Vulkan Video
Core API Introduction

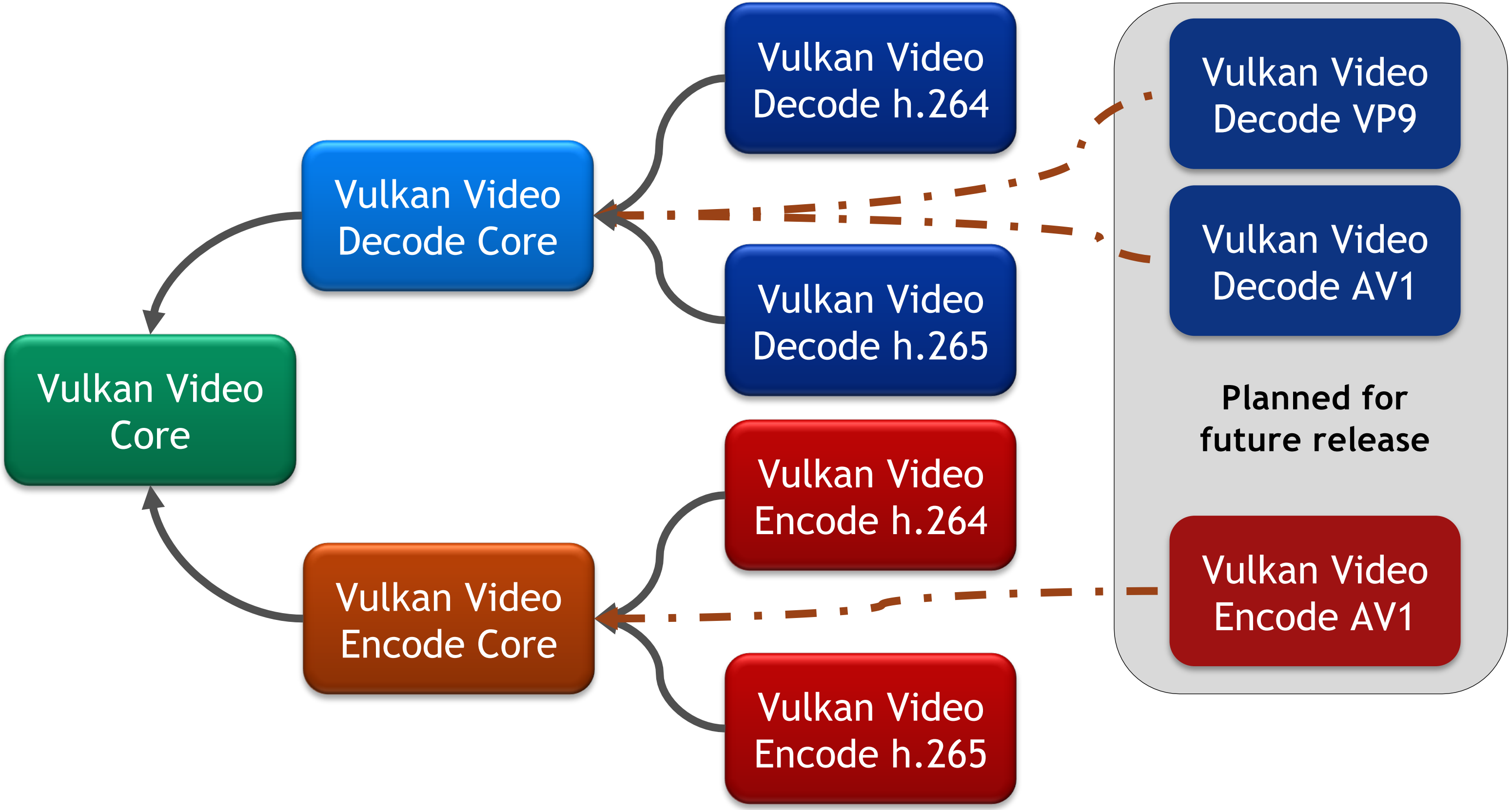
Tony Zlatinski, NVIDIA

April 2021

Vulkan Video Design Goals

- **Low-level stateless management of hardware for efficiency and flexibility**
  - Low-level synchronization for lower processing latency and efficient hardware scheduling
  - Low execution overhead
  - Low CPU/GPU/HW and memory resource utilization
- **Suitable for low-power/memory embedded devices to high-performance servers**
- **Distribution of video processing across multiple CPU cores and video-codec devices**
- **Closer integration with Vulkan Graphics and Displays**

Vulkan Video Core and Codec Extensions



Vulkan Video Core

Vulkan Video Decode Core

Vulkan Video Encode Core

Vulkan Video Decode h.264

Vulkan Video Decode h.265

Vulkan Video Encode h.264

Vulkan Video Encode h.265

Vulkan Video Decode VP9

Vulkan Video Decode AV1

Planned for future release

Vulkan Video Encode AV1

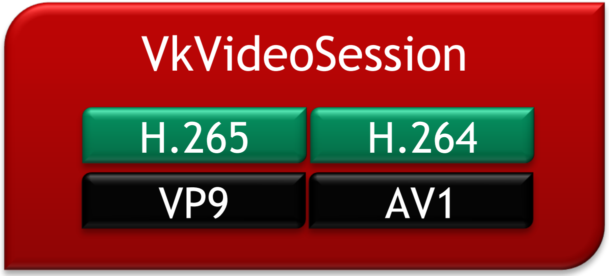
Vulkan Video Profiles

- **Vulkan Video Profiles are containers of formats describing the compressed bitstream**
- **VkVideoProfile describes:**
  - videoCodecOperation – codec operations such as h264 encode, h265 encode, etc.
  - chromaSubsampling – YCBCr 4:4:4, 4:2:2, 4:2:0, 4:0:0 color subsampling mode
  - lumaBitDepth and chromaBitDepth describe luminance & chroma channel bit depth – 8,10,12-bit
- **Video Profile structure must be included when obtaining device properties or creating Vulkan objects that will be used with Vulkan Video**

Vulkan API	Vulkan Structure to Extend
vkGetPhysicalDeviceFormatProperties2	VkFormatProperties2
vkCreateImage	VkImageCreateInfo
vkCreateImageView	VkImageViewCreateInfo
vkCreateBuffer	VkBufferCreateInfo
vkCreateQueryPool	VkQueryPoolCreateInfo

Video Session Object

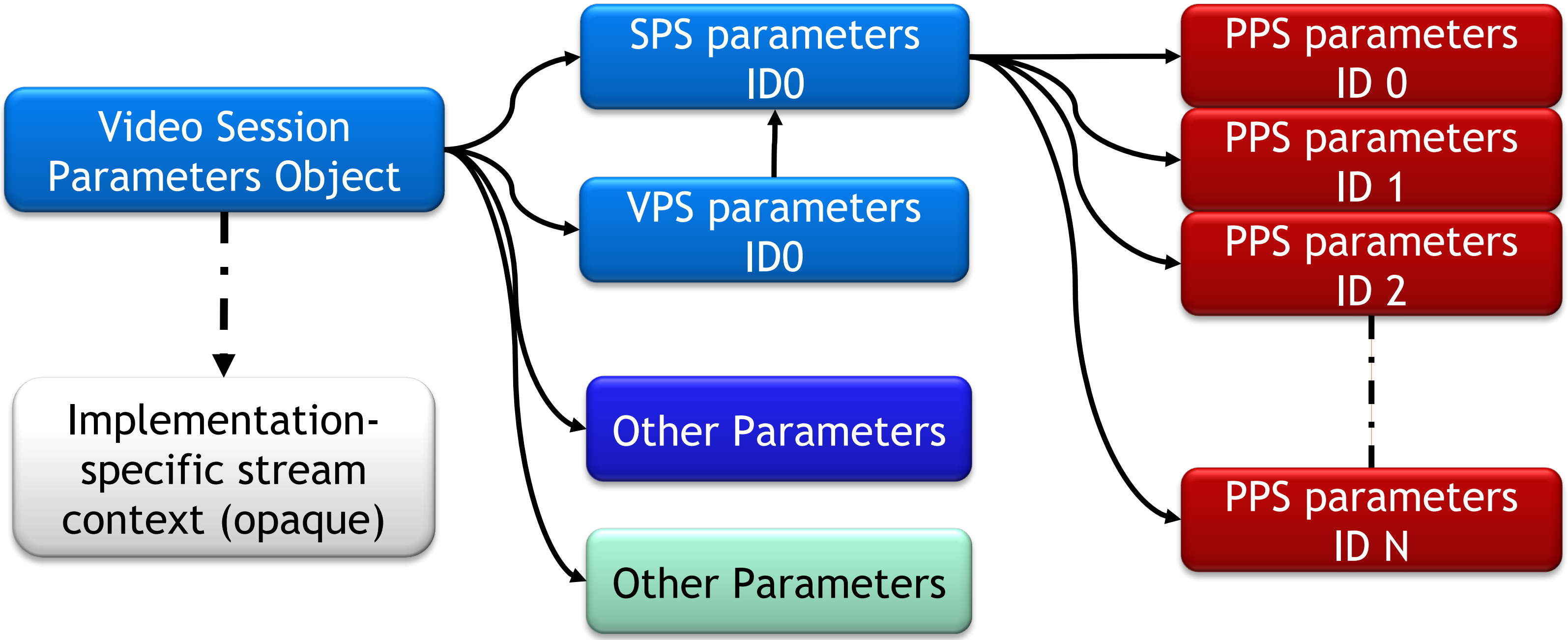
- **VkVideoSession object contain (read-only) stream configuration parameters and maintains the context associated with the stream**
  - One session object per video stream
- **Created before using any video decode or encode operations**
  - Specifies the video profile and maximum parameters for the video stream
- **A video session instance supports a single compression standard only**
  - H.264, HEVC, VP9, AV1, etc.
- **Video Session object maintains the device memory heaps**
  - The application allocates and binds VkDeviceMemory objects to the Video Session object which uses it for its memory heaps



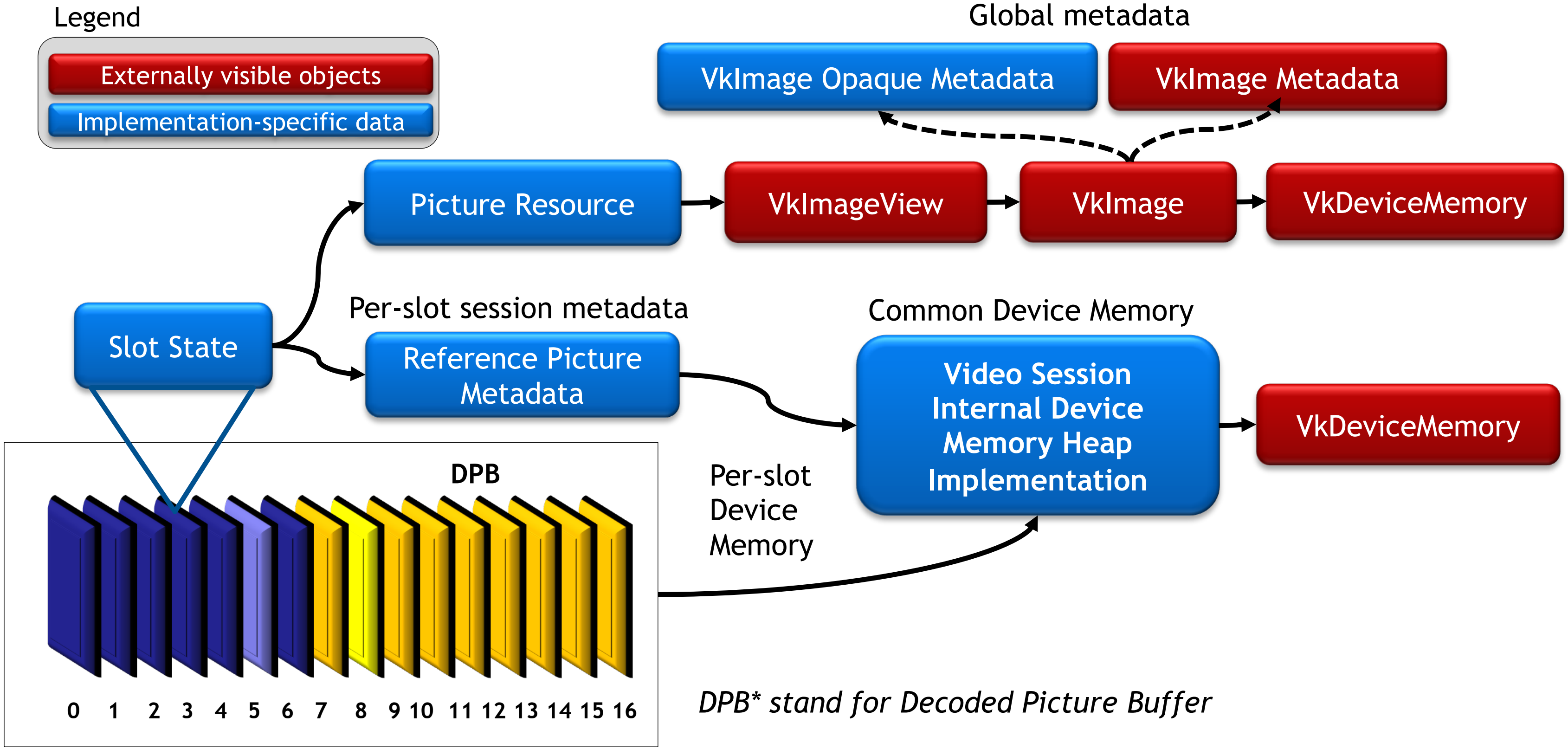
Video Session Parameters Object

- **VkVideoSessionParameters object contains processing parameters**
  - Created against and belongs to a Video Session object
- **Use multiple VkVideoSessionParameters objects to process a stream**
  - An object can apply to the whole stream or a portion
  - Session Parameter object is provided with the vkCmdBeginVideoCoding command and remains in effect until the next vkCmdEndVideoCoding command
- **Can add parameters to a the VkVideoSessionParameters object**
  - Previously parameters cannot be modified
  - Can clone all video parameters into a new Session Parameter object

Example of a Set of HEVC Codec Parameters

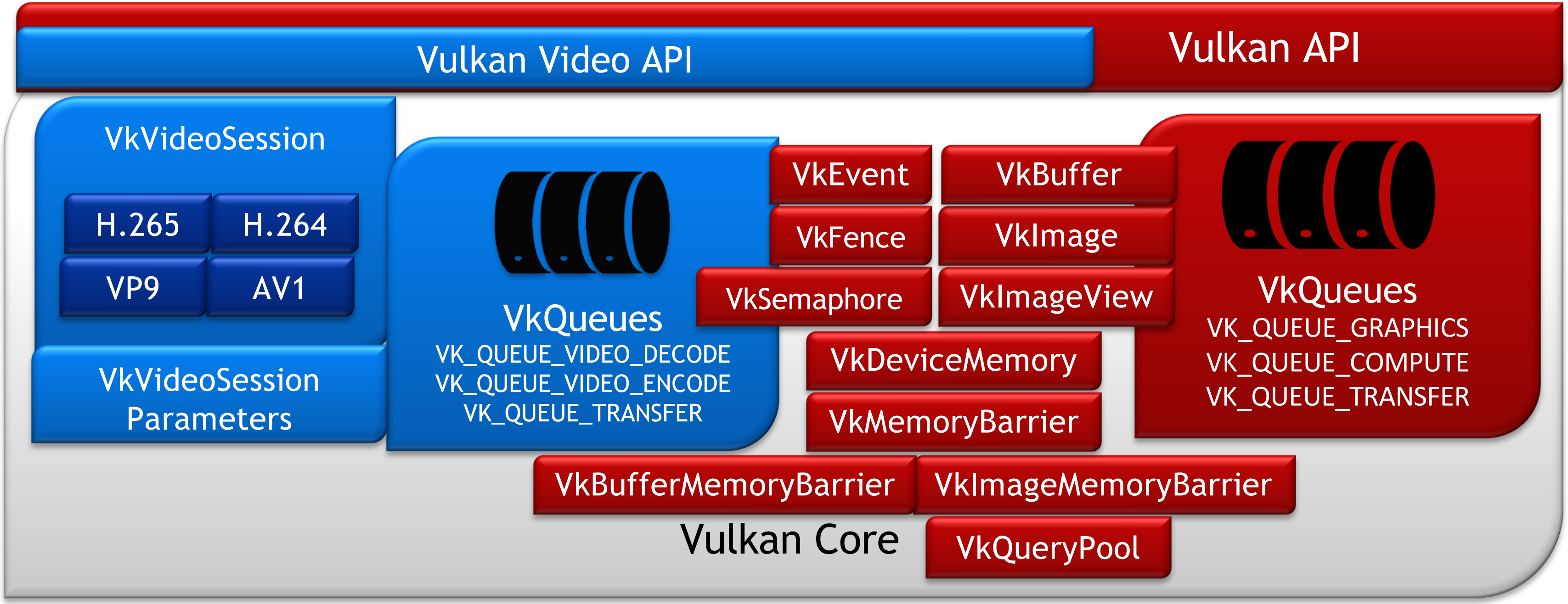


Video Decode/Encode DPB Picture Resources

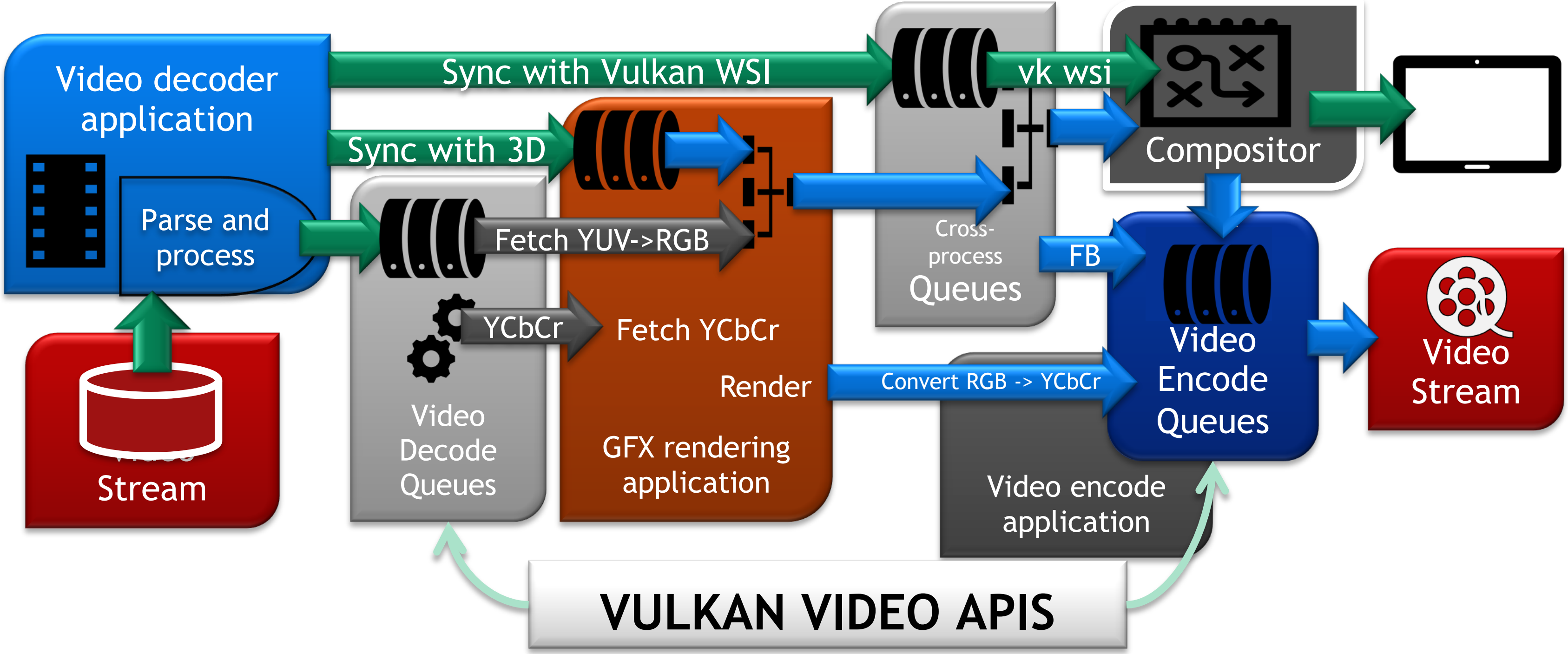


DPB stand for Decoded Picture Buffer*

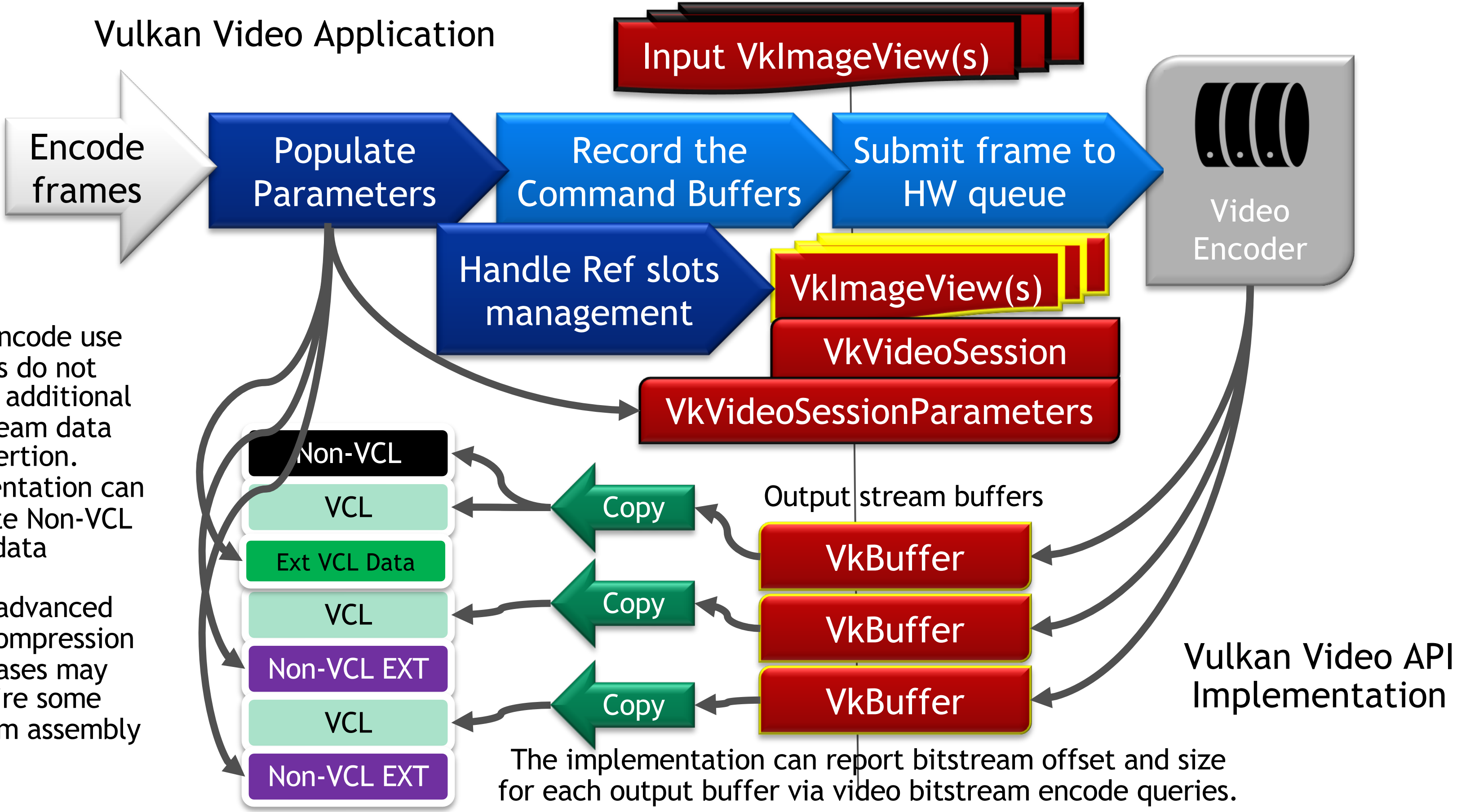
Video and General Vulkan Objects



Typical Vulkan Video Decode & Encode App



Video Encode Processing In Vulkan

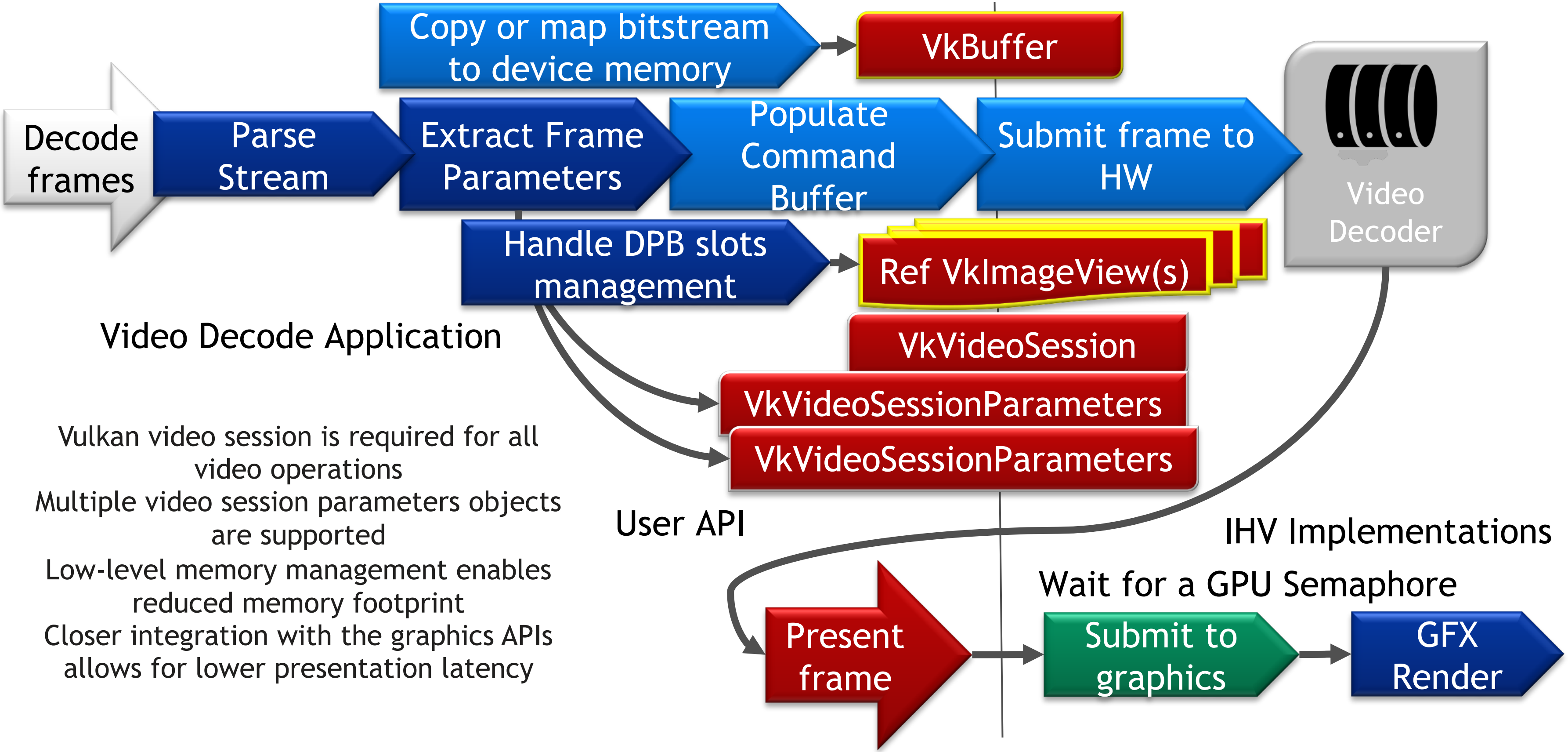


Most encode use cases do not require additional bitstream data insertion. Implementation can generate Non-VCL data

More advanced video compression use cases may require some bitstream assembly

The implementation can report bitstream offset and size for each output buffer via video bitstream encode queries.

Video Decode Processing In Vulkan



Vulkan video session is required for all video operations

Multiple video session parameters objects are supported

Low-level memory management enables reduced memory footprint

Closer integration with the graphics APIs allows for lower presentation latency

Optimizing Memory Usage

- **Create Sessions with the maximum parameters required for video content**
  - Max resolution, max number of DPB, etc.
- **Allocate image and buffer resources on demand**
  - When the content requires those resources
- **Free image or buffer resources that are not required**
- **Strip the resources of their physical memory backing**
  - Using sparse memory binding if supported
- **Enable the output of the decoded images to be directly consumed by Vulkan graphics and display processing pipelines**
- **Enable for the output of the Vulkan graphics or display processing to be consumed directly by the encoder's input**

Vulkan Sparse Resources with Video

- **4k/8k and 10/12-bit video content requires significant memory resources for stream buffers and picture images**
  - One frame can be bigger than 2 MB. Video session may require 3-8 input and/or output images and 4 to 16 references that requires hundreds of megabytes of memory
- **IHVs should support Vulkan Sparse binding for buffers and images for memory efficient resource management**
  - Sparse Partially-Resident Buffers
  - Support for both Sparse Buffers with VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT enables for portions of the Vulkan buffers used for the input or output stream to be unmapped
- **Sparse Partially-Resident Images**
  - VK_IMAGE_CREATE_SPARSE_BINDING_BIT and VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT to support an efficient memory use during content resolution change
  - Use vkQueueBindSparse() before or after the queuing video commands

Reusing images without reallocation on decode size change

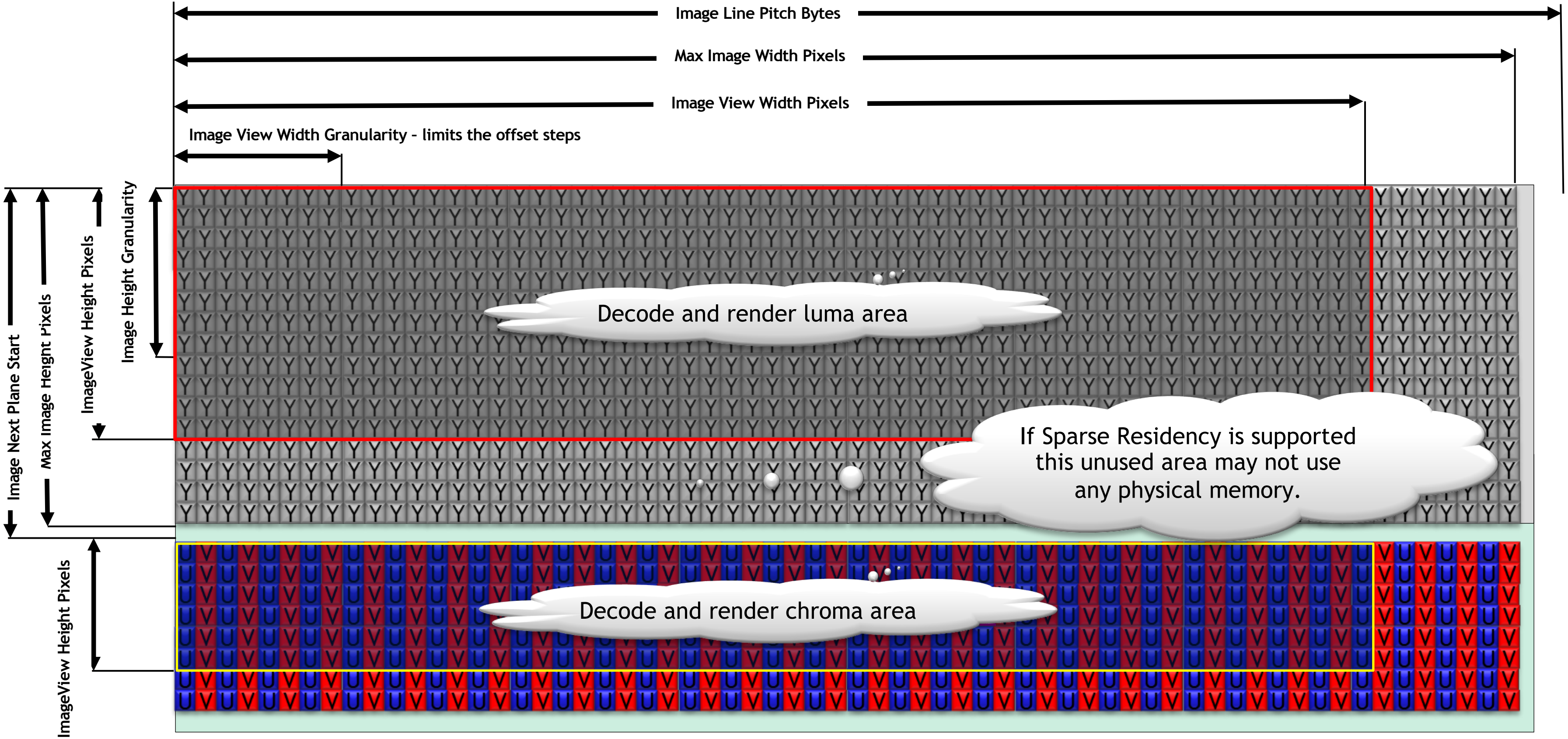
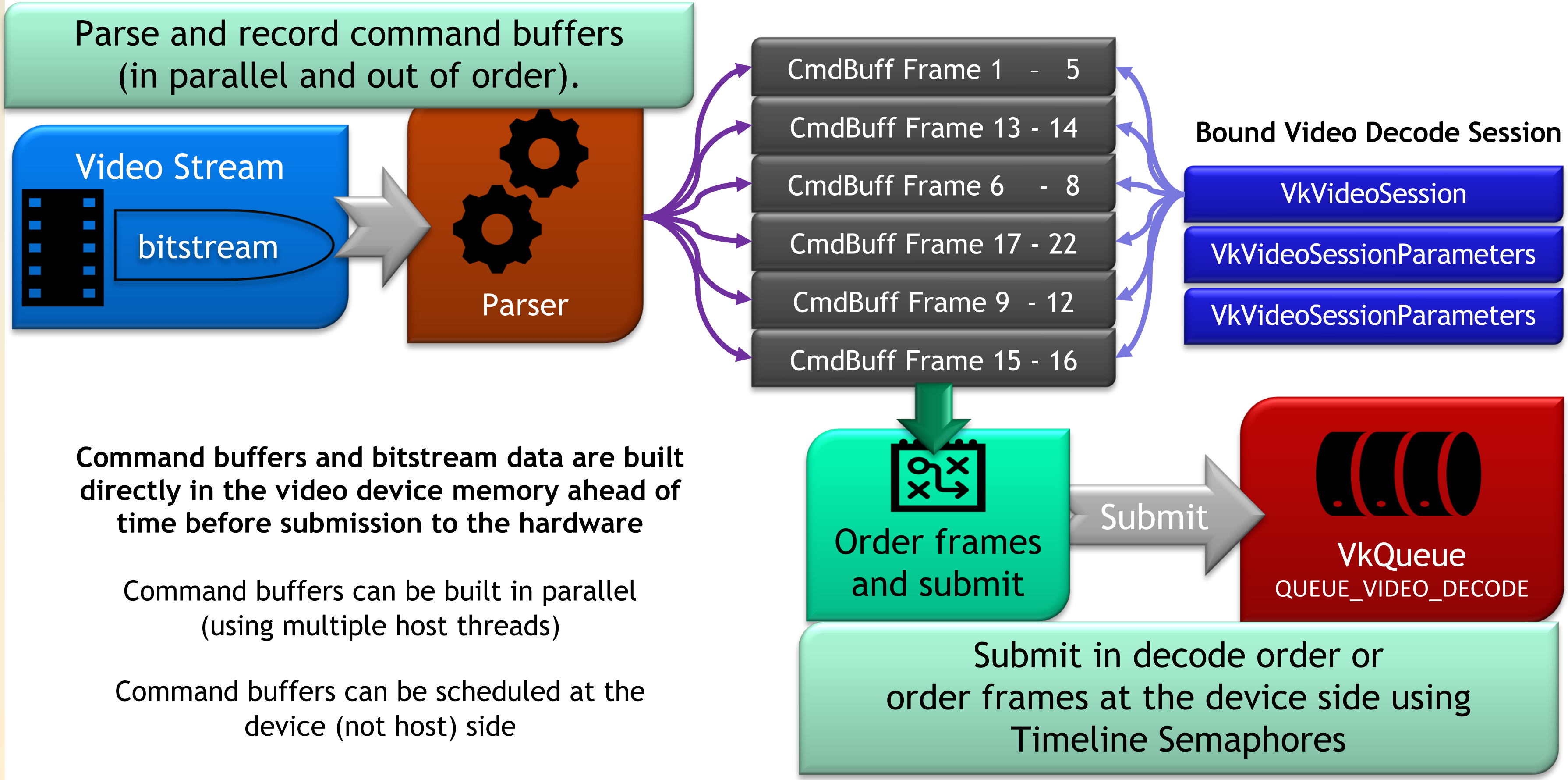


Image Line Pitch Bytes
Max Image Width Pixels
Image View Width Pixels
Image View Width Granularity - limits the offset steps
Image Next Plane Start
Max Image Height Pixels
ImageView Height Pixels
Image Height Granularity
Decode and render luma area
If Sparse Residency is supported this unused area may not use any physical memory.
ImageView Height Pixels
Decode and render chroma area
Y
U
V

Applications can save memory by removing physical memory residency, if supported by the implementation

Vulkan Video API Advantages



Parse and record command buffers (in parallel and out of order).

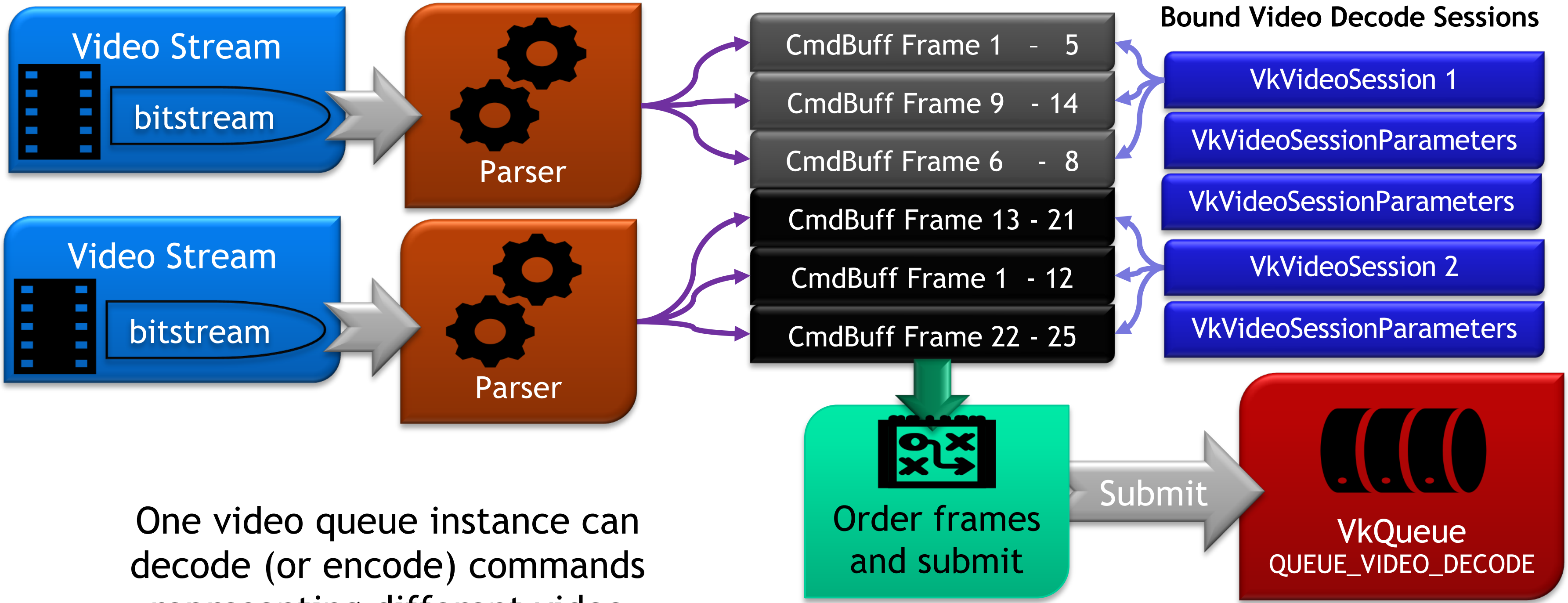
Submit in decode order or order frames at the device side using Timeline Semaphores

Command buffers and bitstream data are built directly in the video device memory ahead of time before submission to the hardware

Command buffers can be built in parallel (using multiple host threads)

Command buffers can be scheduled at the device (not host) side

Vulkan Video API Advantages (2)



Video Stream

bitstream

Parser

Video Stream

bitstream

Parser

CmdBuff Frame 1 – 5

CmdBuff Frame 9 - 14

CmdBuff Frame 6 - 8

CmdBuff Frame 13 - 21

CmdBuff Frame 1 - 12

CmdBuff Frame 22 - 25

Bound Video Decode Sessions

VkVideoSession 1

VkVideoSessionParameters

VkVideoSessionParameters

VkVideoSession 2

VkVideoSessionParameters

Order frames and submit

Submit

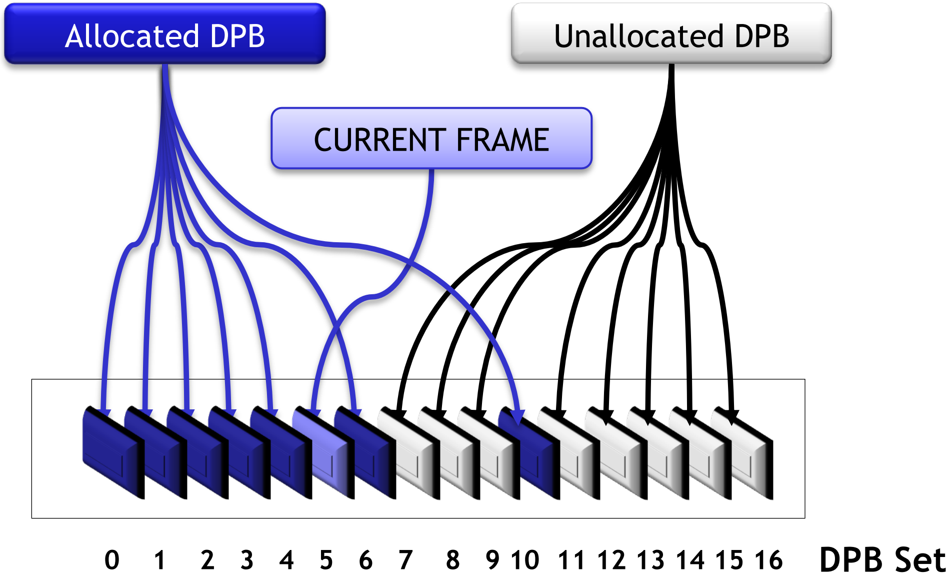
VkQueue
QUEUE_VIDEO_DECODE

One video queue instance can decode (or encode) commands representing different video streams in parallel

Submit in decode order or order frames at the device side using Timeline Semaphores

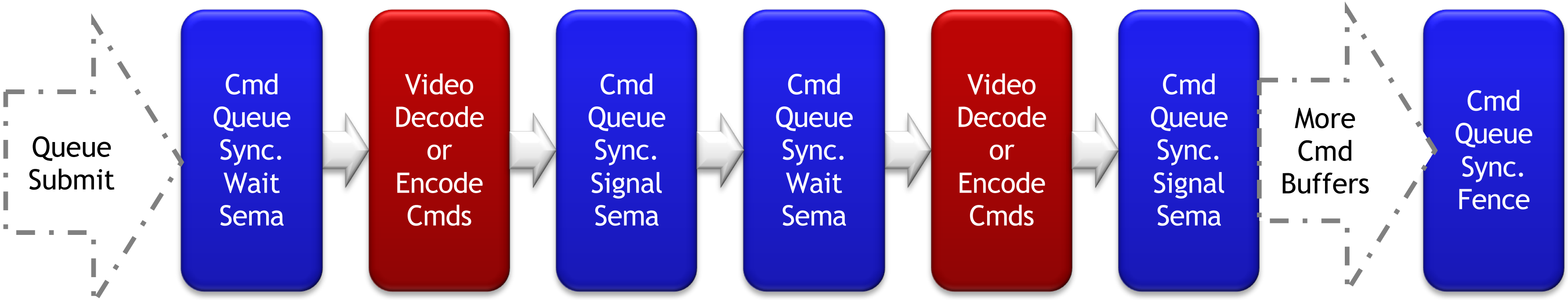
Vulkan Video API Advantages (3)

- **Application can optimize use of system resources**
  - Allocate required decode or encode resources only when needed: Input, Output Picture Images, DPB, Stream, and Command buffers
  - Delete objects and remove the backing physical memory as soon as possible
  - Reuse resources when content size changes



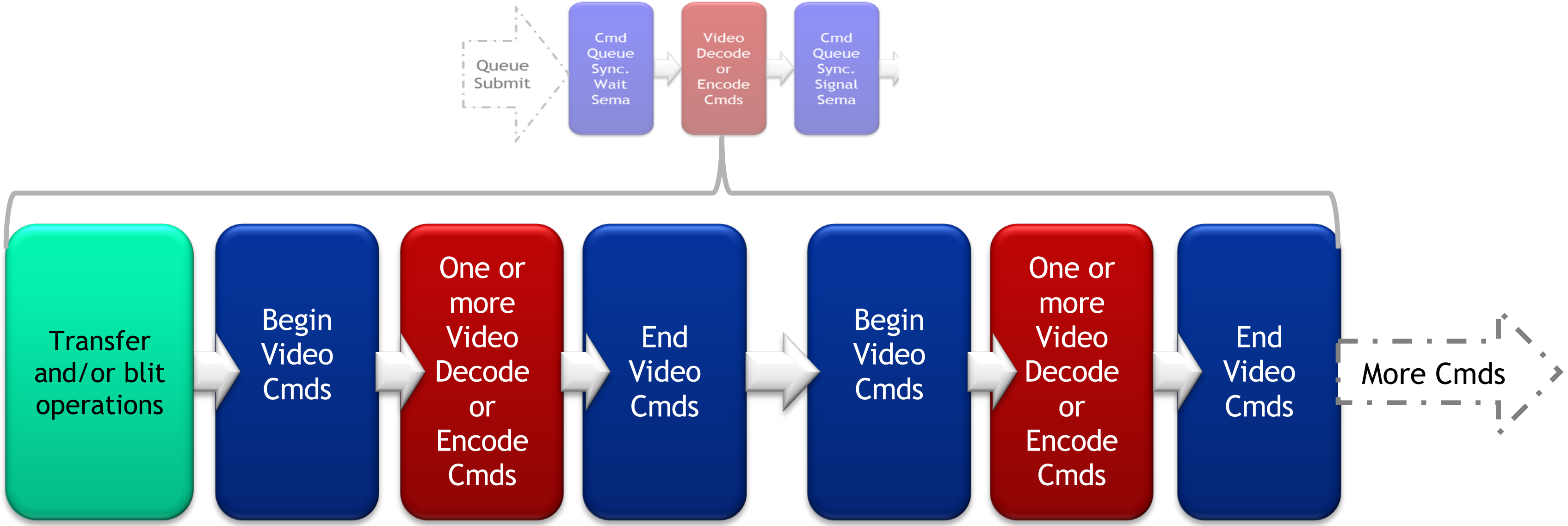
Video VkCommandBuffers Queue Submission

- **Regular Vulkan Queue Submit Sequence:**
  - One or more Recorded Vulkan Video Command Buffers can be submitted
  - The command buffer sequences can be synchronized by binary or timeline semaphores
  - The command buffer sequences can be synchronized with the host CPU via semaphores or a fence


Queue Submit
Cmd Queue Sync. Wait Sema
Video Decode or Encode Cmds
Cmd Queue Sync. Signal Sema
Cmd Queue Sync. Wait Sema
Video Decode or Encode Cmds
Cmd Queue Sync. Signal Sema
More Cmd Buffers
Cmd Queue Sync. Fence

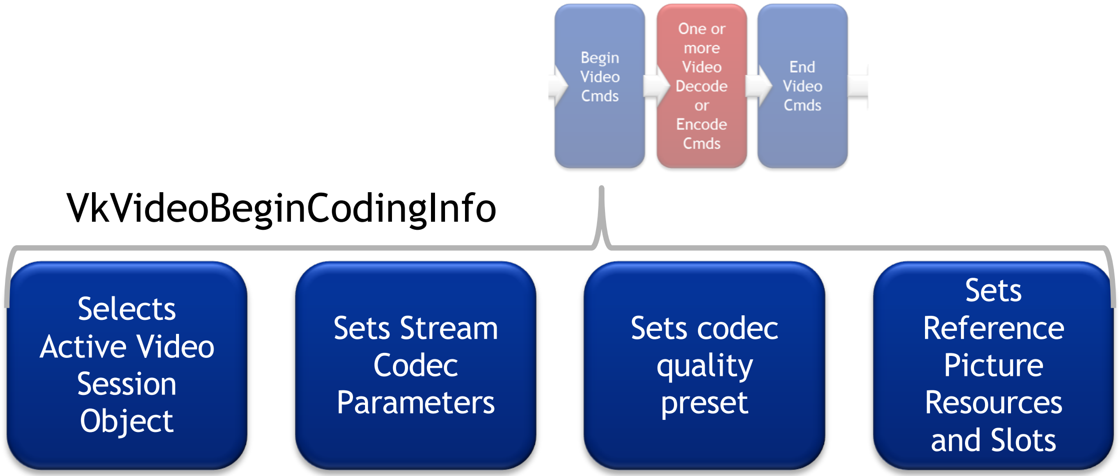
VkCommandBuffers Video Recording Sequence

- All Vulkan Video command sequences start with vkCmdBeginVideoCoding and end with vkCmdEndVideoCodingKHR
- Multiple Video Start/End command sequences are supported
- Implicit ordering guarantees also apply to the video and other commands that belong to the same command buffer


Queue Submit
Cmd Queue Sync. Wait Sema
Video Decode or Encode Cmds
Cmd Queue Sync. Signal Sema
Transfer and/or blit operations
Begin Video Cmds
One or more Video Decode or Encode Cmds
End Video Cmds
Begin Video Cmds
One or more Video Decode or Encode Cmds
End Video Cmds
More Cmds

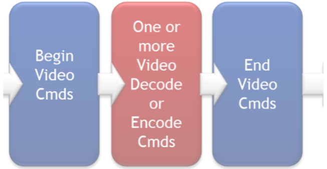
VkCommandBuffers Recording Context Setup

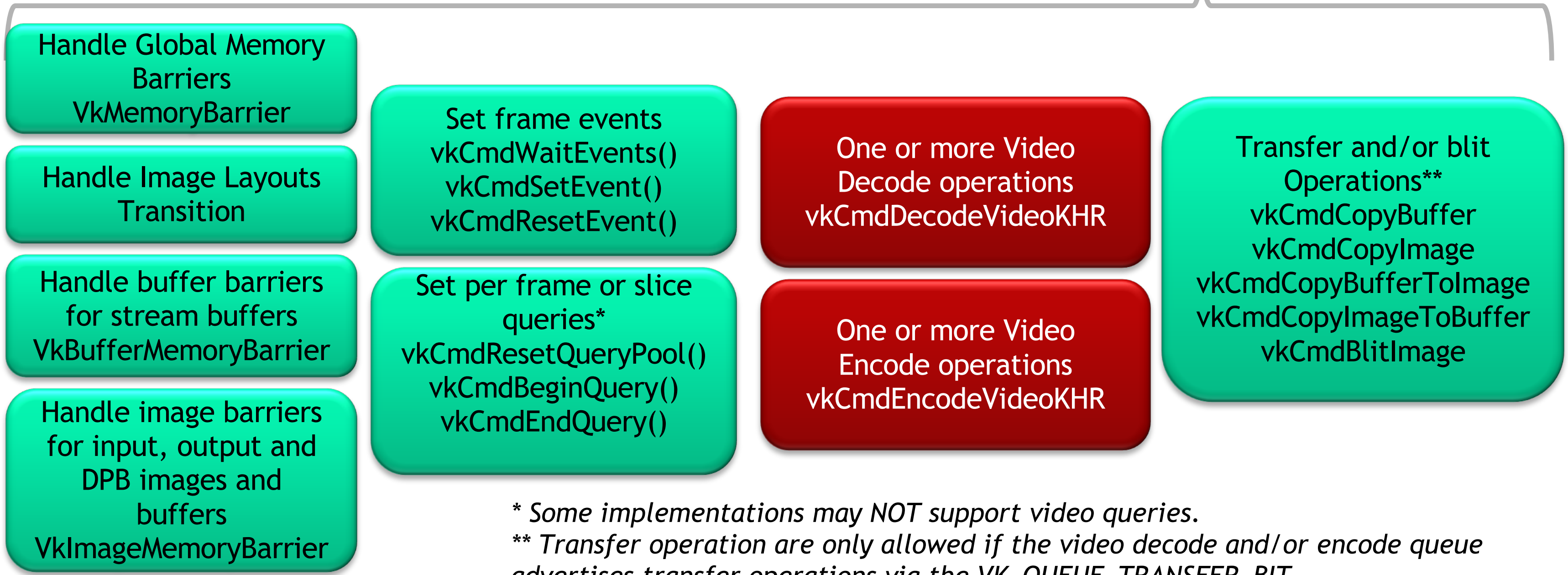
- vkCmdBeginVideoCoding via the VkVideoBeginCodingInfo parameters establishes a context for the subsequent video decode and/or encode commands
- vkCmdEndVideoCodingKHR terminates the context established by the last vkCmdBeginVideoCoding


Begin Video Cmds
One or more Video Decode or Encode Cmds
End Video Cmds
VkVideoBeginCodingInfo
Selects Active Video Session Object
Sets Stream Codec Parameters
Sets codec quality preset
Sets Reference Picture Resources and Slots

Recording VkCommandBuffer Commands

Only commands for decode/encode, barriers/events/query and transfer** operation are supported between VkBeginVideoCoding and VkEndVideoCoding





Handle Global Memory Barriers
VkMemoryBarrier

Handle Image Layouts Transition

Handle buffer barriers for stream buffers
VkBufferMemoryBarrier

Handle image barriers for input, output and DPB images and buffers
VkImageMemoryBarrier

vkCmdPipelineBarrier

Set frame events
vkCmdWaitEvents()
vkCmdSetEvent()
vkCmdResetEvent()

Set per frame or slice queries*
vkCmdResetQueryPool()
vkCmdBeginQuery()
vkCmdEndQuery()

One or more Video Decode operations
vkCmdDecodeVideoKHR

One or more Video Encode operations
vkCmdEncodeVideoKHR

Transfer and/or blit Operations**
vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdBlitImage

** Some implementations may NOT support video queries.*
*** Transfer operation are only allowed if the video decode and/or encode queue advertises transfer operations via the VK_QUEUE_TRANSFER_BIT*

Special Image Layout Transitions

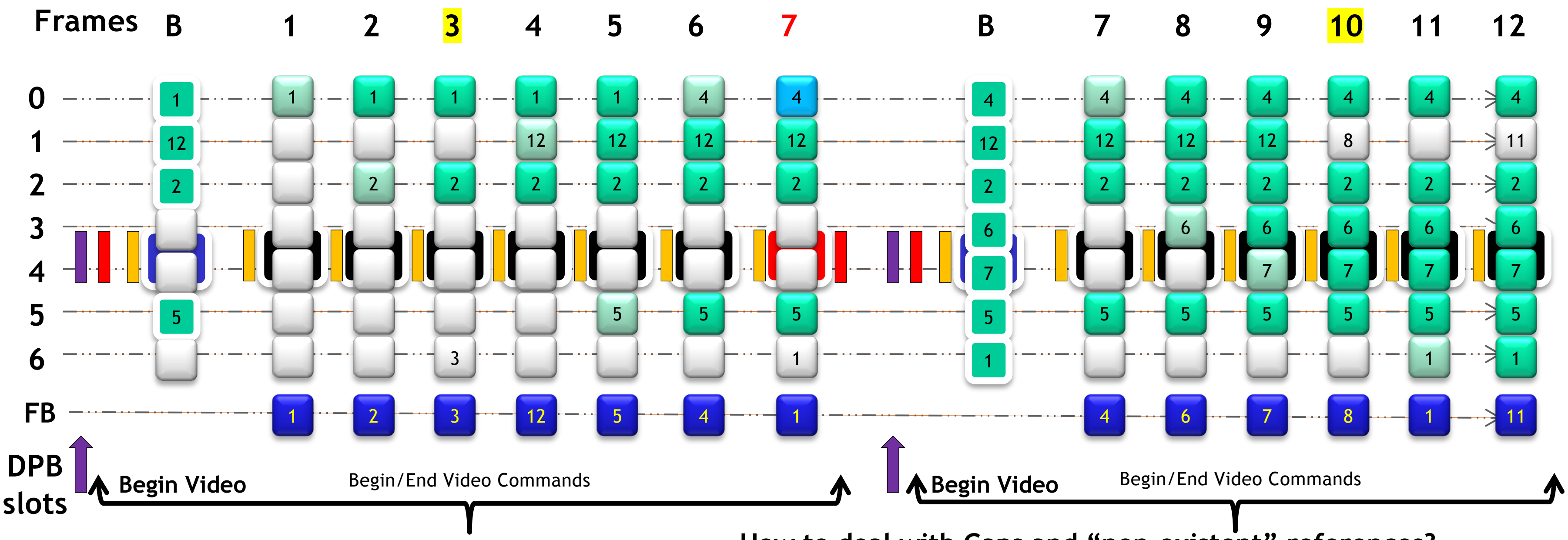
- **DPB image special handling**
  - DPB images implicitly transition to VK_IMAGE_LAYOUT_UNDEFINED when:
    - Image is used for the first time with a video session
    - Content size or other parameters change within a video session
    - VkVideoSession object is reset
    - DPB slot is assigned for the first time with the image view representing the image
    - Video may need a structure like VkSampleLocationsInfoEXT to simplify those rules?
  - DPB images layout should not be affected when:
    - Transitioning reference images from VK_IMAGE_LAYOUT_VIDEO_DECODE_DPB or VK_IMAGE_LAYOUT_VIDEO_DECODE_DST to Gfx/compute friendly layouts
- **Video Input images transition**
  - When the content size or parameters change encode input images implicitly transition from VK_IMAGE_LAYOUT_VIDEO_ENCODE_SRC to VK_IMAGE_LAYOUT_UNDEFINED or VK_IMAGE_LAYOUT_PREINITIALIZED

DPB Slot Management

- **Allocating/Associating DPB reference slots with *slotId***
  - Add entry with slotId and associated VkImage resource in the array of VkVideoBeginCodingInfoKHR::pReferenceSlots within the vkCmdBeginVideoCoding command
- **Making DPB reference slot with slotId valid**
  - Decode (vkCmdDecodeVideoKHR) or Encode (vkCmdEncodeVideoKHR) commands targeting slotId within the pSetupReferenceSlot
- **Invalidating DPB slot with slotId**
  - Replace association of the reference slot with slotId with a different VkImage resource
  - Decode or Encode commands targeting the reference slot with slotId (with pSetupReferenceSlot)
  - Reset the decoder/encoder
  - Replace the content of the associated VkImage resource or unbind the backing memory
  - Change the layout of the associated VkImage resource to an incompatible layout

DPB Slot Management Example

Allocating slots with associated picture resources



Frames	B	1	2	3	4	5	6	7	B	7	8	9	10	11	12
0	1	1	1	1	1	1	4	4	4	4	4	4	4	4	4
1	12				12	12	12	12	12	12	12	12	8		11
2	2		2	2	2	2	2	2	2	2	2	2	2	2	2
3									6		6	6	6	6	6
4									7			7	7	7	7
5	5					5	5	5	5	5	5	5	5	5	5
6				3				1	1					1	1
FB		1	2	3	12	5	4	1		4	6	7	8	1	11

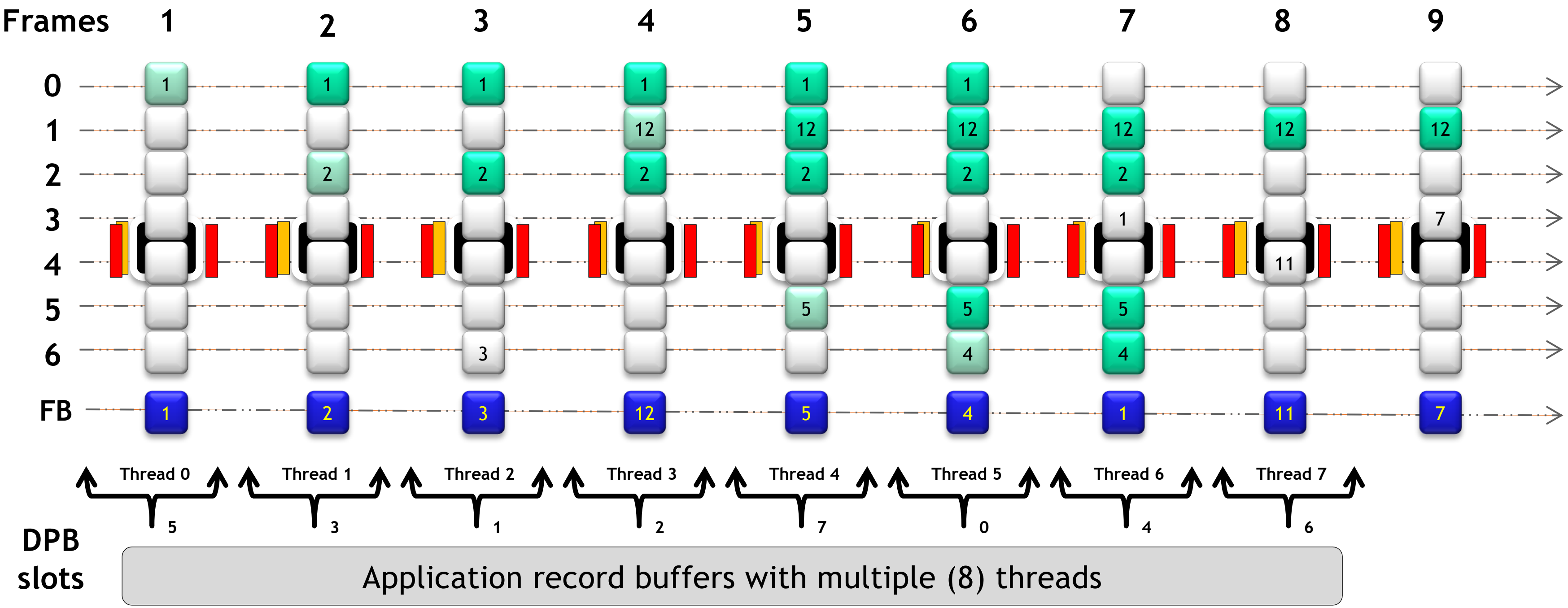
How to deal with Gaps and “non-existent” references?

Note: Sparse may require a different queue family to submit, if the video queue families do not advertise sparse capabilities

⬜ Available DPB slot   🟩 Allocated Target   🟩 Reference   🟦 Current FB   🟨 Image and buffer Memory Barriers   🟥 New Begin/end sequence   🟪 Manage Image/buffer Sparse Bindings

DPB Slots

Multi-threaded cmdBuffer Recording



Frames	1	2	3	4	5	6	7	8	9
0	1	1	1	1	1	1			
1				12	12	12	12	12	12
2		2	2	2	2	2	2		
3							1		7
4								11	
5					5	5	5		
6			3			4	4		
FB	1	2	3	12	5	4	1	11	7

Is it valid to perform the above multi-threaded command buffer recording?

⬜ Available DPB slot 🟩 Allocated Target 🟩 Reference 🟦 Current FB 🟨 Image and buffer Memory Barriers 🟥 New Begin/end sequence 🟪 Manage Image/buffer Sparse Bindings

Video Queries

- **Result Status Query (optional)**
  - Used to check whether a set of operations has been completed successfully
  - Type is VK_QUERY_RESULT_WITH_STATUS_BIT_KHR
  - Can be used with other than video queue families
- **Encode Bitstream Range Query**
  - Describes range of bytes written in the bitstream buffer by video encode commands
  - Type is VK_QUERY_TYPE_VIDEO_ENCODE_BITSTREAM_BUFFER_RANGE_KHR
- **Queries supported with Video**
  - Host side vkGetQueryPoolResults()
- **Queries not supported with Video**
  - Device side: vkCmdCopyQueryPoolResults()

Video Properties and Capabilities

- **Supported codecs for a particular Vulkan video queue**
  - Queried through VkVideoQueueFamilyProperties2KHR, chained to vkGetPhysicalDeviceQueueFamilyProperties() function
- **Supported video decode and encode capabilities**
  - Queried through vkGetPhysicalDeviceVideoCapabilitiesKHR() function
- **Supported video output, input and DPB image formats**
  - Enumerated through vkGetPhysicalDeviceVideoFormatPropertiesKHR() function