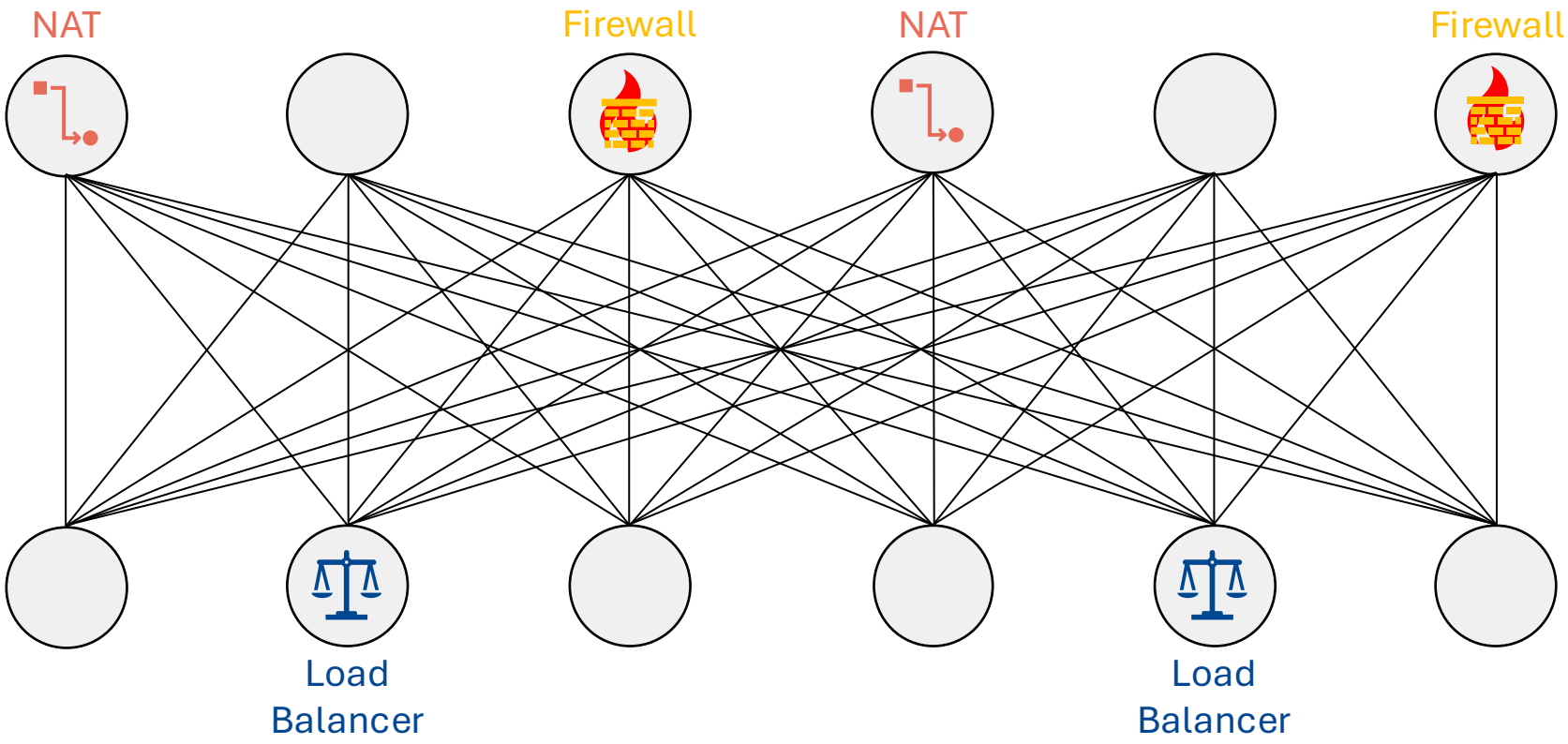
Queue-Mem: Energy-Efficient Hardware Storage for Advanced Network Function Acceleration

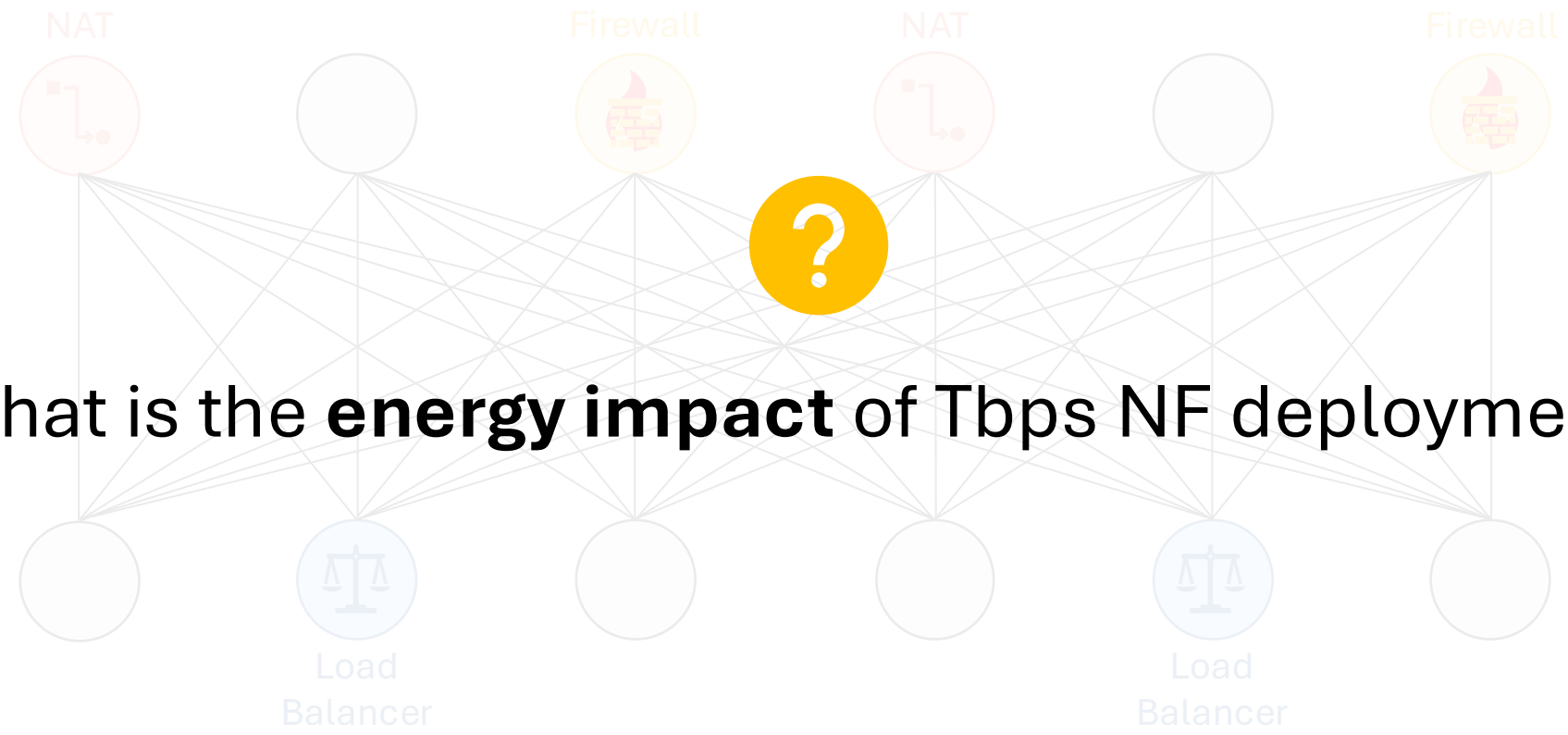
Mariano Scazzariello [†], Tommaso Caiazzi [‡], Hamid Ghasemirahni [*], Dejan Kostić [*], Marco Chiesa [*]

[†] RISE Research Institutes of Sweden
[‡] Roma Tre University
[*] KTH Royal Institute of Technology

Network Functions Are Pervasive

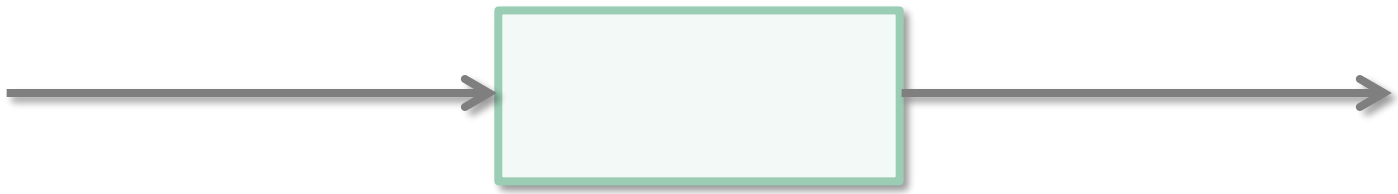


Network Functions Are Pervasive

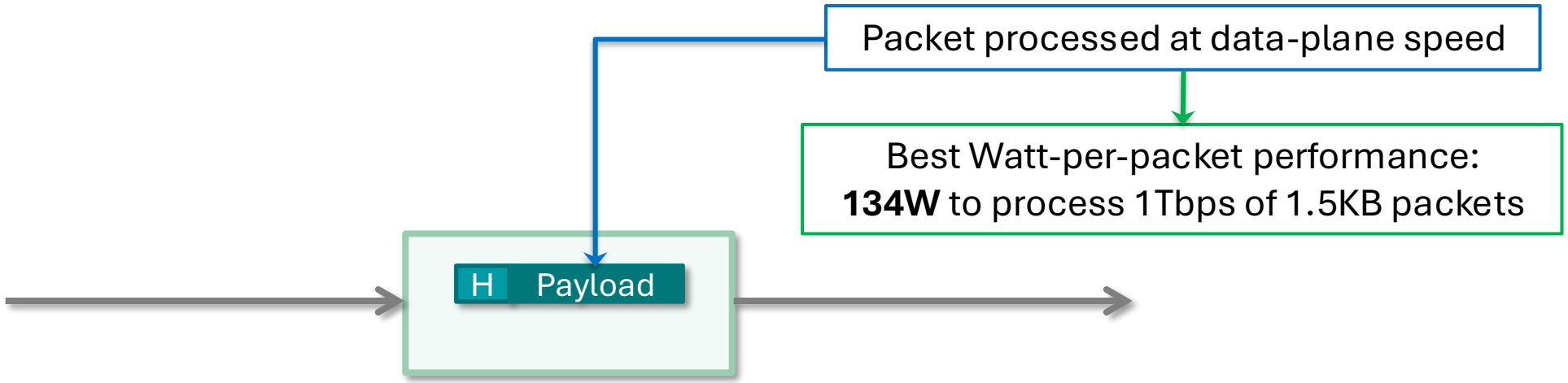


What is the **energy impact** of Tbps NF deployments?

Ideal NF Deployment

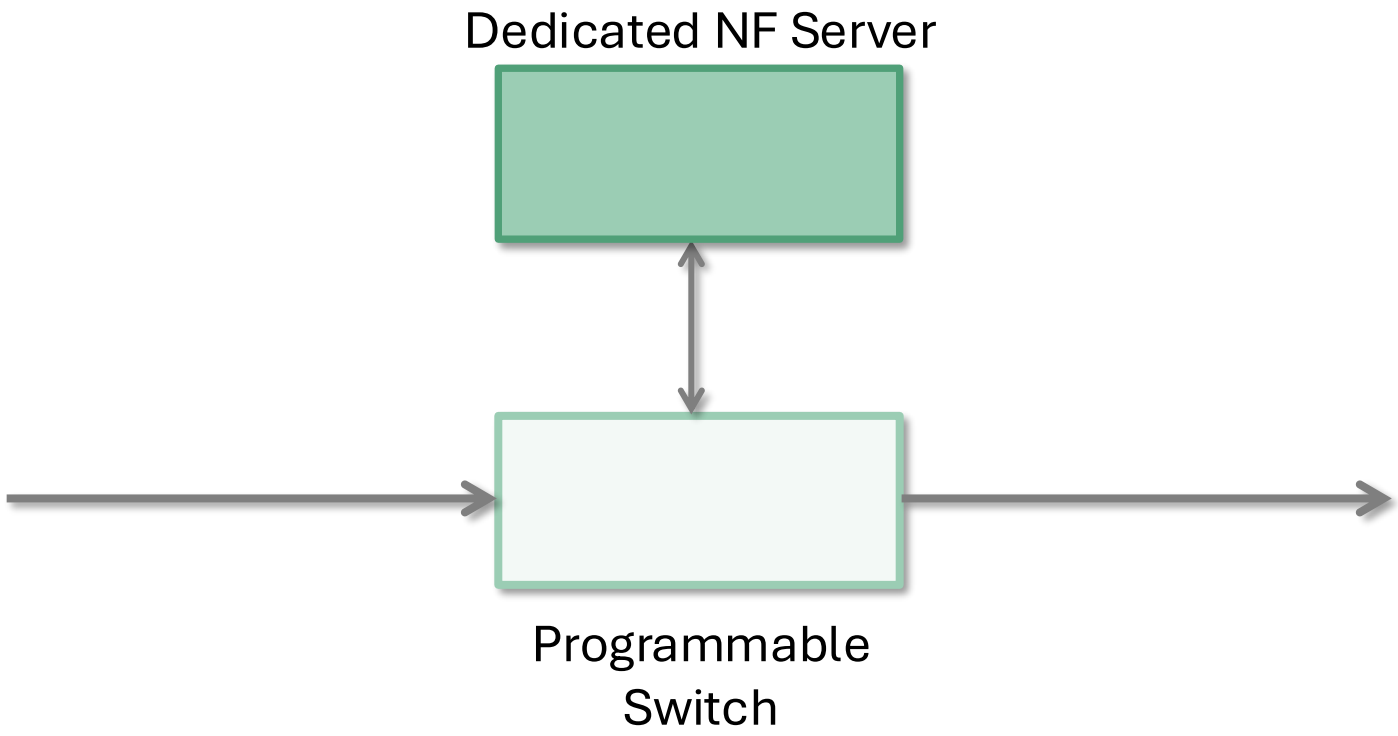


Ideal NF Deployment

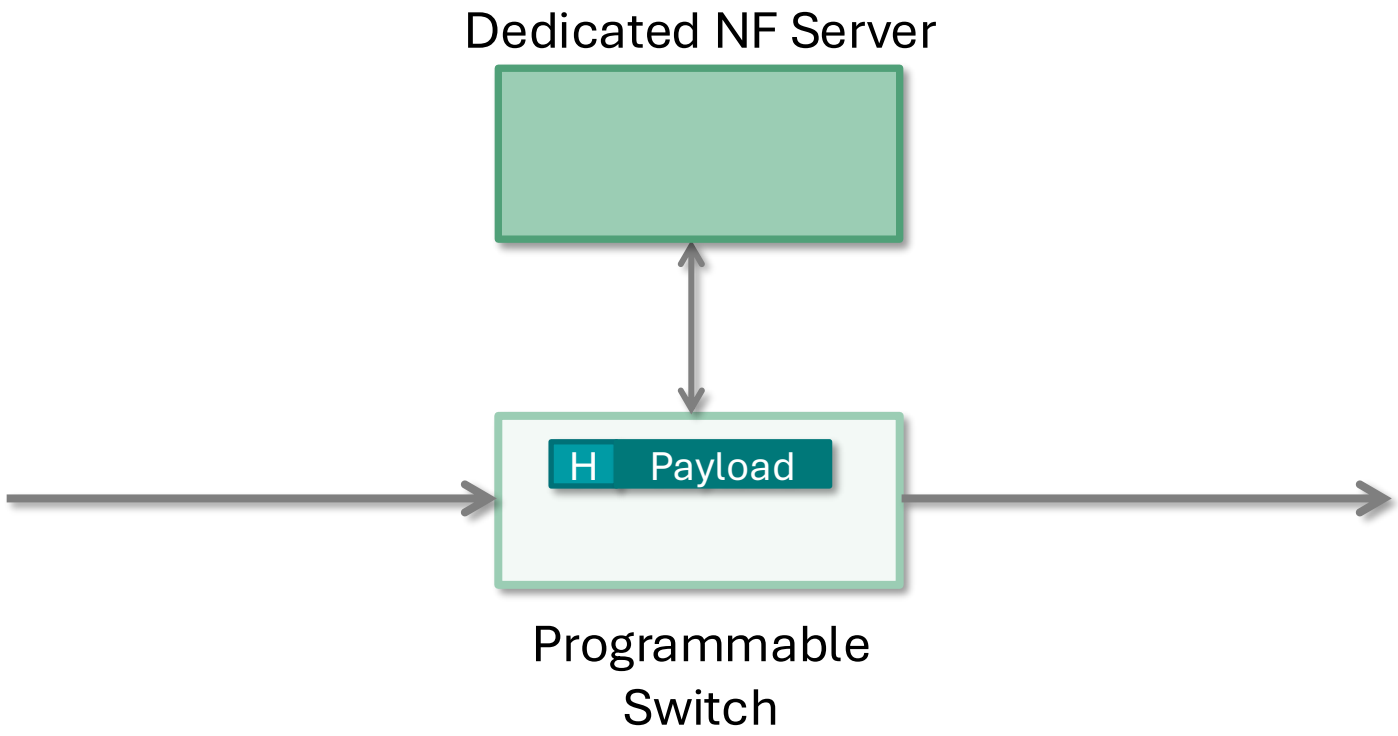


❌ **No support** for advanced stateful NFs!

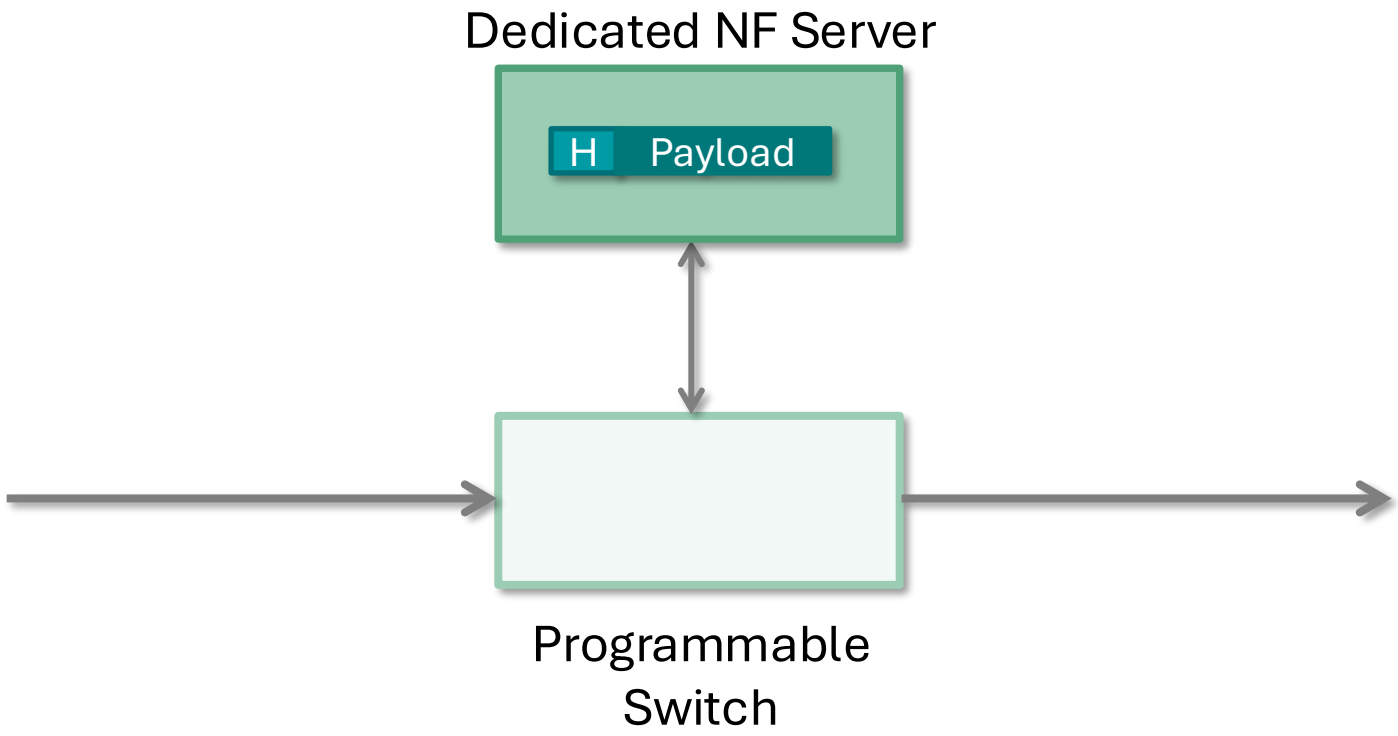
Advanced NF Deployments



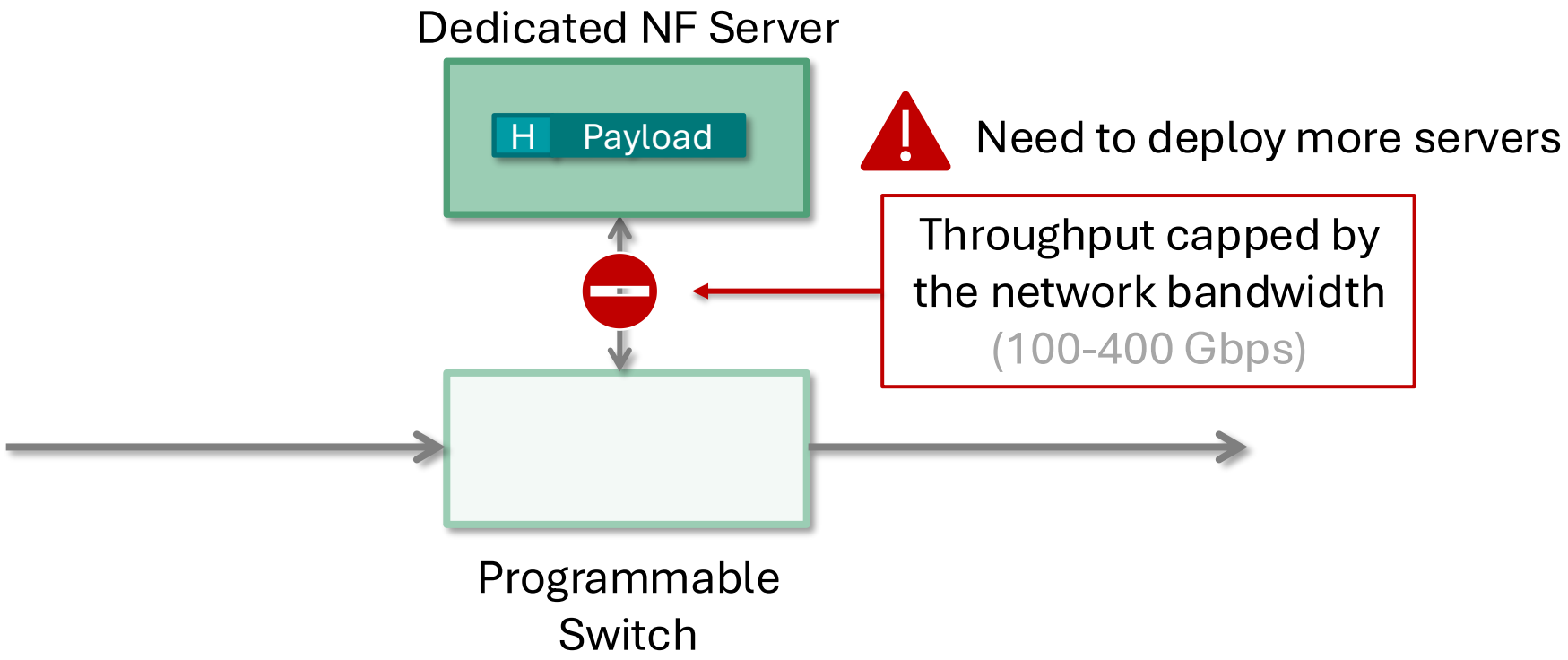
Advanced NF Deployments



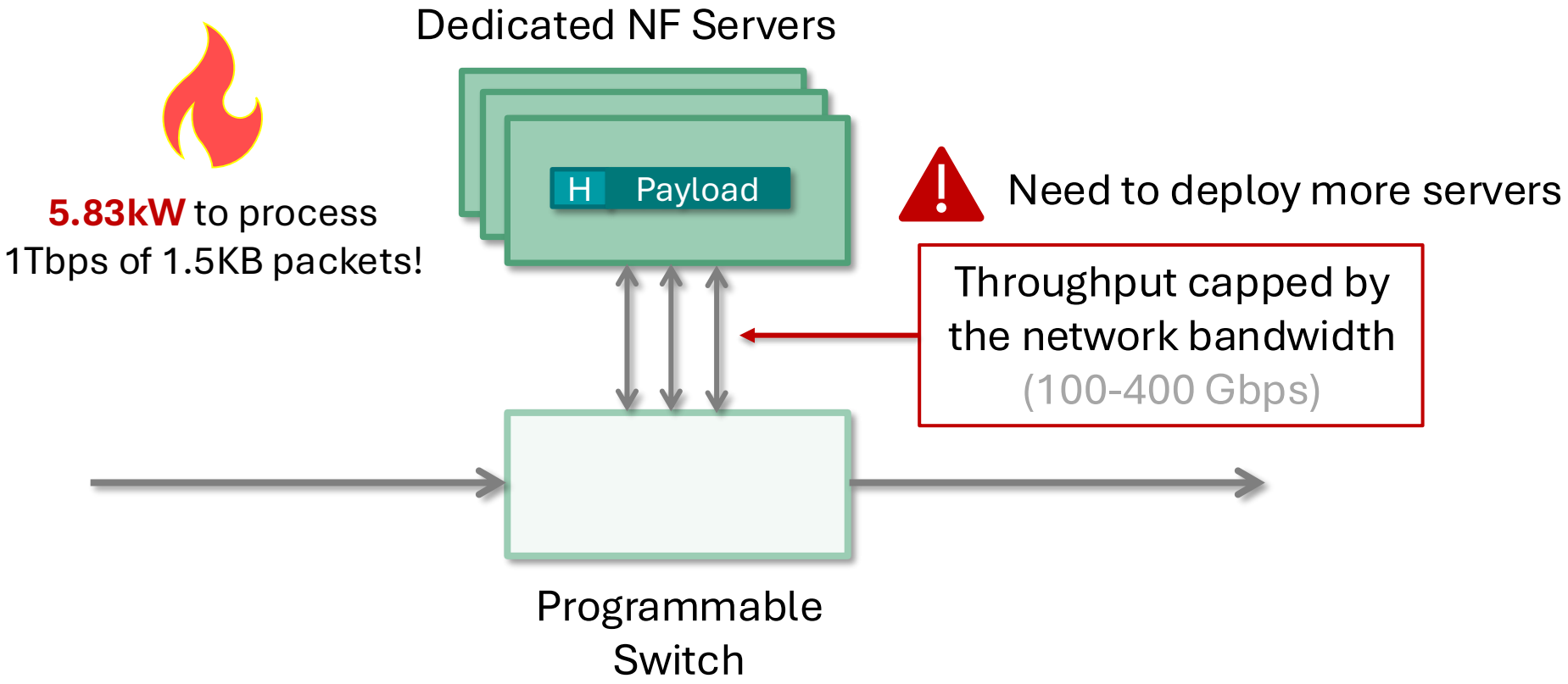
Advanced NF Deployments



The Bandwidth Limit of Advanced NF Deployments



The Bandwidth Limit of Advanced NF Deployments



The Bandwidth Limit of Advanced NF Deployments

Many Network Functions need only headers!

Packet-Splitting Approaches

Packet-Splitting Approaches

State-of-the-art packet-splitting solutions increase NFs throughput

Improve NF Performance

Better CPU cache exploitation

Free up Bandwidth

Reduce the number of dedicated NF servers

Lower energy consumption

Complex Deployment

Require end-host modifications

Require available shared resources

Packet-Splitting Approaches

State-of-the-art packet-splitting solutions increase NFs throughput

Storing Payloads in the On-NIC Memory (nicmem)

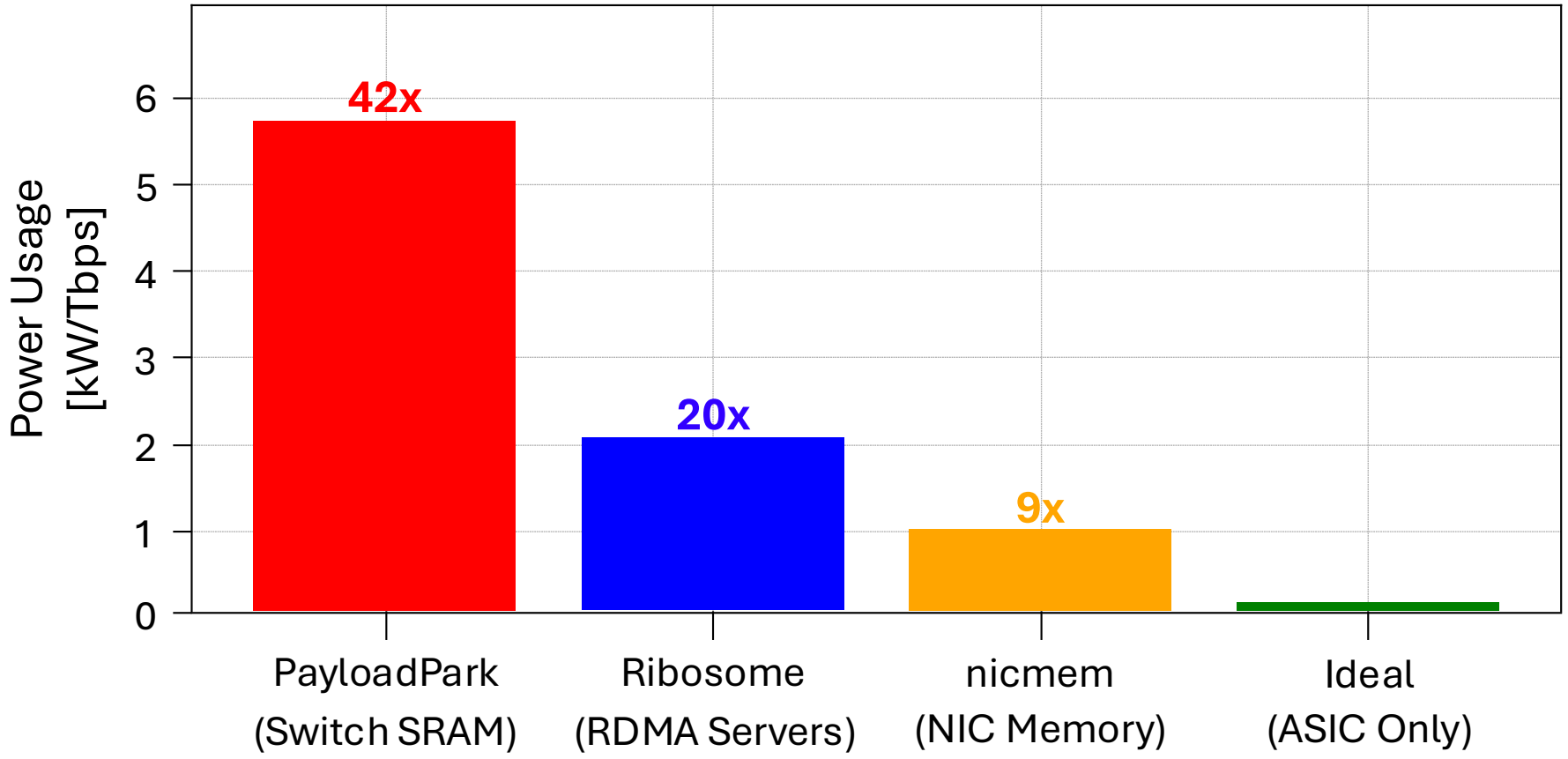
Storing Payloads on Shared Resources (Ribosome)

Storing a Few Bytes in ASIC SRAM (PayloadPark)

Energy Consumption of Packet-Splitting

State-of-the-art packet-splitting solutions increase NFs throughput

… but they are **power inefficient**!



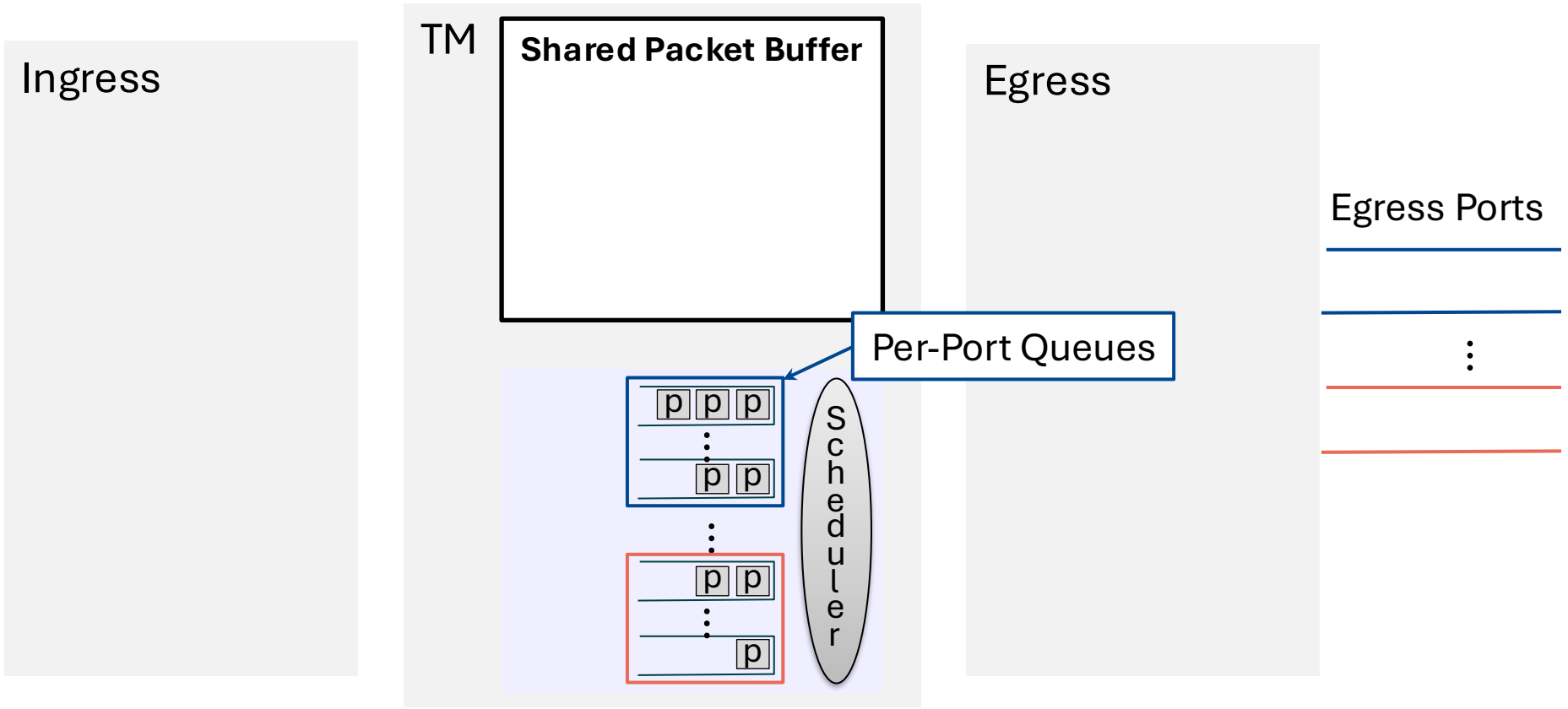
Can we design a system that is power-efficient, easy-to-deploy, and supports advanced network functions?

Core Idea: A Queue-Based Packet Storage

Exploit the switch shared buffer to store payloads

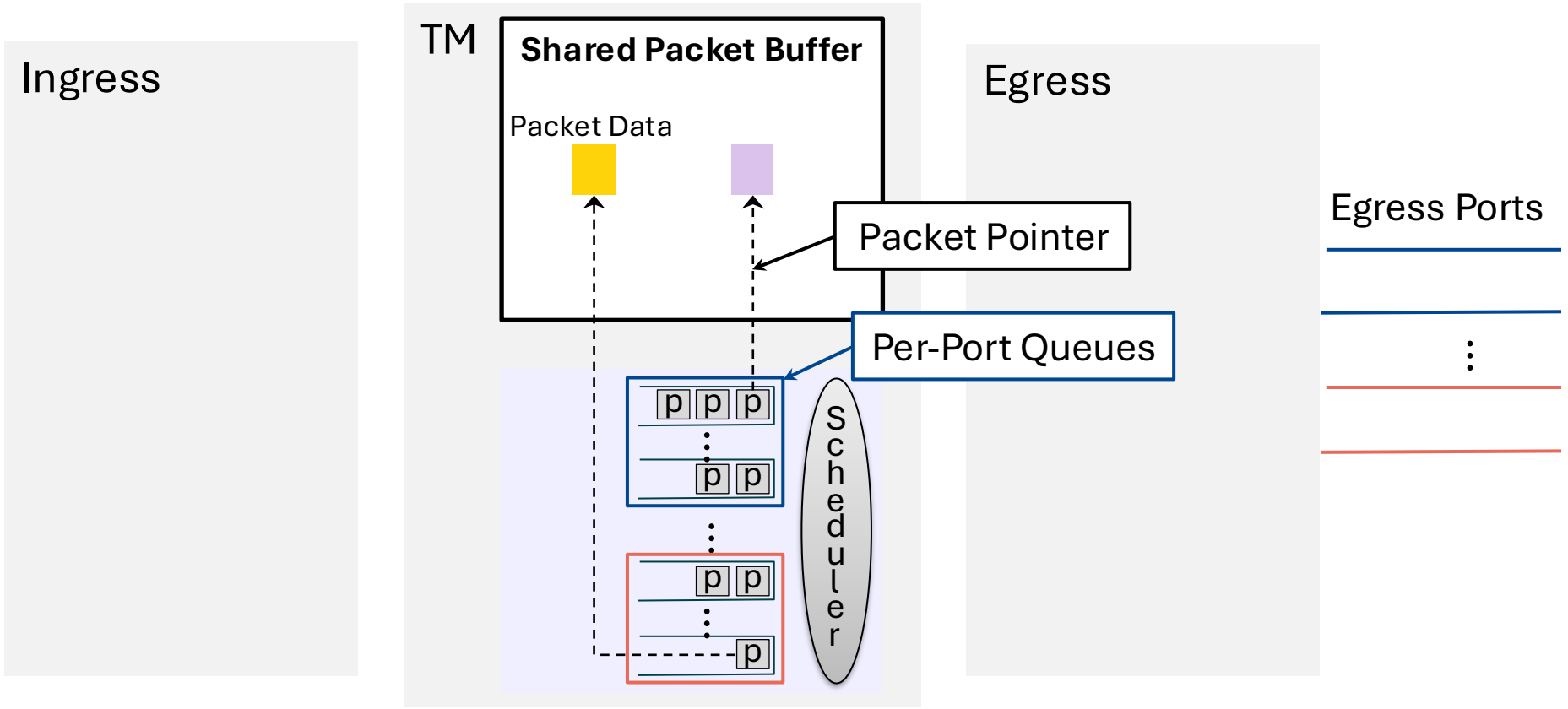
Core Idea: A Queue-Based Packet Storage

Exploit the switch shared buffer to store payloads



Core Idea: A Queue-Based Packet Storage

Exploit the switch shared buffer to store payloads



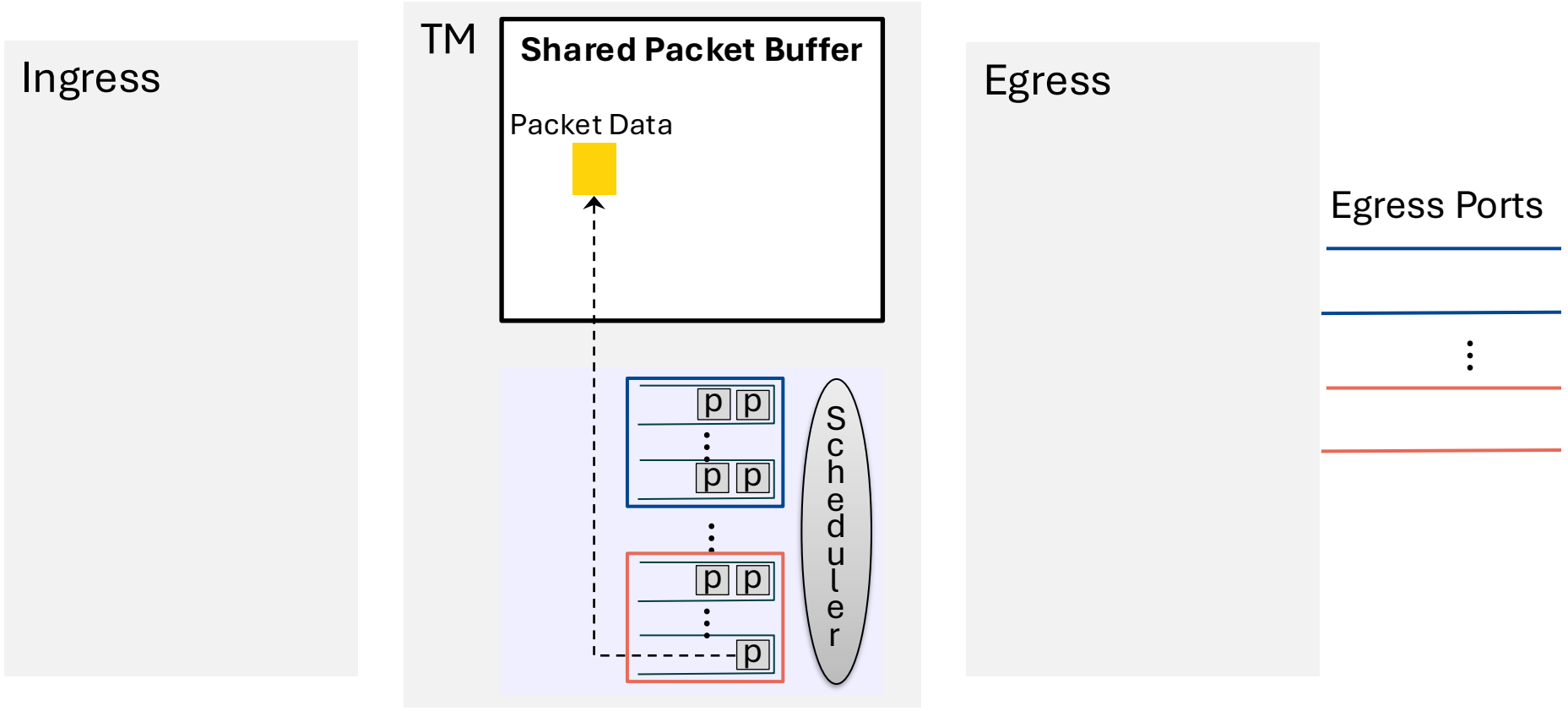
Core Idea: A Queue-Based Packet Storage

Exploit the switch shared buffer to store payloads

✓ Large shared buffer

✗ No programmatic access

? How can we access the packet buffer?



How can we access the packet buffer?

Enqueued packets **implicitly** control the buffer memory

Hold payloads in the port queues until
the processed header from the NF is returned!

How can we control the queues?

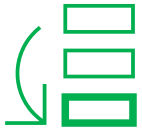
How can we control the queues?

Recirculate payloads until headers are ready

- ❌ Each input port requires a loopback port, **halving switch's throughput**
- ❌ May introduce **packet reordering**

How can we control the queues?

Recirculate payloads until headers are ready



Create copies of payloads to fill up the queues

- ✓ Exploits **full switch's bandwidth**
- ✗ Fills the buffer with dummy copies that **reduce available space**
- ✗ May **release payloads before** processed headers are received back

How can we control the queues?

Recirculate payloads until headers are ready

Create copies of payloads to fill up the queues

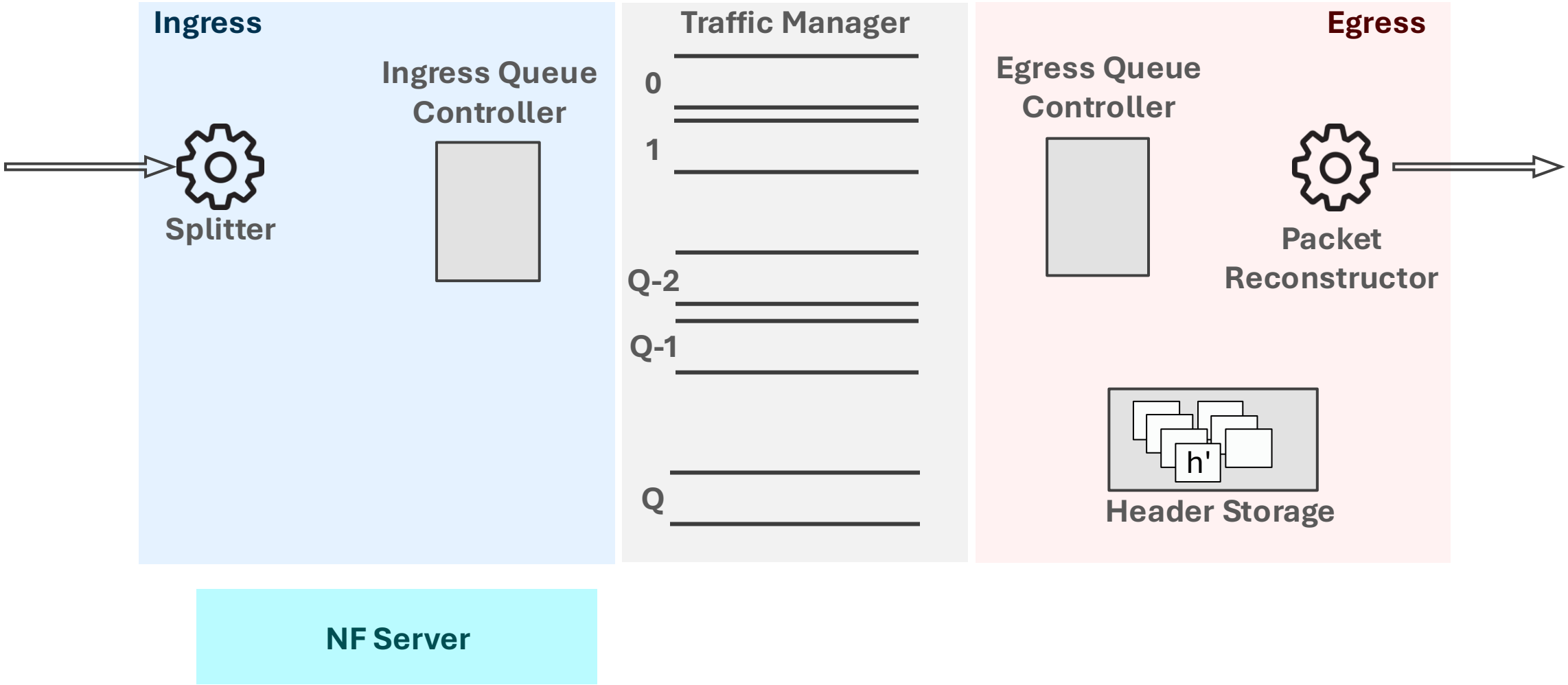
Dynamically pause/resume queues holding payloads

- ✓ Exploits **full switch's bandwidth**
- ✓ **Minimum buffer space utilization** (only what the payloads require)
- ✓ **Supported on modern switches** (for Priority Flow Control)

Queue-Mem

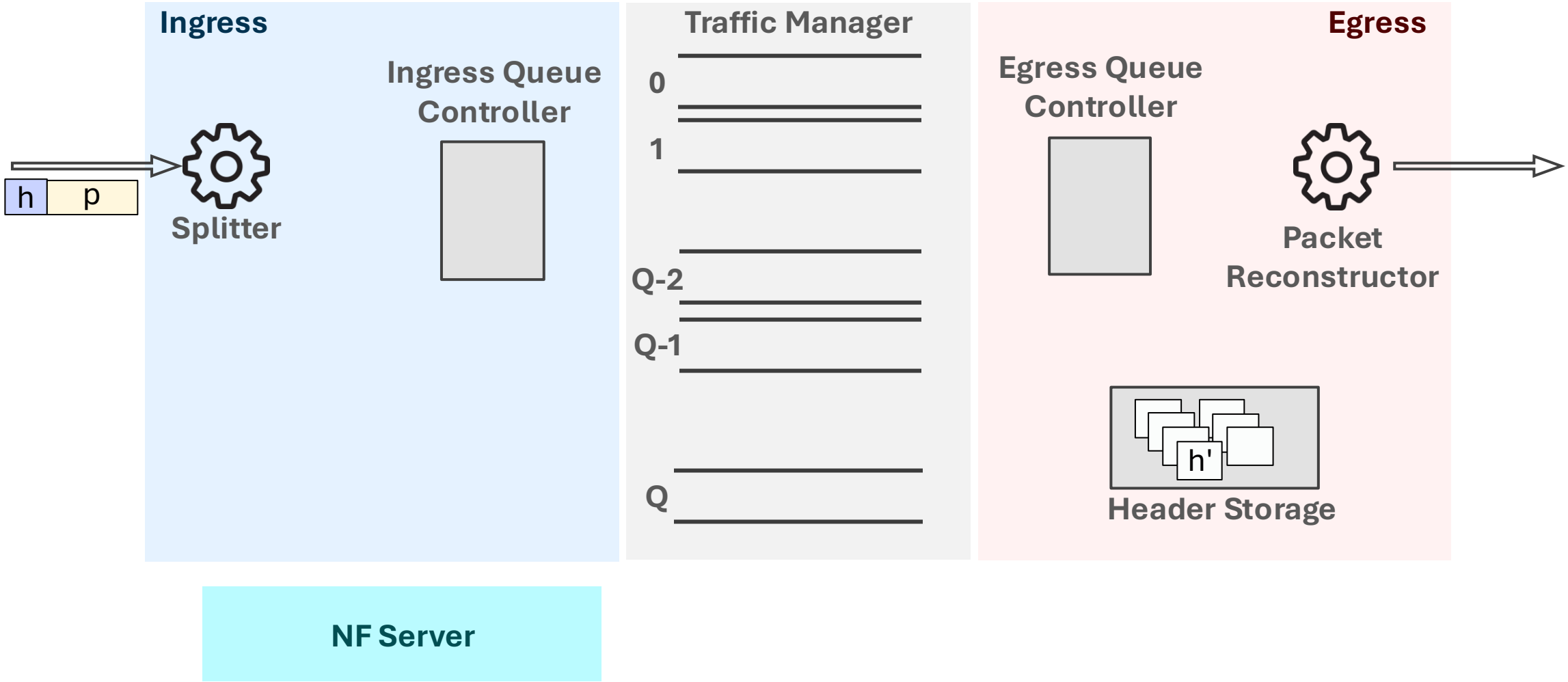
Queue-Mem

Dynamically pause/resume queues holding payloads



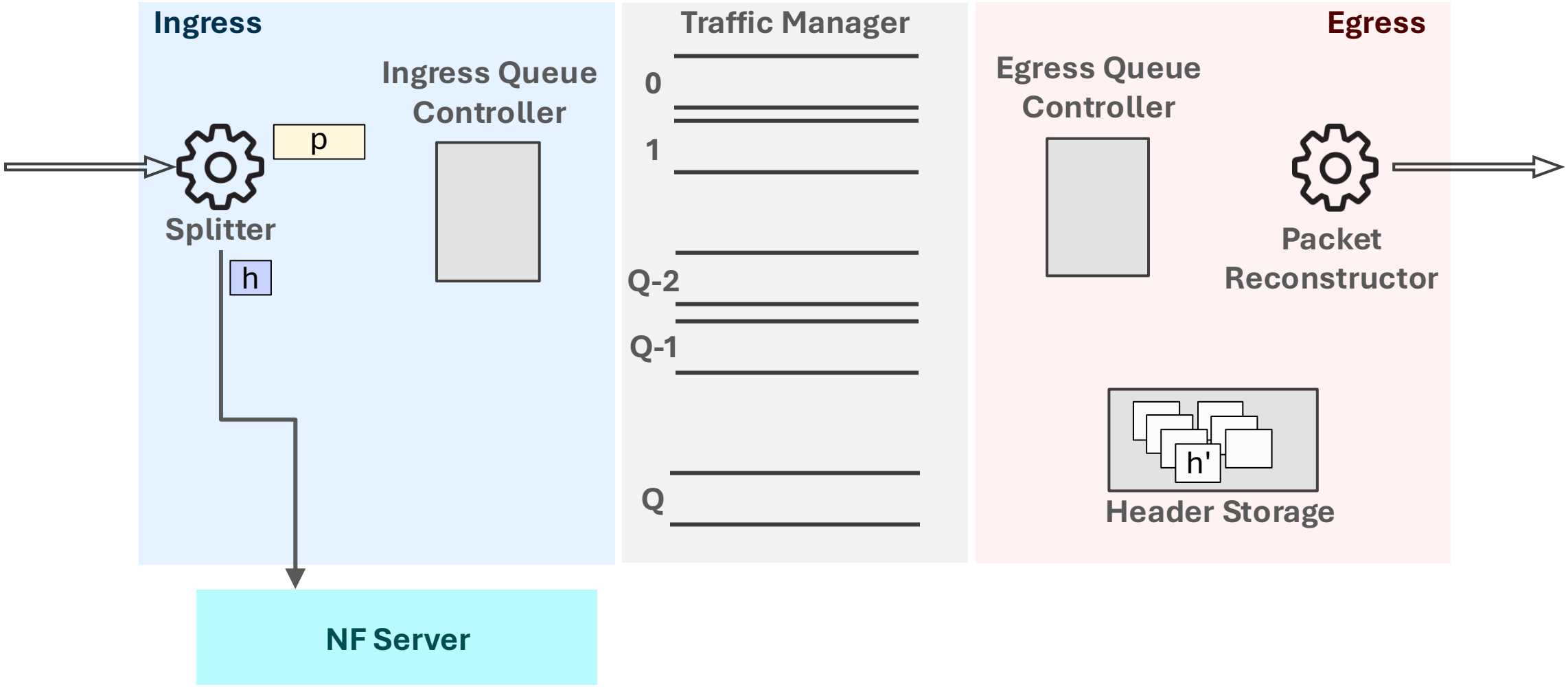
Queue-Mem

Dynamically pause/resume queues holding payloads



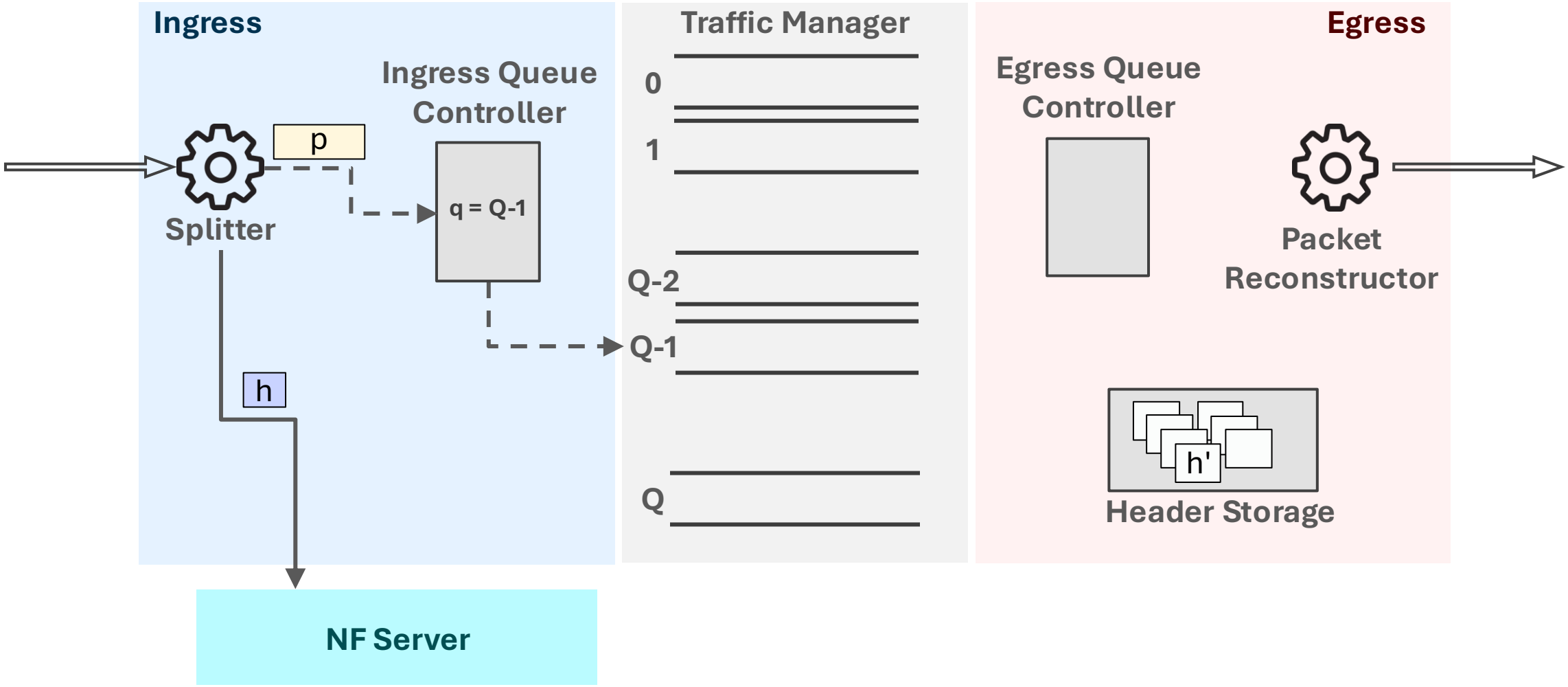
Queue-Mem

Dynamically pause/resume queues holding payloads



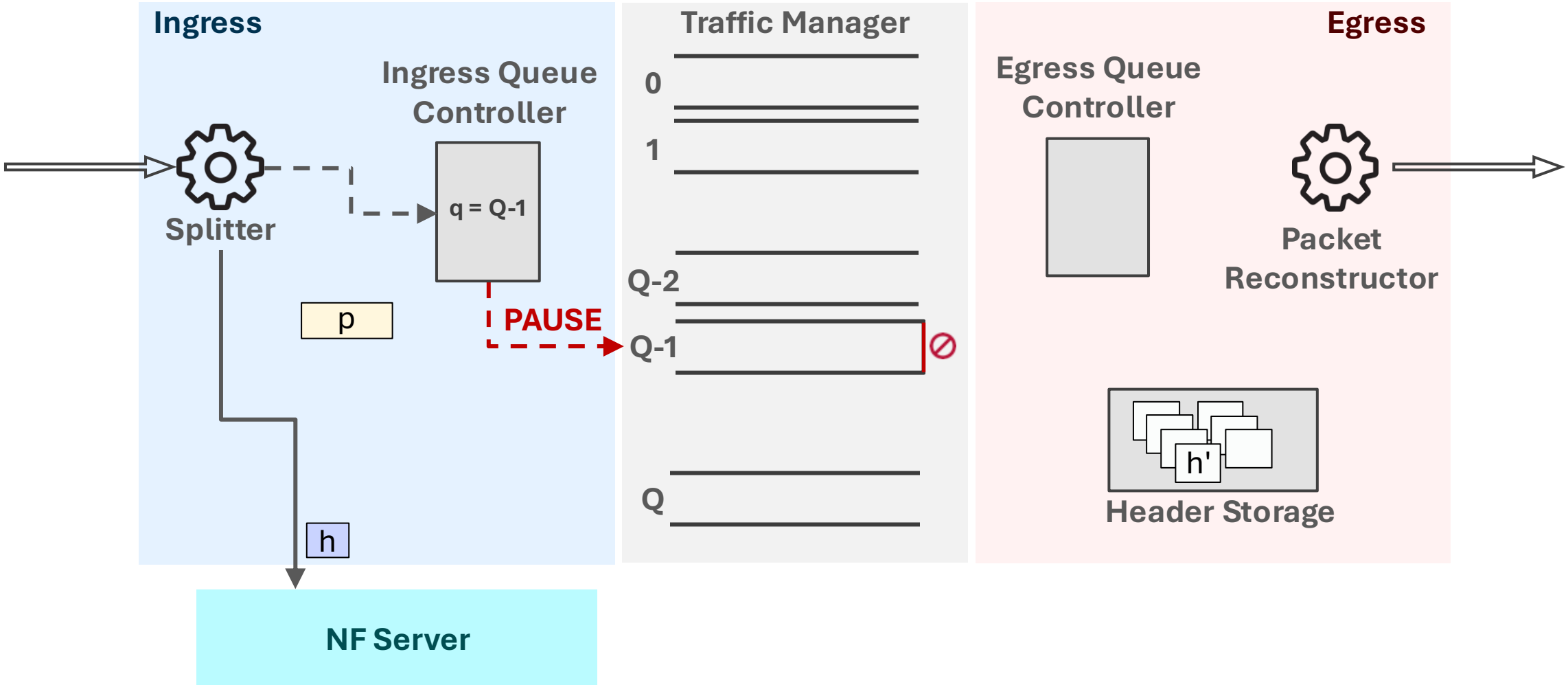
Queue-Mem

Dynamically pause/resume queues holding payloads



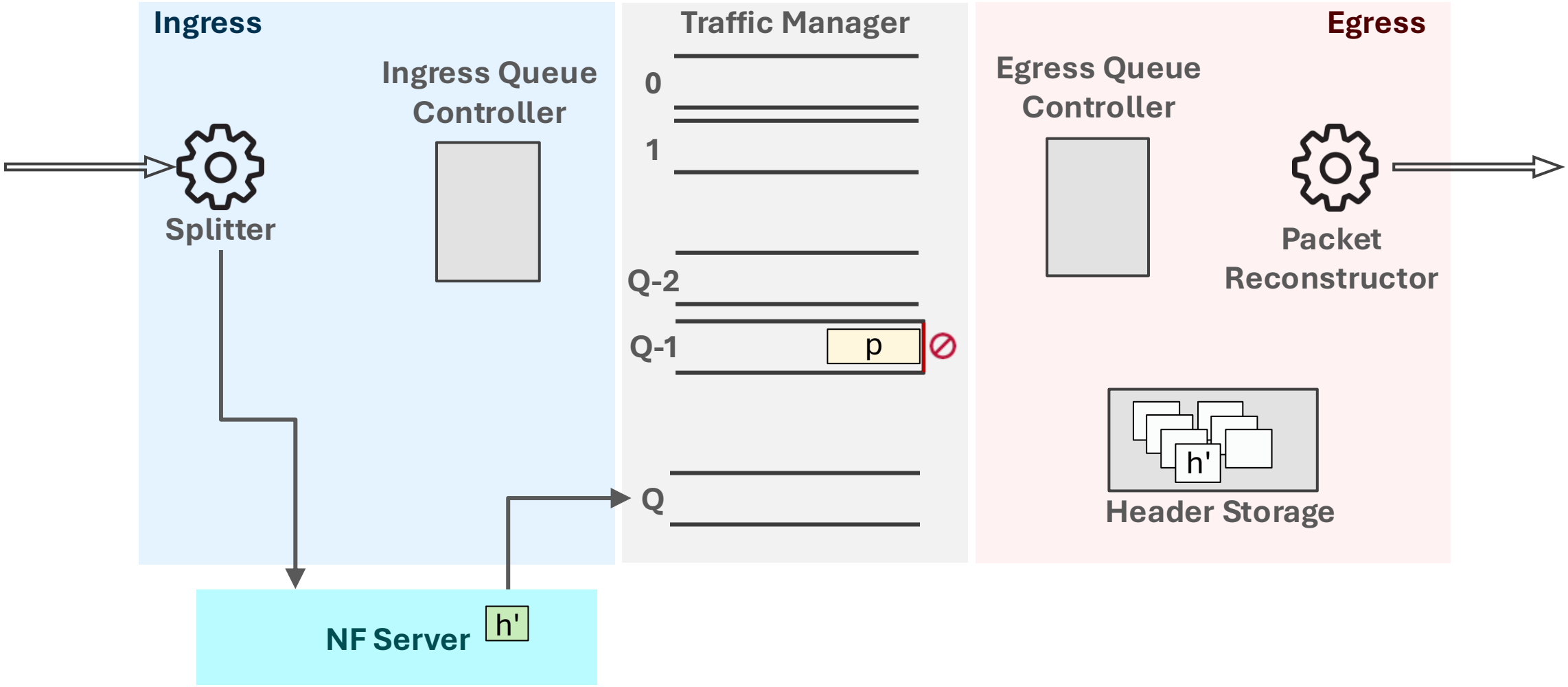
Queue-Mem

Dynamically pause/resume queues holding payloads



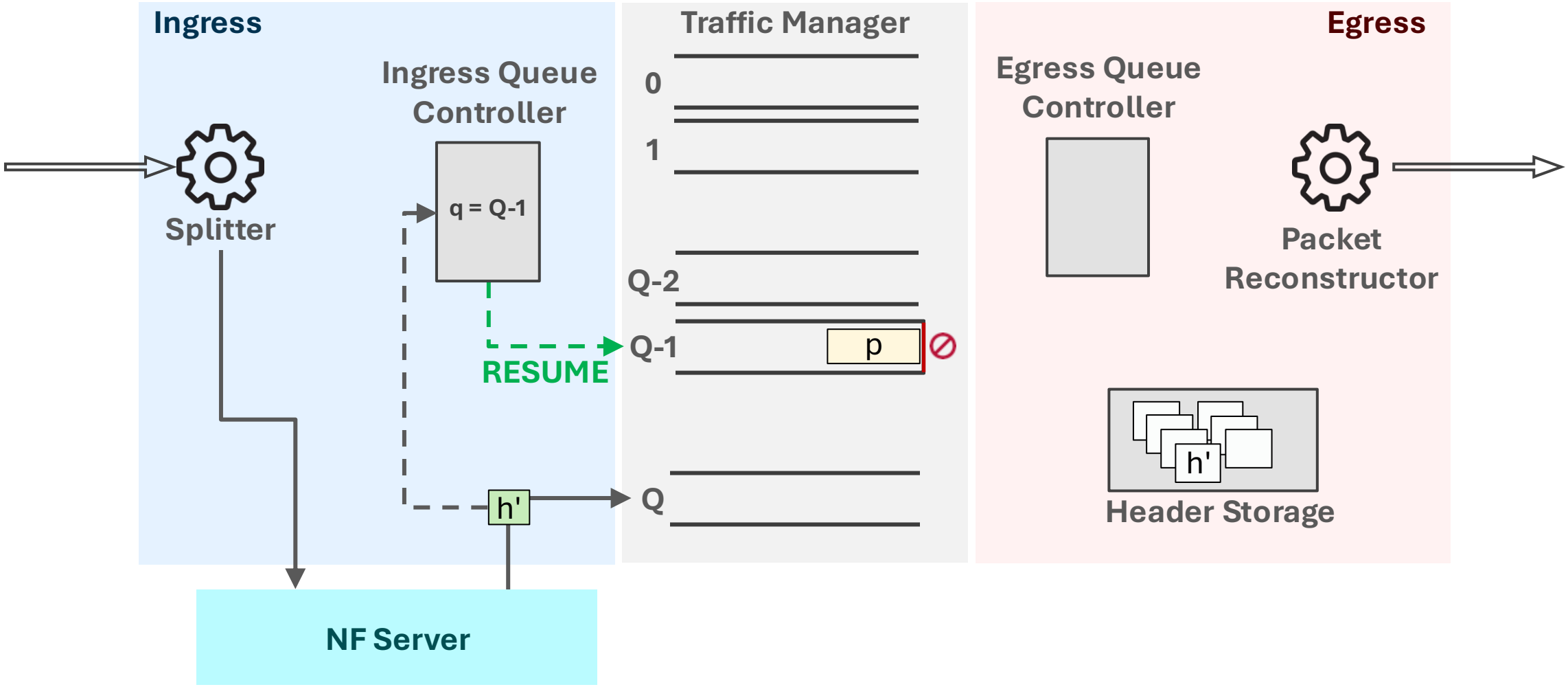
Queue-Mem

Dynamically pause/resume queues holding payloads



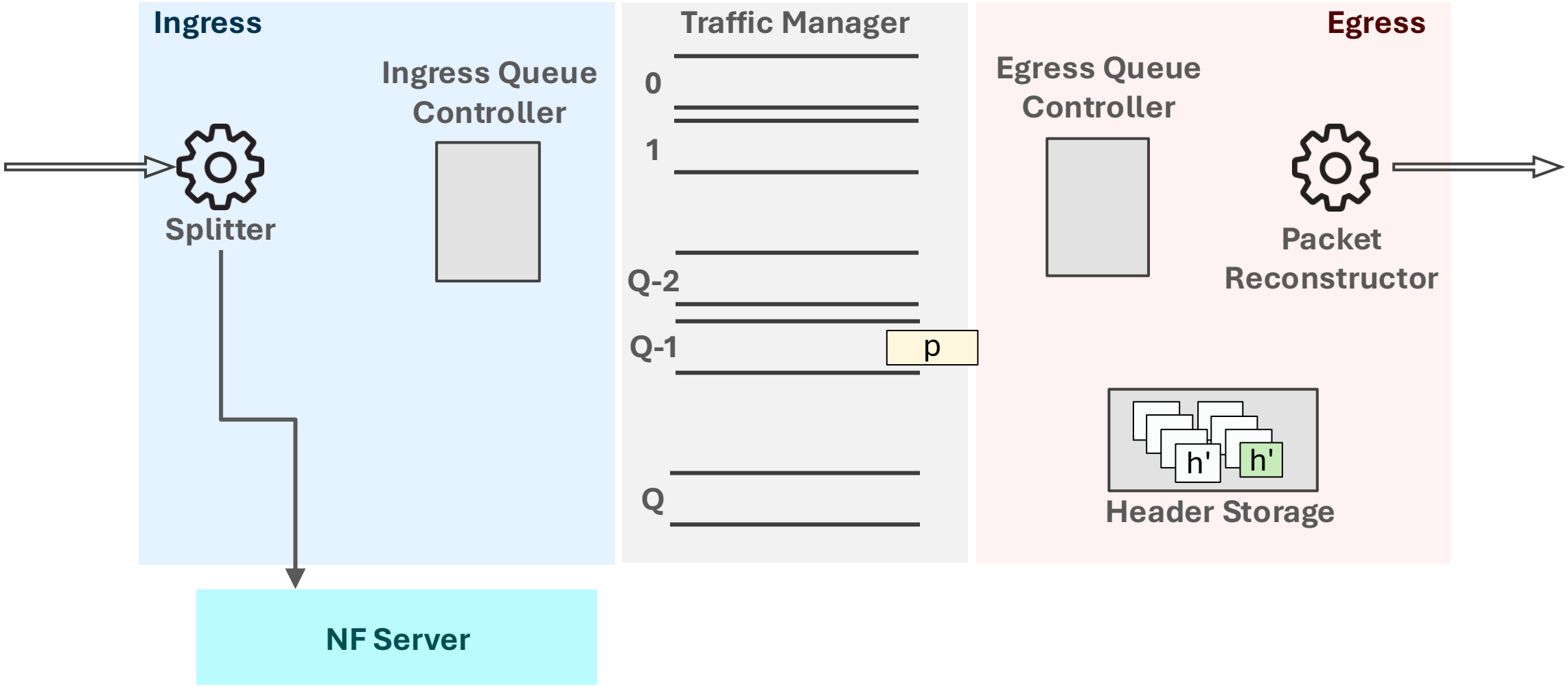
Queue-Mem

Dynamically pause/resume queues holding payloads



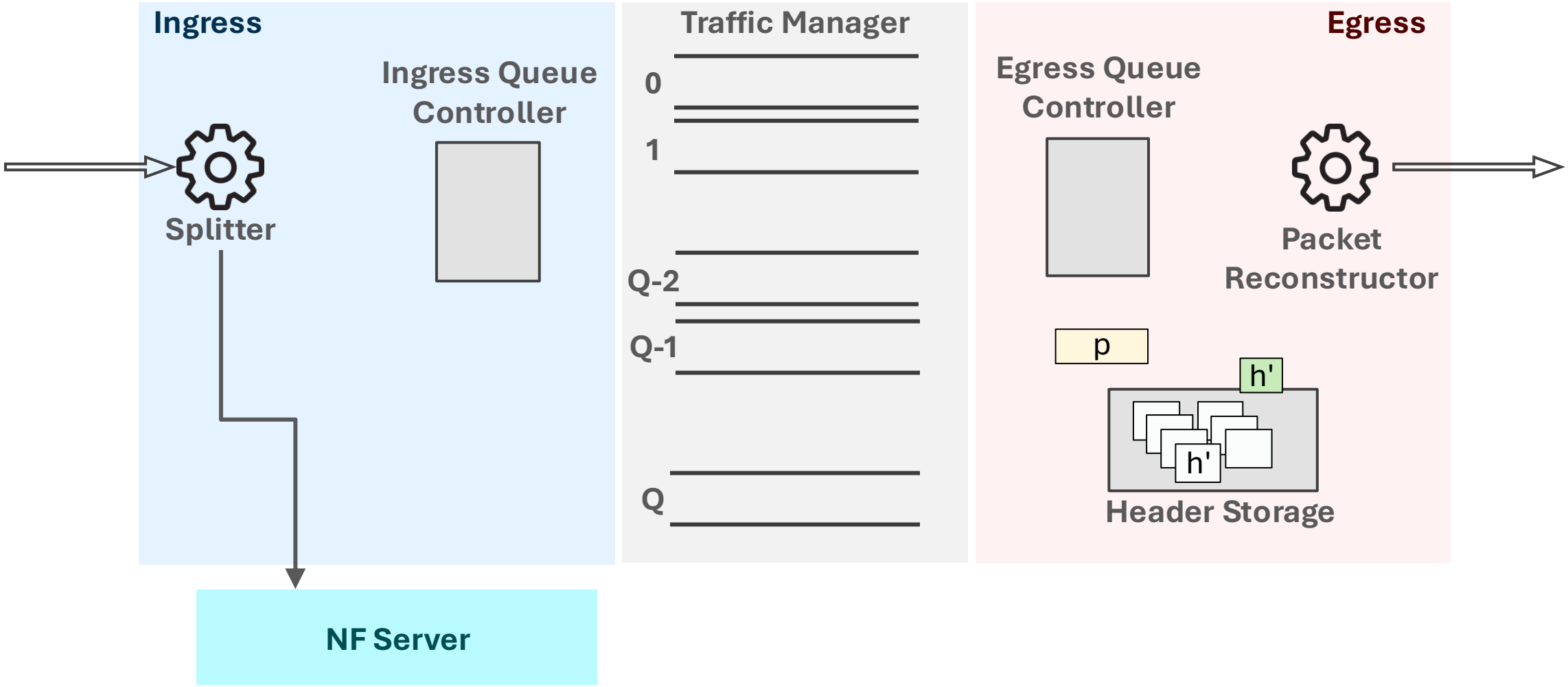
Queue-Mem

Dynamically pause/resume queues holding payloads



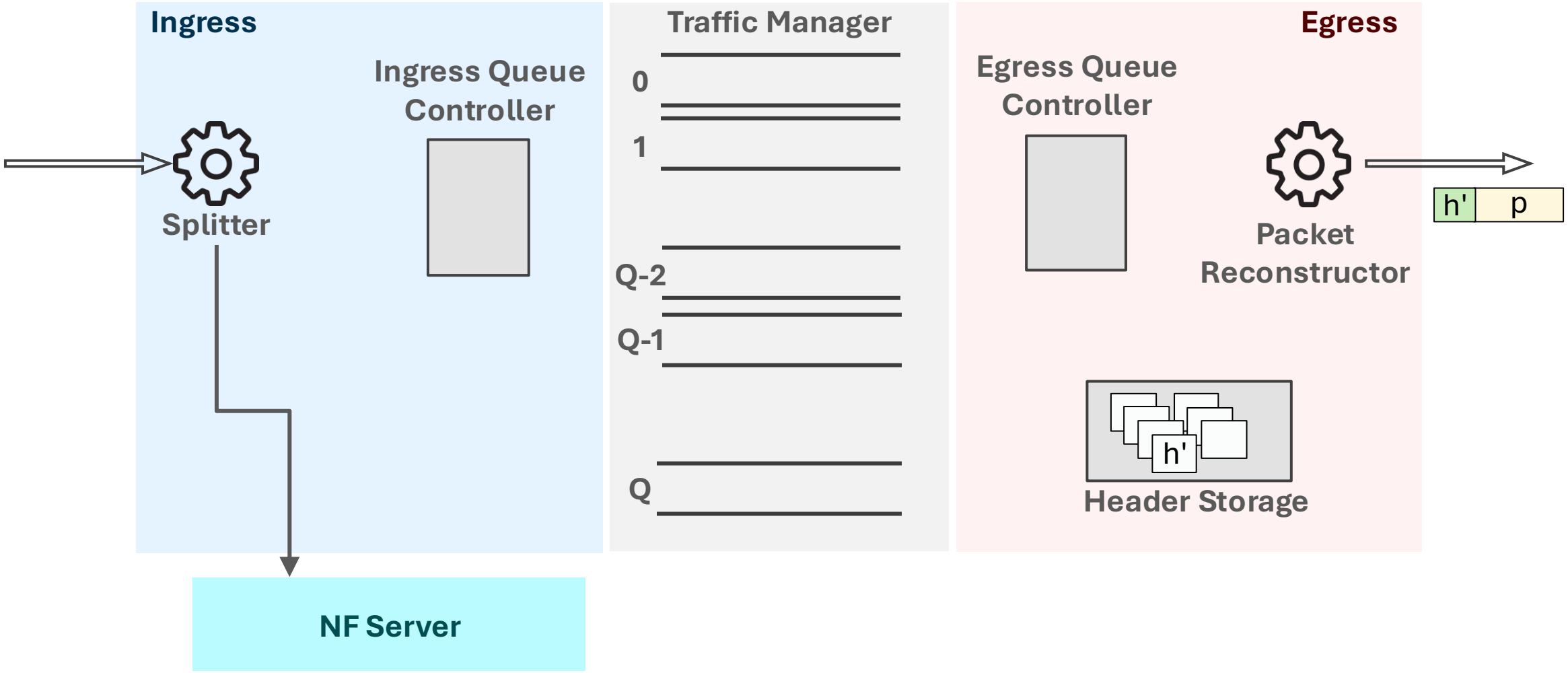
Queue-Mem

Dynamically pause/resume queues holding payloads



Queue-Mem

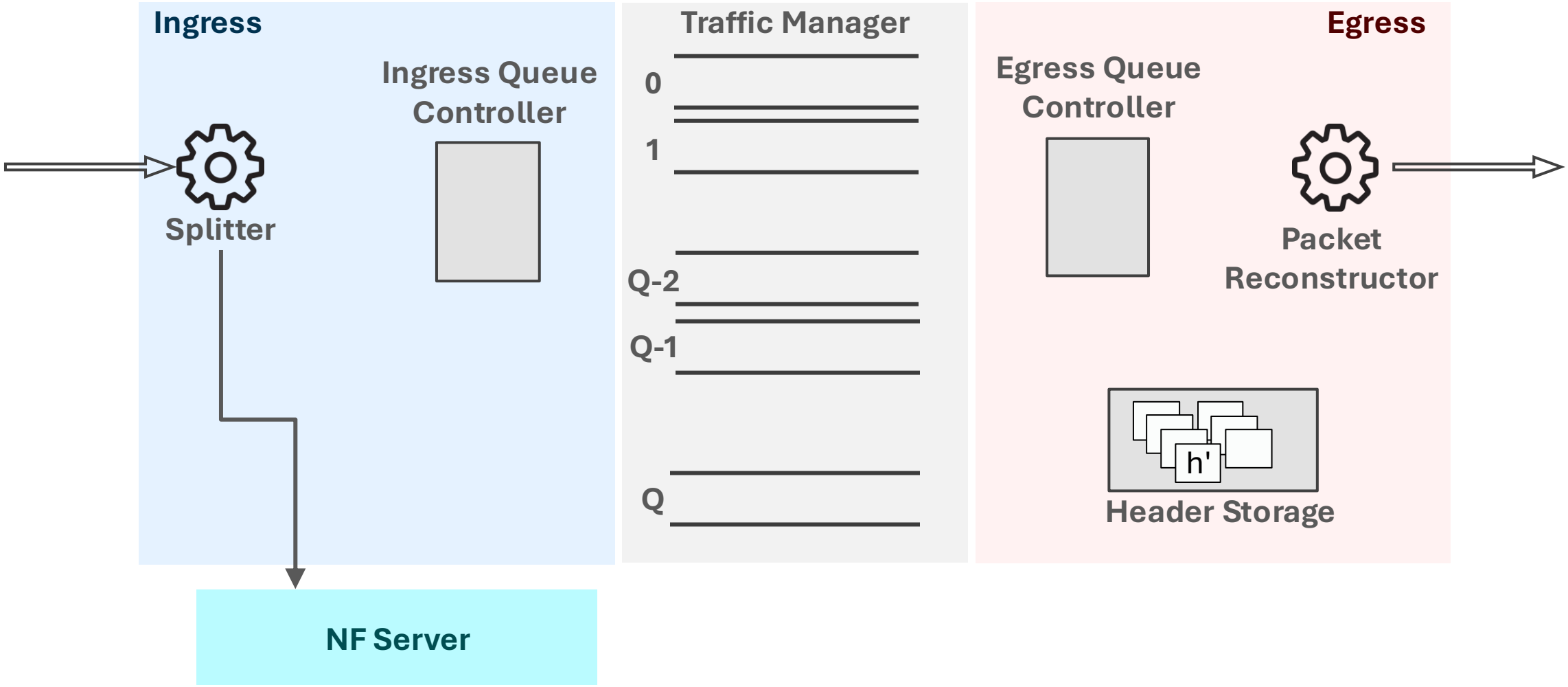
Dynamically pause/resume queues holding payloads



Queue-Mem: Challenges

Pause/Resume cannot dequeue a **single packet**

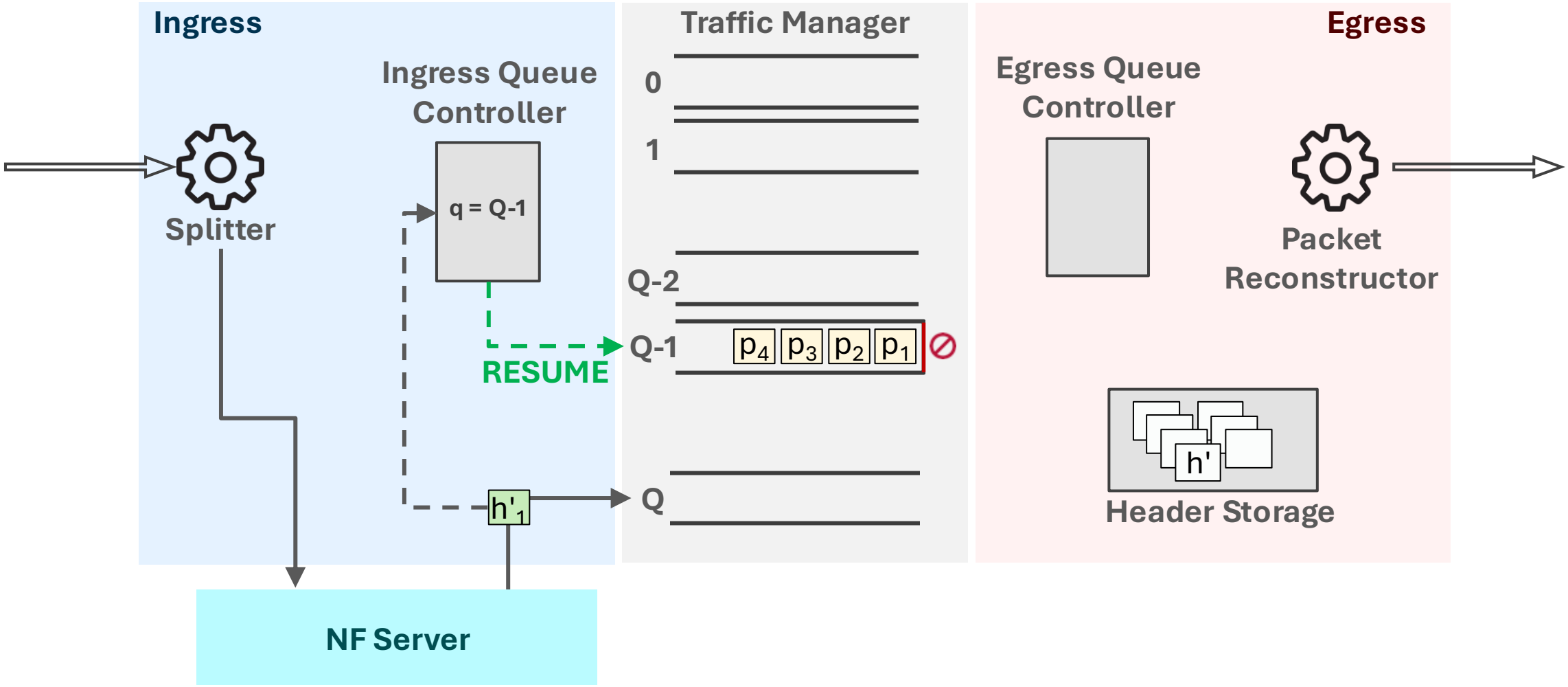
Pause/Resume effect is **delayed**



Queue-Mem: Challenges

Pause/Resume cannot dequeue a **single packet**

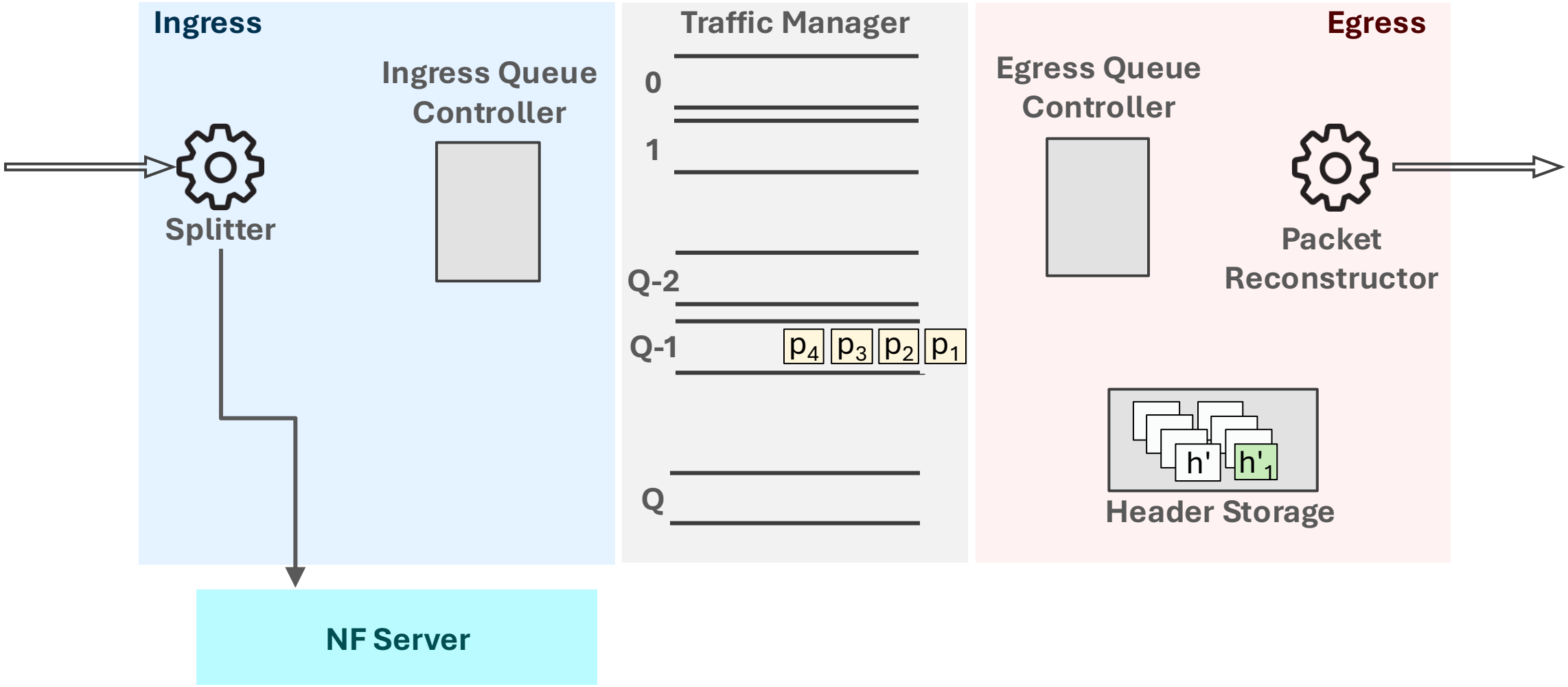
Pause/Resume effect is **delayed**



Queue-Mem: Challenges

Pause/Resume cannot dequeue a **single packet**

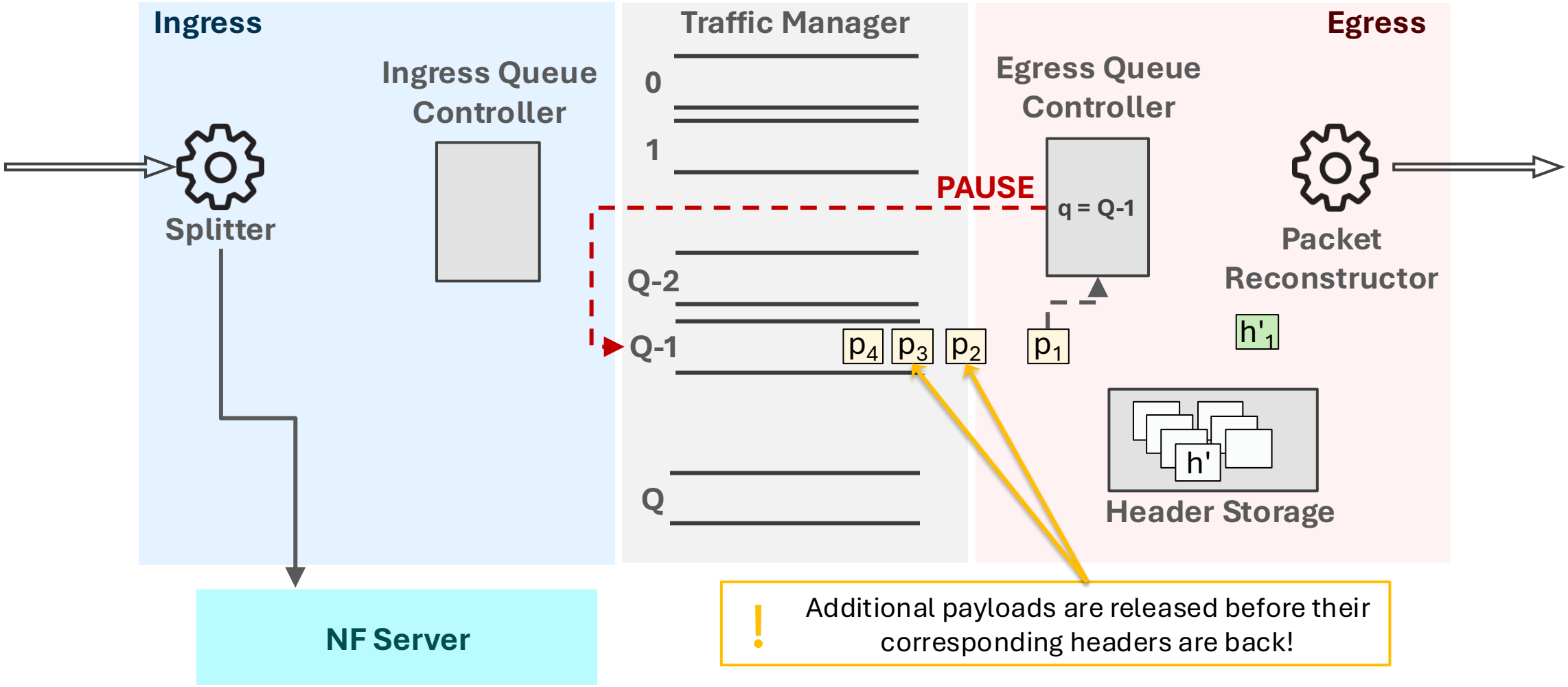
Pause/Resume effect is **delayed**



Queue-Mem: Challenges

Pause/Resume cannot dequeue a **single packet**

Pause/Resume effect is **delayed**

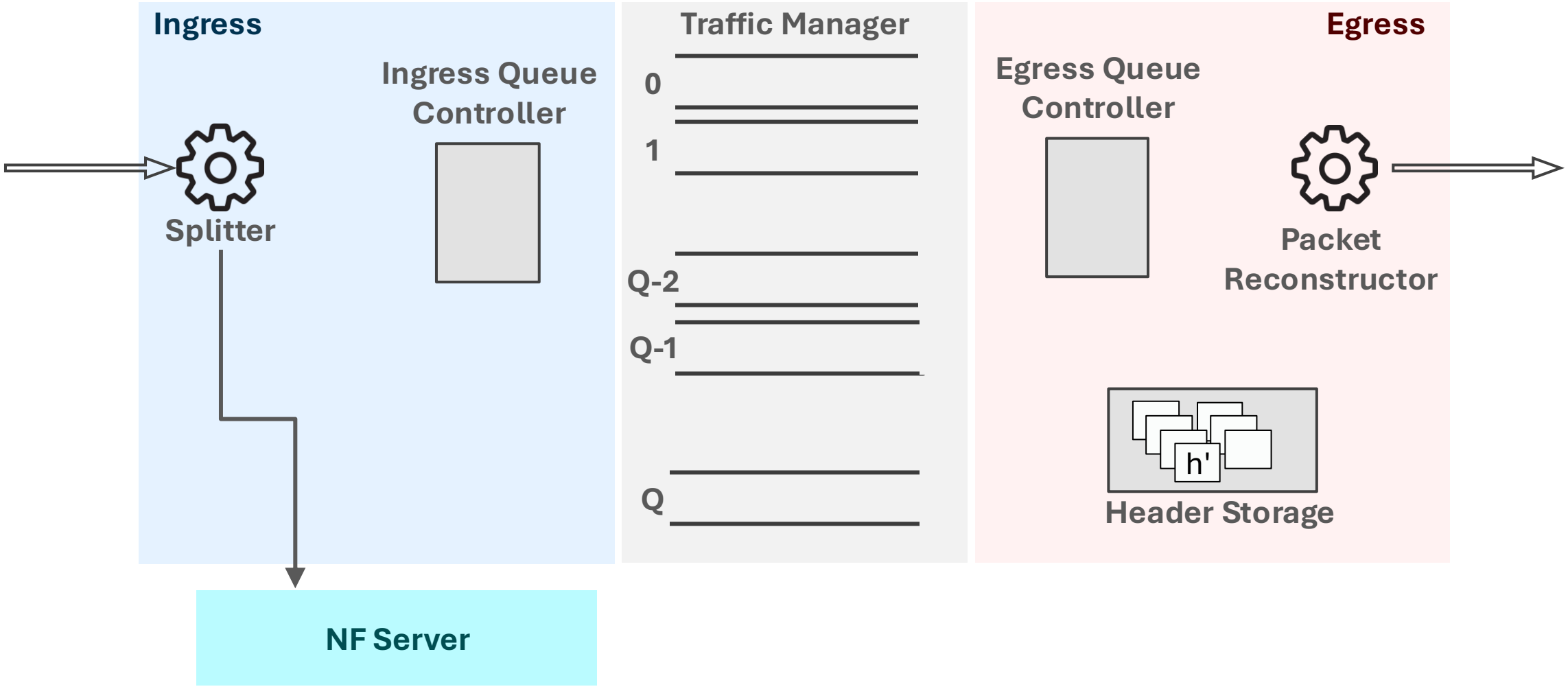


Queue-Mem: Challenges

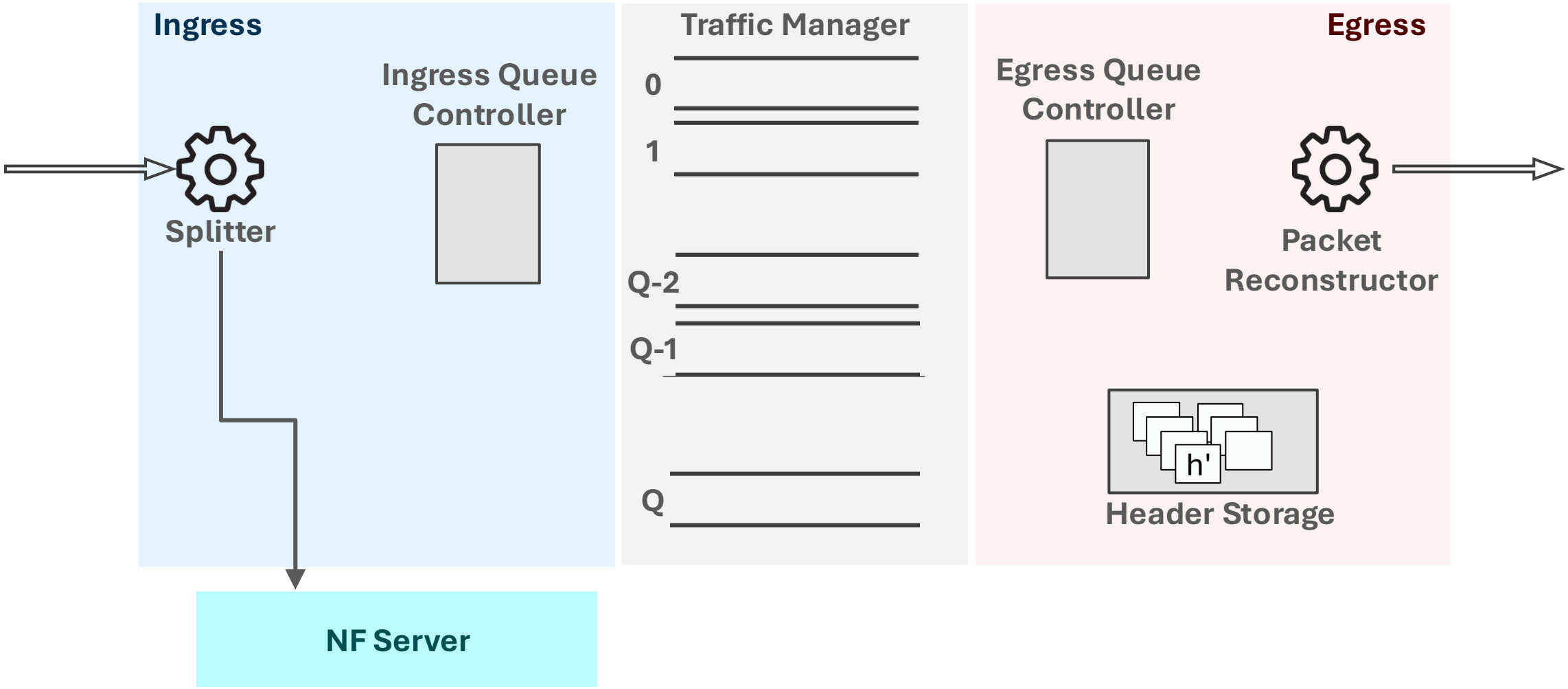
Pause/Resume cannot dequeue a **single packet**

Pause/Resume effect is **delayed**

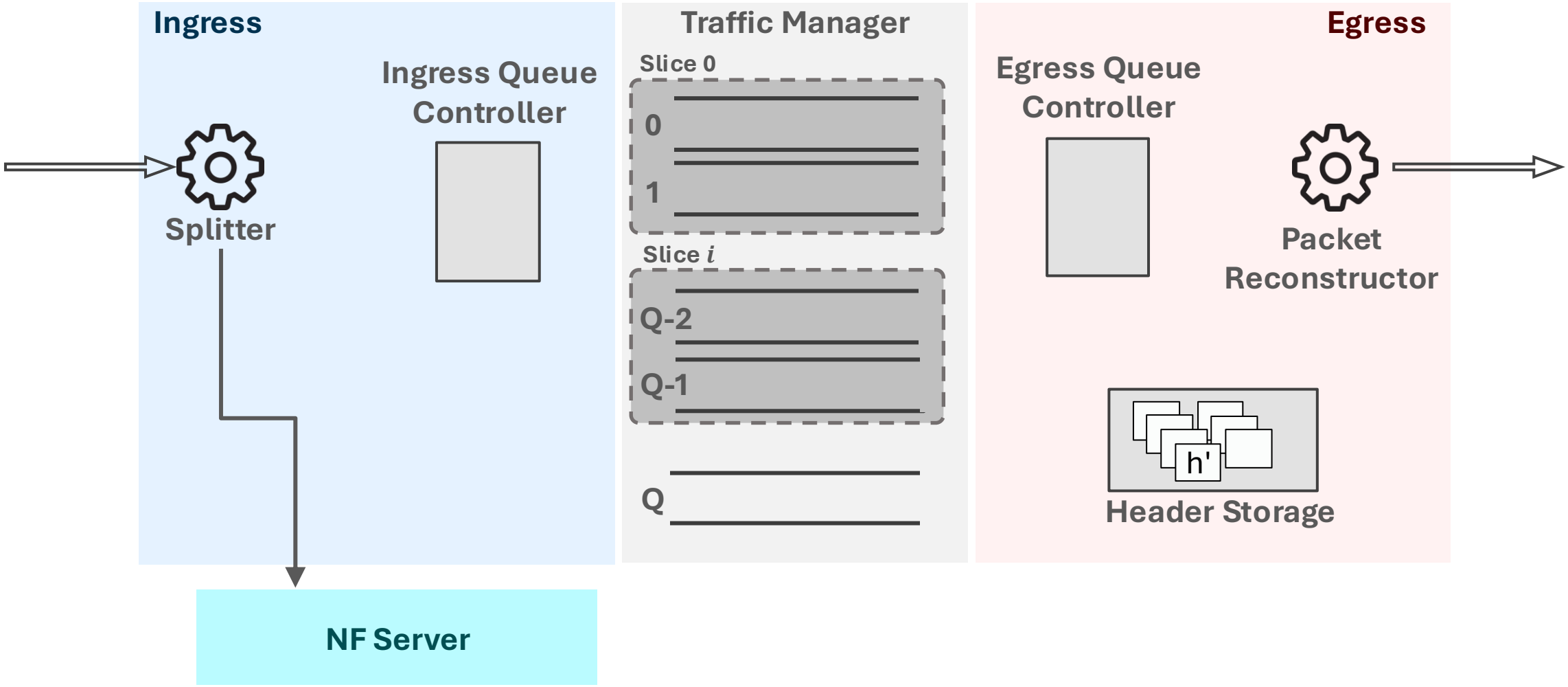
Batch-based queue control



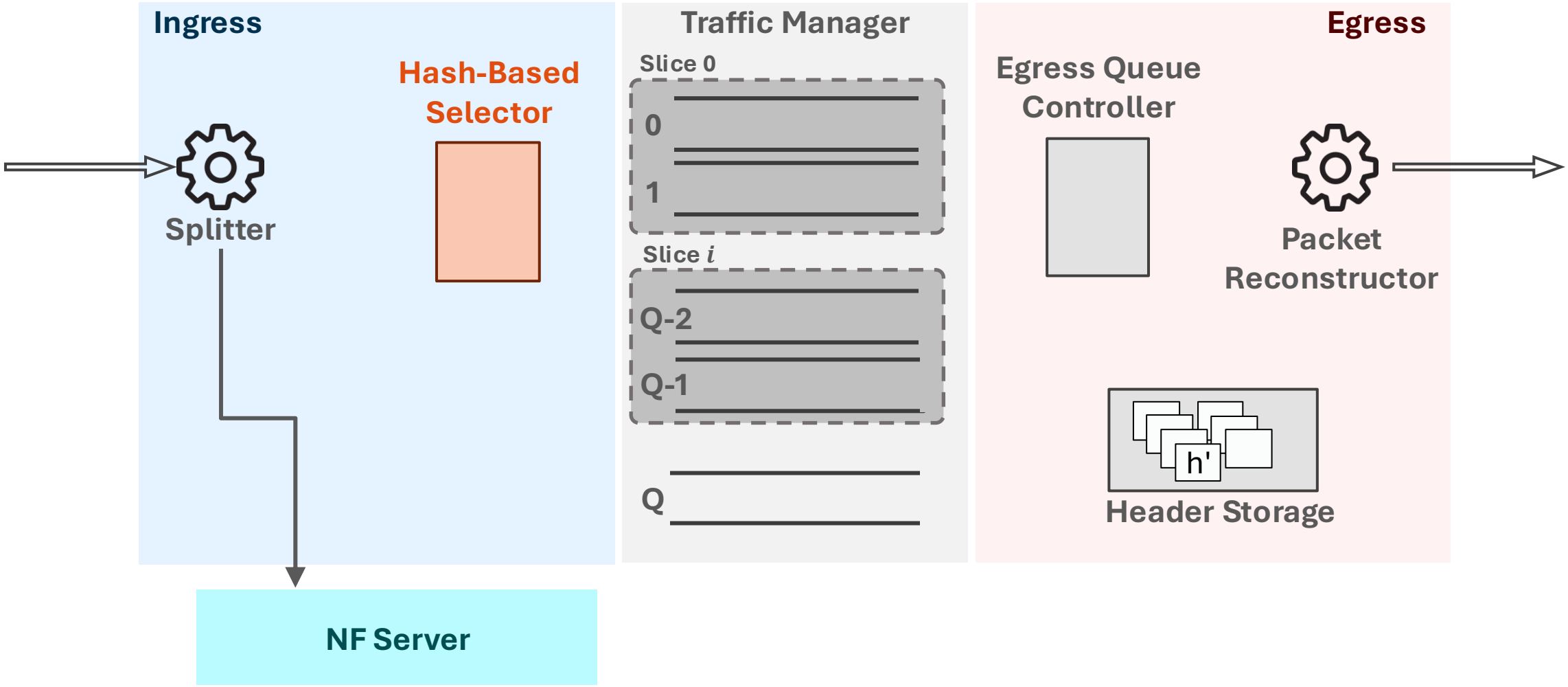
Queue-Mem: **Batch-based** queue control



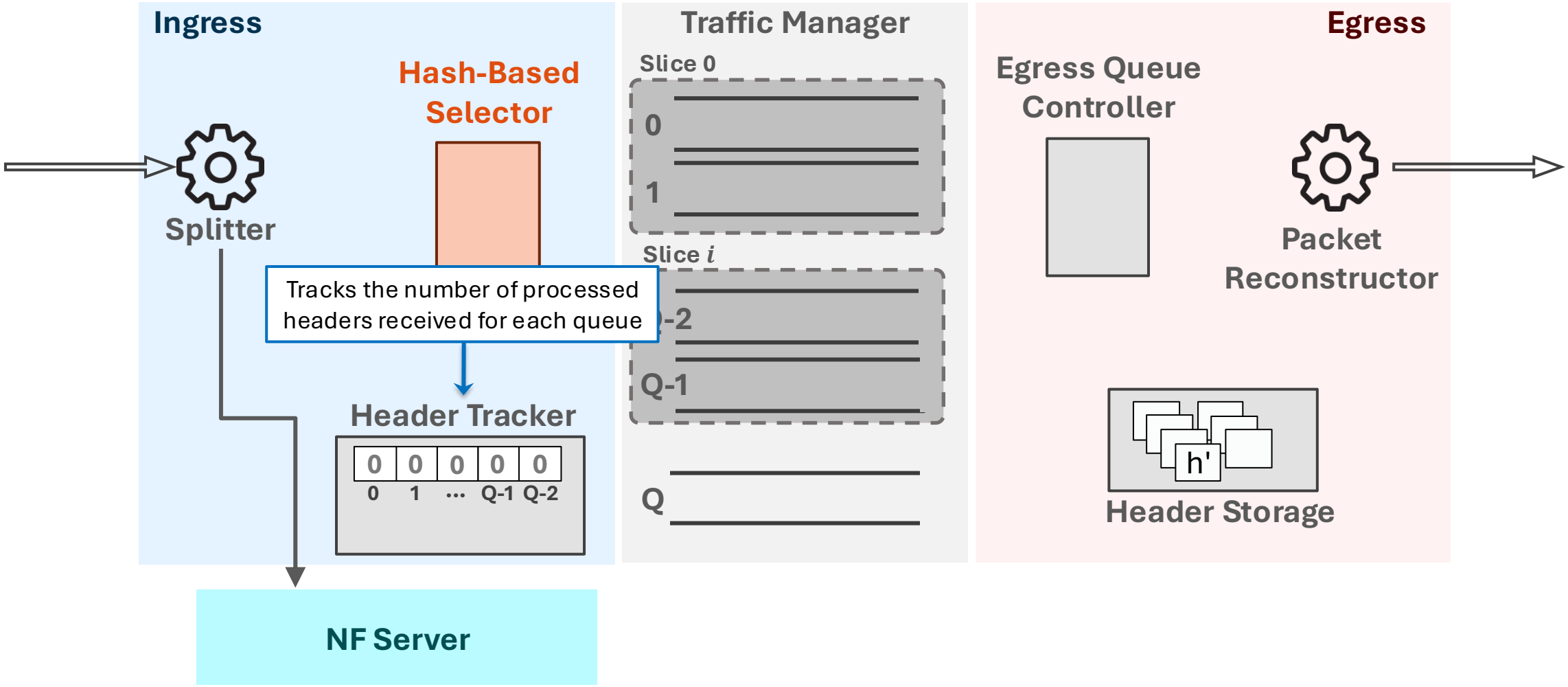
Queue-Mem: **Batch-based** queue control



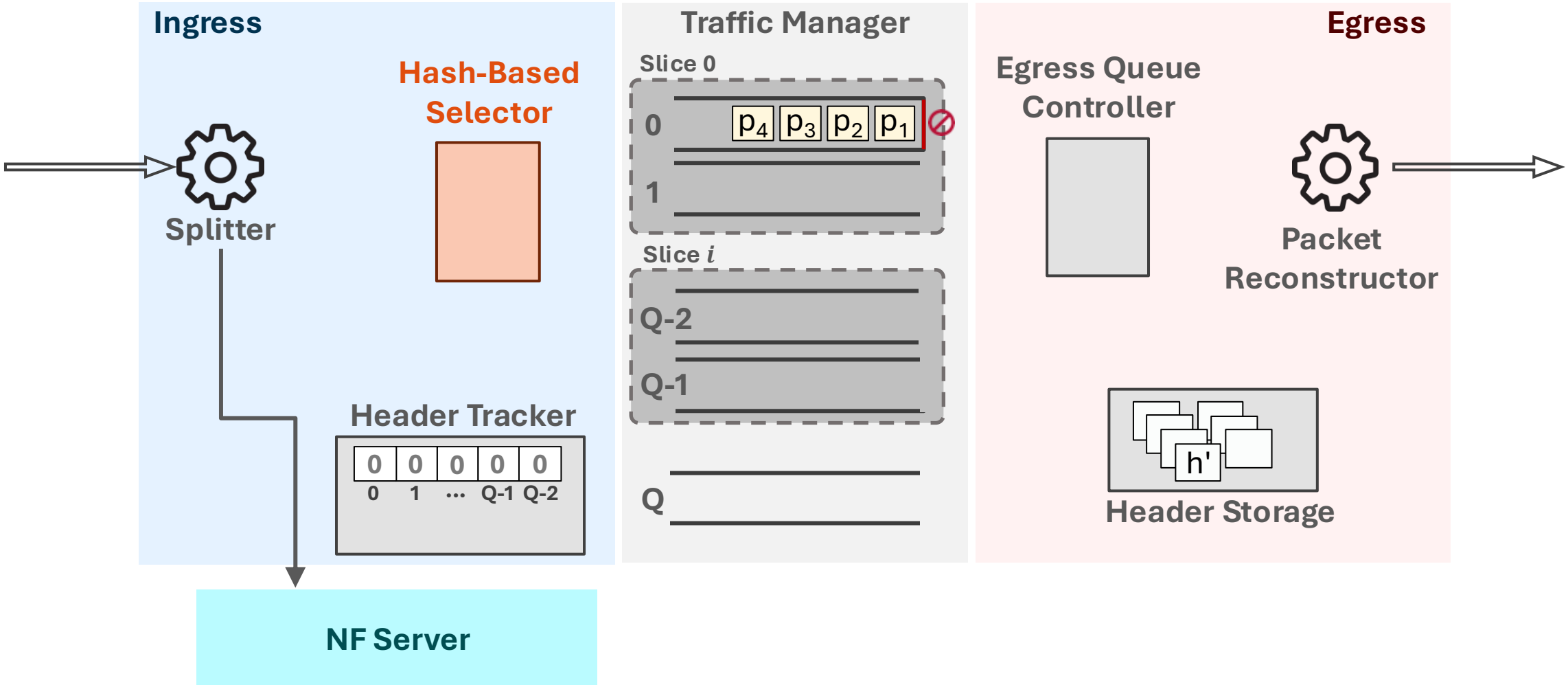
Queue-Mem: Batch-based queue control



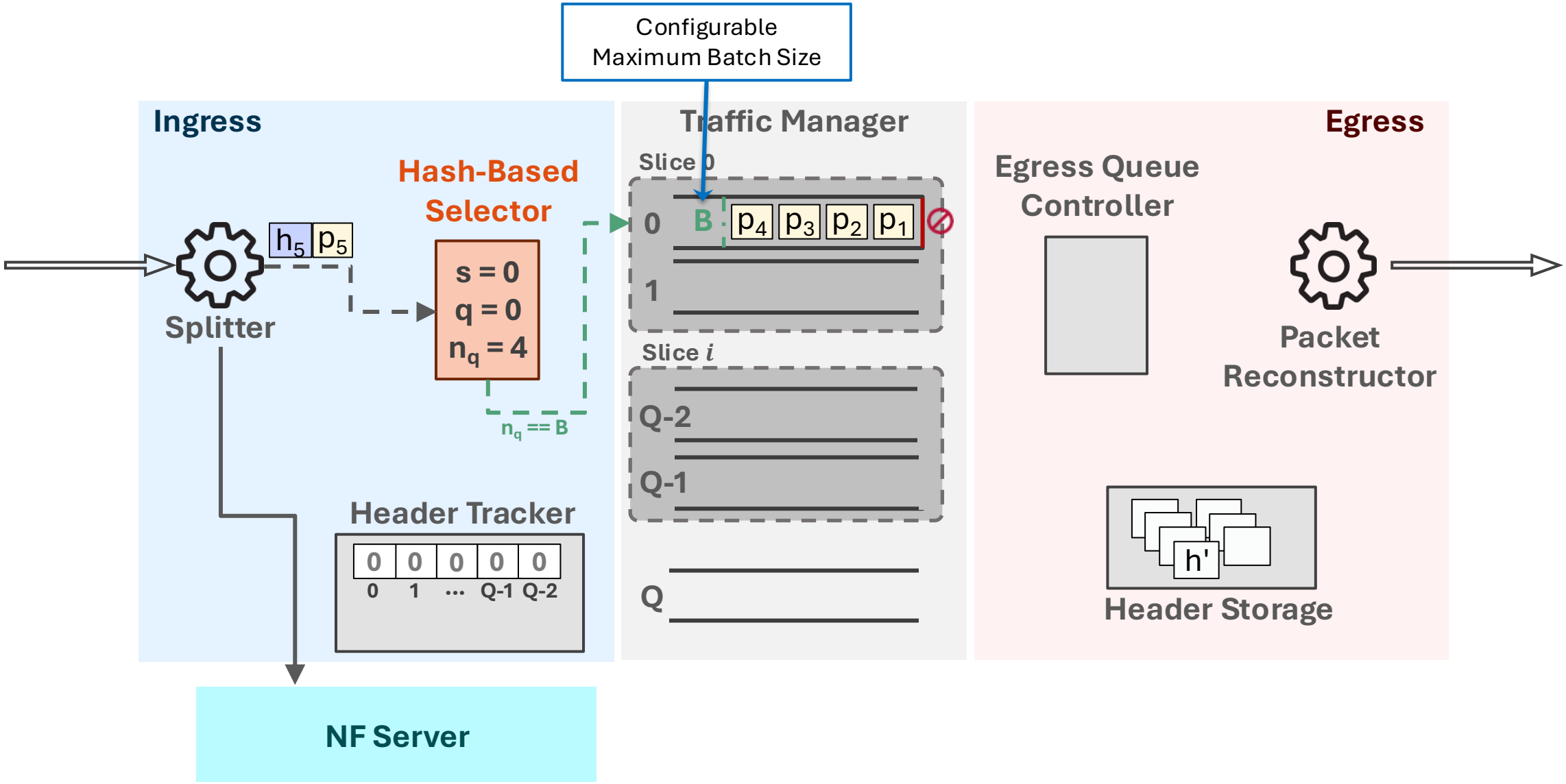
Queue-Mem: Batch-based queue control



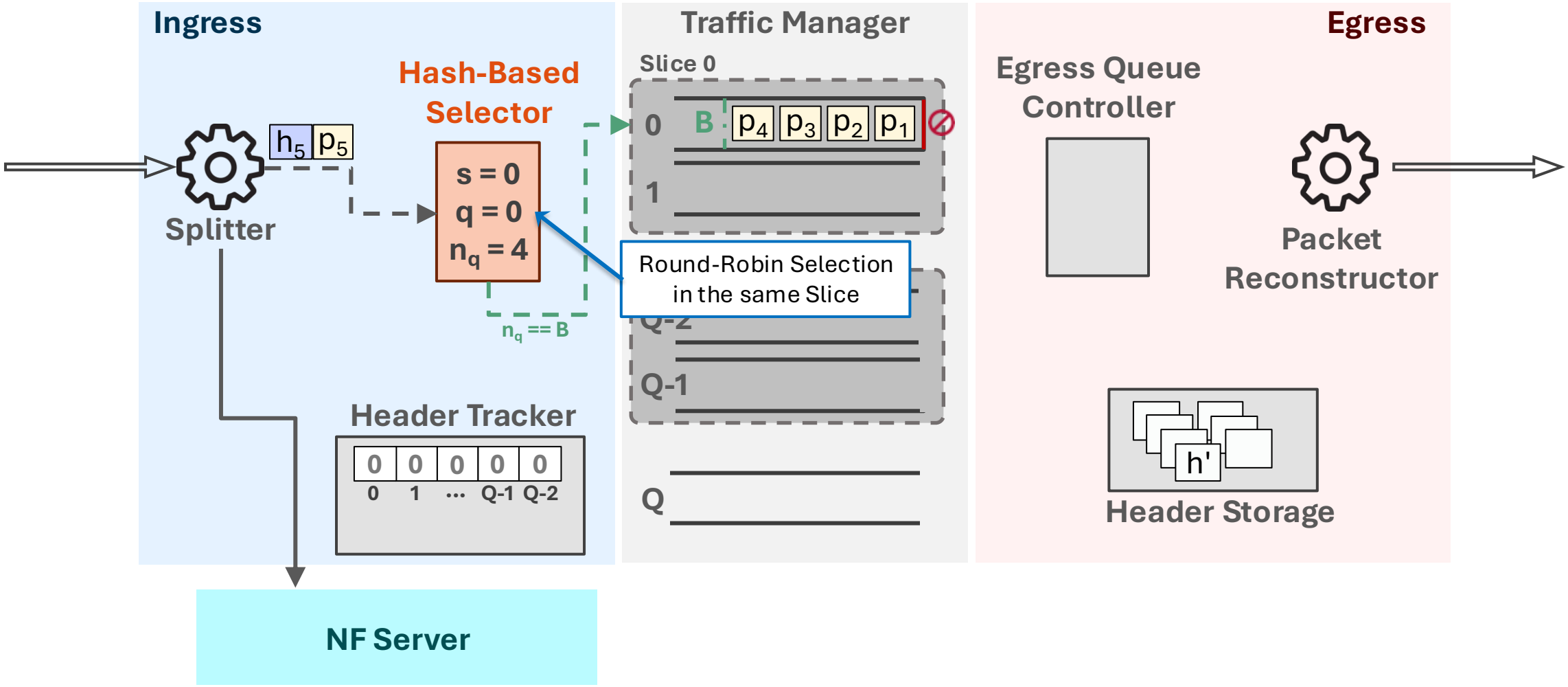
Queue-Mem: Batch Creation



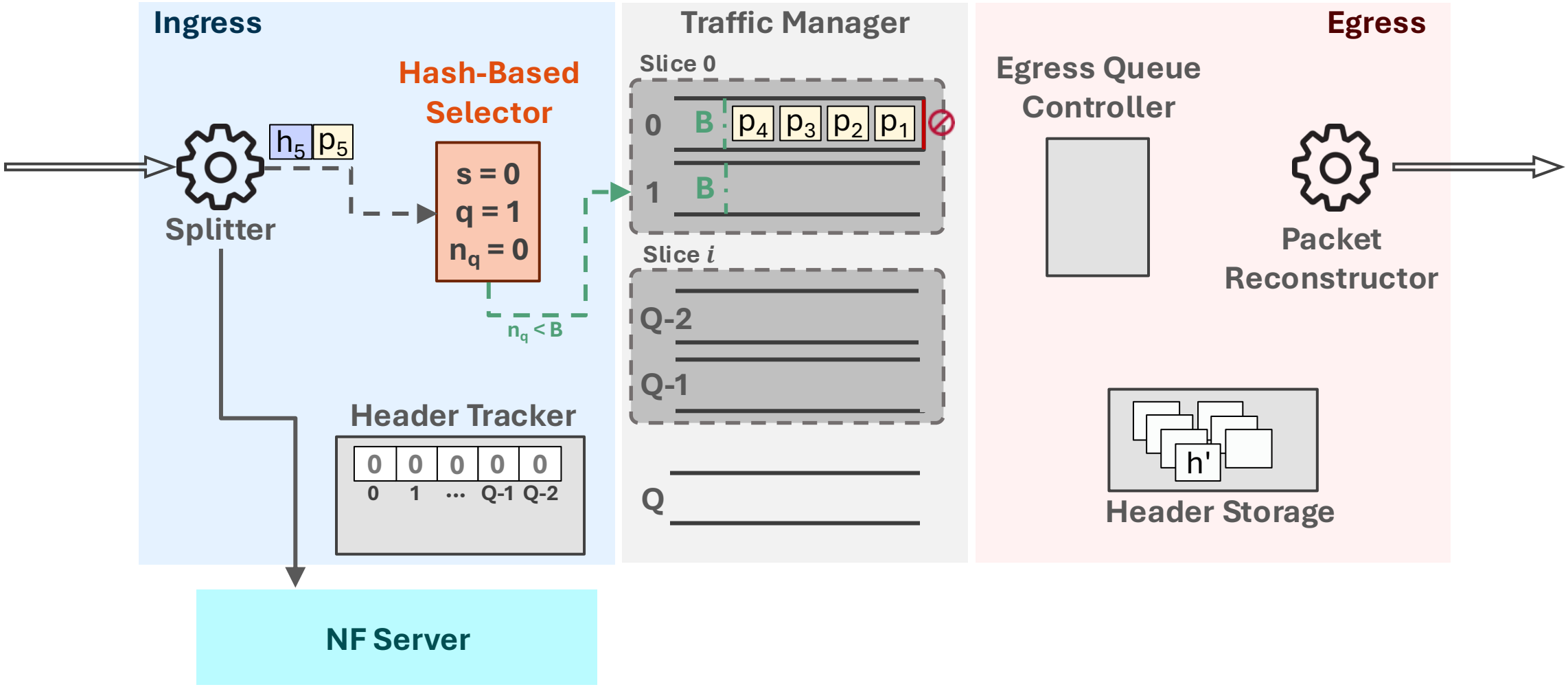
Queue-Mem: Batch Creation



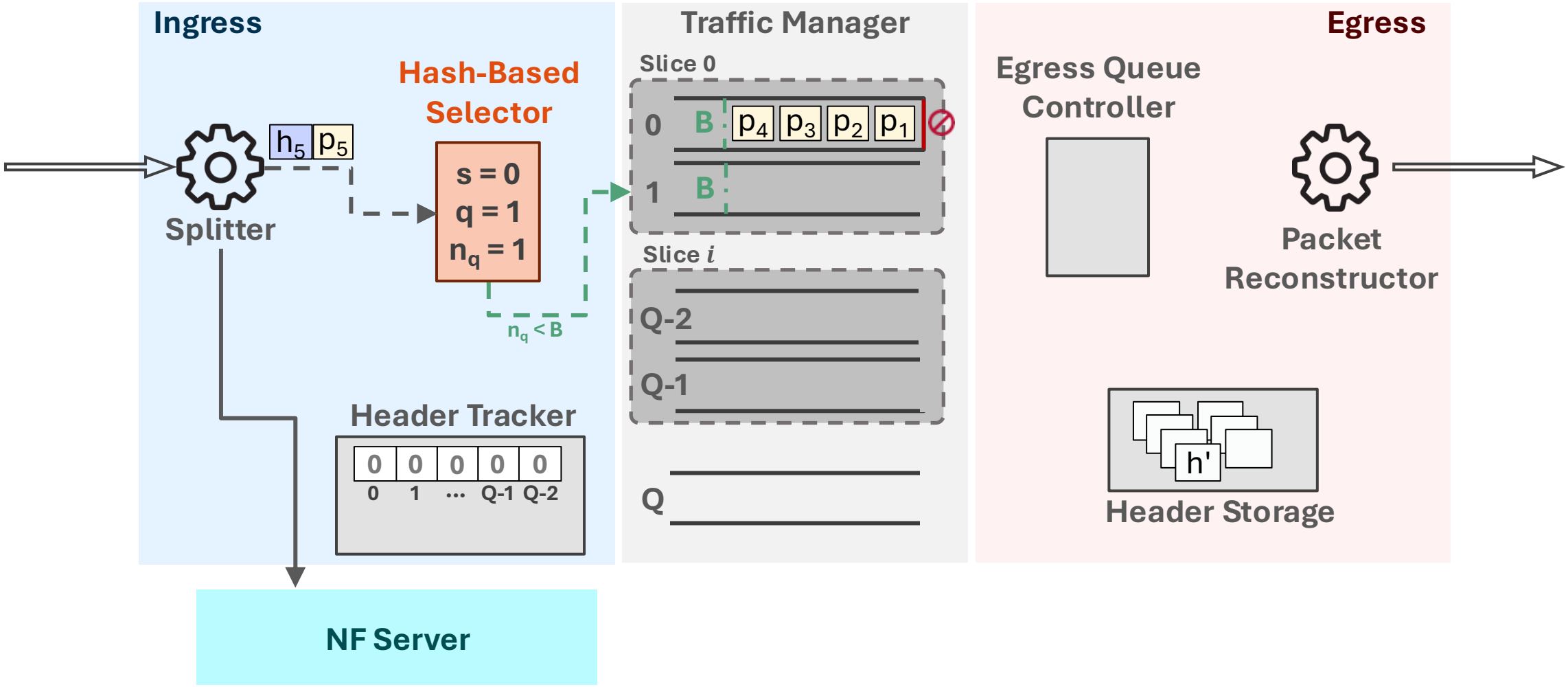
Queue-Mem: Batch Creation



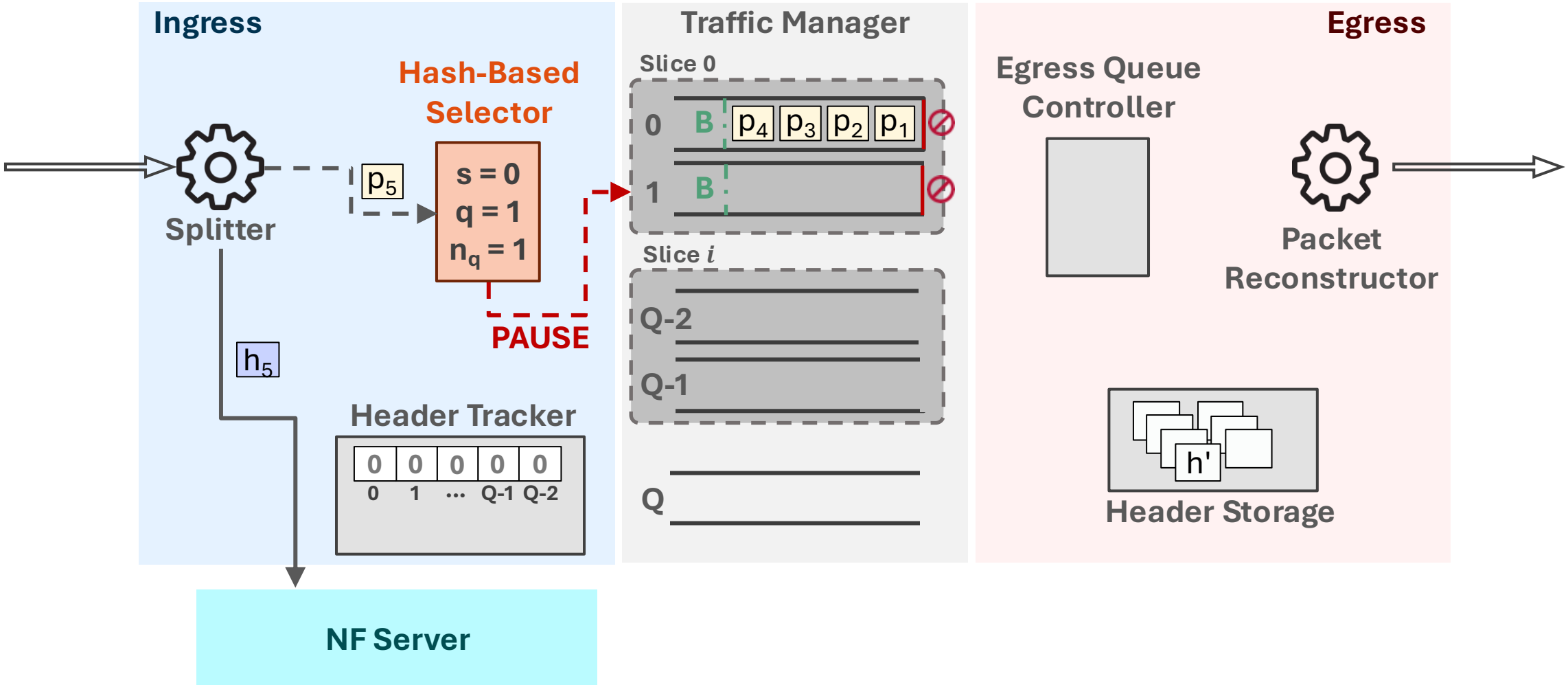
Queue-Mem: Batch Creation



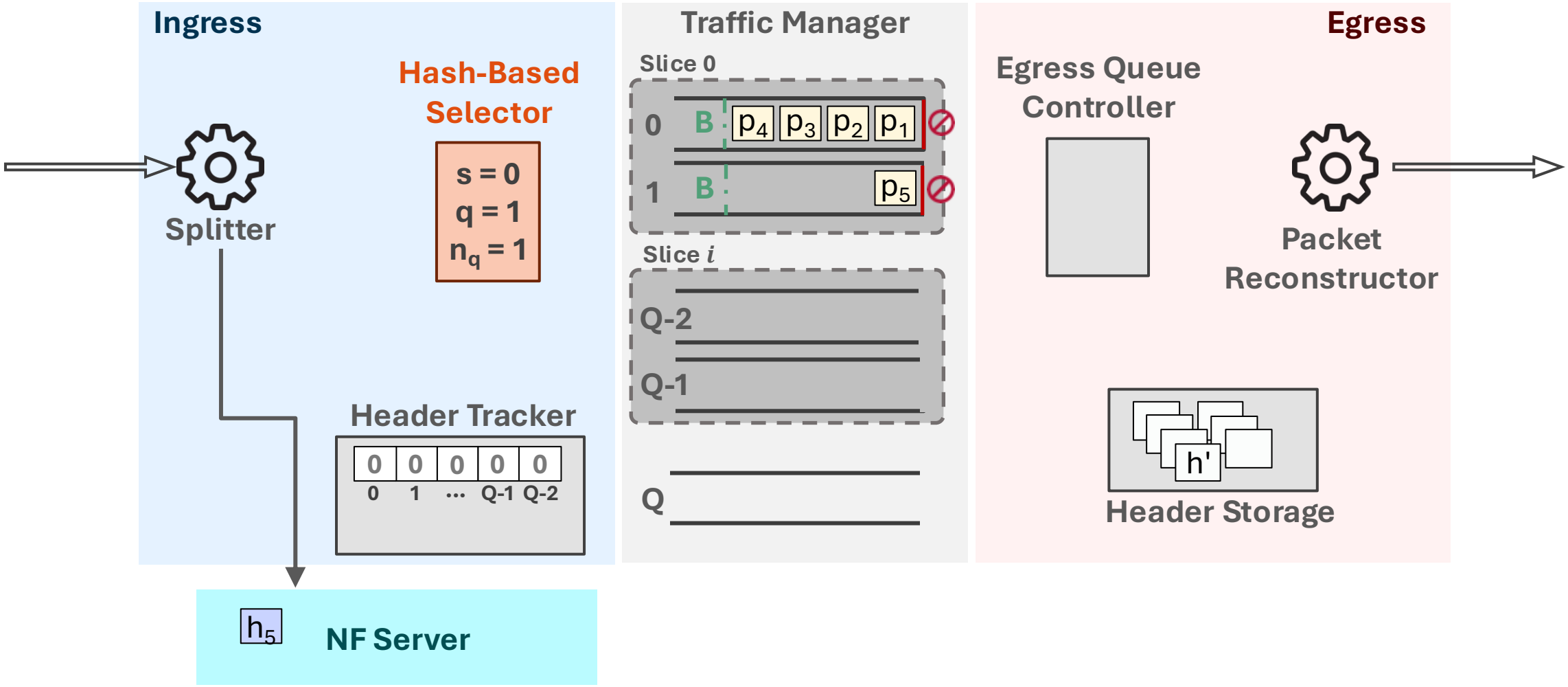
Queue-Mem: Batch Creation



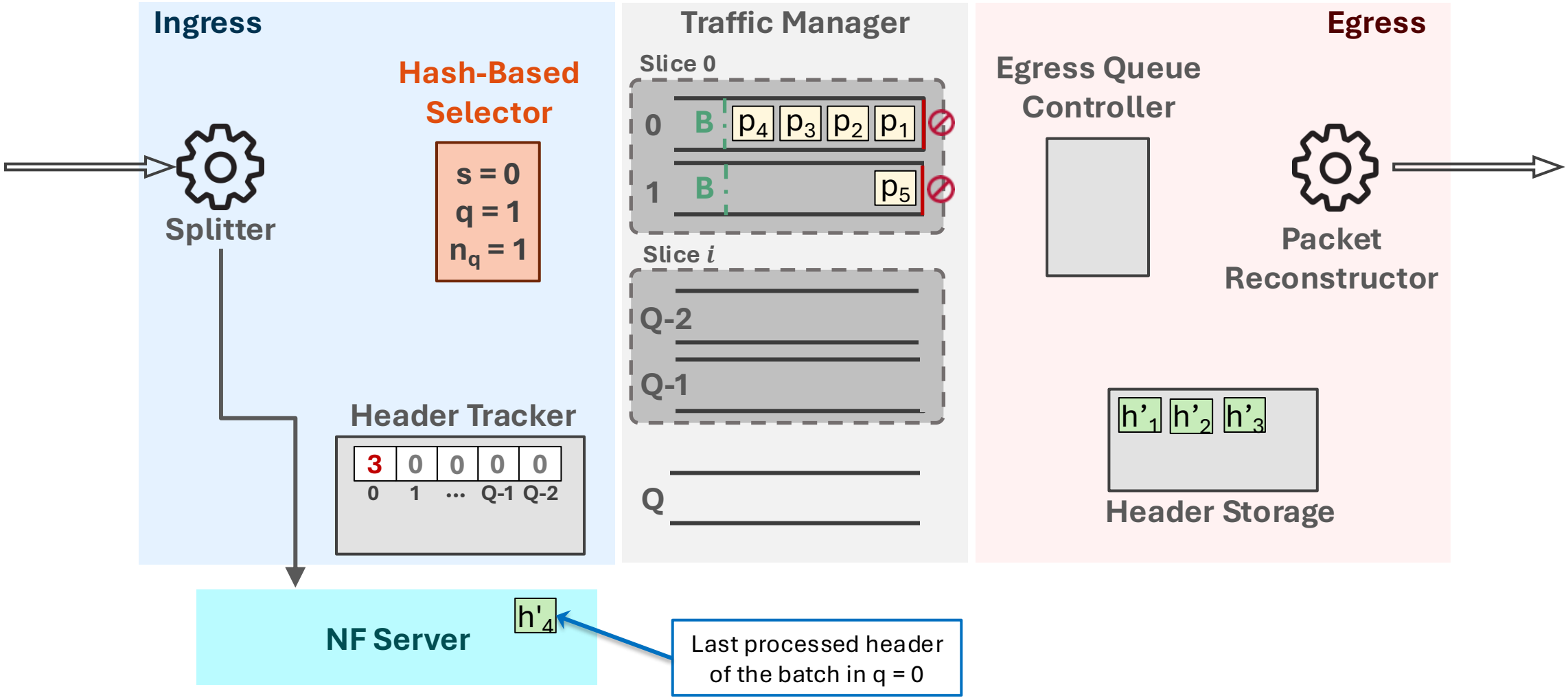
Queue-Mem: Batch Creation



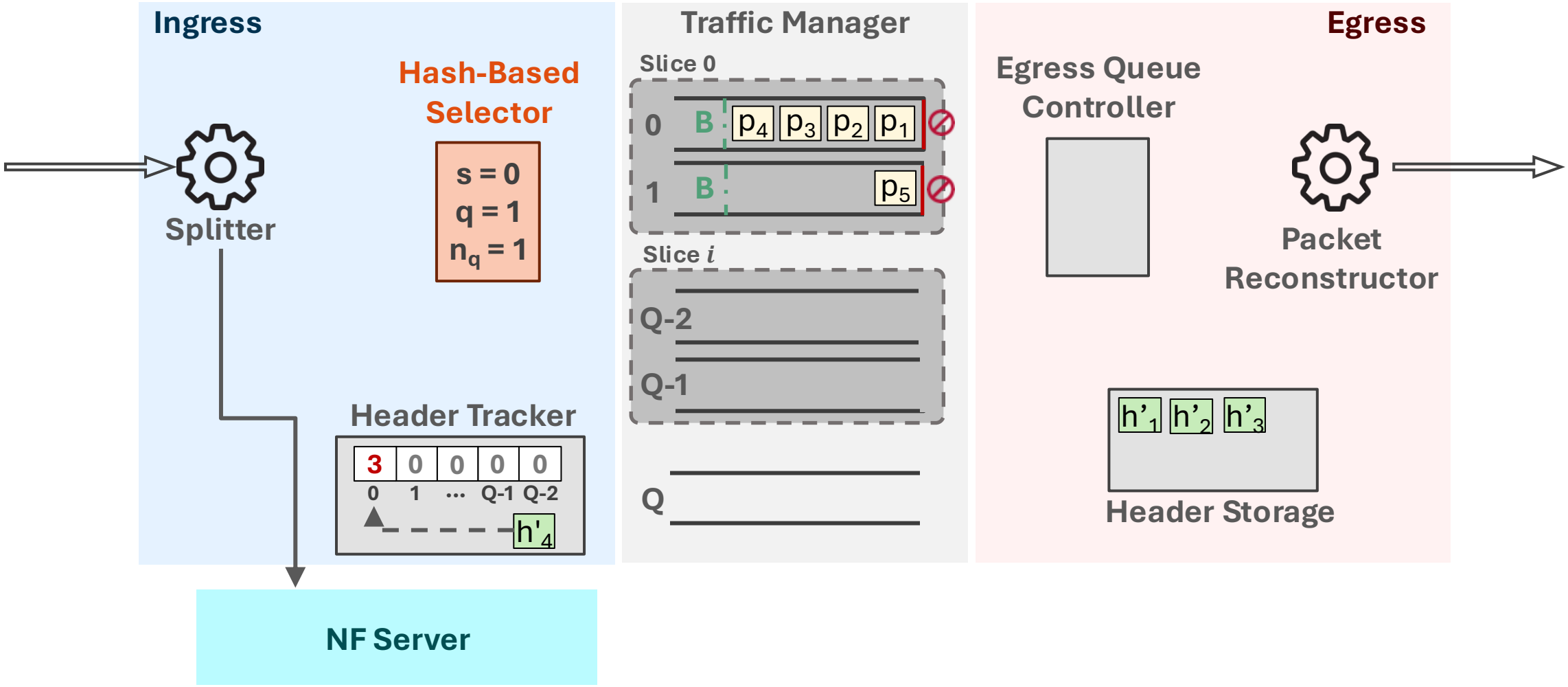
Queue-Mem: Batch Creation



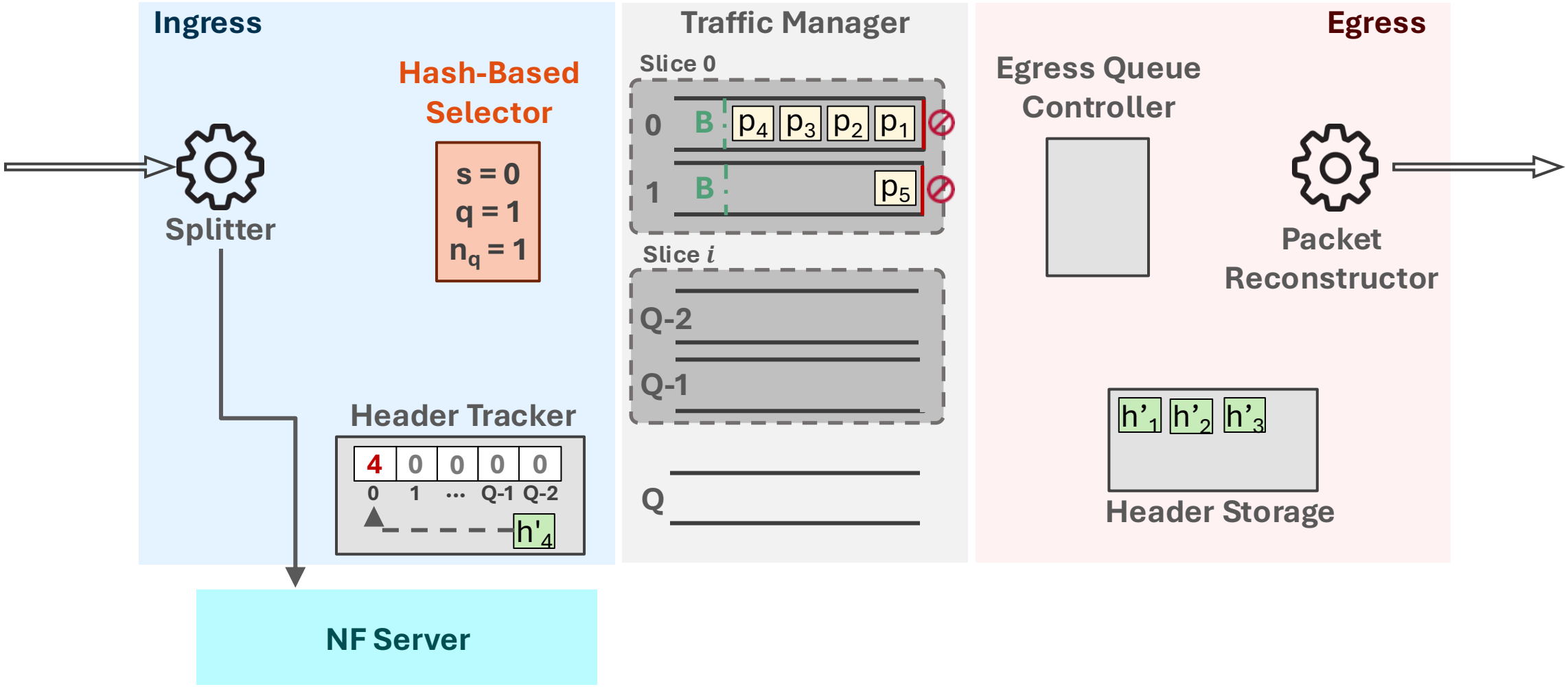
Queue-Mem: Batch Release



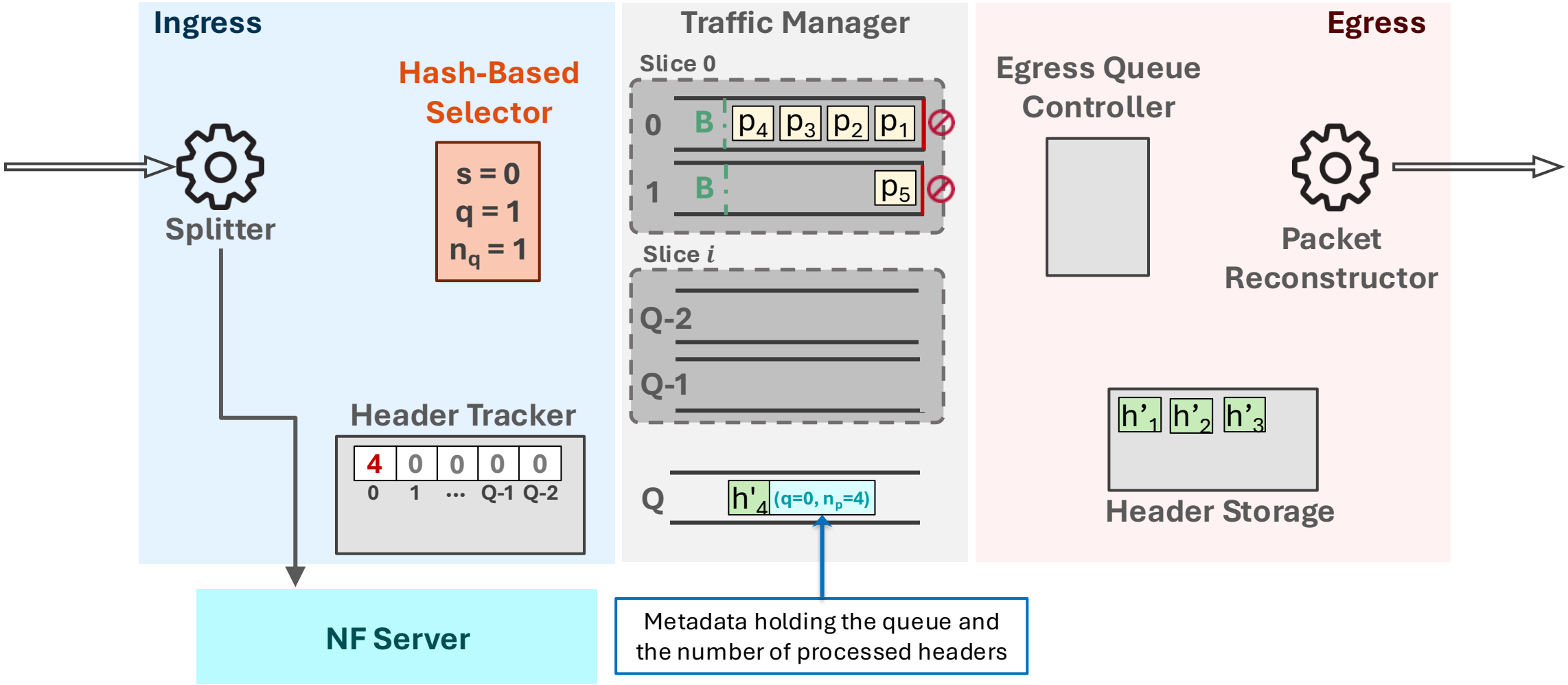
Queue-Mem: Batch Release



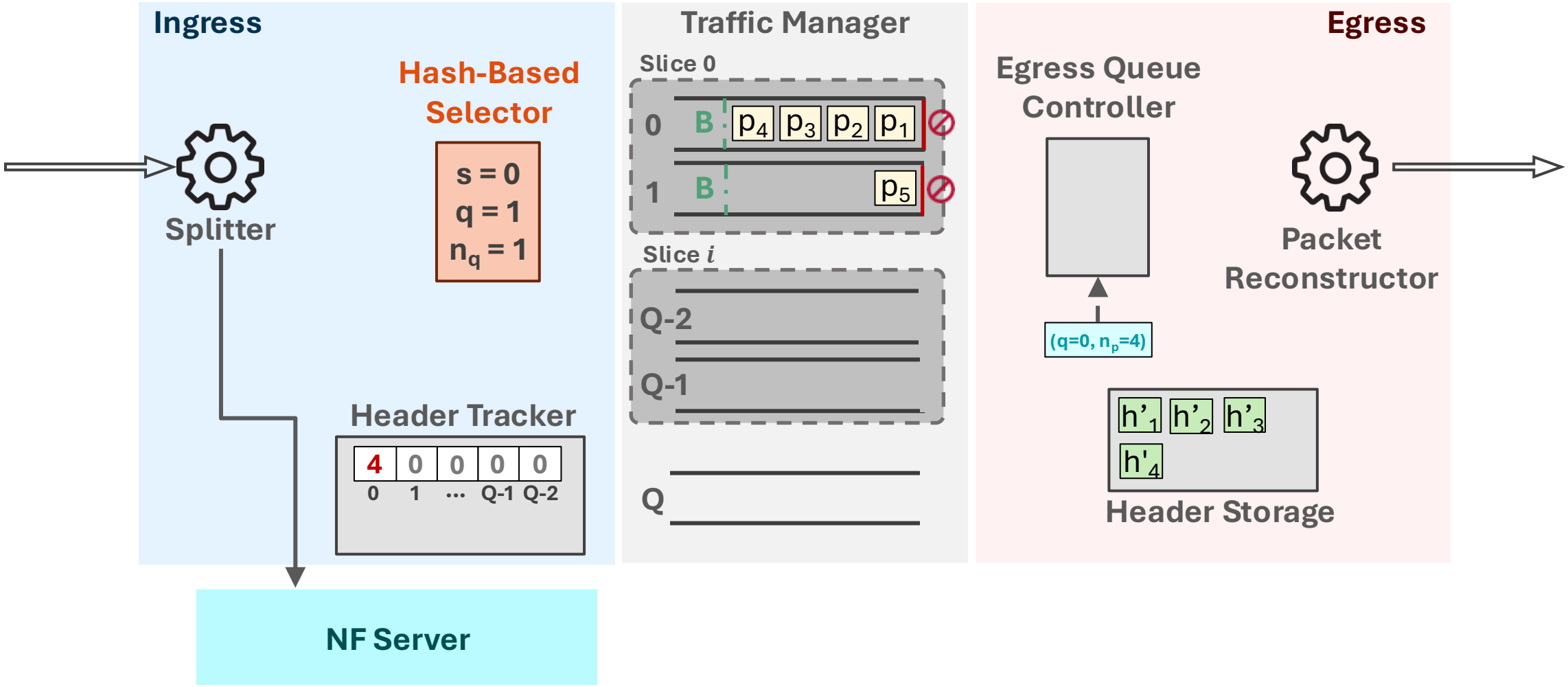
Queue-Mem: Batch Release



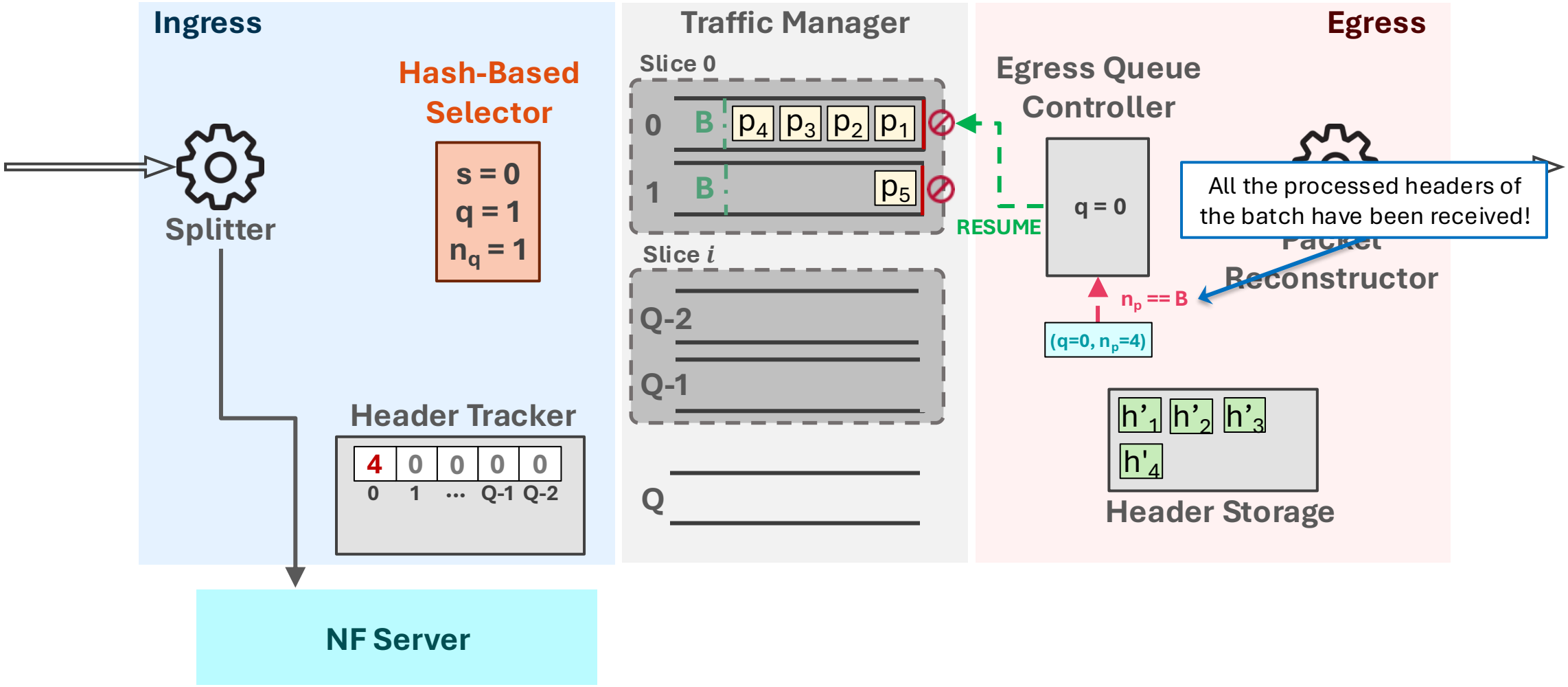
Queue-Mem: Batch Release



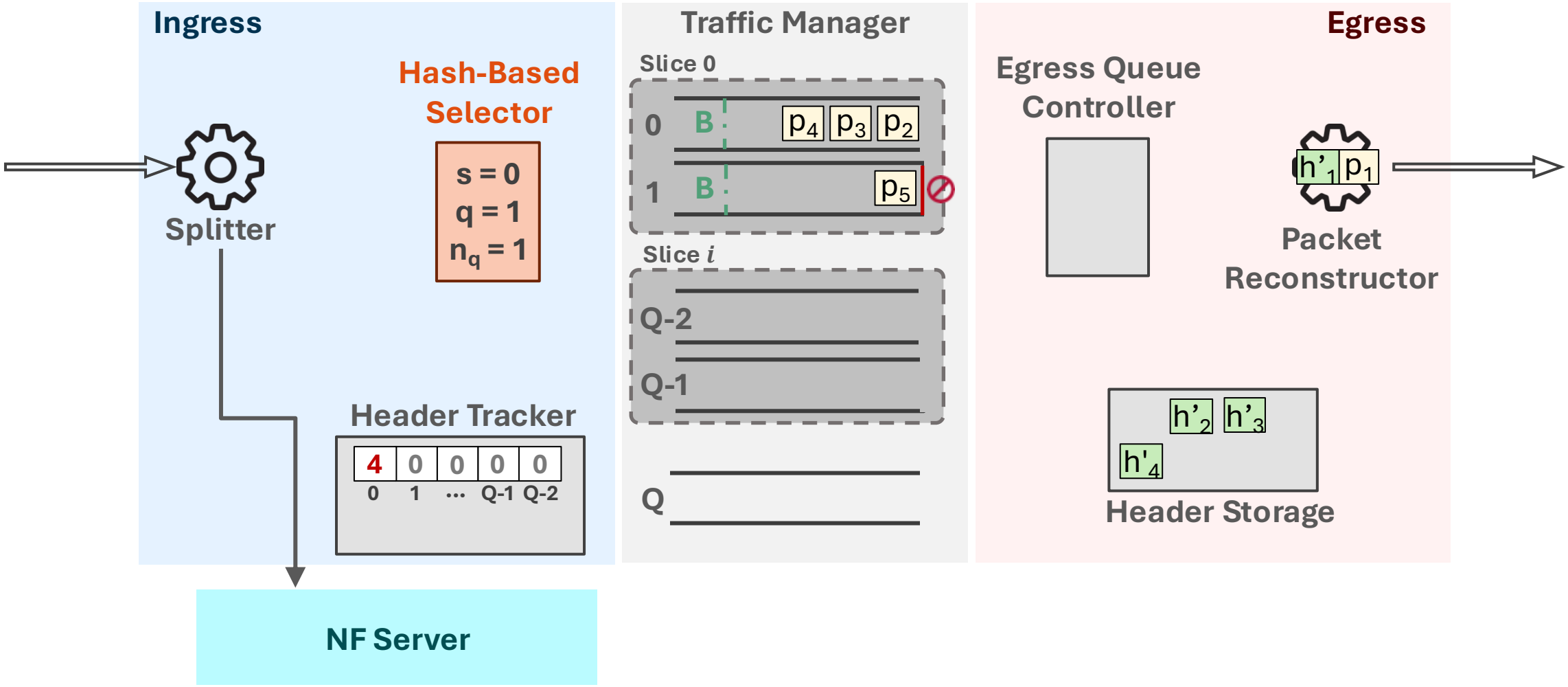
Queue-Mem: Batch Release



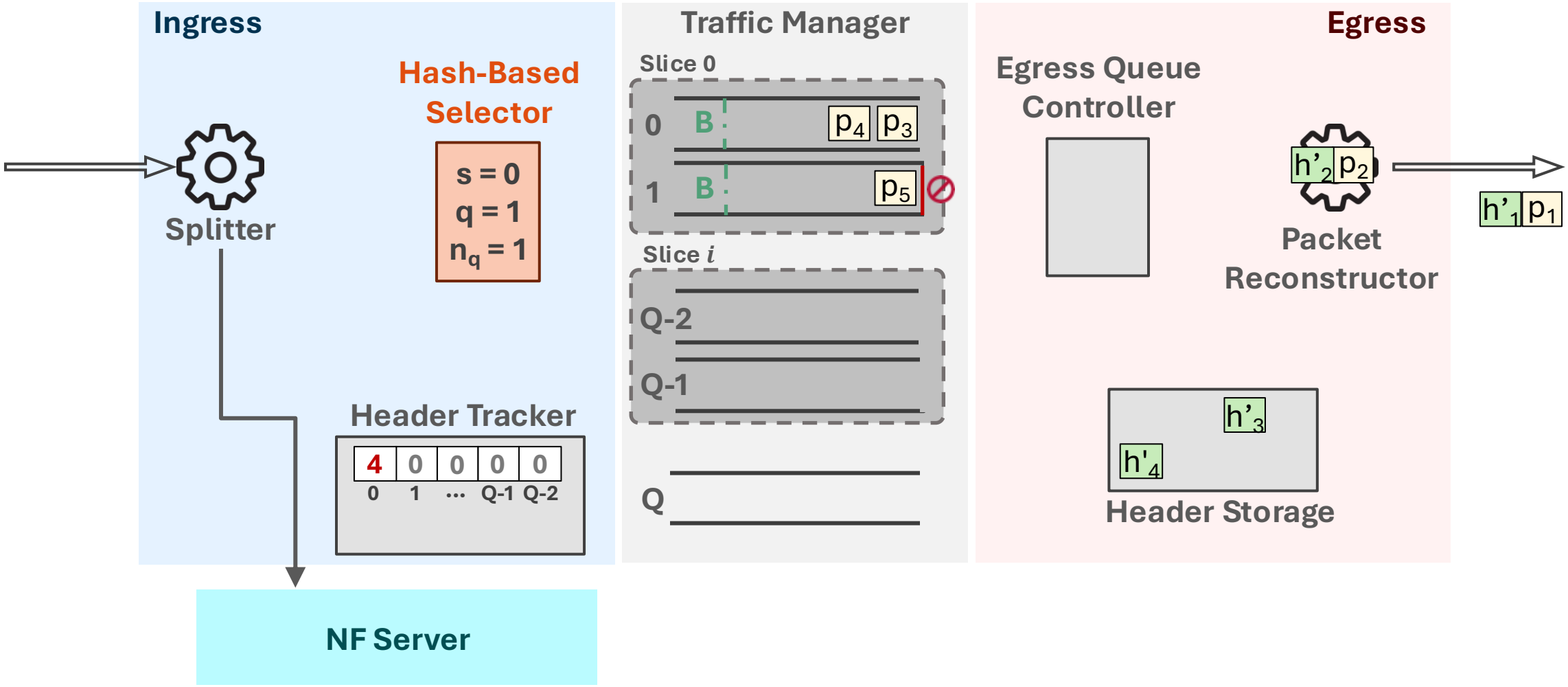
Queue-Mem: Batch Release



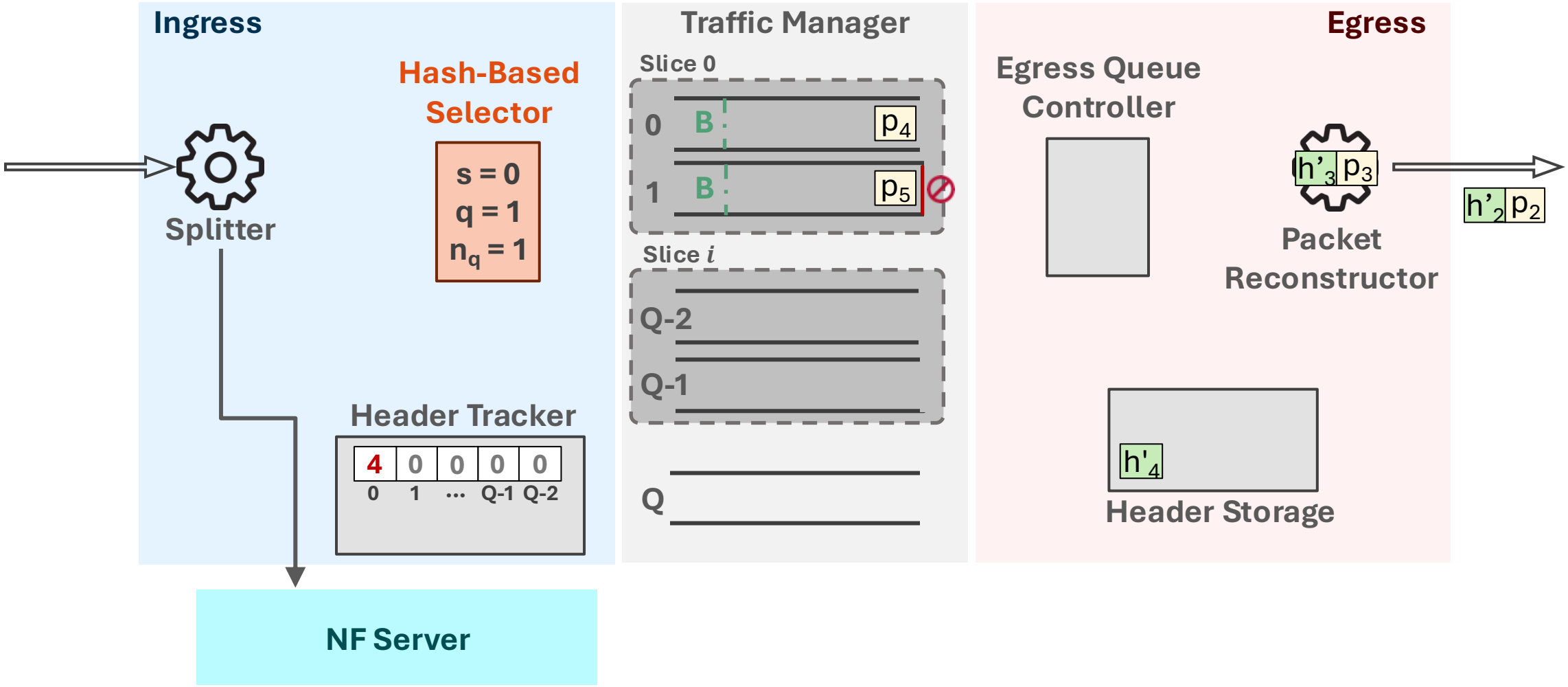
Queue-Mem: Batch Release



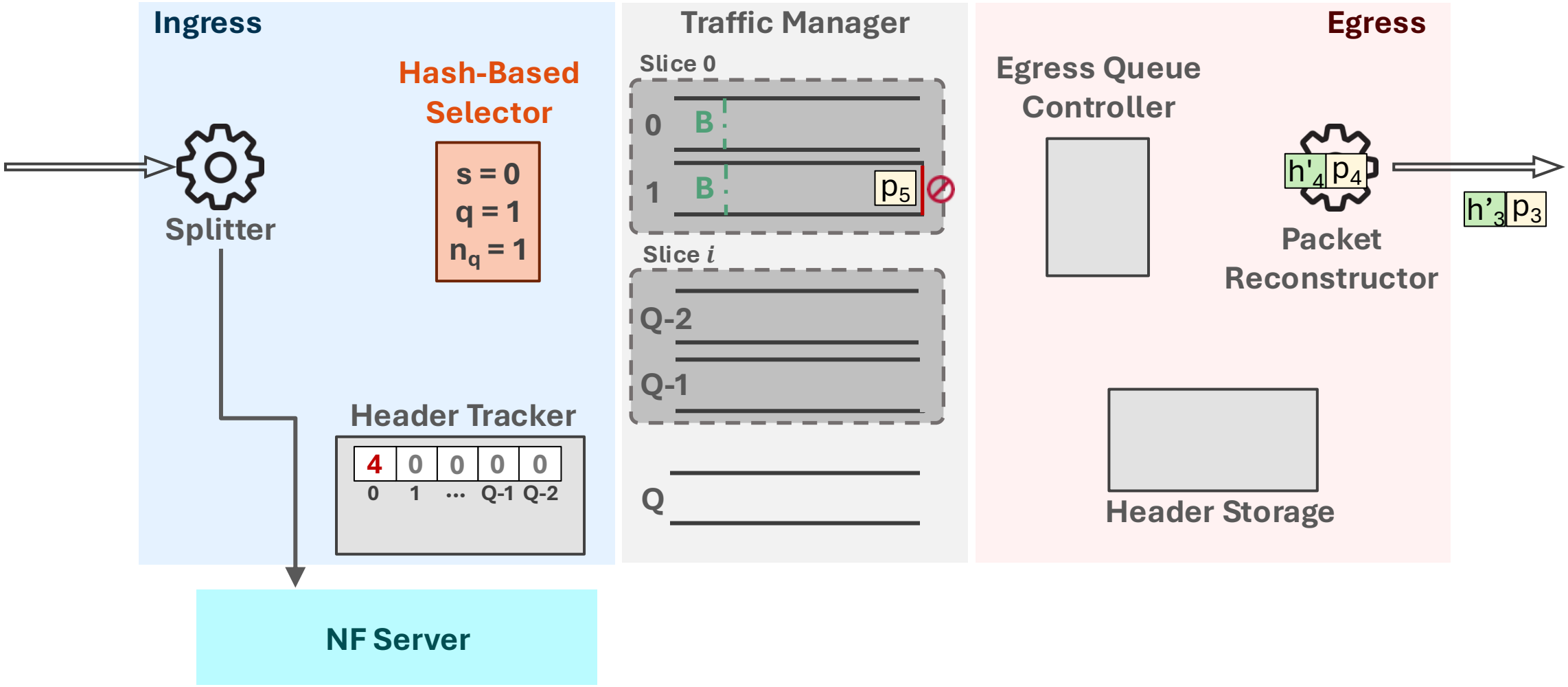
Queue-Mem: Batch Release



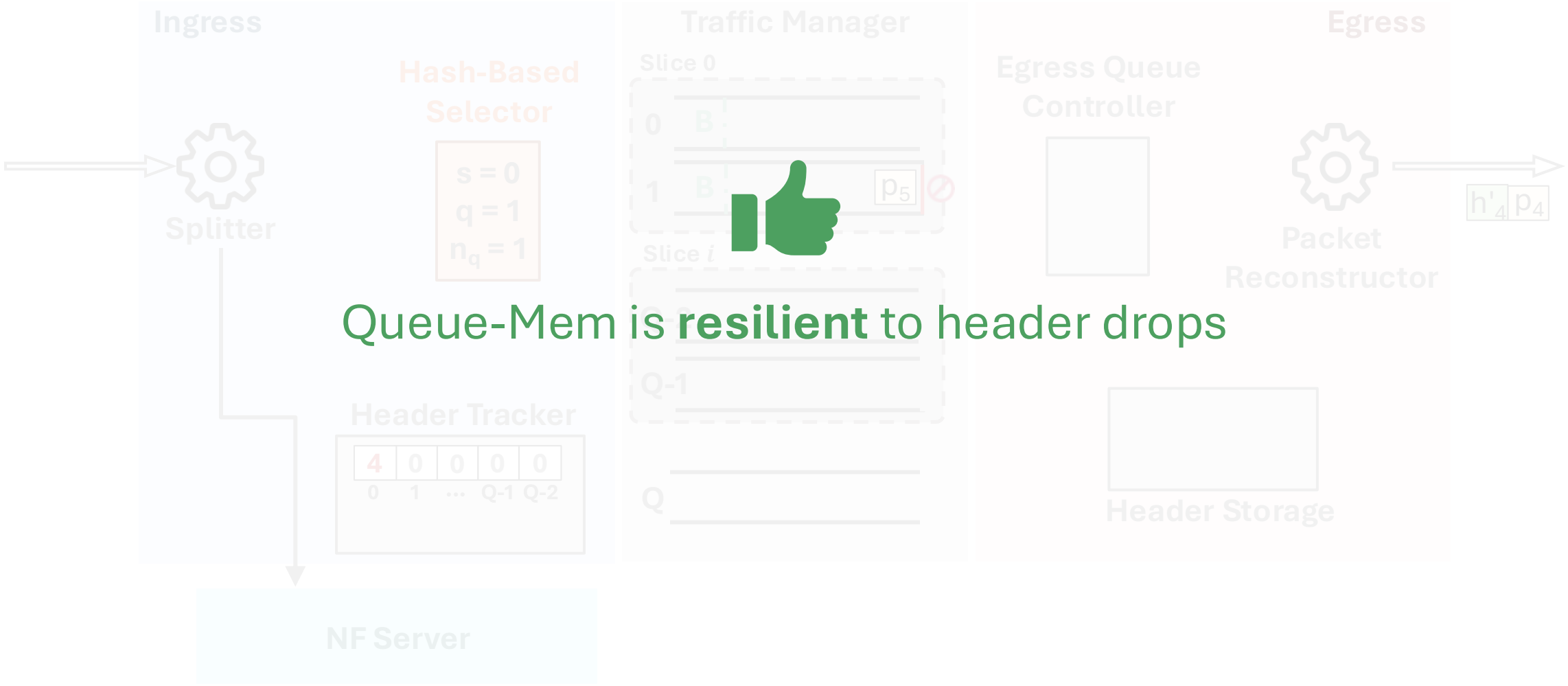
Queue-Mem: Batch Release



Queue-Mem: Batch Release



Queue-Mem: Batch Release

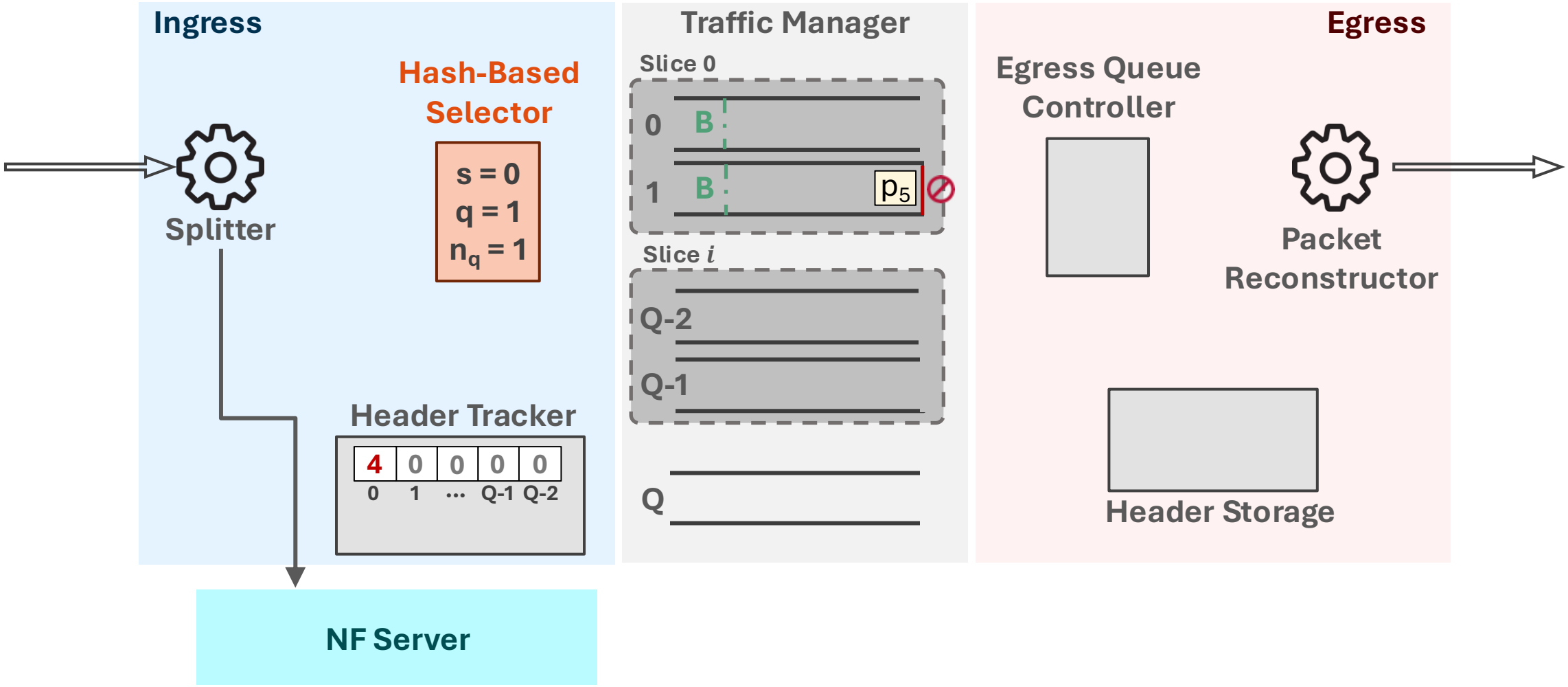


Queue-Mem is **resilient** to header drops

Queue-Mem: Benefits

✓ **No additional resources** to store payloads

✓ **No modifications** on the switch hardware/NF

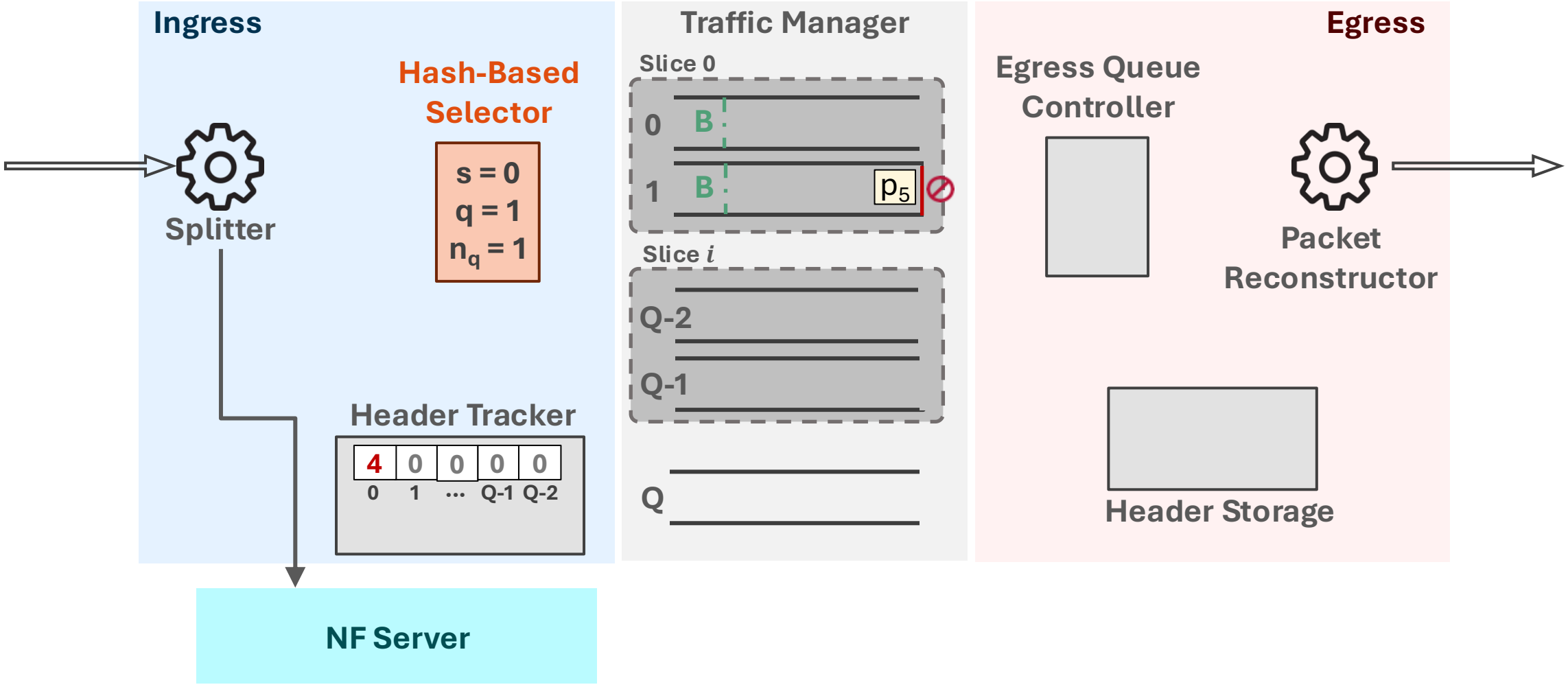


Queue-Mem: Benefits

✓ **No additional resources** to store payloads

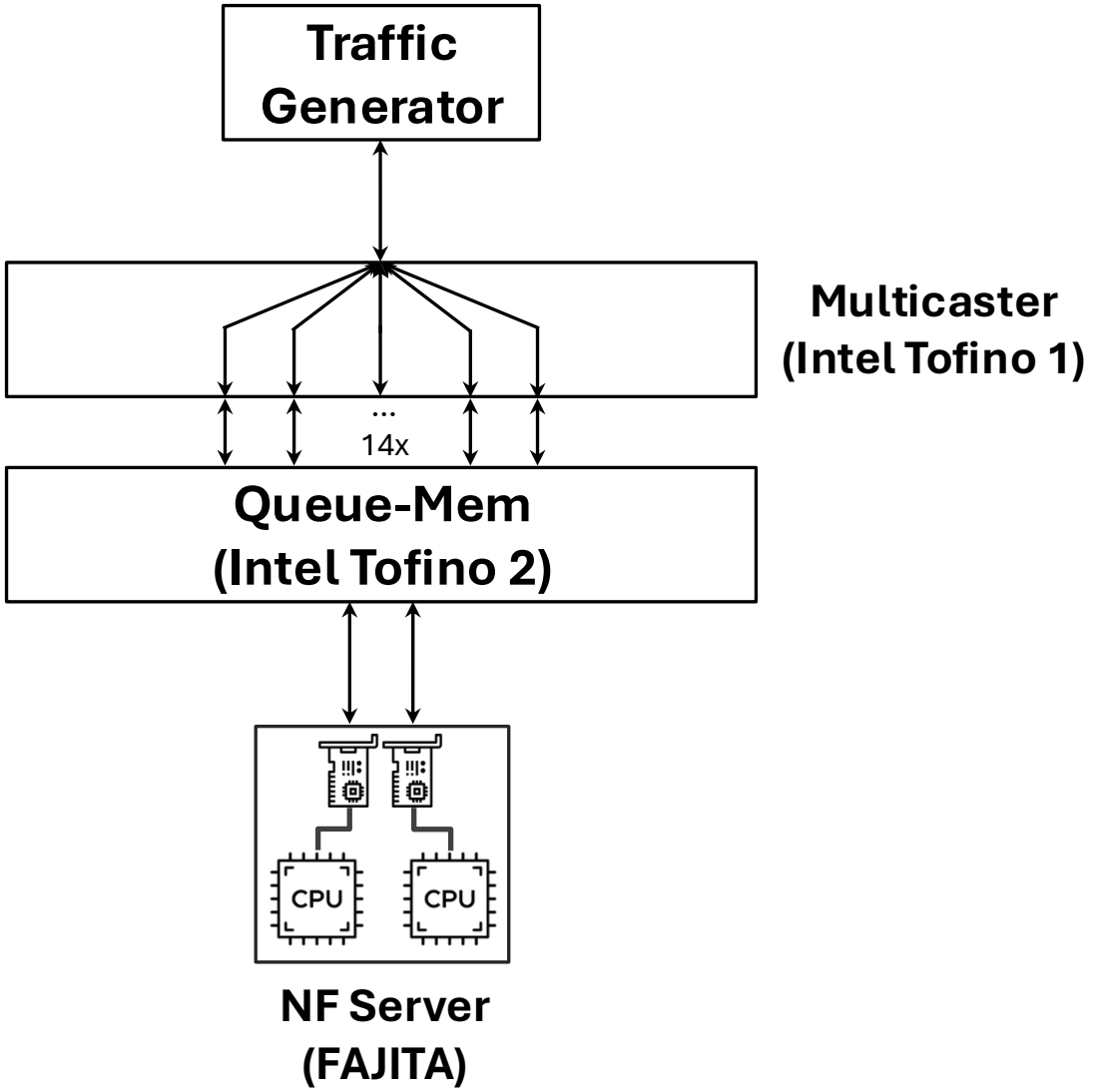
✓ **No modifications** on the switch hardware/NF

Near-optimal power consumption: ~517W to process 1Tbps of 1.5KB packets!



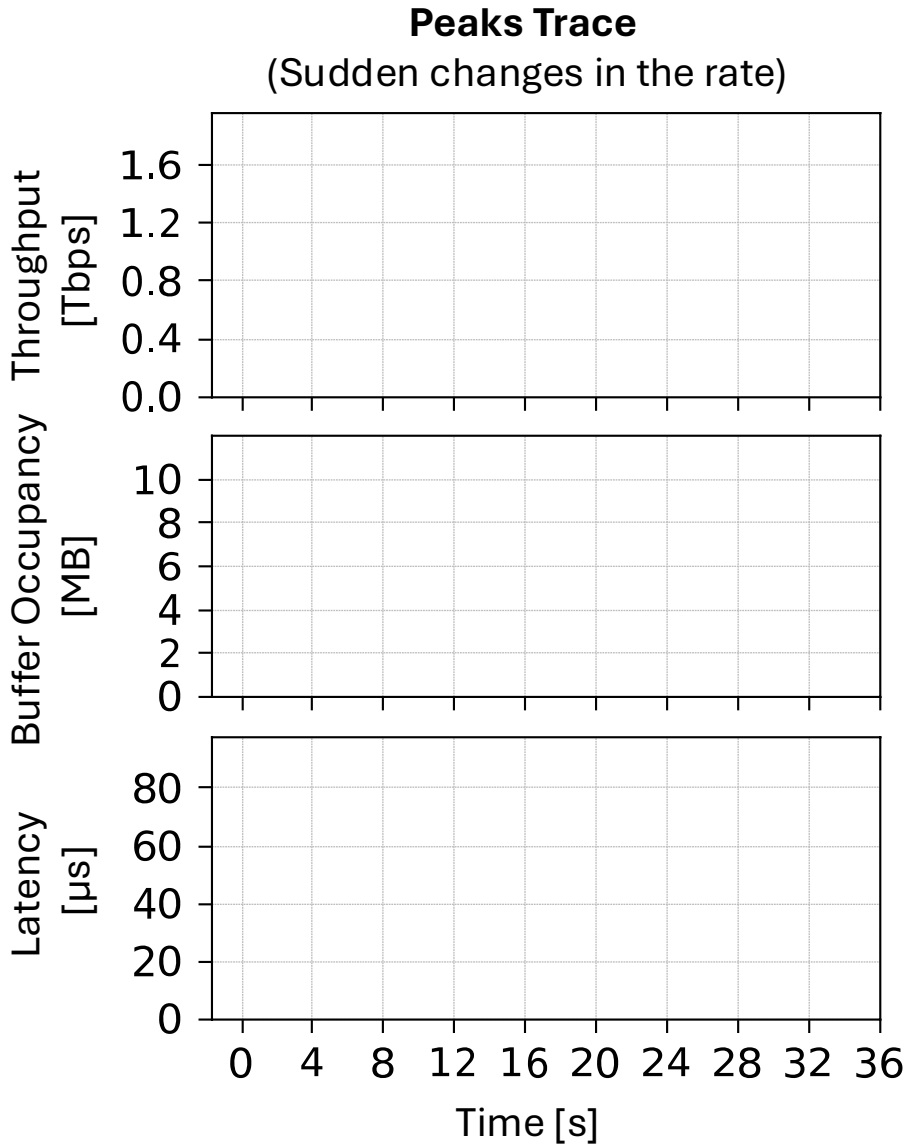
Evaluation

Evaluation: Testbed



Evaluation

Does Queue-Mem handle different traffic patterns?

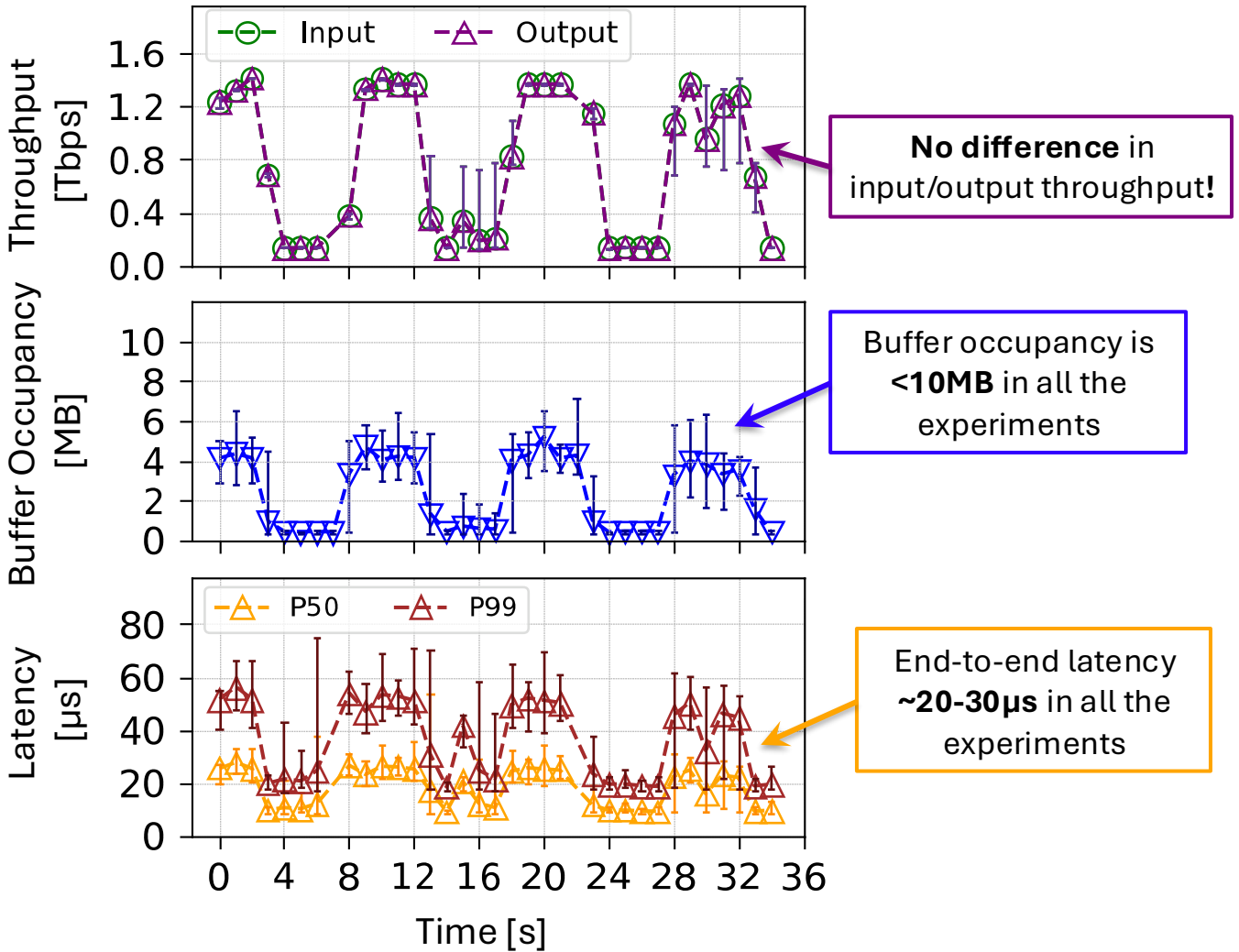


Evaluation

Does Queue-Mem handle different traffic patterns?

Peaks Trace
(Sudden changes in the rate)

More traces in the paper!



No difference in input/output throughput!

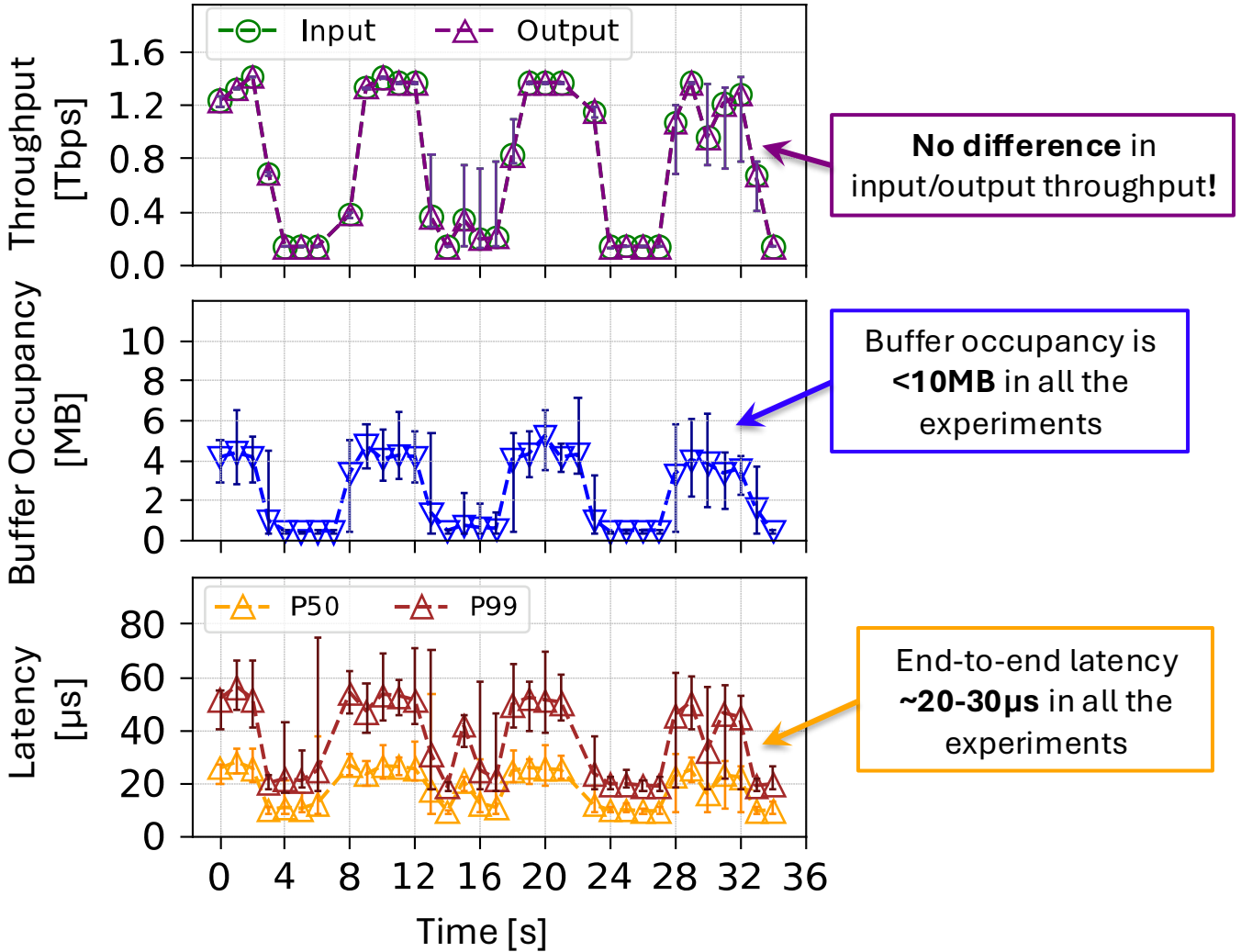
Buffer occupancy is **<10MB** in all the experiments

End-to-end latency **~20-30µs** in all the experiments

Evaluation

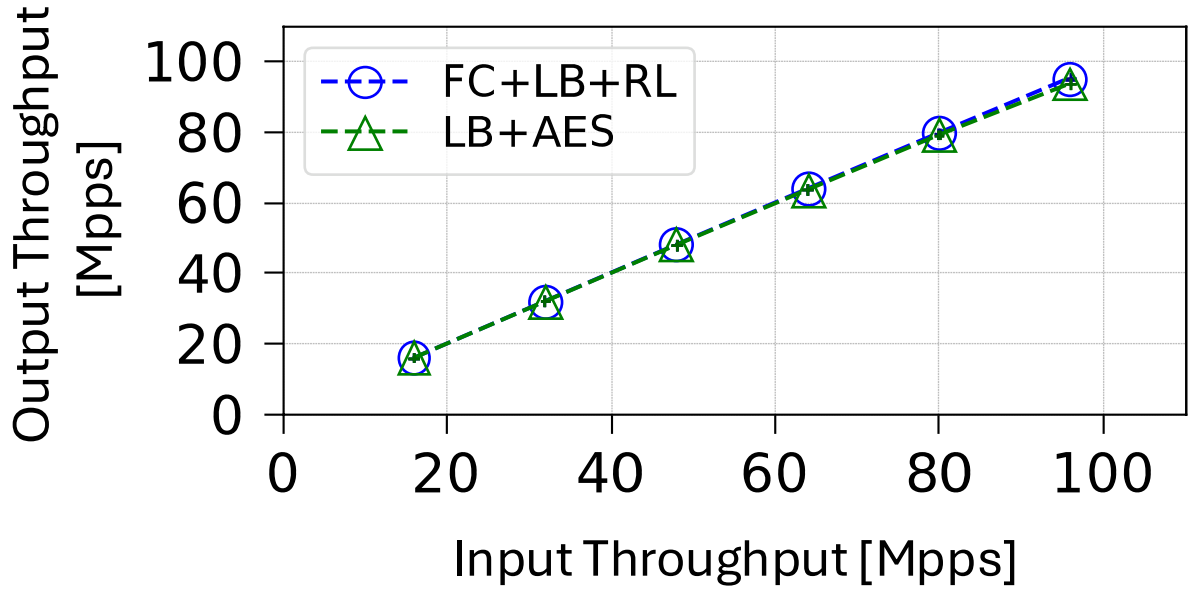
Does Queue-Mem handle different traffic patterns?

Queue-Mem is **robust** across dynamic traffic patterns with consistently **low buffer occupancy**!



Evaluation

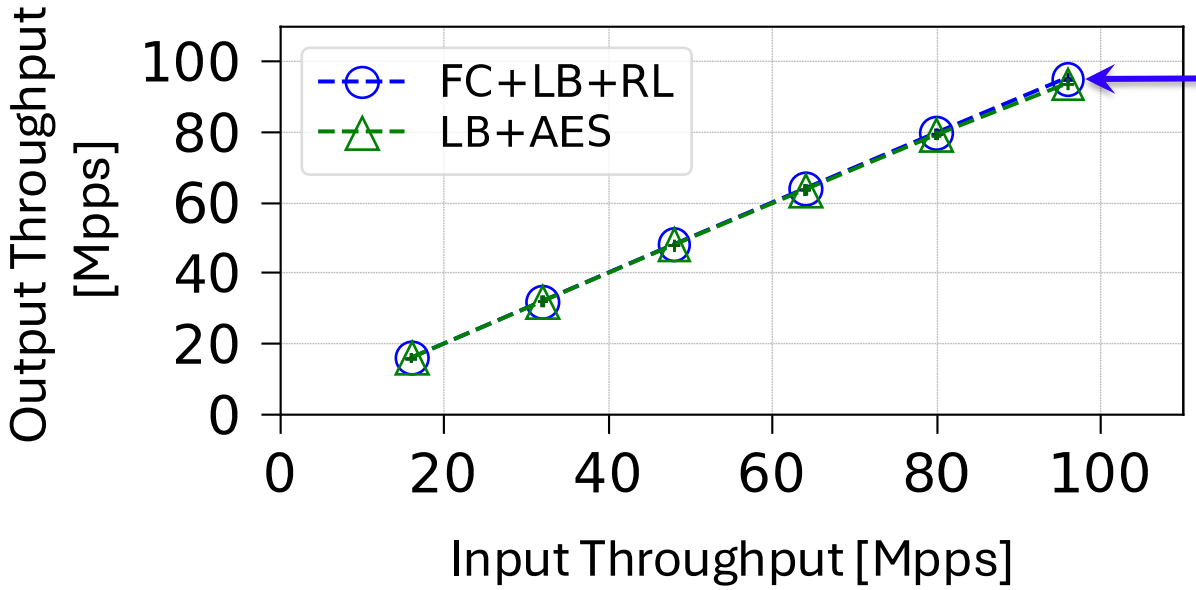
Does Queue-Mem support advanced NFs?

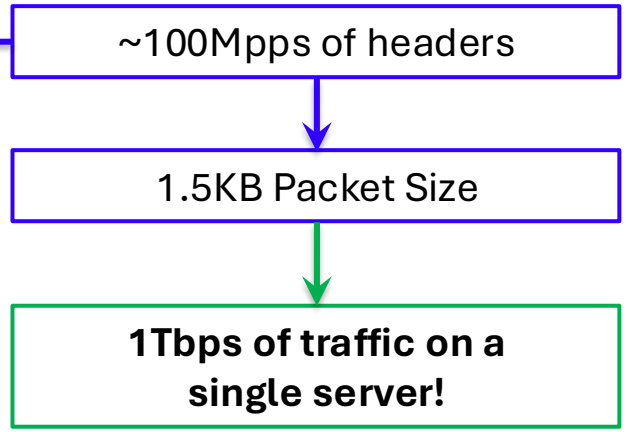


Evaluation

Does Queue-Mem support advanced NFs?

Queue-Mem runs **advanced, stateful NFs at Tbps** with the **lowest energy per bit**!

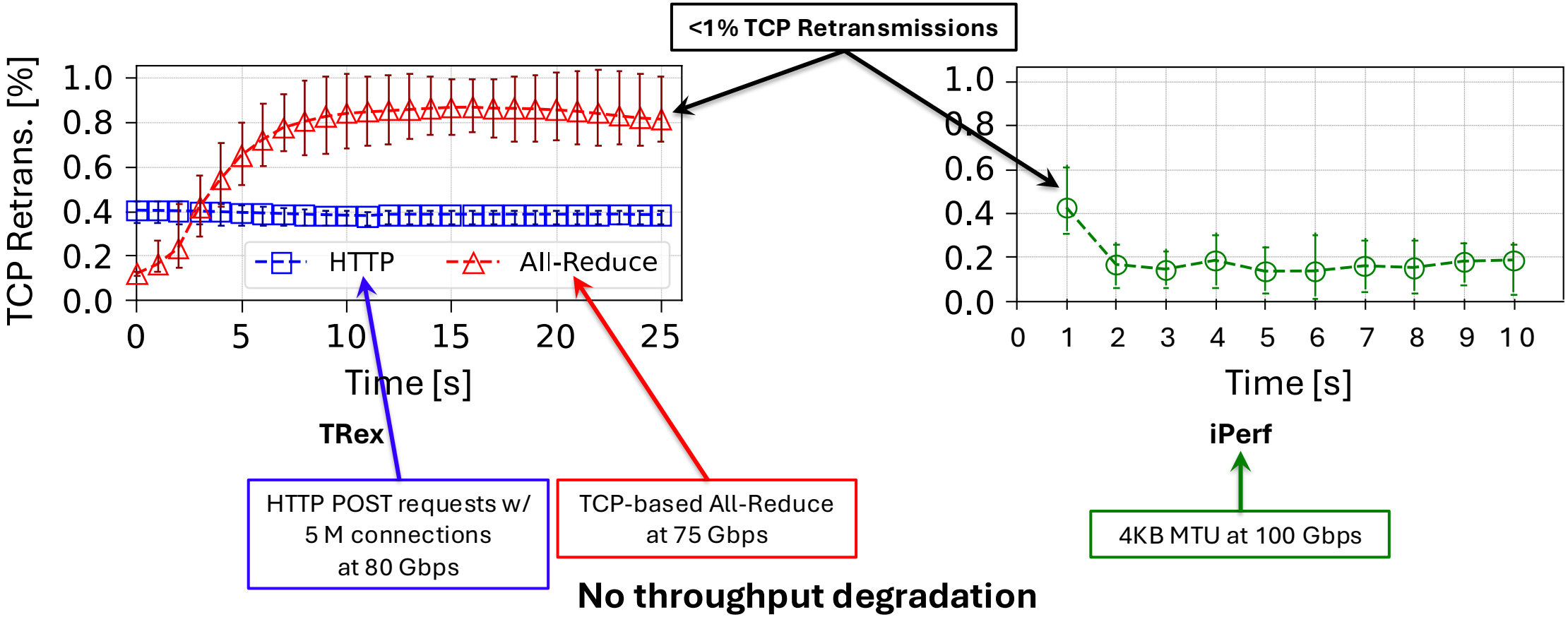




Evaluation

Does Queue-Mem support real workloads?

Queue-Mem sustains **stable TCP performance** under **real workloads**!



Conclusions

The **Queue-Mem** system:

- Presents the first **energy-efficient** packet-splitting approach for Tbps NF processing
- Proposes a **queue-based FIFO buffer** to store payloads on existing ASIC switches
- Supports **advanced NF deployments** and **realistic workloads**
- Spurs a question on whether **expose APIs** for controlling the forwarding buffer