DISTRIBUTED LOAD BALANCING IN THE FACE OF REAPPEARANCE DEPENDENCIES

IMPLEMENTATION BASED ON AGRAWAL ET AL.'S 2024 PAPER

ABDULLAH KHALID & AHTISHAM UDDIN

INTRODUCTION

Problem Statement:

- Load balancing is an important task in distributed systems, when requests involve repeated data access patterns, creating reappearance dependencies that affect performance.

ALGORITHMS USED TO SOLVE THIS IN THE PAPER :

01

GREEDY APROACH

A simple approach that allocates incoming requests to the least loaded server.

02

DELAYED CUCKOO ROUTING

A more advanced approach that handles reappearance dependencies by deferring routing decisions, improving overall load distribution.

SYSTEM ARCHITECTURE

1

SERVERS

Multiple distributed server nodes handling requests.

2

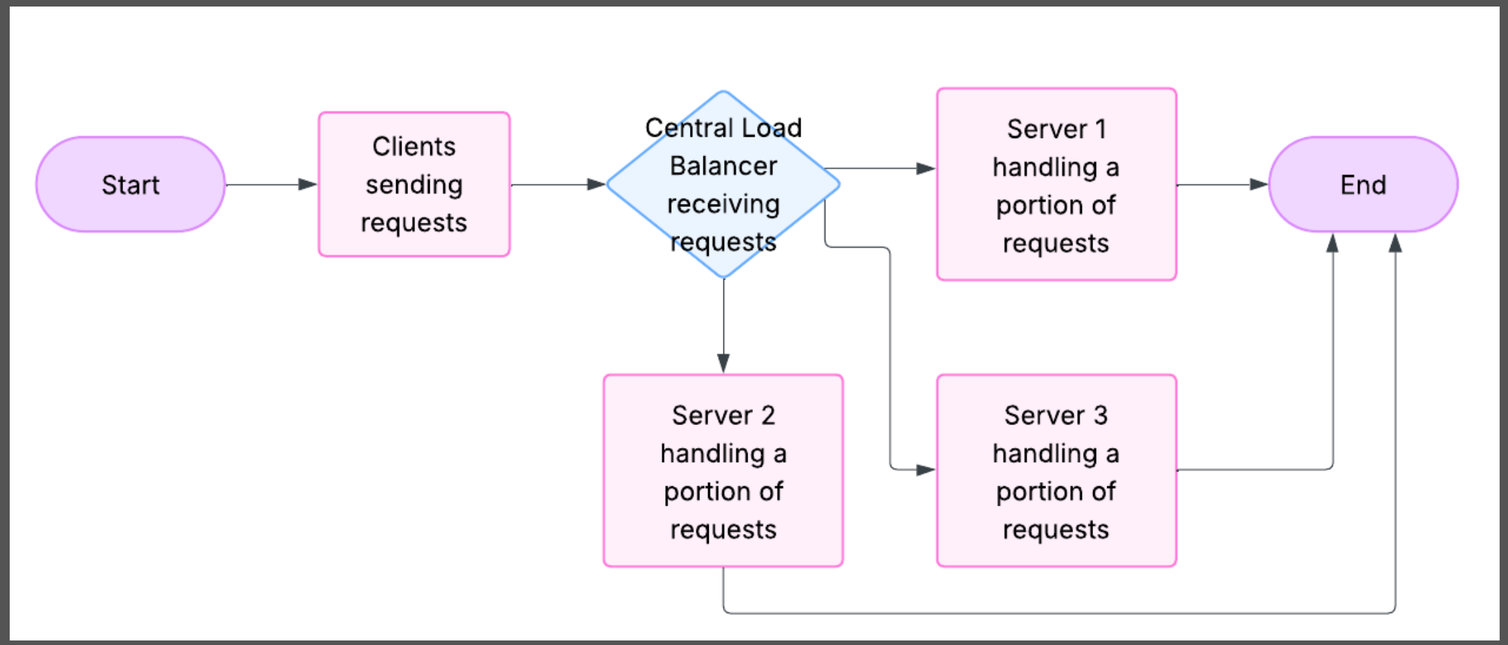
LOAD BALANCER

A central component directing requests to the servers

3

REAPPEARANCE DEPENDENCY DATASET

Carefully grafted reappearence dependency dataset



SYSTEM PARAMETERS

General System Parameters

- m = number of servers.
- q = queue size in each server, representing the maximum number of requests a server can handle before rejecting new ones.
- d = duplication factor, indicating the number of servers each chunk is replicated to.
- n = total number of data chunks.
- g = processing power of each server, or the number of requests each server can handle per time step.

Cuckoo Routing Parameters

- $\mathbf{Q}_i$: The queue at server i for handling requests.
- $\mathbf{P}_i$: The second queue at server i for handling requests that have already been requested in the current phase.
- Q'_i: The additional queue at server i for handling requests carried over from the previous phase.
- P'_i: The second additional queue at server i for requests carried over from the previous phase.
- $\mathbf{J}$: The phase index of the algorithm.
- $\mathbf{S}_i$: The set of chunks accessed during time step i.
- $\mathbf{T}_i$: The server assignment for each chunk $x \in S_i$ during time step i.

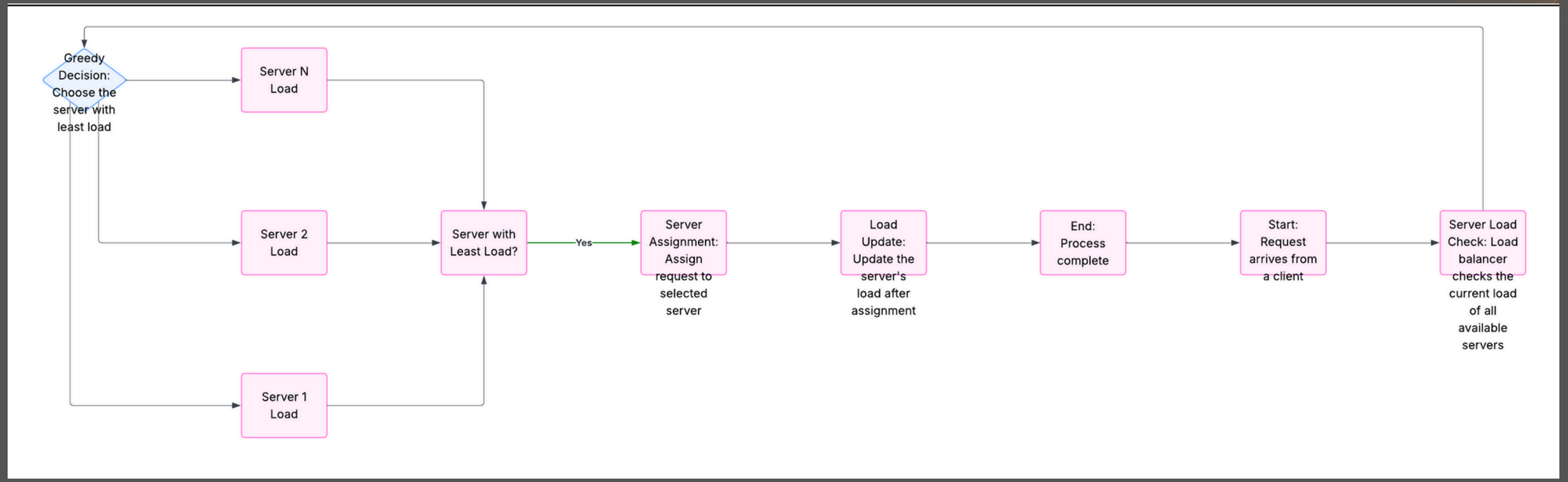
PERFORMANCE METRICS

- **Metrics Tracked:**
  - Latency: Time taken for a request to reach the server and get a response.Deduced from Average Queue size and Worst case Queue size
  - Rejection Rate: Percentage of requests that could not be processed due to server overload.
  - Throughput: The number of requests handled per unit time.

- σ = the set of client requests generated at each time step.
- $T_A(\sigma)$ = the number of accepted requests in the system.
- **Rejection Rate** = $\frac{|\sigma| - T_A(\sigma)}{|\sigma|}$, representing the fraction of requests rejected due to overloaded queues.
- **Latency** = $LA(\sigma_i)$, the number of time steps it takes for each request to be processed by the server.
- **Average Latency** = $\frac{\sum_i LA(\sigma_i)}{|\sigma|}$, the average time steps taken for all requests to be processed.

GREEDY ALGORITHM

- **What it is:**
  - An approach that assigns requests to the least loaded server in real-time, based on availability.
- **Implementation:**
  - Track server loads.
  - Assign requests to the least loaded server immediately.
- **Advantages:**
  - Simple and fast to implement.
  - Low computational overhead.
- **Limitations:**
  - May lead to higher rejection rates under heavy load.



DELAYED CUCKOO ROUTING

- **How it Works:**
  - Track Data Access Patterns:
  - The system tracks which data is accessed frequently across multiple servers.
- **Multiple Queues**
  - There multiple queues for each server which are kept free for the reapprearing chunks which cause rejection.
- **Predict Future Requests:**
  - It decides which queue of server for future requests based on past data access patterns.

BALLS AND BINS SETUP

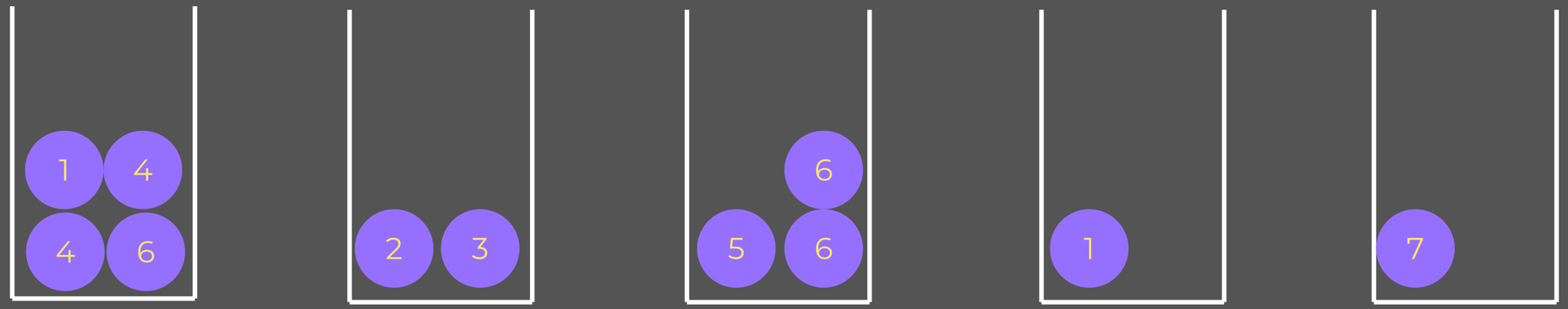
Number of chunks (n) = 7

Total Severs (m) = 5

Duplication per chunk (d) = 2

Server Processing (g) = 1

Queue sizes(q) = 2



Balls Allowed :

1,4,6

Balls Allowed :

2,3,9

Balls Allowed :

4,5,6,3

Balls Allowed :

1,2,7

Balls Allowed :

7,5

BALLS AND BINS

RANDOM APPROACH :

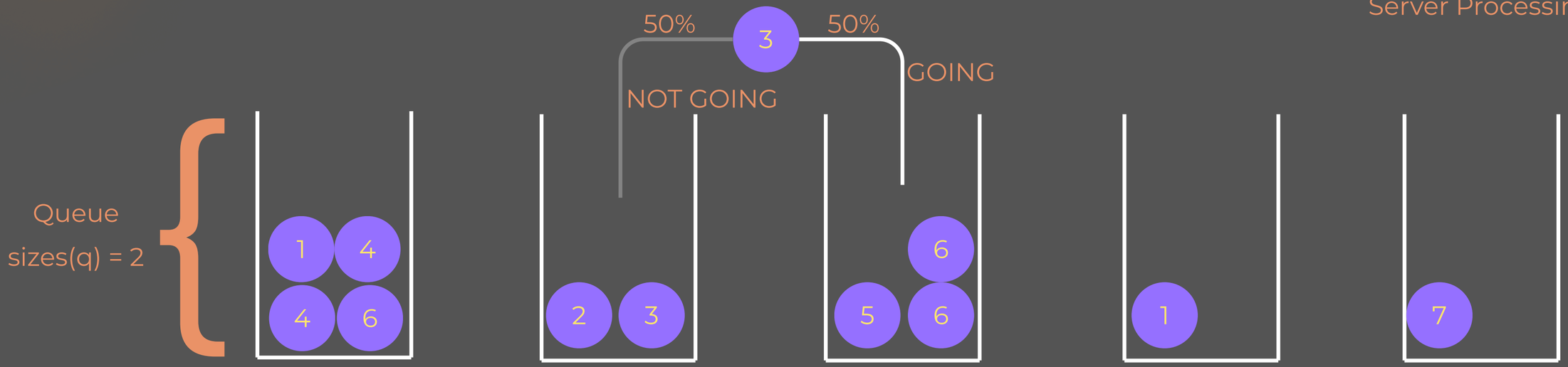
Randomly assign an upcoming Ball to any of the d(Duplicated) servers

Number of chunks (n) = 7

Total Severs (m) = 5

Duplication per chunk (d) = 2

Server Processing (g) = 1



Balls Allowed :

1,4,6

Balls Allowed :

2,3,9

Balls Allowed :

4,5,6,3

Balls Allowed :

1,2,7

Balls Allowed :

7,5

?

ISSUE ?

BALLS AND BINS

REAPPEARANCE DEPENDENCY

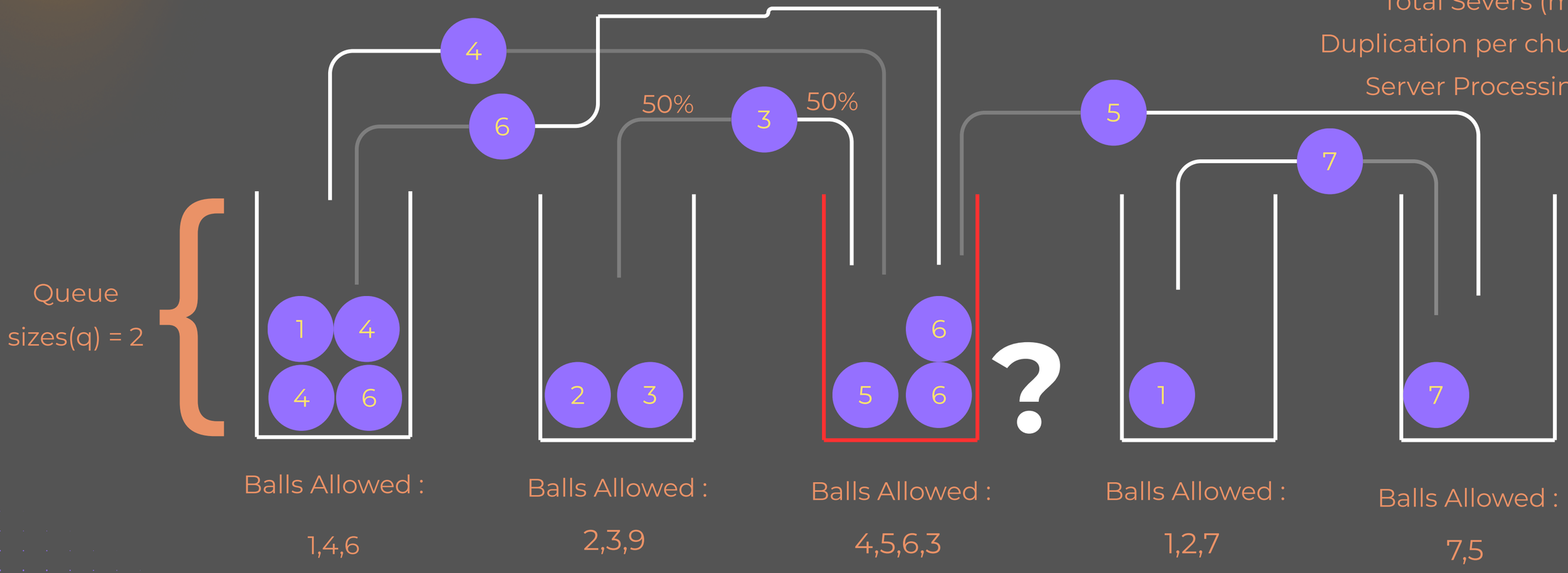
Lets Flood Servers again and again with 4,5,6,3,7

Number of chunks (n) = 7

Total Severs (m) = 5

Duplication per chunk (d) = 2

Server Processing (g) = 1



What will happen to server 3 After 5 reappearance dependencies?

REJECTIONS

BALLS AND BINS

GREEDY APPROACH

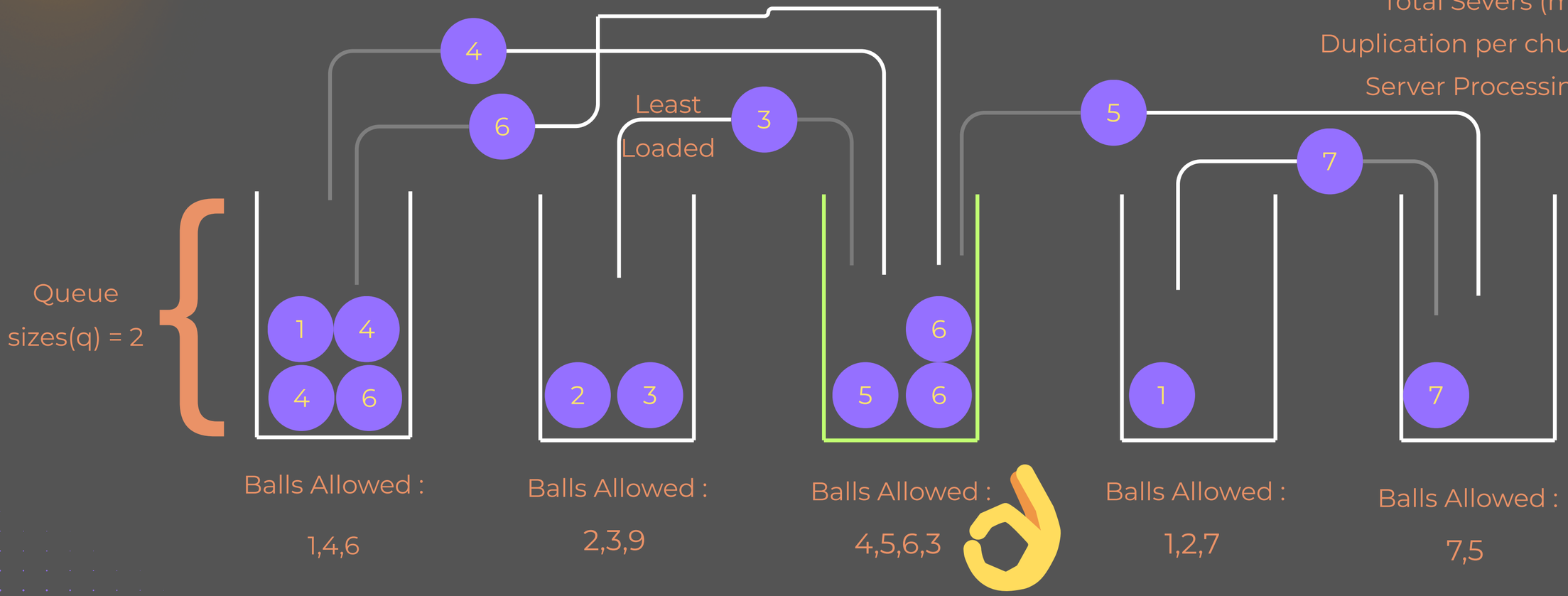
Throw Only to the least loaded bucket(Server)

Number of chunks (n) = 7

Total Severs (m) = 5

Duplication per chunk (d) = 2

Server Processing (g) = 1



- **Choosing the locally optimum options leads to better results RIGHT?**
- **Simply route to least loaded server**
- **Evenly fills up the buckets**
- **Does Local Optimum leads to Global Optimum???**
- **Is it a FUTURE PROOF approach?**

BALLS AND BINS

GREEDY APPROACH BOTTLENECK

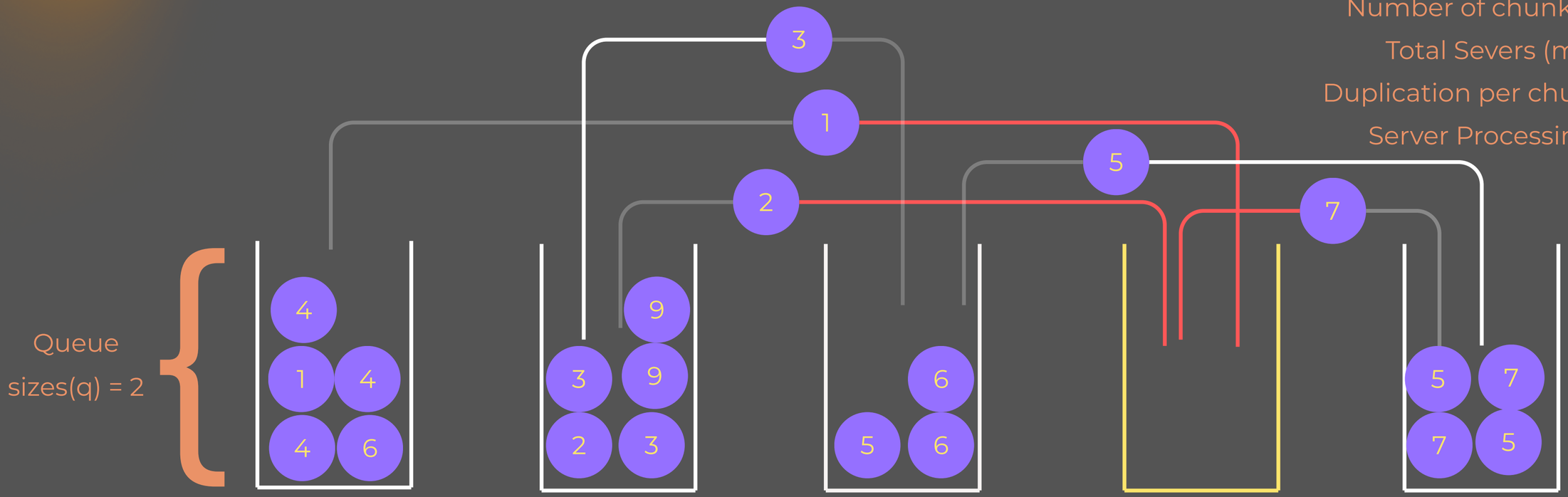
Lets use Reappearance Dependency 1,2,3,5,7

Number of chunks (n) = 7

Total Severs (m) = 5

Duplication per chunk (d) = 2

Server Processing (g) = 1



Balls Allowed :

1,4,6

Balls Allowed :

2,3,9

Balls Allowed :

4,5,6,3

Balls Allowed :

1,2,7

Balls Allowed :

7,5

BALLS AND BINS

GREEDY APPROACH BOTTLENECKS (CONT)

Number of chunks (n) = 8

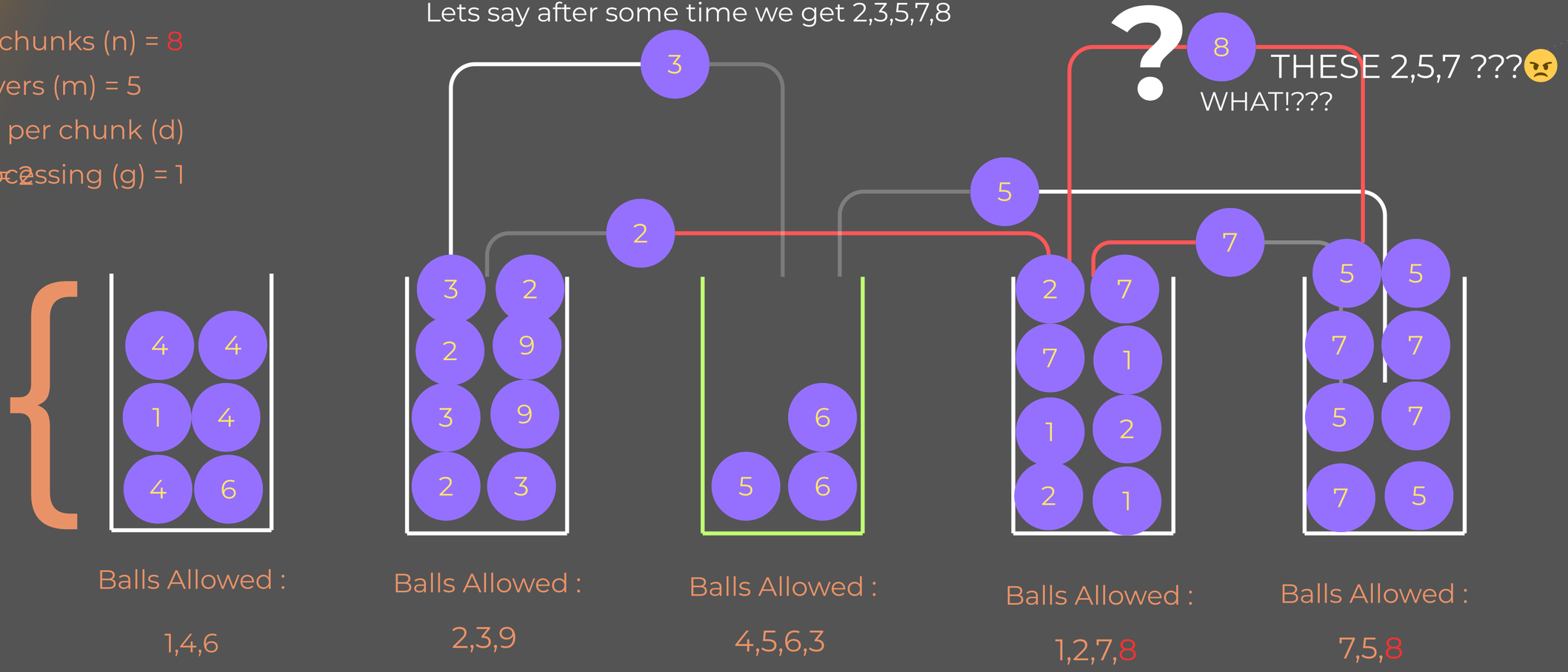
Total Severs (m) = 5

Duplication per chunk (d)

Server Processing (g) = 1

Queue sizes(q) = 2

Lets say after some time we get 2,3,5,7,8



Balls Allowed :

1,4,6

Balls Allowed :

2,3,9

Balls Allowed :

4,5,6,3

Balls Allowed :

1,2,7,8

Balls Allowed :

7,5,8

Why does 8 (newer chunks) have to pay for the mess created by reappearance dependencies (2,1,7)???)

Reappearance dependency are a problem even with greedy approach as they dont CARE for future Coming requests

GREEDY APPROACH Floods a single server if that is empty

GREEDY APPROACH NOT SO NICE

BALLS AND BINS

CUCKOO ROUTING

Number of chunks (n) = 8

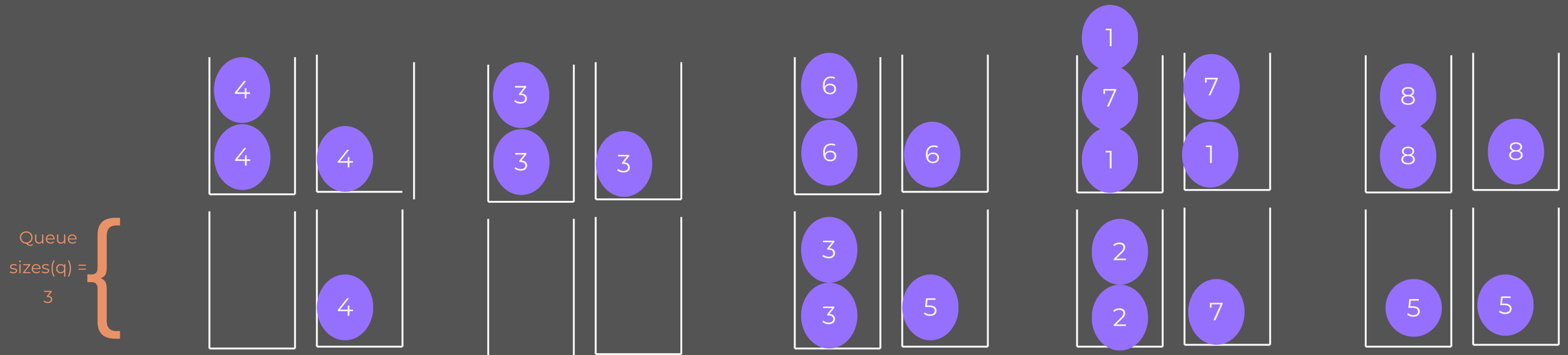
Total Severs (m) = 10

Duplication per chunk (d)= 2

Server Processing (g) = 1

Phase (j) = 3

Lets say after some time we get 2,3,5,7,8



Balls Allowed :

1,4,6

Balls Allowed :

2,3,9

Balls Allowed :

4,5,6,3

Balls Allowed :

1,2,7,8

Balls Allowed :

7,5,8

Why does 8 (newer chunks) have to pay for the mess created by reappearance dependencies (2,1,7)???)

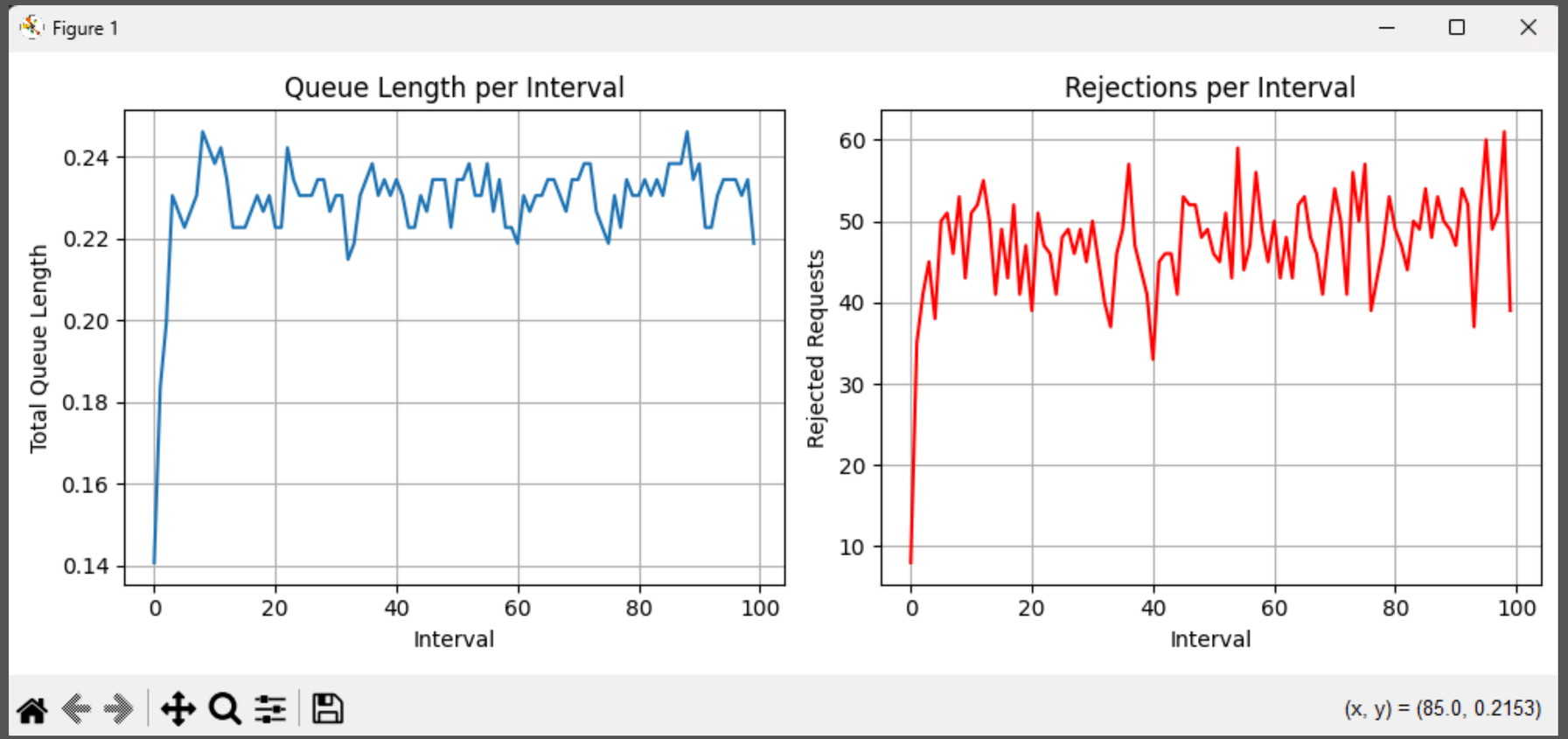
Reappearance dependency are a problem even with greedy approach as they dont CARE for future Coming requests

GREEDY APPROACH Floods a single server if that is empty

GREEDY APPROACH NOT SO NICE

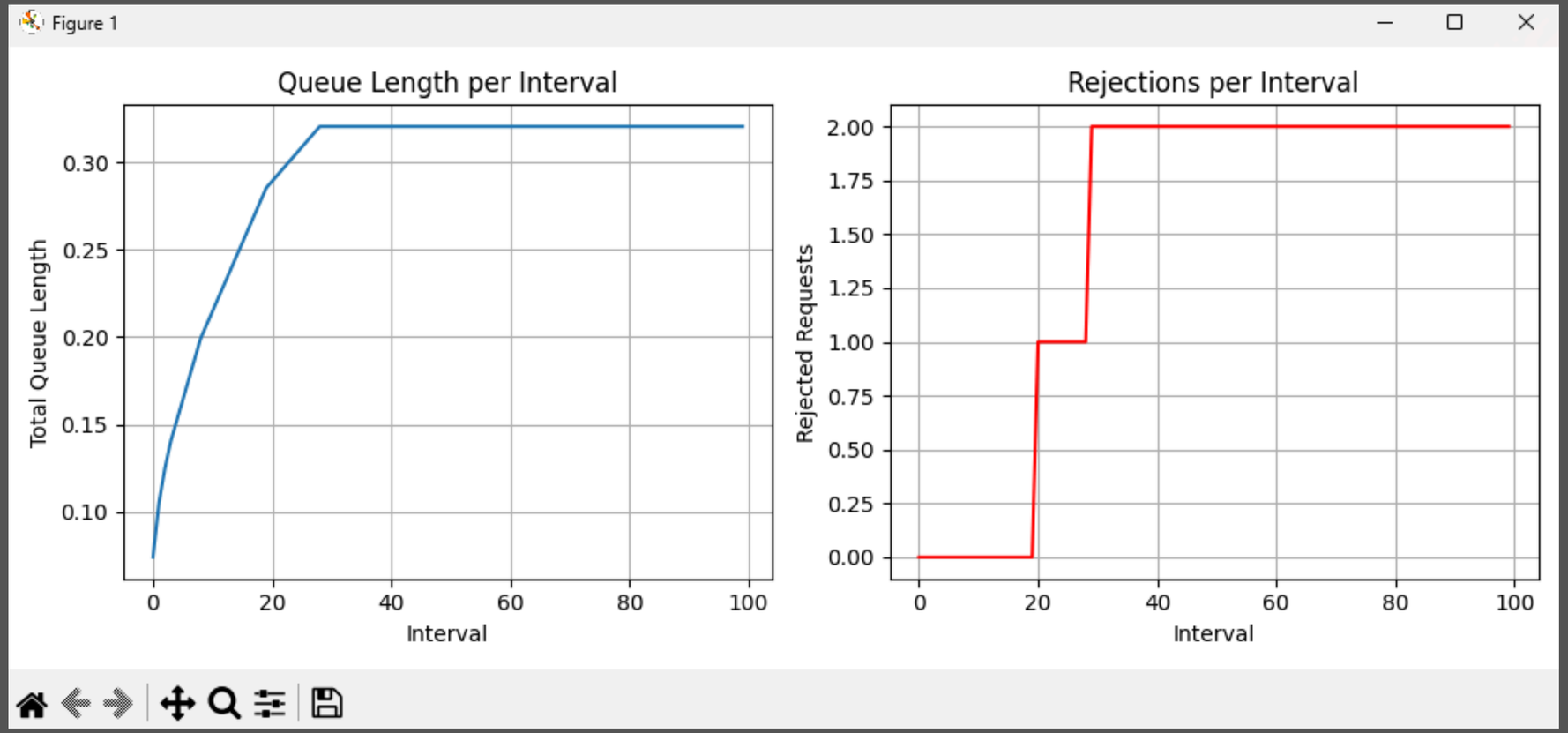
COMPARISON OF RESULTS

Random routing



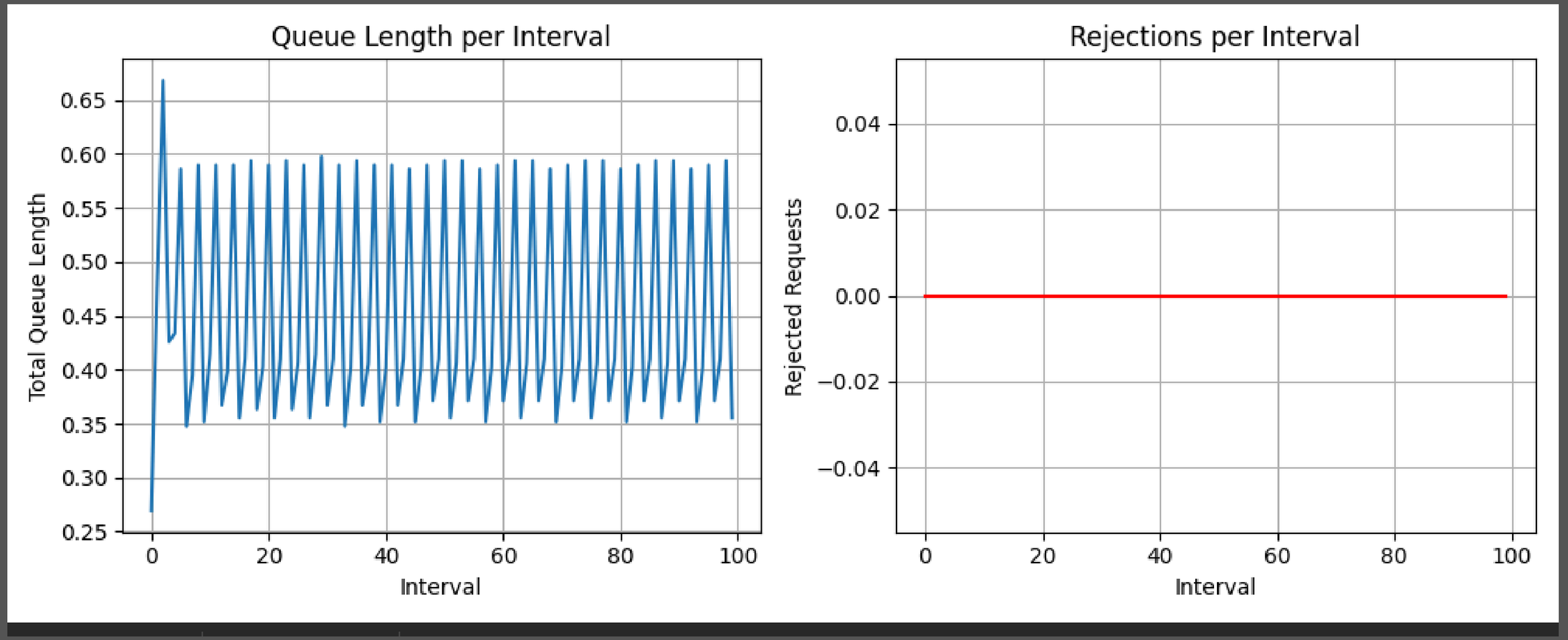

```
--- Simulation Summary ---
Total Requests: 25600
Accepted Requests: 20886
Rejected Requests: 4714
Rejection Rate: 0.1841
```

Greedy approach




```
--- Simulation Summary ---
Total Requests: 25600
Accepted Requests: 25449
Rejected Requests: 151
Rejection Rate: 0.0059
```

Cuckoo routing




```
--- Simulation Summary ---
Total Requests: 25600
Accepted Requests: 25600
Rejected Requests: 0
Rejection Rate: 0.0000
```

CHALLENGES ENCOUNTERED

- **Challenges:**
  - Handling Reappearance Dependencies: Effectively managing repeated data accesses and preventing server overload.
  - Scalability: Ensuring the system performs well as the number of requests and servers increases.
  - Algorithm Trade-Offs: Balancing between low latency and low rejection rates.
- **Solutions:**
  - Introduced tracking mechanisms for reappearance dependencies.
  - Evaluated performance under different load conditions to identify optimal algorithms.

CONCLUSION AND KEY TAKEAWAYS

- **Summary:**
  - Implemented load balancing algorithms based on the paper.
  - Achieved lower rejection rates with delayed cuckoo routing, but at the cost of slightly higher latency.
  - Greedy algorithm offers quick solutions but struggles with high rejection rates under heavy load.
- **Questions?**