# NetCache: Balancing Key-Value Stores with Fast In-Network Caching

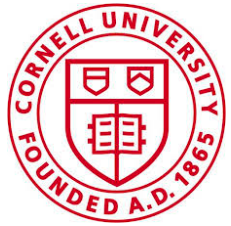
Xin Jin

Xiaozhou Li, Haoyu Zhang, Robert Soule, Jeongkeun Lee,

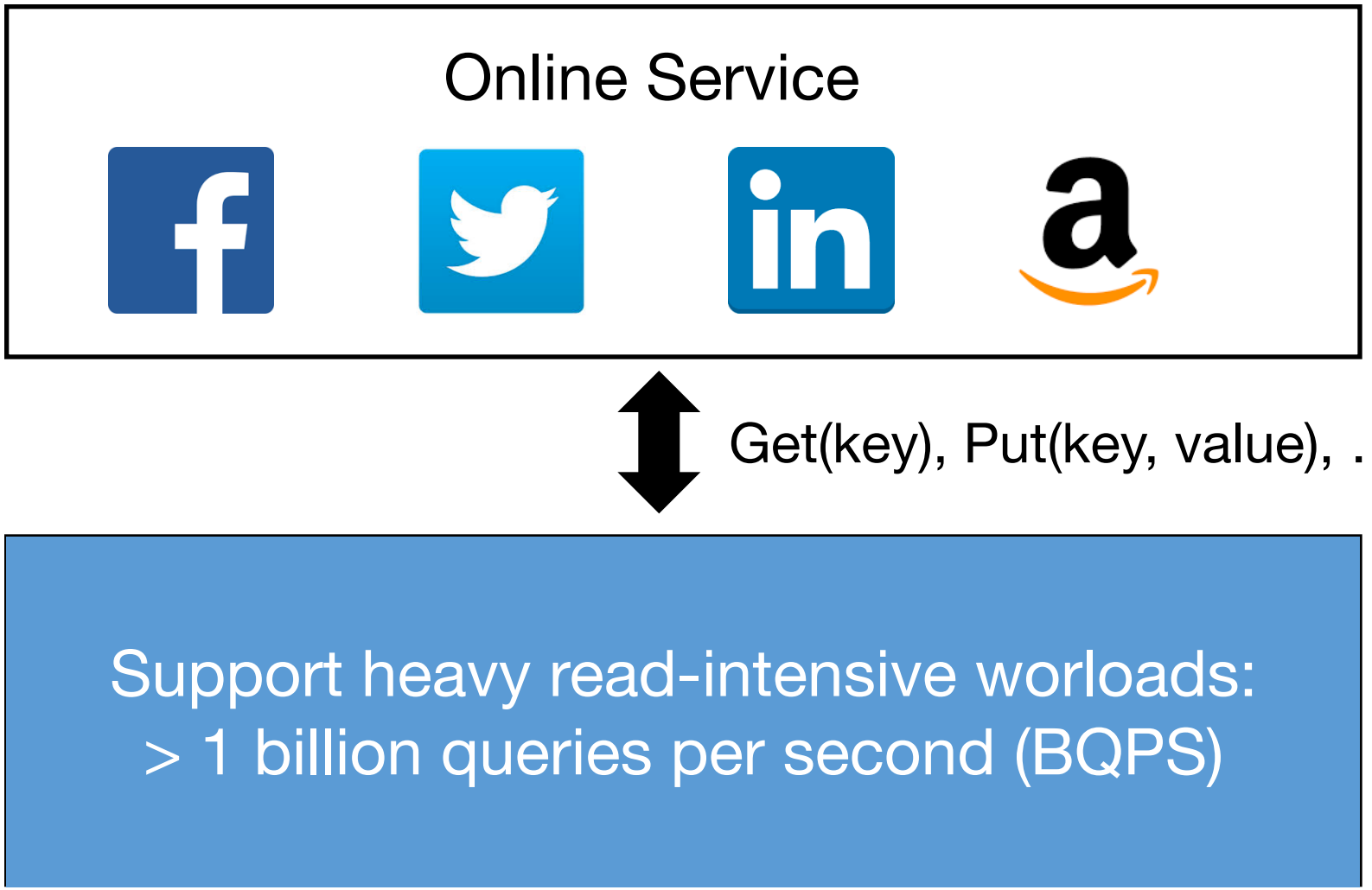
Nate Foster, Changhoon Kim, Ion Stoica



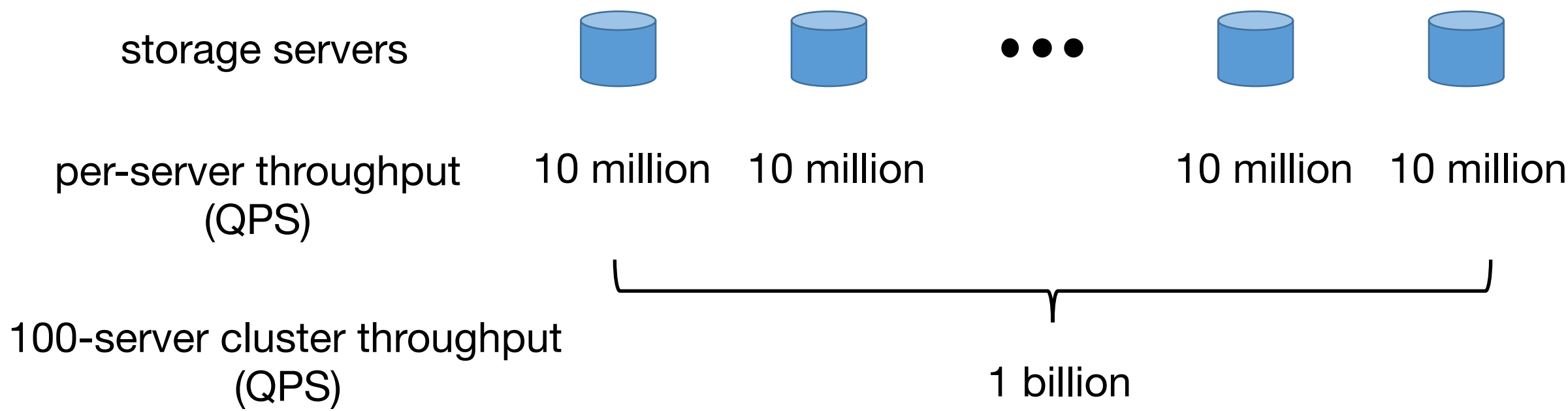


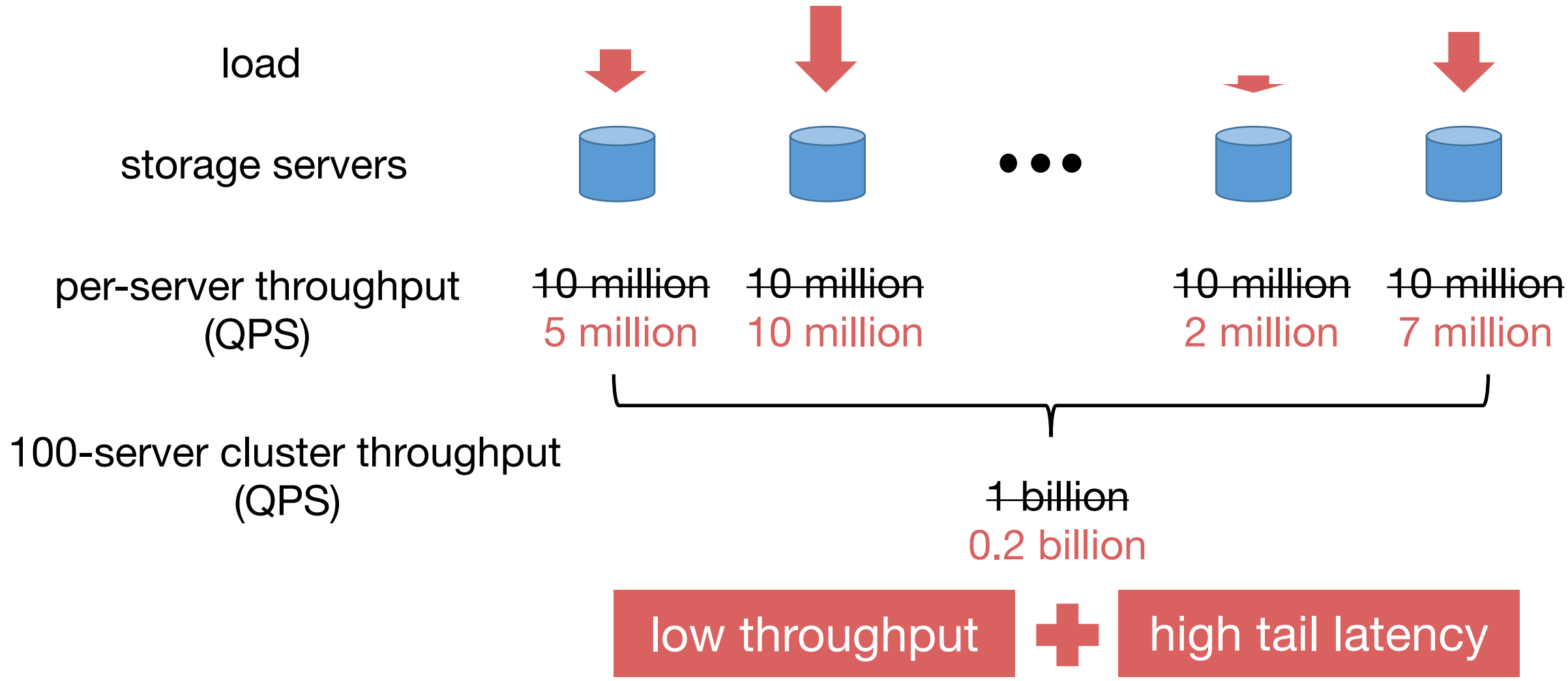
# Key-value stores power online services



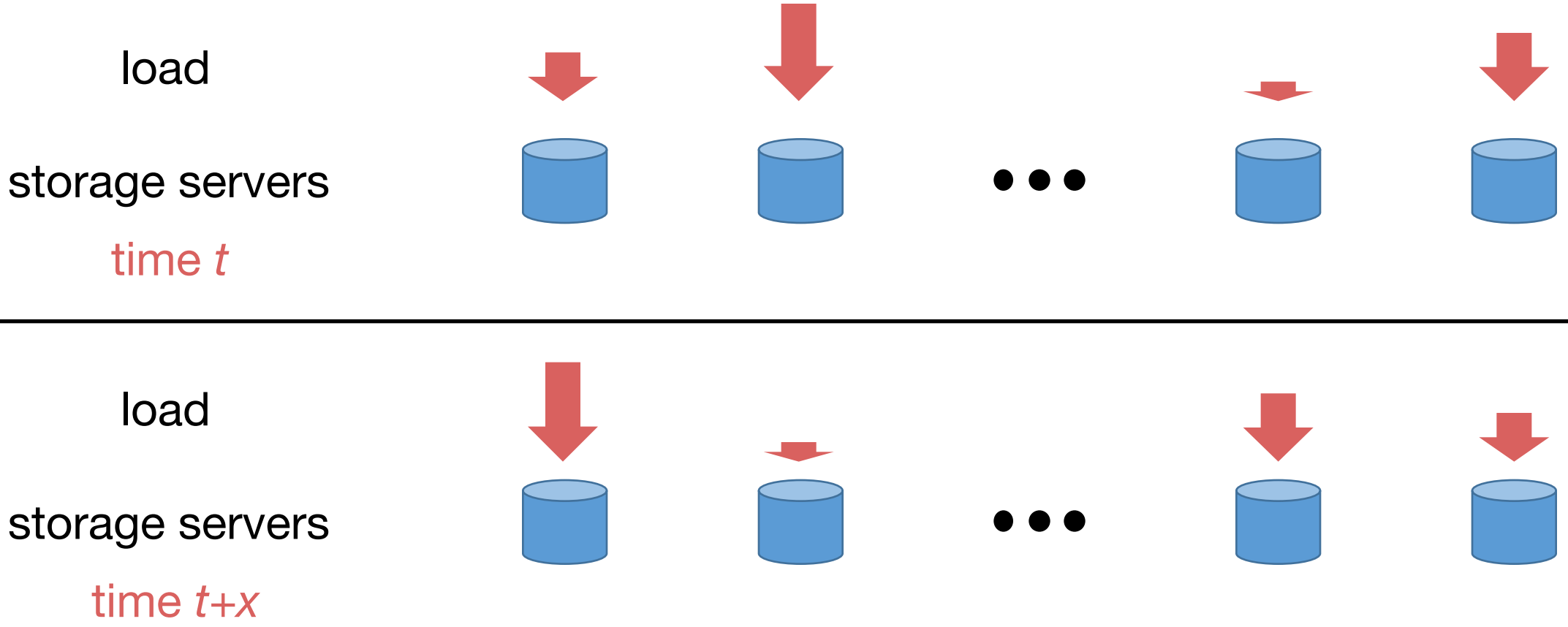
# Scale out key-value stores for high-performance



# Key challenge: Dynamic load balancing

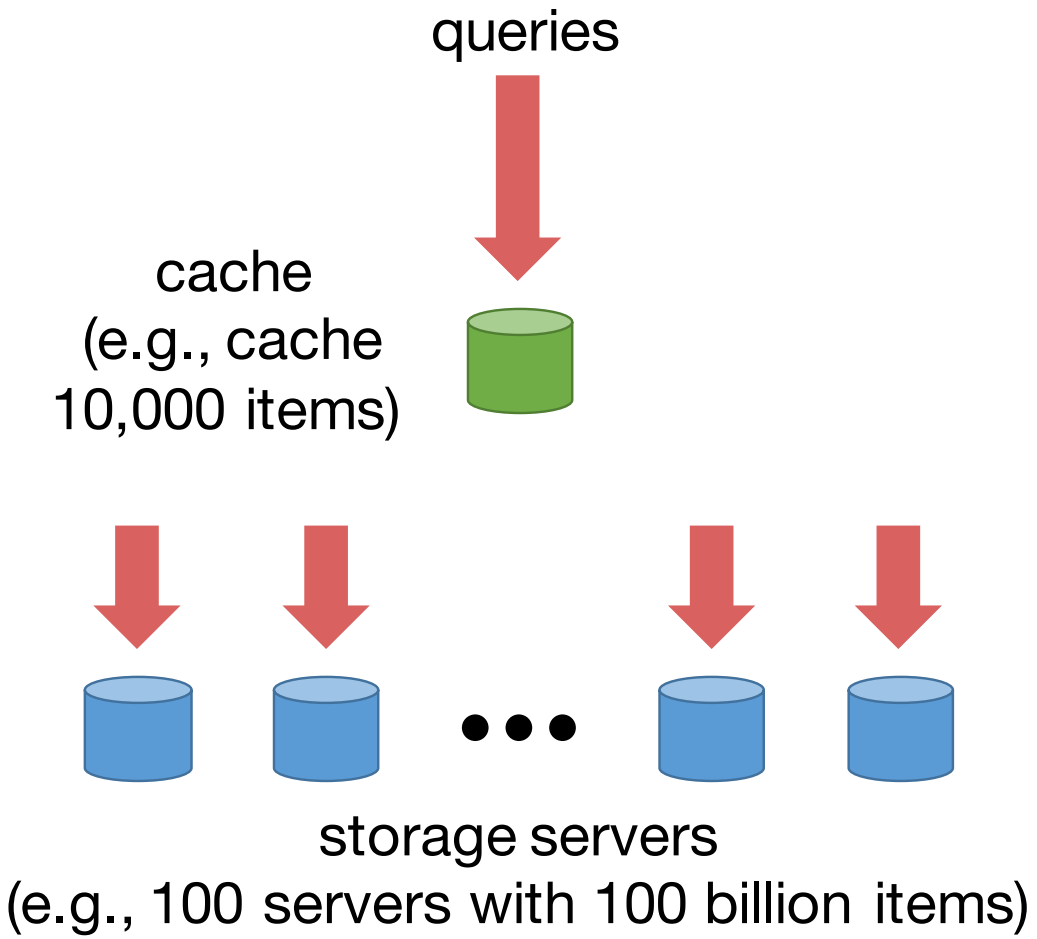


# Key challenge: Dynamic load balancing




How to handle highly-skewed and rapidly-changing workloads?

# Fast, small cache for load balancing




- Cache **O(*N* log *N*)** items [Fan, SOCC'11]
  - N: number of servers
- **Performance guarantee**
  - Throughput: N·T
    - T: per-server throughput
  - Latency: bounded queue length (no server receives more than T load)
  - Regardless of workload **skewness**
- **Requirement**
  - **Cache throughput ≈ N·T**

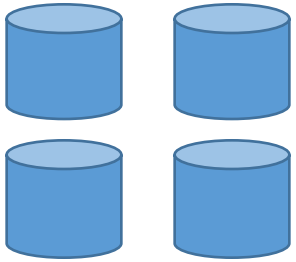
# Towards in-memory key-value stores

cache layer

in-memory

O(10) MQPS

O(1) BQPS



storage layer


flash/disk

per: O(100) KQPS

**total: O(10) MQPS**

towards **BQPS** key-value stores

in-memory

per: O(10) MQPS

**total: O(1) BQPS**

# Towards in-memory key-value stores

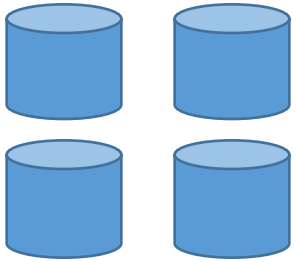
cache layer

in-memory

O(10) MQPS

**in-network**

O(1) BQPS



storage layer


flash/disk

per: O(100) KQPS

**total: O(10) MQPS**

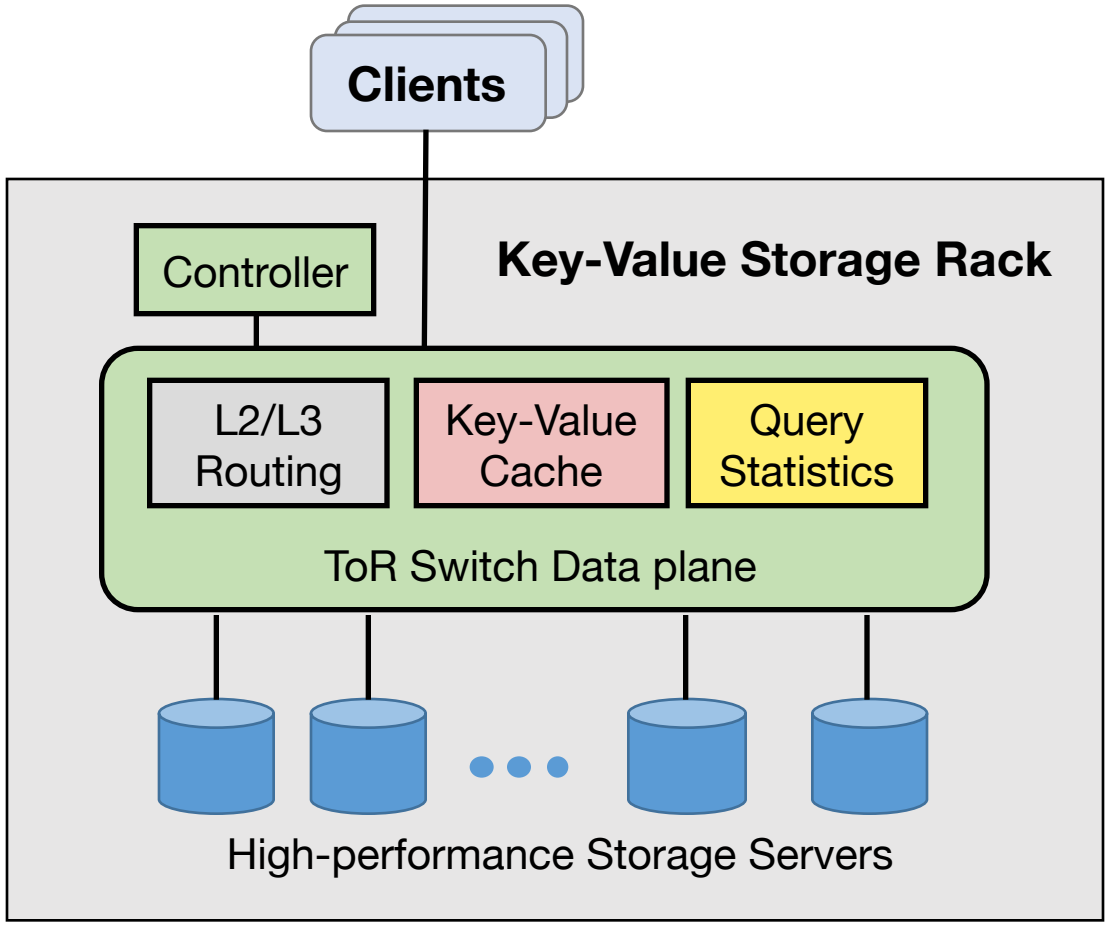
towards **BQPS** key-value stores

in-memory

per: O(10) MQPS

**total: O(1) BQPS**

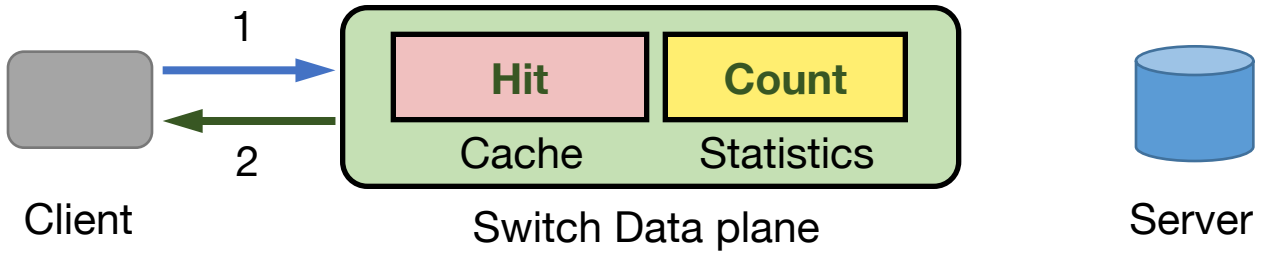
# NetCache Architecture



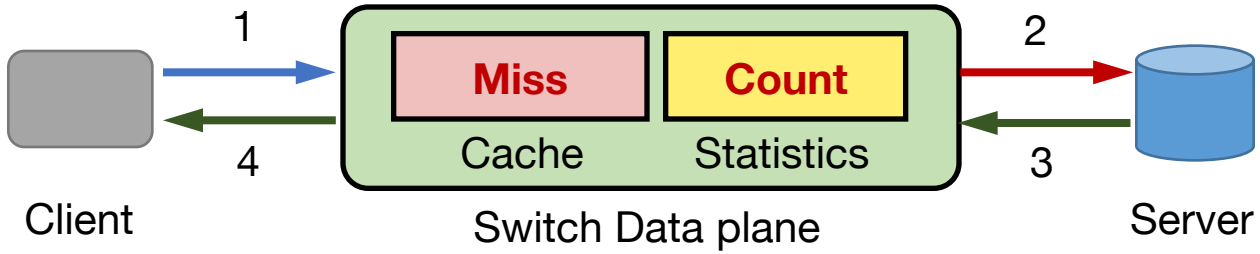

- **Performance guarantee**
  - BQPS throughput with bounded latency with a single rack
  - Regardless of workload skewness
- **Data plane**
  - Unmodified routing
  - Key-value cache to serve hot items
  - Query statistics to detect hot items
- **Control plane**
  - Update cache with hot items
  - Handle dynamic workloads

# Query Handling

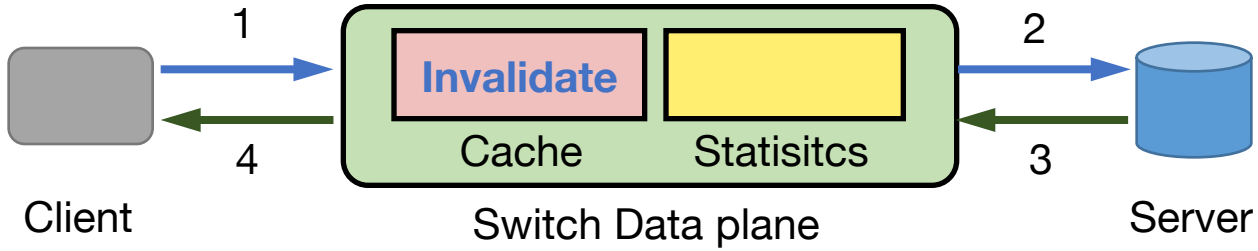
Read Query:
Cache Hit




Read Query:
Cache Miss

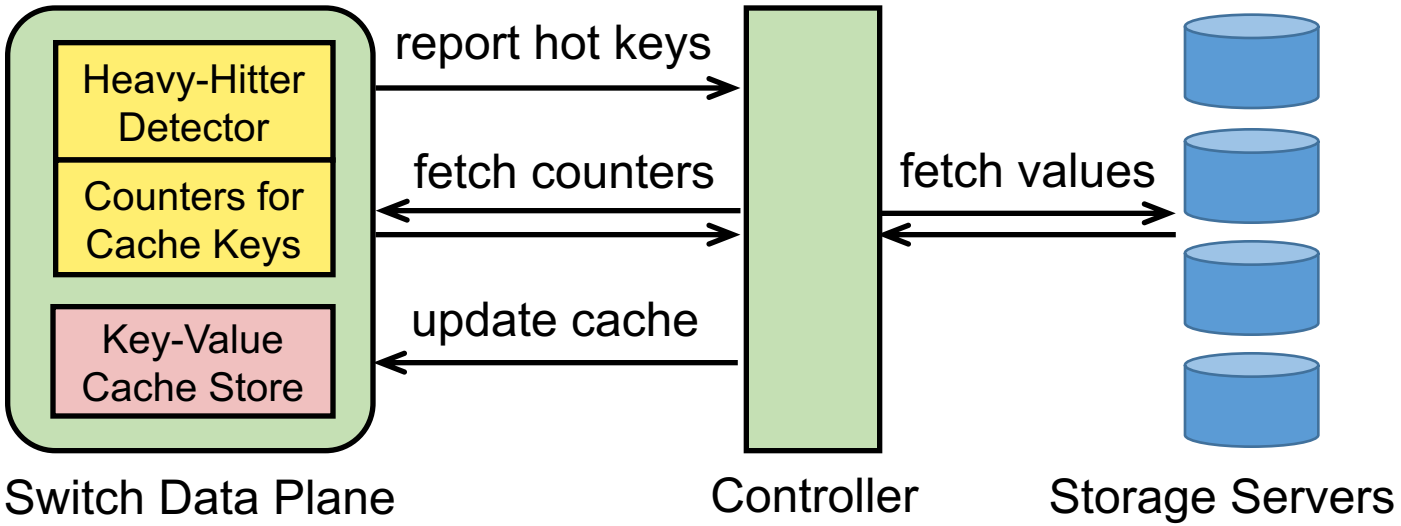



Write Query



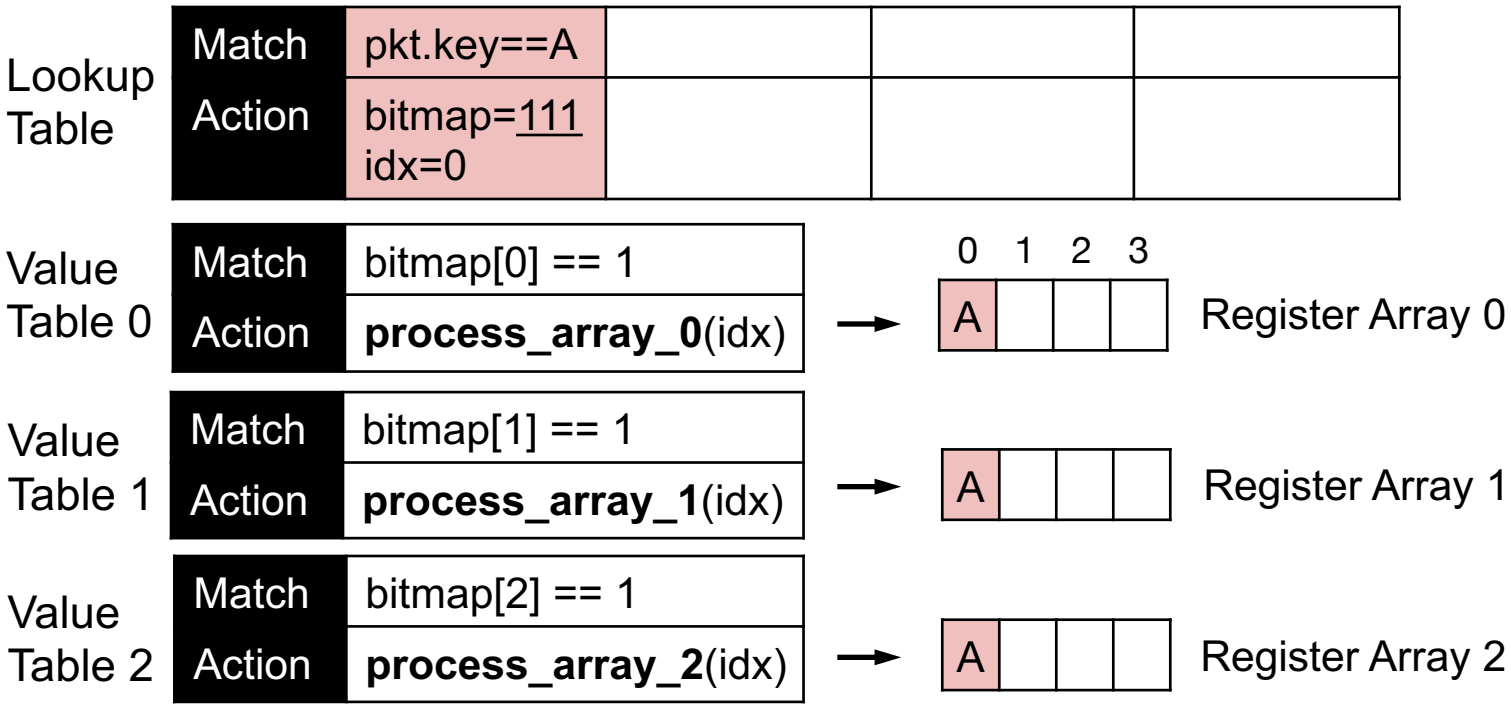

Cache coherence: write-through in the data plane

# Cache Update



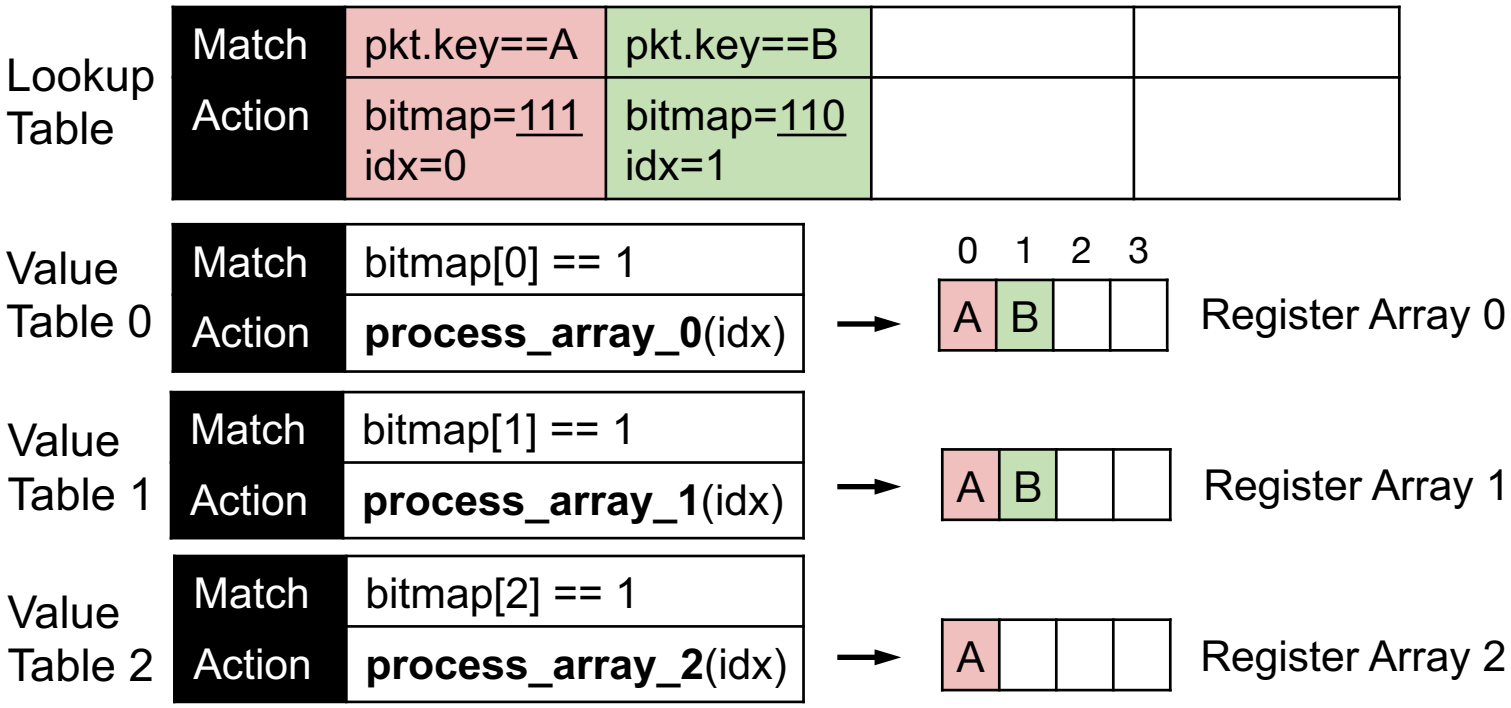

- Compare counters of **new hot keys** and **cached keys**
- Use **sampling** to avoid fetch counters of all cached keys

# Variable-Length On-Chip Key-Value Cache



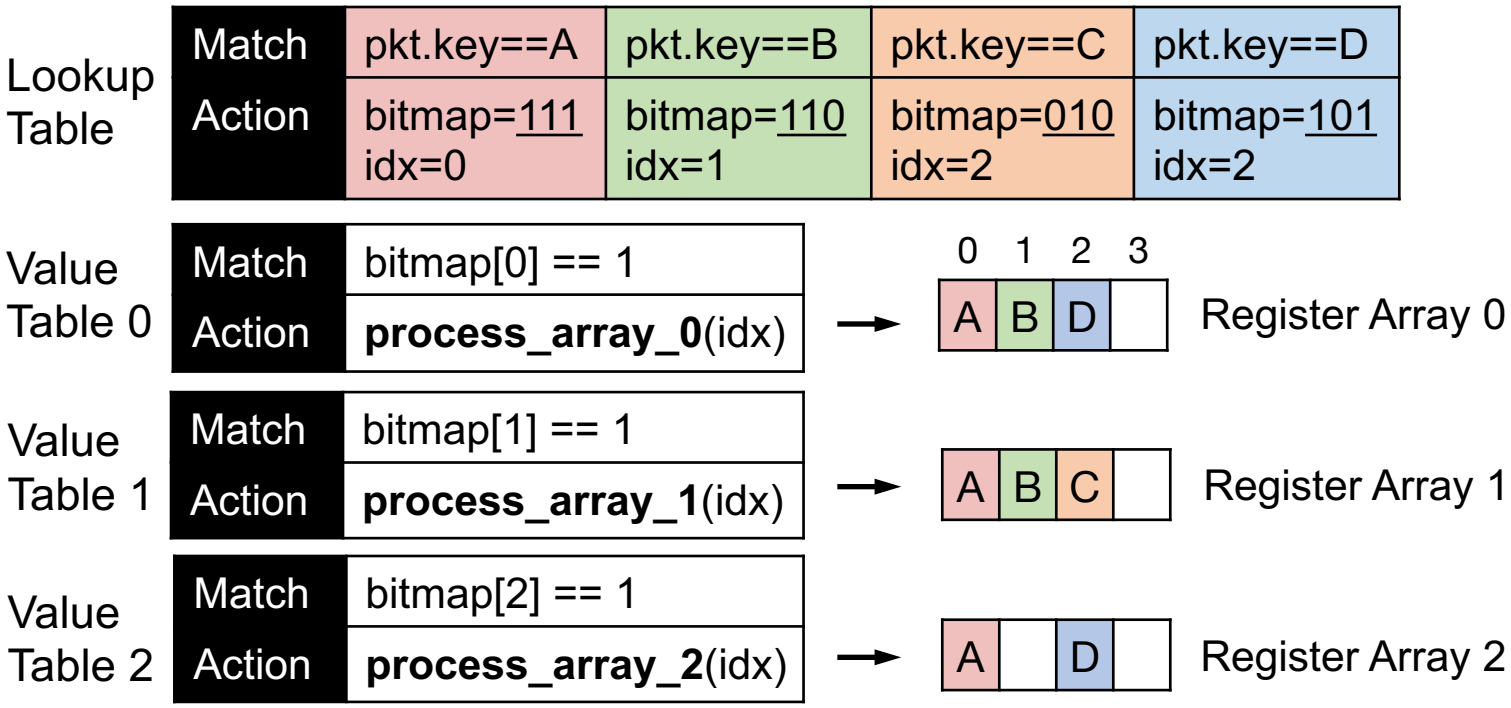

- Lookup table: map a key to a bitmap and an index
- Value table: store value in register arrays

# Variable-Length On-Chip Key-Value Cache



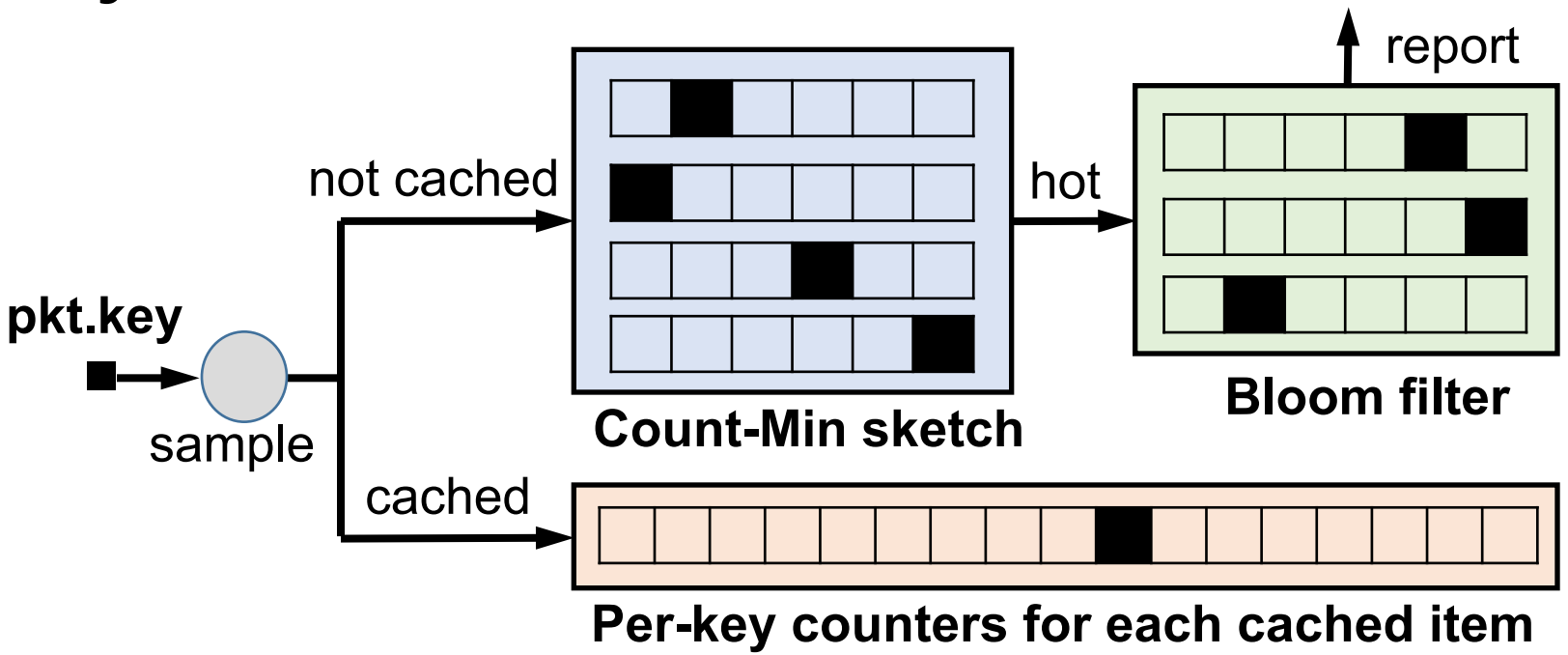

- Lookup table: map a key to a bitmap and an index
- Value table: store value in register arrays

# Variable-Length On-Chip Key-Value Cache




- Lookup table: map a key to a bitmap and an index
- Value table: store values in register arrays

# Query Statistics



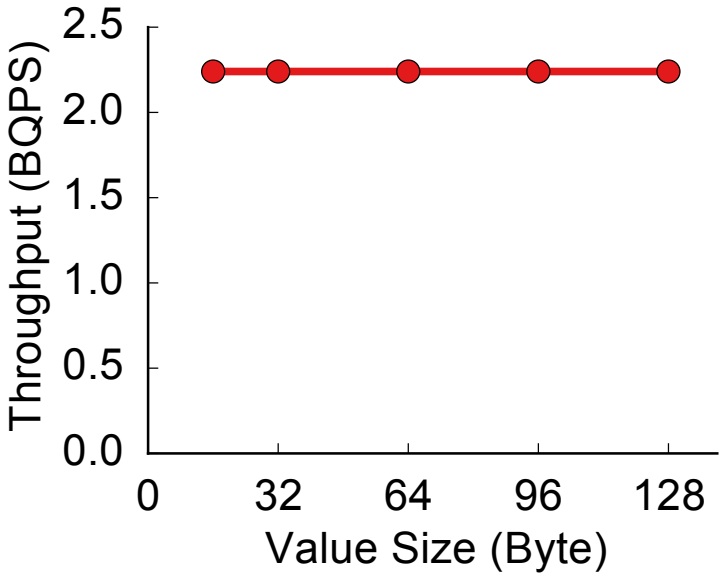

- New hot key
  - Count-Min sketch: report new hot keys
  - Bloom filter: remove duplicate hot key reports
- Cached key: per-key counter array
- Sample: reduce memory usage

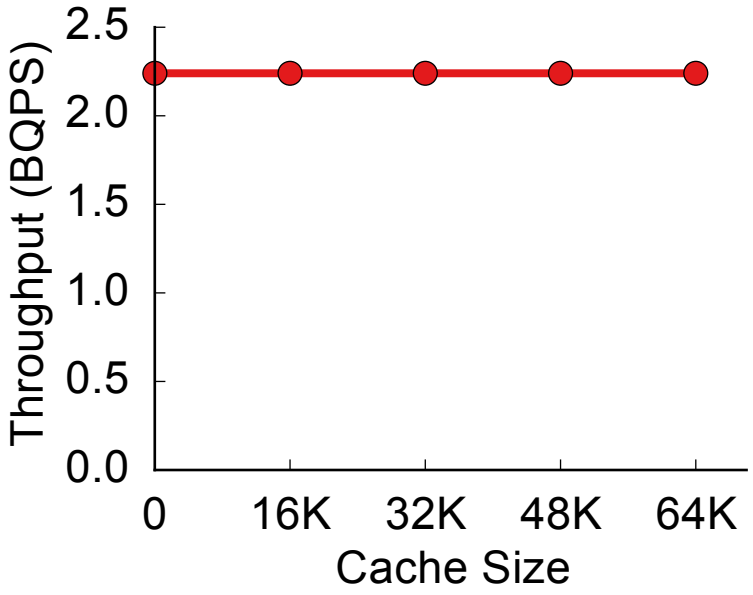
# Implementation

- **Switch: Barefoot Tofino**
  - Throughput: 6.5 Tbps, 4+ bpps; Latency: <1 us
  - Routing: standard L3 routing
  - Key-value cache: 64K items with 16-byte keys and 128-byte values
  - Query statistics: 256K entries for Count-Min sketch, 768K entries for Bloom filter

- **Storage Server**
  - 16-core Intel Xeon E5-2630, 128 GB memory, 40Gbps Intel XL710 NIC
  - Intel DPDK for optimized IO, TommyDS for in-memory key-value store
  - Throughput: 10 MQPS; Latency: 7 us

# Evaluation: Switch Microbenchmark

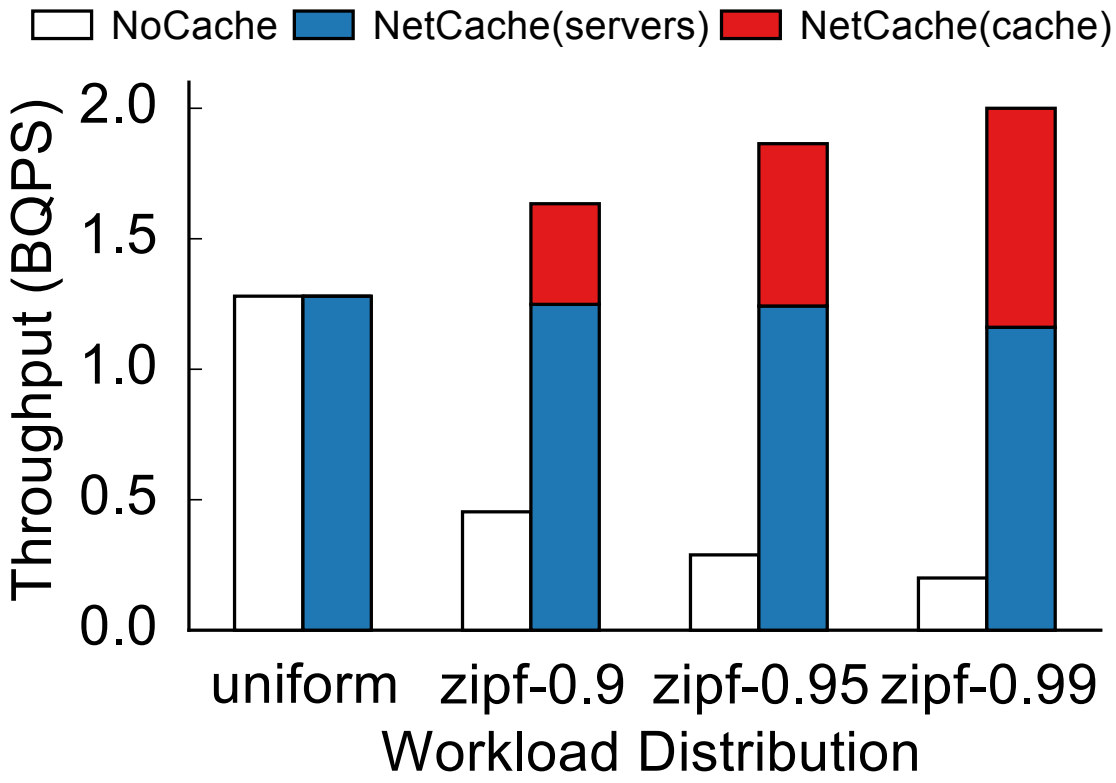



NetCache switch can process **2+ BQPS** for up to **64K items** with **16-byte keys** and **128-byte values**. (Larger values can be supported with more stages and e2e mirroring.)

# Evaluation: System Performance

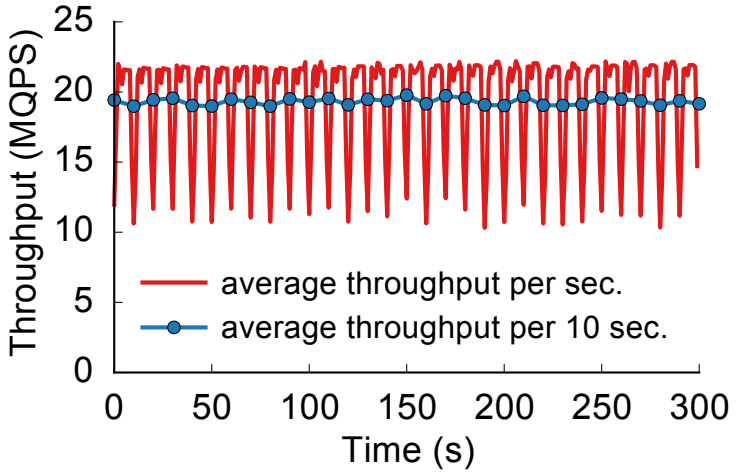
Throughput of a key-value storage rack with one Tofino switch and 128 storage servers.



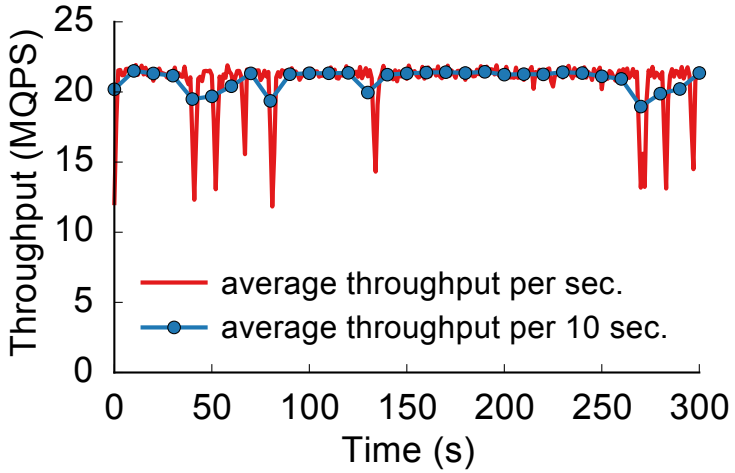

NetCache provides **3-10x throughput improvements**.

# Evaluation: Handling Workload Dynamics

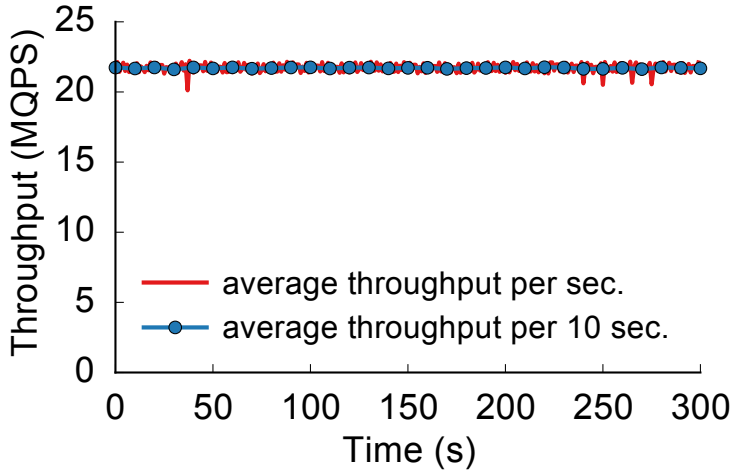
hot-in workload
(radical change)




random workload
(moderate change)




hot-out workload
(small change)




NetCache **quickly and effectively reacts** to a wide range of workload dynamics.

# Conclusion

- **NetCache** is a new key-value store architecture that uses **in-network caching** to balance in-memory key-value stores.
- NetCache exploits programmable switches to efficiently **detect, index, cache and serve** hot items **in the data plane**
- NetCache provides high performance even under **highly-skewed and rapidly-changing** workloads

# Thanks!