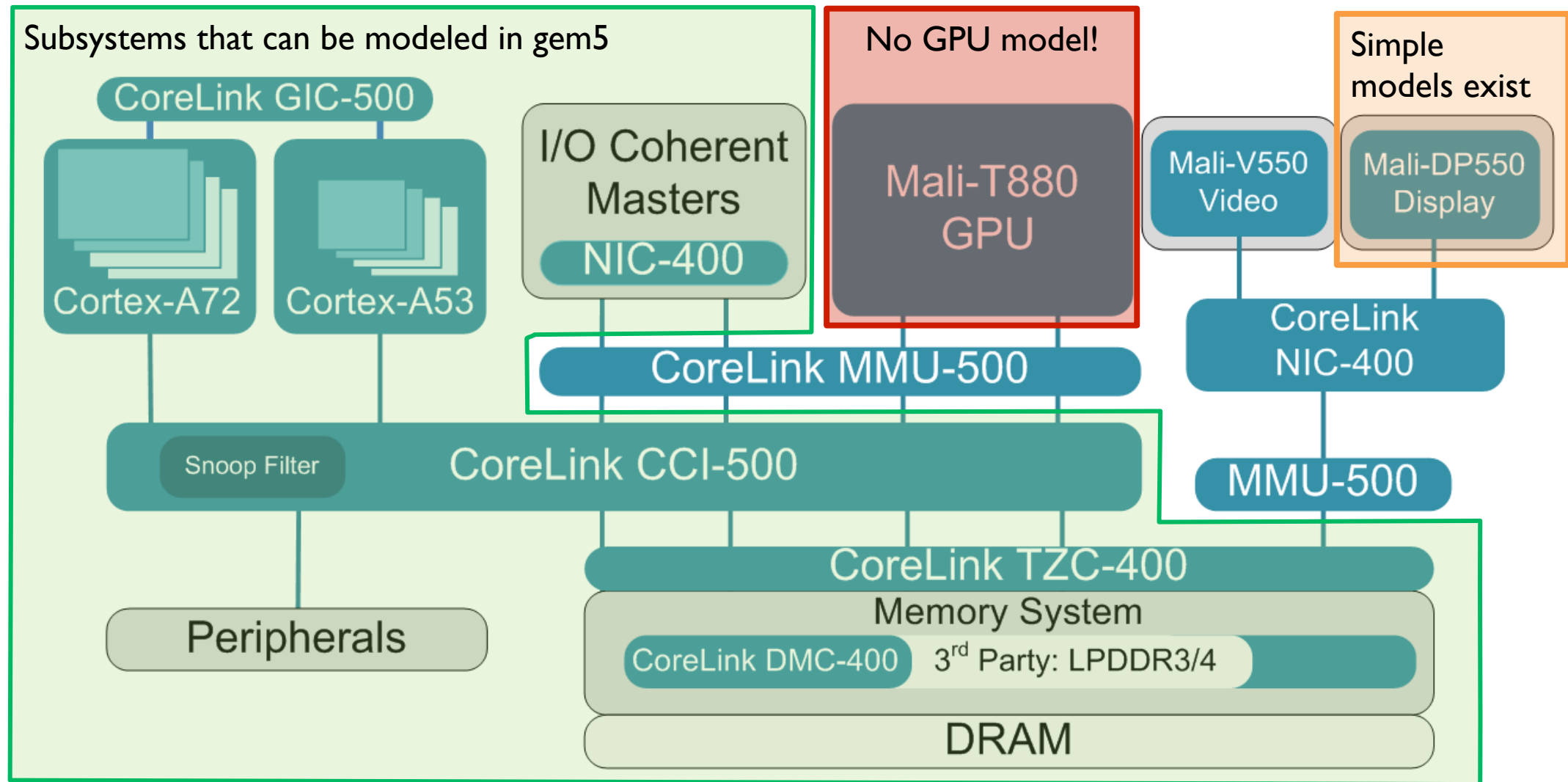


NoMali: Understanding the Impact of Software Rendering Using a Stub GPU

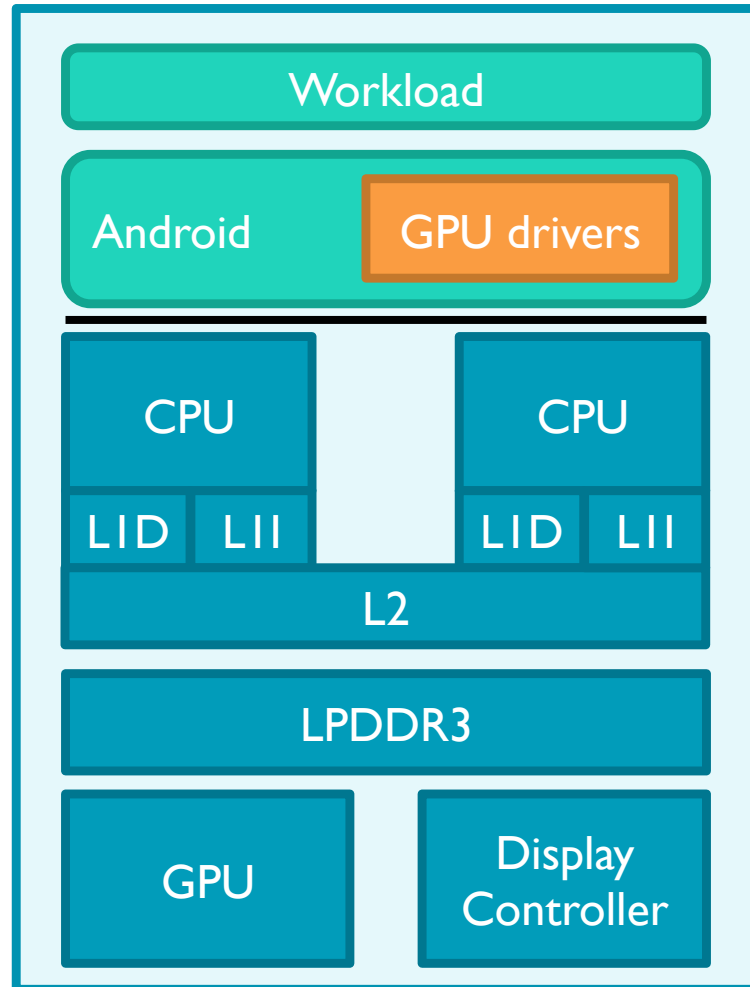
Andreas Sandberg
ARM Research

What can be modeled in gem5?



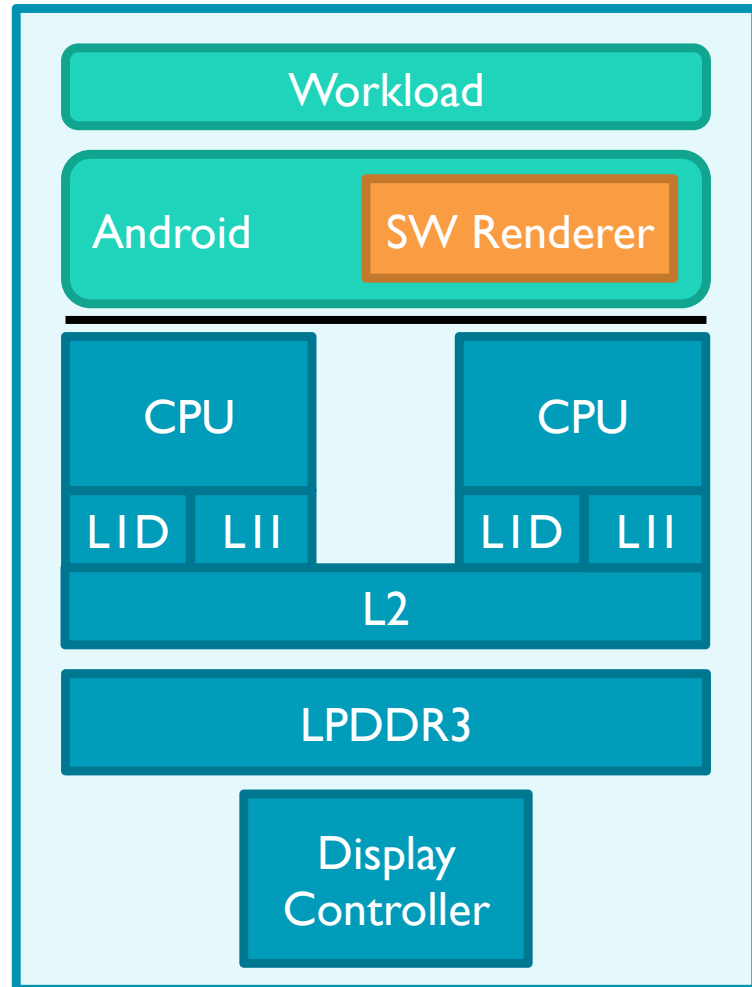
Note: gem5 models the subsystems above, *not* the actual products.

What a real system does



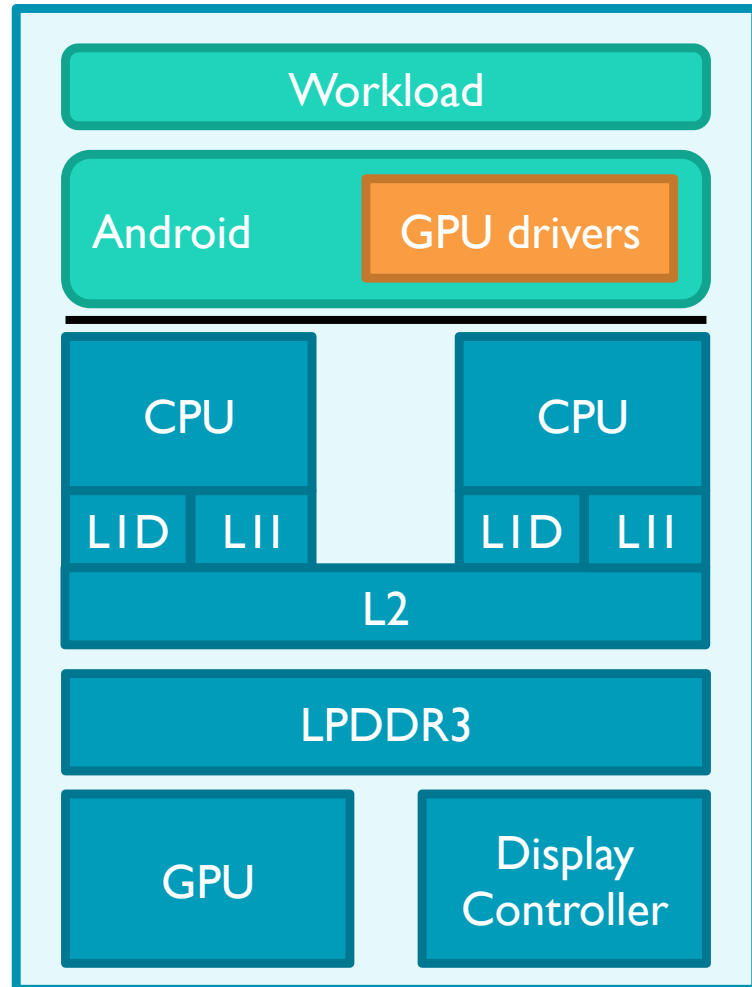
- Modern mobile systems contain a GPU
 - Even watches have GPUs nowadays!
- The GPU is obviously used for 3D
- ... but also used for 2D:
 - Composition & alpha blending
 - Rotation & scaling
- Driver stack is complex:
 - Easily 100k+ lines of code
 - Contains an optimizing compiler
 - Can contribute to around 10M instructions/frame for complex workloads

What we normally model



- Software renderer instead of a real GPU
 - Optimization friendly code
 - Can be vectorized
 - Easy-to-predict branches
 - Large memory foot print
- Workload and software renderer compete for resources
 - Can significantly skew core behavior
- Affects 2D applications *and* 3D applications

What about simulating the GPU?



- **Pros:**

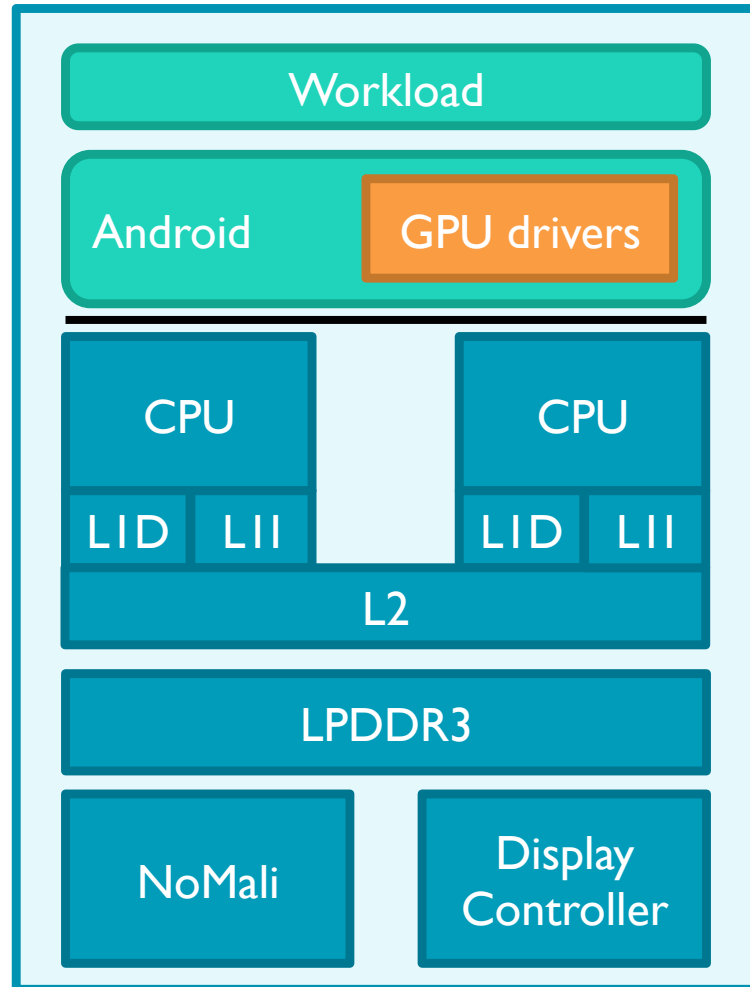
- Golden reference – everything looks like a real system!
- Captures memory system interactions
- Graphics output

- **Cons:**

- GPU models add a lot of simulation overhead
- Realistic models not available to the research community

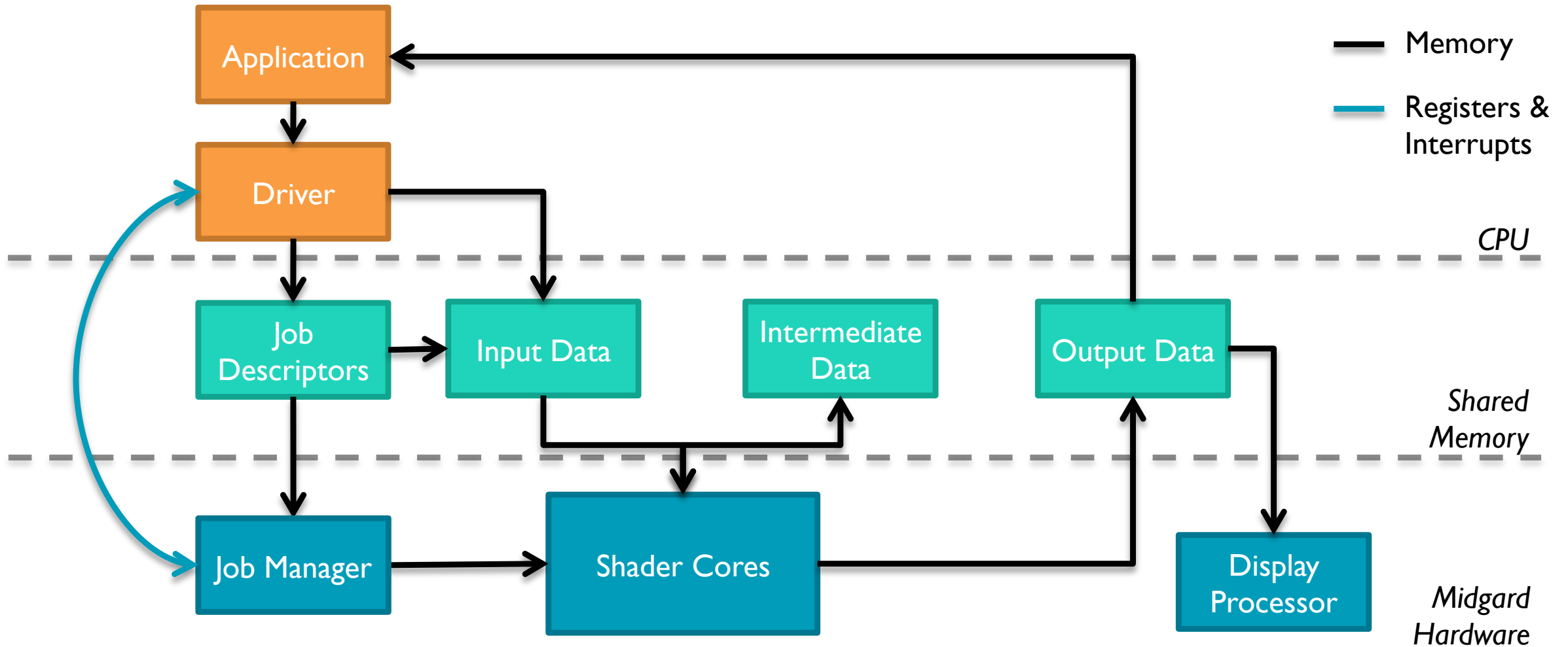
- **Solution:** Don't simulate the GPU!

Introducing NoMali



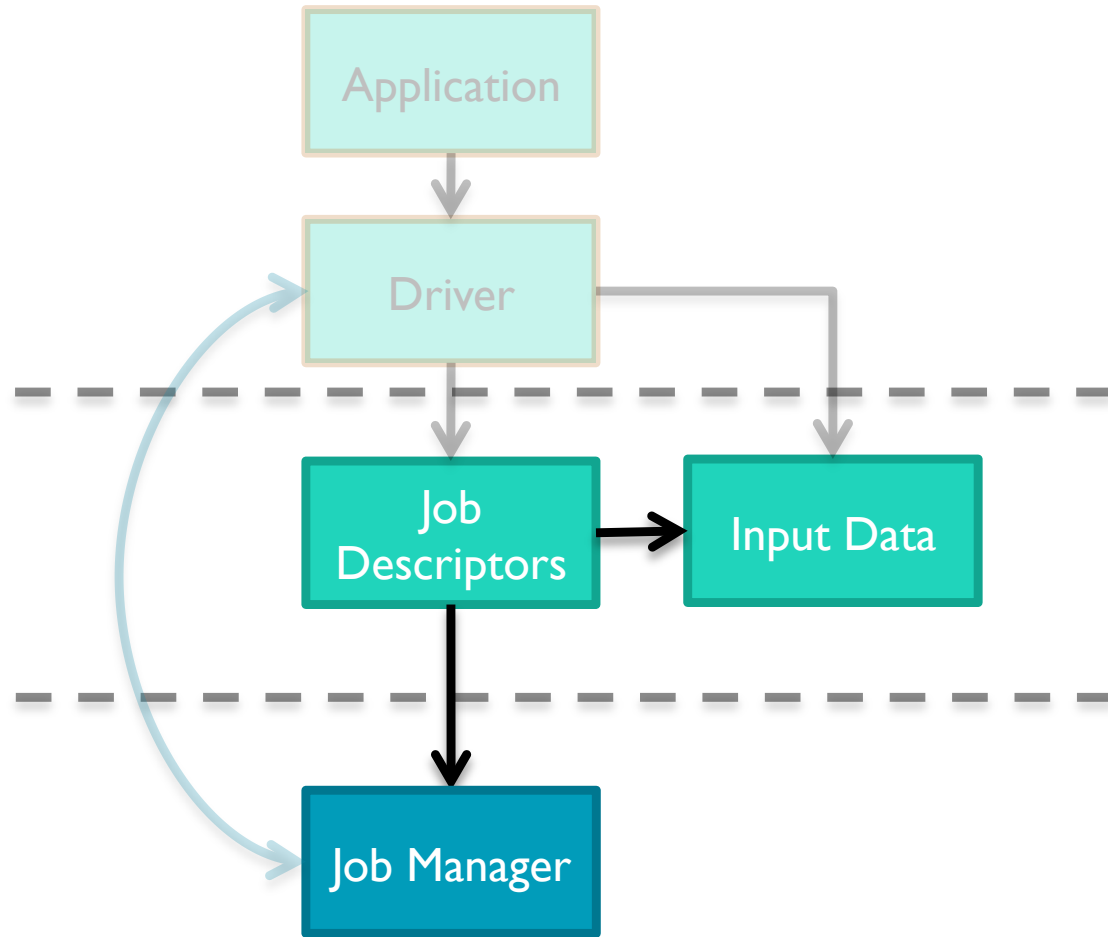
- Looks like a GPU
 - Provides the same register interface
 - Simulates interrupts
- Runs the full driver stack
- Pretends to run rendering jobs
 - Doesn't render anything
 - Signals job completion immediately
- Available to the community
- ... but doesn't produce any display output

Mali GPU overview



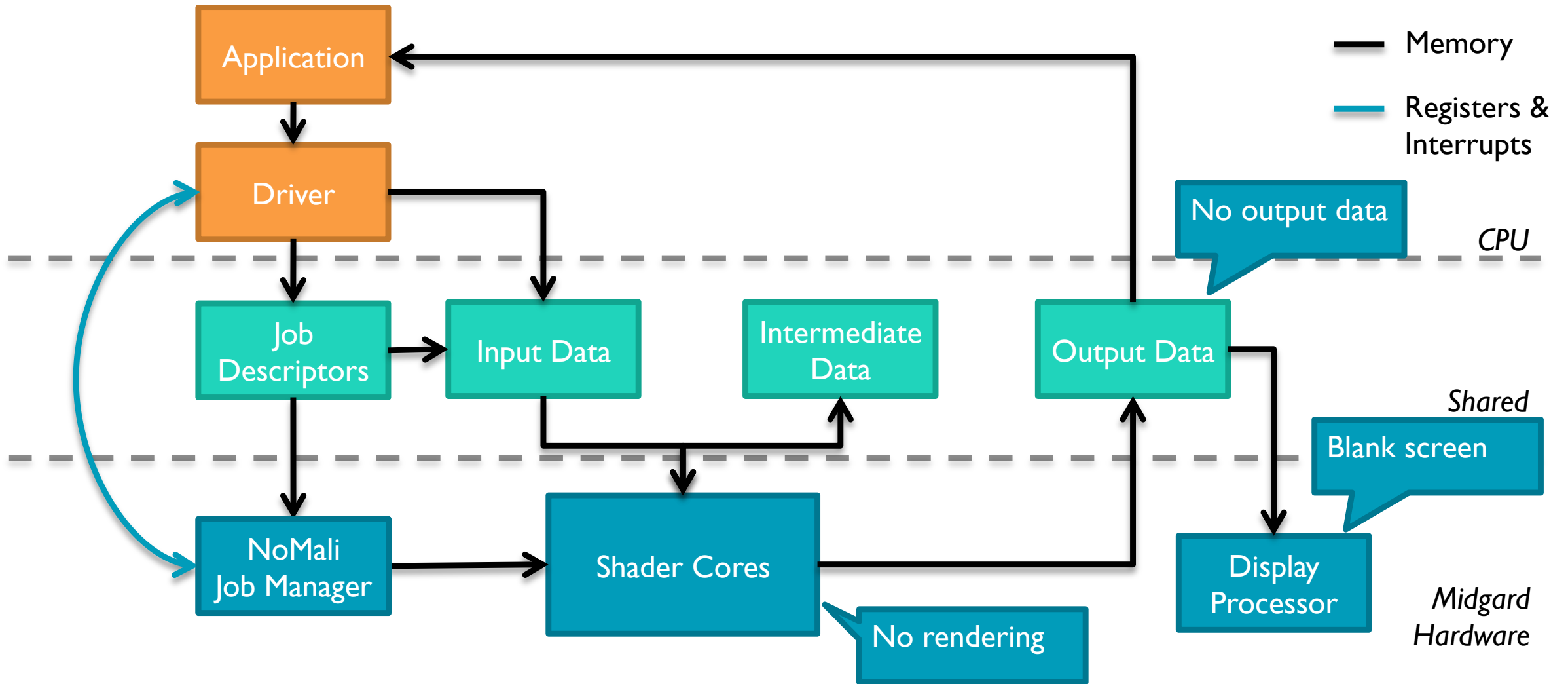
See AnandTech for a good architecture overview

Mali GPU overview: The Job Manager

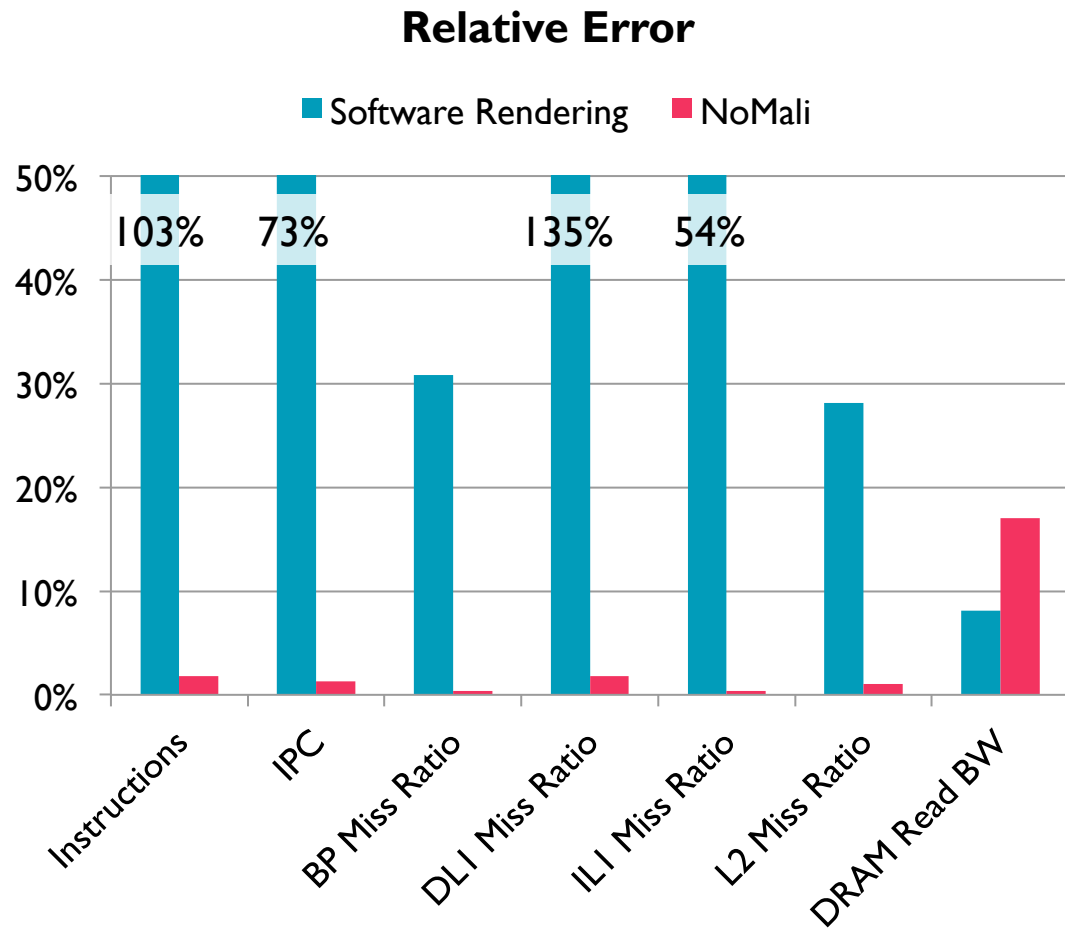


- Abstracts the underlying μ -architecture
- Controls most aspects of the GPU:
 - Job scheduling
 - Interrupts
 - Address translation
 - Caches
 - ...
- Job submission through a register interface
 - Job parameters in main memory: Job Descriptor
- Interrupt on job completion

NoMali overview

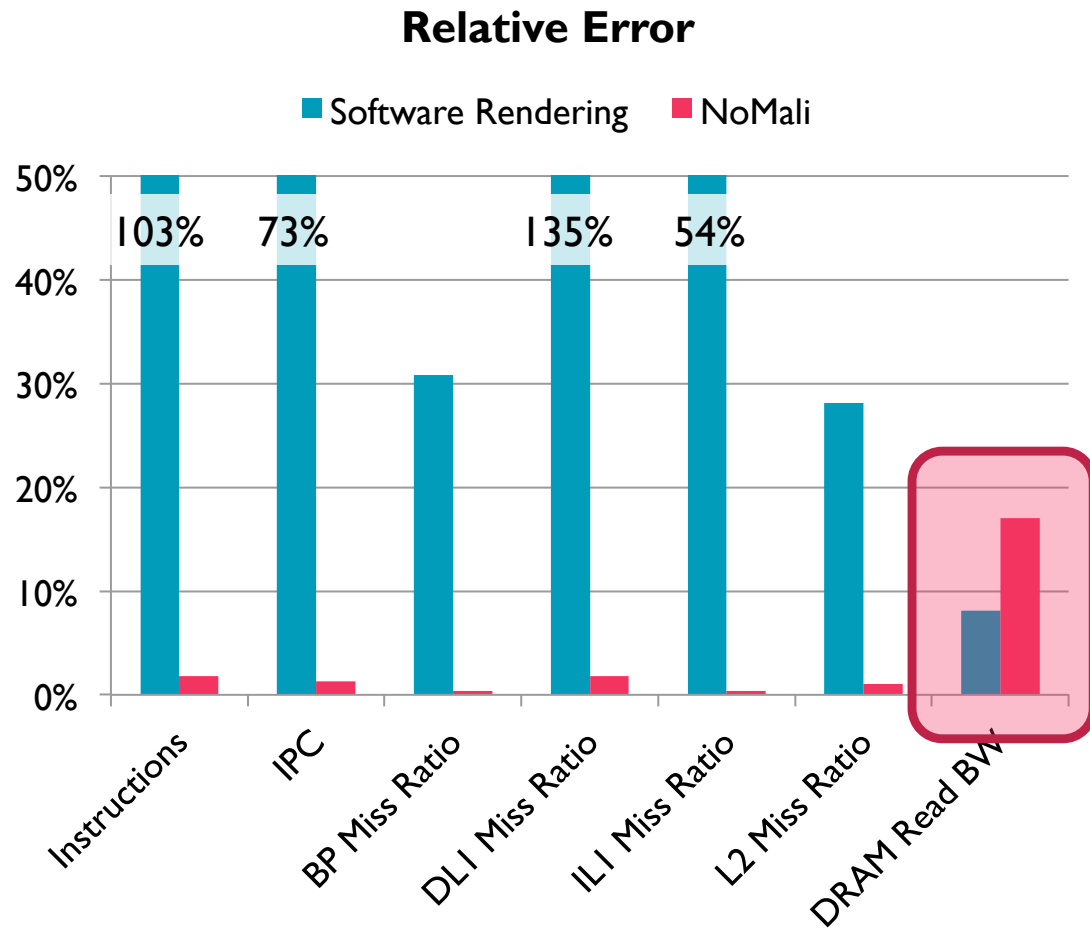


Comparing simulation strategies



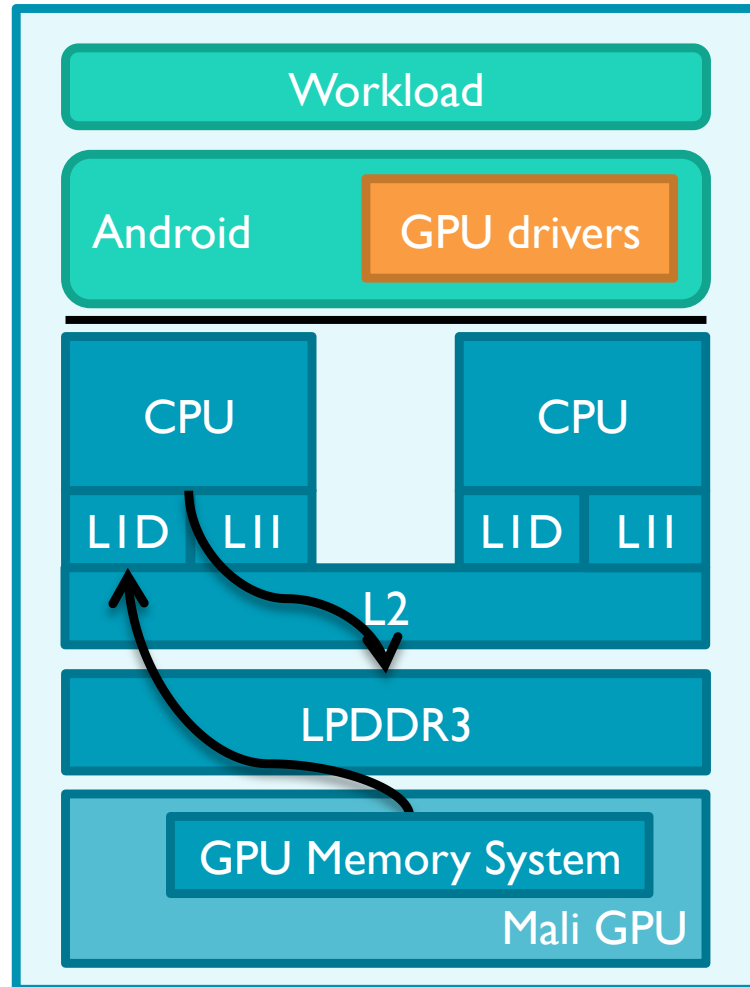
- Three experiments:
 - Software rendering, NoMali, GPU reference
- Experimental setup:
 - Android 4.4 (KitKat)
 - BBench
 - Identical disk images in all experiments
- Software rendering results in useless CPU performance
- NoMali is within 2% for CPU performance
 - ... and 35% faster!

Model limitation: GPU bandwidth



- GPU memory system interactions not simulated
 - Could potentially be faked using traffic generators
- Absolute difference for bbench ~75 MiB/s
 - Not likely to be a problem for CPU-centric studies
- Graphics workloads would experience a larger impact from the GPU
 - NoMali was never intended for that use case.

gem5 Issue: inefficient uncacheable memory



- Mali Midgard-series GPUs are IO coherent
 - GPU snoops into CPU caches
 - CPUs can't snoop into the GPU's caches
- The driver disables caching for many regions used by both the GPU and CPU
- Wasn't handled efficiently by gem5
 - Uncacheable accesses were always strictly ordered
 - Resulted in CPIs ~50 (should've been ~2)
 - Fix committed in early May 2015

gem5 integration plan

- NoMali Model available on GitHub
 - <https://github.com/ARM-software/nomali-model>
- gem5 integration on Review Board [RB2867, RB2869]
- Requires drivers
 - Will make use of drivers available from MaliDeveloper
 - Requires a recent Android version (KitKat or LolliPop)
- Android KitKat build instructions will be on the Wiki shortly

Questions?