Enabling AMD 2D Hardware Acceleration for Video API and Applications

Solomon Chiu

AMD

together we advance_

Agenda

Introduction: Why 2D Acceleration Matters

Overview of AMD 2D Hardware Acceleration

Our Plumbing Works in the Software Stack

Results & Power Efficiency Improvements

Challenges & Call for Collaborations

Introduction: Why 2D Acceleration Matters

Video Processing

- Scaling
- CSC: Color Space/Format Conversion
  - RGB
  - YUV
- Rotation
  - 0/90/180/270 degrees
  - Mirroring
  - Flipping
- Blending
- HDR Video Tone mapping

Real-World Use Cases and Pipelines

- SDR/HDR Video playback
  - File reading → Decoder → CSC + Scaling + Tone Mapping(optional) → Rendering
- Video conferencing
  - Webcam(to remote): capturing YUV frames → CSC+Scaling → Encoding → Transferring
  - Receiving remote video stream: Receiving network video stream → Decoding
  - Presenting in local screen: CSC/scaling for both webcam YUV and video stream→ Blending → composition
- Game DVR
  - Framebuffer rendered by the games(HDR/SDR) → CSC, scaling(optional) → Encoding → file storing
- Cloud gaming
  - Framebuffer rendered by the games → CSC, scaling(optional) → Encoding → Network transferring
- Transcoding
  - File reading → Decoding → CSC/Scaling → Encoding → File Storing

Video APIs of Linux Which Support HW Acceleration

- VA/API: Video Acceleration API
  - Originally designed by Intel, remains the primary, actively supported, cross-vendor hardware API on Linux
  - Supports following operations
    - Decode: MPEG-2, MPEG-4 part 2, VC-1, H.264/AVC, HEVC/H.265, VP8, VP9, AV1, JPG/MJPEG
    - Encoding: MPEG-2, H.264/AVC, HEVC/H.265,VP8, AV1
    - Video post processing
- VDPAU: Video Decode and Presentation API for Unix
  - Originally designed and maintained by nVidia
  - Still widely used by Linux users, but shrinking
    - nVidia has stopped adding new features, and shift to NVDEC/NVENC
    - Modern codecs are not fully supported(only MPEG-1/2, VC-1, H.264/AVC, limited HEVC and VP9, no AV1 support)
    - AMD and Intel only provide translation layers not native VDPAU
- Vulkan Video
  - For video encoding/decoding only, doesn’t support video processing
  - Emerging API with cross-vendor support, unified decode/encode API and AV1/HEC/H.265 coverage

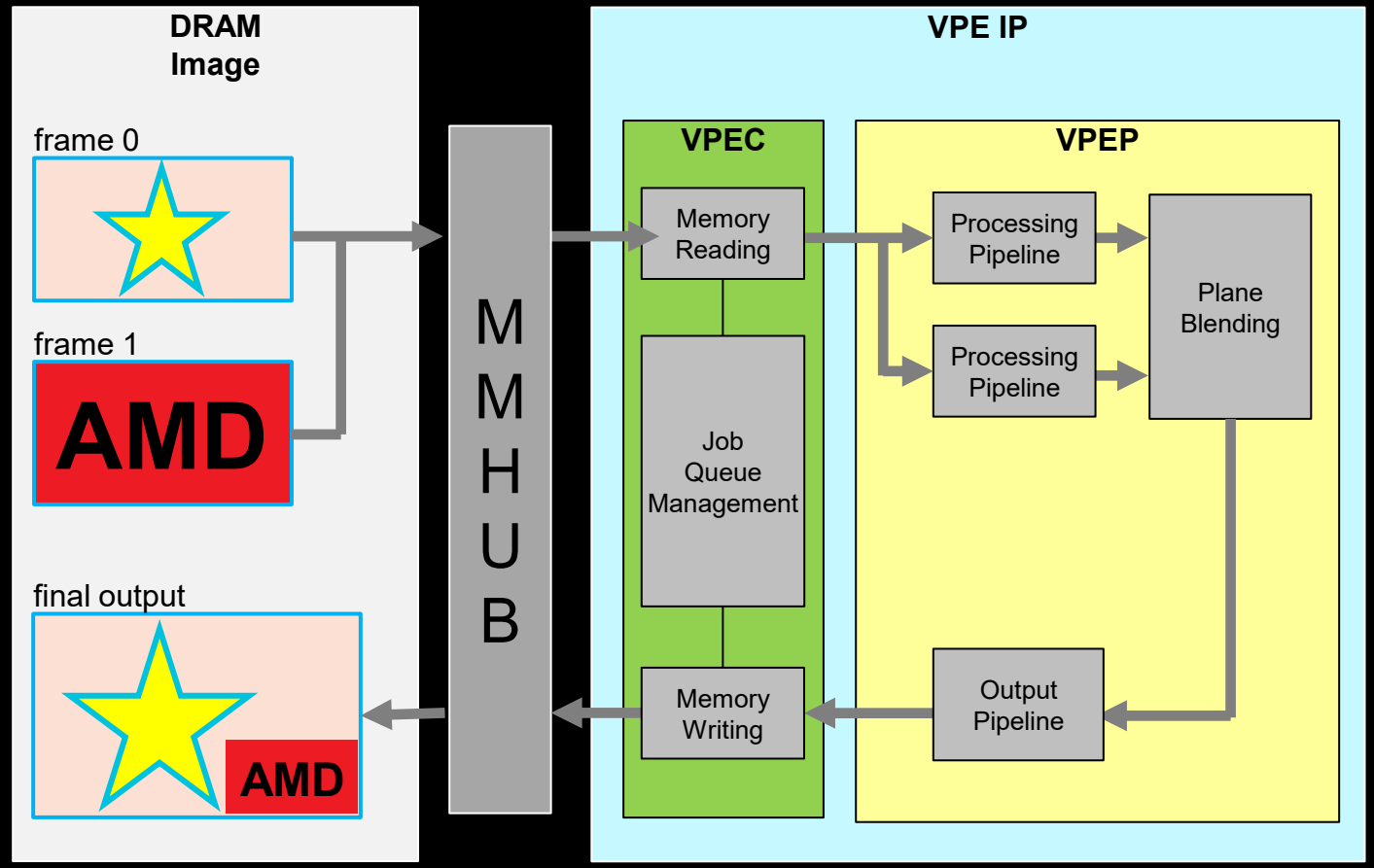
Overview of AMD 2D Hardware Acceleration

AMD’s Video Acceleration Solutions

- VCN – Video Core Next
  - Fixed function core for encoding, decoding and transcoding
- MPO - Multi-plane overlay protocol
  - Help to remove 3D engine overhead to compose video output with desktop
  - For example, decoded NV12 frames are scaled and displayed direct without need to convert to RGB and compose with desktop plane
- Why are we adding new hardware?
  - In previous AMD’s platform, all the post processing works were implemented with shader code on Linux
  - Limited layer support in MPO, scaling ratio limitation, and the number of displays could impact MPO availability
  - Thus, we need a 2D Video Processing hardware acceleration solution - VPE, to help on saving power with following operations

Overview of AMD 2D Hardware Acceleration

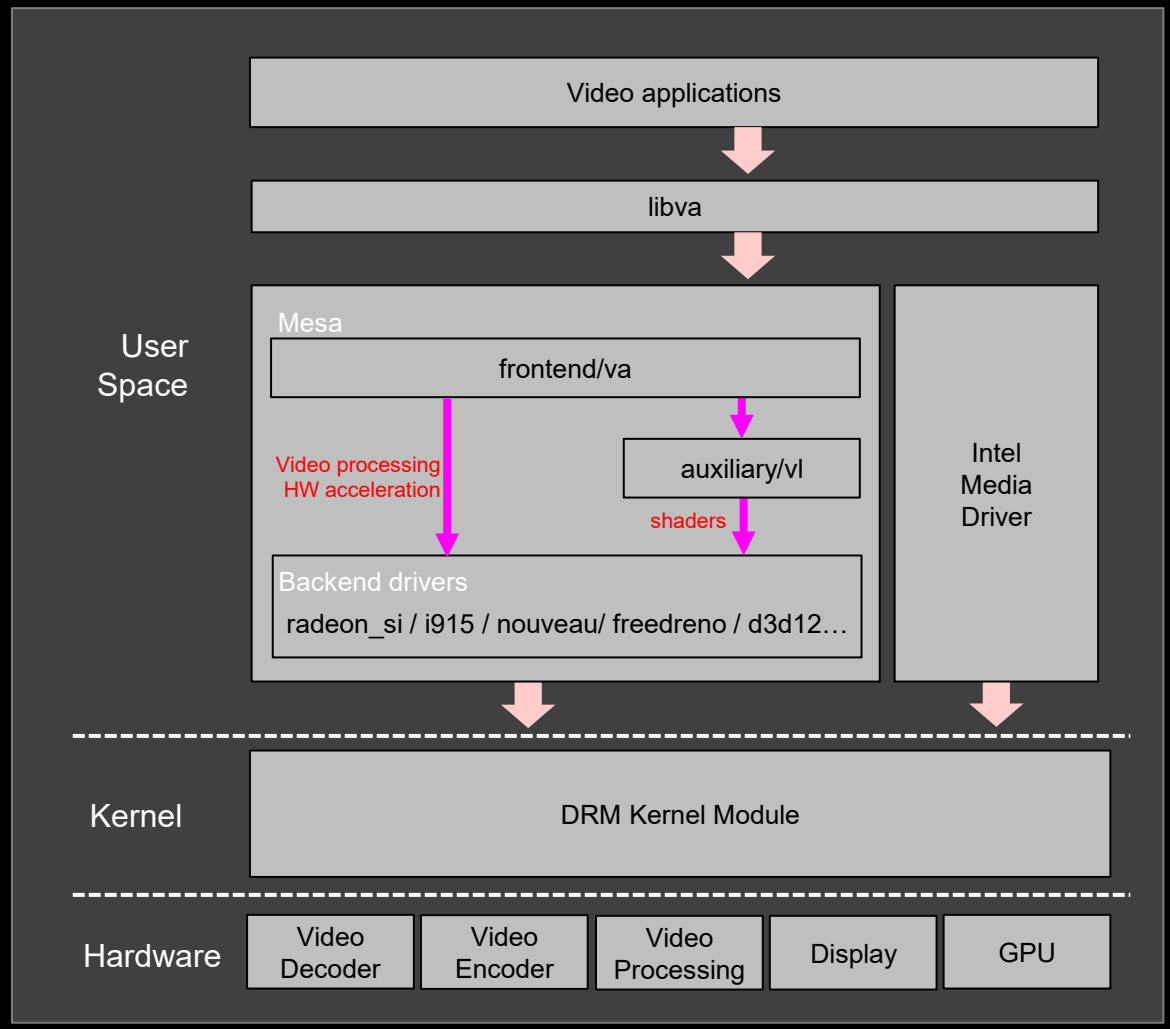
- VPEC: VPE Control & DMA Engine from AMD's SDMA
  - Control engine for clock/power/reset
  - Instruct the Memory Reader and Writer to fetch and write the plane data according
  - To fetch, decode and execute the command
  - Tiling + de-tiling
  - A MCU to execute firmware submitted command in ring buffer, and manage/schedule the execution order of multiple HW queues, based on their band and local priority properties
- VPEP: VPE Process Pipeline from AMD's DCN to do
  - CSC: Read/write of commonly used formats: NV12; P010; Packed 32-bit RGB; ARGB 8; ARGB 1010102; ARGB FP16
  - Scaling
  - Blending
  - Gamut Remap
  - Tone mapping
  - Programmable Gamma



Our Plumbing Work in the Software Stack

Overview of VA/API

- frontend: libva
  - Get the hardware information from display
  - Load the corresponding user-space backend driver
- backend:
  - Mesa: drivers for AMD, legacy intel, nouveau…etc.
  - Intel Media Driver: supports most recently 2D graphics HW, doesn't depend on Mesa.
- Applications:
  - ffmpeg
  - MPV player
  - VLC
  - Chrome
  - Firefox
  - Kodi
  - Jellyfin Media Server
  - OBS Studio



Typical Usage of VA/API

1. Initialization: display, va context

```
// Initialization
vaGetDisplay()
vaInitialize()

// Configuration
vaQueryConfigEntrypoints()
vaGetConfigAttributes()
vaCreateConfig()

// Allocate surface, context
vaCreateSurfaces()
vaCreateContext()
```

2. Query capability of Video Processing Hardware

```
typedef struct _VAProcPipelineCaps {
    uint32_t        pipeline_flags;
    uint32_t        filter_flags;
    uint32_t        num_forward_references;
    uint32_t        num_backward_references;
    VAProcColorStandardType *input_color_standards;
    uint32_t        num_input_color_standards;
    VAProcColorStandardType *output_color_standards;
    uint32_t        num_output_color_standards;

    uint32_t        rotation_flags;
    uint32_t        blend_flags;
    uint32_t        mirror_flags;
    uint32_t        num_additional_outputs;

    uint32_t        num_input_pixel_formats;
    uint32_t        *input_pixel_format;
    uint32_t        num_output_pixel_formats;
    uint32_t        *output_pixel_format;

    uint32_t        max_input_width, max_input_height;
    uint32_t        min_input_width, min_input_height;
    uint32_t        max_output_width, max_output_height;
    uint32_t        min_output_width, min_output_height;
} VAProcPipelineCaps;
VAProcPipelineCaps pipeline_caps;

vaQueryVideoProcPipelineCaps(va_dpy, vpp_ctx,
    filter_bufs, num_filter_bufs,
    &pipeline_caps
);
```

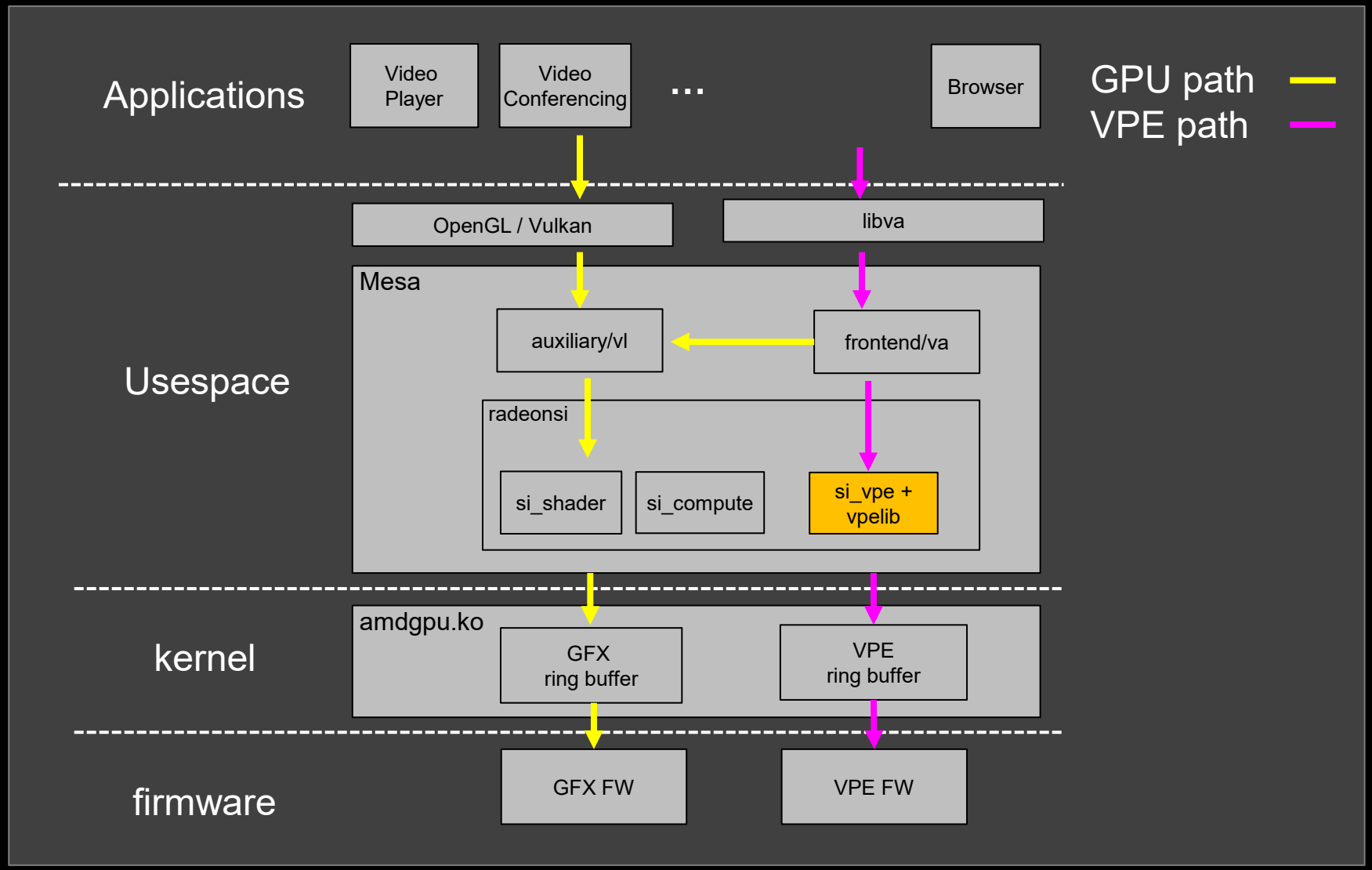
3. Main loop

```
// main loop
while() {
  // Setup required buffers
  vaCreateBuffer()
  vaMapBuffer()
  vaUnmapBuffer()

  // The trilogy
  vaBeginPicture();
  vaRenderPicture();
  vaRenderPicture();
    :
  vaRenderPicture();
  vaEndPicture();

  // Update screen
  vaSyncSurface()
  // or vaPutSurface()
}
```

AMD's VPE Software Stack

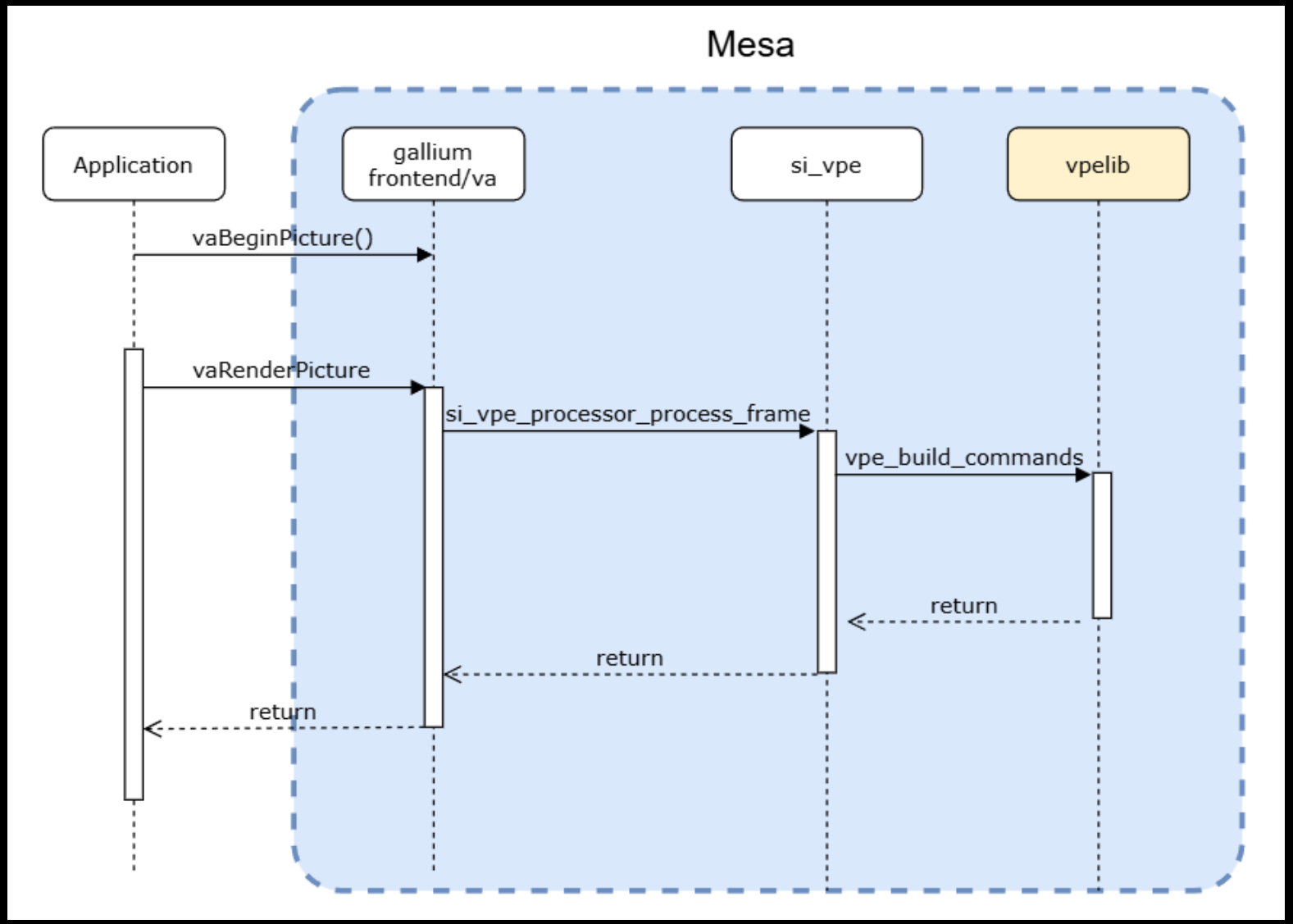


VPElib

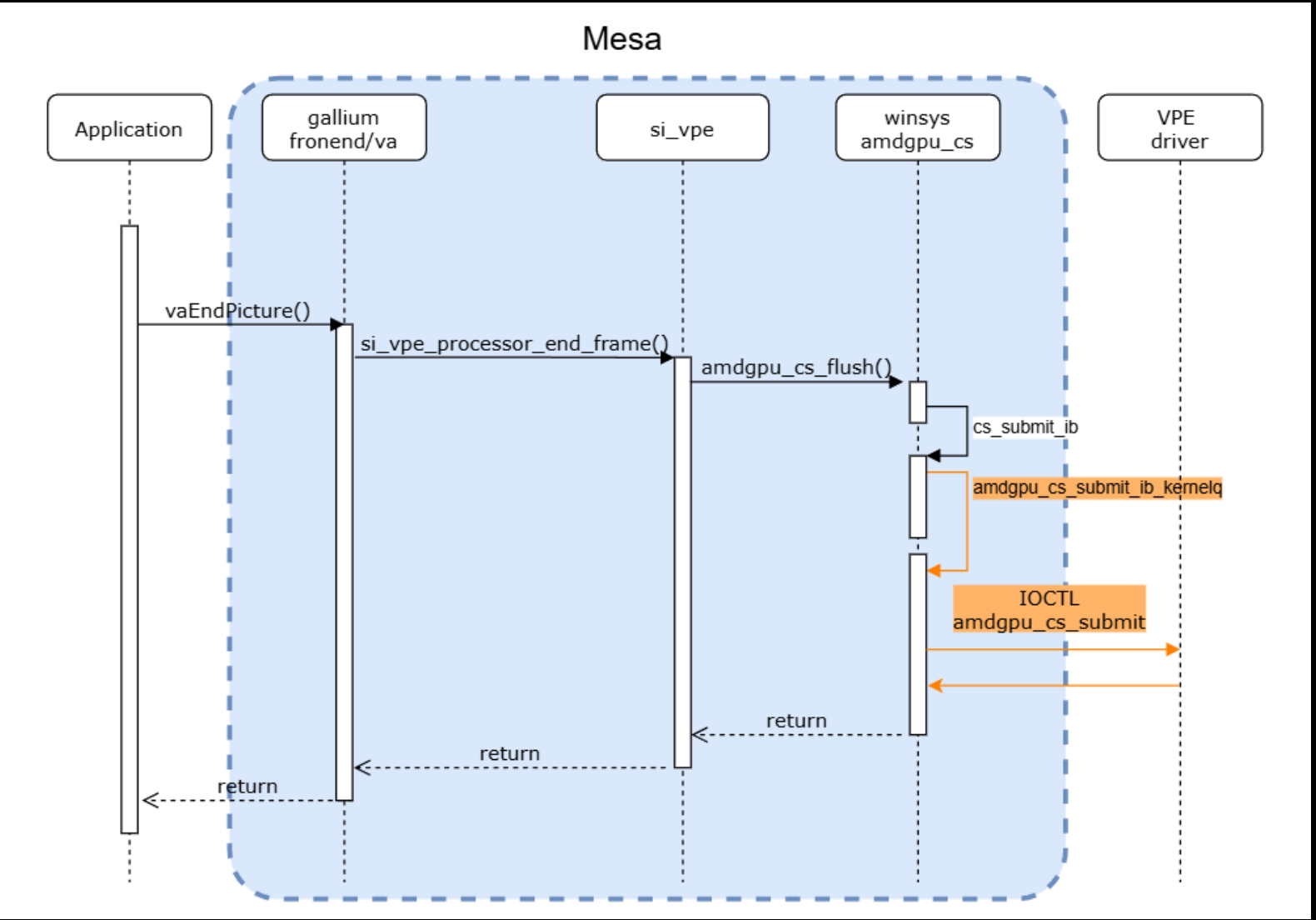
- Constructing a command buffer based on the Video processing requirement.
- The library is in C and shared between Linux and Windows drivers
- To create the content of a command buffer which stores all the VPEC/VPEP register programming sequence
  - Generate all the configurations In the command buffer to be executed later
  - Does not touch directly to VPE HW

Function	Description
`vpe_create()`	Create an VPE instance for VPblit command building. Callers shall create one per video processor. i.e. for one command stream. The created struct vpe allows callers to check the ip version of the engine, and its basic capabilities
`vpe_destroy()`	Destroy the vpe instance .
`vpe_check_vpblit_support()`	This allows the caller to check if the operations can be supported by the engine.
`vpe_build_noops()`	Build a stream of noop commands
`Vpe_build_vpblit()`	It takes the allocated buffer and the vpblit params and start building the command buffer. The result would be all VPE descriptors are created in a command buffer and other descriptors are stored in an embedded buffer for a complete frame process.

vaRenderPicture: Generating VPE Commands:



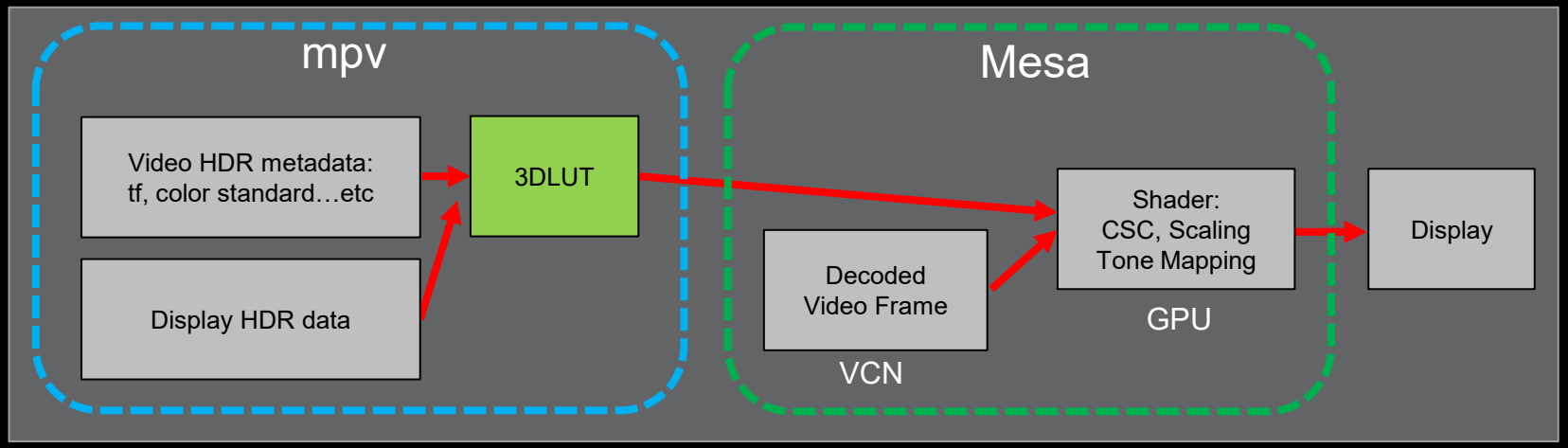
vaEndPicture: Submitting VPE Commands:



HDR Video Playback on MPV player

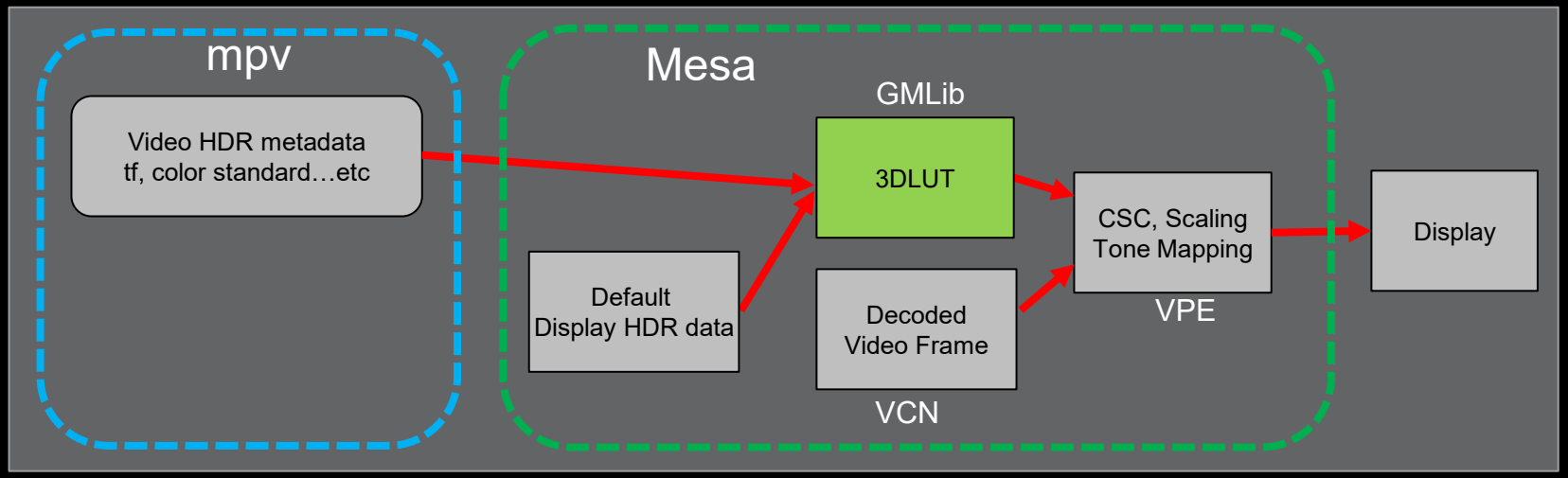
- GPU way:

```
mpv --hwdec=vaapi
```



- VPE way:

```
mpv --hwdec=vaapi
--vo=dmabuf-wayland
--vf=scale_vaapi=format=rgba
```



HDR Video Tone Mapping

- Problem Statements
  - mpv doesn’t pass HDR metadata and 3DLUT to Mesa
  - Mesa doesn’t support 3DLUT filter to receive 3DLUT
  - Given above constraints, we eventually use scale_vaapi filter to send color standard and scaling, CSC settings to Mesa at the same time.
  - Another way we are still trying is receiving data from tonemap_vaapi filter
- GMLib: AMD’s open source tonemap generator
  - Mesa: src/amd/gmlib

```
/* Fill all parametters that GMLib needs to calculate tone mapping 3DLut */
tm_par.tm_handle = vpeproc->gm_handle;
tm_par.lutDim = VPE_LUT_DIM;
/* In */
tm_par.streamMetaData.redPrimaryX               = build_param->streams[0].hdr_metadata.redX;
tm_par.streamMetaData.redPrimaryY               = build_param->streams[0].hdr_metadata.redY;
tm_par.streamMetaData.greenPrimaryX             = build_param->streams[0].hdr_metadata.greenX;
tm_par.streamMetaData.greenPrimaryY             = build_param->streams[0].hdr_metadata.greenY;
tm_par.streamMetaData.bluePrimaryX              = build_param->streams[0].hdr_metadata.blueX;
tm_par.streamMetaData.bluePrimaryY              = build_param->streams[0].hdr_metadata.blueY;
tm_par.streamMetaData.whitePointX               = build_param->streams[0].hdr_metadata.whiteX;
tm_par.streamMetaData.whitePointY               = build_param->streams[0].hdr_metadata.whiteY;
tm_par.streamMetaData.maxMasteringLuminance     = build_param->streams[0].hdr_metadata.max_mastering;
tm_par.streamMetaData.minMasteringLuminance     = build_param->streams[0].hdr_metadata.min_mastering;
tm_par.streamMetaData.maxContentLightLevel      = build_param->streams[0].hdr_metadata.max_content;
tm_par.streamMetaData.maxFrameAverageLightLevel = build_param->streams[0].hdr_metadata.avg_content;
tm_par.inputContainerGamma                      = si_vpe_maps_vpe_to_gm_transfer_function(build_param-
/* Out */
tm_par.dstMetaData.redPrimaryX                  = build_param->hdr_metadata.redX;
tm_par.dstMetaData.redPrimaryY                  = build_param->hdr_metadata.redY;
tm_par.dstMetaData.greenPrimaryX                = build_param->hdr_metadata.greenX;
tm_par.dstMetaData.greenPrimaryY                = build_param->hdr_metadata.greenY;
tm_par.dstMetaData.bluePrimaryX                 = build_param->hdr_metadata.blueX;
tm_par.dstMetaData.bluePrimaryY                 = build_param->hdr_metadata.blueY;
tm_par.dstMetaData.whitePointX                  = build_param->hdr_metadata.whiteX;
tm_par.dstMetaData.whitePointY                  = build_param->hdr_metadata.whiteY;
tm_par.dstMetaData.maxMasteringLuminance        = build_param->hdr_metadata.max_mastering;
tm_par.dstMetaData.minMasteringLuminance        = build_param->hdr_metadata.min_mastering;
tm_par.dstMetaData.maxContentLightLevel         = build_param->hdr_metadata.max_content;
tm_par.dstMetaData.maxFrameAverageLightLevel    = build_param->hdr_metadata.avg_content;
tm_par.outputContainerGamma                     = si_vpe_maps_vpe_to_gm_transfer_function(build_param-

/* If the tone mapping of source is changed during playback, it must be recalculated.
 * Now assume that the tone mapping is fixed.
 */
if (tm_generate3DLut(&tm_par, vpeproc->lut_data)) {
   SIVPE_WARN(vpeproc->log_level, "Generate lut data faied, skip tonemapping\n");
   FREE(vpeproc->lut_data);
   build_param->streams[0].flags.hdr_metadata = 0;
   return;
}
```

VPE Kernel Driver

- Command submission
  - Ring-buffer management: shared code of AMD IPs
  - Unified DRM IOCTL of submission for AMD’s GPU, VCN, and VPE

- Power/clock gating
  - Based on the workload, perform dynamic power management with VPE firmware for S0i3 and Z8

Results & Power Efficiency Improvements

Results & Power Efficiency Improvements

Scenario	Power Consumption	Comments
Idle desktop	4.5W	
HDR Video playback; default	20 W	mpv – 4k 24Hz HDR Video
HDR Video playback with VCN	11.5 W	mpv – 4k 24Hz HDR Video --hwdec=vaapi
HDR Video playback; VCN + dmabuf	8.8 W	mpv – 4k 24Hz HDR Video --hwdec=vaapi --vo=dmabuf-wayland
HDR Video playback; VCN + dmabuf +VPE	9.5 W	mpv – 4k 24Hz HDR Video --hwdec=vaapi --vo=dmabuf-wayland --vf=scale_vaapi=format=rgba

Challenges & Lessons Learned

Challenges & Call for Collaborations

- Adding new color formats
  - libva
  - Mesa
- mpv video filter
  - We didn’t find a good way to pass user’s HDR data of display to VPE
  - ITU-R BT.601 and ITU-R BT.709 are used as default SDR data of display => unable to represent real ability of HDR display
  - The “scale_vaapi” filter need to be specified explicitly => inflexible, inconvenience
  - VPE can perform scale / color conversion / blending / mirror simultaneously, however it has to do those operations iteratively while scale_vaapi filter is given
- Enabling VPE for Chromium and Firefox
- Enabling VPE for Wayland compositor
  - dmabuf / eglImageKHR
- Vulkan supports
  - vkCmdCopyImage() and vkCmdBlitImage() don’t support color conversion and blending

Questions?


AMD