SDC19

Where Now Meets Next

Galaxy GameDev

Vulkanized:

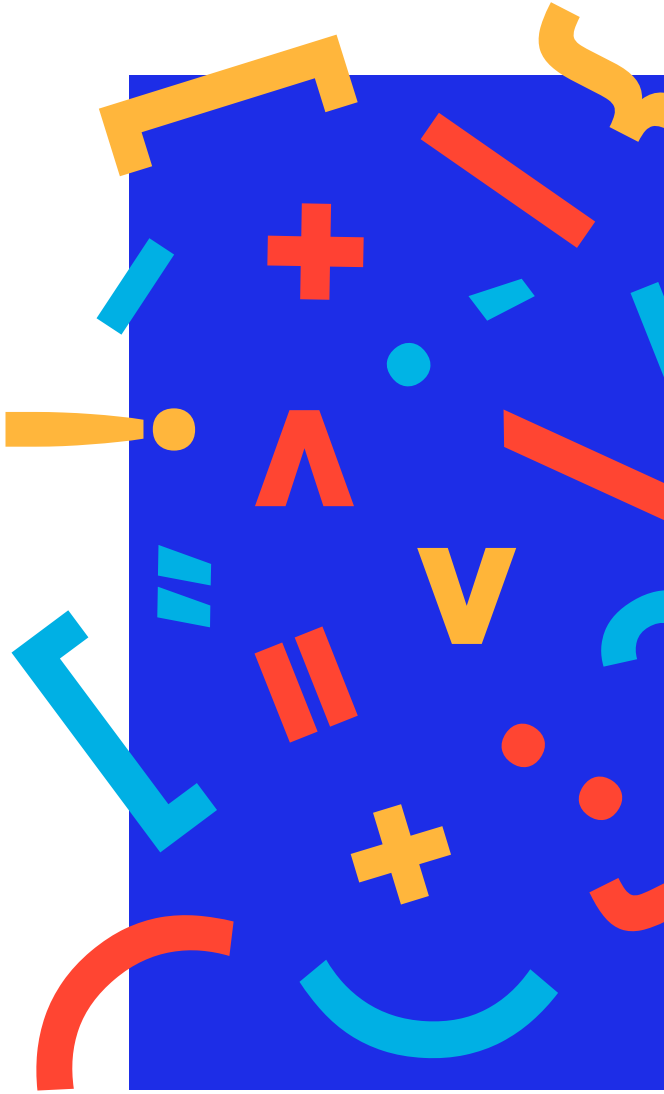
Mobile Game Optimization Techniques

Wei Yao

GameDev Engineer,
Samsung Research China

Igor Nazarov

GameDev Engineer,
Samsung Research Kiev



Galaxy GameDev

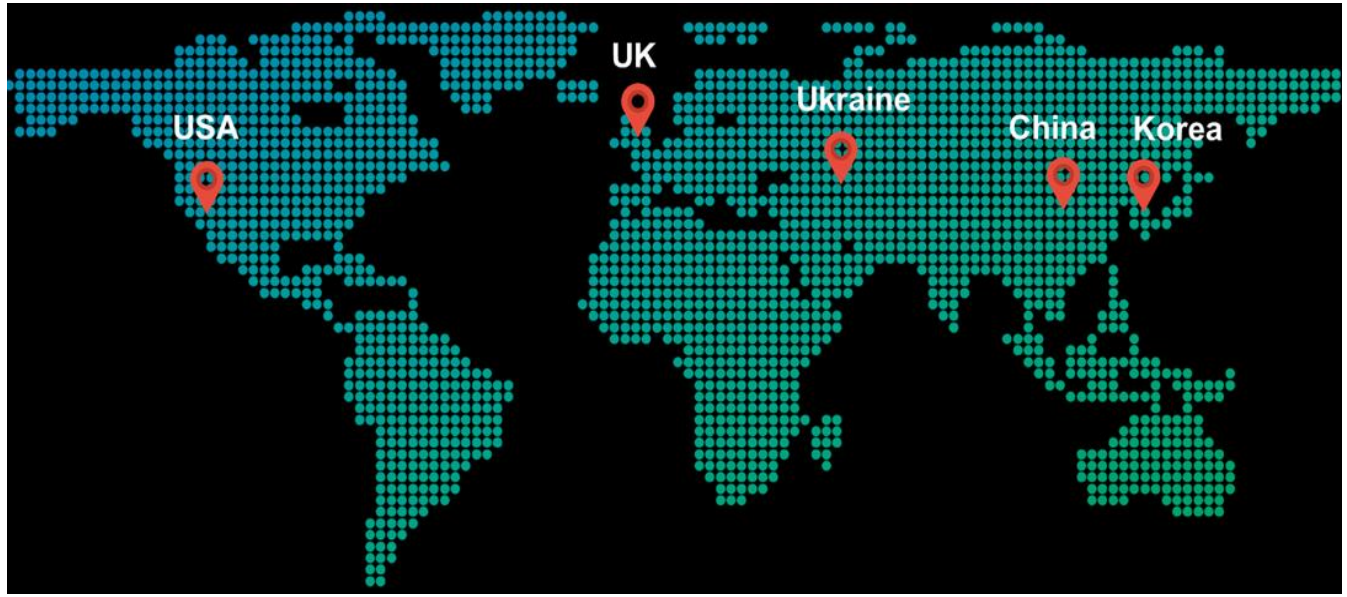
Igor Nazarov

GameDev Engineer, Samsung Research Kiev

Galaxy GameDev

24/7 Mobile Game Developers Support

- Promoting the use of new technologies and features on Android
- On/off-site studios support
- Contribute to game engines
- Profiling and best practices



Vulkan

Unlock Maximum Performance on Android!

- Minimal CPU overhead
- Explicit Control
- Multithreaded
- Cross-platform
- Already has out of the box support by Unity & UE4!

unity

UNREAL ENGINE

CPU & Memory Optimization:

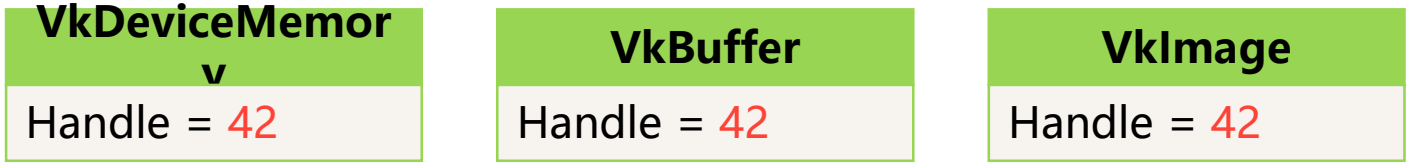
Use Vulkan Object Caching Extensively

- Because Vulkan API is **handle**-based rather than **state**-based like OpenGL API, it is possible to cache frequently used objects;
- Recommended objects for caching are:
  - Pipelines;
  - Descriptor and Pipeline Layouts;
  - Descriptor Sets;
  - Command Buffers;
  - Render Passes;
  - Framebuffers;
  - Shader Modules;
  - Samplers.

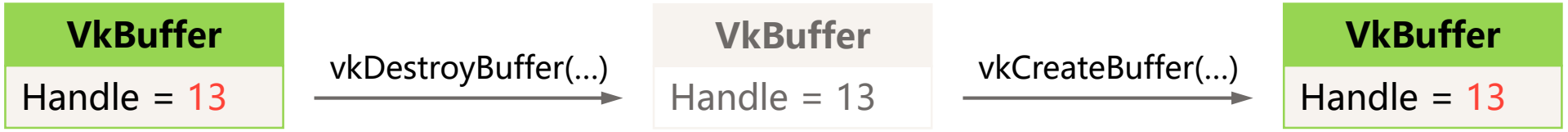
Tip:

Generate Vulkan Object IDs for Use in Caching

- Different types of Vulkan objects **may share same** handle value:



- Handle values **may be reused** after Vulkan object's destruction:



- Direct handle usage for caching **is error prone** and requires careful "dead" object tracking.

Tip:

Generate Vulkan Object IDs for Use in Caching

- Assign **unique ID** for each created object;

VulkanBuffer
VkBuffer Handle = 42
uint32_t HandleID = 144

– Assigned using counter. Used for comparison.

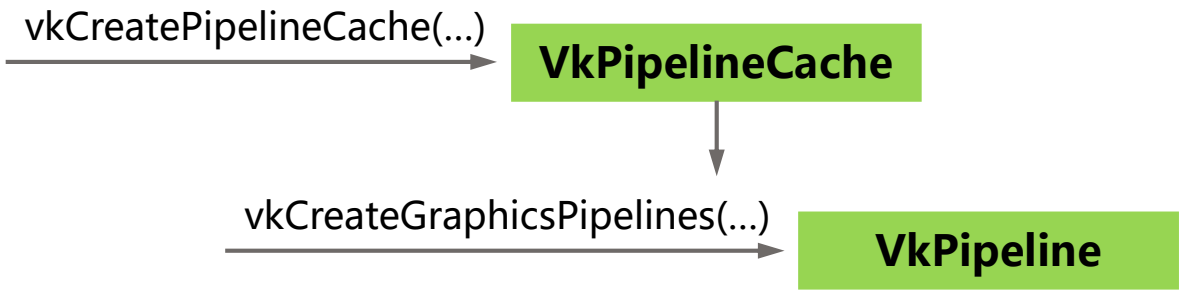
- With IDs, “dead” entries **cause no harm** and may be removed using more efficient GC logic.

- **Avoid** using **global map** to convert Vulkan object handles to IDs;

CPU Optimization:

Use Vulkan Pipeline Cache

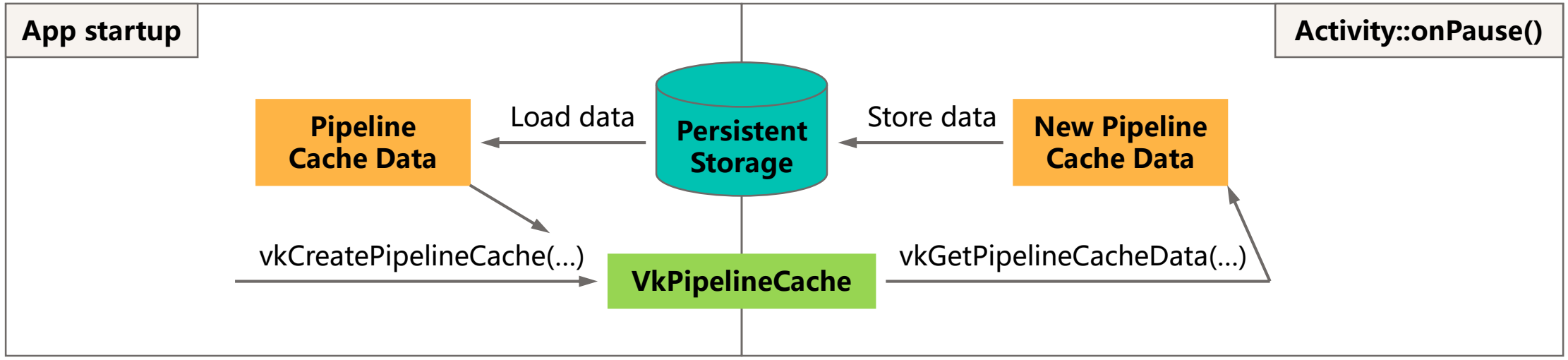
- VkPipeline object controls states for all **shader** and **fixed-function** stages;
- Call to vkCreateGraphicsPipelines(…) is very expensive;
- Use VkPipelineCache object to reduce cost of this call:



CPU Optimization:

Use Vulkan Pipeline Cache

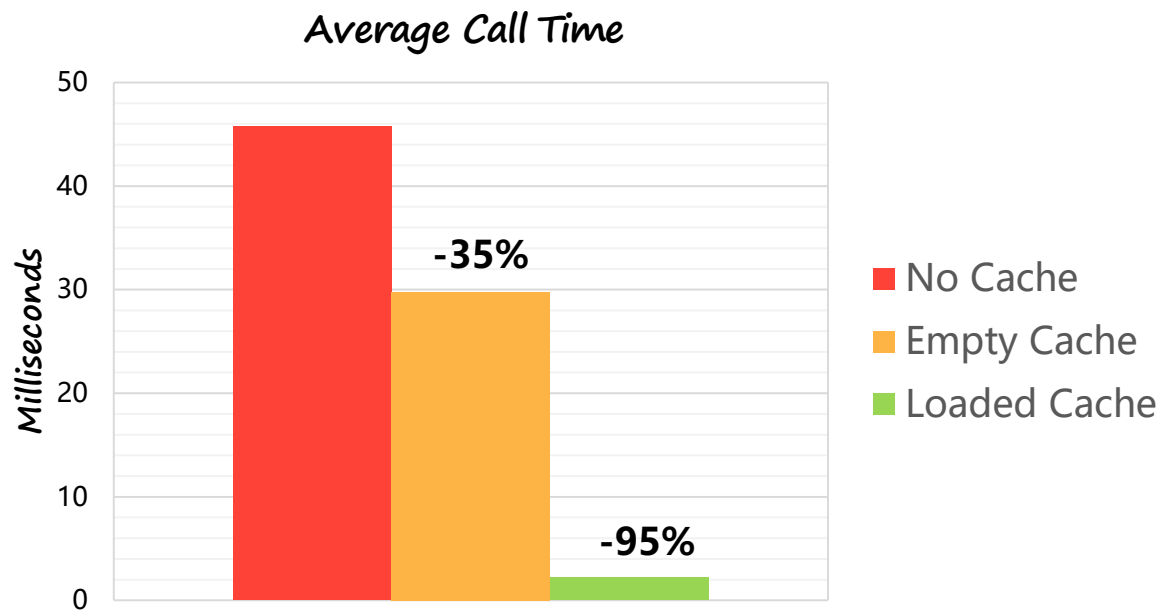
- Simple usage of the VkPipelineCache object is not enough;
- Create VkPipelineCache object from the **Pipeline Cache Data**:



Case Study:

Use Vulkan Pipeline Cache

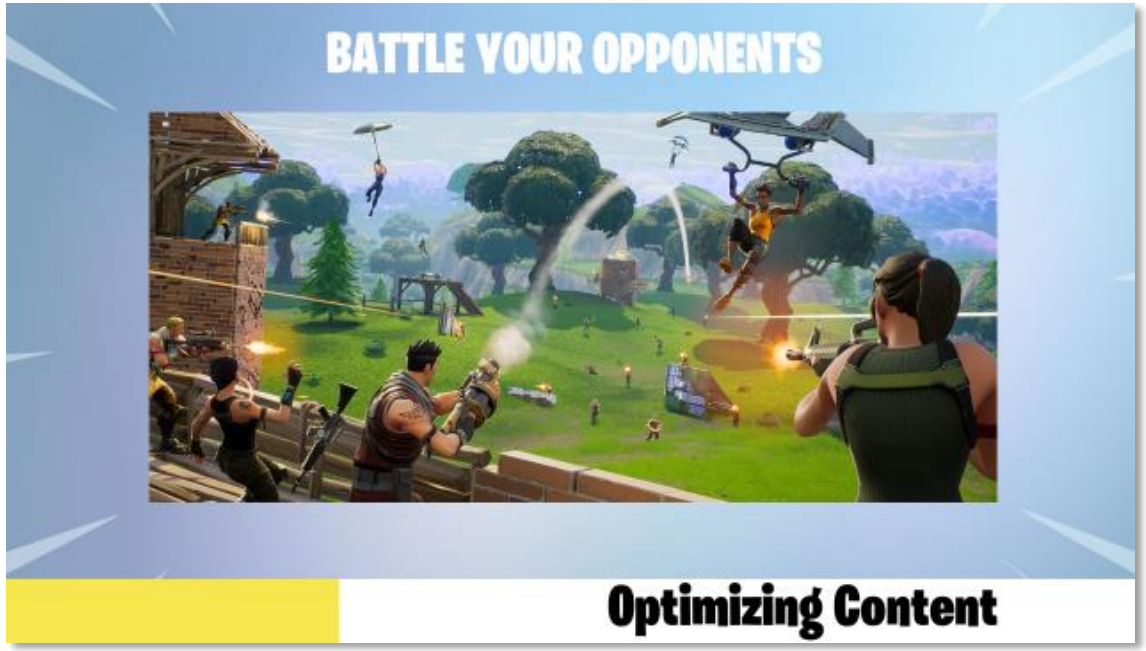
- Performance comparison of vkCreateGraphicsPipelines(...) call, creating approximately **4300** Pipelines in **Fortnite Mobile** game during loading:



Case Study:

Do Not Use "Little Cores" for Pipeline Compilation

- **Fortnite Mobile** creates around **4300** unique Pipelines at loading;
- Initial Pipeline compilation is done in the "**Optimizing Content**" phase;
- **Before** the optimization it used **"Little Cores"**;
- **After** the optimization it uses **"All Cores"**.



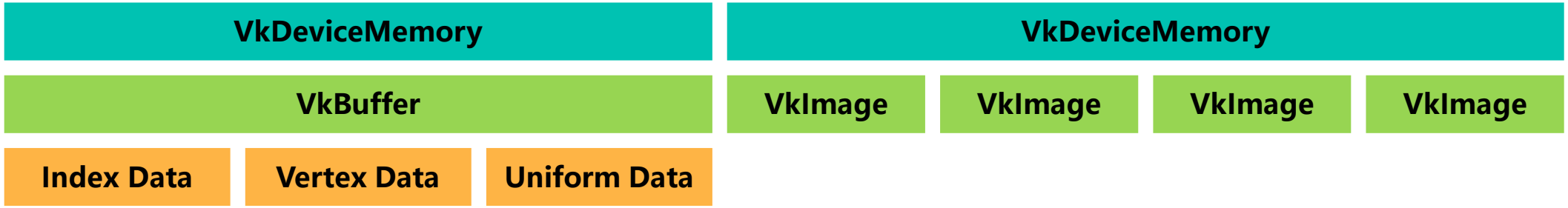
Samsung Galaxy S8:

7 min 5 sec → 2 min 45 sec

CPU & GPU Optimization:

Allocate Vulkan Memory in Large Chunks

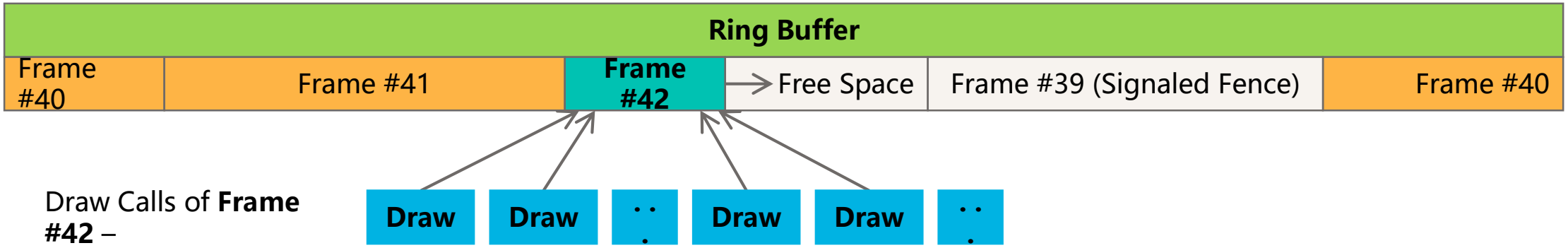
- Vulkan API has maxMemoryAllocationCount limit;
- According to vulkan.gpuinfo.org almost all mobile GPUs has limit of **4096** allocations;
- Prefer single VkBuffer object per each VkDeviceMemory allocation;
- Share large VkDeviceMemory allocation between multiple VkImage objects.



CPU & GPU Optimization:

Use Ring Buffer for Uniform Data That Changes Every Frame

- Dedicate separate large buffer for Uniform Data that changes every frame:

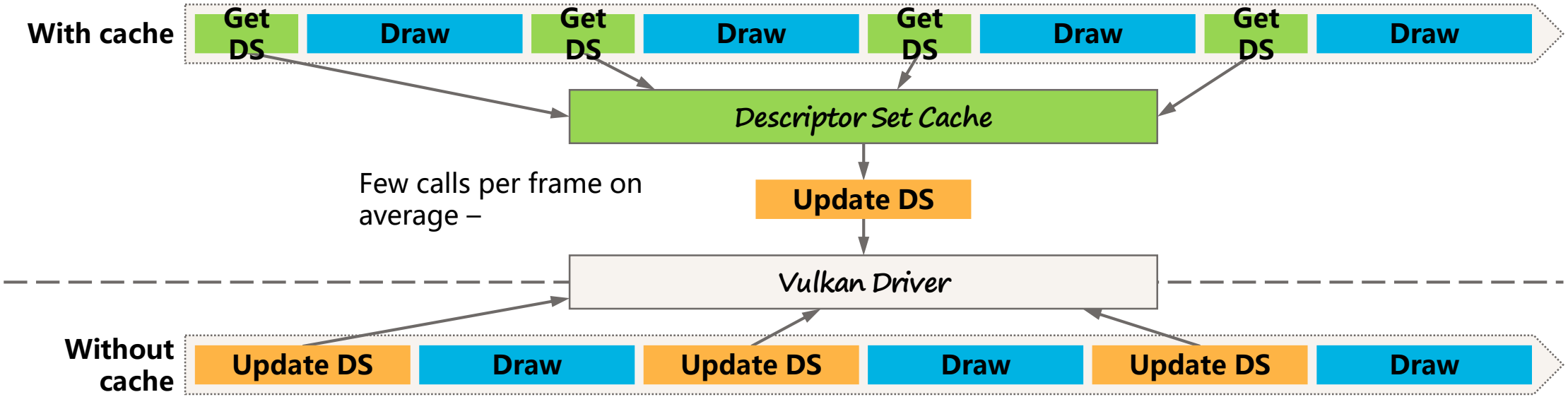


- Assign VkFence object for each frame to track when GPU is done using the Ring Buffer's memory.

CPU Optimization:

Implement Descriptor Set Cache

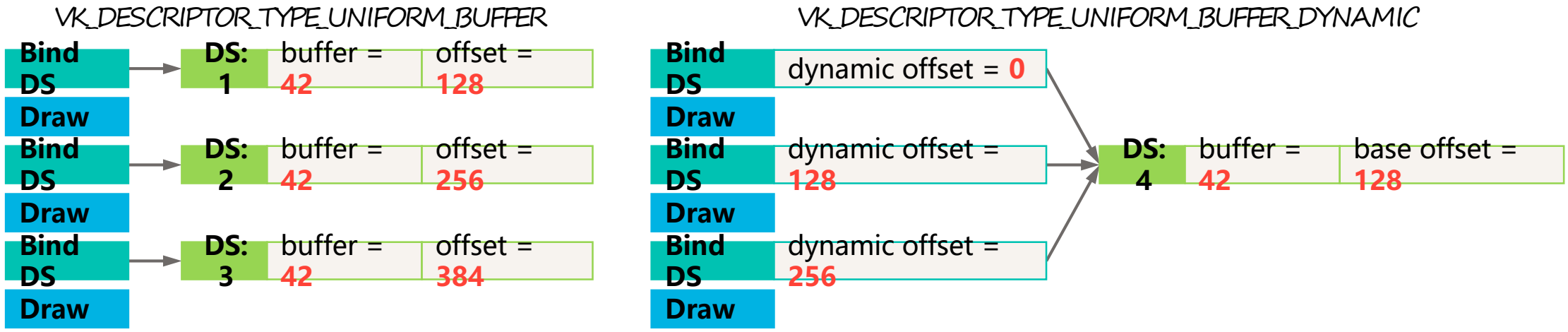
- Calling vkUpdateDescriptorSets(...) for each draw call may be very **expensive**;
- It is better to **reuse** already **updated** Descriptor Sets by implementing efficient cache:



CPU & Memory Optimization:

Use Dynamic Uniform Buffers

- Vulkan API supports two types of Descriptors for Uniform Buffers:
  - VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
  - VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_**DYNAMIC**.
- Dynamic Uniform Buffer allows setting **offset** without Descriptor Set update.



Galaxy GameDev

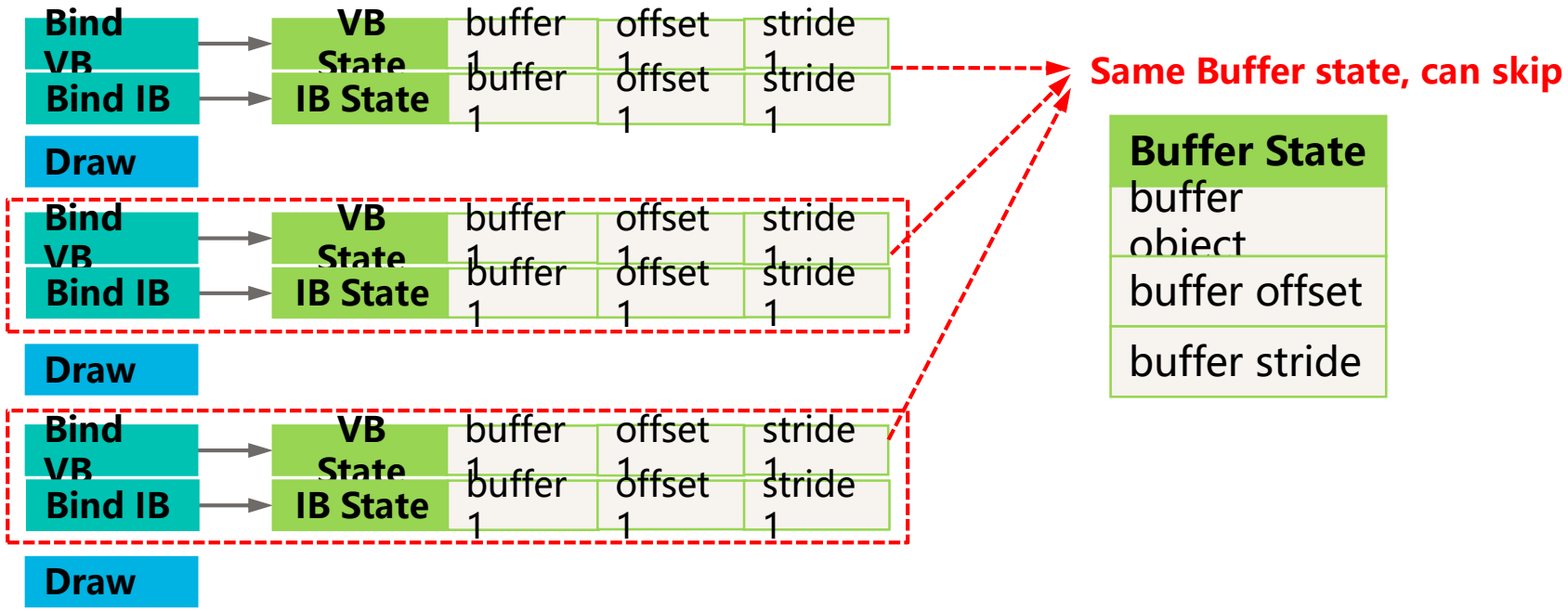
Wei Yao

GameDev Engineer, Samsung Research China

CPU Optimization:

Reduce Unnecessary VB & IB Bind Commands

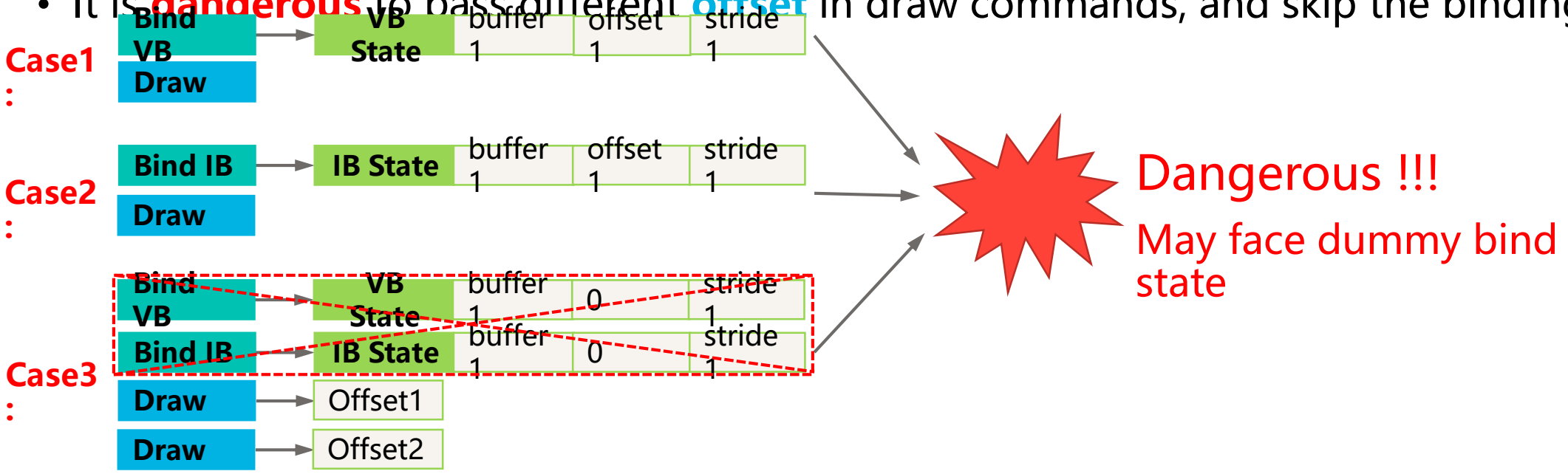
- Cached the states of **Vertex buffer** and **Index buffer**;



Tip:

Reduce Unnecessary VB & IB Bind Commands

- Better skip both **Vertex buffer** and **Index buffer** at the same time, or not skip **both** of them;
- It is **dangerous** to pass different **offset** in draw commands, and skip the binding;



GPU Optimization:

Use proper Pipeline Barriers

- **Pipeline barriers** have a **stage flag** that the application developer can set;
- If waiting too early (when not needing to) then some work will stay for no reason;

Example:

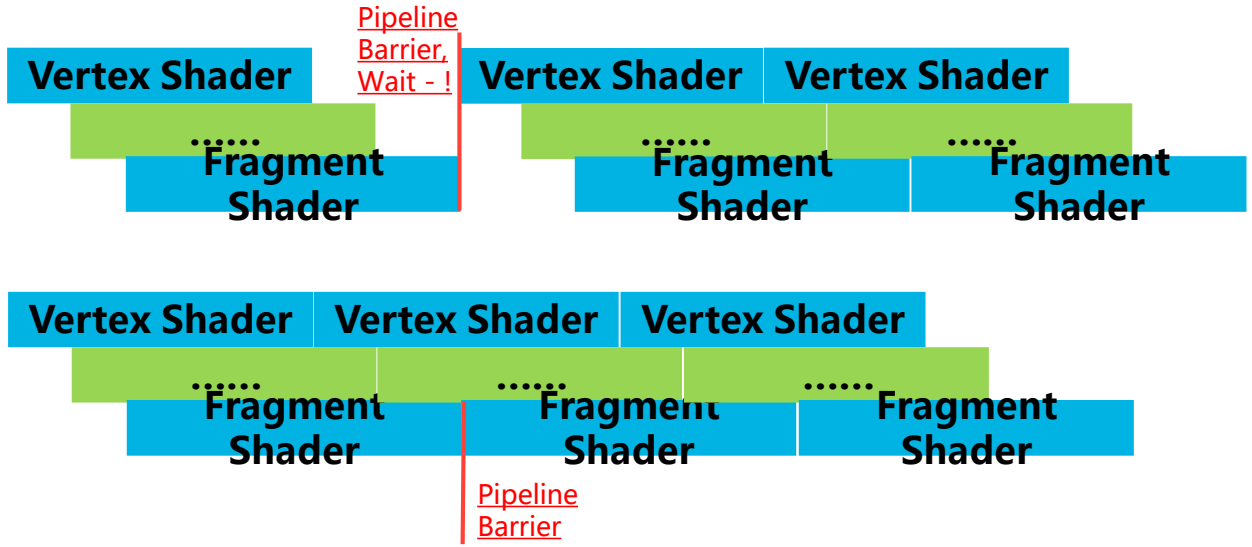
Change image layout to readable

SRC
VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT

DST
VK_PIPELINE_STAGE_VERTEX_SHADER_BIT

SRC
VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT

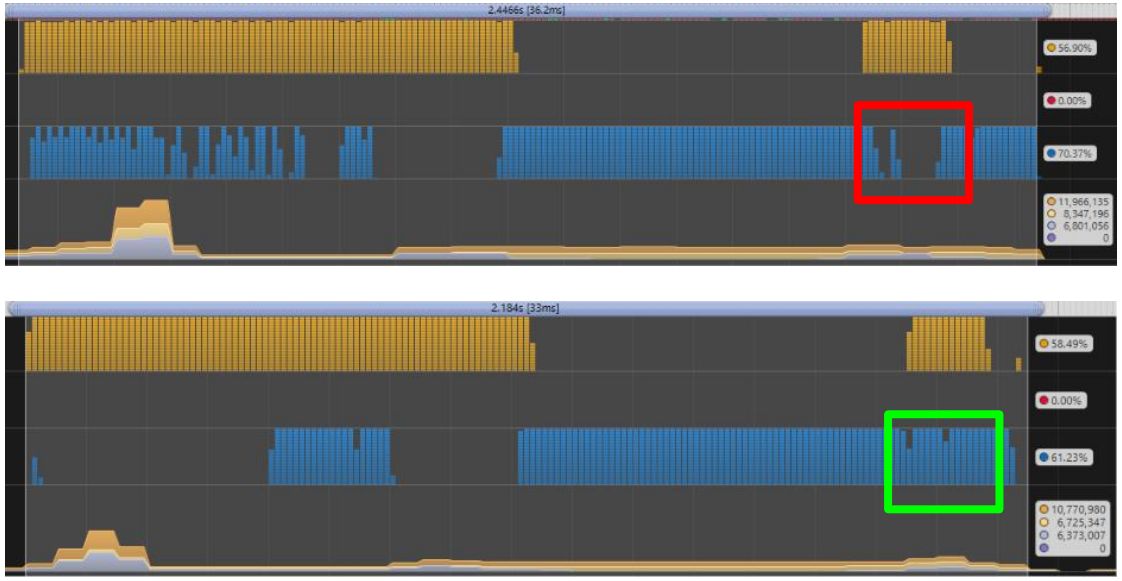
DST
VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT



GPU Optimization:

Use proper Pipeline Barriers

- Heavy **pipeline barrier(stage)** interrupt parallel processing of vertex/fragment job;
- Need to use light & proper pipeline **stage**;



GPU time : 36.2 ms

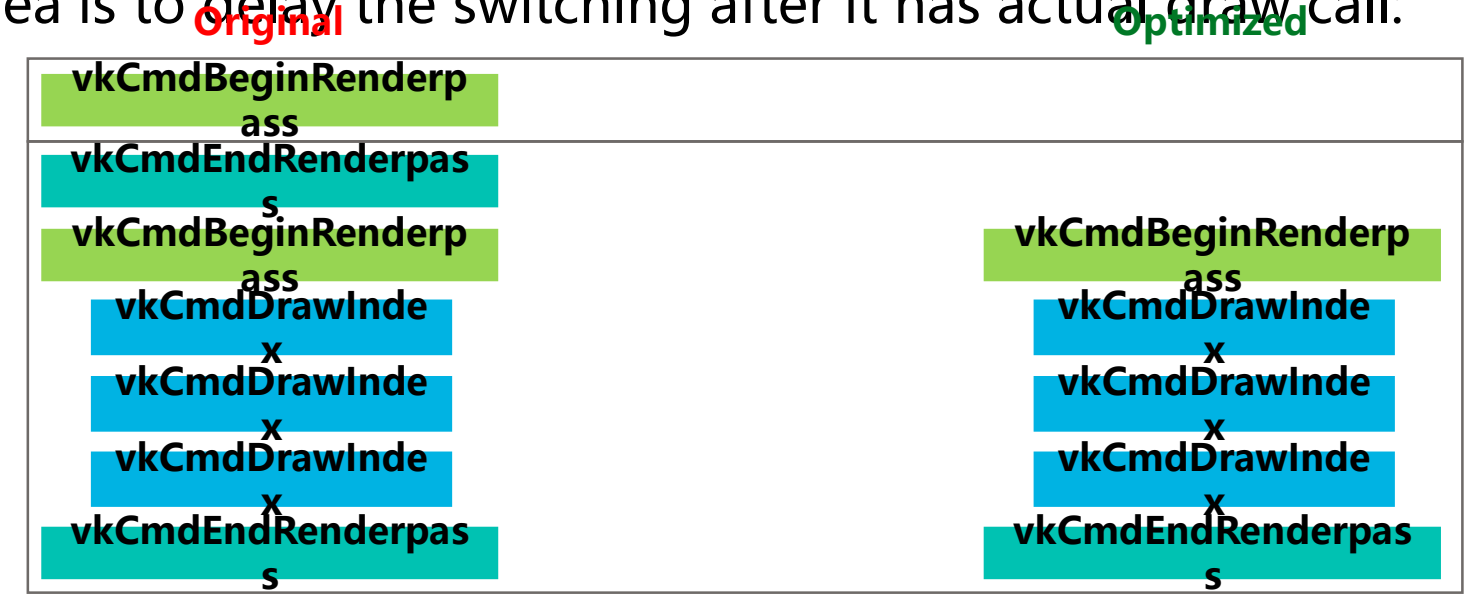
GPU time : 33 ms

GPU Optimization:

Eliminate Empty Renderpass Instances

- In some cases, engine will have some **renderpass** with no draw calls, just for clearing RT;
- **Renderpass switching** is quite heavy operations, should avoid empty **renderpass**;

One idea is to delay the switching after it has actual draw call:

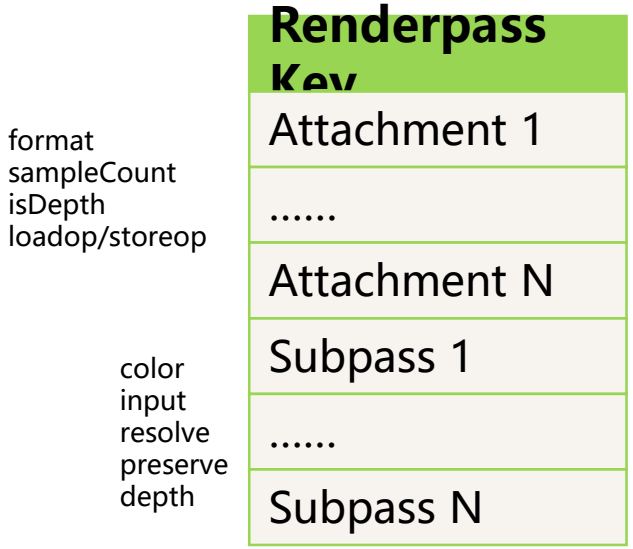


GPU Optimization:

Minimize number of Renderpass Instances

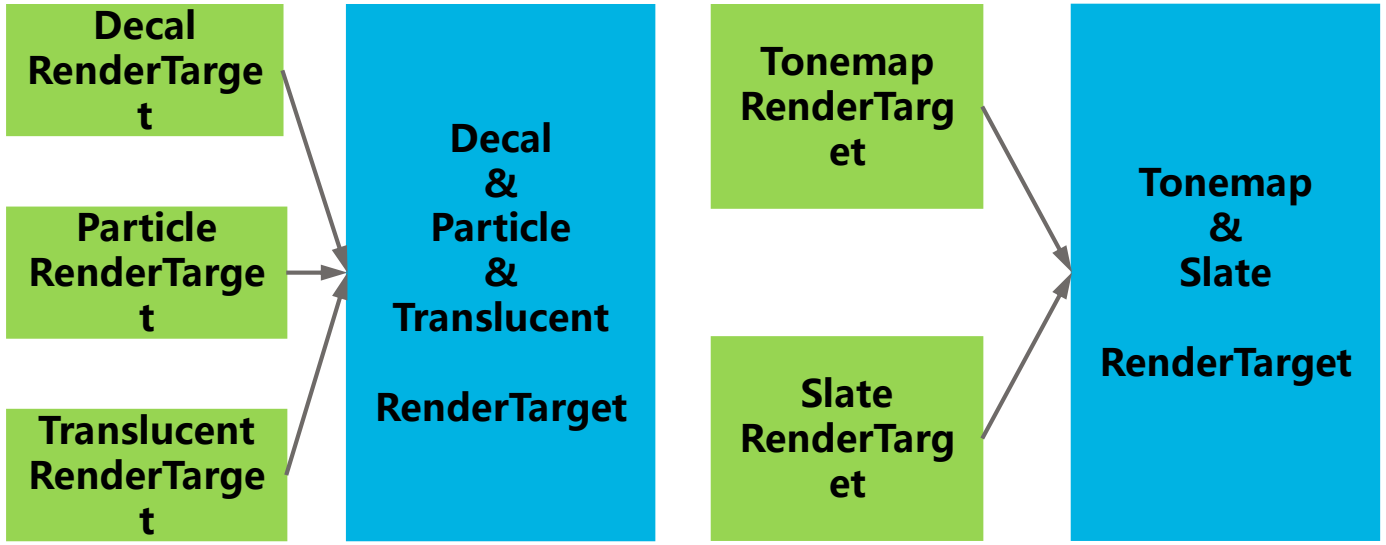
- Use **Attachment** information and **subpass** information as Key; (avoid creating all the time)
- Merging the render target to avoid unnecessary renderpass logic;

Use key to minimize the number of Renderpass



Avoid unnecessary renderpass logic

Here is an example from UE4:



GPU Optimization:

Optimize Render target Load/Store operations

- Vulkan API can **explicitly** specify the operations during start and end the subpass;
- **VkAttachmentDescription** describe the load and store operations of an attachment when used in a subpass;

Loadop
LOAD
CLEAR
DONT_CARE

allows the GPU do nothing before rendering into the attachment;

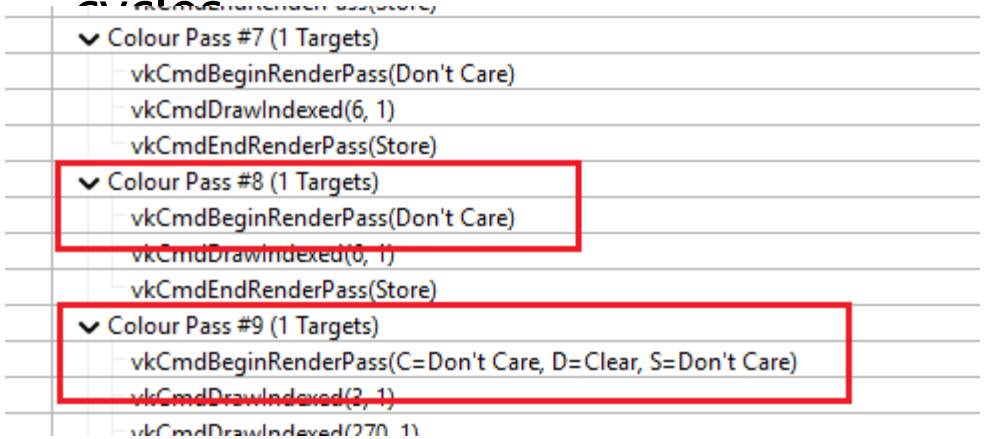
Storeop
Store
DONT_CARE

allows the GPU to discard the contents after executing the subpass;

Tip:

Optimize Render target Load/Store operations

- Use ‘DONT_CARE’ for **loadop** can effectively reduce the bandwidth and GPU job cycles



	Frag cycles	Frag tasks	CPU cycles	IRQ cycles	VTC cycles
Use DONT_CARE	932,435	2,586	1,308,855	167,311	645,170
USE LOAD	900,828	2,586	1,267,408	153,481	640,105

- But you need to make sure:
  - Current attachment don’t need previous rendering result;
  - Current frame is fullscreen rendering.

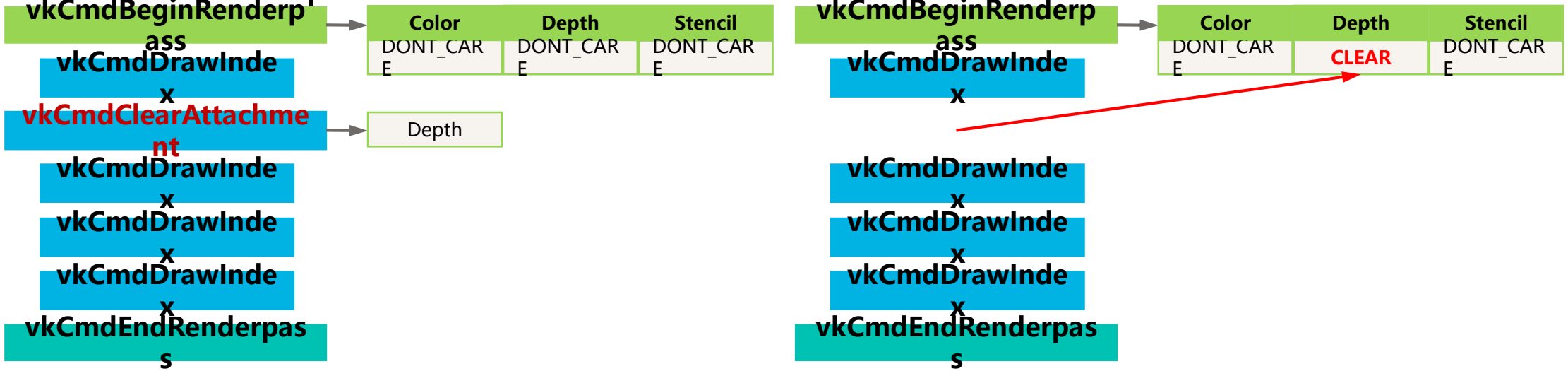
Or you may found dirty region on some drivers.

GPU Optimization:

Clear Render Target by Load Operation rather than Clear Command

- In most cases, we don't need to call clear command inside **renderpass**;
- Remove clear commands by using **loadop** can both reduce the CPU calling and GPU job

Here is an example:



GPU Optimization:

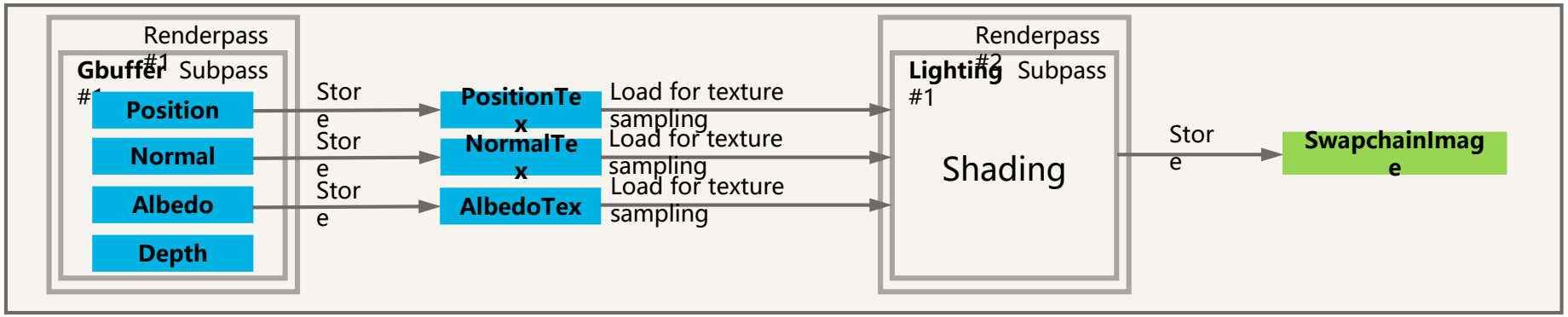
Use Subpasses when possible

- **Subpass** breaks the **renderpass** down into smaller intermediate steps;
- Each **renderpass** have at least one **subpass** and can have multiple **subpasses**;
- Each subpass identifies which attachments of the render pass it uses as inputs and outputs, and which attachment should be used as a depth/stencil buffer;
- By using **Subpass**, we can:
  - Read from attachments as InputAttachment;
  - Write to attachments;
  - Perform multi-sample resolve.

GPU Optimization:

Use Subpasses when possible

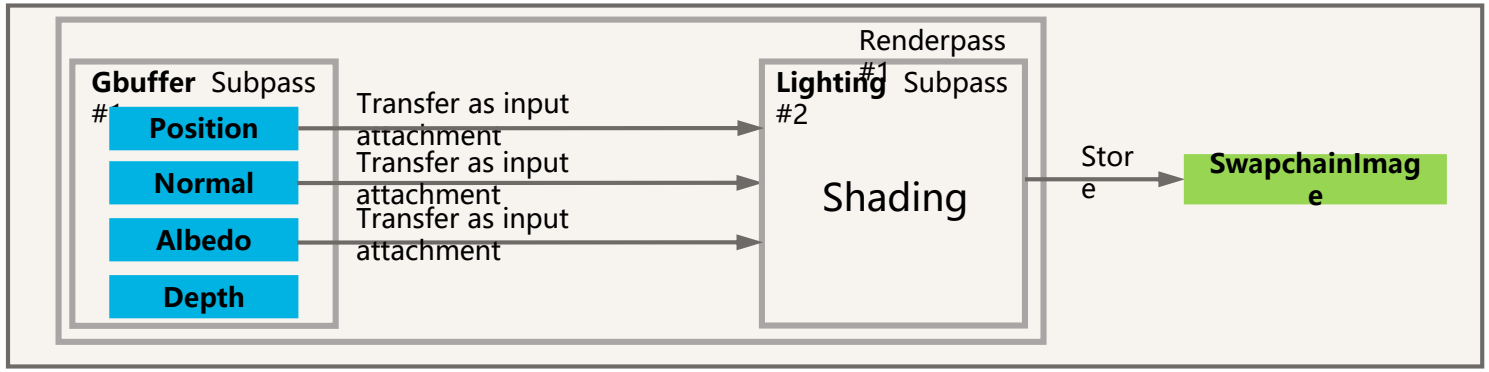
- Traditional process of deferred shading
  - Need two **renderpasses**;
  - First one will rendering the position, normal and albedo to corresponding attachments;
  - Second one will sample the textures and do the shading.



GPU Optimization:

Use Subpasses when possible

- Use multiple **subpasses**
  - Only 1 **renderpass**;
  - Doing Gbuffer and Lighting in 2 different **subpasses**;
  - Read as input attachment and pass from 1st **subpass** to 2nd.



GPU Optimization:

Use Subpasses when possible

- Use multiple **subpasses**
  - Can remove load/store operation;
  - Can reduce the memory bandwidth.

Single Subpass

Multi Subpasses (2 subpass)

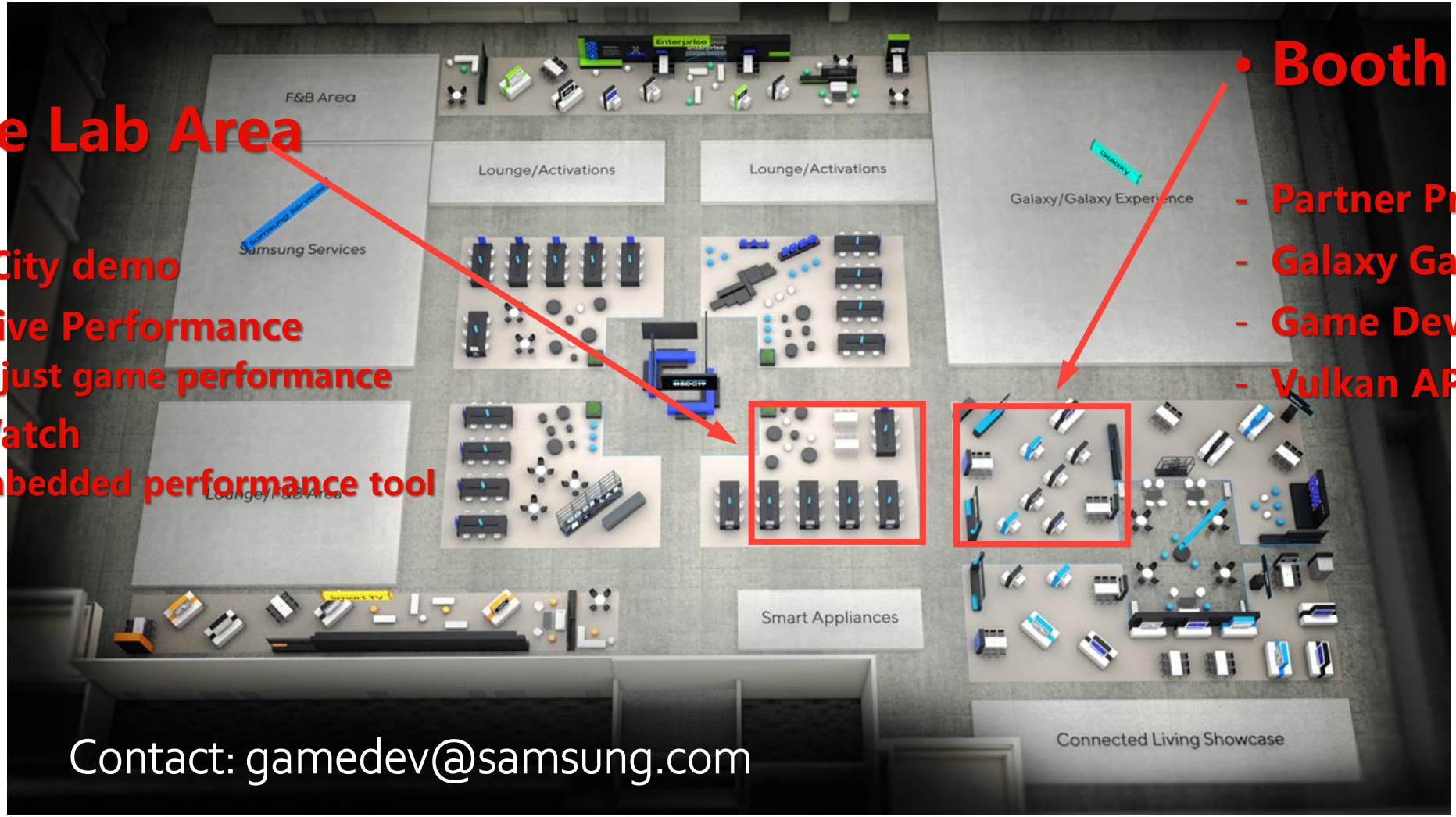
Vulkan

Release your CPU and Power your GPU

SAMSUNG Galaxy

Game Dev Code Lab & Booth

- **Code Lab Area**
  - MegaCity demo
  - Adaptive Performance
    - Adjust game performance
  - GPUWatch
    - Embedded performance tool



- **Booth**
  - Partner Practice
  - Galaxy Game Dev Activ
  - Game Dev Program
  - Vulkan API

Contact: gamedev@samsung.com

Game Sessions in SDC19

Game booster

- **13:30-1415 30th at ROOM 2 10B**
- Game Performance Platform
- Game booster introduction
- AI/ML based game optimization
- Big data assisted solution
- Platform connection interface for developer/studio/publish
- Plug in for additional tweaking
- Introduce best example for collaborating Samsung

Galaxy GameDev

#1: GameDev + Partner Practices

#2: Unity Adaptive Performance

#3: Vulkan Technical Studies

- Join Galaxy GameDev now to optimize your game itself !!!

unity Adaptive Performance

UNREAL ENGINE

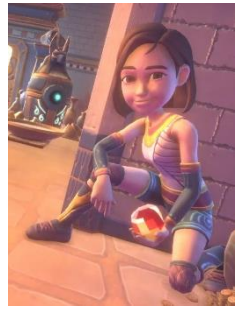
Vulkan

GPUWatch Galaxy GameSDK

AR Emoji SDK

- **14:30-1450 29th at ROOM 2 10A**
- What if You Are in the Game?
  : AR Emoji SDK
- Create your own avatar in yo

AR Emoji unity

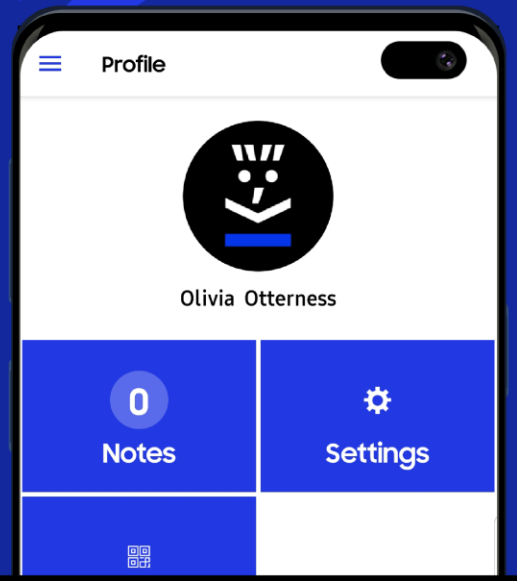


Game Performance Index

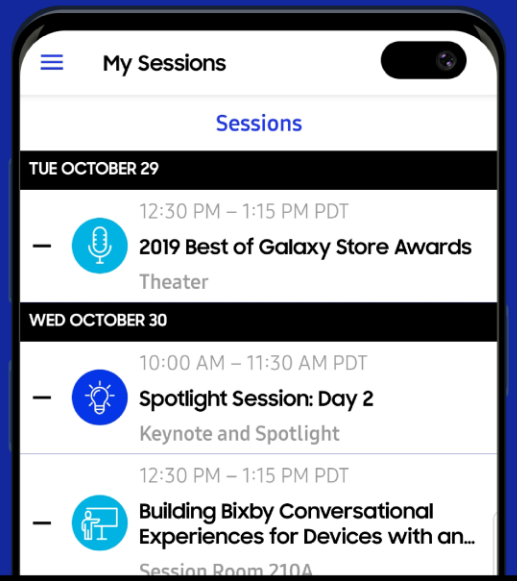
- **1500-1520 30th at ROOM 211A**
- **A new standard to represent mobile game performance**
- Mobile Gaming is more than FPS!
- State of Mobile Gaming
- What is Mobile Gaming Performance?
- Galaxy Gaming Bigdata driven approach

Lucky Draw !!!

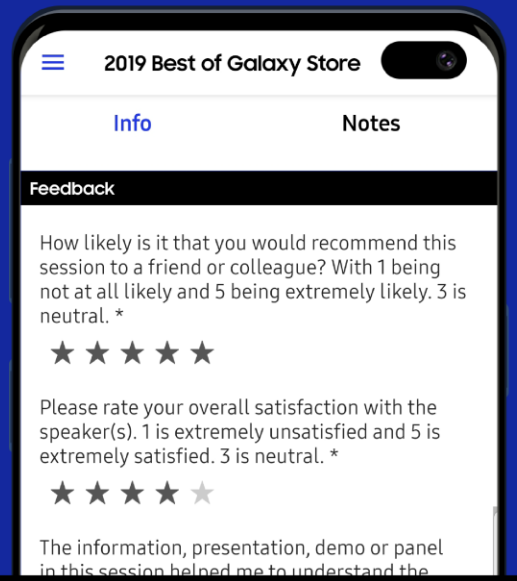
Rate Your Tech Sessions



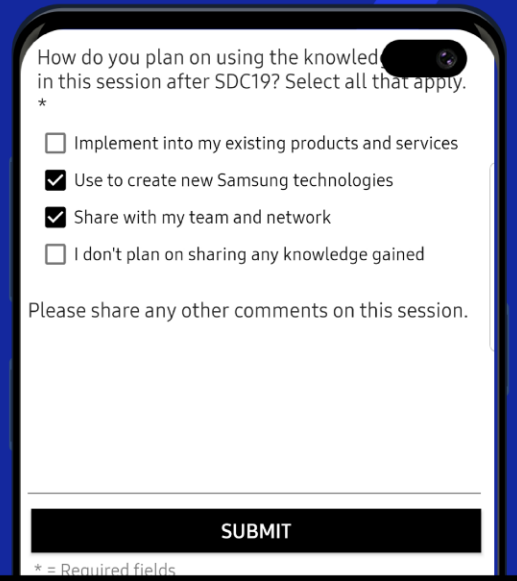
Log in to the SDC19 app using your registration email and confirmation number.



To select, go to "My Surveys" (if you've favorited a session) or search the name under "Sessions."



Provide your feedback at the bottom of the session description.



Hit "Submit" to finish the survey.