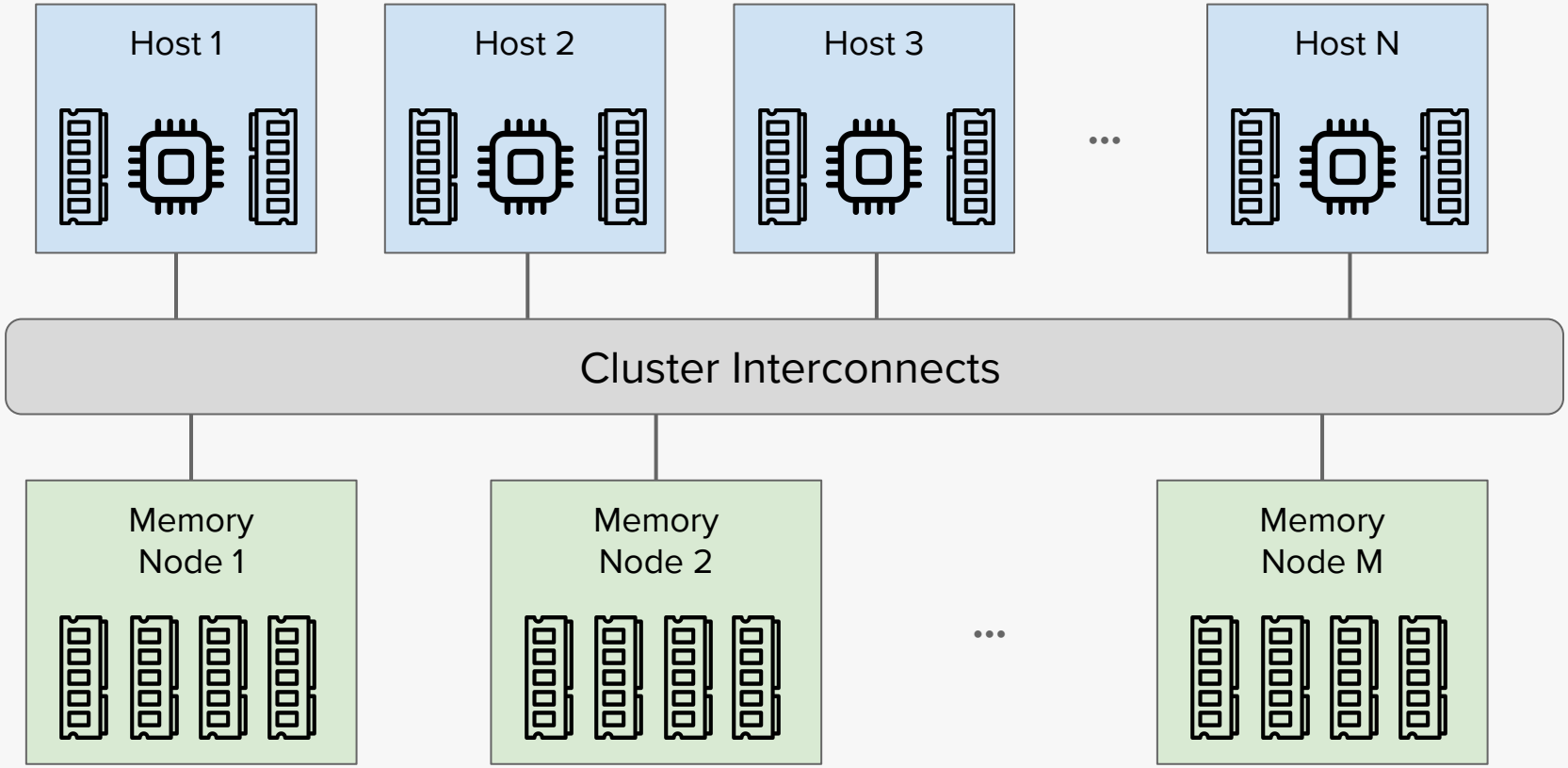
Building A CSFQ-Inspired Transport for Switched CXL Memory Pooling

Zerui Guo, Emily Shriver, Ming Liu

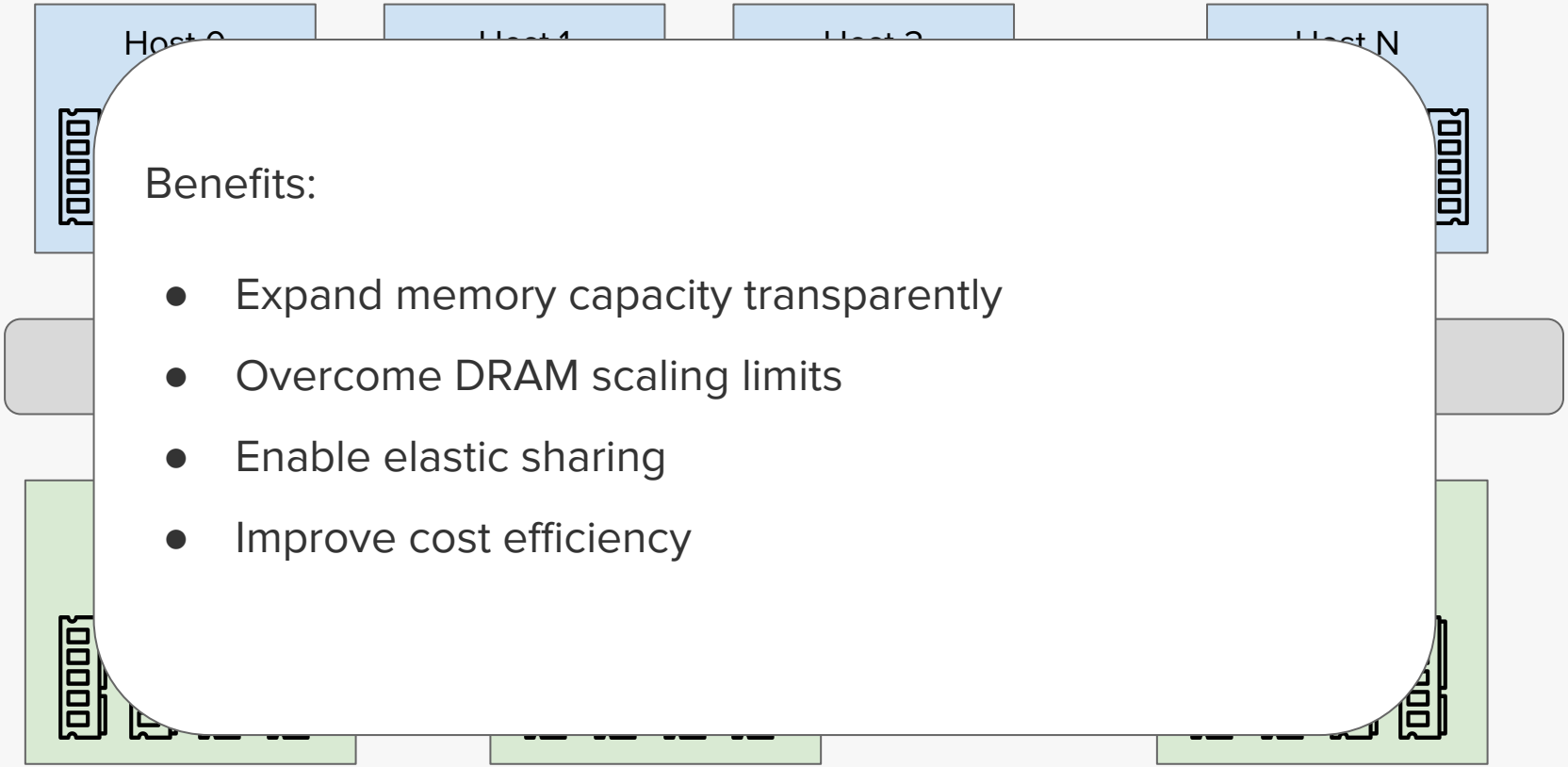
NetLab@UW-Madison

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Memory Pooling



Memory Pooling

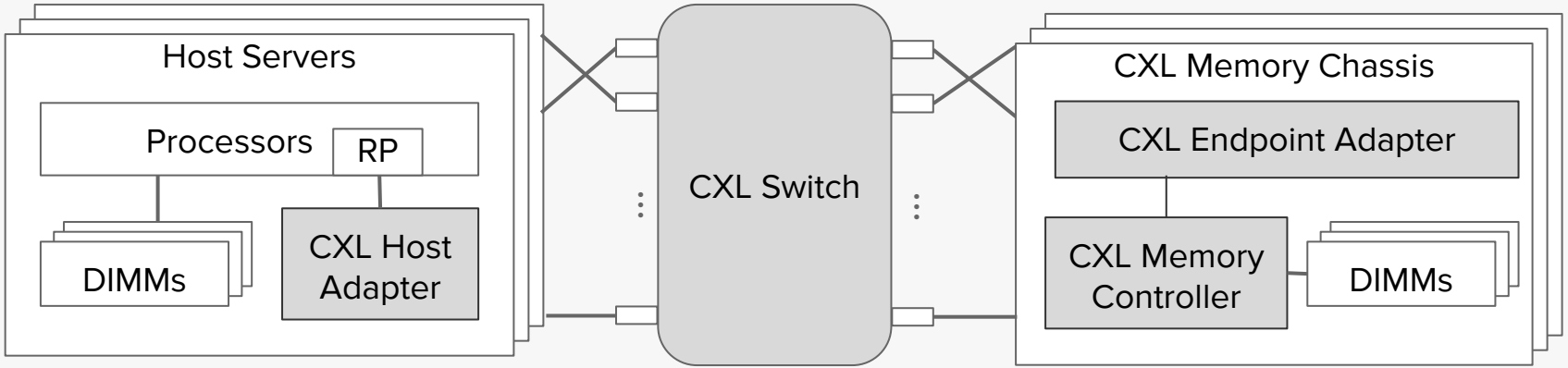


Benefits:

- Expand memory capacity transparently
- Overcome DRAM scaling limits
- Enable elastic sharing
- Improve cost efficiency

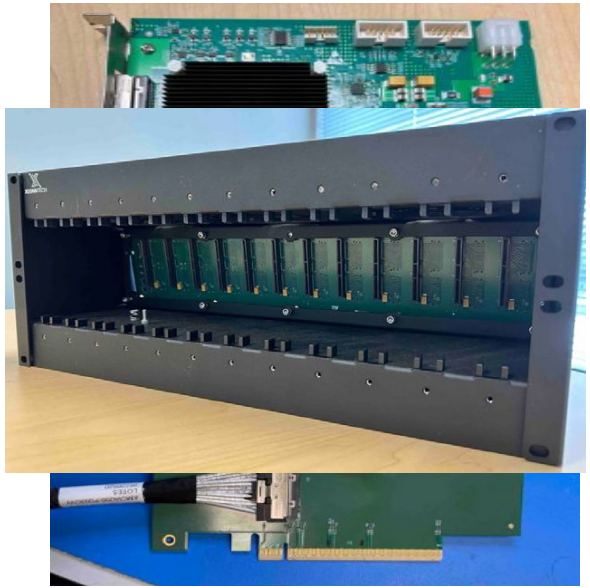
Compute Express Link (CXL)

- An emerging cluster interconnect offering load/store access model
- Enable rack-scale composable memory pooling



Compute Express Link (CXL)

- XConn Titan Evaluation Platform
  - An XConn B2 Apollo CXL switch
    - 14 upstream and downstream ports
  - ASIC-based host and endpoint adapters
    - PCIE Gen 5 x16
  - A memory chassis
    - MCIO backplane
    - Up to 12 CXL DIMMs



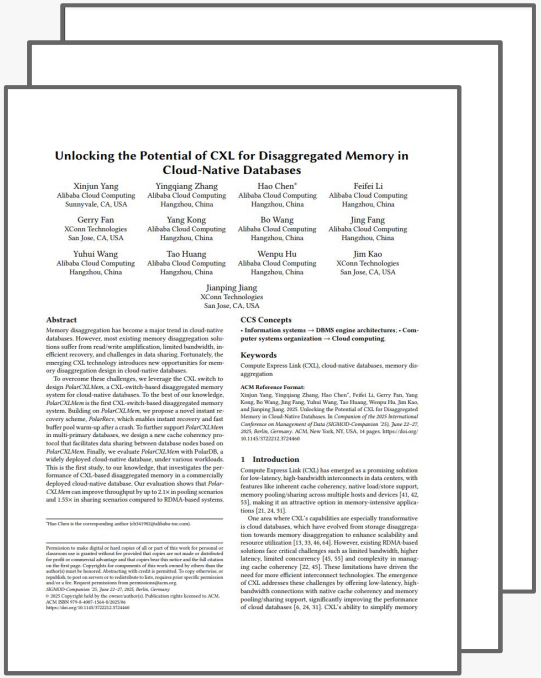
CXL-based Memory Pooling Being Deployed

Cloud Service Providers:

Meta

Azure

Alibaba Cloud



Unlocking the Potential of CXL for Disaggregated Memory in Cloud-Native Databases

Xinjun Yang, Alibaba Cloud Computing, Sunnyvale, CA, USA
Yingqiang Zhang, Alibaba Cloud Computing, Hangzhou, China
Hao Chen*, Alibaba Cloud Computing, Hangzhou, China
Feifei Li, Alibaba Cloud Computing, Hangzhou, China
Gerry Fan, XConn Technologies, San Jose, CA, USA
Yang Kong, Alibaba Cloud Computing, Hangzhou, China
Bo Wang, Alibaba Cloud Computing, Hangzhou, China
Jing Fang, Alibaba Cloud Computing, Hangzhou, China
Yuhui Wang, Alibaba Cloud Computing, Hangzhou, China
Tao Huang, Alibaba Cloud Computing, Hangzhou, China
Wenpu Hu, Alibaba Cloud Computing, Hangzhou, China
Jim Kao, XConn Technologies, San Jose, CA, USA
Jianping Jiang, XConn Technologies, San Jose, CA, USA

Abstract

Memory disaggregation has become a major trend in cloud-native databases. However, most existing memory disaggregation solutions suffer from read/write amplification, limited bandwidth, inefficient recovery, and challenges in data sharing. Fortunately, the emerging CXL technology introduces new opportunities for memory disaggregation design in cloud-native databases.

To overcome these challenges, we leverage the CXL switch to design *PolarCXLMem*, a CXL-switch-based disaggregated memory system for cloud-native databases. To the best of our knowledge, *PolarCXLMem* is the first CXL-switch-based disaggregated memory system. Building on *PolarCXLMem*, we propose a novel instant recovery scheme, *PolarRecv*, which enables instant recovery and fast buffer pool warm-up after a crash. To further support *PolarCXLMem* in multi-primary databases, we design a new cache coherency protocol that facilitates data sharing between database nodes based on *PolarCXLMem*. Finally, we evaluate *PolarCXLMem* with PolarDB, a widely deployed cloud-native database, under various workloads. This is the first study, to our knowledge, that investigates the performance of CXL-based disaggregated memory in a commercially deployed cloud-native database. Our evaluation shows that *PolarCXLMem* can improve throughput by up to 2.1× in pooling scenarios and 1.55× in sharing scenarios compared to RDMA-based systems.

CCS Concepts

• **Information systems** → **DBMS engine architectures**; • **Computer systems organization** → **Cloud computing**.

Keywords

Compute Express Link (CXL), cloud-native databases, memory disaggregation

ACM Reference Format:

Xinjun Yang, Yingqiang Zhang, Hao Chen*, Feifei Li, Gerry Fan, Yang Kong, Bo Wang, Jing Fang, Yuhui Wang, Tao Huang, Wenpu Hu, Jim Kao, and Jianping Jiang. 2025. Unlocking the Potential of CXL for Disaggregated Memory in Cloud-Native Databases. In *Companion of the 2025 International Conference on Management of Data (SIGMOD-Companion '25), June 22–27, 2025, Berlin, Germany.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3722212.3724460

1 Introduction

Compute Express Link (CXL) has emerged as a promising solution for low-latency, high-bandwidth interconnects in data centers, with features like inherent cache coherency, native load/store support, memory pooling/sharing across multiple hosts and devices [41, 42, 53], making it an attractive option in memory-intensive applications [21, 24, 31].

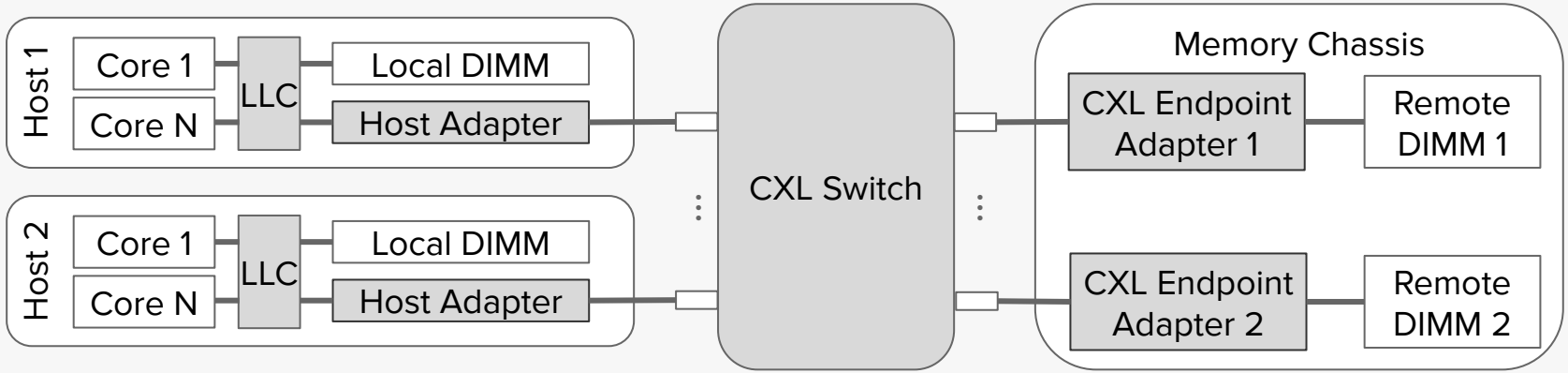
One area where CXL's capabilities are especially transformative is cloud databases, which have evolved from storage disaggregation towards memory disaggregation to enhance scalability and resource utilization [13, 33, 46, 64]. However, existing RDMA-based solutions face critical challenges such as limited bandwidth, higher latency, limited concurrency [45, 55] and complexity in managing cache coherency [22, 45]. These limitations have driven the need for more efficient interconnect technologies. The emergence of CXL addresses these challenges by offering low-latency, high-bandwidth connections with native cache coherency and memory pooling/sharing support, significantly improving the performance of cloud databases [6, 24, 31]. CXL's ability to simplify memory

Outline

- **Characterize Switched CXL Memory Pooling**
- MemChannel Design and Implementation
- MemChannel Evaluation
- Conclusion

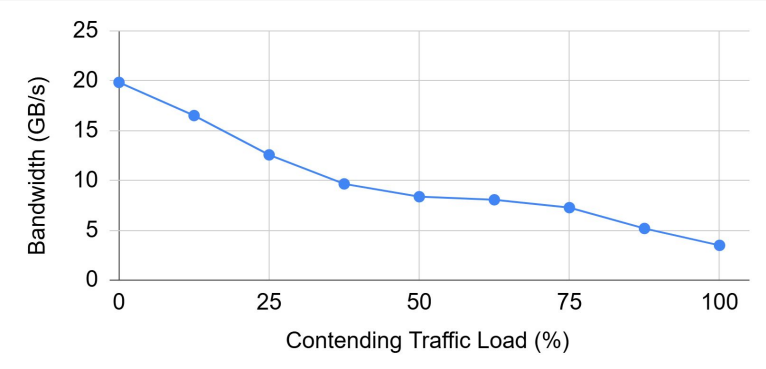
Experimental Methodology

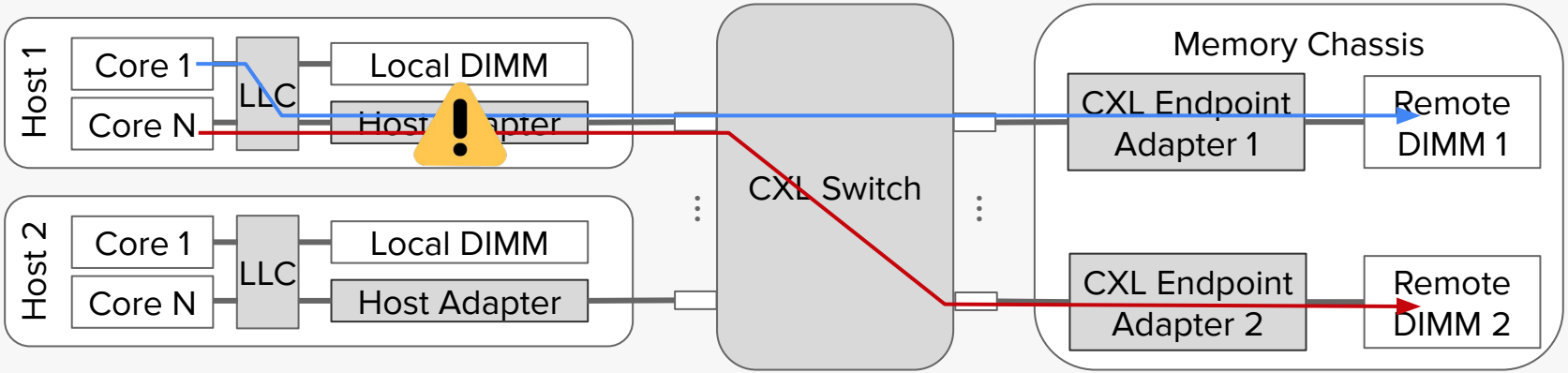
- **Benchmarking framework**
  - Launch memory streams to local/CXL DIMMs
  - Configurable access patterns (e.g., stride width, temporal/non-temporal store, random/sequential)



Issue #1: Intra-host Contention

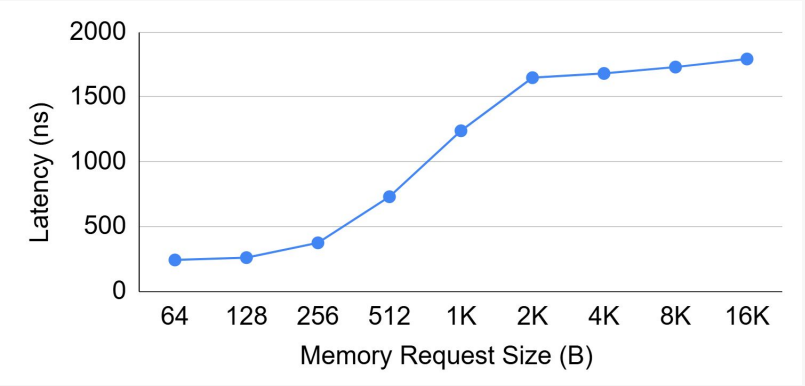
- **Host adapter congestion**
  - Bandwidth dropped by 82.6%
  - CXL FLITs scheduled in a best-effort fashion

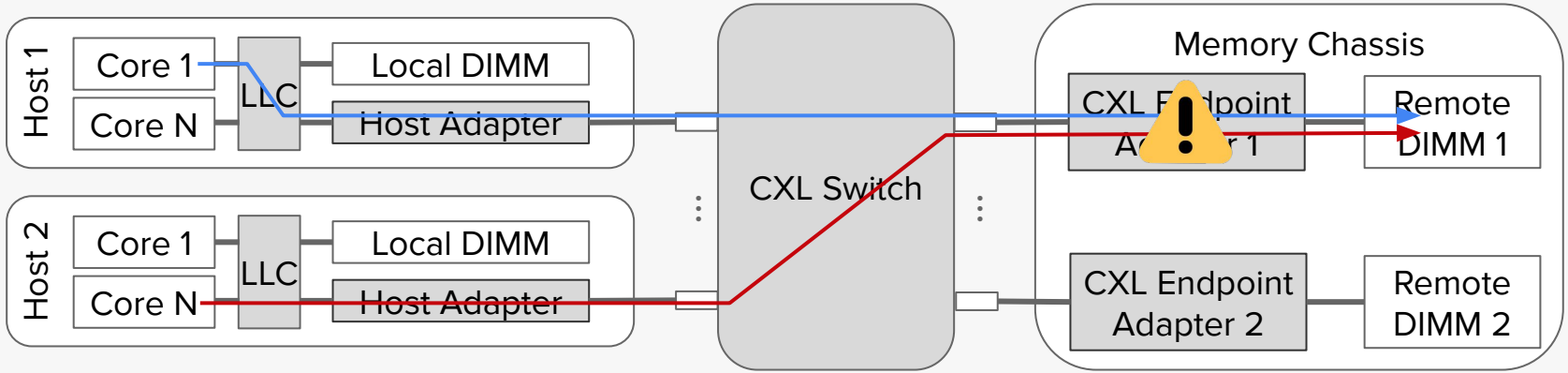




Issue #2: In-fabric Congestion

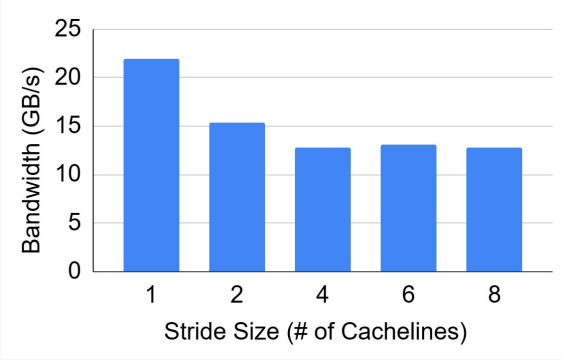
- **Interleaved hetero. memory requests**
  - Cacheline-size accesses: 6.9x slowdown
  - Head-of-line (HoL) blocking

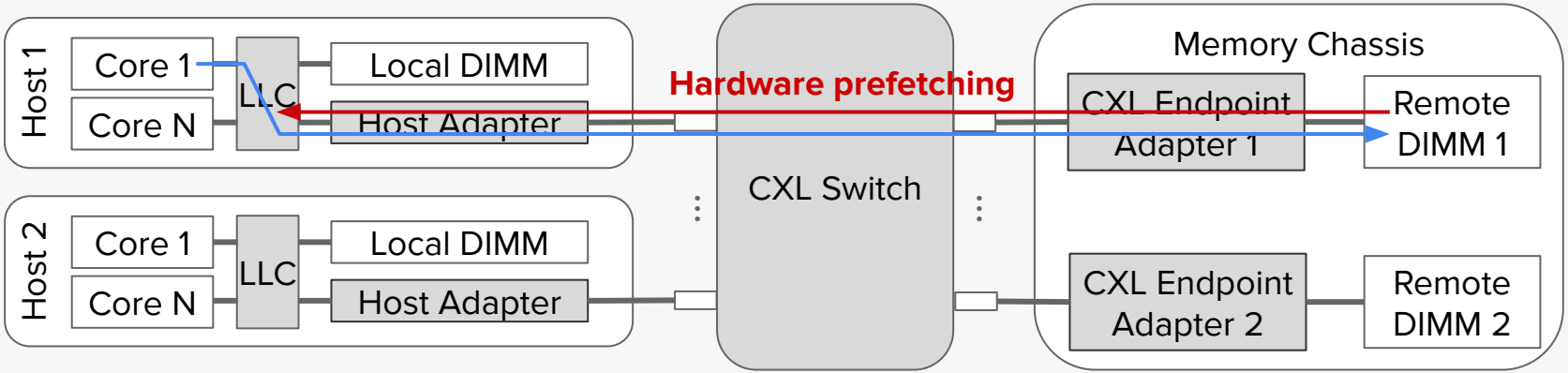




Issue #3: Unmanaged Host-rDIMM Interaction

- **Unpredicted bandwidth oversubscription**
  - Throughput dropped by 41.6%
  - Host prefetching generates more traffic than expected.





Root Cause: Unmanaged data streams in switched CXL memory pooling cause severe performance interference.

Outline

- Characterize Switched CXL Memory Pooling
- **MemChannel Design and Implementation**
- MemChannel Evaluation
- Conclusion

MemChannel Overview

- **Overall design:**
  - Construct a data pipe between a host core and a CXL DIMM
  - Mitigate interference from contending memory streams
  - Regulate remote memory access based on the CXL resources and demands
- **Design goals:**
  - High utilization
  - Efficient multi-tenancy
  - Low overheads

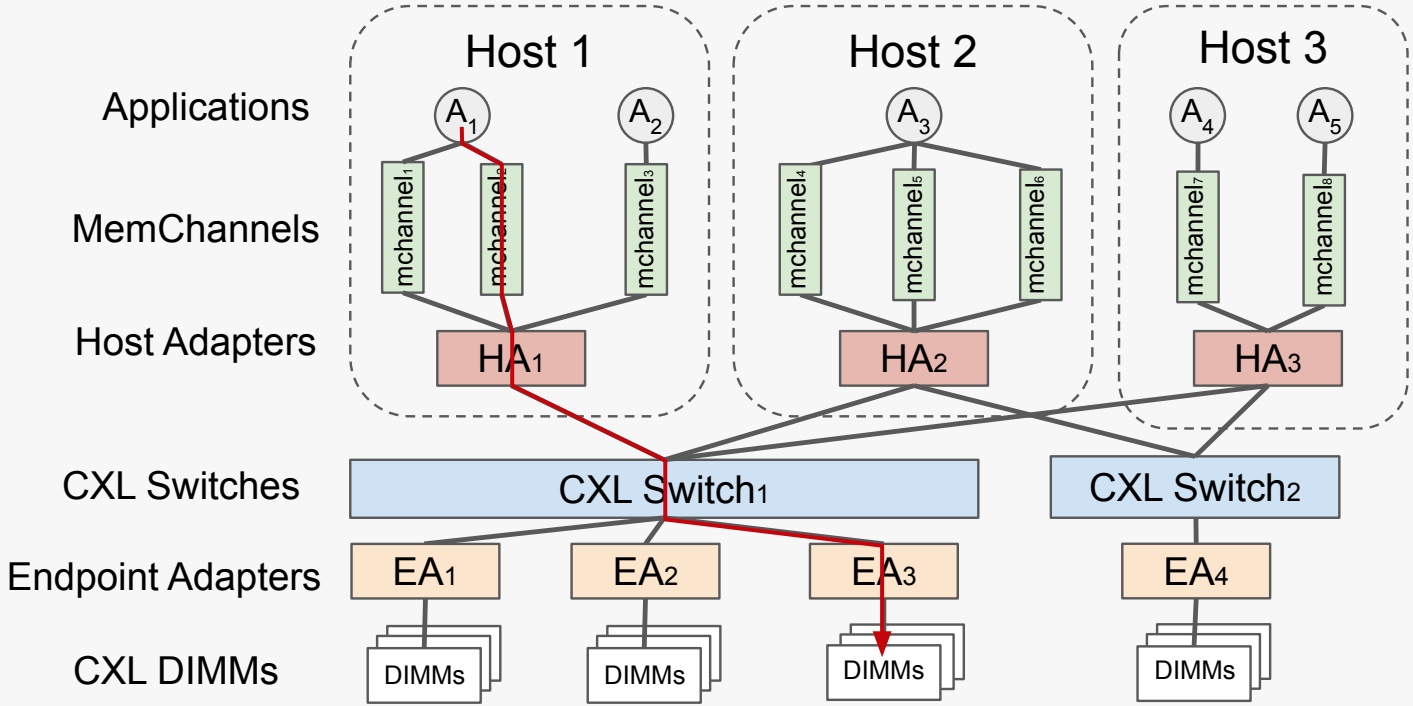
Challenges

- **C1: Hardware non-programmability and opaqueness**
  - Impractical to implement in-network transport layer
- **C2: Implicit transmission**
  - Non-trivial to track individual CXL load/store transactions
- **C3: Fine-grained cross-layer traffic monitoring**
  - Difficult to pinpoint the congested component

Proposed Techniques

Challenges	Techniques
C1: Hardware non-programmability	T1: Sender-driven admission control
C2: Implicit transmission	T2: CSFQ-inspired transport stack
C3: Fine-grained traffic monitoring	T3: Runtime hardware capacity probing

System Model

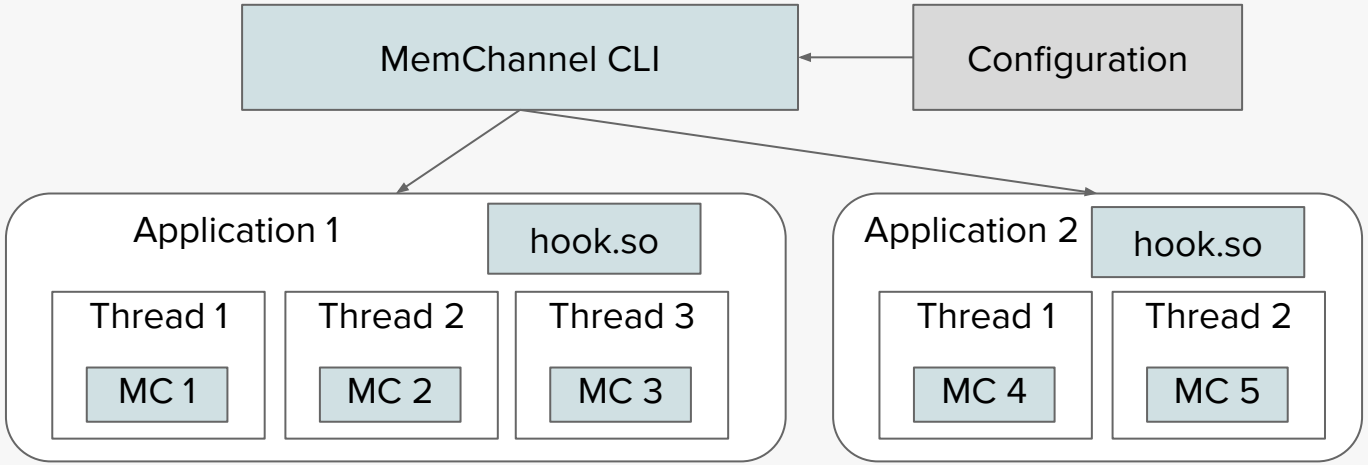


System Model

Interfaces API	Description
mchannel_init	Initialize mchannels from user configs, identify CXL data paths, and allocate shared metadata
mchannel_open	Allocate thread resources, bind CXL memory, and arm timers for mchannel_sched
mchannel_close	Close the thread's mchannel to the remote DIMM, and free allocated resources
mchannel_sched	Enforce bandwidth limits by monitoring CXL requests and controlling thread execution

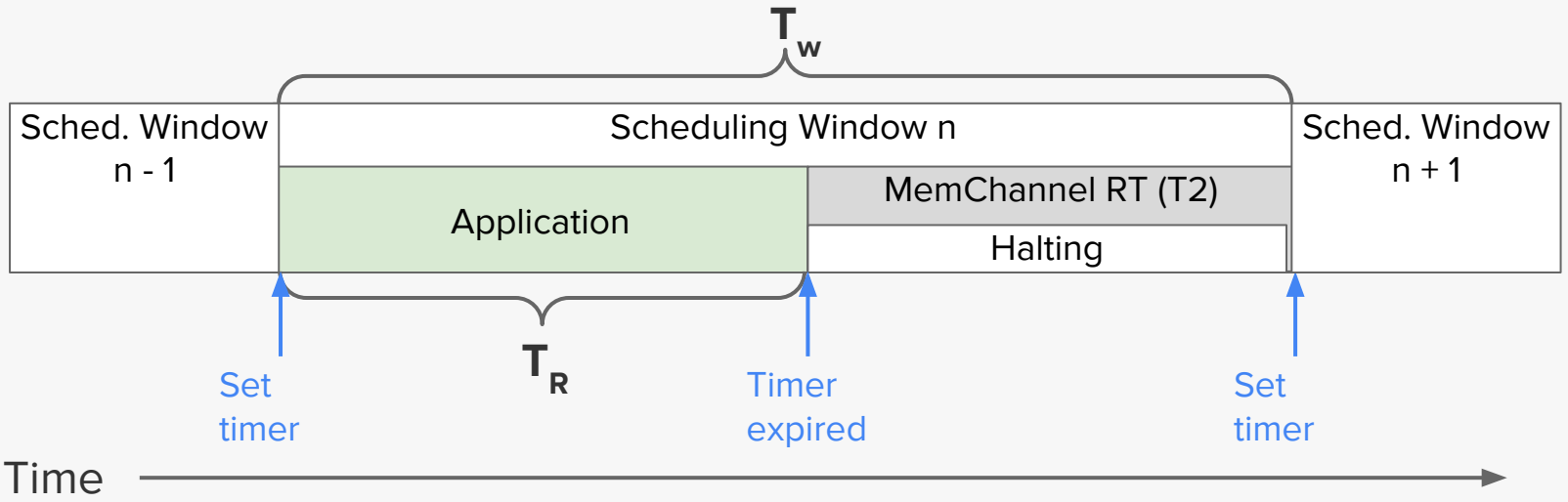
T1: Sender-driven Admission Control

- Support unmodified applications
  - Launch applications as subprocesses based on the configurations
  - Hook into applications (LD_PRELOAD) to create ***mchannels***



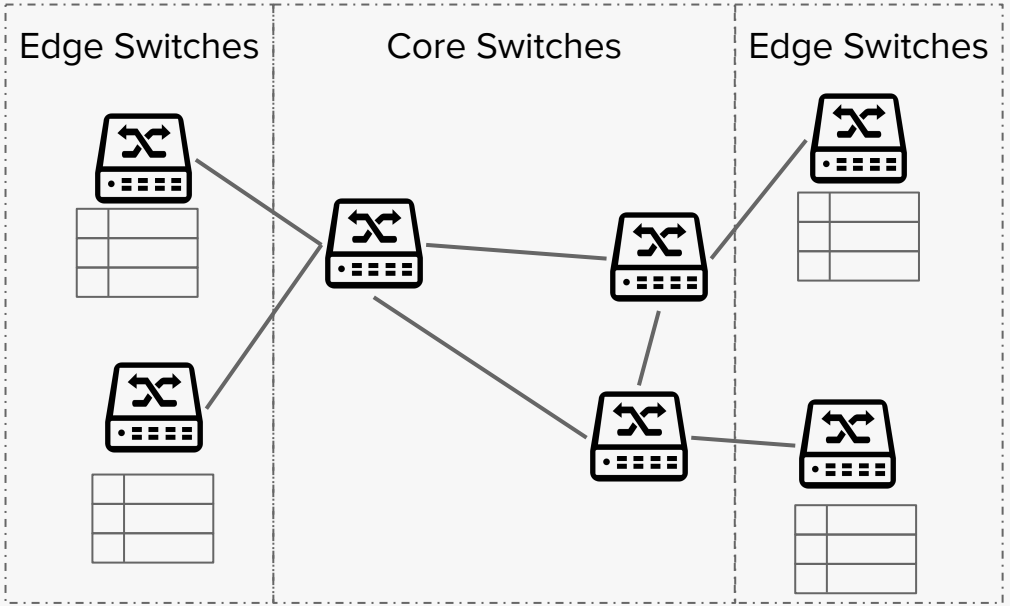
T1: Sender-driven Admission Control

- Sender-driven time-based rate control
  - Adjust application running time based on the transport algorithm
  - Arm a POSIX timer



Core-Stateless Fair Queueing Background

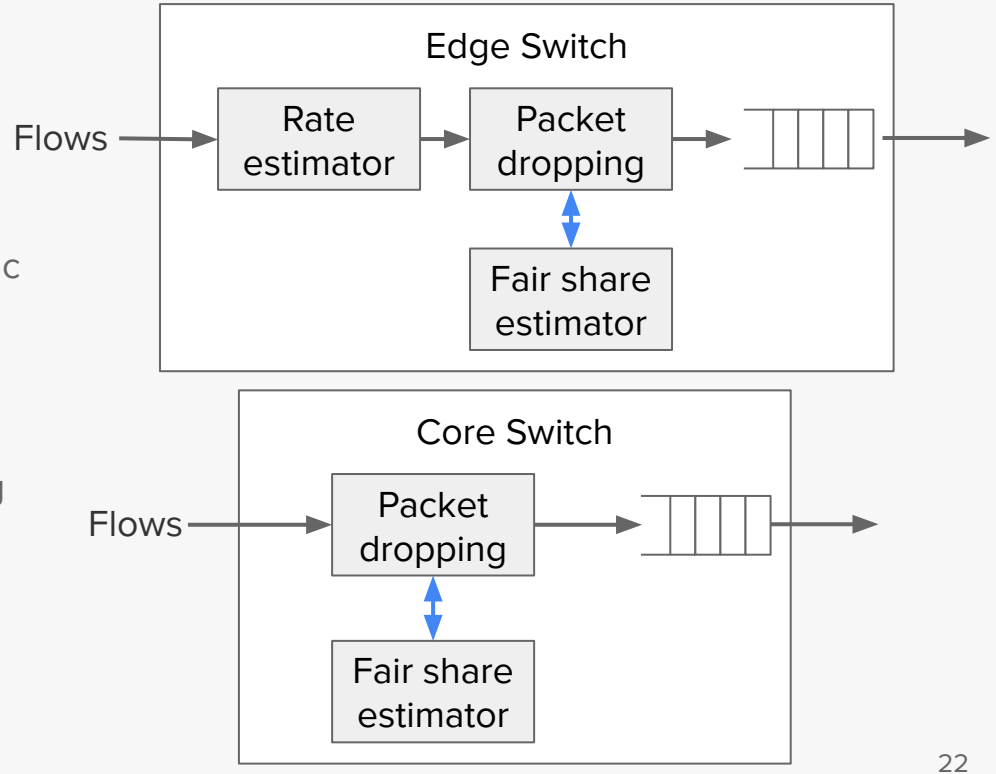
- Edge switches
  - Maintain per-flow state
  - Enforce max-min fairness
- Core switches
  - Operate on aggregated data
  - Enforce max-min fairness



Core Stateless Fair Queueing

CSFQ Techniques

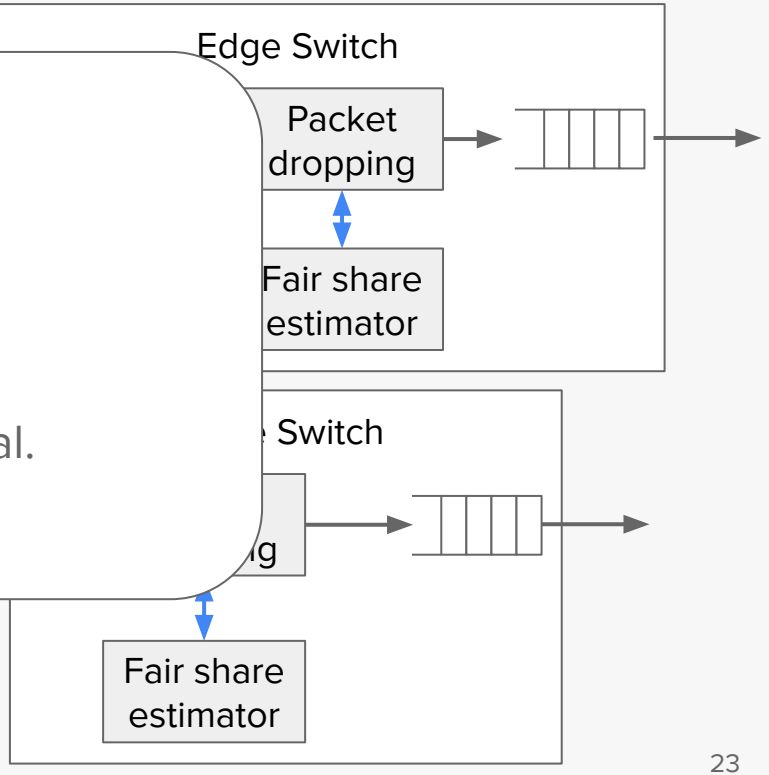
- Rate estimation
  - Compute individual host flow rates
- Fair share estimation
  - Calculate fair-share based on total traffic vs. capacity.
- Traffic control
  - Regulate flow via probabilistic dropping



CSFQ Techniques

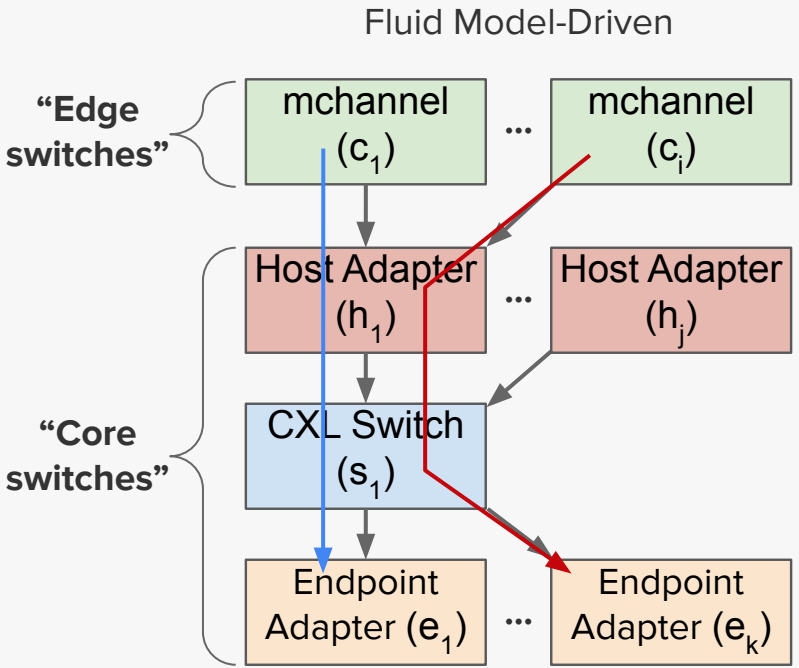
- T1: rate estimator
  -
- T2
  -
- T3
  -

- Why CSFQ for CXL?
  - Scalable
  - Edge-driven
  - Lightweight
- However, applying CSFQ is non-trivial.



T2: CSFQ-inspired Transport Stack

- Model mchannels as edge switches
- Model CXL devices as core switches
  - Host/Endpoint adapters, CXL switches
- Flows in our context are:
  - Directional memory streams from applications
  - Start from mchannels and reach CXL DIMMs

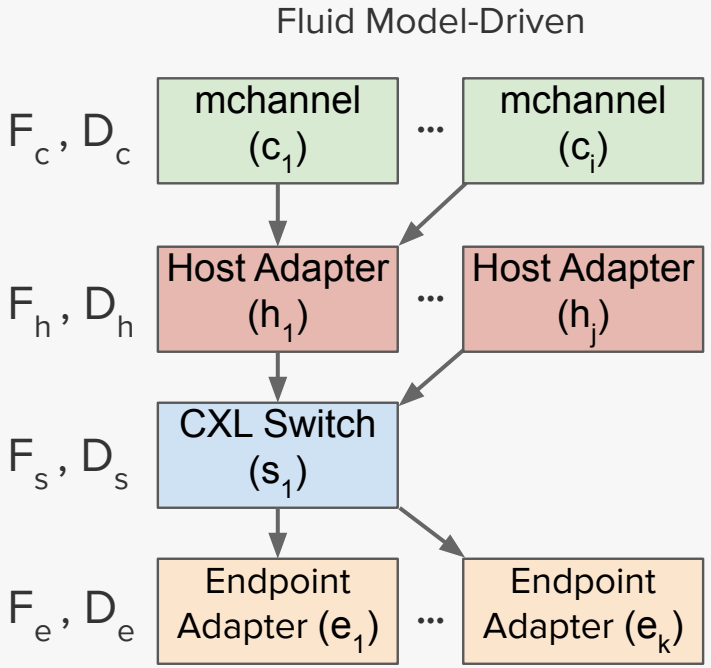


T2: CSFQ-inspired Transport Stack

- **Rate estimation**

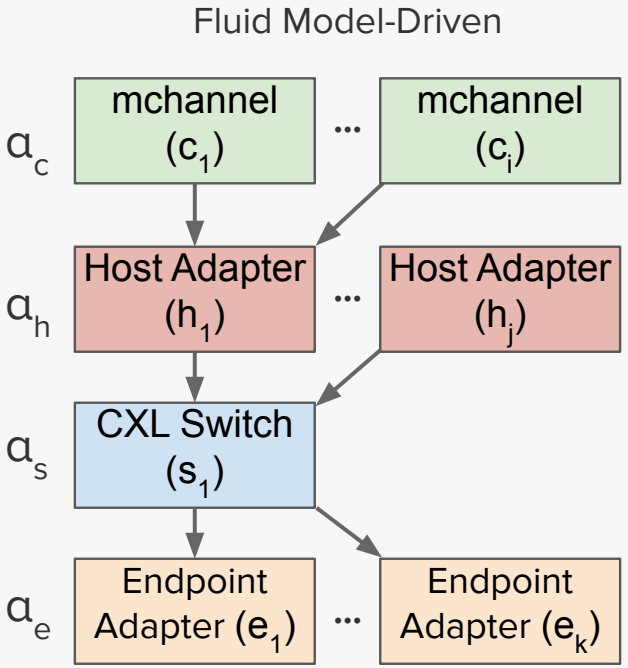
Arrival Rates → Access Rates (F)
Arrival Rates → Demand (D)

- CXL bandwidth monitoring
  - Read off-core response (OCR) event counters
  - Divide by window size (T_w) and running time (T_R)
- Calculate aggregated rates directly
  - Access rate
  - Demand



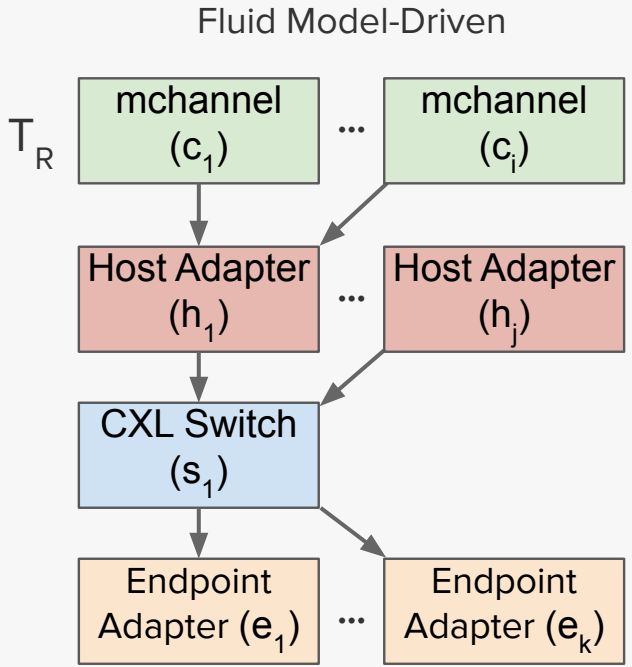
T2: CSFQ-inspired Transport Stack

- **Fair share estimation**
- Calculate hardware fair share on hosts
- Estimate fair share rate (α) iteratively
  - Aggregated access rate (F)
  - Aggregated demand (D)
  - Hardware capacity (C)
- *mchannel* fair share
  - Minimum fair share along the flow path



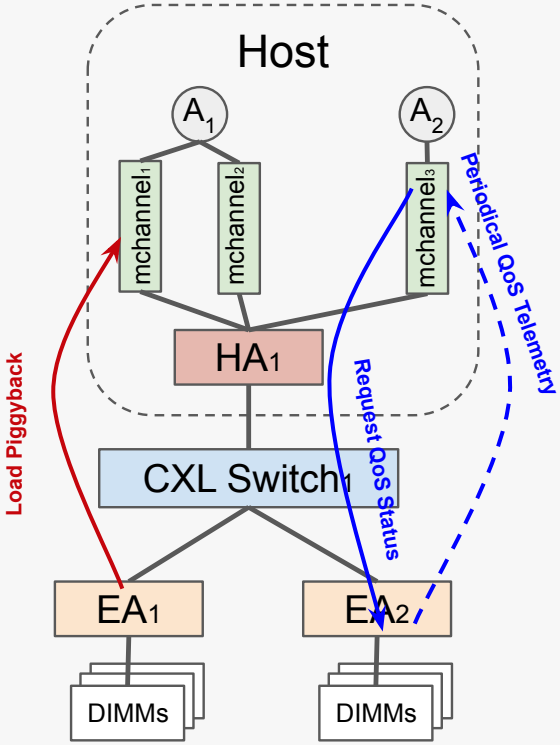
T2: CSFQ-inspired Transport Stack

- **Traffic control**
- Adjust application running time (T_R)
  - Fair-share-to-demand ratio
  - Allocate running time proportionally



T3: Runtime Hardware Capacity Probing

- Congestion Signals
  - Device internal load
    - Piggyback in memory responses
  - QoS telemetry
    - Report periodically
- Translate congestion signals into loading factors
- Load-based congestion control
  - Congestion free: additive increase
  - Congestion avoidance: multiplicative decrease



Outline

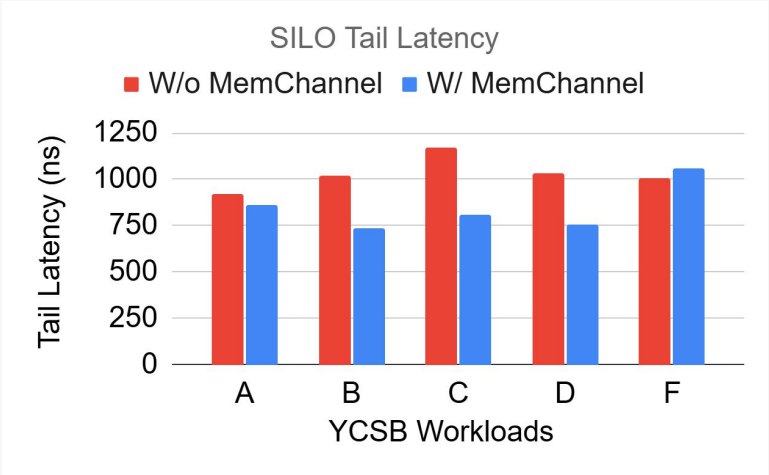
- Characterize Switched CXL Memory Pooling
- MemChannel Design and Implementation
- **MemChannel Evaluation**
- Conclusion

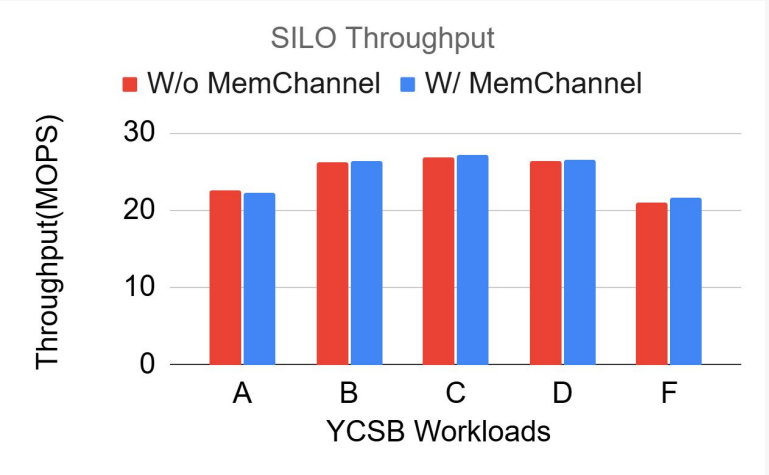
Experimental Setup

- Applications
  - MICA, SILO, GAP benchmark suite, SPEC CPU 2017
- Performance metrics
  - Application throughput
  - Average/tail latency
  - CXL memory bandwidth
- Code
  - https://github.com/netlab-wisconsin/MemChannel

Application Performance

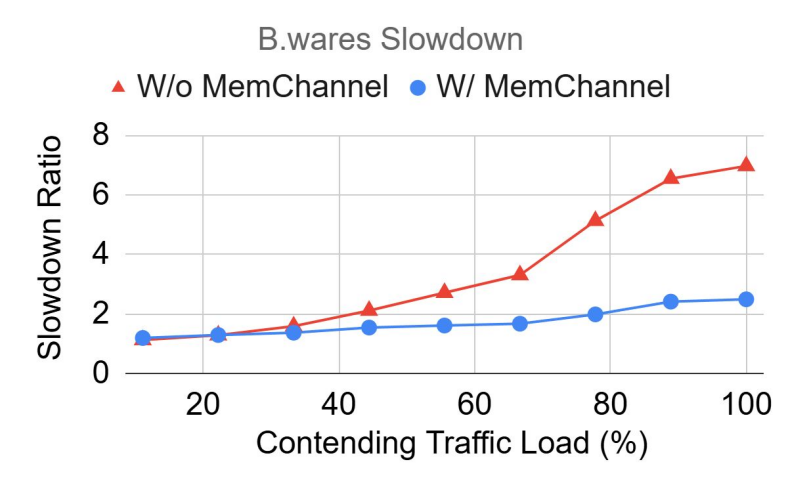
- Evaluate SILO performance under five YCSB workloads
- Reduce tail latency by 17.4% with similar throughput

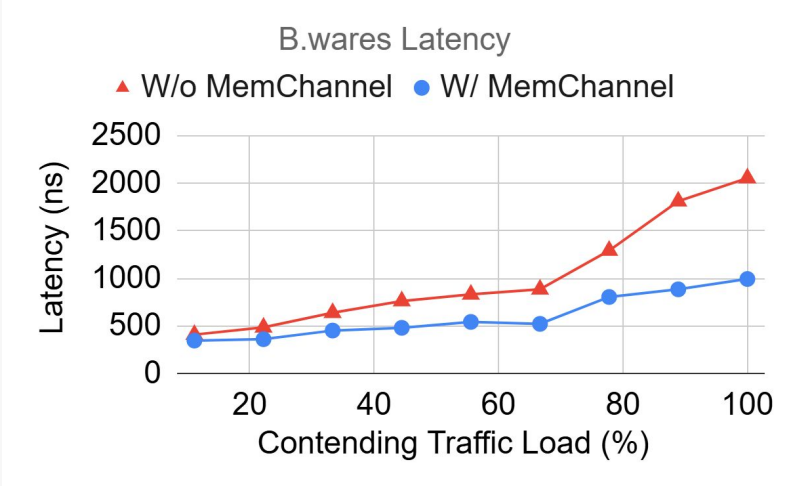




Performance Isolation

- Evaluate B.wares(CPU SPEC) performance under contending memory flows
- 4.5x throughput increase and 51.6% latency reduction





More Evaluations

- Inter-Host Performance Isolation
- Fairness
- System Overheads
- MemChannel Adaptiveness
- .....

Conclusion

- Unmanaged data streams causing severe performance interference
- MemChannel: a transport layer for switched CXL memory pooling
- Data pipes between cores and CXL DIMMs with controlled CXL requests
  - Sender-driven admission control
  - CSFQ-inspired transport stack
  - Runtime hardware capacity probing

https://github.com/netlab-wisconsin/MemChannel

Thanks!