2015 Best Paper Award

Using the On-Chip Network to Optimize Multichannel DRAM Throughput

Drew Wingard
CTO, Sonics, Inc.

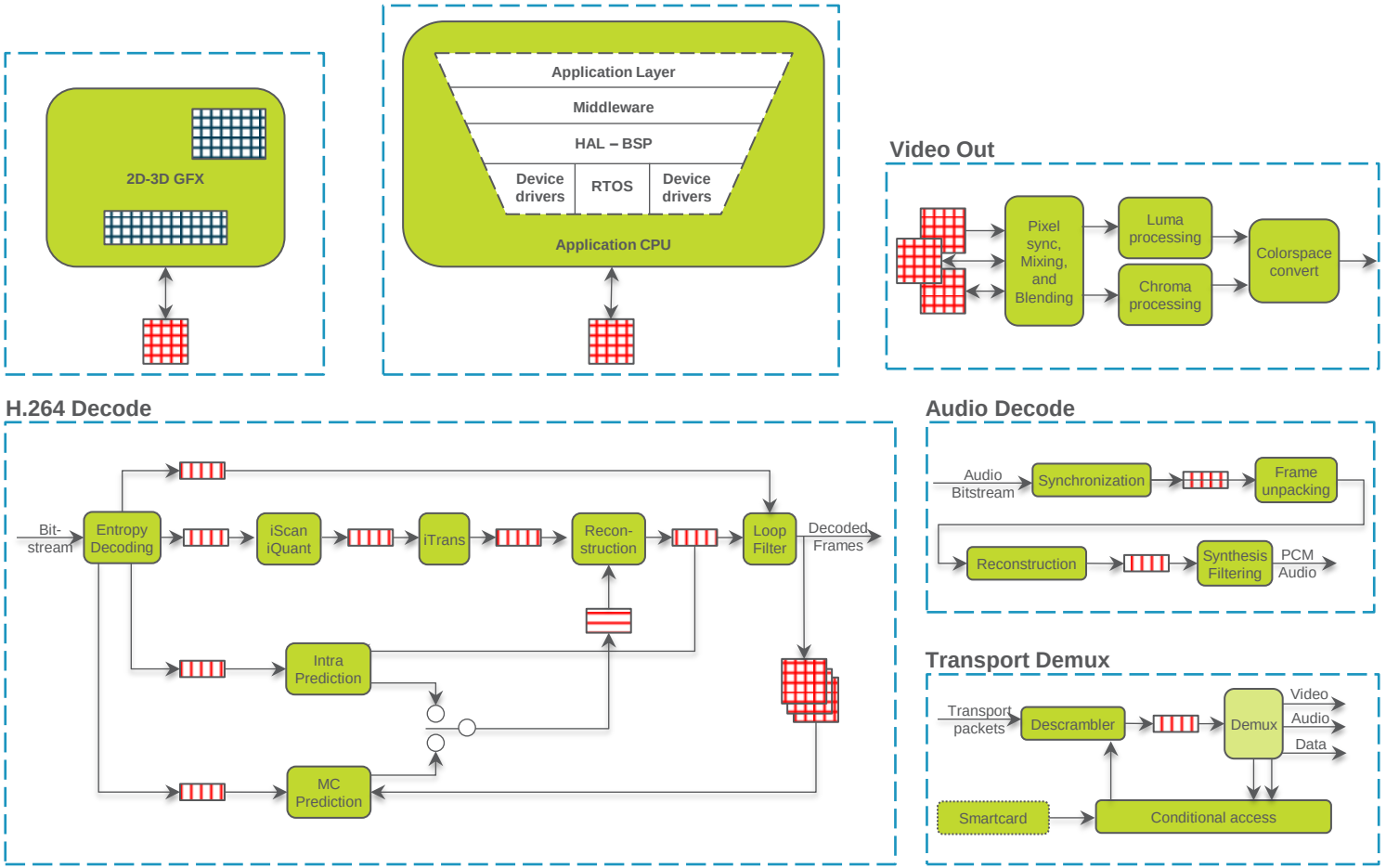
Why SoCs Are Difficult

- Improvement by feature integration has been practiced for decades
  - Why is SoC any different?
- Benefits of an SoC
  - Higher performance
  - Smaller footprint
  - Lower power
  - Lower cost
- Size, power, and cost benefits derive from ***sharing***
  - Control software on CPU (interrupts, etc.)
  - Memory resources (on-chip and off-chip) – ***especially DRAM***
- Building predictable systems with so much sharing is hard
  - Dozens to hundreds of interrupt sources
  - Memory bandwidth bottlenecks + lots of real-time traffic stress Unified Memory Architecture (UMA)

Traffic Concurrency in Consumer SoCs

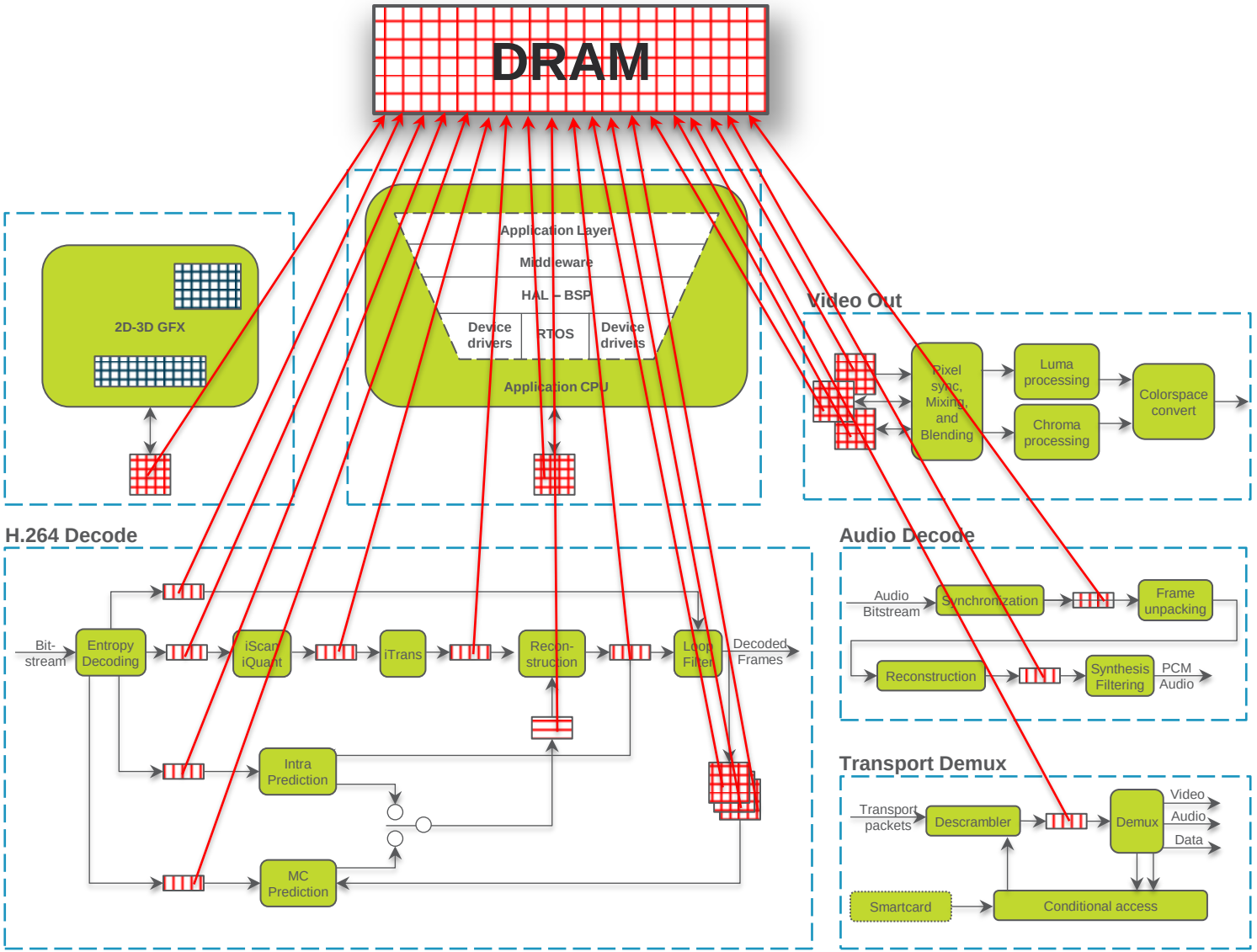
- Assertion: consumer SoC applications have >> 50% of system traffic to/from external DRAM
- Consumer volumes and price points demand ***cheapest DRAM configurations*** that support required performance
- Implications:
  - SoC architecture is mostly a fan-in tree to external DRAM + CPU-focused control network
  - Maximizing delivered DRAM ***throughput*** and ***utilization*** are key

Traffic Concurrency in Consumer SoCs



SONICS

Traffic Concurrency in Consumer SoCs



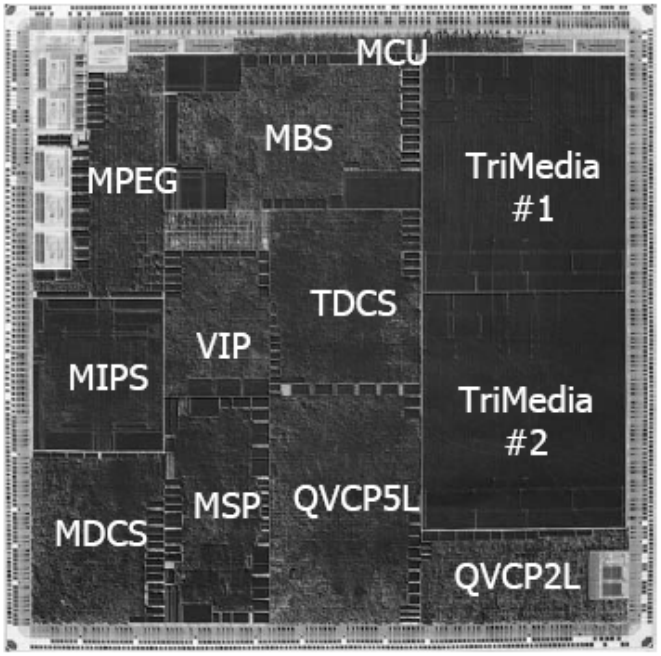
SONICS

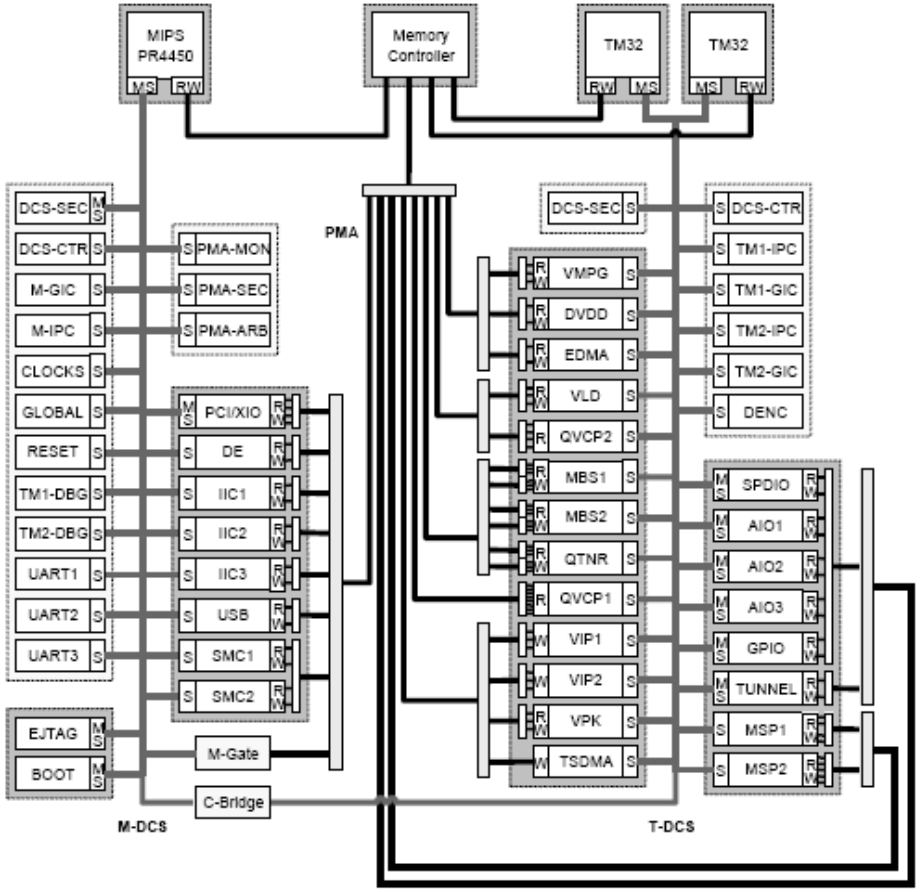
putting it all together

nexperia

Viper2 (PNX8550)

- 0.13 μm
- ~50 M transistors
- ~100 clock domains
- more than 70 IP blocks

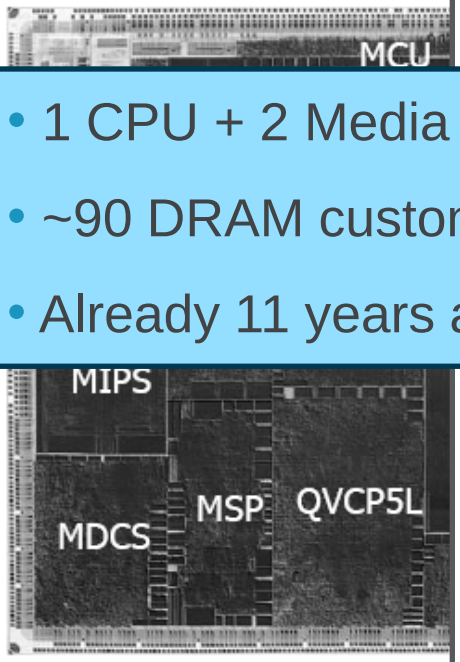




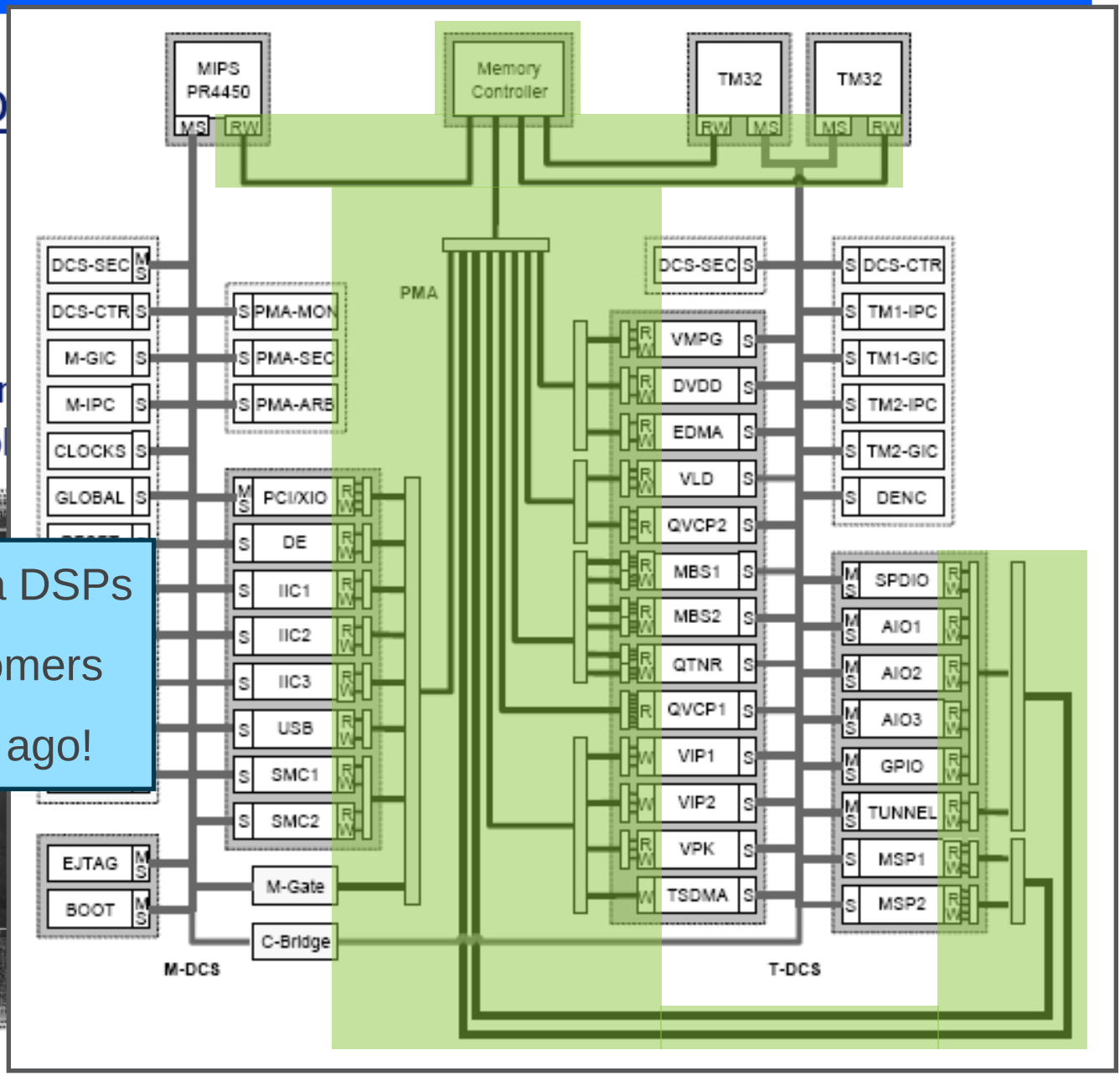
putting it all to

Viper2 (PNX8550)

- 0.13 µm
- ~50 M transistors
- ~100 clock domain
- more than 70 IP bl



- 1 CPU + 2 Media DSPs
- ~90 DRAM customers
- Already 11 years ago!



SoC DRAM Problems and Opportunities

- Problems in current systems
  - In many SoCs, DRAM ***bandwidth*** needs grow faster than ***capacity*** needs
  - Scaling DRAM bandwidth requires ***extra DRAMs***
    - And power-hungry PHYs
  - Wider DRAM interfaces & deeper pipelining increases access ***granularity***, driving need for ***multichannel*** approaches
    - Which causes extra pin costs (after 2 channels)
  - Most DRAM interfaces are a ***bottleneck*** between
    - Lots of parallel initiators (data clients)
    - Lots of parallel DRAM banks (data servers)
- What is the DRAM community doing to address?
  - From smartphones to the data center…

Increasing DRAM Bandwidth

	LPDDR4	Wide IO2	HBM	HMC
Channels/unit	2	4-8	8	32
Bandwidth/unit (GB/s)	26-34	51-68	128-256	240-320
Minimum Efficient Request (B)	32	32	32	32

Breaking through the interface bottleneck

- Increasing channel count
  - Required to control request size
- Increasing reliance on 2.5D/3D integration
  - Required to manage interface costs and I/O power

… But what does this do to the SoC design?

Solution Requirements for Multichannel SoCs

- Optimization for high DRAM ***utilization*** and good Quality of Service (***QoS***) characteristics
- ***Load balancing*** traffic from the increasing numbers of DRAM consumers across increasing numbers of DRAM channels
- ***Eliminating performance loss*** due to ordering dependencies while ***minimizing buffering***
- ***Analysis tooling*** to understand design trade-offs

Solution Requirements for Multichannel SoCs

- Optimization for high DRAM ***utilization*** and good Quality of Service (***QoS***) characteristics
- ***Load balancing*** traffic from the increasing numbers of DRAM consumers across increasing numbers of DRAM channels
- ***Eliminating performance loss*** due to ordering dependencies while ***minimizing buffering***
- ***Analysis tooling*** to understand design trade-offs

How Traffic Impacts DRAM Utilization

Traffic Characteristic / Utilization Impact	Refresh cycles	RD/WR turnaround	Page misses	Partial bursts	Unaligned bursts	Command conflicts	QoS optimizations
Burst lengths/sequences/alignment		X	X	X	X	X	
Addressing across transactions		X	X			X	X
# of outstanding transactions		X	X			X	X
Read/write mix		X					
Ordering constraints		X	X			X	X
Time-domain behavior		X	X			X	X

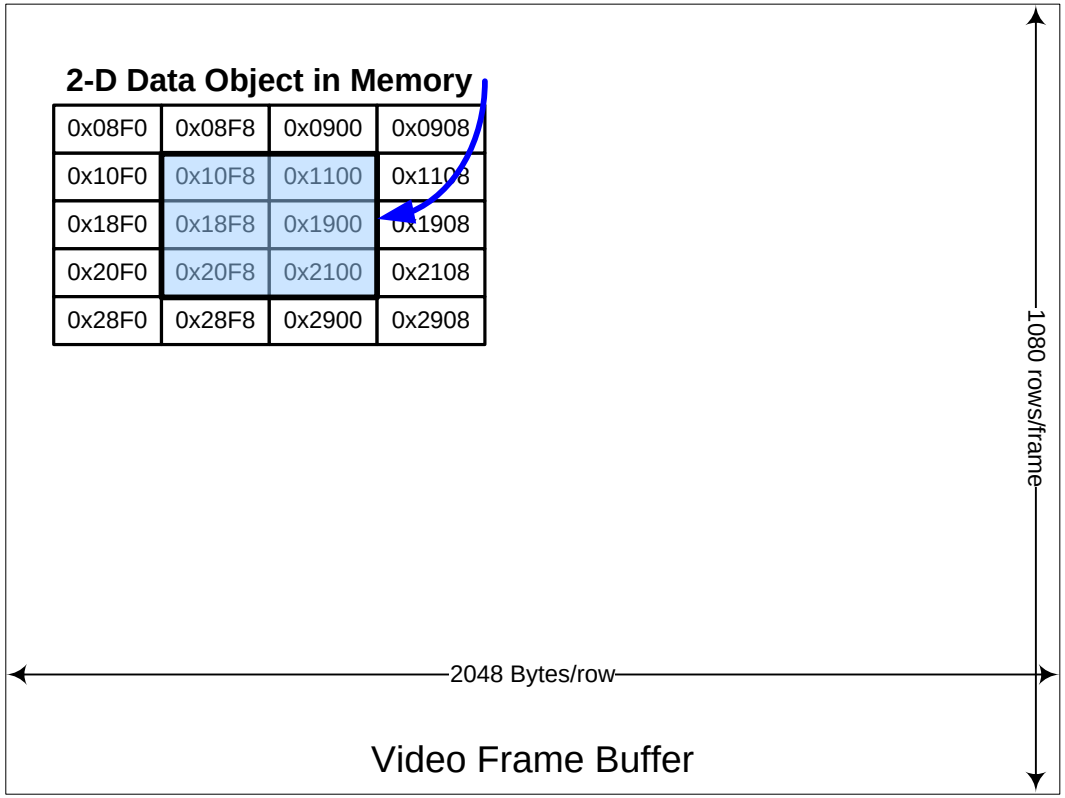
Assumes refresh is independent from traffic

High DRAM Utilization vs. QoS

- High DRAM utilization (throughput) and Quality of Service are in conflict
  - ***Utilization*** prefers long DRAM bursts
    - DRAM operates most efficiently
  - ***QoS*** demands short DRAM bursts
    - Provide low latency service for CPUs
    - Control buffering requirements for real-time users
- Consumer SoCs use the DRAM scheduler to tune the trade-off between utilization and QoS
- Scheduler requirements
  - Collect enough requests to understand choices
  - Understand QoS requirements for each request
  - Consider DRAM bank/page/direction states to choose winner

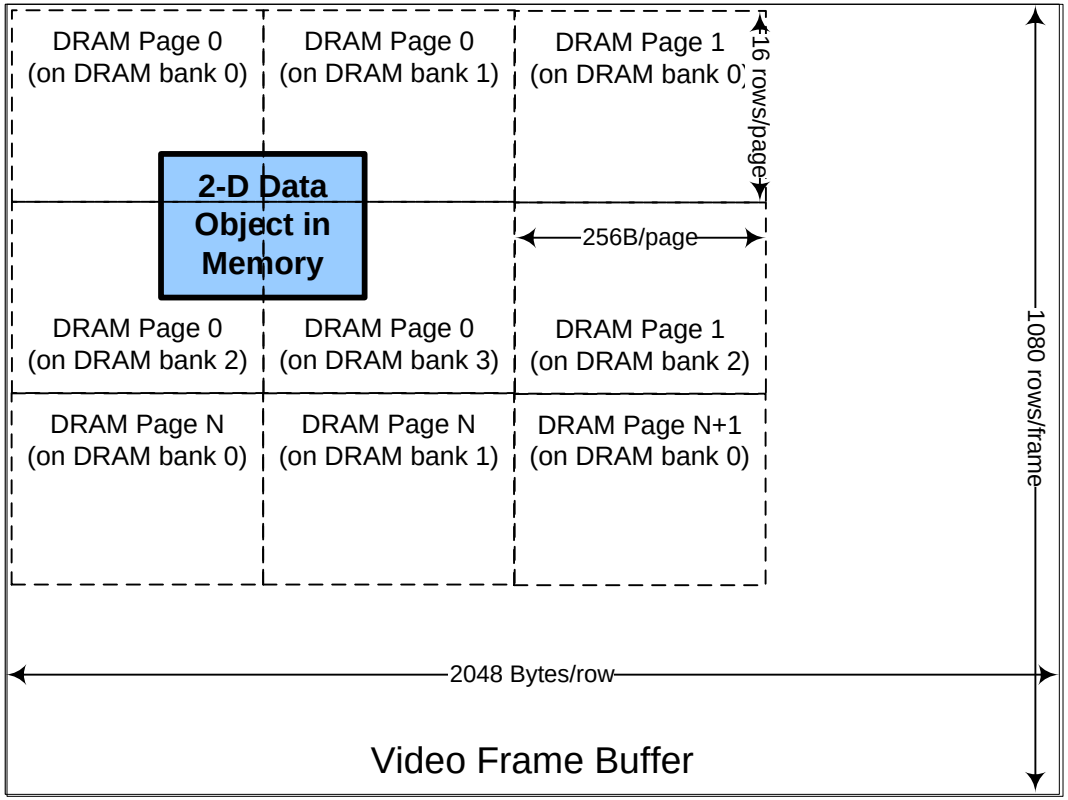
Locality Challenges: DRAM Access Styles

- Exploiting spatial locality is key for high utilization
  - CPUs tend to stay with an O/S page (multiple DRAM pages)
  - Much processing and I/O DMA uses long incrementing bursts
  - Image processing is tougher
- Two-dimensional bursts
  - Popular for video and graphics
  - Operate on a set of adjacent pixels



Locality Challenges: DRAM Access Styles

- Exploiting spatial locality is key for high utilization
  - CPUs tend to stay with an O/S page (multiple DRAM pages)
  - Much processing and I/O DMA uses long incrementing bursts
  - Image processing is tougher
- Two-dimensional bursts
  - Popular for video and graphics
  - Operate on a set of adjacent pixels
- Address tiling
  - Rearrange DRAM address organization to exploit locality
  - Avoids page misses
  - One size doesn't fit all!

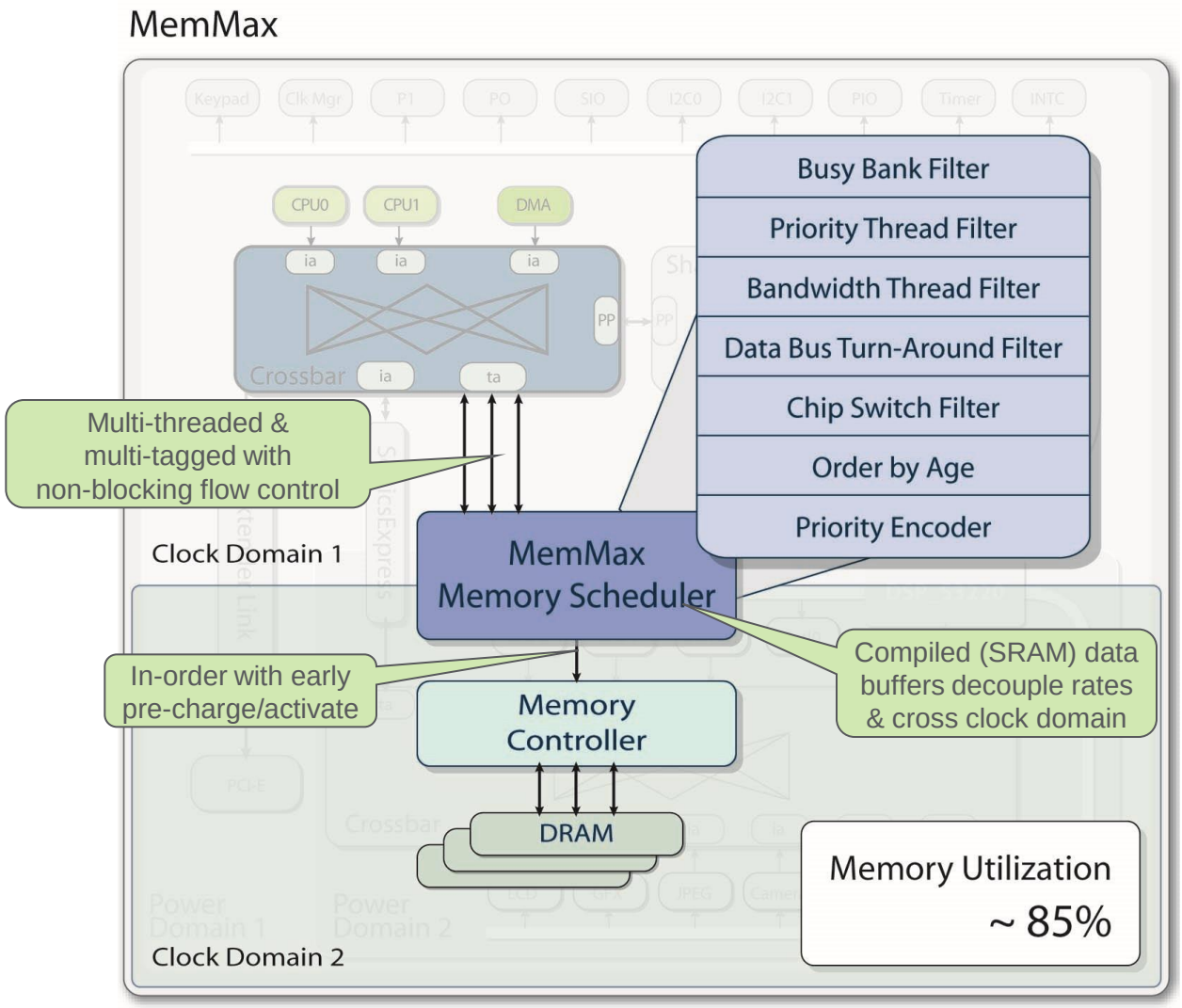


Optimizing for High Utilization

Goal	Approach
Minimize RD/WR turnarounds	Group reads and writes
Hide page misses	Bank scheduling
Avoid page misses to banks with conflicting transfers in flight	Bank state tracking
Maximize page/bank scheduling opportunities at minimum area	Expose intrinsic traffic concurrency to scheduler
Ensure write data bursts at DRAM rate	Buffer write data in DRAM clock domain
Ensure read data absorbed at DRAM rate	Buffer read data in DRAM clock domain
Isolate SoC architecture from DRAM clock	Asynchronous FIFO
Prevent low-bandwidth initiators from stalling DRAM	Decoupling FIFO
Minimize CPU latency	Make CPU highest priority, interleave bursts & ensure paths cannot block
Eliminate page/bank thrashing	Protect groups of DRAM bursts against higher priority traffic
Protect against best-effort traffic starvation	Demote QoS traffic overusing bandwidth & fair best-effort arbitration

Compiled SRAM

Example: MemMax™ Memory Scheduler

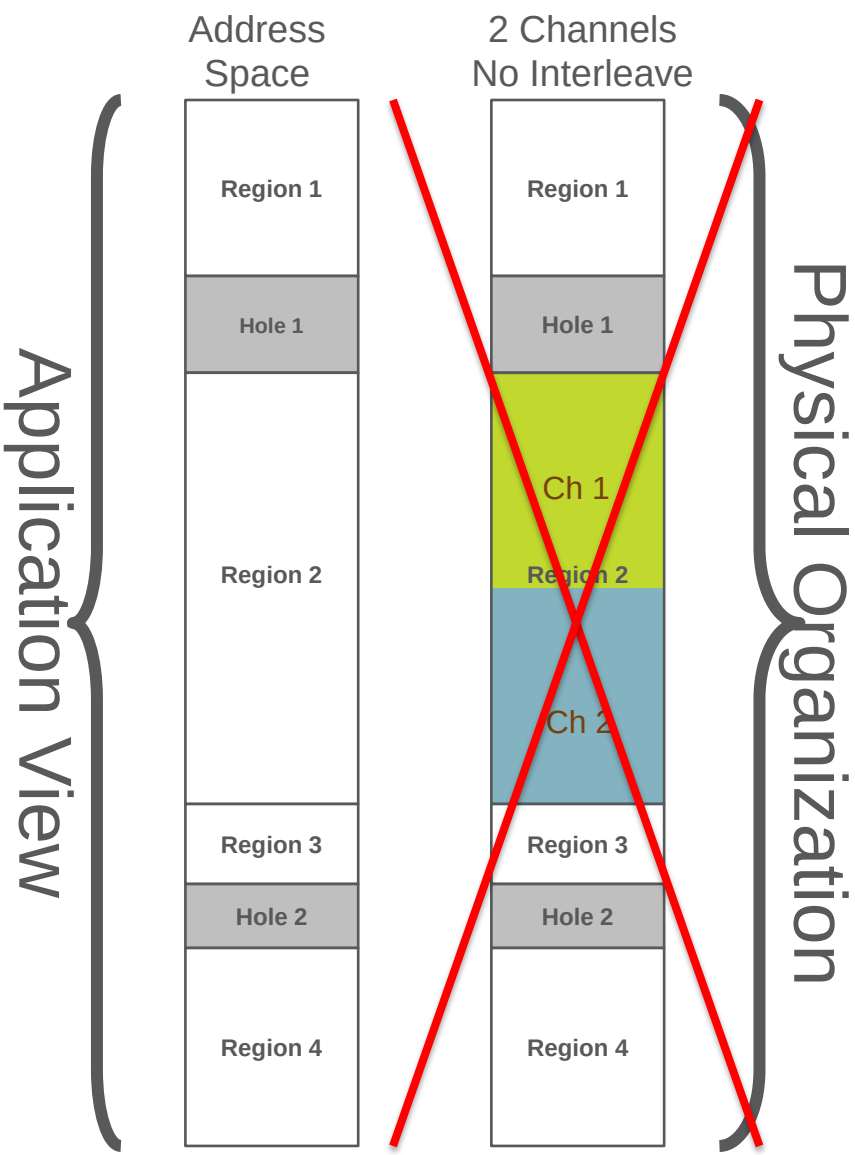


SONICS

Solution Requirements for Multichannel SoCs

- Optimization for high DRAM ***utilization*** and good Quality of Service (***QoS***) characteristics
- ***Load balancing*** traffic from the increasing numbers of DRAM consumers across increasing numbers of DRAM channels
- ***Eliminating performance loss*** due to ordering dependencies while ***minimizing buffering***
- ***Analysis tooling*** to understand design trade-offs

Multichannel Support is *Not* Easy!

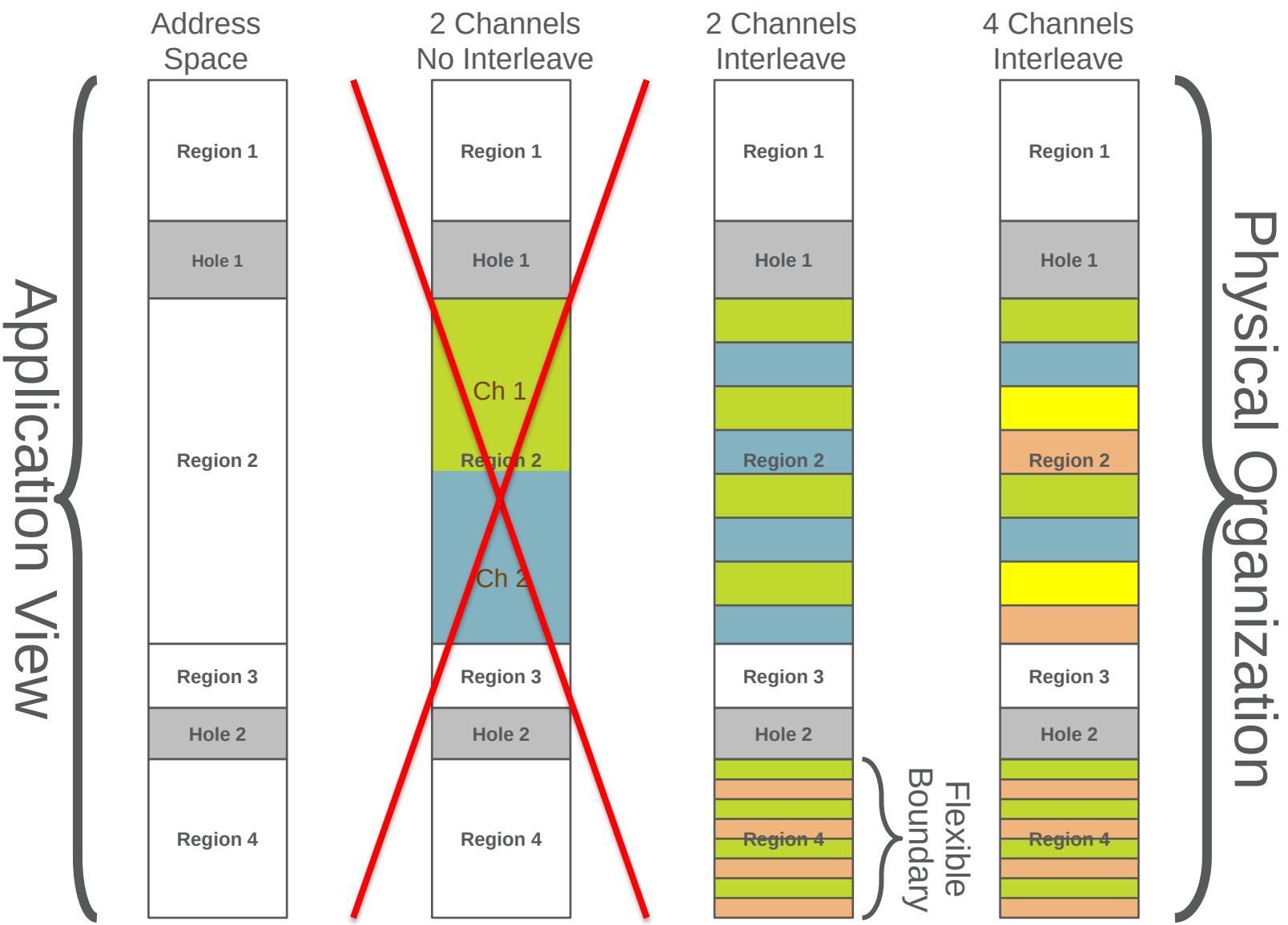


Key Problems:

- Load balancing
  - Must balance memory traffic evenly among channels
- Maintaining throughput
  - Multiple channels cause throughput/ordering problems with pipelined memories
  - DRAM controllers rely on reordering to get higher throughput

This means software and IP cores must manage multiple channels

Transparent Multichannel Interleaving with Access-Optimized Boundaries



SONICS

Selecting Interleaving Boundaries

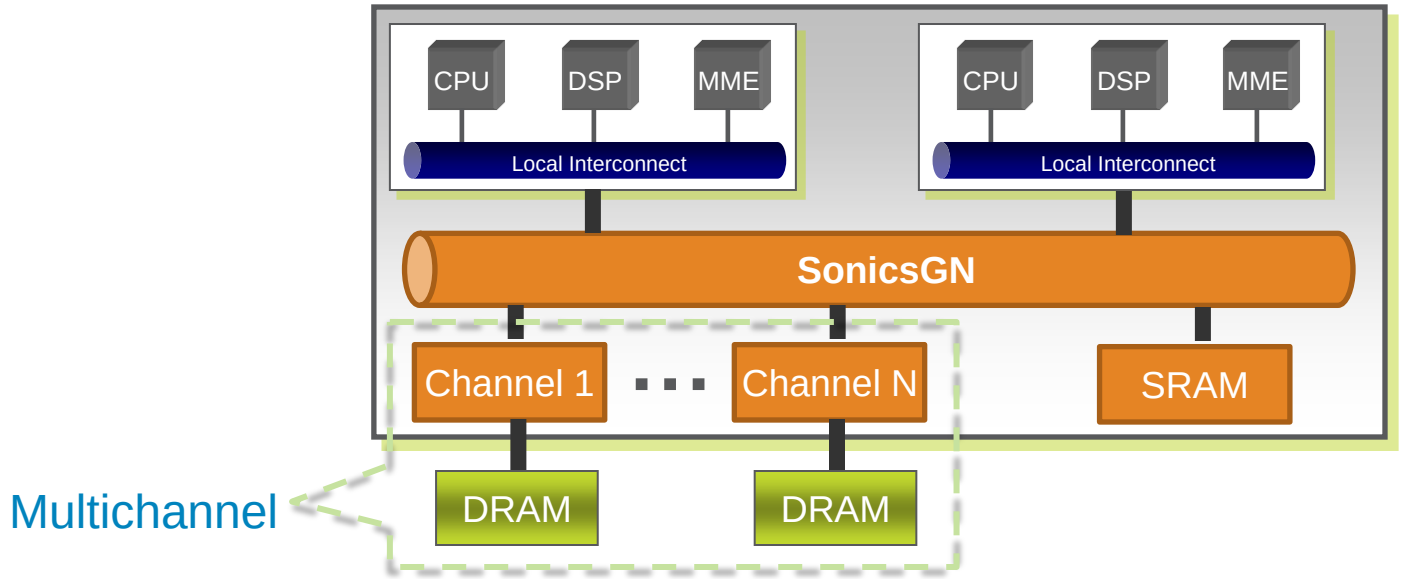
- Interleaving improves traffic load balancing by taking advantage of the ***existing access patterns*** of processors and algorithms in DRAM
  - Code segments for instruction execution
  - Data structures
  - FIFO and circular buffers
- Important to consider the time window over which traffic balance is considered
  - Since goal is to keep DRAM busy, window $\propto$ DRAM queueing delay
- Coarse-grained interleaving (e.g. 4 KByte): quite imbalanced
- Fine-grained interleaving (64-256 Bytes): much better
  - Need to be careful about impact on DRAM page hit behavior
  - Requires transaction splitting (and reassembly) when crossing

Multichannel Interleaving in the Network: Higher Performance, Lower Area, More Scalable

- Interleaving support requires splitting traffic for delivery to proper channel
- Splitting in memory scheduler/controller does not scale
  - Creates routing bottleneck at controller
  - Creates performance bottleneck at internal arbiter
- Interleaved Multichannel Technology (IMT) splitting*
  - Fully distributed architecture enables scalability
  - Network overlaps channel accesses to maximize throughput
  - Optimized protocols minimize reorder buffer area
  - Isolating channels from IP cores makes it transparent to software and other hardware

*Patented

SonicsGN and Multichannel DRAM



- Sonics' IMT™ (Interleaved Multichannel Technology)
  - Delivers transparent memory management for high performance SoCs
  - Manages load balancing and channel splitting automatically
  - First introduced in SonicsSX, 2008
- Also supports multiple transactions outstanding to any collection of targets
  - Manages reordering to satisfy interface requirements (AXI IDs, OCP tags)

Solution Requirements for Multichannel SoCs

- Optimization for high DRAM ***utilization*** and good Quality of Service (***QoS***) characteristics
- ***Load balancing*** traffic from the increasing numbers of DRAM consumers across increasing numbers of DRAM channels
- ***Eliminating performance loss*** due to ordering dependencies while ***minimizing buffering***
- ***Analysis tooling*** to understand design trade-offs

High Throughput with Multichannel DRAMs

- Challenges
  - Initiator interface protocols have ordering requirements
    - Concurrency tags (e.g. AXI A*ID) enable more flexible ordering
  - Fine-grained interleaving chops single transactions into multiple
    - Which need to be re-assembled in-order for response delivery
  - DRAM subsystem achieves highest throughput when out-of-order
  - DRAM subsystem is deeply pipelined to cover high latency
    - Many transactions typically outstanding – many ordering constraints
  - Need to avoid ***ordering deadlocks***
    - Especially when DRAM subsystem has blocking protocols
- High throughput ***requires*** issuing dependent transactions to independent channels – at least some of the time
- A ***reorder buffer*** holds early responses to prevent deadlocks until prior transactions complete
  - Reorder buffers can get large

Optimizing Throughput While Minimizing Area

Techniques to minimize reorder buffer overhead

- Don't use them ☺
  - Unneeded for initiators with high concurrency and low chopping
  - Unneeded for low throughput, latency-insensitive initiators
- Place them at/near the DRAM subsystem
  - Total reorder storage limited by command queue of scheduler (much less than Σ initiator_issuing_capability)
  - DRAM schedulers with multi-thread/VC support provide per-VC flow control to control response ordering
- Minimize them
  - If access patterns known, can optimize depth based on concurrency and throughput vs. DRAM latency characteristics
- Compile them
  - Reasonably sized buffers much lower cost as SRAM

Helping Out the Elders…

- Many legacy initiators need reasonable throughput, but still don't use concurrency IDs
- Even with a single DRAM channel, such traffic may not be efficiently scheduled
  - Especially if the initiator actually merges internal streams:

```
Read bank 0, page 0, column 0
Read bank 0, page 1, column 0
Read bank 0, page 0, column 8
Read bank 0, page 1, column 8
```

Bank conflict!

- Network can generate per-transaction concurrency IDs to enable out-of-order scheduling
  - Use reorder buffer to maintain ordering

Solution Requirements for Multichannel SoCs

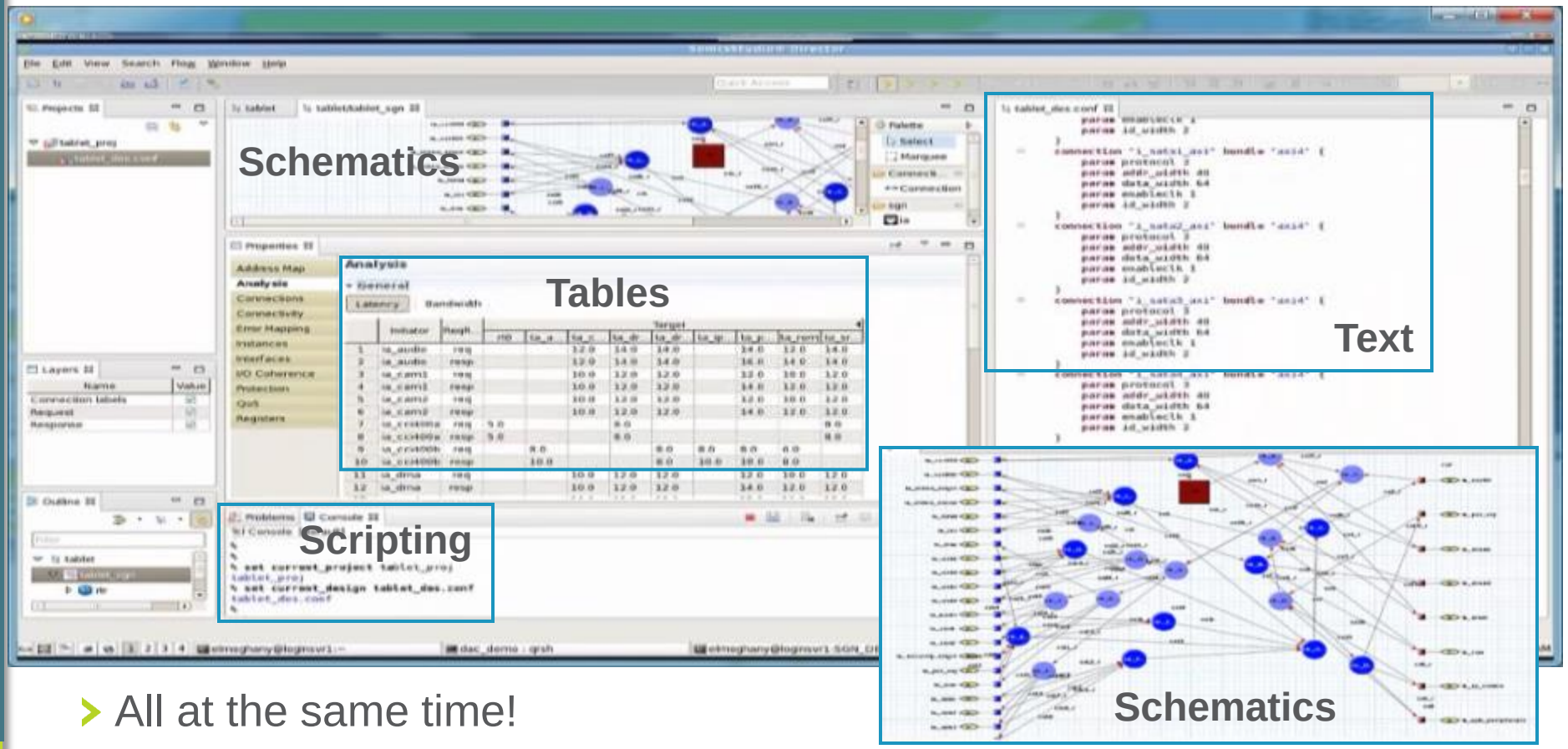
- Optimization for high DRAM ***utilization*** and good Quality of Service (***QoS***) characteristics
- ***Load balancing*** traffic from the increasing numbers of DRAM consumers across increasing numbers of DRAM channels
- ***Eliminating performance loss*** due to ordering dependencies while ***minimizing buffering***
- ***Analysis tooling*** to understand design trade-offs

Multichannel Design Needs Analysis Tools

- Multichannel SoC performance dependent on many factors
  - DRAM organization
  - Number of channels
  - Interleaving boundaries
  - Initiator traffic characteristics
  - Buffer sizing
  - QoS settings
  - Etc.
- Can be very difficult to analyze statically
- Requirements
  - Performance testbench creation
  - Traffic generation
  - ***Fast and accurate*** simulation
  - Performance analysis

SonicsStudio® Director

- Architect's – and integration team's – GUI for designing complex SoCs
- Revolutionary – because it lets ***you*** work the way ***you*** choose



- All at the same time!

Performance Optimization Flow

- The SonicsStudio Flow automates and organizes tasks
  - Generate
    - Generate RTL or ***cycle-accurate SystemC*** models (> 20x faster)
    - Generate ***performance testbenches*** (RTL or SystemC)
    - Generate ***traffic stimulus*** to model customer use cases
  - Simulate
    - ***Run simulations*** (RTL or SystemC)
  - Analyze
    - Import results database
    - ***Analyze in tables, graphs and scripts***

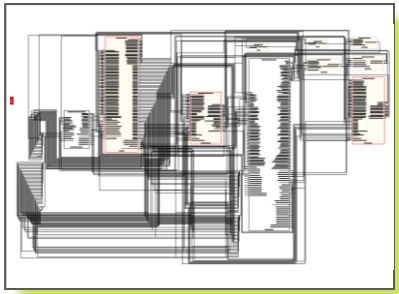
- The Flow is built on top of the Unix “make” facility
  - Maximum portability
  - Familiar to most users

Performance and System Analysis

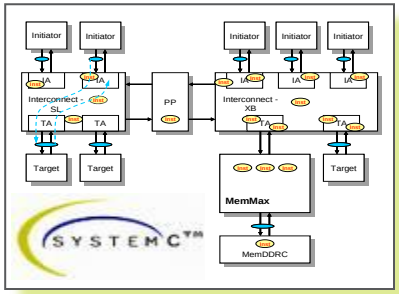
On-Chip

Debug and Performance monitoring

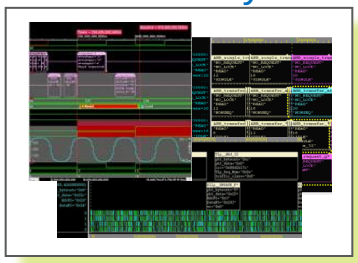
RTL simulation



SystemC simulation



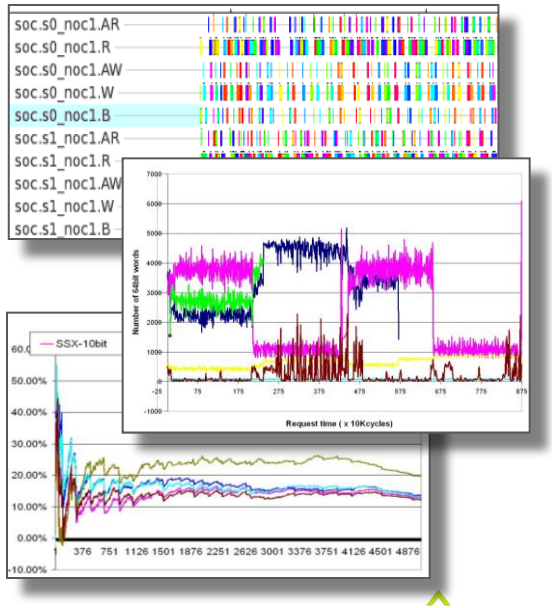
Static analysis



Common Analysis DB

Queries

SPA

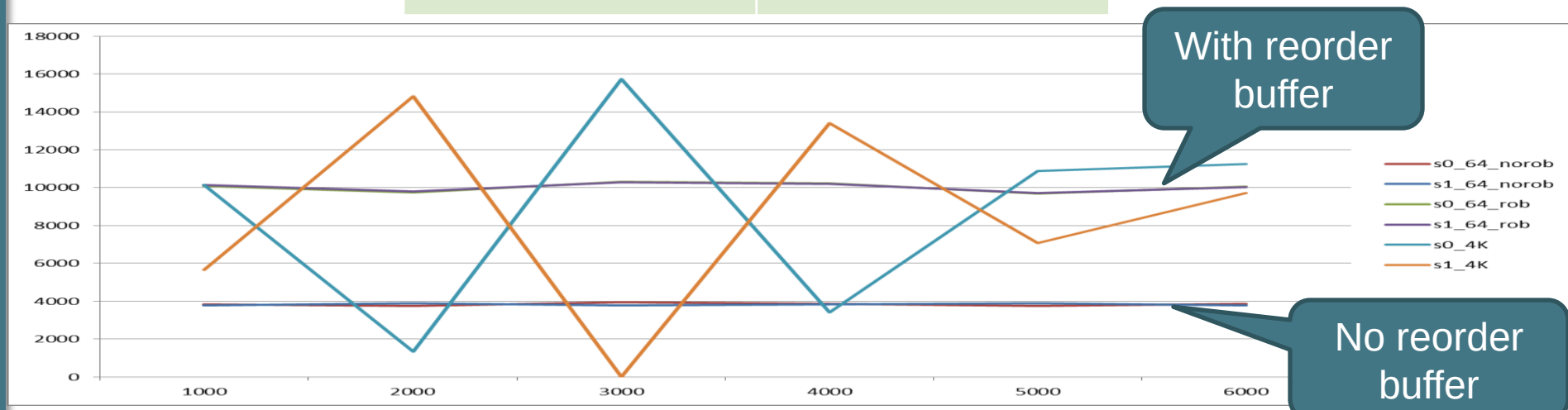


SONICS

Multichannel Performance Results

- Traffic sent to two channels from five initiators (4GB/s) with different interleaving boundaries

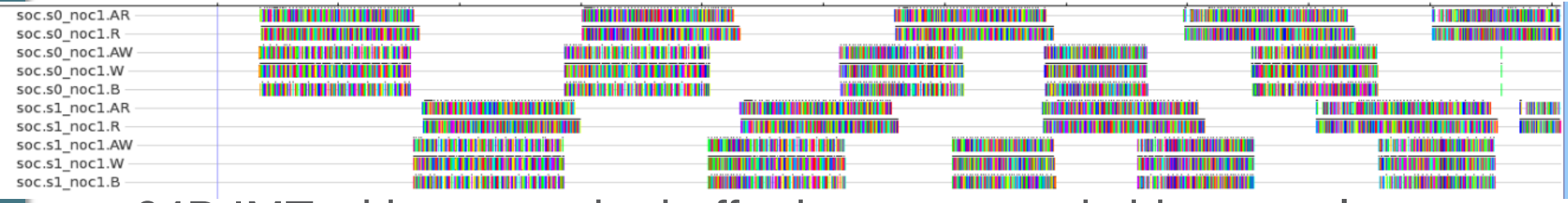
Condition	BW
4K interleave	17.2 GB/s
64B interleave	7.6 GB/s
64B w/reorder	20 GB/s



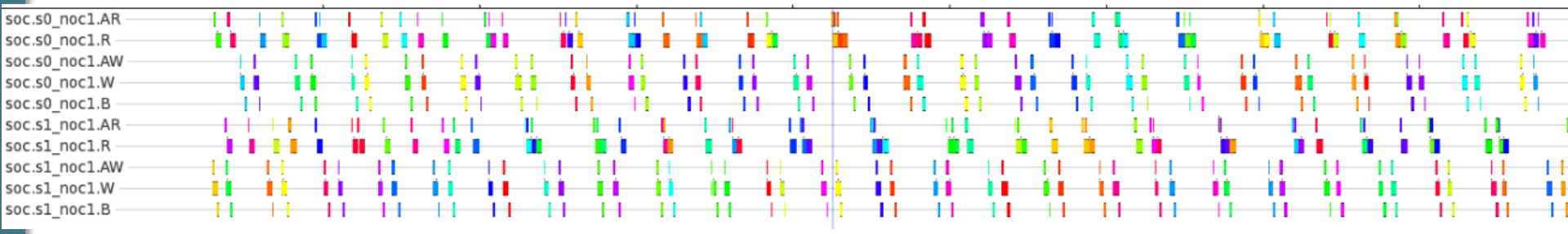
- 64B IMT w/ reorder buffer delivers **16% BW increase** !!!
  - Reorder buffer required so initiators meet BW requirements (4GB/s each)

Multichannel Performance Analysis

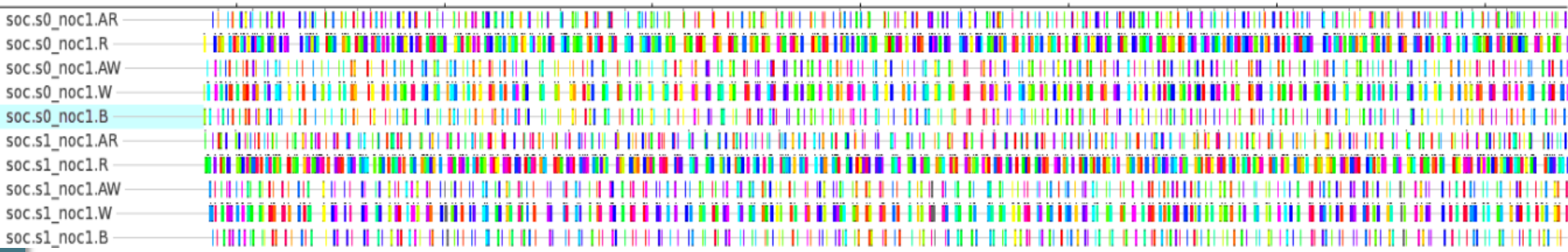
- 4K IMT suffers from non balanced BW on DRAM channels



- 64B IMT without reorder buffer has target switching penalty

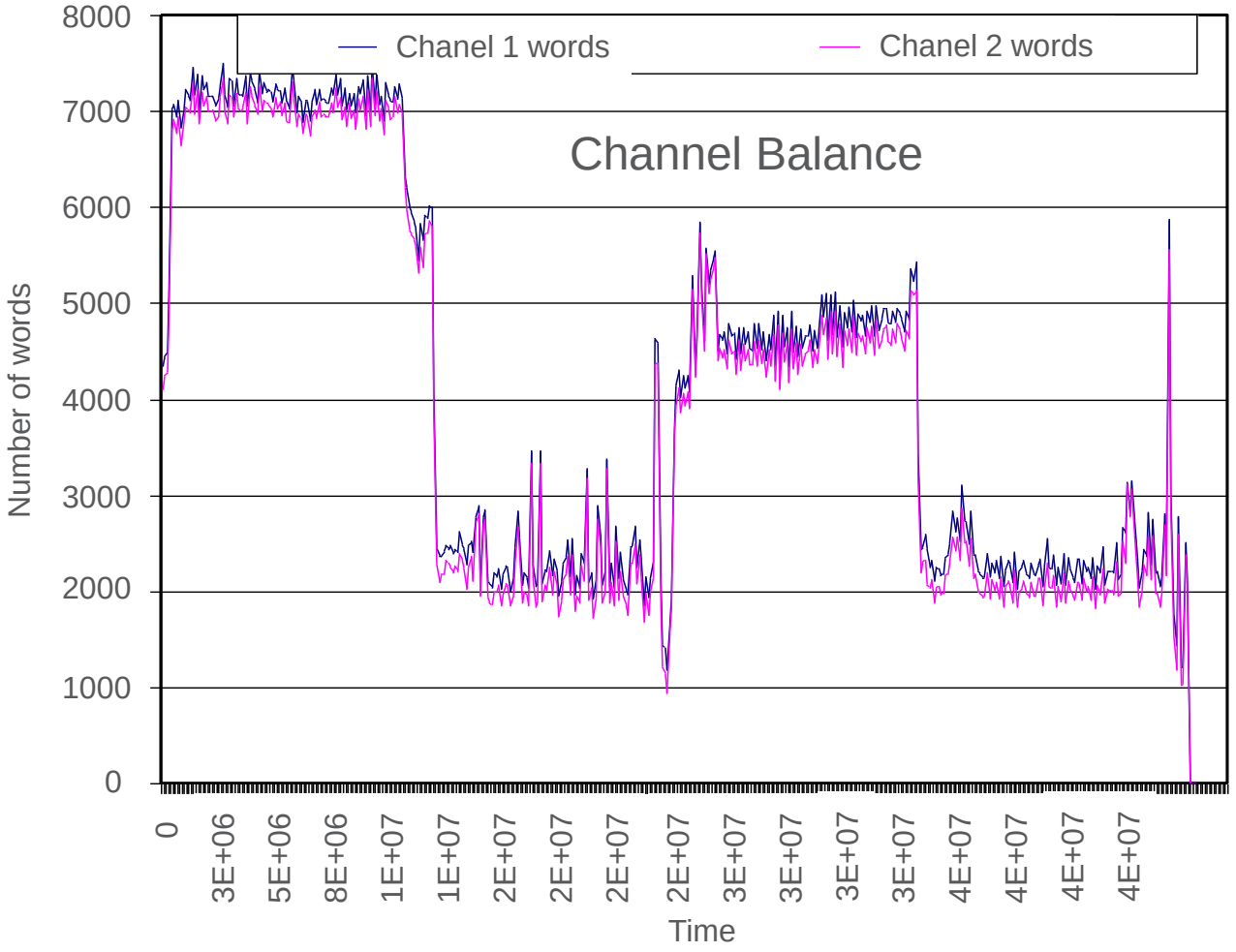


- 64B IMT with re-order buffers delivers superior load balancing



SONICS

Evaluating Load Balance



SONICS

Summary

- Multichannel DRAMs have become the only choice for many SoCs
- Achieving high throughput and QoS requires careful integration
  - DRAM scheduler and on-chip network
- Flexible interleaving is key to achieving load balance
  - And is best implemented in the network for scalability
- Complex subject area, requiring powerful/proven technology and capable tooling

Thank You!!!