

Big-Data-Technologien

Kapitel 3: Hadoop – HDFS

Hochschule Trier
Prof. Dr. Christoph Schmitz

Überblick

- HDFS als verteiltes Dateisystem
- Lesen, Schreiben, Umgang mit Ausfällen
- Praktische Verwendung
- Datentypen und Teilbarkeit von Dateien
- Sicherheit

Google 2003

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google*

ABSTRACT

We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.

While sharing many of the same goals as previous distributed file systems, our design has been driven by observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system assumptions. This has led us to reexamine traditional choices and explore radically different design points.

The file system has successfully met our storage needs. It is widely deployed within Google as the storage platform for the generation and processing of data used by our service as well as research and development efforts that require large data sets. The largest cluster to date provides hun-

1. INTRODUCTION

We have designed and implemented the Google File System (GFS) to meet the rapidly growing demands of Google's data processing needs. GFS shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability. However, its design has been driven by key observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system design assumptions. We have reexamined traditional choices and explored radically different points in the design space.

First, component failures are the norm rather than the exception. The file system consists of hundreds or even thousands of storage machines built from inexpensive commodity parts and is accessed by a comparable number of client machines. The quantity and quality of the components virtually guarantee that some are not functional at

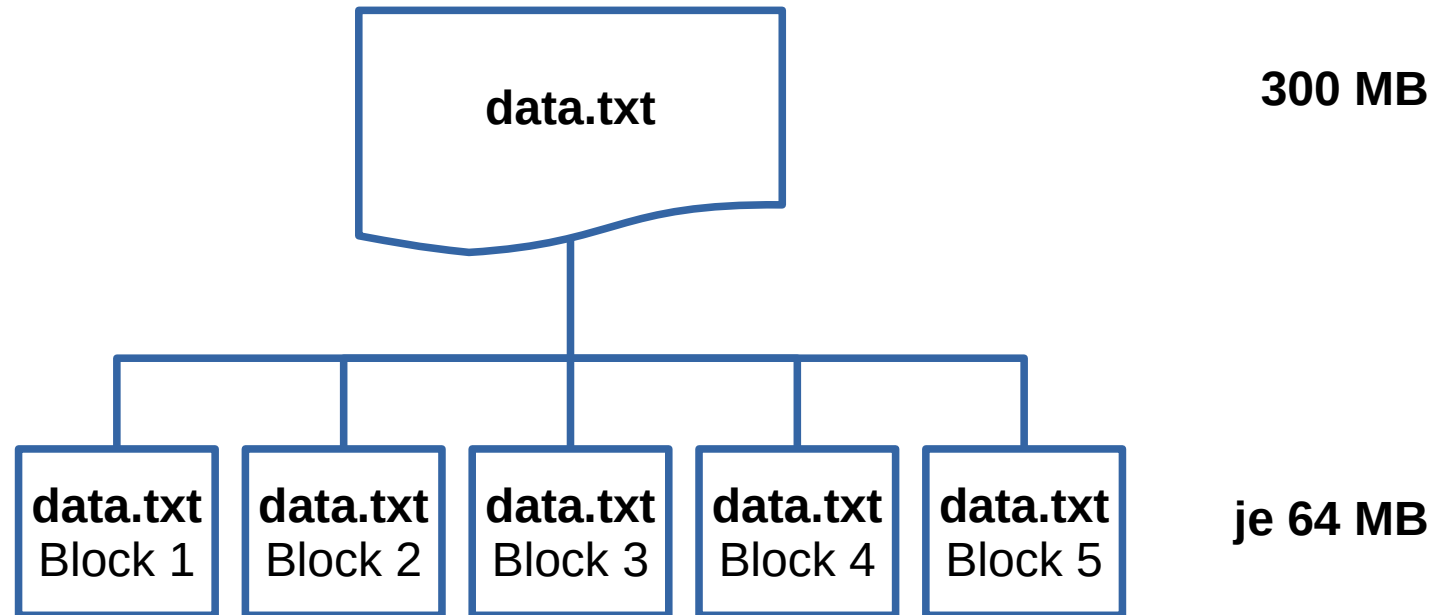
Hadoop HDFS

- **H**adoop **D**istributed **F**ile **S**ystem
- Open-Source-Variante des Google File System
- **Verteiltes Dateisystem** für sehr große Datenbestände
- Ziele
 - Sehr **große Dateien** (GB, TB, ...)
 - Schneller **sequenzieller Zugriff**
 - Unterstützt **verteilte Berechnungen**
 - Billige Hardware → **Robustheit**
- Läuft im **User Space**

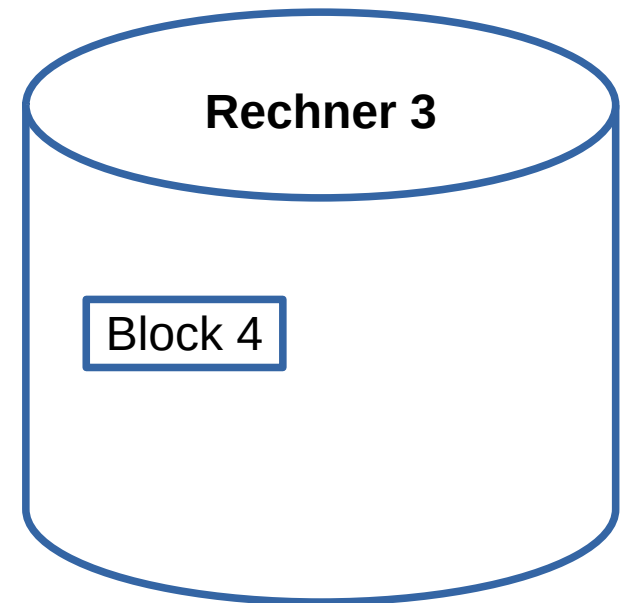
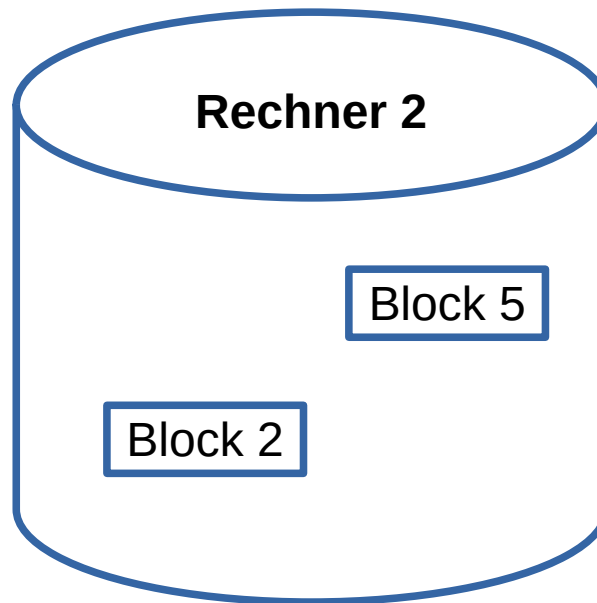
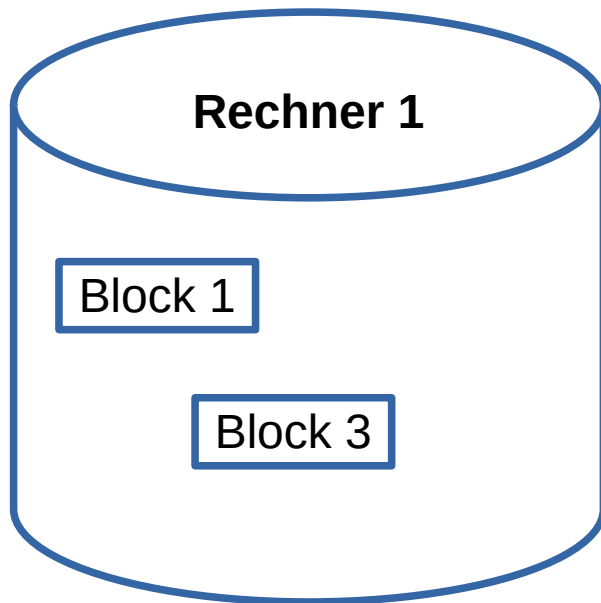
Hadoop HDFS ist nicht...

- Nicht **POSIX** (open, read, write, close)
- Kein „normales“ Dateisystem wie NTFS, ext4, zfs, ...
- Nicht geeignet für
 - **wahlfreien Zugriff** (*random access*)
 - viele **kleine Dateien**
 - **Änderungsoperationen**

Grundidee



Grundidee



Grundidee

data.txt

Rechner 1

Block 1

Block 2

Block 3

Rechner 2

Block 3

Block 5

Block 2

Block 4

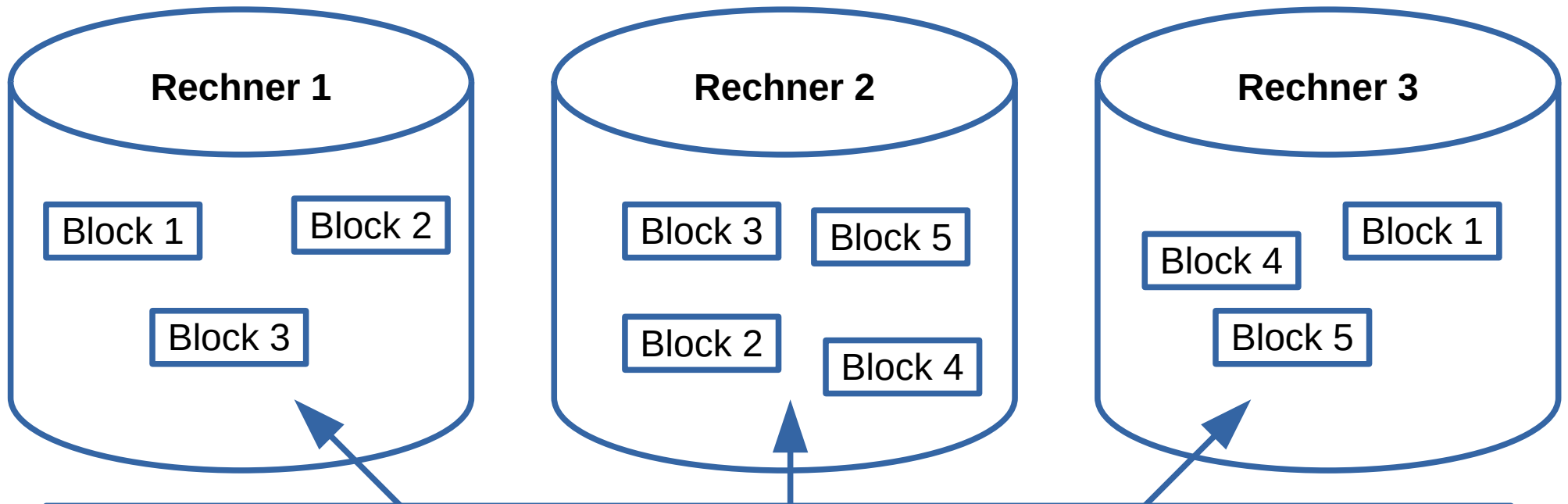
Rechner 3

Block 4

Block 1

Block 5

Grundidee



Namensdienst (NameNode)

data.txt: [5 Blöcke: (R1, R3), (R1, R2), ..., (R2, R3)]
moredata.txt: [3 Blöcke: (R1, R3), (R2, R3), (R1, R2)]

HDFS: Ausfälle sind die Regel

- Beispiel:

Seagate Enterprise NAS HDD

Annualized Failure Rate = 0.63%

- 1.000 HDD im Rechenzentrum

Erwartete Ausfälle: 6.3

$$P(\text{Alle HDD OK}) = (1 - 0.0063)^{1000} = 0.18\%$$

$$P(\geq 5 \text{ Ausfälle}) = 75\%$$

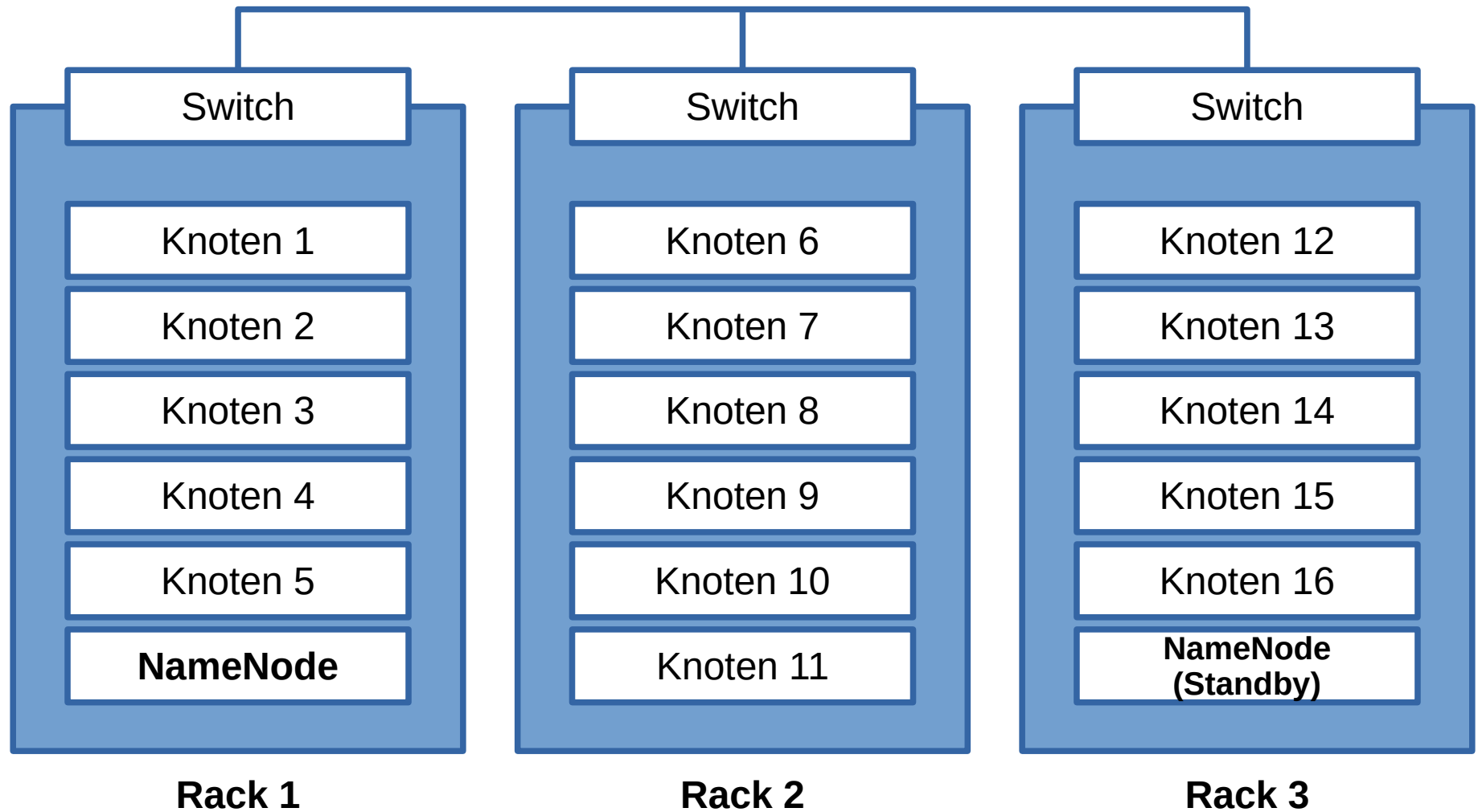
$$P(\geq 10 \text{ Ausfälle}) = 10\%$$

$$P(\text{Block X kaputt}) = 0.0063^3 = 0.000025\%$$

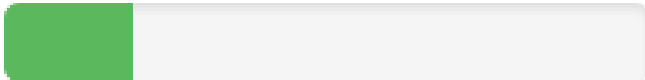
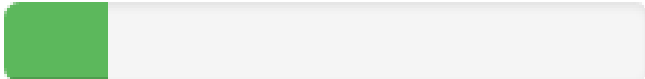
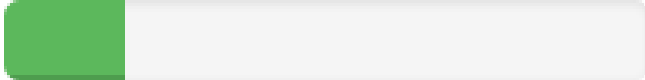
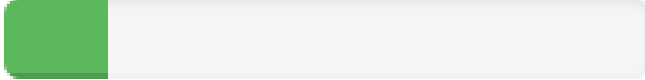
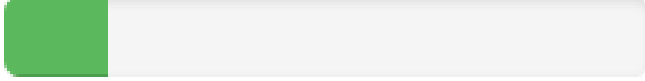
Wo liegen die Blöcke?

- Erste Kopie nah beim Client
- Zweite Kopie in anderem *Rack* (Schrank)
- Dritte Kopie im gleichen Rack wie die zweite
- „**Rack Awareness**“
- Blöcke werden im Hintergrund **balanciert**

Rack Awareness



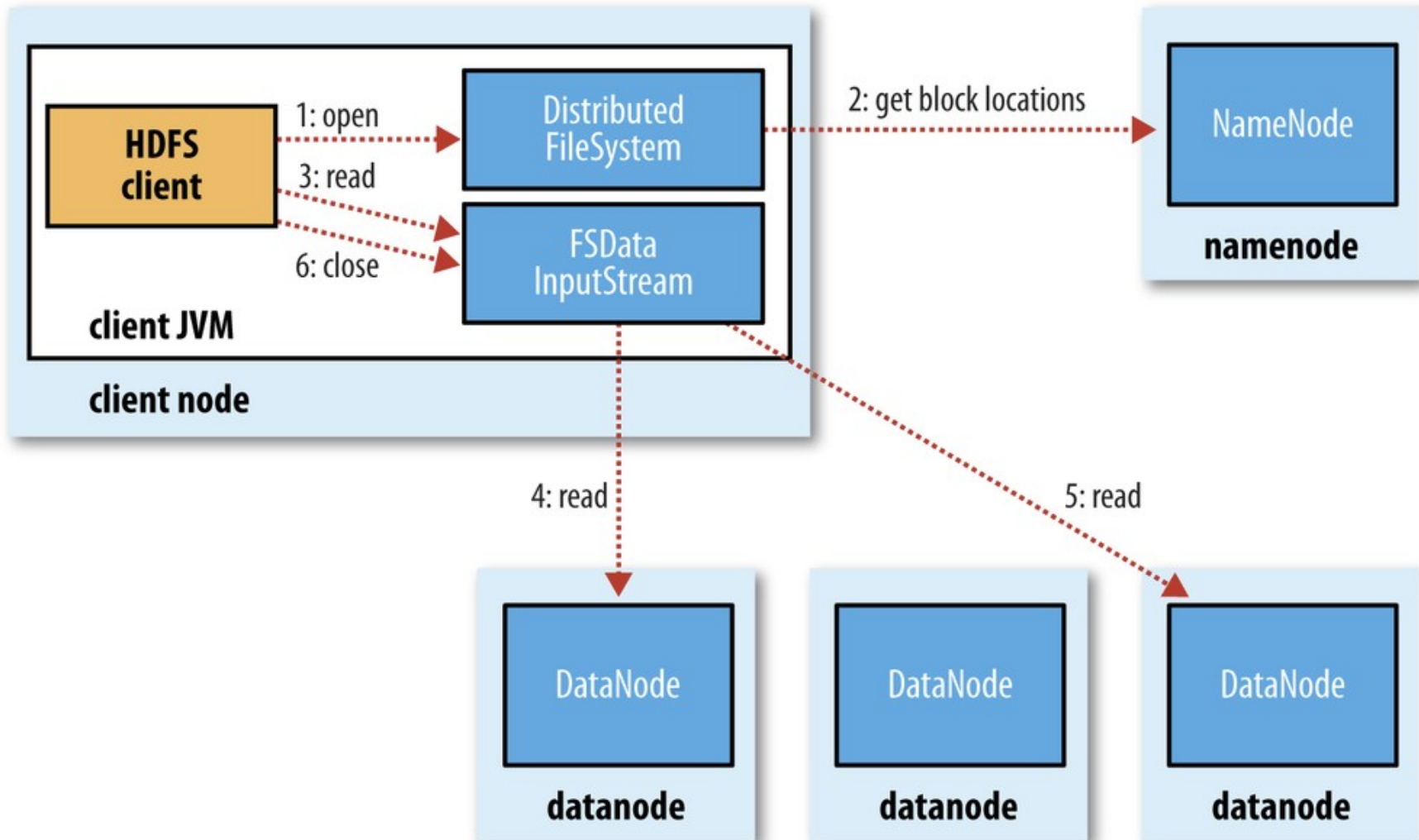
Auslastung balancieren

Node	Capacity
✓ infbdt01.fh-trier.de:1019 (143.93.53.142:1019)	60.18 GB 
✓ infbdt02.fh-trier.de:1019 (143.93.53.143:1019)	60.18 GB 
✓ infbdt03.fh-trier.de:1019 (143.93.53.144:1019)	60.18 GB 
✓ infbdt04.fh-trier.de:1019 (143.93.53.145:1019)	60.18 GB 
✓ infbdt05.fh-trier.de:1019 (143.93.53.146:1019)	60.18 GB 

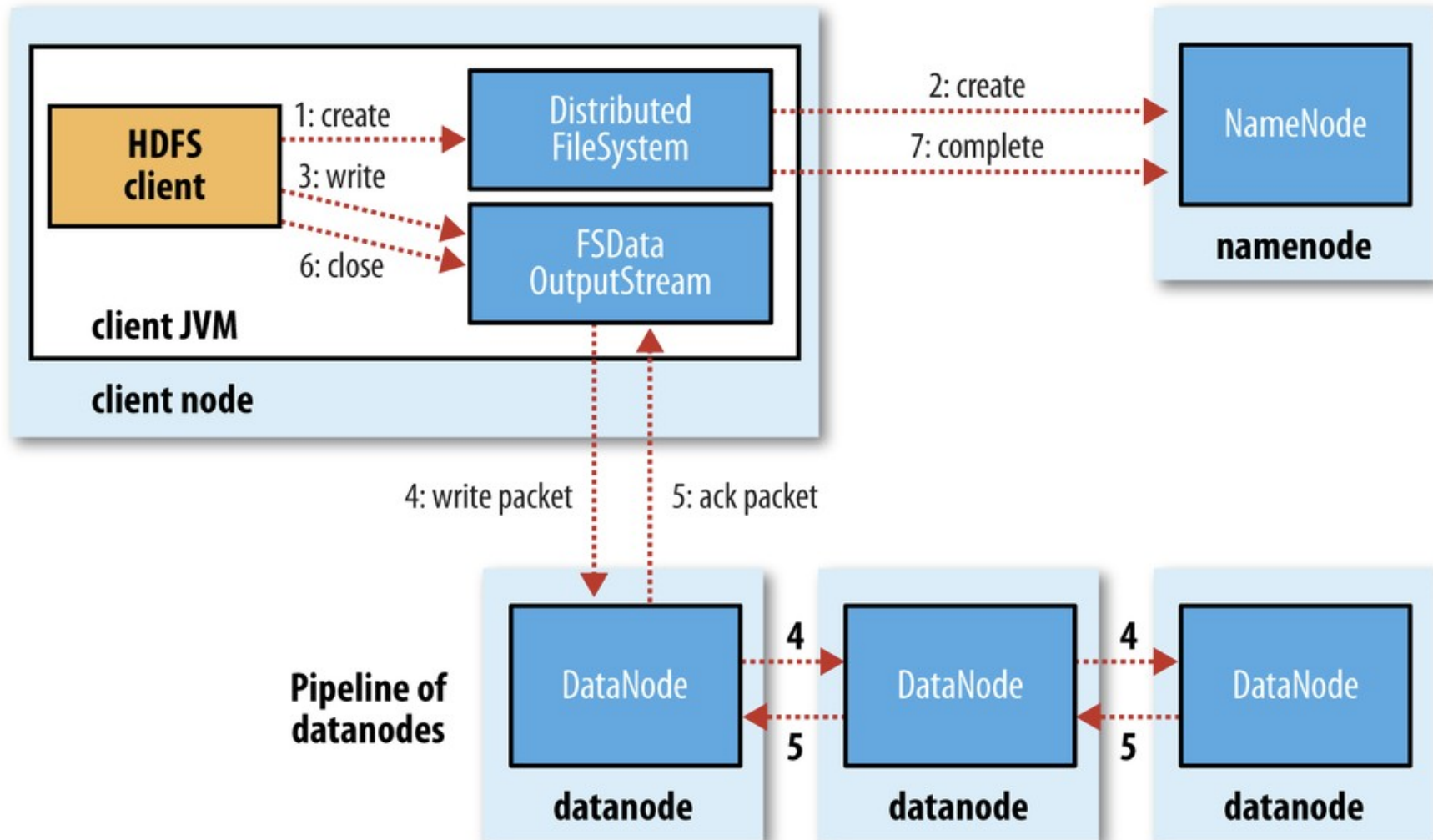
HDFS: Dienste

- Zwei wesentliche Dienste:
 - **DataNode**: verwaltet Nutzdaten auf jedem Knoten
 - **NameNode**: Namensdienst, Metadaten
- Weitere Dienste
 - **JournalNode**: Änderungshistorie für HA-Setup
 - **Secondary NameNode**: Änderungshistorie ohne HA
 - **Balancer**: Gleicht Auslastung der Knoten an
 - ...

Lesen vom HDFS



Schreiben ins HDFS



Und in der Praxis?

```
Configuration conf = new Configuration();
conf.set("fs.defaultFS", "hdfs://infbd01.fh-trier.de/");

try (FileSystem fs = FileSystem.get(conf);
    BufferedReader reader = new BufferedReader(
        new InputStreamReader(
            fs.open(new Path("somefile.txt"))));) {

    for (String line = reader.readLine();
        line != null;
        line = reader.readLine()) {
        System.out.println(line);
    }
}
```

HDFS, Pfade, URLs, Konfiguration

```
Configuration conf = new Configuration();
conf.set("fs.defaultFS", "hdfs://infbd01.fh-trier.de/");

try (FileSystem fs = FileSystem.get(conf);
    BufferedReader reader = new BufferedReader(
        new InputStreamReader(
            fs.open(new Path("somefile.txt"))));) {

    for (String line = reader.readLine();
        line != null;
        line = reader.readLine()) {
        System.out.println(line);
    }
}
```

/etc/hadoop/conf

core-site.xml

HDFS, Pfade, URLs, Konfiguration

- `fs.defaultFS = hdfs://infbd01.fh-trier.de`

`somefile.txt`

→ `hdfs://infbd01.fh-trier.de/user/bigdataXYZ/somefile.txt`

`/tmp/somefile.txt`

→ `hdfs://infbd01.fh-trier.de/tmp/somefile.txt`

`file:///somefile.txt`

→ `file:///somefile.txt`

`hdfs:///tmp/somefile.txt`

→ `hdfs://infbd01.fh-trier.de/tmp/somefile.txt`

- `fs.defaultFS = file:///` (oder leer)

`somefile.txt`

→ `file://$PWD/somefile.txt`

`/tmp/somefile.txt`

→ `file:///tmp/somefile.txt`

`file:///somefile.txt`

→ `file:///somefile.txt`

`hdfs:///tmp/somefile.txt`

→ Fehlermeldung: Namenode nicht zu bestimmen!

Und in der Praxis?

```
$ hdfs dfs -ls
Found 2 items
drwx-----   - bigdata200 bigdata          0 2017-01-27 01:00 .Trash
-rw-r--r--    3 bigdata200 bigdata      588895 2017-02-01 11:15 somefile.txt

$ hdfs dfs -cat somefile.txt | head -5
1
2
3
4
5

$ hdfs dfs -copyToLocal somefile.txt

$ ls
somefile.txt
```

Und in der Praxis?

Browse Directory

/user/bigdata200

Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	bigdata200	bigdata	300 MB	2.2.2017, 09:32:22	3	128 MB	somefile.txt

[Download](#)

Und in der Praxis?

Block information --

Block 2

Block ID: 1073742007

Block Pool ID: BP-748816717-127.0.1.1-1485188107556

Generation Stamp: 1184

Size: 46137344

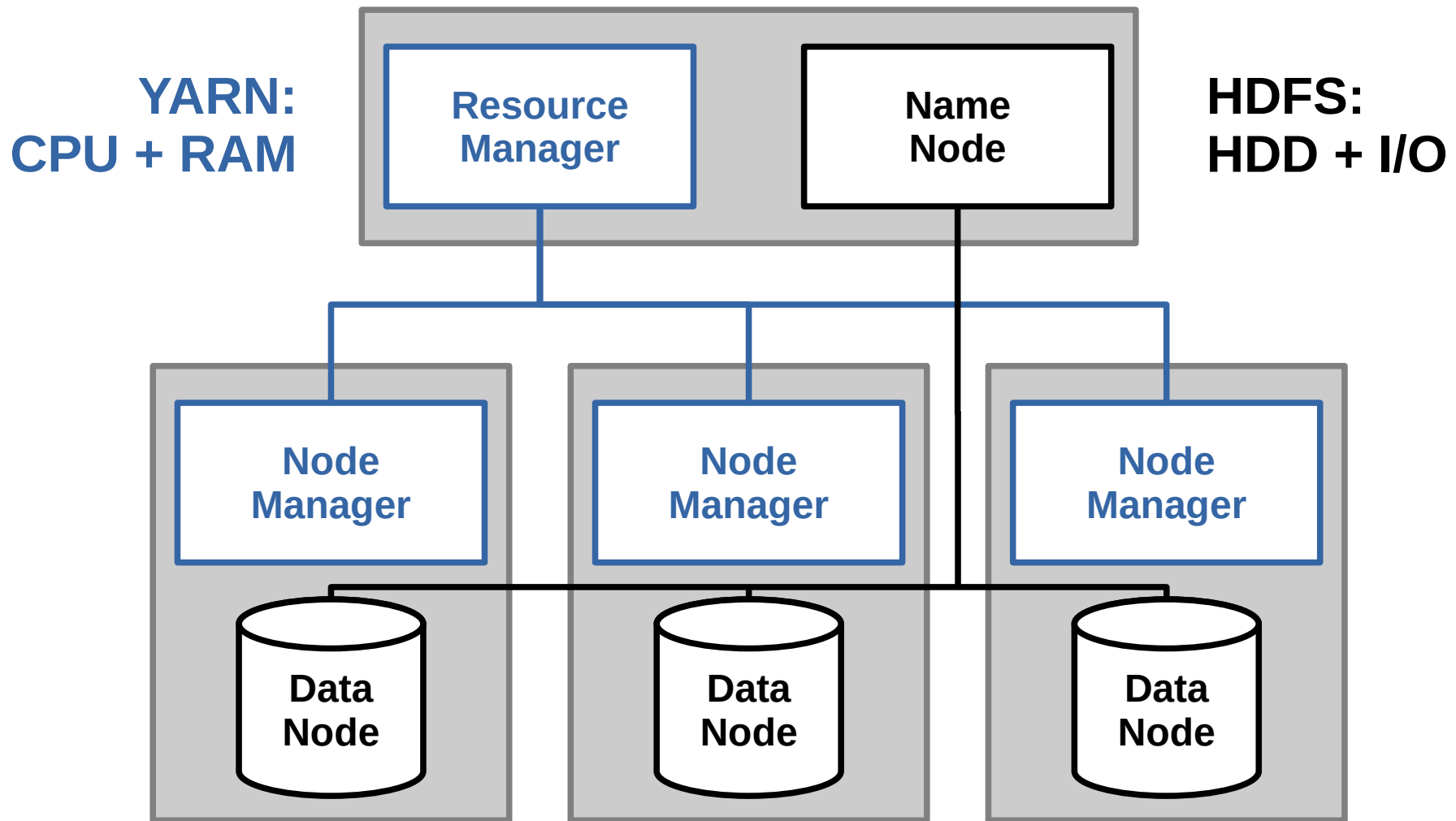
Availability:

- infbdt02.fh-trier.de
- infbdt01.fh-trier.de
- infbdt04.fh-trier.de

Verwendung von HDFS

- Verteilte Berechnung
 - Frameworks berücksichtigen Verteilung: **Lokalität**
 - Daten müssen sich sinnvoll **zerteilen** lassen
- Bandbreite skaliert praktisch linear
- Höhere Latenz als normales FS
- Wie kommen die Daten ins HDFS?
 - **Data Ingestion**
- Möglichst „im System“ bleiben

Lokalität ausnutzen



Dateien im HDFS

- Große Dateien bevorzugt
- Typischerweise komprimiert
- Teilbar oder nicht?
- Strukturiert?

Teilbar oder nicht?

- Teilbar

- Textdateien
- CSV
- Blockweise komprimiert

Lorem ipsum dolor
sit amet,
consectetur
adipiscing elit, sed
do eiusmod tempor
incididunt ut labore
et dolore magna
aliqua.

- Nicht teilbar

- Binärformate
- Multimediaformate
- Im Ganzen komprimiert
- XML
- ...

<data>

<person

name="alice"

age="23"

hair="red"

</person>

</data>



Lösung 1: CompressionCodecs

org.apache.hadoop.io.compress

Interface CompressionCodec

All Known Subinterfaces:

DirectDecompressionCodec, SplittableCompressionCodec

All Known Implementing Classes:

BZip2Codec, DefaultCodec, GzipCodec

```
@InterfaceAudience.Public  
@InterfaceStability.Evolving  
public interface CompressionCodec
```

This class encapsulates a streaming compression/decompression pair.

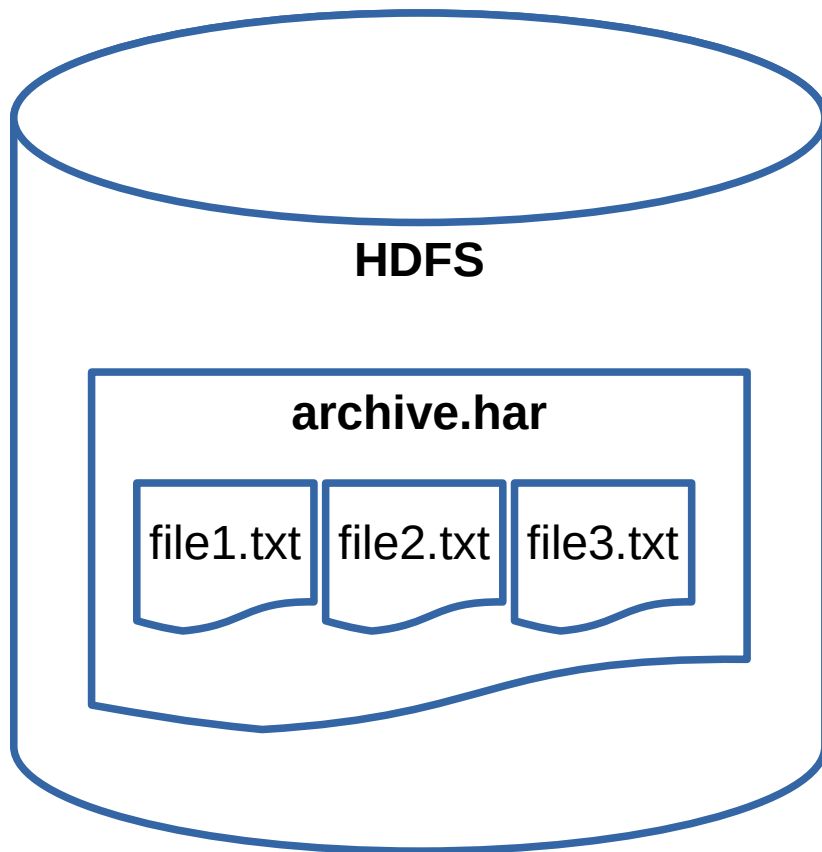
Lösung 2: SequenceFiles

- Strukturiert: Name-Wert
- Blockweise komprimierbar

Key	Value
234	Lorem ipsum
123	Foo
345	Bar
456	Baz
789	Quux

- Teilbar

Lösung 3: Hadoop Archives

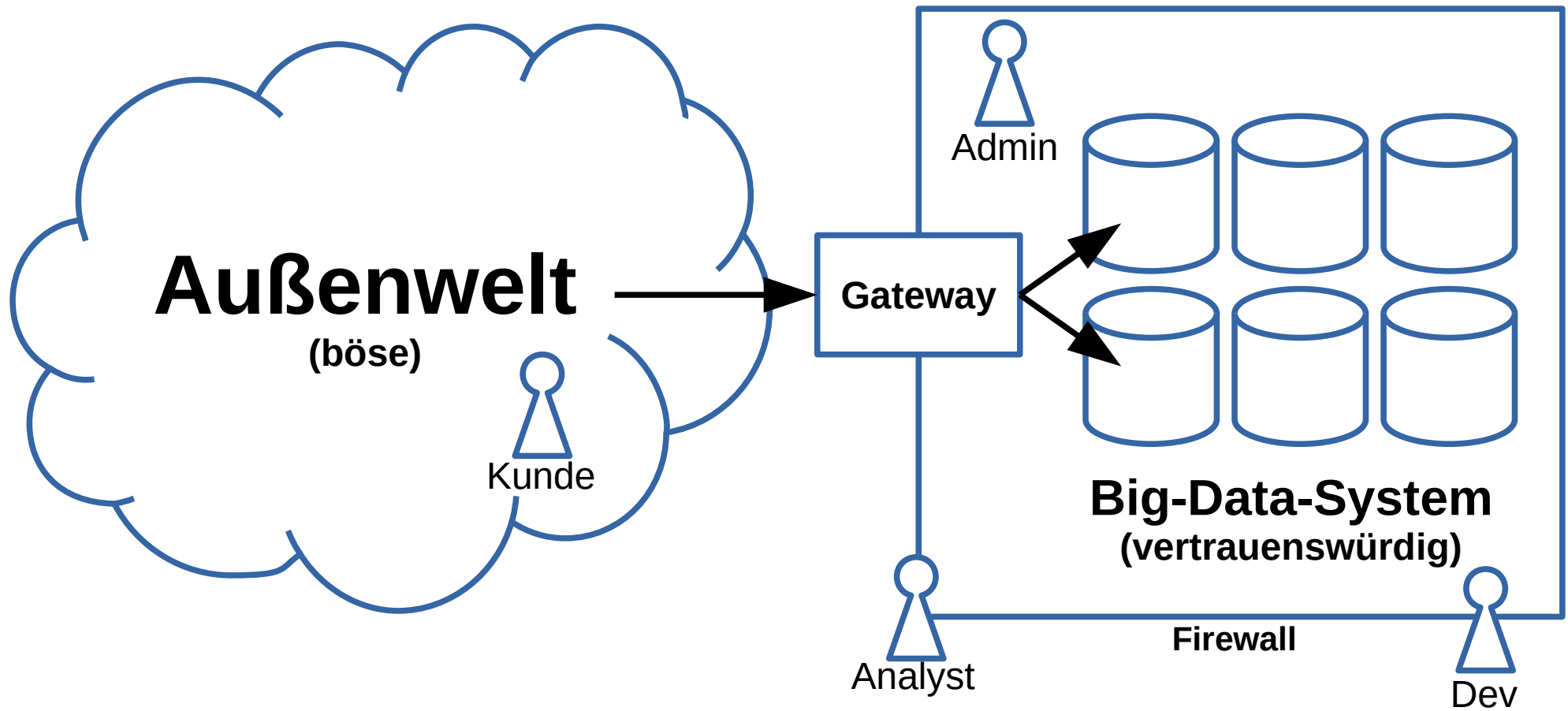


- Nur **ein** Verzeichniseintrag
- Interne Dateisystem-Struktur
- Dateien im Archiv können referenziert werden

Datensicherheit

- Wie schützt man **sensible Daten** in Hadoop?
- Wie identifiziert man Benutzer?
→ **Authentifizierung**
- Wie regelt man den Zugriff?
→ **Autorisierung**
- Was dürfen...
 - Endnutzer?
 - Entwickler?
 - Administratoren?
- Sind **Audits** möglich?

Triviale Lösung: „Perimeter Security“



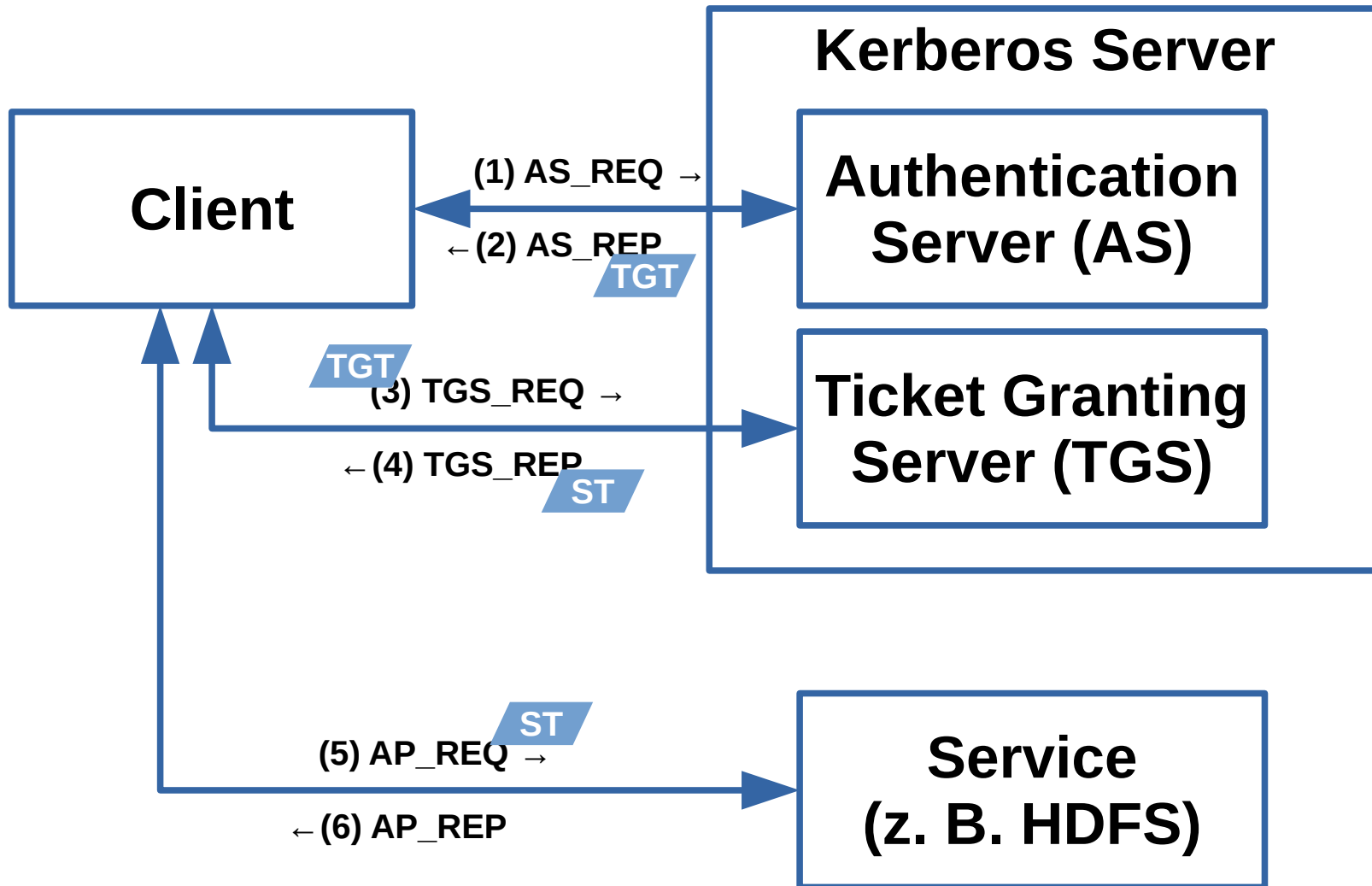
Datensicherheit

- **Autorisierung** im HDFS über Unix-artige **Permissions**:

```
drwxr-xr-x  user group  someDirectory
-rw-r--r--  user group  someFile
```

- Standard ist **Client-seitige „Authentifizierung“**
\$ whoami
\$ hadoop -Dhadoop.job.ugi=someuser ...
- Ansonsten: **Kerberos**

Crashkurs Kerberos



Authentifizierung/Authorisierung: Wer und wo?

- Endbenutzer authentifizieren sich bei den Diensten über Kerberos:

```
$ kinit -p bigdata200@BDT.FH-TRIER.DE  
Password for bigdata200@BDT.FH-TRIER.DE: *****
```

```
$ klist  
Ticket cache: FILE:/tmp/krb5cc_1001  
Default principal: bigdata200@BDT.FH-TRIER.DE
```

Valid starting	Expires	Service principal
15.02.2017 16:26:30	16.02.2017 16:26:30	krbtgt/...@BDT.FH-TRIER.DE

Authentifizierung/Authorisierung: Wer und wo?

- Dienste authentifizieren sich **untereinander** auch über Kerberos
- Passwörter? → **Keytabs**

```
$ ls -l /etc/security/keytabs
```

```
-r----- 1 hdfs      hadoop 408 Jan 24 17:13 dn.service.keytab  
-r--r----- 1 hdfs      hadoop 328 Jan 24 17:13 hdfs.headless.keytab  
-r----- 1 yarn      hadoop 408 Jan 24 17:13 nm.service.keytab  
[...]
```

- Keytabs müssen mit Betriebssystem-Mitteln abgesichert sein!

Verschlüsselung

- ... von **Datentransfers**: „data in flight“
 - Authentifiziert über Kerberos
 - Verschlüsselt mit TLS/SSL
- ... der **abgelegten Daten**: „data at rest“
 - Proprietäre Lösungen
 - Eingebaut in HDFS...
 - ... oder auf Betriebssystem-Ebene

Vermischtes...

- **HDFS Federation:** mehrere NameNodes zur Lastverteilung
- **Block Caching** auf den DataNodes
- **High Availability (HA):** Hochverfügbare NameNodes

Zusammenfassung: HDFS

- **Verteiltes Dateisystem** für das Hadoop-Ökosystem
- Optimiert für **große Dateien**
 - Einmaliges Schreiben
 - Sequenzielles Lesen
- **Redundante** Speicherung in Blöcken
- Optimiert für hohe **Bandbreite**
- Ausnutzen der **Lokalität** bei Berechnungen