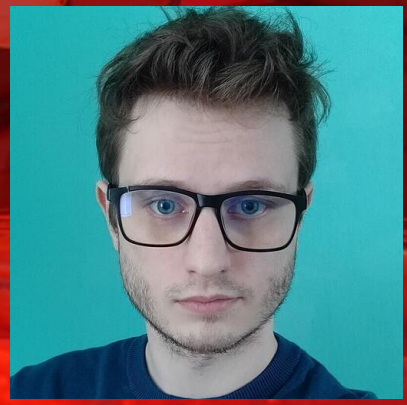
Vulkanised 2025

The 7th Vulkan Developer Conference
Cambridge, UK | February 11-13, 2025

Swapchains Explained: How to manage them effectively

Darius Bozek, Samsung



Contents

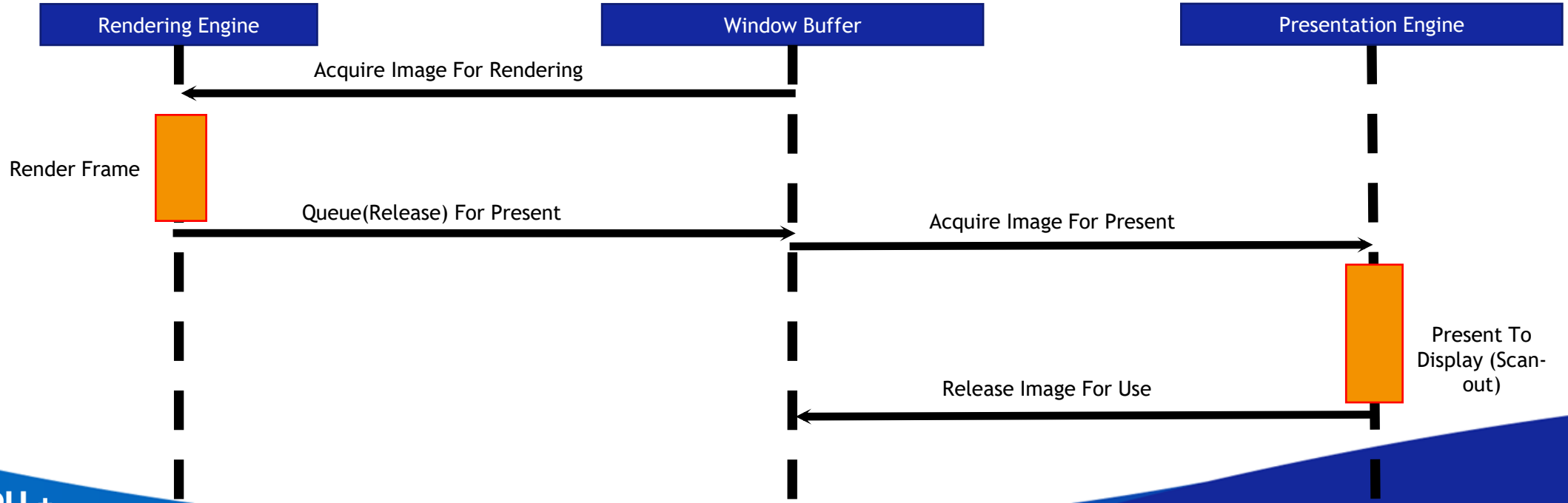
1. Swapchain
2. Pre-rotation
3. Maintenance
4. Frame Pacing

Swapchain

Just what is it?

How to get pixels on the screen

- Two separate processes use a shared resource
- **Rendering Engine(RE)** (e.g. Vulkan) renders an image to the window buffer- Communication happens through swapchain
- **Presentation Engine(PE)** present windows buffer to the screen - Communication happens outside of Graphics API jurisdiction



Swapchain – Glossary

Presentable Image - A VkImage object obtained from a VkSwapchainKHR used to present to a VkSurfaceKHR object.

VkSwapchain - an abstraction for an array of presentable images that are associated with a surface.

VkSurface - Abstracted native platform surface or window objects.

Retired Swapchain - A swapchain that has been used as the oldSwapchain parameter to vkCreateSwapchainKHR. Images cannot be acquired from a retired swapchain, however images that were acquired (but not presented) before the swapchain was retired can be presented.

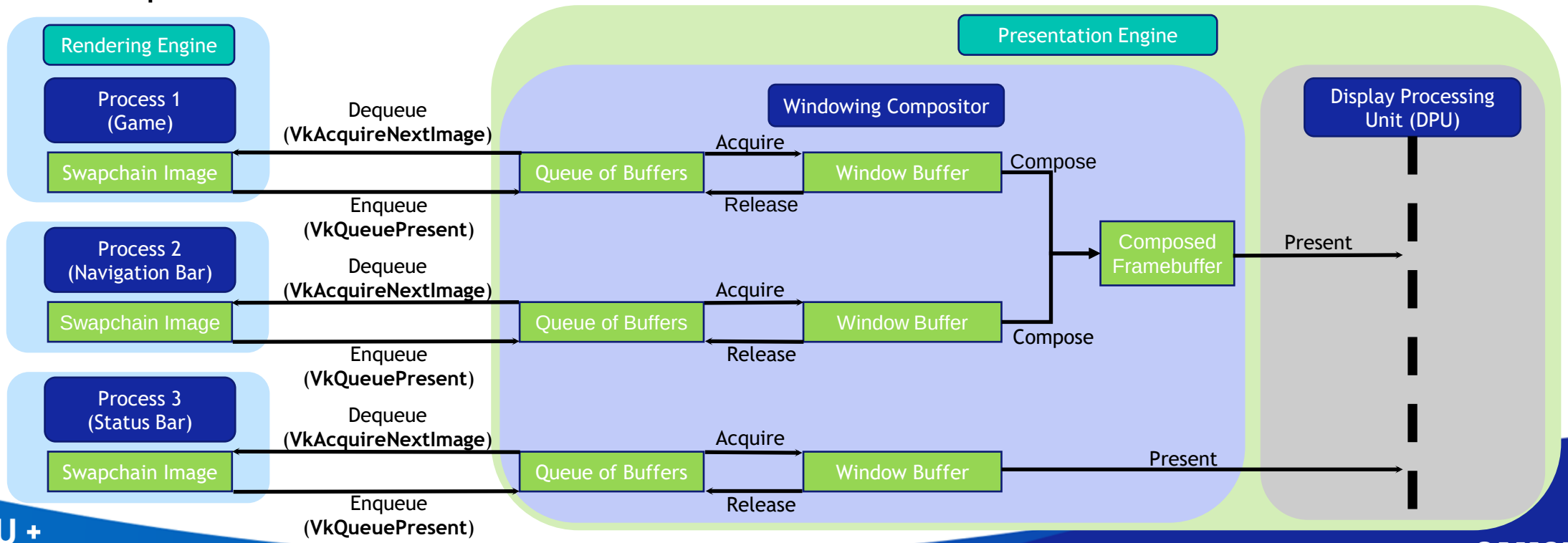
Swapchain – Memory Abstraction

vkCreateSwapchainKHR() doesn't always allocate memory for the images

- Window memory is created/destroyed by PE – (e.g. in android: ANativeWindow)
- Both PE and RE can acquire read/write priority of the memory
- RE acquires priority through the use of swapchain
- The window memory may not be freed when destroying swapchain, it may be deferred until after the presentation engine is no longer using it
- vkAcquireNextImageKHR() gets an index to an image and a sync point indicating when it can be used
- Destroying the VkSwapchain doesn't invalidate the VkSurfaceKHR

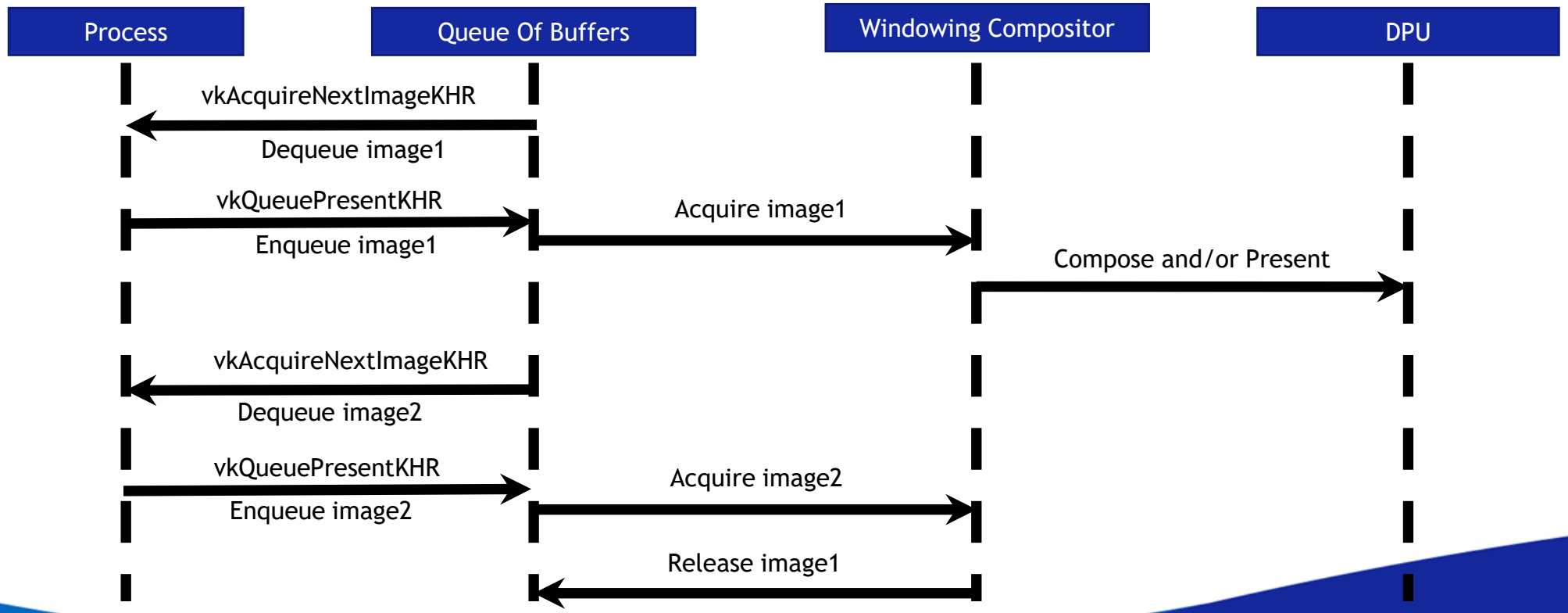
Presentation Engine Composition

- Multiple processes will present their swapchains to Windowing Compositor
- Windowing Compositor will do **D**(isplay)**PU** composition and/or **GPU** composition
- DPU composition is pass through to DPU, GPU composition is done by Windowing Compositor



Presentation Engine Synchronisation

- Windowing Compositor will always* hold **at least** 1 image per app for composition (*on Android)

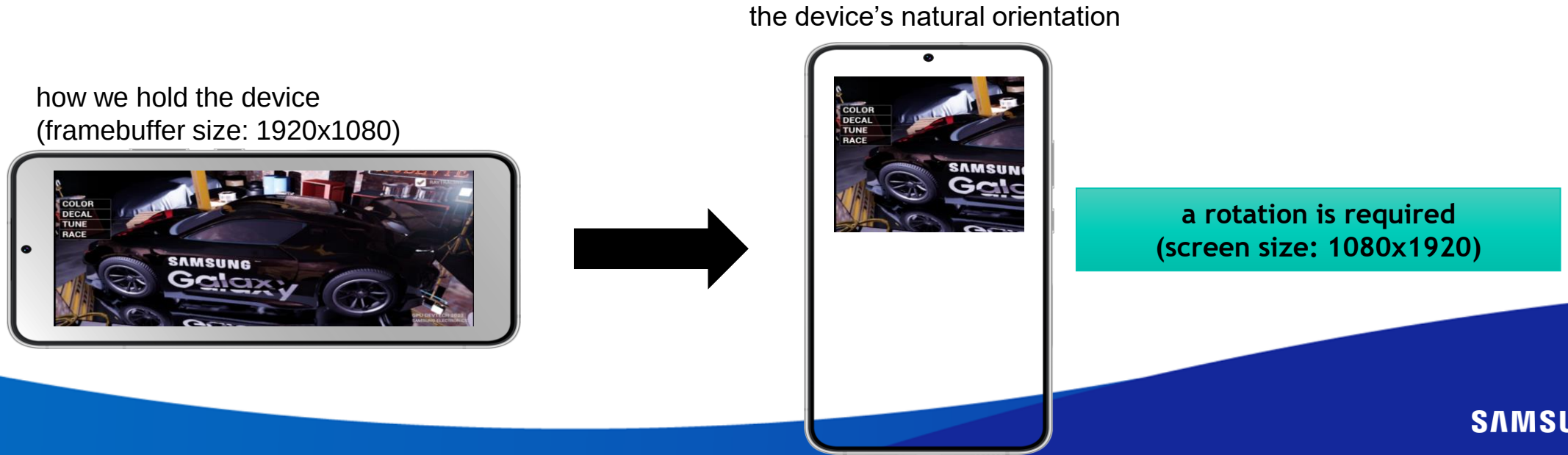


Pre-Rotation

Is it worth it?

What is Pre-rotation and why should I care?

- Android devices support multiple rotations(Usually 4*)
- Surface orientation may be different than the device natural orientation
- The difference in orientation should be accounted for somewhere in the rendering pipeline
- If the rendering pipeline doesn’t account for it the presentation engine will - That may have performance penalty
- Device natural orientation is **not always Portrait mode**!
- Some devices will have more than one natural orientation



Which Transform is which?

Surface:

- VkSurfaceCapabilitiesKHR surfaceCapabilities.currentTransform
- This is the current transform of the surface, it describes what rotation is applied to the surface from natural orientation

Swapchain:

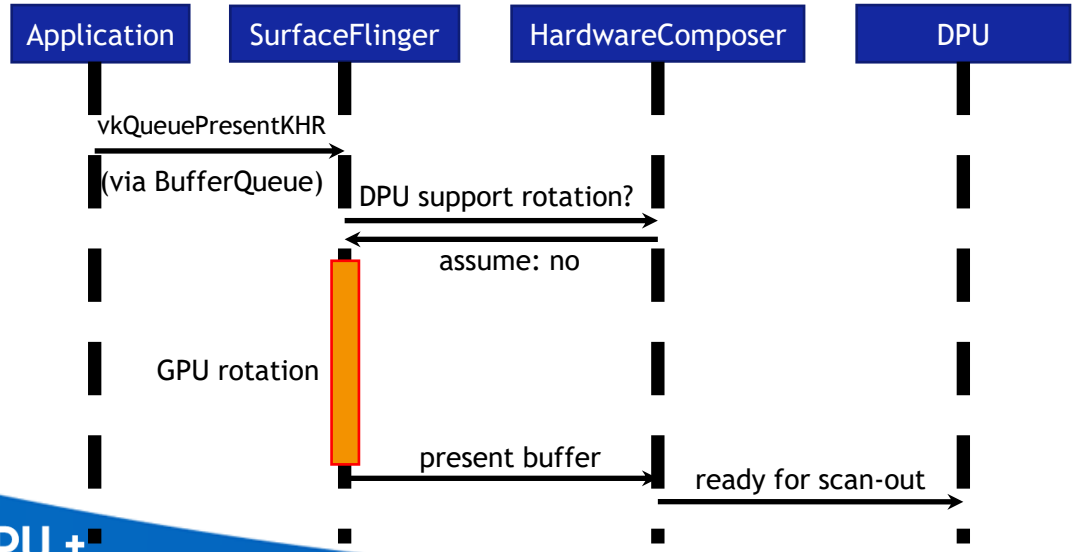
- VkSwapchainCreateInfoKHR swapchainCreateInfo.preTransform
- This describes what rotation the application applied to the image, if it doesn't match surface current transform the presentation engine will have to perform rotation

Suboptimal - Why is it returned?

- DPU units may or may not have HW support for all rotations (You can't check for support)
- Vulkan calls* will return SUBOPTIMAL when Swapchain doesn't match Surface rotation **even if DPU supports that rotation** - Either way Suboptimal means the application should recreate

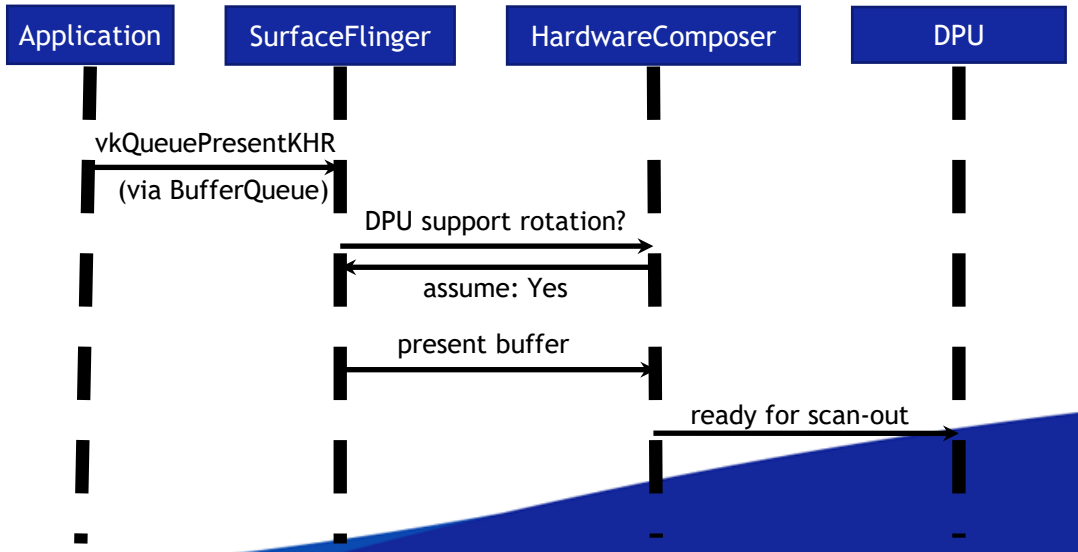
Rotation Not Supported

Returns VK_SUBOPTIMAL_KHR



Rotation Supported

Returns VK_SUCCESS or VK_SUBOPTIMAL_KHR
Success is only returned when surface.currentTransform == swapchain.preTransform



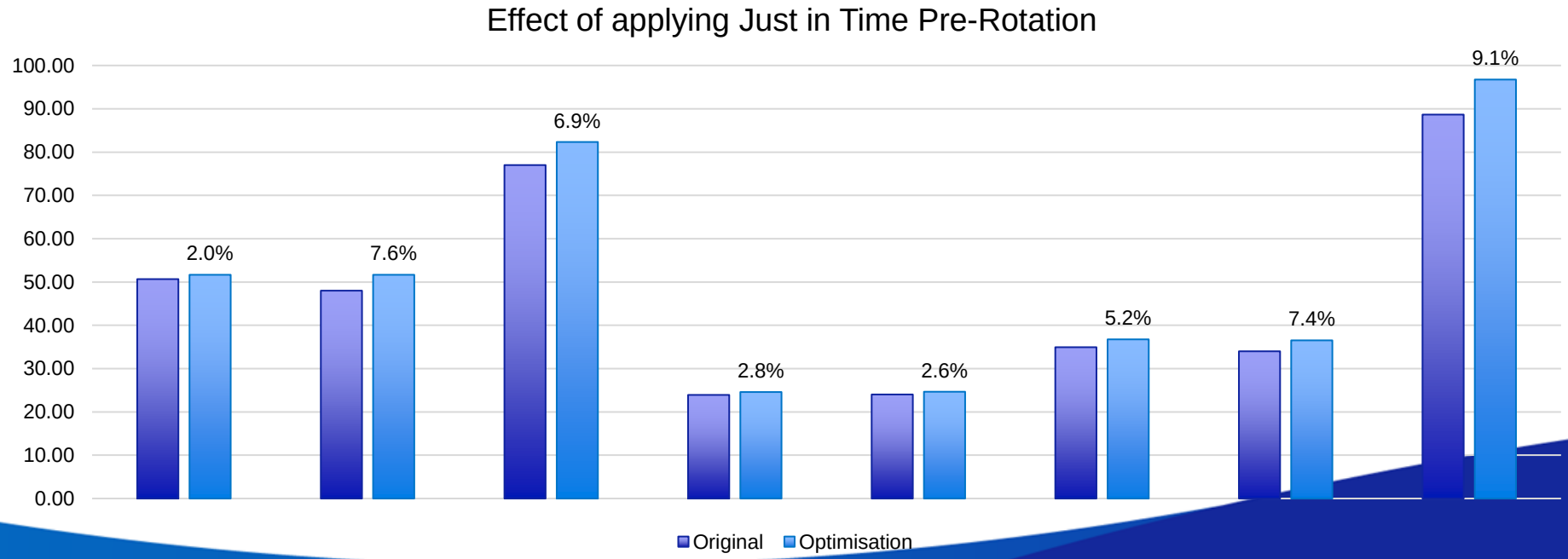
Just rewrite your whole code base lol

The consensus on best practice is to use pre-rotate – so why don't developers do it?

- Potentially having to rewrite most shaders + engine is not feasible
- There are not a lot of resources on *how* to pre-rotate correctly, just general advice to do it
- Many resources that do explain how to do it usually have minor bugs or are not scalable
- Requires all shaders to have at least 4x variants or use uniforms – this can blow up pipeline cache if specialization constants are used
- Fragment Shader Derivatives
- Compute shaders with pixel coordinates….

Just in Time Pre-Rotate

- Instead of keeping track of all shaders and states it is possible to sacrifice some potential performance gains for simplicity
- **Rotate the final image** before presenting, all it takes is **one** Renderpass - Can piggyback on the final renderpass if possible
- Simple & Effective. Up to 9.1% performance improvement in benchmarks and games



Maintenance

When and how to recreate Swapchain?

When To Destroy?

When can swapchain be destroyed:

- “The application must not destroy a swapchain until after completion of all outstanding operations on images that were acquired from the swapchain.”

How to wait on “all outstanding operations”:

- “vkQueueWaitIdle is equivalent to having submitted a valid fence to every previously executed queue submission command that **accepts a fence**, then waiting for all of those fences to signal using vkWaitForFences with an infinite timeout and waitAll set to VK_TRUE”
- “vkDeviceWaitIdle is equivalent to calling vkQueueWaitIdle for all queues owned by device.”

In reality calling vkDevice/QueueWaitIdle is fine on all* implementations when tearing the application down. In case of recreation it is a bit more complicated…

When To Recreate - Fail

When Creation, Acquisition or Presentation fails or…

vkCreateSwapchainKHR

On failure, this command returns

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_DEVICE_LOST
- VK_ERROR_SURFACE_LOST_KHR
- VK_ERROR_NATIVE_WINDOW_IN_USE_KHR
- VK_ERROR_INITIALIZATION_FAILED
- VK_ERROR_COMPRESSION_EXHAUSTED_EXT

vkAcquireNextImageKHR

On failure, this command returns

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_DEVICE_LOST
- VK_ERROR_OUT_OF_DATE_KHR
- VK_ERROR_SURFACE_LOST_KHR
- VK_ERROR_FULL_SCREEN_EXCLUSIVE_MODE_LOST_EXT

vkQueuePresentKHR

On failure, this command returns

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_DEVICE_LOST
- VK_ERROR_OUT_OF_DATE_KHR
- VK_ERROR_SURFACE_LOST_KHR
- VK_ERROR_FULL_SCREEN_EXCLUSIVE_MODE_LOST_EXT

When To Recreate - Suboptimal

On **Success** those commands return:

- VK_SUCCESS
- VK_SUBOPTIMAL_KHR

VK_SUBOPTIMAL_KHR is a success return code, however it indicates that performance could be improved. Any application that cares about performance should handle this.

When To Recreate - Android Hooks

Possible entry points for android activity:

- onResume()
- onPause()
- onOrientationChanged()
- onNativeWindowResized()

However in our experience there are potential issues:

- Android synchronisation
- Knowing when hooks are called(i.e. onNativeWindowResized is not called when orientation changes by 180°)
- Having to recreate the workflow in multiple places if other Vulkan platforms are supported

If swapchain is maintained with use of android hooks then foldable devices **need** onNativeWindowResized() to be used. This is what is called when the screen is folded/unfolded. Be careful as **small screen uses portrait*** as default orientation and **big screen uses landscape***

How To Recreate – Fail

- Straight forward
- Just recreate as soon as possible

Common Mistakes When Recreating

Why is DeviceWaitIdle not enough:

- Swapchain images that were acquired before destruction but were submitted after
- Not being able to release swapchain image after acquiring other than by presenting*
- Multithreading…

VkAcquireImage(OldImage)

VkDeviceWaitIdle

VkDestroySwapchain

VkCreateSwapchain

VkQueuePresent(OldImage)

No operations are allowed on images from a destroyed swapchain

VkAcquireImage(OldImage)

VkCreateSwapchain(oldSC = currentSC)

VkDestroySwapchain(oldSC)

VkQueuePresent(OldImage)

Using old swapchain to create new one is not reusing images. Old Swapchain **should*** be deleted after all acquired images are presented

VkCreateSwapchain(oldSC = currentSC)

VkAcquireImage(oldSC)

VkQueuePresent(OldImage)

It’s not allowed to acquire images from retired swapchain

VK_EXT_swapchain_maintenance1

- Allows the release of an acquired swapchain image without presenting – Easier to recreate
- Adds present fence – Allows the application to know when exactly to destroy swapchain
- Allows the deferral of swapchain image memory allocation – Slightly helps with startup times and can help with lower peak memory usage
- Introduced in 2023 - Only ~25% of android devices and ~20% of desktop GPUs use it (vulkan.gpuinfo.org)
- For Android it was first added to Android 14 – You should **always** use it **if** you can
- But in case you target older devices you will have to do it the hard way…

How To Recreate – Suboptimal

Two possible approaches to recreate swapchain

Flush And Recreate

- Finish recording, submit all pending commands and block recording new ones
- VkDeviceWaitIdle
- VkDestroySwapchainKHR(currentSwapchain)
- VkCreateSwapchainKHR(oldSwapchain == nullptr)
- Unlock recording new commands

Recreate and Check

- VkCreateSwapchainKHR(oldSwapchain==currentSwapchain)
- Keep track of all current passes that are still using old swapchain
- New passes to use new swapchain
- Wait for all old passes that acquired images from old swapchain to present
- VkDestroySwapchainKHR(oldSwapchain)

Flush And Recreate

Issues:

- Requires GPU to finish all pending work - This is very noticable when recreating multiple times
- Blocks recording on the CPU at the point of AcquireNextImage
- CreateSwapchain will fail if surface is still associated with old swapchain
- Any operation, on images acquired from a swapchain that was then deleted, will fail
- The sooner in the frame the AcquireNextImage is done the bigger the penalty – can be helped by deferring acquisition until late
- VK_ERROR_NATIVE_WINDOW_IN_USE_KHR: Wrong "Fix"-> recreate the surface. Real fix -> keep track of all swapchains and delete before creation

Benefits:

- Only one live swapchain at any given point – lower peak memory usage
- It **has** to be right or it won't work – no leaks allowed

Recreate and Check

Issues:

- VkImages from oldSwachain **do not** transfer to new swapchain - Application can't destroy old swapchain until after completion of all outstanding operations on acquired images
- **Can’t acquire** more images from old swapchain. Already acquired images **remain acquired**
- Requires more bookkeeping to know when old swapchain can be destroyed
- Acquired images from old Swapchain can temporarily increase memory utilization
- Tutorials sometimes include bugs that will skip destroying old swapchain - use Vulkan-Samples swapchain-recreation
- There is no simple way to know when a semaphore used for presentation can be reused*

Benefits:

- Able to create and use new swapchain straight away – lower latency
- Old images that have been acquired by application can still be presented
- Exclusive full-screen access will be transferred to new swapchain if used
- Internally the non-acquired VKImages associated with the old swapchain may be destroyed immediately

Frame Pacing

zzz

Presentation Modes

VK_PRESENT_MODE_IMMEDIATE_KHR:

- Doesn't wait for Vsync
- May result in tearing
- No internal queue is used

VK_PRESENT_MODE_MAILBOX_KHR

- Waits for Vsync
- No tearing
- Single-entry internal queue is used. If full then new request replace existing
- **Users may perceive this as stutter**

VK_PRESENT_MODE_FIFO_KHR:

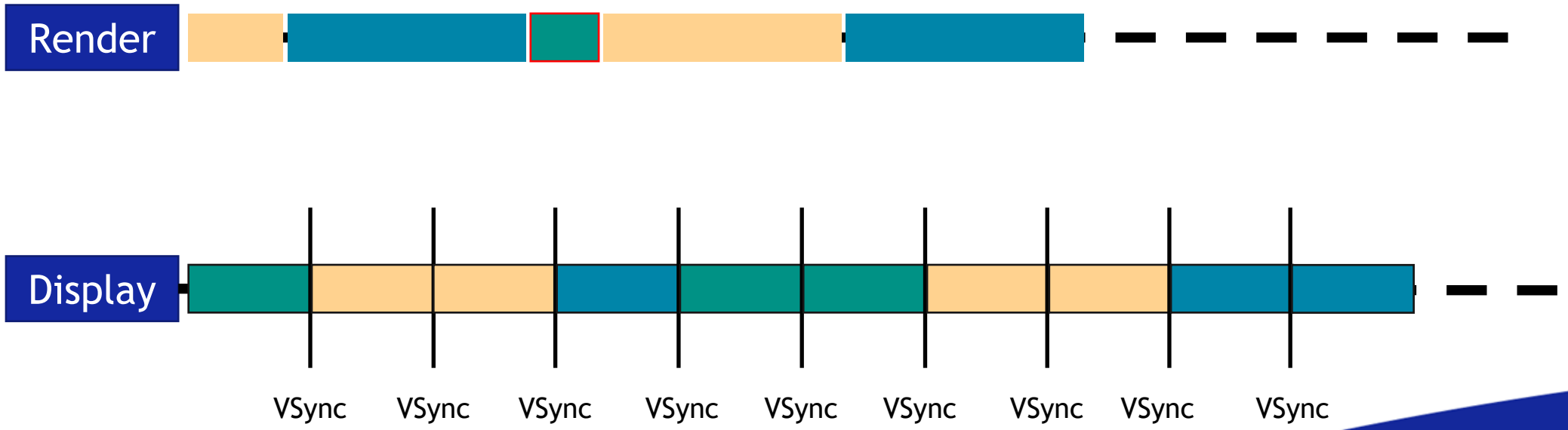
- Waits for Vsync
- No tearing
- Internal queue is used. New requests are appended to the back.
- **The only mode that is required to be supported**

VK_PRESENT_MODE_FIFO_RELAXED_KHR

- Waits for Vsync once per request
- May result in tearing
- Internal queue is used. New requests are appended to the back

Frame Stutter – Why does it happen?

- Frame Stutter happens because frames take different amount of time to render – Frame Pacing helps
- Reported 60/30 doesn't guarantee lack of frame stutter – FPS is an average
- Stutter means that frames are not presented for the same amount of time
- Note: Rendering engine can't rely on main loop synchronization

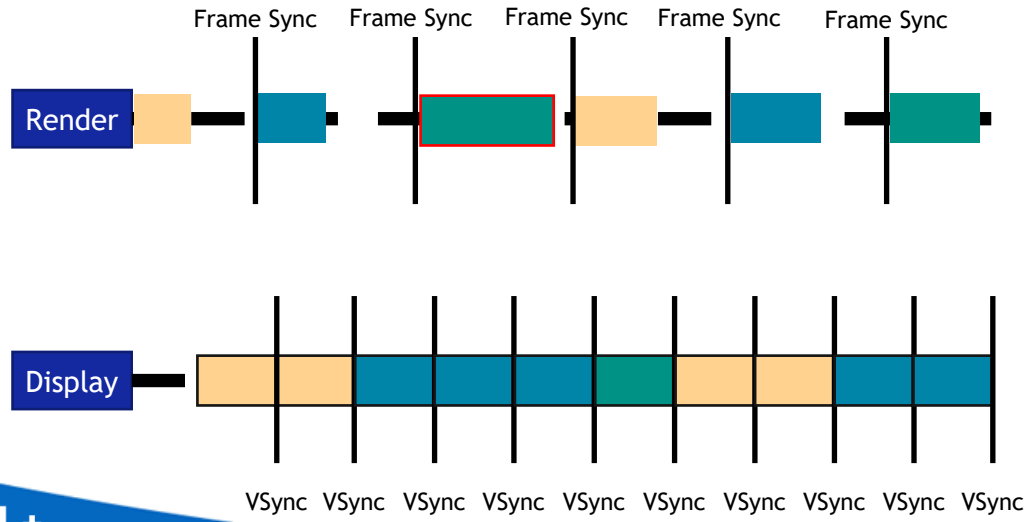


CPU Synchronisation – Why is it not the way to go

- CPU has no way of knowing how long the GPU **will take** to render
- GPU **can't** render in advance
- CPU has no way of knowing when was the frame ready to be presented

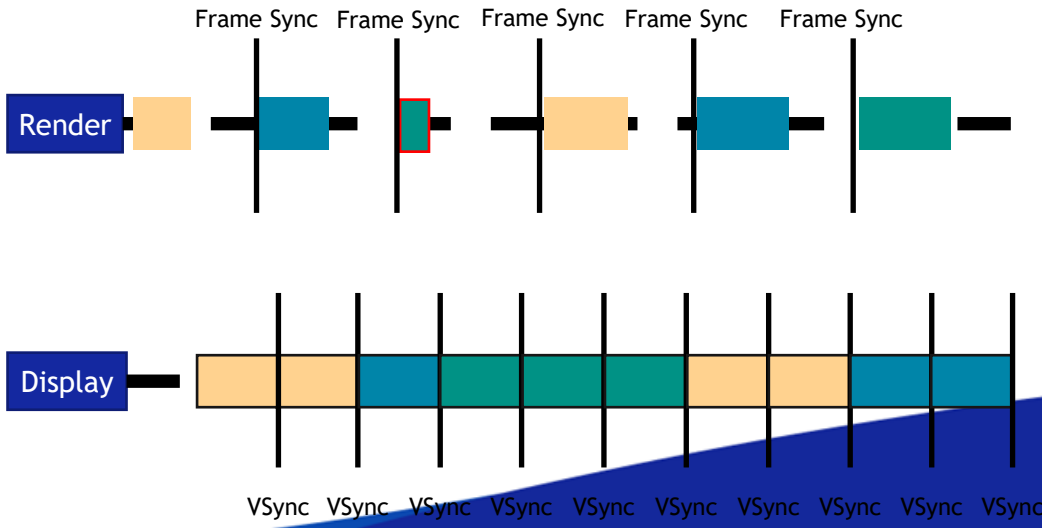
Single Slow Frame

If the next frame is too slow the previous frame has to be displayed for multiple Vsyncs. This is not really stutter as it means frame missed its VSync



Single Fast Frame

If the next frame is too fast previous frame can be displayed for fewer Vsyncs

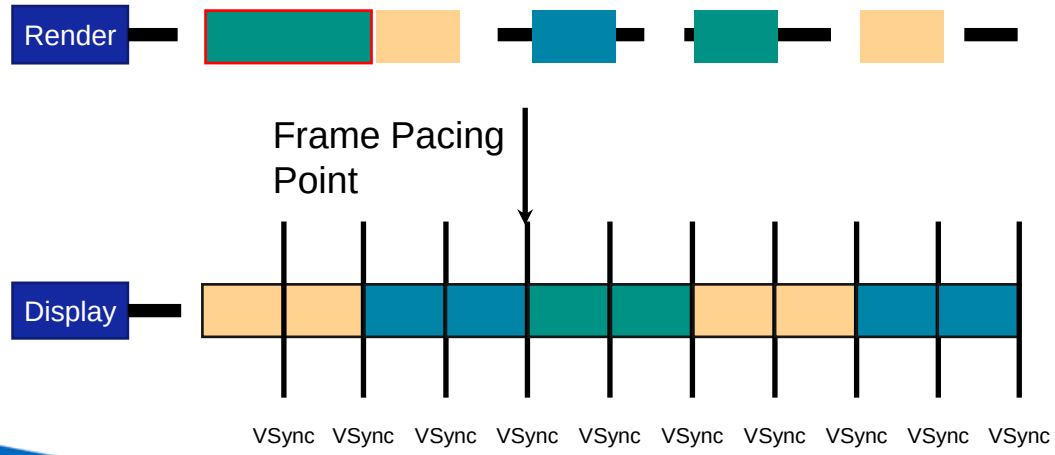


GPU Synchronisation

- CPU can specify when every frame should be displayed
- GPU **can** render in advance
- CPU can query GPU when were frames ready to be presented

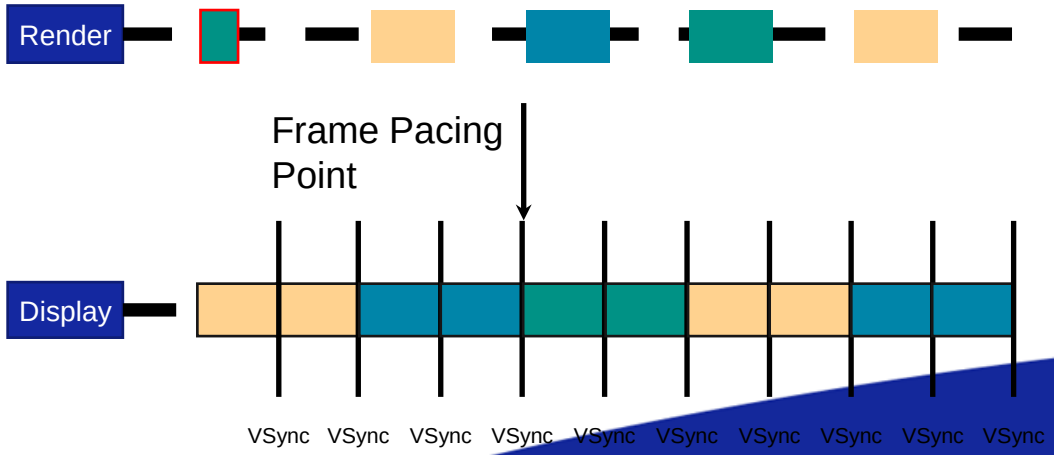
Single Slow Frame

Since we synchronise with presentation time and render as soon as frame is available individual frames taking longer are no longer an issue



Single Fast Frame

We specify when we want the frame to be presented so even though its ready early we still present it at the right time



WSI & VK_GOOGLE_display_timing – Use it

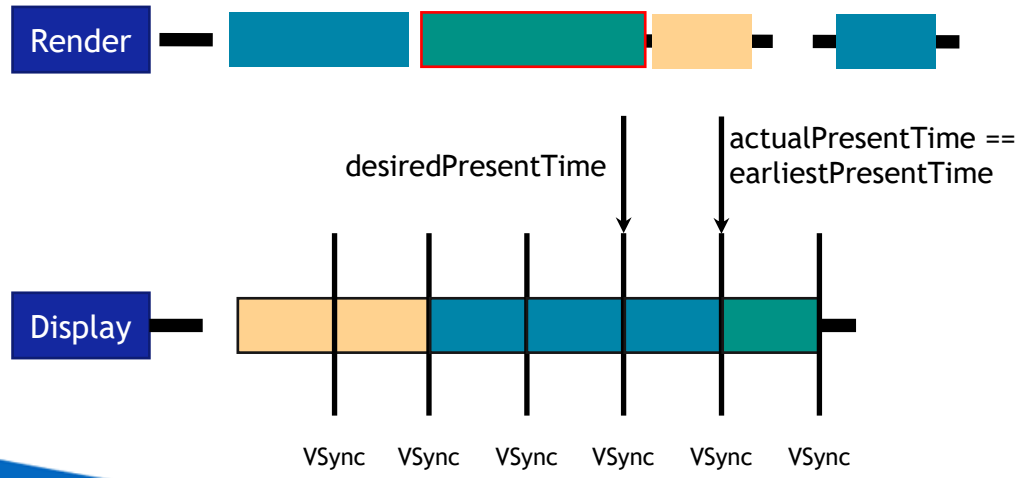
- Display Timing is being shipped as part of Window System Integration (WSI)
- Extends VkPresentInfoKHR to allow specify the earliest time each image should be presented
- Get feedback from GPU on timings of a previously-presented images by using vkGetPastPresentationTimingGOOGLE
- Bug with android – needs to call vkGetPastPresentationTimingGOOGLE with nullptr first otherwise no results will be returned

GetPastPresentationTiming – What is what

- DesiredPresentTime – Time set by the application (you) to indicate that an image not be presented any **sooner** than that time
- ActualPresentTime – Time when the frame was actually presented
- EarliestPresentTime – When the frame **could have** been presented. **May** be earlier than Actual if waiting on Desired

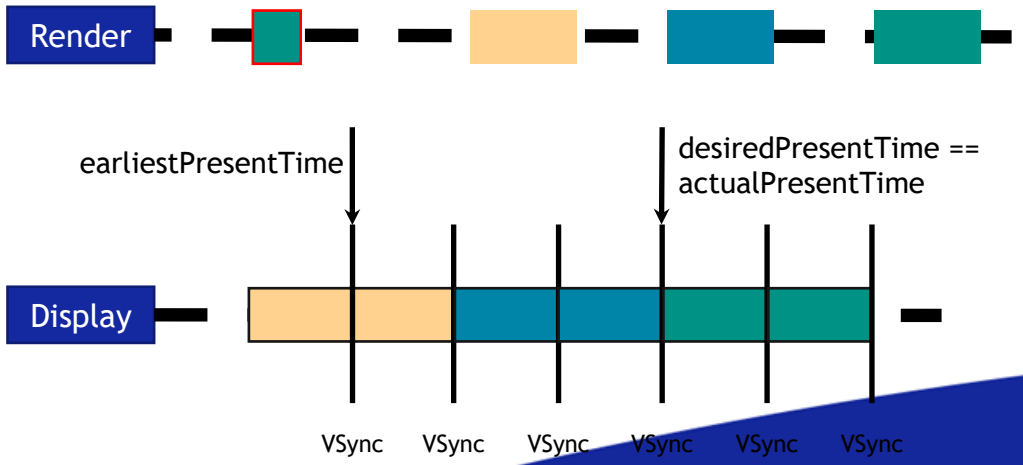
(Actual == Earliest) > Desired

Target FPS was not achieved – If this consistently happens then application should consider lower target FPS



(Actual >= Desired) > Earliest

Frame Pacing working as indented – Frame Rendered in advance but paced to be displayed later



Questions?

Contact info:

gpudev@samsung.com

Thank you