LightStore: Software-defined Network-attached Key-value Drives

Chanwoo Chung[‡], Jinhyung Koo, Junsu Im, Arvind[‡], and Sungjin Lee

DGIST and MIT[‡]

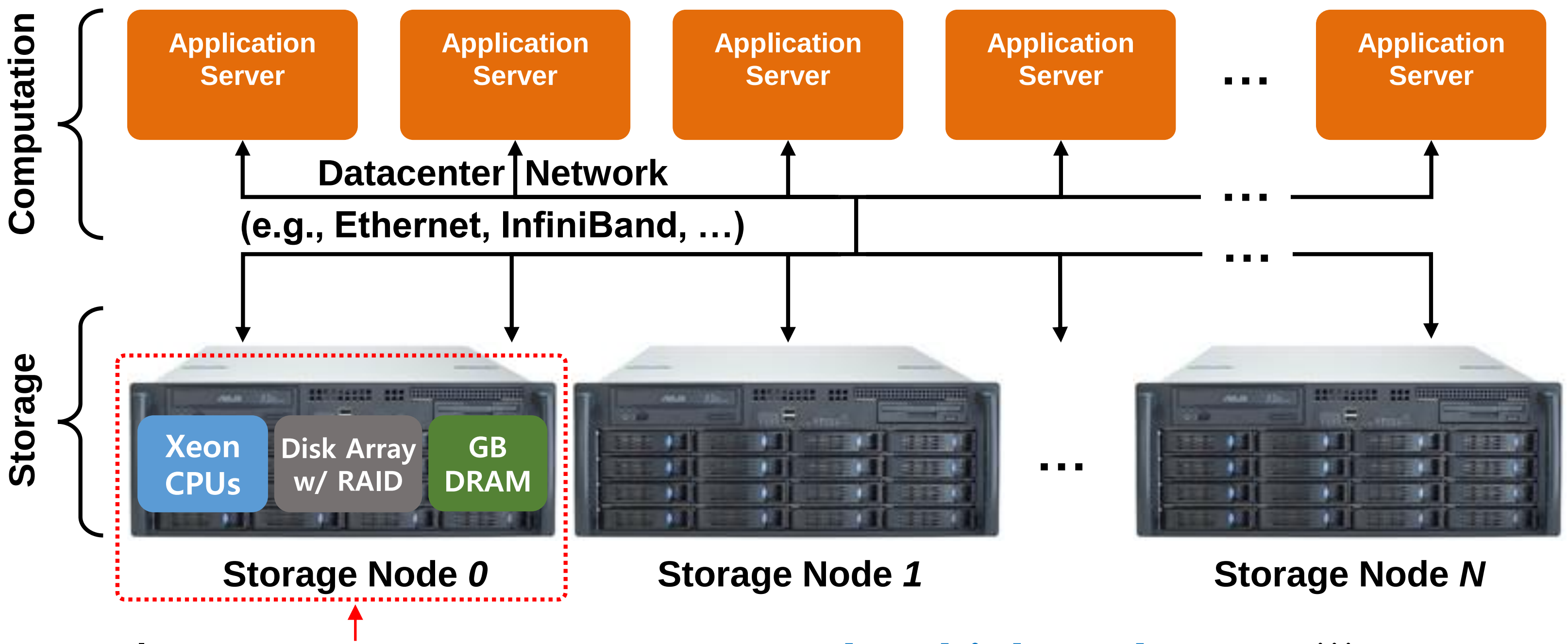
NVRAMOS '19

2019.10.24

대구경북과학기술원
Daegu Gyeongbuk
Institute of Science & Technology

DATA-INTENSIVE
COMPUTING SYSTEMS
LABORATORY

Motivation



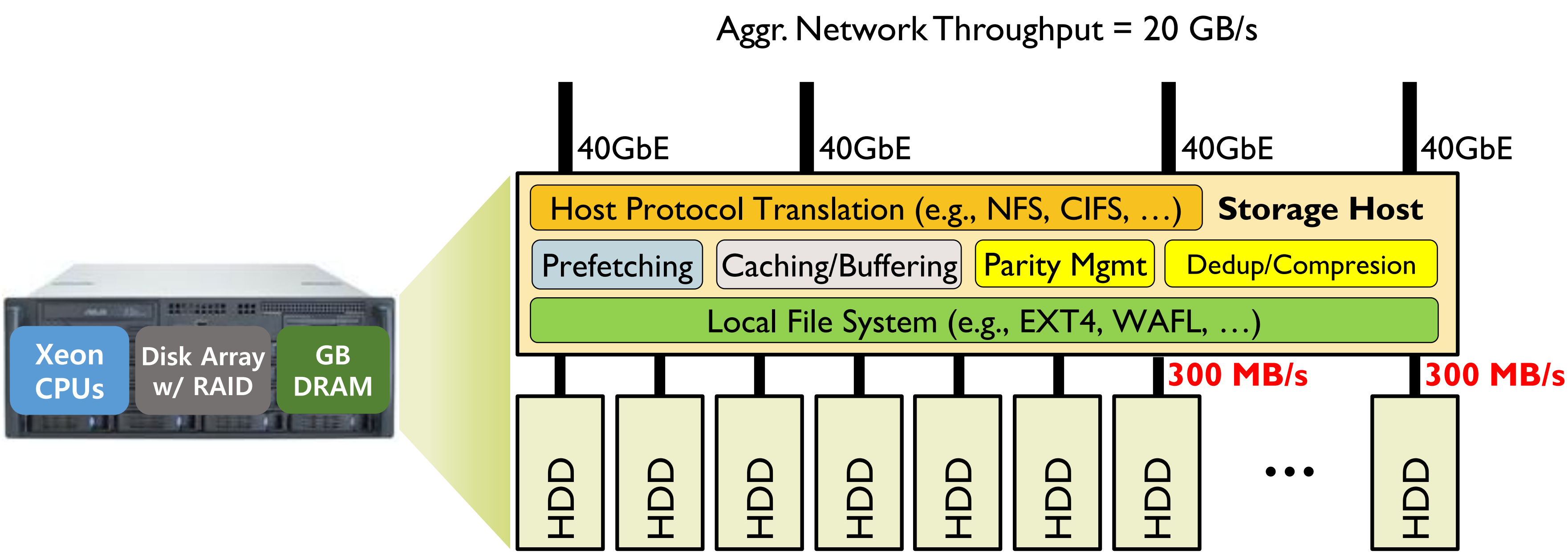
It is not mere storage – it is **another high-end server**!!!

High-end Xeon CPUs
Several GBs of DRAM
An array of SSDs
Large form-factor
…

Power Hungry (e.g., 1700 W)
Expensive (e.g., \$2~40,000 w/o SSDs)
Large Volume (e.g., 2-4 U)
High TCO (e.g., Cooling)
…

This Architecture is Invalid with SSDs

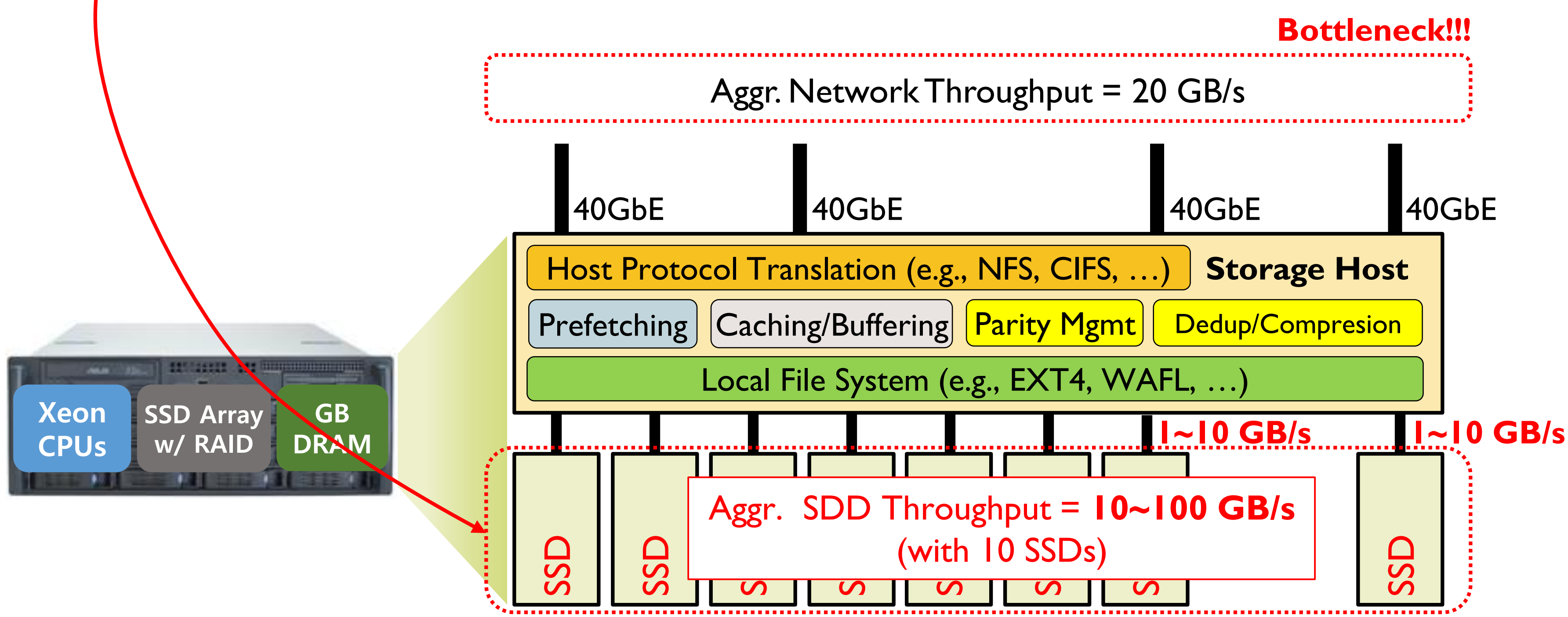
- **HDD is slow – require large DRAM and array of disks**
  - 10 ms latency & 100~300 MB/s throughput
- **HDD is dumb – the host system makes it smarter**
  - Xeon CPUs with advanced algorithms



This Architecture is Invalid with SSDs

- **HDD is slow – require large DRAM and array of disks**
  - 10 ms latency & 100~300 MB/s throughput

  ➡ SSDs are not a bottleneck → Network/CPU are new bottlenecks
- **HDD is dumb – the host system makes it smarter**
  - Xeon CPUs with advanced algorithms



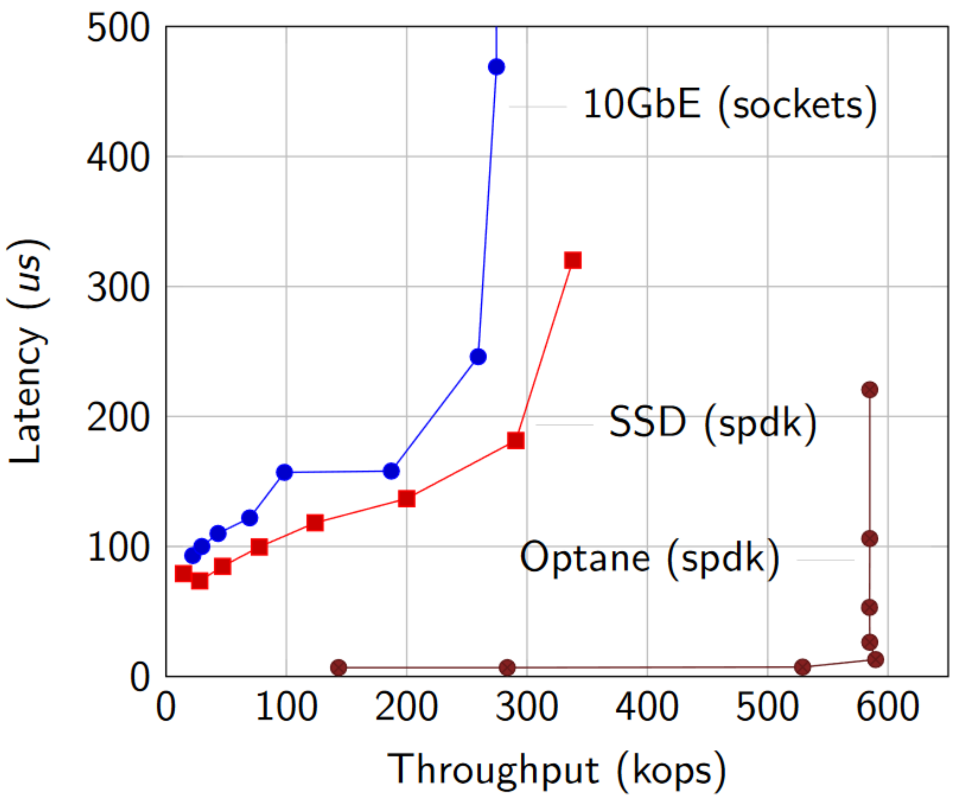
Network/SSD Performance Trend in AFA

		EMC XtremIO	NetApp SolidFire	HPE 3PAR	Hynix AFA
SSD Array	Capacity	36~144TB	46TB	750TB	522TB
	# of SSDs	18~72	12	120	576
	Aggr. Throughput*	18~72 GB/s	12 GB/s	120 GB/s	576 GB/s
Network	Ports	4~8x 10Gb iSCSI	2x 25Gb iSCSI	4~12x 16Gb FC	3x Gen3 PCIe
	Aggr. Throughput	5~10 GB/s	6.25 GB/s	8~24 GB/s	48 GB/s

※ Aggr. SSD throughput was estimated assuming each SSD offers 1GB/s throughput

- **Supported by the latest works**
  - K. Kourtis et al., "Reaping the performance of fast NVM storage with uDepot," *USENIX FAST '19*
  - J. Kim et al., "Alleviating Garbage Collection Interference through Spatial Separation in All Flash Arrays," *USENIX ATC '19*

Network/SSD Performance Trend in AFA



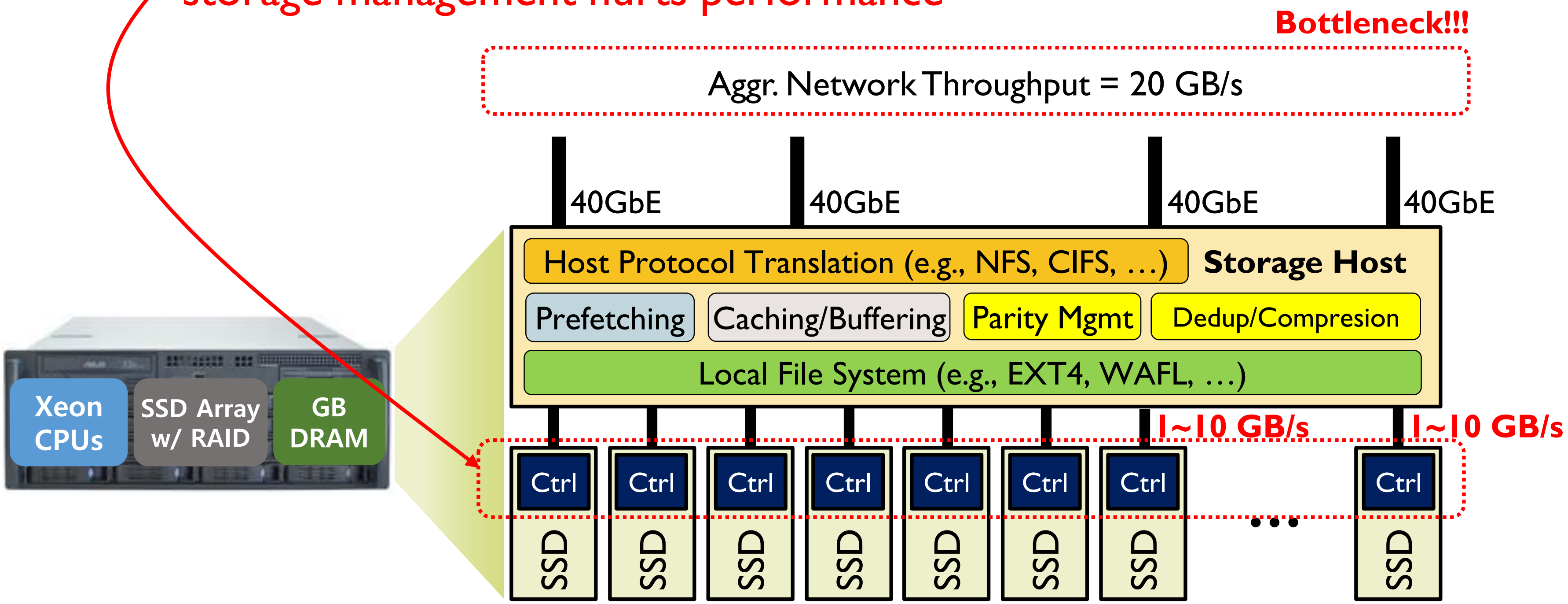
- **Supported by the latest works**
  - K. Kourtis et al., "Reaping the performance of fast NVM storage with uDepot," *USENIX FAST '19*
  - J. Kim et al., "Alleviating Garbage Collection Interference through Spatial Separation in All Flash Arrays," *USENIX ATC '19*

SSD is Not a Dumb Device

- **HDD is slow – require large DRAM and array of disks**
  - 10 ms latency & 100~300 MB/s throughput

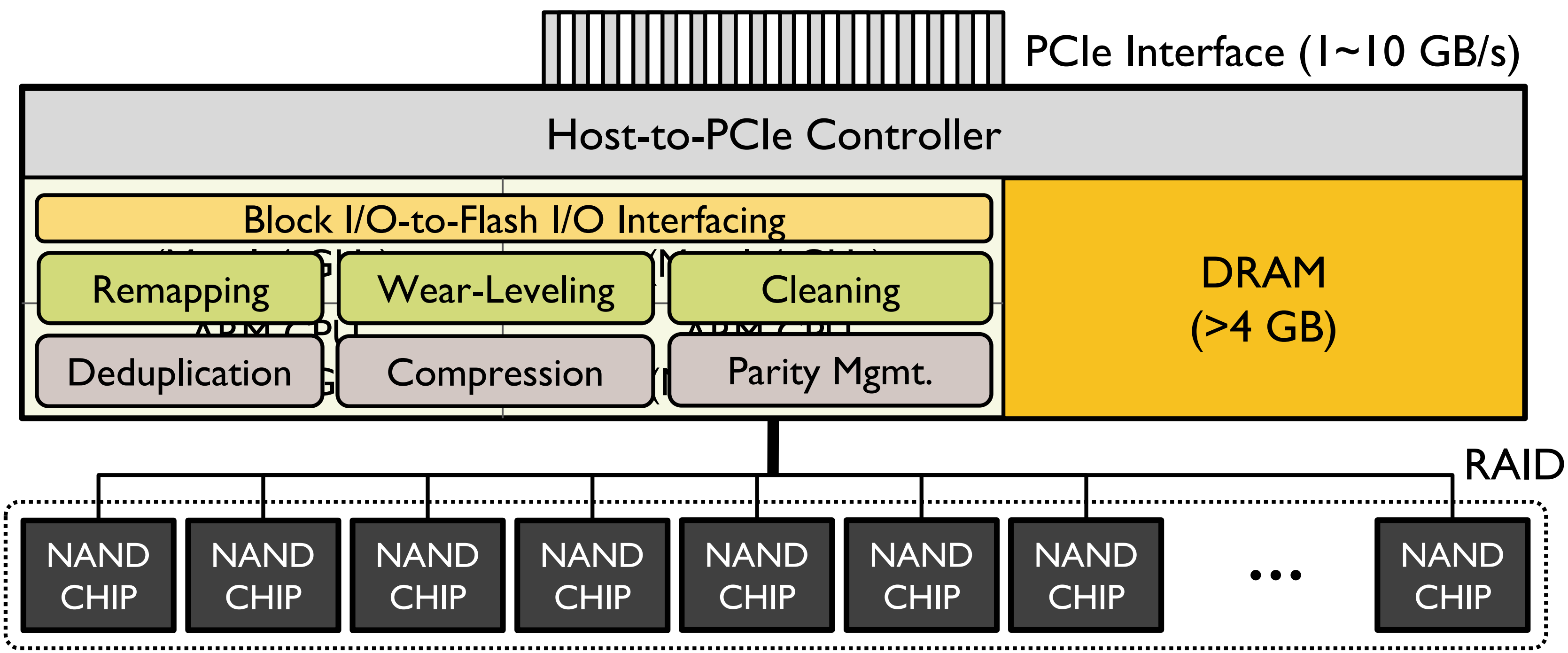
  ➡ SSDs are not a bottleneck → Network/CPU are new bottlenecks
- **HDD is dumb – the host system makes it smarter**
  - Xeon CPUs with advanced algorithms

  ➡ SSDs are smart enough, supporting many features → Duplicate storage management hurts performance

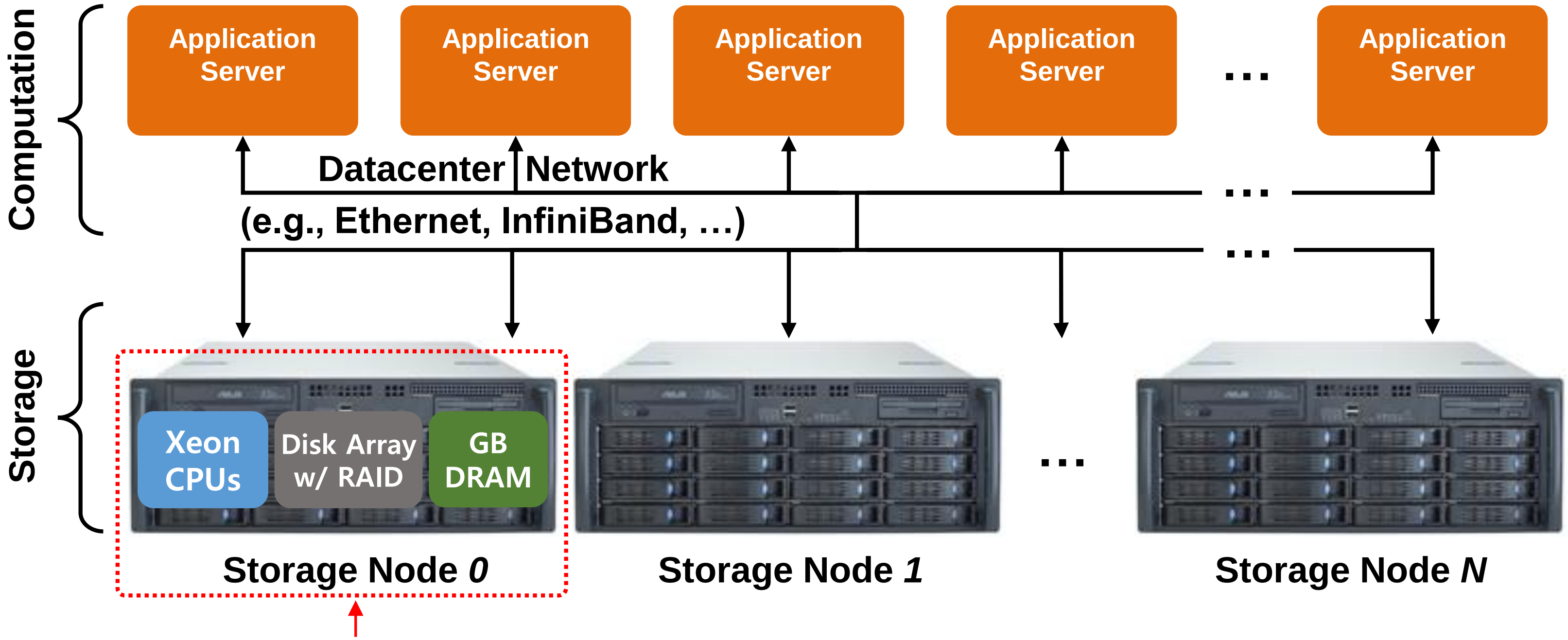


Let's Look into SSDs

- 4 embedded CPUs (ARM) running at 700 MHz to 1.4 GHz and > 1~16GB DRAM that a desktop PC had 10 years ago
- Those resources are required for running firmware (i.e., FTL)



1 + 1: Buy AFA and Get Miroserver!



Let's assume that this storage node has 8TB 72 SSDs (EMC XtremIO)

- # of ARM cores: 4 cores x 72 = 288 ARM cores
- Aggregate DRAM: 8 GB x 72 = 576 GB DRAM

Just for managing NAND flash

Q: Is this a storage node or a low-power microserver?

Possible Solutions?

- **Use simple SSD?**
  - Software Defined Flash (ASPLOS '14)
  - Application-managed Flash (USENIX FAST '16)
  - LightNVM (USENIX FAST '17)
  - → Network/CPU are still bottleneck

- **Use better SSD organization?**
  - SWAN (HotStorage '16; USENIX ATC '19)
  - → Still rely on power-hungry and expensive host

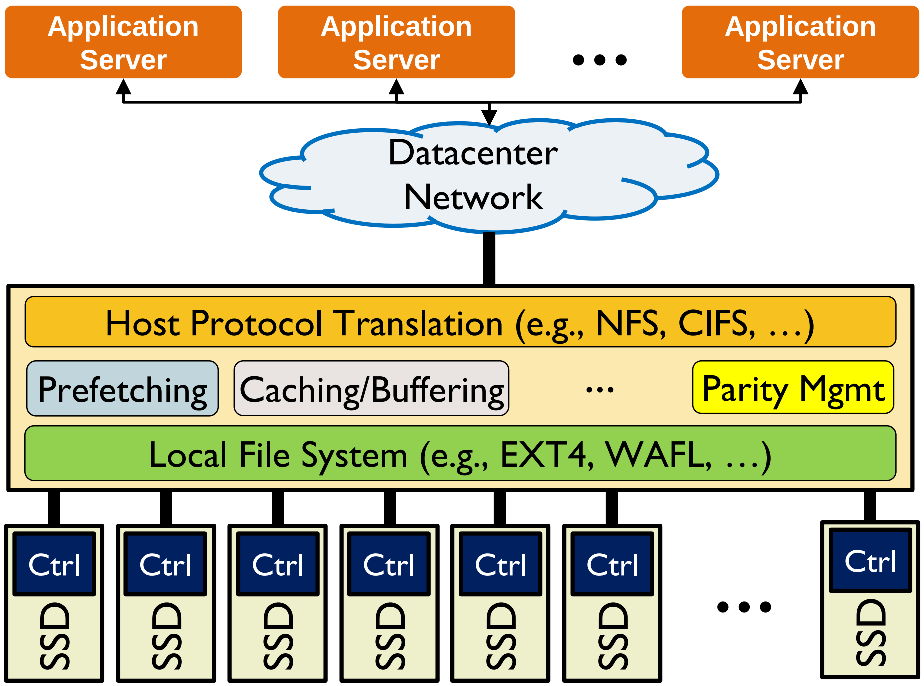
- **Any other solution?**

Index

- Motivation
- **Basic Idea**
- LightStore Software
- LightStore Controller
- LightStore Adapters
- Experimental Results
- Conclusion

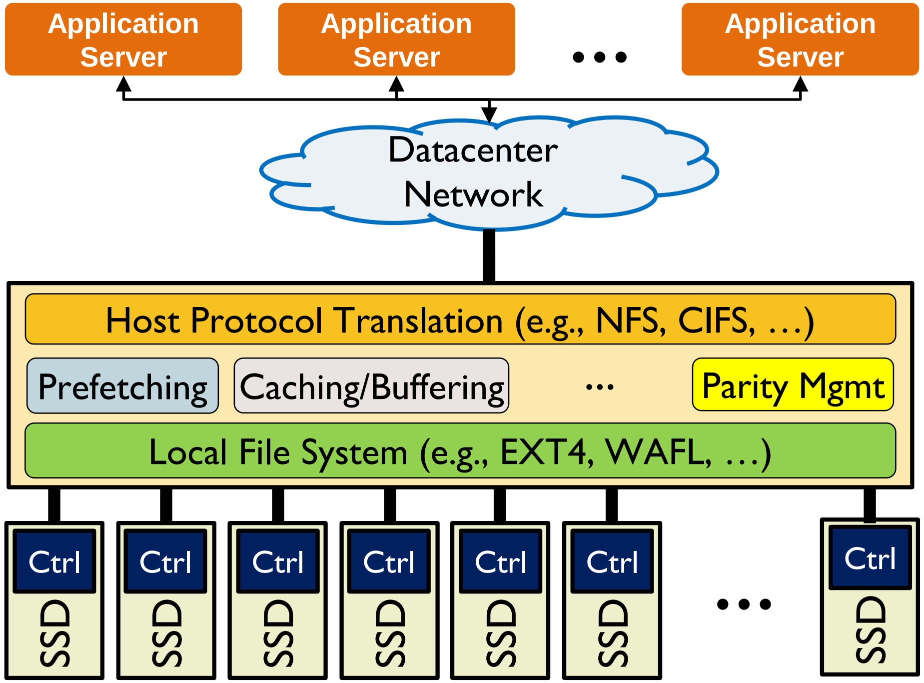
LightStore: Basic Idea

- **Get rid of a space-consuming, expensive, power-hungry host server**
- **Put and run everything in SSDs**
- **Attach SSDs to a datacenter network**
- **Let application servers directly talk to SSDs**



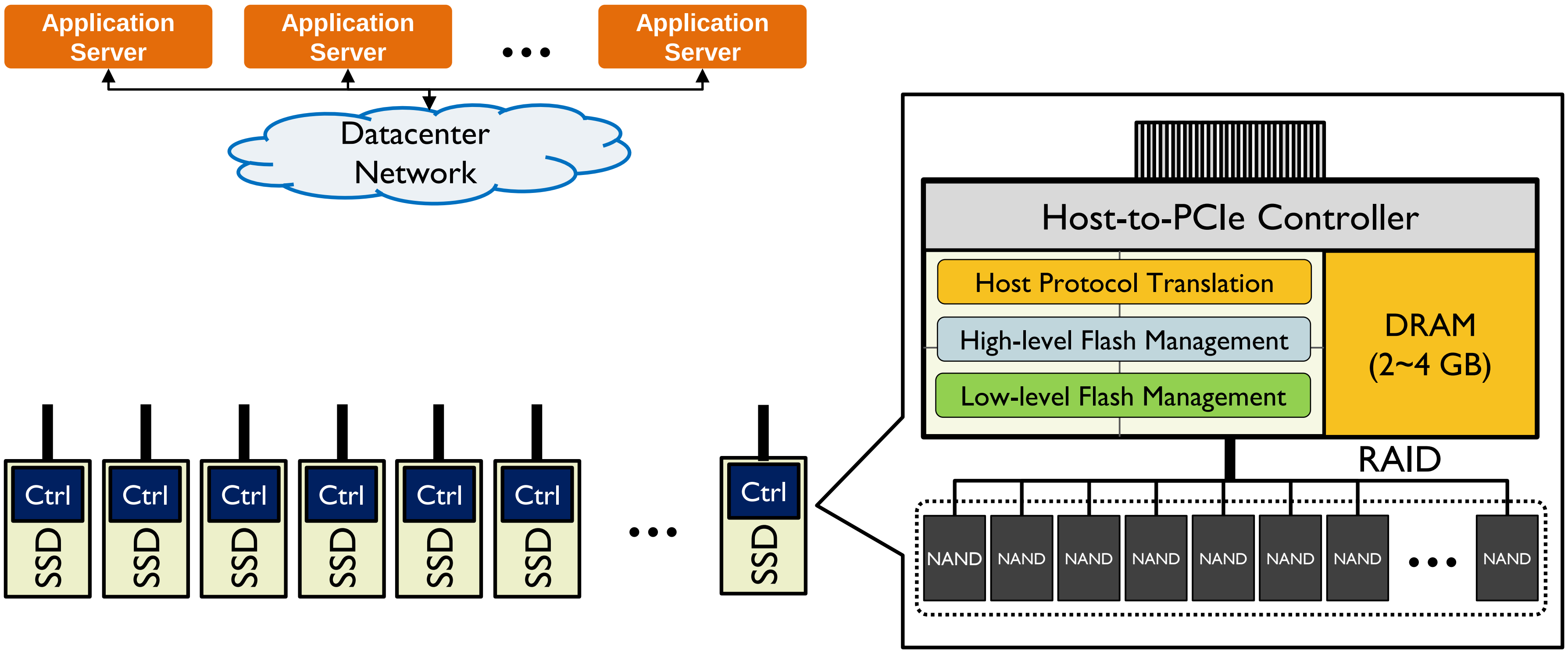
LightStore: Basic Idea

- **Get rid of a space-consuming, expensive, power-hungry host server**
- **Put and run everything in SSDs**
- **Attach SSDs to a datacenter network**
- **Let application servers directly talk to SSDs**



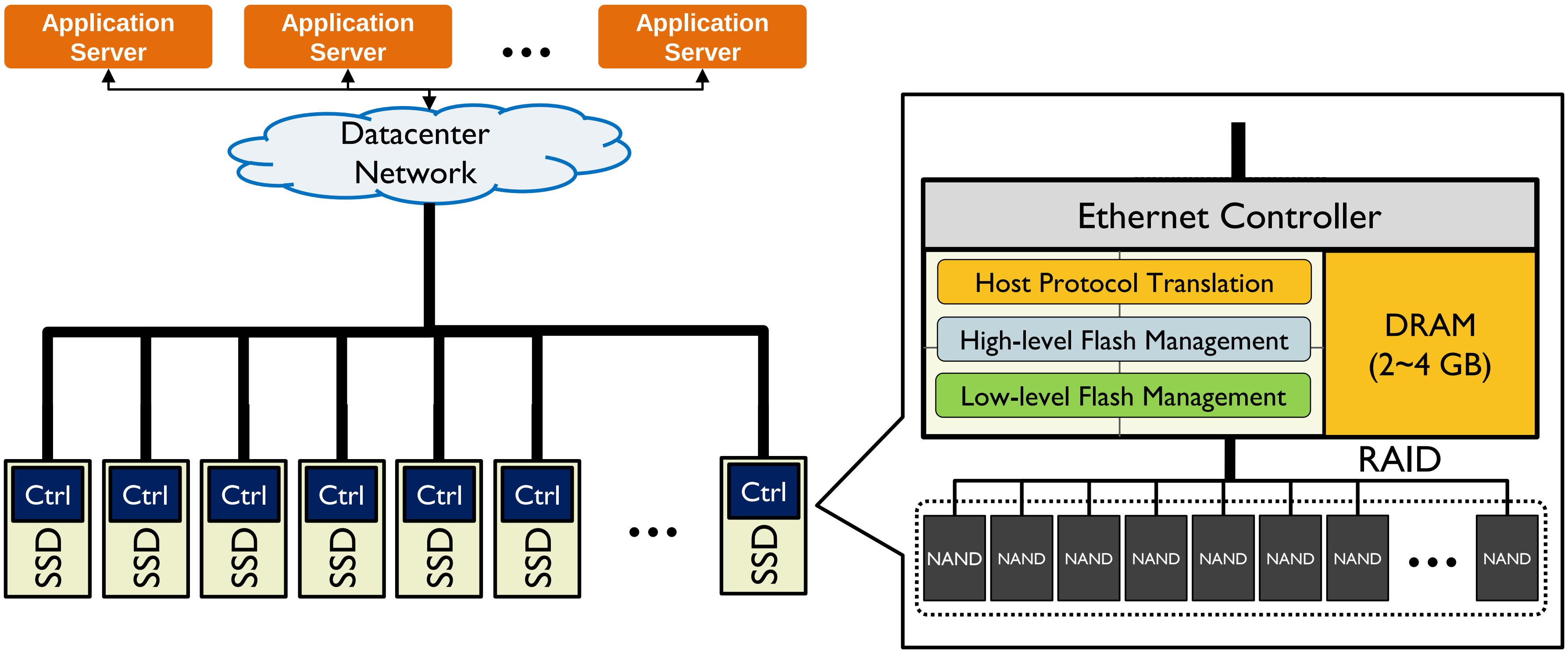
LightStore: Basic Idea

- **Get rid of a space-consuming, expensive, power-hungry host server**
- **Put and run everything in SSDs**
- **Attach SSDs to a datacenter network**
- **Let application servers directly talk to SSDs**



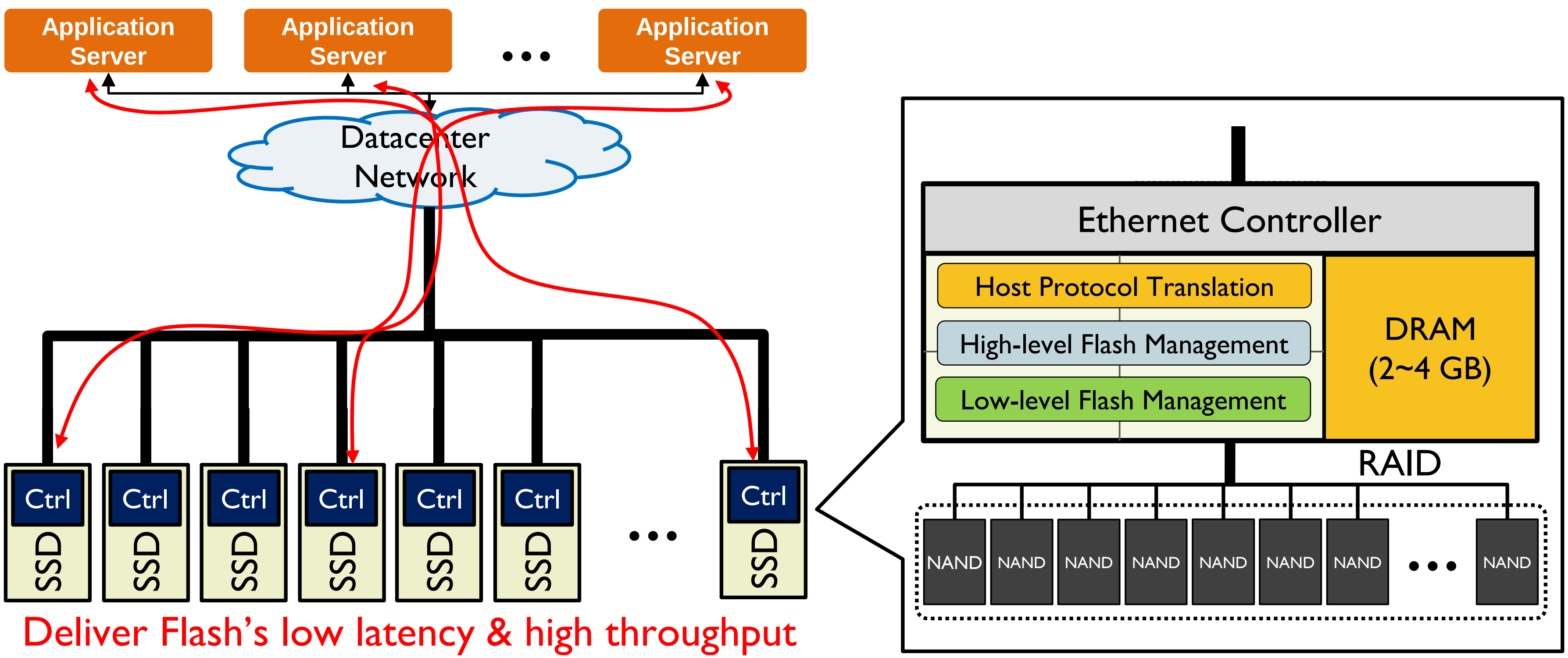
LightStore: Basic Idea

- **Get rid of a space-consuming, expensive, power-hungry host server**
- **Put and run everything in SSDs**
- **Attach SSDs to a datacenter network**
- **Let application servers directly talk to SSDs**



LightStore: Basic Idea

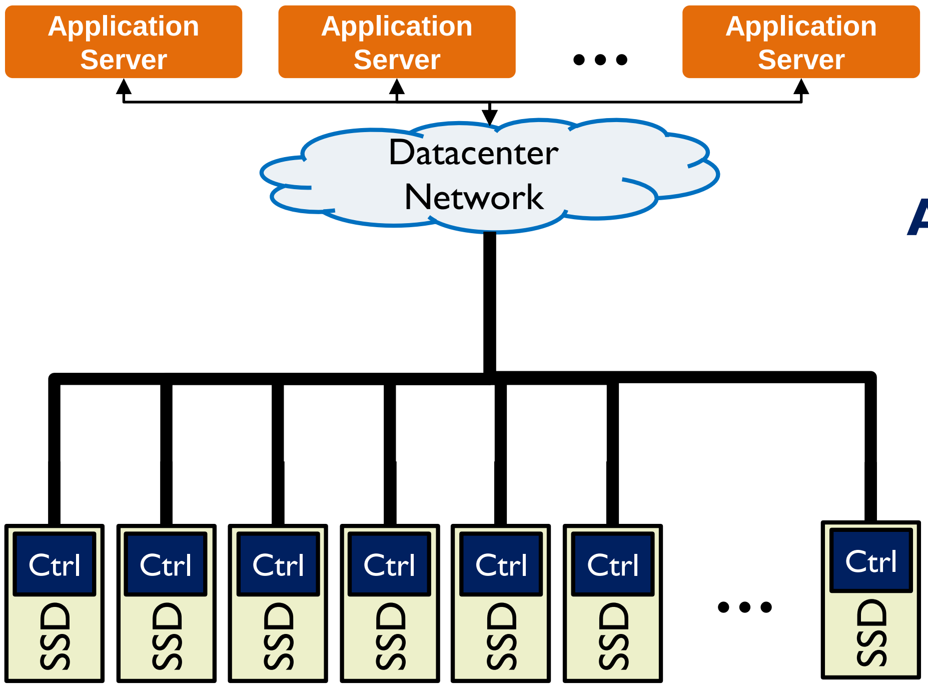
- **Get rid of a space-consuming, expensive, power-hungry host server**
- **Put and run everything in SSDs**
- **Attach SSDs to a datacenter network**
- **Let application servers directly talk to SSDs**



Deliver Flash's low latency & high throughput to network ports!

LightStore: Basic Idea

- **Get rid of a space-consuming, expensive, power-hungry host server**
- **Put and run everything in SSDs**
- **Attach SSDs to a datacenter network**
- **Let application servers directly talk to SSDs**



An x86 storage server with *N* SSDs is replaced with *N* SSDs

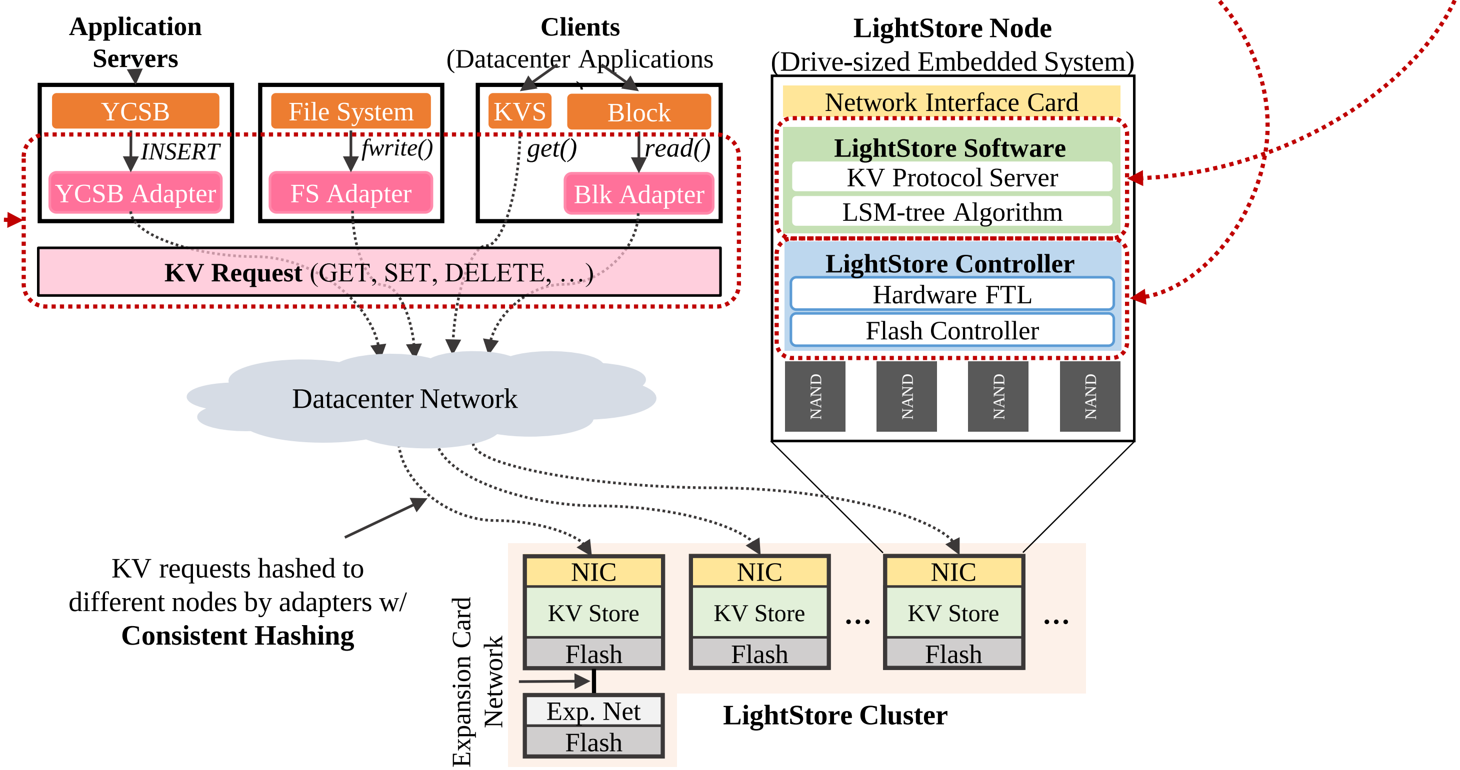
Low Power (e.g., 100 W / 10 SSDs)
Cheap (e.g., Zero server cost)
Small Volume (e.g., Less than 1U)
Low TCO (e.g., Less Cooling)
Scalability (No network bottleneck)

Key Technical Challenges

- **Can we run complicated server software on wimpy ARM cores?**
- **How can we provide the same interface with application servers?**
- **How can we manage unreliable NAND without more ARM cores?**

Overall Architecture of LightStore

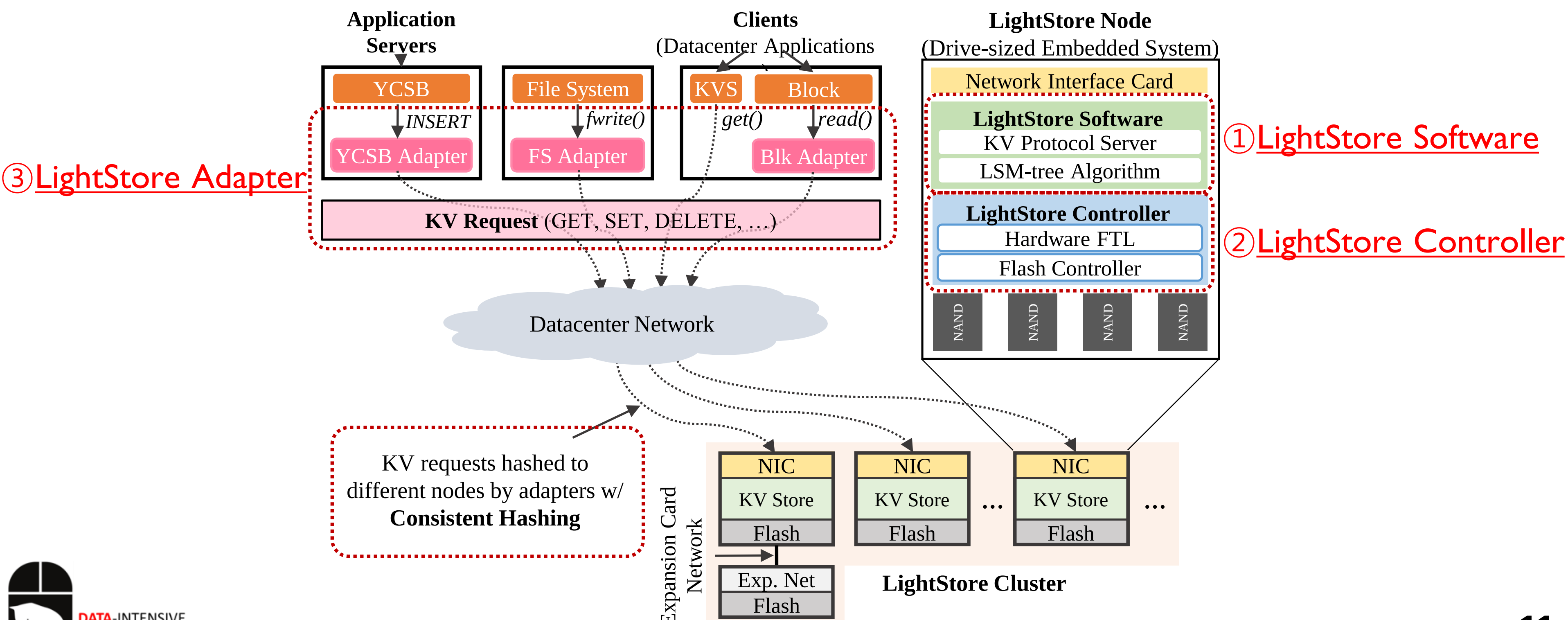
- **Can we run complicated server software on wimpy ARM cores?**
  - ➡ Run a simple KV store (LSM-tree) which exposes a flexible KV interface
- **How can we provide the same interface with application servers?**
  - ➡ Run adaptors on application servers that translate XX-to-KV
- **How can we manage unreliable NAND without more ARM cores?**
  - ➡ Implement FTL in hardware since LSM-tree is append-only



DATA-INTENSIVE COMPUTING SYSTEMS LAB

Overall Architecture of LightStore

- **Can we run complicated server software on wimpy ARM cores?**
  - ➡ Run a simple KV store (LSM-tree) which exposes a flexible KV interface
- **How can we provide the same interface with application servers?**
  - ➡ Run adaptors on application servers that translate XX-to-KV
- **How can we manage unreliable NAND without more ARM cores?**
  - ➡ Implement FTL in hardware since LSM-tree is append-only



DATA-INTENSIVE COMPUTING SYSTEMS LAB

Index

- Motivation
- Basic Idea
- LightStore Software
- LightStore Controller
- LightStore Adapters
- Experimental Results
- Conclusion

Which KVS on LightStore?

- **Hash-based KVS**
  - Simple implementation
  - Unordered keys
    - limited RANGE & SCAN
    - Random==Sequential access
  - Unbounded tail-latency
  - KV-SSDs (mounted on host)
    - Samsung KV-SSD
    - KAML [Jin et. al., HPCA 2017]
    - BlueCache [Shuotao et. al., VLDB 2016]

- **LSM-tree-based KVS**
  - Multi-level search tree
  - Sorted keys
    - RANGE & SCAN
    - Fast sequential access

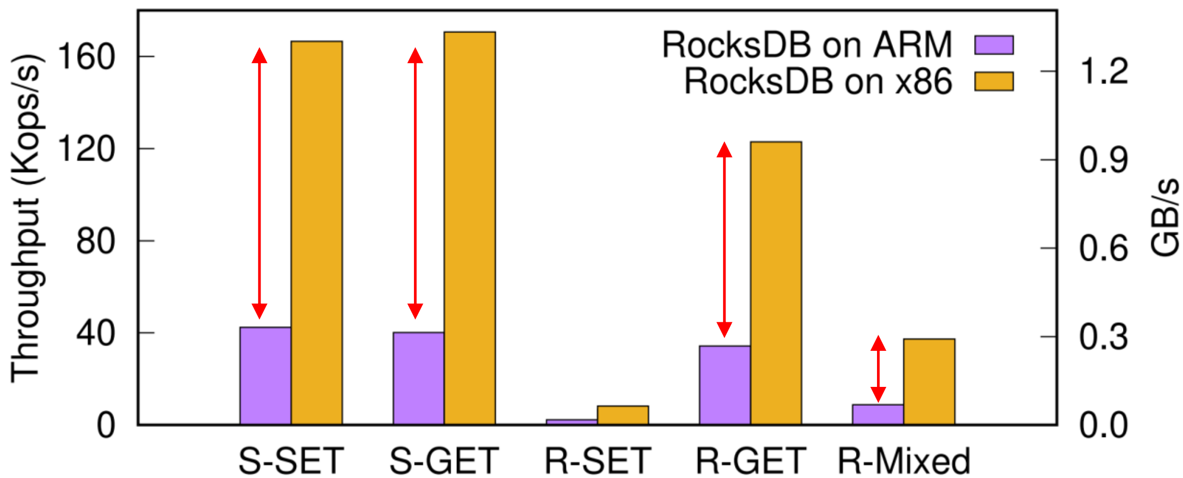
    → Adapter-friendly
  - Bounded tail-latency
  - Append-only batched writes

    → Flash-friendly

Our Choice!

LightStore Software

- **LightStore Software is implemented using the LSM-tree algorithm**
  - Popular algorithm for implementing key-value store (KVS)
  - Suitable for NAND flash since it is append-only
- **How about using existing popular KV software (e.g., RocksDB)?**
  - It is quite heavy to run on ARM cores
- **RocksDB on 4-core ARM + Samsung's 960PRO SSSD**
  - Failed to deliver raw flash throughput to a network port

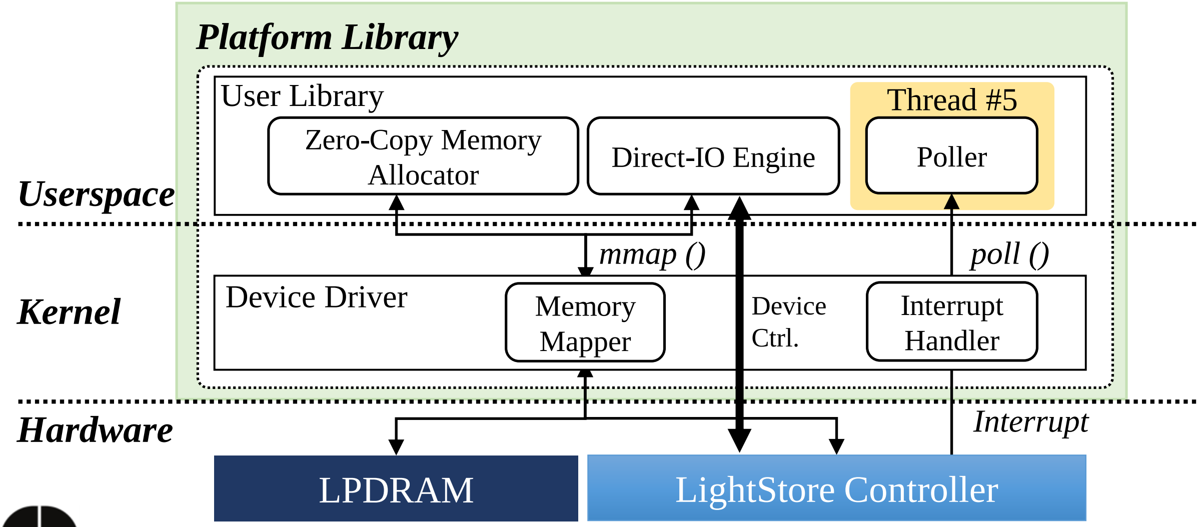


Bottleneck Analysis

- **Three main bottlenecks in running RocksDB on ARM**
  1. Excessive Memory-copy Overhead:
     - memcpy() calls account for up to 30% of the total CPU cycles
     - Partially due to compaction
  2. High Context Switch Overhead:
     - Spawns more than 20 threads for simultaneously processing user requests, flush and compaction
     - 4 cores are available in SSD controller
  3. Deep and Sophisticated Software Stack:
     - Runs atop kernel layers, such as a page cache, a file system and a block I/O layer

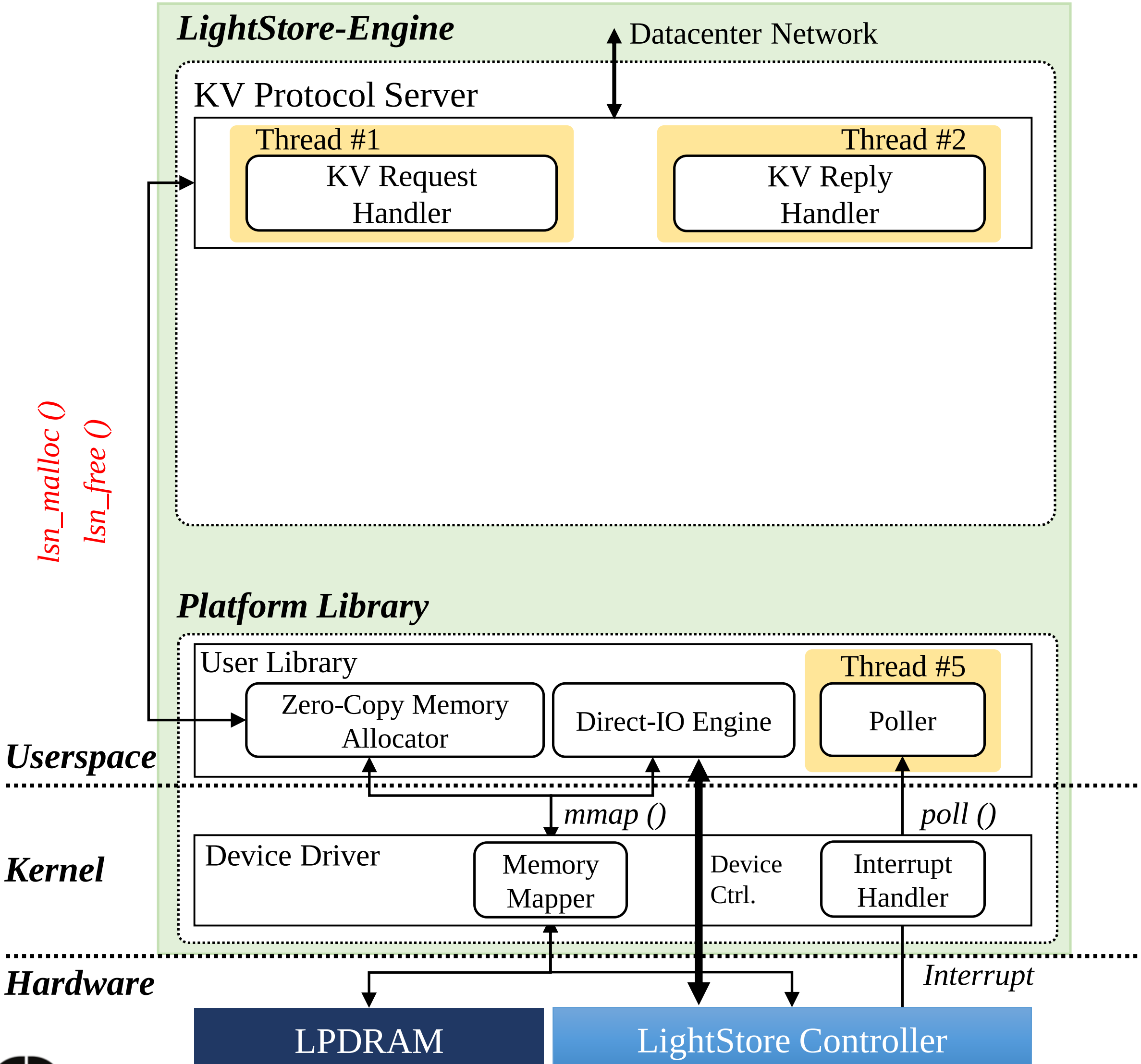
- **Solutions?**
  1. Implement KVS from scratch so that it efficiently runs on ARM
  2. Rebuild a lightweight storage stack

Platform Library



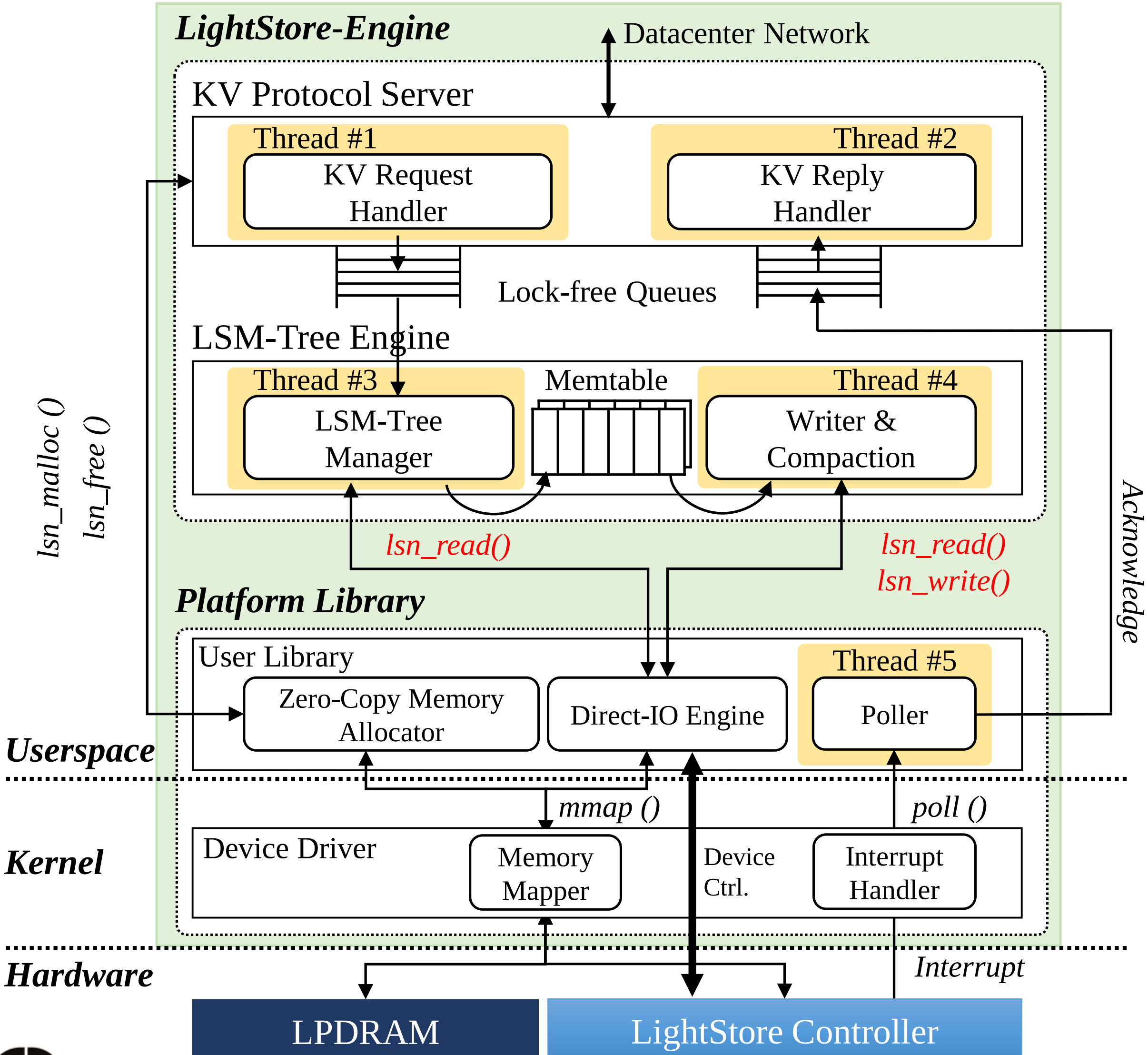
- **Platform Library**
  - Not rely on the kernel too much
  - *Zero-copy memory allocator*: Use mmap() to directly transfer data between DRAM and devices
  - *Direct-IO engine*: Use memory-mapped registers and poll to control HW

KV Protocol Server



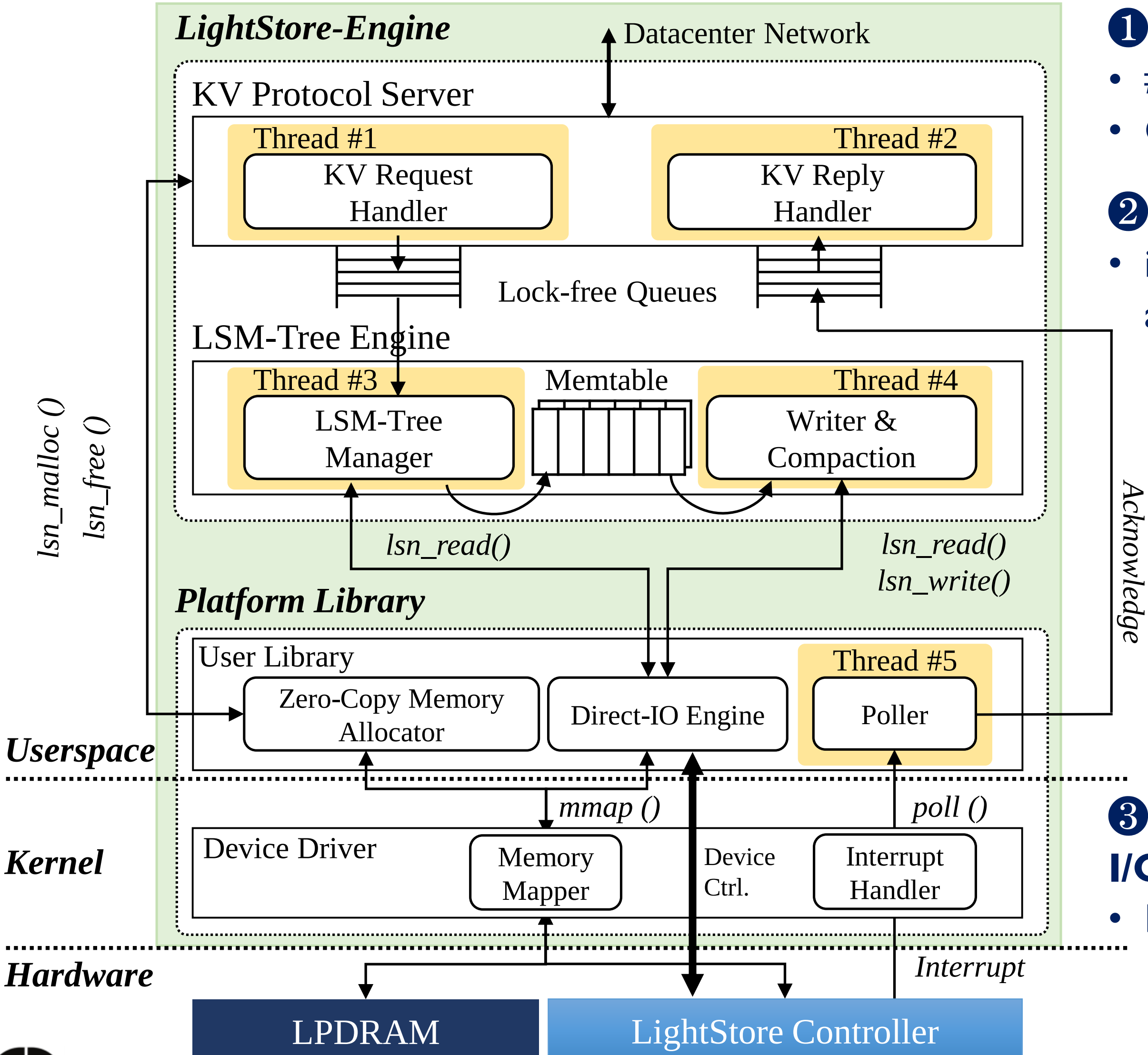
- **KV Protocol Server**
  - A simple socket server to deal with KV requests
  - Use the zero-copy allocator to avoid data copy between NIC and DRAM

LSM-Tree Engine



- **LSM-Tree Engine**
  - Implementation of the LSM-tree algorithm optimized for ARM
    1. Key-value decoupling
    2. Key-table caching
    3. …
  - Use the direct-IO engine to control the LightStore controller
  - Just forward pointers of allocated memory chunks to the LightStore controller

Summary



❶ Less context switch overheads

- # of threads is limited to five
- Glued via lock-free queues

❷ No mem copy across all layers,

- including KV server, LSM-tree engine, and platform library

❸ Less intervention by the deep I/O stack

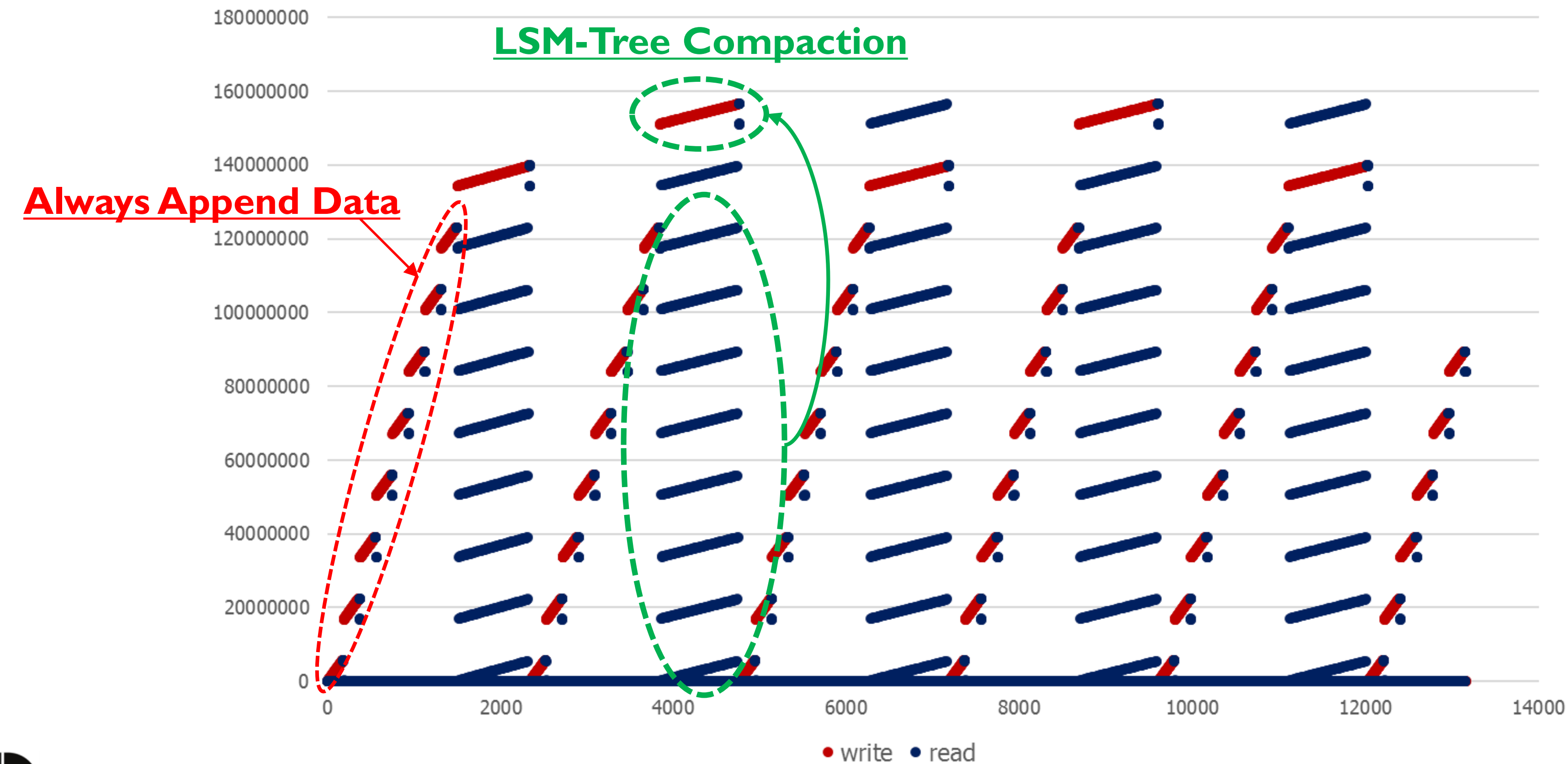
- No block layer, no file system, …

Index

- Motivation
- Basic Idea
- LightStore Software
- LightStore Controller
- LightStore Adapters
- Experimental Results
- Conclusion

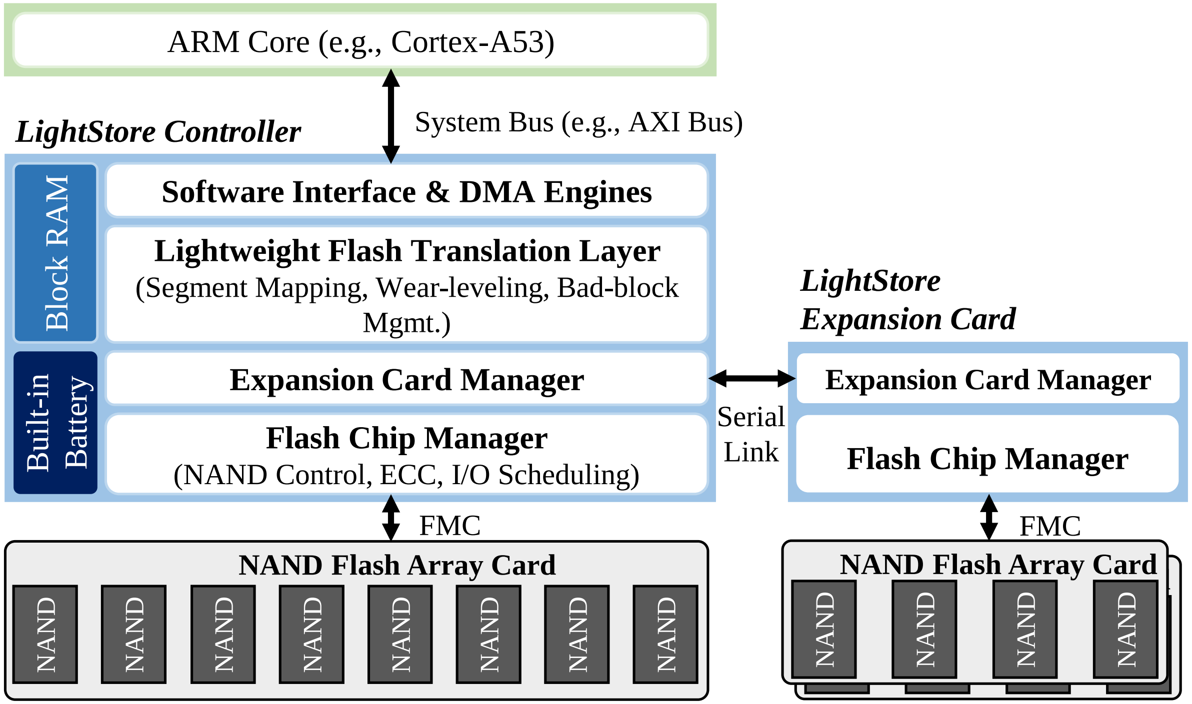
LightStore Controller

- **The LSM-tree writes all the data sequentially all the time**
- **Example:**
  - I/O access patterns of RocksDB based on LSM-tree



LightStore Controller (Cont.)

- **The append-only behaviors of the LSM-tree simplify the FTL design**
  - No fine-grained mapping (e.g., page-level mapping)
  - No garbage collection (i.e., LSM-tree's compaction replaces it)
- **The FTL is *completely* implemented in HW**
  - No ARM CPU is necessary; enables us to use more ARM cores to run software
  - Faster than SW FTL; 700 ns for address translation



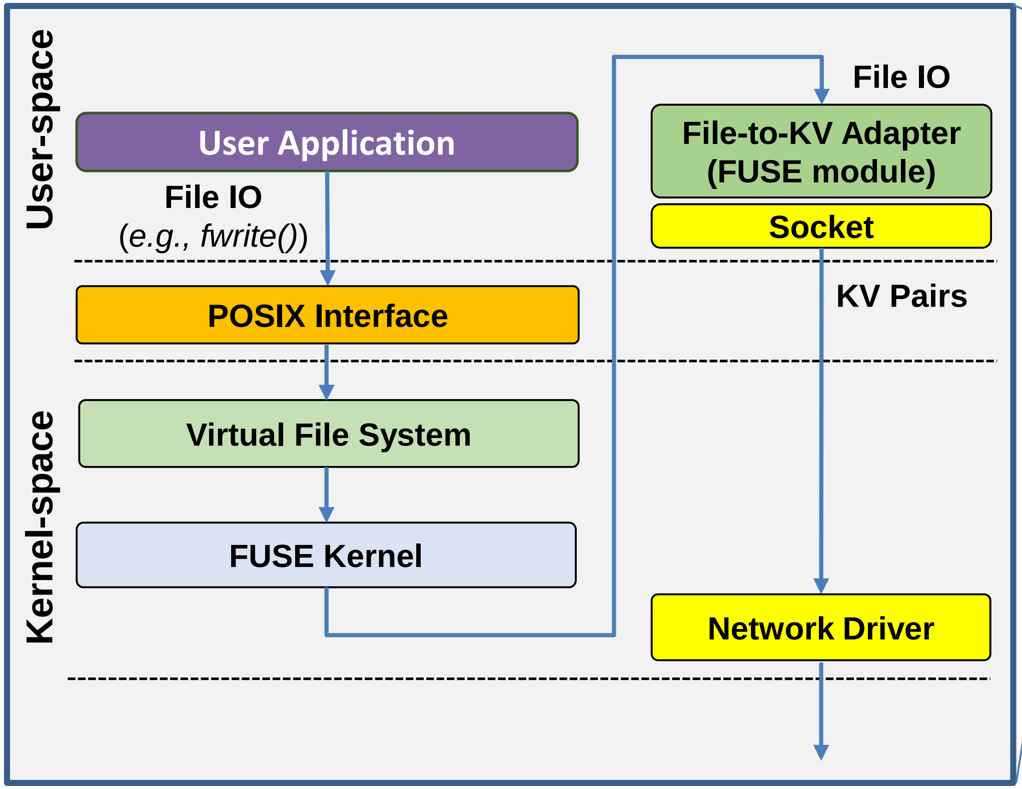
Index

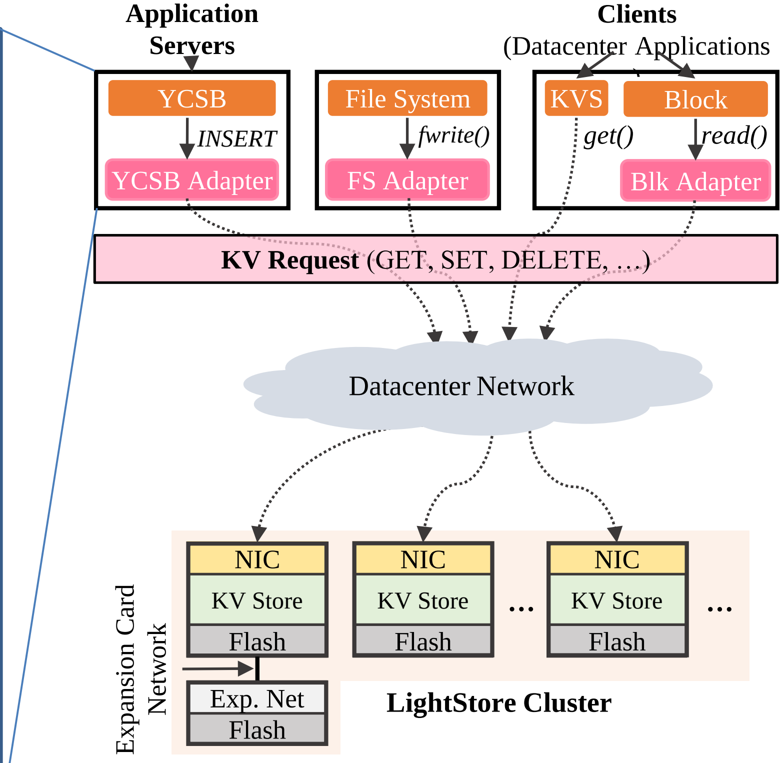
- Motivation
- Basic Idea
- LightStore Software
- LightStore Controller
- LightStore Adapters
- Experimental Results
- Conclusion

LightStore Adapter

- **LightStore adapter is responsible for translating traditional I/O commands into KV pairs**
- **Run on applications server side as FUSE, BUSE, and library**

Example: File-to-KV Adapter





Protocol Translation

- **The flexibility of KV interface makes it possible for us to support various traditional protocols**

- **Four protocols are supported**

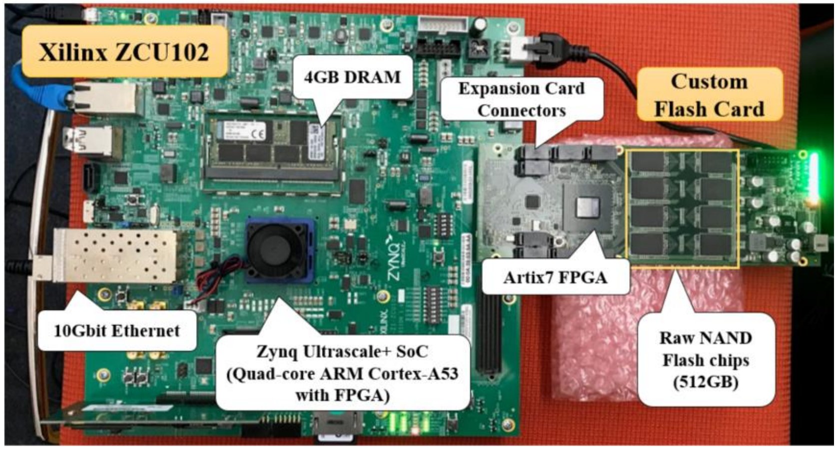
1. Native KV Interface: Get/Put …
   - LightStore supports a KV interface natively
2. YCSB Interface: Read/Insert/Scan …
   - Each YCSB command directly corresponds to a specific KV operation, except for multiple fields
   - Multiple fields can be supported with MGET/MSET
3. Block Interface: Read/Write/Trim
   - A key corresponds to LBA; A value corresponds to 4KB fixed-size data
4. File Interface: fread()/fwrite() …
   - A file can be handled as the form of a key-value object
   - Currently, run a file system atop the block interface

Index

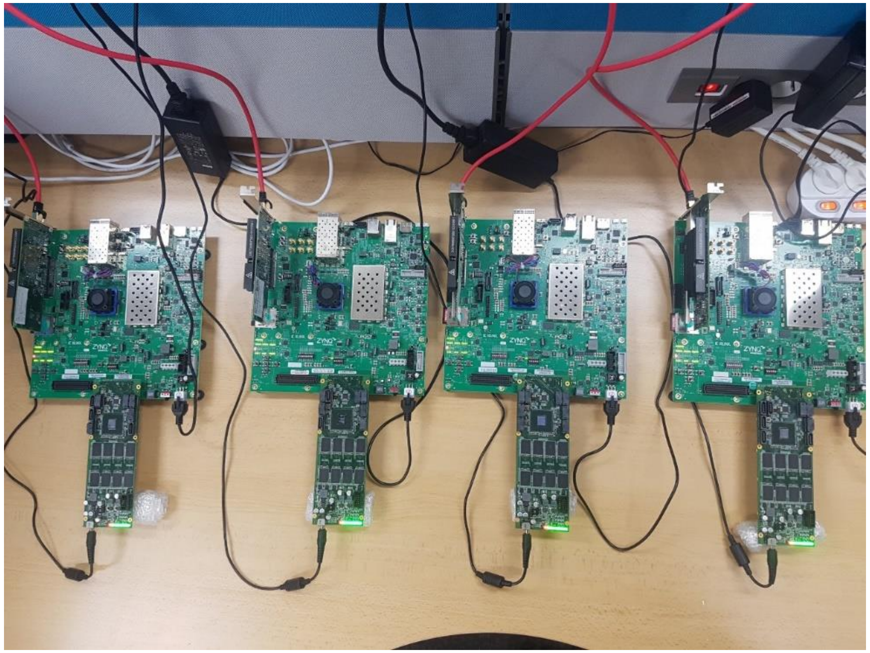
- Motivation
- Basic Idea
- LightStore Software
- LightStore Controller
- LightStore Adapters
- Experimental Results
- Conclusion

LightStore Prototype

- **Each LightStore Prototype node is implemented using a Xilinx ZCU102 evaluation board (w/ Cortex A53 CPU) and a custom flash card**



Four-LightStore cluster



Experimental Setup

	x86-based storage system	LightStore
CPU	Xeon E5-2640 (20 cores @ 2.4 GHz)	ARM Cortex-A53 (4 cores @ 1.2 GHz)
DRAM	32 GB	4 GB
SSD or flash Throughput Latency	Samsung 960 PRO 512 GB SSD 3.21 GB/s / 1.38 GB/s 80 us / 120 us Firmware (FTL, buffers ...)	Custom 512 GB NAND Flash 1.2 GB/s / 430 MB/s 120 us / 480 us Raw Flash
KVS	RocksDB v5.8	Our LSM-tree engine
Client Ifc	ARDB	Our KV protocol server
Network	10 Gbit Ethernet (* up to 1.20 GB/s)	10 Gbit Ethernet (* up to 620 MB/s)
OS	Ubuntu 16.04 (Linux 4.9.0)	

- Clients and storage nodes are connected to the same 10GbE switch

KVS Workloads

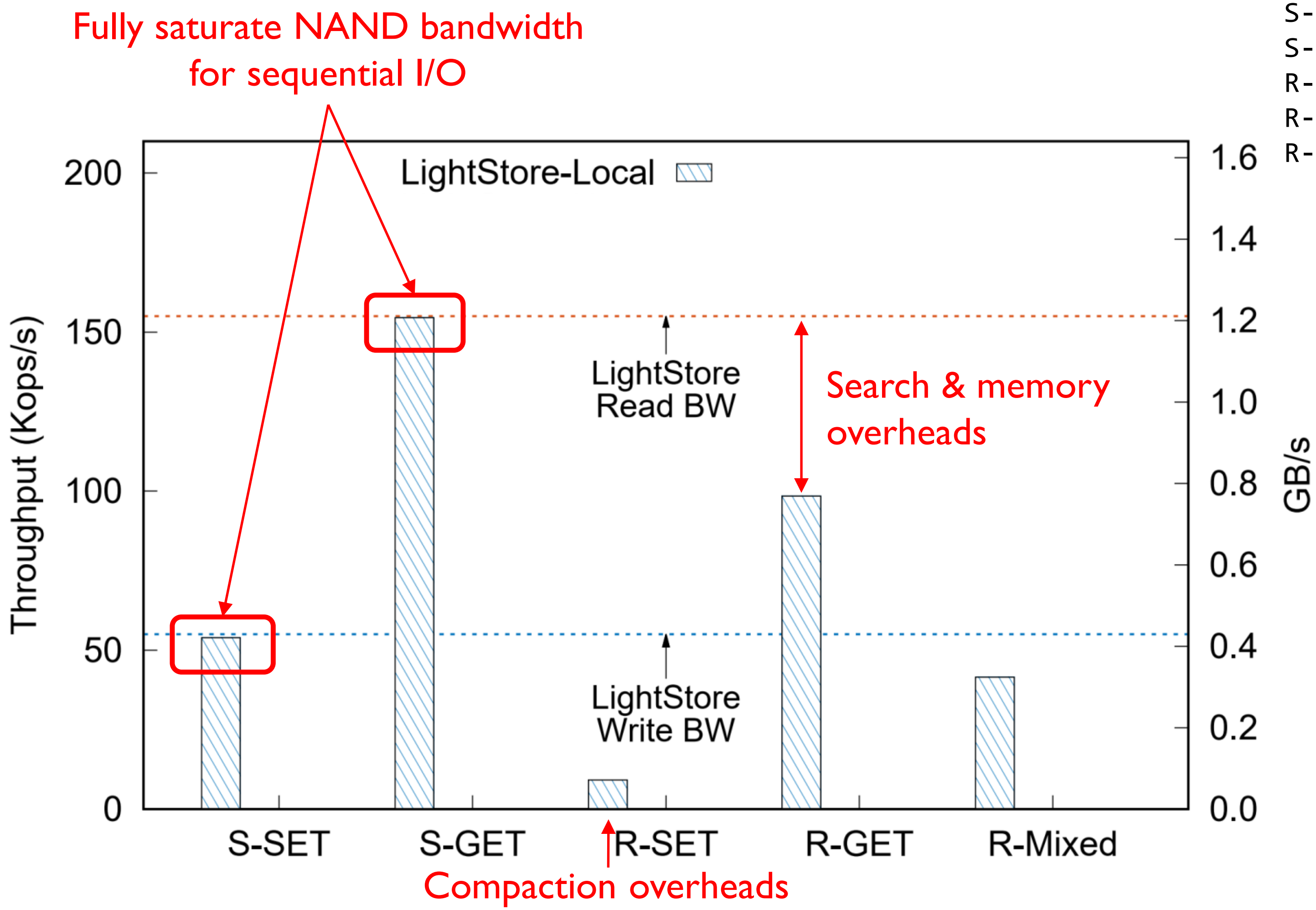
- **5 synthetic workloads to evaluate KVS performance**

Synthetic Workloads	
S-SET	Sequential Write
S-GET	Sequential Read
R-SET	Random Write
R-GET	Random Read
R-Mixed	Random R:W=9:1

- The value size of 8-KB used to match the flash page size
  - The latest version has been improved to support various key/value sizes

Local Performance

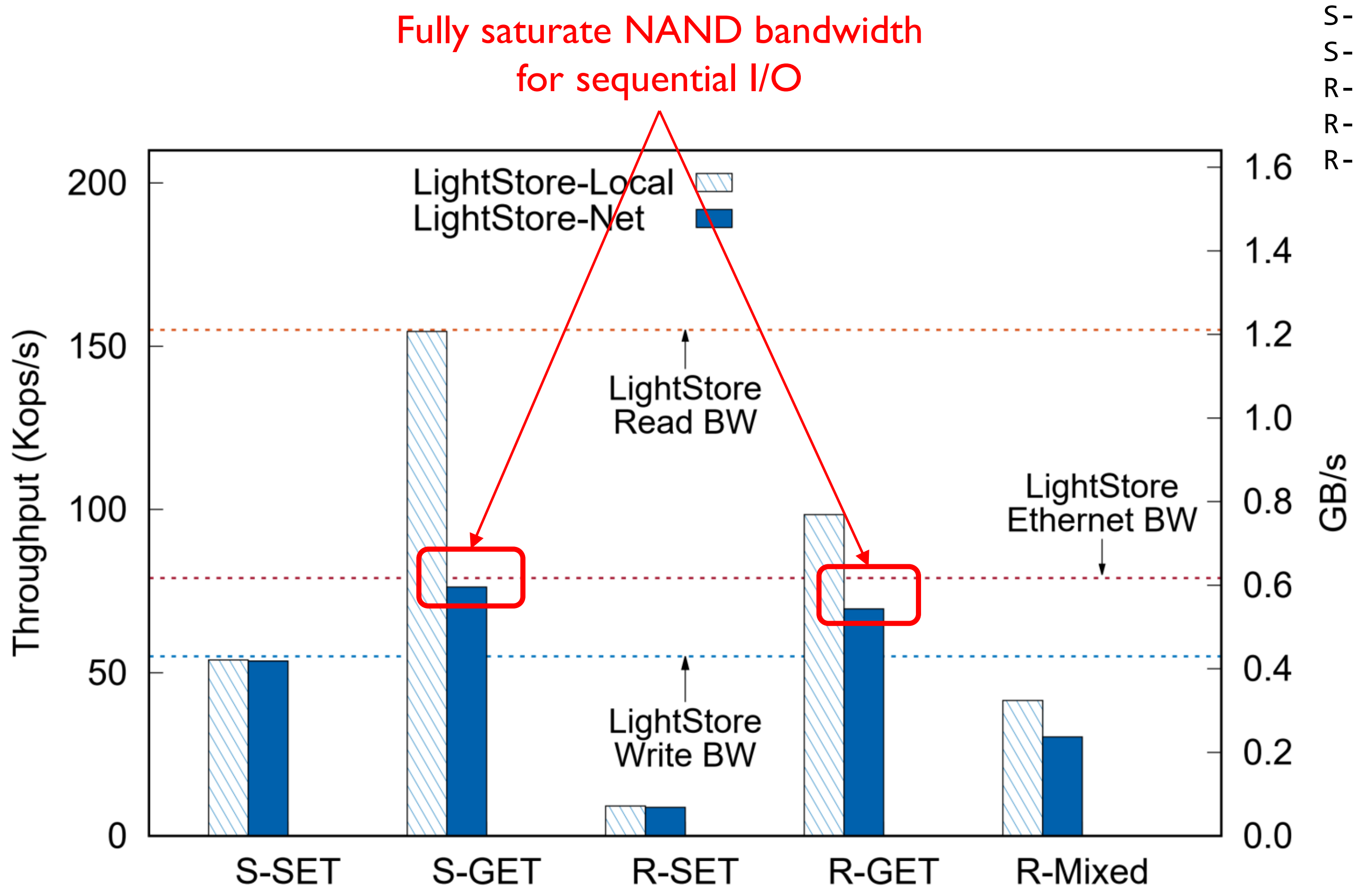
S-SET: Sequential Set
S-GET: Sequential Get
R-SET: Random Set
R-GET: Random Get
R-Mixed: Random Mixed



- **Except for write workloads, LightStore fully saturates flash bandwidth**

Network Performance

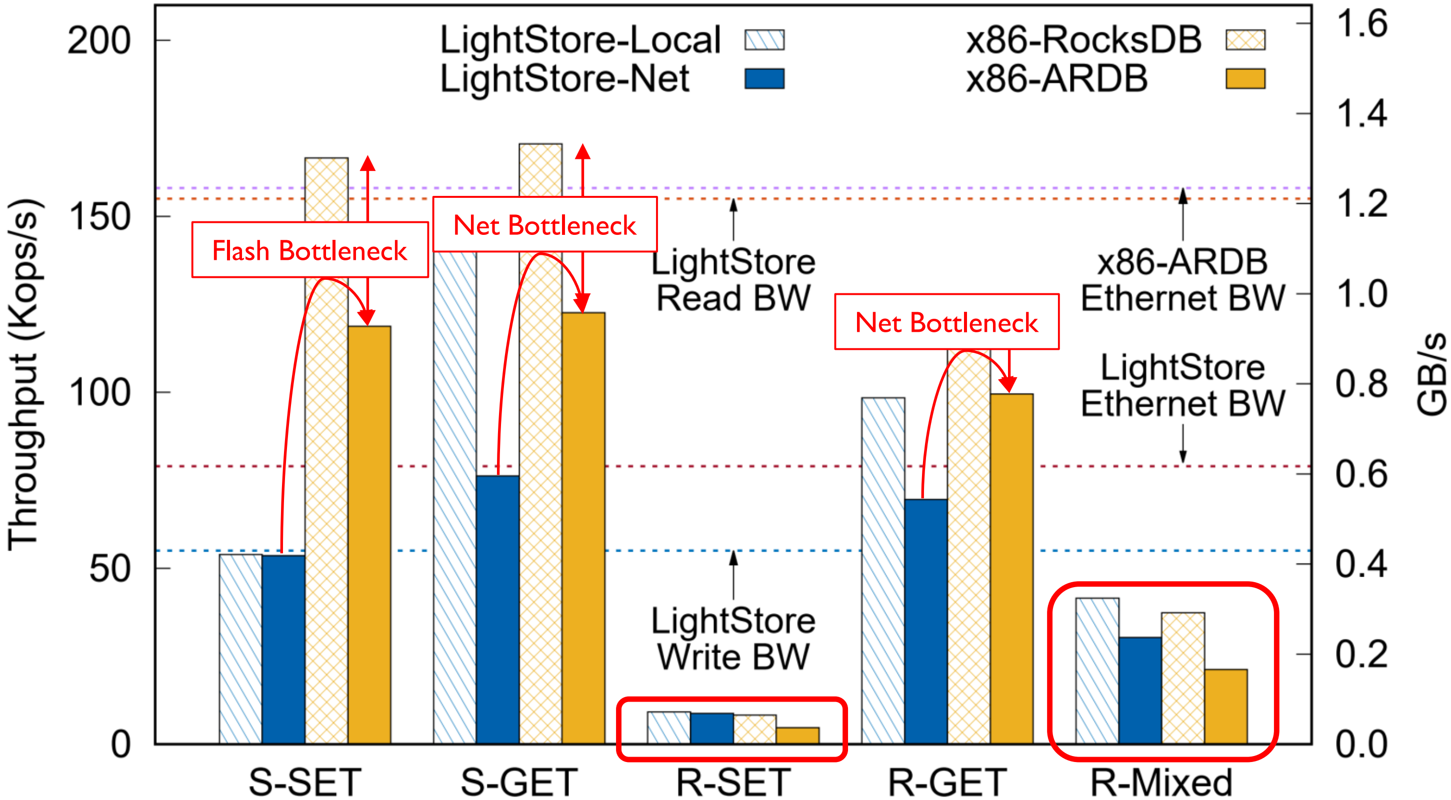
S-SET: Sequential Set
S-GET: Sequential Get
R-SET: Random Set
R-GET: Random Get
R-Mixed: Random Mixed



- **Except for write workloads, LightStore fully saturates Net bandwidth**

Comparison with x86

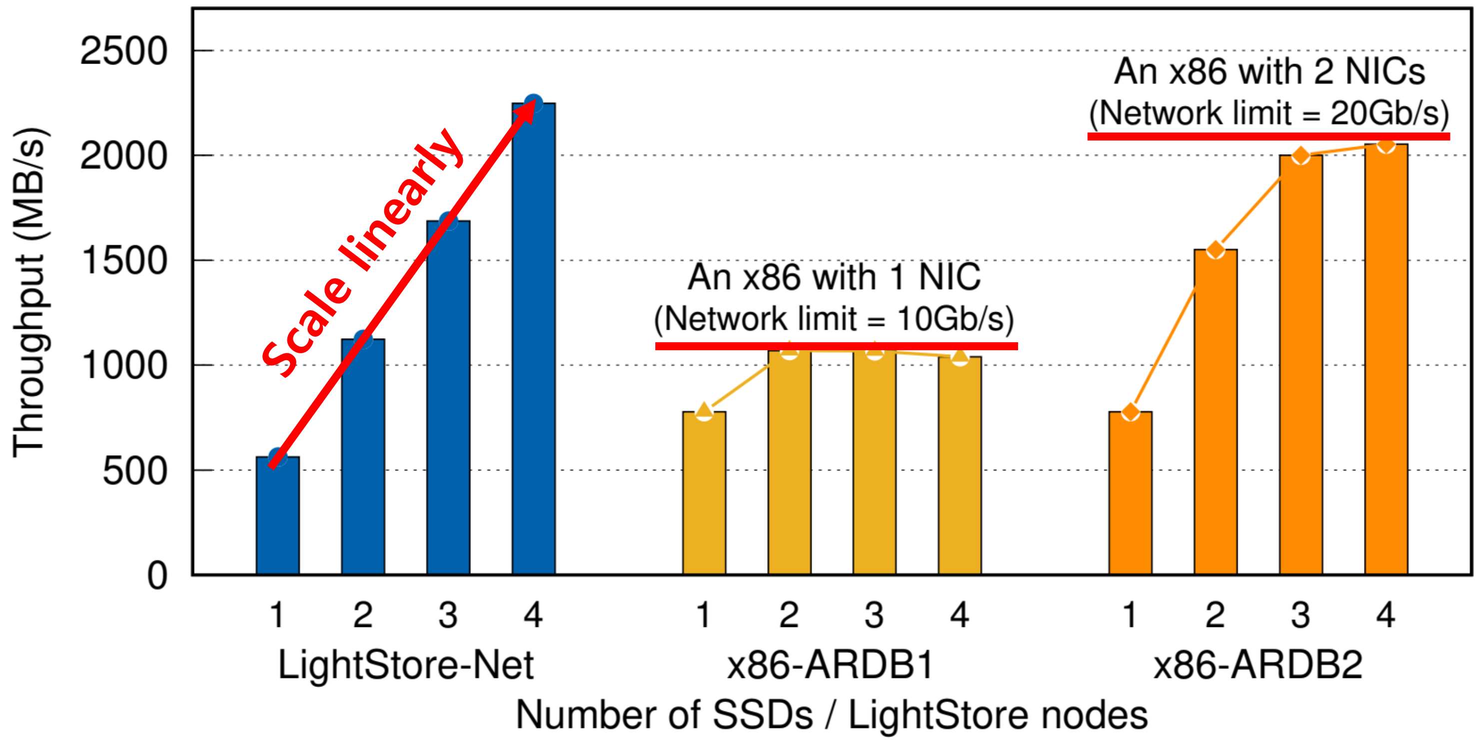
- **x86-RocksDB performs better thanks to high speed of Samsung 960PRO**
- **LightStore outperforms x86 under random writes (e.g., R-SET and R-Mixed)**
- **x86-ARDB suffers from non-trivial software stack overheads**



DATA-INTENSIVE COMPUTING SYSTEMS LAB

DGIST 대구경북과학기술원 Daegu Gyeongbuk Institute of Science & Technology

Scalability



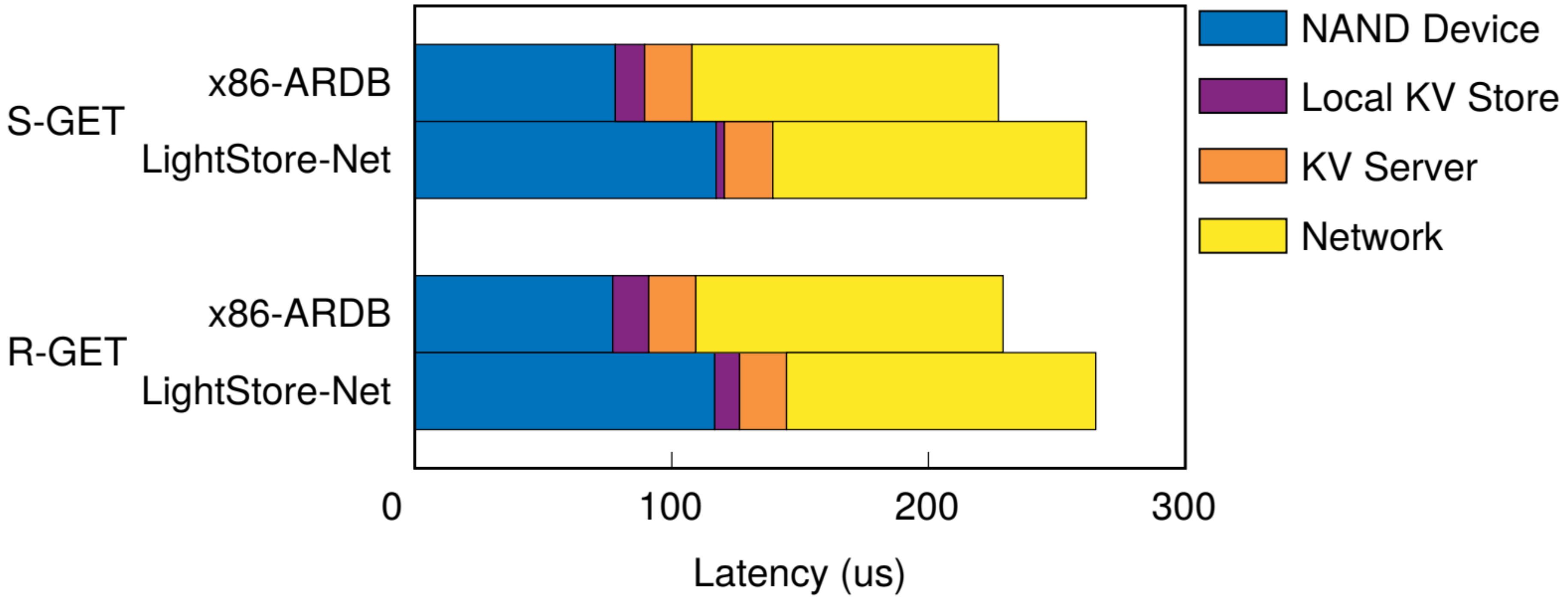
- **LightStore scales linearly according to the number of SSDs added to a cluster**

KVS IOPS-per-Watt

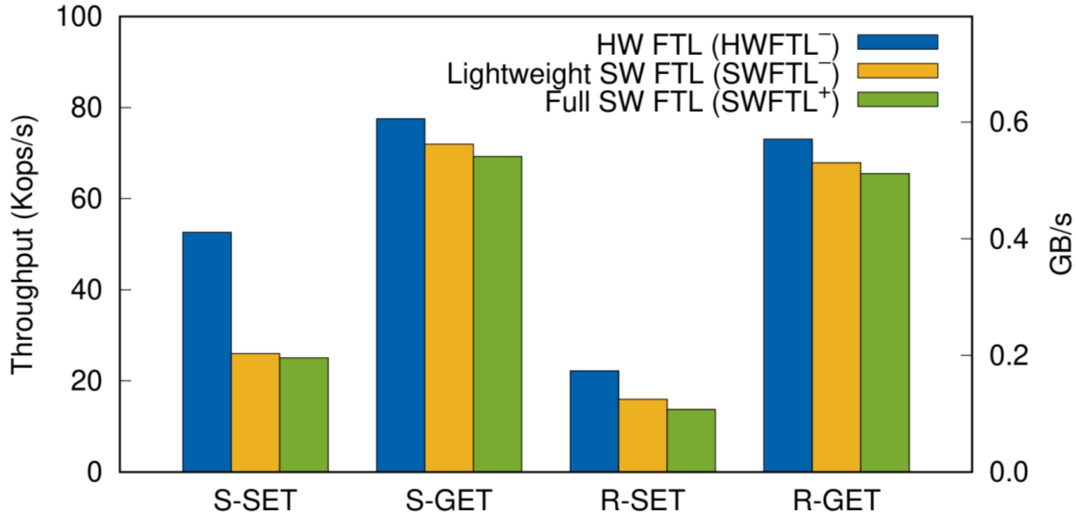
- **Assume that x86-ARDB scales with up to 4 SSDs**
  - 4 times the performance seen previously
- **Peak power**
  - x86-ARDB – 400W, LightStore-Prototype – 25W

IOPS/W	S-SET	S-GET	R-SET	R-GET	R/W mix
LightStore Gain	1.8x	2.5x	7.4x	2.8x	5.7x

Latency Comparison



Impact of HW FTL on Performance



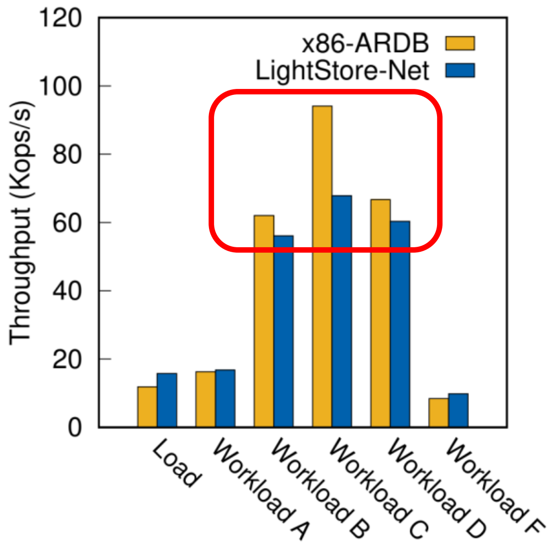
- **HW FTL > Lightweight SW FTL > Full SW FTL**
  - Full SW: page mapping; garbage collection copying overhead
- **Read: 7-10% degradation**
- **Write: 28-50% degradation**
  - Compaction thread very active; More SW FTL tasks

→ Without FPGA (or HW FTL), we would need an extra set of cores
(Trade-off between Cost and Design Efforts)

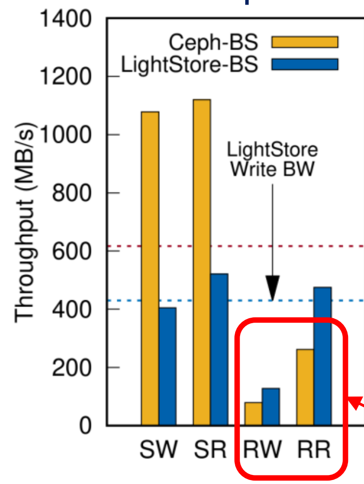
Adapter Performance

- **Network-attached Single Node Performance**

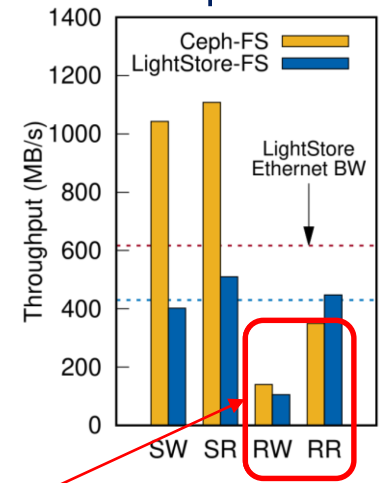
<YCSB performance>



<Block I/O performance>



<File I/O performance>

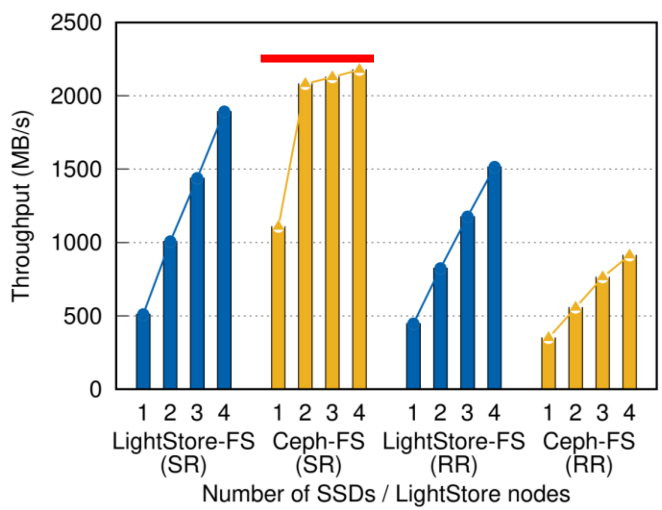


Max. Network

Max. NAND

Ceph is inefficient for handling small data

- **Scalability w/ Multiple Nodes**



Conclusion

- **This work was motivated by two observations in distributed storage**
  1. The existing storage architecture did not scale well
  2. Applications failed to exploit full performance of SSDs over the network

- **LightStore is a lean drive-sized high-speed KV node which plugs directly into a network port**
  1. Lightweight KV storage engine → Deliver full NAND speed to network ports
  2. Hardware FTL → Minimize resource requirements
  3. XX-to-KV adapters → Support various applications w/ no modification

- **A four-node cluster showed a comparable throughput to the AFA with four SSDs and achieved up to 7.4x better ops/J**

Thank you!

https://datalab.dgist.ac.kr