Sep 18, 2024

CS 350 DISCUSSION 2

Processes, Resources and Little’s Law

Processes and Resources

Consider a process that consists of the following phases to complete its workload:

1. **CPU usage** for **6ms**
2. Blocking **network I/O** request that takes **6ms** to complete
3. **CPU usage** for **2ms**
4. Blocking **disk I/O** request that takes **10ms** to complete

Processes and Resources

Consider a process that consists of the following phases to complete its workload:

1. **CPU usage** for **6ms**
2. Blocking **network I/O** request that takes **6ms** to complete
3. **CPU usage** for **2ms**
4. Blocking **disk I/O** request that takes **10ms** to complete

Let's visualize a single execution of such process:

resource\time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	23	23	24
CPU																								
Network																								
Disk																								

Processes and Resources

Consider a process that consists of the following phases to complete its workload:

1. **CPU usage** for **6ms**
2. Blocking **network I/O** request that takes **6ms** to complete
3. **CPU usage** for **2ms**
4. Blocking **disk I/O** request that takes **10ms** to complete

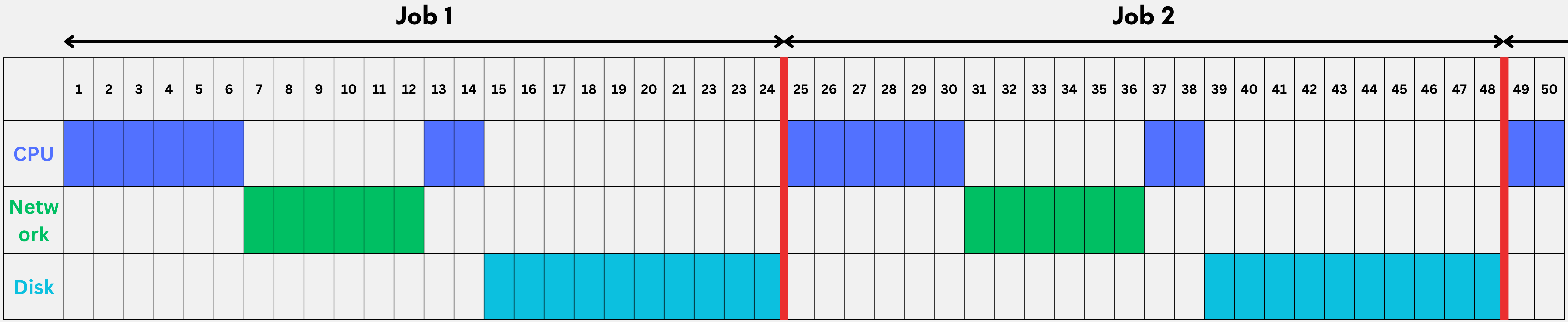
Now let's assume the **multiprogramming level** of our system, **MPL**, is **1**. This means that we are allowed to have only **one active processes** at a time. Once a process finishes, a new process with the same workload starts again.

Processes and Resources

Consider a process that consists of the following phases to complete its workload:

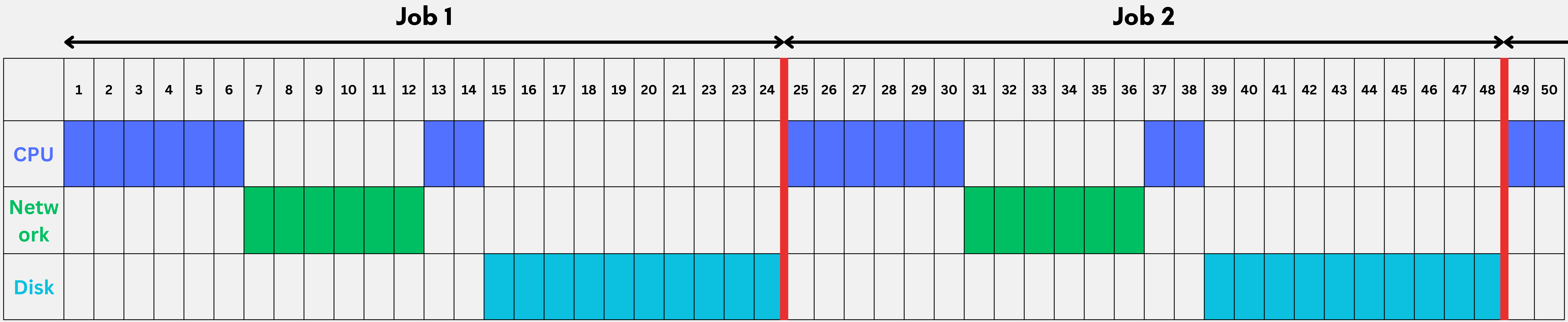
1. **CPU usage** for **6ms**
2. Blocking **network I/O** request that takes **6ms** to complete
3. **CPU usage** for **2ms**
4. Blocking **disk I/O** request that takes **10ms** to complete

Now let's assume the **multiprogramming level** of our system, **MPL**, is **1**. This means that we are allowed to have only **one active processes** at a time. Once a process finishes, a new process with the same workload starts again.



Processes and Resources

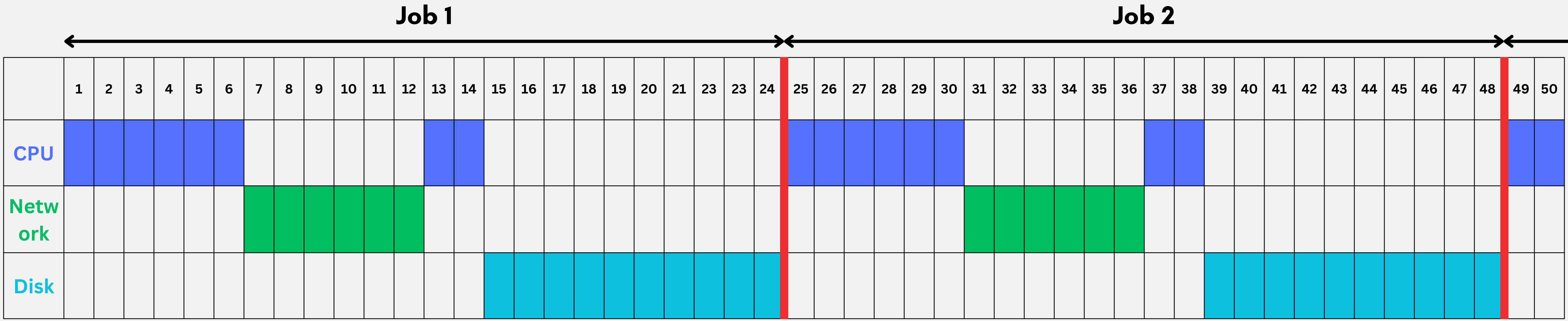
Now let's assume the **multiprogramming level** of our system, **MPL**, is **1**. This means that we are allowed to have only **one active processes** at a time. Once a process finishes, a new process with the same workload starts again.



Consider the following questions:

1. What is the **steady-state utilization** of each resource?
2. What is the **throughput** of the system?

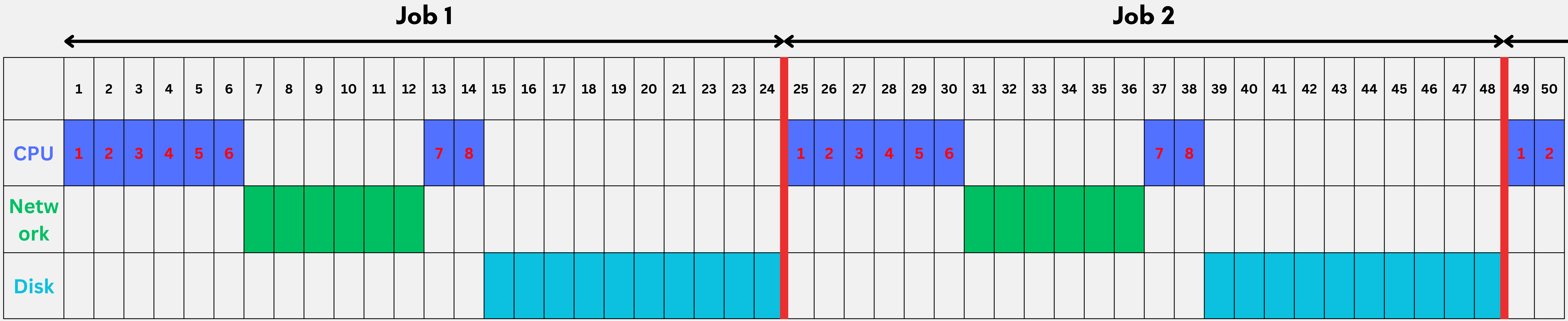
Processes and Resources



Steady-State Utilization:

- CPU
- Network
- Disk

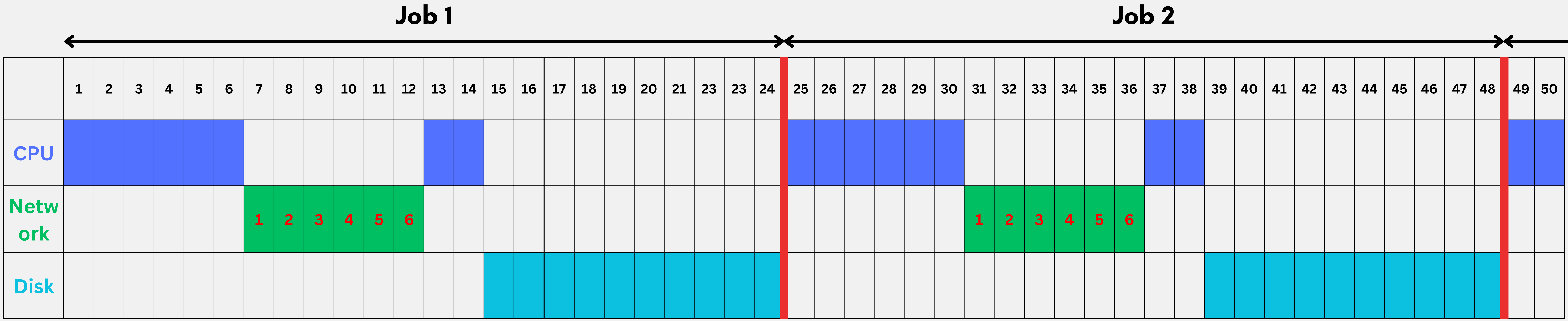
Processes and Resources



Steady-State Utilization:

- CPU = 8/24
- Network
- Disk

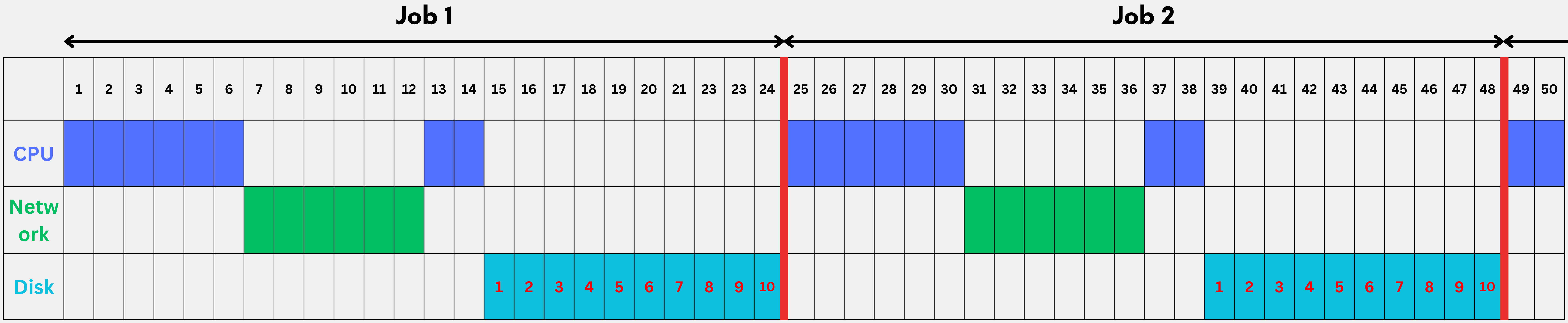
Processes and Resources



Steady-State Utilization:

- CPU = 8/24
- Network = 6/24
- Disk

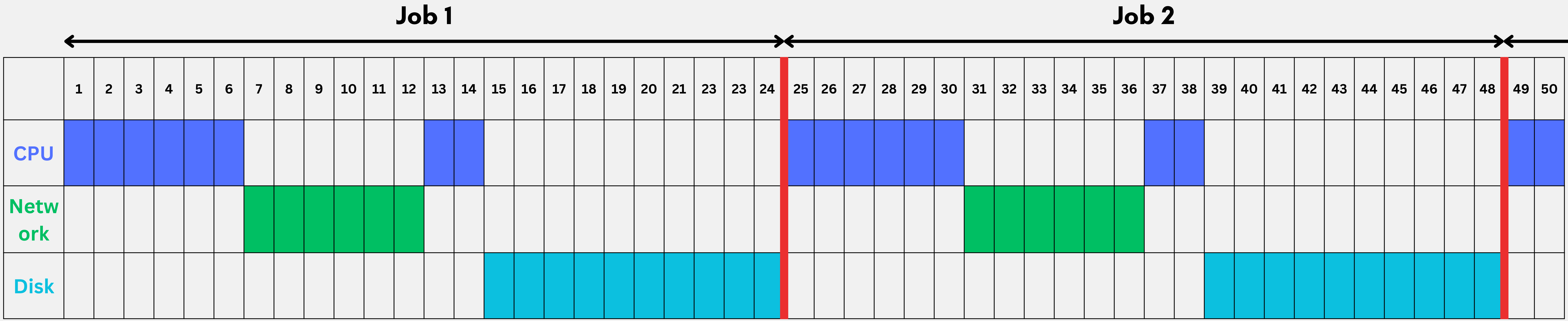
Processes and Resources



Steady-State Utilization:

- CPU = 8/24
- Network = 6/24
- Disk = 10/24

Processes and Resources

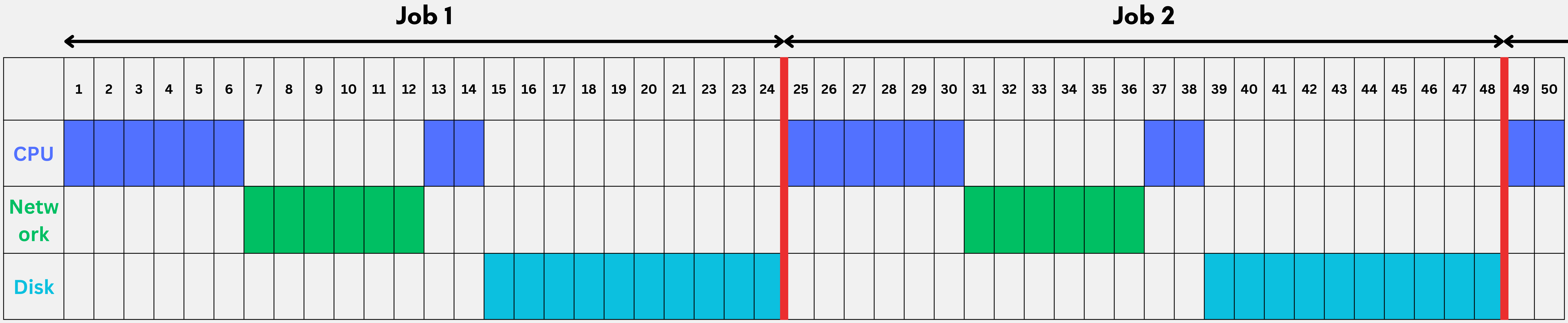


Steady-State Utilization:

- CPU = 8/24
- Network = 6/24
- Disk = 10/24

Throughput:

Processes and Resources



Steady-State Utilization:

- CPU = 8/24
- Network = 6/24
- Disk = 10/24

Throughput:

1/24 processes per millisecond (~41.67 proc/sec)

Processes and Resources

Consider a process that consists of the following phases to complete its workload:

1. **CPU usage** for **6ms**
2. Blocking **network I/O** request that takes **6ms** to complete
3. **CPU usage** for **2ms**
4. Blocking **disk I/O** request that takes **10ms** to complete

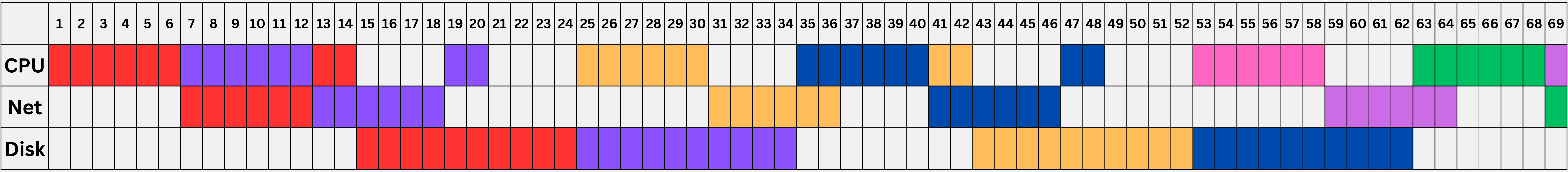
Now let's assume the **multiprogramming level** of our system, **MPL**, is **2**. This means that we are allowed to have only **two active processes** at a time. Two processes are started together at *t=0*, once a process finishes, a new one is started immediately.

Processes and Resources

Consider a process that consists of the following phases to complete its workload:

1. **CPU usage** for **6ms**
2. Blocking **network I/O** request that takes **6ms** to complete
3. **CPU usage** for **2ms**
4. Blocking **disk I/O** request that takes **10ms** to complete

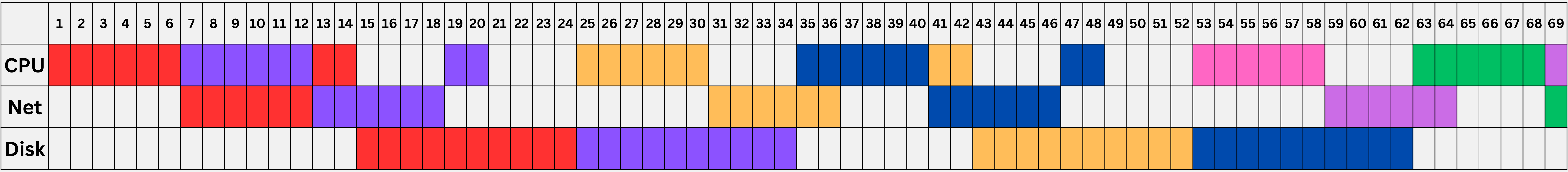
Now let's assume the **multiprogramming level** of our system, **MPL**, is **2**. This means that we are allowed to have only **two active processes** at a time. Two processes are started together at *t=0*, once a process finishes, a new one is started immediately.



**each color corresponds to a new instance of the task*

Processes and Resources

Now let's assume the **multiprogramming level** of our system, **MPL**, is **2**. This means that we are allowed to have only **two active processes** at a time. Two processes are started together at *t=0*, once a process finishes, a new one is started immediately.



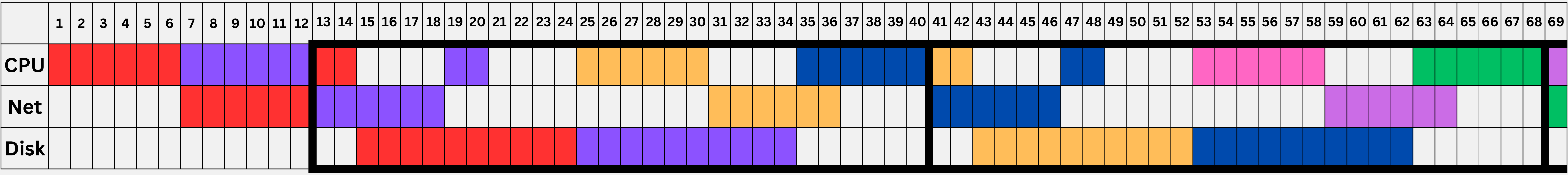
**each color corresponds to a new instance of the task*

Let's answer the same questions as before:

1. What is the **steady-state utilization** of each resource?
2. What is the **throughput** of the system?

Processes and Resources

Now let's assume the **multiprogramming level** of our system, **MPL**, is **2**. This means that we are allowed to have only **two active processes** at a time. Two processes are started together at *t=0*, once a process finishes, a new one is started immediately.

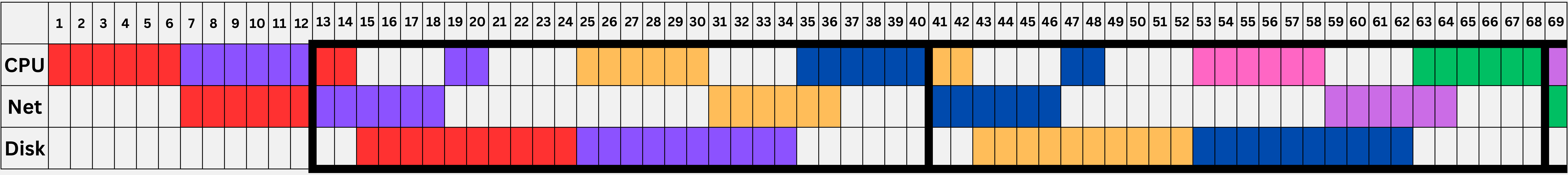


**each color corresponds to a new instance of the task*

Let's answer the same questions as before:

1. What is the **steady-state** **utilization** of each resource?
2. What is the **throughput** of the system?

Processes and Resources

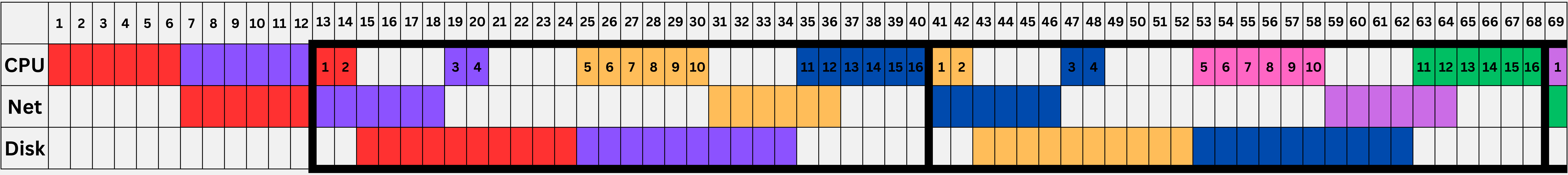


**each color corresponds to a new instance of the task*

Steady-State Utilization:

- **CPU**
- **Network**
- **Disk**

Processes and Resources

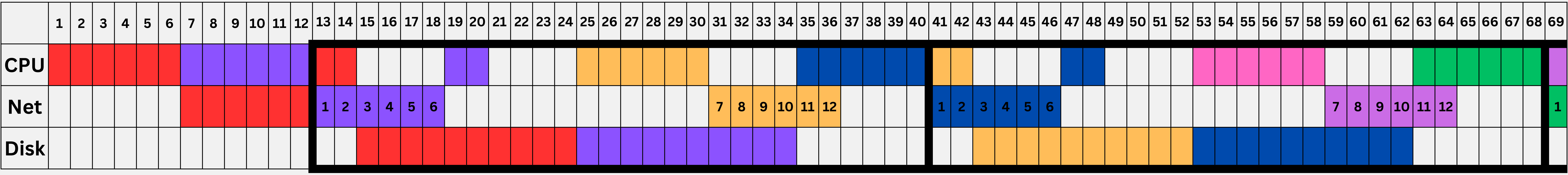


**each color corresponds to a new instance of the task*

Steady-State Utilization:

- **CPU = 16 / (40 - 13 + 1) = 16/28**
- **Network**
- **Disk**

Processes and Resources



**each color corresponds to a new instance of the task*

Steady-State Utilization:

- CPU = 16/28
- Network = 12 / (40 - 13 + 1) = 12/28
- Disk

Processes and Resources

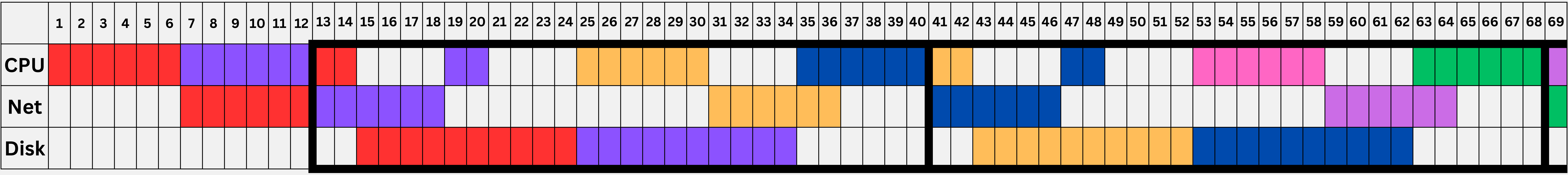
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69
CPU																																																																					
Net																																																																					
Disk															1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20									1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20							

**each color corresponds to a new instance of the task*

Steady-State Utilization:

- **CPU = 16/28**
- **Network = 12/28**
- **Disk = 20 / (40 - 13 + 1) = 20/28**

Processes and Resources



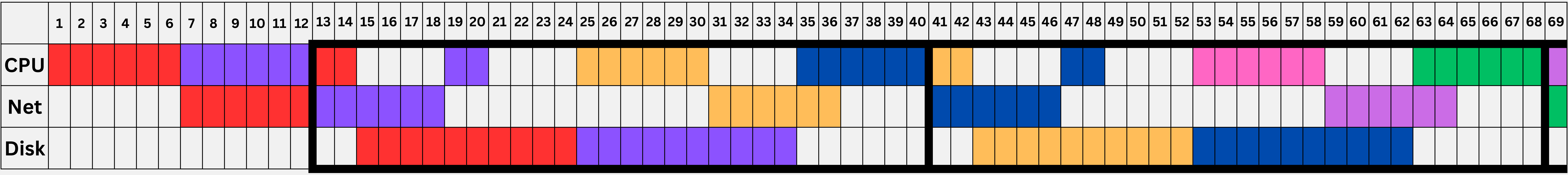
each color corresponds to a new instance of the task

Steady-State Utilization:

- CPU = 16/28
- Network = 12/28
- Disk = 20/28

Throughput:

Processes and Resources



**each color corresponds to a new instance of the task*

Steady-State Utilization:

- **CPU = 16/28**
- **Network = 12/28**
- **Disk = 20/28**

Throughput:

2/28 processes per millisecond (~71.43 proc/sec)

Processes and Resources

Consider a process that consists of the following phases to complete its workload:

1. **CPU usage** for **6ms**
2. Blocking **network I/O** request that takes **6ms** to complete
3. **CPU usage** for **2ms**
4. Blocking **disk I/O** request that takes **10ms** to complete

What is the **capacity**, the maximum throughput that can be sustained, of the system?

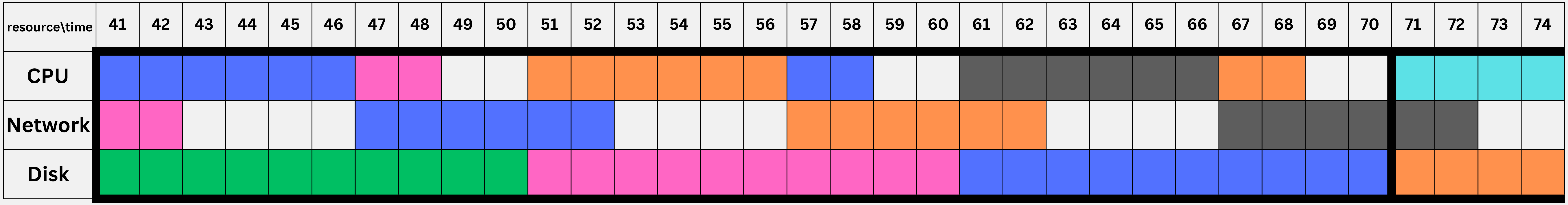
Processes and Resources

Consider a process that consists of the following phases to complete its workload:

1. **CPU usage** for **6ms**
2. Blocking **network I/O** request that takes **6ms** to complete
3. **CPU usage** for **2ms**
4. Blocking **disk I/O** request that takes **10ms** to complete

What is the **capacity**, the maximum throughput that can be sustained, of the system?

At **MPL=3**, our system achieves the following **steady-state:**

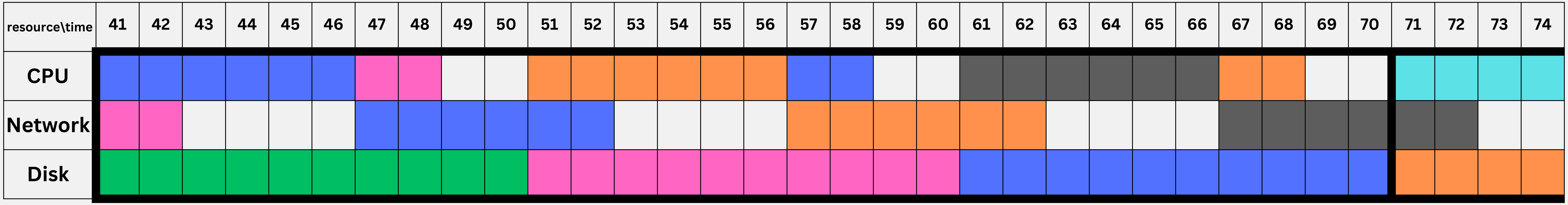


**each color corresponds to a new instance of the task*

Processes and Resources

What is the **capacity**, the maximum throughput that can be sustained, of the system?

At **MPL=3**, our system achieves the following **steady-state:**



**each color corresponds to a new instance of the task*

Steady-State Utilization:

- **CPU**
- **Network**
- **Disk**

Processes and Resources

What is the **capacity**, the maximum throughput that can be sustained, of the system?

At **MPL=3**, our system achieves the following **steady-state:**

resource\time	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74
CPU	1	2	3	4	5	6	7	8			9	10	11	12	13	14	15	16			17	18	19	20	21	22	23	24			1	2	3	4
Network	1	2					3	4	5	6	7	8					9	10	11	12	13	14					15	16	17	18	1	2		
Disk	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	1	2	3	4

**each color corresponds to a new instance of the task*

Steady-State Utilization:

- **CPU = 24 / (70 - 41 + 1) = 0.8**
- **Network = 18 / (70 - 41 + 1) = 0.6**
- **Disk = 30 / (70 - 41 + 1) = 1**

Processes and Resources

What is the **capacity**, the maximum throughput that can be sustained, of the system?

At **MPL=3**, our system achieves the following **steady-state:**

resource\time	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74
CPU	1	2	3	4	5	6	7	8			9	10	11	12	13	14	15	16			17	18	19	20	21	22	23	24			1	2	3	4
Network	1	2					3	4	5	6	7	8					9	10	11	12	13	14					15	16	17	18	1	2		
Disk	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	1	2	3	4

**each color corresponds to a new instance of the task*

Steady-State Utilization:

- **CPU = 0.8**
- **Network = 0.6**
- **Disk = 1** ← **100% Utilization!**

Throughput:

Processes and Resources

What is the **capacity**, the maximum throughput that can be sustained, of the system?

At **MPL=3**, our system achieves the following **steady-state:**

resource\time	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74
CPU	1	2	3	4	5	6	7	8			9	10	11	12	13	14	15	16			17	18	19	20	21	22	23	24			1	2	3	4
Network	1	2					3	4	5	6	7	8					9	10	11	12	13	14					15	16	17	18	1	2		
Disk	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	1	2	3	4

**each color corresponds to a new instance of the task*

Steady-State Utilization:

- **CPU = 0.8**
- **Network = 0.6**
- **Disk = 1** ← **100% Utilization!**

Throughput:

3/30 processes per millisecond (=100 proc/sec)

Little's Law

A theorem by *John Little* which states that under a **steady state** the **average number of items** in a queuing system is **equal** to the average **arrival rate multiplied** by the **average time that an item spends** in the queuing system.

Little's Law

A theorem by *John Little* which states that under a **steady state** the **average number of items** in a queuing system is **equal** to the average **arrival rate multiplied** by the **average time that an item spends** in the queuing system.

Let:

- **L** be the average number of items within the system
- **λ** be the average arrival rate of items into the system
- **W** be the average time an item spends in the system

Thus, we get:

$$\mathbf{L = \lambda \times W}$$

Little’s Law

General application of Little’s Law. You are a restaurant owner who is planning seating to accommodate your customers. You know that on average one customer **arrives every two minutes**. Also, on average a customer **spends 40 minutes** in your establishment. How many seats should you have to prevent your customers from waiting before being seated?

Little’s Law

General application of Little’s Law. You are a restaurant owner who is planning seating to accommodate your customers. You know that on average one customer **arrives every two minutes**. Also, on average a customer **spends 40 minutes** in your establishment. How many seats should you have to prevent your customers from waiting before being seated?

λ = 1 customer / 2 minutes **= 0.5 customers/minute**

Little’s Law

General application of Little’s Law. You are a restaurant owner who is planning seating to accommodate your customers. You know that on average one customer **arrives every two minutes**. Also, on average a customer **spends 40 minutes** in your establishment. How many seats should you have to prevent your customers from waiting before being seated?

λ = 1 customer / 2 minutes **= 0.5 customers/minute**
W = 40 minutes

Little’s Law

General application of Little’s Law. You are a restaurant owner who is planning seating to accommodate your customers. You know that on average one customer **arrives every two minutes**. Also, on average a customer **spends 40 minutes** in your establishment. How many seats should you have to prevent your customers from waiting before being seated?

λ = 1 customer / 2 minutes **= 0.5 customers/minute**
W = 40 minutes

$L = \lambda \times W$

Little's Law

General application of Little's Law. You are a restaurant owner who is planning seating to accommodate your customers. You know that on average one customer **arrives every two minutes**. Also, on average a customer **spends 40 minutes** in your establishment. How many seats should you have to prevent your customers from waiting before being seated?

λ = 1 customer(seats) / 2 minutes **= 0.5 customers(seats)/minute**
W = 40 minutes

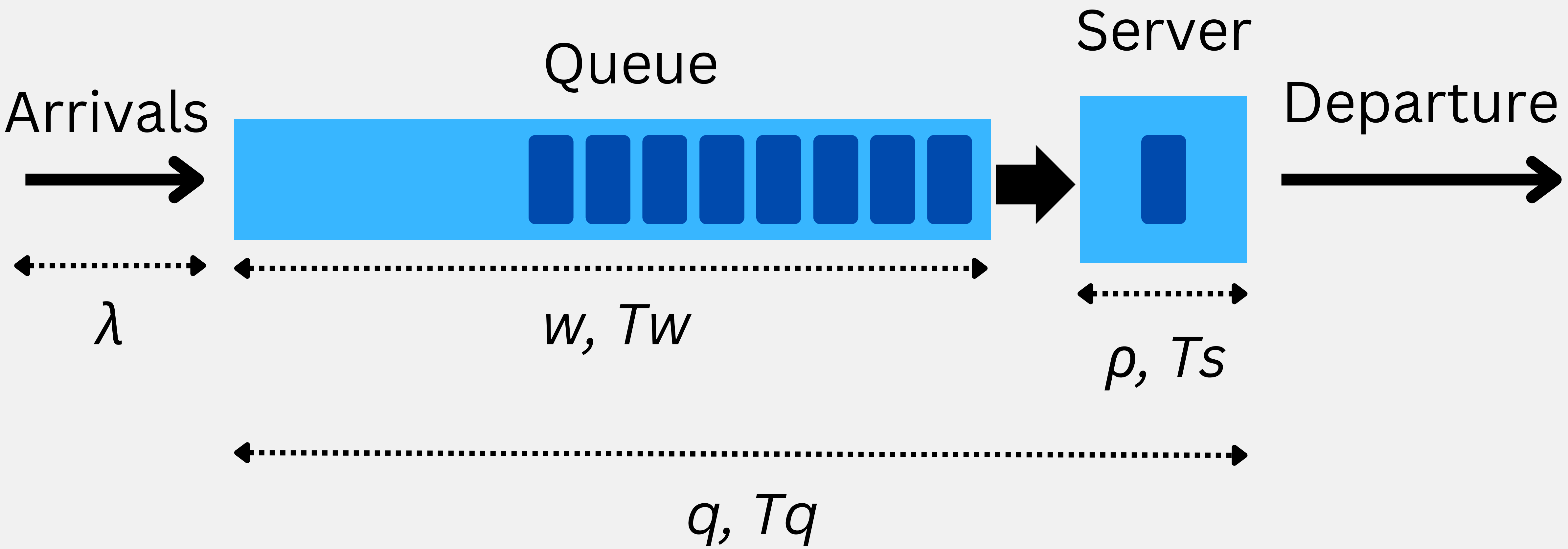
L = λ × W = 0.5 seats/minute × 40 minutes **= <u>20 seats</u>**

Little’s Law

CS350 Application, system with a server and a queue

Little's Law

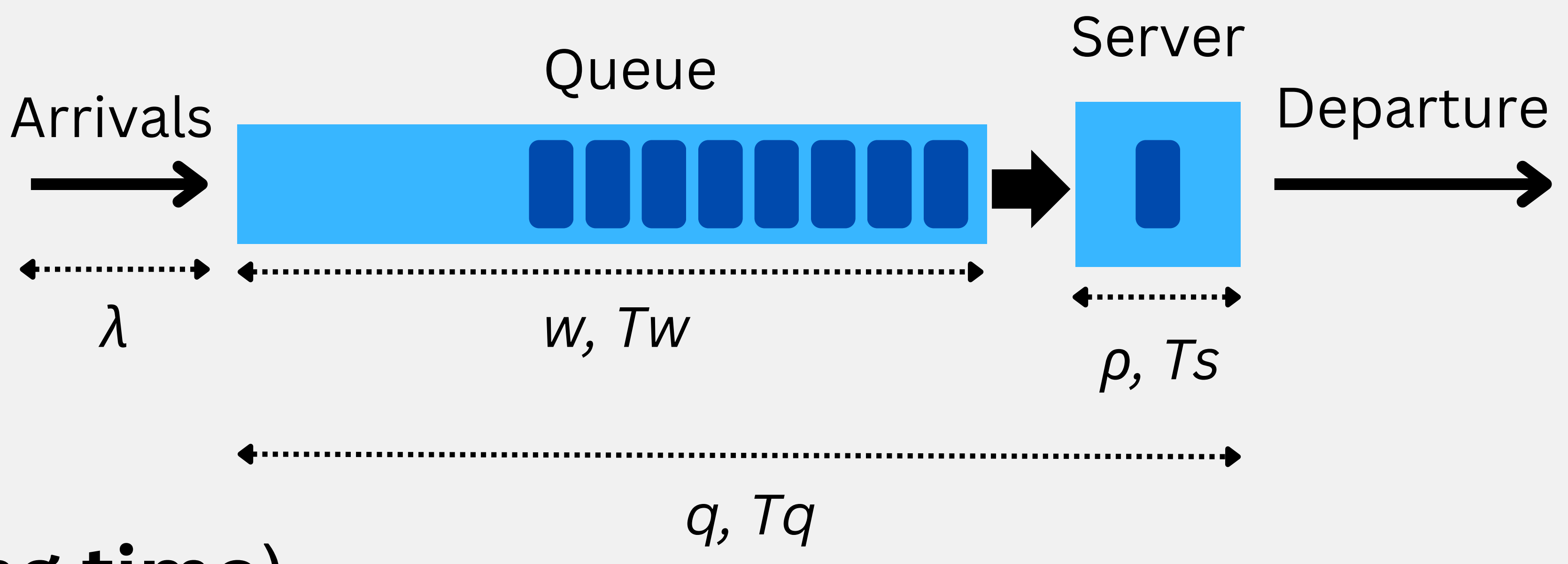
CS350 Application, system with a server and a queue



Notation:

- **λ** — the **arrival rate**
- ***w*** — number of **requests in the queue**
- ***Tw*** — time a request spends in the queue (**waiting time**)
- ***ρ*** — *server **utilization***
- ***Ts*** — *time a server spends processing a request (**service time**)*
- ***q*** — *number of **requests in the whole system***
- ***Tq*** — *time a request spends in the whole system before departure (**response time**)*

Little's Law



Notation:

- **λ** — the **arrival rate**
- ***w*** — number of **requests in the queue**
- ***Tw*** — time a request spends in the queue (**waiting time**)
- ***ρ*** — *server **utilization***
- ***Ts*** — *time a server spends processing a request (**service time**)*
- ***q*** — *number of **requests in the whole system***
- ***Tq*** — *time a request spends in the whole system before departure (**response time**)*

Universal Relationships:

- $\rho = \lambda \times Ts$ valid if $\lambda < 1 / (Ts)$
- $Tq = Tw + Ts$

Under the assumption that the system is at **steady-state** with **no unaccounted deaths**:

- $q = \lambda \times Tq$
- $w = \lambda \times Tw$
- $q = w + \rho$ ← demonstrated in lecture

Let's Practice

Exercise 1

A cloud gaming server allows players to offload the computation necessary to play a video-game. A single interaction between a user and the system consists of the following steps:

1. User submits control inputs over the **network**, it takes **3ms**
2. **CPU** interprets the inputs, it takes **5ms**
3. **GPU** renders the next frame, it takes **7ms**
4. The frame is sent over the **network**, it takes **6ms**

For **MPL=1** answer the following questions:

a) What are the steady-state **utilizations** of each resource?
b) What is the **throughput** of the system?
c) What is the **bottleneck** of the system?

For **MPL=2** answer the following questions:

d) What are the new steady-state **utilizations** of each resource?
e) What is the new **throughput** of the system?

Also, answer the following generalized questions:

f) What is the **maximum MPL** beyond which no further improvements are to be expected?
g) What is the **capacity** of the system?

Problem 3.1 from the book

Let’s Practice

Exercise 2

Take a look at how CodeBuddy works. New submissions are queued up in their order of arrival and then **processed one at a time**. Fast-forward to a future when Renato has implemented a feature that displays the average **inter-arrival time** between student submissions and that such value is **45 seconds**.

Figure out the following about the performance of CodeBuddy:

a) What is the **throughput** of the CodeBuddy system?

Also, you notice that CodeBuddy reports (1) the average **response time** over all the completed submissions which is **30 seconds** and (2) the average **service time** for each request which turns out to be **20 seconds**.
Answer the following:

g) What is the **utilization** of CodeBuddy?

h) If you were to take a timed average of the number of submission queued and waiting to be run (i.e. excluding the one, if any, currently running), what would this number be?

Problem 5.2 from the book (subquestions (b)-(f) were omitted)