11. In-storage Processing

Special Topics in Computer Systems:
Modern Storage Systems
(IC820-01)

Instructor:

Prof. Sungjin Lee (sungjin.lee@dgist.ac.kr)

Outline

- **Why In-storage Processing?**
- Case Studies
  - BlueDBM: Big-data Analytic Platform with In-storage Processing
  - BlueCache: Hardware-accelerated Key-value Store
  - Biscuit: User-programmable In-storage Processing Framework
  - IESSD: Inference-Enabled SSDs

In-storage Computing

near - data processing의 storage

NDP: data source 옆에서 바로 연산해버리자.

- **Extension of near-data processing (NDP) for storage devices**
  - The idea of NDPL: computation is performed right at the data source
    - e.g., ***Processing in Memory (PIM)***: the integration of a processor with RAM

데이터 이동에 비용 커서
스토리지에서 계산하고
위로 올리기.

- **Efficient when the cost of moving data is very high**
  - Example: All-flash array

PIM이 있는데 하는 이유는
스토리지가 빨라져서. 네트워크가 bottleneck.

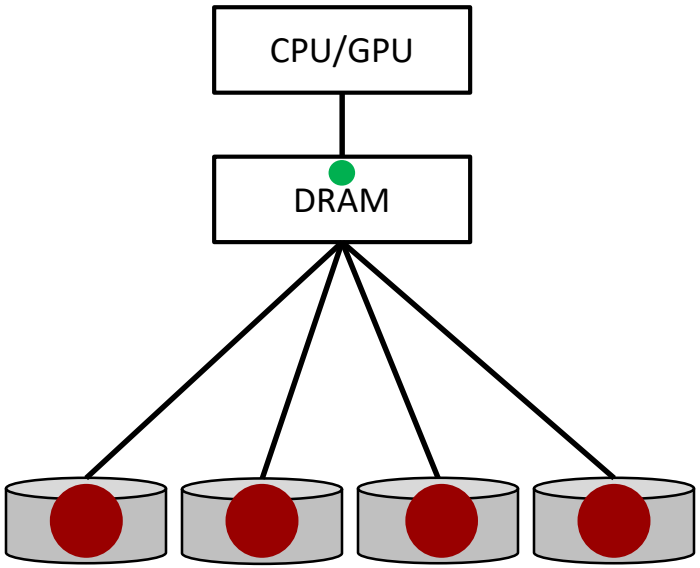
    - PCIe bandwidth is about 64 GB/s, but a single SSD offers 5~10 GB/s
    - Modern storage servers are composed of many SSDs (10~250 SSDs)
    - ***The PCIe bus is bottleneck!***

All - flash array의 경우, SSD가 굉장히 많기에 PCIe bandwidth가 bottleneck이 되어서
SSD 쪽에서 연산을 해서 조금만 위로 보내는 게 더 낫을 수 있다는 것.

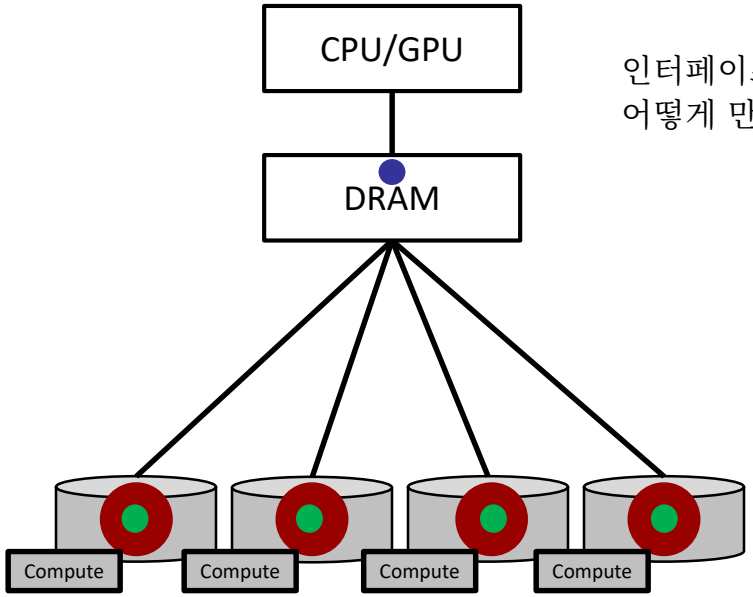
In-storage Computing (Cont.)

- **Transferring large amounts of data to DRAM is time and energy consuming**
  - PCIe bus / network are becoming a bottleneck

- **In-storage acceleration (or near-data process) is one of emerging solutions**
  - Offload computation logics into storage side; return only results to host side
  - Better performance, lower energy consumption, and higher scalability



Traditional System



In-storage computing

인터페이스를
어떻게 만들어야 하는가?

Outline

- Why In-storage Processing?
- Case Studies
  - **BlueDBM: Big-data Analytic Platform with In-storage Processing**
  - BlueCache: Hardware-accelerated Key-value Store
  - Biscuit: User-programmable In-storage Processing Framework
  - IESSD: Inference-Enabled SSDs

BlueDBM Project

- The BlueDBM project started around early 2013 with the aim of accelerating big data analytics on NAND flash clusters
- Various companies, including Quanta, Samsung, Hynix, and Lincoln Lab, supported or are supporting this project
- Now, it becomes an excellent playground for flash research ....

  and is *evolving now*

Big Bata Analytics

- **Analysis of previously unimaginable amount of data can provide deep insight**
  - Google has predicted flu outbreaks a week earlier than the Center for Disease Control (CDC)
  - Analyzing personal genome can determine predisposition to diseases
  - Social network chatter analysis can identify political revolutions before newspapers
  - Scientific datasets can be mined to extract accurate models

A Currently Popular Solution: RAM Cloud

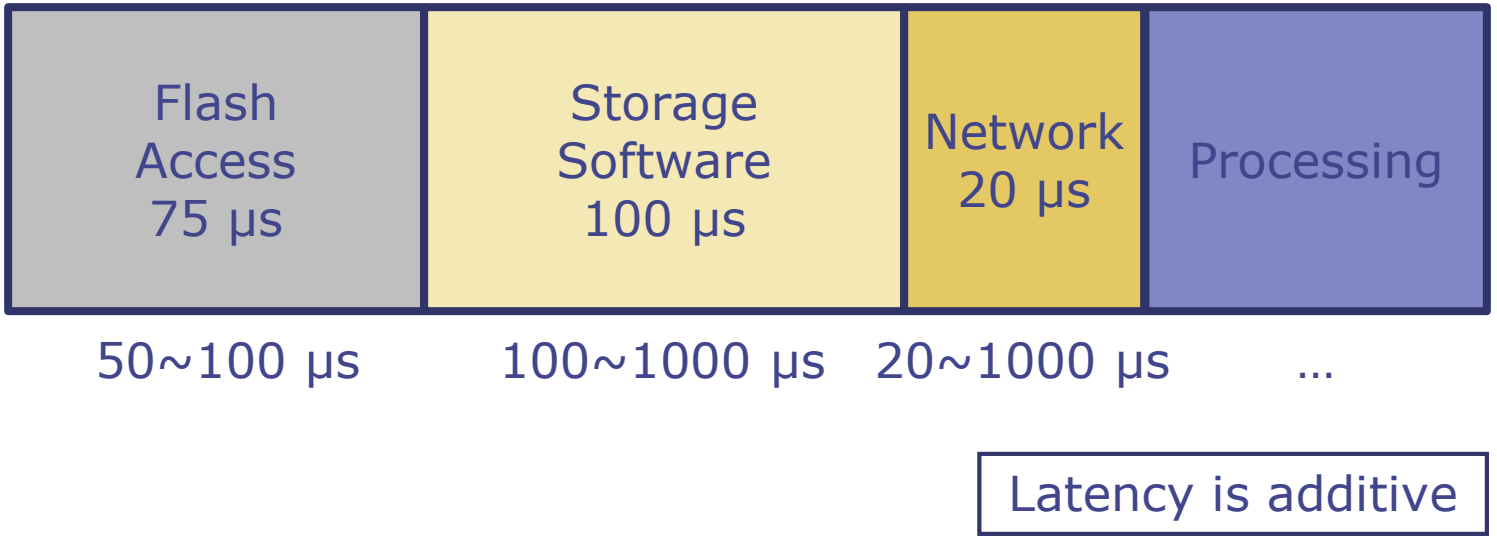
- **Cluster of machines with large DRAM capacity and fast interconnect**
  - \+ Fastest as long as data fits in DRAM
  - \- Power hungry and expensive
  - \- Performance drops when data doesn't fit in DRAM

What if enough DRAM isn't affordable?

- **Flash-based solutions may be a better alternative**
  - \+ Faster than Disk, cheaper than DRAM
  - \+ Lower power consumption than both
  - \- Legacy storage access interface is burdening
  - \- Slower than DRAM

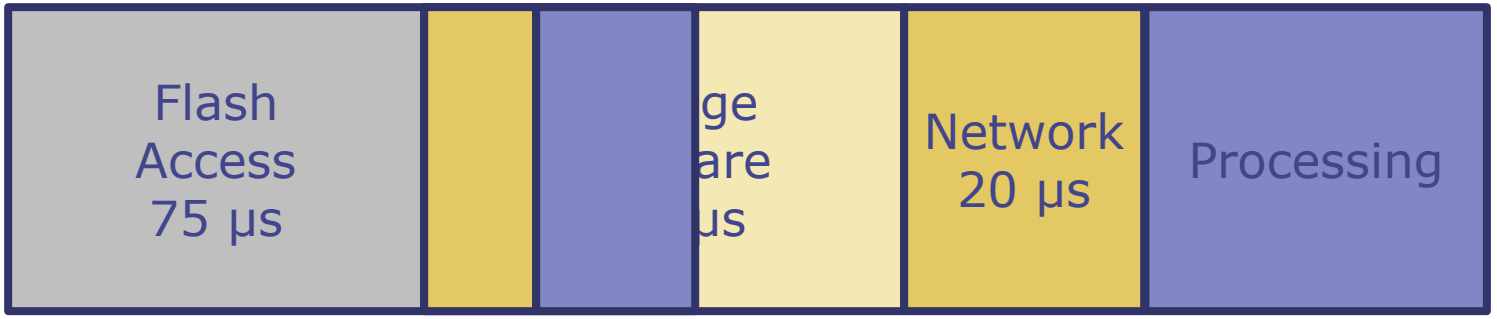
Latency Profile of Distributed Flash-based Analytics

- **Distributed processing involves many system components**
  - Flash device access
  - Storage software (OS, FTL, …)
  - Network interface (10gE, Infiniband, …)
  - Actual processing

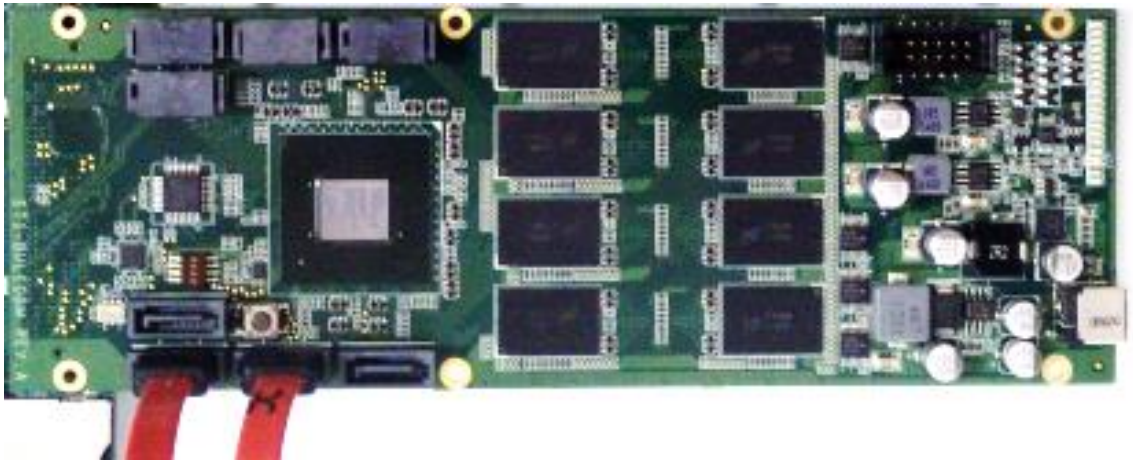


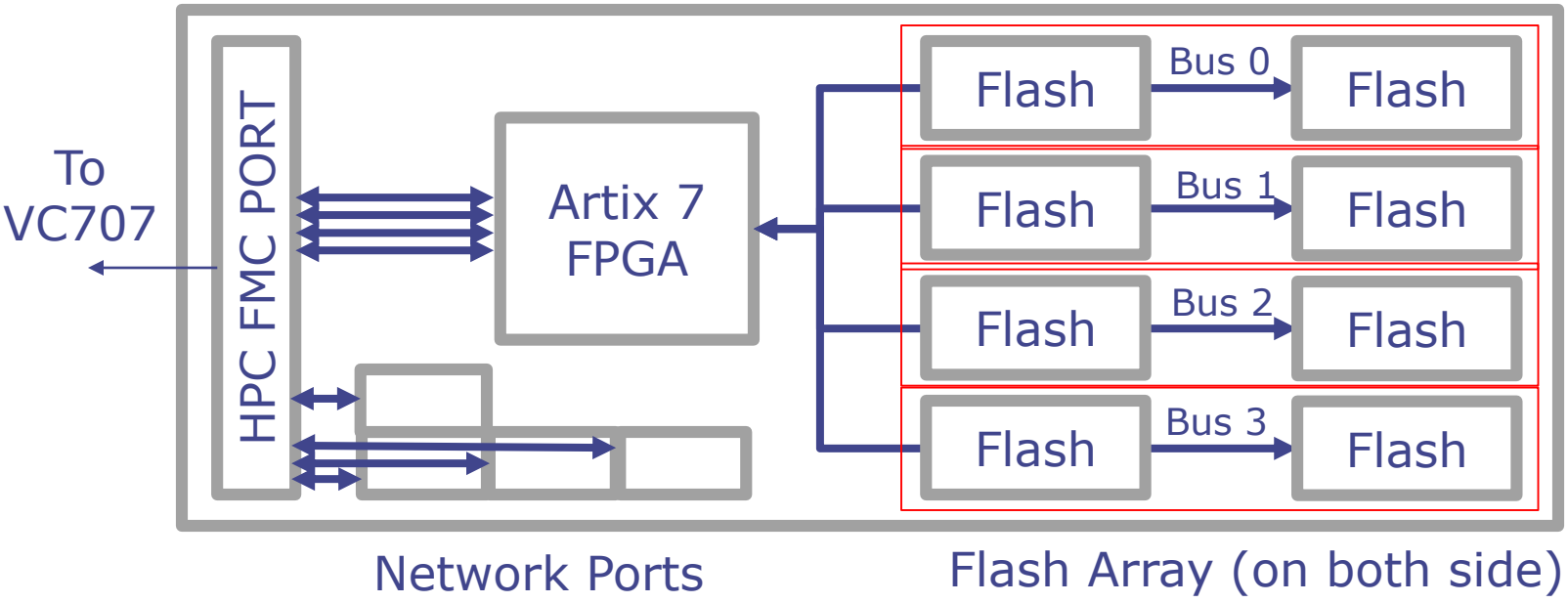
Latency Profile of Distributed Flash-based Analytics

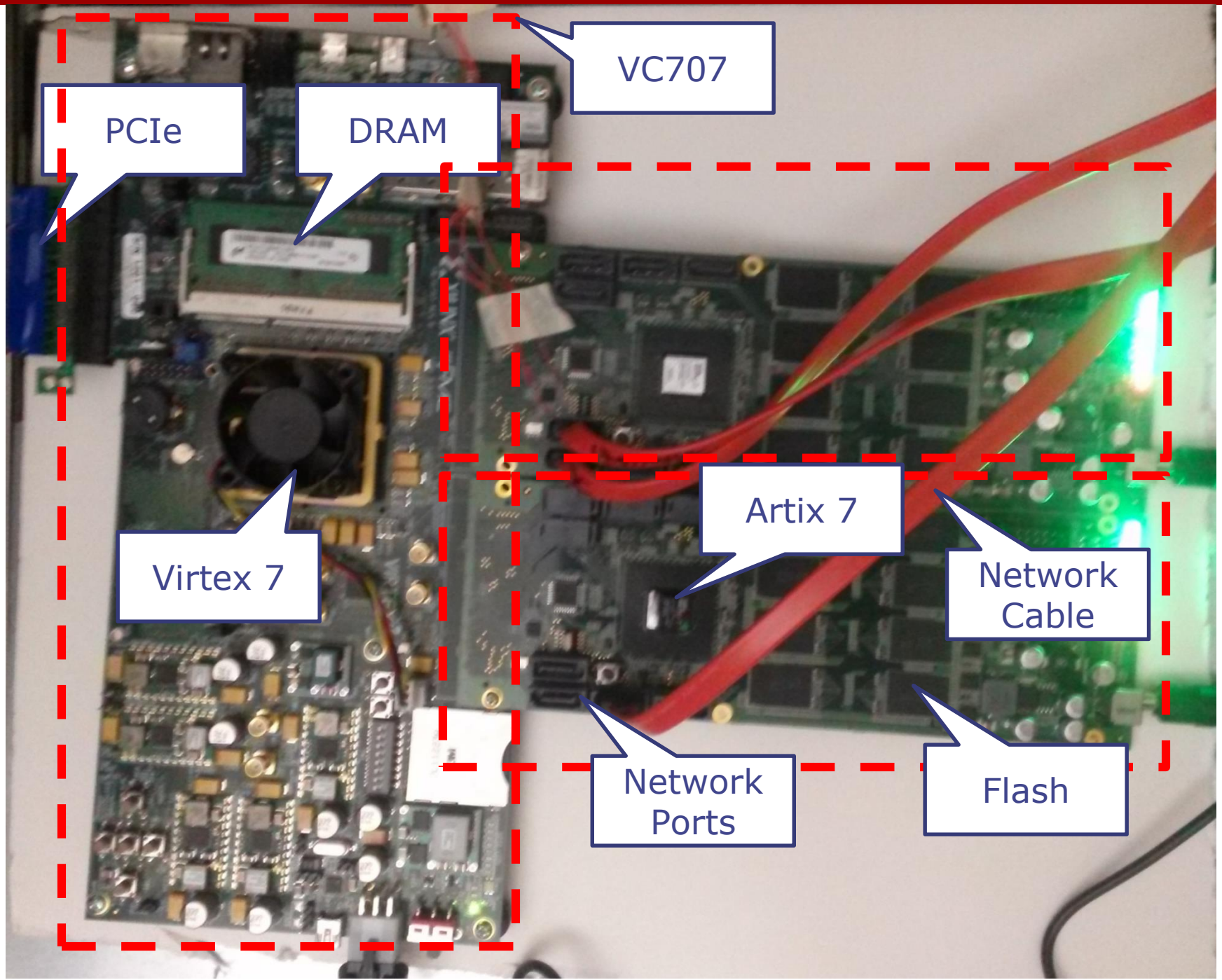
- **Architectural modifications can remove unnecessary overhead**
  - Near-storage processing
  - Cross-layer optimization of flash management software*
  - Dedicated storage area network
  - Accelerator



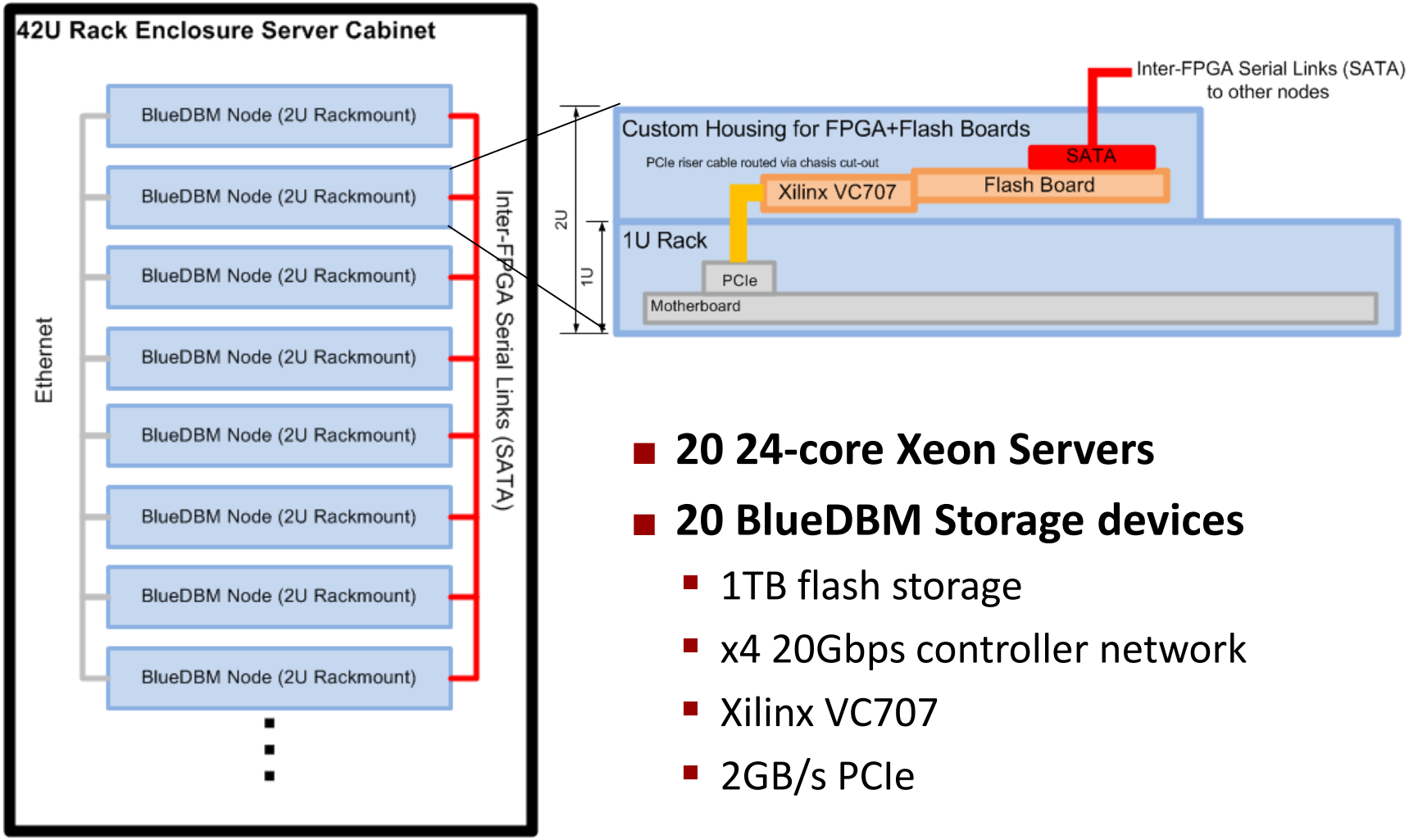
Custom Flash Card Had to be Built





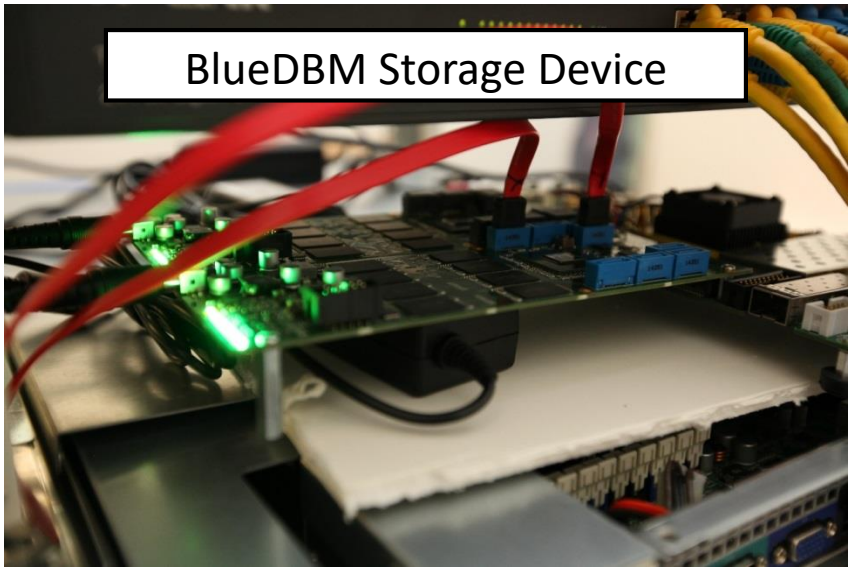

VC707
PCIe
DRAM
Virtex 7
Artix 7
Network Cable
Network Ports
Flash

BlueDBM: Platform with In-storage Processing and Inter-controller Networks



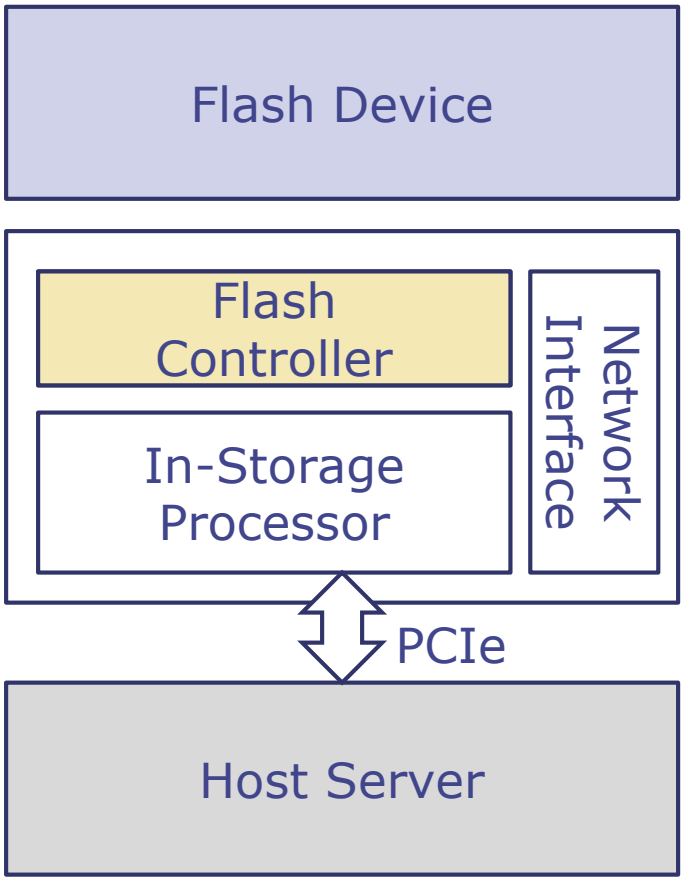
- **20 24-core Xeon Servers**
- **20 BlueDBM Storage devices**
  - 1TB flash storage
  - x4 20Gbps controller network
  - Xilinx VC707
  - 2GB/s PCIe

BlueDBM: Platform with In-storage Processing and Inter-controller Networks



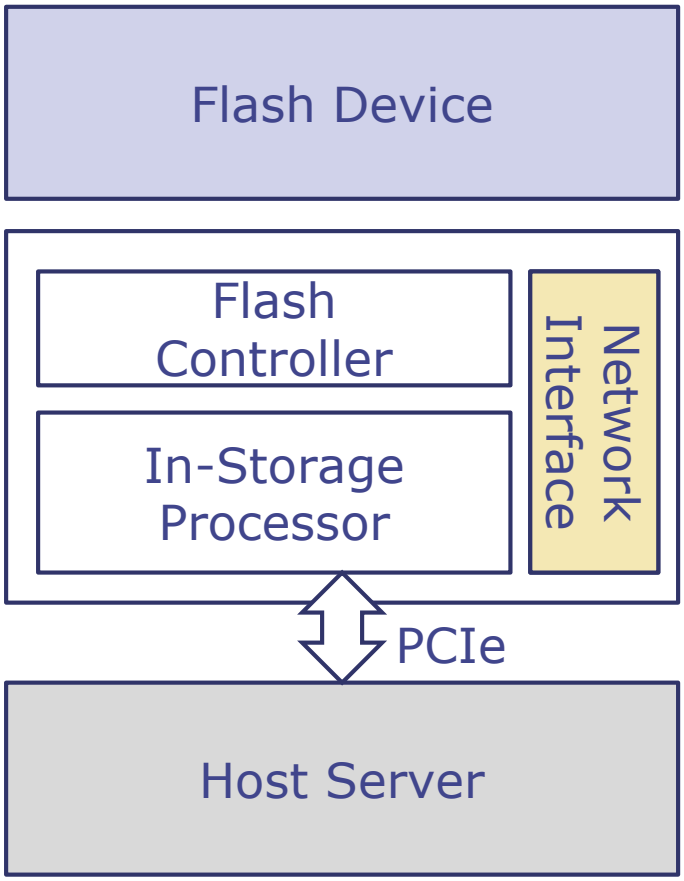
- **20 24-core Xeon Servers**
- **20 BlueDBM Storage devices**
  - 1TB flash storage
  - x4 20Gbps controller network
  - Xilinx VC707
  - 2GB/s PCIe

BlueDBM Node Architecture

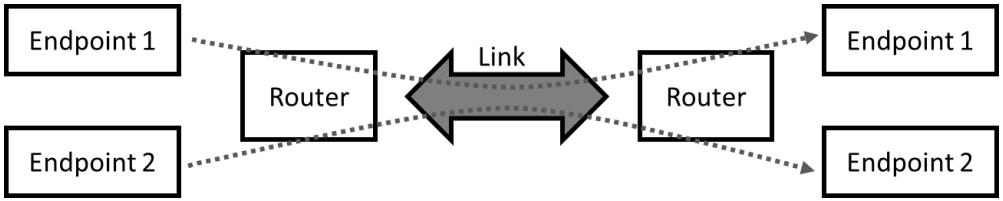


- **Lightweight flash management with very low overhead**
  - Adds almost no latency
  - ECC support

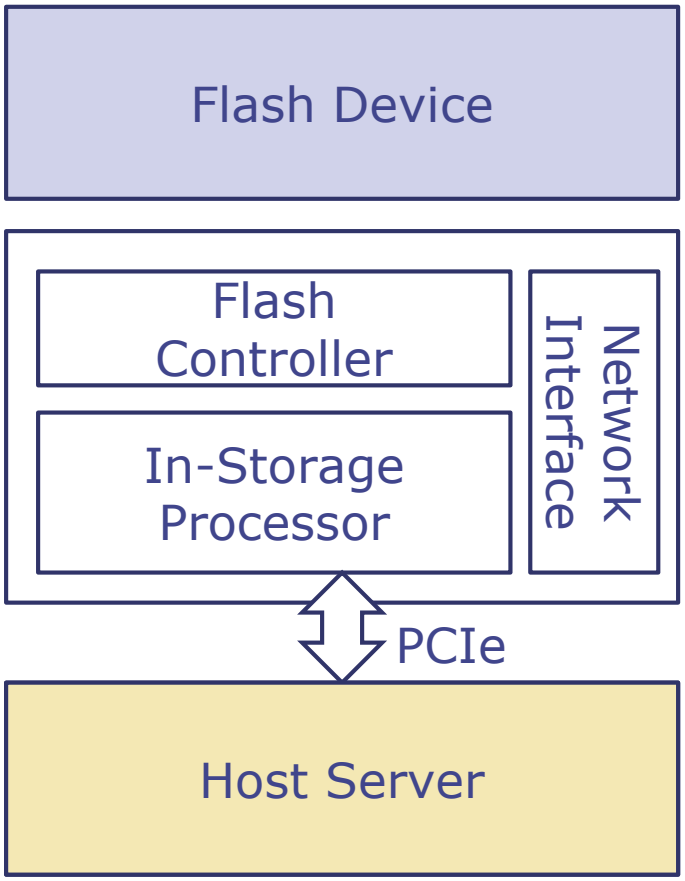
BlueDBM Node Architecture



- **Custom network protocol with low latency/high bandwidth**
  - x4 20Gbps links at 0.5us latency
  - Virtual channels with flow control

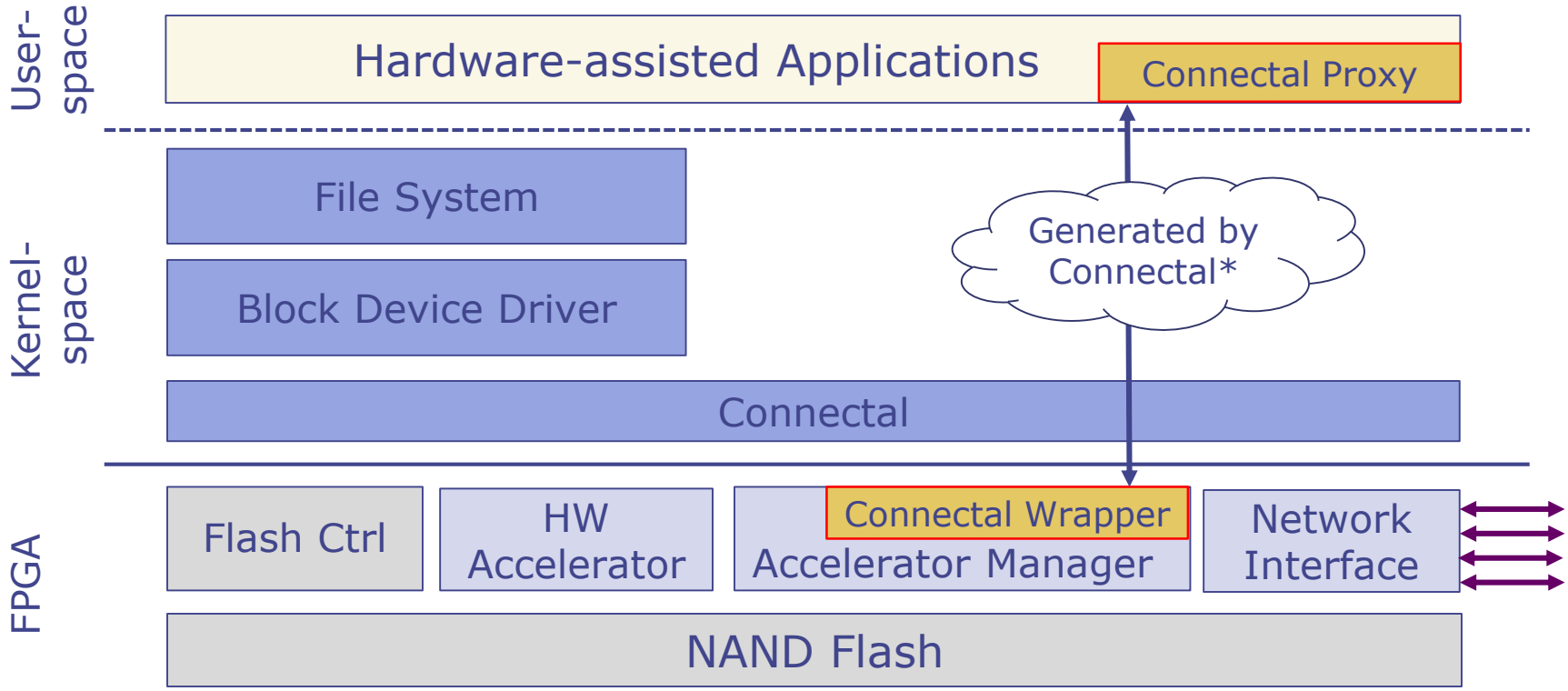


BlueDBM Node Architecture



- **Software has very low level access to flash storage**
  - High level information can be used for low level management
  - FTL implemented inside file system

BlueDBM Software View



- BlueDBM provides a generic file system interface as well as an accelerator-specific interface (Aided by Connectal)

Power Consumption is Low

Component	Power (Watts)
VC707	30
Flash Board (x2)	10
Storage Device Total	40

Storage device power consumption
is a very conservative estimate

Component	Power (Watts)
Storage Device	40
Xeon Server	200+
Node Total	240+

GPU-based accelerator will double the power

BlueDBM Applications

- **Content-based image search**
  - Faster flash with accelerators as replacement for DRAM-based systems
- **Graph analytics**
  - Benefits of lower latency access into distributed flash for computation on large graphs

Content-based Image Retrieval

- **Takes a query image and returns similar images in a dataset of tens of million pictures**
- **Image similarity is determined by measuring the distance between histograms of each image**
  - Histogram is generated using RGB, HSV, “edgeness”, etc
  - Better algorithms are available!

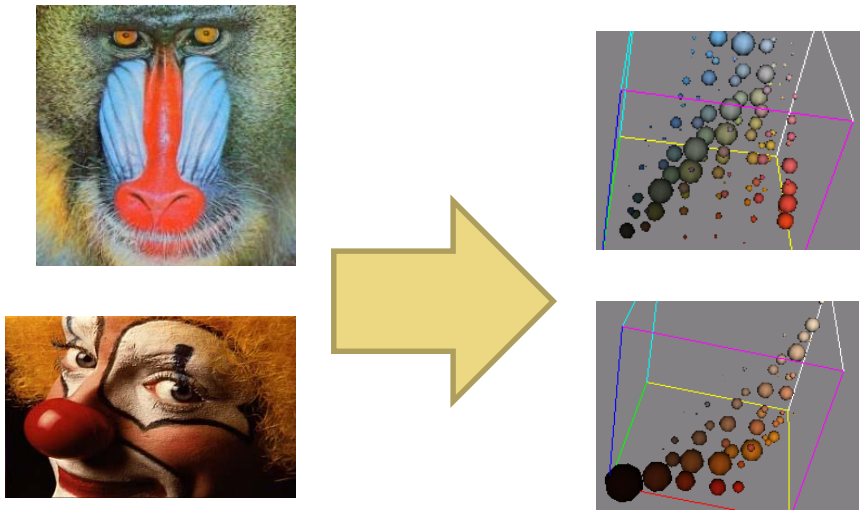


Image Search Accelerator

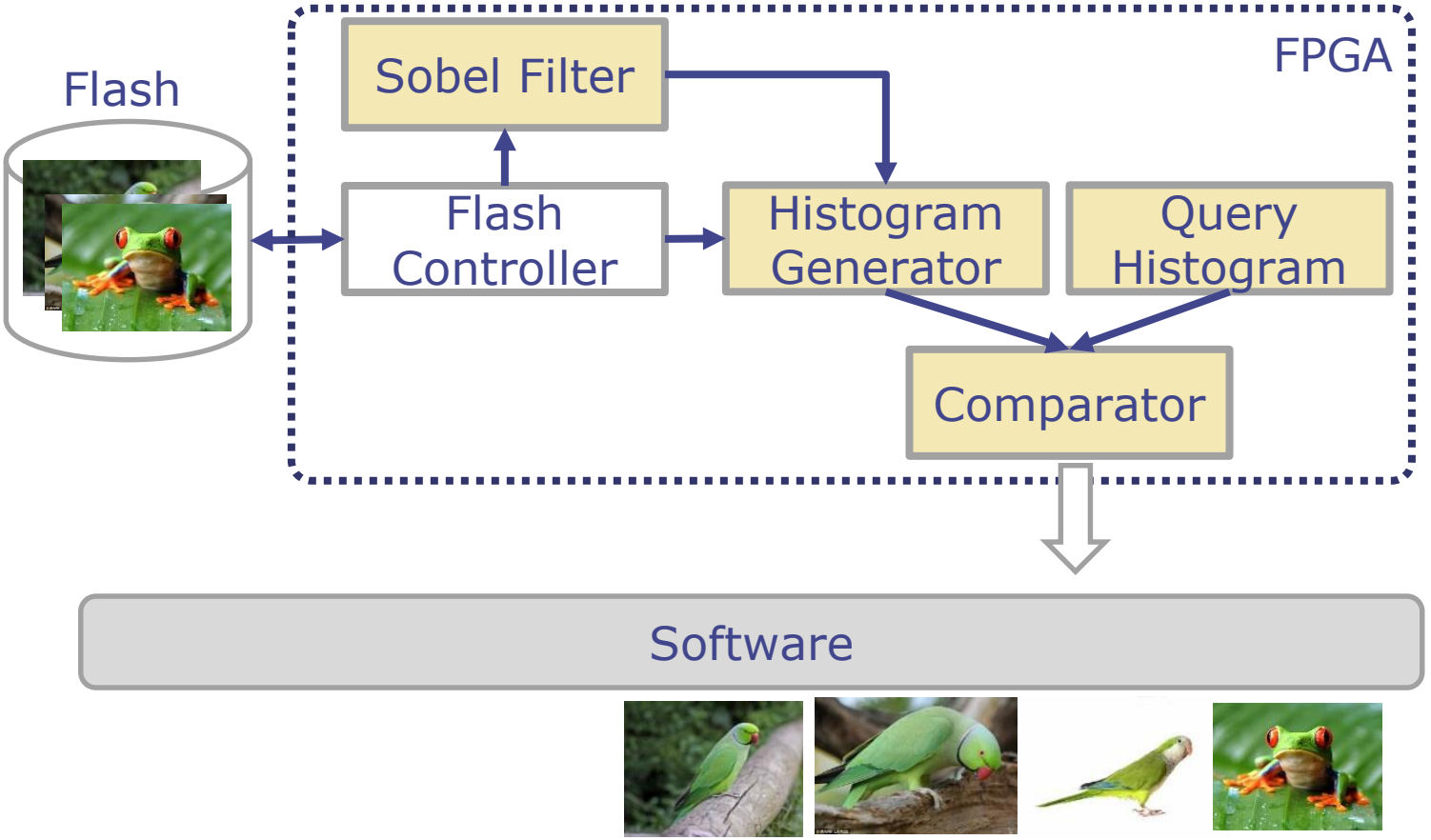
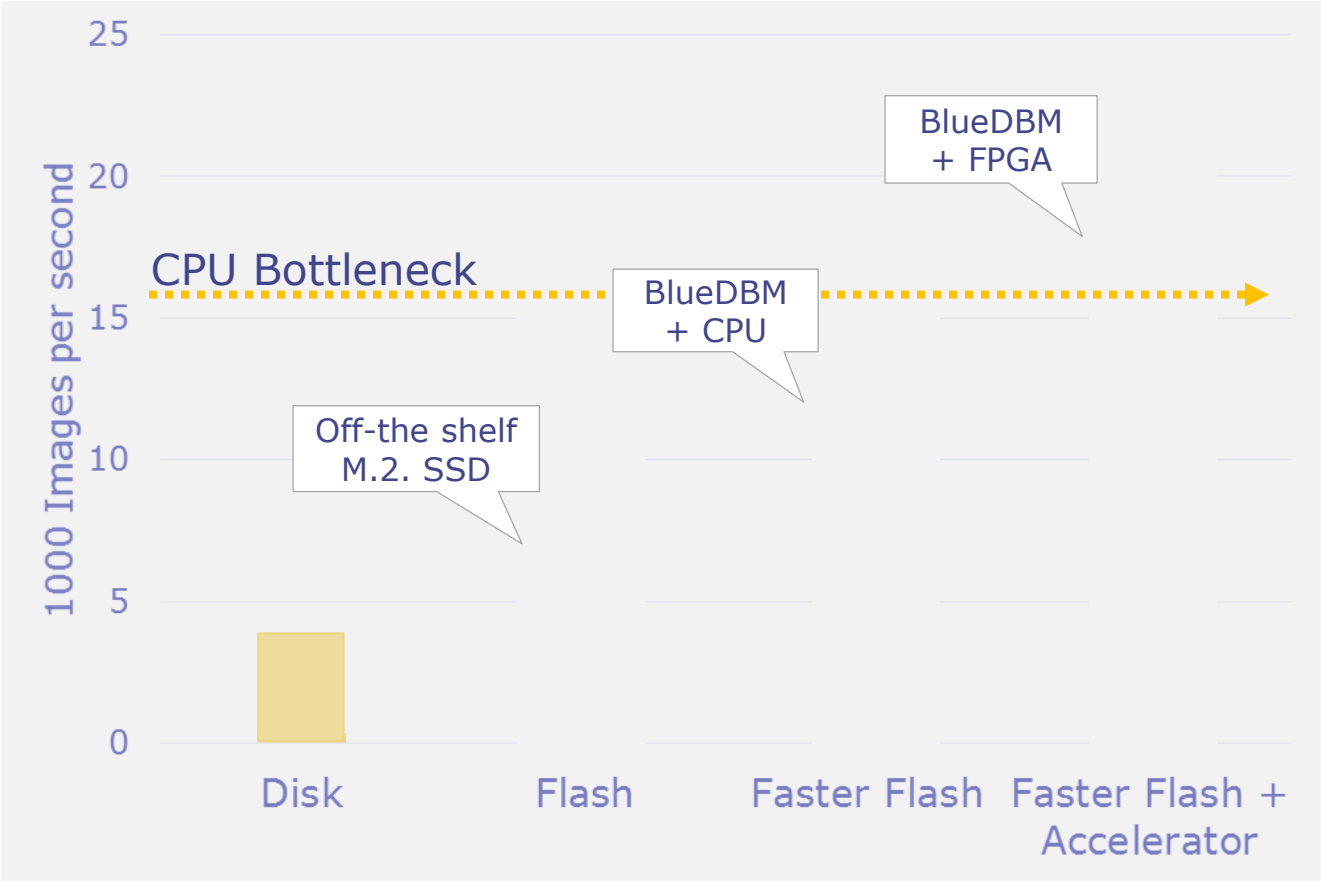


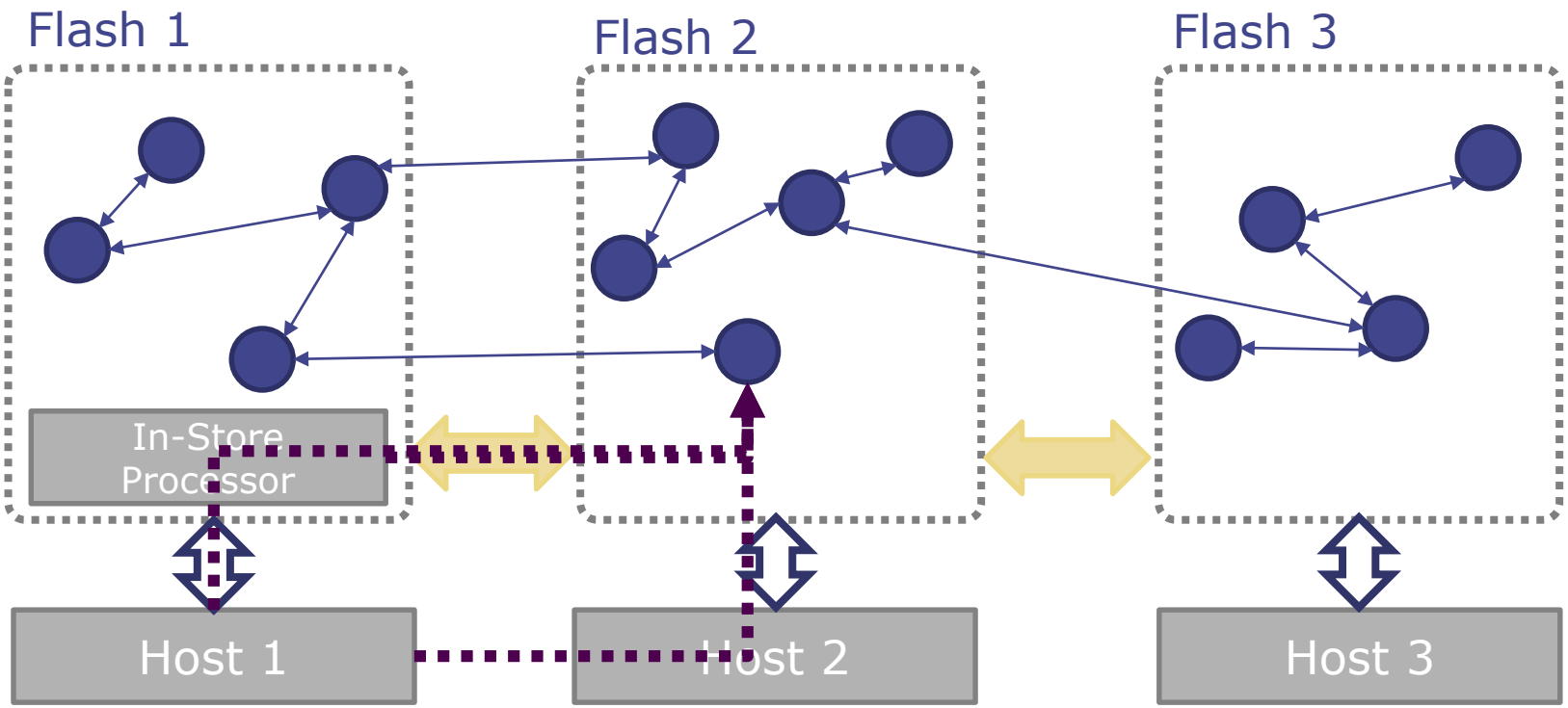
Image Query Performance



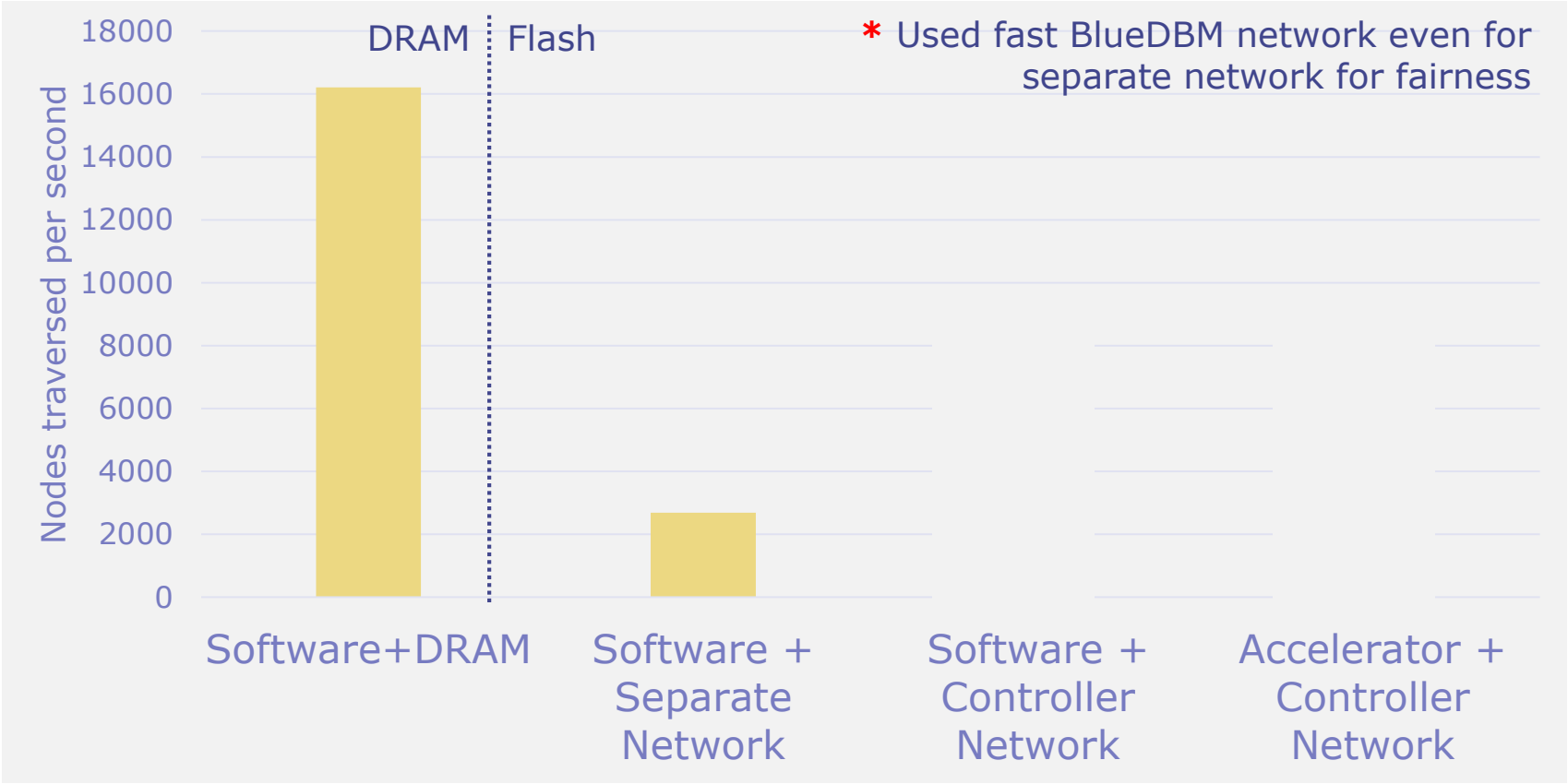
- **Faster flash with acceleration can perform at DRAM speed**

Graph Traversal

- **Very latency-bound problem, because often cannot predict the next node to visit**
  - Beneficial to reduce latency by moving computation closer to data



Graph Traversal Performance



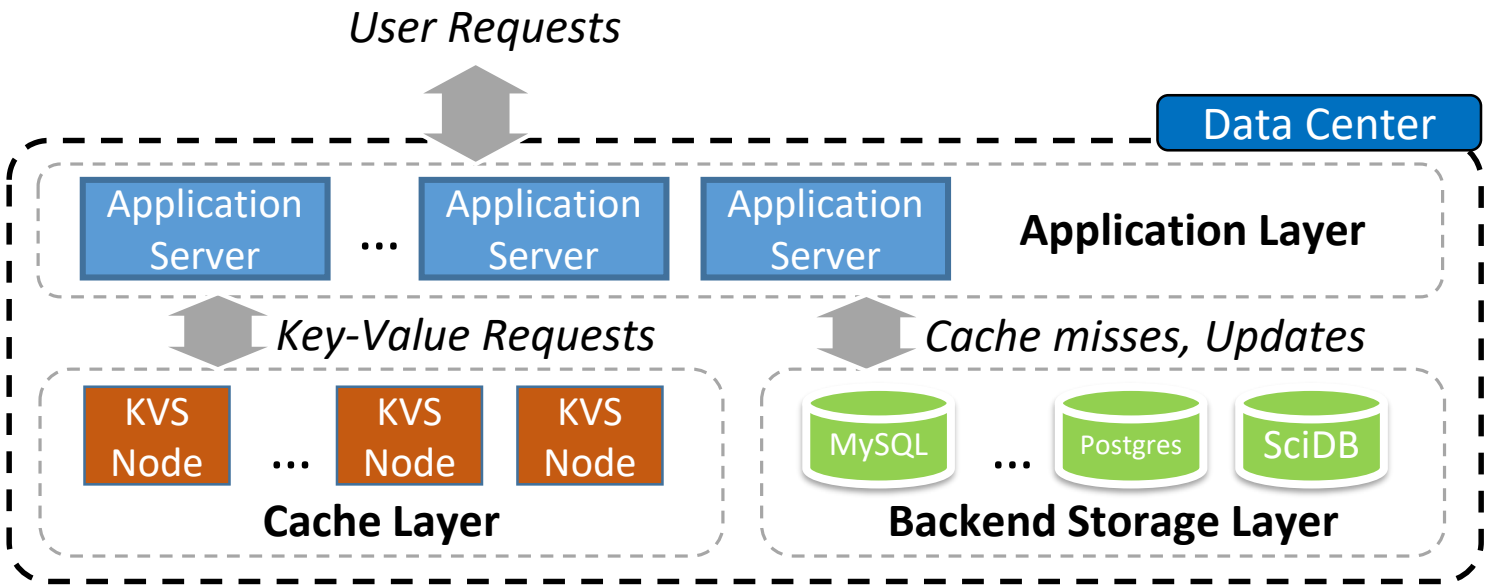
- Flash based system can achieve comparable performance with a much smaller cluster

Outline

- Why In-storage Processing?
- **Case Studies**
  - BlueDBM: Big-data Analytic Platform with In-storage Processing
  - **BlueCache: Hardware-accelerated Key-value Store**
  - Biscuit: User-programmable In-storage Processing Framework
  - IESSD: Inference-Enabled SSDs

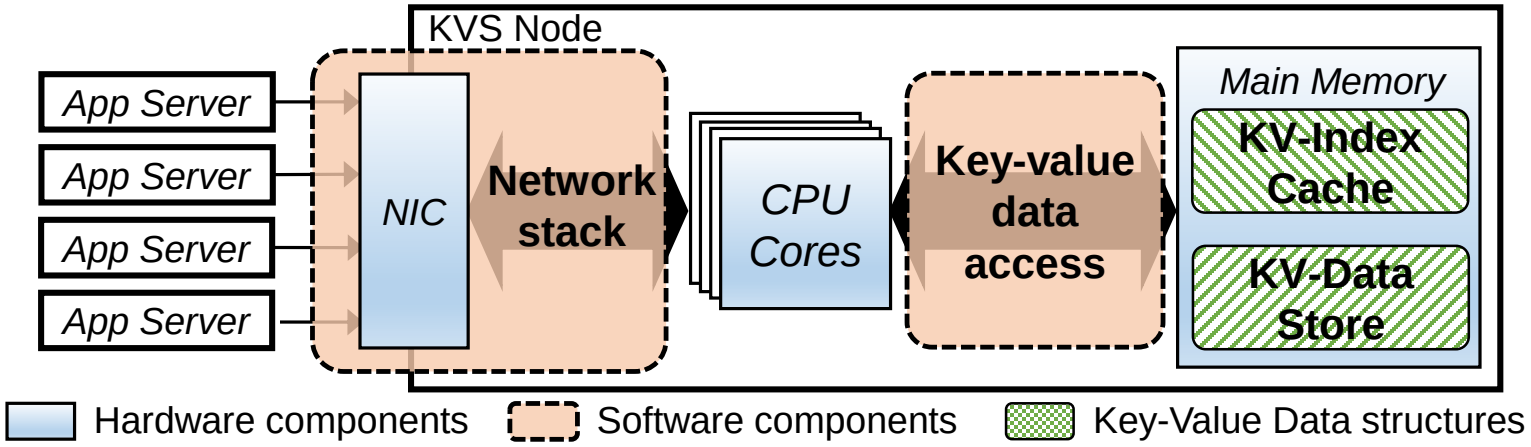
KVS Cache in Data Center

- **Application servers transforms a user *read-request* into multiple GET requests**
  - A cache miss requires 1) backend queries and 2) cache refill by a SET request
- ***Write-request* deletes relevant KV pairs and updates the backend**
  - Subsequent read-request is a miss and automatically refill the cache



X86-based KVS Architecture

- **Application servers communicate with the KVS server via NIC**
- **x86 processors processes key-value requests stored in memory**
  - KV-Index Cache only keeps the pointers to the objects
  - KV-Data Store keeps the object data



OOO Processors Efficient for KVS?

- **State-of-the-art KVSs (>120+ MRPS)**
  - Direct packet injection from NIC to LLC (Intel DDIO)
  - MICA[ISCA'15] , Mega-KV[VLDB'15]

- **Higher throughput needs more hardware**
  - MICA uses 24 cores and 12 10GbE to produce 120MRPS
  - Mega-KV uses 2 GTX 780 GPUs to produce 160MRPS
  - 399.2W for MICA, and 899.2W for Mega-KV

Hardware accelerators can efficiently process KVS queries with low power

KVS cache's Performance Depends on Aggregate Memory Size

- **Popular Solution: Add more DRAMs**
  - \+ Cache capacity increase with no performance loss
  - \- expensive and power hungry
  - \- limited by ~ 256GB/server
  - \- performance drops dramatically if data does not fit in DRAM

working set이 memcached보다 크면 miss 확

- **Alternative solution: Use flash memory**
  - \+ More cache capacity, 10X cheaper than DRAM
  - \+ 10-100X lower power consumption than DRAM
  - \+ 100X higher storage density than DRAM
  - \- Slower than DRAM

Existing Flash-based Solution

- **SSDs as a simple swap device for DRAM is undesirable for KVS**
  - Small random writes leads to poor performance

- **A better solution is to move KV-Data Store to flash**
  - KV-Index Cache keeps abbreviated keys and data pointers in DRAM
  - KV-Data Store keeps both keys and values as a log on flash
  - Keys need to be compared to produces a hit
  - FlashStore[VLDB'10], Twitter's Fatcache, …

FTL is Burdensome

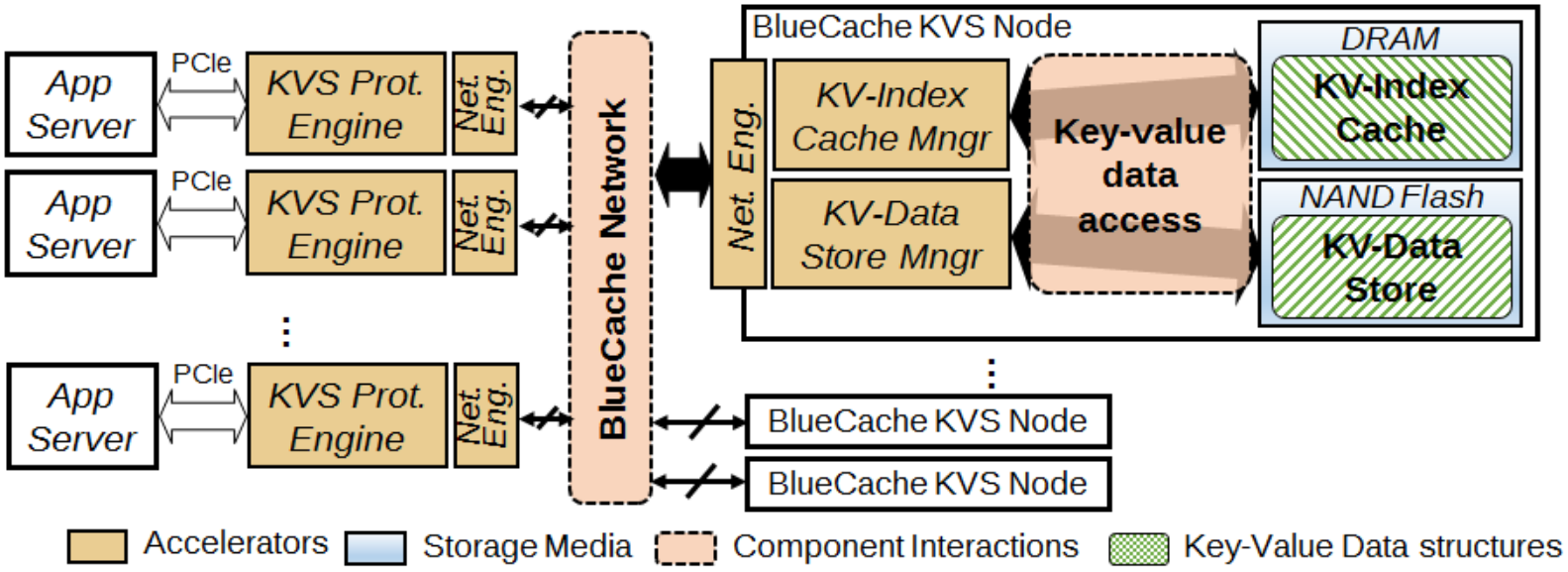
- **Flash translation layer is deployed in SSDs to provide hard-drive-like behaviors**
  - Manages wear-leveling, bad block management, address mapping and garbage collection
  - Uses significant hardware (multi-core ARM with a large DRAM)

- **Benefits have been shown for no-FTL flash-based file systems, such as SDF [ASPLOS'15], AMF [FAST'16]**

A KVS design that eliminates FTL, and *directly* manages raw NAND flash chips can harness the full device performance

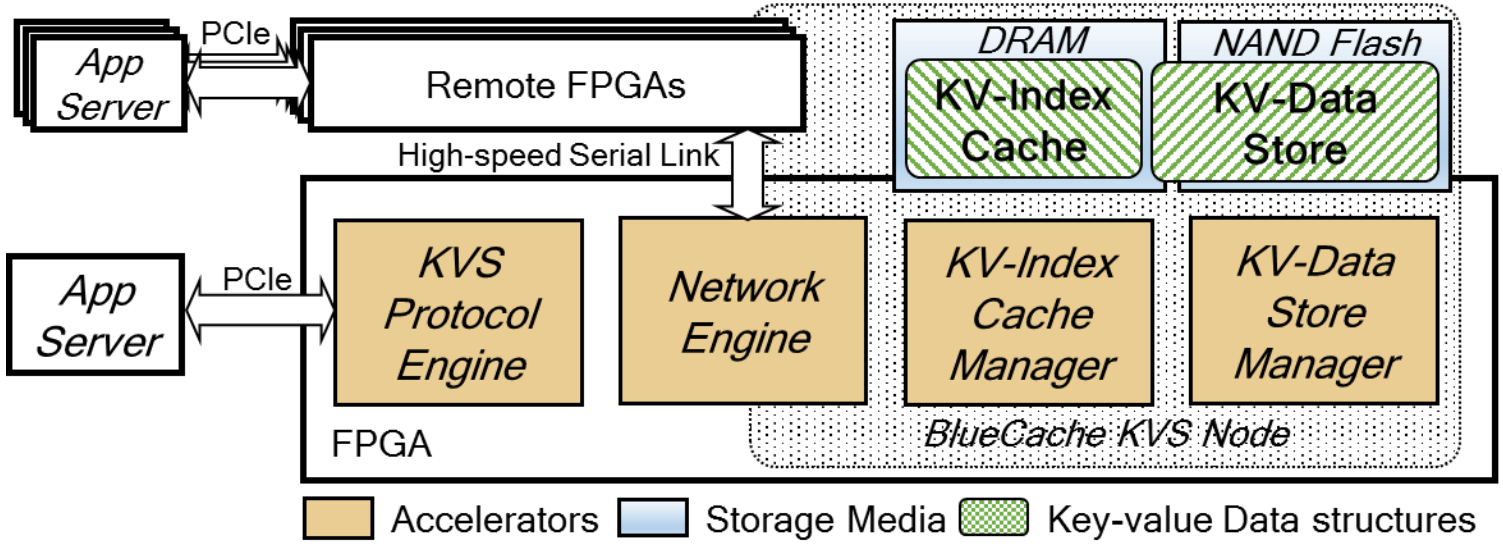
BlueCache Architecture

- **A Distributed NoFTL flash-based KVS with HW accelerators**
  - Application servers uses KVS Protocol Engine via PCIe to communicate with KVS nodes
  - KVS node uses hardware accelerators which directly manage KV pairs on NAND-flash array of chips



BlueCache Architecture (Cont.)

- **BlueCache implements hardware accelerators on a single FPGA board on BlueDBM**
  - KVS Protocol Engine auto-batches KVS requests
  - In-storage Network Engines with dynamic allocation of virtual channels per application
  - 4-way set-associative KV-index cache
  - Log-structured KV-Data Store without FTL

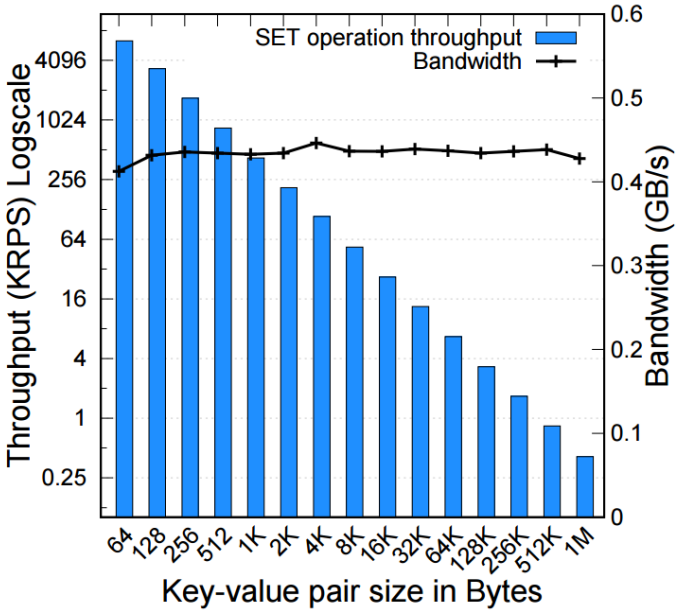


Software Interface

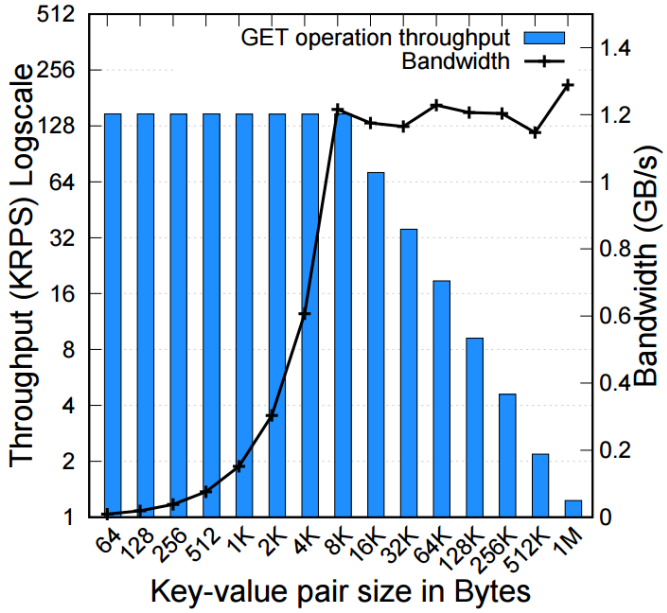
- **Provides a multi-thread C++ library to applications**
  - Provides three synchronous C++ functions of SET, GET, DELETE
  - Accessible to JAVA applications via JNI
  - Multiple threads needed to saturate BlueCache node bandwidth (>128 threads)

Single-node Performance

- **SETs saturate flash chip write bandwidth (430MB/s)**
- **GETs saturate flash chip random read bandwidth**
  - For KV Size < 8KB, operation throughput is 148 KRPS
  - For KV size >= 8KB, operation throughput limited by flash read bandwidth (1.2GB/s)



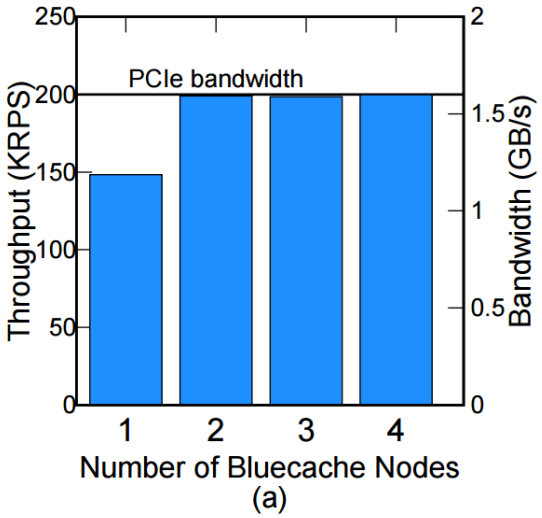
(a) SET Throughput

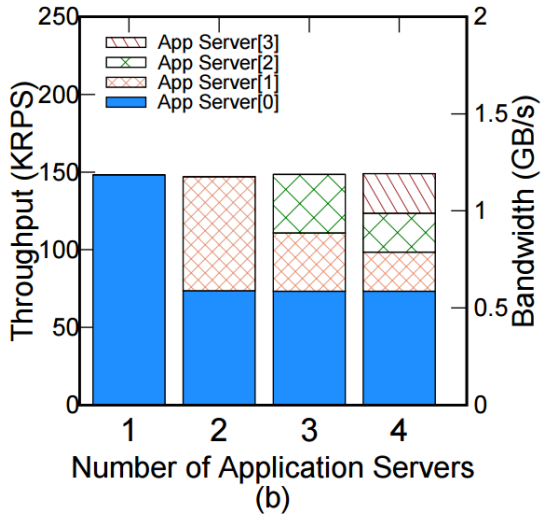


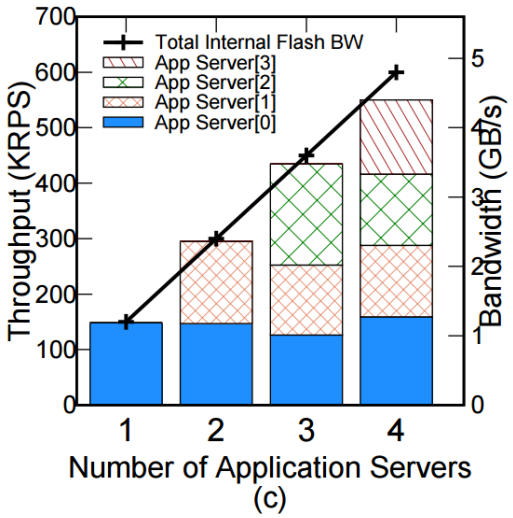
(b) GET Throughput

Multi-node Performance

- **Single application server, multiple Bluecache nodes**
  - Limited by PCIe bandwidth
- **Multiple application servers, single BlueCache node**
  - Limited by flash bandwidth
  - Favors application server where KVS is attached
- **Multiple application servers, multiple BlueCache nodes**
  - Bandwidth fair distributed







BlueCache is Low Power

- A BlueCache node uses 40 Watts at peak
- 20-node system consumes 800 Watts and provides 20TB storage
- > 25X lower GB/Watt than x86-based systems

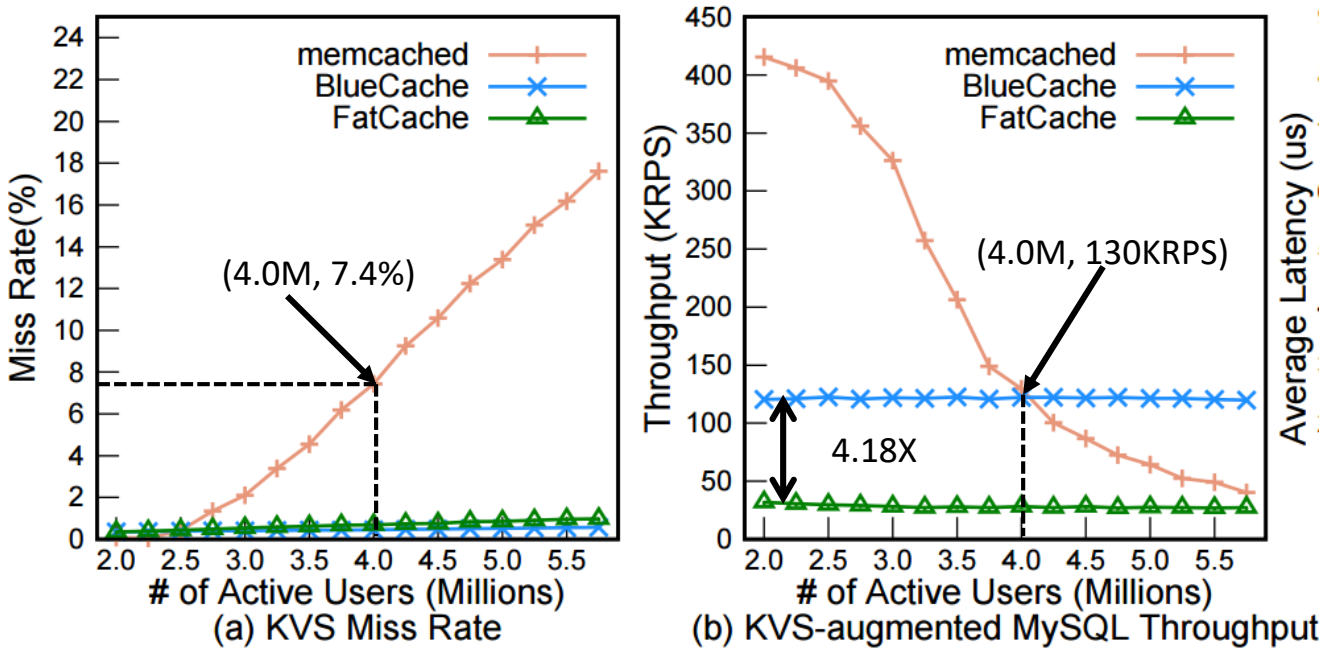
Platforms	Capacity (GB)	Power (Watt)	Capacity/Power (GB/Watt)
BlueCache	20,000	800	25.00
FlashStore [VLDB'10]	80	83.5	0.96
MICA [ISCA'15]	128	399.2	0.32
Mega-KV [VLDB'15]	128	899.2	0.14

A Social Networking Example

- **We ran BG [CIDR'13], a social networking benchmark, to evaluate the efficacy of BlueCache against other KVS systems**
  - Transforms facebook-like social actions into MySQL queries
  - Uses KVS to cache MySQL queries
  - Open source code

Key Result

- BlueCache can outperform in-memory KVS when the latter has more than 7.4% misses
- BlueCache is 4.18X faster than x86-based KVS with flash memory of similar performance

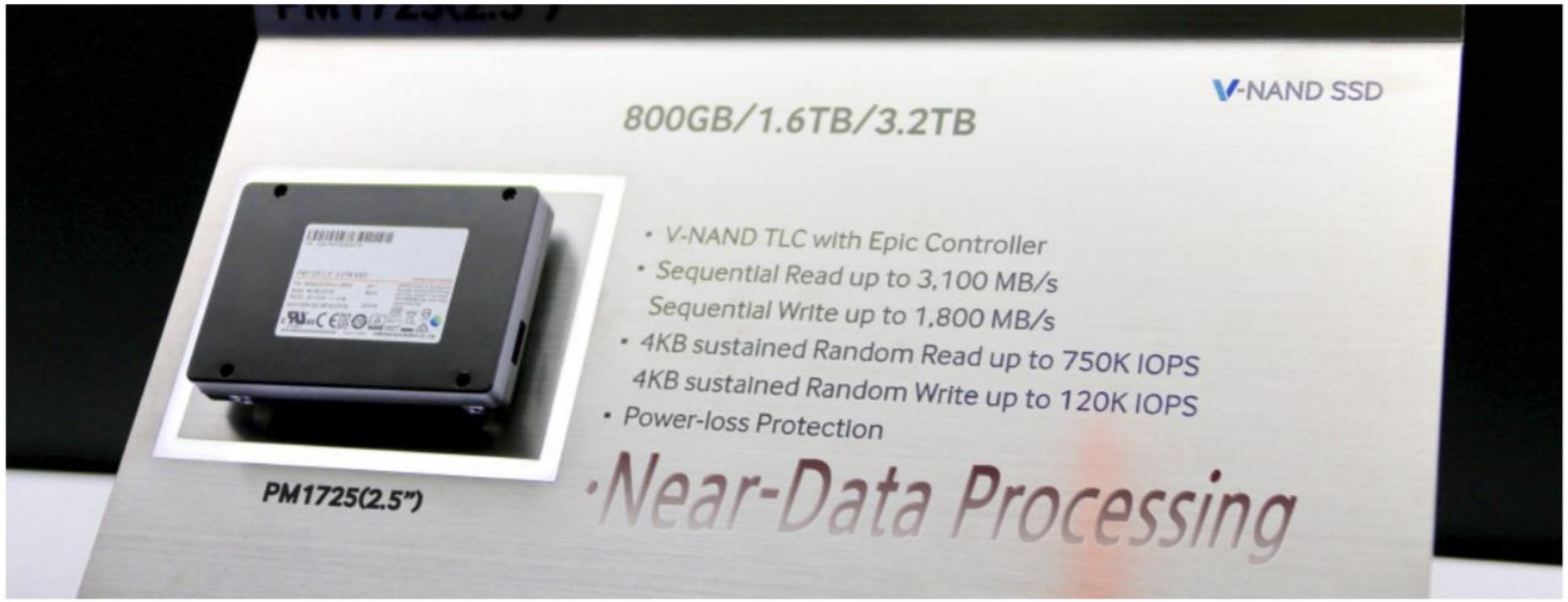


Outline

- Why In-storage Processing?
- **Case Studies**
  - BlueDBM: Big-data Analytic Platform with In-storage Processing
  - BlueCache: Hardware-accelerated Key-value Store
  - **Biscuit: User-programmable In-storage Processing Framework**
  - IESSD: Inference-Enabled SSDs

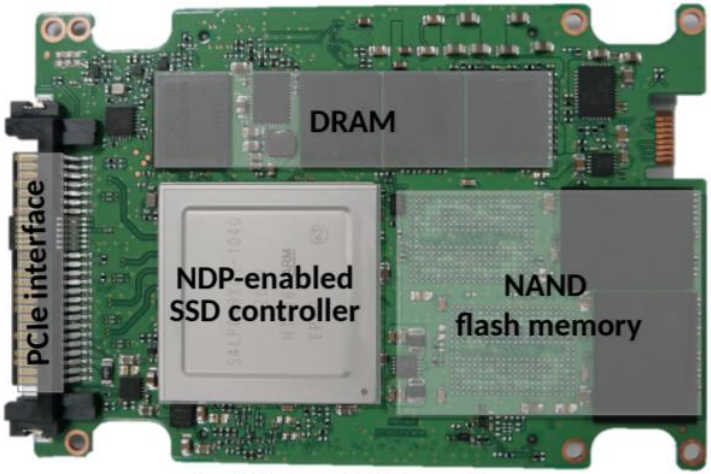
Biscuit

- **A user-programmable in-storage processing framework for SSDs and data-intensive applications**
  - C++ language support (C++ standard library)
  - Dynamic loading of user programs
  - Mutithreading, multi-core support



SSD Hardware

- **Use dedicated MPU for in-storage processing**



[Inside of PM1725]

Item	Description
Host interface	PCIe Gen.3 ×4 (3.2 GB/s)
Protocol	NVMe 1.1
Device density	1 TB
SSD architecture	Multiple channels/ways/cores
Storage medium	Multi-bit NAND flash memory
Compute resources for Biscuit	Two ARM Cortex R7 cores @750MHz with MPU
On-chip SRAM	< 1 MiB
DRAM	≥ 1 GiB

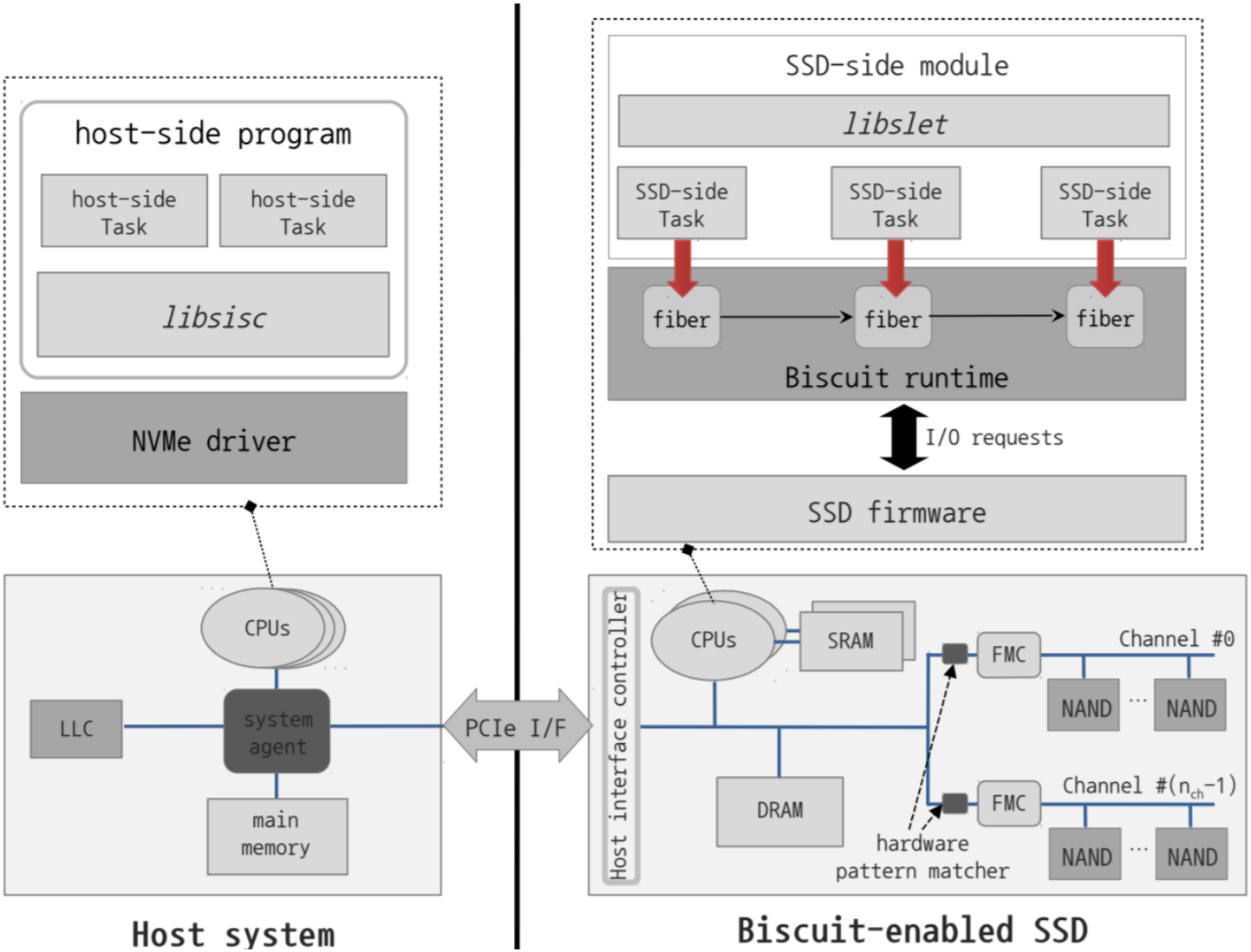
  - Limitations: Low compute power, no cache coherence, a small amount of fast memory, no MMU, and restrictive synchronization primitives

- **How to run in-storage software logics inside Biscuit?**

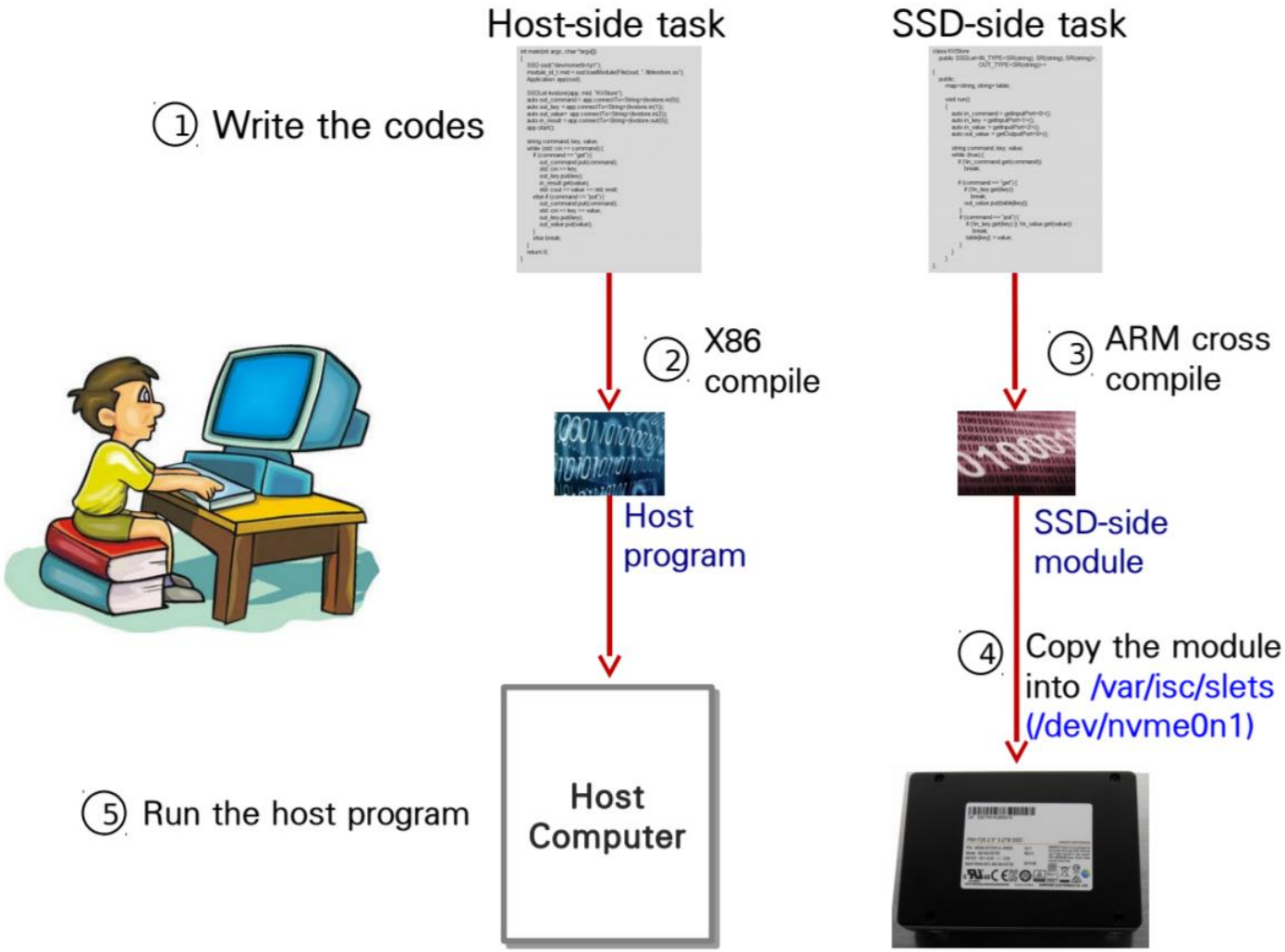
Biscuit Runtime

- **Cooperative multithreading**
  - A limited form of multithreading (fiber as a scheduling unit)
  - Less context switching overhead
  - Safe resource sharing without locking
- **Shared nothing architecture**
  - Enforced by the programming model and APIs
  - C++11 move semantics supported
- **Dynamic loader for user programs**
  - User program as position-independent code (PIC)
  - Symbol relocation to locate each program in a separate address space

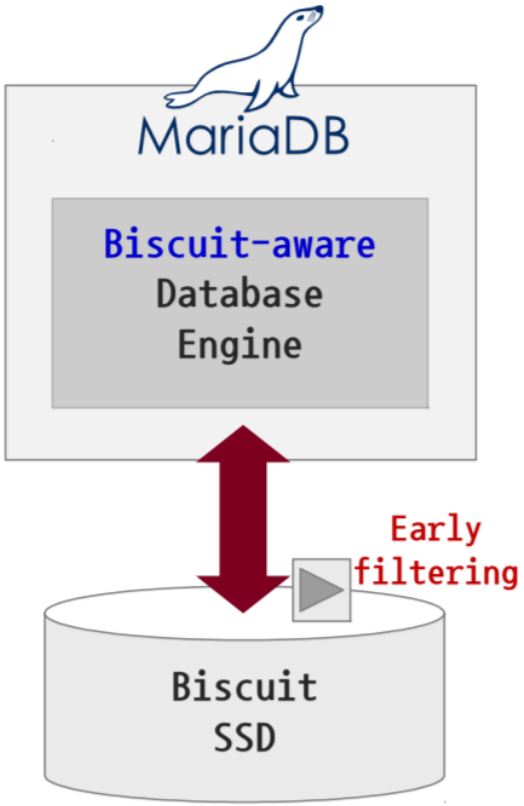
Biscuit System Architecture



Development Process



Application Level Results



- **Data analytics with a real DB engine**
  - MariaDB 5.5.42
    - TPC-H dataset at a scale factor of 100 (160 GiB)
  - Modified the query planner to
    - Identify a candidate table amenable for offloading;
    - Estimate its selectivity1 using a sampling method;
    - Determine whether the table is indeed a good target (based on a selectivity threshold);
    - and finally offload the identified filter to the SSD.

Sample Queries

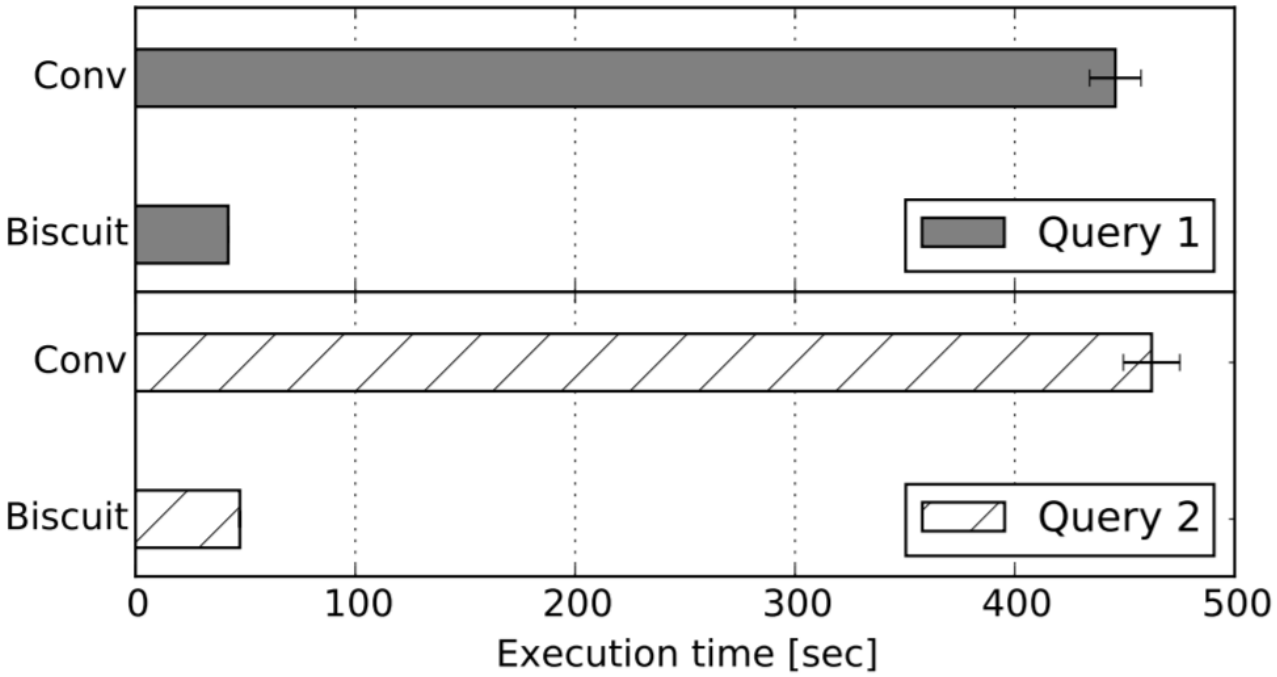
- Query 1 (single filter predicate, selectivity for shipdate: 0.02)

```
SELECT l_orderkey, l_shipdate, l_linenumber
FROM lineitem
WHERE l_shipdate = '1995-1-17'
```

- Query 2 (more complex WHERE clause, selectivity for shipdate: 0.04)

```
SELECT l_orderkey, l_shipdate, l_linenumber
FROM lineitem
WHERE (
       l_shipdate = '1995-1-17' OR
       l_shipdate = '1995-1-18’
       )
AND
       (l_linenumber = 1 OR l_linenumber = 2)
```

Experimental Results



- Biscuit achieves speed-ups of about 11× and 10×
  - Large internal bandwidth of Biscuit
  - Rapid scanning of the dataset by the hardware pattern matcher
- Execution times on Biscuit were very consistent

Outline

- Why In-storage Processing?
- **Case Studies**
  - BlueDBM: Big-data Analytic Platform with In-storage Processing
  - BlueCache: Hardware-accelerated Key-value Store
  - Biscuit: User-programmable In-storage Processing Framework
  - **IESSD: Inference-Enabled SSDs**

IESSD: Inference-Enabled SSD

- **IESSD is in capable of performing DNN operations inside an SSD, avoiding frequent data movements between application servers and data storage**

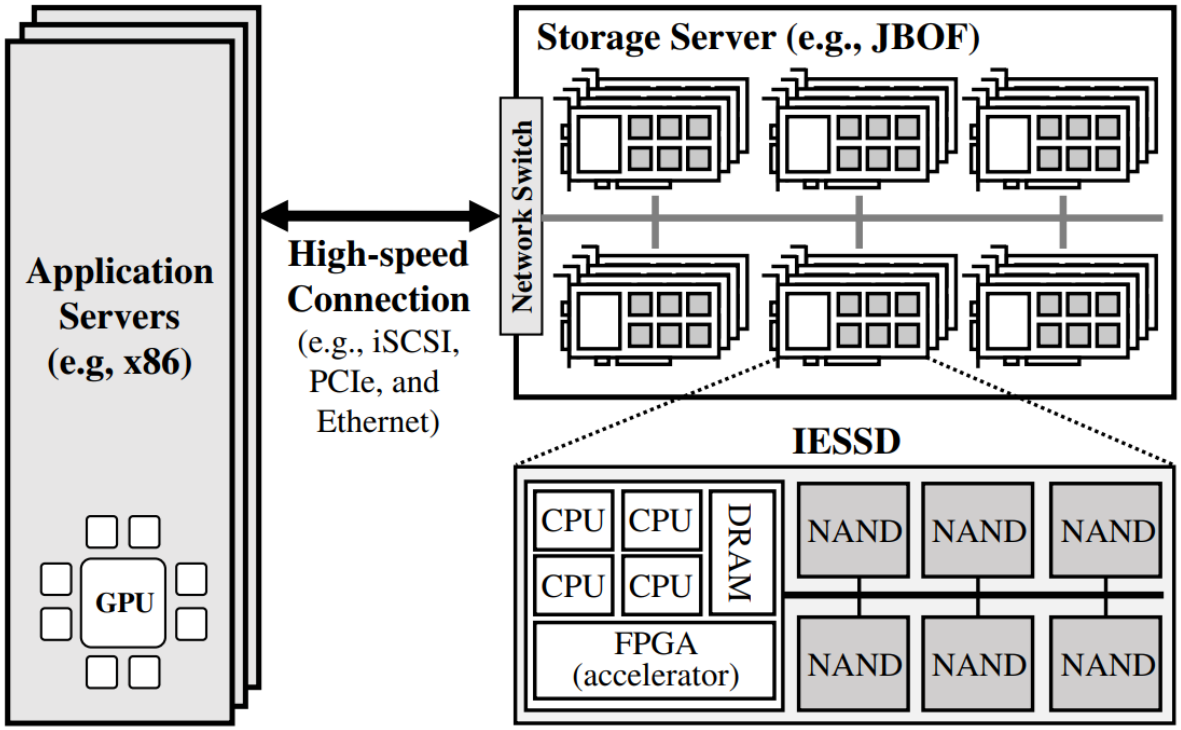
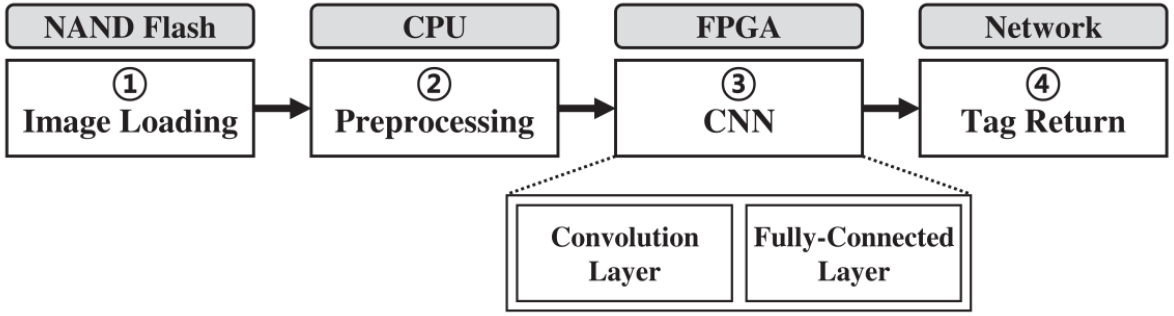


Fig. 1. Overall architecture of AI acceleration system with IESSDs

Image Classification with DNN

Image loading - > Preprocessing - > CNN - > Tag Return

- **Four main steps of the image tagging process and entities that perform at each step in the baseline IESSD design**



- **Latency of each step of image classification**

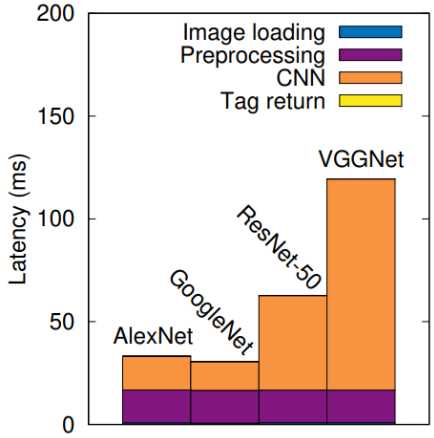


Fig. 3. Latency of each step

Optimization

- To maximally exploit HW resources available in an SSD, we employ three optimization techniques

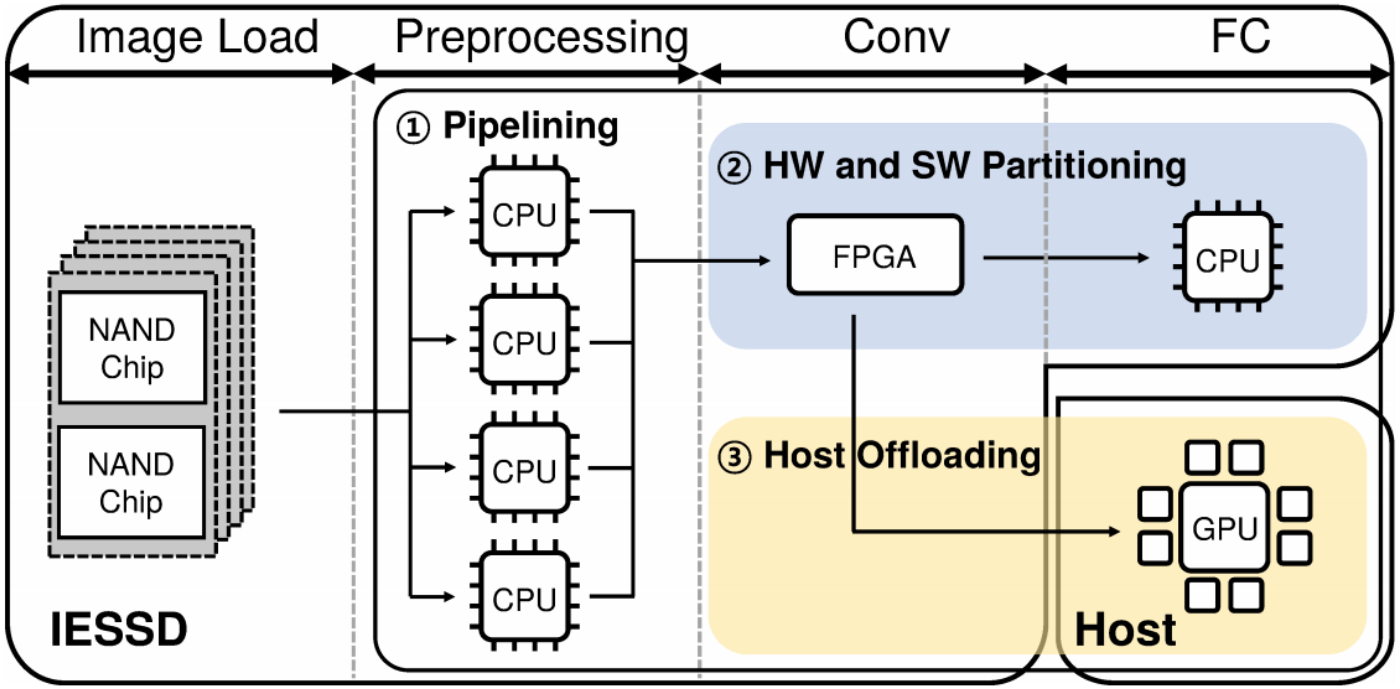
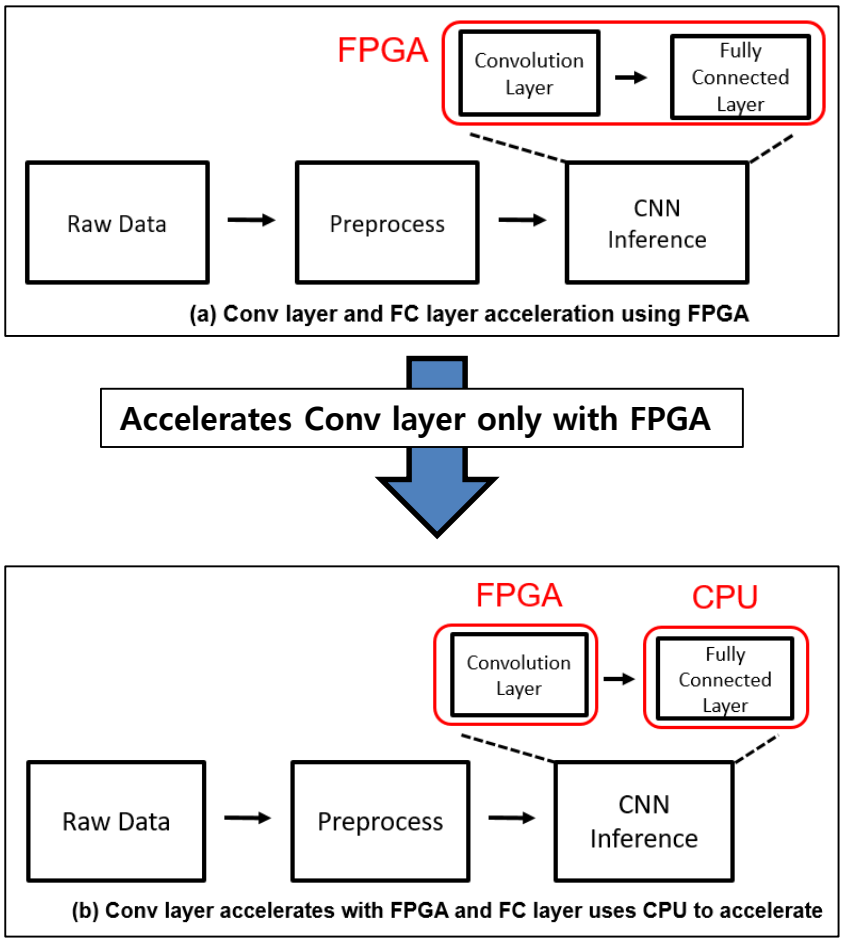


Fig. 4. Three optimization techniques in IESSD

HW-SW Partitioning

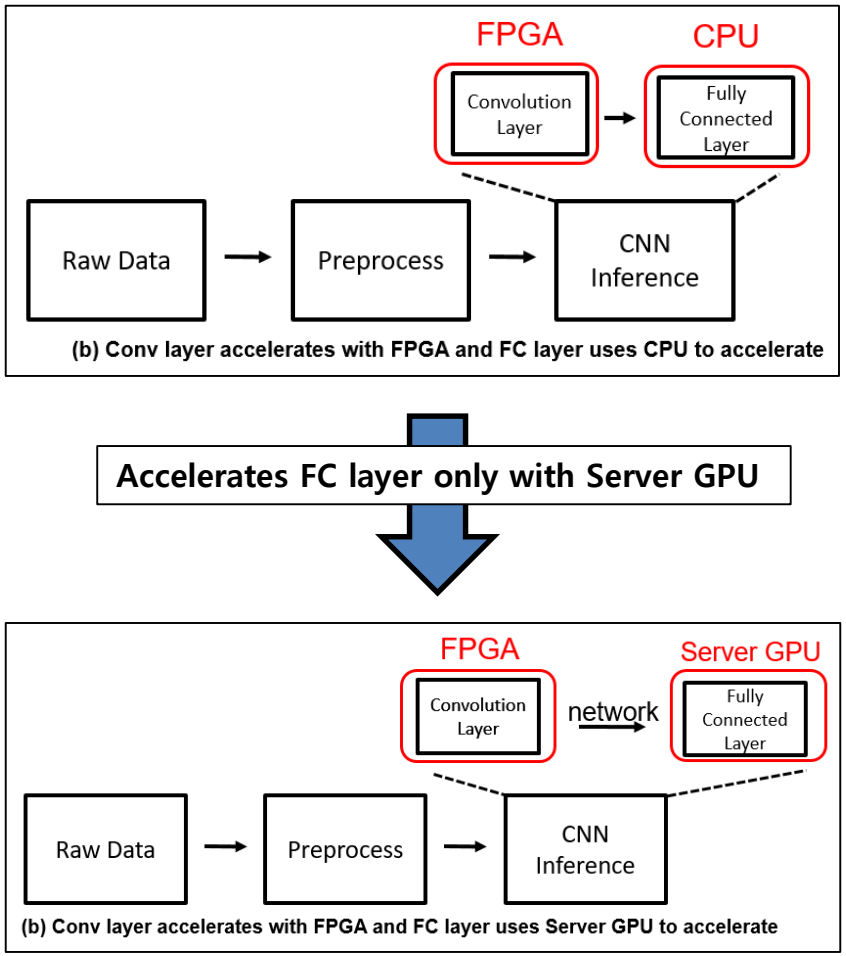
- **Run the convolution layer in FPGA and the fully connected layer in CPU → Better utilize FPGA**



[HW - SW Partitioning]
conv layer는 FPGA에서 하고 FC는 CPU에서 하자.
이러면 FPGA를 더 잘 활용할 수 있다.

Host Offloading

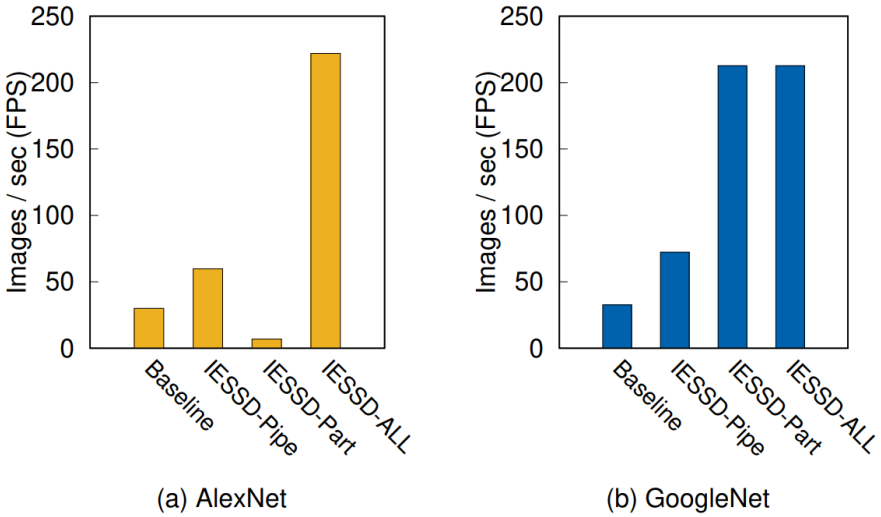
- **Split the layers and offload the fully connected layer to the host GPU**



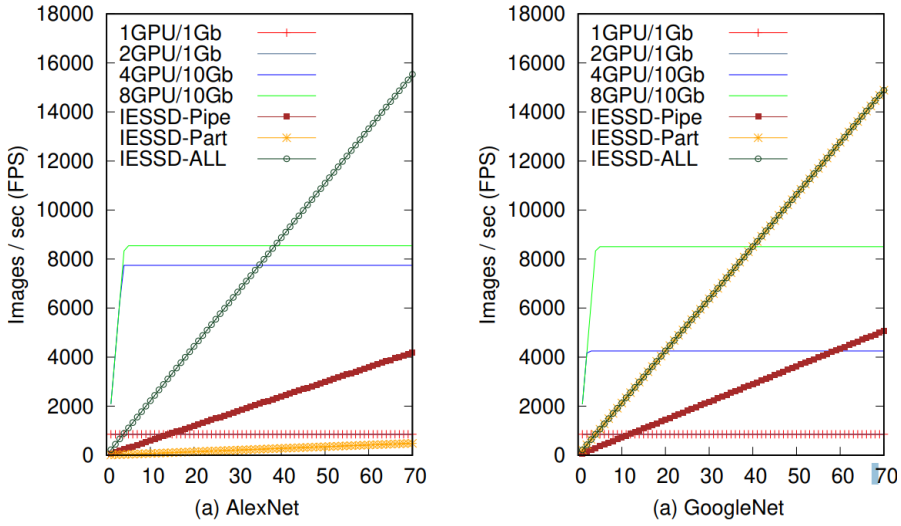
그런데 FC를 CPU에서 안하고 Server GPU로까지 network로 보내는
Host offloading

Experimental Results

- IESSD-ALL exhibited the best performance, achieving 221.9 and 212.7 throughput (images/sec) for AlextNet and GoogleNet
- IESSD-ALL outperformed 8 GPU/10 GbE when more than 39 IESSDs were employed in AFA
- IESSD consumes 1.78x and 5.31x less power than 8 GPUs at the cross points where IESSD overtakes 8 GPU/10 GbE



Single IESSD performance



Multiple IESSD performance

End of Chapter 11