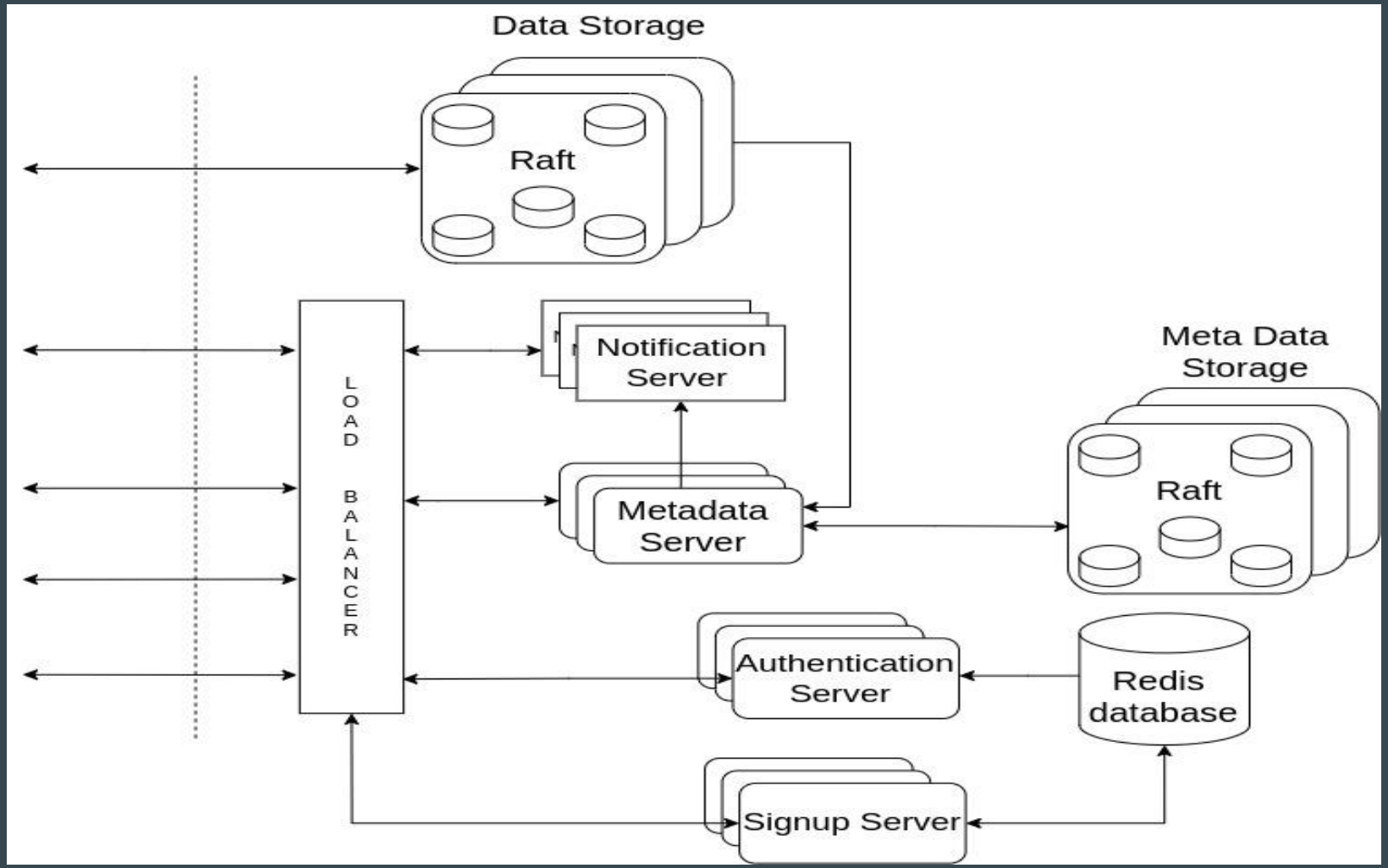
Distributed Storage System

Bratin Mondal 21CS10016
Swarnabh Mandal 21CS10068
Soukhin Nayek 21CS10062

System Assumptions & High-Level Considerations

- **Local LAN Focus:** Assumes deployment in a controlled LAN environment.
- **Scalability**: Designed with horizontal scale in mind - adding nodes should be straightforward.
- **Eventual Consistency:** Updates propagate asynchronously, so temporary inconsistencies may occur.
- **Simple Authentication:** Basic mechanisms (e.g., username/password with token-based) are used.
- **Complete File Transfer:** File chunking not implemented. Sending complete file for update. Possible improvement with Delta propagation and file compression
- **File type:** Tested with .txt files but implementation is universal.
- **Reliable communication:** using HTTP request for client server communication which uses reliable TCP connections.
- **Server Monitoring:** Automatic Detection and Recovery not implemented.


Data Storage
Raft
LOAD BALANCER
Notification Server
Metadata Server
Meta Data Storage
Raft
Authentication Server
Redis database
Signup Server

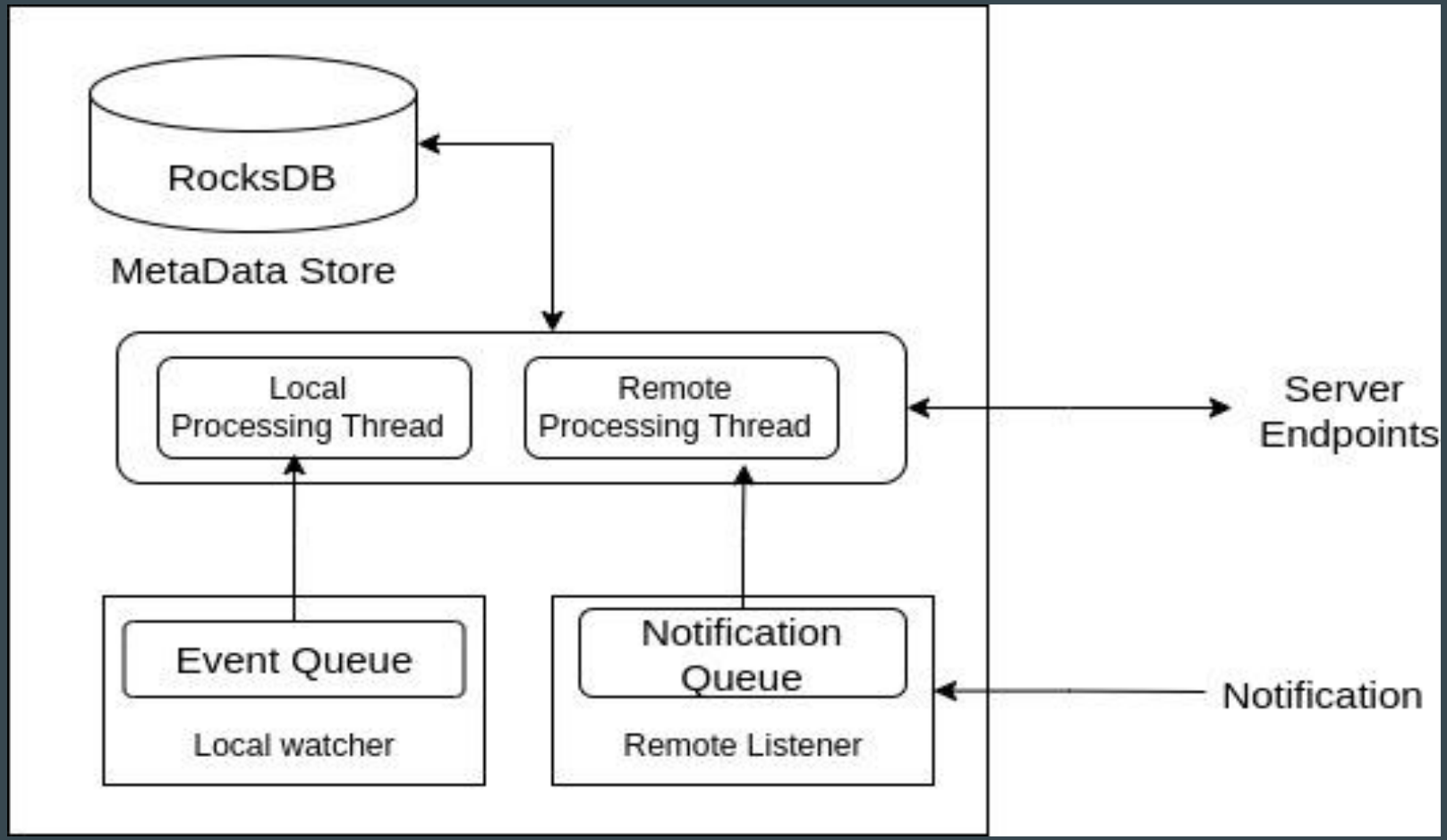
Server Components

- Authentication Server :
  - **/signup :** create user with username (unique) and password.
  - **/login :** from each device with the userID and password. Server returns Access_token to access each server. Access_token expires after sometime.
  - User Redis instance to store username and password.
- Metadata Server: need to send access_token and path in each route.
  - **/create-directory :** Route for creating directory. Error for wrong path or directory already exist.
  - **/list_directory :** route for listing sub-directories and files.
  - **/create-file :** route for creating file metadata.
  - **/get-file-endpoints :** for getting the block server endpoints where file is stored.
  - **/delete :** for deletion of file and directories.
- Notification Server:
  - **/subscribe :** subscribe to a notification server for a userID. Long polling with the connection till timeout(10sec) or new notification comes. On new notification close the connection and open another connection.
  - **/broadcast :** endpoint for metadata server to send the notification with a particular userID. Notification server broadcasts the notification with all the connections with same userIDs.

Server Components

- **Metadata Database :**
  - Key is “user_id:path” so even if path becomes same for two users user_id is unique for every user.
  - According to the key hash we will choose a cluster.
  - Each cluster contain 5 copies of the data. We use raft to choose replica leader.
  - So our database is **fault tolerant and it supports replication.**

- **Block Server Database :**
  - Similar to metadata database we create the key and choose a cluster from the key hash.
  - Similarly each cluster contains 5 copies of the data and raft is used to choose replica leader.
  - So block server is also **fault tolerant and it supports replication.**
  - We are maintaining a version with file content to handle concurrent update.
    - Content is updated when **version number matches.** And version number increased at block server for rejecting concurrent or outdated updates.
    - On rejection client side **generates conflicted file** and fetch updated file from server.
  - After successful update send confirmation to metaserver which sends notification to relevant users.


RocksDB
MetaData Store
Local
Processing Thread
Remote
Processing Thread
Server
Endpoints
Event Queue
Local watcher
Notification
Queue
Remote Listener
Notification

Client Components

- **Watcher** :
  - Monitors the directory and pushes events in queue
- **Listener:**
  - Listens for incoming changes from the Notification Server and addes them to queue
- **Event Processor:**
  - Remote Event Processing
  - Local Event Processing