UCLA Samueli School of Engineering

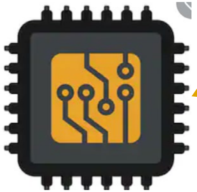
Semeru

A Memory-Disaggregated Managed Runtime

Chenxi Wang, Haoran Ma, Shi Liu, Yuanqi Li,	UCLA
Zhenyuan Ruan,	MIT
Khanh Nguyen,	Texas A&M University
Michael D. Bond,	Ohio State University
Ravi Netravali, Miryung Kim and Harry Xu	UCLA

Disaggregated Datacenter

CPU Servers
small memory

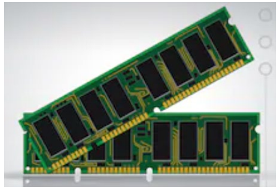


Network
(e.g., InfiniBand)

Storage Servers
weak cores

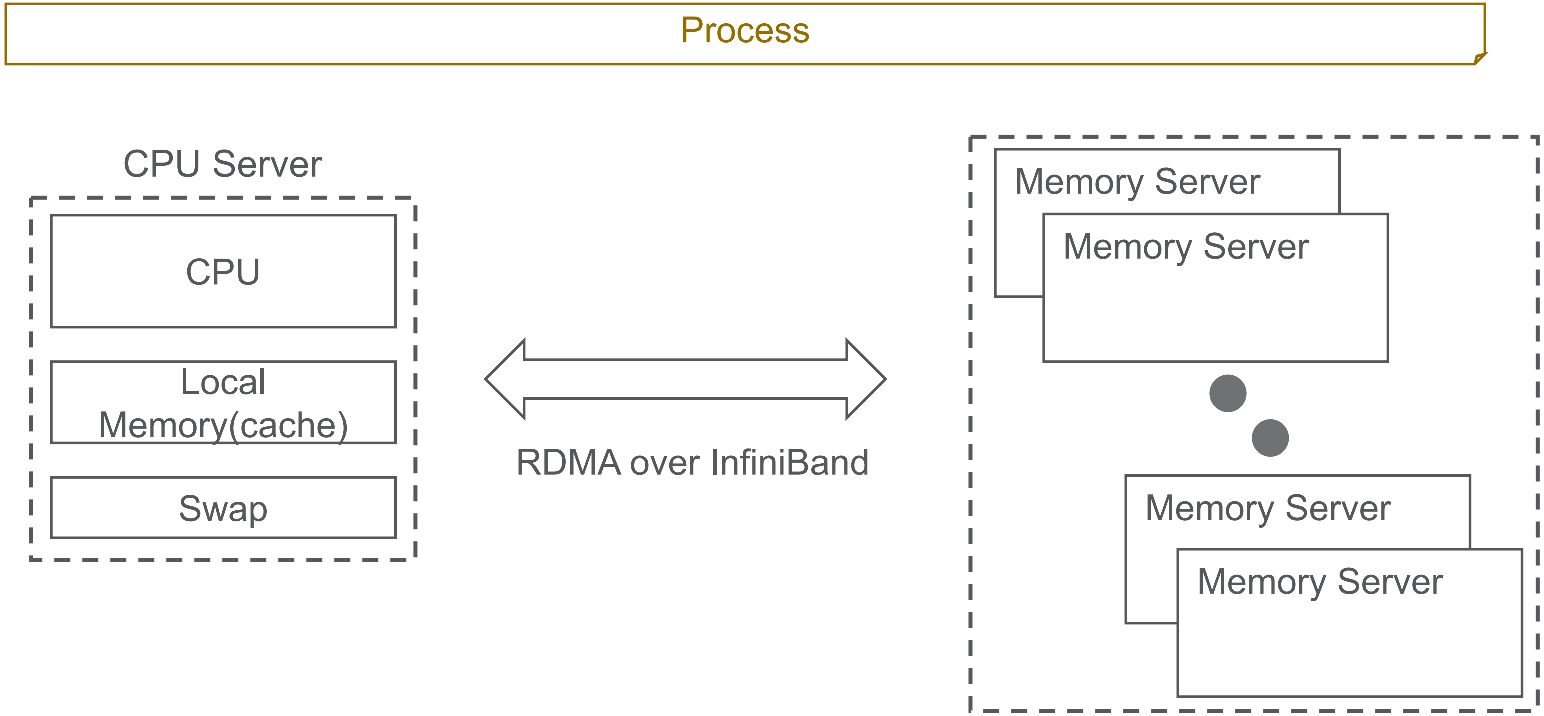
- Reliable network
- Micro-second latency
- Tens of GBs bandwidth

Memory Servers
weak cores

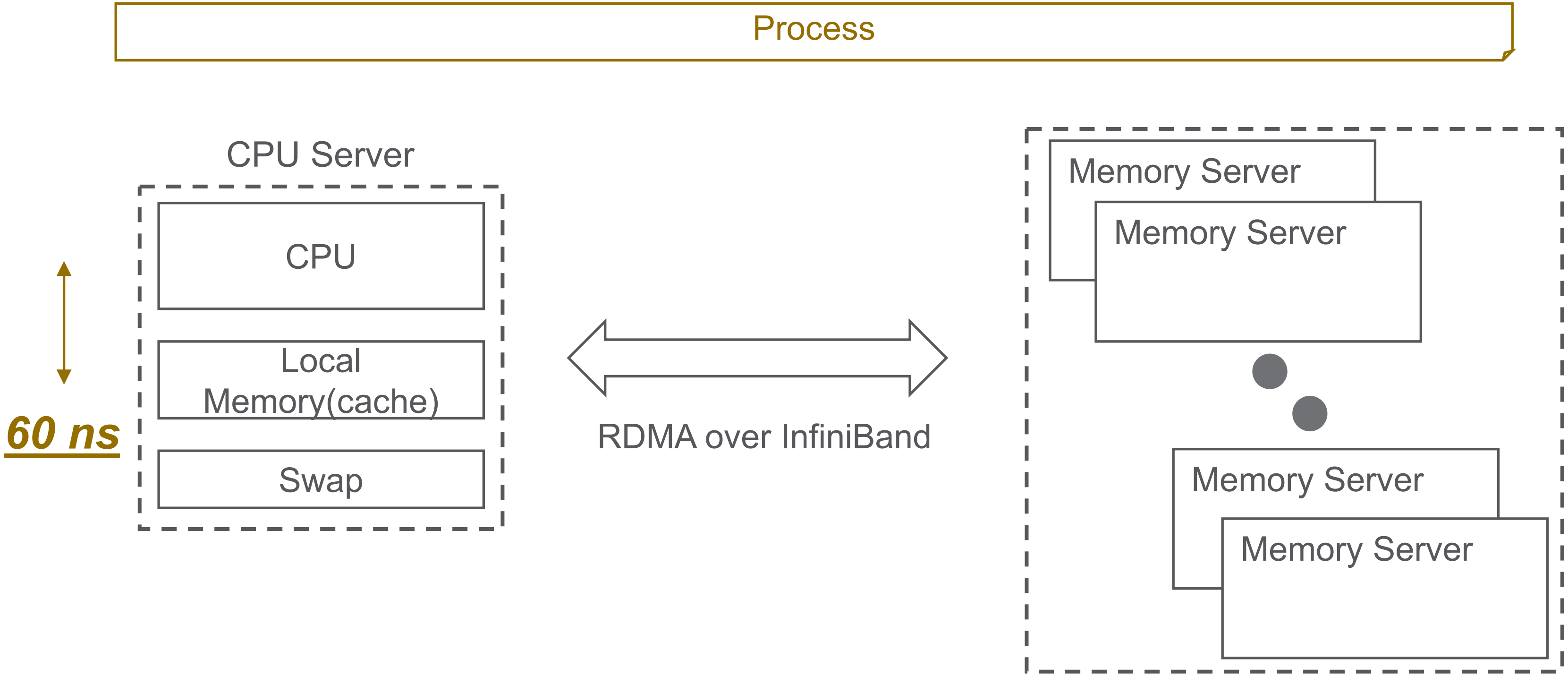


UCLA Samueli School of Engineering

Process Execution Model

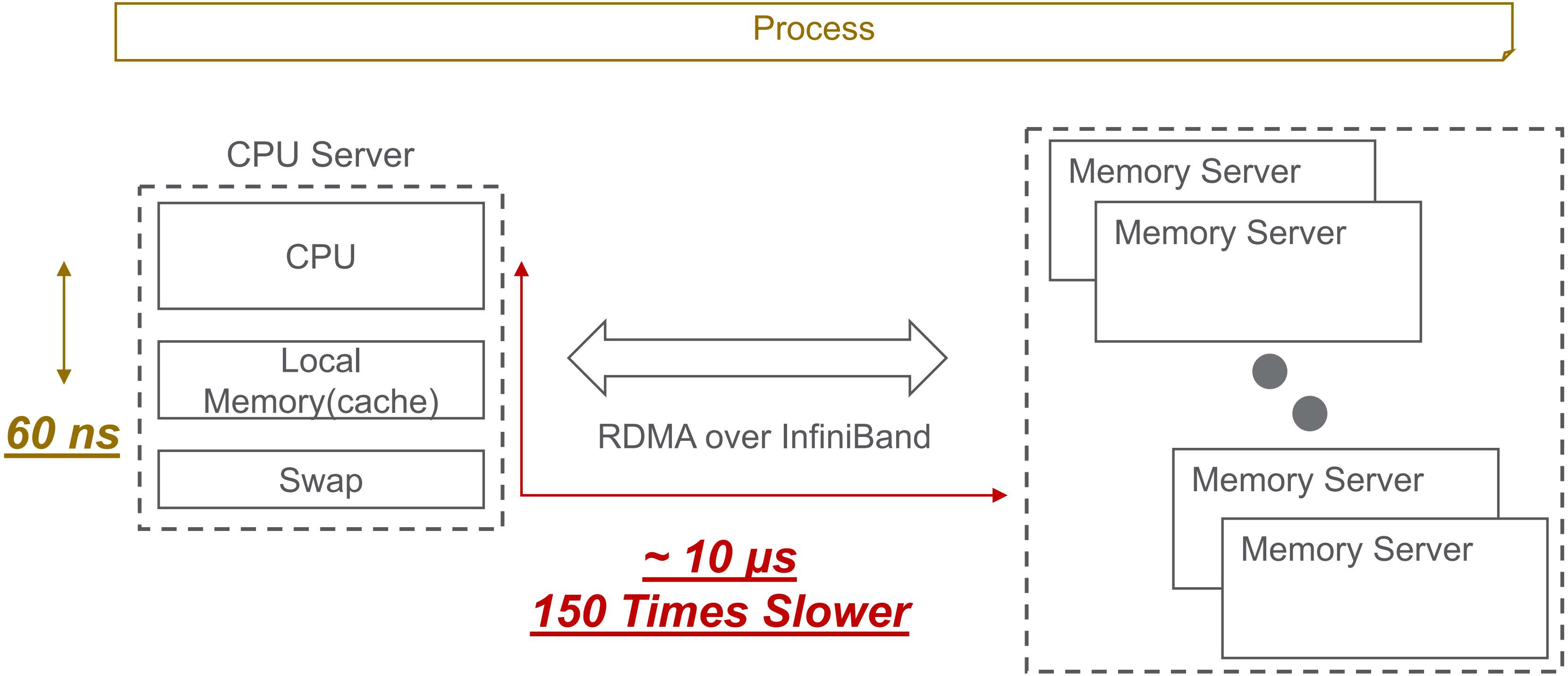


Process Execution Model



UCLA Samueli School of Engineering

Process Execution Model



UCLA Samueli School of Engineering

Limitations of Previous Work

- Previous works focus on semantics-agnostic optimizations
  - Reduce or hide the remote access latency
  - Prefetch data to reduce the remote access frequency

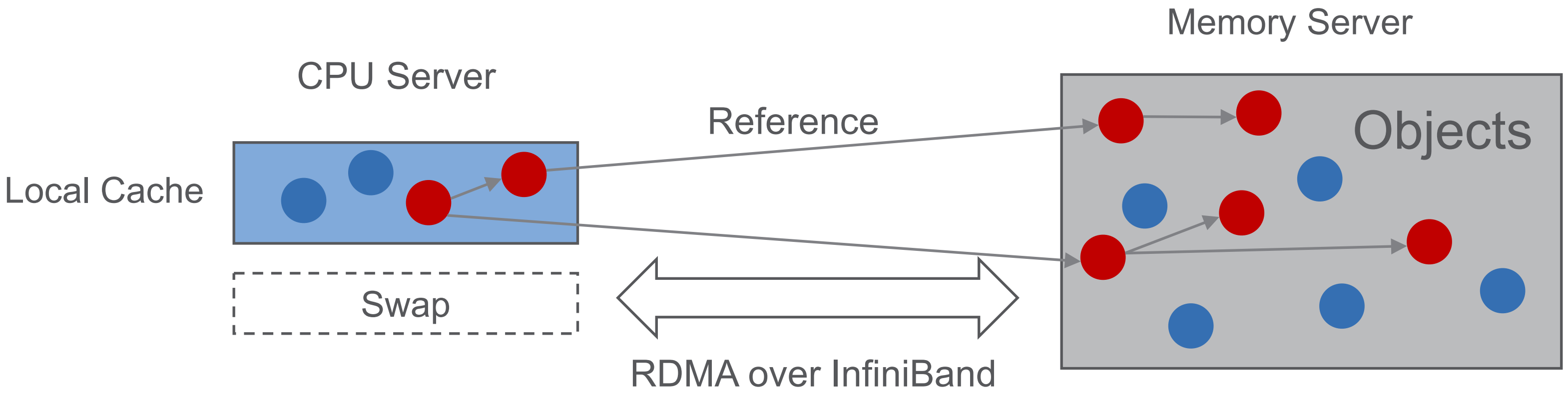
- Cloud applications – written in managed languages
  - Heap space: Reserved virtual space from OS
  - Garbage Collection (GC): Automatic memory management
  - Object-oriented data structures

Managed language applications often have poorer locality than native programs

Poor Data Locality

Object-oriented data structures

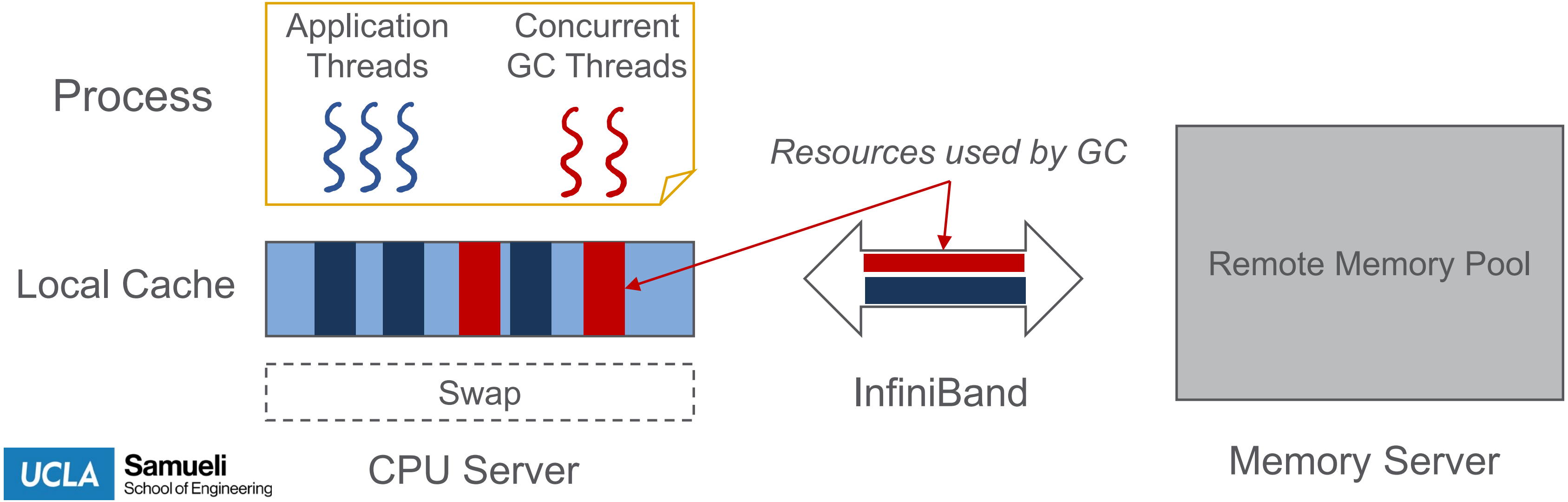
- ➢Random memory access – poor locality, hard to predict access pattern
- ➢Pointer-chasing memory access – latency sensitive



Resources Racing

GC slows down the applications

- ➢ The concurrent GC threads race resources, e.g., local cache and InfiniBand bandwidth, with the application threads



Slowdown of Spark Applications

Cache Ratio	Apps	GC	Total Time
No Swap	1.0	1.0	1.0
50%	2.0X	24.7X	*8.4X*
25%	5.3X	53.5X	*18.9X*

Spark GraphX TriangleCounting

Cache Ratio	Apps	GC	Total Time
No Swap	1.0	1.0	1.0
50%	1.2X	2.0X	*1.4X*
25%	2.0X	3.3X	*2.3X*

Spark MLlib KMeans

- Both applications and GC slow down significantly on a disaggregated cluster
- GC is on the critical path
  - GC increases the pause time
  - GC slows down the application's execution

UCLA Samueli School of Engineering

Major Insights

- Offload part of GC to memory servers where the data is located
  - Good fit for weak compute on memory servers
  - Near memory computing for high throughput
  - GC can run *concurrently* and *continuously*
- Utilize GC to adjust the data layout for applications

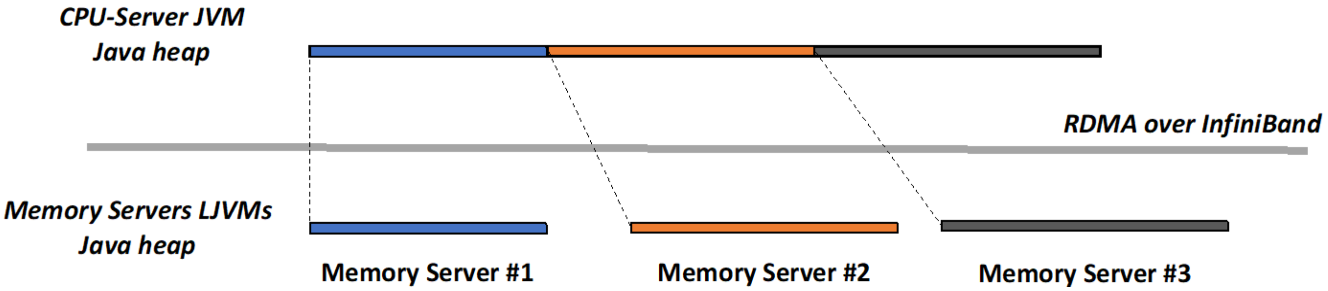
Semeru – A Disaggregated Managed Runtime

Challenges

- ➢ #1 What memory abstraction to provide ?
  - Universal Java Heap (UJH)
- ➢ #2 What to offload ?
- ➢ #3 How to efficiently swap data ?

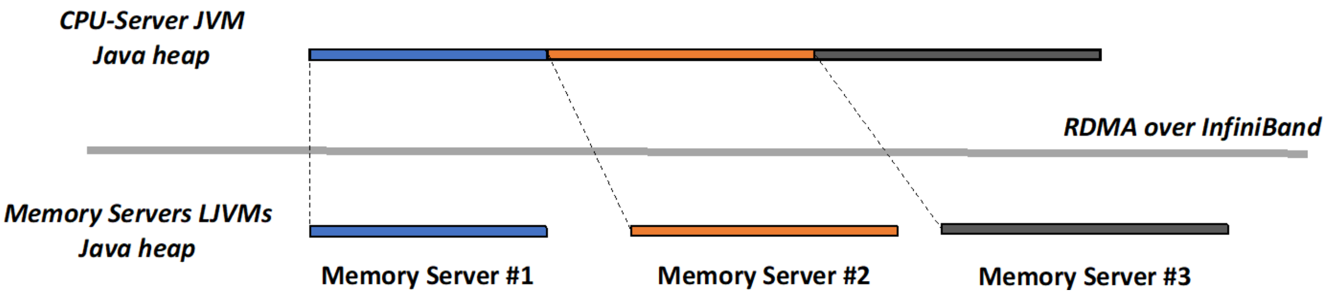
Universal Java Heap (UJH)

- A normal JVM runs on the CPU server, accessing the whole Java heap



Universal Java Heap (UJH)

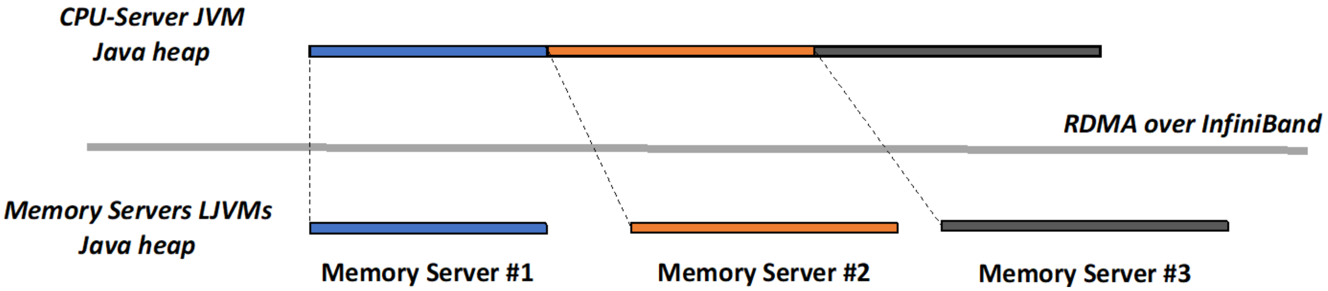
- A normal JVM runs on the CPU server, accessing the whole Java heap



- A Lightweight-JVM (LJVM) runs on each memory server, accessing its assigned Java heap range

Universal Java Heap (UJH)

➢ A normal JVM runs on the CPU server, accessing the whole Java heap



➢ A Lightweight-JVM (LJVM) runs on each memory server, accessing its assigned Java heap range

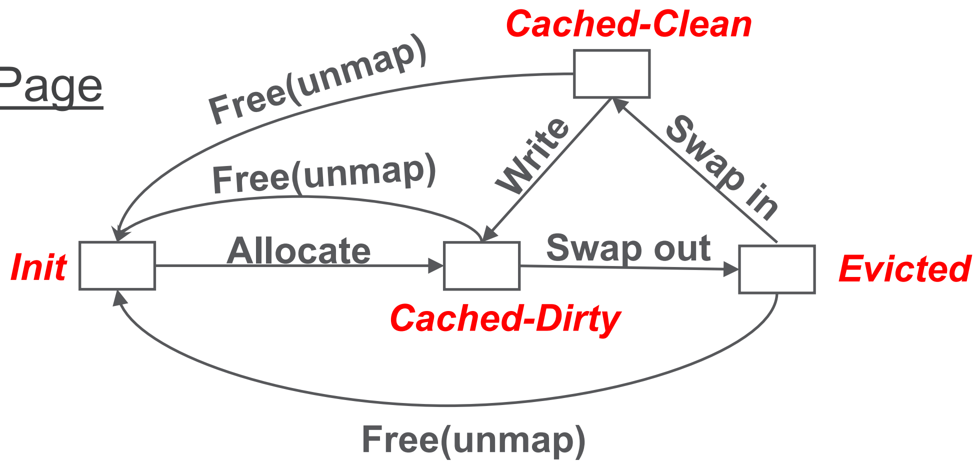
➢ Each object has the *same virtual address* on both the CPU server and memory servers

UCLA Samueli School of Engineering

CPU Server Cache Management

- Write-back policy
  - Objects are allocated in CPU server memory(local cache)
  - Only *dirty* pages are evicted to memory servers
  - When a page is freed by GC, it returns to the *Init* state

State Machine of Virtual Page



UCLA Samueli School of Engineering

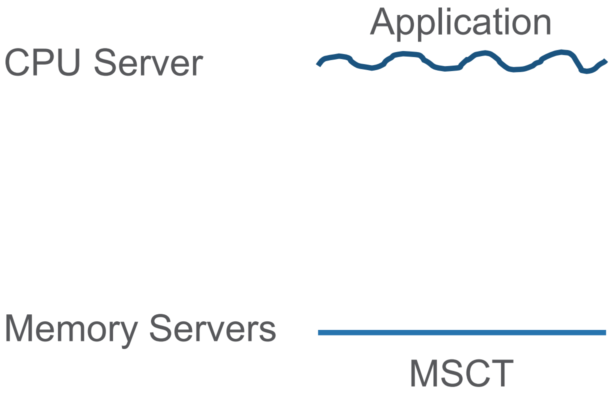
Challenges

- Universal Java Heap (UJH)
- #2 What to offload ?
  - Memory Server Concurrent Tracing (MSCT)
- #3 How to efficiently swap data ?

UCLA Samueli School of Engineering

Disaggregated GC Overview

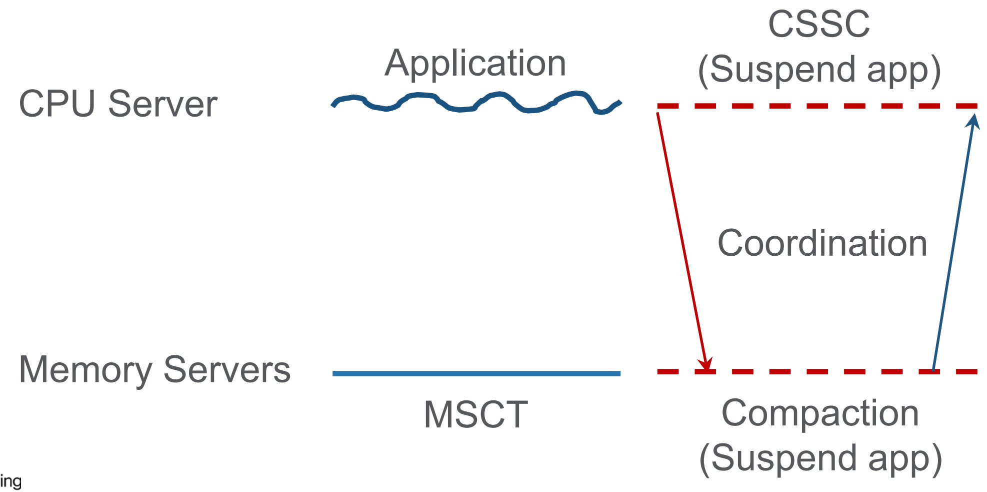
- Offload *tracing* to memory servers
  - Memory Server Concurrent Tracing (MSCT)



Disaggregated GC Overview

- Offload *tracing* to memory servers
  - Memory Server Concurrent Tracing (MSCT)

- Keep a GC phase on CPU server for *memory reclamation*
  - CPU Server Stop-the-world Collector (CSSC)

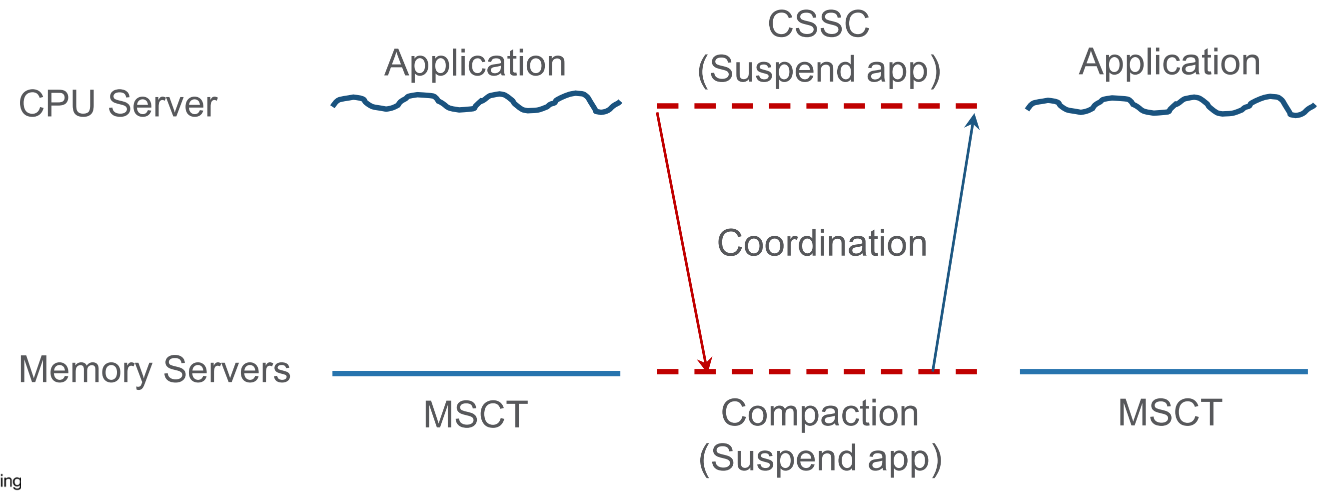


UCLA Samueli School of Engineering

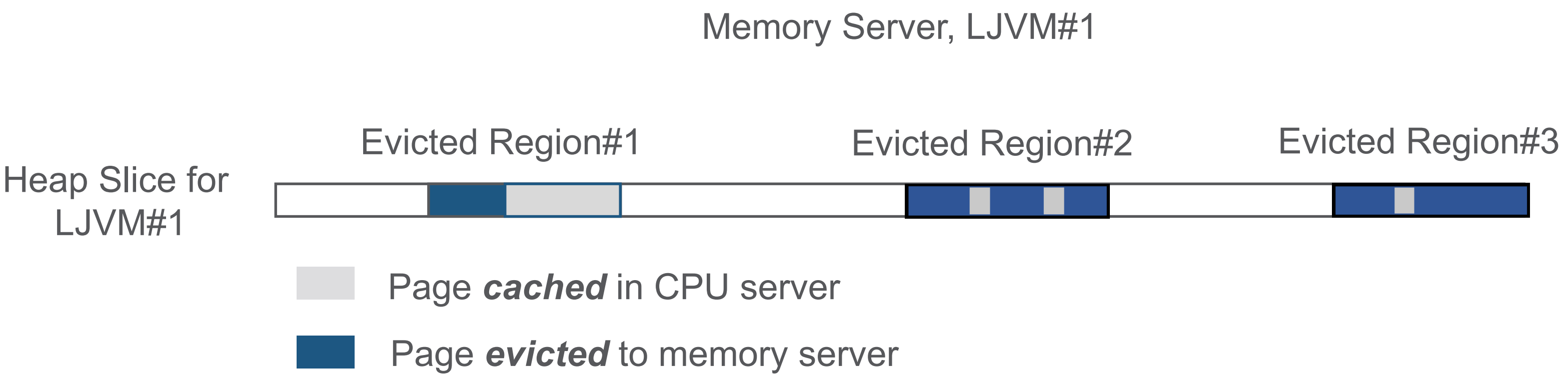
Disaggregated GC Overview

- Offload *tracing* to memory servers
  - Memory Server Concurrent Tracing (MSCT)

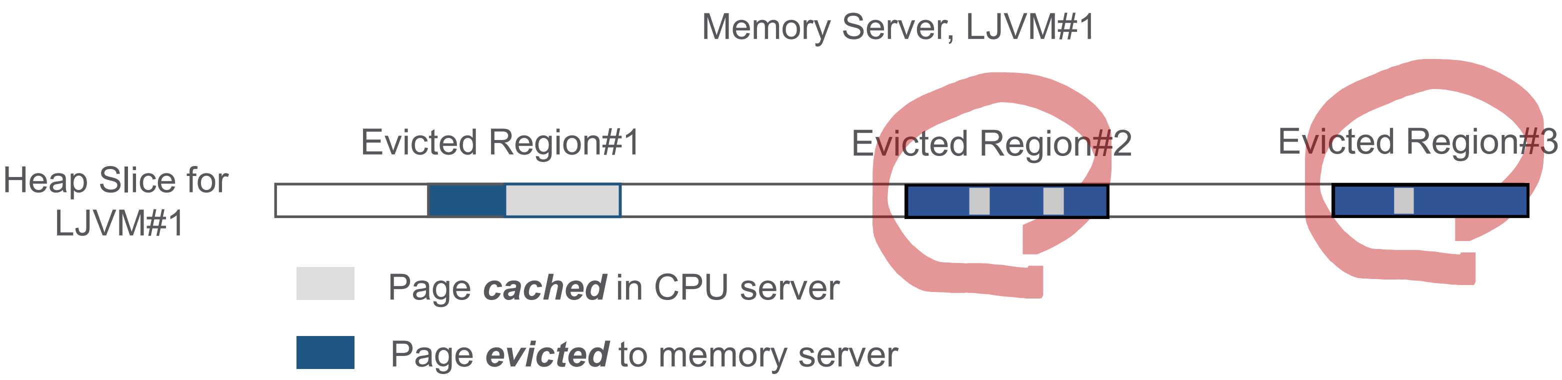
- Keep a GC phase on CPU server for *memory reclamation*
  - CPU Server Stop-the-world Collector (CSSC)



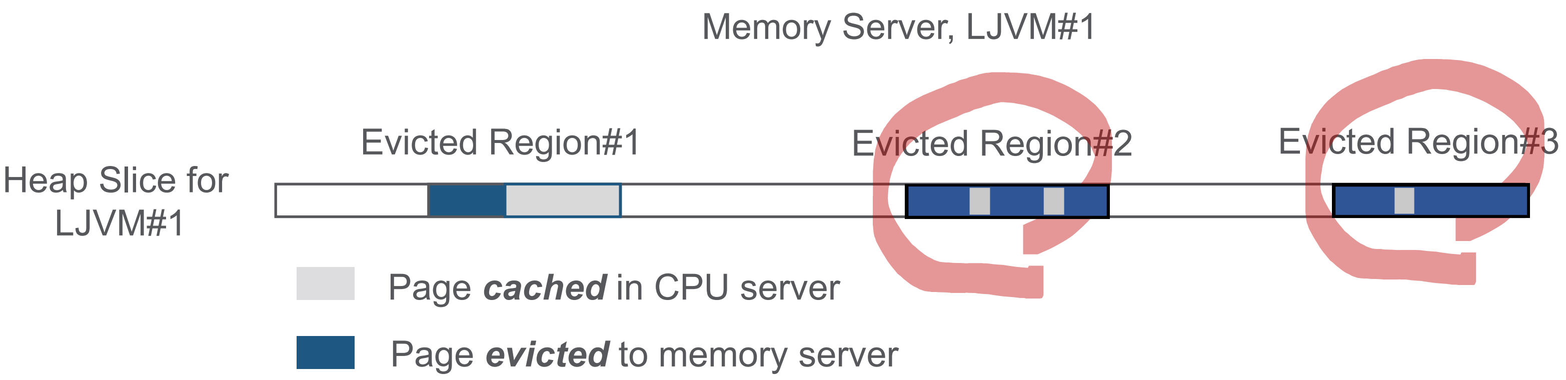
MSCT – Regions to be Traced



MSCT – Regions to be Traced



MSCT – Regions to be Traced



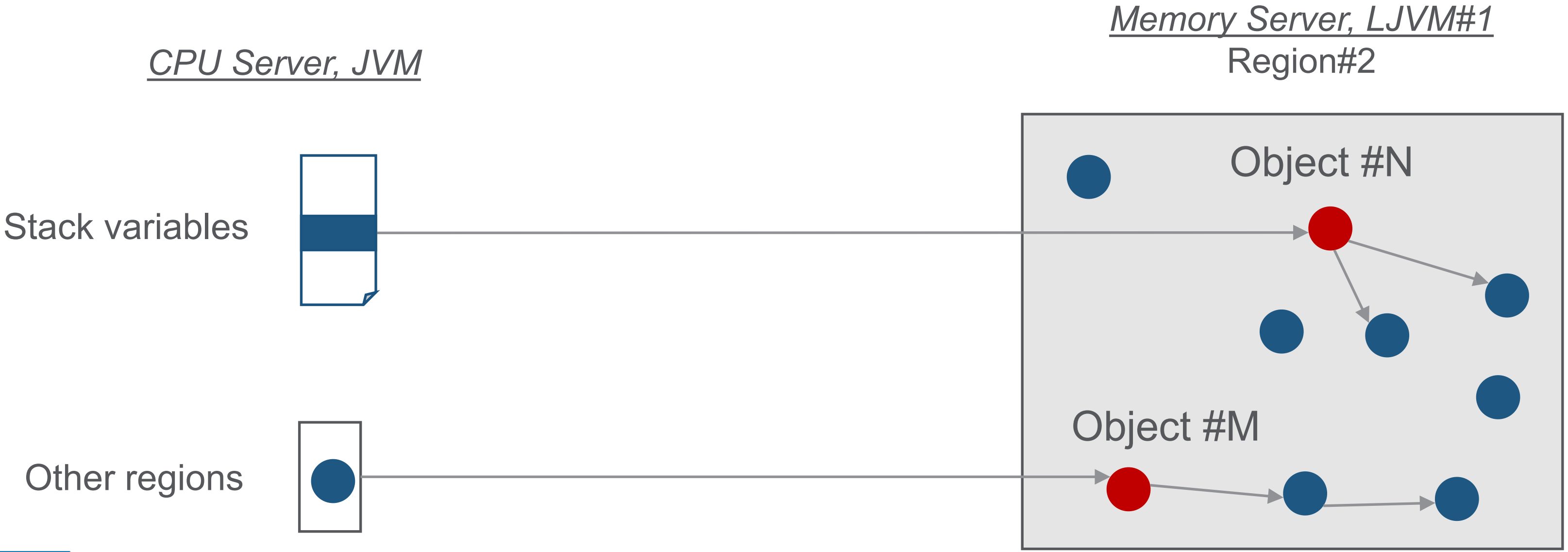
Tracing Order : Region#2 *(age 1)* → Region#3 *(age 2)*

Generation Hypothesis:
Newly allocated objects are more likely to die

MSCT – Tracing Roots

Tracing roots for each region

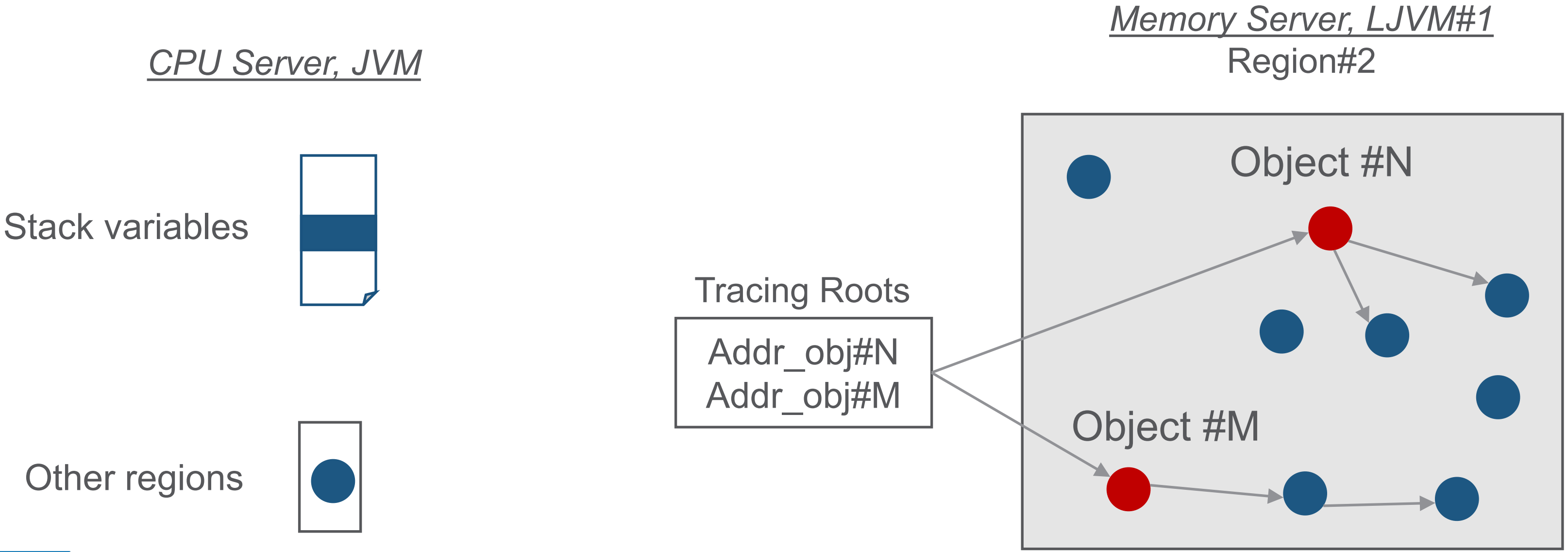
- References from stack variables
- References from other regions



MSCT – Tracing Roots

Tracing roots for each region

- References from stack variables
- References from other regions



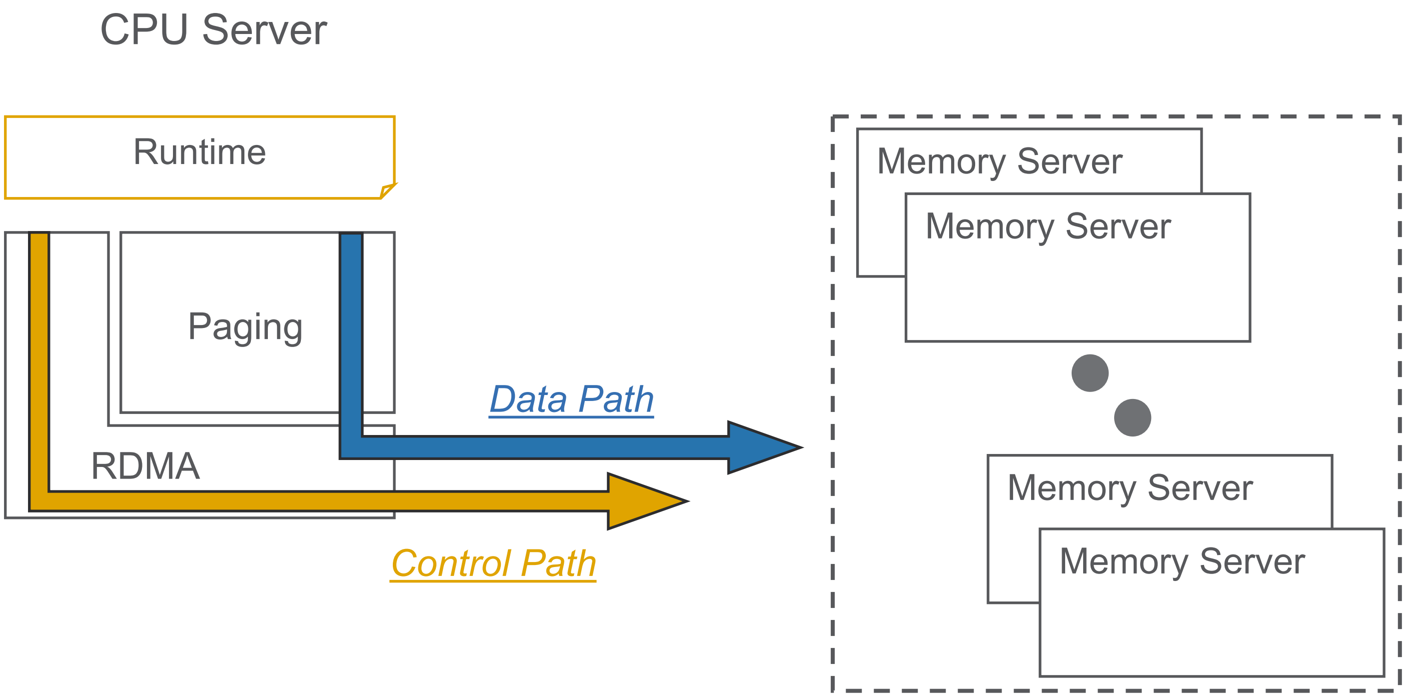
CPU Server Stop-The-World Collection (CSSC)

- CPU server GC is the main collection phase
  - Trace the cached regions on the CPU server
  - Coordinate CPU server and memory servers for space compaction
  - Adjust the data layout for applications

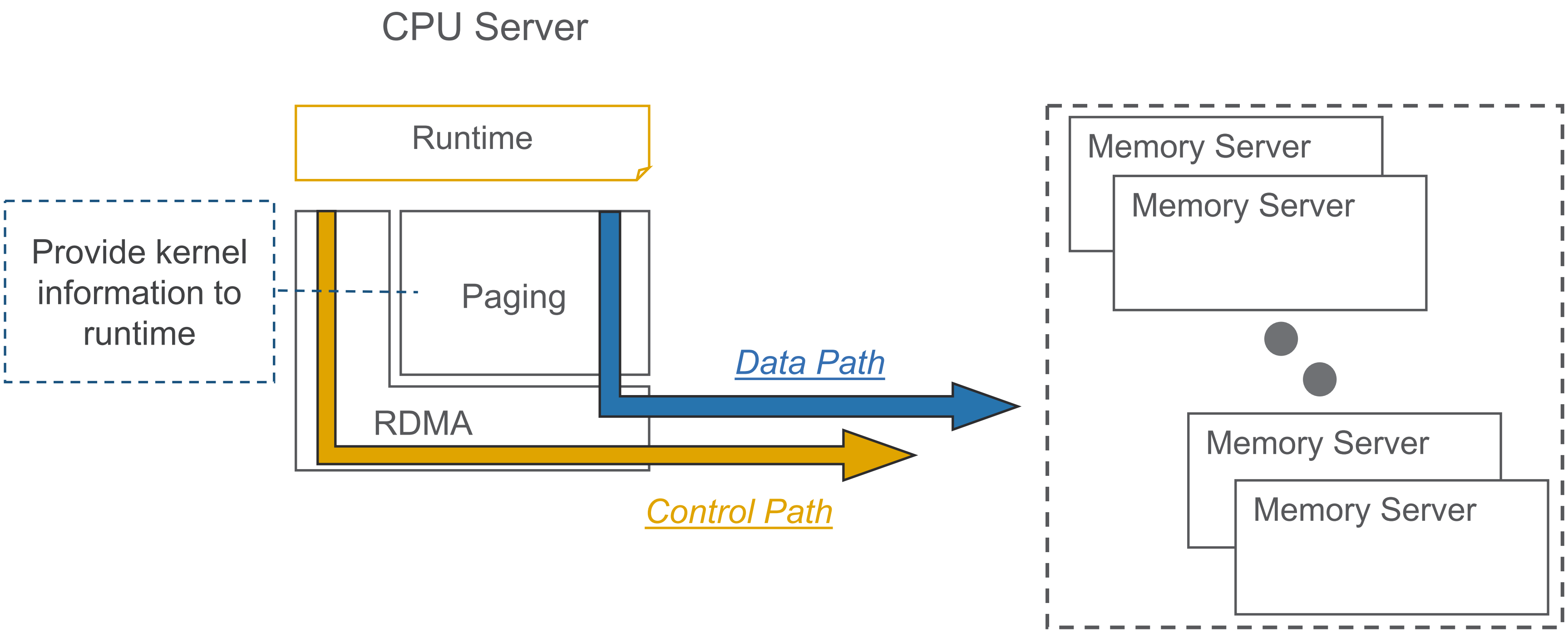
Semeru Design Outline

- Universal Java Heap (UJH)
- Disaggregated GC
  - Memory Server Concurrent Tracing (MSCT)
  - CPU Server Stop-The-World Collection (CSSC)
- #3 How to design the swap system ?

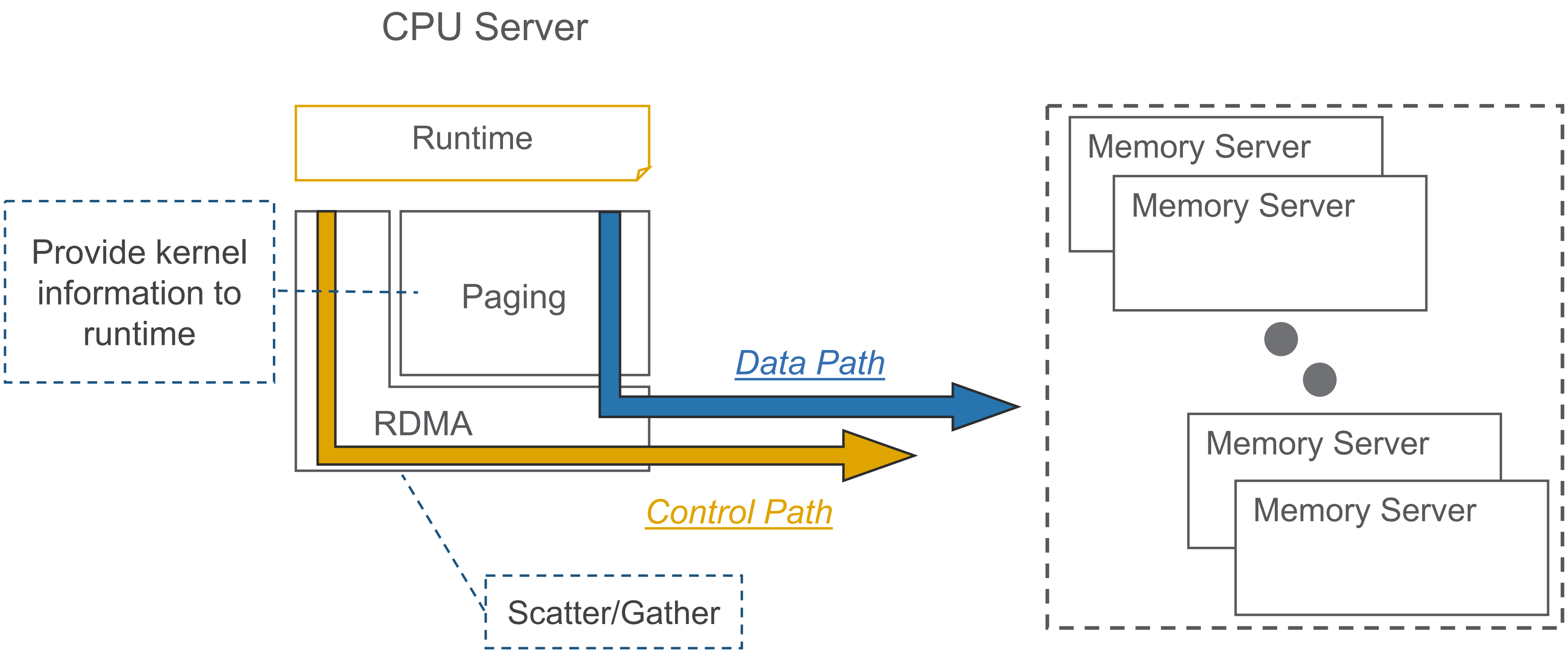
Swap System Overview



Swap System Overview



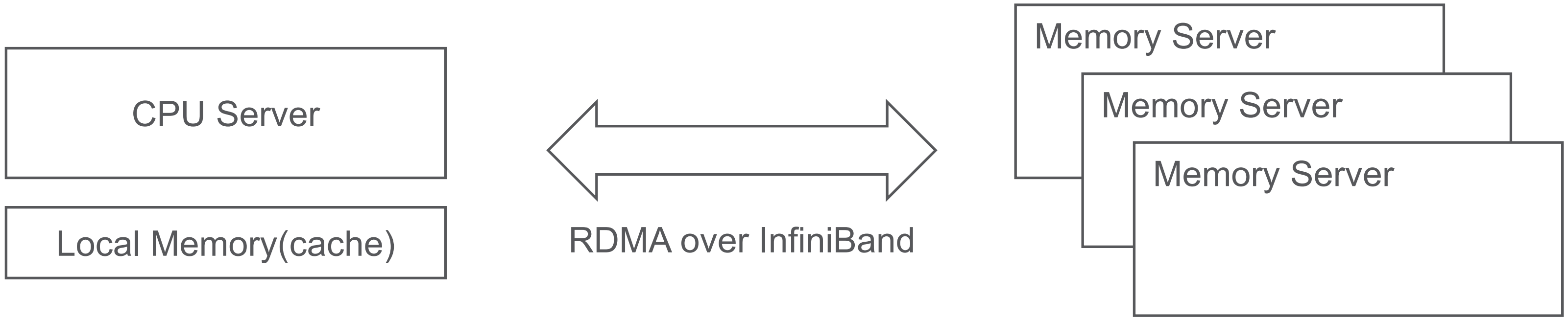
Swap System Overview



Experiment Setup

- 2 CPUs per server
  - Intel Xeon E5-2640 v3 @2.60GHz, 8 cores
- InfiniBand
  - ConnectX®-3 , MT4099, 40Gb/s
- CPU Local Memory
  - DDR4-1866, Limit capacity by CGroup

- 3 memory servers per application
- 2 cores per server
  - Intel Xeon E5-2640 v3
    - Limit number of cores
    - Fix CPU freq to 1.2GHz / 2.6GHz



Overall Performance

- Workloads
  - 5 Spark applications
  - 3 Flink applications
- Datasets
  - Wikipedia
  - KDD
- Configurations
  - Baseline: No swap
  - NVMe-oF
  - RAMDisk

50% Cache	Apps	GC	Total Time
G1-NVMe-oF	2.00X	4.44X	*2.24X*
G1-RAMDisk	1.82X	2.79X	*1.87X*
Semeru	1.06X	1.42X	*1.08X*

25% Cache	Apps	GC	Total Time
G1-NVMe-oF	3.85X	14.13X	*4.58X*
G1-RAMDisk	3.16X	4.59X	*3.23X*
Semeru	1.22X	2.67X	*1.32X*

Memory-Server Tracing Performance

- GC Improvement

Configuration	Tracing Performance	
	Throughput (MB/s)	Core Utilization
(Memory Server) Single core, 1.2 GHz	*418.3*	*29.0%*
(Memory Server) Single core, 2.6 GHz	*922.2*	*12.4%*
(CPU Server) Single core, 2.6 GHz	*93.9*	*N/A*

- *Offload tracing to memory servers increases throughput 8.8X*
- *Weak core is powerful enough to do continuous tracing on memory servers*

Conclusions

- Semeru achieves superior efficiency on the disaggregated cluster via
  - A co-design of the runtime and swap system
  - Careful coordination of different GC tasks

- Disaggregation performance could benefit much more from a redesigned runtime than semantics-agnostic optimizations

Q&A

Thanks

wangchenxi@cs.ucla.edu