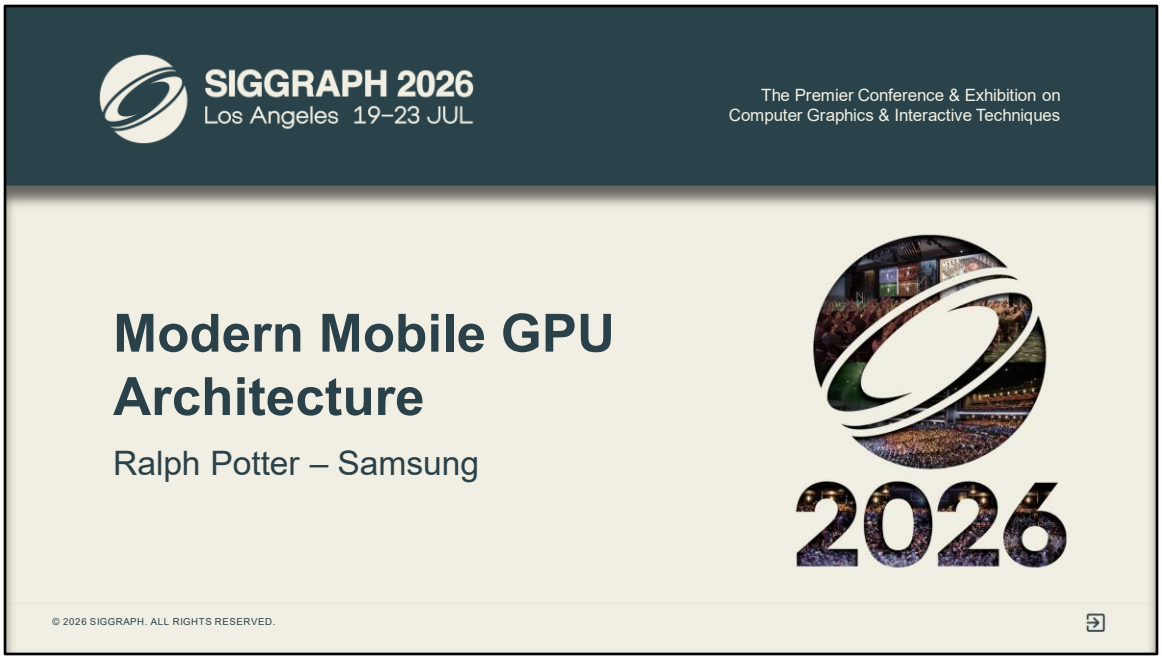

SIGGRAPH 2026
Los Angeles 19–23 JUL
The Premier Conference & Exhibition on
Computer Graphics & Interactive Techniques
Modern Mobile GPU Architecture
Ralph Potter – Samsung
2026
© 2026 SIGGRAPH. ALL RIGHTS RESERVED.

Talk Roadmap

1. Framing — The Power Constraint
2. Memory & Data Movement
3. Mobile Workloads & Thermals
4. IMR vs. Tile-Based Rendering
5. IMR vs. TBDR Continuum
6. Graphics API Implications
7. Ray Tracing & ML Acceleration
8. Developer Takeaways

It's Not a Weaker GPU. It's a Different Problem.

~100x

lower sustained power budget than a single discrete desktop GPU

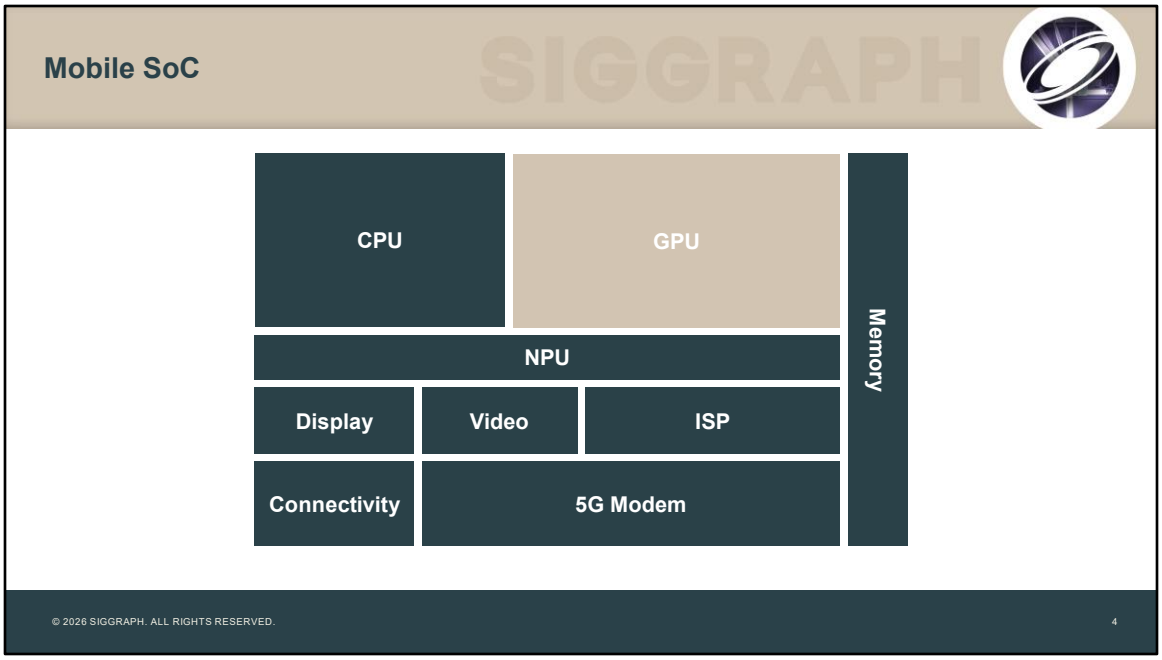
— A mobile SoC runs its entire chip — CPU, GPU, NPU, modem — inside a sustained budget of a few watts

— A discrete desktop GPU alone can burn two orders of magnitude more, on its own

— It's not a feature gap — mobile GPUs support the same core functionality, including ray tracing and ML acceleration

— This is not a scaling difference — it's a design-philosophy difference

— Every architectural choice in this talk follows from this one constraint: tiling, fixed-function hardware, ML offload, and the API decisions built on top of them

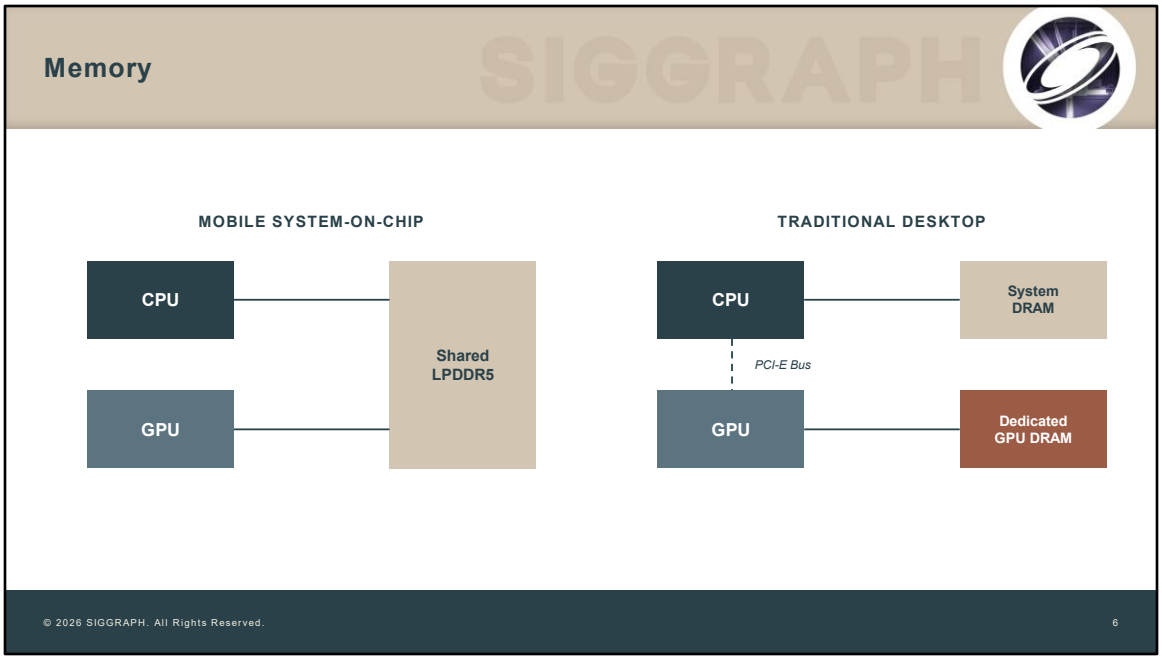


Mobile SoC's pack a lot of functionality into a small space. This block diagram was produced from a specific high-end 2024 SoC, but it's pretty representative of the set of IP blocks and their relative areas for most modern SoCs.

A couple of take aways from this slide:

- There are a lot of dedicated IP blocks each performing specialized tasks in an SoC
- They all share resources, particularly memory (and thermal headroom).
- The system will aggressively power down blocks that aren't in use…
- …but it's pretty easy to dream up use cases that require large parts of the SoC.

One example is an AR application utilising the ISP (because it uses the camera), the CPU for simulation and animation, the GPU for rendering, the display processor for compositing and output, and the connectivity blocks for location data or communication to servers. All of those blocks will communicate via memory. We are pretty close to powering on the whole SoC at that point.



Memory is a particularly important consideration on mobile devices.

In a traditional desktop system with a dedicated GPU, we typically see large amounts of dedicated graphics RAM located on the addon card. This RAM is designed to favour bandwidth over latency, and is typically coupled with an extremely wide bus. This provides huge memory bandwidth - around 1TB/s. Meanwhile the CPU in such systems operates in conjunction with more traditional DRAM which favours latency. With both compute devices operating on their own RAM, there is relatively low contention between them.

By contrast, on mobile devices we have a single pool of RAM. This means that we are sharing the available bandwidth between the CPU, GPU and any other components of the SoC that are active at the time.

What Mobile GPUs Actually Render

The Dominant Workload

— Layered 2D UI — stacked, often semi-transparent panels, text, and widgets, not complex 3D scenes

— Relatively static — most of the screen doesn't change from one frame to the next

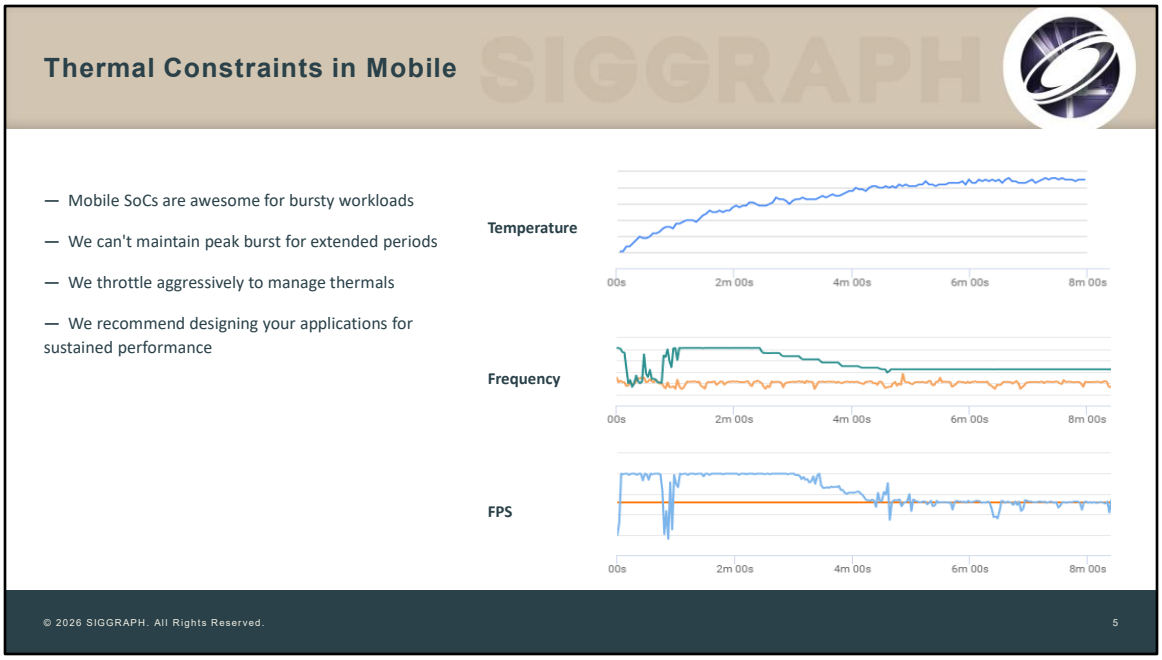
— Bursty and event-driven — rendering fires in response to touch, scroll, and animation, not a continuous simulation loop

Architectural Implication

— Geometry processing matters less — few triangles, mostly simple quads and layers to transform

— Fragment shading and hidden-surface removal matter more — many overlapping, blended layers make overdraw the dominant cost

— This is exactly the workload tile-based, fragment-first architecture is built to optimize



As I alluded to on the previous slide, mobile devices are passively cooled and therefore thermally constrained. Mobile SoCs handle short bursts of activity extremely well. This correlates well with typical consumer usage patterns. When you want to scroll over a complex webpage we can turn on a large CPU core, clock the GPU up and ramp the refresh rate up to 250Hz. This gives you a great, fluid user experience. What we can't do is maintain that level of performance for extended periods of time.

All electronics generate heat as a by-product, and that heat needs to be dissipated somehow. In a passively cooled phone, that heat is ultimately conducted to the surface of the device, and dissipated into the air. The amount of energy we can dissipate in this manner is related to the surface area of the conductor (i.e. the size of your phone) and the difference between the ambient air temperature and the surface temperature of your device.

These factors are relatively immutable. Consumers want phones that fit comfortably in their pockets, and they don't want to get burned while holding them. Even if we got a revolution in battery technology, our power budget is not going to significantly grow. Consequently mobile GPU design is all about doing more with less.

When Samsung engineers collaborate on optimization with game studios, one of our first steps is typically to lock the CPU and GPU frequencies on a handset to levels that can be sustained indefinitely, and to optimize against that rather than against peak.

Immediate-Mode Rendering

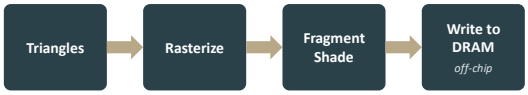
— Rasterize triangles as they arrive; write fragments straight out to a framebuffer in off-chip memory (GDDR / HBM)

— Bandwidth treated as an abundant resource — solved with wider, faster memory buses

— Discrete memory pools; PCIe transfer is the main CPU–GPU friction point

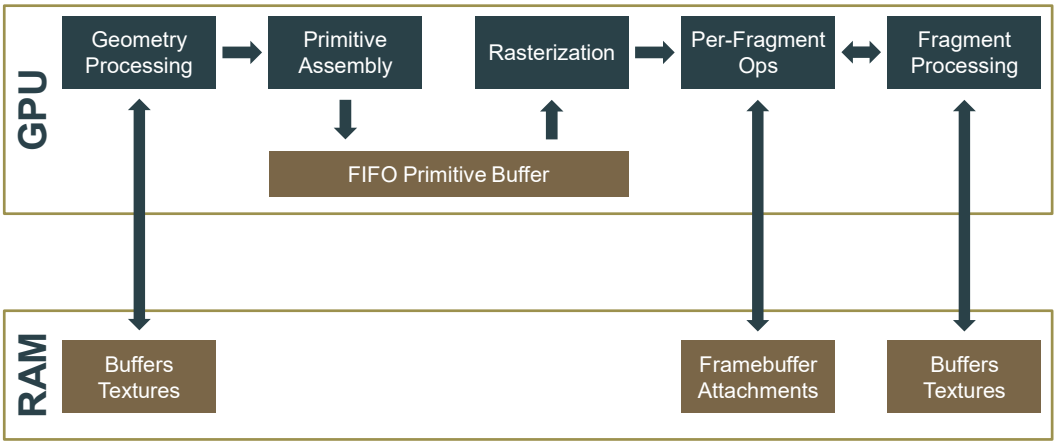
— The whole design assumes power is available to spend

DESKTOP PIPELINE



Every fragment write and every texture sample crosses the off-chip memory bus. Power cost scales with GDDR/HBM traffic, so the answer is simply a wider, faster bus.

Immediate-Mode Rendering



Tile-Based (Deferred) Rendering

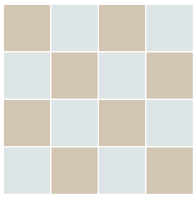
— On-chip SRAM access is far cheaper, in power, than off-chip DRAM access

— A binning pass sorts geometry into small screen-space tiles (e.g. 32×32)

— Each tile is shaded entirely in on-chip memory, then resolved to DRAM once

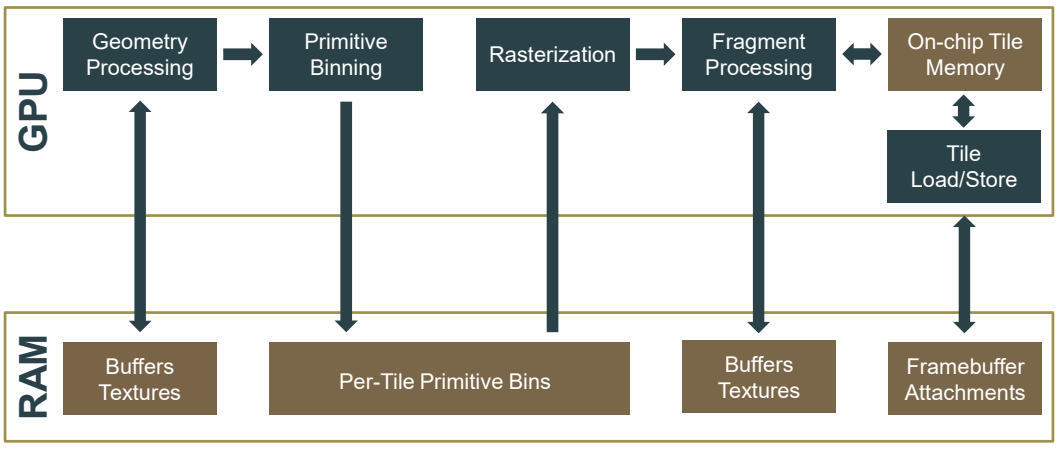
— The deferred variant adds hidden-surface removal before shading — an overdraw win on top of the DRAM-avoidance win

TILED PIPELINE



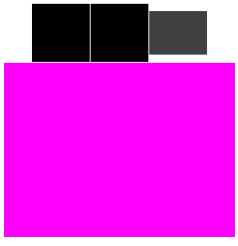
Each tile shaded independently, entirely on-chip

Tile-Based (Deferred) Rendering



IMR vs. Tile-Based Rendering

IMR (per-primitive)



TBDR (per-tile)

Real GPUs on the IMR ↔ Tiler Continuum

IMMEDIATE-MODE — FULL TILE-BASED DEFERRED

1 — 2 — 3

Xclipse — IMR-Leaning

Samsung's GPU, built on AMD RDNA IP. Inherits a more immediate-mode-rooted rendering pipeline from its desktop lineage. No tiling, but some primitive binning.

Adreno — Hybrid

Qualcomm's GPU. A binning-based hybrid tiler that sits in the middle of the spectrum. Dedicated GMEM to support large tiles. Able to switch between immediate and tiled rendering modes.

Mali — Classic Tiler

Arm's GPU. Tile-based deferred renderer, designed around on-chip tile memory. Fragment pre-pass for hidden-surface removal

No mobile GPU is a pure point on this line — every vendor sits somewhere between full immediate-mode and full tile-based deferred rendering.

Graphics API Implications of TBDR

Structure render passes so the tiler keeps work in on-chip tile memory — and avoids round trips to main memory.

1

Keep render passes intact

Don’t break a render pass unnecessarily — every split forces the tile to resolve to DRAM and reload.

2

Use free clears & invalidation

Rely on load-op clears and invalidate attachments when you’re done. The tiler handles these on-chip, at no bandwidth cost.

3

Release attachments early

Don’t retain attachments such as depth longer than needed — holding them forces needless write-backs to memory.

Design your content to be TBDR-friendly, and mobile IMR performance will be fine too.

Ray Tracing on Mobile: Hybrid by Necessity

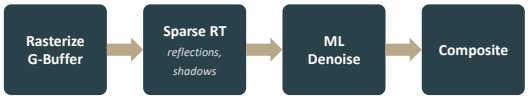
— Hardware ray tracing exists on mobile now — this is not a roadmap slide

— Rasterized G-buffer + a sparse, high-value ray budget (reflections, contact shadows), composited

— Denoising is essentially mandatory given sparse ray budgets

— BVH build/traversal is its own power cost — dynamic scenes pay for it twice

HYBRID RT PIPELINE



Ray budgets sit orders of magnitude below desktop or console path tracing — the pipeline is shaped around scarcity, not abundance.

The Next Wave: ML Acceleration

Machine-learning acceleration is the next wave of mobile hardware evolution — and the industry has not yet converged on how to implement it.

1

Tensor cores

Dedicated matrix-multiply units inside the GPU, in the style of NVIDIA's tensor cores.

2

WMMA on compute units

Wave Matrix Multiply-Accumulate instructions that reuse the existing GPU compute units, as in AMD RDNA.

3

Dedicated NPU

Offload ML entirely to a separate neural processing unit, purpose-built for the workload.

API interfaces are also evolving — co-op matrix, co-op vector, and data graphs.

Opportunities for ML in Mobile Pipelines

1 **Temporal Upscaling & Reconstruction**

Hitting visual targets without brute-forcing native resolution

2 **ML-Assisted Denoising**

Making sparse ray-traced samples look dense

3 **Frame Generation**

Extending perceived frame rate within a fixed power budget

4 **ML-Based Texture / Material Compression**

Trading compute for bandwidth savings

Same throughline as every other technique in this talk: more perceptual quality per watt.

One Constraint, Cascading Everywhere

The line between “mobile” and “desktop” is blurring.
RDNA is moving down into mobile, while Adreno moves up into desktop on Windows-on-Arm devices.

The gap isn’t feature set or capability — it’s all about power and thermal budgets.

Developer Takeaways

Design for the mobile power budget — the one constraint that shapes every decision above.

1

Respect the memory hierarchy

Keep work in on-chip tile memory. Structure render passes to avoid DRAM round trips — don't break passes or over-retain attachments.

2

Design for sustained, not peak

Mobile throttles hard. Target steady, sustained frame pacing rather than short bursts of peak performance.

3

Spend rays & ML wisely

Apply ray tracing and ML where they pay off — sparse rays, upscaling, denoising — always inside the thermal envelope.

Same capabilities as desktop — winning on mobile means spending your power budget wisely.