# Distributed File System

*Group 1*

# Table of contents

# 1.Introduction

- What is it ?
  - Client/server based application
  - Client can access and process data in the server just like in its own computer

# 1.Introduction

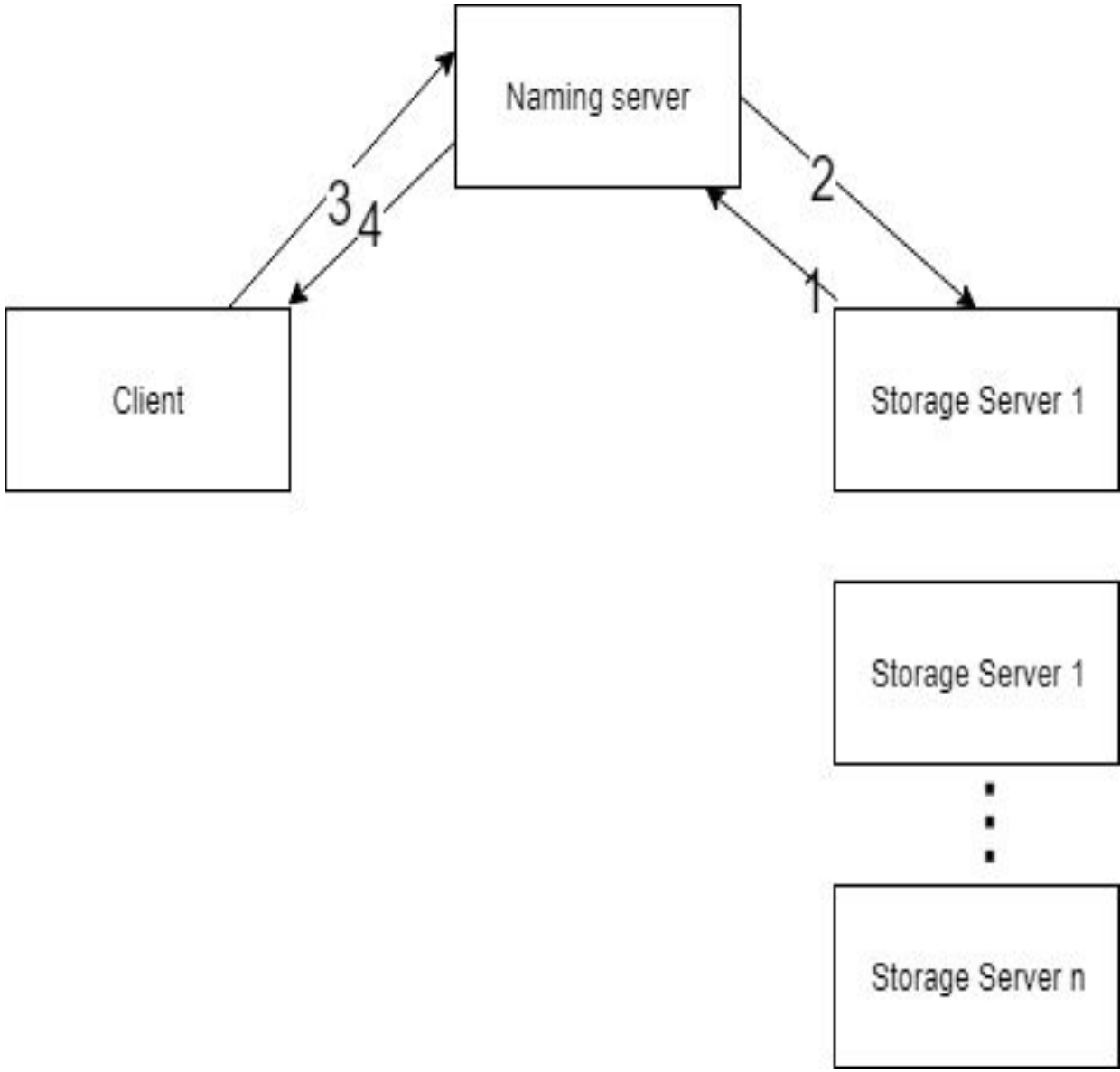
- What is it ?
  - Client/server based application
  - Client can access and process data in the server just like in its own computer
- Why do we need it?
  - More spaces
  - Centralized storage
  - File sharing
  - Back up

# 2. Project overview

- **3 components:** Client, Naming server, Storage server
  - **Storage server:** Store data
  - **Naming server:** The smart middle man
  - **Clients:** Access the network using provided APIs
- Uses **Java RMI** to communicate

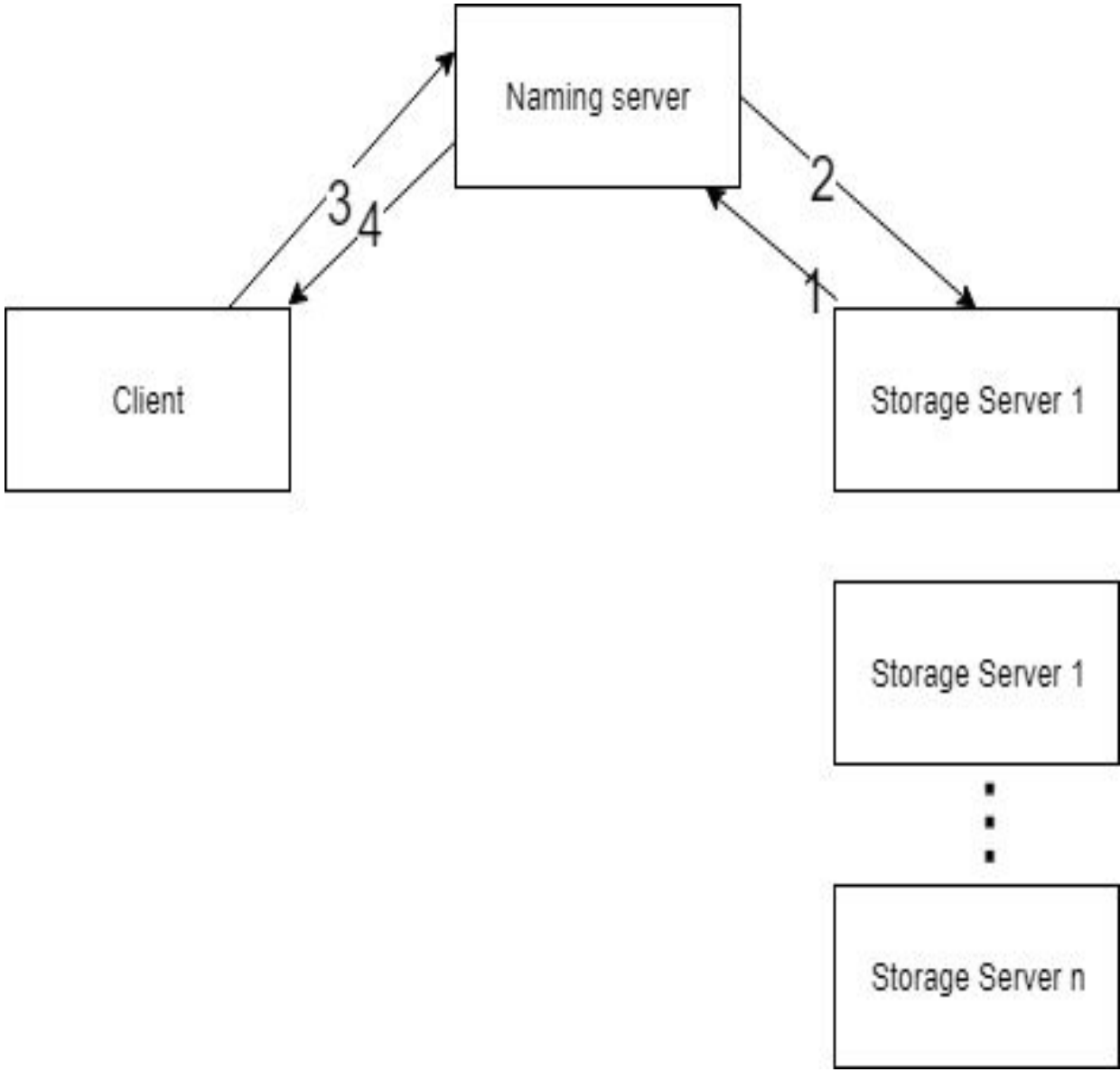
# 3. Architecture

- Storage servers - Naming server communication
- Client - Naming server communication



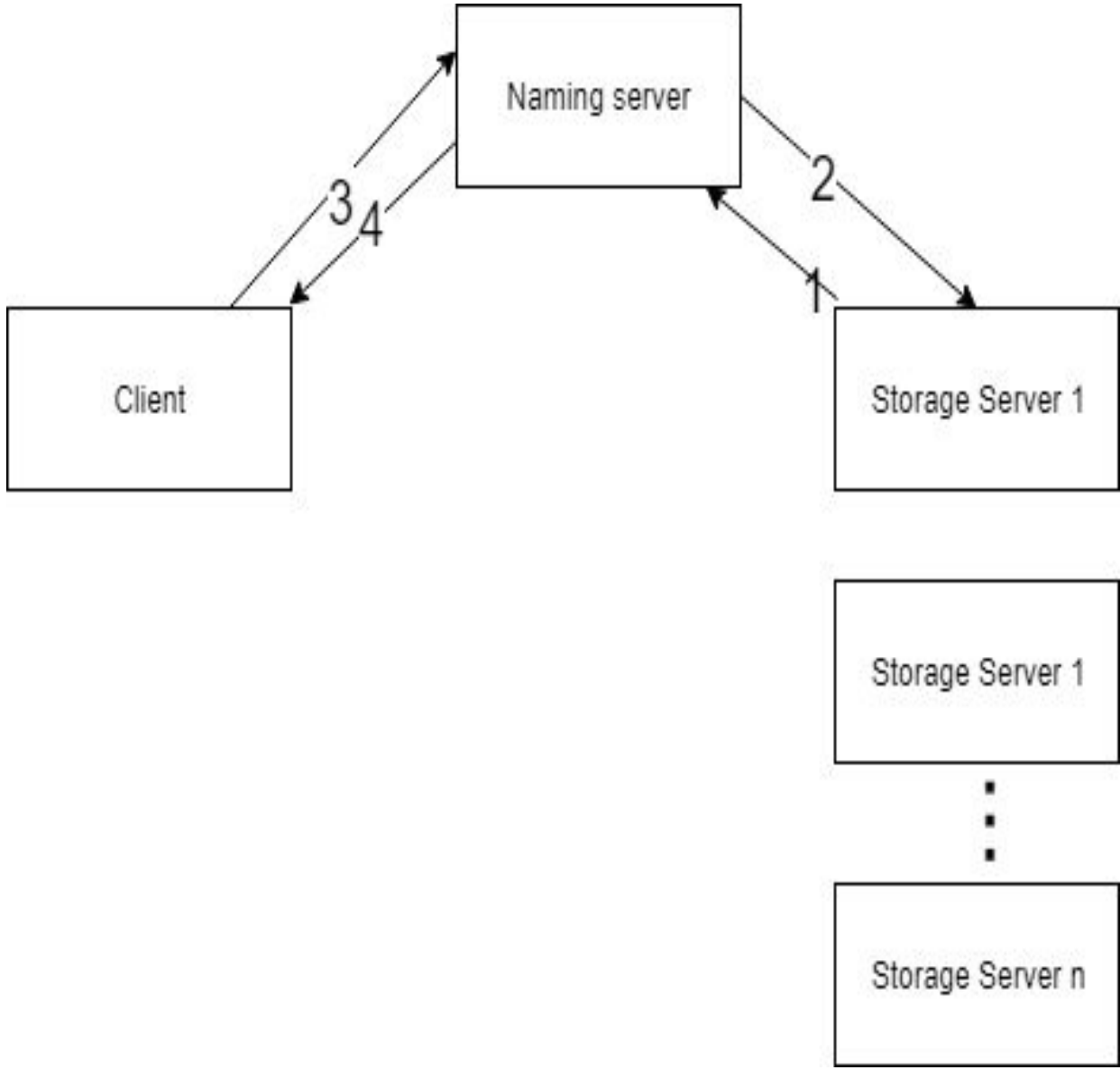
# 3. Architecture

- Storage servers - Naming server communication
    - **Storage servers** send their list of file paths which they are storing
    - **Naming server** build the directory tree
    - Tree leaves that represent files also contain a Stub of its respective **Storage server**



# 3. Architecture

- Client - Naming server communication
    - Client contacts server and create a session
    - Directory tree node operations:
        - List
        - Get current working directory
        - Change directory
          => Perform directly on Naming Server
    - File-related operations:
        - Read
        - Upload
        - Delete
          => Perform through Storage servers



# 4. Implementation

1. Multi-client
2. Data replication

# 4. Implementation – Multi client

Uses **Remote-Session** pattern

- Login:

```
public class LoginImpl implements Login {
  private NamingServer namingServer = new NamingServer();

  @Override
  public Session login() throws RemoteException {
     return new SessionImpl(namingServer);
  }
}
```

# 4. Implementation – Multi client

Uses **Remote-Session** pattern

- Session:

```
public class SessionImpl extends UnicastRemoteObject implements Session, Unreferenced {
  private NamingServer namingServer;
  DirectoryTreeNode currentNode;

  public SessionImpl(NamingServer namingServer) throws RemoteException {
     super();
     this.namingServer = namingServer;
     currentNode = namingServer.rootNode;
  }

 public void logout() throws RemoteException {
  unexportObject(this, true);
 }
}
```

# 4. Implementation – Replication

- Number of replicas for each files = 2
- Policy:
  - Upload: randomly selected 2 connected storage servers and write to them
  - Read: get the list of storage servers hosting the file, iterate through the list until the file is successfully read

# 5. Conclusion

- What was done:
  - Basic architecture of the DFS
  - Basic operations: upload, read, delete, directory traversal
  - Data replication
  - Multi-client
- What was not  done:
  - Security
  - Concurrency
  - Multiple naming servers

# 6. Demo

# Thank You