# LEFT: LightwEight and FasT packet Reordering for RDMA

Peihao Huang, Xin Zhang, **Zhigang Chen***, Can Liu, **Guo Chen***

**204701003@csu.edu.cn**

**August 3, 2024**

# RDMA is hot topic in HPC and DCN

- Lower latency, Higher bandwidth, Lower CPU utilization

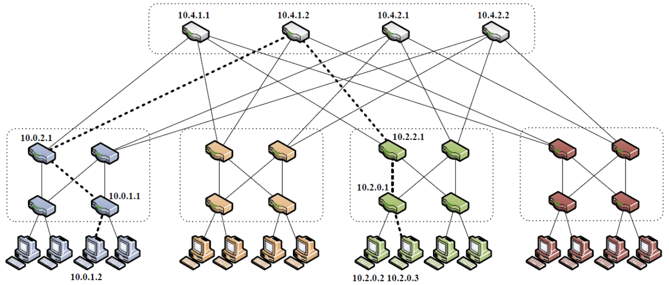
DSLR’18
DrTM+H’18
NAM-DB’17
FaRM’14
HERD’14
CoRM’21
Wukong’16
RAMCloud’15
Octopus’17
RDMP-KV’20
Catfish’19
FileMR’14
FaSST’16
XSTORE’20
ccNUMA’18

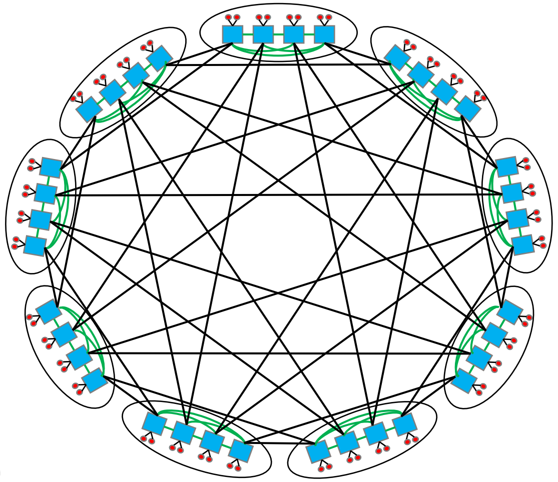
**RDMA**

HydraDB’15
Derecho’19
Hermes’20
Grappa’15
A1’20
DrTM+R’16
Crail’19
SparkRDMA’14
C-Hint’14
ScaleRPC’19
Dare’15
Storm’19
DaRPC’14
DrTM’15
TH-DPMS’16
Hyperloop’18
APUS’17

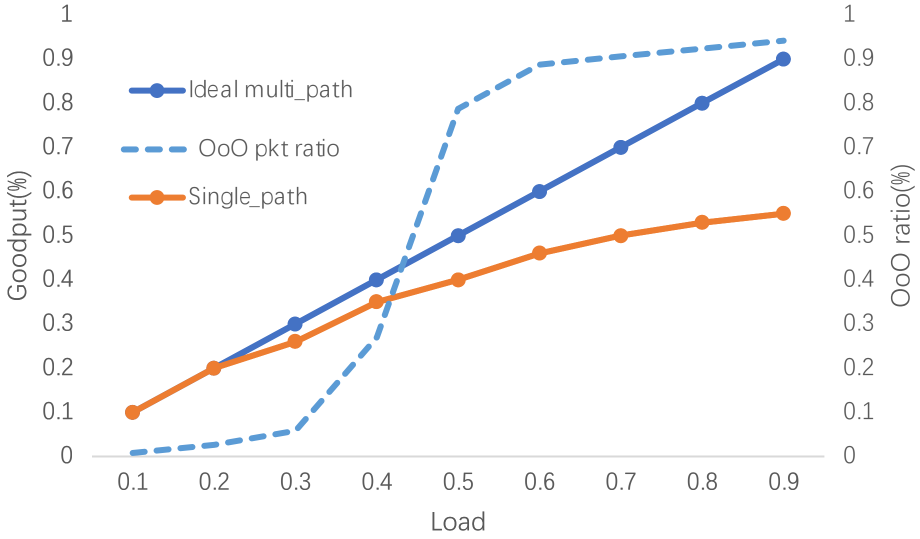
# RDMA needs to deal with OoO packets

- Abundant parallel paths in DCN



- Single-path transmission cannot fully utilize network bandwidth
  - Uneven load
  - Hash conflict
- Multipath brings massive OoO packets
  - Path delays, switch queue lengths vary
  - OoO packets ratio related to the load

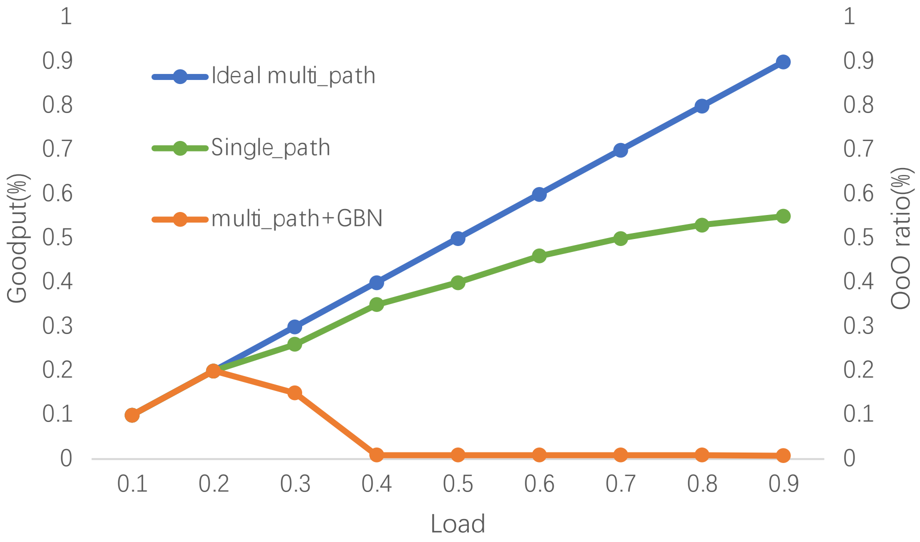


# RDMA needs to deal with OoO packets

- Current RDMA behaves very poor in deal with OoO

  **Go-Back-N (GBN) loss recovery has very low efficiency**

  - Mis-treat OoO as loss, abundant unnecessary retransmission



# RDMA needs to deal with OoO packets

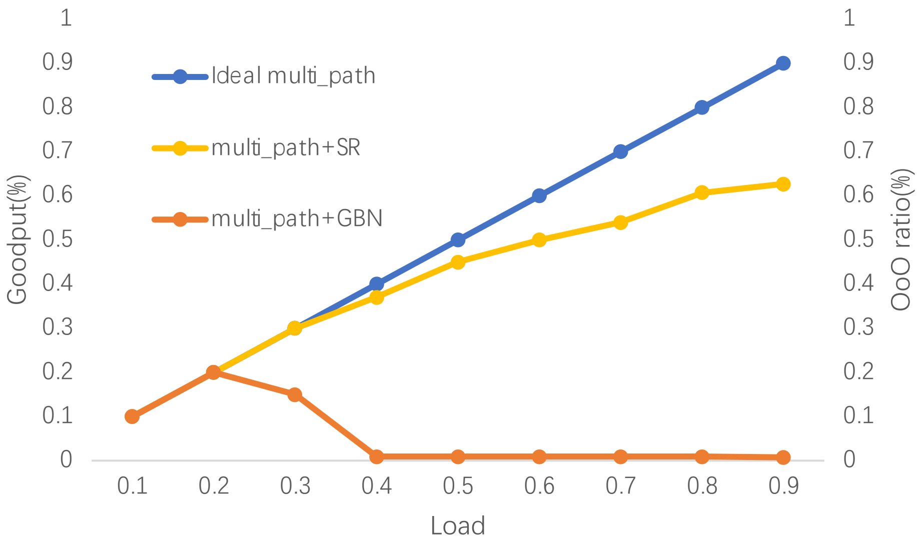
- Current RDMA behaves very poor in deal with OoO

**Go-Back-N (GBN) loss recovery has very low efficiency**

- Mis-treat OoO as loss, abundant unnecessary retransmission

**Selective retransmission (SR) is still inefficient to deal with OoO**

- Still Mis-treat OoO as loss
  - Sending rate decreased due to CC miss-acting to OoO, and unnecessary retransmission
- Bitmap record OoO packets, which consumes too much RNIC memory
  - 5BDP bitmap required per connection
  - BDP: 200G * 32us (1KB MTU) ~ 100B
  - 2.5MB required for  5k connections



# Related work

- To reduce OoO packets, limit load-balancing granularity and flexibility thus less pressure on RNIC reordering
  - Letflow[1], Conweave[2], MPRDMA[3]

**Inferior network utilization due to sub-optimal load-balancing**

- Improve RNIC SR efficiency
  - IRN[4] , SRNIC[5] , MELO[6]

**Mismatch to the OoO scenario, and burden on RNIC**
- **SR consumes much memory**
- **SR is slow in processing with massive OoO packets**

**Fast and memory-efficient reorder scheme on RNIC to support fine-grained per-packet multi-path load balancing!**


[1] Let It Flow: Resilient Asymmetric Load Balancing with Flowlet Switching, NSDI'2017
[2] Network Load Balancing with In-network Reordering Support for RDMA, SIGCOMM'2023
[3] Multi-Path Transport for RDMA in Datacenters, NSDI'2018
[4]Revisiting Network Support for RDMA. SIGCOMM'2018
[5]SRNIC: A Scalable Architecture for RDMA NICs. USENIX2023
[6] Memory Efficient Loss Recovery for Hardware-based Transport in Datacenter. APNET'2017

# Challenges

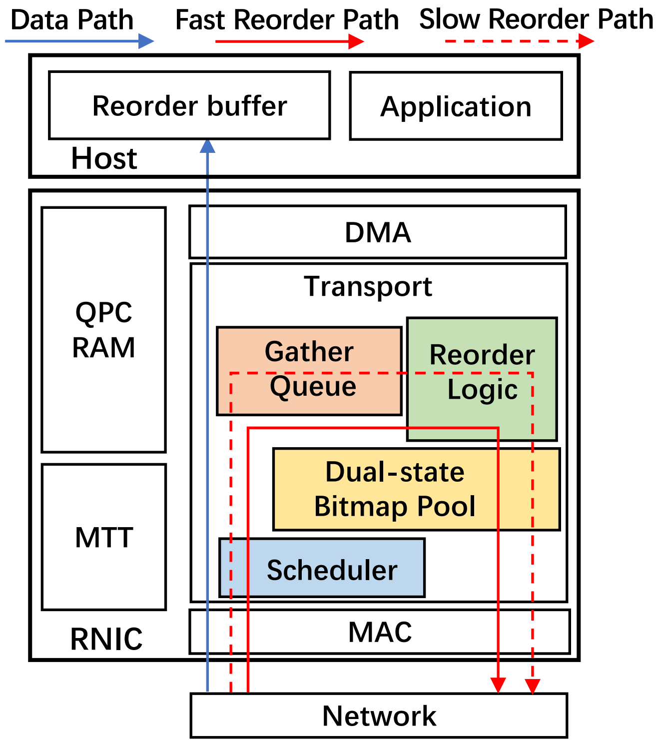
- Key point
  - **Recording OoO, bitmaps must be lightweight and fast**
  - CC should differentiate normal OoO from packet loss

- Challenge 1： Massive Bitmaps in Small On-chip Memory
  - Bitmap Pooling for massive storage efficiency
  - Bitmap Caching for frequent  random access efficiency

- Challenge 2： High Disorder Processing within Small Time Constraints
  - Packet gather to reduce average bitmap latency
  - Dynamic scheduling with fast and slow paths

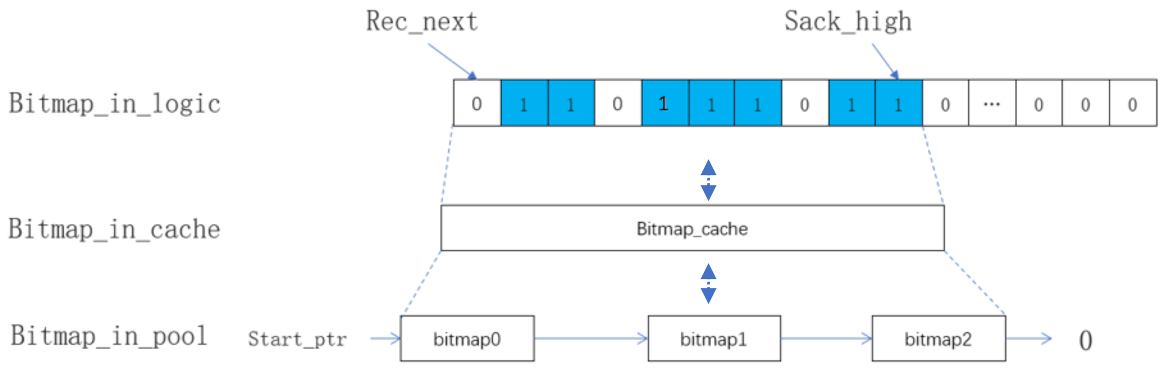
# Overview of LEFT

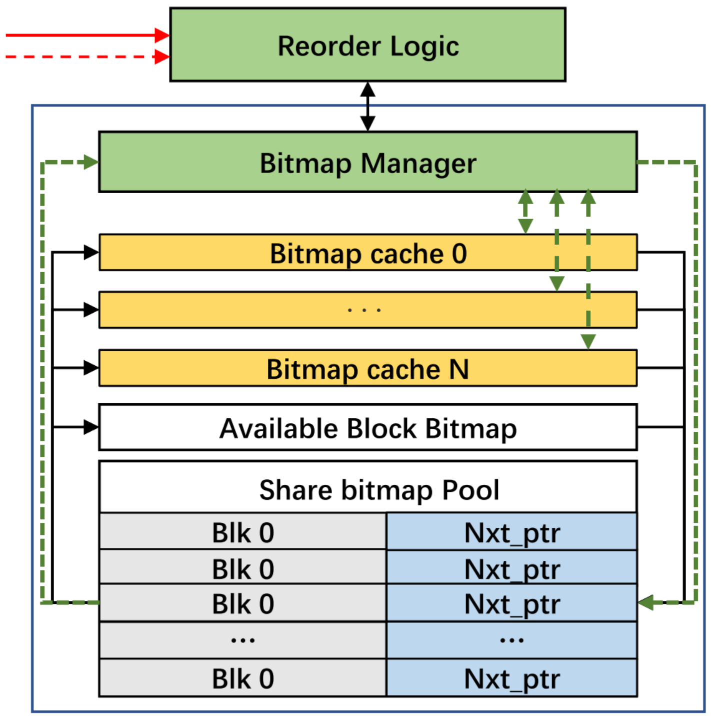
- **Data Path**
  - Reorder buffer is allocated to each connection for OoO data
  - Submit to the application after OoO recovery
- **Control Path**
  - Reorder: OoO packets no longer trigger retransmission and CC
  - Dual-state bitmap pool: Small, fast bitmap shared by all connections
  - Gather queue: Aggregating packets from different connections
  - Scheduler: Selects the path for incoming packets based on the degree of packet reordering



# Dual-state bitmap pool

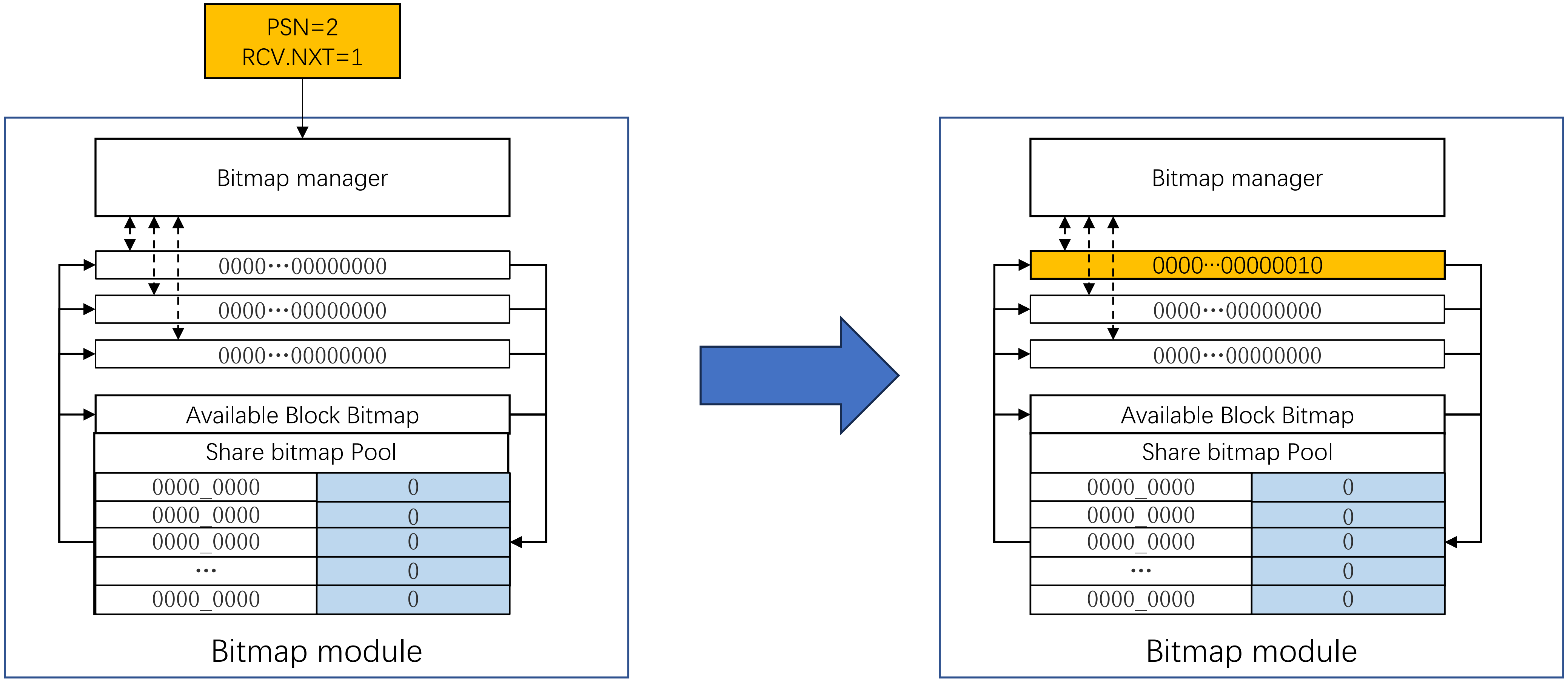
- **Structure**
  - Available block bitmap marks the available bitmap blocks in pool
  - Bitmap block as the smallest bitmap sharing unit
  - Bitmap blocks of the same connection are linked by Nxt_ptr
  - Bitmap cache consists of loop registers
- **Connection bitmap will be stored in the cache or pool**
  - Simple bitmap in cache, fast but memory inefficient
  - Link list bitmap in pool, slow but memory efficient [6]



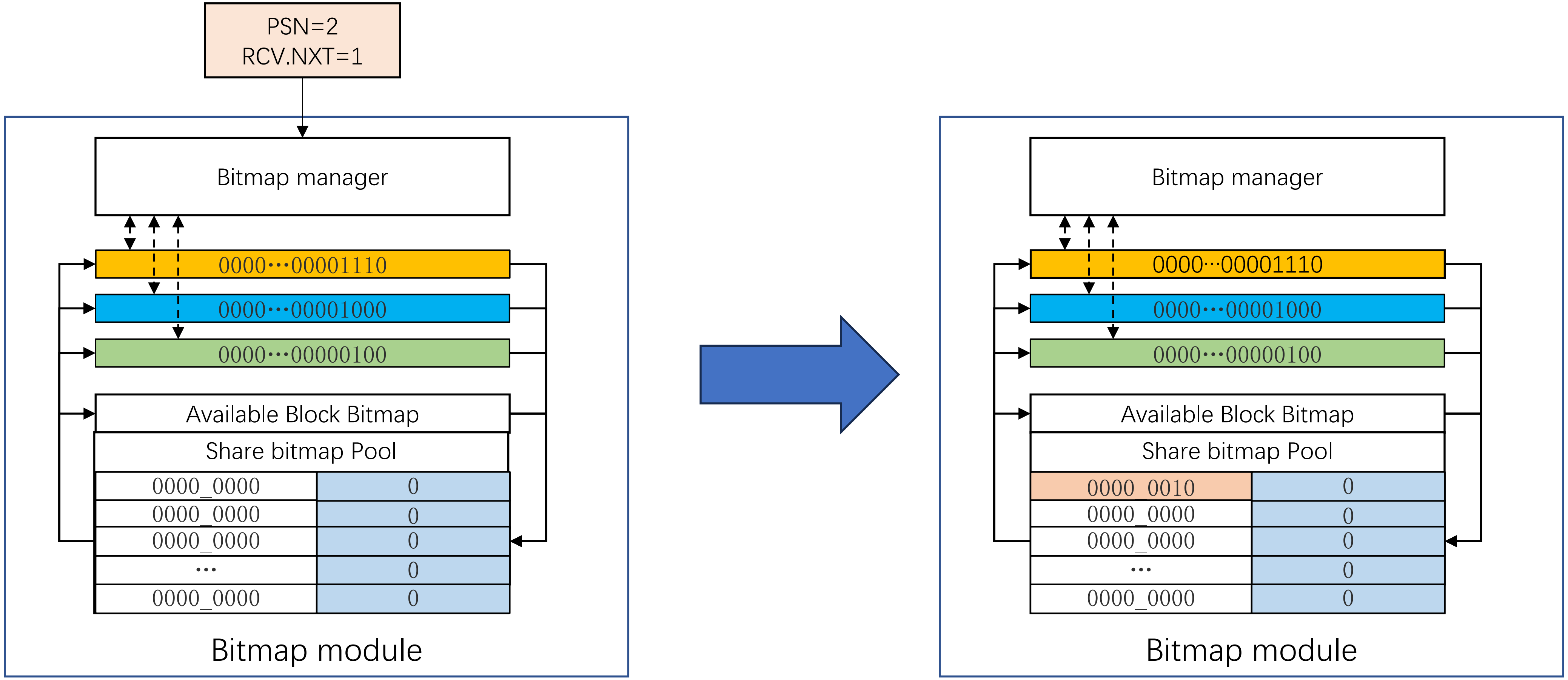
# Dual-state bitmap pool

- Cache hits, bitmap updated in cache



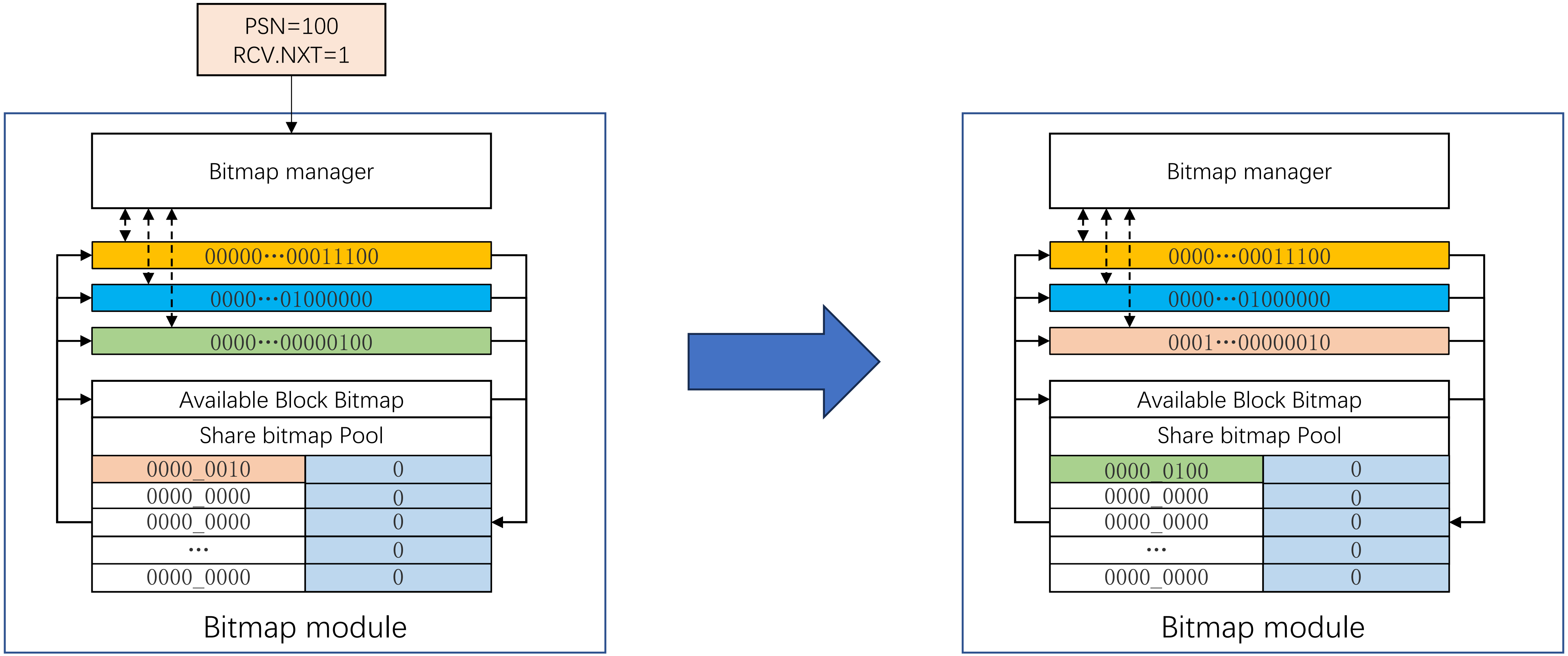
# Dual-state bitmap pool

- Cache miss, bitmap updated in bitmap pool



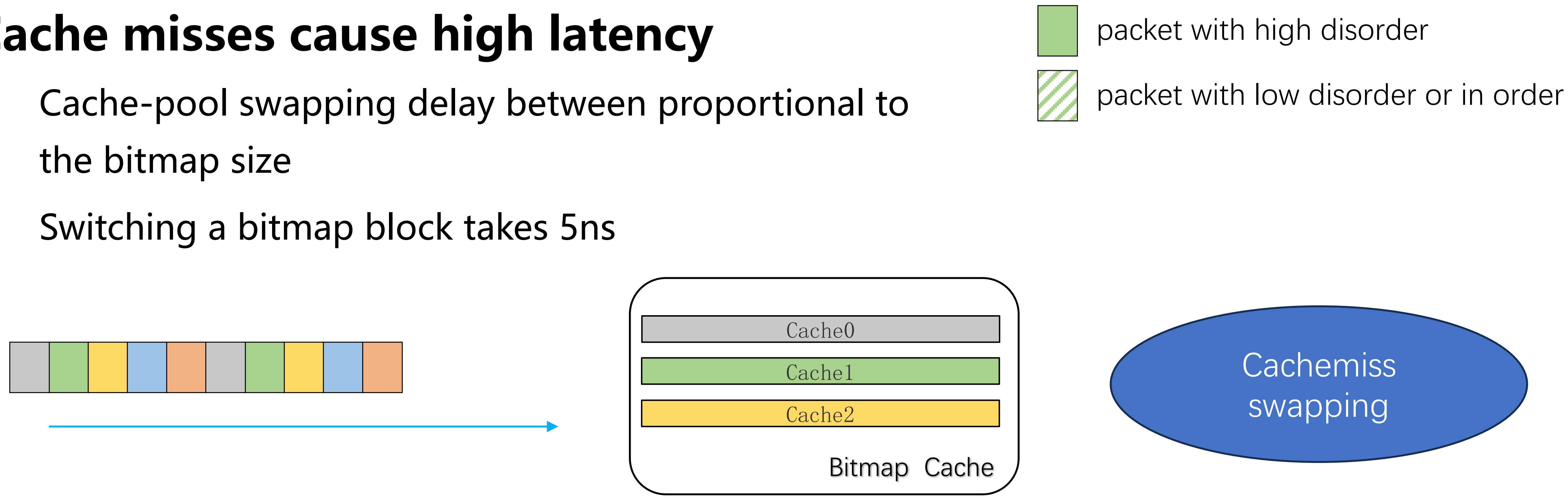
# Dual-state bitmap pool

- Cache miss, bitmap updated in bitmap cache



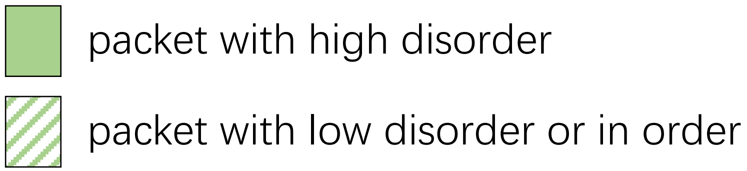
# Bitmap Acceleration

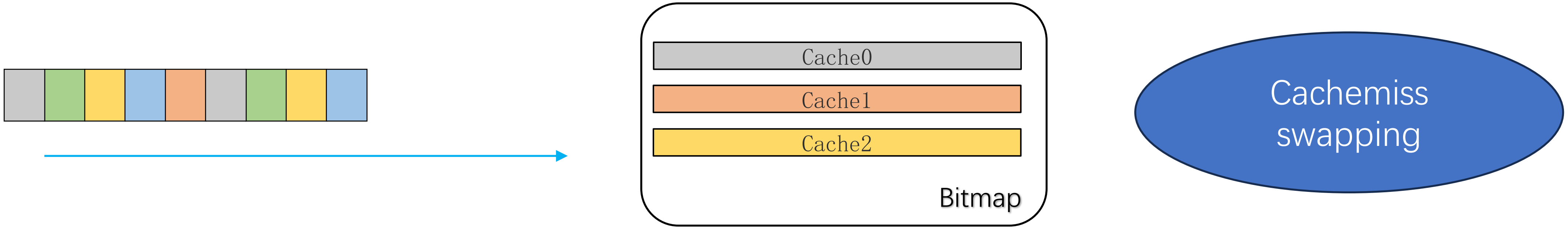
- **Cache misses cause high latency**
  - Cache-pool swapping delay between proportional to the bitmap size
  - Switching a bitmap block takes 5ns



# Bitmap Acceleration

- **Cache misses cause high latency**
  - Cache-pool swapping delay between proportional to the bitmap size
  - Switching a bitmap block takes 5ns



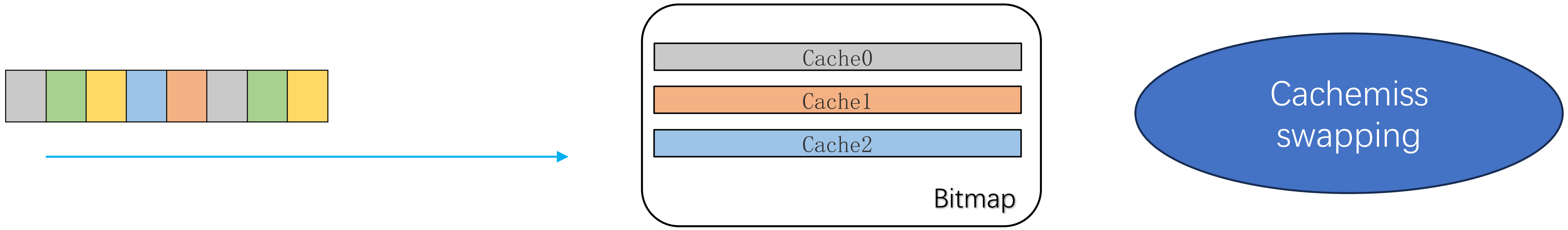



# Bitmap Acceleration

packet with high disorder

packet with low disorder or in order

- **Cache misses cause high latency**
  - Cache-pool swapping delay between proportional to the bitmap size
  - Switching a bitmap block takes 5ns

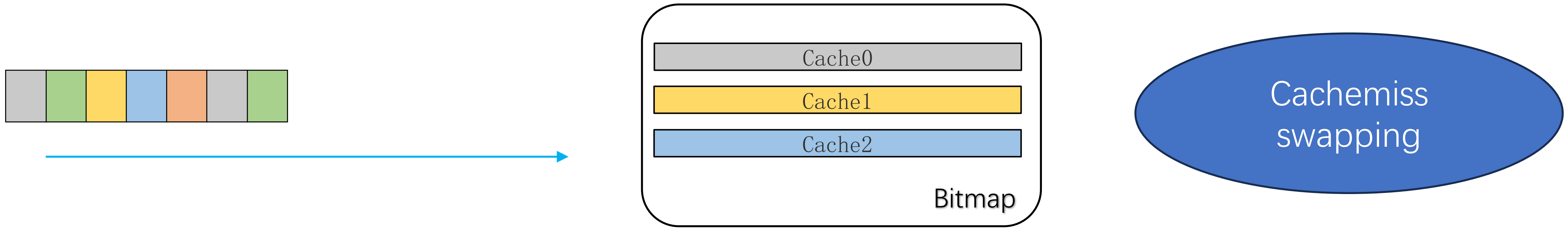


# Bitmap Acceleration

- **Cache misses cause high latency**
  - Cache-pool swapping delay between proportional to the bitmap size
  - Switching a bitmap block takes 5ns

packet with high disorder

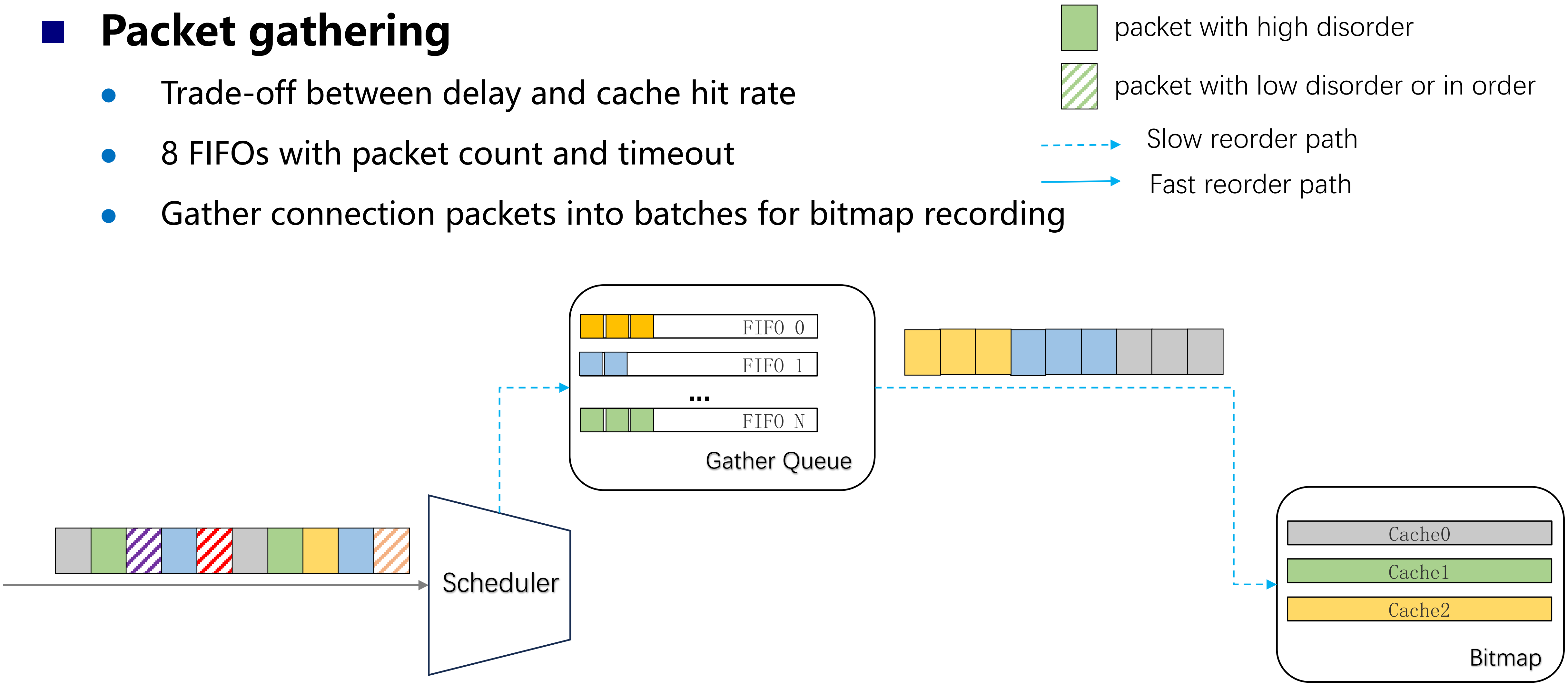
packet with low disorder or in order




**Frequent cache misses cause high bitmap latency and slow RNIC speeds!**

# Bitmap Acceleration

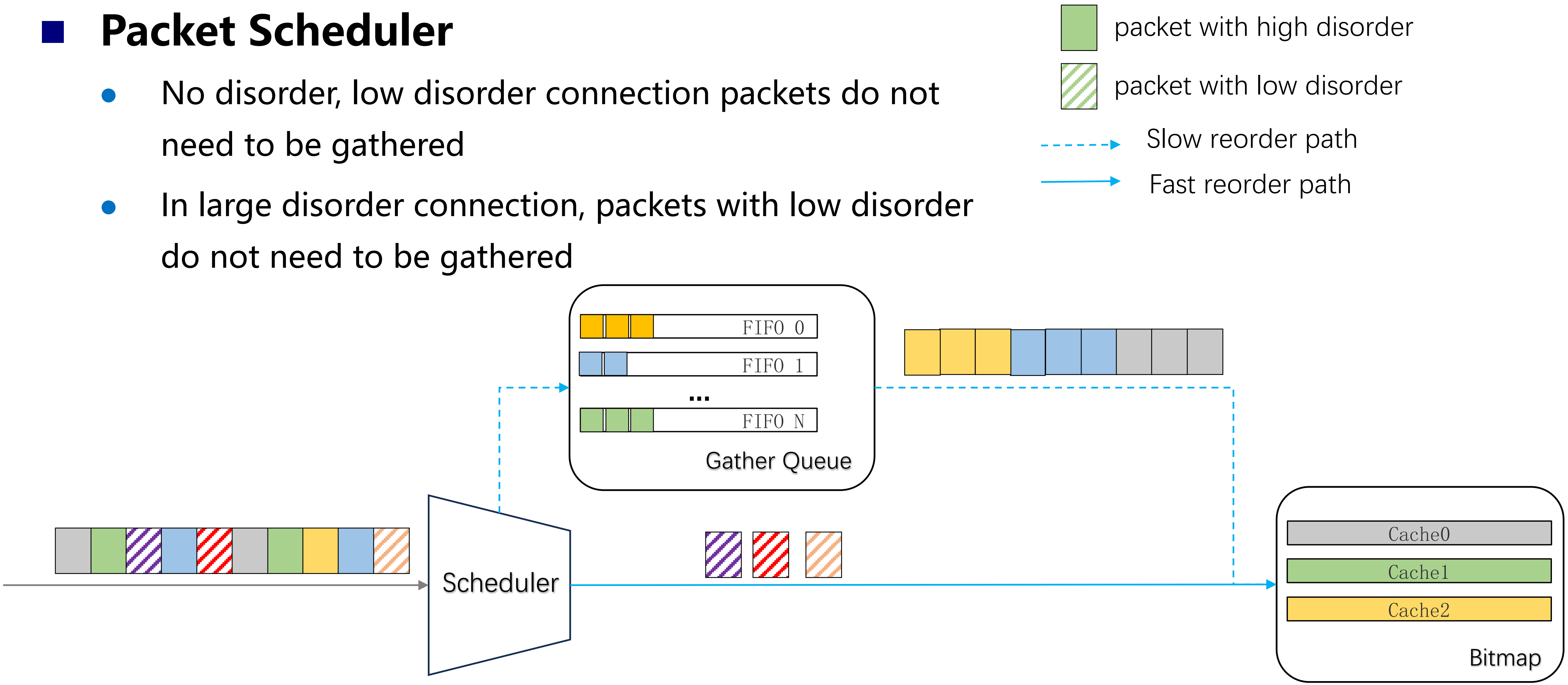
- **Packet gathering**
  - Trade-off between delay and cache hit rate
  - 8 FIFOs with packet count and timeout
  - Gather connection packets into batches for bitmap recording



# Bitmap Acceleration

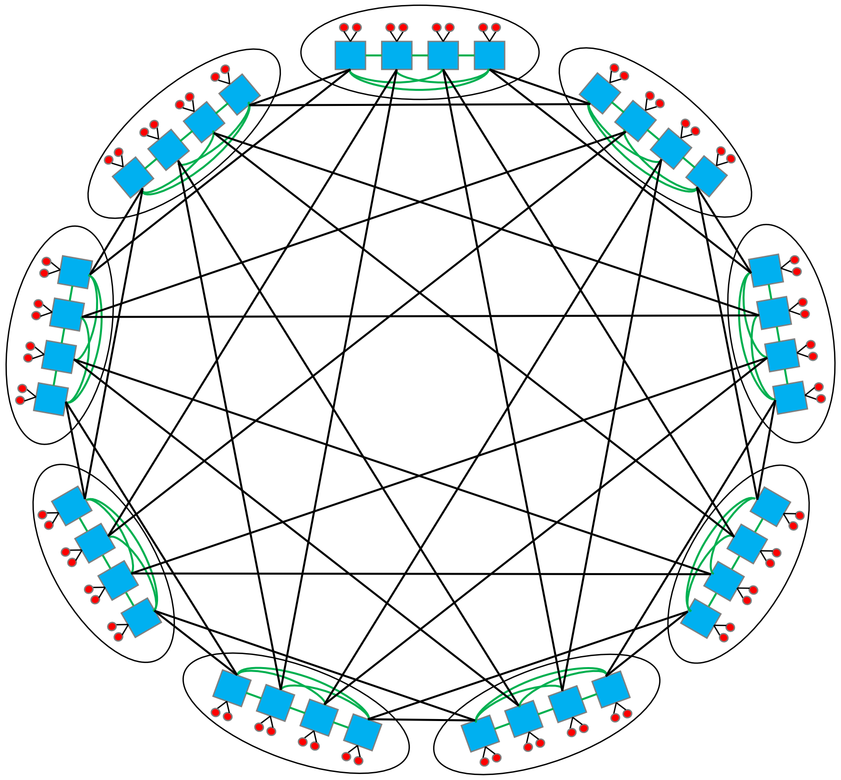
## Packet Scheduler

- No disorder, low disorder connection packets do not need to be gathered
- In large disorder connection, packets with low disorder do not need to be gathered



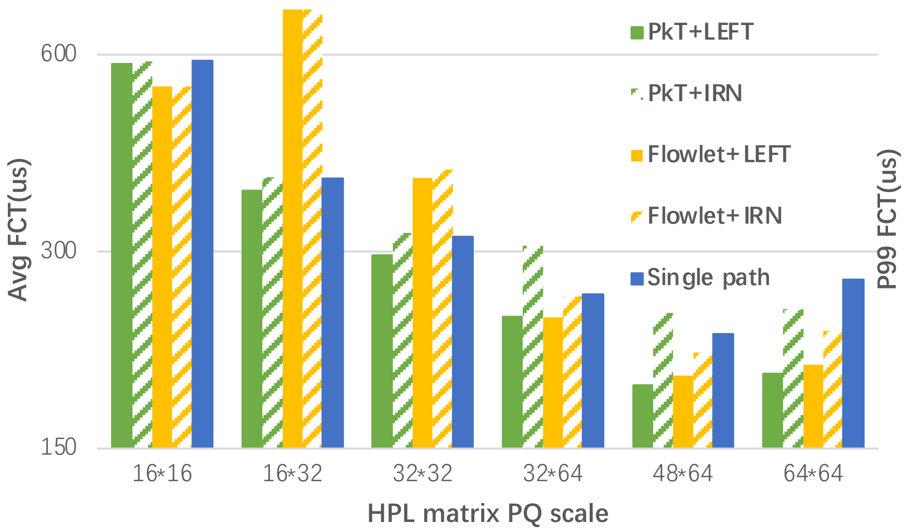
# NS3 Evaluation

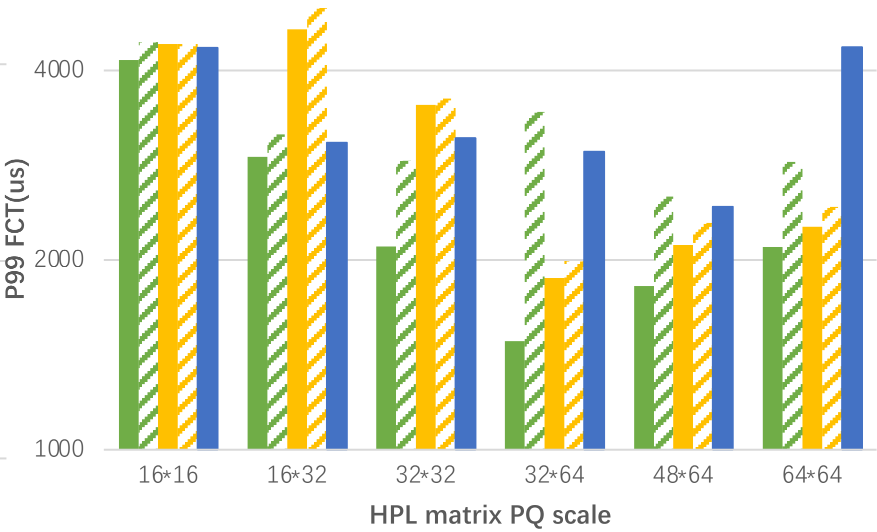
- Setup
  - NS3: integrate LEFT into HPCC
- Topology
  - Dragonfly, 400Gbps direct between groups
  - Bandwidth: 200Gbps，link latency: 1us
  - MTU: 1KB
  - Bits Pool Size: 2Kb
  - Bitmap caches: 3, aggregate queues: 8
- Comparison solutions
  - Perflow, IRN, LEFT-(without scedule), MELO+(with cache)

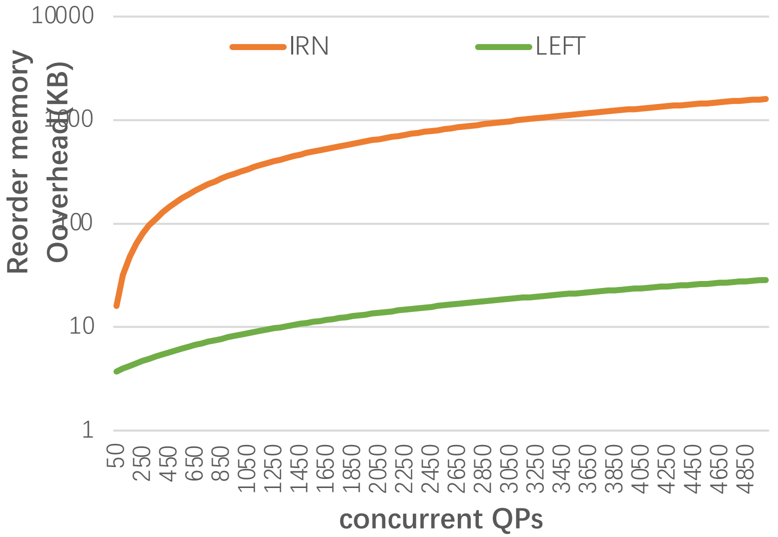


# Evaluation-Benefit FCT

- Average FCT:
  - LEFT improve 28.1% compared with per-flow, 20% and 12% reduction compared with per-packet and per-flowlet without LEFT equipped specifically
- 99% tail FCT:
  - 48.2% reduction compared with per-flow, 26.7% and 8.1% reduction compared with per-packet and per-flowlet without LEFT equipped specifically
- LEFT adds 27KB memory overhead, only 1.7% of the baseline (IRN)

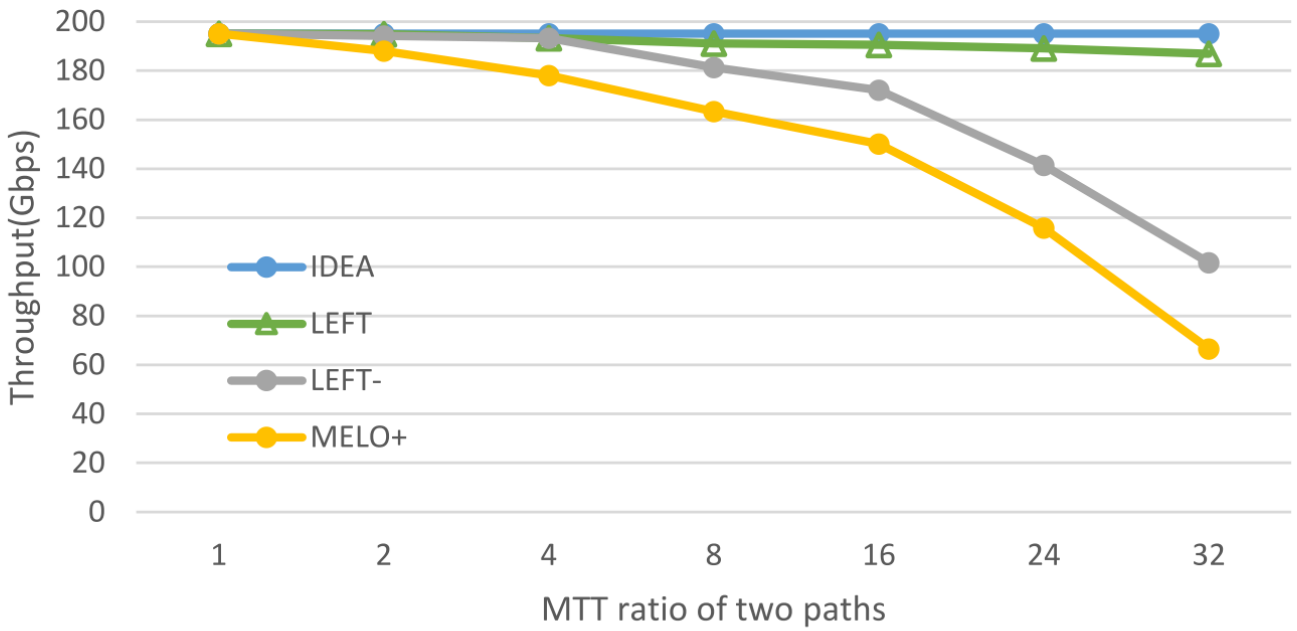


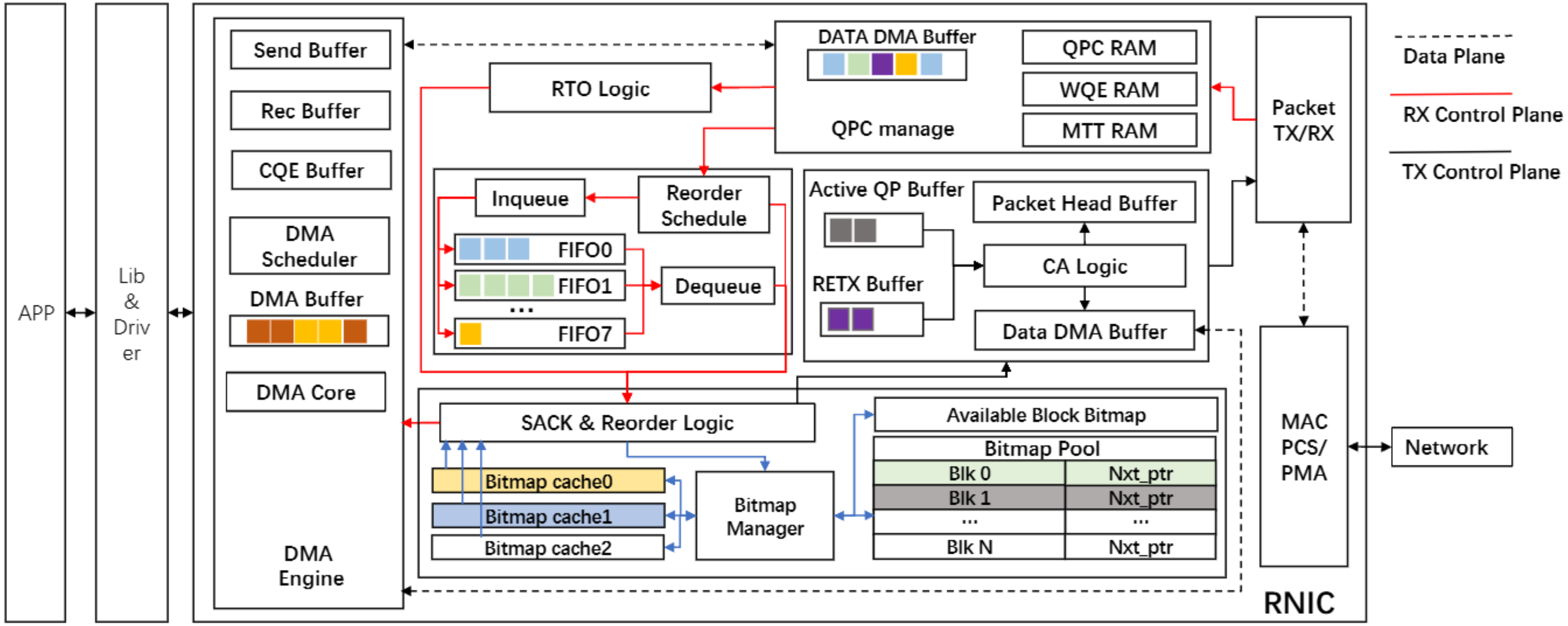
# Evaluation-Benefit Throughput

- Setup:
  - Two servers connected by two links, and one of the link have different delay(1us,2us...32us)
  - evenly distribute all packets on two path
- 94% throughput under path RTT differs by 32 times. 83% and 180% higher than LEFT- and MELO+, respectively.



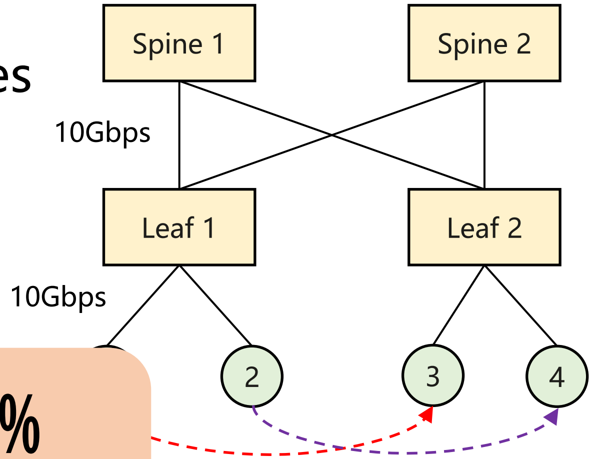
# LEFT FPGA implementation

- LEFT is implemented in XCKU040-FFVA1156-2I Xilinx FPGA.
- The board has four 10Gbps ports and one 25Gbps port.
- The RTL code utilizes VIVADO 2022.1 for logic simulation, synthesis, layout, and routing



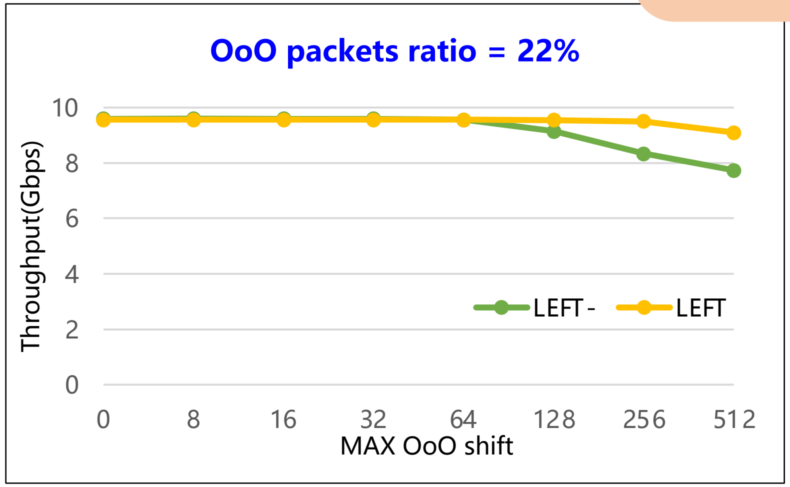
# FPGA Result

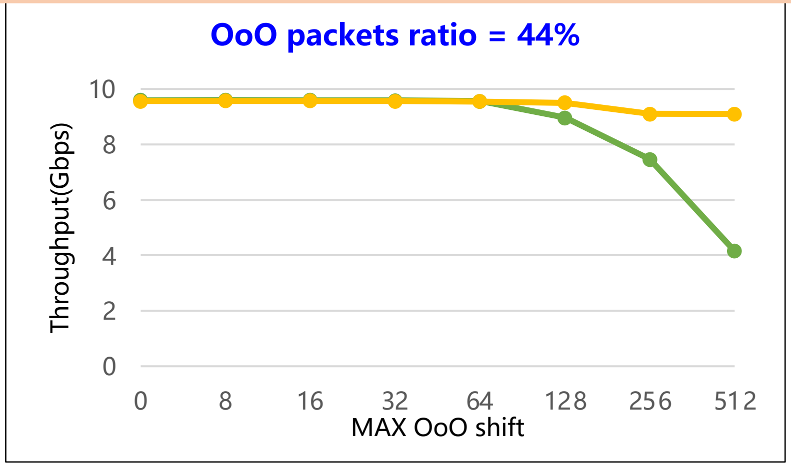
- Setup
  - ① sends ③, ② sends ④, 5 connections between each pair of nodes
  - Bandwidth: 10Gbps，link latency: 1us
  - MTU: 75B
  - Bits Pool Size: 2Kb, Bitmap caches: 2, aggregate queues: 8

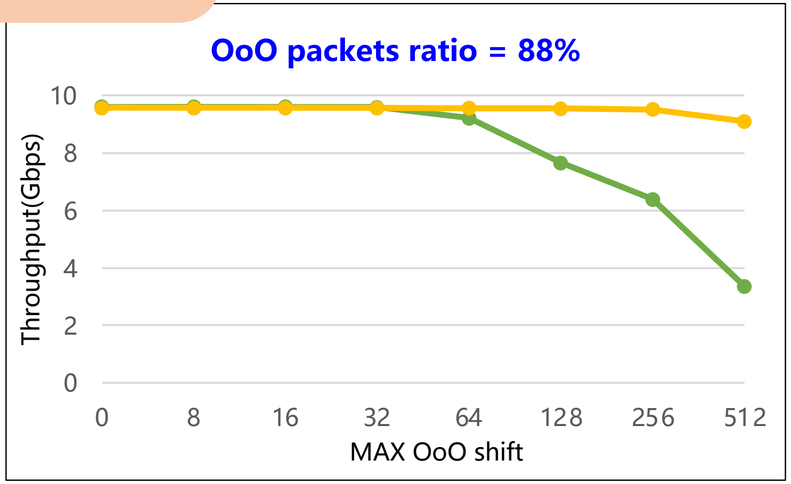



- Comparison solu
  - LEFT, LEFT-

**92% throughput under 88% OoO packets ratio**



# Conclusion & future work

- **LEFT supporting flexible multi-path transmission without affecting RNIC performance!**
  - By dual-state bitmap pool and In addition, LEFT achieves lightweight memory overhead, with an average of only 7B additional storage overhead per connection
  - By fast and slow reordering path, LEFT achieves exceptional speed, maintaining 94% throughput even when the path RTT differs by 32 times

- **There are some problems worth future study.**
  - Modeling and analysis of the number and size of bitmap blocks, caches, and gather queues
  - Large-scale testbed implementation
  - More efficient judgment of packet loss and reduced overall bitmap overhead

# Thanks for listening

# Q & A

Contact：204701003@csu.edu.cn