STORAGE DEVELOPER CONFERENCE

SDC22

Fremont, CA
September 12-15, 2022

BY Developers FOR Developers

A SNIA Event

SNIA Computational Storage APIs

Oscar P Pinto, Principal Engineer

Samsung Semiconductor Inc.

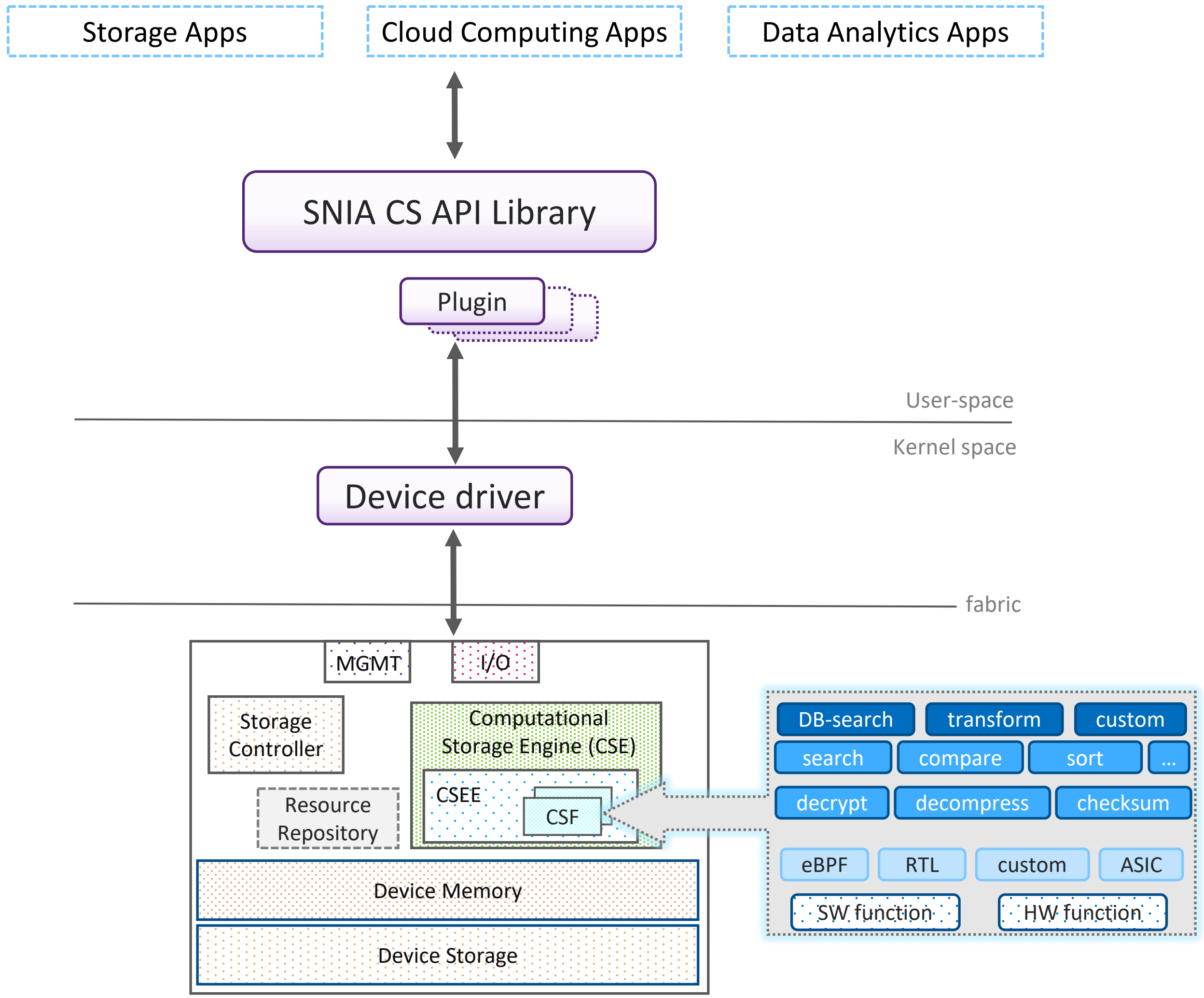
Agenda

- SNIA CS APIs
- Programming Model
  - Discover CSx Resources
  - Configure CSx Resources
  - Discover CSF
  - Execute CSF
- Programming Example
- CS APIs and NVMe
- Summary

SNIA CS APIs

SNIA Computational Storage APIs

- One API set for all CSx types
  - CSP, CSD, CSA
- APIs hide device details
  - Hardware, Connectivity (local/remote)
- Abstracts device specific details
  - Discovery
  - Access
  - Device Memory (mapped/unmapped)
  - Near Storage Access
  - Copy Device Memory
  - Download CSFs
  - Execute CSFs
  - Device Management
- Hides vendor specific implementation details
- Extensible Interface
  - Plugins connect CSx to abstracted APIs
- APIs are OS agnostic



Computational Storage Drive (CSD)

STORAGE DEVELOPER CONFERENCE SDC 22

SNIA CS API Update

- Computational Storage API v0.8 approved by SNIA for public review
  - https://www.snia.org/publicreview

- Updates since last public release
  - Simplified API Set for Resource Usage
  - Interface for Query & Configure Resource
    - Discovery and Access
    - Configuration and Activation
  - Download & Configure CSFs
    - Mechanism to download CSF to device
    - Configure and Activate CSF
  - CSF Discovery
    - Ability to discover one or more CSFs
    - Ability to choose CSF by Performance and Power Characteristics

API Overview

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

Programming Model

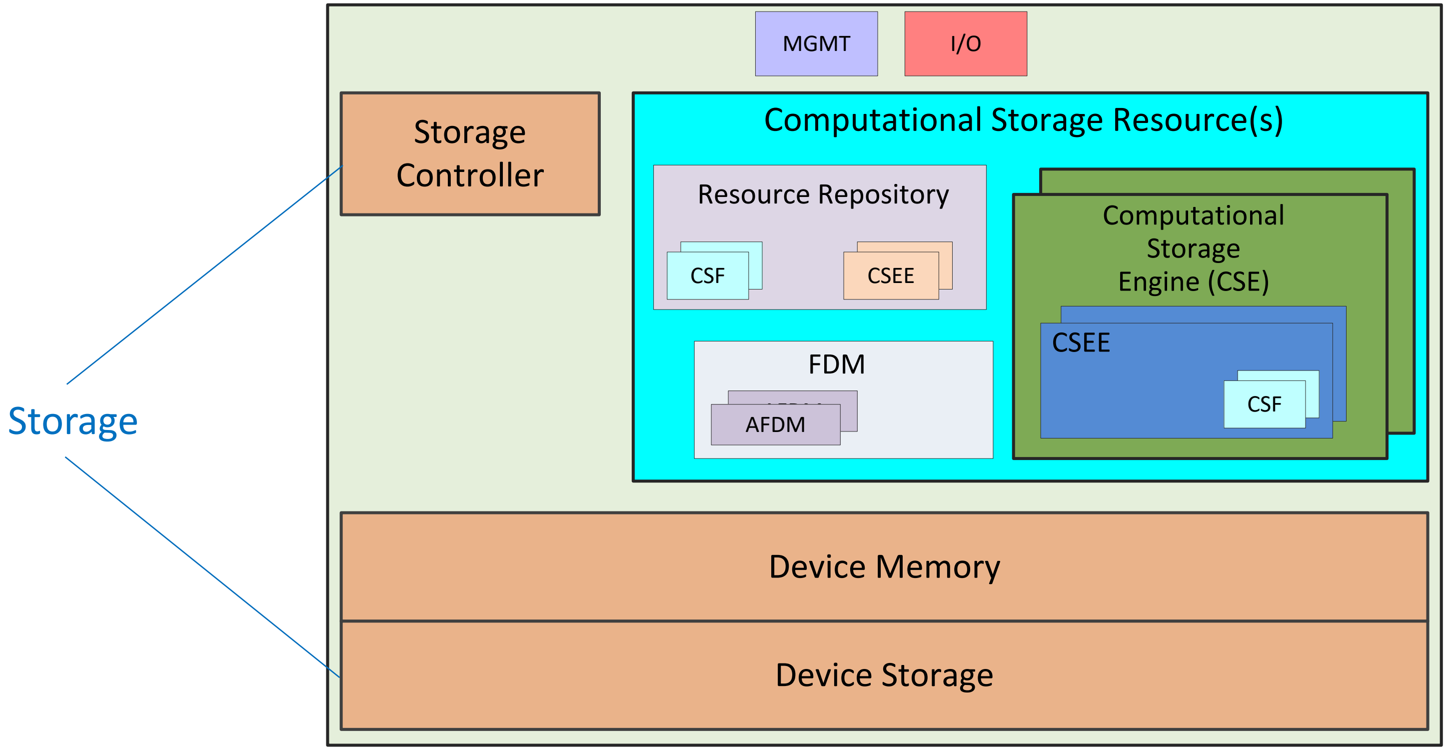
Computational Storage Programming Model

Discover Resources	Configure CSEE	Configure CSF	Discover CSFs	Execute CSFs
1	2	3	4	5
• Discover individual CS resources	• Configure execution environment for usage	• Configure functions for usage	• Discover CSFs for Execution • Choose CSFs	• Execute CSFs

← Privileged operation → (steps 1–3) ← Normal operation → (steps 4–5)

May be preconfigured by Manufacturer

CSx Overview



Computational Storage Drive (CSD)

Computational Storage Processor (CSP)

- Contains CSRs and Device Memory
- Able to execute one or more CSFs
- Storage association implementation specific

Computational Storage Drive (CSD)

- Contains CSRs, Device Memory & Storage Controller
- Able to execute one or more CSFs
- Provides persistent data storage

Computational Storage Array (CSA)

- Contains CSRs, Device Memory, Storage Controller & Control Software
- Provides virtualization to storage services, storage devices and CSRs
- Able to execute one or more CSFs
- Provides persistent data storage
- CSRs may be centrally located/distributed across CSDs/CSPs with array

CSF – Computational Storage Function
CSEE - Computational Storage Execution Environment
FDM – Function Data Memory
AFDM – Allocated Function Data Memory

CSF Overview

CSF

- Defines a set of specific operations that may be configured and executed by a CSE
- Performs only the defined operations
- May be pre-installed or downloaded
- Must be activated prior to execution

CSF

Pre-installed by Manufacturer

- Fixed Function
- May not be removed/unloaded
- May be activated/deactivated (*manufacturer dependent*)
- Fixed copies as provided

Downloaded by Host

- Downloaded to Repository
- May be unloaded
- May be activated/deactivated
- Multiple copies may be executed depending on CSEE

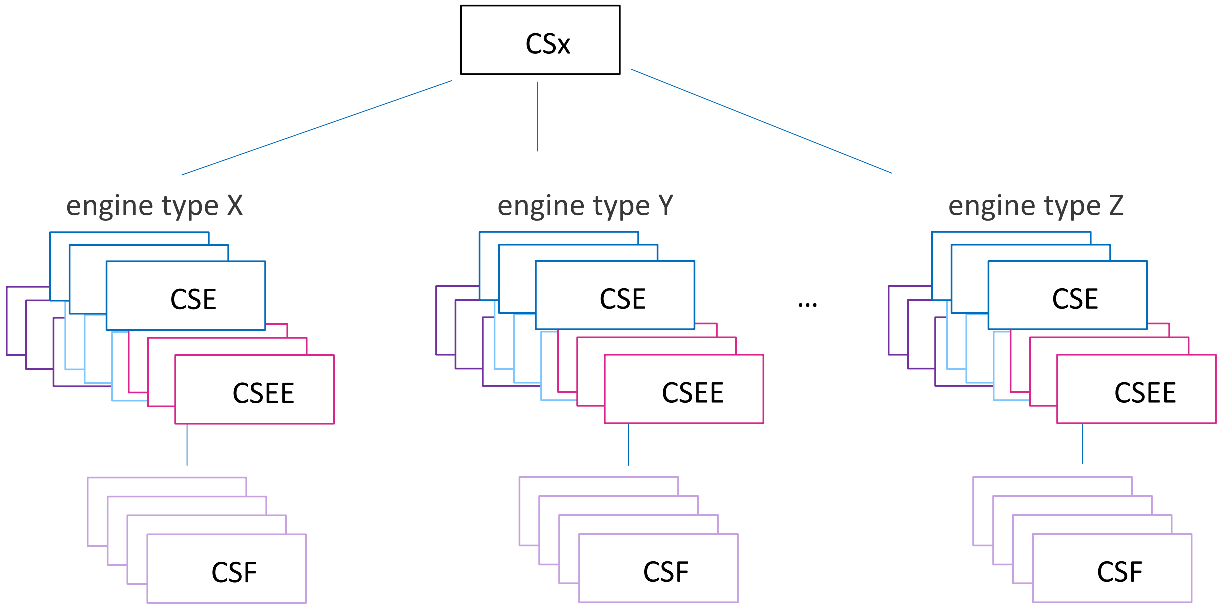
Discover Resources

1

CS_STATUS **csQueryDeviceProperties**(CS_DEV_HANDLE DevHandle, CS_RESOURCE_TYPE Type,
int *Length, CsProperties *Buffer)

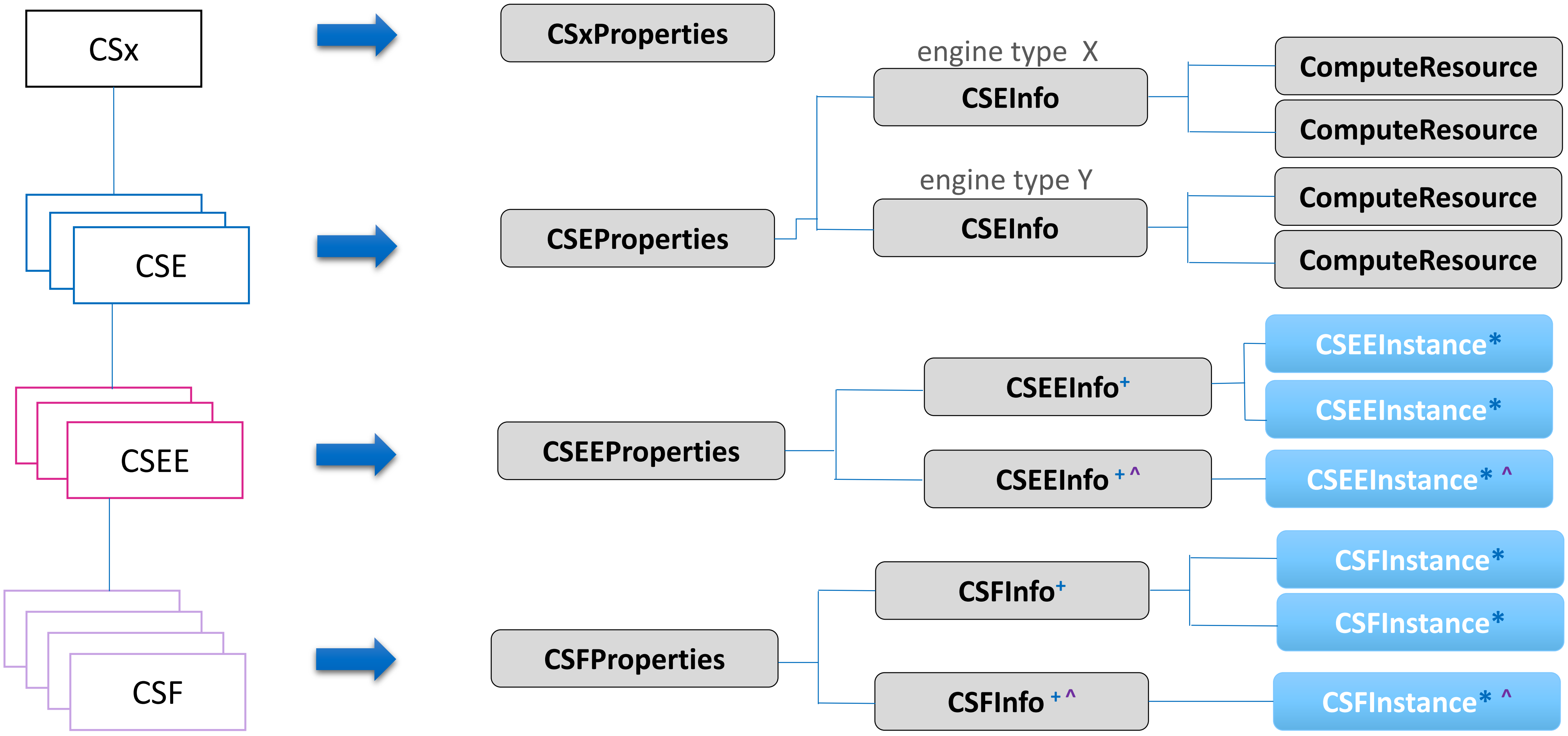
- API to Query CSx resources
- Returns resource list by type

Input	Output
RESOURCE TYPE	**PROPERTY**
CSx	CSxProperties
CSE	CSEProperties
CSEE	CSEEProperties
CSF	CSFProperties
VENDOR_SPECIFIC	CSVendorSpecific



STORAGE DEVELOPER CONFERENCE

CSx Resources Hierarchy



***CSFInstance, CSEEInstance** – activated for usage*
***+CSEEInfo, CSFInfo** – each in repository*

***^CSEEInfo, CSEEInstance** – hard-coded in CSE*
***^CSFInfo, CSFInstance** – hard-coded in CSEE*

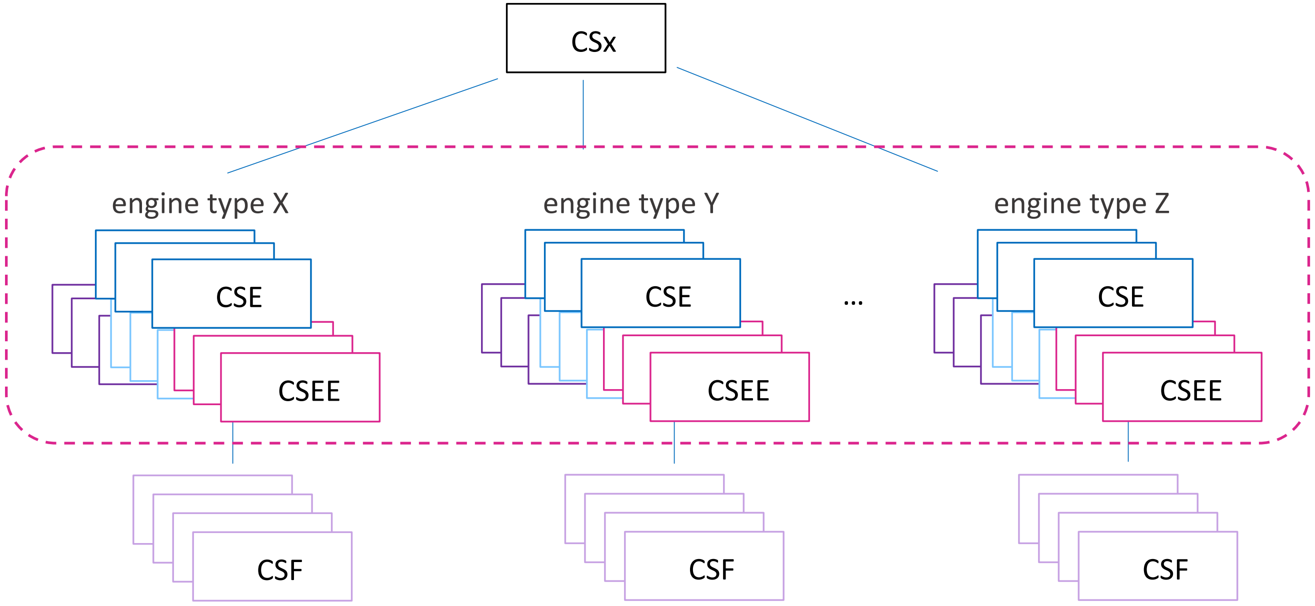
STORAGE DEVELOPER CONFERENCE

Configure CSEE

2

```
CS_STATUS csConfig(CS_DEV_HANDLE DevHandle, CsConfigInfo *Info, int *Length,
                   CsConfigData *Data)
```

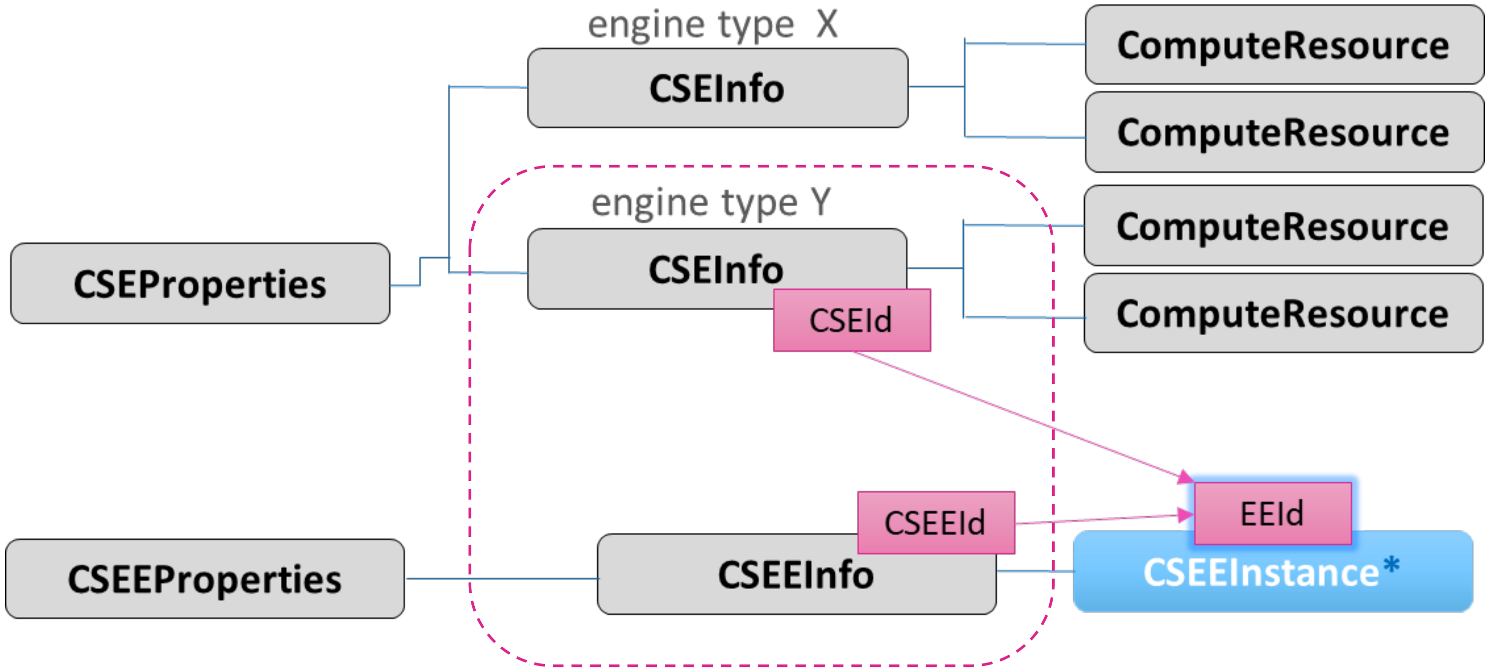
- API to Configure CSEE
  - Creates an Activated Instance
  - Select CSE & CSEE from Repository
  - Activate CSEE Instance
- Returns Activated CSEE Instance



Input | **Property**

CONFIG TYPE	PROPERTY
CSEE	CSEEActivateConfig

Configure CSEE - *cont.*



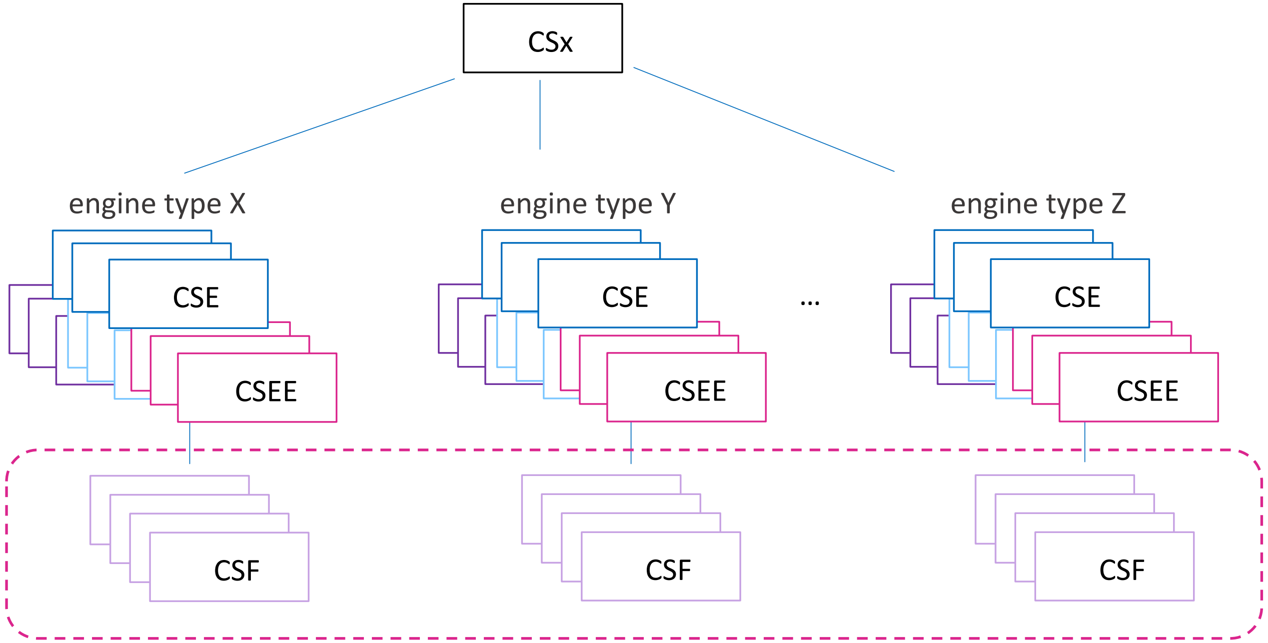
- Pair CSEInfo with CSEEInfo (*must be of the same CSETypeToken*)
- Activation creates a new Instance of the resource

Configure CSF

3

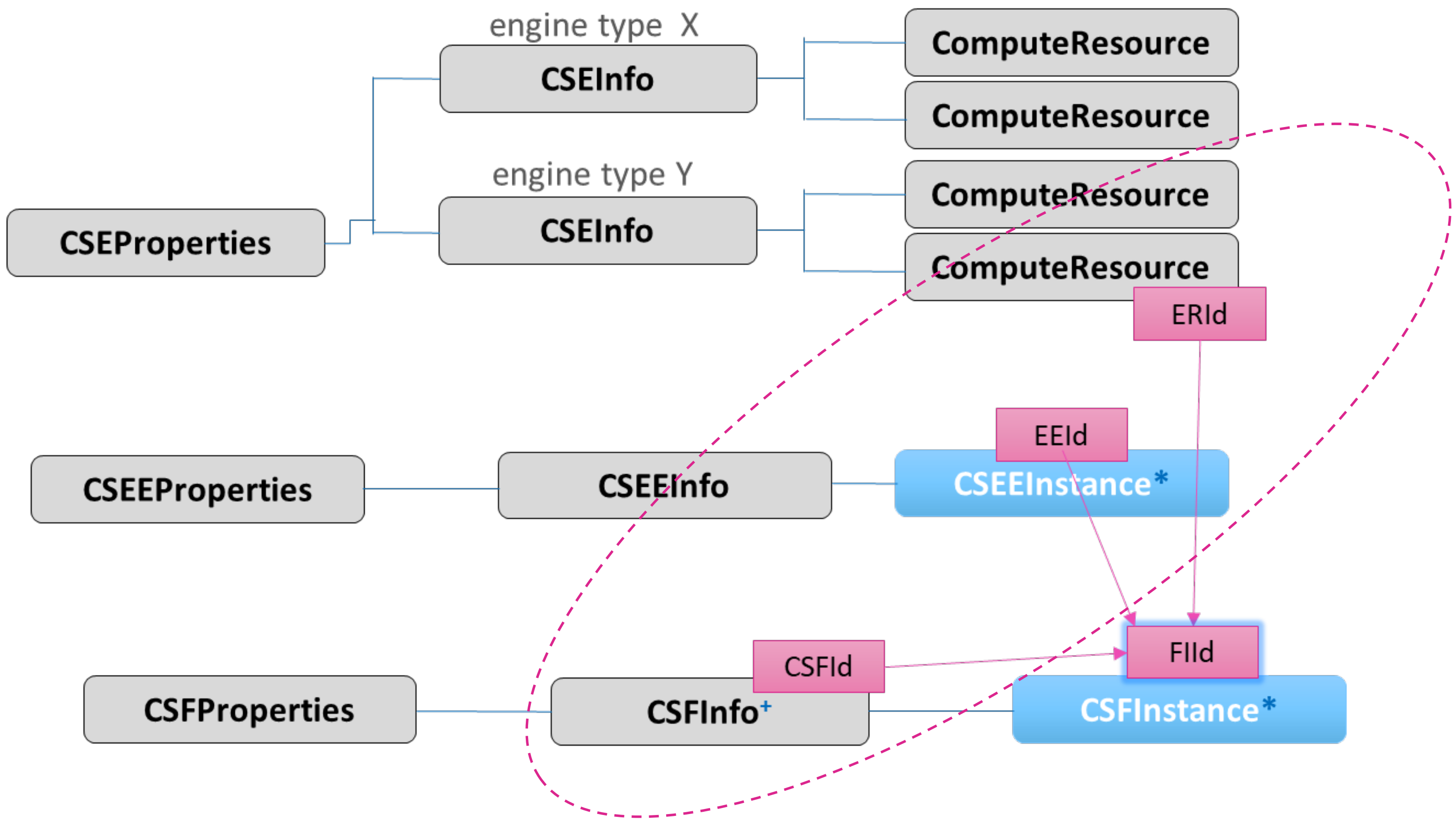
CS_STATUS **csConfig**(CS_DEV_HANDLE DevHandle, CsConfigInfo *Info, int *Length,
CsConfigData *Data)

- API to Configure CSF
  - Creates an Activated Instance
  - [*optional*] Download CSF to Repository
  - Select CSF & Activated CSEE Instance
    - Select Compute Resources as needed
  - Activate CSF Instance
- Returns Activated CSF Instance



Input	Property
CONFIG TYPE	PROPERTY
CSF	CSFActivateConfig

Configure CSF - *cont.*



- Activation creates a new Instance of the CSF
- Only an Activated CSF Instance is available for Execution

Discover CSFs

4

CS_STATUS **csQueryCSFList**(char *Path, int *Length, char *Buffer)

- API to Discover CSFs before Access
  - Helps choose CSx by available CSF types
  - Find all (Activated) CSFs for CSx by a valid Path
    - Or across all CSxes with a NULL Path
- Returns a list of CSFs by name

CS_STATUS **csGetCSFId**(CS_DEV_HANDLE DevHandle, char *CSFName, int *Length, CSFIfInfo *Buffer)

- API to Discover CSFs after Access
  - Helps choose desired CSF in CSx
  - Choose by Performance, Power and Instances
- Returns a list of CSFs by specific characteristics

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


Execute CSF

5

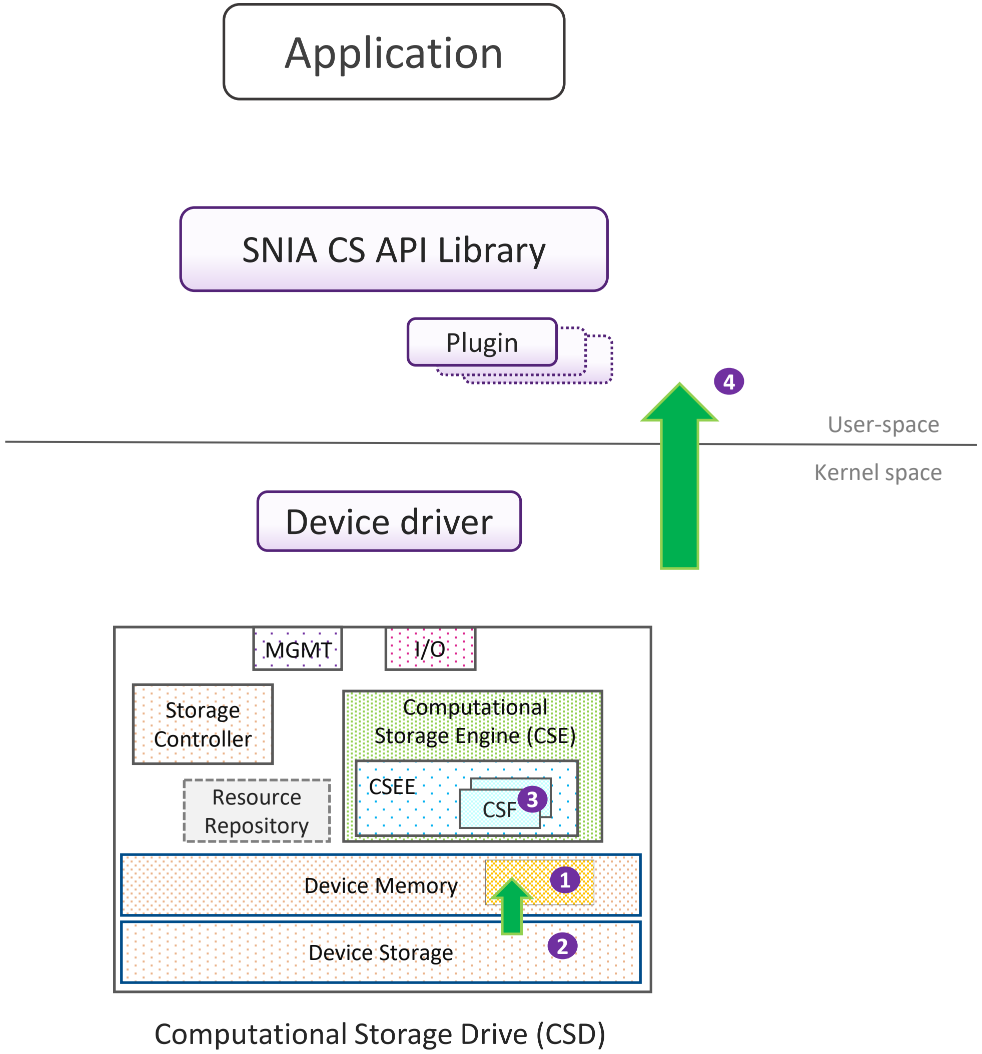
```
CS_STATUS csQueueComputeRequest(CsComputeRequest *Req, void *Context,
                                csQueueCallbackFn CallbackFn, CS_EVENT_HANDLE EventHandle,
                                u64 *CompValue)
```

- API to Execute CSF with CS request
  - Queues a compute request to CSx
  - Request describes CSF input/output parameters
  - Supports Synchronous/Asynchronous completion modes
    - Asynchronous supports callback or event mode
- Synchronous mode: Returns only after request completes
- Asynchronous mode: Returns immediately after queuing the request

```
typedef struct {
    CS_CSF_ID CSFId;        // unique Identifier to schedule compute work
    int NumArgs;            // total number of arguments to CSF
    CsComputeArg Args[1];   // Argument list
} CsComputeRequest;
```


Programming Example

Example: Run Data Filter

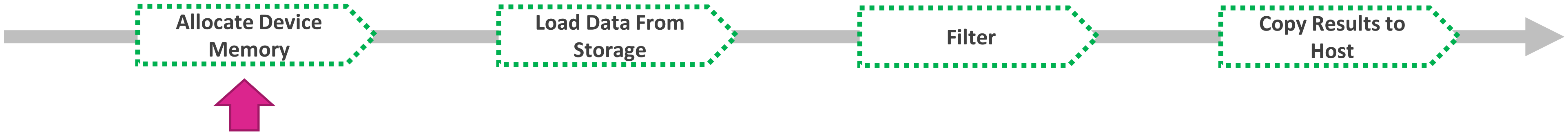


1. Allocate Device Memory
2. Load Storage data in Device Memory
3. Run Data Filter CSF
4. Copy Results to Host Memory

Example: Allocate Device Memory

1. Allocate Device Memory
   - Allocate memory for required buffers
     - Buffer1 - load data from storage
     - Buffer2 – collect results of filter

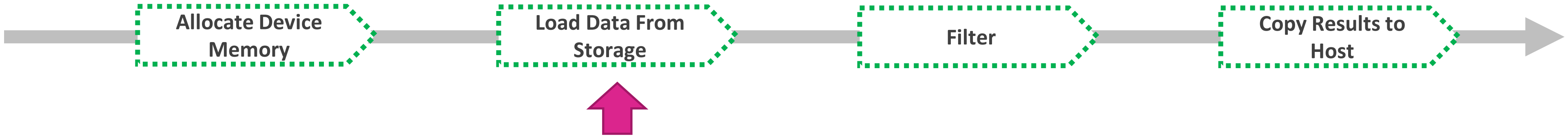
```
// allocate device memory for input and output buffers
status = csAllocMem(devHandle, CHUNK_SIZE, 0, &inputMemHandle, NULL);
status = csAllocMem(devHandle, CHUNK_SIZE, 0, &resultsMemHandle, NULL);
```



Example: Load Storage Data

2. Load Storage Data directly in Device Memory

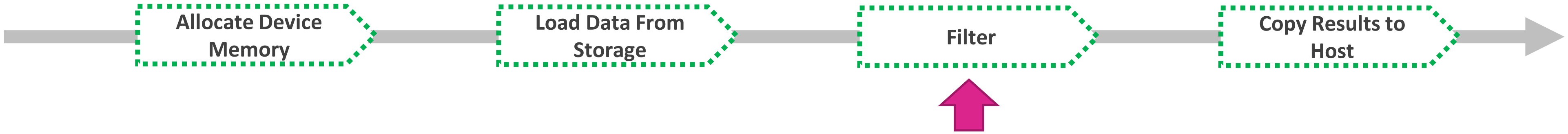
```
// allocate storage request & read chunk size data from file handle fd
storReq = calloc(1, sizeof(CsStorageRequest));
if (!storReq) { ERROR_OUT("memory alloc error\n"); }

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: Run Data Filter

3. Execute Data Filter CSF in CSx

```
// allocate compute request for 3 args & issue compute request API
compReq = calloc(1, sizeof(CsComputeRequest) + (sizeof(CsComputeArg) * 3));
if (!compReq) { ERROR_OUT("memory alloc error\n"); }

compReq->CSFId = ScanQueryId;
compReq->NumArgs = 3;
argPtr = &compReq->Args[0];
csHelperSetComputeArg(&argPtr[0], CS_AFDM_TYPE, inputMemHandle, 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);
```



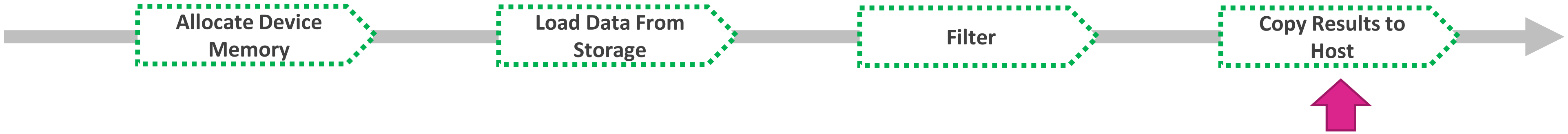
 |

Example: Copy Results

4. Copy Output Results to Host Memory

- Copy Device Memory Contents to Host

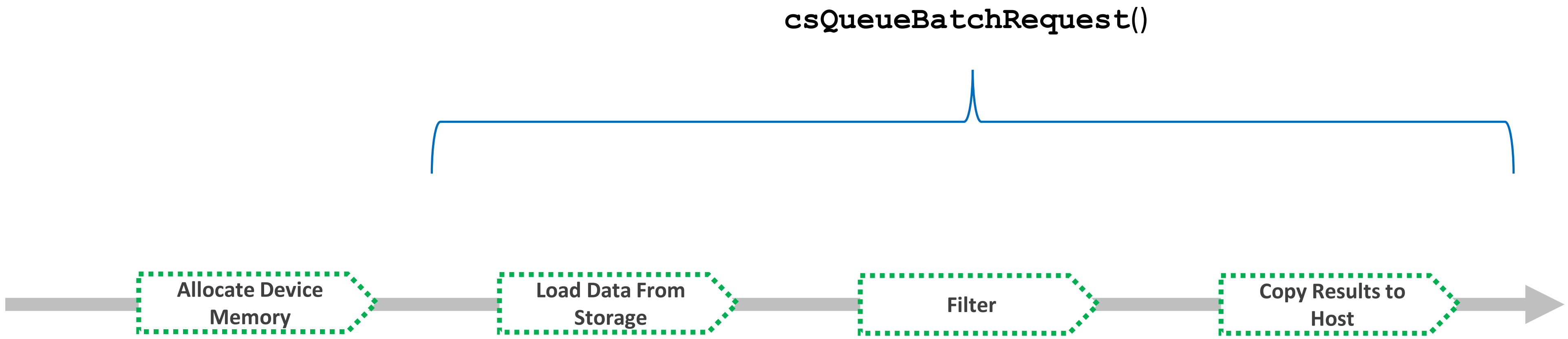
```
// allocate copy request & copy results to host buffer
copyReq = calloc(1, sizeof(CsCopyMemRequest));
if (!copyReq) { ERROR_OUT("memory alloc error\n"); }

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);
```



STORAGE DEVELOPER CONFERENCE SDC 22

Batching the Request

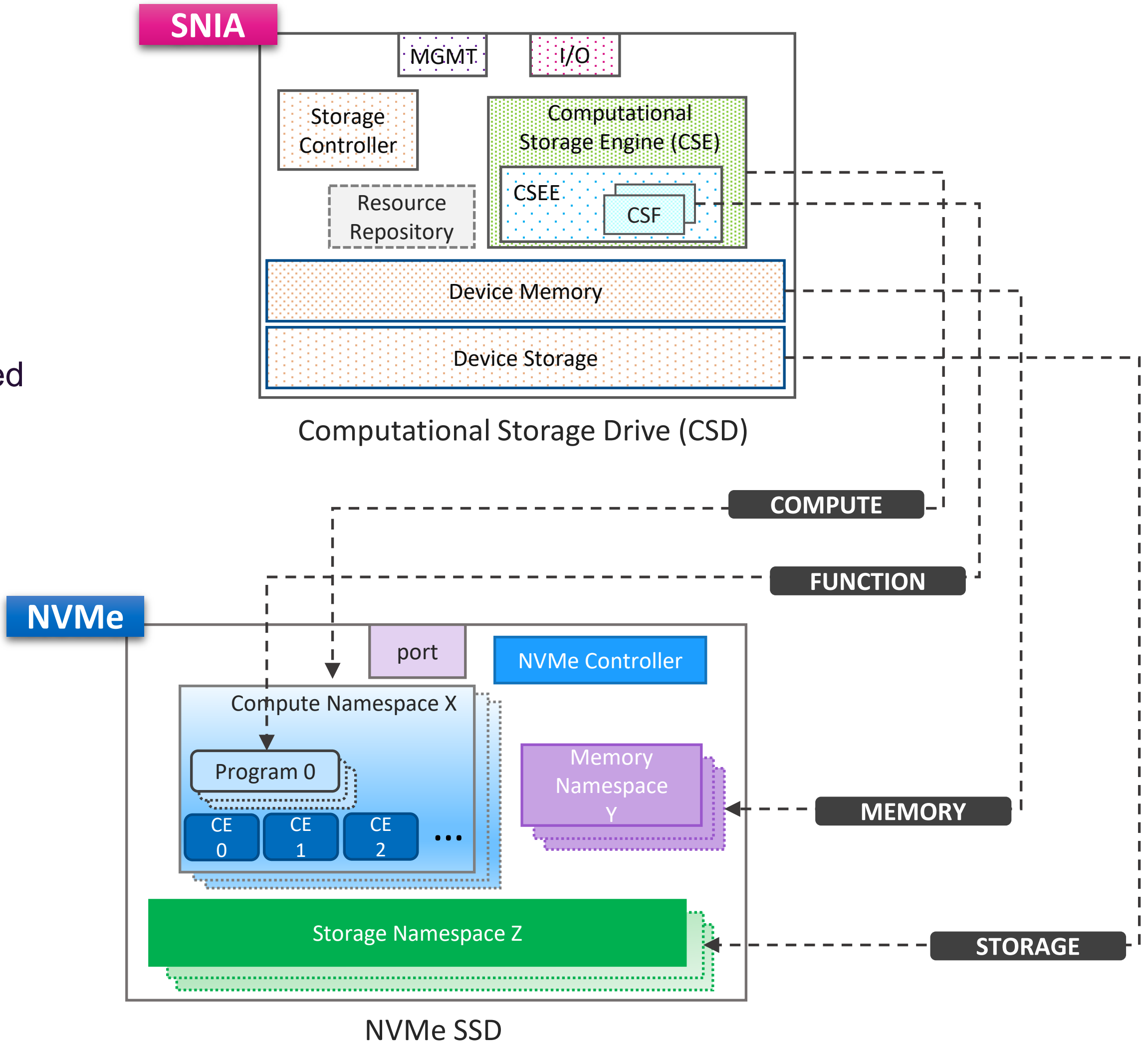
- Create one Batch request that includes all requests in one job
  - Optimization for recurring jobs
  - Submit request and get notified on final Results



CS APIs & NVMe

Mapping to NVMe for Computational Storage

- NVMe is developing an interface for Computational Storage*
  - Compute Namespace [*new*]
    - 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
  - Memory Namespace [*new*]
    - Subsystem level scope
    - Used by Computational Programs
    - New I/O command set
  - Storage Namespace
  - Map to a virtualized environment
- SNIA abstractions map to NVMe CS developments



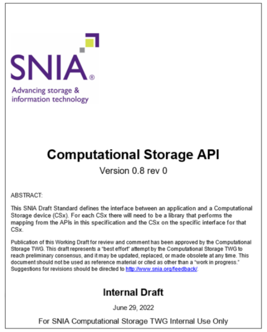
Optional support in NVMe

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 device solutions
- Simple to follow and scalable
- v0.8 available for public review
- Attend other Computational Storage sessions



SNIA
Advancing storage & information technology

Computational Storage API

Version 0.8 rev 0

ABSTRACT:

This SNIA Draft Standard defines the interface between an application and a Computational Storage device (CSx). For each CSx there will need to be a library that performs the mapping from the APIs in this specification and the CSx on the specific interface for that CSx.

Publication of this Working Draft for review and comment has been approved by the Computational Storage TWG. This draft represents a "best effort" attempt by the Computational Storage TWG to reach preliminary consensus, and it may be updated, replaced, or made obsolete at any time. This document should not be used as reference material or cited as other than a "work in progress." Suggestions for revisions should be directed to http://www.snia.org/feedback/.

Internal Draft

June 29, 2022

For SNIA Computational Storage TWG Internal Use Only

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

Please take a moment to rate this session.

Your feedback is important to us.

