TOSHIBA
Leading Innovation >>>

High Performance Hardware-Accelerated Flash Key-Value Store

Shingo Tanaka
Corporate R&D Center
Toshiba Corporation

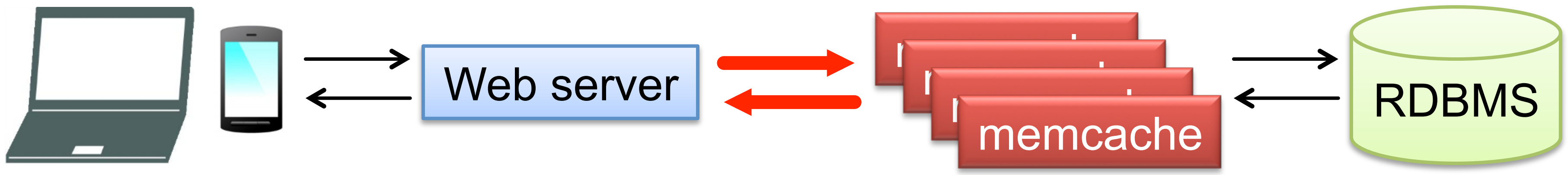
Christos Kozyrakis
EE & CS Department
Stanford University

Agenda

- **Key-value store**
  - Memcache
  - Inefficiencies with modern processors
- **Hardware-accelerated architecture**
  - Key designs
  - Micro benchmarks

Key-Value Store

- **Provides associative array**
  - Uses a hash table for the association
  - Values can be accessed by specifying its associated key
  - GET(key), SET(key, value), DELETE(key), etc
- **Memcache**
  - DRAM based data cache server in large scale web 2.0 systems which relieves the burden of back-end database
  - Thousands of memcache servers are used to meet the requirement of:
    - A significant amount of qps(query per second) throughput
    - Low(<1ms) query latency



Inefficiencies with modern processors

- **High instruction cache miss rates**
  - Instruction caches and associated next-line prefetchers are inadequate for scale-out workloads
- **High last-level data cache miss rates**
  - Last-level data cache becomes ineffective under the condition of random-access to a large size of memory
- **Underutilization of deep pipeline**
  - Low instruction- and memory-level parallelism in scale-out applications can not saturate the large core pipelines
- **Underutilization of bus bandwidth**
  - Scale-out workloads see no benefit from high memory and core-to-core communication bandwidth

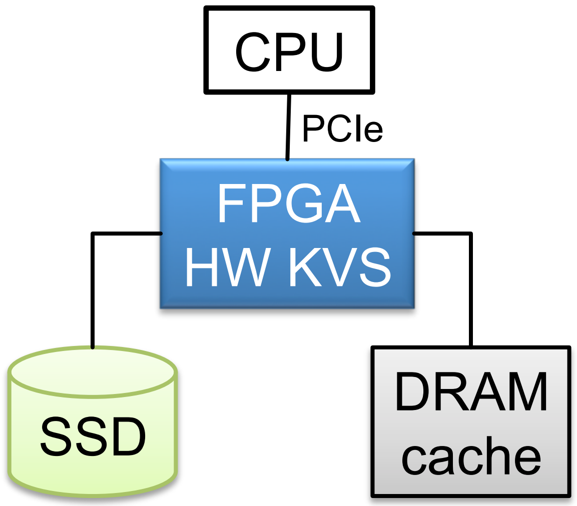
Big opportunity to improve the efficiency

M. Ferdman, et al, "A Study of Emerging Scale-out Workloads on Modern Hardware," In Proceedings of the 17th Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2012), 2012.

Objective & Approaches

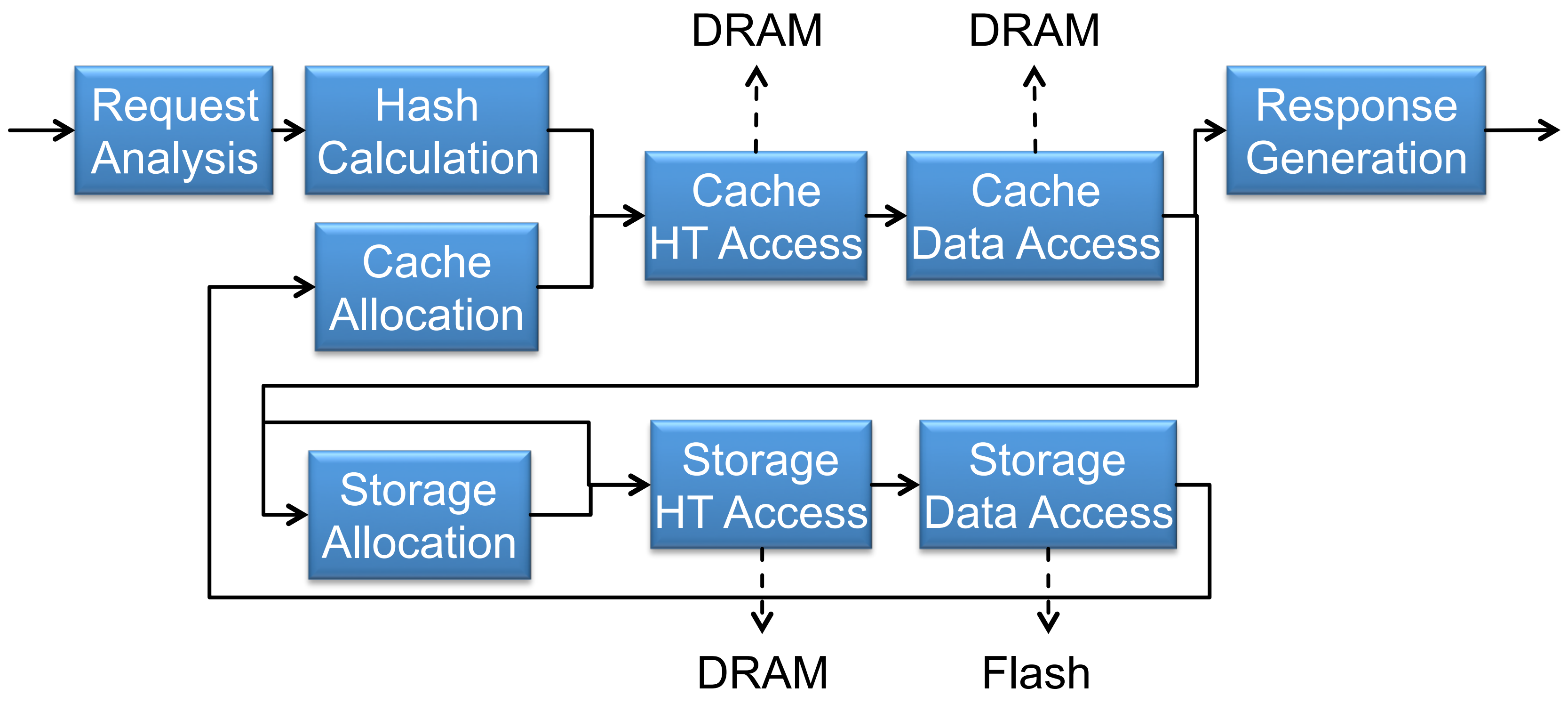
- **Achieve an order of magnitude higher performance by using HW accelerator**
  - Lower server cost, lower power consumption and smaller server space by consolidating 10 servers to 1

- **Approaches**
  - Full HW offloading
    - To get maximum performance without any CPU restriction
  - Leveraging both DRAM caching and flash storage
    - Keep using DRAM to provide optimal bit cost throughout the workload



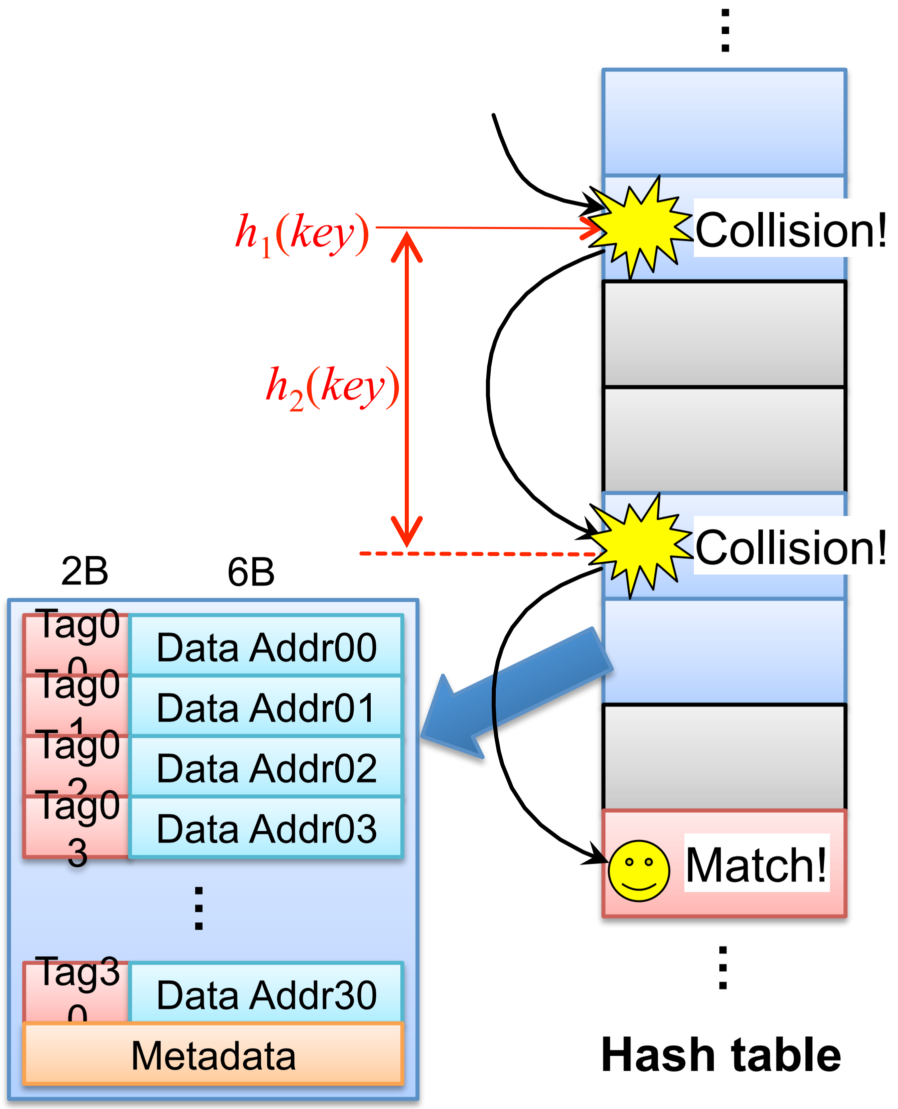
KVS block diagram

- **Fully pipelined architecture throughout query processing**
  - Each pipeline stage processes each query in <10 cycles at 200MHz
  - Each block is connected to each other via common streaming interface
    - Packet communication (common header + key-value payload)



Hashing

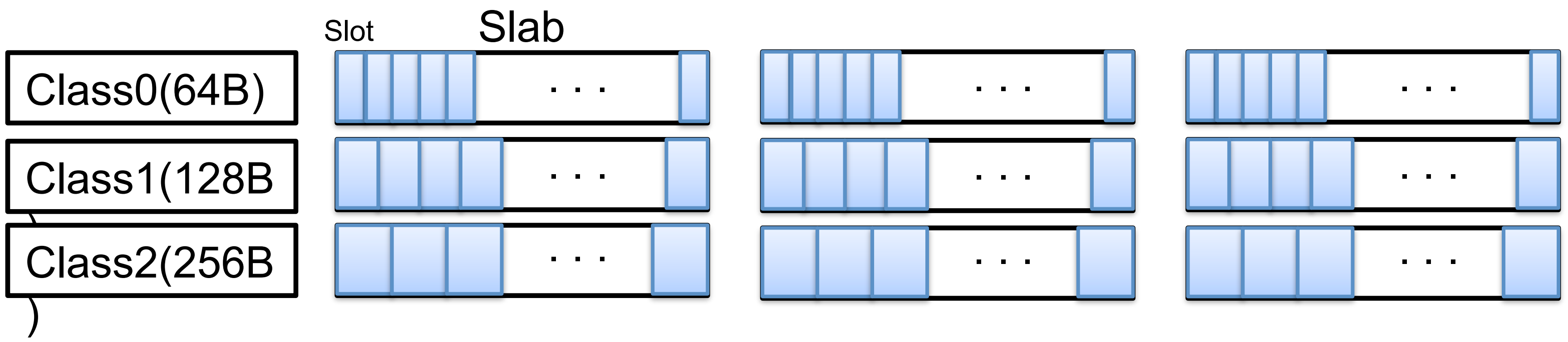
- **HW-aware algorithm**
  - Easy for HW implementation with good enough performance
    - Double hashing
      - 1st func: 1st place to start
      - 2nd func: Interval to probe
    - Bucket hashing
      - Leverage wide parallelism of hardware processing and wide bandwidth of DRAM seq access
    - Averagely 1 access to reach the expected entry ~90% occupancy
  - Use 2 bytes tag for key matching before accessing data on cache/ flash
    - Reduce hash table size on DRAM for key-value data on flash
    - < 1% false positive



Bucket data structure(256B)

DRAM Caching

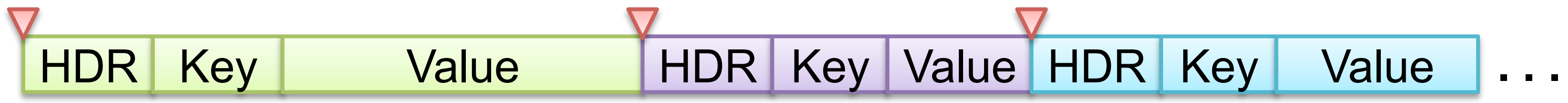
- **Slab-based data structure**
  - Classified by data size(header + Key + Value)
    - 64B, 128B, 256B, 512B, …, 1MB (Scale factor = 2)
  - Every class uses 10MB "Slab"
  - Each cache data is stored in "slot"



  - Caching policy: FIFO
    - Minimum DRAM access
    - Evicted data is chosen sequentially in a cyclic manner within a class

Data structure on Flash

- **Log-structured format**
  - Every key-value is stored in order SET commands come
  - Each head addr is stored in hash table
  - DELETE only deletes corresponding hash table entry
    - Actual data is deleted by background garbage collection

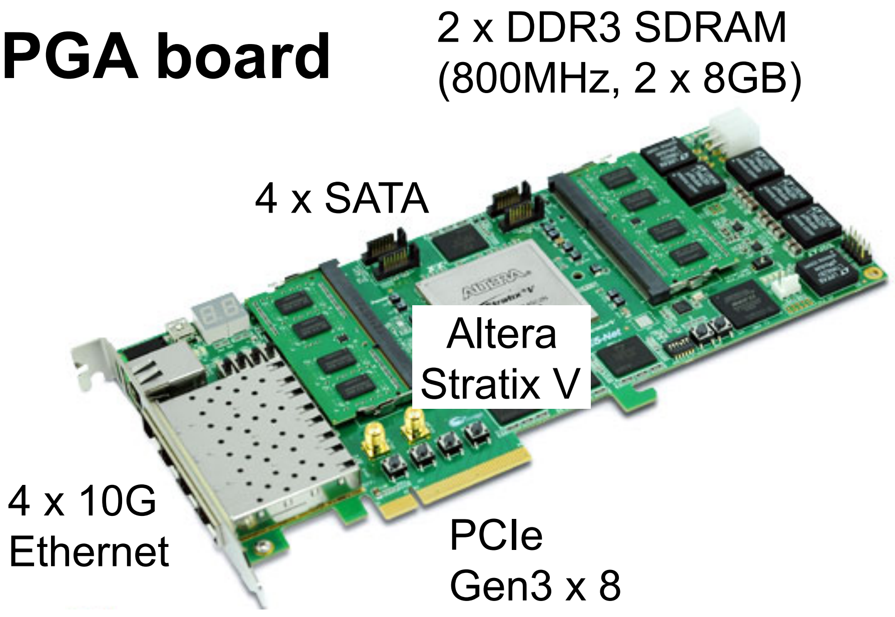


  - Access pattern
    - Write: sequential access
    - Read: random access

Prototyping

- **Micro benchmarks using FPGA board**
  - Throughput
  - Latency
  - Power consumption (Estimate)

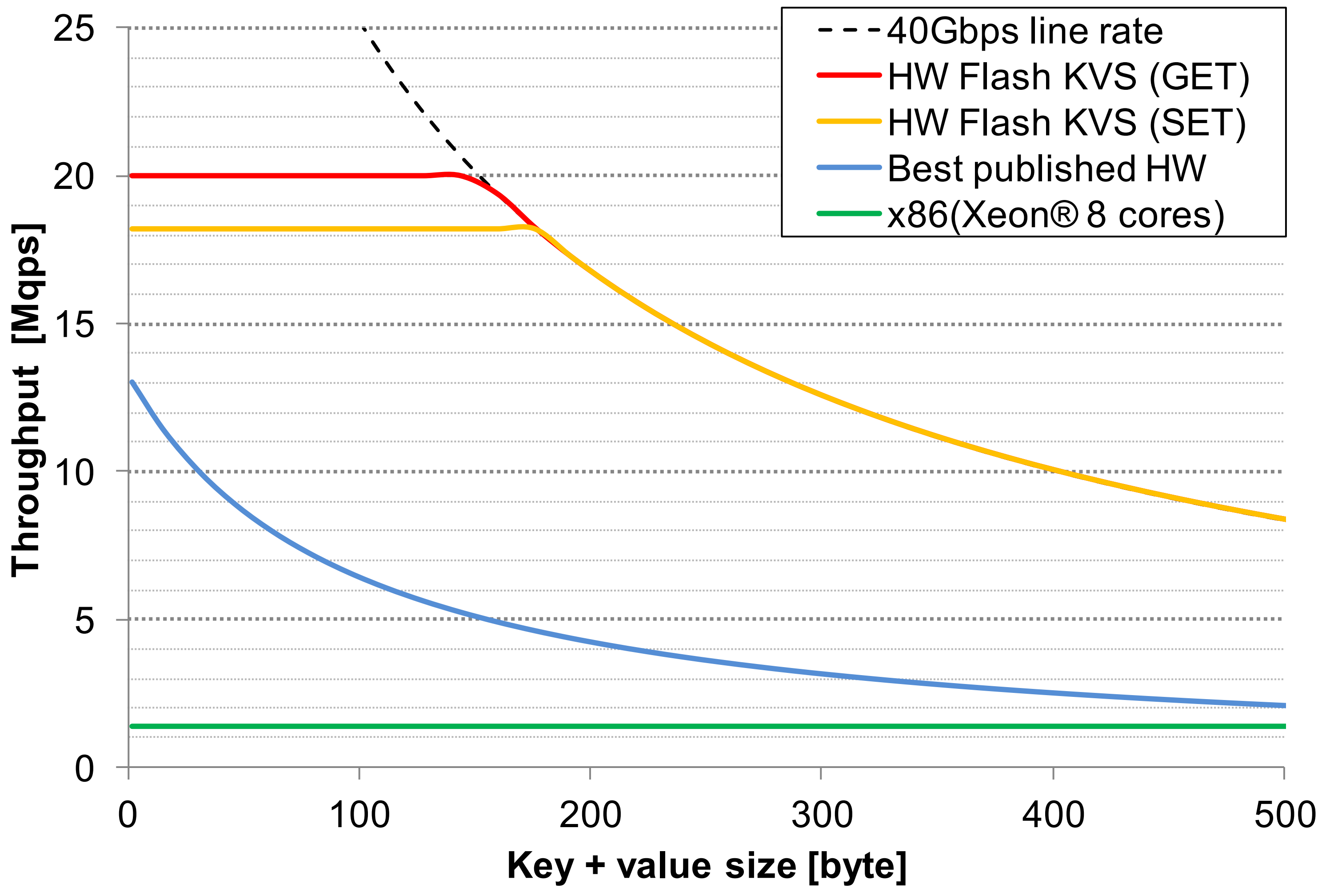
- **Restrictions**
  - 8GB DRAM for data cache
  - Emulated SSD behavior
  - UDP memcache binary protocol
  - GET/SET/DELETE commands



Terasic Net-DE5

http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=158&No=526

Throughput (cache hit)

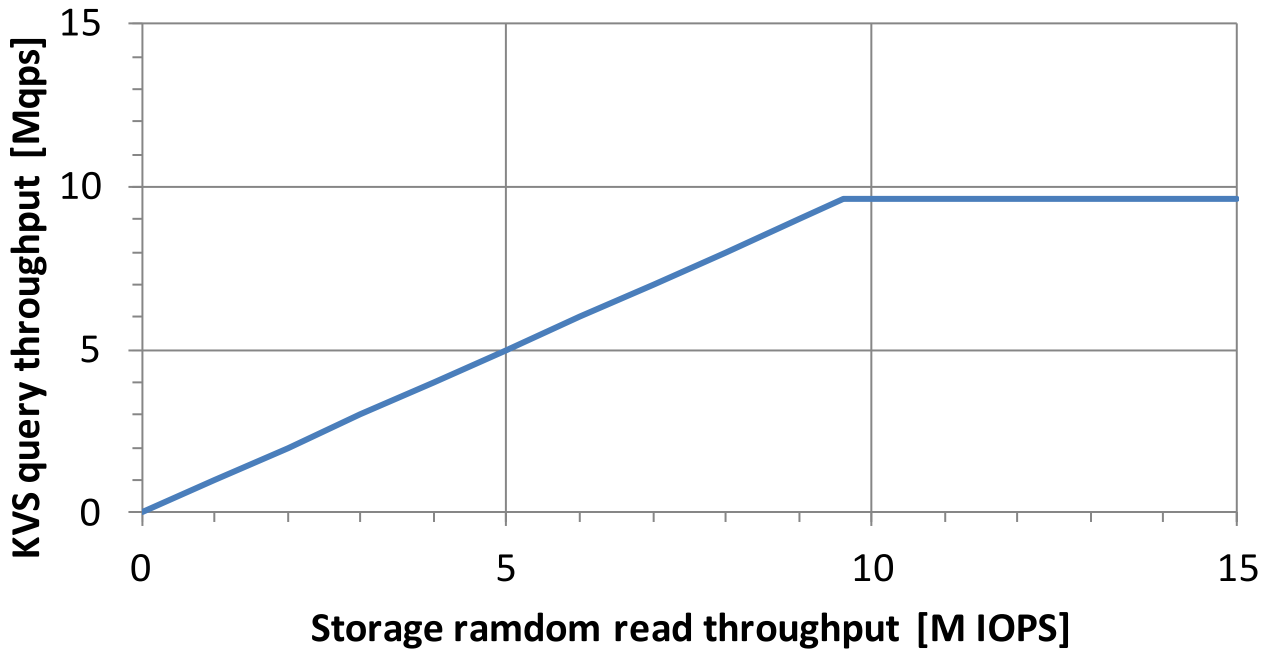


Achieved 20M qps

Throughput (cache miss)

- **GET/SET 9.1M qps**
  - Lower than cache hit because of cache eviction
  - But requires the same amount of storage IOPS
    - To exact Key matching on flash
    - Practically limited by storage IOPS (<9.1M IOPS)

-> high Enough to exploit current SSD performance



Energy efficiency and latency

- **Energy efficiency**
  - FPGA 15W (static analysis)
  - HW Flash KVS: 20M qps/ (100W + 15 W) = 173K qps/W

    (x86: 1.4M qps / 100 W =14K qps/W)

- **Latency**
  - KVS core (RX to TX): 1us (cache hit), 2us (cache miss)
  - Ethernet MAC/PHY (RX + TX): 2us
  - TOE (RX + TX, estimate): 1us
  - Total: 4us (cache hit)

    55us (cache miss (SSD: 50us))

    (x86: 200 ~ 300us)
  - Constant with workload variation

Summary & Next

- **Achieved >10x higher performance key-value store than x86-based memcache with HW accelerator**
  - Fully pipelined architecture and HW-aware algorithms
  - DRAM caching and flash storage
  - Cache hit: 20M qps throughput, 4us latency
  - Cache miss: ~9.1M qps throughput, 55 us latency (SSD 50us)

- **Next**
  - TCP/IP support
  - Use large host memory(~1TB) for data cache with DMA via PCIe
  - Trials in real use cases