Shell-Befehle interagieren direkt mit HDFS

<http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HDFSCommands.html>

```
$ cat logins-2018-03-25.txt
alex      192.168.12.125   Sun Mar 25 22:33 - 22:46 (00:12)
bob       192.168.8.251    Sun Mar 25 21:04 - 21:28 (00:24)
charlie   192.168.12.82     Sun Mar 25 21:02 - 23:14 (02:12)
doug      192.168.8.13      Sun Mar 25 20:30 - 21:03 (00:33)
```

```
$ hadoop fs -mkdir /logins
```

```
$ hadoop fs - logins-2018-03-25.txt /logins
```

```
$ hadoop fs -ls -R /logins
-rw-r--r--    3 hdfs hadoop  175802352 2018-03-26 01:38
  /logins/logins-2018-03-25.txt
```

Hands-on: Eine vorkonfigurierte **virtuelle Maschine** erleichtert die erste Begegnung mit Hadoop. *Cloudera*, *Hortonworks* und *MapR* haben alle freie Images, z.B. <http://de.hortonworks.com/downloads/#sandbox>

Hands-on 2: Im LKIT (LI 137) gibt es ein Hadoop Cluster für Sie.

HDFS in der Praxis: Shell Befehle

"hadoop fsck" zeigt Meta-Informationen des HDFS

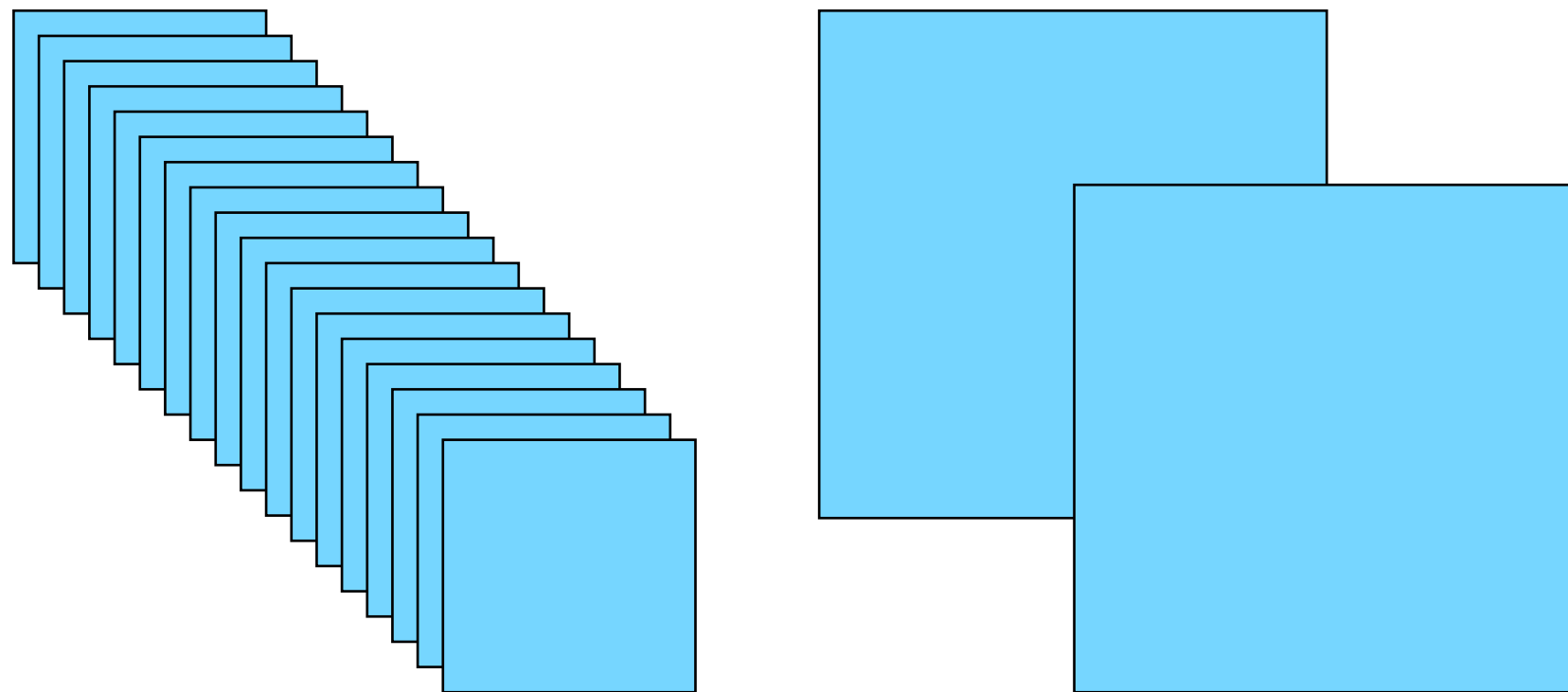
```
$ hadoop fs -cat /logins/logins-2018-03-25.txt
alex      192.168.12.125    Sun Mar 25 22:33 - 22:46 (00:12)
bob       192.168.8.251     Sun Mar 25 21:04 - 21:28 (00:24)
...

$ hadoop fsck /logins/logins-2018-03-25.txt -files -blocks -locations

/logins/logins-2018-03-25.txt 175802352 bytes, 2 block(s) :
OK
0. blk_-1821909382043065392_1523 len=134217728
  repl=3 [10.100.0.249:50010, 10.100.1.4:50010, 10.100.0.252:50010]
1. blk_2733341693279525583_1524 len=41584624
  repl=3 [10.100.0.255:50010, 10.100.1.2:50010, 10.100.1.5:50010]
```

HDFS in der Praxis: kleine Dateien

- Wenn Daten in *vielen kleinen Dateien* in HDFS gespeichert werden, wird die Rechenleistung von MapReduce erheblich beeinträchtigt
 - Jeder Block erzeugt einen **Task** mit gewissem Overhead



- Kleine Dateien konsolidieren; HDFS-API oder MapReduce Job

Storage Framework für HDFS

Abstraktion von Operationen auf der Datenmenge

Pail realisiert Abstraktionen für Speicheroperationen auf HDFS

- **Pail "Eimer"**: Ordner für Datensätze mit Metadaten
- Komfortfunktionen für **Anhängen** und **Konsolidieren**
- **Prüfung** der Datenformate (ggf. Neukodierung) und Partitionierung
- Operationen über **MapReduce** realisiert

<https://github.com/nathanmarz/dfs-datastores>

Daten in Pail schreiben



Erstellt Standard-Pail
im Verzeichnis

Erstelle Ausgabe-Stream
zu neuer Datei im Pail

```
public static void simpleIO()  
    throws IOException {  
    Pail pail = Pail.create("/tmp/mypail");  
    TypedRecordOutputStream os = pail.openWrite();  
    os.writeObject(new byte[] {1, 2, 3});  
    os.writeObject(new byte[] {1, 2, 3, 4});  
    os.writeObject(new byte[] {1, 2, 3, 4, 5});  
    os.close();  
}
```

Schließe Datei

Pail ohne Metadaten
speichert Byte-Arrays

Daten in Pail schreiben

Datensätze sind in Pail
Dateien gespeichert

```
root:/ $ ls /tmp/mypail  
f2fa3af0-5592-43e0-a29c-fb6b056af8a0.pailfile  
pail.meta
```

Metadata beschreibt Inhalt und
Struktur des Pail (siehe unten)

```
root:/ $ cat /tmp/mypail/pail.meta  
---  
format: SequenceFile  
args: {}
```

Argumente beschreiben den Pail
Inhalt; leere Map bedeutet
unkomprimierte Byte-Arrays

Format der Dateien im Pail;
Standard-Pail speichert Daten in Key/
Value-Paaren in Hadoop SequenceFiles

Operationen auf Pails: absorb und consolidate

```
import java.io.IOException;
import backtype.hadoop.pail.Pail;

public class PailMove {

    public static void mergeData(String masterDir, String updDir)
        throws IOException
    {
        Pail target = new Pail(masterDir); // master directory
        Pail source = new Pail(updDir); // update directory
        target.absorb(source);
        target consolidate();
    }
}
```

Instanziiere Pail Objekte
für vorhandene Ordner

Pail Dateien auf einheitliche
Größe bringen

Übernehme Daten aus Pail
source in Pail target und
passe sie bei Bedarf an

Eigene Pails konfigurieren

- **PailStructure<T>** Interface implementieren
 - Definiert *Serialisierung* und *Partitionierung*
- **Serialisierung**
 - **serialize(...)** wandelt Objekt in Bytes; **deserialize(...)** zurück
 - Hier kann auch Thrift verwendet werden
- **Partitionierung**
 - **getTarget(...)** liefert relativen Pfad der Partition für ein Objekt
 - **isValidTarget(...)** prüft Pfadname syntaktisch

```
public interface PailStructure<T> extends Serializable {  
    public boolean isValidTarget(String... dirs);  
    public T deserialize(byte[] serialized);  
    public byte[] serialize(T object);  
    public List<String> getTarget(T object);  
    public Class getType();  
}
```

Mehr Pail im Labor

Anforderungen prüfen

	<i>Anforderung</i>	<i>Lösung in Pail</i>
Write	<i>Effizientes Anhängen neuer Daten</i>	<i>Eigene gesicherte Operation</i>
Read	<i>Skalierbarer Speicher</i>	<i>Konsolidierung entlastet den Namenode</i>
	<i>Unterstützung für parallele Verarbeitung</i>	<i>Konsolidierung reduziert MapReduce Tasks</i>
beide	<i>Optimierung von Speicher- und Verarbeitungskosten</i>	<i>Anpassbare Dateiformate und Komprimierung</i>
	<i>Durchsetzbare Unveränderbarkeit</i>	<i>Berechtigungen auf HDFS-Ebene</i>

Zusammenfassung und Ausblick

Heute haben wir...

- ...Speicheranforderungen für den Master Datenmenge untersucht
- ...Verteilte Dateisysteme als Speicher verwendet
- ...Vertikale Partitionierung diskutiert
- ...HDFS kennengelernt
- ...auf HDFS mit **Pail** gearbeitet

Als nächstes wollen wir...

- **Views** aus dem Master Datensatz ableiten
- **Batch-Verarbeitung** mit **MapReduce** durchführen
- **Vorher:** *Datenmodellierung* und *Speicherung* praktisch im **Labor** ausprobieren

Literatur und Links

- Nathan Marz, James Warren, "Big Data: Principles and best practices of scalable realtime data systems", Manning, 2015, Kapitel 4+5
- Tom White, "Hadoop: the definitive guide: storage and analysis at internet scale", 4. ed., O'Reilly, 2015, ISBN: 978-1-491-90163-2
- Michael Frampton, "Big Data Made Easy: A Working Guide to the Complete Hadoop Toolset", Apress, 2015