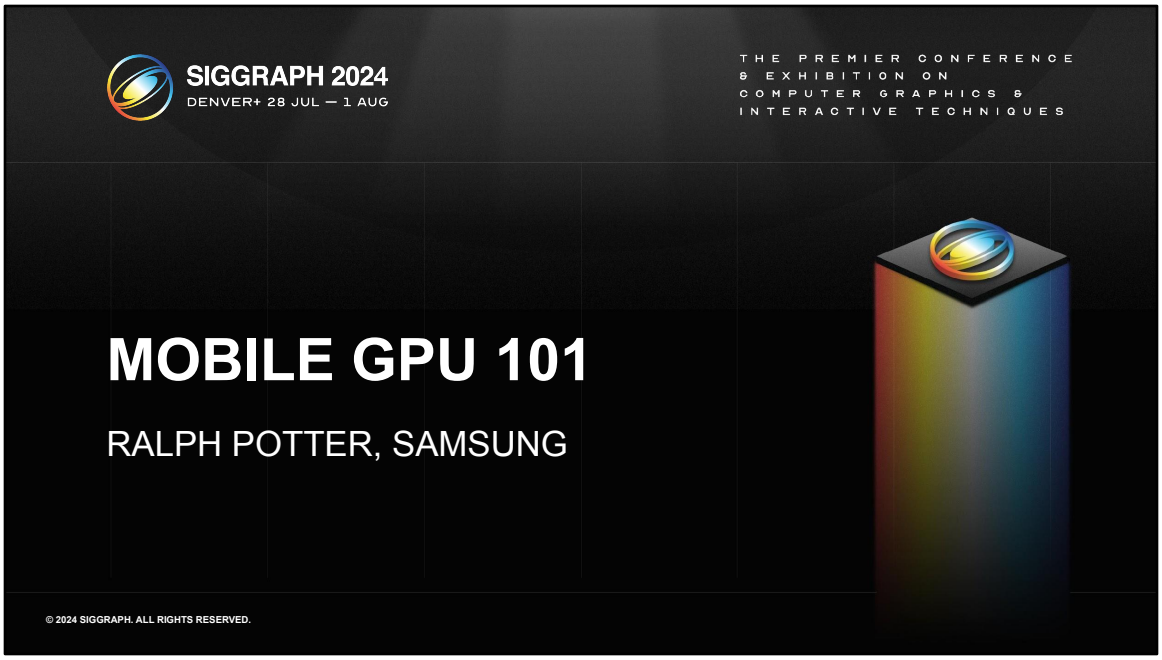
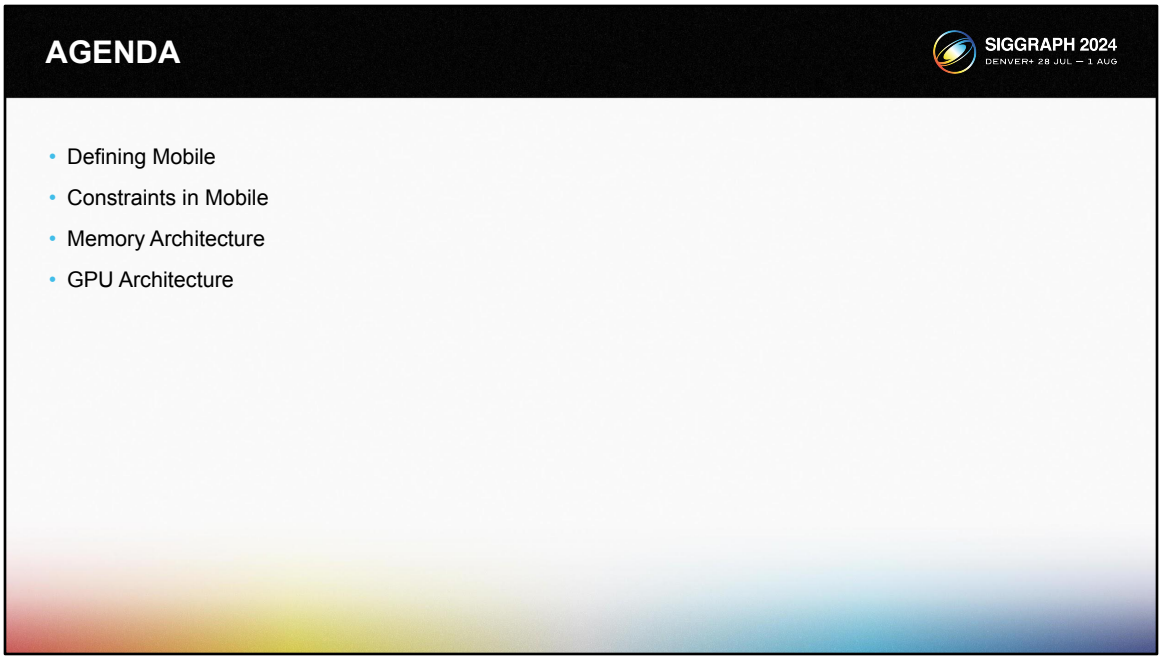



Hello everyone. I'm Ralph Potter. I'm a GPU engineer at Samsung Mobile, where have acted as Samsung's primary representative to Khronos and particularly the Vulkan Working Group for the past 5 years. Prior to that I've been a GPU driver engineer and GPU compiler/programming model researcher.

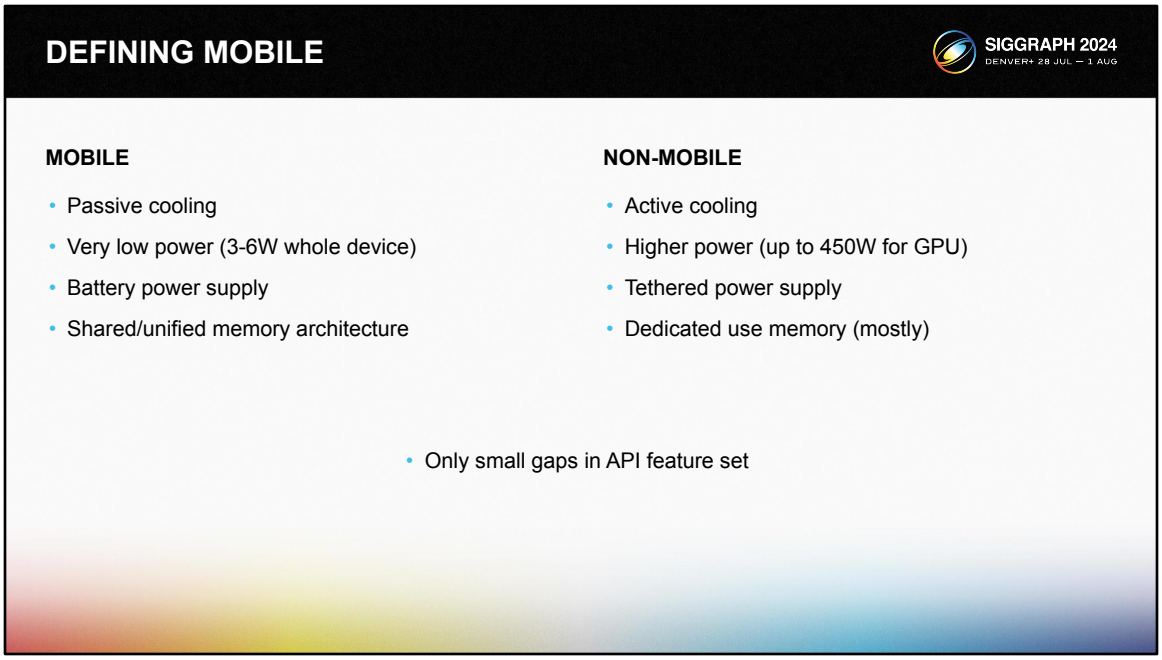
My team within Samsung has worked on handsets using GPUs from every mobile hardware vendor. This year we will ship devices using at least three different GPU architectures, so I have a broad view of the current offerings.

In this talk I'm going to talk about the architectural basics of mobile GPUs and how they differ from traditional desktop parts. Essentially I'm aiming to provide the context for all the cool new stuff that the other presenters are going to talk about today.




In the course of this talk, we are going to cover:

- A definition of “mobile” so that we are all on the same page
- Some discussion of the constraints that mobile devices operate under
- The differences in memory architecture between mobile devices with unified memory architectures and PC addon cards with dedicated memory, and the importance of those differences
- An overview of mobile GPU architectures and how the most common devices differ from more mainstream desktop parts




So what do we mean by “Mobile GPUs”? We are talking “phone-like”. These devices are typically passively cooled, battery powered, thermally-constrained devices.

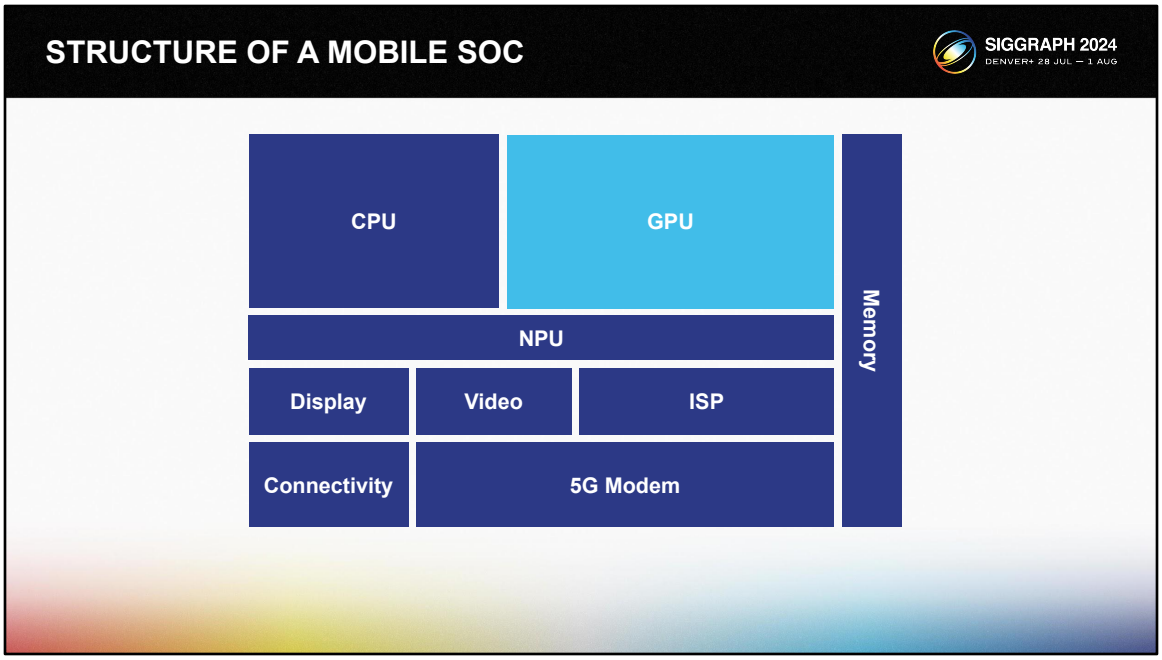
Power budgets are a huge factor here. A flagship desktop GPU can draw upwards of 400 W when heavily loaded. You might see certain flagship mobile handsets temporarily skip up to about 8 or 9W for the entire handset, but typical sustained power draw is lower – around 3-6 W. Notice that I didn’t say 8-9W for the GPU. That’s the whole handset.

There is a continuum here. You will see GPUs from what are traditionally “Mobile GPU IP” vendors in actively cooled devices like Windows-on-ARM laptops or handheld gaming devices. Those might architecturally be the same GPU, but the addition of active cooling changes the operating constraints. Architecture-specific optimization guidance might be the same for both sets of devices, but there is a lot more headroom available in the actively cooled devices. The Steam Deck and certain variants of the Samsung S24 are both RDNA 2 GPUs, with almost equivalent feature sets but one of them has active cooling, a much higher power budget and a much shorter expected battery life without charging.

For the purposes of this talk, we are restricting ourselves to devices like smartphones,

tablets and maybe some embedded devices.

One final point, the feature set functionality gap between mobile and desktop is actually very small today. For example, HW ray tracing is available from all mobile SoC vendors. We are seeing neural graphics techniques start to appear on mobile. There are a few features that aren't prevalent yet for architectural reasons, but the gap is now small. The distinction gap between mobile and non-mobile is largely a numbers game i.e. can you fit content into the resource constraints, rather than a question of whether functionality is available at all.




In mobile devices, we pack a huge amount of functionality into a single silicon package. We refer to this as a System-on-Chip or SoC.

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. You will see that the GPU is the largest individual block. Along with the GPU, we have a cluster of general-purpose CPU cores, neural processing accelerators, an image signal processor for the camera, HW video accelerators…the list goes on and on.

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.

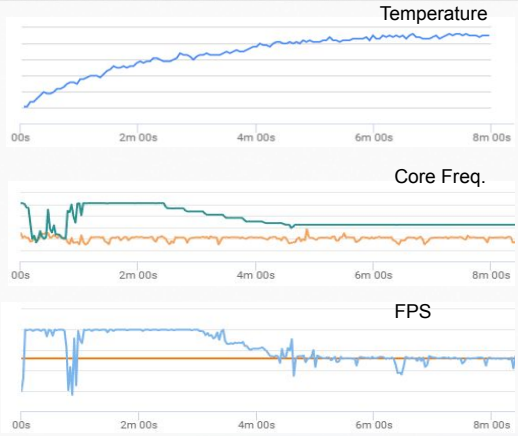
One example is an augmented reality application. Think of something like Pokemon Go. In an application like this we are utilising the ISP (because we are using the camera), the CPU for simulation and animation, the GPU for rendering, the display processor for compositing and output, and the connectivity 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.

## THERMAL CONSTRAINTS IN MOBILE

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

- Mobile SoCs are awesome for bursty workloads
- We can't maintain peak burst for extended periods
- We throttle aggressively to manage thermals
- We recommend designing your applications for sustained performance

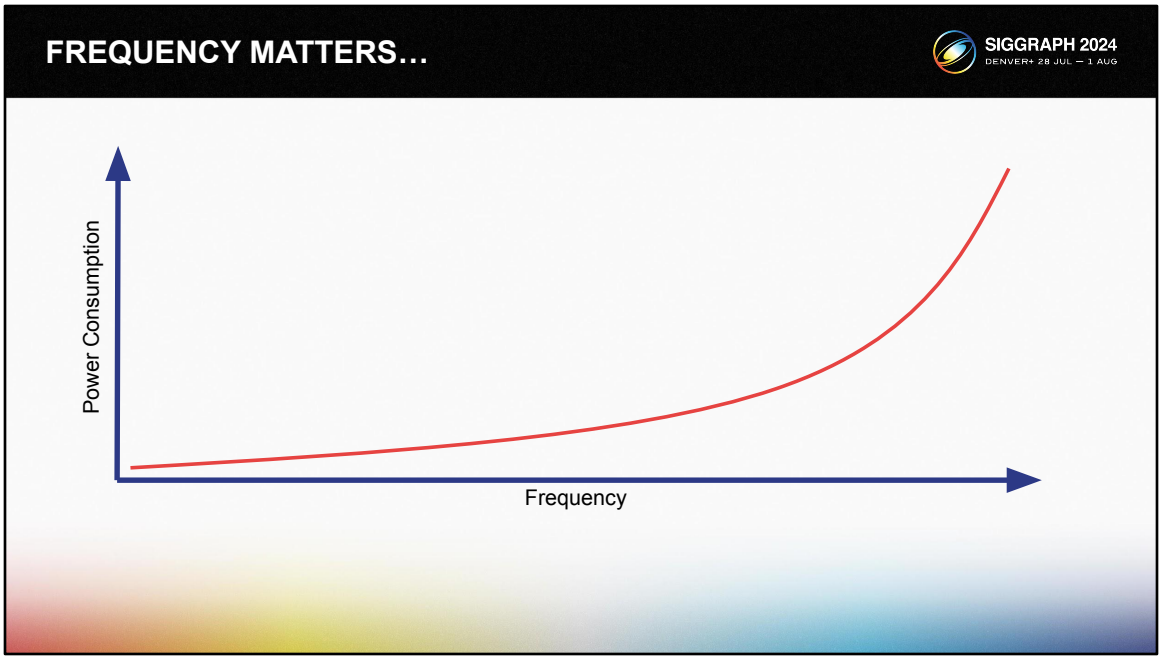



As I alluded to on the previous slide, mobile devices are passively cooled and therefore thermally constrained. Mobile SoCs handle short bursts of activity extremely well. This correlates well with typical consumer usage patterns. When you want to scroll over a complex webpage we can turn on a large 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 your 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.

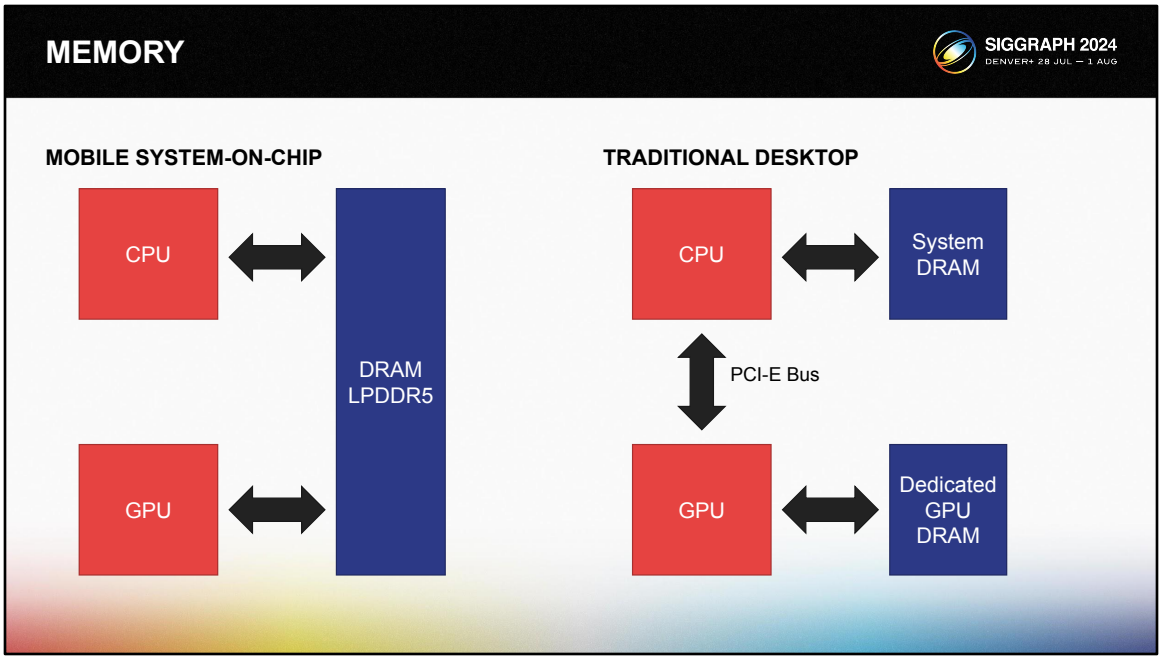
When Samsung engineers collaborate on optimization with game studios, one of our first steps is typically to lock the CPU and GPU frequencies on a handset to levels that can be sustained indefinitely, and to optimize against that rather than against peak. I strongly recommend profiling your applications for sustained performance, not the performance you get in the first 30 seconds.




I mentioned on the previous slide that we will aggressively throttle frequencies to manage thermals.

Increasing frequency requires us to increase voltages, and that means we see a quadratic scaling of power consumption as frequency increases.

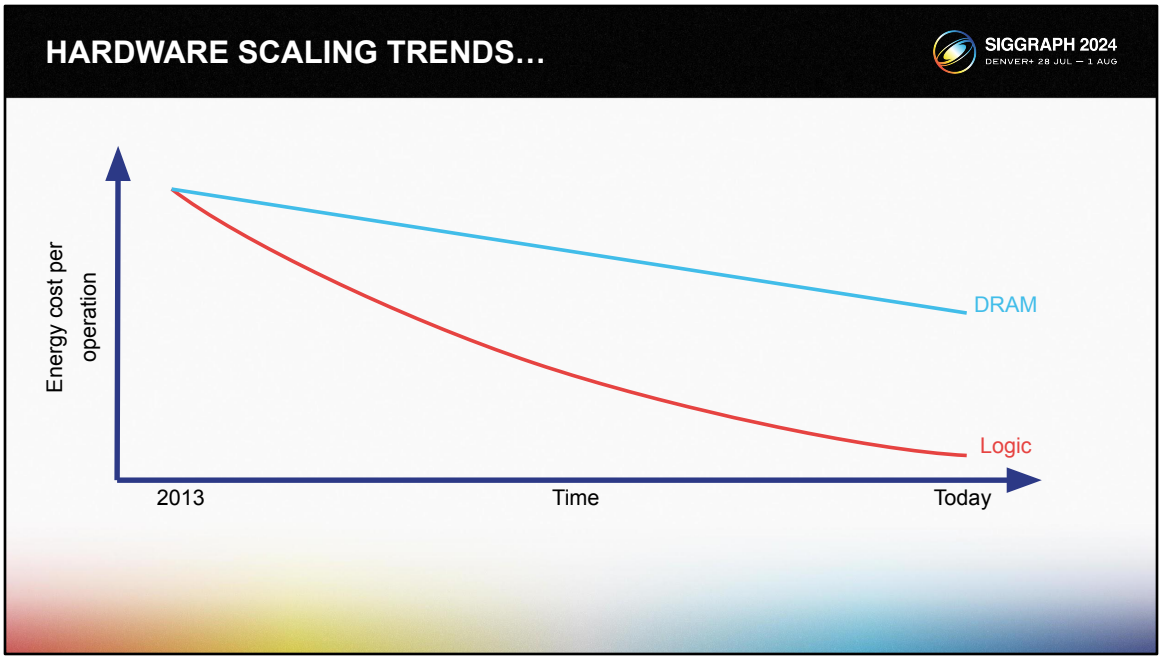
Since power consumption maps to heat generation, lowering frequencies results in large reductions in heat generation




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. This RAM is optimized for general purpose workloads, where latency matters. We also have much narrower buses. You can expect see available memory bandwidth to be at least an order of magnitude lower.

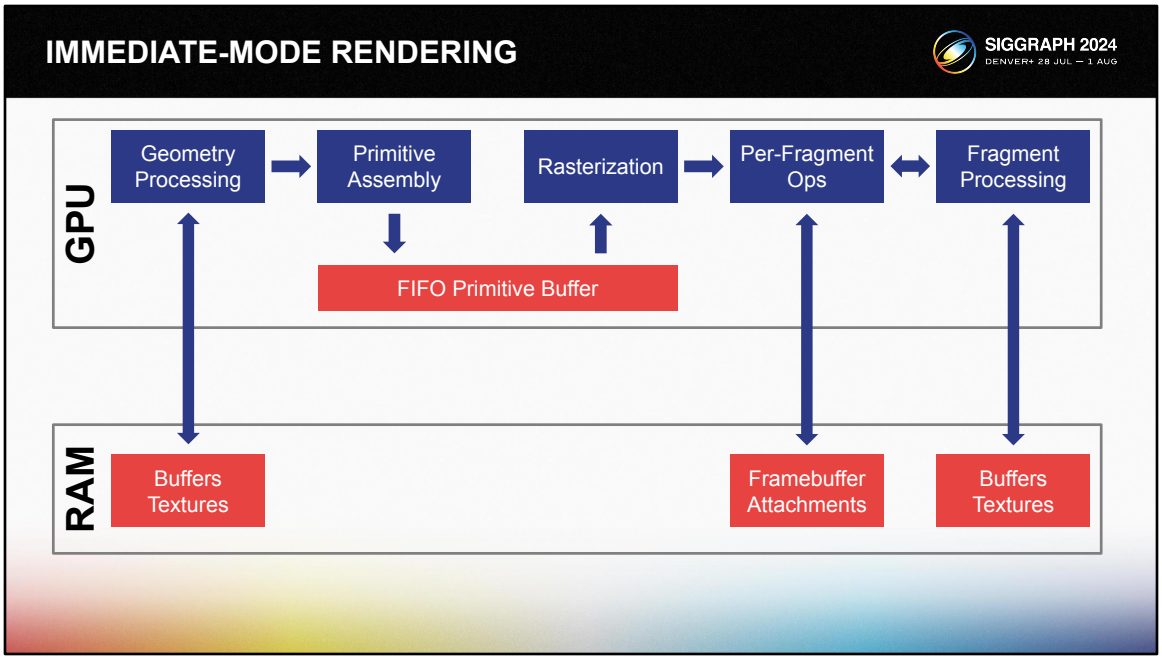



Continuing on the topic of memory, it's helpful to look at how energy costs of arithmetic logic and memory operations have scaled over time.

We are managing to pack more and more arithmetic logic into the same power budget. DRAM energy costs have improved, but not at the same rates.

We also see a similar story for throughput here, where the amount of computation mobile GPUs can do is scaling faster than memory bandwidth.

The take away here is that touching memory is expensive, and you should think about whether your algorithms can minimize memory accesses.




Next up, we are going to spend a few moments talking about two common architectural models for GPUs. If this was 2021, I would have told you that all mobile GPUs are primarily tile-based GPUs, and desktop parts are immediate-mode GPUs. Today there is a bit more nuance to that. Tile-based GPUs still dominate the market, but some Samsung handsets are now based on an IMR.

For anyone coming from a desktop environment, or who has learned about GPU architecture from an introductory graphics course, this is probably the model that you are familiar with.

* Vertex data is pulled from the input vertex buffers, transformed via vertex shaders, assembled into primitives (typically triangles) and streamed into a first-in, first-out buffer.
* Triangles are pulled from that FIFO buffer as fast as the rasterizer can, rasterized into fragments, and depending on Z-buffer tests, may ultimately be emitted into the frame buffer

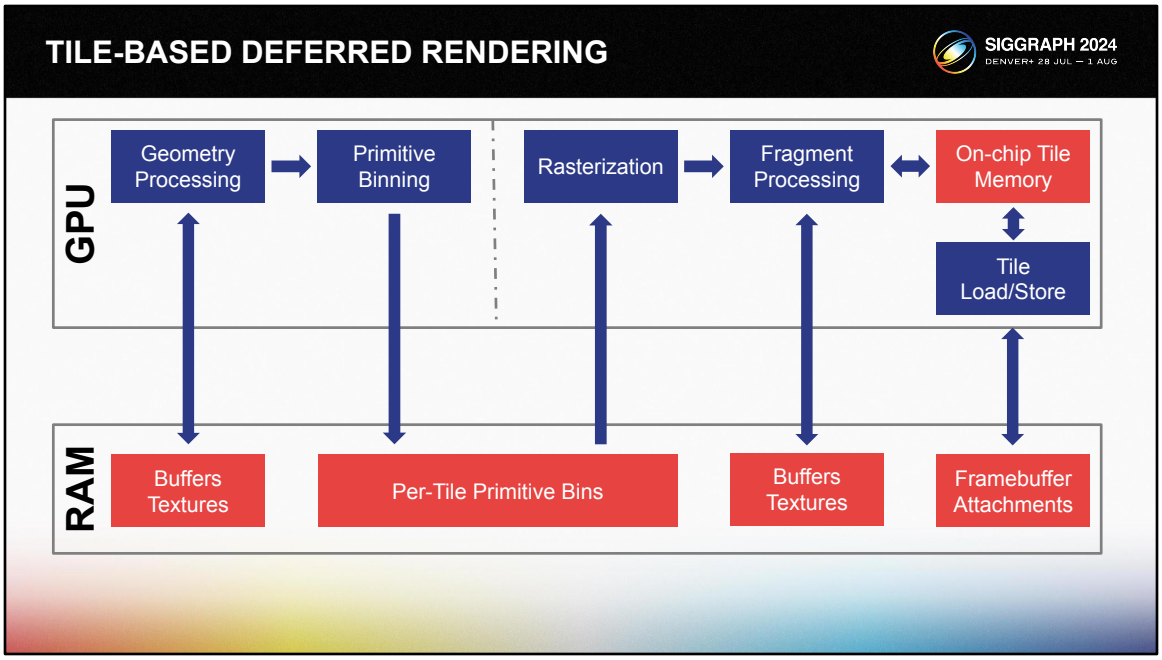
In this model, we are essentially streaming the input geometry. Only a small amount of transformed geometry exists within the pipeline at any one point in time. If a rendering algorithm relies on multiple passes, then the results of each pass will be

written out to DRAM and read back again when used as an input for a subsequent pass. This incurs memory bandwidth costs at the fragment processing end of the pipeline.

## IMMEDIATE-MODE RENDERING

This animation illustrates how immediate mode rasterization constructs a final output frame buffer. The image is constructed triangle-by-triangle, rendering directly to DRAM.

You'll also see that some triangles get overwritten by later triangles as the process progresses. This is effectively wasted memory bandwidth




Tile-based GPUs follow a slightly different pattern. The fundamental idea here is to divide the geometry and frame buffer into tiles, where shading each tiles can be done in parallel as if you were rendering a miniature image. In order to do this, we need to transform the geometry positions up front so we can assign each primitive, based on it's coverage, into some binning structure that can be processed as tiles.

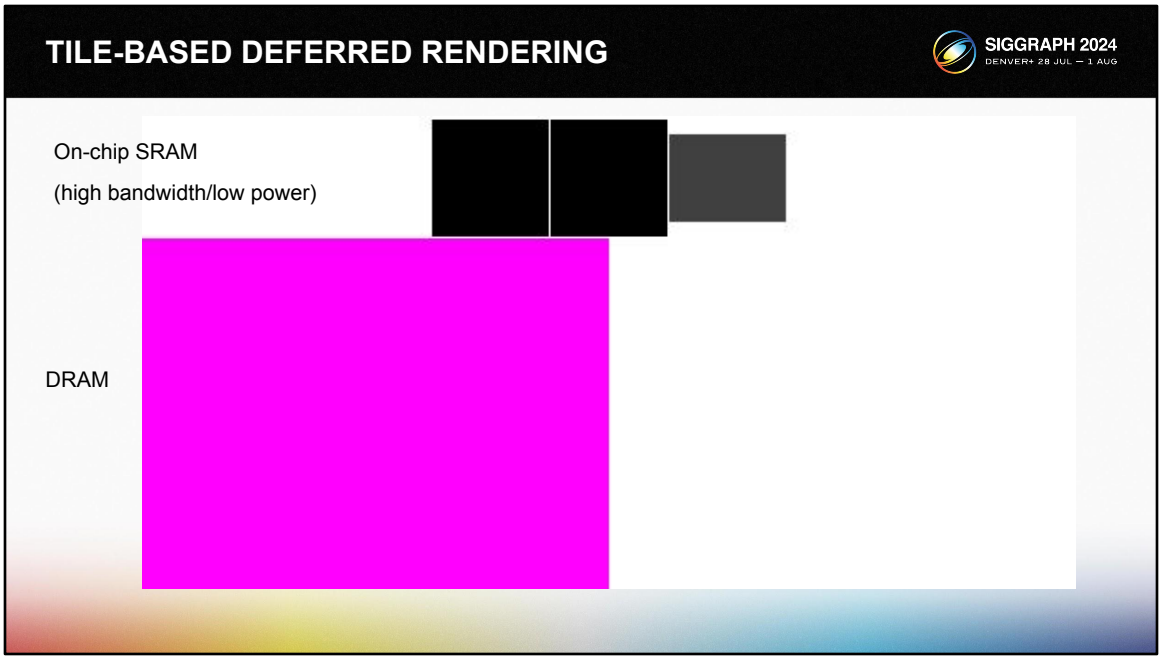
Once the binning structure has been created, the GPU will then iterate over each framebuffer tile, rasterizing all of the geometry from with it. There are variations between implementations primarily around tile size and geometry handing, but the common theme is the decomposition into tiles that can be processed independently. The key win here is that all the framebuffer memory for a tile can now fit in a small, fast piece of memory located within the GPU itself – the 'on-chip tile memory' in this diagram. This allows all framebuffer operations to stay within the GPU and only framebuffer attachments that are needed would be written to system memory. So, for example, if the depth buffer was not required after rasterization, it can be discarded without being stored to RAM.

We are making a trade-off by adopting a TBDR architecture – they assume that reducing the cost of framebuffer operations is more important than reducing the cost of geometry processing, so workloads that are very geometry heavy can

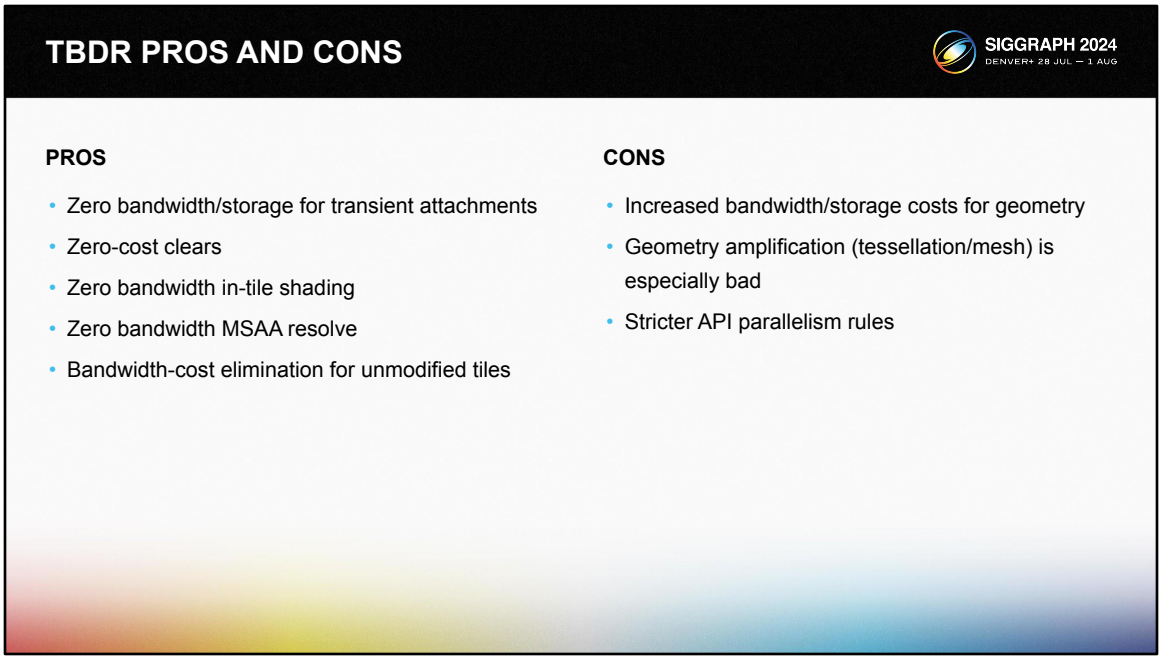
underperform an IMR. They can also incur higher frame latency because of the need to first undertake geometry binning.

For fragment heavy workloads there are major wins at the fragment shading end of the pipe. You can get clears for (almost) free because they operate on the tile, you can perform hidden surface removal on the binned geometry and reduce overdraw, and any overdraw you do get is relatively cheap because it occurs on-tile. Only the final rendered image needs to be committed to DRAM. Even then, some hardware will checksum the output block vs. the existing content and skip writes where nothing changed.

Deferred lighting algorithms are a good example of how this might all combine provide a win. In these algorithms, we are going to generate a set of intermediate values (albedo, surface normal, depth) and then combine them together to generate a final image. If we can keep those intermediate values on-chip while we perform lighting, and then discard them without writing to system memory, then significant bandwidth savings are achieved.




In a TBDR, the GPU selects a small tile of the image, rasterizes that in high-performance on-chip memory, and then only stores the final output to main system DRAM. Any overdraw happens on-chip (and may be entirely eliminated in some cases)




Tile-based architectures bring us a lot of wins.

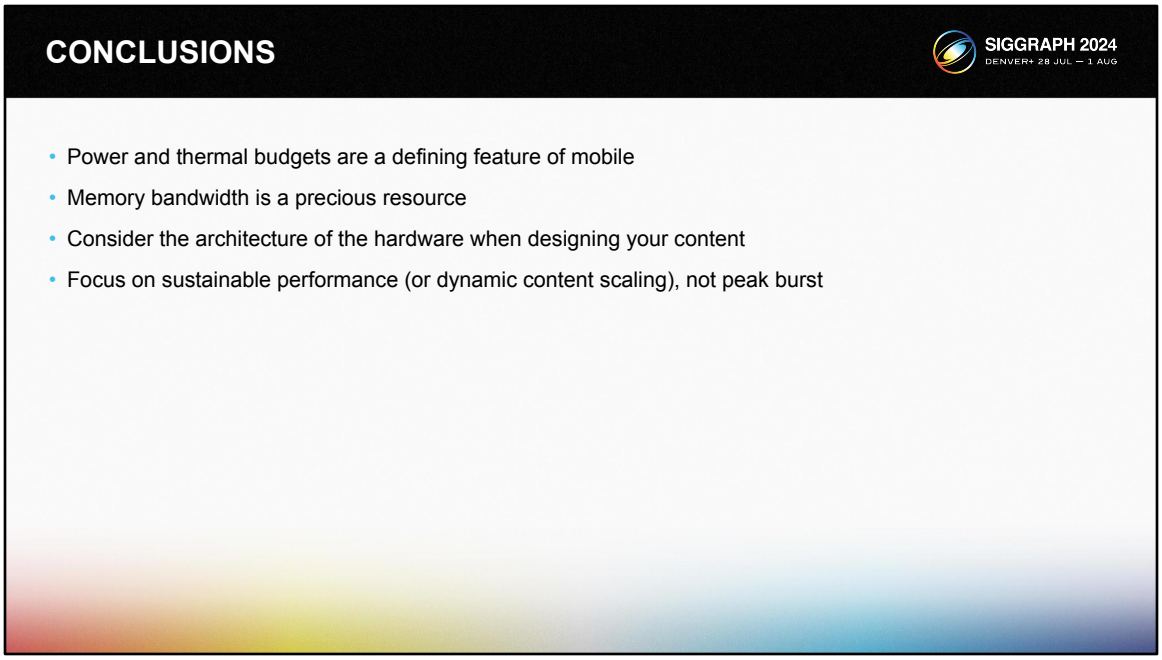
- Transient attachments that are only used to generate intermediate outputs during the composition of a final scene can be kept on-chip, at minimal bandwidth cost
- Similarly, clears and discards can be performed on-chip, without touching DRAM
- The same gains can be seen with multi-sampled anti-aliasing. The intermediate multi-sampled image can remain on-chip, and only the final resolved image written out to DRAM

All of this comes at some cost though.

- In an IMR architecture, we can start shading geometry as soon as we had assembled the first triangle. With TBDR we need the geometry binned first and that has to be stored in DRAM
- This makes both excessive micro-geometry and geometry amplification such as tessellation or mesh shaders a poor fit.

It’s useful to take a step back here, and think about what a mobile handset is doing the majority of the time when it’s not being used to play complex 3D games. UI rendering on a mobile handset is typically made up of a few hundreds of triangles

worth of 2D geometry with complex layering and blending. Most of that geometry is also not in motion the majority of the time. This is a great match for the strengths of tile-based GPUs, and it's not an accident that they have ended up with these properties.




To wrap up:

- Power and thermal behaviour are defining features of mobile
- Mobile devices will throttle if they start exceeding thermal limits
- Memory bandwidth is a precious resource, a major consumer of power, and a shared resource across the whole SoC – it's not just dedicated to the GPU
- Think about the architecture of mobile GPUs when you design your content. If you design a game with massive amounts of sub-pixel triangles, then performance will suffer on the most common GPUs in the market
- Sustainable performance is important. If you are optimizing by making a small tweak, testing for 30 seconds, and then letting the device cool down, then easy to get mislead.


SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG
THANK YOU