Flash Memory Summit

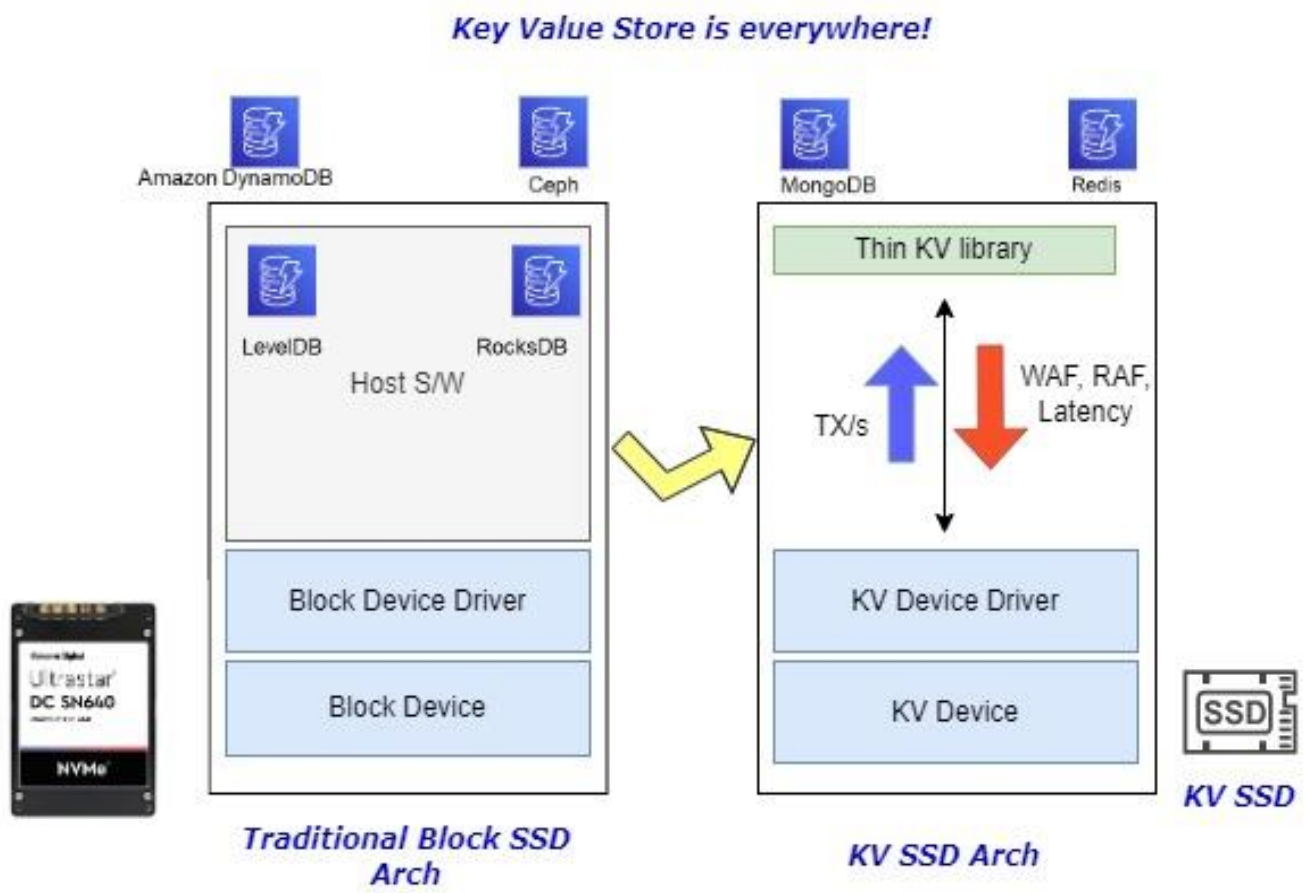
Challenges In Designing KV SSD

Vishwas Saxena

Senior Technologist, Firmware Engineering, Western Digital

KV Stores Are Ubiquitous In Unstructured Data Storage

- Web Indexing, Caching – Twitter, Facebook, LinkedIn
- Media – Images, Videos
- Document Stores
- Blockchain Data

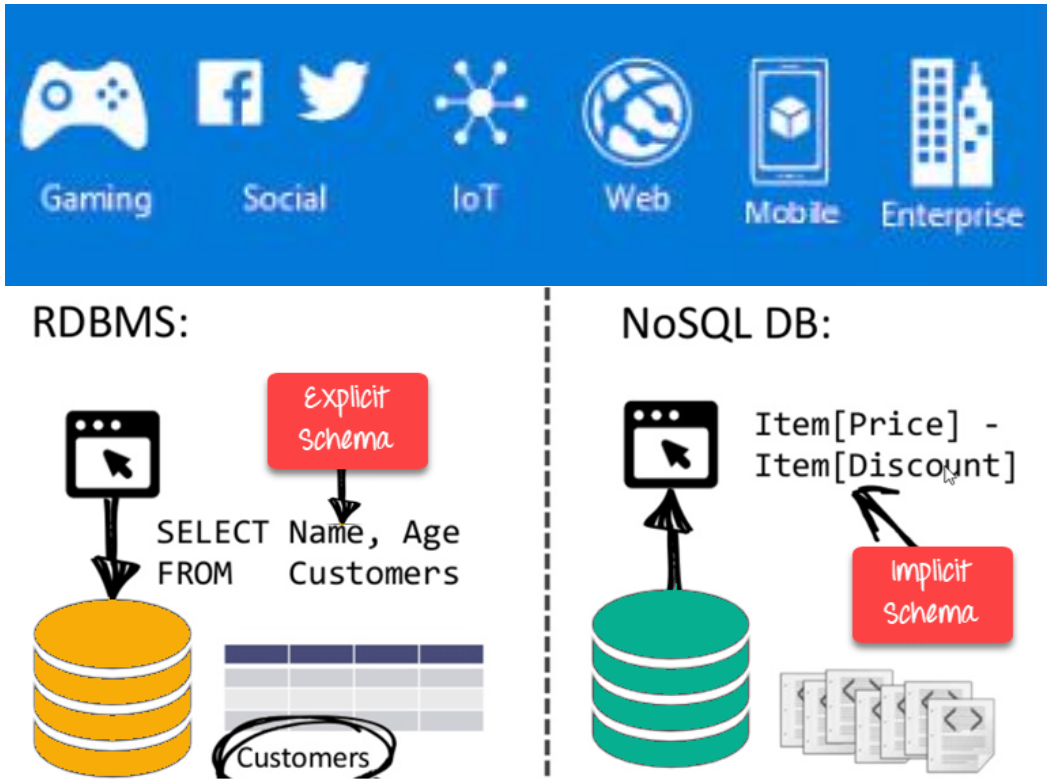


How Are They Useful ?

Nature of Big Data

- Big data is all unstructured and access is heavy on writes and light on reads
- Big Data storage needs to scale smoothly as volume increases
- KV map provides an easy representation of data providing O(1) write operations

KV Apps and Comparison with Relational DB



KV Store Algorithms

Log Structured Hash

- Supports iterate
- Requires less changes in FTL based L2P architecture
- Require more space for Hashed Mapping Tables => needs more DRAM

Log Structured Merge

- Requires HW Accelerator for merge sort
- Does not cover operation to iterate over a list of KV pairs
- Needs building Bloom Filters

Challenges In Designing KV SSD

Limited Compute and DRAM inside SSD

DRAM

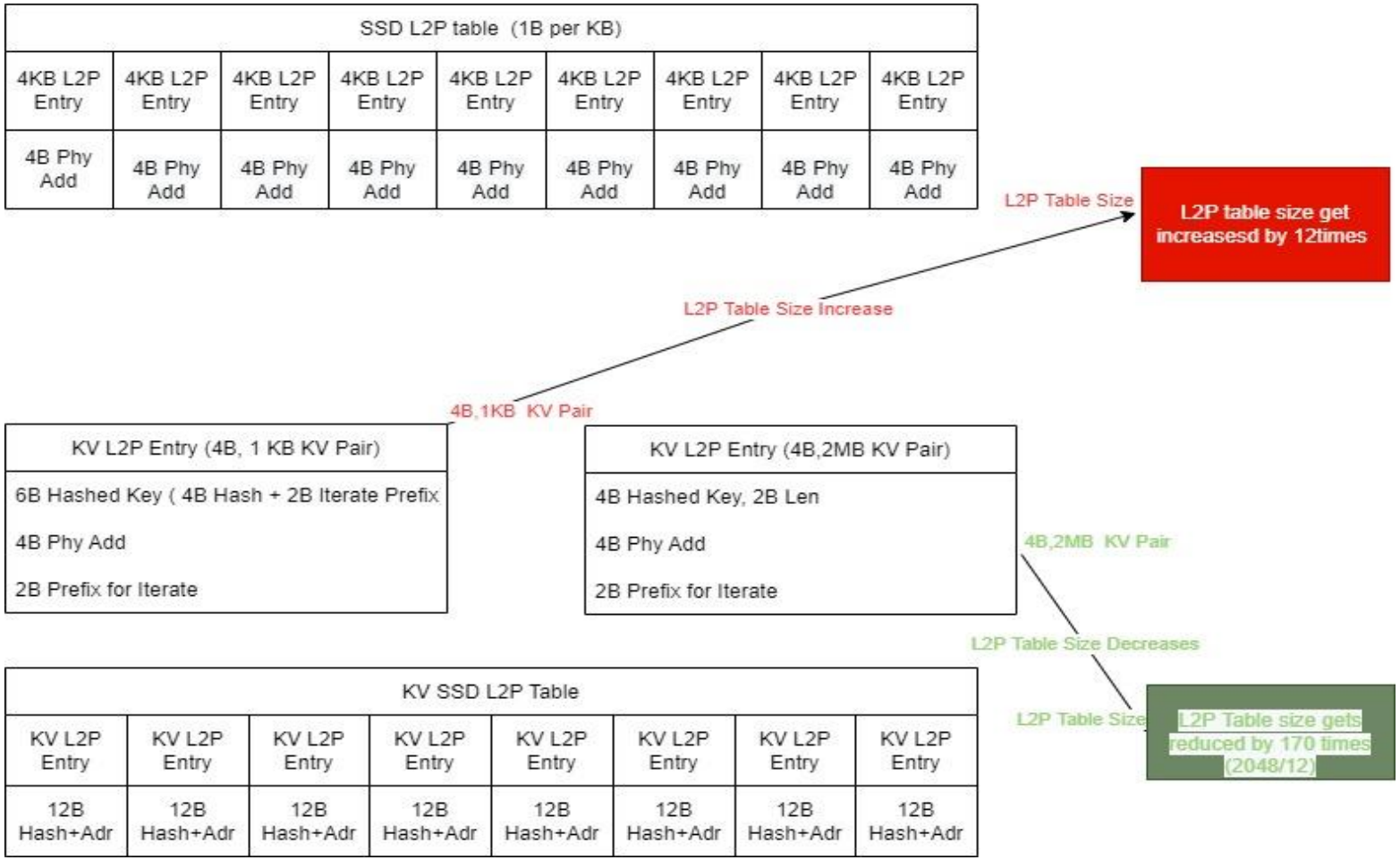
- SSDs usually have DRAM as much as 0.1% of NAND for L2P Cache indexing
- DRAM scales slower than NAND and costly than NAND, can’t add more

CPU

- SSD have low CPU Power
- Optimized for low power (5-15W active power consumption)

Impact Of Variable Size KV Pairs

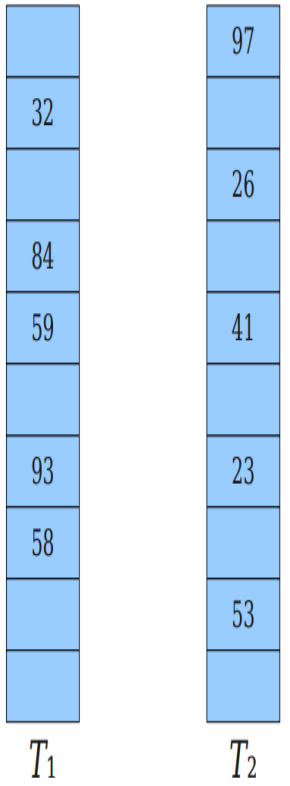
- L2P Tables in KV SSD require 12 Byte Entries instead of 4 Byte entries in Block SSDs
- L2P Tables in SSD can be greatly optimized by Fixed size KV Pairs. L2P Table size affects both NAND L2P Tables and DRAM caching
- Small Size (32B, 1KB) KV Pairs greatly increases the L2P Table Size in NAND by 12 times
- Large Size (32B, 2MB) KV Pairs greatly reduces the L2P Table size in NAND by 170 times



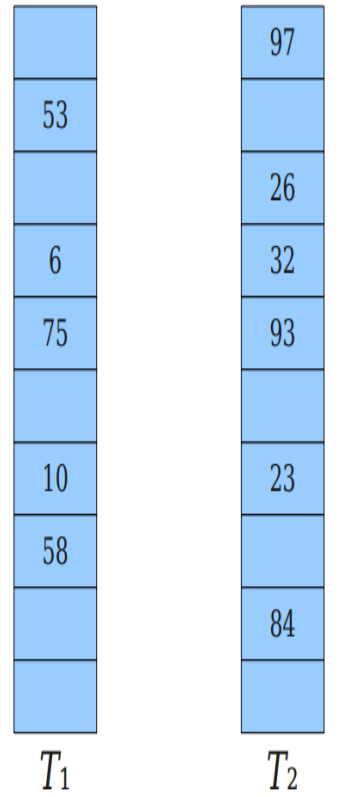
Need For Cuckoo Hashing?

- Hash collisions are costly and impacts KV SSD architecture
- Need separate mapping tables for collisions in cache and NAND.
- Cuckoo Hashing is a simple scheme for
  - Resolving collisions that allows for constant-time worst-case lookup and deletion operations
  - Amortized constant-time insertion operations
  - Maintains two hash tables T1 and T2 with two independent hash functions h1 and h2, respectively, and a key is stored in exactly one of two locations, with one possible location in each hash table

- Maintain two tables, each of which has m elements.
- We choose two hash functions h_1 and h_2 from $\mathcal{U}$ to $[m]$.
- Every element $x \in \mathcal{U}$ will either be at position $h_1(x)$ in the first table or $h_2(x)$ in the second.



- To insert an element x, start by inserting it into table 1.
- If $h_1(x)$ is empty, place x there.
- Otherwise, place x there, evict the old element y, and try placing y into table 2.
- Repeat this process, bouncing between tables, until all elements stabilize.



ZNS – Nice Alternative To Improve KV Latency

- Leverage Host ecosystem - ZenFS etc

- Interlocked host/device protocol, where the host manages placement and levels of OP can approach 0%
- Deliver maximum capacity and lowers SSD Cost

- ZNS solves
  - Data Center KV Store Latency problem
  - Storage disaggregation problem

- But the same problem exists in client and consumer workloads as
  - More and more apps caches media
  - Unstructured text used by Conference Apps

Summary

- SSD CPU and DRAM are scarce resources
  - Hashing, Merge, Sort operations of KV SSD can't work with limited CPU and DRAM resources of SSD
  - Need HW Accelerators for improved hashing, merging, sorting
- Variable size KV Pairs are not optimal for KV SSD L2P Tables
- ZNS ecosystem provides a nice alternative to solve the system latency problem of KV operations
- KV operations in Client SSD workloads on PCs remain a challenging problem to solve with limited CPU and DRAM resources of SSD

Thank You