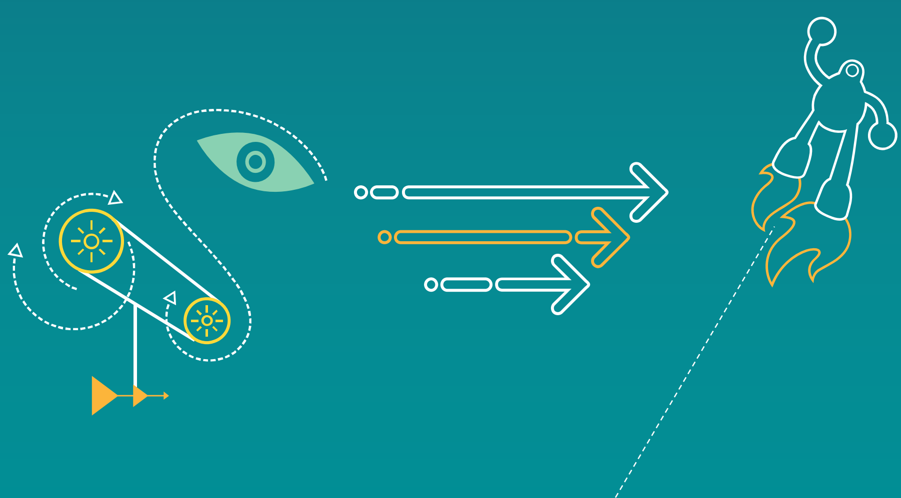
Graphics Power and Performance Overview

Qualcomm Technologies, Inc.

80-NP885-1 Rev. B



Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Revision History

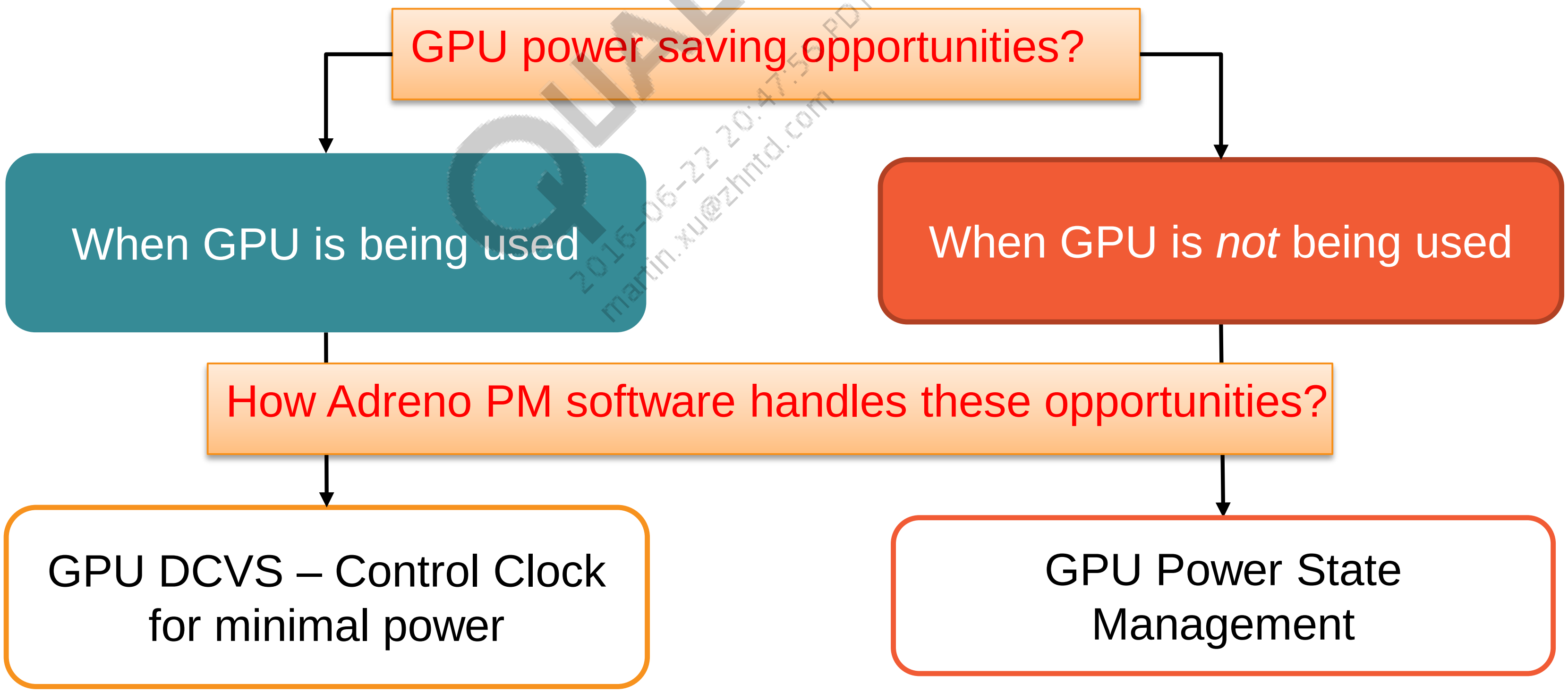
Revision	Date	Description
A	Jul 2014	Initial release
B	June 2015	Added cases study for GPU power profiling. Updated GPU Power States Diagram. Slides restructures and reorganizations.

Agenda

Adreno Software Power Management Overview

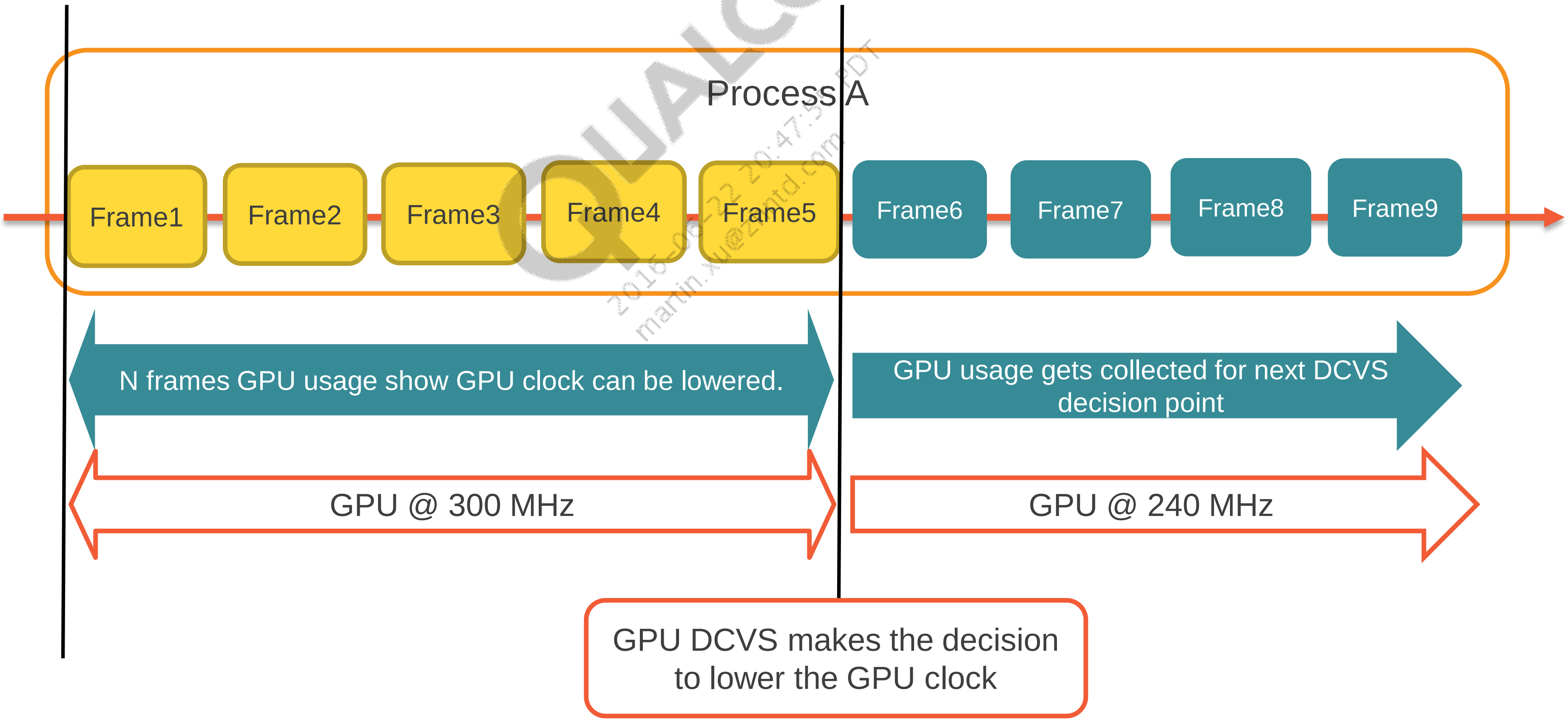
Adreno Power Management Software Overview (1 of 7)

- Adreno™ GPU Power Management software is based on a specific system's GPU usage, taking every opportunity to control power saving with optimal performance.

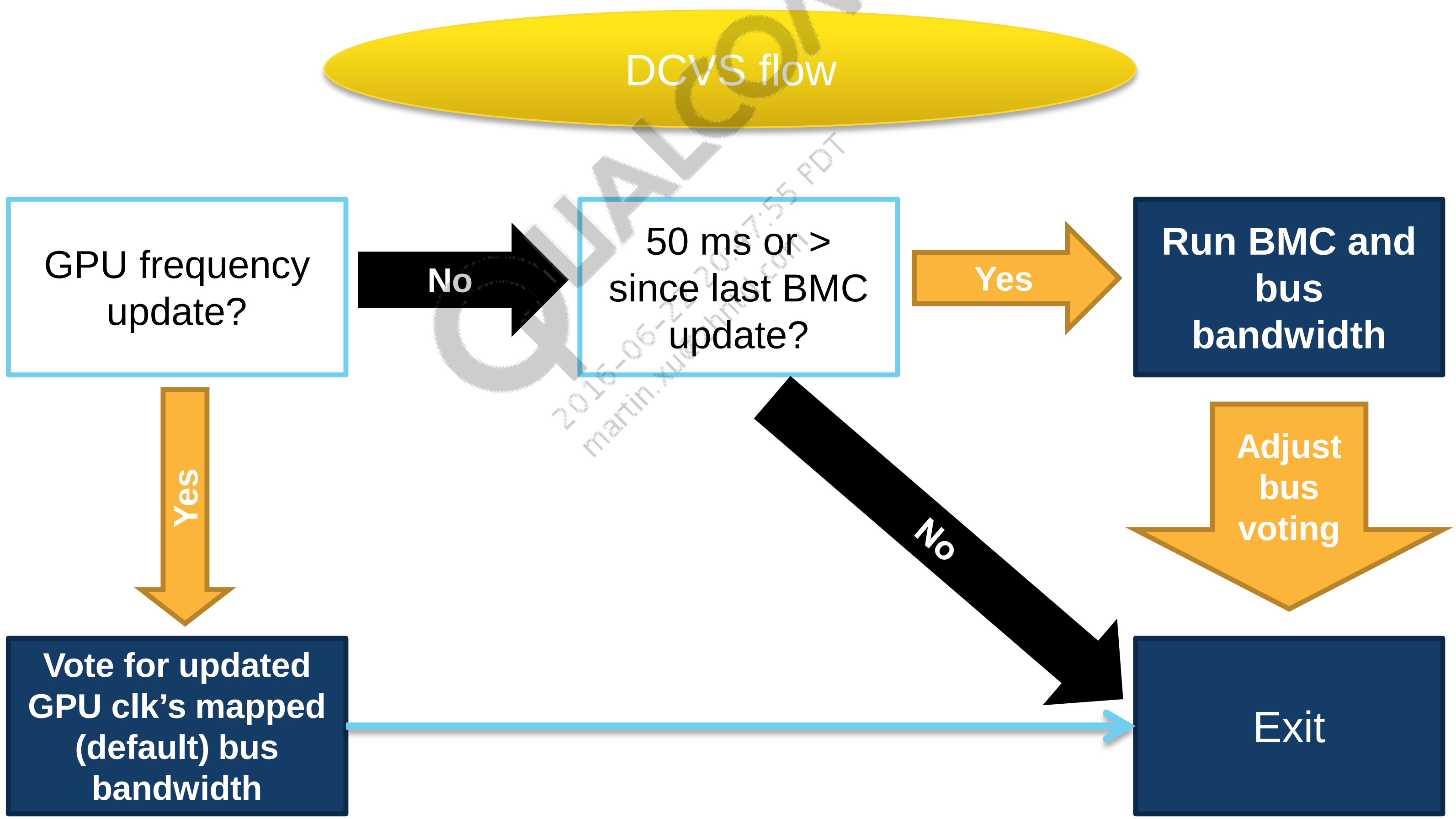


Adreno Power Management Software Overview (2 of 7)

- The BIG picture – When the GPU is being used, GPU DCVS controls the power.

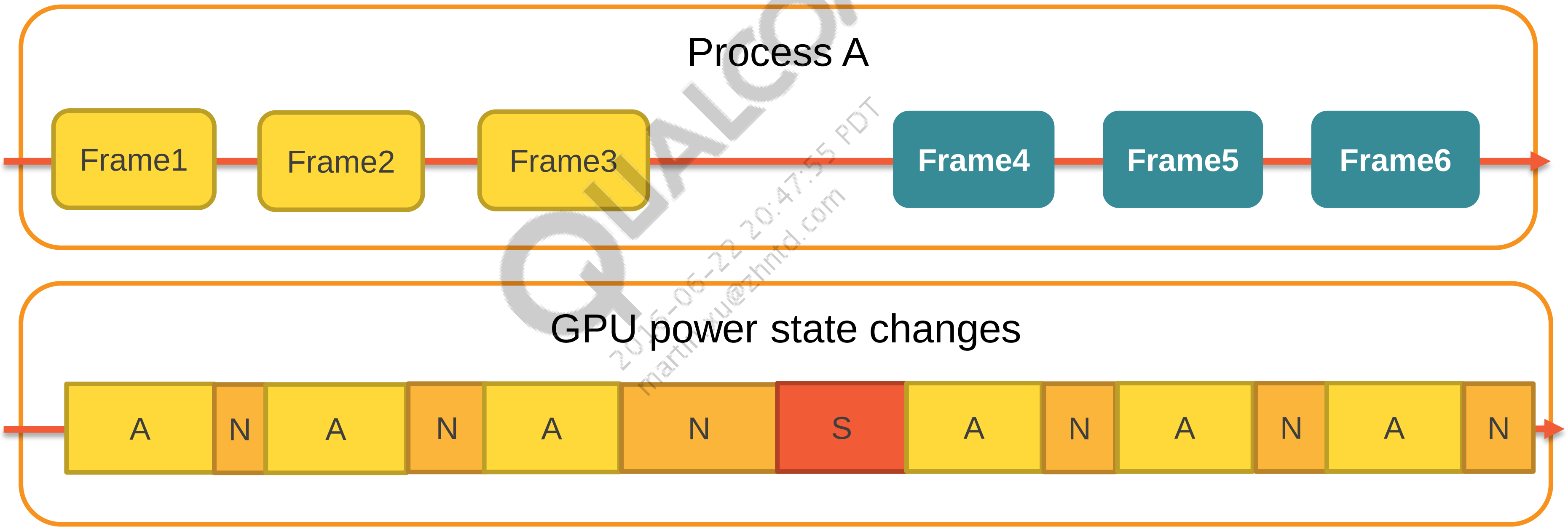


Adreno Power Management Software Overview (3 of 7)



Adreno Power Management Software Overview (4 of 7)

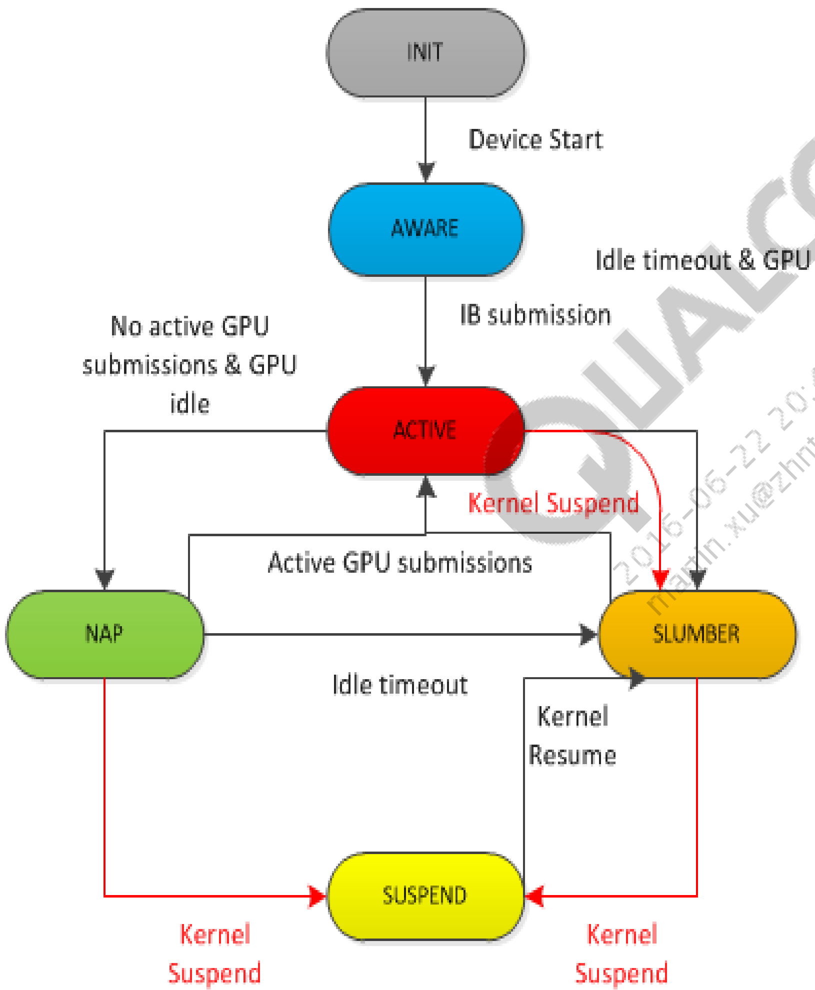
- When the GPU is not being used, the GPU power state management kicks in.



Legend

- Active state
- Nap state
- Sleep/Slumber state

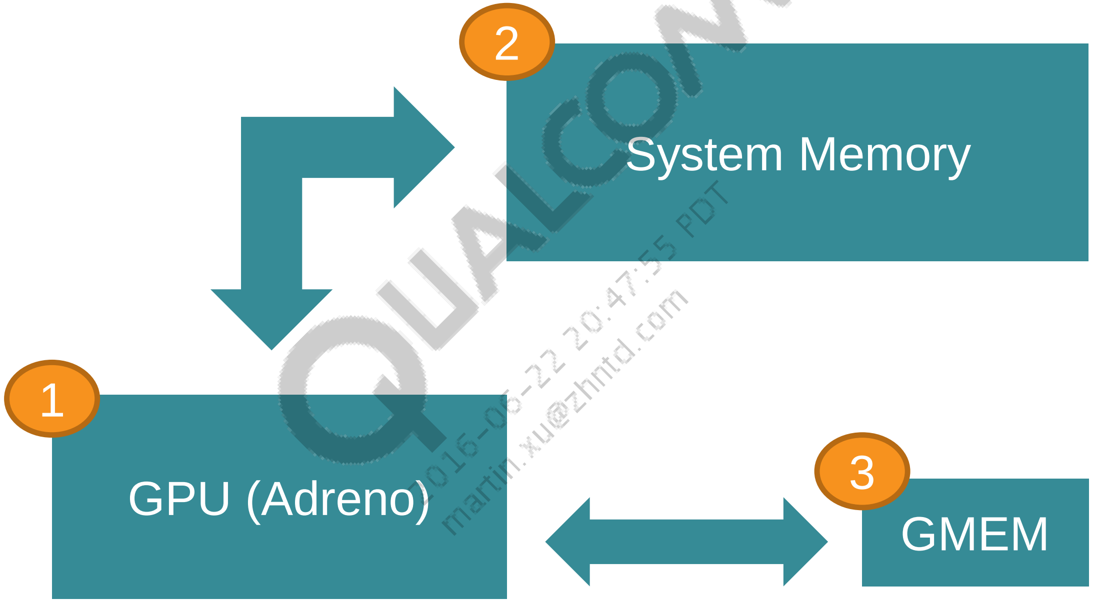
Adreno Power Management Software Overview (5 of 7)



State	Description
INIT	Device init is successful after device probe
NAP	CLOCKS are turned OFF (s/w clock gating)
SLUMBER	CLOCKS/IRQ/BUS VOTES/GPU RAIL are turned OFF. GPU submissions accepted
SUSPEND	Power resources are OFF. No GPU submissions accepted
ACTIVE	All power resources ON. Timers and GPU IRQ are ON.
AWARE	All power resources ON. Timers and GPU IRQ are OFF

Adreno Power Management Software Overview (6 of 7)

- Three physical pieces of hardware work together in a typical graphics rendering scenario.*



- Adreno GPU Power Management software controls these hardware blocks both directly and indirectly for optimal balance between power and performance.
- Adreno Linux Kernel Driver, KGSL, is at the core of the GPU power management software.

* There are other hardware blocks in the actual rendering process, such as CPU, but their power management is done outside the KGSL.

Adreno Power Management Software Overview (7 of 7)

Terms used in document	Description
GPU Power Levels	A set of the GPU's operational power levels that include the GPU clock and the default bus level for the clock; typically, Power Level 0 is mapped to maximum GPU clock
GPU DCVS	Dynamic Current/Voltage Scaling feature for the GPU clock
GPU Bus DCVS	Dynamic Current/Voltage Scaling feature for memory bandwidth (system bus/GMEM bus clock) voting
GPU Devfreq Governor	Implementation of GPU DCVS and GPU bus DCVS based on the Linux Devfreq framework
Device Bindings	A set of KGSL/Adreno device attributes statically defined in the .dtsi file
Initial GPU Power Level	The default power level used when the GPU is initialized or resumed after suspension
Default Bus Level	The default bus level defined inside each GPU power level to be set when the GPU power level changes
Static Bus Bandwidth Mapping	One-to-one mapping of the bus level to each GPU power level. With static bus bandwidth mapping, there is no GPU bus DCVS. Low-tier chipsets often use this mapping
Dynamic Bus Bandwidth Mapping	Many-to-one mapping of bus levels to each GPU power level. Mid- and high-tier chipsets support this mapping

Device Bindings

- For each target chip, the GPU device bindings can be found in: /kernel/arch/arm/boot/dts/qcom/
  For example:
  – /kernel/arch/arm/boot/dts/qcom/apq8084-gpu.dtsi for APQ8084
  – /kernel/arch/arm/boot/dts/qcom/msm8916-gpu.dtsi for MSM8916
- Understanding the entries in the .dtsi file can clarify how the GPU attributes of a specific MSM or APQ are set up in the GPU's device driver
- For details on the entries in the file, refer to the documentation located at: /kernel/Documentation/devicetree/bindings/gpu/

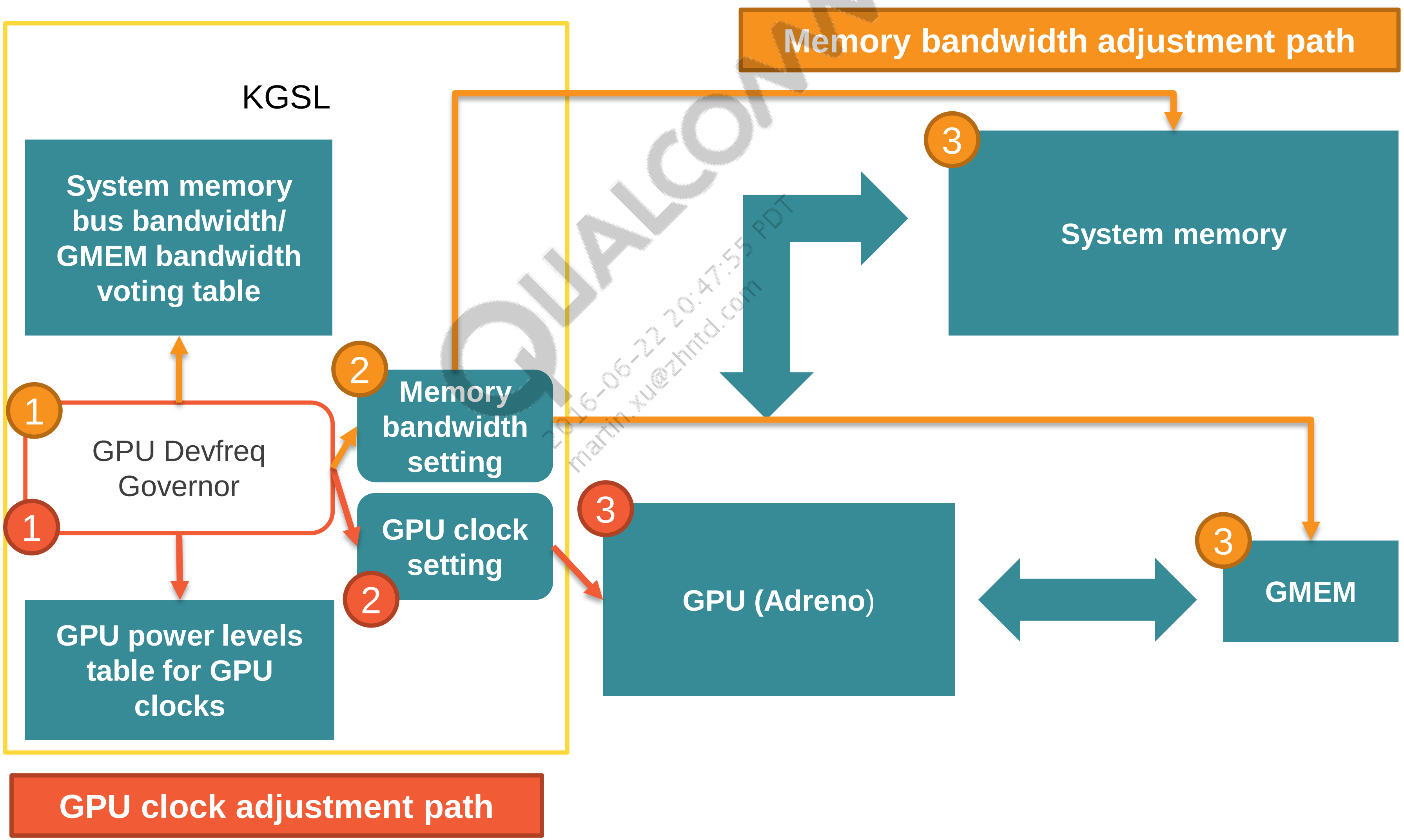
Linux Devfreq Framework

- Provides a generic framework to manage a non-CPU device's power usage based on a governor strategy used with the CPUFreq framework
  - For implementation details, go to http://lwn.net/Articles/445044/
- The QTI-proprietary Adreno powerscale-based governor has been adapted to the Devfreq framework
  - The governor's name is now changed to adreno_tz_governor.
  - /sys/class/kgsl/kgsl-3d0/devfreq/ contains relevant sysfs nodes for the GPU's devfreq governor.
  - There is no change in the DCVS algorithm backing the governor→it is still a QTI-proprietary algorithm based on GPU load statistics.

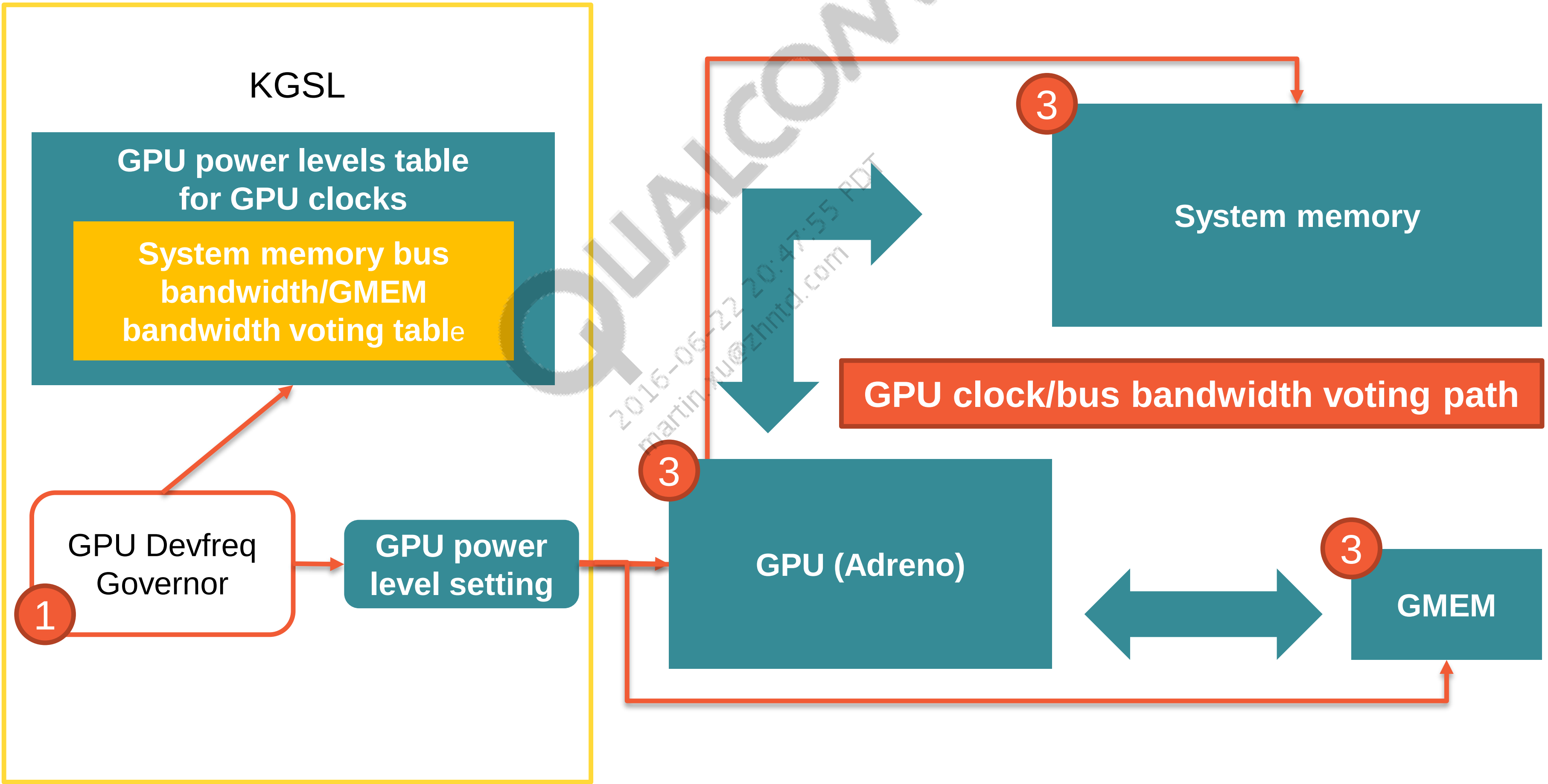
GPU DCVS vs. GPU Bus DCVS

- GPU DCVS has been used as an umbrella term for Adreno's GPU clock and bus bandwidth voting mechanism
- Depending on each chipset's clock-level configurations, GPU Bus DCVS may or may not be available
  - For low-tier chipsets, GPU DCVS takes care of GPU bus bandwidth voting statically mapped to each GPU clock.
  - Checking /sys/class/kgsl/kgsl-3d0/bus_split
    - 1 – GPU Bus DCVS is enabled→dynamic bus bandwidth mapping to GPU clocks
    - 0 – GPU Bus DCVS is *not* enabled→static bus bandwidth mapping to GPU clocks
- Adreno DCVS governor algorithm adaptation to the Devfreq framework, along with the opportunity to decouple GPU clocks setting from GPU bus bandwidth voting for better power management:
  - GPU DCVS is for GPU clock
  - GPU Bus DCVS is for bus bandwidth voting
  - System Bus Voting (RAM) and GMEM Bandwidth Voting are still tightly coupled.

KGSL GPU Governor Power Management with GPU DCVS and GPU Bus DCVS

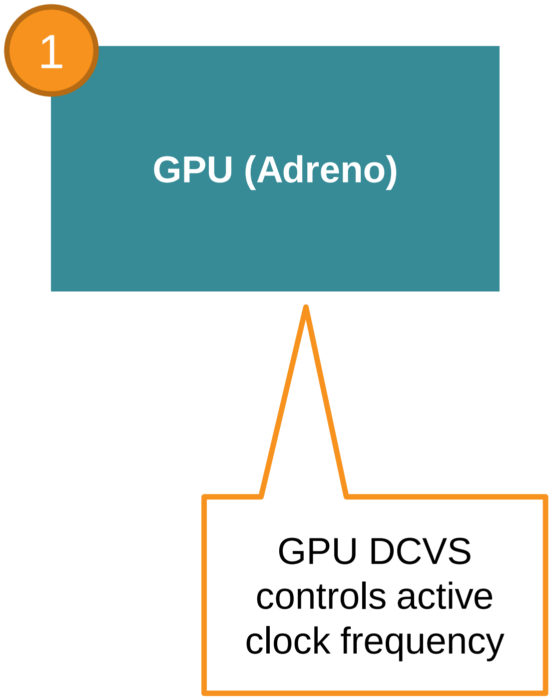


KGSL GPU Governor Power Management with GPU DCVS Only



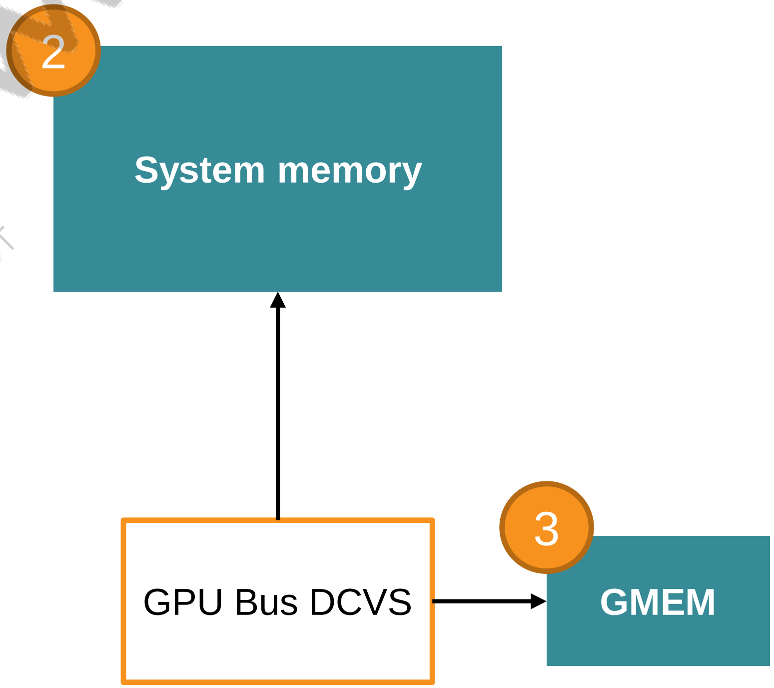
GPU DCVS

- The GPU is at the core of graphics processing. Its power-related attributes/settings are configured statically by the kernel device binding and managed dynamically by the GPU DCVS governor
- Static attributes
  - Initial power level
  - Idle timeout
  - Maximum power level
  - Minimum power level
  - Thermal power level
- Dynamic attribute
  - Active (running) power level
- Power levels – A group of predefined GPU clock levels that have different power source levels, such as SVS, Nonimal, and Turbo
- Idle Timeout – A set timer limit for the GPU not to power collapse

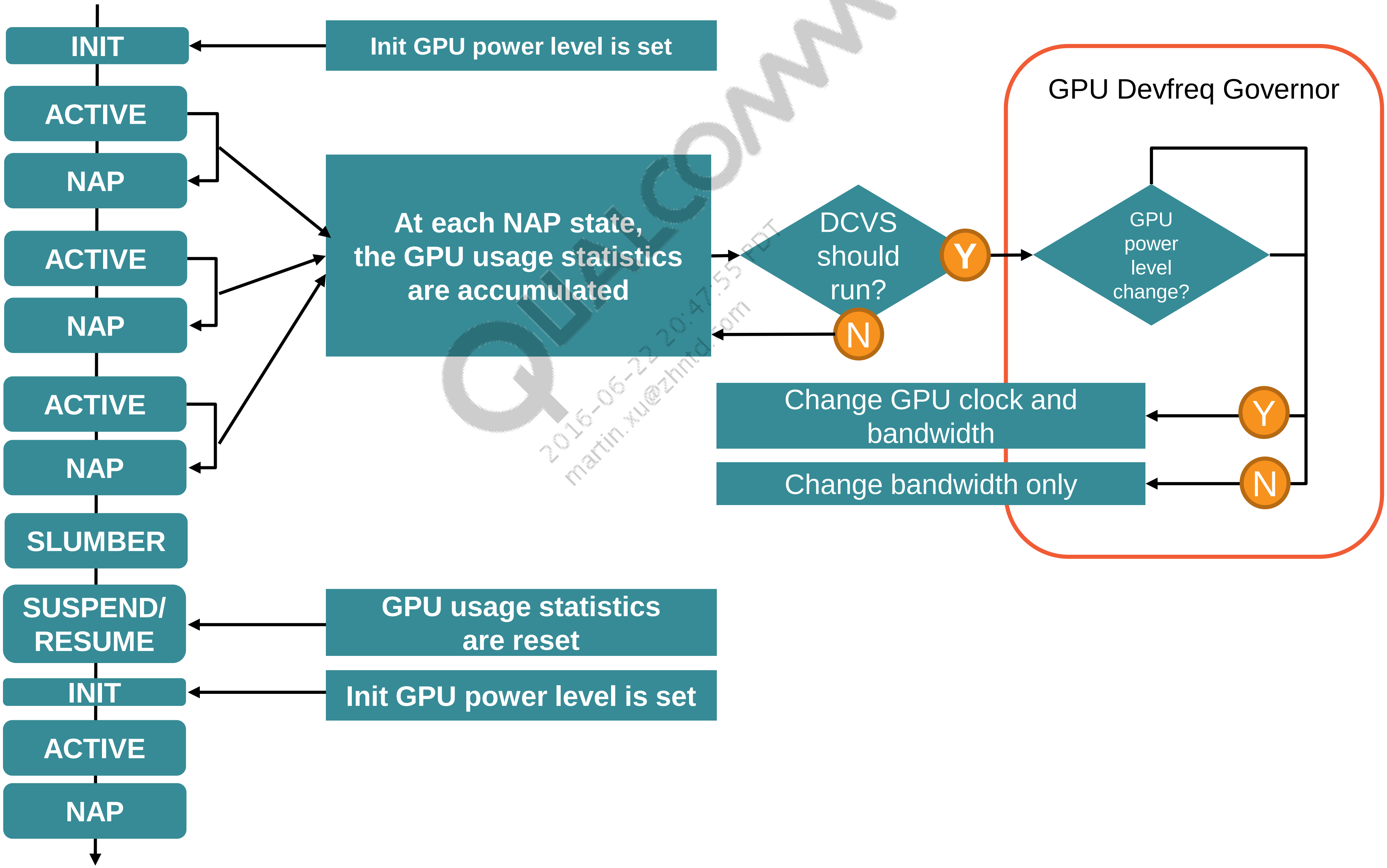


GPU Bus DCVS

- The GPU requires system memory access for its external resources, such as textures. Based on the bandwidth usage history, GPU Bus DCVS* votes for system memory bandwidth dynamically
- The GPU utilizes a part of GMEM for fast GPU memory operations
- GPU Bus DCVS is not the same as GPU DCVS – Even though there is one governor controlling both the GPU clock level and the GPU bus bandwidth level, running the GPU clock and bus bandwidth vote are decoupled within the same power level



Typical GPU Power State Transition with GPU DCVS Governor



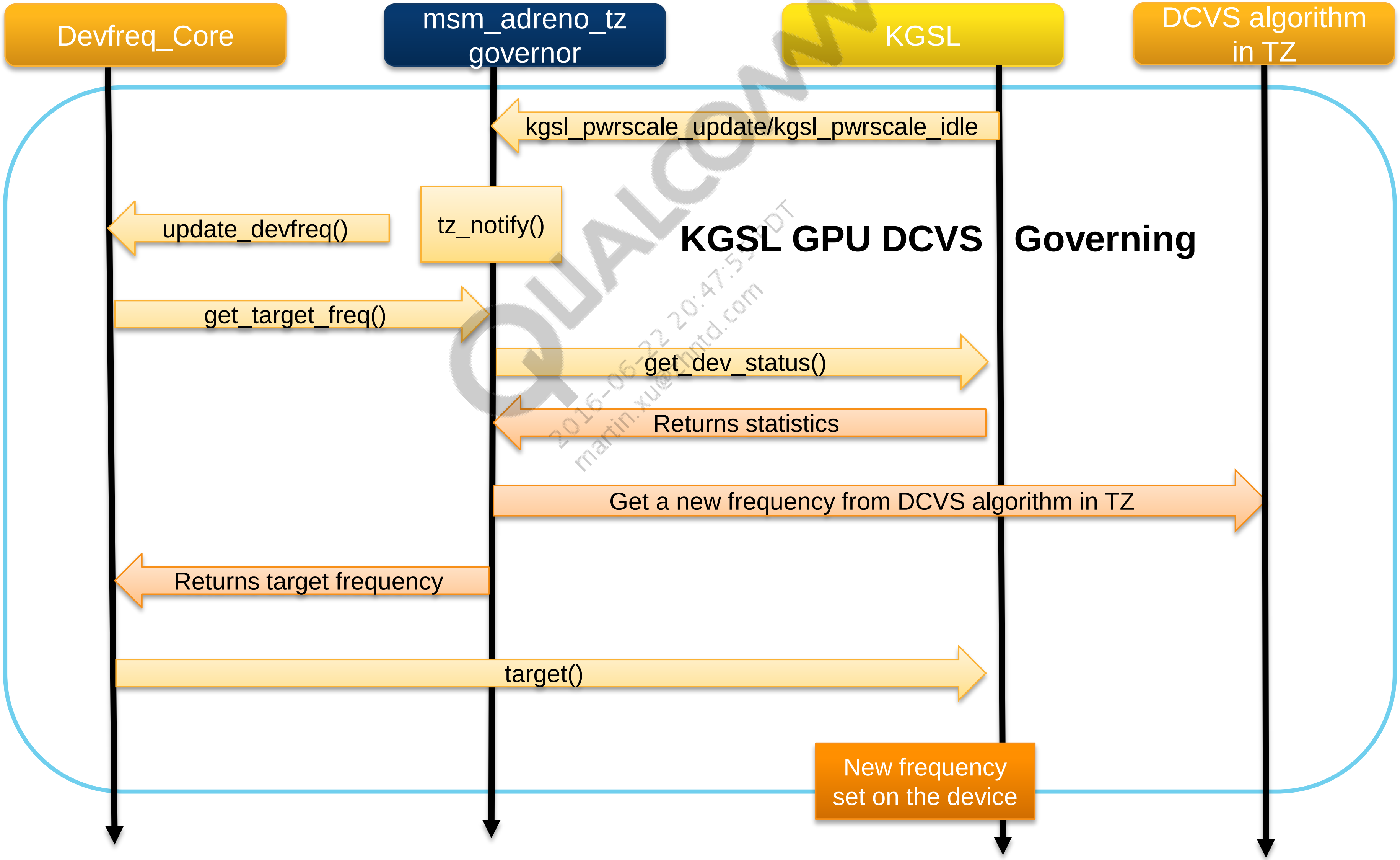
msm-governor-tz – QTI Default Governor for GPU DCVS, GPU Load-Based

- The KGSL driver communicates with the governor directly at moments that are important for the driver.*
  1. KGSL driver uses Linux notifier mechanism to call the governor's tz_notify() function.
  2. tz_notify () just calls update_devfreq().
  3. update_devfreq() calls the governor's get_target_freq() function.
  4. get_target_freq() gets the device status information (total and busy time) and calls the DCVS algorithm executed in TrustZone (TZ) for a new value for the device frequency.
  5. The reevaluated frequency is passed to the driver with its target() function.

* Important events from KGSL perspective

- When the GPU makes the transition from the ACTIVE state to the NAP state – consider how long the GPU was active at a specific power level
- When the GPU makes a power-level adjustment – consider the power-level changes

msm-governor-tz – QTI Default Governor for GPU DCVS



Relevant Code Locations

- KGSL – /kernel/drivers/gpu/msm
- Power code in multiple files under the KGSL directory
  - kgsl.c – Generic kernel driver hooks
  - kgsl_pwrctrl.c/h – Power-specific, shared device code
  - kgsl_pwrscale.c/h – Power framework for switching DCVS control to devfreq
- Devfreq framework for device frequency switching
  - /kernel/drivers/devfreq/governor_msm_adreno_tz.c – KGSL-specific governor with TZ algorithm
  - /kernel/include/linux/msm_adreno_devfreq.h – KGSL-specific governor .h file, used to sync devfreq and kgsl
- Platform-specific device tree files that are translated to fill out the kgsl_pdata struct
  - /kernel/arch/arm/boot/dts/qcom
    - msm8974-gpu.dtsi
    - msm8974-v2.dtsi
    - msm8974-v2.2.dtsi
    - msm8974pro.dtsi
    - msm8226-gpu.dtsi
    - msm8610-gpu.dtsi
    - apq8084-gpu.dtsi

Devfreq GPU DCVS Governor (1 of 4)

- Devfreq is an infrastructure introduced in Linux 3.2 to support Dynamic Voltage and Frequency Scaling (DVFS) for non-CPU devices
- There are three main components in devfreq:
  - Devfreq core
    - The main engine orchestrating frequency scaling for devices.
    - Source code can be found at: kernel/drivers/devfreq/
  - Governors
    - The predefined set includes simple_ondemand, performance, power save, and user space governors
    - The simple_ondemand and performance governors are not included in the MSM™ kernel build
    - The msm_adreno_tz governor was added as a proprietary governor
    - The source code for the governors is located at the same kernel/drivers/devfreq directory
  - Device drivers (Devfreq enabled)
    - Provides device statistics (total time, busy time) and current frequency to governors
    - A callback to be used by devfreq core to set the newly calculated frequency

Devfreq GPU DCVS Governor (2 of 4)

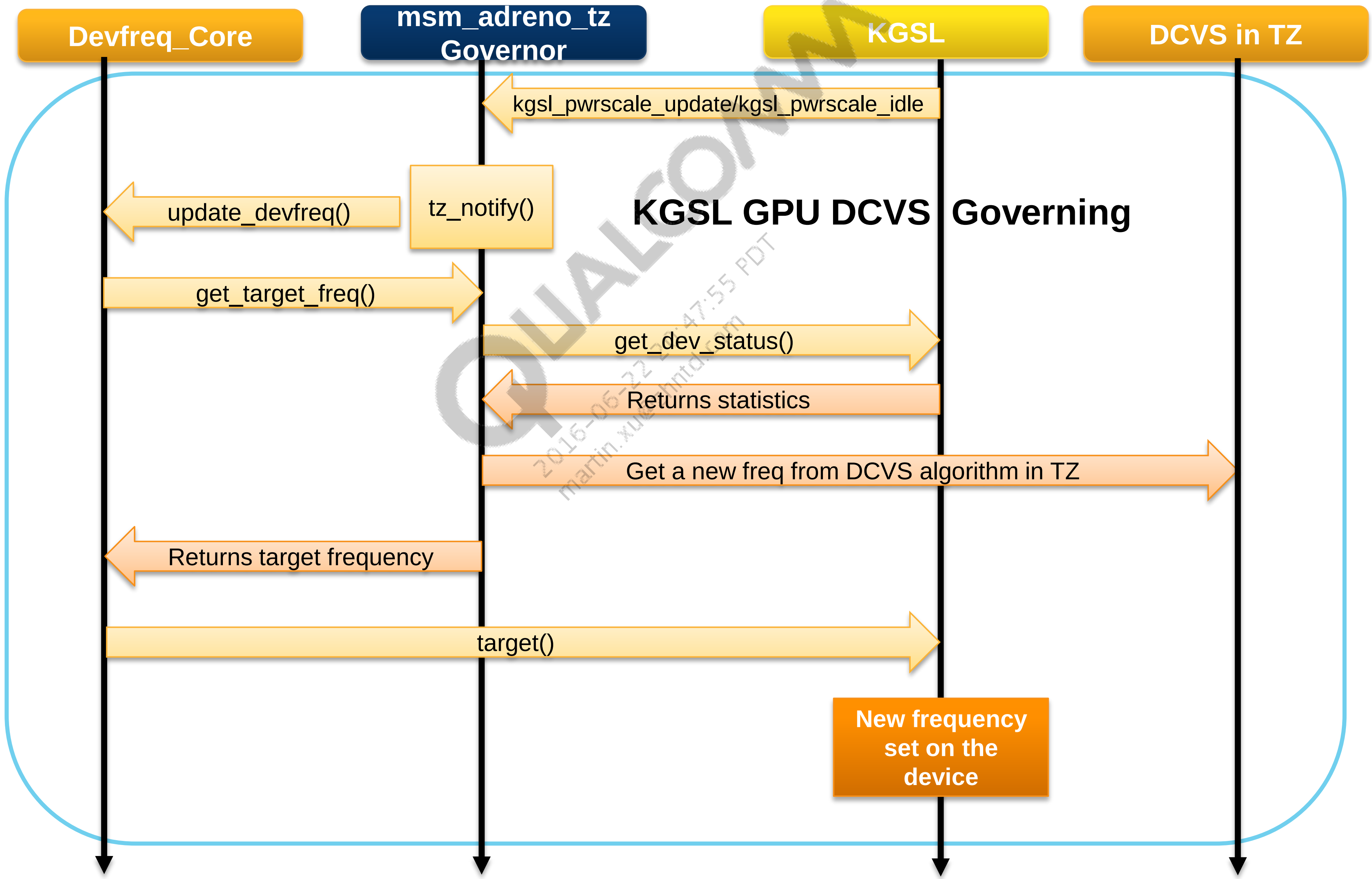
- Devfreq-enabled governor (Governor) exposes two callback functions

Callbacks	What it does	Called by
get_target_freq()	Calculates and returns target frequency to be set	Devfreq_core
event_handler()	Notifies the governor about lifecycle events	

- Device driver (driver) implements three callbacks for governors and devfreq core

Callbacks	What it does	Called by
get_dev_status()	Provide device performance statistics (total time, busy time)	Governor
get_cur_freq()	Provides the devices current frequency	Governor
target()	Set the devices current frequency	Devfreq_core

Devfreq GPU DCVS Governor (3 of 4)



Devfreq GPU DCVS Governor (4 of 4)

- KGSL driver and msm_adreno_tz governor* do not use time interval polling. Instead, the KGSL driver communicates with the governor directly at moments that are important for the driver.
  1. KGSL driver uses Linux notifiers mechanism to call the governor's tz_notify() function.
  2. tz_notify() just calls update_devfreq().
  3. update_devfreq() calls the governor's get_target_freq() function.
  4. get_target_freq() gets the device status information (total and busy time) and calls the DCVS algorithm executed in TZ for a new value for the device frequency.
  5. The reevaluated frequency is passed to the driver with its target() function.

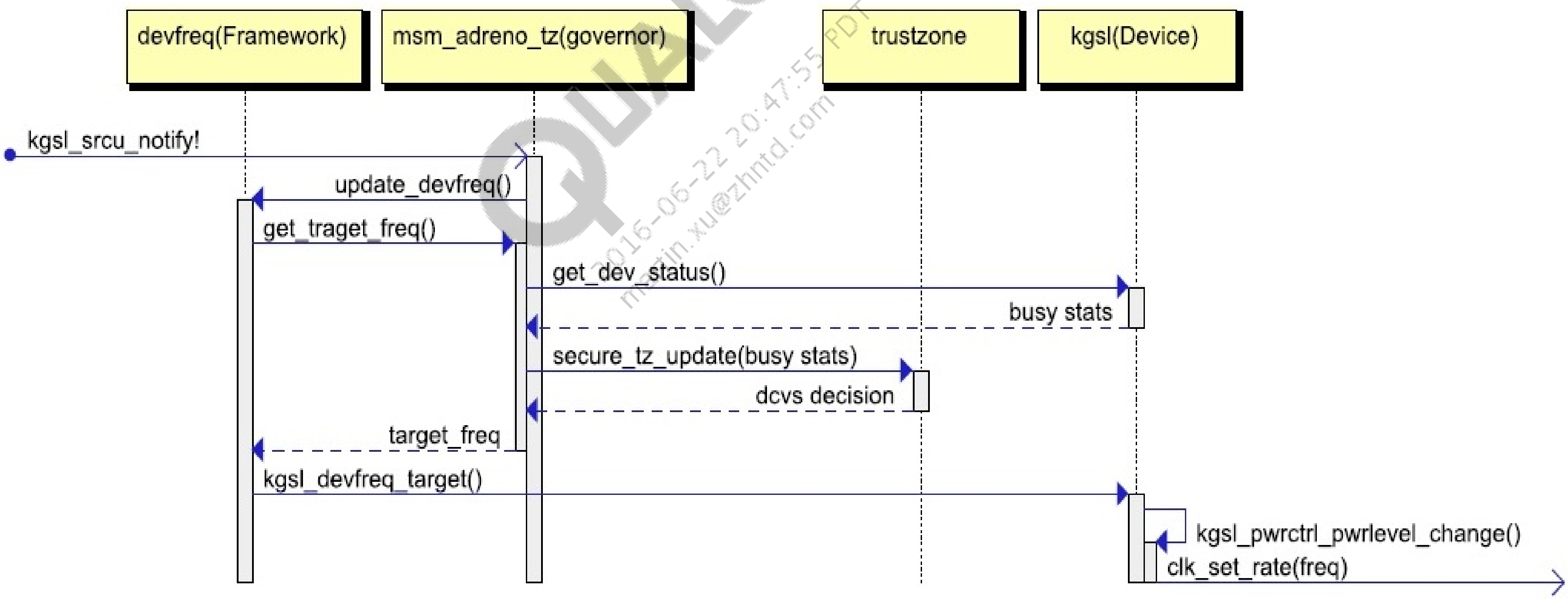
* With Devfreq adaptation, kgsl_pwrscale DCVS,and its policies, relevant sysfs files are now deprecated.

Devfreq GPU DCVS Governor – Code Sequence

GPU DCVS

To vote for the right clock we use one devfreq device and its governor.

DEVICE	**GOVERNOR**
======	==========
kgsl(kgsl_pwrscale.c)	**msm-adreno-tz (governor_msm_adreno_tz.c)**



GPU Bus DCVS

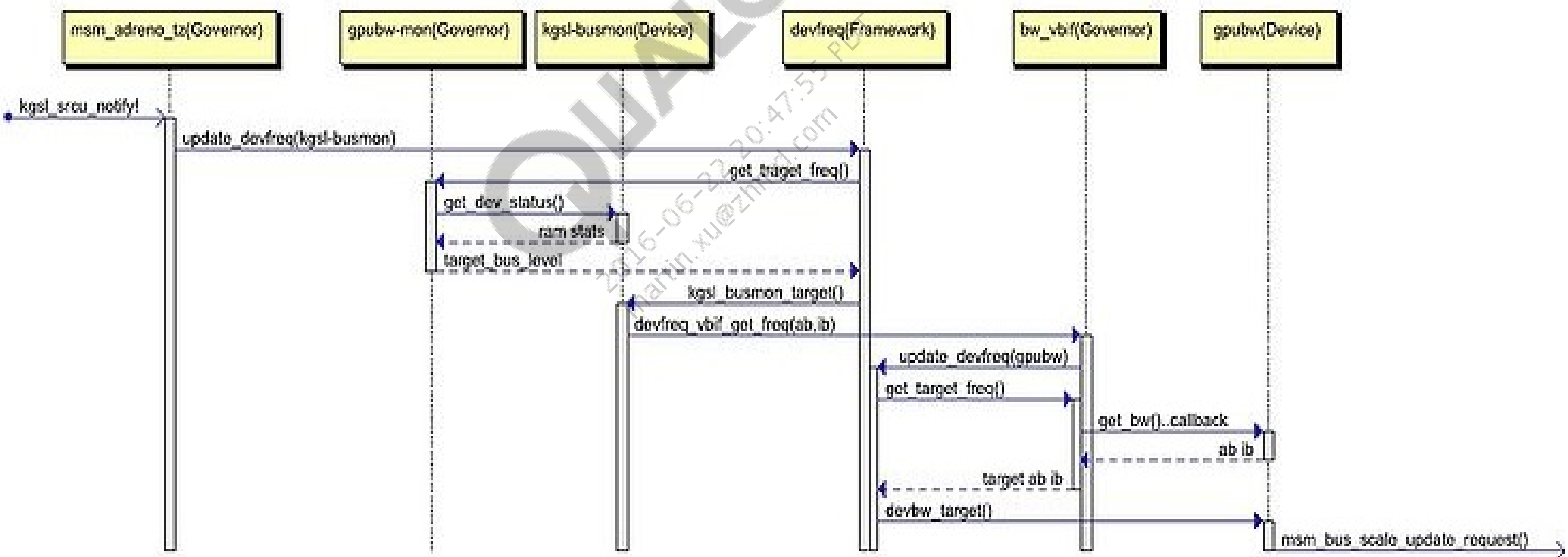
- GPU DCVS vs. GPU bus DCVS
  - GPU DCVS has been used as an umbrella term for the Adreno GPU clock and bus bandwidth voting mechanism
  - Depending on each chipset's clock-level configuration, GPU bus DCVS may or may not be available
    - For low-tier chipsets, GPU DCVS takes care of GPU bus bandwidth voting that is statically mapped to each GPU clock
    - Checking /sys/class/kgsl/kgsl-3d0/bus_split
      - 1 – GPU bus DCVS is enabled→Dynamic bus bandwidth mapping to GPU clocks
      - 0 – GPU bus DCVS is not enabled→Static bus bandwidth mapping to GPU clocks
  - However, in the Adreno DCVS governor algorithm adaptation to the Devfreq framework, GPU DCVS is for the GPU clock and GPU bus DCVS is for bus bandwidth voting. The algorithm provides the opportunity to decouple the GPU clocks setting from the GPU bus bandwidth voting for better power management.
    - System bus voting (RAM) and GMEM bandwidth voting are still tightly coupled.

GPU Bus DCVS – Code Sequence

Bus Modifier Computation.

To make bus calculations and vote for bus we use three devices and their respective governor.

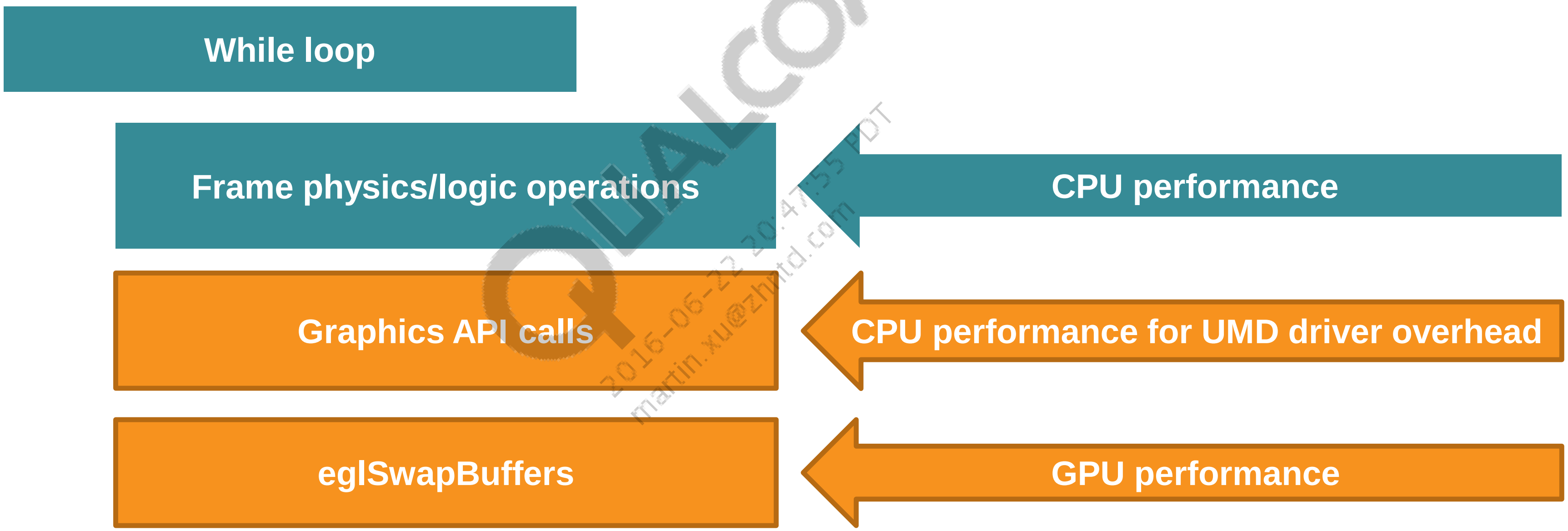
DEVICE	GOVERNOR
kgsl(adreno.c/kgsl_pwrscale.c)	msm-adreno-tz (governor_msm_adreno_tz.c - governor to initiate b/w calculations)
kgsl-busmon(adreno.c/kgsl_pwrscale.c)	gpubw-mon (governor_gpubw_mon.c - governor that calculates what the b/w should it)
gpubw(devfreq_devbw.c)	bw_vbif (governor_bw_vbif.c - governor that finally does the b/w voting)



Adreno Software Performance Overview

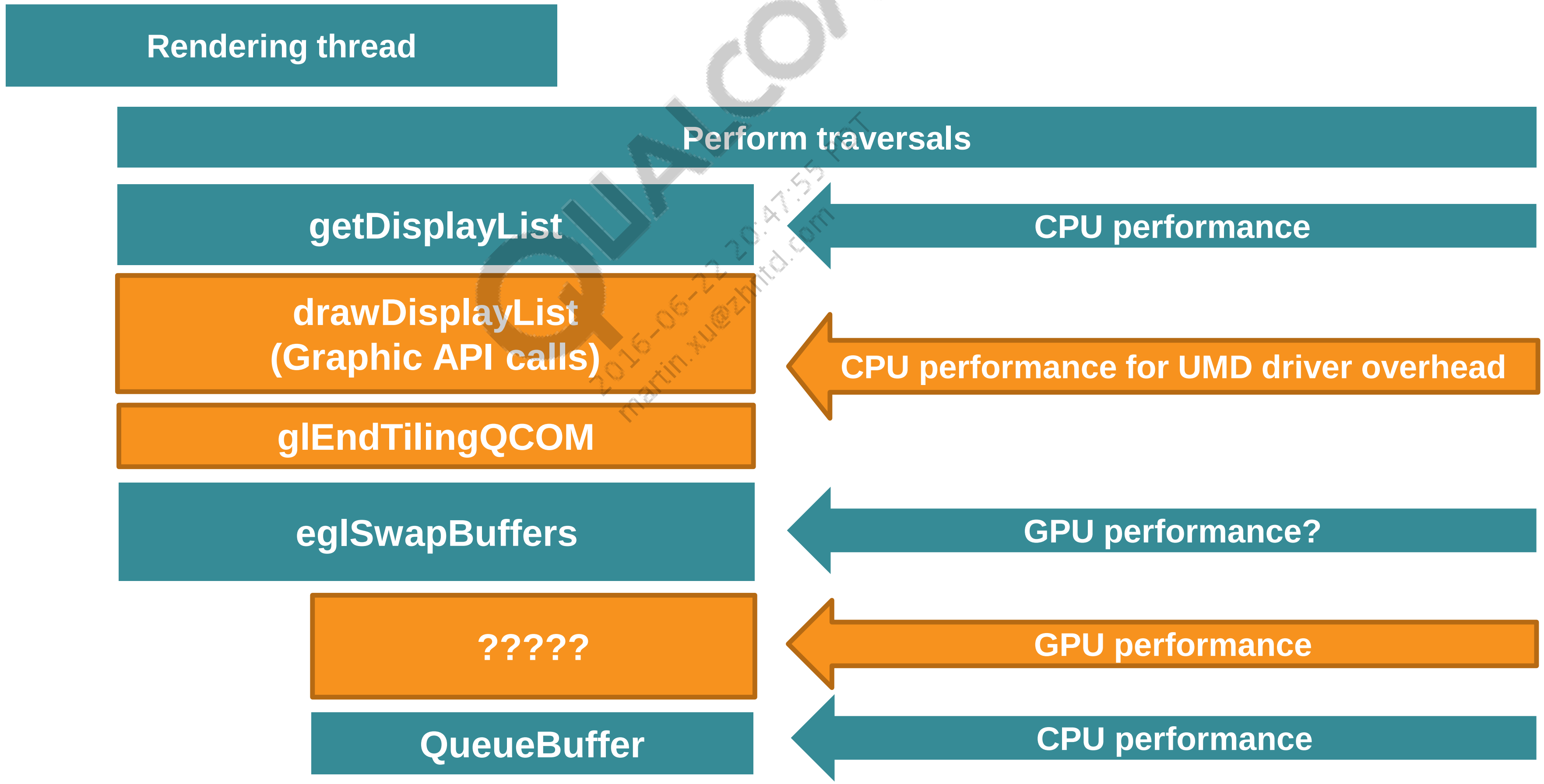
Android Graphics Performance Analysis Overview (1 of 5)

- Typical frame rendering and performance points in an OpenGL ES application



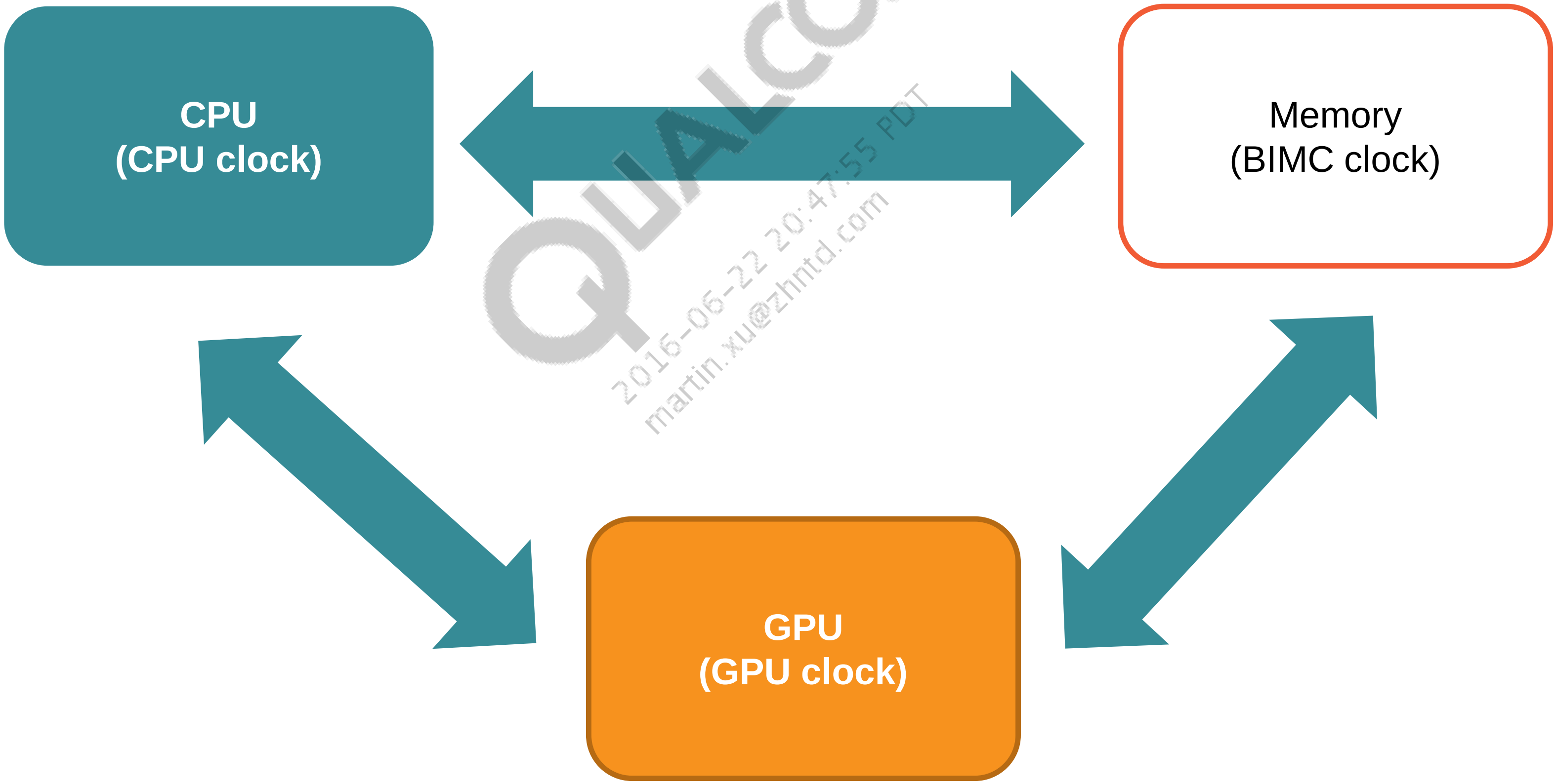
Android Graphics Performance Analysis Overview (2 of 5)

- Android HWUI application



Android Graphics Performance Analysis Overview (3 of 5)

- Hardware blocks associated to overall picture



Android Graphics Performance Analysis Overview (4 of 5)

- GPU performance issues can instead be complex system issues (such as CPU scheduling/ CPU DCVS or DDR/Memory).*
  - GPGPU→OpenCL, Renderscript
- Performance GFX domain
  - UMD performance optimization
    - When the driver overhead for a specific gl API is identified
  - GPU performance optimization
    - When GPU DCVS does not meet the performance requirement
- Performance non-GFX domain
  - Other CPU performance optimization (scheduling, CPU clock governance)
  - System performance optimization (bus clock governance)
  - Application issues (glFinish calls, mid-frame resolve)

* GPGPU performance is out of the scope of this document. GPGPU training will cover this separately.

Android Graphics Performance Analysis Overview (5 of 5)

- The BIG PICTURE
  - Adreno GPU Performance Management software is based on a specific system's GPU usage, increasing the GPU's clock to meet the required performance level.
- Terms
  - CPU/System bound – An application that spends most of its time doing CPU/system tasks and/or its performance is dominated by CPU/system
  - GPU bound – An application that spends most of its time doing GPU tasks and/or its performance is dominated by the GPU
- Performance vs. power
  - The GPU DCVS governor is at the heart of balancing between GPU performance and GPU power
    - tz_adreno_governor is power-centric, not performance-centric governor, meaning:
      - It will only increase GPU clock when it is necessary
      - It starts GPU at low level and goes up, not at high level, and stays there

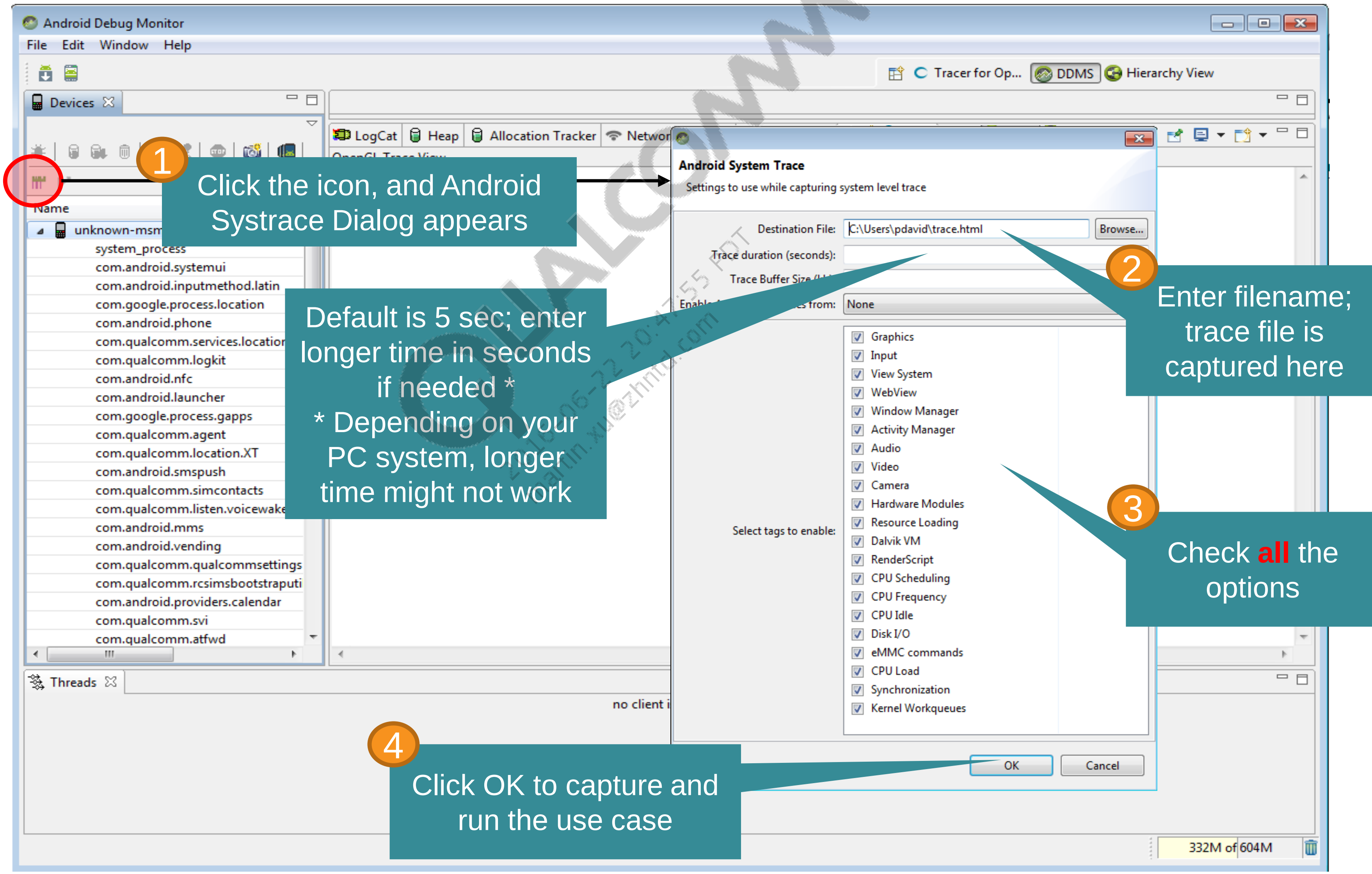
Android GFX Performance Analysis with Systrace (Hands-On Session)

Android GFX Performance Analysis with Systrace (1 of 4)

- Systrace is an indispensable tool for Android performance issue analysis
  - Systrace capture shows:
    - CPU, GPU, BIMC (System Memory), and other clocks
    - OpenGL ES calls (once enabled in Developer Options)
    - Other critical latency information (GPU wake up, touch latency)
- To effectively capture all the relevant data on the trace, install the latest Android SDK and tools
- Once Android SDK and tools are correctly installed, make sure the following is also set:
  - <android_sdk_root>\tools is in your PATH variable
  - <android_skd_root>\platform-tools is in your PATH variable (for adb)
  - adb root
  - adb remount
- Run monitor.bat* from <android_sdk_root>/tools directory to begin

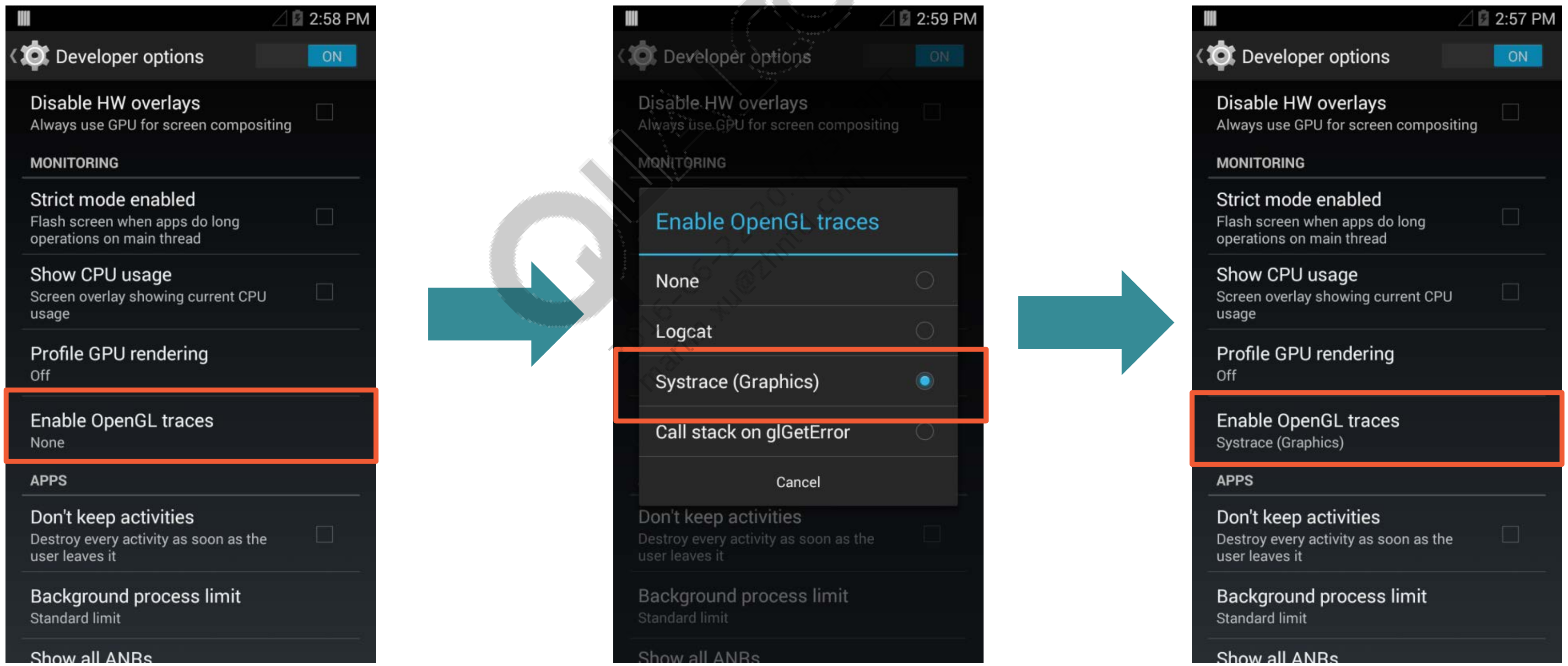
* Systrace command line functionalities will not be covered in this training.

Android GFX Performance Analysis with Systrace (2 of 4)



Android GFX Performance Analysis with Systrace (3 of 4)

- An output .html file can be viewed only by a Chrome browser.
- For OpenGL ES APIs logging with Systrace, go to Settings→Developer Options and do the following:

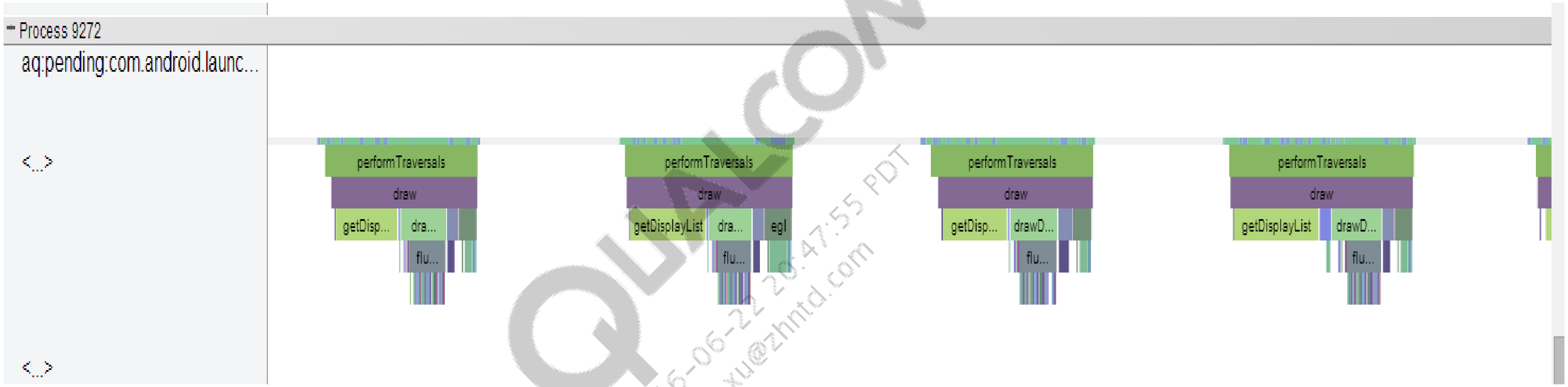


- Finally, do adb shell stop and start:

```
>adb shell stop
>adb shell start
```

Android GFX Performance Analysis with Systrace (4 of 4)

- Sample Output (App menu scroll)

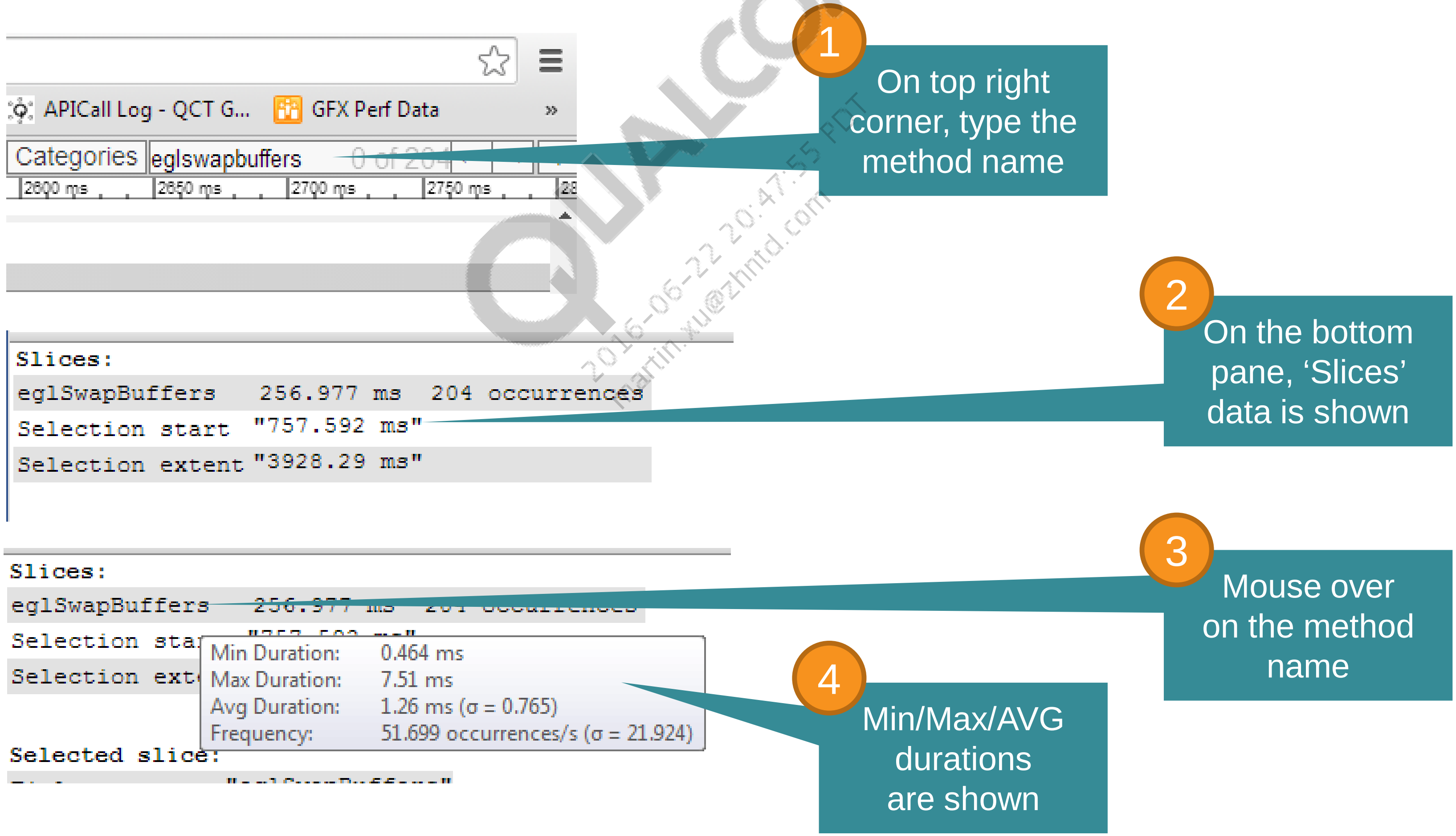


- Now it is time for hands-on and extensive analysis!

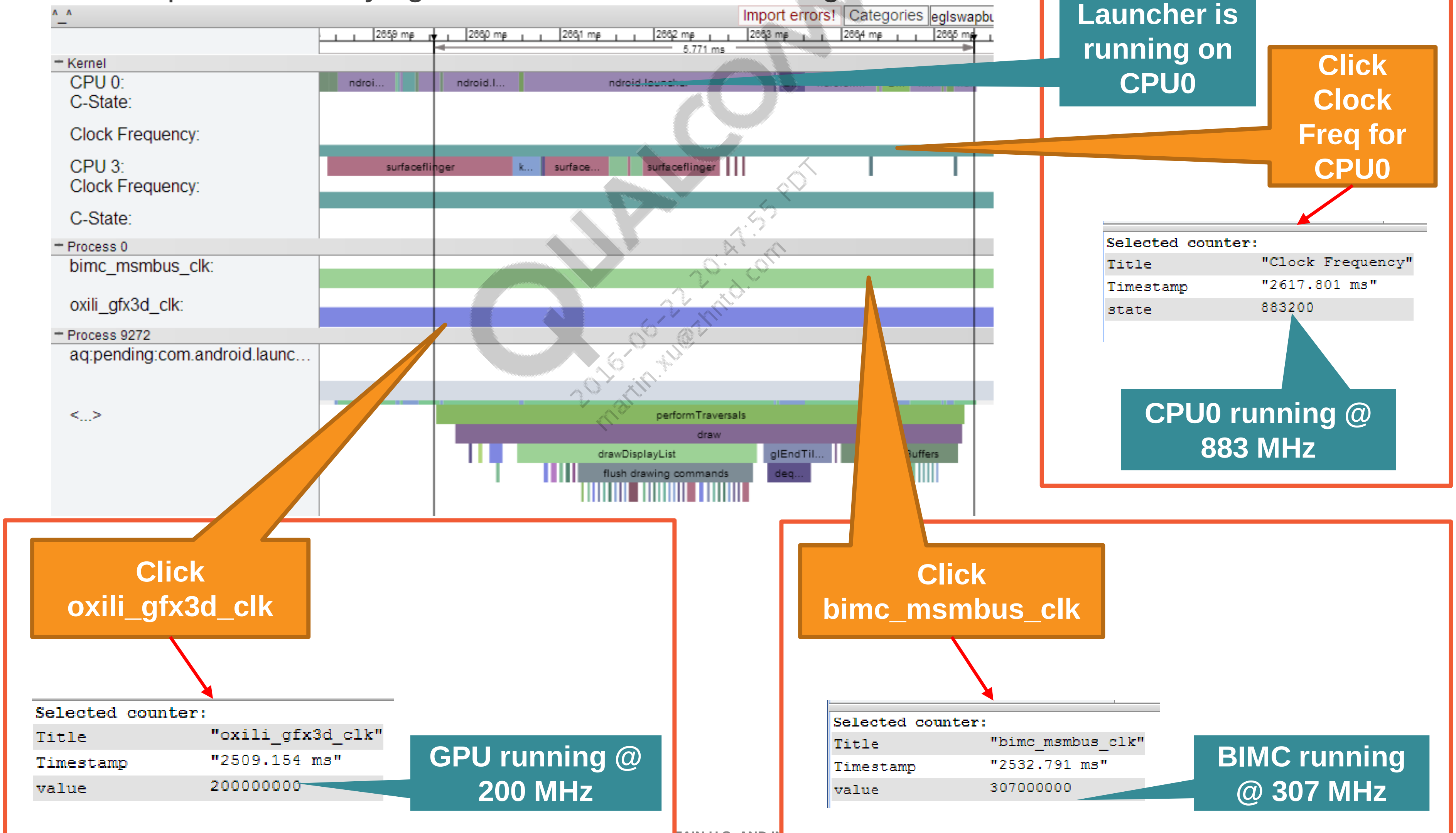
Advanced Systrace Analysis Techniques for GFX Performance Issues

Advanced Systrace Analysis Techniques for GFX Performance Issues (1 of 7)

- Technique 1 – Finding out minimum/maximum/average latency
  - Let us try to find eglSwapBuffers min/max/avg latency.

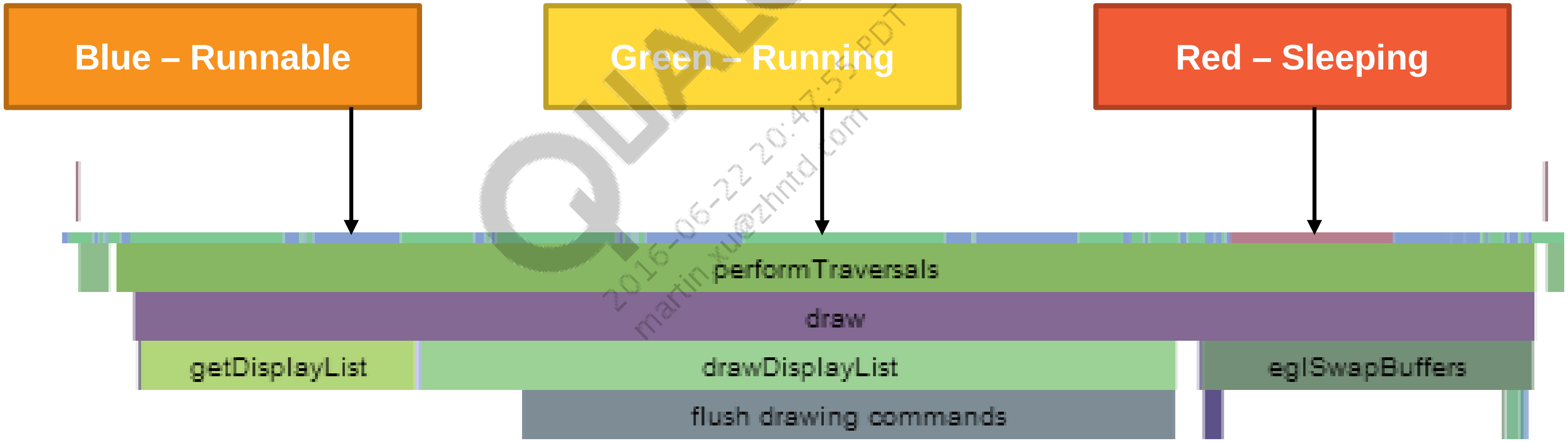


- Technique 2 – Identifying CPU/GPU/BIMC clocks at given duration



Advanced Systrace Analysis Techniques for GFX Performance Issues (3 of 7)

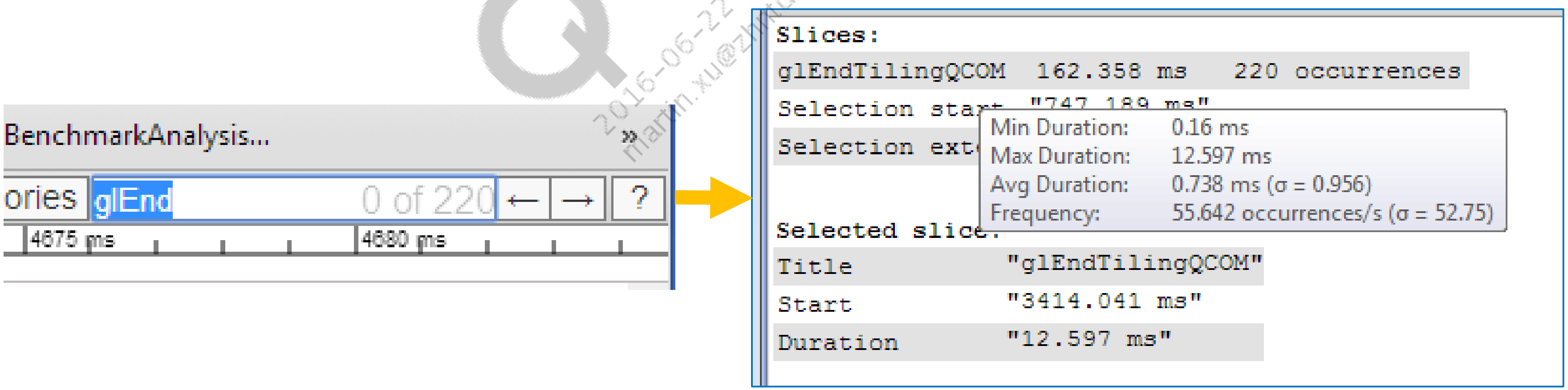
- Technique 3 – Identifying process states
  - GL call (UMD) latency can be misled by CPU scheduling – A process can be in running, runnable, or in uninterruptable sleep state. These states are shown in Systrace with different colors.



  - To find the actual block time, we must subtract Runnable/Sleeping times, only counting Running time.

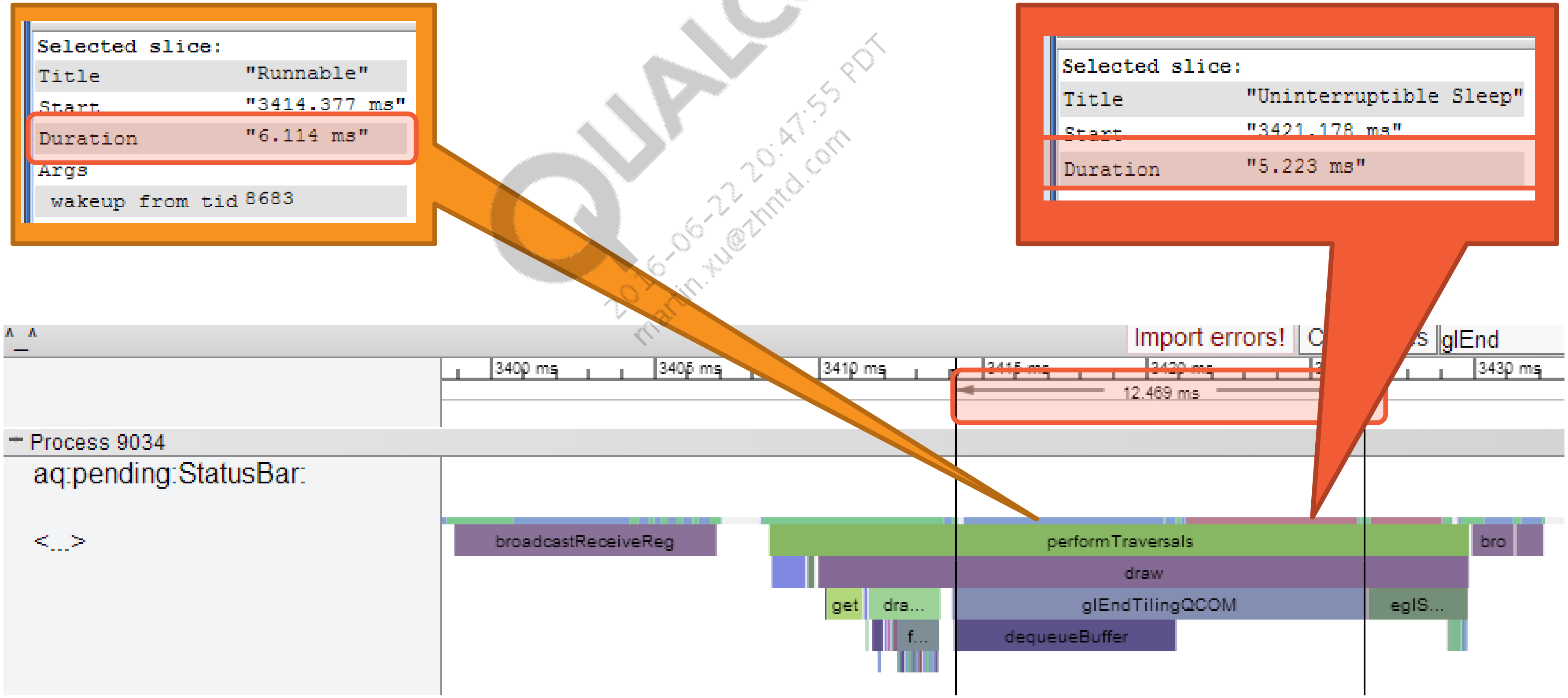
Advanced Systrace Analysis Techniques for GFX Performance Issues (4 of 7)

- Put all three techniques together to analyze one example.
  - With one Systrace capture, one glEndTilingQCOM taking >12 ms with Status Bar rendering is found, system.ui process (Technique 1).



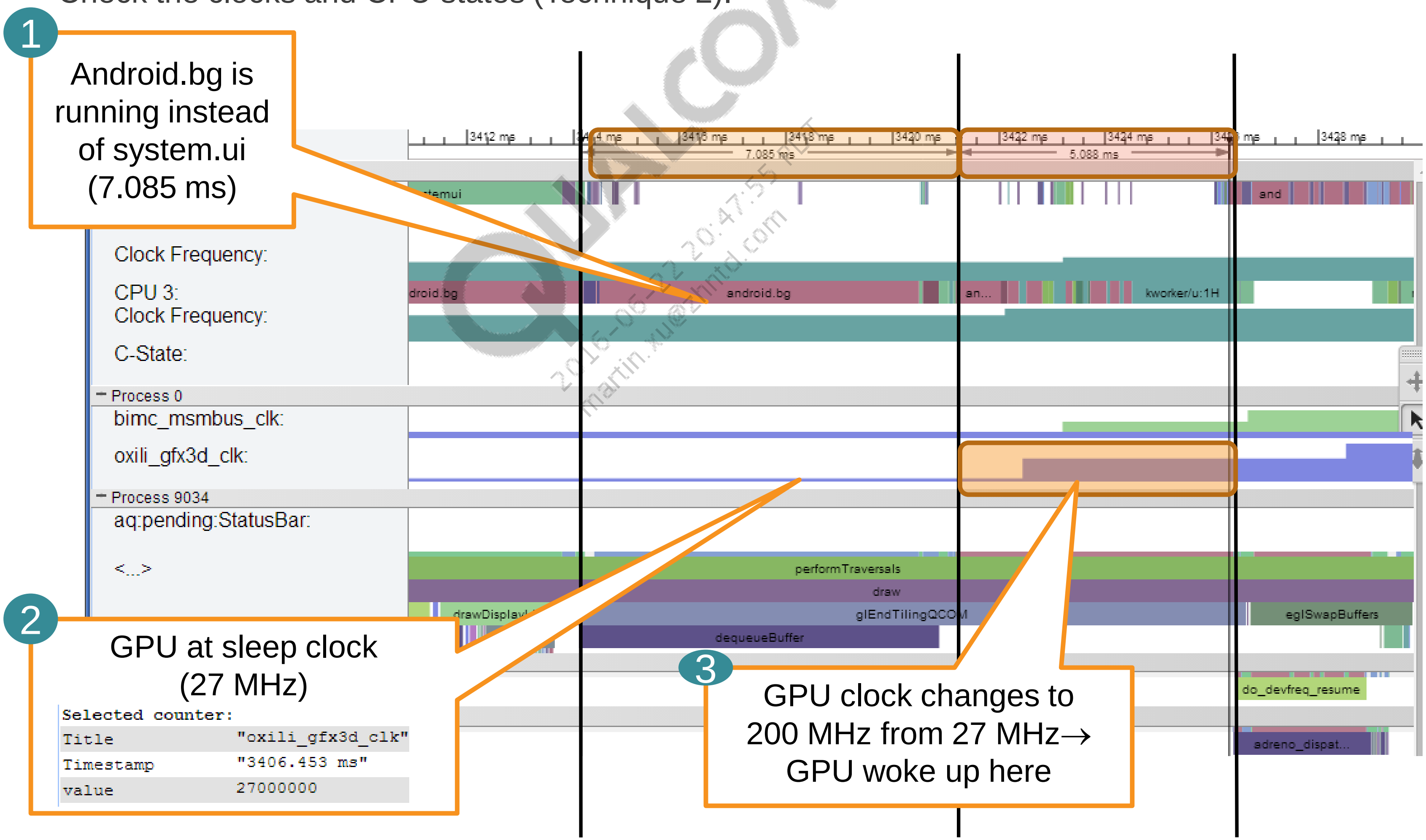
Advanced Systrace Analysis Techniques for GFX Performance Issues (5 of 7)

- Put all three techniques together to analyze one example (continued)
  - But for more than 90% of the 12 ms, glEndTilingQCOM is either in the Runnable (not actually running) or Uninterruptable Sleep state (Technique 3).



Advanced Systrace Analysis Techniques for GFX Performance Issues (6 of 7)

- Put all three techniques together to analyze one example. (continued)
  - Check the clocks and CPU states (Technique 2).



Advanced Systrace Analysis Techniques for GFX Performance Issues (7 of 7)

- Put all three techniques together to analyze one example (continued)
  - Analysis conclusion?

1

Android.bg is running instead of system.ui (7.085 ms)

2

GPU at sleep clock (27 MHz)

Selected counter:

Title	"oxili_gfx3d_clk"
Timestamp	"3406.453 ms"
value	27000000

Out of 12 ms, ~7 ms is spent outside of the GFX domain.

Turning all CPU cores can help this issue.

(More cores, less chance of a given process do context switching)

3

GPU clock changes to 200 MHz from 27 MHz→ GPU woke up here

~5 ms GPU wakeup latency is known latency.

Trying to disable SLUMBER by setting idle_timer to 1000000 could be used for further confirmation/verification point.

Logging/Debugging

Logging (1 of 3)

- KGSL power events logging is available through Linux Ftrace.
- All KGSL events, not only power events, can be found under Ftraces's events/kgsl directory.
  - sys/kernel/debug/tracing/events/kgsl
  - ls sys/kernel/debug/tracing/events/kgsl will list all KGSL Ftrace events
- The following events are KGSL power events:

Logging event	Log description
kgsl_a3xx_irq_status	IRQ status
kgsl_clk	GPU clock status (logged when GPU clock changes)
kgsl_rail	GPU power rail on/off status
kgsl_irq	GPU IRQ on/off status
kgsl_bus	GPU bus voting on/off status
kgsl_pwrlevel	GPU power-level changes
kgsl_buslevel	GPU bus voting-level changes
kgsl_pwrstats	GPU usage statistics changes/updates
kgsl_pwr_set_state	GPU Power state (ACTIVE, NAP, SLUMBER, etc.) changes
kgsl_pwr_active_count	GPU's current active count (0 means not active)

Logging (2 of 3)

To collect trace logs

1. Mount debugfs
   – adb shell mount -t debugfs none /sys/kernel/debug
2. See available events
   – adb shell cat /sys/kernel/debug/tracing/available_events
   – adb shell cat /sys/kernel/debug/tracing/available_events | grep kgsl
3. Increase the trace buffer size
   – echo 16384 > /sys/kernel/debug/tracing/buffer_size_kb
4. Make a text file with the events you want, some possibilities are:
   – kgsl:kgsl_a2xx_irq_status
   – kgsl:kgsl_a3xx_irq_status
   – kgsl:kgsl_clk
   – kgsl:kgsl_irq
   – kgsl:kgsl_rail
   – kgsl:kgsl_bus
   – kgsl:kgsl_pwrlevel
   – kgsl:kgsl_buslevel
   – kgsl:kgsl_pwrstats
   – kgsl:kgsl_pwr_set_state
   – kgsl:kgsl_pwr_request_state
   – kgsl:kgsl_active_count
5. Push it
   – adb push events.txt /sys/kernel/debug/tracing/set_event
6. Run the use case
7. Pull the log
   – adb pull /sys/kernel/debug/tracing/trace

Logging (3 of 3)

- KGSL Power Events Logging
  - Analyzing the log without a general understanding of KGSL code is not easy and not required for customers
  - When a GPU power-related issue is filed, it is often required for customers to get the KGSL power events log
  - Use case by use case, QTI will provide analysis and help customers understand the log, and the issue behind the log if any

Debugging (1 of 4)

- When it comes to debugging GPU power issues, customers are asked to provide logs and results on changing a few GPU settings through sysfs node changes.

Sysfs operations	Description	Command sequences
Force the GPU clocks/bus vote/power rail always on	Forcing the GPU clocks/bus vote/power rail always on will prevent GPU from power collapsing.	cd /sys/class/kgsl/kgsl-3d0 echo 1 > force_clk_on /* Clocks are not SW gated during standard rendering */ echo 0 > force_clk_on /* Resume SW gating of clocks */ echo 1 > force_rail_on /* Power rail never turned off */
Change the idle timer or disable SLEEP/SLUMBER through sysfs	Higher values may be tried for power reasons, or just set a high value (it is in ms) to disable SLEEP/SLUMBER entirely	cd /sys/class/kgsl/kgsl-3d0 echo 1000000 > idle_timer
Turn off/on GPU DCVS through sysfs	Use the performance governor for max GPU frequency or the power governor for min GPU frequency command sequence	cd /sys/class/kgsl/kgsl-3d0/devfreq echo performance > governor cd /sys/class/kgsl/kgsl-3d0/devfreq echo powersave > governor

Debugging (2 of 4)

Sysfs operations	Description	Command sequences
Find the available GPU power levels through sysfs	To ensure you request a supported frequency check availability	cat /sys/class/kgsl/kgsl-3d0/gpu_available_frequencies
Set the min/max power level for GPU	To test GPU power levels power consumption or have a fixed GPU clock profiling	cd /sys/class/kgsl/kgsl-3d0 echo 1 > max_pwrlevel echo 1 > min_pwerlevel //this will fix GPU clock to be at power level 1's GPU Frequency
GPU busy statistics	The first value is the GPU busy time and second one is the total system time (~1 sec)	cd /sys/class/kgsl/kgsl-3d0/ cat gpubusy //(first_value/second_value)*100 gives percentage of the last sec the GPU core was //busy
Checking running GPU clock	To ensure GPU is running at optimal clock frequency for given use case	cd /sys/kernel/debug/clk/oxili_gfx3d_clk/ cat measure

Checking GPU bus vote	To ensure GPU is voting at optimal bus bandwidth	cd /sys/kernel/debug/msm-bus-dbg/client-data/
		cat grp3d
		
		//master 26 slave 512 is for system bus (BIMC)
		//master 89 slave 604 is for OCEM
		//example output
		// ab: 120000000 0 → Bus at 1.2 Gbps
		ab/ GMEM at 0 Gbps ab
		// ib: 245600000 5280000000 → Bus at 2.456 Gbps
		ib/ GMEM at 5.28 Gbps ib

Debugging (3 of 4)

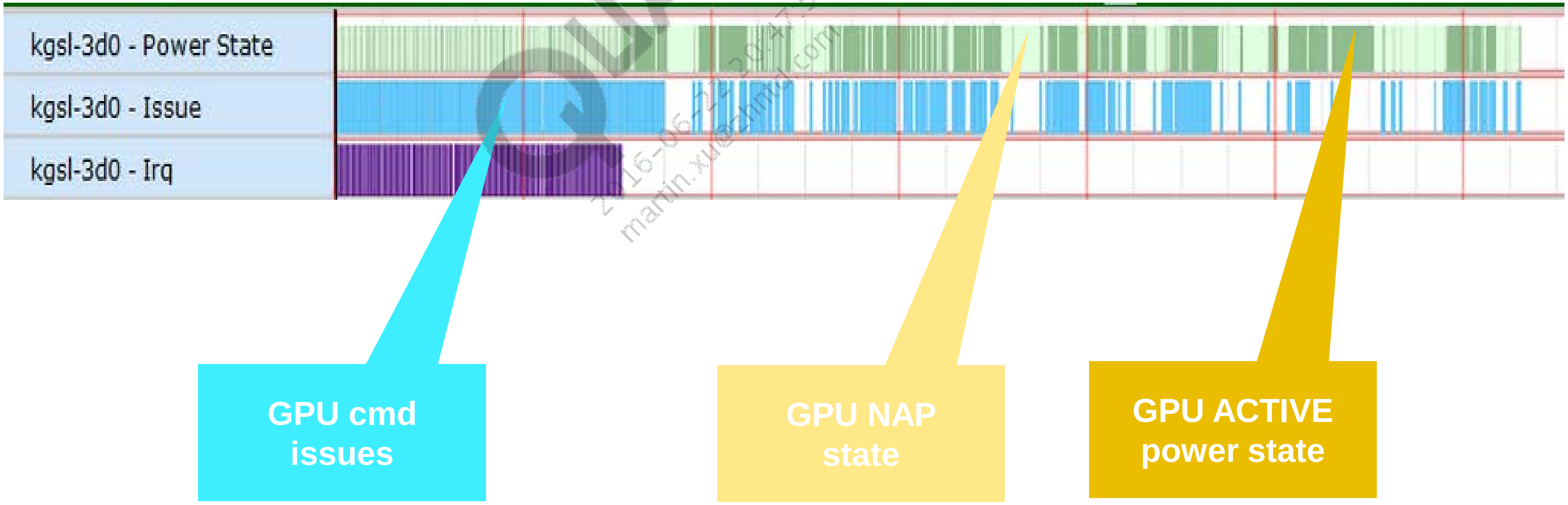
- Android Systrace also often shows GPU clock change information and bus clock information that can be useful to debug GPU clock-related issues
- For more information on Android Systrace and how to use it, see [R1].

Selected counter:	
Title	"oxili_gfx3d_clk_src"
Timestamp	"2583.904 ms"
value	320000000

Selected counter:	
Title	"bimc_clk"
Timestamp	"1315.617 ms"
value	512000000

Debugging (4 of 4)

- QView (Qualcomm Snapdragon™ Performance Visualizer) Tool with GPU Profiling can also be used for GPU power state changes along with other important information
- QView may require additional kernel configuration enabled to enable GPU Profiling. See [Q2] for QView how-tos.



Case Study: GPU Power Profiling

GPU Power Profiling

Important reminder: Three major GPU power factors

Power Factor	Controlling Mechanism	Logging/Debugging Info
GPU Clock	GPU DCVS	Sysfs/Debugfs Nodes; KGSL Power Events Logs
GPU Bus Vote	GPU Bus DCVS	Debugfs Nodes; KGSL Power Event Logs
GPU Power States	GPU Power State Management	KGSL Power Event Logs

Note: All three major factors will likely be different from the OEM target device and the MTP (assuming the same MSM chipset) **if the display resolution is different**. (**Higher resolution will often lead to higher GPU power**).

GPU Power Profiling – GPU Clock During YouTube Full Screen Playback (1 of 8)

- Check available GPU frequencies for the target:

```
root@msm8994:/ # cat /sys/class/kgsl/kgsl-3d0/gpu_available_frequencies
cat /sys/class/kgsl/kgsl-3d0/gpu_available_frequencies
600000000 510000000 450000000 390000000 305000000 190000000
```

- Check the currently running GPU clock during playback:

```
root@msm8994:/ # cat /d/clk/oxili_gfx3d_clk/measure
cat /d/clk/oxili_gfx3d_clk/measure
189999166
```

- We are running at the lowest clock level.
- To continuously check the running GPU clock, use the following command:

```
# while true; do cat /d/clk/oxili_gfx3d_clk/measure; sleep 0.05; done
```

Note: All example commands were run after “adb root” and “adb remount”

GPU Power Profiling – GPU Clock During YouTube Full Screen Playback (2 of 8)

- Assume that on a target device the GPU is running at 305 MHz, or not at the lowest GPU clock level

- First check min_pwrlevel and determine if it is indeed set at the lowest level

```
root@msm8994:/ # cat /sys/class/kgsl/kgsl-3d0/min_pwrlevel
cat /sys/class/kgsl/kgsl-3d0/min_pwrlevel
5
```

- If min_pwrlevel is not set at the lowest level, the OEM needs to check possible changes from OEM side

- If min_pwrlevel is set correctly, analyze the GPU busy status more closely

GPU Power Profiling – GPU Clock During YouTube Full Screen Playback (3 of 8)

- The GPU busy status or GPU busy rate can be acquired by polling sysfs node and KGSL power event trace
- Polling /sys/class/kgsl/kgsl-3d0/gpubusy can provide the GPU busy rate on a sampling time basis. It can be useful to check for **large deltas** between the MTP and the OEM's target device.
- For **more accurate comparisons**, KGSL power event (kgsl_pwr_stats) trace is more useful (when filing a case, capture kgsl_pwrstats event logs rather than gpubusy logs)
- For GPU clock changes, the devfreq's trans_stat can be used

GPU Power Profiling – GPU Clock During YouTube Full Screen Playback (4 of 8)

- GPU busy rate from /sys/class/kgsl/kgsl-3d0/gpubusy polling

```
root@msm8994:/ # while true; do cat /sys/class/kgsl/kgsl-3d0/gpubusy; sleep 0.05; done
ue; do cat /sys/class/kgsl/kgsl-3d0/gpubusy; sleep 0.05; done                <
 149799 1000251
 149799 1000251
 149799 1000251
 149799 1000251
 149799 1000251
 149799 1000251
 149799 1000251
 149799 1000251
 149799 1000251
 149799 1000251
 149799 1000251
 149799 1000251
 149799 1000251
 149799 1000251
 148848 1049617
 148848 1049617
 148848 1049617
 148848 1049617
 148848 1049617
 148848 1049617
 148848 1049617
 148848 1049617
 148848 1049617
 148848 1049617
 148848 1049617
 148848 1049617
 148848 1049617
 148848 1049617
 148848 1049617
```

149799 1000251

GPU ACTIVE time per *sampling*

GPU Total time per *sampling*

Busy Rate = ACTIVE Time / Total Time

In this sample, it is 149799/1000251 = 15% per 1 second sampling *

GPU Power Profiling – GPU Clock During YouTube Full Screen Playback (5 of 8)

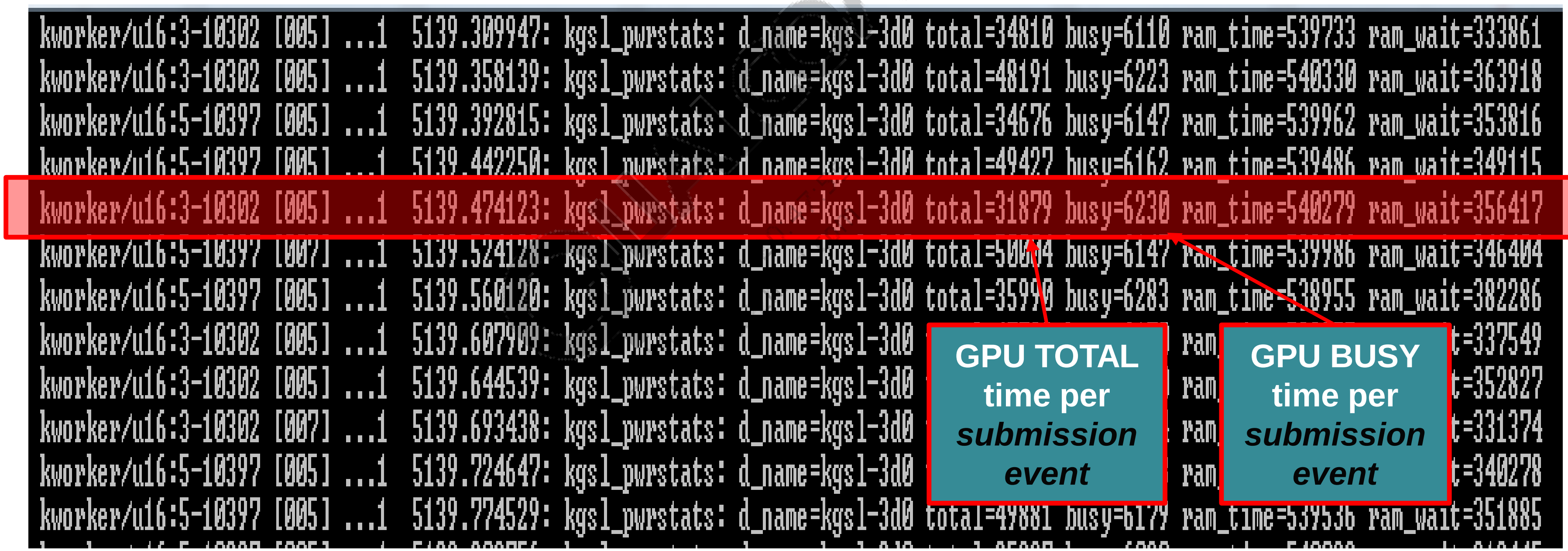
- GPU busy rate from KGSL Event tracing.

```
>adb shell
#cd d/tracing/events/kgsl/kgsl_pwrstats
#echo 1 > enable
#cd /d/tracing
#cat trace_pipe | grep kgsl_pwrstats
```

```
kworker/u16:3-10302 [005] ...1  5139.309947: kgsl_pwrstats: d_name=kgsl-3d0 total=34810 busy=6110 ram_time=539733 ram_wait=333861
kworker/u16:3-10302 [005] ...1  5139.358139: kgsl_pwrstats: d_name=kgsl-3d0 total=48191 busy=6223 ram_time=540330 ram_wait=363918
kworker/u16:5-10397 [005] ...1  5139.392815: kgsl_pwrstats: d_name=kgsl-3d0 total=34676 busy=6147 ram_time=539962 ram_wait=353816
kworker/u16:5-10397 [005] ...1  5139.442250: kgsl_pwrstats: d_name=kgsl-3d0 total=49427 busy=6162 ram_time=539486 ram_wait=349115
kworker/u16:3-10302 [005] ...1  5139.474123: kgsl_pwrstats: d_name=kgsl-3d0 total=31879 busy=6230 ram_time=540279 ram_wait=356417
kworker/u16:5-10397 [007] ...1  5139.524128: kgsl_pwrstats: d_name=kgsl-3d0 total=50004 busy=6147 ram_time=539986 ram_wait=346404
kworker/u16:5-10397 [005] ...1  5139.560120: kgsl_pwrstats: d_name=kgsl-3d0 total=35990 busy=6283 ram_time=538955 ram_wait=382286
kworker/u16:3-10302 [005] ...1  5139.607909: kgsl_pwrstats: d_name=kgsl-3d0 total=47789 busy=6170 ram_time=539855 ram_wait=337549
kworker/u16:3-10302 [005] ...1  5139.644539: kgsl_pwrstats: d_name=kgsl-3d0 total=36629 busy=6180 ram_time=539731 ram_wait=352827
kworker/u16:3-10302 [007] ...1  5139.693438: kgsl_pwrstats: d_name=kgsl-3d0 total=48900 busy=6114 ram_time=540366 ram_wait=331374
kworker/u16:5-10397 [005] ...1  5139.724647: kgsl_pwrstats: d_name=kgsl-3d0 total=31208 busy=6138 ram_time=539938 ram_wait=340278
kworker/u16:5-10397 [005] ...1  5139.774529: kgsl_pwrstats: d_name=kgsl-3d0 total=49881 busy=6179 ram_time=539536 ram_wait=351885
```

GPU Power Profiling – GPU Clock During YouTube Full Screen Playback (6 of 8)

- GPU busy rate from KGSL Event tracing



```
kworker/u16:3-10302 [005] ...1  5139.309947: kgsl_pwrstats: d_name=kgsl-3d0 total=34810 busy=6110 ram_time=539733 ram_wait=333861
kworker/u16:3-10302 [005] ...1  5139.358139: kgsl_pwrstats: d_name=kgsl-3d0 total=48191 busy=6223 ram_time=540330 ram_wait=363918
kworker/u16:5-10397 [005] ...1  5139.392815: kgsl_pwrstats: d_name=kgsl-3d0 total=34676 busy=6147 ram_time=539962 ram_wait=353816
kworker/u16:5-10397 [005] ...1  5139.442250: kgsl_pwrstats: d_name=kgsl-3d0 total=49427 busy=6162 ram_time=539486 ram_wait=349115
kworker/u16:3-10302 [005] ...1  5139.474123: kgsl_pwrstats: d_name=kgsl-3d0 total=31879 busy=6230 ram_time=540279 ram_wait=356417
kworker/u16:5-10397 [007] ...1  5139.524128: kgsl_pwrstats: d_name=kgsl-3d0 total=500[illegible]4 busy=6147 ram_time=539986 ram_wait=346404
kworker/u16:5-10397 [005] ...1  5139.560120: kgsl_pwrstats: d_name=kgsl-3d0 total=359[illegible]0 busy=6283 ram_time=538955 ram_wait=382286
kworker/u16:3-10302 [005] ...1  5139.607909: kgsl_pwrstats: d_name=kgsl-3d0 ... ram ... t=337549
kworker/u16:3-10302 [005] ...1  5139.644539: kgsl_pwrstats: d_name=kgsl-3d0 ... ram ... t=352827
kworker/u16:3-10302 [007] ...1  5139.693438: kgsl_pwrstats: d_name=kgsl-3d0 ... ram ... t=331374
kworker/u16:5-10397 [005] ...1  5139.724647: kgsl_pwrstats: d_name=kgsl-3d0 ... ram ... t=340278
kworker/u16:5-10397 [005] ...1  5139.774529: kgsl_pwrstats: d_name=kgsl-3d0 total=49881 busy=6179 ram_time=539536 ram_wait=351885
```

GPU Power Profiling – GPU Clock During YouTube Full Screen Playback (7 of 8)

- GPU Devfreq trans_stat shows overall GPU clock transitions* for the entire device up time from device power up*.
  - adb shell “cat /sys/class/kgsl/kgsl-3d0/devfreq/trans_stat”

GPU Running Clock at Polling Moment

```
root@msm8994:/sys/class/kgsl/kgsl-3d0/devfreq # cat trans_stat
cat trans_stat
   From  :   To
         :600000000510000000450000000390000000305000000180000000   time(ms)
 600000000:        0       2       0       0       3       1      2210
 510000000:        6       0       2       0       0       0       820
 450000000:        0       6       0       2       1       0       420
 390000000:        0       0       7       0       1       0       950
 305000000:        0       0       0       6       0      31    282100
*180000000:        0       0       0       0      31       0    560120
Total transition : 99
```

Milliseconds spent at each clock

- Cat trans_stat before your usecase, run your usecase, and cat trans_stat again at the end of your usecase, and then check the deltas for GPU clock transitions and residency times.

* Due to the limitation on Devfreq, there is currently no way to reset the statistics table except to reset the device.

GPU Power Profiling – GPU Clock During YouTube Full Screen Playback (8 of 8)

- Before:

```
C:\Users\pdavid>adb shell "cat /sys/class/kgsl/kgsl-3d0/devfreq/trans_stat"
   From  :   To
         :600000000510000000450000000390000000305000000180000000   time(ms)
 600000000:        0        2        0        0        3        1      2210
 510000000:        6        0        2        0        0        0       820
 450000000:        0        6        0        2        1        0       420
 390000000:        0        0        7        0        1        0       950
*305000000:        0        0        0        6        0       37    504840
 180000000:        0        0        0        0       38        0   2243160
Total transition : 112
```

- After (YouTube Playback)

```
C:\Users\pdavid>adb shell "cat /sys/class/kgsl/kgsl-3d0/devfreq/trans_stat"
   From  :   To
         :600000000510000000450000000390000000305000000180000000   time(ms)
 600000000:        0        2        0        0        3        1      2210
 510000000:        6        0        2        0        0        0       820
 450000000:        0        6        0        2        1        0       420
 390000000:        0        0        7        0        1        0       950
*305000000:        0        0        0        6        0       38    517710
 180000000:        0        0        0        0       39        0   2323590
Total transition : 114
```

Clk Transitions	From 305 MHz to 180 MHz	From 180 MHz to 305 MHz	Time spent at 305 MHz	Time spent at 180 MHz
2 (114 – 112)	1 (38 – 37)	1 (39 – 38)	12861 ms (517710 – 504840)	80430 ms (2323590 – 2243260)

References

References

Ref.	Document		
Qualcomm Technologies			
Q1	*Application Note: Software Glossary for Customers*		CL93-V3077-1
Q2	*SPV (QView) 8.0 User Guide*		80-N4717-1
Resources			
R1	*Android Systrace*	http://developer.android.com/tools/help/systrace.html http://developer.android.com/tools/debugging/systrace.html	

Questions?

You may also submit questions to: https://support.cdmatech.com