



SONICS



# Conquering On-chip Memory Bandwidth Bottlenecks

September 2010

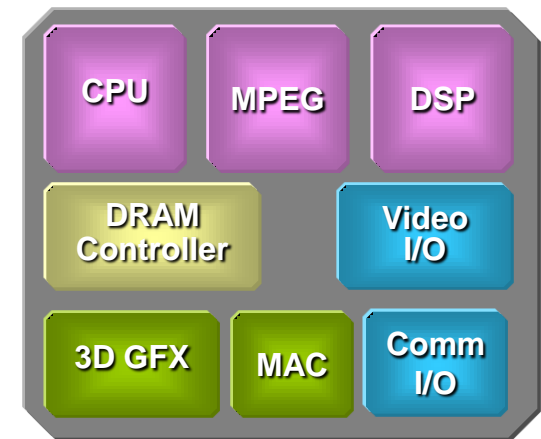
## *Driving SoC Complexity and Memory Bandwidth*

- Relentless push for higher quality user experience – at minimum system cost!
- Feature convergence – Video, Voice, Data, Audio (in every consumer device!)



*Massive feature integration: Driving SoC memory subsystem complexity to the extreme*

- Multiple processors
  - CPU processor
  - DSP processor
  - Graphic processor
- Many high-performance engines
  - Video cores (decoder, blitter...)
  - DMA engines
- All traffic goes to DRAM



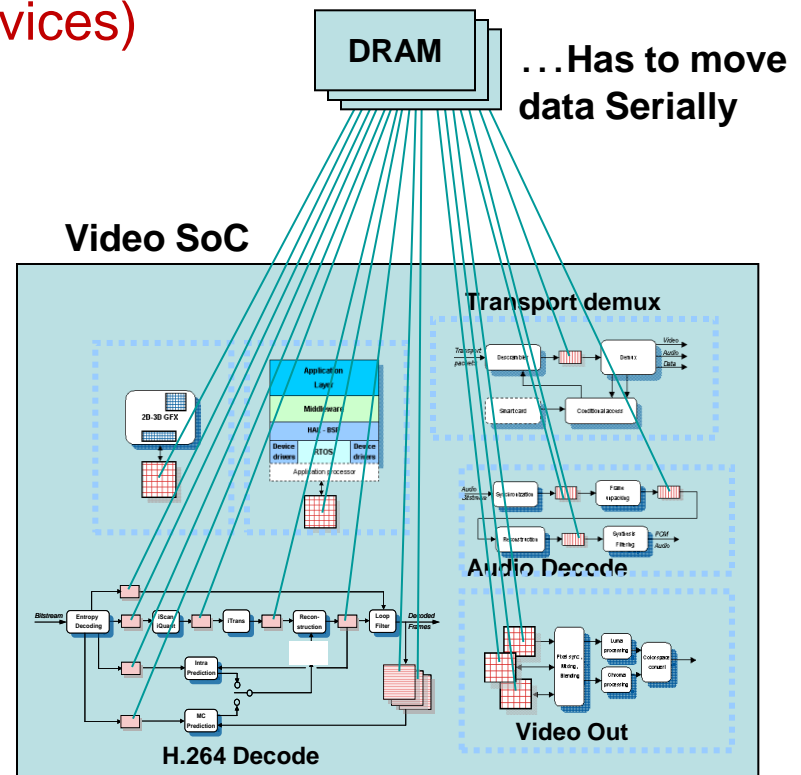
**System On Chip**

## Distributed Heterogeneous Architectures

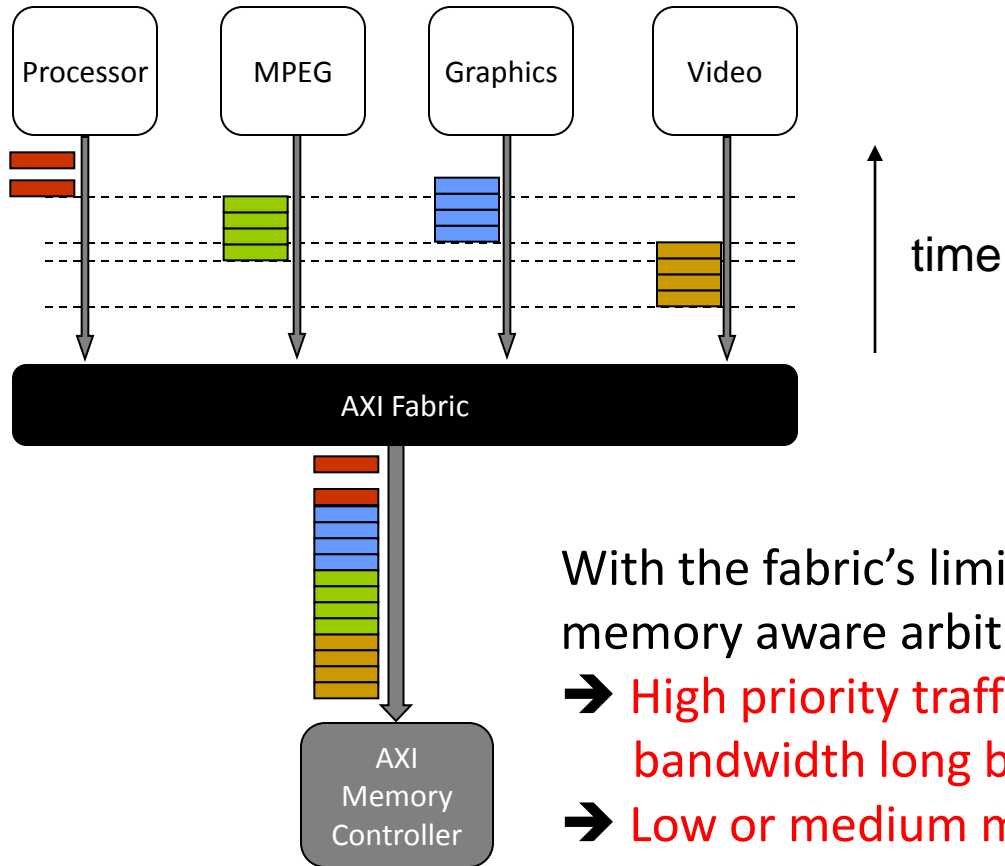
# The Ubiquitous Memory Bottleneck

*Consumer SoCs process data in parallel, but communicate...*

- Embedded SoCs require large amounts of memory (such as those found in advanced video devices)
- Limited bandwidth at the DRAM continues to be one of the unresolved challenges
- Competition among cores for DRAM degrades system performance, increasing cost and power consumption
- Advanced memory scheduling is one of today's most complex issues in advanced SoC design



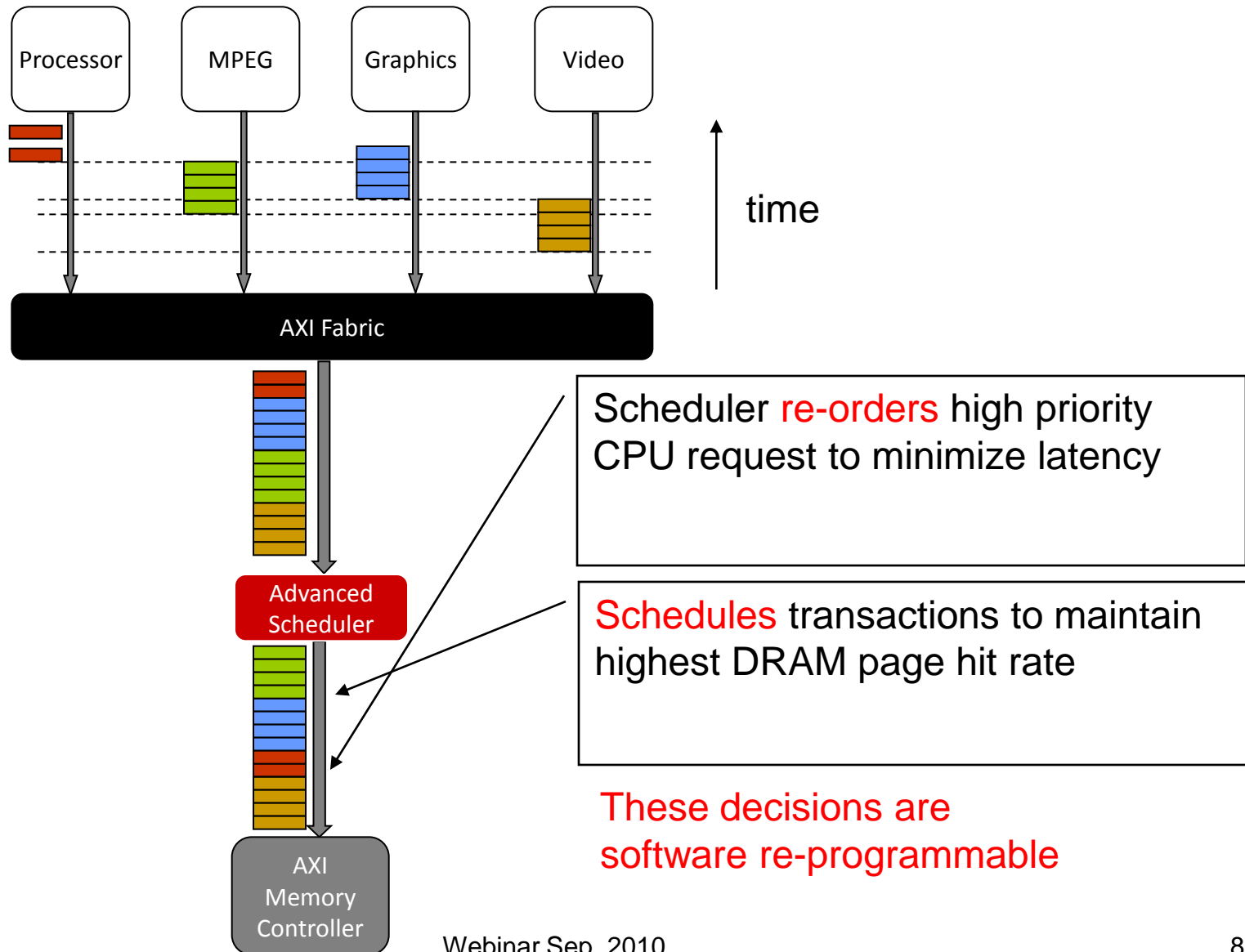
# The Memory Bottleneck Problem!



- Memory controllers today offer some degree of scheduling support
  - Limited/finite efficiency – 55-60% maximum
- Companies are forced to design their own controllers
  - Still with limited efficiency
- Labor to optimize off-the-shelf solutions
  - Minimal improvement for a large effort (<60%)
  - Add additional DRAM to the system – Adds COST!
- Focus remains on the PHY
  - Analog portion often dictates the controller

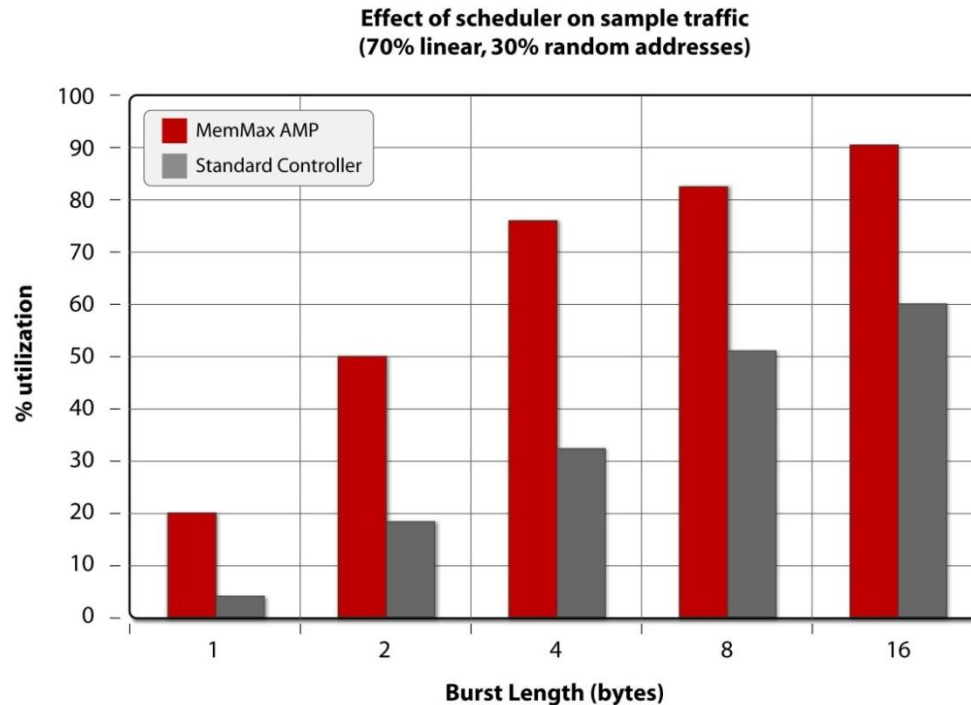
- **Speed upgrade DDR → DDR2 → DDR3, is it that simple?**
  - Higher frequency = higher power consumption
  - Higher frequency = tougher timing closure
  - Higher frequency = more expensive solution, maybe
- **Wider is better! Is it?**
  - Wider = more pins
  - Wider = bigger package
  - Wider = board layout challenges, more PCB layers
- **Add a 2<sup>nd</sup> memory port**
  - Architecture change = Software change (i.e. load balancing)
  - Now the scalability is lost

# The Importance of Efficient Scheduling



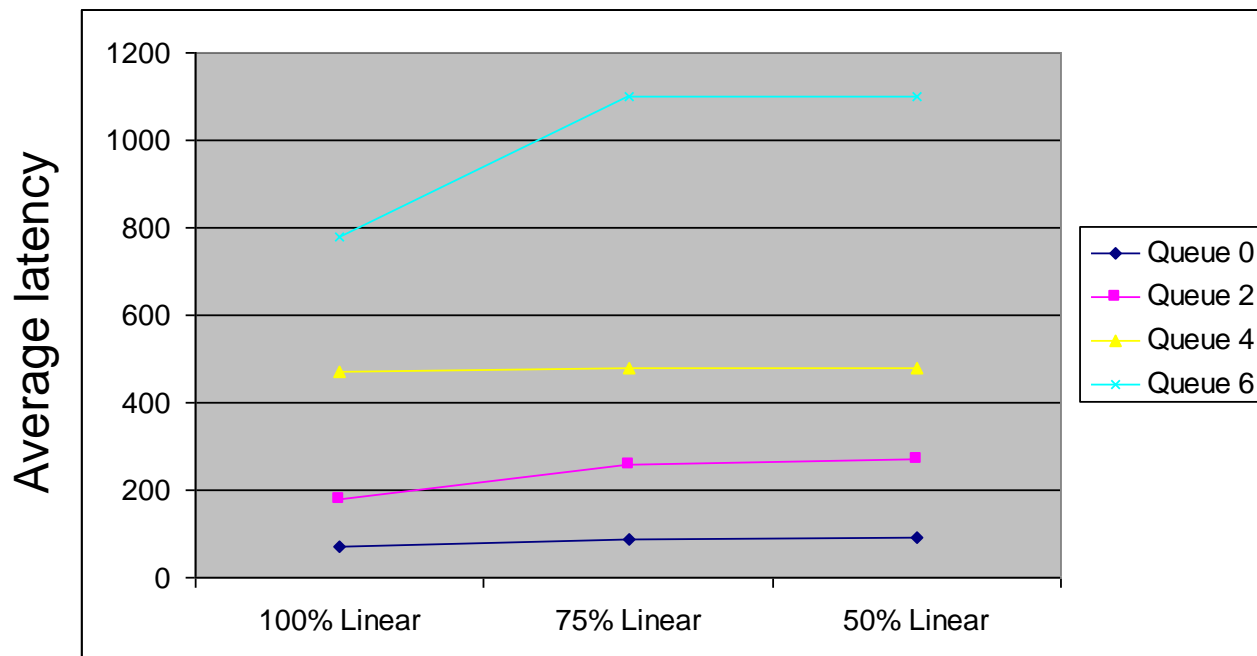
## ■ AMPLify DRAM efficiency with an intelligent scheduler

- Transactions and Request Grouping → Efficiency
- ID-based software reprogrammable scheduler → QoS
- Drop-in between existing AXI fabric and controller → Low Risk
- Instant boost of memory efficiency → TTM

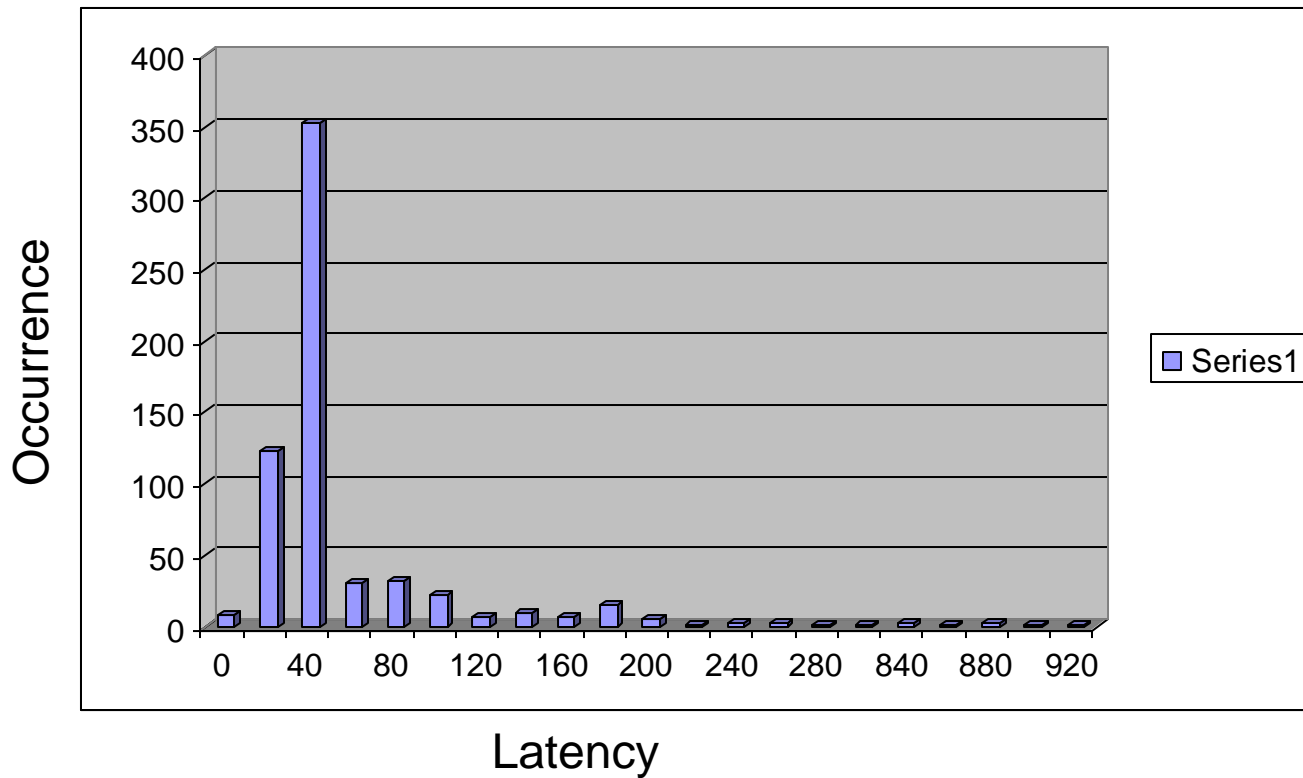


# The Impact of High Priority Queueing

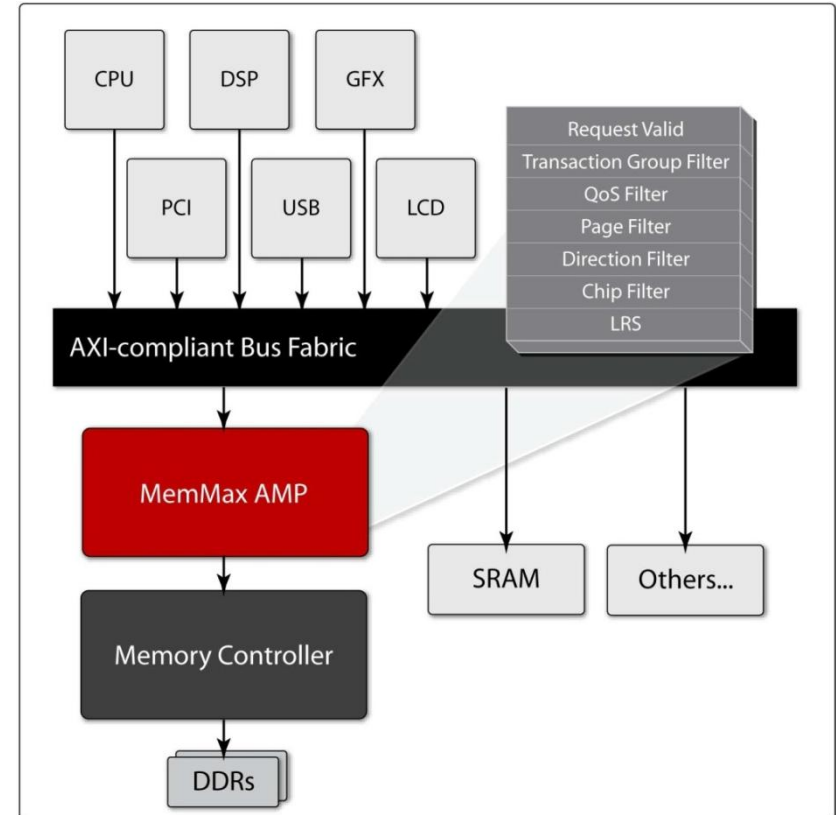
- Dedicated High Priority Queue 0/1 for CPU
- Maintains minimum average latency regardless of address patterns
  - 100% → 75% → 50% linear



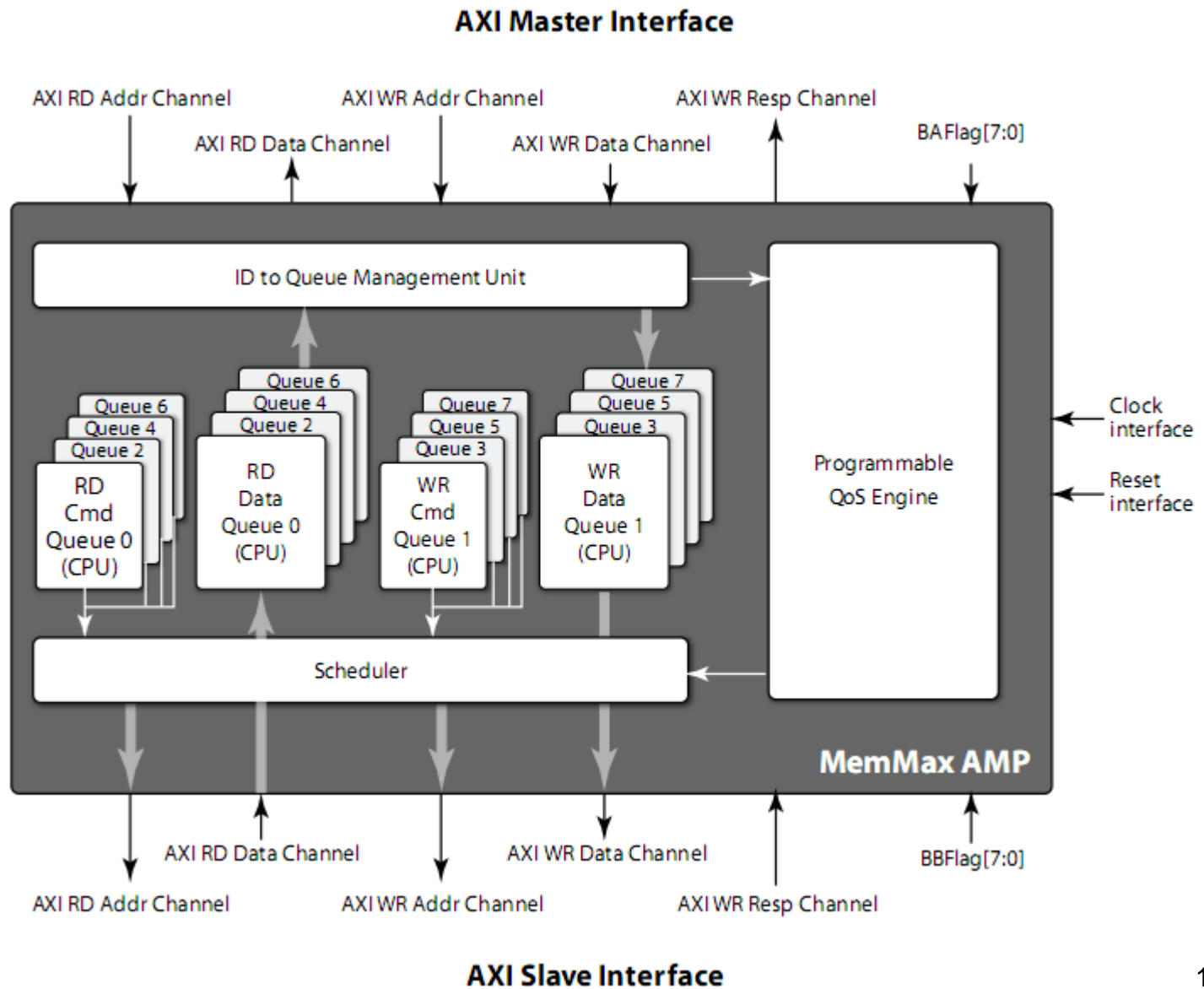
# Sample histogram for Queue 0



- AXI-compliant fabric and controller interfaces
- Memory state aware scheduler with 8 queues (4 reads and 4 writes)
- ID-based assignment among queues
- Dedicated CPU (high priority) queues
- Find the perfect balance between efficiency and QoS
  - Multi-level filtering mechanism
  - Software reprogrammable QoS level and dynamic bandwidth allocation for each queue



# MemMax AMP Internal Diagram



- Use upper 2 bits of incoming AXI ID[7:6] for queue allocations
- Separated Read (even) and Write (odd) Queues
- Queue 0/1 are reserved as high priority queues (but software changeable)
- Queue 2/3/4/5/6/7 are best effort by default and software can change QoS level and bandwidth allocation

## ■ The FACTS

- Longer the burst, better the efficiency
- Shorter the burst, better the QoS

## ■ The perfect balance between Efficiency and QoS

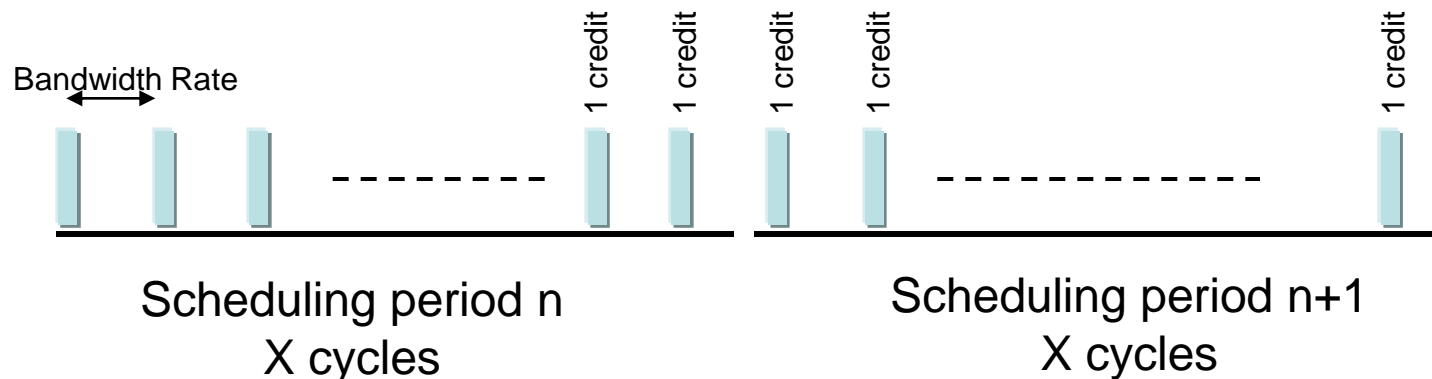
- Based on current memory state, MemMax AMP schedules new request to increase page hit rate
- When a high priority request arrives, MemMax AMP re-orders the request so it will be issued next
- Not just scheduling and re-ordering, MemMax AMP chops long system burst into many smaller DRAM bursts to create ***best interleaving*** opportunity

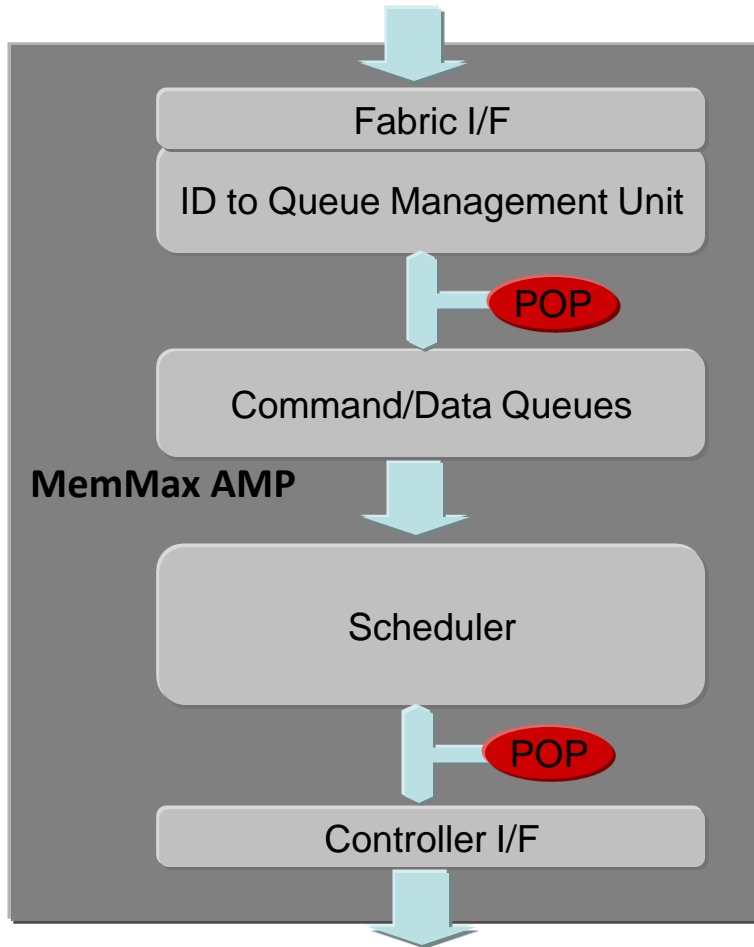
## ■ Can't decide all these at design time?

- No problem! All QoS levels and bandwidth allocation registers are software reprogrammable

## ■ Per queue register

- **Prerequisite:** Determine scheduling period (X cycles) from application perspective
- **QoS Mode:** Priority, Bandwidth or Best Effort
- **Bandwidth Rate:** How often (number of cycles) can it accumulate a one word credit?
- **Word / Scheduling Period:** The total words (credit) allowed =  $X / \text{Bandwidth Rate}$
- **Request Group Size:** Number of transactions should be grouped together if possible

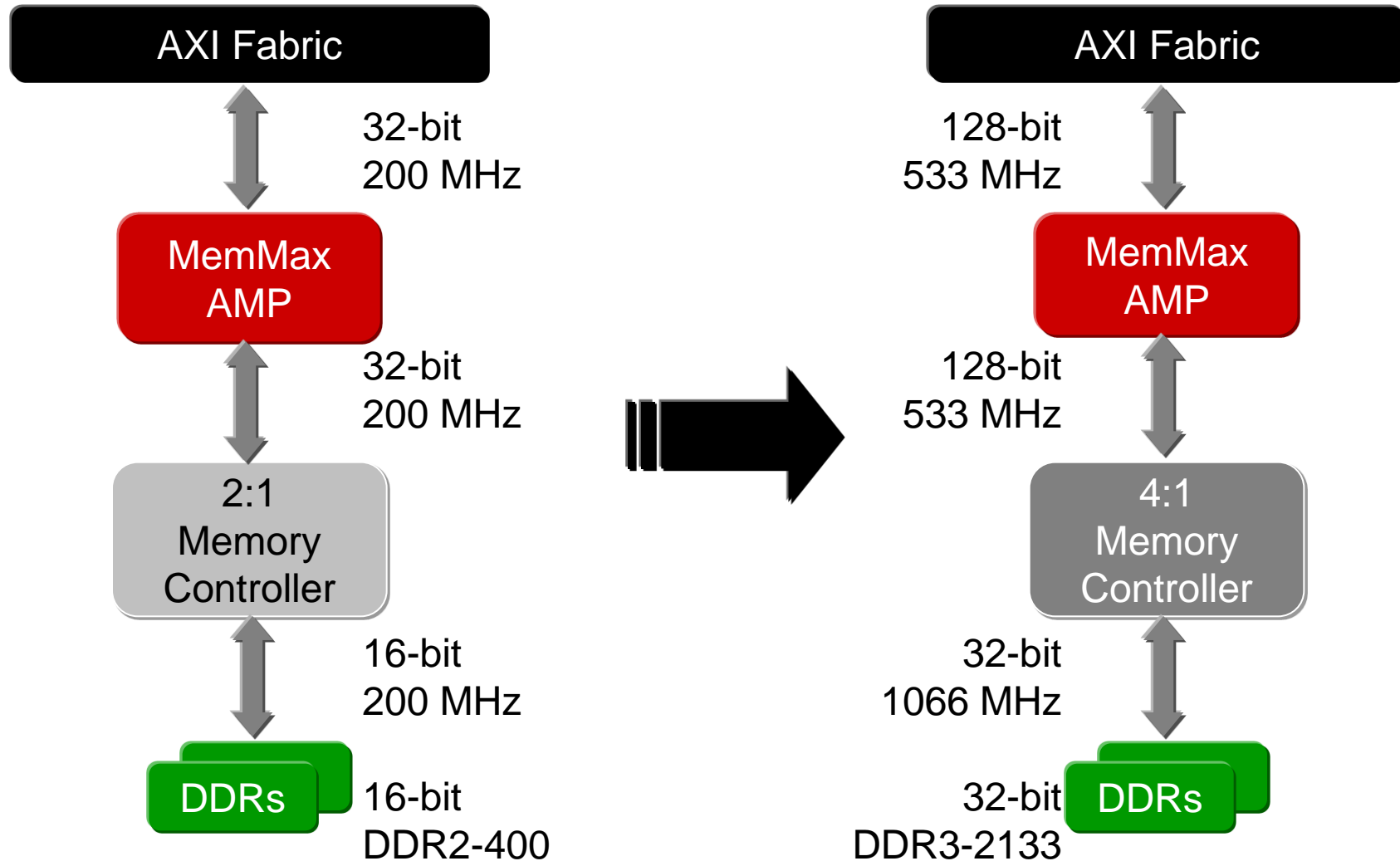




- 2 built-in POPs (fabric side & controller side)
- Provide tracing capability
- Calculate Min/Avg/Max latency
- Calculate bandwidth and efficiency

POP = Performance Observation Point

# MemMax AMP-based Solution Examples



- Amplify memory bandwidth efficiency
  - No architectural change with instant boost on memory efficiency while maintaining QoS (latency & bandwidth)
  
- Easy to adopt and shortens SoC performance optimization process
  
- Application-proven QoS implementation to ensure sensitive data traffic arrives on-time, every time