DISTRIBUTED STORAGE SYSTEM (DSS) - MILESTONE DESIGN DOCUMENT
==========

GROUP 35 - SOCKET PROGRAMMING PROJECT
Milestone Commands Implementation: register-user, register-disk, configure-dss, deregister-user, deregister-disk

==========
1. DESIGN DOCUMENT (50%) - Description of DSS Application Program Design
==========

The Distributed Storage System (DSS) is implemented as a client-server architecture using UDP sockets for communication. The system consists of three main components:

- Manager: Central coordinator that handles registration, configuration, and coordination
- Users: Clients that can configure DSS systems and perform storage operations
- Disks: Storage nodes that provide actual storage capacity

The system uses a JSON-based message format for all communications and maintains state information at the manager to track registered users, disks, and configured DSS systems.

==========
(a) MESSAGE FORMAT FOR EACH COMMAND IMPLEMENTED FOR THE MILESTONE
==========

(a.i) REGISTER-USER Command

Request Format:
```
{
   "command": "register-user",
   "parameters": {
      "user_name": "U1",
      "ipv4_addr": "127.0.0.1",
      "m_port": 18520,
      "c_port": 18521
   },
   "sender": "U1"
}
```

Response Format:
```
{
   "status": "SUCCESS" | "FAILURE",
   "message": "Error message if failure"
}
```

(a.ii) REGISTER-DISK Command

Request Format:

```
{
   "command": "register-disk",
   "parameters": {
      "disk_name": "D1",
      "ipv4_addr": "127.0.0.1",
      "m_port": 18510,
      "c_port": 18511
   },
   "sender": "D1"
}
```

Response Format:

```
{
   "status": "SUCCESS" | "FAILURE",
   "message": "Error message if failure"
}
```

(a.iii) CONFIGURE-DSS Command

Request Format:

```
{
   "command": "configure-dss",
   "parameters": {
      "dss_name": "DSS1",
      "n": 3,
      "striping_unit": 1024
   },
   "sender": "U1"
}
```

Response Format:

```
{
   "status": "SUCCESS" | "FAILURE",
   "message": "Error message if failure"
}
```

(a.iv) DEREGISTER-USER Command

Request Format:

```
{
   "command": "deregister-user",
   "parameters": {
      "user_name": "U1"
   },
   "sender": "U1"
}
```

Response Format:

```
{
   "status": "SUCCESS" | "FAILURE",
   "message": "Error message if failure"
}
```

(a.v) DEREGISTER-DISK Command

Request Format:

```
{
   "command": "deregister-disk",
   "parameters": {
      "disk_name": "D1"
   },
   "sender": "D1"
}
```

Response Format:

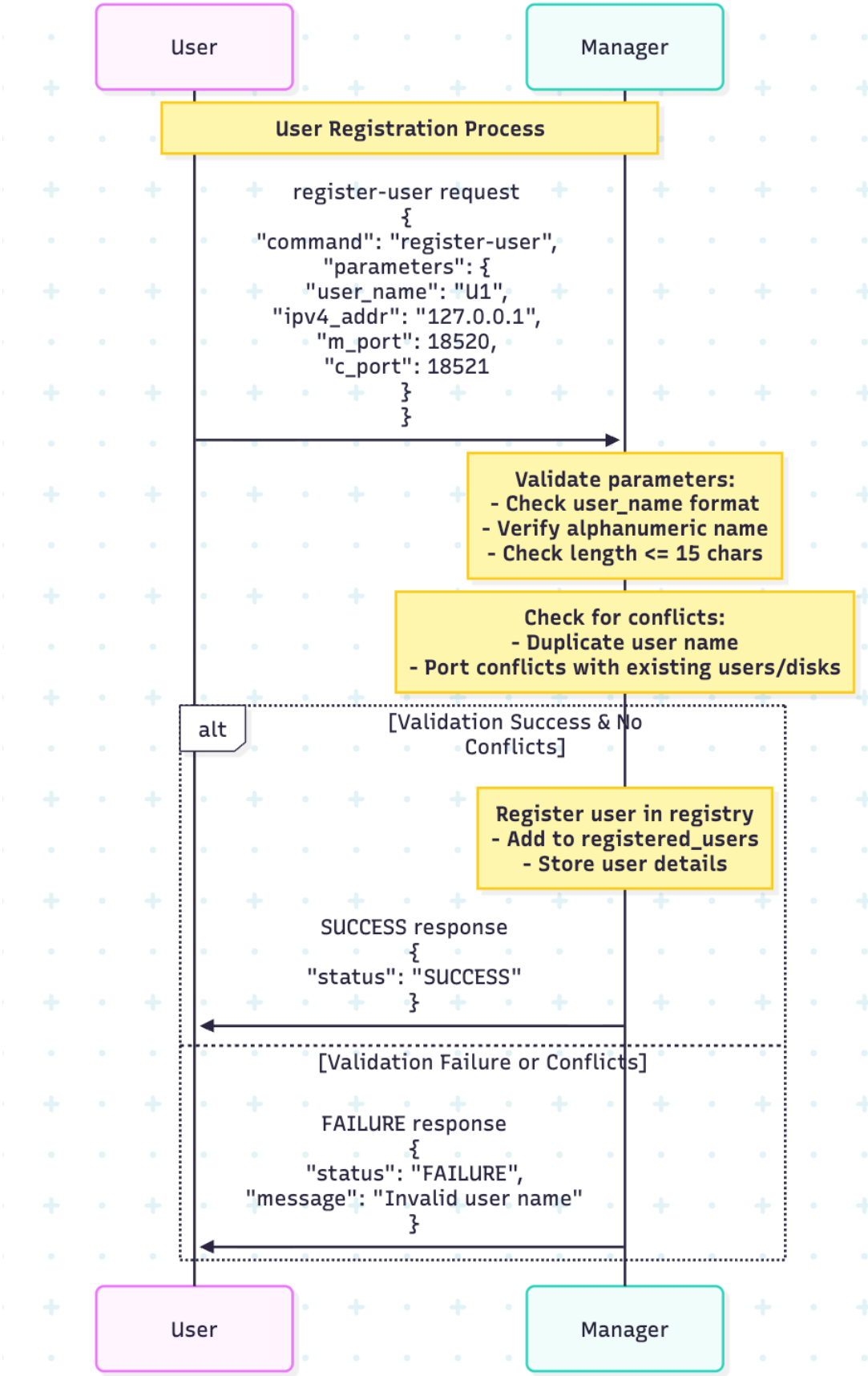
```
{
   "status": "SUCCESS" | "FAILURE",
   "message": "Error message if failure"
}
```

==========

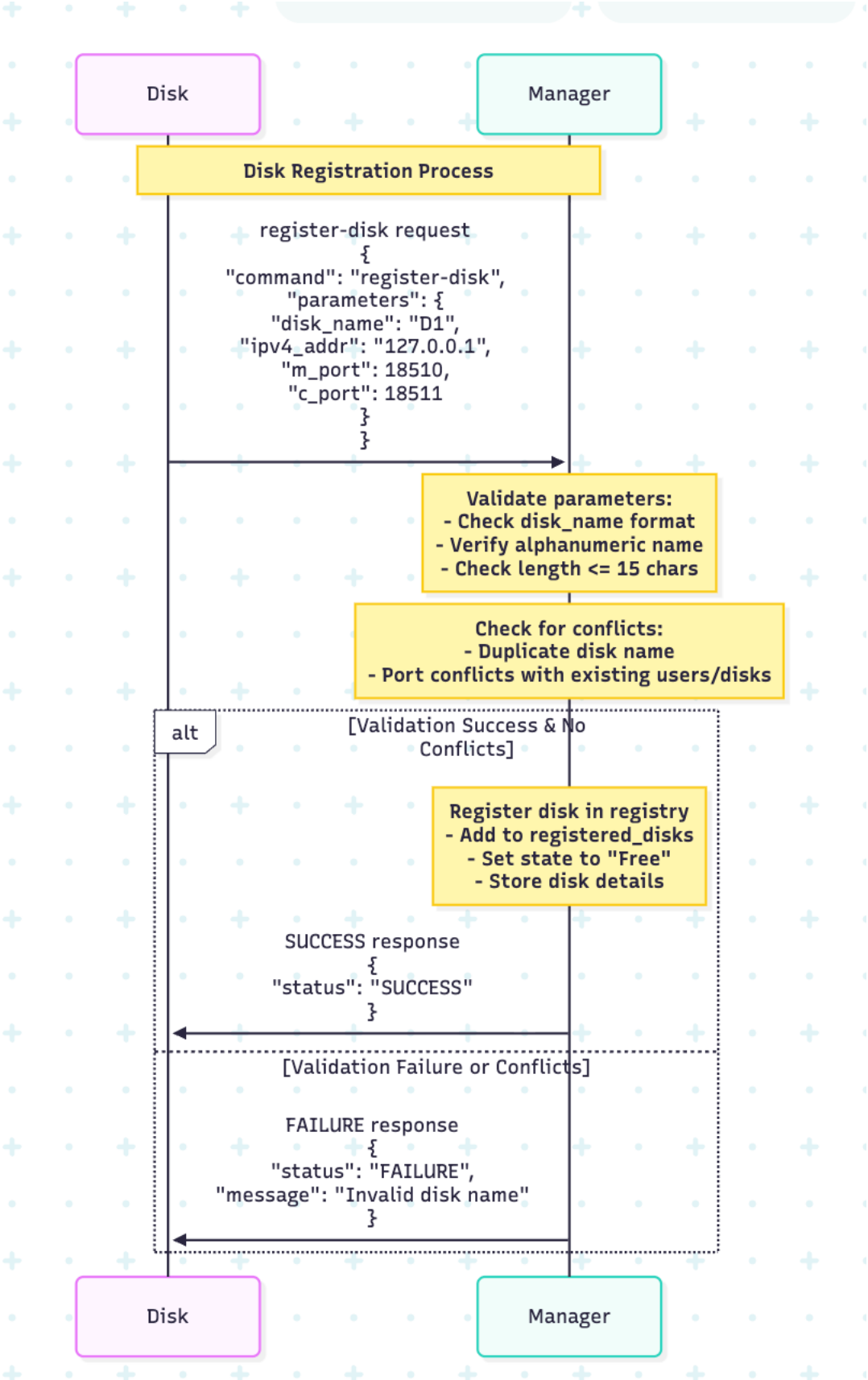
(b) TIME-SPACE DIAGRAMS FOR EACH COMMAND IMPLEMENTED

==========

(b.i) REGISTER-USER Protocol Flow:

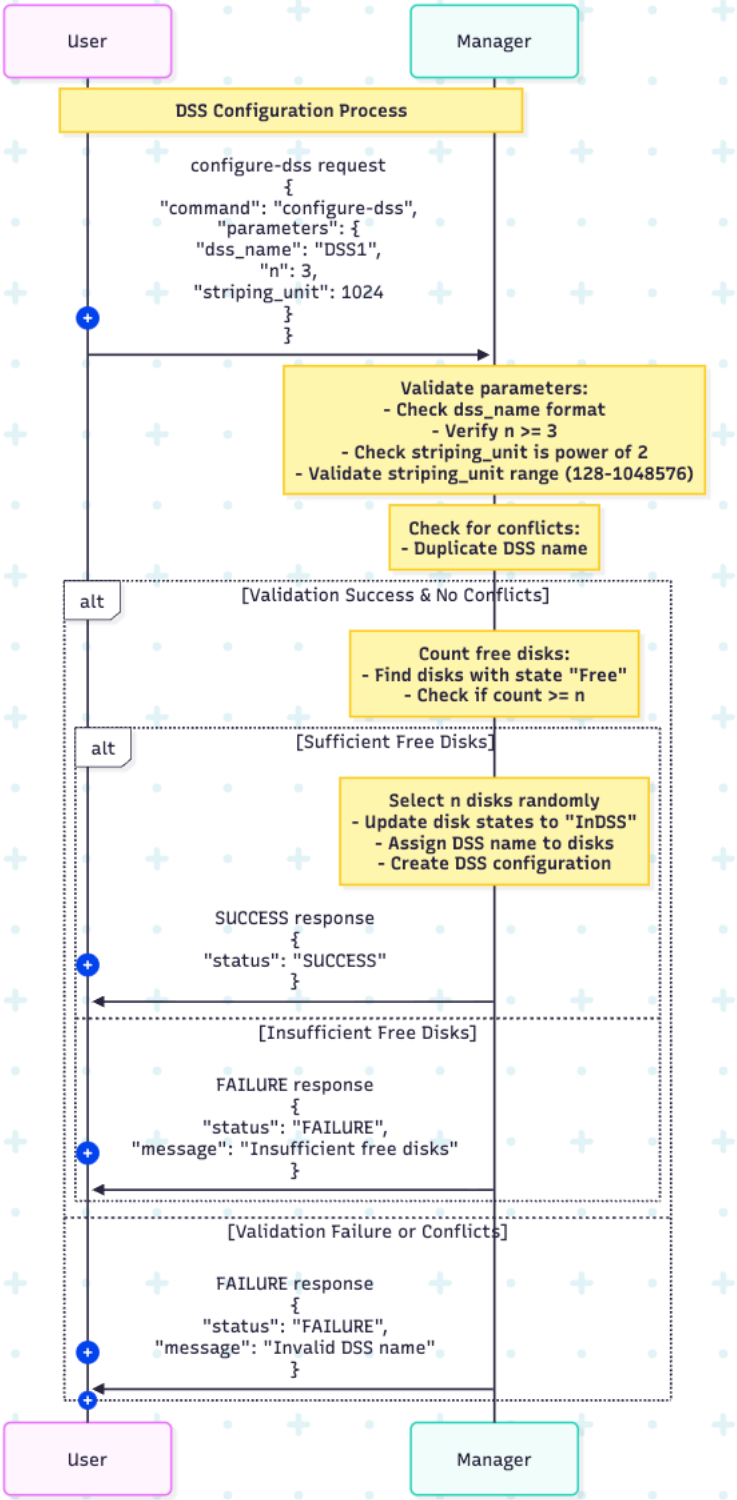


(b.ii) REGISTER-DISK Protocol Flow

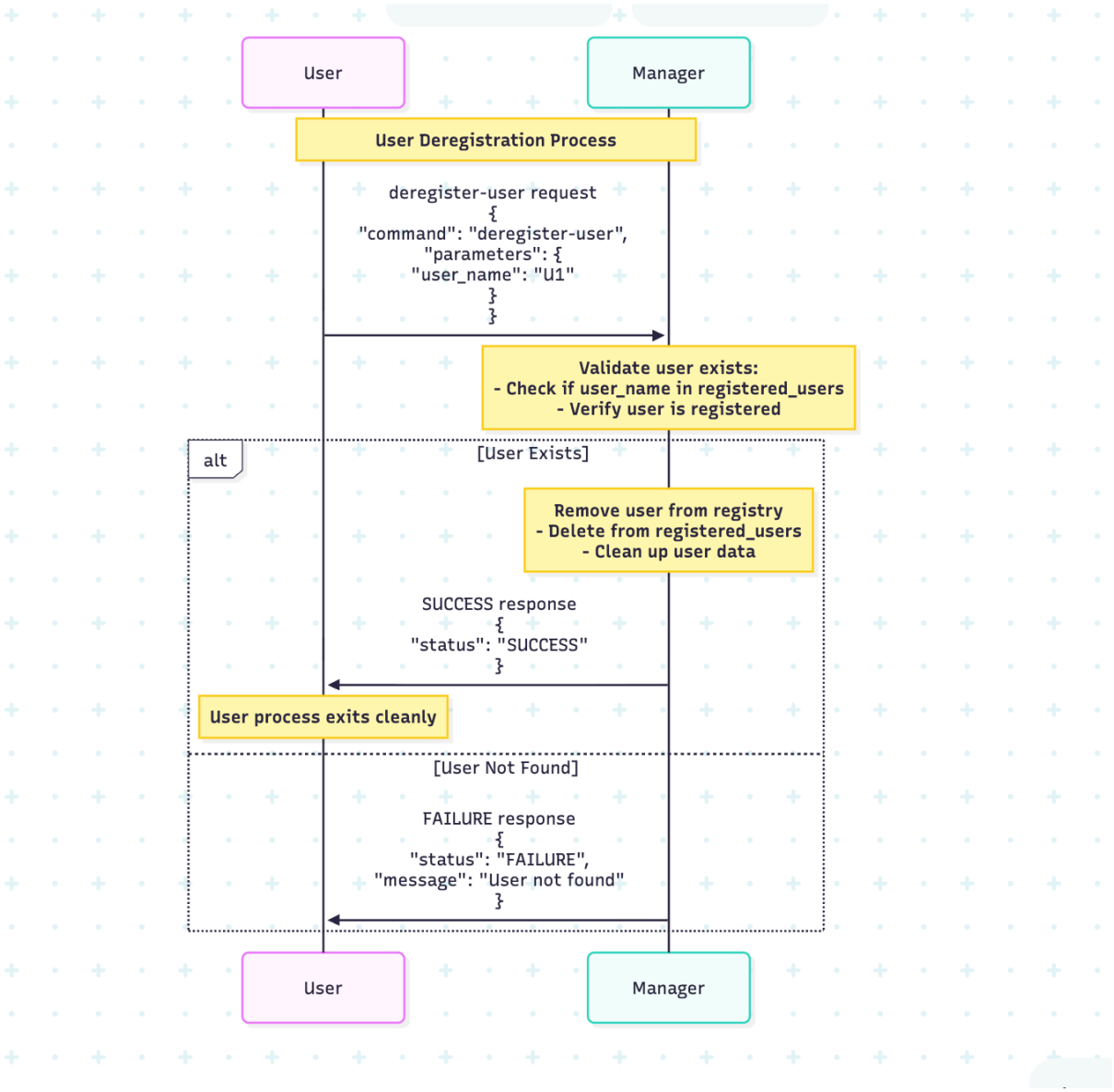


(b.iii)

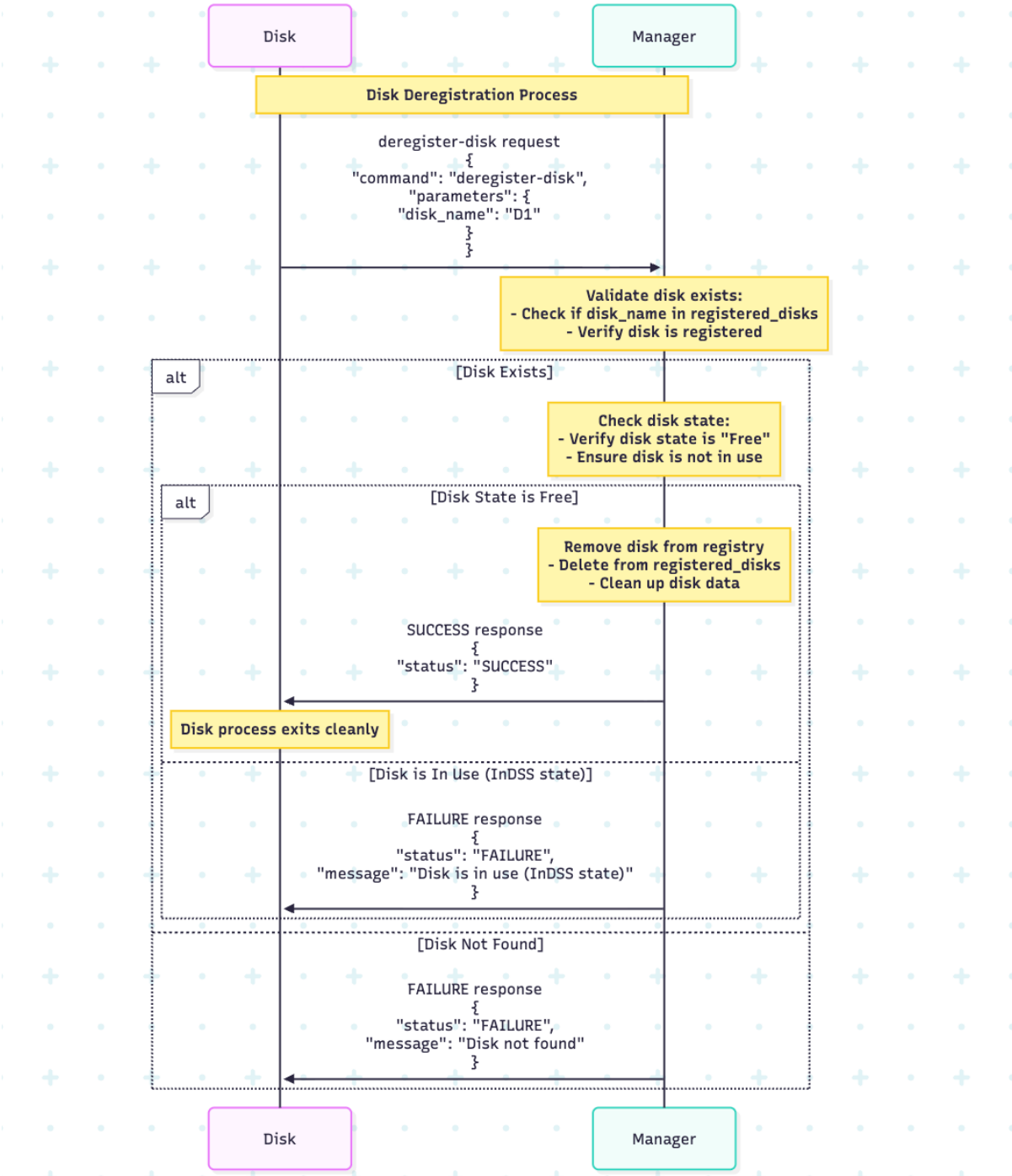
CONFIGURE-DSS Protocol Flow:


User
Manager
DSS Configuration Process
configure-dss request
{
"command": "configure-dss",
"parameters": {
"dss_name": "DSS1",
"n": 3,
"striping_unit": 1024
}
}
Validate parameters:
- Check dss_name format
- Verify n >= 3
- Check striping_unit is power of 2
- Validate striping_unit range (128-1048576)
Check for conflicts:
- Duplicate DSS name
alt
[Validation Success & No Conflicts]
Count free disks:
- Find disks with state "Free"
- Check if count >= n
alt
[Sufficient Free Disks]
Select n disks randomly
- Update disk states to "InDSS"
- Assign DSS name to disks
- Create DSS configuration
SUCCESS response
{
"status": "SUCCESS"
}
[Insufficient Free Disks]
FAILURE response
{
"status": "FAILURE",
"message": "Insufficient free disks"
}
[Validation Failure or Conflicts]
FAILURE response
{
"status": "FAILURE",
"message": "Invalid DSS name"
}
User
Manager

(b.iv) DEREGISTER-USER Protocol Flow:



(b.v) DEREGISTER-DISK Protocol Flow:



==========
(c) DATA STRUCTURES USED AND DESIGN DECISIONS MADE
==========

c.1 Manager State Information:

The Manager maintains three main data structures:

1. registered_users (dict):
   Key: user_name (string)

```
  Value: {
     "user_name": str,
     "ipv4_addr": str,
     "m_port": int,
     "c_port": int
  }

2. registered_disks (dict):
  Key: disk_name (string)
  Value: {
     "disk_name": str,
     "ipv4_addr": str,
     "m_port": int,
     "c_port": int,
     "state": "Free" | "InDSS",
     "dss_name": str (if InDSS)
  }

3. configured_dsses (dict):
  Key: dss_name (string)
  Value: {
     "dss_name": str,
     "n": int,
     "striping_unit": int,
     "disks": [list of disk names],
     "files": {} (for future file management)
  }
```

c.2 Design Decisions:

- JSON Message Format: Chosen for readability, debugging, and extensibility
- UDP Sockets: Selected for simplicity and stateless communication
- Port Range: Group 35 uses ports 18500-18999 (calculated from group number)
- Random Disk Selection: For configure-dss, disks are selected randomly from free disks
- State Management: Disks have "Free" or "InDSS" states to prevent conflicts
- Validation: Comprehensive parameter validation for all commands
- Error Handling: Consistent error responses with descriptive messages

c.3 Port Allocation Strategy:

- Manager: Uses command-line specified port (e.g., 18500)
- Users: Each user gets unique m_port and c_port
- Disks: Each disk gets unique m_port and c_port
- Port conflicts are checked during registration

c.4 Threading Model:

- Users and Disks use threading for concurrent management and command handling

- Manager uses single-threaded UDP server with message parsing
- Socket timeouts prevent blocking operations

==========
(d) GITHUB REPOSITORY SCREENSHOTS
==========

(d.i) Git Log Output:

```
sarjanpatel@Sarjans-MacBook-Pro cn p-1 % cd "/Users/sarjanpatel/cn p-1"
git log --pretty=format:"%h - %an, %ad (Commit) - %cd (Author)"
fd347de - Sarjan-Patel, Sun Sep 28 17:02:54 2025 -0700 (Commit) - Sun Sep 28 17:02:54 2025 -0700 (Author)
981e82a - Sarjan-Patel, Sun Sep 28 16:20:24 2025 -0700 (Commit) - Sun Sep 28 16:20:24 2025 -0700 (Author)
717f4fb - Sarjan-Patel, Sun Sep 28 10:57:51 2025 -0700 (Commit) - Sun Sep 28 10:57:51 2025 -0700 (Author)
sarjanpatel@Sarjans-MacBook-Pro cn p-1 %
```

(d.ii) Commit History Screenshot:

(d.iii) Git Reflog Output:

```
sarjanpatel@Sarjans-MacBook-Pro cn p-1 % cd "/Users/sarjanpatel/cn p-1"
git reflog
fd347de (HEAD -> main, origin/main) HEAD@{0}: Branch: renamed refs/heads/mai
n to refs/heads/main
fd347de (HEAD -> main, origin/main) HEAD@{2}: commit: Initial commit: Socket
 Programming Project - Distributed Storage System
981e82a HEAD@{3}: Branch: renamed refs/heads/master to refs/heads/main
981e82a HEAD@{5}: commit: Initial commit with project milestone code
717f4fb HEAD@{6}: commit (initial): Initial project setup with basic skeleto
n code for DSS project Group 35
sarjanpatel@Sarjans-MacBook-Pro cn p-1 %
```

d.4

```
sarjanpatel@Sarjans-MacBook-Pro cn p-1 % cd "/Users/sarjanpatel/cn p-1"
git fsck
Checking object directories: 100% (256/256), done.
sarjanpatel@Sarjans-MacBook-Pro cn p-1 %
```

==========
(e). VIdeo Link
==========

Video Link: https://youtu.be/WxtfClOhmJ0

Time Stamp:

0:00 - 00:38 (a) Compile your `manager`, `user` and `disk` programs (if applicable).

1:46 - 2:12 (b) Your milestone demo needs to use at least two distinct end-hosts.

0:38 - 1:45 (c) Start your `manager` program. Then start three `disk` processes and two `user` processes that each register with the `manager`. Be sure to assign port numbers from your port number space (see §3.3).

2:15 - 2:40 (d) Have one `user` issue a `configure-dss` command to build a DSS of size `n` = 3 with other parameters of your choice.

2:40 - 3:09 (e) Have the other `user` issue a `configure-dss` command; this should fail due to insufficient disks.

3:09 - 4:37 (f) Now deregister each `user` and `disk`, and terminate your `manager`