Deliberately Congesting a Switch for Better Network Functions Performance

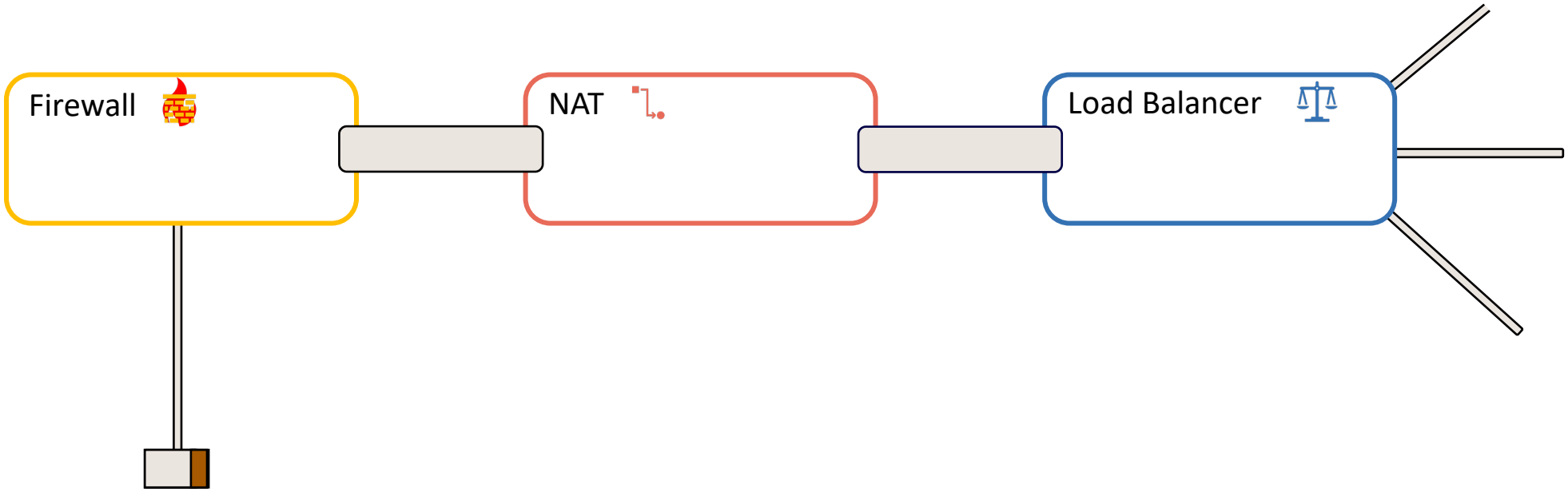
Mariano Scazzariello [*], Tommaso Caiazzi [‡], Marco Chiesa[*]

[*] KTH Royal Institute of Technology, Sweden – [‡] Roma Tre University, Italy

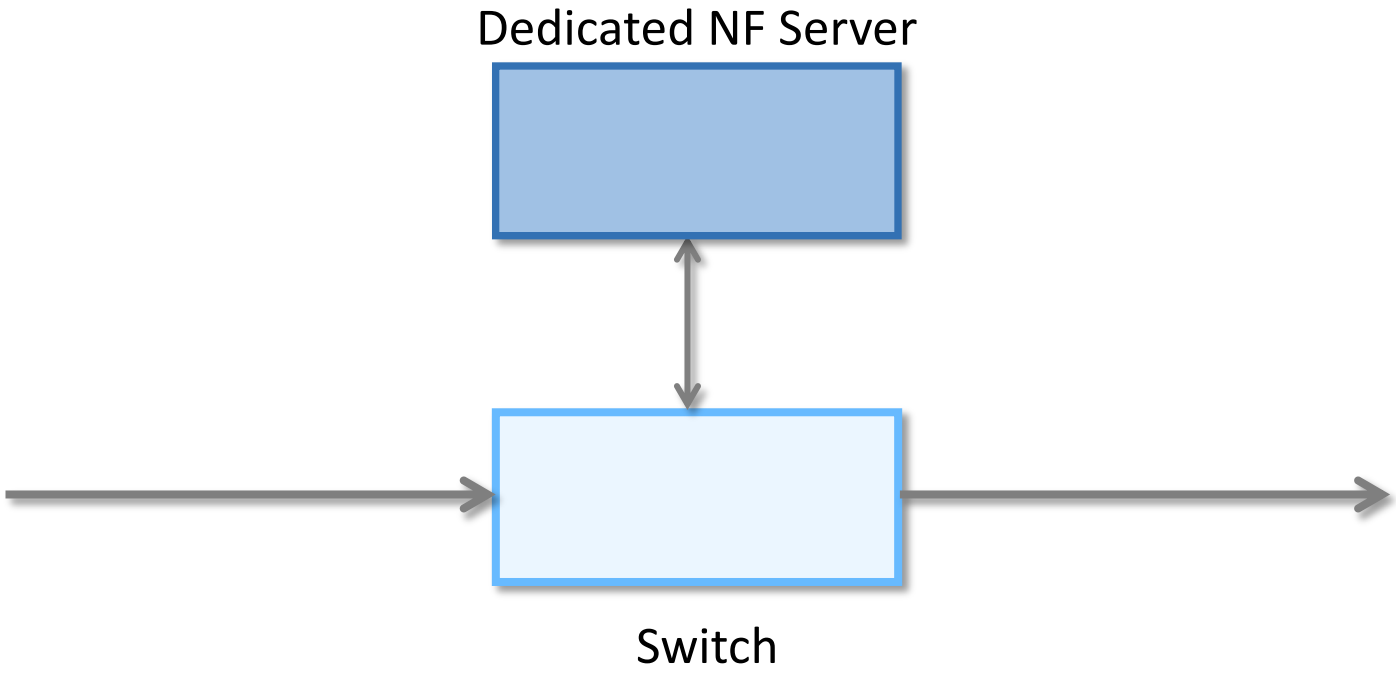
Network Functions Are Pervasive

Network Functions Are Pervasive

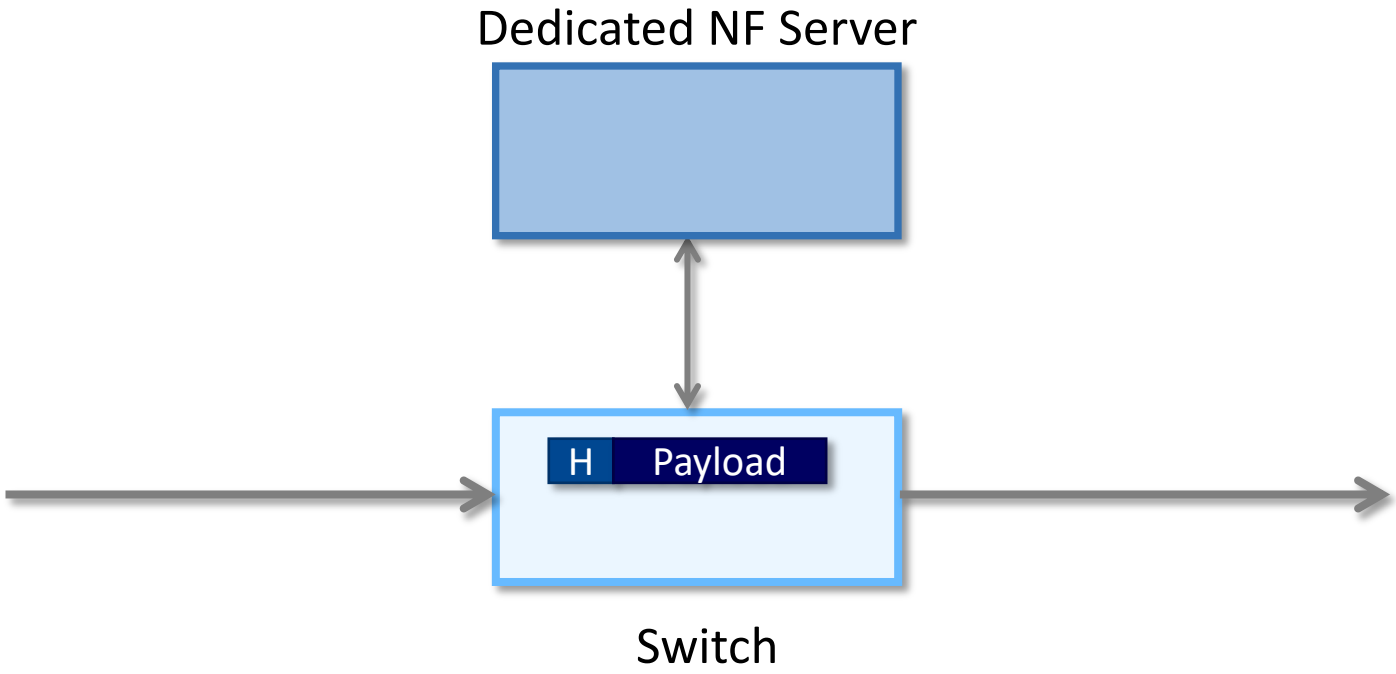
Network Functions Virtualization is an essential architectural paradigm of today's networks



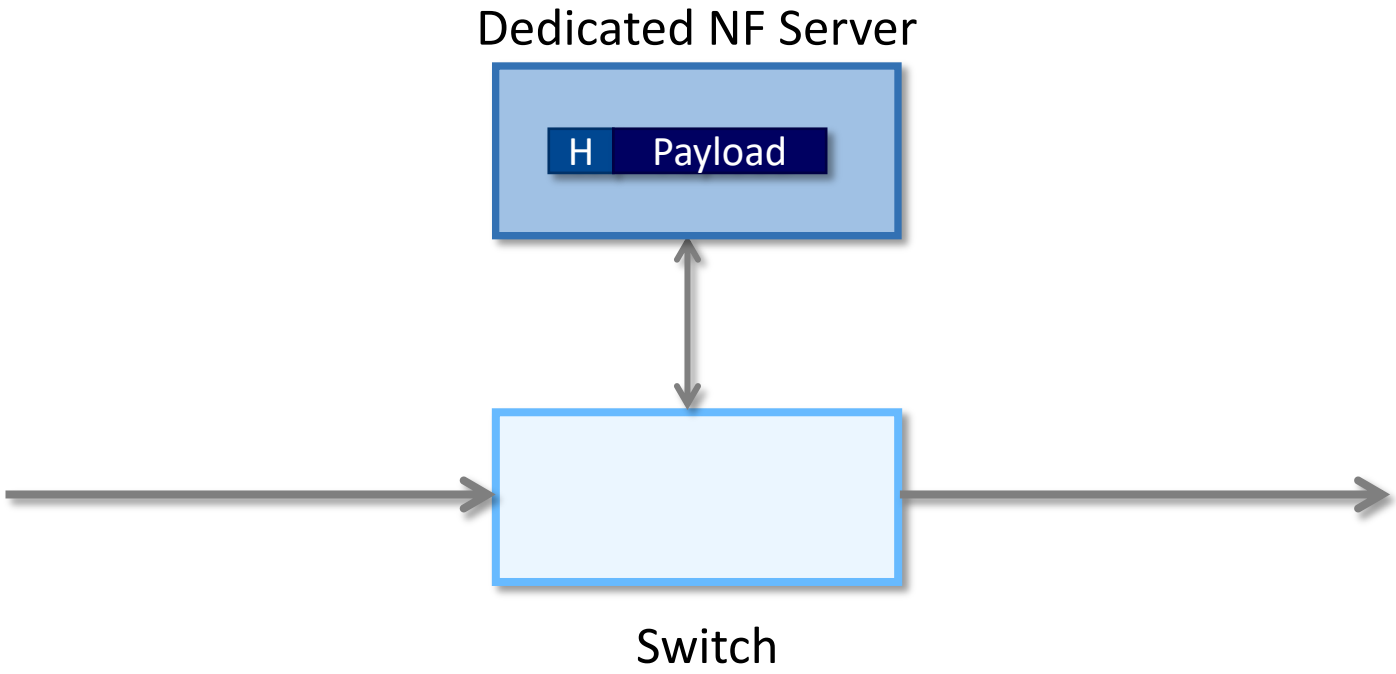
Advanced NF Deployments



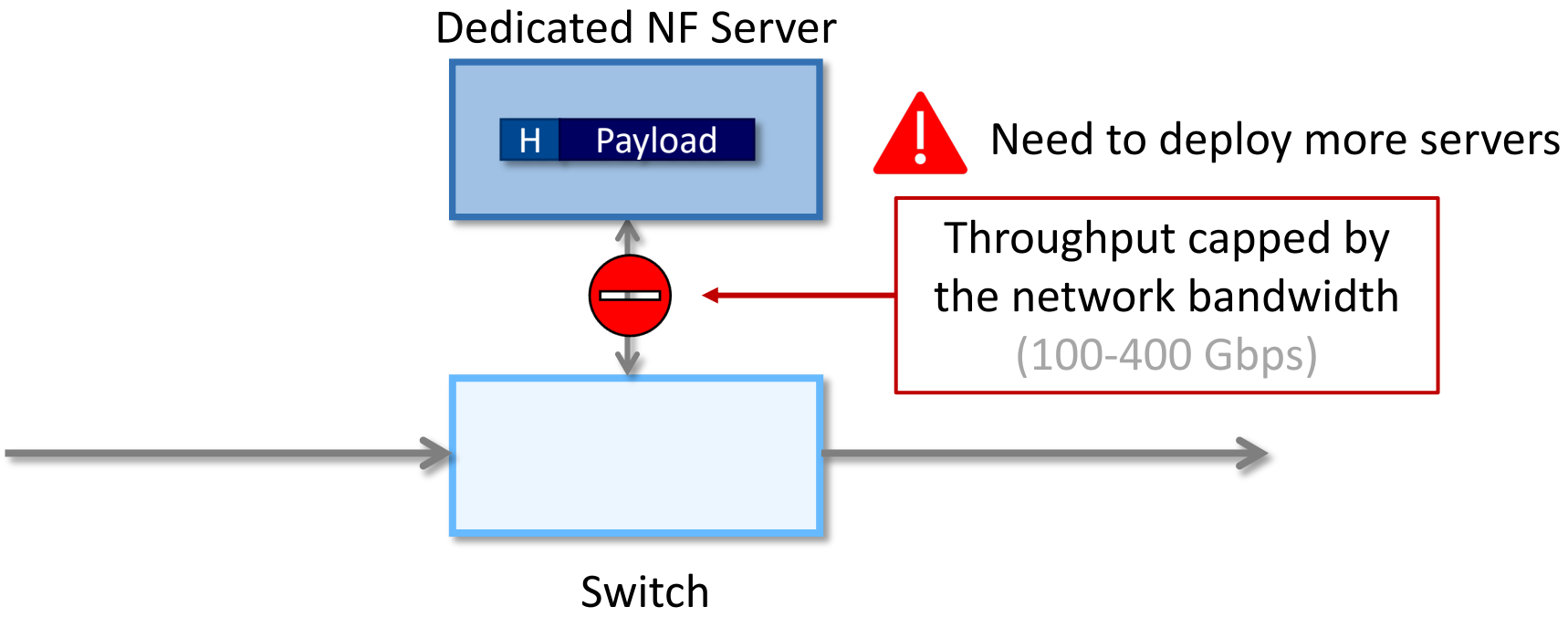
Advanced NF Deployments



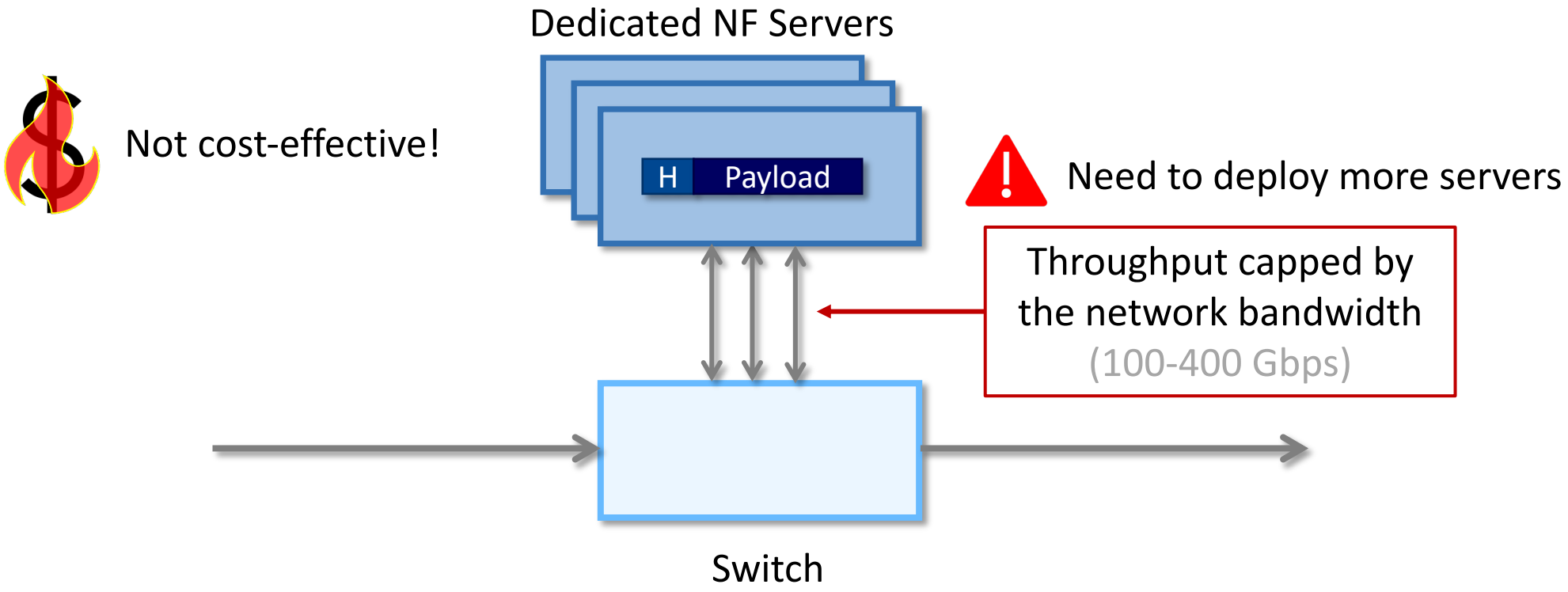
Advanced NF Deployments



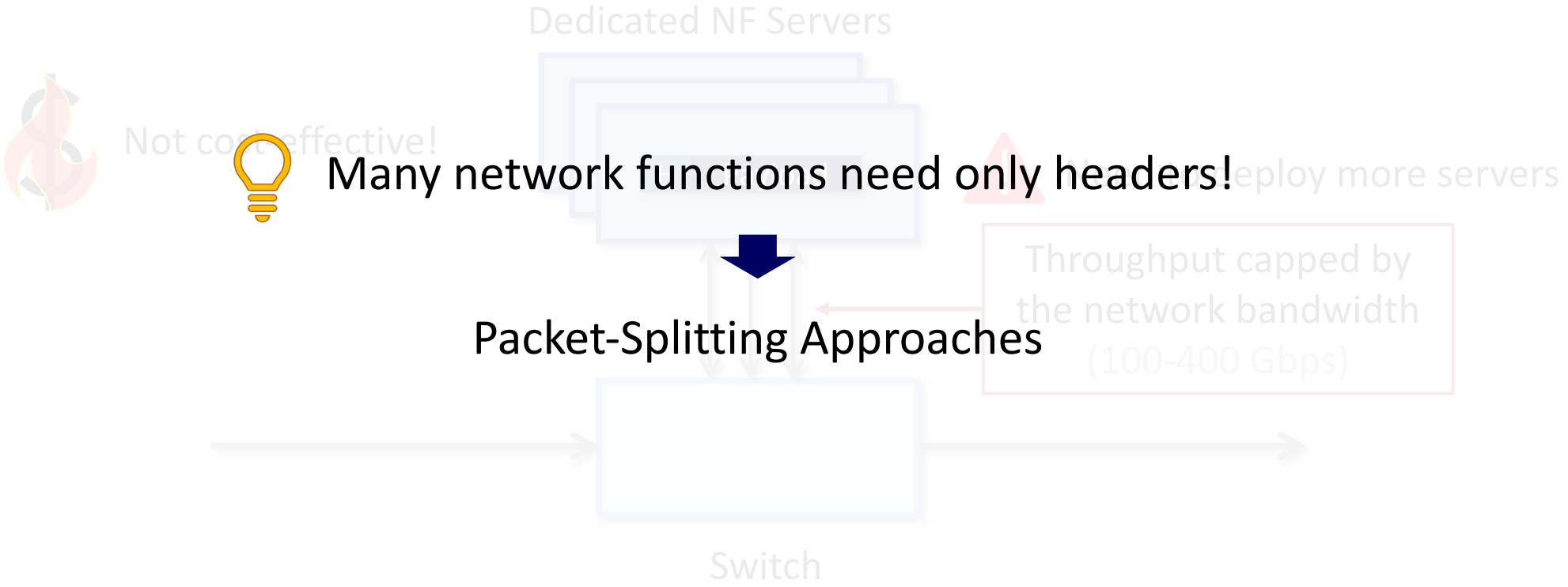
The Bandwidth Limit of Advanced NF Deployments



The Bandwidth Limit of Advanced NF Deployments



The Bandwidth Limit of Traditional NF Deployments



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) 

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) 

Packet-Splitting Approaches

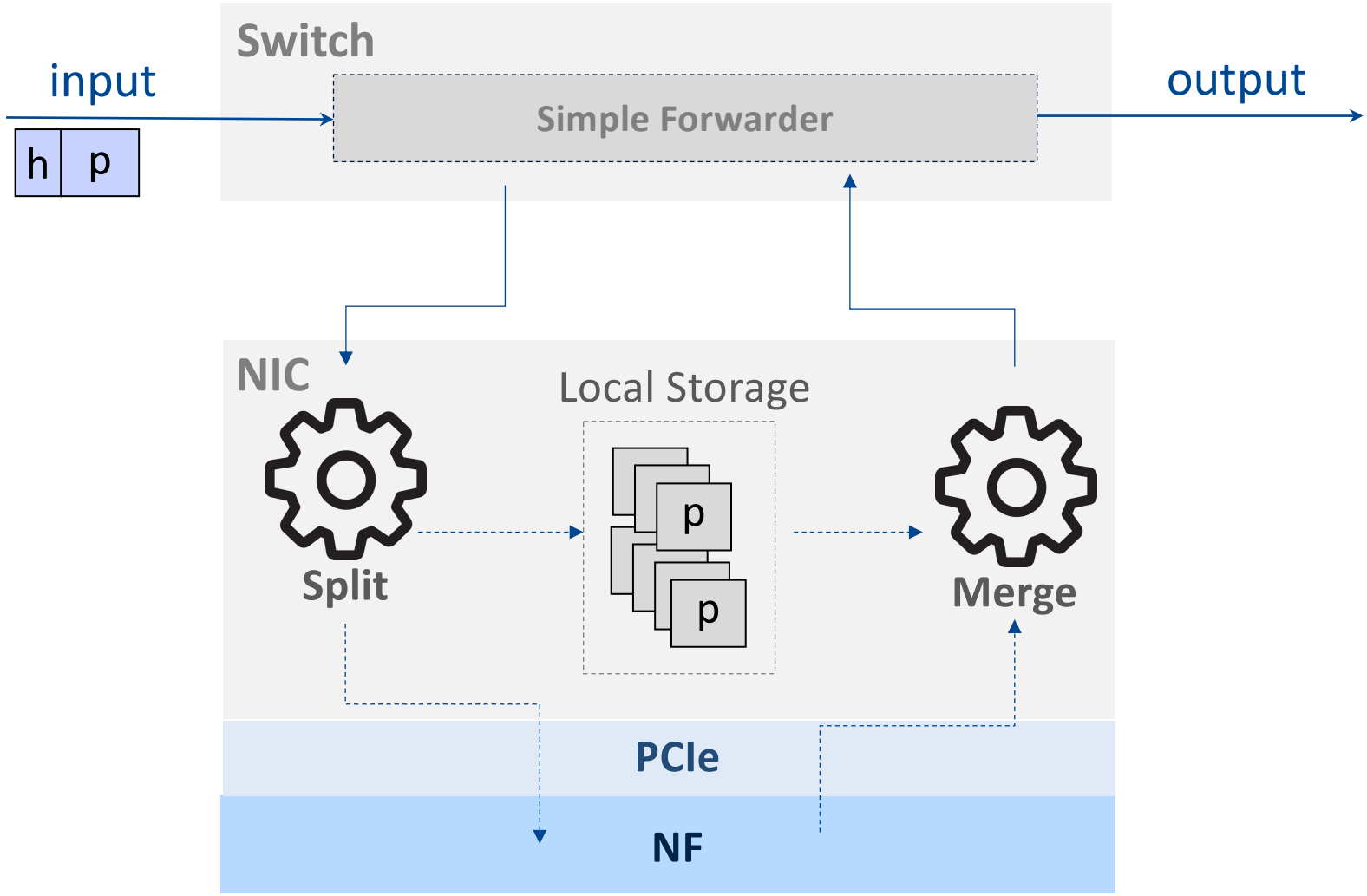
State-of-the-art packet-splitting solutions increase NFs throughput

Storing Payloads in the On-NIC Memory (nicmem) 

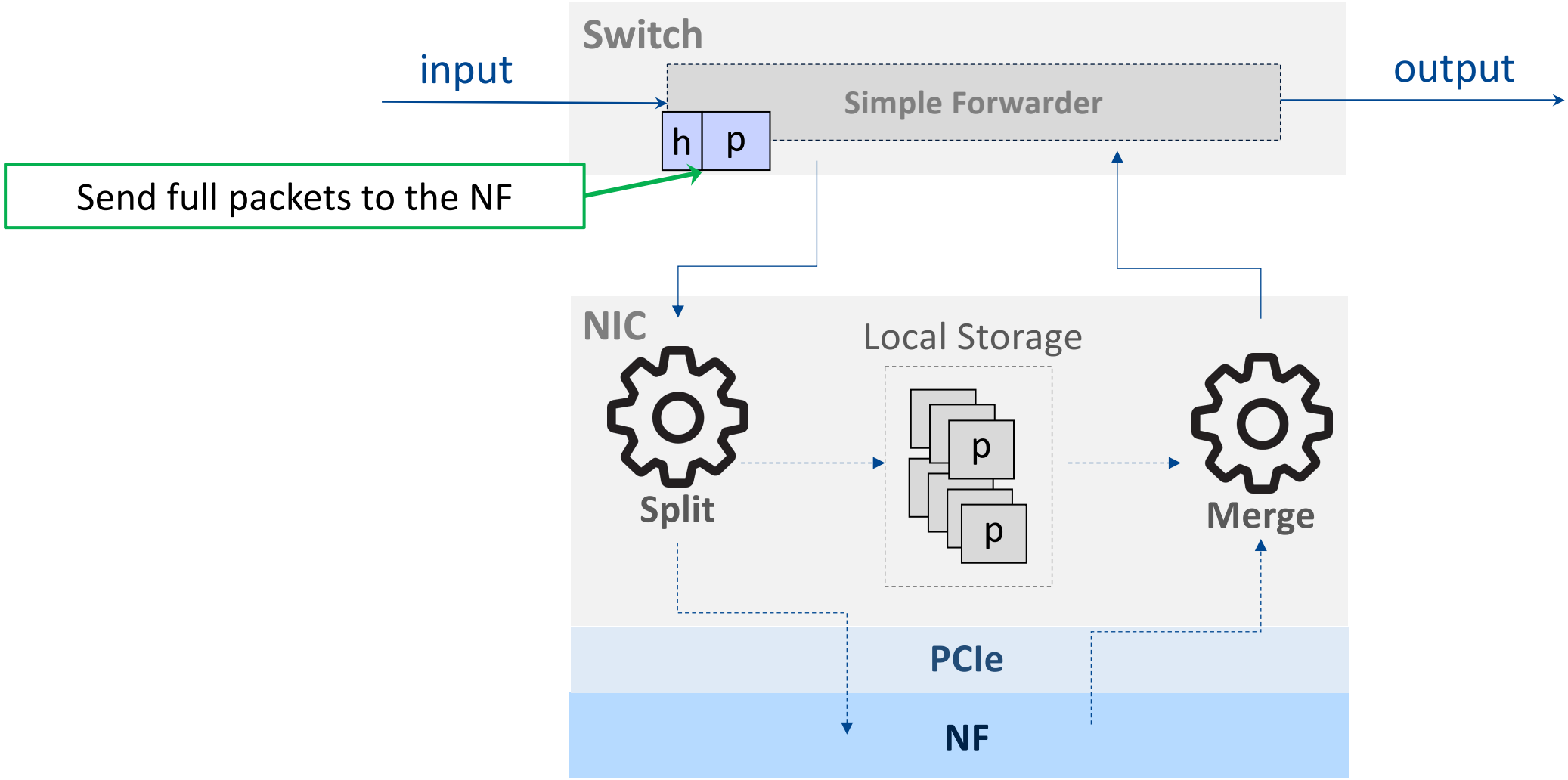
Storing Payloads on Shared Resources (Ribosome) 

Storing a Few Bytes in ASIC SRAM (PayloadPark) 

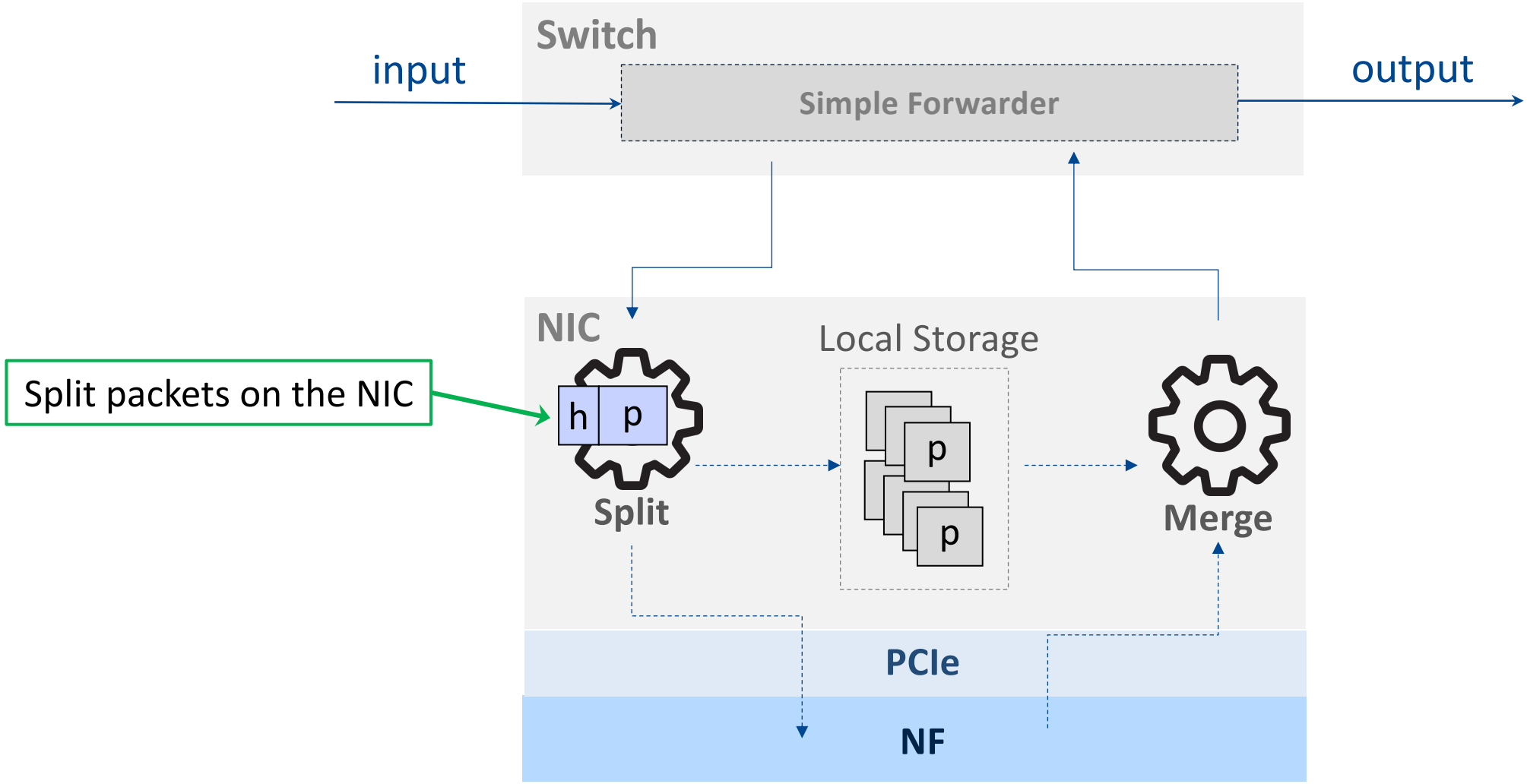
Storing Payloads in the On-NIC Memory (nicmem)



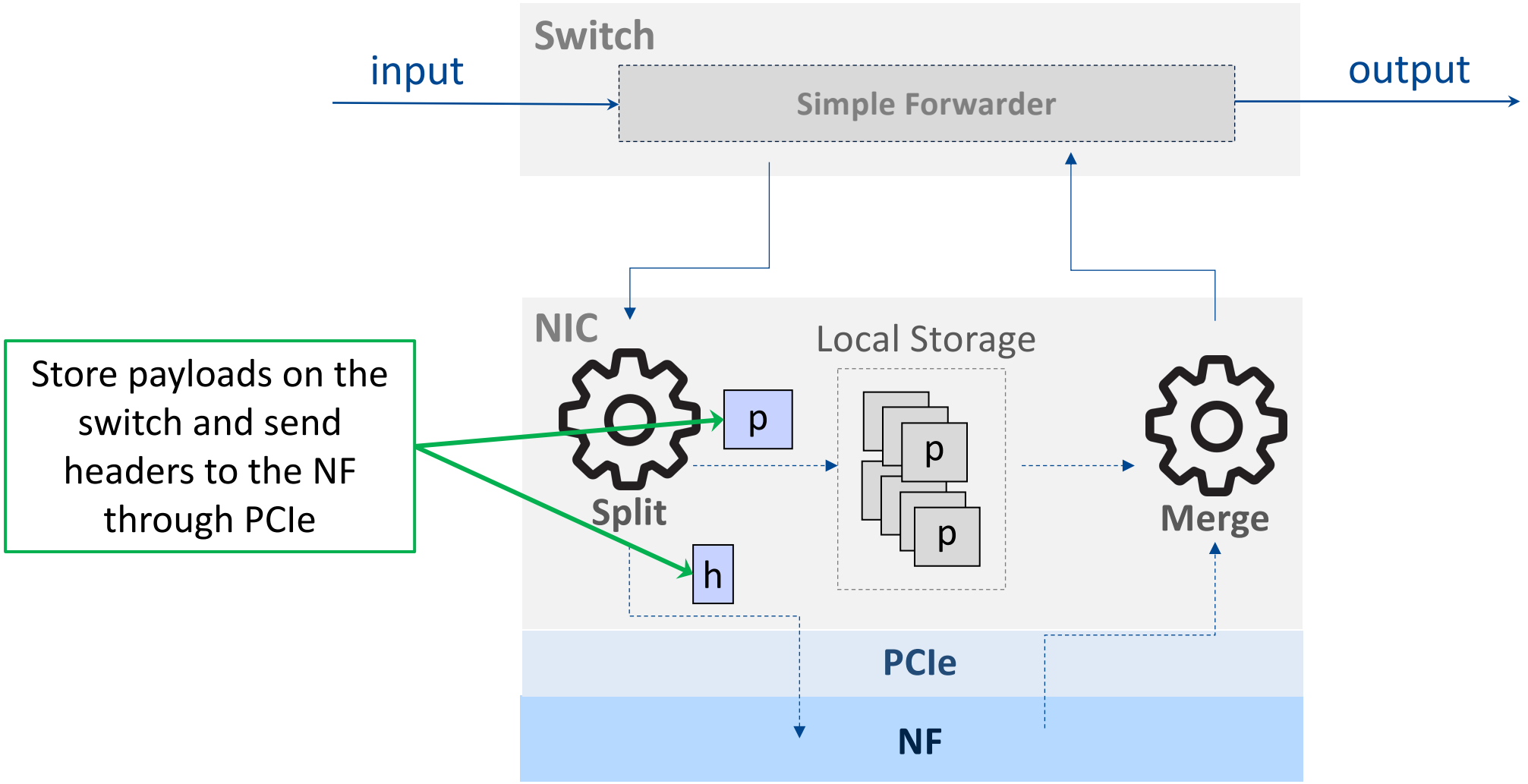
Storing Payloads in the On-NIC Memory (nicmem)



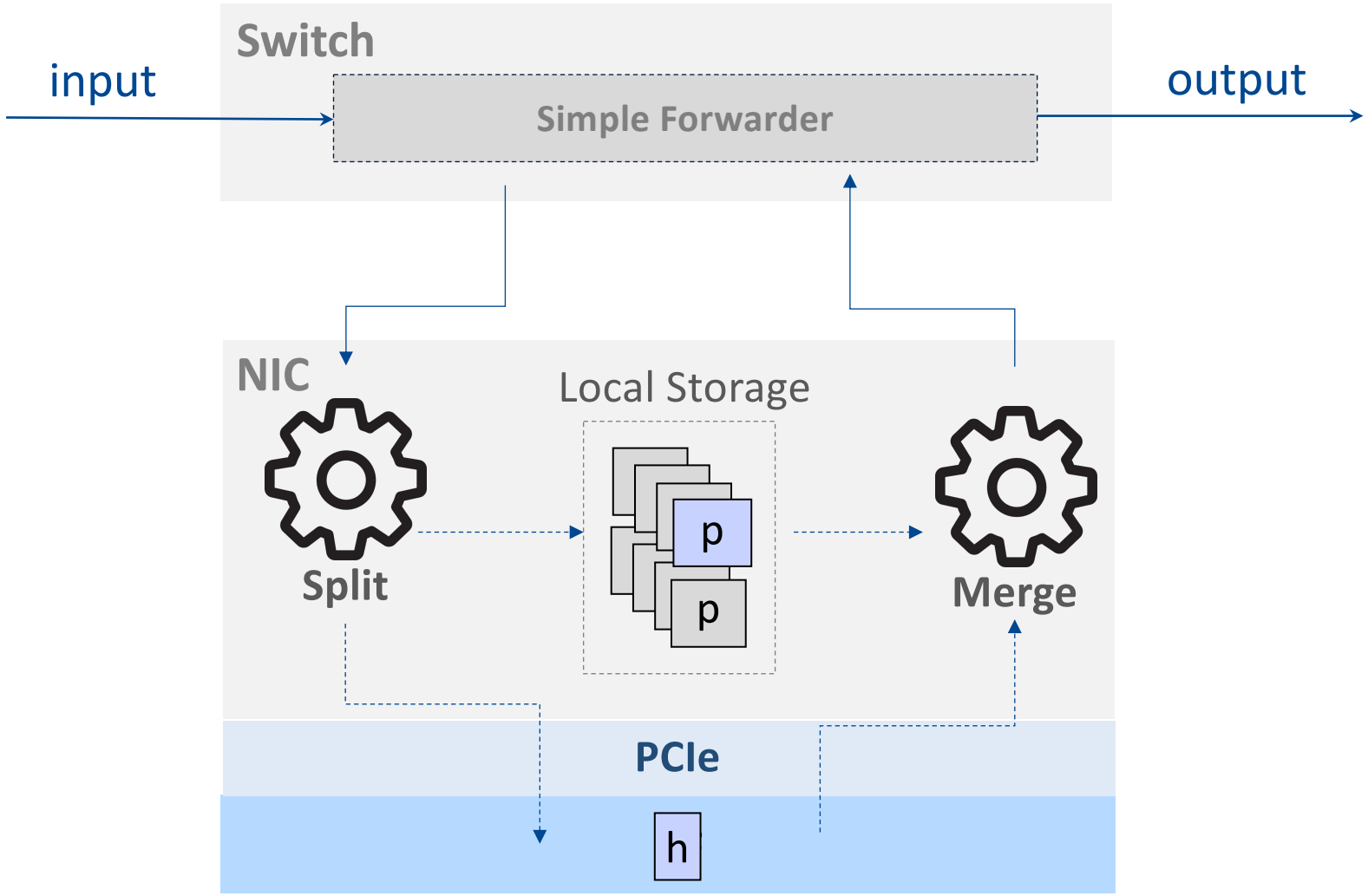
Storing Payloads in the On-NIC Memory (nicmem)



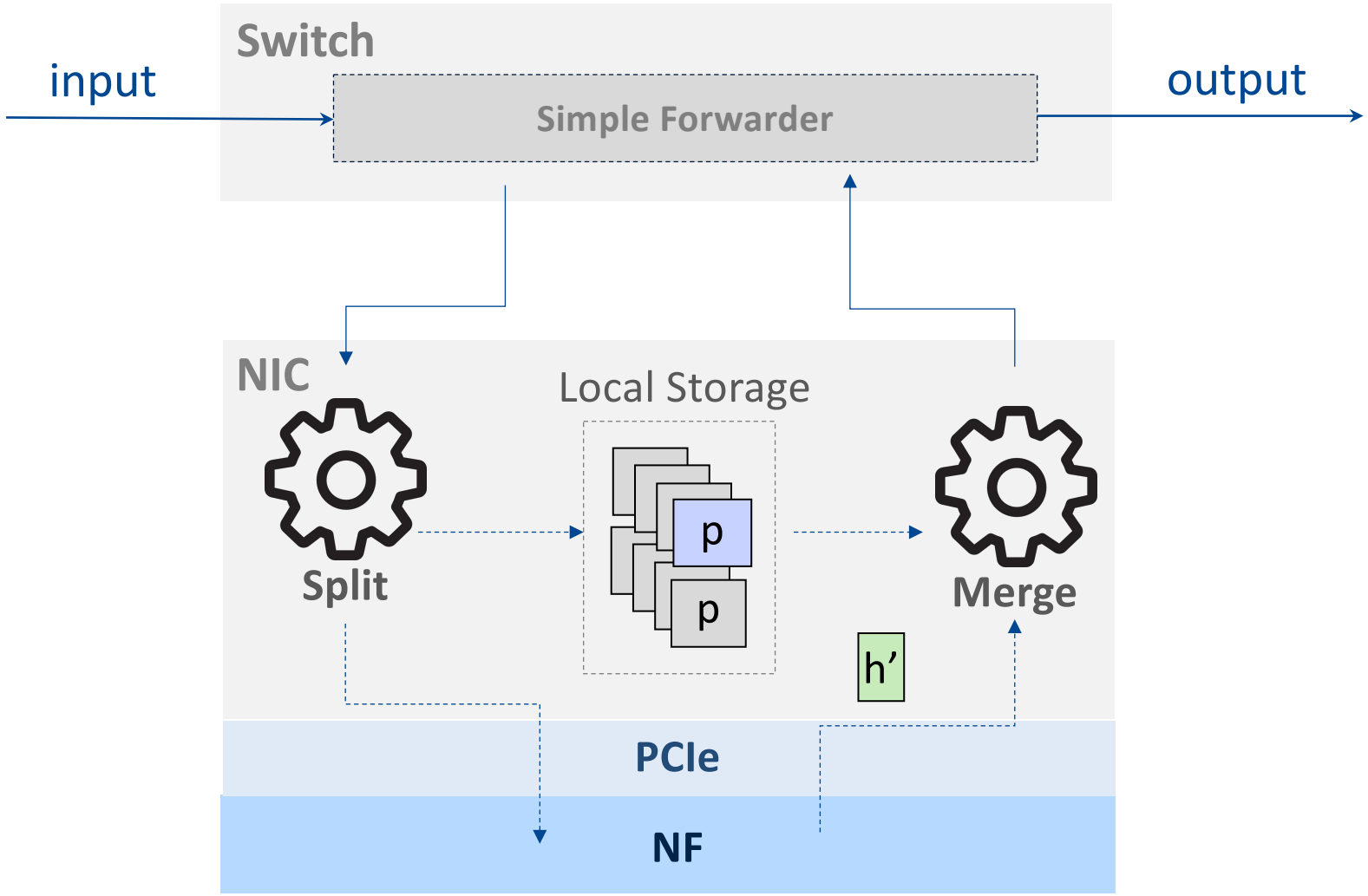
Storing Payloads in the On-NIC Memory (nicmem)



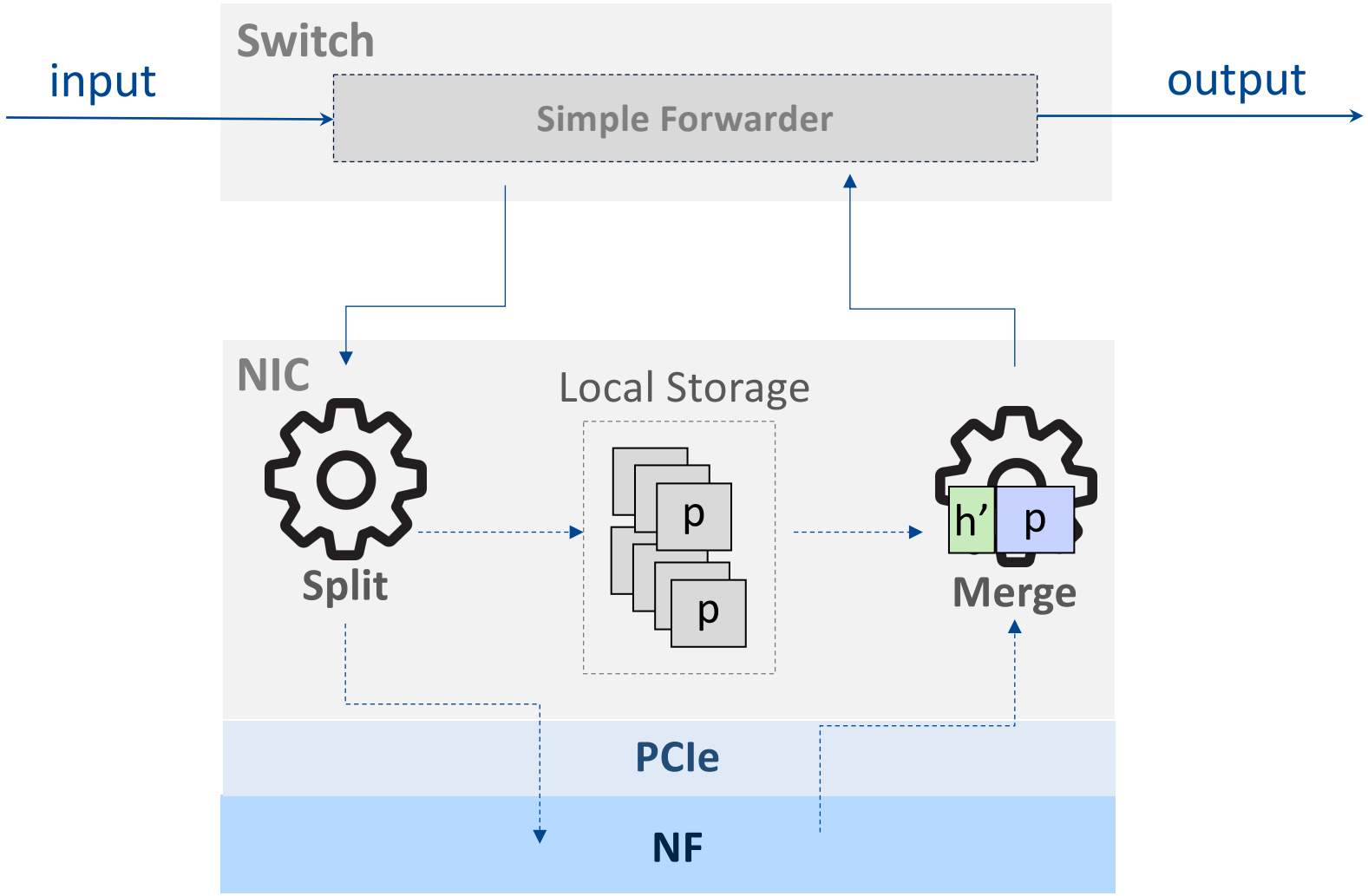
Storing Payloads in the On-NIC Memory (nicmem)



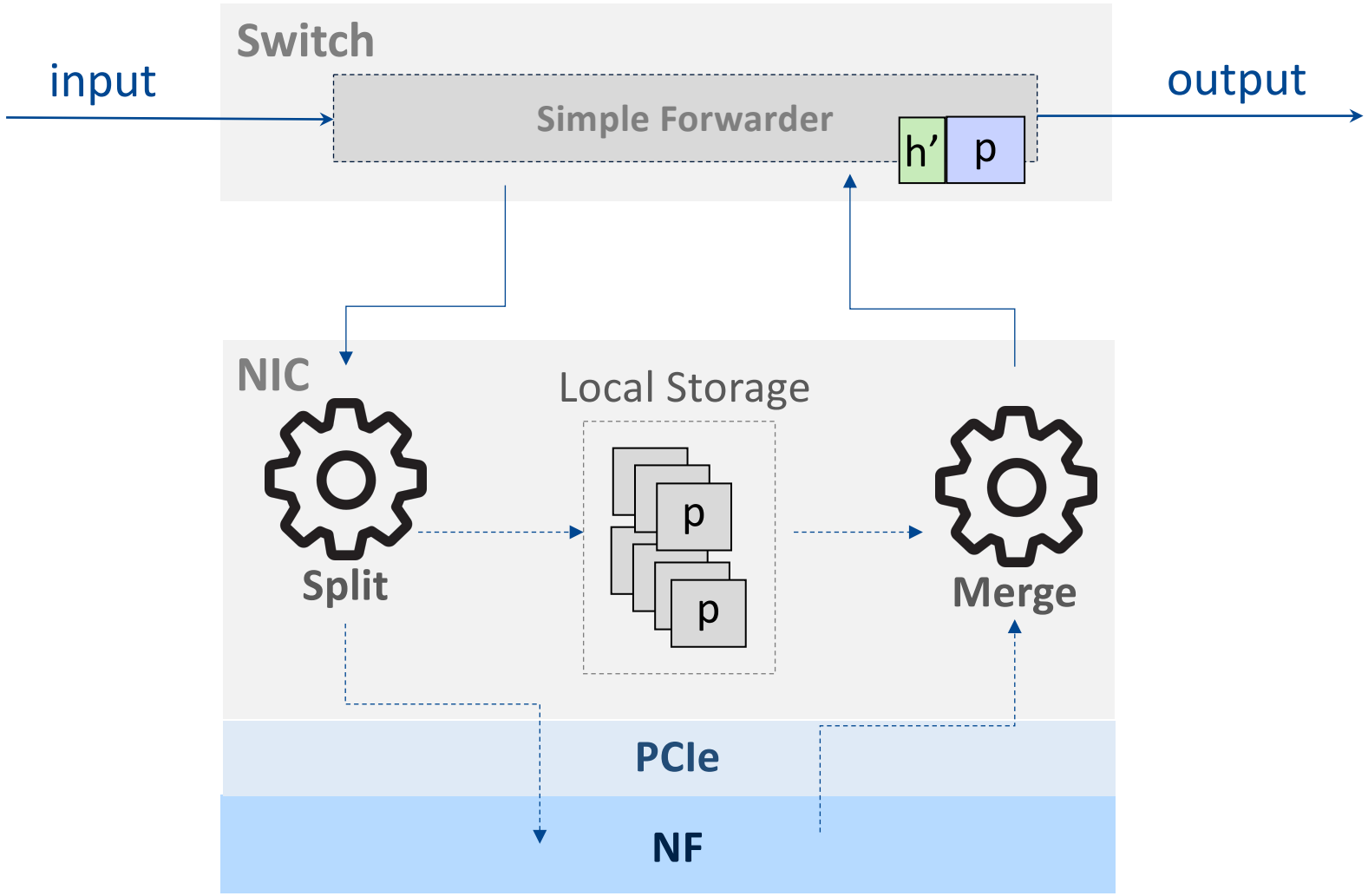
Storing Payloads in the On-NIC Memory (nicmem)



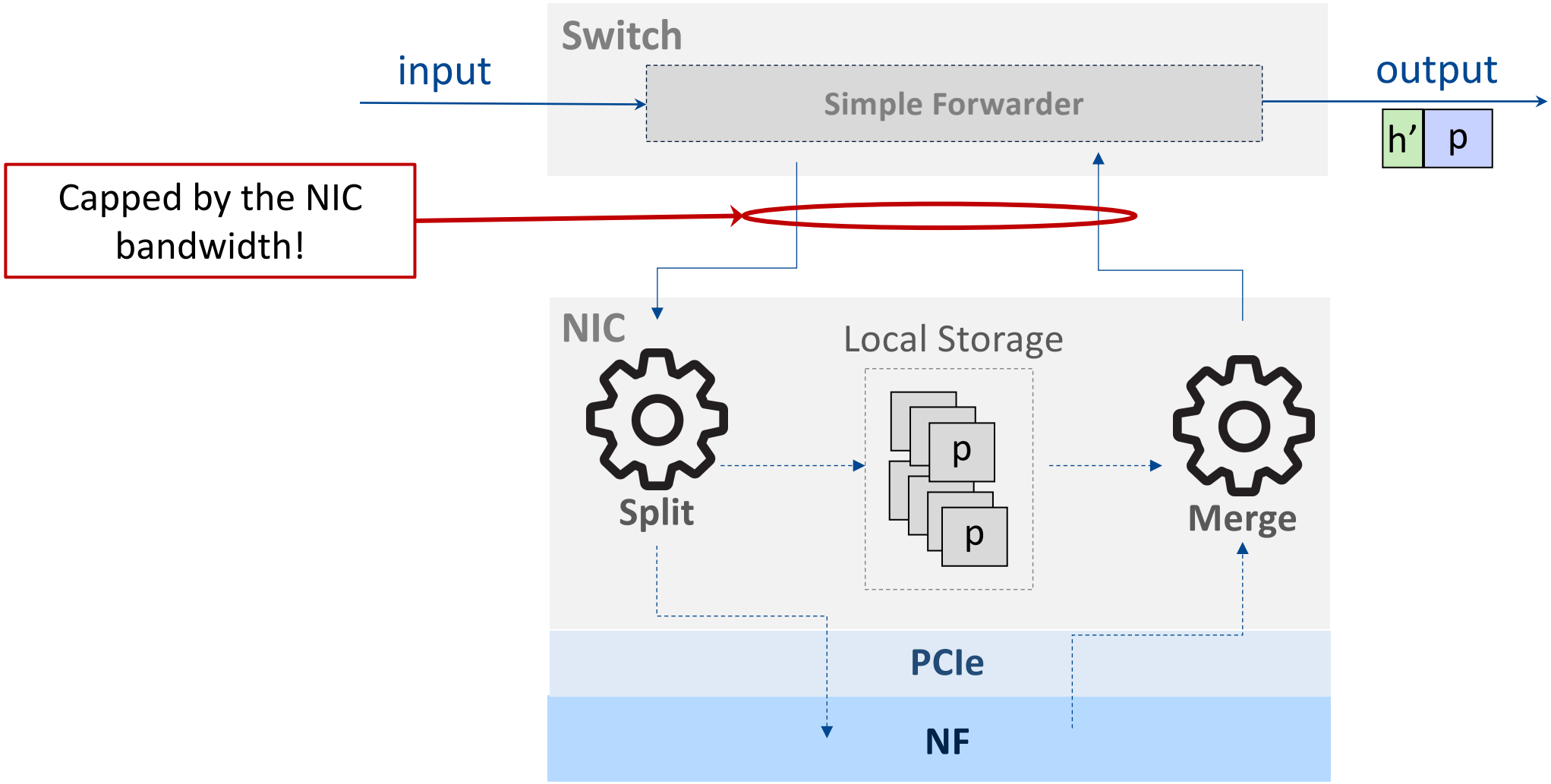
Storing Payloads in the On-NIC Memory (nicmem)



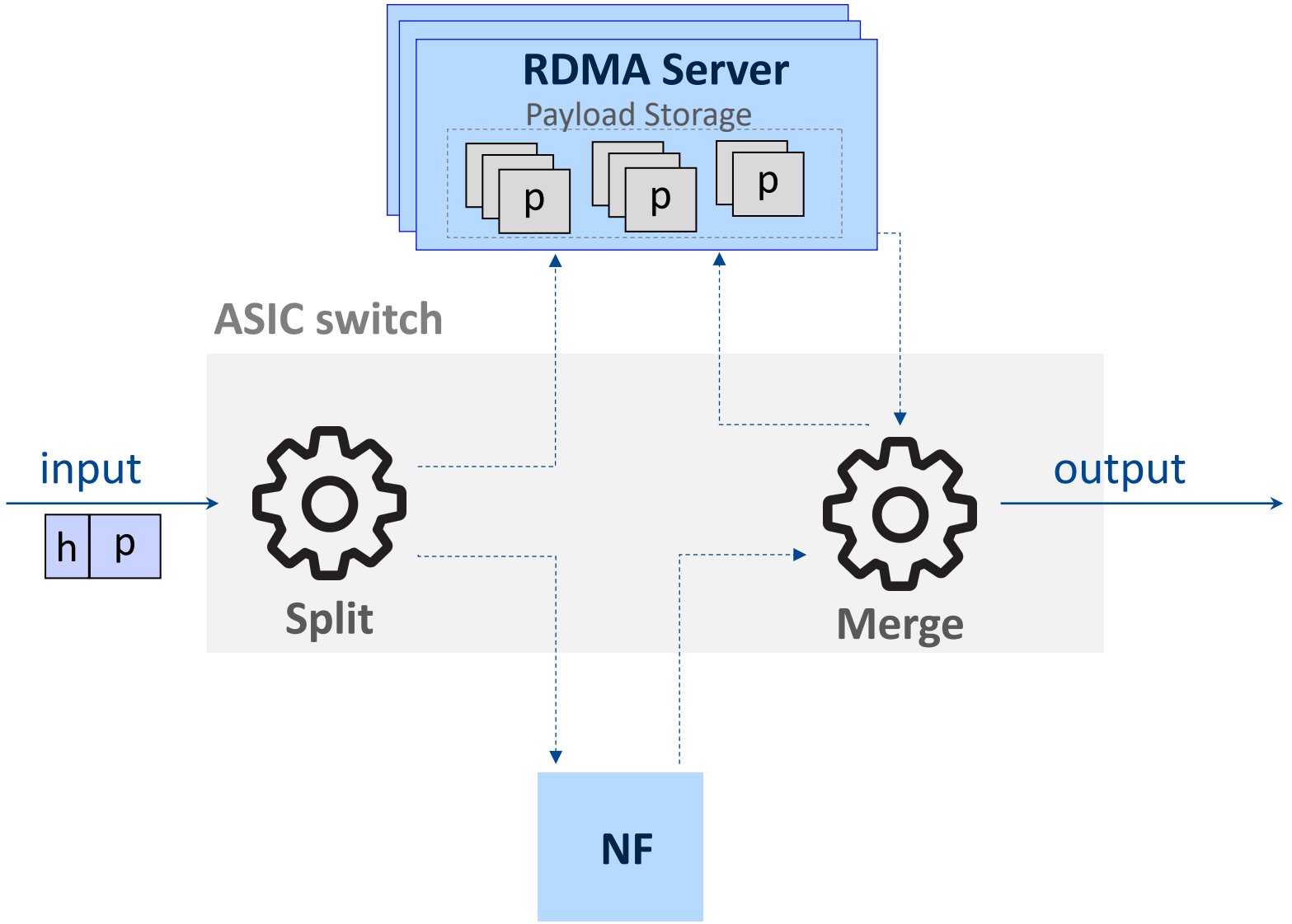
Storing Payloads in the On-NIC Memory (nicmem)



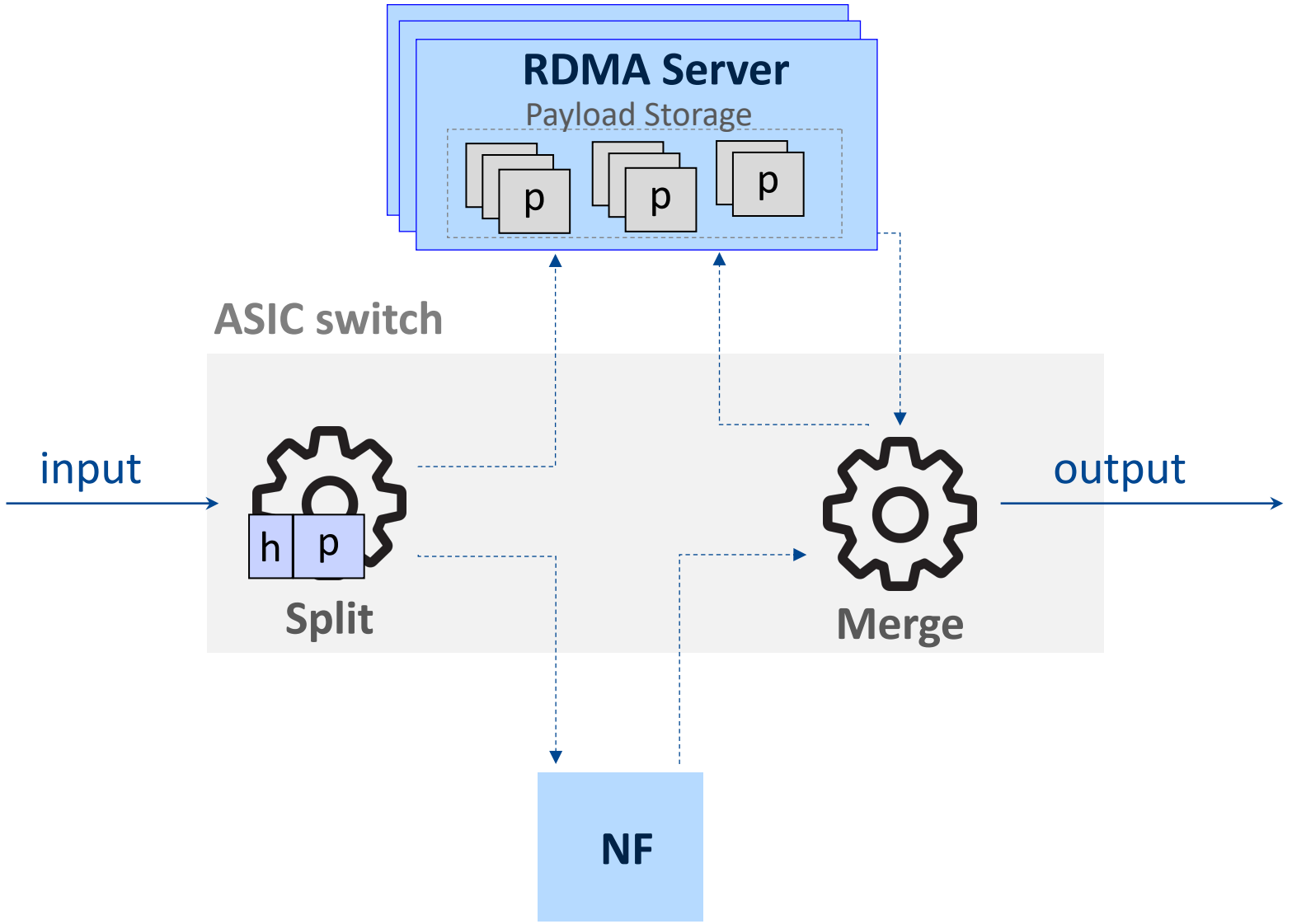
Storing Payloads in the On-NIC Memory (nicmem)



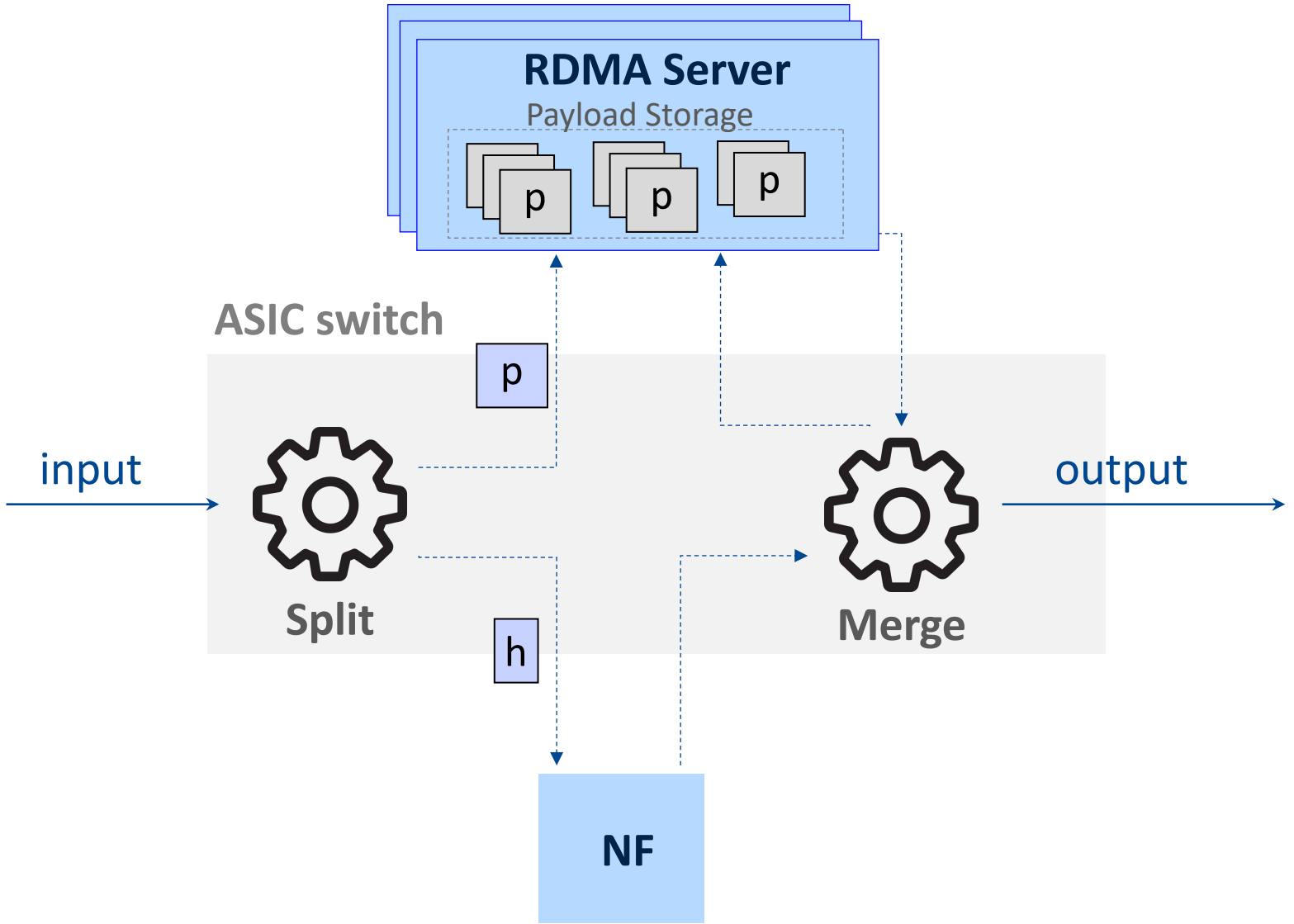
Storing Payloads on Shared Resources (Ribosome)



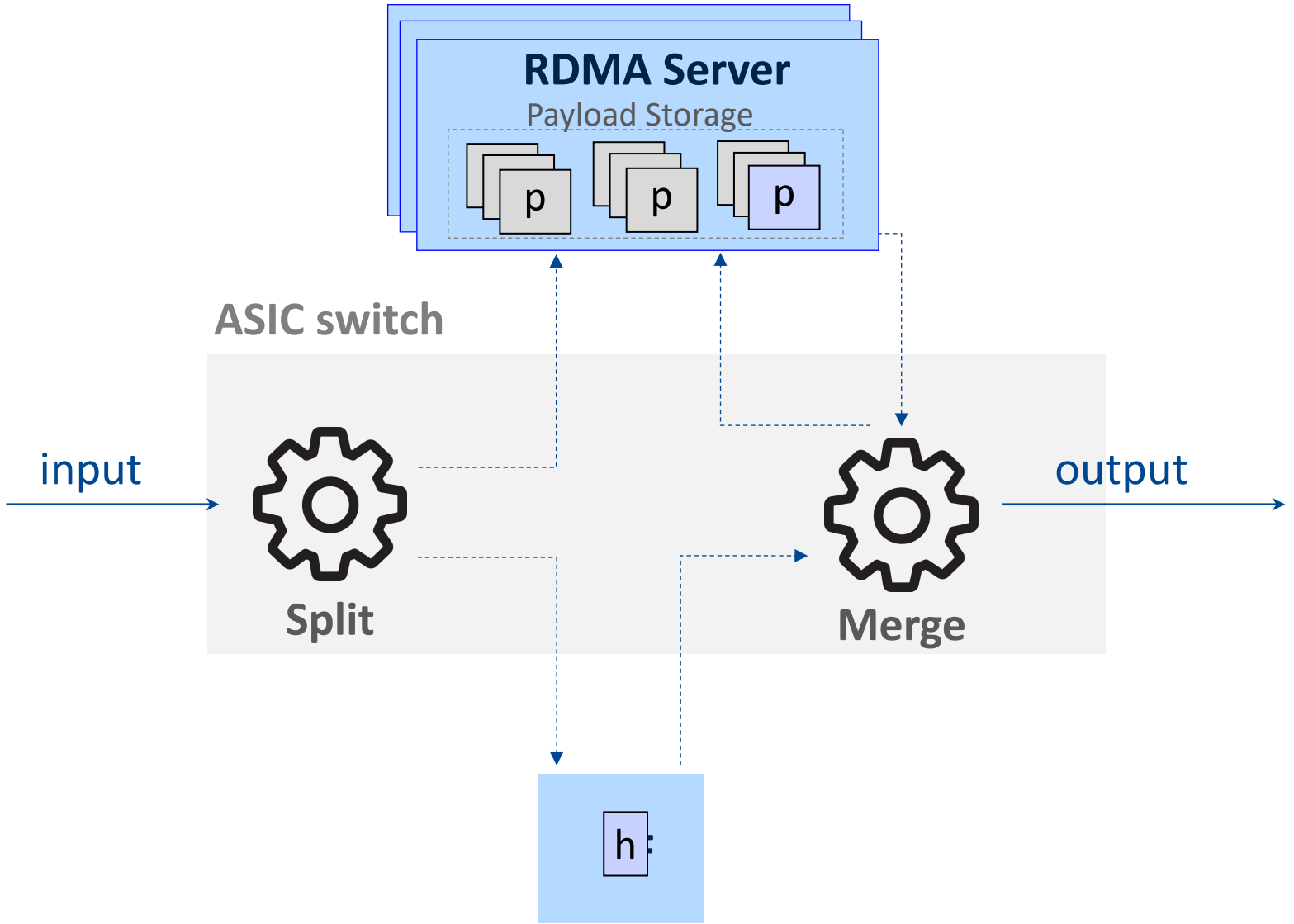
Storing Payloads on Shared Resources (Ribosome)



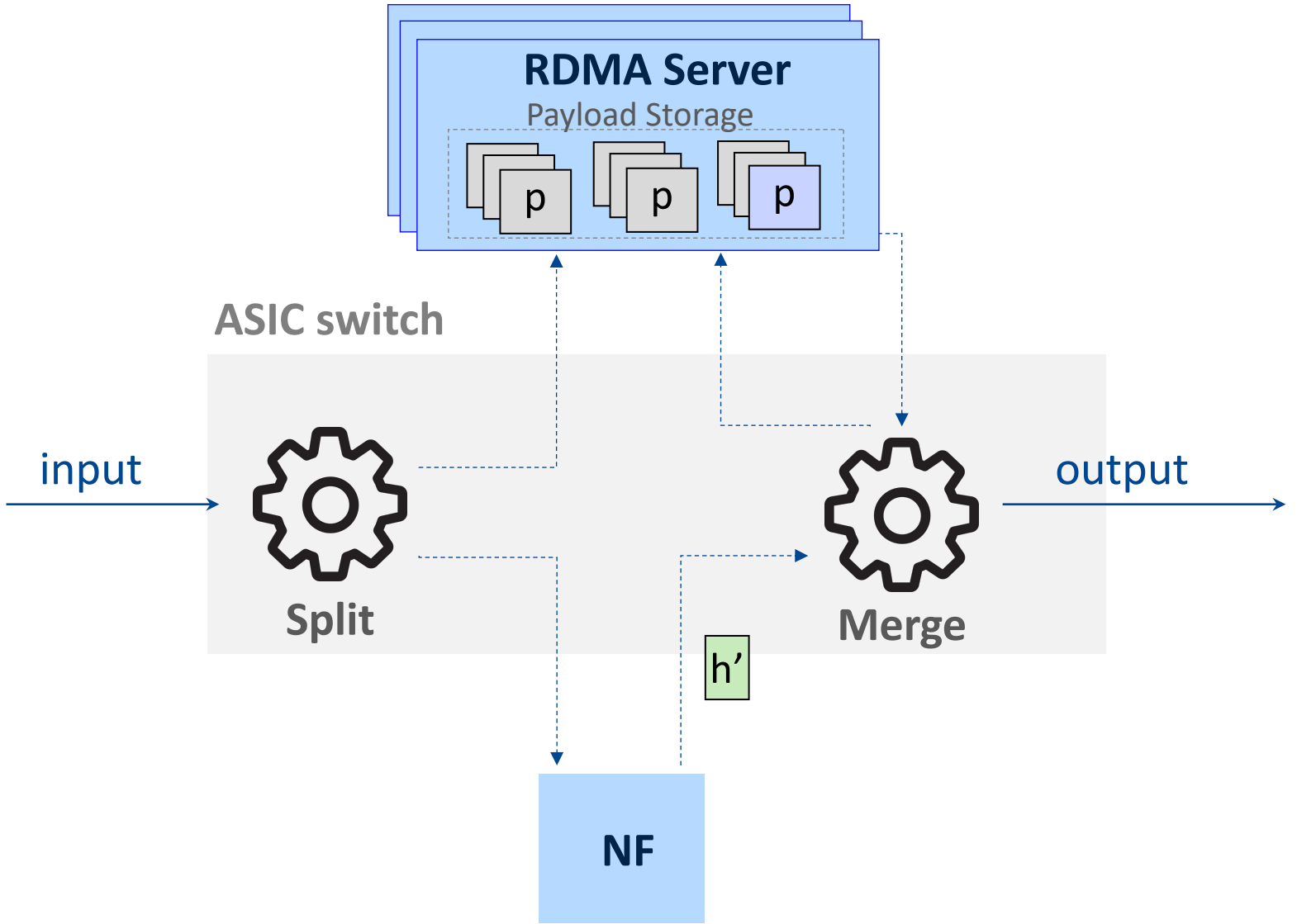
Storing Payloads on Shared Resources (Ribosome)



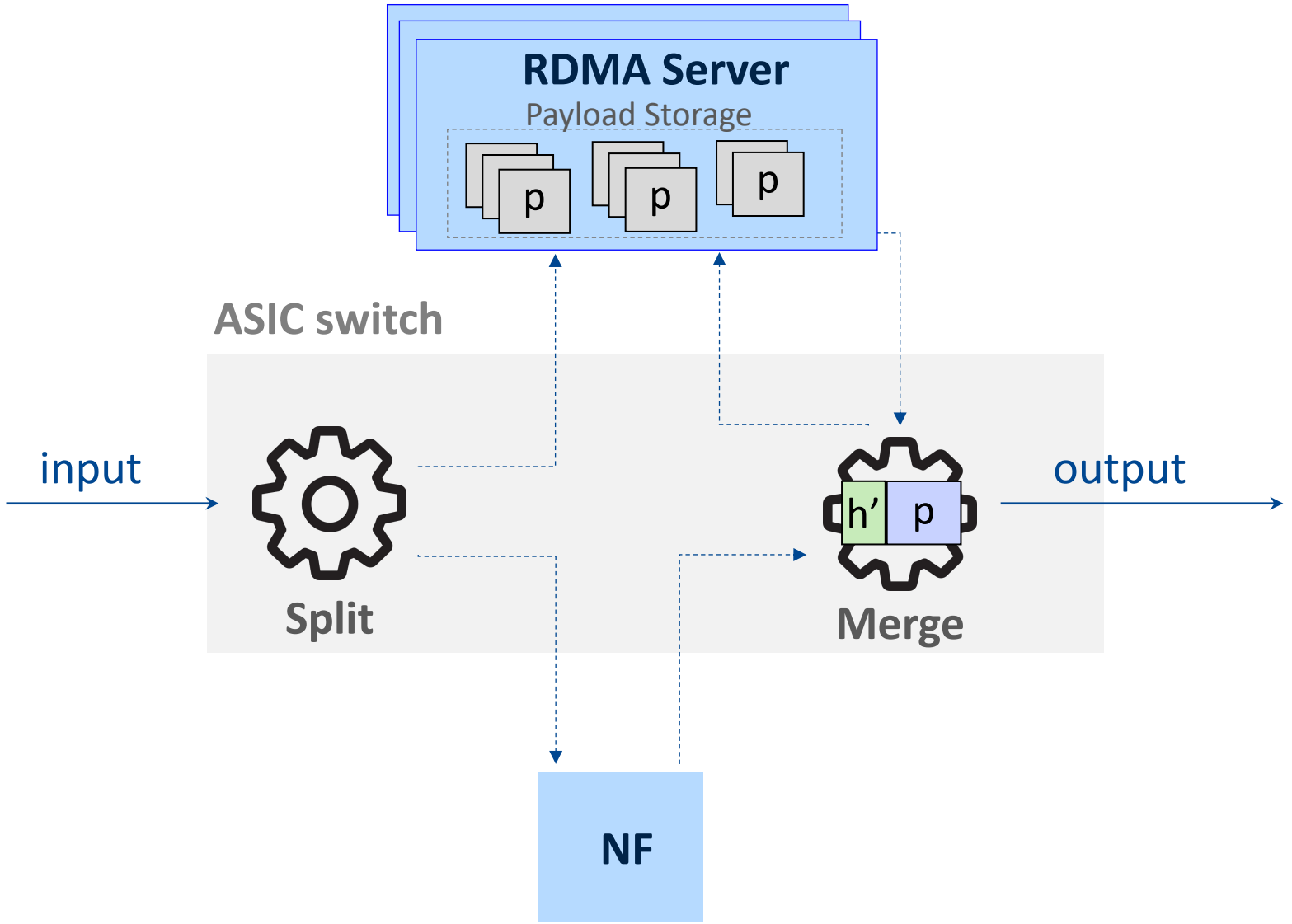
Storing Payloads on Shared Resources (Ribosome)



Storing Payloads on Shared Resources (Ribosome)

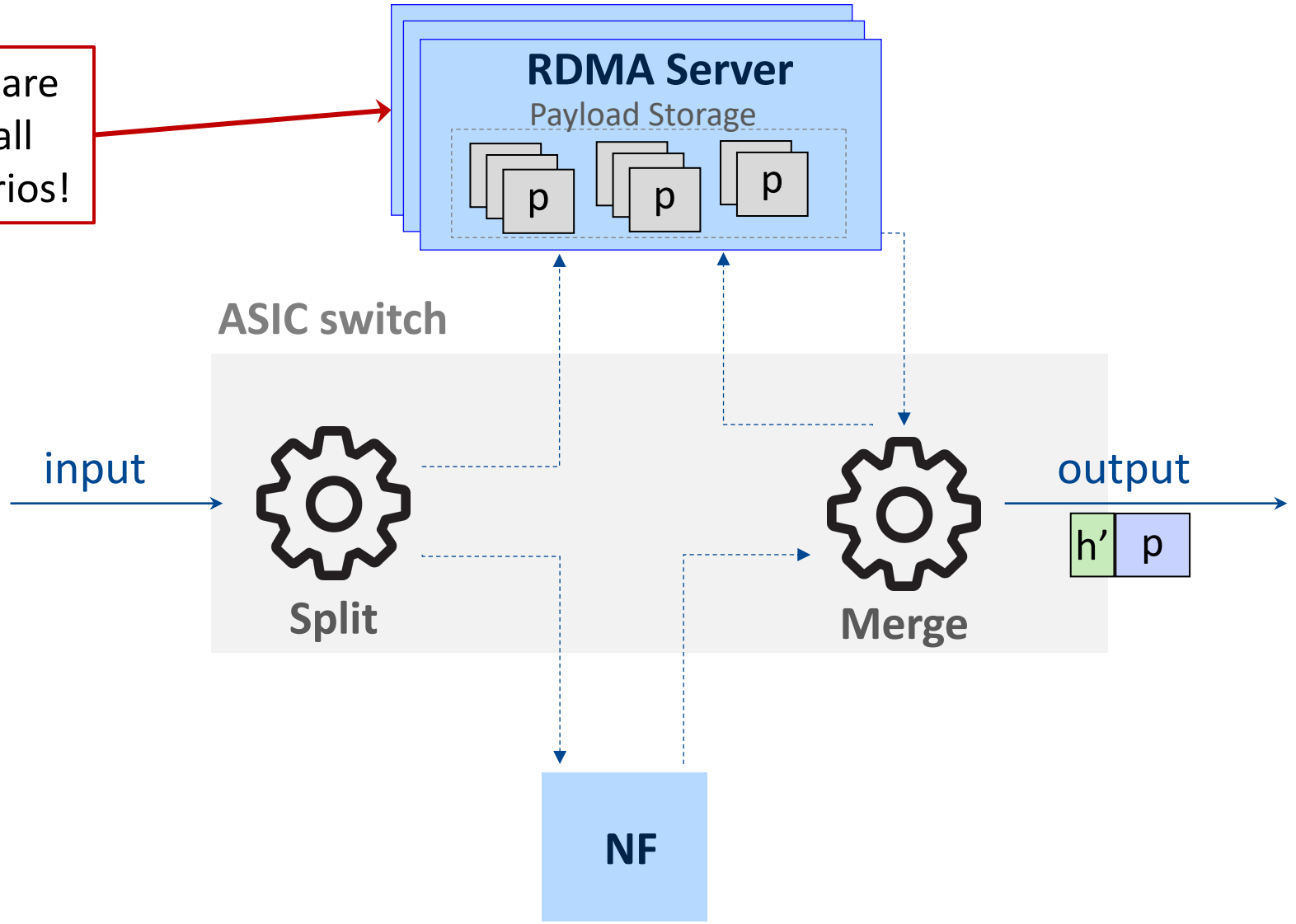


Storing Payloads on Shared Resources (Ribosome)

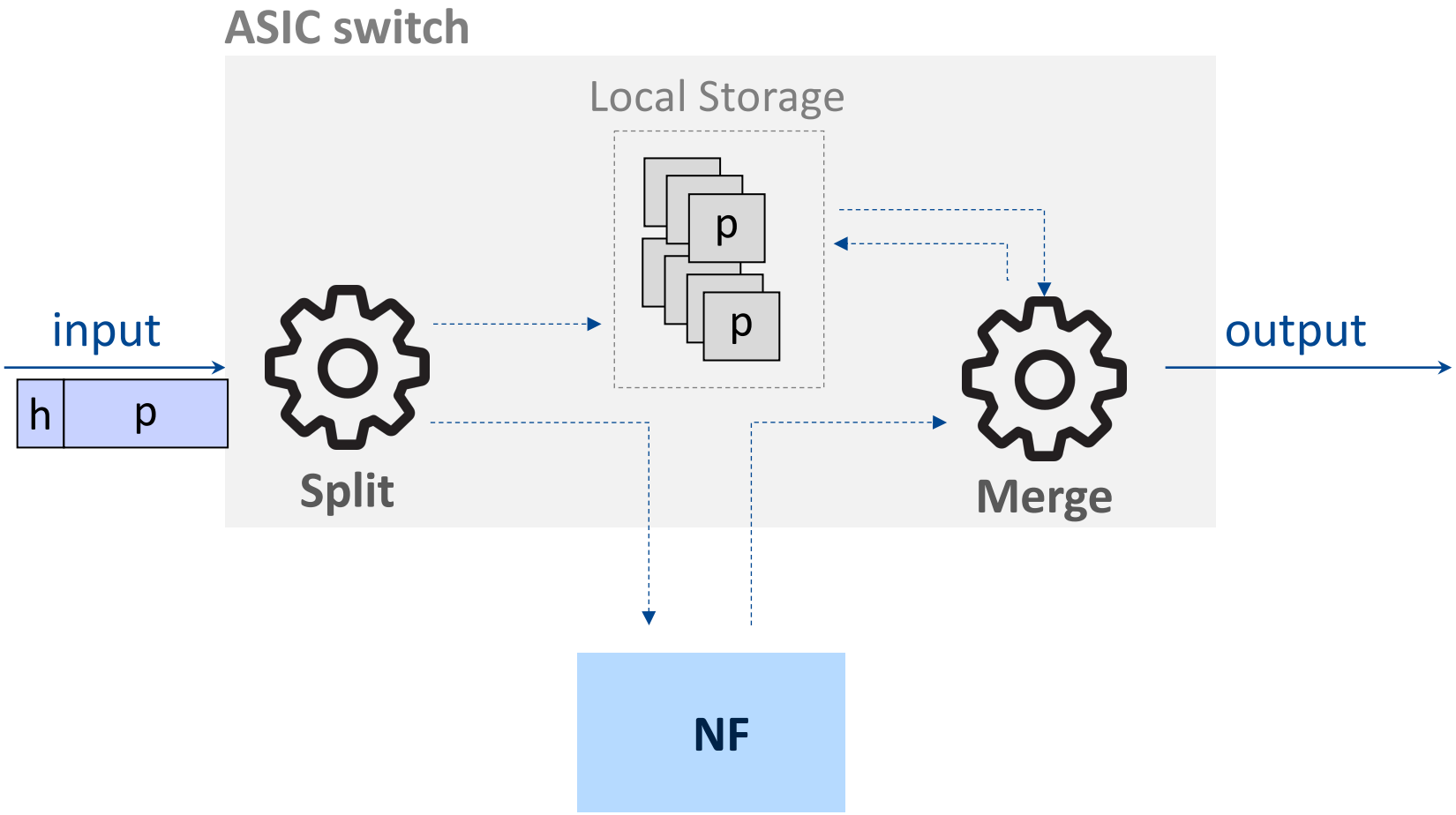


Storing Payloads on Shared Resources (Ribosome)

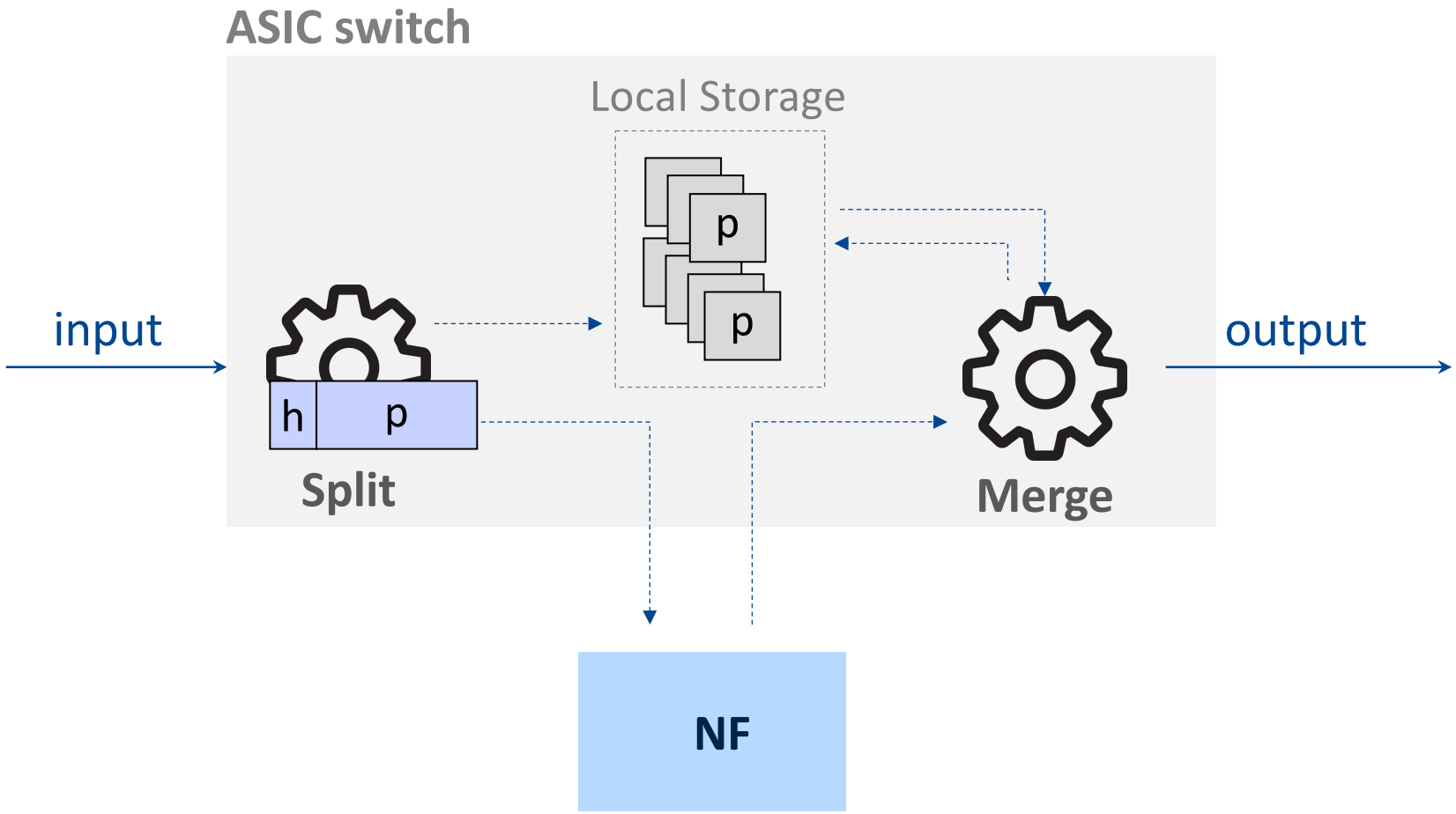
Shared resources are not available in all operational scenarios!



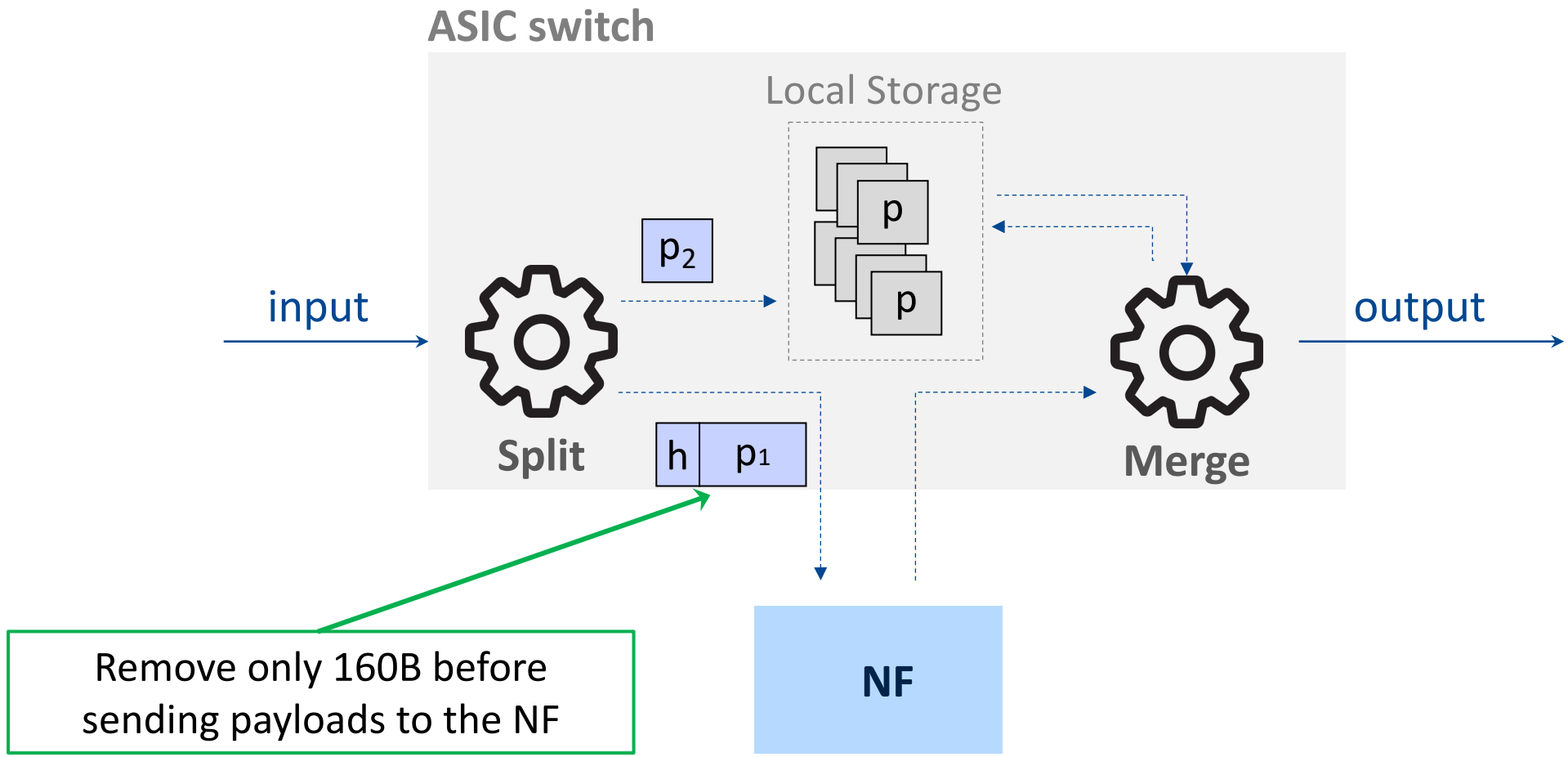
Storing a Few Bytes in ASIC SRAM (PayloadPark)



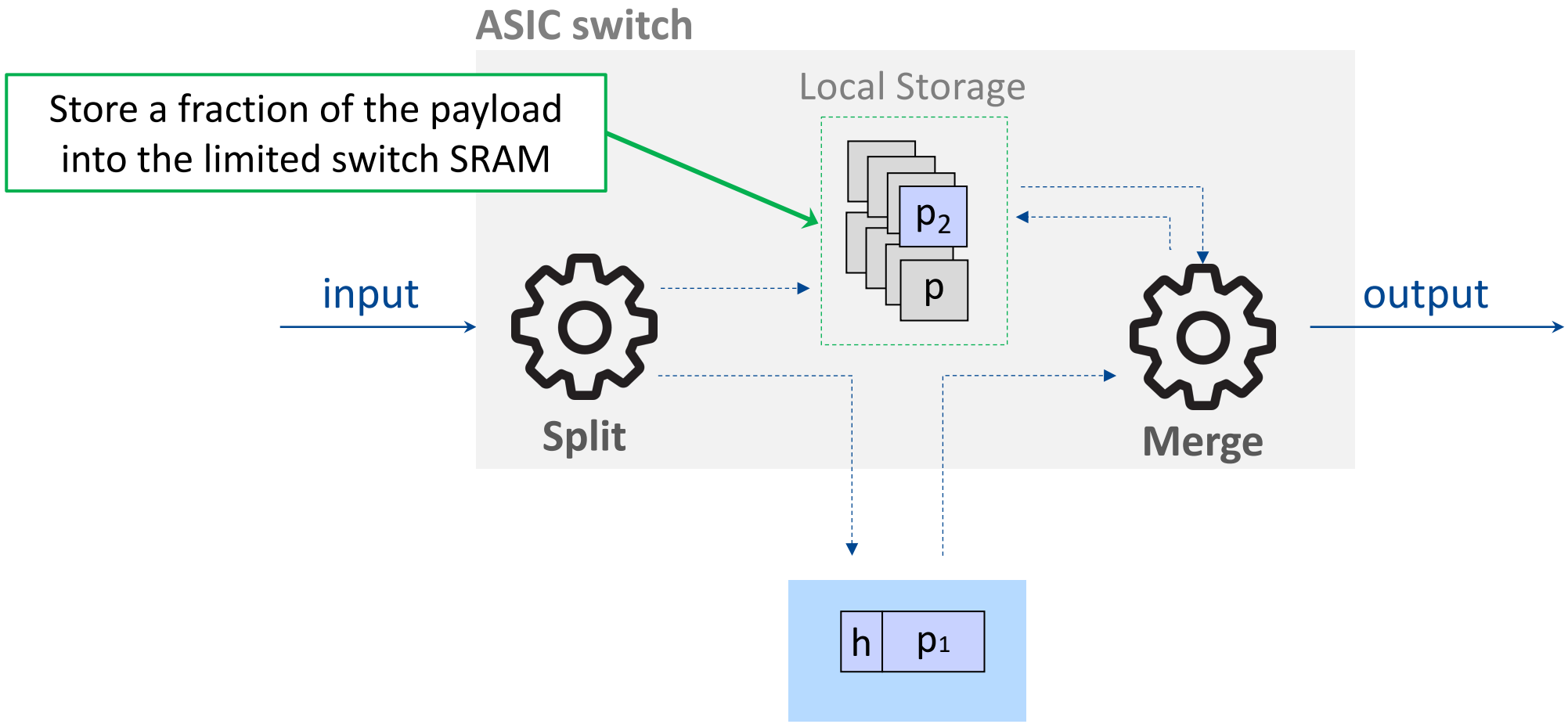
Storing a Few Bytes in ASIC SRAM (PayloadPark)



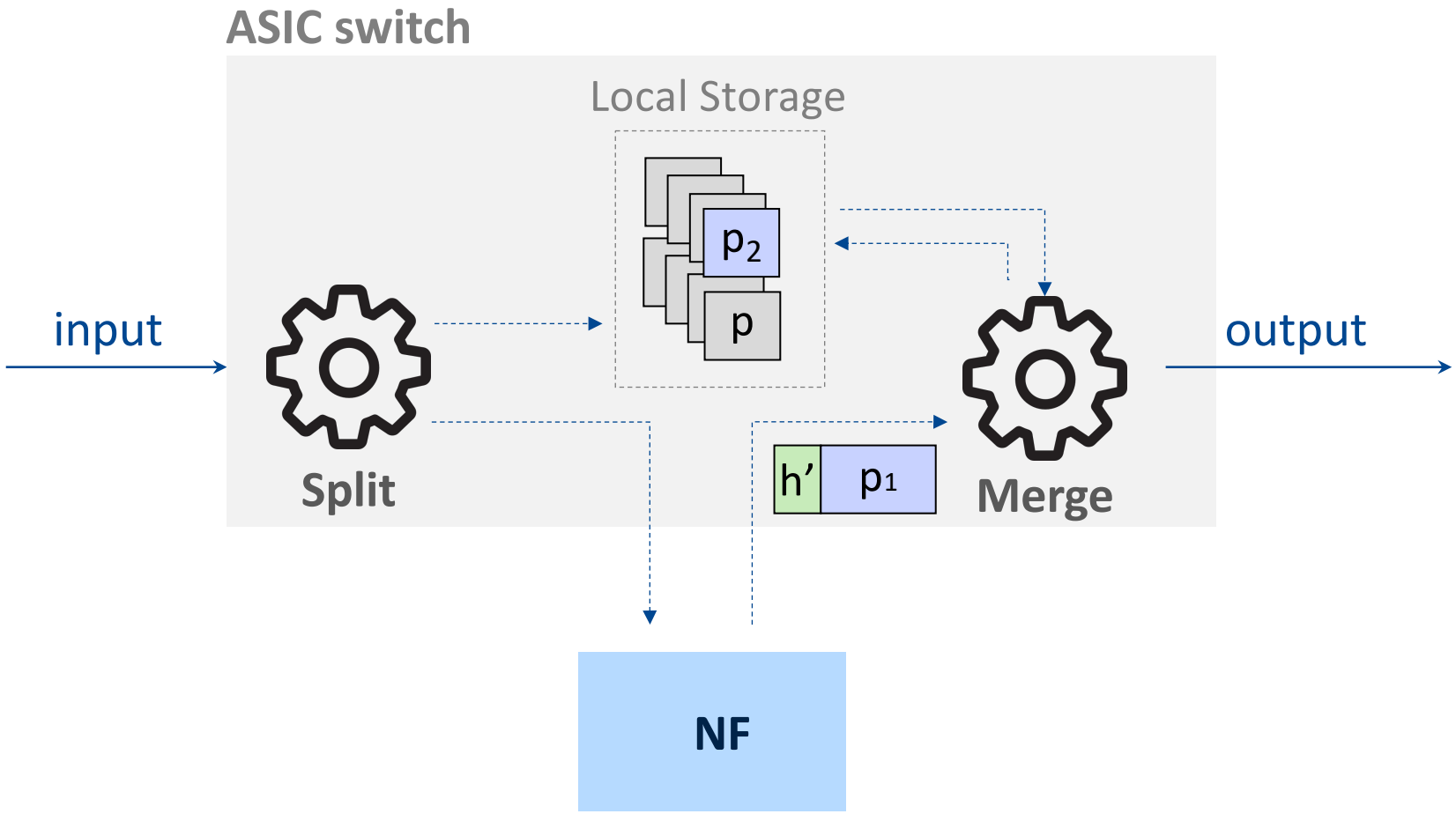
Storing a Few Bytes in ASIC SRAM (PayloadPark)



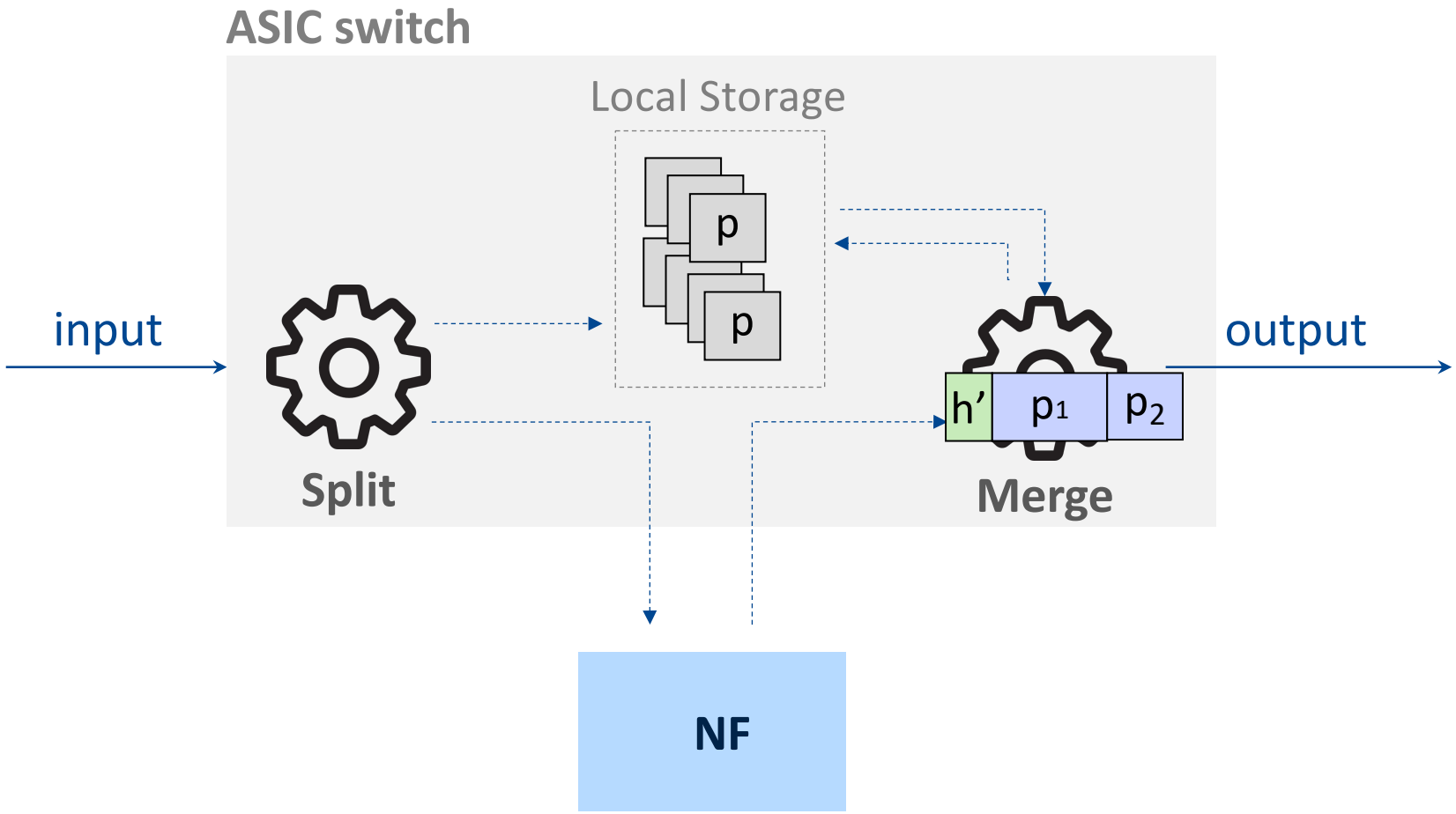
Storing a Few Bytes in ASIC SRAM (PayloadPark)



Storing a Few Bytes in ASIC SRAM (PayloadPark)



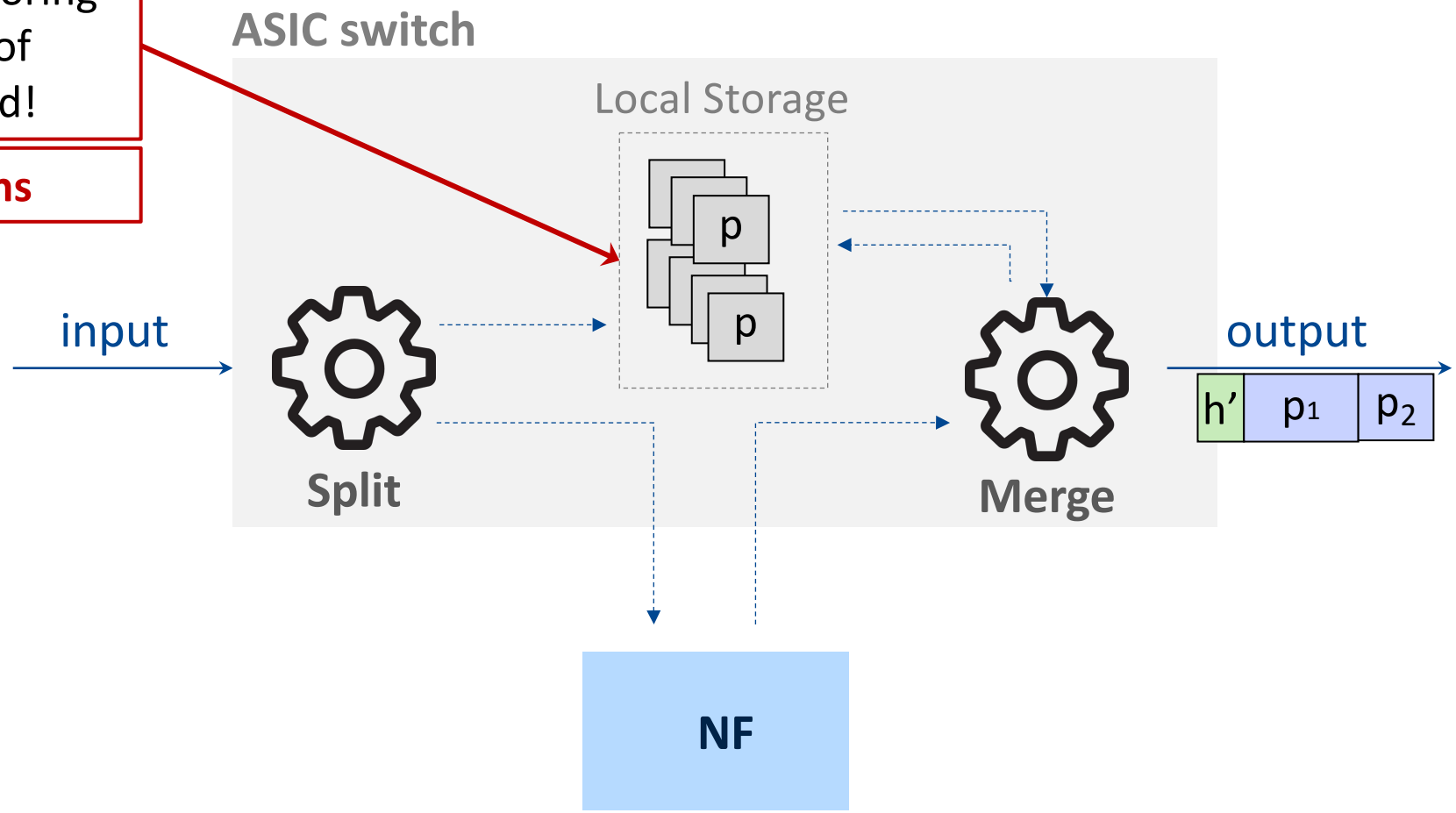
Storing a Few Bytes in ASIC SRAM (PayloadPark)



Storing a Few Bytes in ASIC SRAM (PayloadPark)

SRAM allows storing only a part of each payload!

Limited gains



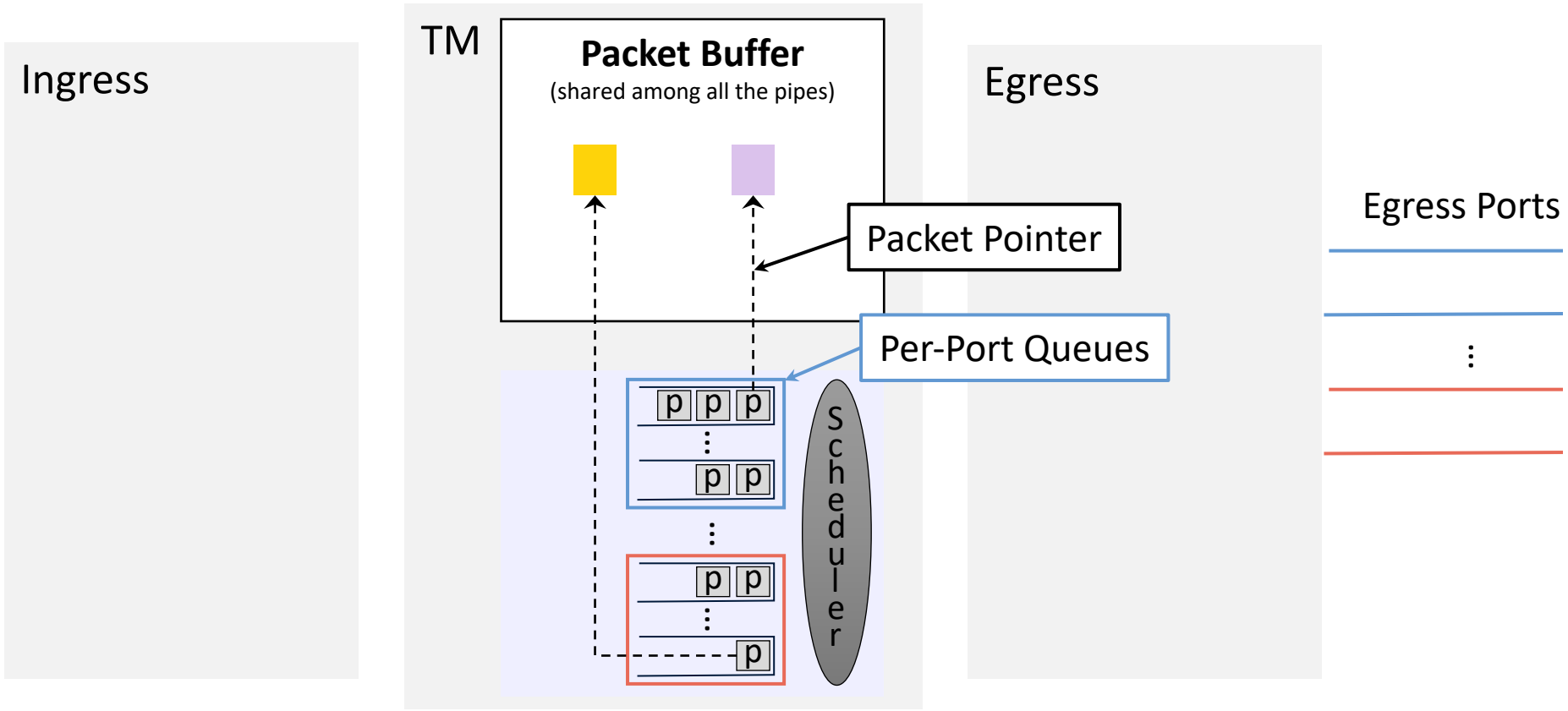
Can we store the entire payload within the switch without requiring external devices?

Core Idea: A Queue-Based Packet Storage

Exploit the switch shared buffer to store payloads

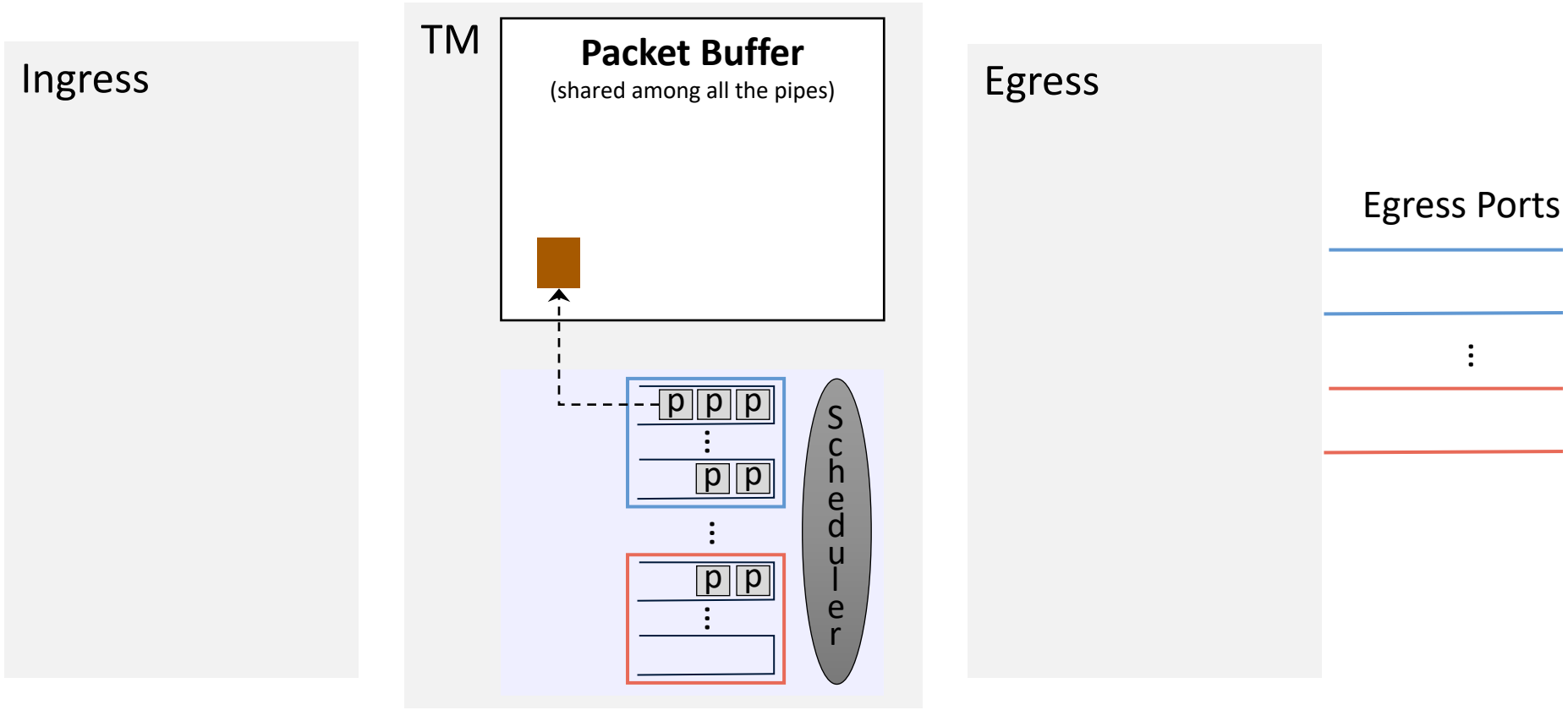
Core Idea: A Queue-Based Packet Storage

Exploit the switch shared buffer to store payloads



Core Idea: A Queue-Based Packet Storage

Exploit the switch shared buffer to store payloads



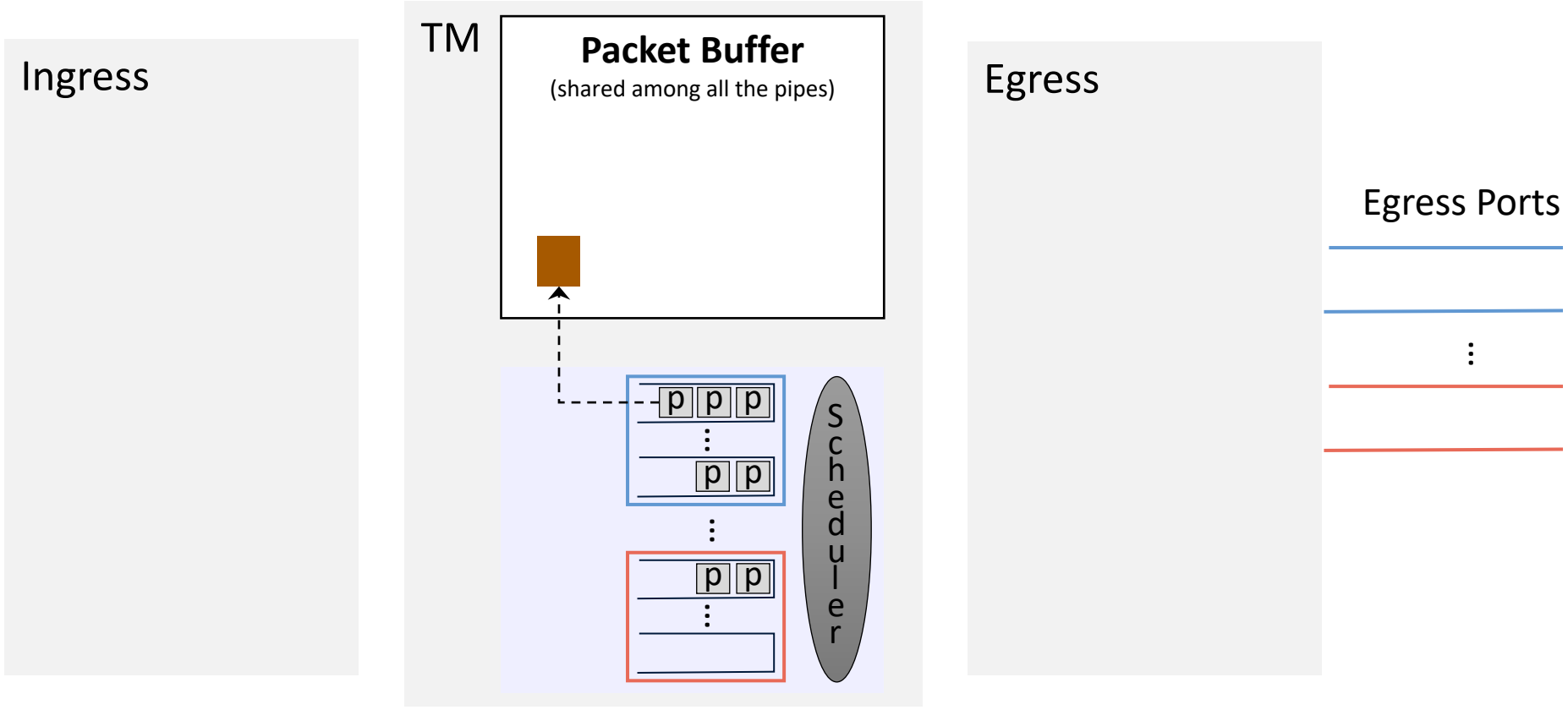
Core Idea: A Queue-Based Packet Storage

Exploit the switch shared buffer to store payloads

✓ Large shared buffer

✗ No programmatic access

? How can we access the packet buffer?



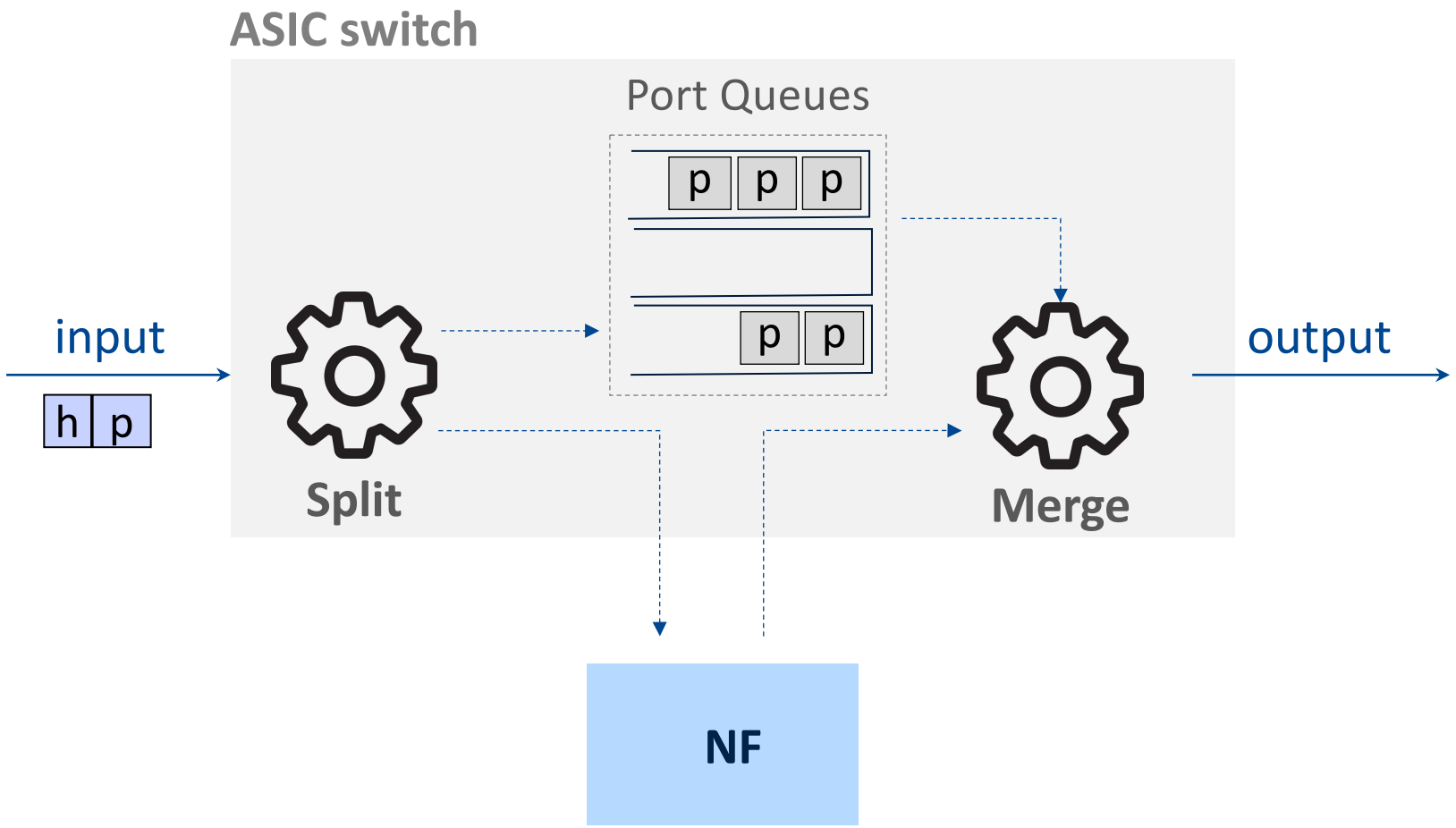
How can we access the packet buffer?

Enqueued packets **implicitly** control the buffer memory

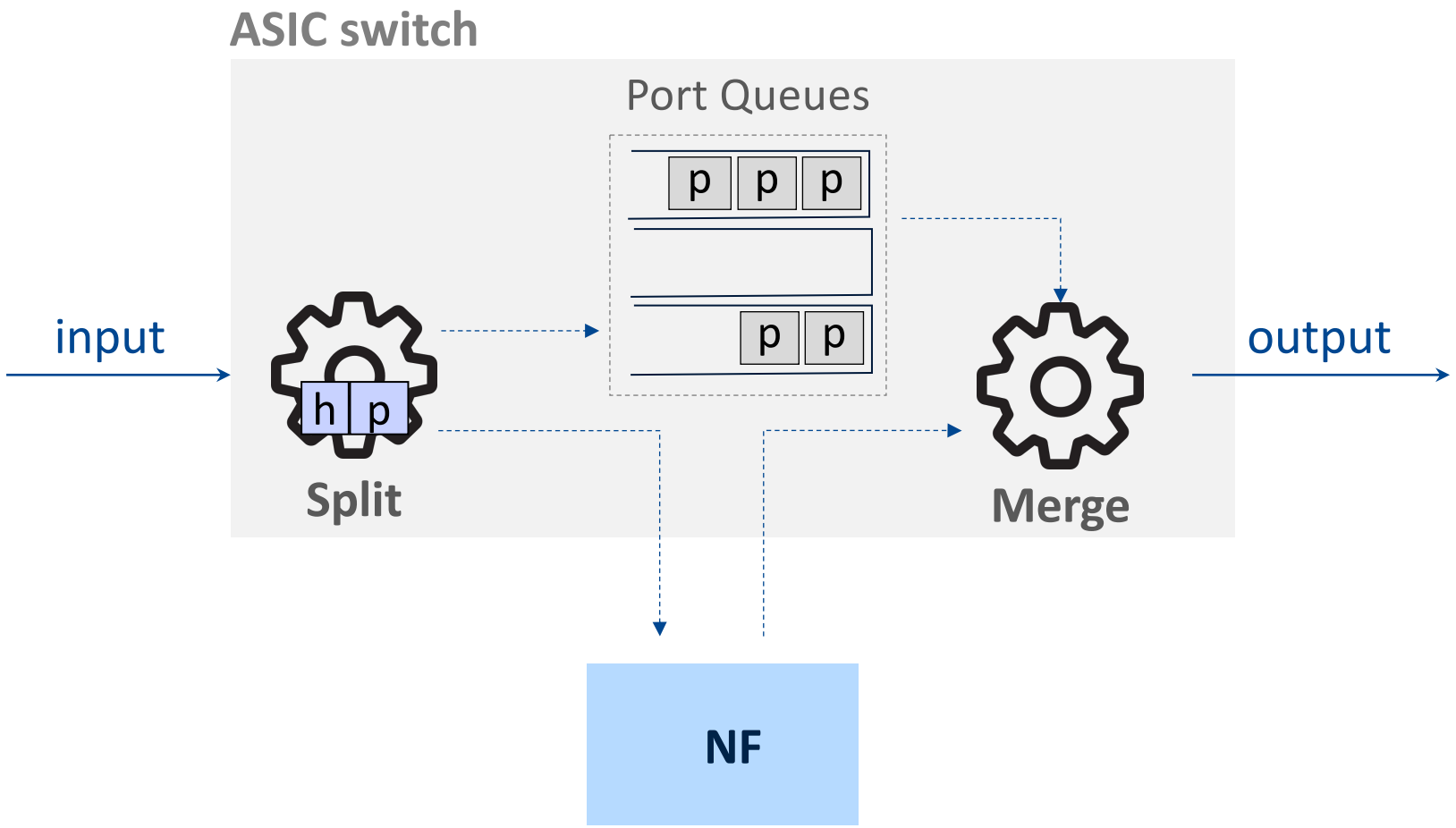
Hold payloads in the port queues until
the processed header from the NF is returned!

Would it work for real?

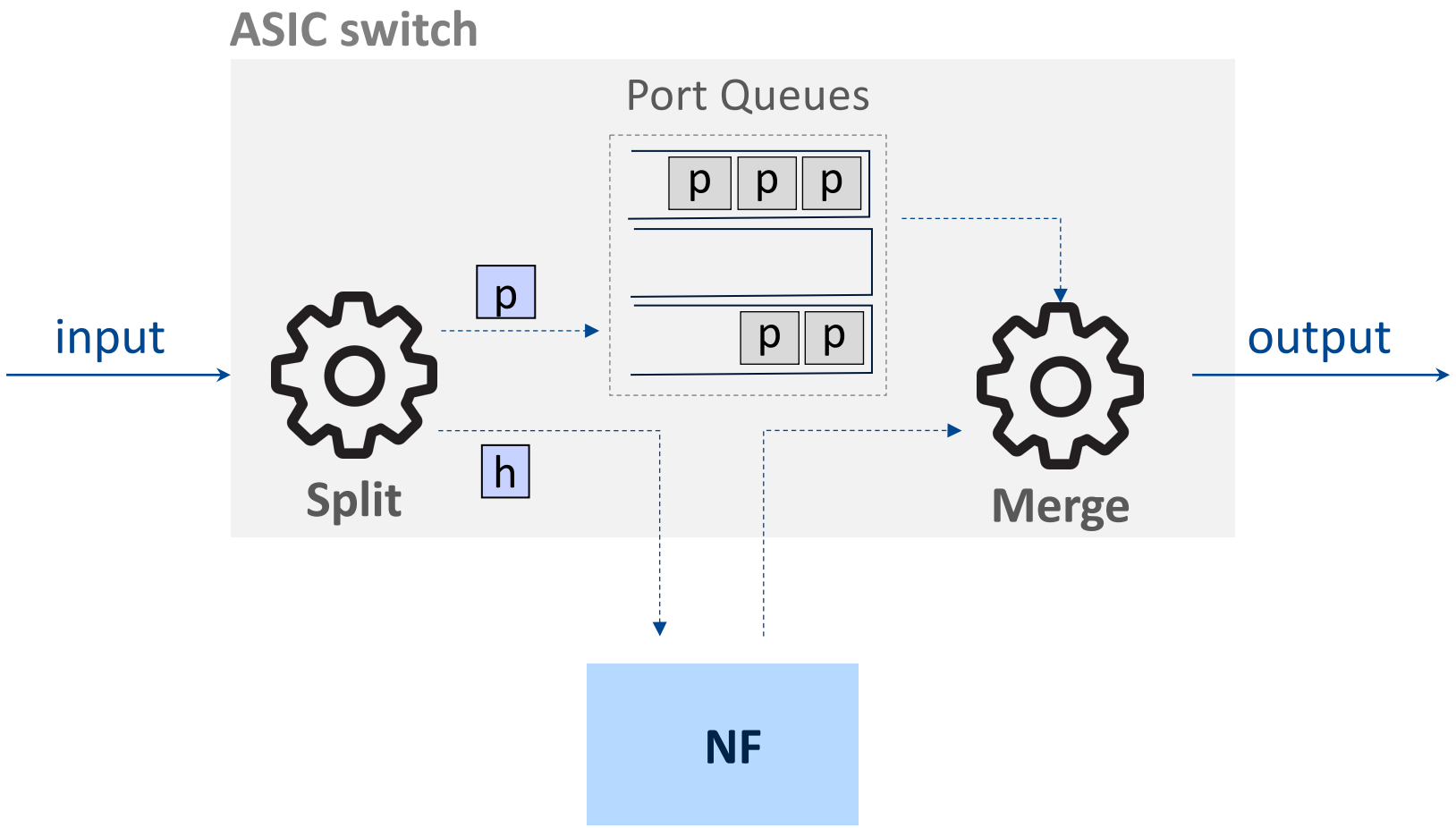
Challenges


ASIC switch
Port Queues
p
p
p
p
p
input
h
p
Split
Merge
output
NF

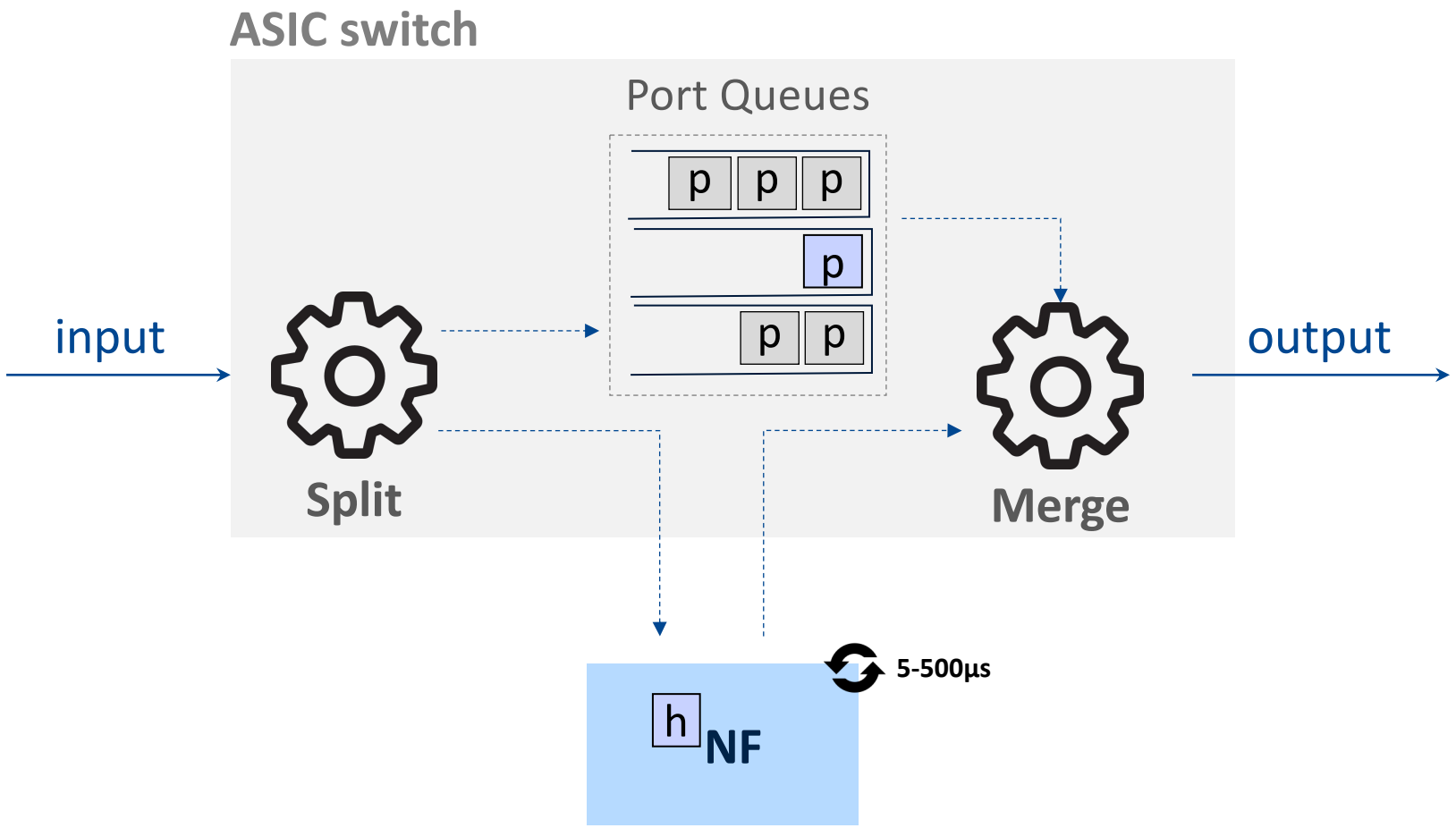
Challenges


ASIC switch
Port Queues
p
p
p
p
p
input
h
p
Split
Merge
output
NF

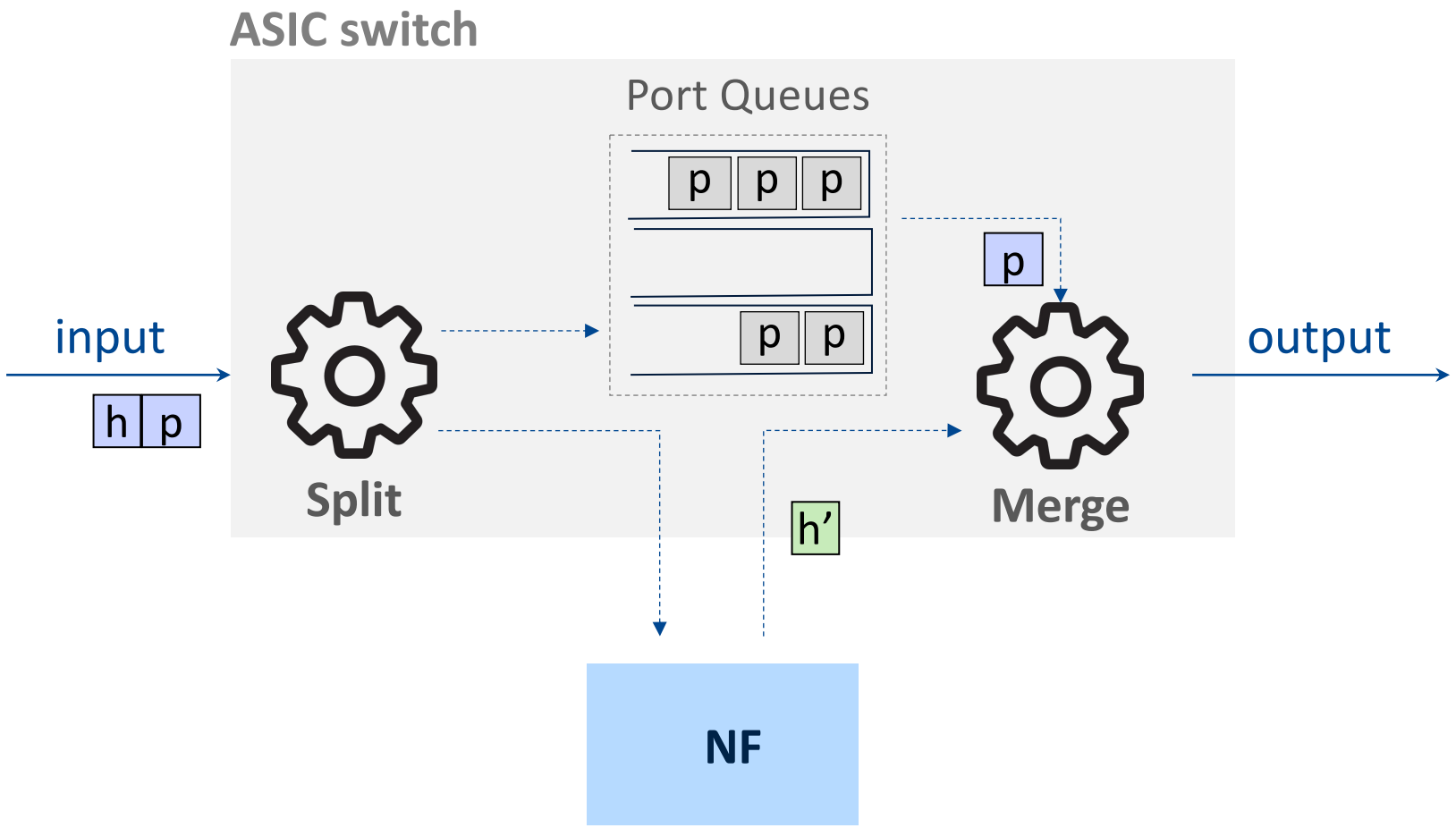
Challenges


ASIC switch
Port Queues
p
p
p
p
p
input
p
output
h
Split
Merge
NF

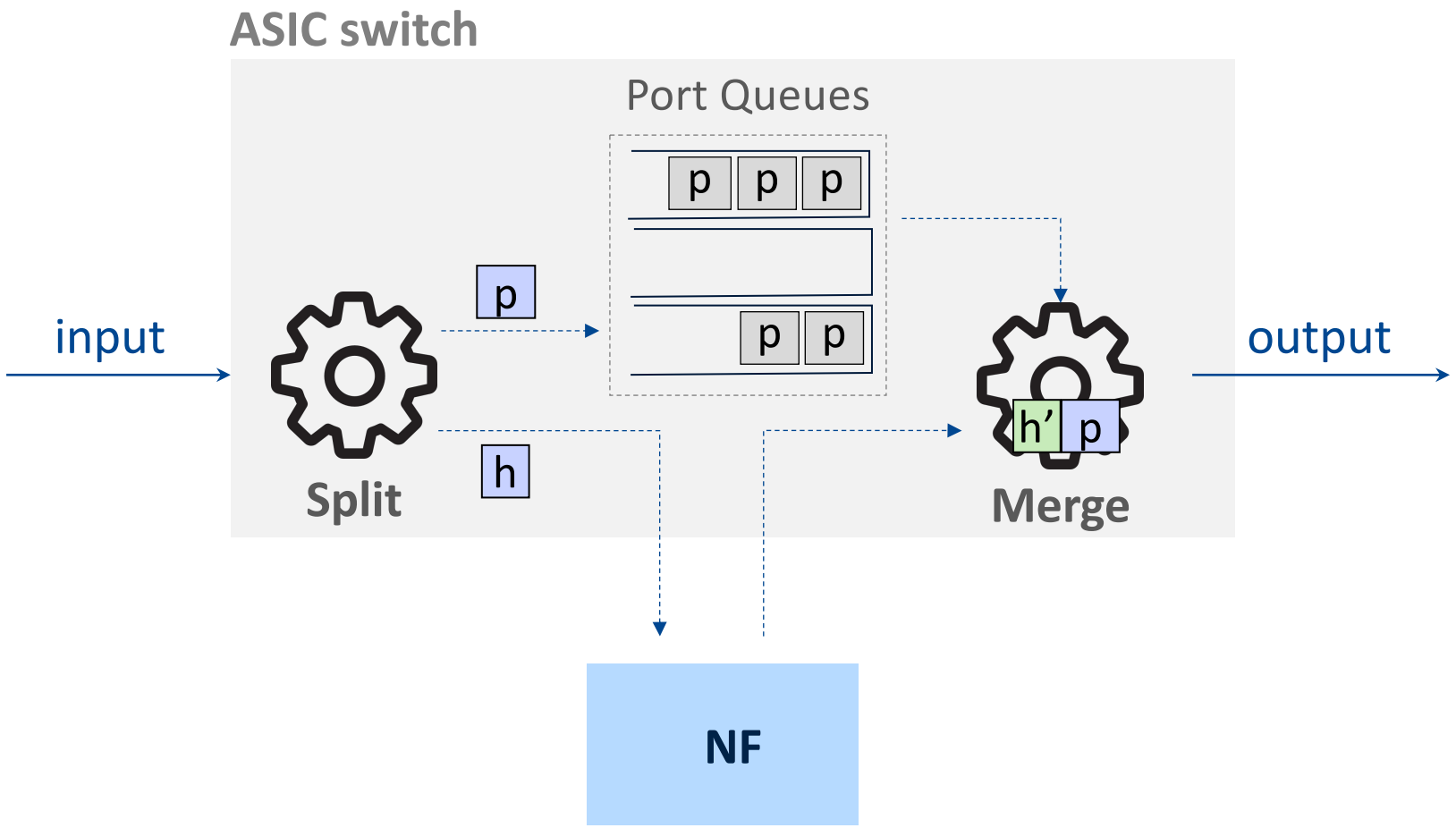
Challenges


ASIC switch
Port Queues
p
p
p
p
p
p
input
output
Split
Merge
h
NF
5-500µs

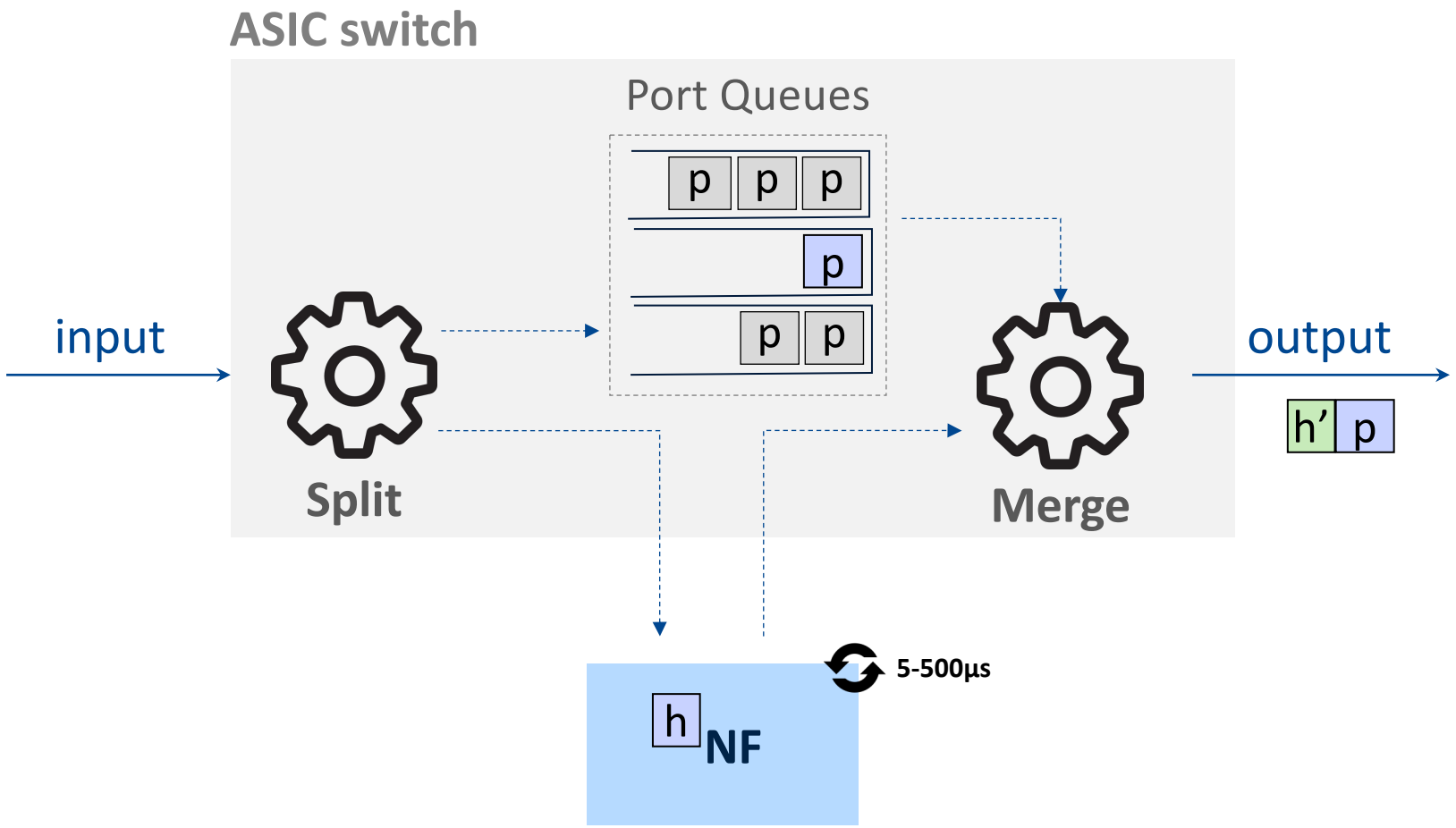
Challenges


ASIC switch
Port Queues
p p p
p p
p
input
h p
output
Split
Merge
h'
NF

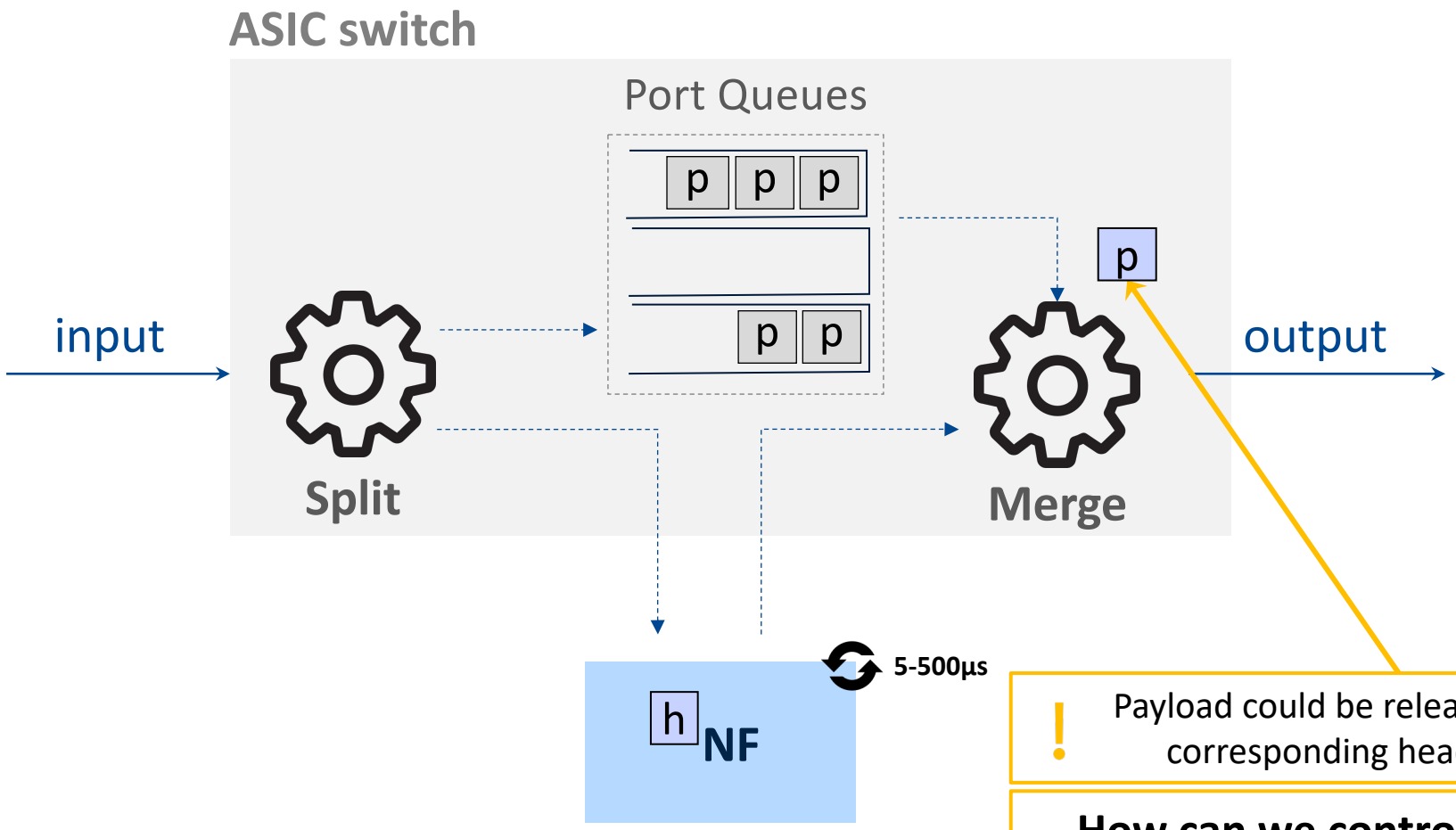
Challenges


ASIC switch
Port Queues
p
p
p
p
p
input
p
output
h'
p
h
Split
Merge
NF

Challenges


ASIC switch
Port Queues
p
p
p
p
p
p
input
output
h'
p
Split
Merge
5-500µs
h
NF

Challenges


ASIC switch
Port Queues
p
p
p
p
p
p
input
output
Split
Merge
h
NF
5-500µs
!
Payload could be released before the corresponding header is back!
How can we control the queues?

How can we control the queues?

Congest the egress queues to delay payloads release by the time required for NF processing

How can we control the queues?

Congest the egress queues to delay payloads release by the time required for NF processing

Recirculate payloads until headers are ready

How can we control the queues?

Congest the egress queues to delay payloads release by the time required for NF processing

Recirculate payloads until headers are ready

Create copies of payloads to fill up the queues

How can we control the queues?

Congest the egress queues to delay payloads release by the time required for NF processing

Recirculate payloads until headers are ready

Create copies of payloads to fill up the queues

TurboSwitch

TurboSwitch

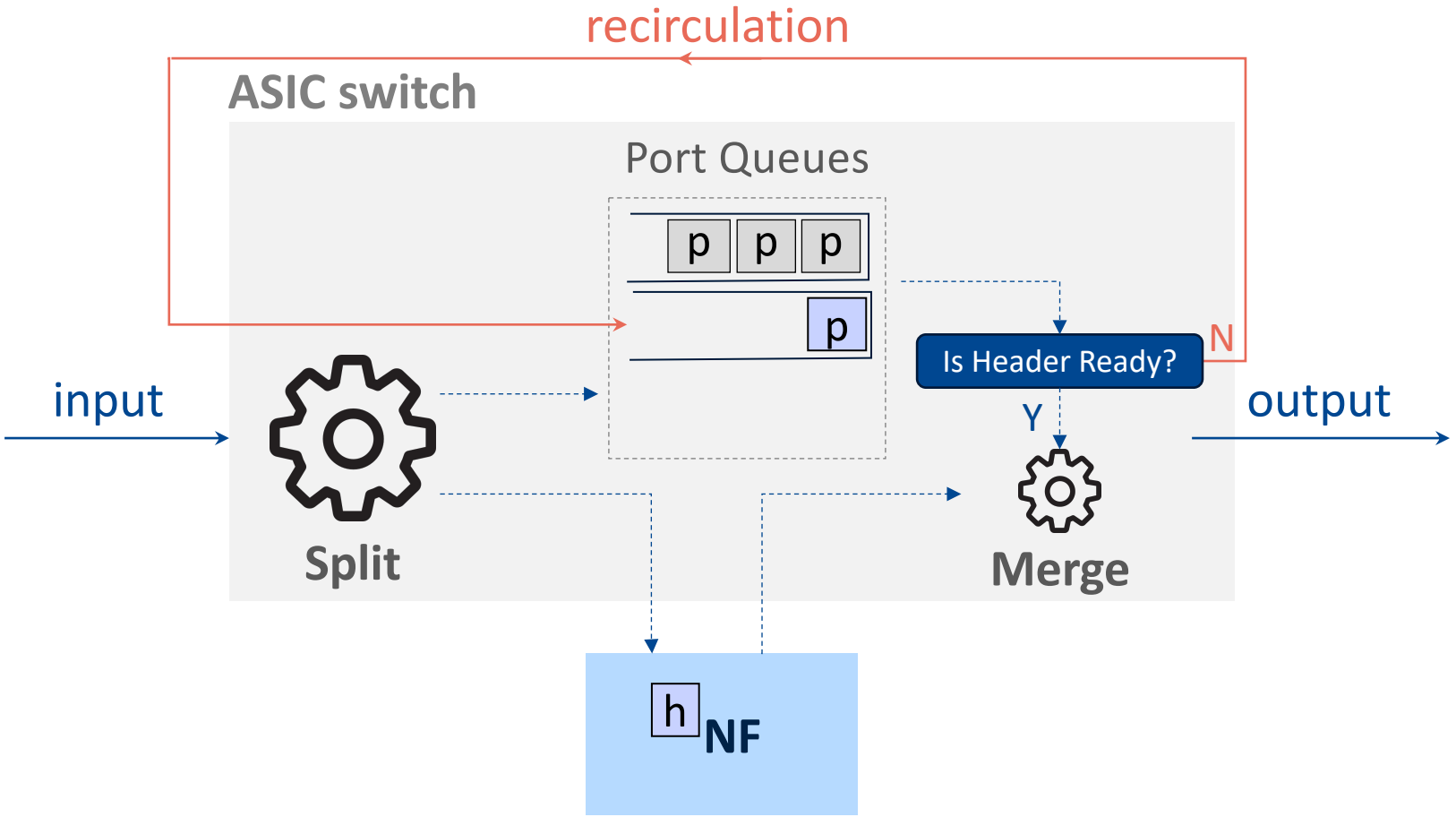
Recirculate payloads until headers are ready

Create copies of payloads to fill up the queues

TurboSwitch

Recirculate payloads until headers are ready

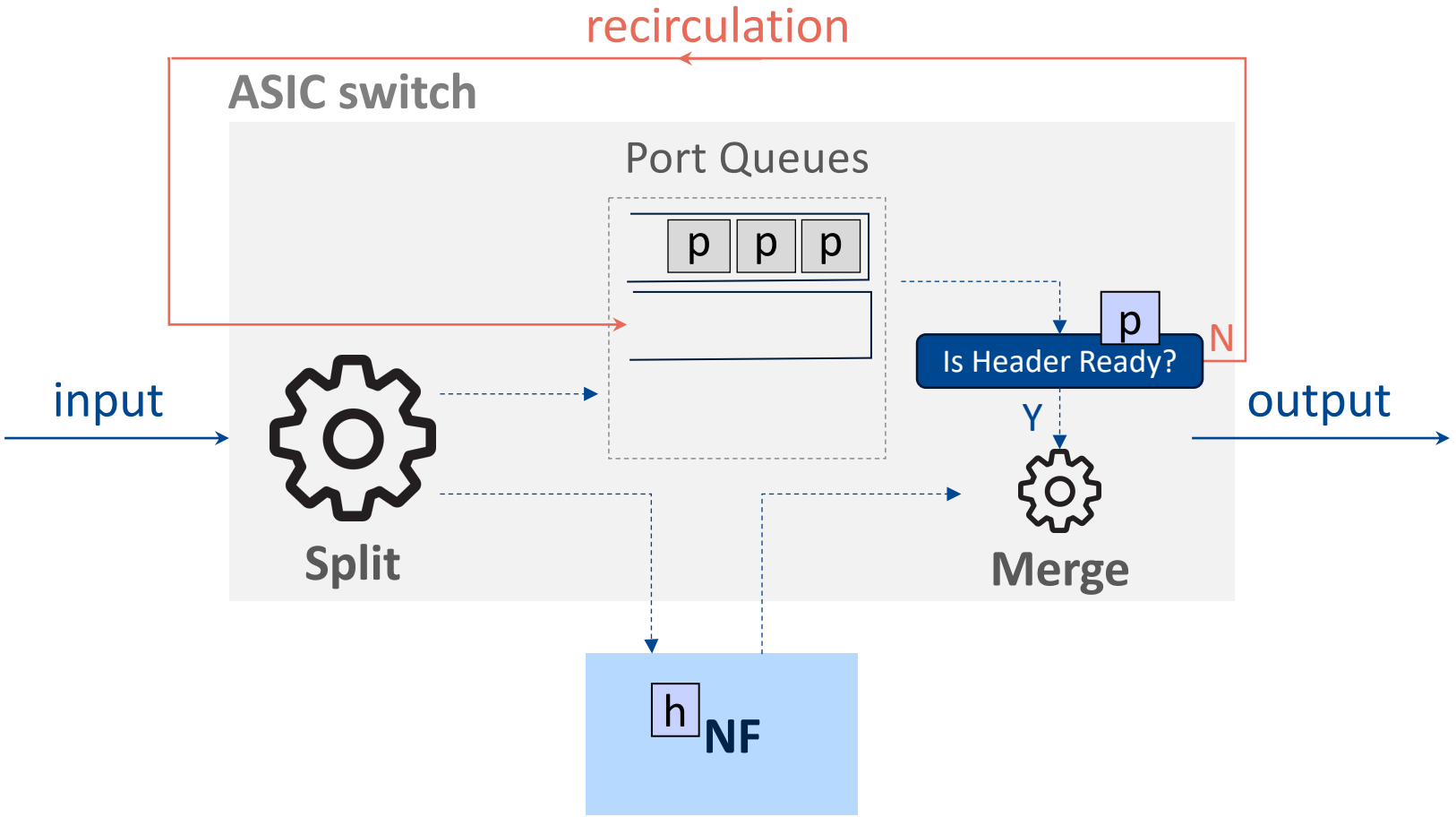
Create copies of payloads to fill up the queues



TurboSwitch

Recirculate payloads until headers are ready

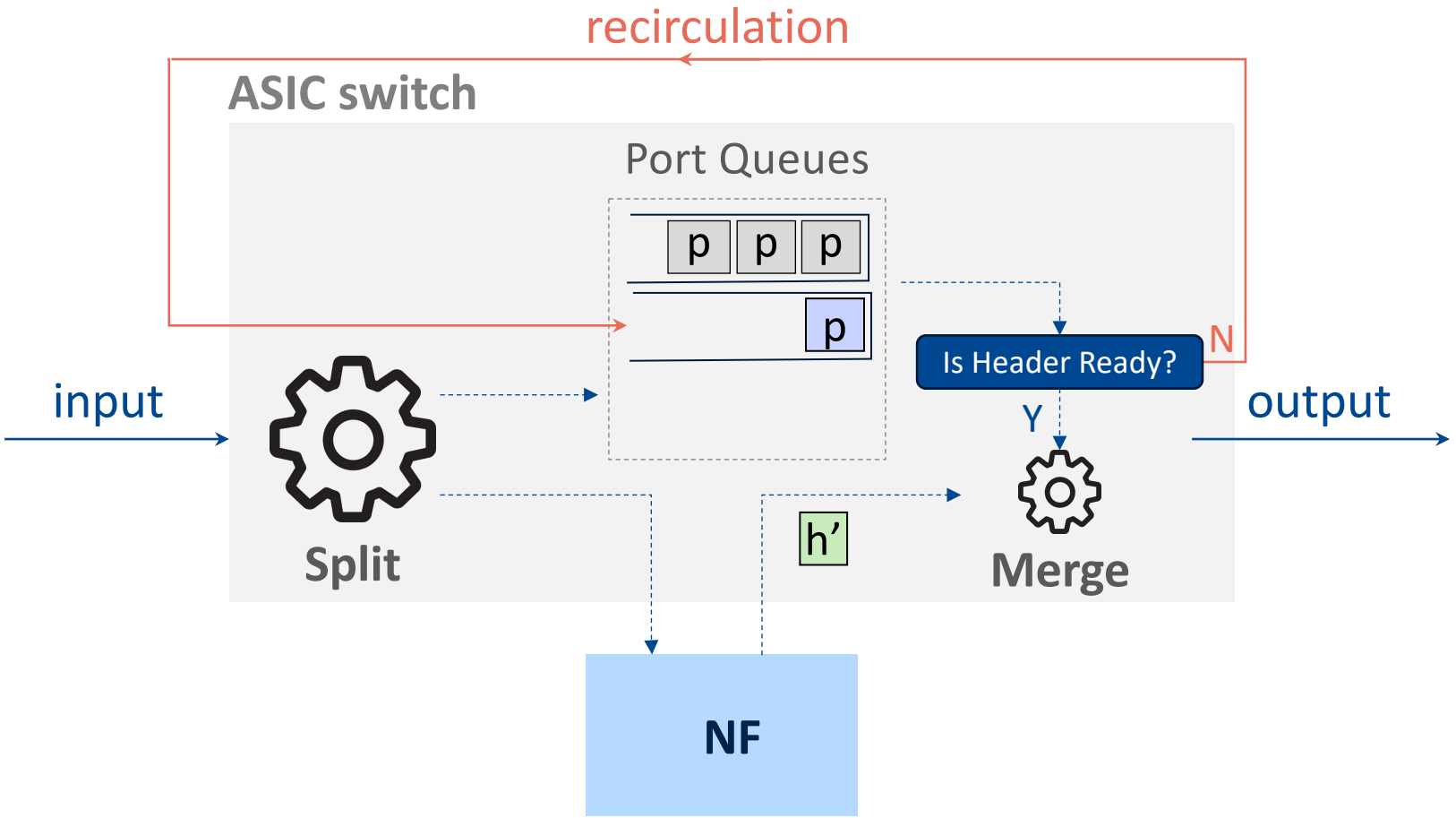
Create copies of payloads to fill up the queues



TurboSwitch

Recirculate payloads until headers are ready

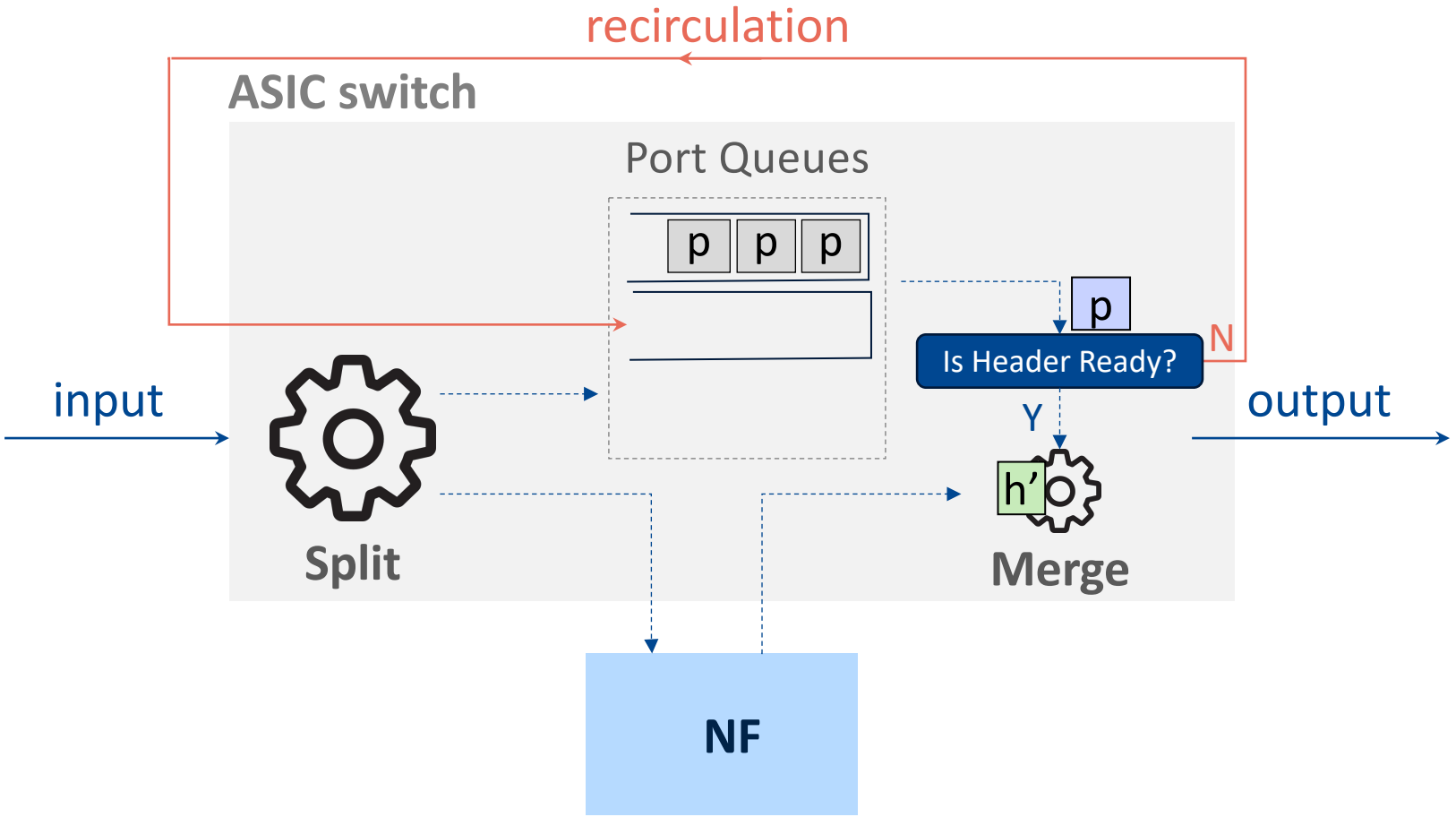
Create copies of payloads to fill up the queues



TurboSwitch

Recirculate payloads until headers are ready

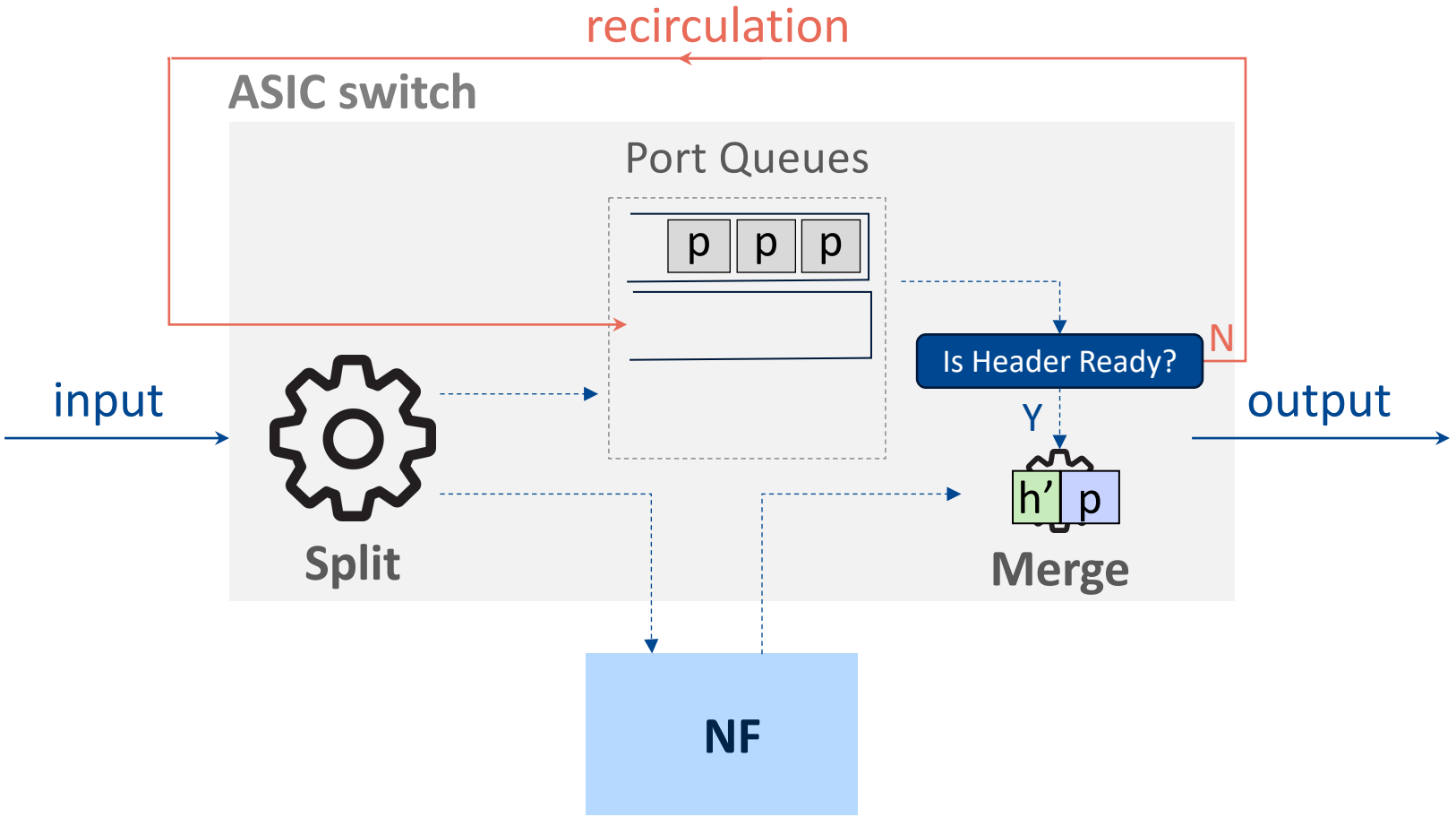
Create copies of payloads to fill up the queues



TurboSwitch

Recirculate payloads until headers are ready

Create copies of payloads to fill up the queues

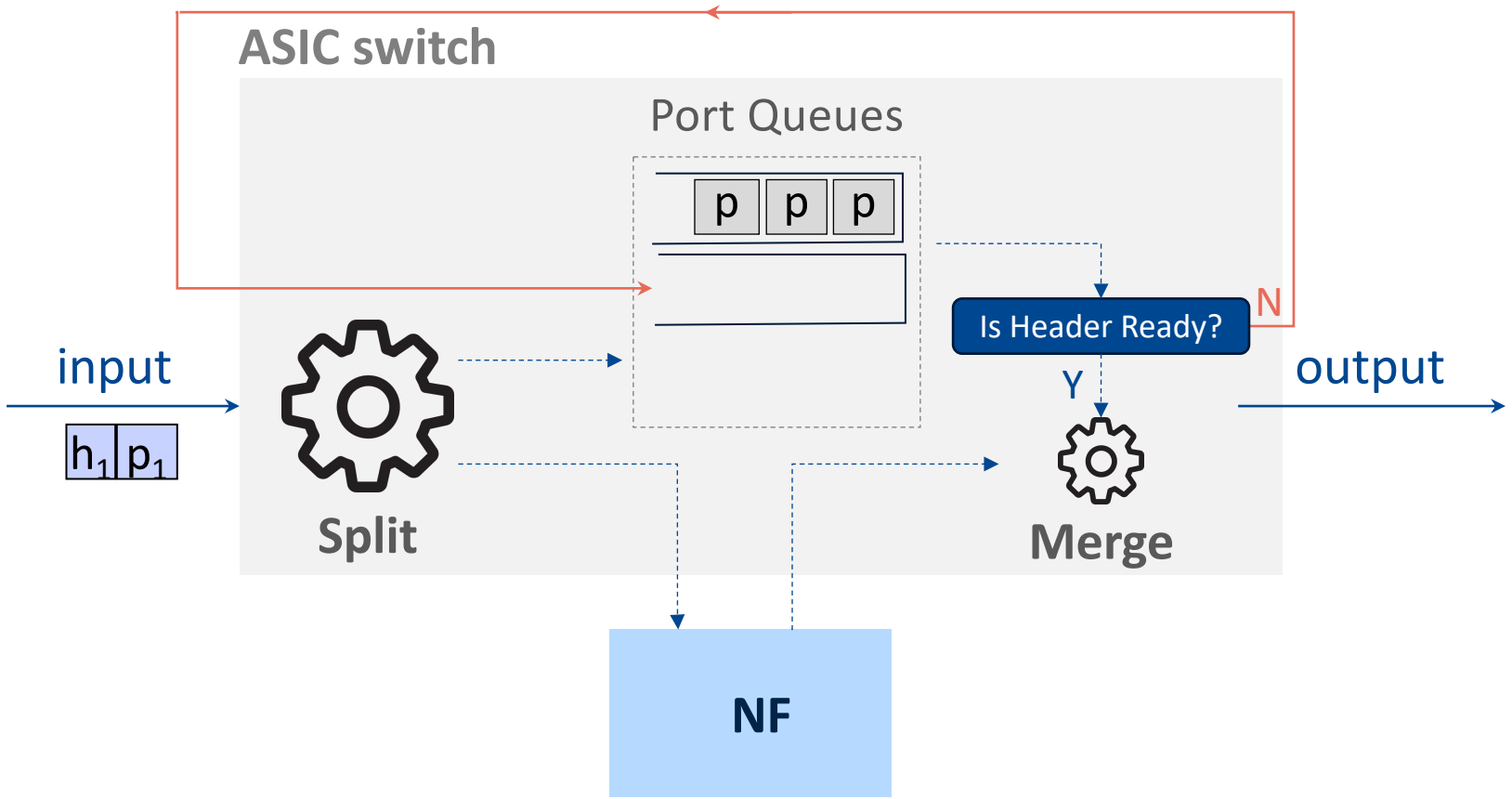


TurboSwitch

Recirculate payloads until headers are ready

Create copies of payloads to fill up the queues

Queue **self-clocking**

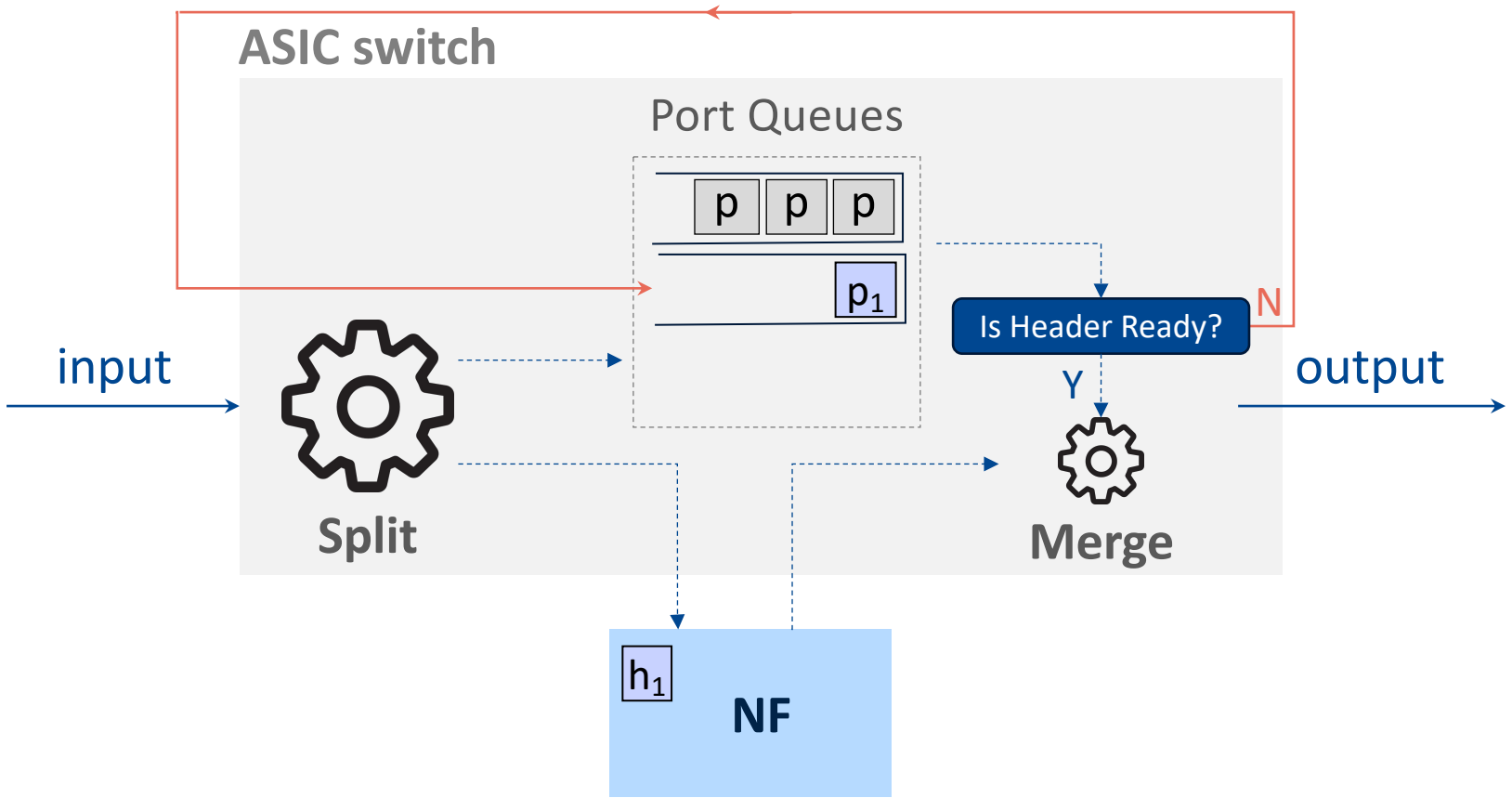


TurboSwitch

Recirculate payloads until headers are ready

Create copies of payloads to fill up the queues

Queue **self-clocking**

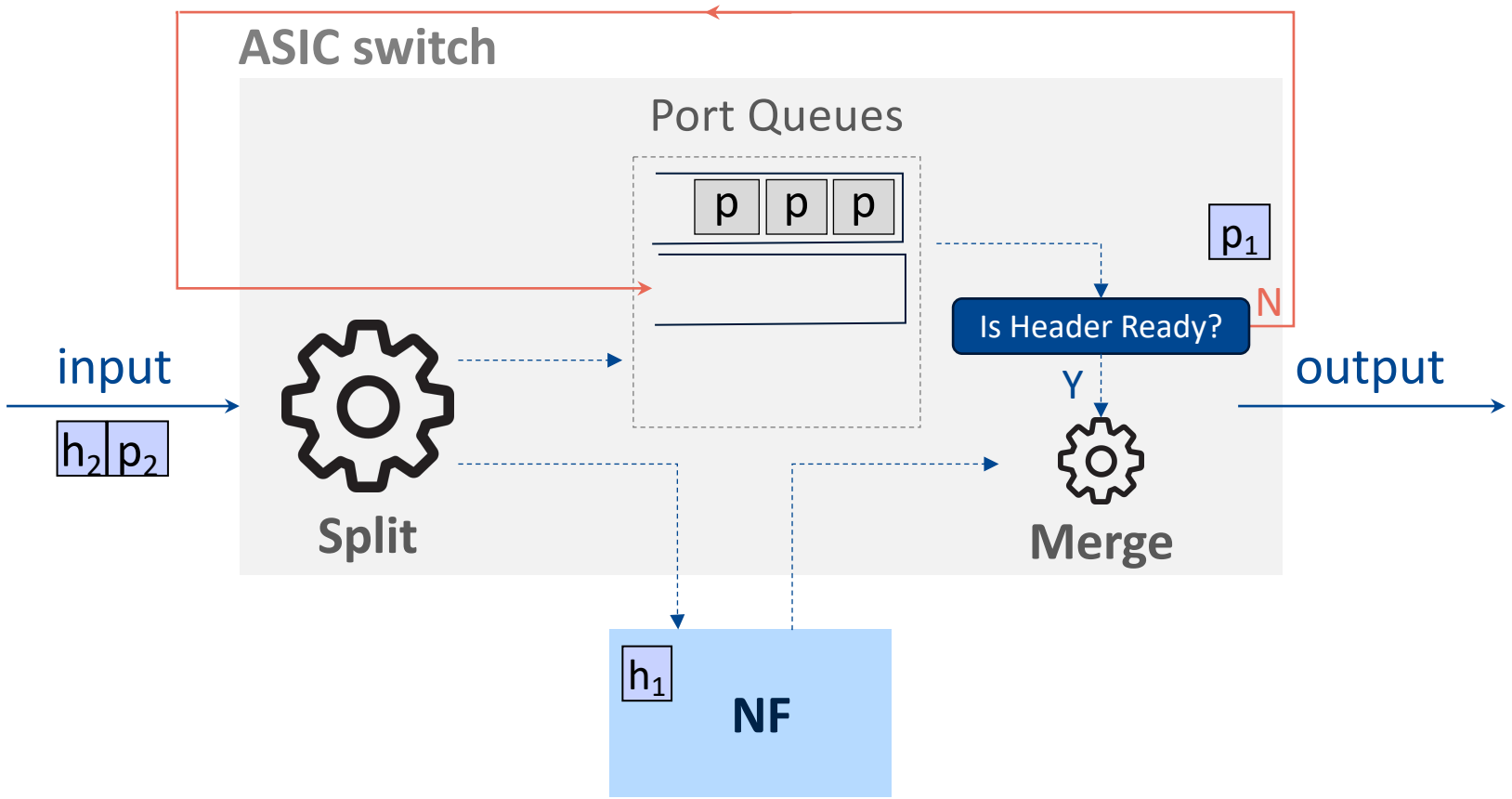


TurboSwitch

Recirculate payloads until headers are ready

Create copies of payloads to fill up the queues

Queue **self-clocking**

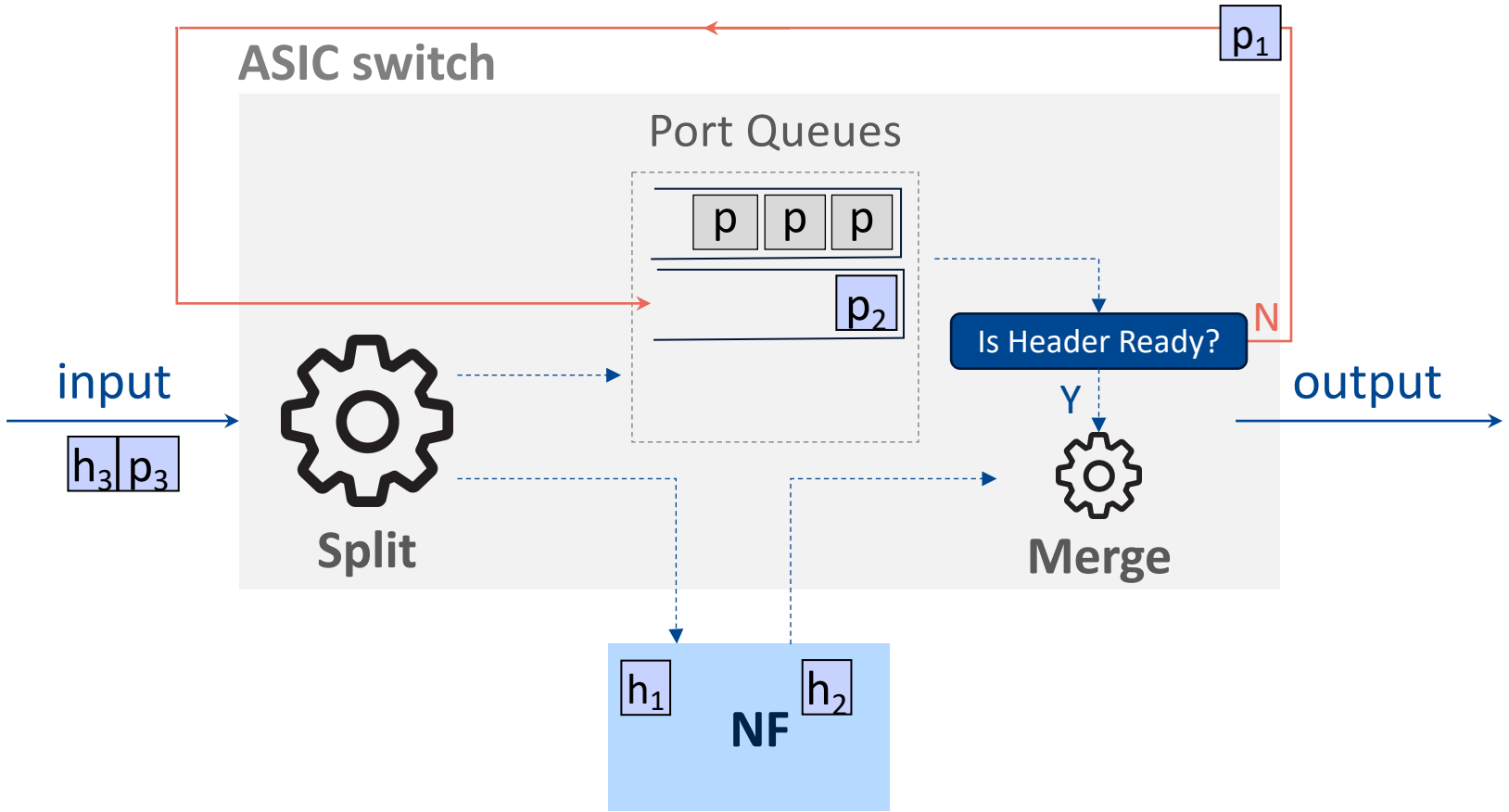


TurboSwitch

Recirculate payloads until headers are ready

Create copies of payloads to fill up the queues

Queue **self-clocking**

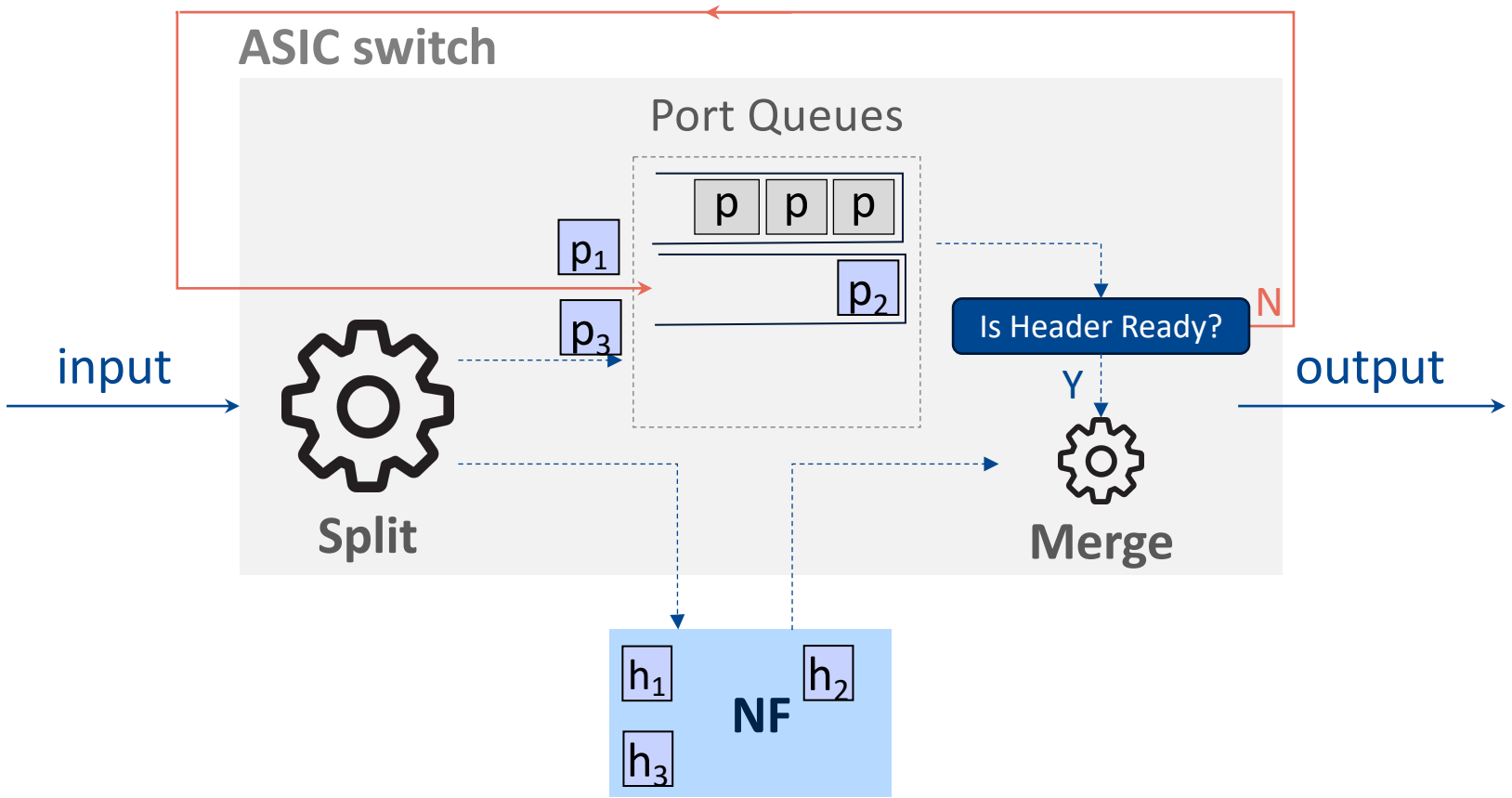


TurboSwitch

Recirculate payloads until headers are ready

Create copies of payloads to fill up the queues

Queue **self-clocking**

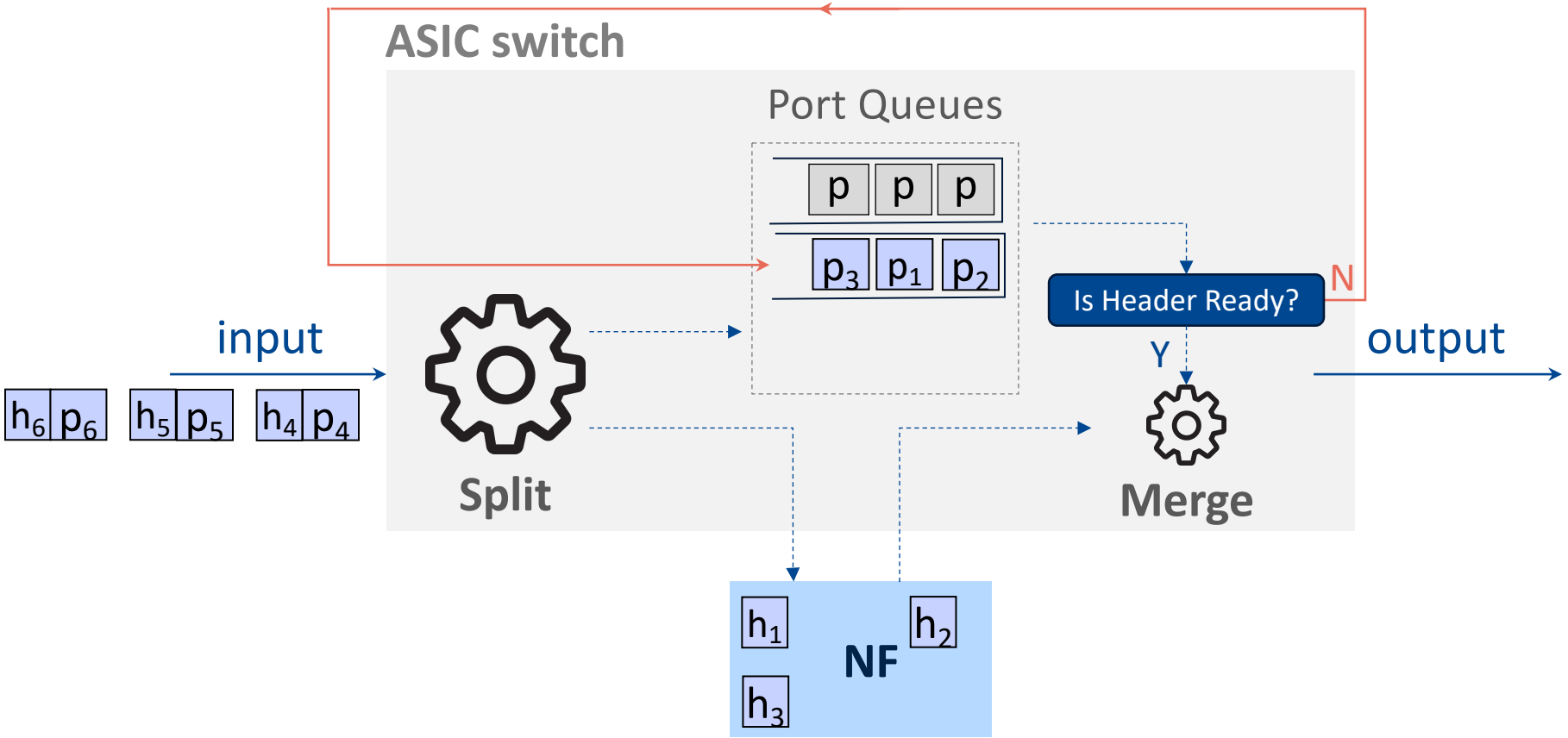


TurboSwitch

Recirculate payloads until headers are ready

Create copies of payloads to fill up the queues

Queue **self-clocking**

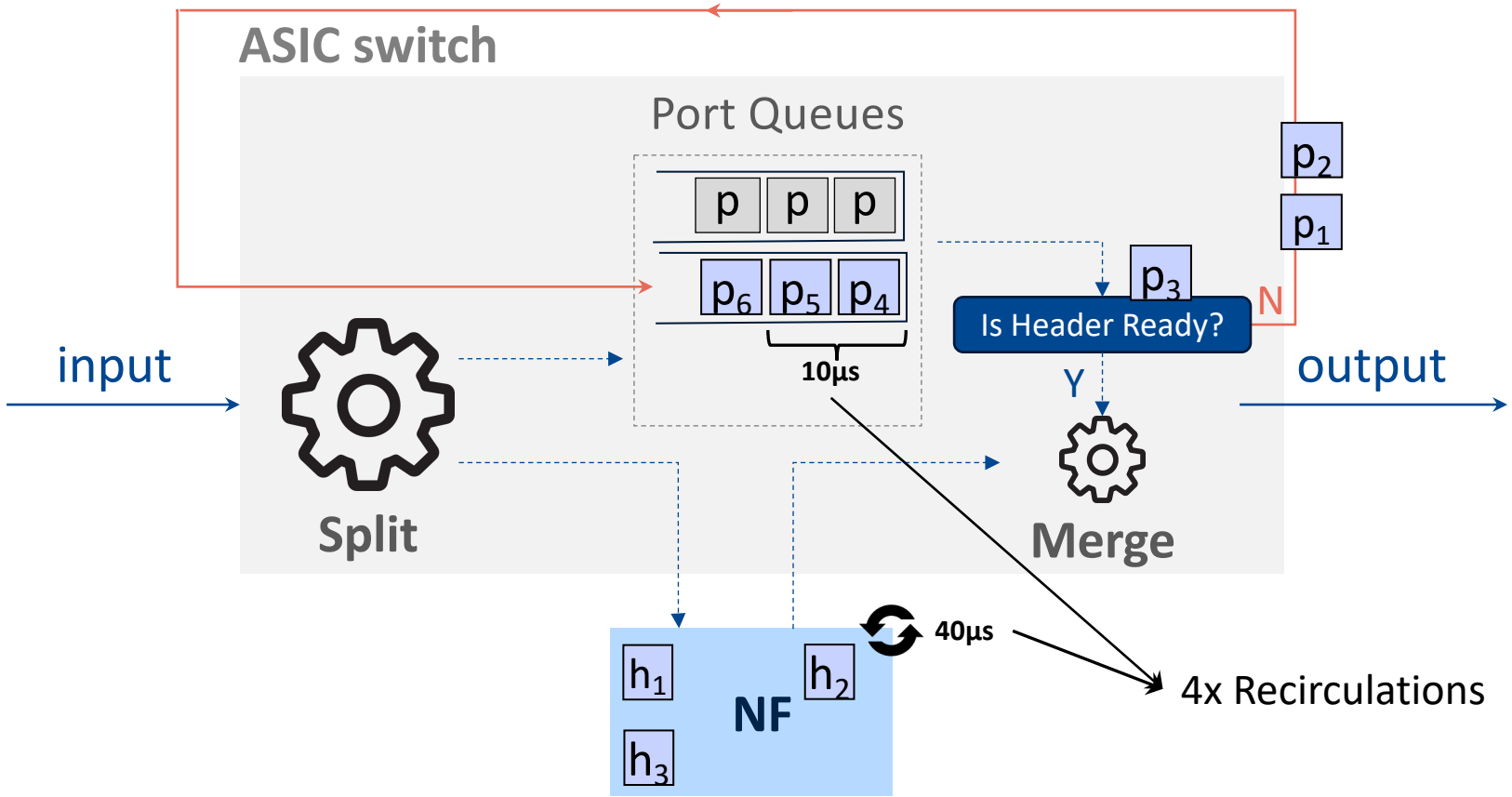


TurboSwitch

Recirculate payloads until headers are ready

Create copies of payloads to fill up the queues

Queue **self-clocking**



TurboSwitch

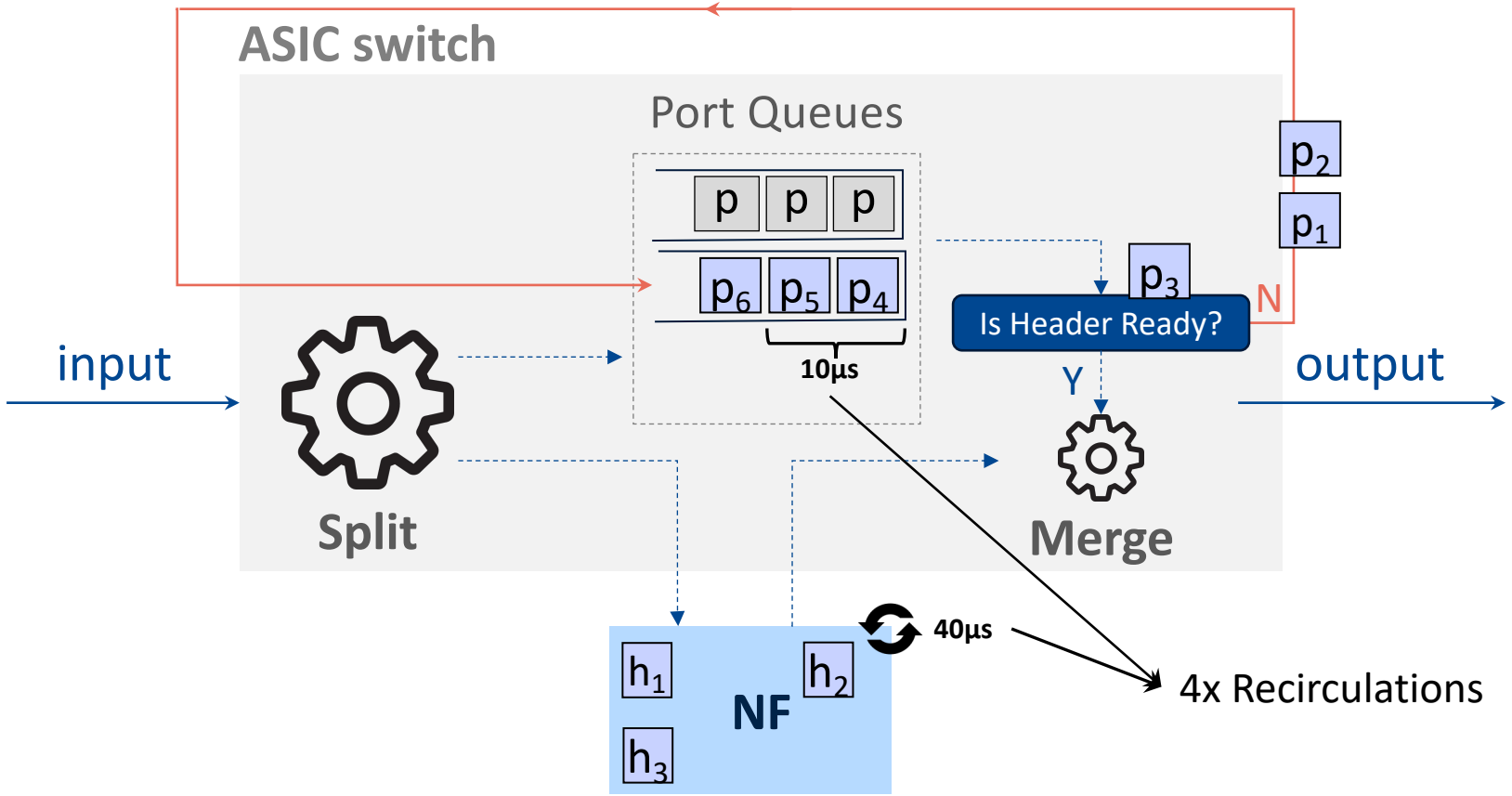
Recirculate payloads until headers are ready

Create copies of payloads to fill up the queues

Queue **self-clocking**

At steady state, # recirculations is predictable

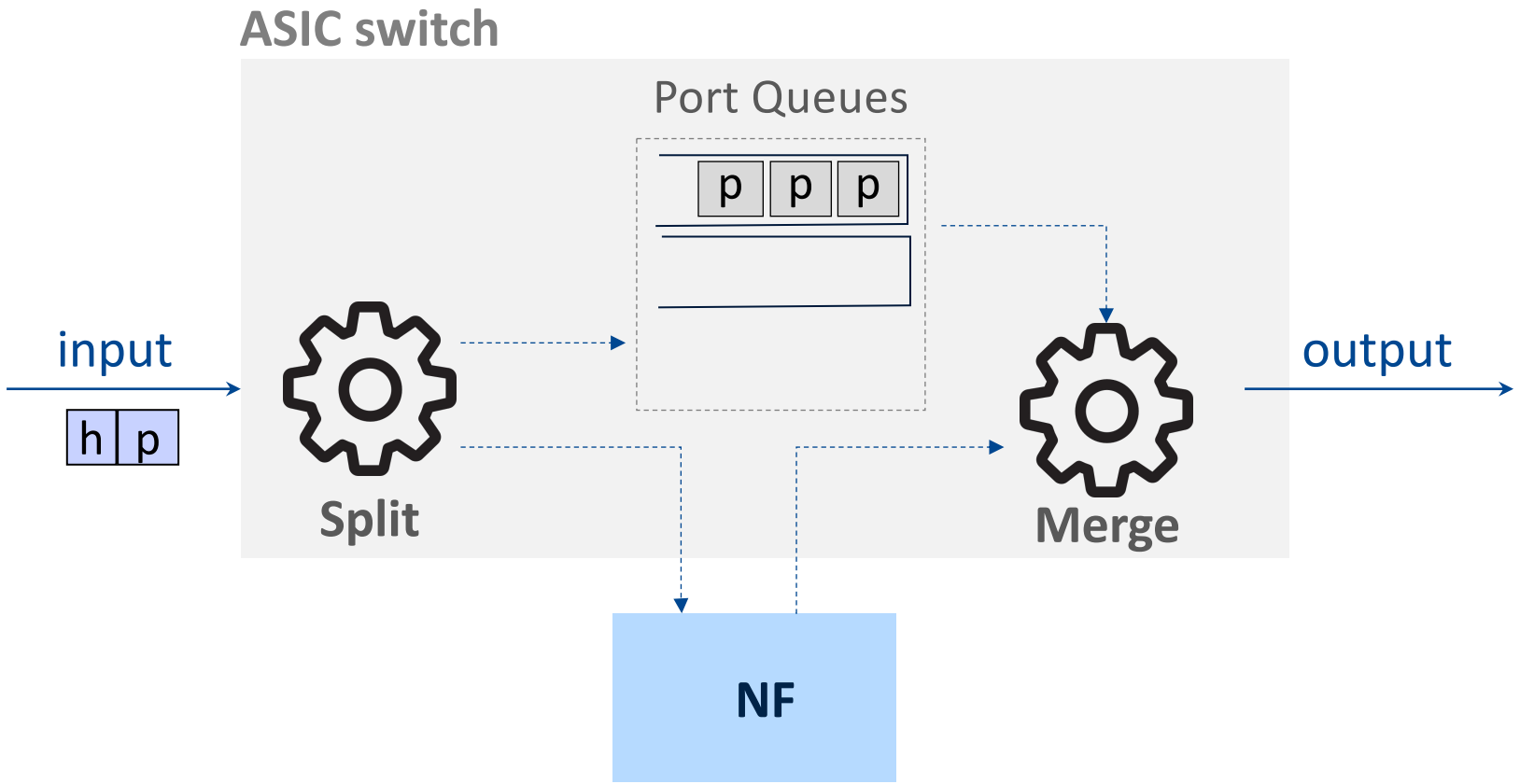
Queue latency converges to the correct value w/o setting any parameter!



TurboSwitch

Create copies of payloads to fill up the queues

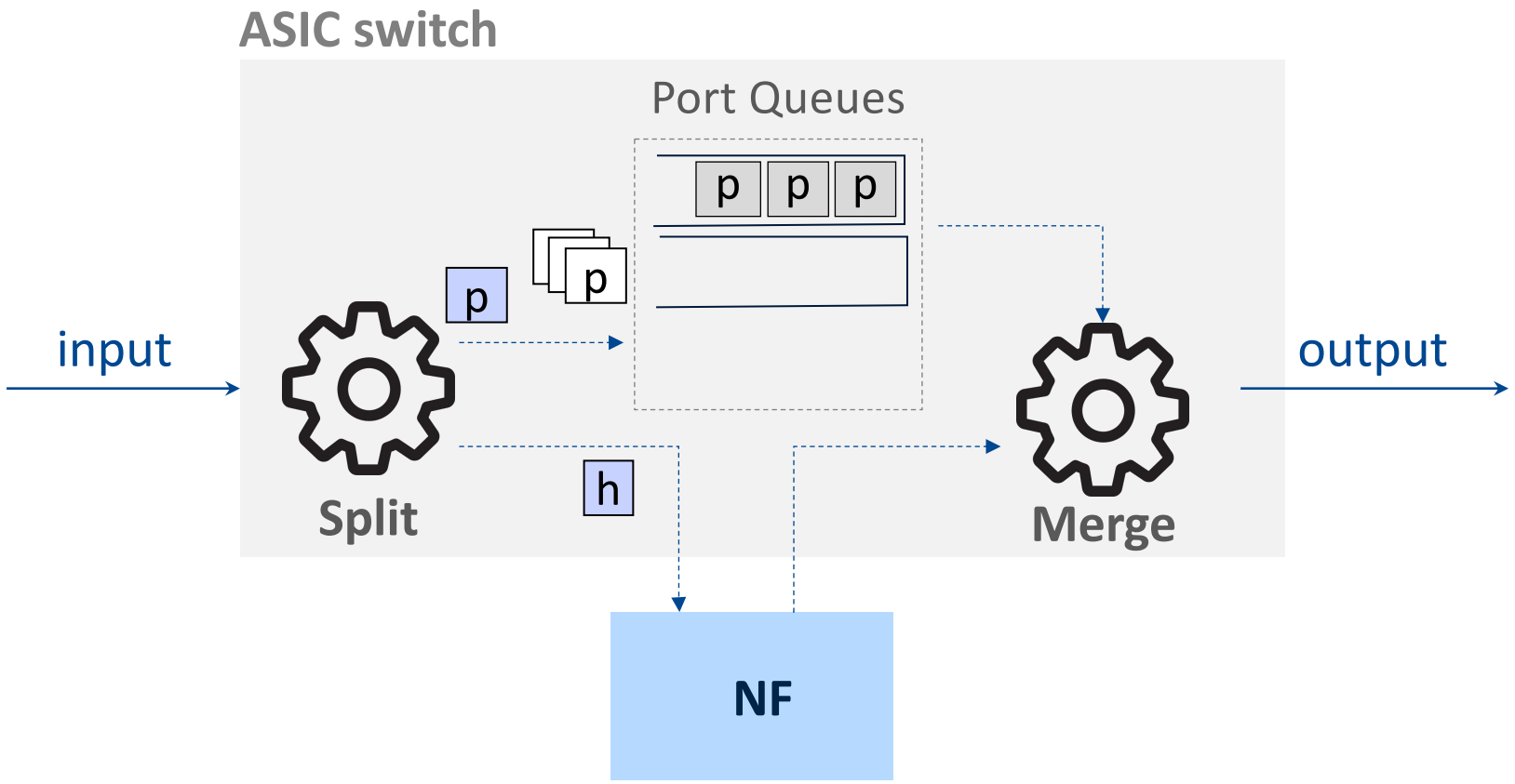
Recirculate payloads until headers are ready



TurboSwitch

Create copies of payloads to fill up the queues

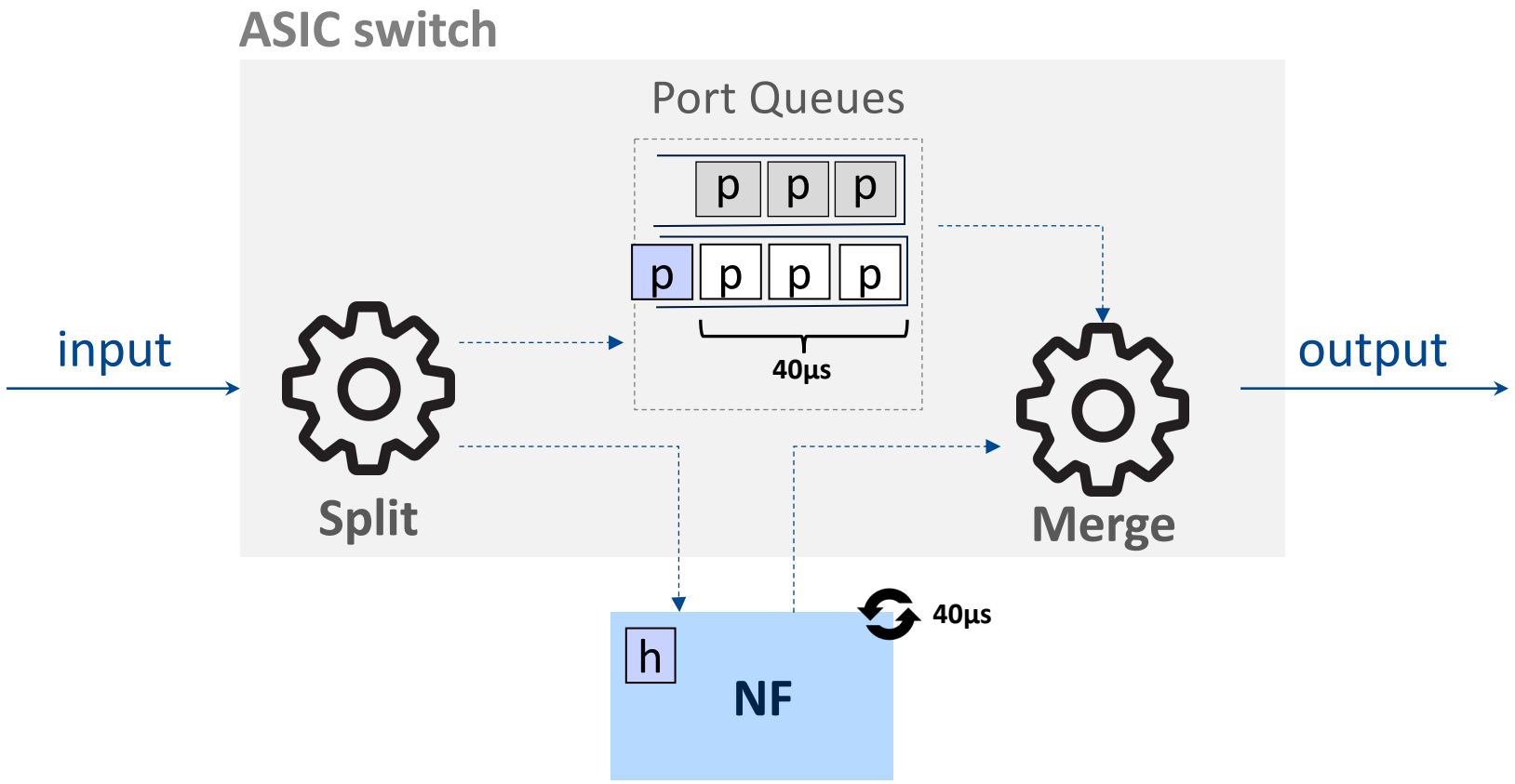
Recirculate payloads until headers are ready



TurboSwitch

Create copies of payloads to fill up the queues

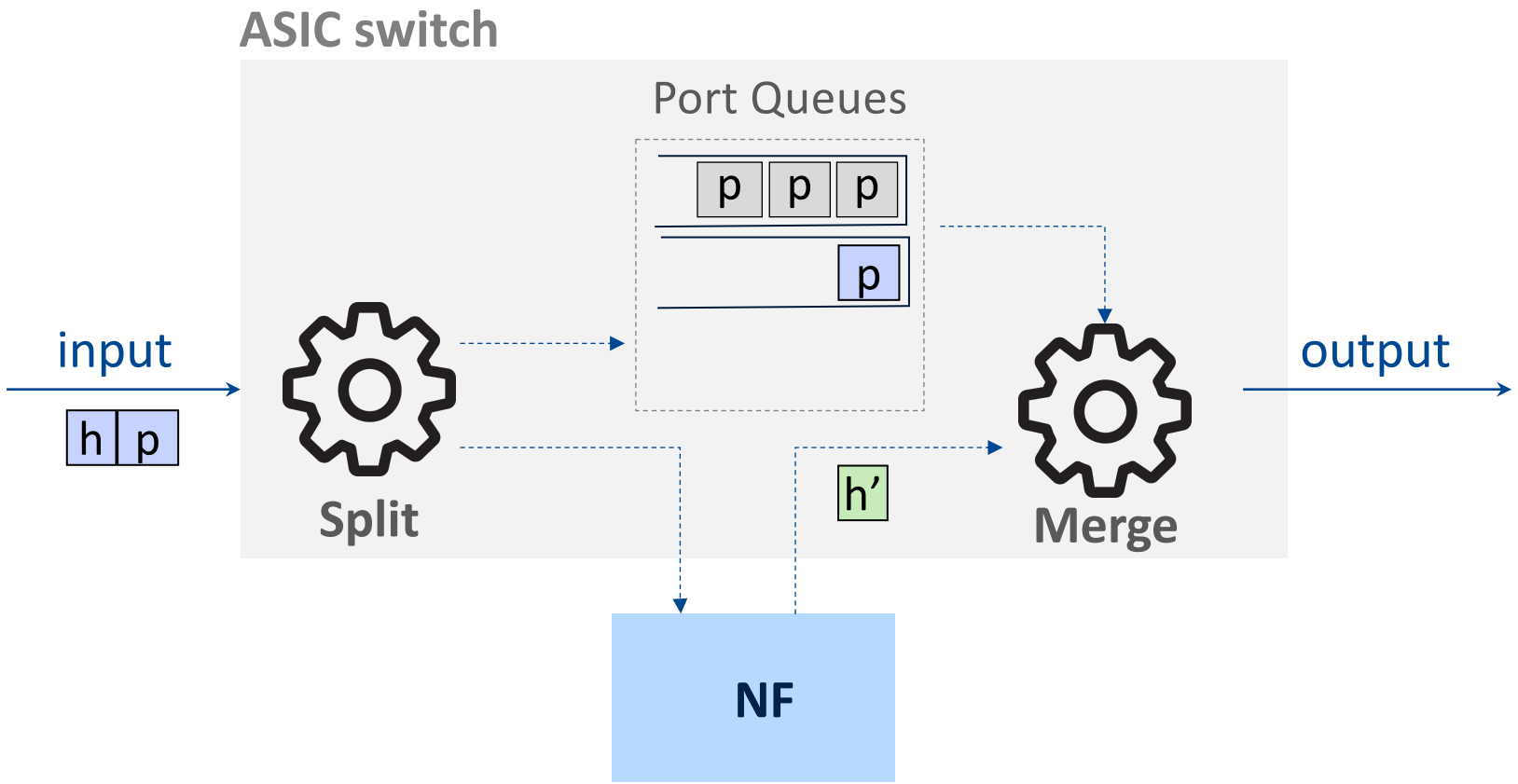
Recirculate payloads until headers are ready



TurboSwitch

Create copies of payloads to fill up the queues

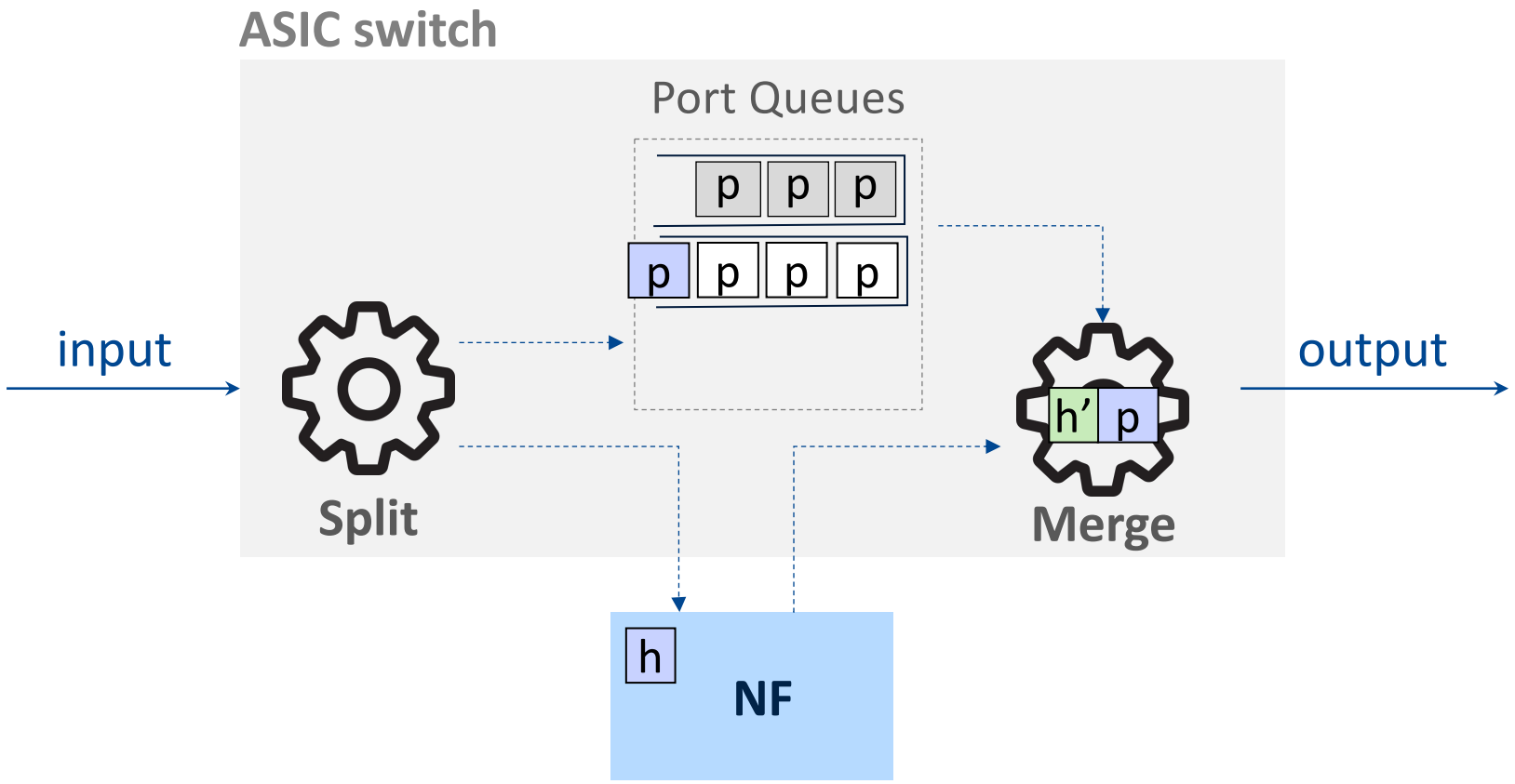
Recirculate payloads until headers are ready



TurboSwitch

Create copies of payloads to fill up the queues

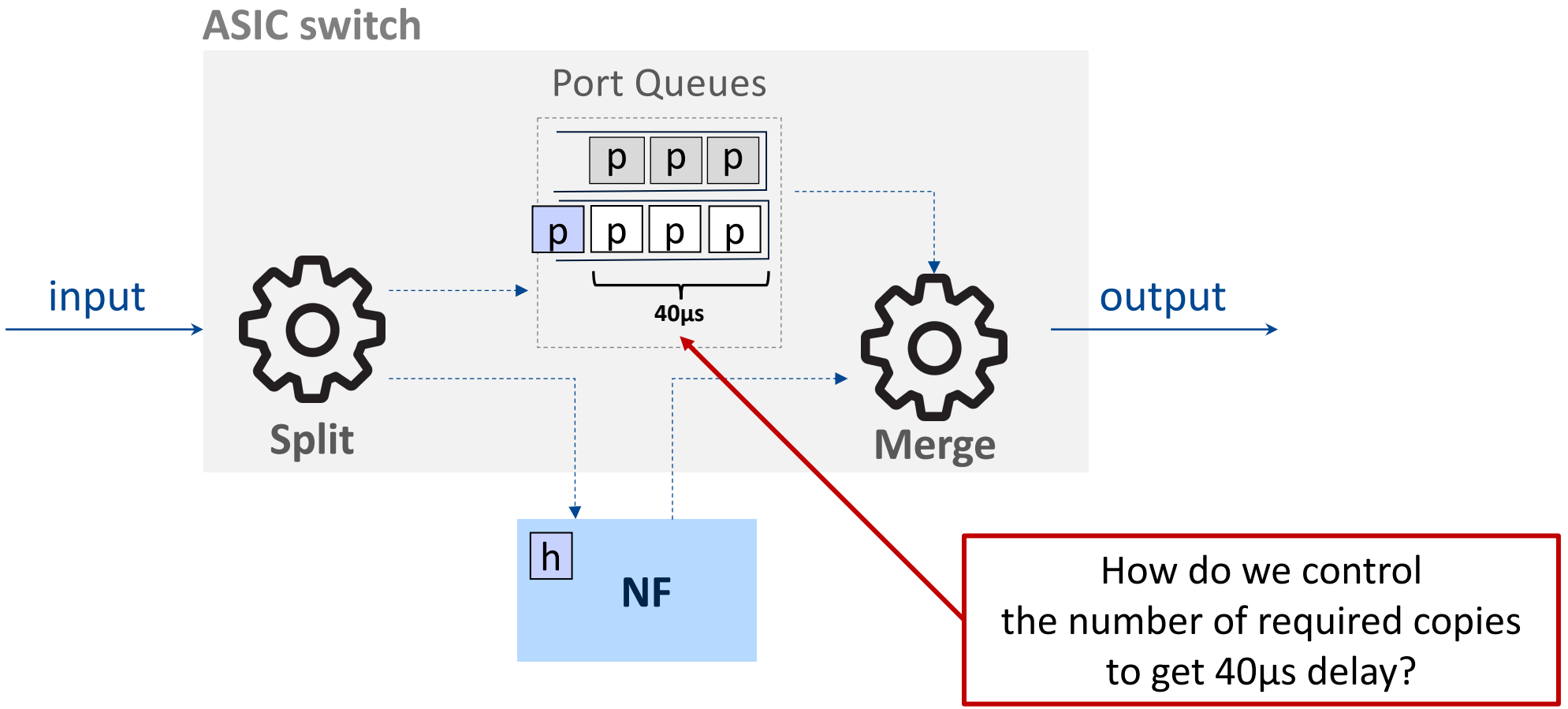
Recirculate payloads until headers are ready



TurboSwitch

Create copies of payloads to fill up the queues

Recirculate payloads until headers are ready

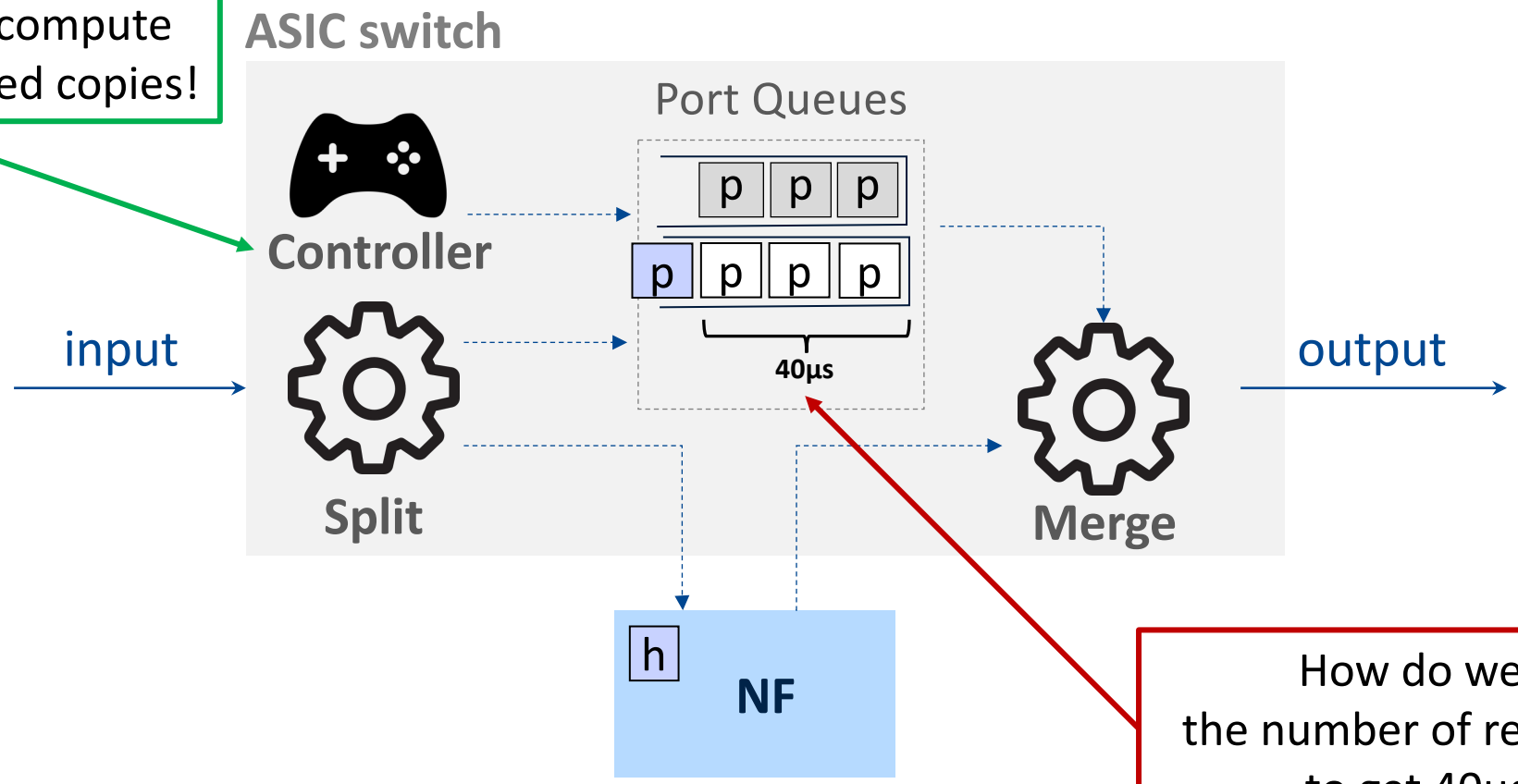


TurboSwitch

Create copies of payloads to fill up the queues

Recirculate payloads until headers are ready

Use Queue Stats to compute the number of required copies!



How do we control the number of required copies to get 40µs delay?

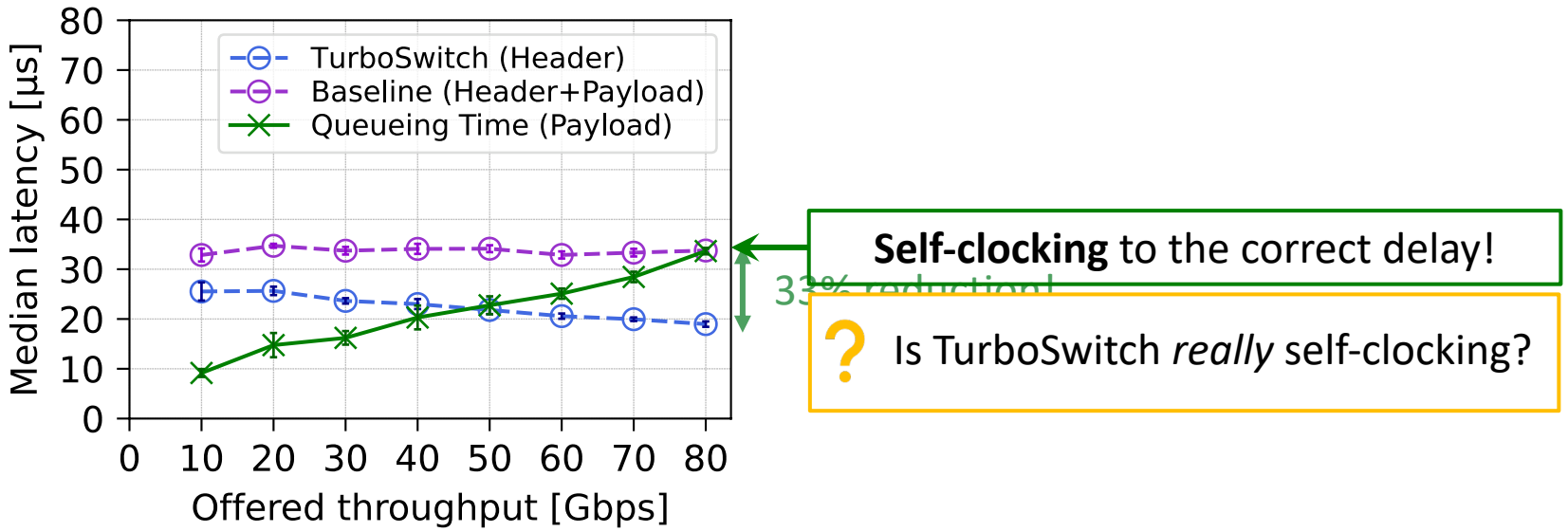
Evaluation

Evaluation

Recirculate payloads

- Intel Tofino ASIC
- FastClick Forwarder NF ⟹ 20µs Average Latency
- 100Gbps wiring

What are the NF latency gains of TurboSwitch?

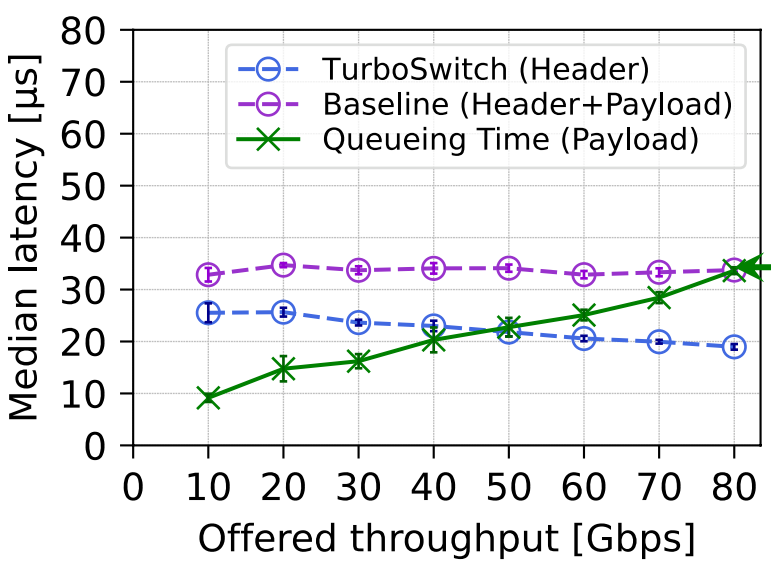


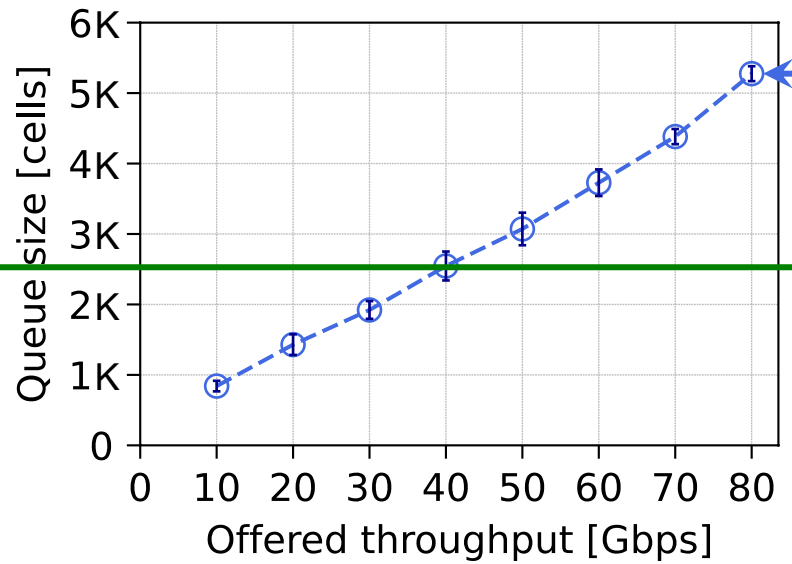
Evaluation

Recirculate payloads

- Intel Tofino ASIC
- FastClick Forwarder NF ⟹ 25µs Average Latency
- 100Gbps wiring

Is TurboSwitch *really* self-clocking?





Queue size grows linearly w.r.t. input throughput

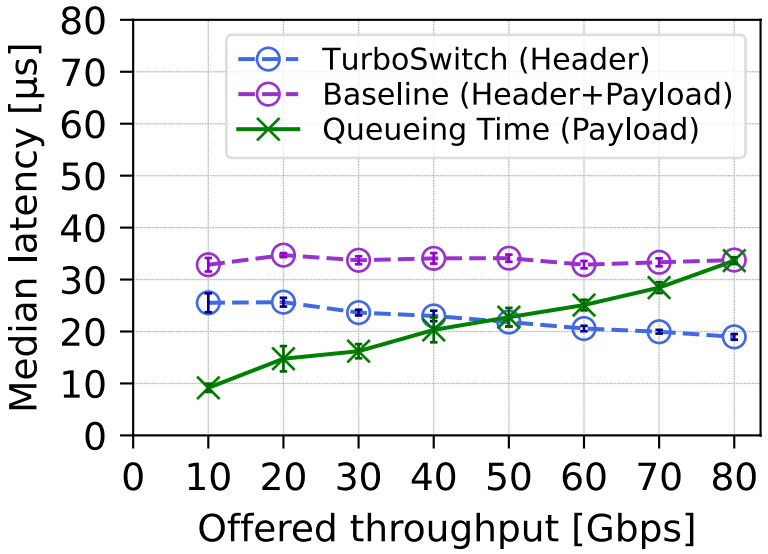
Linear relationship between payload queueing time and queue size

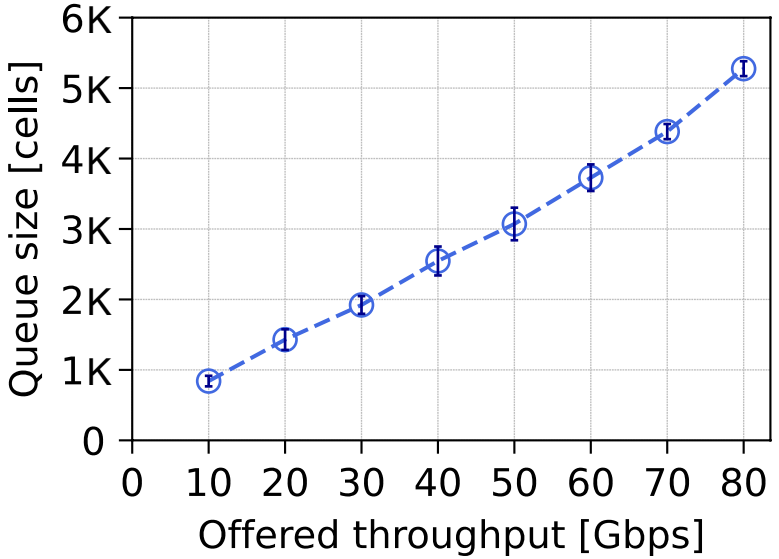
Evaluation

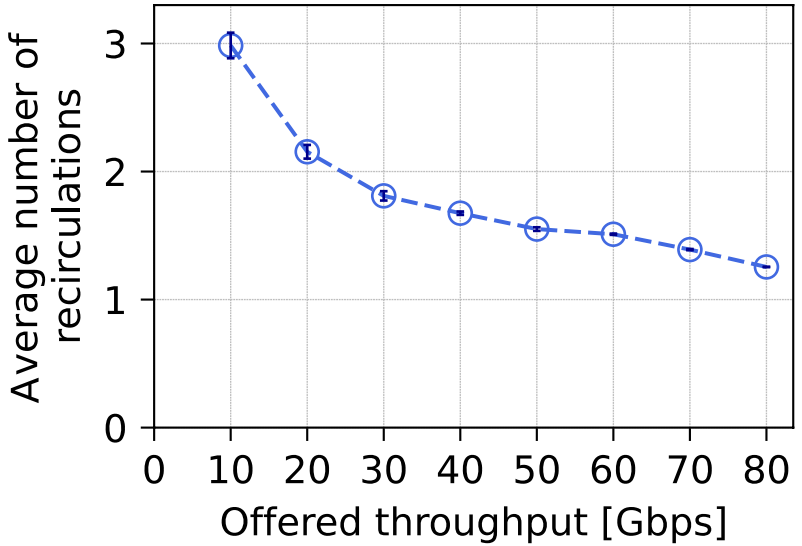
Recirculate payloads

- Intel Tofino ASIC
- FastClick Forwarder NF ⟹ 25µs Average Latency
- 100Gbps wiring

Is TurboSwitch *really* self-clocking?





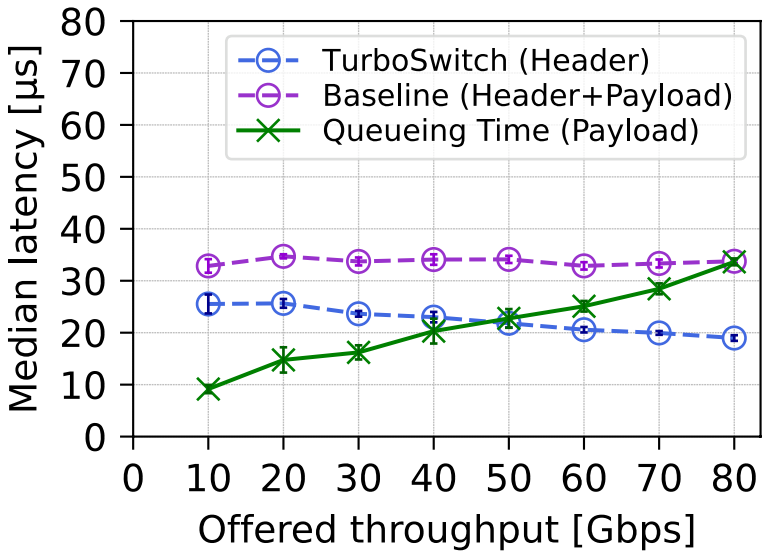


Evaluation

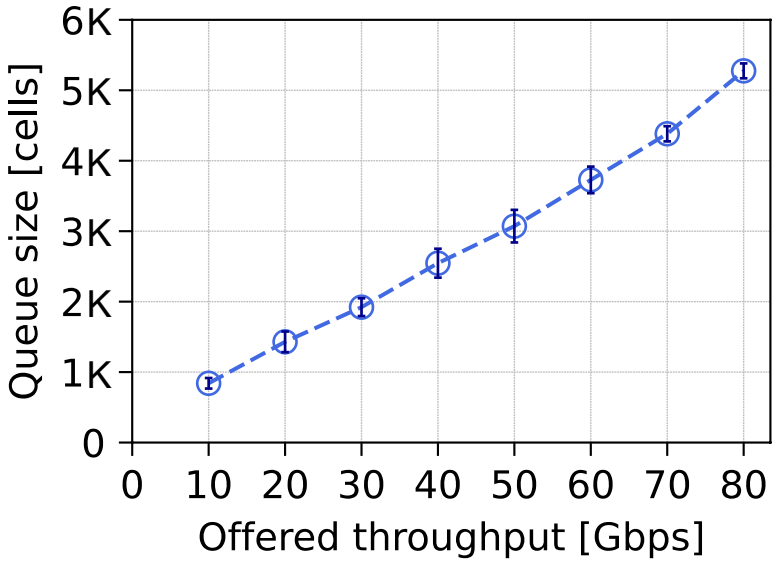
Recirculate payloads

- Intel Tofino ASIC
- FastClick Forwarder NF ⟹ 25µs Average Latency
- 100Gbps wiring

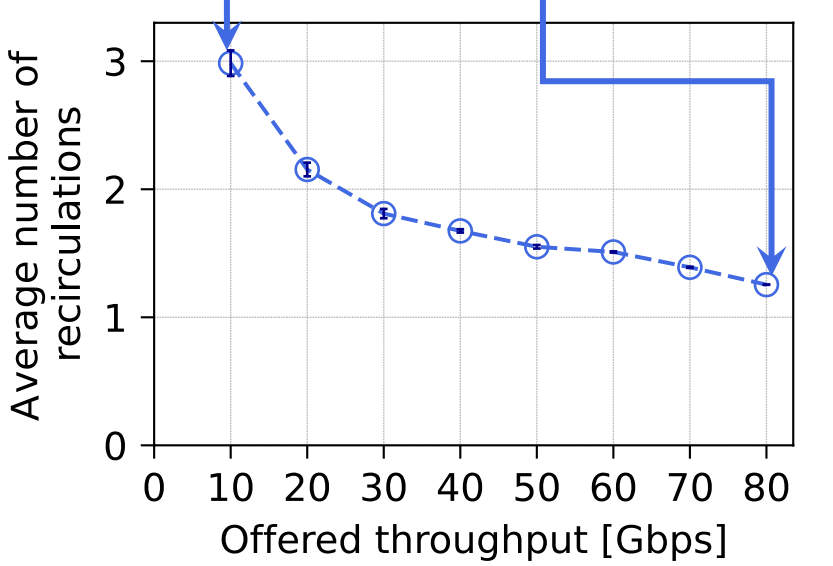
Is TurboSwitch *really* self-clocking?



More recirculations
to keep a steady state



~1 recirculation per packet



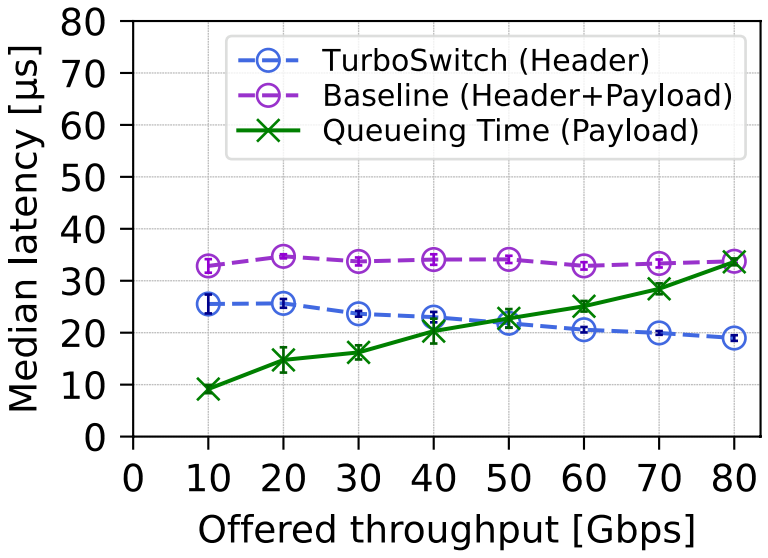
Evaluation

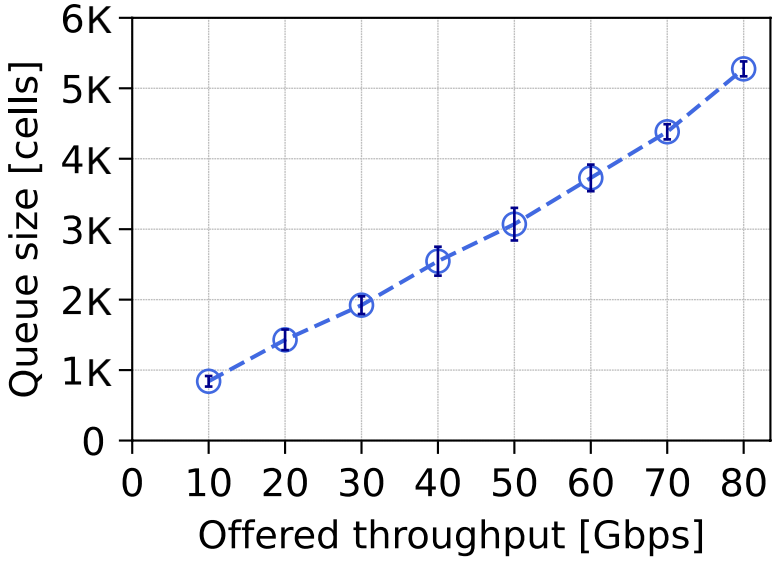
Recirculate payloads

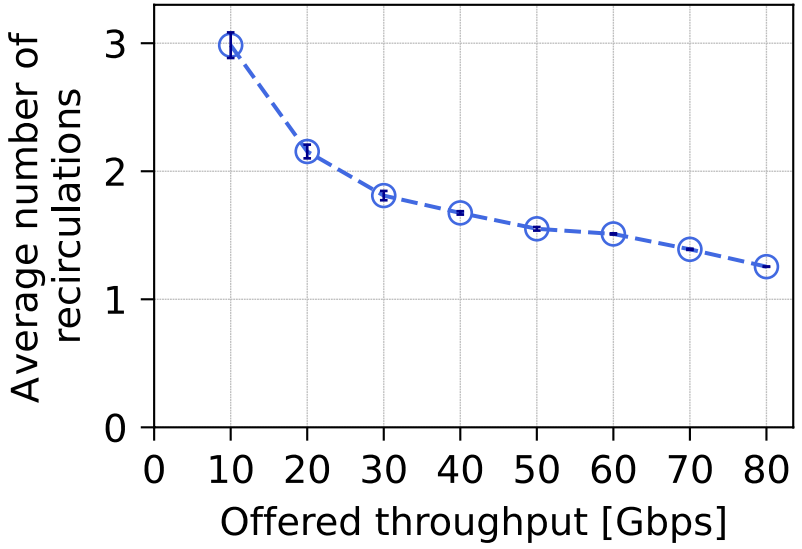
- Intel Tofino ASIC
- FastClick Forwarder NF ⟹ 25µs Average Latency
- 100Gbps wiring

Is TurboSwitch *really* self-clocking?

Turboswitch **improves** NF performance by inducing **self-clocked congestion**!







Evaluation

Create copies of payloads

- Python Simulator – 20µs NF Processing Latency
- 100Gbps w/ Poisson arrival (64B-1.5KB pkt size)
- Controller ⟹ 1µs Queue Stats Update Delay

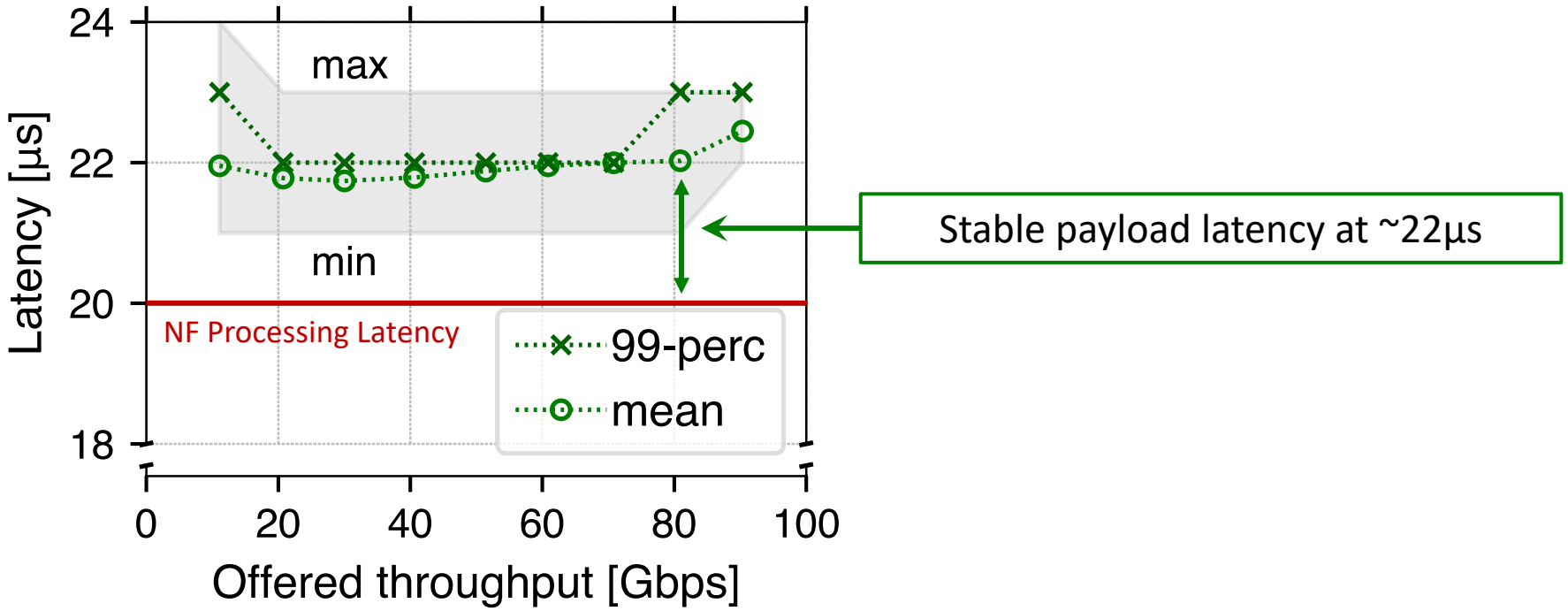
Can TurboSwitch's controller enforce the correct delay to payloads?

Evaluation

Create copies of payloads

- Python Simulator – 20µs NF Processing Latency
- 100Gbps w/ Poisson arrival (64B-1.5KB pkt size)
- Controller ⟹ 1µs Queue Stats Update Delay

Can TurboSwitch's controller enforce the correct delay to payloads?

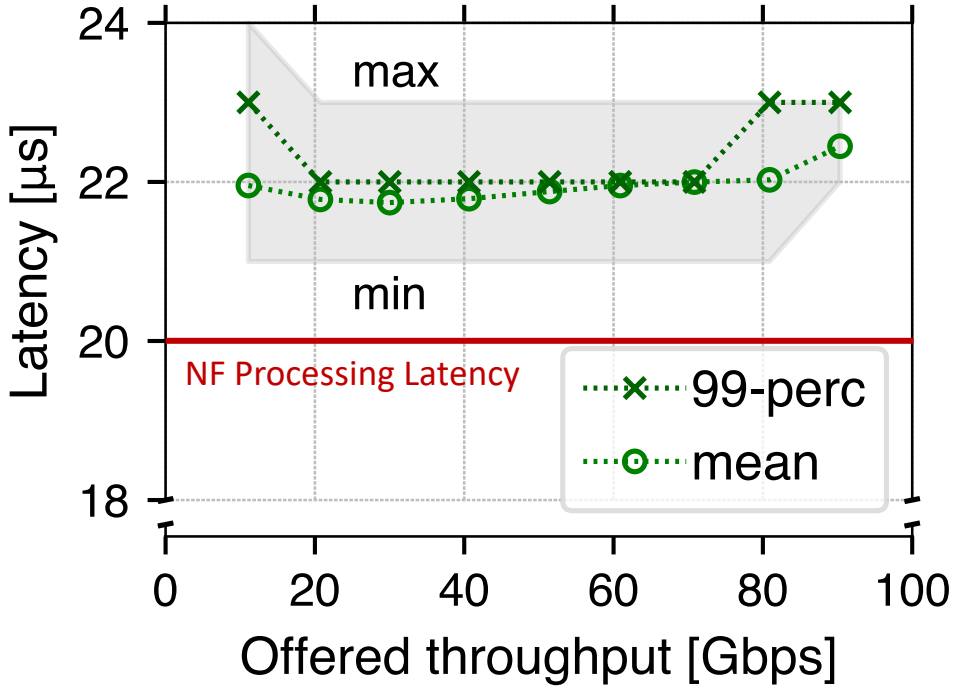


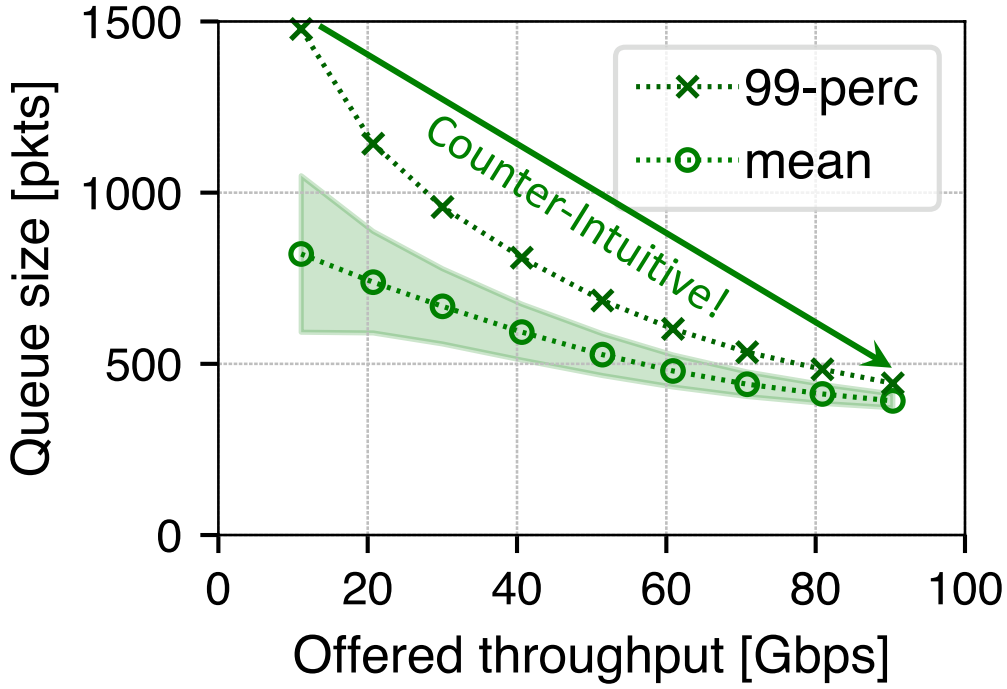
Evaluation

Create copies of payloads

- Python Simulator – 20µs NF Processing Latency
- 100Gbps w/ Poisson arrival (64B-1.5KB pkt size)
- Controller ⟹ 1µs Queue Stats Update Delay

Can TurboSwitch's controller enforce the correct delay to payloads?



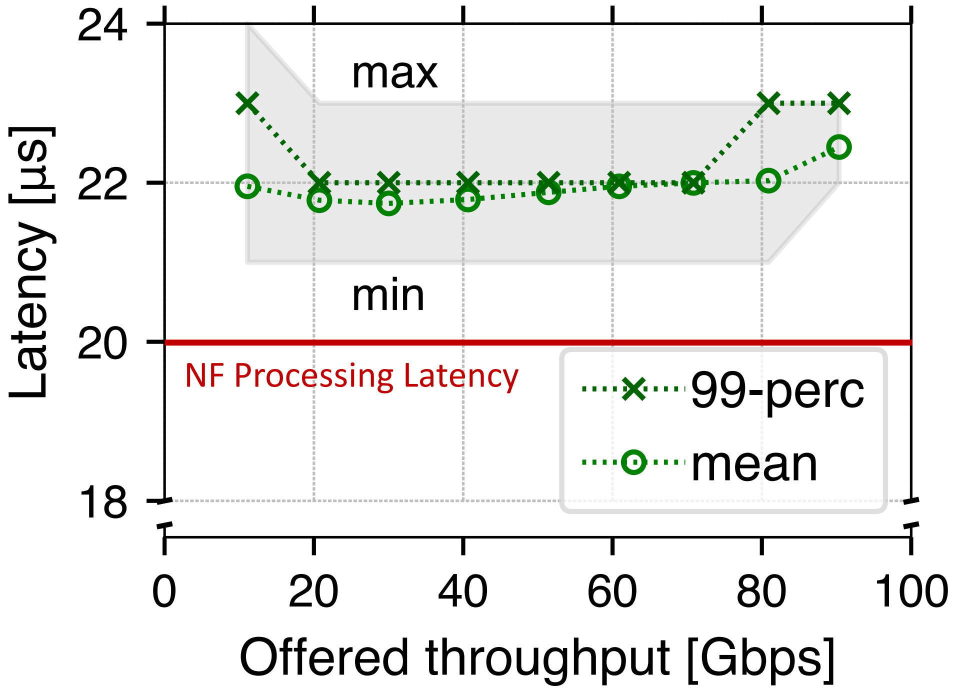


Evaluation

Create copies of payloads

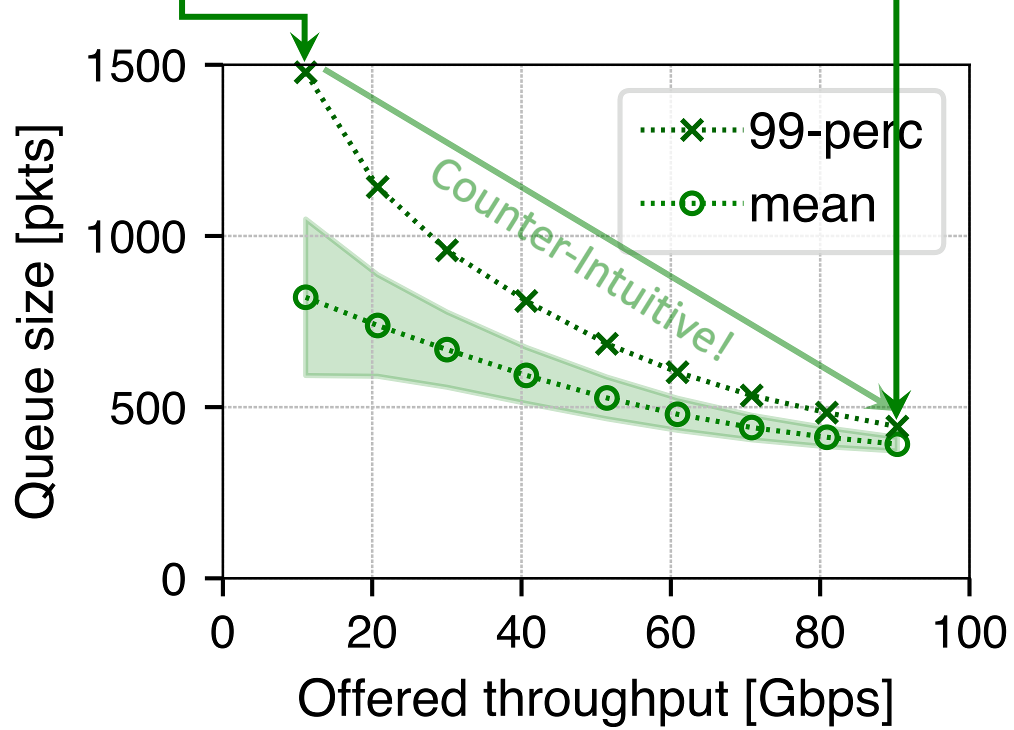
- Python Simulator – 20µs NF Processing Latency
- 100Gbps w/ Poisson arrival (64B-1.5KB pkt size)
- Controller ⟹ 1µs Queue Stats Update Delay

Can TurboSwitch's controller enforce the correct delay to payloads?



↑ **Less** incoming packets, controller creates **more copies**

↓ **More** incoming packets, controller creates **less copies**



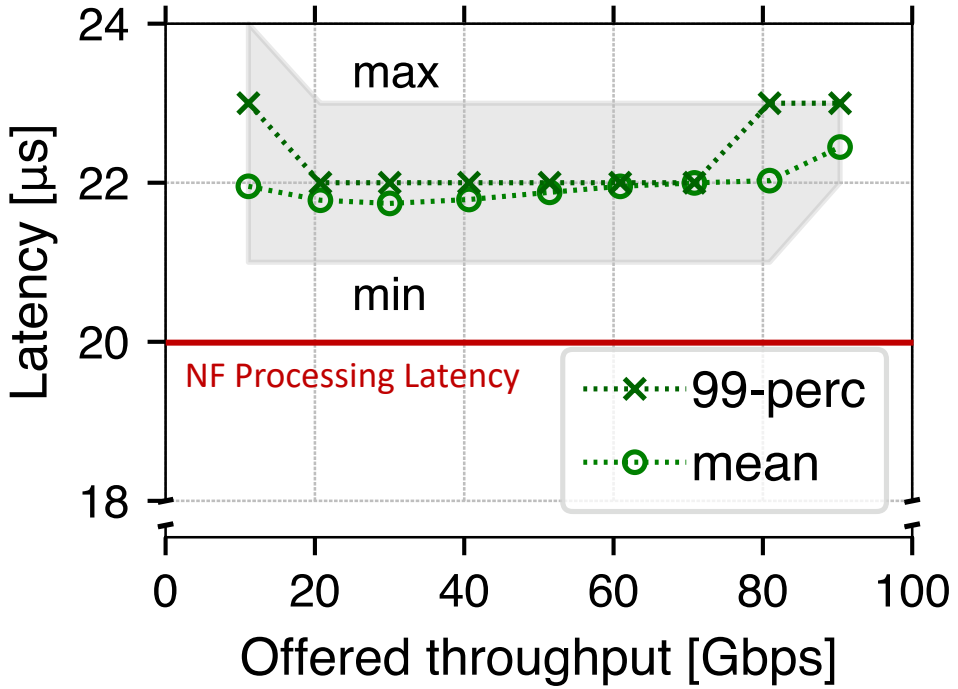
Evaluation

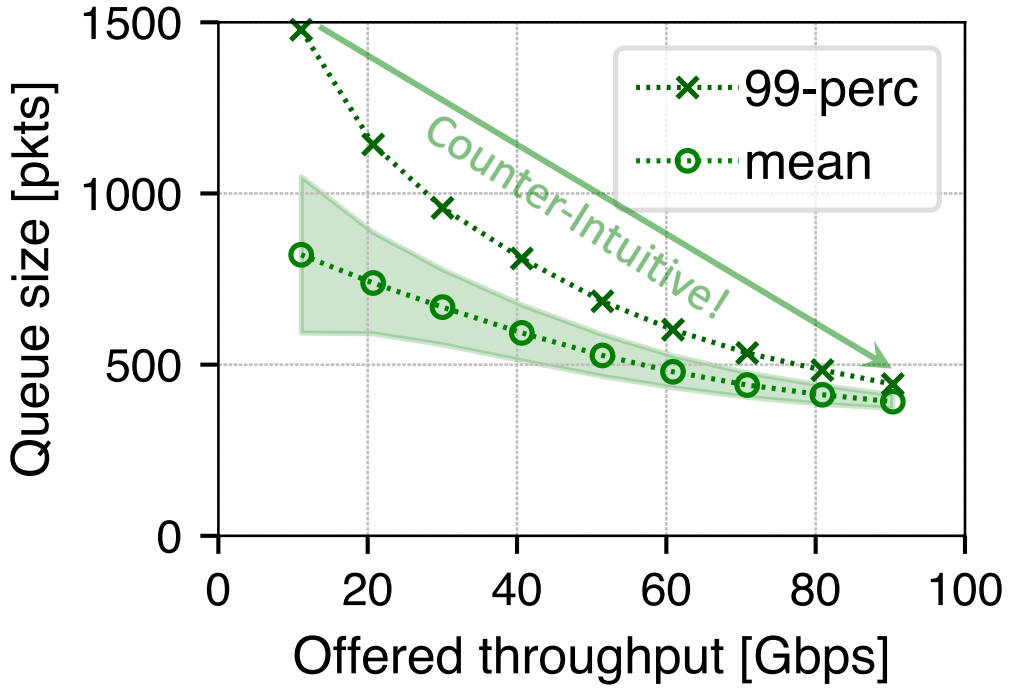
Create copies of payloads

- Python Simulator – 20µs NF Processing Latency
- 100Gbps w/ Poisson arrival (64B-1.5KB pkt size)
- Controller ⟹ 1µs Queue Stats Update Delay

Can TurboSwitch's controller enforce the correct delay to payloads?

TurboSwitch **correctly sizes** queues to **enforce** the NF processing latency!





Conclusions

The **TurboSwitch** system:

- Presents a packet-splitting approach that **does not require external resources**
- Proposes a **queue-based buffer** to store payloads on existing ASIC switches
- **Reduces** NF processing time with **zero configuration**
- Spurs a question on whether **expose APIs** for controlling the forwarding buffer