

The Challenges of System Design

Raising Performance
and Reducing Power Consumption



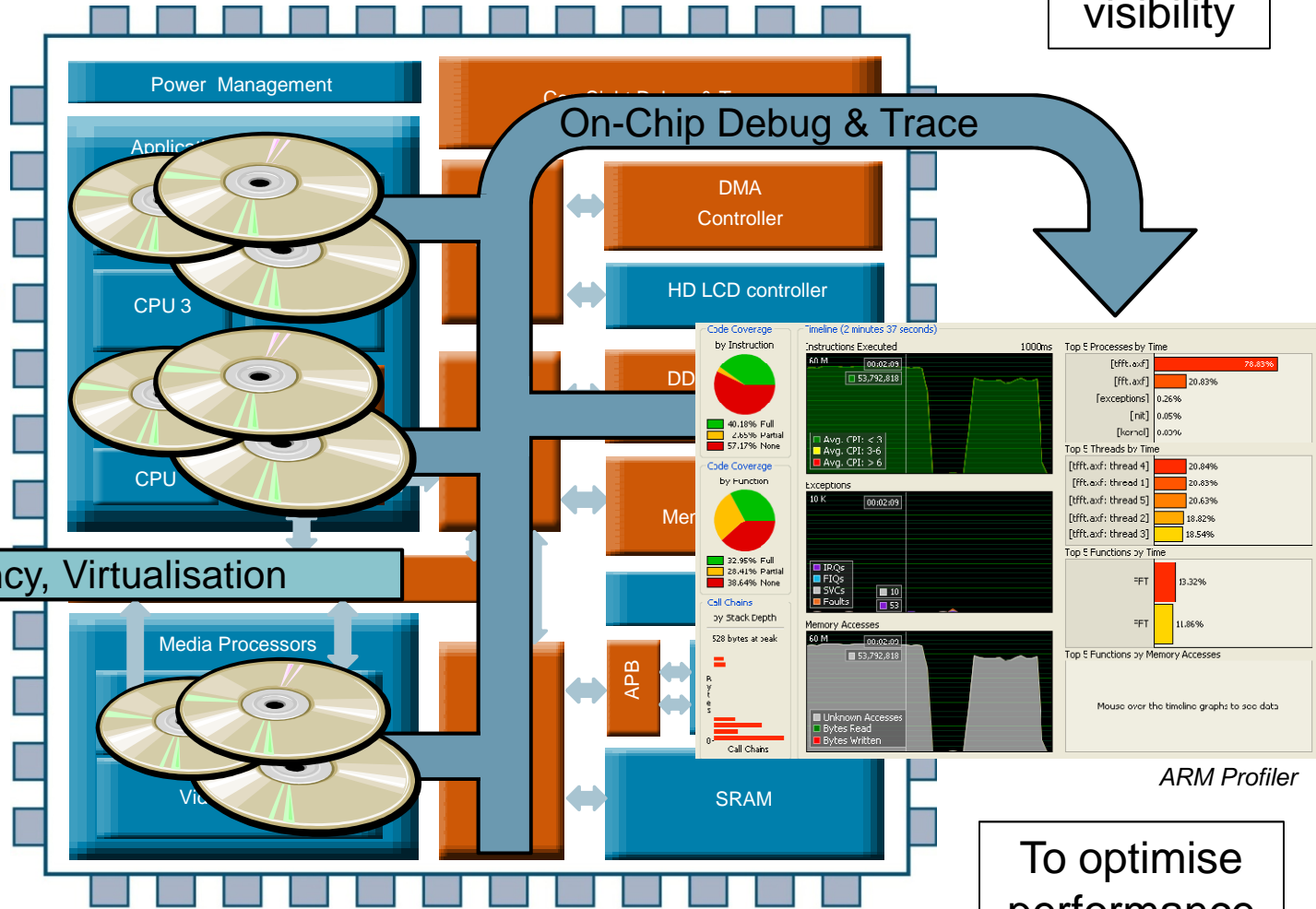
Agenda

- The key challenges
- Visibility for software optimisation
- Efficiency for improved PPA

Product Challenge - Software

For software engineers

Good visibility

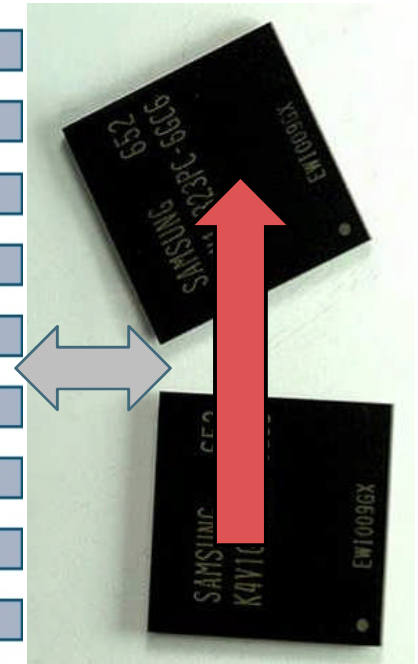
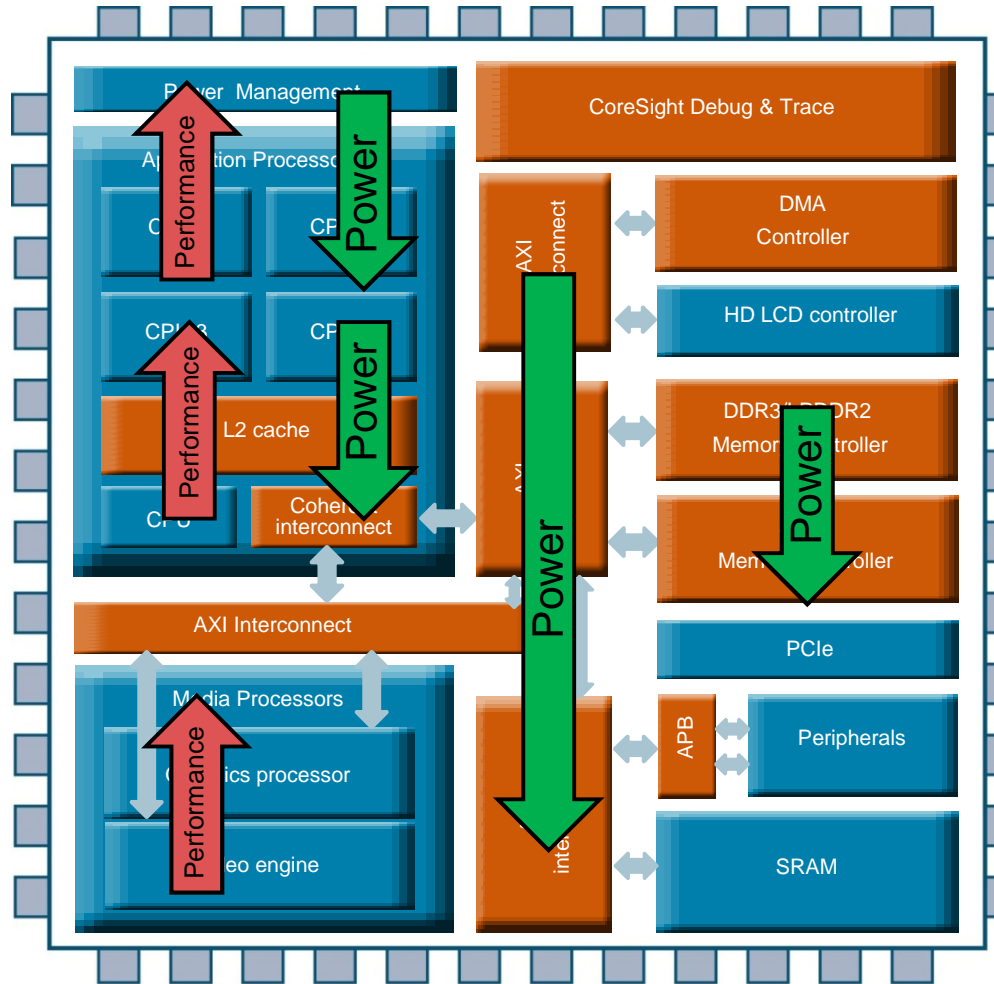


Coherency, Virtualisation

A standard easy-to-program h/w platform

To optimise performance

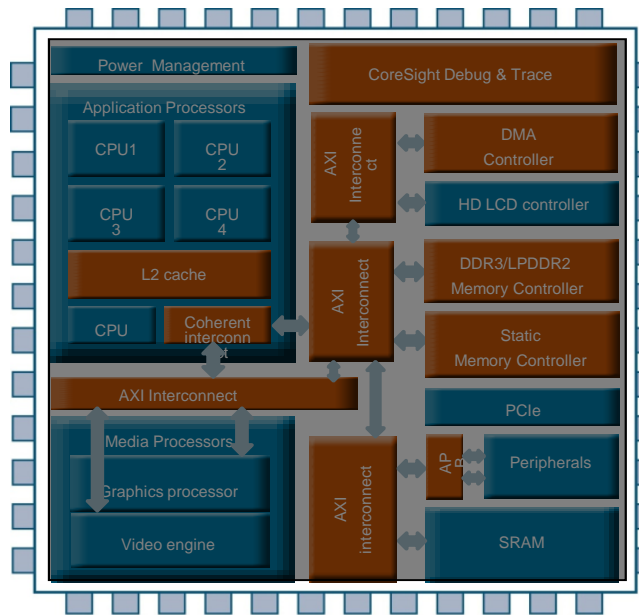
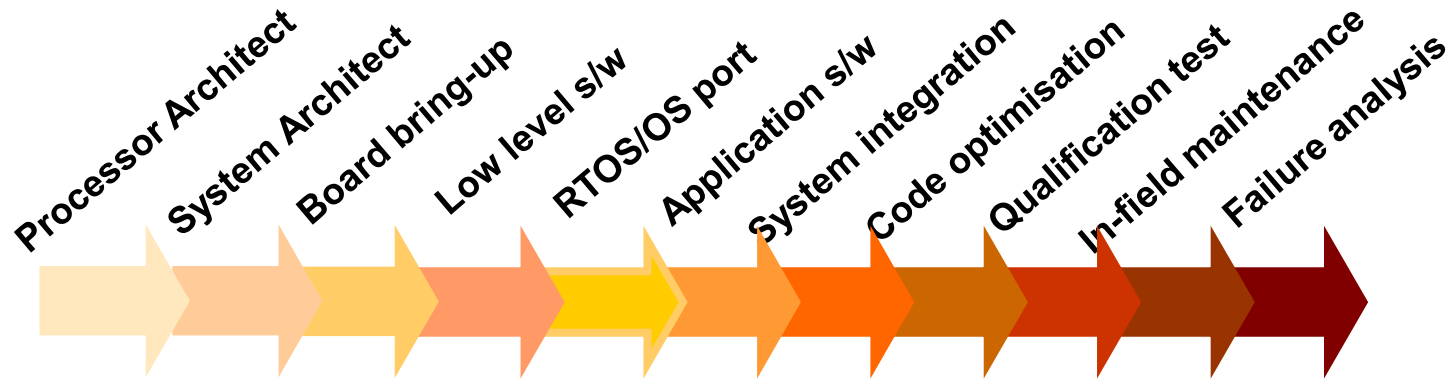
Design Challenge - PPA



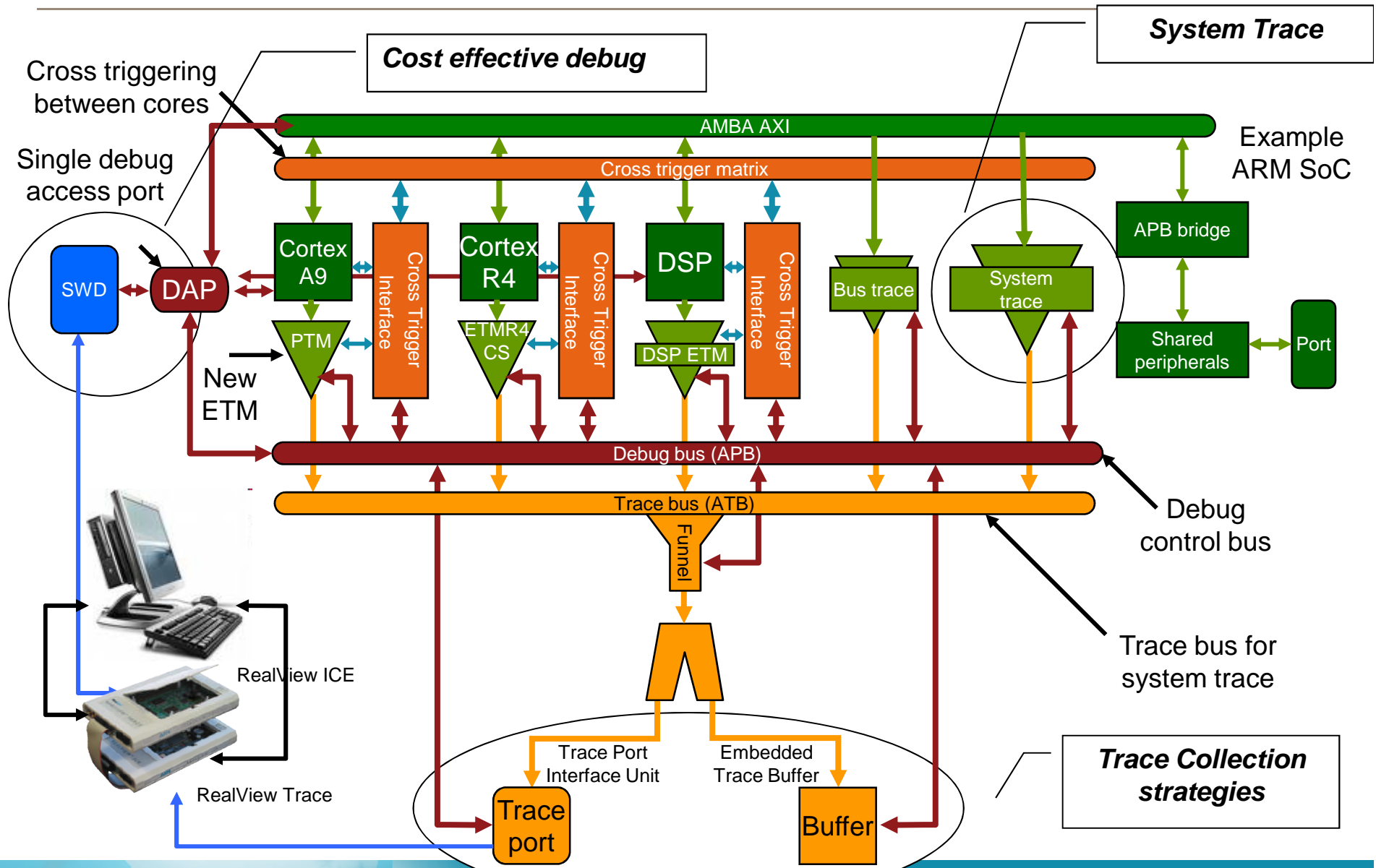
How to optimise your software and understand what your design

VISIBILITY FOR OPTIMISATION

On Chip Visibility: a key requirement



Typical CoreSight System



Software profiling using CPU Trace

- Top-down insight into the analyzed software
 - Starting with overview screen, containing top 5 functions by Self Time, Delay and Memory access



- Detailed information on the source code and its derived assembly code, annotated with performance information

Code coverage

```

84     unsigned int * scrn = (unsigned int *)buff;
85     #else
582    unsigned short * scrn = (unsigned short *)buff;
87     #endif
            
```

CC	Time	Count	Address	Opcode	Br	CPI	I	Disassembly
	291	291	0x001026d4	e59f01b4		1		LDR r0, {pc}+0x1bc ; 0x102890
	873	291	0x001026d8	e3a08000		3	•	MOV r8, #0
	291	291	0x001026dc	e590a000		1		LDR r10, [r0, #0]
	291	291	0x001026e0	e59f01ac		1		LDR r0, {pc}+0x1b4 ; 0x102894
	873	291	0x001026e4	e59f61ac		3	•	LDR r6, {pc}+0x1b4 ; 0x102898

Source associated instructions

Cycles per instruction

Interlock information

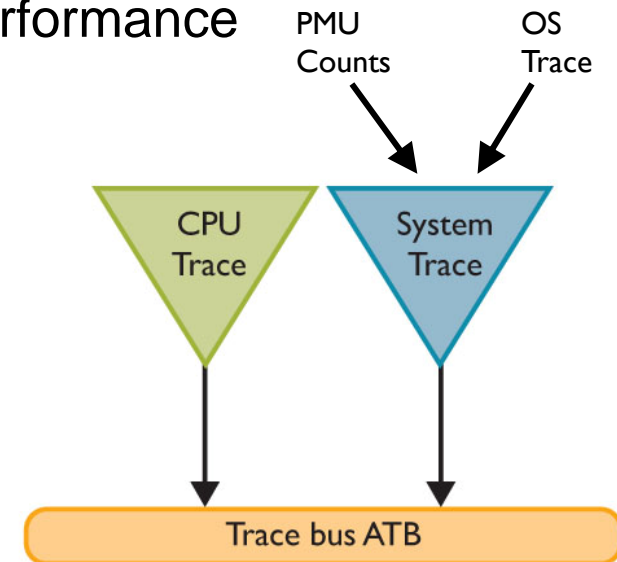
System Trace Macrocell - STM

- System level visibility required by application development up to final product
 - Debug and tuning of s/w applications running on OS
 - Tracing of system events and system performance

- System Trace Macrocell enables
 - High level application software view
 - Tuning of system performance
 - Tracing of SoC internal signals

- Benefits

- Flexible and affordable hardware based debug for applications and system level developers
- Complements CPU trace, MIPI STPv2 compliant

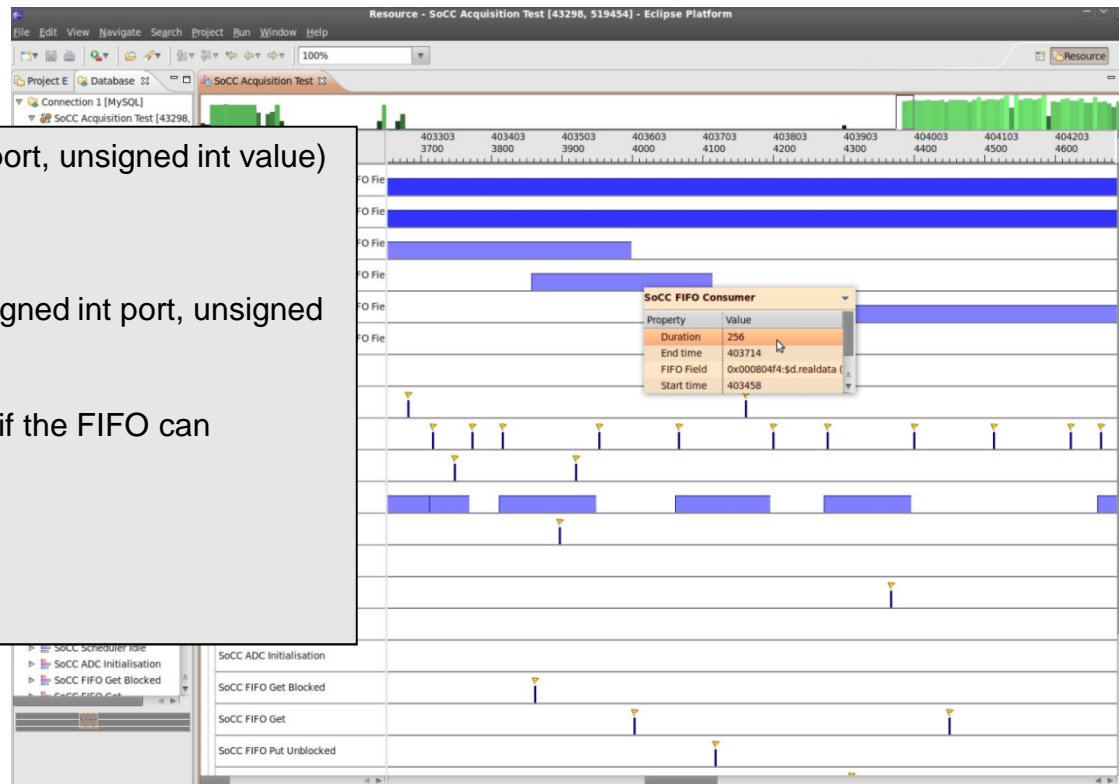


System Level Code Instrumentation

- System level debug information can be sent through trace to debug your System

```
static inline void stm_emit(unsigned int port, unsigned int value)
{
    stm_addr[port] = value;
}
static inline void stm_emit_blocking(unsigned int port, unsigned int value)
{
    // Reading from an stm port returns 1 if the FIFO can
    // accept data, 0 if it is full.
    while(!stm_addr[port]);
    stm_addr[port] = value;
}
```

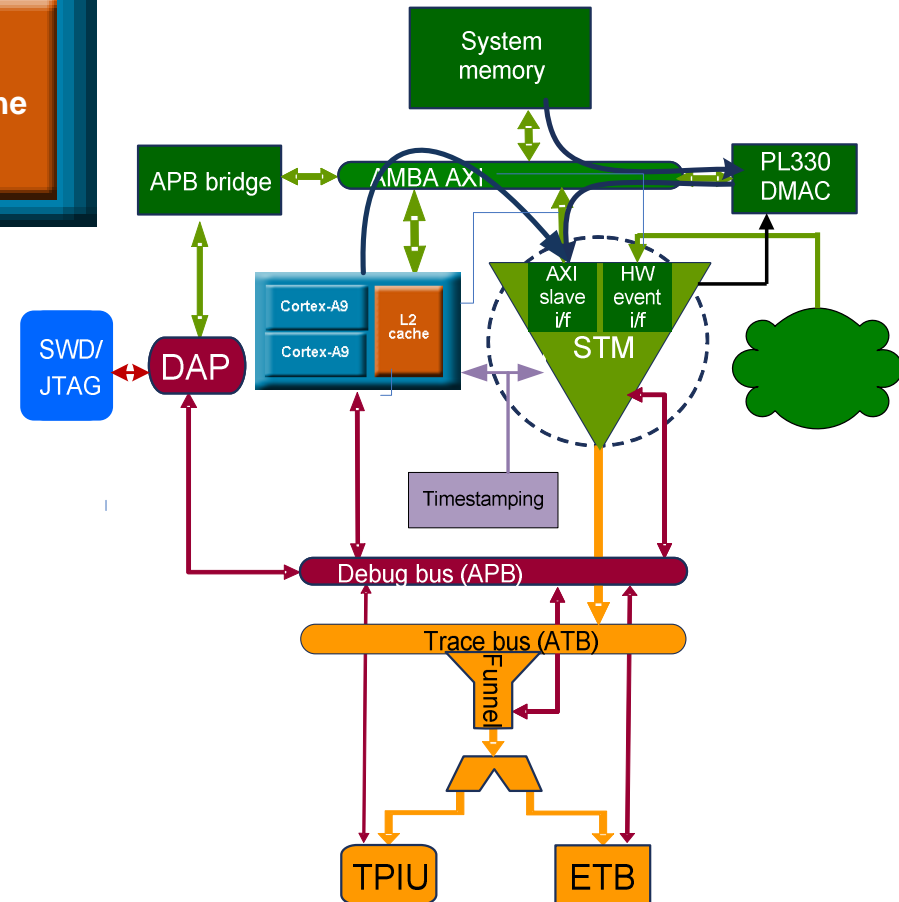
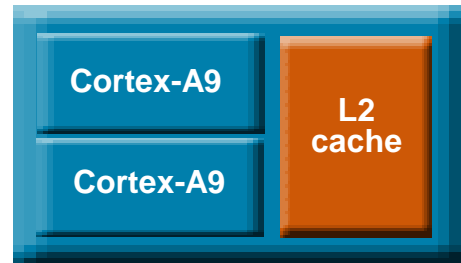
Export debug information



Visualise system level data

Event Profiling using STM

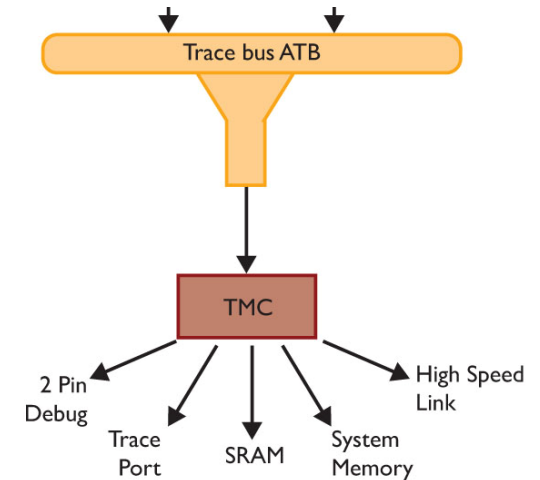
Value	Description
0x00	Software increment. The register is incremented only on writes to the Software Increment Register.
0x01	Instruction fetch that causes a refill at (at least) the lowest level(s) of instruction or unified cache.
0x02	Instruction fetch that causes a TLB refill at (at least) the lowest level of TLB.
0x03	Data Read or Write operation that causes a refill at (at least) the lowest level(s) of data or unified cache.
0x04	Data Read or Write operation that causes a cache access at (at least) the lowest level(s) of data or unified cache.
0x05	Data Read or Write operation that causes a TLB refill at (at least) the lowest level of TLB.
0x06	Data Read architecturally executed.
0x07	Data Write architecturally executed.
0x08	Instruction architecturally executed.



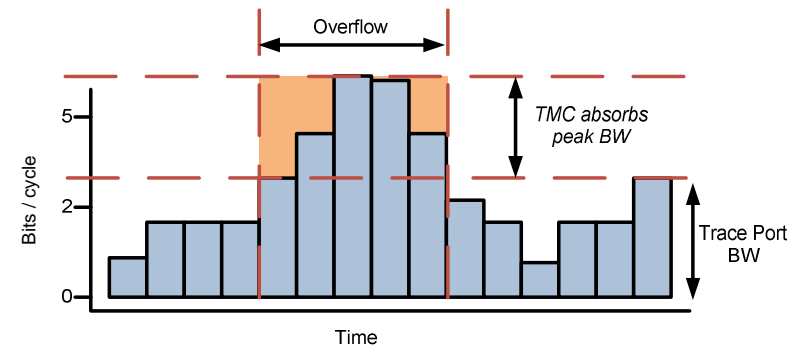
Pin	Description
CO	Eviction (CastOUT) of a line from the L2 cache.
DRHIT	Data read hit in the L2 cache.
DRREQ	Data read lookup to the L2 cache. Subsequently results in a hit or miss.
DWHIT	Data write hit in the L2 cache.
DWREQ	Data write lookup to the L2 cache. Subsequently results in a hit or miss.
DWTREQ	Data write lookup to the L2 cache with Write-Through attribute. Subsequently results in a hit or miss.
IRHIT	Instruction read hit in the L2 cache.
IRREQ	Instruction read lookup to the L2 cache. Subsequently results in a hit or miss.
SPNIDEN	Secure privileged non-invasive debug enable.
PF	Prefetch linefill sent to L3. This counts both internally generated prefetch accesses and prefetch hints.
WA	Allocation into the L2 cache caused by a write (with Write-Allocate attribute) miss.

Trace Memory Controller

- Single solution for cost effective and flexible trace collection
 - SoC visibility in final product with only 2 pins
 - Storage of trace using low cost system memory
 - Routing to Gigabit links such as HSSTP or Ethernet
 - Existing modes with ETB (SRAM) & Trace Port (TPIU)



- Reduce trace overflows and trace port size by averaging out trace bandwidth

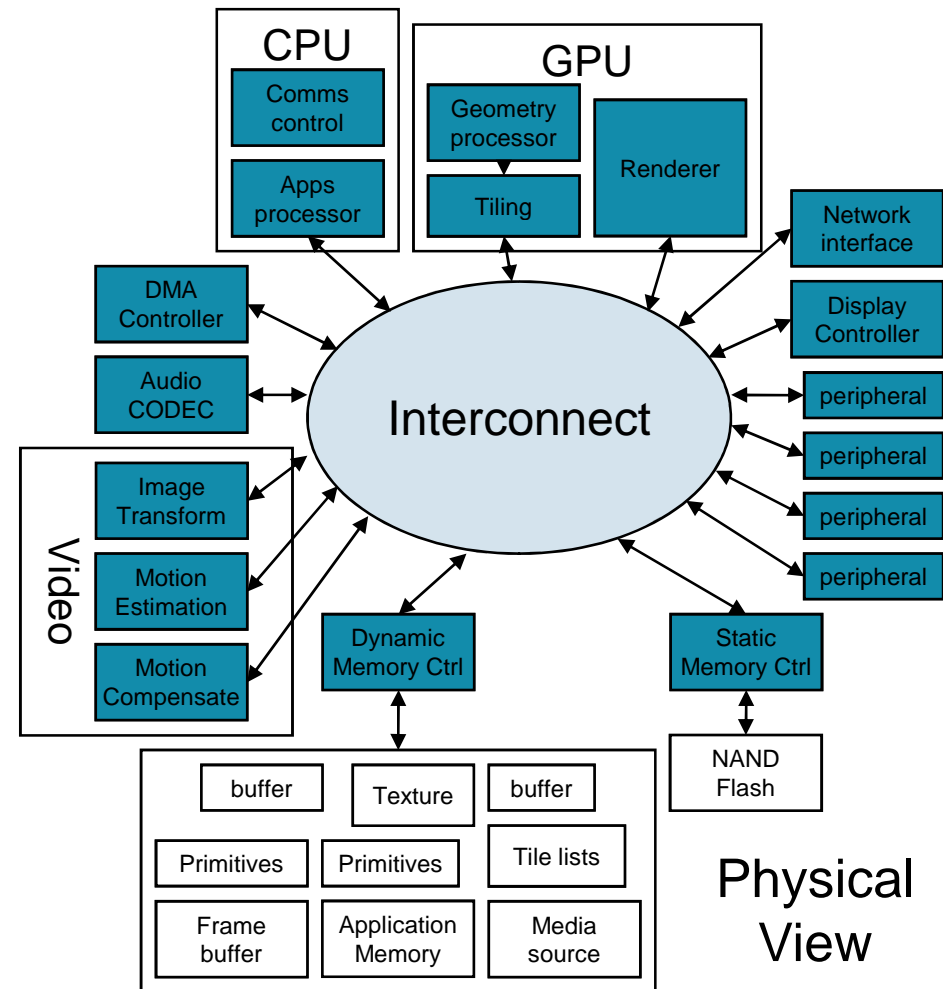


Getting the highest performance at the lowest power consumption

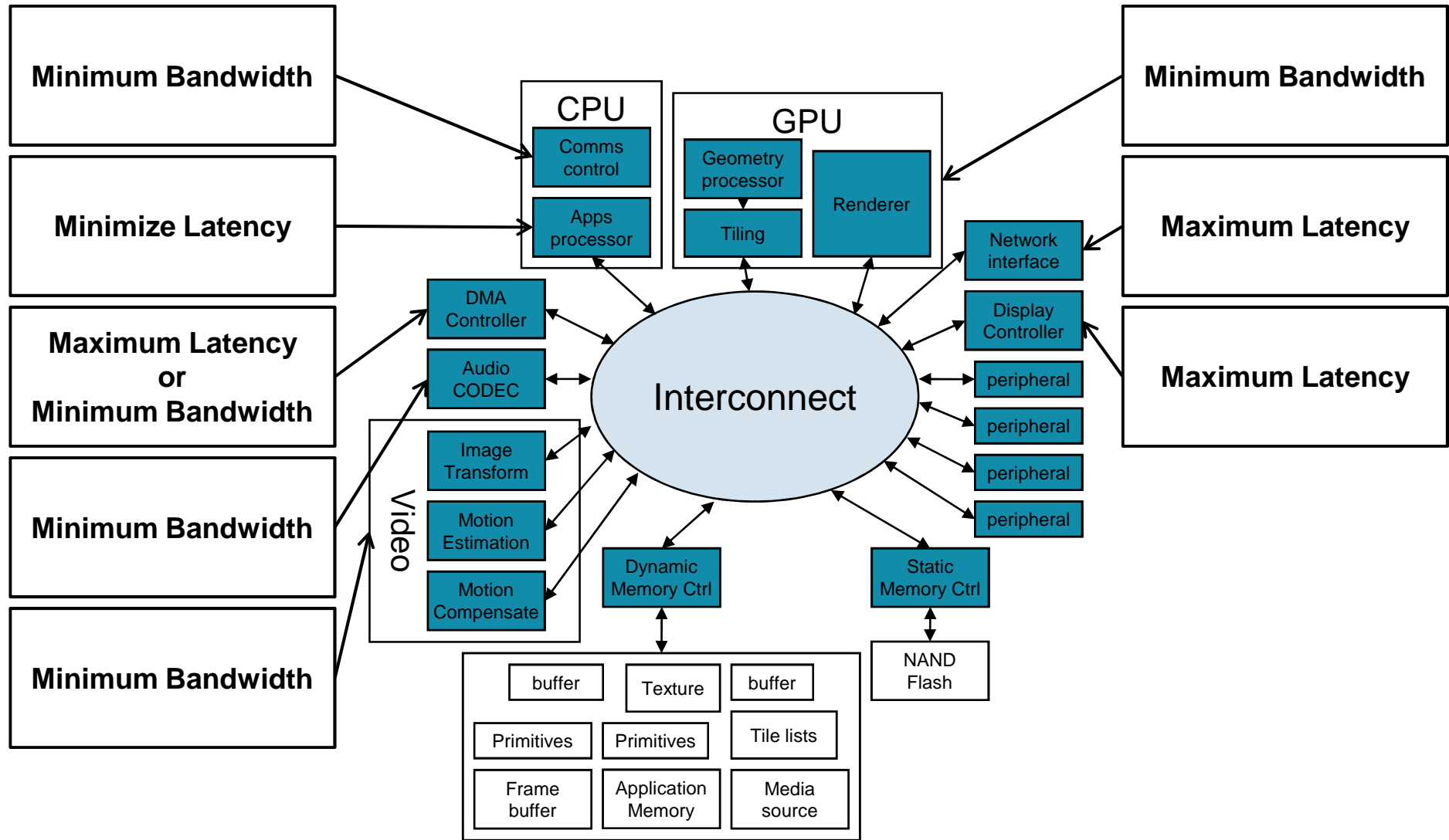
EFFICIENT SOC DESIGN

Introduction

- Systems use external memory
 - Large address space
 - Low cost-per-bit
 - Large interface bandwidth required
 - Access to memory depends on accesses from other processing elements
- Challenge:
 - Manage the flow of data to and from external memory to present the best bandwidth and latency characteristics to each processing element



QoS Contracts

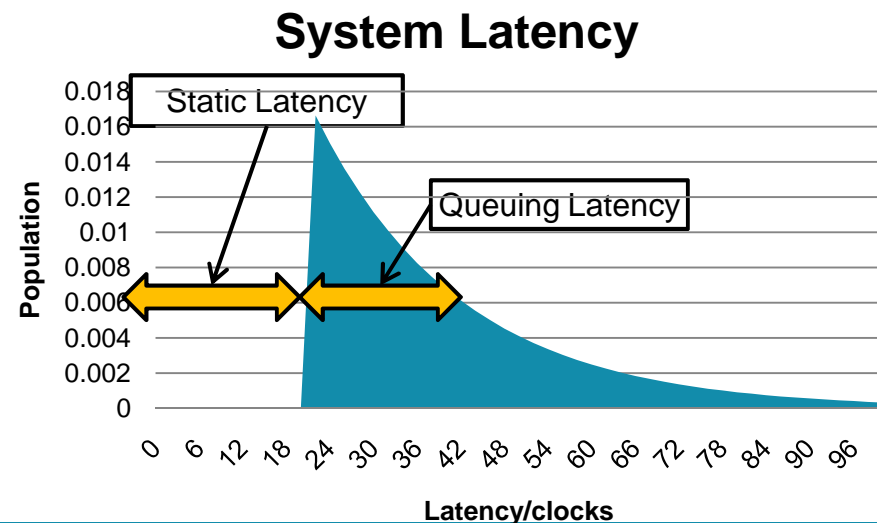
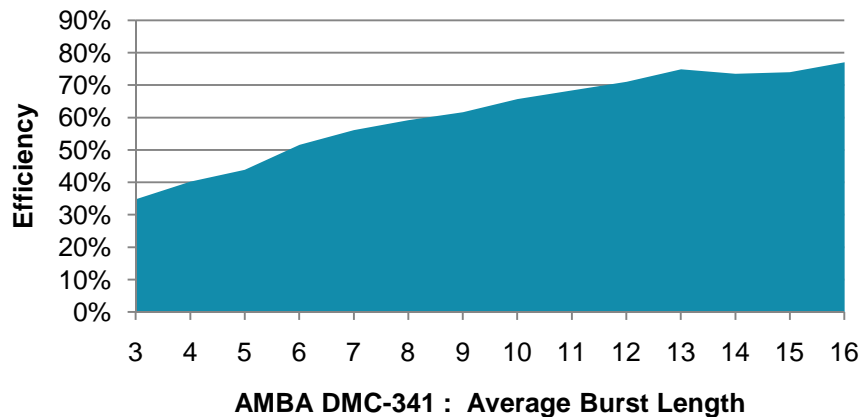


QoS Objectives

- Allocate system capacity (latency and bandwidth) to each master to meet the contract
 - Dynamically vary the priority to react to changes in bus traffic
- If there is excess capacity –
 - Allocate excess to where it can offer the most improvement
 - Usually reducing the CPU latency
 - Allocate excess to masters that can reduce performance later
- If there is insufficient capacity –
 - Remove capacity from masters that have the least impact on system performance

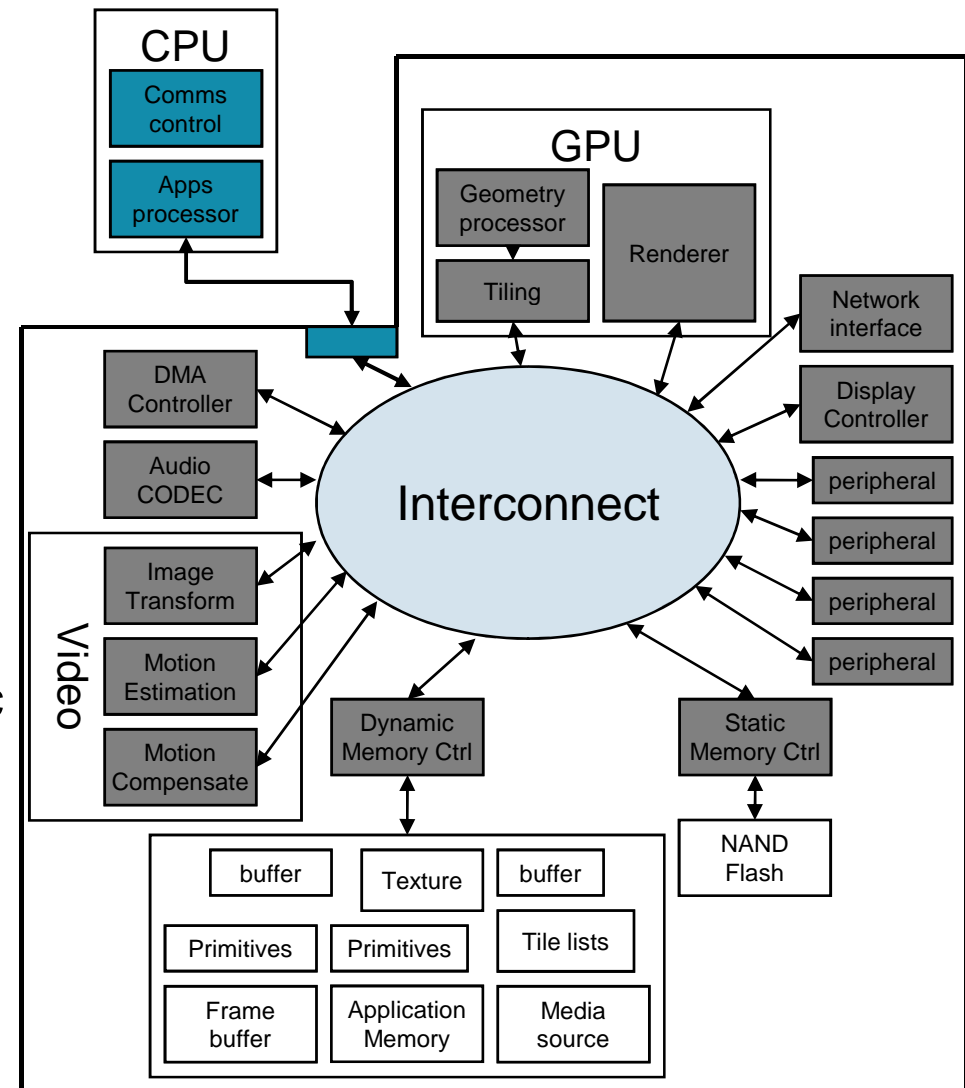
System Latency

- Latency is added throughout the system in two forms:
 - **Static latency** – the delay through pipeline stages
 - Constant and specific to the path from master to slave
 - **Queuing latency** – the delay at arbitration points in the system
 - The delay for each transaction depends on the number of transactions ahead of it in the queue and the rate at which they are processed
 - The queue length depends on the capacity of the slave (memory type and efficiency), and the desired throughput
 - Efficiency of the Memory Controller is a function of:
 - Queue length, Burst length, Read-write mix, Address distribution



System Interface Characteristics

- The performance on the master is determined by the latency characteristics that it sees from the system
 - Determined by the bandwidth from the other masters
 - Also by the efficiency of the memory controller
 - Depends on the burst characteristics of the traffic
- Other masters can be replaced with traffic profile generators (VPE)
 - Calibrated to generate the same traffic behaviour



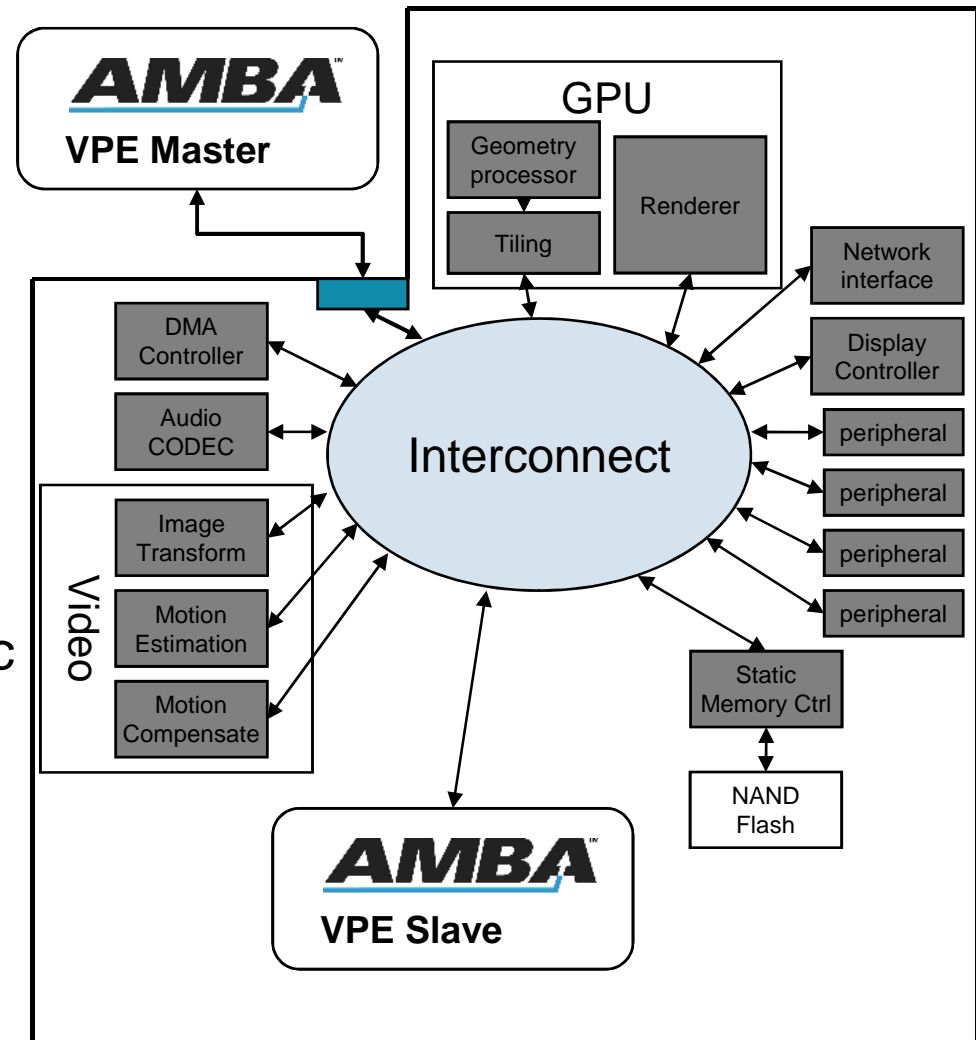
VPE – Verification and Performance Exploration



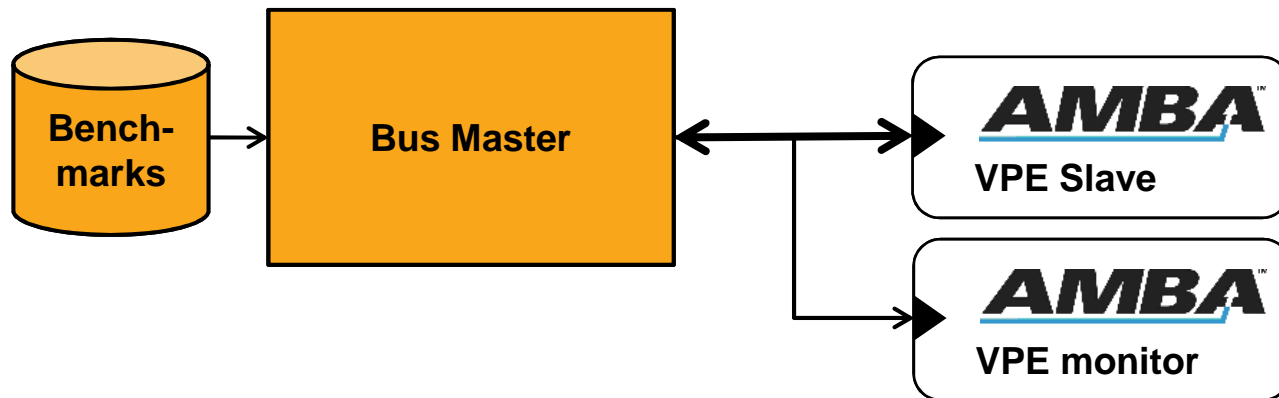
- The AMBA VPE design tool is for verification of the system performance:
 - A graphical profiling toolkit to generate & view traffic profiles
 - 3 verification components: AXI Monitor, AXI Master, AXI Slave
 - Runs on all of the big 3 RTL simulation tools
- Speeds up RTL simulation by
 - ‘Giving-up’ execution of functions (e.g. CPU, GPU) in favour of emulating their traffic
 - No need to model their cycle-accurate behaviour as a result
 - Replacing real data with constrained random data
 - Can test typical and worst case scenarios

System Interface Characteristics

- The performance on the master is determined by the latency characteristics that it sees from the system
 - Determined by the bandwidth from the other masters
 - Also by the efficiency of the memory controller
 - Depends on the burst characteristics of the traffic
- Other masters can be replaced with traffic profile generators (VPE)
 - Calibrated to generate the same traffic behaviour



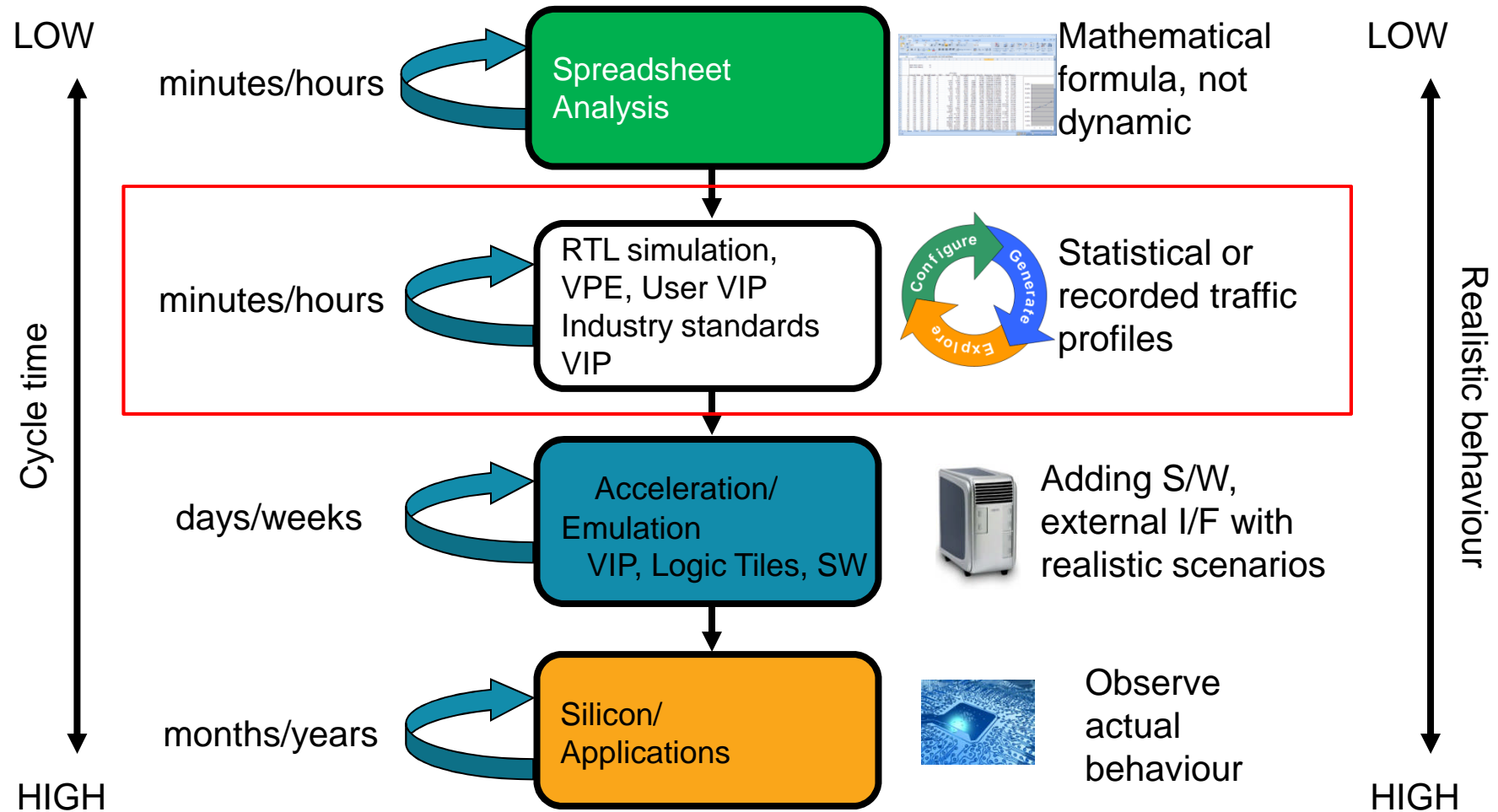
Calibrating VPE Master Behaviour



- Run benchmark applications on the bus master
 - VPE Monitor captures the traffic profile
 - VPE Slave varies the latency seen by the master
- System Architect selects a representative set of benchmarks
 - Benchmark results provide bandwidth and latency contracts
 - Traffic profile and latency sensitivity results are used to generate a VPE model of the bus master

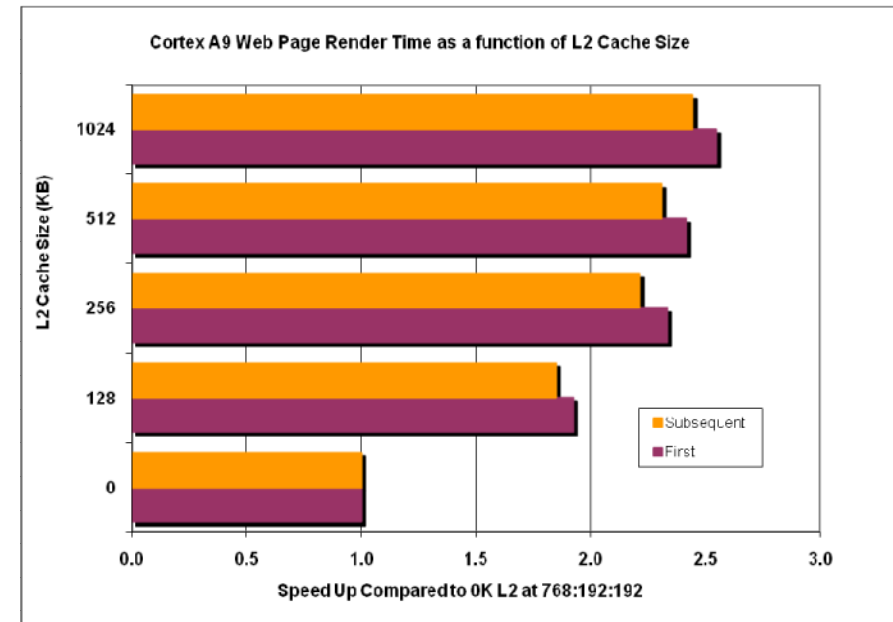
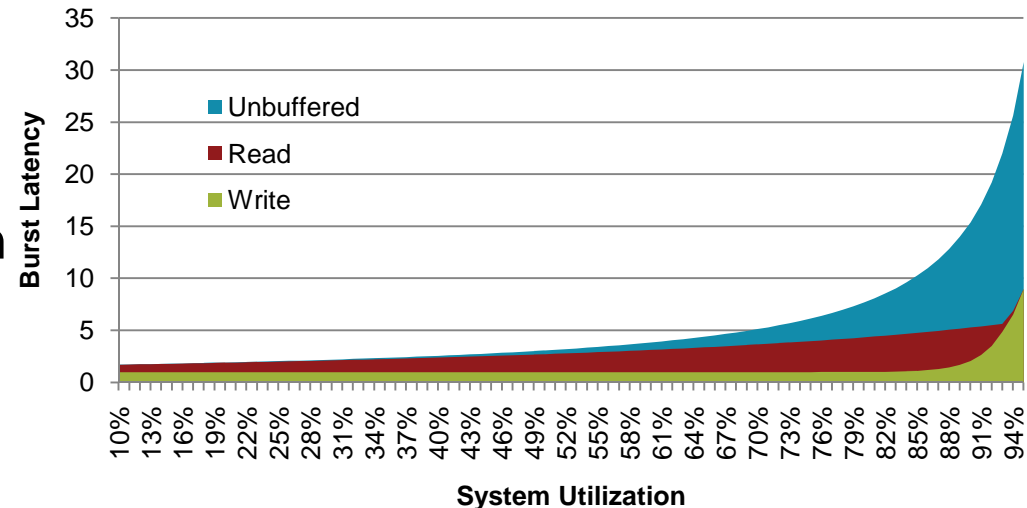
Better designs more quickly

- Iteration time of a spreadsheet with the accuracy approaching RTL simulation



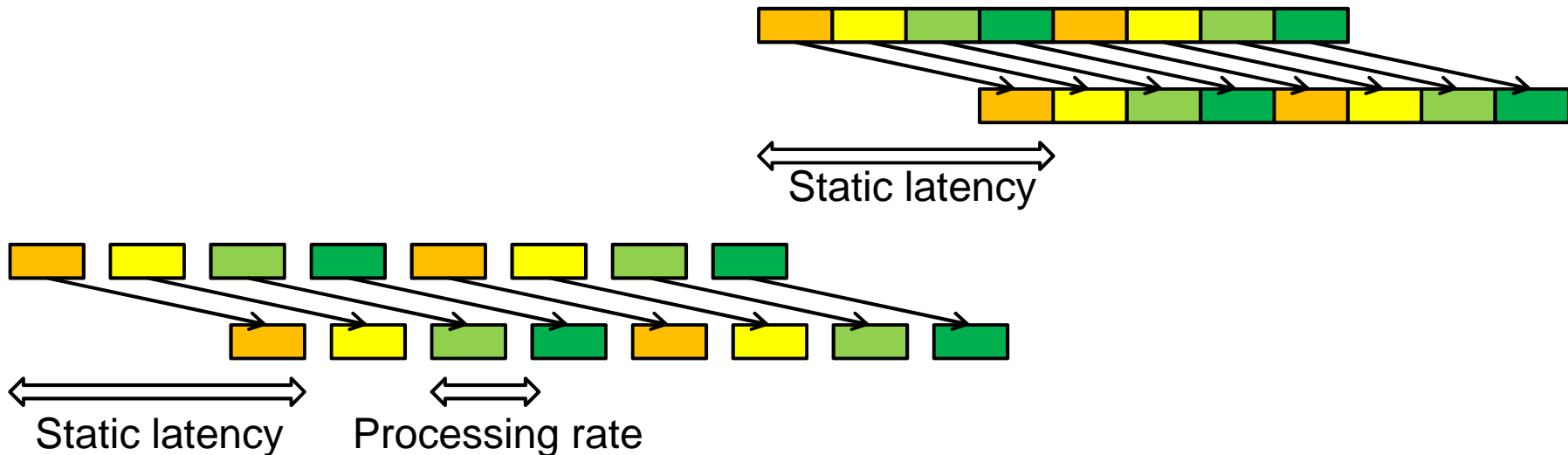
Reducing Visible System Latency

- Write data can be buffered
 - As long as coherency is managed
- The latency for write traffic seen by the system is significantly reduced
- Can be used to reduce read latency
 - Prioritize reads
- Cache memory reduces latency seen by the master
 - Also reduces system bandwidth which reduces latency to other masters
- Diminishing returns from increases in cache size



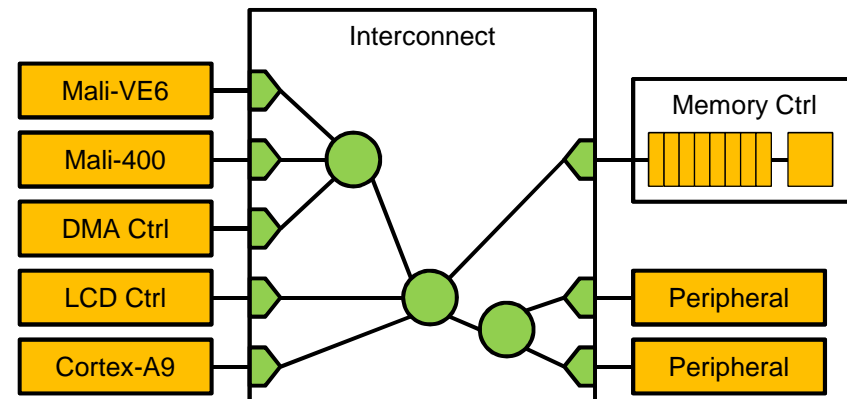
Increasing Latency Tolerance

- Masters that generate transactions that are weakly dependent on the completion of previous transactions
 - Can issue multiple outstanding transactions
- Multiple outstanding transactions can eliminate the effects of static (pipeline) latency
 - They have no impact on the effects from dynamic latency
 - Additional outstanding transactions will increase the queue length



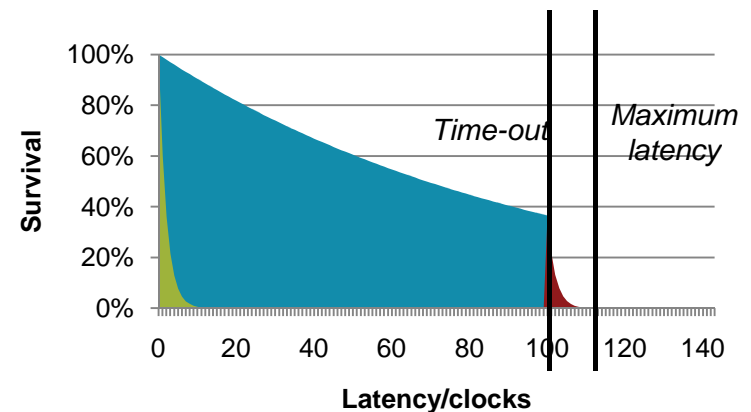
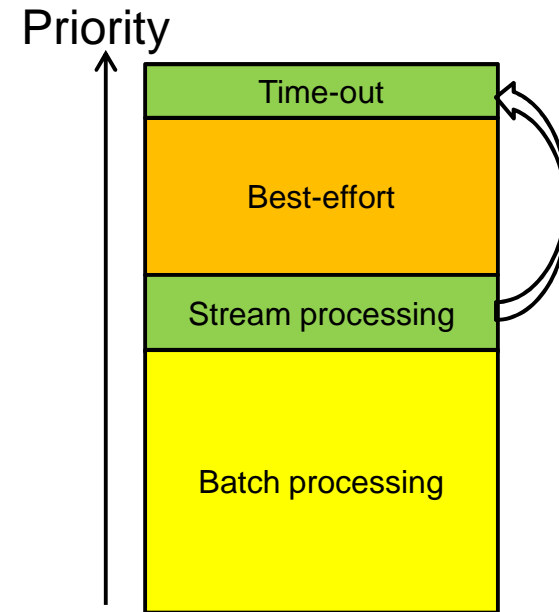
Queue location

- A queue is implemented in the memory controller
 - Allows re-ordering of transactions to maximize efficiency
 - If the queue fills it extends through the interconnect
 - Interconnect arbitration only operates when the queue extends through the interconnect
 - For effective QoS, the arbitration policy should be consistent throughout the system
- System topology influences the performance
 - CPU and LCD Ctrl placed close to the memory controller
 - Lower latency
 - Mali and DMA on a separate level
 - Hierarchy improves performance



Stream Processing Masters

- Adding latency does not affect performance
 - While latency is less than the maximum
- Entry priority set third highest
 - Reduces the latency to the Best-effort masters
- If the transaction is still waiting after a time-out period
 - Promoted to highest priority
 - Only higher priority than Best-effort masters when necessary

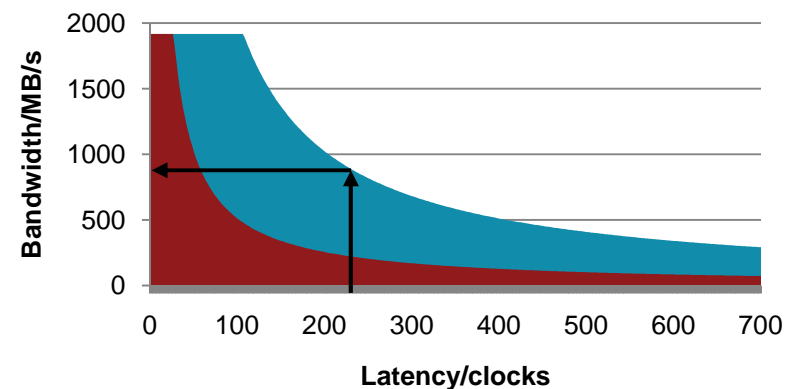
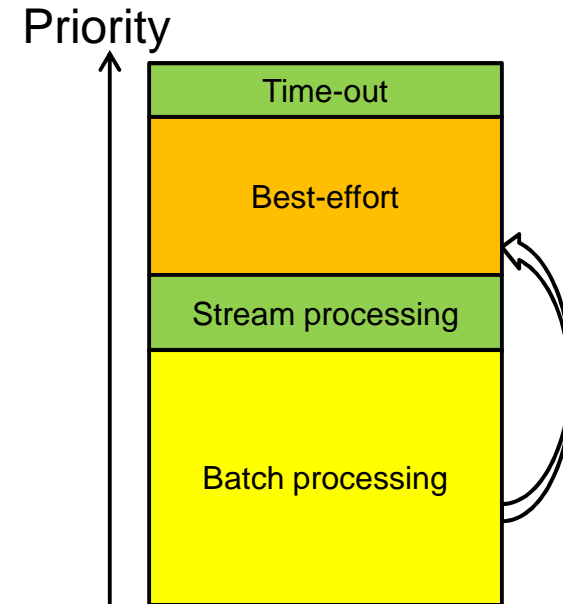


Batch Processing Masters

- Average latency, bandwidth and queue length related by Little's Law

$$E(L) = \lambda \cdot E(S)$$

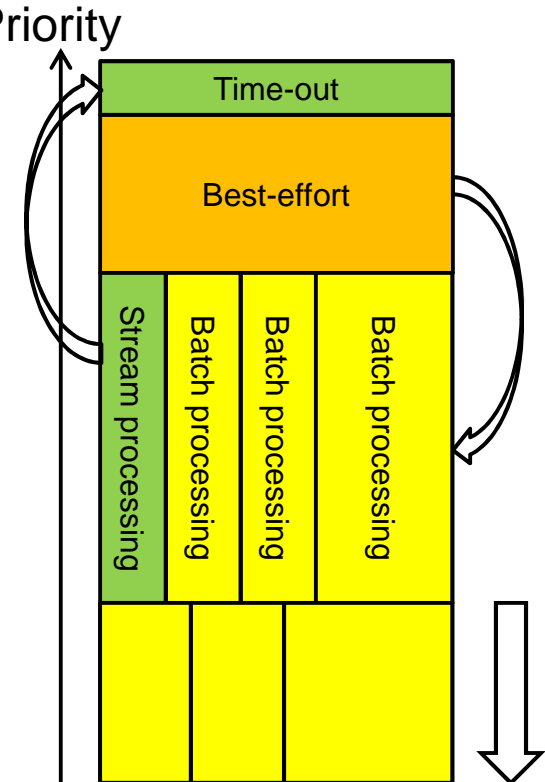
- Hold queue length constant
 - Measure average latency and control priority
 - Priority controls latency which controls bandwidth
- Excess bandwidth is used
 - Priority only exceeds other masters when
 - Insufficient bandwidth obtained
 - Minimizes transactions prioritized over Best-effort masters



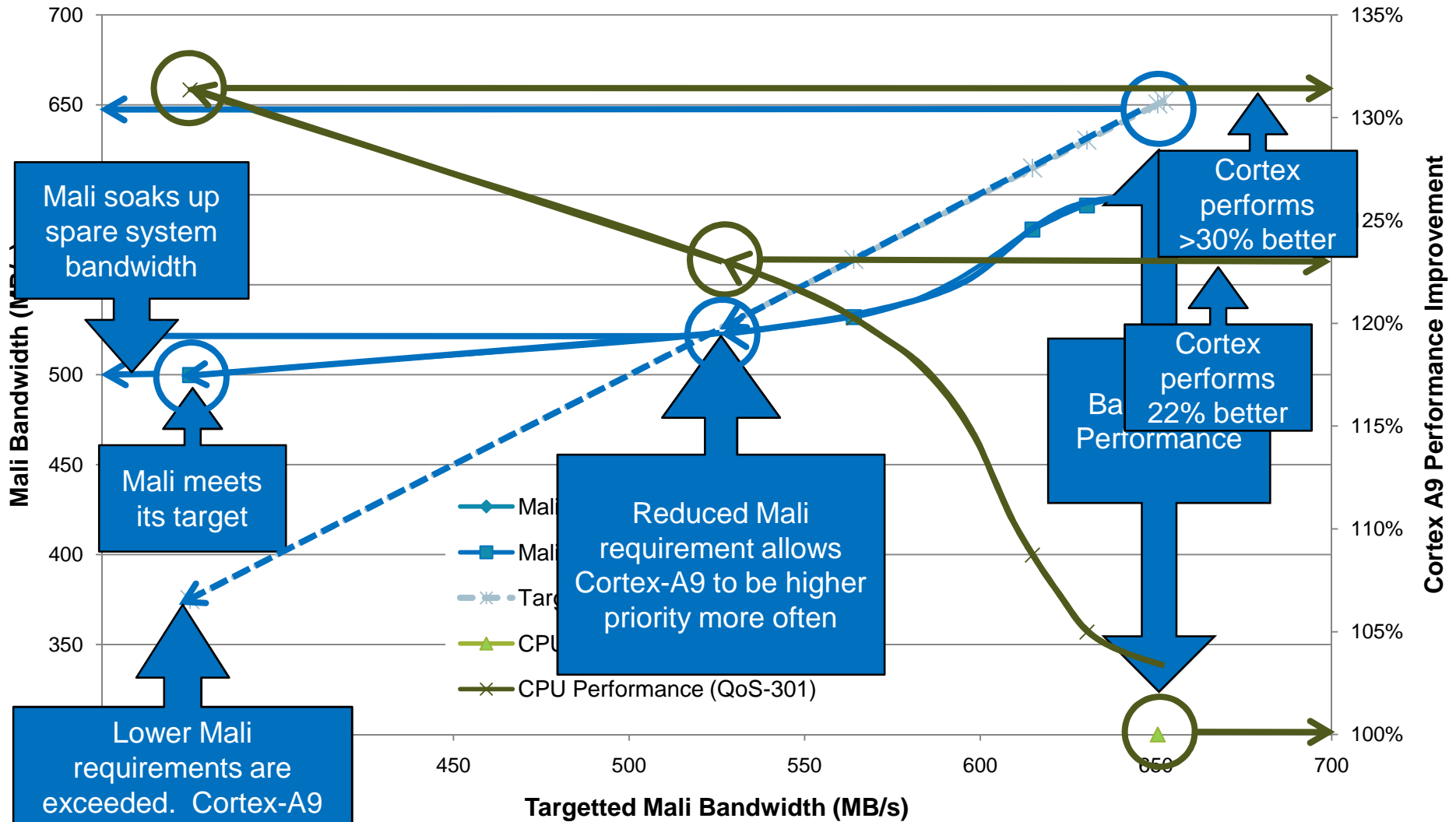
QoS with Existing Memory Controllers

- Existing memory controllers have only 3 priority levels

- CPU given high priority but demoted if there is insufficient minimum bandwidth available for the batch processors
- Batch processor bandwidth is partitioned by varying the number of outstanding transactions
- Batch processing bandwidth increases to use any available bandwidth
 - Increase proportional to outstanding transactions
 - Hard regulation can set a maximum bandwidth for a batch processing master
 - Excess bandwidth partitioned between other masters



30% Browsing Boost with QoS-301



Mali soaks up spare system bandwidth

Mali meets its target

Lower Mali requirements are exceeded. Cortex-A9 is highest priority most of the time

Reduced Mali requirement allows Cortex-A9 to be higher priority more often

Cortex performs >30% better

Cortex performs 22% better

Baseline Performance

Optimizing Efficiency

- The performance of a system depends on
 - Maximizing the efficiency from the memory controller
 - Using Cache to minimize the system bandwidth
 - And reduce latency to the masters
 - Using write buffering to minimize the latency from the system
- Performance is optimized by
 - Implementing a consistent arbitration policy throughout the system
 - Exploiting the different latency sensitivities of masters
- Roadmap to QoS
 - Consistent, system-wide, priority-based arbitration policy
 - Priority controllers for the system masters
 - Time-out mechanism in the system queue
 - High efficiency memory controller with write buffering

