Clio: A Hardware-Software Co-Designed Disaggregated Memory System

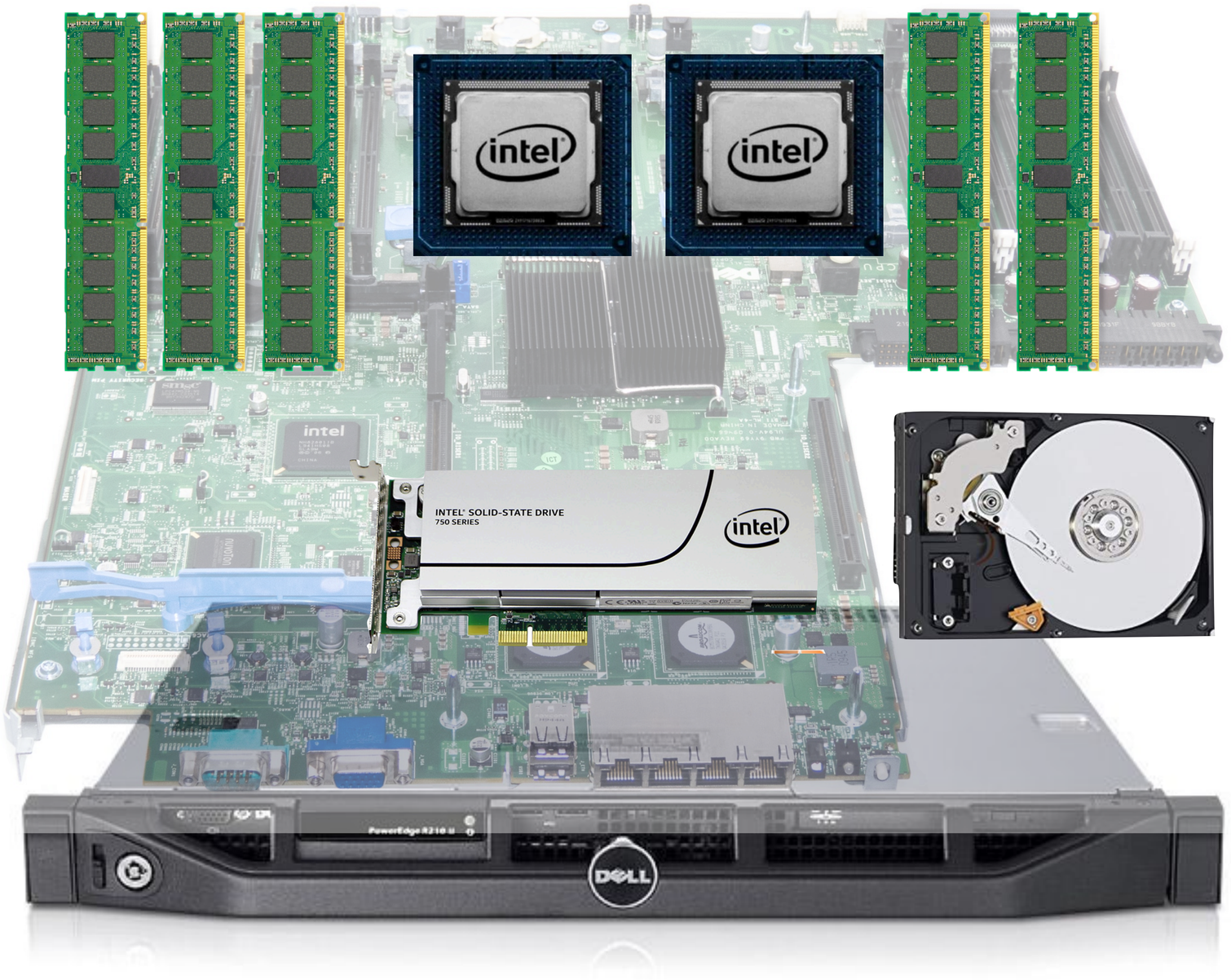
Zhiyuan Guo*, Yizhou Shan* (* equal contribution)
Xuhao Luo, Yutong Huang, **Yiying Zhang**

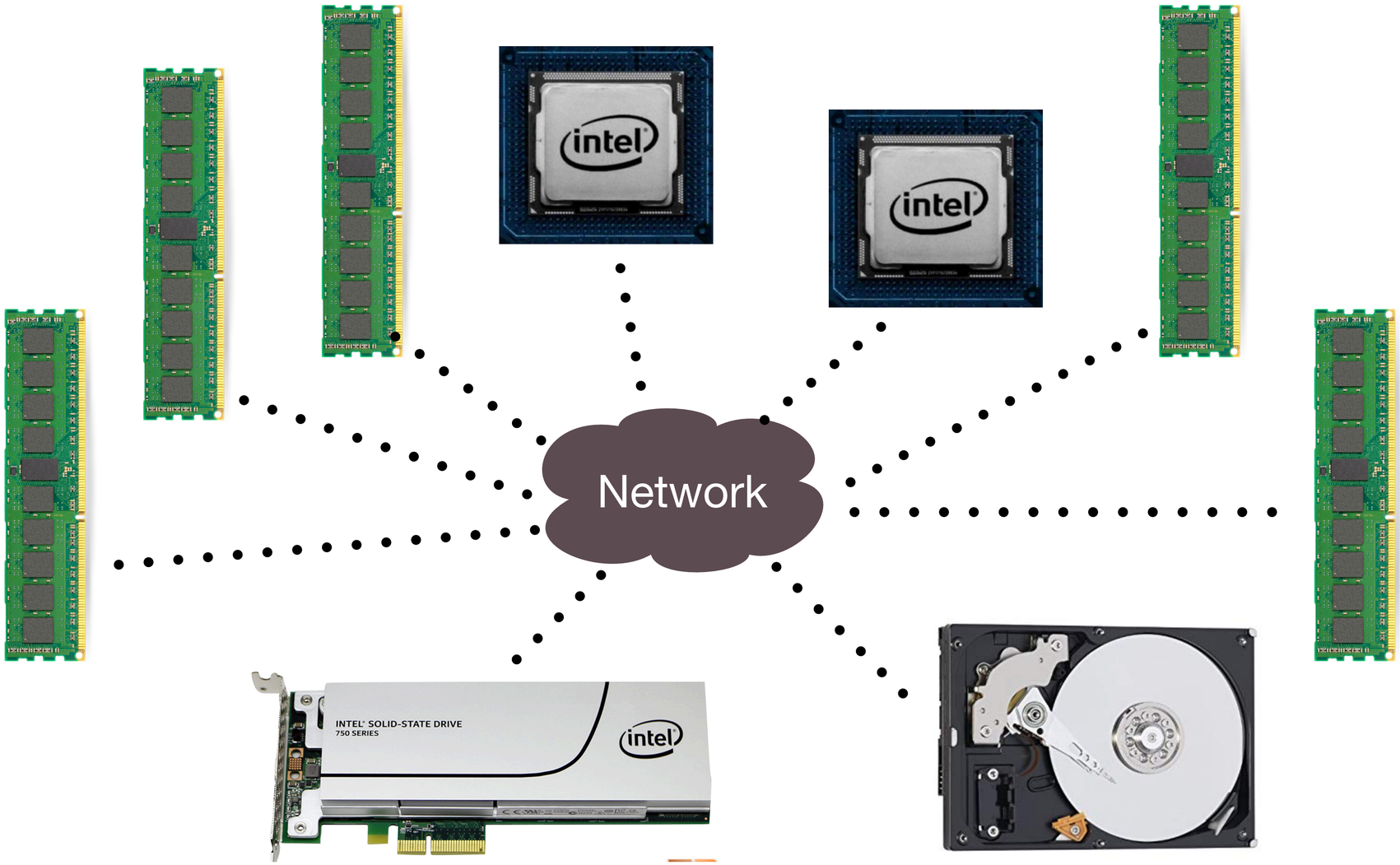
UCSDCSE
Computer Science and Engineering

WukLab

Hardware Resource Disaggregation:

Breaking monolithic servers into distributed, network-attached hardware components


intel
intel
intel
INTEL® SOLID-STATE DRIVE
750 SERIES
intel
DELL


Network
INTEL SOLID-STATE DRIVE
750 SERIES

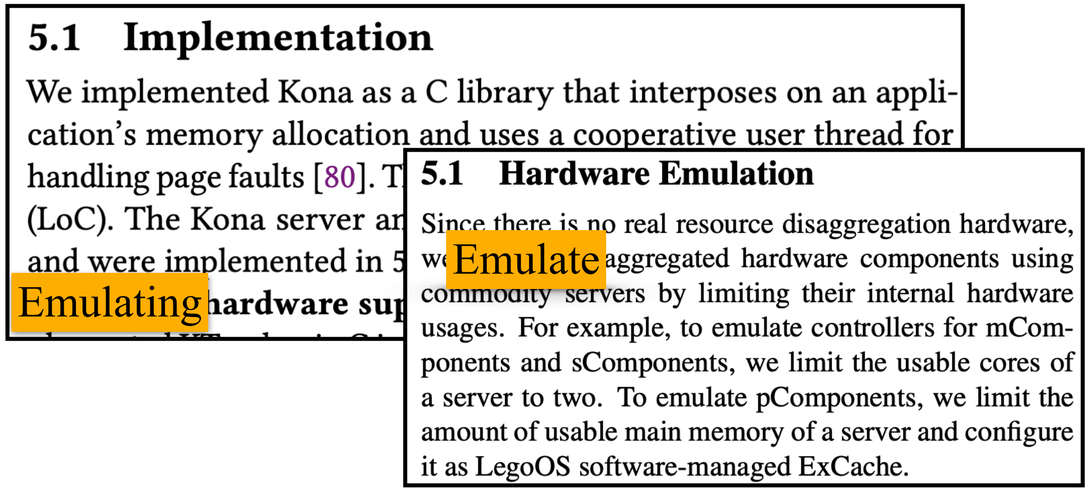
Workshop on Resource Disaggregation and Serverless (WORDS 2022)

- Website: https://www.wordsworkshop.org/
- Submission deadline: **9/29/2022**
- Workshop date: 11/17/2022 (virtual or hybrid)
- Types of papers
  - Vision paper, completed new work (up to 5 pages)
  - published work (2 page abstract)
- PC chairs
  - Arvind Krishnamurthy, University of Washington
  - Yiying Zhang, University of California San Diego

Existing Disaggregated Memory Systems

- LegoOS [OSDI '18]
- FastSwap [EuroSys '20]
- AIFM [OSDI '20]
- Semeru [OSDI '20]

....



5.1 Implementation

We implemented Kona as a C library that interposes on an application's memory allocation and uses a cooperative user thread for handling page faults [80]. T
(LoC). The Kona server ar
and were implemented in 5
Emulating hardware sup

5.1 Hardware Emulation

Since there is no real resource disaggregation hardware, we Emulate aggregated hardware components using commodity servers by limiting their internal hardware usages. For example, to emulate controllers for mComponents and sComponents, we limit the usable cores of a server to two. To emulate pComponents, we limit the amount of usable main memory of a server and configure it as LegoOS software-managed ExCache.

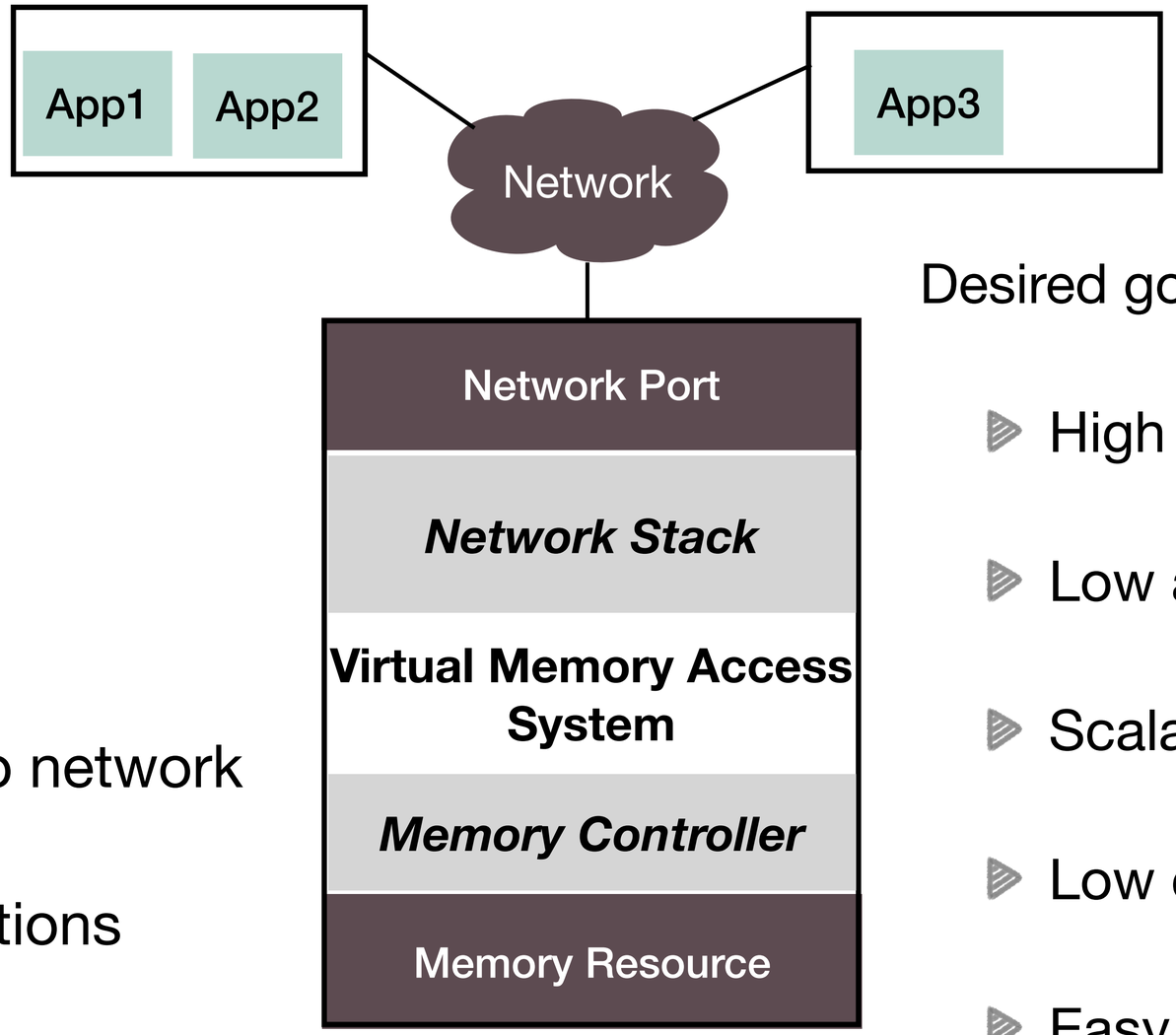
All existing works use **server** to
Build/Emulate disaggregated memory devices

How about real hardware?

Outline

- Introduction
- **Motivation: Why do we need real hardware?**
- Clio Overview: Interface and overall approach
- Design: How we remove “state”
- Implementation and evaluation results

Disaggregated Memory Hardware



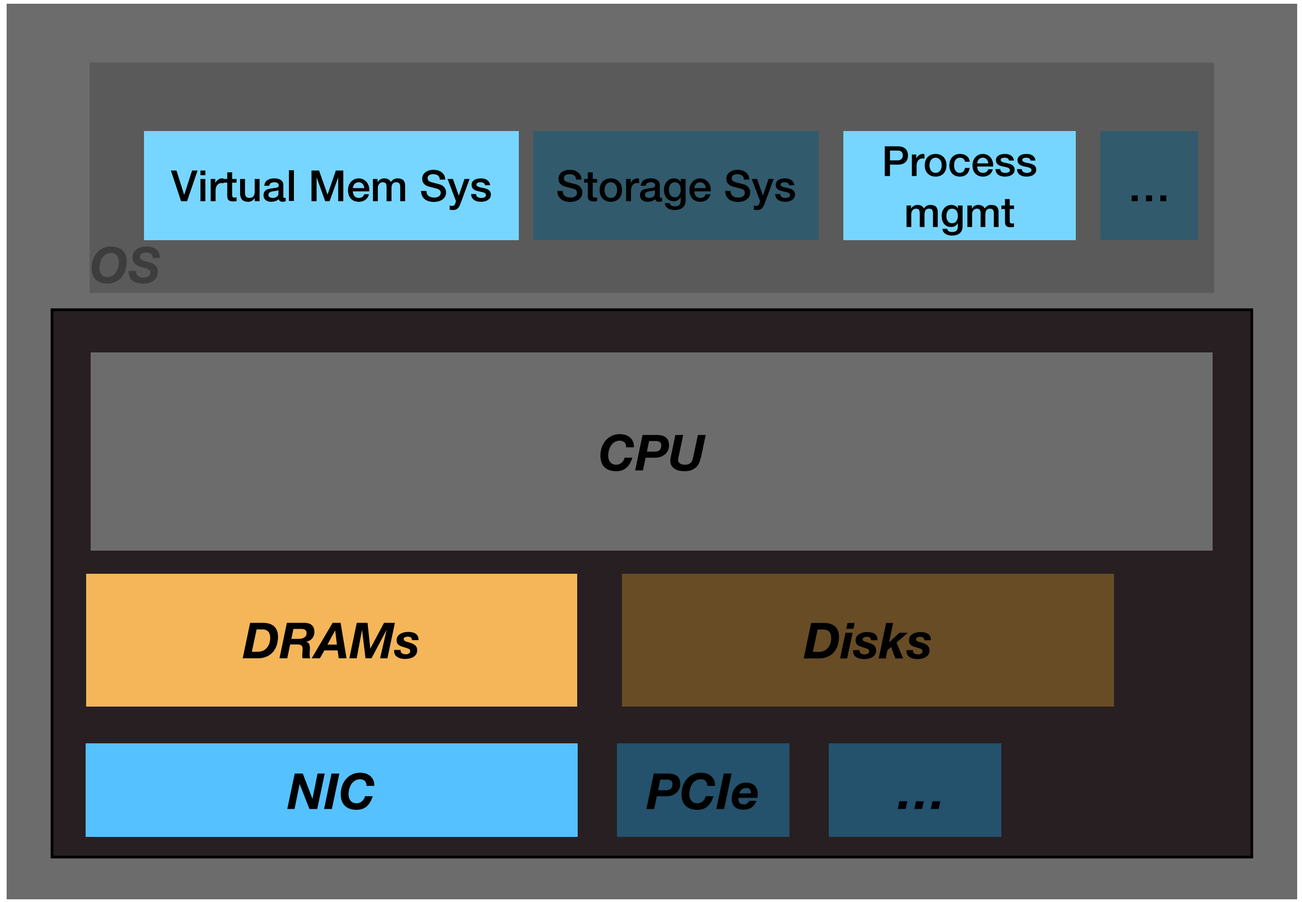
Features

- Standalone
- Host memory
- Directly connect to network
- Shared by applications

Desired goals

- High throughput
- Low avg and tail latency
- Scalability and capacity
- Low cost
- Easy to use and versatile

Could Server Emulation work?



Memory Node (Server)

- Unused resources in server

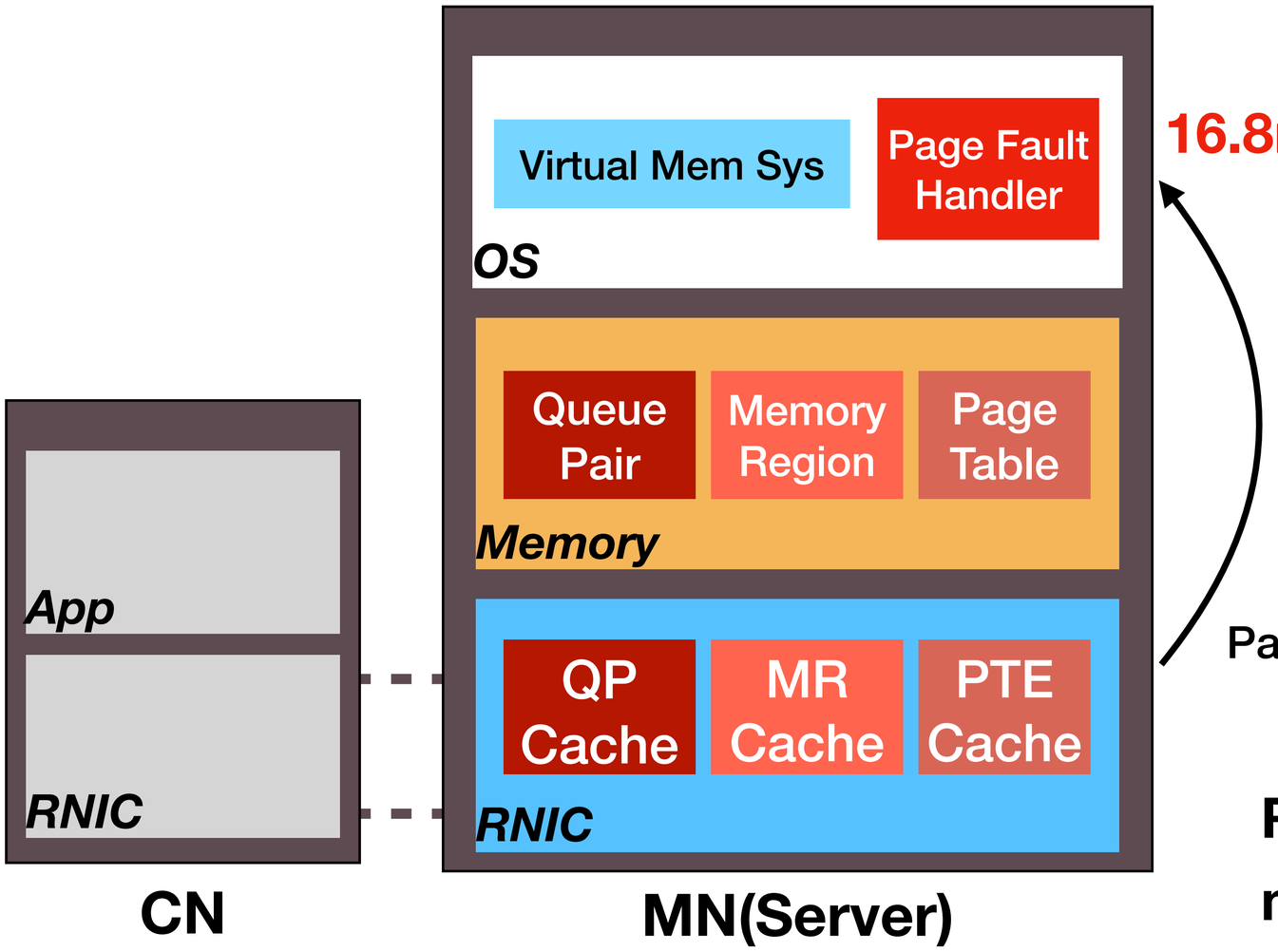
Low Cost ✗

- Limited DRAM size

Capacity ✗

Servers are overkill for memory disaggregation.

Could RDMA work?



16.8ms!

Page Fault

CXL?

Need specialized interconnect

Not immediately available

-Limited NIC cache of OS management structures

Scalability ✗

-Slow page fault operations in data path

Tail Latency ✗

RDMA is not designed for standalone memory disaggregation

What we build: Clio [ASPLOS'22]

a **hardware-based disaggregated memory** system that virtualizes, protects, and manages disaggregated memory at **standalone memory nodes**

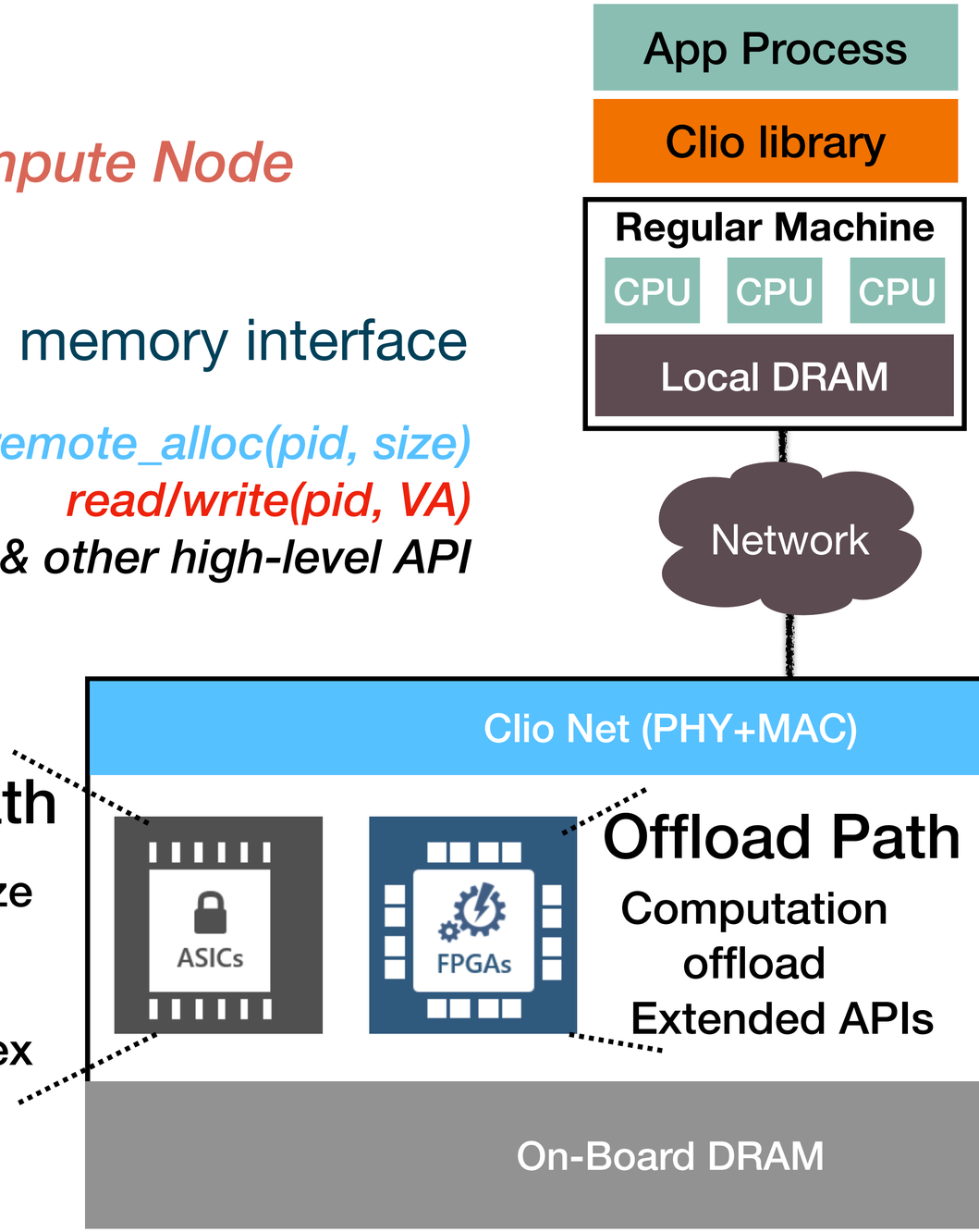
Outline

- Introduction
- Motivation: Why we need real hardware
- **Clio Overview: Interface and overall approach**
- Design: How we remove “state”
- Implementation and evaluation results

Compute Node

virtual memory interface

remote_alloc(pid, size)
read/write(pid, VA)
key-value & other high-level API



Memory Node (Clio Board)

Data Path

Virtualize

Protect

Multiplex

Offload Path

Computation offload

Extended APIs

Control Path

Allocation

Distributed Support

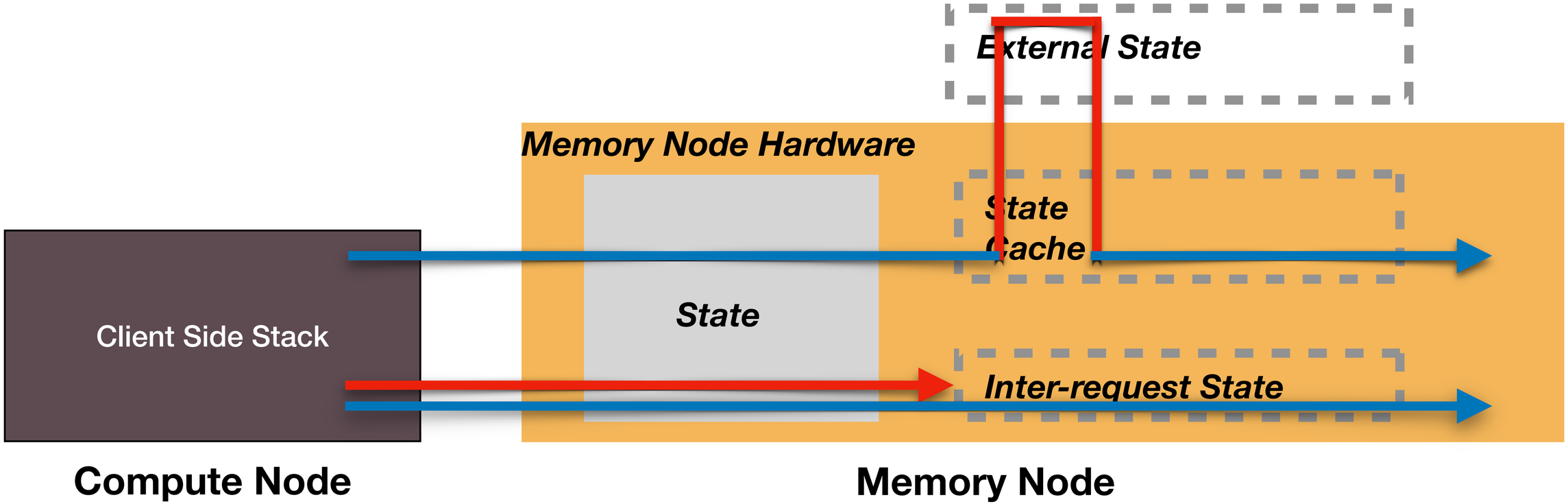
Metadata

Main Idea:

Eliminate *state* from hardware

"*state*":
*M*etadata stored on the memory node that need to be accessed or updated when processing requests.

Benefit of Removing State



- Minimizing state can **reduce cost**, **reduce tail latency**, and improve **scalability**
- Avoiding inter-request state can **make the pipeline smooth**

Outline

- Introduction
- Motivation: Why we need real hardware
- Clio Overview: Interface and overall approach
- **Design: How we remove “state”**
- Implementation and evaluation results

How to eliminate state from MN hardware

Overall Approach: Co-designing hardware, network, and software

1. Reduce state in disaggregated memory protocol
2. Move state to compute node
3. Remove state from critical path
4. Optimize hard-to-remove state to bounded size

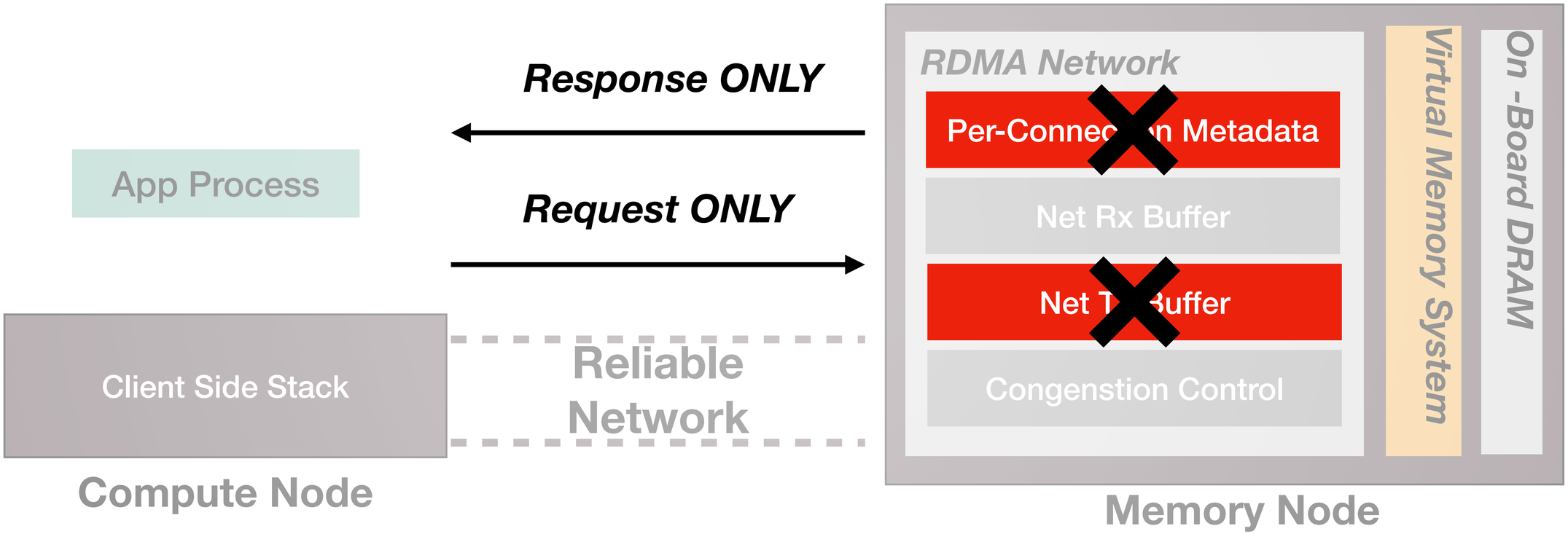
How to eliminate state from MN hardware

Overall Approach: Co-designing hardware, network and software

1. **Reduce state in disaggregated memory protocol**
2. Move state to compute node
3. Remove state from critical path
4. Optimize hard-to-remove state to bounded size

Reduce state in disaggregated memory protocol >

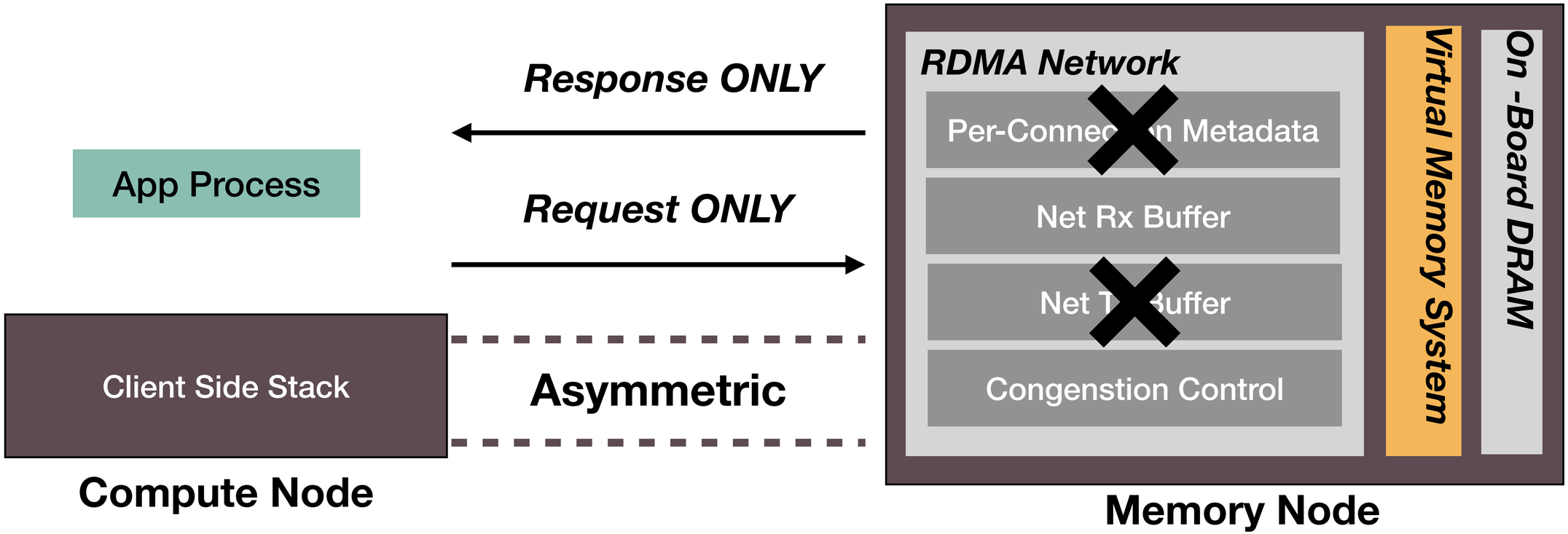
Asymmetric Memory Request Protocol



- **Observation:** accesses to MNs are always in the **request-response** style

Reduce state in disaggregated memory protocol >

Asymmetric Memory Request Protocol

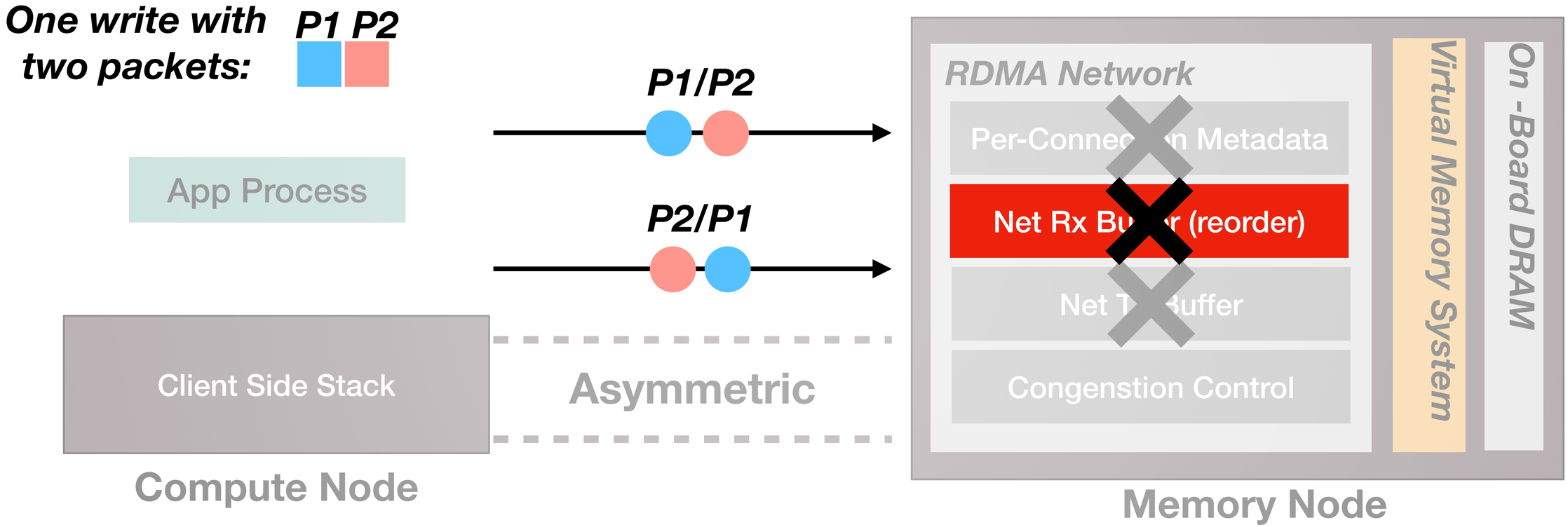


- **Observation:** accesses to MNs are always in the **request-response** style

  Asymmetric RPC-style, connection-less network protocol

Reduce state in disaggregated memory protocol >

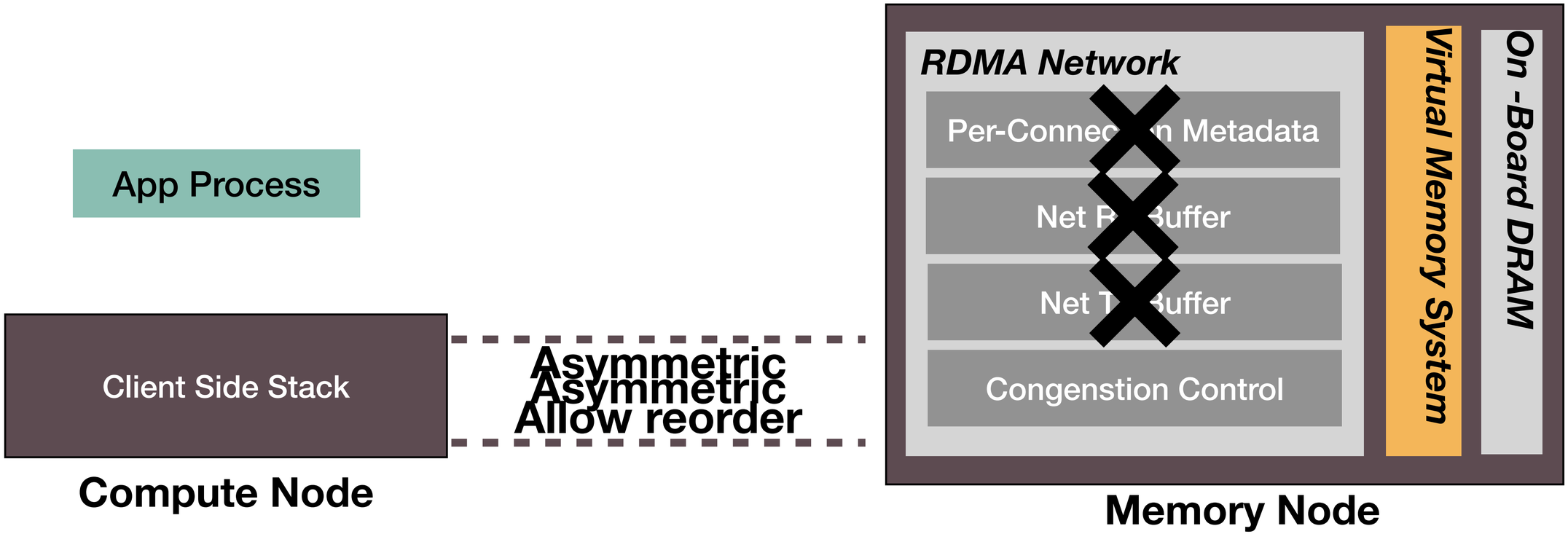
Network Ordering



- **Observation: Memory requests can tolerate certain network reordering**

Reduce state in disaggregated memory protocol >

Network Ordering



- **Observation: Memory requests can tolerate certain network reordering**

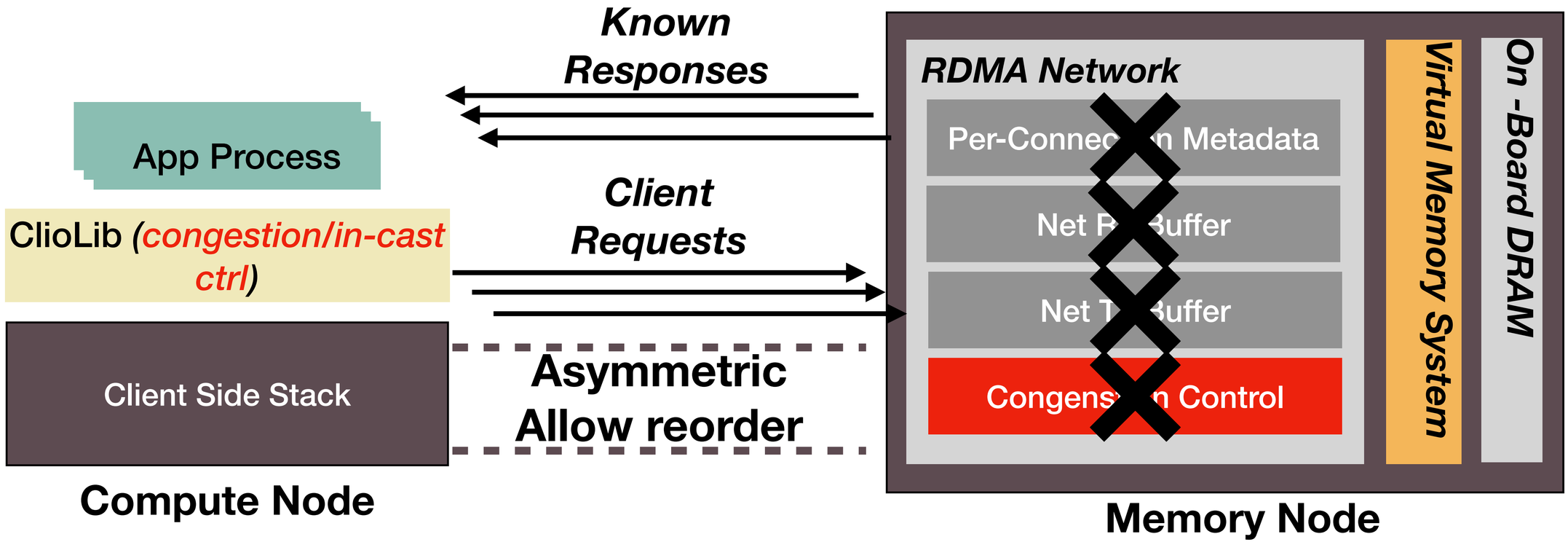
  Release networking ordering requirements

How to eliminate state from MN hardware

1. Reduce state in system protocol
2. **Move state to compute node**
3. Remove state from critical path
4. Optimize hard-to-remove state to bounded size

Move state to Compute Node >

Congestion and Flow Control

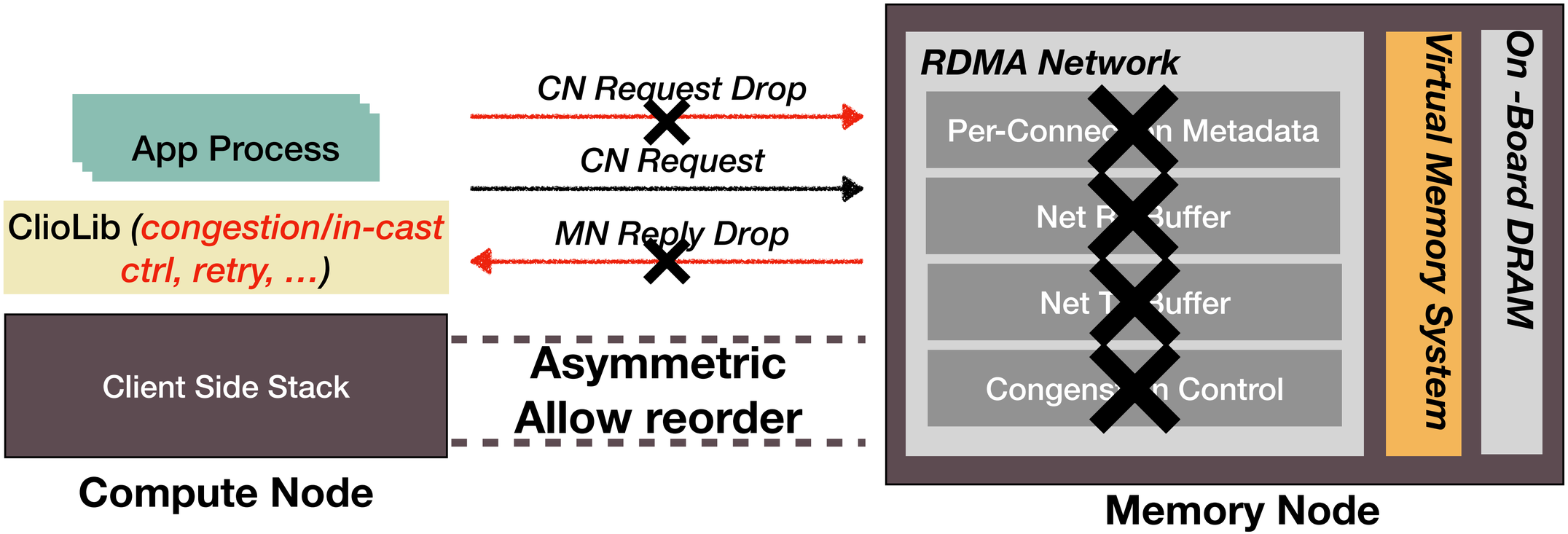


- **Observation: CN knows the size of both requests and responses**

  Move congestion and in-cast control to CN side

Move state to Compute Node >

Handle Retry

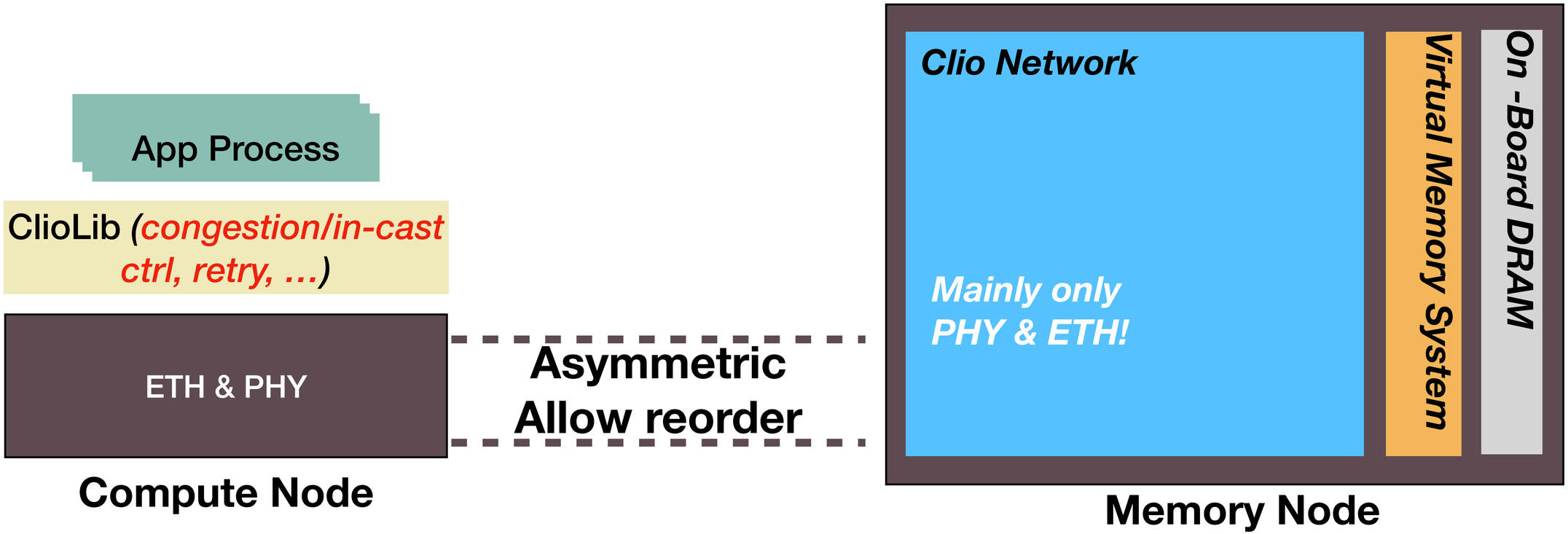


- **Observation**: Network losses are rare and fully observed by CN

  Let CN side software handling retry

Move state to Compute Node >

Handle Retry



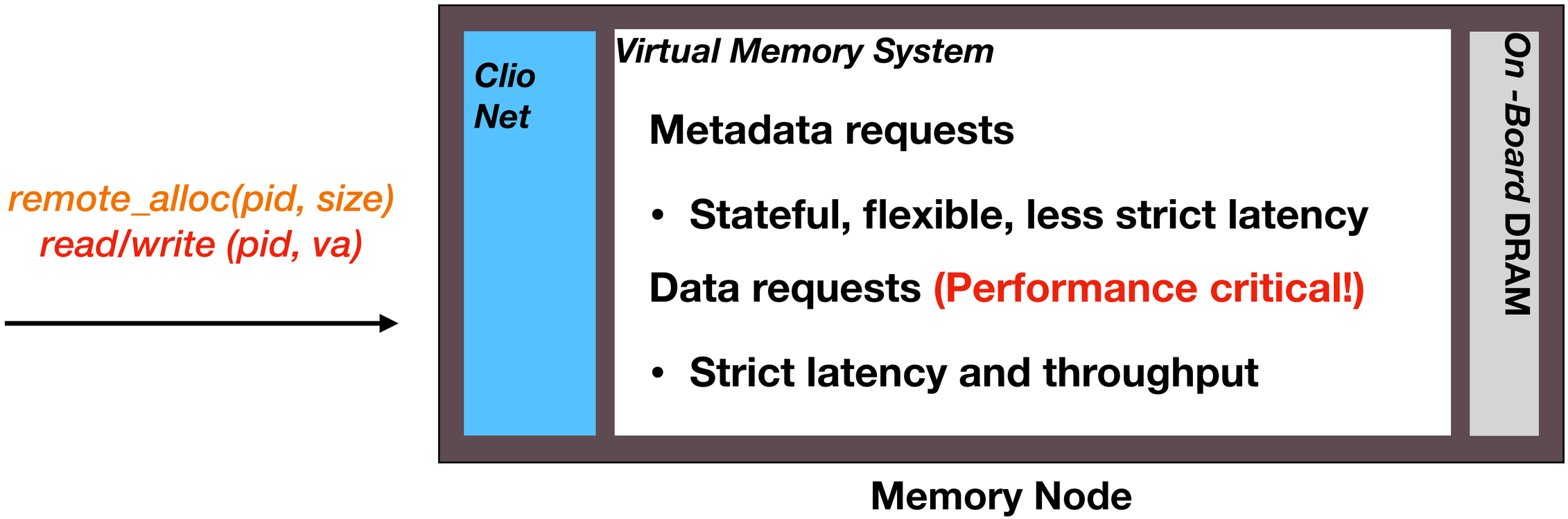
How to eliminate state from MN hardware

1. Reduce state in system protocol
2. Move state to compute node
3. **Remove state from critical path**
4. Optimize hard-to-remove state to bounded size

Remove state from critical path >

Splitting Fast Path and Slow Path

remote_alloc(pid, size)
read/write (pid, va)

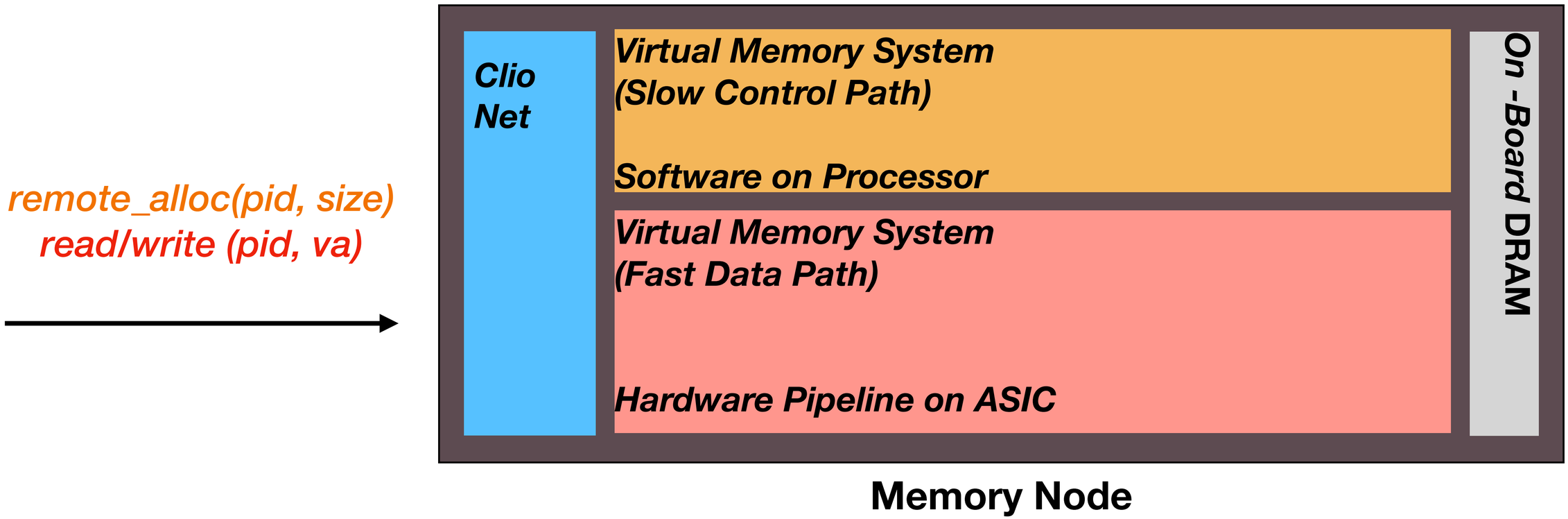


- **Observation**: Metadata and data requests have different state and performance requirements

Remove state from critical path >

Splitting Fast Path and Slow Path

remote_alloc(pid, size)
read/write (pid, va)

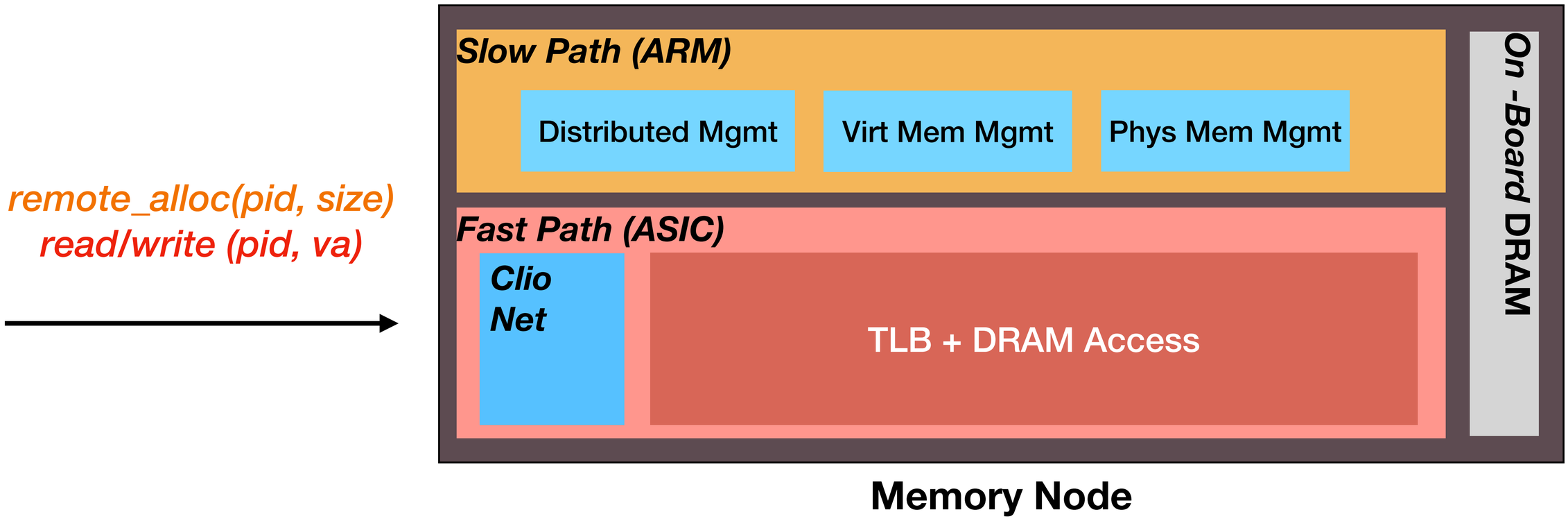


Splitting virtual memory system into fast path and slow path

Remove state from critical path >

Splitting Fast Path and Slow Path

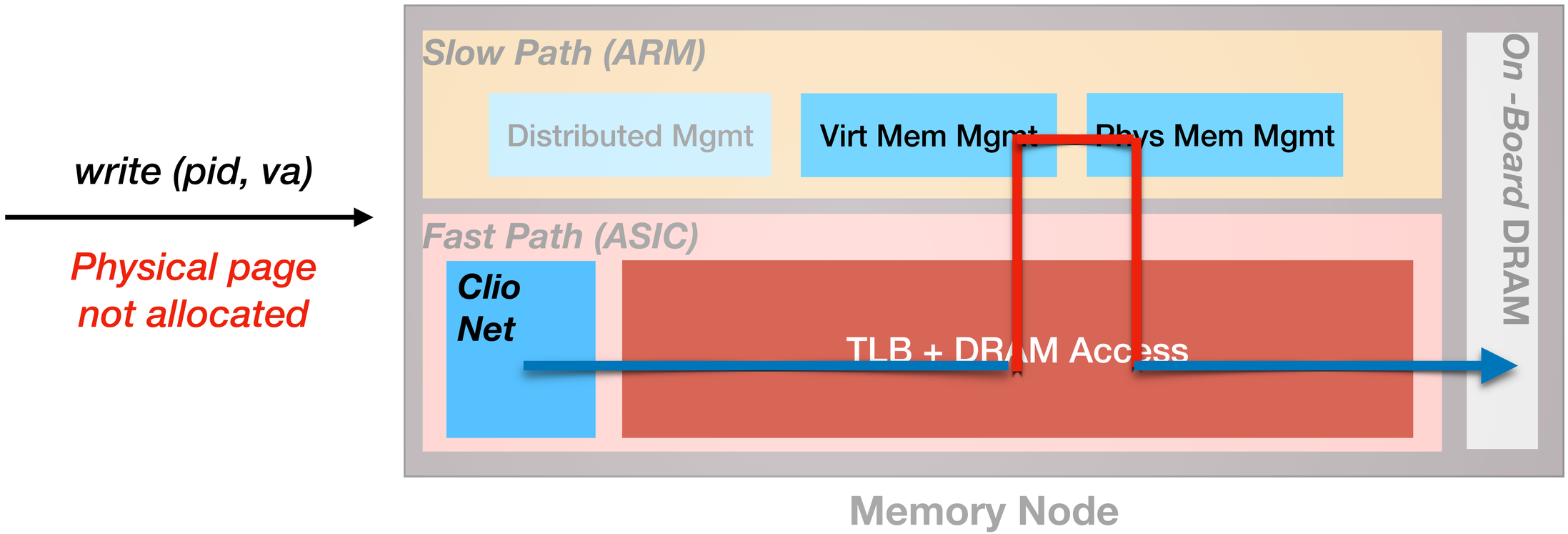
remote_alloc(pid, size)
read/write (pid, va)



Solution: Splitting virtual memory system into fast path and slow path

Remove state from critical path >

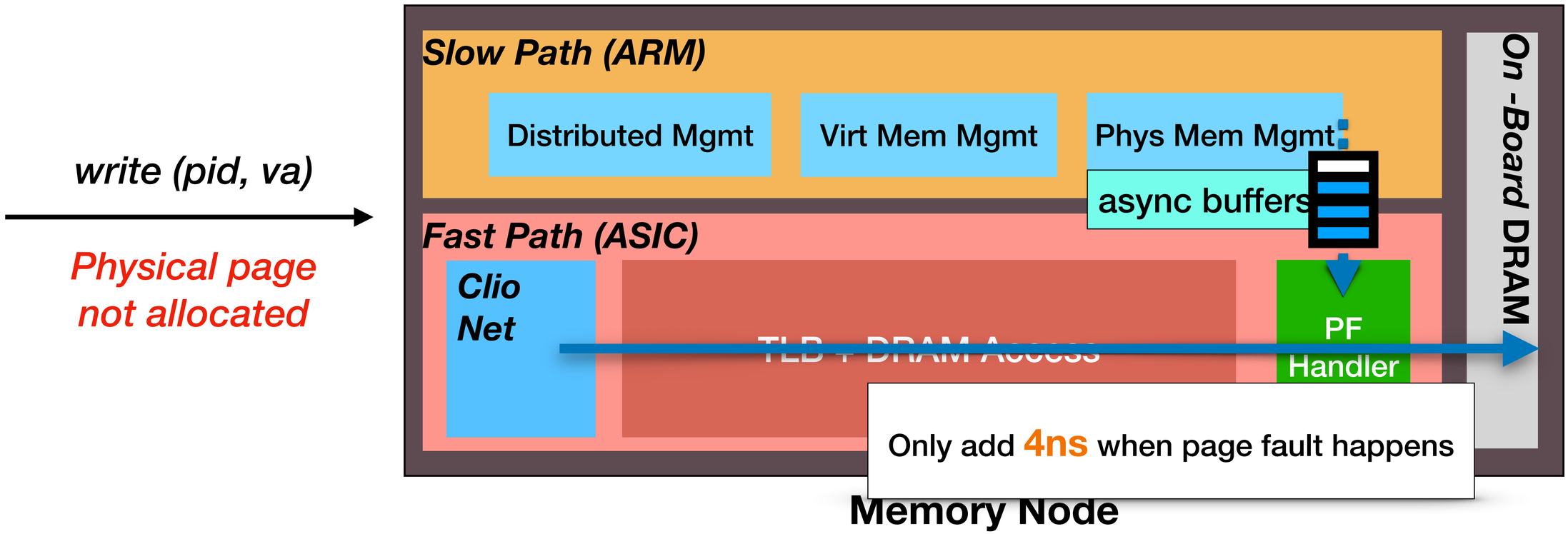
Handling Page Fault



- **Observation**: **Access requests with pagefault need stateful allocation operations**

Remove state from critical path >

Handling Page Fault



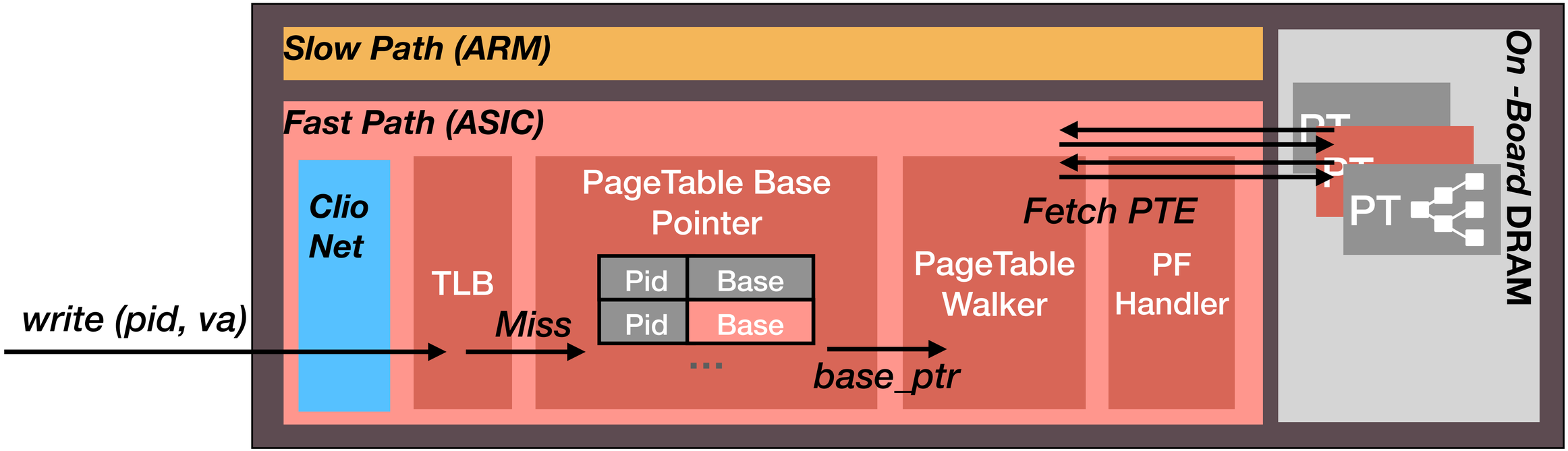
- Observation: Access requests with pagefault need stateful allocation operations
  Solution: Handle page fault pre-allocated physical pages in async buffers

How to eliminate state from MN hardware

1. Reduce state in system protocol
2. Move state to compute node
3. Remove state from critical path
4. **Optimize hard-to-remove state to bounded size**

Optimize state to bundled size >

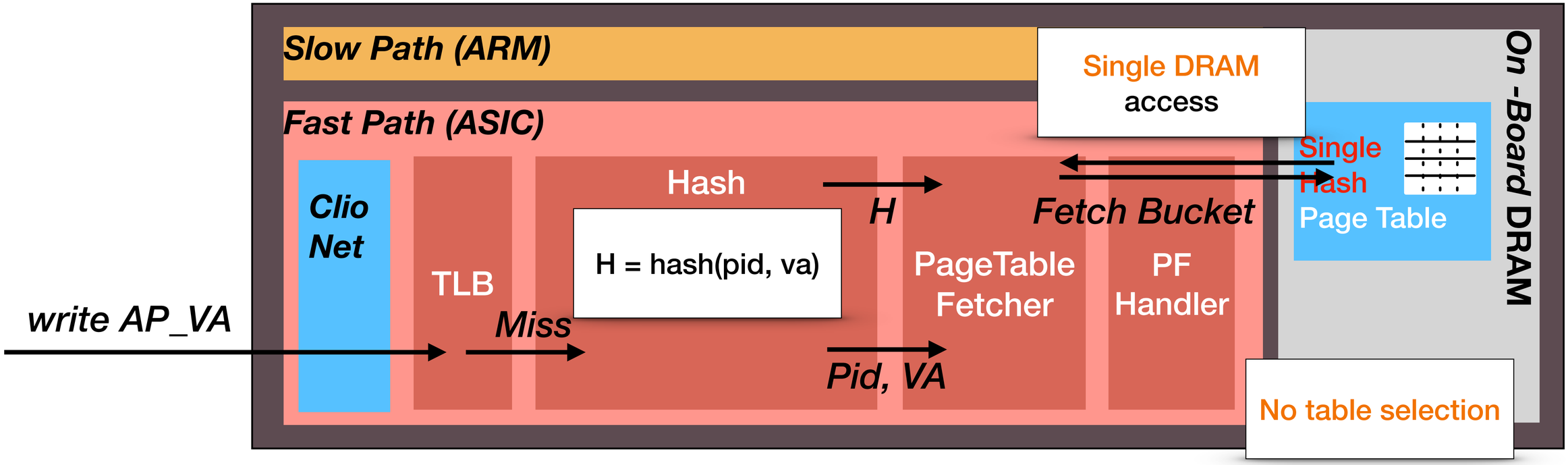
Traditional Page Table Design (Strawman)



- **Observation: Size of base pointer table and page tables grow with number of clients, needs multiple DRAM accesses to walk page tables**

Optimize state to bundled size >

Hash-Based Page Table



Hash Page Table for **bounded size** and **access time (single DRAM access)**

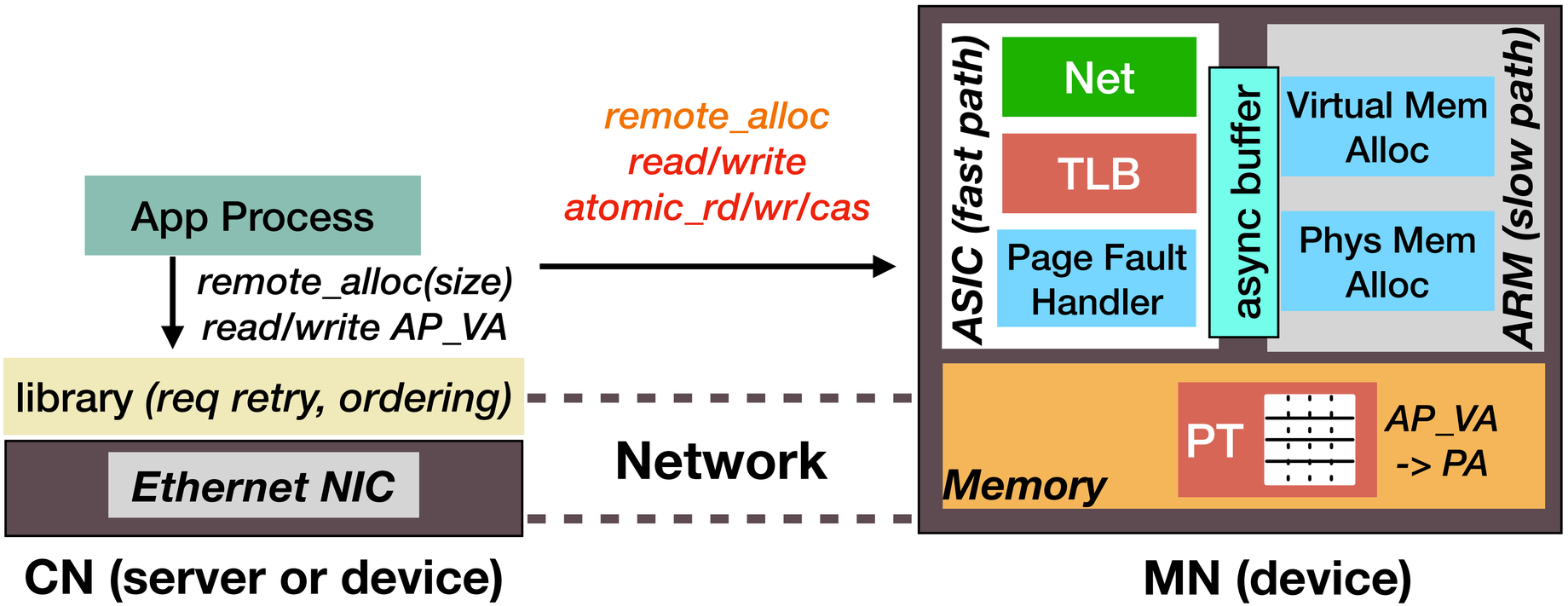
“eliminate state” summary

- Reduce state in system protocol: Disaggregated protocol, consistency model, …
- Move state to compute node: Congestion control, retry, dependency check, …
- Remove state from critical path: Hardware pagefault, memory region, …
- Optimize state to bounded size: Hash-based pagetable, atomic operations, …

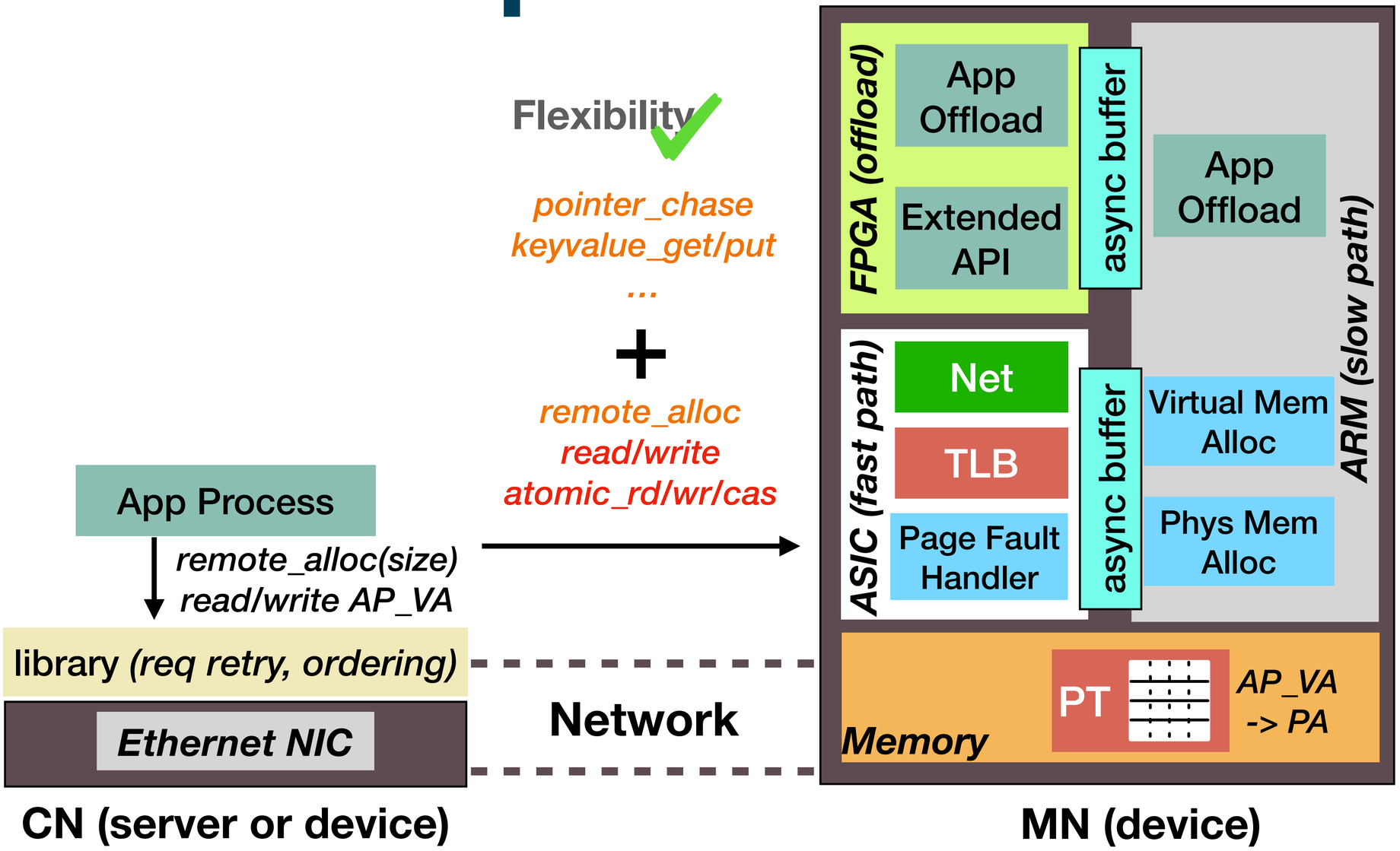
Low Performance Overhead

Throughput ✓ **Latency** ✓ **Tail Latency** ✓ **Low Cost** ✓ **Scalability** ✓ **Flexibility** ?

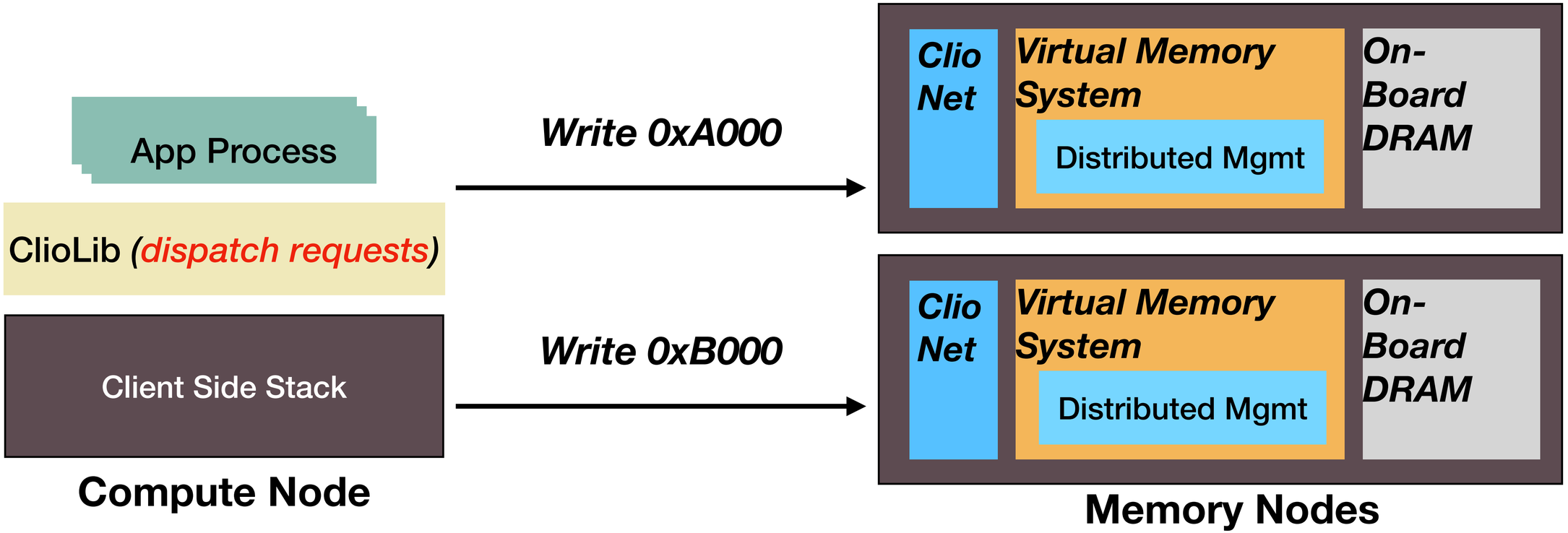
Extend computation offloading



Extend computation offloading



Distributed System Support



- **Multiple Clio boards can form a distributed system, single virtual memory space can span multiple memory nodes.**

Outline

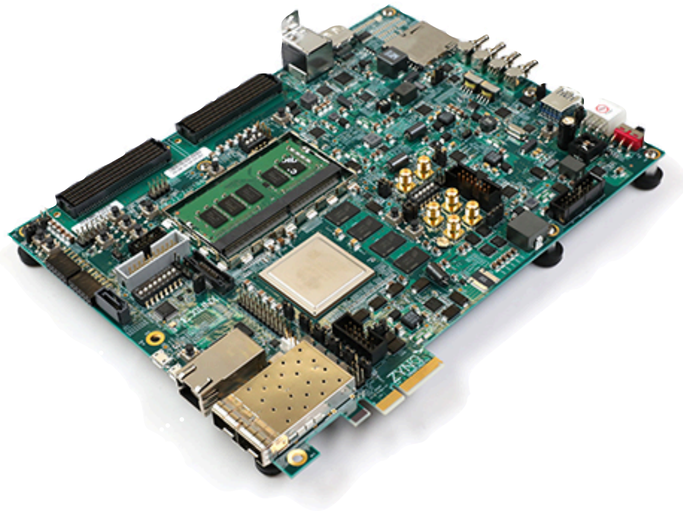
- Introduction
- Motivation: Why we need real hardware
- Clio Overview: Interface and overall approach
- Design: How we remove “state”
- **Implementation and evaluation results**

Implementation and Application

- Fast path and extended path implemented in hardware using SpinalHDL
- Prototype with Xilinx ZCU106 ARM-FPGA board

- Implemented five applications
  - Image compression
  - Multi-version object store
  - key-value store
  - pointer dereferencing
  - data analytics operation

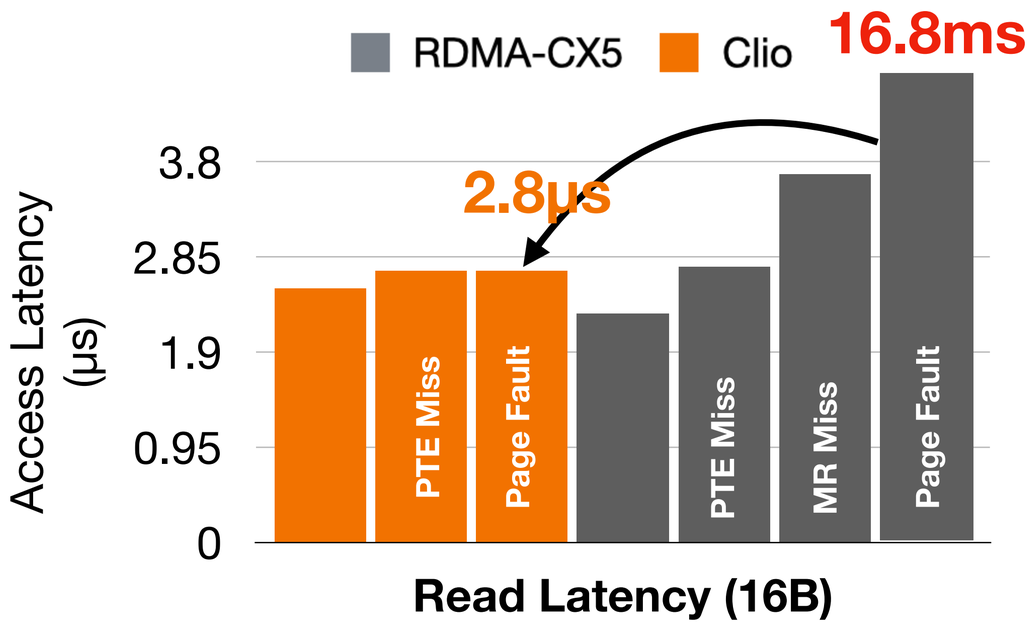
Clio prototype on the Xilinx ZCU106 board

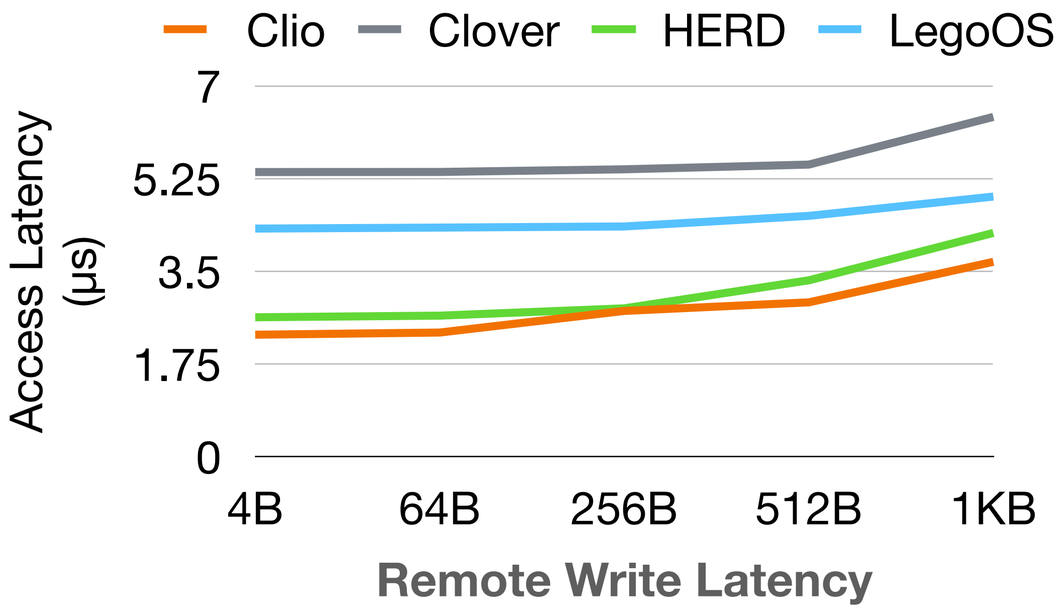


Evaluation Results

Basic Performance

- **100Gbps** throughput, **2.8µs (avg) 3.2µs (p99)** latency
- **Orders of magnitude lower tail latency than RDMA**
- **Outperforms Clover [ATC'20], LegoOS [OSDI'18], and HERD [SIGCOMM'14]**

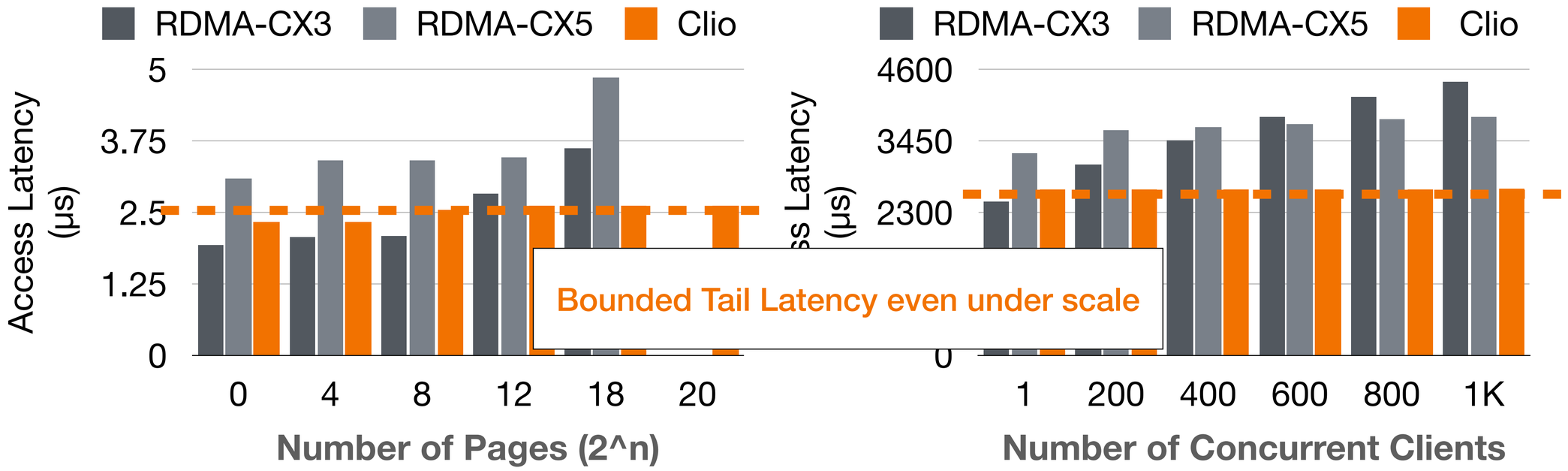




Evaluation Results

Concurrent Clients and Memory Size

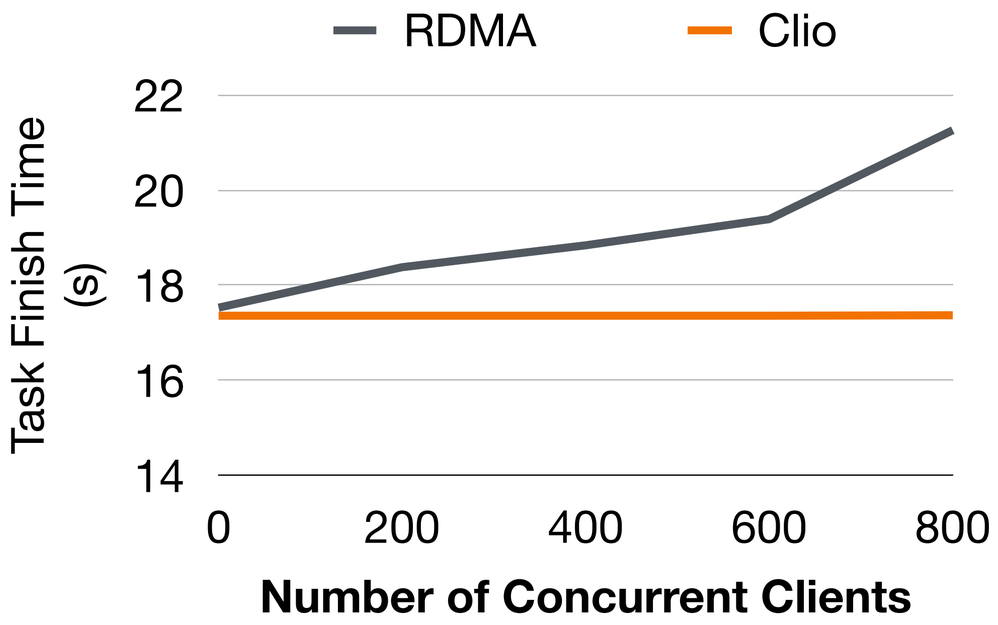
- **Clio provides bounded access time for data requests**
- **Clio scales well with concurrent clients and total memory size**

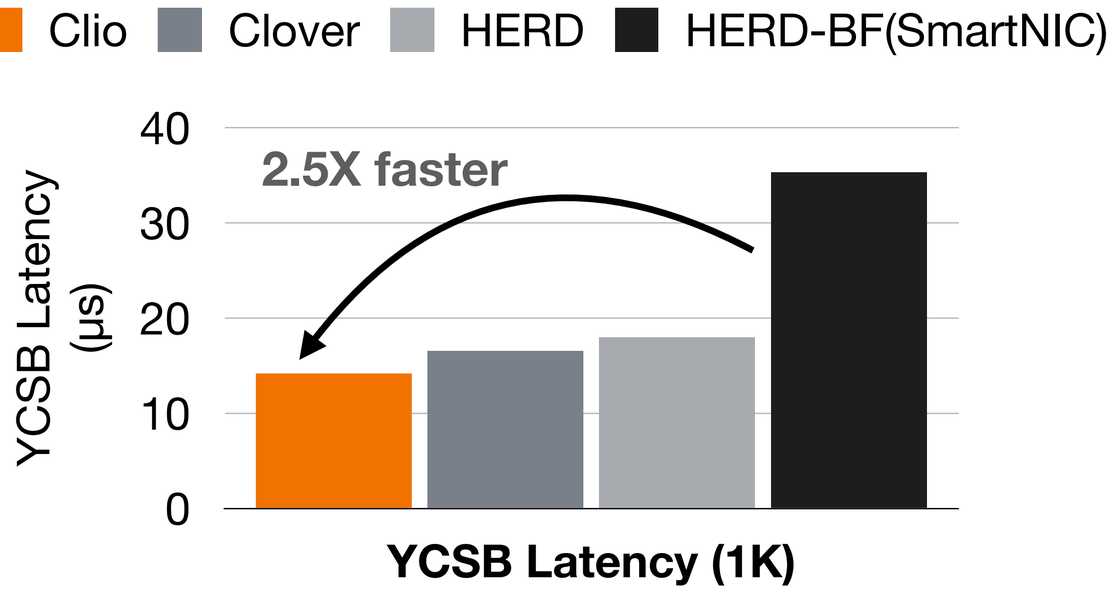


Evaluation Results

Disaggregated Applications

- **Applications benefits from stable latency and scalability**
- **Extended path outperforms CPU based offloading [Herd-BlueField]**





Summary

We built **Clio**, a real hardware disaggregated memory system

Achieves all requirements of memory disaggregation:
performance, cost, scalability and flexibility.

Conclusion

- Real benefits of hardware resource disaggregation comes from real hardware
- Building OS functionalities in hardware is feasible but needs new design
- The nature of disaggregation indicates new opportunities and challenges.
- Co-designing software and hardware systems is key in building real hardware.

Clio is a starting point for more real disaggregated hardware

Other Recent/Ongoing Disaggregation Works

- Network disaggregation (hardware implementation)
- Serverless computing on disaggregation
- Secure disaggregation (hardware implementation)
- …

Thank you!

Get Clio at https://github.com/WukLab/Clio

UCSDCSE
Computer Science and Engineering

wuklab.io