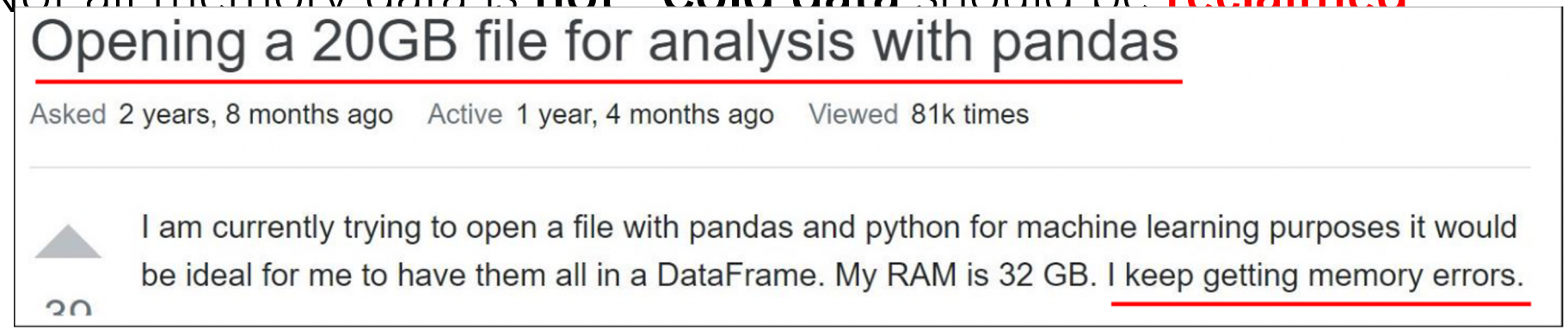
AIFM: High-Performance, Application-Integrated Far Memory

AIFM: High-Performance, Application-Integrated Far Memory

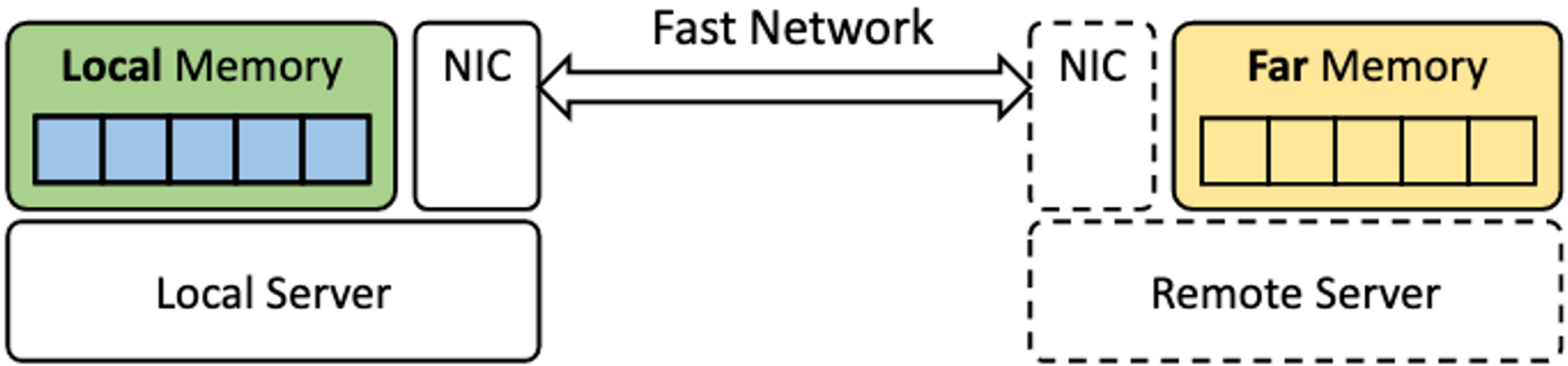
- Memory is becoming the most constrained resource.
- Why ?
  - The **average memory utilization** on servers is **60%** but only **40%** for **CPU utilization**.
    - (In Google and Alibaba' s data center)
  - Memory is **inelastic.**
    - run out of available memory => be **killed**.
  - Not all memory data is **hot**. **Cold data** should be **reclaimed**



Expensive solution: overprovision memory for peak usage.

Trending Solution: Far Memory

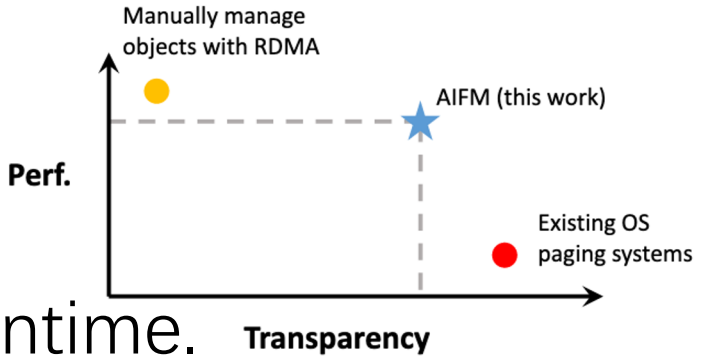
- Leverage the idle memory of remote servers (with fast network)



Some Background

- OS Swapping and Far Memory
  - OS swap physical memory pages out into **secondary storage**, e.g. SSD.
  - Now they have a faster option, far memory.
- Disaggregated Memory
  - Requires new hardware not yet in production.
- Distributed Shared Memory
  - the core is **Shared**, which requires **cache coherency protocol** that impairs performance.
- Technologies to access remote data
  - TCP/IP or RDMA (need new hardware)
- I/O amplification
  - page fault always means **no less than one page swapping**.

AIFM’ s Design Overview

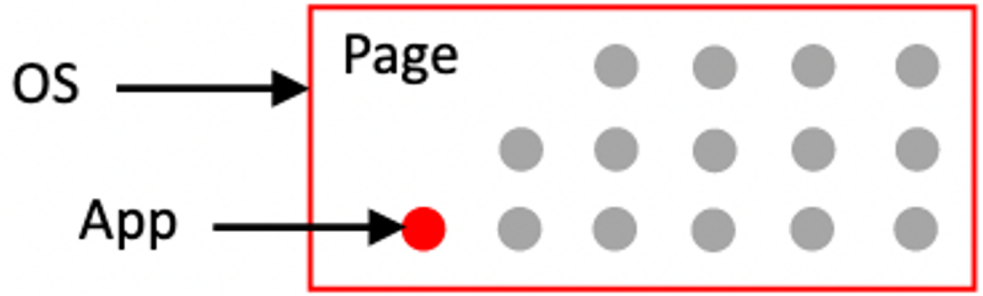


- Key idea: swap memory using a userspace runtime.

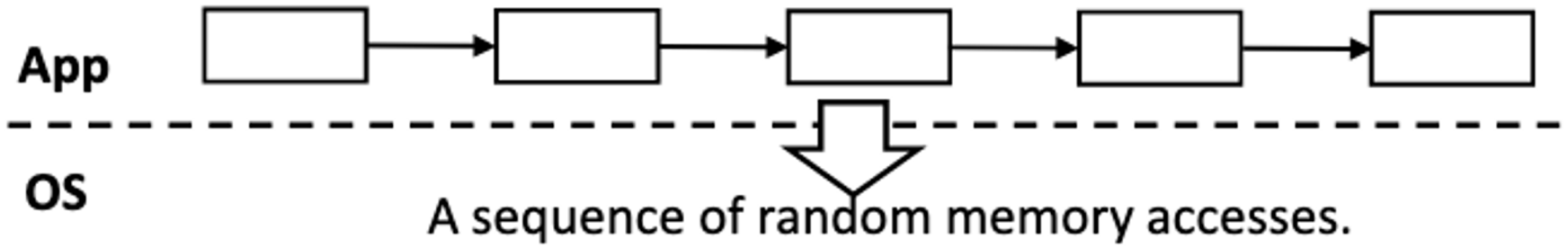
Challenge	Solution
1. Semantic gap (Amplification, Hard to prefetch)	Remoteable Data structure library
2. Kernel overheads (page faults, busy poll for net I/O)	Userspace runtime
3. Impact of Memory Reclamation (pause app threads)	Pauseless evacuator
4. network BW < DRAM BW	Remote Agent

Challenge 1: Semantic Gap

- OS page swapping -- page granularity => **R/W amplification**.

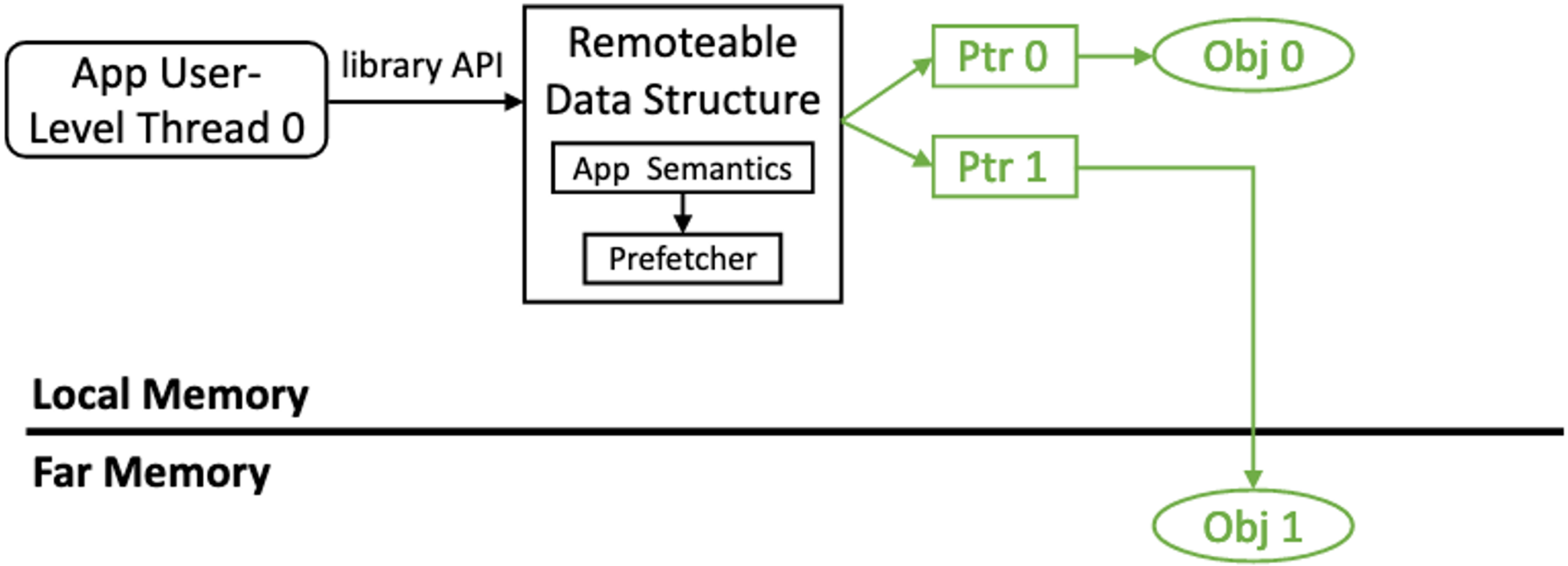


- OS lacks app knowledge => hard to prefetch, etc.



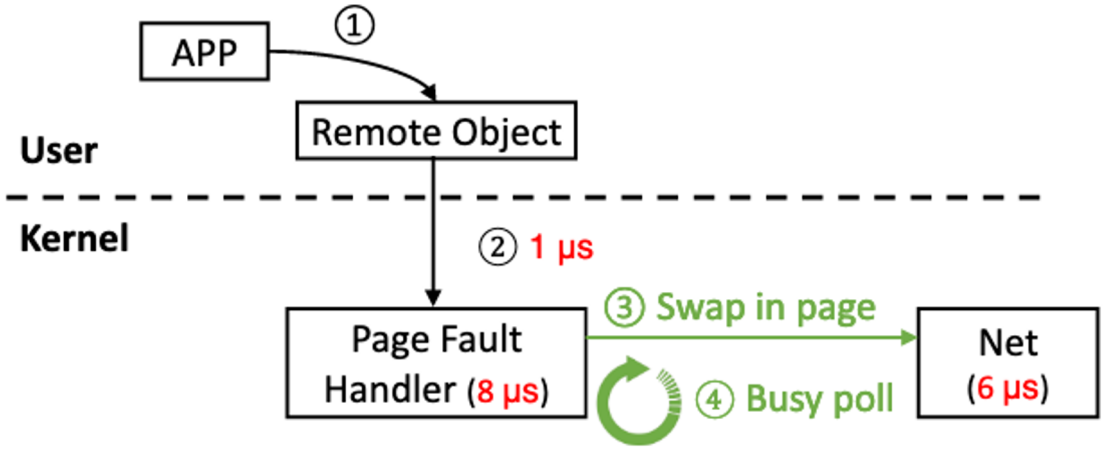
To solve challenge 1, semantic gap

- Remoteable Data Structure Library
  - Provide API to swap data between Local Memory and Far Memory at **fine-grained**.

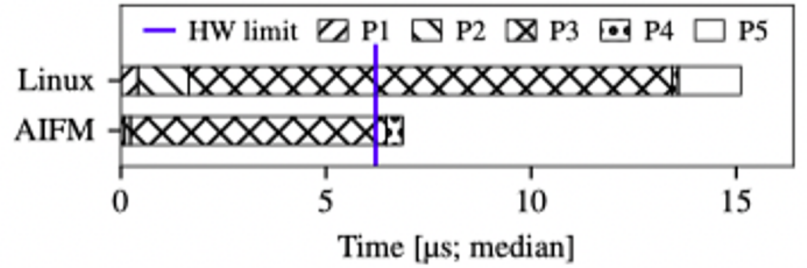


Challenge 2: High Kernel Overheads.

- Expensive page faults.
- **Busy Polling** for in-kernel net I/O => burn CPU cycles.

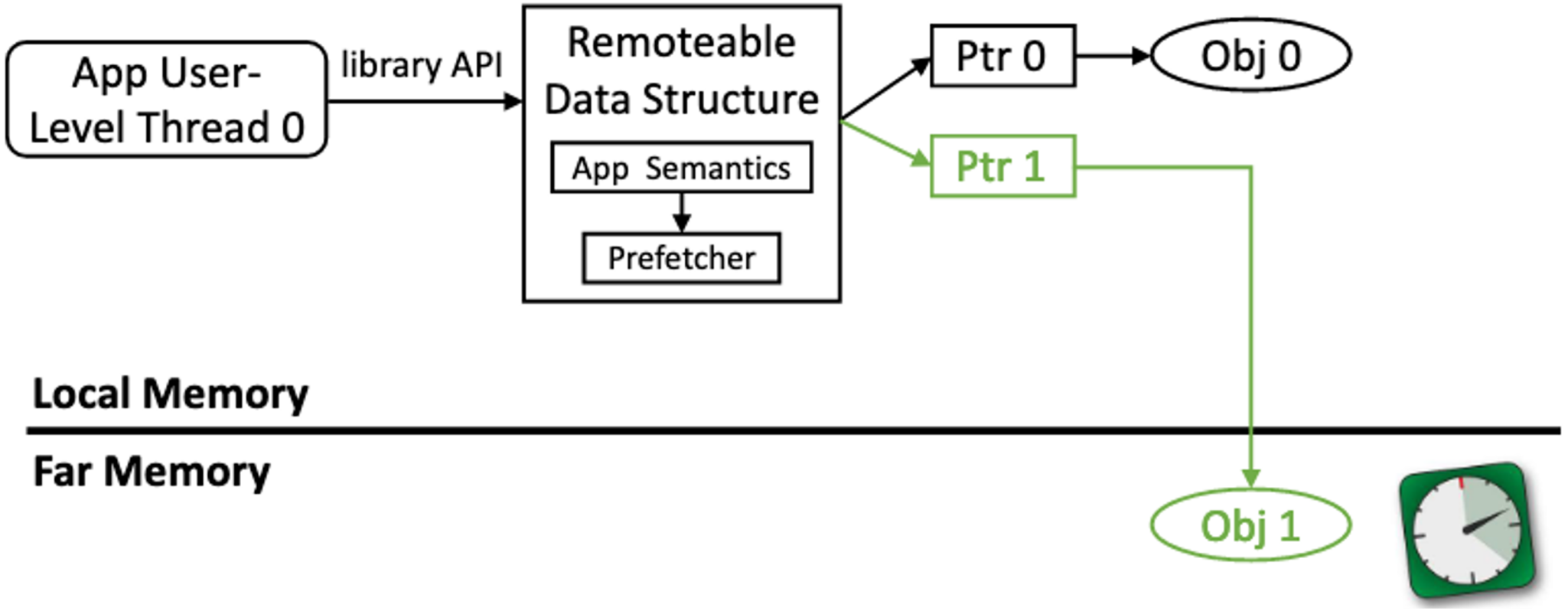


Phase	Linux Kernel Swapping	AIFM
P1	Page fault, trap to kernel	Deref far pointer, issue I/O
P2	Lock, get PTE, allocate page frame, allocate swap cache entry	Lightweight context-switch
P3	Issue read I/O, spin, insert PFN in global LRU list	Run another green thread
P4	cgroup accounting, reclaim memory if past limit	I/O completion, context-switch back
P5	Set page mapping, unlock	—



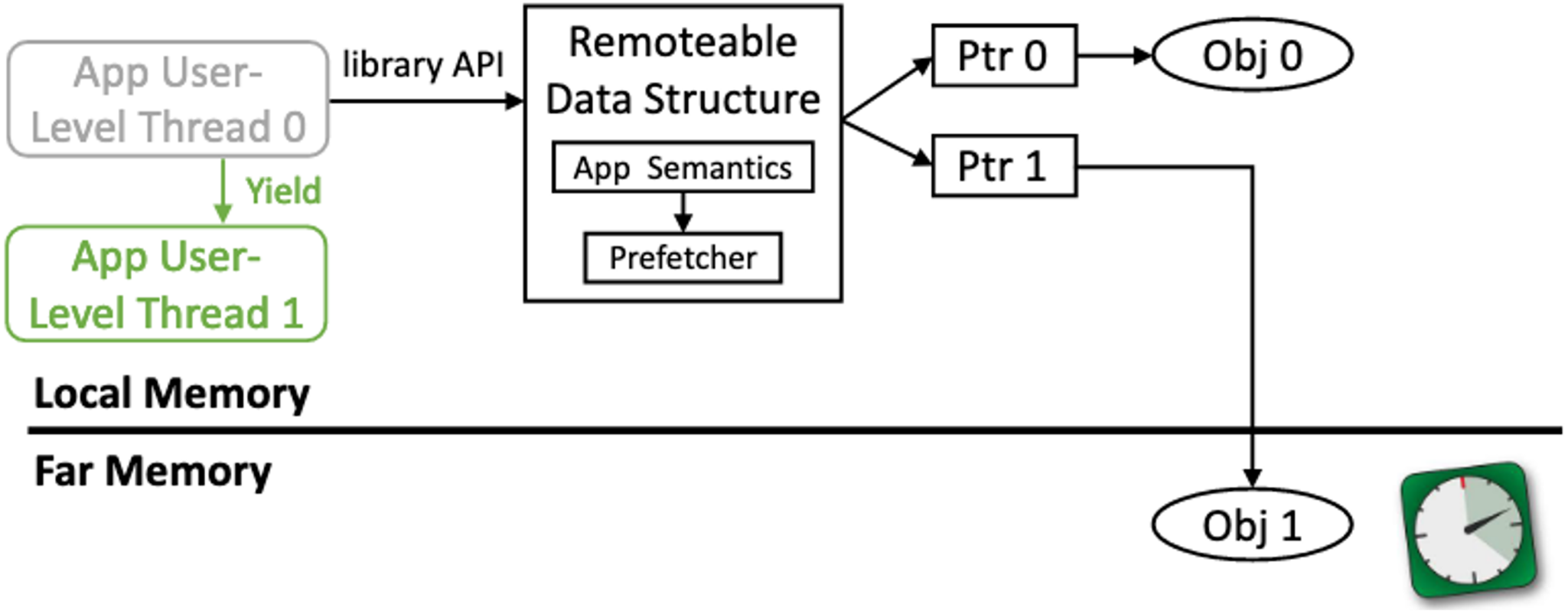
To solve challenge 2, kernel overheads

- Hide Remote Access Latency



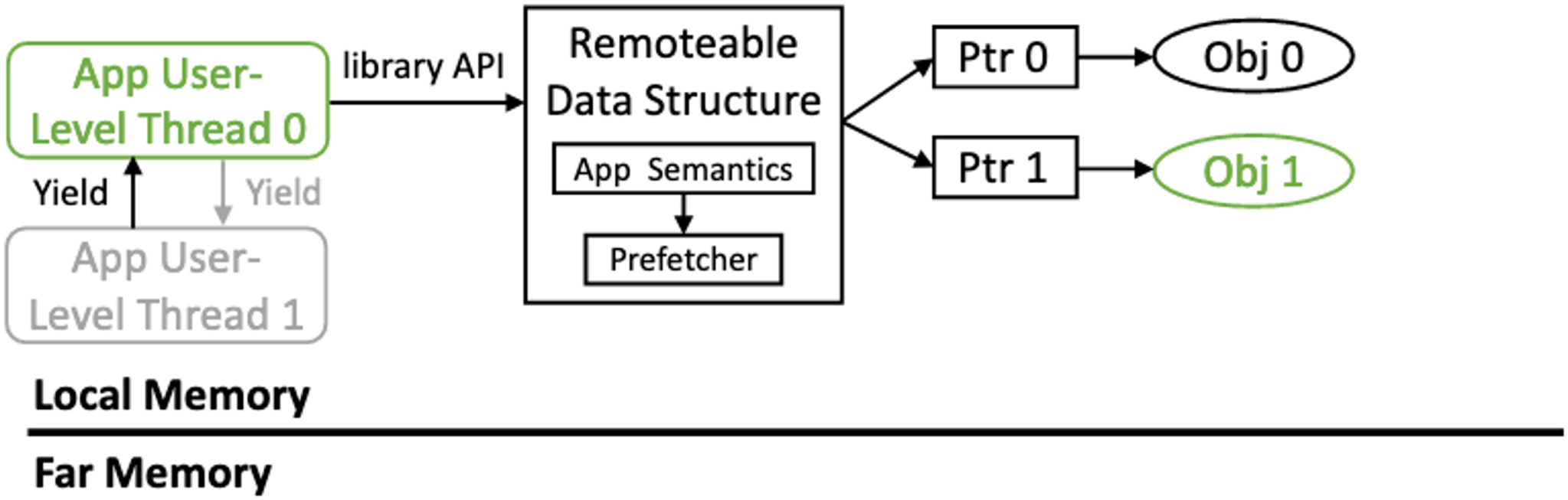
To solve challenge 2, kernel overheads

- Hide Remote Access Latency



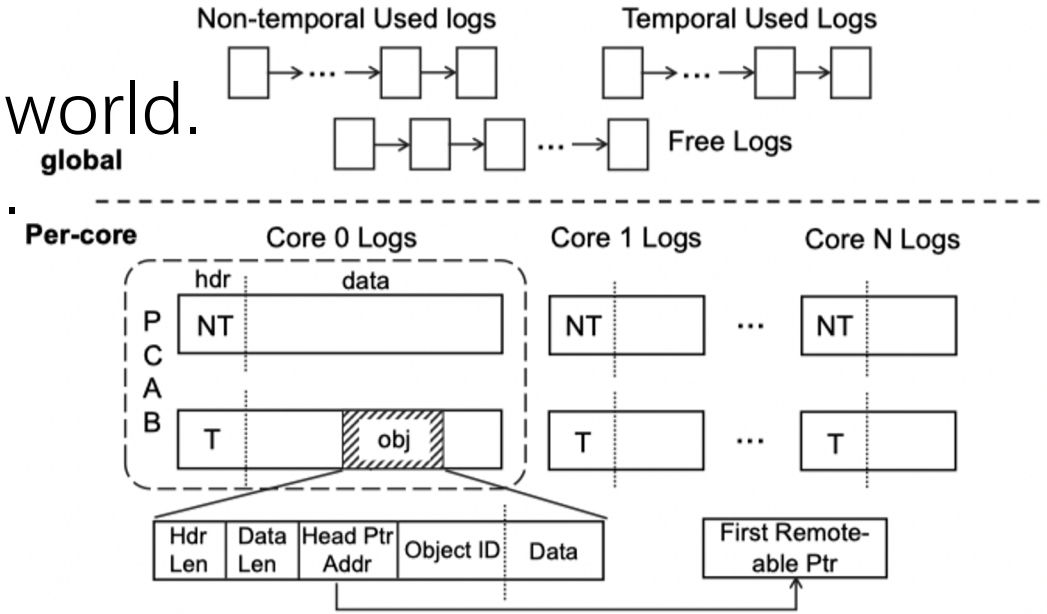
To solve challenge 2, kernel overheads

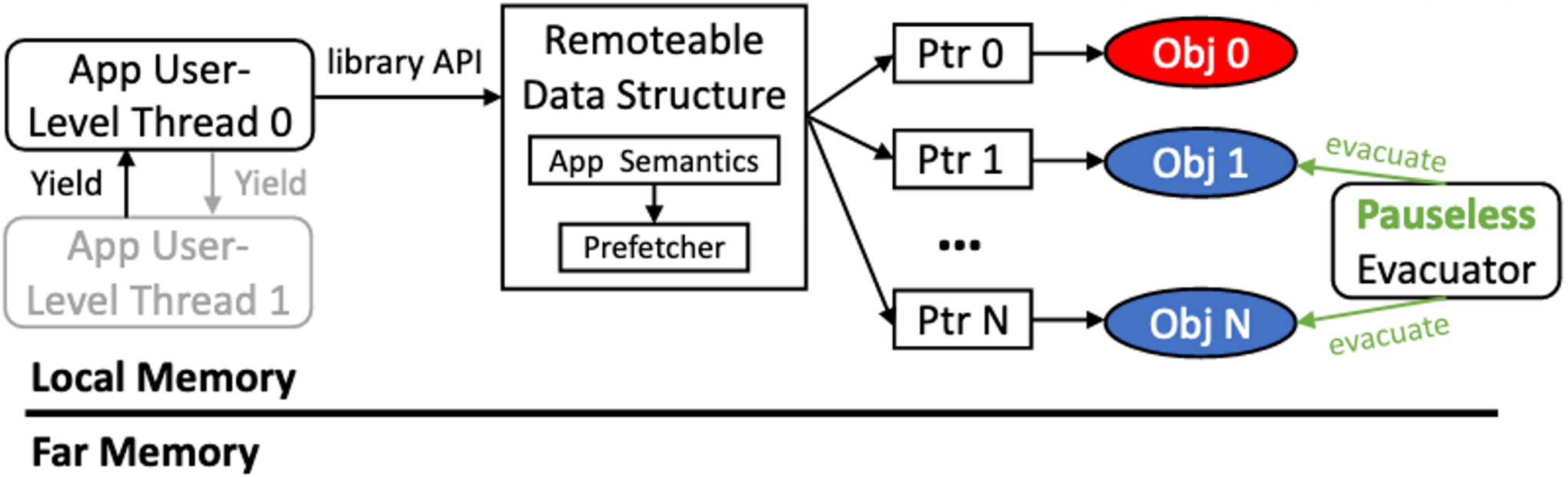
- Hide Remote Access Latency



To solve challenge 3: Impact of Memory Reclamation

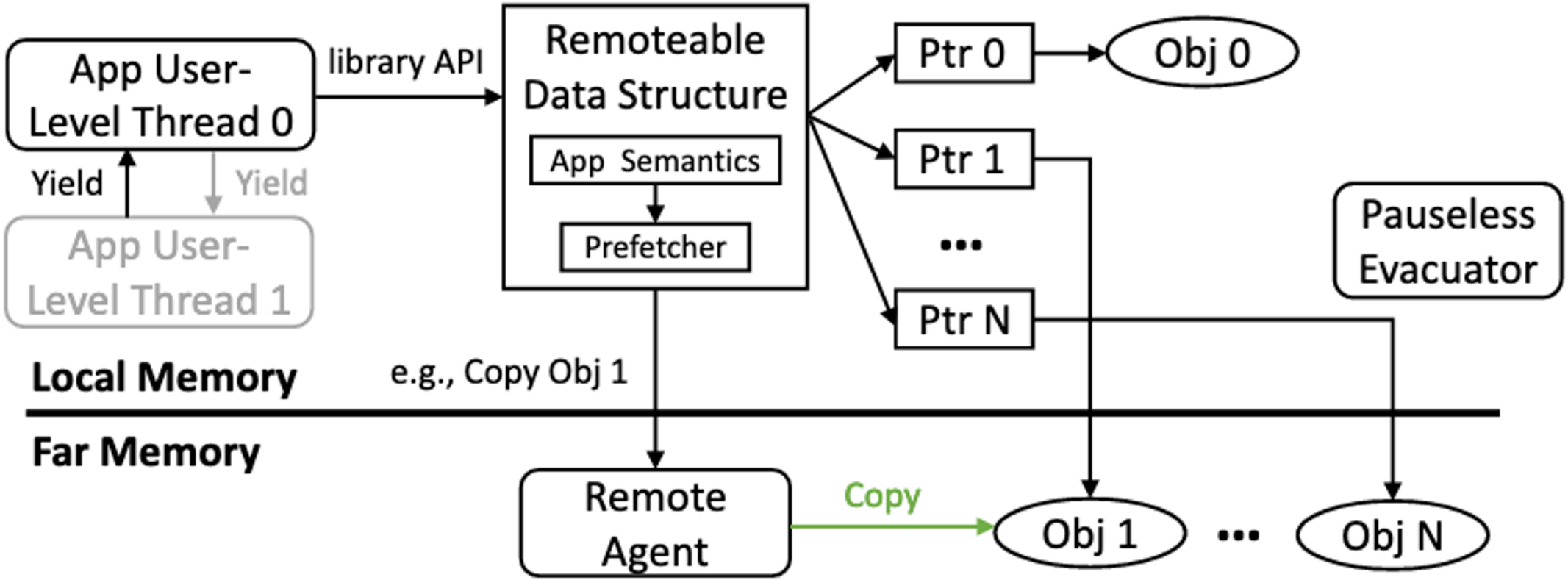
- Memory reclaiming may pause all the world.
- Based on **Log-structured memory[1]**.
  - 1. Log Selection Phase.
  - 2. Concurrent Marking Phase.
  - 3. Evacuator Waiting Phase.
  - 4. Concurrent Evacuation Phase.





To solve challenge 4: network BW < DRAM BW

- It seems to implement a small **remote agent** at the interface of far memory node. (**Not clearly**)



Conclusion

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

Discussion

- Disaggregated Memory with Multiple Nodes
  - For each node, all the other nodes are Far Memory Pool.
  - We can only pass the MPK to any other.
  - In the end, each node gets its hot data in its local memory.
- Prefetcher
  - No more help from the programmer.
  - Pick the dataflow from disassembling and learn the suitable prefetch approach.