Design of a Key-Value SSD for All Workload Types

Wonil Choi
Hanyang University

NVRAMOS 2025

Introduction

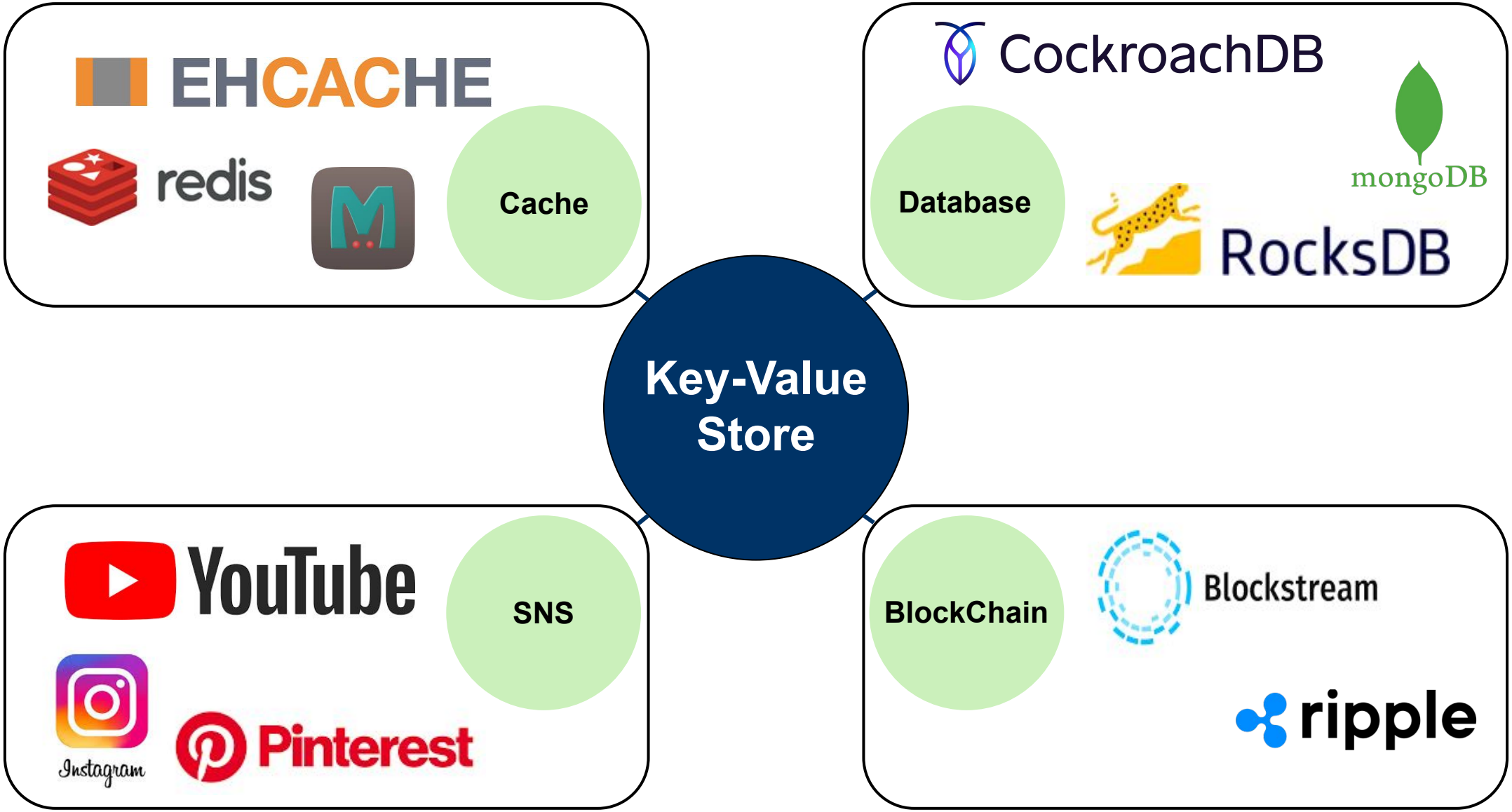
Motivation

Proposed Design

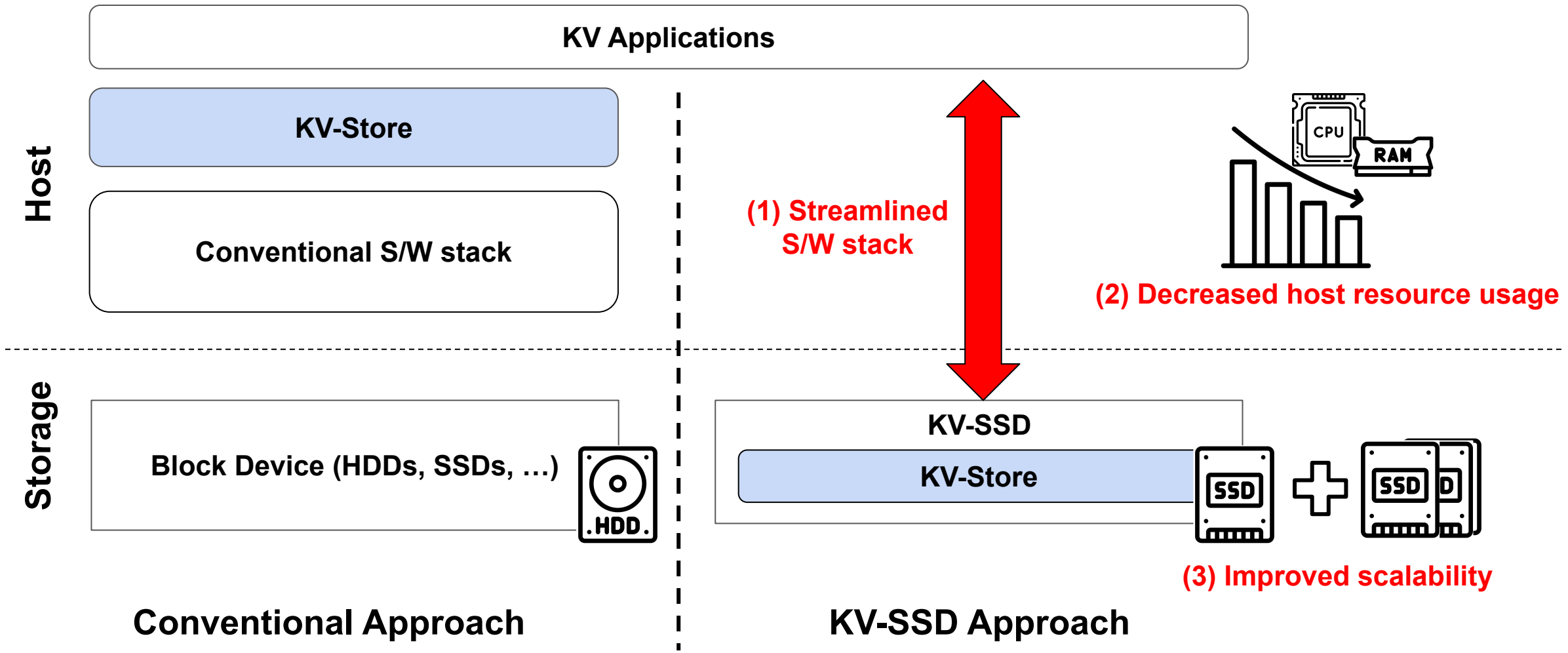
Evaluation

Conclusions

Explosive Spread of KV Store-Based Applications/Services

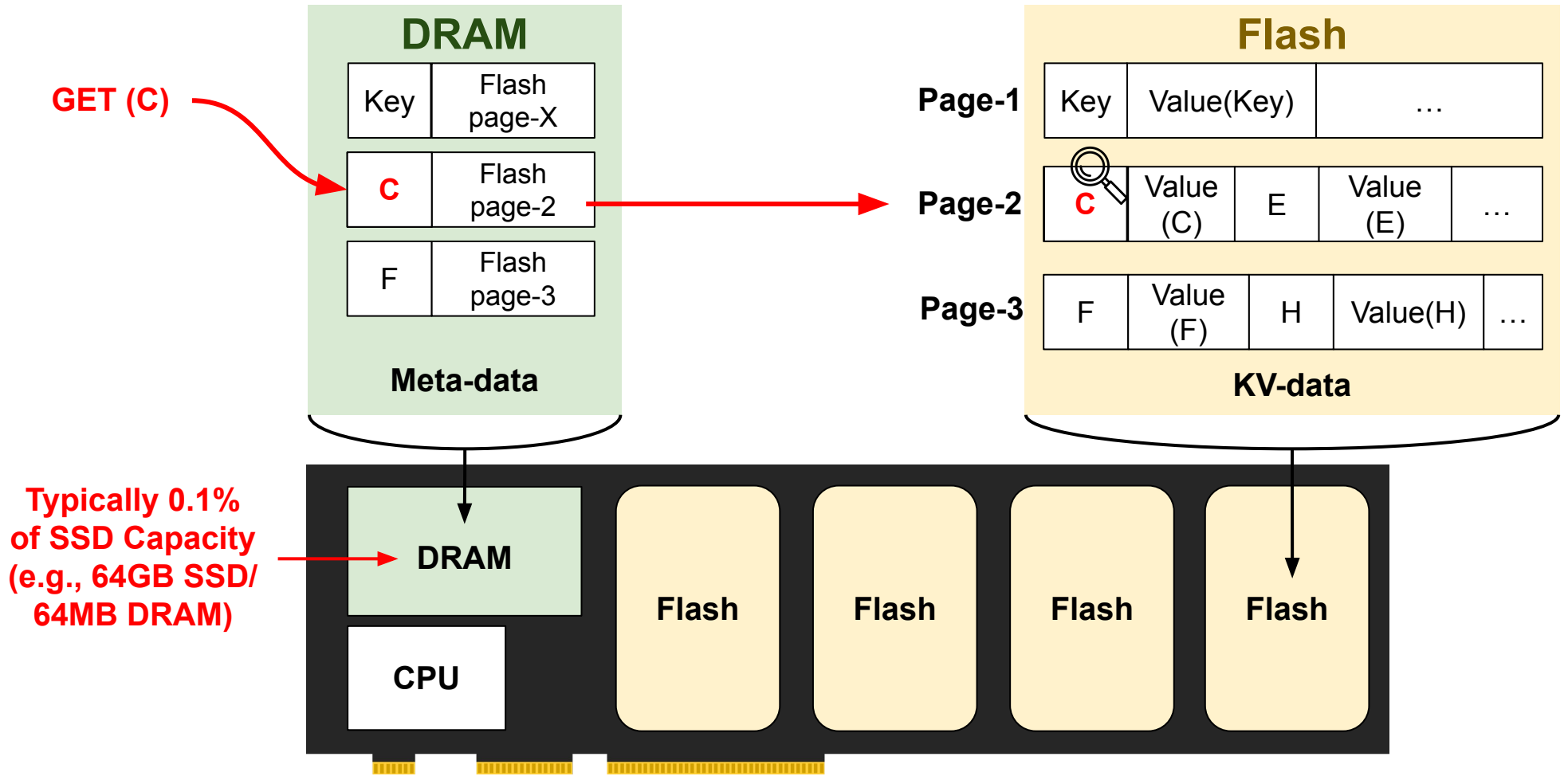


KV-SSD: A Potential Approach for Executing KV Stores



Our goal is to propose a high-performance KV-SSD design

Architecture of a KV-SSD



KV-data are stored in flash while meta-data indicating the KV-data are maintained in DRAM

Introduction

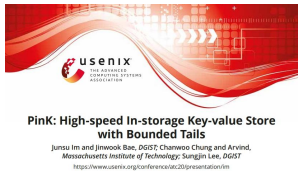
Motivation

Proposed Design

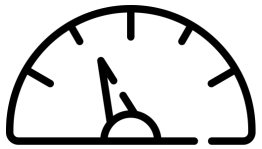
Evaluation

Conclusions

1. Existence of Unexplored Workload Types

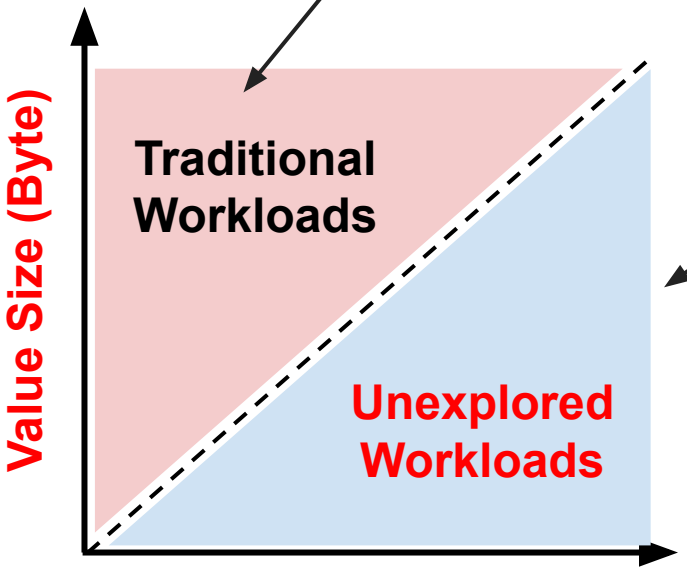


W-PinK
key (32B)
value (1024B)



YCSB
key (20B)
value (1000B)

Samsung
key (16B)
value (4096B)



Xbox

Xbox LIVE primetime online game
key (94B) value (1200B)

UDB

Facebook storage layer for social graph
key (27B) value (127B)

Cache

Twitter's cache cloud
key (42B) value (188B)

ZippyDB

Object metadata of Facebook store
key (48B) value (43B)

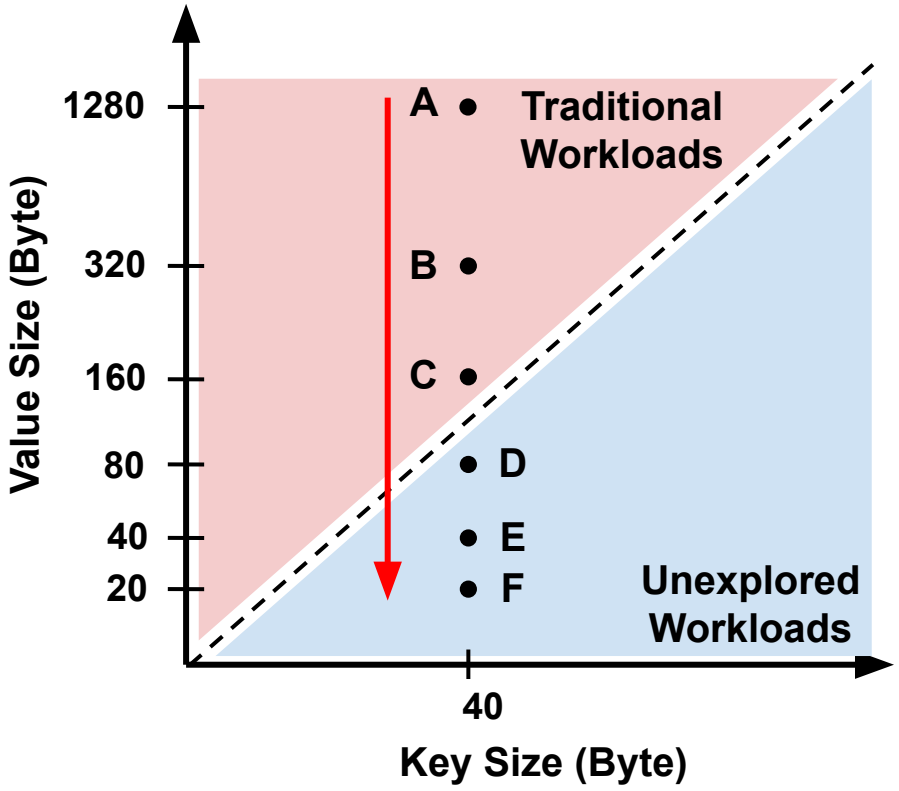
Crypto1

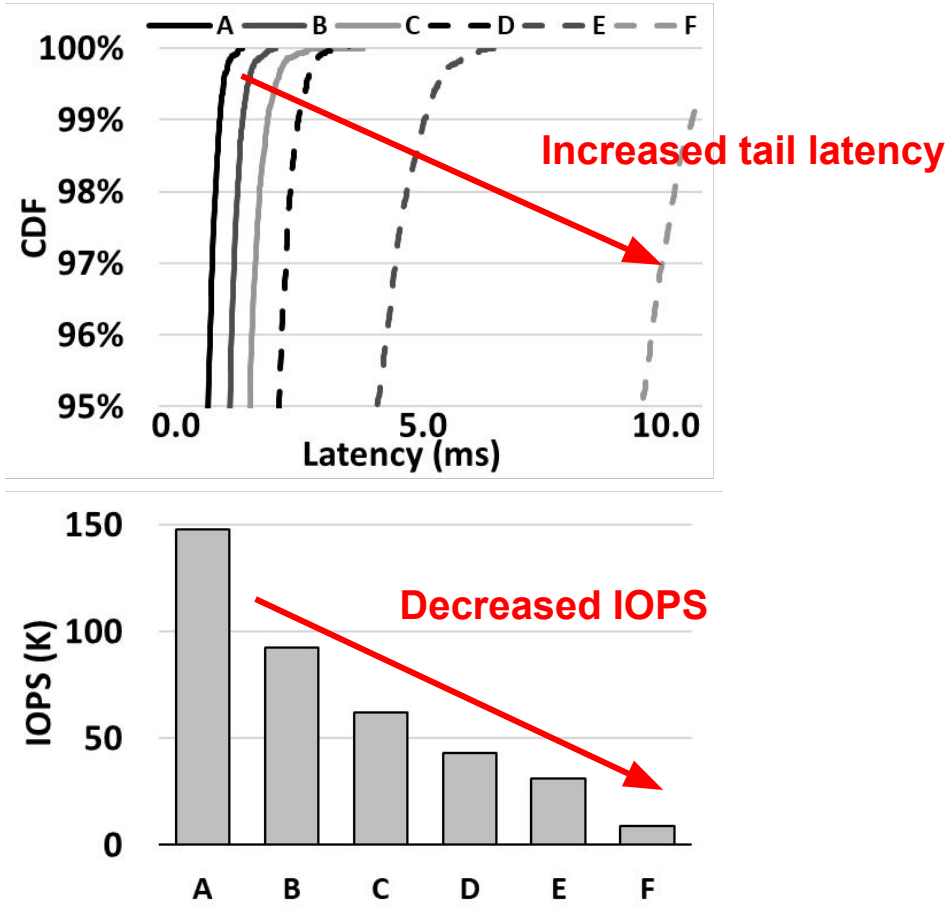
BlockStream's store for Bitcoin explorer
key (76B) value (50B)

In fact, we observed that there are many unexplored workloads where the key sizes are relatively large

2. Poor Performance under Unexplored Workloads

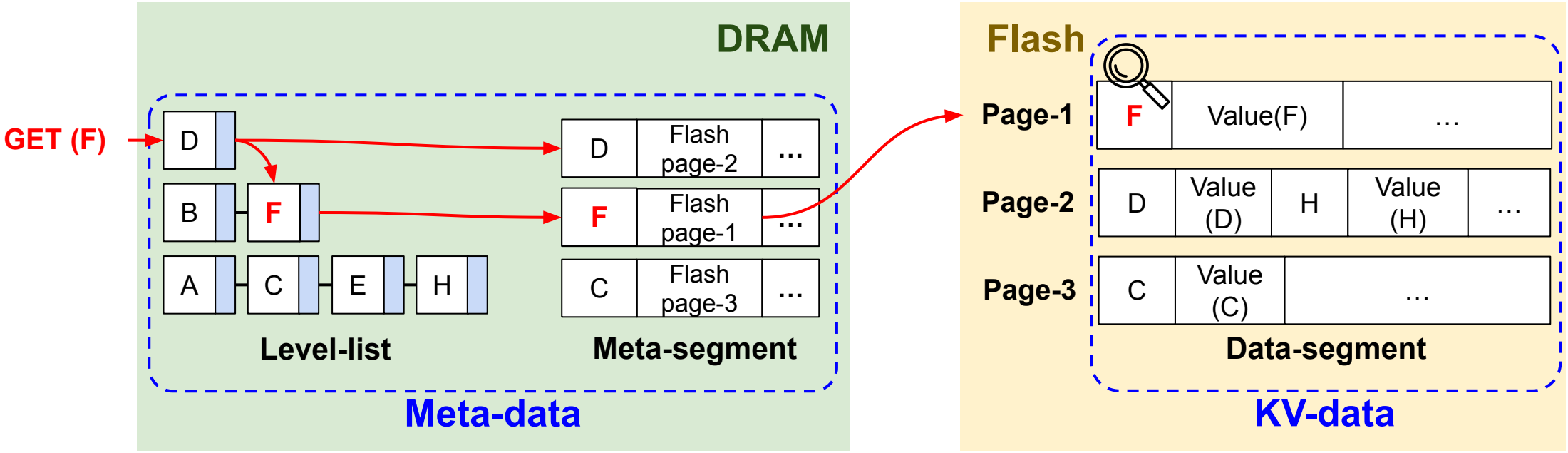
We re-evaluated current KV-SSD designs using the unexplored workloads

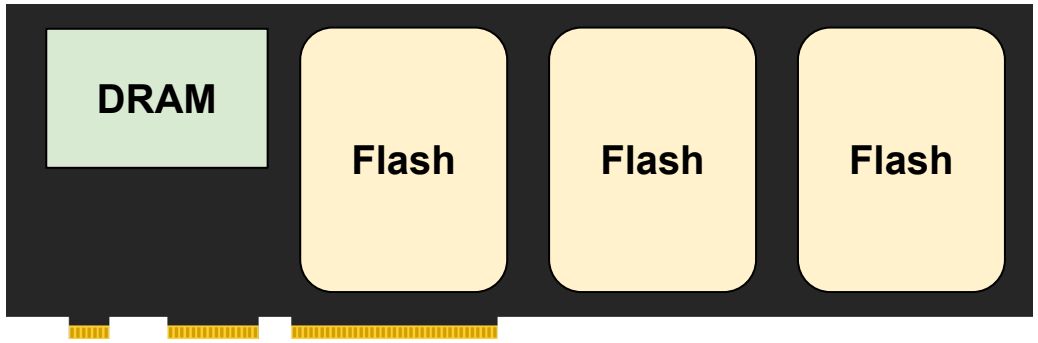




Overall, the state-of-the-art design performs poorly under unexplored workloads

State-of-The-Art KV-SSD Design [USENIX ATC'20]

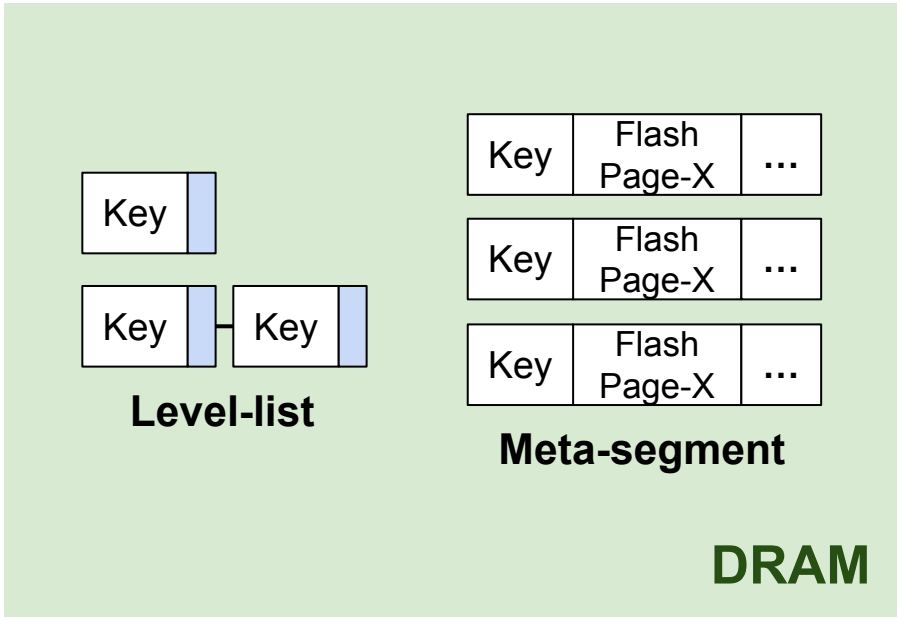




The mainstream KV-SSD design uses a log-structured merge (LSM)-tree to manage meta-data
Level-lists and meta-segments are the two meta-data types

Analysis on Meta-Data Sizes & Their Locations

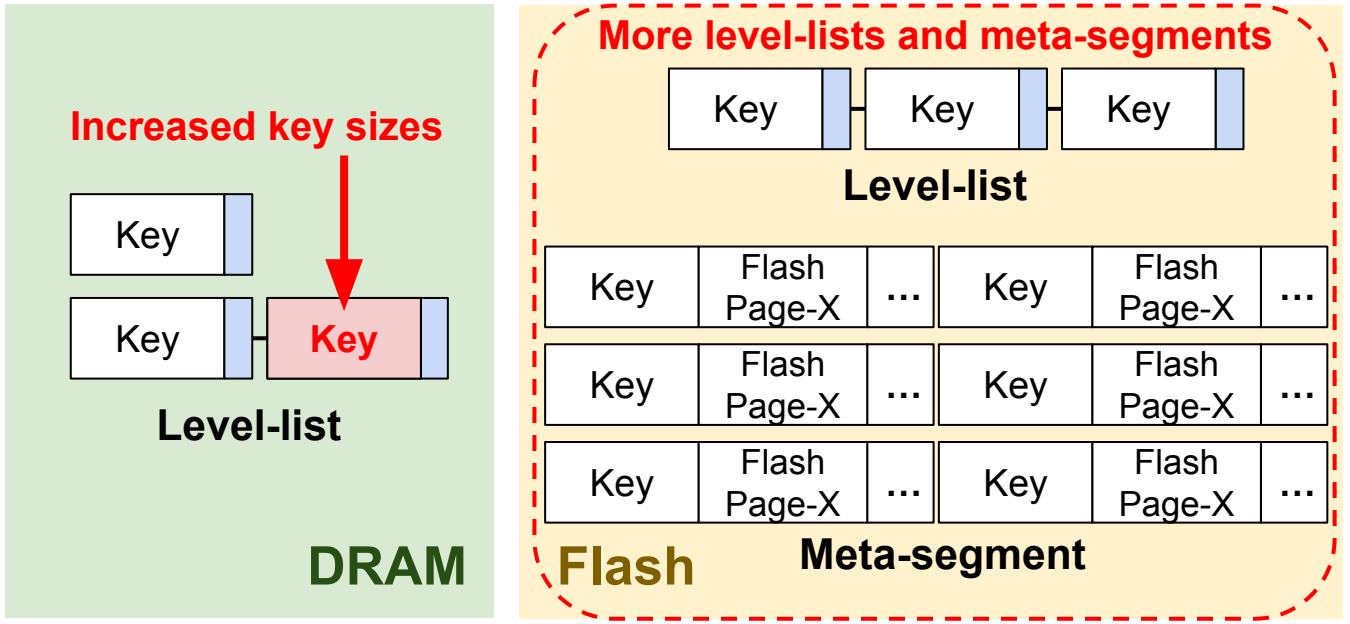
Under traditional workloads



64GB Flash / 64MB DRAM assumed

Workload	Level List Size (MB)	Meta Segment Size (MB)	Sum (MB)	Stored in Flash (MB)
YCSB	4.65	78.48	83.13	20.13
Samsung	0.84	25.42	26.26	0

Under unexplored workloads

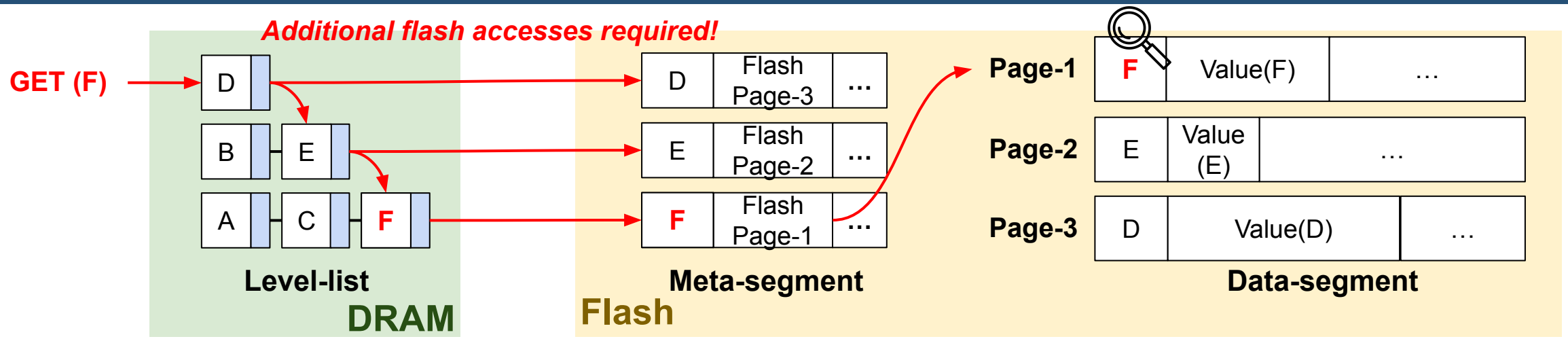


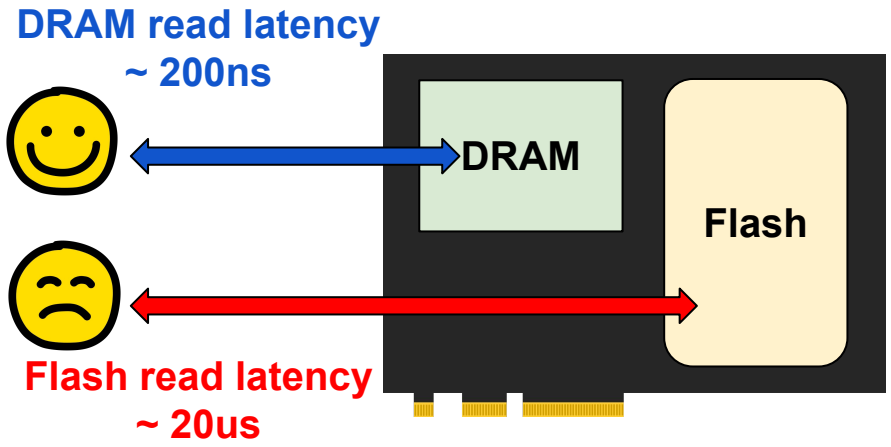
64GB Flash / 64MB DRAM assumed

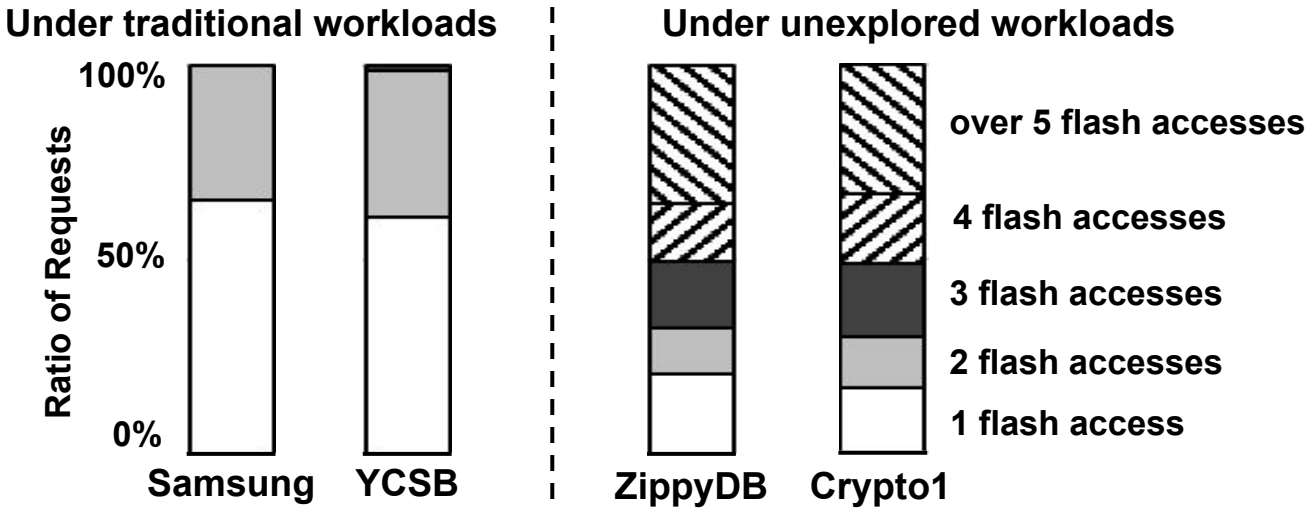
Workload	Level List Size (MB)	Meta Segment Size (MB)	Sum (MB)	Stored in Flash (MB)
Crypto1	331.4	759.97	1091.37	1028.37
ZippyDB	188.89	689.17	878.06	815.06

A considerable amount of meta-data is also stored in the flash

When Meta-Data Is Stored in Flash







We observed that several flash accesses occurred during a single read under unexplored workloads

Introduction

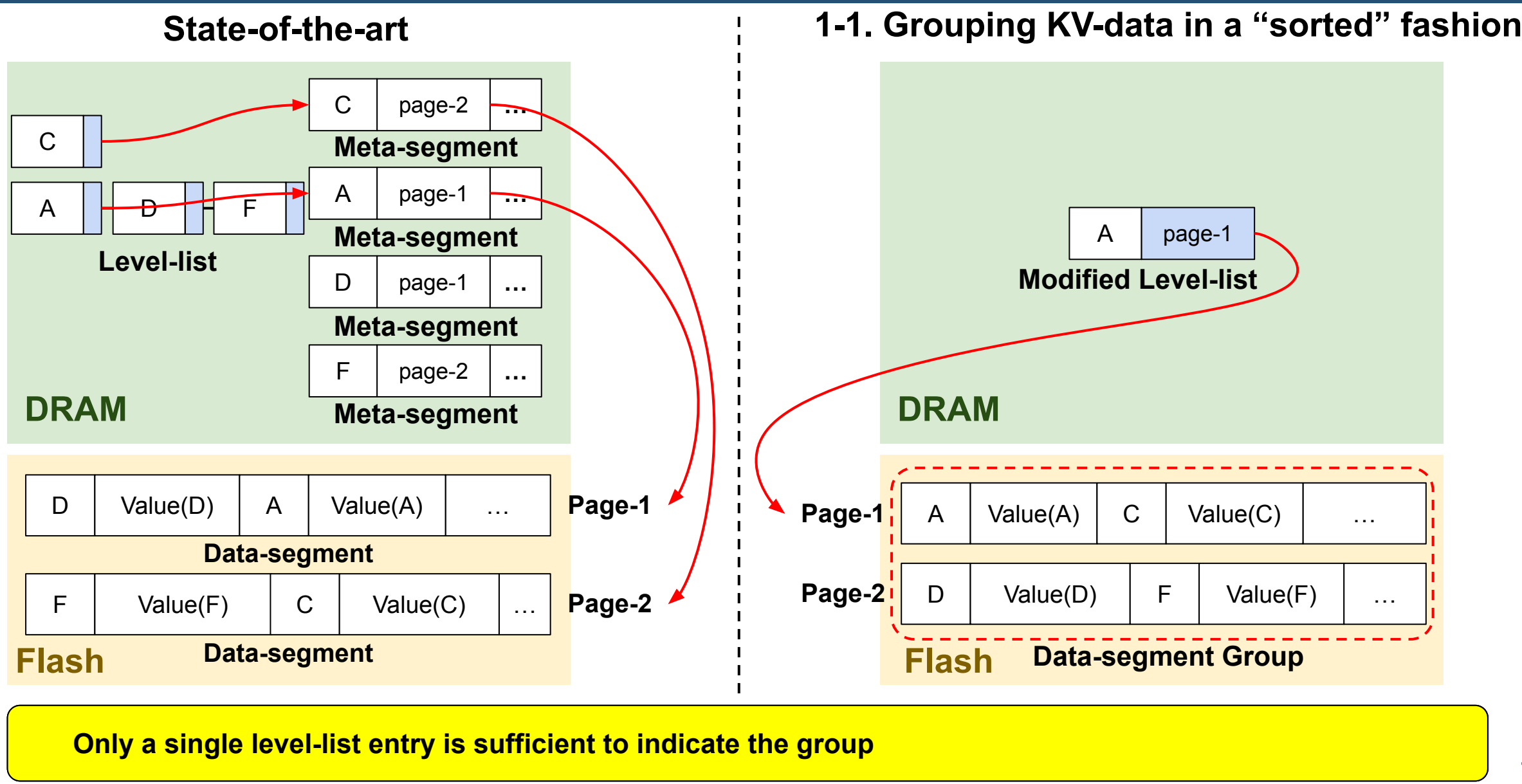
Motivation

Proposed Design

Evaluation

Conclusions

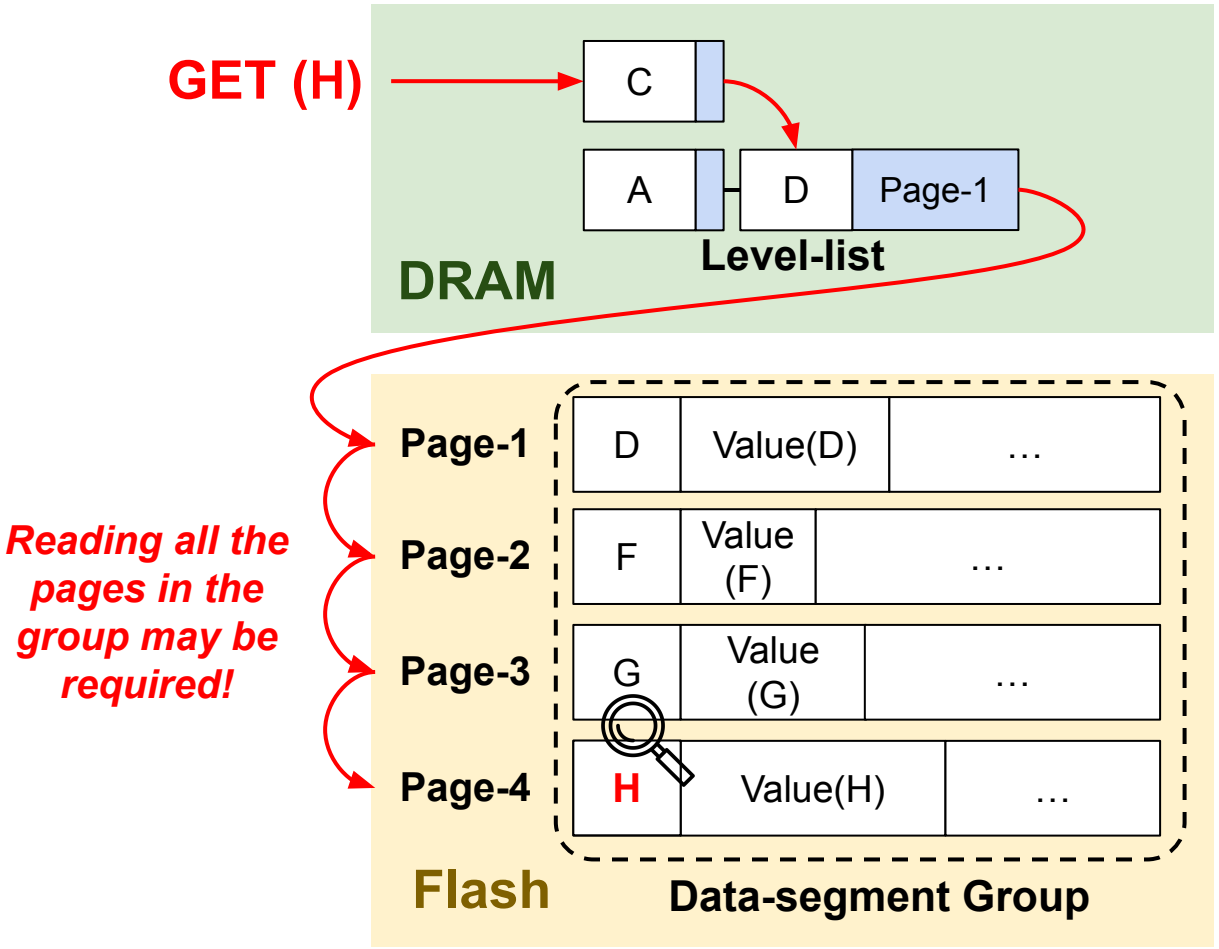
Design Principle 1: Minimizing Meta-Data Sizes



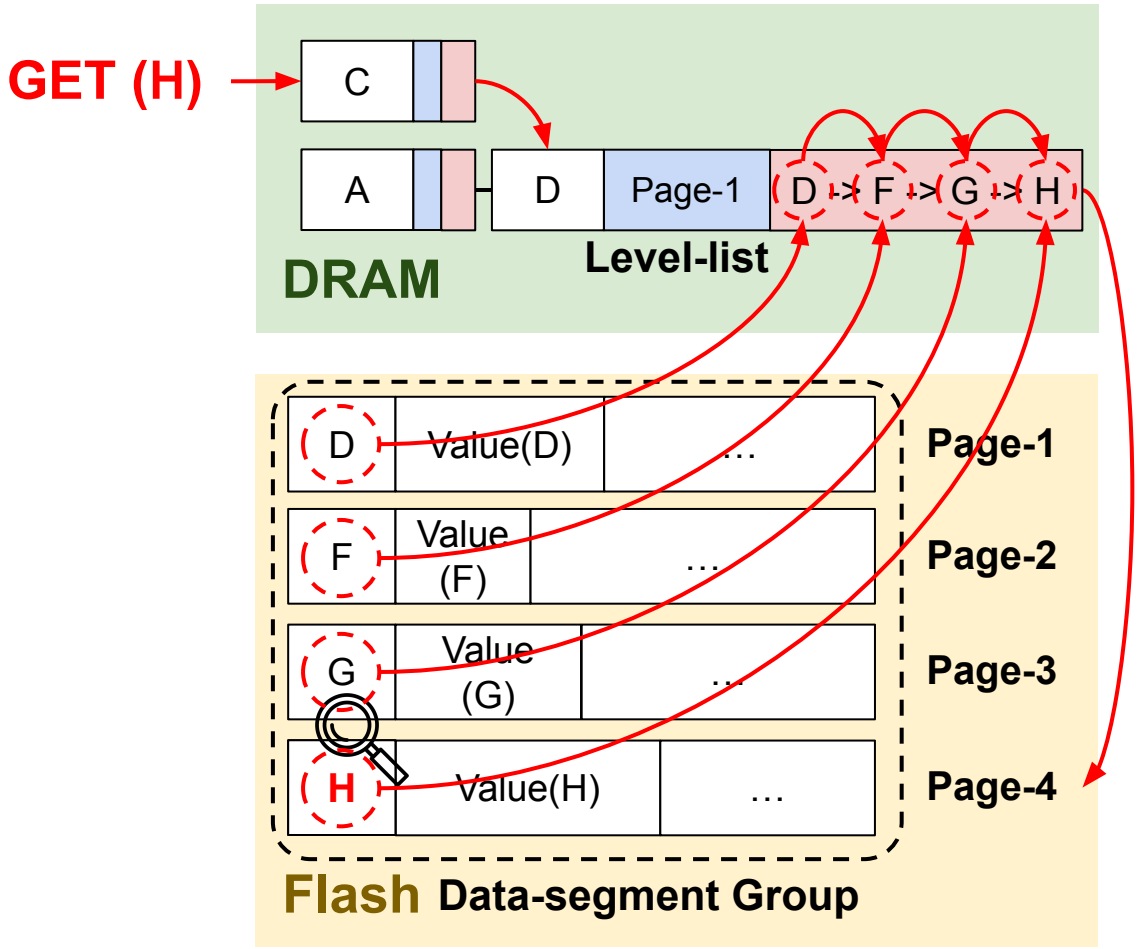
Only a single level-list entry is sufficient to indicate the group

Design Principle 1: Minimizing Meta-Data Sizes

1-1. Grouping KV-data in a "sorted" fashion



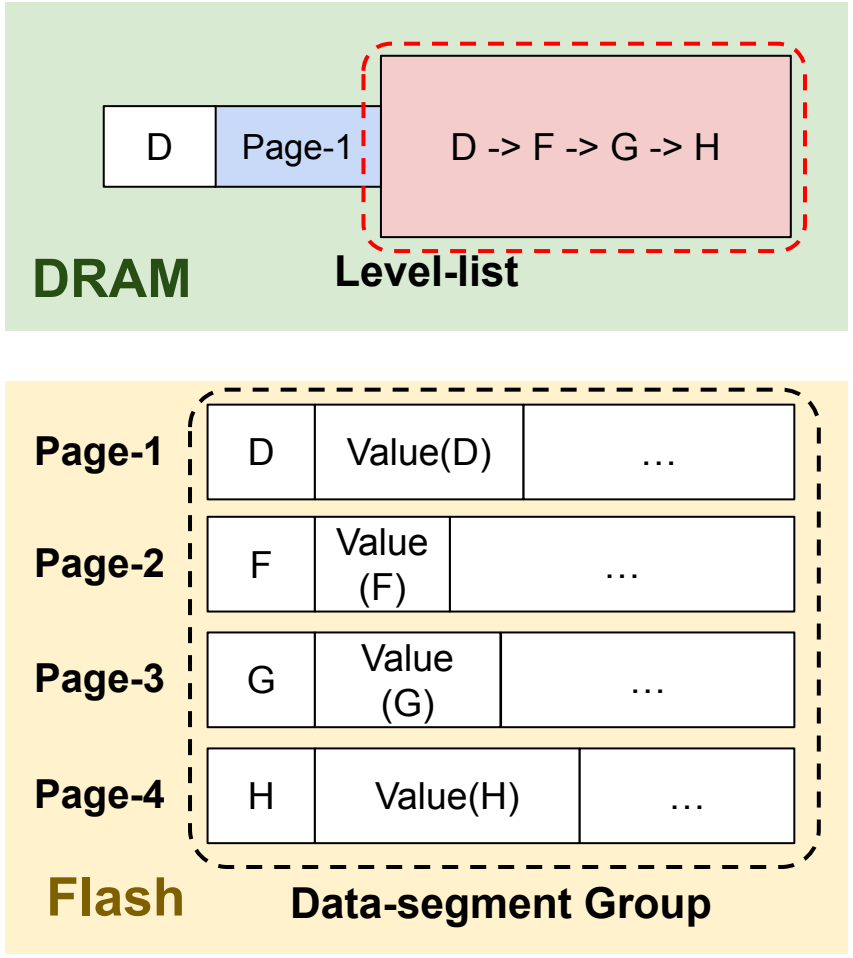
1-2. Maintaining the 1st key of each page



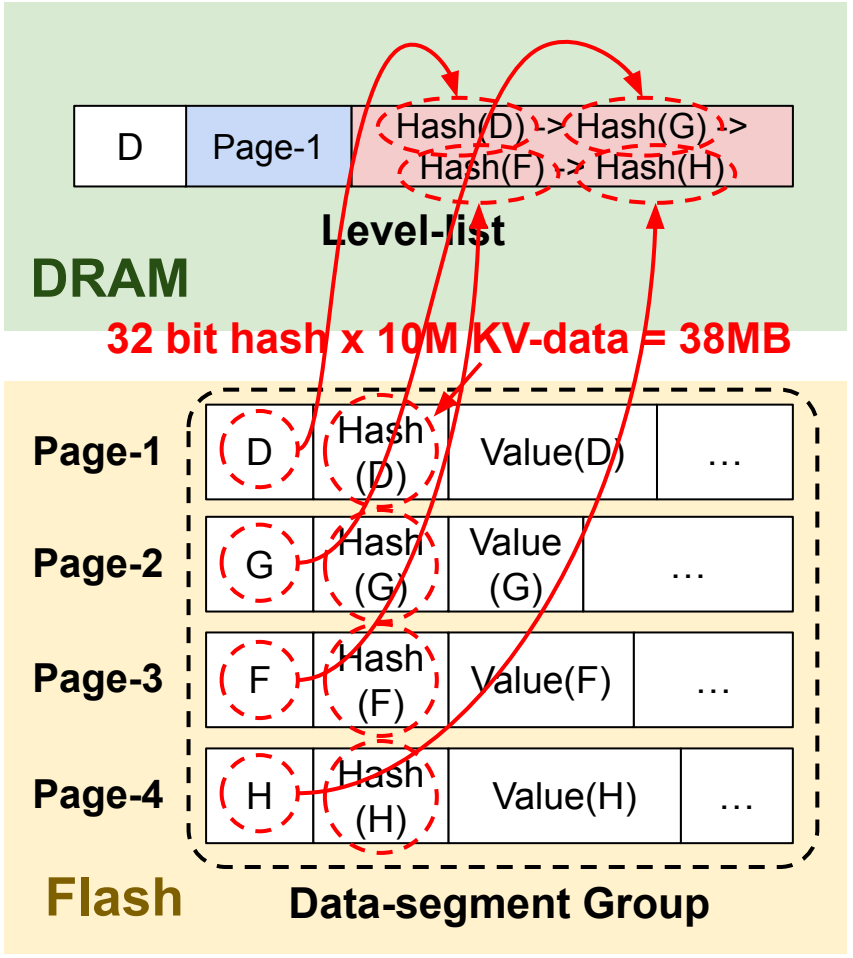
We maintain the first keys of all the pages in the level-list entry

Design Principle 1: Minimizing Meta-Data Sizes

1-2. Maintaining the first key of each page

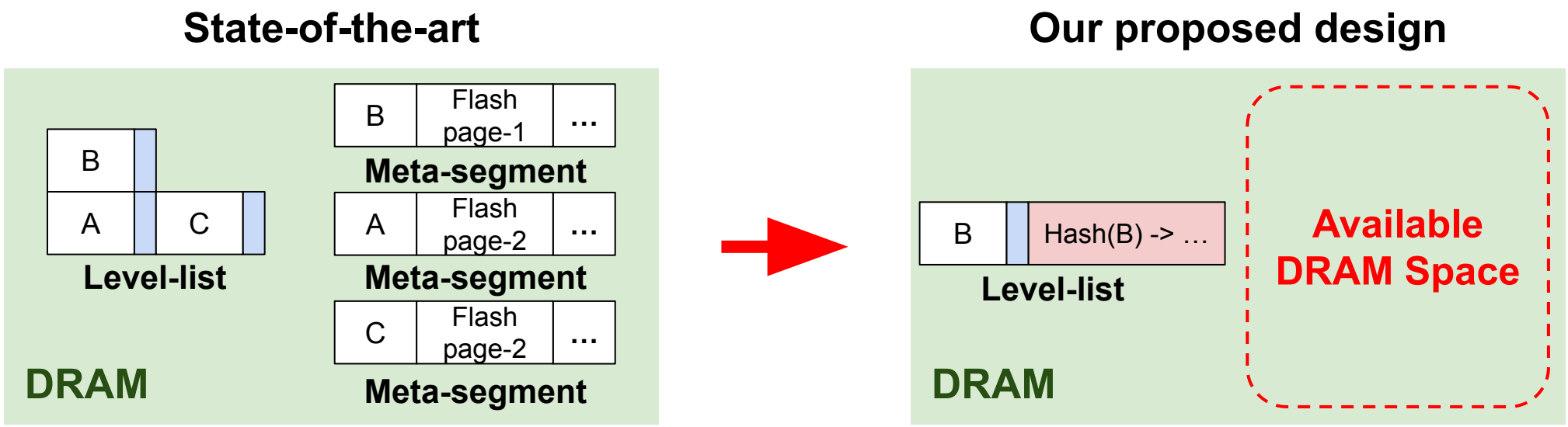


1-3. Using small-sized hash value



The KV-data in a group are sorted based the hash values of the keys

Design Principle 2: Exploiting Available DRAM Space



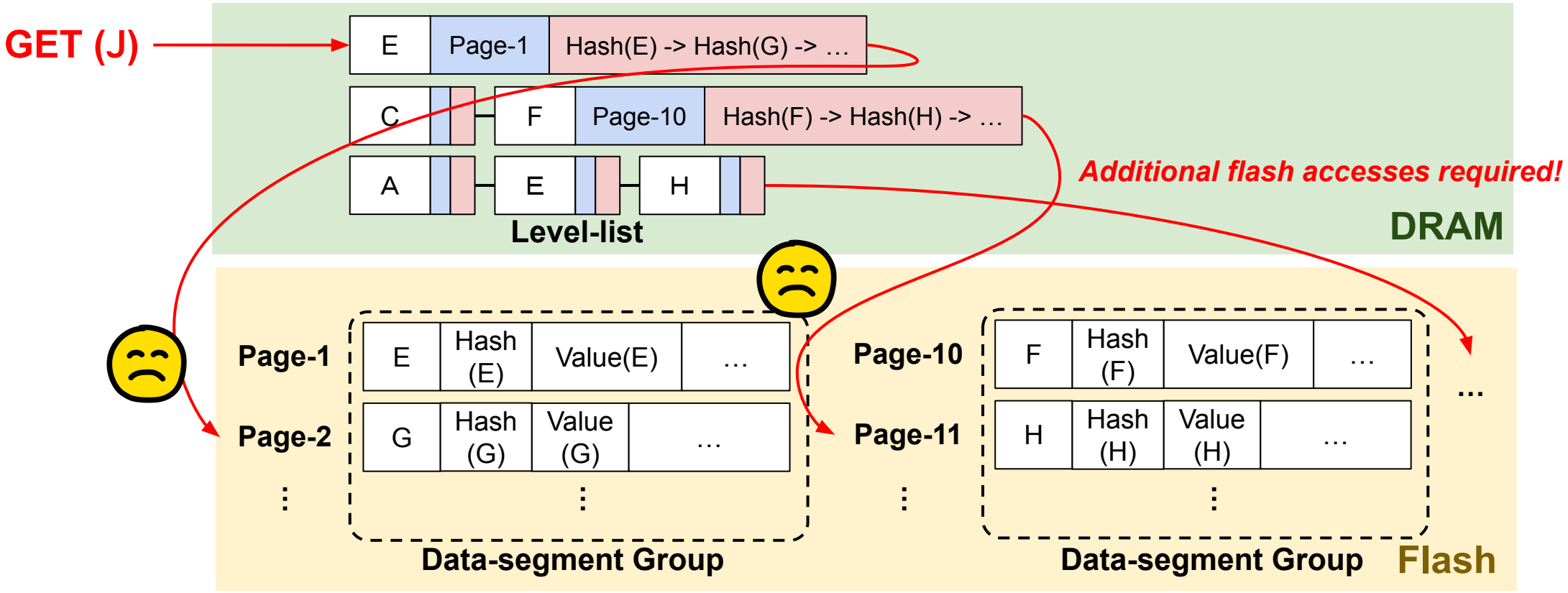
64GB Flash / 64MB DRAM assumed

Workloads Key(B) / Value(B)	Level List Size (MB)	DRAM size (MB)	Available DRAM size (MB)
80B / 80B	38	64	26
60B / 120B	34	64	30
40B / 160B	29	64	35

This available space can be sometimes significant

Design Principle 2: Exploiting Available DRAM Space

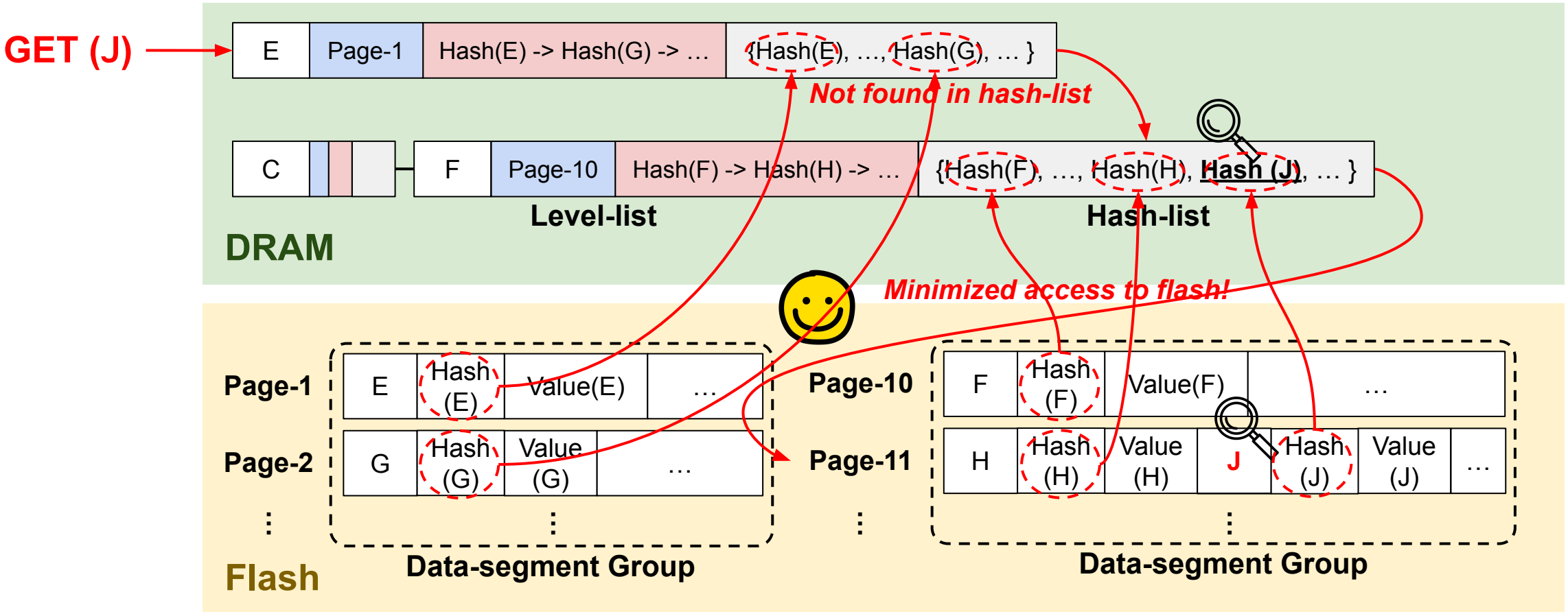
Even if target key falls into a level-list entry, the corresponding group may not hold the target KV-data



Since level list maintains key ranges, we cannot guarantee that the target KV-data is indeed within the group
This situation can increase the number of flash accesses

Design Principle 2: Exploiting Available DRAM Space

We add "the hash values of all keys in the group" to the level-list entry



Using the hash list, we can verify if the identified group truly contains the target KV-data without accessing flash

Introduction

Motivation

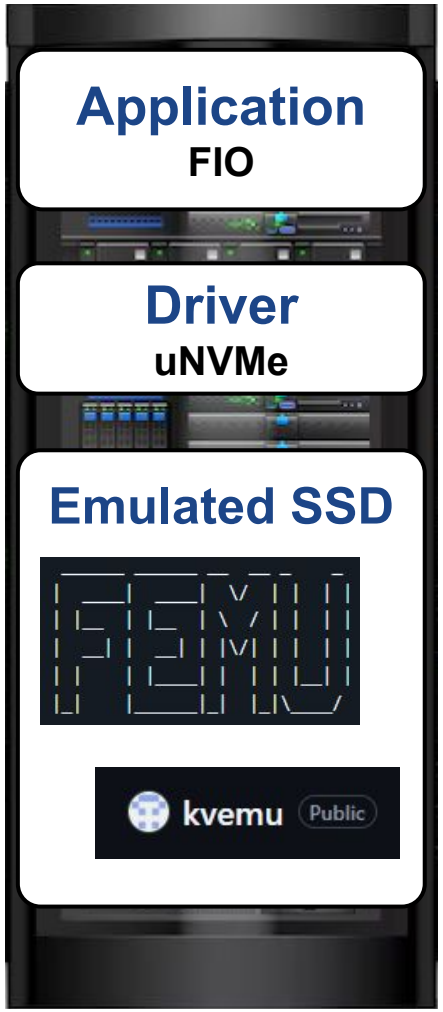
Proposed Design

Evaluation

Conclusions

Evaluation Setup

Testbed



Traditional workloads

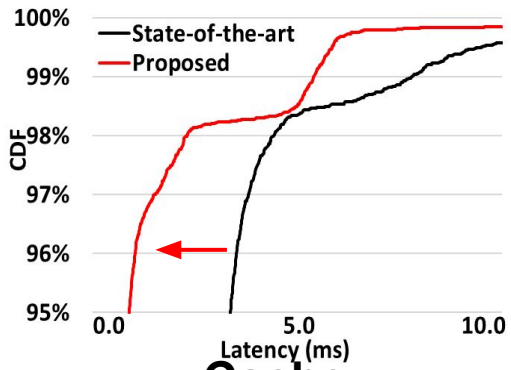
Workload Name	Key Size (Byte)	Value Size (Byte)
Samsung	16	4096
YCSB	20	1000
W-PinK	32	1024

Unexplored workloads

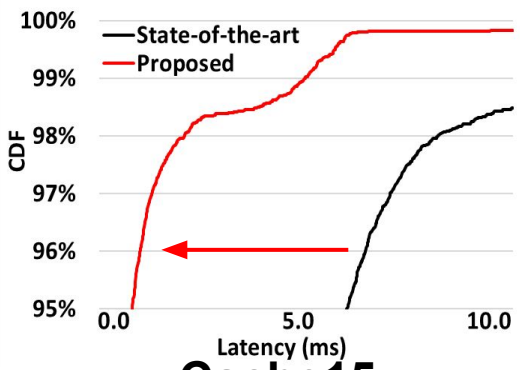
Workload Name	Key Size (Byte)	Value Size (Byte)
Xbox	94	1200
ETC	41	358
UDB	27	127
Cache	42	188
VAR	35	115
Crypto2	37	110
Dedup	20	44
Cache15	38	38
ZippyDB	48	43
Crypto1	76	50
RTDATA	24	10

Tail Latencies

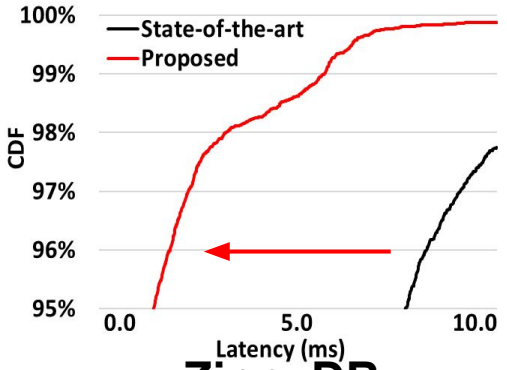
Unexplored workloads



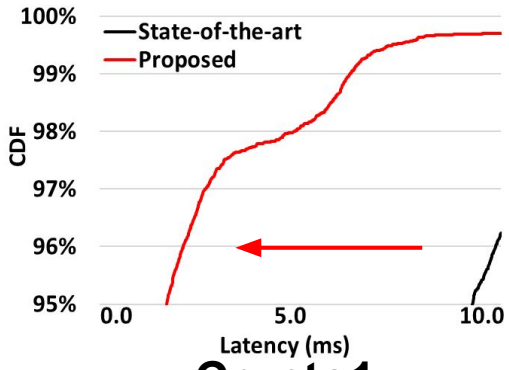
Cache



Cache15



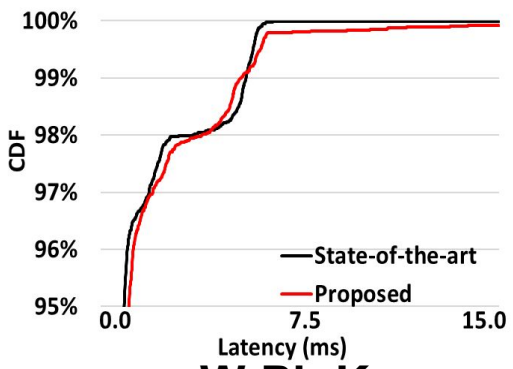
ZippyDB



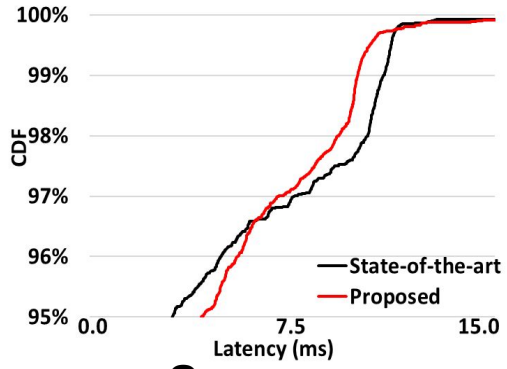
Crypto1

The state-of-the-art exhibits long tail latencies, while our proposed design reduces them significantly

Traditional workloads



W-PinK

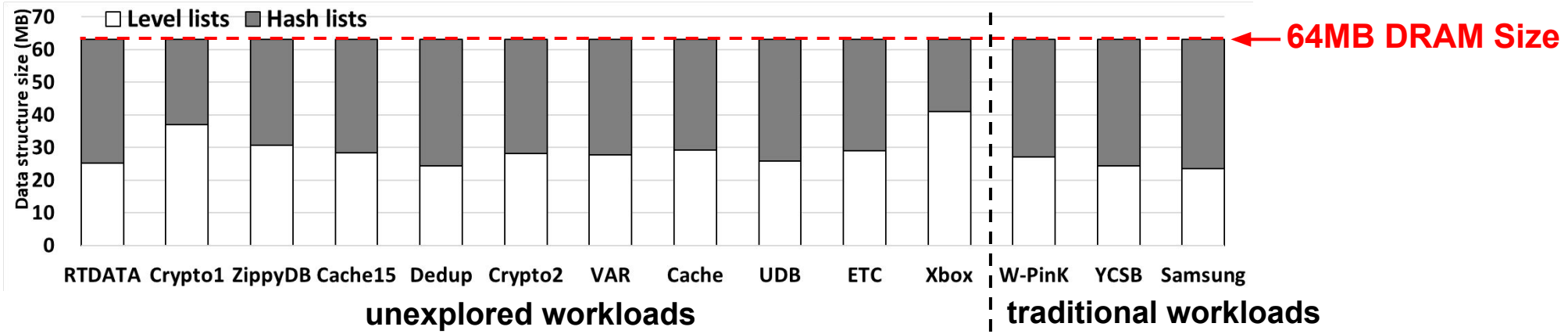


Samsung

Our proposed design also exhibits short latencies comparable to the state-of-the-art

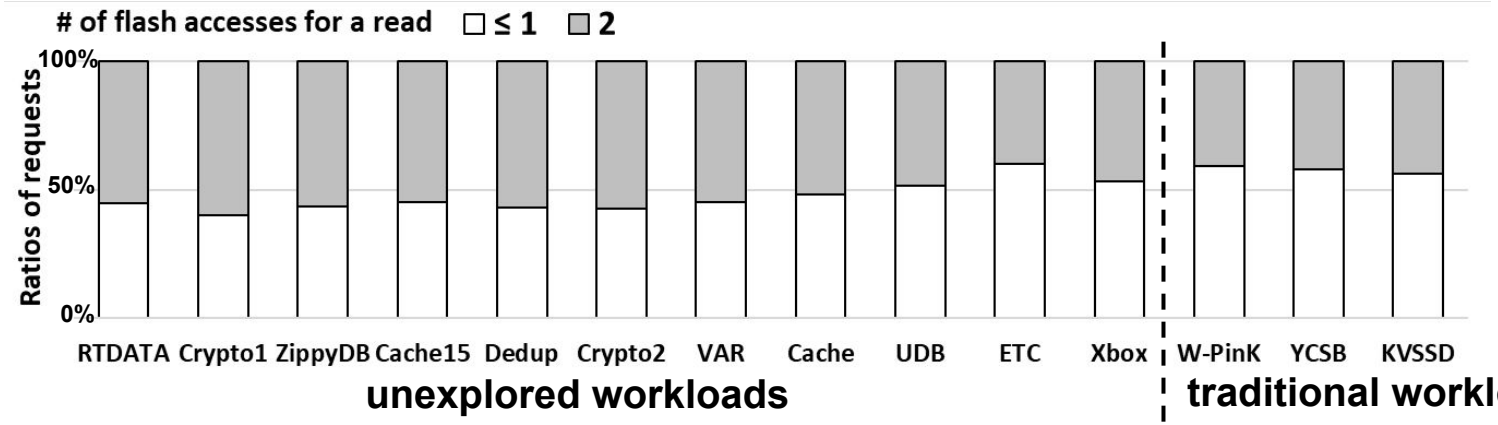
Analysis on Meta-data Sizes

Meta-data sizes and their locations



In our proposed design, all metadata (including level lists and hash lists) can be accommodated in the DRAM

Flash access count for read operations



The number of flash accesses for read operations can be limited to at most 2

Conclusions

KV-SSDs are a potential storage solution for KV-store applications

Motivation:

- **There exist another type of workload where the key sizes are relatively large**
- **Current KV-SSD designs experience significantly-degraded performance under such unexplored workloads**
- **Our analysis reveals that the performance problem originates from the increased size of meta-data**

Proposed KV-SSD design:

- **Minimizes the size of meta-data under the unexplored workloads**
- **Further reduces flash accesses using a new meta-data type**
- **Outperforms the state-of-the-art**