PERSISTENT MEMORY + SUMMIT 2022 COMPUTATIONAL STORAGE

VIRTUAL EVENT • MAY 24-25, 2022

Programming with Computational Storage

Oscar P Pinto, Principal Engineer

Samsung Semiconductor Inc.

Agenda

- Overview
- Computational Storage
- SNIA and CS APIs
- Working with an Example
- Mapping APIs to Device
- Summary

Adopting Computational Storage

- Data is being created at a exponential rate
- Storage has also grown to account for this growth

- NVMe SSDs provide better performance than ever before
  - But their bandwidth not fully utilized by Host
- General purpose CPUs not able to fully tap this bandwidth
  - Scaling limited by PCIe lanes
- SSDs have more internal bandwidth than utilized

- Fabrics overloaded with transferring data for processing and results
  - What if data is processed where it resides, near storage?

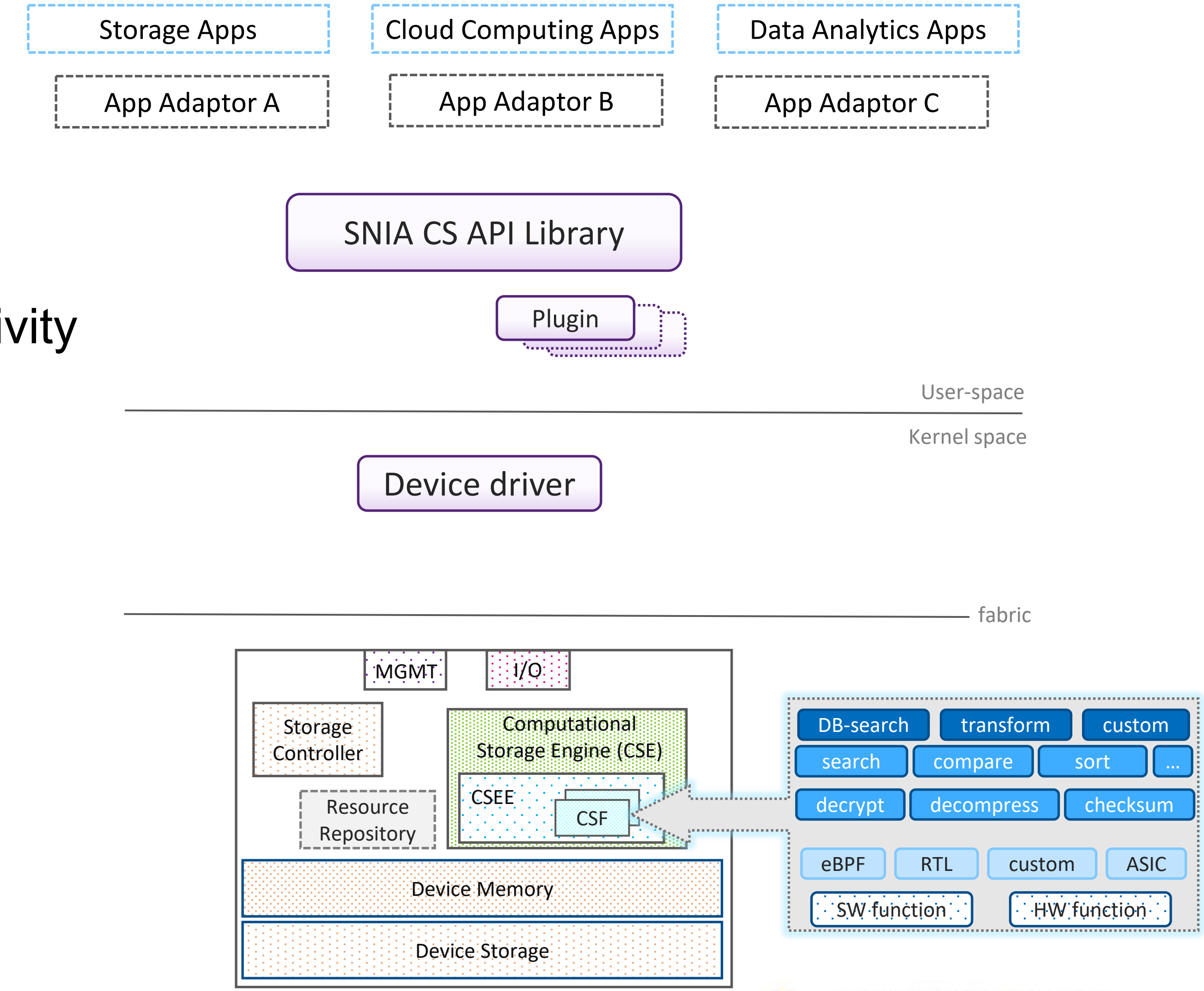
- Computational Storage & Offloads tap into this
  - Process data near storage
  - Add compute to storage

SNIA CS API Library

About the Library

SNIA: Computational Storage APIs

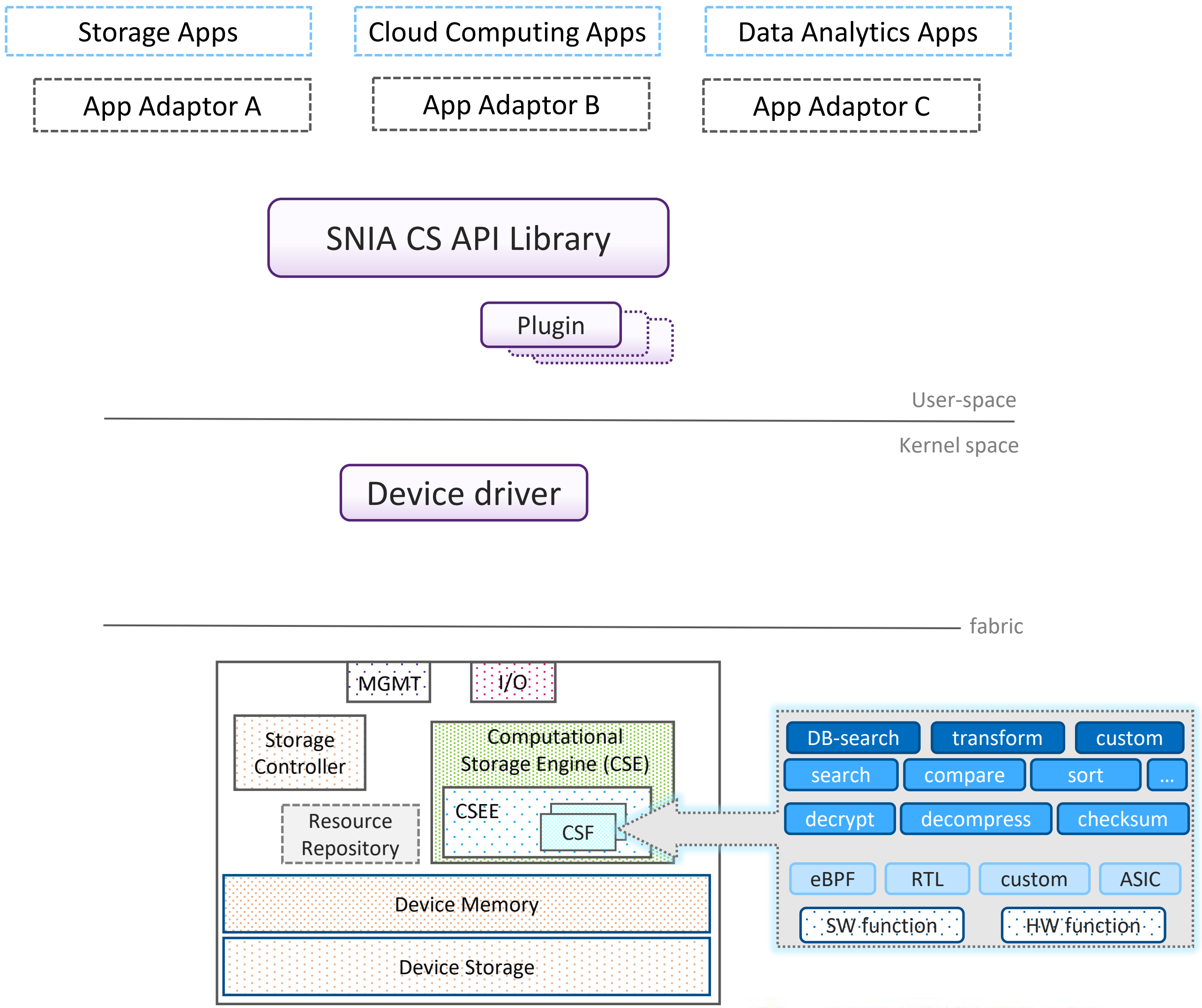
- One Set of APIs across all CSx types
  - CSP, CSD, CSA
  - Common set of APIs for different CS devices
- One interface to different device and connectivity choices
  - Hardware ASIC, CPU, FPGA, etc
  - NVMe/NVMe-oF, PCIe, custom, etc
- Configurations may be local/remote attached
- Hides vendor specific implementation details below library
- Abstracts device specific details
- APIs to be OS agnostic



Computational Storage Drive (CSD)

SNIA: CS API Overview

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



Computational Storage Drive (CSD)

Key APIs of Interest

Functionality	API	Details
Discovery		
	csQueryCSxList()	• Discover available Computational Storage Devices (CSxes)
	csGetCSxFromPath()	• Identify CSx associated with storage path
	csQueryCSFList()	• Discover available Computational Storage Functions (CSFs) in given storage path
Access		
	csOpenCSx()	• Access a CSx
	csCloseCSx()	• Release access to previously opened CSx
Memory		
	csAllocMem()	• Allocate memory for CSF usage
	csFreeMem()	• Free previously allocated memory
Storage		
	csQueueStorageRequest()	• Issue a read/write request to transfer data between storage and device memory
Copy		
	csQueueCopyMemRequest()	• Transfer data between device memory and host memory
Compute		
	csGetCSFId()	• Get access to a CSF to execute
	csQueueComputeRequest()	• Schedule a CSF to execute work on device
Management		
	csQueryDeviceProperties()	• Query device resources
	csConfig()	• Configure device resource
	csDownload()	• Download a CSF to device

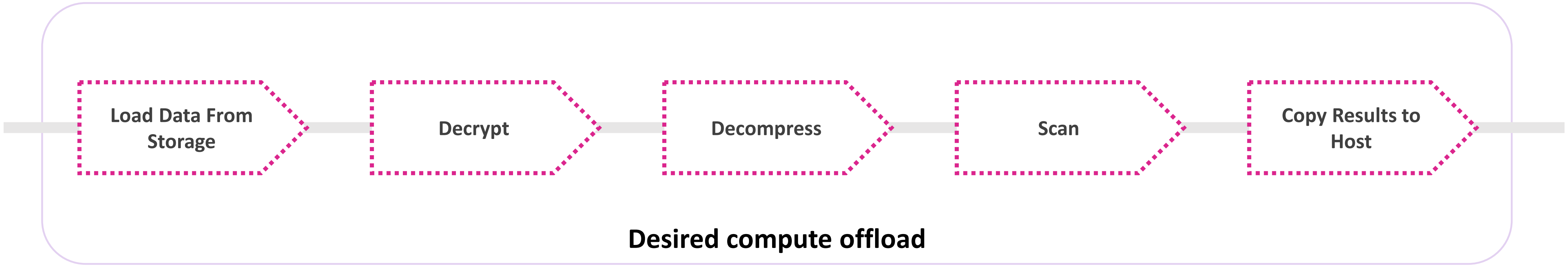
APIs by Example

A step-by-step guide

Example: Find Specific Data

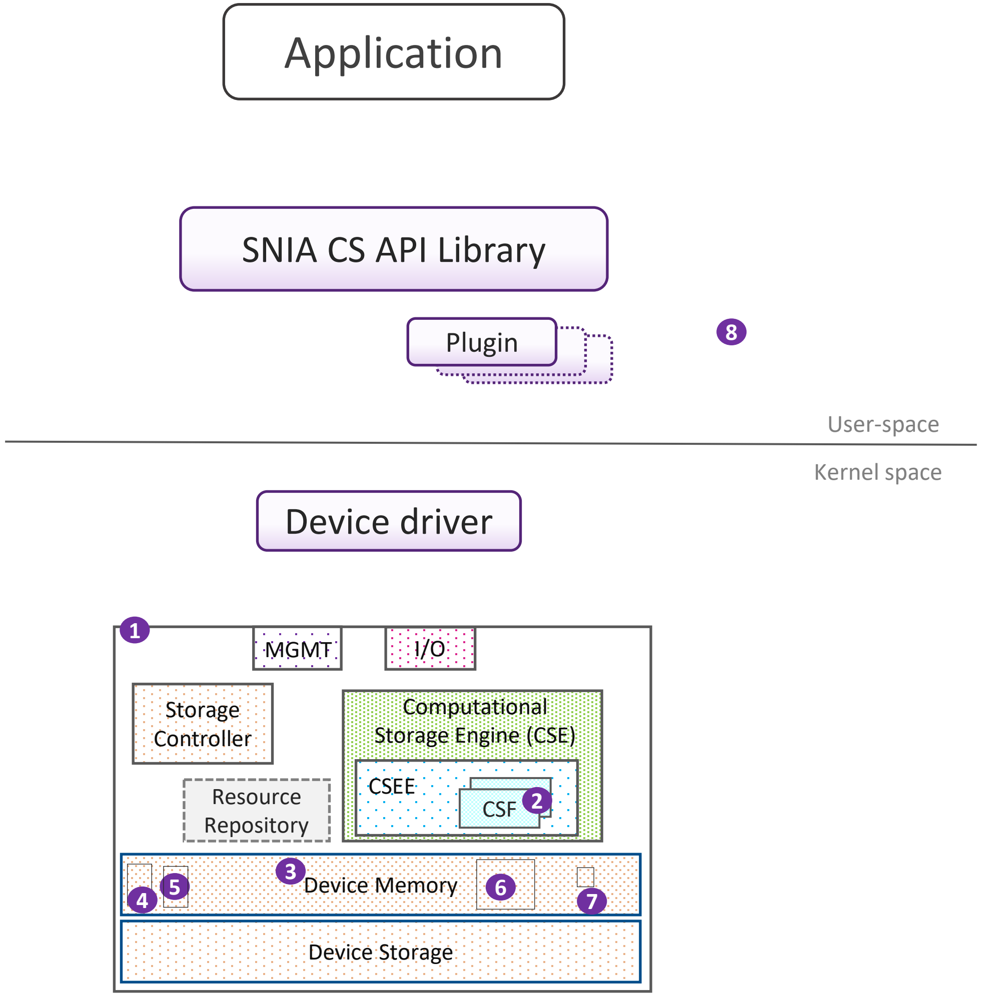
Price Sold	< $800,000
Bedrooms	
Baths	
Single Family	
City	
Zipcode	

Search Criteria



Example: Find Specific Data - Steps

1. Discover CSx & Access
2. Find CSFs
3. Allocate Device Memory
4. Load Storage data in Device Memory
5. Decrypt Data
6. Decompress Data
7. Run Scan Filter
8. Copy Results



Example: Discovery

1. Discover CSx & Access it
   a. Discover your Computational Storage Device (CSx)
   b. Get access to CSx

```
// discover my CS device (CSx)
length = sizeof(csxBuffer);
status = csGetCSxFromPath(file_path, &length, &csxBuffer);
// gain access
status = csOpenCSx(csxBuffer, &MyDevContext, &devHandle);
```

2. Discover Functions in CSx

```
// discover CSFs using csGetCSFId API
status = csGetCSFId(devHandle, “decrypt”, &infoLength. &csfInfo);
decryptId = buffer.CSFId;
...
// download CSFs if required
status = csDownload(devHandle, &programInfo);
```

```
typedef struct {
    CS_CSF_ID CSFId;           // unique CSF Identifier
    u8 RelativePerformance;    // values [1-10]; higher is better
    u8 RelativePower;          // values [1-10]; lower is better
    u8 Count;                  // number of CSF instances available
} CSFIdInfo;
```

**API return status values are not shown to check for success and errors to ease readability*

 |

Example: Allocate Device Memory

3. Allocate Device Memory
   - Allocate memory for all required buffers
     - Buffer1 - load data from storage
     - Buffer2 – hold decrypted data from Buffer1
     - Buffer3 – hold decompressed data from Buffer2
     - Buffer4 – collect results of search

```
// allocate device memory for input and output buffers
status = csAllocMem(devHandle, CHUNK_SIZE, 0, &inputMemHandle, NULL);
status = csAllocMem(devHandle, CHUNK_SIZE, 0, &decryptMemHandle, NULL);
status = csAllocMem(devHandle, MAX_CHUNK_SIZE, 0, &decompMemHandle, NULL);
status = csAllocMem(devHandle, CHUNK_SIZE, 0, &resultsMemHandle, NULL);
```


Example: Load Storage Data

4. Load Storage Data directly in Device Memory

```
// allocate storage request & read chunk size data from file handle fd
storReq = calloc(1, sizeof(CsStorageRequest));
if (!storReq) { ERROR_OUT("memory alloc error\n"); }
storReq->Mode = CS_STORAGE_FILE_IO;
storReq->DevHandle = devHandle;
storReq->u.CsFileIo.Type = CS_STORAGE_LOAD_TYPE;
storReq->u.CsFileIo.FileHandle = fd;
storReq->u.CsFileIo.Offset = 0;
storReq->u.CsFileIo.Bytes = CHUNK_SIZE;
storReq->u.CsFileIo.DevMem.MemHandle = inputMemHandle;
storReq->u.CsFileIo.DevMem.ByteOffset = 0;
status = csQueueStorageRequest(storReq, storReq, NULL, NULL, NULL);
```

Example: Decrypt Data

5. Decrypt Storage Data Loaded in Device Memory

- Run Decrypt CSF in device

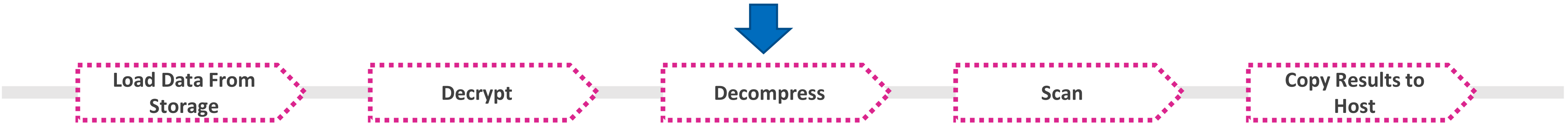
```
// allocate compute request for 3 args & issue compute request
compReq = calloc(1, sizeof(CsComputeRequest) + (sizeof(CsComputeArg) * 3));
if (!compReq) { ERROR_OUT("memory alloc error\n"); }
compReq->DevHandle = devHandle;
compReq->FunctionId = decryptId;
compReq->NumArgs = 3;
argPtr = &compReq->Args[0];
csHelperSetComputeArg(&argPtr[0], CS_AFDM_TYPE, inputMemHandle, 0);
csHelperSetComputeArg(&argPtr[1], CS_32BIT_VALUE_TYPE, CHUNK_SIZE);
csHelperSetComputeArg(&argPtr[2], CS_AFDM_TYPE, decryptMemHandle, 0);
status = csQueueComputeRequest(compReq, NULL, NULL, NULL, NULL);
```

Example: Decompress Data

6. Decompress the Decrypted Data in Device Memory

- Run Decompress CSF in device

```
// allocate compute request for 3 args & issue compute request
compReq = calloc(1, sizeof(CsComputeRequest) + (sizeof(CsComputeArg) * 3));
if (!compReq) { ERROR_OUT("memory alloc error\n"); }
compReq->DevHandle = devHandle;
compReq->FunctionId = decompId;
compReq->NumArgs = 3;
argPtr = &compReq->Args[0];
csHelperSetComputeArg(&argPtr[0], CS_AFDM_TYPE, decryptMemHandle, 0);
csHelperSetComputeArg(&argPtr[1], CS_32BIT_VALUE_TYPE, CHUNK_SIZE);
csHelperSetComputeArg(&argPtr[2], CS_AFDM_TYPE, decompMemHandle, 0);
status = csQueueComputeRequest(compReq, NULL, NULL, NULL, NULL);
```



Example: Scan Data

7. Scan the Decompressed Data for Records
   - Run Scan Query Filter in device

```
// allocate compute request for 3 args & issue compute request
compReq = calloc(1, sizeof(CsComputeRequest) + (sizeof(CsComputeArg) * 3));
if (!compReq) { ERROR_OUT("memory alloc error\n"); }
compReq->DevHandle = devHandle;
compReq->FunctionId = ScanId;
compReq->NumArgs = 3;
argPtr = &compReq->Args[0];
csHelperSetComputeArg(&argPtr[0], CS_AFDM_TYPE, decompMemHandle, 0);
csHelperSetComputeArg(&argPtr[1], CS_32BIT_VALUE_TYPE, MAX_CHUNK_SIZE);
csHelperSetComputeArg(&argPtr[2], CS_AFDM_TYPE, resultsMemHandle, 0);
status = csQueueComputeRequest(compReq, NULL, NULL, NULL, NULL);
```

Example: Copy Results

8. Copy Output Results to Host

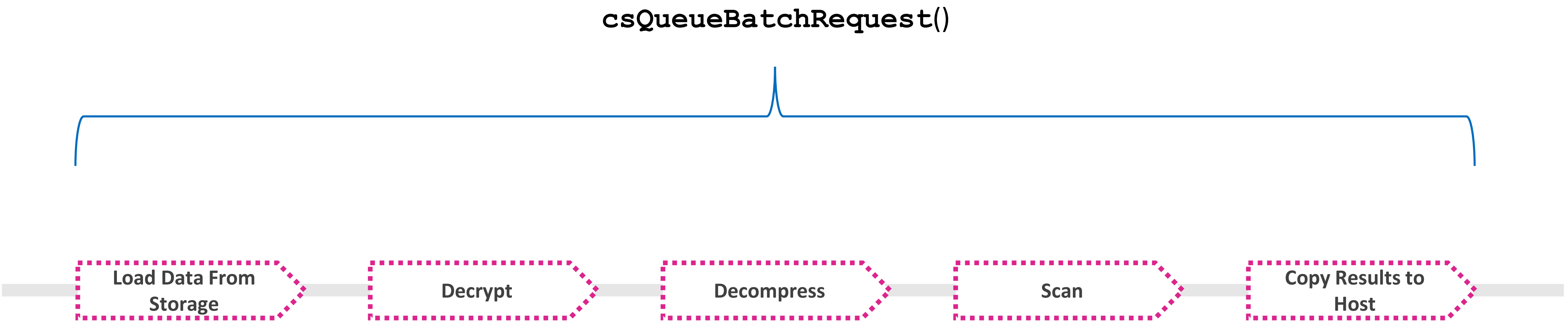
- Copy Device Memory Contents to Host

```
// allocate copy request & copy results to host buffer
copyReq = calloc(1, sizeof(CsCopyMemRequest));
if (!copyReq) { ERROR_OUT("memory alloc error\n"); }
copyReq->Type = CS_COPY_FROM_DEVICE;
copyReq->HostVAddress = results_buf;
copyReq->DevMem.MemHandle = resultsMemHandle;
copyReq->DevMem.ByteOffset = 0;
copyReq->Bytes = CHUNK_SIZE;
status = csQueueCopyMemRequest(copyReq, NULL, NULL, NULL, NULL);
```

PERSISTENT MEMORY + SUMMIT 2022 COMPUTATIONAL STORAGE

The Batch Request

- Create one Batch request that includes other requests in one job
  - Optimization for recurring jobs
  - Submit request and get notified on Results



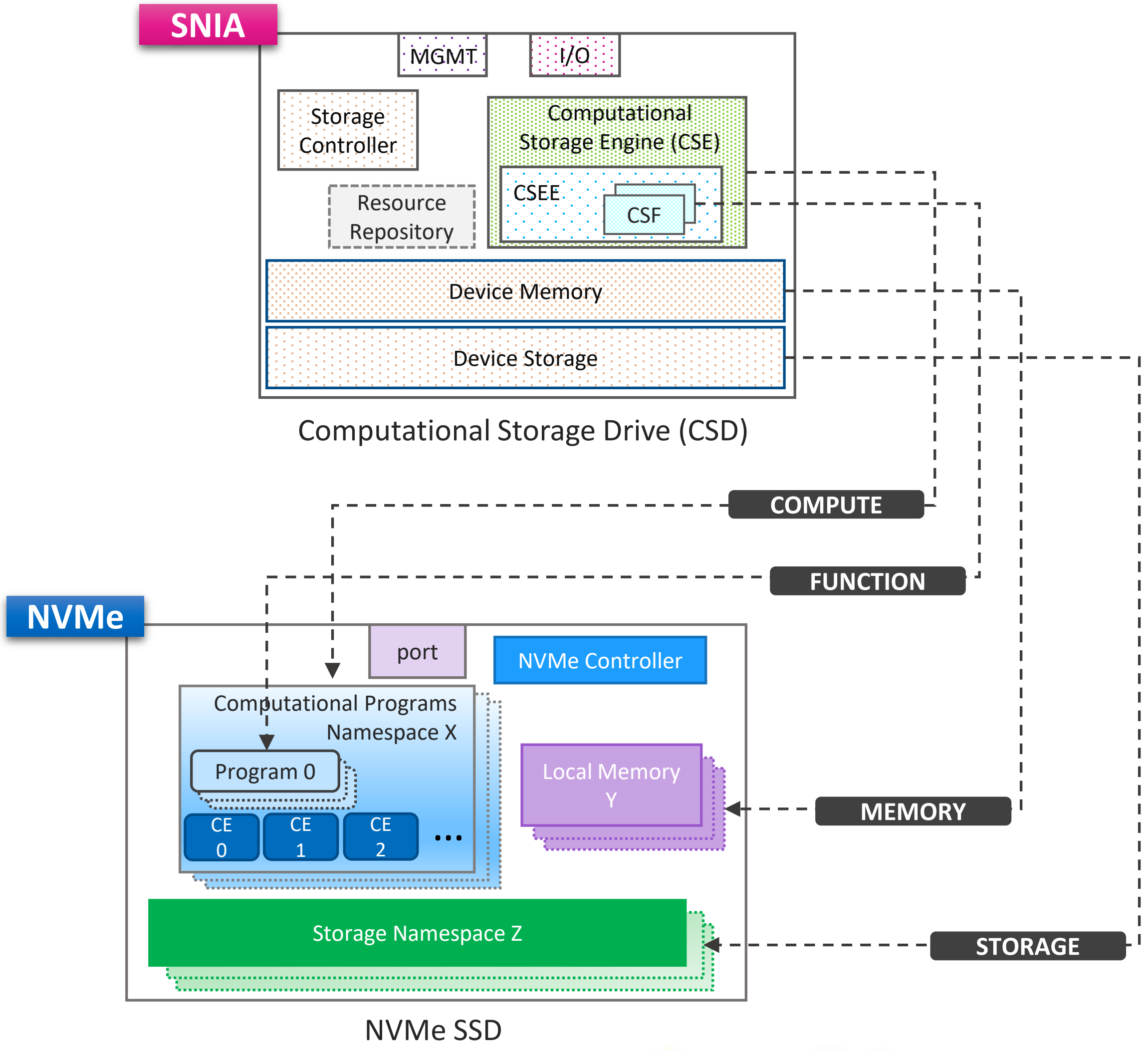
PERSISTENT MEMORY + SUMMIT 2022 COMPUTATIONAL STORAGE

CS APIs with NVMe

How do they work?

Mapping to NVMe for Computational Storage

- NVMe is developing an interface for Computational Storage*
  - Computational Programs Namespace
    - Support one or more Compute Engines (CE)
    - Support one or more Computational Programs
      - Computational Programs may be device-defined or downloaded
    - New I/O command set
  - Local Memory
    - Subsystem level scope
    - Used by Computational Programs
  - Storage Namespace
  - Map to a virtualized environment
- SNIA abstractions map to NVMe CS developments



*Optional support in NVMe

This presentation discusses NVMe work in progress, which is subject to change without notice

Summary

Summary

- SNIA: a generic Programming Interface for Computational Storage
- APIs map to different solutions
- Simple to follow and scalable
- Attend other Computational Storage sessions at the Summit

- Join the standardization efforts
  - SNIA, NVMe
- Help build the ecosystem

Please take a moment to rate this session.

Your feedback is important to us.

