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

Xin Jin, **Xiaozhou Li**, Haoyu Zhang, Robert Soulé
Jeongkeun Lee, Nate Foster, Changhoon Kim, Ion Stoica

NetCache is a **rack-scale key-value store** that leverages **in-network data plane caching** to achieve

billions QPS throughput & **~10 μs latency**

even under

highly-skewed & **rapidly-changing**

workloads.

New generation of systems enabled by programmable switches ☺

Goal: fast and cost-efficient rack-scale key-value storage

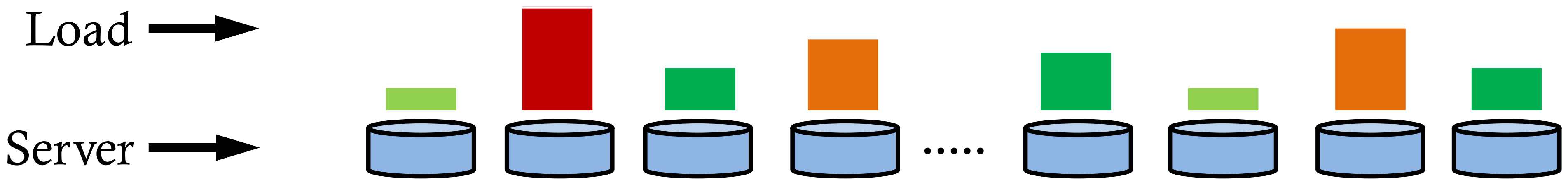
- **Store, retrieve, manage key-value objects**
  - Critical building block for large-scale cloud services

    . . .
  - Need to **meet aggressive latency and throughput objectives efficiently**

- **Target workloads**
  - Small objects
  - Read intensive
  - **Highly skewed and dynamic key popularity**

Key challenge: highly-skewed and rapidly-changing workloads

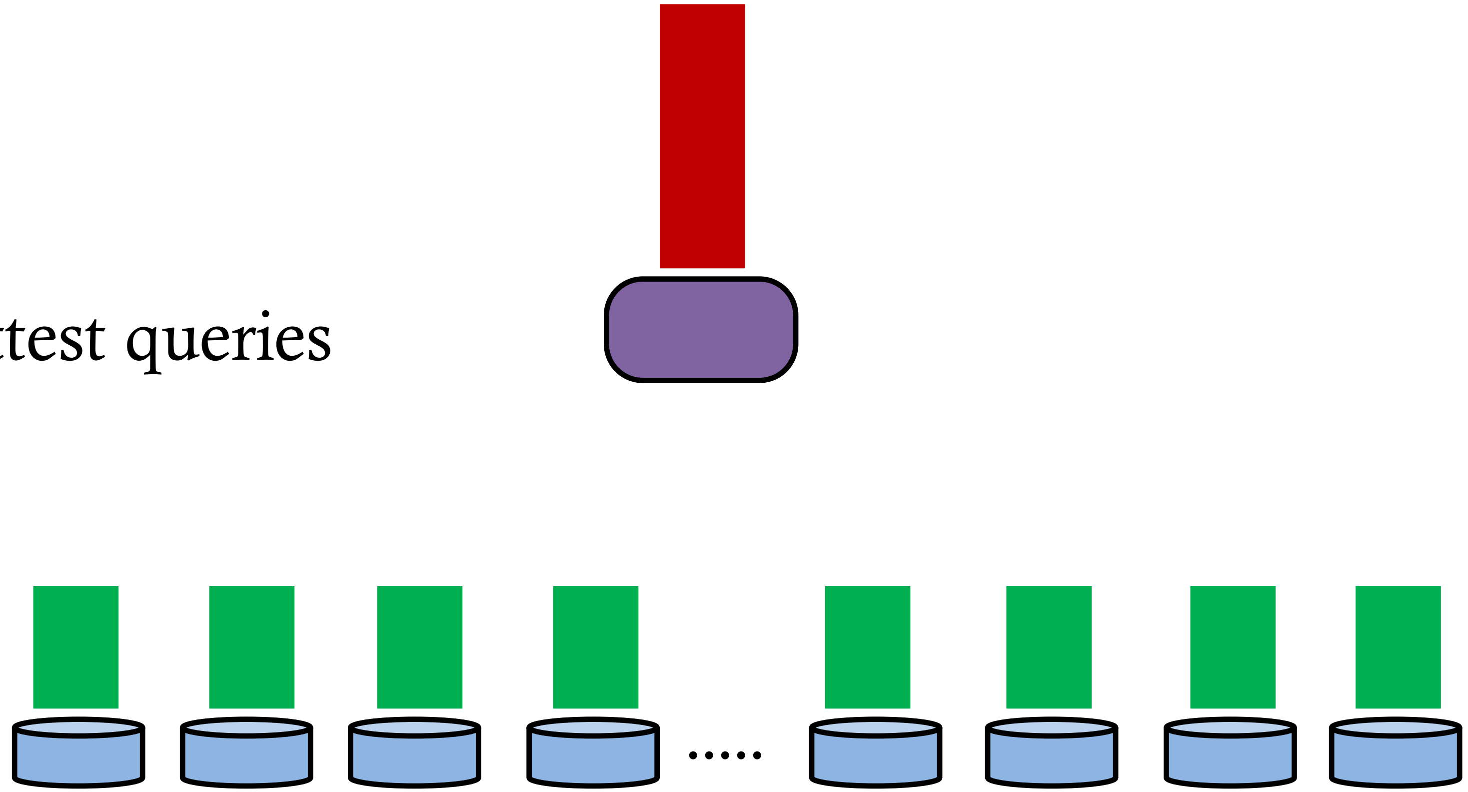
low throughput & high tail latency



Q: How to provide effective dynamic load balancing?

Opportunity: fast, small cache can ensure load balancing

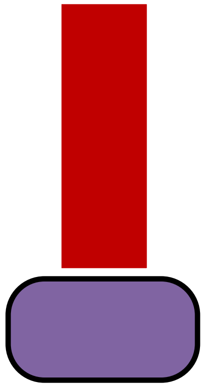
Cache absorbs hottest queries



Balanced load

Opportunity: fast, small cache can ensure load balancing

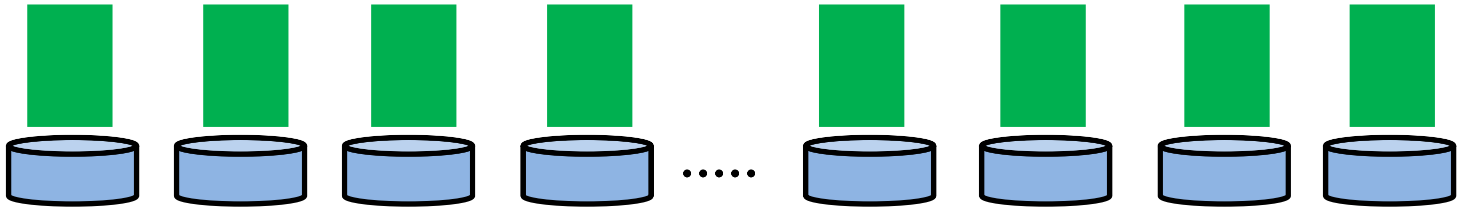
[B. Fan et al. **SoCC'11**, X. Li et al. **NSDI'16**]



Cache **O(*N* log *N*)** hottest items

E.g., 10,000 hot objects

N: # of servers

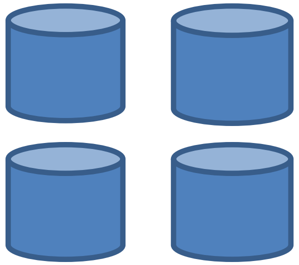


E.g., 100 backends with 100 billions items

Requirement: cache throughput ≥ backend aggregate throughput

NetCache: towards billions QPS key-value storage rack

Cache needs to provide the **aggregate** throughput of the storage layer

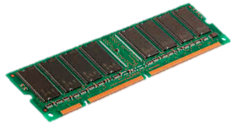


flash/disk

each: O(100) KQPS

total: O(10) MQPS

cache →

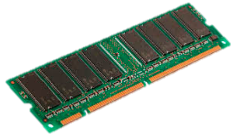


in-memory

O(10) MQPS

storage layer

cache layer



in-memory

each: O(10) MQPS

total: O(1) BQPS

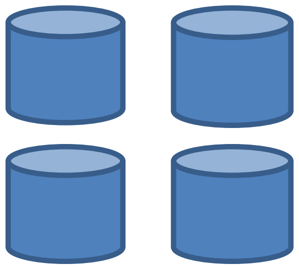
cache →

?

O(1) BQPS

NetCache: towards billions QPS key-value storage rack

Cache needs to provide the **aggregate** throughput of the storage layer



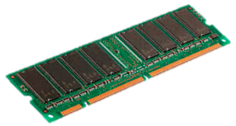
flash/disk

each: O(100) KQPS

total: O(10) MQPS

storage layer

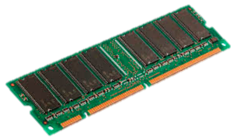
cache



in-memory

O(10) MQPS

cache layer



in-memory

each: O(10) MQPS

total: O(1) BQPS

cache

in-network

O(1) BQPS

Small on-chip memory?

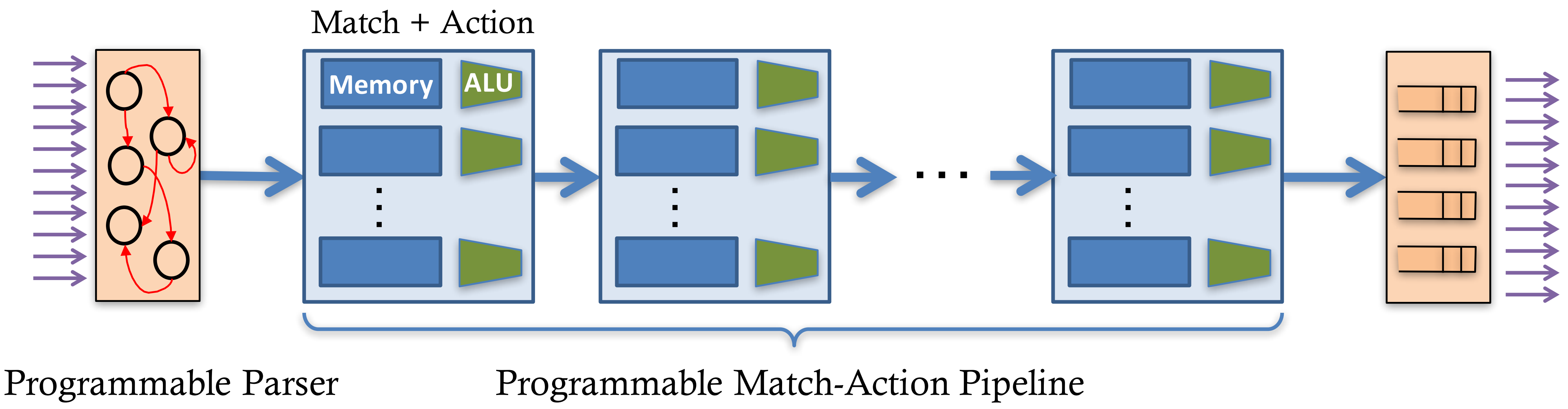
Only cache **O(*N* log *N*) small** items

Key-value caching in network ASIC at line rate ?!

- ❑ How to identify application-level packet fields ?
- ❑ How to store and serve variable-length data ?
- ❑ How to efficiently keep the cache up-to-date ?

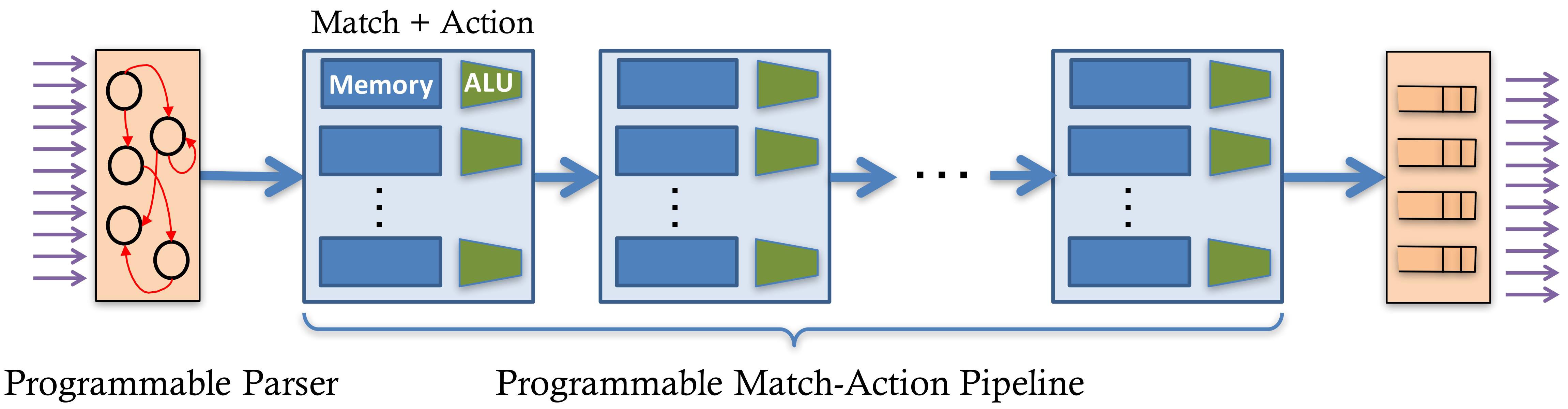
PISA: Protocol Independent Switch Architecture

- **Programmable Parser**
  - Converts packet data into metadata
- **Programmable Mach-Action Pipeline**
  - Operate on metadata and update memory states

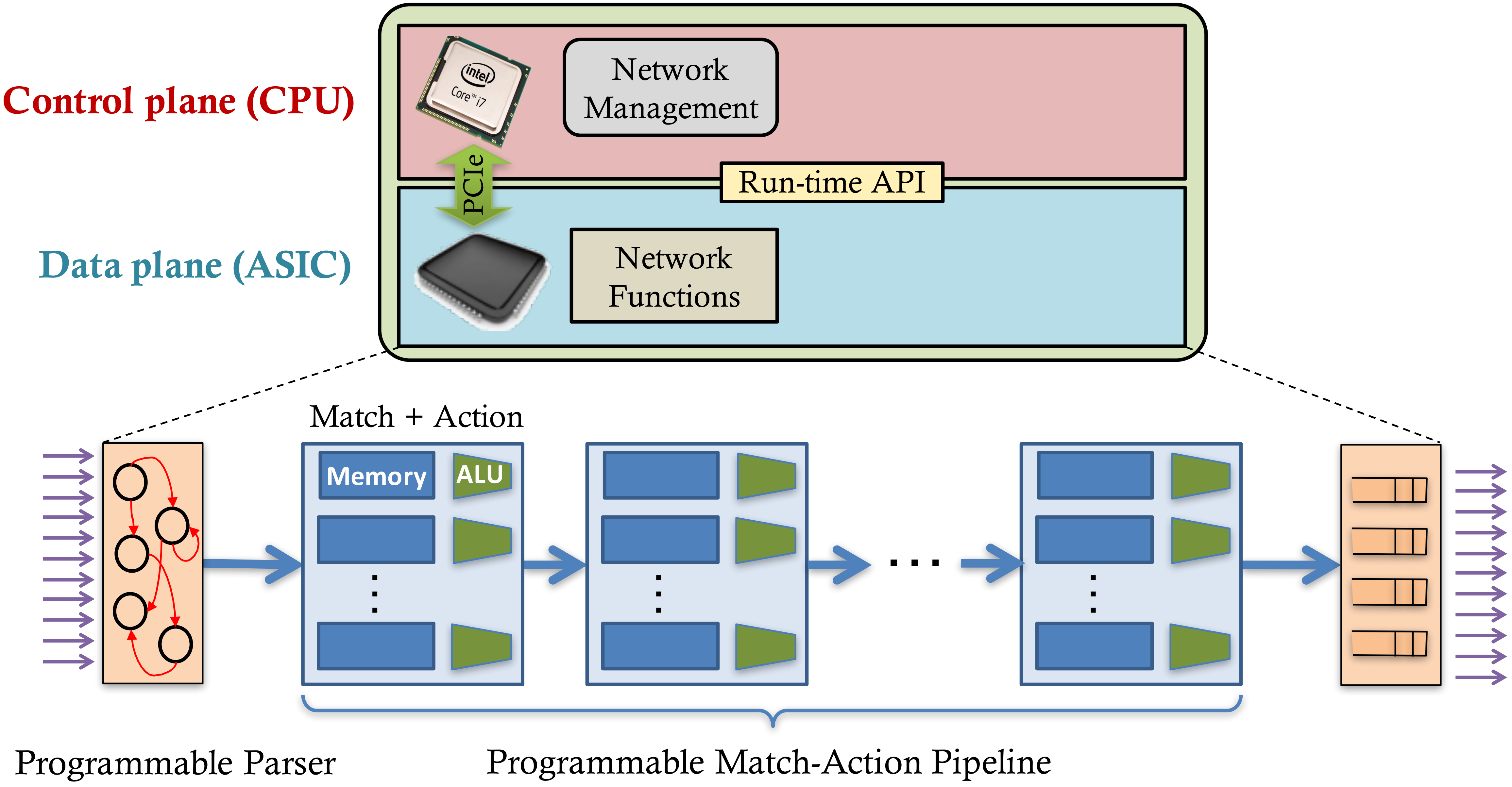


PISA: Protocol Independent Switch Architecture

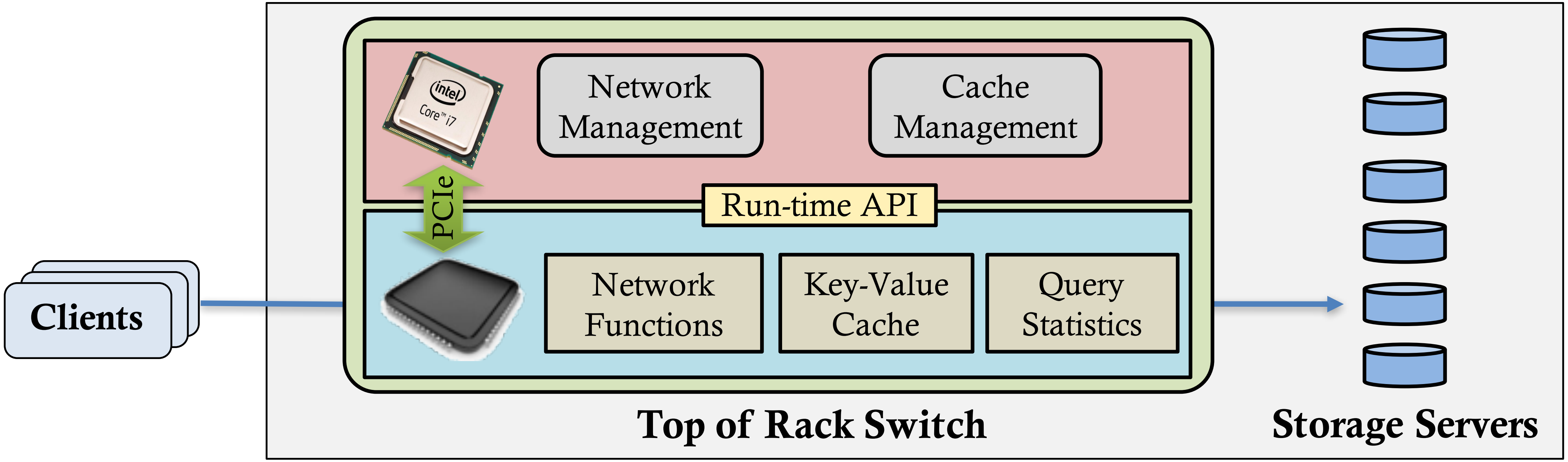
- **Programmable Parser**
  - Parse custom key-value fields in the packet
- **Programmable Mach-Action Pipeline**
  - Read and update key-value data
  - Provide query statistics for cache updates



PISA: Protocol Independent Switch Architecture

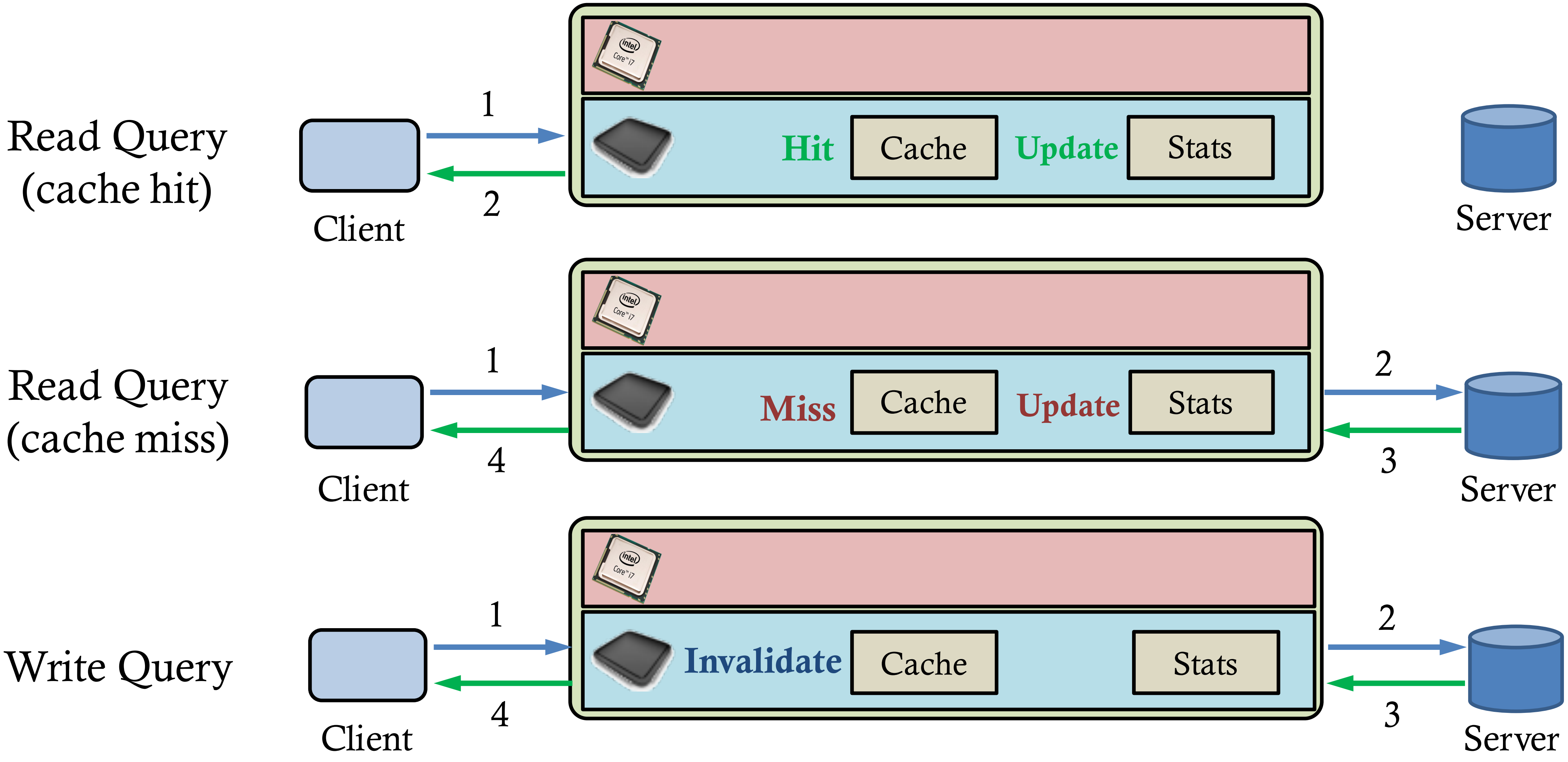


NetCache rack-scale architecture



- **Switch data plane**
  - Key-value store to serve queries for cached keys
  - Query statistics to enable efficient cache updates
- **Switch control plane**
  - Insert hot items into the cache and evict less popular items
  - Manage memory allocation for on-chip key-value store

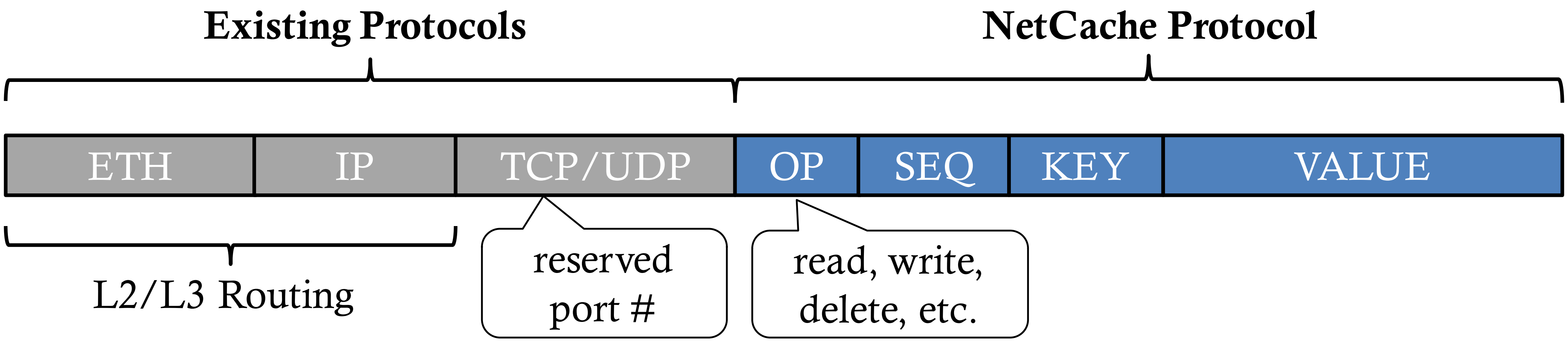
Data plane query handling



Key-value caching in network ASIC at line rate

- ❑ How to identify application-level packet fields ?
- ❑ How to store and serve variable-length data ?
- ❑ How to efficiently keep the cache up-to-date ?

NetCache Packet Format



- Application-layer protocol: compatible with existing L2-L4 layers
- Only the top of rack switch needs to parse NetCache fields

Key-value caching in network ASIC at line rate

- ❑ How to identify application-level packet fields ?
- ❑ How to store and serve variable-length data ?
- ❑ How to efficiently keep the cache up-to-date ?

Key-value store using register array in network ASIC

Match	pkt.key == A	pkt.key == B
Action	**process_array**(0)	**process_array**(1)

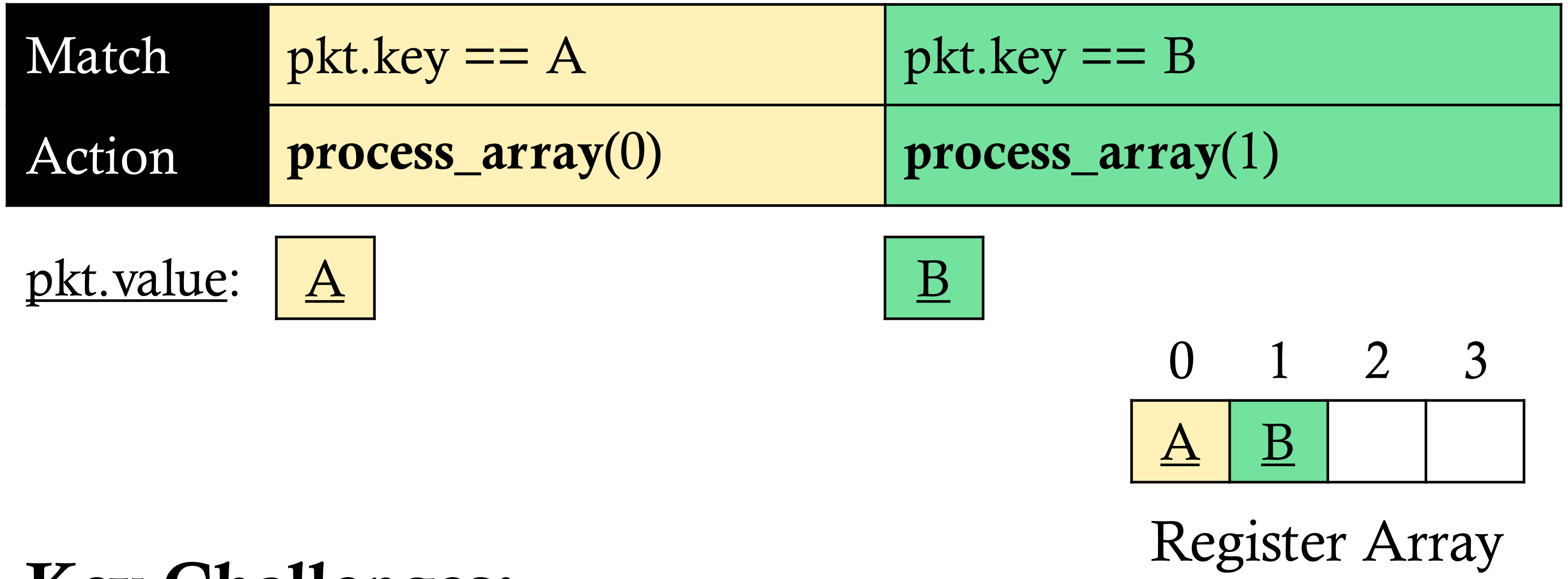
pkt.value: A B

```
action process_array(idx):
  if pkt.op == read:
    pkt.value ⟵ array[idx]
  elif pkt.op == cache_update:
    array[idx] ⟵ pkt.value
```

0	1	2	3
A	B		

Register Array

Variable-length key-value store in network ASIC?



Key Challenges:

- No loop or string due to strict timing requirements
- Need to minimize hardware resources consumption
  - Number of table entries
  - Size of action data from each entry
  - Size of intermediate metadata across tables

Combine outputs from multiple arrays

Bitmap indicates arrays that store the key's value

Index indicates slots in the arrays to get the value

Minimal hardware resource overhead

Lookup Table

Match	pkt.key == A
Action	bitmap = **111** index = **0**

pkt.value:

A0	A1	A2

Value Table 0

Match	bitmap[0] == 1
Action	**process_array_0** (index)

→ Register Array 0

0	1	2	3
A0			

Value Table 1

Match	bitmap[1] == 1
Action	**process_array_1** (index)

→ Register Array 1

A1			

Value Table 2

Match	bitmap[2] == 1
Action	**process_array_2** (index)

→ Register Array 2

A2			

Combine outputs from multiple arrays

Lookup Table

Match	pkt.key == A	pkt.key == B	pkt.key == C	pkt.key == D
Action	bitmap = **111** index = **0**	bitmap = **110** index = **1**	bitmap = **010** index = **2**	bitmap = **101** index = **2**
pkt.value:	A0 A1 A2	B0 B1	C0	D0 D1

Value Table 0

Match	bitmap[0] == 1
Action	**process_array_0** (index)

Register Array 0

0	1	2	3
A0	B0	D0	

Value Table 1

Match	bitmap[1] == 1
Action	**process_array_1** (index)

Register Array 1

A1	B1	C0	

Value Table 2

Match	bitmap[2] == 1
Action	**process_array_2** (index)

Register Array 2

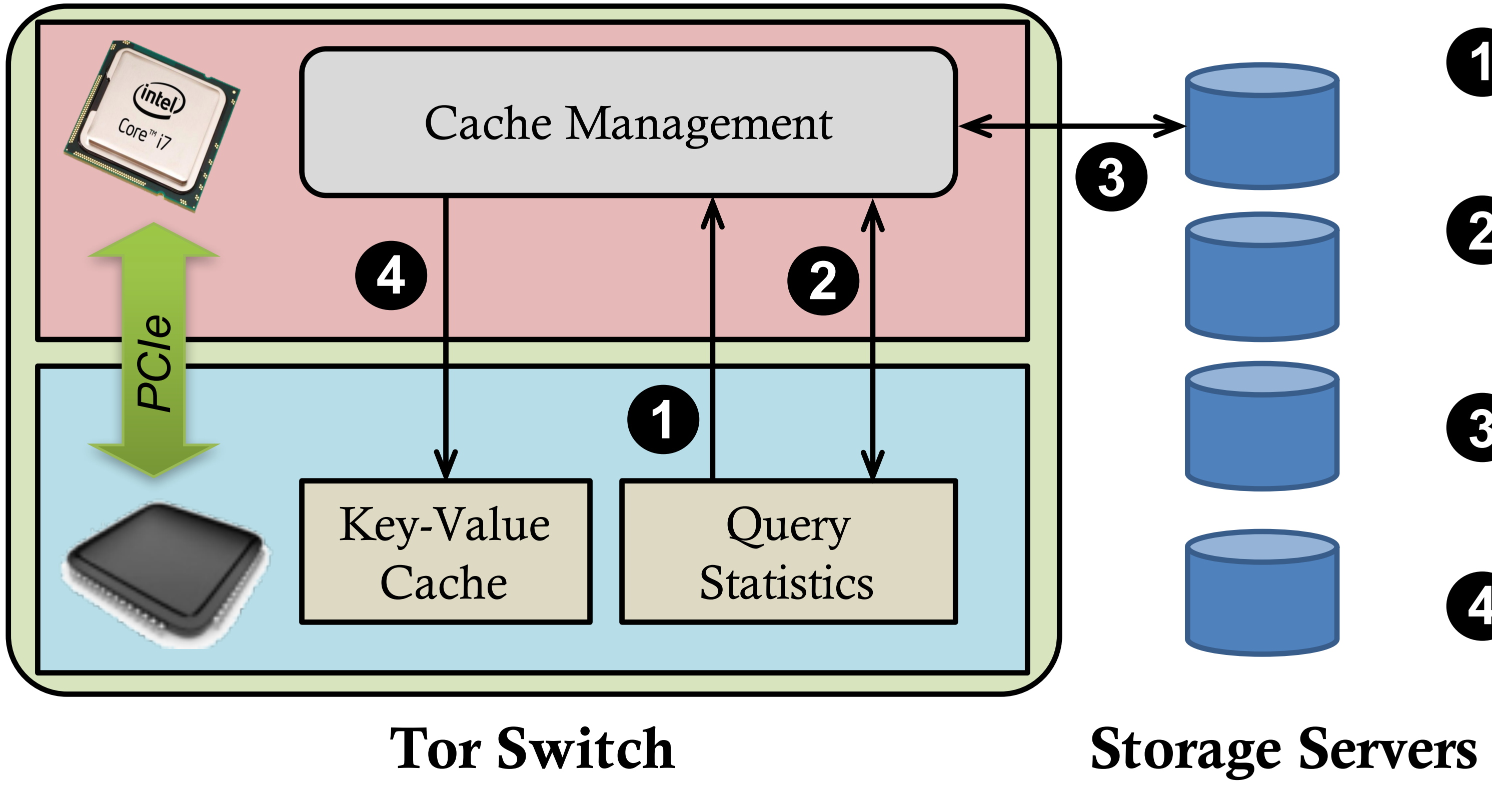
A2		D1	

Key-value caching in network ASIC at line rate

- ❑ How to identify application-level packet fields ?
- ❑ How to store and serve variable-length data ?
- ❑ How to efficiently keep the cache up-to-date ?

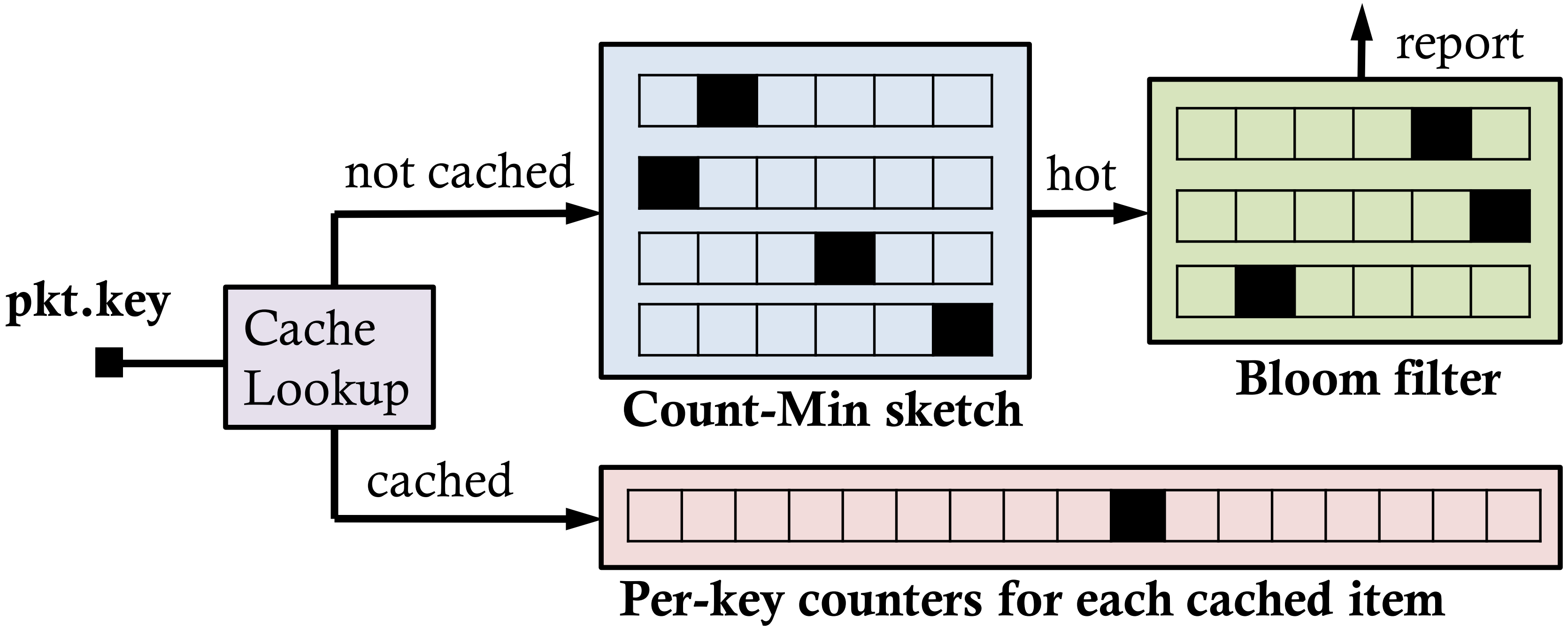
Cache insertion and eviction

- Challenge: cache the hottest O($N \log N$) items with **limited insertion rate**
- Goal: react quickly and effectively to workload changes with **minimal updates**



1. Data plane reports hot keys
2. Control plane compares loads of new hot and sampled cached keys
3. Control plane fetches values for keys to be inserted to the cache
4. Control plane inserts and evicts keys

Query statistics in the data plane



- Cached key: per-key counter array
- Uncached key
  - Count-Min sketch: report new hot keys
  - Bloom filter: remove duplicated hot key reports

Evaluation

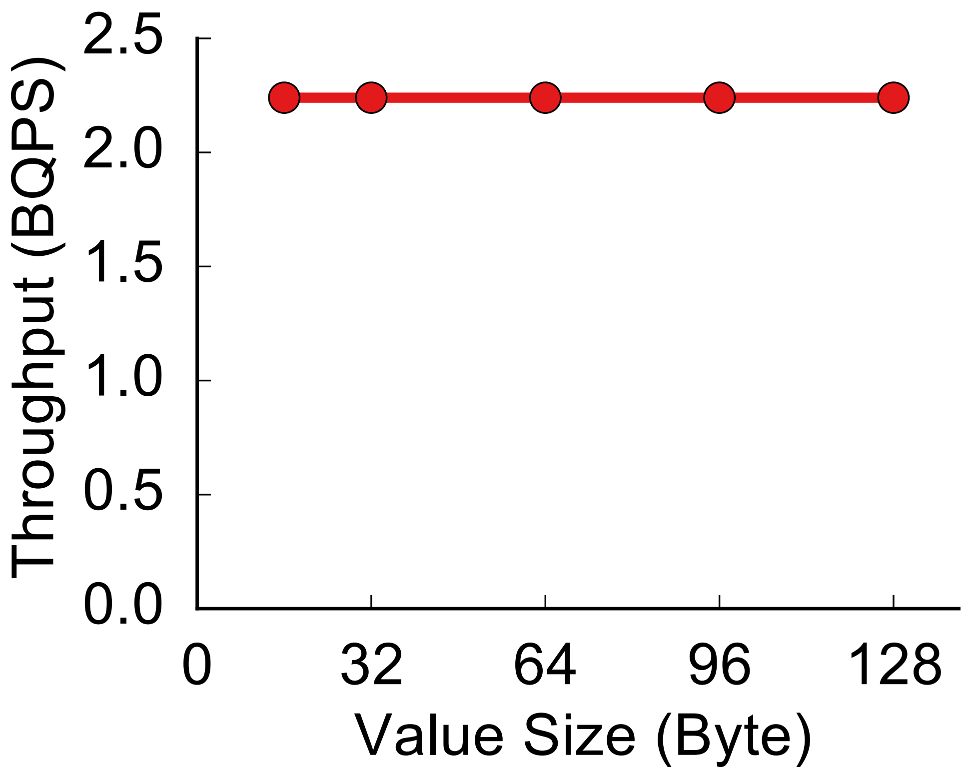
- Can NetCache run on programmable switches at line rate?
- Can NetCache provide significant overall performance improvements?
- Can NetCache efficiently handle workload dynamics?

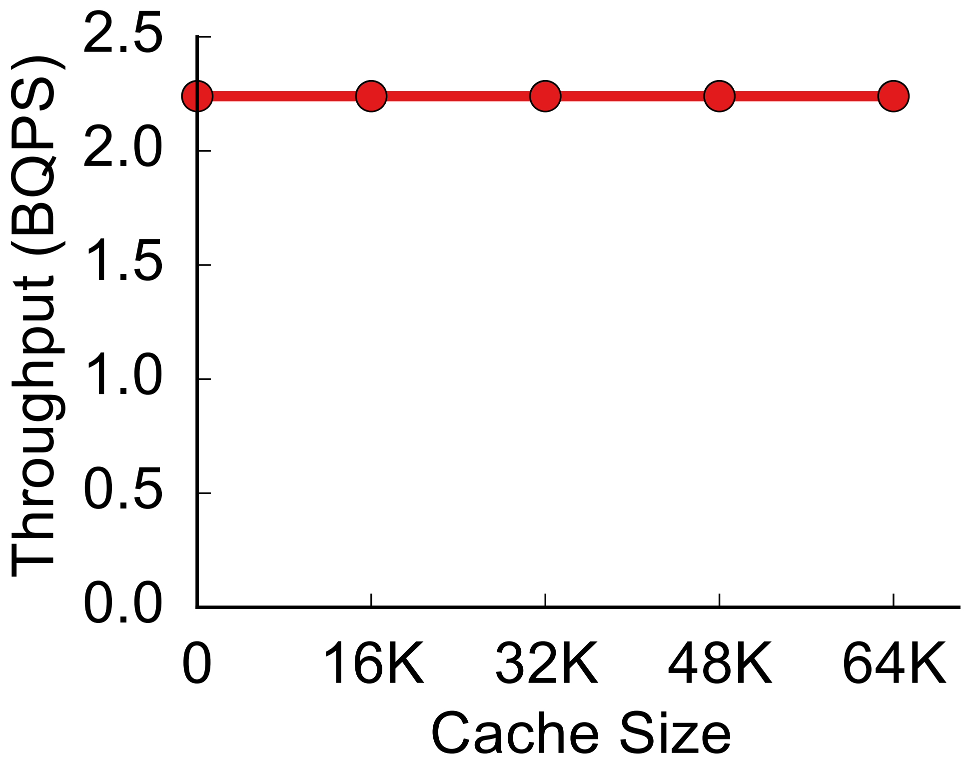
Prototype implementation and experimental setup

- Switch
  - P4 program (~2K LOC)
  - Routing: basic L2/L3 routing
  - Key-value cache: **64K items** with **16-byte key** and up to **128-byte value**
  - Evaluation platform: one 6.5Tbps Barefoot Tofino switch
- Server
  - 16-core Intel Xeon E5-2630, 128 GB memory, 40Gbps Intel XL710 NIC
  - TommyDS for in-memory key-value store
  - Throughput: **10 MQPS**; Latency: **7 us**

The “boring life” of a NetCache switch

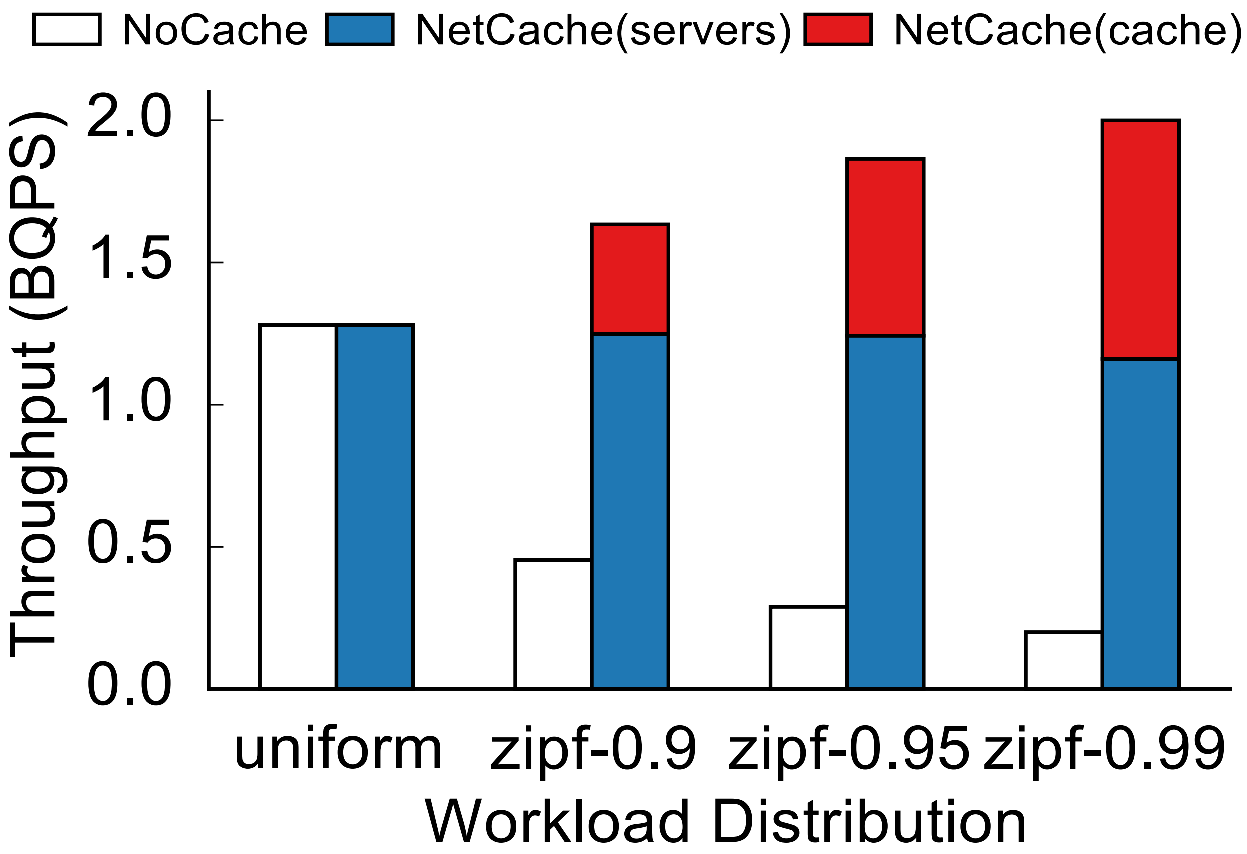
Single switch benchmark





And its "not so boring" benefits

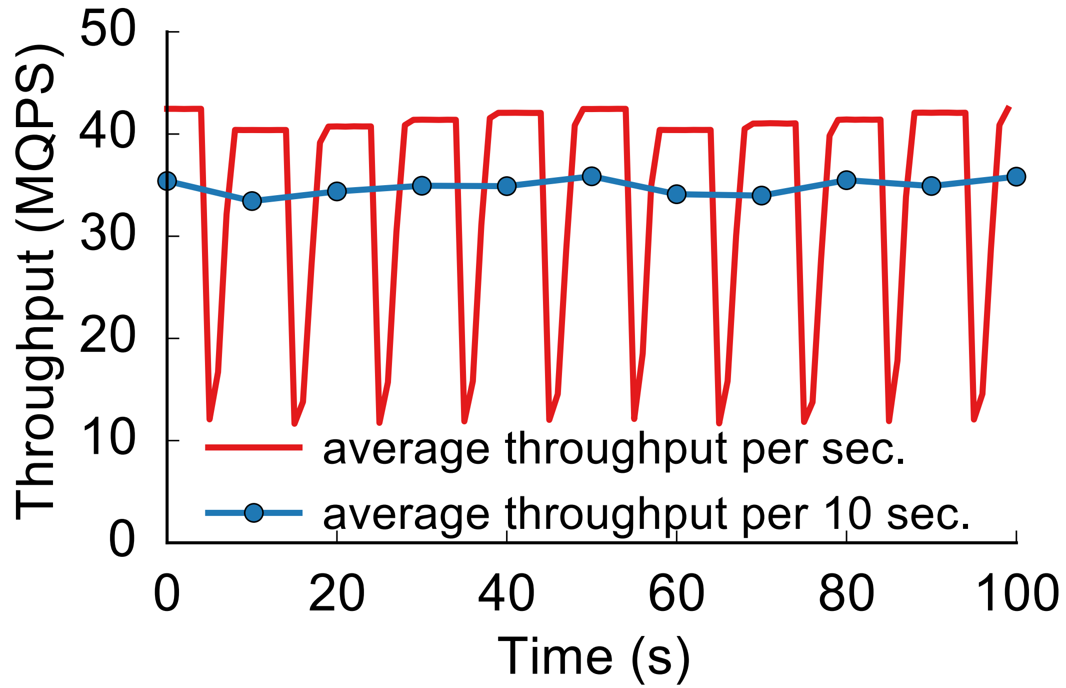
1 switch + 128 storage servers



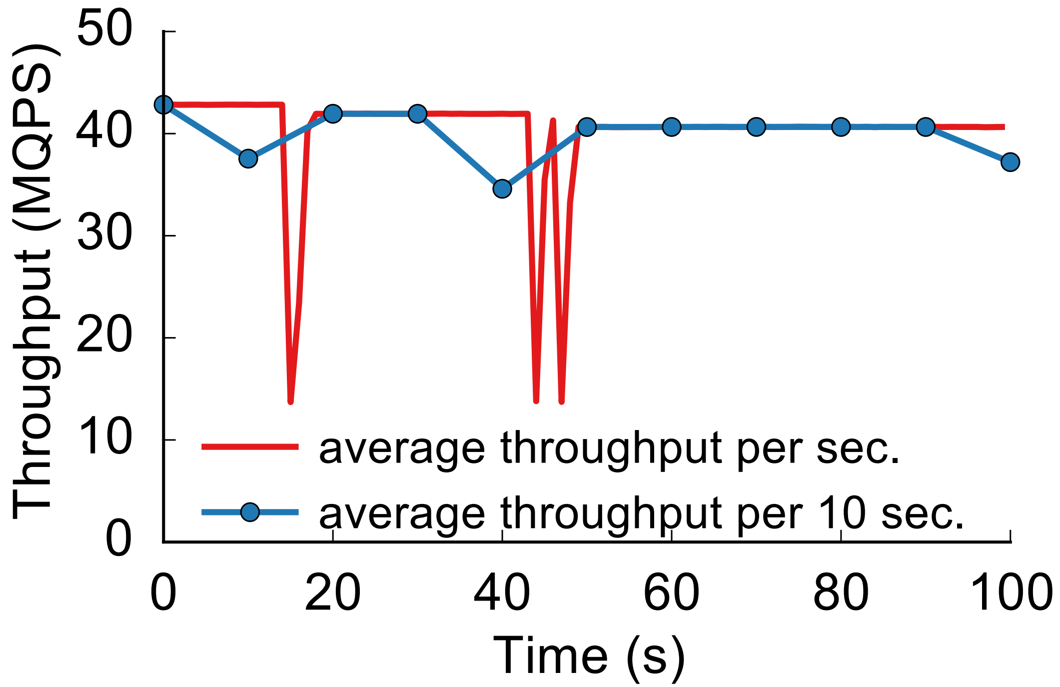
3-10x throughput improvements

Impact of workload dynamics

hot-in workload (radical change)



random workload (moderate change)



Quickly and effectively reacts to a wide range of workload dynamics.

(2 physical servers to emulate 128 storage servers, performance scaled down by 64x)

NetCache is a **rack-scale key-value store** that leverages **in-network data plane caching** to achieve

billions QPS throughput & **~10 μs latency**

even under

highly-skewed & **rapidly-changing**

workloads.

Conclusion: programmable switches beyond networking

- **Cloud datacenters are moving towards …**
  - Rack-scale disaggregated architecture
  - In-memory storage systems
  - Task scheduling at microseconds granularity
- **Programmable switches can do more than packet forwarding**
  - Cross-layer co-design of compute, storage and network stacks
  - Switches help on caching, coordination, scheduling, etc.
- **New generations of systems enabled by programmable switches ☺**