# NetChain: Scale-Free Sub-RTT Coordination


Xin Jin

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

Nate Foster, Changhoon Kim, Ion Stoica

Conventional wisdom: avoid coordination

NetChain: lightning fast coordination
enabled by programmable switches

Open the door to rethink distributed systems design

# Coordination services: fundamental building block of the cloud

Applications

hadoop | Apache Spark | Apache kafka | MESOS

Coordination Service

Google Chubby | Apache Zookeeper | etcd

# Provide critical coordination functionalities

Applications

Coordination Service

Configuration Management

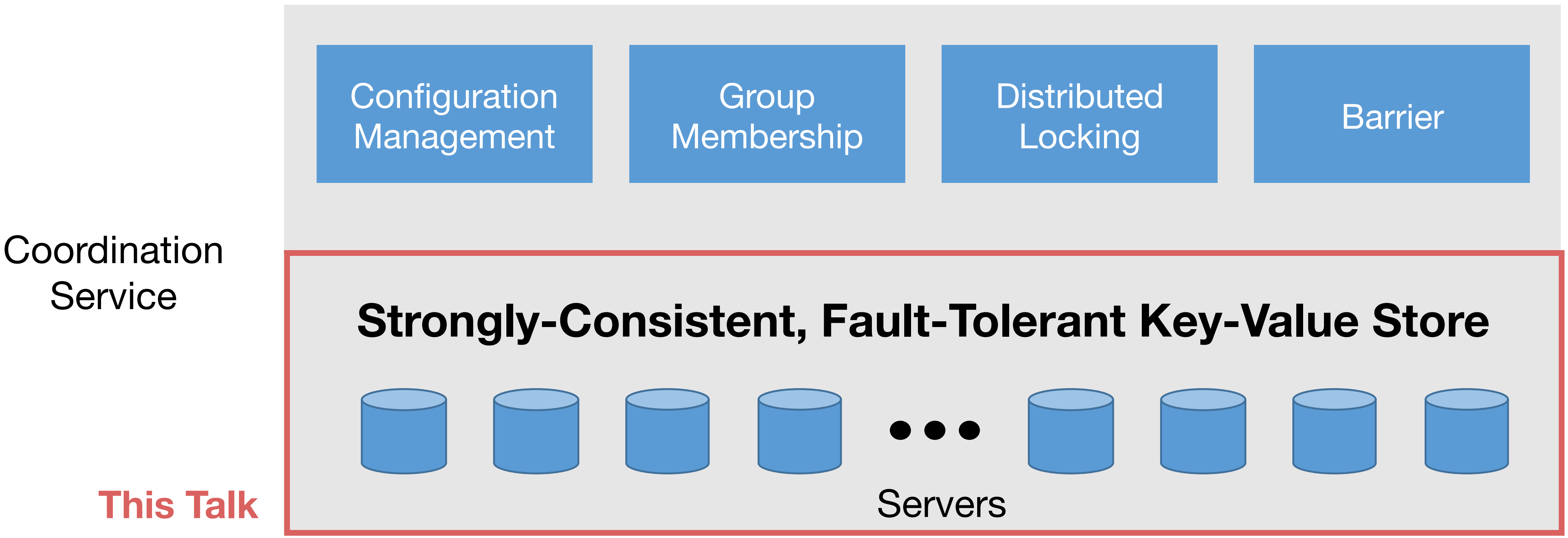
Group Membership

Distributed Locking

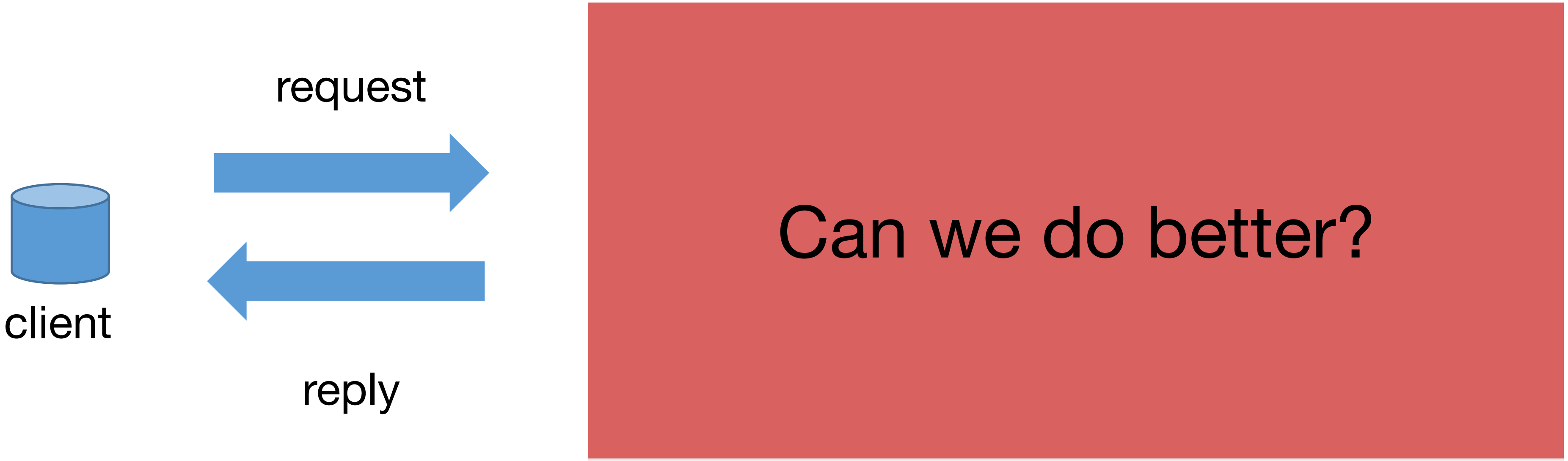
Barrier

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

Applications

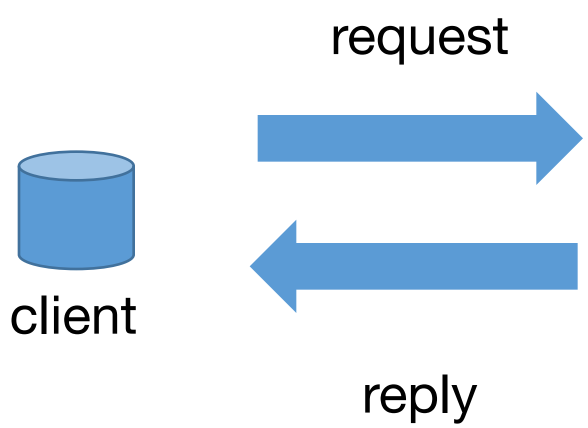


# Workflow of coordination services




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

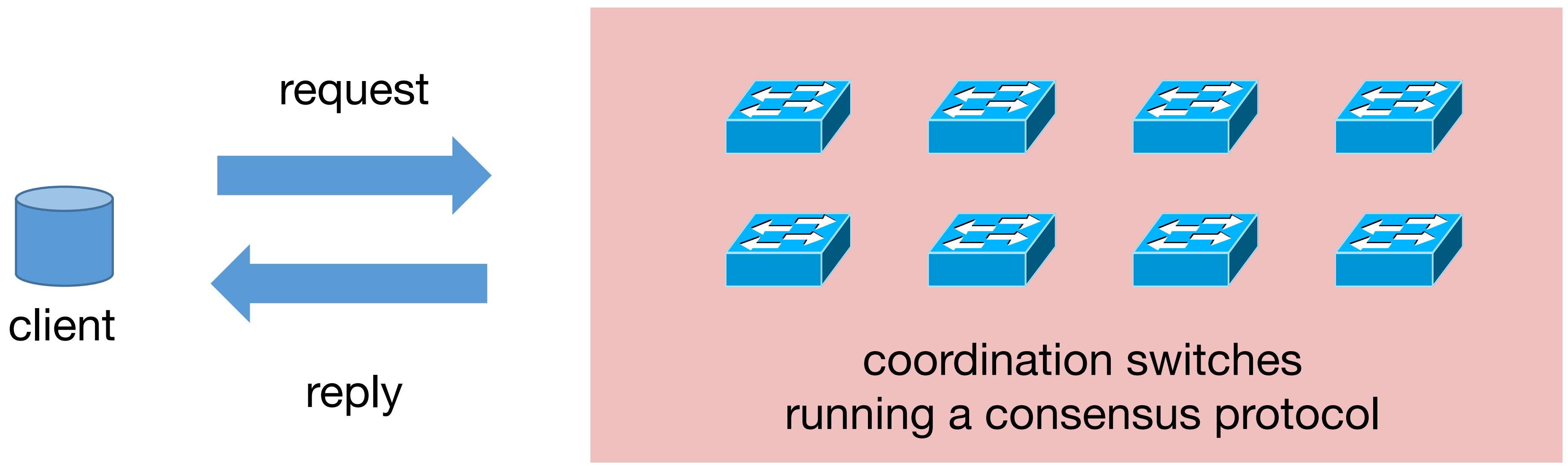
# Opportunity: in-network coordination




Distributed coordination is **communication-heavy,** not computation-heavy.

| | **Server** | **Switch** |
|---|---|---|
| Example | [NetBricks, OSDI'16] | **Barefoot Tofino** |
| Packets per second | 30 million | **A few billion** |
| Bandwidth | 10-100 Gbps | **6.5 Tbps** |
| Processing delay | 10-100 us | **< 1 us** |

# Opportunity: in-network coordination




- Throughput: switch throughput
- Latency: half of an RTT

# Design goals for coordination services

- High throughput
- Low latency

Directly from
high-performance switches

- Strong consistency
- Fault tolerance

How?

# Design goals for coordination services

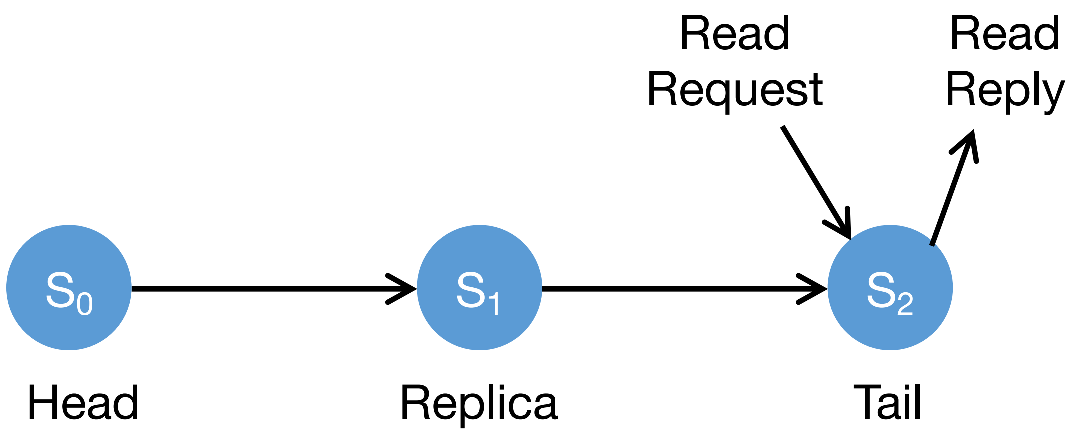
- High throughput
- Low latency

Directly from high-performance switches

- Strong consistency
- Fault tolerance

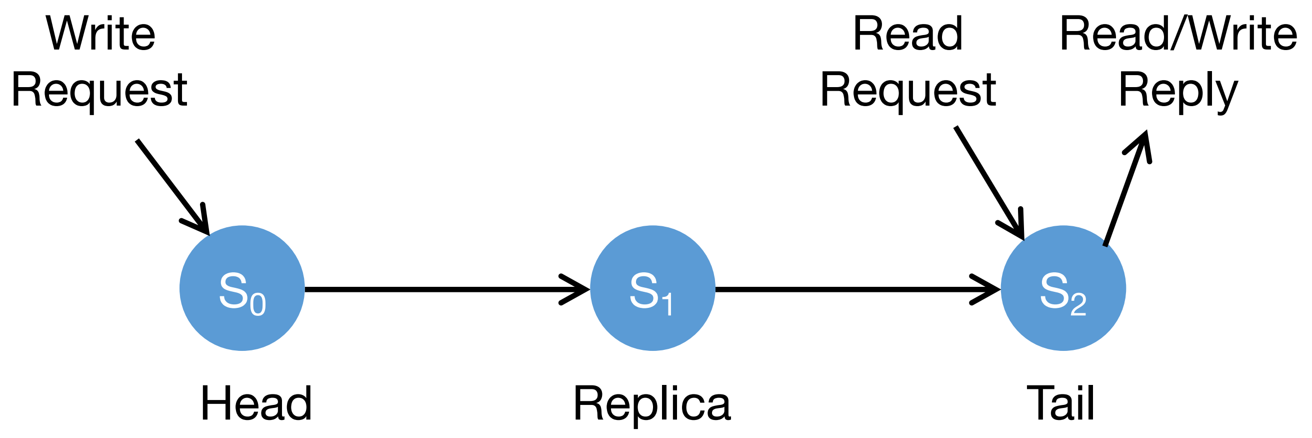
Chain replication in the network

# 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

**Chain Replication**

**Storage Nodes**

- Optimize for high-performance to handle read & write requests
- Provide strong consistency

**Auxiliary Master**

- Handle less frequent reconfiguration
- Provide fault tolerance

**Network Architecture**

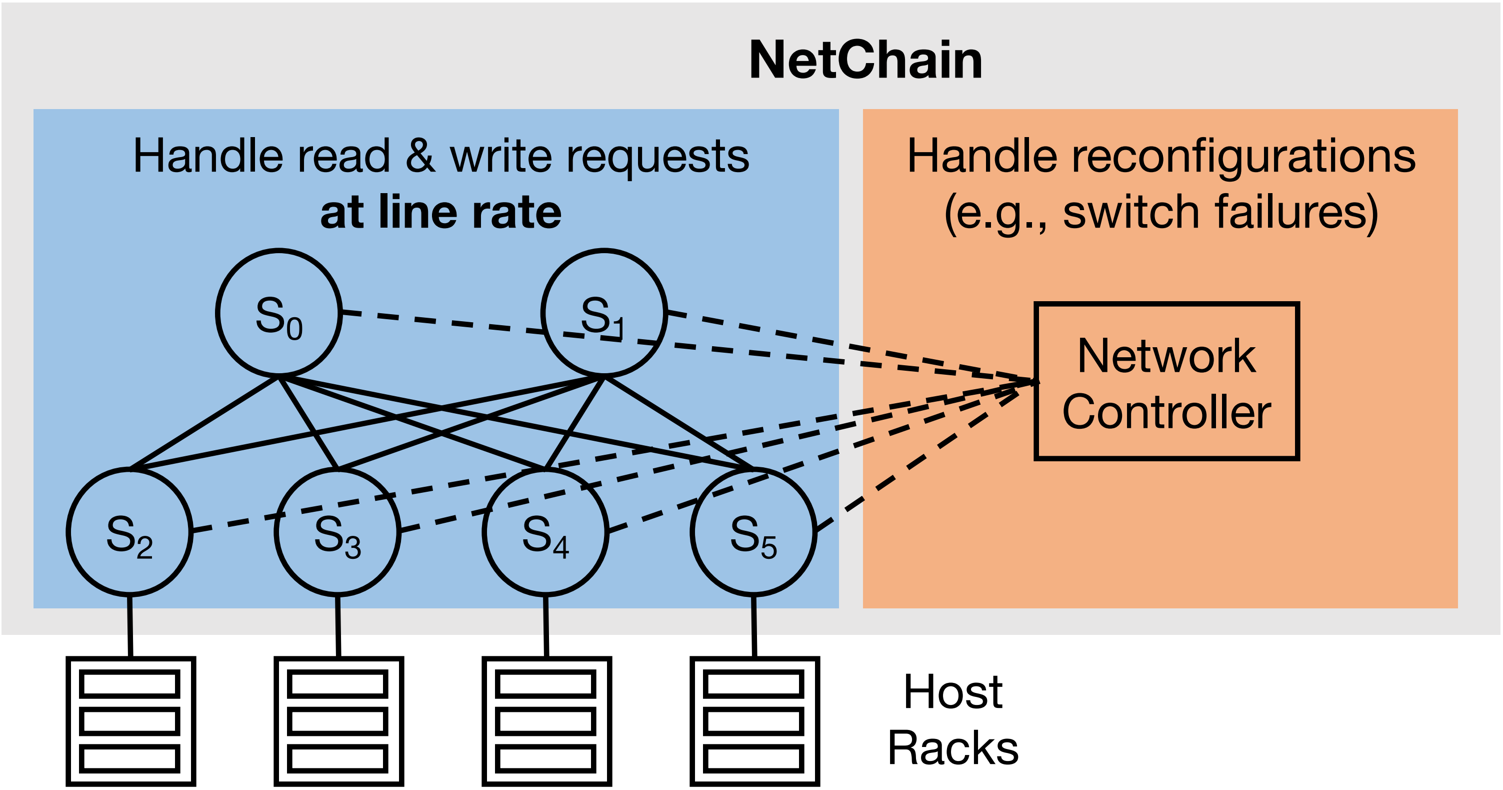
**Network Data Plane**

- Handle packets at line rate

**Network Control Plane**

- Handle network reconfiguration

# NetChain overview



# How to build a strongly-consistent, fault-tolerant, in-network 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?

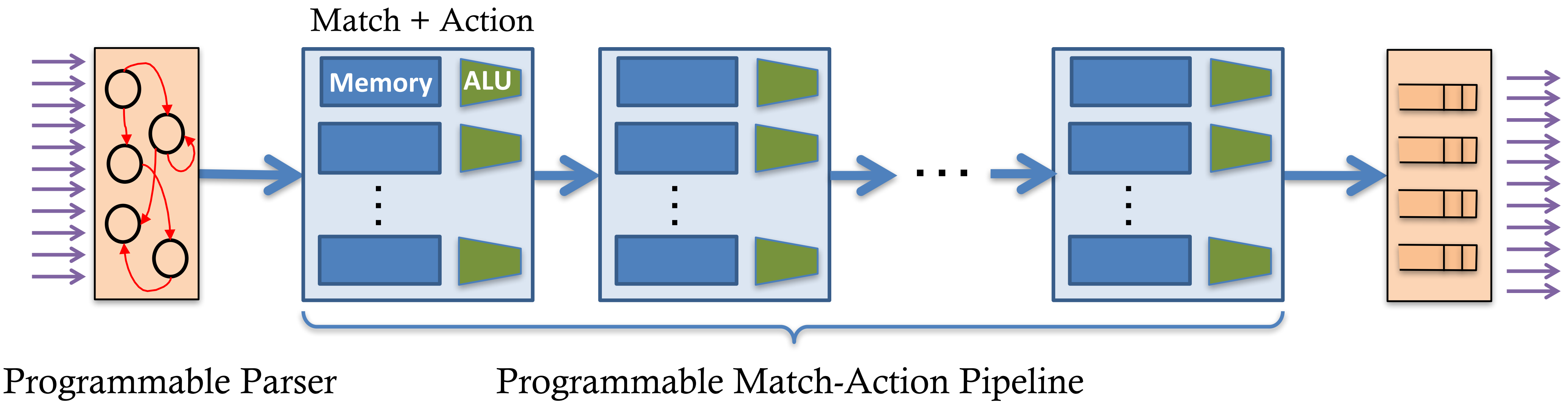
Data Plane

- How to handle switch failures?

Control Plane

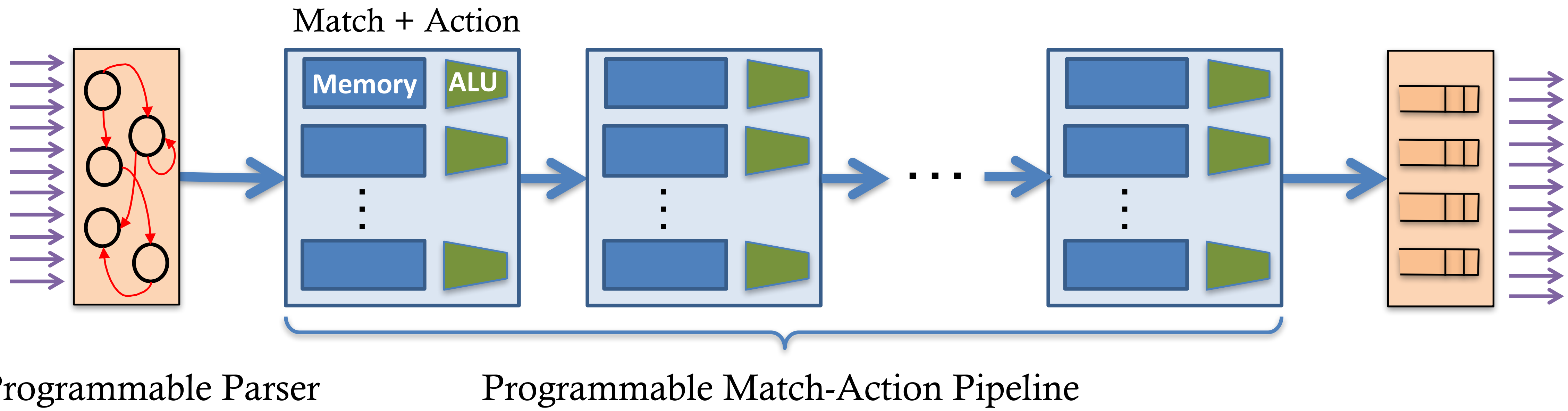
# PISA: Protocol Independent Switch Architecture

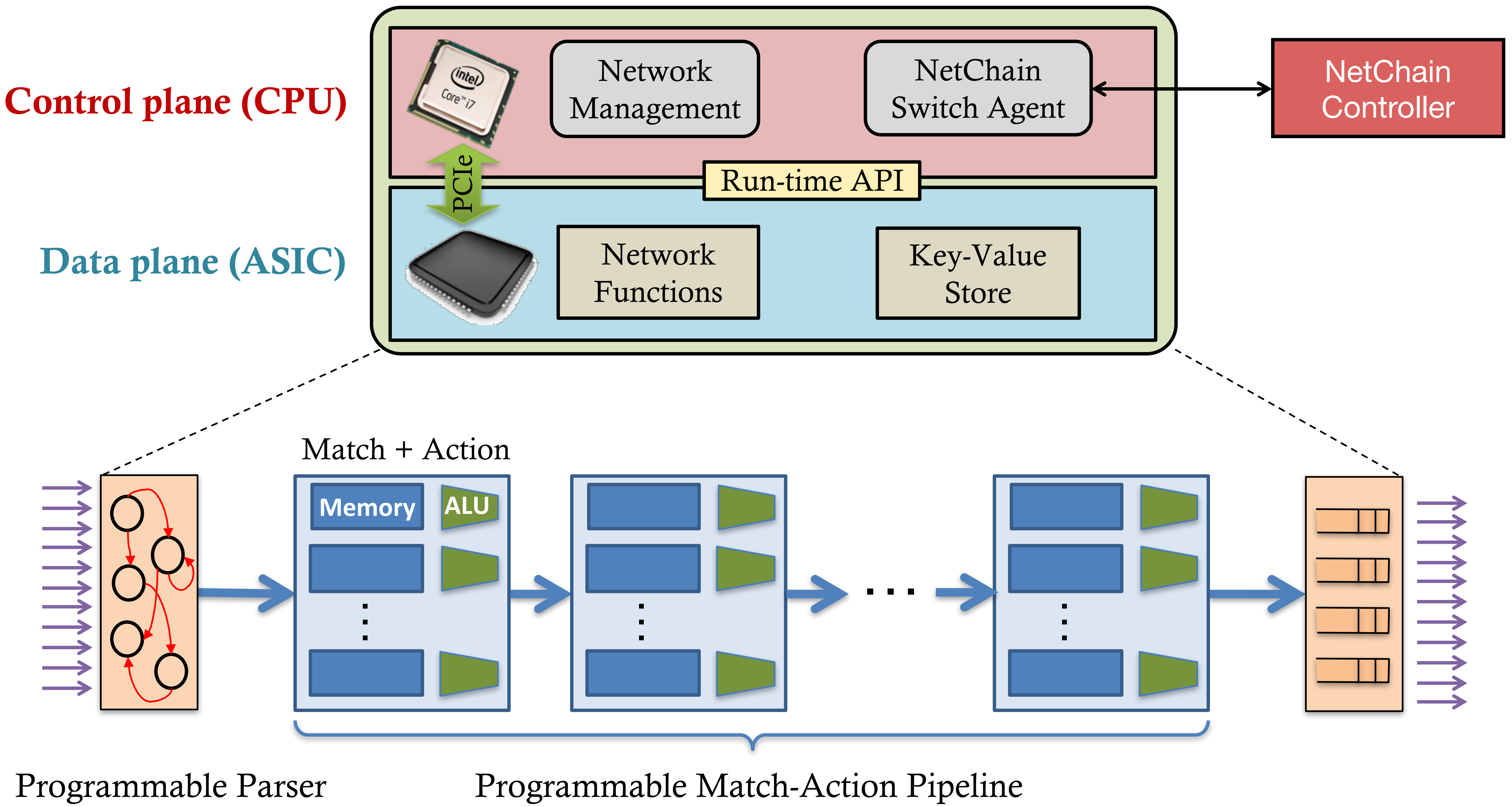
- Programmable Parser
  - Convert packet data into metadata
- Programmable Mach-Action Pipeline
  - Operate on metadata and update memory state



# 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 **at line rate**




Control plane (CPU)
Data plane (ASIC)
Network Management
NetChain Switch Agent
NetChain Controller
PCIe
Run-time API
Network Functions
Key-Value Store
Match + Action
Memory
ALU
Programmable Parser
Programmable Match-Action Pipeline

# How to build a strongly-consistent, fault-tolerant, in-network key-value store

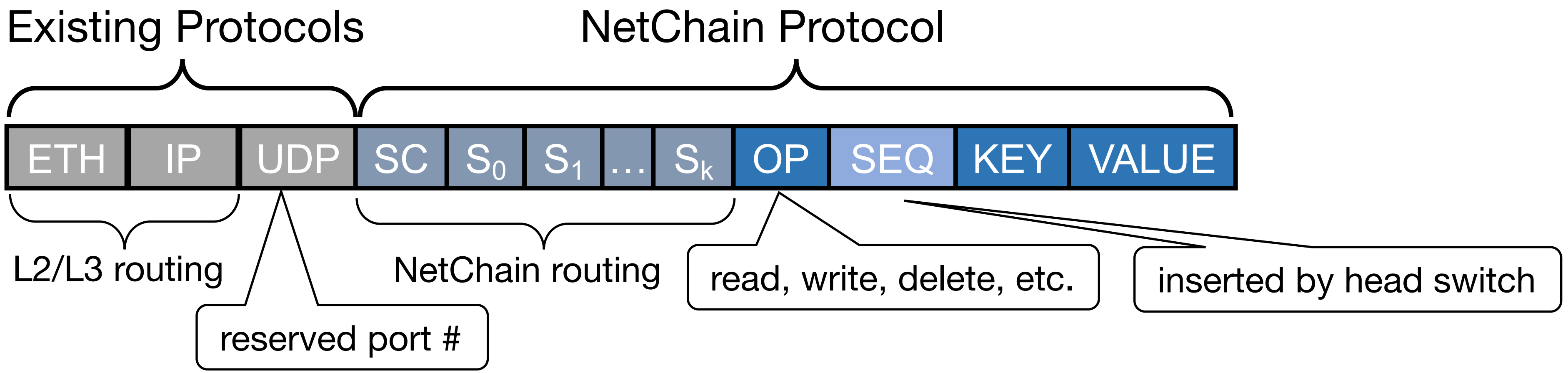
- How to store and serve key-value items?
- How to route queries according to chain structure?
- How to handle out-of-order delivery in network?

Data Plane

- How to handle switch failures?

Control Plane

# NetChain packet format



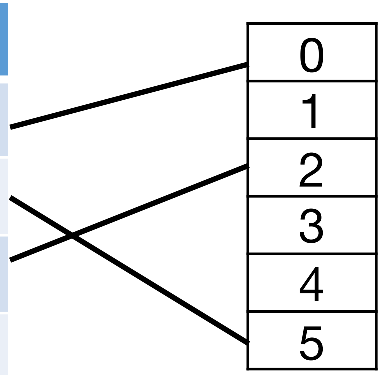

- ➢ Application-layer protocol: compatible with existing L2-L4 layers
- ➢ Invoke NetChain with a reserved UDP port

# In-network key-value storage

Match-Action Table

| Match | Action |
|---|---|
| Key = X | Read/Write RA[0] |
| Key = Y | Read/Write RA[5] |
| Key = Z | Read/Write RA[2] |
| Default | Drop() |

Register Array (RA)




- Key-value store in a single switch
  - Store and serve key-value items using register arrays [SOSP'17, NetCache]

- Key-value store in the network
  - Data partitioning with consistent hashing and virtual nodes

# How to build a strongly-consistent, fault-tolerant, in-network key-value store

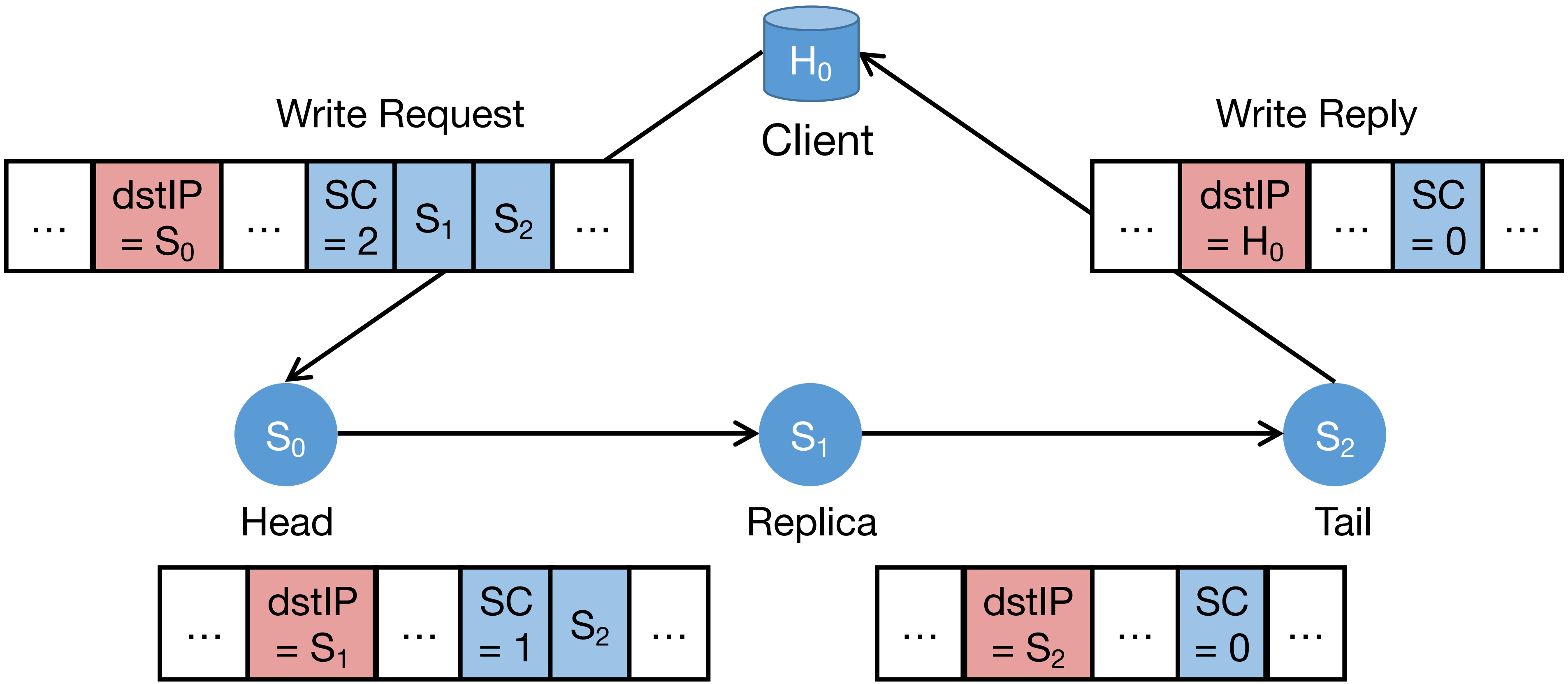
- How to store and serve key-value items?
- How to route queries according to chain structure?
- How to handle out-of-order delivery in network?

Data Plane

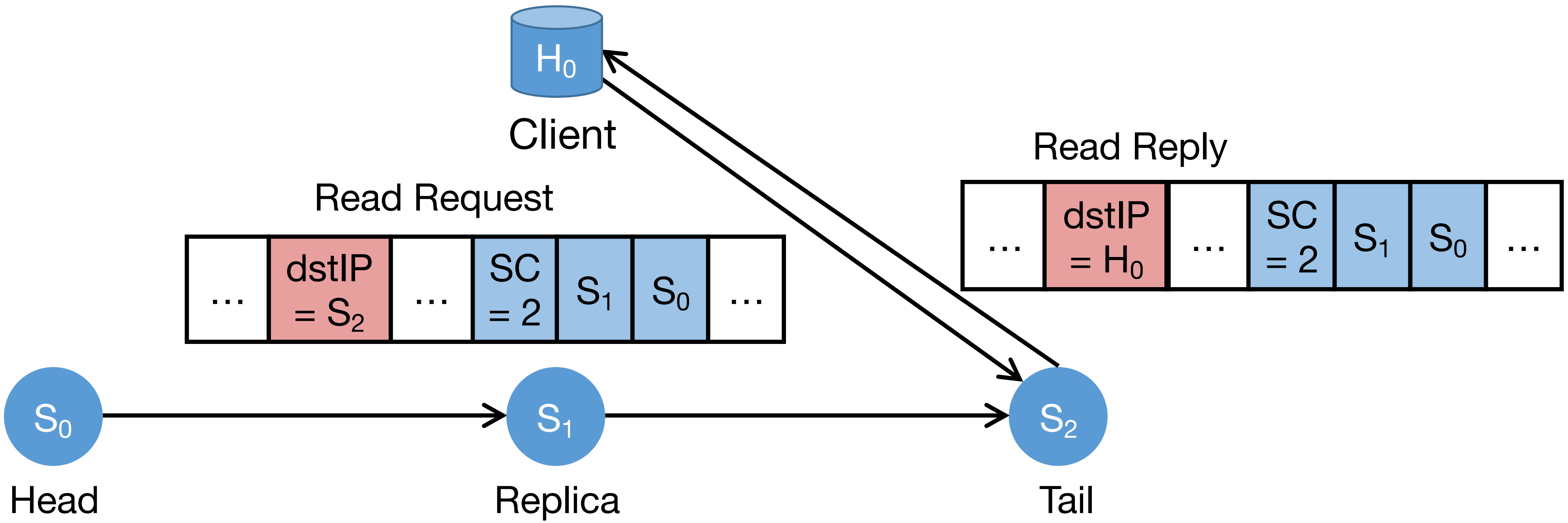
- How to handle switch failures?

→ Control Plane

# NetChain routing: segment routing according to chain structure



# NetChain routing: segment routing according to chain structure



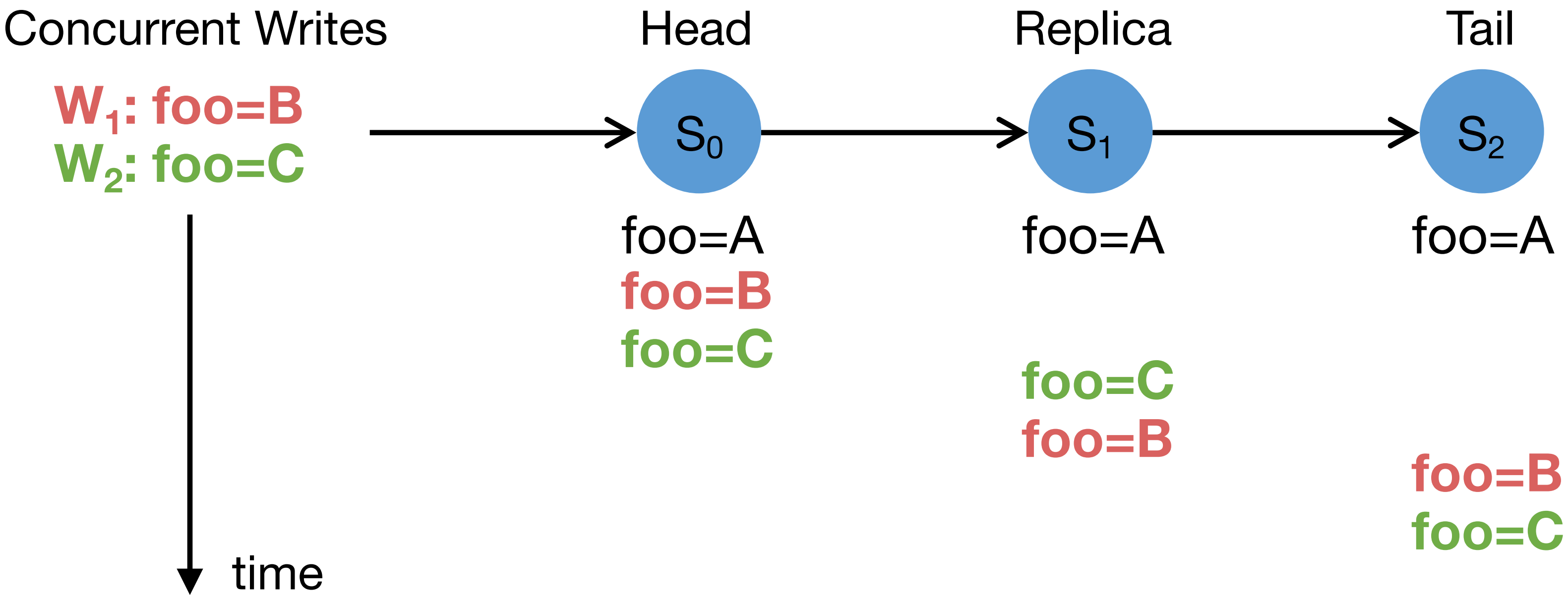
# How to build a strongly-consistent, fault-tolerant, in-network key-value store

- ➢ 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

# Problem of out-of-order delivery




Serialization with sequence number

# How to build a strongly-consistent, fault-tolerant, in-network 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?

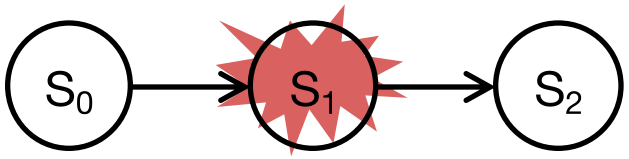
Data Plane

- How to handle switch failures?

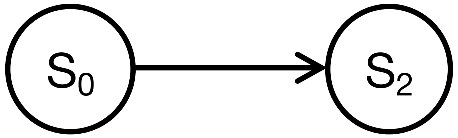
Control Plane

# Handle a switch failure

Before failure: tolerate f failures with f+1 nodes

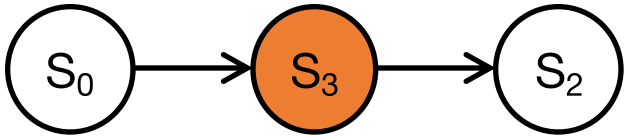



## Fast Failover




- Failover to remaining f nodes
- Tolerate f-1 failures
- Efficiency: only need to update neighbor switches of failed switch

## Failure Recovery



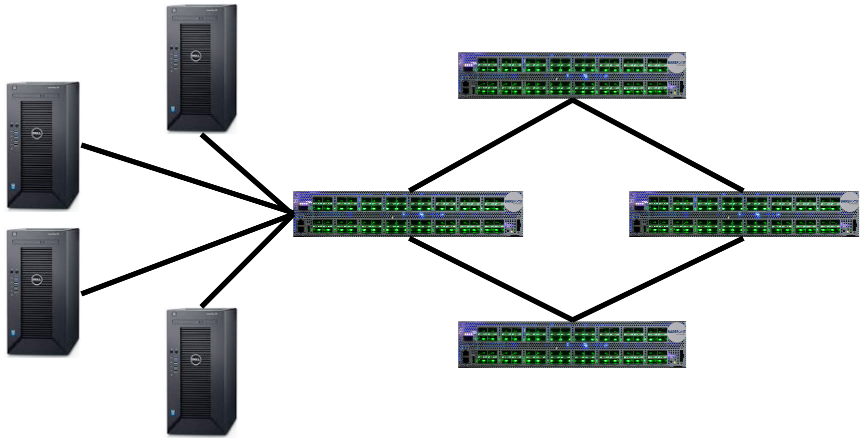

- Add another switch
- Tolerate f+1 failures again
- Consistency: two-phase atomic switching
- Minimize disruption: virtual groups

# Protocol correctness

**Invariant.** For any key k that is assigned to a chain of nodes [$S_1$, $S_2$, …, $S_n$], if $1 \leq i < j \leq n$ (i.e., $S_i$ is a predecessor of $S_j$), then $State^{S_i}[k].seq \geq State^{S_j}[k].seq$.

- Guarantee strong consistency under packet loss, packet reordering, and switch failures
- See paper for TLA+ specification

# 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

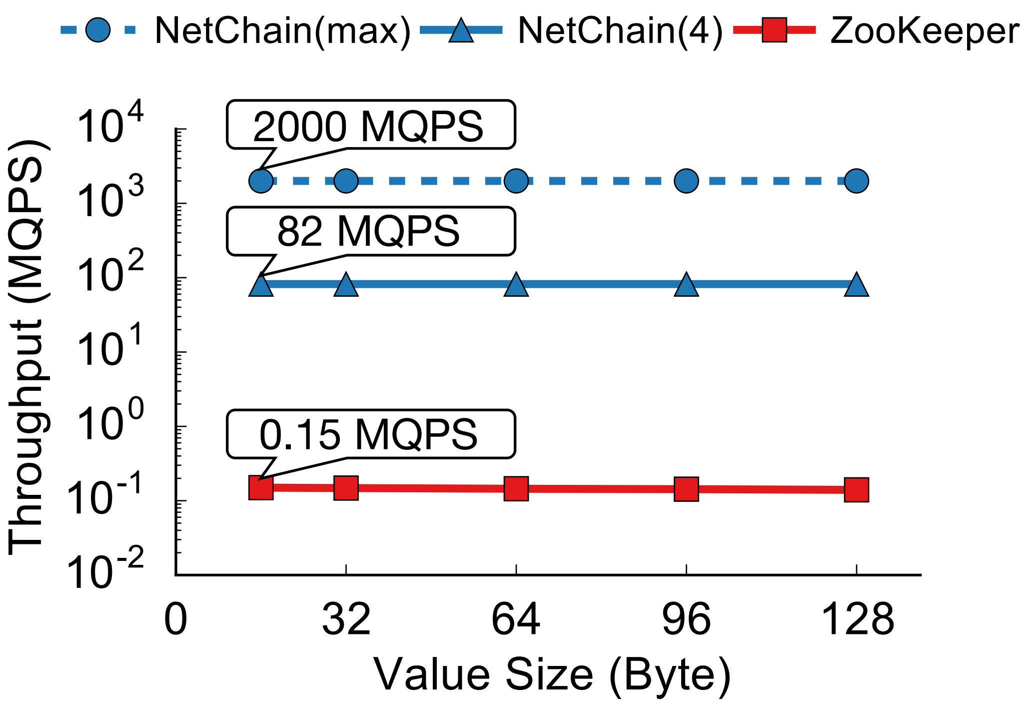
# Evaluation

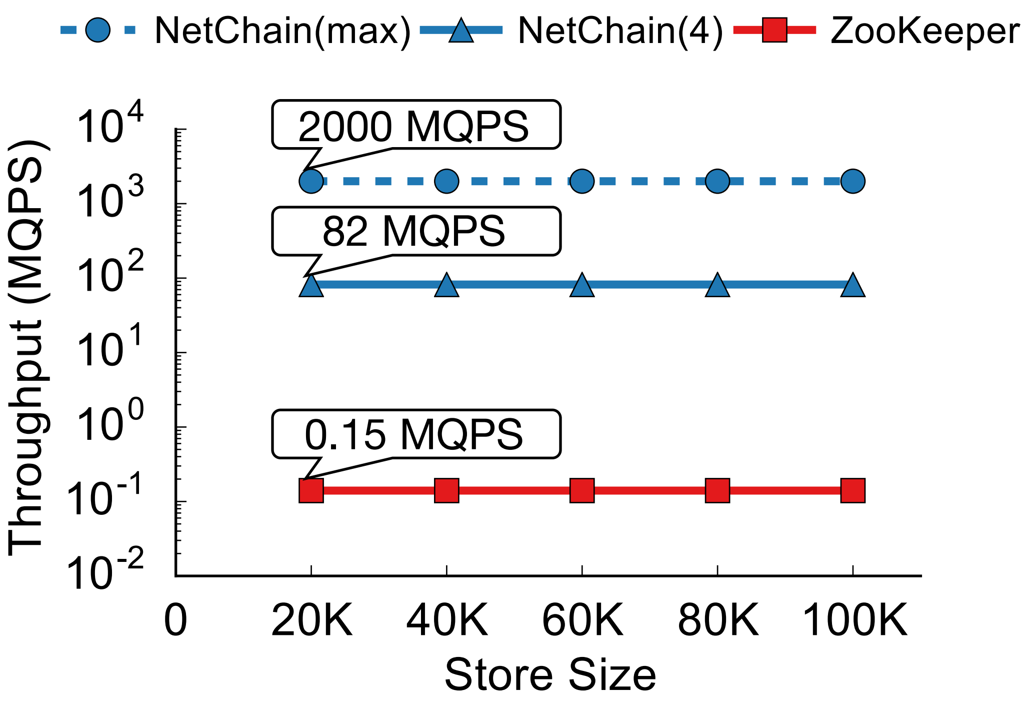
- Can NetChain provide significant performance improvements?
- Can NetChain scale out to a large number of switches?
- Can NetChain efficiently handle failures?
- Can NetChain benefit applications?

# Evaluation

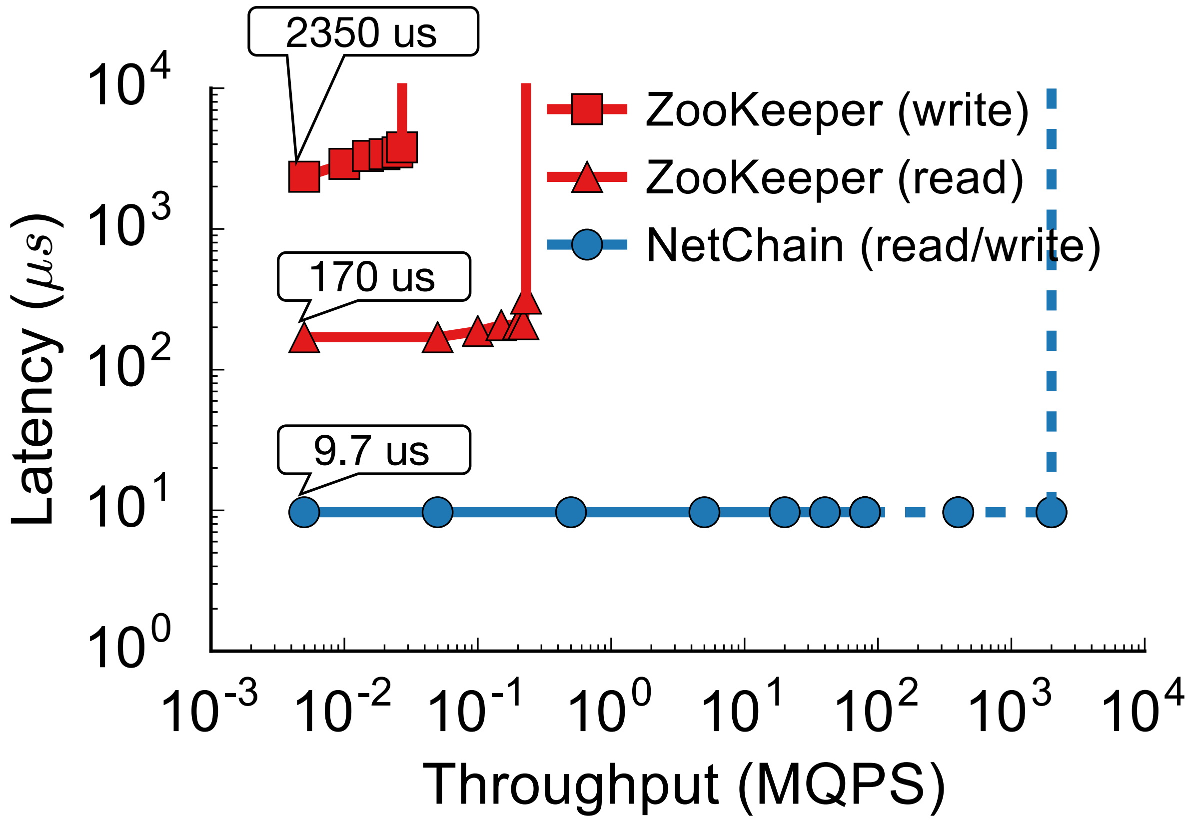
- Can NetChain provide significant performance improvements?
- Can NetChain scale out to a large number of switches?
- Can NetChain efficiently handle failures?
- Can NetChain benefit applications?

# Orders of magnitude higher throughput

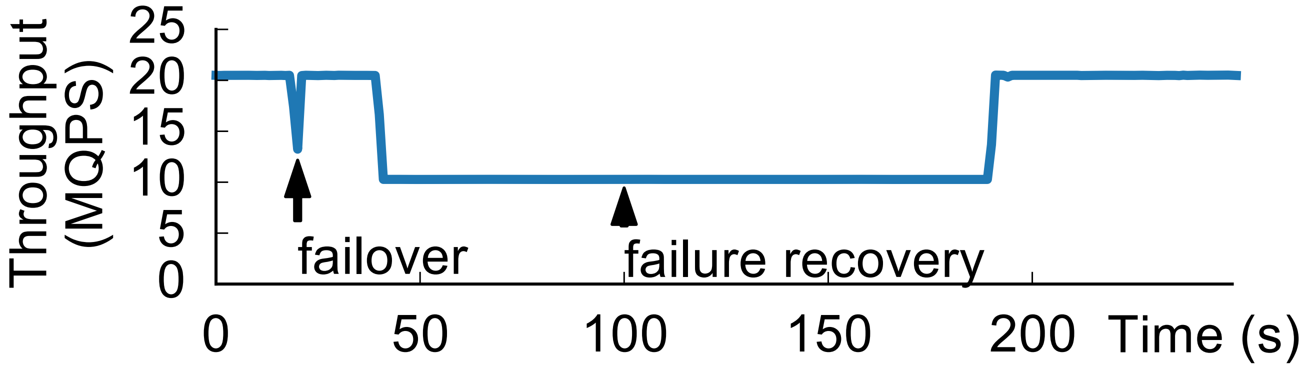


# Orders of magnitude lower latency



# Handle failures efficiently




(a) 1 Virtual Group.

# Conclusion

- NetChain is an in-network coordination system that provides billions of operations per second with sub-RTT latencies
- Rethink distributed systems design
  - Conventional wisdom: avoid coordination
  - NetChain: lightning fast coordination with programmable switches
- 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-heavy workloads enabled by PISA switches

Thanks!