Spirit: Fair Allocation of Interdependent Resources in Remote Memory Systems
SOSP' 2025

Author: Seung-seob Lee, Jachym Putta, etc. Yale University

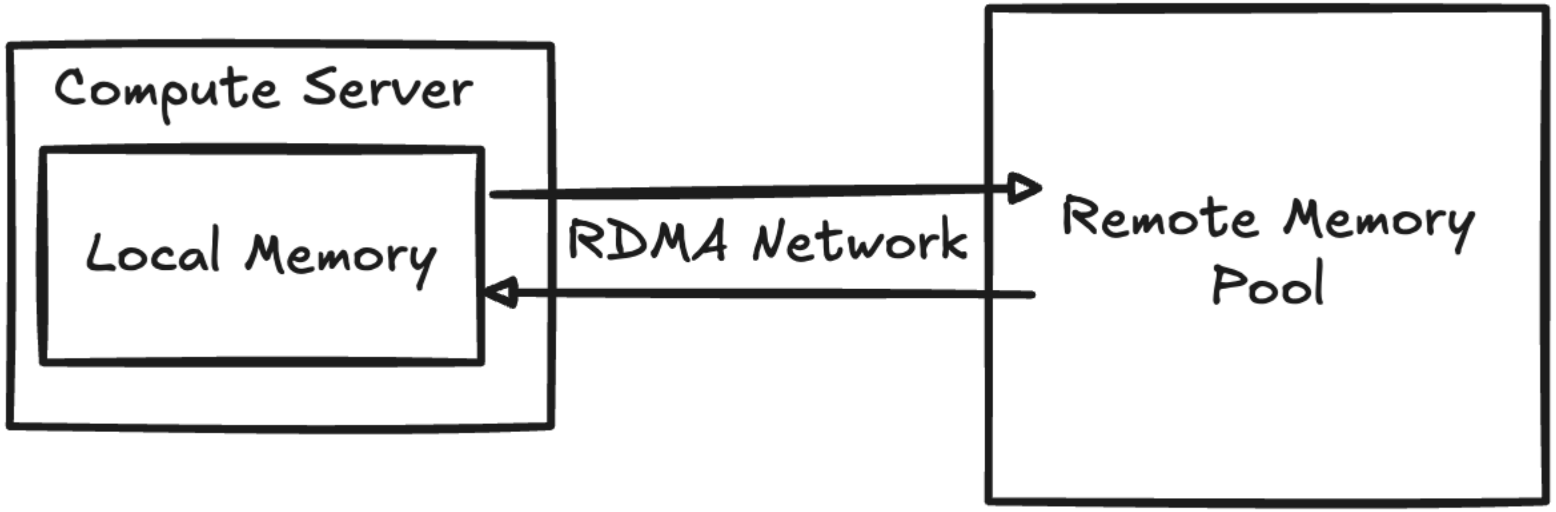
Presenter: Yicheng Zhang USTC

2025.11.25

Outline

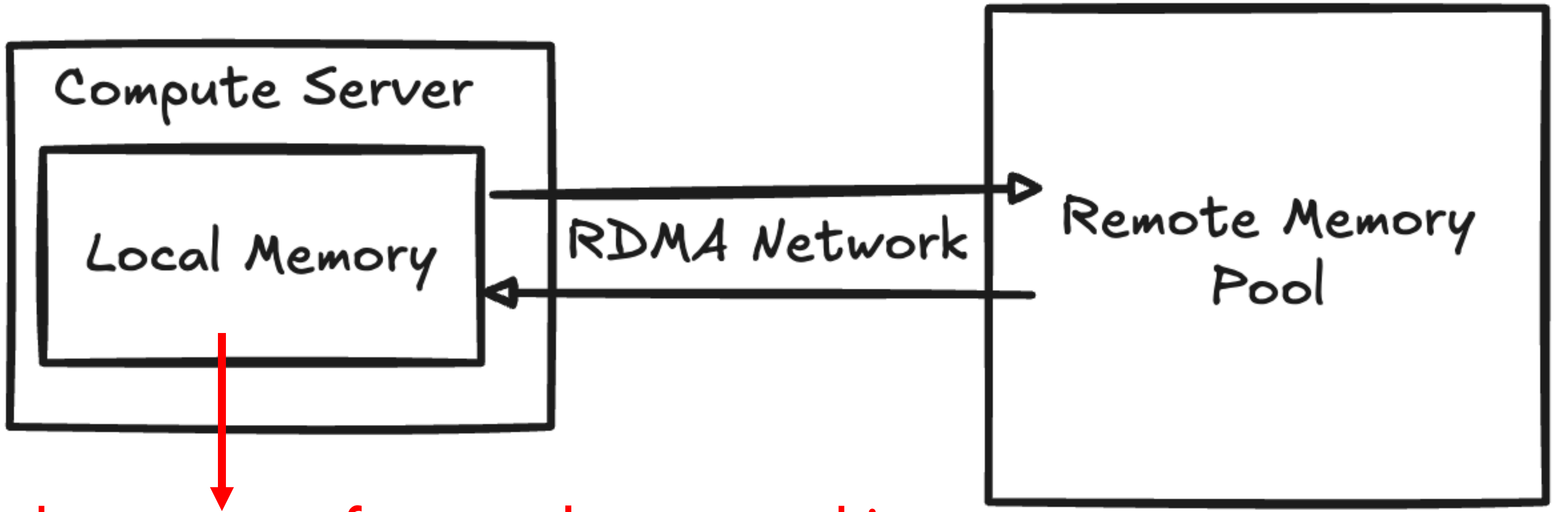
The Rise of Remote Memory Systems

- Remote memory systems extend server memory capacity, improving application performance and memory utilization across servers.
- Linux swap subsystem offers a zero-effort path to enable remote memory for legacy applications.



The Rise of Remote Memory Systems

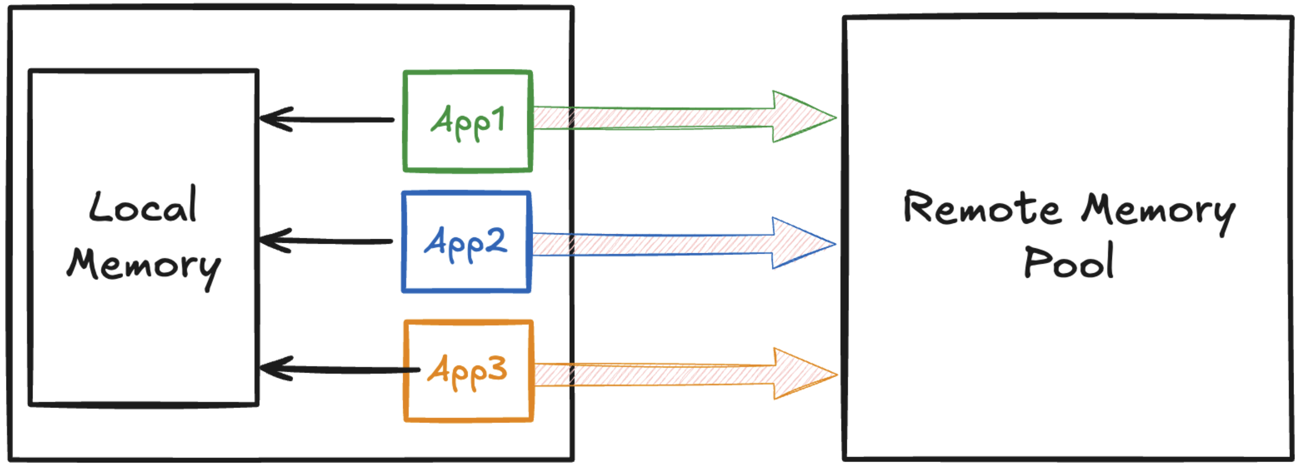
- ❑Remote memory systems extend server memory capacity, improving application performance and memory utilization across servers.
- ❑Linux swap subsystem offers a zero-effort path to enable remote memory for legacy applications.

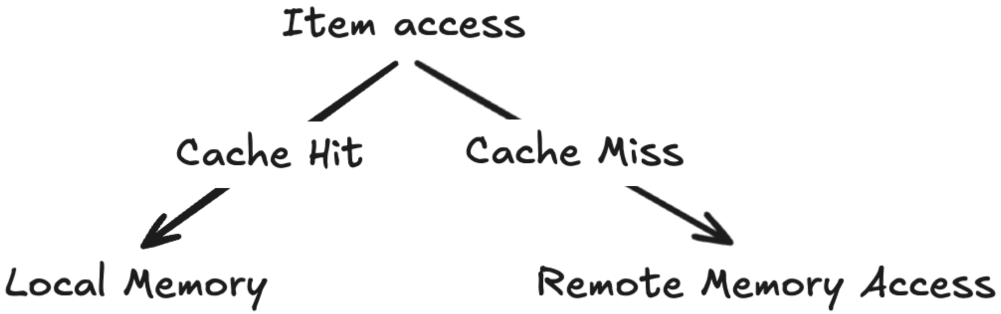


Act as a cache to store frequently accessed items

Multi-Tenant Environment & Resource Allocation

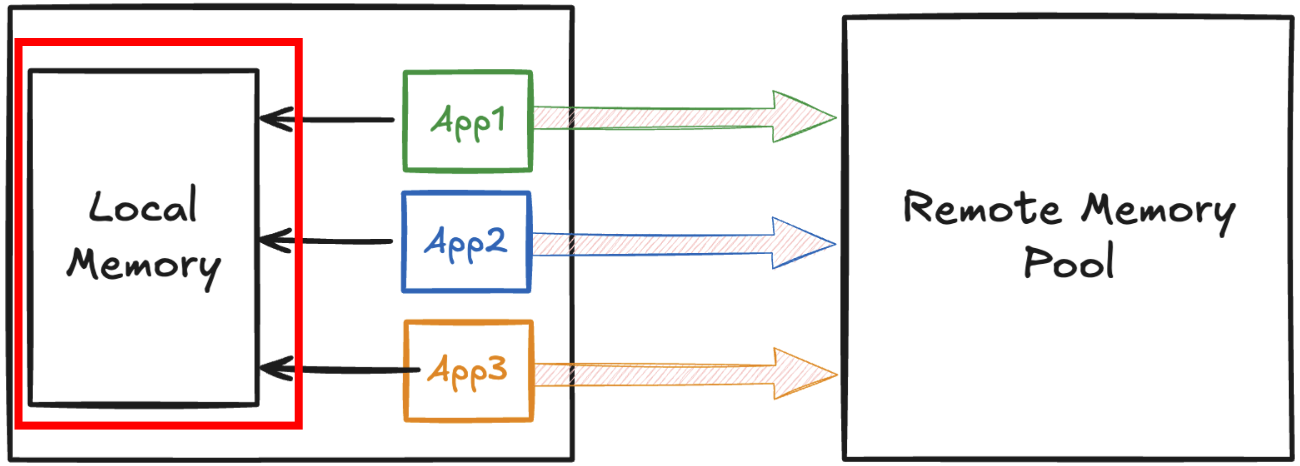
- Multi-Tenant environment is common in cloud service, and resource allocation is the core problem.



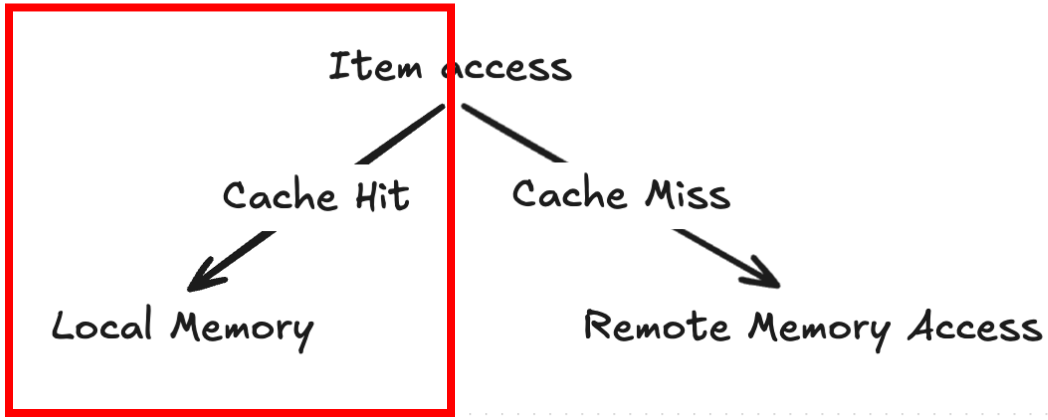


Multi-Tenant Environment & Resource Allocation

- Multi-Tenant environment is common in cloud service, and resource allocation is the core problem.



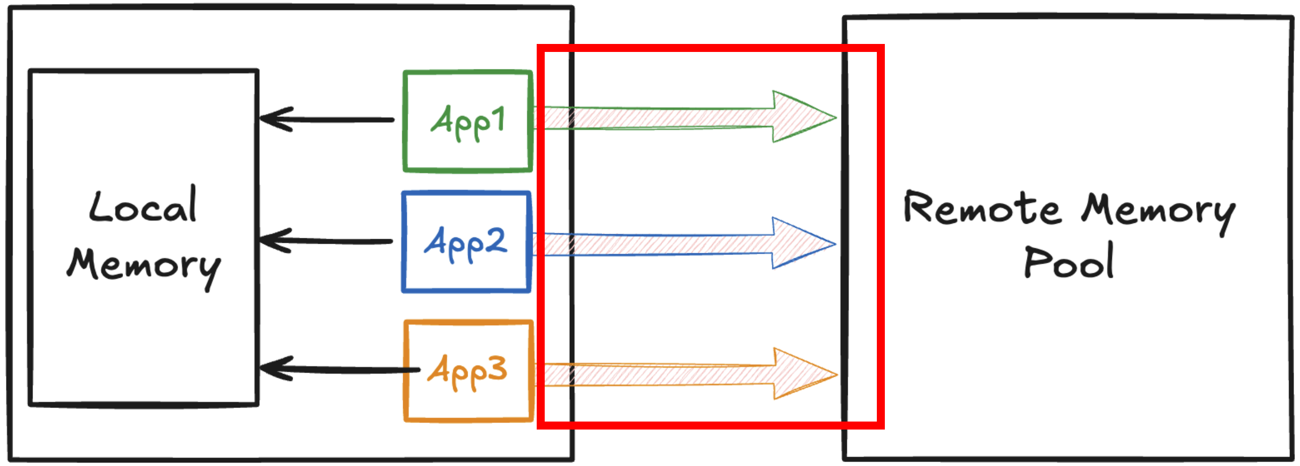
Memory Allocation



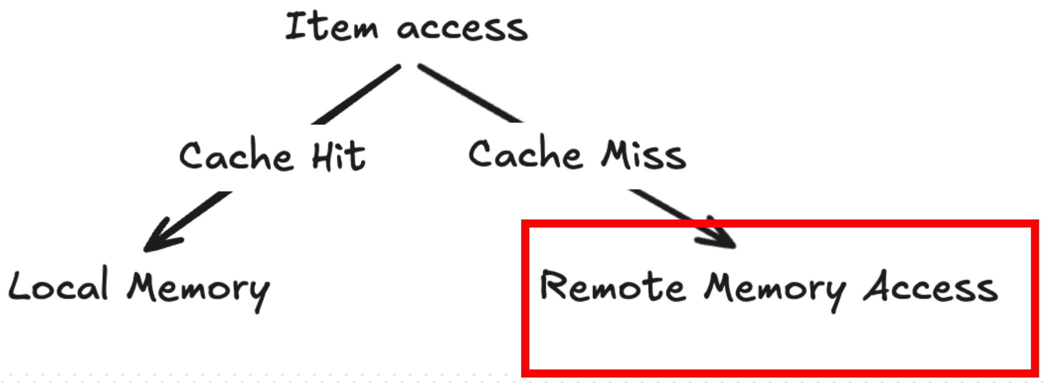
Memory Allocation

Multi-Tenant Environment & Resource Allocation

- Multi-Tenant environment is common in cloud service, and resource allocation is the core problem.



Bandwidth Allocation



Bandwidth Allocation

The Interdependence of Cache and Bandwidth

- An application with a larger cache allocation may need less bandwidth.
- The nature of this relationship is application-dependent.
- Tend to be unknown before application deployment. (workload matters)

Key-Value store → With skewed memory access pattern → Cache Size ↑ Bandwidth ⬇

Stream Processing → With poor temporal locality → Cache Size ⬆ Bandwidth ↓

Limitations of Existing Schemes

- ❑ Classical multi-resource allocation schemes, like DRF
  - ❖ Applications are forced to specify a fixed cache and bandwidth demand
  - ❖ Require applications to submit their resource demands upfront
- ❑ Recent Approaches
  - ❖ Using runtime feedback from application about their demands, need to modify applications.

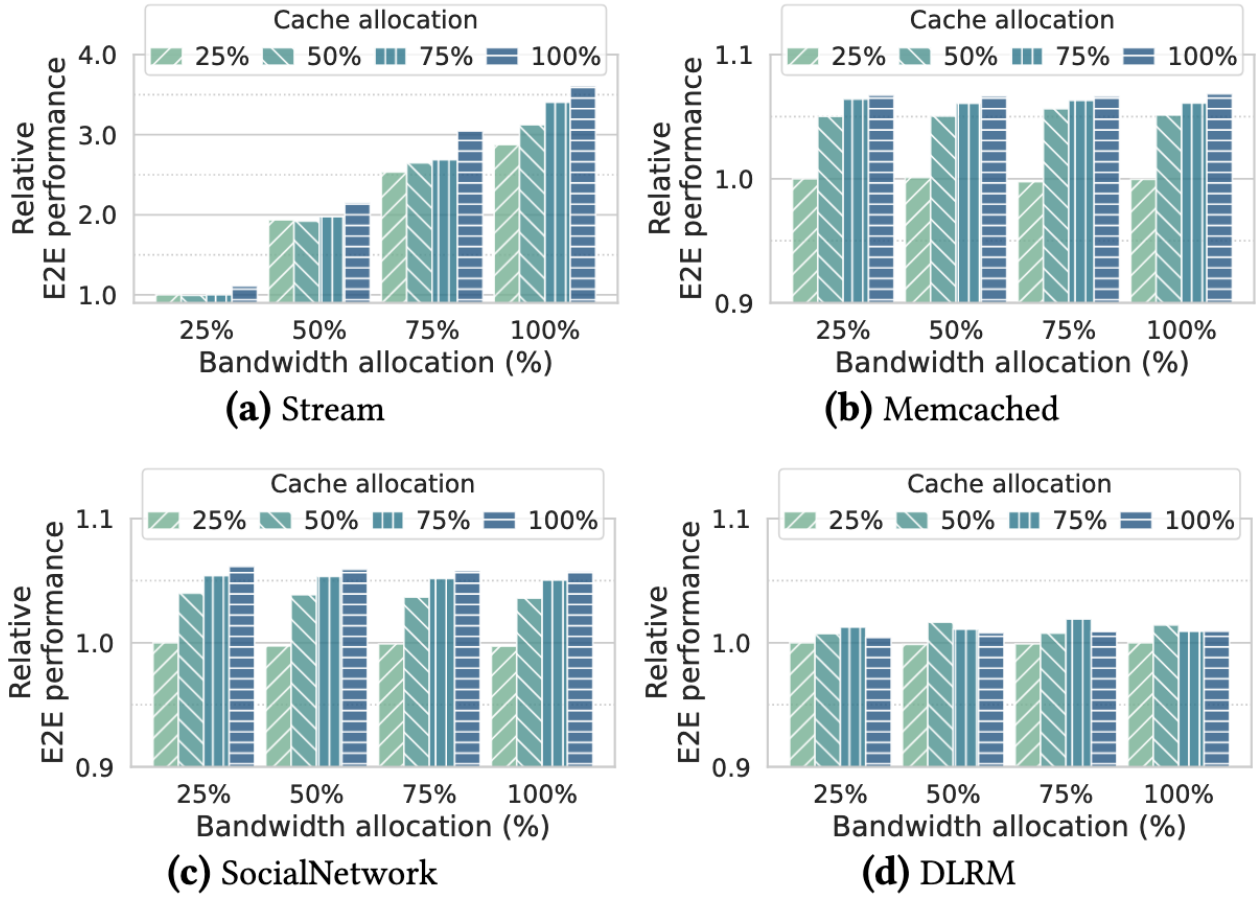
Limitations of Existing Schemes

- ❑ Classical multi-resource allocation schemes, like DRF
  - ❖ Applications are forced to specify a fixed cache and bandwidth demand
  - ❖ Require applications to submit their resource demands upfront
- ❑ Recent Approaches
  - ❖ Using runtime feedback from application about their demands, need to modify applications.

All of them don't consider the interdependence of cache and bandwidth!

Observation of various applications

- ❑Memcached (key-value store), dubbed STREAM (stream processing), DLRM (recommendation model), DeathStarBench(social network)

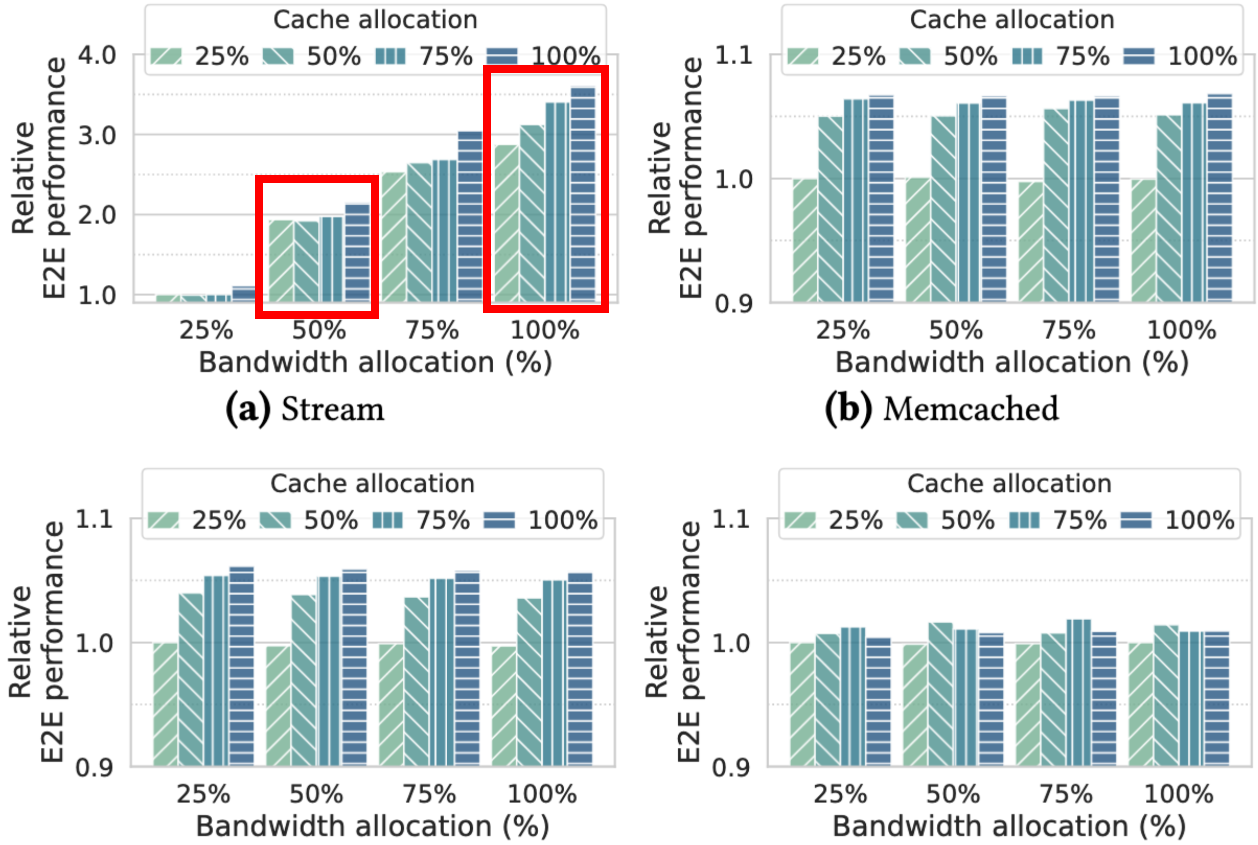


(a) Stream (b) Memcached (c) SocialNetwork (d) DLRM

Observation of various applications

- Memcached (key-value store), dubbed STREAM (stream processing), DLRM (recommendation model), DeathStarBench(social network)

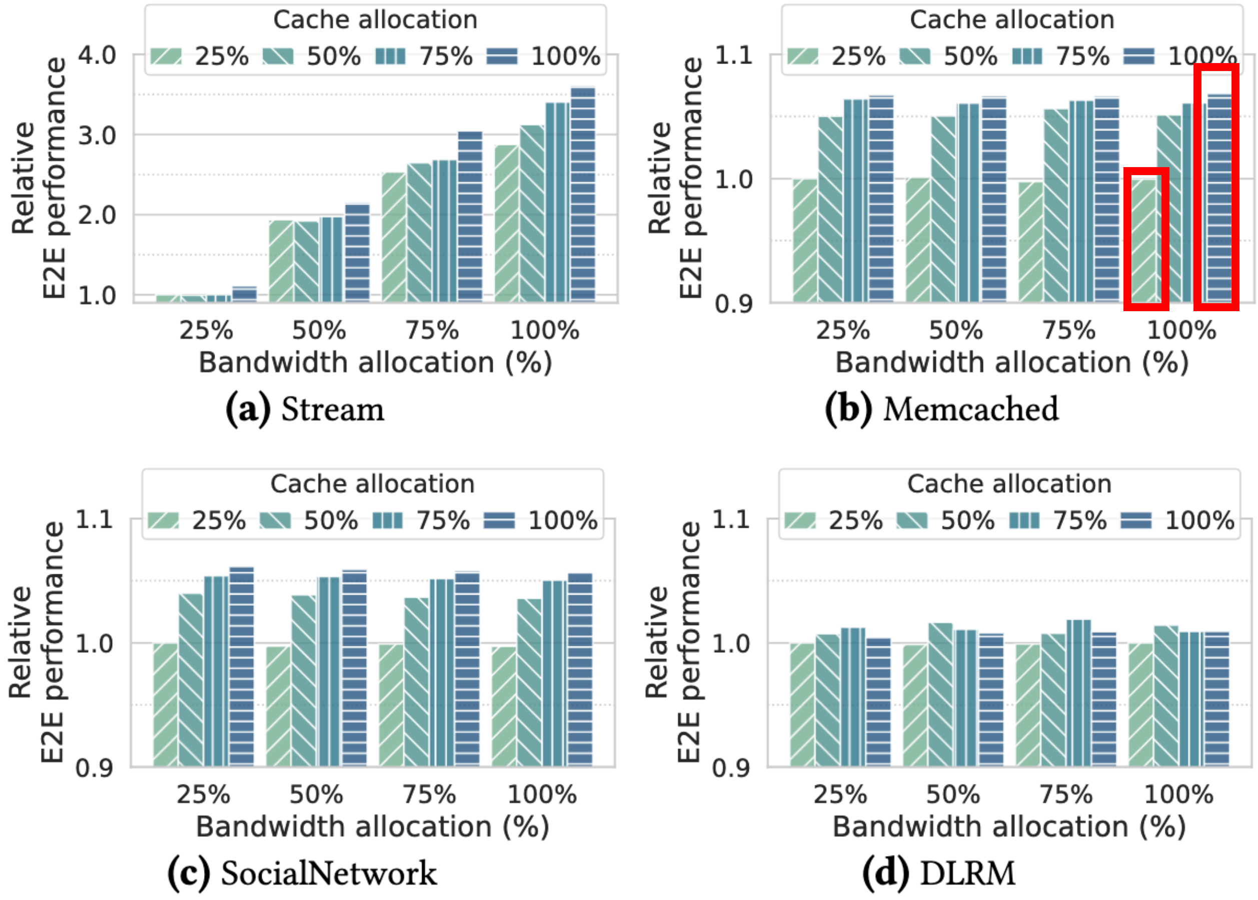
Shows performance variance of different bandwidth



(a) Stream **(b)** Memcached **(c)** SocialNetwork **(d)** DLRM

Observation of various applications

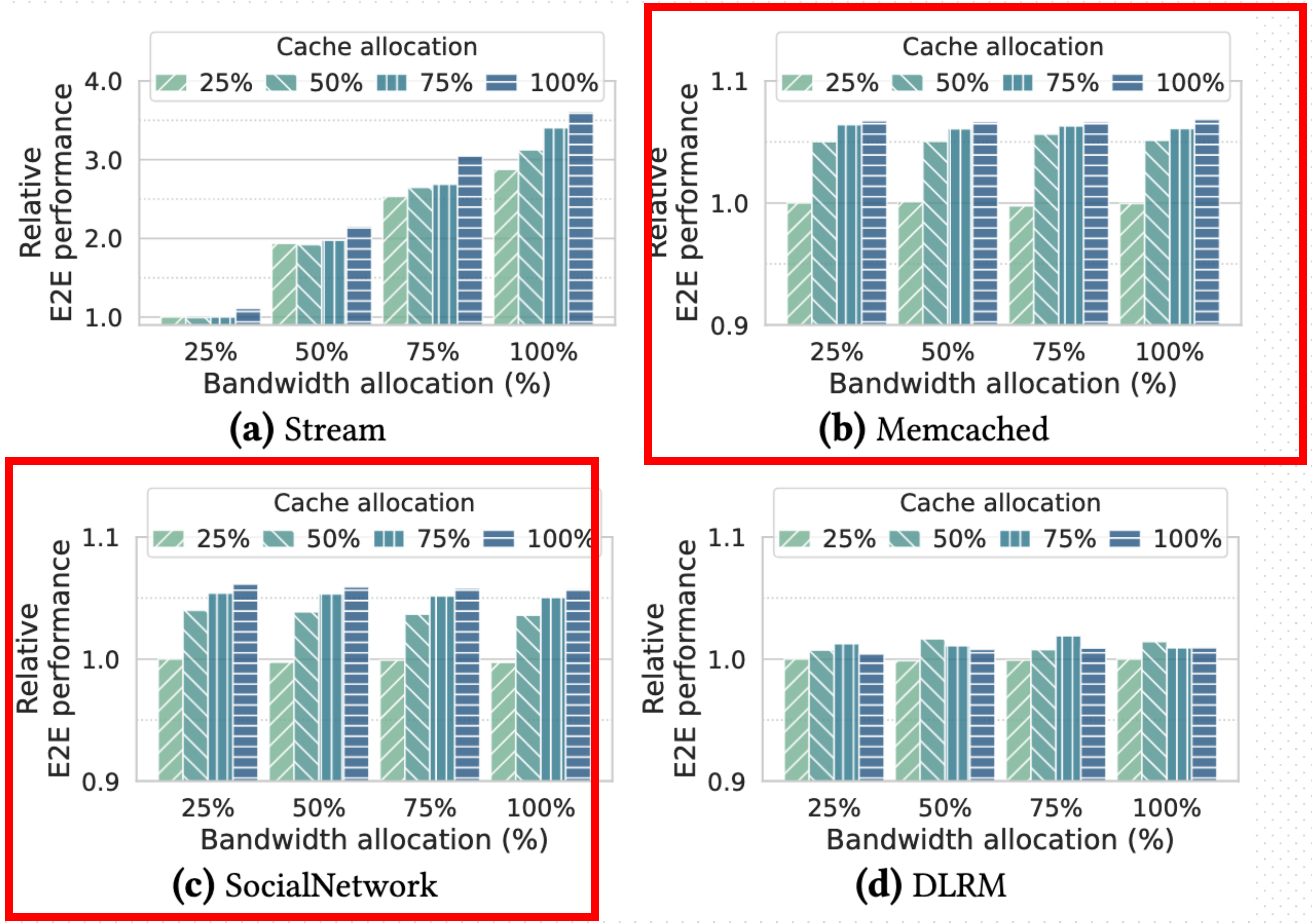
- Memcached (key-value store), dubbed STREAM (stream processing), DLRM (recommendation model), DeathStarBench(social network)



Shows performance variance of different cache size

Observation of various applications

- Memcached (key-value store), dubbed STREAM (stream processing), DLRM (recommendation model), DeathStarBench(SocialNetwork)

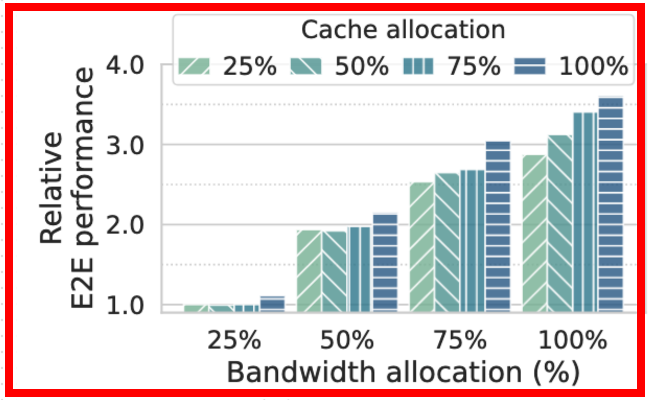


Memcached & SocialNetwork are cache-sensitive

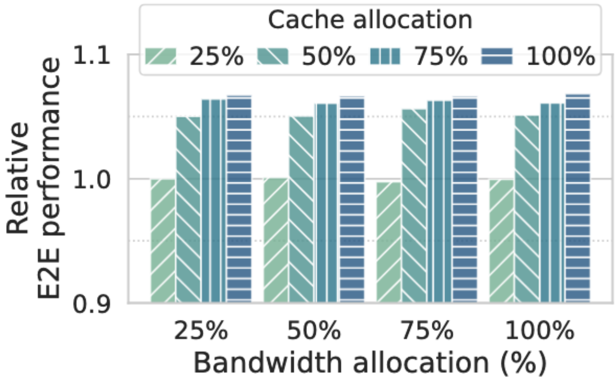
Observation of various applications

- Memcached (key-value store), dubbed STREAM (stream processing), DLRM (recommendation model), DeathStarBench(social network)

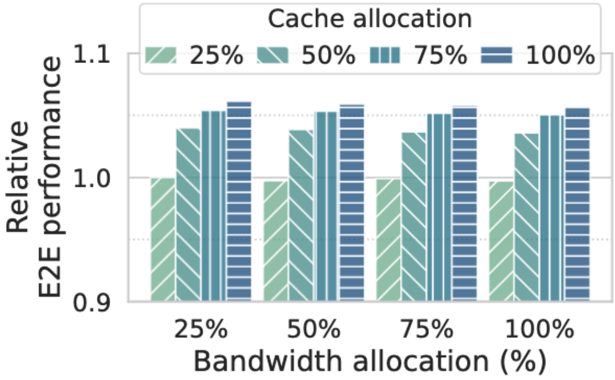
Bandwidth-sensitive



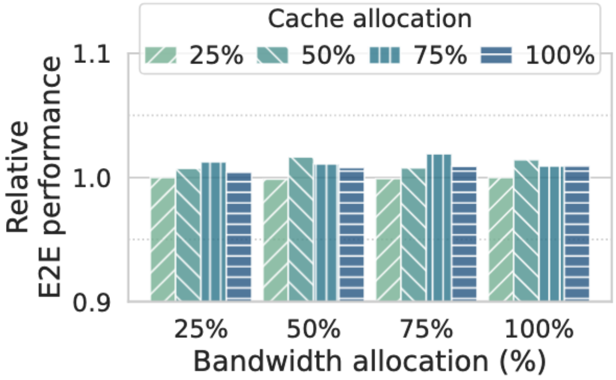
(a) Stream



(b) Memcached



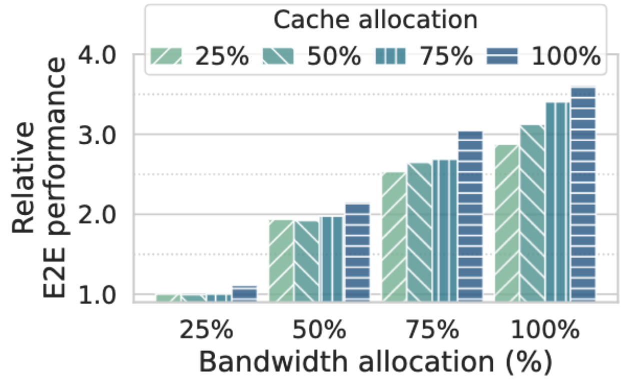
(c) SocialNetwork



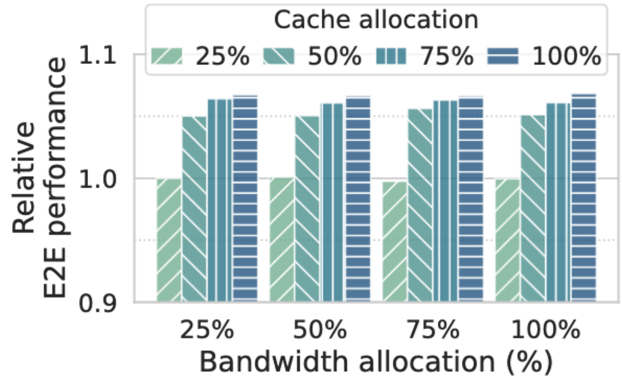
(d) DLRM

Observation of various applications

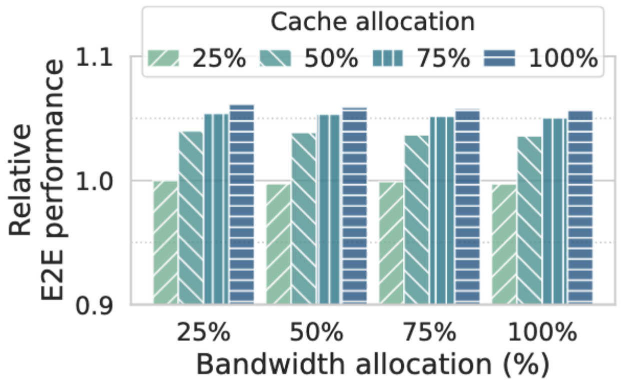
- Memcached (key-value store), dubbed STREAM (stream processing), DLRM (recommendation model), DeathStarBench(social network)



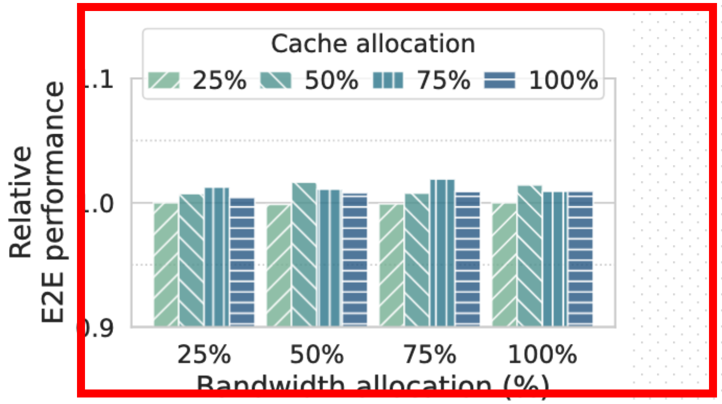
(a) Stream



(b) Memcached



(c) SocialNetwork



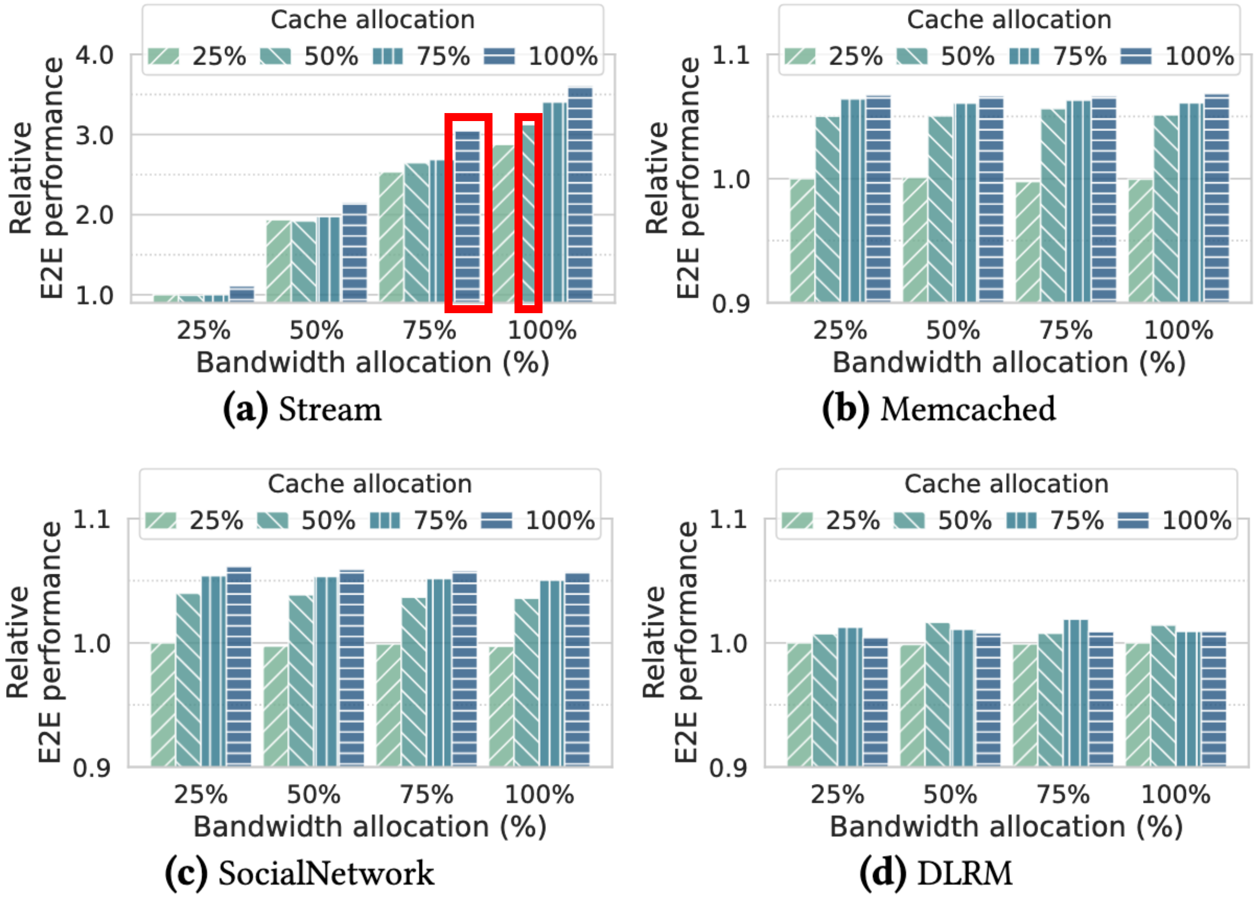
(d) DLRM

Compute-sensitive

Observation of various applications

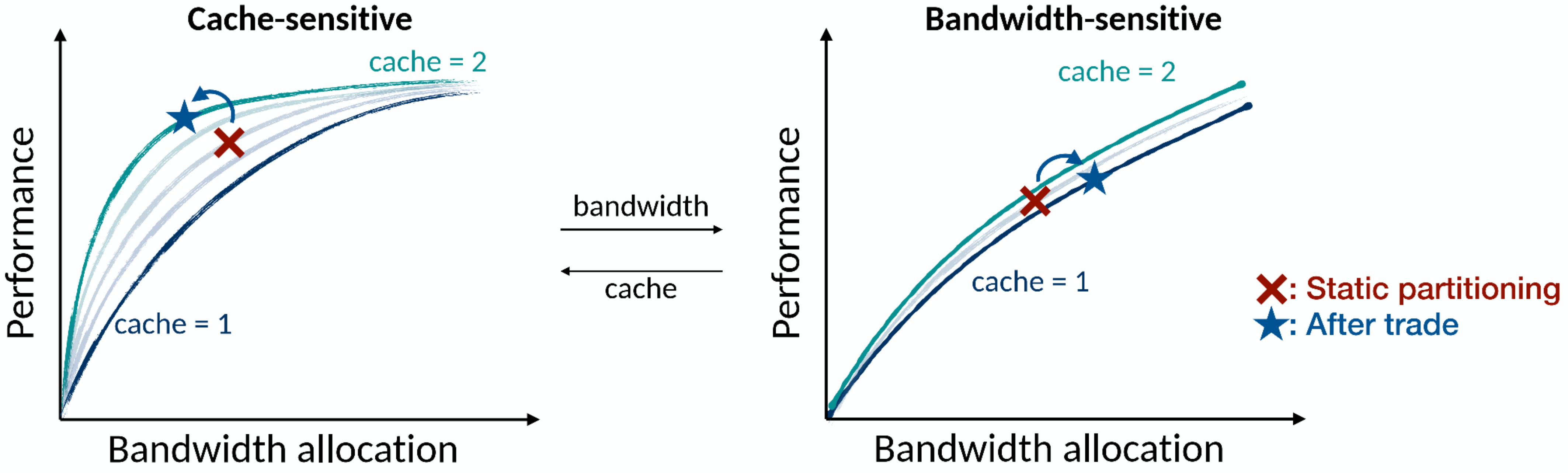
- Memcached (key-value store), dubbed STREAM (stream processing), DLRM (recommendation model), DeathStarBench(social network)

Almost same perfotmance



Key Ideas: Trading for Mutual Benefit

- Using the interdependence to achieve performance fairness



Outline

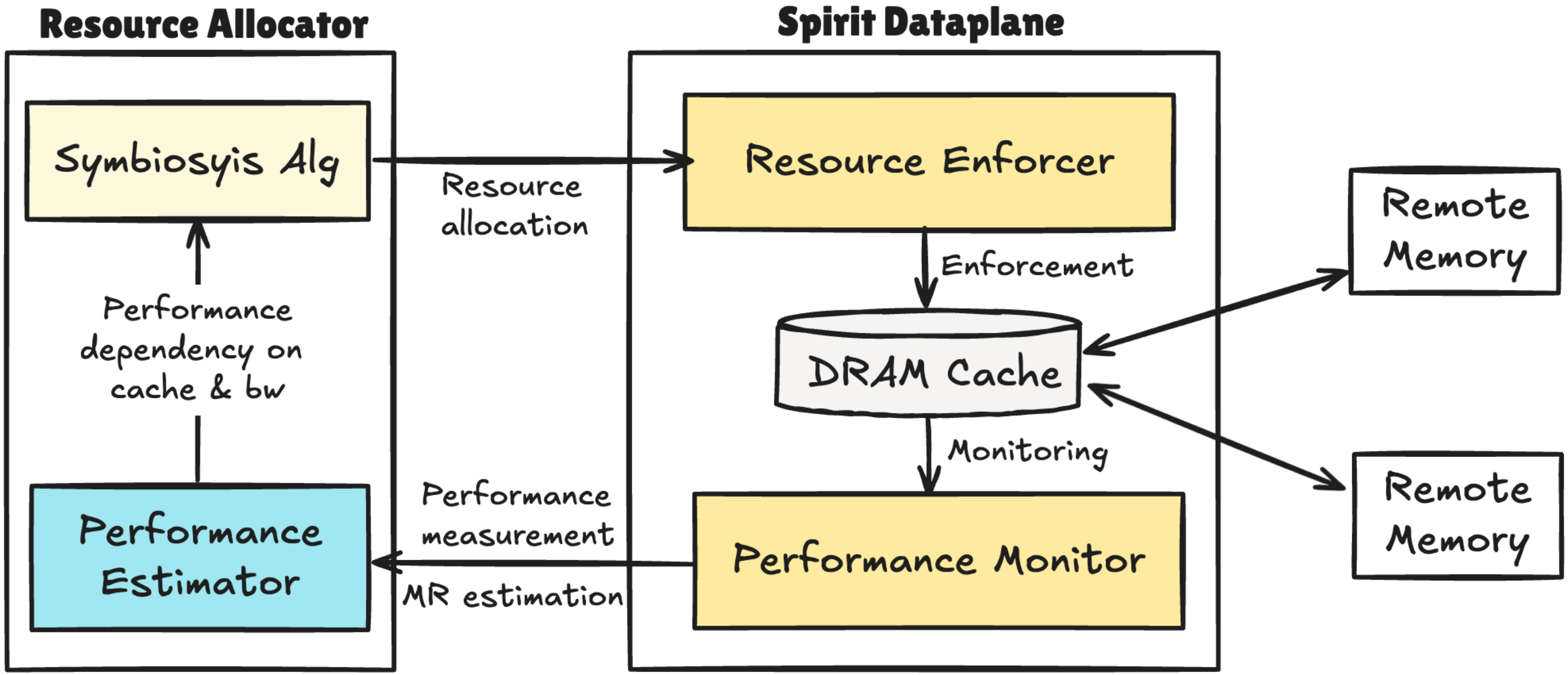
1 Background & Motivation

2 Design

3 Evaluation

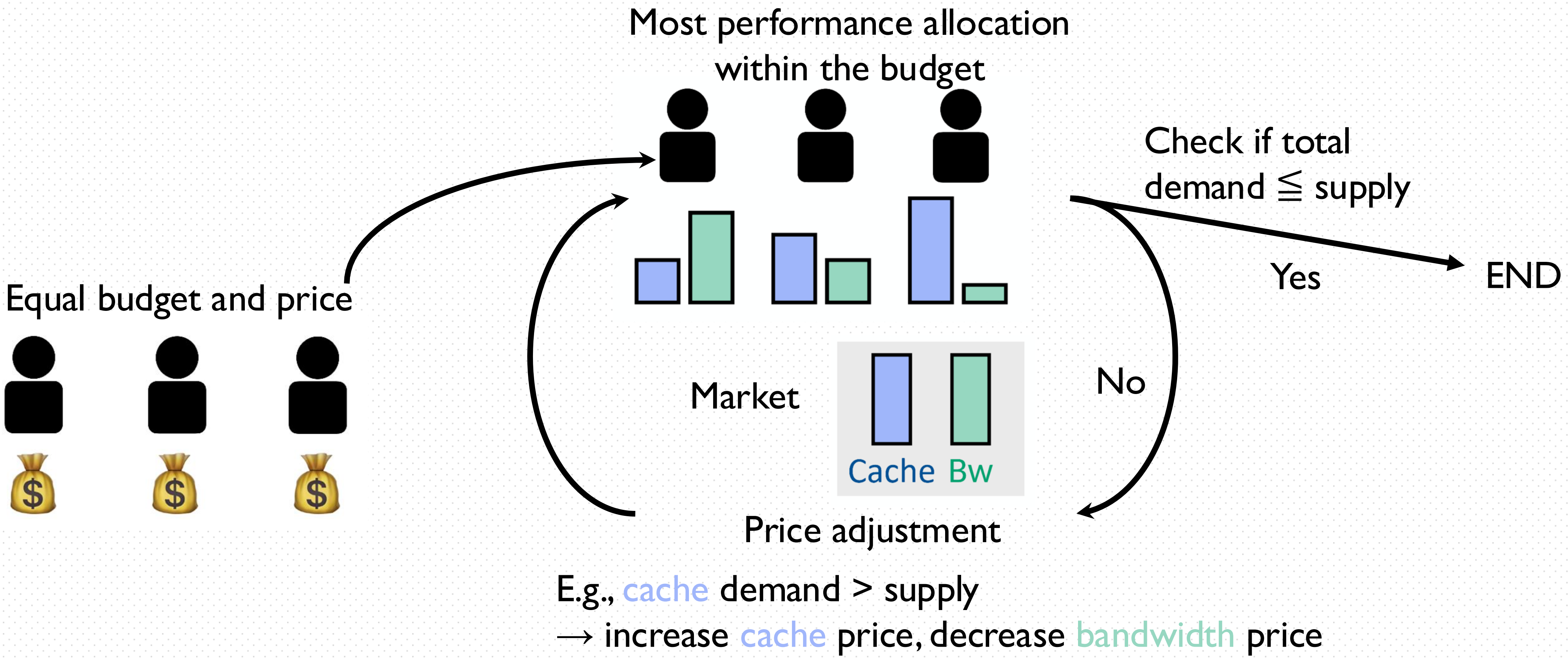
4 Discussion

Architecture



Symbiosis Algorithm

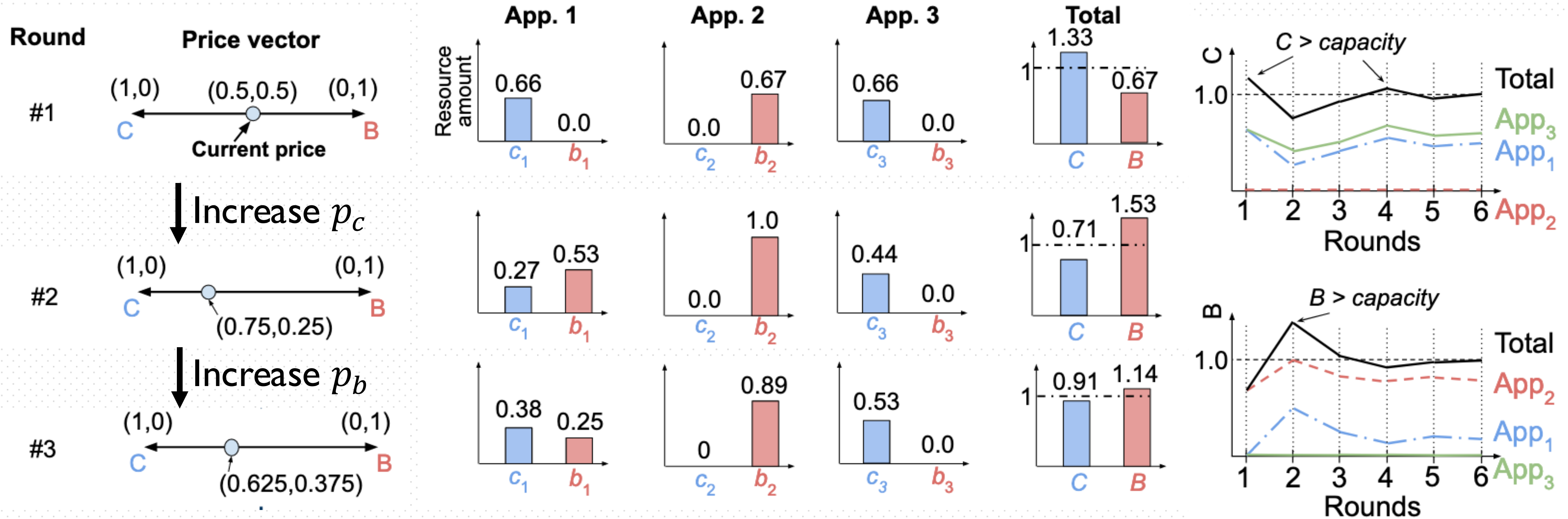
- ❑Borrow from auction-based allocation schemes from microeconomic theory



Symbiosis Algorithm: An Example

- Denoting c as cache size, and b as bandwidth, f is the performance function with c&b

A: $f_1(c,b) = \frac{1}{2}c^2 + cb$ B: $f_2(c,b) = b$ C: $f_3(c,b) = c$



How to estimate f_i?

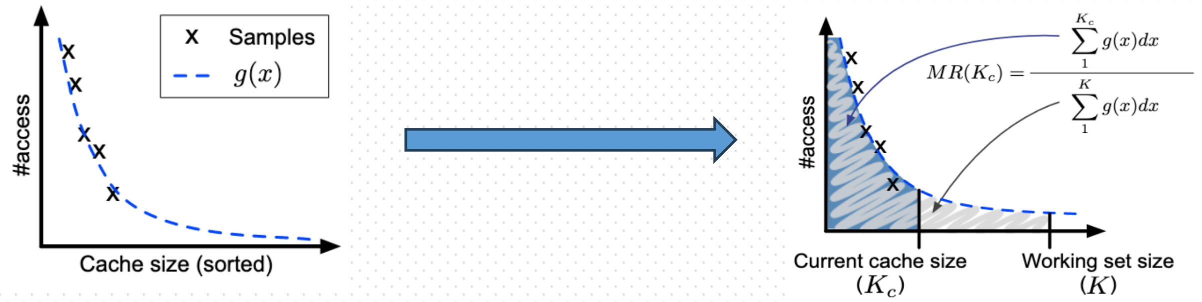
- ❑Using regression-based Miss Ratio Curve to estimate f_i
- ❑Directly computing MRC is infeasible
  - ❖First, supported hardware sampling methods simply do not provide enough samples
  - ❖ Second, computing incurs too high overheads to support any runtime estimations.

How to estimate f_i?

- Using regression-based Miss Ratio Curve to estimate f_i
- Leveraging the well-established power law relationship between cache entries (sorted by popularity) and their access frequencies

$$g(x) = \alpha_1(1 + x/\alpha_2)^{\alpha_3} + \alpha_4$$

Phase1: Using Intel PEBS sample and adjust params (α_2, α_3)



Phase2: Get accurate miss ratio to fine grained adjust params (α_1, α_4)

How to estimate f_i by MRC?

- Using MRC represents slowdown ratio of f_i, thus can estimate performance of (c_t, b_t) relative to the current configuration (c, b)

$$\frac{f(c_t, b_t)}{f(c, b)} = \frac{slowdown(c, b)}{slowdown(c_t, b_t)}$$

$$slowdown(c_t, b_t) = \boxed{1} + \boxed{MR(c_t)} \times \boxed{\gamma} \times \boxed{\max(1, \frac{b^{req}}{b_t})}$$

Full local cache size

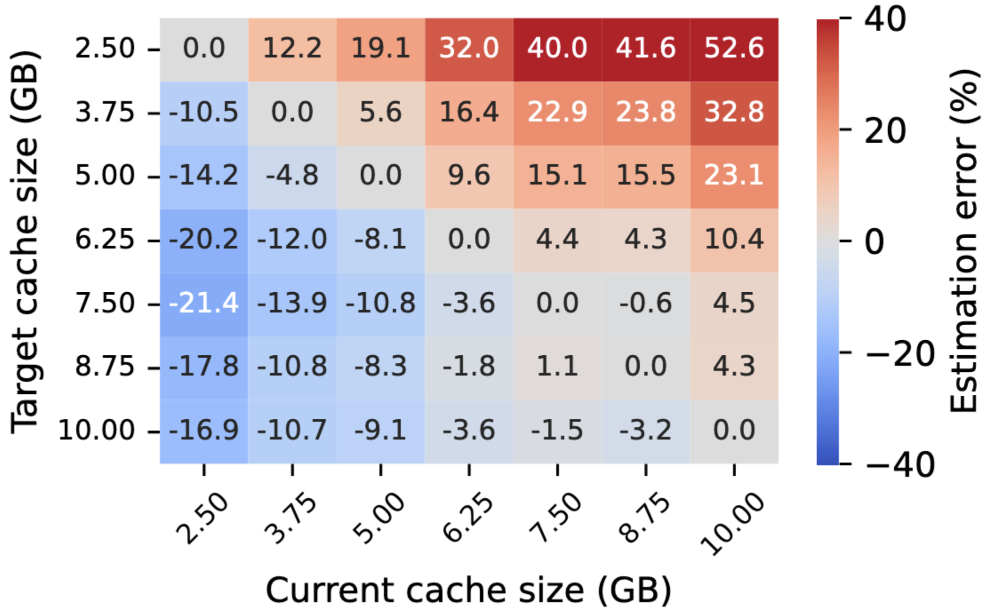
Cache miss ratio with c_t cache size

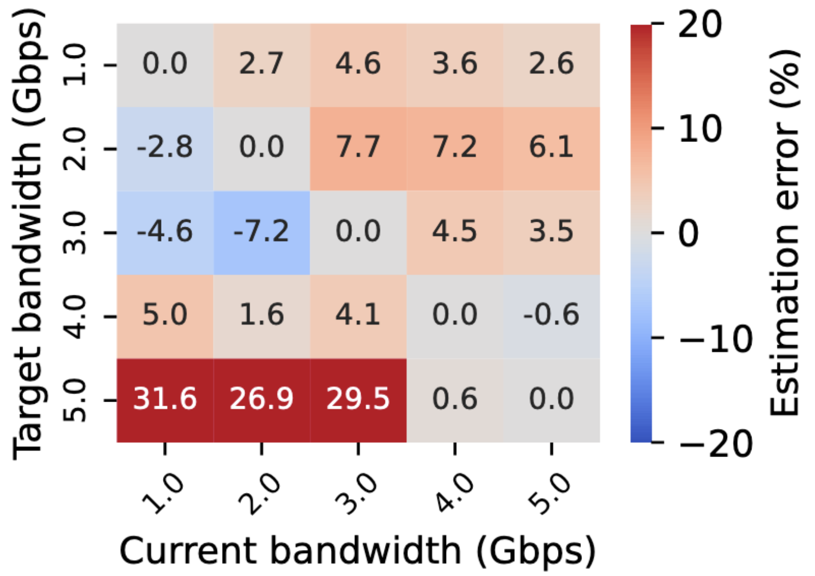
$$\frac{Speed_{Memory}}{Speed_{RDMA}}$$

App bandwidth when change from b to b_t

How to compute $arg\, max_{c,b} f(c,b)$?

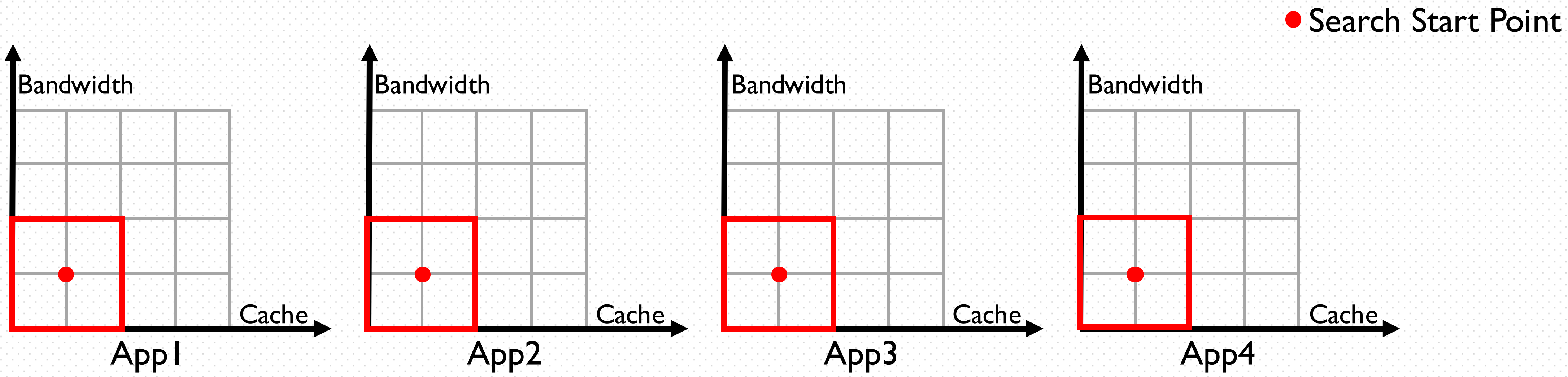
- Use polynomial-time approximation scheme to search $arg\, max_{c,b} f(c,b)$
  - Split the total cache and bandwidth into 200 equidistant discrete values (ϵ).
  - The search is restricted to a $\pm 5\epsilon$ vicinity of the current cache size, ensuring estimation errors remain under 10%





How to compute $arg\,max_{c,b} f(c,b)$?

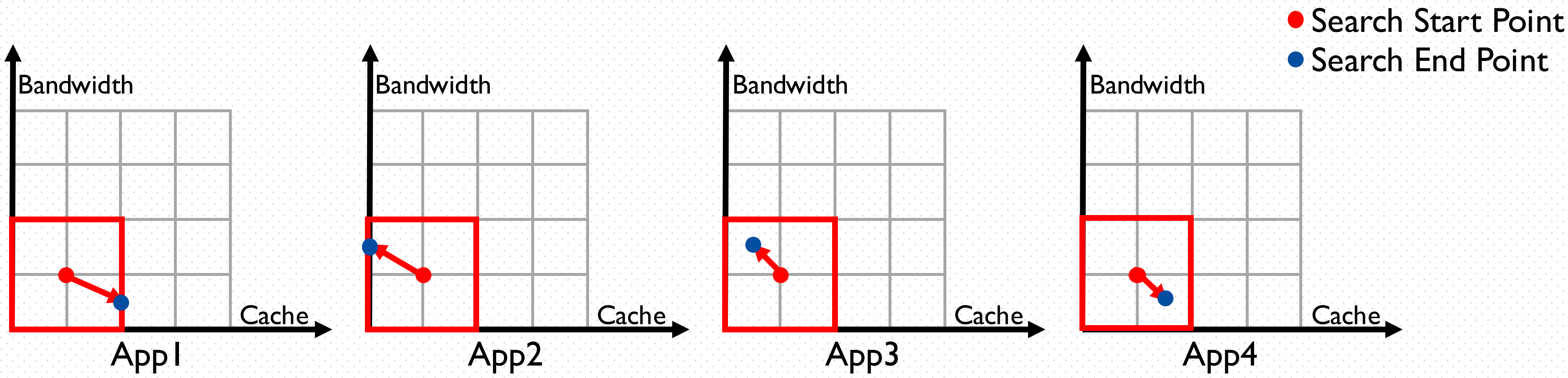
Symbiosis Allocation Round 1



Begin Symbiosis Round 1, and start point is resource fail allocation points

How to compute $arg\ max_{c,b} f(c, b)$?

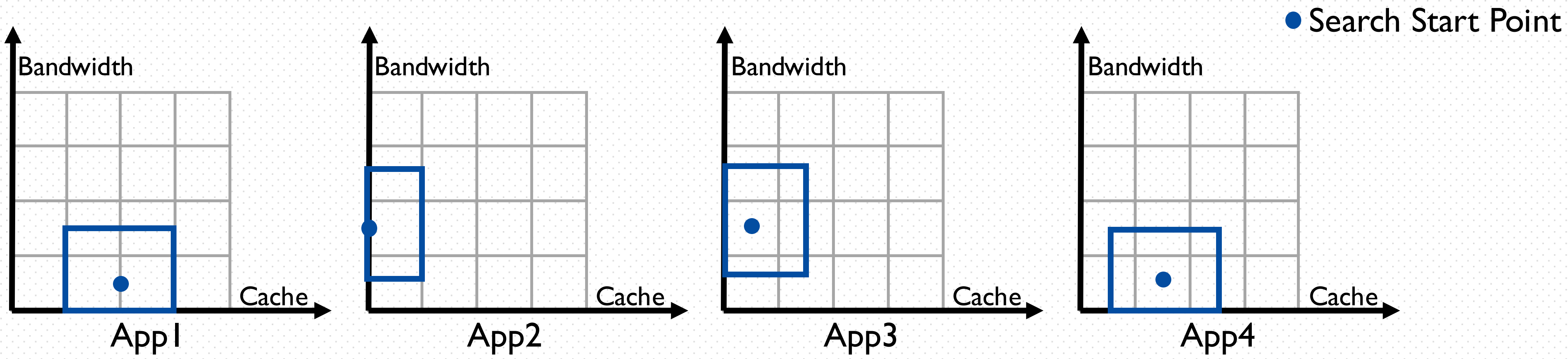
Symbiosis Allocation Round 1



Round 1 finish, allocate and wait for next Symbiosis round

How to compute $arg\,max_{c,b} f(c,b)$?

Symbiosis Allocation Round 2



Begin Round 2, and now start points change to last configuration

Spirit Data Plane

- ❑ Performance Monitoring
  - ❖ Samples LLC misses via Intel PEBS
- ❑ Enforcing Resource Allocations
  - ❖ Using `docker update` to manage containers' memory size
  - ❖ Setting `io.max` in `cgroups` to allocate bandwidth

Outline

Evaluation Setup

- Modeling AWS EC2 instance (m5a.8xlarge), Linux v6.113
  - 32 vCPUs, 128 GB memory, and 7.5 Gbps bandwidth
  - 10 to 20 GB as local memory (among 128 GB)

- Diverse applications & sensitivity
  - **STREAM** : sensitive to cache & bw
  - **Memcached** : sensitive to cache
  - **SocialNetwork** : sensitive to cache
  - **DLRM** : compute-intensive

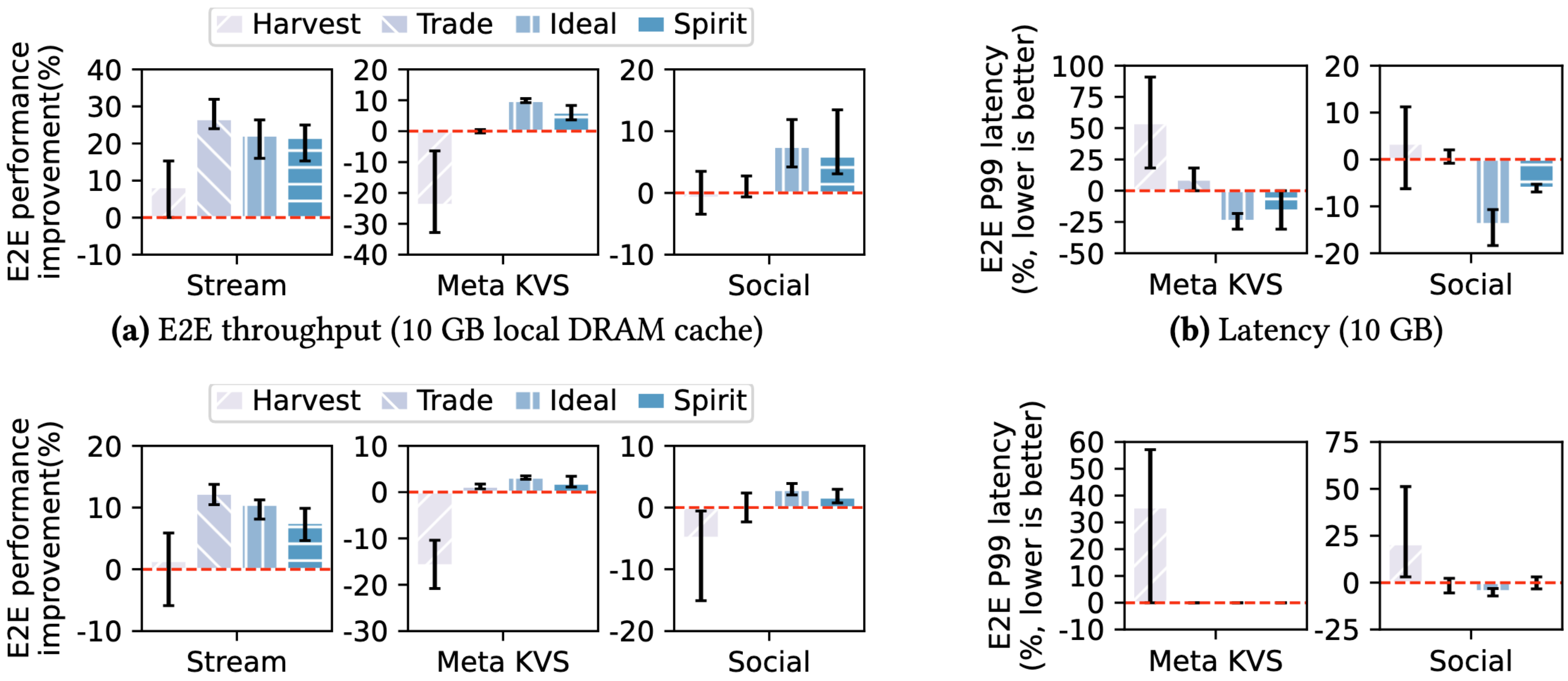
Evaluation Baselines

❑Compared schemes

- ❖**Baseline**： DRF, allocate same cache size and bandwidth for each app
- ❖**Harvest**: harvest from the most performant and reassign to the least (PARTIES, ASPLOS'19)
- ❖**Trade:** trade cache and bandwidth without using pricing (cache and bandwidth is fixed at 1-to-1)
- ❖**Ideal**: hand-picked best solution

End to End Result

- Deploy 24 app instances, six per server (3 STREAM, 1 Memcached, 1 DLRM and 1 SocialNetwork) across 4 servers.



(a) E2E throughput (10 GB local DRAM cache)

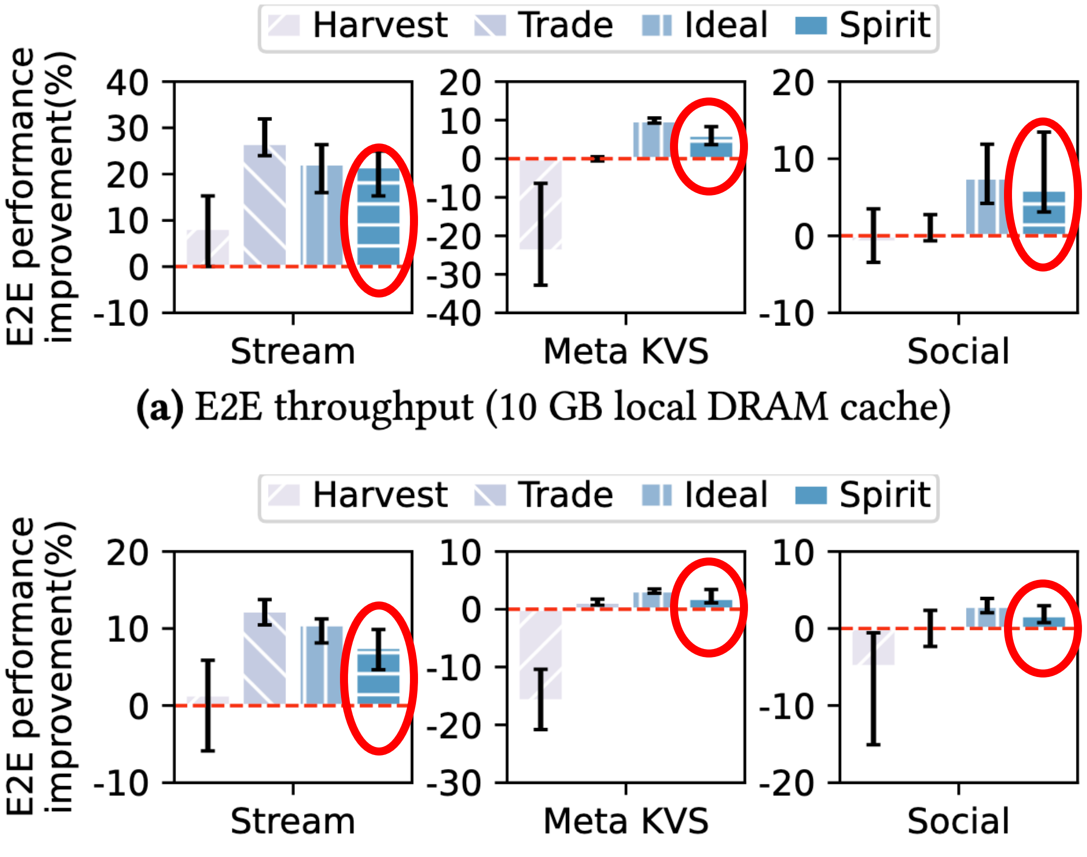
(b) Latency (10 GB)

(c) E2E throughput (20 GB local DRAM cache)

(d) Latency (20 GB)

End to End Result

- Deploy 24 app instances, six per server (3 STREAM, 1 Memcached, 1 DLRM and 1 SocialNetwork) across 4 servers.



(a) E2E throughput (10 GB local DRAM cache)

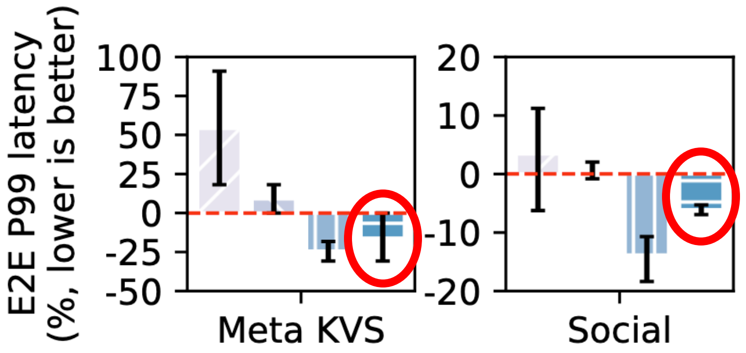
(c) E2E throughput (20 GB local DRAM cache)

End-to-end performance improves up to 21.6%, preserving fairness across applications

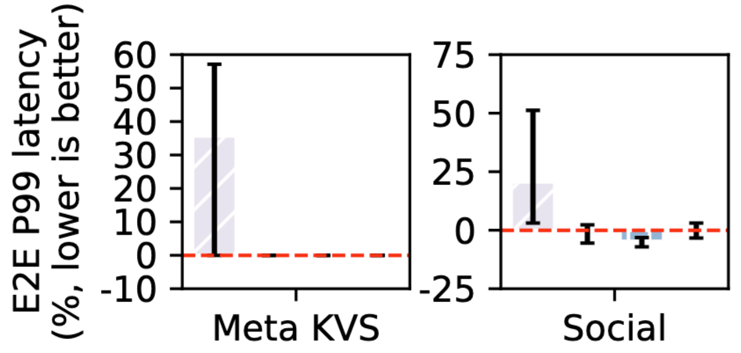
End to End Result

- ❑Deploy 24 app instances, six per server (3 STREAM, 1 Memcached, 1 DLRM and 1 SocialNetwork) across 4 servers.

Spirit reduces P99 latency by up to 16.8%



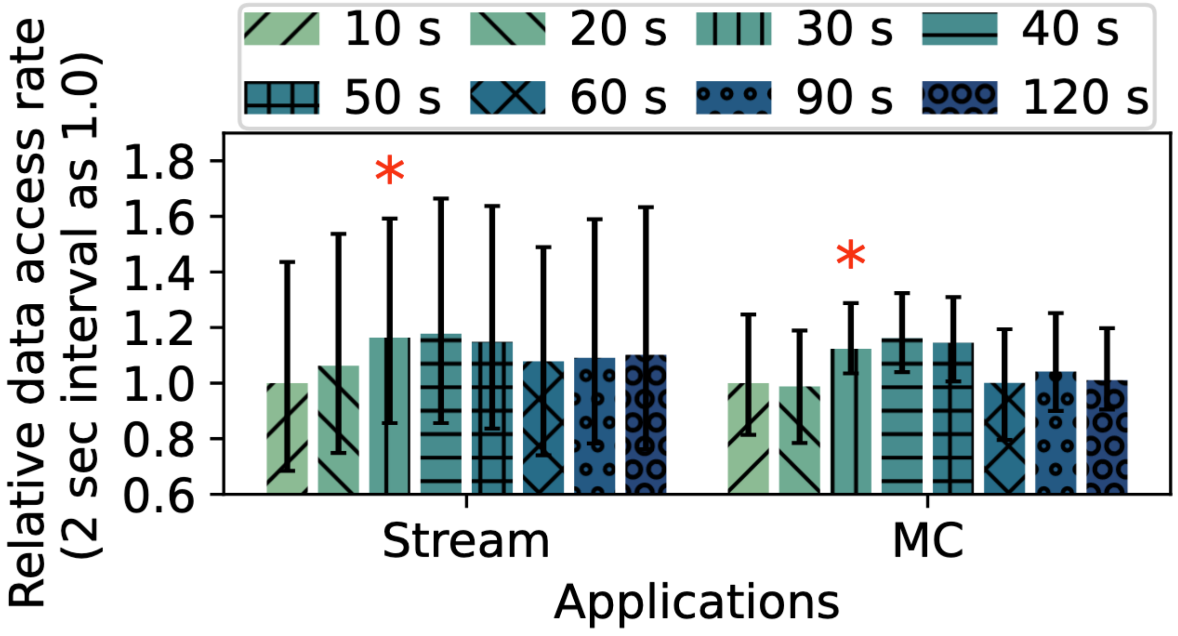
(b) Latency (10 GB)



(d) Latency (20 GB)

Sensitivity and Overhead Analysis

- Focus on Stream and Memcached to analyze sensitivity of epoch size (time between two allocations via Symbiosis)



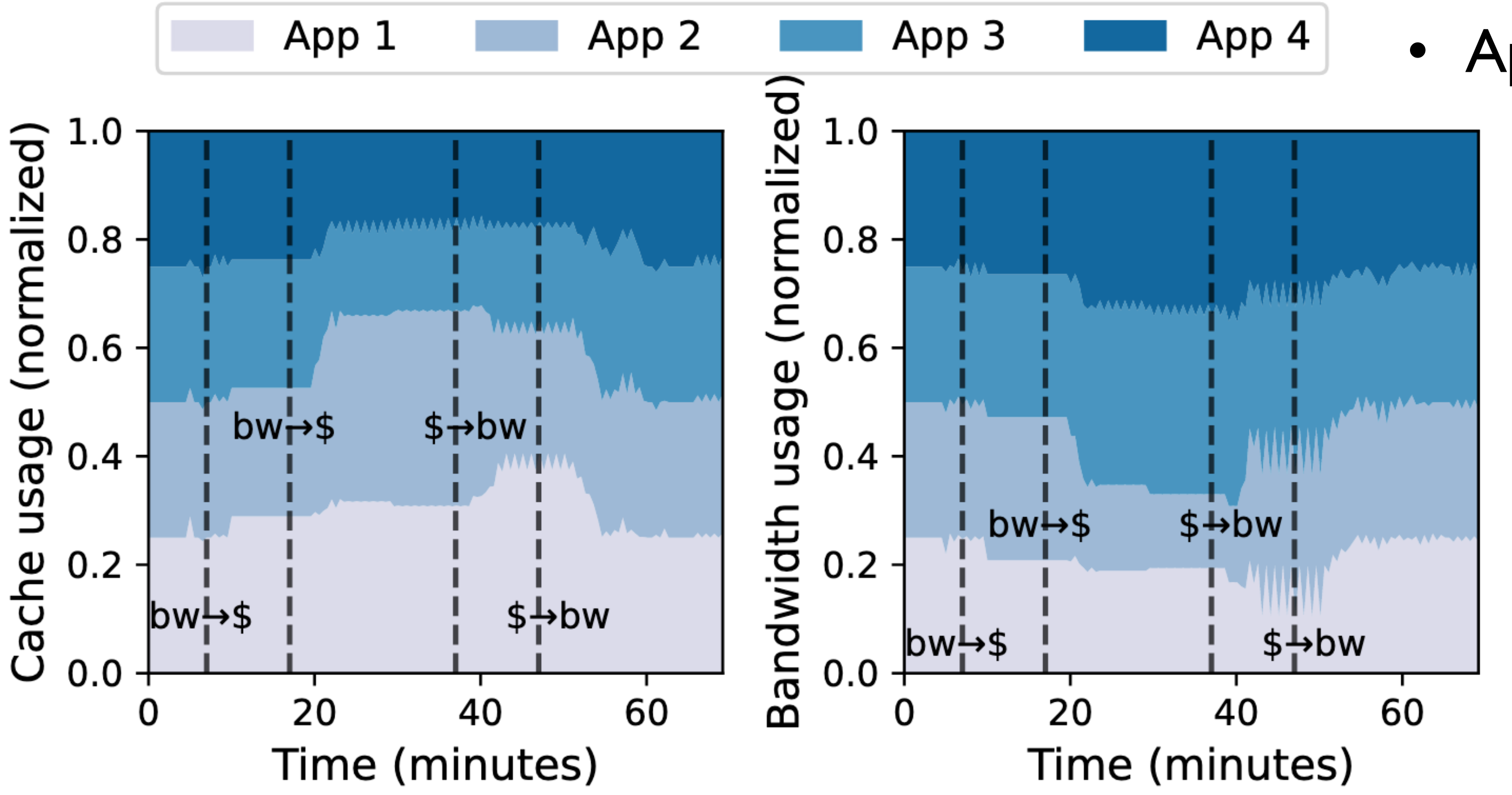
Too small, results performance degradation due to reallocation overhead

Too large, Symbiosis does not react fast enough

Adapting to dynamic

- How Symbiosis adapts to runtime changes in f_i for participating applications?

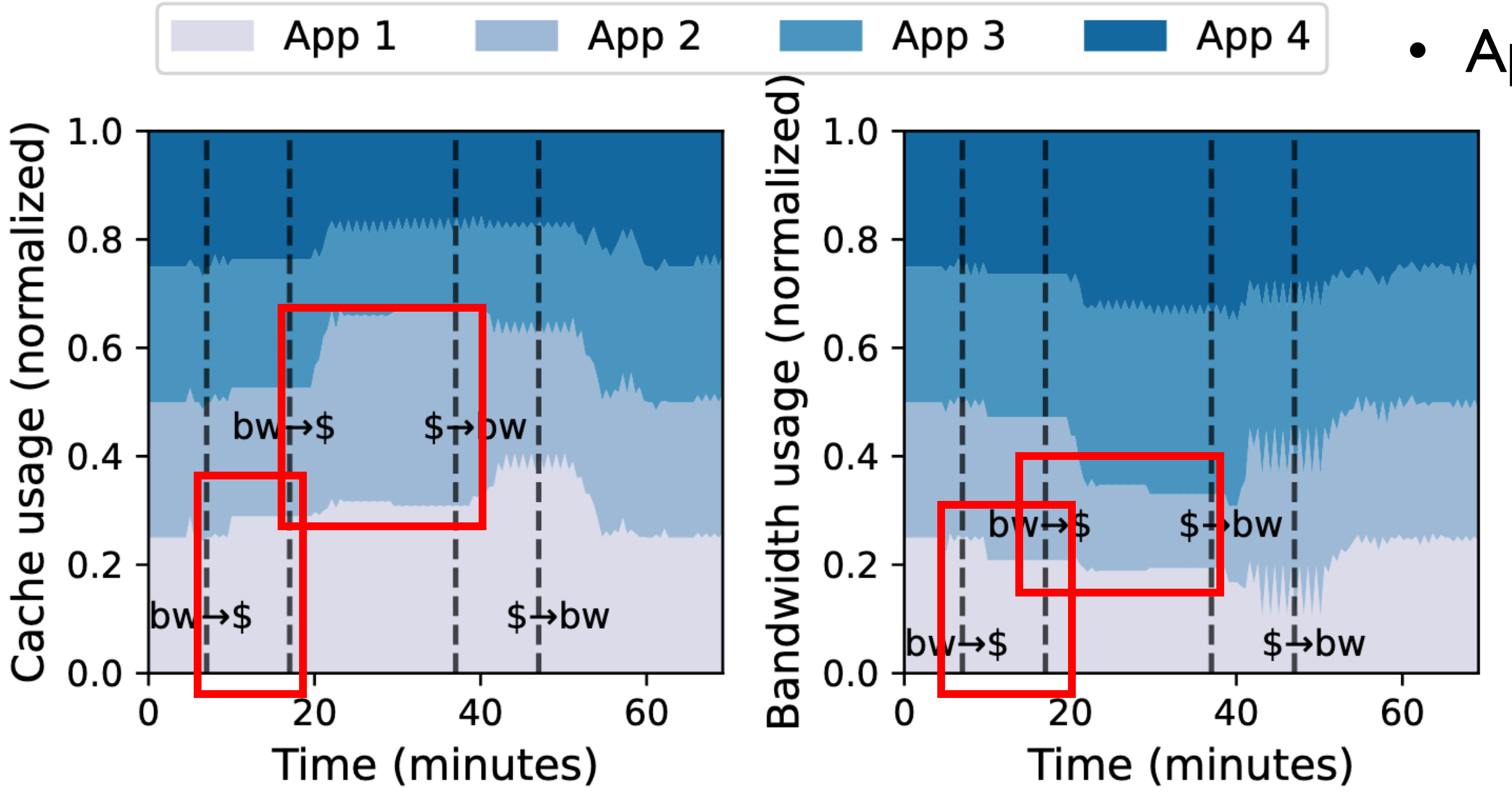
- Apps 1&2: Stream ⇔ Memcached
- Apps 3&4: Stream



\$: cache sensitive
bw: bandwidth sensitive

Adapting to dynamic

- How Symbiosis adapts to runtime changes in f_i for participating applications?



- Apps 1&2: Stream ⇔ Memcached
- Apps 3&4: Stream

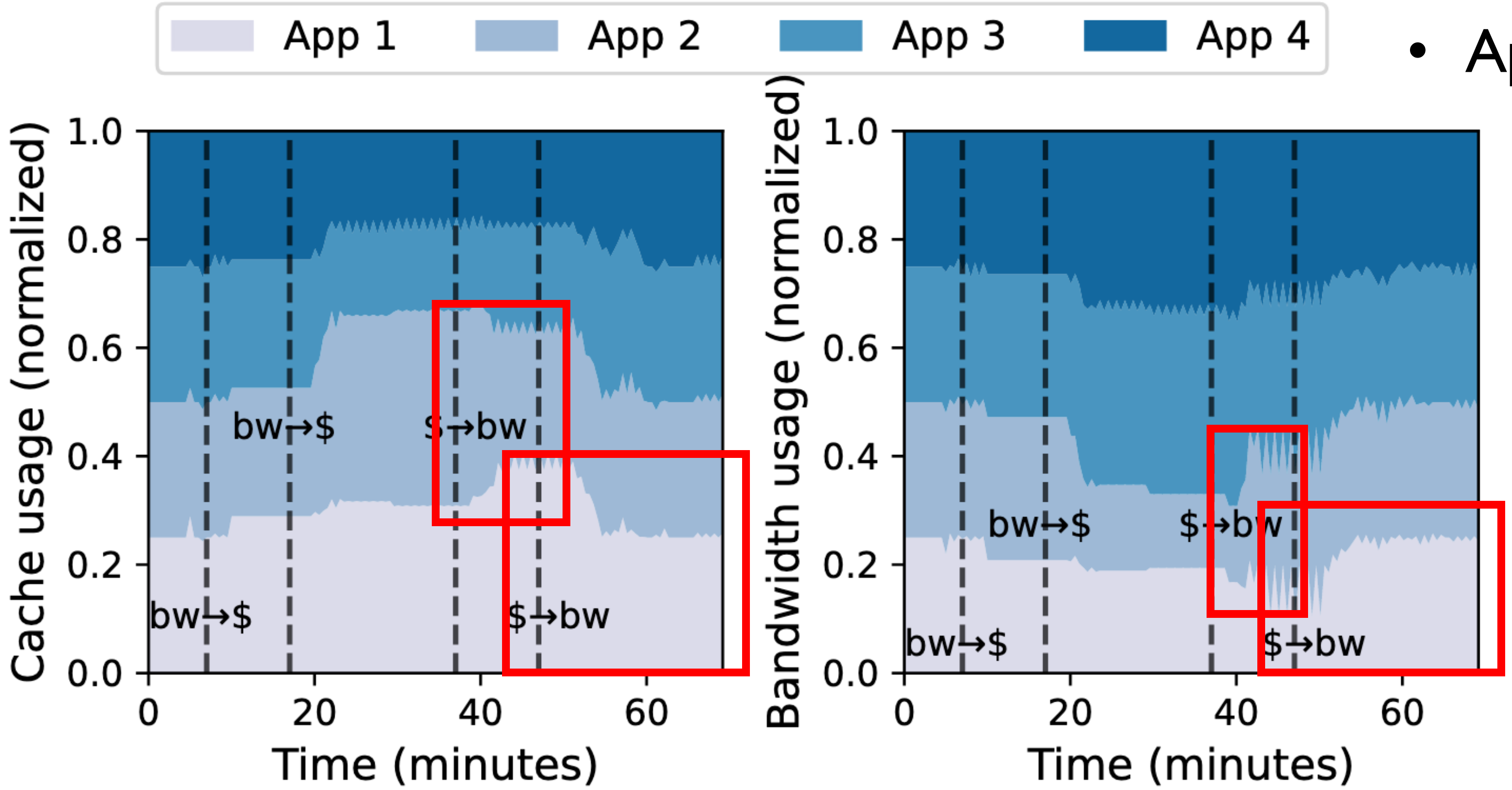
$: cache sensitive
bw: bandwidth sensitive

Allocate more cache and less bw

Adapting to dynamic

❑How Symbiosis adapts to runtime changes in f_i for participating applications?

- Apps 1&2: Stream ⇔ Memcached
- Apps 3&4: Stream



$: cache sensitive
bw: bandwidth sensitive

Allocate more bw and less cache

Outline

1. Background & Motivation
2. Design
3. Evaluation
4. Discussion

Conclusion & Discussion

- Performance fair resource allocation for remote memory systems
- Address the interdependence of cache size and bandwidth

- Do new perspectives emerge when dealing with heterogeneous media (e.g. CXL, NVMe SSDs), or in domains like MLSys where both GPU memory and communication are critical bottlenecks?

Thanks!