Flash Memory Summit

Programming with Computational Storage

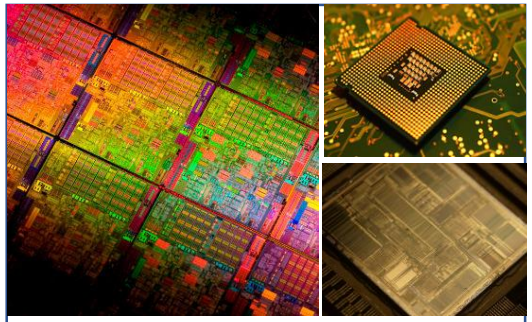
Presenter: Oscar P Pinto

Samsung Semiconductor Inc

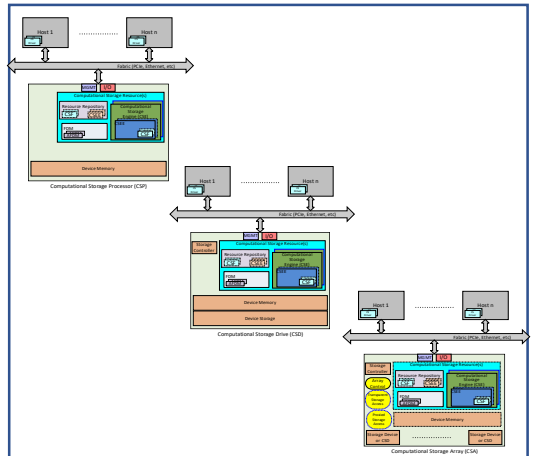
Agenda

- Overview
- Programming Usage
- Applying CS APIs
  - Step by Step Example
- Specification Update
- Summary

Why Computational Storage (CS) APIs?



- Many Compute Interfaces available
- Memory based only



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

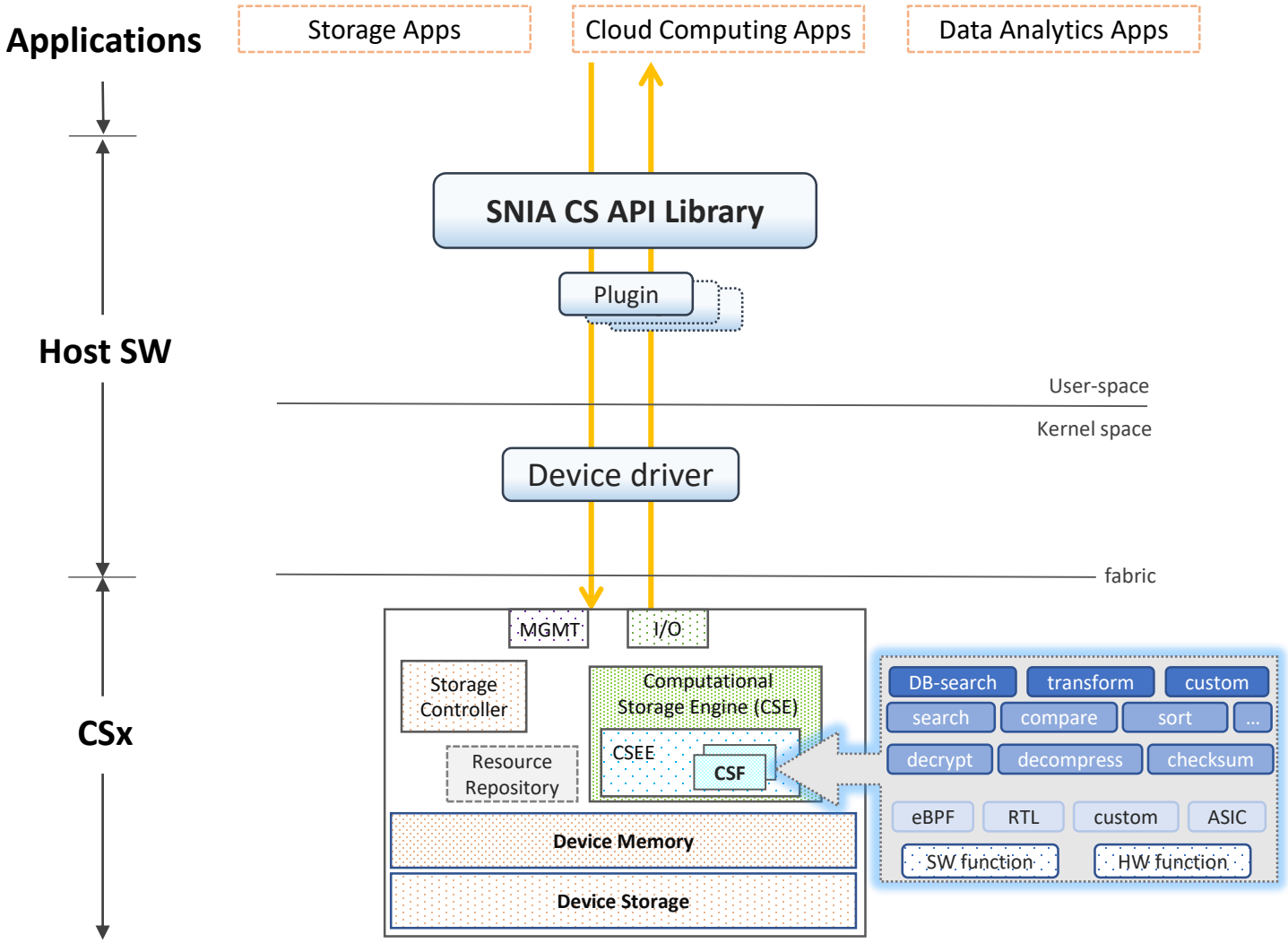
SNIA CS APIs

- Takes near storage compute into account

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

CSF – Computational Storage Function



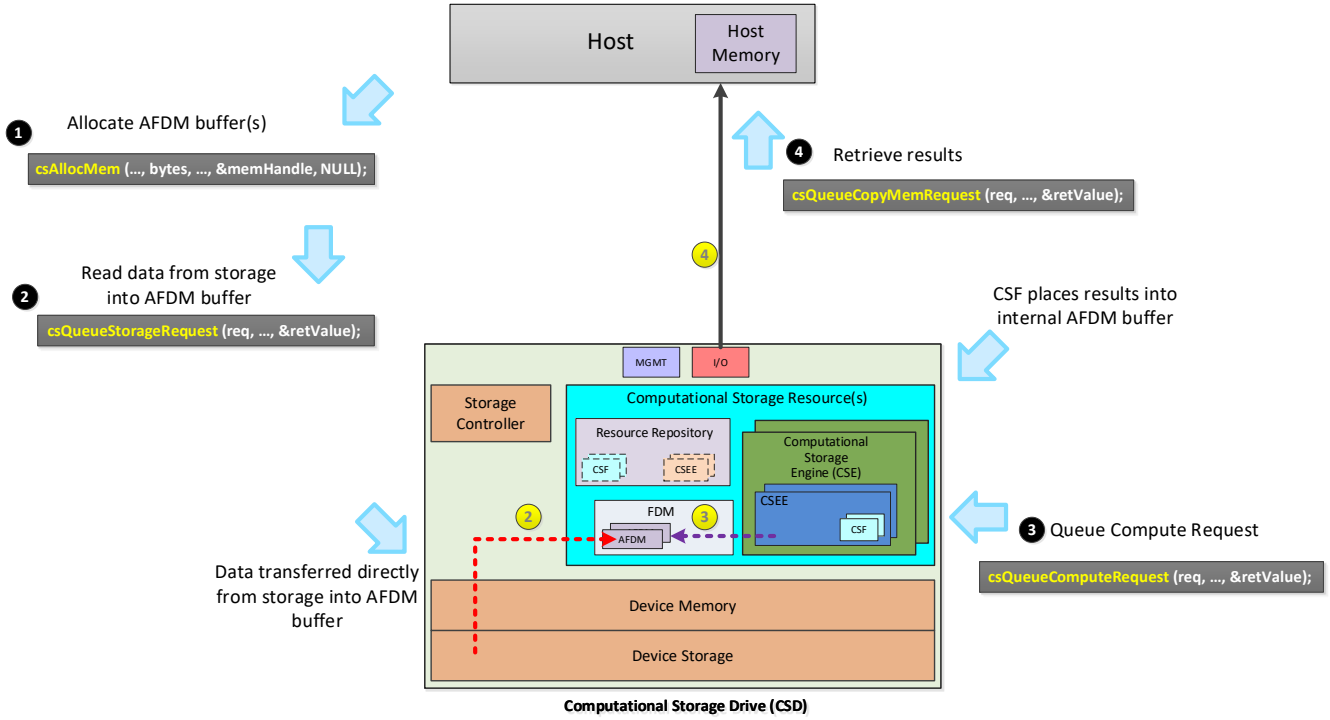
Computational Storage Drive (CSD)

Programming Usage

- 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

Example

- Execute Data Filter CSF
  - ✓Allocate Device Memory (FDM)
  - ✓Load Data from Storage
  - ✓Run Data Filter CSF on loaded Data
  - ✓Copy Results to Host Memory



CSF – Computational Storage Function
FDM – Function Device Memory

Prepare for Computational Storage (Setup)

- Discover by name
- Access device

- Discover the function(s) you want to execute

- Allocate Device Memory

CSx – Computational Storage Device
CSF – Computational Storage Function

Discover Device

Discover CSx

- Discover by name/storage entity
- Access device

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 prealloc buffers
status = csOpenCSx(csxBuffer, &MyDevContext, &devHandle);
if (status != CS_SUCCESS)
    ERROR_OUT("Could not access CSx\n");
```

Access CSx

Discover Function

Flash Memory Summit

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

- Discover your CSF
- Pick the best one if > 1 available

Find CSF by name

Pick most performant in list

```
// Get access to the CSF to run
status = csGetCSFId(devHandle, “filter”, &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);
```

Allocate Device Memory

Allocate FDM

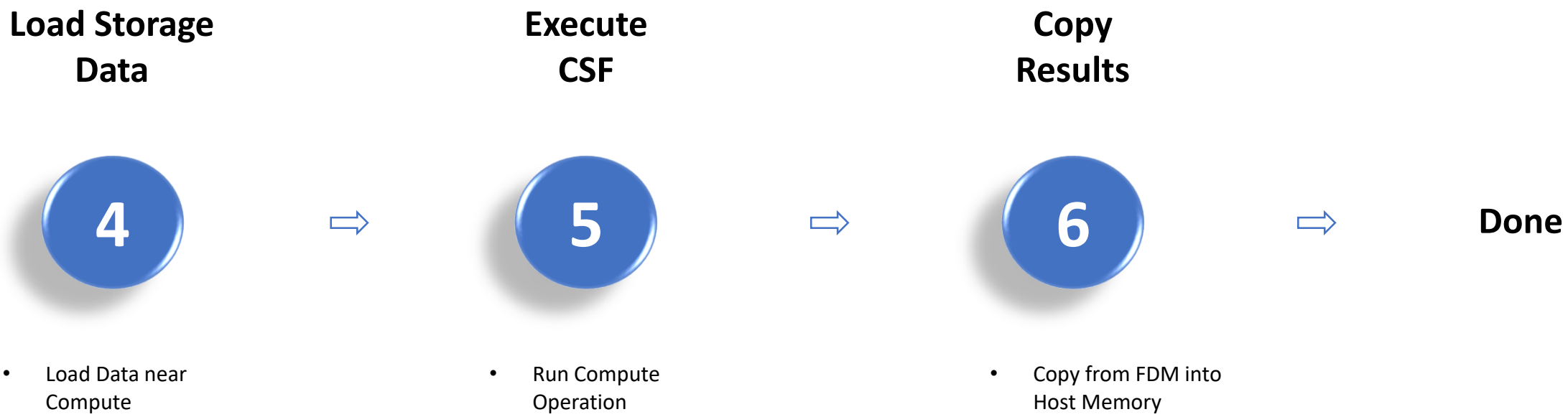
- Pick from list if > 1 available
- Allocate FDM as necessary

```
// 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");
```

Pick most performant FDM in list if > 1

Allocate required size

Perform CS (Run)



Load Storage Data (P2P)

Load Storage Data

- Load data from storage into FDM
- Data does not leave the device

```
// 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");
```

Event for completion

Callback for completion

Synchronous: NULL, NULL
Asynchronous: Callback / Event

Execute Function

Flash Memory Summit

step **5** **Execute CSF**

- Run compute on loaded data

```
// 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");
```

Event for completion

Callback for completion

Synchronous: NULL, NULL
Asynchronous: Callback / Event

Copy Results

Flash Memory Summit

step 6

Copy FDM contents to Host

- Copy Data from Device Memory to Host

```
// 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 data from FDM\n");
```

Event for completion

Callback for completion

Synchronous: NULL, NULL
Asynchronous: Callback / Event

Usage Summary

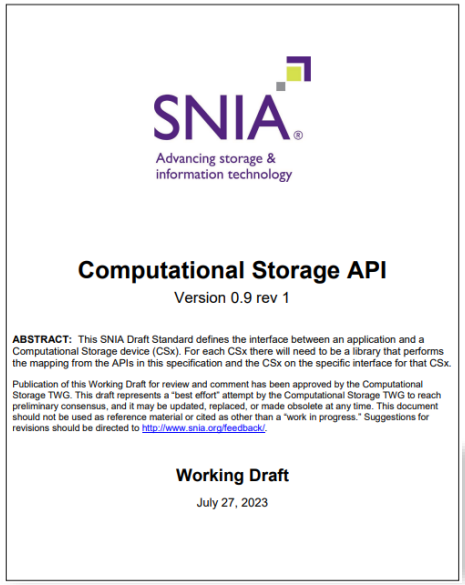
6 Steps

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*)

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



Summary

- SNIA CS APIs v0.9r1 Specification Completed
  - Available for Public Review
- Simplified Programming Interface for CS
- Addresses NVMe CS Architecture
- Minimal steps to adopt

Backup

The Complete Code


```
// 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, &MyDevContext, &devHandle);
if (status != CS_SUCCESS)
    ERROR_OUT("Could not access CSx\n");
```

1

```
// Get access to the CSF to run
status = csGetCSFId(devHandle, “filter”, &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;
    }
}
```

2

```
// 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");
```

3

```
// 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");
```

4

```
// 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");
```

5

```
// 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);
```

6