Mnemo: Boosting Memory Cost Efficiency in Hybrid Memory Systems

Thaleia Dimitra Doudali, Ada Gavrilovska
thdoudali@gatech.edu, ada@cc.gatech.edu

1. Motivation

Thaleia created a new web platform that uses a cloud-based in-memory key-value store, in order to accelerate data retrieval. She has a lot of data to host, that are trending frequently, so she needs as much memory capacity as she can afford.

DRAM takes 70% of the total Virtual Machine cost. Her budget allows only for a small amount of DRAM.

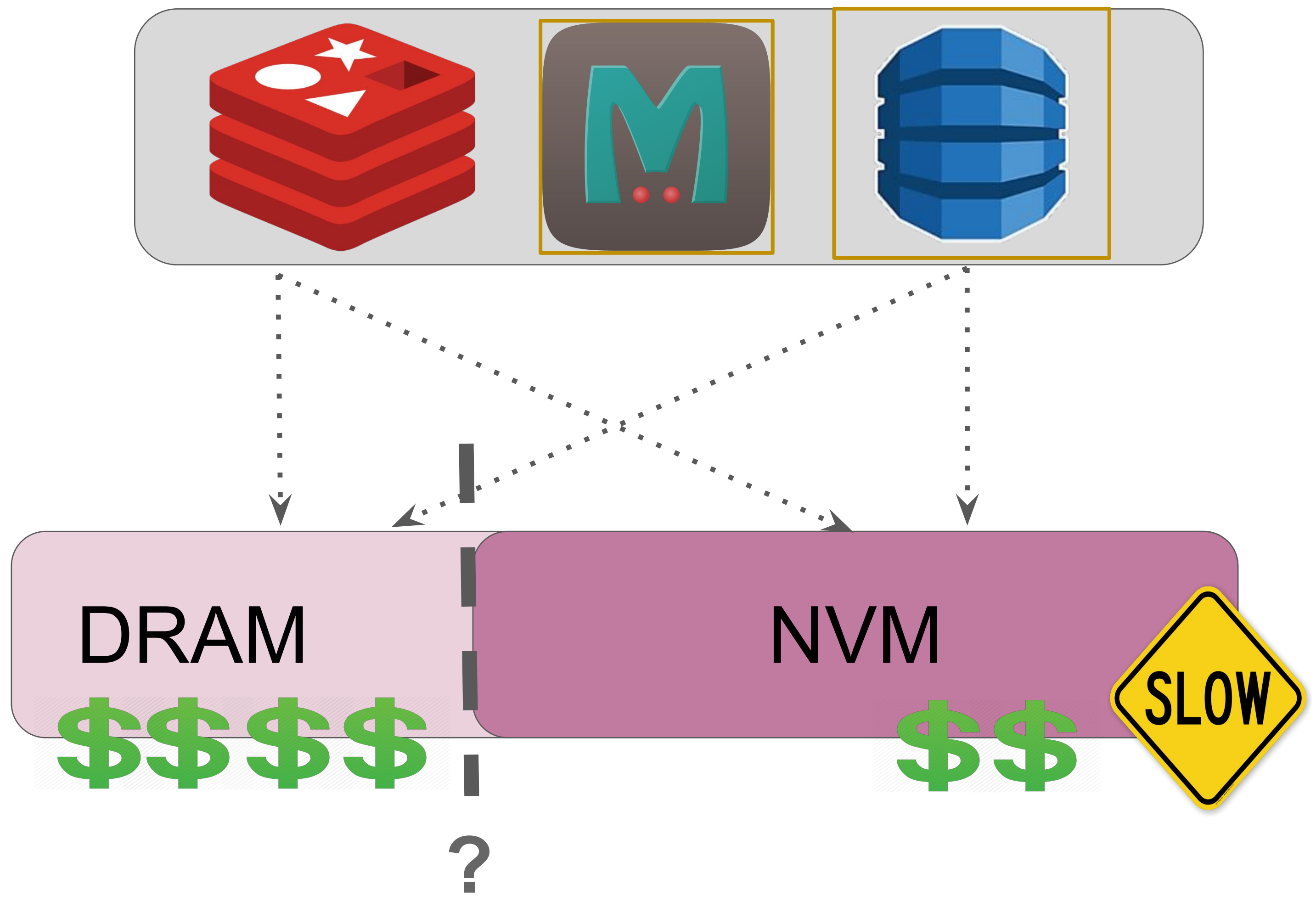
Thaleia learns that Non Volatile memory is cheaper, so she decides to buy both DRAM and NVM, spending around the same amount of money for more capacity.

However, Thaleia has some high priority clients and needs to guarantee certain latency for data retrieval. Introducing NVM to her memory subsystem, will affect and potentially violate these SLA agreements.

How can Thaleia quickly figure out how much DRAM to NVM capacity ratio to use, so as to have the desired performance, while being the most cost-efficient choice?

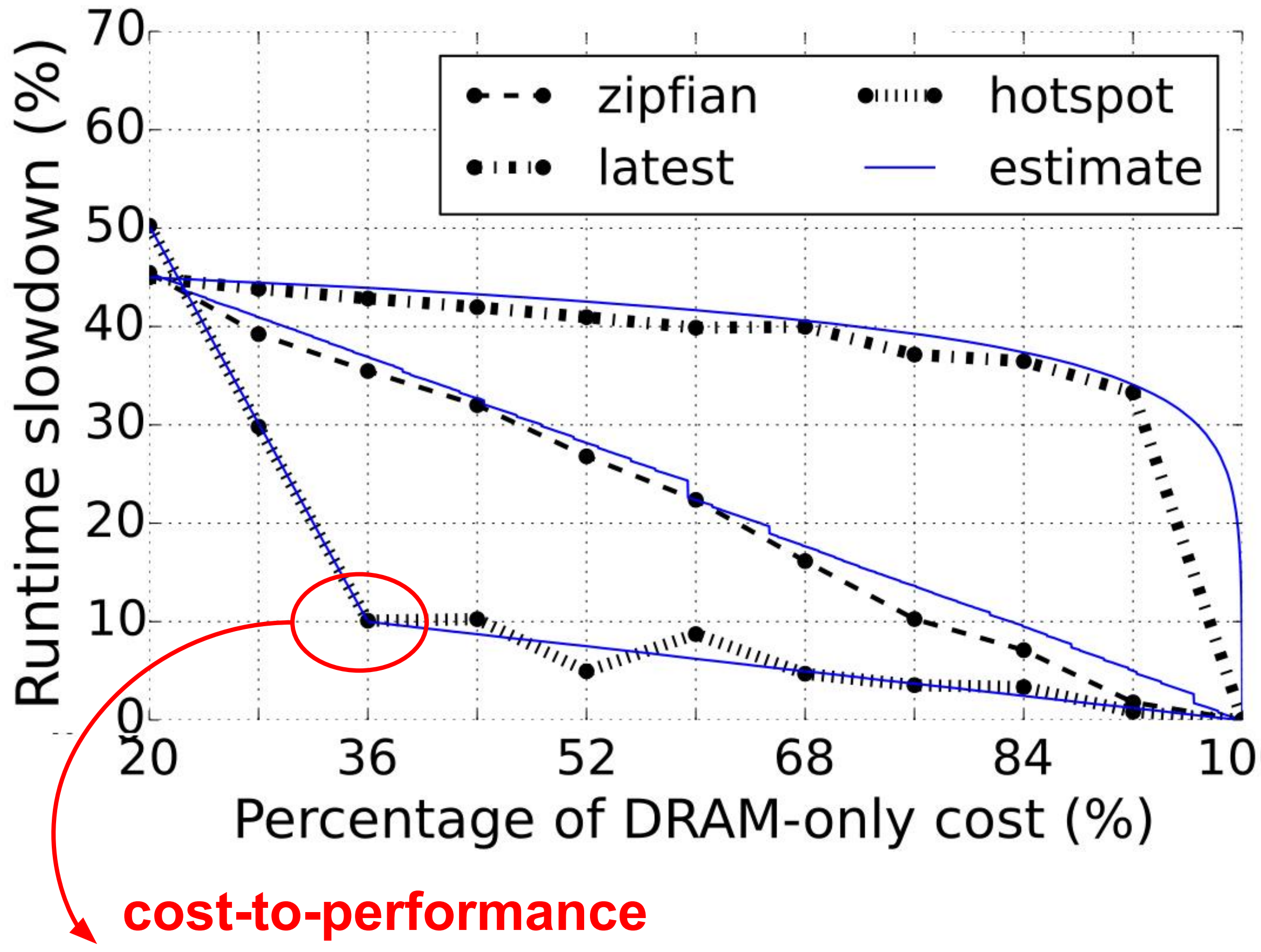
2. Problem Statement

In-memory key-value stores



Problem: How to quickly decide the ratio of DRAM to Non Volatile Memory, that provides the desired application performance together with high cost efficiency?

3. Why is it important?

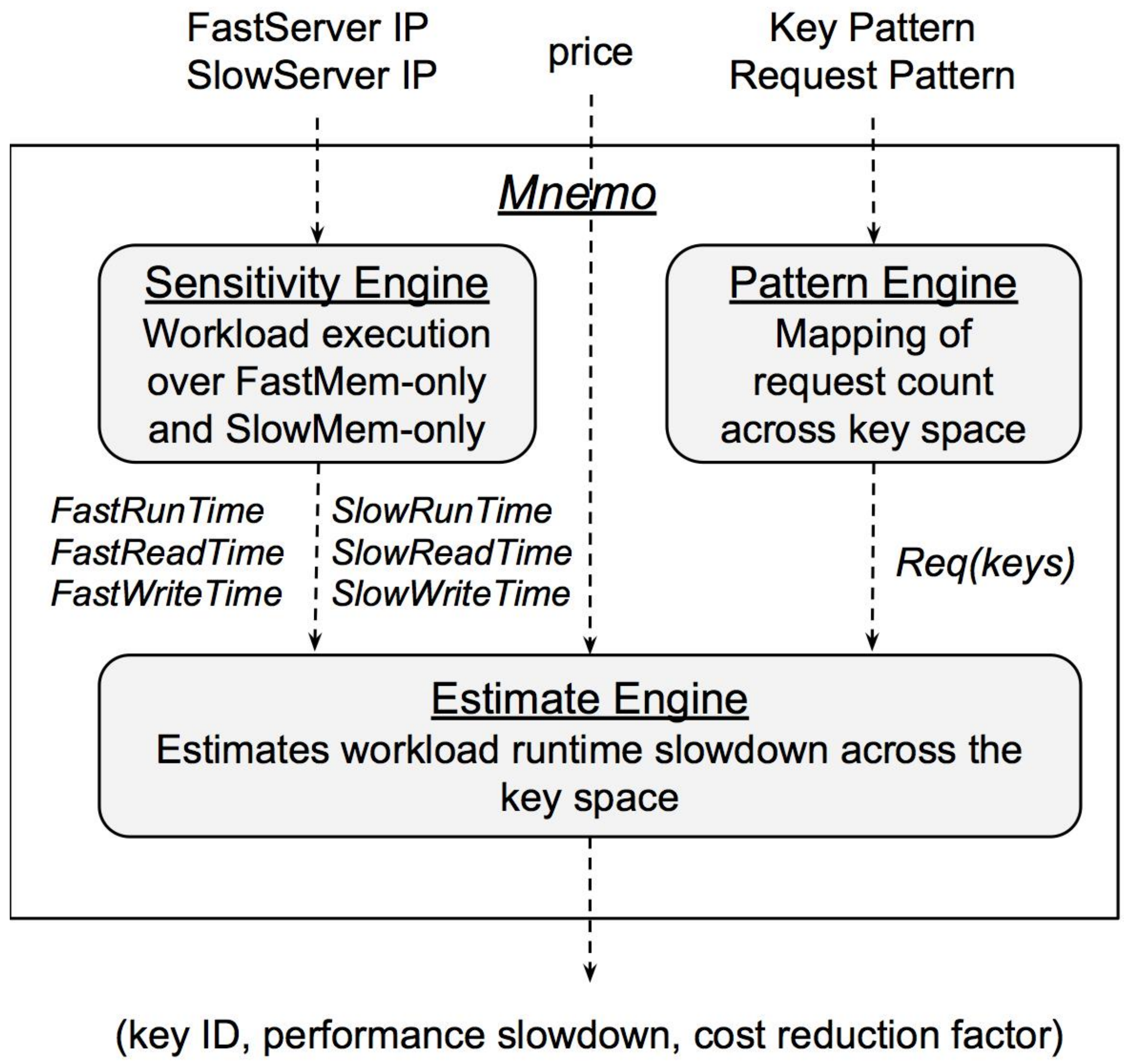


cost-to-performance sweet spot

There are workloads, like hotspot (Trending), where the placement in DRAM of the hottest only keys, increases application runtime by a small and acceptable amount (10%). The cost savings can be huge, assuming that NVM $/byte can be only 20% of the DRAM $/byte.

Solution: Build a profiling tool that estimates the application slowdown for incremental DRAM to NVM capacity ratio, thus cost, on a hybrid memory system.

4. Mnemo Design



FastServer IP
SlowServer IP

price

Key Pattern
Request Pattern

Mnemo

Sensitivity Engine
Workload execution over FastMem-only and SlowMem-only

Pattern Engine
Mapping of request count across key space

FastRunTime
FastReadTime
FastWriteTime

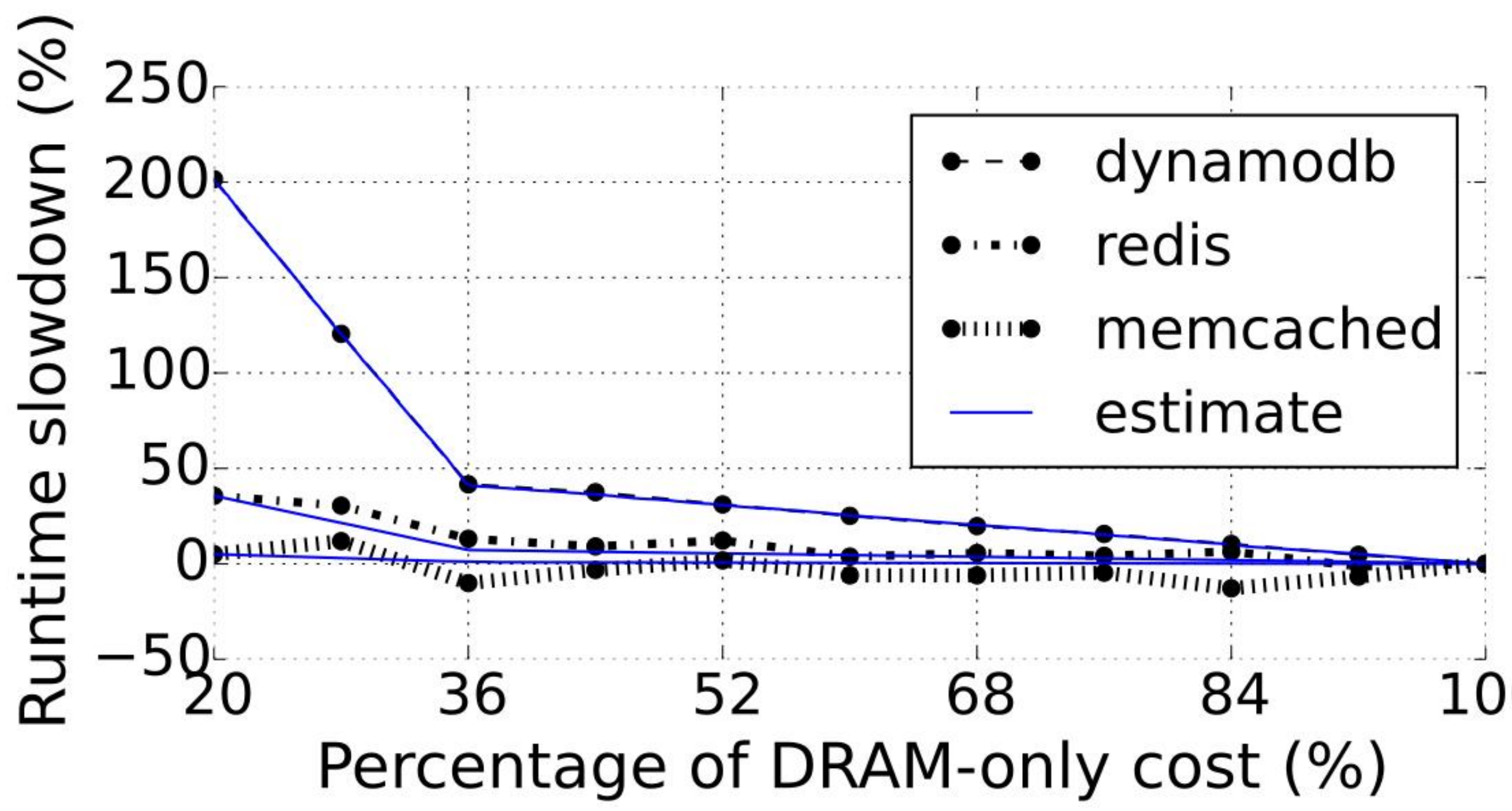
SlowRunTime
SlowReadTime
SlowWriteTime

Req(keys)

Estimate Engine
Estimates workload runtime slowdown across the key space

(key ID, performance slowdown, cost reduction factor)

5. Results



Mnemo accurately produces performance slowdown estimates for incremental DRAM to NVM capacity ratio (left to right), across the three top-ranked in-memory key-value stores.
Users of Mnemo can then extract the cost-to-performance configuration, that suits their budget and performance guarantees.

6. Highlights

High Accuracy: Mnemo uses a simple yet extremely accurate model to estimate the performance degradation. (0.75% median error)

Trivial Overhead: Mnemo's pattern analysis and estimation model run instantaneously. The overhead is the time to get the performance baselines.

Robustness to Downsampling: Mnemo's estimate is accurate even for downsized versions of workloads, that retain their request access pattern.

All work is part of the projects:
ECP SICM (Software Interface to Complex Memories)
SSIO Unity (Unified Memory and Storage Space)