INTRODUCTION TO HDFS

Introduction to HDFS

- **Rationale**
- Block Storage
- HDFS Daemons
- Fault Tolerance
- HDFS Clients
- HDFS Write Path
- HDFS Read Path

HDFS Rationale

- HDFS is a **fault-tolerant**, **distributed file system** that provides **high-throughput** access to **large amounts** of data
- In HDFS, Big Data is **partitioned** into blocks and **replicated** for fault-tolerance
- HDFS manages the distributed storage across **a cluster** of machines (connected via network), and provides an interface performing supported **filesystem operations**, **abstracting the complexities** of underlying distributed system away

HDFS Design Goals

- HDFS is appropriate for Big Data stored in **very large files**
- HDFS is optimized for a **write-once**, **read-many** data processing
- HDFS can run on **commodity hardware**
- HDFS is **resistant** to the failures

HDFS Design Goals

- HDFS is **not** for **low-latency**, **record-level** data access
  - It's for high throughput when accessing a large portion of the data
- HDFS performs **poorly** when data is stored in **many small files**
  - This is because the master node of the HDFS holds the FS metadata in memory
- HDFS does **not** allow **multiple writers** and **arbitrary modifications** on a file

analyticscenter

Introduction to HDFS

- Rationale
- **Block Storage**
- HDFS Daemons
- Fault Tolerance
- HDFS Clients
- HDFS Write Path
- HDFS Read Path

HDFS Blocks

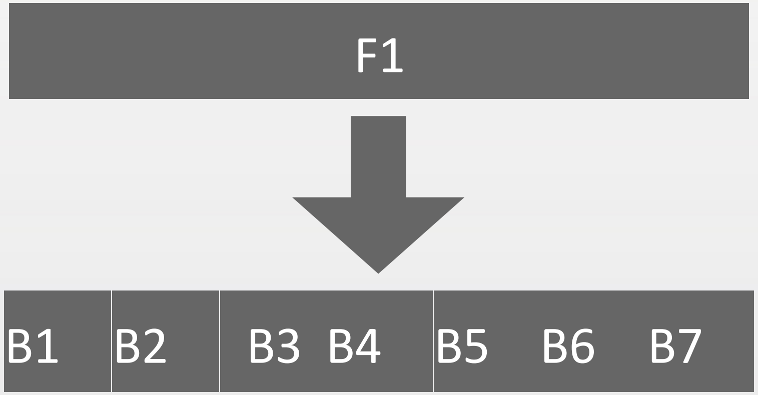
- When a file is written into HDFS, it is **split into blocks** of configured size
    - Default block size is **128 MB**
- Each block is **replicated** with a configured factor
    - Default replication factor is **3**
- The blocks are stored as actual, local files in the underlying storage nodes
- A block size should be small enough for data to be input to parallel processing, and large enough to reduce the seek time while reading the file

HDFS Blocks

- Block abstraction allows file sizes being independent of capacities of individual disks
    - A file in HDFS can be even **larger than the largest individual disk capacity** in a cluster
- Blocks are just **chunks of data**
    - No metadata other than the file-block mappings need to be stored as metadata by HDFS
- Block sizes and replication factors are **per file**

Original Data

Blocks

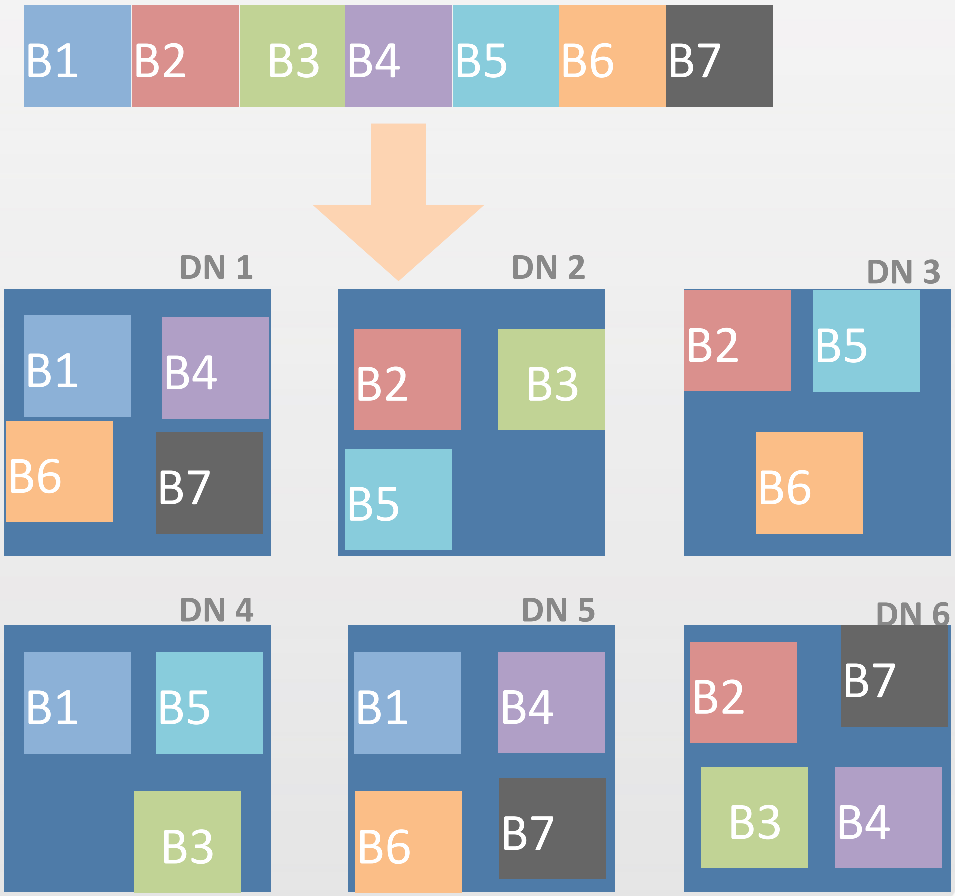


HDFS Block Storage

HDFS Block Storage

Blocks

HDFS Storage



Introduction to HDFS

- Rationale
- Block Storage
- **HDFS Daemons**
- Fault Tolerance
- HDFS Clients
- HDFS Write Path
- HDFS Read Path

HDFS Daemons

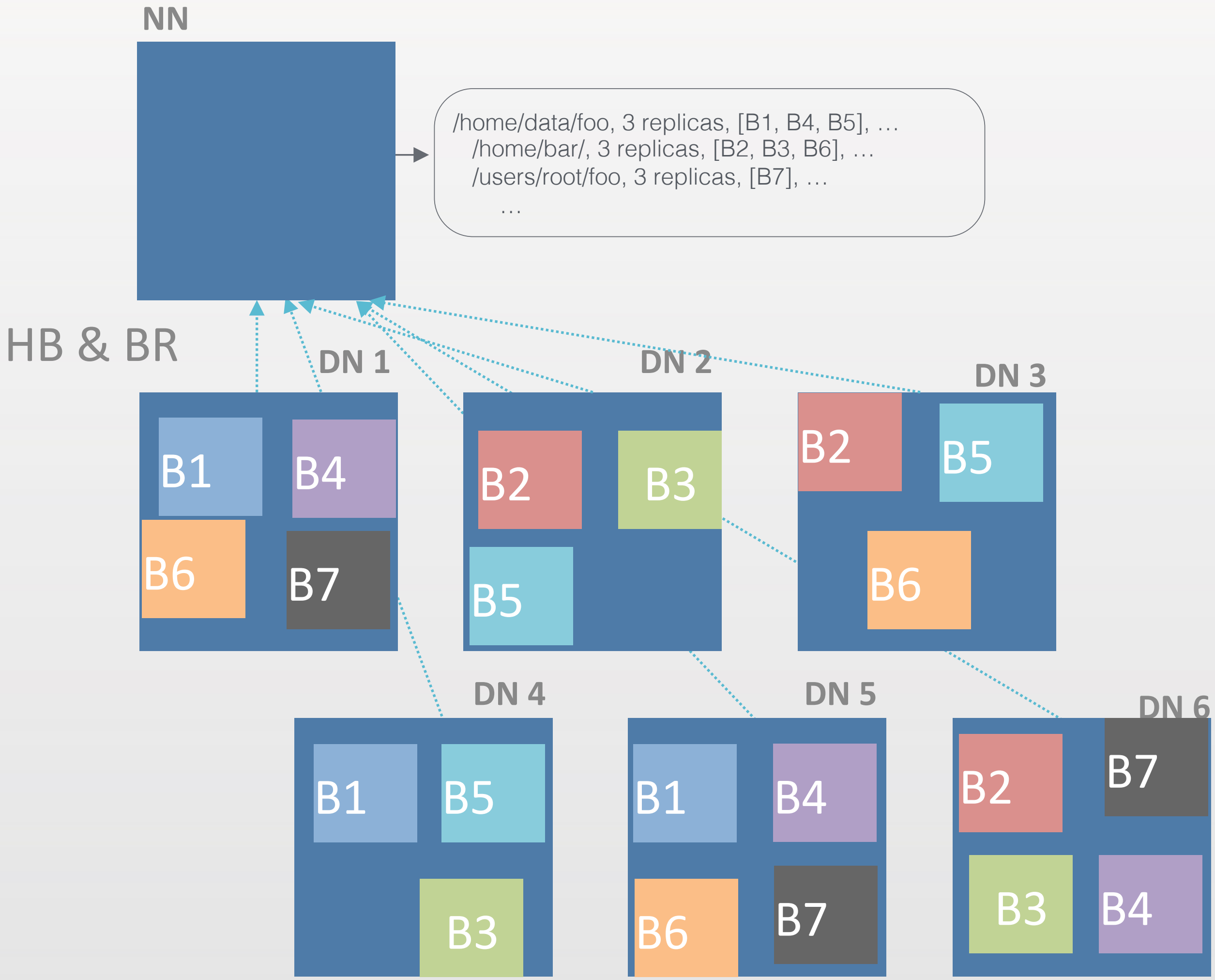
- HDFS has two kinds of daemons:
  - DataNodes
  - NameNode
- The **worker nodes** storing the blocks on their local filesystems are referred to as **DataNodes**
- The **master node** of an HDFS cluster managing the HDFS namespace is the **NameNode**

analyticscenter

DataNode

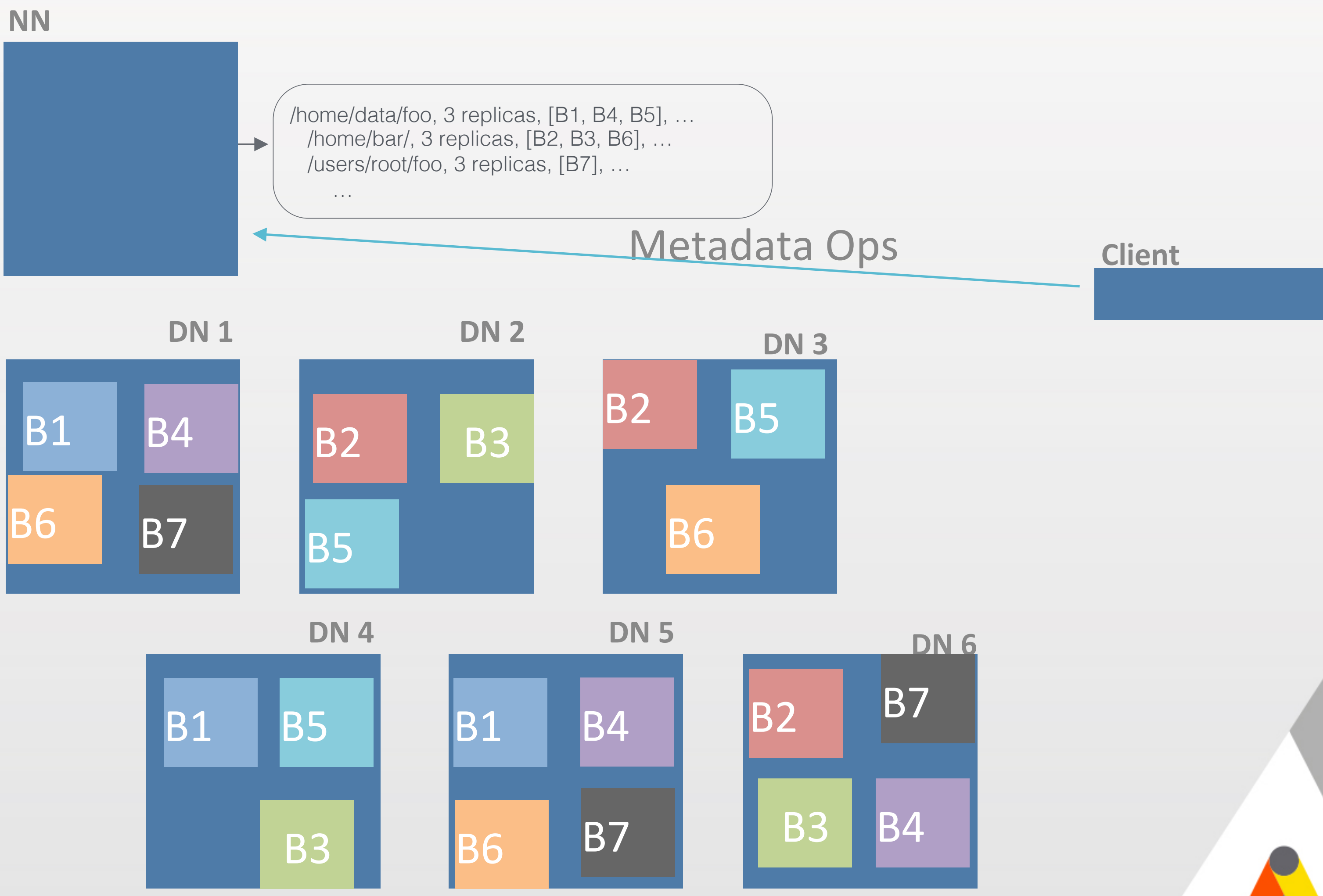
- DataNodes **store the actual blocks**, and **serve the read/write** requests for the blocks
- Block **creation**, **replication**, and **deletion** are also performed by DataNodes **upon instruction from the NameNode**
- A DataNode periodically sends **Heartbeats** to the NameNode
    - A Heartbeat indicates that a DataNode is functioning properly
- A DataNode periodically sends **BlockReports** to the NameNode
    - A BlockReport contains a list of all blocks stored on that DataNode

An HDFS Cluster

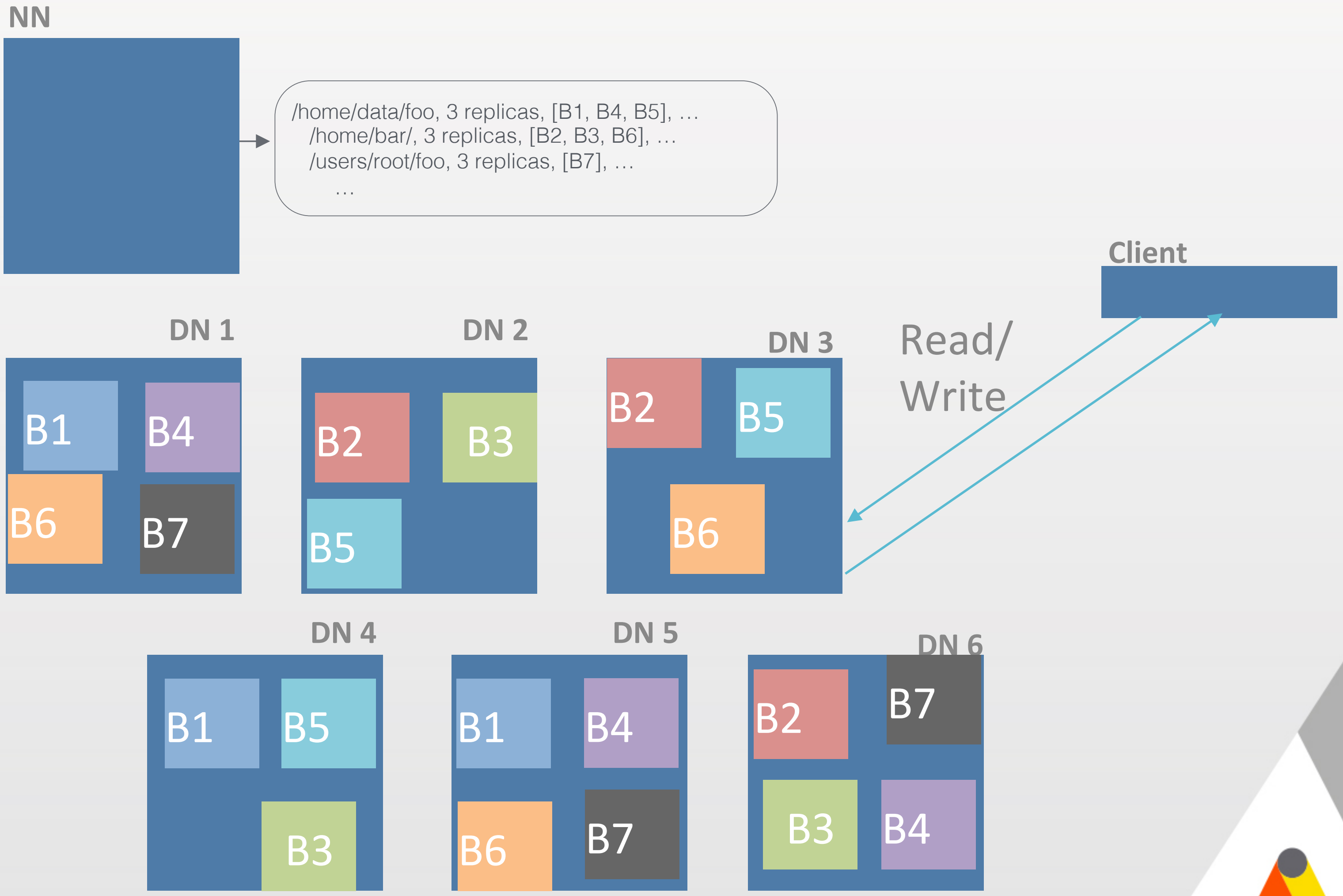


analyticscenter

An HDFS Cluster



An HDFS Cluster



Introduction to HDFS

- Rationale
- Block Storage
- HDFS Daemons
- **Fault Tolerance**
- HDFS Clients
- HDFS Write Path
- HDFS Read Path

Fault Tolerance

- NameNode blocks I/O to a DataNode if it doesn't function properly
  - When a DataNode fails to send Heartbeat
- NameNode decides re-replication of a block when necessary
- Re-replication occurs when a block is under-replicated:
  - a DataNode unavailability,
  - a replica corruption,
  - a disk failure in a DataNode,
  - an increase in the replication factor

Introduction to HDFS

- Rationale
- Block Storage
- HDFS Daemons
- Fault Tolerance
- **HDFS Clients**
- HDFS Write Path
- HDFS Read Path

HDFS Clients

- An HDFS client should be able to have access to both:
  - the NameNode for metadata operations
  - and the DataNodes to actually read/write data
- HDFS can be accessed from the:
  - FS Shell
    - A commandline interface letting a user interact with the data in HDFS
  - DFSAdmin
    - DFSAdmin is also a command, letting the clients administer HDFS
  - Browser Interface
    - An HDFS installation is configured with a Web server exposing the HDFS namespace

analyticscenter

FS Shell

- FS Shell commands have a syntax similar to other shells (e.g. bash)

```
# The following command creates a directory
# called /foo
# Note the hadoop fs -cmd [args] syntax

$ hadoop fs -mkdir /foo
```

FS Shell

- FS Shell commands have a syntax similar to other shells (e.g. bash)

```
# The following command removes a directory
# called /foo
# Note the hadoop fs -cmd [args] syntax

$ hadoop fs -rm -R /foo
```

analyticscenter

FS Shell

- FS Shell can be used to add a file to HDFS

```
# The following command adds the file
# /path/to/local/bar to hdfs:///data/bar

$ hadoop fs -put /path/to/local/bar  /data/bar

# The following command is identical

$ hadoop fs -copyFromLocal /path/to/local/bar /data/bar
```


analyticscenter

FS Shell

- FS Shell can be used to fetch a file in HDFS to the local computer

```
# The following command retrieves the file
# hdfs:///data/bar into /path/to/local/bar

$ hadoop fs -get /data/bar /path/to/local/bar

# The following command is identical

$ hadoop fs -copyToLocal /data/bar /path/to/local/bar
```


analyticscenter

Demo

Basic HDFS Shell Usage

Demo

Exploring the Block Storage

Java API

- **`o.a.h.fs.FileSystem`** is the generic base class (abstract) for a generic file system, which may be HDFS
- A Java client connecting to the HDFS should initialize a **`FileSystem`** with **`o.a.h.conf.Configuration`**, using the factory method **`FileSystem.get`**

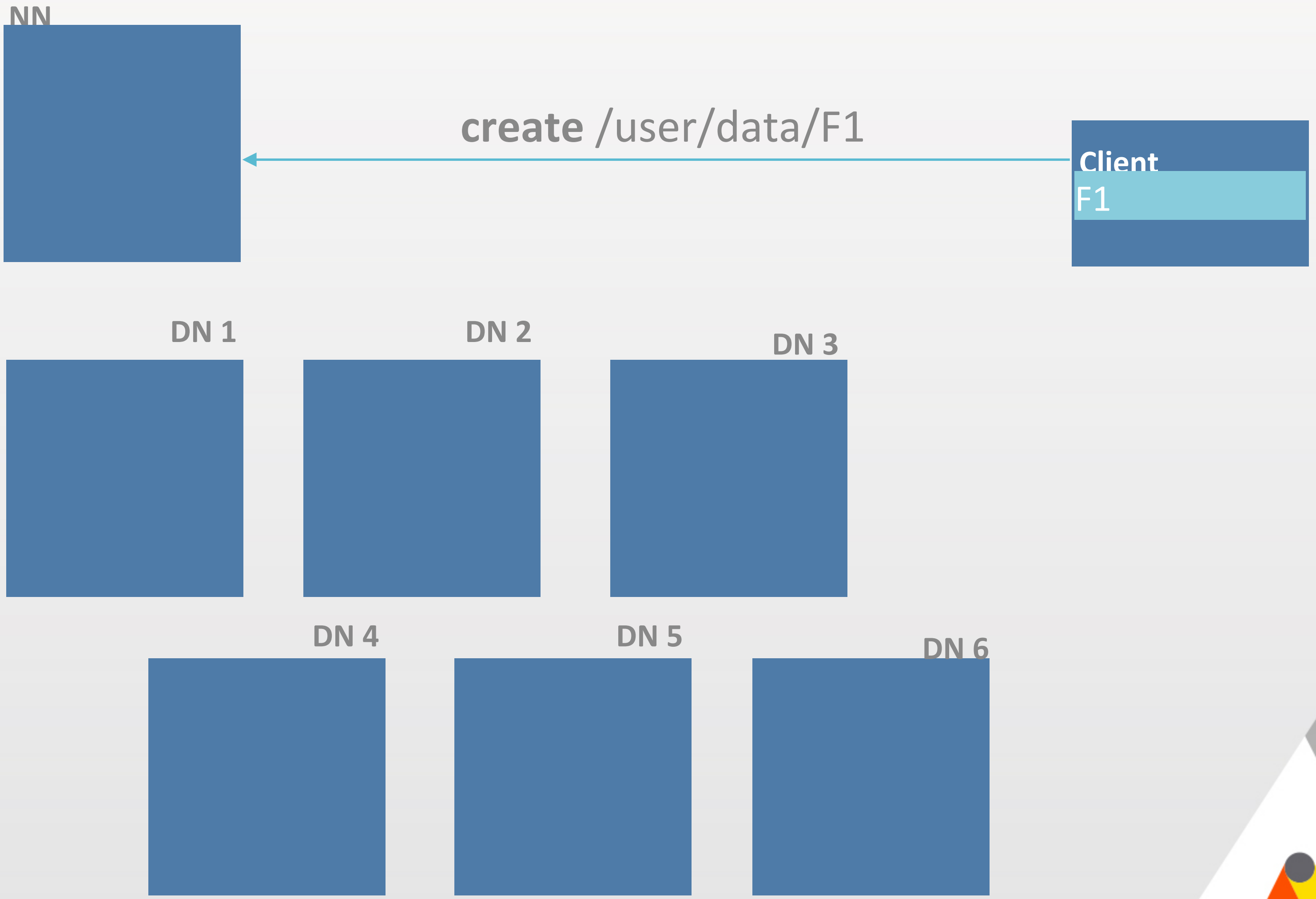
Introduction to HDFS

- Rationale
- Block Storage
- HDFS Daemons
- Fault Tolerance
- HDFS Clients
- **HDFS Write Path**
- HDFS Read Path

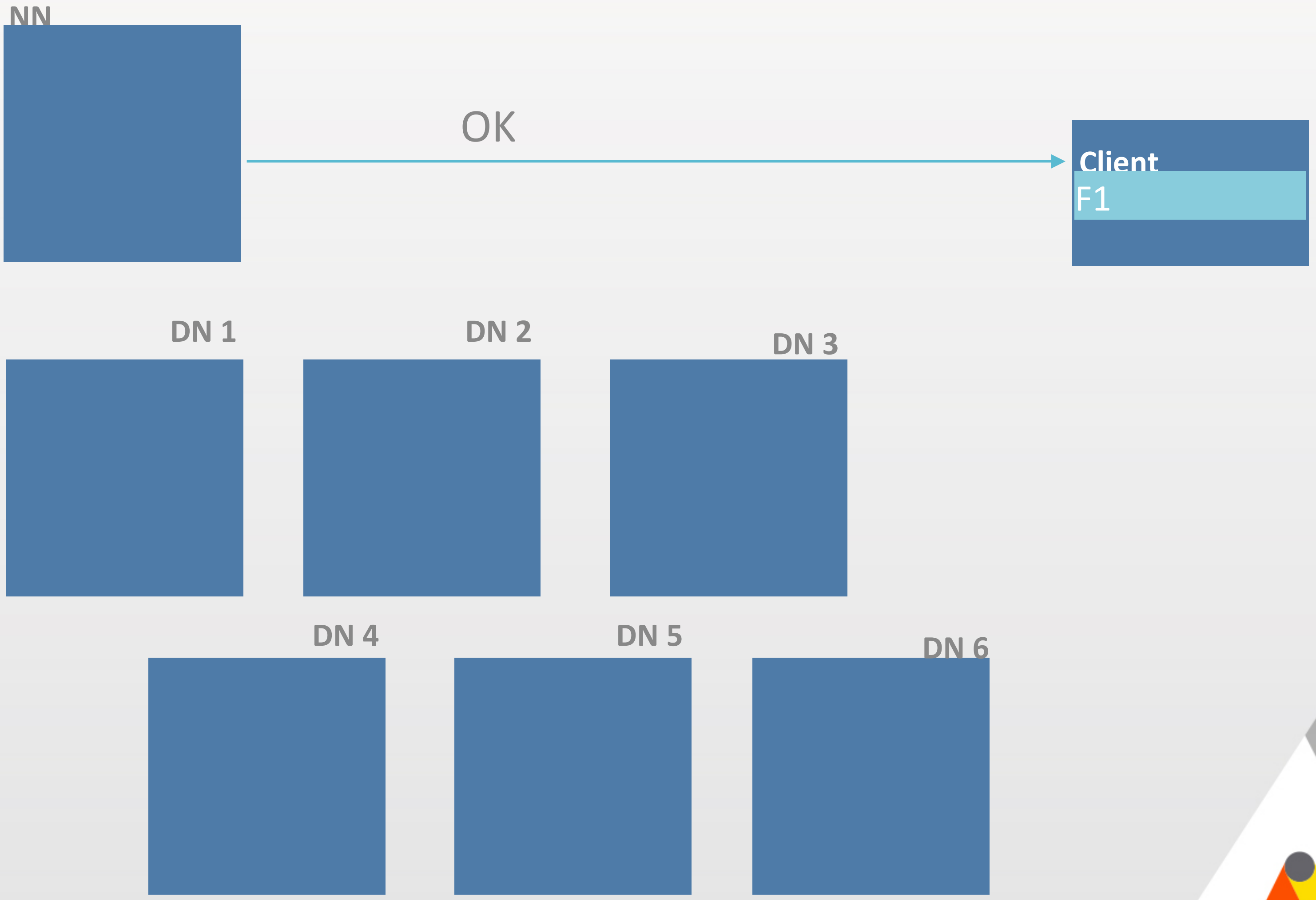
Writing to HDFS

- When a client wants to write a file into HDFS
  i. it first **stages** temporary local files of a size of a block
  ii. as a block of data is accumulated, the client **connects to the NameNode**
  iii. the **NameNode allocates a block**, and responds to the client with the **identity of the DataNodes** (determined by the replica placement strategy) and the **destination data block**
  iv. the **client connects to the first DataNode** and **sends the accumulated block in smaller portions** to the DataNode
  v. as the first **DataNode** receives the block in small portions, it **flushes** the portion **to its local filesystem** and then **sends** it to the **next DataNode** in the **replication pipeline**

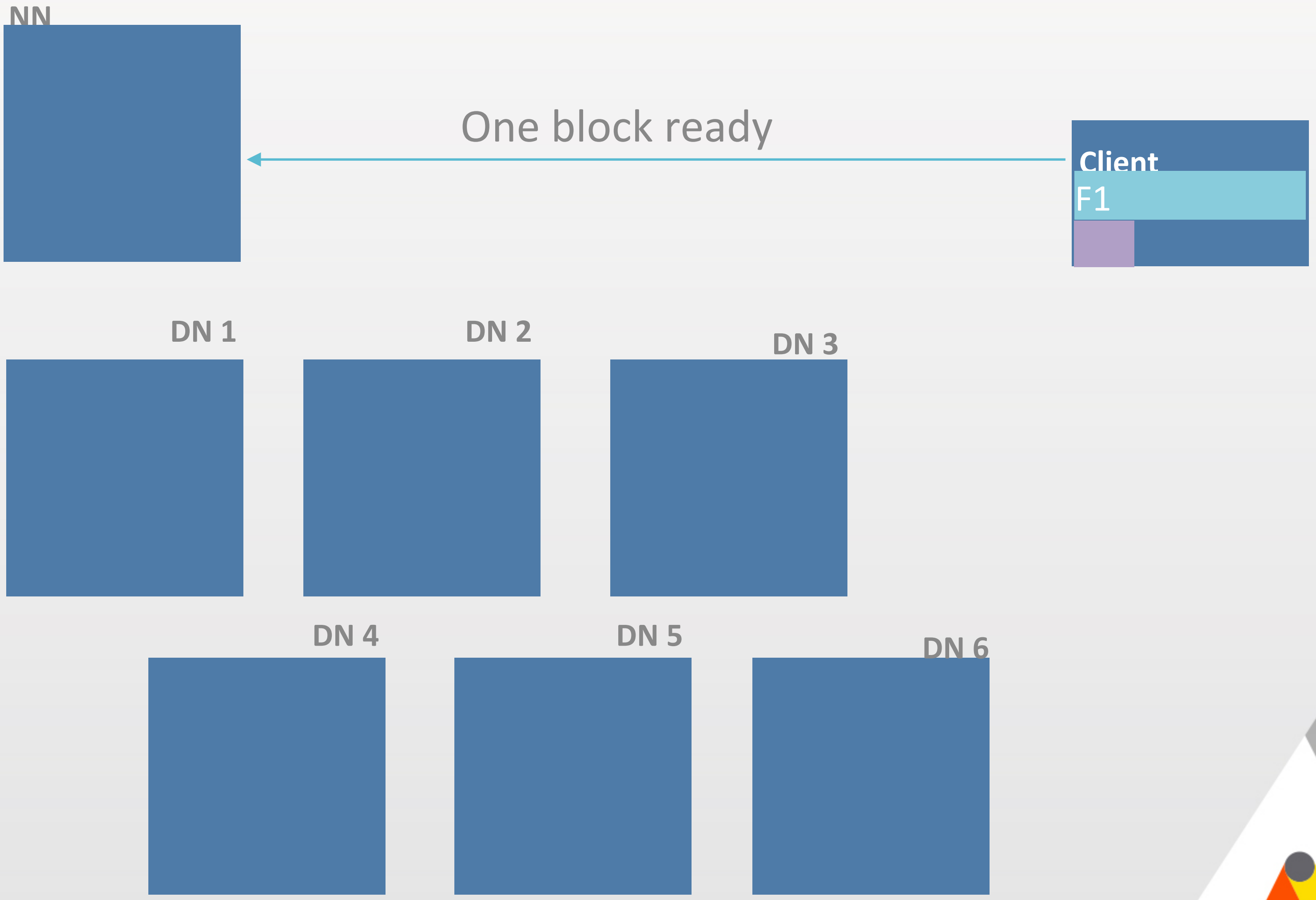
Writing to HDFS



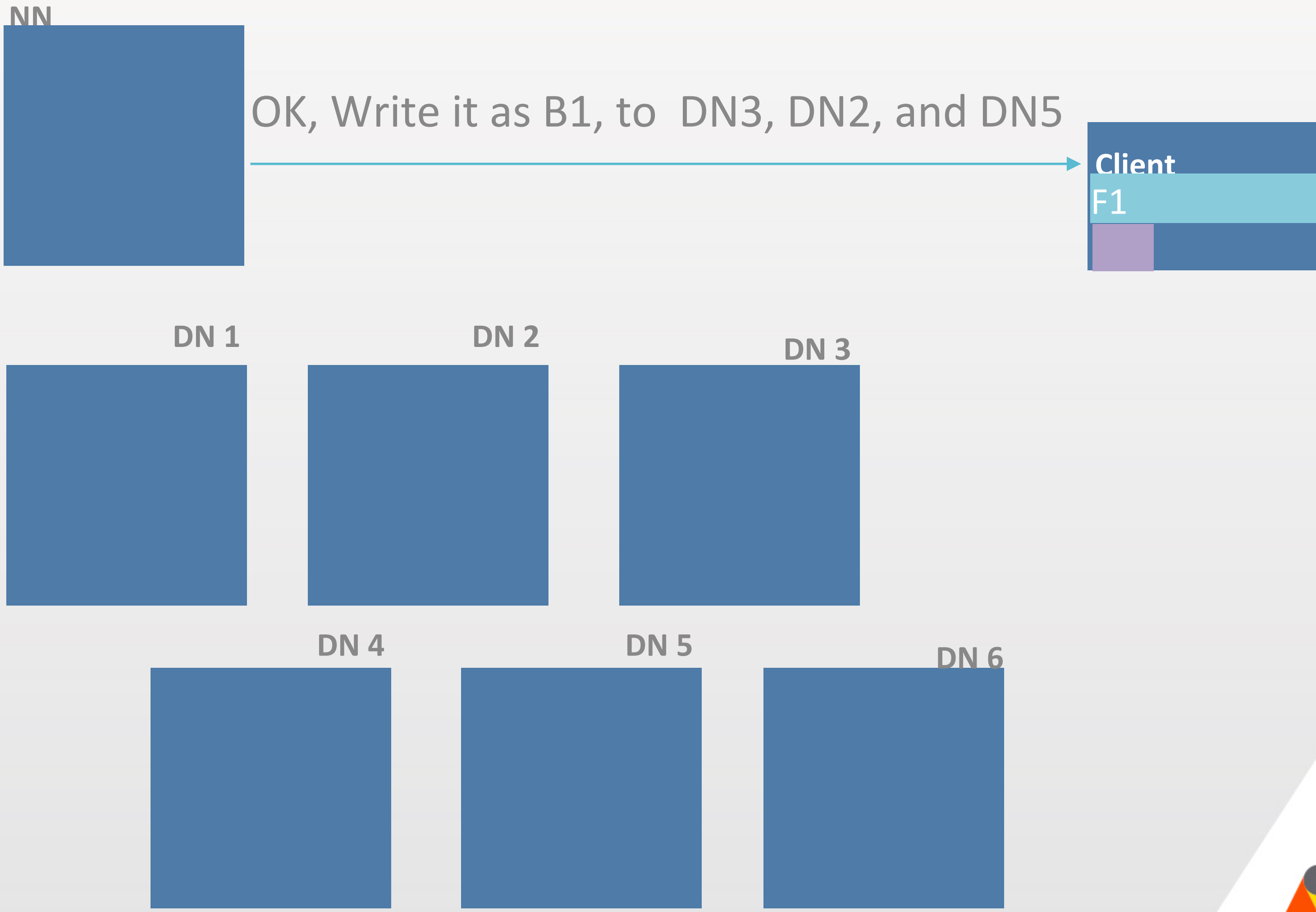
Writing to HDFS



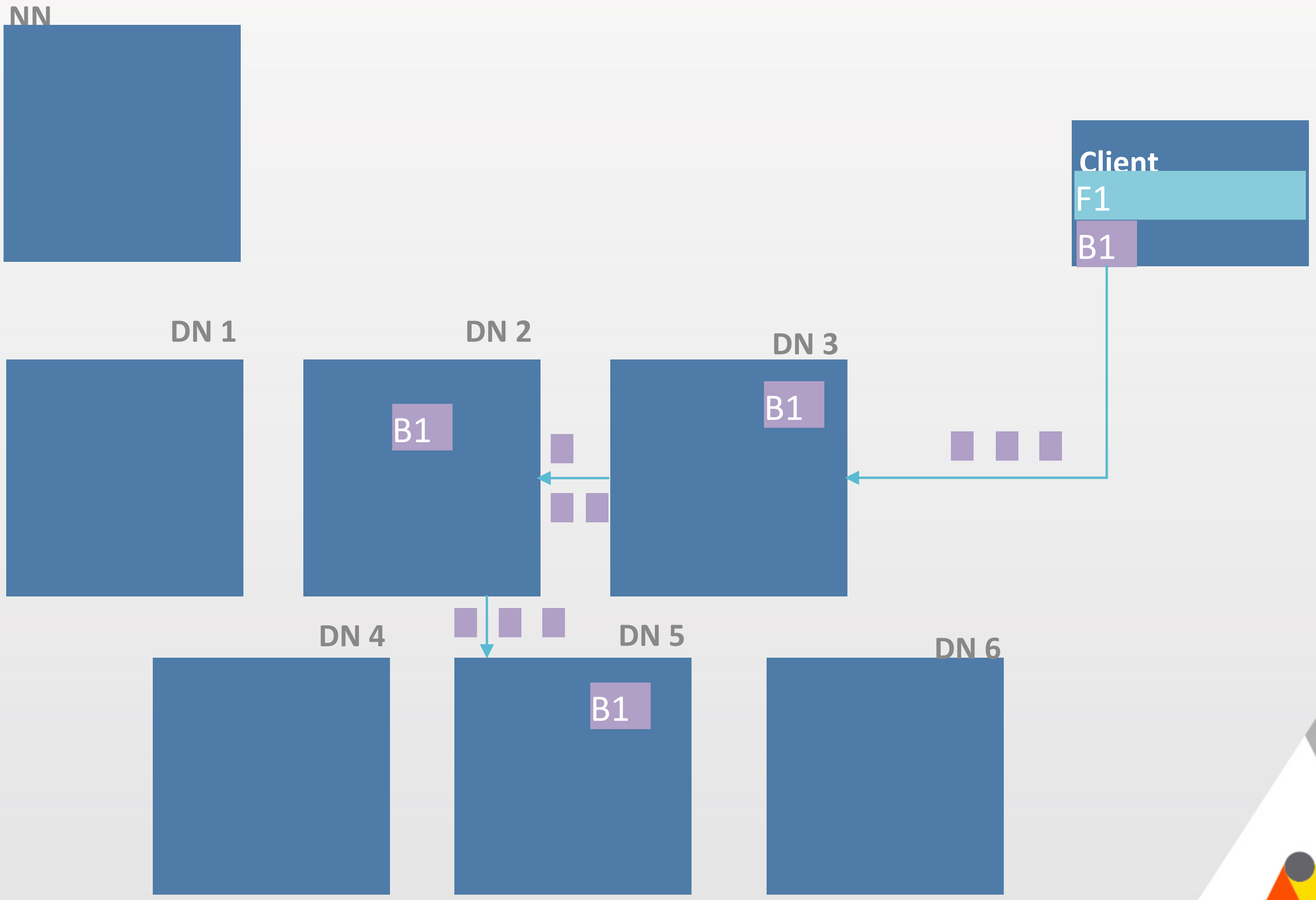
Writing to HDFS



Writing to HDFS



Writing to HDFS

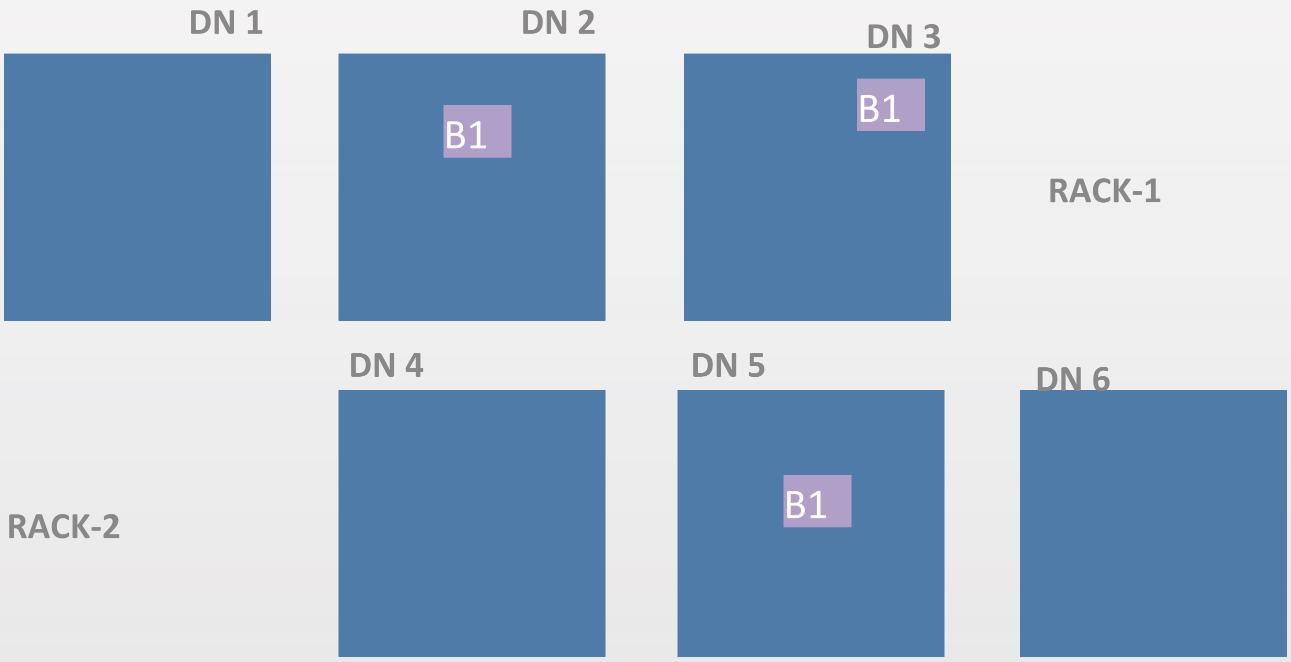


Replica Placement

- The **first replica** is preferred to be put on a DataNode in the local rack
  - If the client is itself a DataNode, the first replica is put on that particular node
- The **second replica** is put on a randomly selected DataNode, different from the first DataNode, **in the local rack**
- The **third replica** is put on a randomly selected DataNode **in a different rack**

analyticscenter

Writing to HDFS



Introduction to HDFS

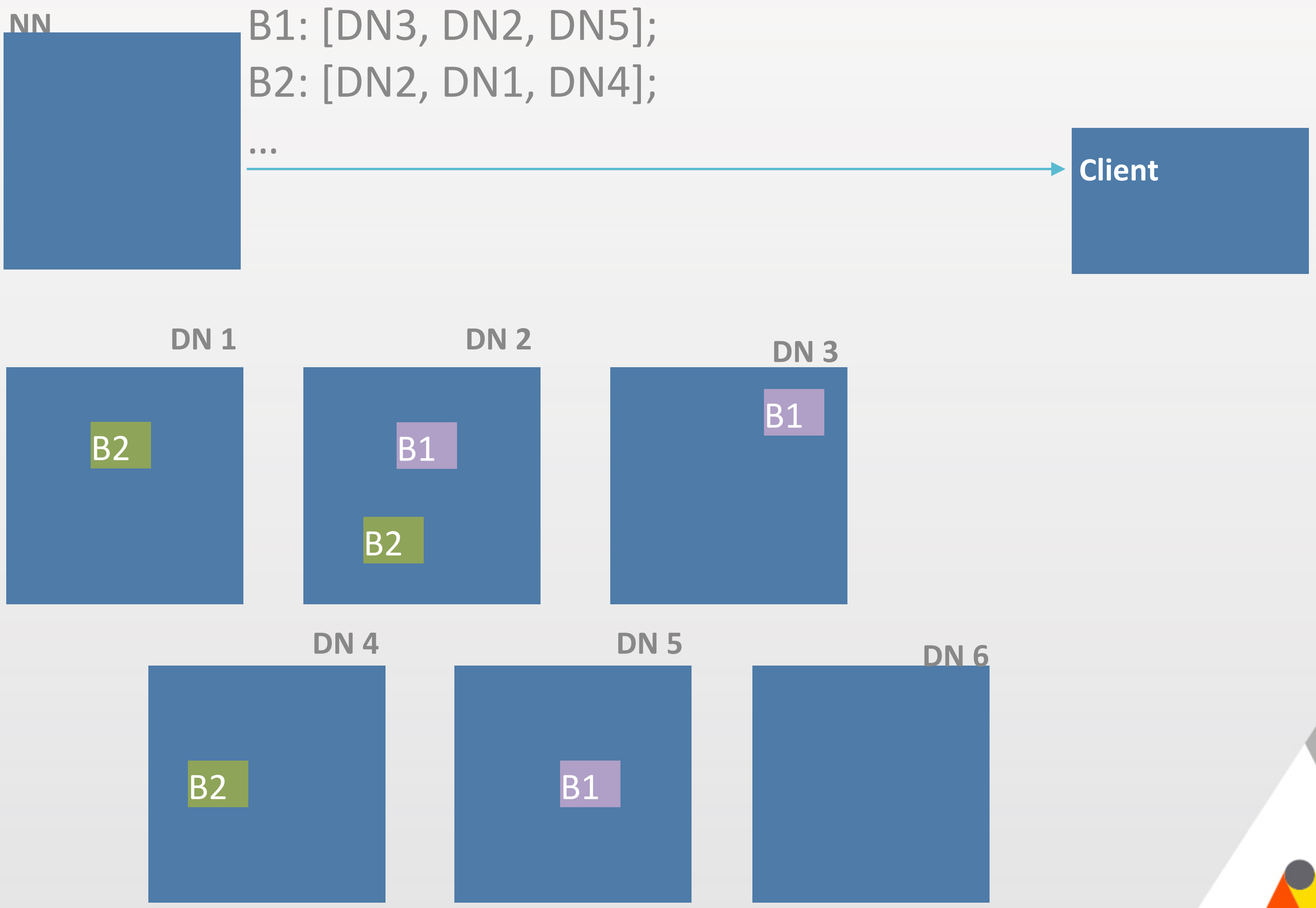
- Rationale
- Block Storage
- HDFS Daemons
- Fault Tolerance
- HDFS Clients
- HDFS Write Path
- **HDFS Read Path**

Reading from HDFS

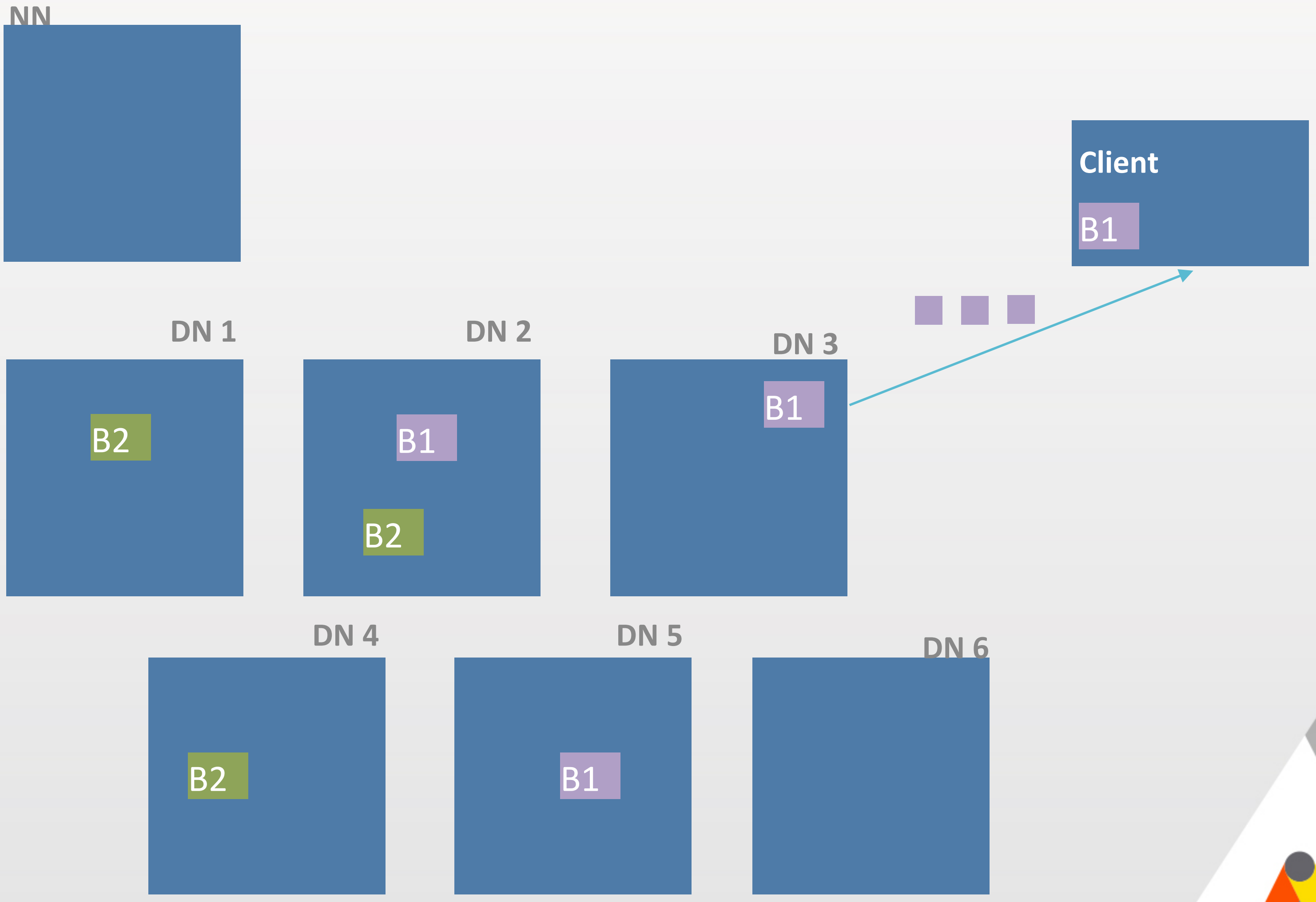
- When a client wants to read a file from HDFS
  1. it first **connects** to the **NameNode** and ask for the block locations for the next batch of blocks (a few)
  2. the client is responded with the block locations:
     - Block locations are **sorted by proximity** as described in 'Replica Selection'
  3. blocks are **read** by the client **in order**:
     1. the client connects to the **closest DataNode** containing the block
     2. **block data is streamed** from the DataNode to the client
     3. if the read fails, the client tries the next closest DataNode, and reports the failed DataNode to the NameNode

Reading from HDFS

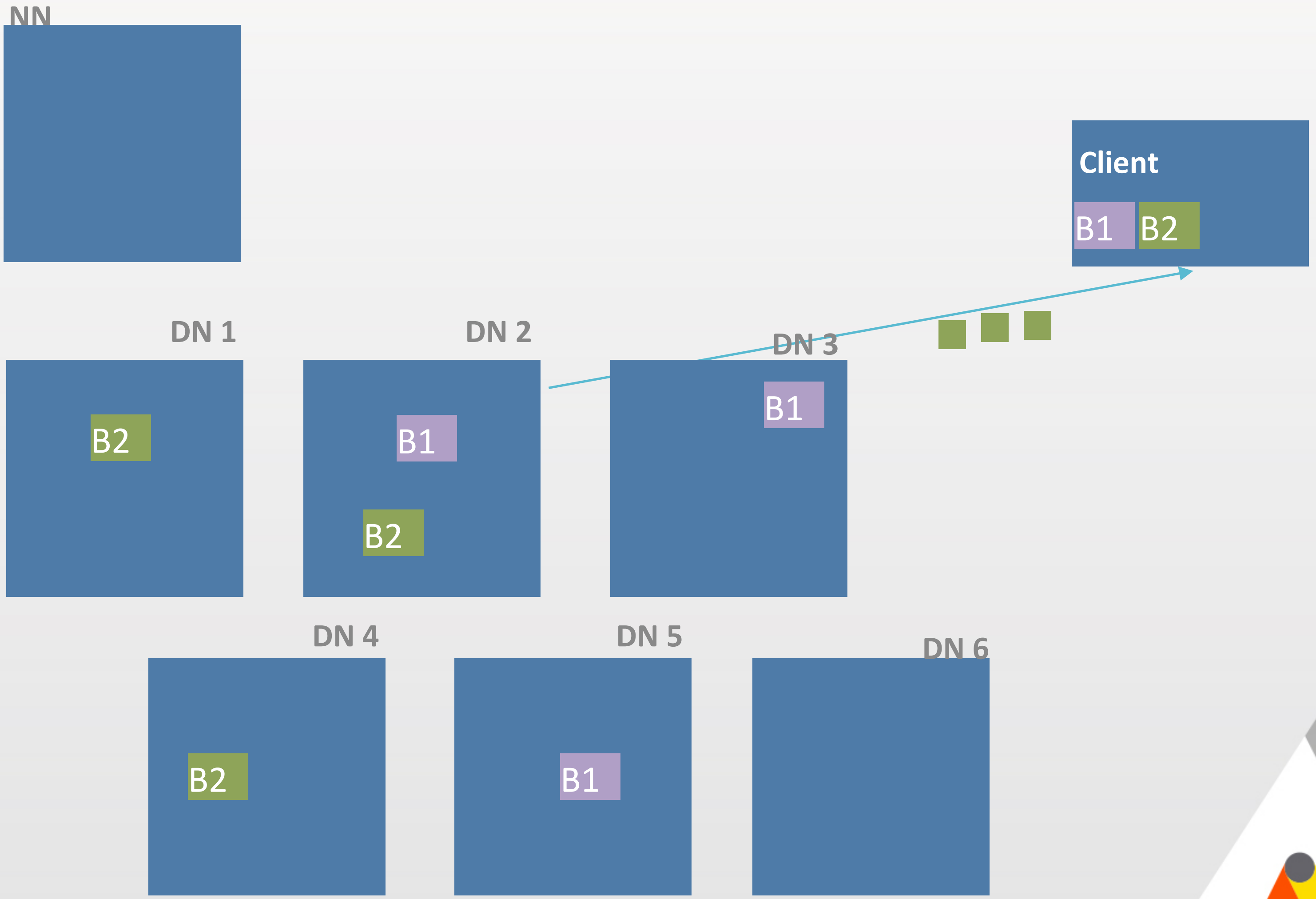
Reading from HDFS



Reading from HDFS



Reading from HDFS



Replica Selection

- The **closest replica** is selected to satisfy a read request with the following strategy:
  - If there exists a replica on the local node, the read can be 'shortcut'
    - This should be configured by setting:
      - '`dfs.client.read.shortcircuit`' to '`true`'
      - '`dfs.domain.socket.path`' to a path

      on both the DataNode and the client
  - If there exists a replica on the local rack, that replica is selected
  - If HDFS spans multiple data centers, a replica in the local data center is selected

Introduction to HDFS End of Chapter