EURO/SYS'25

Occamy

A Preemptive Buffer Management for On-chip Shared-memory Switches

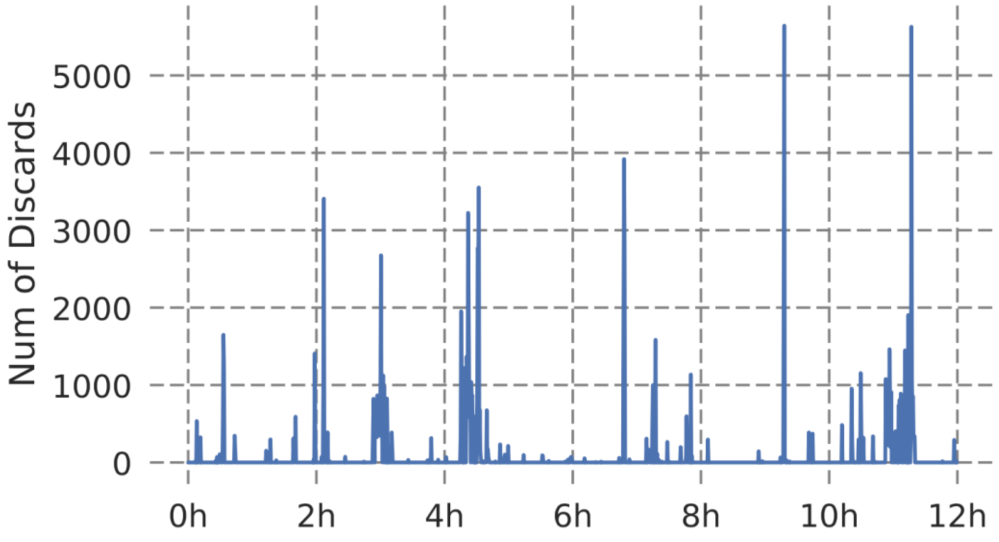
Danfeng Shan, Yunguang Li, Jinchao Ma, Zhenxing Zhang,
Zeyu Liang, Xinyu Wen, Hao Li, Wanchun Jiang, Nan Li, Fengyuan Ren

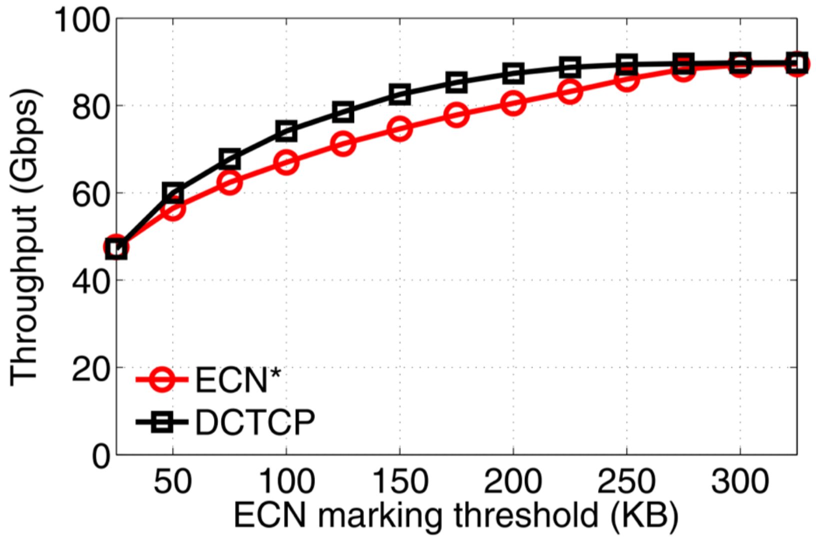
https://github.com/ants-xjtu/Occamy

西安交通大學 XI'AN JIAOTONG UNIVERSITY
HUAWEI
中南大學 CENTRAL SOUTH UNIVERSITY
清華大学 Tsinghua University

Switches and Buffers

- Switch Buffer
  - Short flows: Absorb transient bursts
  - Long flows: Maintain high-throughput





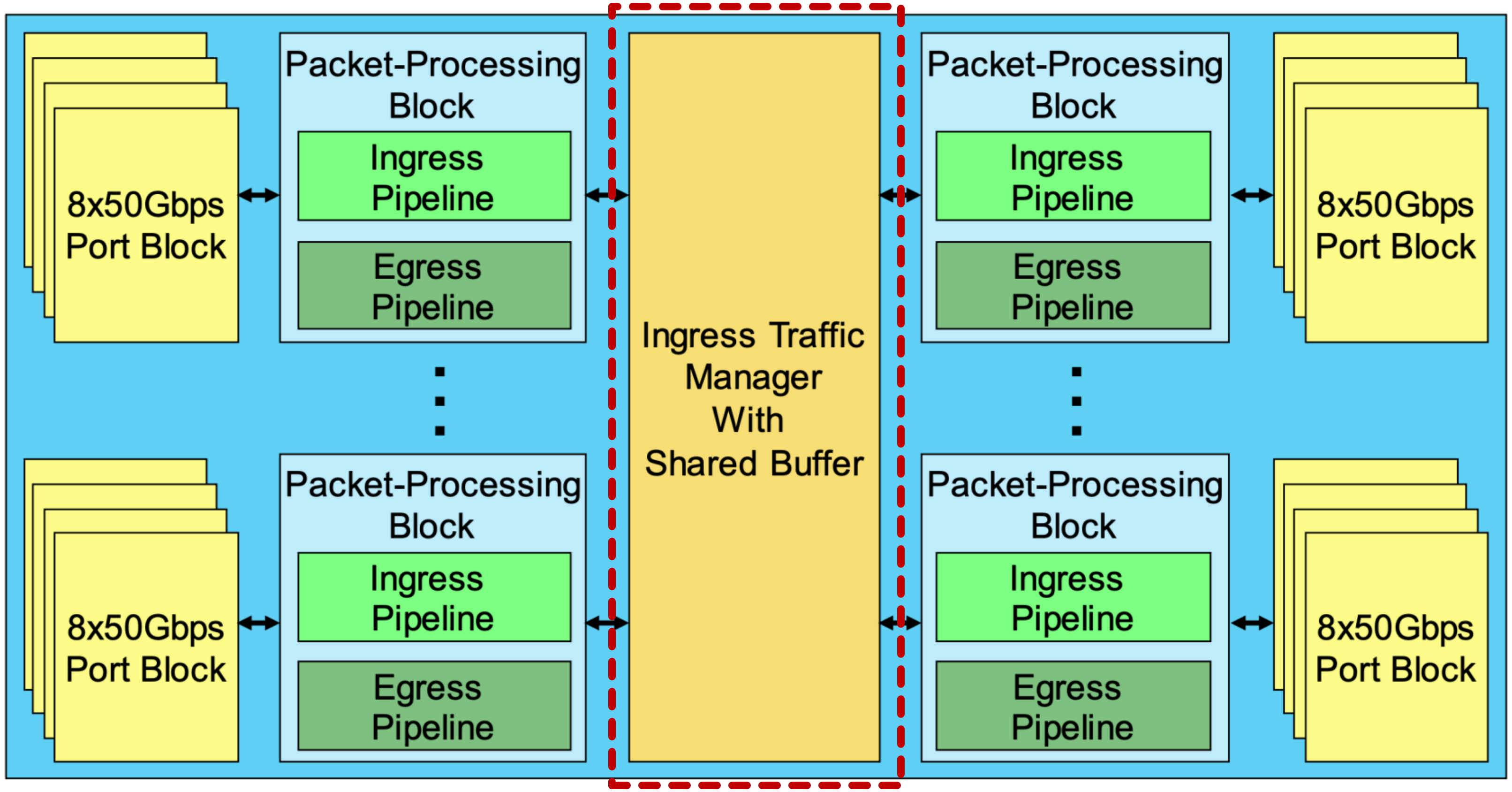
Microbursts encompass most congestion events[1]   DCTCP requires ~60-70% BDP buffering for 100% throughput[2]

[1] Zhang Q, Liu V, Zeng H, et al. High-resolution measurement of data center microbursts[C]//Proceedings of ACM IMC. 2017: 78-85.
[2] Bai W, Hu S, Chen K, et al. One more config is enough: Saving (DC) TCP for high-speed extremely shallow-buffered datacenters[J]. IEEE/ACM Transactions on Networking, 2020, 29(2): 489-502.

Switches and Buffers

- Today's DCN Switch: On-ship Shared-Memory

Globally shared on-chip packet buffer

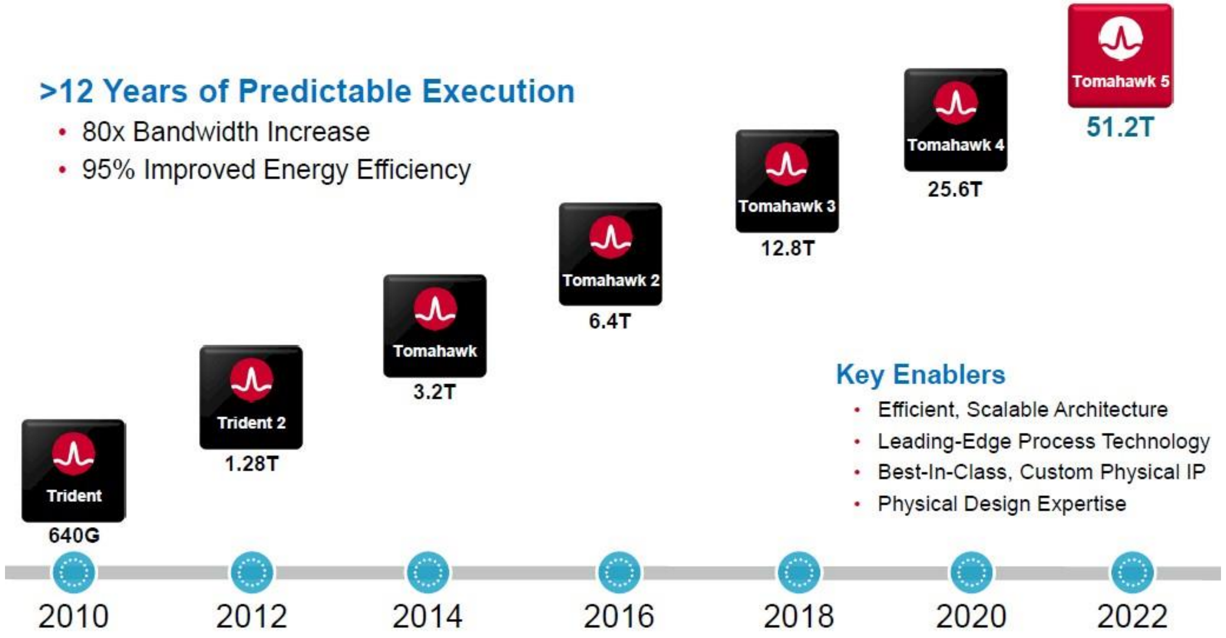


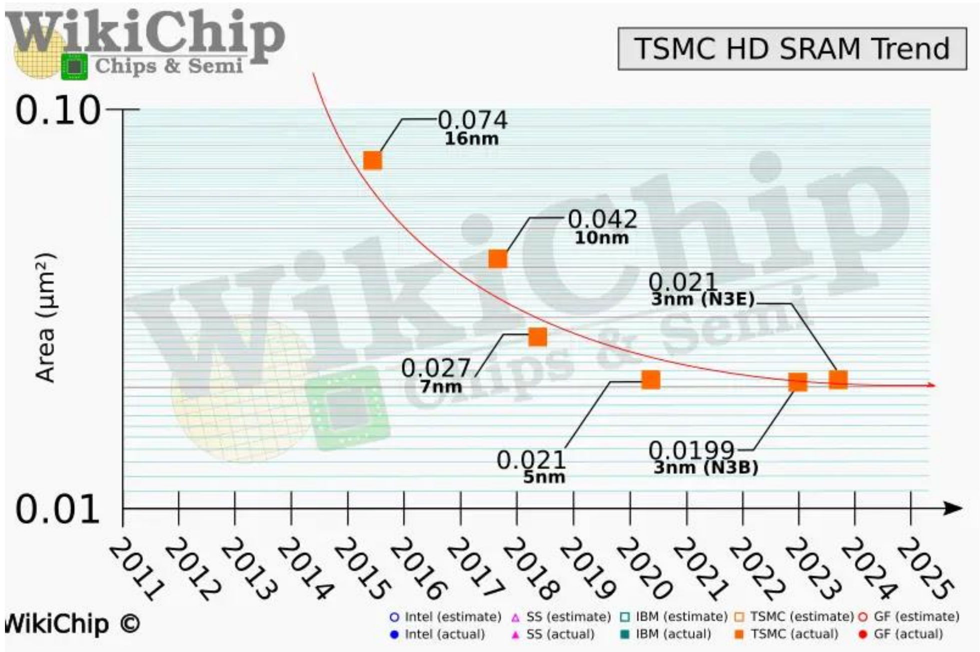
Broadcom Tomahawk 4 switch chip[1]

[1] https://docs.broadcom.com/docs/12398014

Switches and Buffers

☐ Trends of Switch Buffer





Doubling the switching capacity every two years[1]

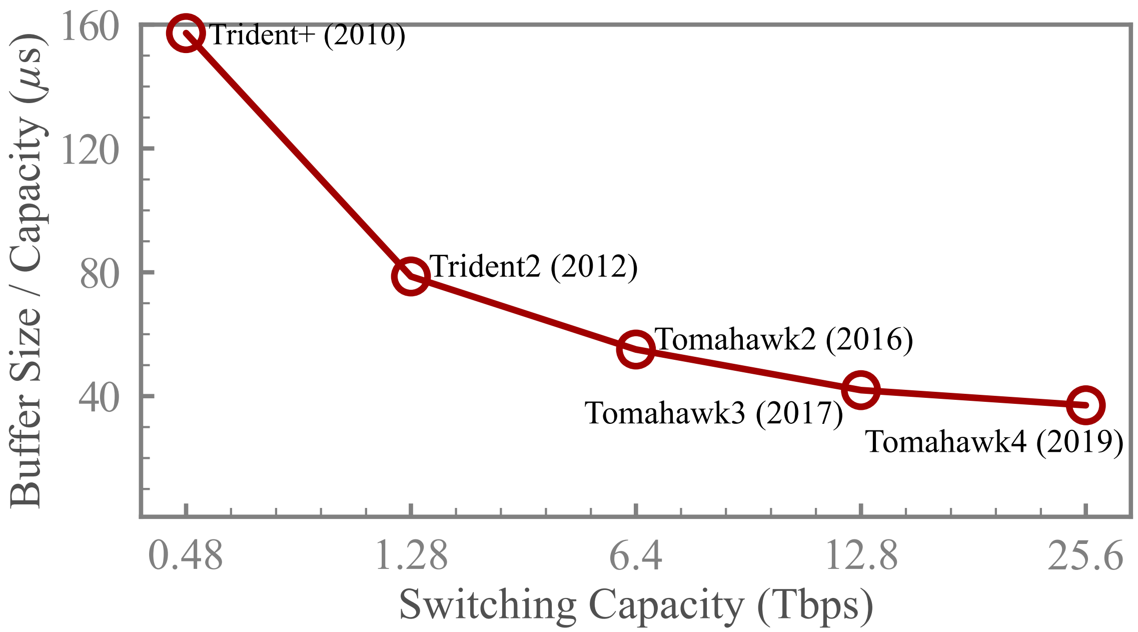
SRAM scaling appears to have completely collapsed[2]

[1] https://www.broadcom.com/blog/driving-the-data-center-into-the-future

[2] https://fuse.wikichip.org/news/7343/iedm-2022-did-we-just-witness-the-death-of-sram

Switches and Buffers

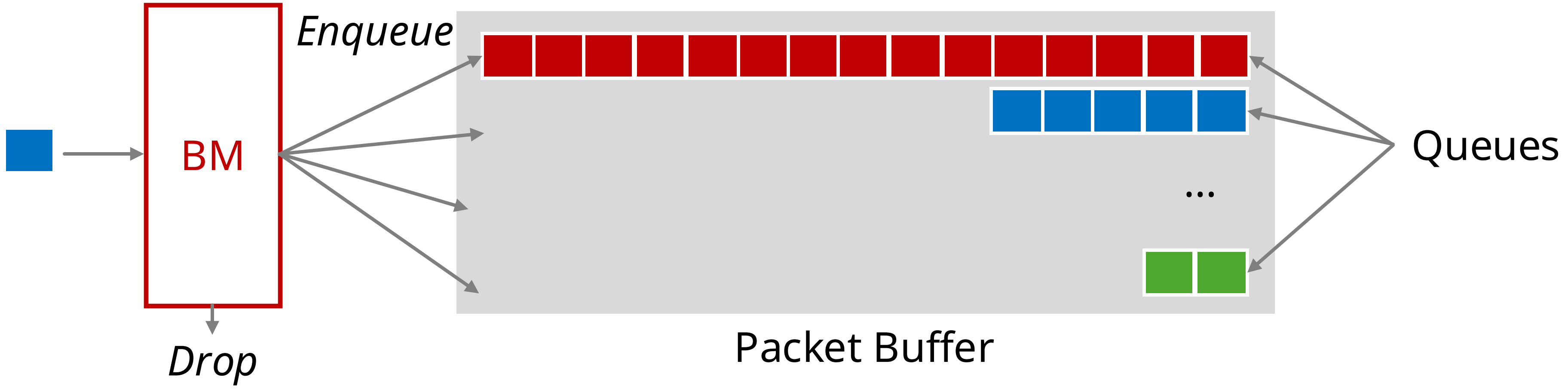
- Trends of Switch Buffer



The switch buffer (relative to capacity) has been decreased by 4x

Buffer Management (BM)

- Buffer Management (BM): Allocate buffer across queues

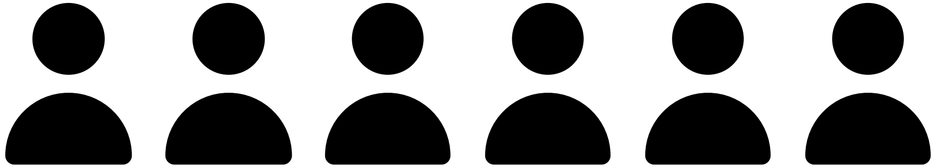


- Goals of BM
  - Fair: Don't starve queues when facing dynamic traffic
  - Efficient: Don't waste the scarce buffer for maximizing burst absorption
  - Simple: Easy to be implemented in high-speed switch chip

Buffer Management (BM)

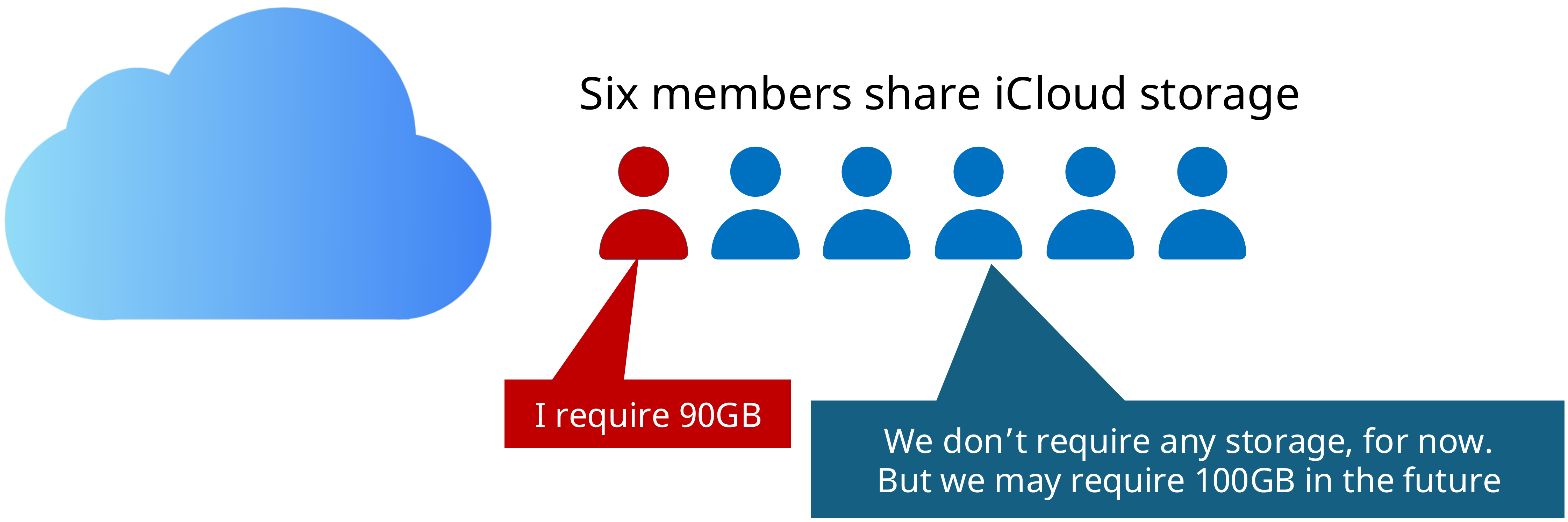
- Analogy: iCloud Storage Sharing

Six members share iCloud storage



Buffer Management (BM)

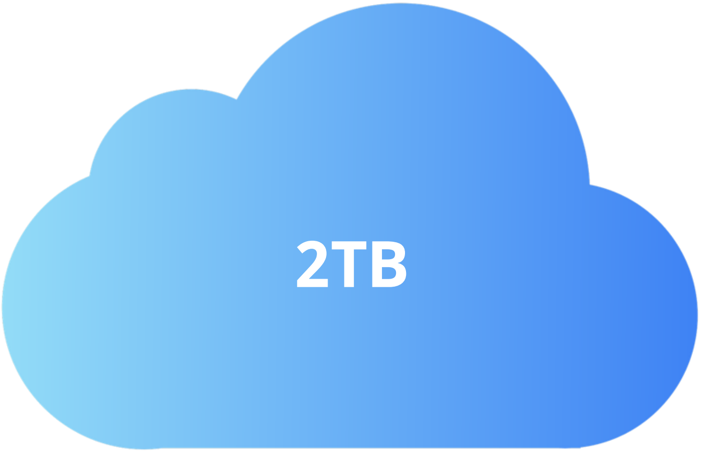
- Analogy: iCloud Storage Sharing



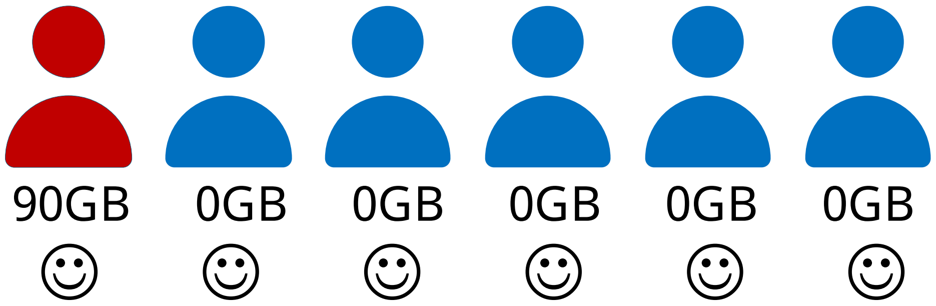
How to Share the iCloud Storage?

- Scheme 1: Sufficient Reservation
  - Example BMs: Complete Partition, DT with a small α

We are rich.
Let's just buy 2TB storage.
Everyone gets 333GB.



Six members share iCloud storage



✓ Fair ✓ Simple ✗ Efficient

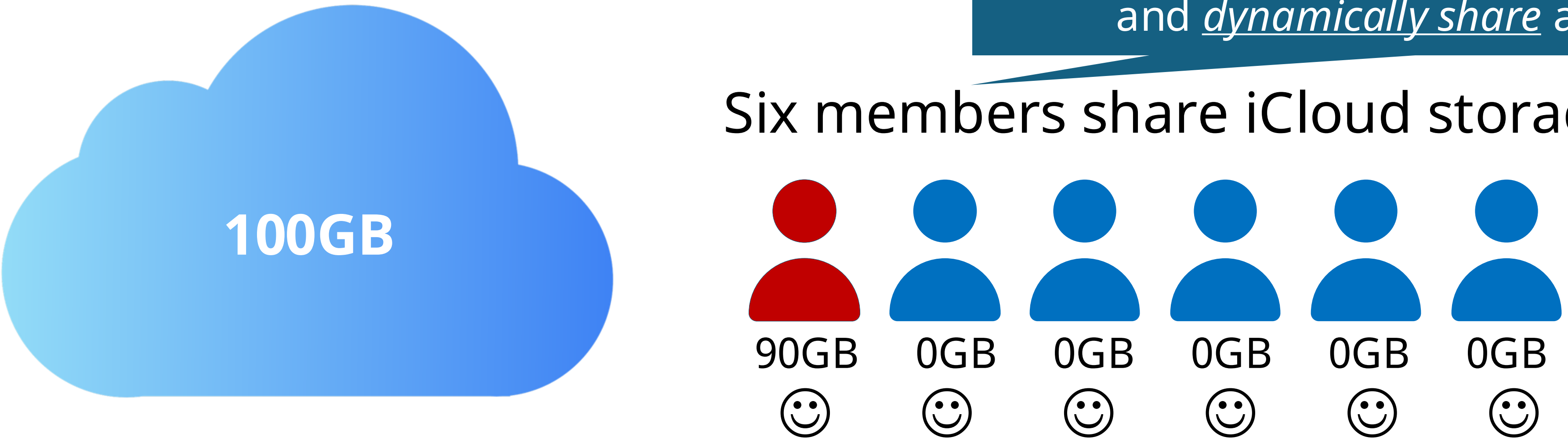
- Effective when buffer is sufficient
- Status quo: insufficient buffer

How to Share the iCloud Storage?

- Scheme 2: On-demand Allocation
  - Example BMs: Complete Sharing, DT with a large α

We are poor and can only afford 100GB storage.
Let’s buy 100GB storage,
and *dynamically share* among members

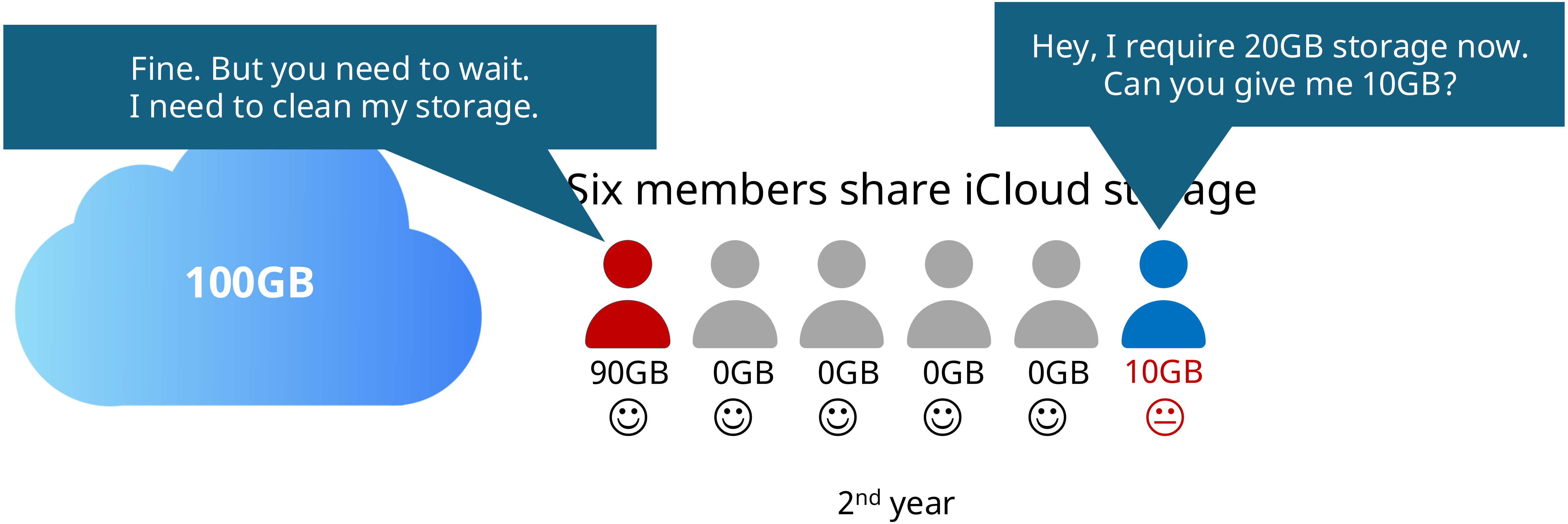
Six members share iCloud storage



1st year

How to Share the iCloud Storage?

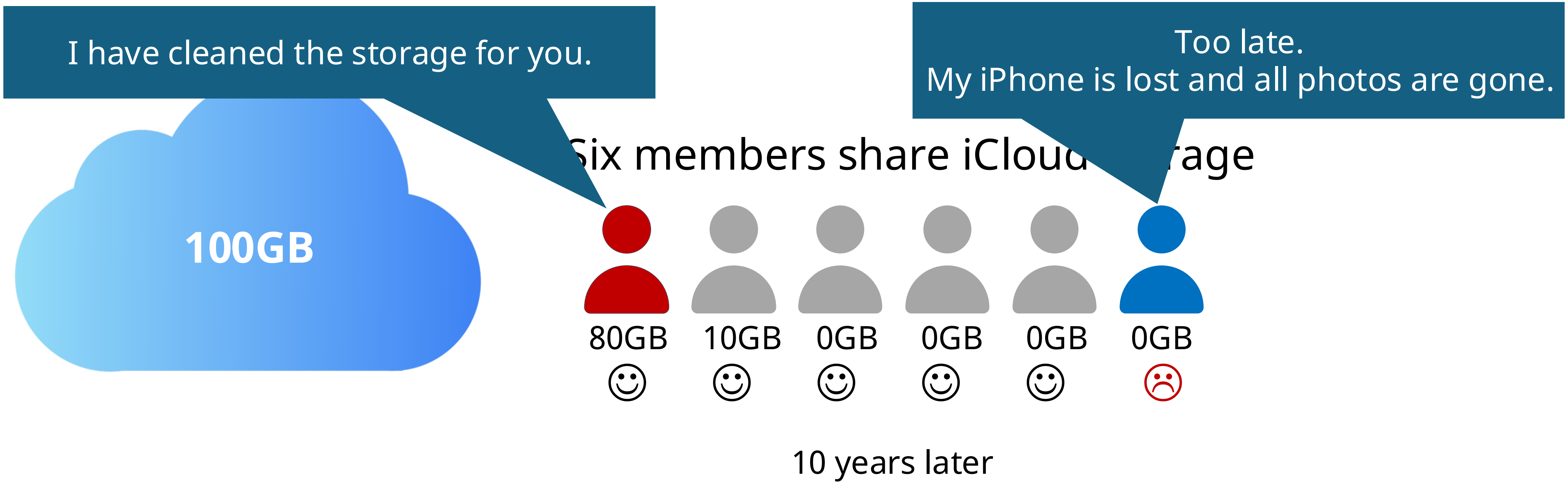
- Scheme 2: On-demand Allocation
  - Example BMs: Complete Sharing, DT with a large α



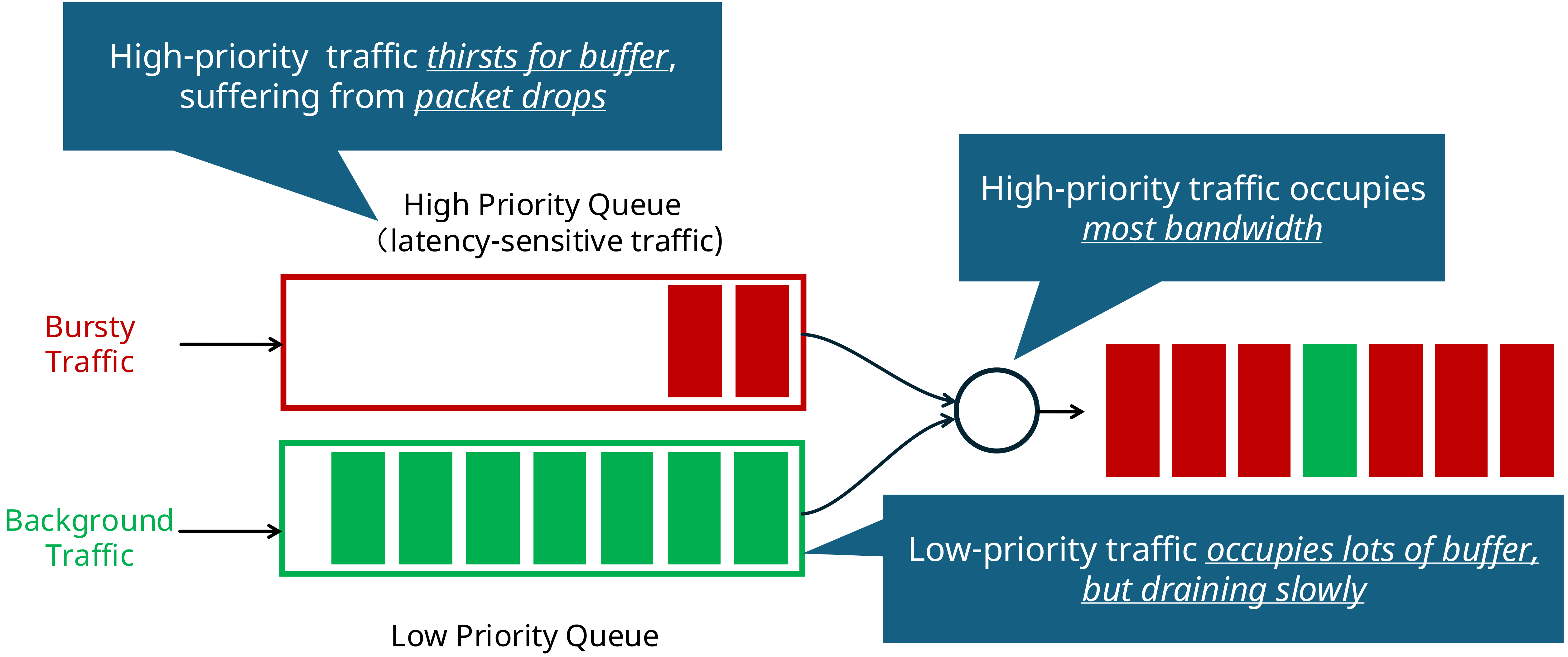
10 YEARS LATER

How to Share the iCloud Storage?

- Scheme 2: On-demand Allocation
  - Example BMs: Complete Sharing, DT with a large α



The Buffer Choking Problem

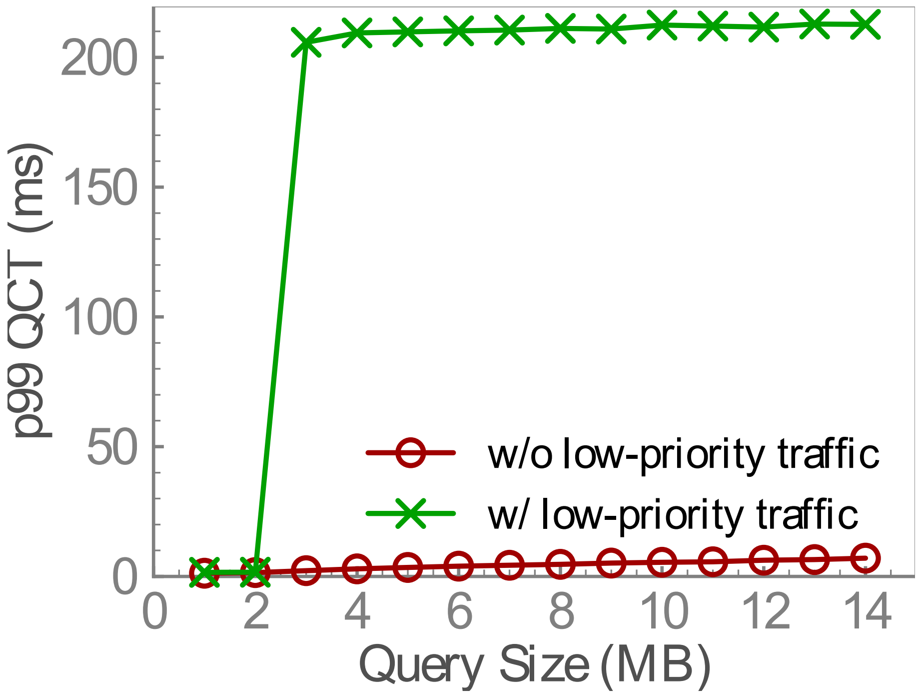


The Buffer Choking Problem

Experiments on Huawei CE6865 switch

Average Query Completion Time

degraded by up to 8×



99th Query Completion Time

degraded by up to 90×

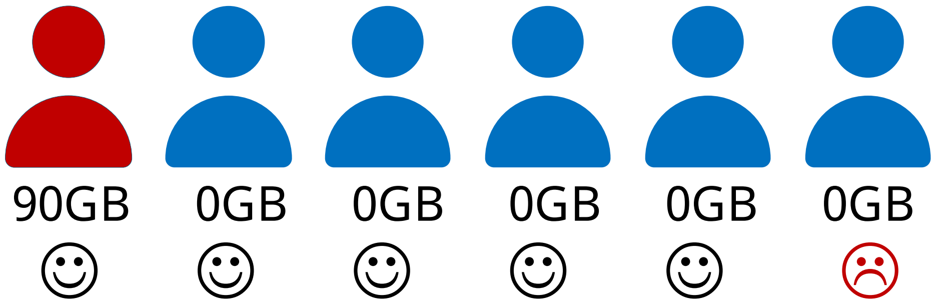
Buffer choking can significantly degrade the transmission performance

How to Share the iCloud Storage?

- Scheme 2: On-demand Allocation
  - Example BMs: Complete Sharing, DT with a large α

We are poor and can only afford 100GB
Let’s buy 100GB storage,
and *dynamically share* among members

Six members share iCloud storage



✓ Efficient ✓ Simple ✗ Fair

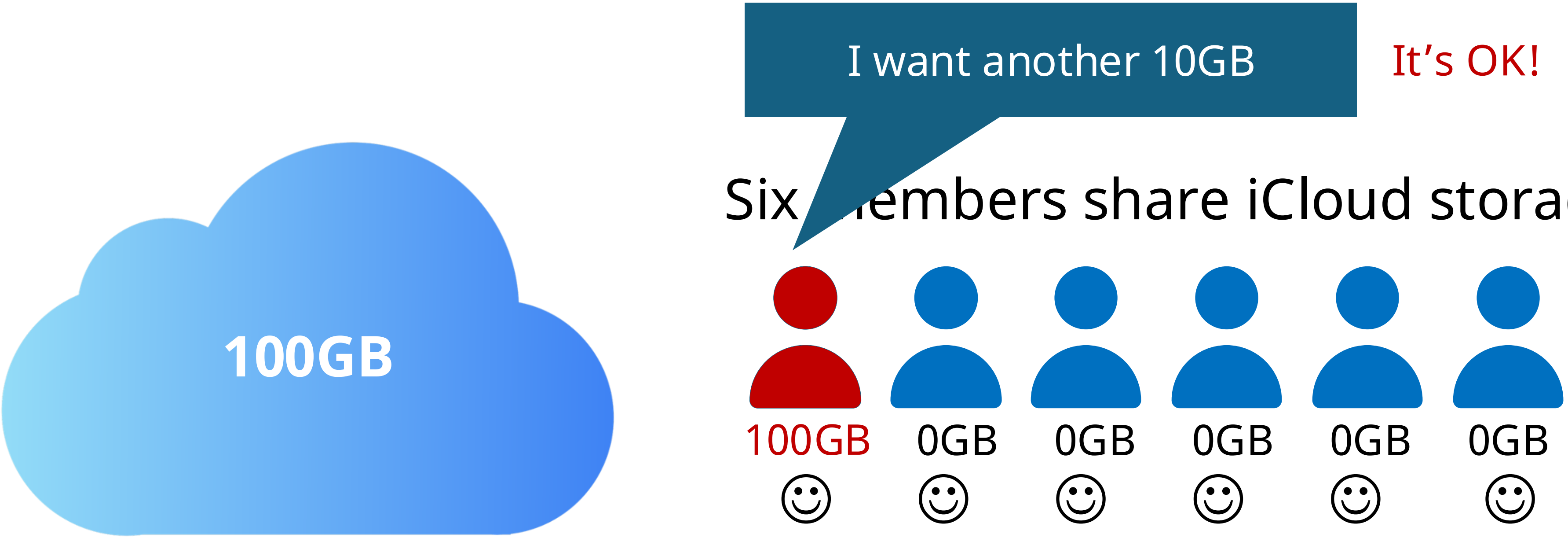
- Effective with very smooth traffic
- Status quo: highly bursty traffic (*e.g.*, 1000:1 incast)

How to Share the iCloud Storage?

- Why on-demand allocation is not fair?
  - Non-preemption: *Passively* wait for others to *naturally* free the space

How to Share the iCloud Storage?

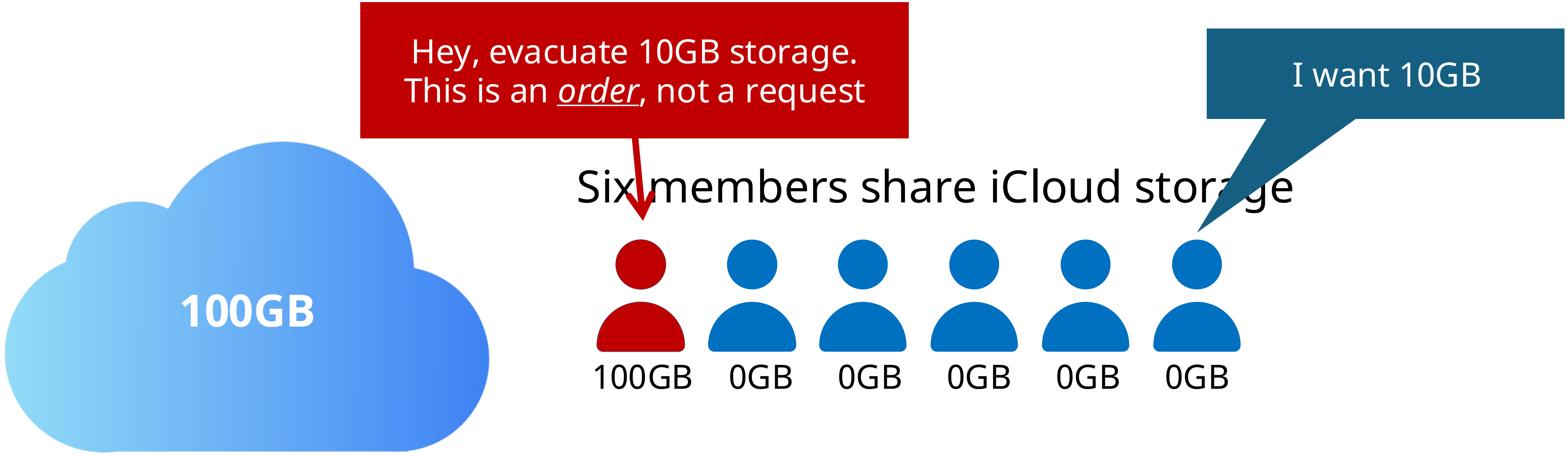
- An optimal scheme for (poor) people (*i.e.*, Pushout)



① Everyone can get space whenever there is free storage

How to Share the iCloud Storage?

- An optimal scheme for (poor) people (*i.e.*, Pushout)

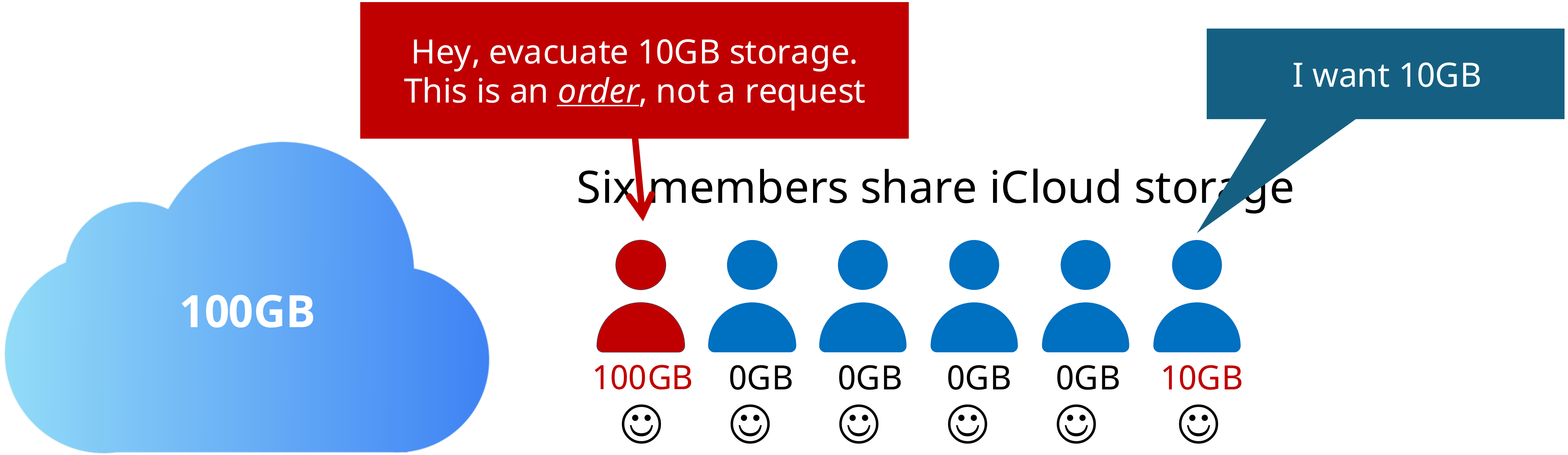


① Everyone can get space whenever there is free storage

② If someone requires space while storage is full, reclaim the storage of the person *using the most storage*.

How to Share the iCloud Storage?

- An optimal scheme for (poor) people (*i.e.*, Pushout)

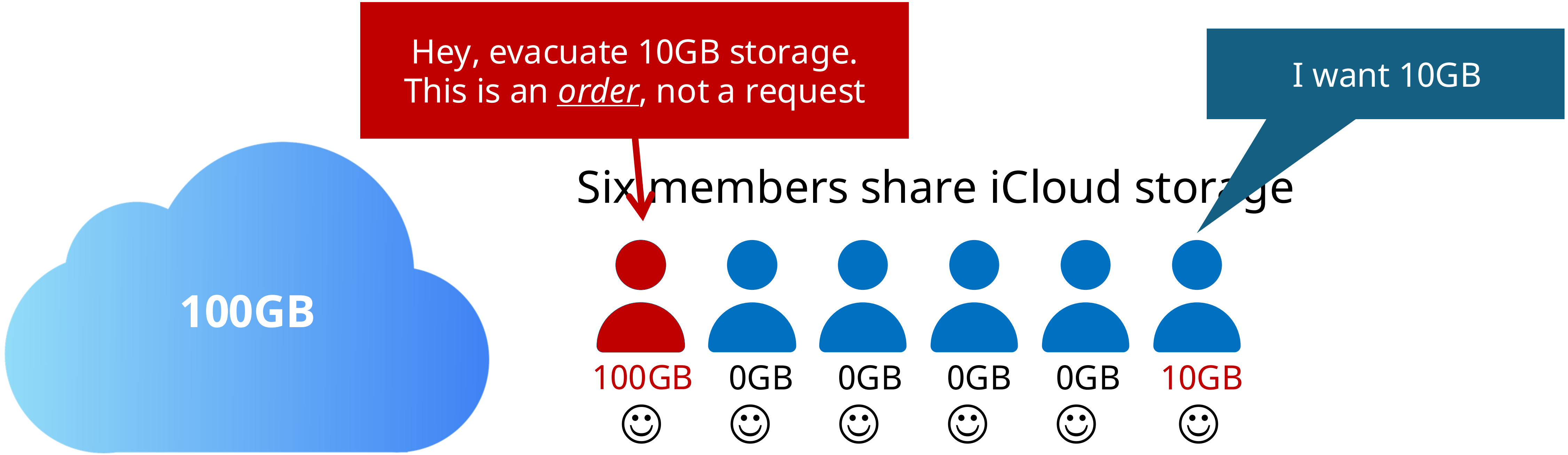


① Everyone can get space whenever there is free storage

② If someone requires space while storage is full, reclaim the storage of the person *using the most storage*.

How to Share the iCloud Storage?

- An optimal scheme for (poor) people (*i.e.*, Pushout)



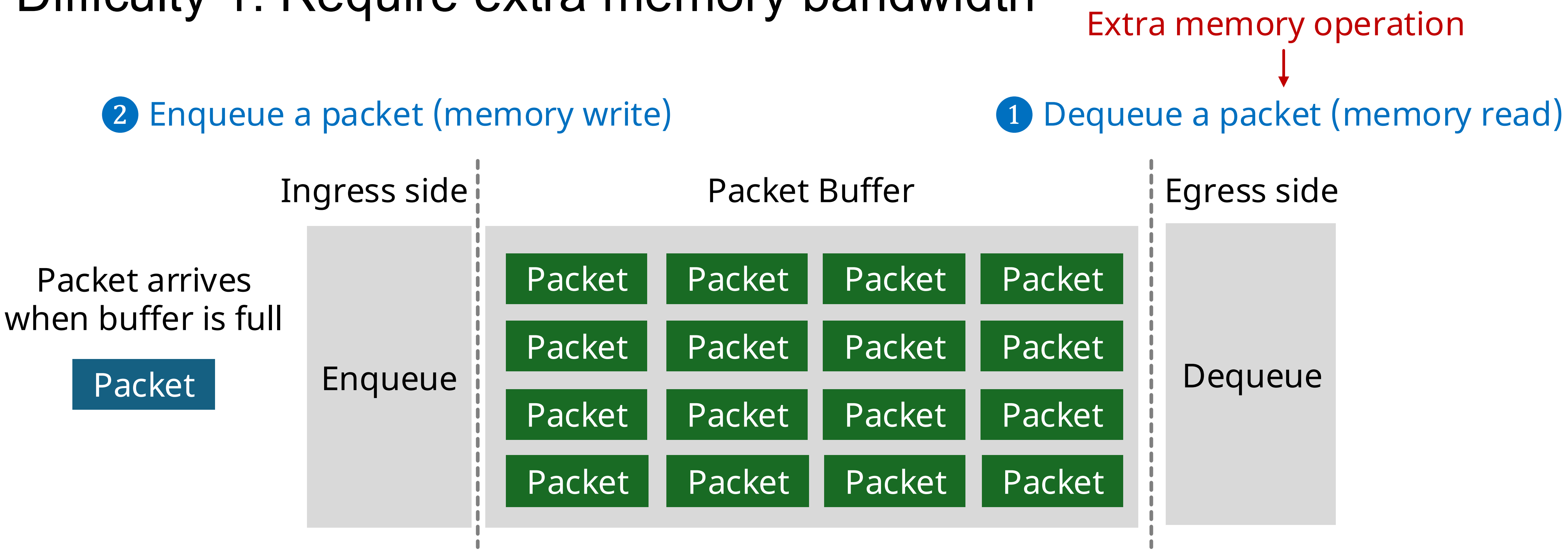
① Everyone can get space whenever there is free storage

② If storage is full and someone needs space, remove the data of the person *using the most storage*.

✓ Efficient ✓ Fair ✗ Simple

Why the Optimal Scheme is not Simple

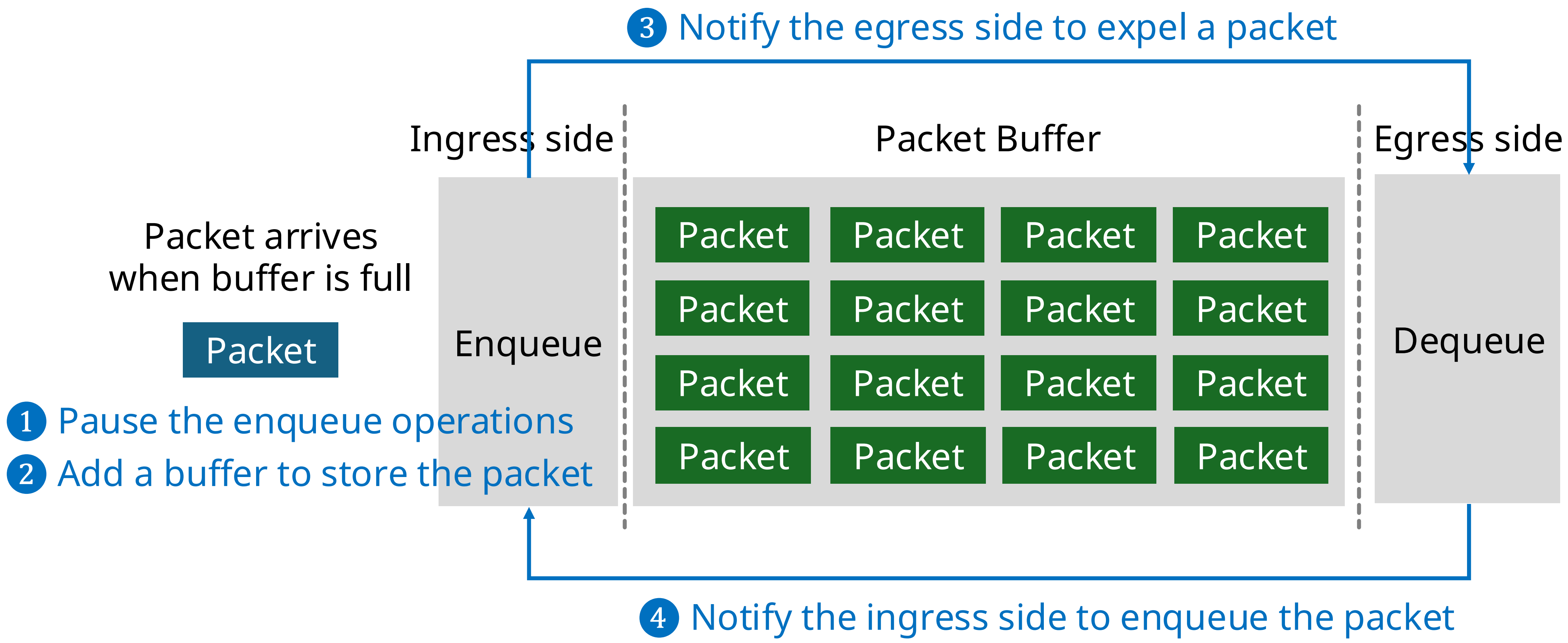
- Difficulty 1: Require extra memory bandwidth



- Unacceptable for traditional off-chip shared-memory switch
- Status quo: On-chip shared-memory switch *significantly extends memory bandwidth*

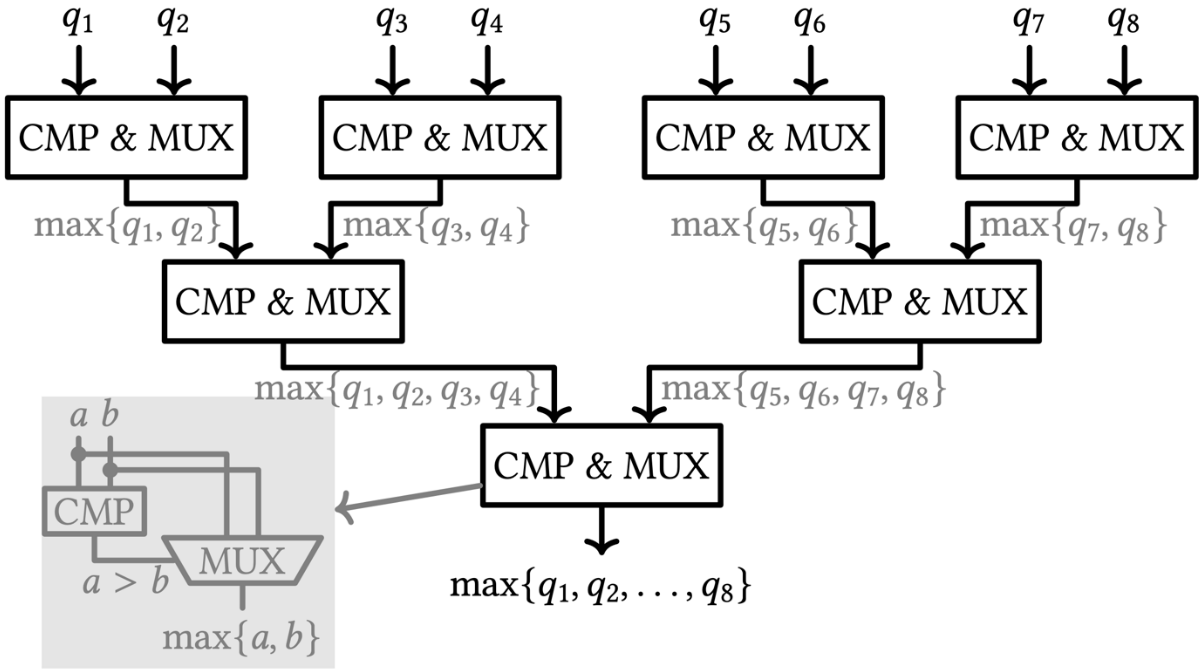
Why the Optimal Scheme is not Simple

- Difficulty 2: Require complex enqueue operations



Why the Optimal Scheme is not Simple

☐ Difficulty 3: Monitoring the longest queue in real time



8-input maximum finder based on binary comparator tree

Area complexity: $O(kN)$ ✓ Acceptable

Time complexity: $O(\log_2 K \times \log_2 N)$ X Unacceptable

Example

- 64 queues, 16-bit queue length → *7ns* latency
- Queue length changes *every 1ns*

Occamy

A preemptive buffer management scheme

- ✓ Efficient: (Almost) fully utilize the buffer
- ✓ Fair: Quickly adjust the buffer allocation
- ✓ Simple: Easy to be implemented in switch chip

Occamy

- Expels packets in a round-robin manner

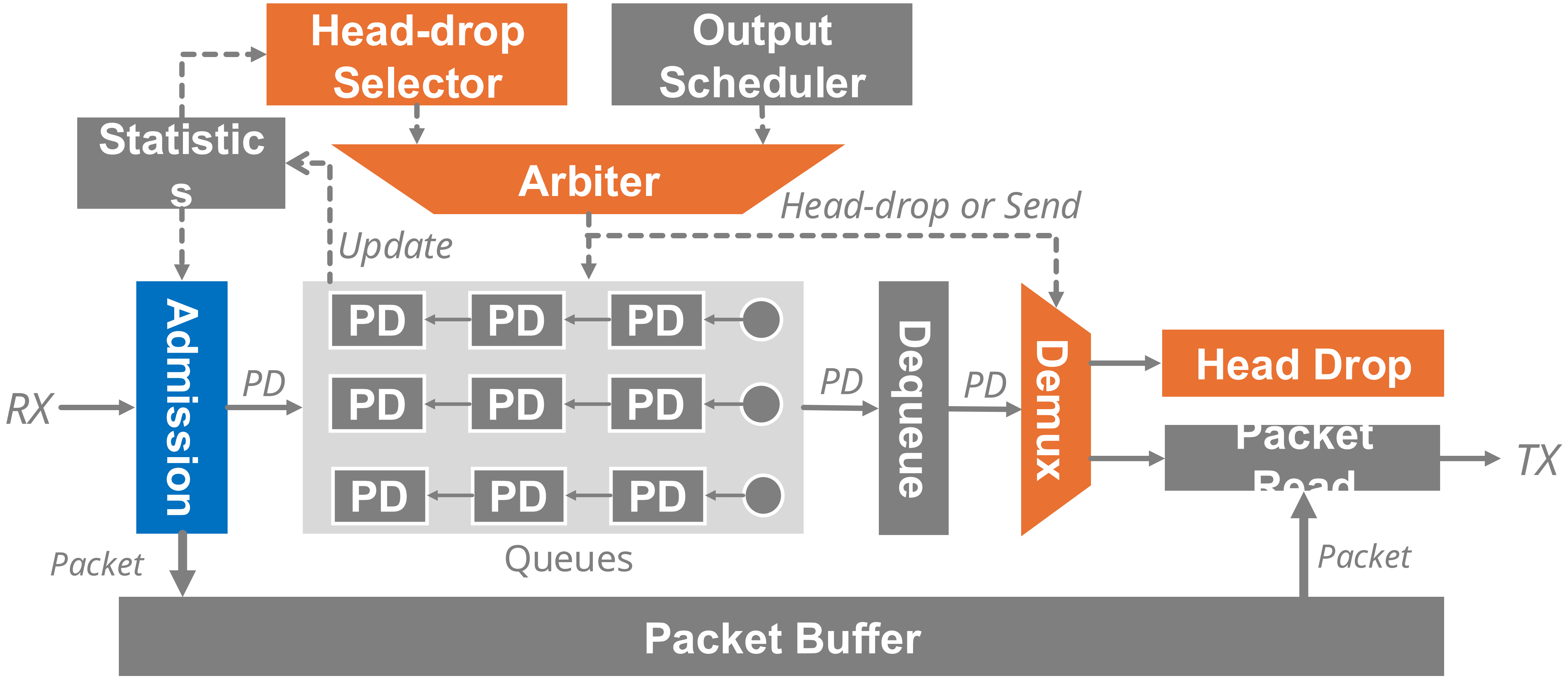
Overcomes the 3rd difficulty

- Proactively reserves a small fraction of free buffer

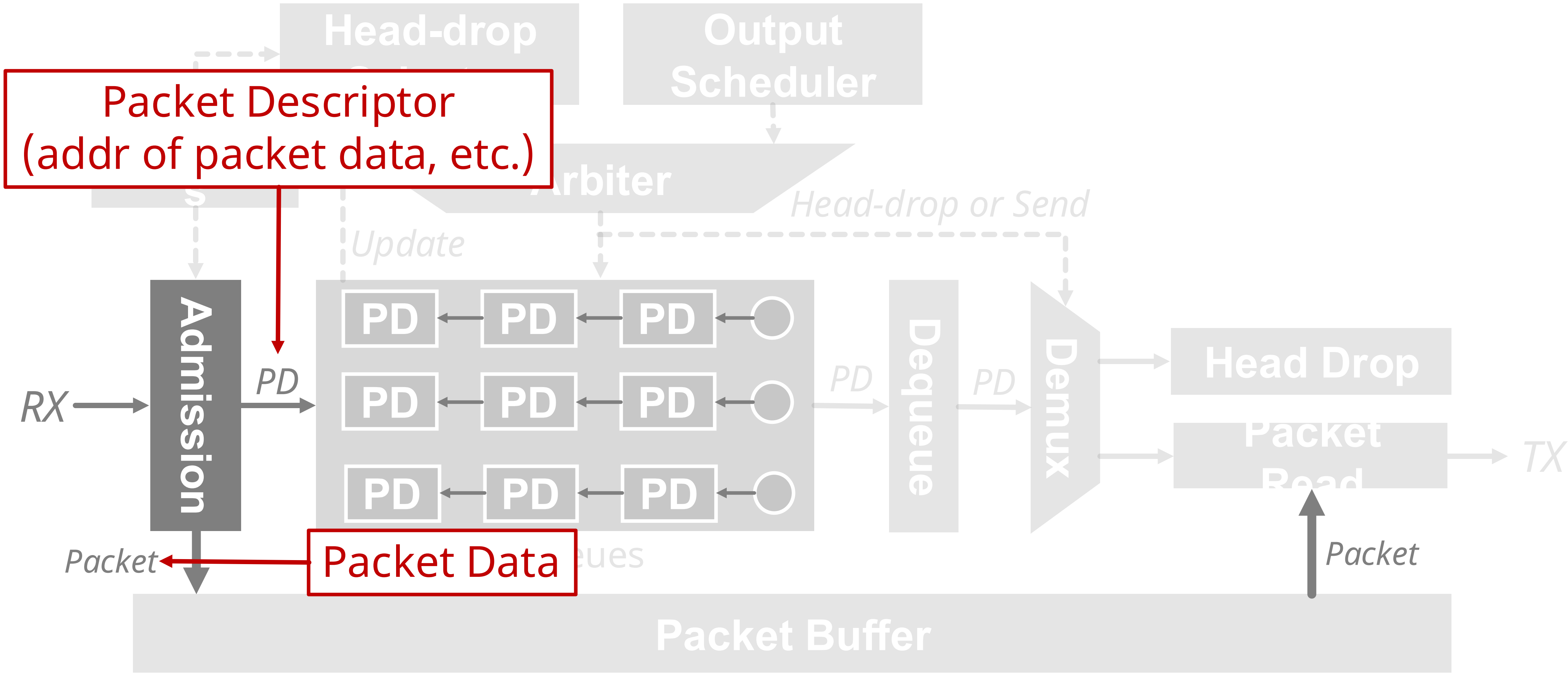
Overcomes the 2nd difficulty

- Keeps admission and expulsion mutually independent

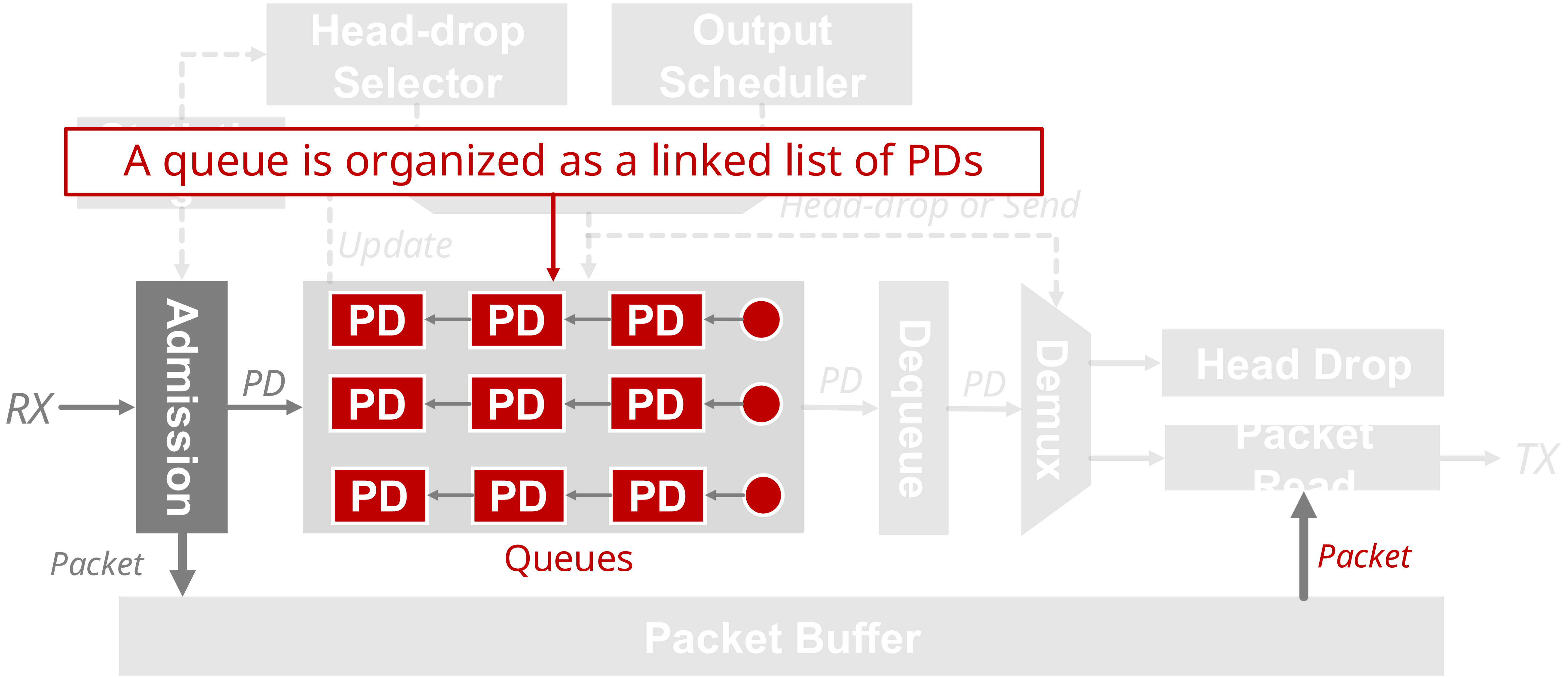
Occamy


Head-drop Selector
Output Scheduler
Statistic s
Arbiter
Head-drop or Send
Update
RX
Admission
PD
PD PD PD
PD PD PD
PD PD PD
Queues
PD
Dequeue
PD
Demux
Head Drop
Packet Read
TX
Packet
Packet
Packet Buffer

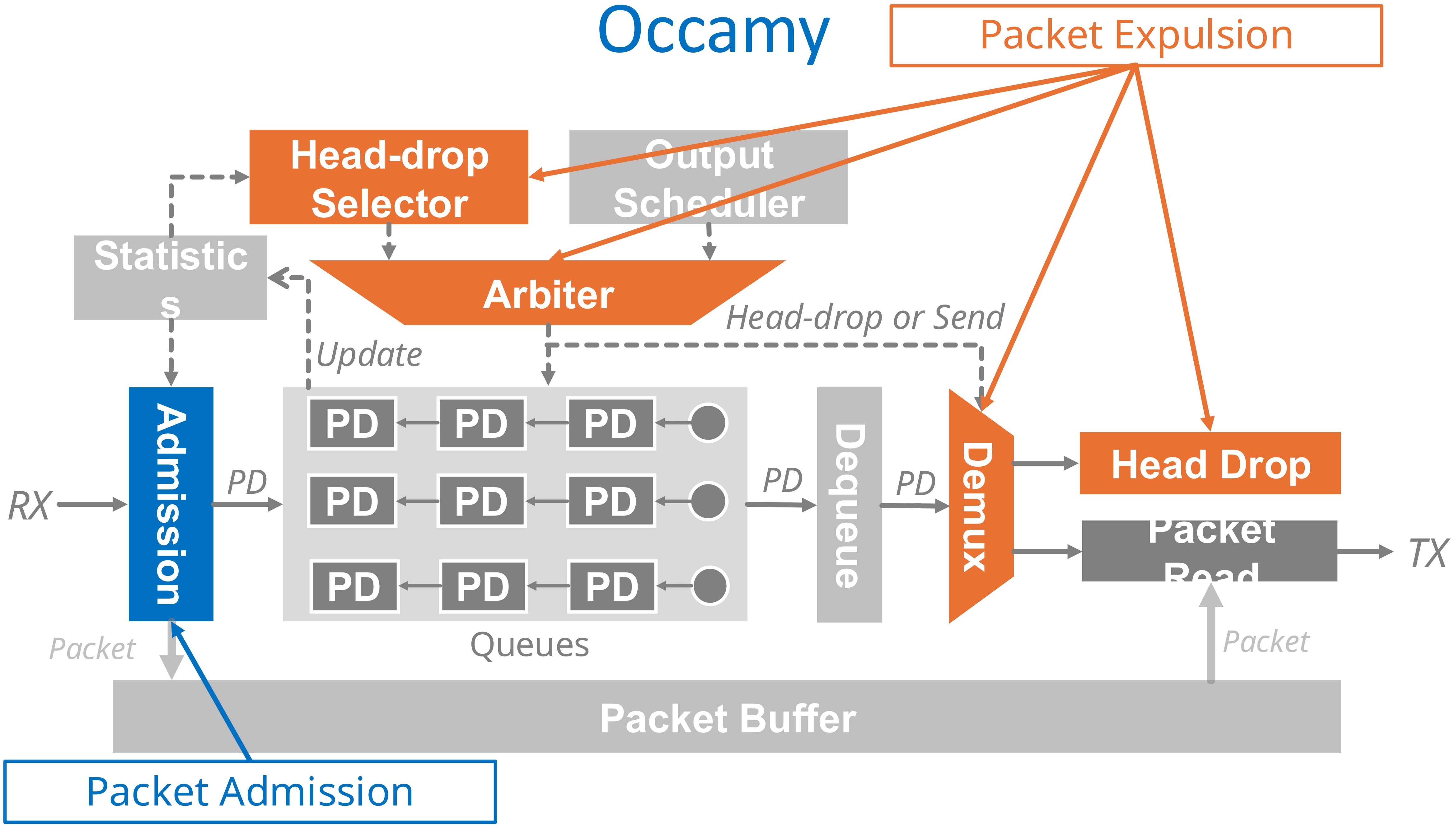
Occamy


Head-drop
Output Scheduler
Packet Descriptor
(addr of packet data, etc.)
Head-drop or Send
Update
RX
Admission
PD
PD
PD
PD
PD
PD
PD
PD
PD
PD
PD
PD
Dequeue
PD
Demux
Head Drop
Packet
TX
Packet
Packet Data
Packet
Packet Buffer

Occamy



Head-drop Selector
Output Scheduler
A queue is organized as a linked list of PDs
Head-drop or Send
Update
RX
Admission
PD
PD PD PD
PD PD PD
PD PD PD
PD
Dequeue
PD
Demux
Head Drop
Packet Read
TX
Packet
Queues
Packet
Packet Buffer

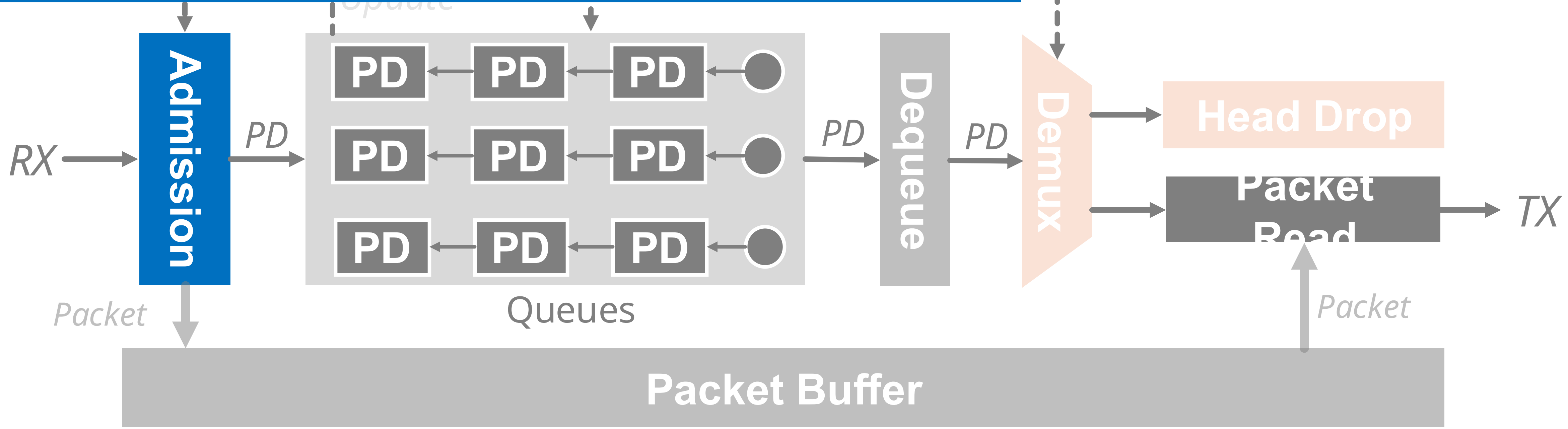

Occamy
Packet Expulsion
Head-drop Selector
Output Scheduler
Statistics
Arbiter
Head-drop or Send
Update
Admission
RX
PD
PD
PD
PD
PD
PD
PD
PD
PD
PD
PD
PD
Queues
Dequeue
Demux
Head Drop
Packet Read
TX
Packet
Packet
Packet Buffer
Packet Admission

Occamy

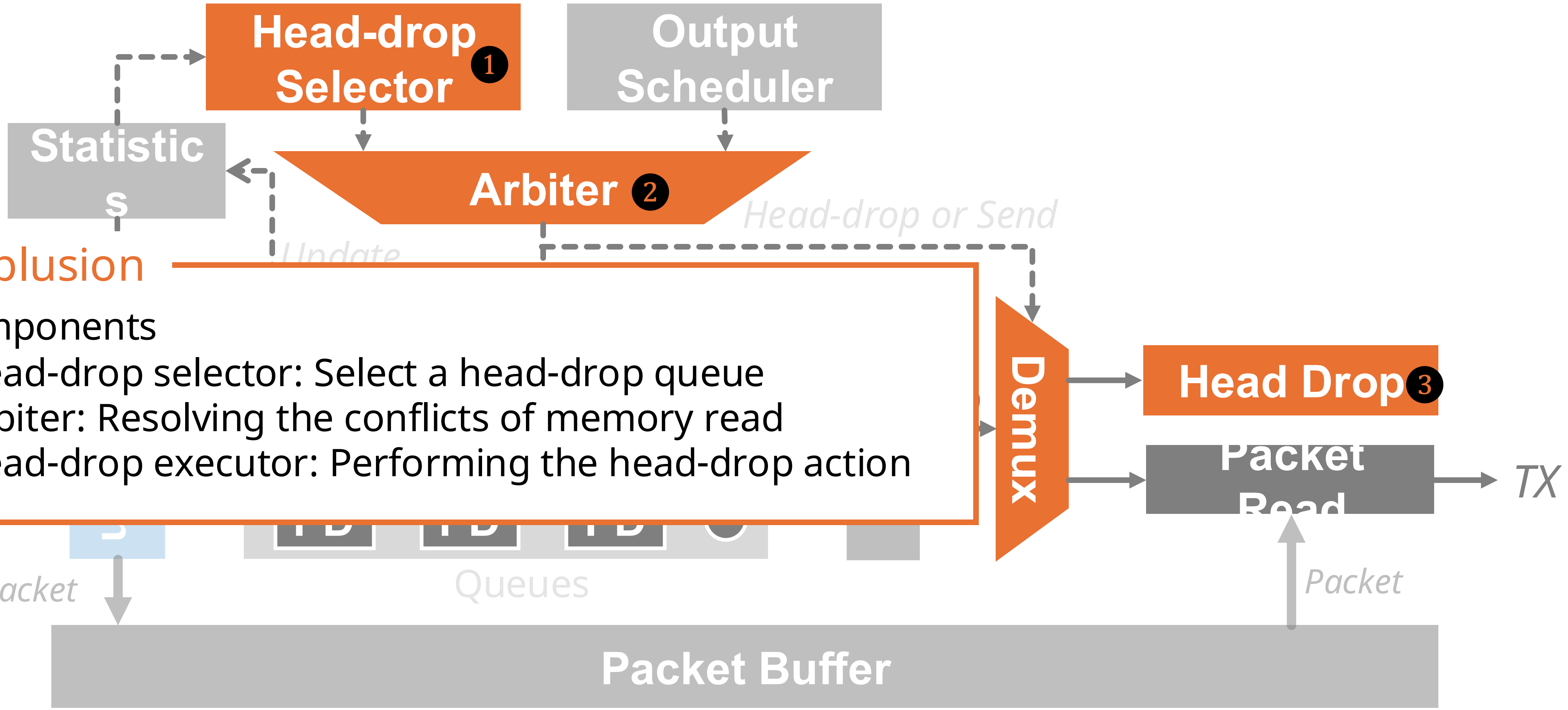
Admission

Utilize *off-the-shelf BM* in commodity switch chips

- DT：Widely adopted in the commodity switch chip
- α=8：Achieve an efficiency of 88.9% in the worst case



Occamy



Explusion

3 components

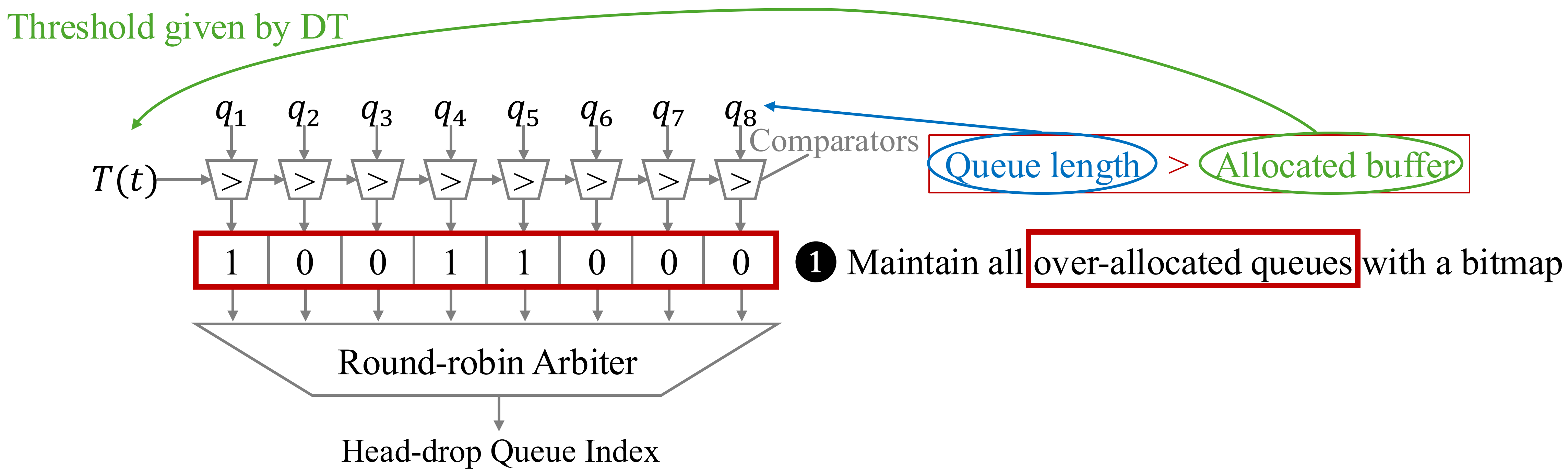
① Head-drop selector: Select a head-drop queue

② Arbiter: Resolving the conflicts of memory read

③ Head-drop executor: Performing the head-drop action

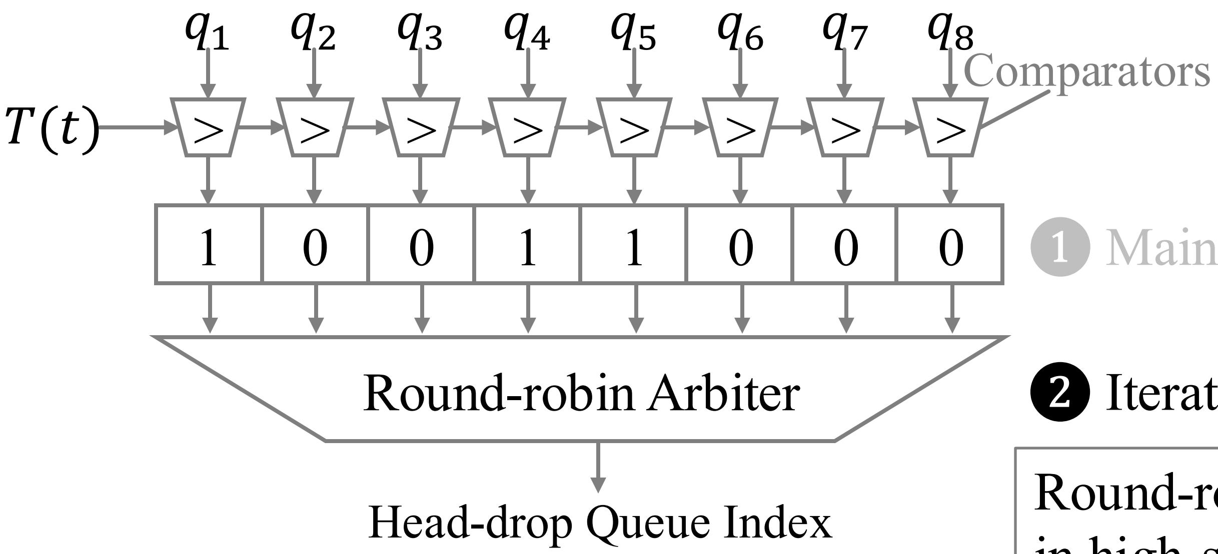
Occamy

- Head-drop selector: select a head-drop queue



Occamy

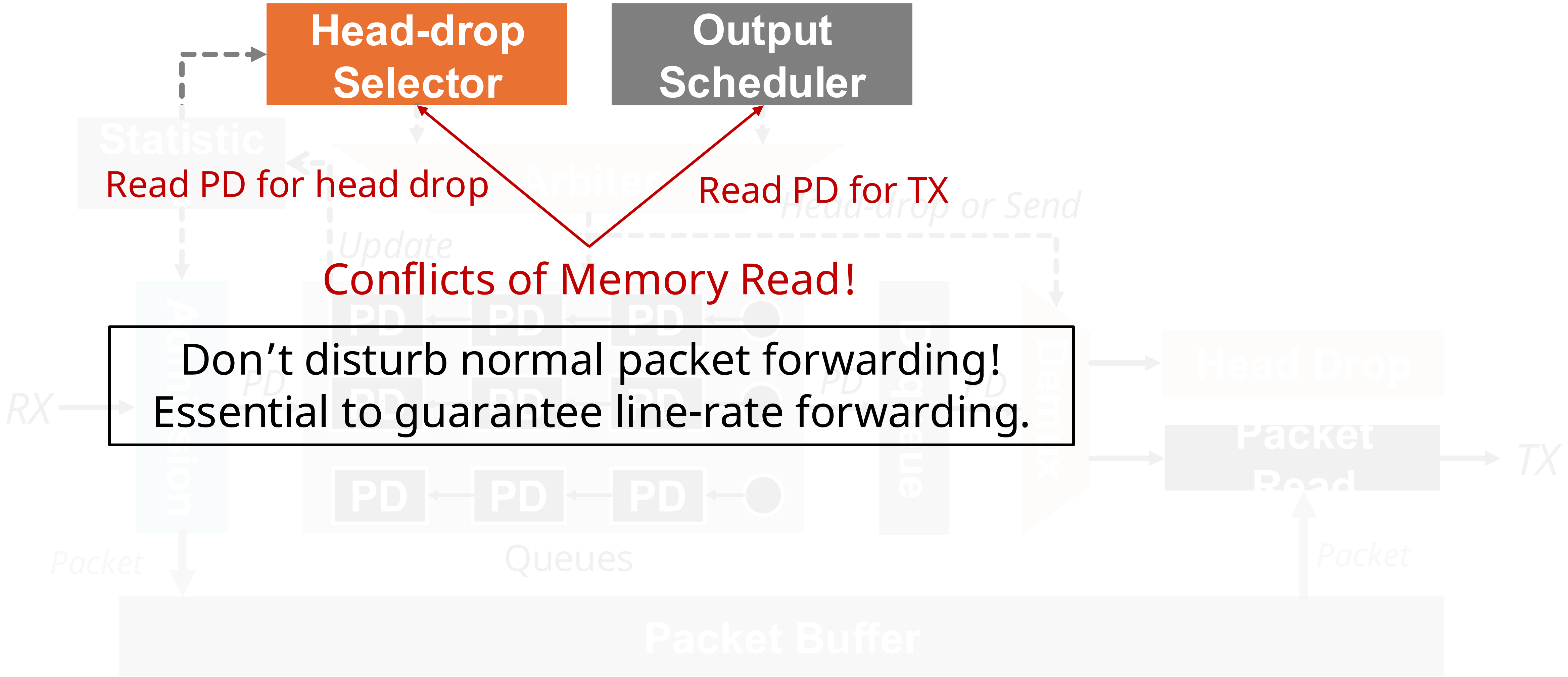
- Head-drop selector: select a head-drop queue



1. Maintain all over-allocated queues with a bitmap
2. Iterates over over-allocated queues

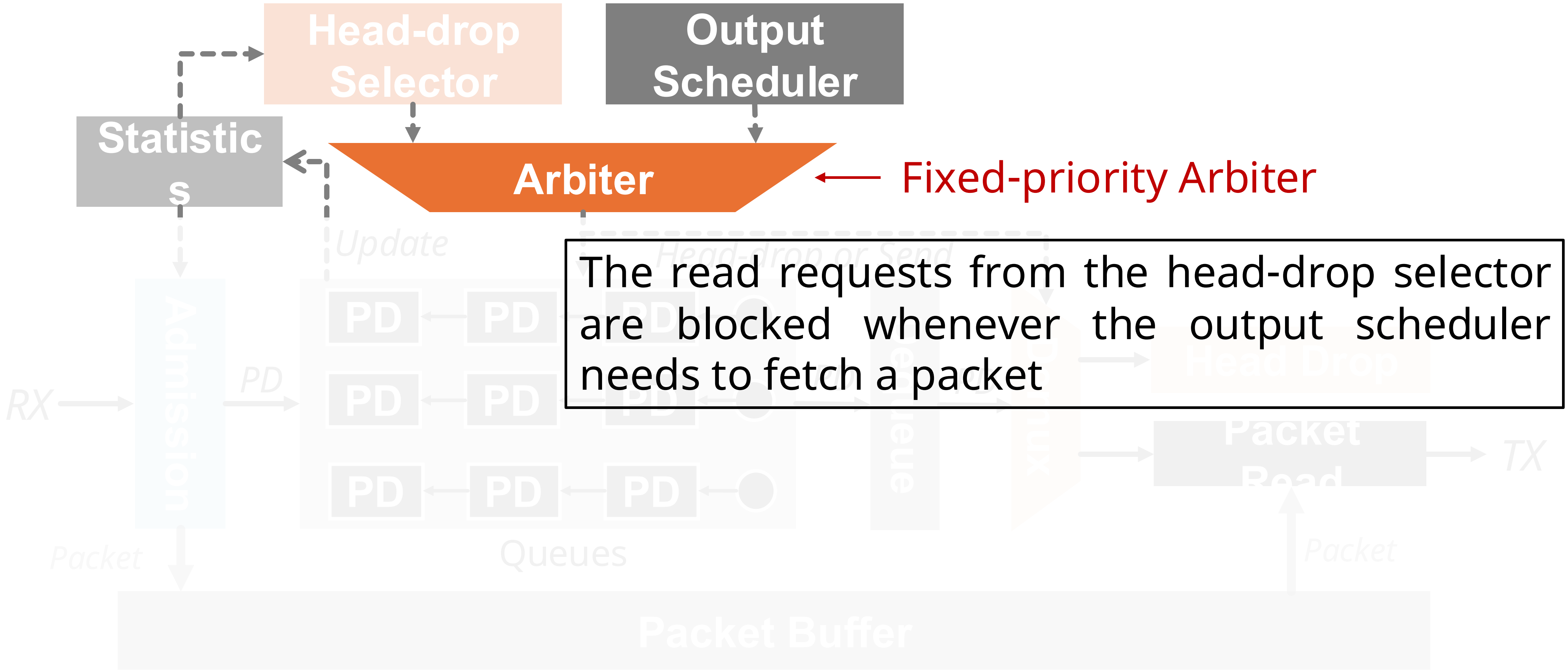
Round-robin arbiter: a common hardware component in high-speed crossbar switches

Occamy



Head-drop Selector
Output Scheduler
Read PD for head drop
Read PD for TX
Conflicts of Memory Read!
Don’t disturb normal packet forwarding!
Essential to guarantee line-rate forwarding.

Occamy

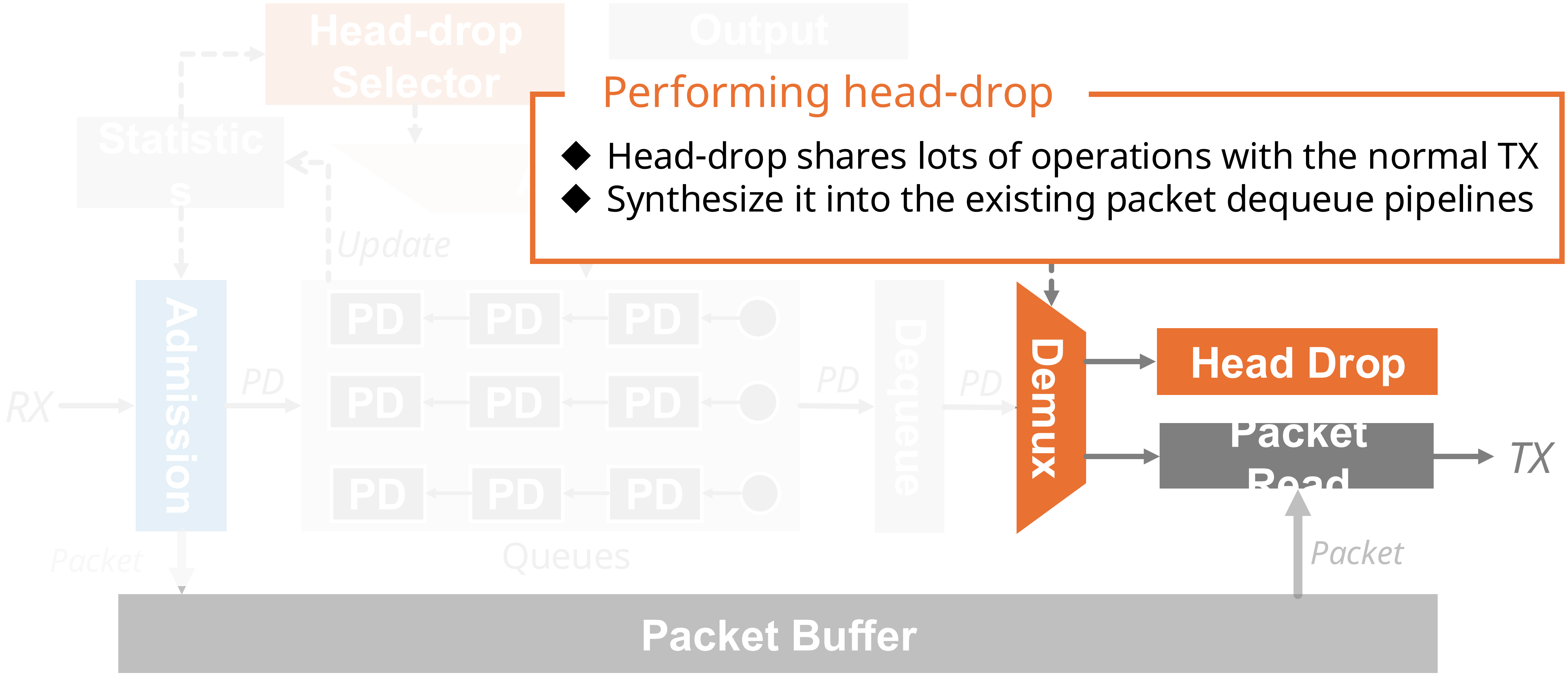


The read requests from the head-drop selector are blocked whenever the output scheduler needs to fetch a packet

Occamy

Performing head-drop

- Head-drop shares lots of operations with the normal TX
- Synthesize it into the existing packet dequeue pipelines



Occamy

	Cycle 1	Cycle 2	Cycle 3	Cycle 4
PD Memory	① Read PD	② Dequeue PD	Read Next PD	Dequeue Next PD
Cell Pointer Memory	Read Prev. Cell Ptr	③Read Cell Ptr	③Read Cell Ptr	Read Next Cell Ptr
	Free Prev. Cell	Free Prev. Cell	④ Free Cell	④ Free Cell
Cell Data Memory	Read Prev. Cell Data	Read Prev. Cell Data	⑤ Read Cell Data	⑤ Read Cell Data

TX pipeline

① Read a PD from PD memory
② Dequeue the PD
③ Read cell pointer from cell pointer memory
④ Fee cell (by moving the cell pointer to the free cell ptr list)
⑤ Read cell data

Occamy

	Cycle 1	Cycle 2	Cycle 3	Cycle 4
PD Memory	① Read PD	② Dequeue PD	Read Next PD	Dequeue Next PD
Cell Pointer Memory	Read Prev. Cell Ptr	③Read Cell Ptr	③Read Cell Ptr	Read Next Cell Ptr
	Free Prev. Cell	Free Prev. Cell	④ Free Cell	④ Free Cell
Cell Data Memory	Read Prev. Cell Data	Read Prev. Cell Data	⑤ Read Cell Data	⑤ Read Cell Data

Head-drop pipeline

① Read a PD from PD memory
② Dequeue the PD
③ Read cell pointer from cell pointer memory
④ Fee cell (by moving the cell pointer to the free cell ptr list)
~~⑤ Read cell data~~

Occamy

	Cycle 1	Cycle 2	Cycle 3	Cycle 4
PD Memory	① Read PD	② Dequeue PD	Read Next PD	Dequeue Next PD
Cell Pointer Memory	Read Prev. Cell Ptr	③Read Cell Ptr	③Read Cell Ptr	Read Next Cell Ptr
	Free Prev. Cell	Free Prev. Cell	④ Free Cell	④ Free Cell
Cell Data Memory	Read Prev. Cell Data	Read Prev. Cell Data	⑤ Read Cell Data	⑤ Read Cell Data

Synthesized pipeline

① Read a PD from PD memory
② Dequeue the PD
③ Read cell pointer from cell pointer memory
④ Fee cell (by moving the cell pointer to the free cell ptr list)
⑤ Read cell data if TX

Implementations

- Verilog implementation of core components
- P4-based hardware prototype
- DPDK-based software prototype
- Ns-3-based Simulator

https://github.com/ants-xjtu/Occamy

Evaluations

Module	FPGA Cost		ASIC Cost		
	LUTs	Flip Flops	Timing (ns)	Area (mm^2)	Power (mW)
Selector	1262	47	1.49	0.023	0.895
Arbiter	3	0	0.17	2.3e-5	0.003
Executor	47	7	0.38	7.3e-4	0.044

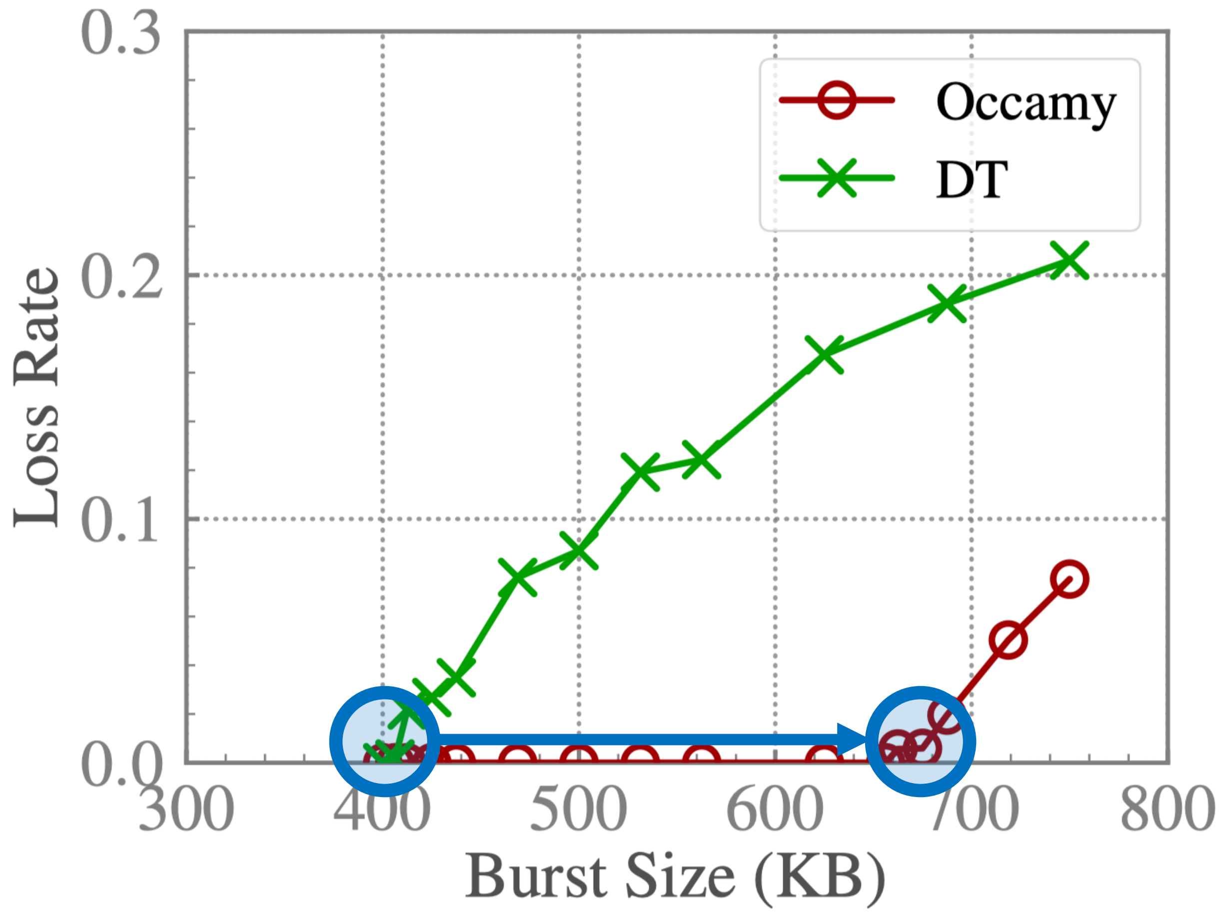
FPGA cost by Vivado

- <1300 LUTs and 60 Flip Flops

ASIC cost by Design Compiler

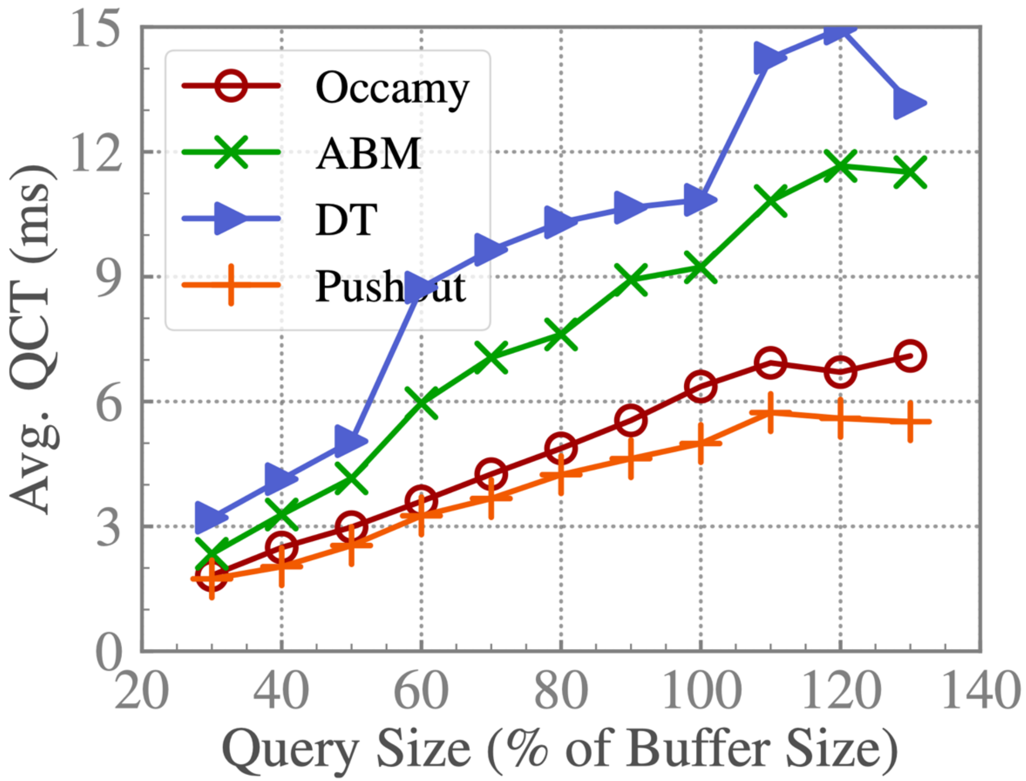
- 1.5ns timing
- 0.03mm^2 area cost and 1mW power

Evaluations --- P4-based HW Prototype

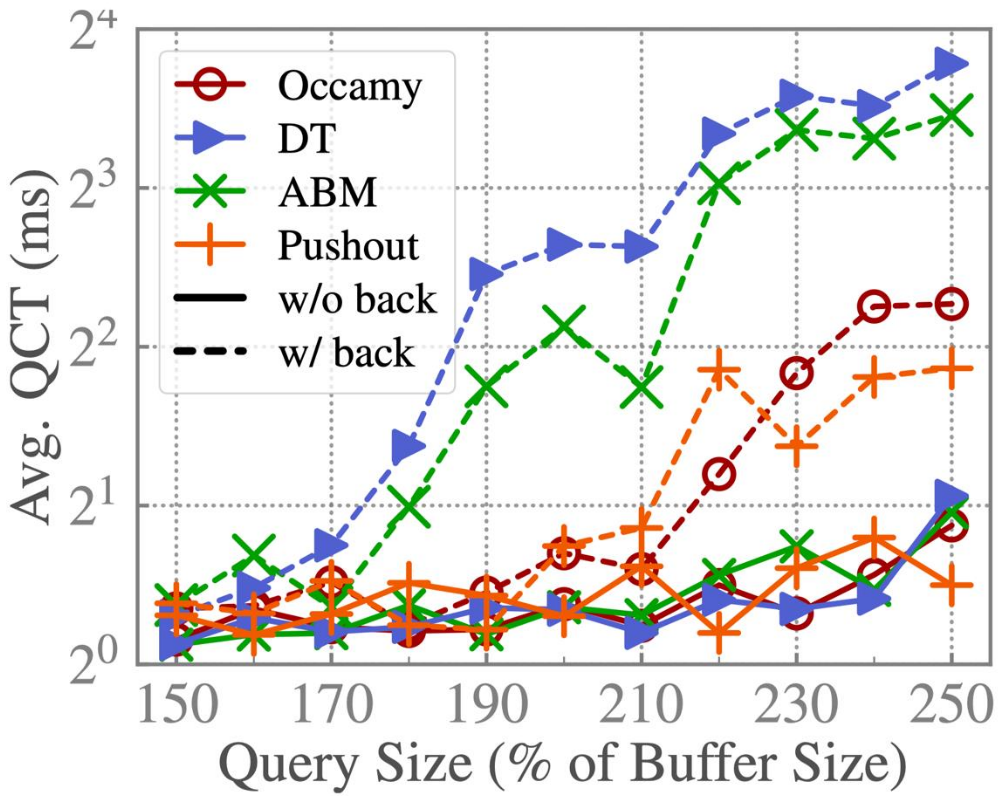


Occamy can absorb 57% more bursty traffic than DT (α=4)

Evaluations --- DPDK-based HW Prototype

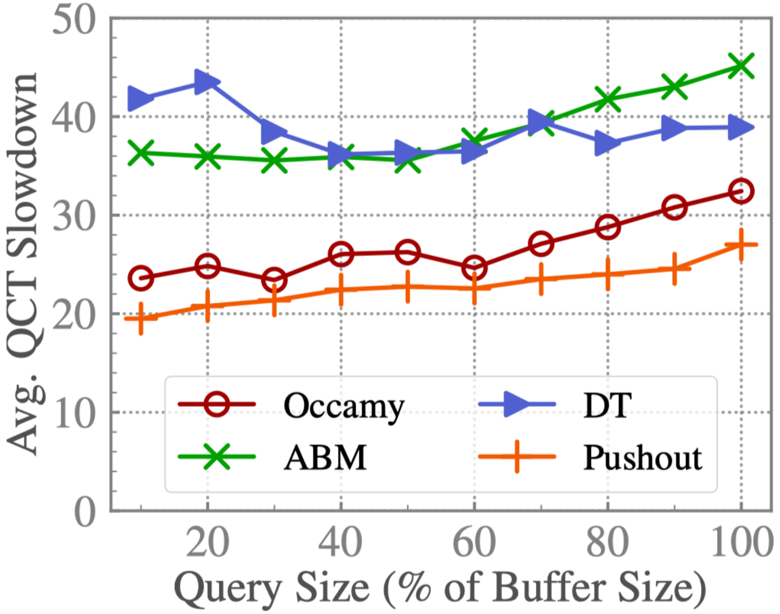


Occamy can reduce the average query completion time by up to ~55%

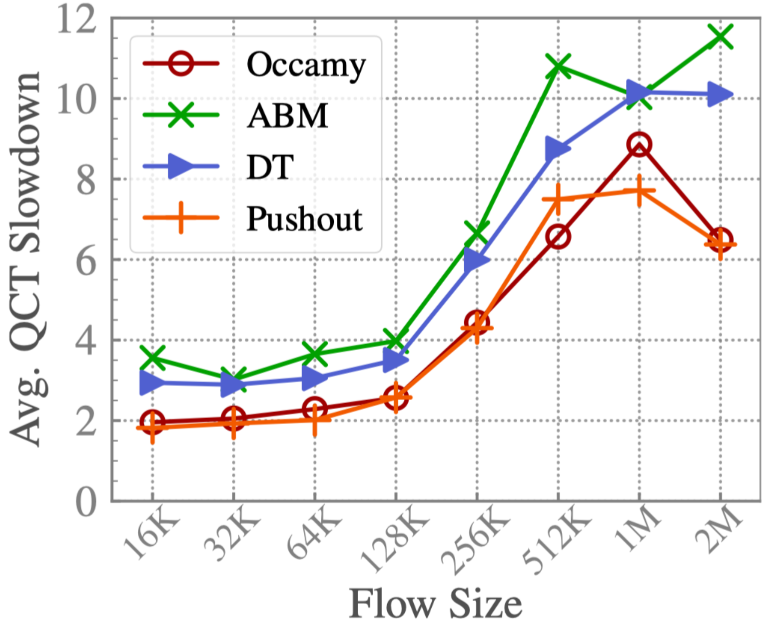


Occamy achieves similar performance to Pushout when facing buffer choking

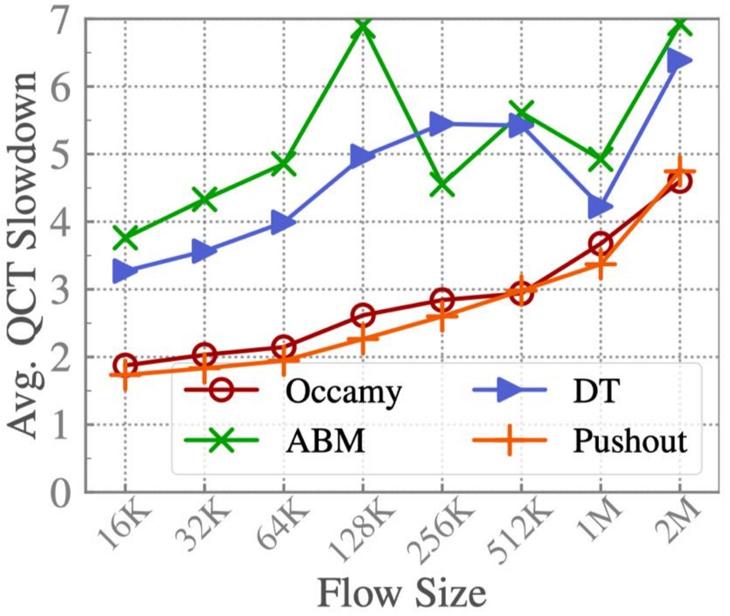
Evaluations --- ns-3 simulations



Traditional DCN Traffic



All-to-all Traffic



All-reduce Traffic

Occamy significantly improves the query completion time with various traffic patterns

Conclusion

- This paper answers 3 questions:
  - What are the fundamental requirements of BMs with insufficient buffer and intense traffic bursts?
  - Answer: BM should be *highly agile*
  - What are the intrinsic limitations of current BMs in meeting the requirements in DCN?
  - Answer: It is the *non-preemptive nature* that confines the agility of current BM
  - Is it possible to break through these limitations with the recent advances on buffer architecture?
  - Answer: Yes. We design *Occamy*, a simple yet effective preemptive BM

Thank you!

https://github.com/ants-xjtu/Occamy