In [8]:
```
import random
import time
from collections import deque
```

In [9]:
```
# Part 1: Simulating Client Requests
class ClientRequest:
    def __init__(self, request_id: int, arrival_time: float, processing_time:
        self.request_id = request_id
        self.arrival_time = arrival_time
        self.processing_time = processing_time

    def __repr__(self):
        return f"Request(ID={self.request_id}, Arrival={self.arrival_time}, Pr


def generate_client_request(request_id: int) -> ClientRequest:
    """Generate a new client request with random arrival and processing times.
    arrival_time = random.uniform(1, 5)  # Random arrival time between 1 and 5
    processing_time = random.uniform(1, 3)  # Random processing time between 1
    return ClientRequest(request_id, arrival_time, processing_time)
```

In [10]:
```
# Part 2: Round Robin Load Balancer
class Server:
    def __init__(self, server_id: int):
        self.server_id = server_id
        self.busy = False

    def __repr__(self):
        return f"Server(ID={self.server_id}, Busy={self.busy})"

class LoadBalancer:
    def __init__(self, servers: list):
        self.servers = servers

    def round_robin(self, request: ClientRequest):
        """Assign the request to the next server in the list (Round Robin)."""
        server = self.servers.pop(0)  # Get the first server in the list
        self.assign_request_to_server(server, request)
        self.servers.append(server)  # Put the server at the end of the list

    def assign_request_to_server(self, server: Server, request: ClientRequest)
        """Assign the request to a server."""
        server.busy = True
        print(f"Assigned {request} to {server}")
        time.sleep(request.processing_time)  # Simulate processing time
        self.release_server(server)

    def release_server(self, server: Server):
        """Release the server after processing the request."""
        server.busy = False
```

In [11]:
```
# Part 3: Simulation Loop
def simulate_requests(load_balancer: LoadBalancer, num_requests: int = 10):
    """Simulate client requests and process them using the Round Robin load ba
    for request_id in range(1, num_requests + 1):
        client_request = generate_client_request(request_id)
        load_balancer.round_robin(client_request)  # Use Round Robin algorithm
        time.sleep(random.uniform(0.5, 1.5))  # Random time between arrivals
```

In [12]:
```
# Running the simulation with 5 servers and a Round Robin load balancer
if __name__ == "__main__":
    # Initialize 5 servers
    servers = [Server(i) for i in range(1, 6)]
    load_balancer = LoadBalancer(servers)

    # Start the simulation with 10 client requests
    simulate_requests(load_balancer, num_requests=10)
```

```
Assigned Request(ID=1, Arrival=2.9888815814140277, Processing=1.603189130098
1316) to Server(ID=1, Busy=True)
Assigned Request(ID=2, Arrival=4.752978059046239, Processing=1.8130939610715
242) to Server(ID=2, Busy=True)
Assigned Request(ID=3, Arrival=3.147016992258956, Processing=1.4867099955517
873) to Server(ID=3, Busy=True)
Assigned Request(ID=4, Arrival=3.237211286243421, Processing=2.9593376721067
948) to Server(ID=4, Busy=True)
Assigned Request(ID=5, Arrival=2.2510423054153286, Processing=2.272065675036
159) to Server(ID=5, Busy=True)
Assigned Request(ID=6, Arrival=2.0834482015466174, Processing=2.225491843331
9926) to Server(ID=1, Busy=True)
Assigned Request(ID=7, Arrival=3.771062676319657, Processing=1.9336983300410
07) to Server(ID=2, Busy=True)
Assigned Request(ID=8, Arrival=1.0122589502528032, Processing=1.079309343473
6085) to Server(ID=3, Busy=True)
Assigned Request(ID=9, Arrival=1.7471984756500851, Processing=2.541472772185
2173) to Server(ID=4, Busy=True)
Assigned Request(ID=10, Arrival=1.0573010147351312, Processing=2.03605996762
0597) to Server(ID=5, Busy=True)
```

In [ ]: