



Expeditious High-Concurrency MicroVM SnapStart in PMem with an Augmented Hypervisor

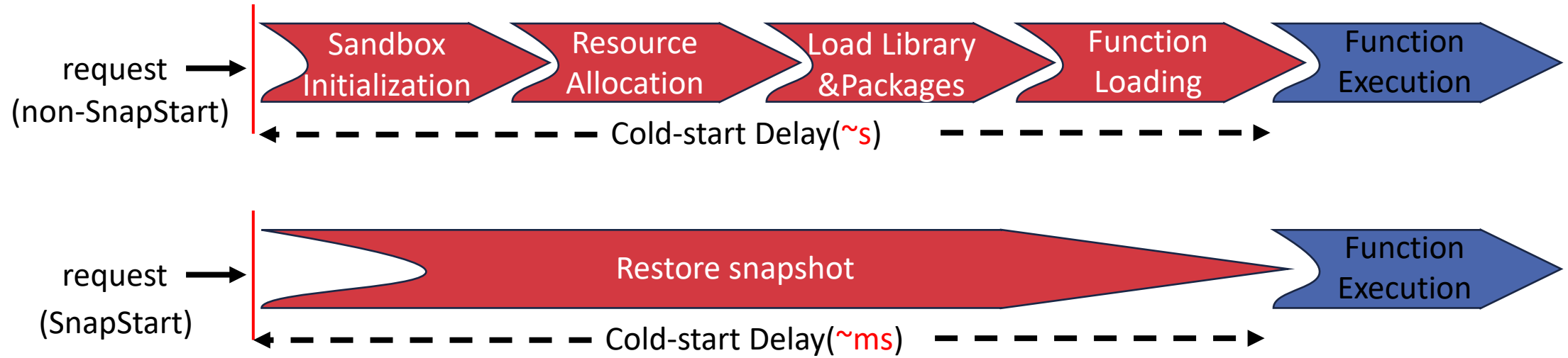
Xingguo Pang, Yanze Zhang, Liu Liu, Dazhao Cheng, Chengzhong Xu, and Xiaobo Zhou

xingguo.pang@connect.um.edu.mo




澳門大學
UNIVERSIDADE DE MACAU
UNIVERSITY OF MACAU

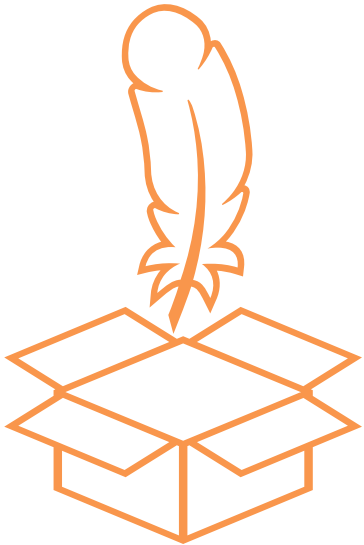
Background: SnapStart



SnapStart is an approach that leverages *snapshot-restore* technique to bypass the time-consuming cold start and expedites function execution by *reusing* previously saved memory state.

Background: MicroVM SnapStart

-  AWS Firecracker[NSDI'20] MicroVM
 - A lightweight VM tailored for short-lived workloads
 - An ideal **sandboxing** solution for *production* SnapStart.



Lightweight Virtualization



VM-like Security



Rapid Startup

Background: Why SnapStart



Booting time mitigation

LightVM [SOSP'17]

Firecracker [NSDI'20]

Pagurus[OSDI'22]

Nephele [EuroSys '23]



Memory state reuse

Keep-alive (warm start)

SEUSS [EuroSys'20]

Icebreaker [ASPLOS'22]

ORION [OSDI'22]



Snapshots

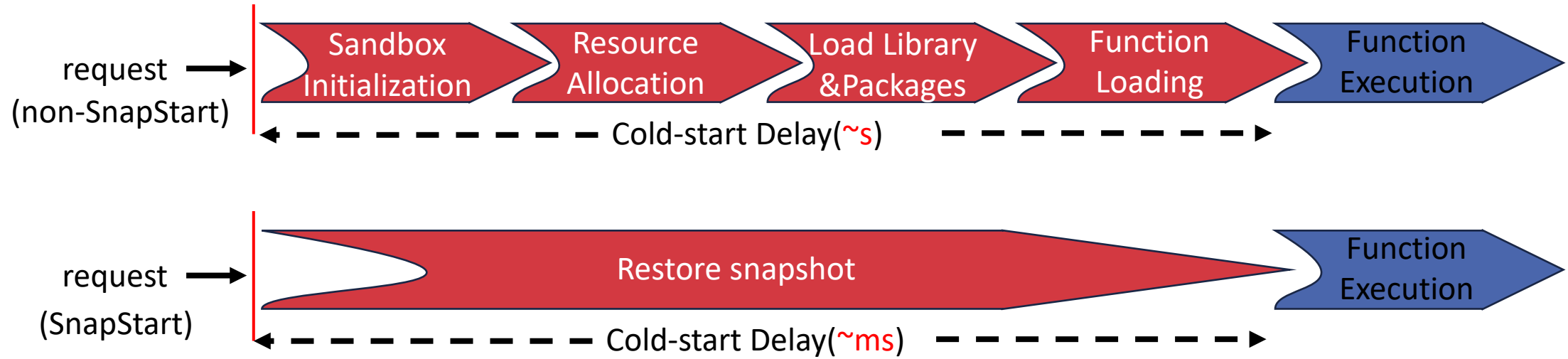
Catalyzer [ASPLOS'20]

Firecracker snapshots

- REAP [ASPLOS'21]
- FaaSnap [EuroSys'22]

SnapStart serves as a tradeoff between cold start latency & memory resource occupation.

Background: SnapStart



- *Fast to start*

- `mmap` memory file (previous snapshot of guest) as guest memory
- Resume VM immediately
- Load guest pages from disk on demand (demand paging)

- *Slow to finish*

- Guest accessing pages not in memory causes major page faults
- Context switches, scheduling, data movement
- Slow down execution

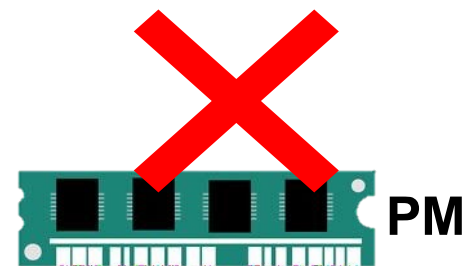
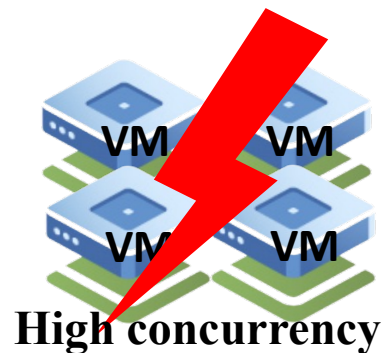
State of the art: FaaSnap [EuroSys'22]

- Reduce major page faults by **prefetching working set**
- Phase 1: **Record guest working set** and save it to a compact file
- Phase 2 (invoke): **Prefetch working set** from disk

- Prefetch stable working set

↘ *fewer* guest major page faults

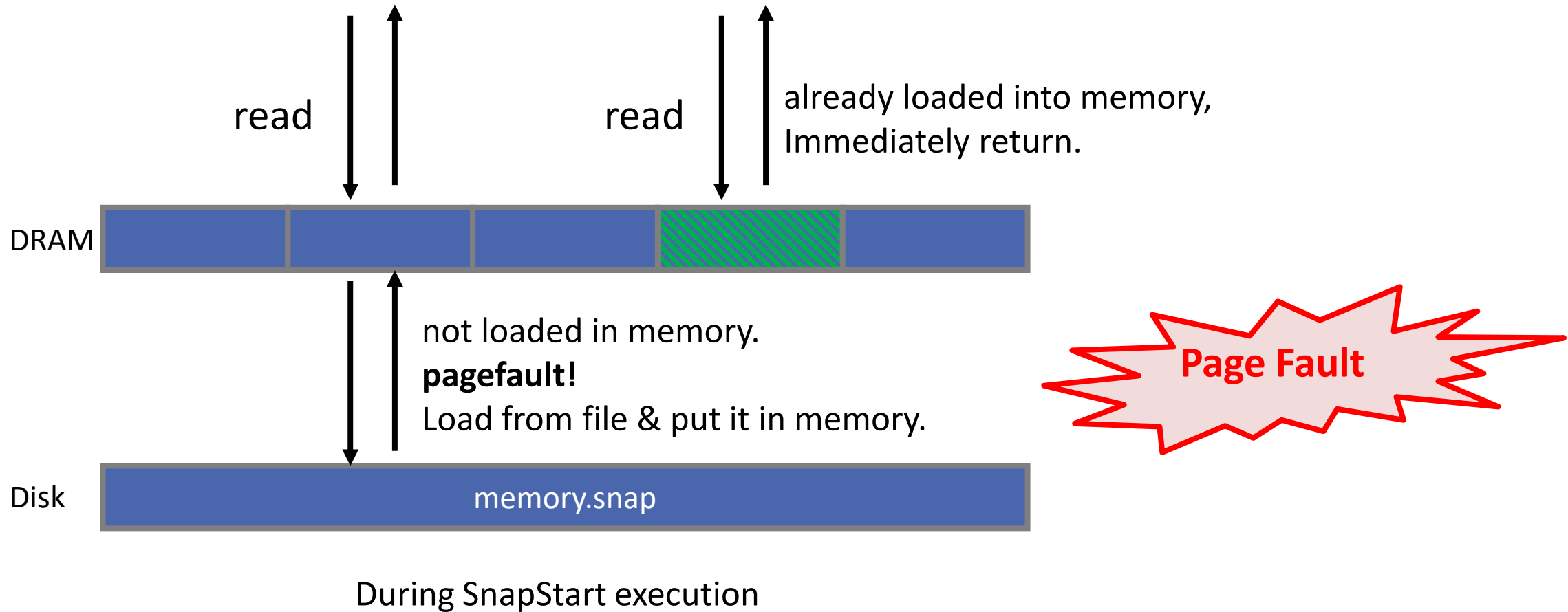
↘ *faster* function execution



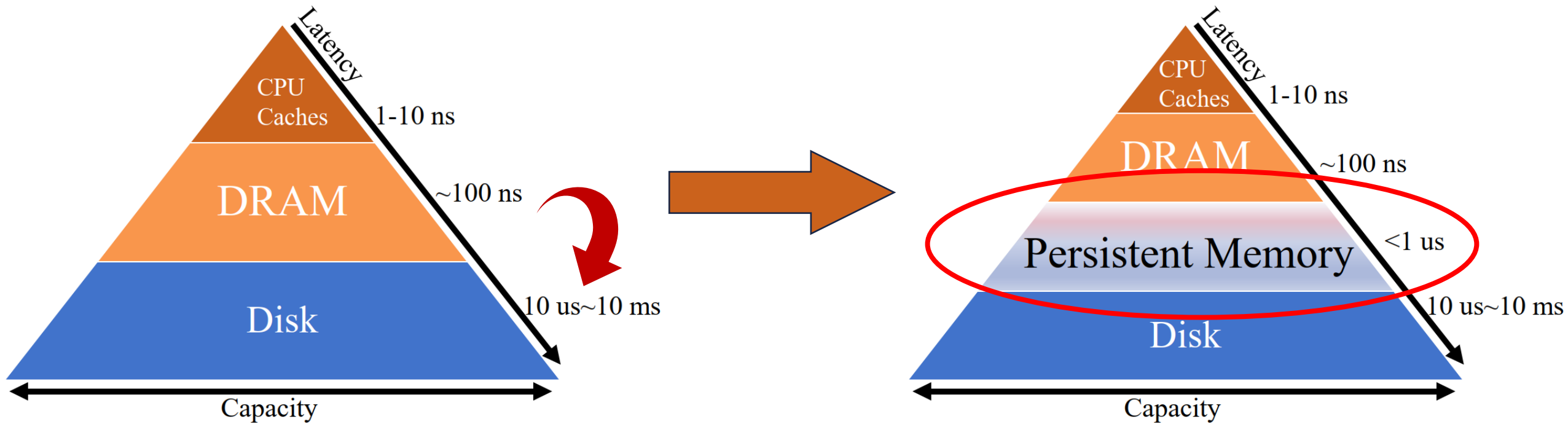
Motivation:

- **Single-tiered** Persistent Memory
- An alternative to replace DRAM+SSD tiered architecture
 - **High cost-effectiveness**
 - ✓ Huge capacity with low price
 - **Good performance**
 - ✓ Persistence & fast recovery capability

Motivation: Single-tiered PMem



Motivation: Single-tiered PMem



- Huge gap
- Costly data movement

- Single-tiered Memory
- Minimal gap

Motivation: Single-tiered PMem

Fast

Byte addressable

Big capacity

Non-Volatile



Cost-effective

Operation latency

100~300 ns

Cacheline & XPlane

64 bytes / 256 bytes

Max capacity

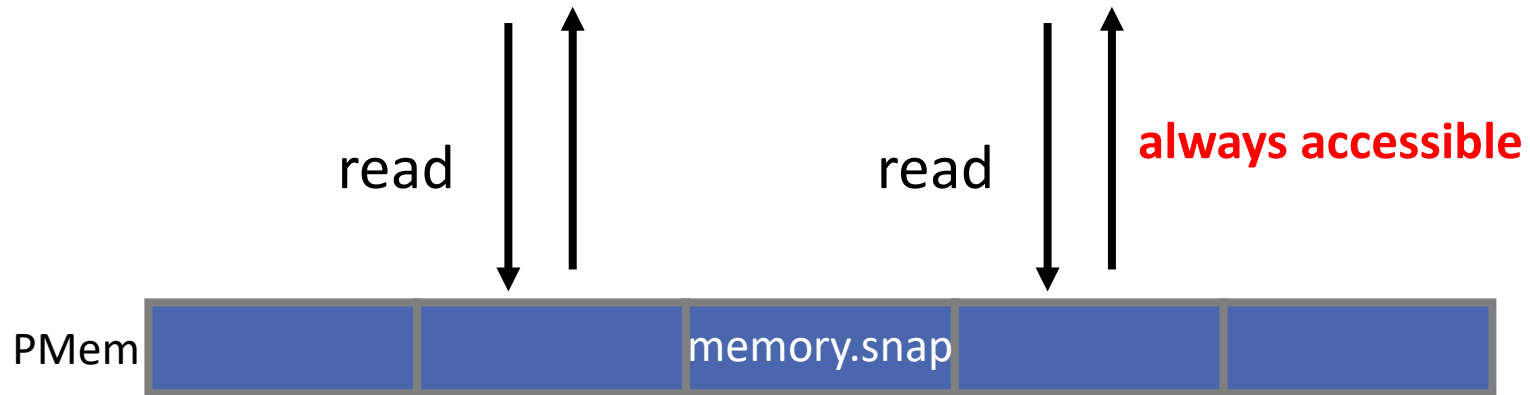
8 TB per machine

persistency

retain data without power

Motivation: Single-tiered PMem

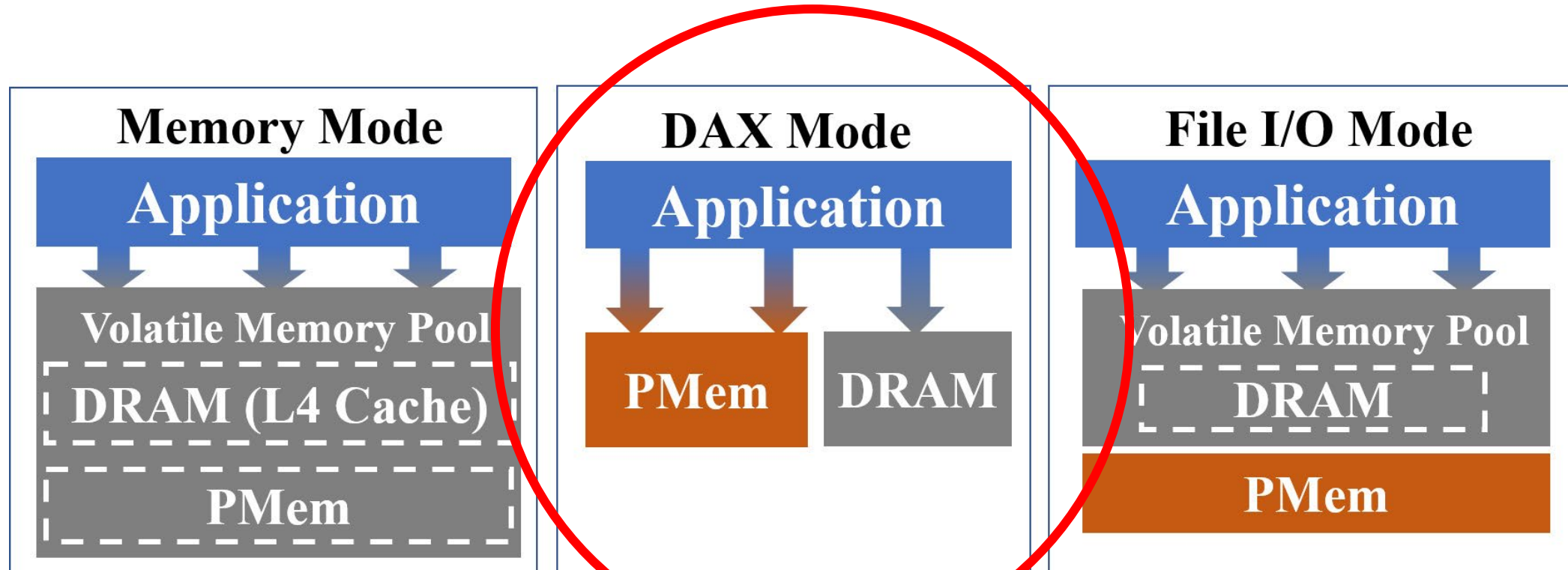
- No tiered access
- No data movement from slow disk.



Stable VM access

Motivation: Single-tiered PMem

- Intel Optane persistent memory instructions
- Unified address management



Different configurations of PMem.

PMem with the orange color (in the PMem and File I/O modes) indicates the use of PMem as persistent storage while the gray color (in the Memory Mode) means its usage as volatile memory.

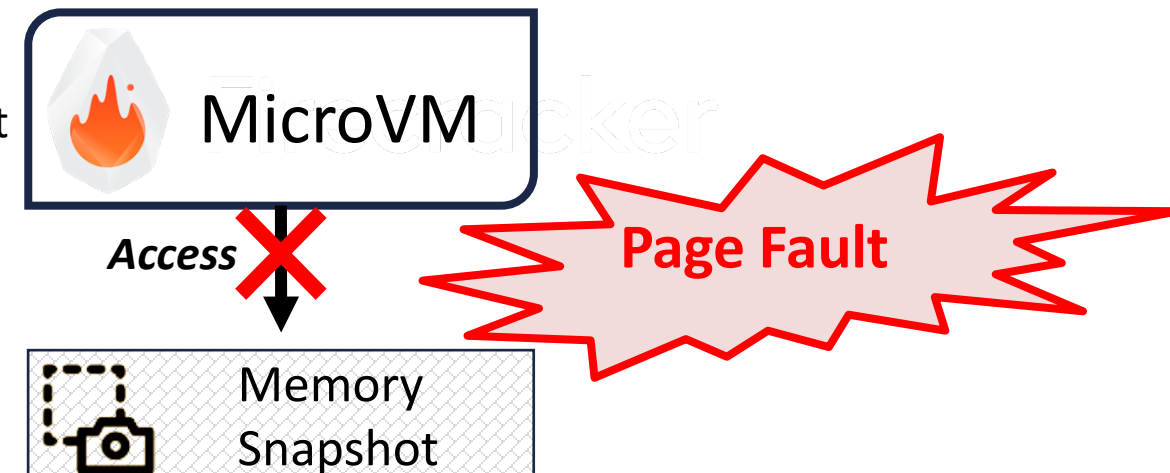
Challenge #01 Efficient page mapping in PMEM

Memory Loading	VM Restoration Time	Function Execution Time	Extra Resource Requirement
Lazy	+	+++	No
Partial	++	++	Yes
Full	+++	+	No

Default Scheme

Efficient page mapping in PMEM.

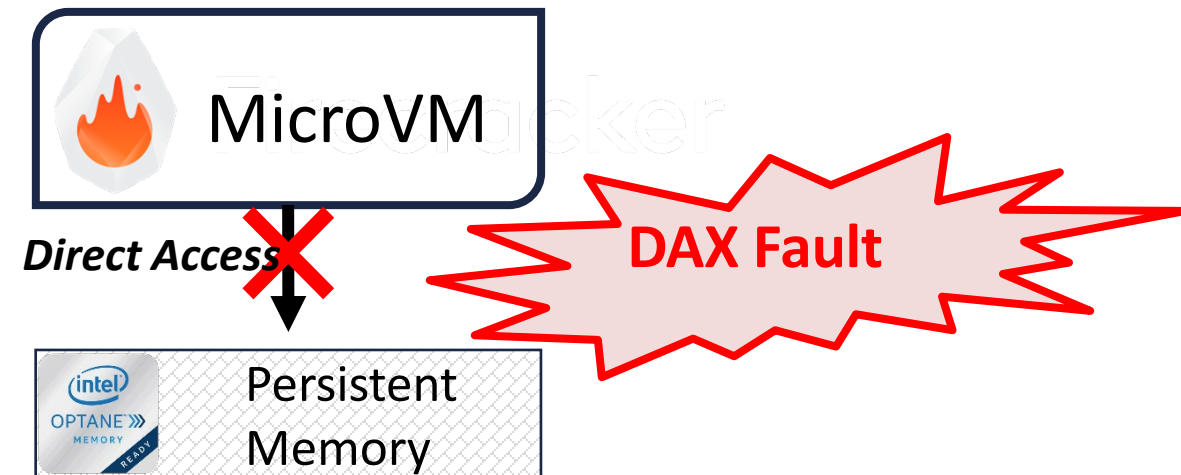
fully leverage PMEM's capabilities for a MicroVM's SnapStart



During SnapStart.

Challenge #02 Direct data access in PMEM.

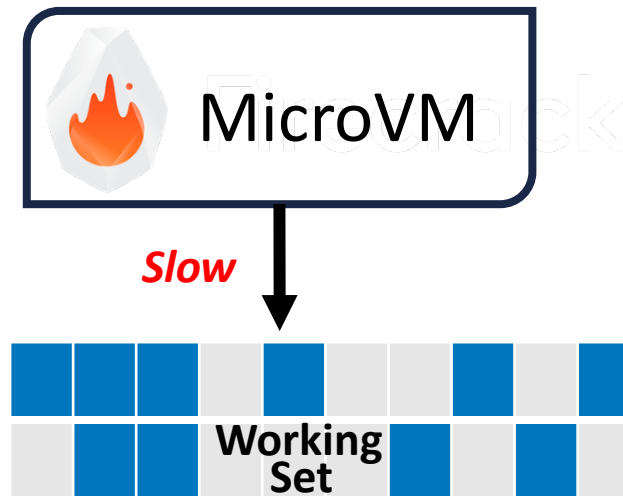
- Direct data access(DAX) in PMEM.
- bypassing cache intermediaries
- Simply mounting snapshots on a DAX-enabled filesystem



During SnapStart.

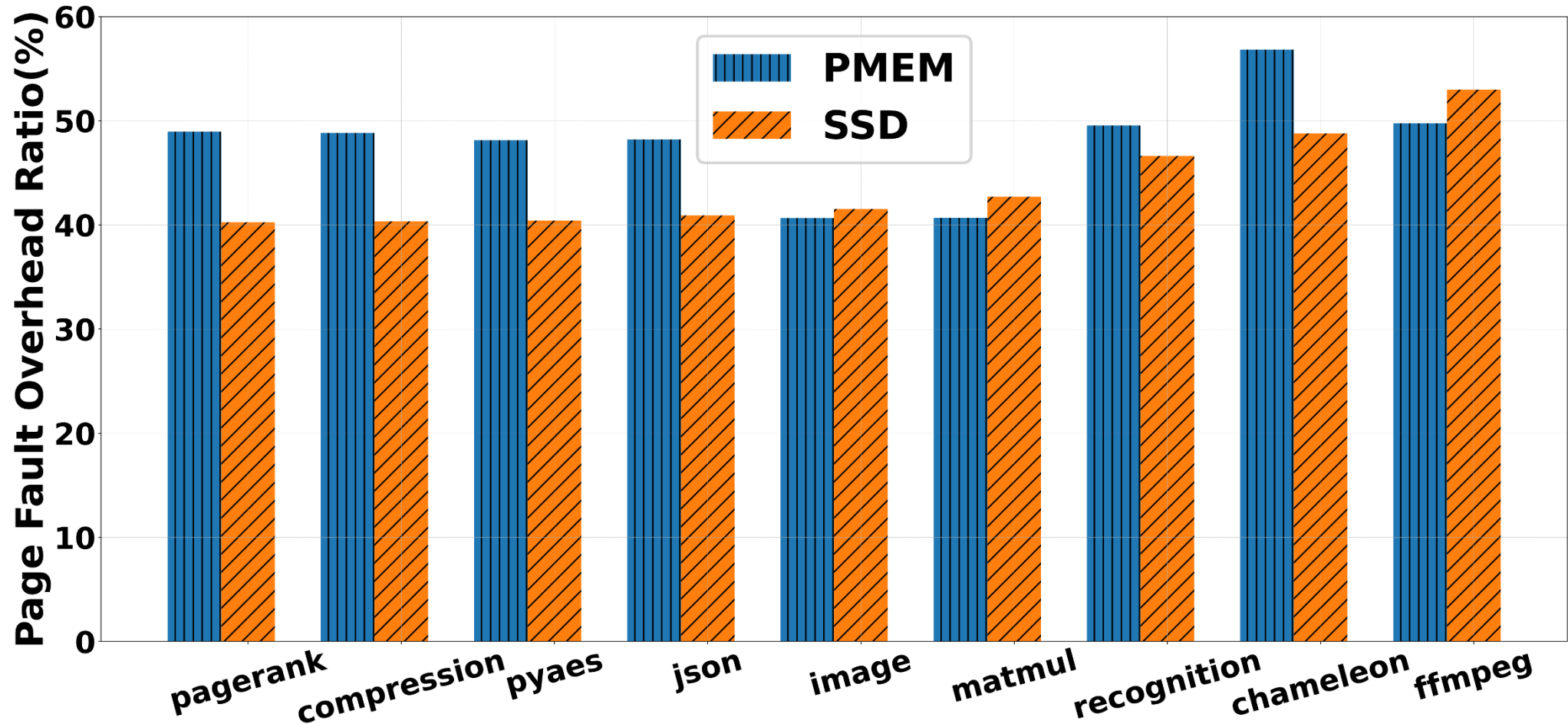
Challenge #03 Ephemeral workloads

- non-iterative, single-access workloads
- ephemeral pages:
 - data accessed temporarily and then quickly discarded
- unnecessary **replication** overhead from traditional caching-based methods



During SnapStart.

Case Study #01: Page Fault Overhead in SnapStart



Page fault overhead in SnapStart execution.

Page fault is a **bottleneck** for end-to-end SnapStart execution in microVM.

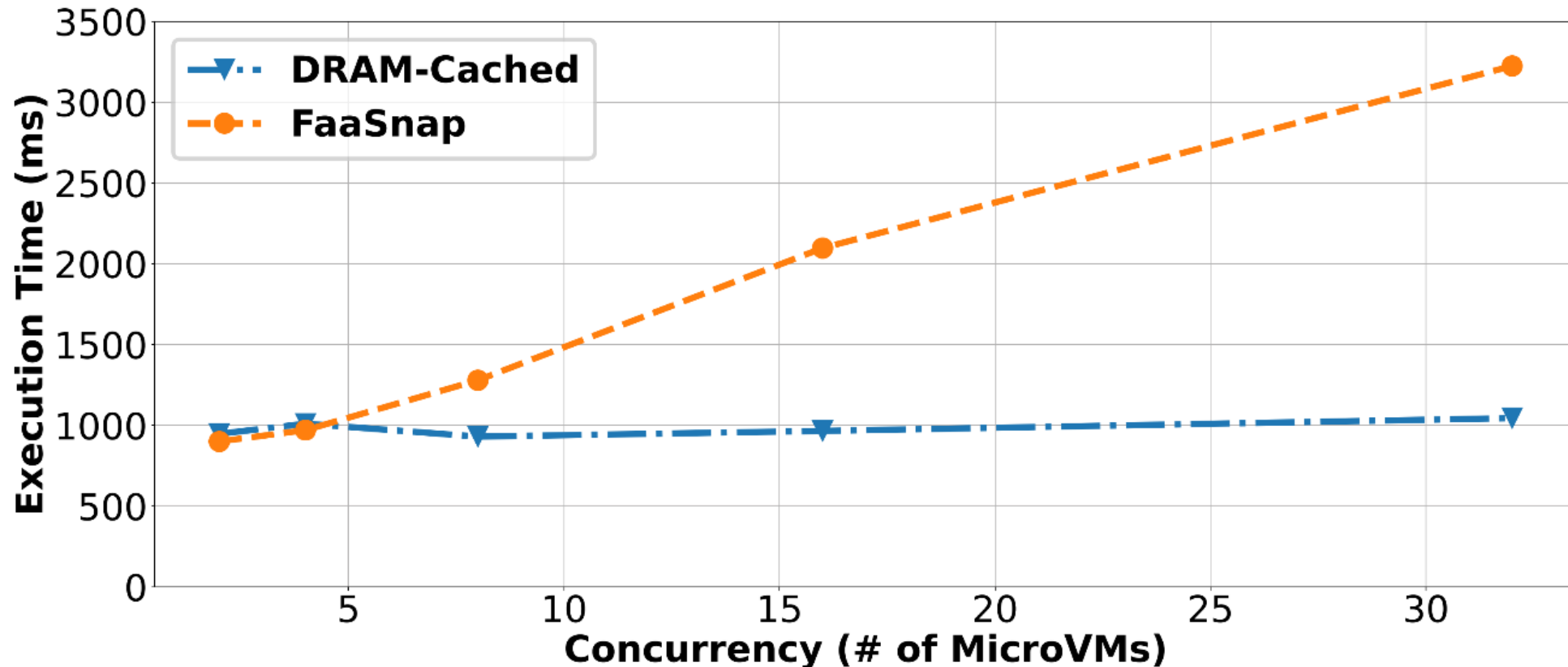
Case Study #02: SnapStart under Ephemeral Workloads

Table: SnapStart performance on an ephemeral workload.

Approach \ Metric	execution time	memory footprint
Lambda SnapStart	240 ms	83 MB
FaaSnap	218 ms	243 MB
DRAM-cached	204 ms	1,024 MB

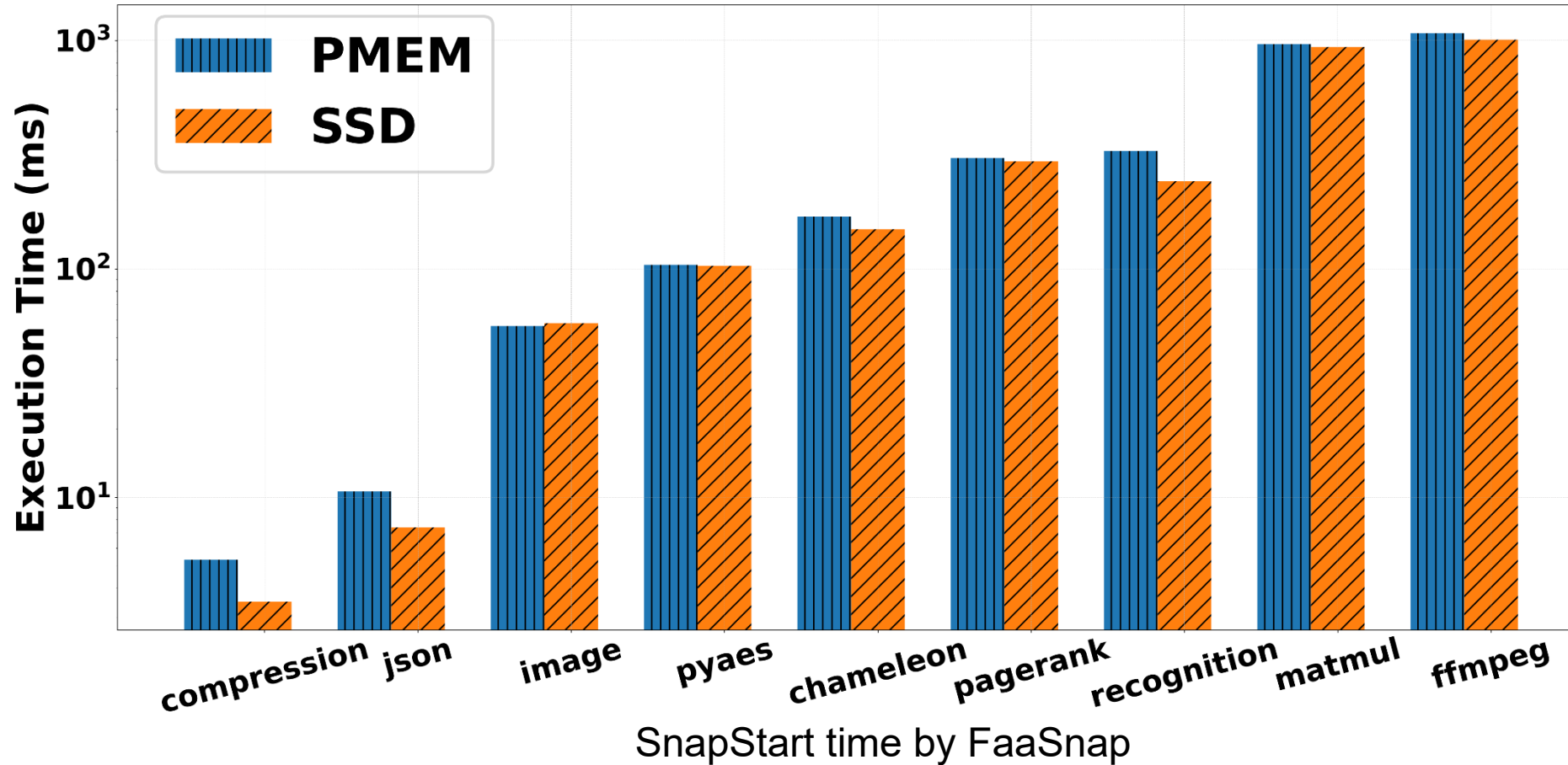
Ephemeral workloads does not have heavy cache dependencies for latency performance.

Case Study #03: SnapStart under High Concurrency



- *FaaSnap has **bad** SnapStart performance under High Concurrency.*

Case Study #04: FaaSnap in a PMEM Filesystem



- *FaaSnap can not work properly for SnapStart on PMem.*

Case Study #05:DAX Faults in a PMEM Filesystem

Event	Time	Proportion
End-to-End Execution	517 ms	100%
Page Fault ^a	301 ms	58.2%
DAX Fault ^b	147 ms	28.4%

^aMeasured by `kvm_mmu_page_fault`.

^bMeasured by `dax_iomap_pte_fault`, a part of page fault overhead.

DAX fault is **severe** to performance of SnapStart execution.

~30% overhead

Case Study Summary

- 1) Page fault overhead critically hinders SnapStart execution time;
- 2) The DAX feature in PMEM can enhance memory access efficiency for certain ephemeral workloads;
- 3) High concurrency significantly degrades FaaSnap's performance;
- 4) FaaSnap's prefetching is incompatible with PMEM DAX feature;
- 5) MicroVM memory snapshots on a PMEM filesystem underutilize DAX capabilities.

Opportunity:

Direct management of PMEM, avoiding traditional filesystems, may optimize SnapStart's use of PMEM's performance potential.



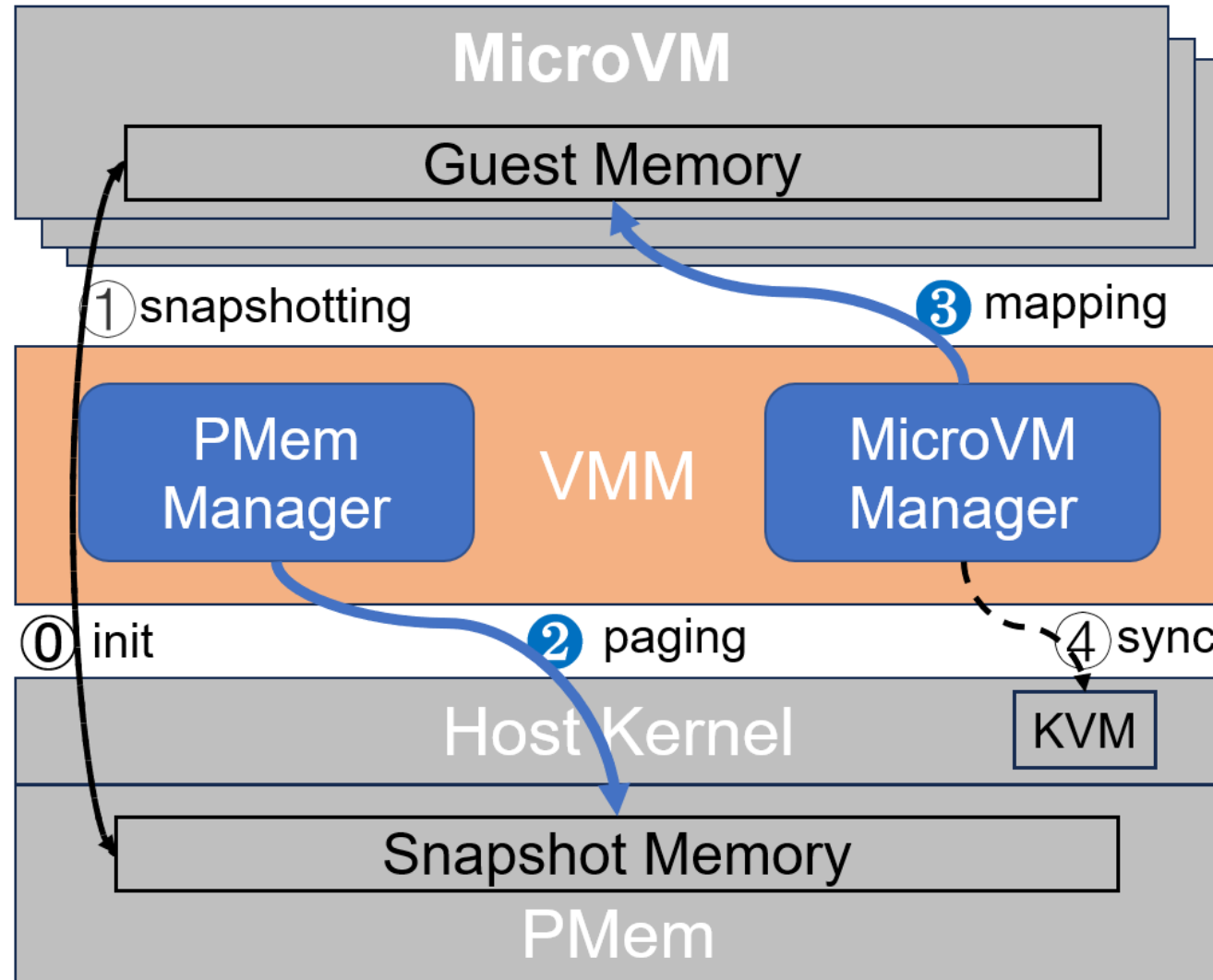
Goal

Reducing time spent on page faults and data movement

Approach

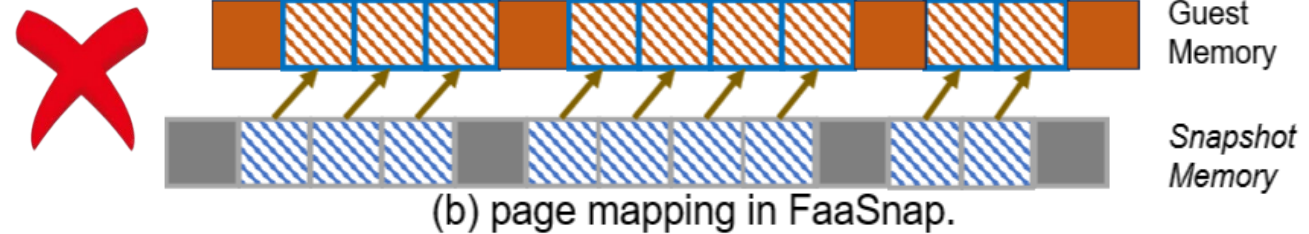
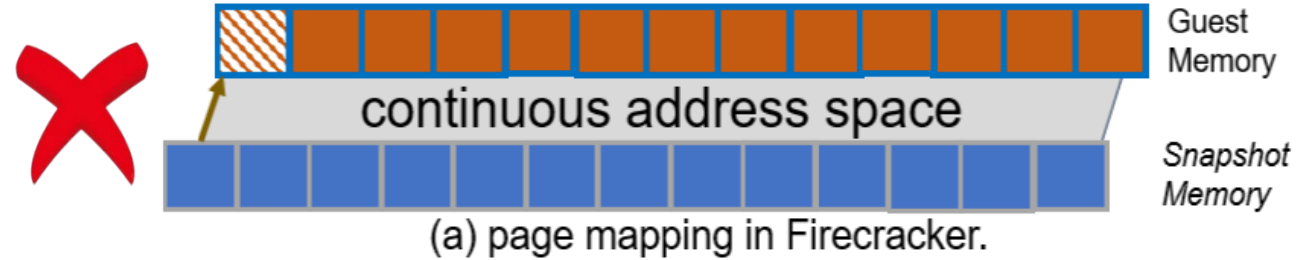
Leveraging *PMem Direct Access* into MicroVM with
Complete Address Indexing

Design Overview

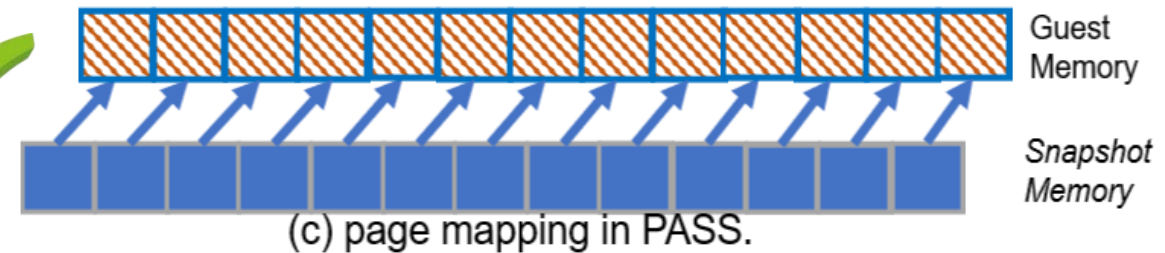


Design #01: Pre-fault Page Mapping

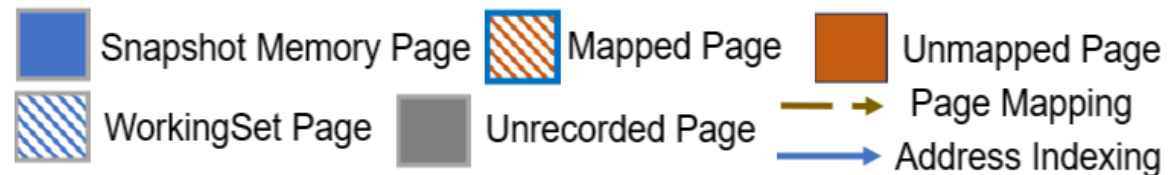
- Complete Address Indexing
 - Memory snapshot to VM memory
 - No fragmented mapping
 - No online mapping reconstruction
 - No data movement(only metadata)



- Address Indexing Lifetime Management
 - VM start to VM shutdown
 - Unique for this VM

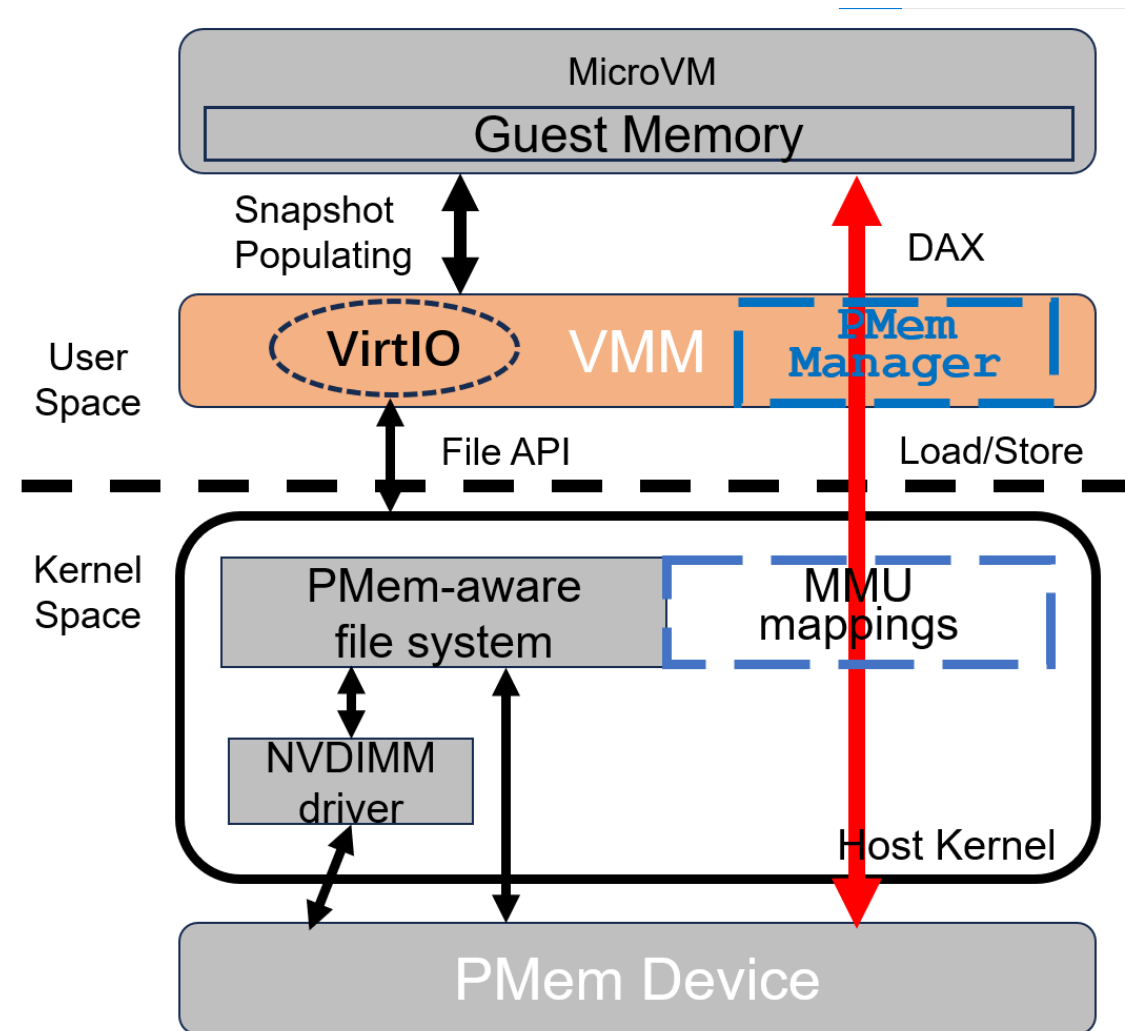


- Address Indexing Synchronization
 - Sync to KVM space



Design #02: Zero-copy On-demand Paging

- Embedded PMem manager
- Space is stable
- File system is redundant.
- Less context switch
- User-space memory
 - Memory load/store
 - Direct access



MicroVM -> VMM -> PMem-aware filesystem -> VMM -> MicroVM
MicroVM -> VMM (PMem manager) -> MicroVM

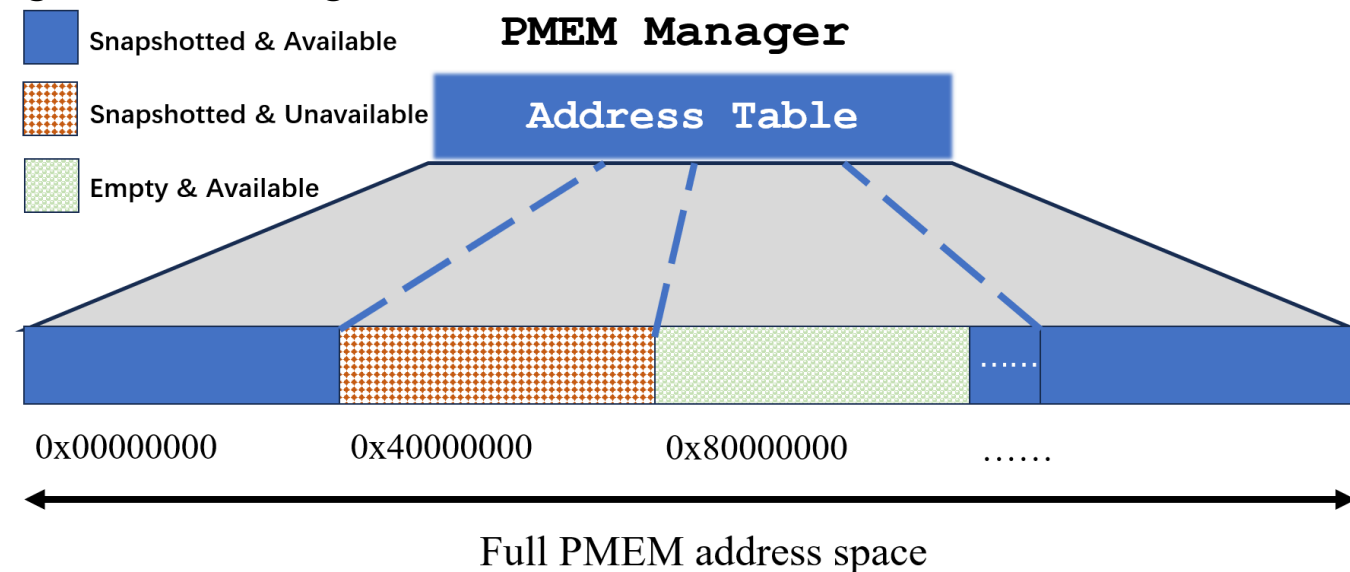
Design:

- VM-Snapshot Memory match

- Build a match between a microVM and its snapshot memory on PMem
- replacing the need for snapshot file locating in a DAX file system.

- Pre-built Indexing on PMem

- Pass pre-built indexing (hash table) from PMem to guest memory
- Avoids rebuilding the mapping and frequent page faults during VM execution



❖ More Details

- PMem Address Management
- Programming Interface
-

❖ *Refer to our paper for more details!*



Expeditious High-Concurrency MicroVM SnapStart in Persistent Memory with an Augmented Hypervisor

Xingguo Pang¹, Yanze Zhang¹, Liu Liu¹, Dazhao Cheng², Chengzhong Xu¹, Xiaobo Zhou^{1*}

University of Macau¹ Wuhan University²

Abstract

The industry has embraced snapshotting to tackle the cold starts and efficiently manage numerous short-lived functions for microservice-native architectures, serverless computing, and machine learning inference. A cutting-edge research approach FaaSnap, while innovative in reducing page faults during on-demand paging through prefetching the profiled working set pages into DRAM, incurs high caching overheads and I/O demands, potentially degrading system efficiency.

This paper introduces PASS, a system leveraging byte-addressable persistent memory (PMEM) for cost-effective and highly concurrent MicroVM SnapStart execution. PASS, functioning as a PMEM-aware augmented hypervisor in the user space, revolutionizes MicroVM memory restoration. It constructs complete address indexing of the guest memory mapped to single-tier PMEM space, enabling zero-copy on-demand paging by exploiting PMEM's direct access feature. This approach bypasses the cache layer and maintains guest OS transparency, avoiding invasive modifications. Experimental results, derived from real-world applications, reveal that PASS substantially decreases SnapStart execution time, achieving up to 72% reduction compared to the Firecracker hypervisor on the PMEM filesystem, and 47% reduction compared to FaaSnap. Moreover, PASS achieves double the maximum concurrency compared to both Firecracker and FaaSnap. It improves the cost-effectiveness by 2.2x and 1.6x over the Firecracker and FaaSnap, respectively.

1 Introduction

The cold start issue, characterized by the latency incurred during instance initialization, significantly impacts short-lived functions, leading to extended response times and negative user experiences [4, 15, 23, 26, 33]. To address this issue, the industry is increasingly adopting a snapshot-based approach, particularly in MicroVM environments. This approach, known as SnapStart, leverages a hypervisor feature

*The corresponding author. Email: waynezhou@um.edu.mo.

to perform comprehensive memory state checkpointing of MicroVMs, storing these states as files. SnapStart dramatically reduces startup times by restoring a MicroVM's memory from a pre-saved snapshot, thus bypassing the time-consuming process of initializing and setting up dependencies from scratch. Beyond accelerating startup times, SnapStart also shortens overall execution times. This is particularly beneficial for short-lived functions in microservice-native architectures [13, 14, 18, 52], serverless computing frameworks [32, 36, 44], and machine learning inference workloads [21, 47, 51], where minimizing latency is crucial.

In current production platforms, such as AWS Lambda SnapStart [35], the MicroVM snapshot restoration process encounters a significant performance bottleneck. This issue stems from frequent page faults during on-demand paging, particularly problematic within modern tiered memory architectures. While lazy loading techniques are implemented to reduce initialization time and improve memory efficiency, they inadvertently cause a high frequency of page faults, which in turn, significantly slow down function execution.

FaaSnap, a forefront research approach cited in [7], introduces a non-blocking method that prefetches the profiled working set pages into DRAM, thus accelerating MicroVM SnapStart execution. This technique involves copying pages in batches of 1,024 into user-space memory buffers before remapping them, significantly reducing the overhead typically associated with page faults. However, each prefetching cycle involves a notable pre-warm time due to the movement of data from disk to DRAM. This becomes inefficient, especially for ephemeral workloads where the majority of pages are accessed only once, leading to the underutilization of resources by caching snapshots into DRAM. Furthermore, the reliance on DRAM caching limits the capacity for concurrent SnapStart executions. The intensive prefetching also demands a higher level of I/O, in contrast to on-demand methods like AWS Lambda SnapStart, where pages are loaded more gradually. This approach, while reducing memory overhead, can interfere with concurrent workload disk operations, potentially deteriorating the overall system efficiency.

Evaluation

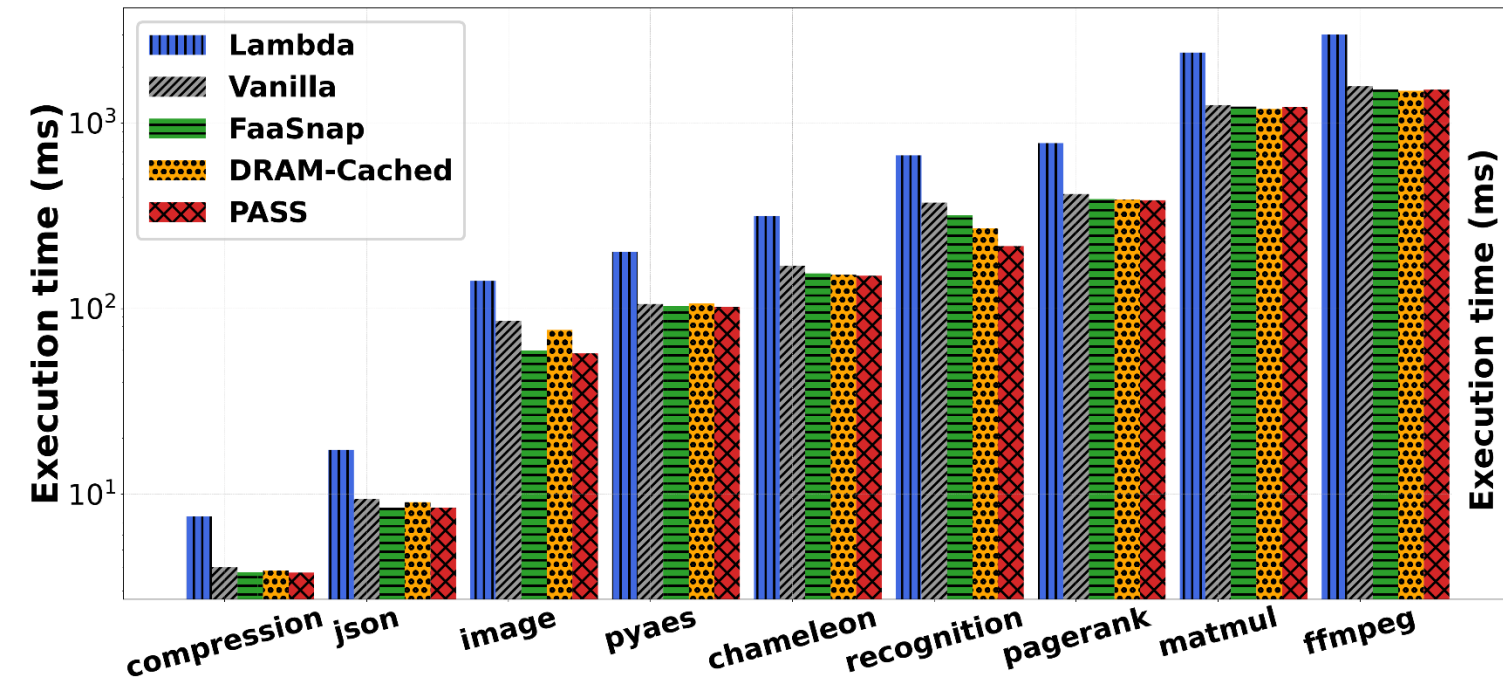
- For comparison with PASS, we evaluated the following approaches for SnapStart execution:
 - **Lambda SnapStart**: The standard approach on SSD.
 - **Vanilla**: Lambda SnapStart on the PMEM filesystem(ext4-dax), enabling DAX for performance enhancement
 - **FaaSnap**: The state-of-the-art. It accelerates MicroVM SnapStart by employing a prefetching technique.
 - **DRAM-Cached**: While effective, it is unsuitable for production platforms due to substantial memory demands.

- Variety of applications
- Intel Optane PMEM 200 series
 - an Interleaved four-DIMM
 - 512GB
 - in AppDirect mode
- 1vCPUs, 1GB RAM per function

Function	Description	Input
Compression	file compression	file
Json	deserialize and serialize json	json
Image	rotate a JPEG image	JPEG
Pyaes	AES encryption	string
Chameleon	render HTML table	table size
Recognition	PyTorch ResNet image recognition	JPEG
PageRank	igraph PageRank	graph size
Matmul	matrix multiplication	matrix size
FFmpeg	apply grayscale filter	video

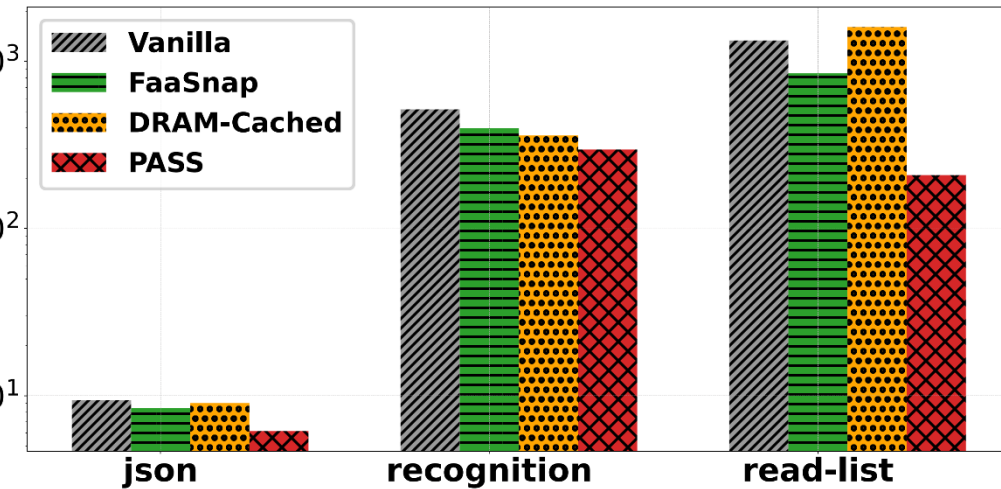
Functions from FunctionBench [Cloud'19],
Sprocket [SoCC'18], and SeBS [Middleware'21]

Evaluation: End-to-End SnapStart Time



SnapStart execution time of different approaches.

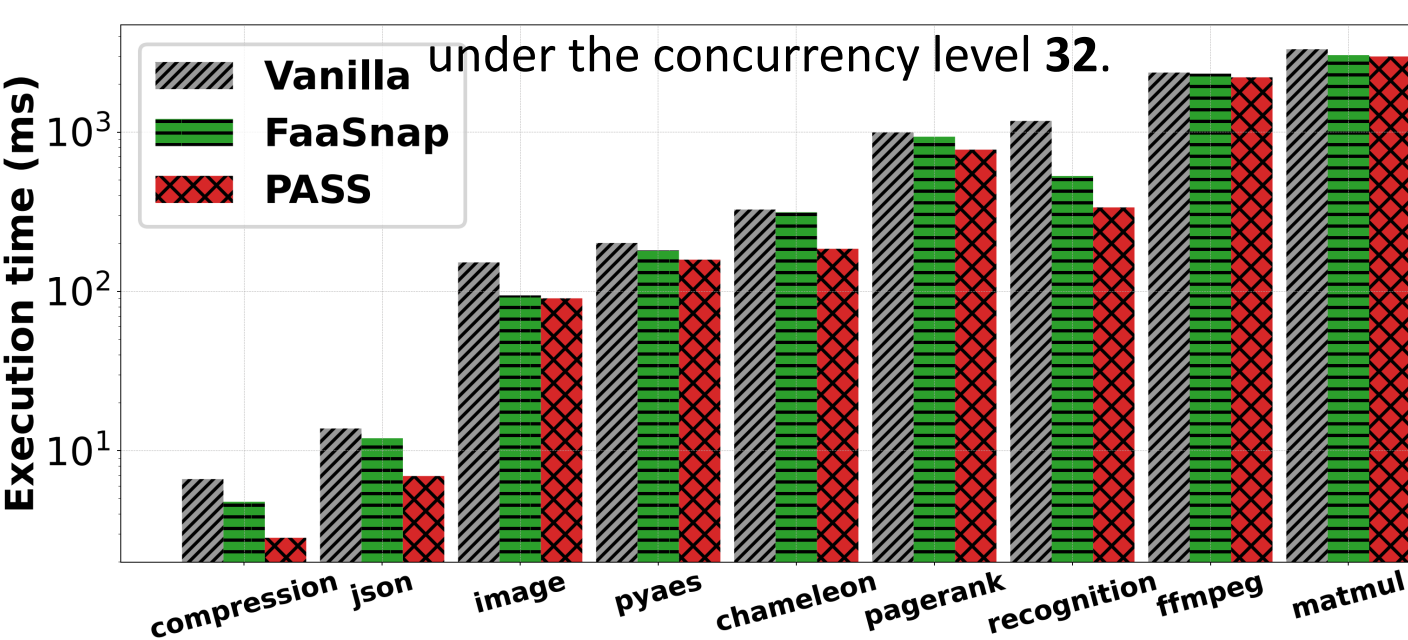
improved SnapStart time by **1% to 47%** v.s. FaaSnap
3% to 72% v.s. Vanilla
(at a worst-case) less than 3% v.s. DRAM-Cached



Execution latency of ephemeral workloads.

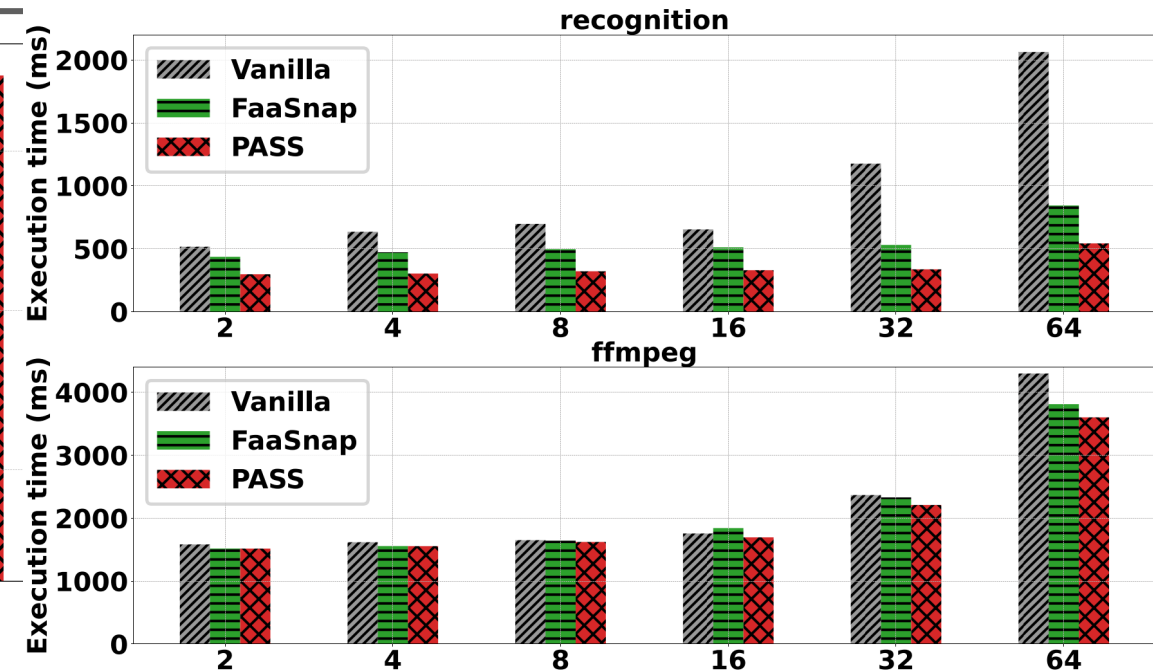
In the "read-list" test,
6.38x(4x) performance increase compared to **Vanilla (FaaSnap)**.

Evaluation: Scalability



SnapStart execution time under a high concurrency.

PASS outperformed Vanilla (by **1.56x to 6.38x**)
and FaaSnap (by **1.38x to 4x**)
in the execution time under the concurrency level 32.



SnapStart execution under different concurrency levels.
(top: "recognition"; bottom: "ffmpeg")

In the "recognition" test,
3.5x(1.6x) performance increase
compared to **Vanilla (FaaSnap)**.

In Conclusion



- Problem:
 - Slow SnapStart for short-lived workloads
- Technology:
 - Pre-fault Page Mapping
 - Zero-copy On-demand Paging
- Achievement:
 - 72% SnapStart time reduction than Vanilla,
 - 47% SnapStart time reduction than FaaSnap.
 - **2x** throughput than Firecracker & FaaSnap
 - **2.2x** cost-effectiveness than Firecracker

Thank you! Q&A