

---

# Graphics Power and Performance Overview

---

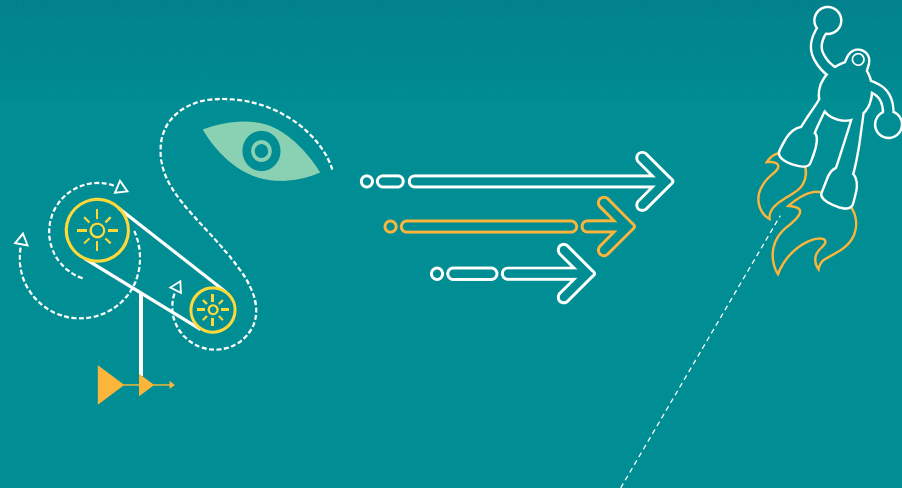


Qualcomm Technologies, Inc.

80-NP885-1 C

Confidential and Proprietary – Qualcomm Technologies, Inc.

**Restricted Distribution:** Not to be distributed to anyone who is not an employee of either Qualcomm Technologies, Inc. or its affiliated companies without the express approval of Qualcomm Configuration Management.



# Confidential and Proprietary – Qualcomm Technologies, Inc.

---

Qualcomm  
2018-07-23 23:39:16 PDT  
songpeng2@huagiqin.com

**NO PUBLIC DISCLOSURE PERMITTED:** Please report postings of this document on public servers or websites to: [DocCtrlAgent@qualcomm.com](mailto:DocCtrlAgent@qualcomm.com).

Not to be used, copied, reproduced, or modified in whole or in part, nor its contents revealed in any manner to others without the express written permission of Qualcomm Technologies, Inc.

Qualcomm is a trademark of Qualcomm Incorporated, registered in the United States and other countries. Other product and brand names may be trademarks or registered trademarks of their respective owners.

This technical data may be subject to U.S. and international export, re-export, or transfer (“export”) laws. Diversion contrary to U.S. and international law is strictly prohibited.

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

© 2014 - 2016 Qualcomm Technologies, Inc. and/or its affiliated companies. All rights reserved.

# Revision History

---

Revision	Date	Description
A	July 2014	Initial release
B	June 2015	Added cases study for GPU power profiling. Updated GPU Power States Diagram. Slides restructures and reorganizations.
C	July 2016	Added slide 39 for Android M/N and Adreno 5XX Updated GPU dtsi binding information in slide 13 Added thermal impact on GPU power level in slide 37

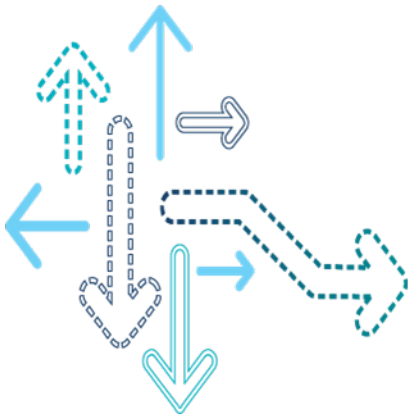
# Contents

---

- Adreno Software Power Management Overview
- Adreno Software Performance Overview
- Updates for Android M/N and Adreno 5XX
- Android GFX Performance Analysis with Systrace (Hands-On Session)
- Advanced Systrace Analysis Techniques for GFX Performance Issues
- Logging/Debugging
- Case Study: GPU Power Profiling
- References
- Questions?

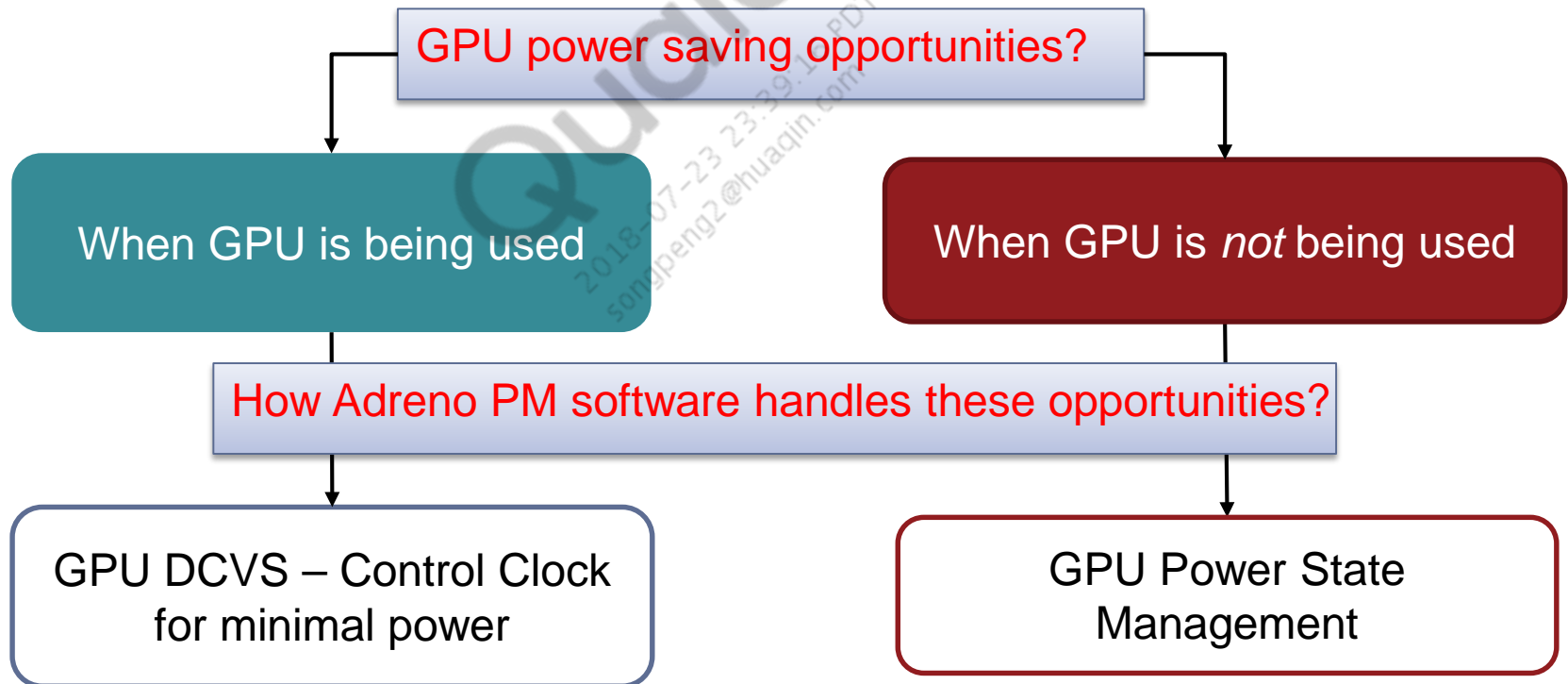
Qualcomm  
2018-07-23 23:39:16 PDT  
ongpeng2@huawei.com

# Adreno Software Power Management Overview



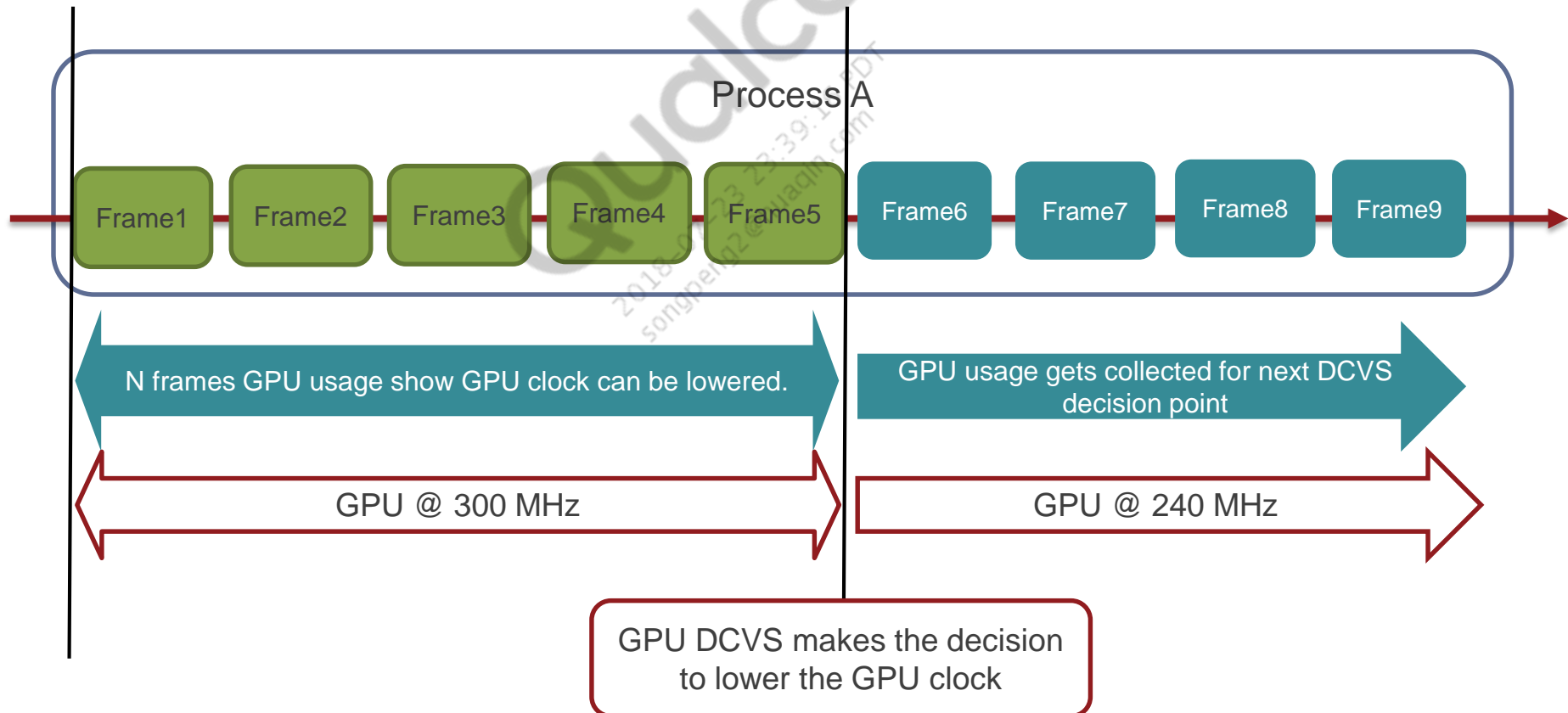
# Adreno Power Management Software Overview

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

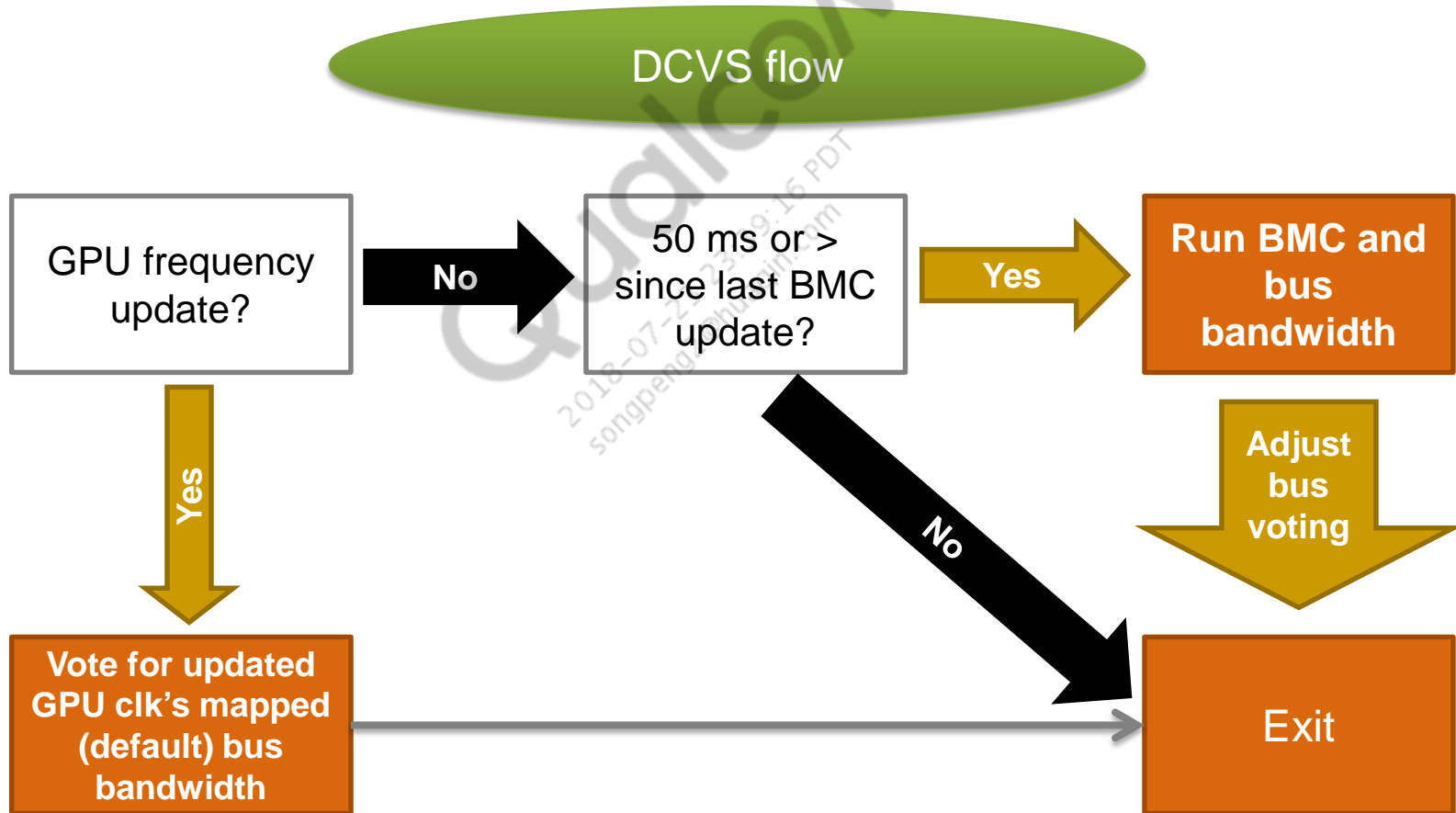


# Adreno Power Management Software Overview (cont.)

The BIG picture — When the GPU is being used, GPU DCVS controls the power



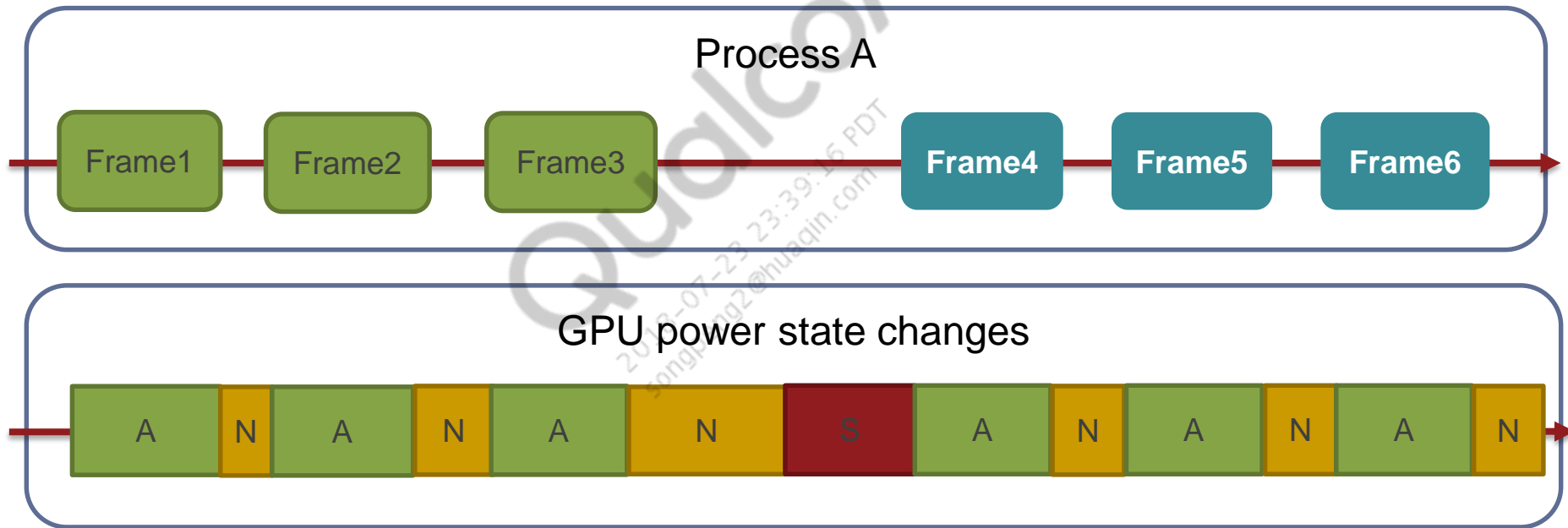
# Adreno Power Management Software Overview (cont.)








# Adreno Power Management Software Overview (cont.)

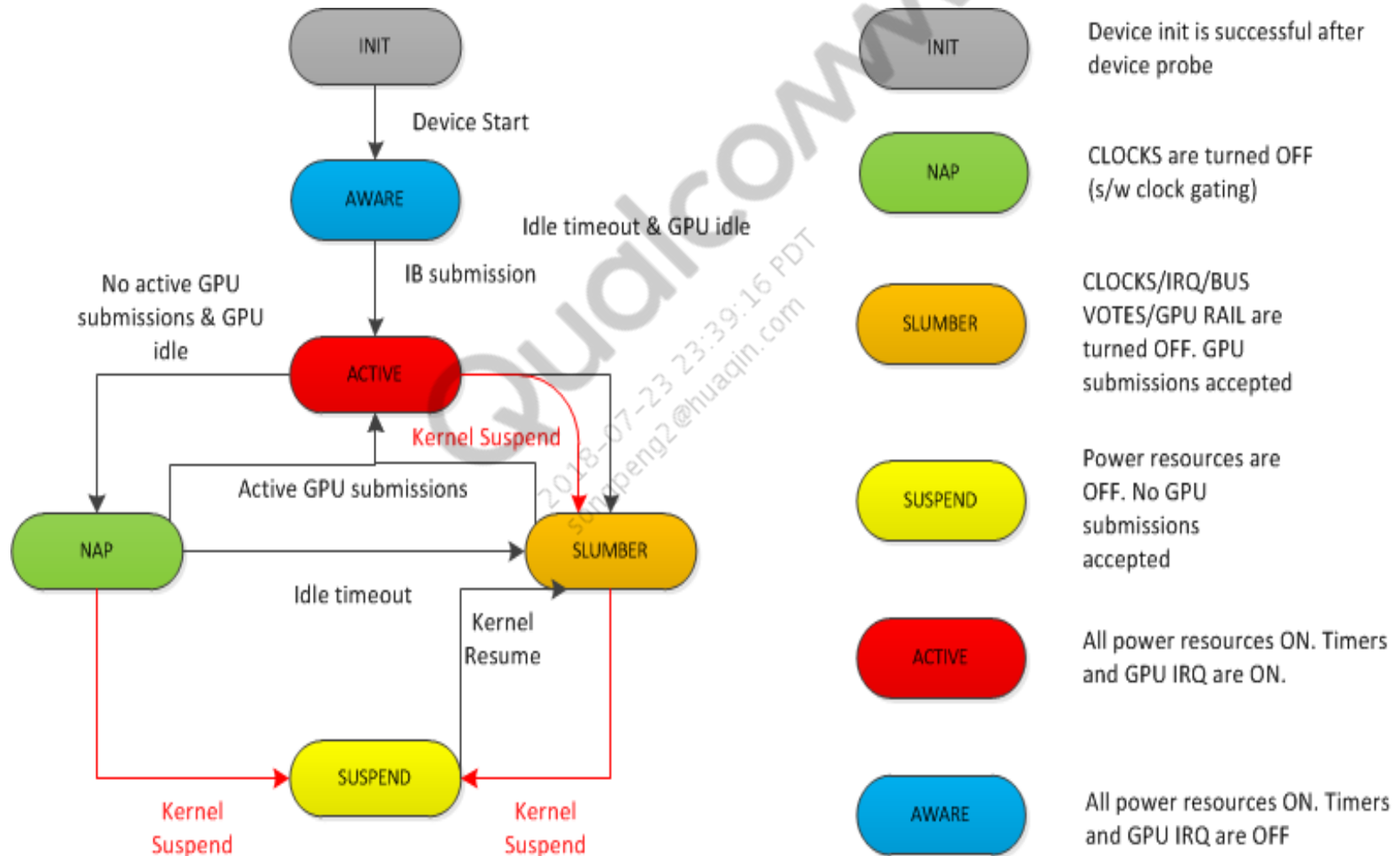
When the GPU is not being used, the GPU power state management kicks in



## Legend

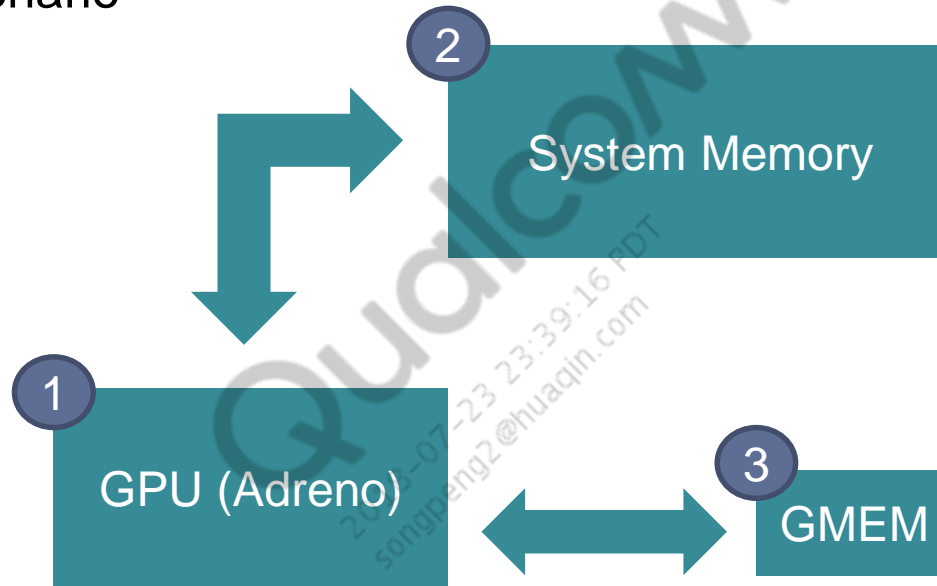
-  Active state
-  Nap state (NAP or DEEP\_NAP)
-  Sleep/Slumber state

# Adreno Power Management Software Overview (cont.)



# Adreno Power Management Software Overview (cont.)

- 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, are 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 (cont.)

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/ (32-bit build) and  
/kernel/arch/arm64/boot/dts/qcom/ (64-bit build)  
For example:
  - /kernel/arch/arm/boot/dts/qcom/msm8996-gpu.dtsi for MSM8996 (32-bit)
  - /kernel/arch/arm64/boot/dts/qcom/msm8996-gpu.dtsi for MSM8996 (64-bit)
- With different chip revision, the bindings are often overridden per chip revision. For example, MSM8996 V3's GPU bindings are overridden in /kernel/arch/arm64/boot/dts/qcom/msm8996-v3.dtsi.
- 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

---

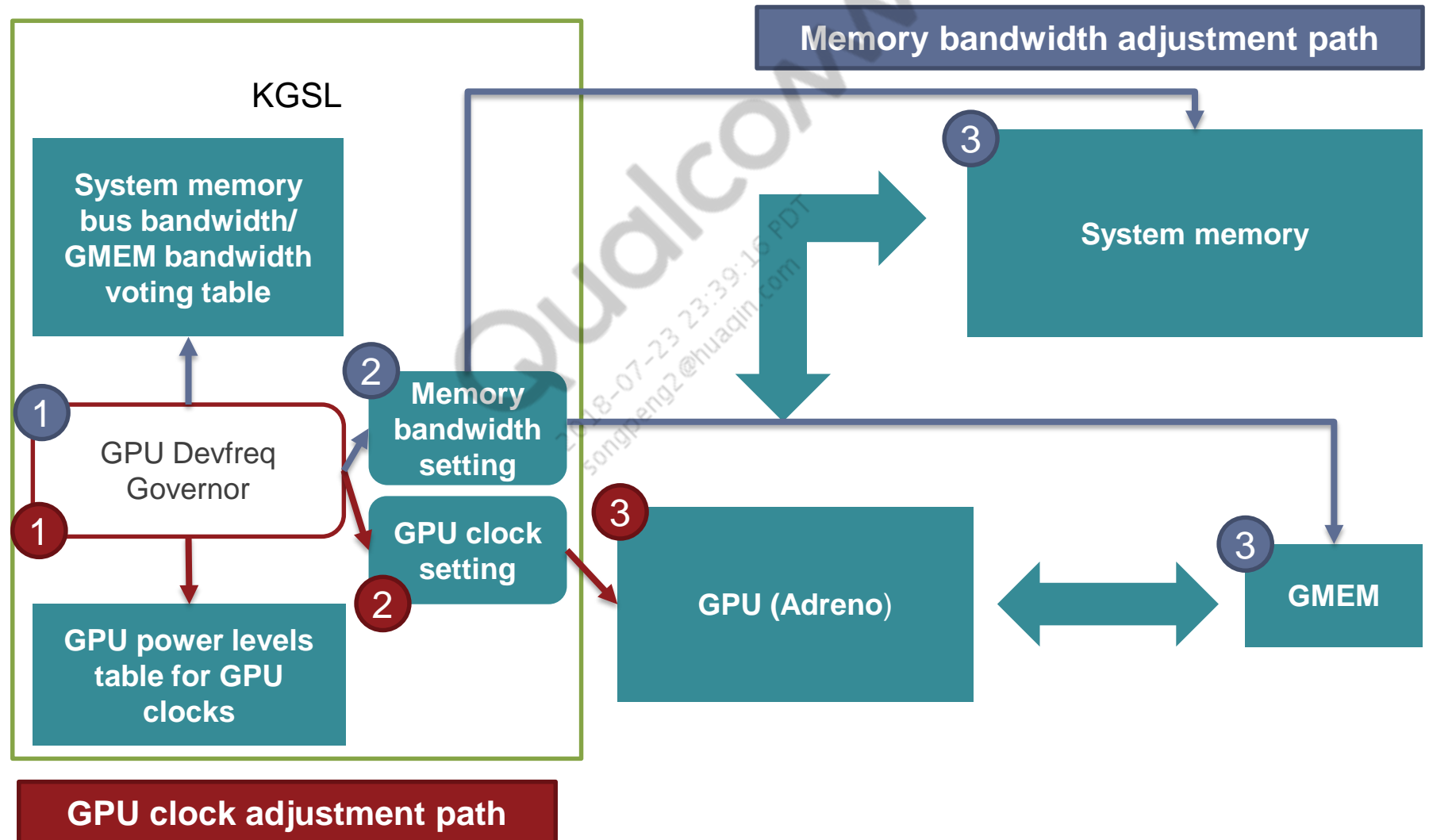
- Provides a generic framework to manage a non-CPU device's power usage based on a governor strategy used with the CPUFreq framework
  - For implementation details, go to <http://lwn.net/Articles/445044/>
- The QTI-proprietary Adreno powerscale-based governor has been adapted to the Devfreq framework
  - 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

---

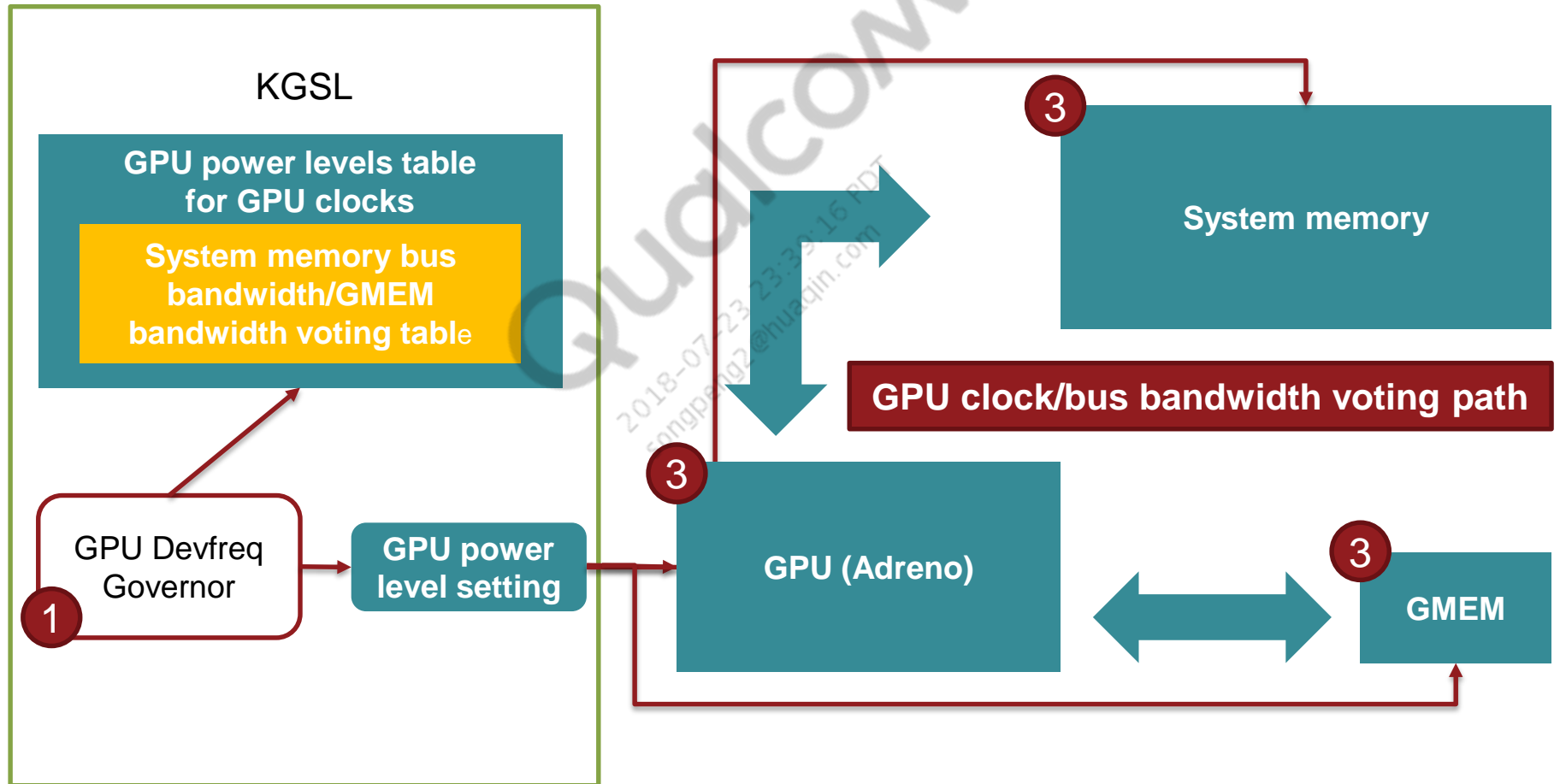
- GPU DCVS has been used as an umbrella term for Adreno's GPU clock and bus bandwidth voting mechanism
- Depending on each chipset's clock-level configurations, GPU Bus DCVS may or may not be available
  - For low-tier chipsets, GPU DCVS takes care of GPU bus bandwidth voting statically mapped to each 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



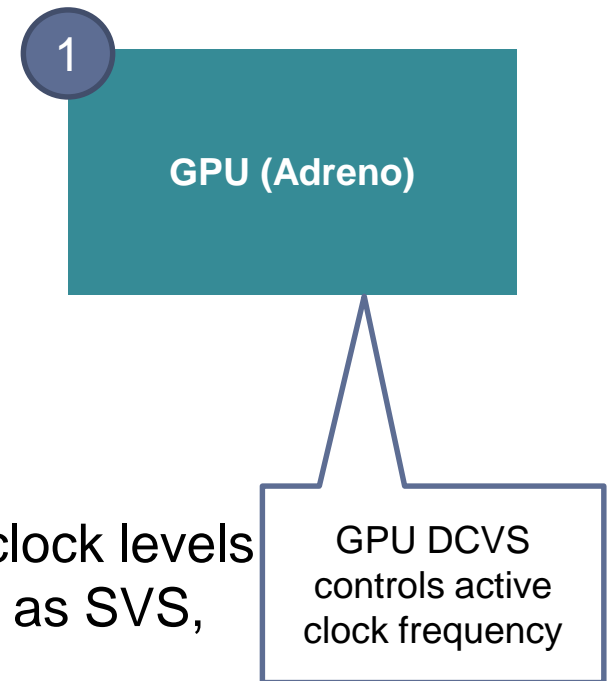


# KGSL GPU Governor Power Management with GPU DCVS Only



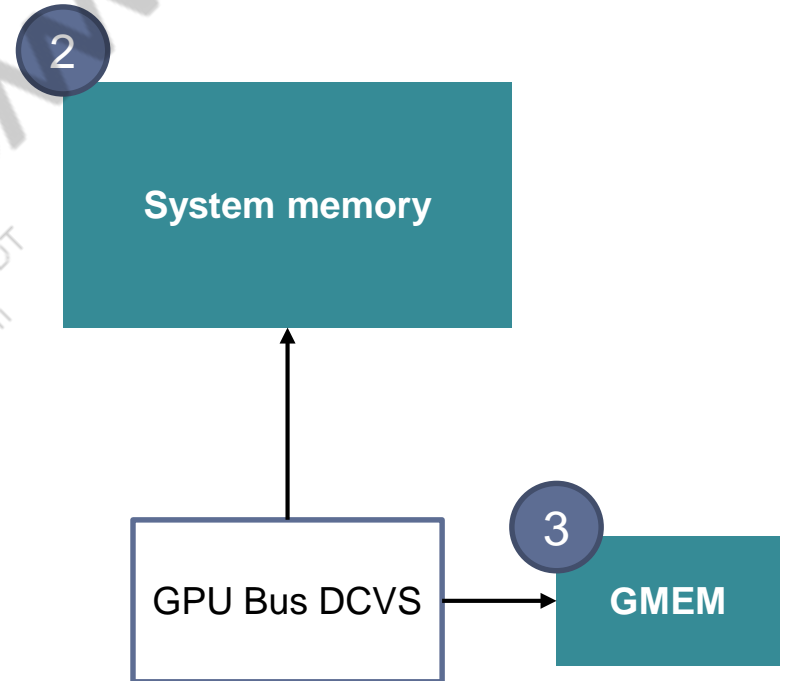
# GPU DCVS

- 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

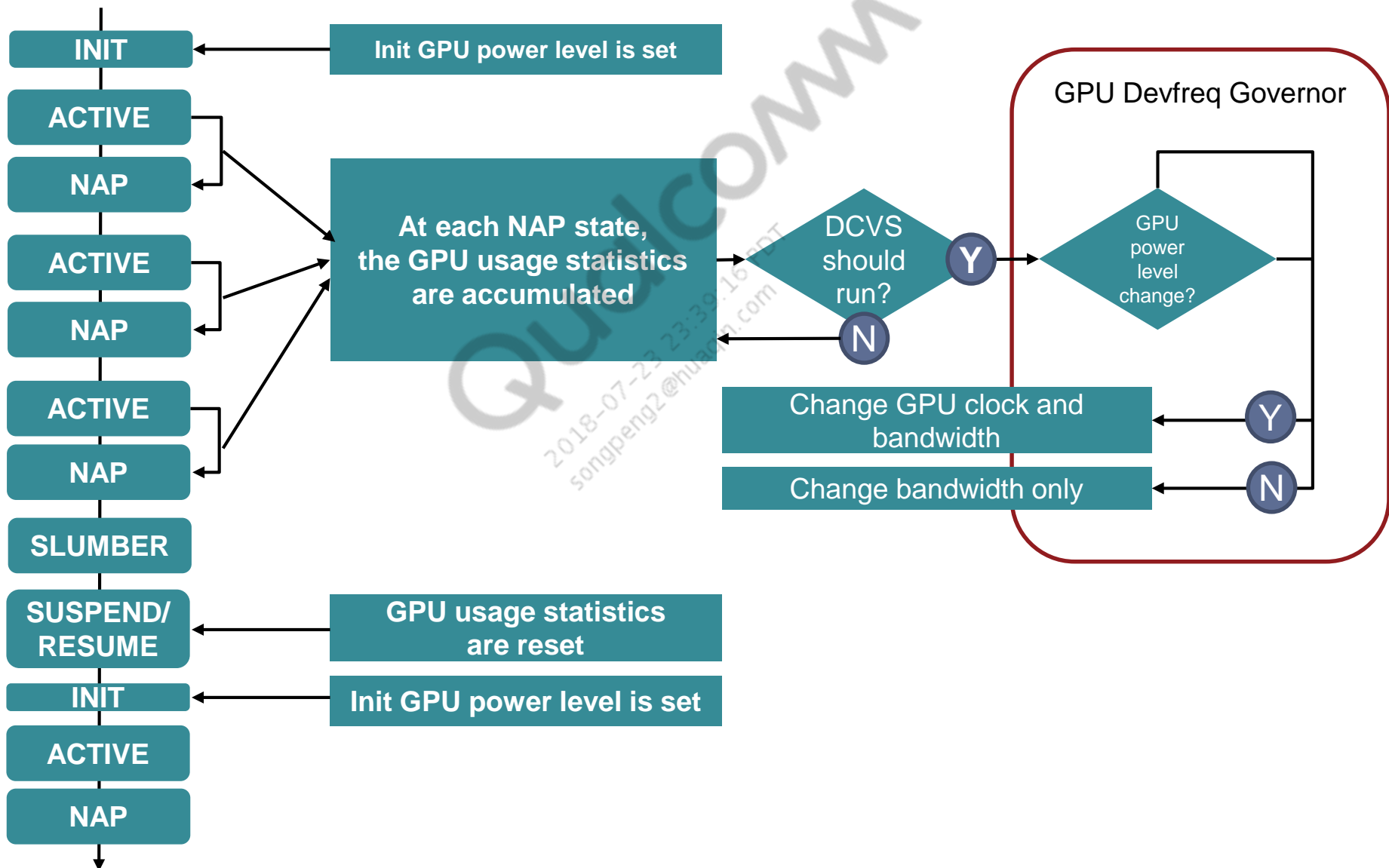


# GPU Bus DCVS

- 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

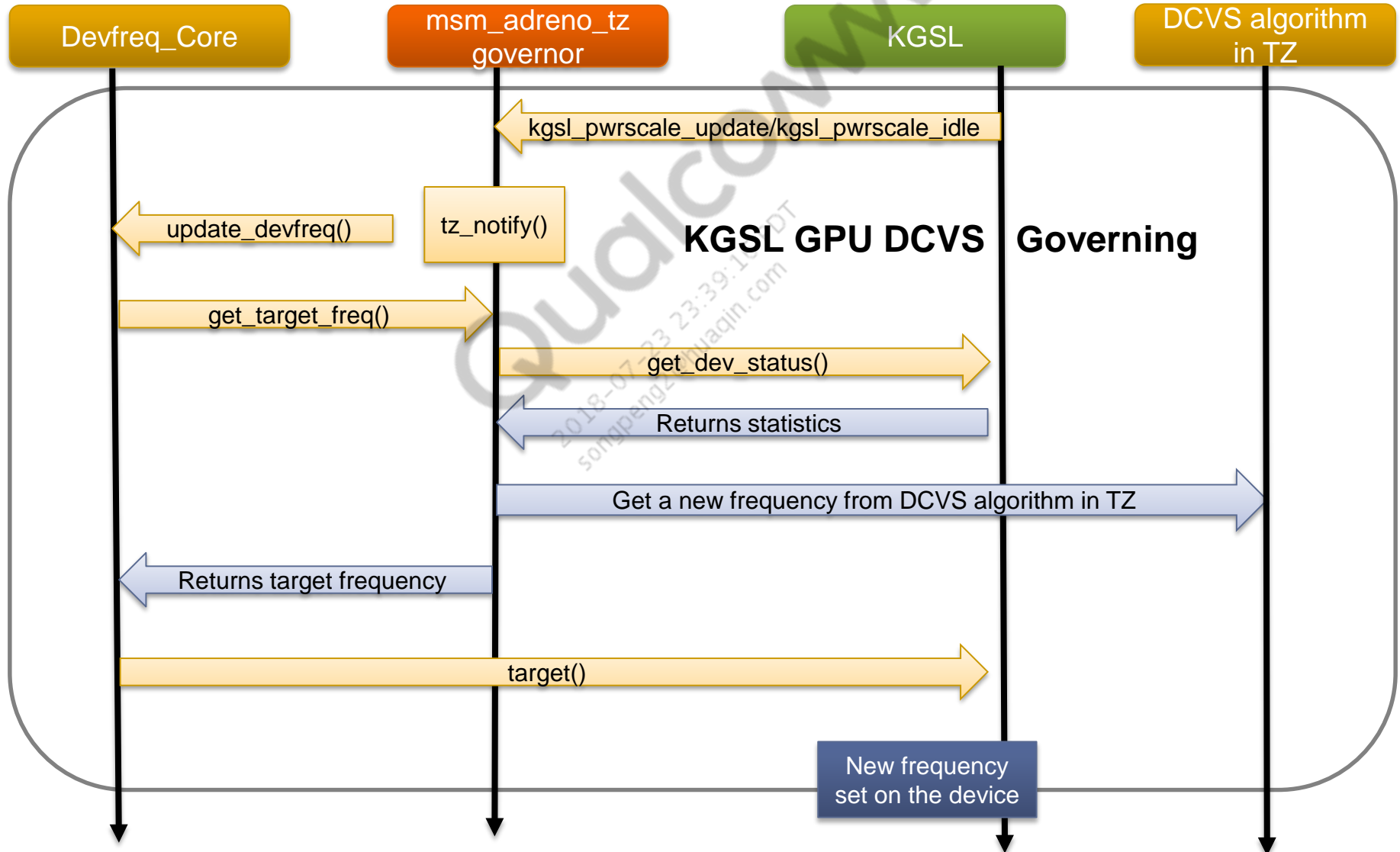


# msm-governor-tz – QTI Default Governor for GPU DCVS, GPU Load-Based

---

- The KGSL driver communicates with the governor directly at moments that are important for the driver.\*
  - KGSL driver uses Linux notifier mechanism to call the governor's `tz_notify()` function
  - `tz_notify ()` just calls `update_devfreq()`
  - `update_devfreq()` calls the governor's `get_target_freq()` function
  - `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
  - 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

# mzm-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

---

- 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 (cont.)

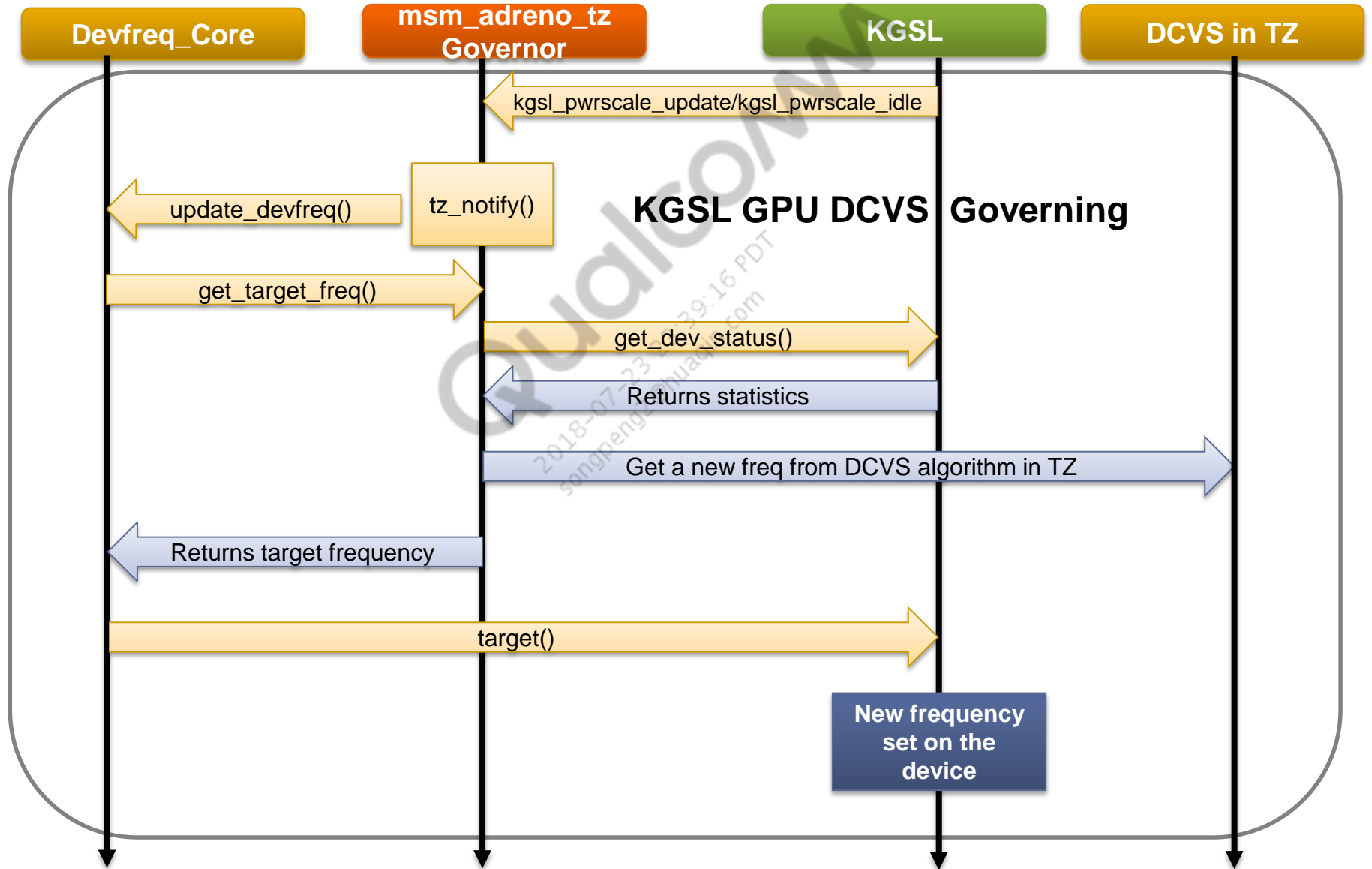
- Devfreq-enabled governor (Governor) exposes two callback functions

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 (cont.)



## Devfreq GPU DCVS Governor (cont.)

---

- 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
  - KGSL driver uses Linux notifiers mechanism to call the governor's tz\_notify() function
  - tz\_notify() just calls update\_devfreq()
  - update\_devfreq() calls the governor's get\_target\_freq() function
  - 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
  - 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

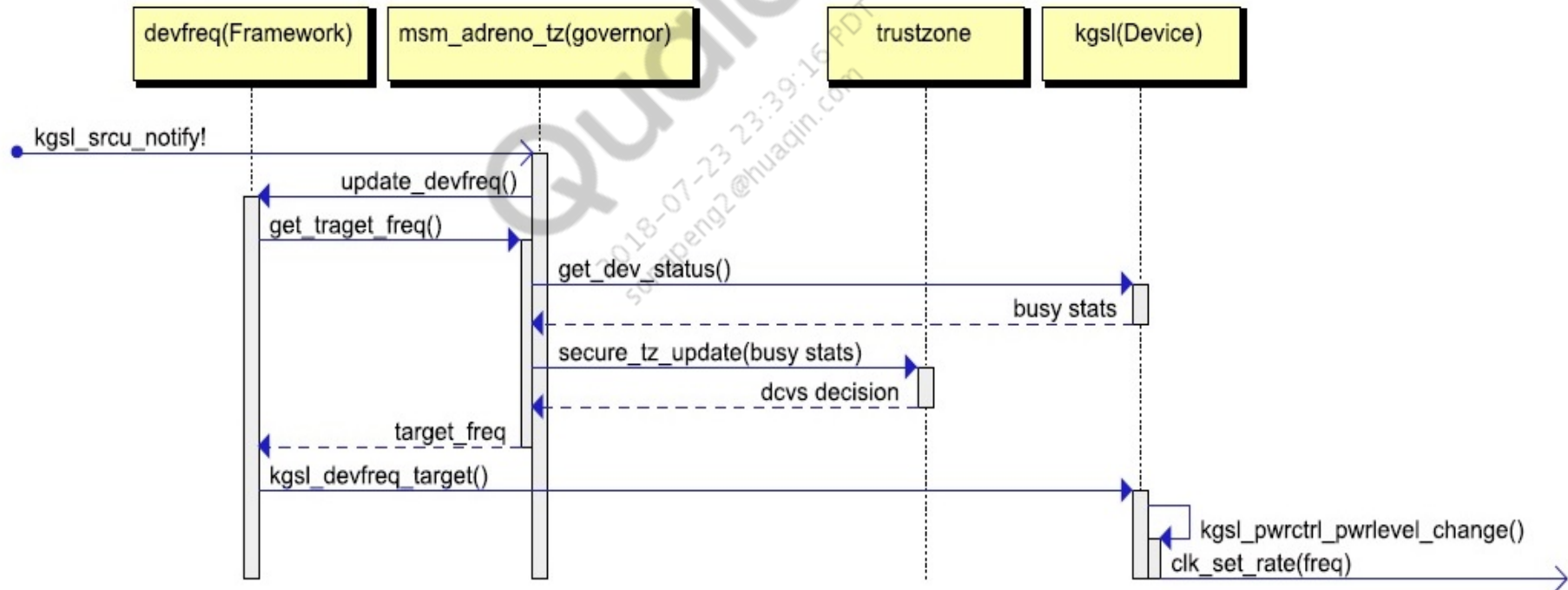
=====

kgsl(kgsl\_pwrscale.c)

### GOVERNOR

=====

msm-adreno-tz (governor\_msm\_adreno\_tz.c)



# GPU Bus DCVS

---

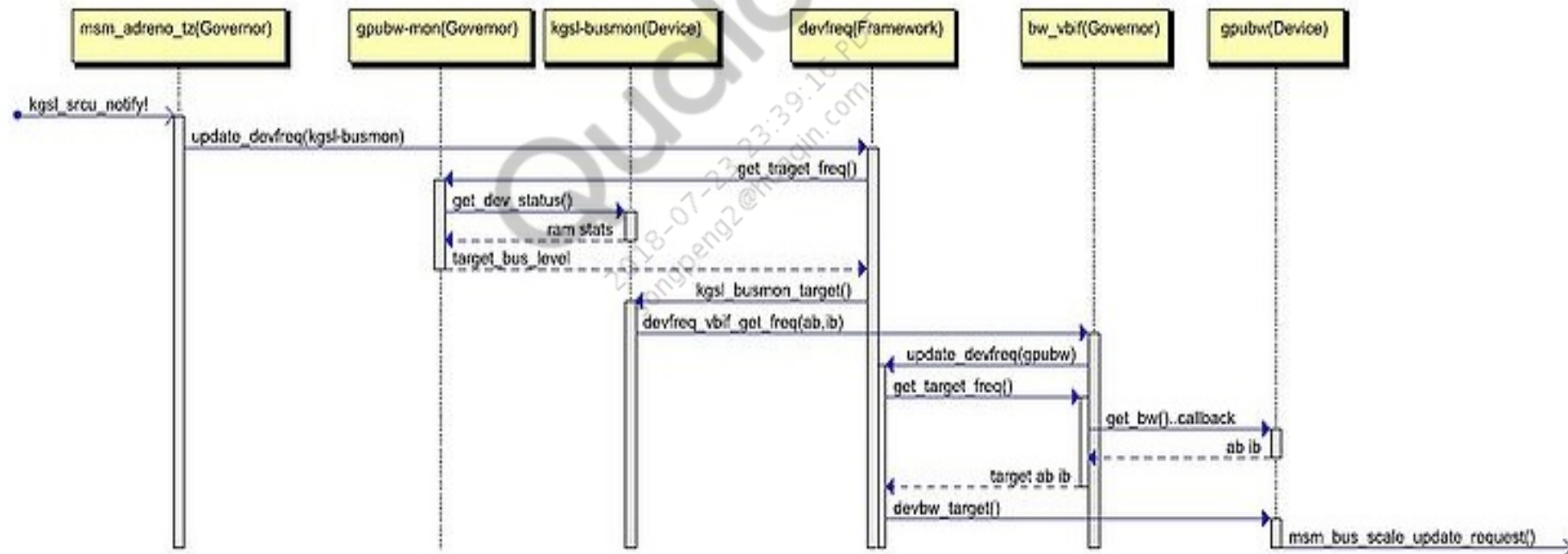
- GPU DCVS vs. GPU bus DCVS
  - GPU DCVS has been used as an umbrella term for the Adreno GPU clock and bus bandwidth voting mechanism
  - Depending on each chipset's clock-level configuration, GPU bus DCVS may or may not be available
    - For low-tier chipsets, GPU DCVS takes care of GPU bus bandwidth 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.

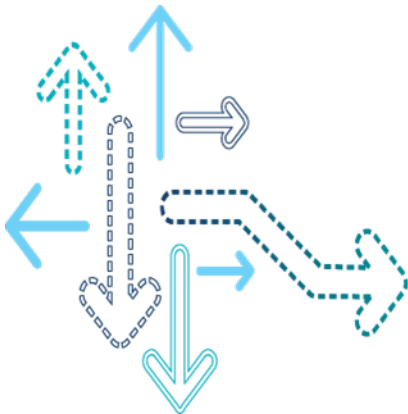
To make bus calculations and vote for bus we use three devices and their respective governor.

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)



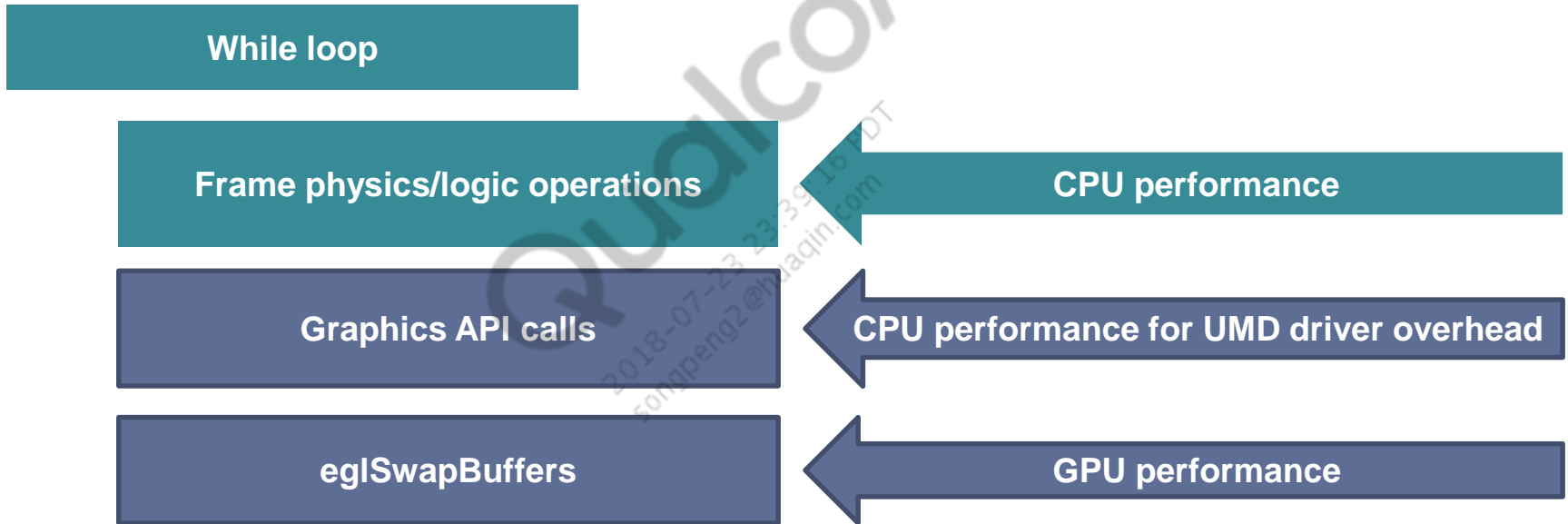
Qualcomm  
2018-07-23 23:39:16 PDT  
songpeng2@huawei.com

## Adreno Software Performance Overview



# Android Graphics Performance Analysis Overview

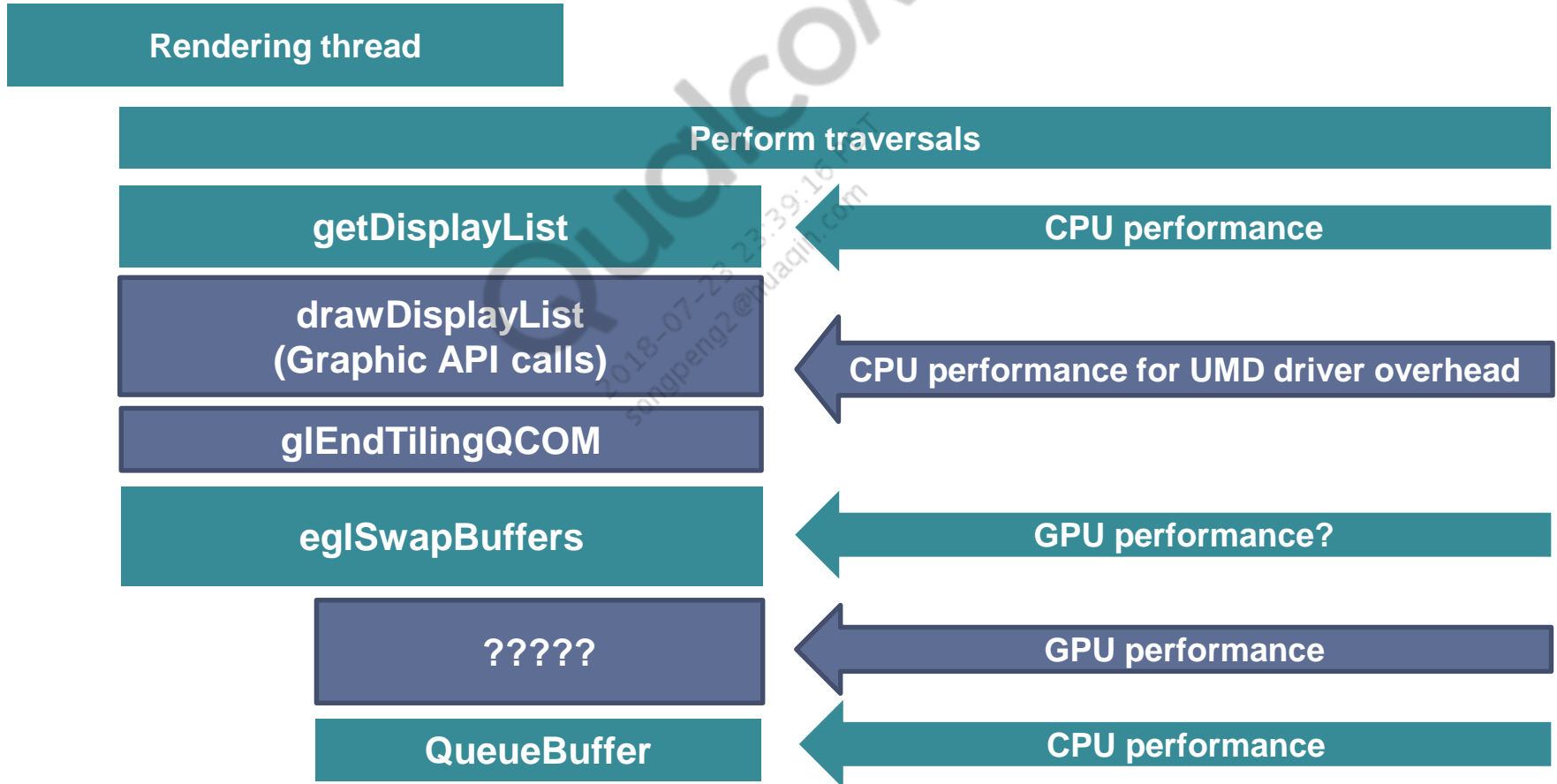
Typical frame rendering and performance points in an OpenGL ES application





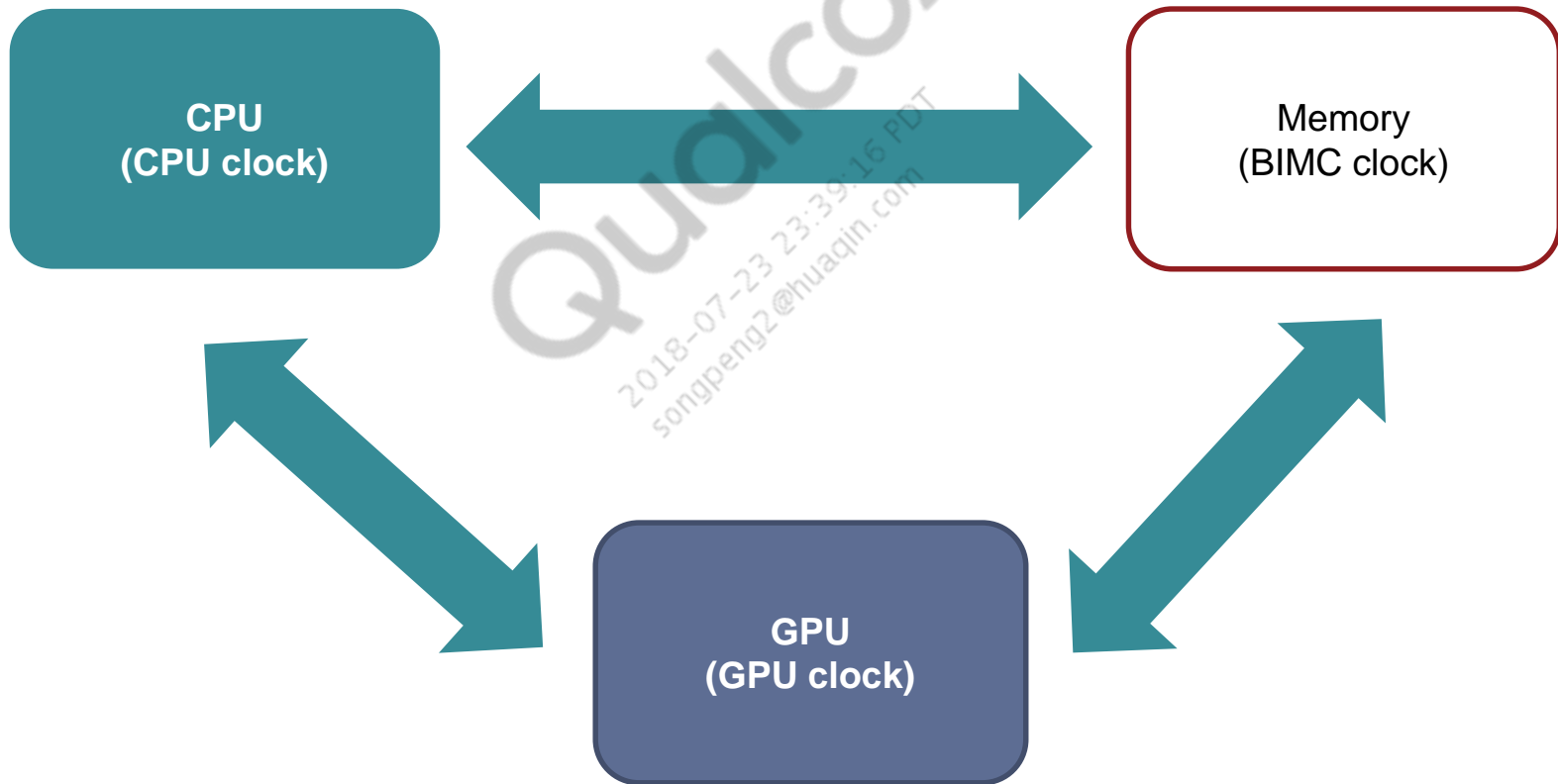
# Android Graphics Performance Analysis Overview (cont.)

## Android HWUI application



# Android Graphics Performance Analysis Overview (cont.)

Hardware blocks associated to overall picture



# Android Graphics Performance Analysis Overview (cont.)

---

- 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 (cont.)

---

- 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

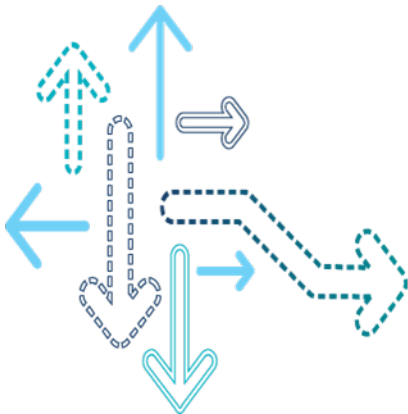
# Thermal Mitigation Impact on GPU

---

- If an OEM chooses to throttle GPU clock as a part of thermal mitigation strategy, GPU performance could be impacted
- Thermal mitigation's occurrence can be checked by reading the thermal\_pwrlevel node
  - adb shell cat /sys/class/kgsl/kgsl-3d0/thermal\_pwrlevel
  - Any non-zero value means GPU is being throttled for thermal mitigation
- If thermal\_pwrlevel is set to non-zero value, GPU's max\_pwrlevel will have lower priority than thermal\_pwrlevel
  - Even in GPU performance mode, GPU will run at thermally capped GPU clock
- For quickly check the impact of GPU's thermal throttling, disabling thermal engine can help
  - adb shell stop thermal-engine

Qualcomm  
2018-07-23 23:39:16 PDT  
congpeng2@huawei.com

## Updates for Android M/N and Adreno 5XX

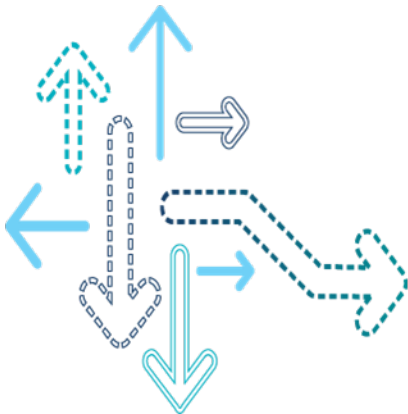


# Updates for Android M/N and Adreno 5XX

---

- Intermediate power state DEEP\_NAP state was added to KGSL power states
  - GPU enters DEEP\_NAP state from NAP state (after spending 20ms in NAP state)
  - In DEEP\_NAP state,
    - GPU retention clock gets turned off
    - KGSL makes pm\_qos request to power manager for system power management
  - GPU can switch to ACTIVE state from either NAP or DEEP\_NAP
- Android N (as of this document's publication) doesn't have 'Enable OpenGL ES trace' option. Future of this option will be verified once N release makes public release
- Systrace UI on Android M changed as well as information pane's locations; however, the debugging method using Systrace is still valid
- A5XX GPUs have dedicated GMEM inside GPU, not on OCMEM

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





# Android GFX Performance Analysis with Systrace

---

- 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\_sdk\_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 (cont.)

The screenshot shows the Android Debug Monitor interface. On the left, the 'Devices' tab is active, showing a list of processes. A red circle highlights the Systrace icon (a small icon with three vertical bars) in the top toolbar, with a callout '1' pointing to it. The 'Android System Trace' dialog is open in the center, titled 'Settings to use while capturing system level trace'. It has a 'Destination File' field set to 'C:\Users\pdauid\trace.html' with a 'Browse...' button. Below this are 'Trace duration (seconds):' and 'Trace Buffer Size (MB):' fields. A 'Select tags to enable:' section contains a list of system components, all of which are checked. At the bottom of the dialog are 'OK' and 'Cancel' buttons. Callout '2' points to the 'Destination File' field, and callout '3' points to the list of checked tags. Callout '4' points to the 'OK' button. A large teal callout box with callout '1' contains the text: 'Click the icon, and Android Systrace Dialog appears'. Another teal callout box with callout '2' contains: 'Enter filename; trace file is captured here'. A third teal callout box with callout '3' contains: 'Check **all** the options'. A fourth teal callout box with callout '4' contains: 'Click OK to capture and run the use case'. A fifth teal callout box with callout '5' contains: 'Default is 5 sec; enter longer time in seconds if needed \* \* Depending on your PC system, longer time might not work'.

1 Click the icon, and Android Systrace Dialog appears

2 Enter filename; trace file is captured here

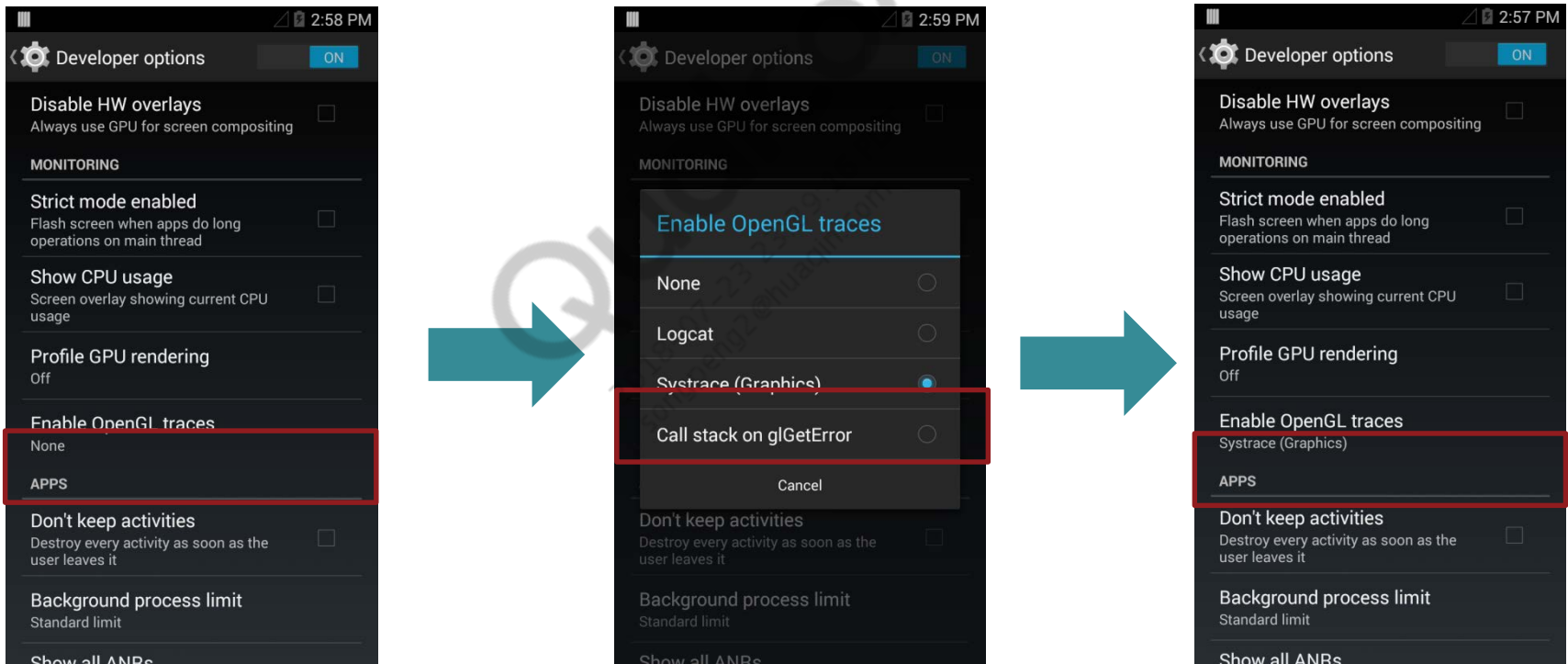
3 Check **all** the options

4 Click OK to capture and run the use case

5 Default is 5 sec; enter longer time in seconds if needed \*  
\* Depending on your PC system, longer time might not work

# Android GFX Performance Analysis with Systrace (cont.)

- 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:

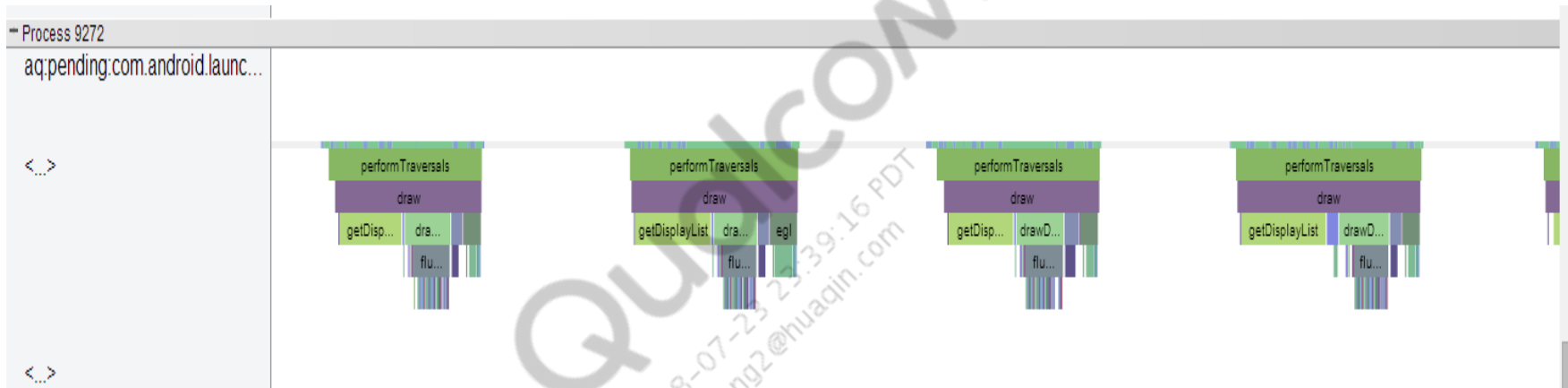


- Finally, do adb shell stop and start:

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

# Android GFX Performance Analysis with Systrace (cont.)

- Sample Output (App menu scroll)

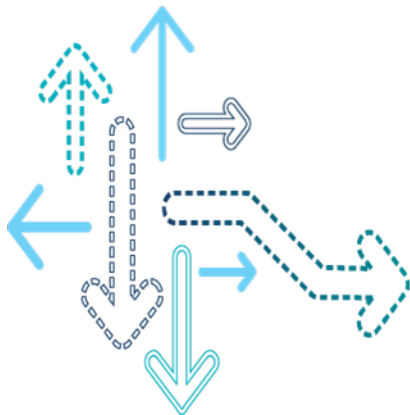


- Now it is time for hands-on and extensive analysis!

Qualcomm

2018-07-23 23:39:16 PDT  
songpeng2@huawei.com

## Advanced Systrace Analysis Techniques for GFX Performance Issues



# Advanced Systrace Analysis Techniques for GFX Performance Issues

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

The screenshot shows the Systrace API Call Log interface. At the top, there's a search bar with 'Categories' set to 'eglswapbuffers'. Below this, a table lists various API calls with their durations. The 'Slices' section shows a summary for 'eglSwapBuffers' with a duration of 256.977 ms and 204 occurrences. A selection box highlights a specific slice, and a tooltip displays detailed statistics for that slice.

API Call	Duration (ms)	Occurrences
eglSwapBuffers	256.977	204

**Slices:**  
eglSwapBuffers 256.977 ms 204 occurrences  
Selection start "757.592 ms"  
Selection extent "3928.29 ms"

**Slices:**  
eglSwapBuffers 256.977 ms 204 occurrences  
Selection start "757.592 ms"  
Selection extent "3928.29 ms"

**Selected slice:**  
Min Duration: 0.464 ms  
Max Duration: 7.51 ms  
Avg Duration: 1.26 ms ( $\sigma = 0.765$ )  
Frequency: 51.699 occurrences/s ( $\sigma = 21.924$ )

1

On top right corner, type the method name

2

On the bottom pane, 'Slices' data is shown

3

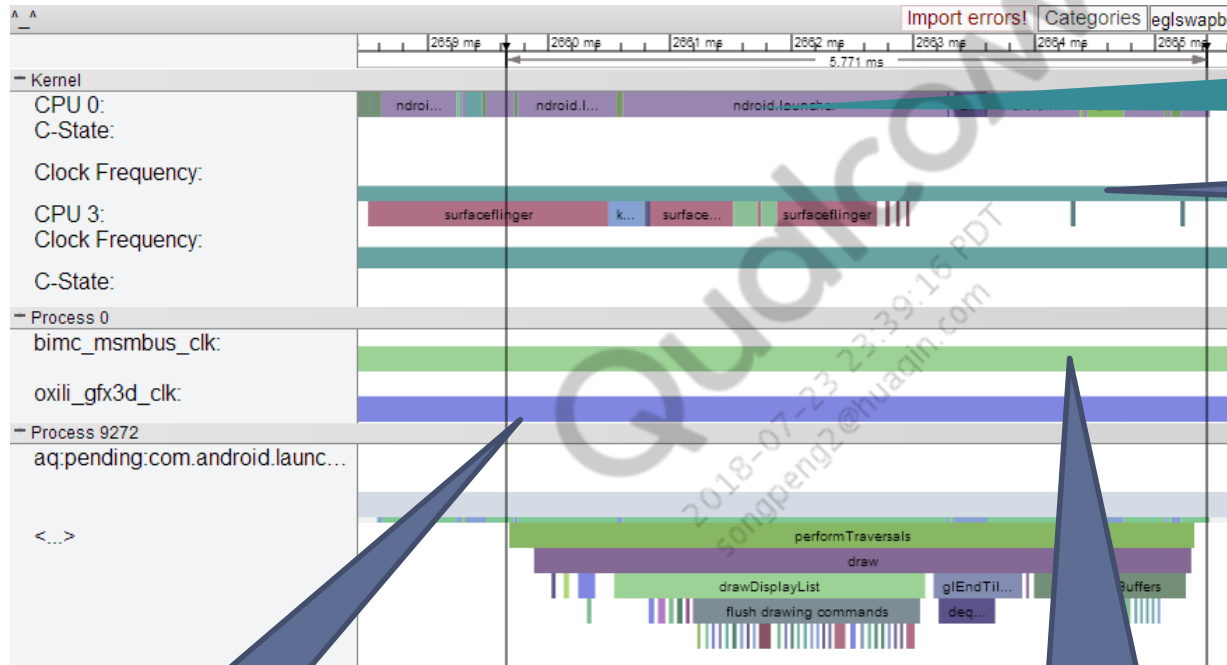
Mouse over on the method name

4

Min/Max/AVG durations are shown

# Advanced Systrace Analysis Techniques for GFX Performance Issues (cont.)

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



Launcher is running on CPU0

Click Clock Freq for CPU0

Selected counter:  
Title "Clock Frequency"  
Timestamp "2617.801 ms"  
state 883200

CPU0 running @ 883 MHz

Click oxili\_gfx3d\_clk

Selected counter:  
Title "oxili\_gfx3d\_clk"  
Timestamp "2509.154 ms"  
value 200000000

GPU running @ 200 MHz

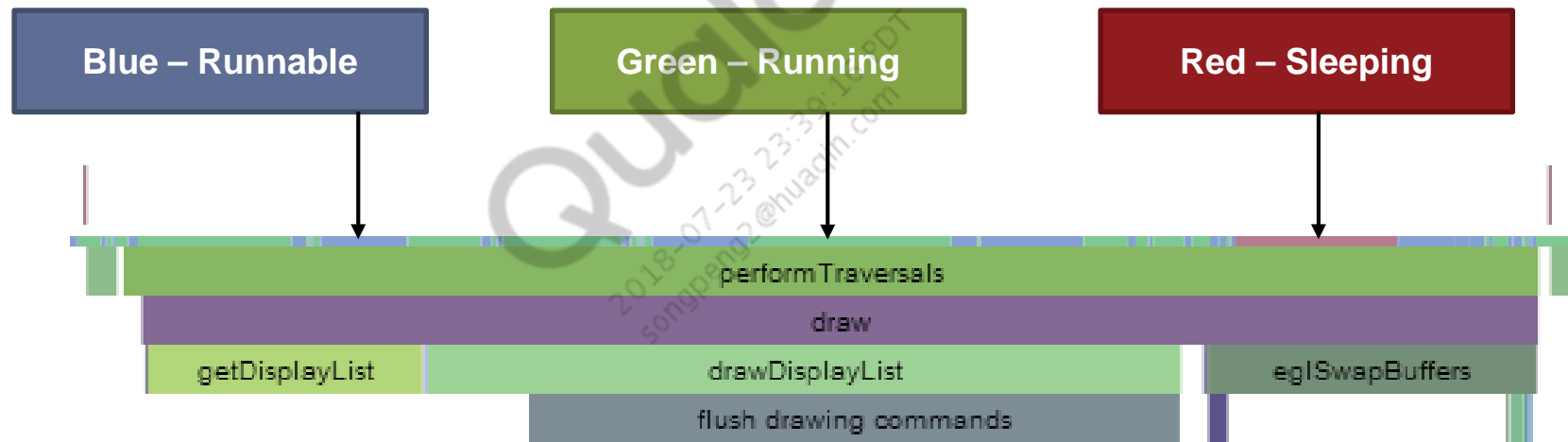
Click bimc\_msmbus\_clk

Selected counter:  
Title "bimc\_msmbus\_clk"  
Timestamp "2532.791 ms"  
value 307000000

BIMC running @ 307 MHz

# Advanced Systrace Analysis Techniques for GFX Performance Issues (cont.)

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



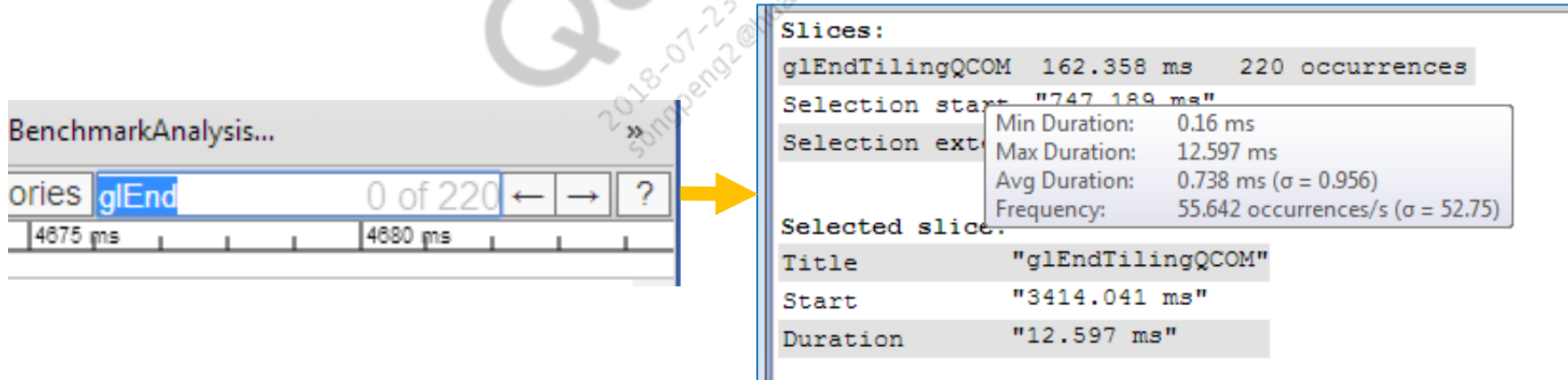
- To find the actual block time, we must subtract Runnable/Sleeping times, only counting Running time



# Advanced Systrace Analysis Techniques for GFX Performance Issues (cont.)

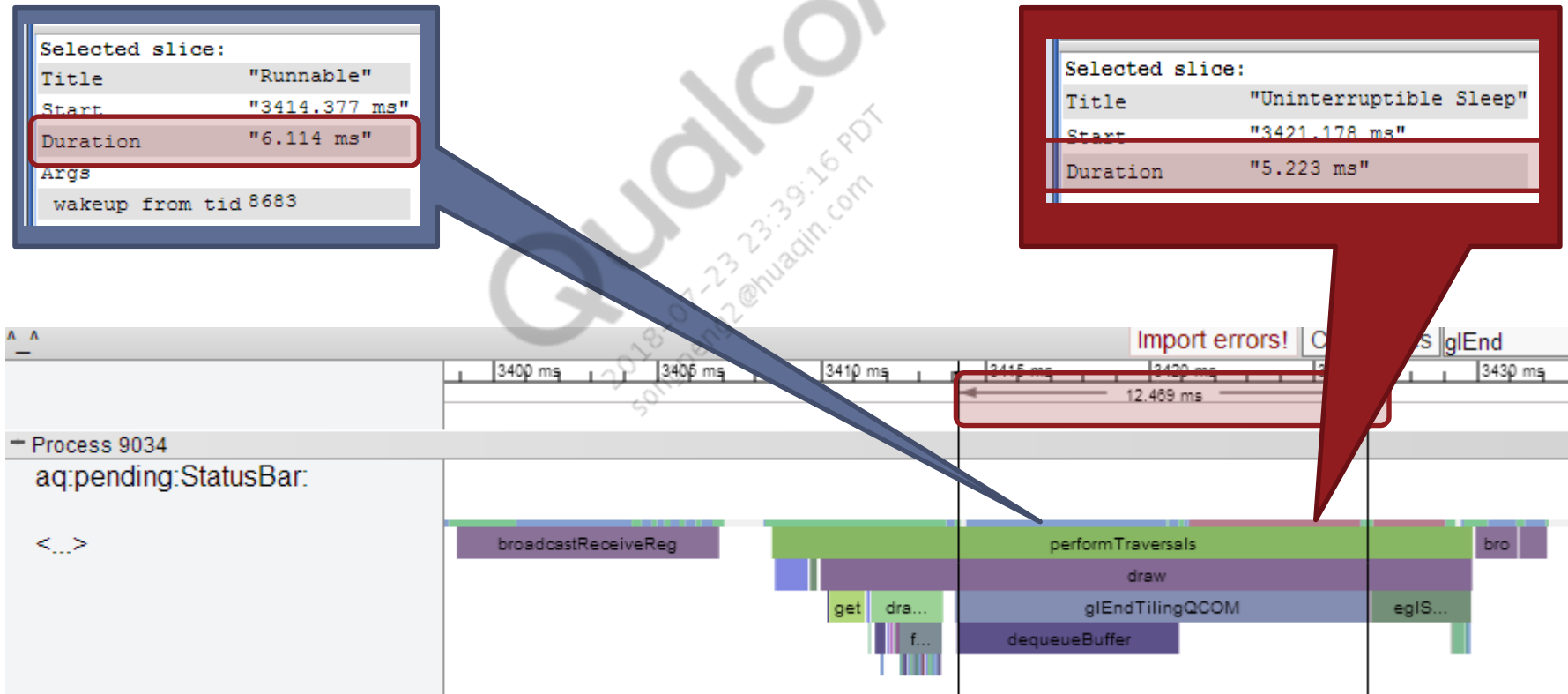
Put all three techniques together to analyze one example

- With one Systrace capture, one glEndTilingQCOM taking >12 ms with Status Bar rendering is found, system.ui process (Technique 1)



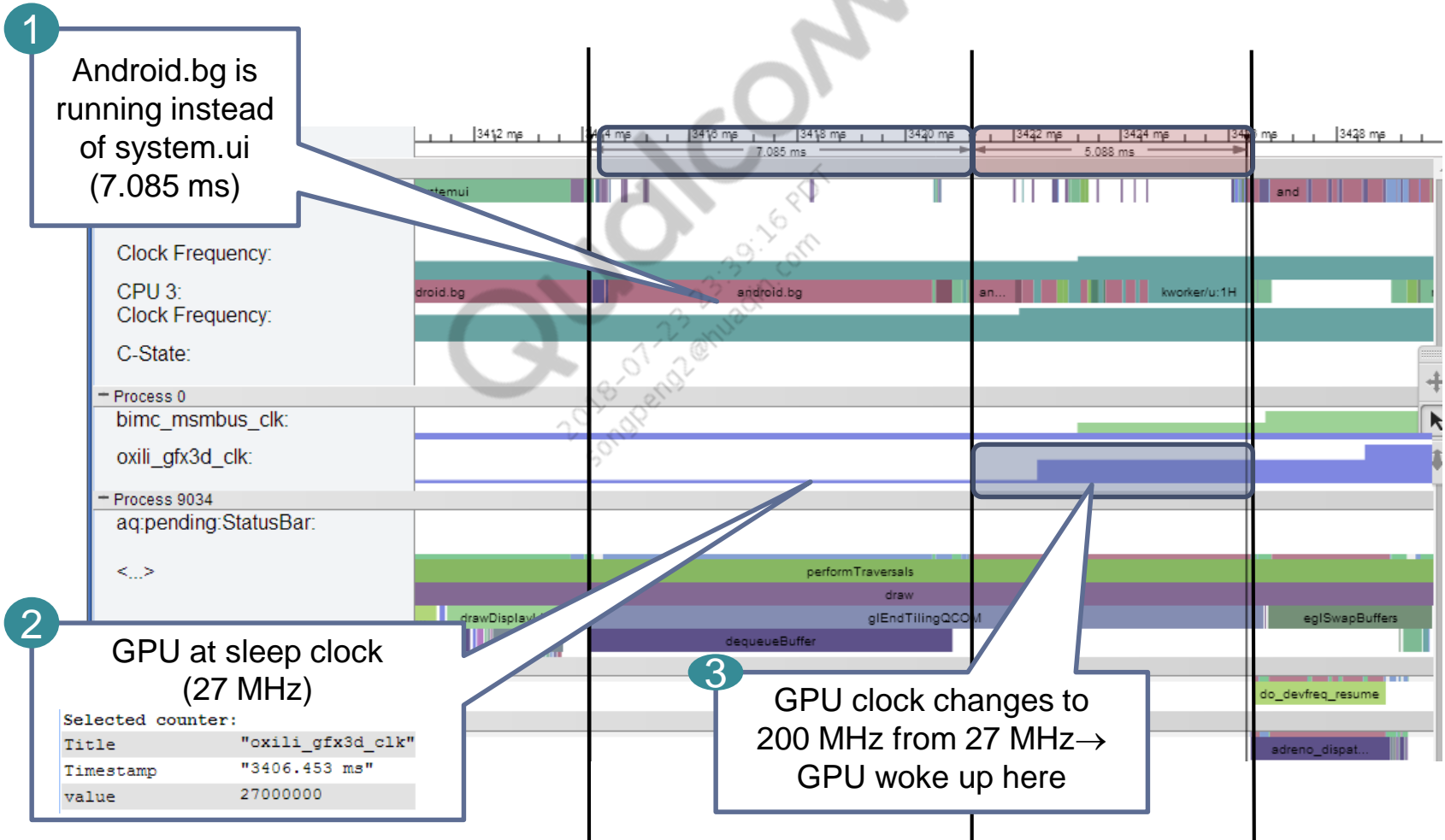
# Advanced Systrace Analysis Techniques for GFX Performance Issues (cont.)

- But for more than 90% of the 12 ms, glEndTilingQCOM is either in the Runnable (not actually running) or Uninterruptible Sleep state (Technique 3).



# Advanced Systrace Analysis Techniques for GFX Performance Issues (cont.)

- Check the clocks and CPU states (Technique 2)



# Advanced Systrace Analysis Techniques for GFX Performance Issues (cont.)

- Analysis conclusion?

1

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

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

2

GPU at sleep clock  
(27 MHz)

```
Selected counter:  
Title      "oxili_gfx3d_clk"  
Timestamp  "3406.453 ms"  
value      27000000
```

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

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

# Logging (cont.)

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 kgsi`

## 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:

- `kgsi:kgsi_a2xx_irq_status` — `kgsi:kgsi_pwrlevel`
- `kgsi:kgsi_a3xx_irq_status` — `kgsi:kgsi_buslevel`
- `kgsi:kgsi_clk` — `kgsi:kgsi_pwrstats`
- `kgsi:kgsi_irq` — `kgsi:kgsi_pwr_set_state`
- `kgsi:kgsi_rail` — `kgsi:kgsi_pwr_request_state`
- `kgsi:kgsi_bus` — `kgsi:kgsi_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 (cont.)

---

- 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

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.

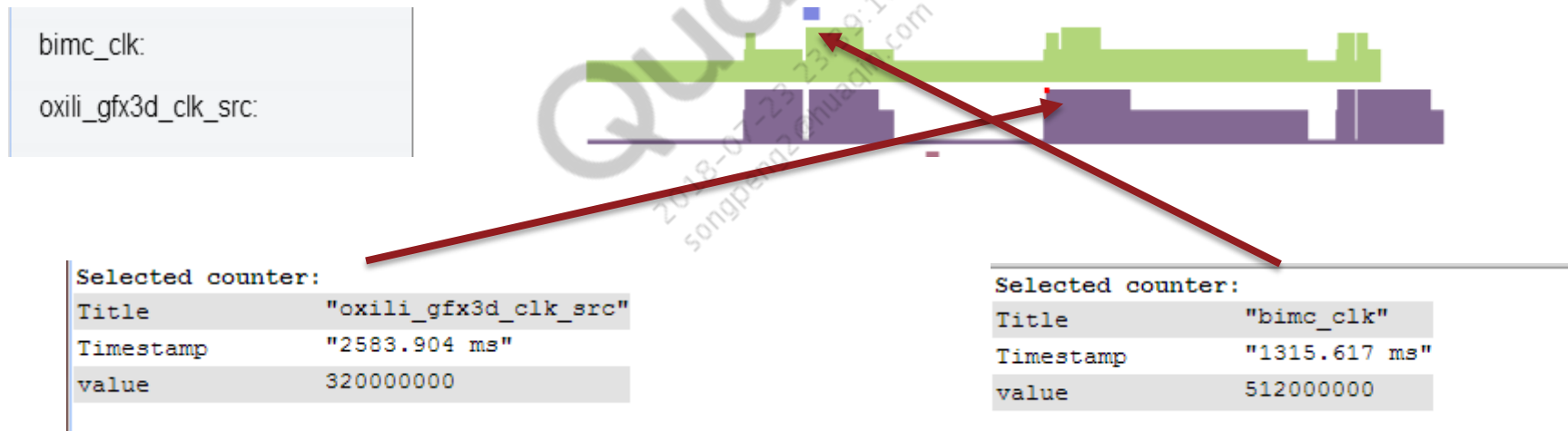
Sysfs operations	Description	Command sequences
Force the GPU clocks/bus vote/power rail always on	Forcing the GPU clocks/bus vote/power rail always on will prevent GPU from power collapsing.	<pre>cd /sys/class/kgsl/kgsl-3d0 echo 1 &gt; force_clk_on /* Clocks are not SW gated during standard rendering */ echo 0 &gt; force_clk_on /* Resume SW gating of clocks */ echo 1 &gt; force_rail_on /* Power rail never turned off */</pre>
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	<pre>cd /sys/class/kgsl/kgsl-3d0 echo 1000000 &gt; idle_timer</pre>
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	<pre>cd /sys/class/kgsl/kgsl-3d0/devfreq echo performance &gt; governor  cd /sys/class/kgsl/kgsl-3d0/devfreq echo powersave &gt; governor</pre>

# Debugging (cont.)

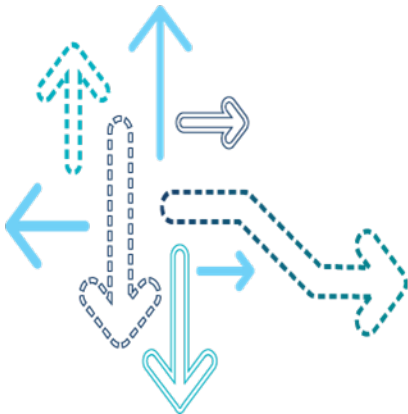
Sysfs operations	Description	Command sequences
Find the available GPU power levels through sysfs	To ensure you request a supported frequency check availability	<code>cat /sys/class/kgsl/kgsl-3d0/gpu_available_frequencies</code>
Set the min/max power level for GPU	To test GPU power levels power consumption or have a fixed GPU clock profiling	<code>cd /sys/class/kgsl/kgsl-3d0</code> <code>echo 1 &gt; max_pwrlevel</code> <code>echo 1 &gt; min_pwrlevel</code>  //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)	<code>cd /sys/class/kgsl/kgsl-3d0/</code> <code>cat gpubusy</code>  //(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	<code>cd /sys/kernel/debug/clk/oxili_gfx3d_clk/</code> <code>cat measure</code>
Checking GPU bus vote	To ensure GPU is voting at optimal bus bandwidth	<code>cd /sys/kernel/debug/msm-bus-dbg/client-data/</code> <code>cat grp3d</code>  //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 (cont.)

- 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]



## 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

- 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 (cont.)

---

- 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 (cont.)

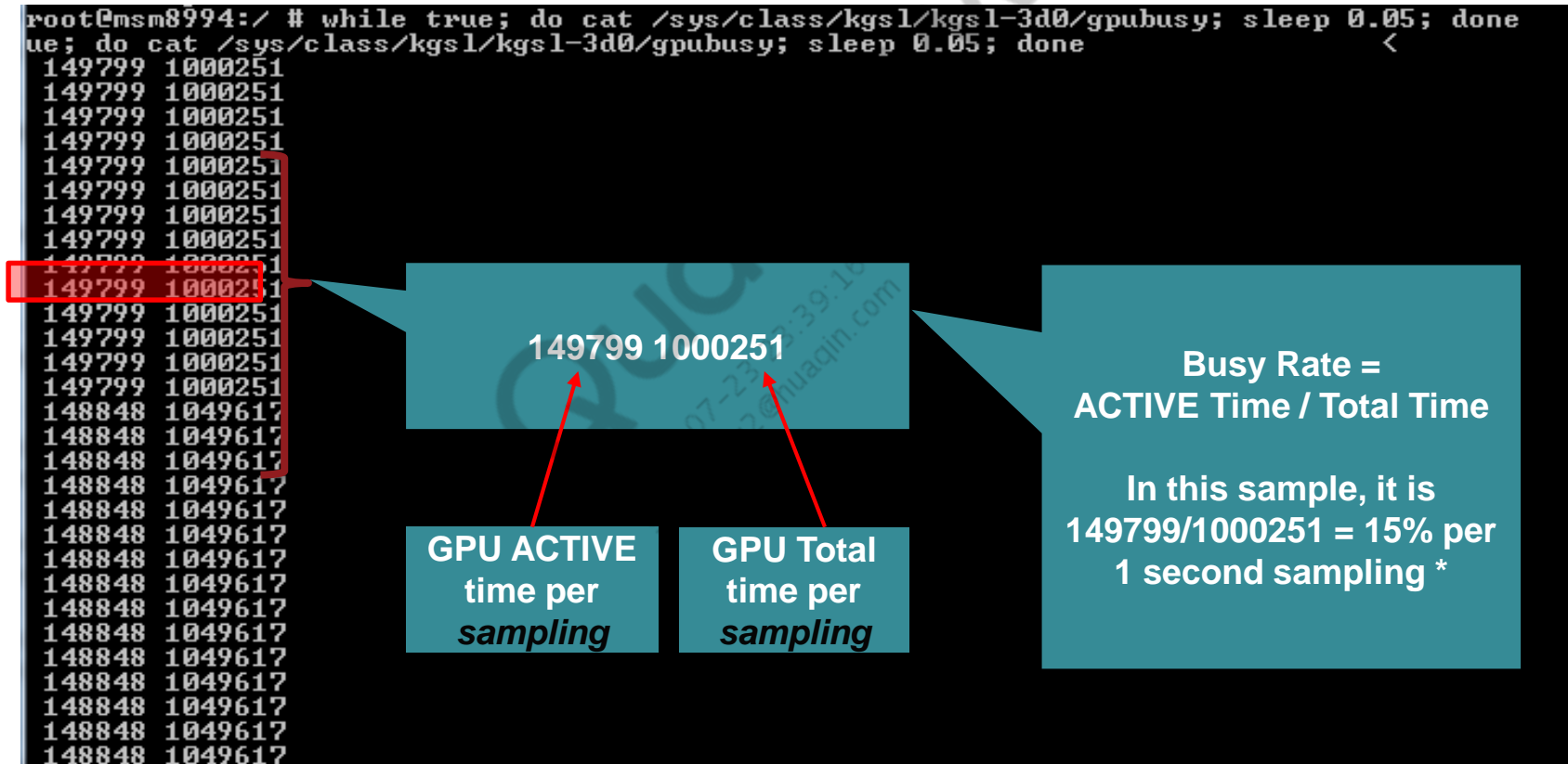
---

- 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 (cont.)

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



# GPU Power Profiling – GPU Clock During YouTube Full Screen Playback (cont.)

- 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 (cont.)

- 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=50014 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=337549 busy=6179 ram_time=539536 ram_wait=351885
kworker/u16:3-10302 [005] ...1 5139.644539: kgsl_pwrstats: d_name=kgsl-3d0 total=352827 busy=6179 ram_time=539536 ram_wait=351885
kworker/u16:3-10302 [007] ...1 5139.693438: kgsl_pwrstats: d_name=kgsl-3d0 total=331374 busy=6179 ram_time=539536 ram_wait=351885
kworker/u16:5-10397 [005] ...1 5139.724647: kgsl_pwrstats: d_name=kgsl-3d0 total=340278 busy=6179 ram_time=539536 ram_wait=351885
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 TOTAL**  
time per  
submission  
event

**GPU BUSY**  
time per  
submission  
event

# GPU Power Profiling – GPU Clock During YouTube Full Screen Playback (cont.)

- 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
        :6000000005100000004500000003900000003050000001800000000  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
```

Millisecond  
s 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 (cont.)

- 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 30 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

---

Title	Number
<b>Qualcomm Technologies, Inc.</b>	
<i>Application Note: Software Glossary for Customers</i>	CL93-V3077-1
<b>Resources</b>	
<i>Android Systrace</i>	<a href="http://developer.android.com/tools/help/systrace.html">http://developer.android.com/tools/help/systrace.html</a> <a href="http://developer.android.com/tools/debugging/systrace.html">http://developer.android.com/tools/debugging/systrace.html</a>

Qualcomm  
2018-07-23 23:16 PDT  
songpeng2@qualcomm.com

Qualcomm  
2018-07-23 23:39:16 PDT  
songpeng2@huanqin.com

## Questions?

<https://createpoint.qti.qualcomm.com>

