AMD CROSSFIRE DIRECTX® 11 SAMPLE

ALEX KHARLAMOV
ANAS LASRAM

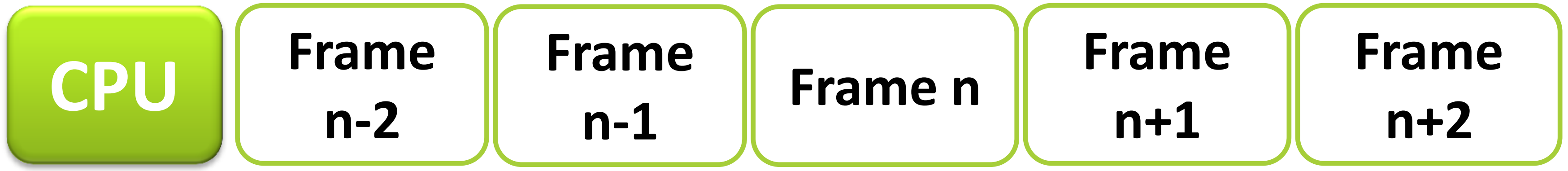
TABLE OF CONTENTS

AMD CROSSFIRE DRIVER

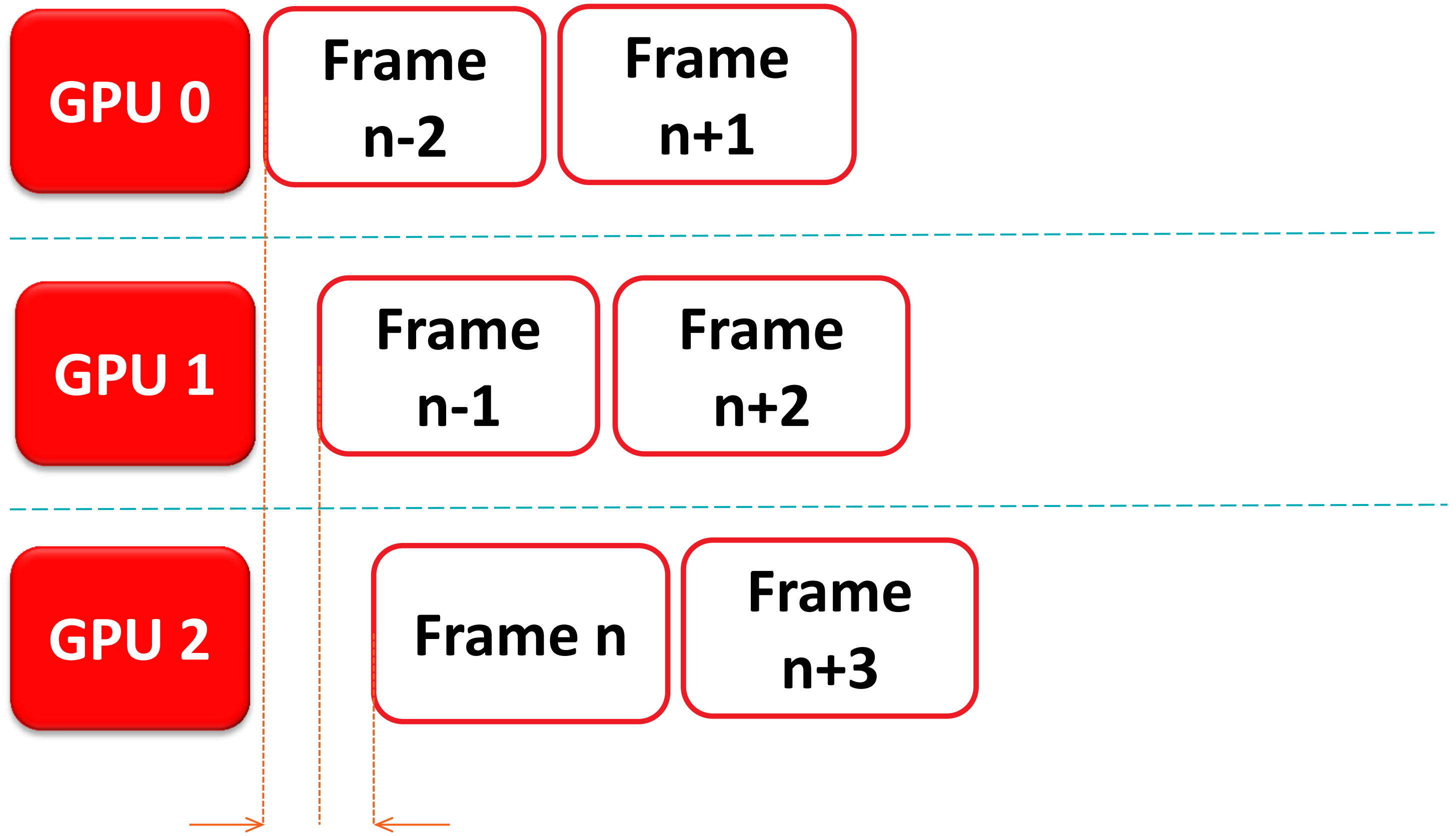
DEFAULT BEHAVIOR

AMD CROSSFIRE

OPTIMAL APPLICATION FRAME SUBMISSION ON MULTI GPU SYSTEMS



Maximize frame rendering overlap between frames for best M-GPU scaling



Lost performance opportunity

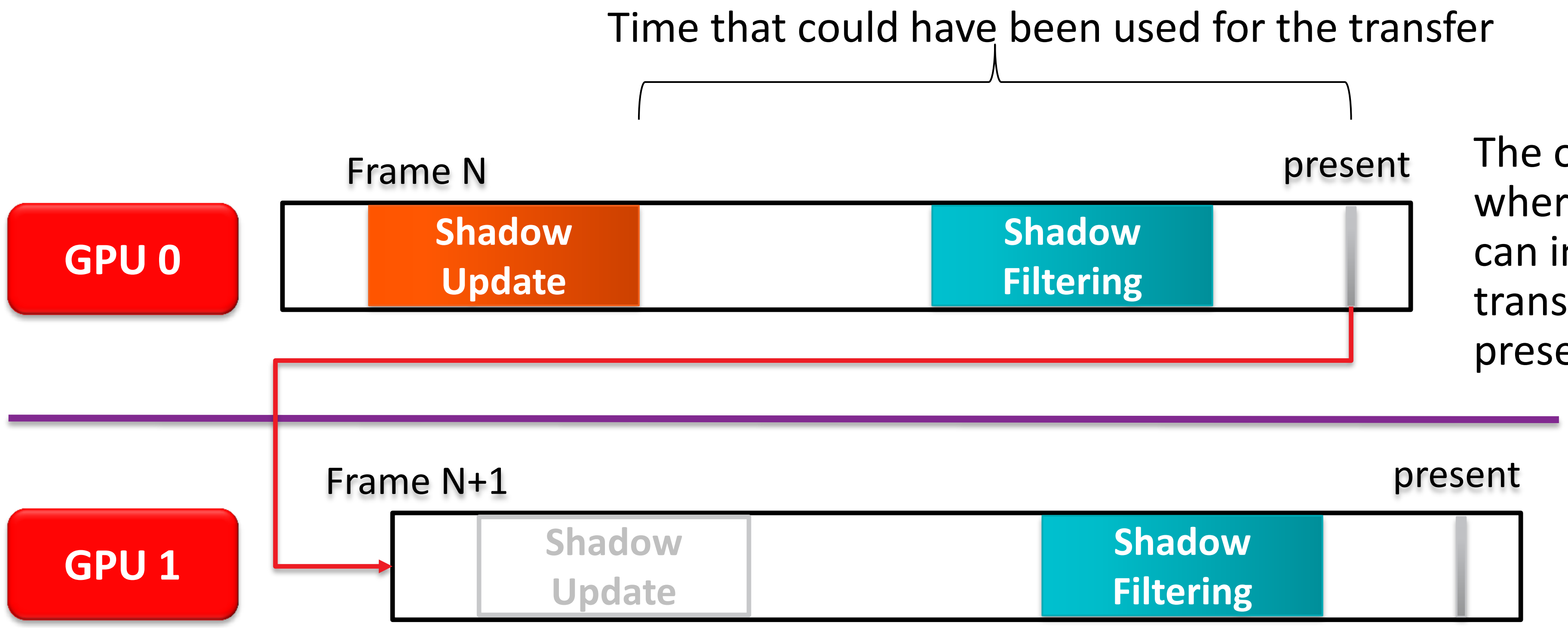
COMPATIBLE-AFR TRANSFER HEURISTICS

- **Heuristic 1**: A resource that is updated before it gets used within a frame is not stale

- **Heuristic 2**: Let N be the number of GPUs in the system. A resource that is updated for N frames in a row before it gets used is not stale

AMD CROSSFIRE: DEFAULT BEHAVIOR

IF RESOURCE IS NOT UPDATED : TRANSFERS



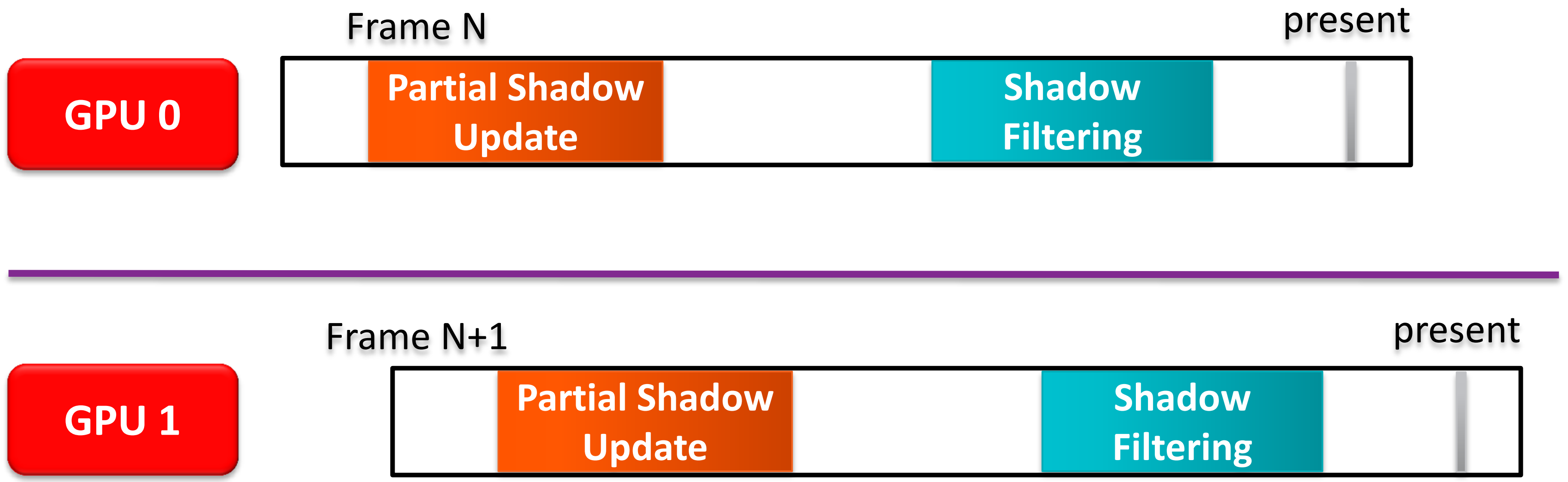
The only place where the driver can initialize a transfer is at present()

Application doesn't update shadows on frame N+1

Application uses a resource that is "stale" driver needs to transfer if from a previous frame

AMD CROSSFIRE: DEFAULT BEHAVIOR

IF RESOURCE IS PARTIALLY UPDATED : CORRUPTION



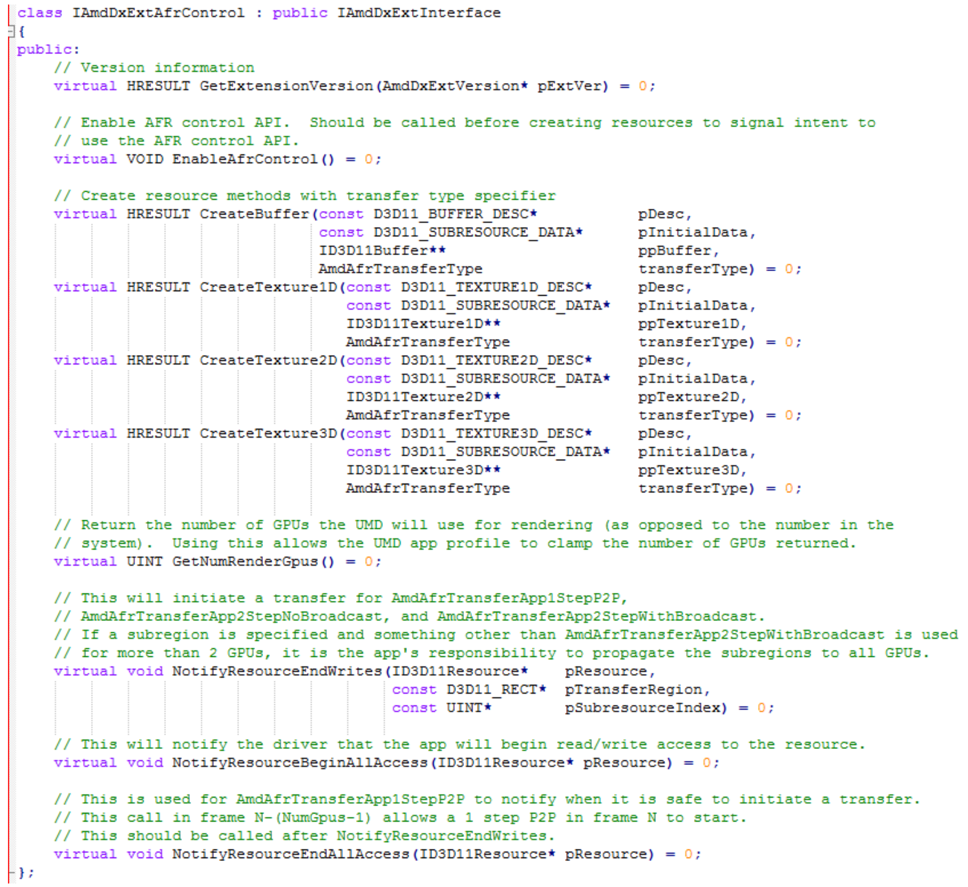
Application performs a partial update on a resource.

Application uses a resource that is partially updated but partially "stale". Driver has no way to tell the difference.

AMD CROSSFIRE API

AMD CROSSFIRE API

- Driver extension headers
  - Plain headers
  - Application is responsible for loading driver DLLs and querying interfaces

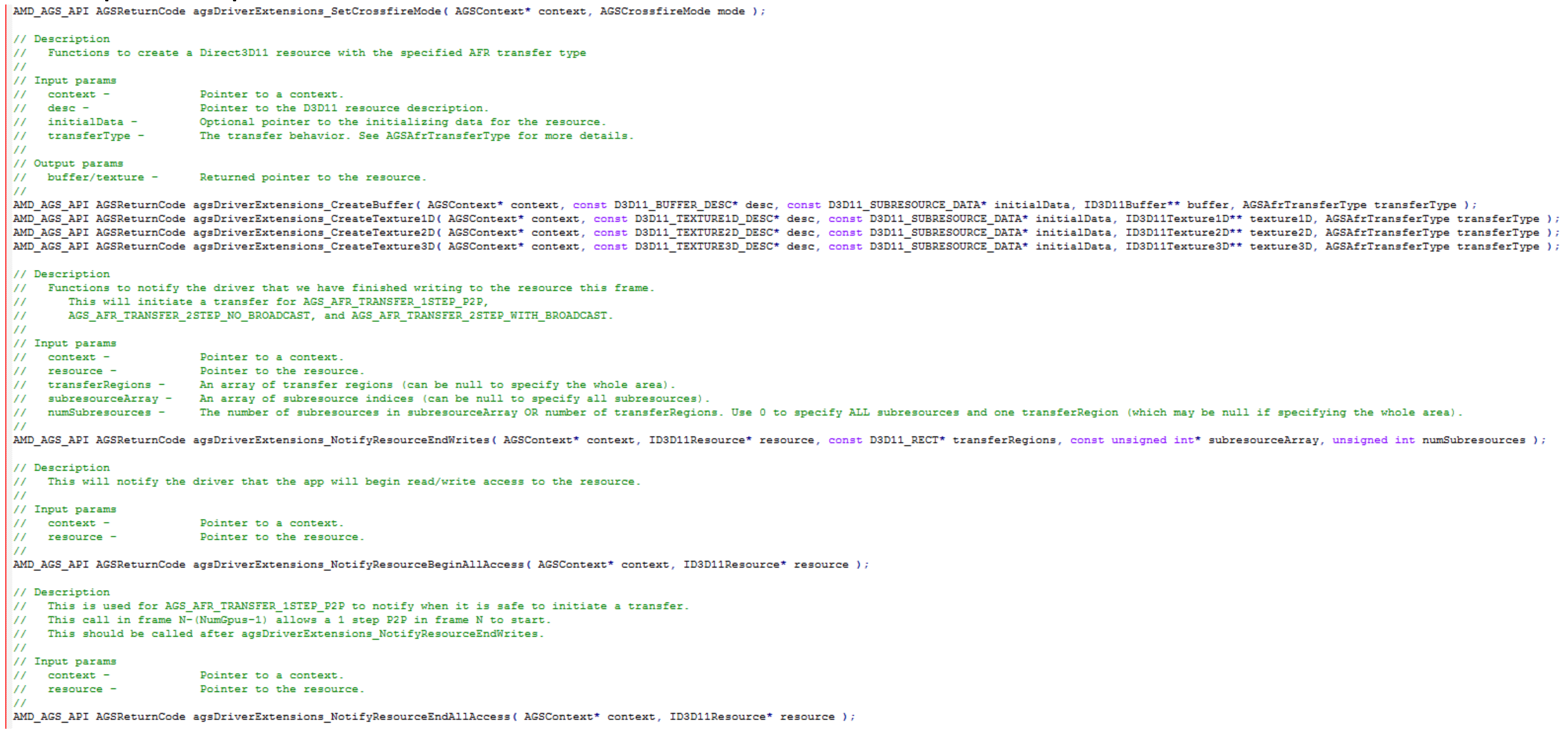


```cpp
class IAmdDxExtAfrControl : public IAmdDxExtInterface
{
public:
    // Version information
    virtual HRESULT GetExtensionVersion(AmdDxExtVersion* pExtVer) = 0;

    // Enable AFR control API.  Should be called before creating resources to signal intent to
    // use the AFR control API.
    virtual VOID EnableAfrControl() = 0;

    // Create resource methods with transfer type specifier
    virtual HRESULT CreateBuffer(const D3D11_BUFFER_DESC*           pDesc,
                                 const D3D11_SUBRESOURCE_DATA*      pInitialData,
                                 ID3D11Buffer**                     ppBuffer,
                                 AmdAfrTransferType                 transferType) = 0;
    virtual HRESULT CreateTexture1D(const D3D11_TEXTURE1D_DESC*     pDesc,
                                    const D3D11_SUBRESOURCE_DATA*   pInitialData,
                                    ID3D11Texture1D**               ppTexture1D,
                                    AmdAfrTransferType              transferType) = 0;
    virtual HRESULT CreateTexture2D(const D3D11_TEXTURE2D_DESC*     pDesc,
                                    const D3D11_SUBRESOURCE_DATA*   pInitialData,
                                    ID3D11Texture2D**               ppTexture2D,
                                    AmdAfrTransferType              transferType) = 0;
    virtual HRESULT CreateTexture3D(const D3D11_TEXTURE3D_DESC*     pDesc,
                                    const D3D11_SUBRESOURCE_DATA*   pInitialData,
                                    ID3D11Texture3D**               ppTexture3D,
                                    AmdAfrTransferType              transferType) = 0;

    // Return the number of GPUs the UMD will use for rendering (as opposed to the number in the
    // system).  Using this allows the UMD app profile to clamp the number of GPUs returned.
    virtual UINT GetNumRenderGpus() = 0;

    // This will initiate a transfer for AmdAfrTransferApp1StepP2P,
    // AmdAfrTransferApp2StepNoBroadcast, and AmdAfrTransferApp2StepWithBroadcast.
    // If a subregion is specified and something other than AmdAfrTransferApp2StepWithBroadcast is used
    // for more than 2 GPUs, it is the app's responsibility to propagate the subregions to all GPUs.
    virtual void NotifyResourceEndWrites(ID3D11Resource*    pResource,
                                         const D3D11_RECT*  pTransferRegion,
                                         const UINT*        pSubresourceIndex) = 0;

    // This will notify the driver that the app will begin read/write access to the resource.
    virtual void NotifyResourceBeginAllAccess(ID3D11Resource* pResource) = 0;

    // This is used for AmdAfrTransferApp1StepP2P to notify when it is safe to initiate a transfer.
    // This call in frame N-(NumGpus-1) allows a 1 step P2P in frame N to start.
    // This should be called after NotifyResourceEndWrites.
    virtual void NotifyResourceEndAllAccess(ID3D11Resource* pResource) = 0;
};
```

AMD CROSSFIRE API

SAMPLES USE AGS LIBRARY

- AGS library
  - ISV developed and maintained library
  - Wraps multiple driver extensions



```
AMD_AGS_API AGSReturnCode agsDriverExtensions_SetCrossfireMode( AGSContext* context, AGSCrossfireMode mode );

// Description
//   Functions to create a Direct3D11 resource with the specified AFR transfer type
//
// Input params
//   context -              Pointer to a context.
//   desc -                 Pointer to the D3D11 resource description.
//   initialData -          Optional pointer to the initializing data for the resource.
//   transferType -         The transfer behavior. See AGSAfrTransferType for more details.
//
// Output params
//   buffer/texture -       Returned pointer to the resource.
//
AMD_AGS_API AGSReturnCode agsDriverExtensions_CreateBuffer( AGSContext* context, const D3D11_BUFFER_DESC* desc, const D3D11_SUBRESOURCE_DATA* initialData, ID3D11Buffer** buffer, AGSAfrTransferType transferType );
AMD_AGS_API AGSReturnCode agsDriverExtensions_CreateTexture1D( AGSContext* context, const D3D11_TEXTURE1D_DESC* desc, const D3D11_SUBRESOURCE_DATA* initialData, ID3D11Texture1D** texture1D, AGSAfrTransferType transferType );
AMD_AGS_API AGSReturnCode agsDriverExtensions_CreateTexture2D( AGSContext* context, const D3D11_TEXTURE2D_DESC* desc, const D3D11_SUBRESOURCE_DATA* initialData, ID3D11Texture2D** texture2D, AGSAfrTransferType transferType );
AMD_AGS_API AGSReturnCode agsDriverExtensions_CreateTexture3D( AGSContext* context, const D3D11_TEXTURE3D_DESC* desc, const D3D11_SUBRESOURCE_DATA* initialData, ID3D11Texture3D** texture3D, AGSAfrTransferType transferType );

// Description
//   Functions to notify the driver that we have finished writing to the resource this frame.
//      This will initiate a transfer for AGS_AFR_TRANSFER_1STEP_P2P,
//      AGS_AFR_TRANSFER_2STEP_NO_BROADCAST, and AGS_AFR_TRANSFER_2STEP_WITH_BROADCAST.
//
// Input params
//   context -              Pointer to a context.
//   resource -             Pointer to the resource.
//   transferRegions -      An array of transfer regions (can be null to specify the whole area).
//   subresourceArray -     An array of subresource indices (can be null to specify all subresources).
//   numSubresources -      The number of subresources in subresourceArray OR number of transferRegions. Use 0 to specify ALL subresources and one transferRegion (which may be null if specifying the whole area).
//
AMD_AGS_API AGSReturnCode agsDriverExtensions_NotifyResourceEndWrites( AGSContext* context, ID3D11Resource* resource, const D3D11_RECT* transferRegions, const unsigned int* subresourceArray, unsigned int numSubresources );

// Description
//   This will notify the driver that the app will begin read/write access to the resource.
//
// Input params
//   context -              Pointer to a context.
//   resource -             Pointer to the resource.
//
AMD_AGS_API AGSReturnCode agsDriverExtensions_NotifyResourceBeginAllAccess( AGSContext* context, ID3D11Resource* resource );

// Description
//   This is used for AGS_AFR_TRANSFER_1STEP_P2P to notify when it is safe to initiate a transfer.
//   This call in frame N-(NumGpus-1) allows a 1 step P2P in frame N to start.
//   This should be called after agsDriverExtensions_NotifyResourceEndWrites.
//
// Input params
//   context -              Pointer to a context.
//   resource -             Pointer to the resource.
//
AMD_AGS_API AGSReturnCode agsDriverExtensions_NotifyResourceEndAllAccess( AGSContext* context, ID3D11Resource* resource );
```

AMD CROSSFIRE API

Resource Creation Flag	Description
DEFAULT	Behavior depends on Radeon Settings Crossfire mode
DISABLE	Turn off driver resource tracking
1STEP_P2P	app controlled GPU to next GPU transfer
2STEP_NO_BROADCAST	app controlled GPU to next GPU transfer using intermediate system memory
2STEP_WITH_BROADCAST	app controlled GPU to all render GPUs transfer using intermediate system memory

Radeon Settings CrossFire mode	Implicit control: The CrossFire API is not queried by the application	Explicit control: The CrossFire API is successfully queried by the application
Disabled	The application runs in single GPU mode	The CrossFire API cannot be queried. The application runs in single GPU mode
Default mode	If the application has a driver profile it will be used. If a driver profile does not exist the application will run in single GPU mode	If the application has a driver profile it will be used. If a driver profile does not exist, the CrossFire API cannot be queried and the application will run in single GPU mode
AFR friendly	The application runs in MGPU mode but resource tracking is disabled	Resource transfers are only applied when the application notifies the driver to do so. Resources created with the flag AFR_TRANSFER_DEFAULT or through the normal D3D11 API will not be tracked by the driver
Optimize 1x1	The application runs in MGPU mode and resources will be tracked by the driver and will be rendered on each GPU if their resolution is 1x1	Resource transfers are applied when the application notifies the driver to do so. Resources created with the flag AFR_TRANSFER_DEFAULT or through the normal D3D11 API will be tracked by the driver.Resources with dimensions of 1x1 will be rendered on each GPU.
AFR Compatible	The application runs in MGPU mode and resource tracking is enabled	Resource transfers are applied when the application notifies the driver to do so. Resources created with the flag AFR_TRANSFER_DEFAULT or through the normal D3D11 API will be tracked by the driver
Use AMD pre-defined profile	The mode allows using one of the existing driver profiles and applying it to the current application	The mode allows using one of the existing driver profiles and applying it to the current application

AMD CROSSFIRE API

IF RESOURCE IS PARTIALLY UPDATED

Signal **BeginAllAccess**()

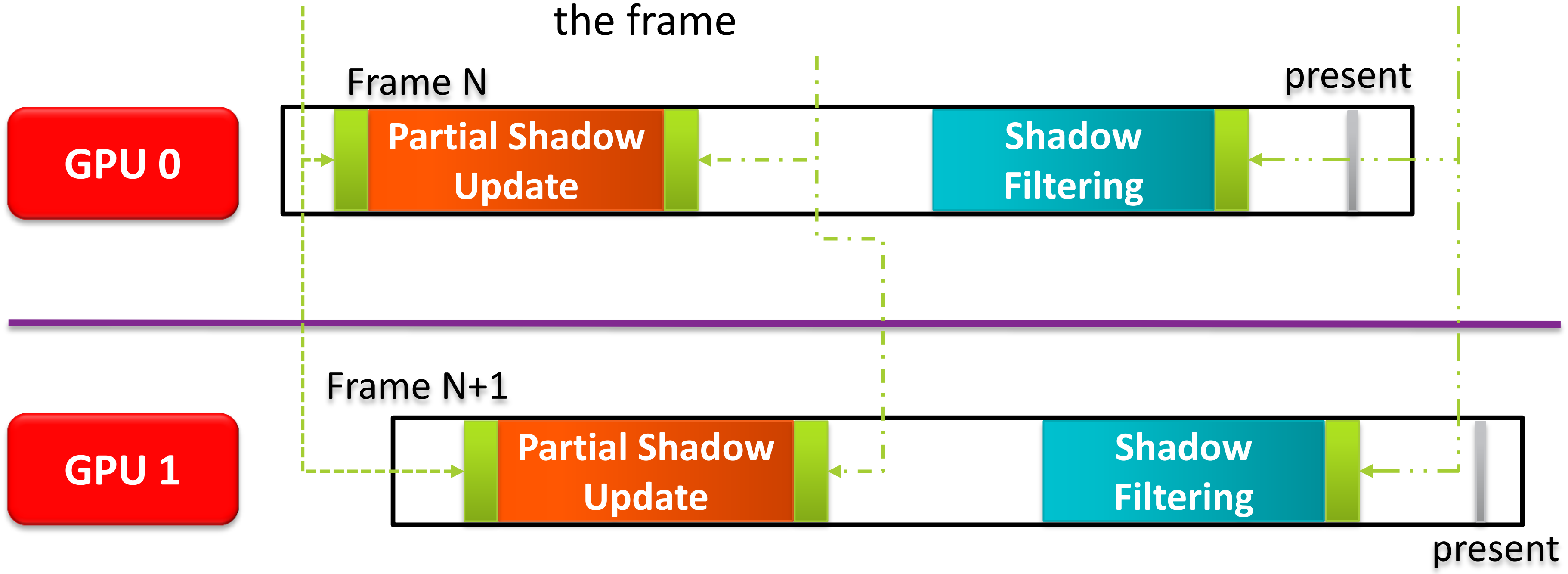
Before the first use of a resource (read or write)

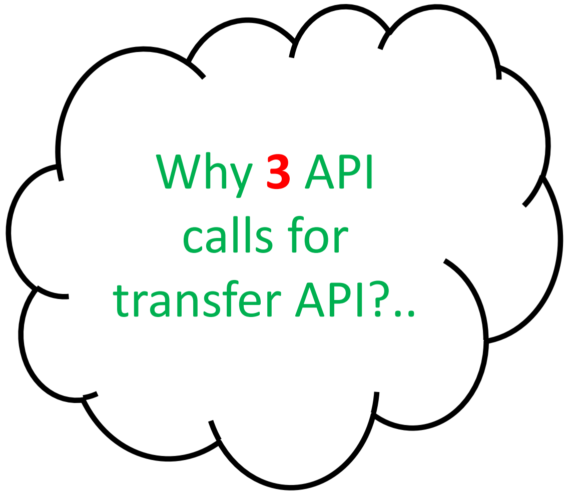
Signal **EndWrite**()

After the resource has been modified for the last time in the frame

Signal **EndAllAccess**()

After the resource has been used for the last time within a frame





AMD CROSSFIRE API

IF RESOURCE IS PARTIALLY UPDATED

Signal **BeginAllAccess()**

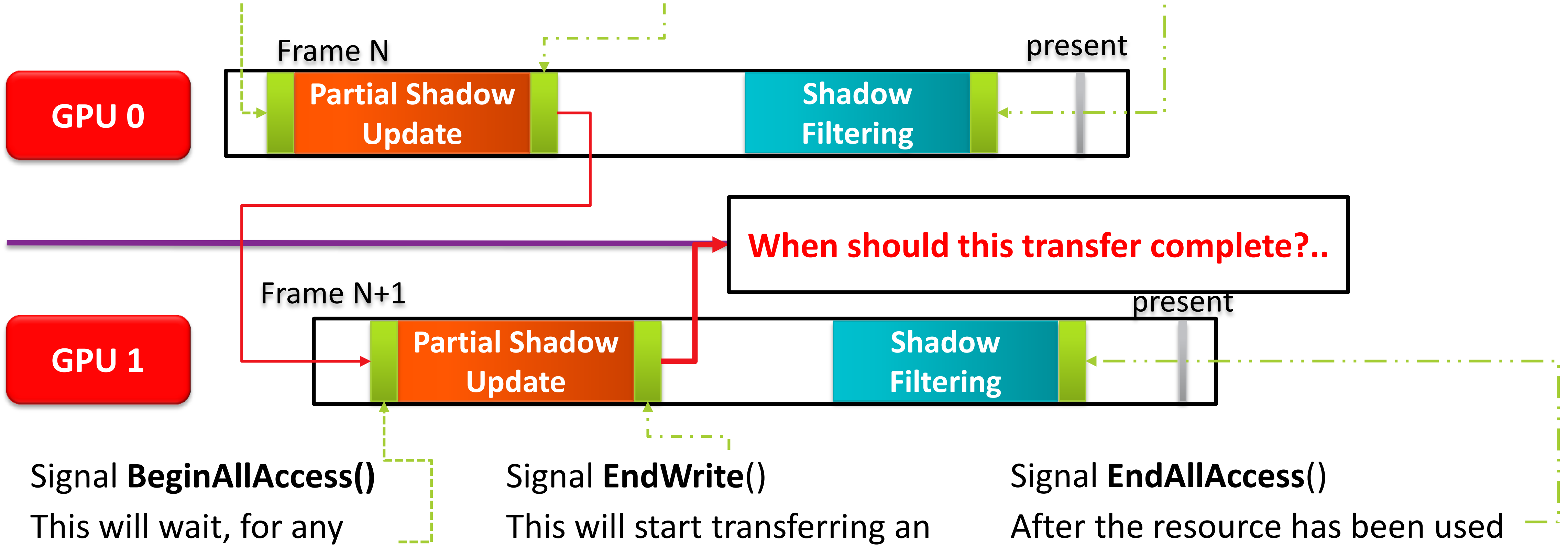
This will wait, for any incoming transfers from frame **N-1**

Signal **EndWrite**()

This will start transferring an updated resource from GPU 0 to GPU 1 for frame **N+1**

Signal **EndAllAccess**()

after the resource has been used for the last time within a frame



Signal **BeginAllAccess()**

This will wait, for any incoming transfers from frame **N**

Signal **EndWrite**()

This will start transferring an updated resource from GPU 1 to GPU 0 for frame **N+2**

Signal **EndAllAccess**()

After the resource has been used for the last time within a frame

AMD CROSSFIRE API

IF RESOURCE IS PARTIALLY UPDATED

Signal **BeginAllAccess()**

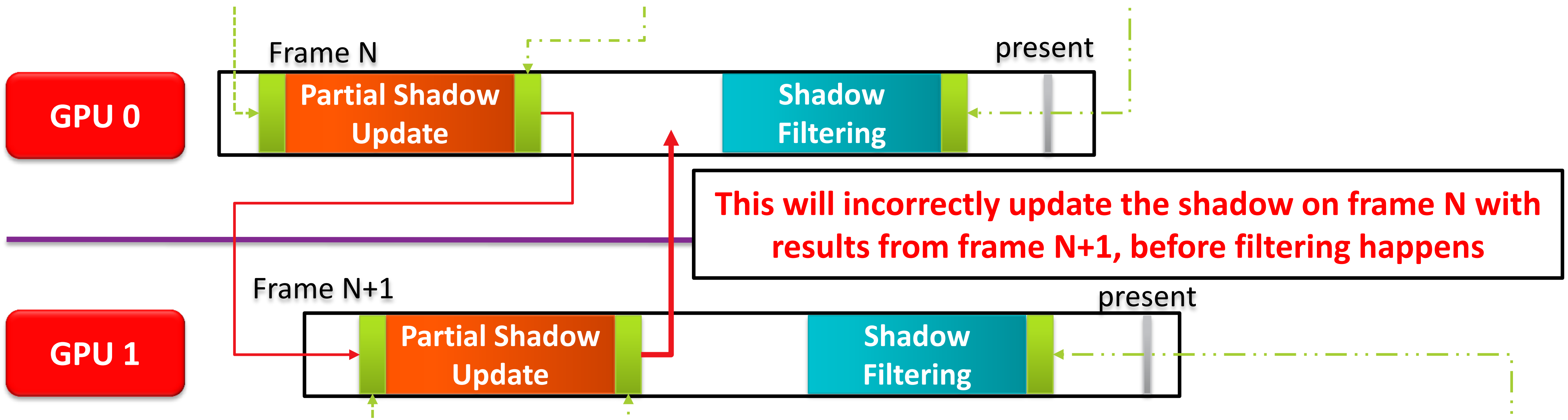
This will wait, for any incoming transfers from frame **N-1**

Signal **EndWrite**()

This will start transferring an updated resource from GPU 0 to GPU 1 for frame **N+1**

Signal **EndAllAccess**()

After the resource has been used for the last time within a frame



This will incorrectly update the shadow on frame N with results from frame N+1, before filtering happens

Signal **BeginAllAccess()**

This will wait, for any incoming transfers from frame **N**

Signal **EndWrite**()

This will start transferring an updated resource from GPU 1 to GPU 0 for frame **N+2**

Signal **EndAllAccess**()

After the resource has been used for the last time within a frame

AMD CROSSFIRE API

IF RESOURCE IS PARTIALLY UPDATED

Signal **BeginAllAccess()**

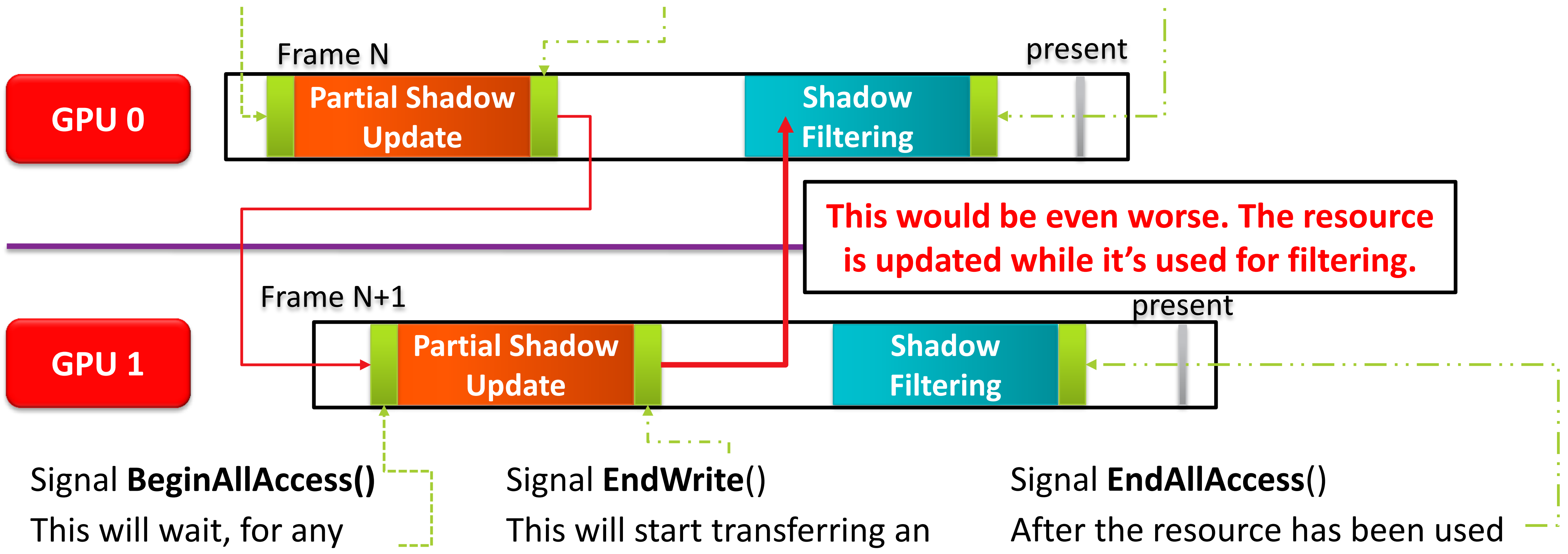
This will wait, for any incoming transfers from frame **N-1**

Signal **EndWrite**()

This will start transferring an updated resource from GPU 0 to GPU 1 for frame **N+1**

Signal **EndAllAccess**()

After the resource has been used for the last time within a frame



This would be even worse. The resource is updated while it's used for filtering.

Signal **BeginAllAccess()**

This will wait, for any incoming transfers from frame **N**

Signal **EndWrite**()

This will start transferring an updated resource from GPU 1 to GPU 0 for frame **N+2**

Signal **EndAllAccess**()

After the resource has been used for the last time within a frame

AMD CROSSFIRE API

IF RESOURCE IS PARTIALLY UPDATED

Signal **BeginAllAccess()**

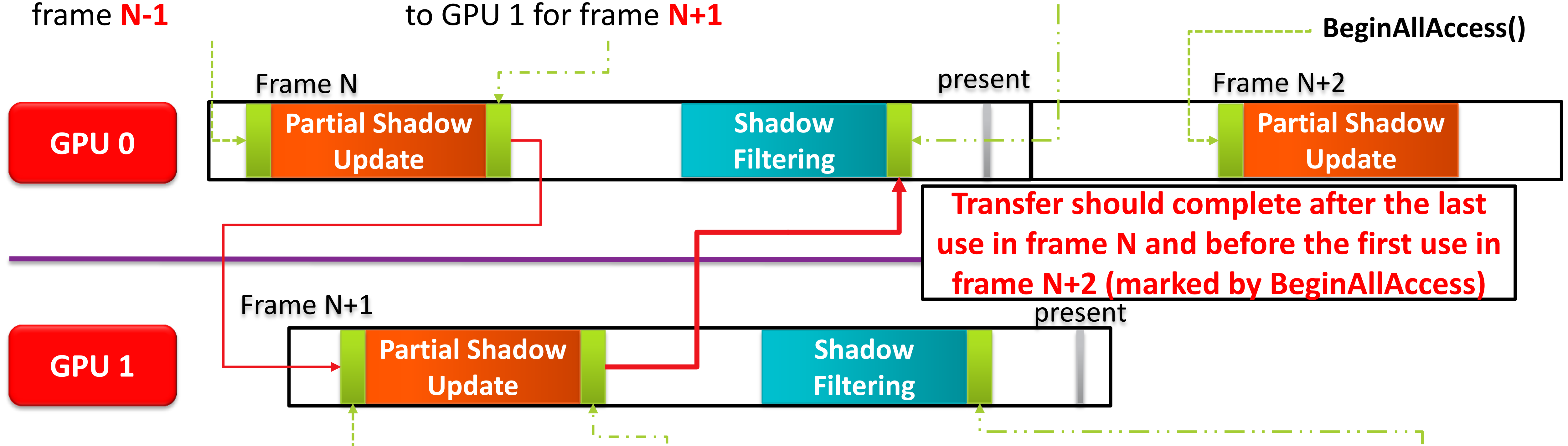
This will wait, for any incoming transfers from frame **N-1**

Signal **EndWrite**()

This will start transferring an updated resource from GPU 0 to GPU 1 for frame **N+1**

Signal **EndAllAccess**()

After the resource has been used for the last time within a frame



Transfer should complete after the last use in frame N and before the first use in frame N+2 (marked by BeginAllAccess)

Signal **BeginAllAccess()**

This will wait, for any incoming transfers from frame **N**

Signal **EndWrite**()

This will start transferring an updated resource from GPU 1 to GPU 0 for frame **N+2**

Signal **EndAllAccess**()

After the resource has been used for the last time within a frame

AMD CROSSFIRE API

- EndWrite – starts a transfer
  - This is optional, if a resource isn’t updated within a frame don’t initiate transfers
- BeginAllAccess – waits on any previous transfers to complete
- EndAllAccess – gates any incoming transfers until resource can be safely updated
- Time to perform a resource transfer is from EndAllAccess until the next BeginAllAccess
  - **>=** (from Present until next BeginAllAccess)
  - EndAllAccess can be naively called at Present()
  - If EndAllAccess is placed as early as the engine allows this maximizes available time for transfer

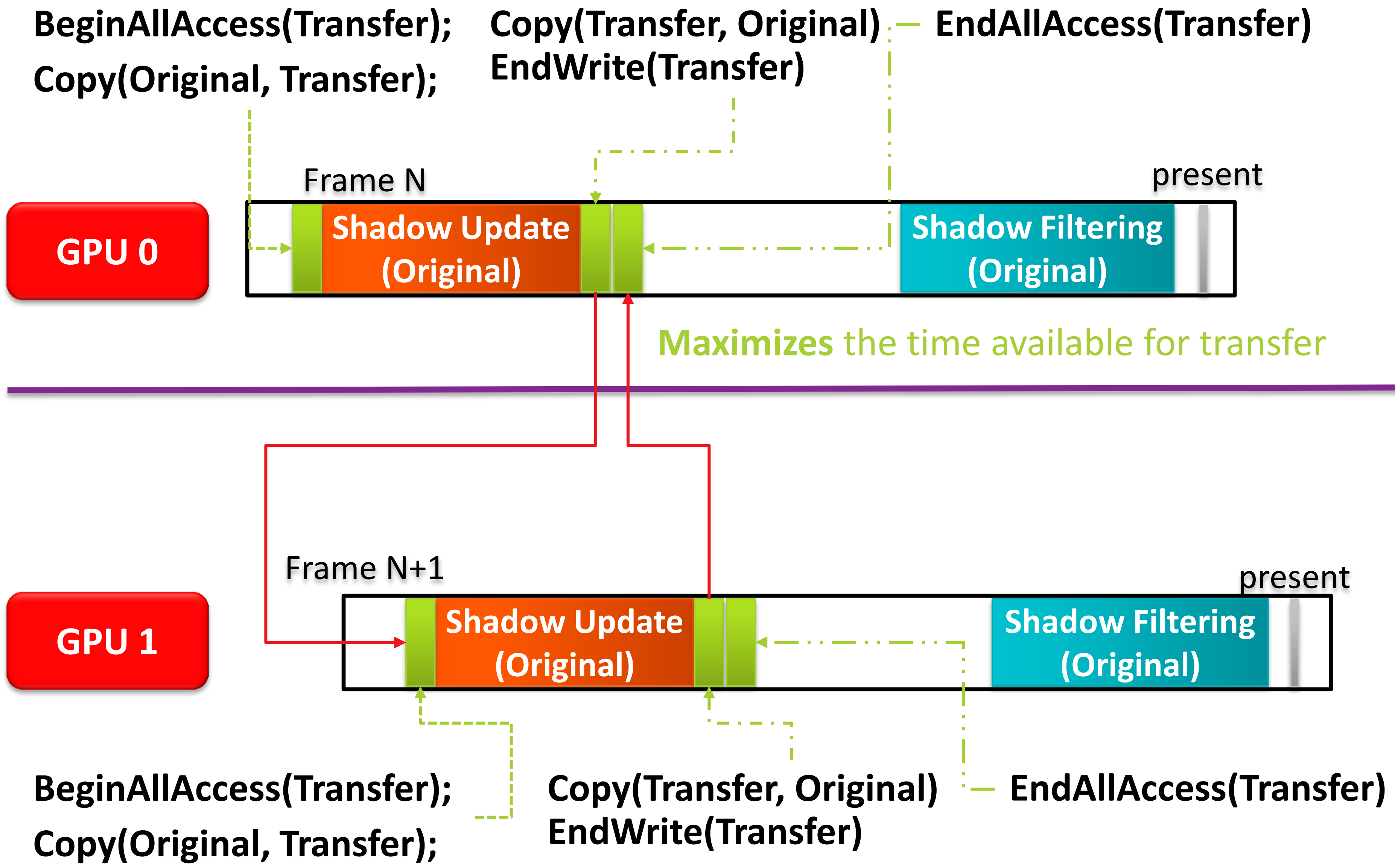
AMD CROSSFIRE API

TRANSFER_2STEP_GPU

- Transfer API has a clear dependency on how the resource is used
  - 2 STEP GPU TRANSFER is an optimization that allows to remove dependencies
  - Also simplifies CFX API integration
- General idea:
  - Given a resource that is updated irregularly
    - it is referred to as the “Original” resource
  - Create “Original” resource with DISABLE transfer flag
  - Create a copy of that resource (same format and size) with applicable transfer flag (1STEP, 2STEP or 2STEP_BROADCAST)
    - It is referred to as the “Transferred” resource
  - On frame *N* when “Original” needs to be updated:
    - BeginAllAccess(“Transferred”) → if there were any updates from previous frames, wait for them
    - Copy(“Original”, Transferred”) → if there were any updates from previous frames, they need to be copied into “Original” now
    - Update(“Original”)
    - Copy(“Transferred”, “Original”) → whatever was updated in “Original” needs to be copied into “Transferred”
    - EndWrites(“Transferred”) → “Transferred” is the resource we pass around
    - EndAllAccess(“Transferred”) → Because we know “Transferred” will not be used throughout the frame we are done with it!
- This is not an API flag, it’s implemented directly in SW and uses an applicable API flag

AMD CROSSFIRE API

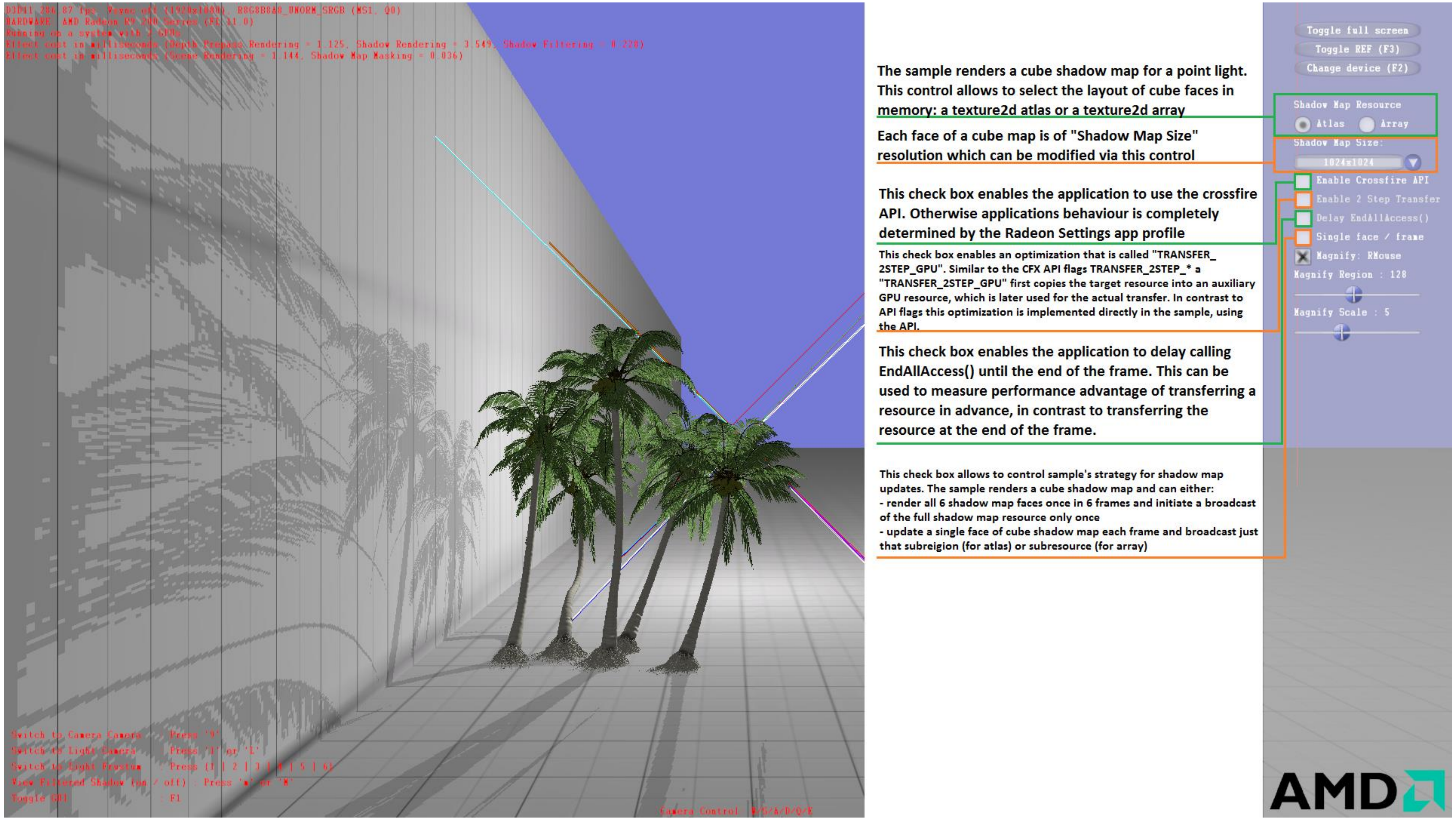
TRANSFER_2STEP_GPU



CROSSFIRE DIRECTX® 11 SAMPLE

CROSSFIRE DIRECTX® 11 SAMPLE

UI



```
D3D11 286.87 fps  Vsync off (1920x1080), R8G8B8A8_UNORM_SRGB (MS1, Q0)
HARDWARE: AMD Radeon R9 200 Series (FL 11.0)
Running on a system with 2 GPUs
Effect cost in milliseconds (Depth Prepass Rendering = 1.125, Shadow Rendering = 3.549, Shadow Filtering = 0.228)
Effect cost in milliseconds (Scene Rendering = 1.144, Shadow Map Masking = 0.036)
```

```
Switch to Camera Camera      : Press '9'
Switch to Light Camera       : Press 'I' or 'L'
Switch to Light Frustum      : Press (1 | 2 | 3 | 4 | 5 | 6)
View Filtered Shadow (on / off) : Press 'm' or 'M'
Toggle GUI                   : F1
```

```
Camera Control: W/S/A/D/Q/E
```

```
Toggle full screen
Toggle REF (F3)
Change device (F2)
Shadow Map Resource
[✓ Atlas]  Array
Shadow Map Size:
1024x1024
[ ] Enable Crossfire API
[ ] Enable 2 Step Transfer
[ ] Delay EndAllAccess()
[ ] Single face / frame
[✓] Magnify: RMouse
Magnify Region : 128
Magnify Scale : 5
```

The sample renders a cube shadow map for a point light. This control allows to select the layout of cube faces in memory: a texture2d atlas or a texture2d array

Each face of a cube map is of "Shadow Map Size" resolution which can be modified via this control

This check box enables the application to use the crossfire API. Otherwise applications behaviour is completely determined by the Radeon Settings app profile

This check box enables an optimization that is called "TRANSFER_2STEP_GPU". Similar to the CFX API flags TRANSFER_2STEP_* a "TRANSFER_2STEP_GPU" first copies the target resource into an auxiliary GPU resource, which is later used for the actual transfer. In contrast to API flags this optimization is implemented directly in the sample, using the API.

This check box enables the application to delay calling EndAllAccess() until the end of the frame. This can be used to measure performance advantage of transferring a resource in advance, in contrast to transferring the resource at the end of the frame.

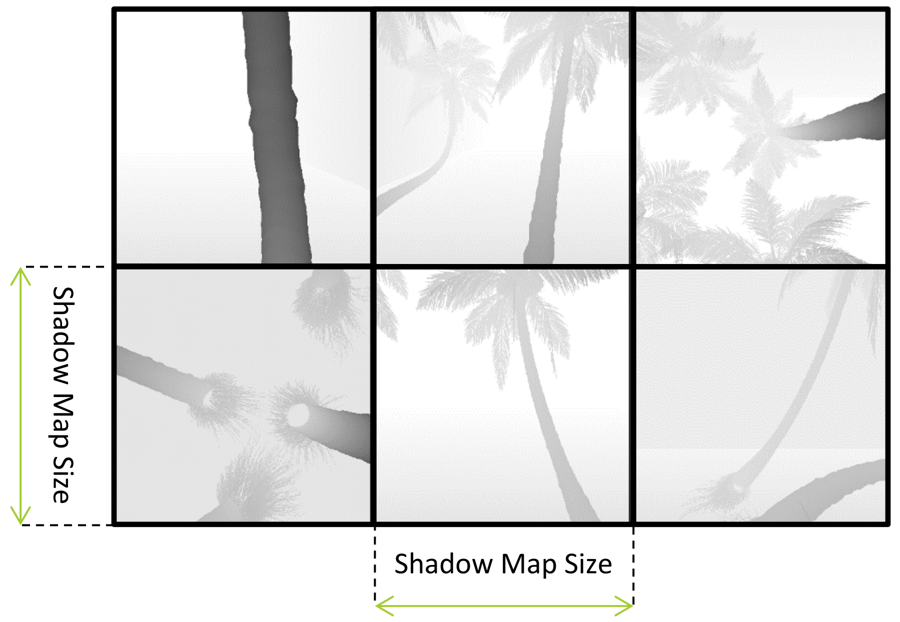
This check box allows to control sample's strategy for shadow map updates. The sample renders a cube shadow map and can either:
- render all 6 shadow map faces once in 6 frames and initiate a broadcast of the full shadow map resource only once
- update a single face of cube shadow map each frame and broadcast just that subreigion (for atlas) or subresource (for array)

CROSSFIRE DIRECTX® 11 SAMPLE

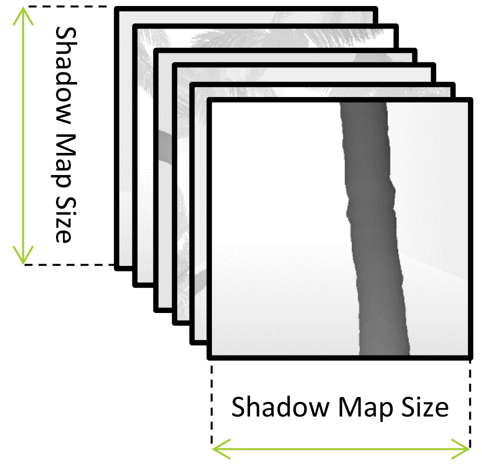
UI

- Sample renders a cube shadow map for a point light
  - A total of 6 shadow faces
- "Shadow Map Size" controls the size of a single cube face
  - So 1024^2 really means $6x1024^2$
- Shadow map layout
  - A 2D texture atlas
  - A 2D texture array, each cube face located in a separate array slice
- Shadow map updating schemes
  - Update all 6 shadow faces once in 6 frames
    - Frame *I* where (*I* % 6) == 0 renders more geometry and is slower
    - Frame *I* where (*I* % 6) == 0 broadcasts the full resource
  - Update 1 shadow face each frame
    - Each frame broadcasts a single sub-region (or a sub-resource) of the full resource
  - On average, both updating schemes:
    - render the same amount of geometry
    - transfer the same amount of data

Texture2D Atlas Layout



Texture2D Array Layout



PERFORMANCE

PERFORMANCE CHARACTERISTICS (2 RADEON R9 290X)

FOREWORD

- Numeric values in the performance tables are FPS
- Test Hardware
  - Two Radeon R9 290X
  - Core i5 3330
- Expectations for default driver modes (when CFX API is not used):
  - AFR-Friendly mode
    - Flicker is **expected** for both shadow map update schemes
    - This profile provides us with the Speed of Light (SOL) Crossfire performance
  - AFR-Compatible mode
    - Flicker is **expected** when 1 shadow face is updated each frame
      - The driver fails to detect a stale resource. This is explained on **slides 6-7**
    - This profile provides us with default Crossfire transfer performance
    - Crossfire API transfer performance is expected to be higher because sample initiates transfers earlier
- Crossfire API was tested with Radeon Settings Crossfire profile for the sample set to AFR-Friendly
  - Crossfire API performance is expected to be : AFR-Friendly ≥ Crossfire API ≥* AFR-Compatible
    - 2-STEP-GPU-TRANSFER has overhead for resource copies
    - * AFR-Compatible mode **will be as fast as AFR-Friendly** for shadow map update scheme {1 shadow face updated each frame) because the driver doesn't detect a stale resource.

PERFORMANCE CHARACTERISTICS

TRANSFER FLAG: 2STEP_WITH_BROADCAST

Texture2D Atlas (each subregion 512^2)	Single GPU	AFR Friendly	AFR Compatible	Crossfire API	Crossfire API: 2-STEP-GPU TRANSFER	Crossfire API: Delay EndAllAccess	Crossfire API 2-STEP-GPU TRANSFER + Delay EndAllAccess
6 shadow maps / updates once every 6 frames	274	352 Flicker	294	298	318	294	298
1 shadow map updated every frame	273	468 Flicker	466 Flicker	235	428	235	419

Texture2D Array (subresource 512^2)	Single GPU	AFR Friendly	AFR Compatible	Crossfire API	Crossfire API: 2-STEP-GPU TRANSFER	Crossfire API: Delay EndAllAccess	Crossfire API 2-STEP-GPU TRANSFER + Delay EndAllAccess
6 shadow maps / updates once every 6 frames	273	351 Flicker	297	294	300	289	276
1 shadow map updated every frame	273	465 Flicker	455 Flicker	405	400	402	390

PERFORMANCE CHARACTERISTICS

TRANSFER FLAG: 2STEP_WITH_BROADCAST

Texture2D Atlas (each subregion 1024^2)	Single GPU	AFR Friendly	AFR Compatible	Crossfire API	Crossfire API: 2-STEP-GPU TRANSFER	Crossfire API: Delay EndAllAccess	Crossfire API 2-STEP-GPU TRANSFER + Delay EndAllAccess
6 shadow maps / updates once every 6 frames	270	346 Flicker	130	178	209	179	209
1 shadow map updated every frame	268	449 Flicker	450 Flicker	95	342	95	336

Texture2D Array (subresource 1024^2)	Single GPU	AFR Friendly	AFR Compatible	Crossfire API	Crossfire API: 2-STEP-GPU TRANSFER	Crossfire API: Delay EndAllAccess	Crossfire API 2-STEP-GPU TRANSFER + Delay EndAllAccess
6 shadow maps / updates once every 6 frames	269	347 Flicker	130	174	169	173	169
1 shadow map updated every frame	269	449 Flicker	448 Flicker	303	297	307	297

PERFORMANCE CHARACTERISTICS

TRANSFER FLAG: 2STEP_WITH_BROADCAST

Texture2D Atlas (each subregion 2048^2)	Single GPU	AFR Friendly	AFR Compatible	Crossfire API	Crossfire API: 2-STEP-GPU TRANSFER	Crossfire API: Delay EndAllAccess	Crossfire API 2-STEP-GPU TRANSFER + Delay EndAllAccess
6 shadow maps / updates once every 6 frames	256	322 Flicker	40	74	71	73	73
1 shadow map updated every frame	251	424 Flicker	424 Flicker	25	123	25	123

Texture2D Array (subresource 2048^2)	Single GPU	AFR Friendly	AFR Compatible	Crossfire API	Crossfire API: 2-STEP-GPU TRANSFER	Crossfire API: Delay EndAllAccess	Crossfire API 2-STEP-GPU TRANSFER + Delay EndAllAccess
6 shadow maps / updates once every 6 frames	256	320 Flicker	40	81	74	80	74
1 shadow map updated every frame	257	429 Flicker	425 Flicker	136	121	135	120

PERFORMANCE CHARACTERISTICS

TRANSFER FLAG: 2STEP_WITH_BROADCAST

Texture2D Atlas (each subregion 4096^2)	Single GPU	AFR Friendly	AFR Compatible	Crossfire API	Crossfire API: 2-STEP-GPU TRANSFER	Crossfire API: Delay EndAllAccess	Crossfire API 2-STEP-GPU TRANSFER + Delay EndAllAccess
6 shadow maps / updates once every 6 frames	218	254 Flicker	12	22	20	22	20
1 shadow map updated every frame	198	348 Flicker	250 Flicker	7	32	7	32

Texture2D Array (subresource 4096^2)	Single GPU	AFR Friendly	AFR Compatible	Crossfire API	Crossfire API: 2-STEP-GPU TRANSFER	Crossfire API: Delay EndAllAccess	Crossfire API 2-STEP-GPU TRANSFER + Delay EndAllAccess
6 shadow maps / updates once every 6 frames	218	256 Flicker	12	25	23	25	23
1 shadow map updated every frame	218	355 Flicker	361 Flicker	36	32	36	32

DISCLAIMER & ATTRIBUTION