AIFM: High-Performance, Application-Integrated Far Memory

Zain (Zhenyuan) Ruan[*] Malte Schwarzkopf[†] Marcos K. Aguilera[‡] Adam Belay[*]

[*]MIT CSAIL [†]Brown University [‡]VMware Research

In-Memory Applications

Data Analytics

Web Caching

Database

Graph Processing

Memory Is Inelastic

- Limited by the server physical boundary.

Memory Is Inelastic

- Limited by the server physical boundary.

➢Applications cannot overcommit memory.

Opening a 20GB file for analysis with pandas

Asked 2 years, 8 months ago Active 1 year, 4 months ago Viewed 81k times

I am currently trying to open a file with pandas and python for machine learning purposes it would be ideal for me to have them all in a DataFrame. My RAM is 32 GB. I keep getting memory errors.

Memory Is Inelastic

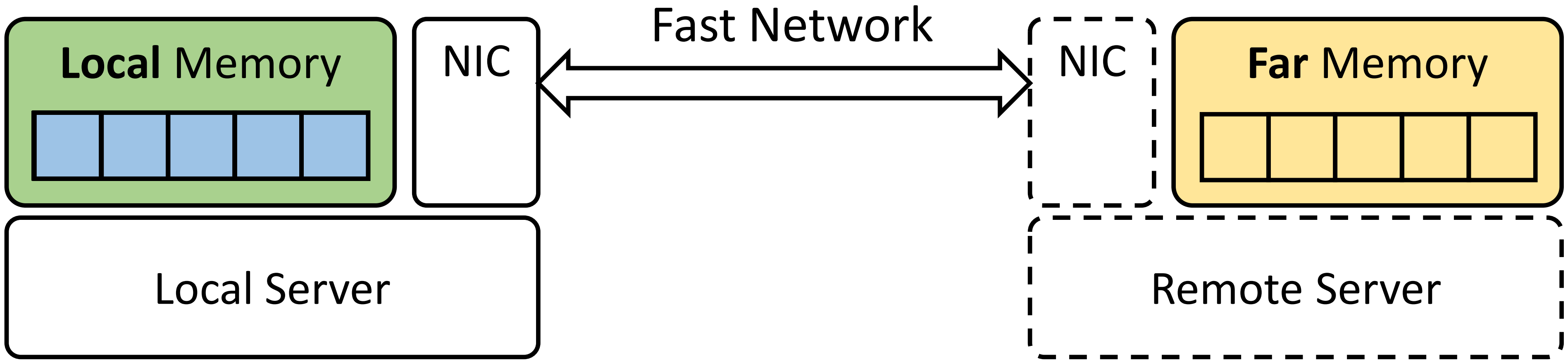
- Limited by the server physical boundary.
- Applications cannot overcommit memory.

> **Opening a 20GB file for analysis with pandas**
>
> Asked 2 years, 8 months ago Active 1 year, 4 months ago Viewed 81k times
>
> 30
>
> I am currently trying to open a file with pandas and python for machine learning purposes it would be ideal for me to have them all in a DataFrame. My RAM is 32 GB. I keep getting memory errors.

➢ Expensive solution: overprovision memory for peak usage.

Trending Solution: Far Memory

➢Leverage the idle memory of remote servers (with fast network).

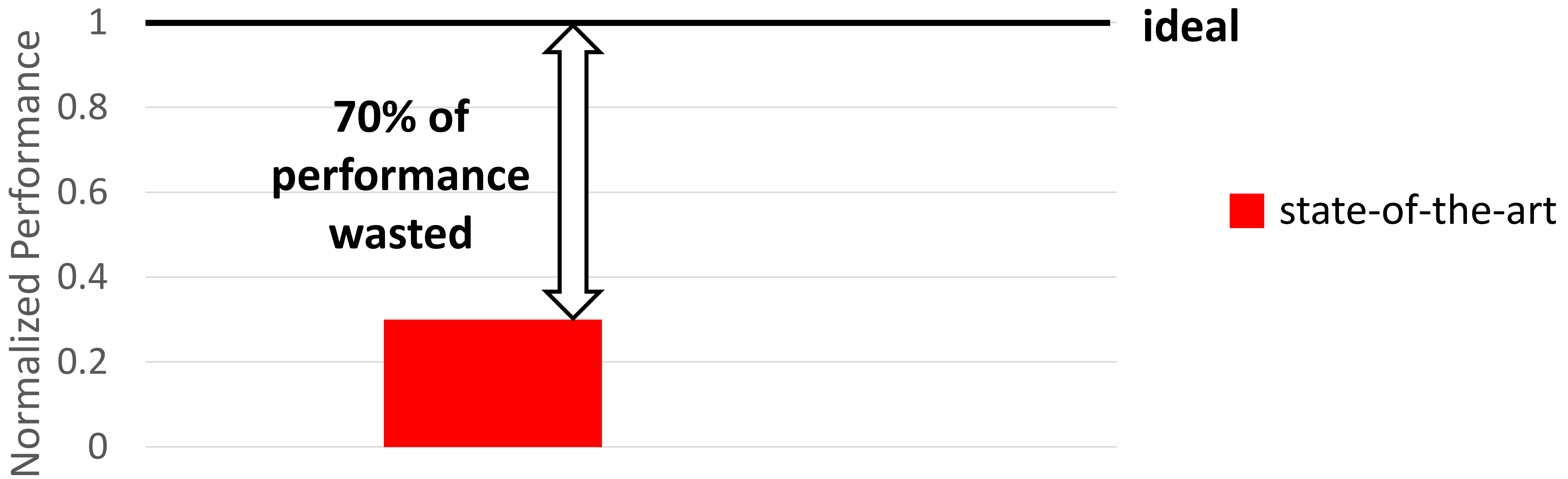


Existing Far-Memory Systems Perform Poorly

- Real-world Data Analytics from Kaggle.

Existing Far-Memory Systems Perform Poorly

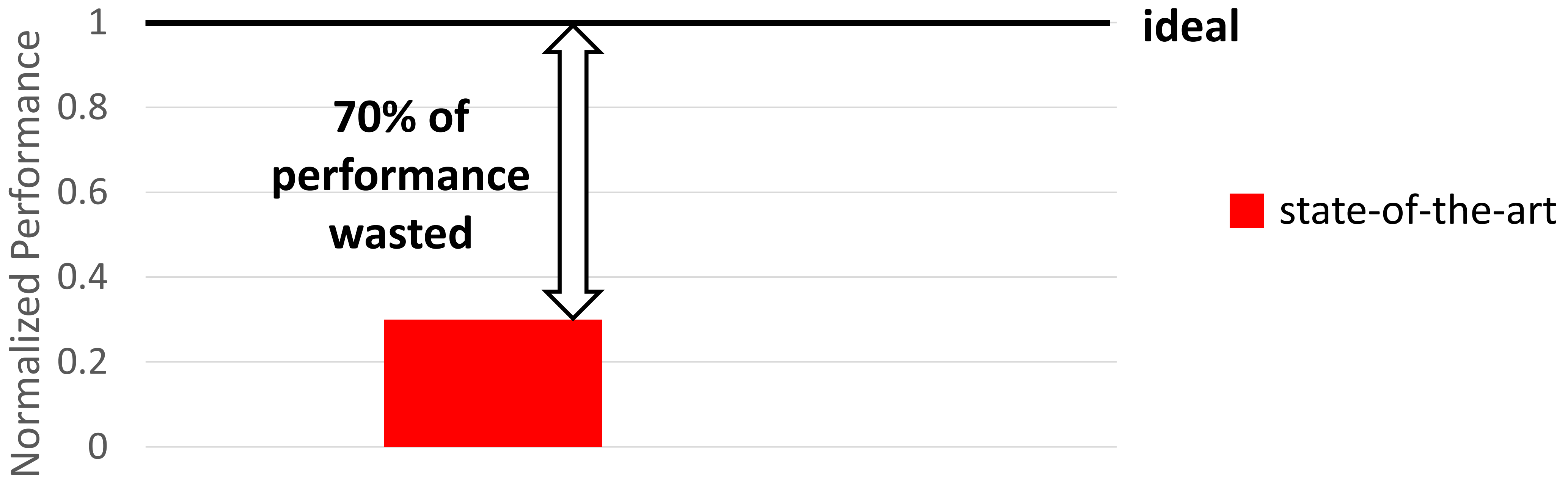
- Real-world Data Analytics from Kaggle.
  - ➢Provision **25%** of working set in local mem



Existing Far-Memory Systems Perform Poorly

- Real-world Data Analytics from Kaggle.
  - Provision **25%** of working set in local mem.

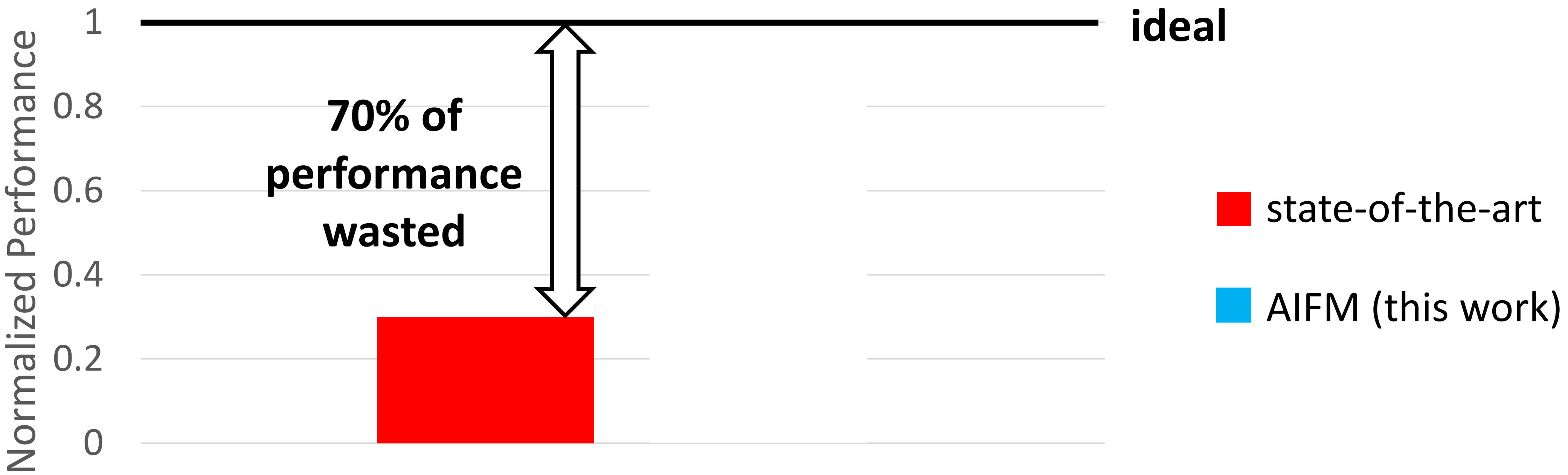
➢Goal: reclaim the wasted performance.



Existing Far-Memory Systems Perform Poorly

- Real-world Data Analytics from Kaggle.
  - Provision **25%** of working set in local mem.

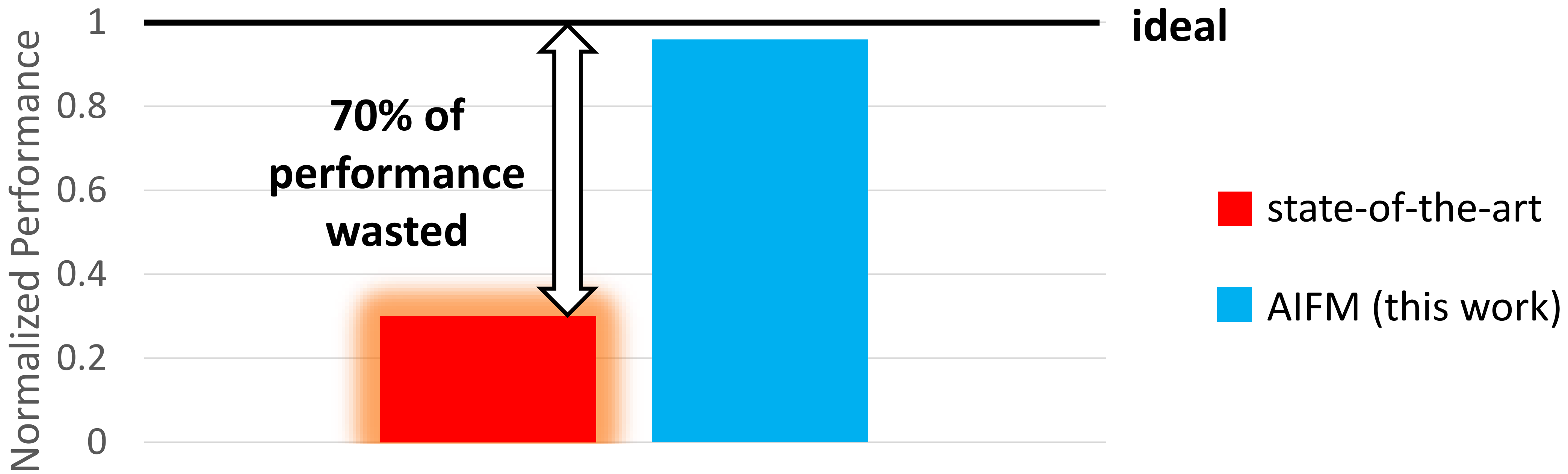
➢Goal: reclaim the wasted performance.



Existing Far-Memory Systems Perform Poorly

- Real-world Data Analytics from Kaggle.
  - Provision **25%** of working set in local mem.

➢Goal: reclaim the wasted performance.

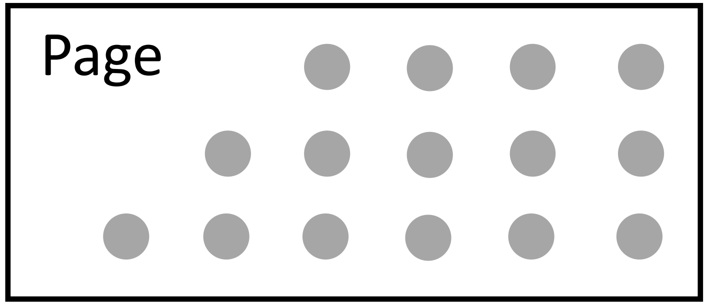


Why Do Existing Systems Waste Performance?

- Problem: based on **OS paging**.
  - Semantic gap.
  - High kernel overheads.

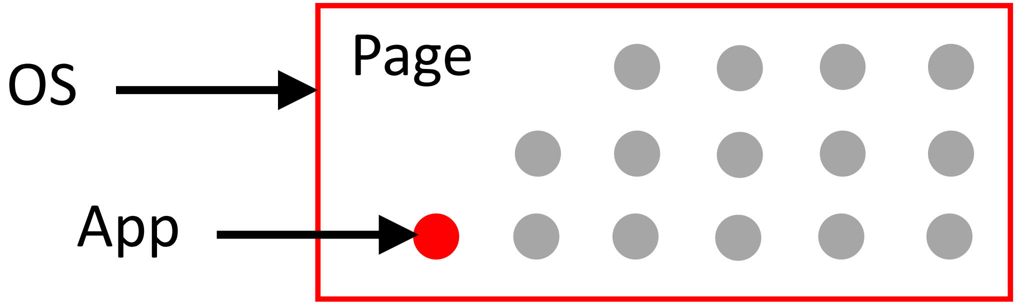
Challenge 1: Semantic Gap

- ➢Page granularity → **R/W amplification.**



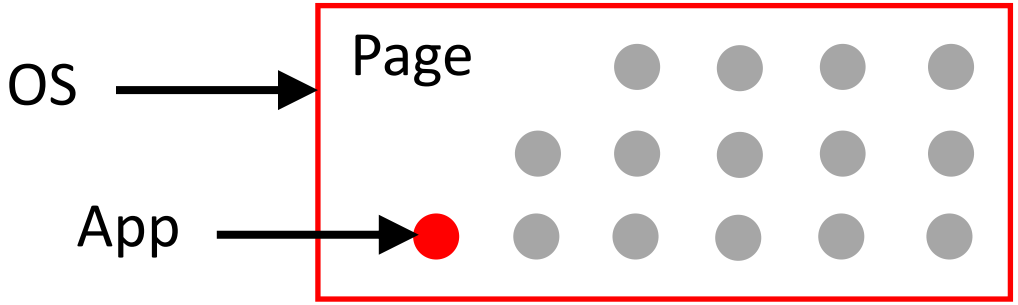
Challenge 1: Semantic Gap

- ➢Page granularity → **R/W amplification.**



Challenge 1: Semantic Gap

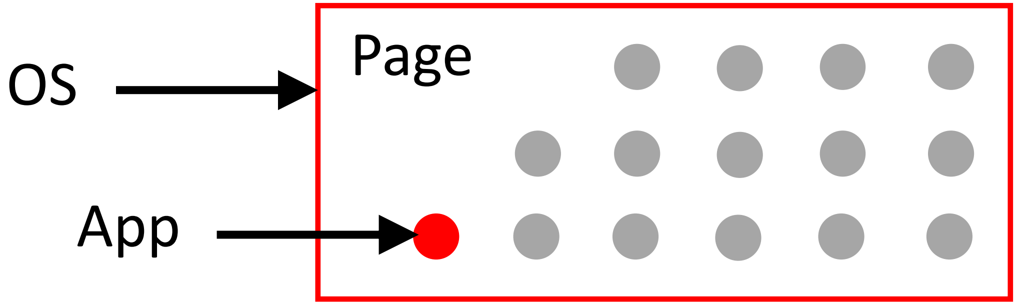
- Page granularity → **R/W amplification.**



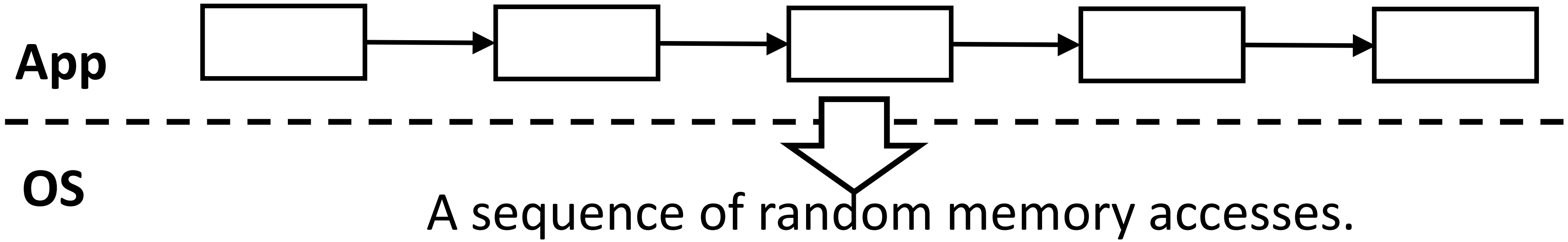
➢OS lacks app knowledge → **hard to prefetch, etc.**

Challenge 1: Semantic Gap

- Page granularity → **R/W amplification.**



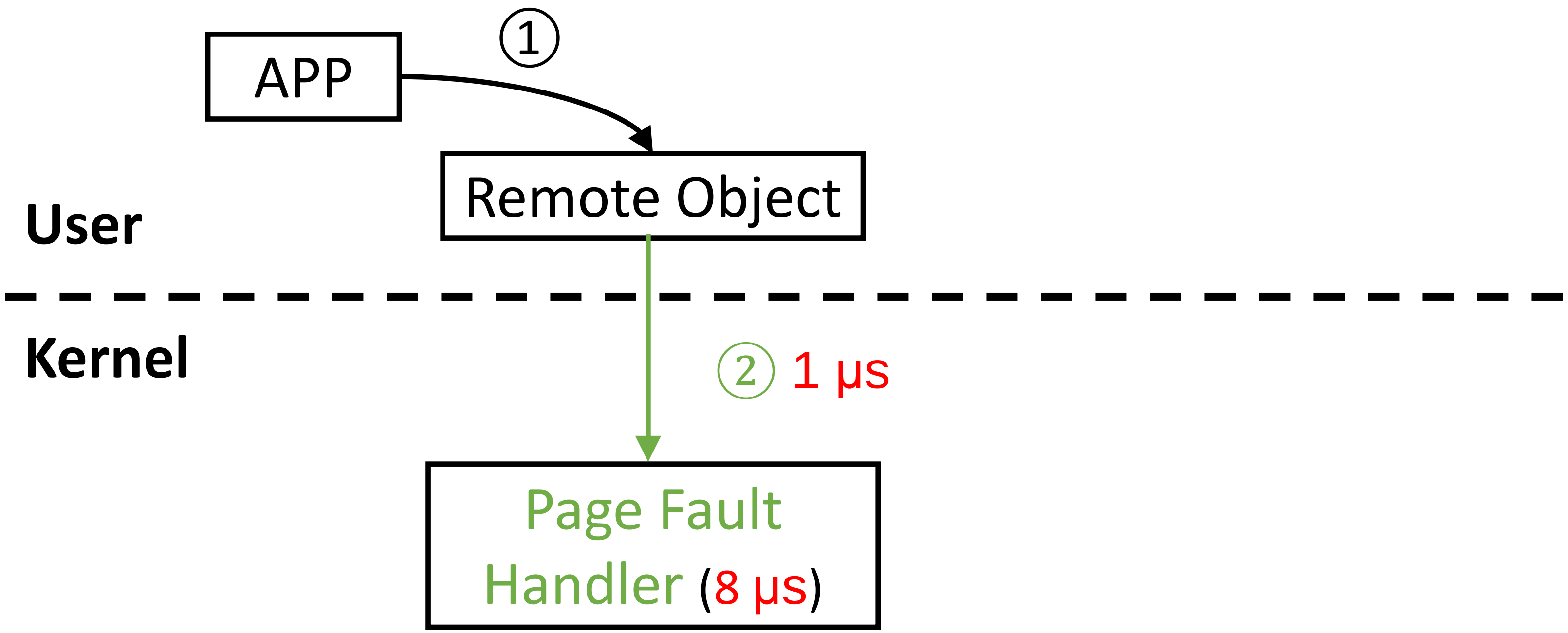
➢OS lacks app knowledge → **hard to prefetch, etc.**



Challenge 2: High Kernel Overheads

Challenge 2: High Kernel Overheads

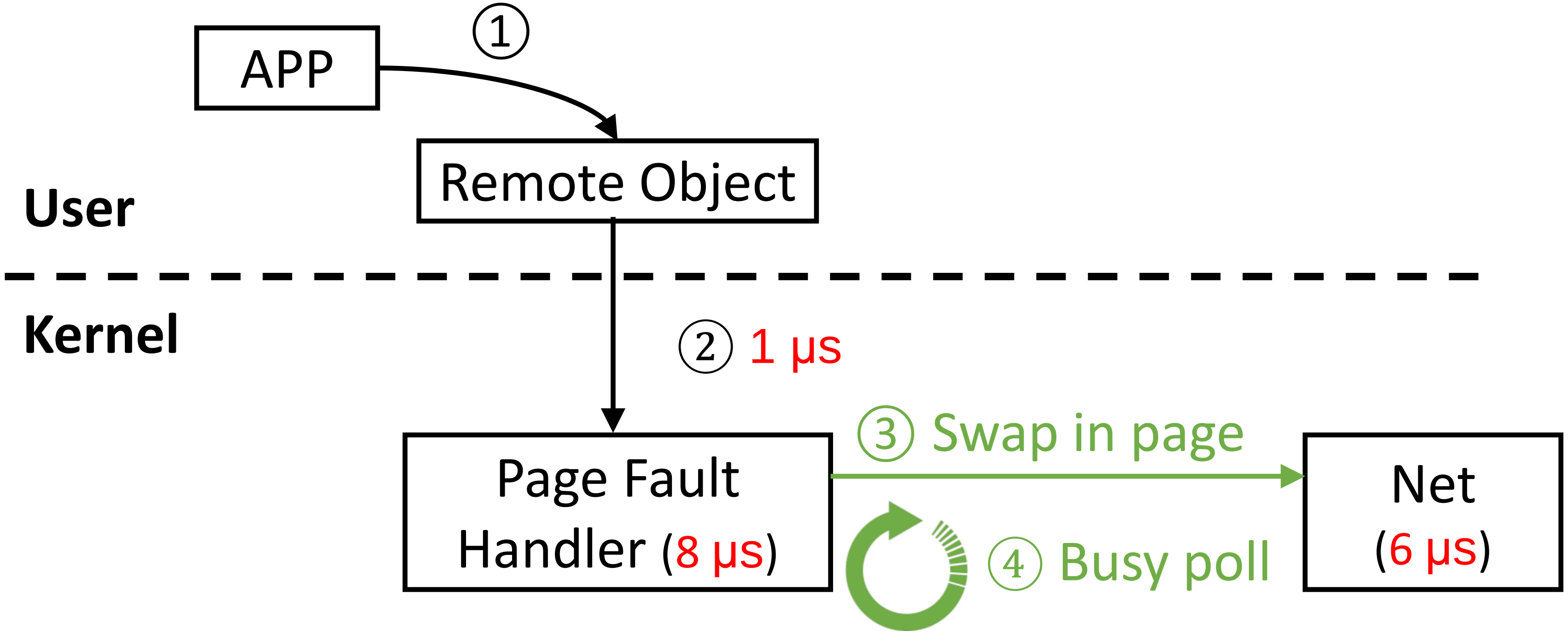
➢ **Expensive page faults.**



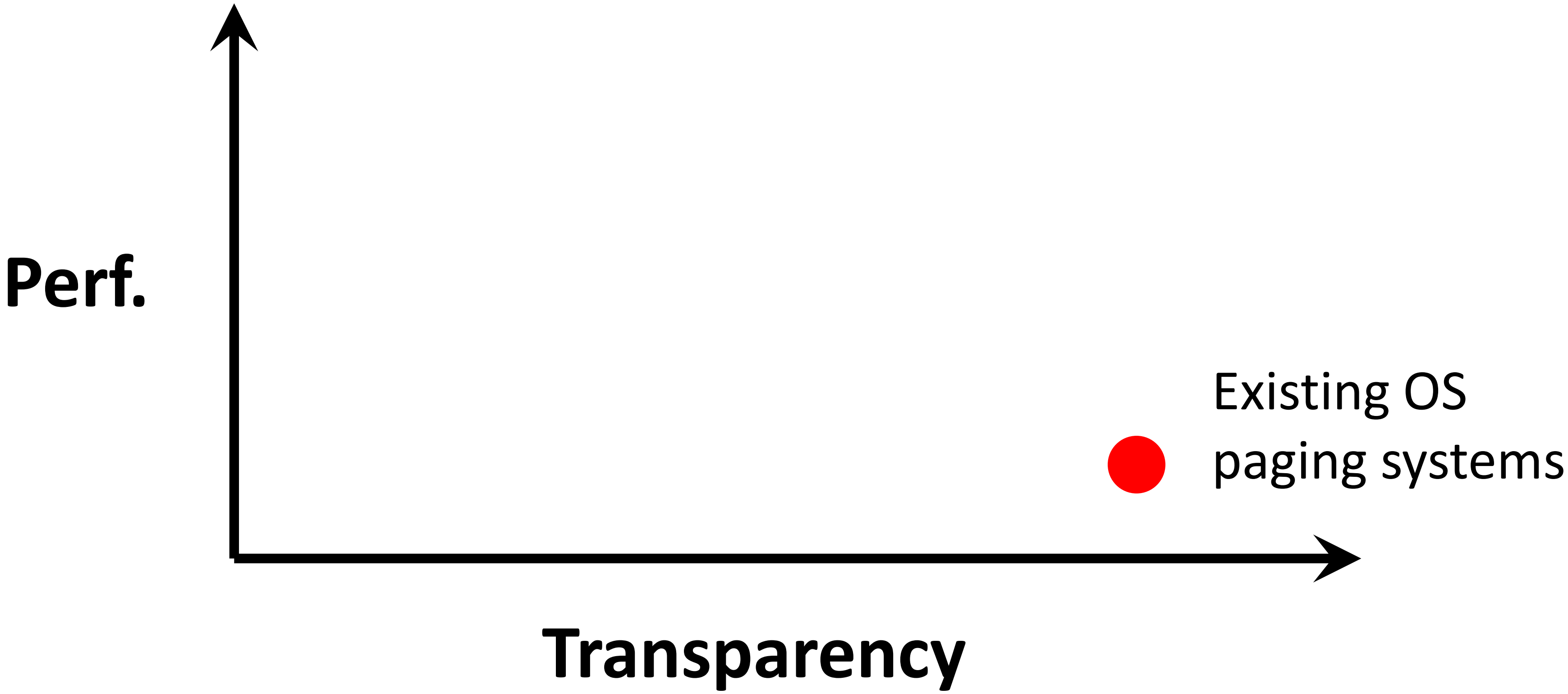
Challenge 2: High Kernel Overheads

- **Expensive page faults.**

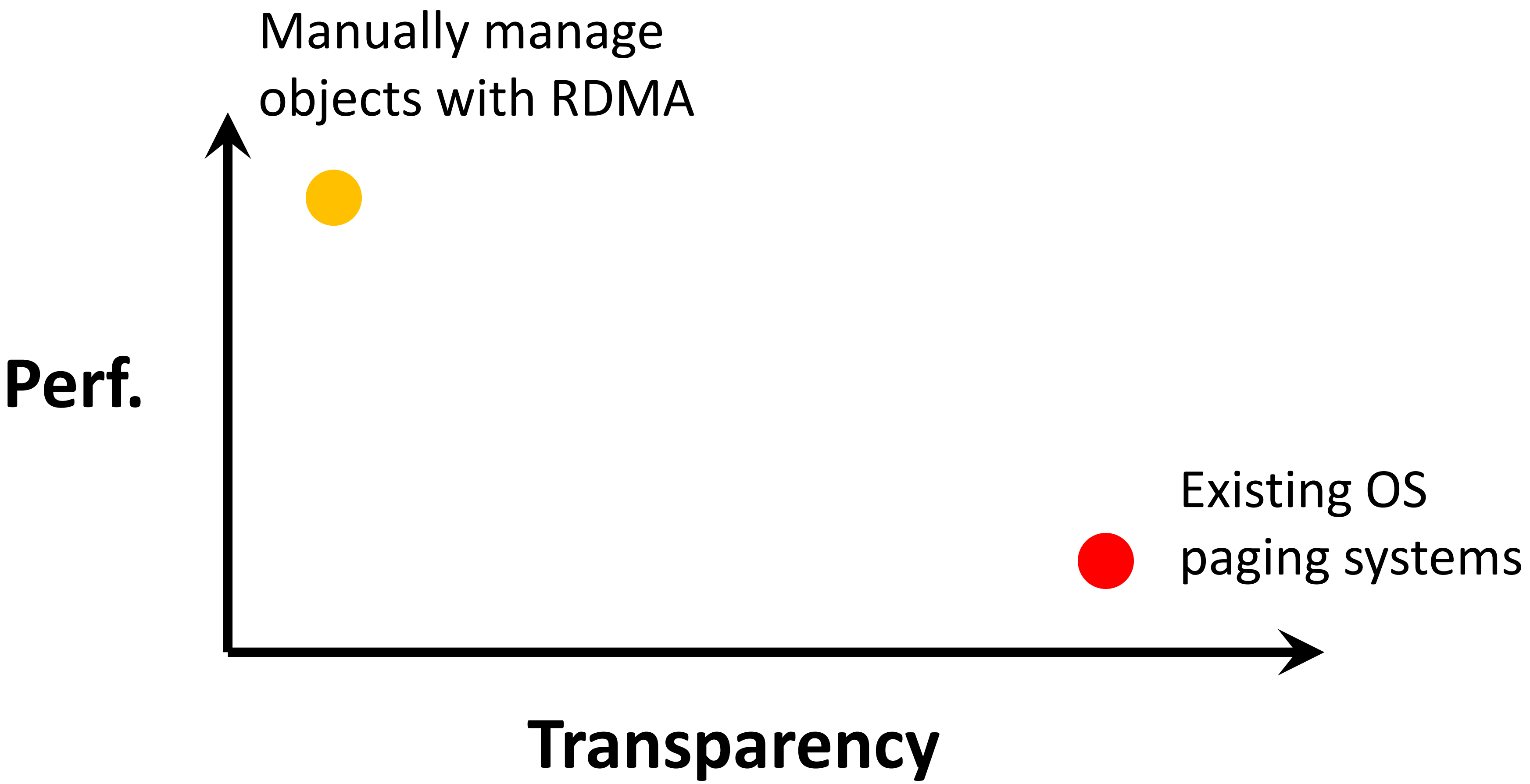
➢ Busy Polling for in-kernel net I/O → **burn CPU cycles**.



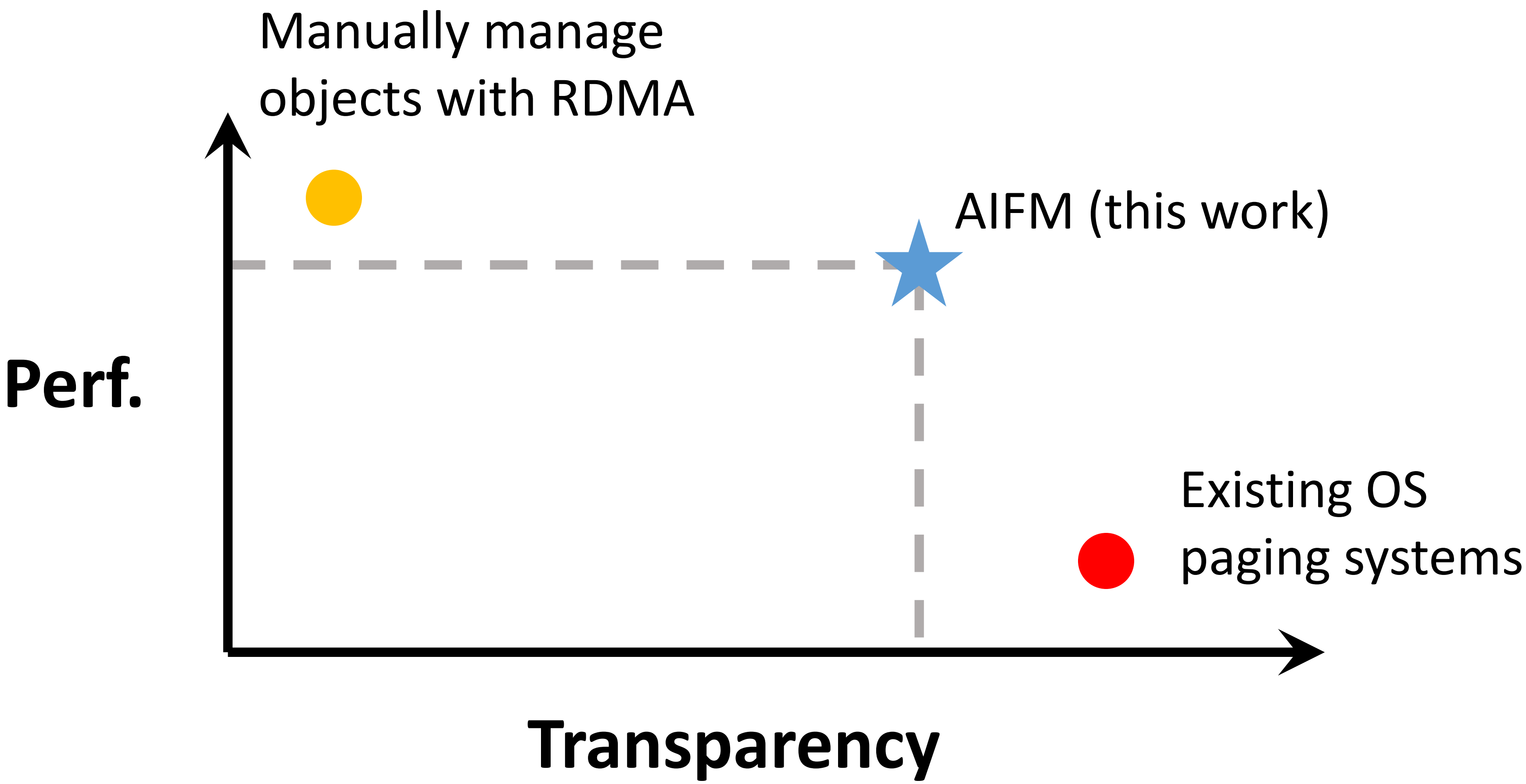
Design Space



Design Space



Design Space



AIFM’s Design Overview

- Key idea: swap memory using a userspace runtime.

AIFM’s Design Overview

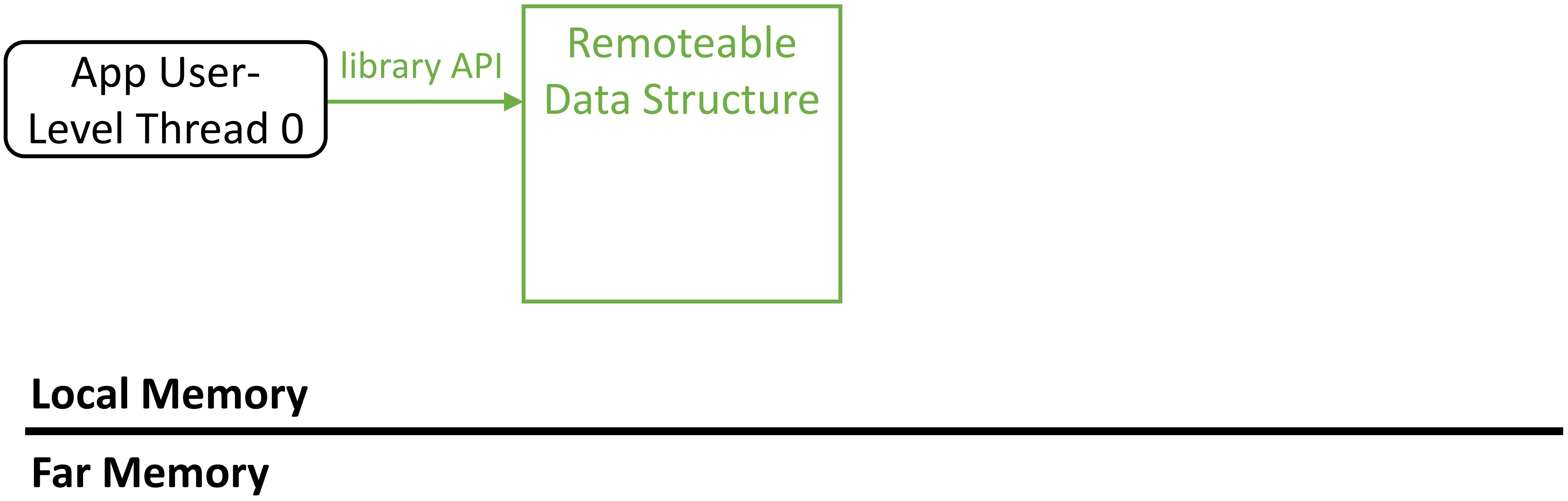
- Key idea: swap memory using a userspace runtime.

Challenge	Solution
1. Semantic gap (Amplification, Hard to prefetch)	Remoteable Data structure library
2. Kernel overheads (page faults, busy poll for net I/O)	Userspace runtime
3. Impact of Memory Reclamation (pause app threads)	Pauseless evacuator
4. network BW < DRAM BW	Remote Agent

AIFM in Action

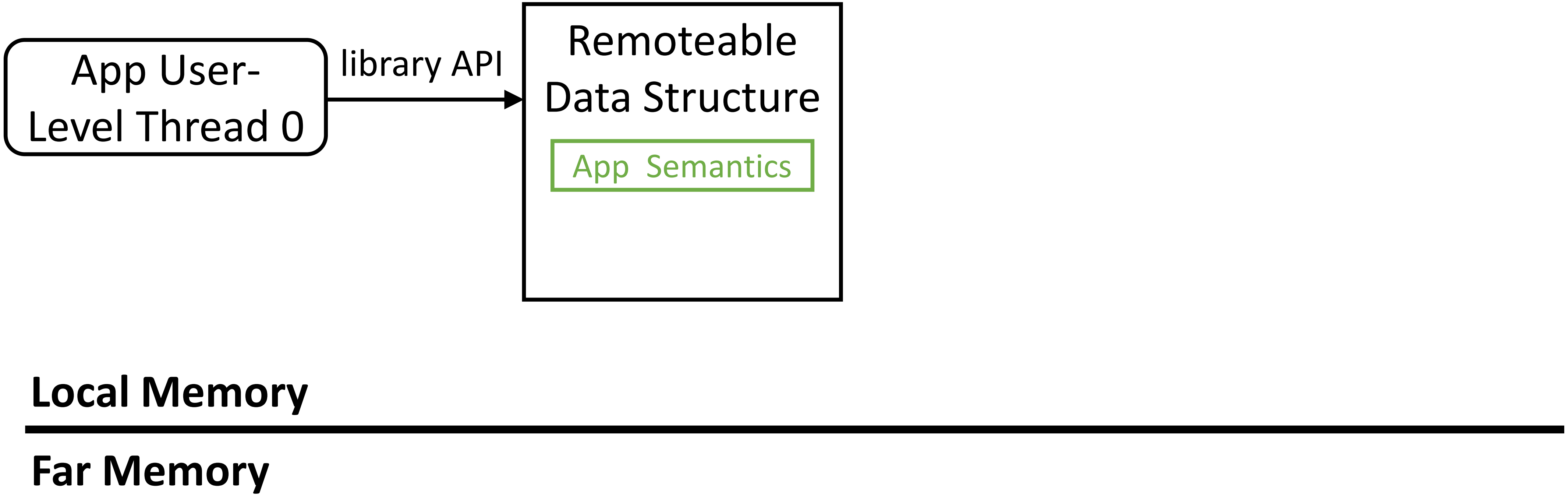
1. Remoteable Data Structure Library

➢Solved challenge: semantic gap.



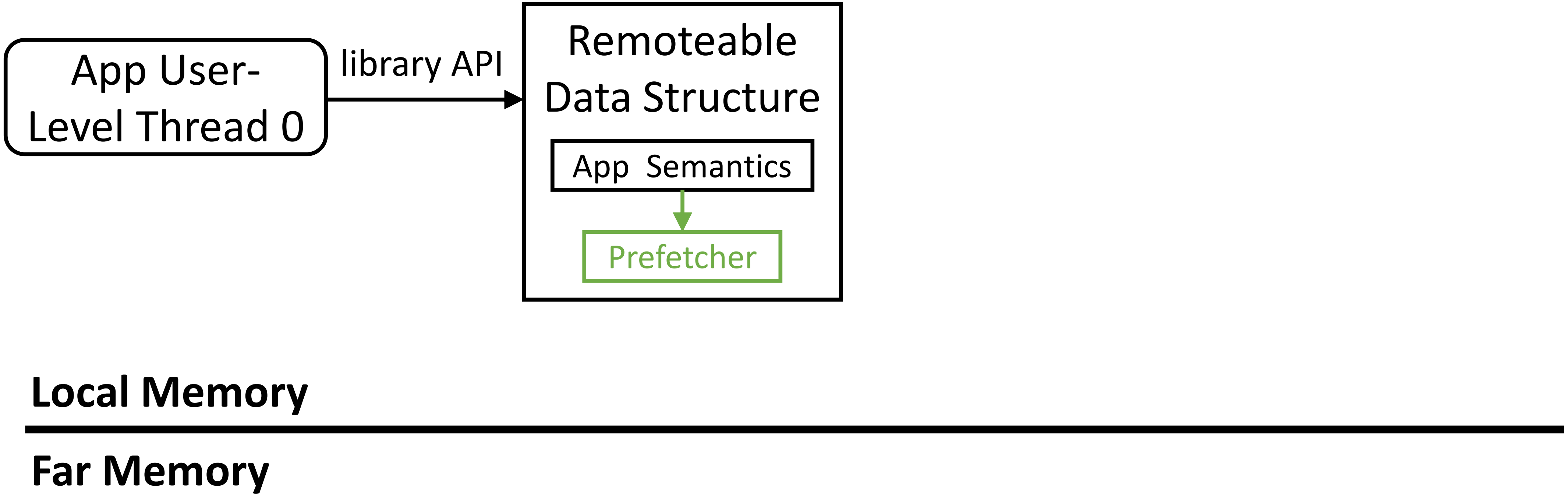
1. Remoteable Data Structure Library

- Solved challenge: semantic gap.



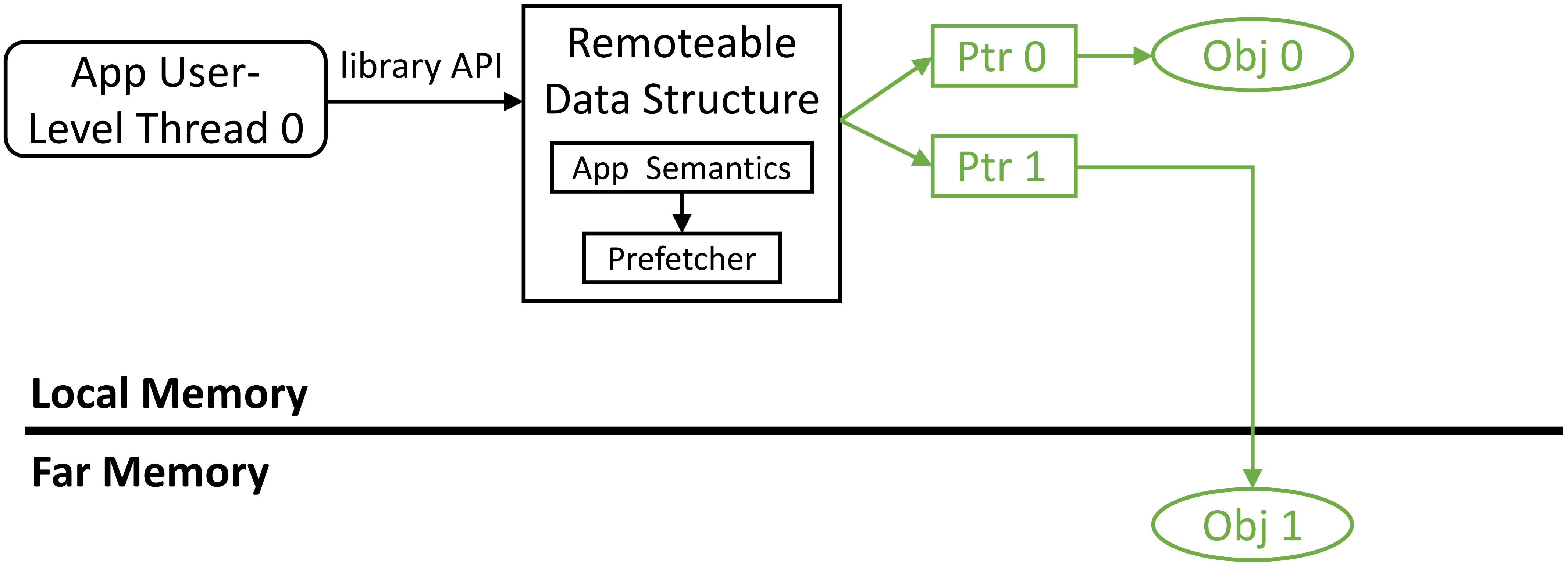
1. Remoteable Data Structure Library

➢Solved challenge: semantic gap.



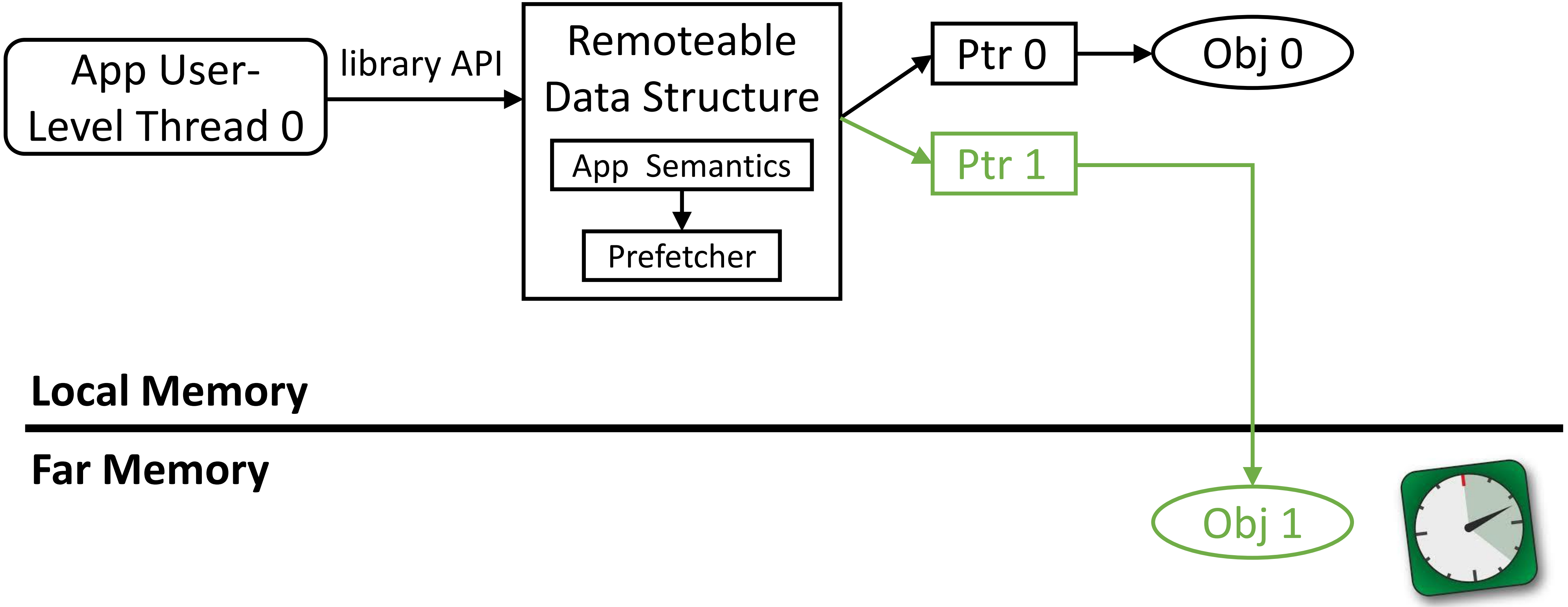
1. Remoteable Data Structure Library

- ➢Solved challenge: semantic gap.



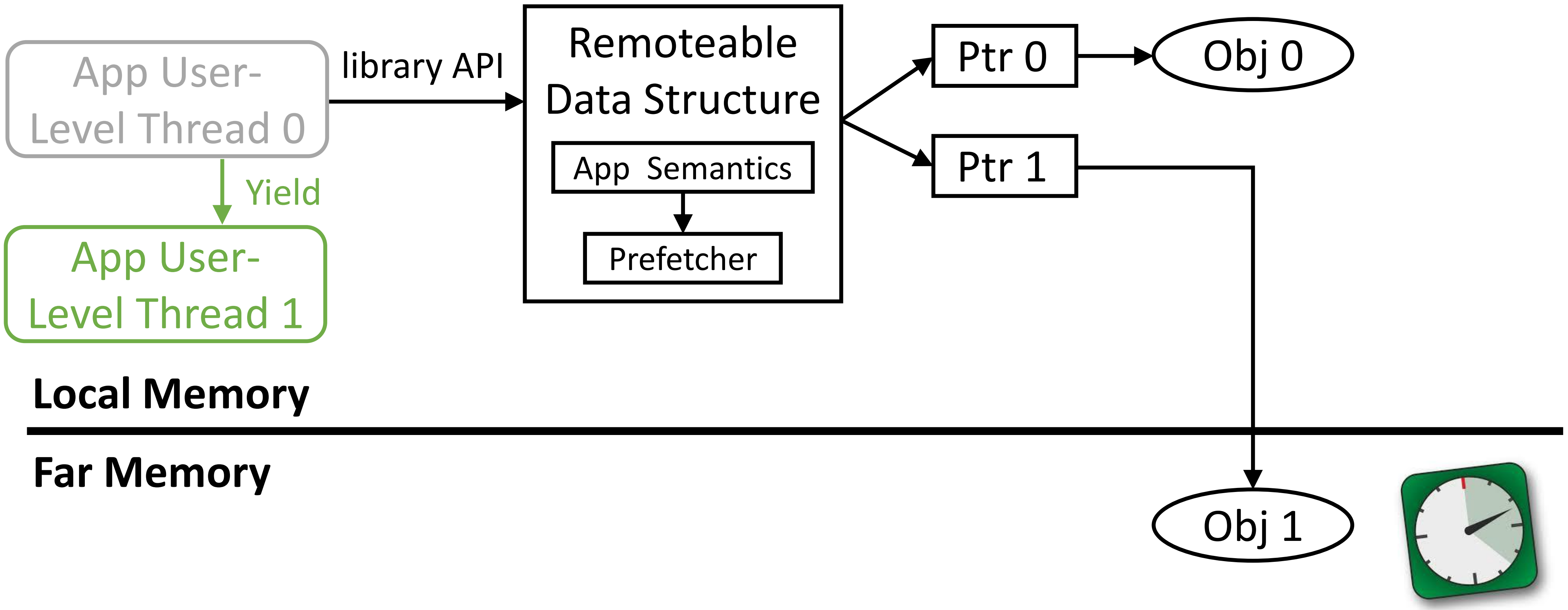
2. Userspace Runtime

- Solved challenge: kernel overheads.



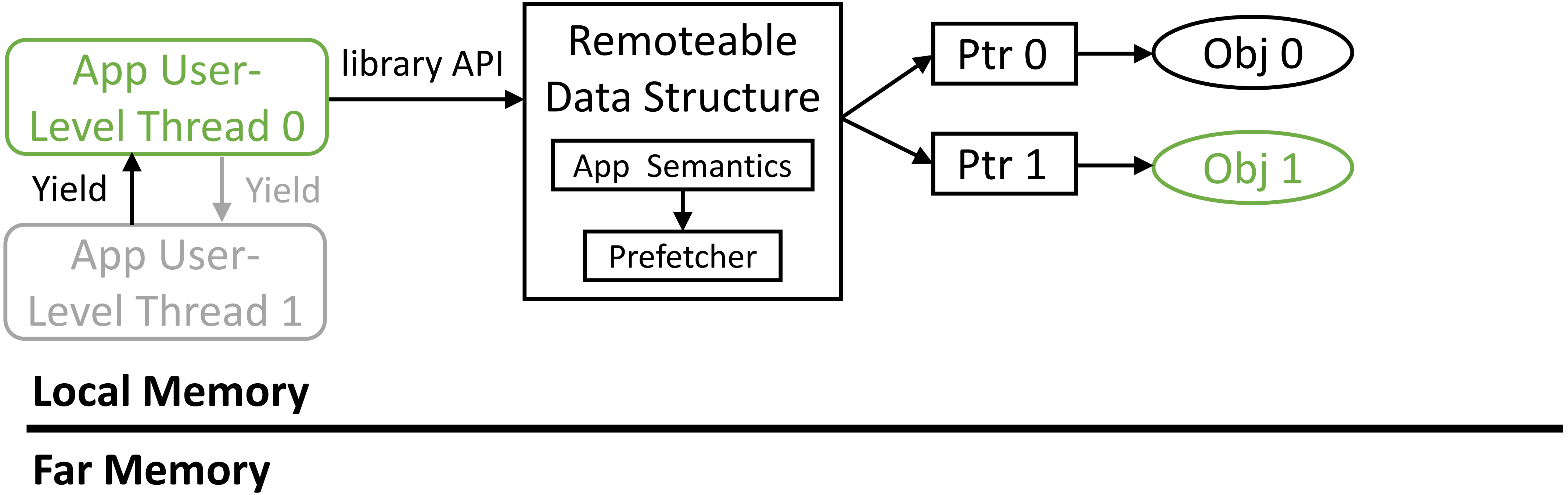
2. Userspace Runtime

- Solved challenge: kernel overheads.



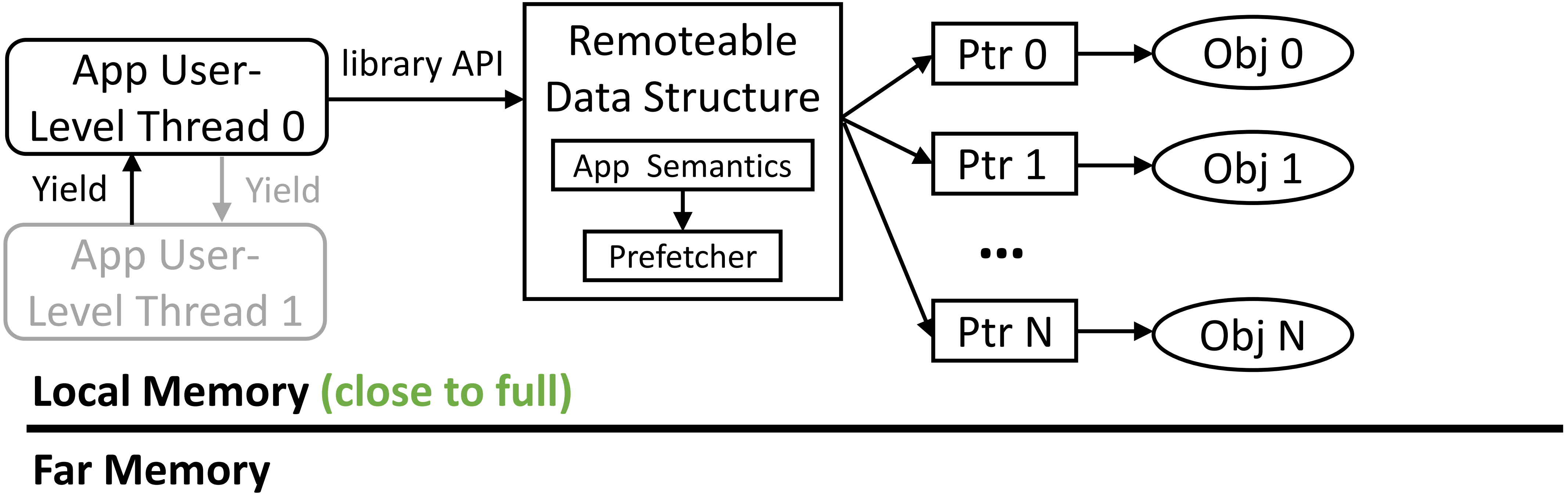
2. Userspace Runtime

➢Solved challenge: kernel overheads.



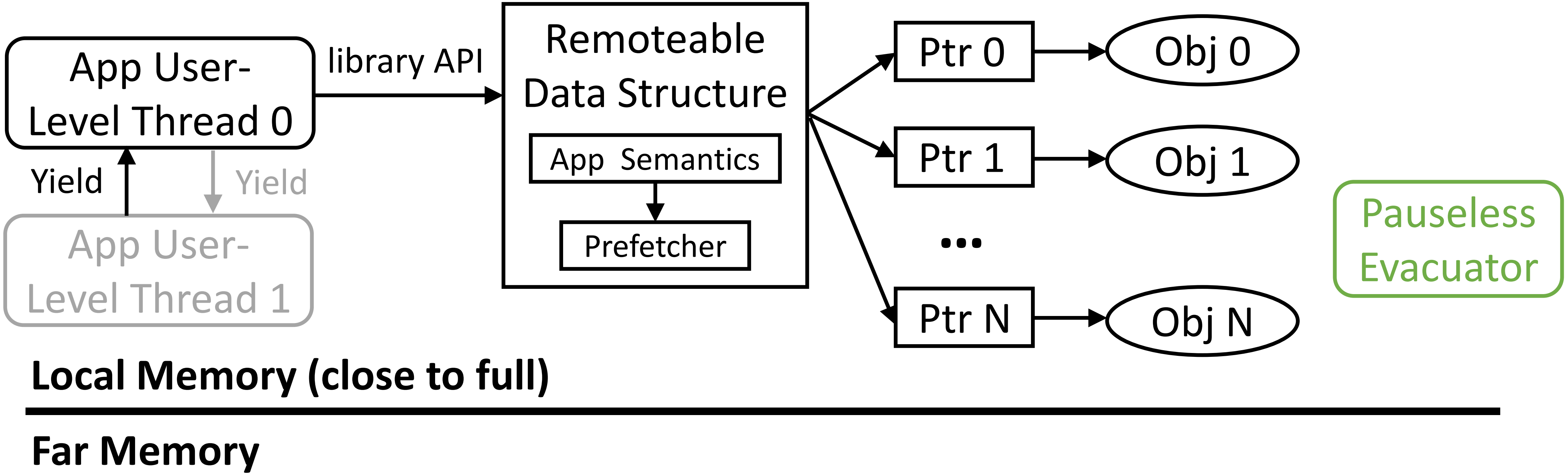
3. Pauseless Evacuator

- ➢Solved challenge: impact of memory reclamation.



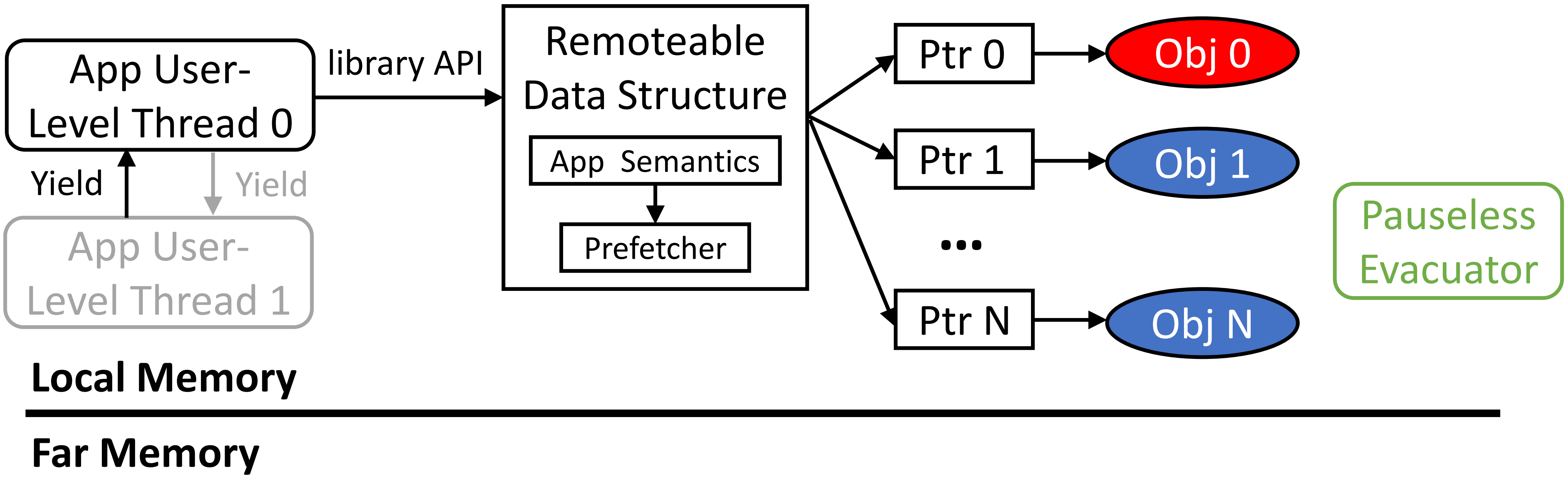
3. Pauseless Evacuator

- Solved challenge: performance impact of memory reclamation.



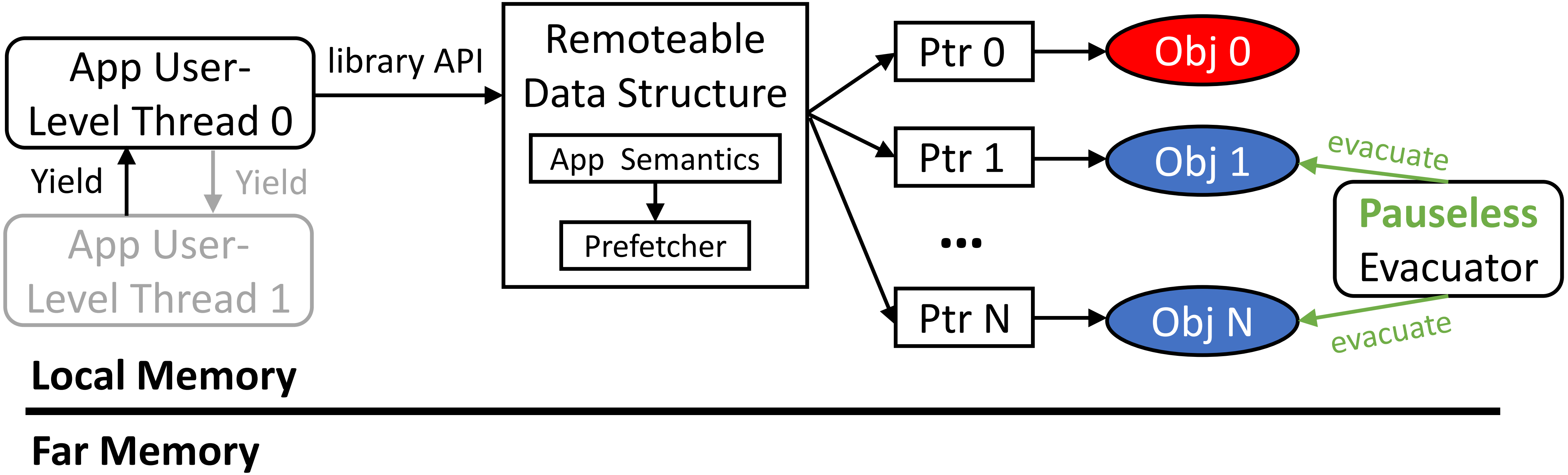
3. Pauseless Evacuator

- Solved challenge: impact of memory reclamation.



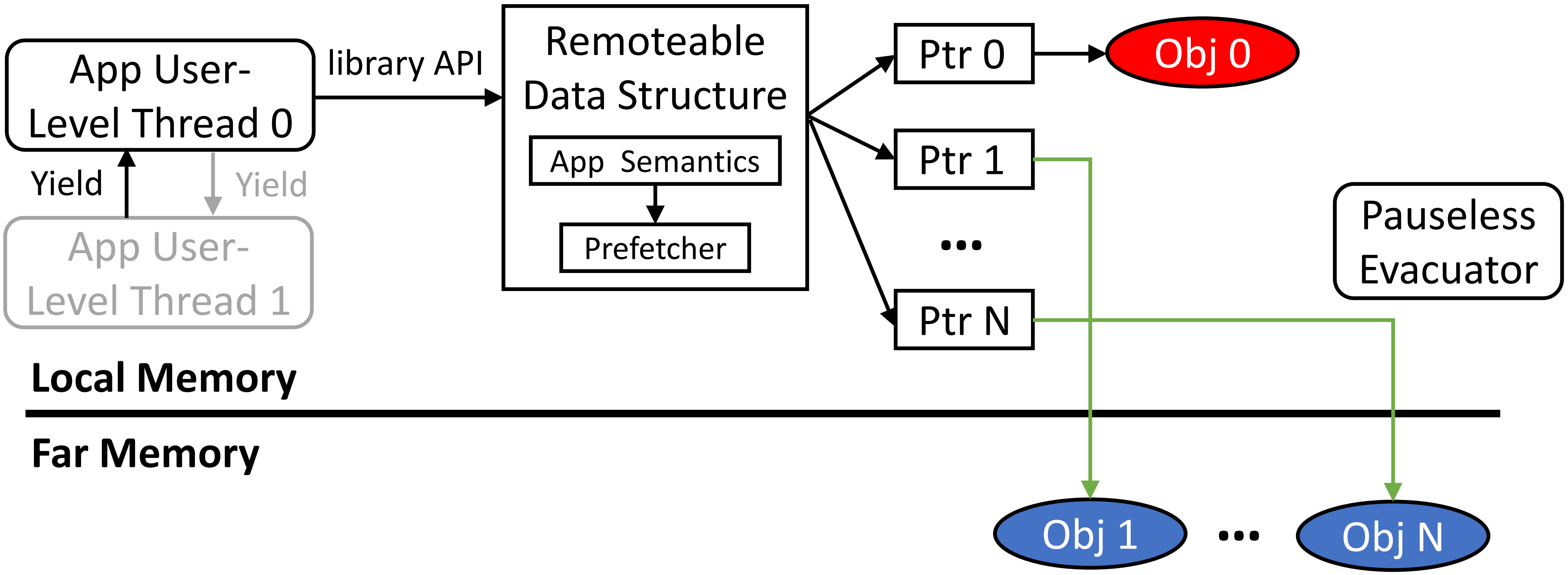
3. Pauseless Evacuator

- Solved challenge: impact of memory reclamation.



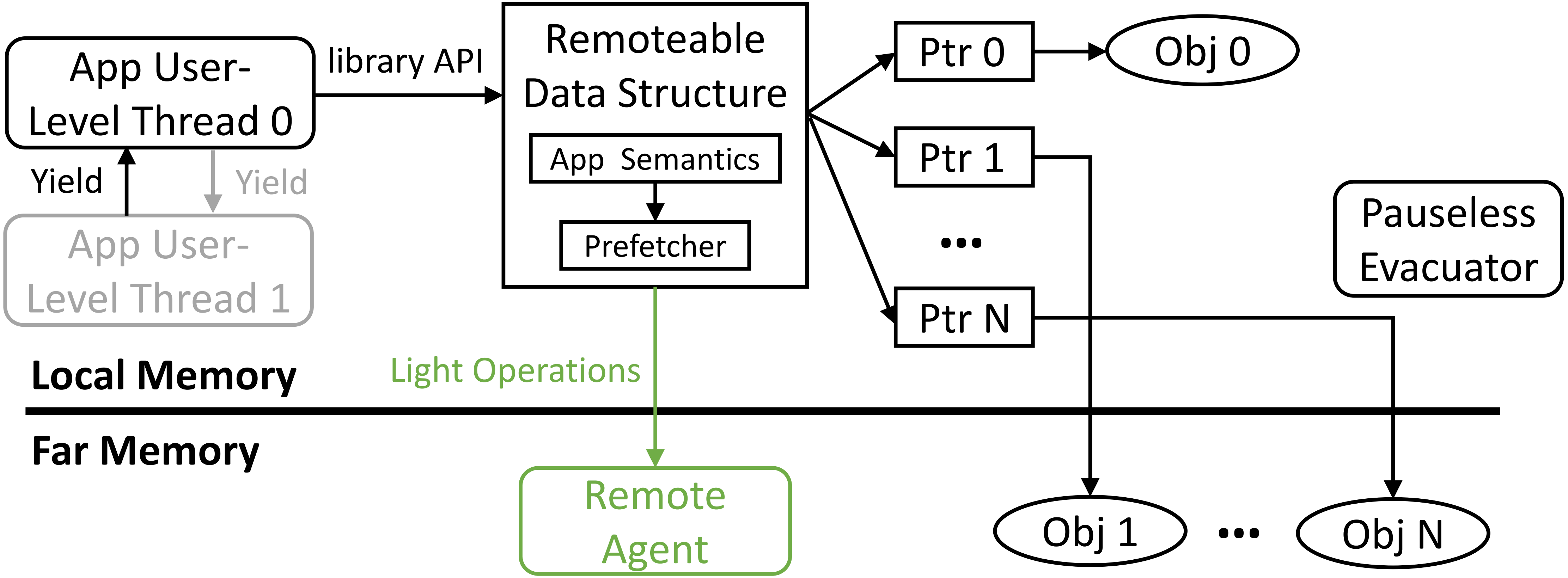
3. Pauseless Evacuator

- Solved challenge: impact of memory reclamation.



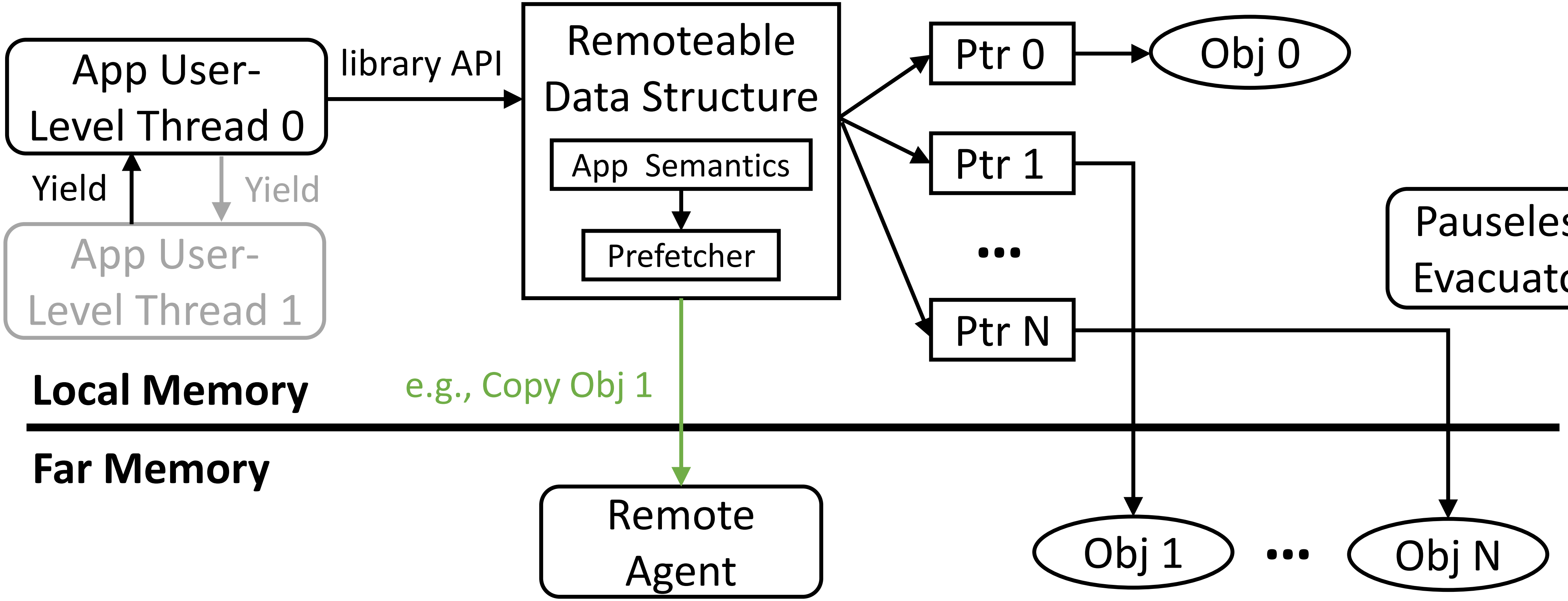
4. Remote Agent

➢Solved challenge: network BW < DRAM BW.



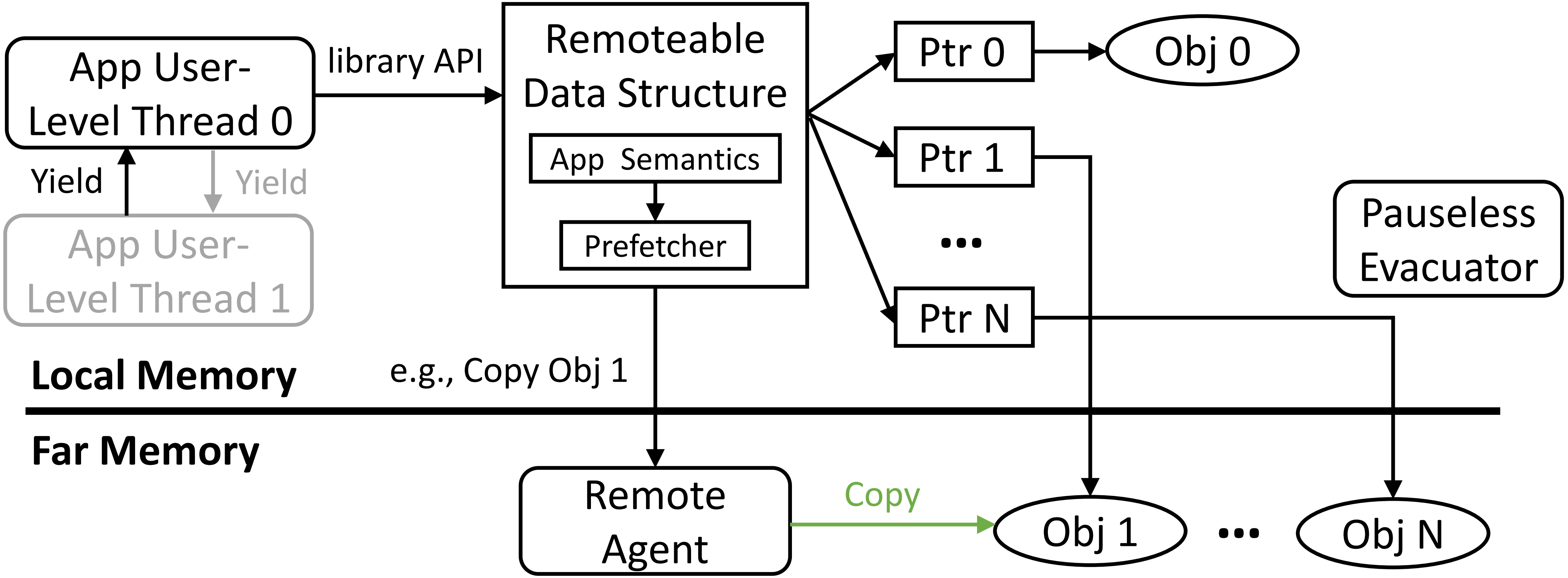
4. Remote Agent

- ➢Solved challenge: network BW < DRAM BW.



4. Remote Agent

- Solved challenge: network BW < DRAM BW.



Sample Code

```
std::unordered_map<key_t, int> hashtable;
std::array<LargeData> arr;

LargeData foo(std::list<key_t> &keys_list) {
    int sum = 0;
    for (auto key : keys_list) {

        sum += hashtable.at(key);
    }

    LargeData ret = arr.at(sum);
    return ret;
}
```

Sample Code

```
RemHashTable<key_t, int> hashtable;
RemArray<LargeData> arr;

LargeData foo(RemList<key_t> &keys_list) {
    int sum = 0;
    for (auto key : keys_list) {

        sum += hashtable.at(key);
    }

    LargeData ret = arr.at(sum);
    return ret;
}
```

Sample Code

```
RemHashTable<key_t, int> hashtable;
RemArray<LargeData> arr;

LargeData foo(RemList<key_t> &keys_list) {
    int sum = 0;
    for (auto key : keys_list) {
        DerefScope scope;
        sum += hashtable.at(key, scope);
    }
    DerefScope scope;
    LargeData ret = arr.at(sum, scope);
    return ret;
}
```

Ensure the accessed objects will not be moved by the evacuator.

Sample Code

```
RemHashTable<key_t, int> hashtable;
RemArray<LargeData> arr;

LargeData foo(RemList<key_t> &keys_list) {
    int sum = 0;
    for (auto key : keys_list) {
        DerefScope scope;
        sum += hashtable.at(key, scope);
    }
    DerefScope scope;
    LargeData ret = arr.at</*don't cache*/ true>(sum, scope);
    return ret;
}
```

Sample Code

```
RemHashTable<key_t, int> hashtable;
RemArray<LargeData> arr;

LargeData foo(RemList<key_t> &keys_list) {
  int sum = 0;
  for (auto key : keys_list) {                    Prefetch list data.
    DerefScope scope;
    sum += hashtable.at(key, scope);
  }
  DerefScope scope;
  LargeData ret = arr.at</*don't cache*/ true>(sum, scope);
  return ret;
}
```

Sample Code

```
RemHashTable<key_t, int> hashtable;
RemArray<LargeData> arr;

LargeData foo(RemList<key_t> &keys_list) {
  int sum = 0;
  for (auto key : keys_list) {                    Prefetch list data.
    DerefScope scope;
    sum += hashtable.at(key, scope);              Cache hot objects.
  }
  DerefScope scope;
  LargeData ret = arr.at</*don't cache*/ true>(sum, scope);
  return ret;
}
```

Sample Code

```
RemHashTable<key_t, int> hashtable;
RemArray<LargeData> arr;

LargeData foo(RemList<key_t> &keys_list) {
  int sum = 0;
  for (auto key : keys_list) {                          Prefetch list data.
    DerefScope scope;
    sum += hashtable.at(key, scope);                    Cache hot objects.
  }
  DerefScope scope;
  LargeData ret = arr.at</*don't cache*/ true>(sum, scope);   Avoid polluting local mem.
  return ret;
}
```

Implementation

- Implemented 6 data structures.
  - Array, List, Hashtable, Vector, Stack, and Queue.

Implementation

- Implemented 6 data structures.
  - Array, List, Hashtable, Vector, Stack, and Queue.

➢Runtime is built on top of Shenango [NSDI' 19].

Implementation

- Implemented 6 data structures.
  - Array, List, Hashtable, Vector, Stack, and Queue.
- Runtime is built on top of Shenango [NSDI’ 19].

➢TCP far-memory backend.

Implementation

- Implemented 6 data structures.
  - Array, List, Hashtable, Vector, Stack, and Queue.
- Runtime is built on top of Shenango [NSDI' 19].
- TCP far-memory backend.

➢LoC: 6.5K (runtime) + 5.5K (data structures) + 0.8K (Shenango)

Evaluation

- Setup: 1 compute server + 1 far memory server, 25 GbE.

Evaluation

- Setup: 1 compute server + 1 far memory server, 25 GbE.
- How does AIFM
  - … perform on applications with different compute intensities?

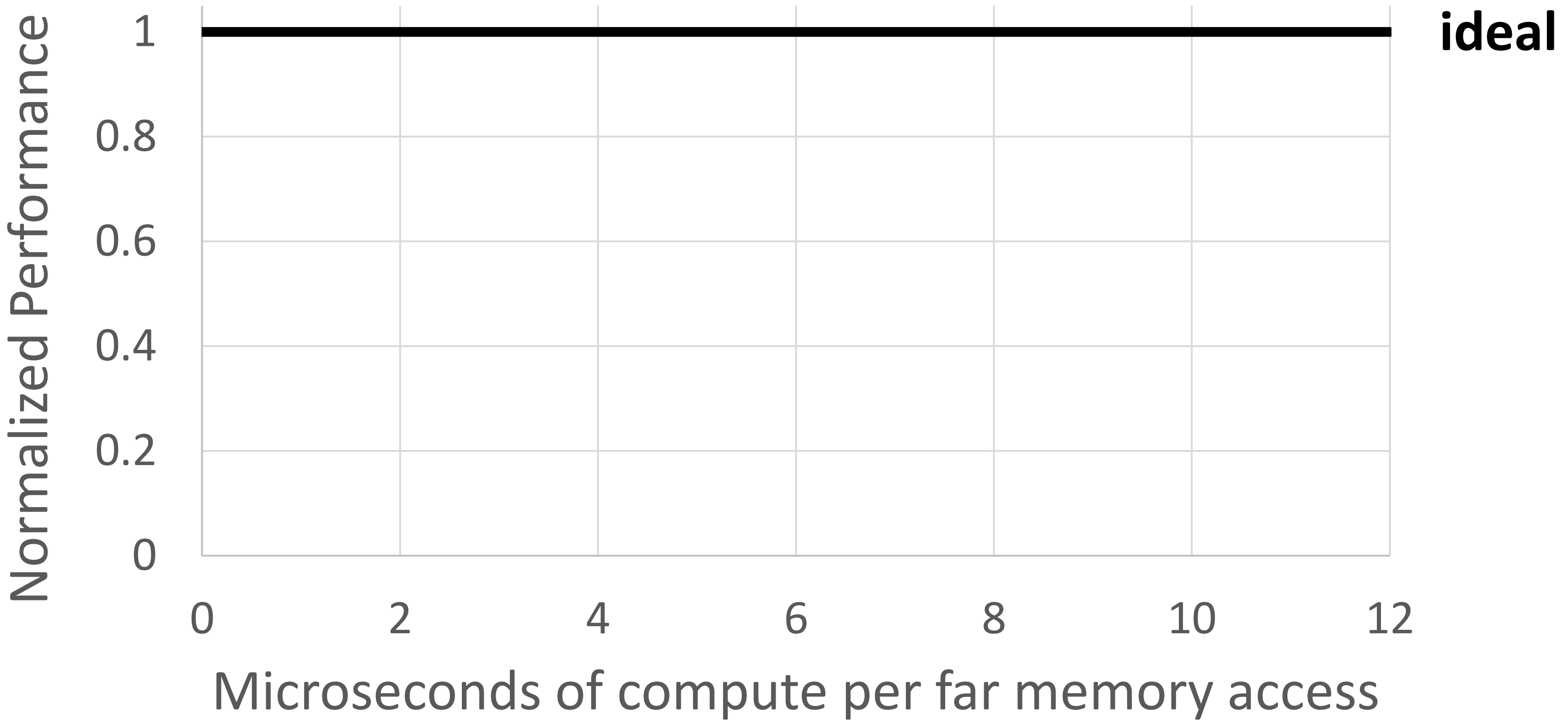
Evaluation

- Setup: 1 compute server + 1 far memory server, 25 GbE.
- How does AIFM
  - … perform on applications with different compute intensities?
  - ➢ … compare to the local-only (ideal) system?

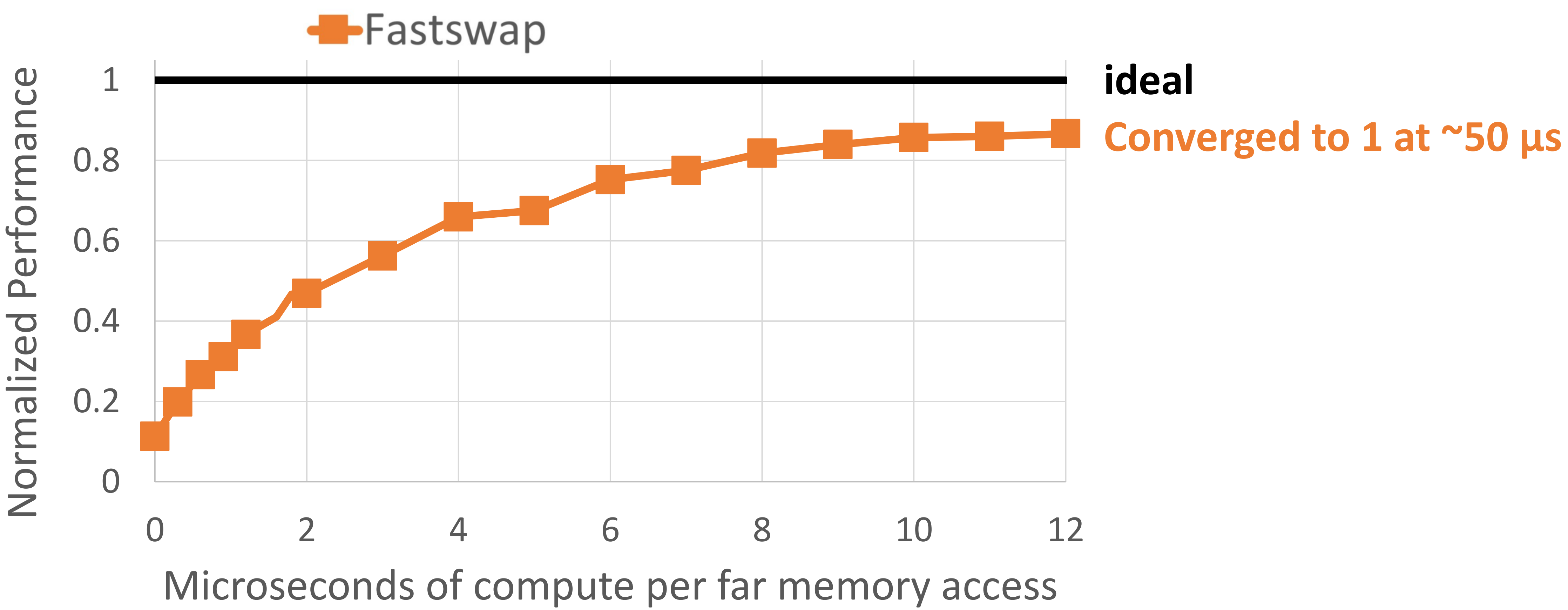
Evaluation

- Setup: 1 compute server + 1 far memory server, 25 GbE.
- How does AIFM
  - … perform on applications with different compute intensities?
  - … compare to the local-only (ideal) system?
  - ➢ … compare to the state-of-the-art paging system, Fastswap [EuroSys' 20]?

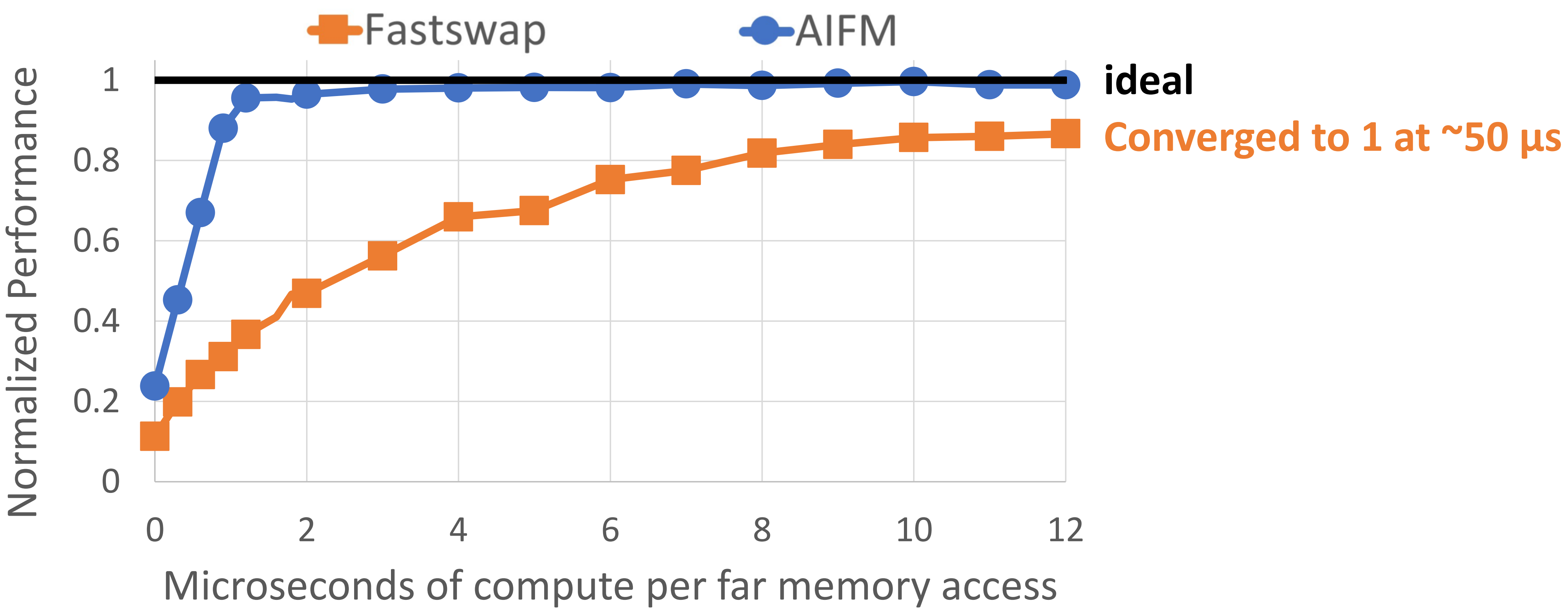
Performance on Different Compute Intensities



Performance on Different Compute Intensities

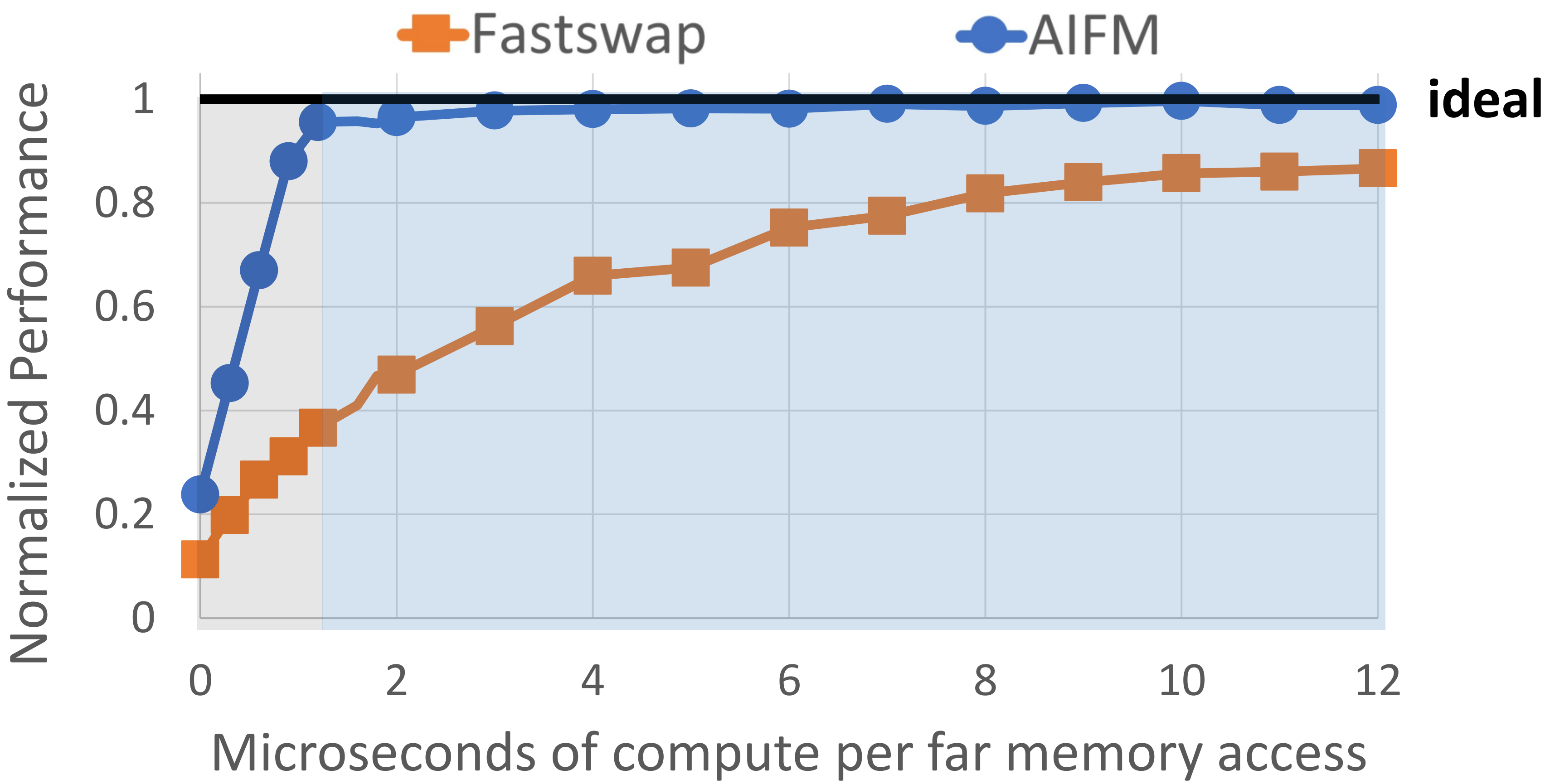


Performance on Different Compute Intensities



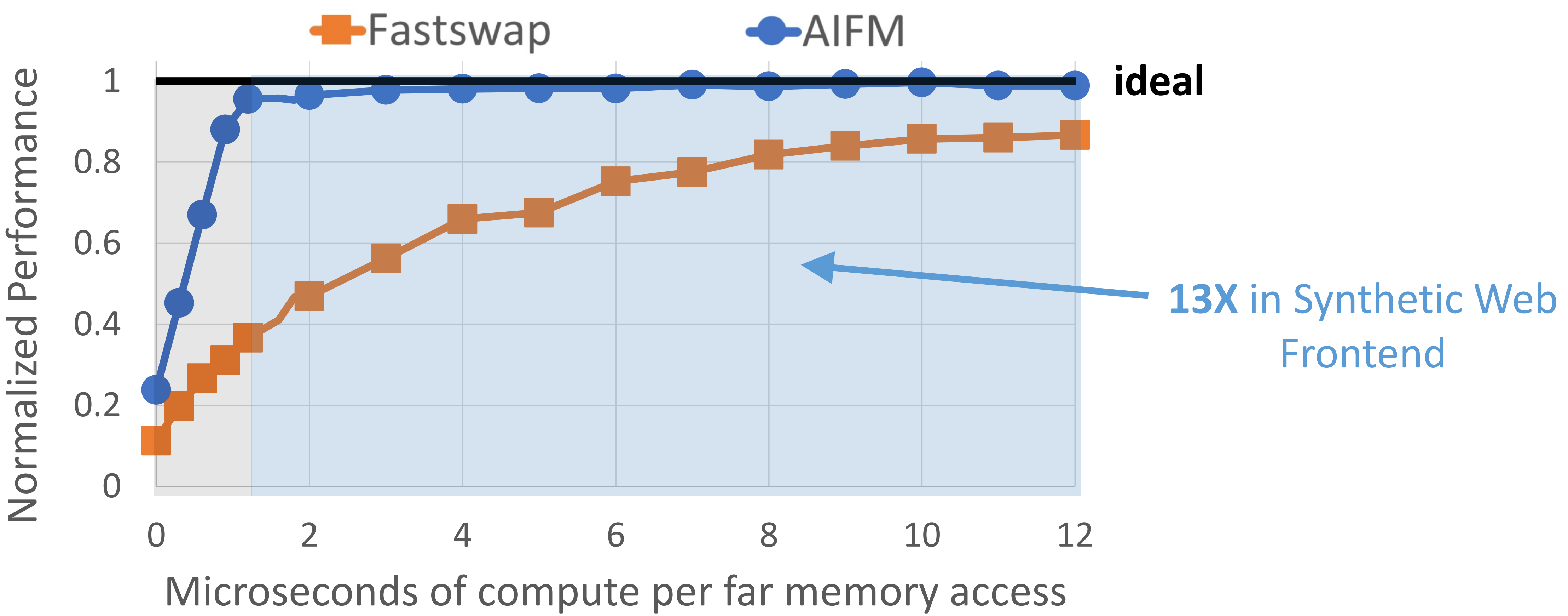
AIFM hides far memory latency with moderate compute.

Performance on Different Compute Intensities

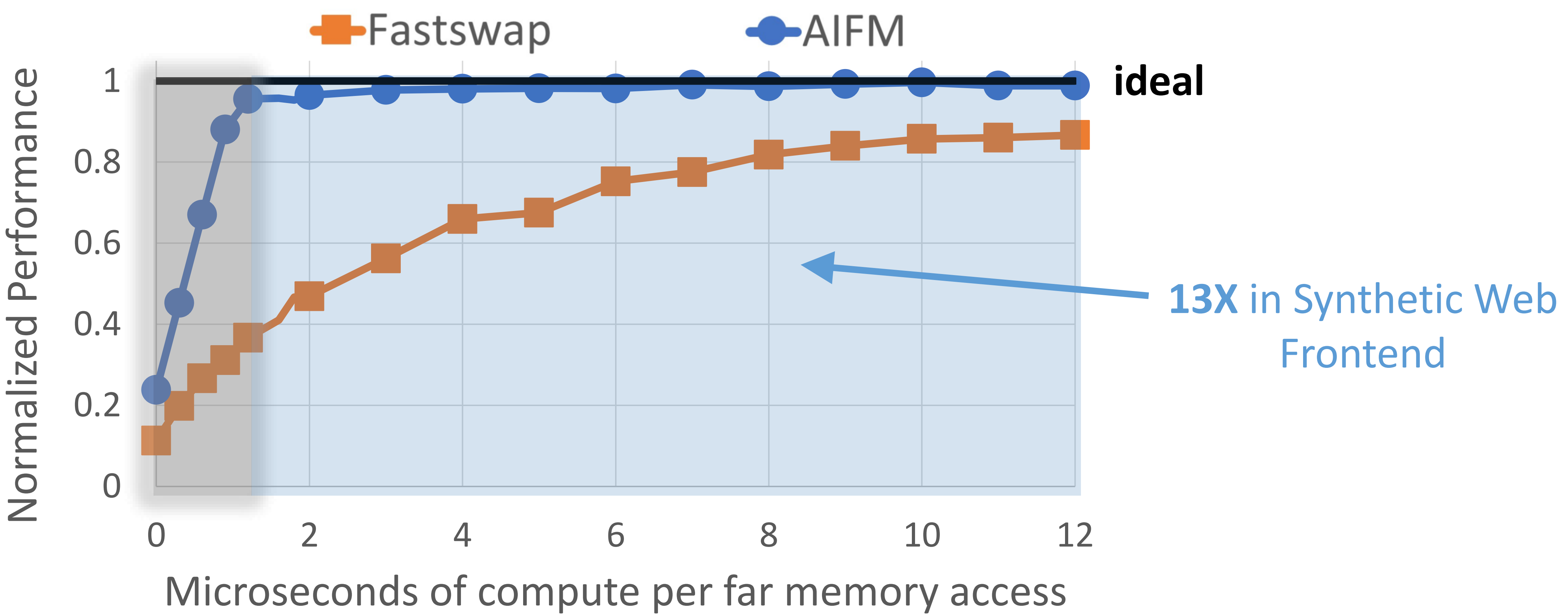


AIFM hides far memory latency with moderate compute.

Performance on Different Compute Intensities



Performance on Different Compute Intensities



NYC Taxi Analysis (C++ DataFrame)

- DataFrame: data analytical framework, similar to Python Pandas.

NYC Taxi Analysis (C++ DataFrame)

- DataFrame: data analytical framework, similar to Python Pandas.

➢Real Kaggle workload

  - Working set size = 31 GB.
  - Modify 1.4K LoC (out of 24.3K LoC), five person-days.

NYC Taxi Analysis (C++ DataFrame)

- DataFrame: data analytical framework, similar to Python Pandas.
- Real Kaggle workload
  - Working set size = 31 GB.
  - Modify 1.4K LoC (out of 24.3K LoC), five person-days.

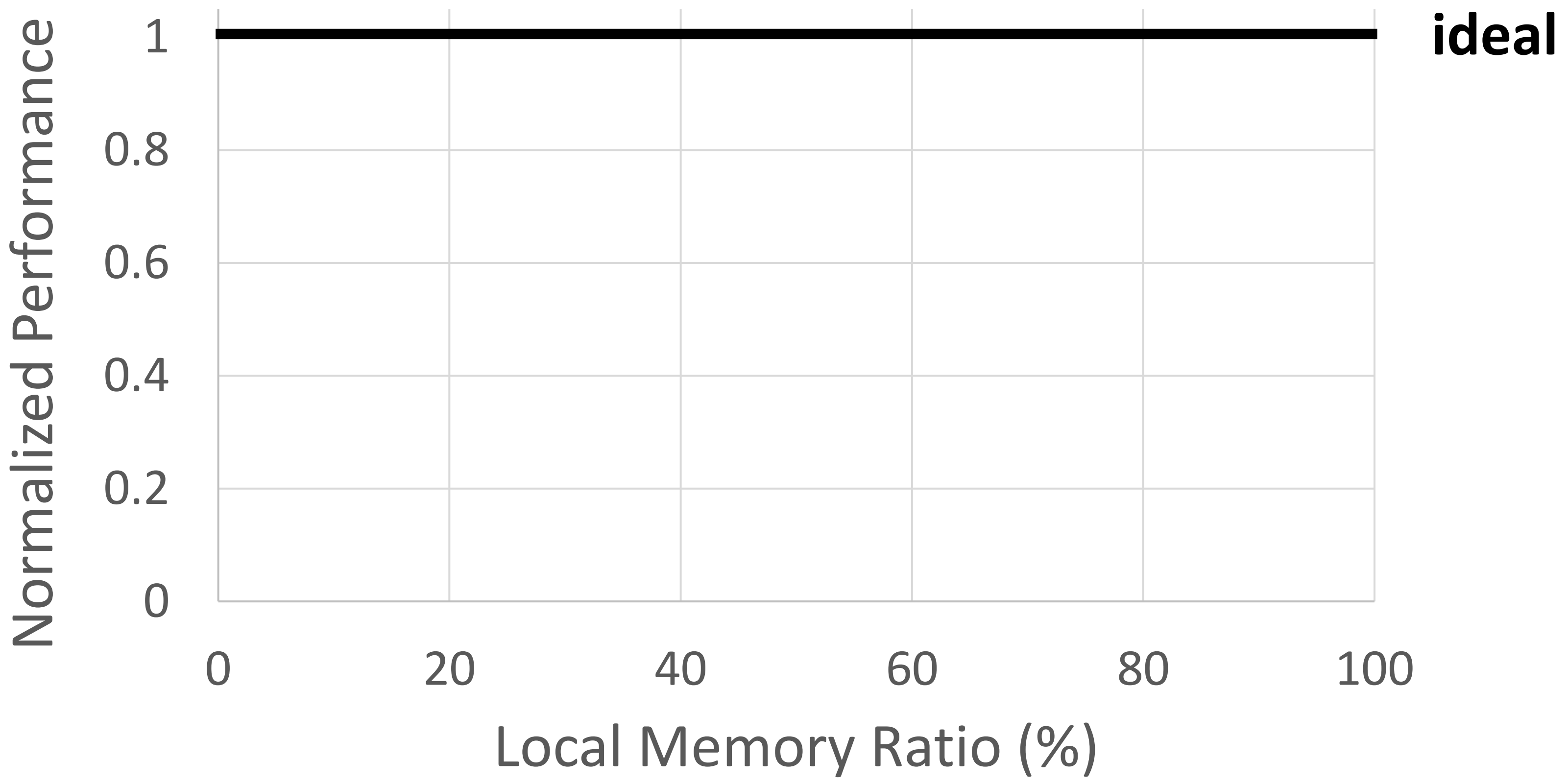
➢Relatively low compute intensity → Unable to hide far-mem latency.

NYC Taxi Analysis (C++ DataFrame)

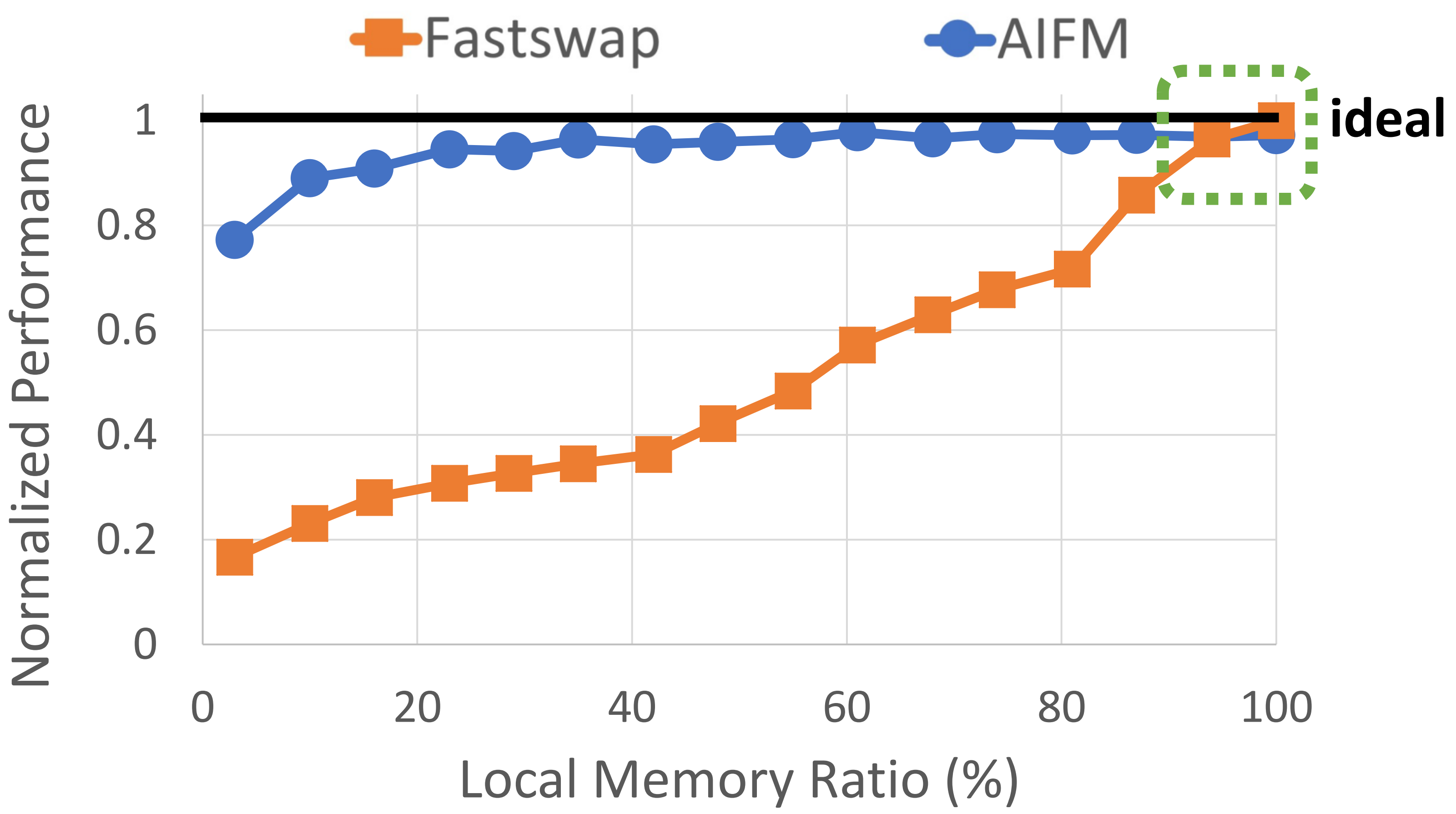
- DataFrame: data analytical framework, similar to Python Pandas.
- Real Kaggle workload
  - Working set size = 31 GB.
  - Modify 1.4K LoC (out of 24.3K LoC), five person-days.
- Relatively low compute intensity → Unable to hide far-mem latency.

➢Keep complex operations local and **offload** very light operations.
  - Significantly reduces expensive data transfer over network.

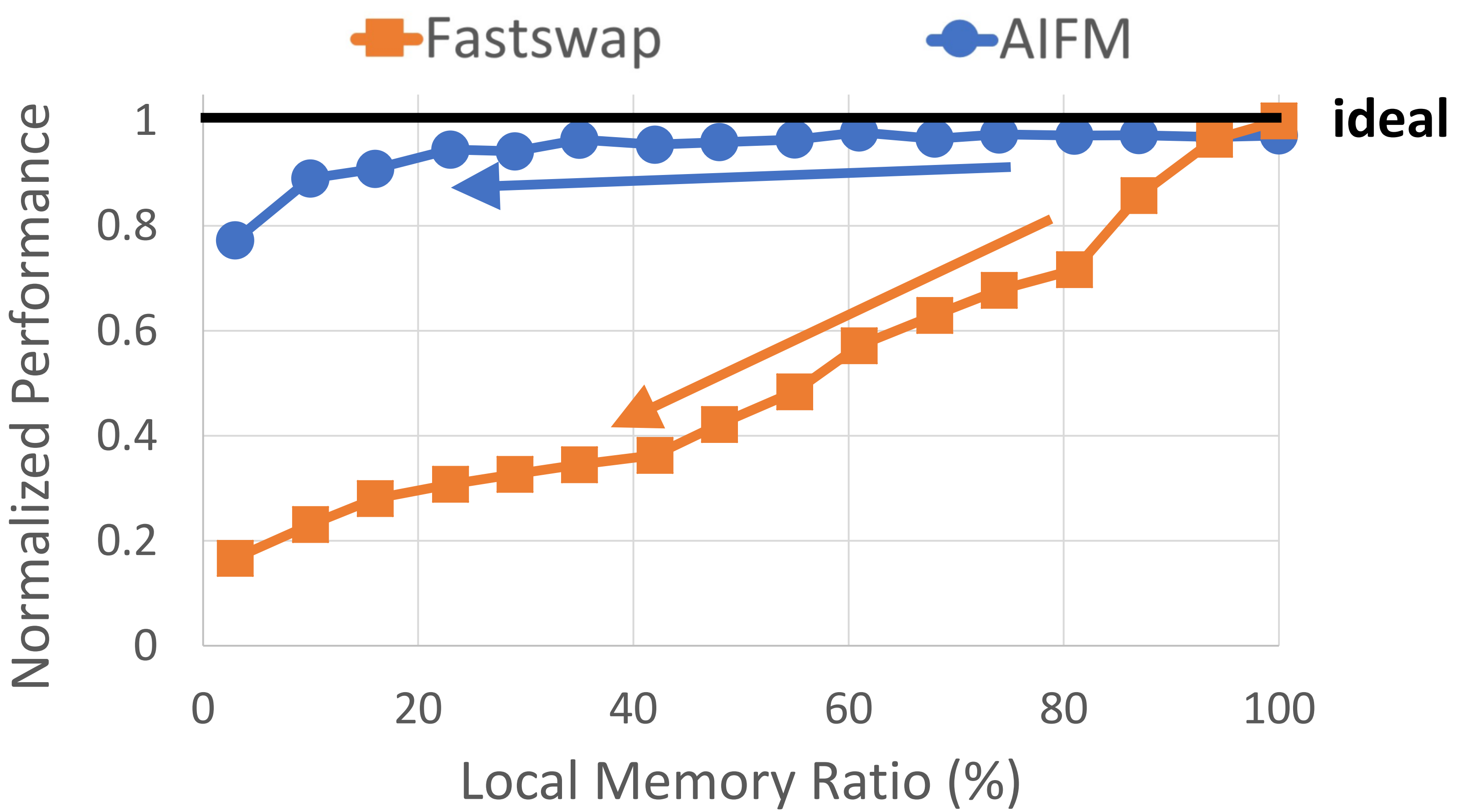
NYC Taxi Analysis (C++ DataFrame)



NYC Taxi Analysis (C++ DataFrame)

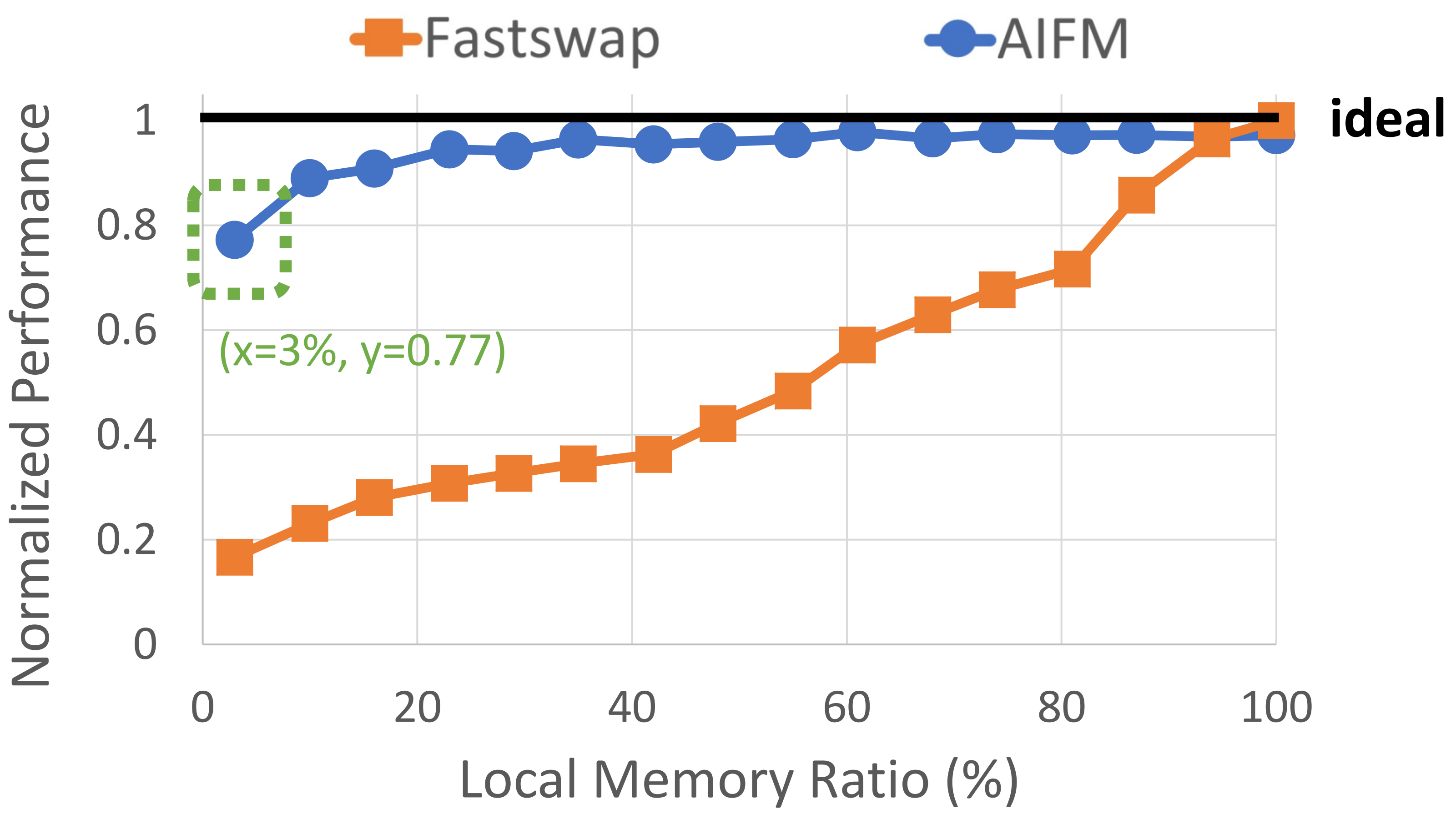


NYC Taxi Analysis (C++ DataFrame)



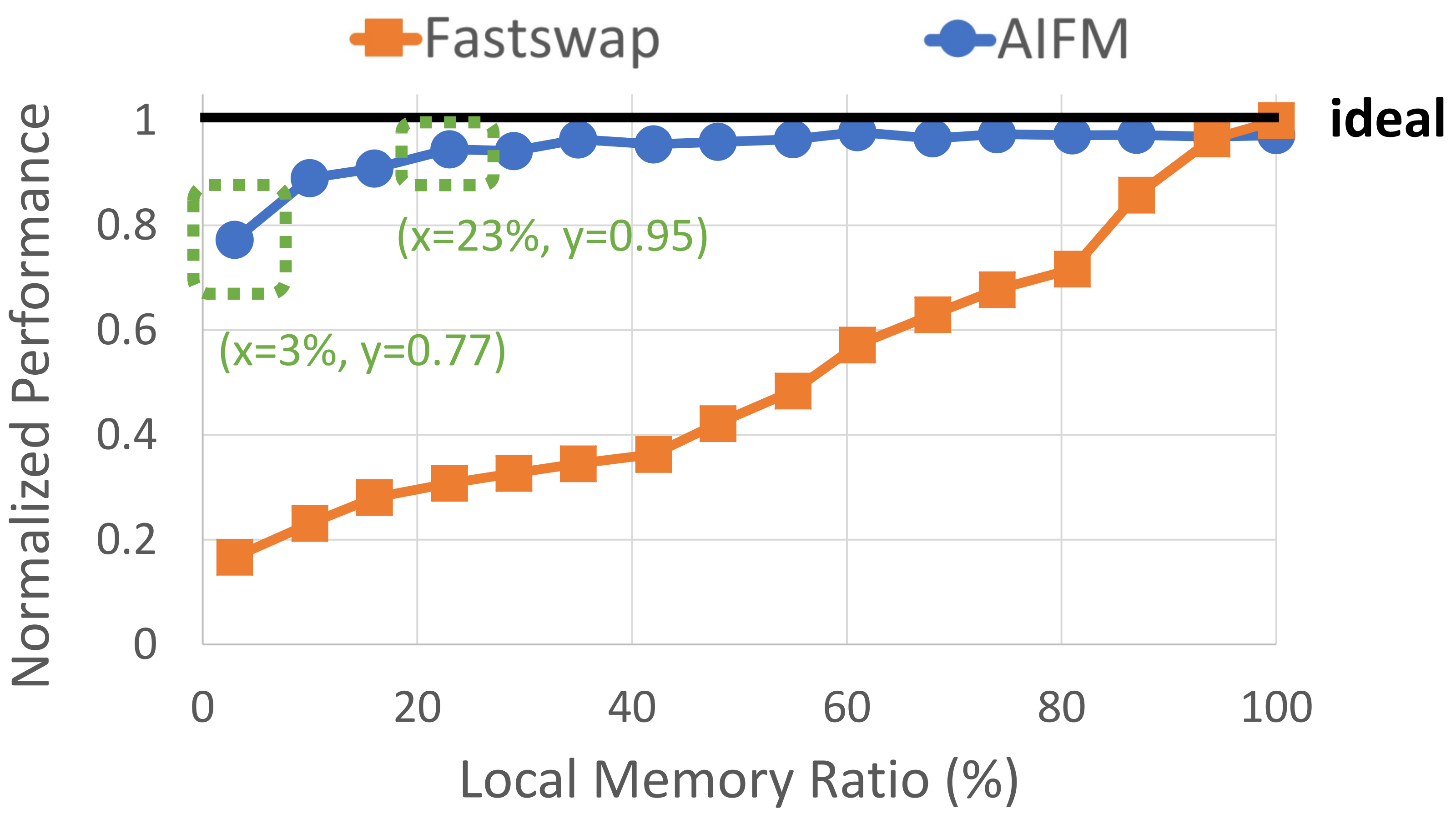
AIFM achieves near-ideal performance with small local memory.

NYC Taxi Analysis (C++ DataFrame)



AIFM achieves near-ideal performance with small local memory.

NYC Taxi Analysis (C++ DataFrame)



AIFM achieves near-ideal performance with small local memory.

Other Experiments

- Synthetic web frontend: up to **13X end-to-end** speedup.
- Data structures microbenchmarks: up to **61X** speedup.
- Design Drill-Down.

Read our paper for details.

Related Work

- OS-paging systems.
  - Fastswap [EuroSys’ 20], Leap [ATC’ 20]
- Distributed shared memory.
  - Treadmarks [IEEE Computer’ 96]
- Garbage collection (GC).

Conclusion

- AIFM: Application-Integrated Far Memory.

Conclusion

- AIFM: Application-Integrated Far Memory.

➢ Key idea: swap memory using a userspace runtime.

Conclusion

- AIFM: Application-Integrated Far Memory.
- Key idea: swap memory using a userspace runtime.
    - ➢Data Structure Library: captures application semantics.

Conclusion

- AIFM: Application-Integrated Far Memory.
- Key idea: swap memory using a userspace runtime.
  - Data Structure Library: captures application semantics.
  - ➢Userspace Runtime: efficiently manages objects and memory.

Conclusion

- AIFM: Application-Integrated Far Memory.
- Key idea: swap memory using a userspace runtime.
  - Data Structure Library: captures application semantics.
  - Userspace Runtime: efficiently manages objects and memory.

➢ Achieves 13X end-to-end speedup over Fastswap.

Conclusion

- AIFM: Application-Integrated Far Memory.
- Key idea: swap memory using a userspace runtime.
  - Data Structure Library: captures application semantics.
  - Userspace Runtime: efficiently manages objects and memory.
- Achieves 13X end-to-end speedup over Fastswap.

➢ Code released at https://github.com/AIFM-sys/AIFM

Please send your questions to us

zainruan@csail.mit.edu