Pond: CXL-Based Memory Pooling Systems for Cloud Platforms

Huaicheng Li, *Daniel S. Berger, Lisa Hsu,* Daniel Ernst, *Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal,* Mark D. Hill, *Marcus Fontoura, Ricardo Bianchini*

ASPLOS'23

VIRGINIA TECH

Microsoft

UNIVERSITY of WASHINGTON

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

The Public Cloud Memory Setting

- Cloud providers targets
  - *Performance* comparable to on-premise datacenters
  - Competitive hardware *cost*

The Public Cloud Memory Setting

- Cloud providers targets
  - *Performance* comparable to on-premise datacenters
  - Competitive hardware ***cost***
- Memory allocation gold standard
  - VM memory is ***statically pinned*** to the *same NUMA node*

The Public Cloud Memory Setting

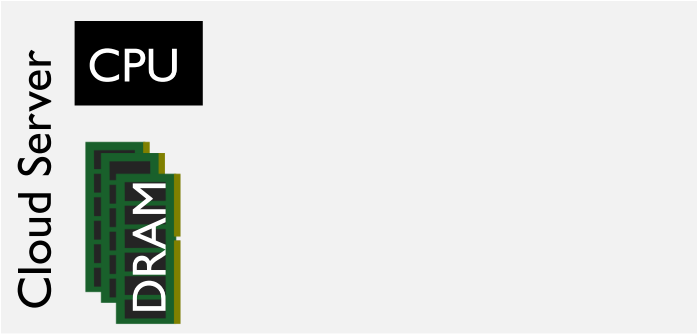
- Cloud providers targets
  - *Performance* comparable to on-premise datacenters
  - Competitive hardware ***cost***
- Memory allocation gold standard
  - VM memory is ***statically pinned*** to the *same NUMA node*
- DRAM is costly
  - ***~50%*** **of the server cost** for Azure!

Stranded and Untouched Cloud Memory

- ❑ Memory **stranding**
  - No free CPU cores but memory left

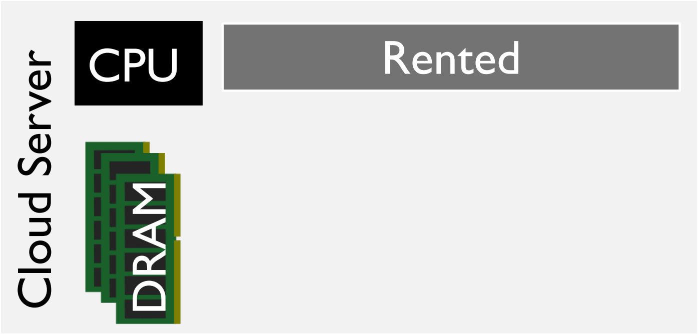
Stranded and Untouched Cloud Memory

- ❑ Memory **stranding**
  - ▪ No free CPU cores but memory left



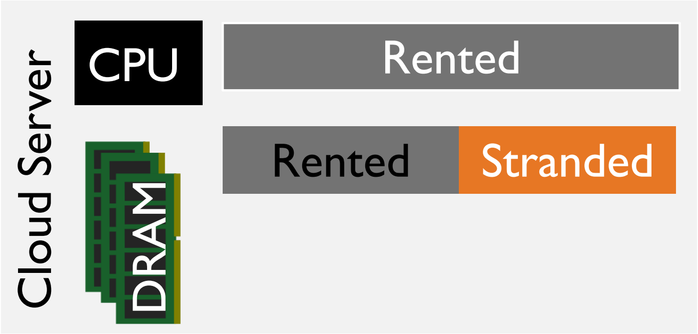
Stranded and Untouched Cloud Memory

- ❑ Memory **stranding**
  - No free CPU cores but memory left



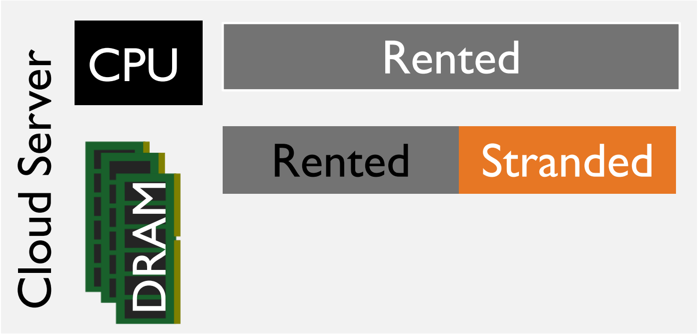
Stranded and Untouched Cloud Memory

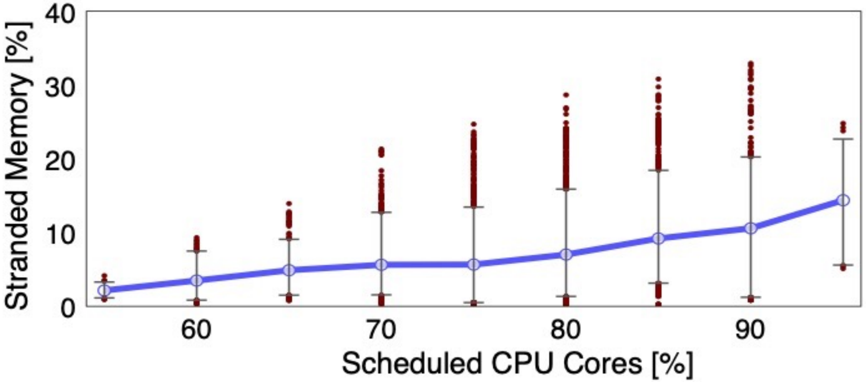
- **Memory stranding**
  - No free CPU cores but memory left



Stranded and Untouched Cloud Memory

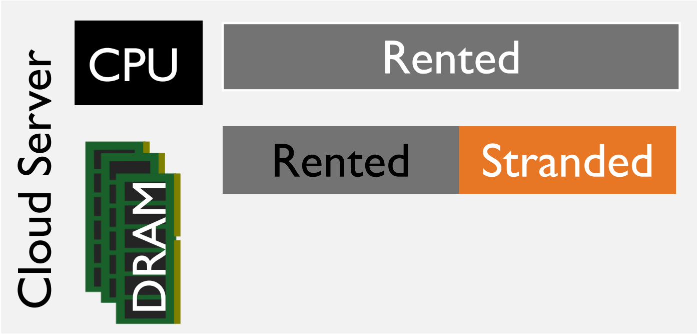
- Memory **stranding**
  - No free CPU cores but memory left

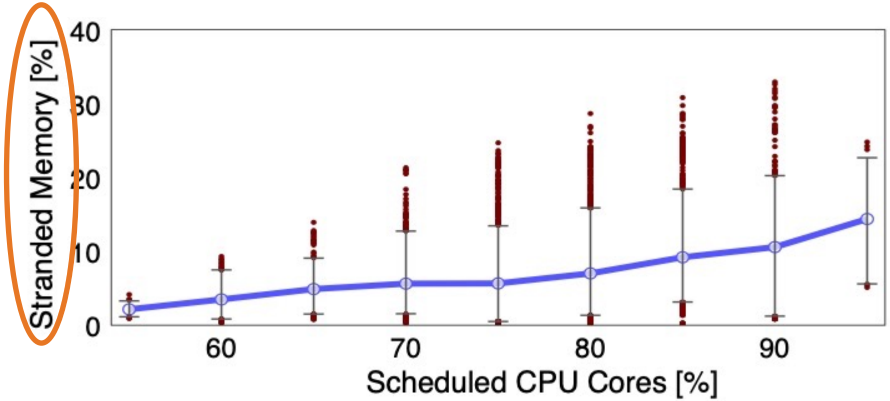




Stranded and Untouched Cloud Memory

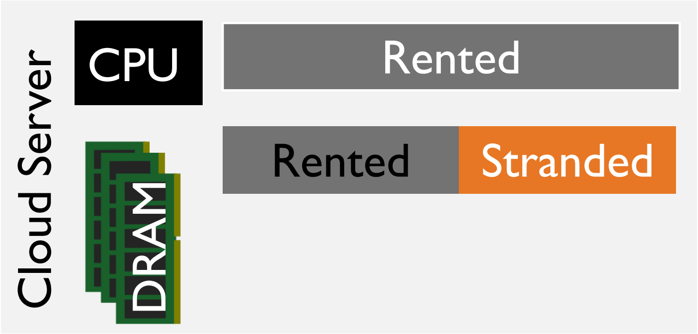
- Memory **stranding**
  - No free CPU cores but memory left

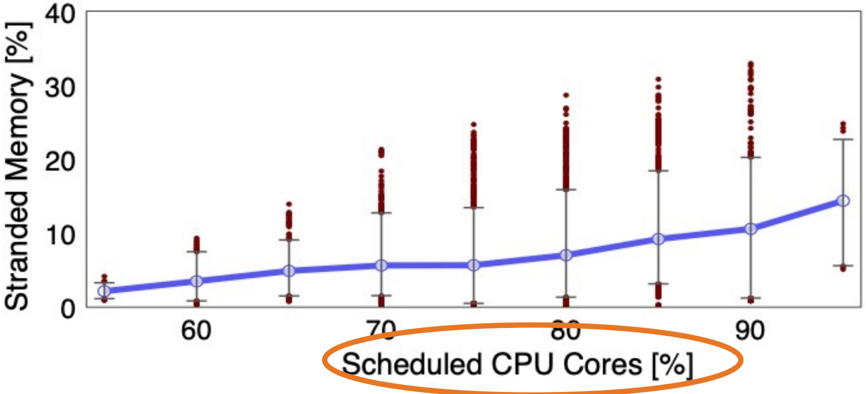




Stranded and Untouched Cloud Memory

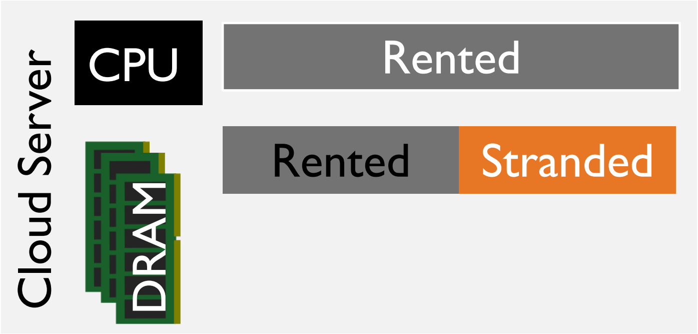
- Memory **stranding**
  - No free CPU cores but memory left

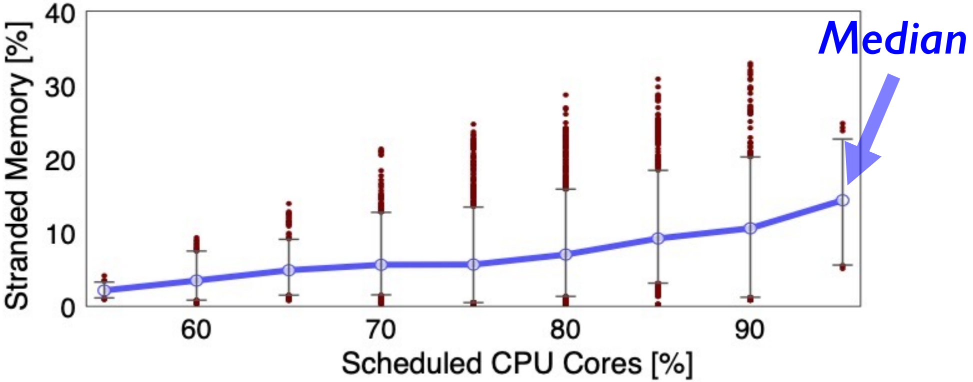




Stranded and Untouched Cloud Memory

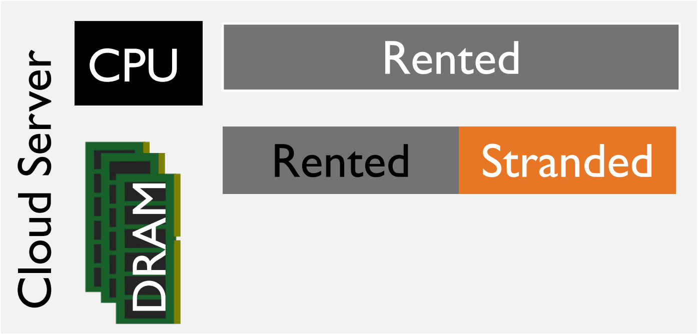
- Memory **stranding**
  - No free CPU cores but memory left

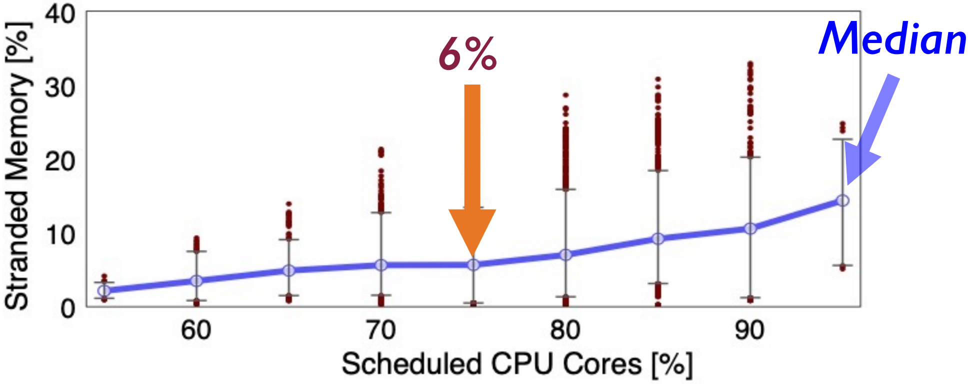




Stranded and Untouched Cloud Memory

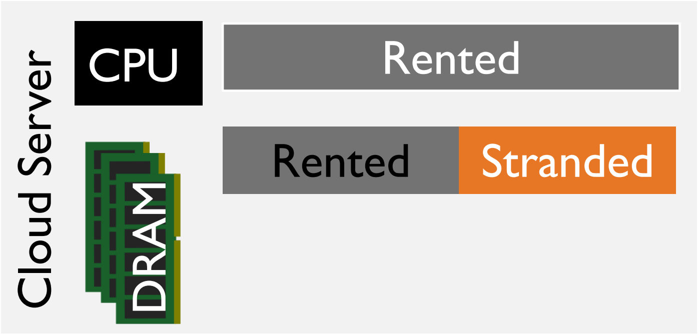
- Memory **stranding**
  - No free CPU cores but memory left

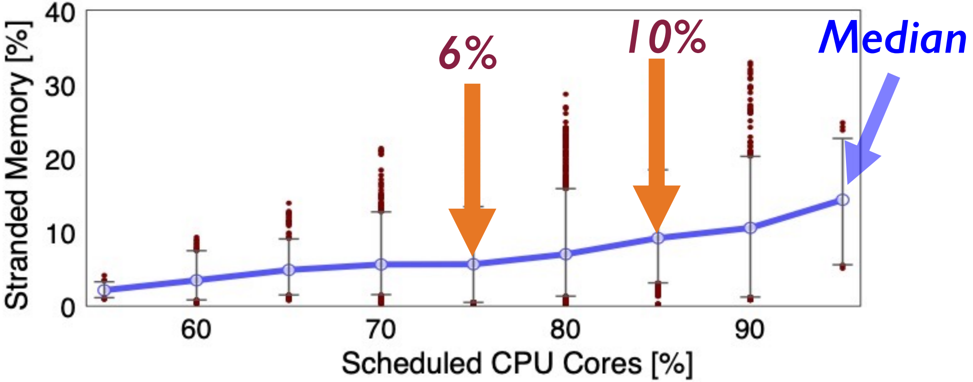




Stranded and Untouched Cloud Memory

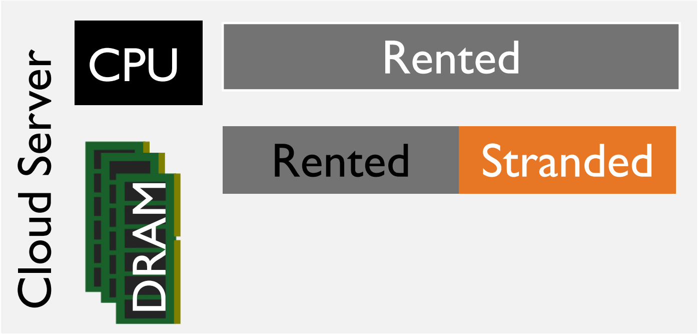
- ❑ Memory **stranding**
  - No free CPU cores but memory left

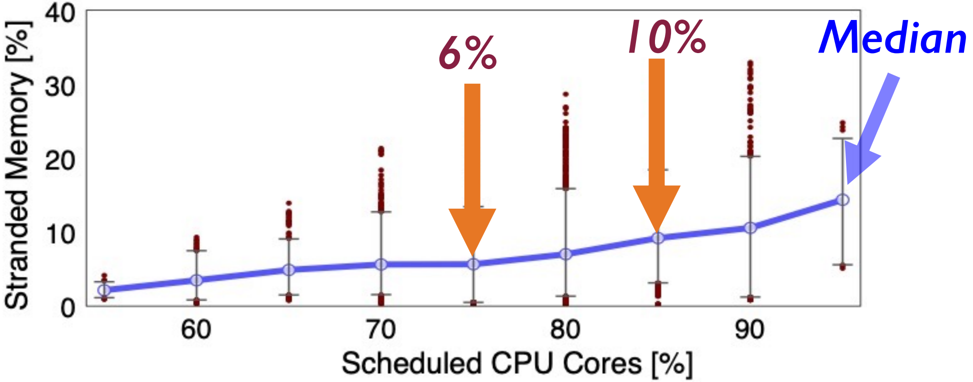




Stranded and Untouched Cloud Memory

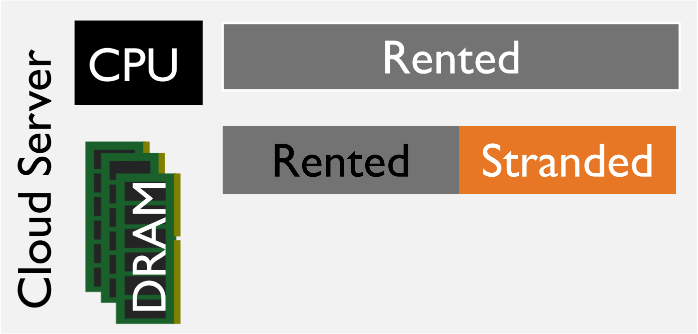
- **Memory stranding**
  - No free CPU cores but memory left
  - Up to 25% stranded memory at *95th percentile*

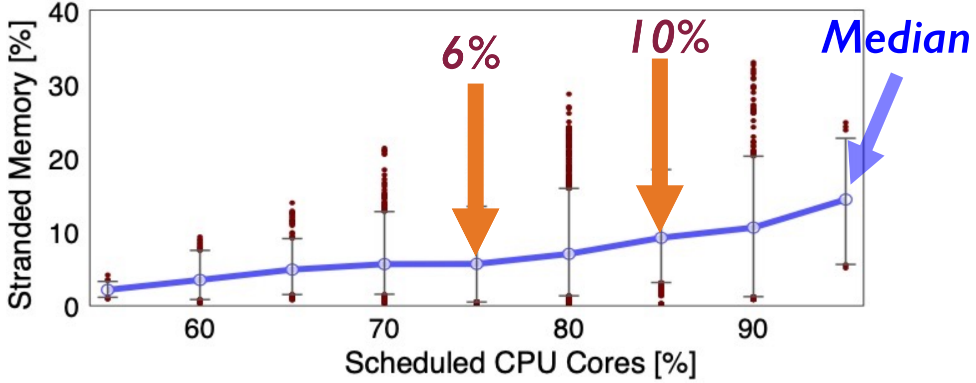




Stranded and Untouched Cloud Memory

- Memory **stranding**
  - No free CPU cores but memory left
  - Up to 25% stranded memory at *95th percentile*
- **Untouched** memory due to overprovisioning

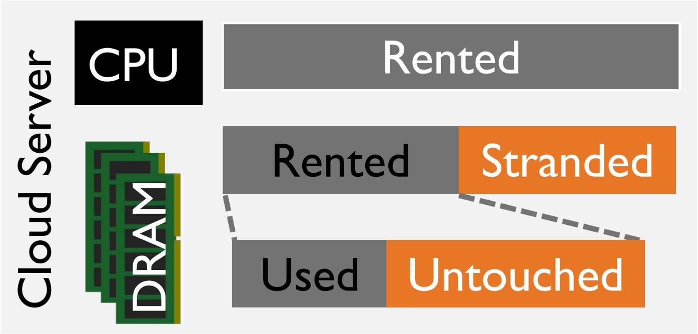


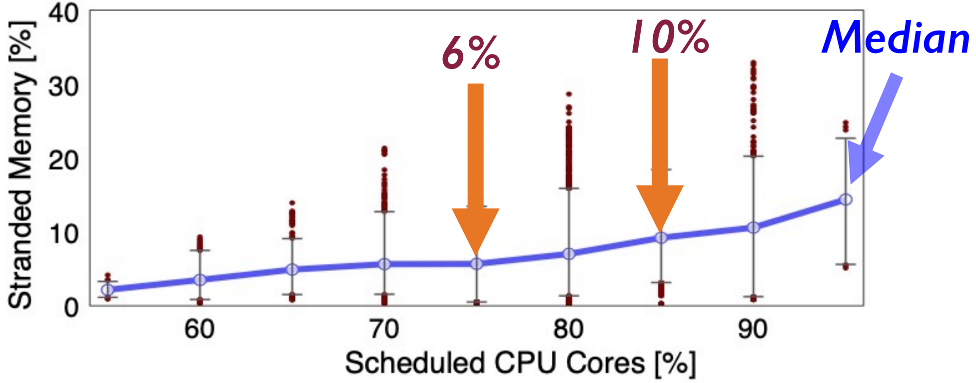


Stranded and Untouched Cloud Memory

- **Memory stranding**
  - No free CPU cores but memory left
  - Up to 25% stranded memory at *95th percentile*

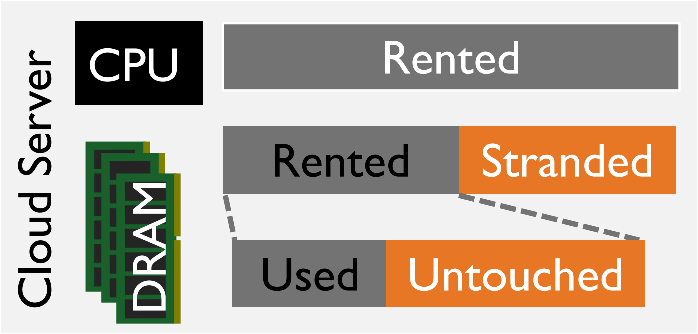
- **Untouched** memory due to overprovisioning

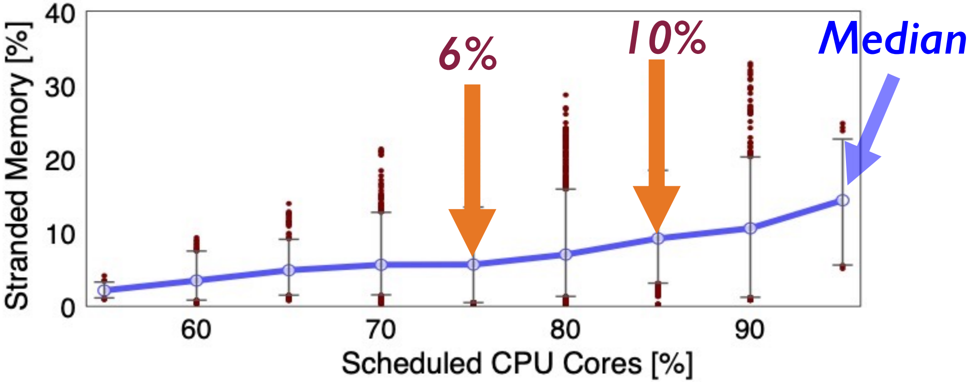




Stranded and Untouched Cloud Memory

- Memory **stranding**
  - No free CPU cores but memory left
  - Up to 25% stranded memory at *95th percentile*
- **Untouched** memory due to overprovisioning
  - 45% of untouched memory for half of the VMs



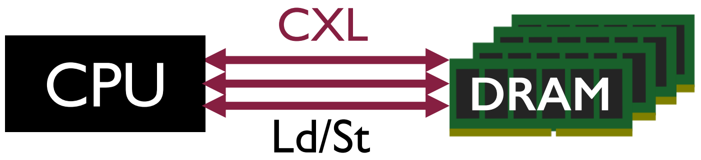


Pooling via CXL: Opportunities and Challenges

- CXL enables practical and performant pooling ☺

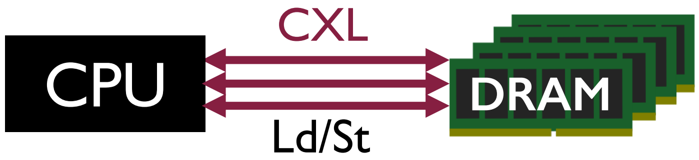
Pooling via CXL: Opportunities and Challenges

- CXL enables practical and performant pooling ☺
  - *Load/Store* access over PCIe 5.0 ("*CXL.mem*" protocol)
  - More practical than RDMA-based disaggregation designs



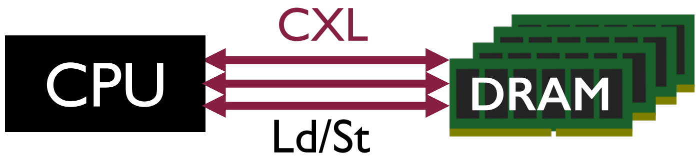
Pooling via CXL: Opportunities and Challenges

- CXL enables practical and performant pooling ☺
  - *Load/Store* access over PCIe 5.0 ("*CXL.mem*" protocol)
  - More practical than RDMA-based disaggregation designs
- CXL has *higher access latency than local DRAM* ☹



Pooling via CXL: Opportunities and Challenges

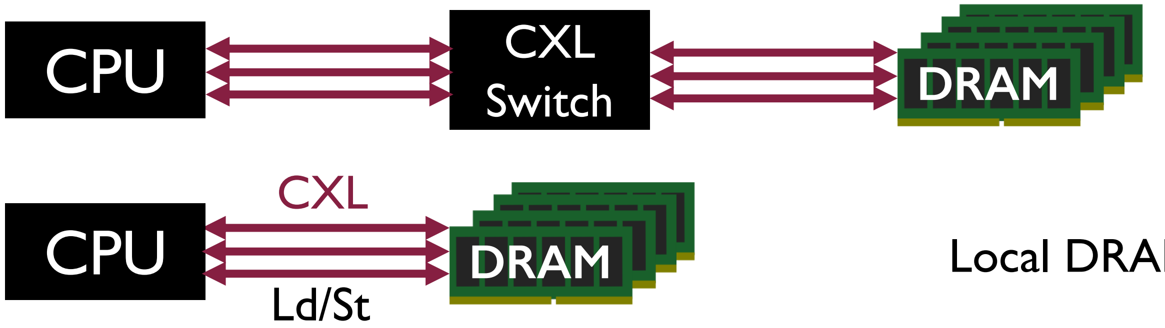
- CXL enables practical and performant pooling ☺
  - *Load/Store* access over PCIe 5.0 ("*CXL.mem*" protocol)
  - More practical than RDMA-based disaggregation designs
- CXL has ***higher access latency than local DRAM*** ☹
  - CPU-less node with **additional 70~90ns (~2x)**



Local DRAM (~90ns) + CXL (70~90ns)

Pooling via CXL: Opportunities and Challenges

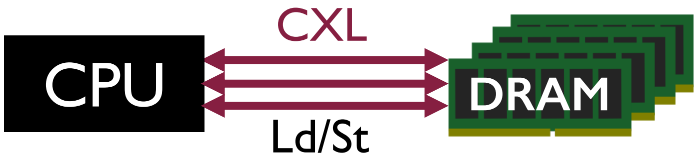
- CXL enables practical and performant pooling ☺
  - *Load/Store* access over PCIe 5.0 (“*CXL.mem*” protocol)
  - More practical than RDMA-based disaggregation designs
- CXL has *higher access latency than local DRAM* ☹
  - CPU-less node with **additional 70~90ns (~2x)**
  - CXL switches are slow and will add more latencies



Local DRAM (~90ns) + CXL (70~90ns)

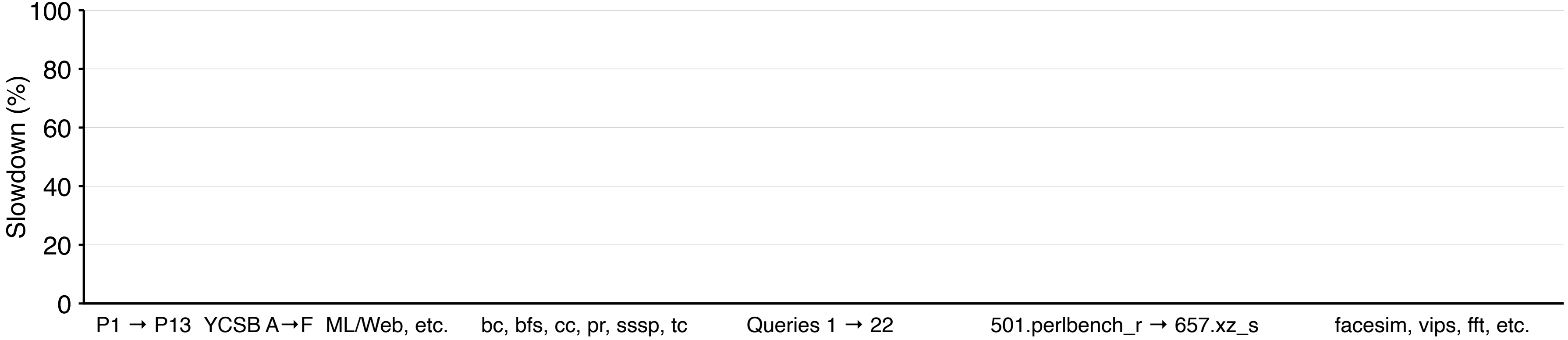
Pooling via CXL: Opportunities and Challenges

- CXL enables practical and performant pooling ☺
  - *Load/Store* access over PCIe 5.0 (“*CXL.mem*” protocol)
  - More practical than RDMA-based disaggregation designs
- CXL has *higher access latency than local DRAM* ☹
  - CPU-less node with **additional 70~90ns (~2x)**
  - CXL switches are slow and will add more latencies
  - ***Latency-sensitive workloads will suffer from CXL latencies***

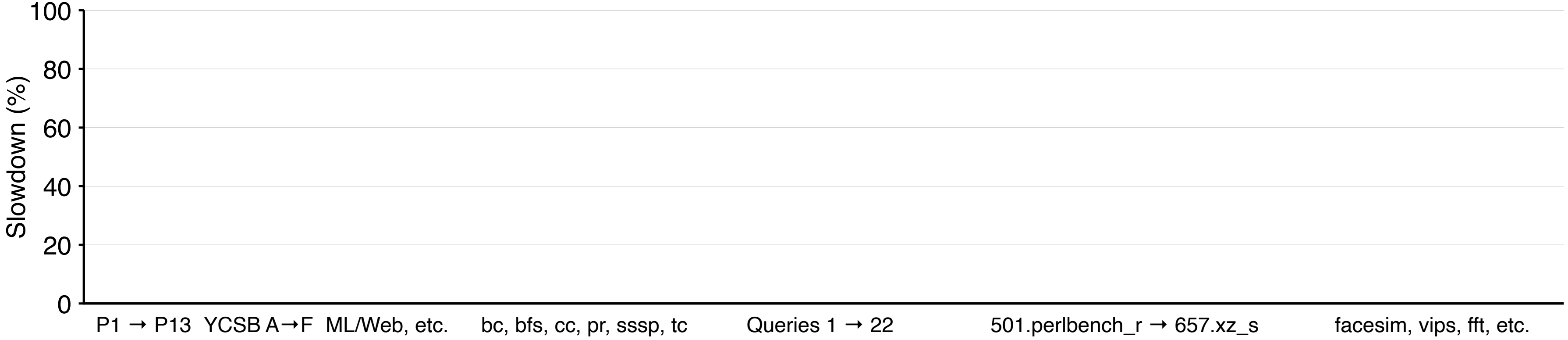


Local DRAM (~90ns) + CXL (70~90ns)

CXL Performance Impact



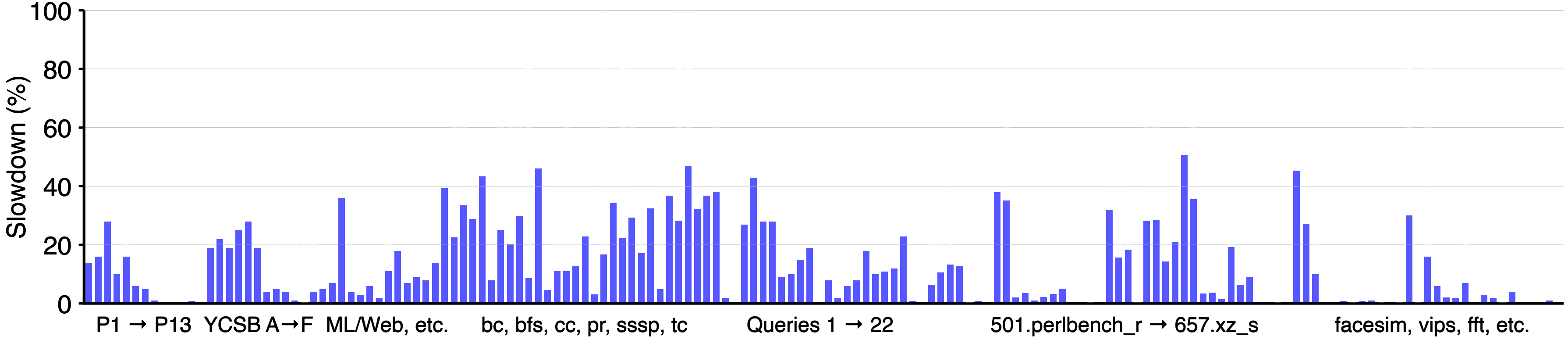
CXL Performance Impact



***158 workloads:** Proprietary, Redis, VoltDB, Spark, GAPBS, TPC-H, SPEC CPU 2017, etc.*

CXL Performance Impact

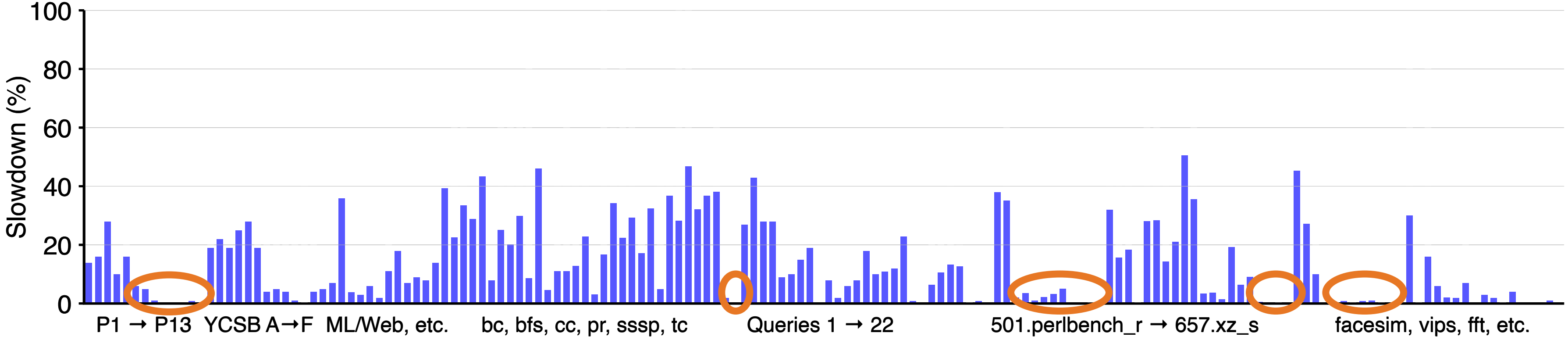
Approximated CXL latencies: 142ns (*182%*),



158 workloads: *Proprietary, Redis, VoltDB, Spark, GAPBS, TPC-H, SPEC CPU 2017, etc.*

CXL Performance Impact

Approximated CXL latencies: 142ns (182%),

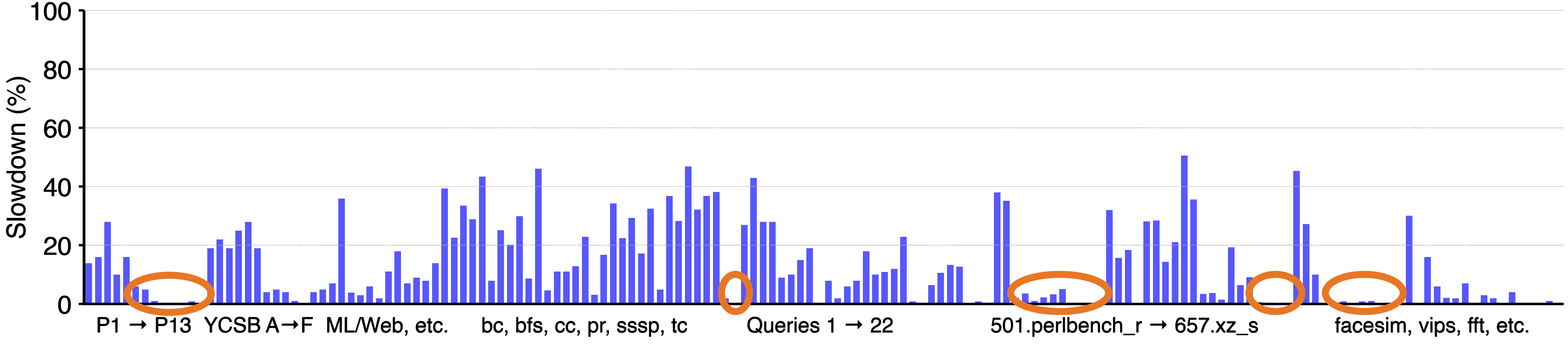


158 workloads: *Proprietary, Redis, VoltDB, Spark, GAPBS, TPC-H, SPEC CPU 2017, etc.*

(a) A small fraction of workloads are ***not*** sensitive to CXL latencies

CXL Performance Impact

Approximated CXL latencies: 142ns (182%),



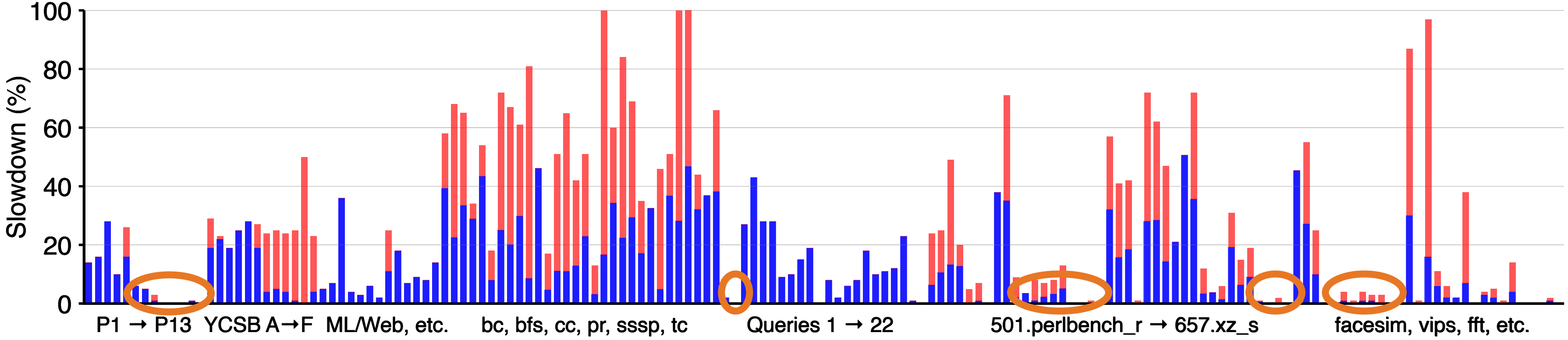
158 workloads: *Proprietary, Redis, VoltDB, Spark, GAPBS, TPC-H, SPEC CPU 2017, etc.*

(a) A small fraction of workloads are ***not*** sensitive to CXL latencies

(b) ~60% of the workloads see more than 5% slowdowns

CXL Performance Impact

Approximated CXL latencies: 142ns (182%), and 255ns (222%)



***158 workloads:** Proprietary, Redis, VoltDB, Spark, GAPBS, TPC-H, SPEC CPU 2017, etc.*

(a) A small fraction of workloads are ***not*** sensitive to CXL latencies

(b) ~60% of the workloads see more than 5% slowdowns

(c) Latency-sensitive workloads see ***bigger*** impact under higher CXL latencies

How to pool stranded and untouched memory via CXL for efficiency without sacrificing (much) performance at scale?

How to pool stranded and untouched memory via CXL for efficiency without sacrificing (much) performance at scale?

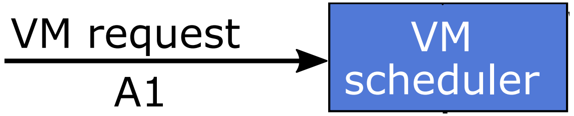
e.g., 95% of NUMA-local VM performance

How to pool stranded and untouched memory via CXL for efficiency without sacrificing (much) performance at scale?

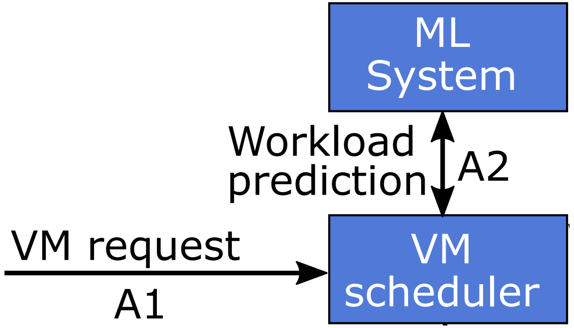
e.g., 95% of NUMA-local VM performance

Idea: *Predict the amount of VM memory that can be safely allocated from the pool while satisfying the performance requirement via QoS monitoring.*

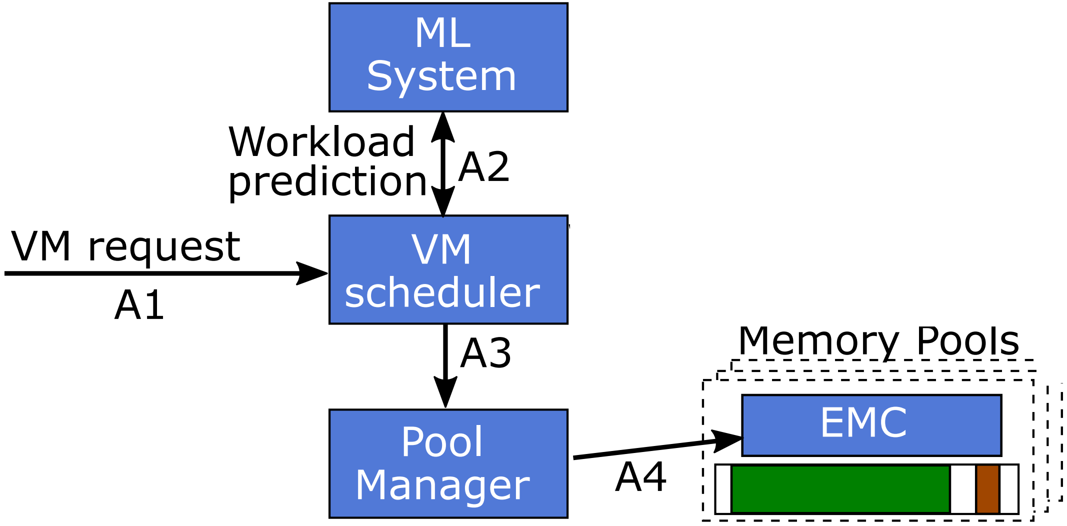
Pond: A Full-Stack Memory Pooling System



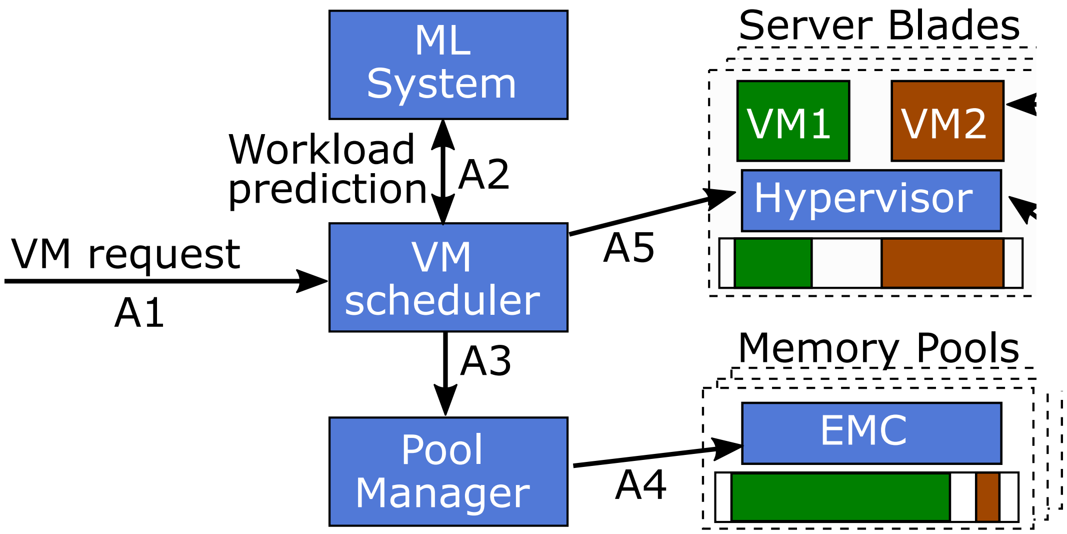
Pond: A Full-Stack Memory Pooling System



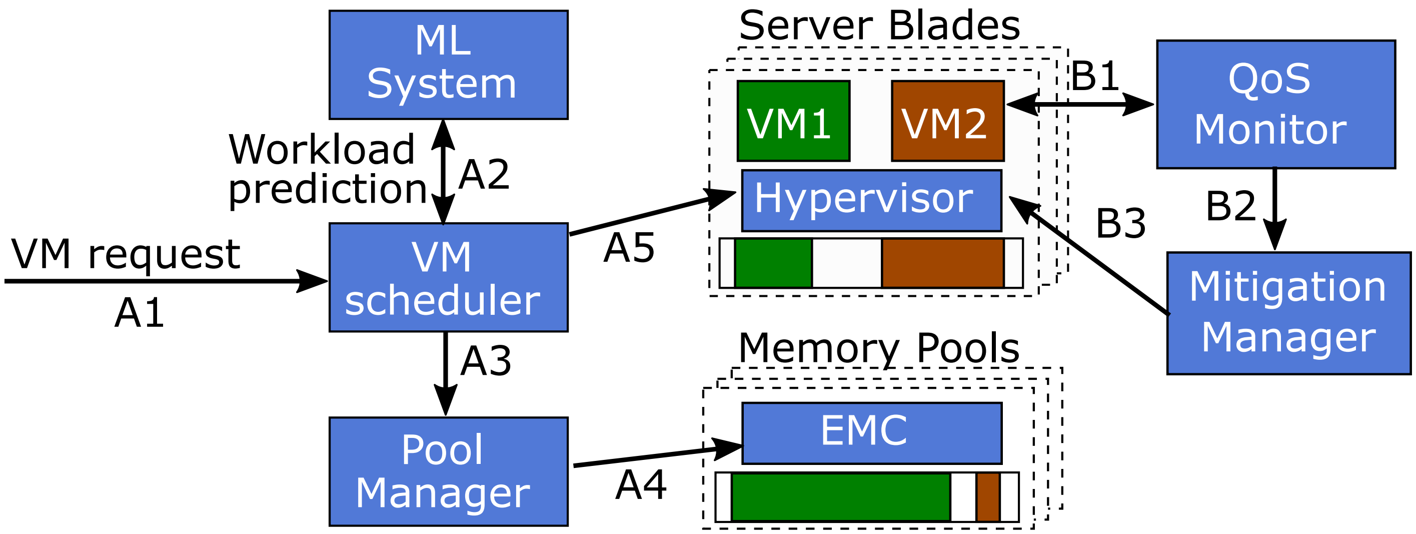
Pond: A Full-Stack Memory Pooling System



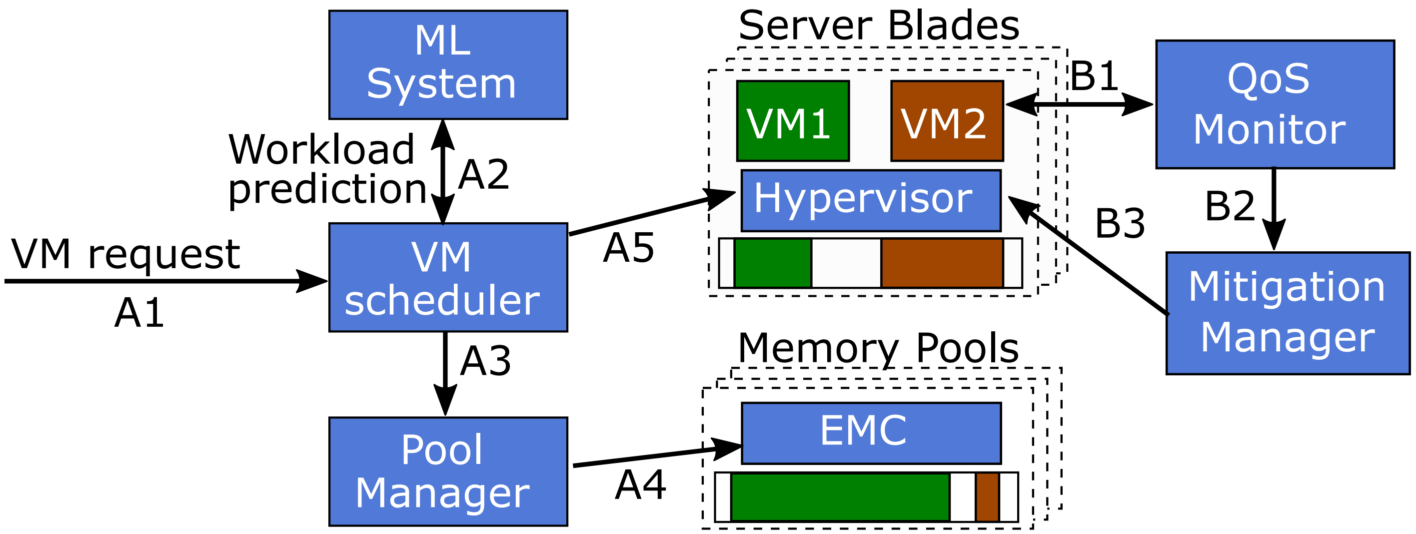
Pond: A Full-Stack Memory Pooling System



Pond: A Full-Stack Memory Pooling System



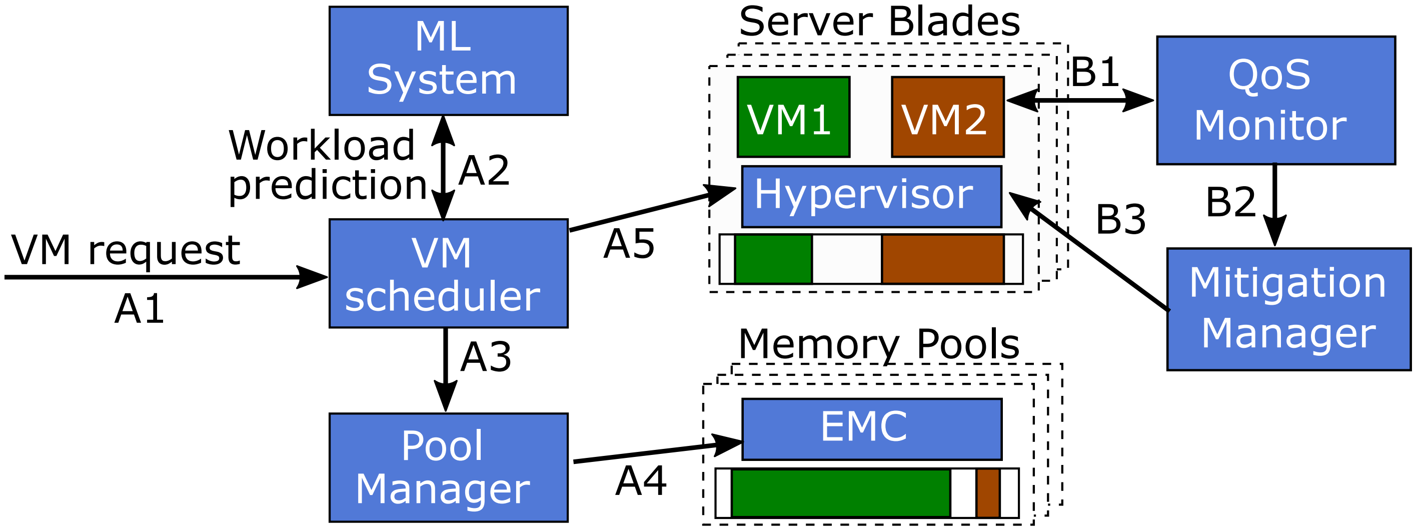
Pond: A Full-Stack Memory Pooling System



Pond contributions:

Hardware, system software, and distributed system layers to manage pooled memory

Pond: A Full-Stack Memory Pooling System

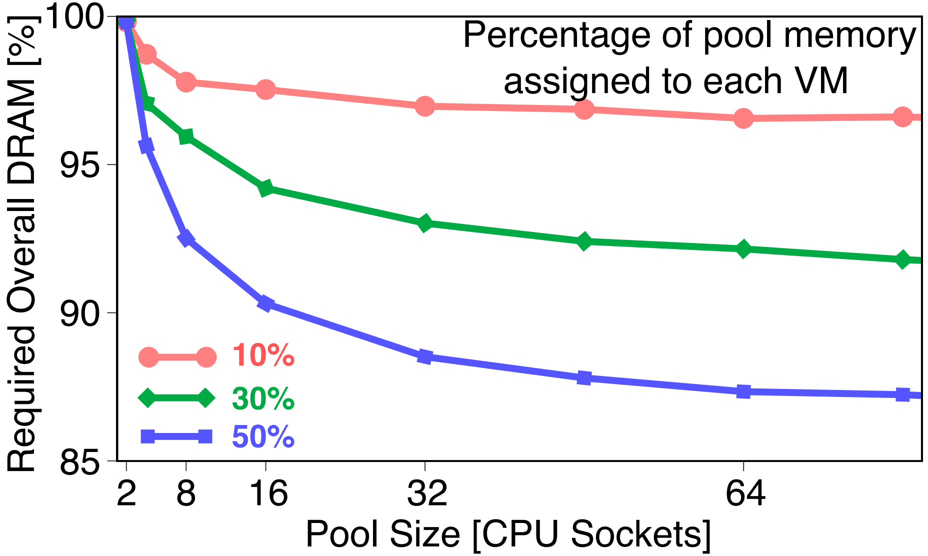


Pond contributions:
Hardware, system software, and distributed system layers to manage pooled memory

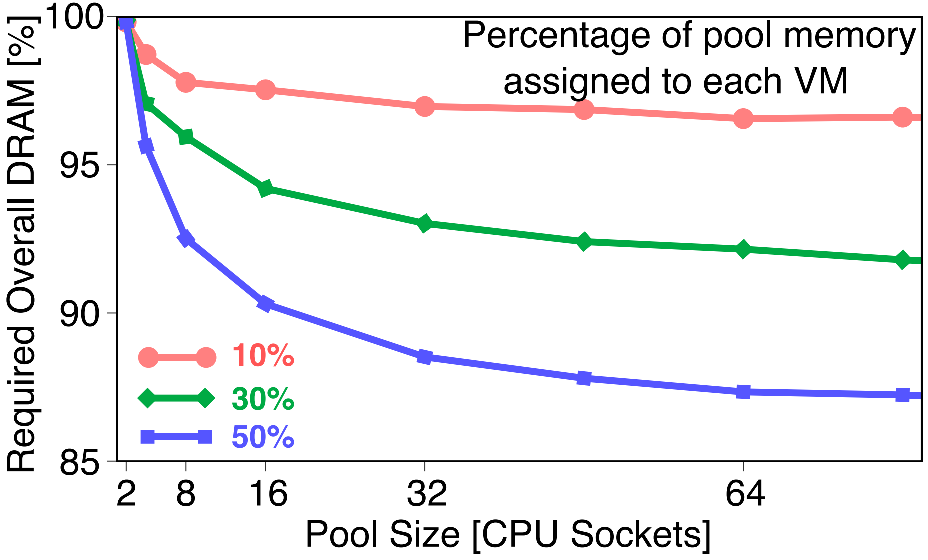
Pond benefits:
Reduce DRAM needs by 7% with a small pool → 3.5% reduction in cloud server cost

- Background & Motivation
- Pond Design
  - Overview
  - Memory pool scope
  - zNUMA
  - Prediction-assisted VM memory allocation
- Evaluation
- Conclusion

Pond Small Low Latency Pool



Pond Small Low Latency Pool

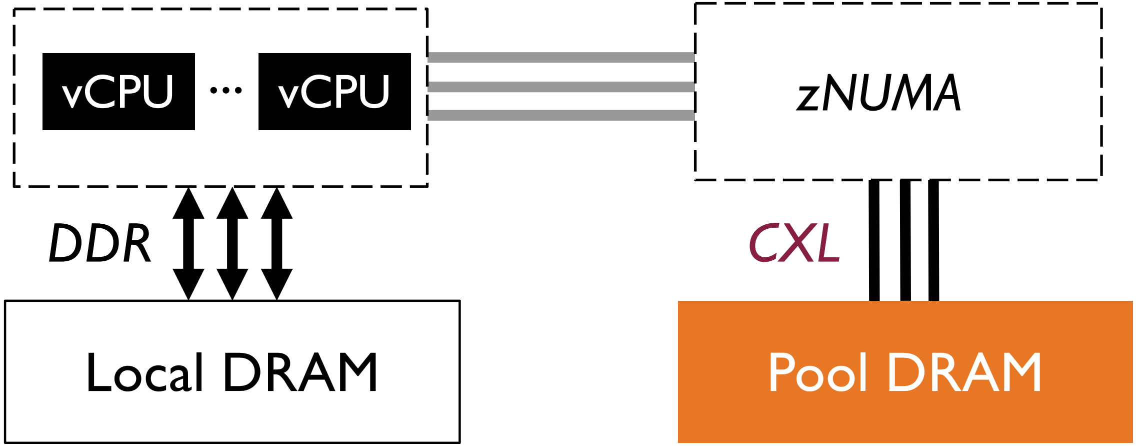


Small pools are effective!

While larger pools get diminishing returns.

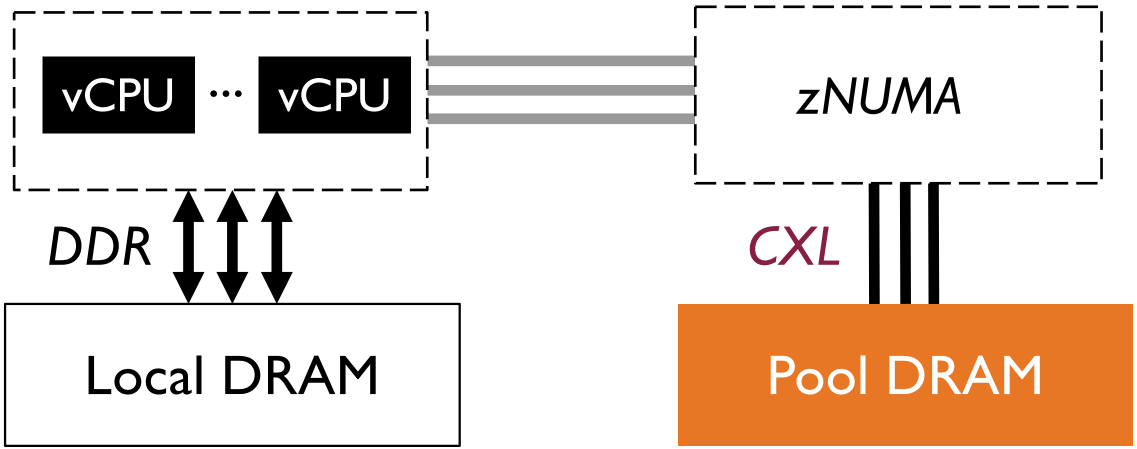
CXL Memory as a zNUMA Node to the VMs

- ❑ *Zero-core NUMA (zNUMA)*



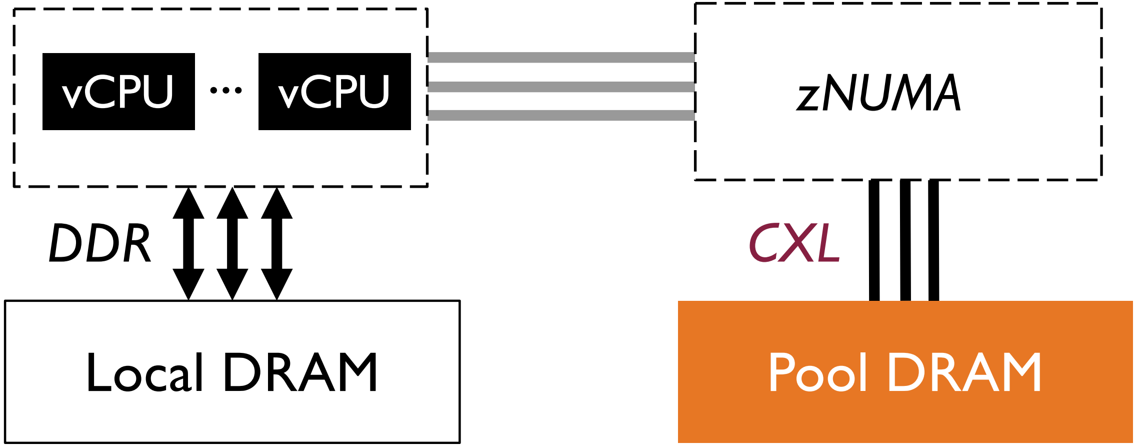
CXL Memory as a zNUMA Node to the VMs

- ❑ *Zero-core NUMA (zNUMA)*
- ❑ Funneling VM memory accesses by reusing existing OS memory management schemes (local-memory preference)



CXL Memory as a zNUMA Node to the VMs

- *Zero-core NUMA (zNUMA)*
- Funneling VM memory accesses by reusing existing OS memory management schemes (local-memory preference)
- No spilling under correct predictions



Pond Prediction-based VM Memory Provisioning

if (workload latency insensitive)

Entire pool/CXL DRAM

Pond Prediction-based VM Memory Provisioning

if (workload latency insensitive)

Entire pool/CXL DRAM

else if (no untouched memory)

Entire local DRAM

Pond Prediction-based VM Memory Provisioning

if (workload latency insensitive)

Entire pool/CXL DRAM

else if (no untouched memory)

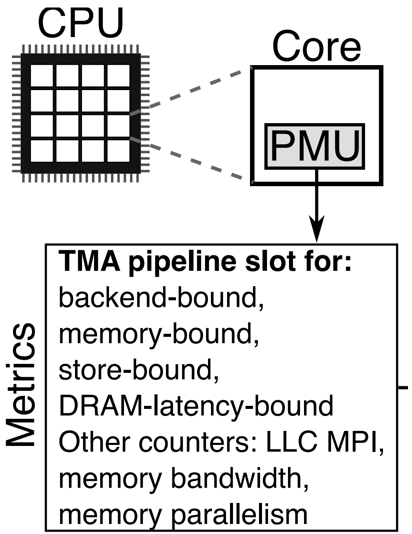
Entire local DRAM

else

zNUMA: Pool DRAM = Untouched

Latency Sensitivity Prediction

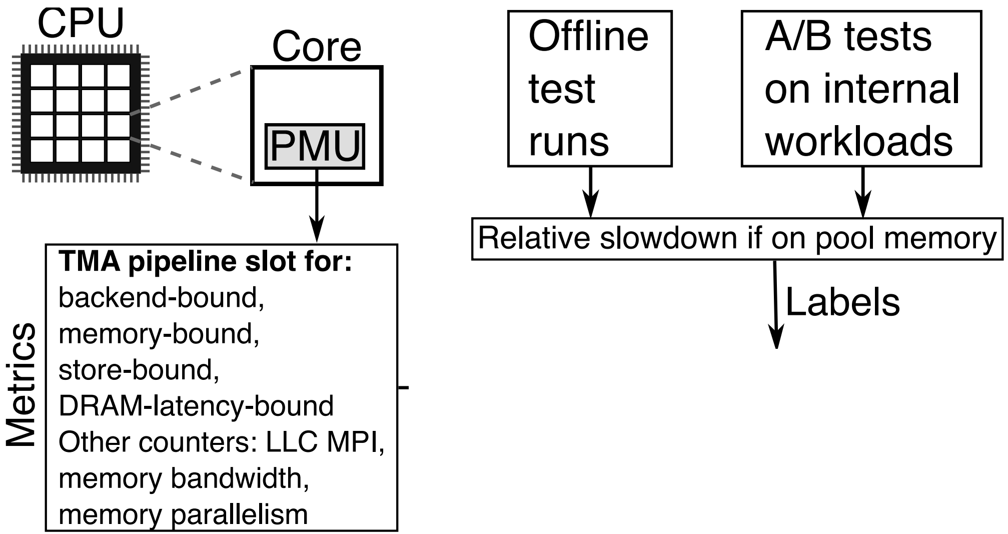
Features from opaque VMs
➔ existing HW counters



Latency Sensitivity Prediction

VM latency insensitive
➔ slowdown on CXL < 5%

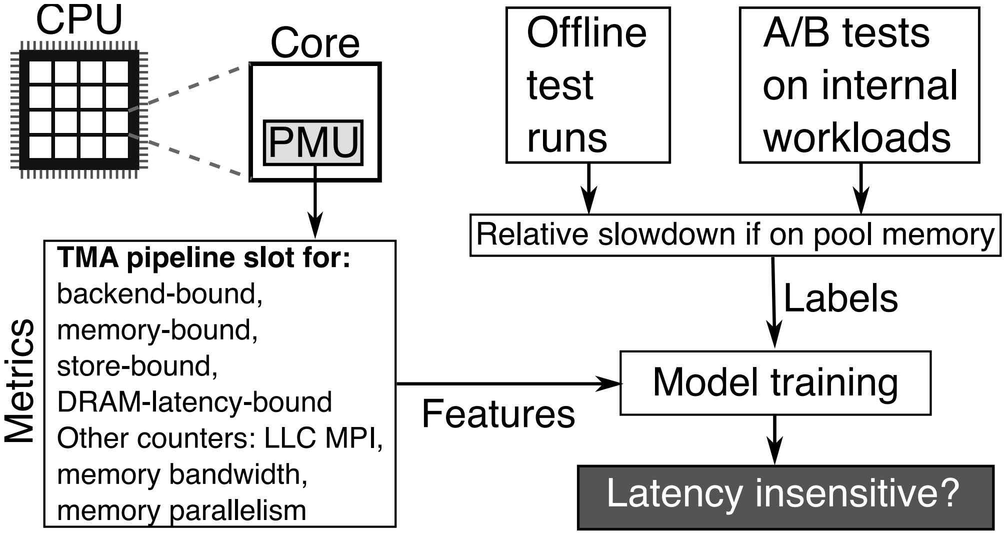
Features from opaque VMs
➔ existing HW counters



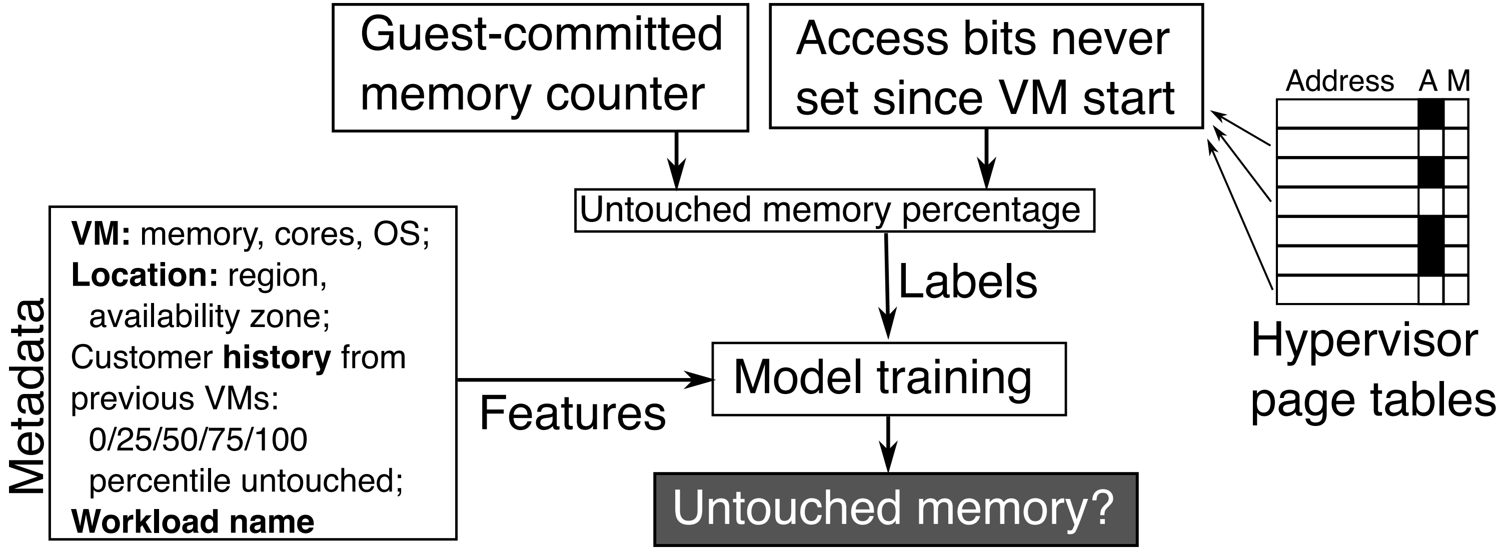
Latency Sensitivity Prediction

VM latency insensitive
➔ slowdown on CXL < 5%

Features from opaque VMs
➔ existing HW counters

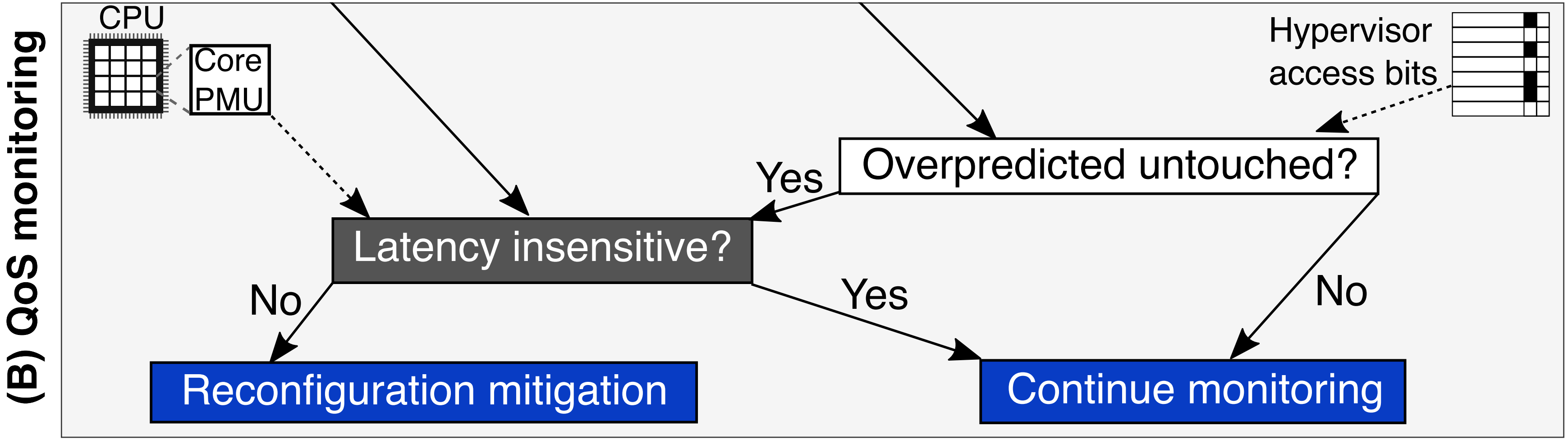


Untouched Memory Prediction



Prediction target: the amount of untouched memory (GB)

Misprediction Handling

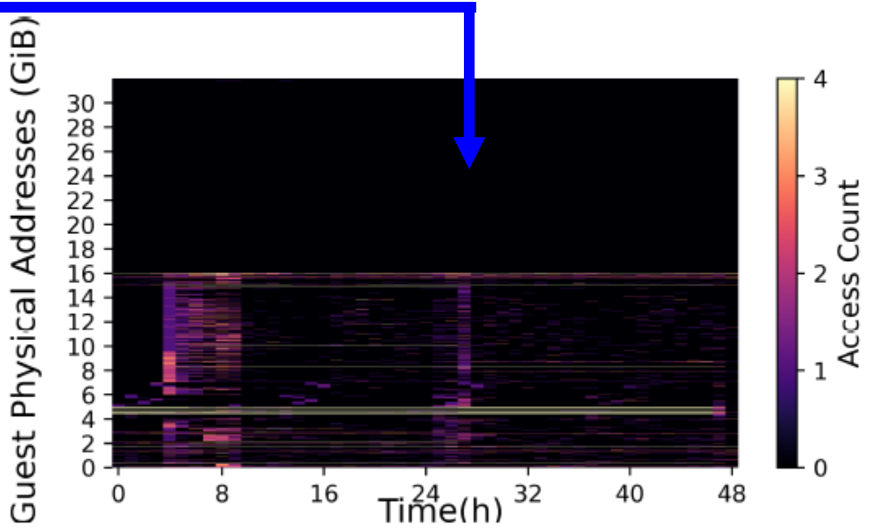


zNUMA Effectiveness

Workloads	zNUMA traffic
Video	0.25%
Database	0.06%
KV store	0.11%
Analytics	0.38%

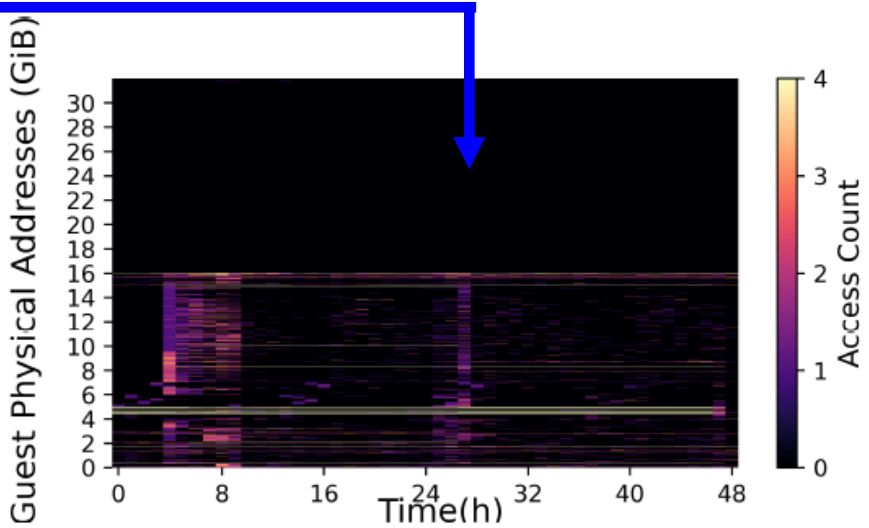
zNUMA Effectiveness

Workloads	zNUMA traffic
Video	0.25%
Database	0.06%
KV store	0.11%
Analytics	0.38%



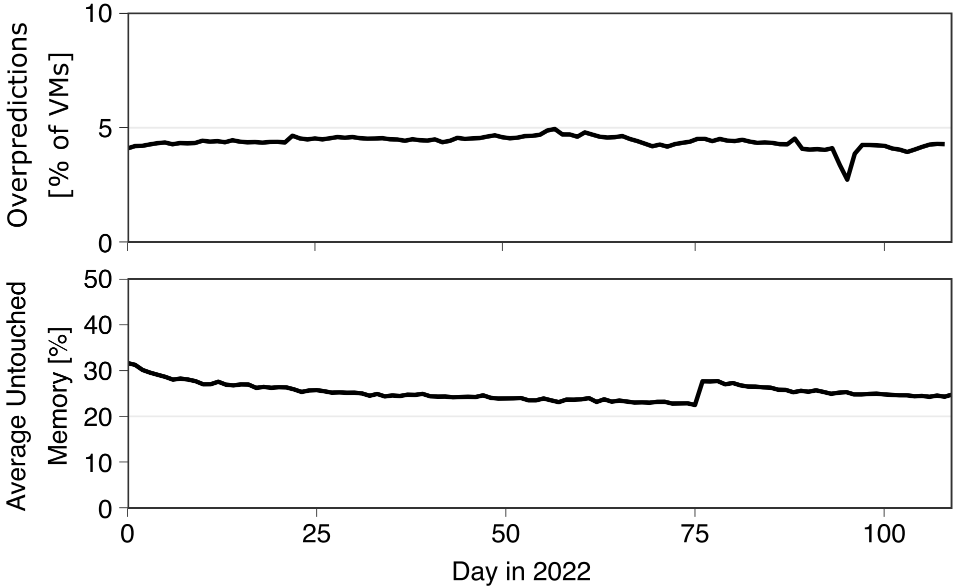
zNUMA Effectiveness

Workloads	zNUMA traffic
Video	0.25%
Database	0.06%
KV store	0.11%
Analytics	0.38%



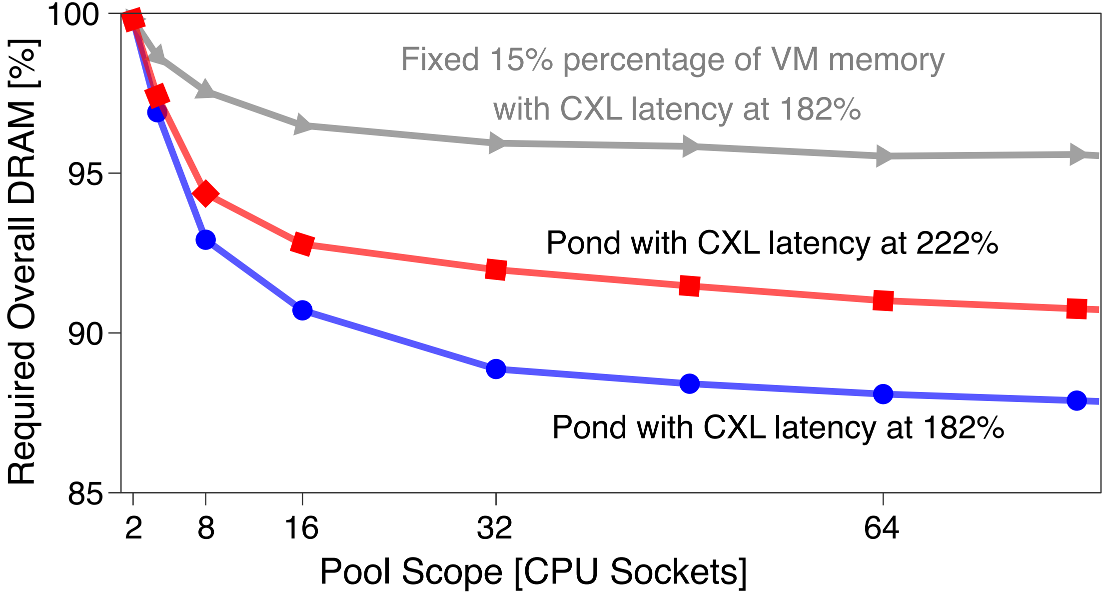
zNUMA is effective for correct predictions

Pond Prediction Model Accuracy



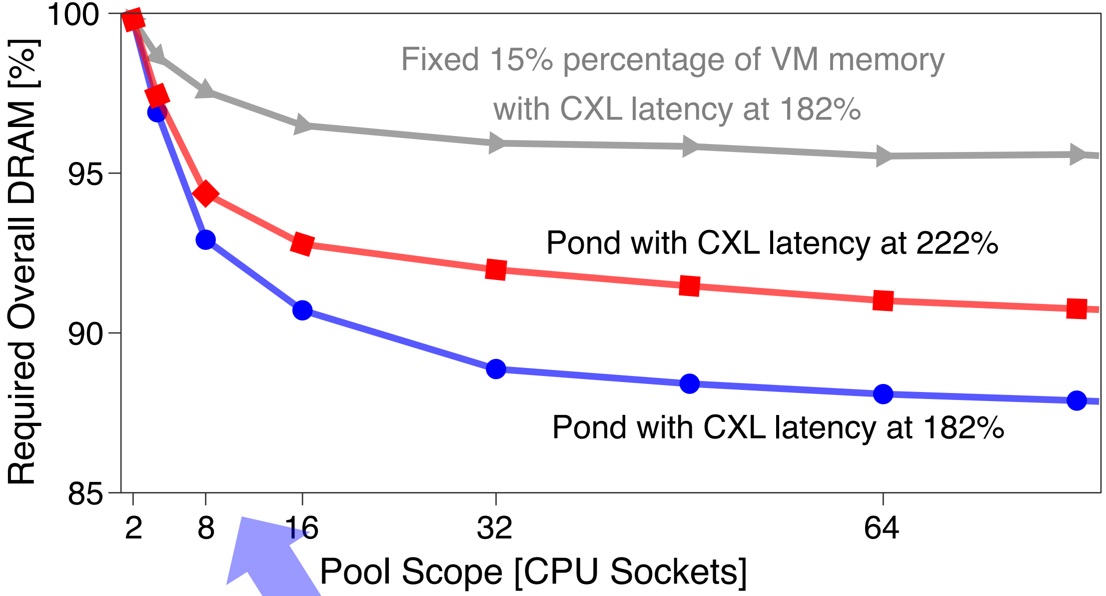
*Pond prediction model identifies **25% of** untouched memory while only overpredicting 4% of VMs*

Pond End-to-end Memory Savings



Configured to target <5% slowdown for 98% of VMs

Pond End-to-end Memory Savings



8-16 socket pool, 7-9% DRAM savings

Configured to target <5% slowdown for 98% of VMs

More Details in the Paper

- Detailed trace study results and analysis over 100 production clusters
- EMC and pool memory management
- Details of the prediction models
- More evaluation results
- ...

More Details in the Paper

- ❑ Detailed trace study results and analysis over 100 production clusters
- ❑ EMC and pool memory management
- ❑ Details of the prediction models
- ❑ More evaluation results
- ❑ …

Pond: CXL-Based Memory Pooling Systems for Cloud Platforms

Huaicheng Li
Virginia Tech
Carnegie Mellon University
USA

Daniel S. Berger
Microsoft Azure
University of Washington
USA

Lisa Hsu
Unaffiliated
USA

Daniel Ernst
Microsoft Azure
USA

Pantea Zardoshti
Microsoft Azure
USA

Stanko Novakovic
Google
USA

Monish Shah
Microsoft Azure
USA

Samir Rajadnya
Microsoft Azure
USA

Scott Lee
Microsoft
USA

Ishwar Agarwal
Intel
USA

Mark D. Hill
Microsoft Azure
University of Wisconsin-Madison
USA

Marcus Fontoura
Stone Co
USA

Ricardo Bianchini
Microsoft Azure
USA

ABSTRACT

Public cloud providers seek to meet stringent performance requirements and low hardware cost. A key driver of performance and cost is main memory. Memory pooling promises to improve DRAM utilization and thereby reduce costs. However, pooling is challenging under cloud performance requirements. This paper proposes Pond, the first memory pooling system that both meets cloud performance goals and significantly reduces DRAM cost. Pond builds on the Compute Express Link (CXL) standard for load/store access to pool memory and two key insights. First, our analysis of cloud production traces shows that pooling across 8-16 sockets is enough to achieve most of the benefits. This enables a small-pool design with low access latency. Second, it is possible to create machine learning models that can accurately predict how much local and pool memory to allocate to a virtual machine (VM) to resemble same-NUMA-node memory performance. Our evaluation with 158 workloads shows that Pond reduces DRAM costs by 7% with performance within 1-5% of same-NUMA-node VM allocations.

CCS CONCEPTS

• **Computer systems organization** → **Cloud computing**; • **Hardware** → **Emerging architectures**.

ASPLOS '23, March 25–29, 2023, Vancouver, BC, Canada

ACM ISBN 978-1-4503-9916-6/23/03.
https://doi.org/10.1145/3575693.3578835

KEYWORDS

Compute Express Link; CXL; memory disaggregation; memory pooling; datacenter; cloud computing.

ACM Reference Format:
Huaicheng Li, Daniel S. Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo Bianchini. 2023. Pond: CXL-Based Memory Pooling Systems for Cloud Platforms. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '23), March 25–29, 2023, Vancouver, BC, Canada.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3575693.3578835

1 INTRODUCTION

Motivation. Many public cloud customers deploy their workloads in the form of virtual machines (VMs), for which they get virtualized compute with performance approaching that of a dedicated cloud, but without having to manage their own on-premises datacenter. This creates a major challenge for public cloud providers: achieving excellent performance for opaque VMs (*i.e.*, providers do not know and should not inspect what is running inside the VMs) at a competitive hardware cost.

A key driver of both performance and cost is main memory. The gold standard for memory performance is for accesses to be served by the same NUMA node as the cores that issue them, leading to latencies in tens of nanoseconds. A common approach is to preallocate all VM memory on the same NUMA node as the VM's cores. Preallocating and statically pinning memory also facilitate the use of virtualization accelerators [4], which are enabled by default, for example, on AWS and Azure [12, 14]. At the same time, DRAM has become a major portion of hardware cost due to its poor scaling properties with only nascent alternatives [72]. For example,

Pond Summary

Applicable for locally attached CXL

Feasible hardware implementation

Close to local DRAM performance

7-9% DRAM savings by pooling
~25% untouched memory

Pond Summary

Applicable for locally attached CXL

Feasible hardware implementation

Close to local DRAM performance

7-9% DRAM savings by pooling
~25% untouched memory

Pond CXL emulation tool: [*https://github.com/vtess/pond*](https://github.com/vtess/pond)

Pond Summary

Applicable for locally attached CXL

Feasible hardware implementation

Close to local DRAM performance

7-9% DRAM savings by pooling
~25% untouched memory

Pond CXL emulation tool: https://github.com/vtess/pond

Thank you!