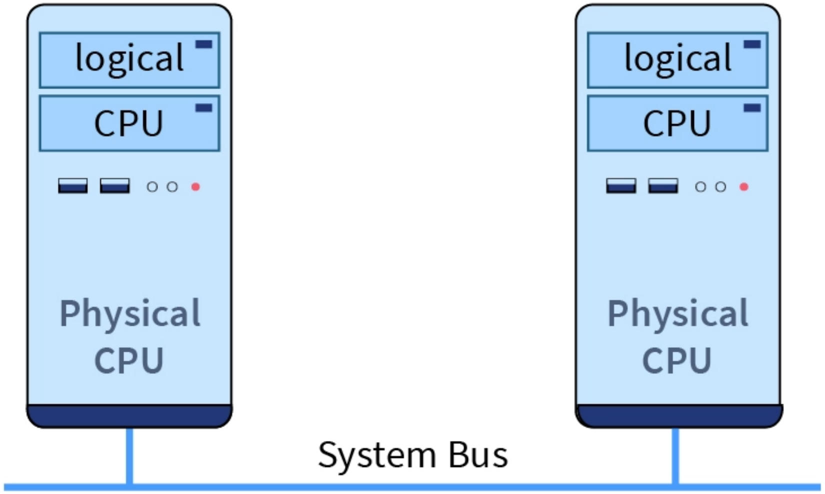
# Multiprocessor Scheduling & HU's Algorithm



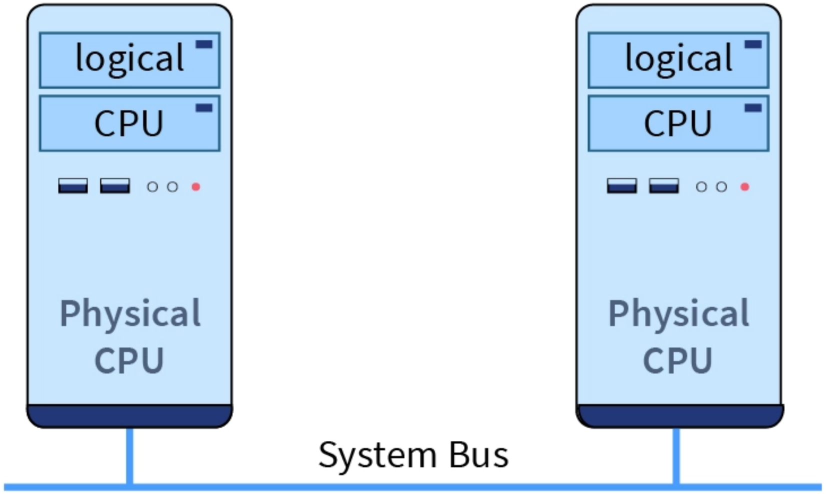

Abdullah Al Forkan
Matriculation Number: 2190314

**Challenges:**

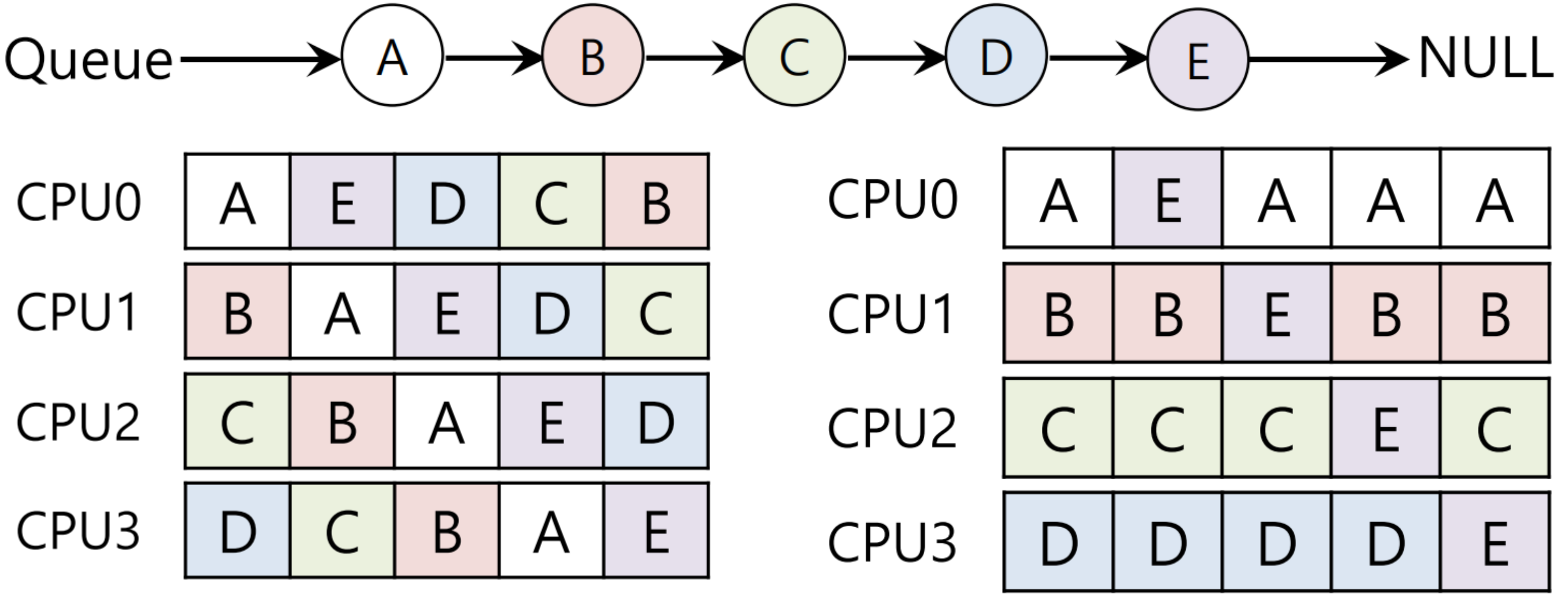
- Complex Problems
- Larger Dataset
- Slow Processors
- Parallel Programming

**Factors:**

- Powerful Processor
- System Performance
- Task Dependencies
- Optimal
- Short Time



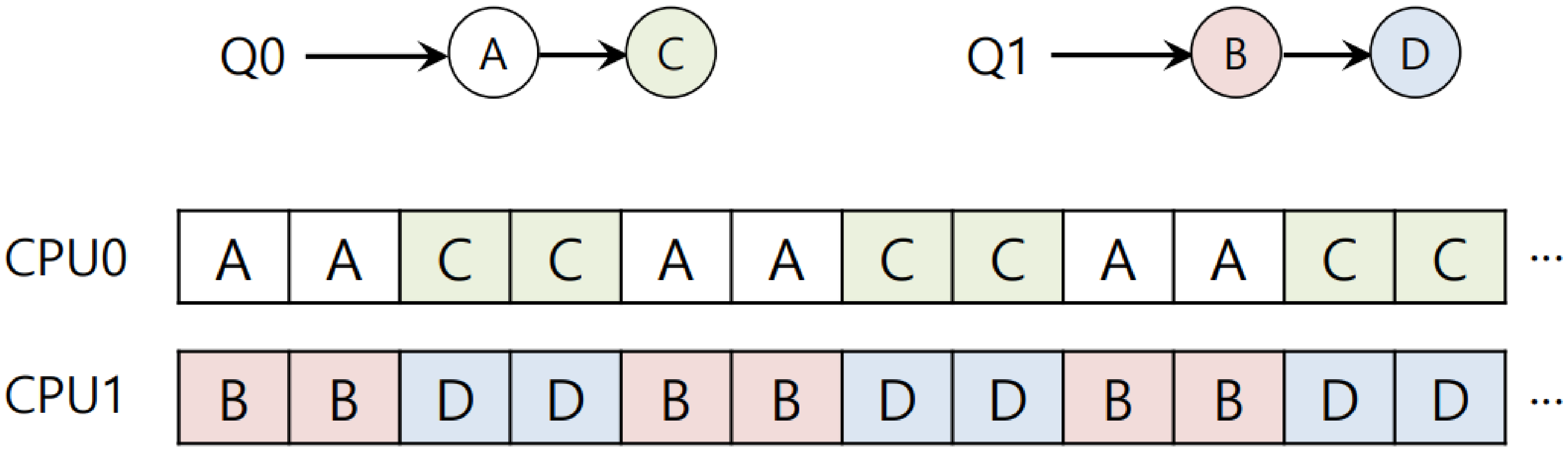
## Symmetric Multiprocessing




(a). SM with cache affinity

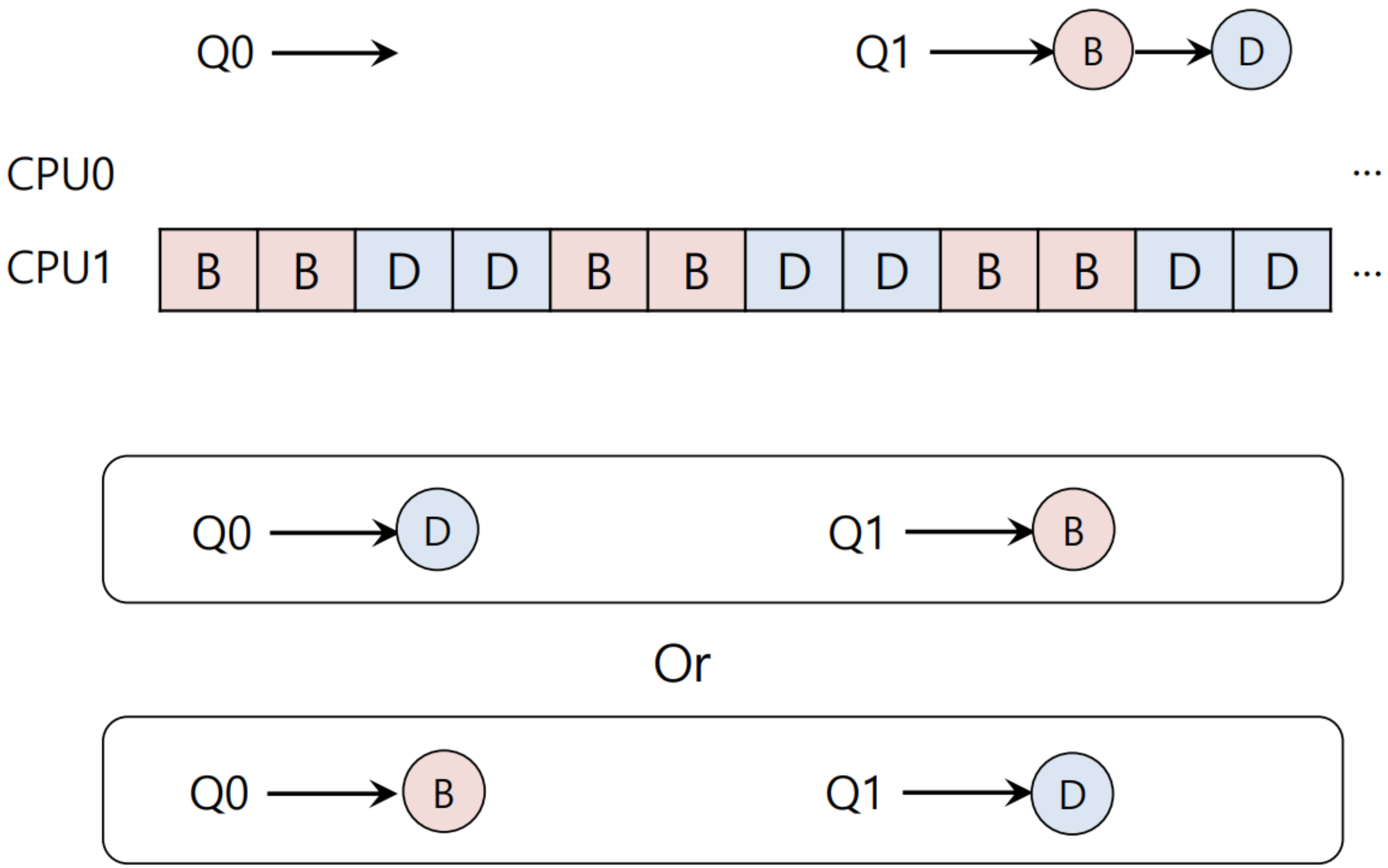
(b). Preserving affinity for most

## Asymmetric Multiprocessing



HOCHSCHULE HAMM-LIPPSTADT

## Load Balancing



# HU’s Algorithm

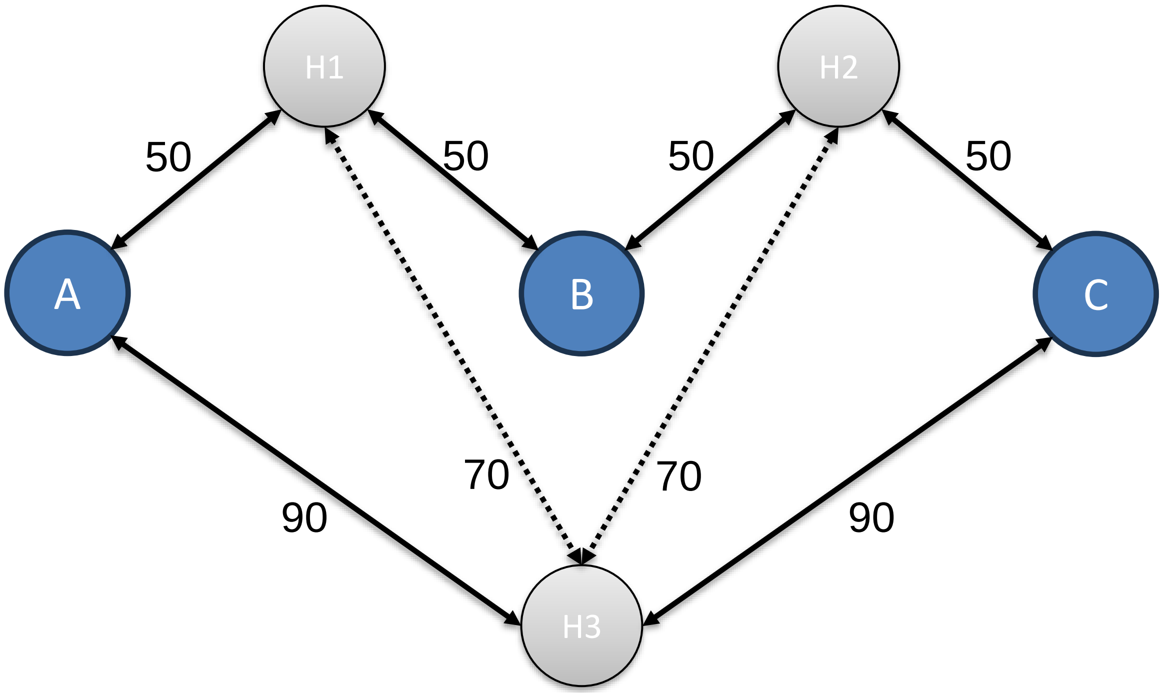
- Task dependencies
- Divided into multiple sub tasks.
- Recipe that has to be cocked to maintain an order
- The optimal
- Reduce the total time needed

## Task Graph & Assign Priorities

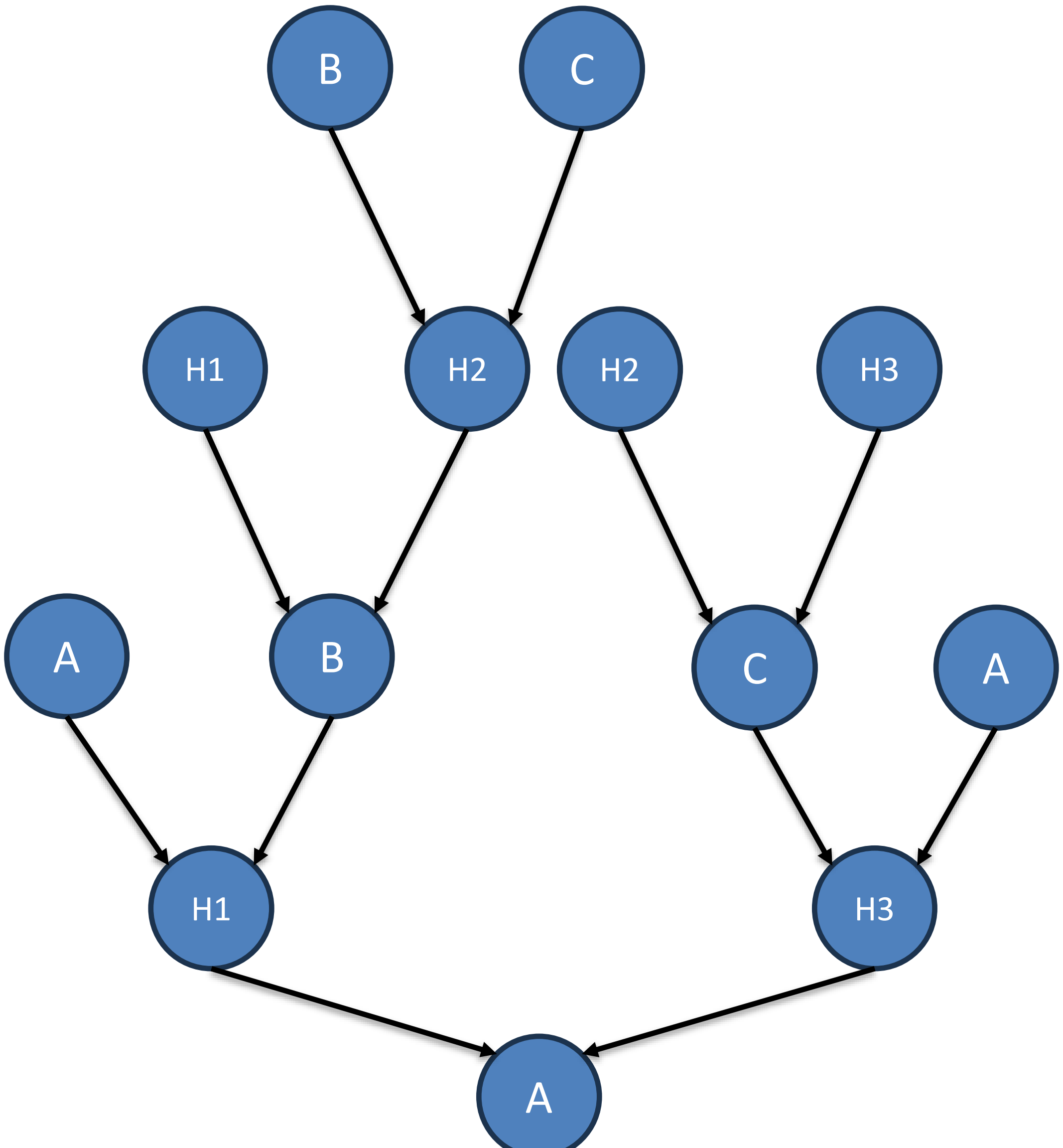
| City | Hub | Hub dependency |
|---|---|---|
| A | H1 | A, B |
| B | H2 | B, C |
| C | H3 | A, C |

HOCHSCHULE HAMM-LIPPSTADT

## Task Graph & Assign Priorities



# HU's Algorithmic Steps



HOCHSCHULE HAMM-LIPPSTADT

# 1. First Step

```python
import networkx as nx
import random

cities = ["A", "B", "C"]
hubs = ["H1", "H2", "H3"]

G = nx.DiGraph()

for node in cities + hubs:
    G.add_node(node)

G.add_edge("A", "H1", distance=50)
G.add_edge("B", "H1", distance=50)
G.add_edge("C", "H1", distance=150)
G.add_edge("H1", "H2", distance=100)
```

HOCHSCHULE HAMM-LIPPSTADT

## 1. Second Step

```python
containers = []
for i in range(10):
    origin = random.choice(cities)
    destination = origin
    while destination==origin:
        destination= random.choice(cities)
    containers.append((origin, destination))
```

```python
def calculate_priority(origin, destination):
    try:
        distance = nx.shortest_path_length(G, origin, destination, weight='distance')
    except nx.NetworkXNoPath:
        return float('inf')  # Highest priority if no path (so it gets filtered out)
    return distance
```

## 1. Third Step

```python
schedule = []
for container in containers:
    origin, destination = container
    try:
        distance, path = nx.single_source_dijkstra(G, origin, destination, weight='distance')
```

```python
for container, path, _ in schedule:
    print(f"Moving container {container} along shortest path: {path}")
```

## Result

```
C:\Users\mdfor\anaconda3\e
Moving container ('C', 'B') along shortest path: ['C', 'H2', 'B']
Moving container ('B', 'A') along shortest path: ['B', 'H1', 'A']
Moving container ('B', 'C') along shortest path: ['B', 'H2', 'C']
Moving container ('A', 'B') along shortest path: ['A', 'H1', 'B']
Moving container ('A', 'B') along shortest path: ['A', 'H1', 'B']
Moving container ('A', 'B') along shortest path: ['A', 'H1', 'B']
Moving container ('B', 'C') along shortest path: ['B', 'H2', 'C']
Moving container ('B', 'C') along shortest path: ['B', 'H2', 'C']
Moving container ('C', 'A') along shortest path: ['C', 'H3', 'A']
Moving container ('C', 'A') along shortest path: ['C', 'H3', 'A']
Press any key to continue . . .
```