STORAGE DEVELOPER CONFERENCE

SDC 22

Fremont, CA
September 12-15, 2022

BY Developers FOR Developers

A SNIA Event

RETINA: Exploring Computational Storage (SmartSSD) Usecase

Vishwanath Maram, Director of Software Engineering
Samsung Semiconductor Inc

Changwoo Min, Assistant Professor
Virginia Tech

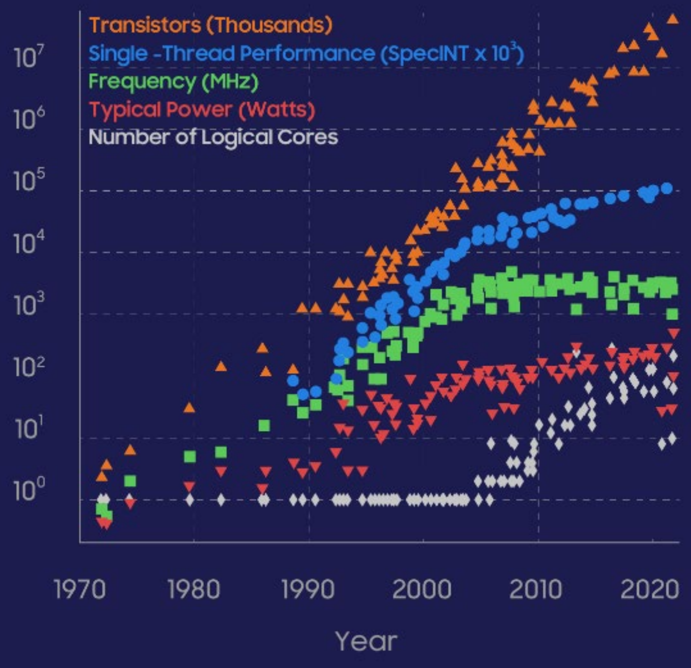
Agenda

- Industry Trend
  - Exponential Data Growth vs. Limited Processor Scaling
- Computational Storage
  - SNIA CSAPI
  - Samsung SmartSSD
- RETINA: End-to-End (Compute+Storage) Framework for CS
  - Cross-Layered RETINA Key-Value Store
  - Dynamic Composable RETINA Computational Pipeline
- Summary
- Call for Action

Exponential Data Growth vs. Limited Processor Scaling

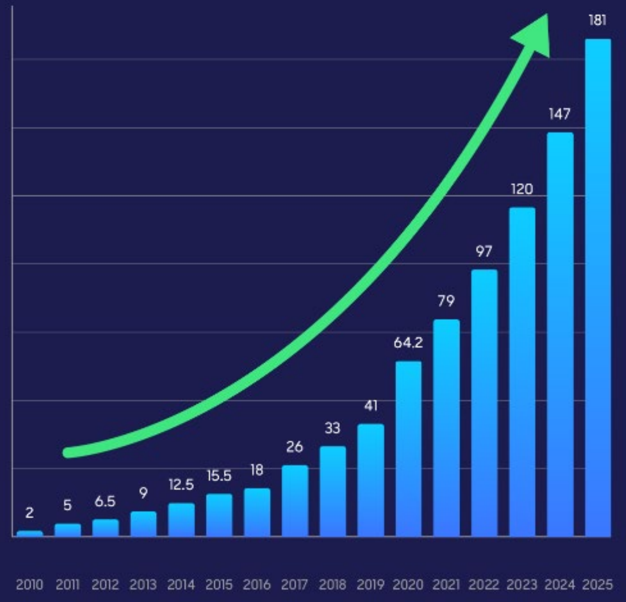
CPU Performance Growth is slowing down

50 Years of Microprocessor Trend Data



Original data up to the year 2010 collected and plotted by M. Horowitz, F, Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2021 by K. Rupp

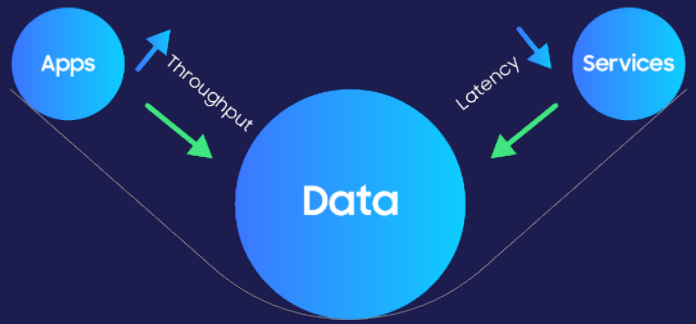
Volume of Data exponentially increases



Source: Statista

Data Gravity

Moving Compute closer to data source can address these issues

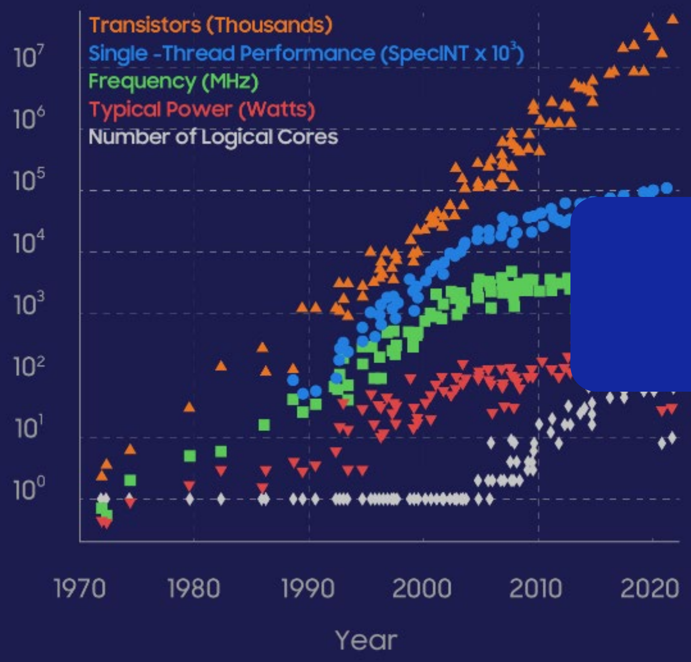


Source: Medium

Exponential Data Growth vs. Limited Processor Scaling

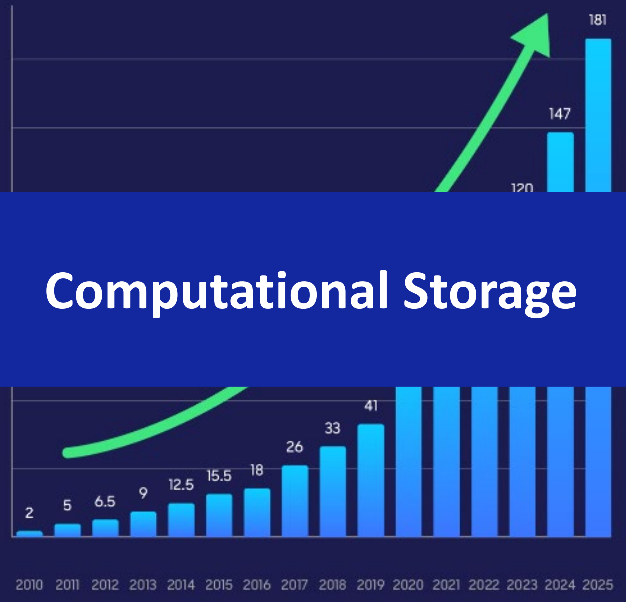
CPU Performance Growth is slowing down

50 Years of Microprocessor Trend Data



Original data up to the year 2010 collected and plotted by M. Horowitz, F, Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2021 by K. Rupp

Volume of Data exponentially increases

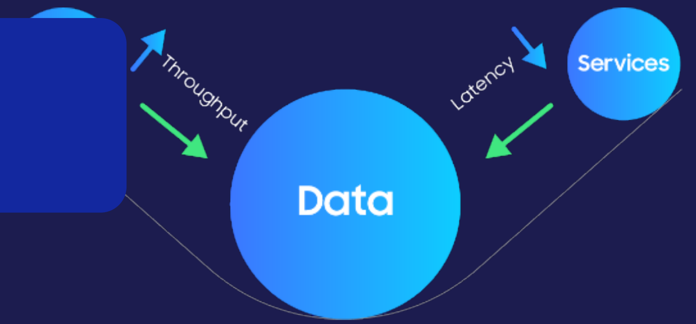


Source: Statista

Computational Storage

Data Gravity

Moving Compute closer to data source can address these issues



Source: Medium

STORAGE DEVELOPER CONFERENCE SDC 22

Computational Storage

- Computational Storage
  - CSD, CSA, CSP
- What is CSD (Computational Storage Drive)?
  - CSD = Persistent data storage + Computation
- Samsung SmartSSD®
  - SSD + HW acceleration engines
- Standard
  - NVMe computational storage (TP4091, TP4131)
  - SNIA
    - Computational storage architecture and programming model
    - Computational storage API

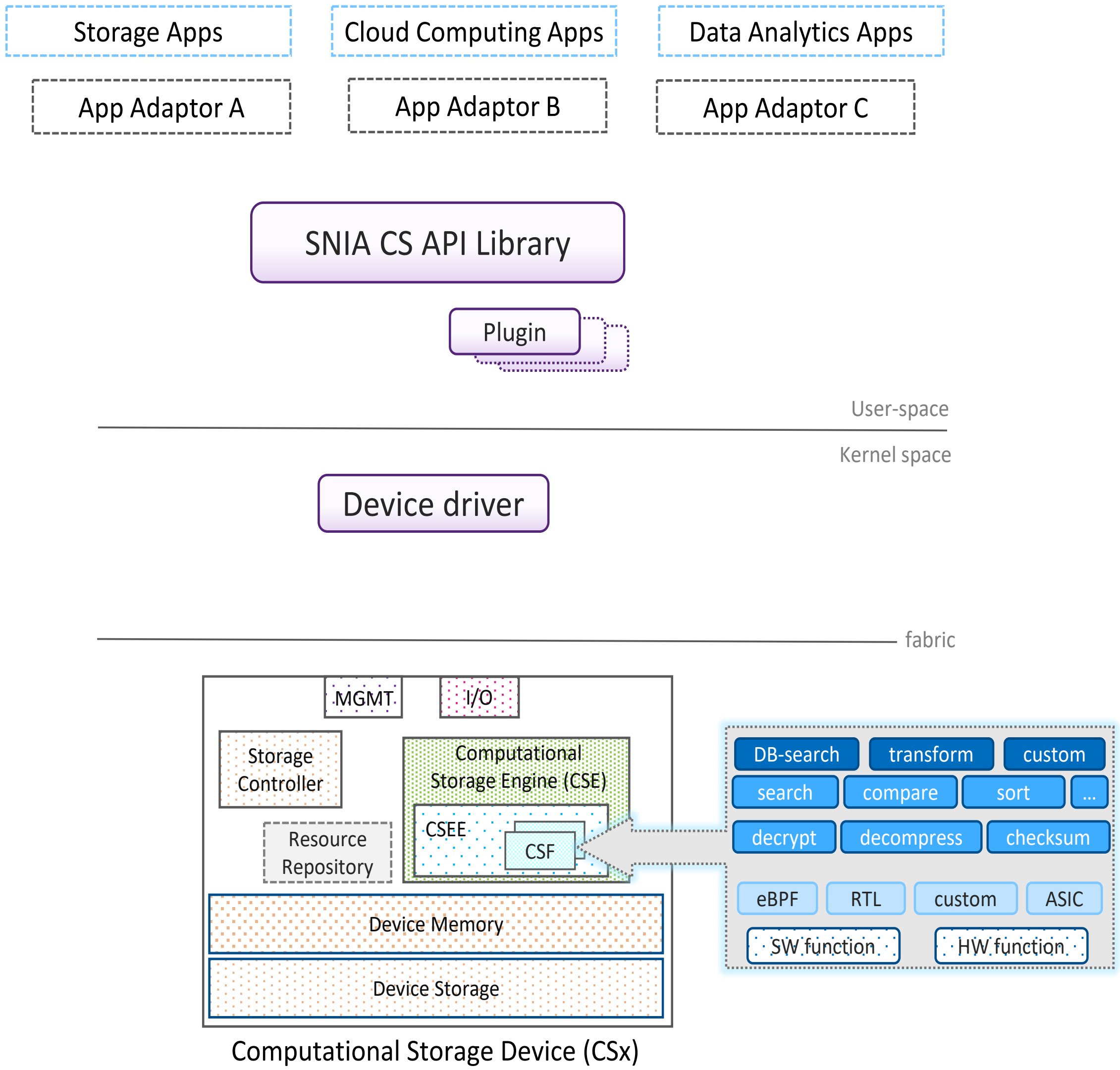
NVM Express ™

Computational Programs Command Set Specifications

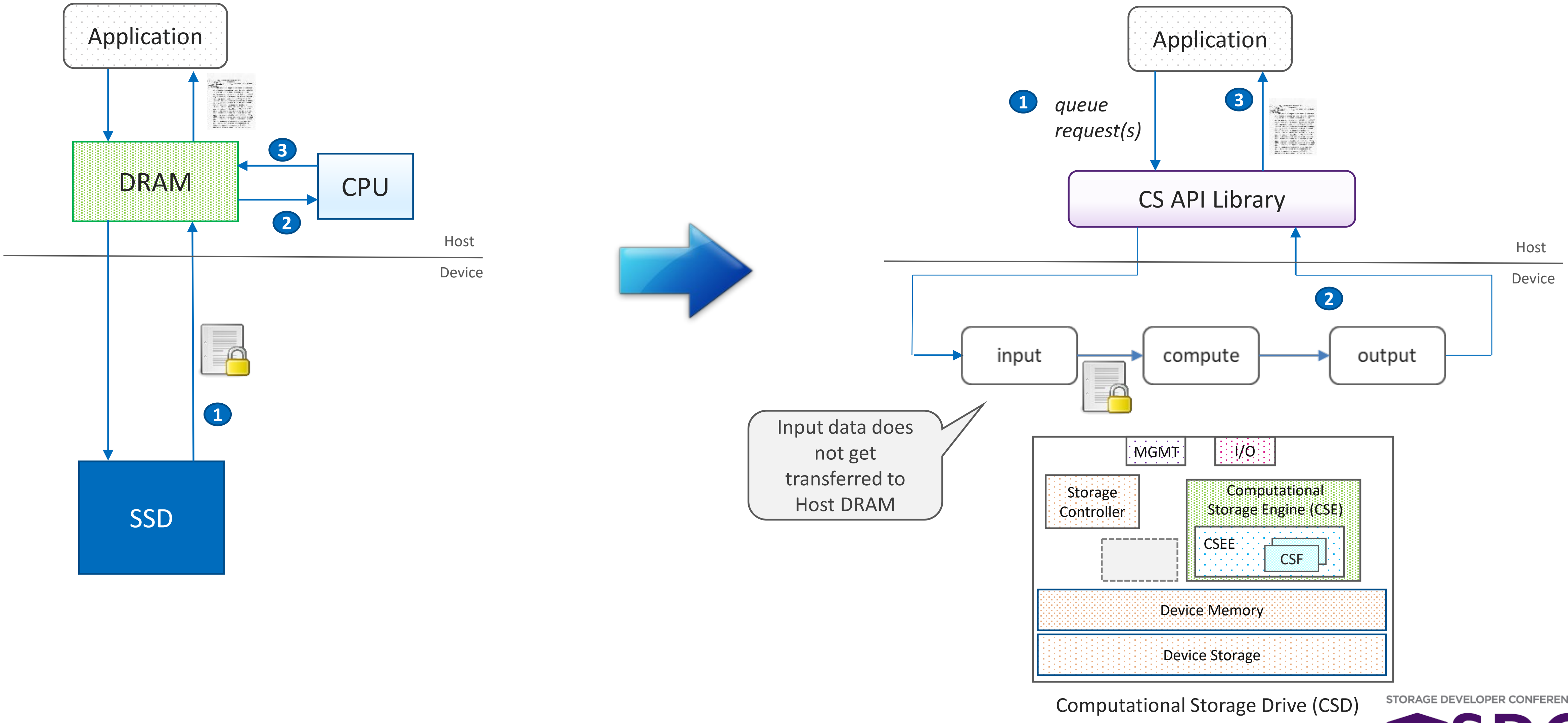
Computational Storage API v0.8 rev 0	Jul-2022
Computational Storage Architecture and Programming Model v0.9	Jun-2022

SNIA CSAPI Library

- Uniform interface for multiple configurations
  - APIs provided in common library
- Each CSx managed through its own device stack
  - Library may interface with additional plugins based on implementation requirements
  - Plugins help connect CSx to abstracted CS interfaces
- Extensible Interface
- CS APIs abstract
  - Discovery
  - Device Access
  - Device Memory (mapped/unmapped)
  - Near Storage Access
  - Copy Device Memory
  - Download CSFs
  - Execute CSFs
  - Device Management

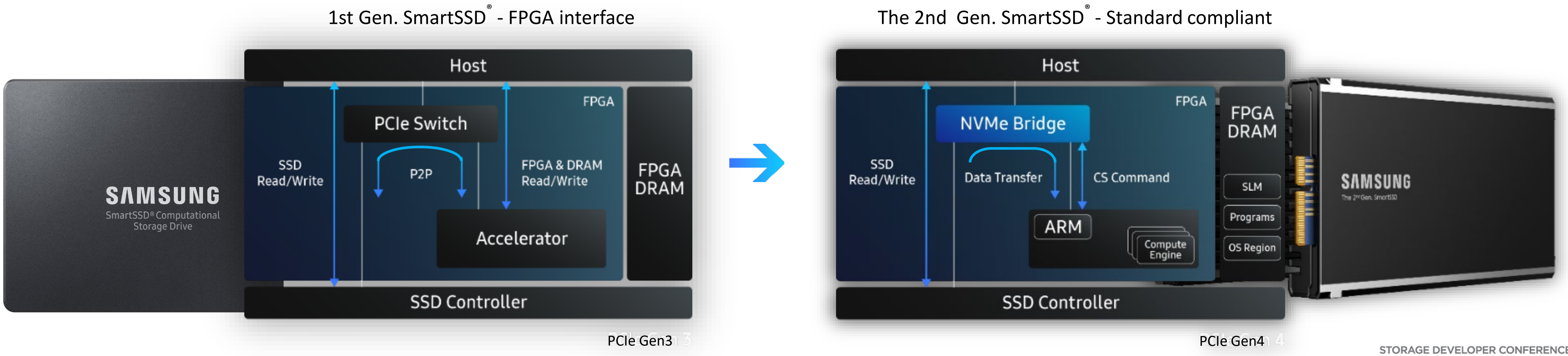


Applying Computational Storage



Samsung SmartSSD®

- SSD + HW acceleration engines
  - HW logic for data intensive operations (e.g., Image Resizer, Insert, Lookup, DB scan/filter, etc.)
  - At-Rest data processing
- The 1st Gen. SmartSSD® : FPGA interface based SmartSSD®
- The 2nd Gen. SmartSSD® : NVMe (TP4091) standard compliant SmartSSD®
  - Standard compliant eBPF for orchestration of offloaded SW + HW processing



* SLM: Subsystem Local Memory

Agenda

- Industry Trend
  - Exponential Data Growth vs. Limited Processor Scaling
- Computational Storage
  - SNIA CSAPI
  - Samsung SmartSSD
- RETINA: End-to-End (Compute+Storage) Framework for CS
  - Cross-Layered RETINA Key-Value Store
  - Dynamic Composable RETINA Computational Pipeline
- Summary
- Call for Action

Data Pipeline Today

Storage stack
(e.g., File System, Key-Value Store)

Application
(e.g., TensorFlow)

1. Indexing
Find out where the data is located

→

2. Retrieval
Fetch the data from the storage

→

3. Compute
Process the data

`open("cat002.jpg")`

`read("cat002.jpg")`

DNN Model Training

Database scan, filtering, and aggregation

TensorFlow

PyTorch

Pre-processing

```
jpg_decode(…)
resize(…)
crop(…)
mirror(…)
```

DNN model training

```
model_training(…)
```

How should the data pipeline be re-designed for Computational Storage?

RETINA: End-to-End (Storage+Compute) Framework for CS

RETINA = Key-Value Store + Computational Pipeline on CS

Application

1. Indexing
Find out where the data is located

→

2. Retrieval
Fetch the data from the storage

→

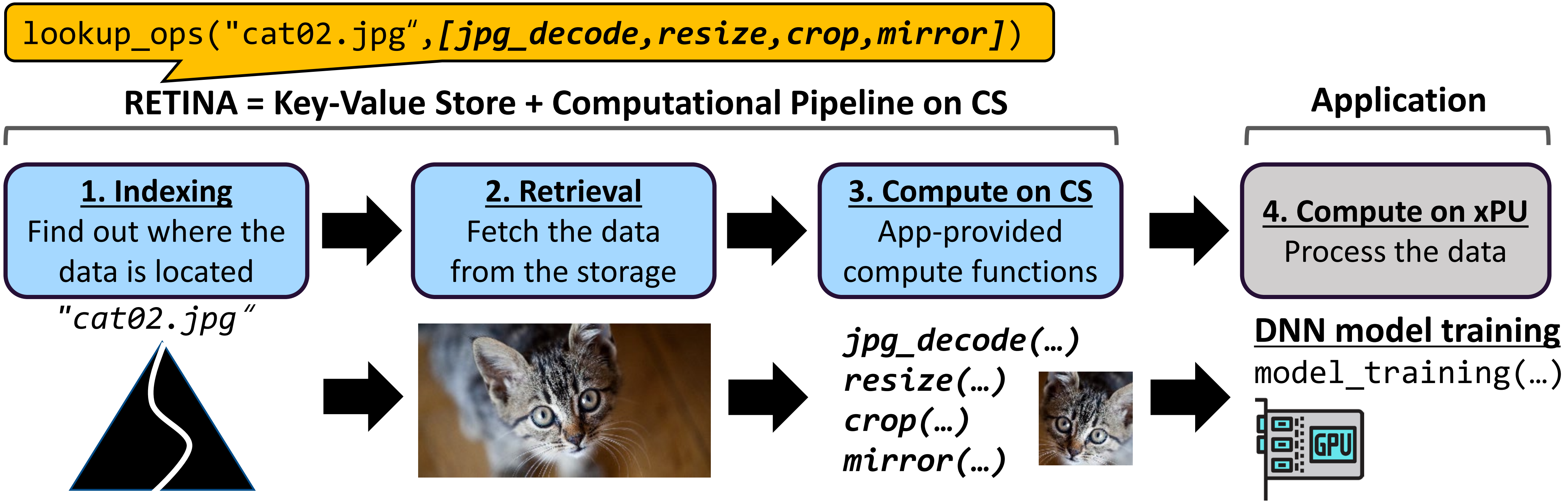
3. Compute on CS
App-provided compute functions

→

4. Compute on xPU
Process the data

- We propose **RETINA**, an end-to-end framework for Computational Storage
  - **Data management: Cross-Layered RETINA Key-Value Store**
    - Indexing, crash consistency, concurrency, etc.
  - **At-Rest data processing: Dynamically Composable Computational Pipeline**
    - Offload computation to Computational Storage
    - Chain compute functions as requested

RETINA: End-to-End (Storage+Compute) Framework for CS

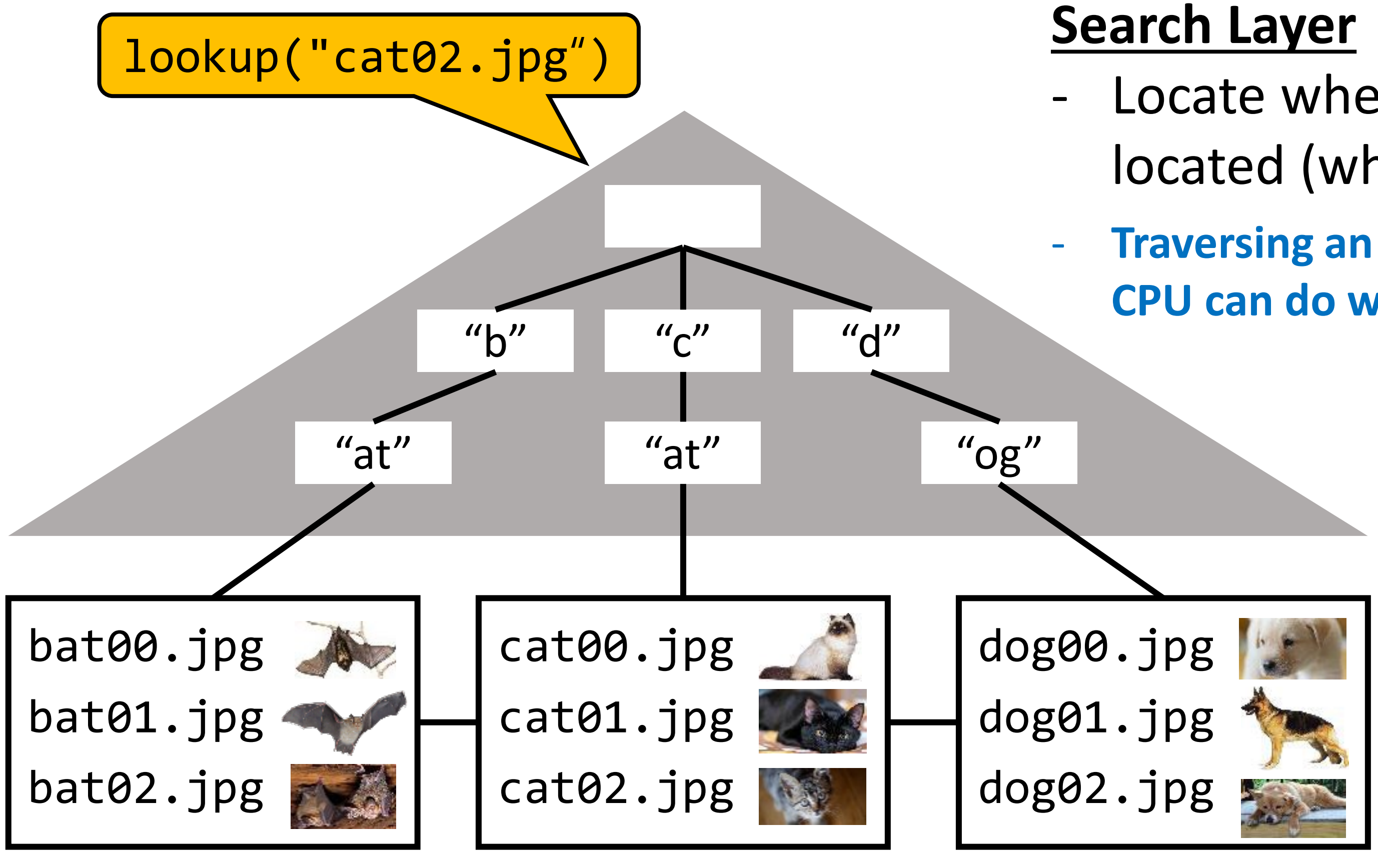


- **Advantages of RETINA approach**
  - Exploit the fast internal (peer-to-peer) bandwidth inside CSD (SmartSSD®)
  - Reduce data movement especially when the computed data is smaller
  - Allow sharing a CSD with multiple applications and tenants having different computations

Cross-Layered RETINA Key-Value Store

- **Let's exploit what host and CS can do best for each.**
  - Relieve CPU from data movement → reduce power and bandwidth consumption
  - Abstain CS accelerator (FPGA) from control plane operations (e.g., concurrency, OS interaction) → reduces the complexity
- **Use CPU as a control plane**
  - Communicate with FPGA and OS
  - Manage concurrency, caching, etc.
- **Use FPGA to perform compute at-rest**
  - Use high speed interconnect between FPGA and SSD
  - Offload CPU-intensive compute operations (data decoding, compression)
  - Reduces data movement by bringing only the end-user data

Why Cross-Layered Approach?



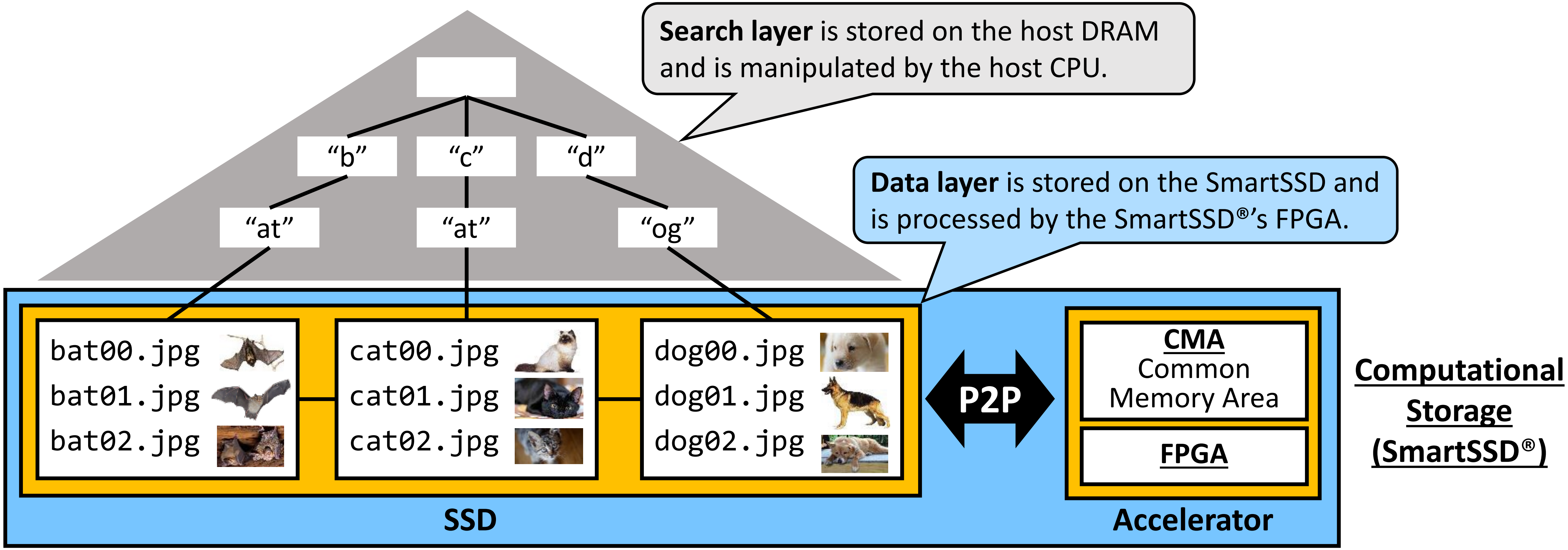
Search Layer

- Locate where the requested data is located (which leaf node)
- **Traversing an index is branch-divergent so CPU can do well.**

Data Layer

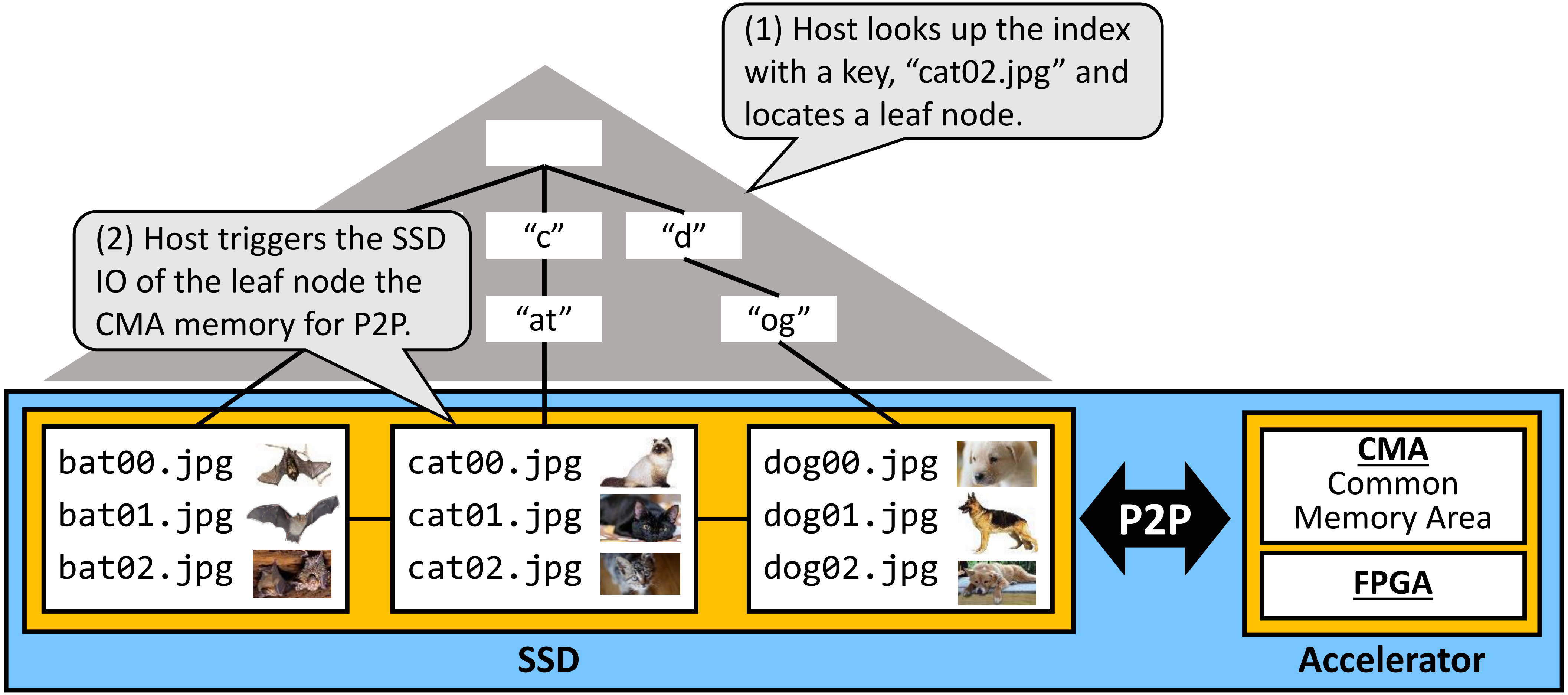
- Store actual data in leaf nodes, which is an array of key-value pairs
- **Requiring a large data movement so CS can do well.**

Cross-Layered RETINA Key-Value Store Architecture



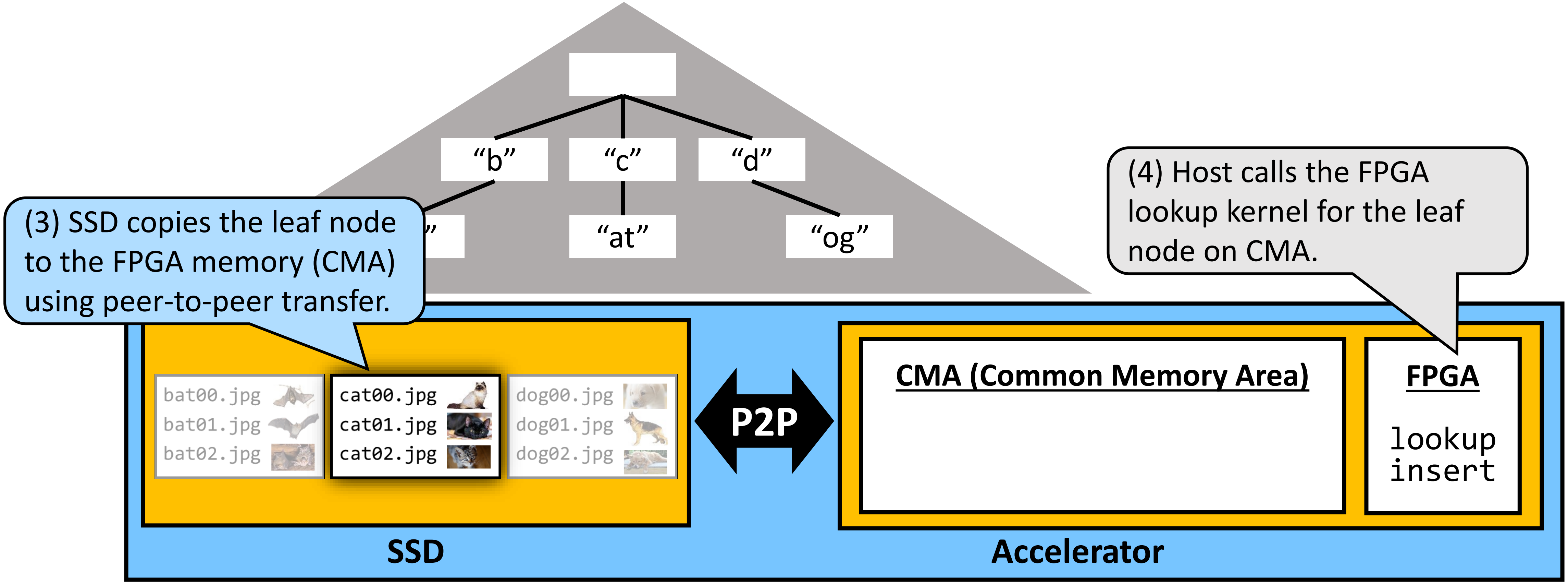
- **Host CPU as control plane:** triggering FPGA kernel call & SSD IO, concurrency, etc.
- **FPGA as at-rest data processing plane:** manipulating and handling data on SSD

RETINA Key-Value Store In Action: `lookup(cat02.jpg)`



 |

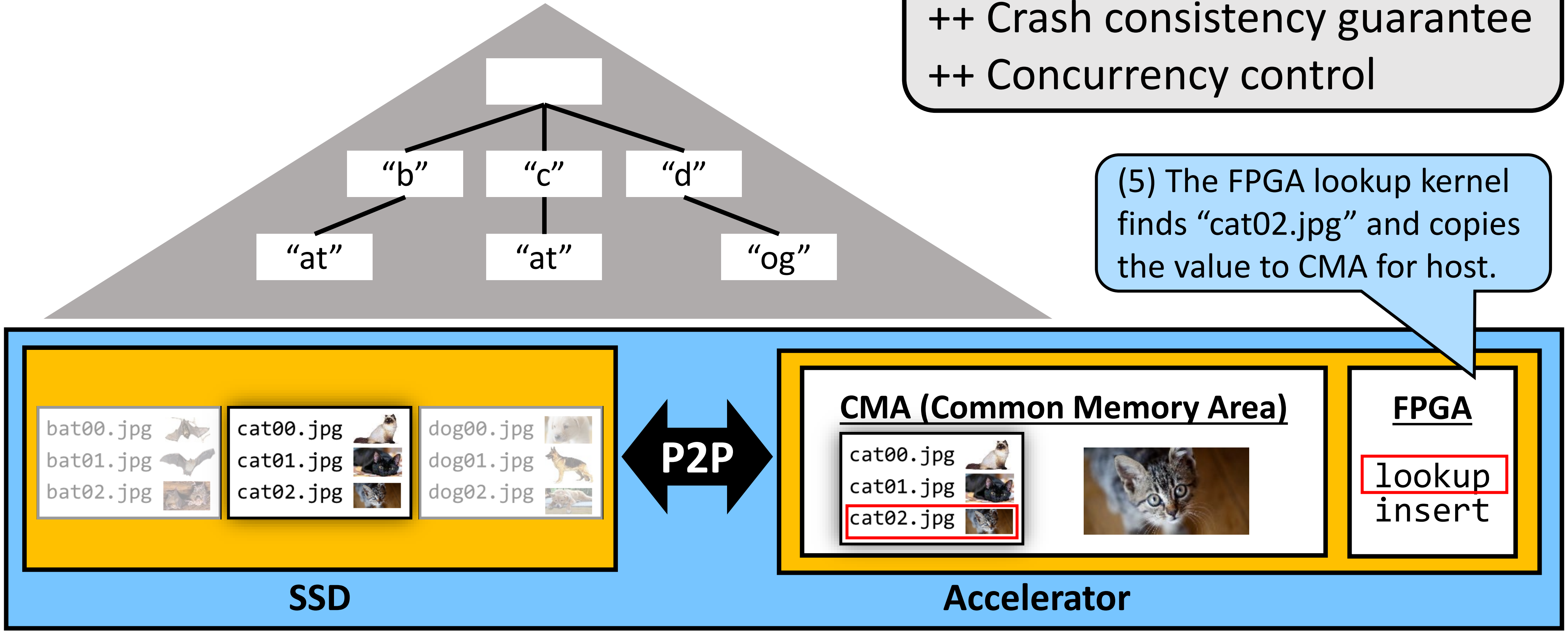
RETINA Key-Value Store In Action: `lookup(cat02.jpg)`



RETINA Key-Value Store In Action: `lookup(cat02.jpg)`

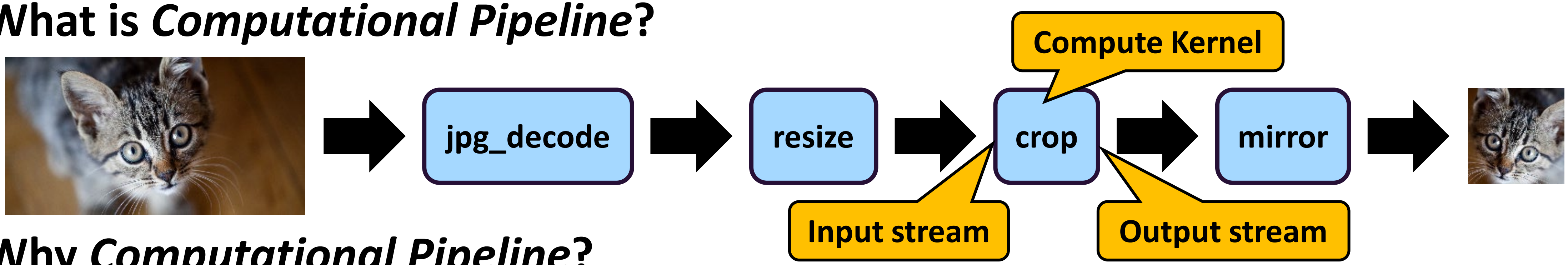
++ Cross-layered cache design
++ Crash consistency guarantee
++ Concurrency control

(5) The FPGA lookup kernel finds “cat02.jpg” and copies the value to CMA for host.



Dynamically Composable RETINA Computational Pipeline

- **What is *Computational Pipeline*?**

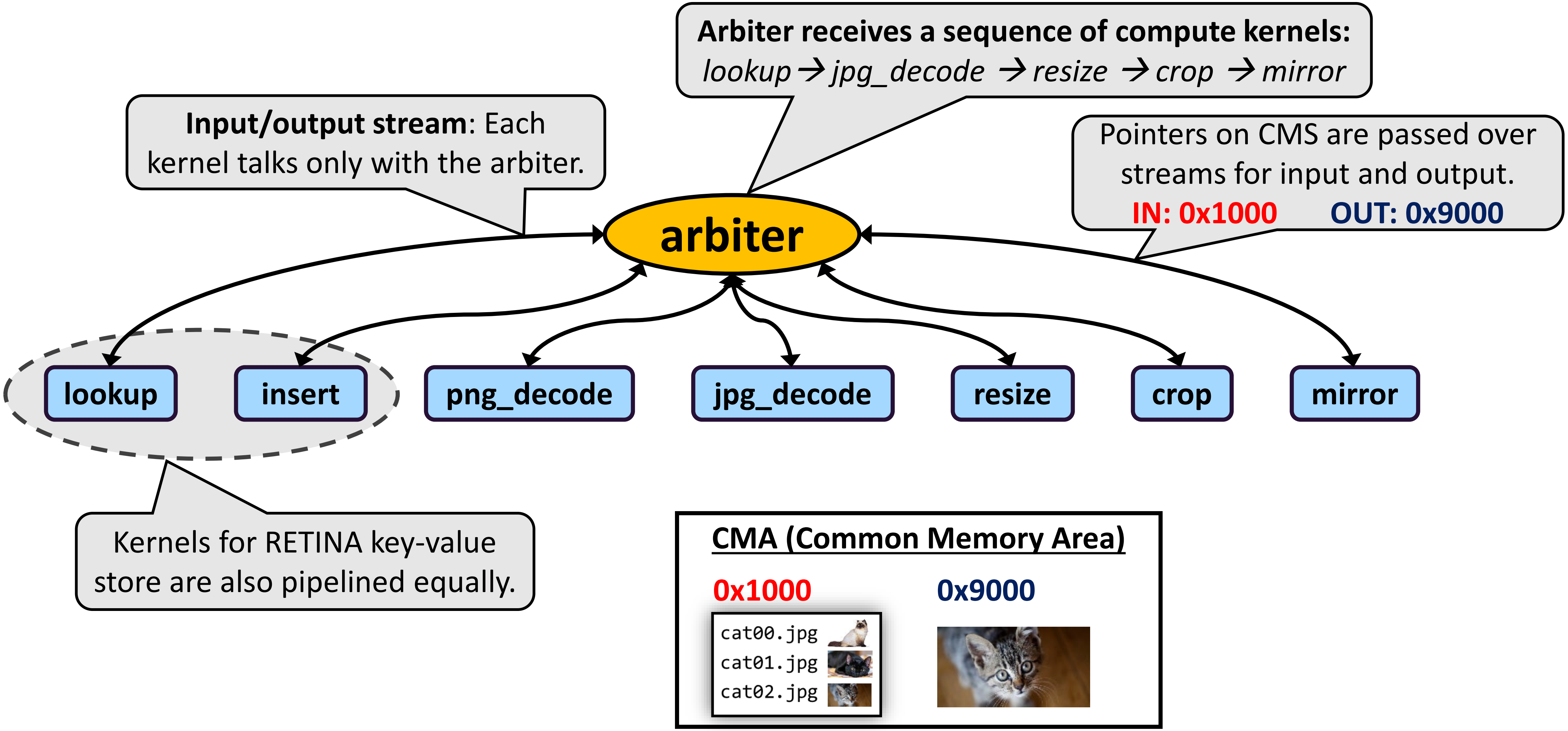


- **Why *Computational Pipeline*?**
  - Simple and well-defined interface ➔ only interact with input and output streams
  - Easy to integrate different types of kernels as long as following the input/output streams
  - Naturally exploit the pipeline parallelism for accelerator
- **Why *Dynamically Composing Kernels* is important?**
  - Hardwired pipelines are not generic enough
  - Applications may require different compute kernels or different orders of compute kernels
  - A compute kernel can be re-used in multiple applications
  - SmartSSD® are shared by multiple applications and tenants

How to Achieve Dynamic Composability?

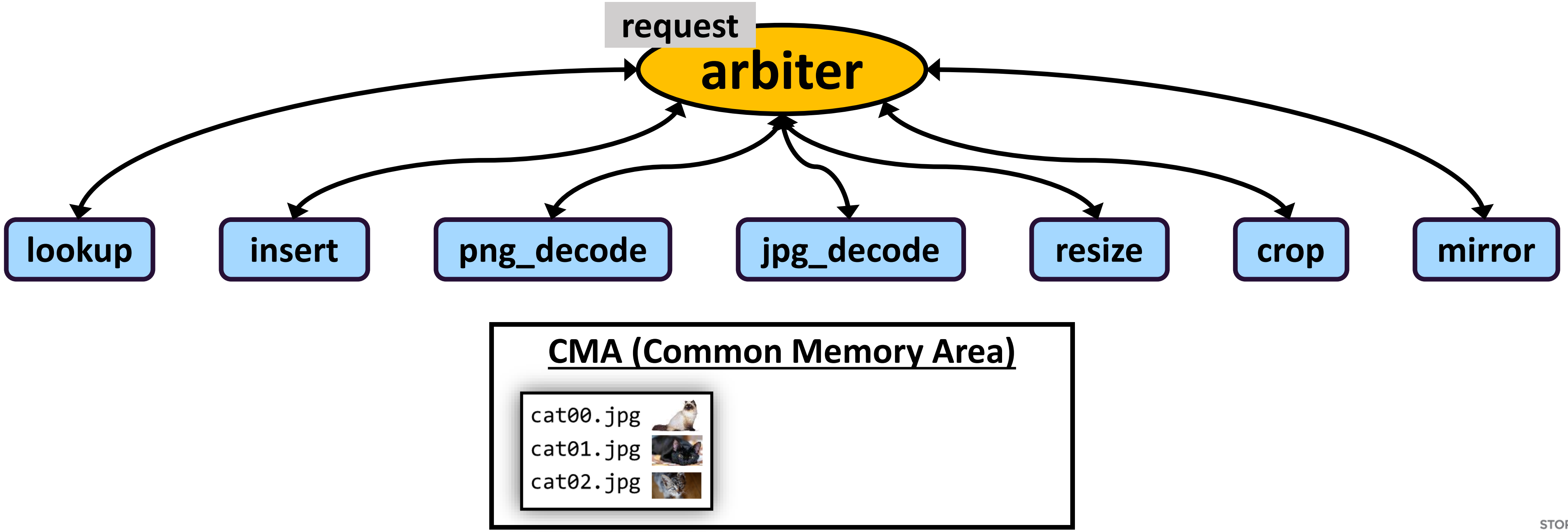
- **Goal:** dynamically compose compute kernels as per user request on the fly
- ***Central pipeline manager (arbiter) based approach***
- **Compute kernels**
  - Already installed on FPGA
  - Communicate via streams
  - Don’t communicate with each other directly like in the hardwired pipeline architecture
- **Central pipeline manager (arbiter)**
  - Instead, kernels communicate from/to arbiter using streams which manages the IO forwarding order
  - Arbiter performs scheduling of compute kernels and manages FPGA memory

RETINA Computational Pipeline Architecture



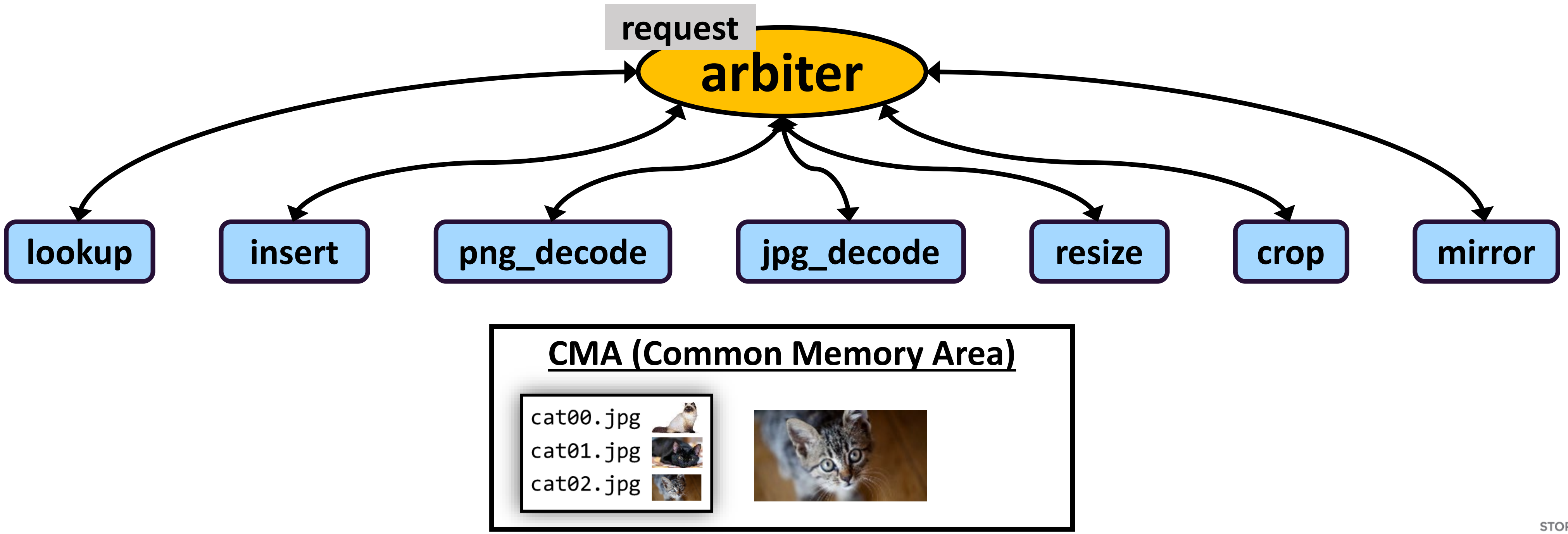
RETINA in Action: `lookup_ops("cat02.jpg",[jpg_decode,resize,crop,mirror])`

- Host CPU looks up the search layer and performs SSD IO for the lead node.
- Host CPU sends a request to arbiter with lookup→jpg_decode→resize→crop→mirror.



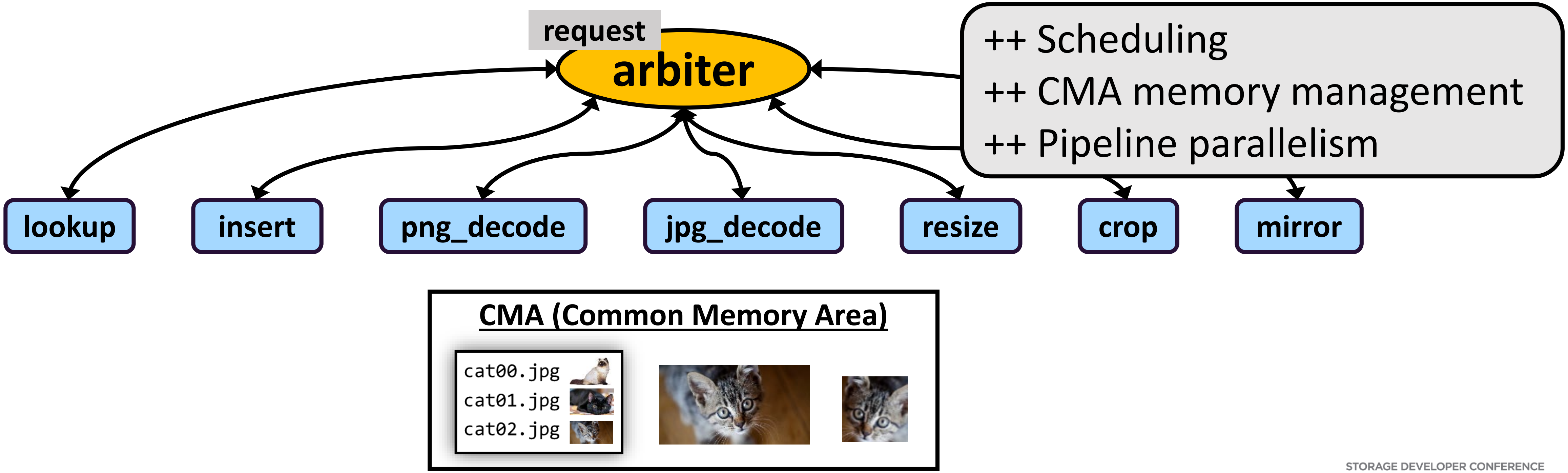
RETINA in Action: `lookup_ops("cat02.jpg",[jpg_decode,resize,crop,mirror])`

- Host CPU looks up the search layer and performs SSD IO for the lead node.
- Host CPU sends a request to arbiter with lookup→jpg_decode→resize→crop→mirror.
- Arbiter forwards input & output as per the requested kernel order.



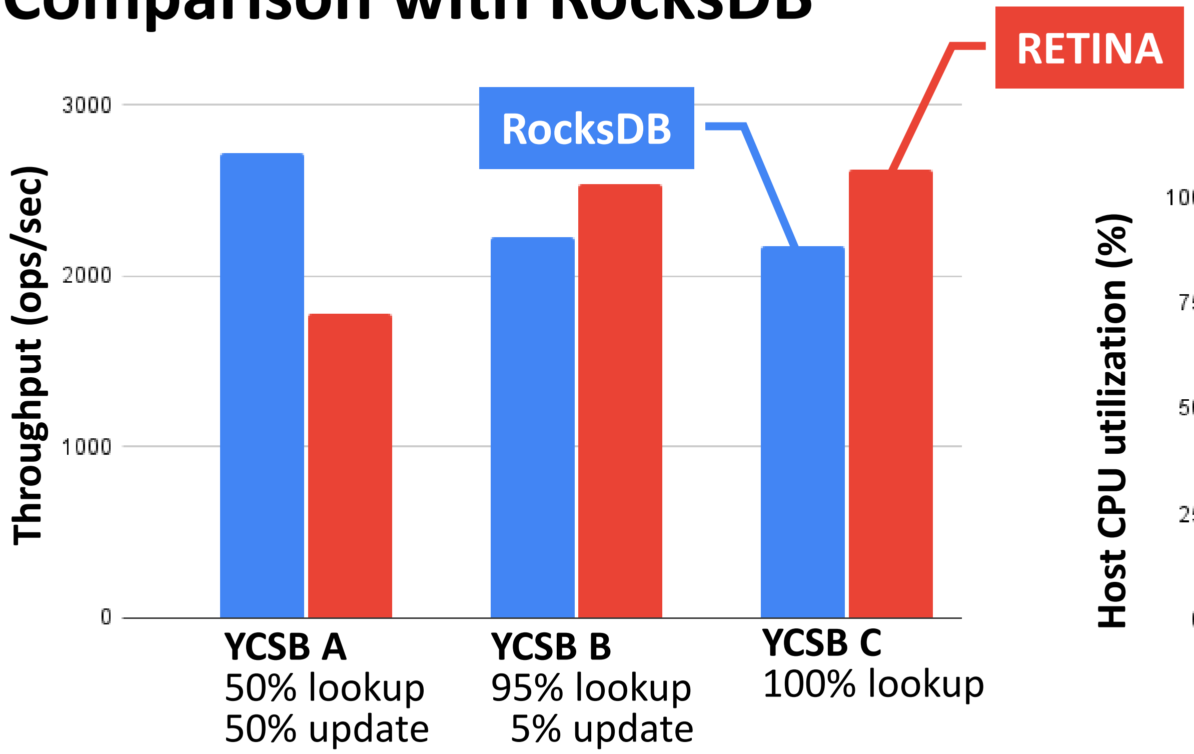
RETINA in Action: `lookup_ops("cat02.jpg",[jpg_decode,resize,crop,mirror])`

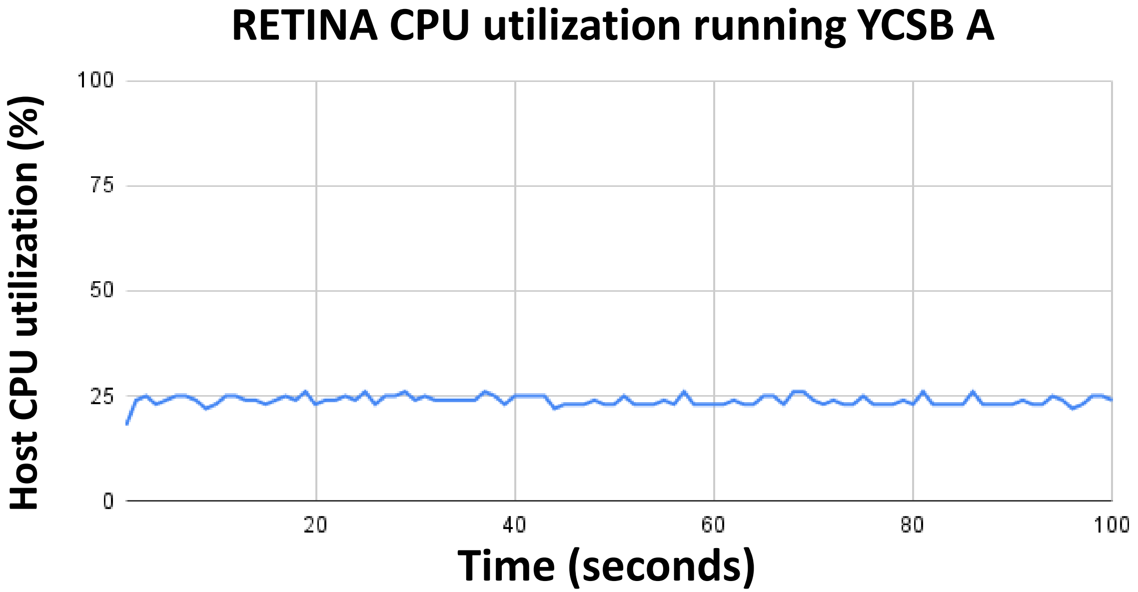
- Host CPU looks up the search layer and performs SSD IO for the lead node.
- Host CPU sends a request to arbiter with lookup→jpg_decode→resize→crop→mirror.
- Arbiter forwards input & output as per the requested kernel order.



RETINA Performance Evaluation

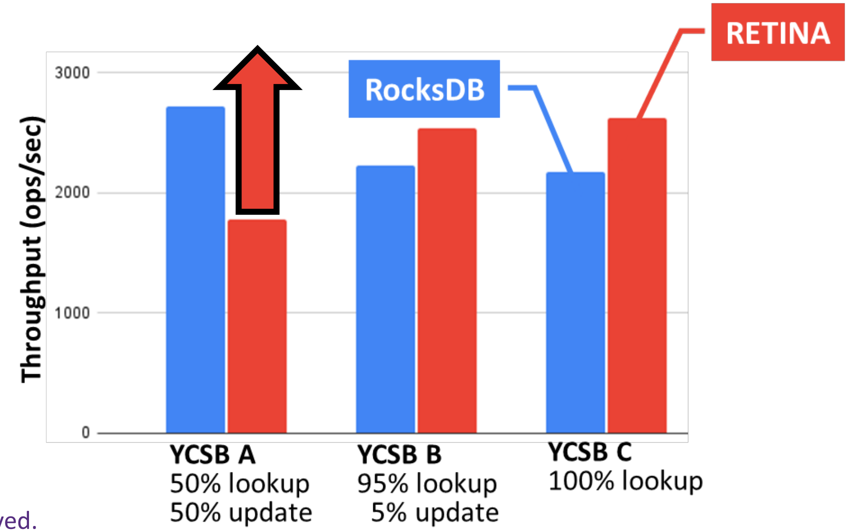
- **Ran the popular YCSB key-value store workload**
  - The baseline performance without compute kernels (the worst-case)
- **Comparison with RocksDB**





Evolving RETINA for Upcoming SNIA CS API Architecture

- The early version of RETINA is implemented with OpenCL and tested on the 1st Gen. SmartSSD®
- Using SNIA CS API makes RETINA more performance efficient
  - Further reduces the CPU utilization
  - Rewriting using CS API is in progress
- RETINA can also benefit from the 2nd Gen. SmartSSD®
  - Further reduces the host intervention, reducing the overall latency



Summary

- Computational Storage is needed to tackle the challenges of exponential data growth and limited processor scaling.
  - Computational Storage API is standardized.
  - Samsung SmartSSD® are available now (Gen 1 and Gen 2).
- RETINA is a end-to-end framework for Computational Storage
  - Cross-layered key-value store for data management
  - Dynamically Composable computational pipeline as a generic At-Rest data processing framework
- We expect more performance boost of RETINA with SNIA CS API and 2nd Gen SmartSSD®.

Call for Action

- Other sessions on Computational Storage
  - Samsung Keynote – Yang Seok Ki
  - Computational Storage APIs – Oscar Pinto
  - Green Computing with Computational Storage Devices – Changho Choi, Yangwook
  - Accelerating Near Real-time Analytics with High Performance Object Storage – Nithya, Mayank
  - Multiple sessions from SNIA CS TWG and NVMe CS
- Help build the ecosystem
- Join the standardization efforts
  - SNIA, NVMe

Please take a moment to rate this session.

Your feedback is important to us.

STORAGE DEVELOPER CONFERENCE

SDC 22

