DPDK

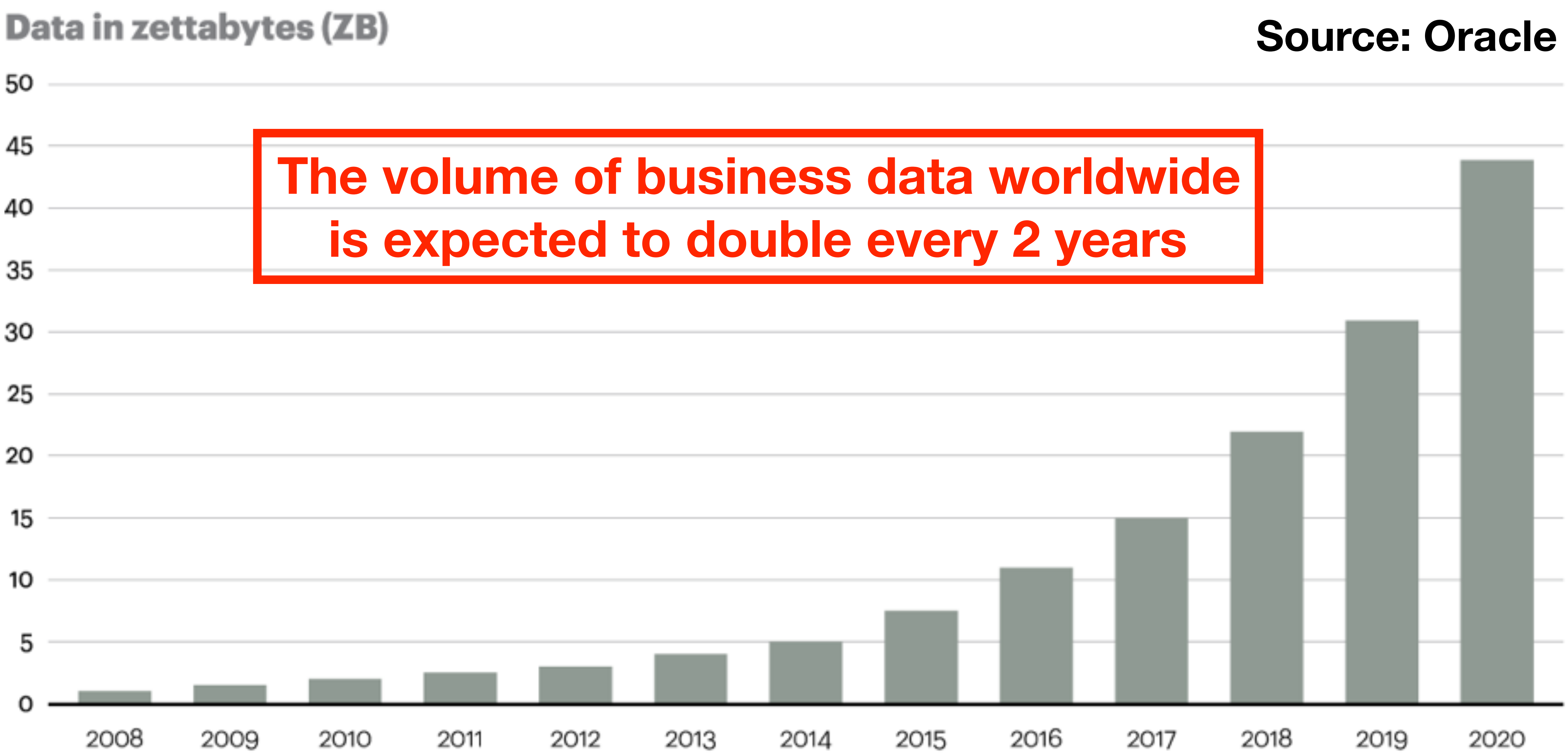
DATA PLANE DEVELOPMENT KIT

Building High-Performance Networked Systems with Innovative Hardware and Software Techniques

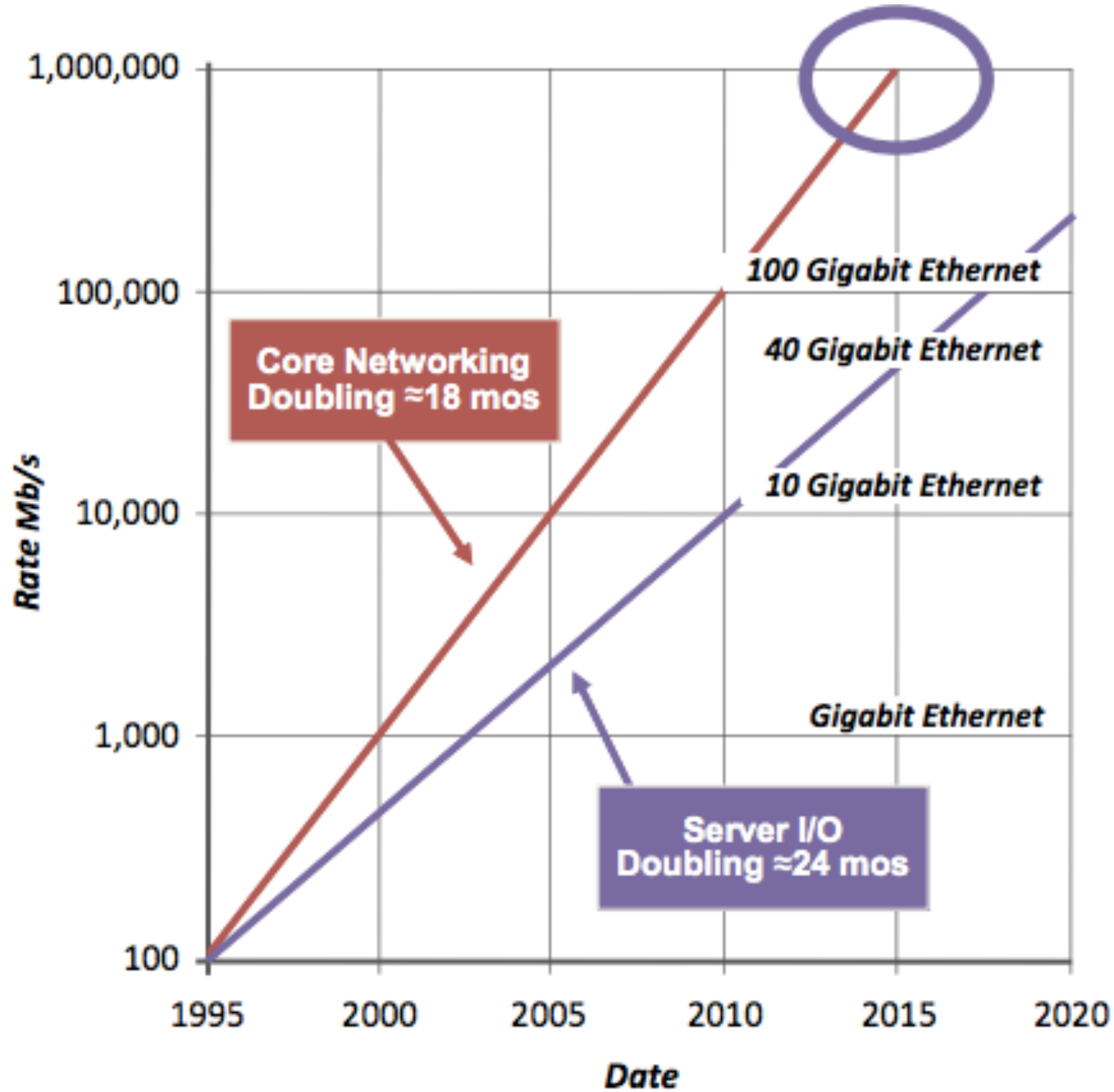
Kai Zhang
University of Science and Technology of China

The Growth Trend of Business Data

Source: Oracle



The Growth Trend of Network Speed



Source: IEEE 802.3 Higher Speed Study Group - Tutorial

Solutions for Next Generation Networked Systems

Load Ba

How to utilize?

NETWORKED SYSTEM

New drivers, bypass the OS

DPDK, Netmap, …

OPERATING SYSTEM

GPUs

CPUs with Integrated GPUs

CPUs

Xeon Phi

HARDWARE

Solutions for Next Generation Networked Systems

- Demonstration of Two Networked Systems
  - Mega-KV
    - A key-value store system with the highest throughput
    - 100x higher throughput than Memcached
  - Snort with DPDK
    - Enhance the efficiency of network I/O of Snort with DPDK
    - A cooperation between USTC and Intel for educational purpose
      - To be a course lab for Advanced Computer Networks

Mega-KV: A Case for GPUs to Maximize the Throughput of In-Memory Key-Value Stores

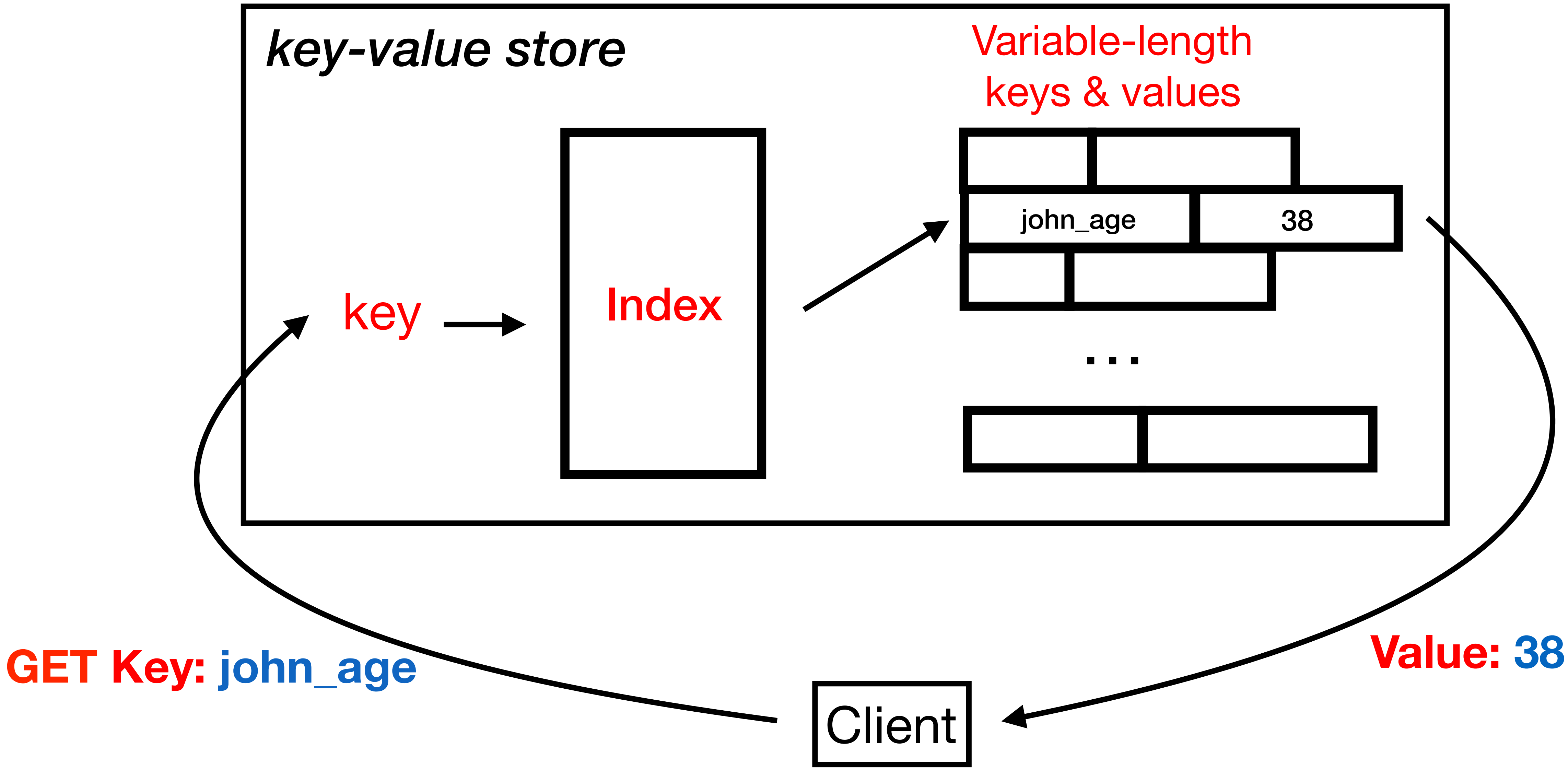
Key-Value Stores

- A simple but effective method to manage data where a data record (or a value) is stored and retrieved with its associated key
  - variable type and length of record (value)
  - simple or no schema
  - easy software development for many applications

- Key-value stores have been widely deployed in data processing production systems:

Simple and Easy Interfaces of Key-Value Stores

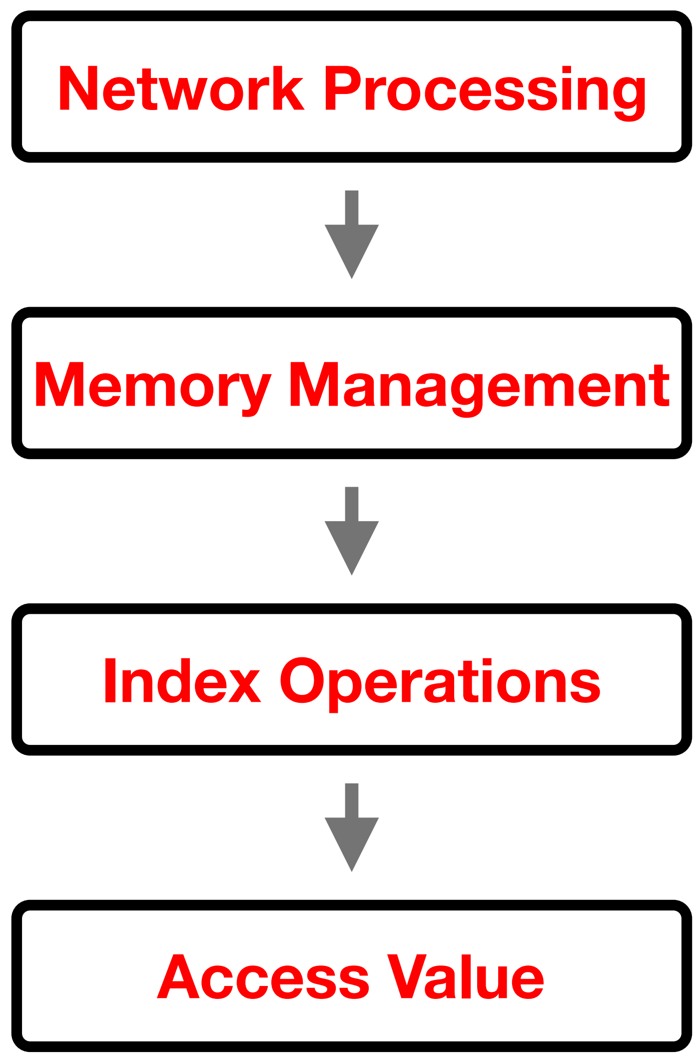
- **set** (key, value)
- value = **get** (key)



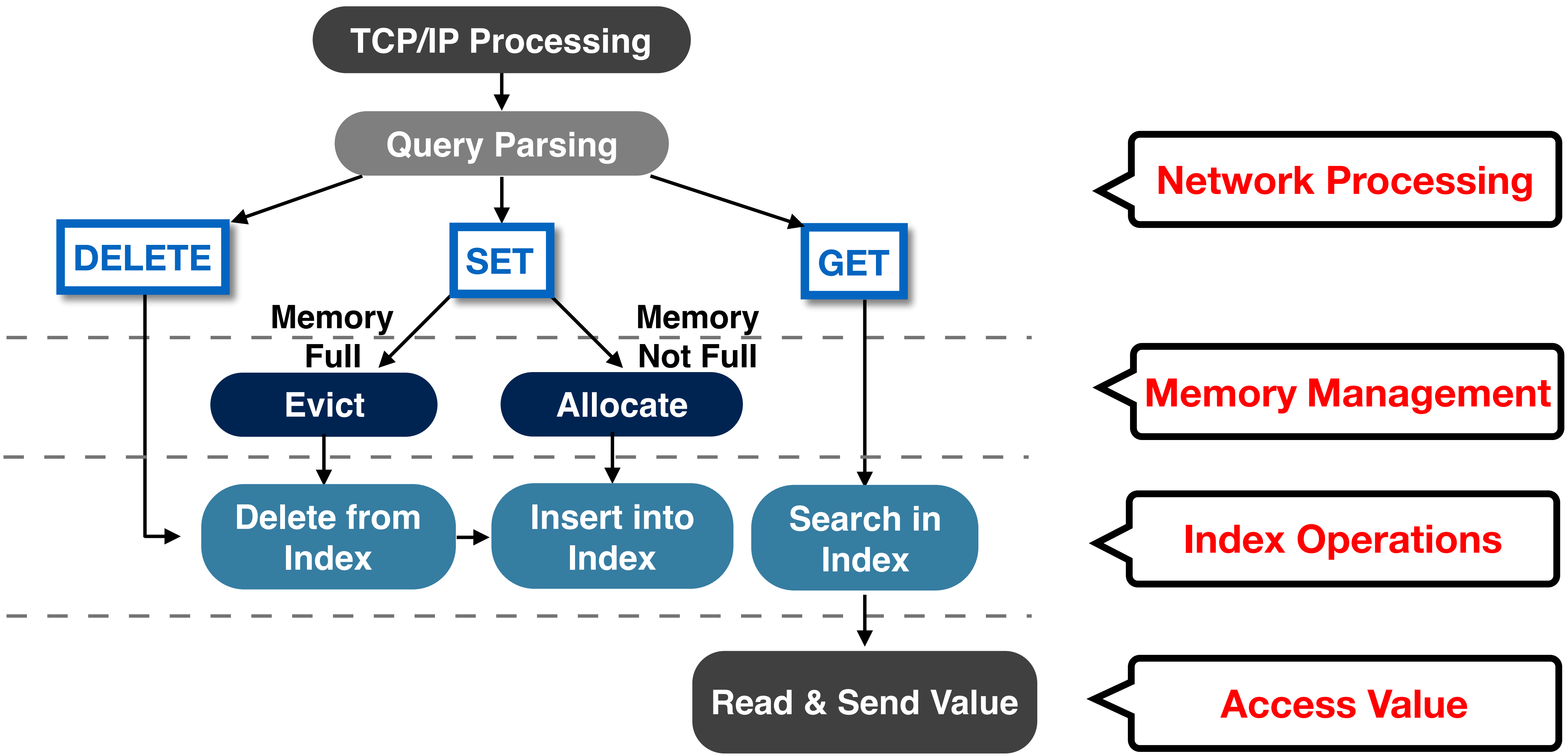
Key-Value Stores: Examples

	Keys	Values
Amazon	**Customer ID**	**Customer profile (e.g., credit card, buying history)**
Facebook, Twiter	**User ID**	**User profile (e.g., friends, photos, posts)**
iCloud/iTunes	**Movie/song name**	**Movie, Song**
Distributed file Systems	**Block ID**	**Block**

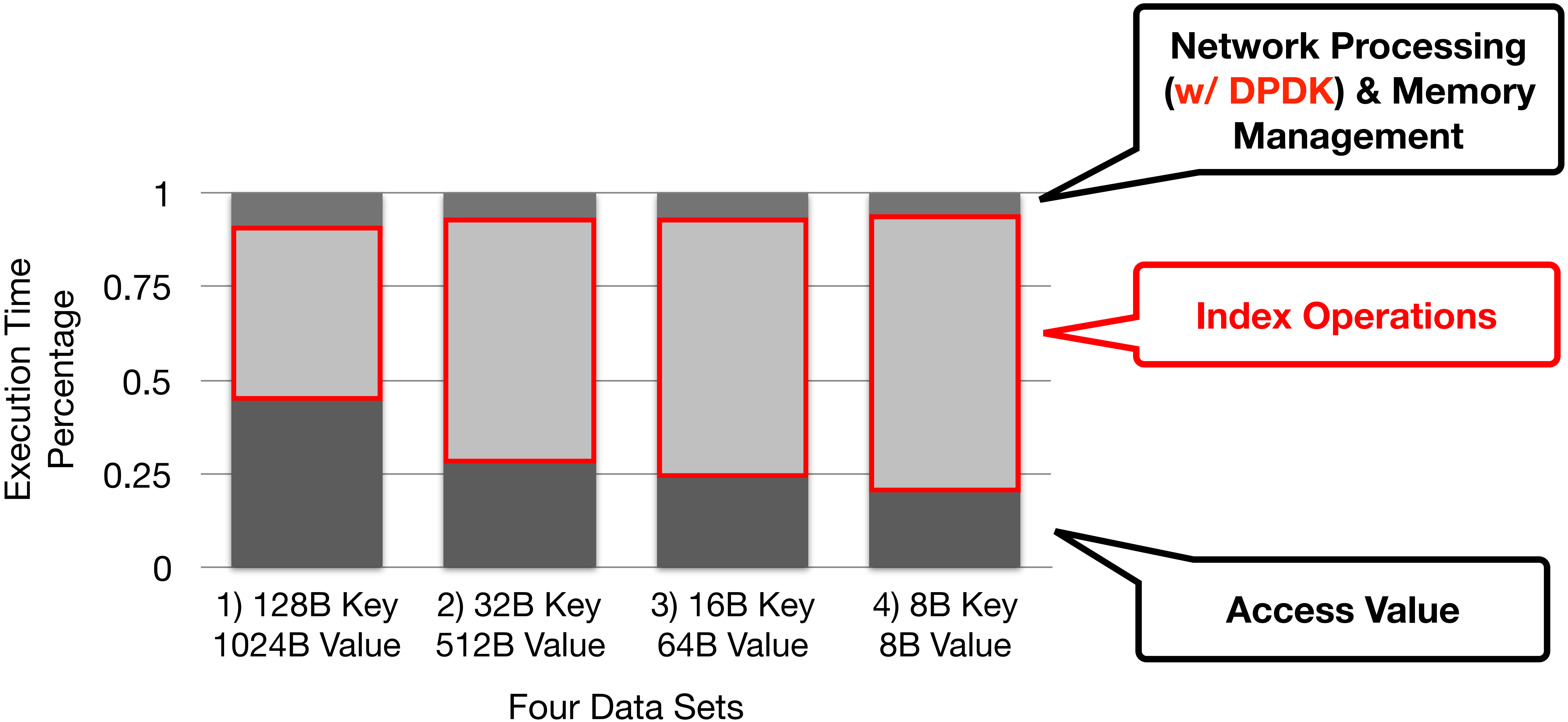
Workflow of a Typical In-Memory Key-Value Store



Workflow of a Typical In-Memory Key-Value Store

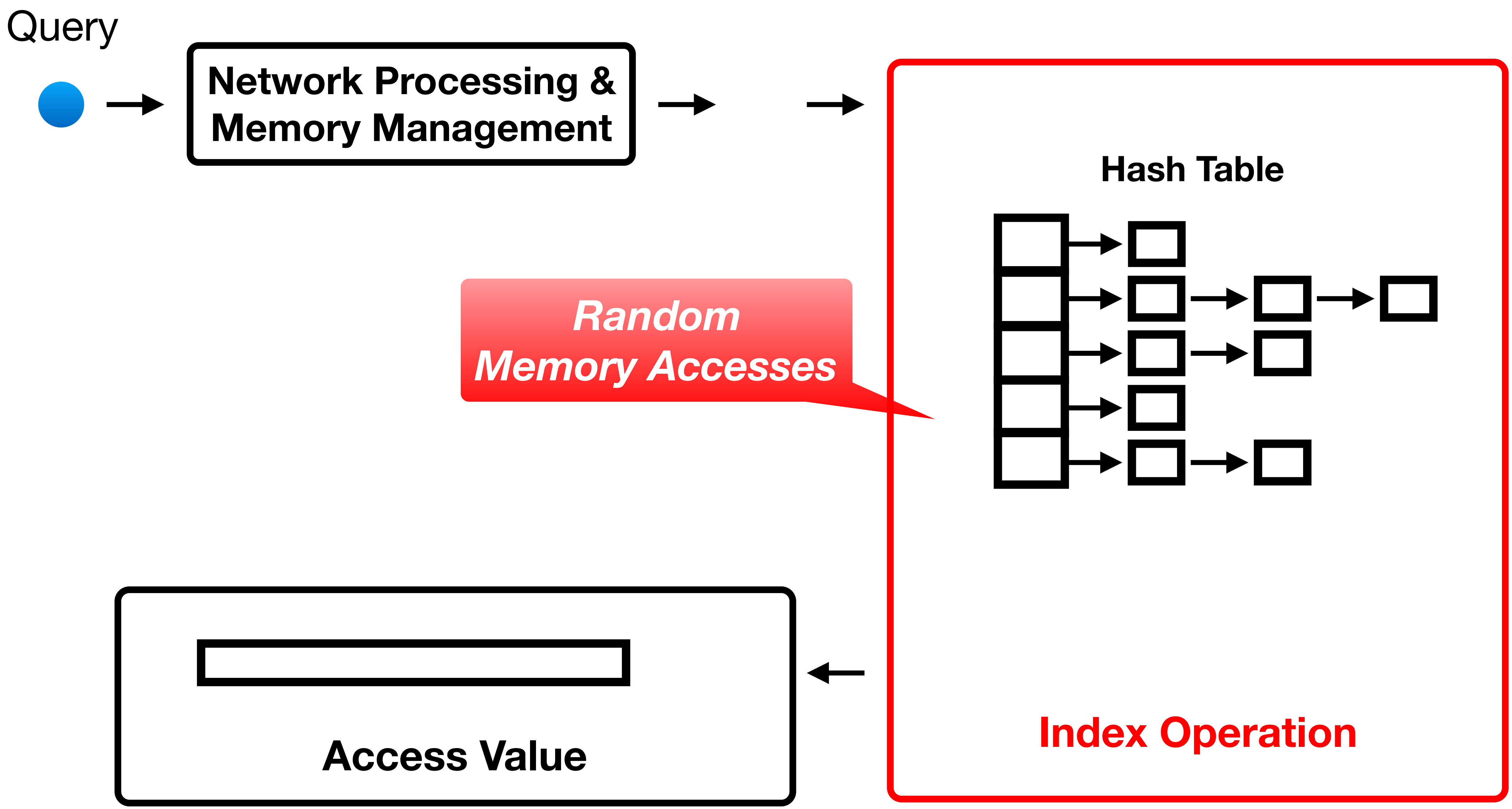


Where does time go in KV-Store MICA [NSDI'14]

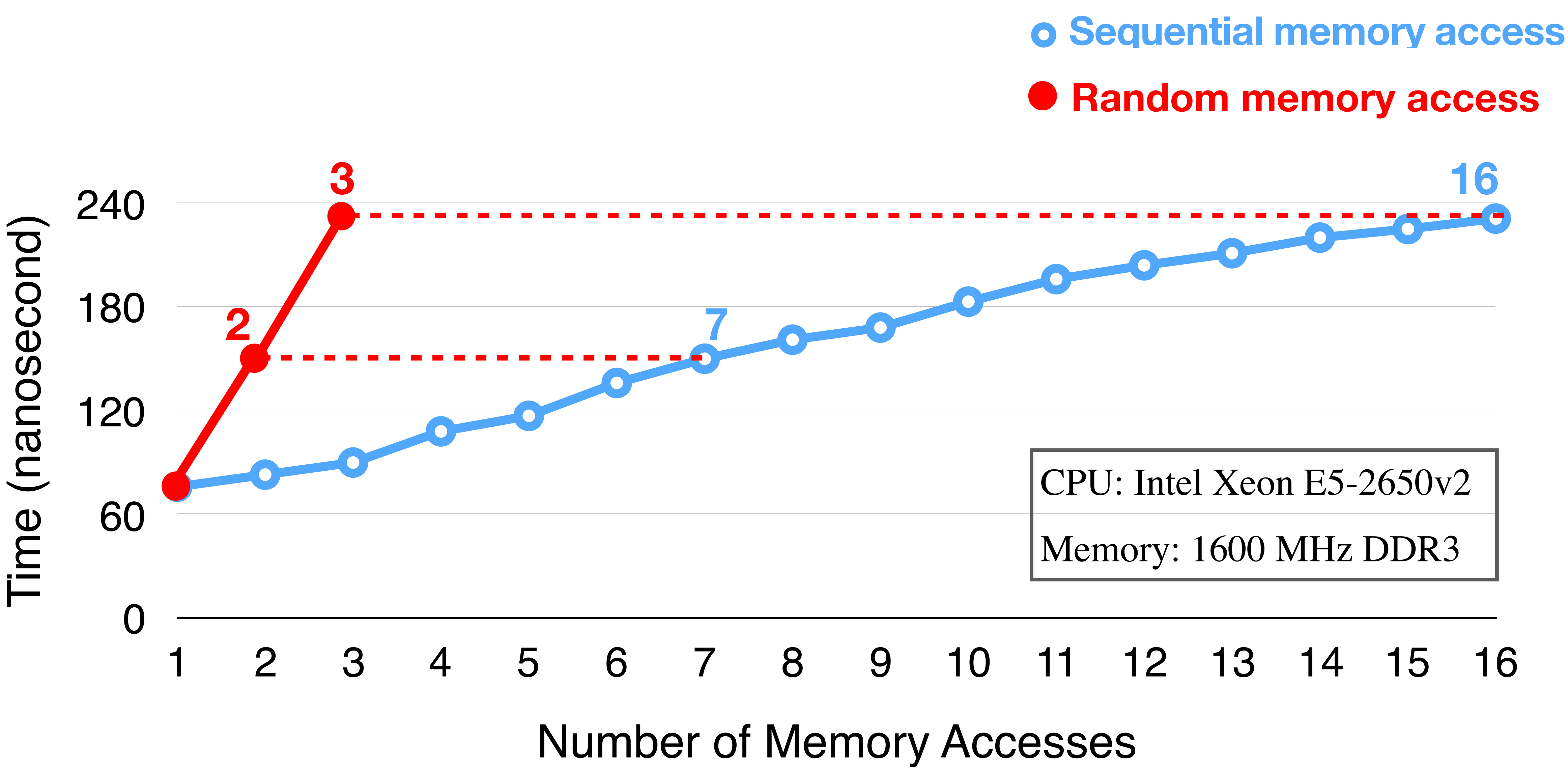


Index operation becomes one of the major bottlenecks

Random Memory Accesses in In-Memory Key-Value Stores



Random Memory Accesses of Indexing are Expensive



Inabilities for CPUs to Accelerate Random Memory Accesses

1. **Cache**
   - Working set is large (~100 GB), CPU cache is small (~10 MB)
2. **Prefetch**
   - Not easy to predict next memory address
3. **Multithreading**
   - Limited number of hardware threads
   - Limited number of Miss Status Holding Registers (MSHRs)

CPU spends a large portion of its time idling, waiting for data

Mega-KV addresses two issues: large number of queries and random memory access delay

Network Processing → **DPDK**, Multiget, UDP

Memory Management → Bitmap, Optimistic concurrent access

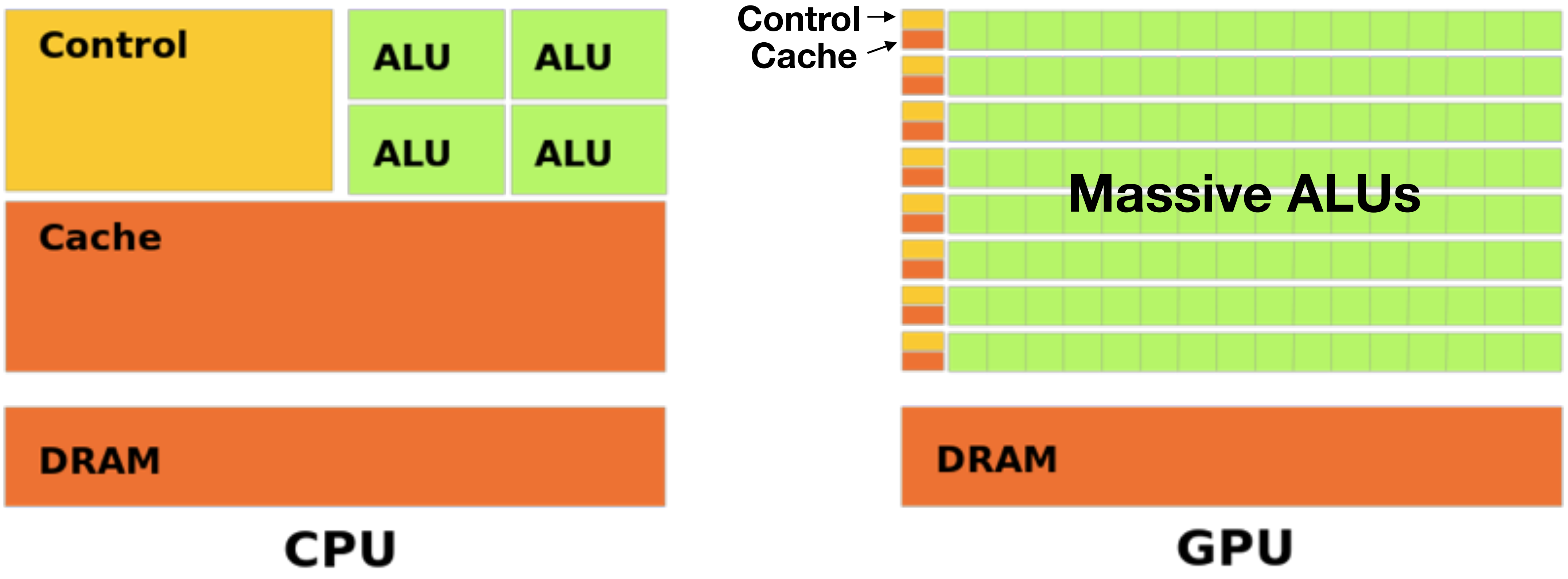
Index Operation → **To accelerate it by GPUs**

Access Value → Prefetch

The Goal of Mega-KV

- **Throughput** is the critical issue in big data environment
  - Throughput measures the capability of a key-value store system to process a growing amount of queries on an increasingly large data set
- Acceptable in-memory key-value store latency
  - < **1 millisecond**, e.g. Facebook, Amazon, …
- Our goal: **Maximize throughput subject to an acceptable latency**

CPU vs. GPU

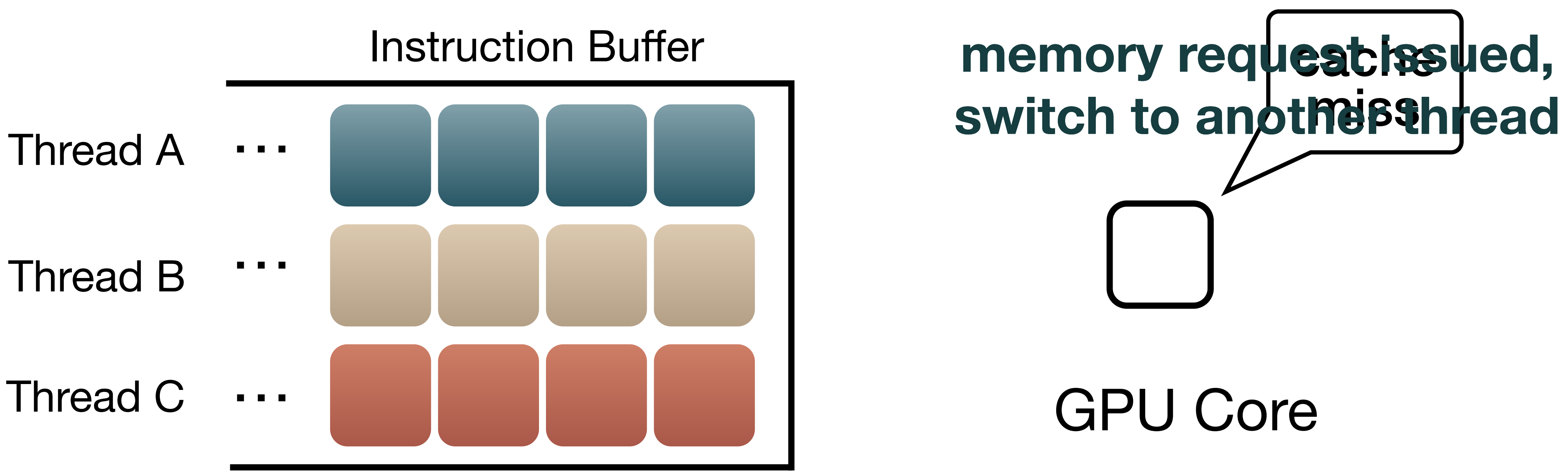


Intel Xeon E5-2650v2:
2.3 billion Transistors
8 Cores
59.7 GB/s memory bandwidth

Nvidia GTX 780:
7 billion Transistors
2,304 Cores (12 SMXs)
288.4 GB/s memory bandwidth

Two Advantages of GPUs for Key-Value Stores

1. **Massive Processing Units to Address Large Number of Concurrent Queries**
   - KV Store — simple independent memory access operations
   - GPU — thousands of cores for parallel processing
2. **Massively Hiding Memory Access Latency**
   - KV Store — random memory accesses in index operations
   - GPUs can effectively hide memory access latency with massive hardware threads and zero-overhead thread scheduling (a GPU hardware support)



Basic Design of Mega-KV

Network processing, Memory management

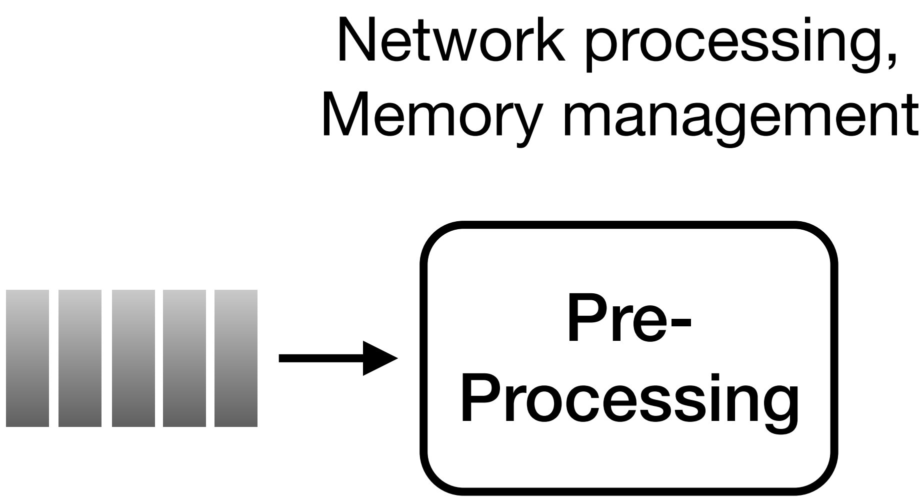
Index Operations

Read & Send Value

RX → **Pre-Processing** → **GPU Processing** → **Post-Processing** → TX

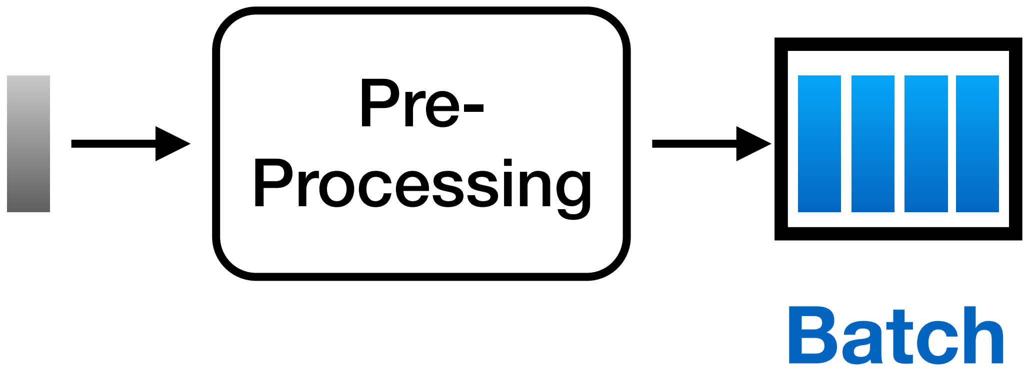
DPDK **DPDK**

Basic Design

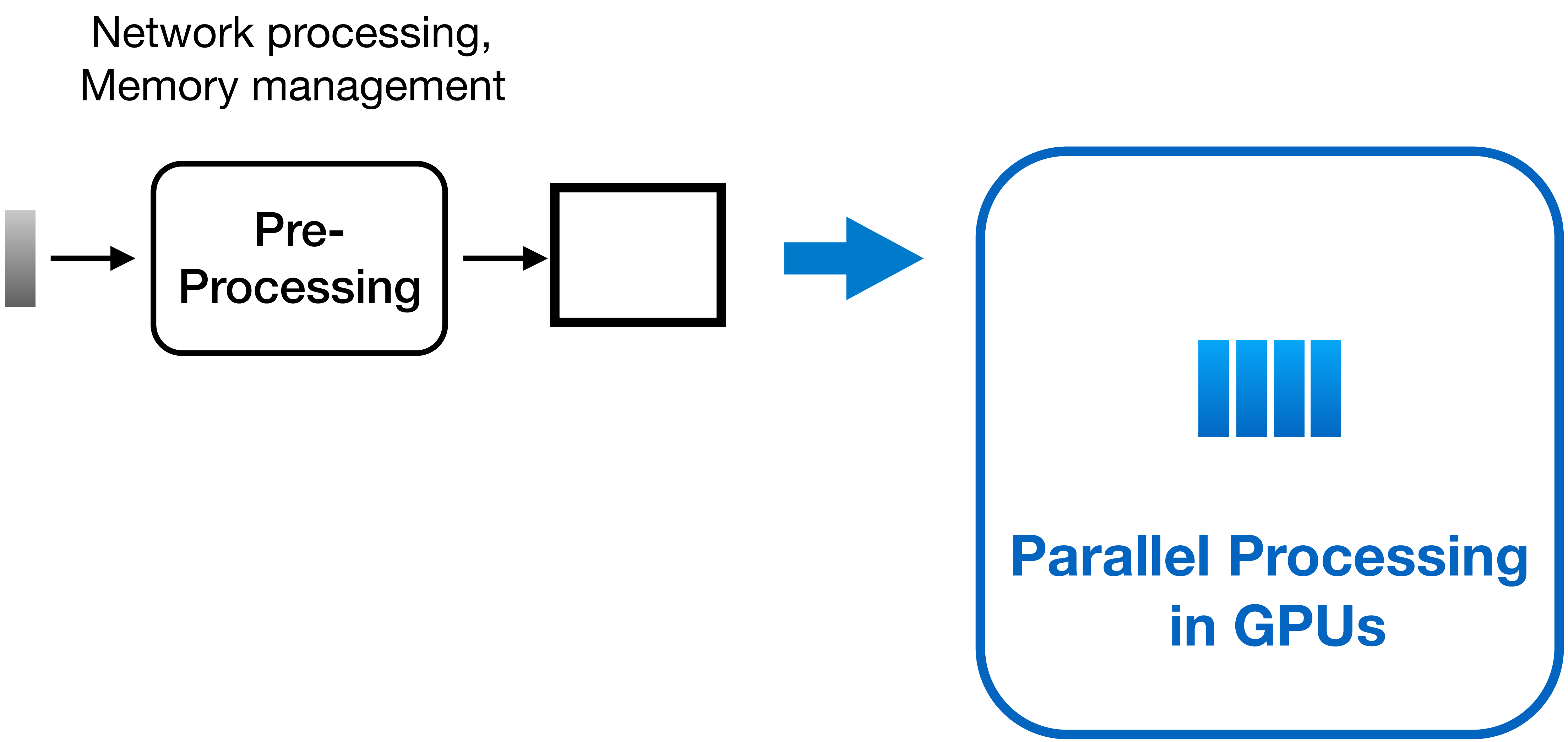


Basic Design

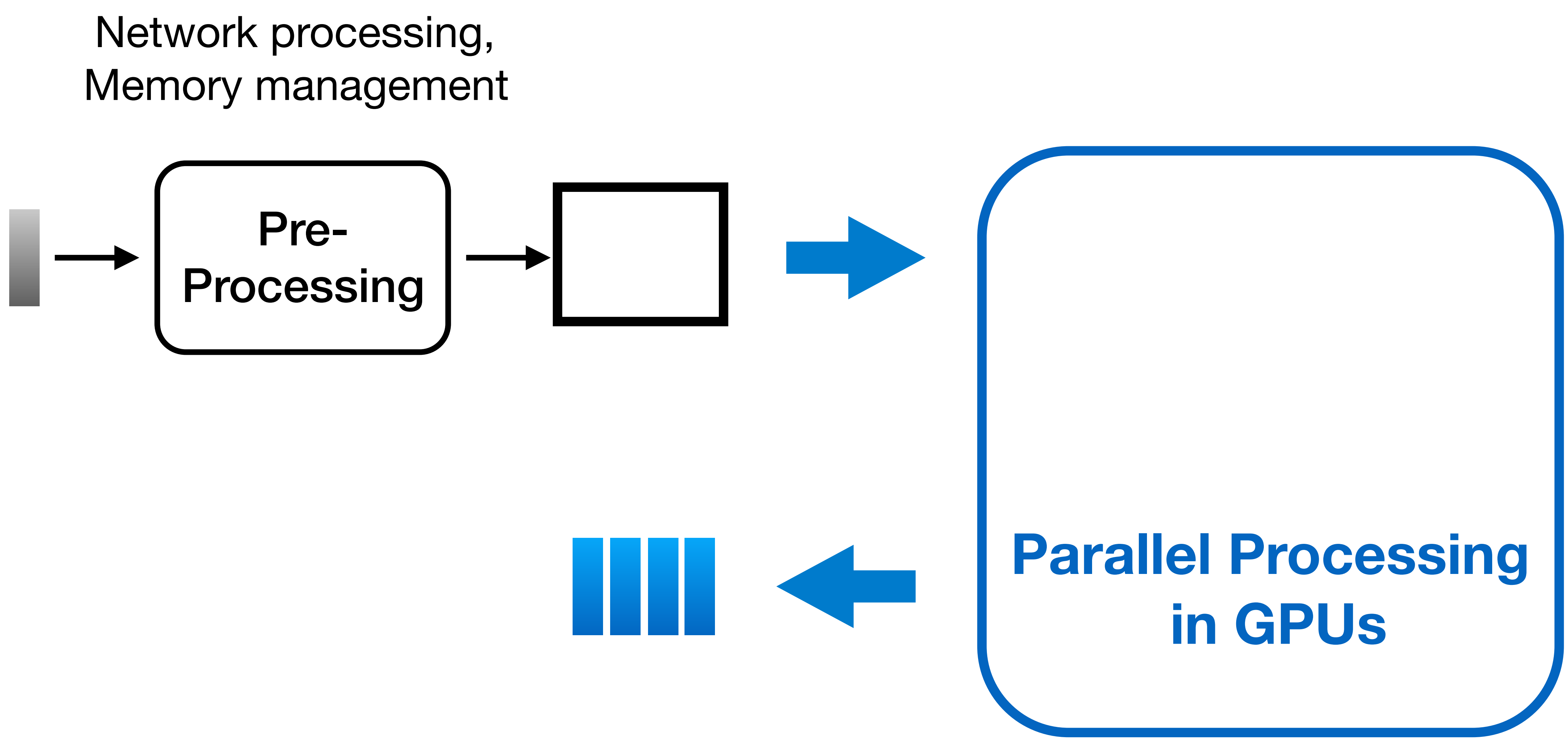
Network processing,
Memory management



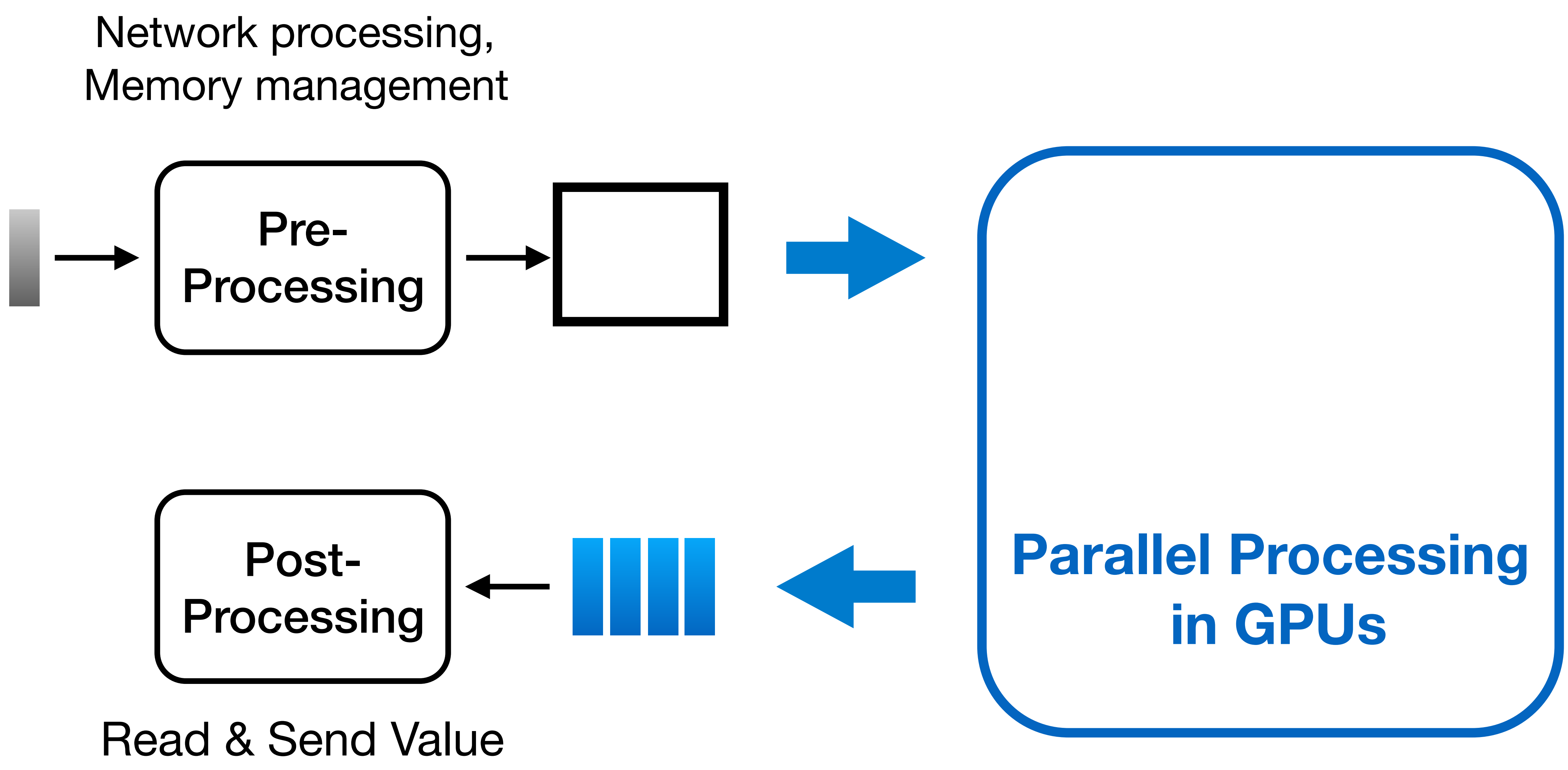
Basic Design



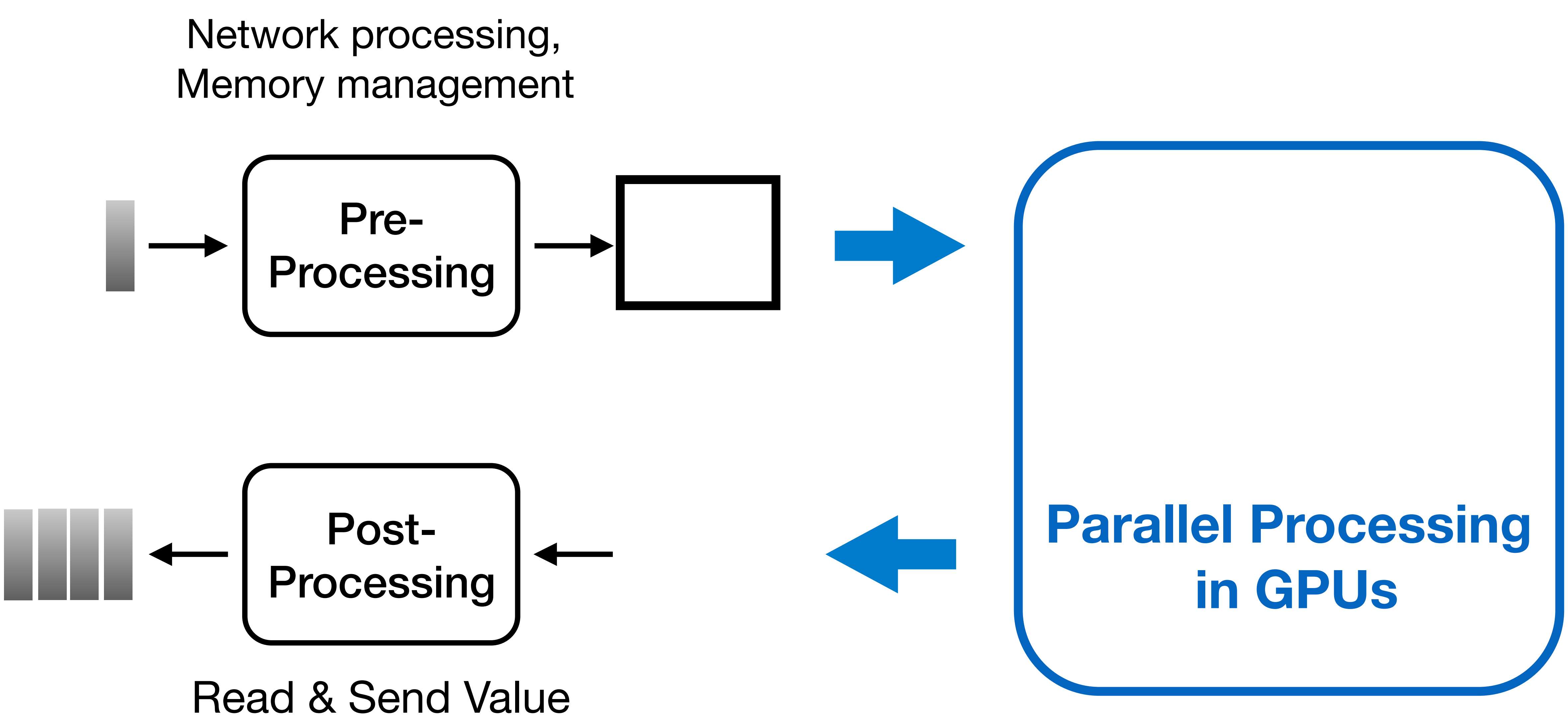
Basic Design



Basic Design



Basic Design



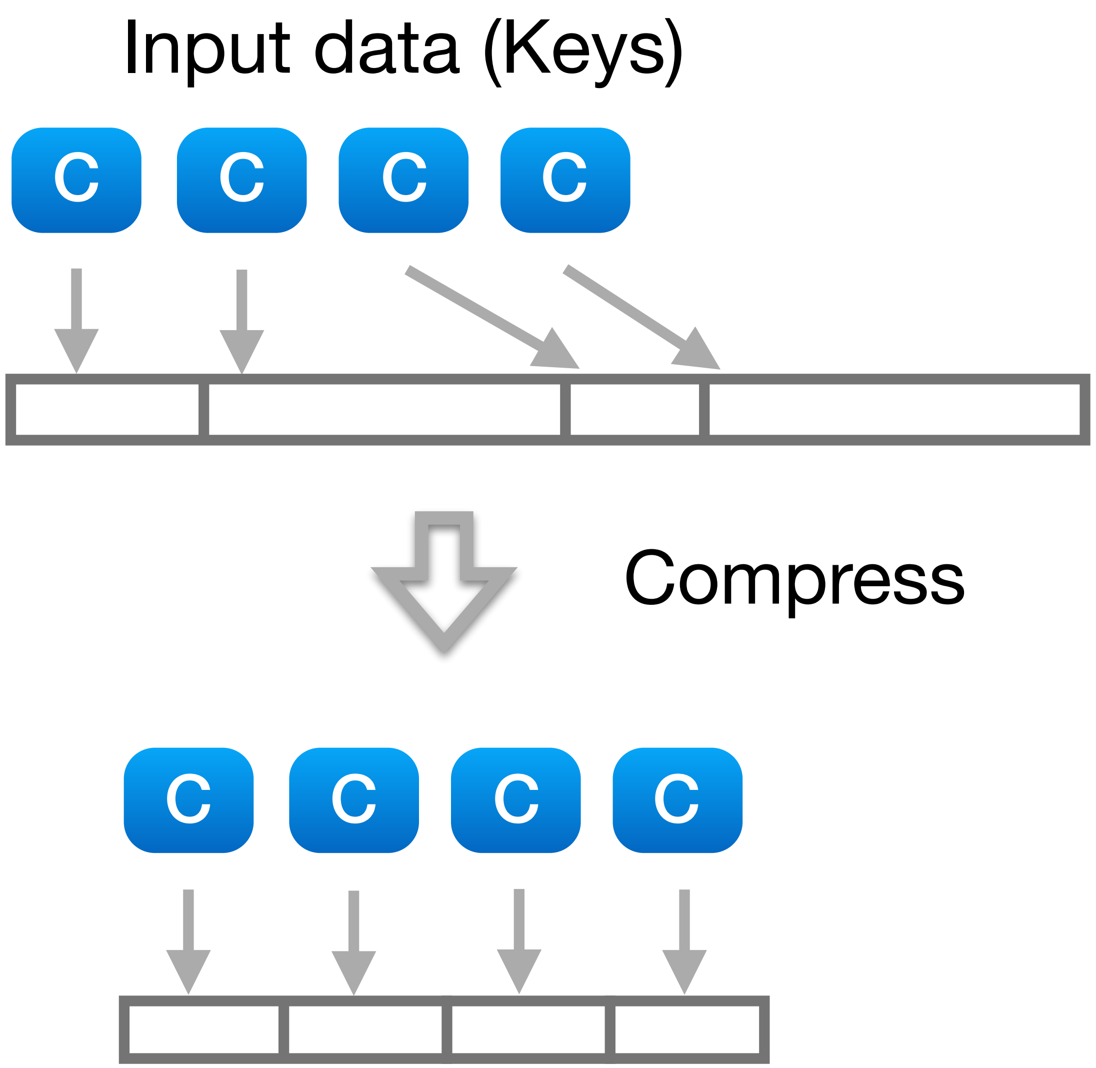
Challenges of Offloading Index Operations to GPUs

1. GPUs' memory capacity is small: ~10 GB
   - Working set may be hundreds of gigabytes

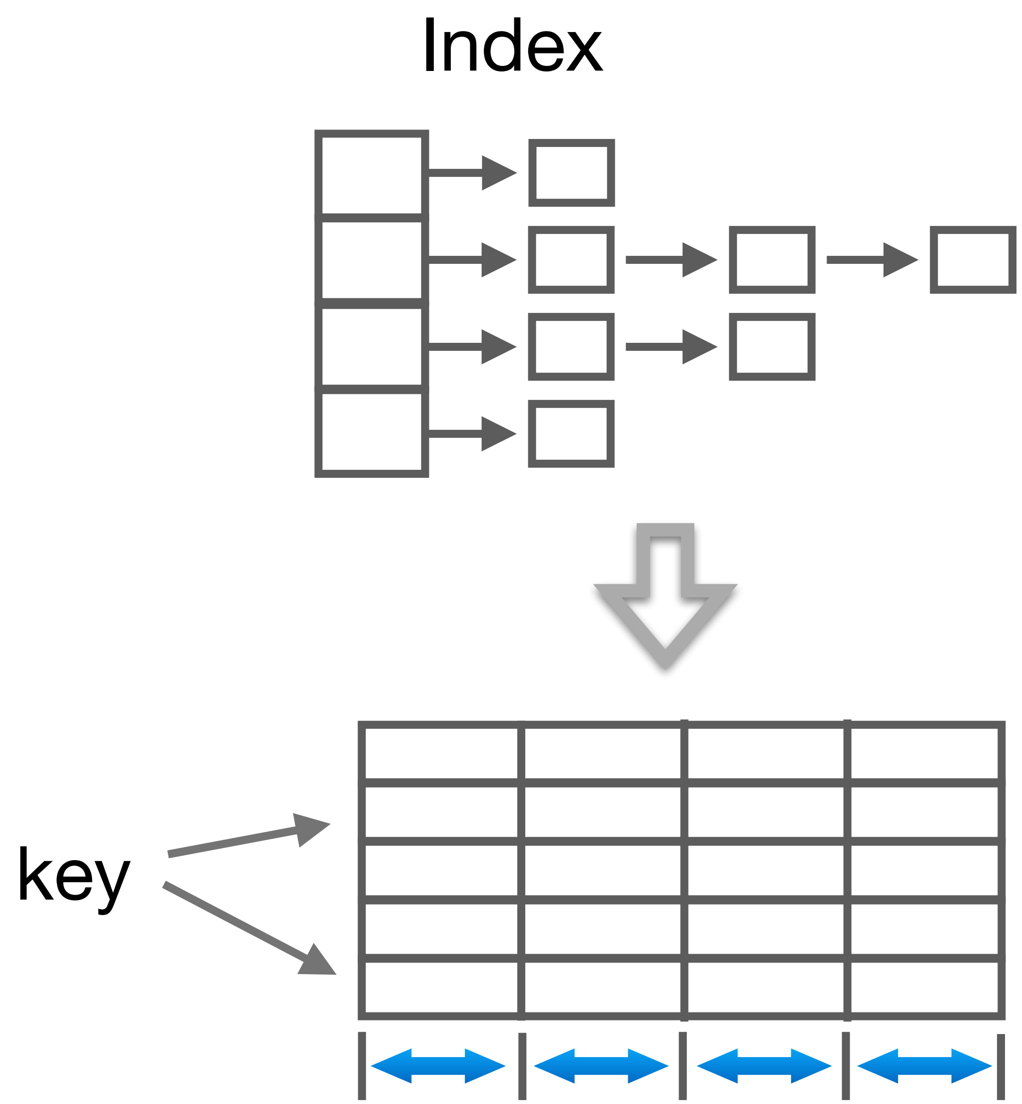
2. Low PCIe bandwidth
   - PCIe is generally the bottleneck of GPUs if large bulk of data needs to be transferred

3. Handling variable-length data is inefficient for GPUs
   - Imbalance load between GPU cores

Our Approach



Compressed fixed-length signatures

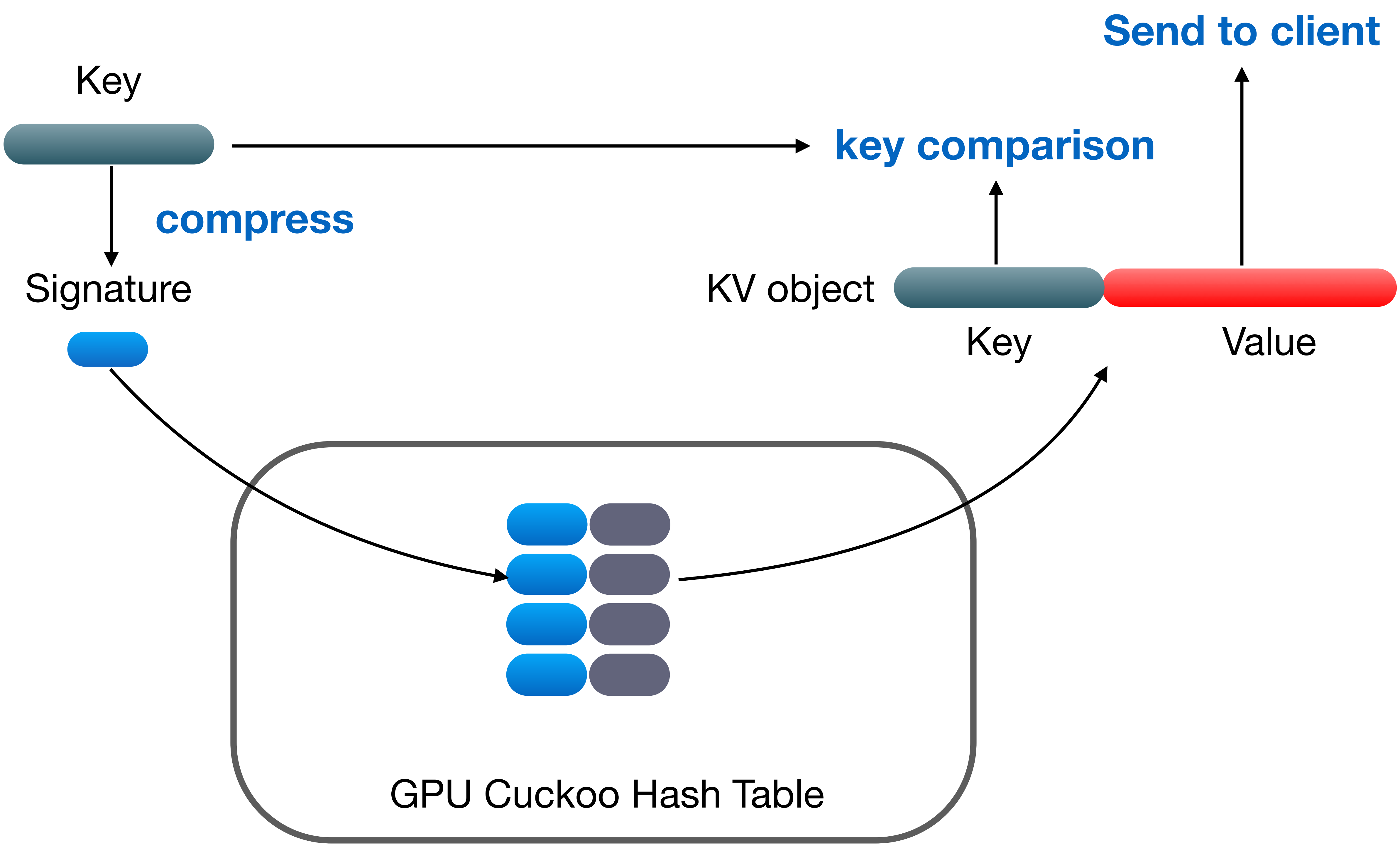


GPU optimized cuckoo hash table that stores key signatures and value locations

Address challenges 2, 3
(PCIe bandwidth and variable length data)

Address challenges 1, 3
(GPU memory capacity and variable length data)

Our Approach: Search Index



Evaluation - Hardware Setup

CPU:

Intel Xeon E5-2650v2 octa-core, 2.6GHz

Total **16** CPU cores

GPU:

Nvidia GTX 780, 2304 cores, 863MHz

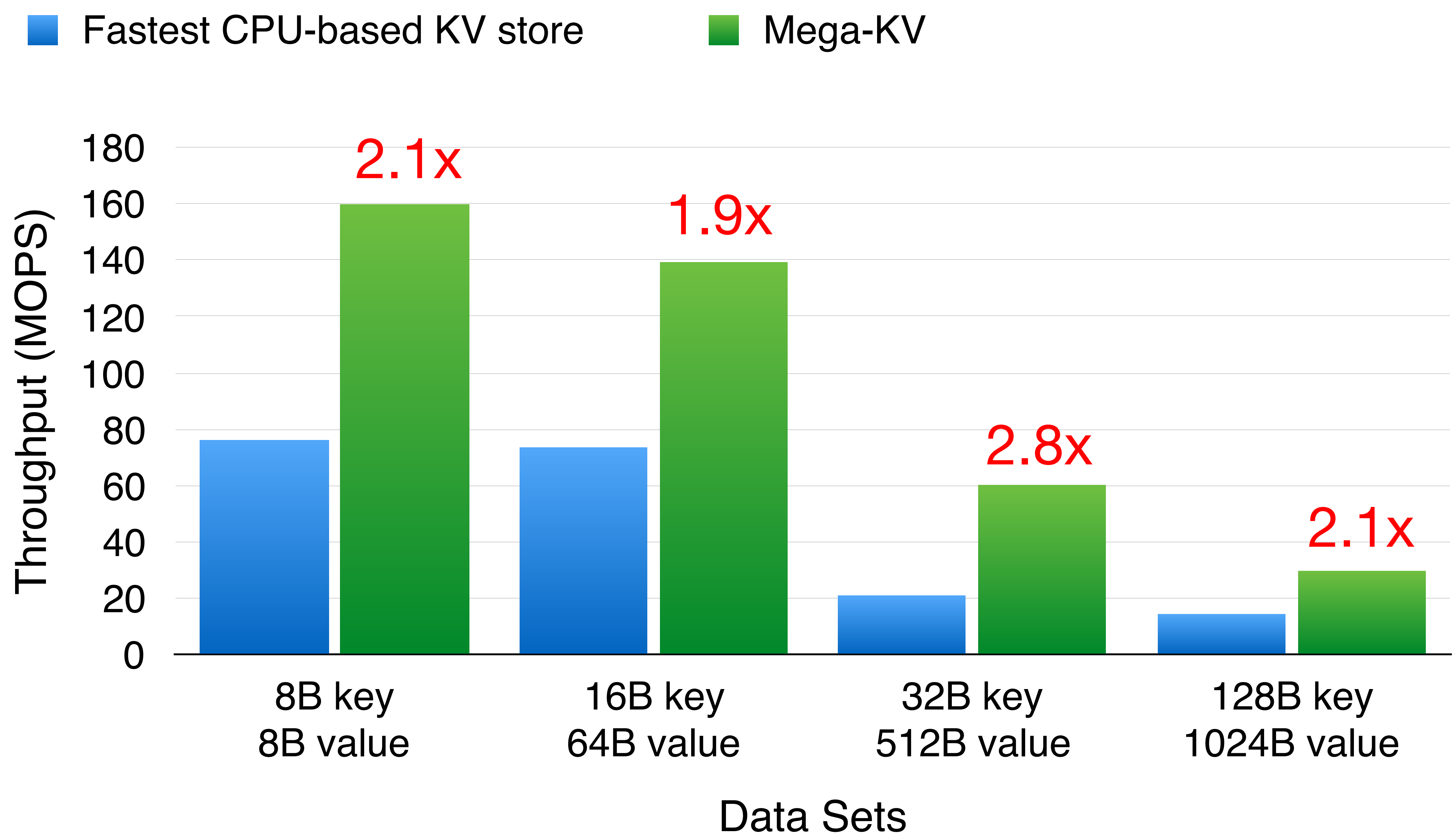
Total **4608** cores

NIC:

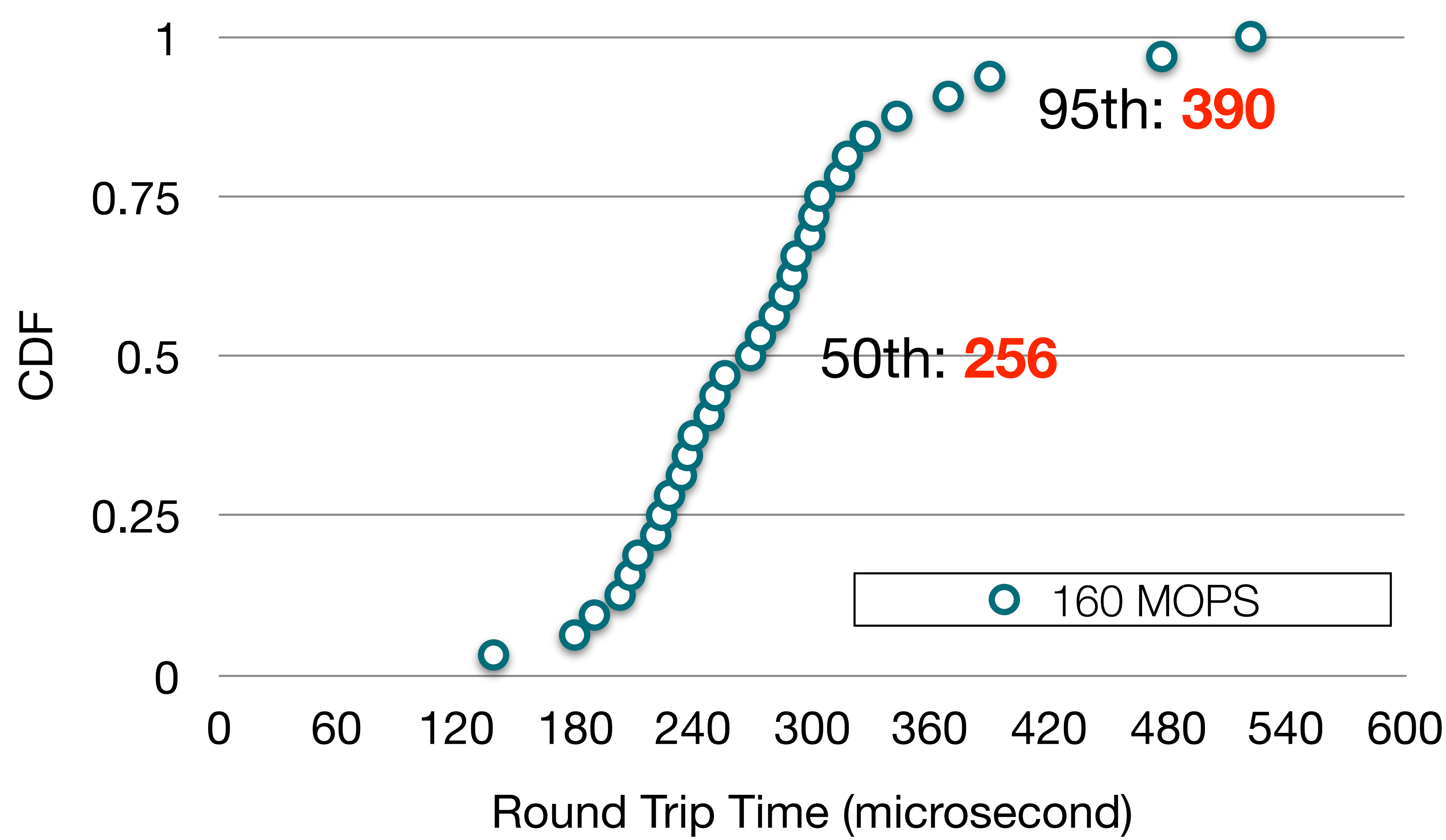
Intel dual-port 10Gbps NIC

Total **40** Gbps

Reaching a Record High Throughput



Latency



Compared with Facebook

1,200 (95th)

300 (50th)

Accelerating the Network I/O of Snort with DPDK

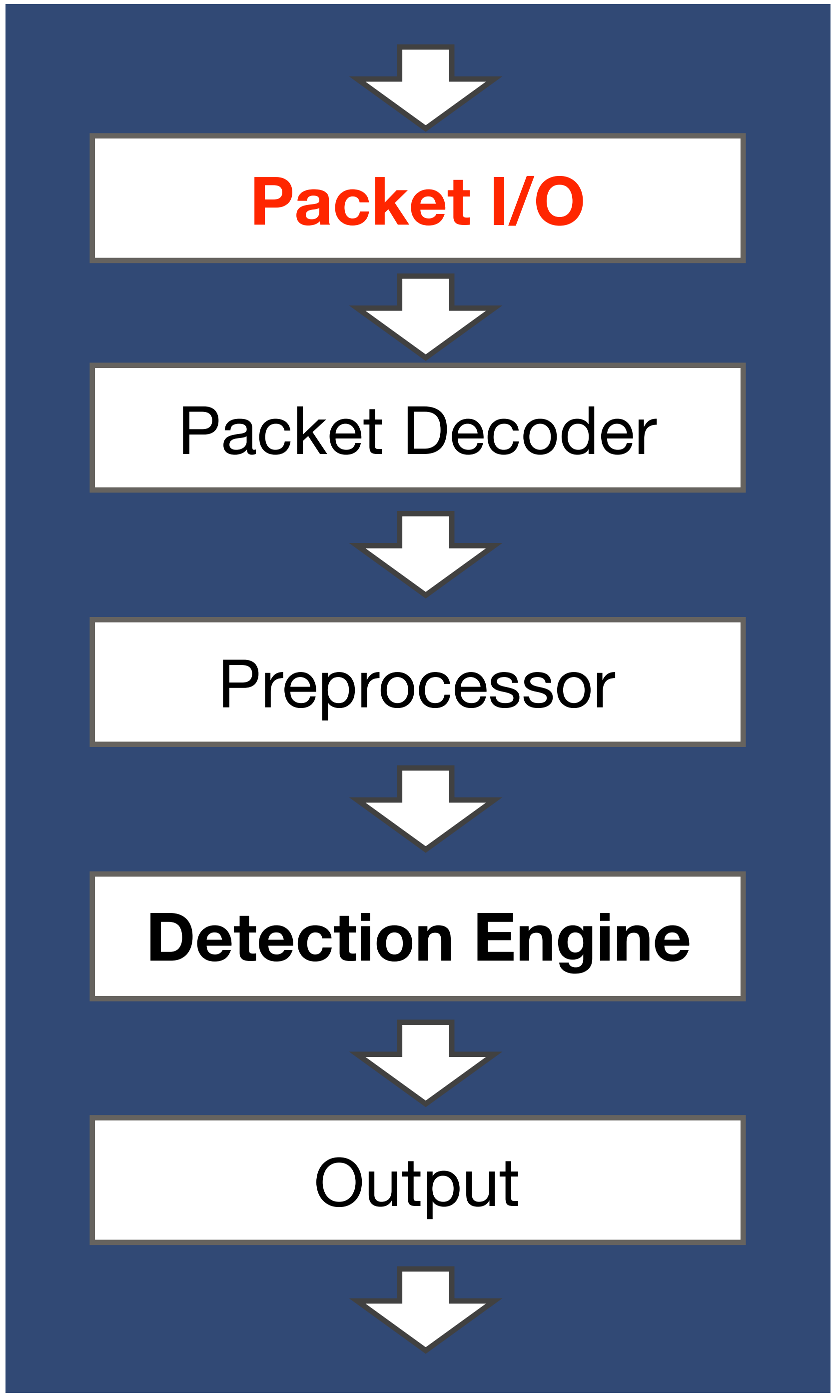
Snort

- Snort is a multi-mode packet analysis tool
  - Sniffer
  - Packet Logger
  - Forensic Data Analysis tool
  - Network Intrusion Detection System

- Snort is able to perform network traffic analysis both in real-time and for forensic post processing

- Snort "Metrics"
  - Fast (High probability of detection for an attack on high speed networks)
  - Configurable (Easy rules language, many reporting/logging options)

Snort Architecture



Incapability in Handling 10Gbps Network Traffic

Cycles Needed in Snort

1,200 + **400 - 25,000…**

Packet I/O | **Detection, etc.**

Your Budget

1,400

10Gbps, min-sized packets, quad-core 2.66GHz CPUs

(in x86, cycle numbers are from RouteBricks [Dobrescu09] and PacketShader[Han10])

Incapability of Snort for High Speed Networks

- Snort was designed to detect attacks on 100Mbps links
    - Current network speed reaches 10Gbps and 40Gbps
    - Snort becomes incapable of detecting intrusions on current backbone and data center network
- Accelerate network I/O with DPDK
    - Snort 2.9 introduces the Data Acquisition library (DAQ) for packet I/O
    - Current supported: pcap, AF_PACKET, netmap, ipfw
    - Our work: add DPDK support in DAQ

Opportunities from DPDK

Cycles Needed	1,200	+	400 - 25,000…
	Packet I/O		Detection, etc.

DPDK

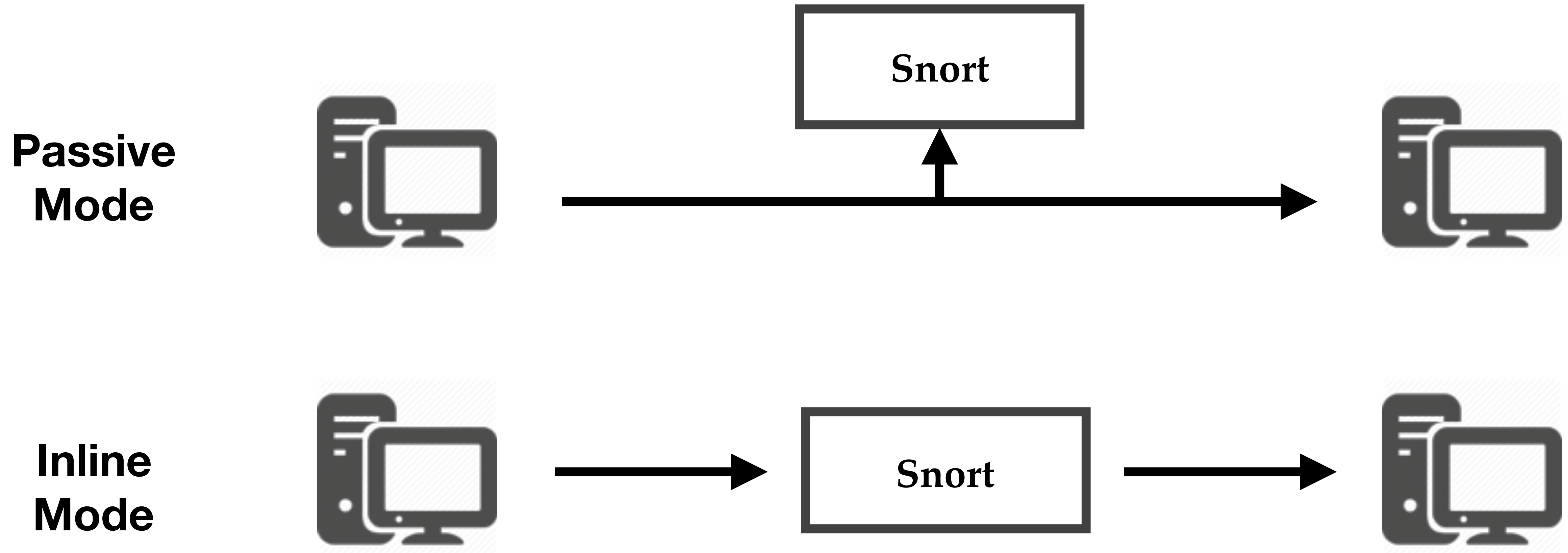
80

EXPERIMENTAL SETUP

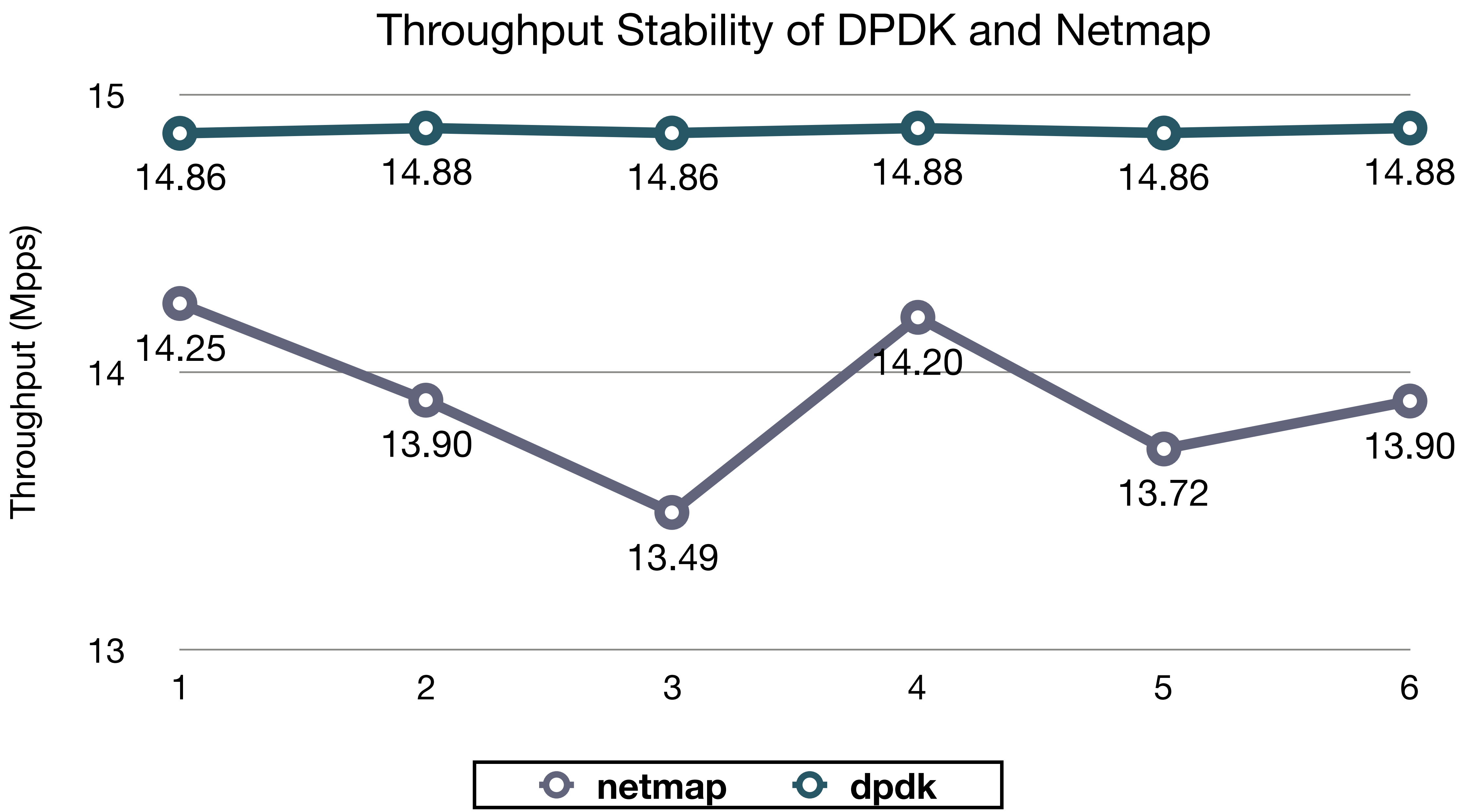
- Hardware
  - CPU: Intel(R) Xeon(R) CPU E5-2650 v3
  - NIC: 82599ES 10-Gigabit SFI/SFP+ Network Cards
- Software
  - Linux 3.19.0
  - Snort 2.9.8.0
  - DPDK 2.1.0

Snort Modes

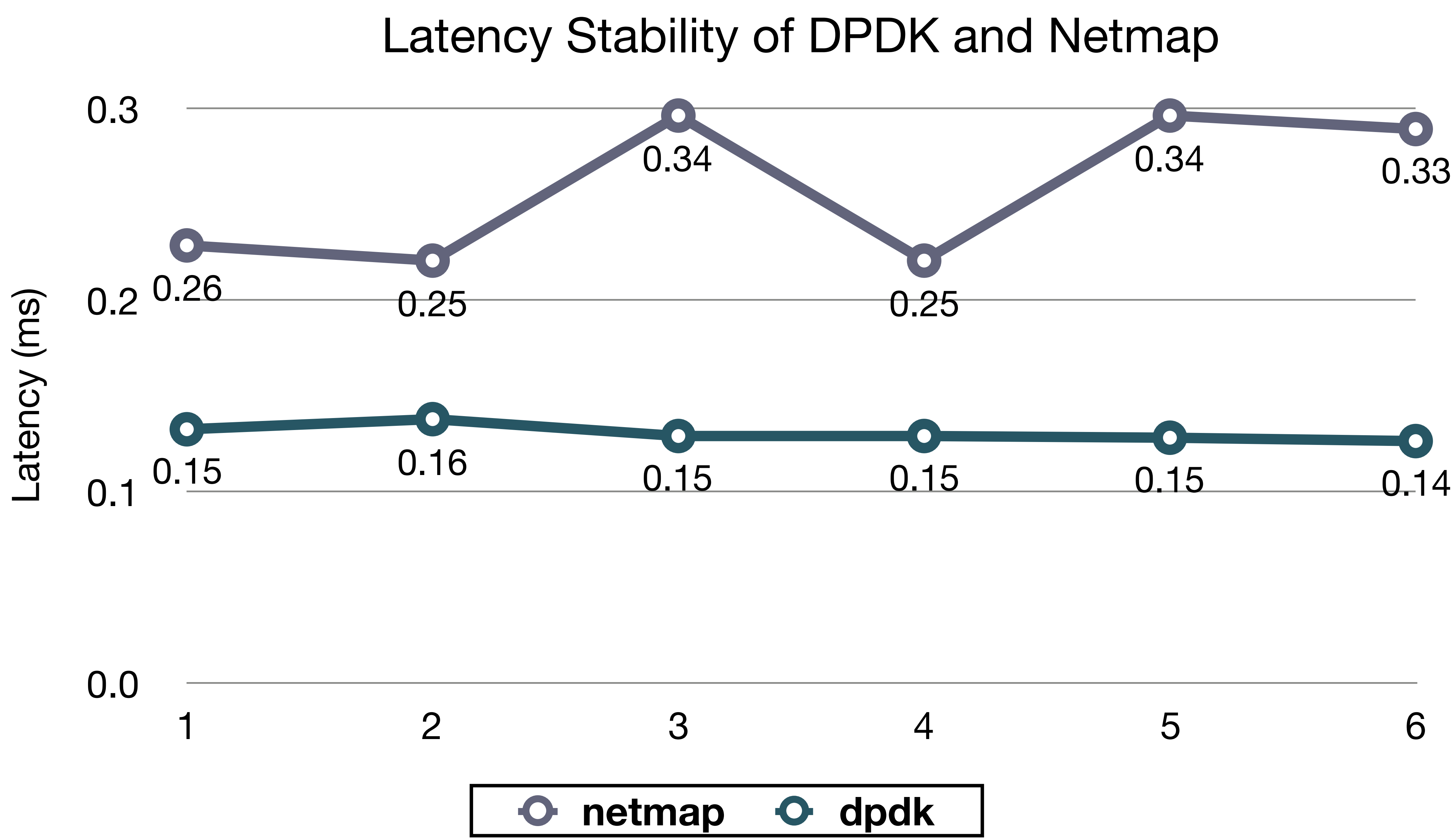
- Two Snort modes: Passive Mode and Inline Mode
  - Snort is bypassed in the Passive Mode
  - Snort filters packets in the Inline Mode



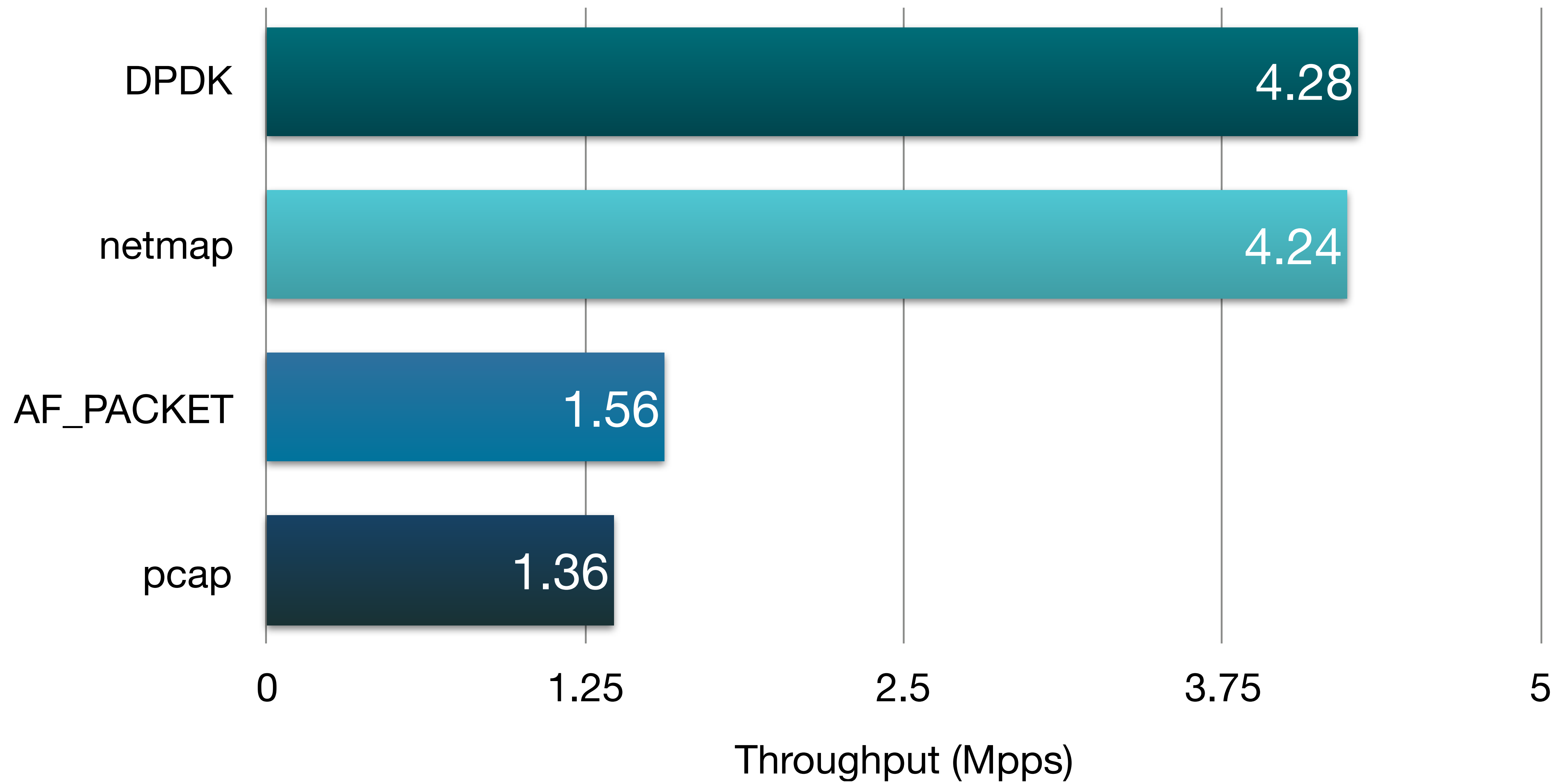
Snort Throughput (Inline Mode w/o DPI)



Latency (Inline Mode w/o DPI)



Snort Throughput (Passive Mode w/ DPI)



Summary

- Mega-KV provides the highest throughput
    - Fastest in-memory key-value store on commodity processors
    - More than 100x faster than Memcached
    - Open source at *http://kay21s.github.io/megakv/*

- Integrating DPDK into Snort
    - Improving the latency and throughput of Snort
    - For educational purpose: To be a course assignment in USTC

Thanks!

DPDK

DATA PLANE DEVELOPMENT KIT