# Break on Through to The Other Side: Pooling Memory Elastically with RamRyder

Yanbo Zhou, Erci Xu†, Dongjoo Seo*, Adam Manzanares*, Steven Swanson

UC San Diego
†Shanghai Jiao Tong University
*Samsung Semiconductor

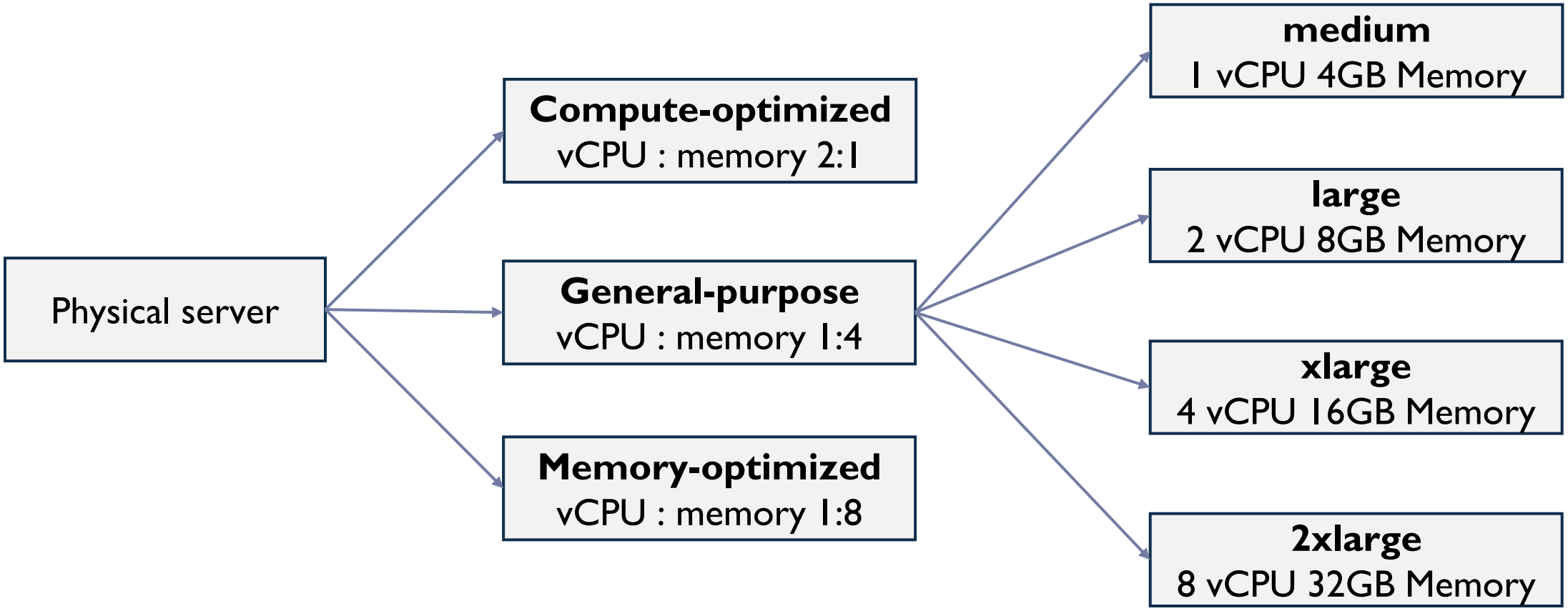
**OSDI 2026**



# The State of Memory Allocation in the Cloud

## The memory allocation in the cloud is less flexible

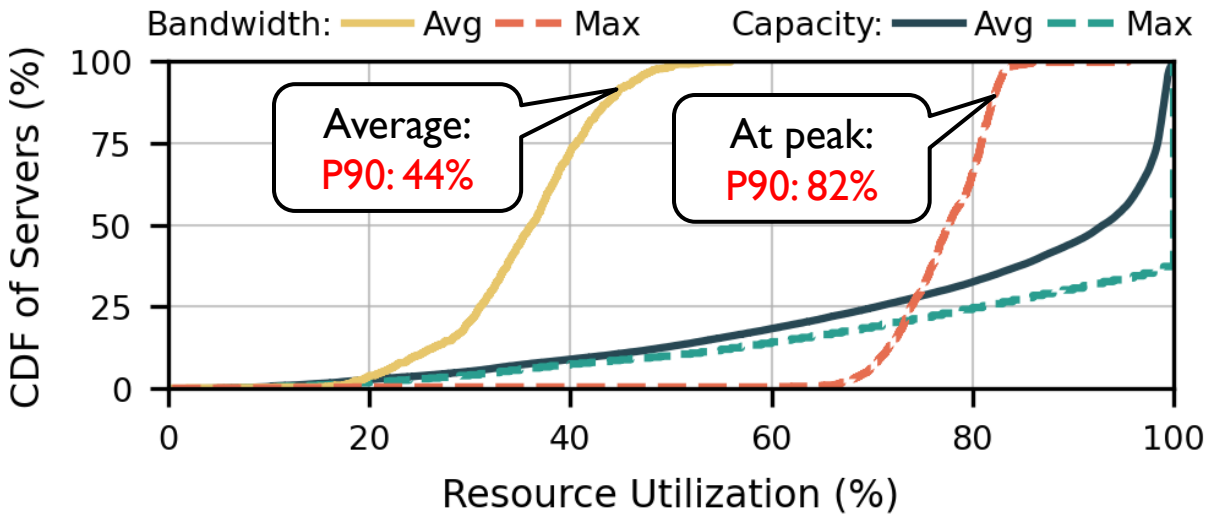
- Define physical servers to provide multiple CPU-to-memory ratios (**types**)
- Split physical servers into multiple CPU and memory units as VMs (**sizes**)



# Bandwidth is Underutilized yet Understudied

## Observation 1:

## Severely stranded bandwidth



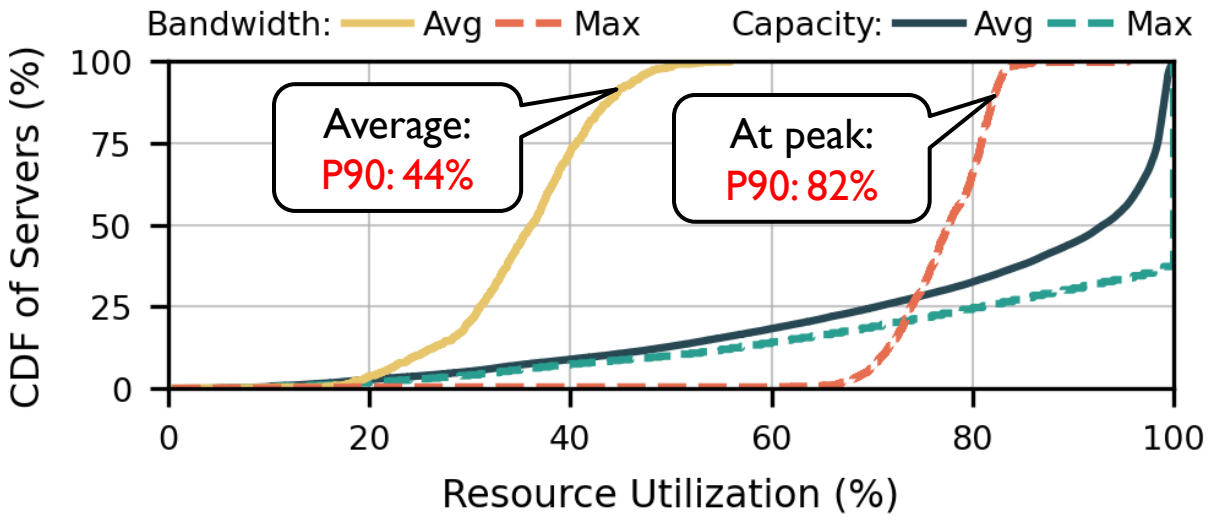

Optimizing memory resource efficiency should consider both capacity and bandwidth

**Data are from real cluster traces of cloud vendors*

# Bandwidth is Underutilized yet Understudied

## Observation 1:

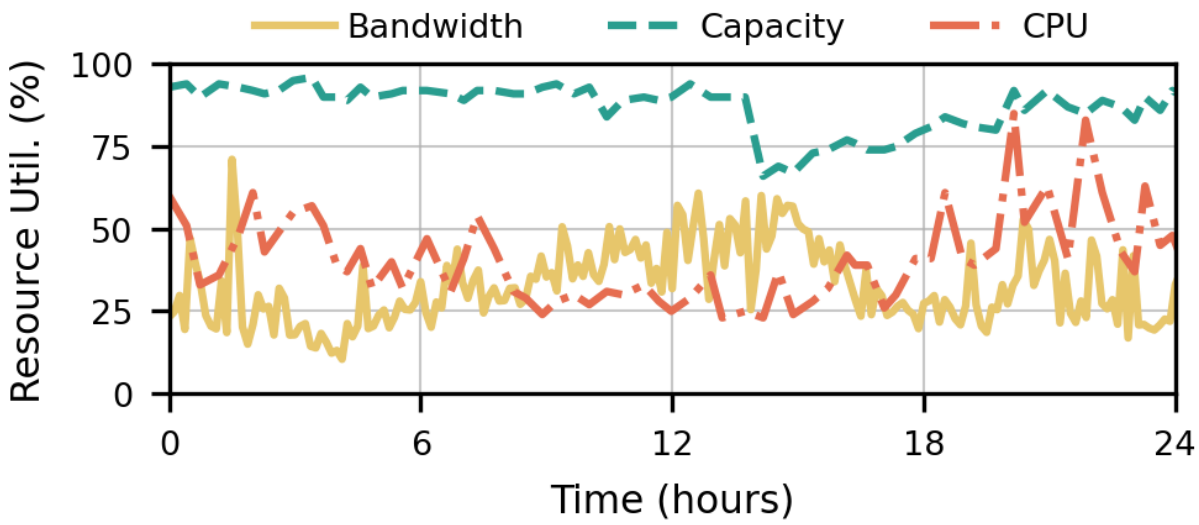
### Severely stranded bandwidth




Optimizing memory resource efficiency should consider both capacity and bandwidth

## Observation 2:

### Decoupled resource utilization

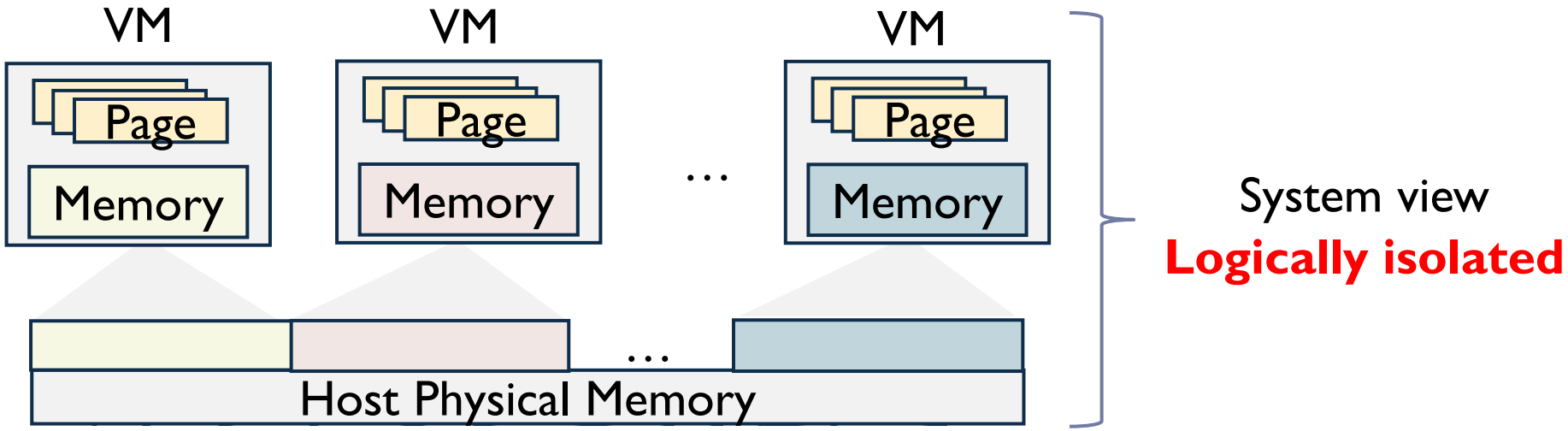



Independently managing capacity and bandwidth can potentially improve overall memory resource utilization

**Data are from real cluster traces of cloud vendors*

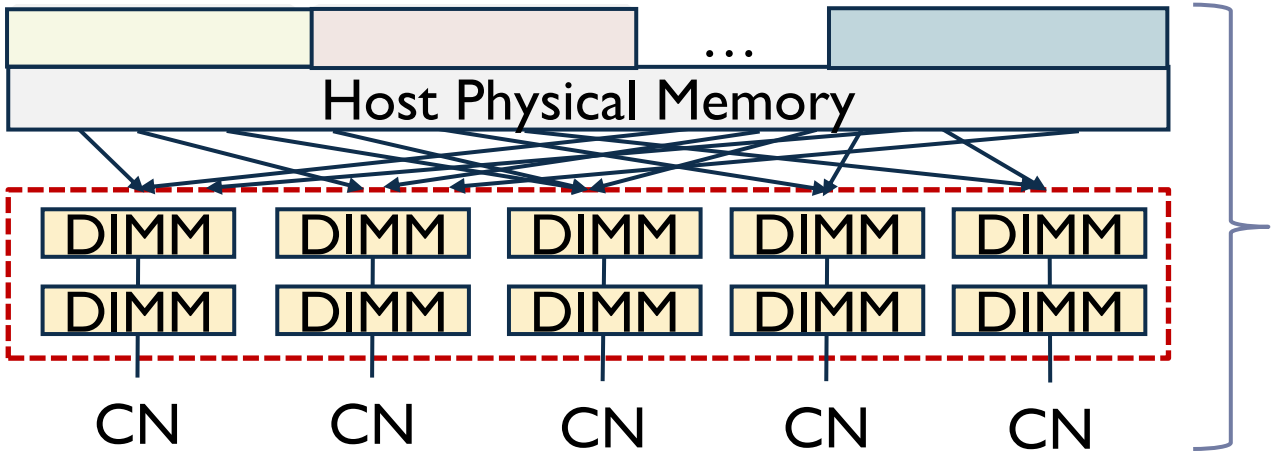
# The Root Causes of Memory Underutilization

## 1. Spatial over-provisioning



# The Root Causes of Memory Underutilization

## 1. Spatial over-provisioning



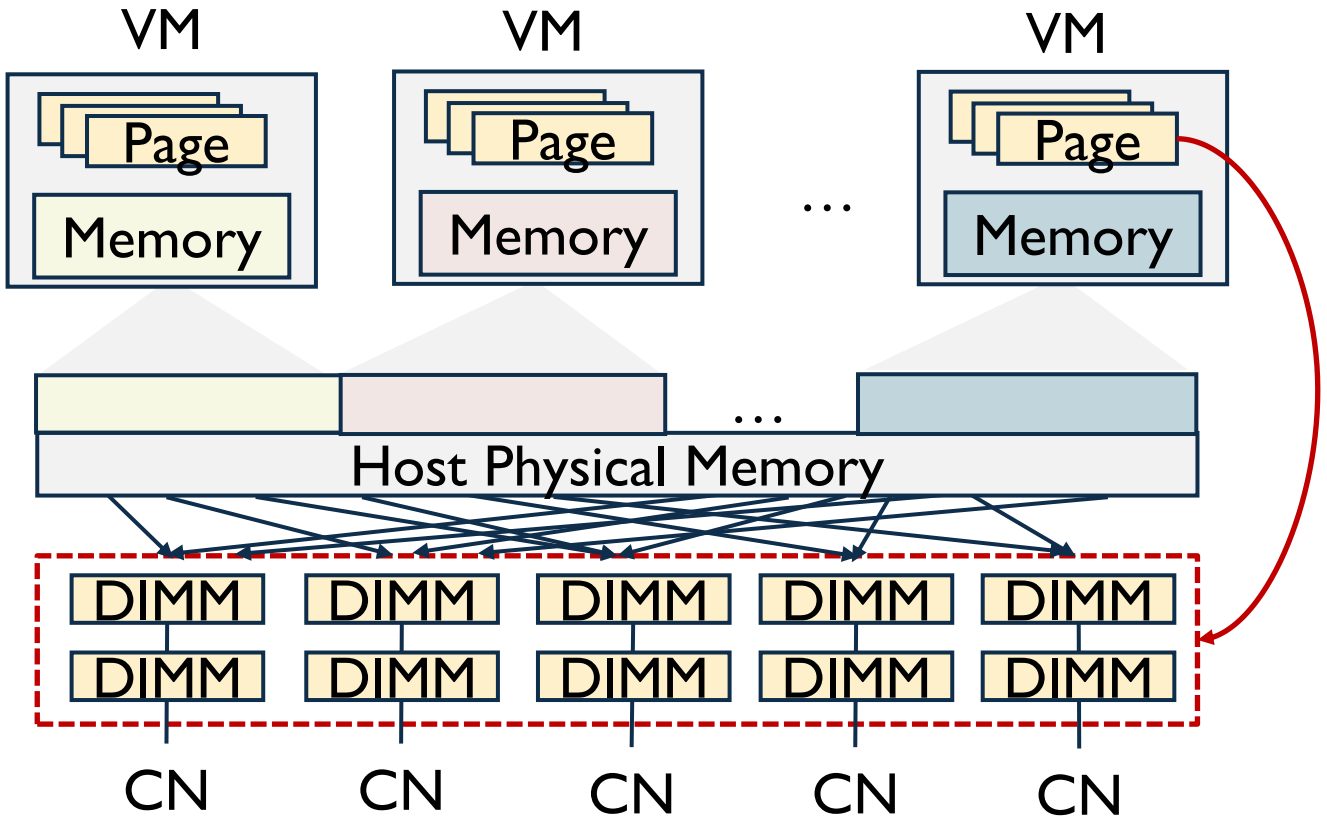

Architecture view

**Physically shared**

CN: Channel

# The Root Causes of Memory Underutilization

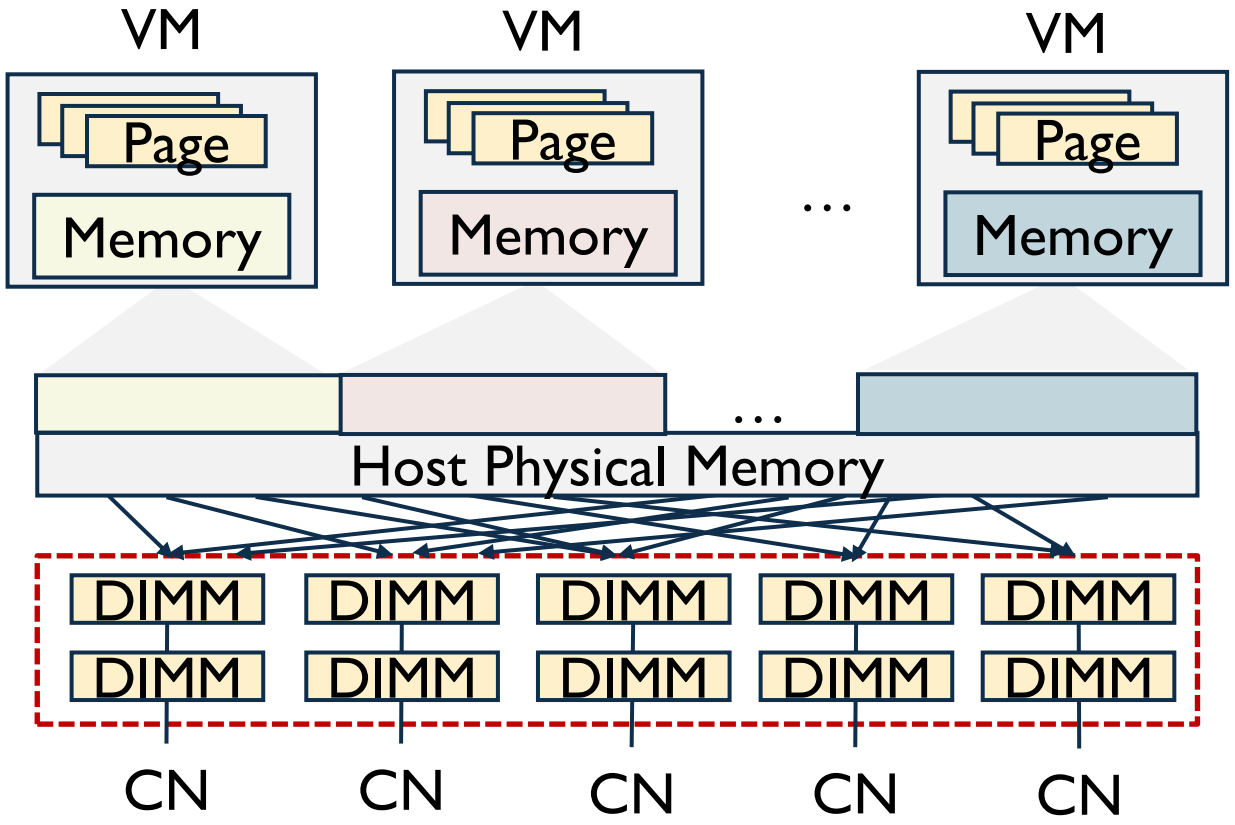
## 1. Spatial over-provisioning




CN: Channel

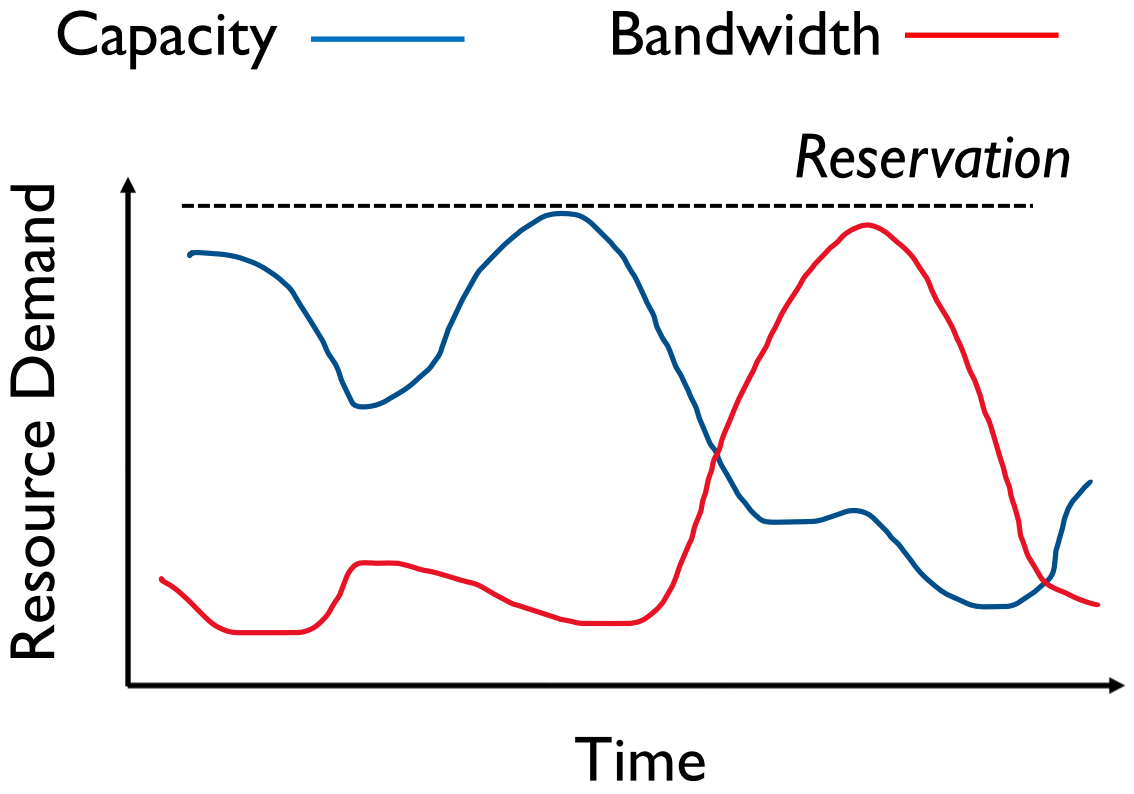
# The Root Causes of Memory Underutilization

## 1. Spatial over-provisioning




CN: Channel

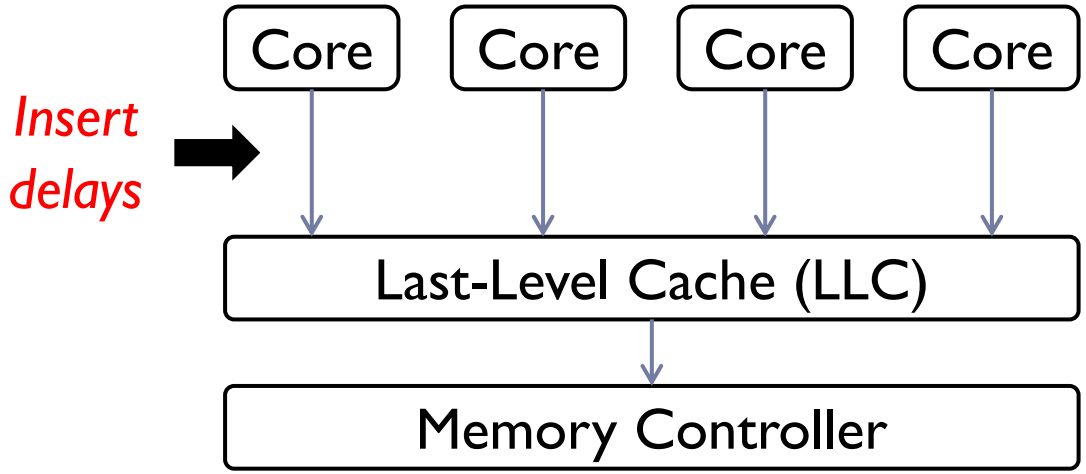
## 2. Temporal over-provisioning



# Existing Approaches: Limited Bandwidth Control

- Memory pooling & disaggregation (CXL/RDMA): focus on capacity utilization
- Bandwidth & QoS related approaches:

## 1. Hardware throttling



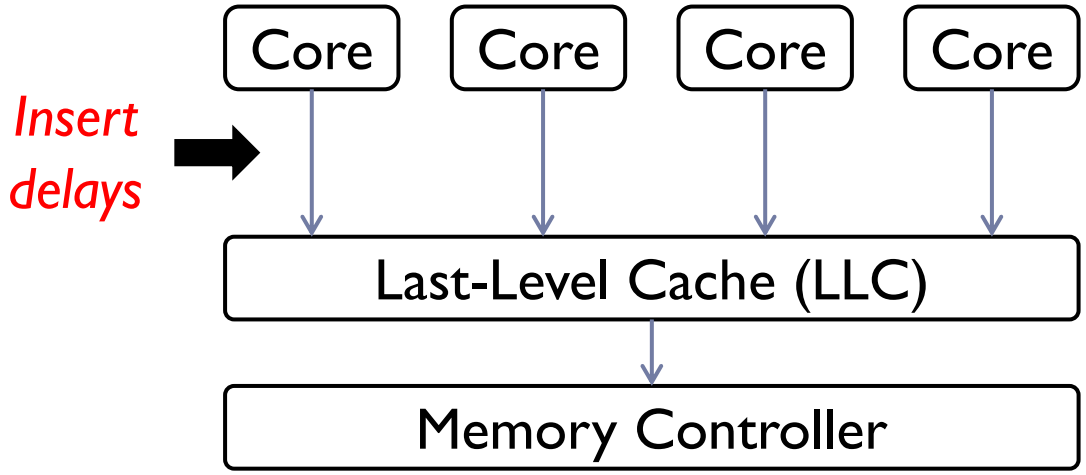

Unprecise control for bandwidth

Waste CPU cycles due to stalling

# Existing Approaches: Limited Bandwidth Control

- Memory pooling & disaggregation (CXL/RDMA): focus on capacity utilization
- Bandwidth & QoS related approaches:

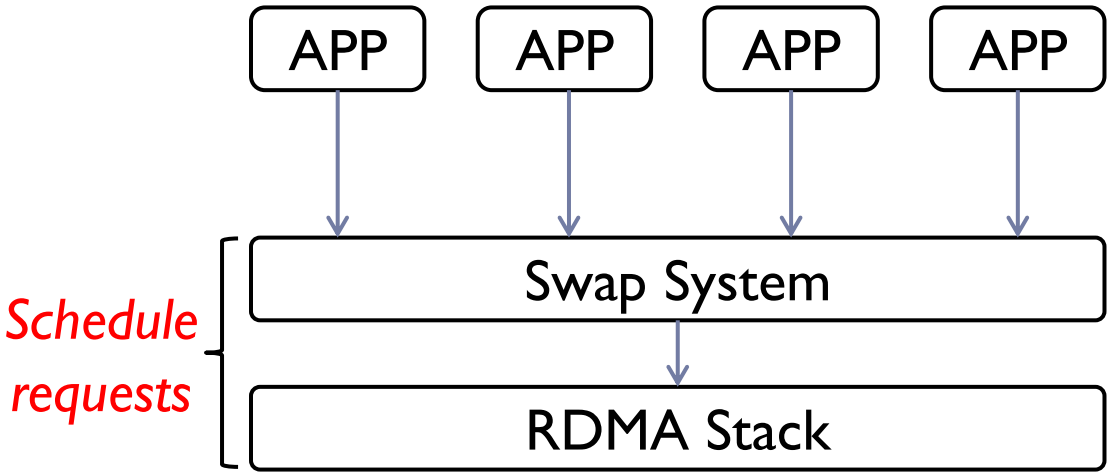
## 1. Hardware throttling




Unprecise control for bandwidth

Waste CPU cycles due to stalling

## 2. Software throttling



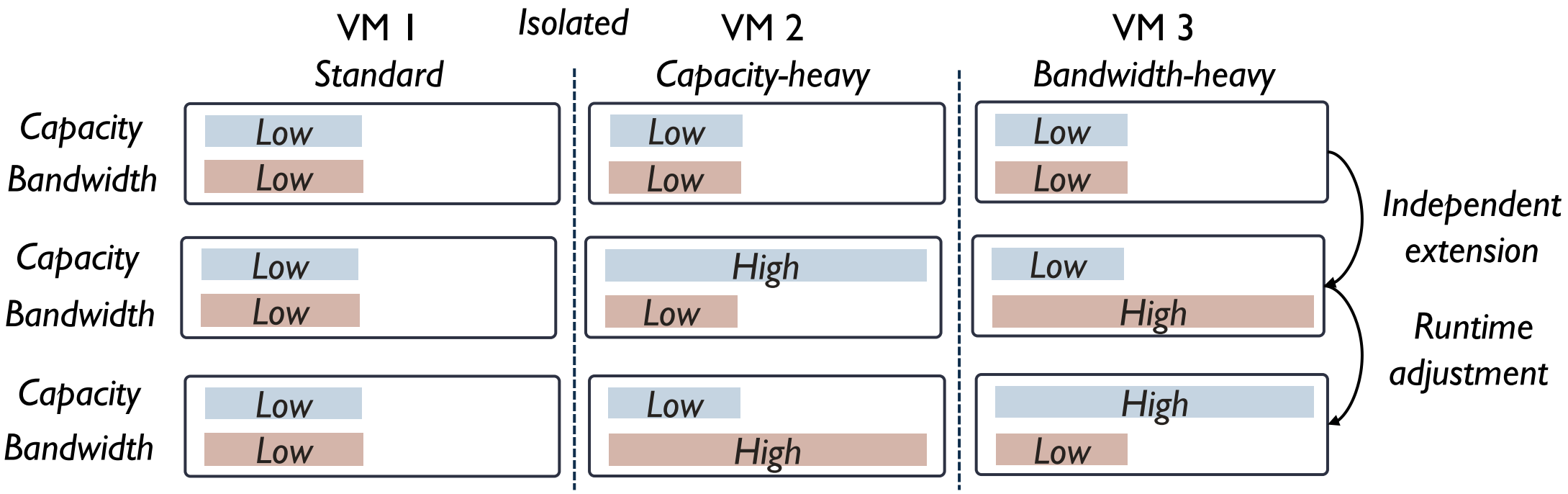

Effective on RDMA-based far memory

Cannot throttle cache-coherent devices

# RamRyder: A Software-Defined Elastic Memory

Goals

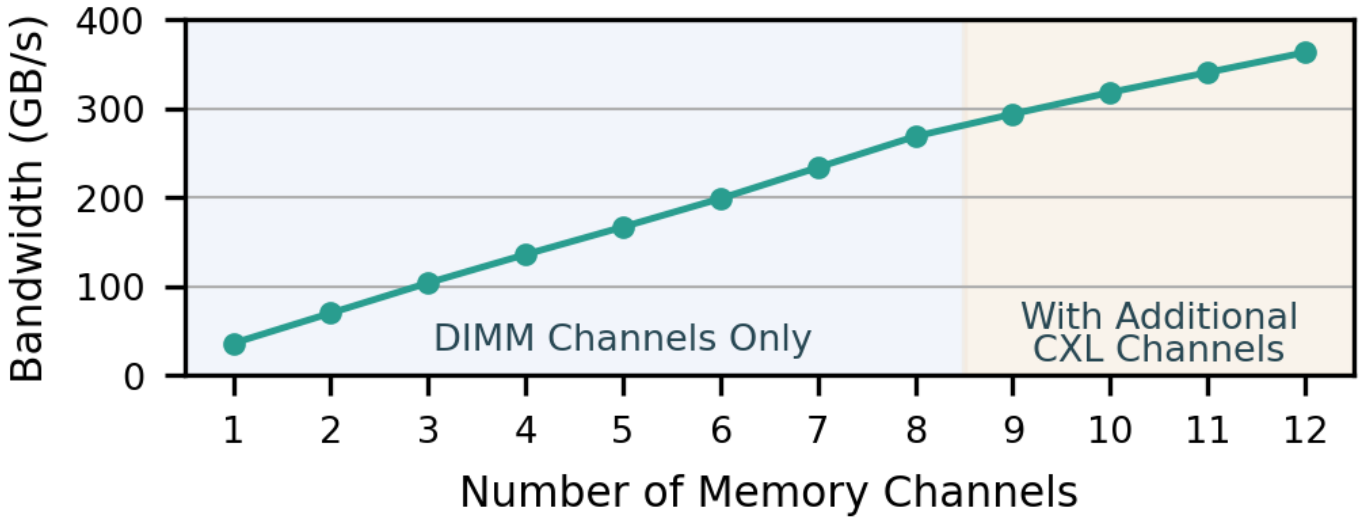
- **Goal 1**: Bandwidth allocation and performance isolation
- **Goal 2**: Independent bandwidth and capacity extension
- **Goal 3**: Runtime bandwidth and capacity adjustment



# RamRyder: A Software-Defined Elastic Memory

Key insight: managing memory at channel granularity (**new abstraction**)

→ RamRyder directly manages channel allocation and mapping in software

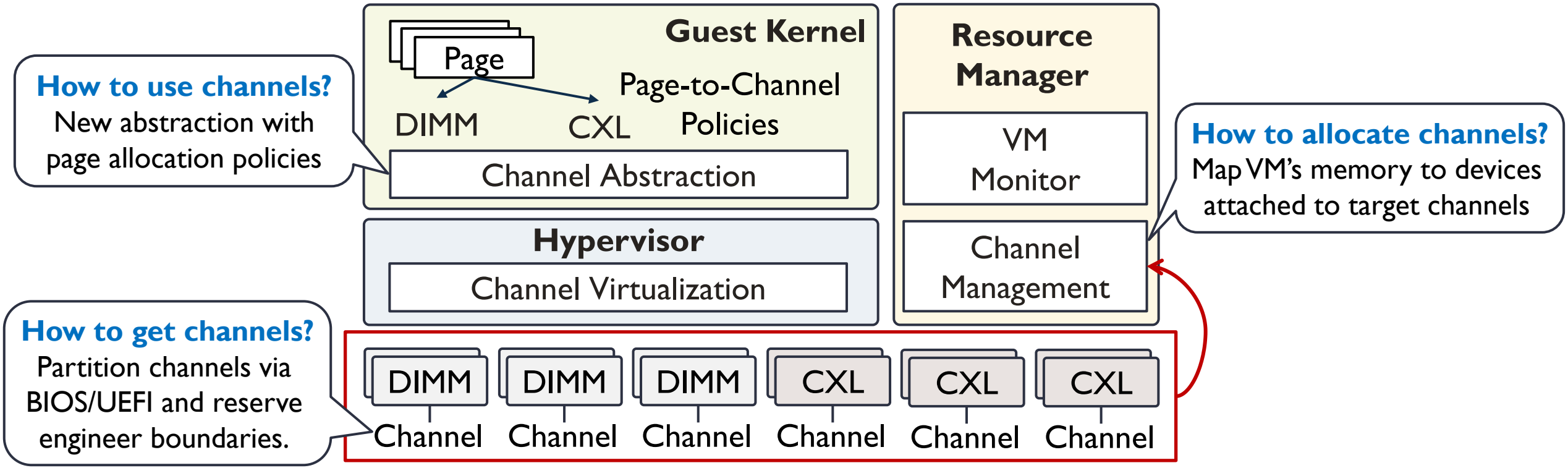



- Memory bandwidth **scales with memory channels**
- Congestion stems from channel sharing **caused by HW-controlled interleaving**

# RamRyder: A Software-Defined Elastic Memory

Key insight: managing memory at channel granularity (**new abstraction**)

→ RamRyder directly manages channel allocation and mapping in software



# How to allocate bandwidth and achieve isolation?

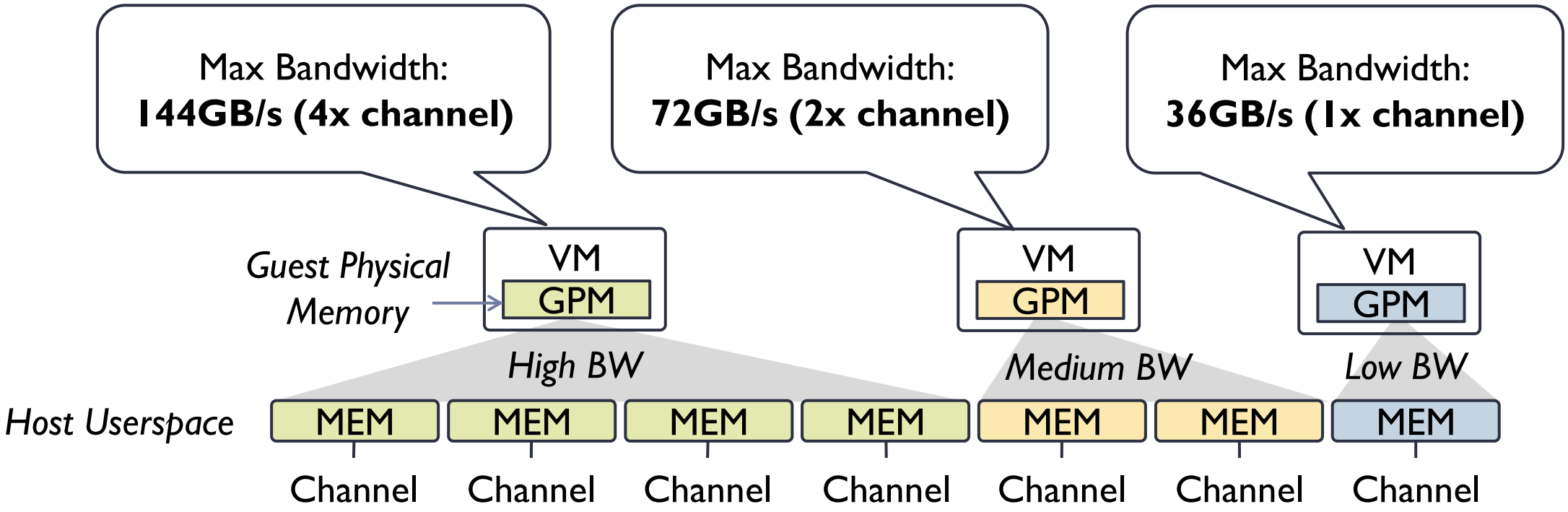
## Key tech #1: software-controlled page-to-channel mapping

1. **Channel allocation**: override **hardware-controlled** channel interleaving **by software**

# How to allocate bandwidth and achieve isolation?

## Key tech #1: software-controlled page-to-channel mapping

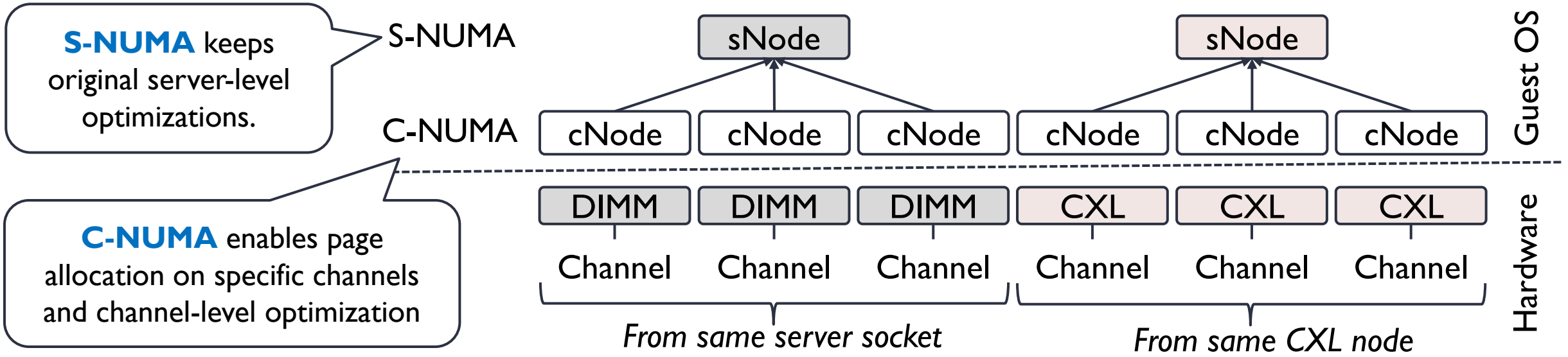
1. **Channel allocation**: override **hardware-controlled** channel interleaving **by software**



# How to allocate bandwidth and achieve isolation?

## Key tech #1: software-controlled page-to-channel mapping

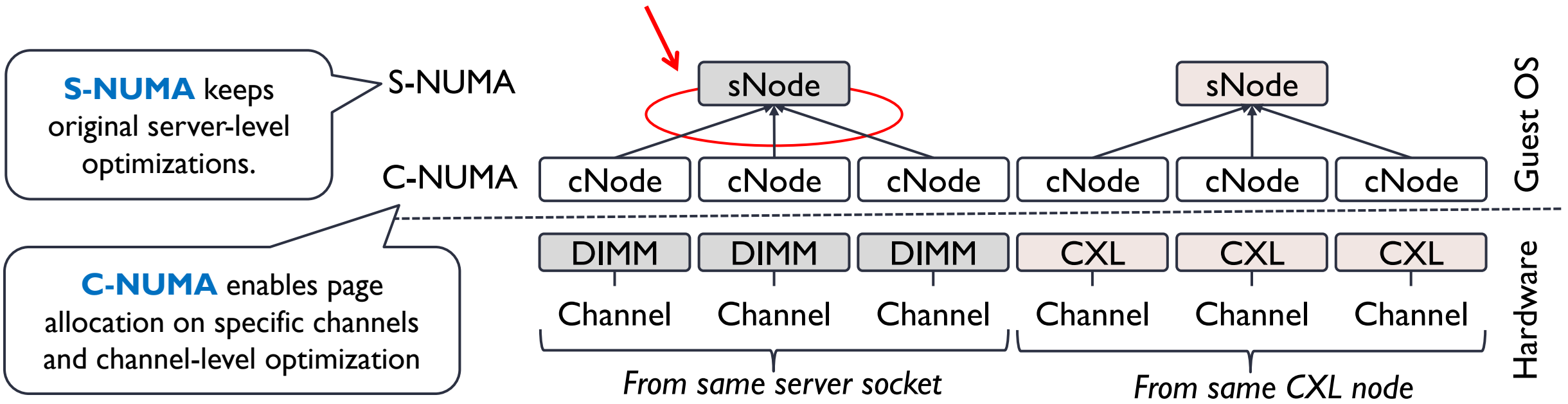
1. **Channel allocation**: override **hardware-controlled** channel interleaving **by software**
2. **Channel abstraction**: **C-NUMA** for channels and **S-NUMA** for original NUMA



# How to allocate bandwidth and achieve isolation?

## Key tech #1: software-controlled page-to-channel mapping

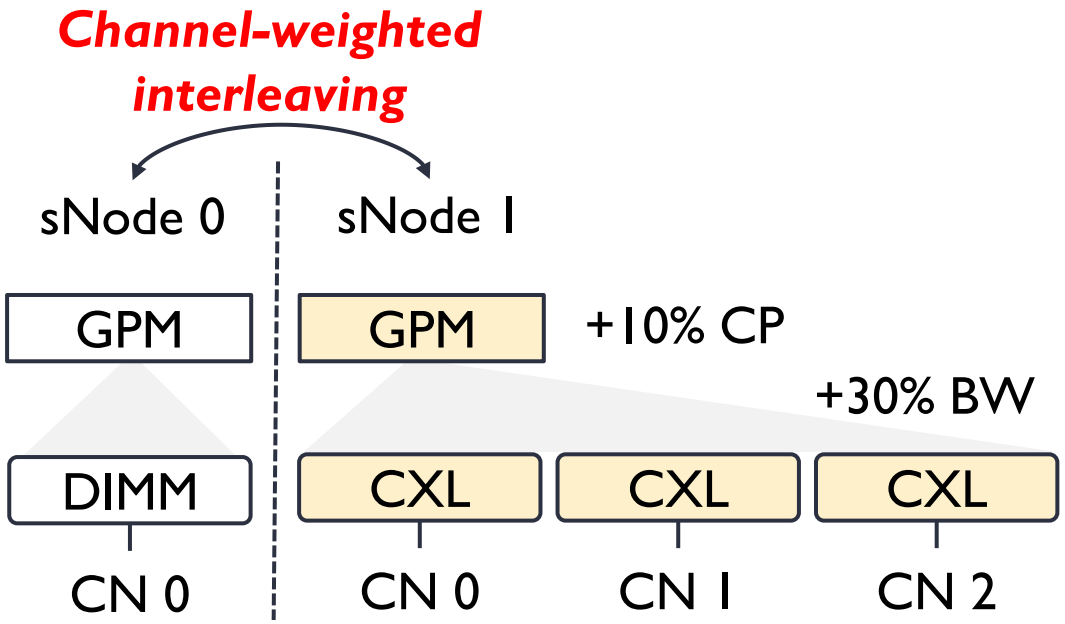
1. **Channel allocation**: override **hardware-controlled** channel interleaving **by software**
2. **Channel abstraction**: **C-NUMA** for channels and **S-NUMA** for original NUMA
3. **Page-to-channel allocation**: **interleave equally** across all cNodes under each sNode



# How to extend bandwidth and capacity independently?

## Key tech #2: channel-weighted interleaving (page allocation policy)

- **Host:** allocate additional capacity and channels from CXL devices
- **Guest**: new cross-tier policy -- interleave across tiers based on bandwidth




**Example: additional bandwidth**

Example:

DIMM Channel: 36GB/s
CXL Channel: 24GB/s

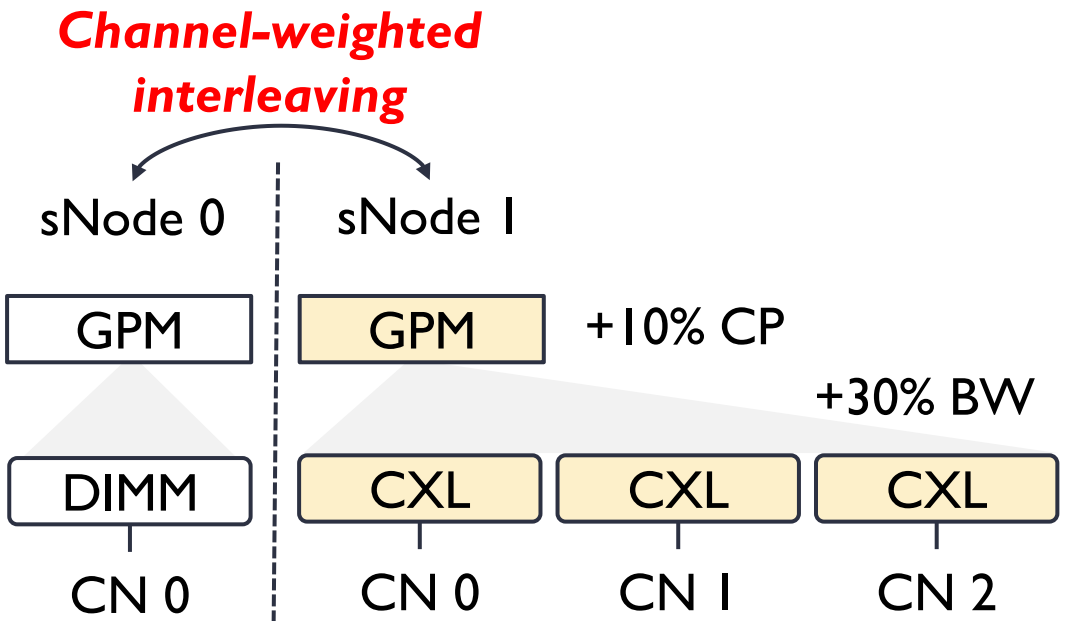
Weight Ratio:
36 x 1 : 24 x 3 = 1 : 2

GPM: Guest Physical Memory  CP: Capacity  BW: Bandwidth  CN: Channel

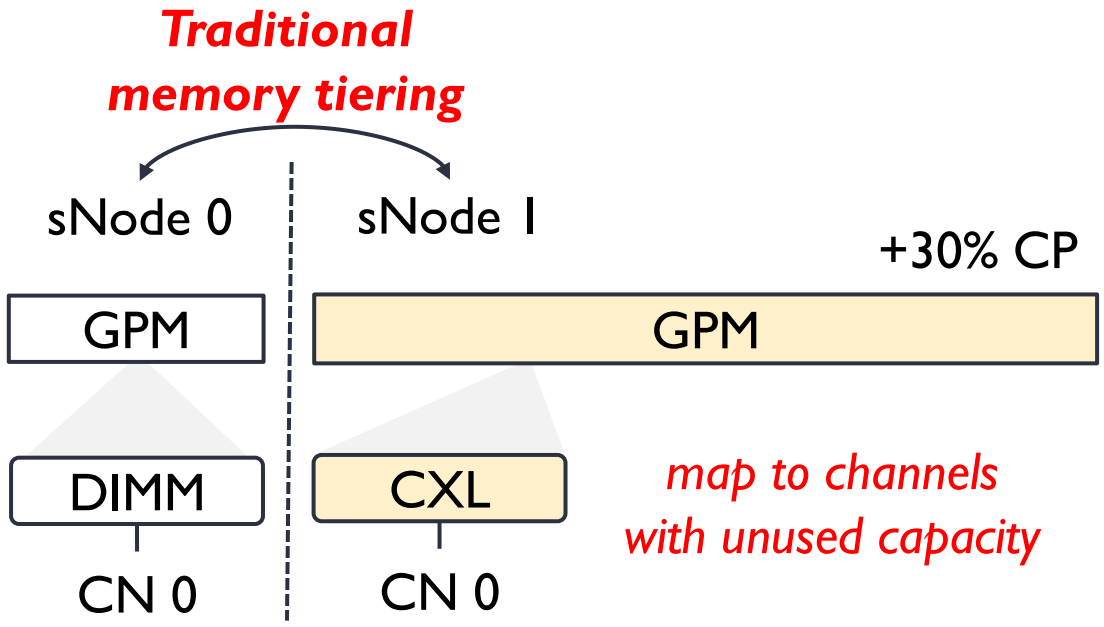
# How to extend bandwidth and capacity independently?

## Key tech #2: channel-weighted interleaving (page allocation policy)

- **Host:** allocate additional capacity and channels from CXL devices
- **Guest**: new cross-tier policy -- interleave across tiers based on bandwidth




**Example: additional bandwidth**

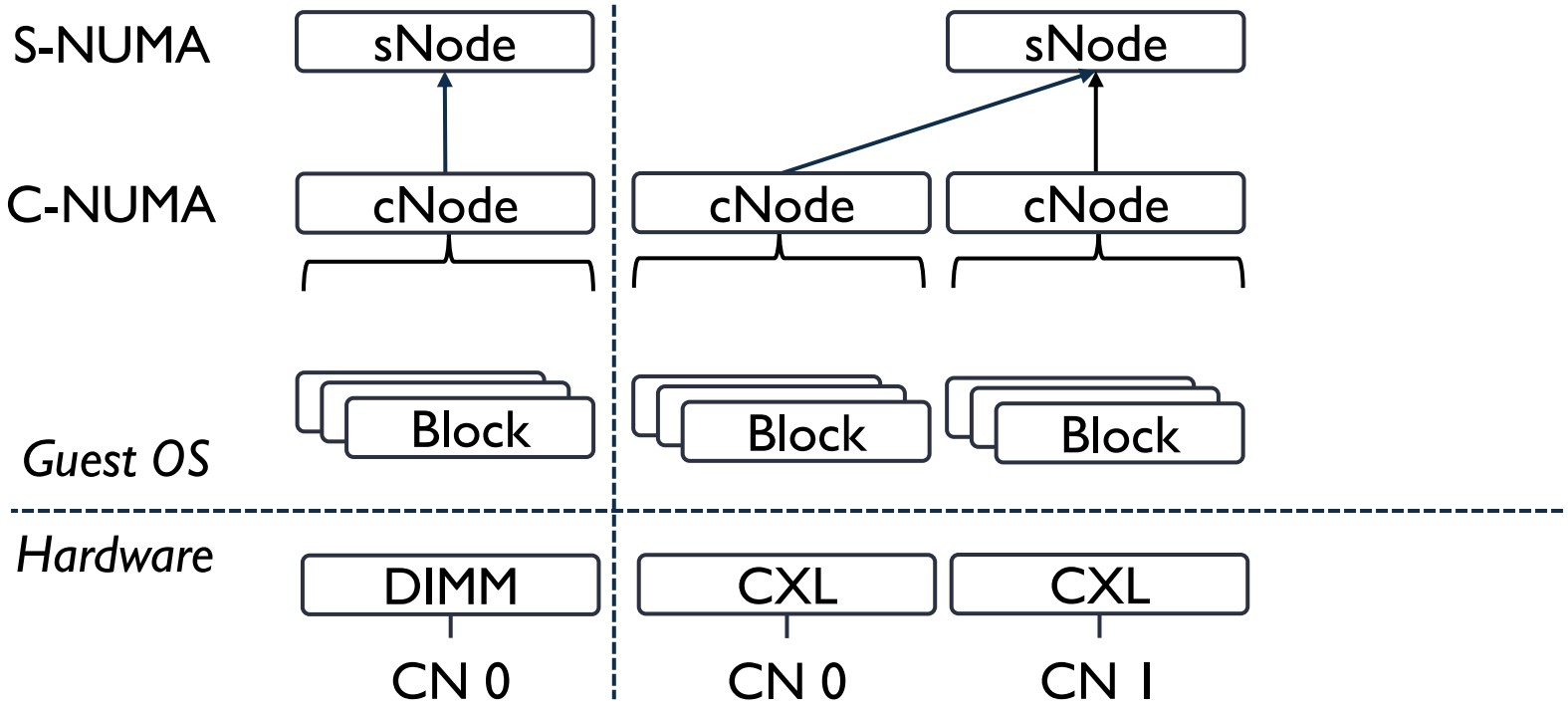



**Example: additional capacity**

GPM: Guest Physical Memory CP: Capacity BW: Bandwidth CN: Channel

# How to adjust bandwidth and capacity dynamically?

## Key tech #3: channel hot-plug/unplug and page redistribution

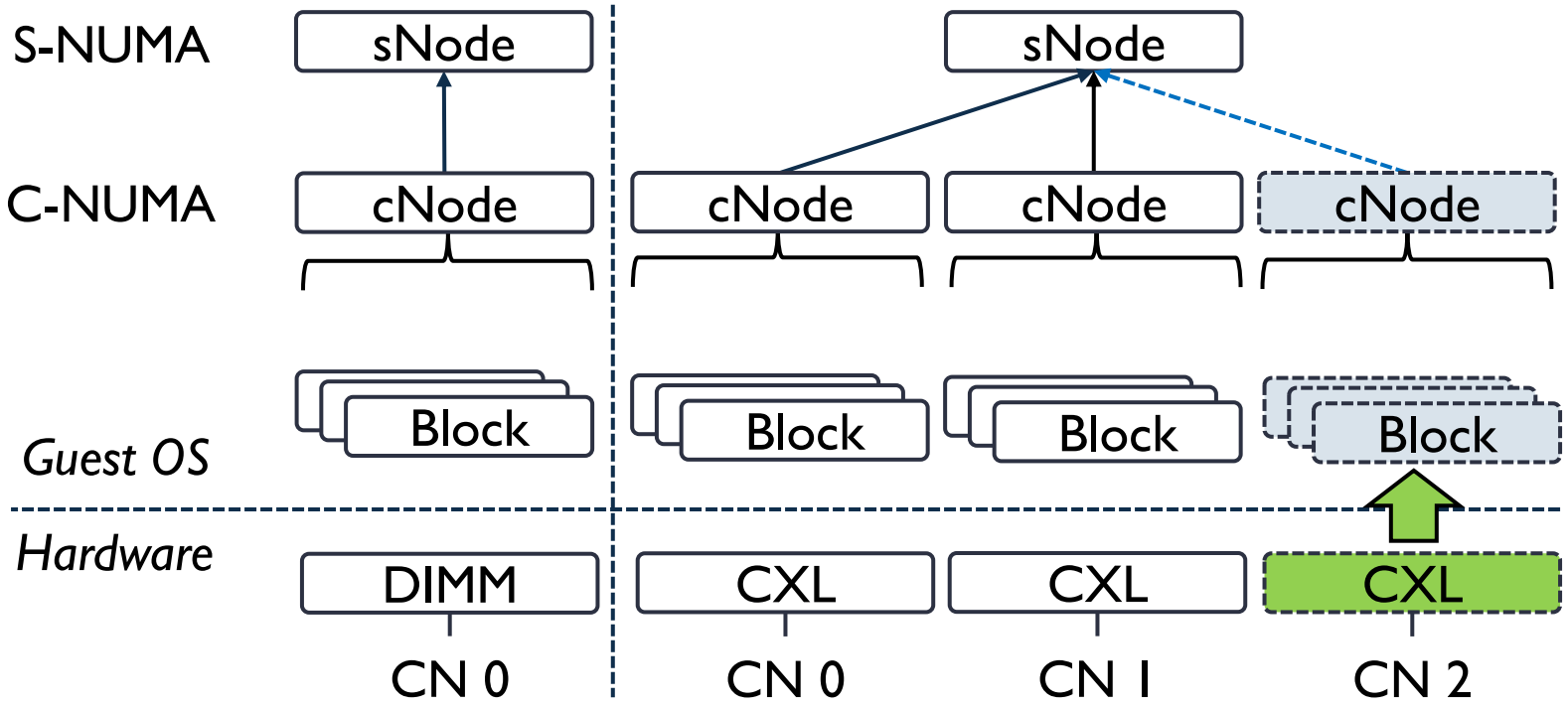



CN: Channel

# How to adjust bandwidth and capacity dynamically?

## Key tech #3: channel hot-plug/unplug and page redistribution

1. **Channel hot-plug**: hot-plug additional channels to guest for bandwidth extension

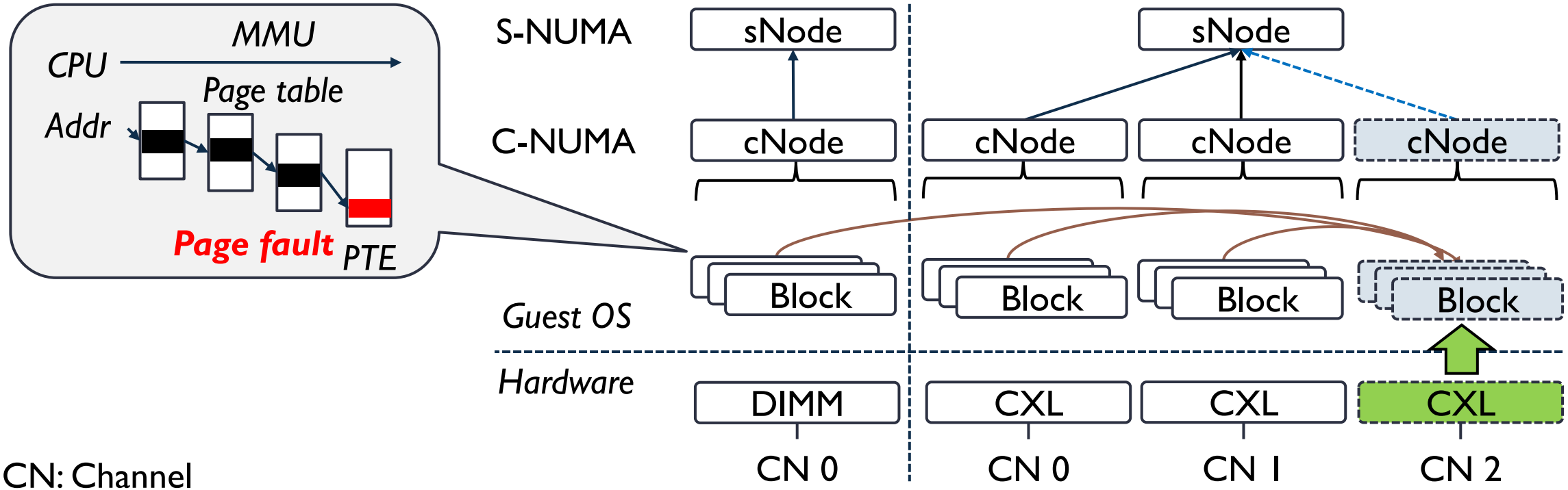



CN: Channel

# How to adjust bandwidth and capacity dynamically?

## Key tech #3: channel hot-plug/unplug and page redistribution

1. **Channel hot-plug**: hot-plug additional channels to guest for bandwidth extension
2. **Page redistribution**: redistribute existing pages through **lazy migration**

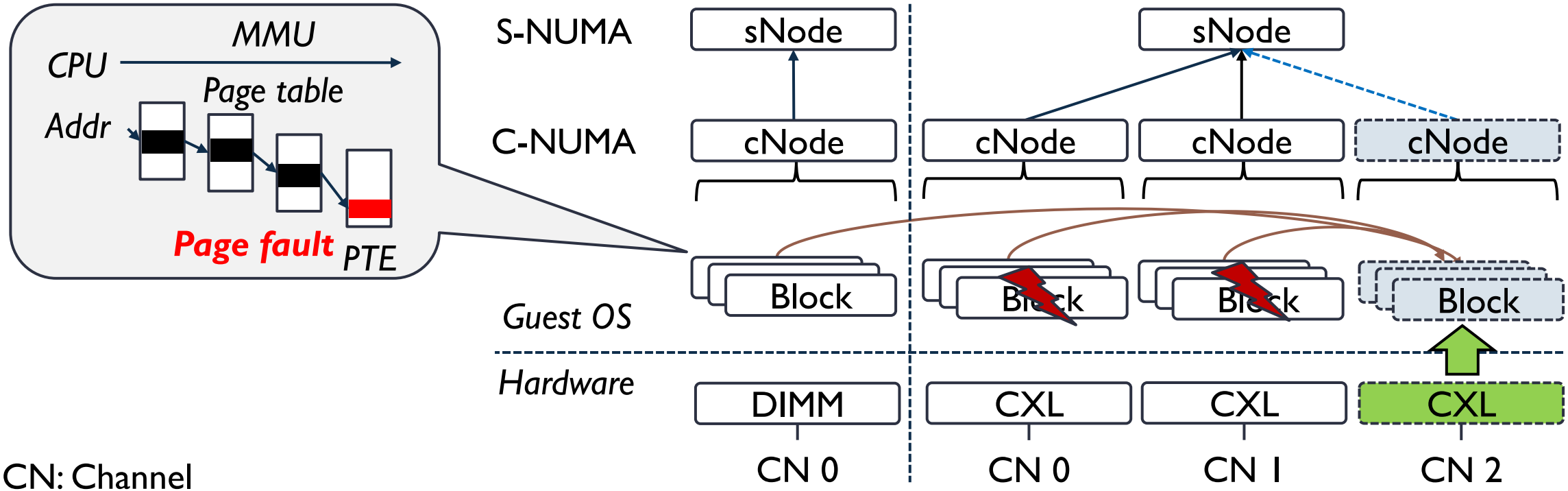



CN: Channel

# How to adjust bandwidth and capacity dynamically?

## Key tech #3: channel hot-plug/unplug and page redistribution

1. **Channel hot-plug**: hot-plug additional channels to guest for bandwidth extension
2. **Page redistribution**: redistribute existing pages through **lazy migration**
3. **Memory reclamation**: reclaim memory blocks to maintain the same capacity




CN: Channel

# Evaluation

## Testbed

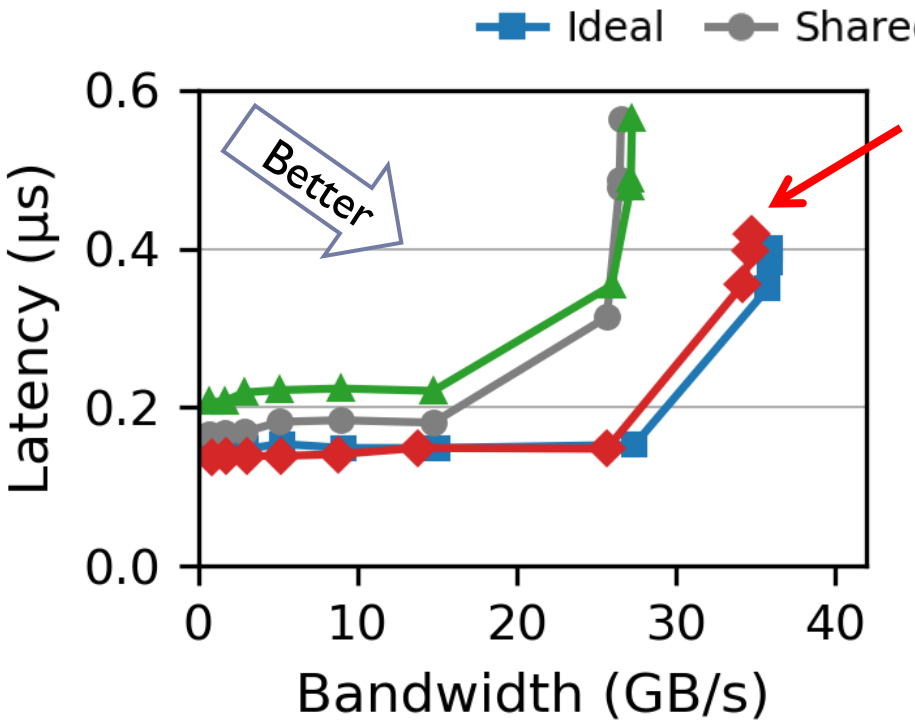
- AMD EPYC Zen5 Processor -- 128 cores
- 12 DDR5 channels, 32GB DDR5 DIMM per channel
- 4 256GB CXL 2.0 memory devices (one channel per device)

## Goals

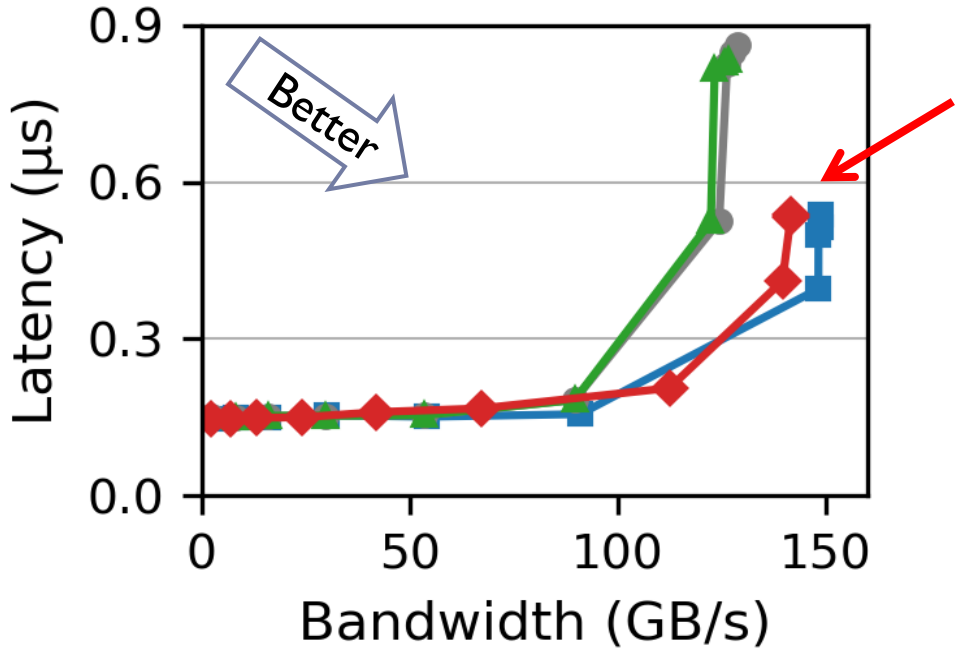
- Can RamRyder achieve bandwidth allocation & isolation (**performance benefits**)?
- Can RamRyder improve bandwidth utilization (**utilization benefits**)?
- What are the major overheads of RamRyder (**major overheads**)?
- More experiments are in the paper

# Performance Benefits

Microbenchmarks --- four VMs concurrently run streaming workloads




**Small VM: 16 vCPUs**



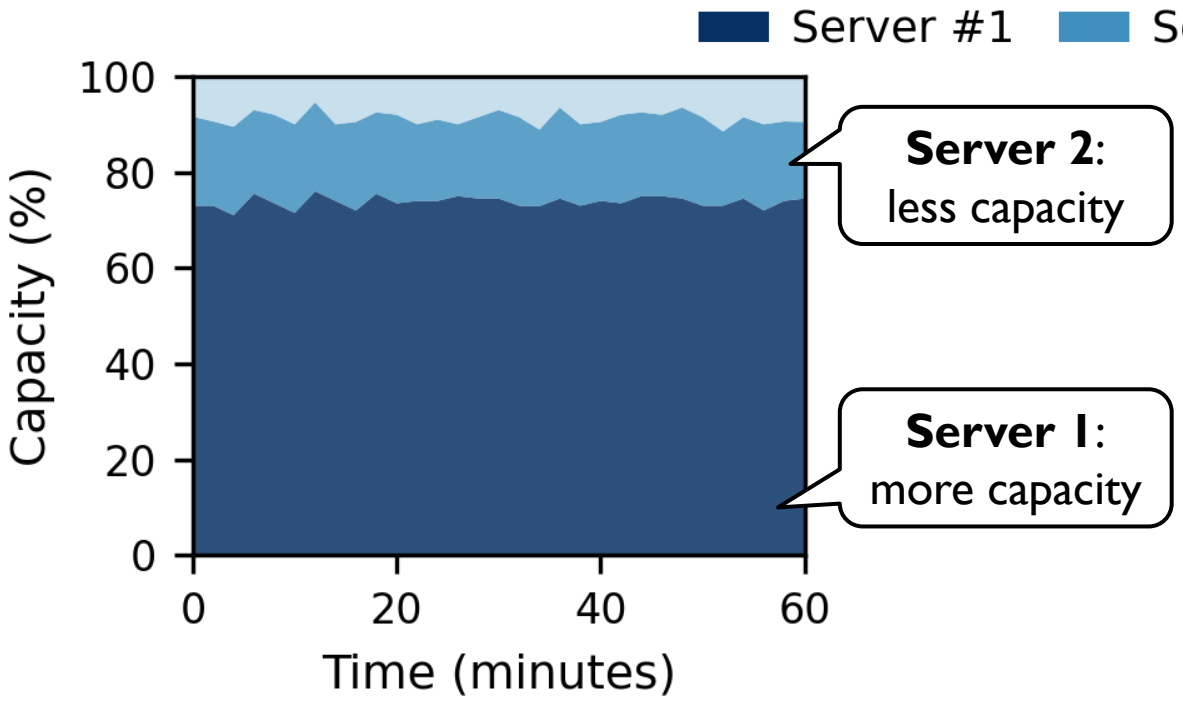

**Large VM: 48 vCPUs**

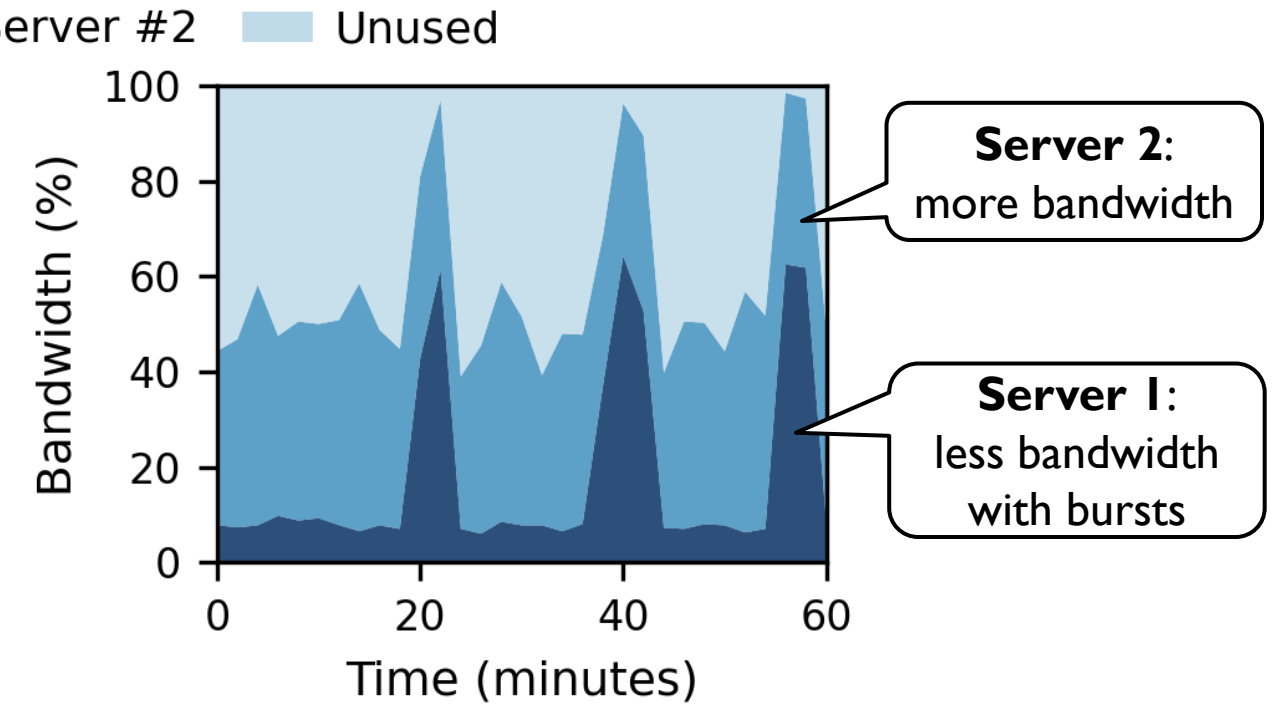
Smaller VMs are more vulnerable to interference from co-located VMs.
RamRyder guarantees performance QoS without dedicated hardware

# Utilization Benefits

Reproducing memory workloads from cluster traces

- Pair servers with complementary resource requirements

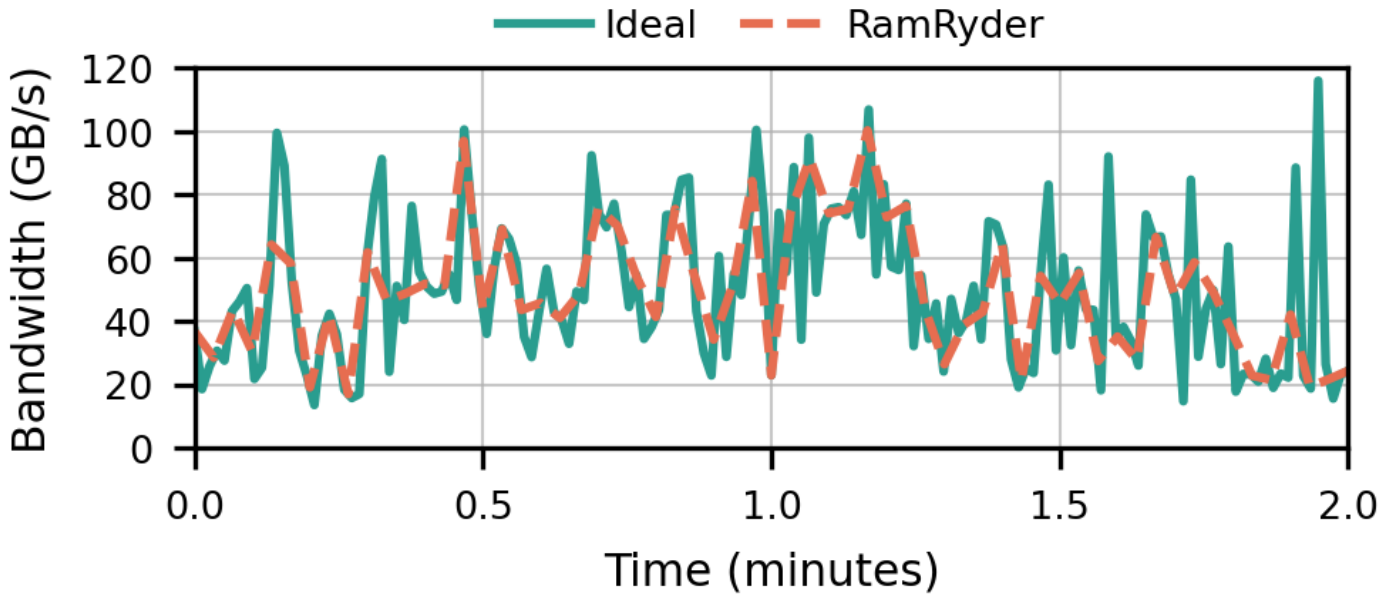



By consolidating VMs with complementary resource requirements, RamRyder improve avg. bandwidth utilization at P90 by 43%

# Major Overheads

- Software-defined interleaving overhead (<5% vs. hardware interleaving)
- Dynamic bandwidth allocation overhead (second-level)




**Overhead breakdown**

- Channel hot-plug/unplug: < 1s
- Page redistribution: several seconds depending on the amount of pages

**Optimization**:

1. Prioritize migrating hot pages that dominate bandwidth.
2. Predict bandwidth usage patterns and adjust bandwidth in advance.

# Conclusion

- **Memory allocation** in the cloud is **less flexible** and thus leads to stranded capacity and bandwidth.

- **RamRyder** is a new approach that **takes the role of hardware** and directly manages memory channel allocation and mapping in software

https://github.com/ramryder-project/ramryder

# Thank You!

# Q & A