# Protego: Overload Control for Applications with Unpredictable Lock Contention

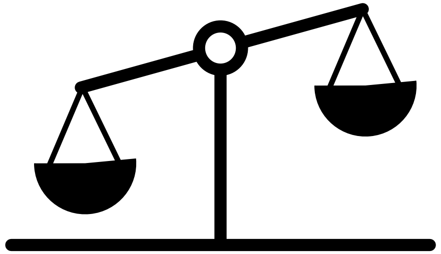
**Inho Cho**[*] Ahmed Saeed[†] Seo Jin Park[*]
Mohammad Alizadeh[*] Adam Belay[*]

[*] MIT CSAIL

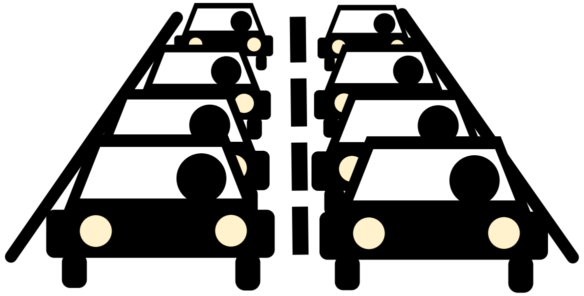
[†] Georgia Tech

# Server Overload

Load Imbalance

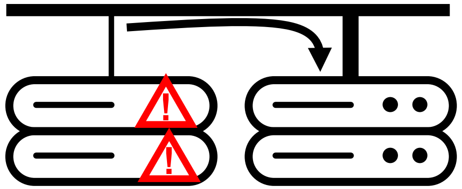


Unexpected user traffic

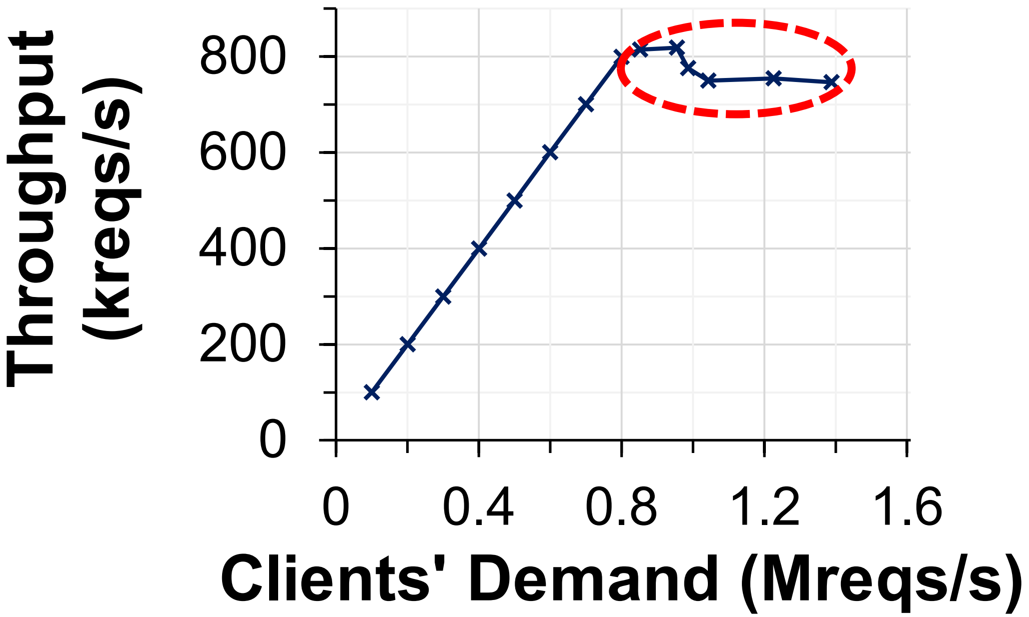


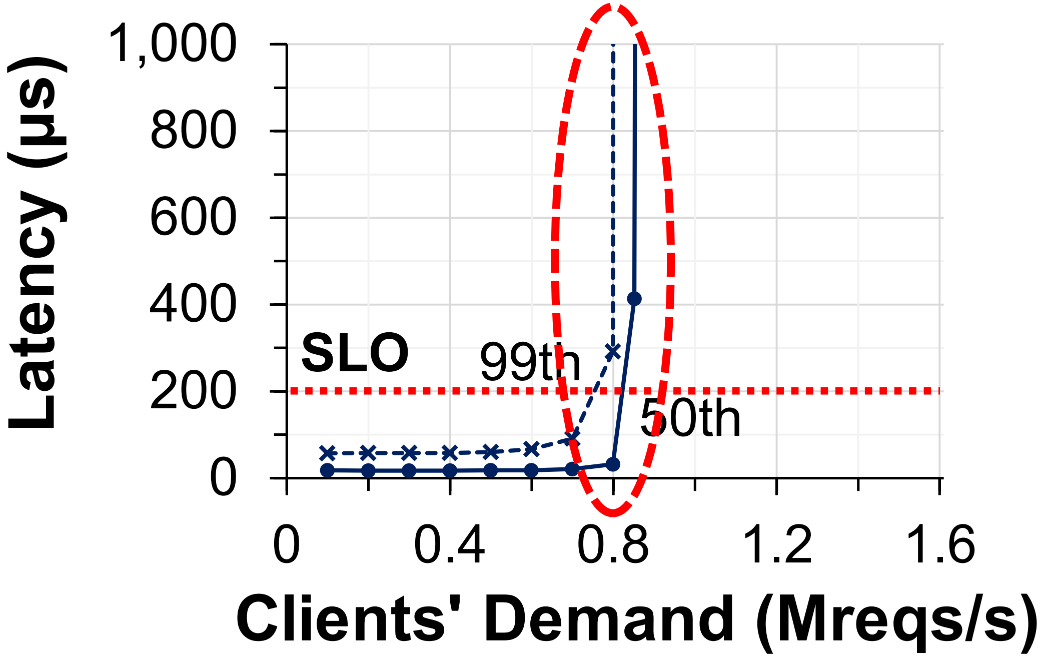
Packet bursts

Redirected traffic due to failure

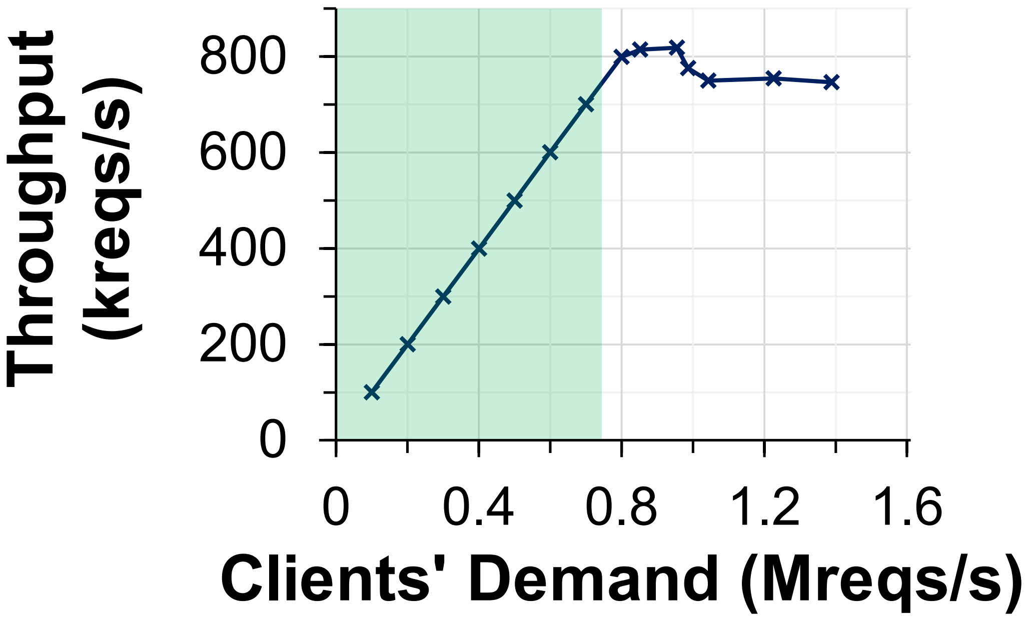


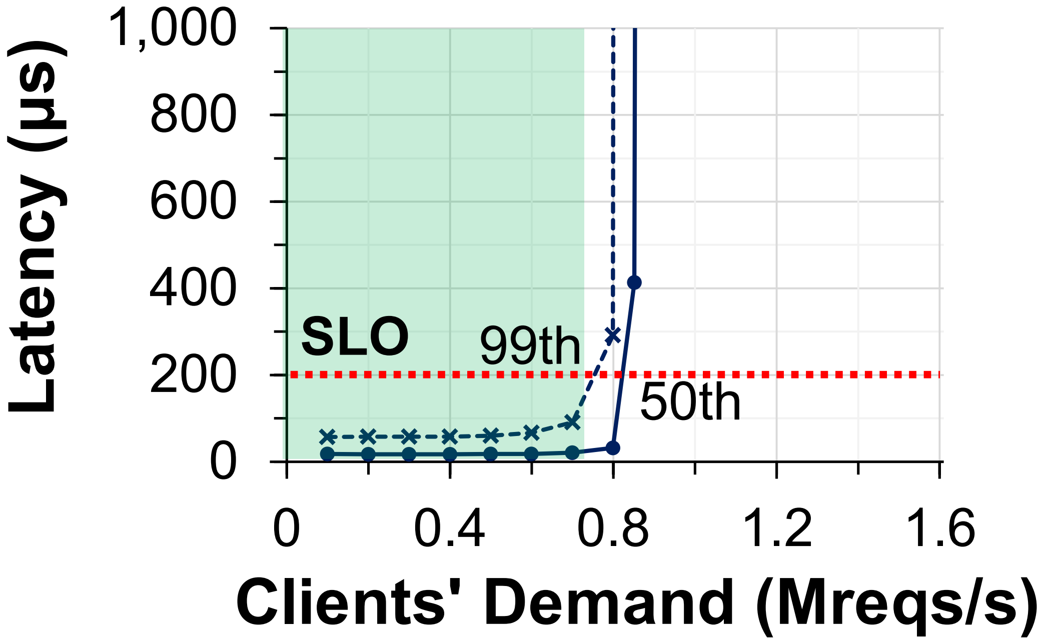
# Congestion Collapse



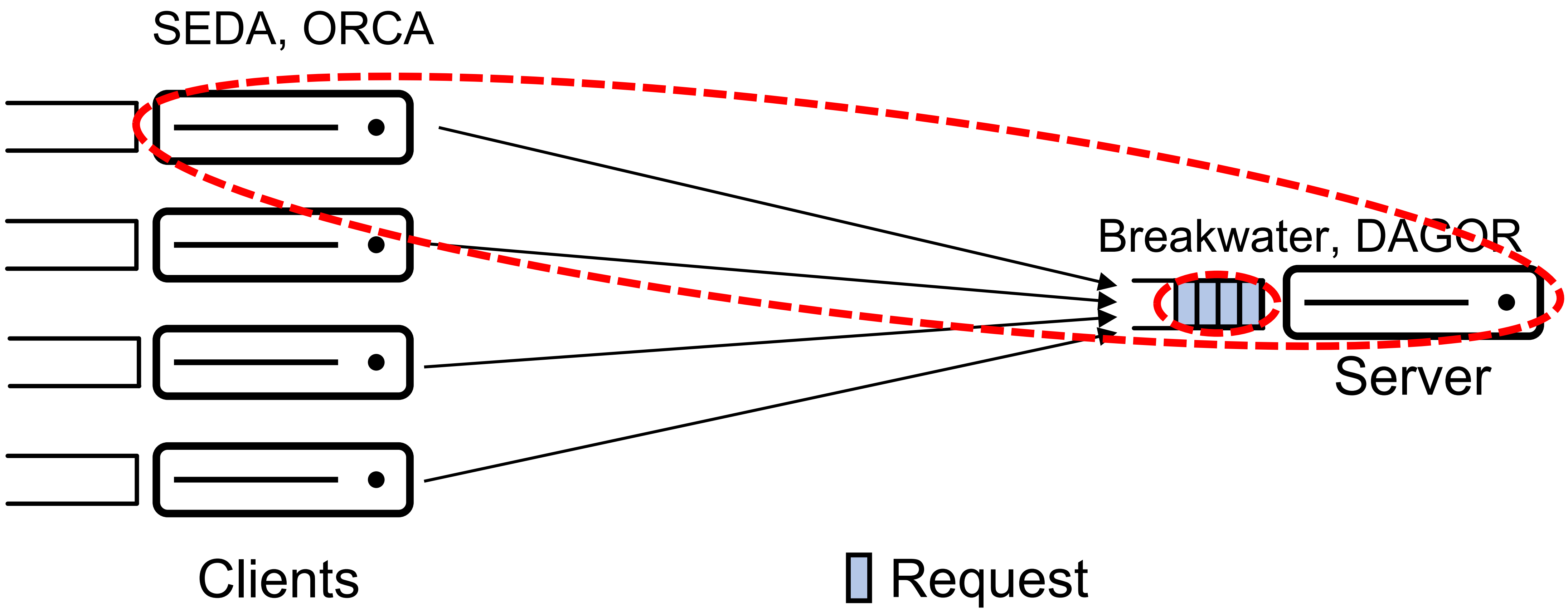


# Overload Control





# Delay as Congestion Control Signals



SEDA, ORCA

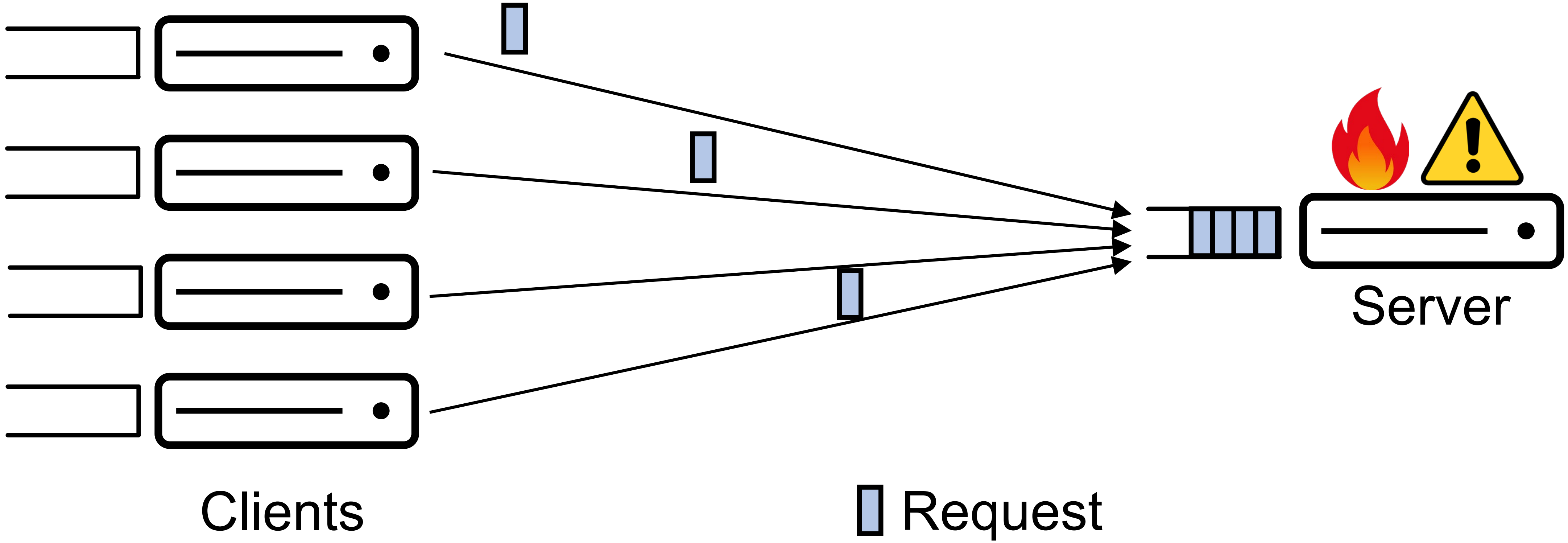
Breakwater, DAGOR

Server

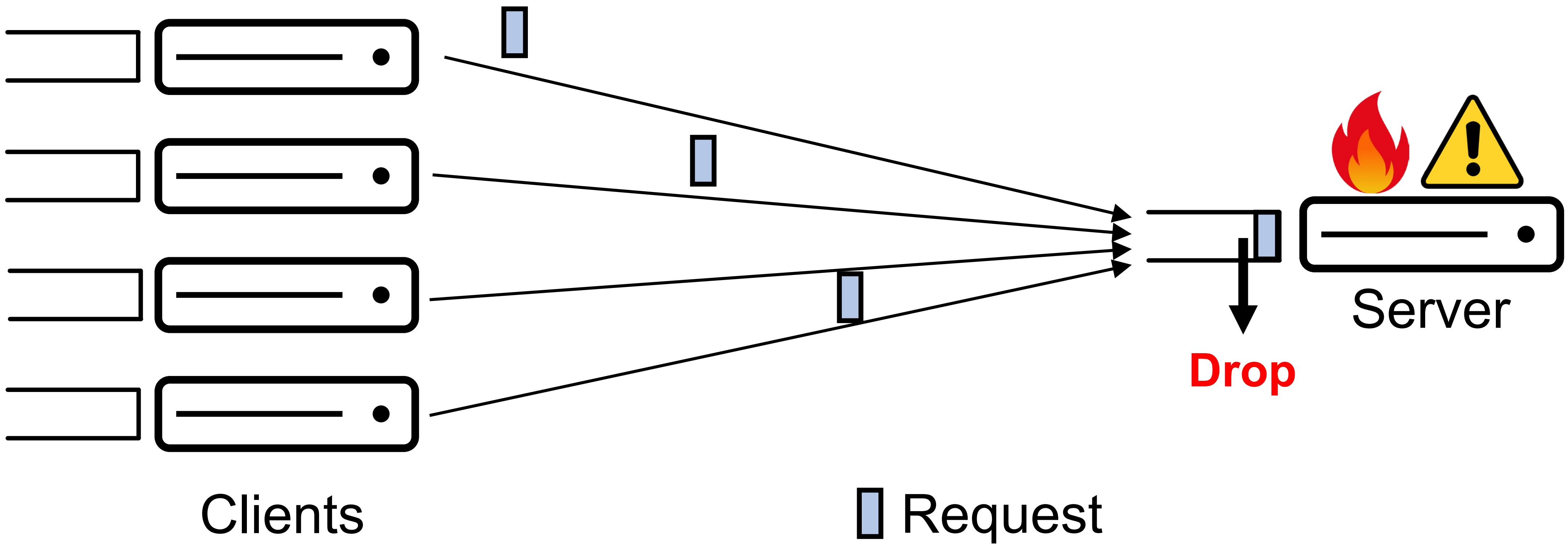
Clients

▯ Request

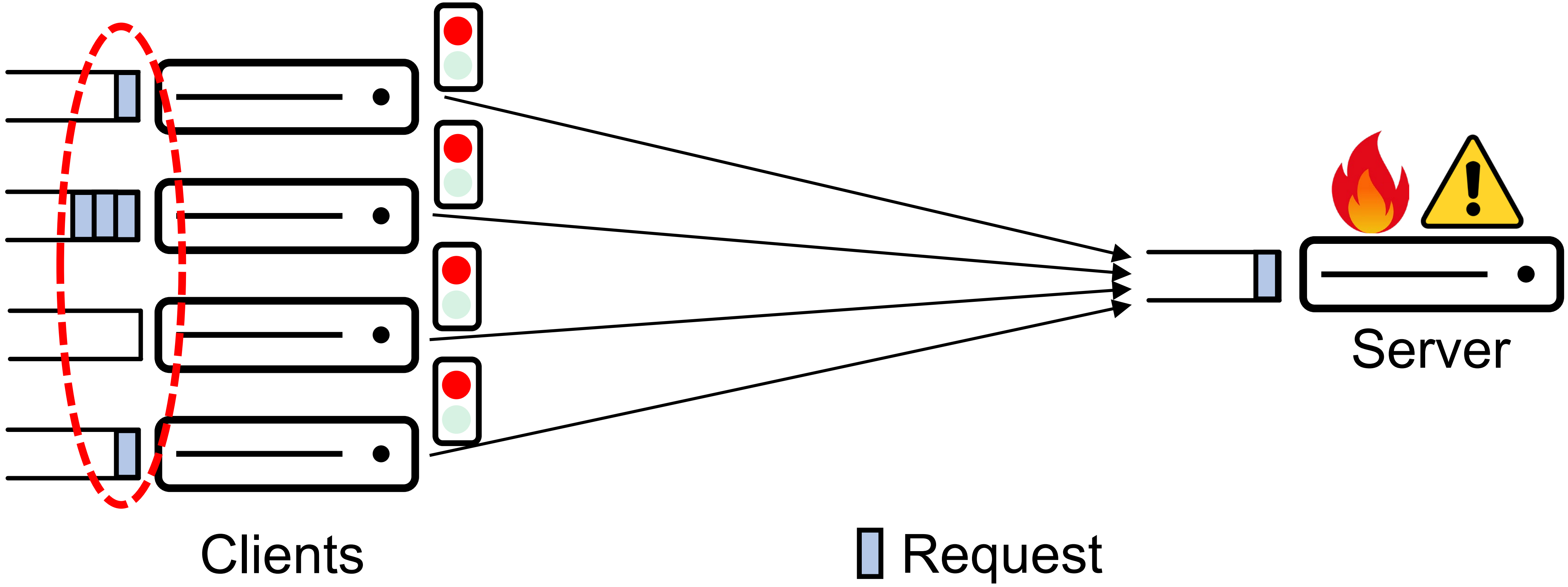
# Overload Control: AQM



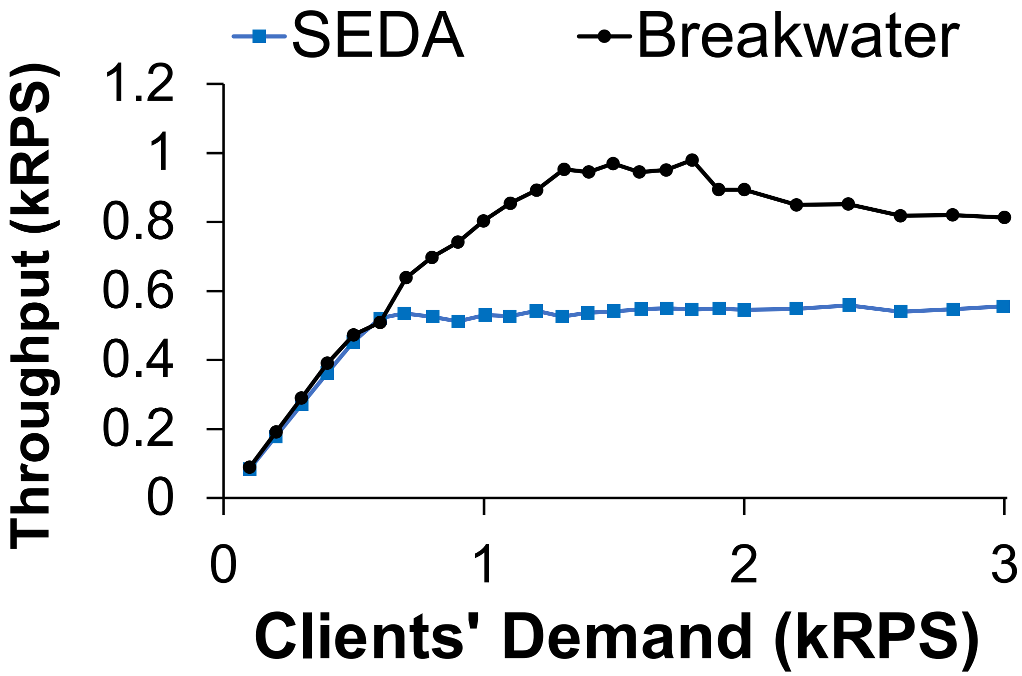
# Overload Control: AQM

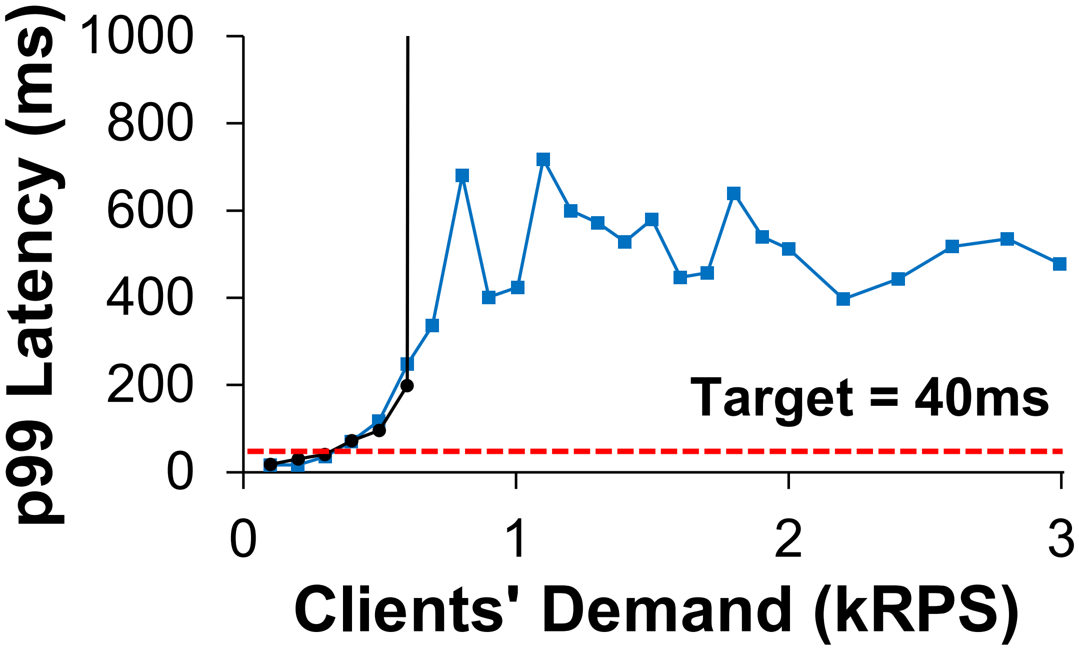


# Overload Control: Admission Control

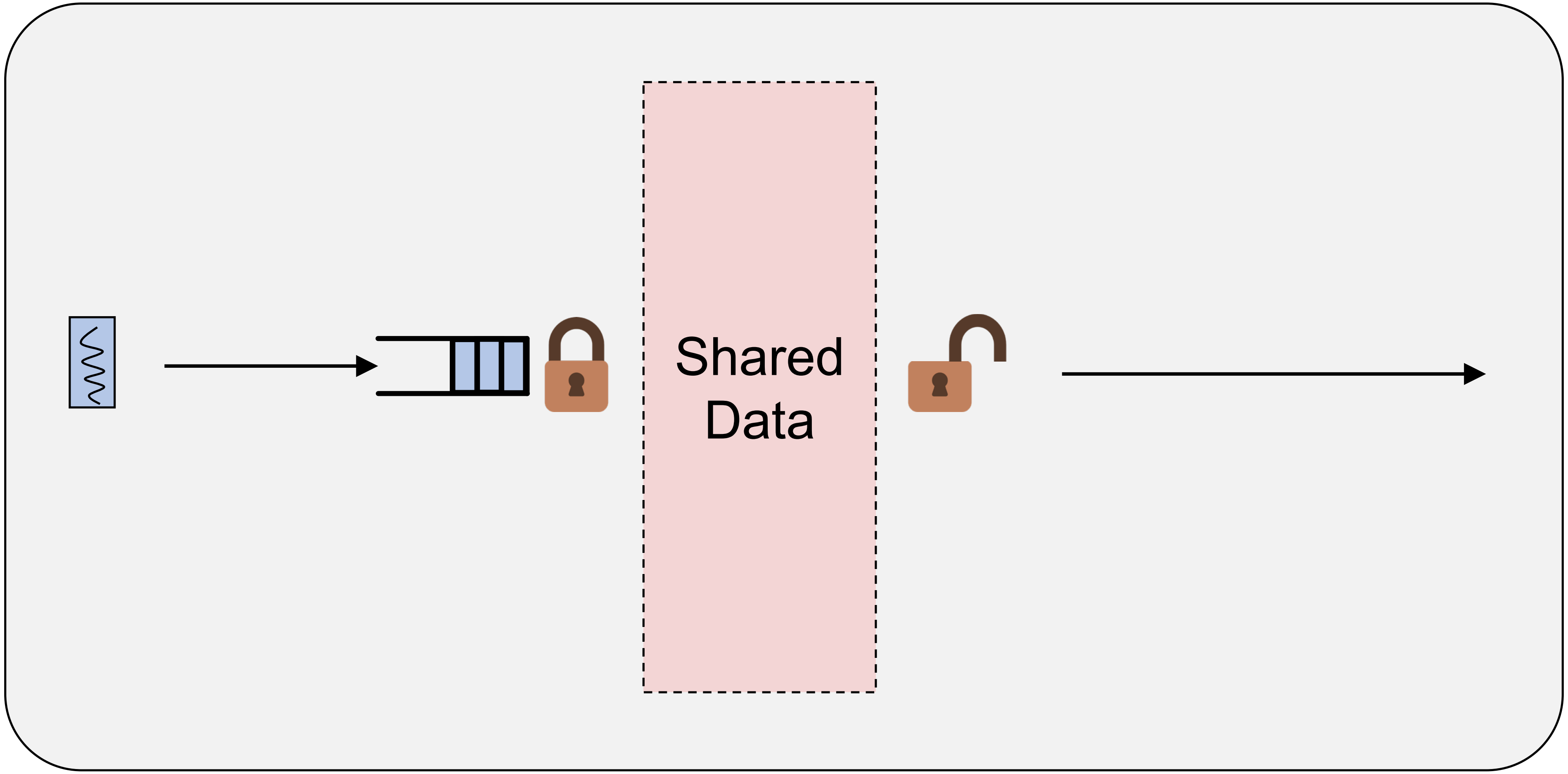


# Overload Control Mystery

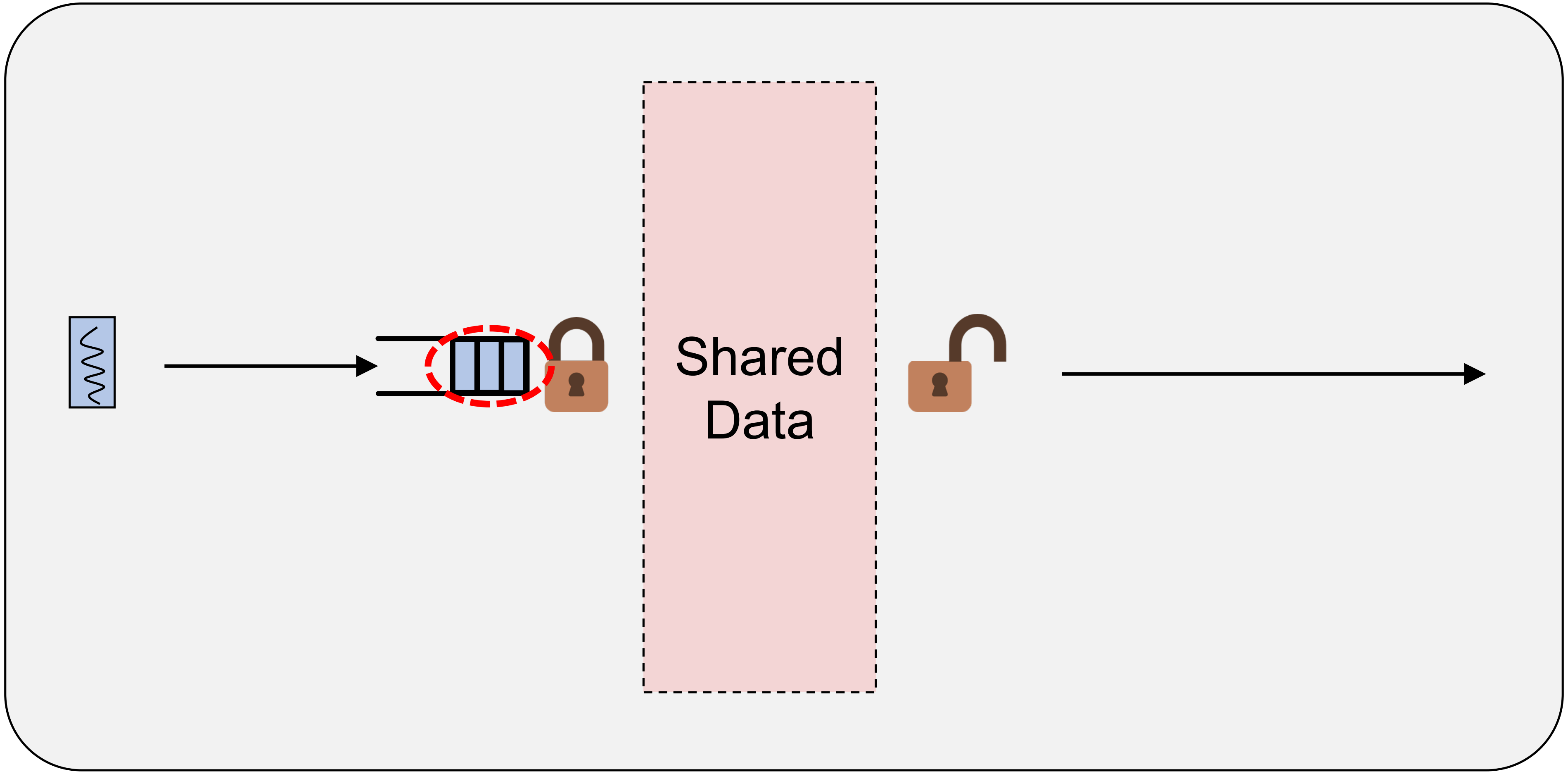




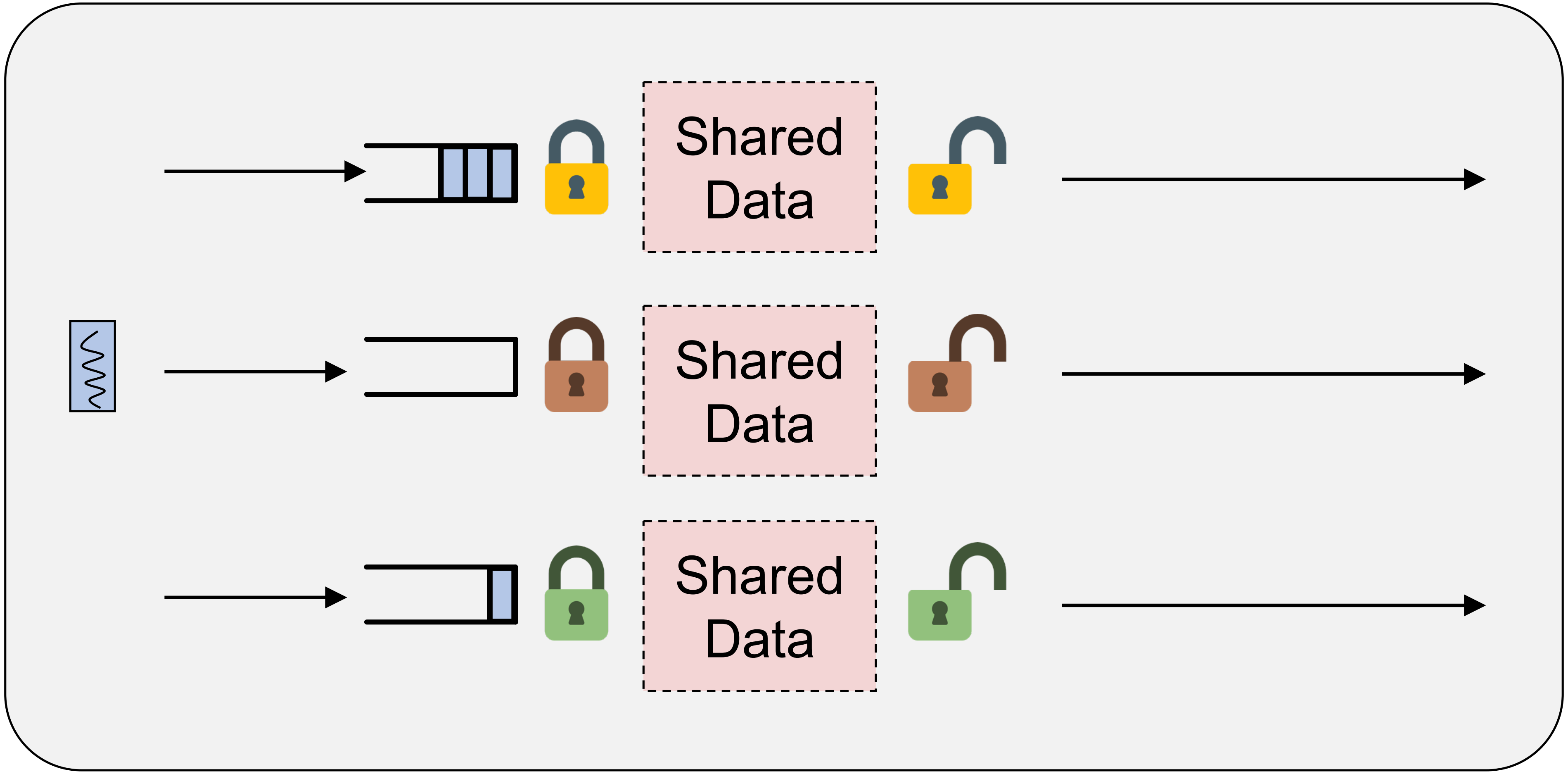
# Synchronization for Shared Data



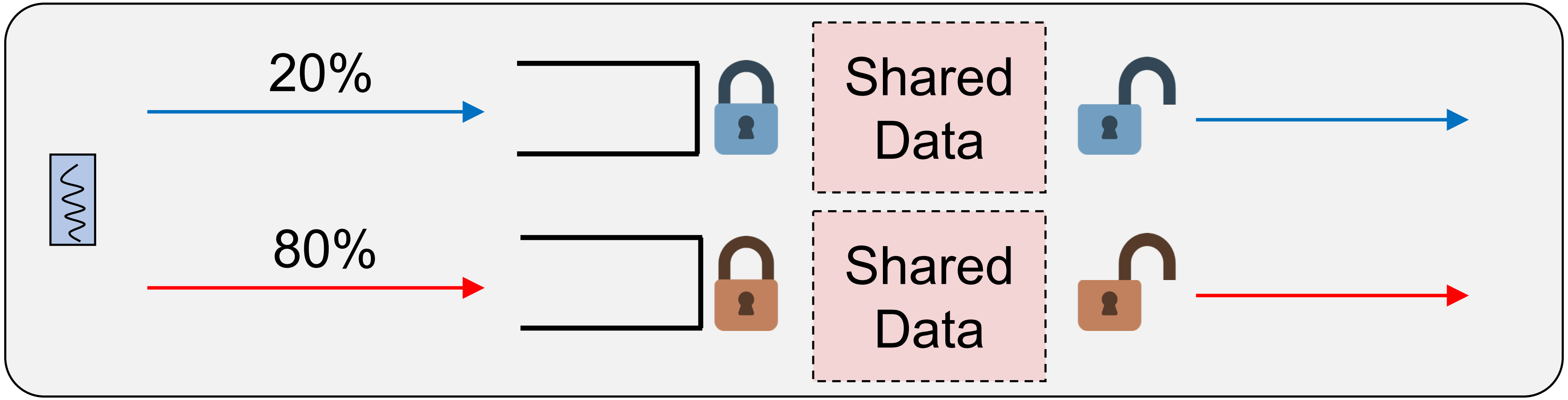
# Synchronization for Shared Data

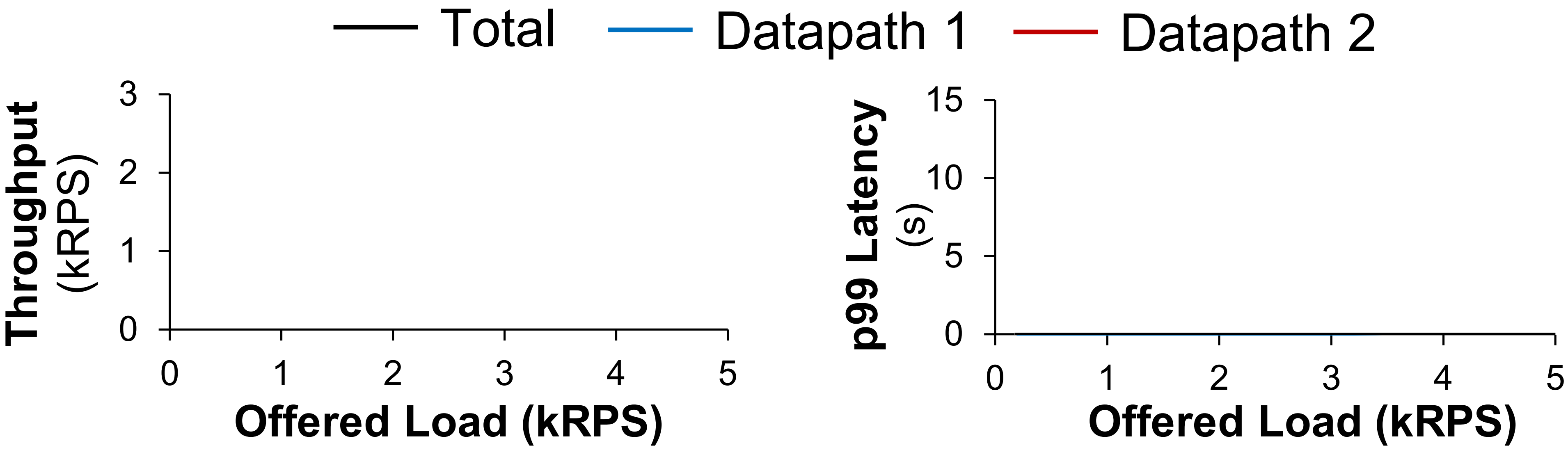


# Unpredictable Lock Contention

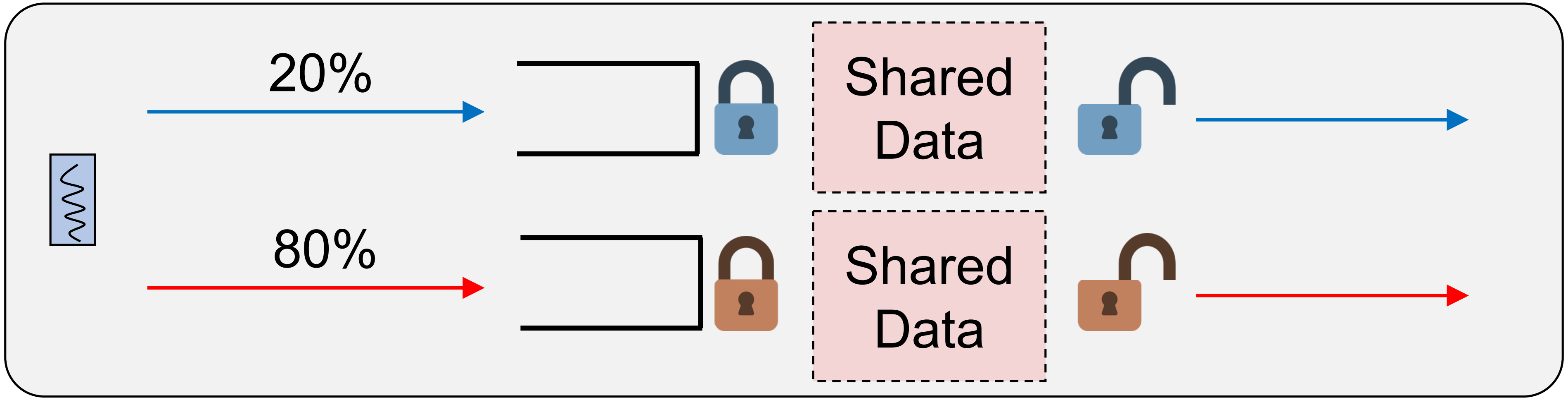


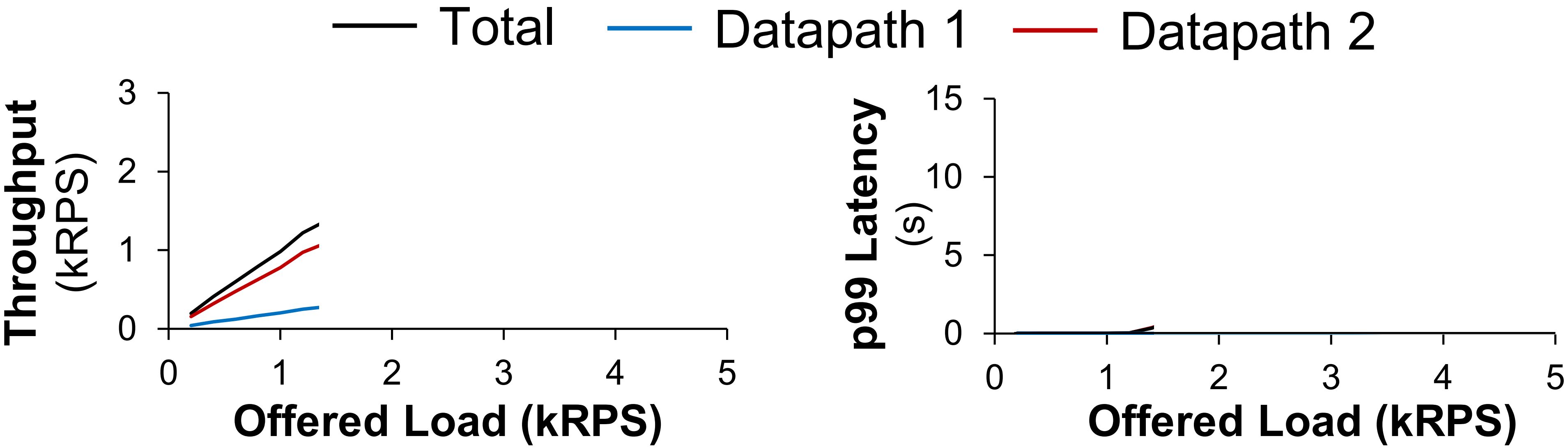
# Performance with Lock Contention



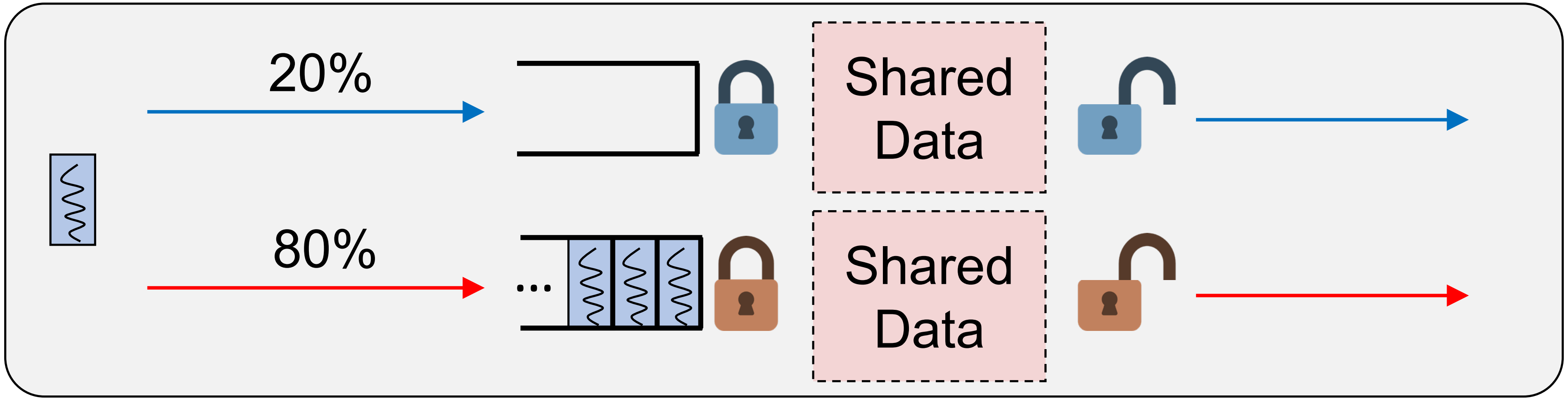


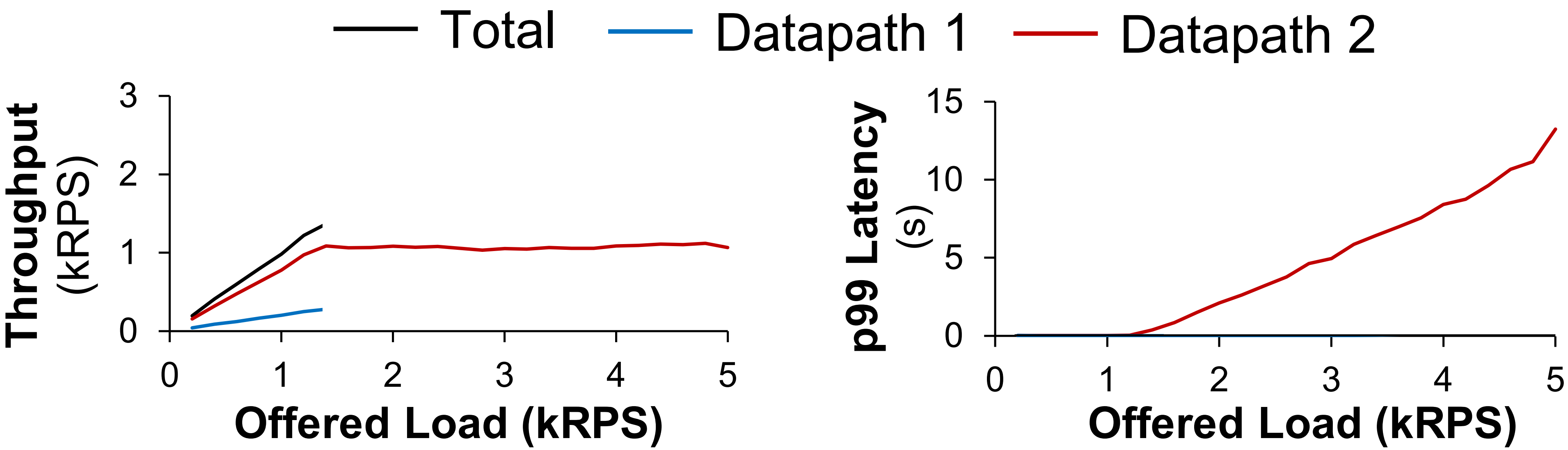
# Performance with Lock Contention



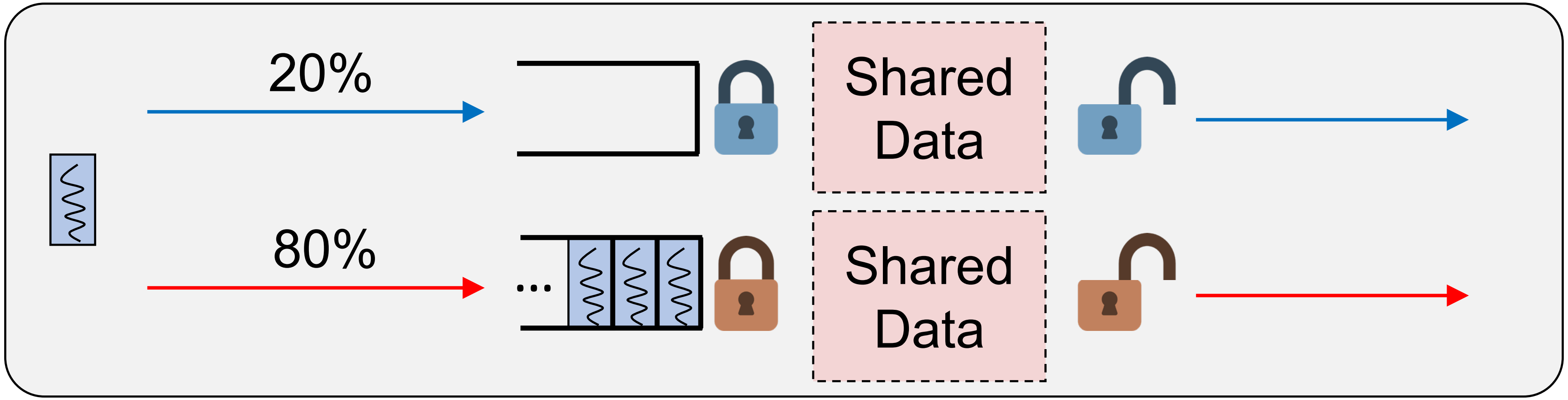


# Performance with Lock Contention





# Performance with Lock Contention



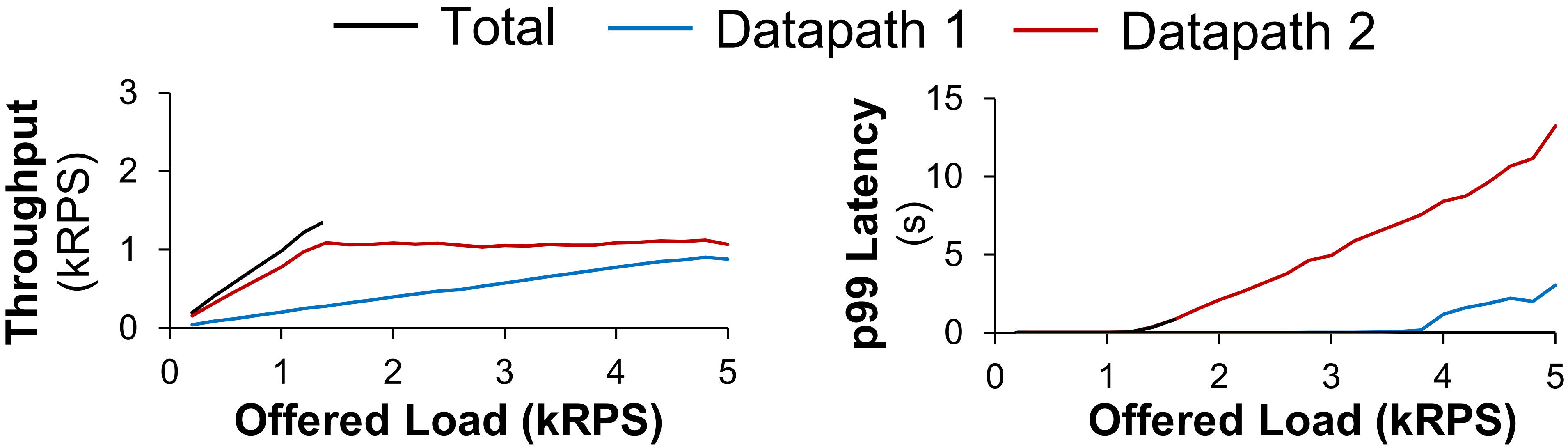
20%

80%

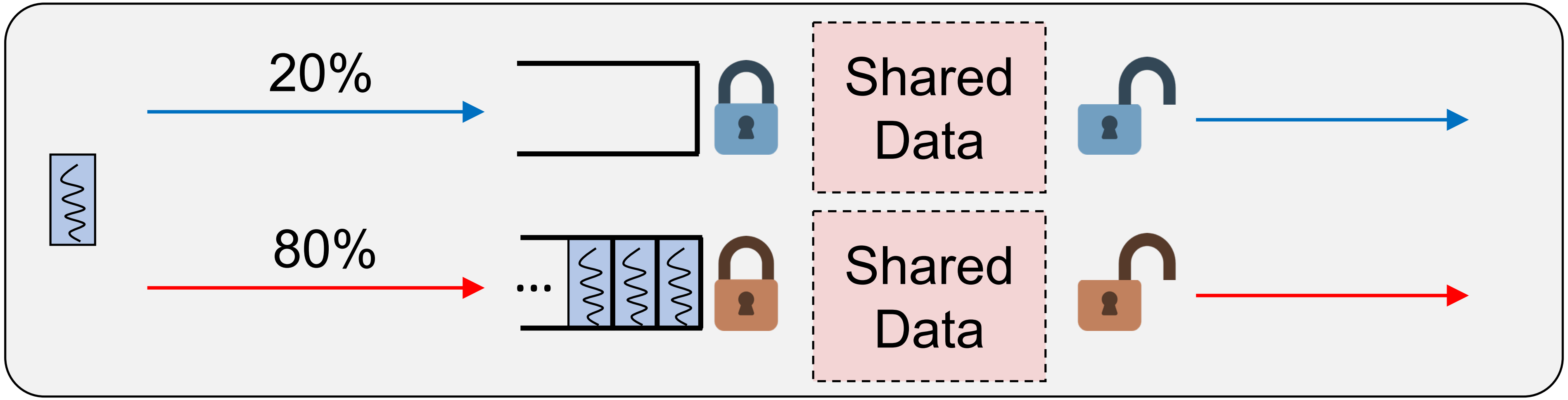
…

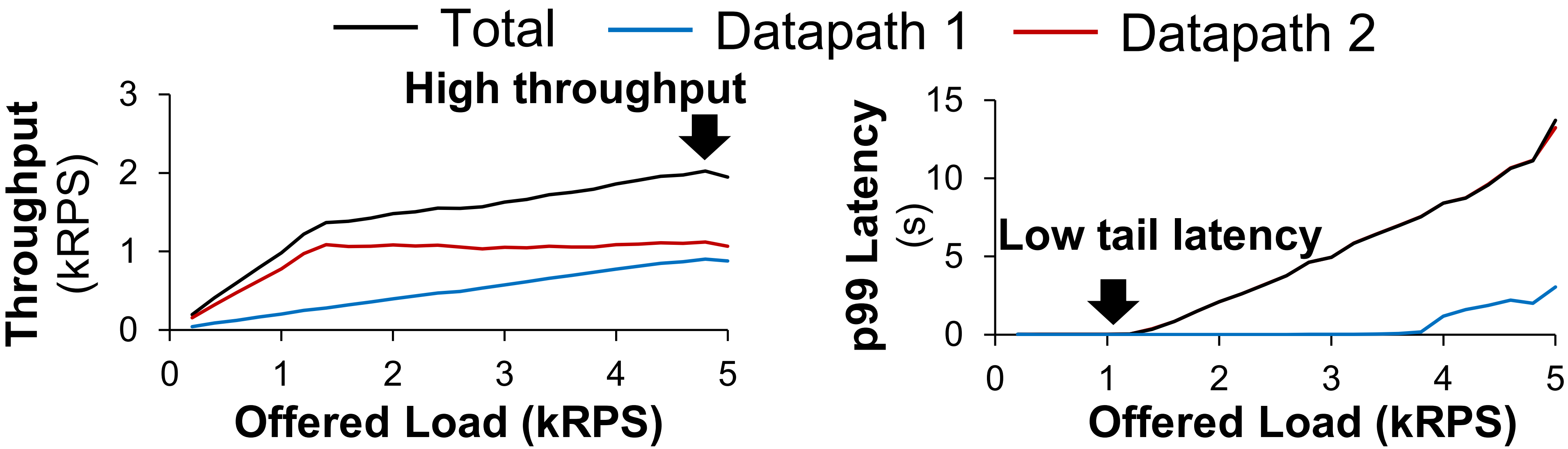
Shared Data

Shared Data



# Performance with Lock Contention

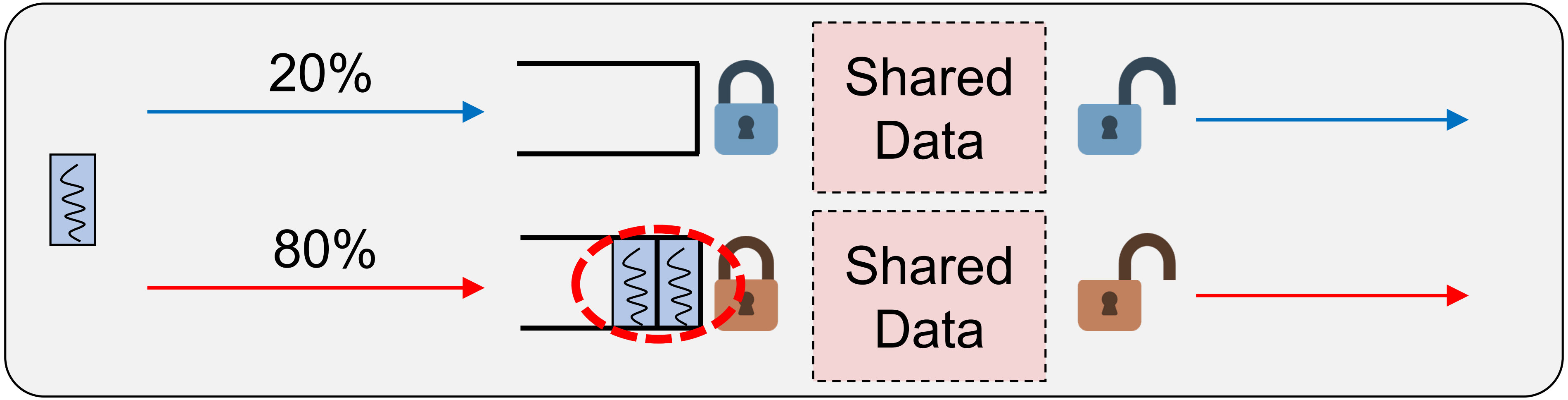


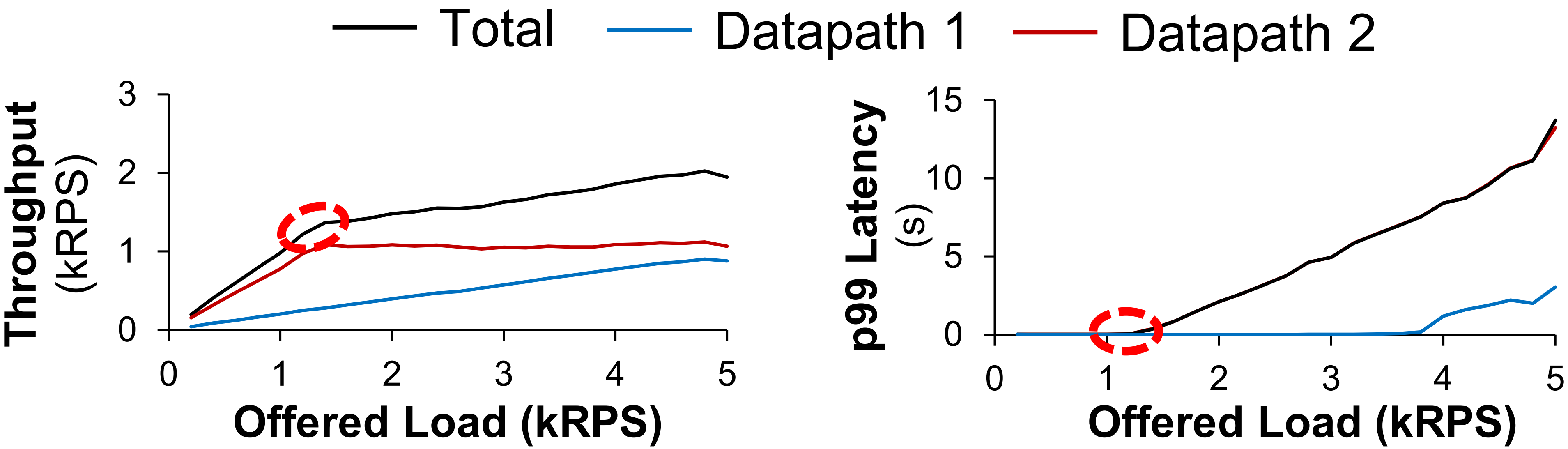


High throughput

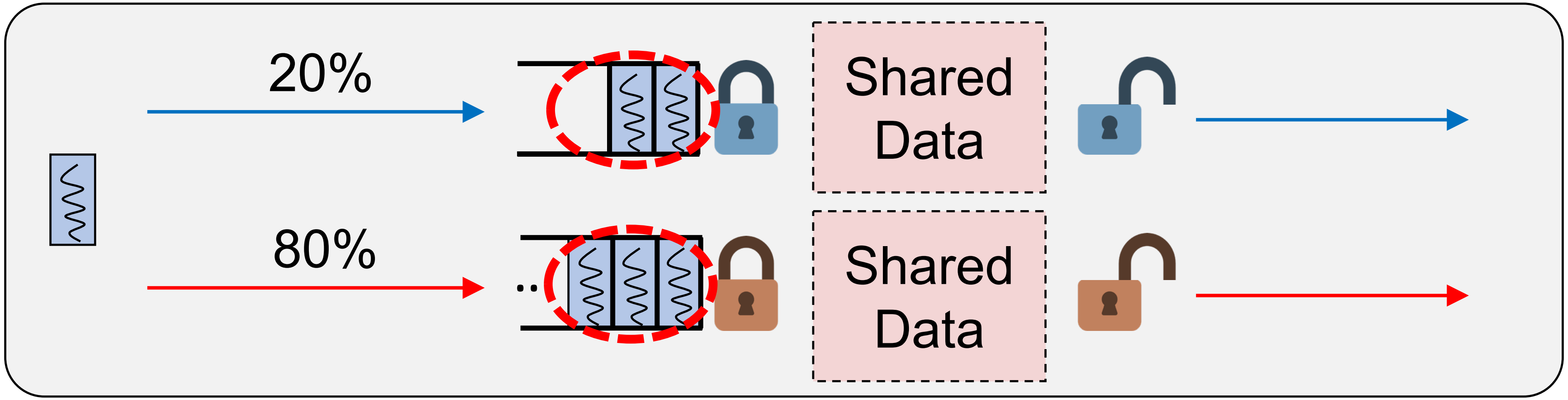
Low tail latency

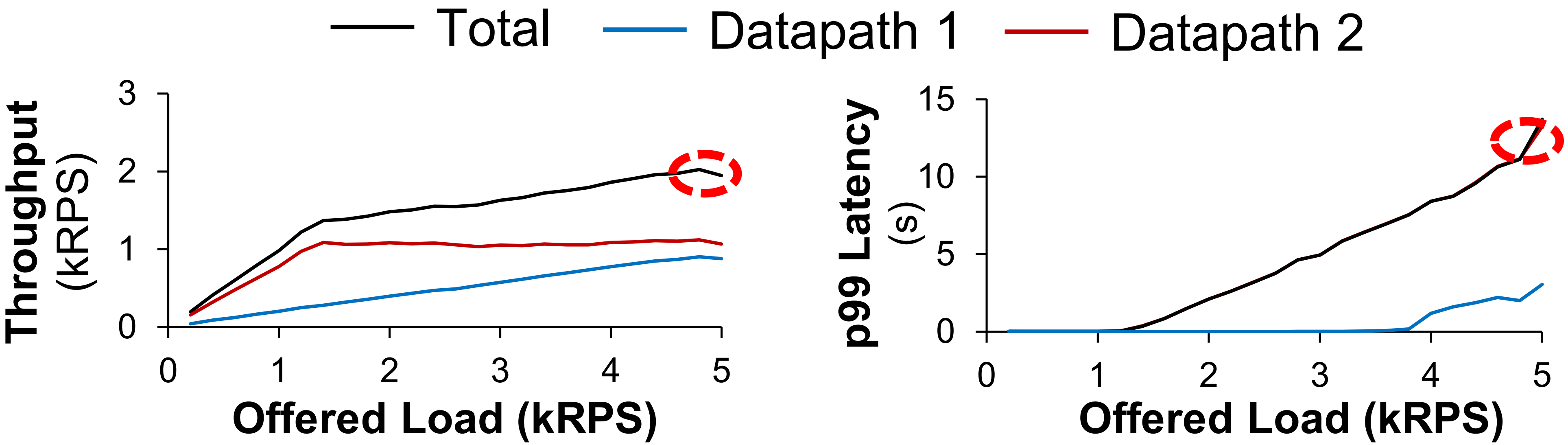
# Performance with Lock Contention



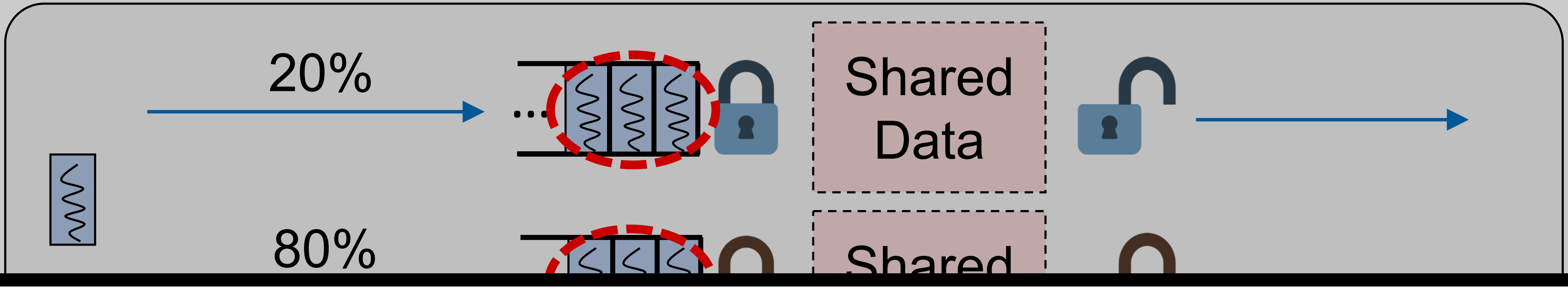


# Performance with Lock Contention

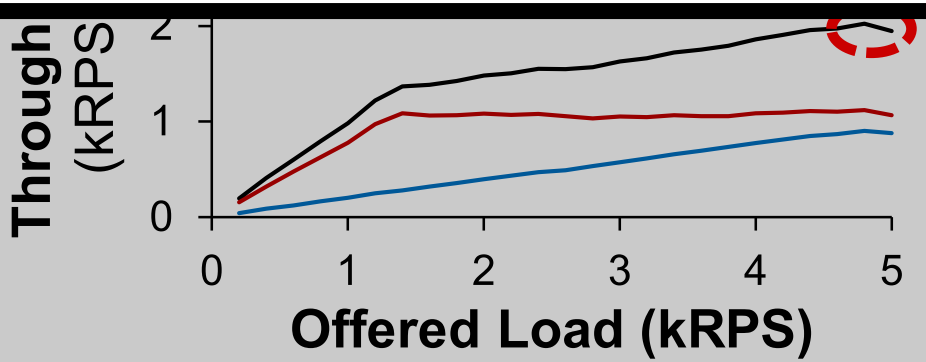


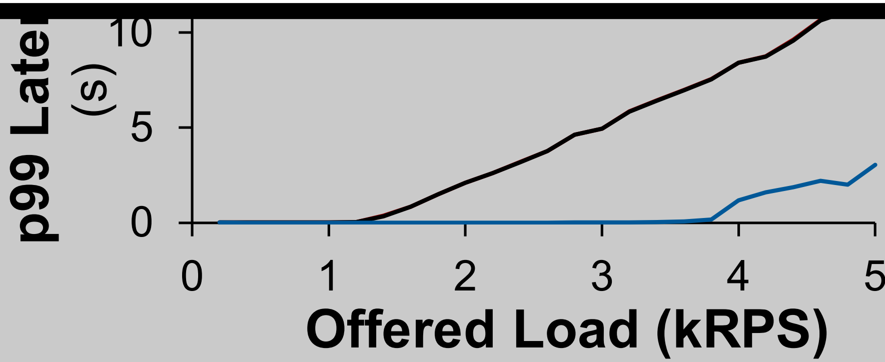


# Performance with Lock Contention



**How can we achieve both high throughput and low tail latency?**



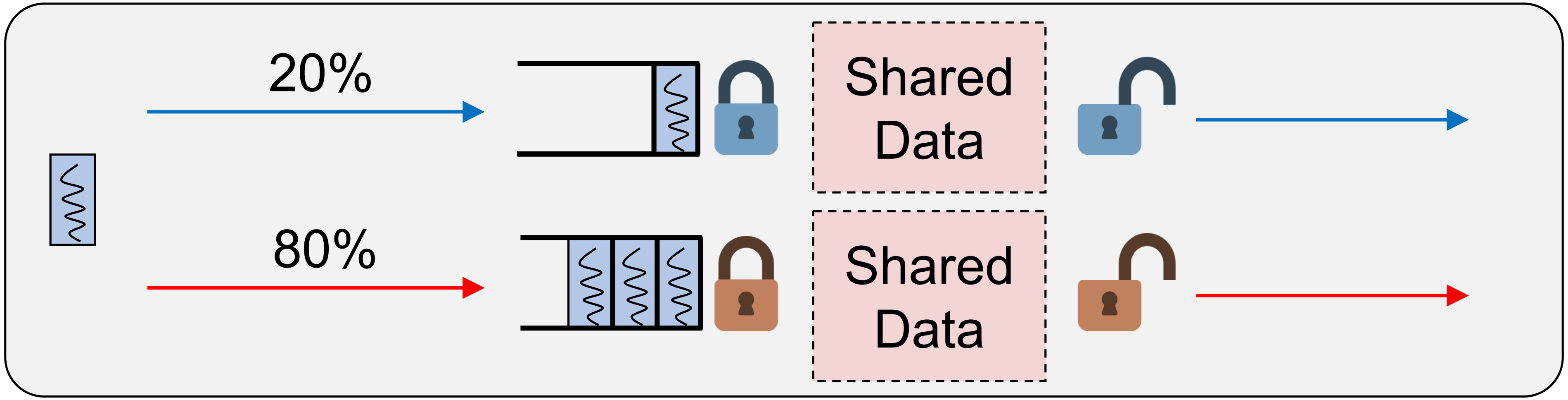


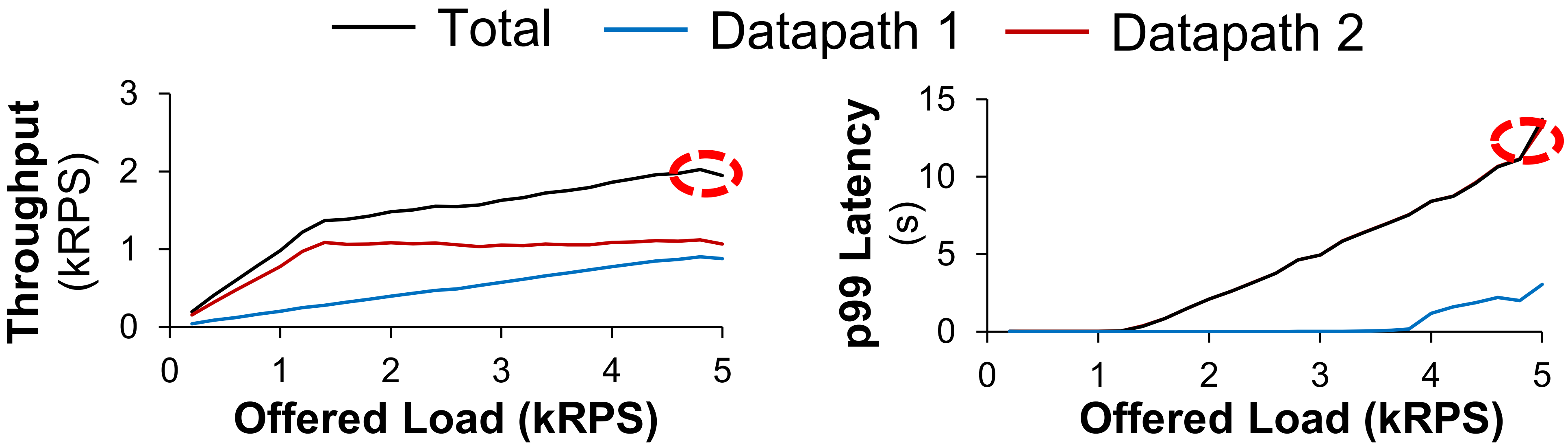
# Protego

Overload control for applications with unpredictable lock contention

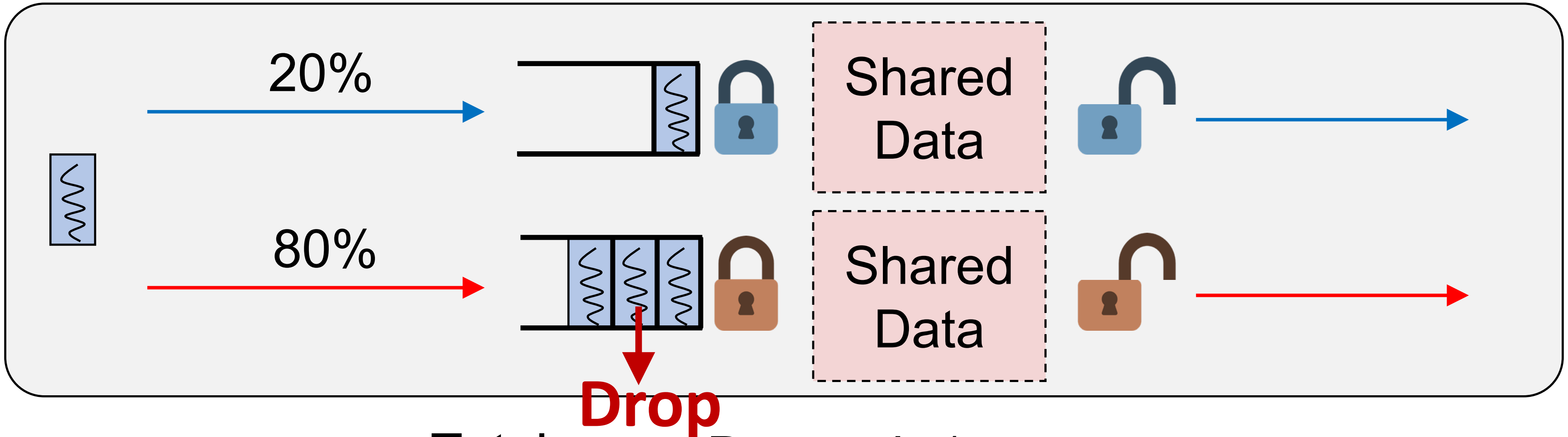
| Component | Role |
|---|---|
| Active Synchronization Queue Management (ASQM) | Ensure low latency for all the datapath |
| Performance-driven Admission Control | Achieve high throughput with efficient resource usage |

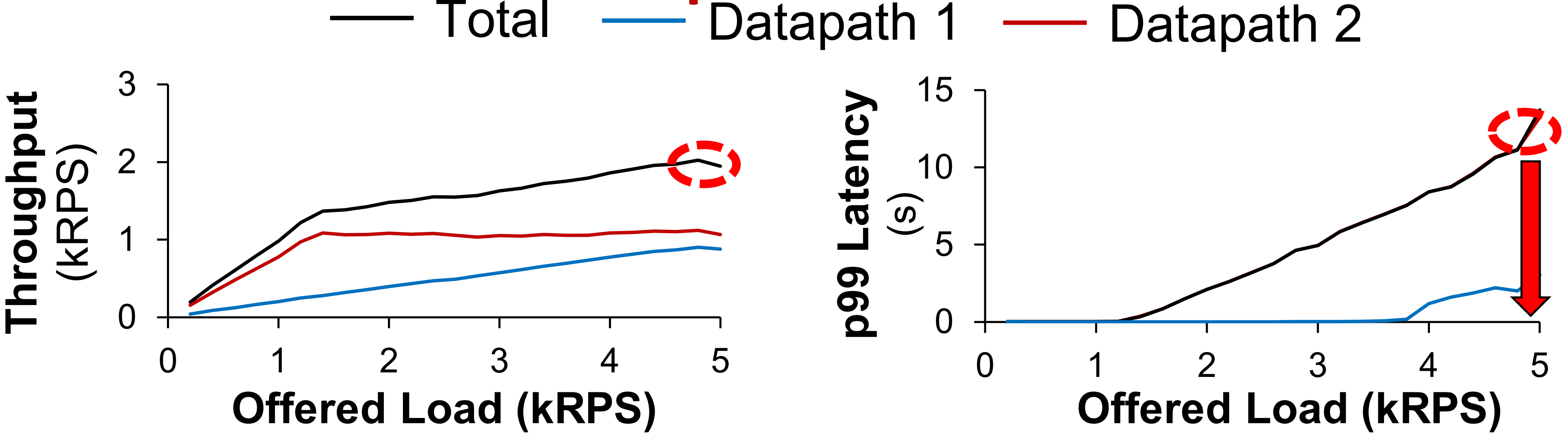
# Request Drop is Inevitable





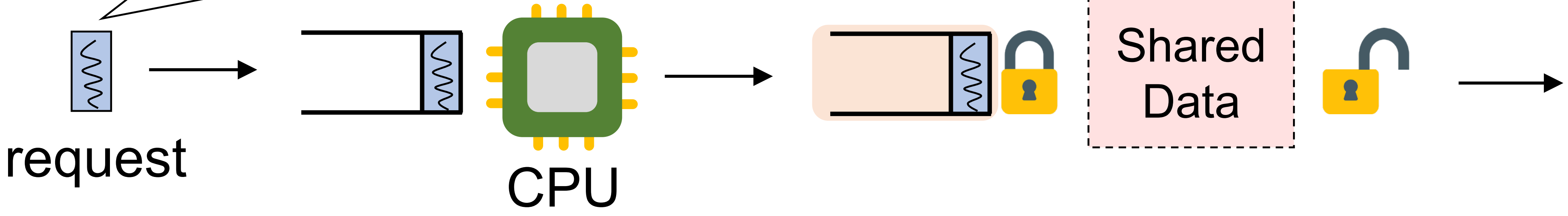
# Request Drop is Inevitable





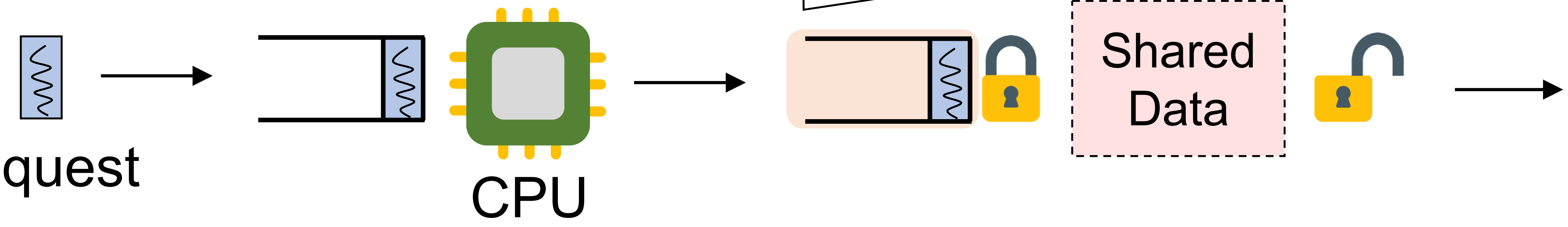
# Active Synchronization Queue Management (ASQM)

```
At request arrival:
budget := target - p99_network - p99_service
```



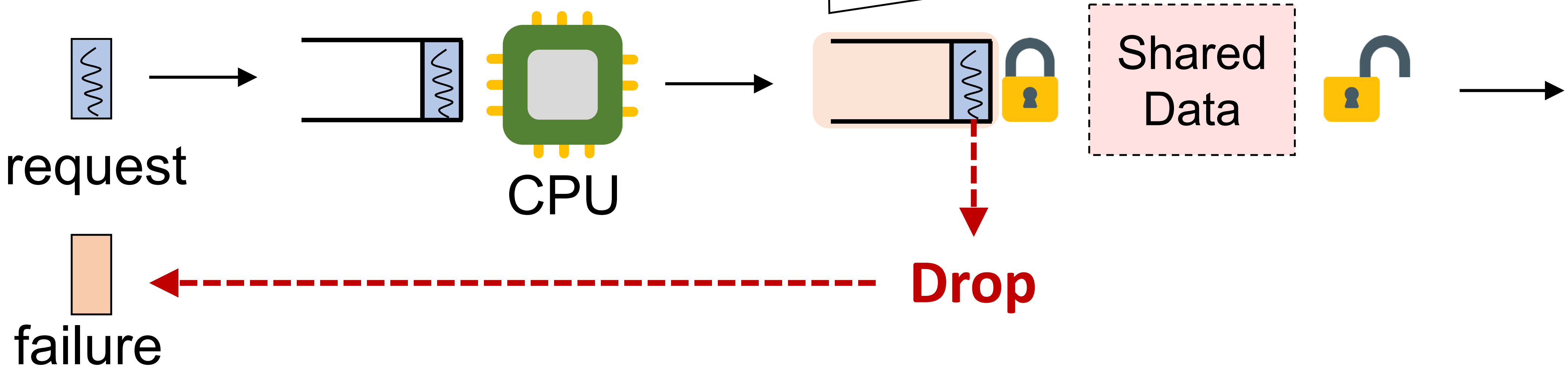
# Active Synchronization Queue Management (ASQM)

```
At enqueue:
qdelay = now - oldest.enque_tsc
if budget >= qdelay:
   enqueue
```



# Active Synchronization Queue Management (ASQM)

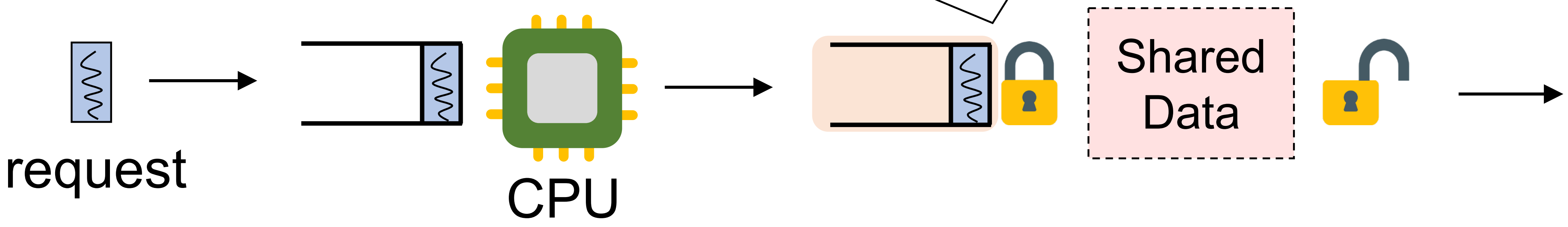
```
At enqueue:
qdelay = now - oldest.enque_tsc
if budget >= qdelay:
   enqueue
else:
   drop the request
```



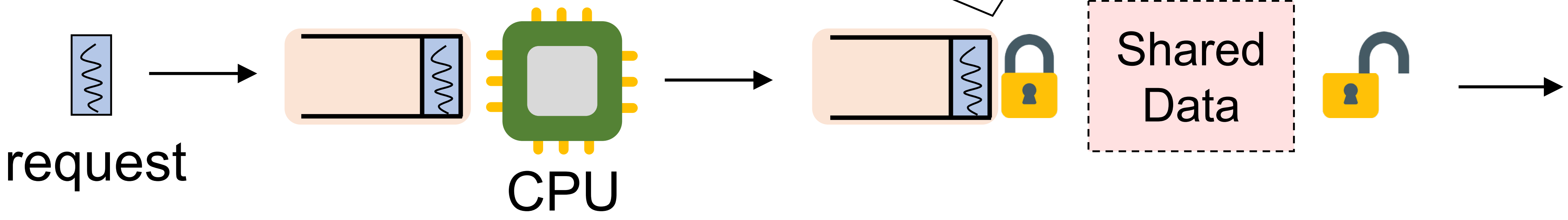
# Active Synchronization Queue Management (ASQM)

**At dequeue:**

```
budget -= queueing delay
```

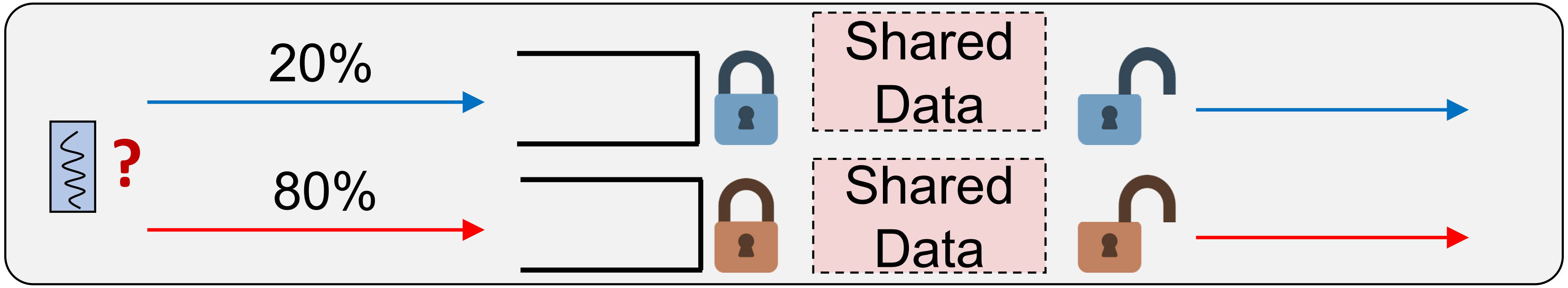


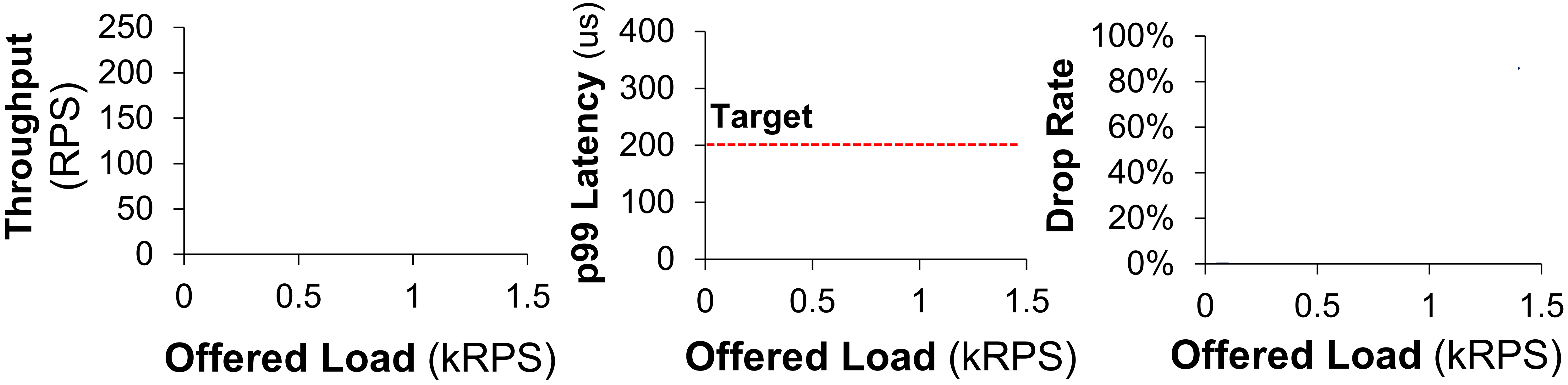
# Active Synchronization Queue Management (ASQM)



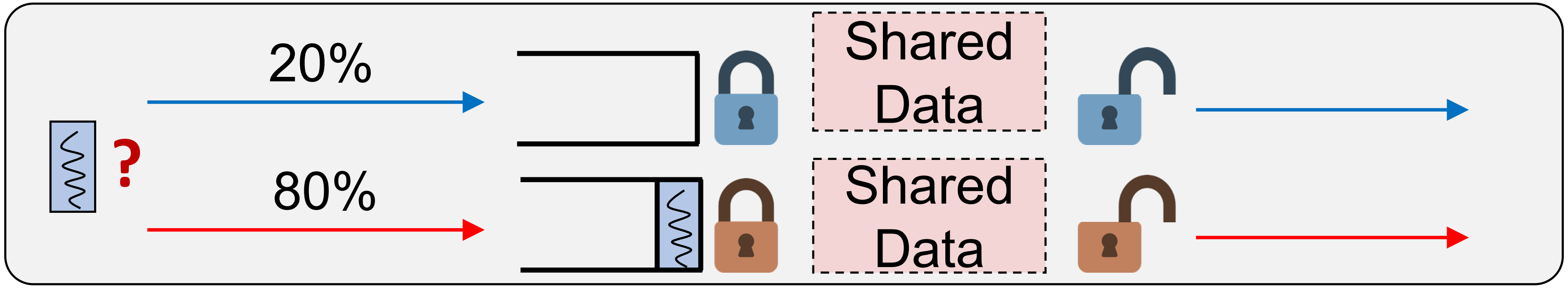
**At dequeue:**
```
budget -= queueing delay
```

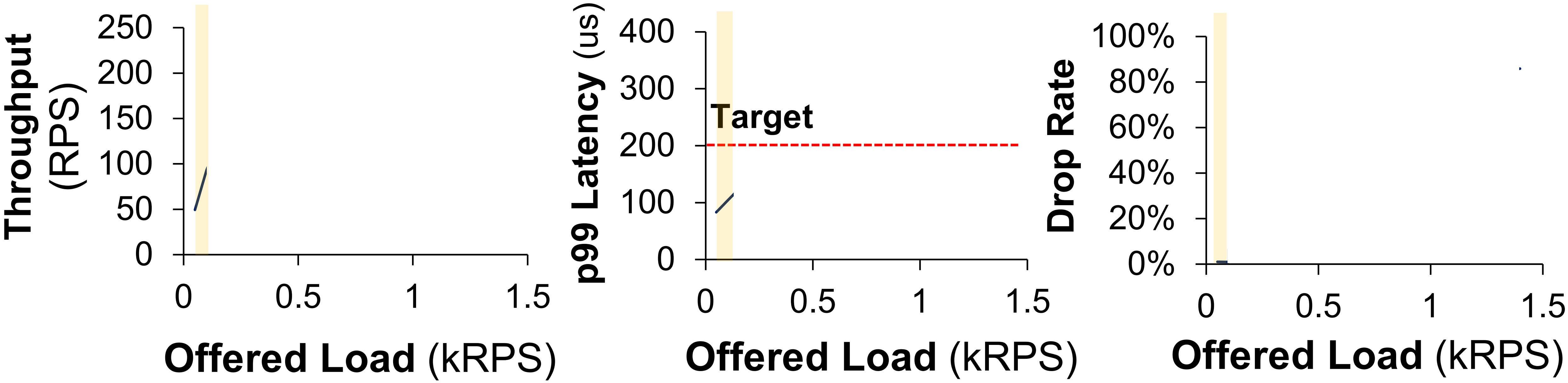
# Performance with ASQM



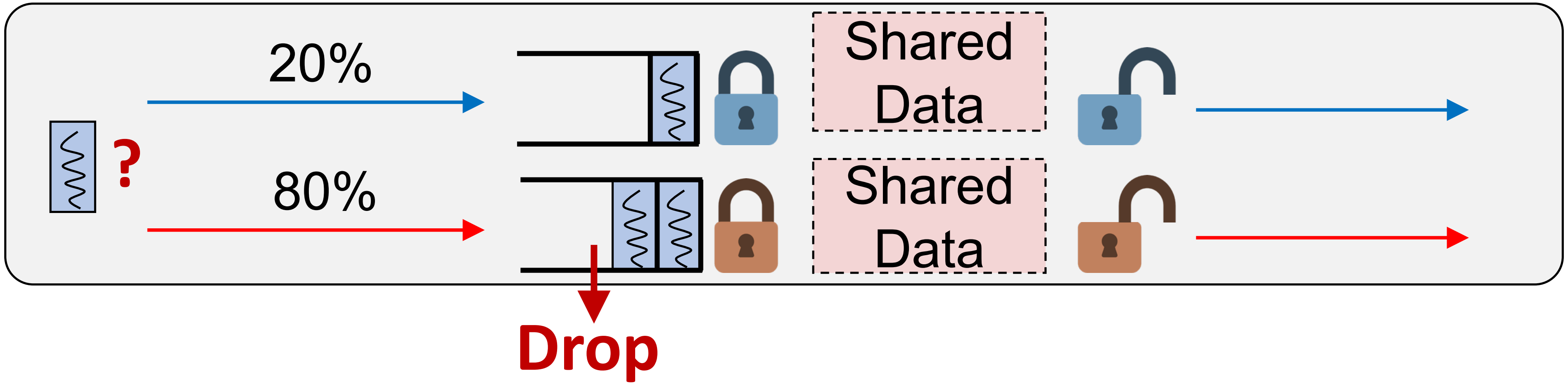


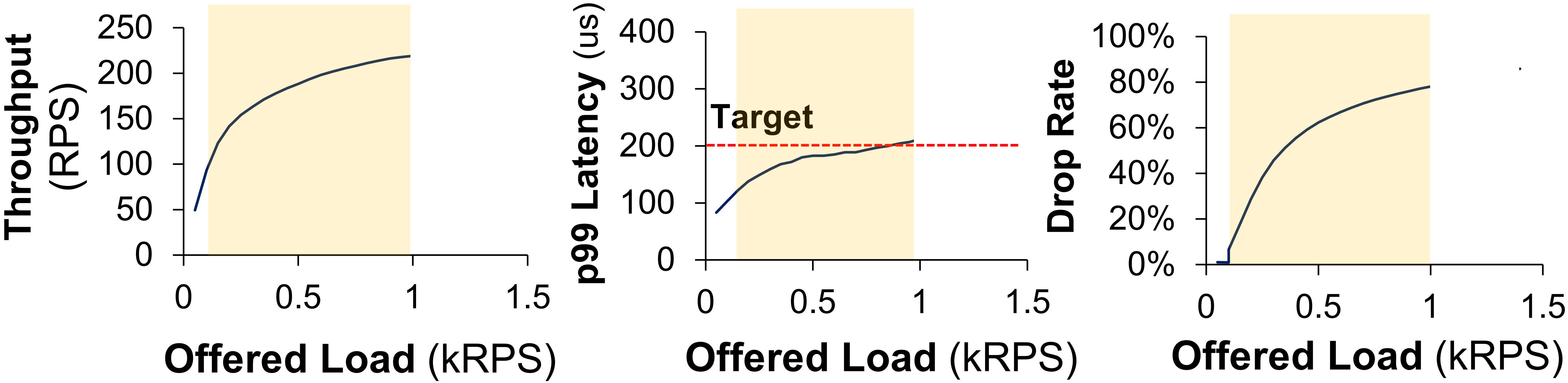
# Performance with ASQM (uncongested)



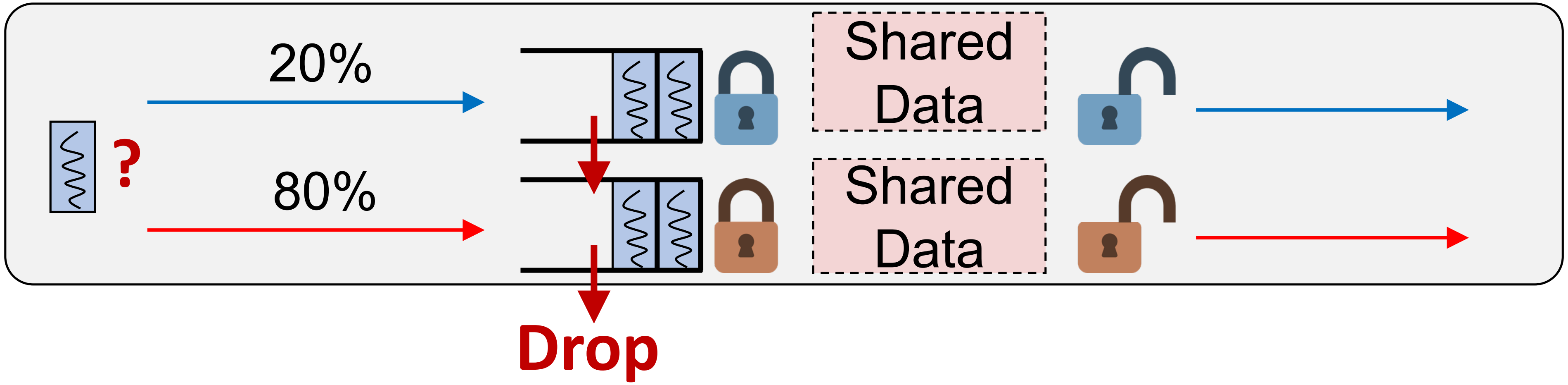


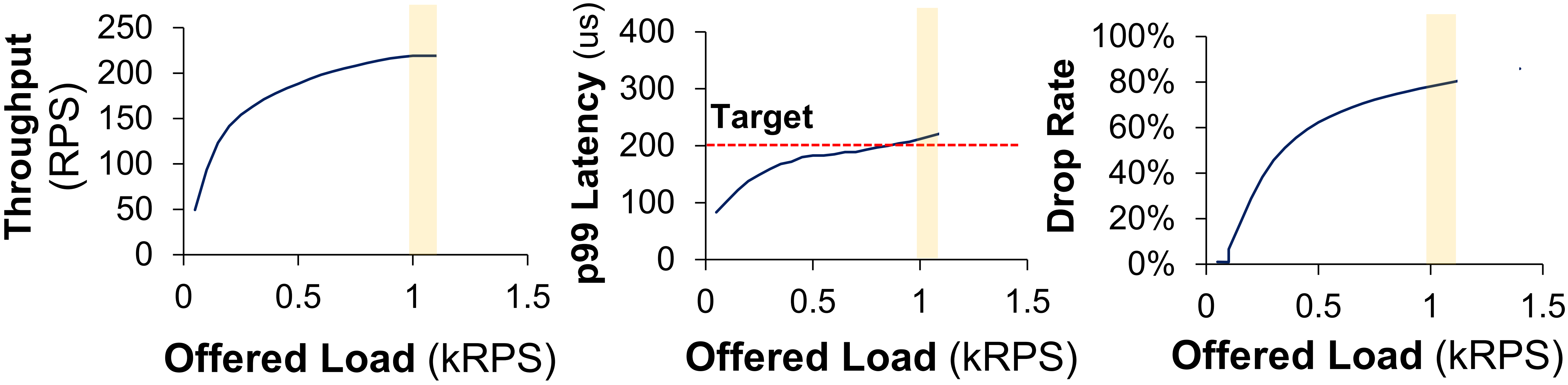
# Performance with ASQM (partially congested)



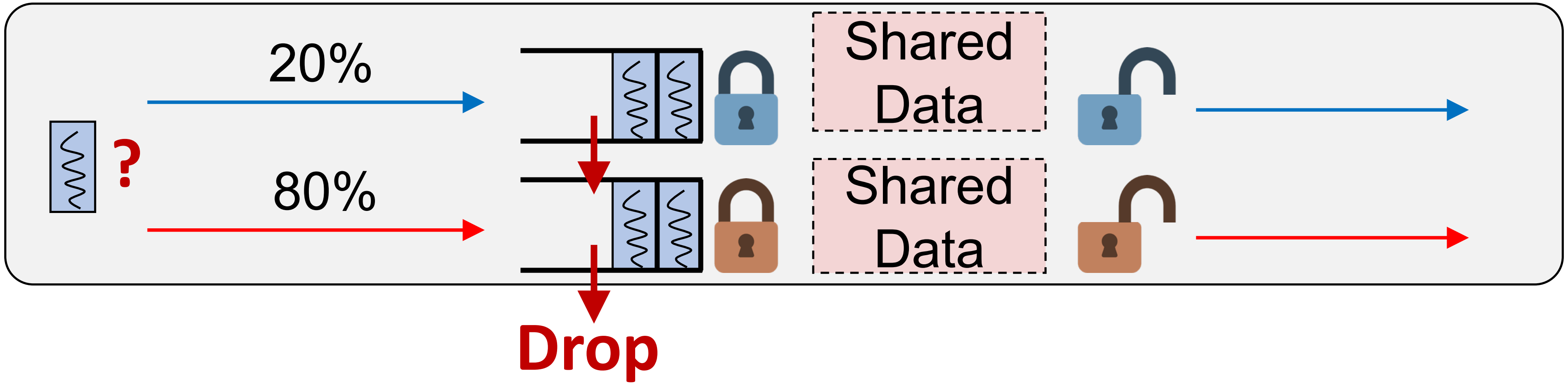


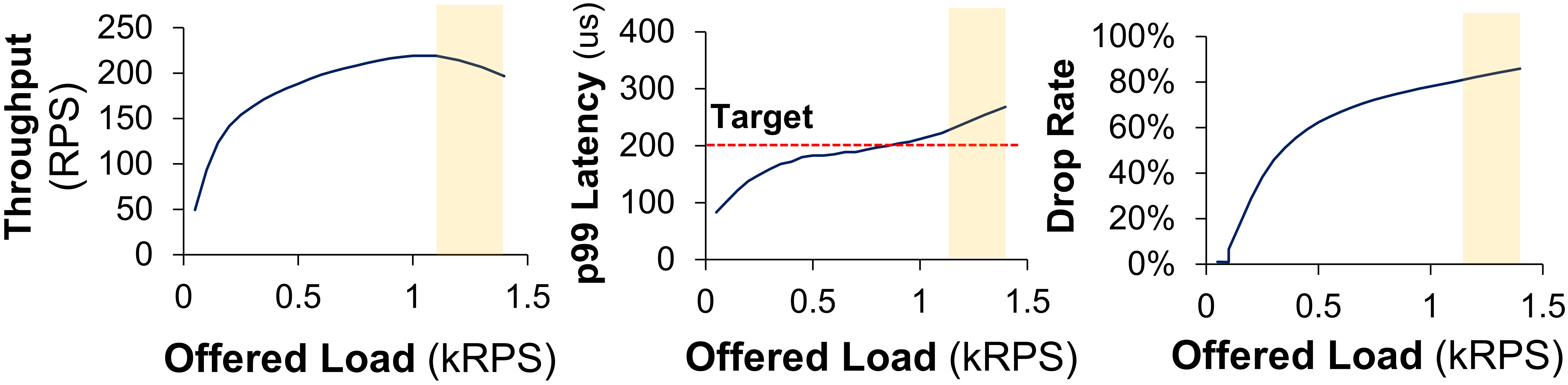
# Performance with ASQM (congested)



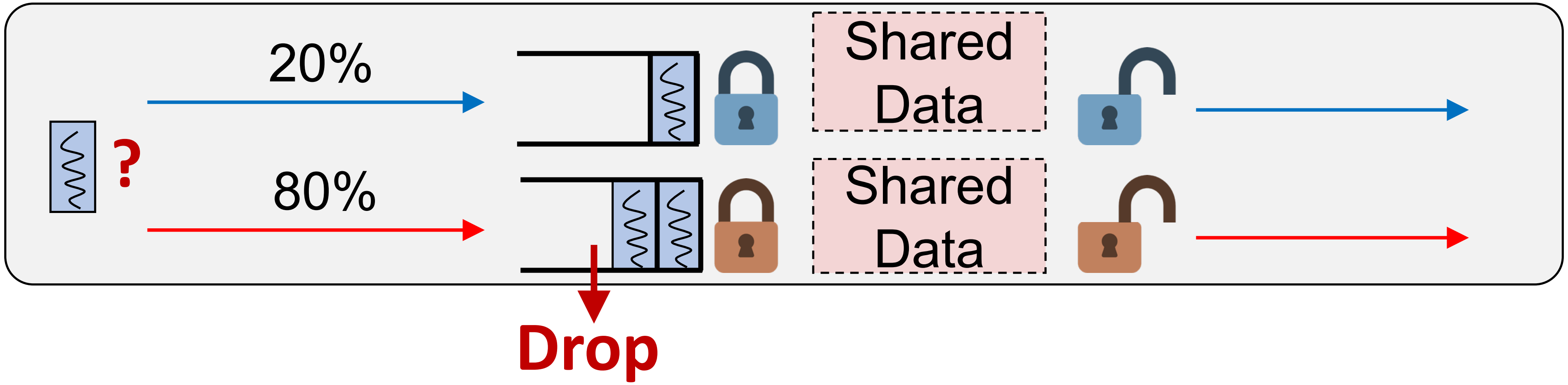


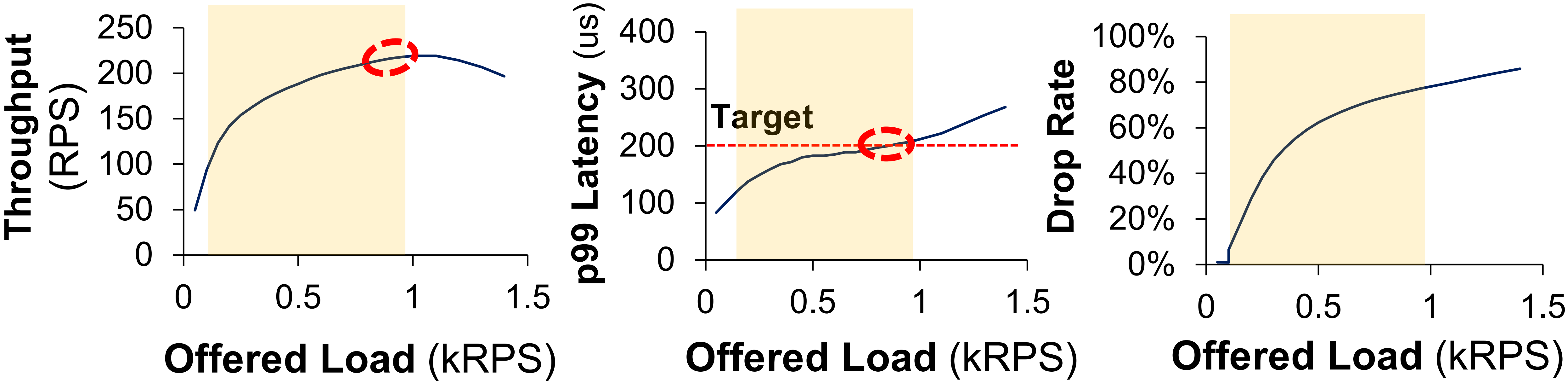
# Performance with ASQM (congestion collapse)



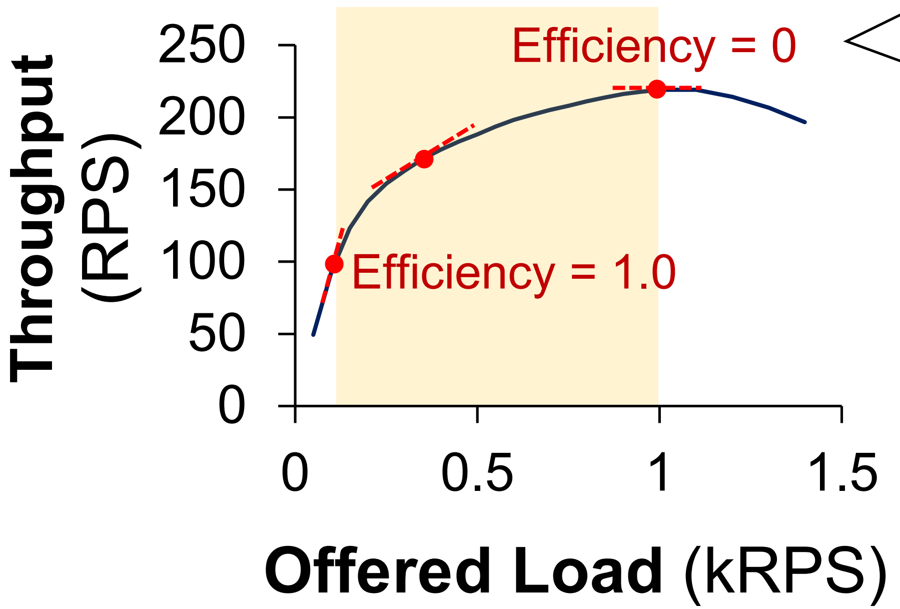


# Ideal Operation Point



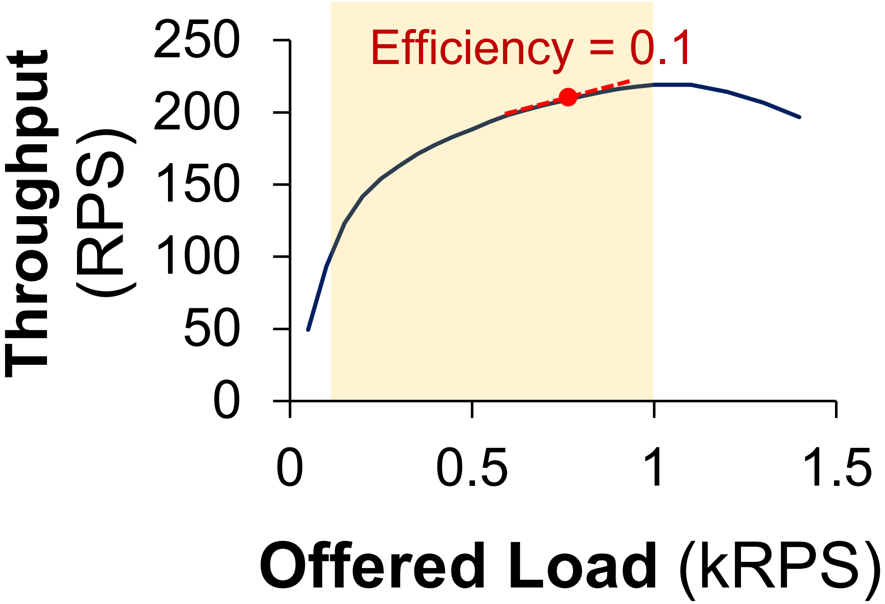


# Performance-driven Admission Control

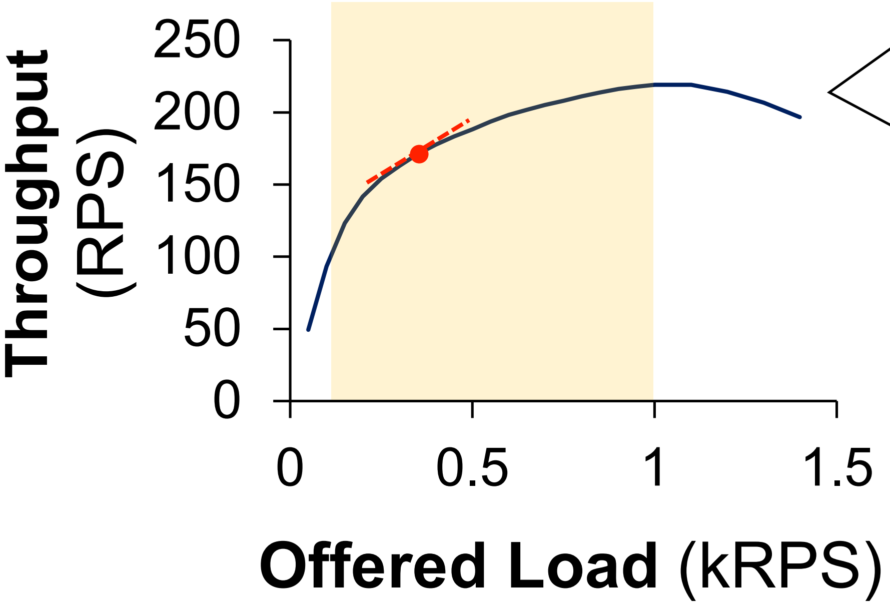


$$\texttt{Efficiency} = \frac{\Delta\, out\_resp}{\Delta\, in\_req}$$

# Performance-driven Admission Control

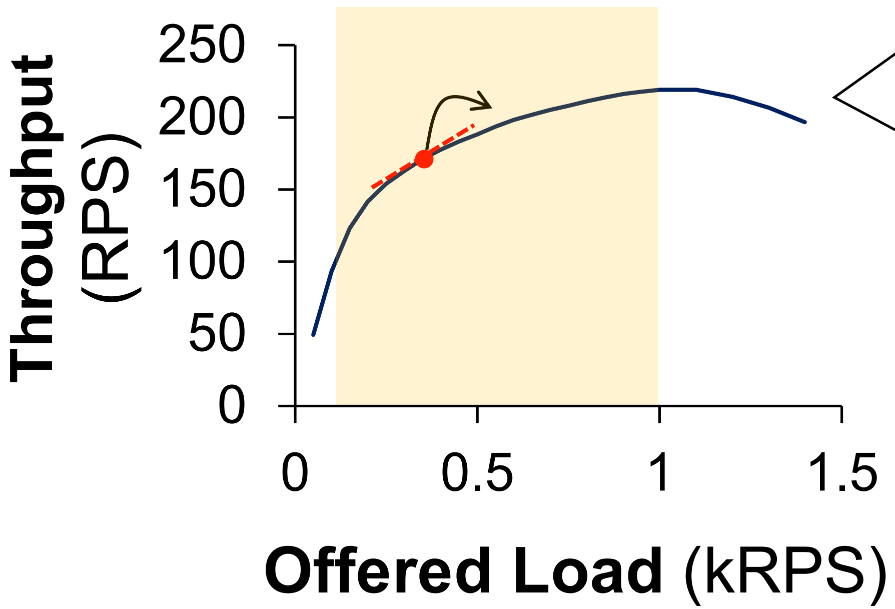


# Performance-driven Admission Control



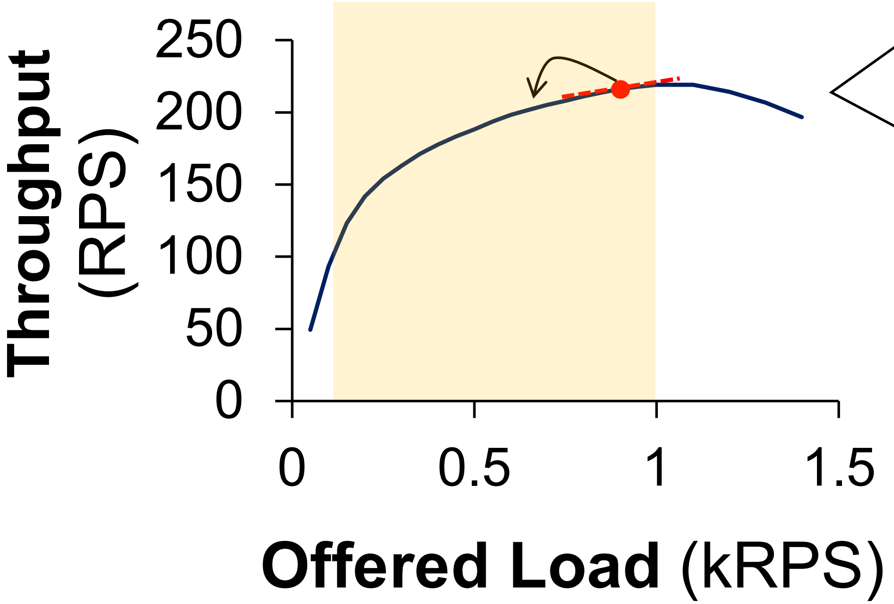
```
For every 4 end-to-end RTTs:
```

# Performance-driven Admission Control



```
For every 4 end-to-end RTTs:
   if **efficiency > target (10%)**:
      admit more load
```

# Performance-driven Admission Control



```
For every 4 end-to-end RTTs:
  if efficiency > target (10%):
    admit more load
  else: # efficiency <= target (10%)
    admit less load
```

# Evaluation

## Testbed Setup

- xl170 in Cloudlab
- 11 machines are connected to a single switch
- 10 client machines / 1 server machine
- Implementation on Shenango as a RPC layer

## Metric

- **Goodput:** Throughput of the responses whose end-to-end latency is less than the target delay

# Evaluation

(1) Does Protego achieves high throughput and low tail latency under unpredictable lock contention?
(2) How fast the client is notified with the rejected requests?
(3) How request drop affects end-to-end latency?

**Baselines:**

**Breakwater**

credit-based overload control with server-side queueing delay

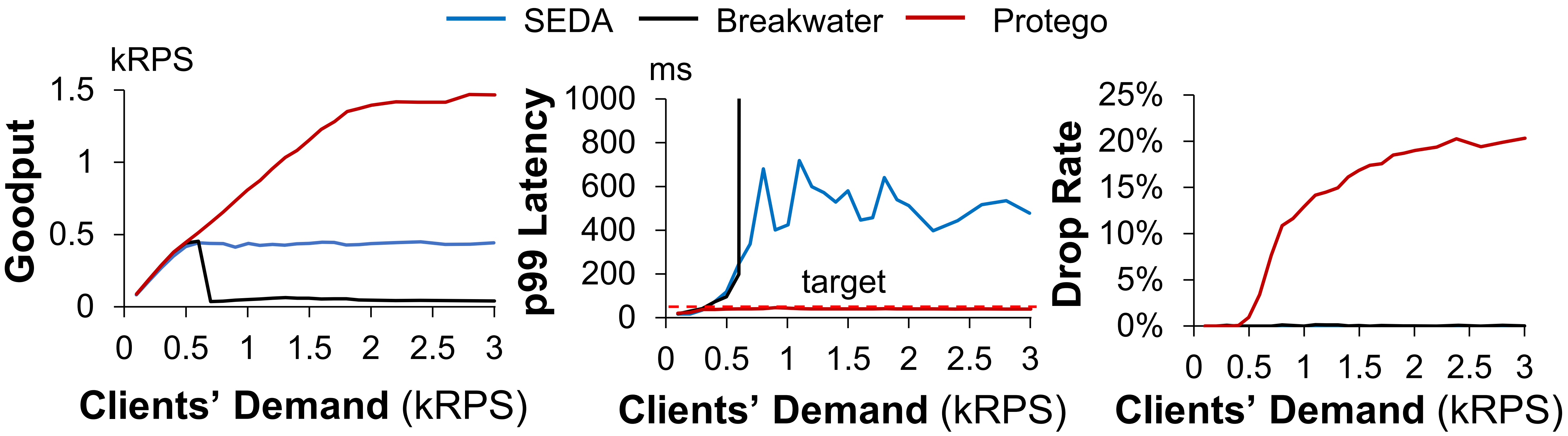
**SEDA**

end-to-end latency-based adaptive overload control for staged event-driven architecture

# Evaluation: Lucene

## COVID Tweet workload

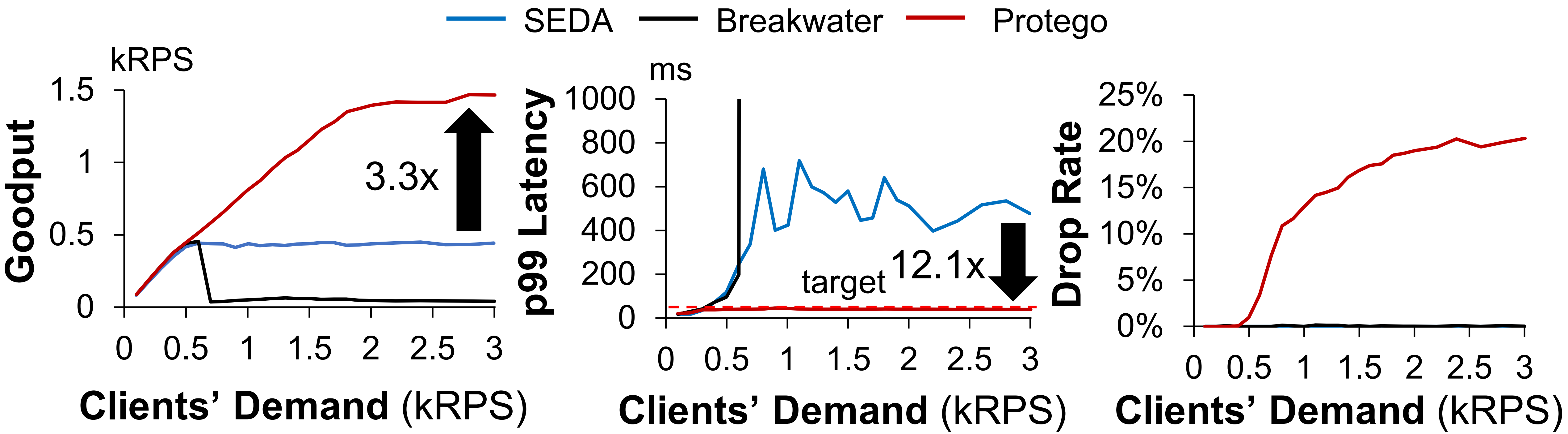
- 403,619 COVID-related tweets
- Query word distribution follows word distribution in tweets



# Evaluation: Lucene

## COVID Tweet workload

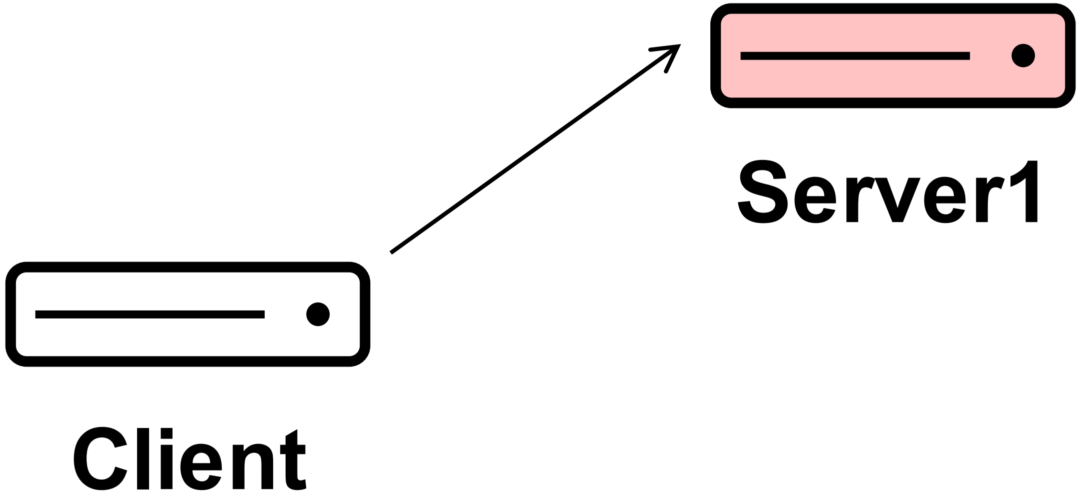
- 403,619 COVID-related tweets
- Query word distribution follows word distribution in tweets



# Evaluation: Memcached

## SET-heavy workload (VAR)

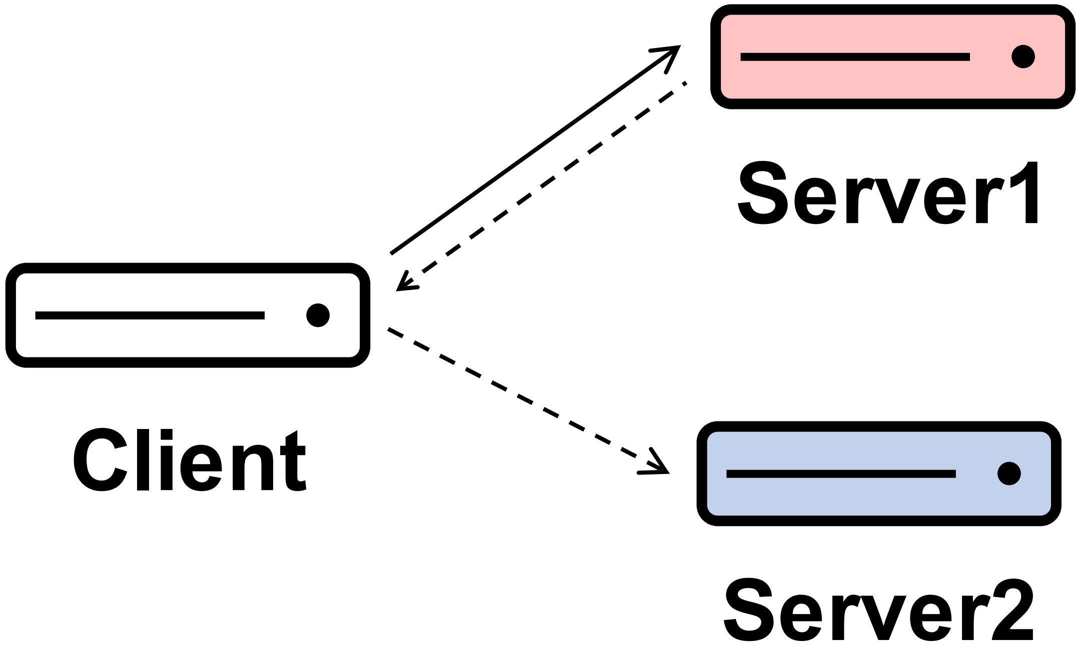
- 82% SET, 18% GET requests
- 10% of the key used by 90% of the requests



# Evaluation: Memcached

## SET-heavy workload (VAR)

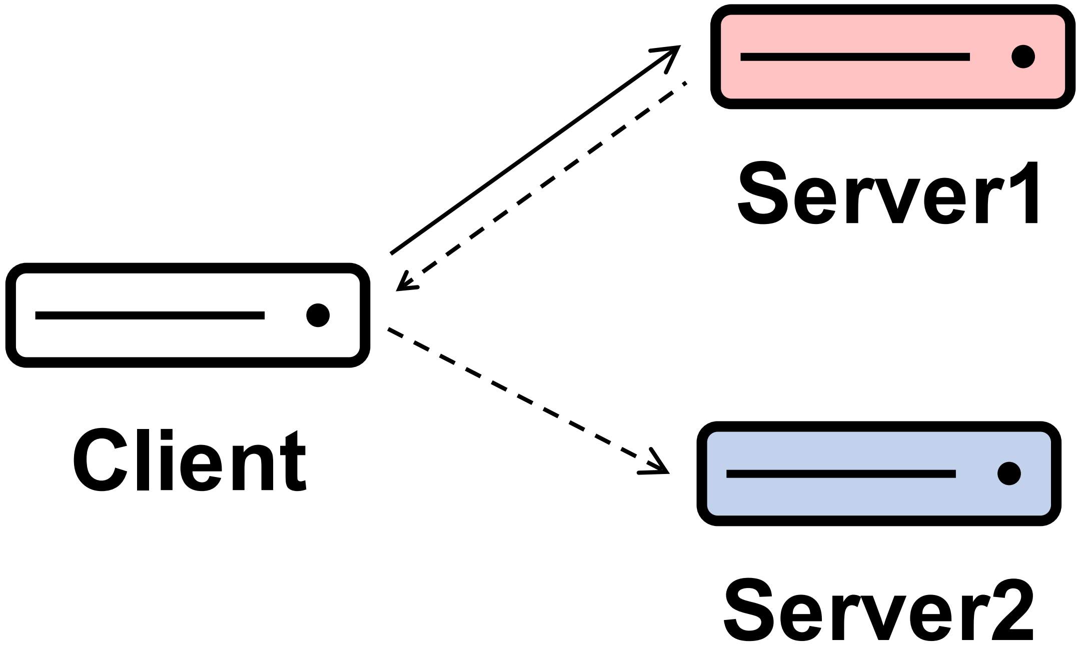
- 82% SET, 18% GET requests
- 10% of the key used by 90% of the requests

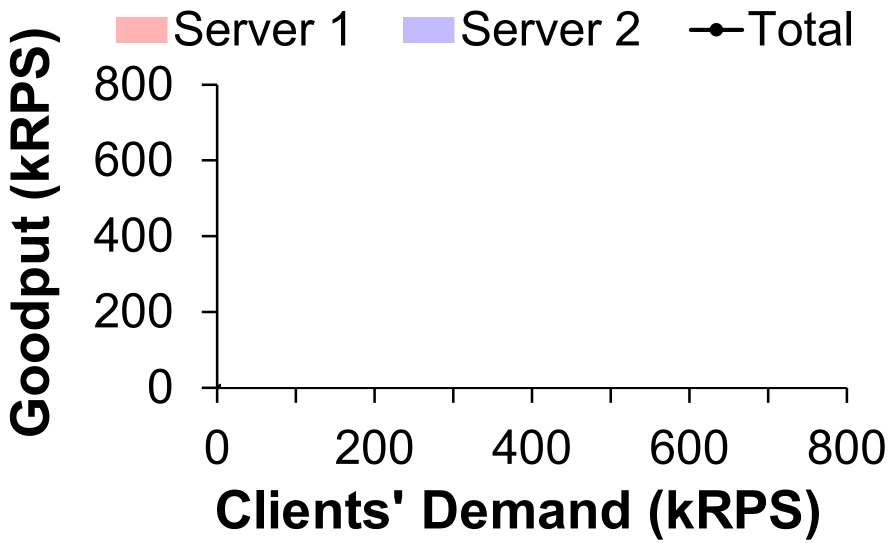


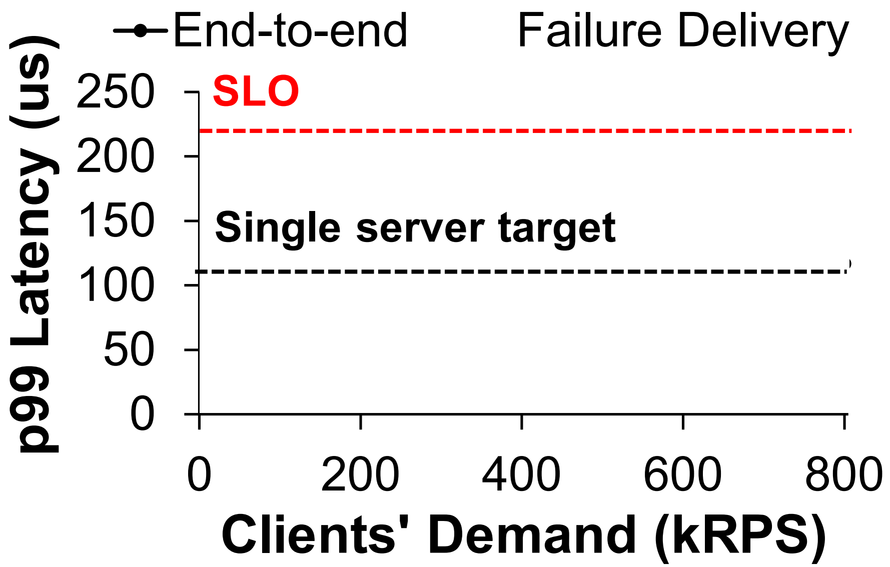
# Evaluation: Memcached

## SET-heavy workload (VAR)

- 82% SET, 18% GET requests
- 10% of the key used by 90% of the requests



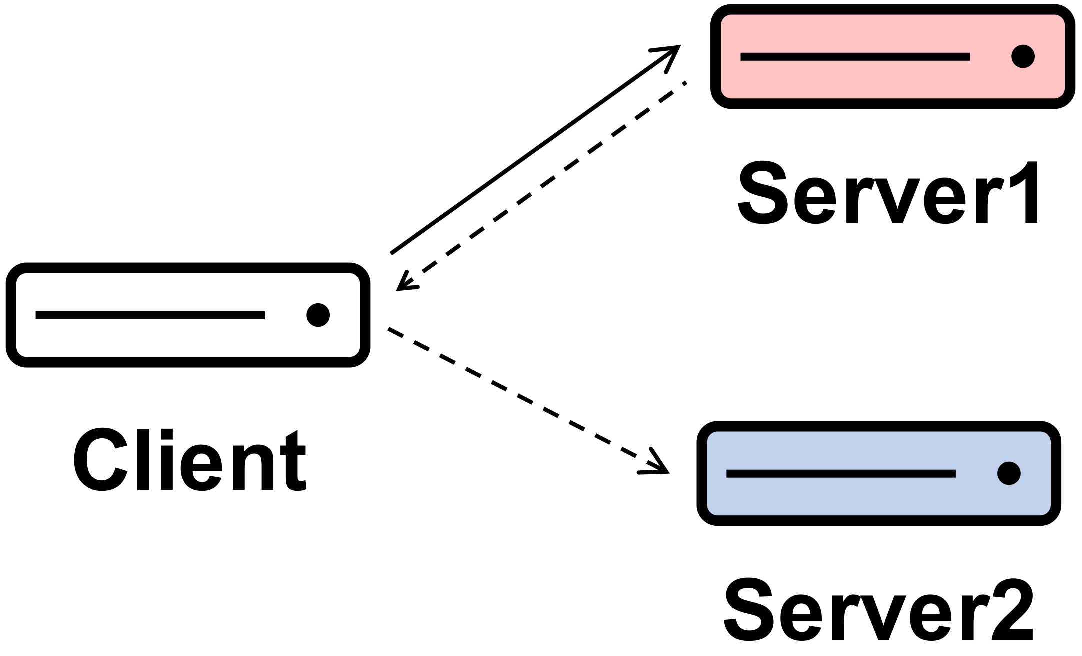


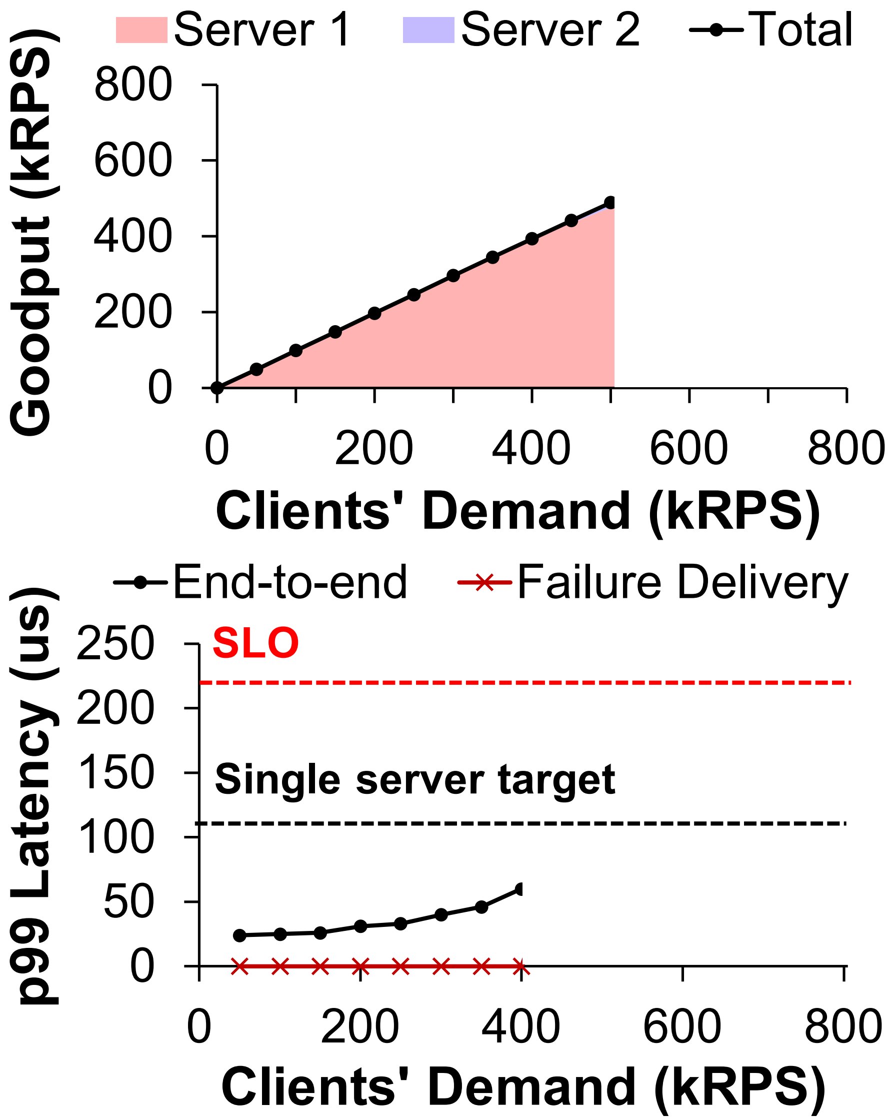


# Evaluation: Memcached

## SET-heavy workload (VAR)

- 82% SET, 18% GET requests
- 10% of the key used by 90% of the requests

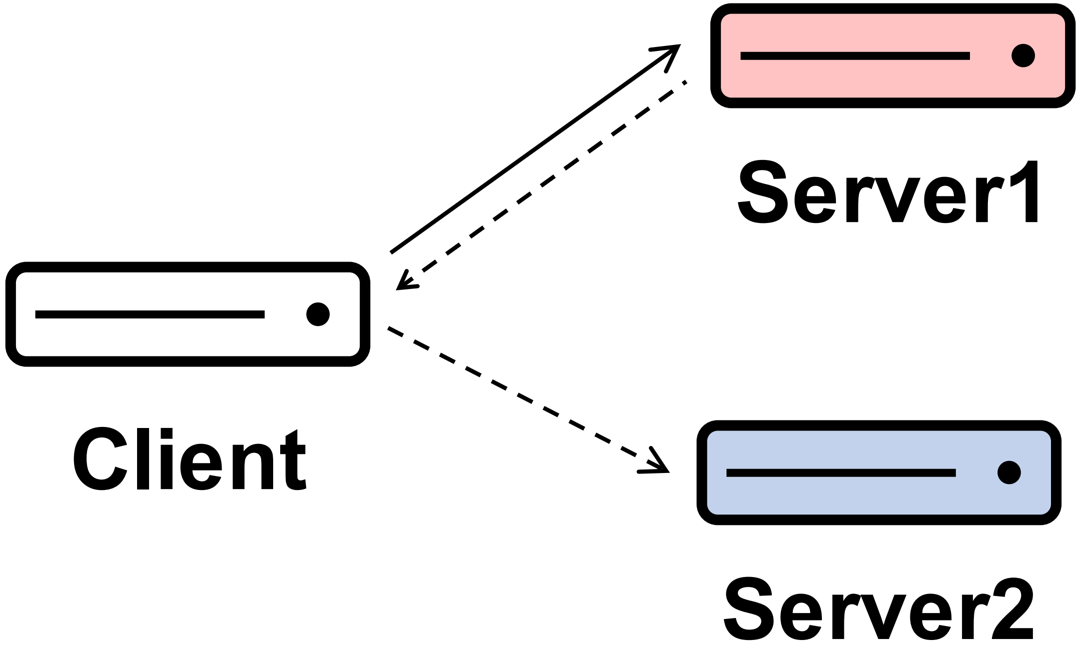


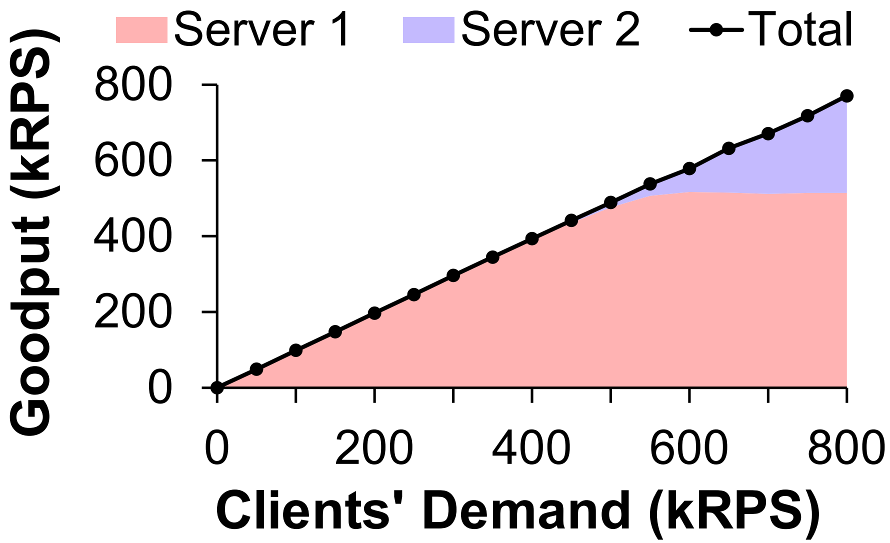


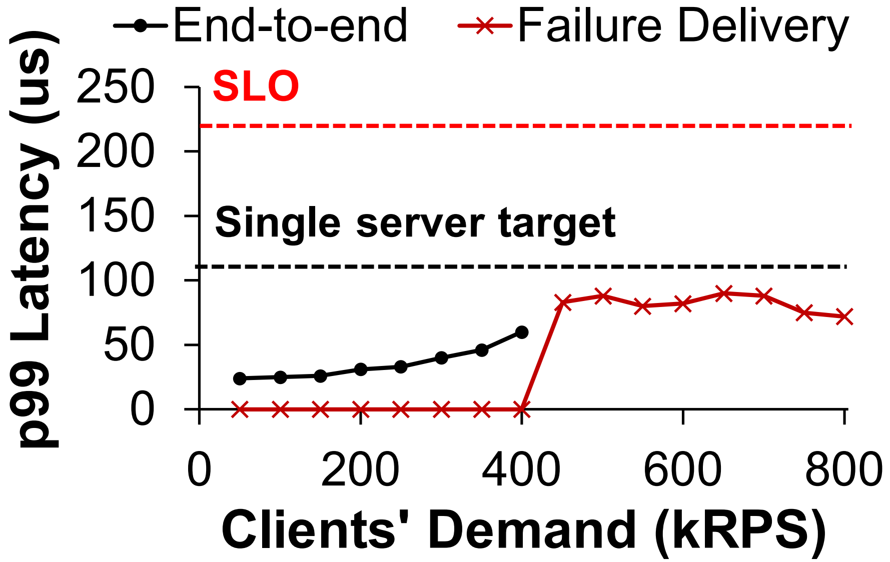
# Evaluation: Memcached

## SET-heavy workload (VAR)

- 82% SET, 18% GET requests
- 10% of the key used by 90% of the requests



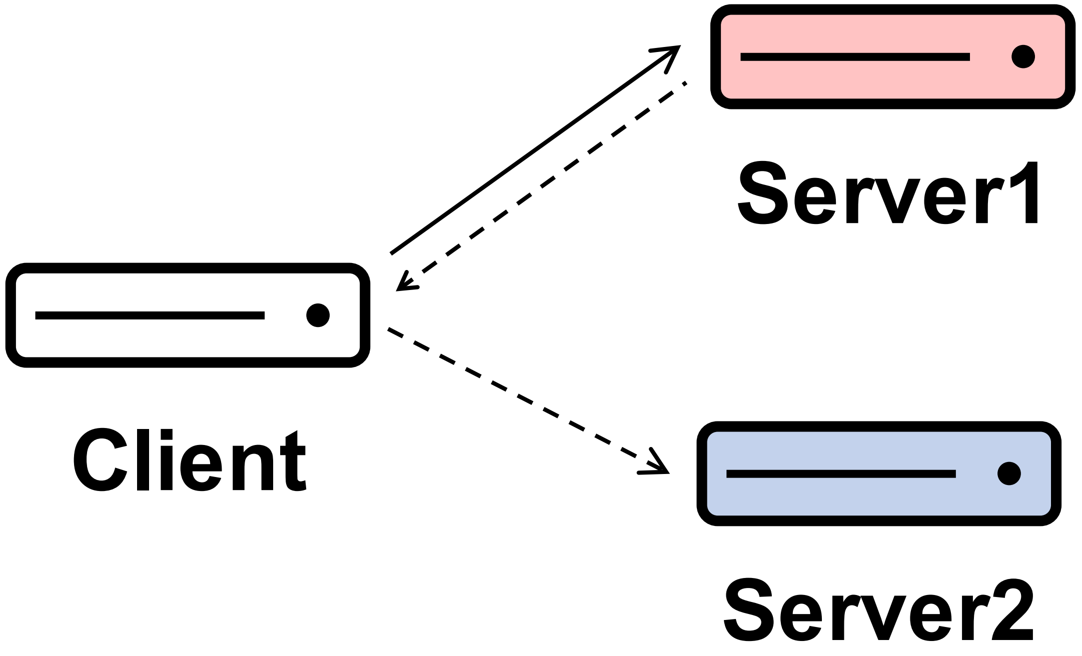


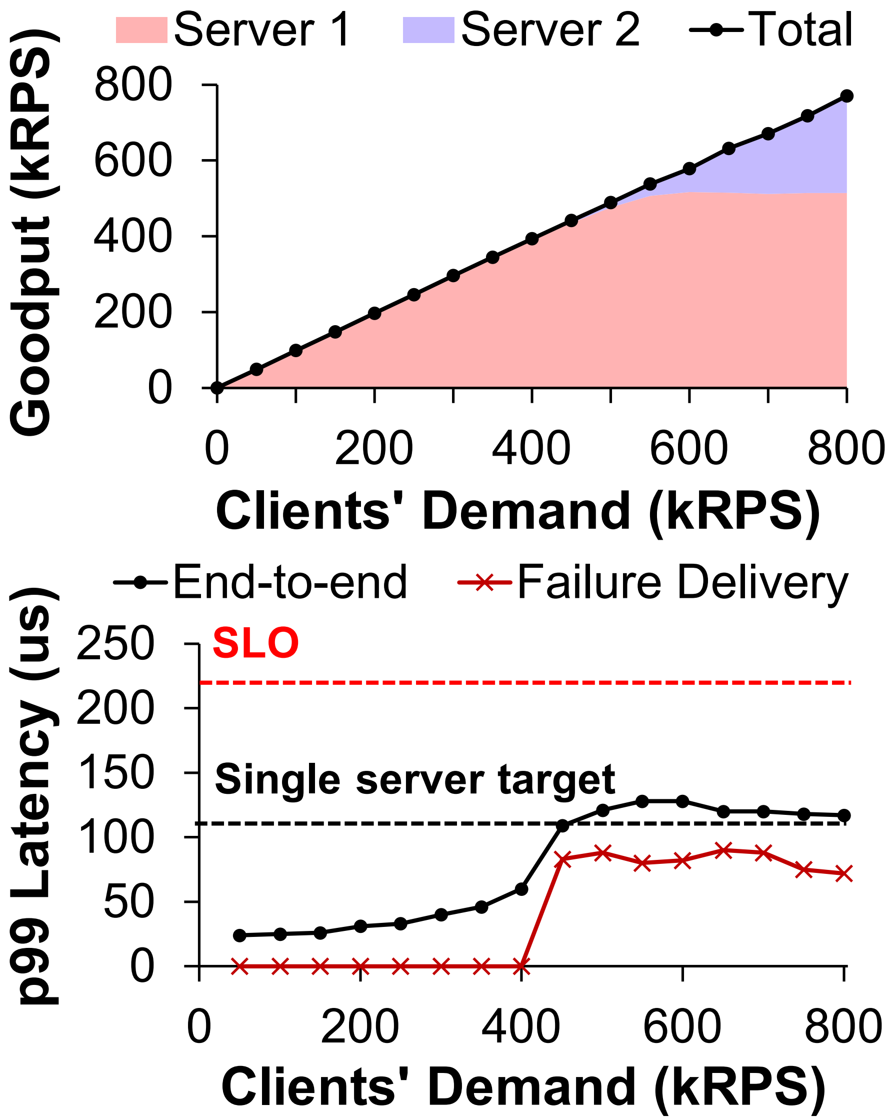


# Evaluation: Memcached

## SET-heavy workload (VAR)

- 82% SET, 18% GET requests
- 10% of the key used by 90% of the requests





# Conclusion

- Protego is an overload control designed to handle **unpredictable lock contention** effectively.
- Protego's key components include
  (1) Active Synchronization Queue Management (ASQM)
  (2) Performance-based Admission Control
- Our evaluation shows that Protego achieves
  (1) **High throughput** and **low tail latency** under unpredictable lock contention
  (2) **On-time failure delivery** for a rejected request
  (3) **Effective scaling** to multiple machines

# Thank you!