UCLA Samueli School of Engineering

Mako: A Low-Pause, High-Throughput Evacuating Collector for Memory-Disaggregated Datacenters

Haoran Ma, Shi Liu, Chenxi Wang, Yifan Qiao, Michael D. Bond, Stephen M. Blackburn, Miryung Kim, Guoqing Harry Xu

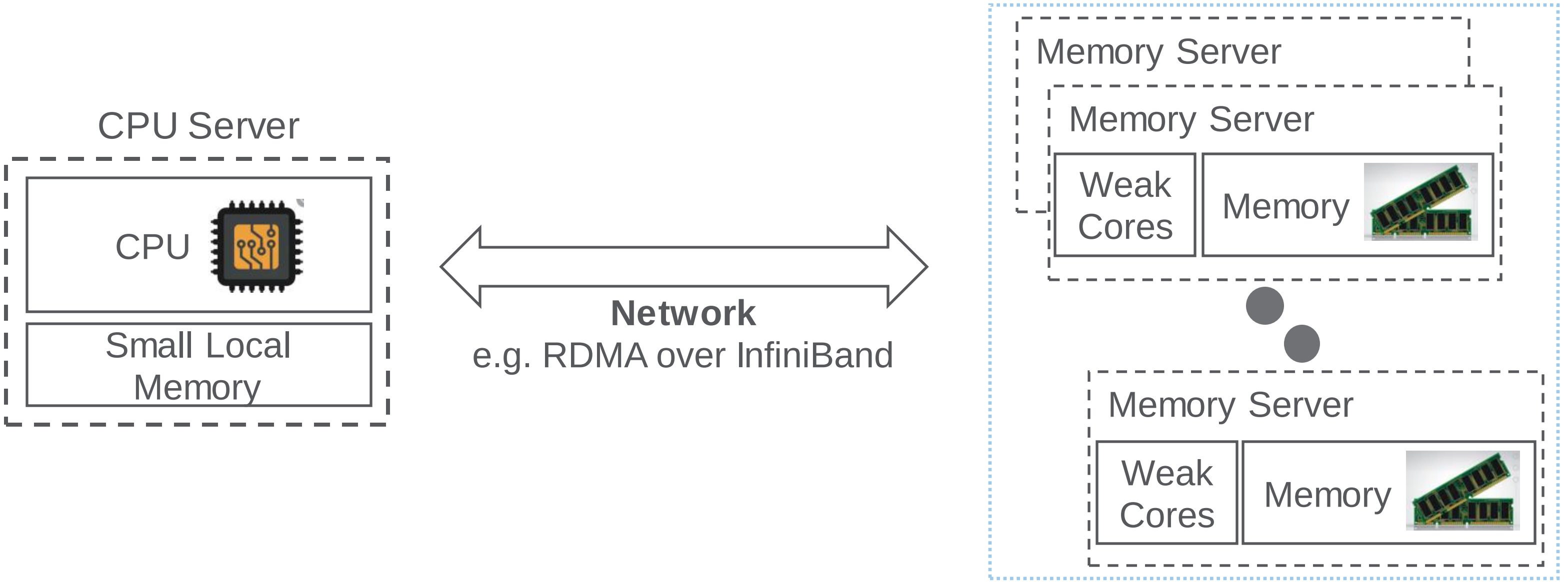
Australian National University

Memory Capacity Bottleneck in Datacenters

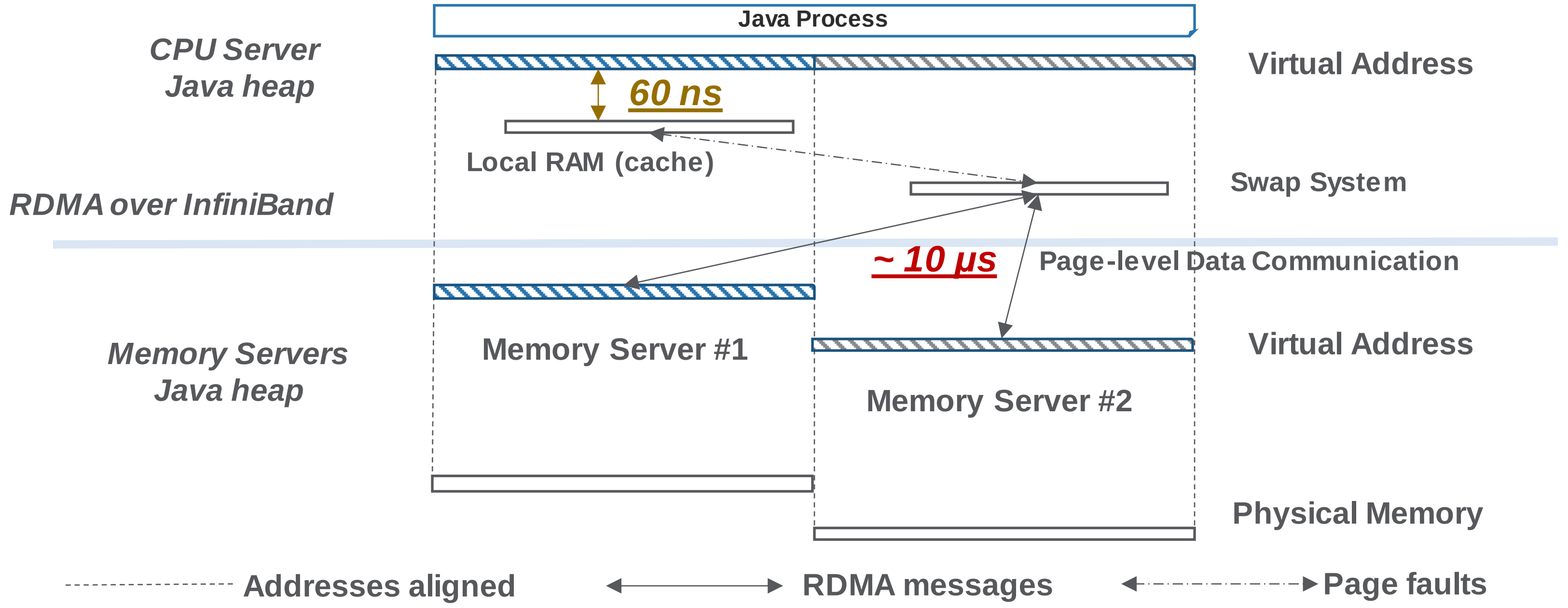
Growing imbalance between processor computation and memory capacity

Memory underutilization in datacenters

Memory Disaggregation

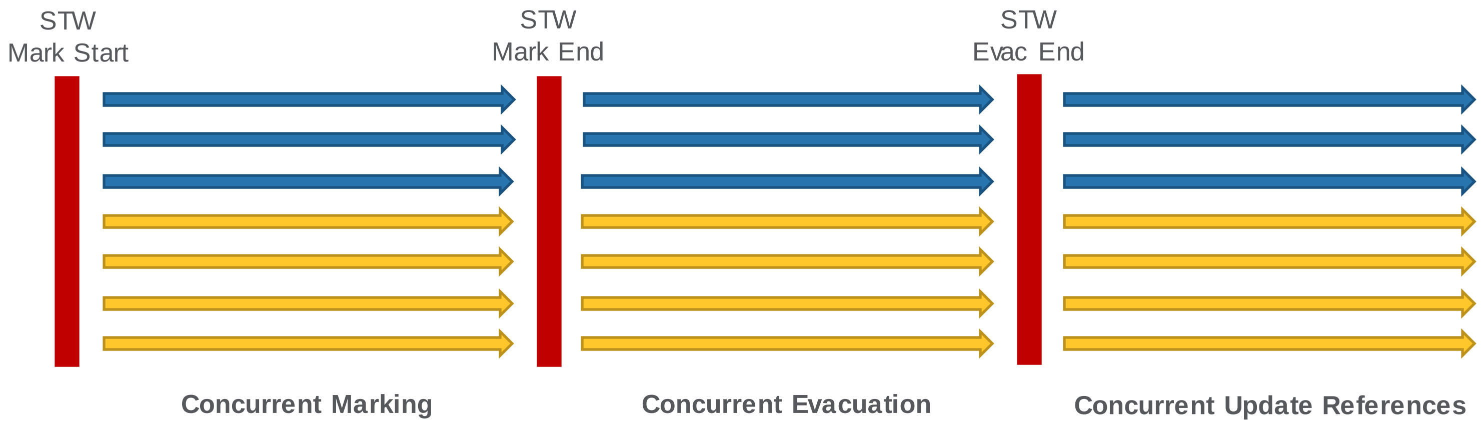


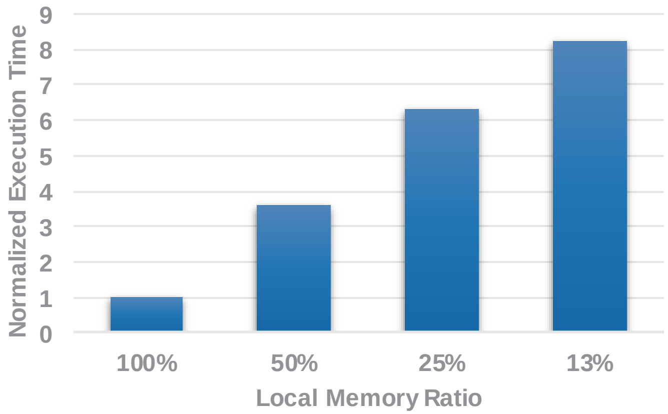
Java Heap Structure

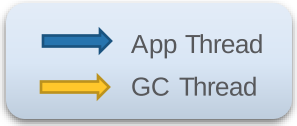


UCLA Samueli School of Engineering

Concurrent GC



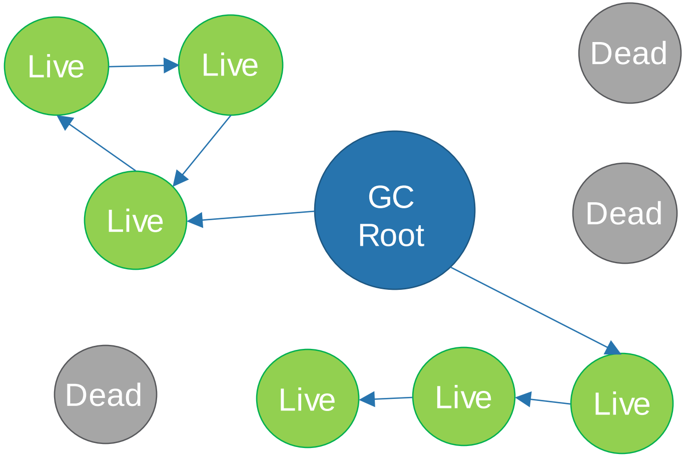




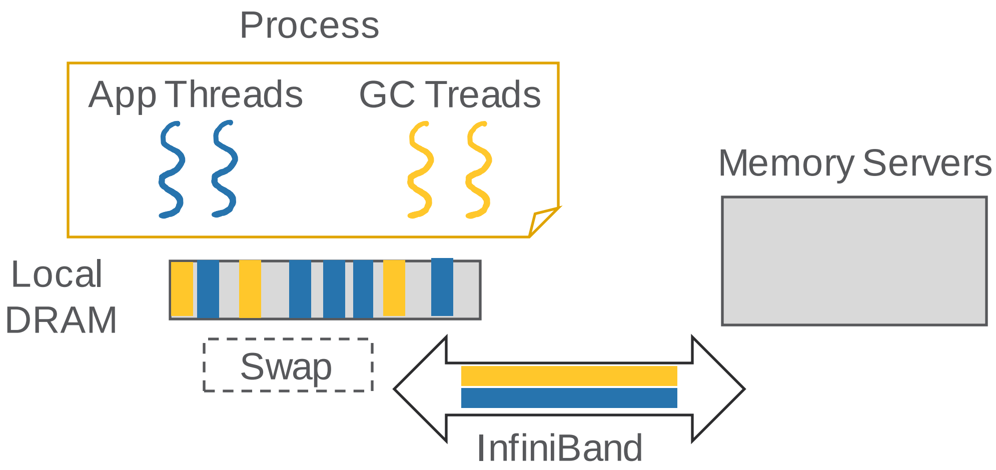
UCLA Samueli School of Engineering

Concurrent GC

No Locality



Resource Contention



Can we move concurrent marking and evacuation to memory servers?

Can we move concurrent marking and evacuation to memory servers?

Reduced Interference

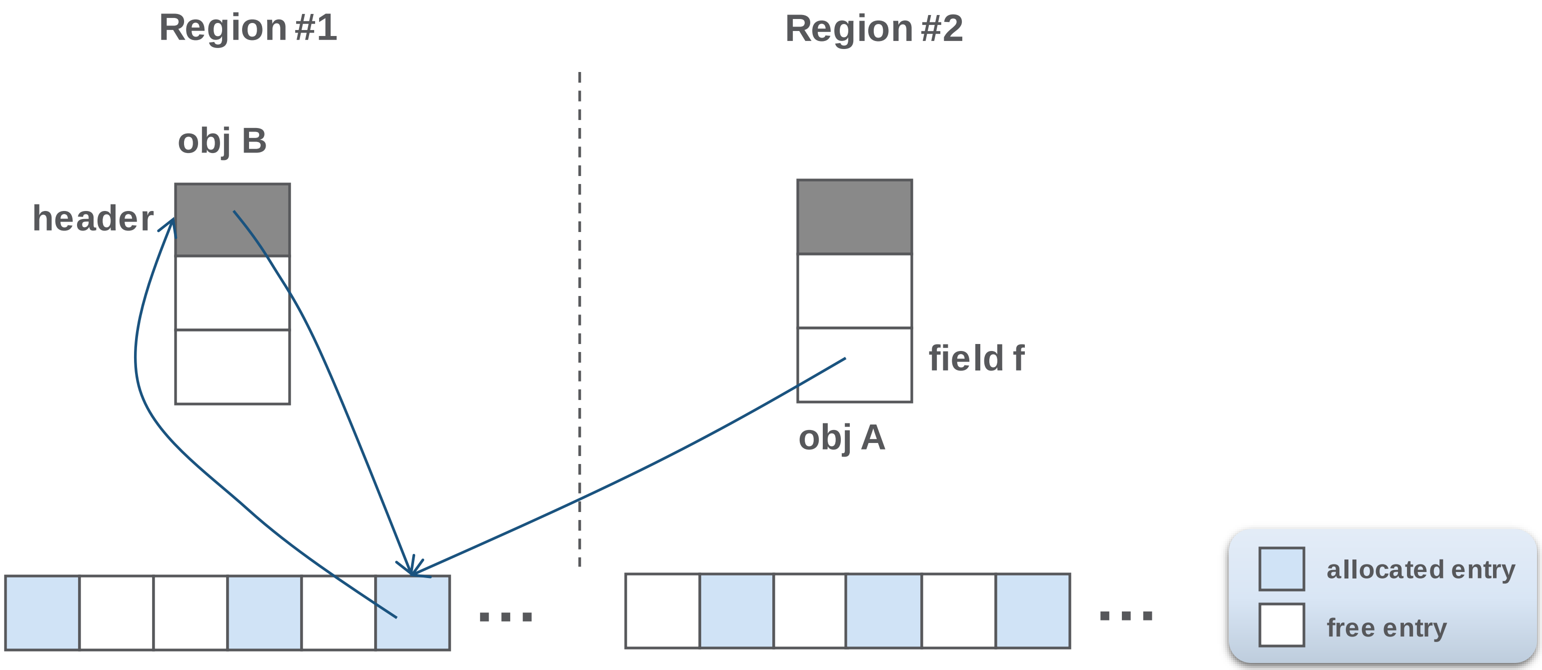
Near-data Computing

Challenges

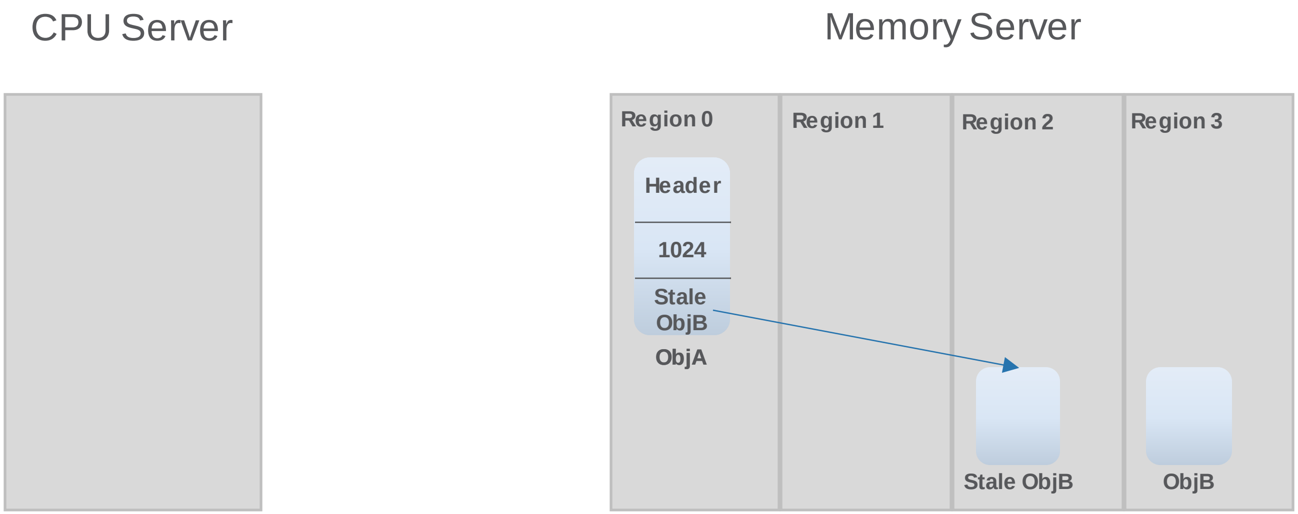
No efficient way to enforce memory coherence between the CPU and memory servers

- Concurrent Updating Reference
  - Problem: Overwritten updated references
- Concurrent Evacuation
  - Problem: Lost forwarding address

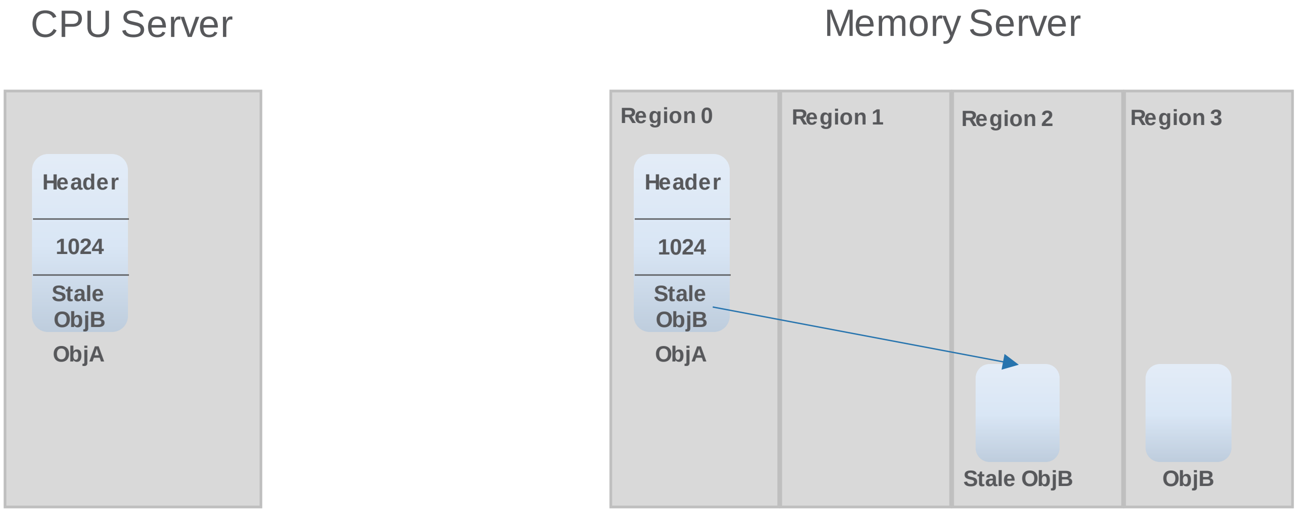
Heap Indirection Table



Problem #1: Concurrent Updating References

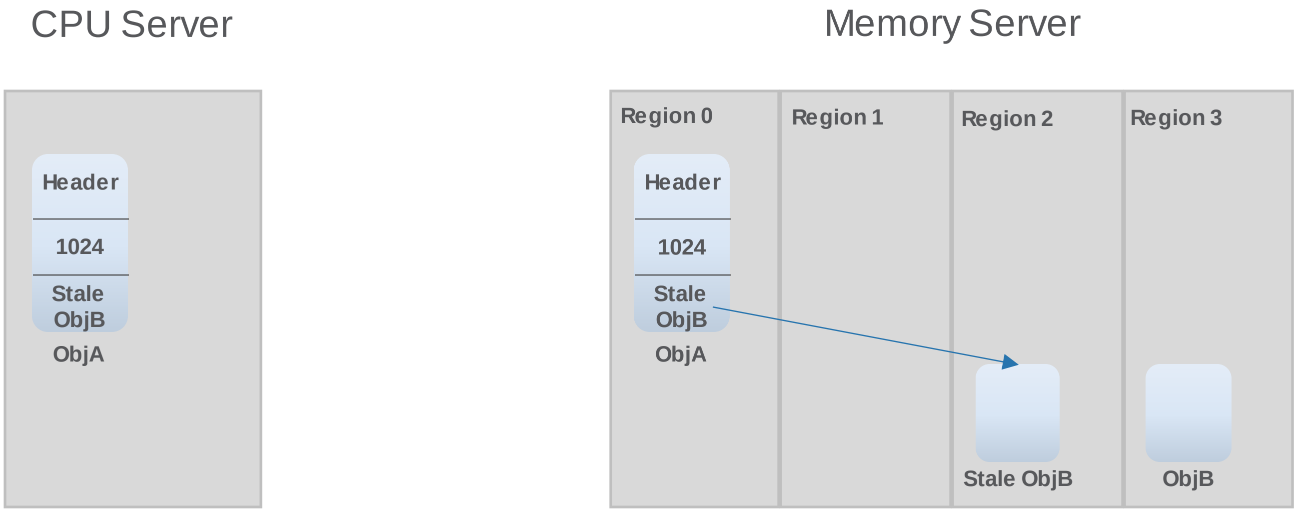


Problem #1: Concurrent Updating References

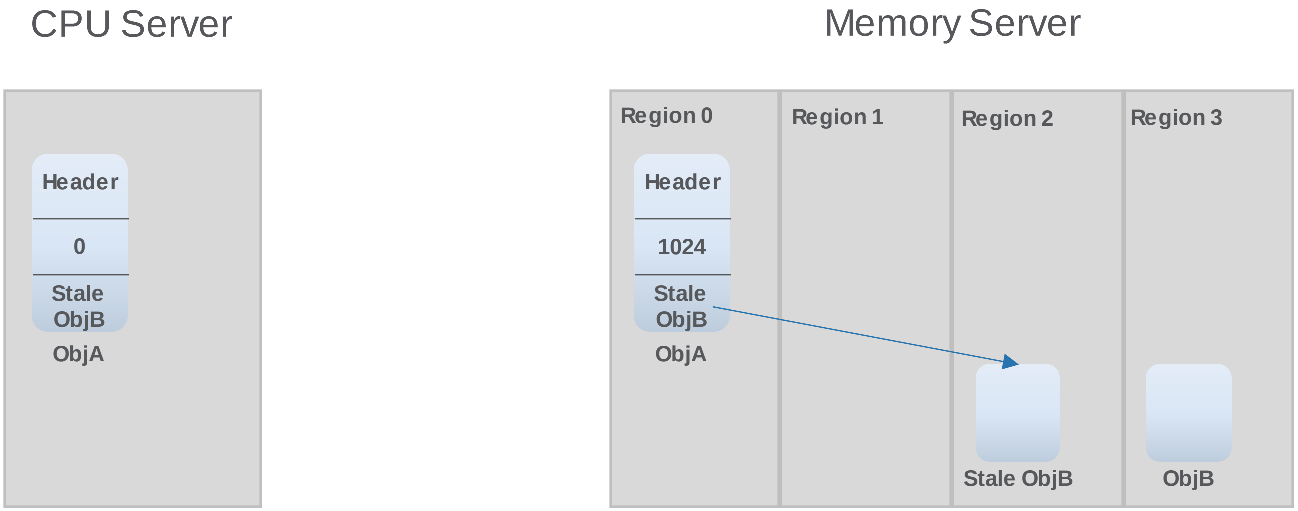


UCLA Samueli School of Engineering

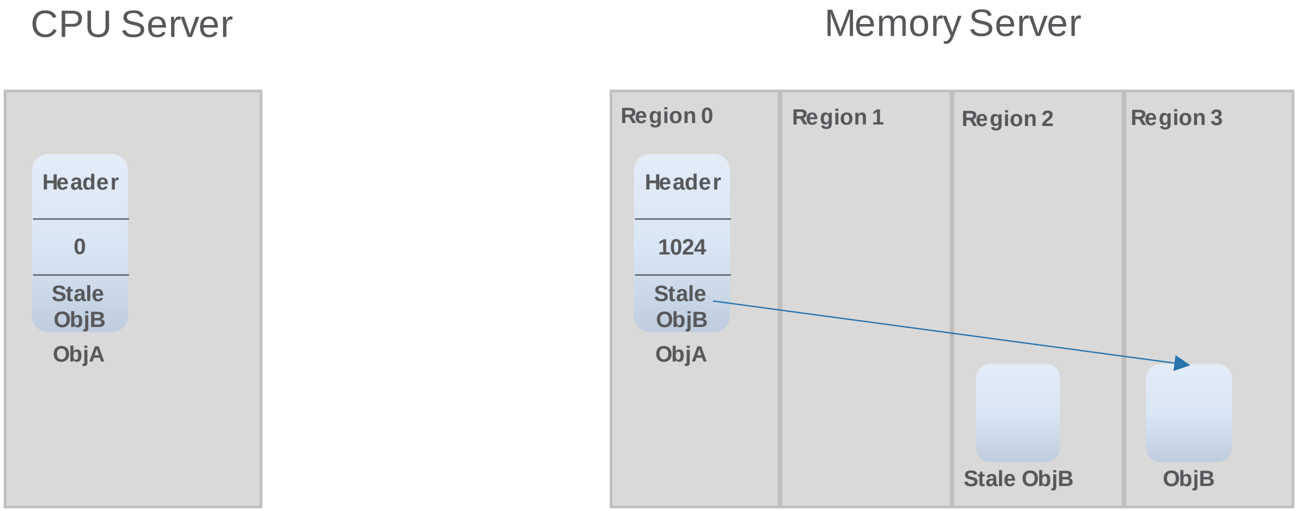
Problem #1: Concurrent Updating References



Problem #1: Concurrent Updating References

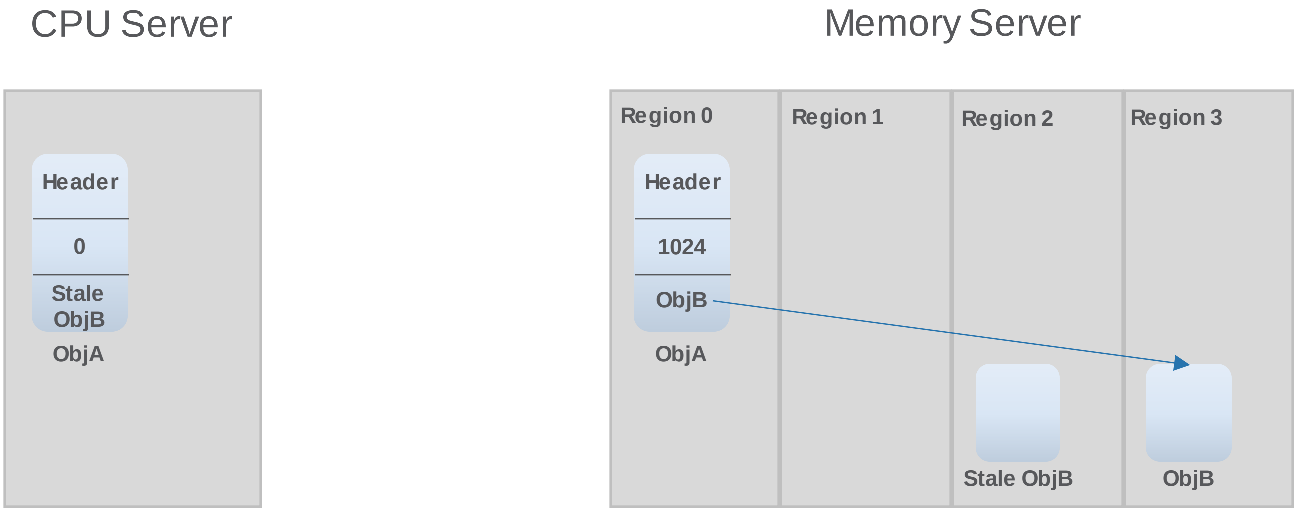


Problem #1: Concurrent Updating References



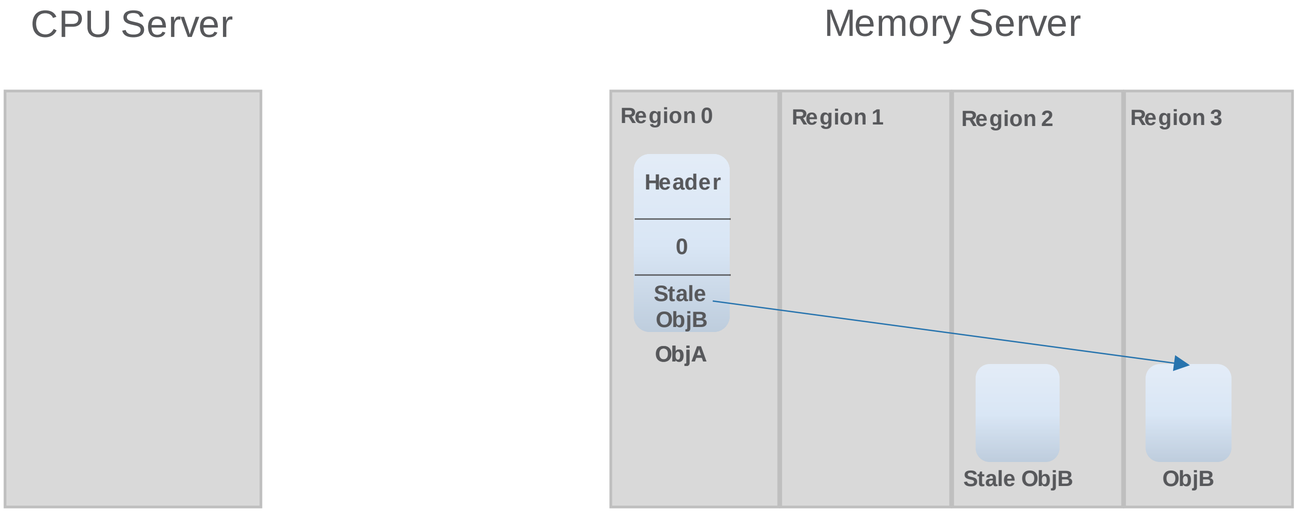
UCLA Samueli School of Engineering

Problem #1: Concurrent Updating References



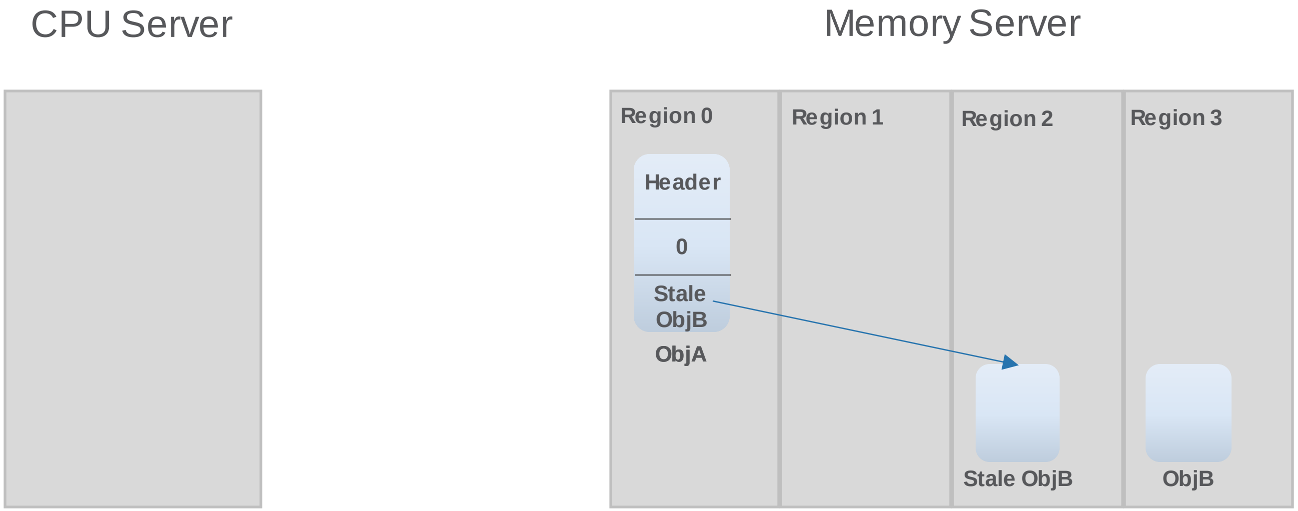
UCLA Samueli School of Engineering

Problem #1: Concurrent Updating References



UCLA Samueli School of Engineering

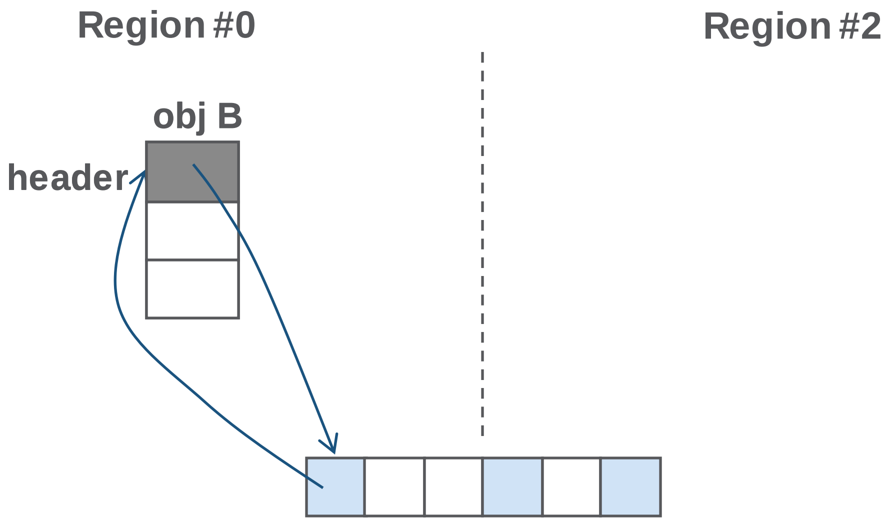
Problem #1: Concurrent Updating References



UCLA Samueli School of Engineering

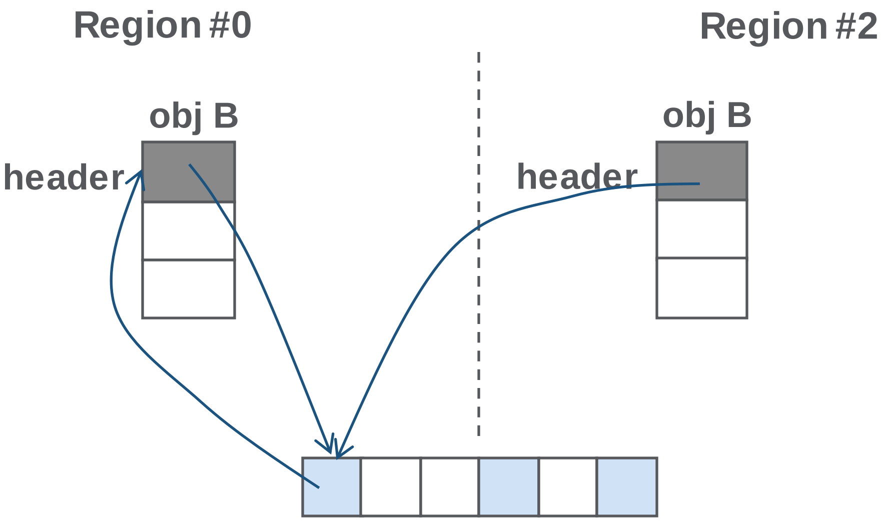
Solution #1

- Eliminate the need to directly update pointers at both the CPU and memory servers



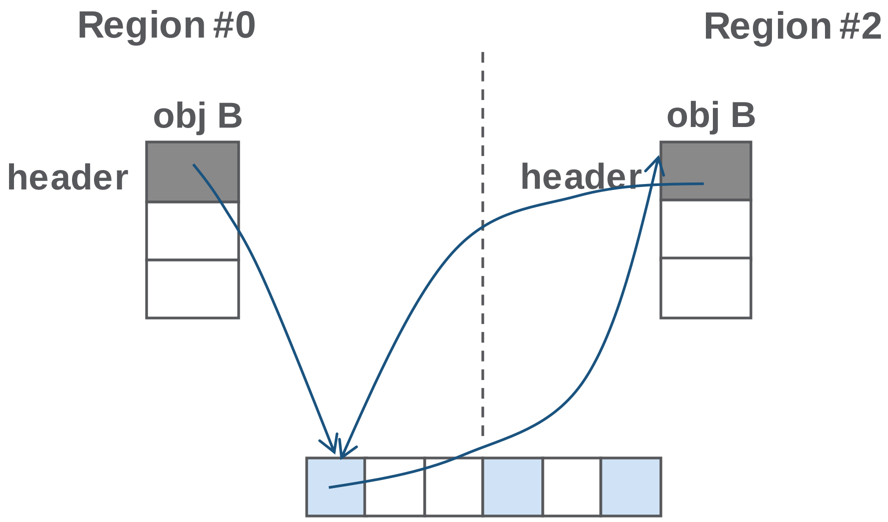
Solution #1

- Eliminate the need to directly update pointers at both the CPU and memory servers

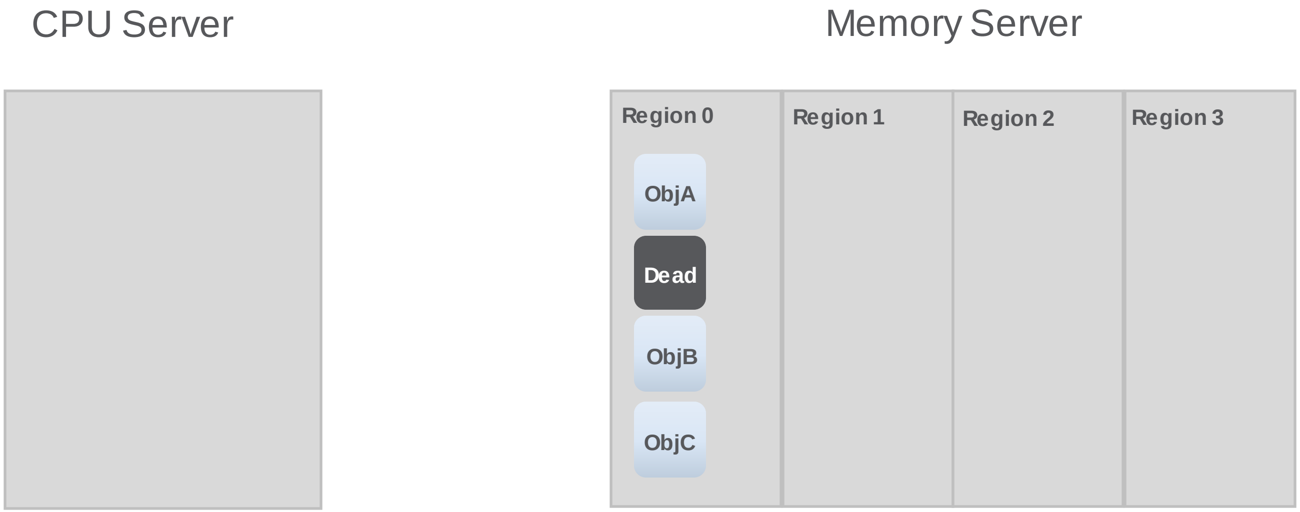


Solution #1

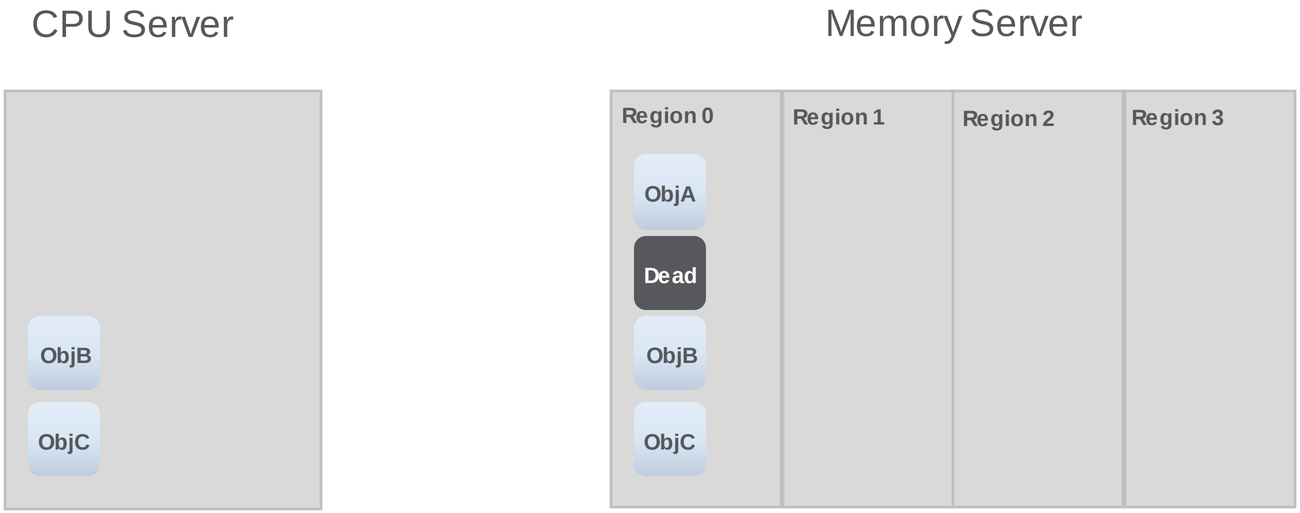
- Eliminate the need to directly update pointers at both the CPU and memory servers



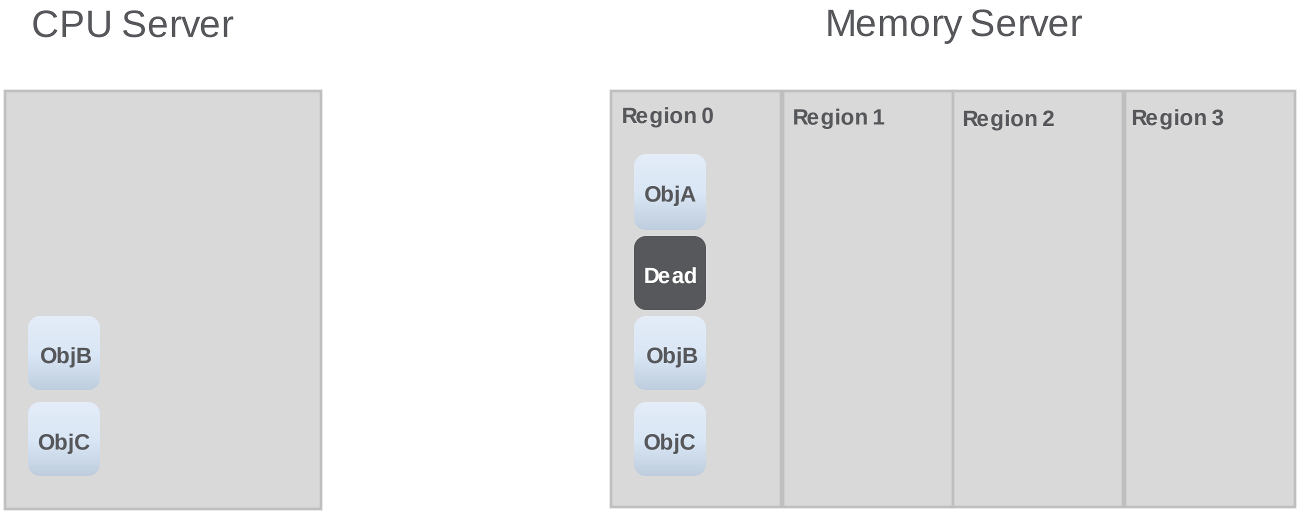
Problem #2: Concurrent Evacuation



Problem #2: Concurrent Evacuation

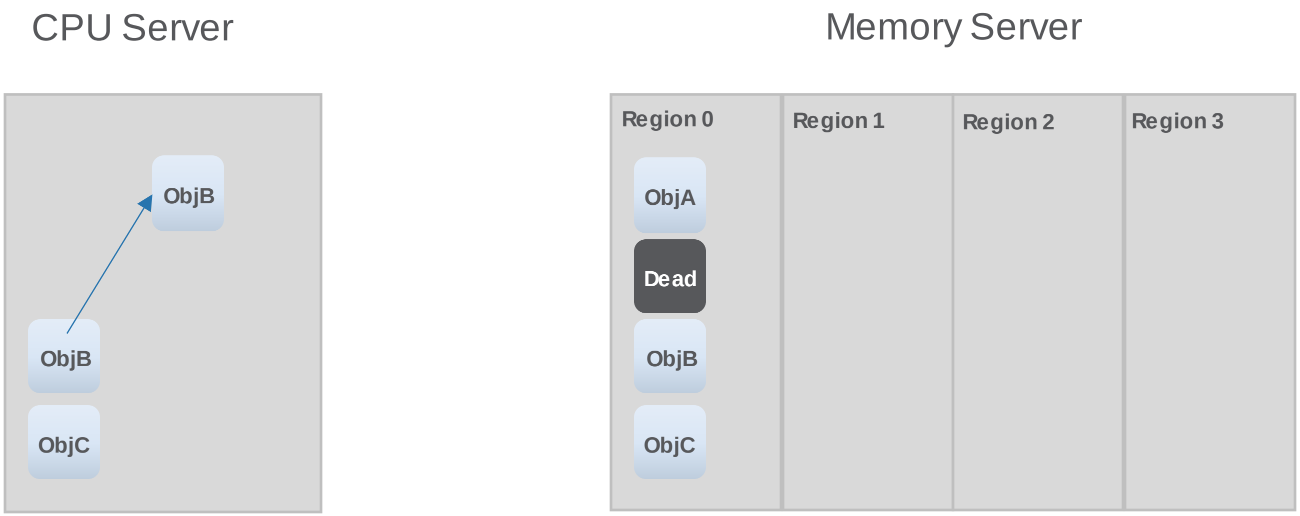


Problem #2: Concurrent Evacuation



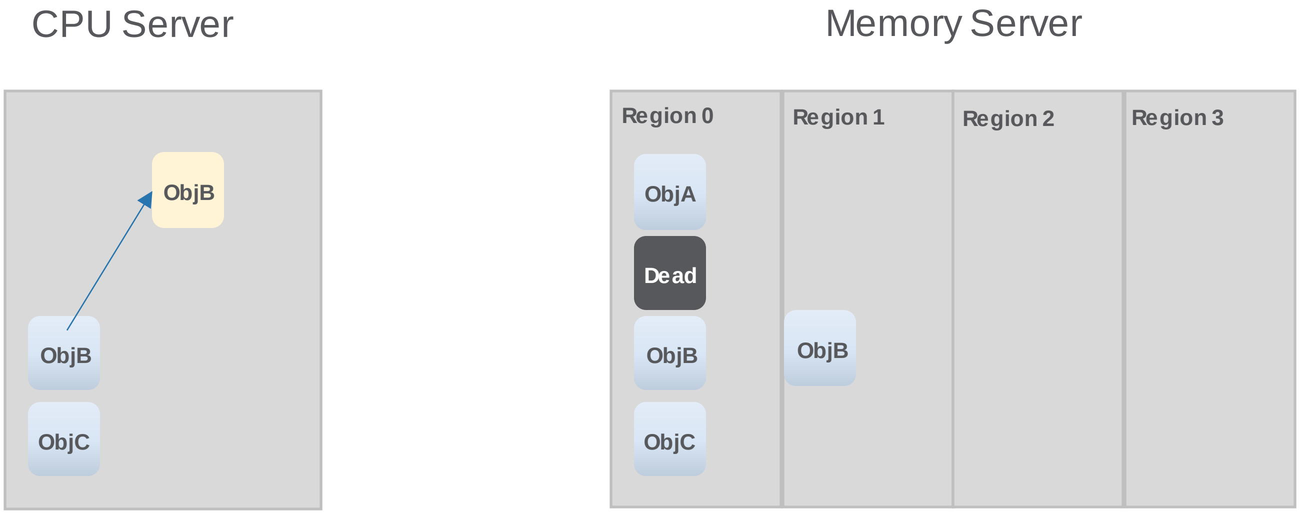
UCLA Samueli School of Engineering

Problem #2: Concurrent Evacuation

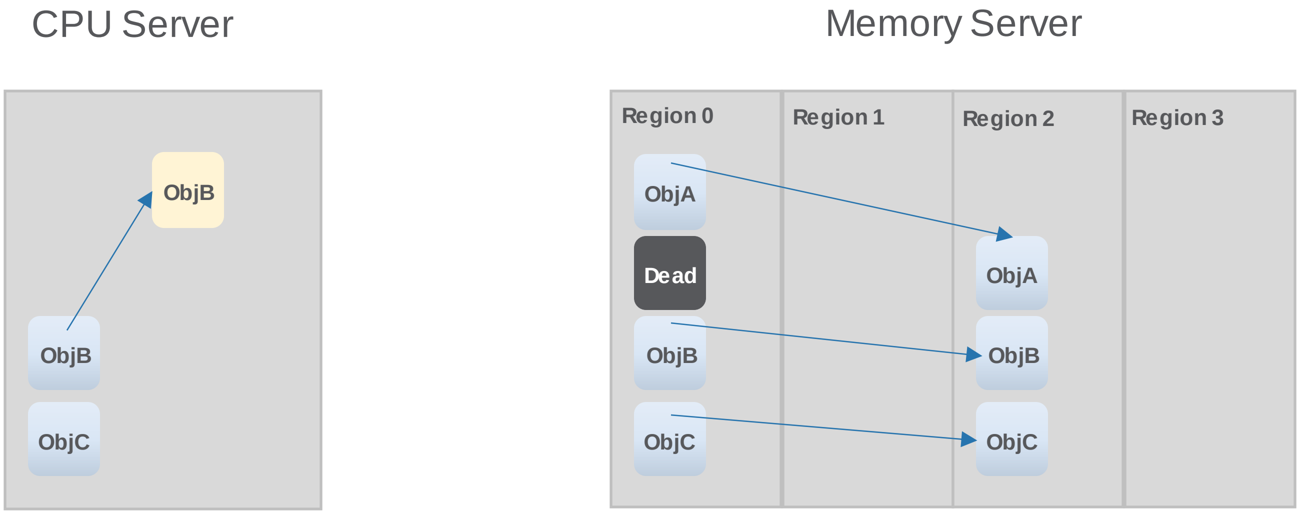


UCLA Samueli School of Engineering

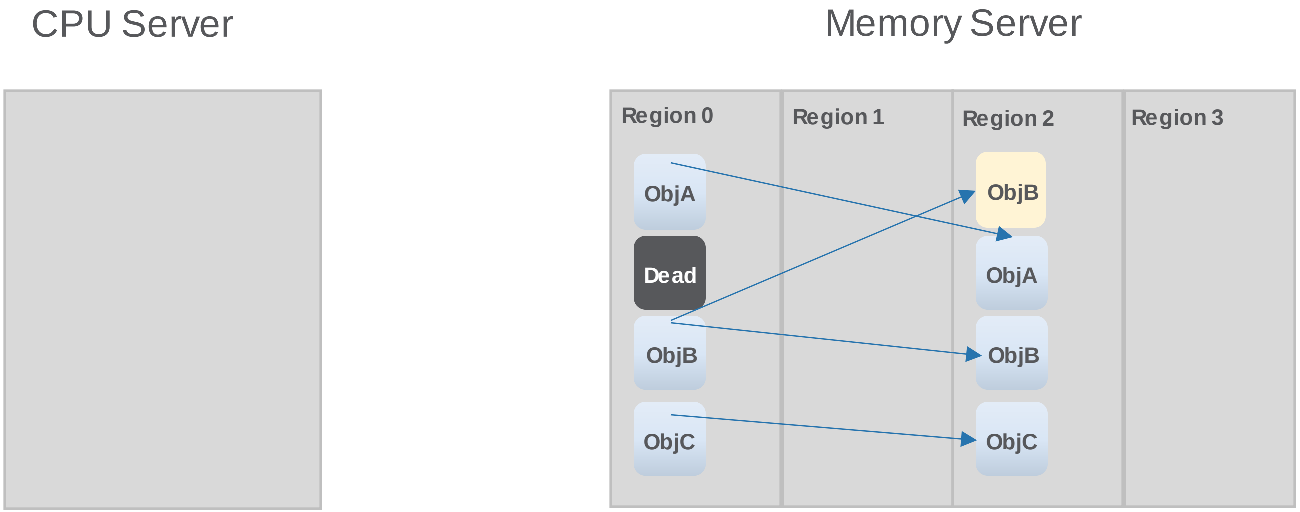
Problem #2: Concurrent Evacuation



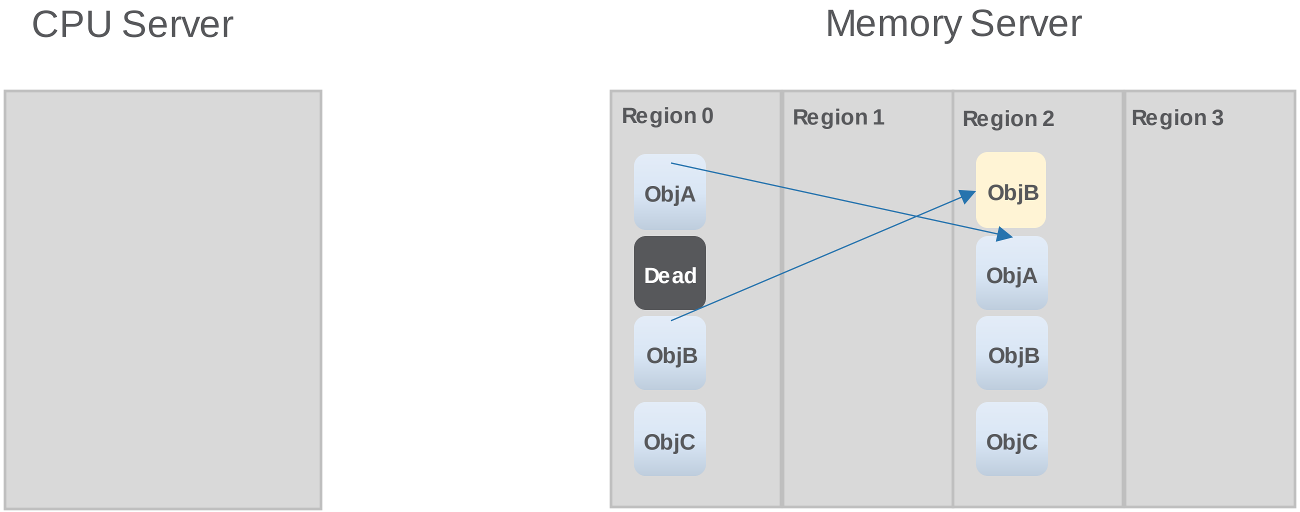
Problem #2: Concurrent Evacuation



Problem #2: Concurrent Evacuation



Problem #2: Concurrent Evacuation



UCLA Samueli School of Engineering

Solution #2

- Causes of Problem #2:
  - JVM accesses data at object-level, while the OS kernel manages data at page-level.
  - Synchronization on each object access would incur too much overhead

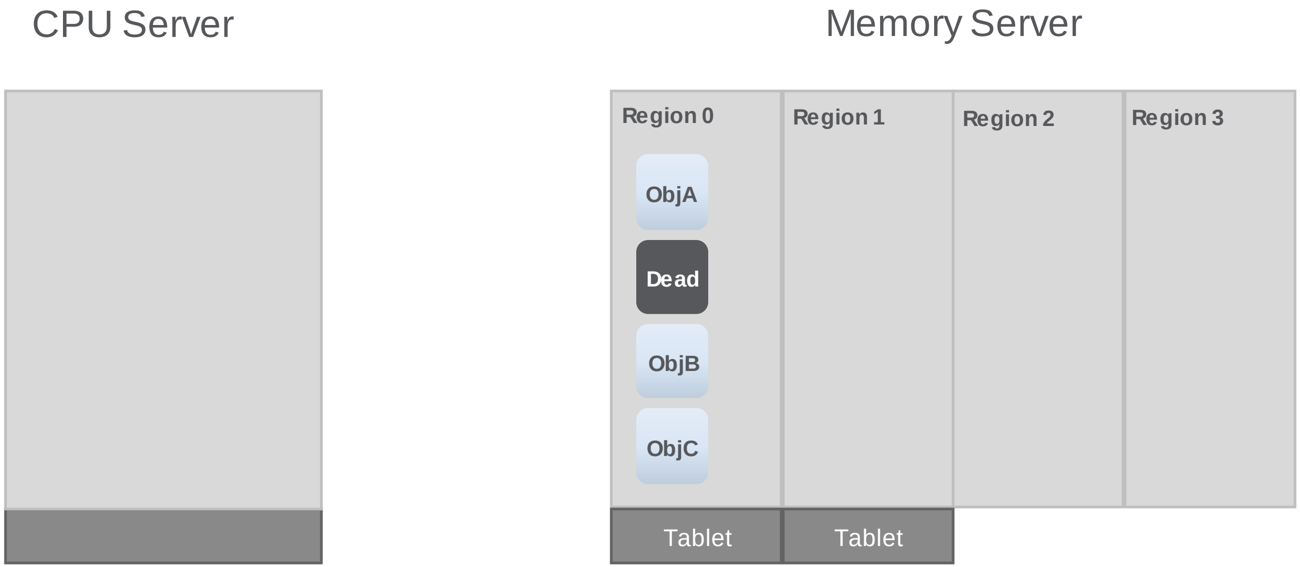
Tablet Lock

↓

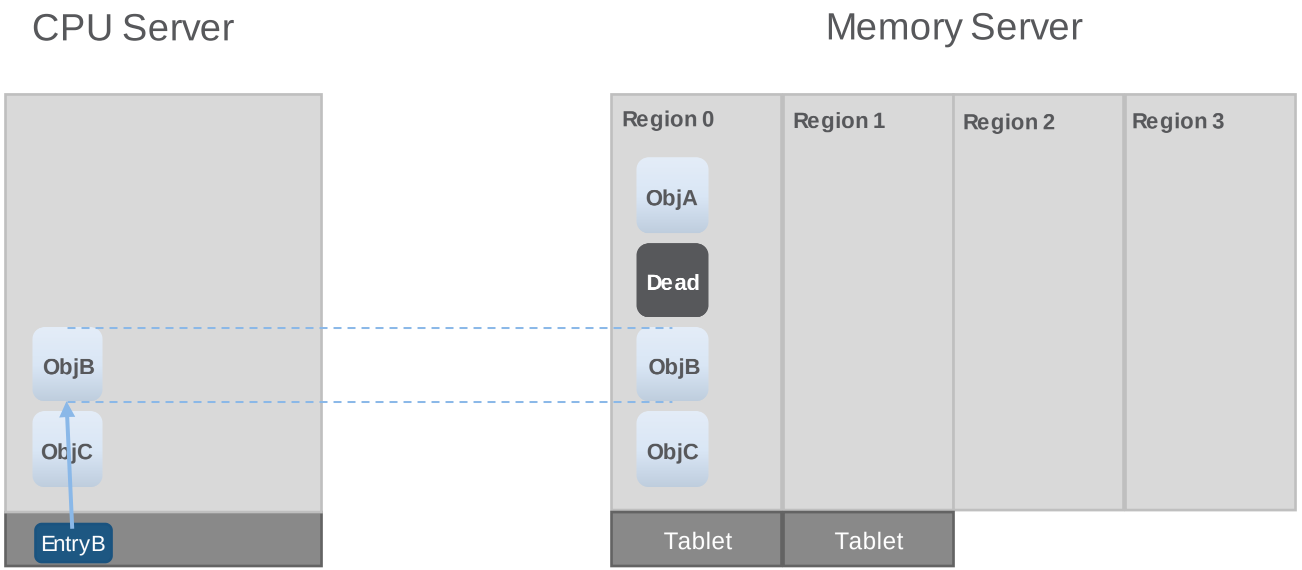
Region-level Synchronization

UCLA Samueli School of Engineering

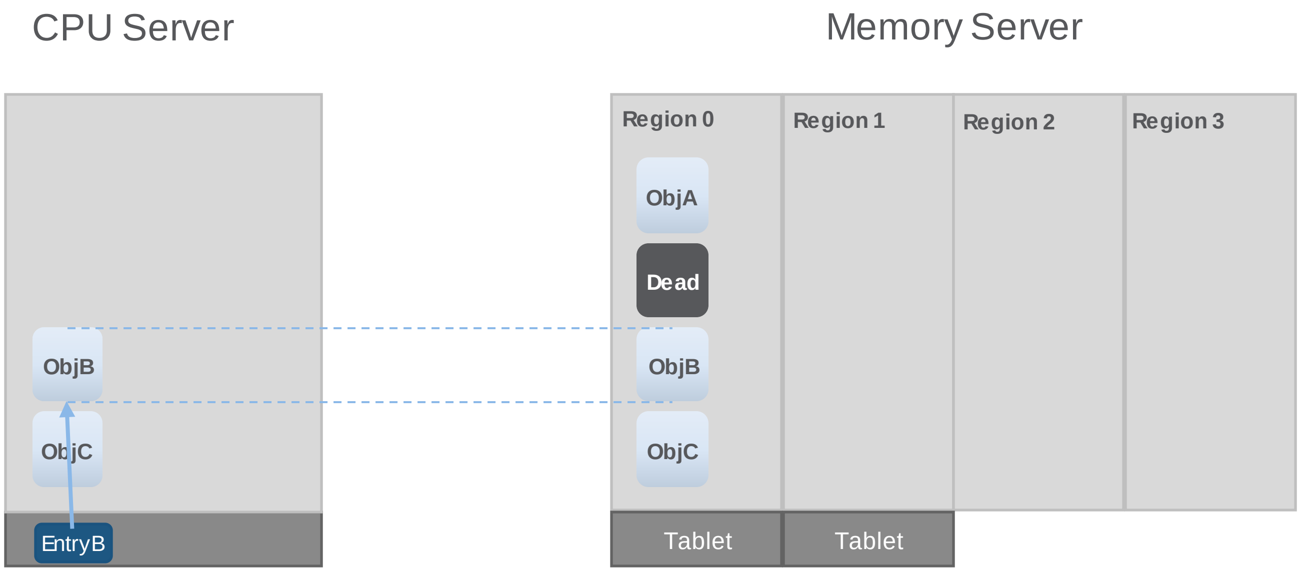
Solution #2


CPU Server
Memory Server
Region 0
Region 1
Region 2
Region 3
ObjA
Dead
ObjB
ObjC
Tablet
Tablet

Solution #2

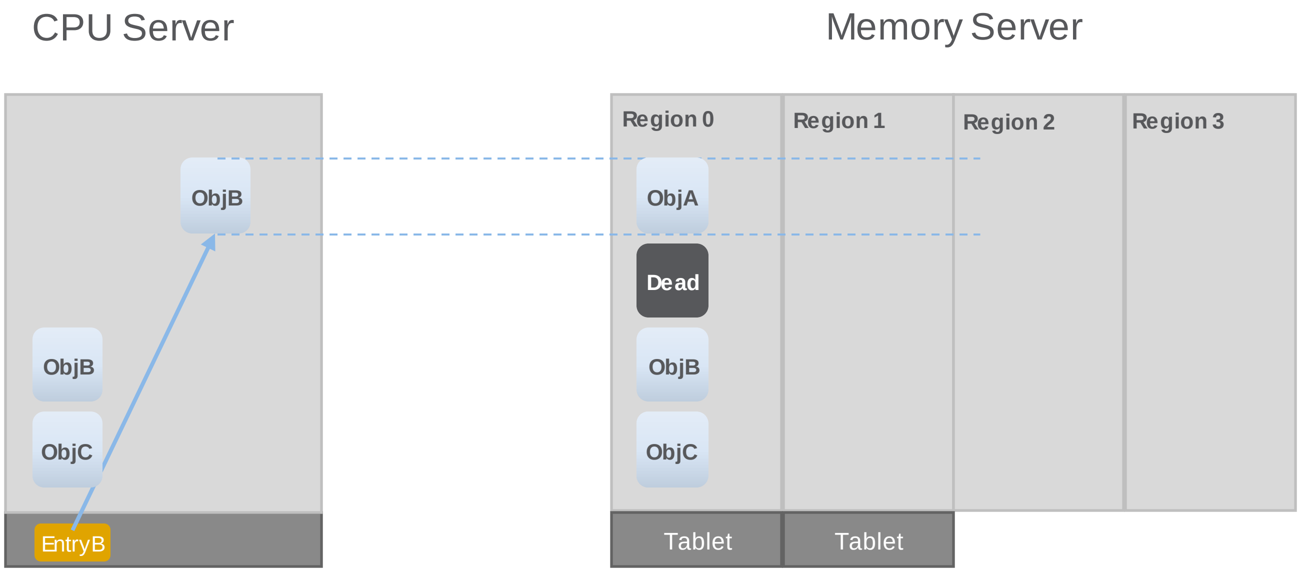

CPU Server
Memory Server
Region 0
Region 1
Region 2
Region 3
ObjA
Dead
ObjB
ObjC
ObjB
ObjC
EntryB
Tablet
Tablet

Solution #2



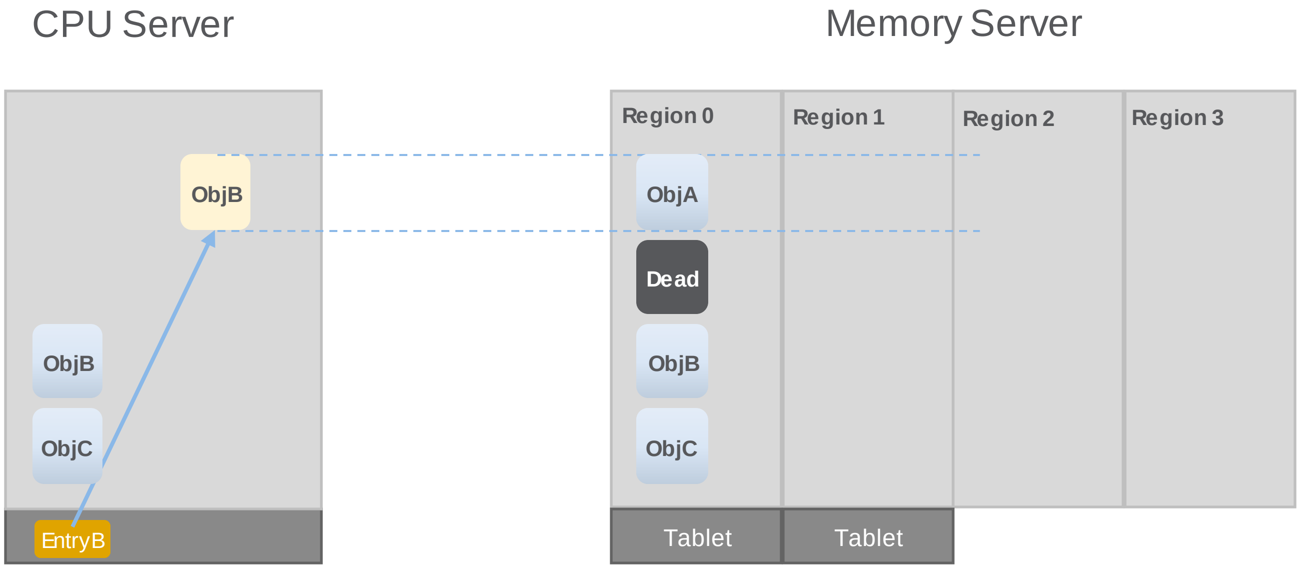
UCLA Samueli School of Engineering

Solution #2



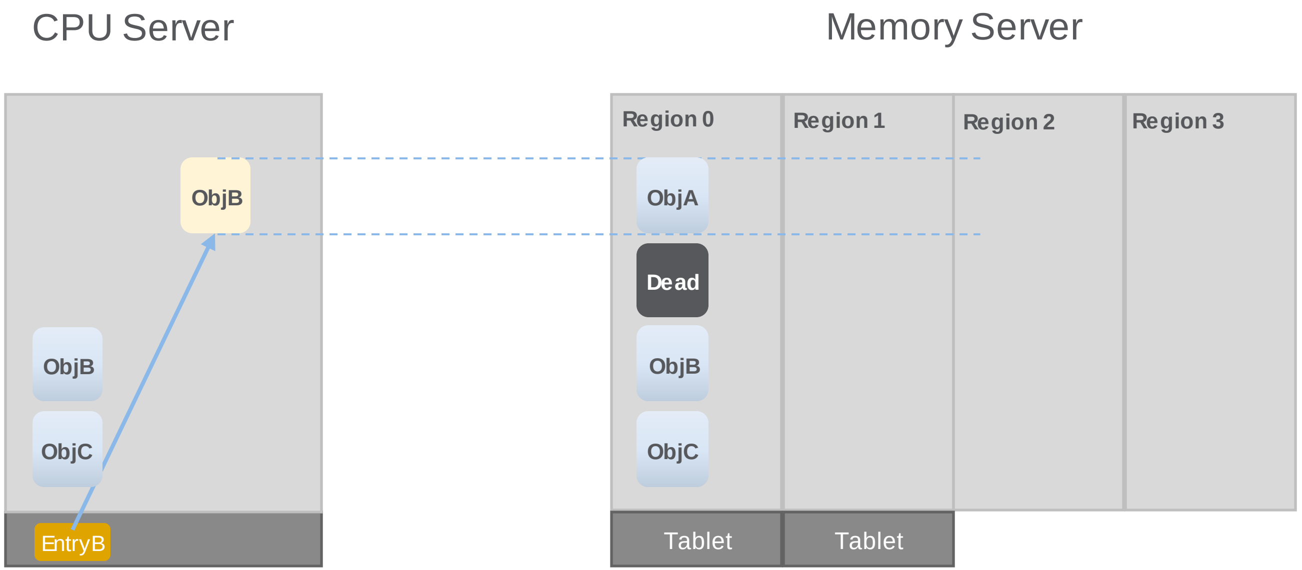
CPU Server
Memory Server
ObjB
ObjB
ObjC
EntryB
Region 0
Region 1
Region 2
Region 3
ObjA
Dead
ObjB
ObjC
Tablet
Tablet

Solution #2

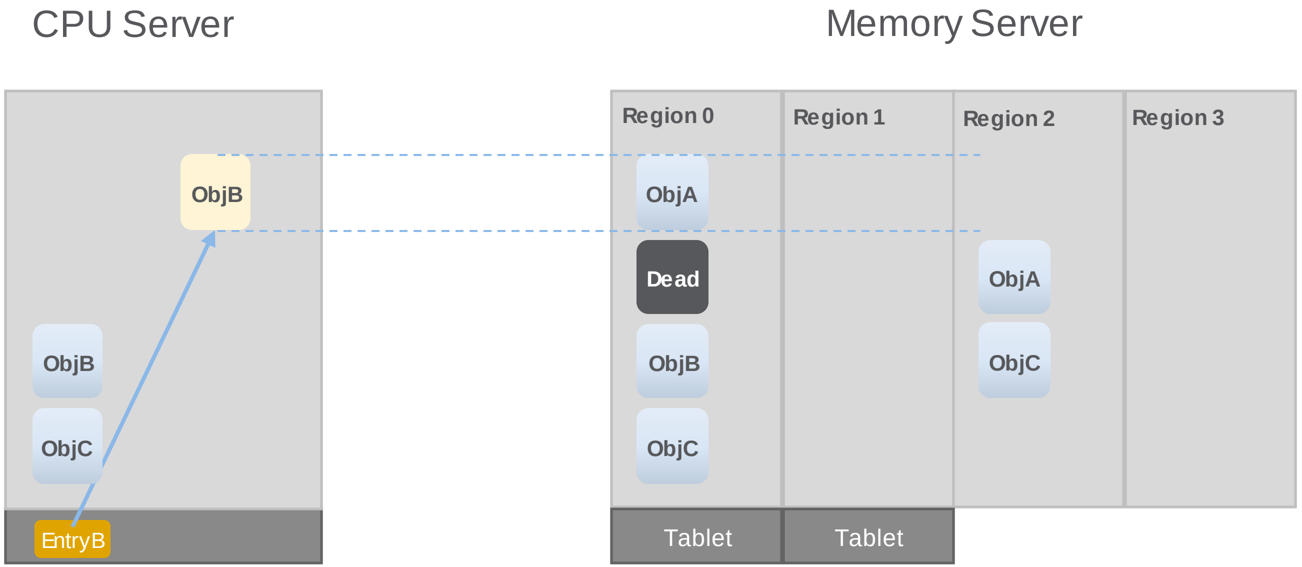


CPU Server
Memory Server
ObjB
ObjB
ObjC
EntryB
Region 0
Region 1
Region 2
Region 3
ObjA
Dead
ObjB
ObjC
Tablet
Tablet

Solution #2

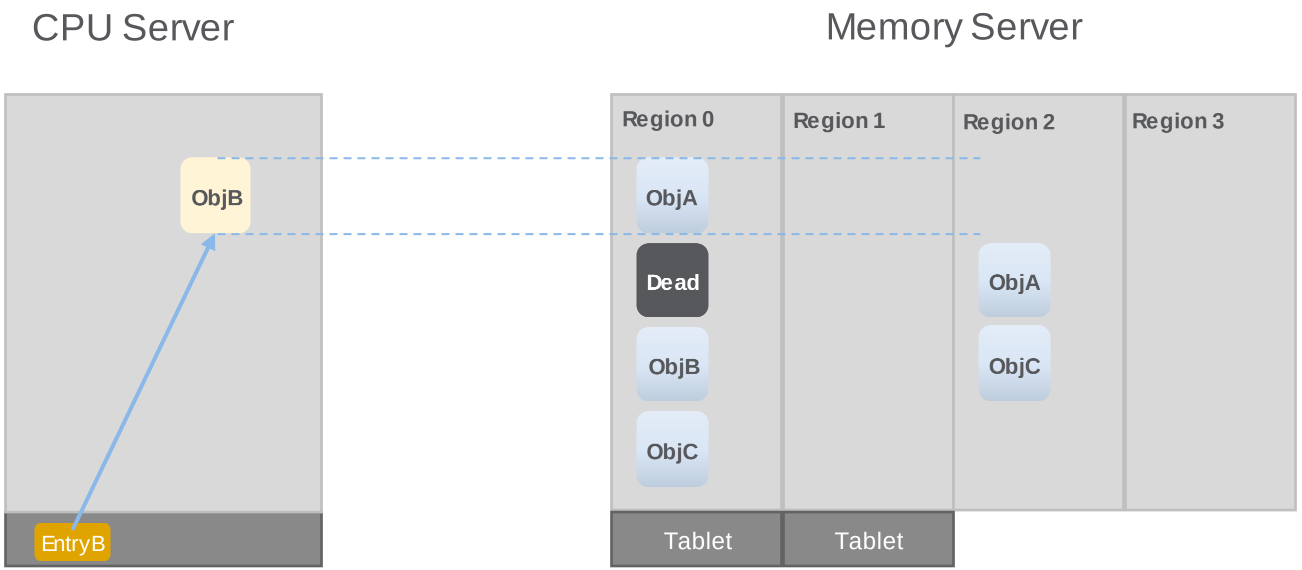

CPU Server
Memory Server
ObjB
ObjB
ObjC
EntryB
Region 0
Region 1
Region 2
Region 3
ObjA
Dead
ObjB
ObjC
Tablet
Tablet

Solution #2



UCLA Samueli School of Engineering

Solution #2



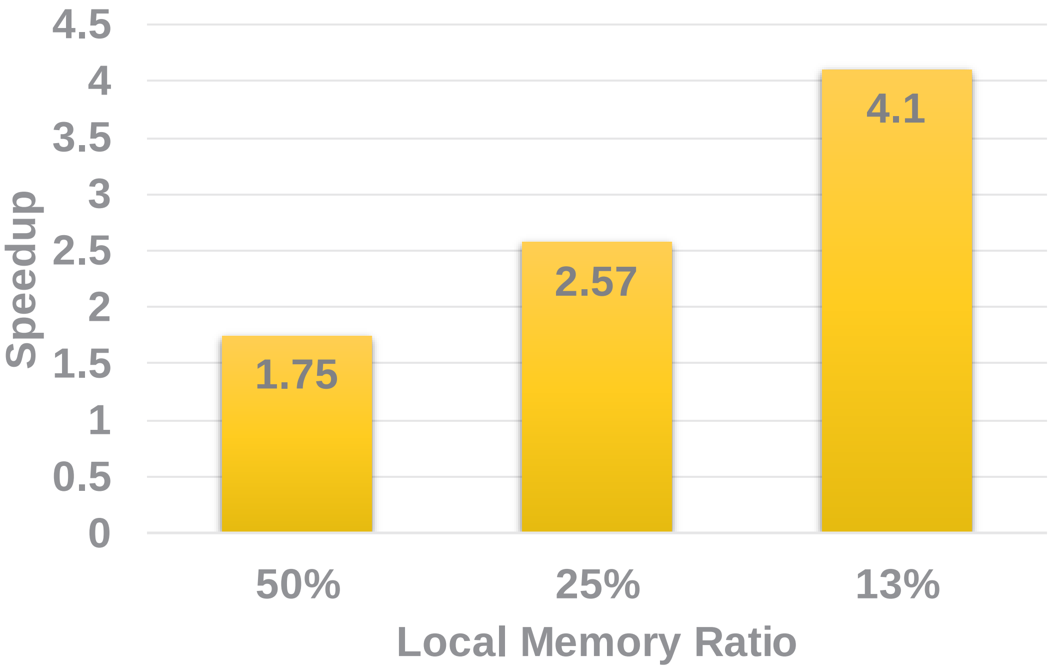
CPU Server
Memory Server
ObjB
EntryB
Region 0
Region 1
Region 2
Region 3
ObjA
Dead
ObjB
ObjC
ObjA
ObjC
Tablet
Tablet

Benchmarks

- We evaluate Mako on 7 workloads under three different local memory ratios: 50%, 25%, and 13%
  - Dacapo: Tradesoap, Tradebeans, H2
  - Apache Cassandra: Insert Intensive, Update Intensive
  - Apache Spark: PageRank, Transitive Closure

- We compare Mako with
  - Shenandoah: a modern concurrent collector in OpenJDK
  - Semeru: a G1-based generational GC for disaggregated memory

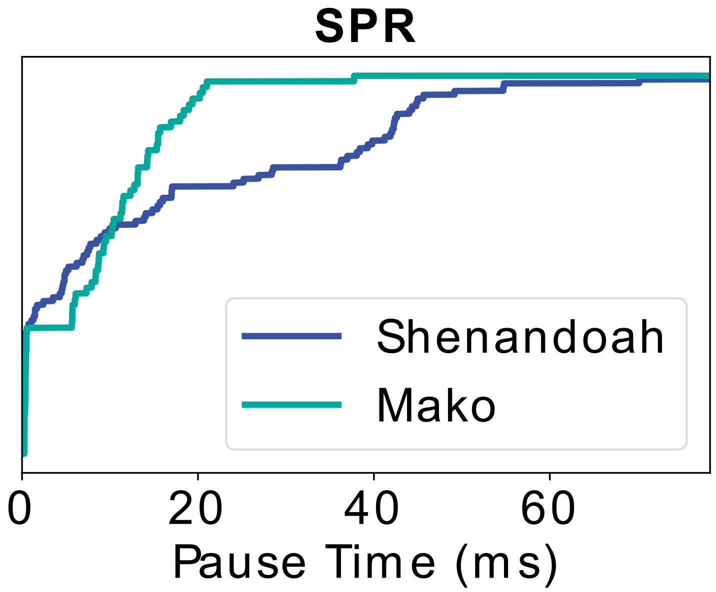
Results: Throughput



UCLA Samueli School of Engineering

Results: Pause Time

- Mako achieves ~12ms at the 90th-percentile pause time
- Semeru’s pauses are 2 to 3 orders of magnitude longer



Key Takeaways

- Under new hardware and system settings, it might be a good idea to bring back some old concepts and techniques. E.g. Heap Indirection Table

- Offloading GC to memory servers makes compute near data, which improves applications’ throughput

UCLA Samueli School of Engineering

Q&A