TOSS: Tiering of Serverless Snapshots for Memory-Efficient Serverless Computing

Theodore Michailidis, Juno Kim, Linsong Guo, Steven Swanson, Jishen Zhao

University of California, San Diego

UCSDCSE
Computer Science and Engineering

Serverless computing

- Serverless computing is a **billion-dollar** industry (exp. $200 billion by 2035)

Azure Functions

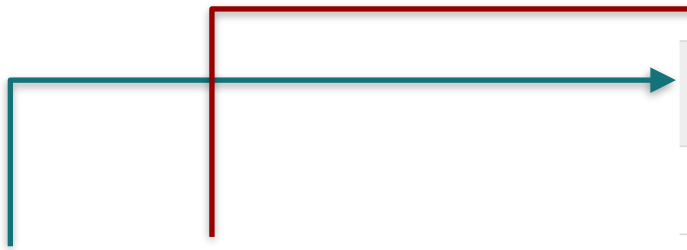
Serverless computing

- ✓Write code
- ✓Choose Virtual Machine (VM) configuration
  - Pay what you use
- ✓Deploy

- ✓VM Lifecycle
- ✓Infrastructure
- ✓Security
- ✓Storage

- ✓Scalability
- ✓Pricing

Serverless Pricing



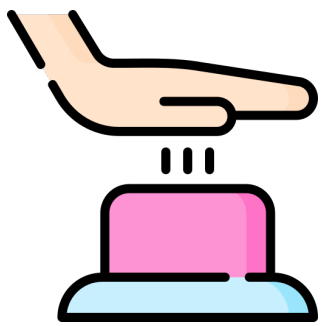
- Choose **memory** & **CPU** configuration
- Have to choose a **memory & CPU bundle**

Memory	vCPU[1]	Price/100ms (Tier 1 Price)	Price/100ms (Tier 2 Price)
128MB	.083 vCPU	$0.000000231	$0.000000324
256MB	.167 vCPU	$0.000000463	$0.000000648
512MB	.333 vCPU	$0.000000925	$0.000001295
1024MB	.583 vCPU	$0.000001650	$0.000002310
2048MB	1 vCPU	$0.000002900	$0.000004060
4096MB	2 vCPU	$0.000005800	$0.000008120
8192MB	2 vCPU	$0.000006800	$0.000009520
16384MB[2]	4 vCPU	$0.000013600	$0.000019040
32768MB[2]	8 vCPU	$0.000027200	$0.000038080

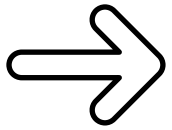
Issues:

1. **Choosing the right configuration might be hard!**
2. **Unused (expensive) memory**

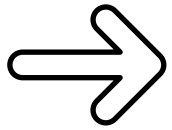
Serverless invocation process



Function invocation



Setup/Startup



Run function

Total invocation time

Serverless invocation process

Running time: **50%** of functions < 1s, **96%** of functions < 10s

Source: Serverless in the Wild, Shahrad et al., ATC'20

Serverless invocation process

Issue: Total invocation time is **dominated by Startup time** (*cold start*)

Function invocation

Startup

160-3,600ms

Run function

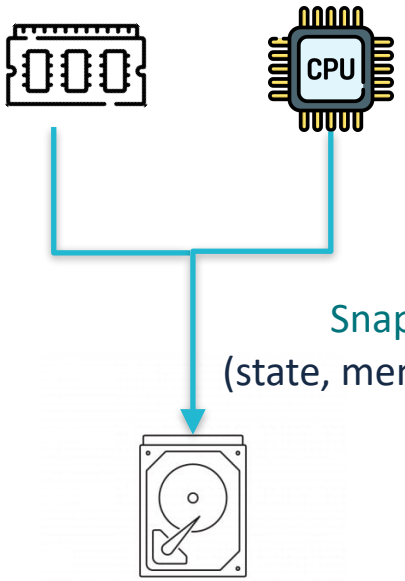
Source: Peeking Behind the Curtains of Serverless Platforms, Wang et al., ATC'18

Solutions for cold start

- Keep-alive
  - Functions remain in memory for predefined time
    - Standard (e.g. 30 minutes)
    - Determined by previous invocation patterns
- **Snapshots**
  - Fast, simple, low-cost solution
  - Lots of attraction both from industry and academia

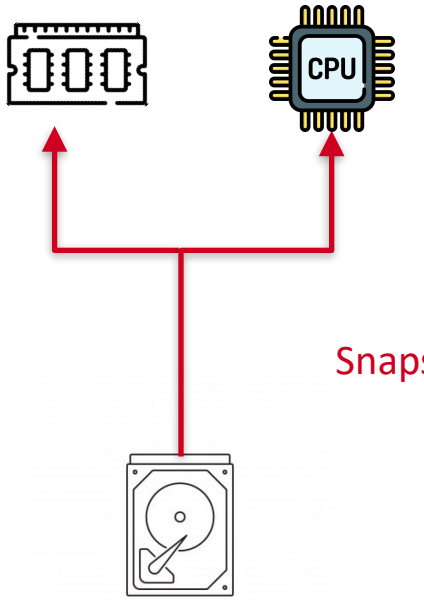
Snapshots

Initial function invocation



Snapshot creation
(state, memory contents etc)

Before subsequent function invocations

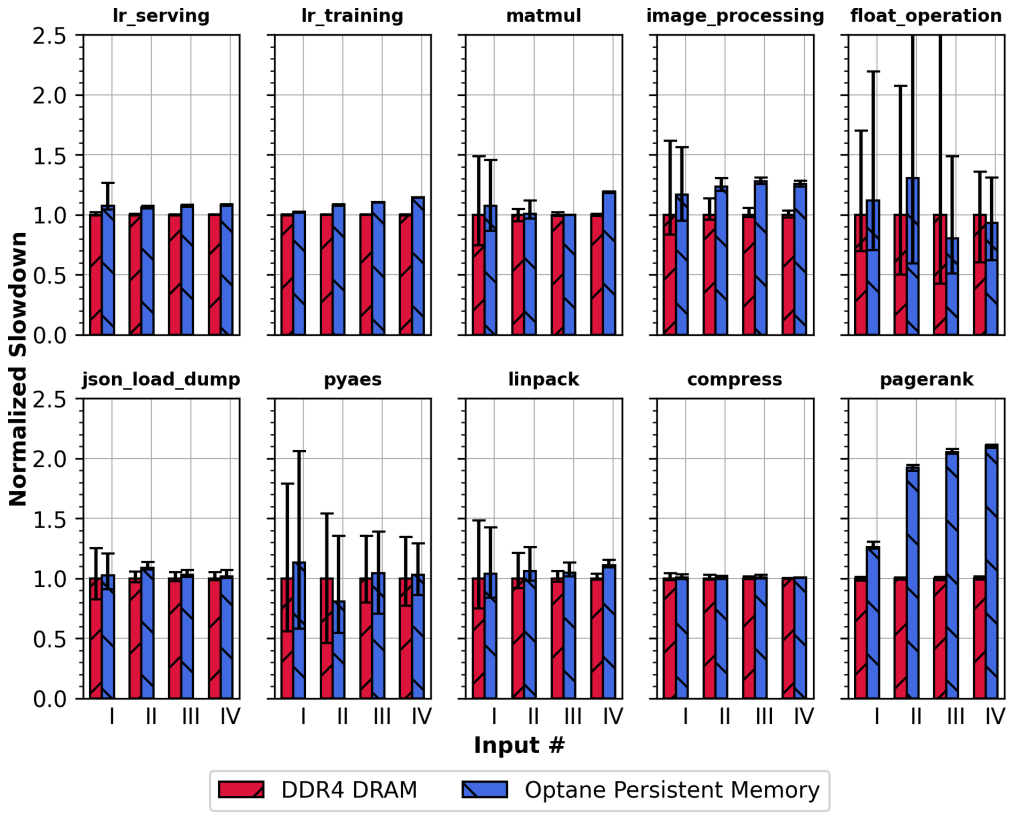


Snapshot Loading

- **Single pass snapshot - not accounting for all inputs**
- **Assumes a single tier of (costly) memory**

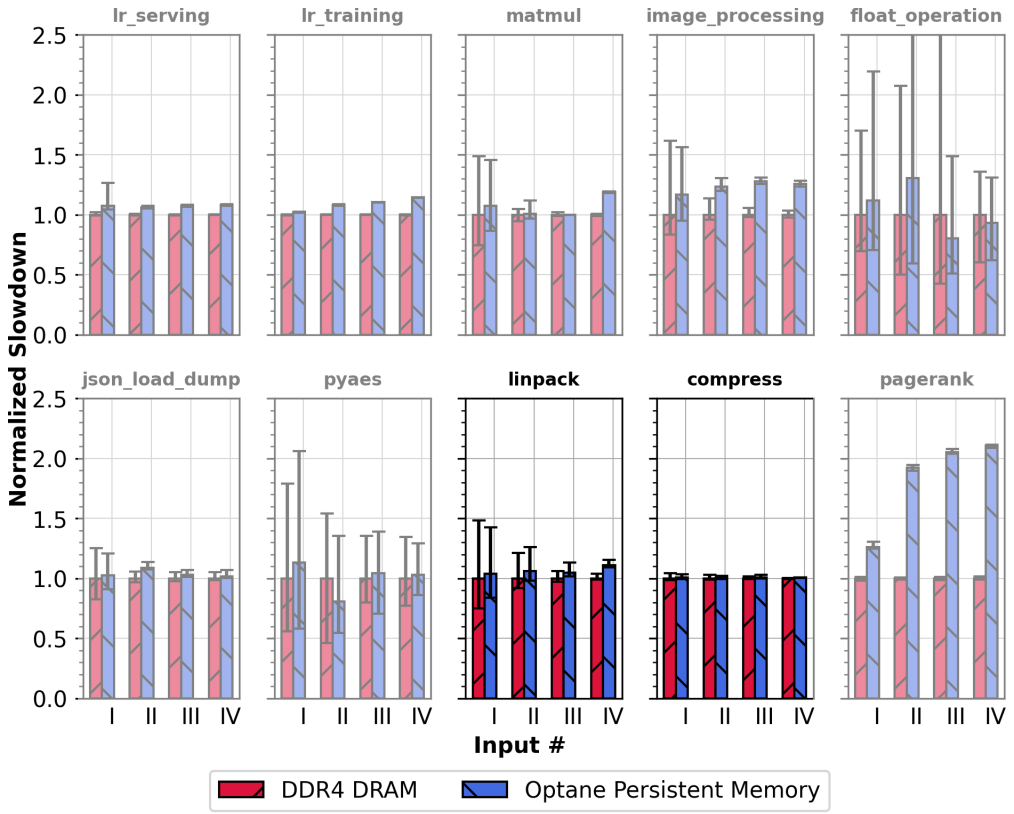
Function’s slowdown on Persistent Memory

- 10 functions, 4 inputs per function (run **entirely** on fast or slow tier)



Function’s slowdown on Persistent Memory

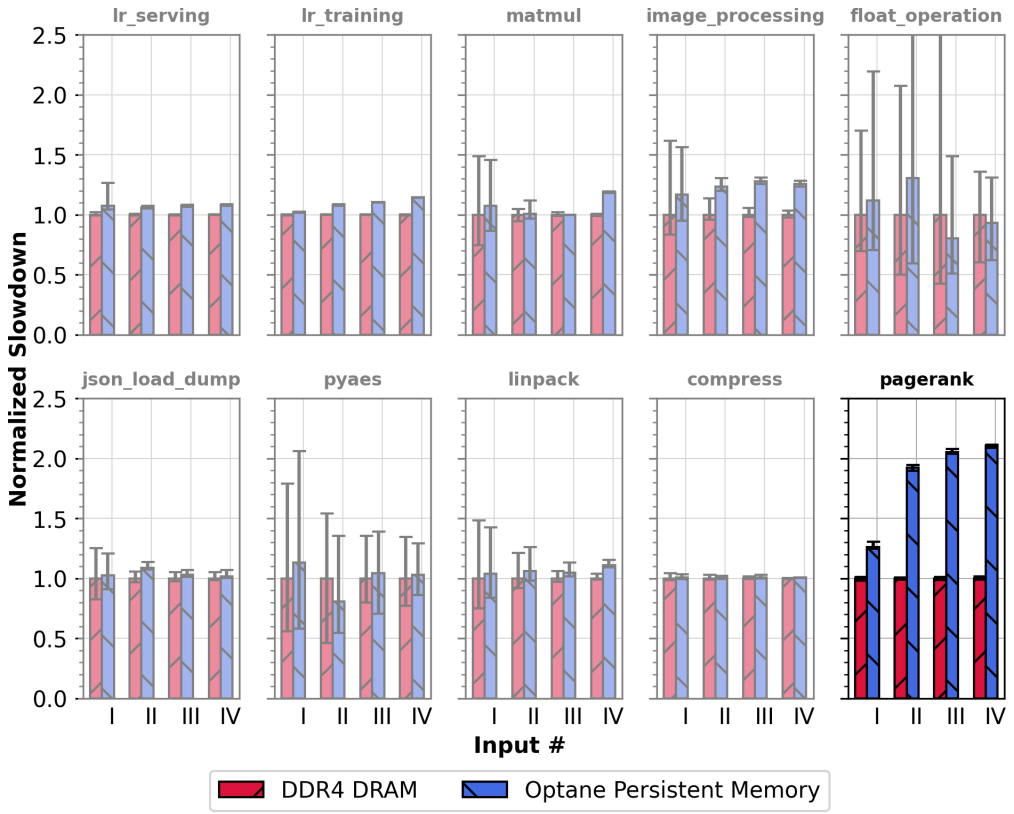
- 10 functions, 4 inputs per function (run **entirely** on fast or slow tier)



✓ Some functions display **minor or no slowdown** on slower memory

Function’s slowdown on Persistent Memory

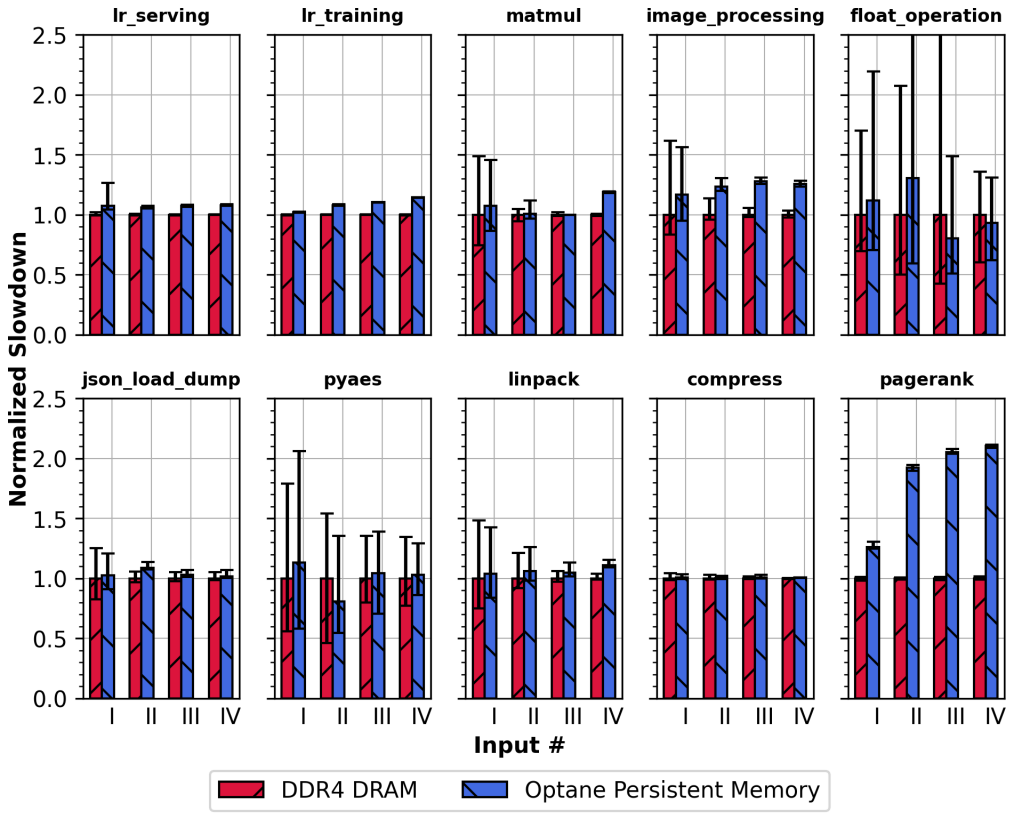
- 10 functions, 4 inputs per function (run **entirely** on fast or slow tier)



- ✓ Some functions display **minor or no slowdown** on slower memory
- ✓ Different inputs display **variable slowdown**
  - **Increased input size leads to increased memory footprint and slowdown**

Function's slowdown on Persistent Memory

- 10 functions, 4 inputs per function (run **entirely** on fast or slow tier)



- ✓ Some functions display **minor or no slowdown** on slower memory
- ✓ Different inputs display **variable slowdown**
  - **Increased input size leads to increased memory footprint and slowdown**

Need for something more sophisticated!

Pathologies on previous approaches

Single tier of memory

- **Expensive:** Memory accounts for 50% of total server cost
- **Redundant:** Usually only a subset of memory should go to the fast tier

Serveless bundle

- Hard to find optimal configuration
- Wasted memory

Single snapshot

- A single snapshot does not represent the diversity of function inputs

Tiering of Serverless Snapshots (TOSS)

TOSS

First Memory Tiering system for serverless

Goal: Reduce serverless memory cost operation

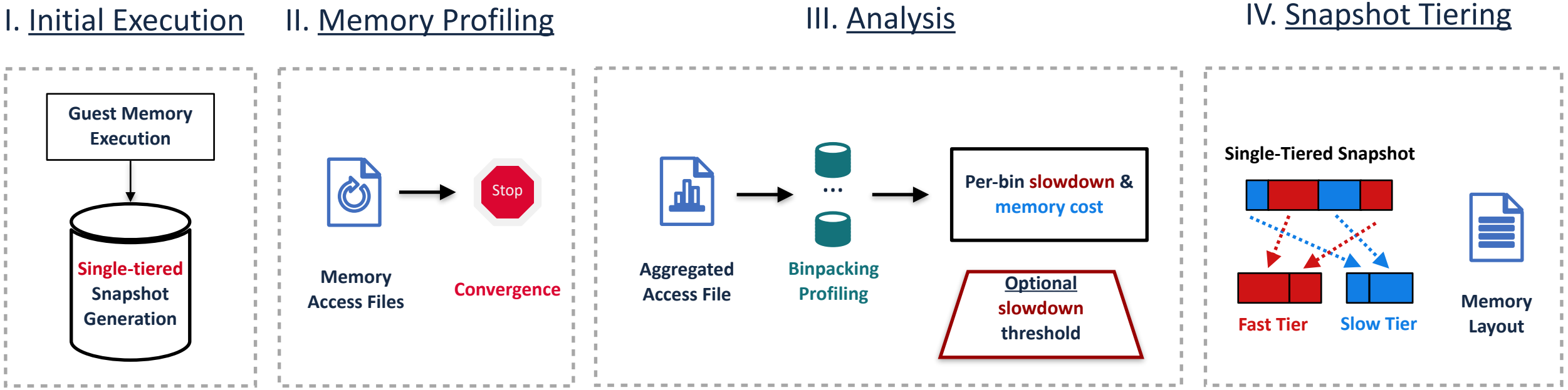
Insights about serverless memory

- Study serverless functions' memory patterns
- Correlation about slowdown and memory cost

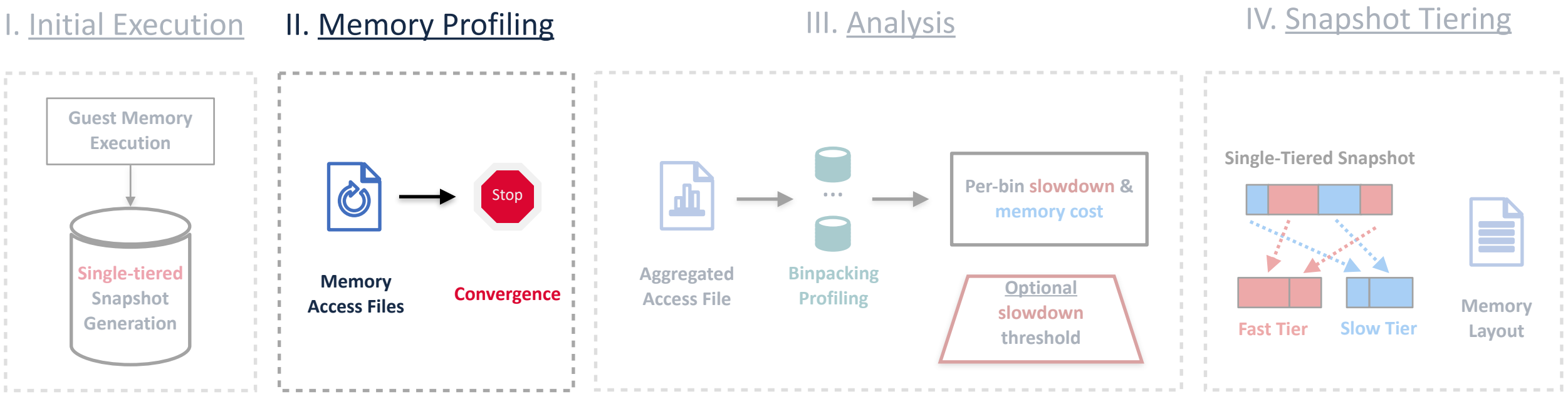
Flexible memory cost formula

- Cost per tier
- Memory per tier
- Slowdown

TOSS Overview



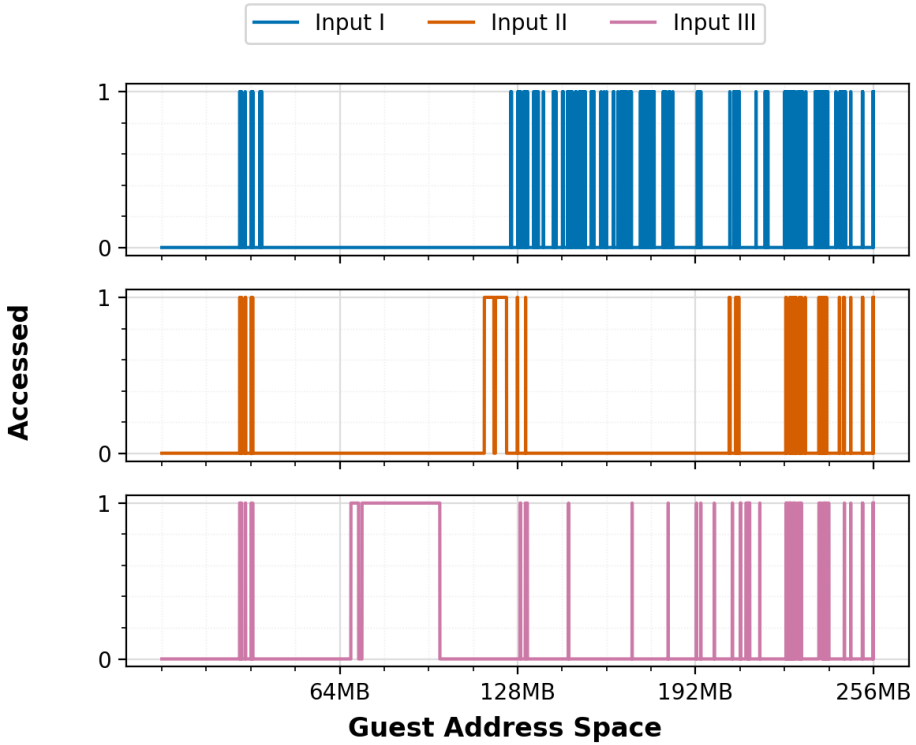
Memory profiling



- **Accurate** with **low overhead**
- Should reflect **different memory patterns** from **different inputs**

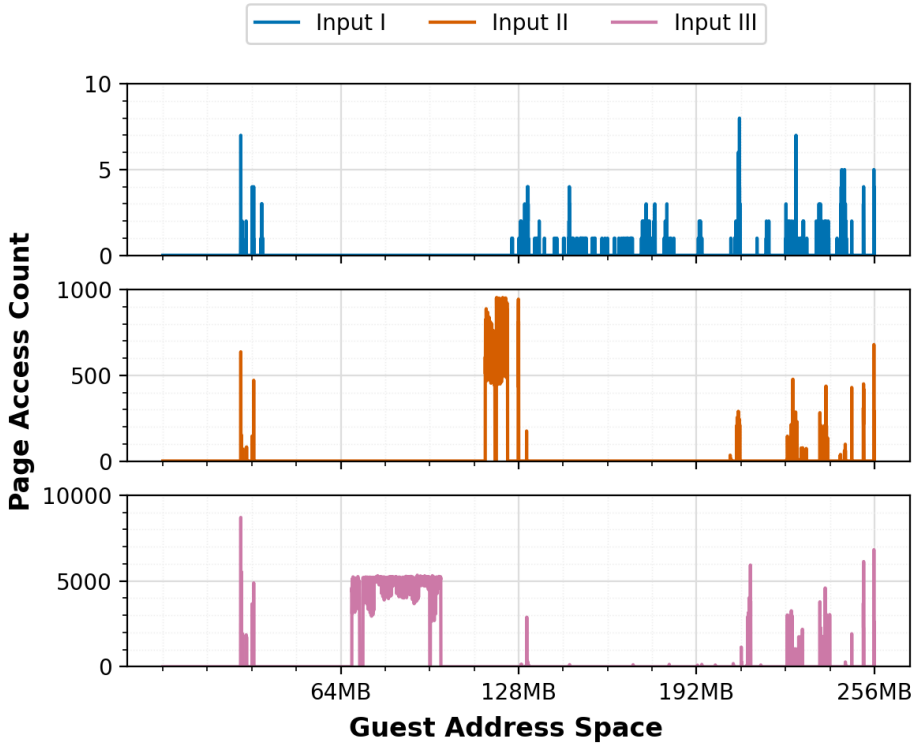
Serverless memory profiling

Previous approaches



- ✳ **Hot** (accessed) vs **Cold** (not-accessed)
- ✳ High overhead **(15% - 87%)**

TOSS (DAMON)



- ✓ Lightweight, accurate and scalable
- ✓ Fine-grained access memory pattern

Memory Cost Formula

$$Memory\ Cost = Slowdown * (MB_{Fast} * CostPerMB_{Fast} + MB_{Slow} * CostPerMB_{Slow})$$

Where: $\text{CostPerMB}_{\text{Fast}} = \text{CostPerMB}_{\text{Slow}} * 2.5$

Memory Cost Formula

Memory in MB that resides in the **fast** and **slow** tiers

$$Memory\ Cost = Slowdown * (MB_{Fast} * CostPerMB_{Fast} + MB_{Slow} * CostPerMB_{Slow})$$

Where: $\text{CostPerMB}_{\text{Fast}} = \text{CostPerMB}_{\text{Slow}} * 2.5$

Memory Cost Formula

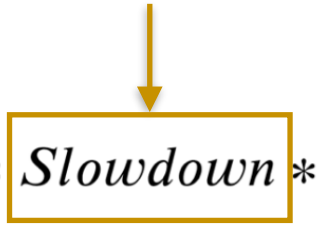
Cost ratio per MB between tiers

$$Memory\ Cost = Slowdown * (MB_{Fast} * CostPerMB_{Fast} + MB_{Slow} * CostPerMB_{Slow})$$

Where: $\text{CostPerMB}_{\text{Fast}} = \text{CostPerMB}_{\text{Slow}} * 2.5$

Memory Cost Formula

Slowdown for this memory configuration



$$Memory\ Cost = Slowdown * (MB_{Fast} * CostPerMB_{Fast} + MB_{Slow} * CostPerMB_{Slow})$$

Where: $\text{CostPerMB}_{\text{Fast}} = \text{CostPerMB}_{\text{Slow}} * 2.5$

Evaluation - Setup & Systems

CPU: 2×20-core Intel® Xeon® Gold 6230
Memory: **96GB** DDR4 DRAM (**fast**) & **192GB** Intel® Optane™ Persistent Memory (**slow**)
SSD: Intel® Optane™ DC (seq read and write of up to **2,500 MB/s** and **2,200 MB/s**)

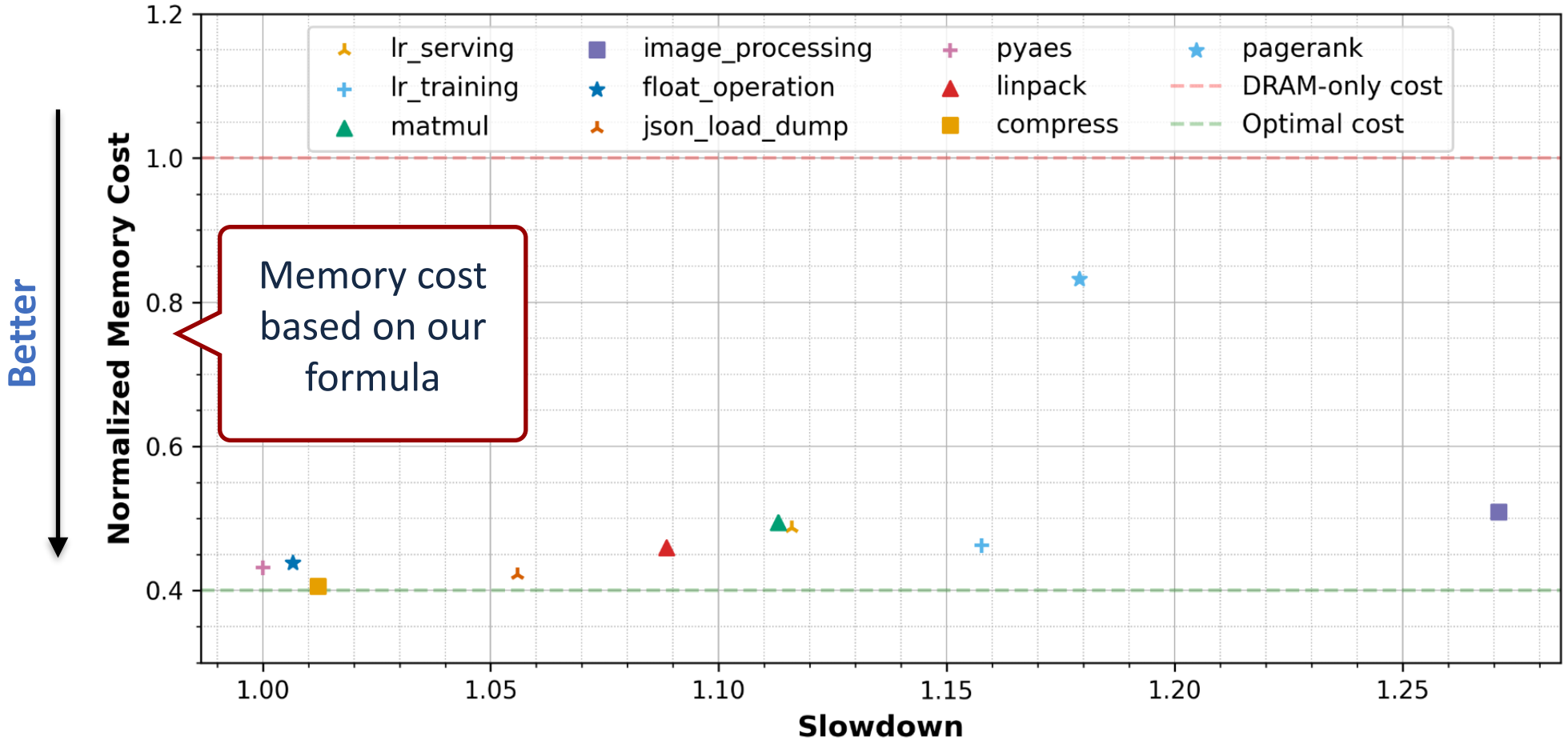
Functions from **FunctionBench** and **SeBS**

- Compression, graph processing, image processing, ML training and inference, microbenchmarks

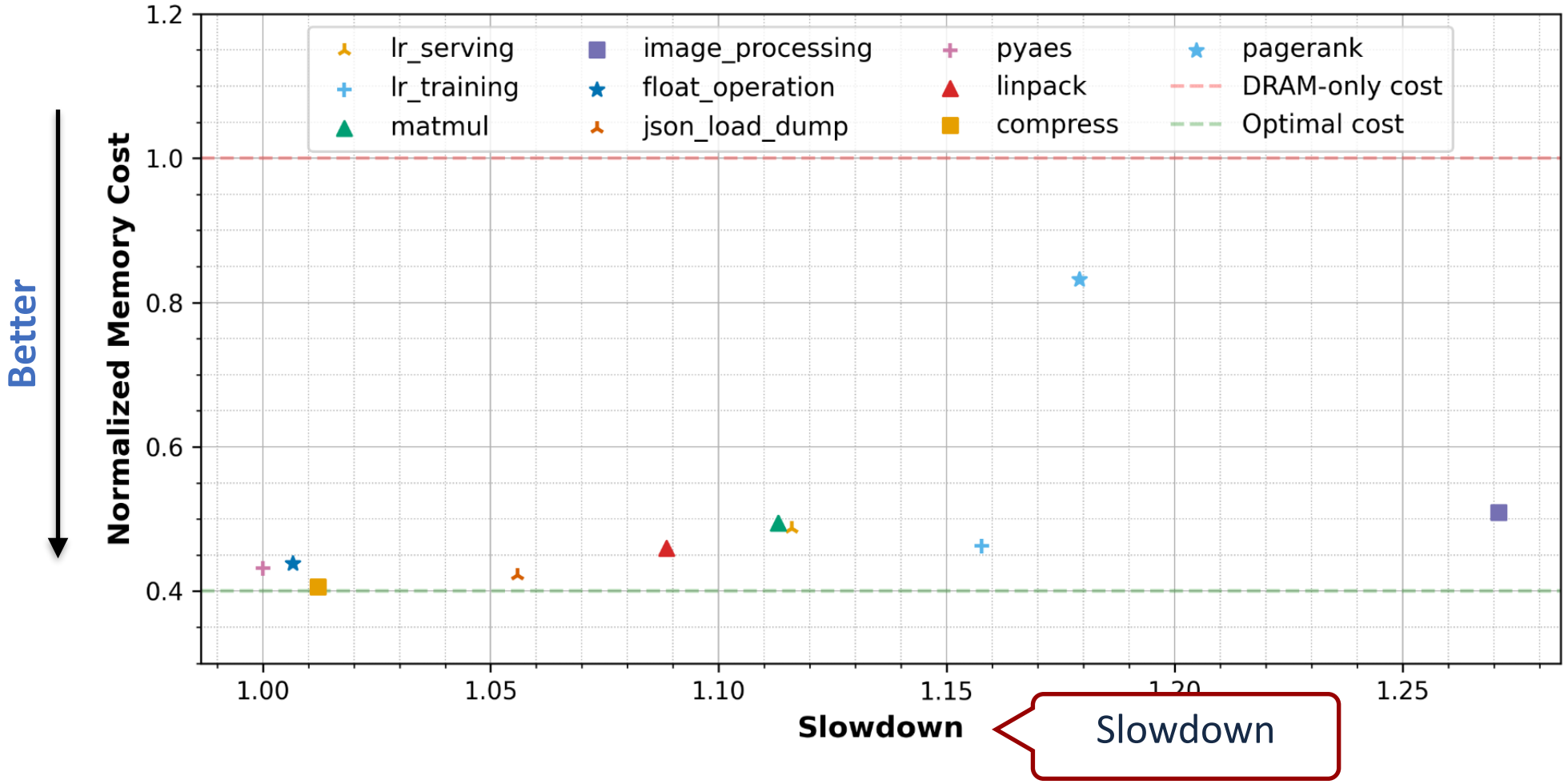
TOSS (input IV, optimal cost configuration) vs **REAP**

- Working set into memory, the rest loaded on-demand
- Reduced page faults & parallelized prefetching

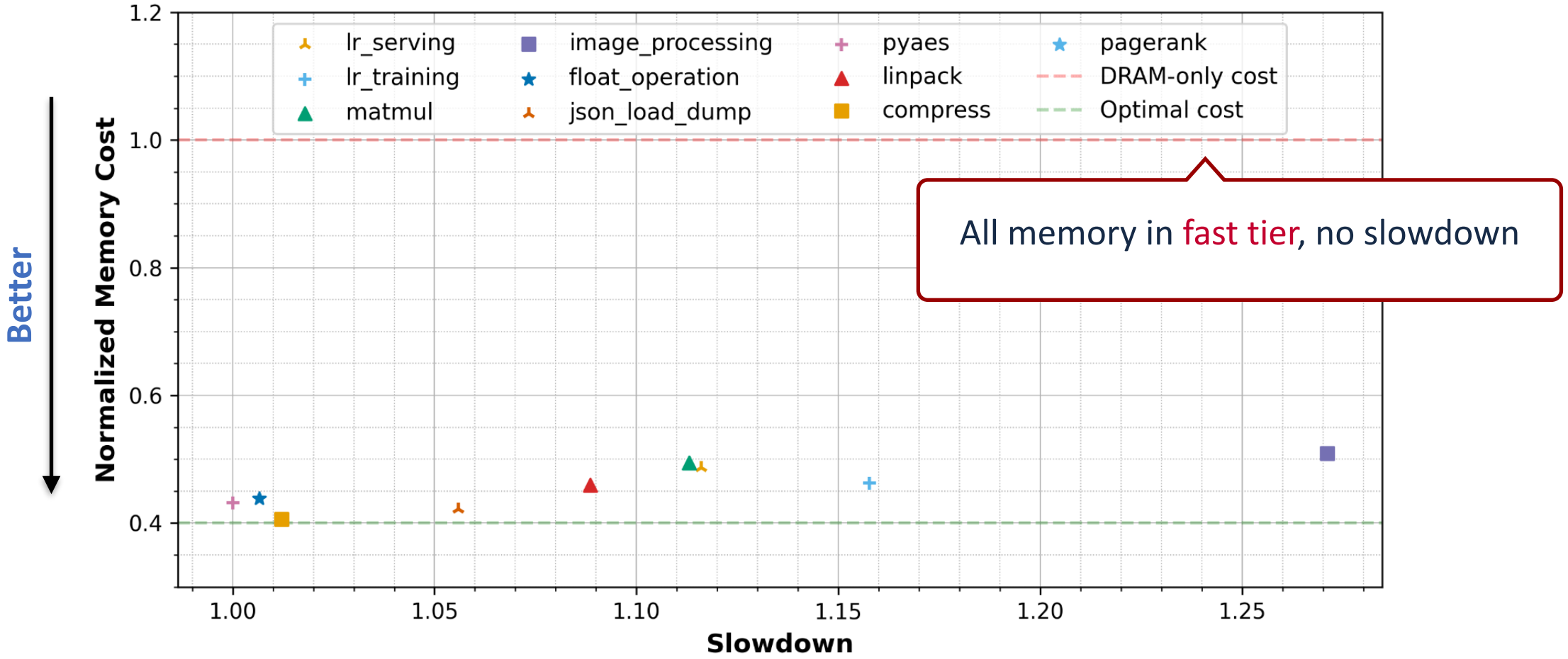
Evaluation - Cost vs Slowdown & Memory offloaded



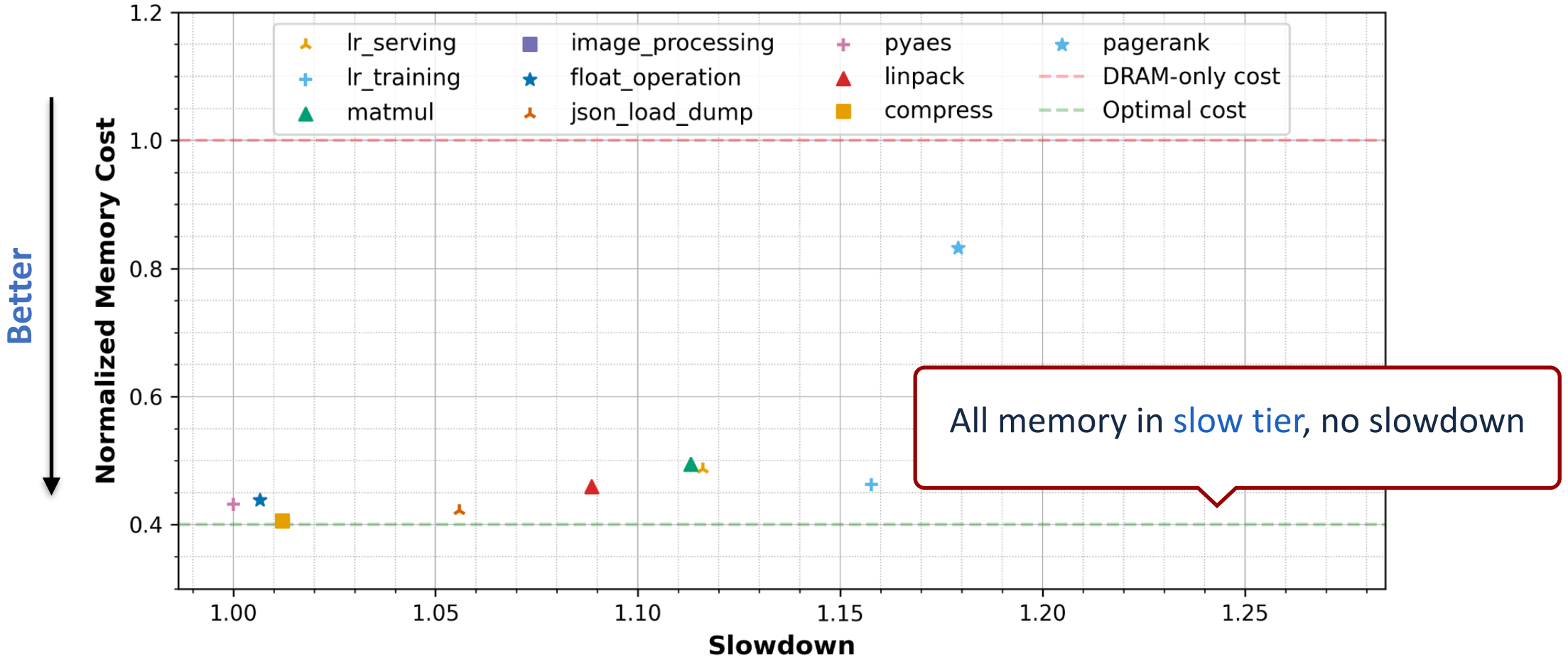
Evaluation - Cost vs Slowdown & Memory offloaded



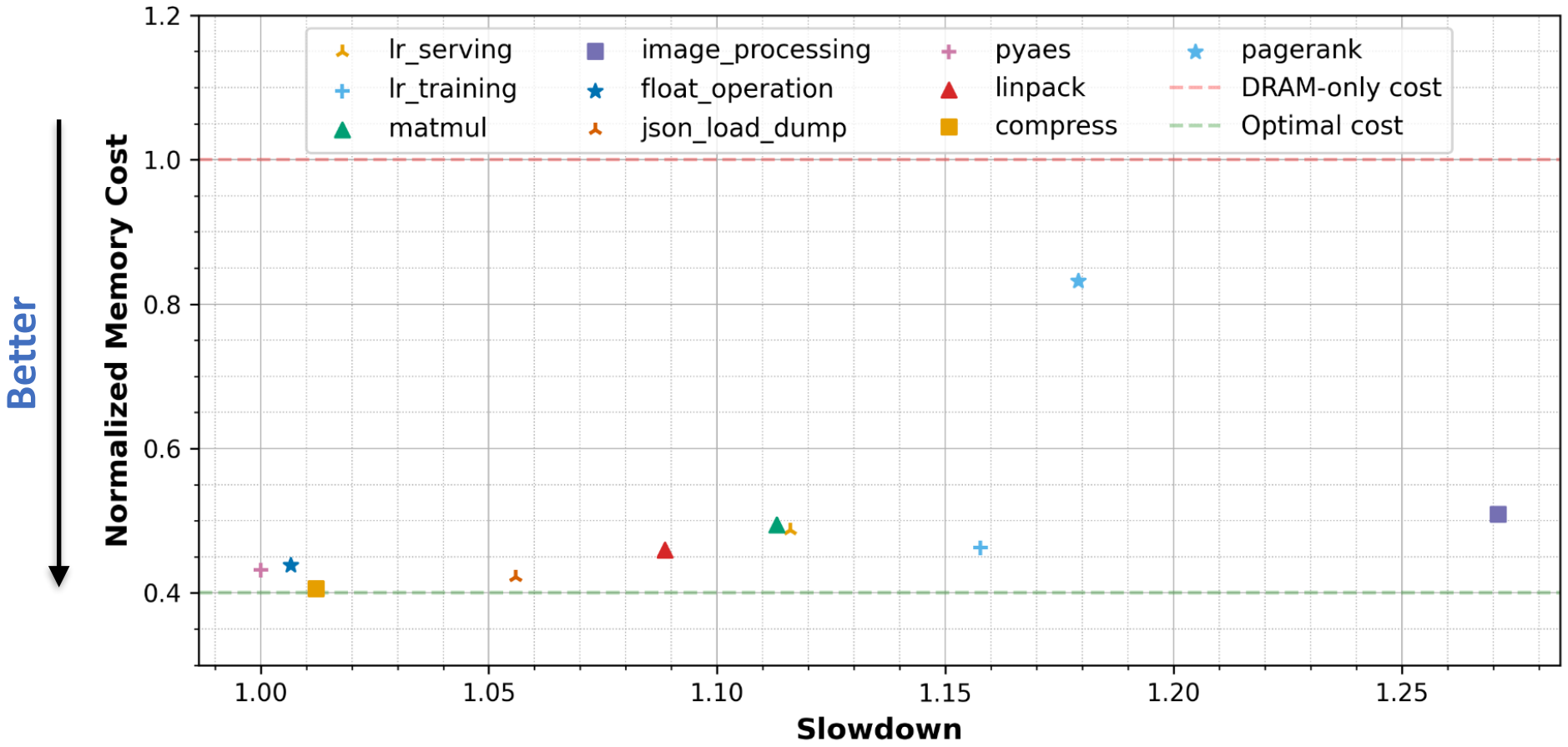
Evaluation - Cost vs Slowdown & Memory offloaded



Evaluation - Cost vs Slowdown & Memory offloaded

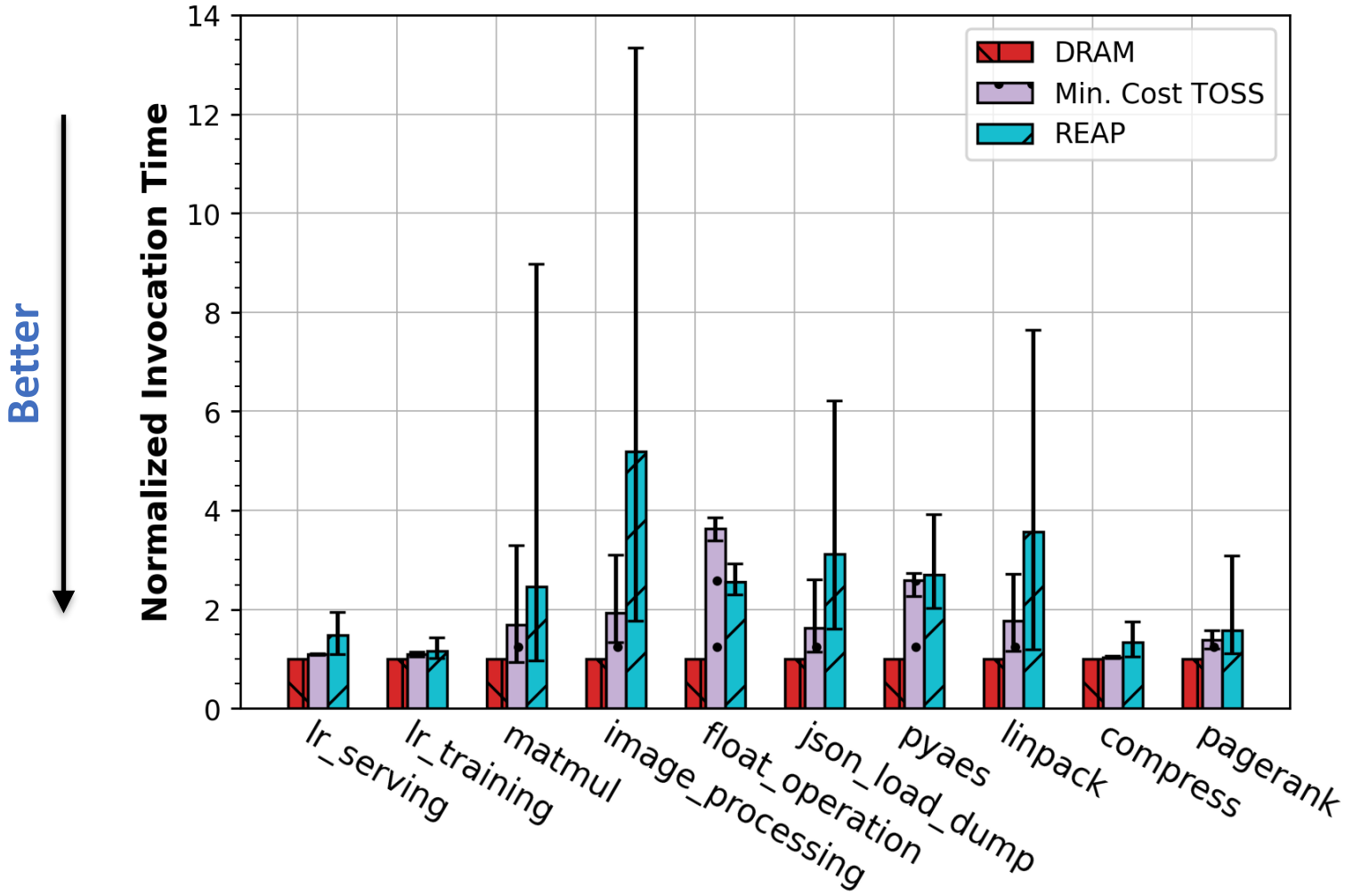


Evaluation - Cost vs Slowdown & Memory offloaded

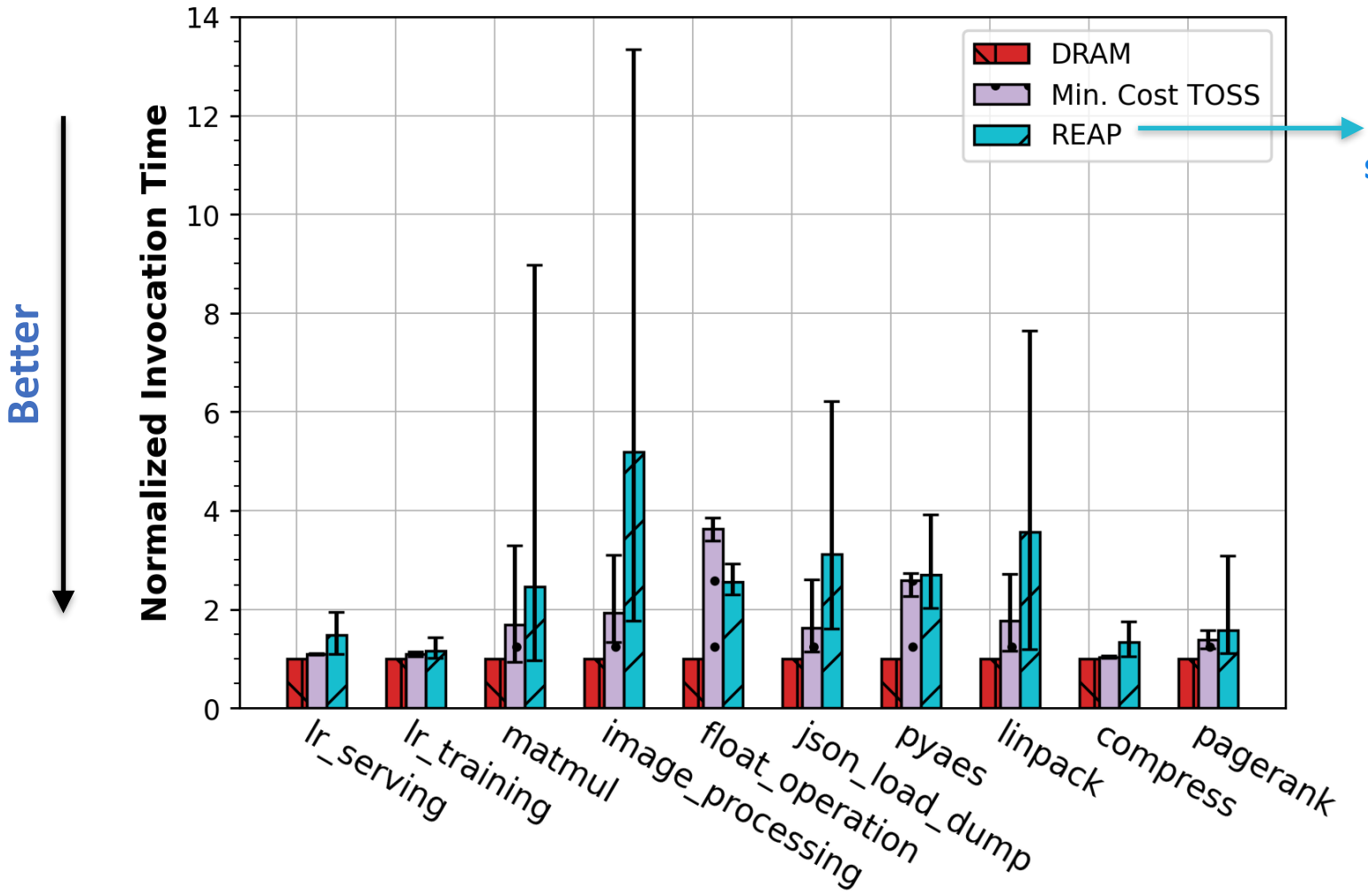


- Memory Cost Reduction
  - Average: **51%**
  - Max: **59% (~ optimal)**
- Function Slowdown
  - Average: **10%**
  - Max: **27%**
- Offloaded memory
  - Average: **92%**
  - Max: **100%**

Evaluation - Total invocation time (Setup & execution)

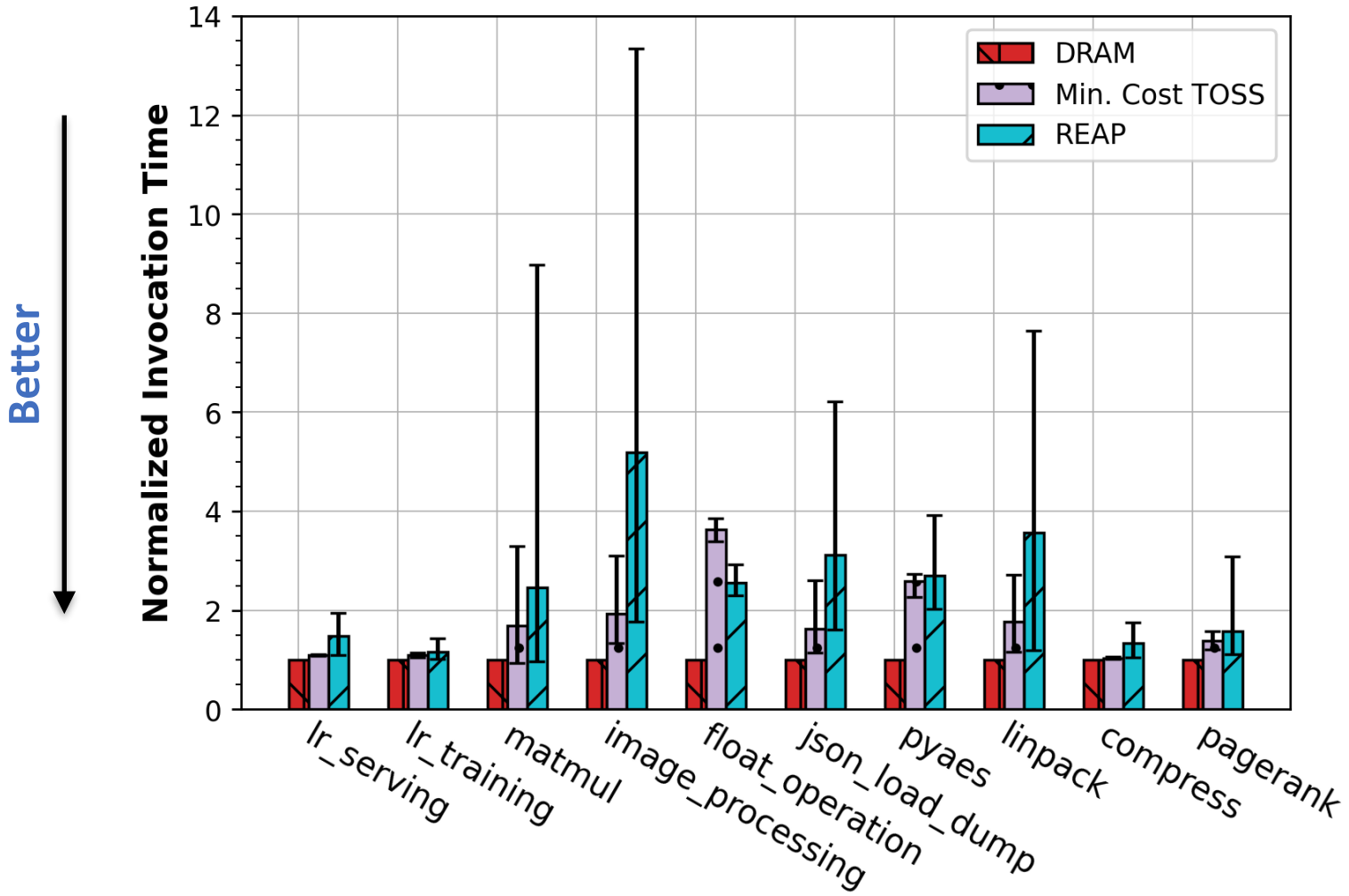


Evaluation - Total invocation time (Setup & execution)



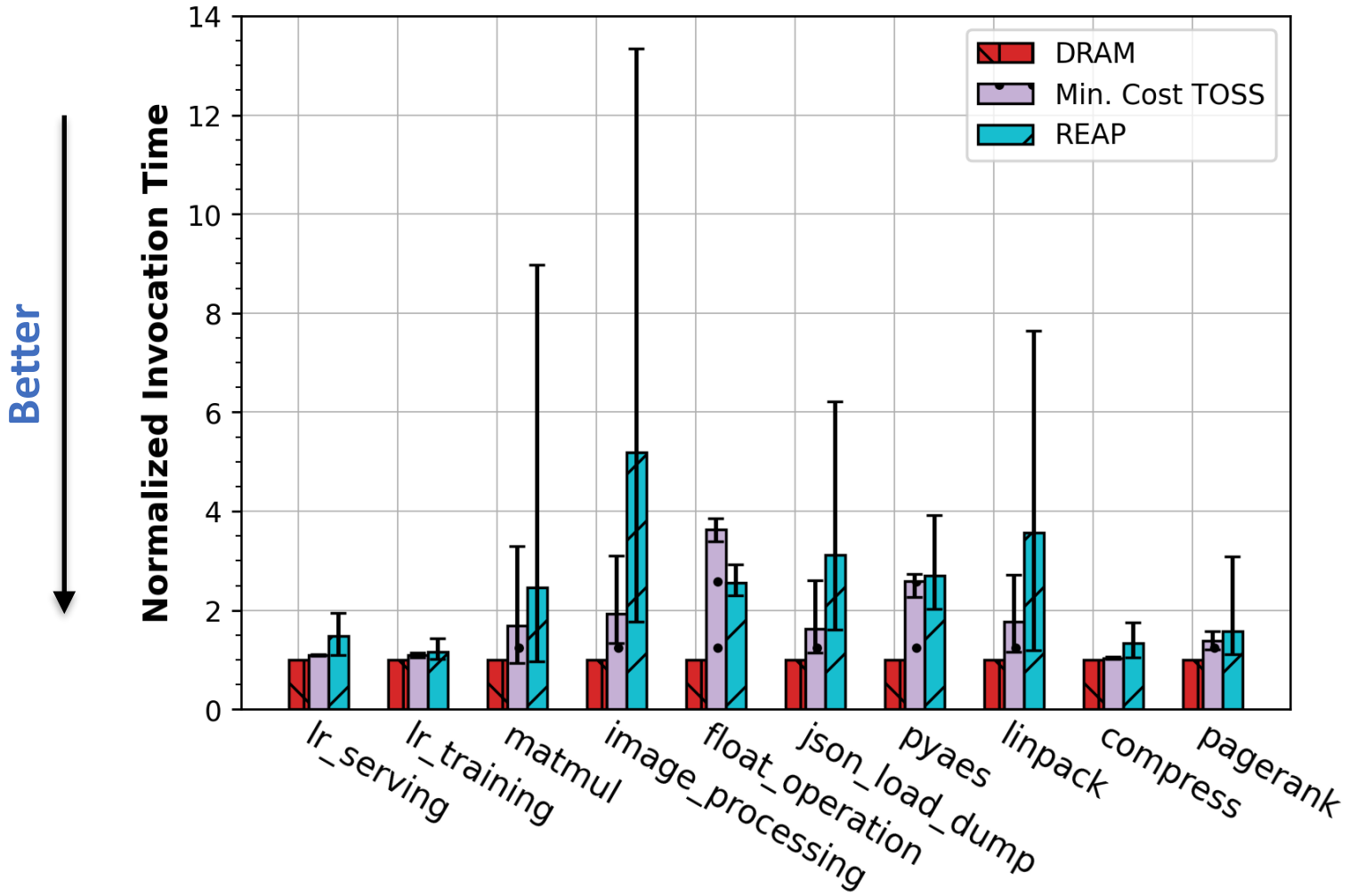
REAP with all combinations of **snapshot** & **execution** inputs

Evaluation - Total invocation time (Setup & execution)



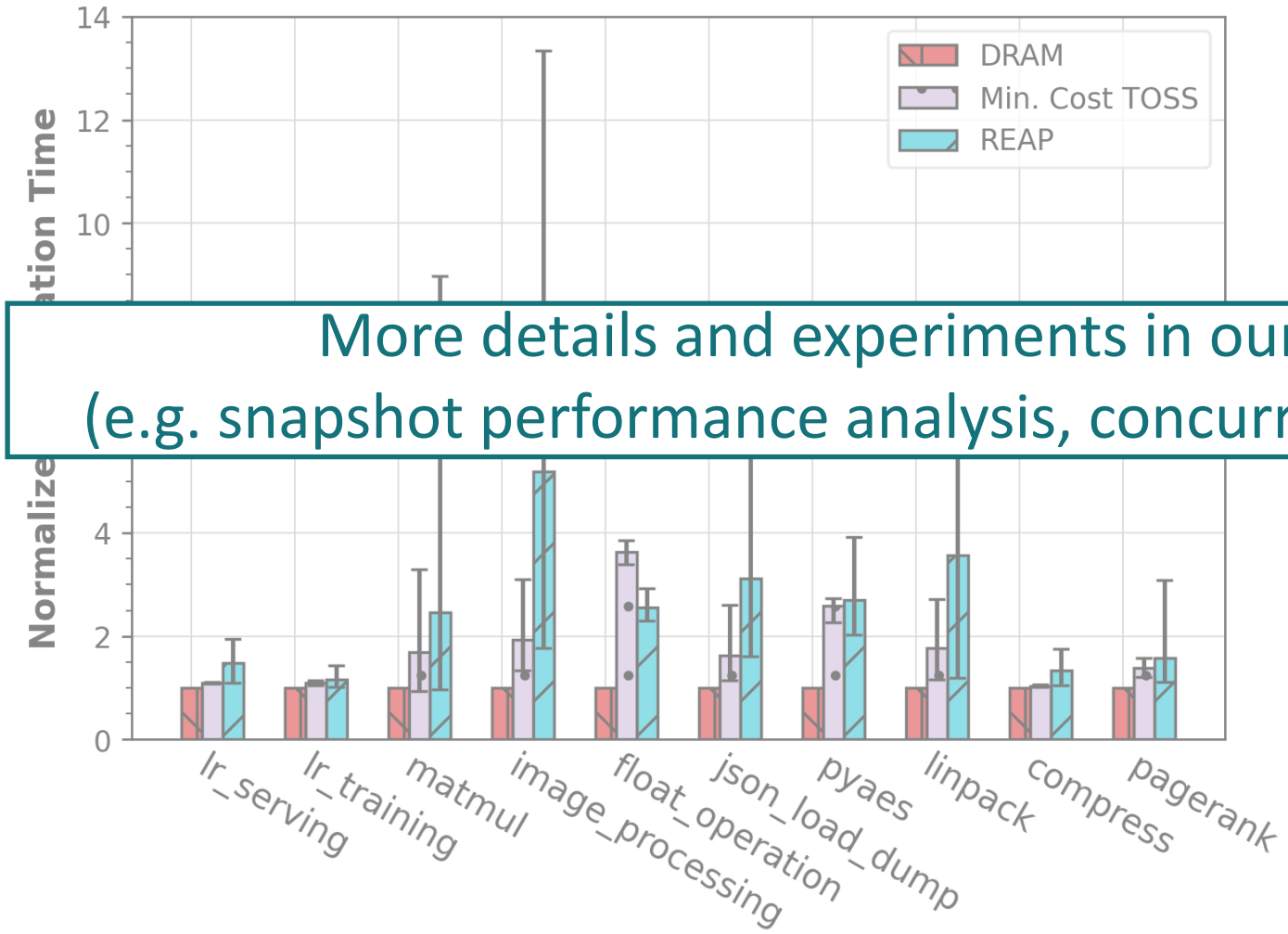
REAP's performance depends on the **similarity** between snapshot & execution input

Evaluation - Total invocation time (Setup & execution)



TOSS achieves on average **1.7×** and up to **4.2×** lower total invocation time

Evaluation - Total invocation time (Setup & execution)



More details and experiments in our paper
(e.g. snapshot performance analysis, concurrent executions)

Summary

- **Problem**: Inflated memory costs by assuming a single memory tier
- **Key insights**
  - Different inputs lead to different memory access patters
  - Functions use heavily a small memory subset → cost-effective memory tiering without major slowdowns
- **TOSS**: First memory tiering mechanism for serverless functions
  - Memory study & characterization
  - Cost model to guide automatic tiered snapshot generation

TOSS: Tiering of Serverless Snapshots for Memory-Efficient Serverless Computing

Questions?