Computational Storage Programming

Presented by
Oscar P Pinto
Samsung Semiconductor Inc.

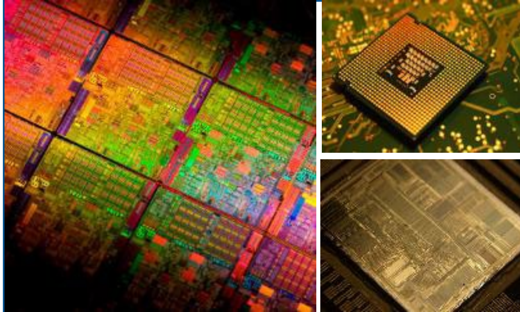
Agenda

- Overview
- Programming Usage
- Walkthrough APIs by Example
- Specification Update
- Summary

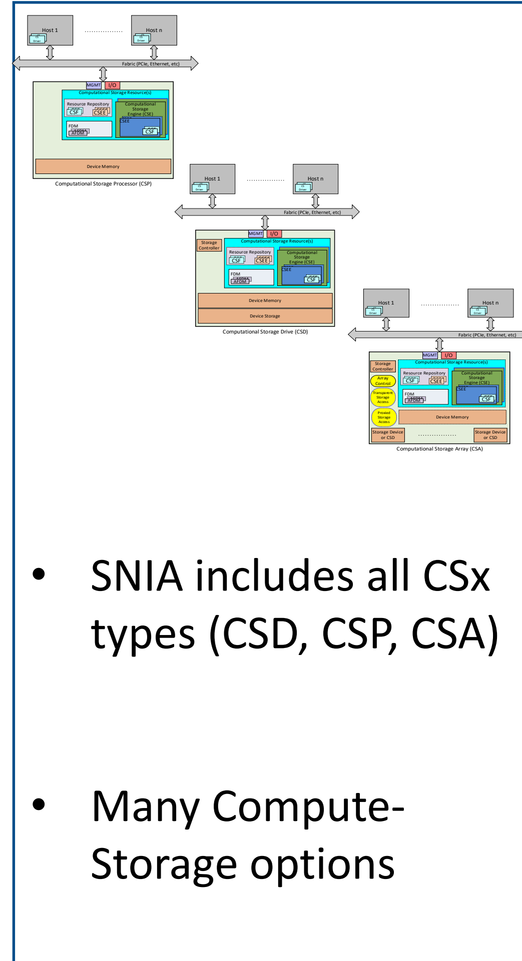
 ©2023 SNIA. All Rights Reserved.

SDC 23

Why Computational Storage APIs?



- Many Compute Interfaces available
- Memory based only



- SNIA includes all CSx types (CSD, CSP, CSA)
- Many Compute-Storage options

SNIA CS APIs

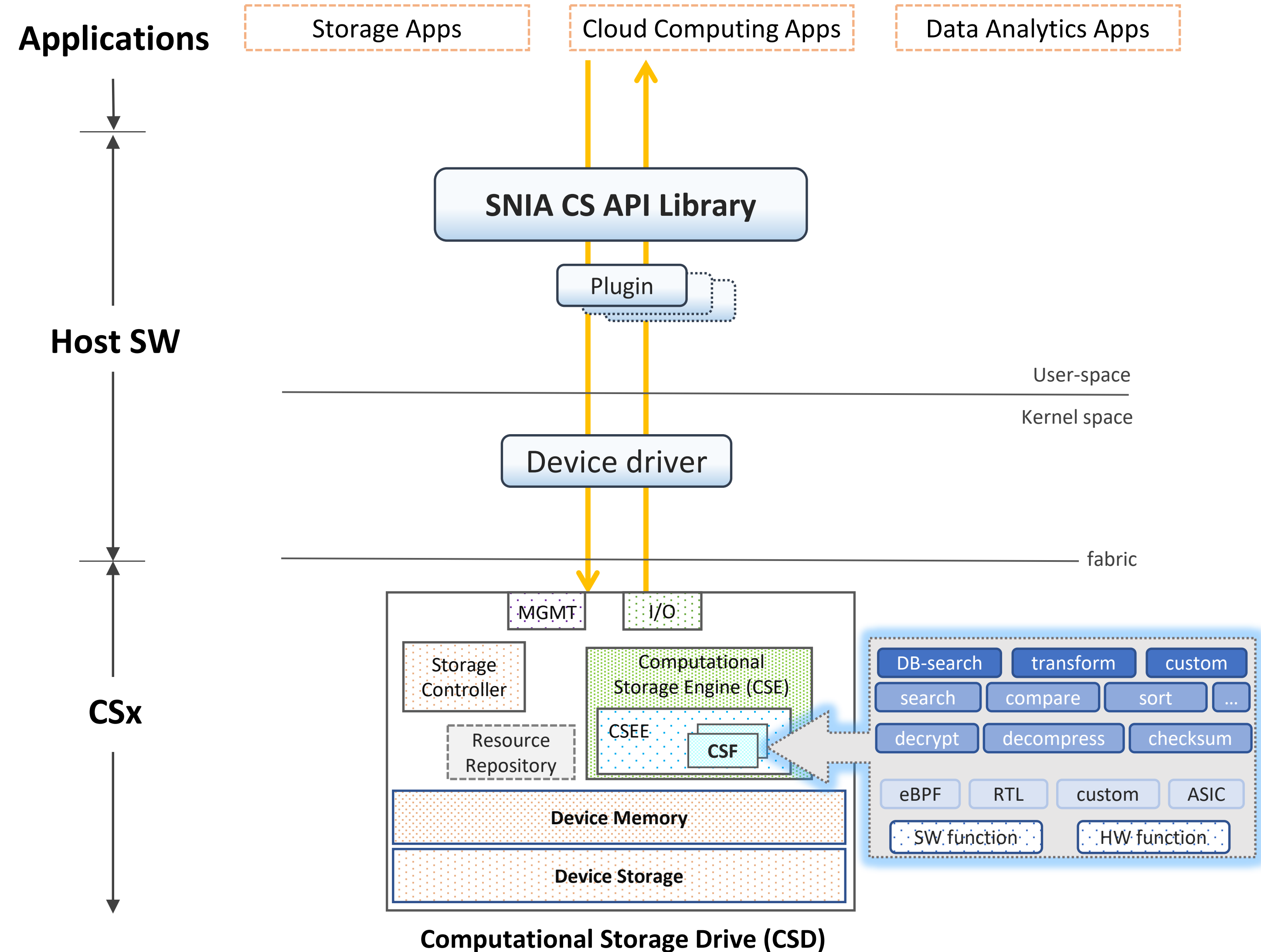
- Takes near storage compute into account

This Photo by Unknown Author is licensed under CC BY-NC-ND

SDC 23

SNIA CS APIs

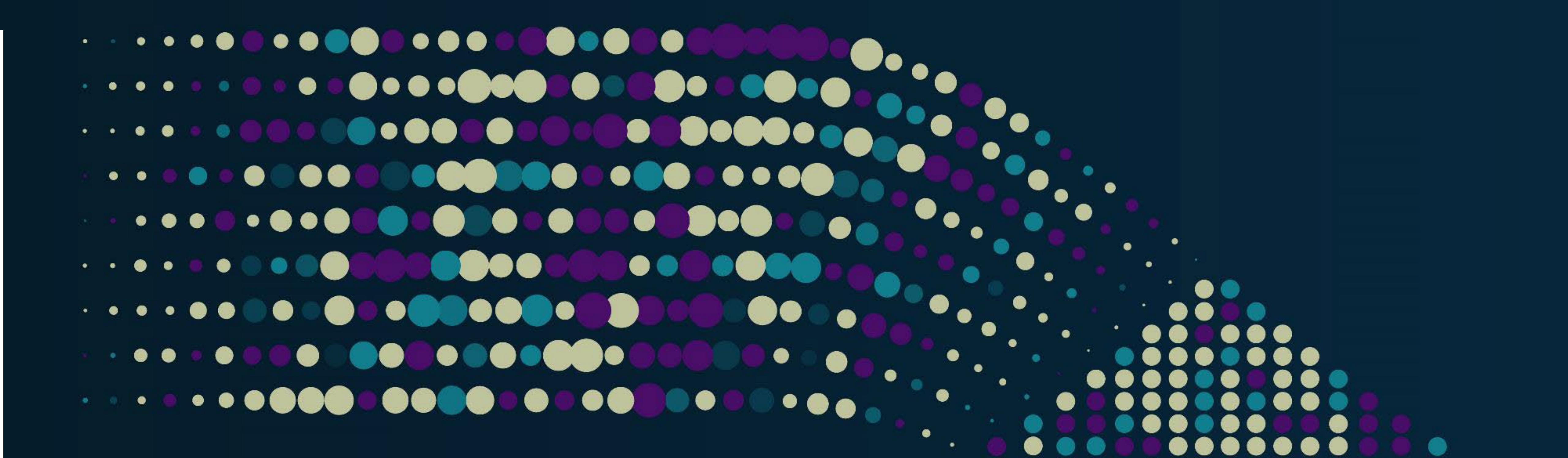
1. One set for all CSx types
   - CSP, CSD, CSA
2. Hides Device Details
   - Hardware, Connectivity (local/remote)
   - Vendor specific Implementations
3. Abstracts Device Interface
   - Discovery
   - Access
   - Device Memory (mapped/unmapped)
   - Near Storage Access
   - Copy Device Memory
   - Download CSFs
   - Execute CSFs
   - Device Management



CSP – Computational Storage Processor
CSA – Computational Storage Array
CSF – Computational Storage Function

 ©2023 SNIA. All Rights Reserved.

SDC 23



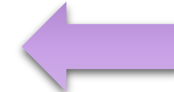
Programming Usage

©2023 SNIA. All Rights Reserved.

SDC 23

Programming Modes

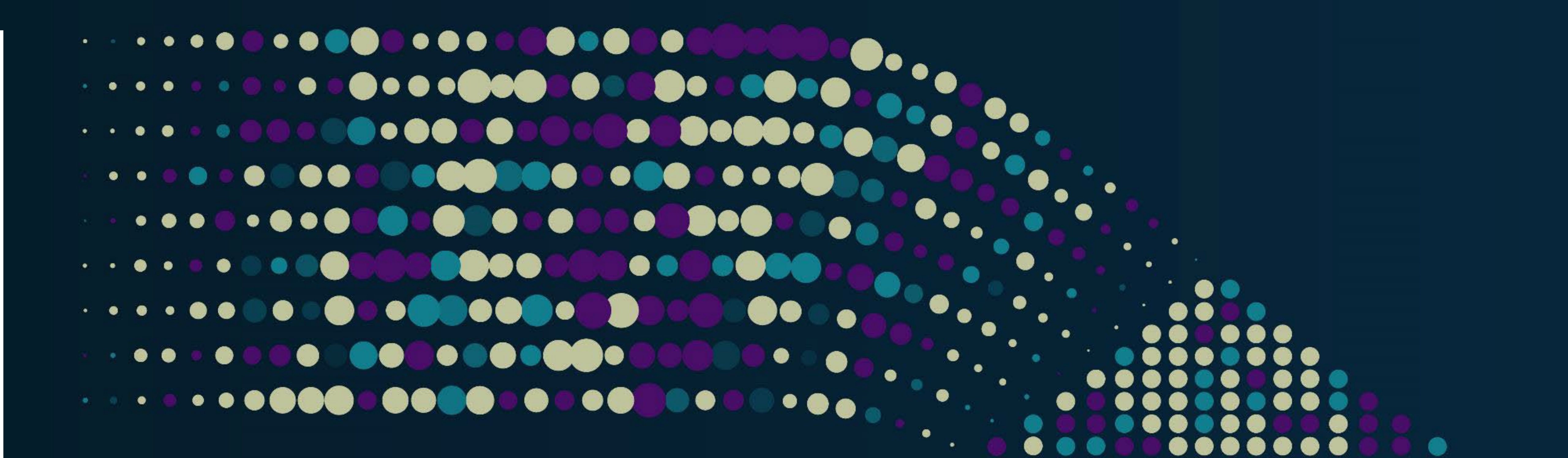
- 2 Programming Modes
  - Privileged Mode Operations (Administrator)
    - Configure CS Resources
    - Download CSFs
    - Manage Device
  - Non-privileged Mode Operations (Normal User)



    - Discover CSx, CSFs
    - Allocate Device Memory
    - Execute CSFs
    - Transfer data between Storage & Device Memory (P2P)
    - Copy data between Device Memory & Host Memory

 ©2023 SNIA. All Rights Reserved.

SDC 23



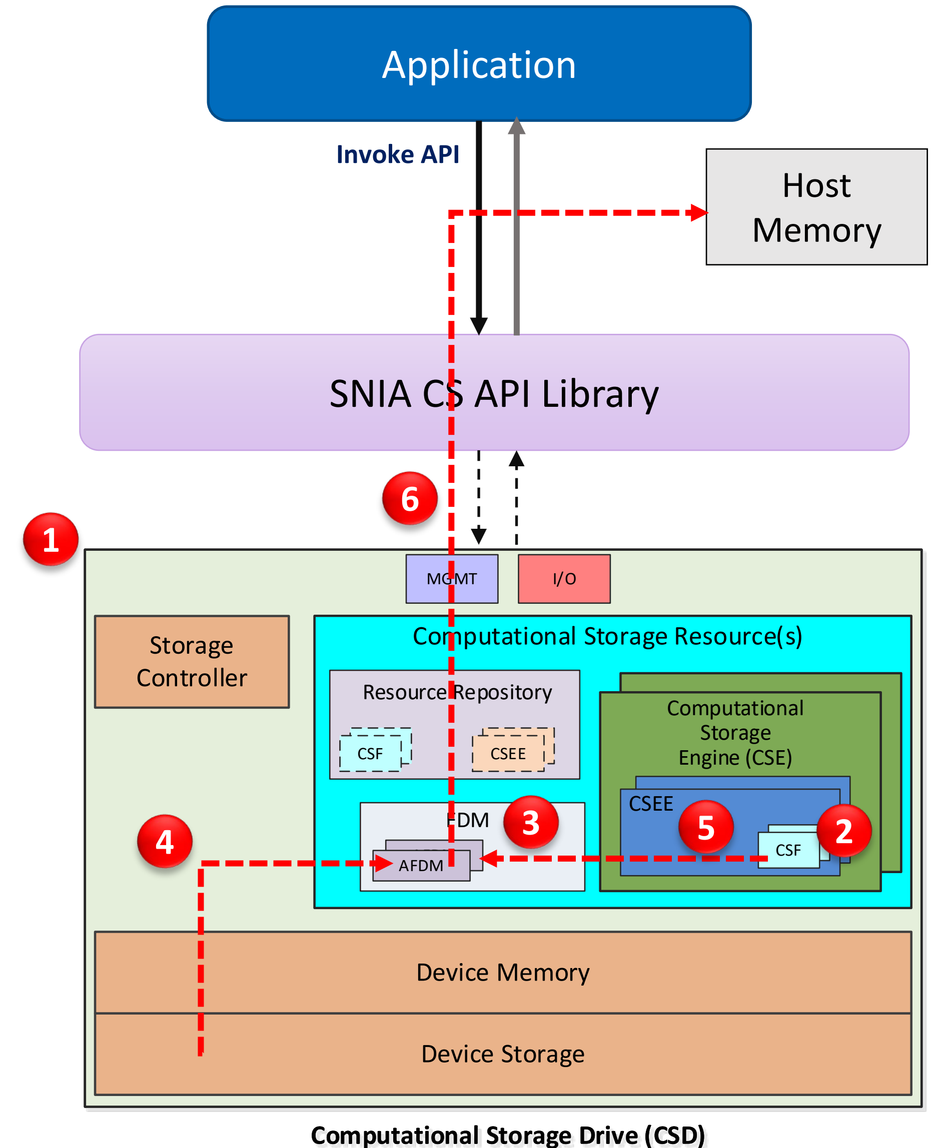
APIs by Example

©2023 SNIA. All Rights Reserved.

SDC 23

Example

- Execute Data Filter
  1. Discovery Device
  2. Discover CSF
  3. Allocate Device Memory (FDM)
  4. Load Data from Storage
  5. Run Data Filter CSF on loaded Data
  6. Copy Results to Host Memory



FDM – Function Device Memory

 ©2023 SNIA. All Rights Reserved.

SDC 23

Prepare for Computational Storage - Setup

Discover CSx

- Discover by name
- Access device

Discover CSF

- Discover the function(s) you want to execute

Allocate FDM

- Allocate Device Memory

Ready

CSx – Computational Storage Device
CSF – Computational Storage Function

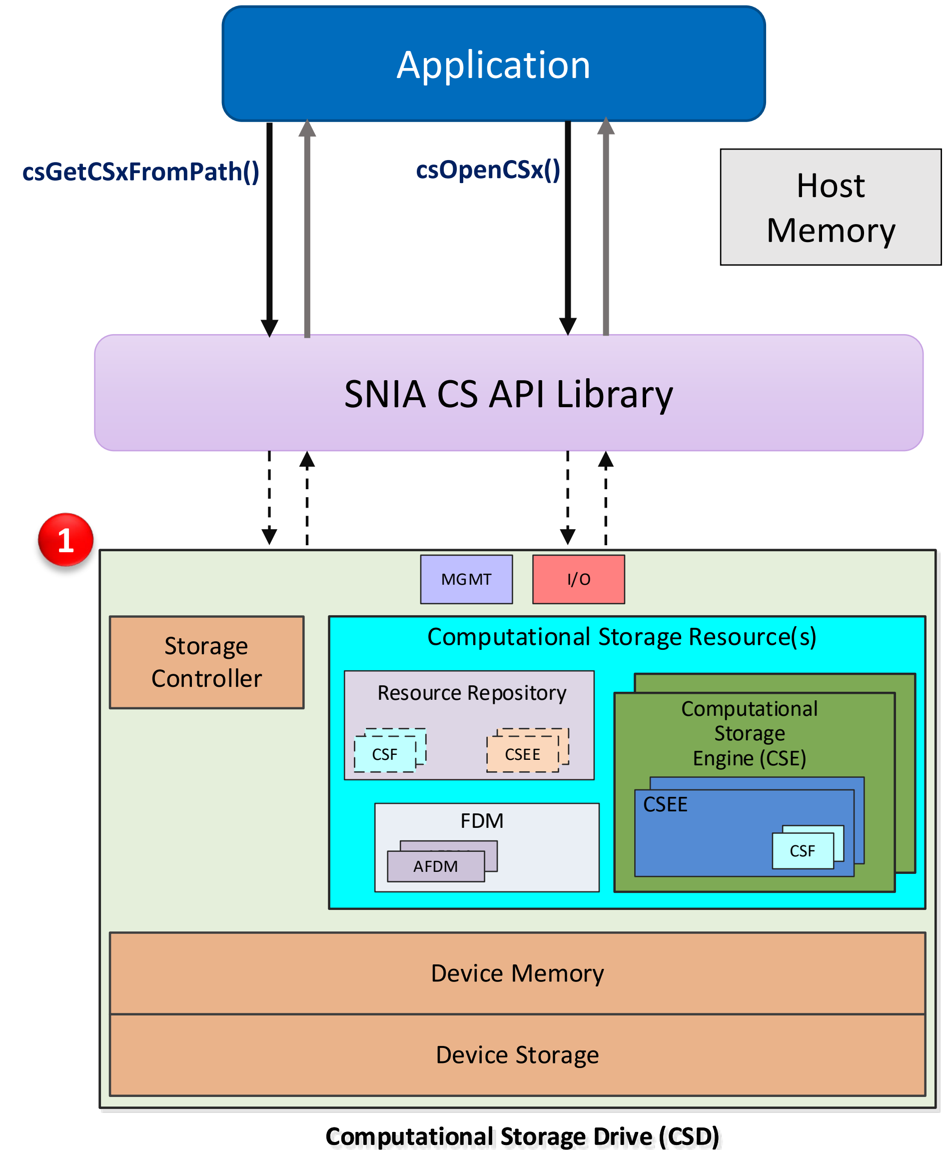
 ©2023 SNIA. All Rights Reserved.

SDC 23

Setup: Discover Device

step 1 **Discover CSx**

- Discover CS device
  - Identify the device
    - By device path, path to file/directory
    - By list of all available CS devices
  - *Skip this step if device is known*
- Access CS device
  - Request access to selected device



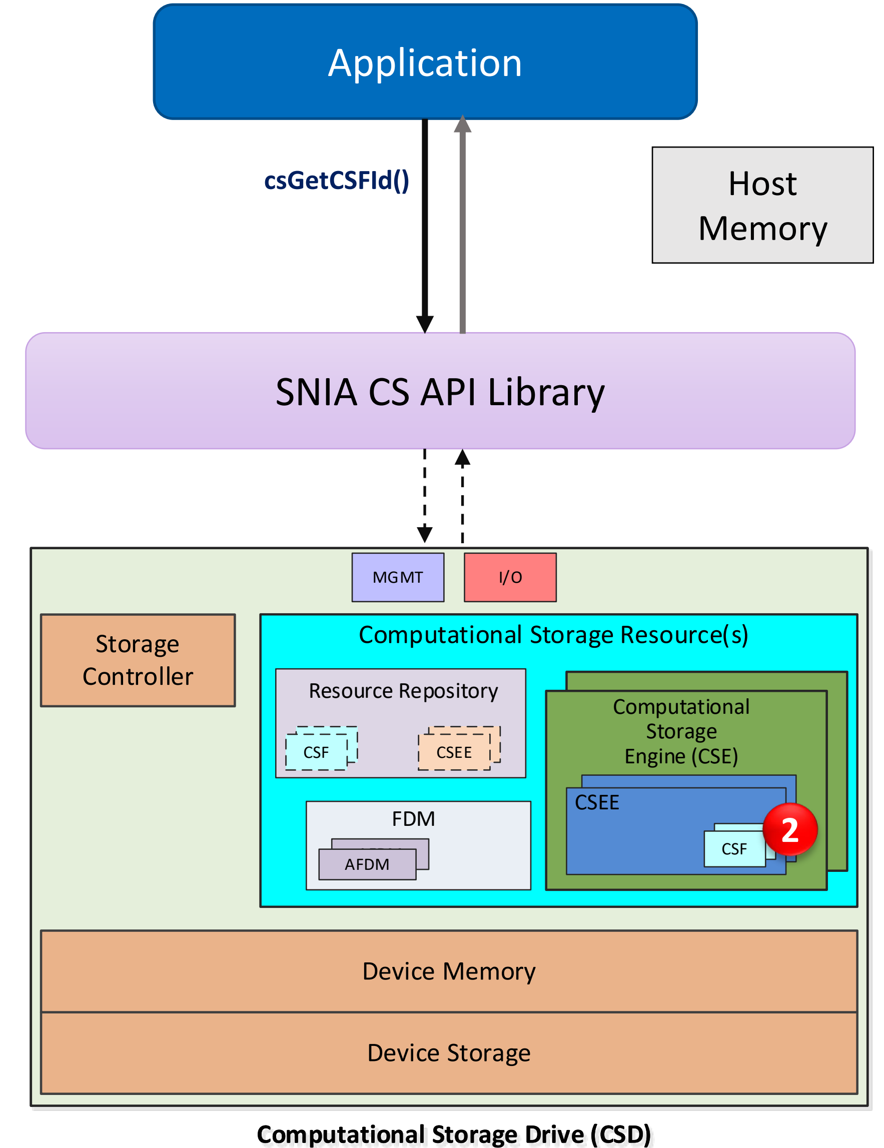
 ©2023 SNIA. All Rights Reserved.

SDC 23

Setup: Discover Function

step **2** **Discover CSF**

- Discover your CSF
  - By Name or Global Identifier
- Returns a list of one or more
  - Each CSF instance contains
    - Relative Performance
    - Relative Power
    - Count for this instance
    - FDMs accessible
- Pick the best one if > 1 available



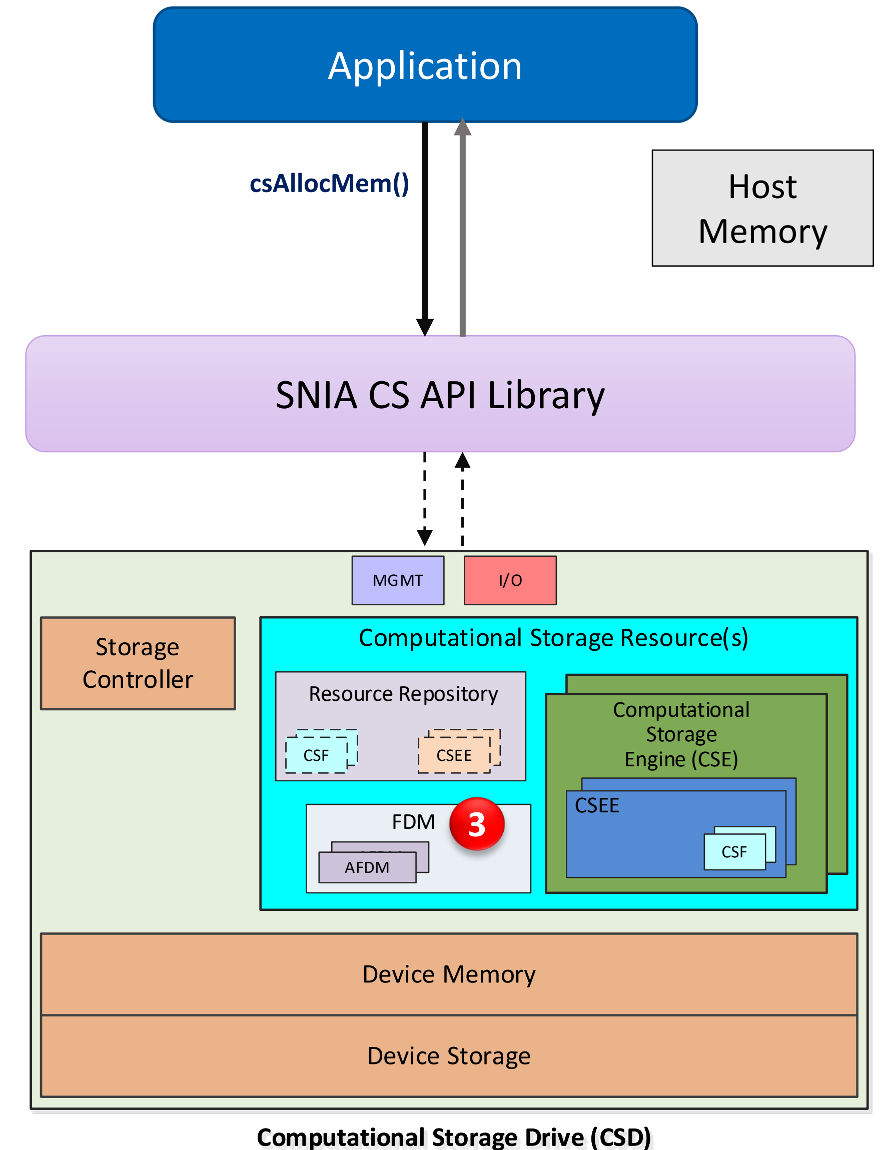
 ©2023 SNIA. All Rights Reserved.

SDC 23

Setup: Allocate Device Memory

step **3** **Allocate FDM**

- Select FDM from Function chosen
  - Select FDM if > 1 available
    - Memory should be accessible by compute flow
- Allocate FDM as necessary
  - Request memory with additional details
    - Initialized (clear/fill)
    - Map to Host address space
      - If device permits



 ©2023 SNIA. All Rights Reserved.

SDC 23

Setup: Code

1

Find CSx near storage

```
// Find my CSx near storage
status = csGetCSxFromPath("my_file_path", &length, &csxBuffer);
if (status != CS_SUCCESS)
    ERROR_OUT("No CSx device found!\n");
// open device, init function and preallocate buffers
status = csOpenCSx(csxBuffer, devContext, &devHandle);
if (status != CS_SUCCESS)
    ERROR_OUT("Could not access CSx\n");
```

Access to CSx

2

Find CSF by name

```
// Get access to the CSF to run
status = csGetCSFId(devHandle, "filter", 0, &infoLength, &count, &csfInfo);
if (status != CS_SUCCESS)
    ERROR_OUT("CSX does not contain any decrypt CSFs \n");
// pick highest performant CSF
CSFIdInfo *p = csfInfo;
CSFIdInfo *myCSF = NULL;
for (i=0; i< count; i++, p++) {
    if ((myCSF == NULL) ||
        ((myCSF != NULL) && (p->RelativePerformance > myCSF->RelativePerformance))) {
        myCSF = p;
    }
}
printf("CSFId: %d, RelativePerformance: %d\n", myCSF->CSFId,
                                               myCSF->RelativePerformance);
```

Pick most performant in list if > 1

3

```
// Pick most performant FDM for CSF chosen
FDMAccess *p = myCSF->FDMList;
FDMAccess *myFDM = NULL;
for (i = 0; i < myCSF->NumFDMs; i++, p++) {
    if ((myFDM == NULL) ||
        ((myFDM != NULL) && (p->RelativePerformance > myFDM->RelativePerformance))) {
        myFDM = p;
    }
}
// allocate FDM for CSF usage
CsMemFlags f;
f.s->FDMId = myFDM->FDMId;
f.s->Flags = 0;   // may also be CS_FDM_CLEAR
status = csAllocMem(devHandle, CHUNK_SIZE, &f, &afdmHandle1, NULL);
if (status != CS_SUCCESS)
    ERROR_OUT("AFDM alloc error\n");
```

Pick most performant FDM in list if > 1

Allocate required size

 ©2023 SNIA. All Rights Reserved.

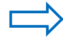
SDC 23

Perform Computational Storage - Run

Load Storage Data

4

- Load Data near Compute



Execute CSF

5

- Run Compute Operation

Copy Results

6

- Copy from FDM into Host Memory

Done

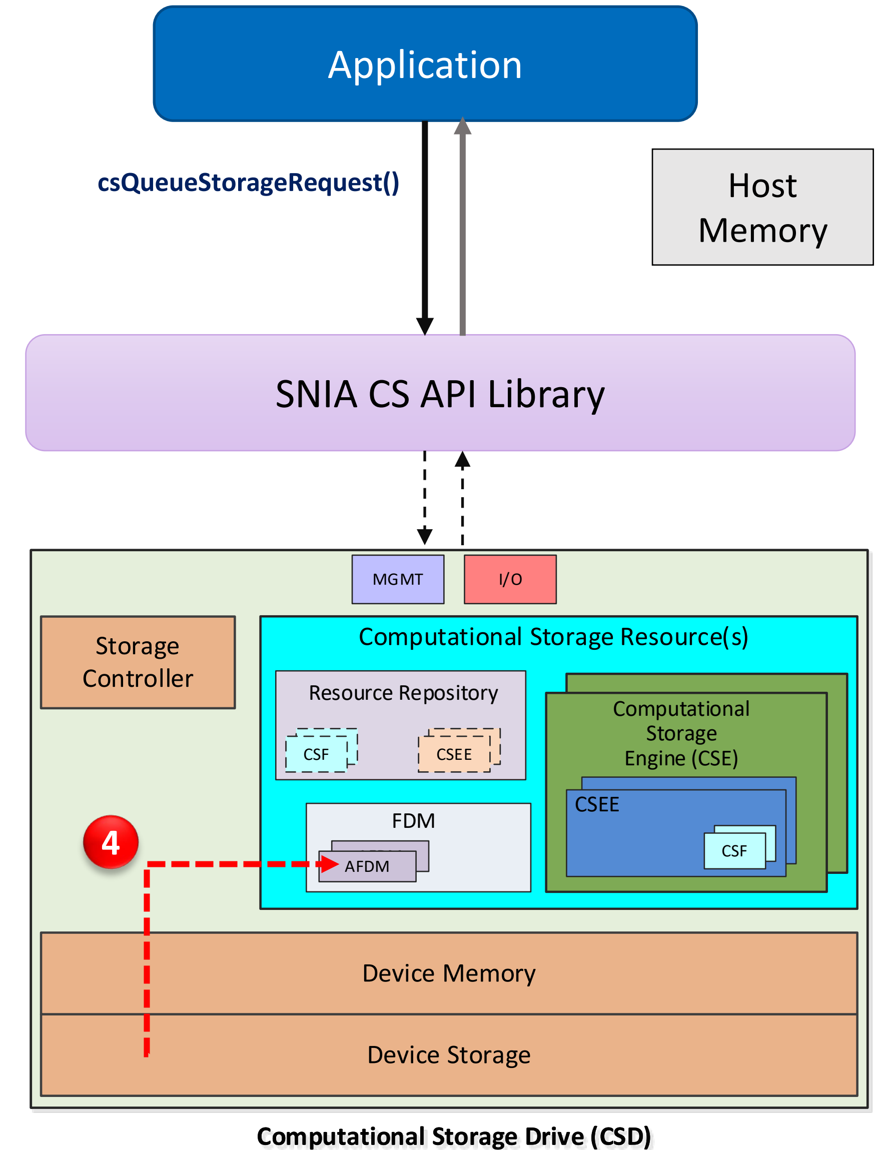
©2023 SNIA. All Rights Reserved.

SDC 23

Run: Load Storage Data

step **4** **Load Storage Data**

- Load data from storage into FDM
  - Data may be described as
    - LBA ranges
    - File handle & offset
- Data does not leave the device (P2P)
  - Save on fabric bandwidth
- More than one Completion options
  - Synchronous
  - Asynchronous
    - Callback or Event

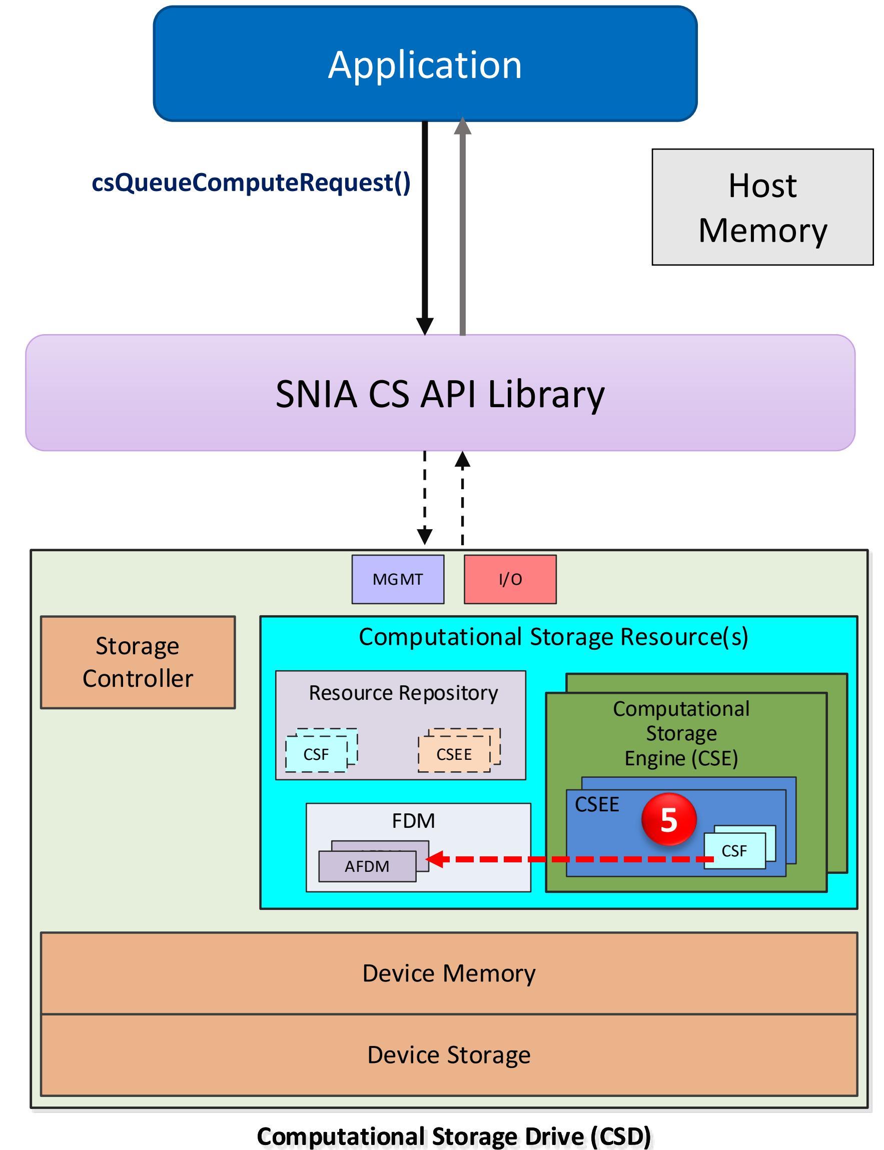


 ©2023 SNIA. All Rights Reserved.

Run: Execute Function

step 5 **Execute CSF**

- Run compute on data loaded in FDM
  - Provide the following
    - CSF to run
    - Parameters to CSF
- More than one Completion options
  - Synchronous
  - Asynchronous
    - Callback or Event



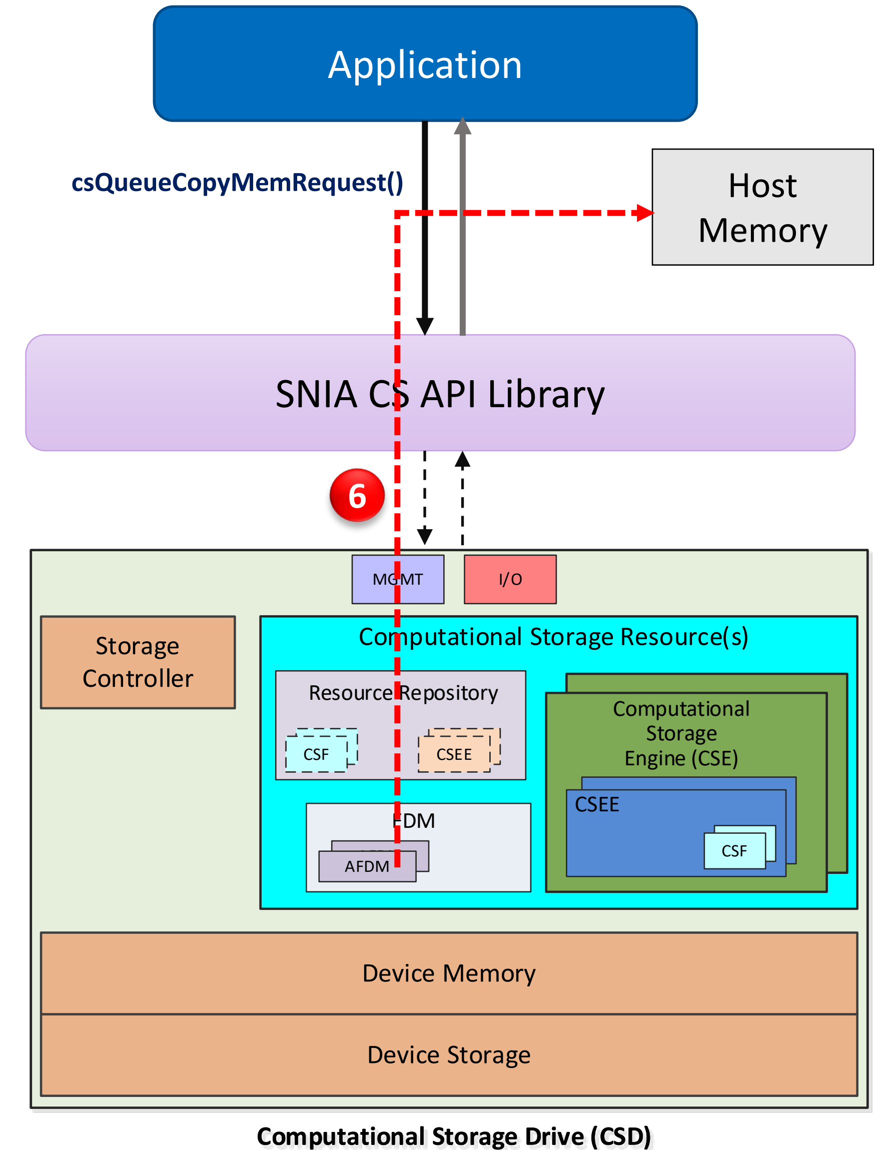
 ©2023 SNIA. All Rights Reserved.

SDC 23

Run: Copy Results

step **6** **Copy FDM contents to Host**

- Copy Results from FDM to Host
- More than one Completion options
  - Synchronous
  - Asynchronous
    - Callback or Event



 ©2023 SNIA. All Rights Reserved.

SDC 23

Run: Code

4

```
// Populate storage request with data from file
csStorageRequest storReq = malloc(sizeof(CsStorageRequest));
if (!storReq) { ERROR_OUT("not enough memory!\n"); }

storReq->Mode = CS_STORAGE_FILE_IO;
storReq->DevHandle = devHandle;
storReq->u.CsFileIo.Type = CS_STORAGE_LOAD_TYPE;
storReq->u.CsFileIo.FileHandle = fd;           // file fd to access for data
storReq->u.CsFileIo.Offset = 0;                // data offset within file
storReq->u.CsFileIo.Bytes = CHUNK_SIZE;
storReq->u.CsFileIo.DevMem.MemHandle = afdmHandle1;
storReq->u.CsFileIo.DevMem.ByteOffset = 0;
status = csQueueStorageRequest(storReq, storReq, NULL, NULL, NULL, &compVal);
if (status != CS_SUCCESS)
    ERROR_OUT("Could not load storage data\n");
```

Callback for completion

Event for completion

6

```
// Populate copy request for results data
CsCopyMemRequest copyReq = malloc(sizeof(CsCopyMemRequest));
if (!copyReq) { ERROR_OUT("memory alloc error\n"); }

copyReq->Type = CS_COPY_FROM_DEVICE;
copyReq->u.HostVAddress = results_buf;
copyReq->DevMem.MemHandle = afdmHandle2;
copyReq->DevMem.ByteOffset = 0;
copyReq->Bytes = CHUNK_SIZE;
status = csQueueCopyMemRequest(copyReq, copyReq, NULL, NULL, NULL,
                               &compVal);
if (status != CS_SUCCESS)
    ERROR_OUT("Could not copy data from FDM\n");
```

5

```
// Populate compute request on data loaded from file
CsComputeRequest compReq = malloc(sizeof(CsComputeRequest) + (sizeof(CsComputeArg) * 3));
if (!compReq) { ERROR_OUT("memory alloc error\n"); }

compReq->CSFId = myCSF->CSFId;                 // filter function identifier
compReq->NumArgs = 3;                          // function accepts 3 parameters

CsComputeArg argPtr = &compReq->Args[0];
csHelperSetComputeArg(&argPtr[0], CS_AFDM_TYPE, afdmHandle1, 0);          // input buffer
csHelperSetComputeArg(&argPtr[1], CS_32BIT_VALUE_TYPE, CHUNK_SIZE);       // length of input
csHelperSetComputeArg(&argPtr[2], CS_AFDM_TYPE, afdmHandle2, 0);          // output buffer
status = csQueueComputeRequest(compReq, compReq, NULL, NULL, NULL, &compVal);
if (status != CS_SUCCESS)
    ERROR_OUT("Error in CSF execution\n");
```

Completion Type	Callback Parameter	Event Parameter
Synchronous	NULL	NULL
Asynchronous - Callback	✓	NULL
Asynchronous - Event	NULL	✓

 ©2023 SNIA. All Rights Reserved.

SDC 23

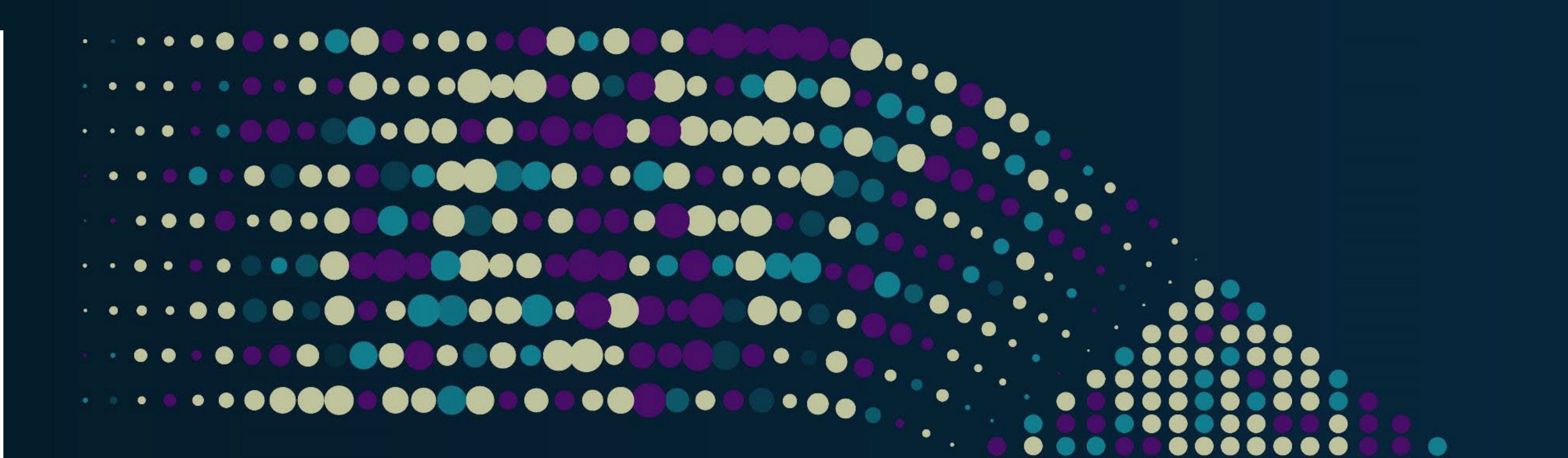
API Summary

6 Steps

1. **Discover Device (*CSx*)**
2. **Discover Function (*CSF*)**
3. **Allocate Device Memory (*FDM*)**
4. **Load Storage Data (*P2P*)**
5. **Execute Function (*CSF*)**
6. **Copy Results (*FDM contents to Host*)**

 ©2023 SNIA. All Rights Reserved.

SDC 23

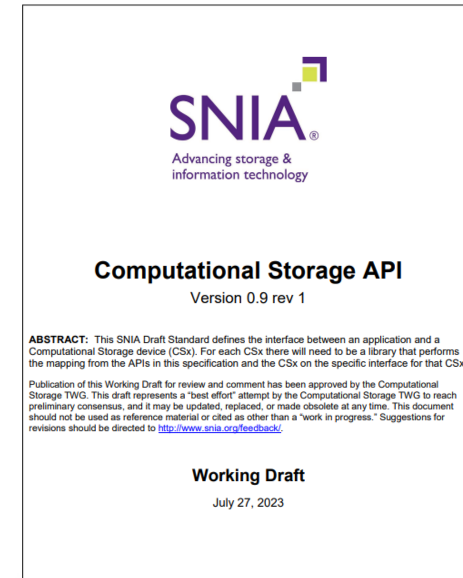


Specification Update

©2023 SNIA. All Rights Reserved.

Specification Update

- Multiple Updates
  - Include LBA Ranges
  - Advanced Device Memory usage
    - Device Memory Pools and Compute Proximity
    - Initialization Options
  - Cancelling I/O, Abort & Reset
  - Compute Function updates
    - Global Identifiers
  - NVMe Support
  - Configuration & Download updates
- SNIA CS API v0.9r1 Specification
  - Public review available
  - In SNIA membership vote towards v1.0



SNIA
Advancing storage & information technology

Computational Storage API

Version 0.9 rev 1

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.

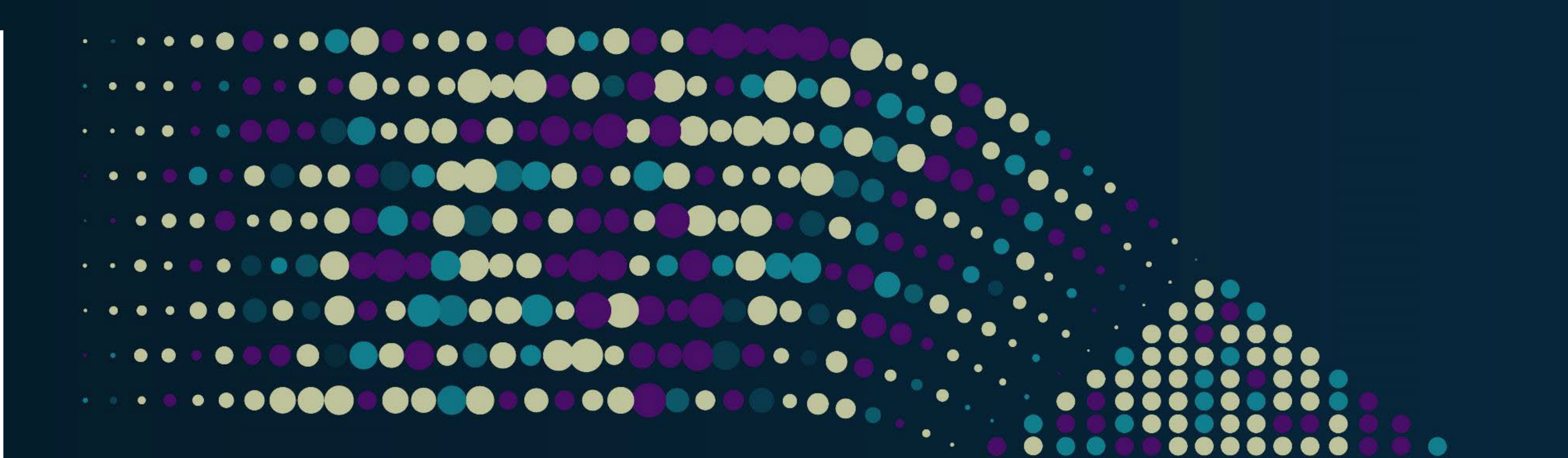
Publication of this Working Draft for review and comment has been approved by the Computational Storage TWG. This draft represents a "best effort" attempt by the Computational Storage TWG to reach preliminary consensus, and it may be updated, replaced, or made obsolete at any time. This document should 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/.

Working Draft

July 27, 2023

©2023 SNIA. All Rights Reserved.

SDC 23



Summary

©2023 SNIA. All Rights Reserved.

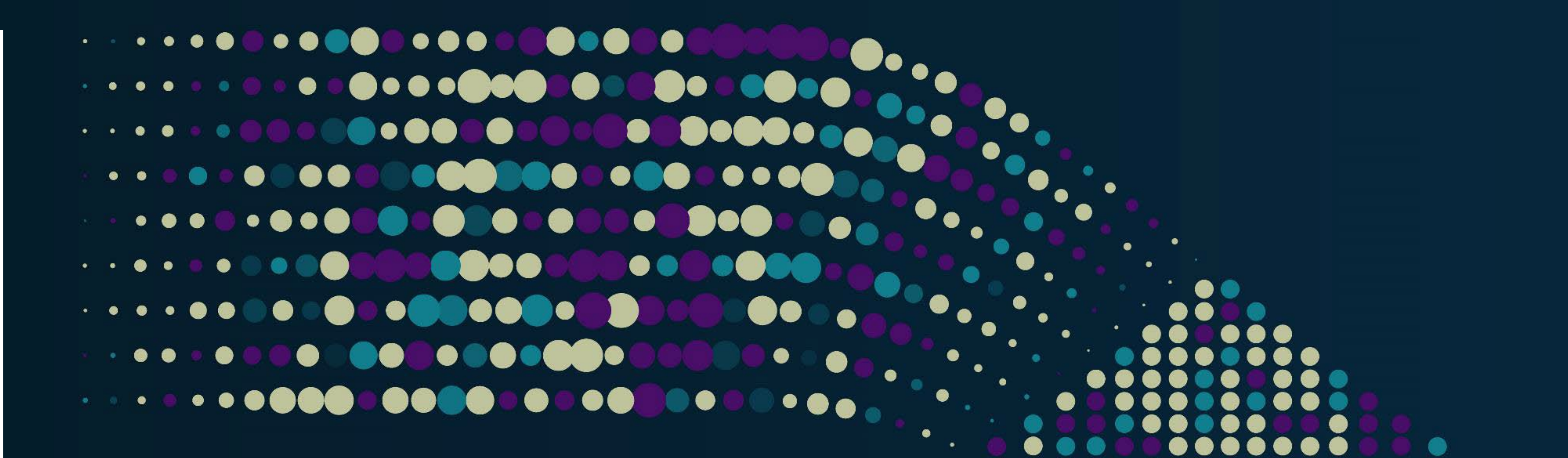
SDC 23

Summary

- Growing Compute requirements are redefining how hardware is used
- Computational Storage and APIs help build these solutions
- SNIA CS APIs v0.9r1 Specification Completed
  - Available for Public Review
- Simplified Programming Interface for CS
- Addresses NVMe CS Architecture
- Minimal steps to adopt

 ©2023 SNIA. All Rights Reserved.

SDC 23



Backup

©2023 SNIA. All Rights Reserved.

SDC 23

Complete Code

1

```
// Find my CSx near storage
status = csGetCSxFromPath(“my_file_path", &length, &csxBuffer);
if (status != CS_SUCCESS)
    ERROR_OUT("No CSx device found!\n");
// open device, init function and prealloc buffers
status = csOpenCSx(csxBuffer, devContext, &devHandle);
if (status != CS_SUCCESS)
    ERROR_OUT("Could not access CSx\n");
```

2

```
// Get access to the CSF to run
status = csGetCSFId(devHandle, “filter”, 0, &infoLength, &count, &csfInfo);
if (status != CS_SUCCESS)
    ERROR_OUT("CSX does not contain any decrypt CSFs \n");
// pick highest performant CSF
CSFIdInfo *p = csfInfo;
CSFIdInfo *myCSF = NULL;
for (i=0; i< count; i++, p++) {
    if ((myCSF == NULL) ||
        ((myCSF != NULL) && (p->RelativePerformance > myCSF->RelativePerformance))) {
        myCSF = p;
    }
}
```

3

```
// Pick most performant FDM from CSFIdInfo
FDMAccess *p = myCSF->FDMList;
FDMAccess *myFDM = NULL;
for (i = 0; i < myCSF->NumFDMs; i++, p++) {
    if ((myFDM == NULL) ||
        ((myFDM != NULL) && (p->RelativePerformance > myFDM->RelativePerformance))) {
        myFDM = p;
    }
}
// allocate FDM for CSF usage
CsMemFlags f;
f.s->FDMId = myFDM->FDMId;
f.s->Flags = 0;   // may also be CS_FDM_CLEAR
status = csAllocMem(devHandle, CHUNK_SIZE, &f, &afdmHandle1, NULL);
if (status != CS_SUCCESS)
    ERROR_OUT("AFDM alloc error\n");
```

4

```
// Populate storage request with data from file
csStorageRequest storReq = malloc(sizeof(CsStorageRequest));
if (!storReq) { ERROR_OUT(“not enough memory!\n"); }
storReq->Mode = CS_STORAGE_FILE_IO;
storReq->DevHandle = devHandle;
storReq->u.CsFileIo.Type = CS_STORAGE_LOAD_TYPE;
storReq->u.CsFileIo.FileHandle = fd;                      // file fd to access for data
storReq->u.CsFileIo.Offset = 0;                           // data offset within file
storReq->u.CsFileIo.Bytes = CHUNK_SIZE;
storReq->u.CsFileIo.DevMem.MemHandle = afdmHandle1;
storReq->u.CsFileIo.DevMem.ByteOffset = 0;
status = csQueueStorageRequest(storReq, storReq, NULL, NULL, NULL, &compVal);
if (status != CS_SUCCESS)
    ERROR_OUT("Could not load storage data\n");
```

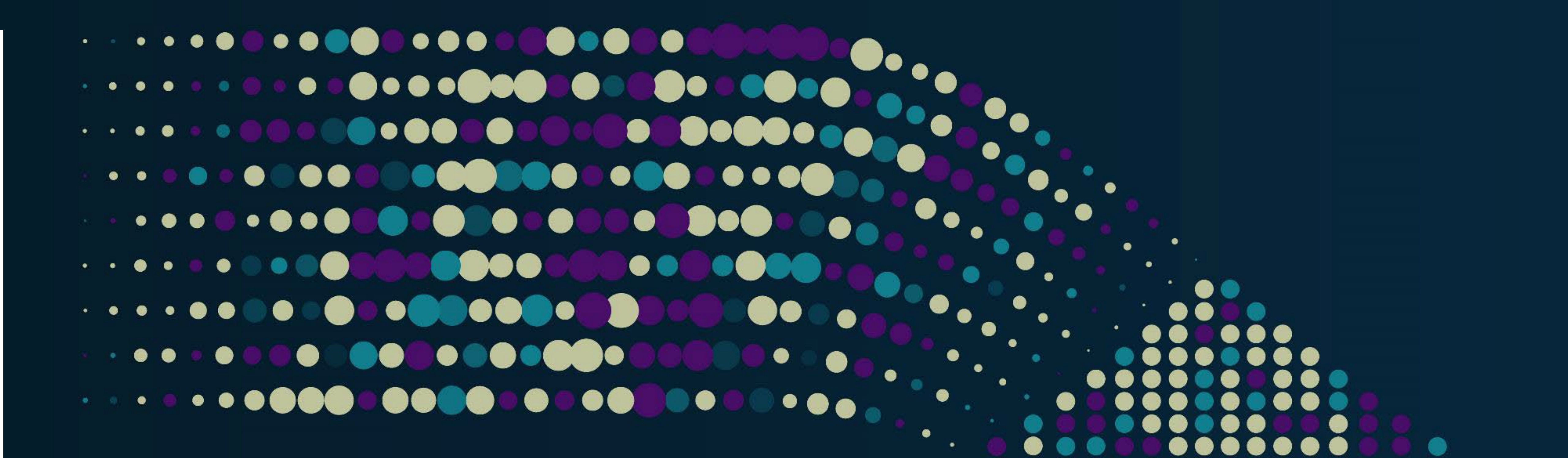
5

```
// Populate compute request on data loaded from file
CsComputeRequest compReq = malloc(sizeof(CsComputeRequest) + (sizeof(CsComputeArg) * 3));
if (!compReq) { ERROR_OUT("memory alloc error\n"); }
compReq->CSFId = myCSF->CSFId;                            // filter function id
compReq->NumArgs = 3;                                     // takes 3 arguments

CsComputeArg argPtr = &compReq->Args[0];
csHelperSetComputeArg(&argPtr[0], CS_AFDM_TYPE, afdmHandle1, 0);        // input buffer
csHelperSetComputeArg(&argPtr[1], CS_32BIT_VALUE_TYPE, CHUNK_SIZE);     // length of input
csHelperSetComputeArg(&argPtr[2], CS_AFDM_TYPE, afdmHandle2, 0);        // output buffer
status = csQueueComputeRequest(compReq, compReq, NULL, NULL, NULL, &compVal);
if (status != CS_SUCCESS)
    ERROR_OUT(“Error in CSF execution\n");
```

6

```
// Populate copy request for results data
CsCopyMemRequest copyReq = malloc(sizeof(CsCopyMemRequest));
if (!copyReq) { ERROR_OUT("memory alloc error\n"); }
copyReq->Type = CS_COPY_FROM_DEVICE;
copyReq->u.HostVAddress = results_buf;
copyReq->DevMem.MemHandle = afdmHandle2;
copyReq->DevMem.ByteOffset = 0;
copyReq->Bytes = CHUNK_SIZE;
status = csQueueCopyMemRequest(copyReq, copyReq, NULL, NULL, NULL, NULL);
if (status != CS_SUCCESS)
    ERROR_OUT(“Could not copy from FDM!\n);
```



Please take a moment to rate this session.

Your feedback is important to us.

 ©2023 SNIA. All Rights Reserved.

SDC 23