

PISA-Based Application Acceleration for IO-Intensive Workloads

Xin Jin



Joint work with Xiaozhou Li, Haoyu Zhang, Nate Foster, Jeongkeun Lee, Robert Soulé,
Changhoon Kim, and Ion Stoica

The revolution in networking

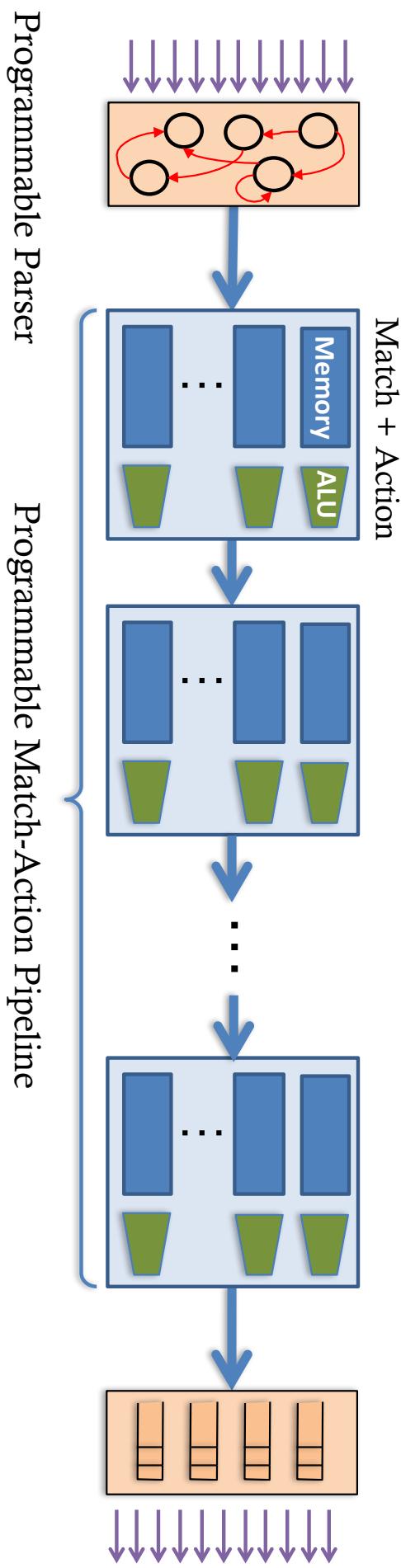


Fixed-function switch

Programmable switch

PISA: Protocol Independent Switch Architecture

- Programmable Parser
 - Convert packet data into metadata
- Programmable Match-Action Pipeline
 - Operate on metadata and update memory state



Programmable switch data planes enable many innovations

Sonata [SIGCOMM'18]

Network Telemetry

Dapper [SOSR'17]

TCP Diagnosis

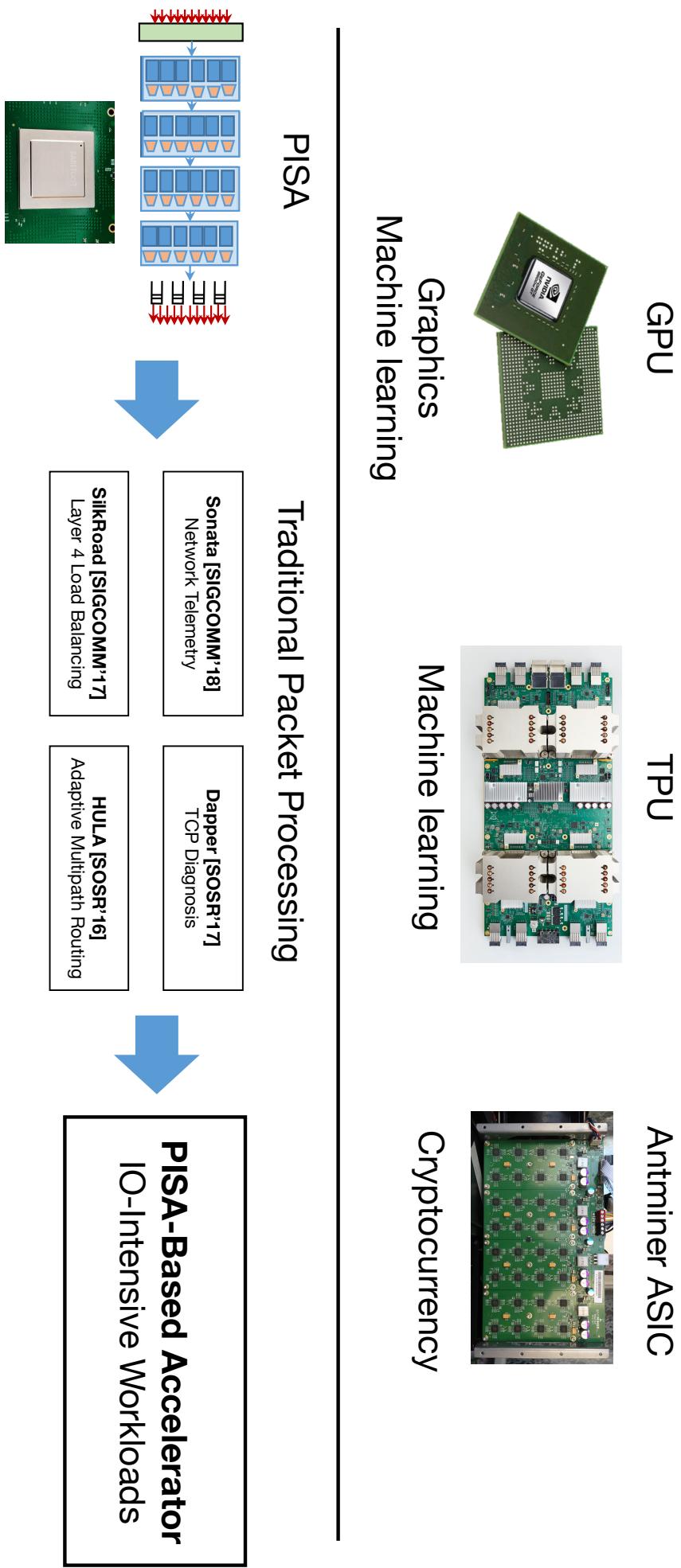
SilkRoad [SIGCOMM'17]

Layer 4 Load Balancing

HULA [SOSR'16]

Adaptive Multipath Routing

The ending of the Moore's Law, and the rise of domain specific processors...



PISA switches as domain specific accelerators for **IO-intensive** workloads

- **NetCache** [SOSP'17]: balancing key-value stores with PISA-based caching
- **NetChain** [NSDI'18, best paper award]: fast coordination with PISA-based chain replication

Joint work with Xiaozhou Li, Haoyu Zhang, Nate Foster, Jeongkeun Lee, Robert Soulé, Changhoon Kim, and Ion Stoica

NetCache is a rack-scale key-value store that leverages
PISA-based caching to achieve

billions QPS throughput

&

$\sim 10 \mu\text{s}$ latency

even under

highly-skewed

&

rapidly-changing

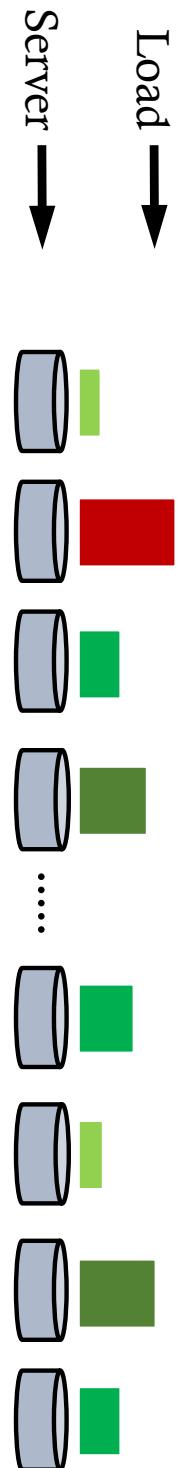
workloads.

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**

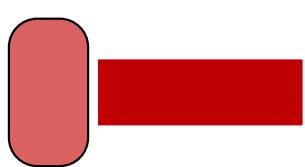
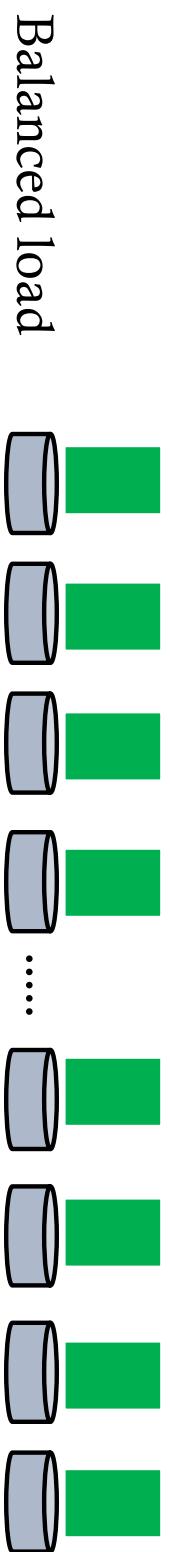
Key challenge: highly-skewed and rapidly-changing workloads

low throughput
& high tail latency



Opportunity: fast, small cache for load balancing

Cache absorbs hottest queries

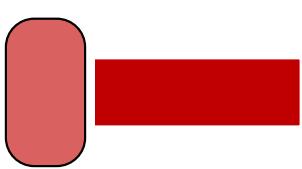


Opportunity: fast, small cache for load balancing

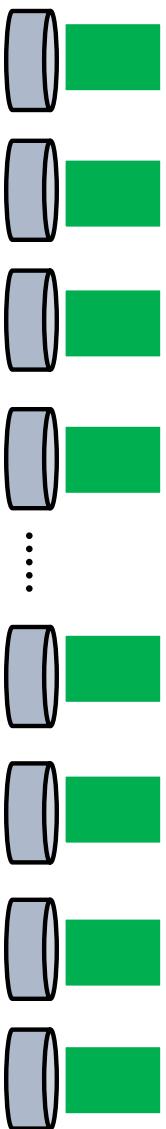
[B. Fan et al. SoCC'11, X. Li et al. NSDI'16]

Cache $O(N \log N)$ hottest items

E.g., 10,000 hot objects



N : # of servers

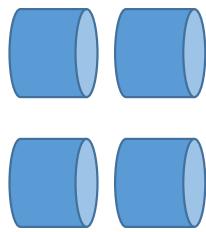


E.g., 100 backends with 100 billions items

Requirement: cache throughput \geq 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

storage layer



in-memory

each: O(10) MQPS

total: O(1) BQPS



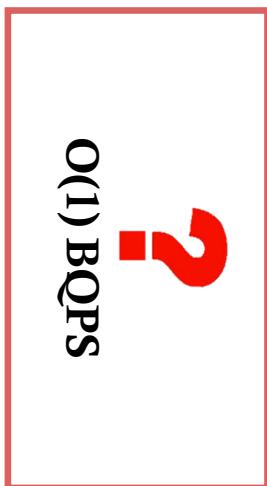
in-memory

O(10) MQPS

cache layer

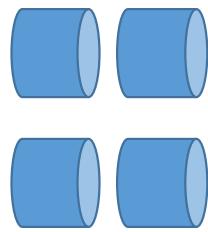


O(1) BQPS



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

PISA-based
O(1) BQPS

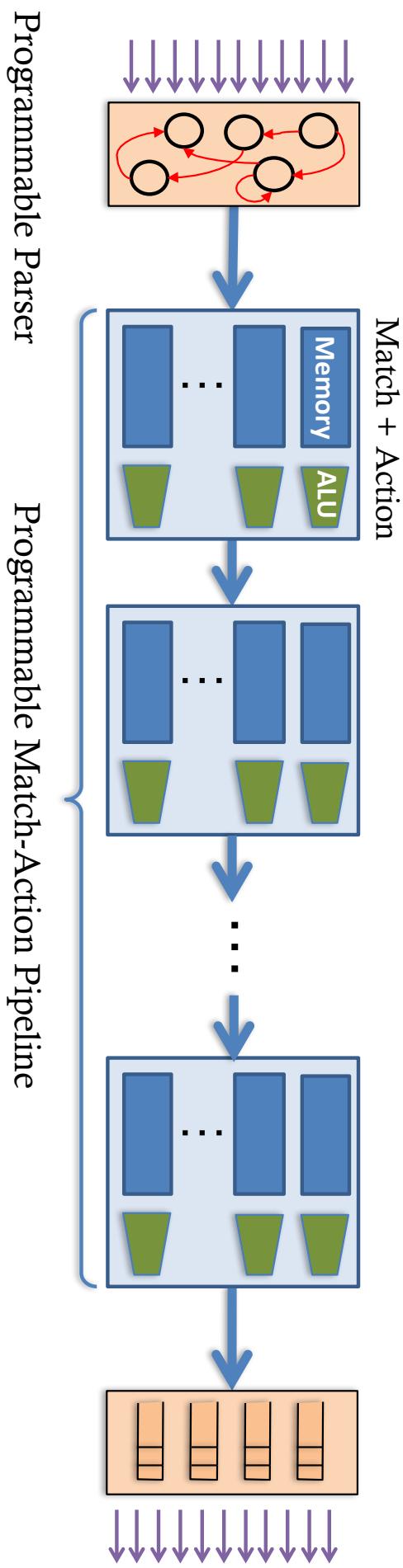
Small on-chip memory?
Only cache **O(N log N)** small items

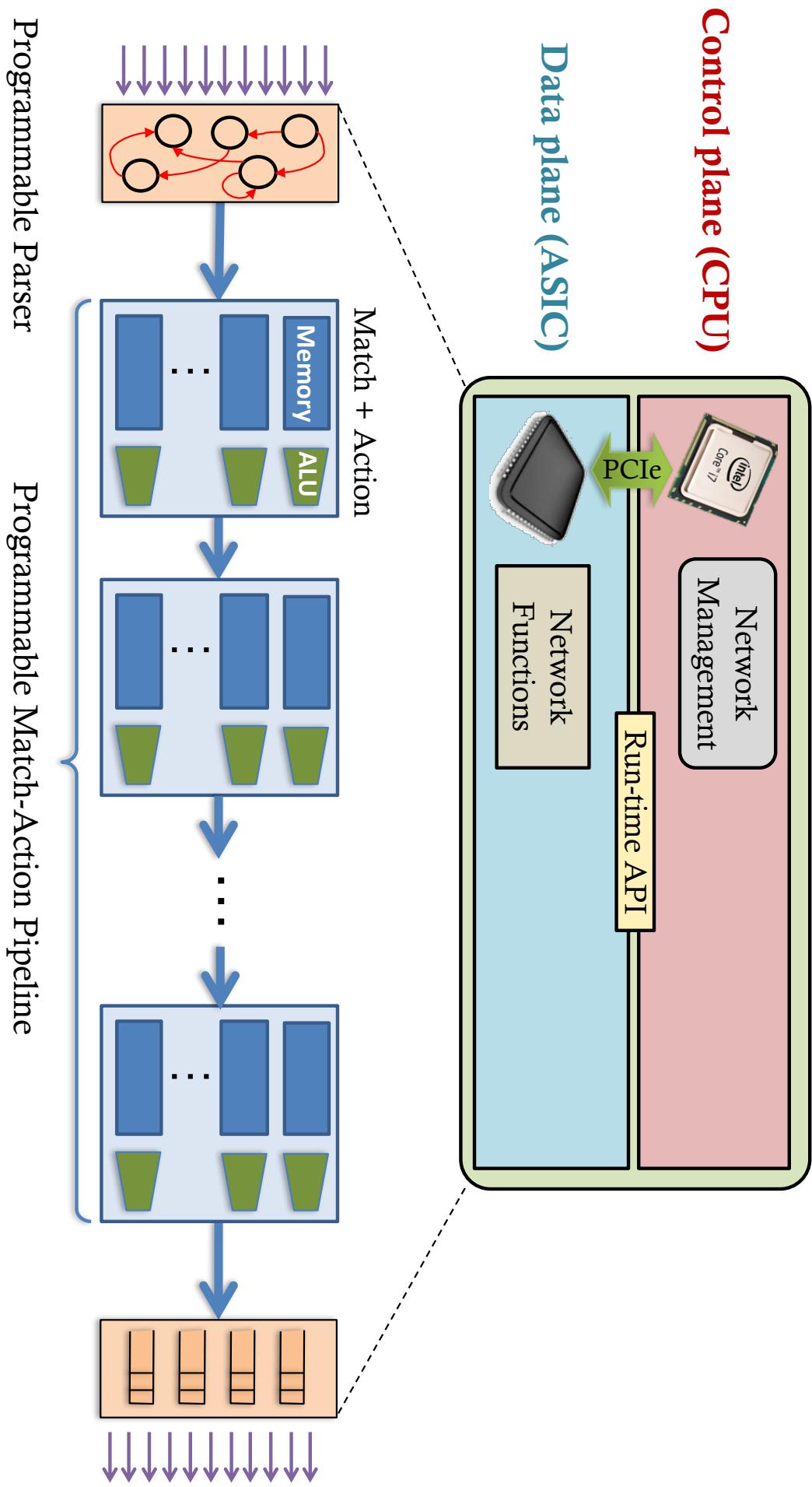
Key-value caching in network ASIC at line rate ?!

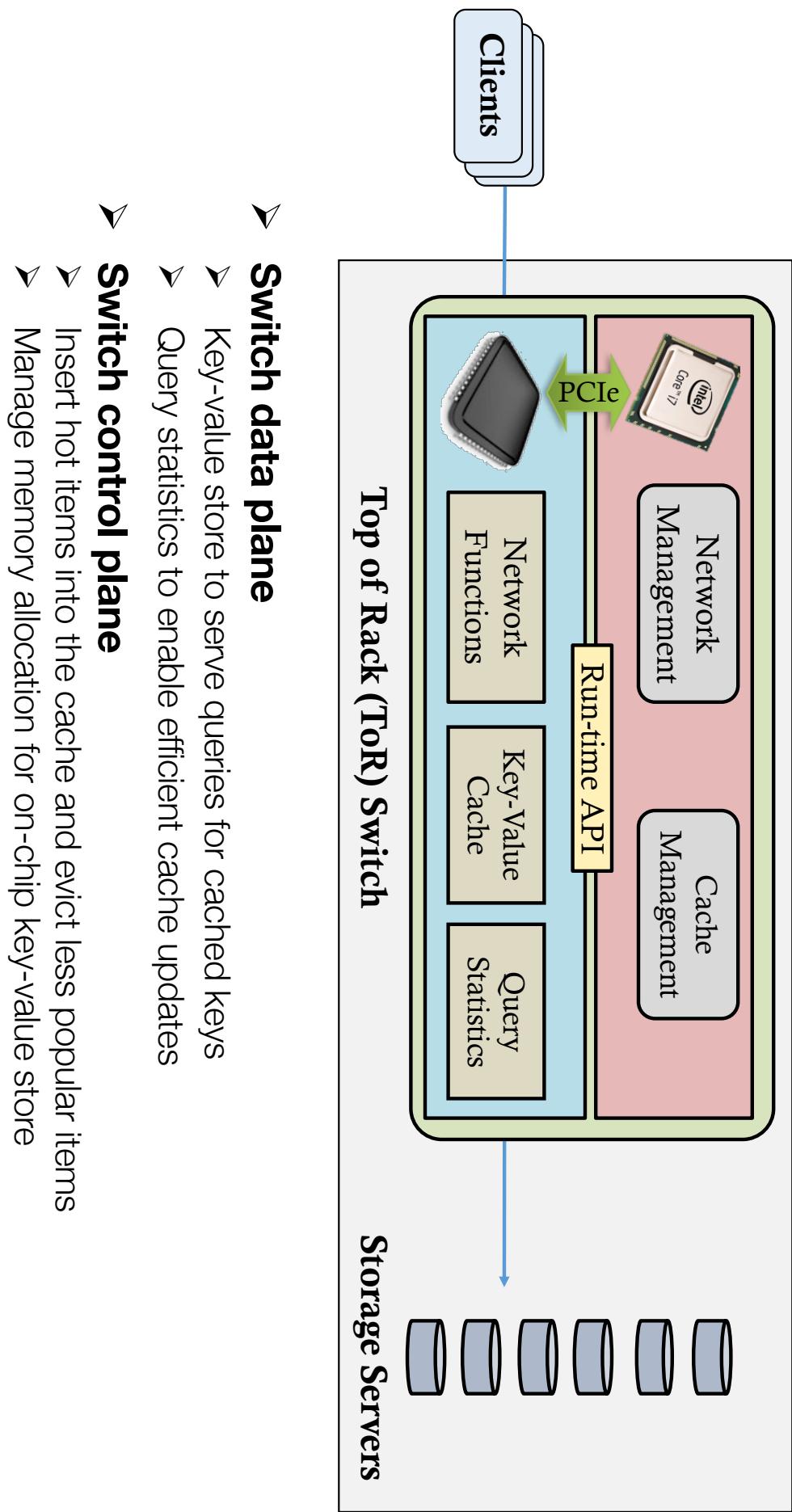
- How to identify application-level packet fields ?
- How to store and serve variable-length data ?
- How to efficiently keep the cache up-to-date ?

PISA: Protocol Independent Switch Architecture

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





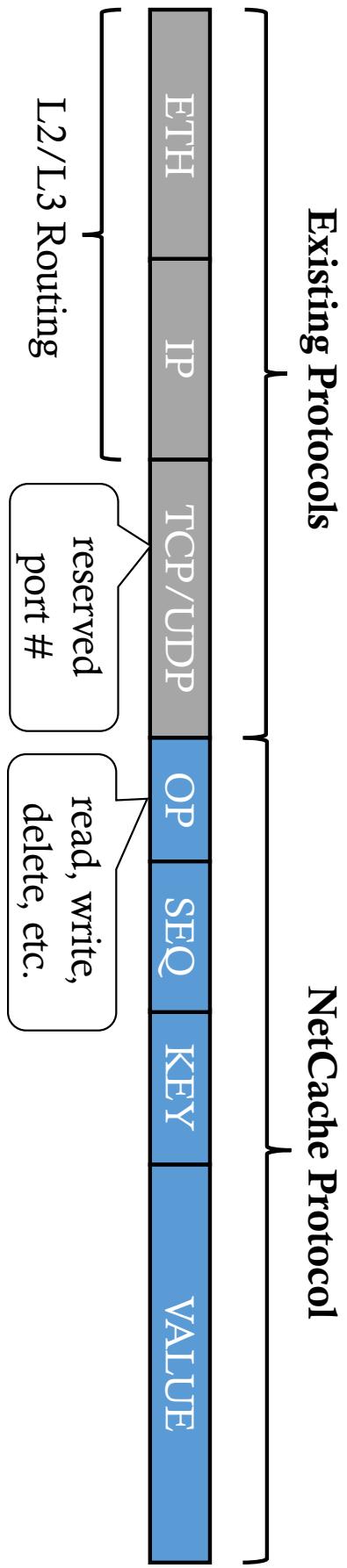


Key-value caching in network ASIC at line rate

→ □ 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 arrays

Match-Action Table Register Array (RA)

Match	Action	0
Key = X	Read RA[0]	1
Key = Y	Read RA[5]	2
Key = Z	Read RA[2]	3
Default	Drop()	4
		5

Key Challenges:

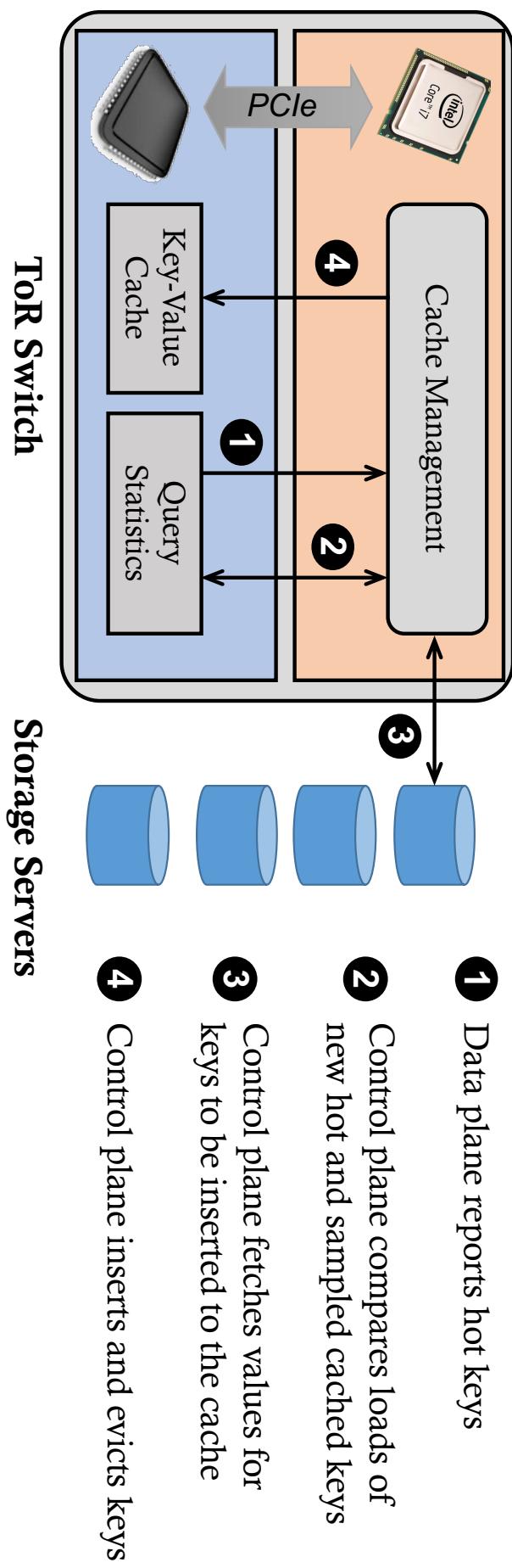
- No loop or string due to strict timing requirements
- Need to minimize hardware resources consumption
 - Number of table entries
 - Size of action data from each entry
 - Size of intermediate metadata across tables

Key-value caching in network ASIC at line rate

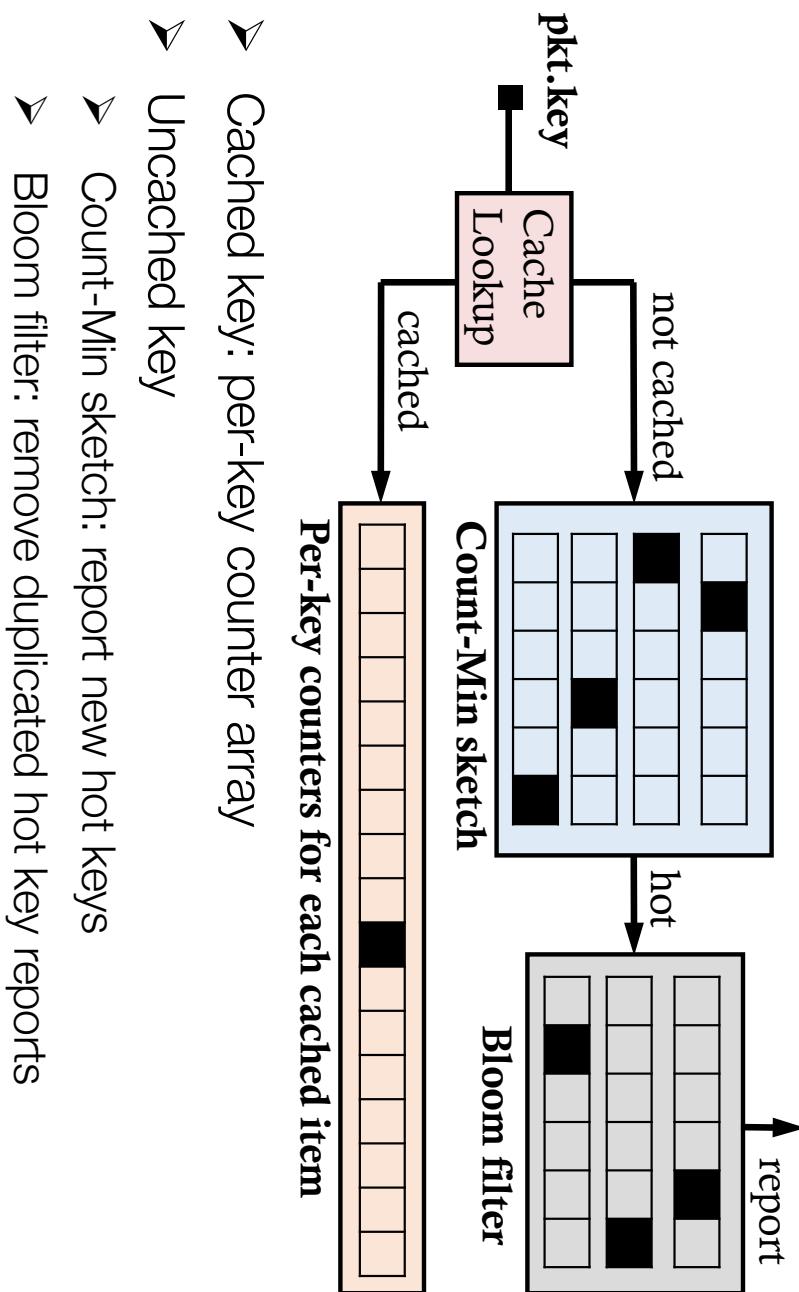
- How to identify application-level packet fields ?
- How to store and serve variable-length data ?
- □ How to efficiently keep the cache up-to-date ?

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**



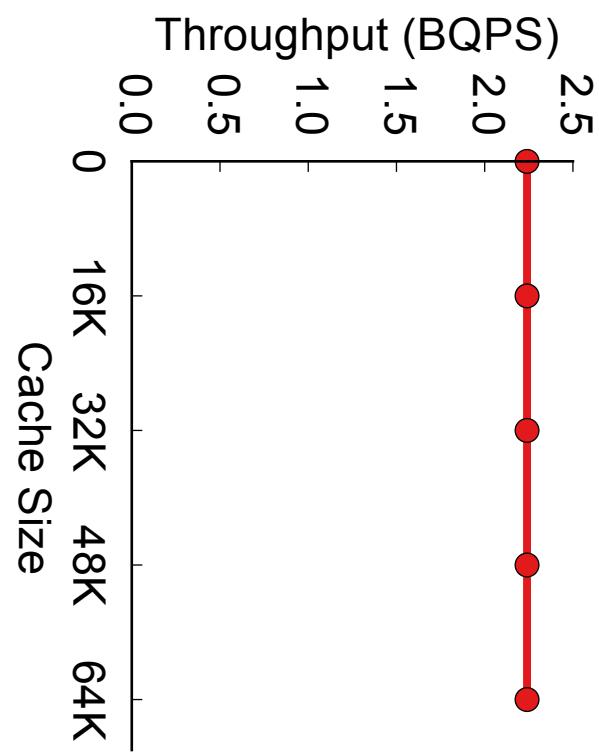
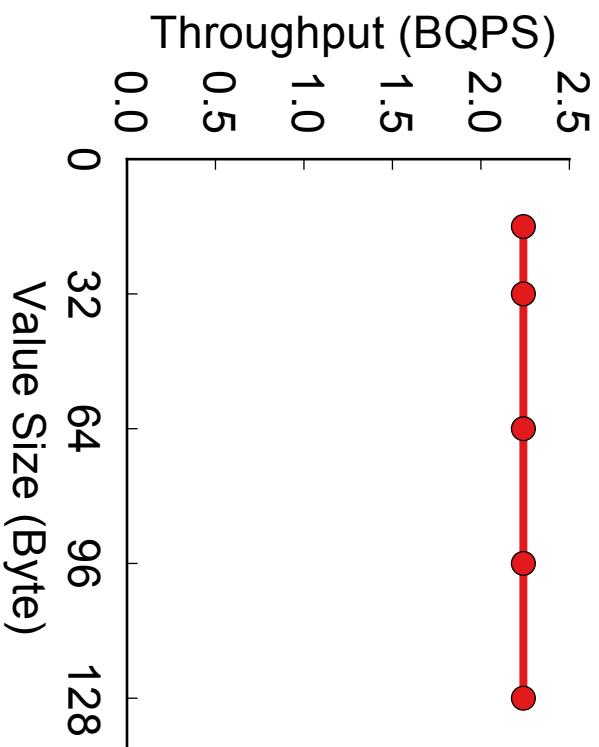
Query statistics in the data plane



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

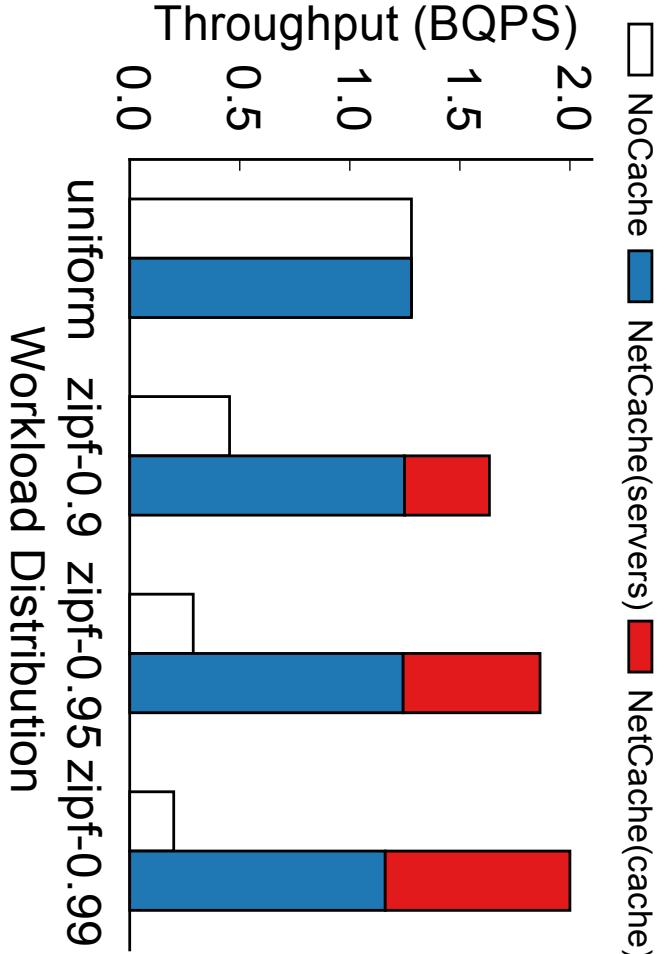
The “boring life” Of a NetCache switch

Single switch benchmark



And its “not so boring” benefits

1 switch + 128 storage servers



3-10x throughput improvements

NetCache is a rack-scale key-value store that leverages
PISA-based caching to achieve

billions QPS throughput

&

$\sim 10 \mu\text{s}$ latency

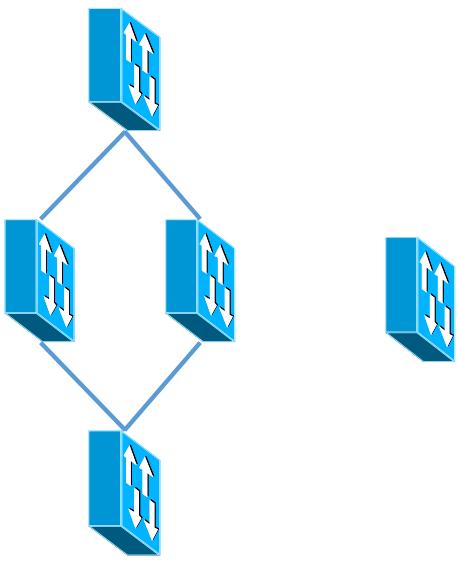
even under

highly-skewed

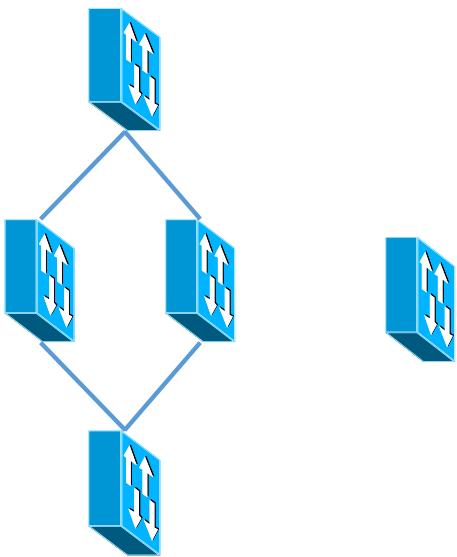
&

rapidly-changing

workloads.



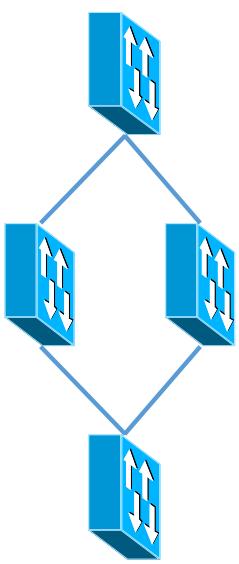
NetCache: lightning fast key-value cache
enabled by PISA switches



NetCache: lightning fast key-value cache
enabled by PISA switches

NetChain: lightning fast coordination
enabled by PISA switches

Conventional wisdom: avoid coordination

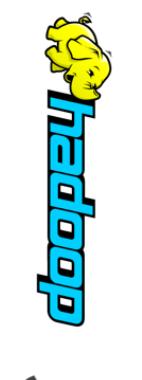


NetChain: lightning fast coordination
enabled by PISA switches

Open the door to rethink distributed systems design

Coordination services: fundamental building block of the cloud

Applications



Coordination Service

Google Chubby



etcd

Provide critical coordination functionalities

Applications



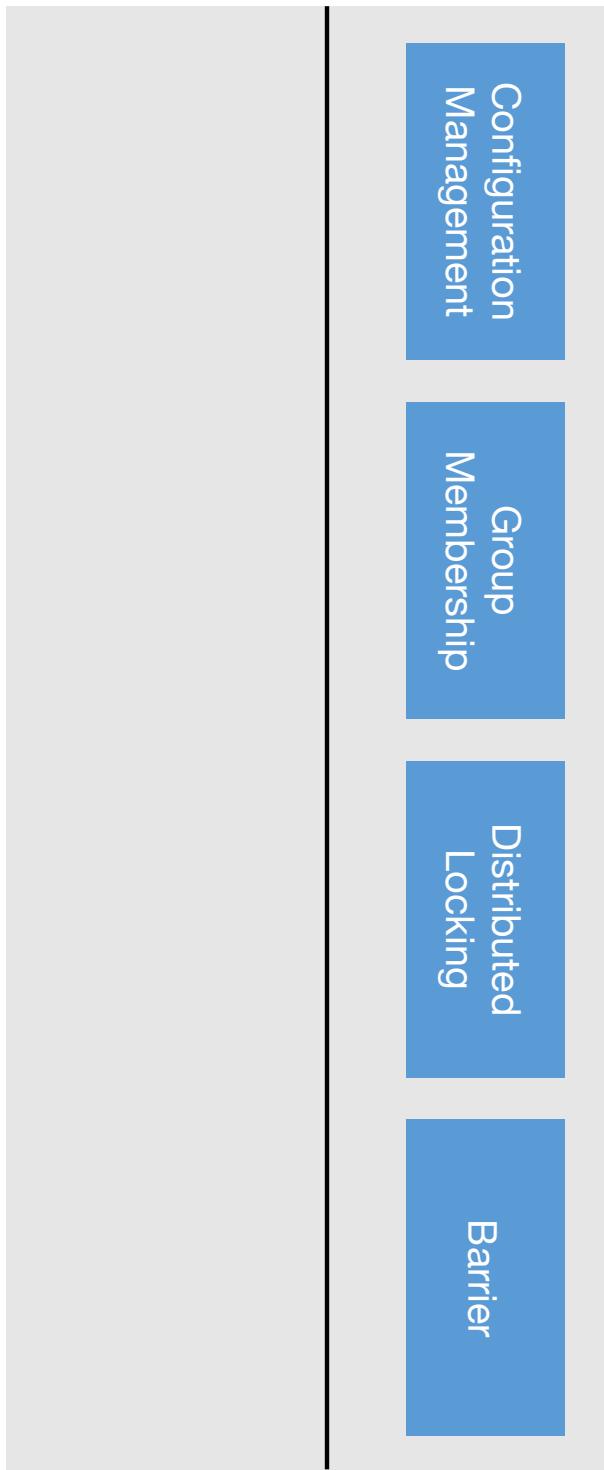
Coordination
Service

Configuration Management

Group Membership

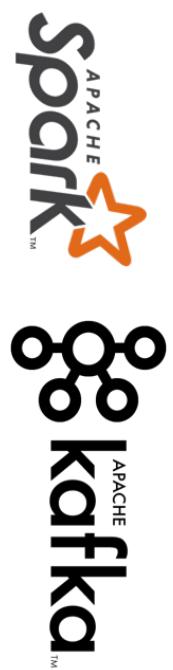
Distributed Locking

Barrier



The core is a strongly-consistent, fault-tolerant key-value store

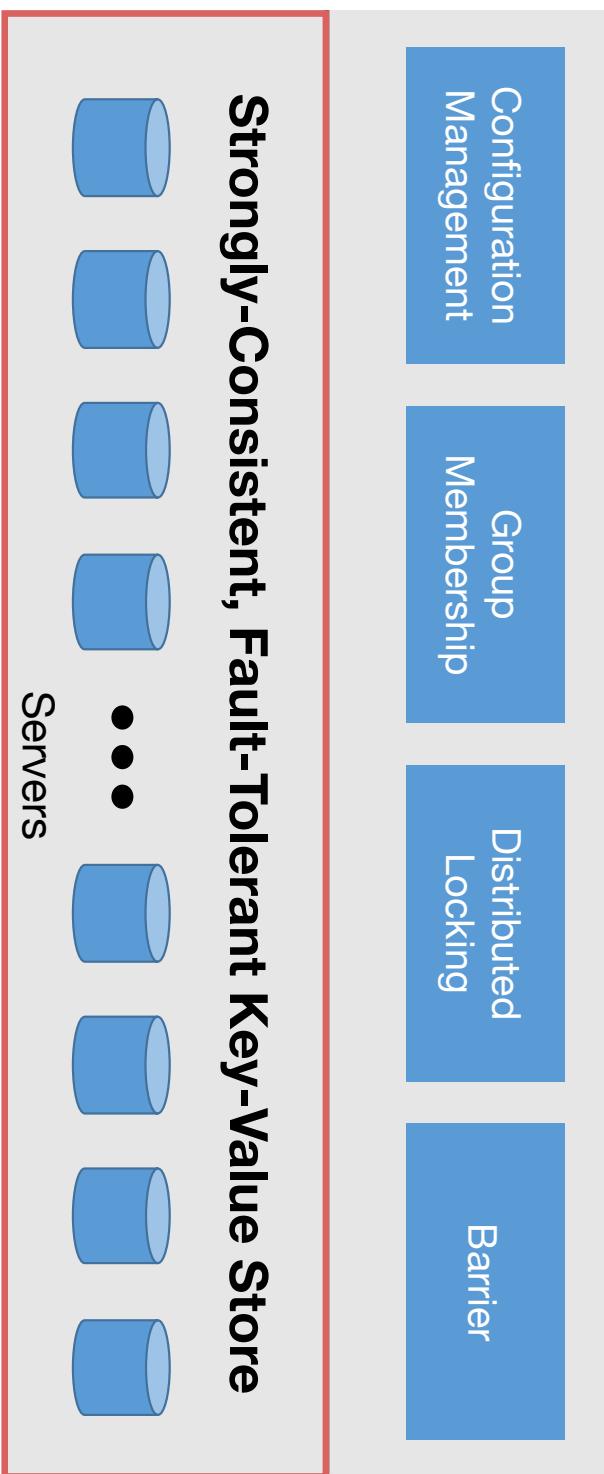
Applications



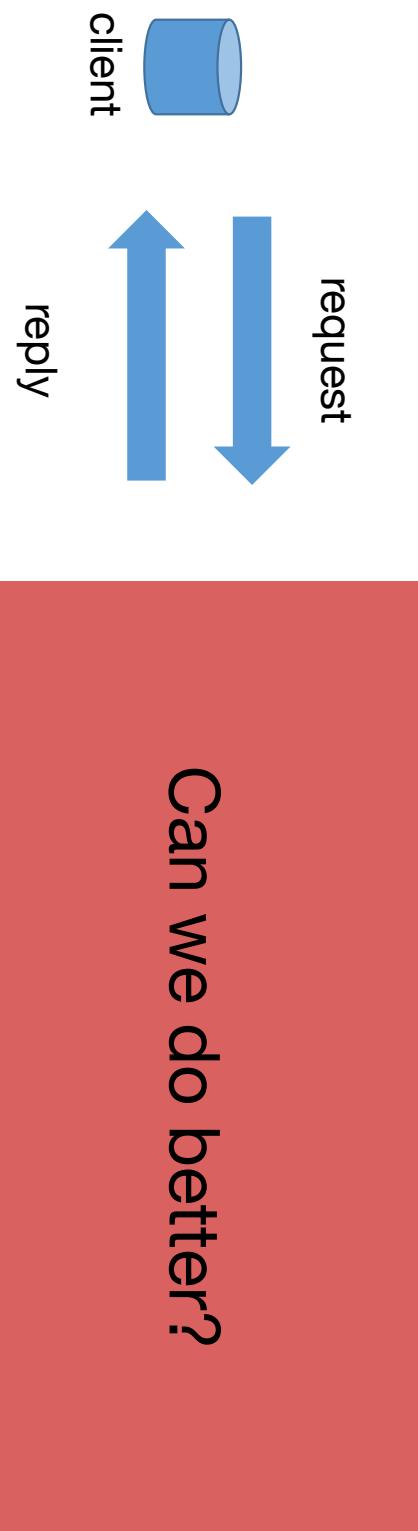
Coordination Service

Strongly-Consistent, Fault-Tolerant Key-Value Store

This Talk

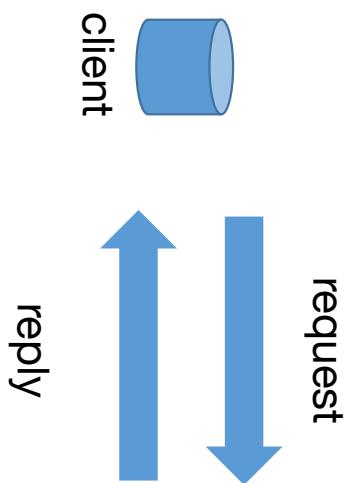


Workflow of coordination services



- Throughput: at most server NIC throughput
- Latency: at least one RTT, typically a few RTTs

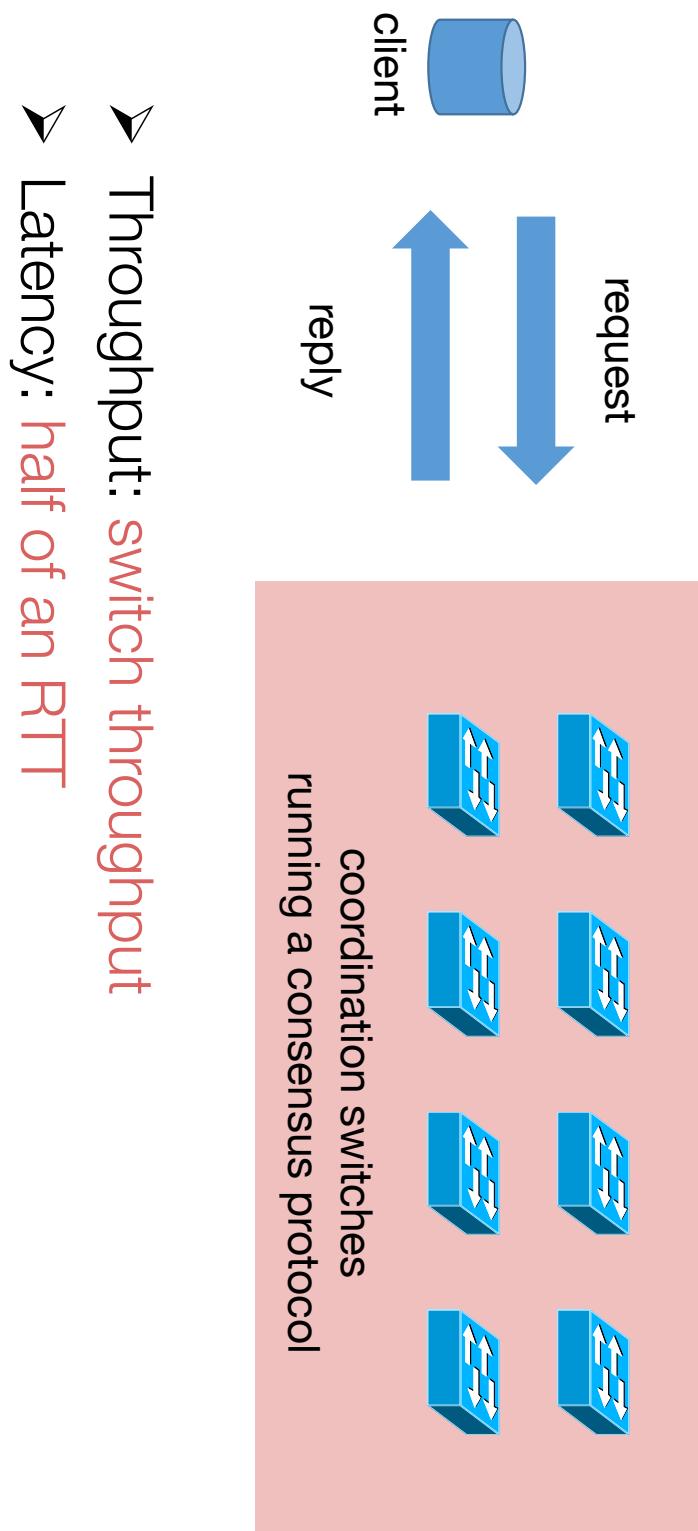
Opportunity: PI-SA-based coordination



Distributed coordination is
IO-intensive,
not computation-intensive.

Server	Switch
Example	[NetBricks, OSDI'16]
Packets per second	30 million
Bandwidth	10-100 Gbps
Processing delay	10-100 us
	< 1 us

Opportunity: PISA-based coordination



Design goals for coordination services

- High throughput
 - Low latency
 - Strong consistency
 - Fault tolerance
- Directly from
high-performance switches

How?

Design goals for coordination services

- High throughput
- Low latency

Directly from
high-performance switches

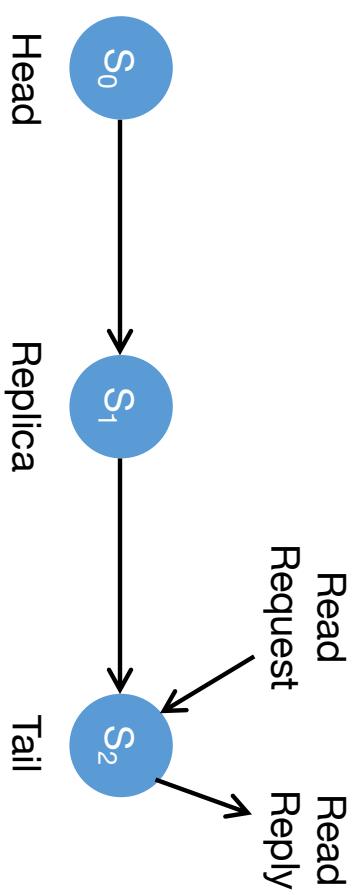
- Strong consistency

Chain replication with PISA switches

- Fault tolerance

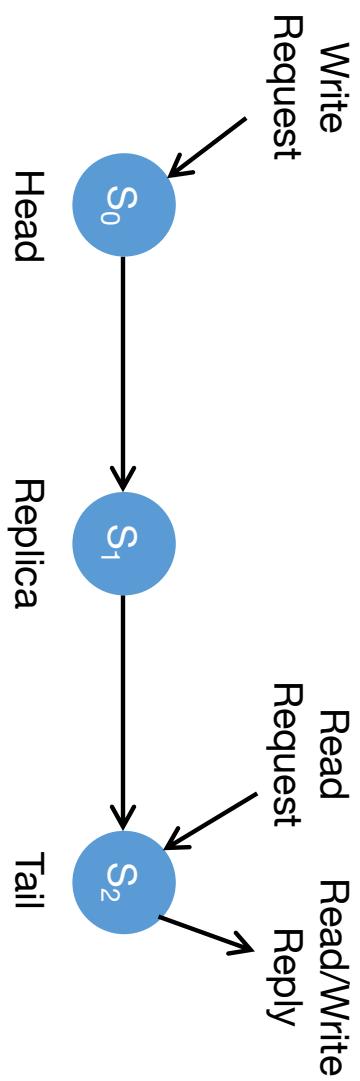


What is chain replication



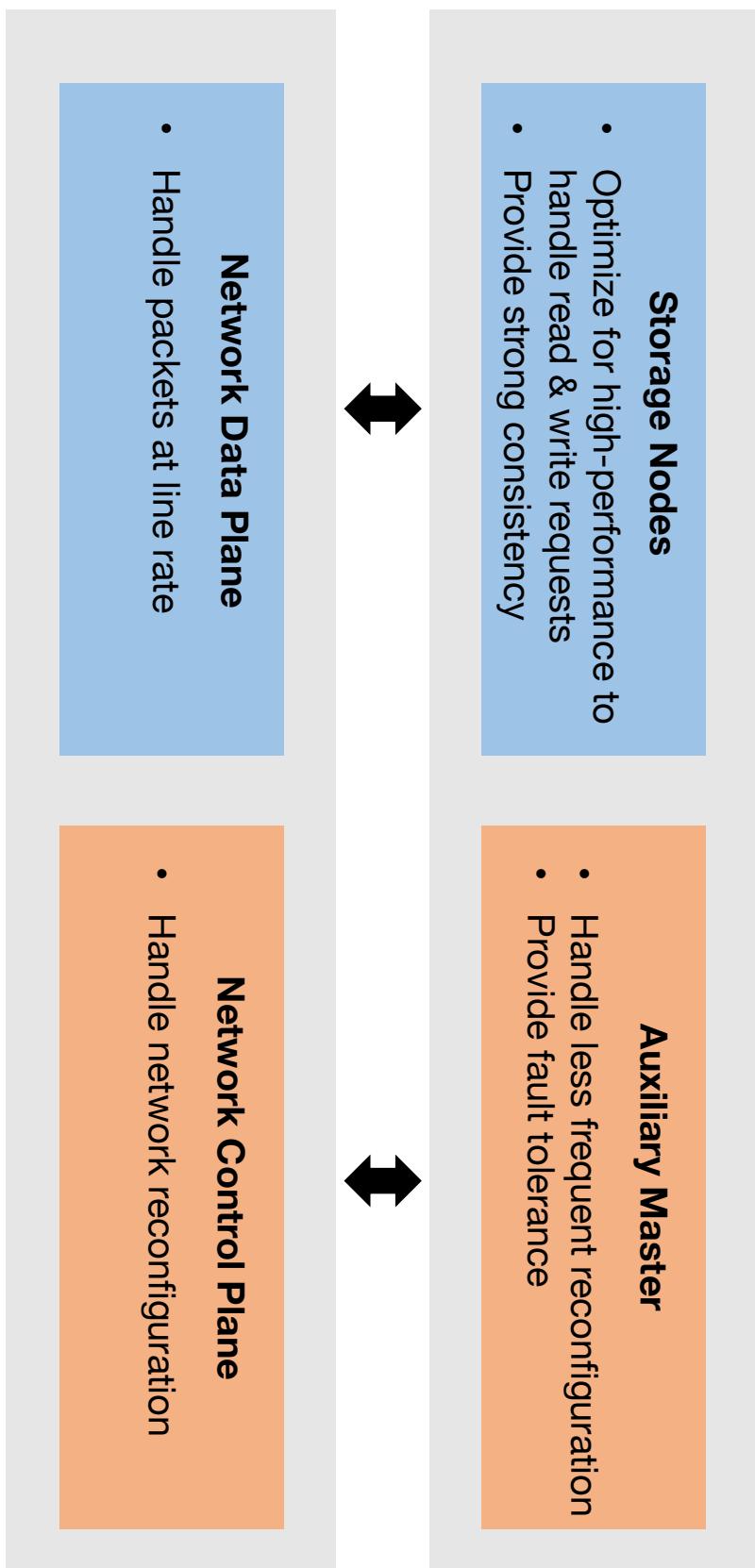
- Storage nodes are organized in a **chain** structure
- Handle operations
 - **Read** from the tail

What is chain replication

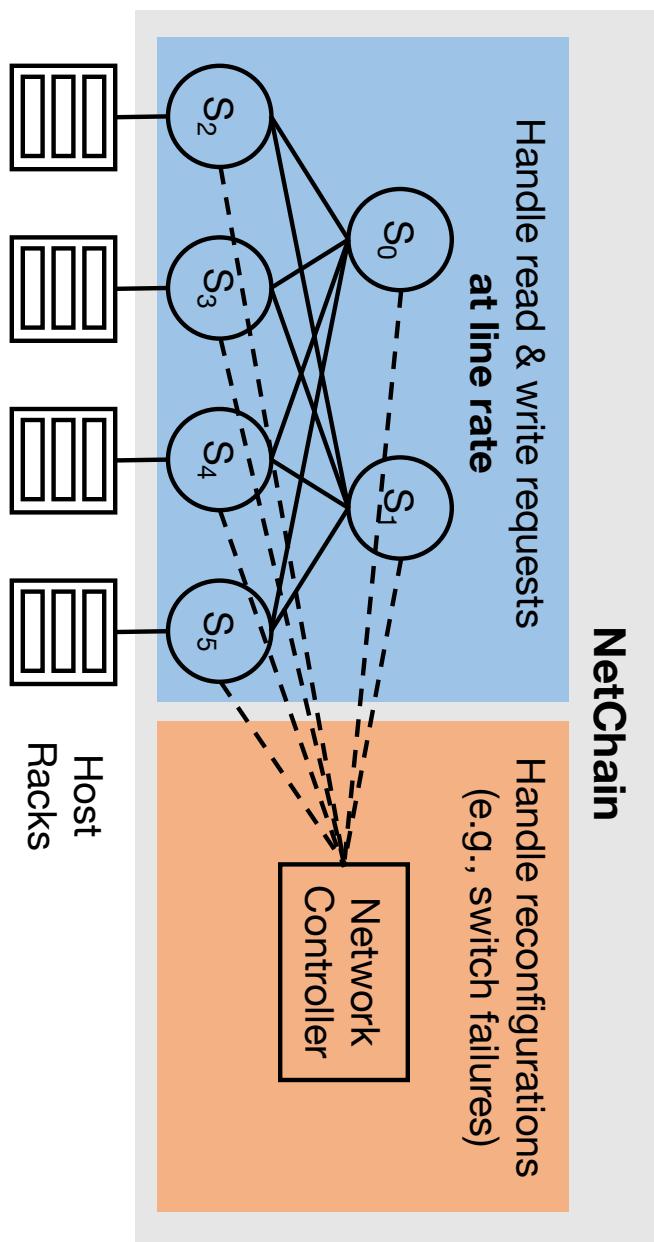


- Storage nodes are organized in a **chain** structure
- Handle operations
 - **Read** from the **tail**
 - **Write** from **head** to **tail**
- Provide strong consistency and fault tolerance
 - Tolerate **f** failures with **f+1** nodes

Division of labor in chain replication: a perfect match to network architecture



NetChain overview



How to build a strongly-consistent, fault-tolerant, PISA-based key-value store

- How to store and serve key-value items?
- How to route queries according to chain structure?
- How to handle out-of-order delivery in network?
- How to handle switch failures?

]

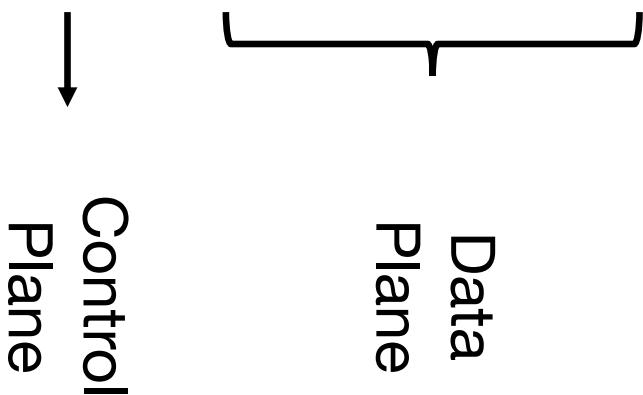
Data
Plane

→

Control
Plane

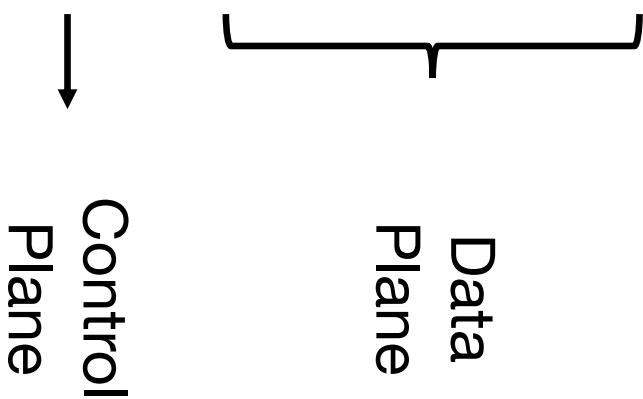
How to build a strongly-consistent, fault-tolerant, PISA-based key-value store

- How to store and serve key-value items?
NetCache ☺
- How to route queries according to chain structure?
- How to handle out-of-order delivery in network?
- How to handle switch failures?

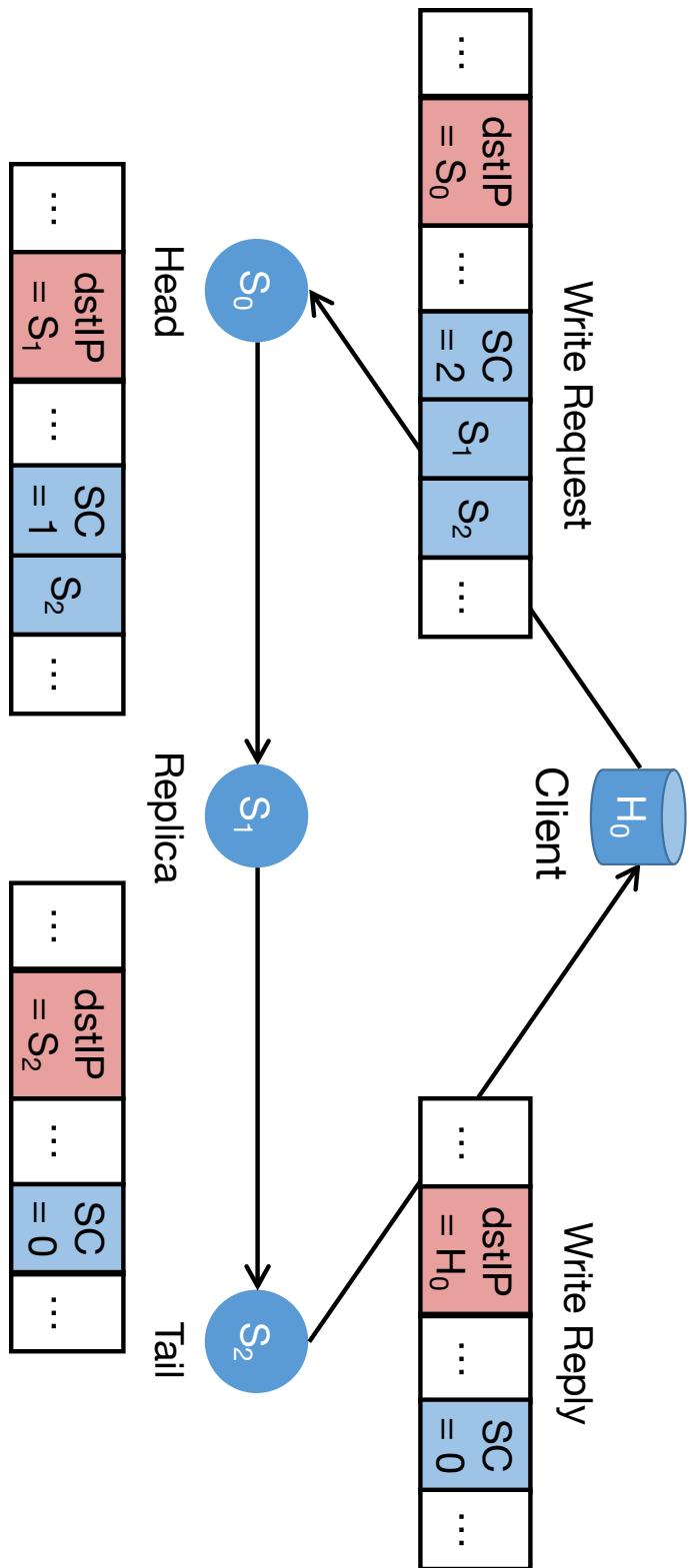


How to build a strongly-consistent, fault-tolerant, PISA-based key-value store

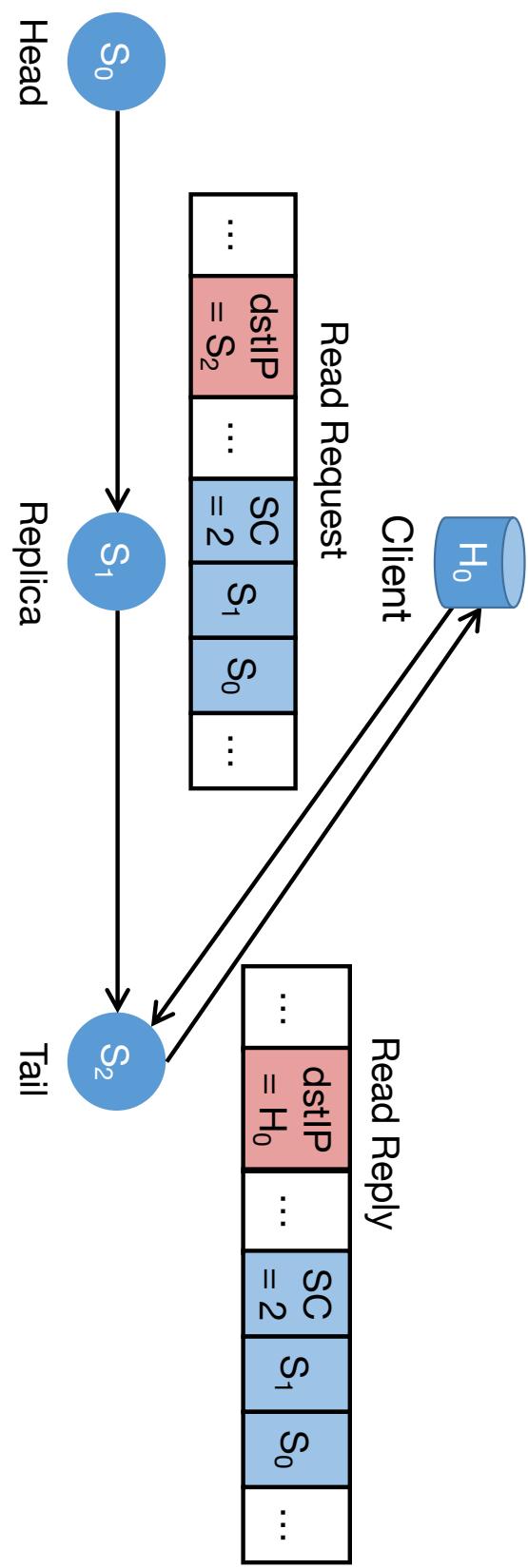
- How to store and serve key-value items?
- *How to route queries according to chain structure?*
- How to handle out-of-order delivery in network?
- How to handle switch failures?



NetChain routing: segment routing according to chain structure

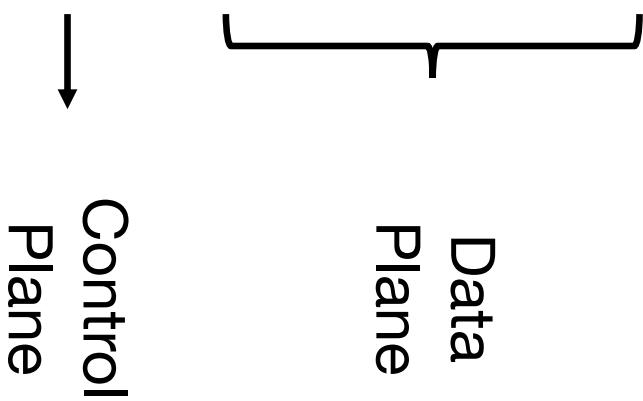


NetChain routing: segment routing according to chain structure

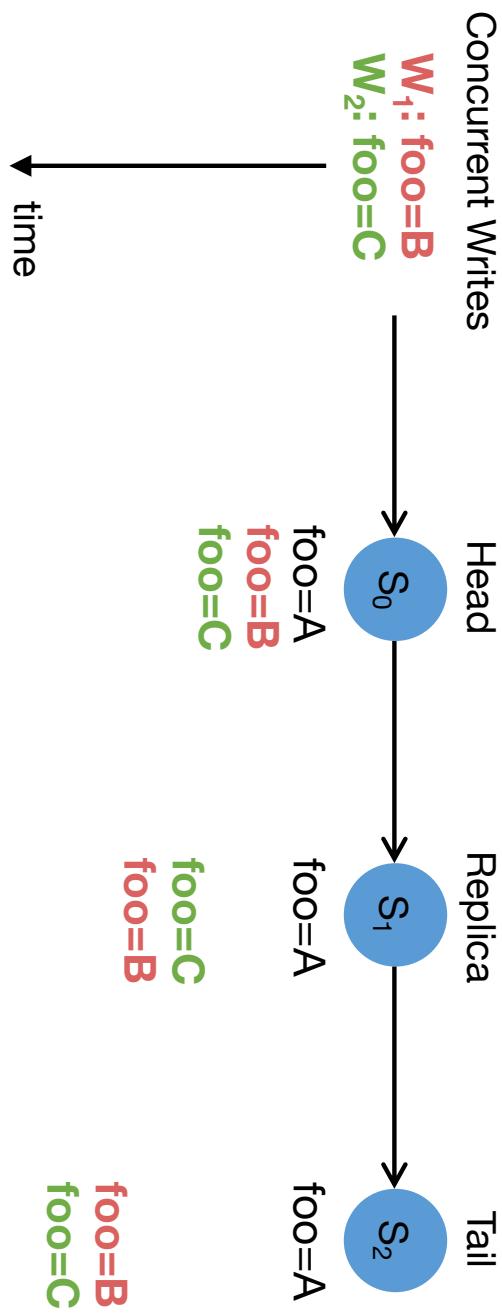


How to build a strongly-consistent, fault-tolerant, PISA-based key-value store

- How to store and serve key-value items?
- How to route queries according to chain structure?
- **How to handle out-of-order delivery in network?**
- How to handle switch failures?



Problem of Out-Of-Order delivery



Serialization with sequence number

How to build a strongly-consistent, fault-tolerant, PISA-based key-value store

- How to store and serve key-value items?
- How to route queries according to chain structure?
- How to handle out-of-order delivery in network?

]

Data
Plane

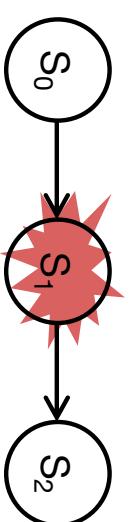
→
Control
Plane

→

➤ **How to handle switch failures?**

Handle a switch failure

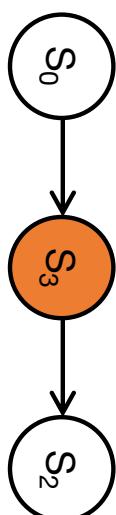
Before failure: tolerate f failures with $f+1$ nodes



Fast Failover

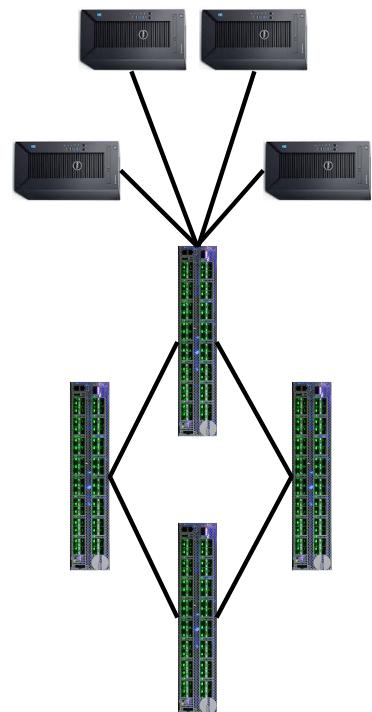


Failure Recovery



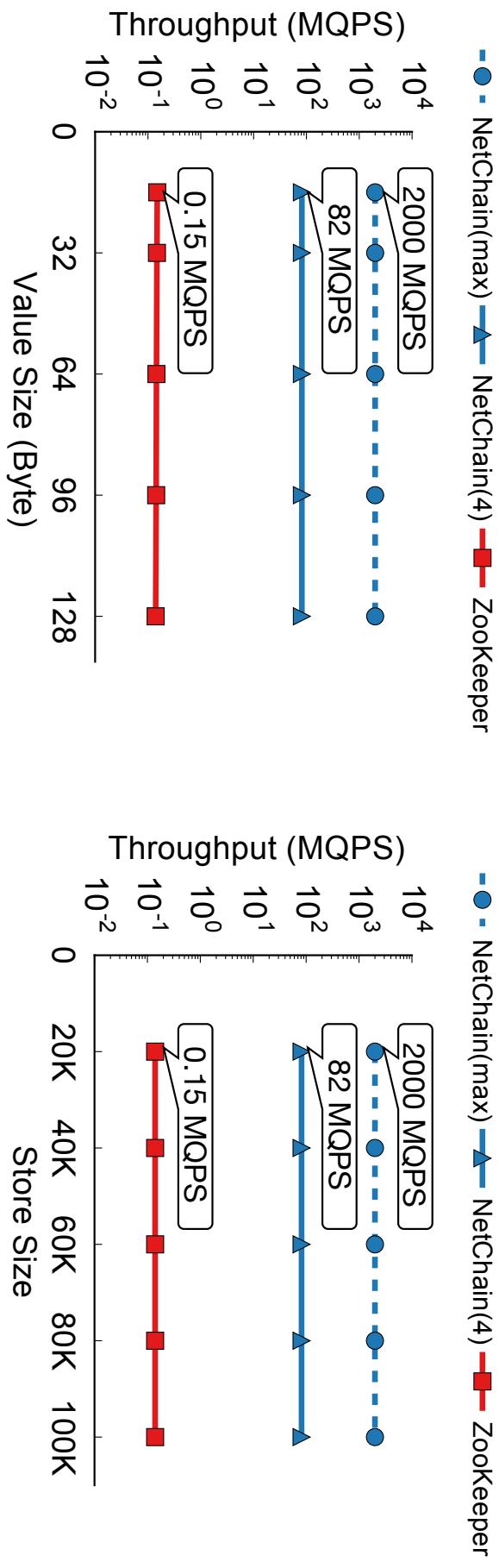
- Failover to remaining f nodes
- Tolerate $f-1$ failures
- Efficiency: only need to update **neighbor** switches of failed switch
 - Add another switch
 - Tolerate f failures again
 - Consistency: **two-phase atomic switching**
 - Minimize disruption: **virtual groups**

Implementation

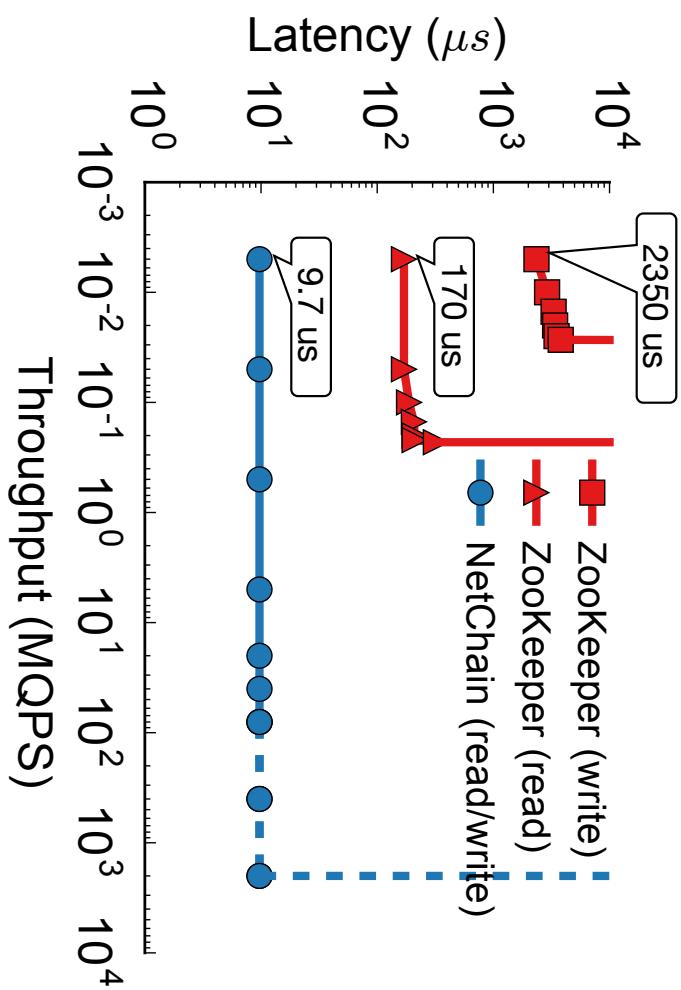


- **Testbed**
 - 4 Barefoot Tofino switches and 4 commodity servers
- **Switch**
 - P4 program on 6.5 Tbps Barefoot Tofino
 - Routing: basic L2/L3 routing
 - Key-value store: up to 100K items, up to 128-byte values
- **Server**
 - 16-core Intel Xeon E5-2630, 128 GB memory, 25/40 Gbps Intel NICs
 - Intel DPDK to generate query traffic: up to 20.5 MQPS per server

Orders of magnitude higher throughput



Orders of magnitude lower latency



Conclusion

- **Moore's law** is ending...
- **Specialized** processors for **domain-specific** workloads: GPU servers, FPGA servers, TPU servers...
- **PISA servers**: new generation of ultra-high performance systems for **IO-intensive workloads** enabled by PISA switches
 - NetCache: fast key-value caching built with PISA switches
 - NetChain: fast coordination built with PISA switches

Thanks!