MeSSMArch – A Memory System Simulator for Hardware Multithreading Architectures

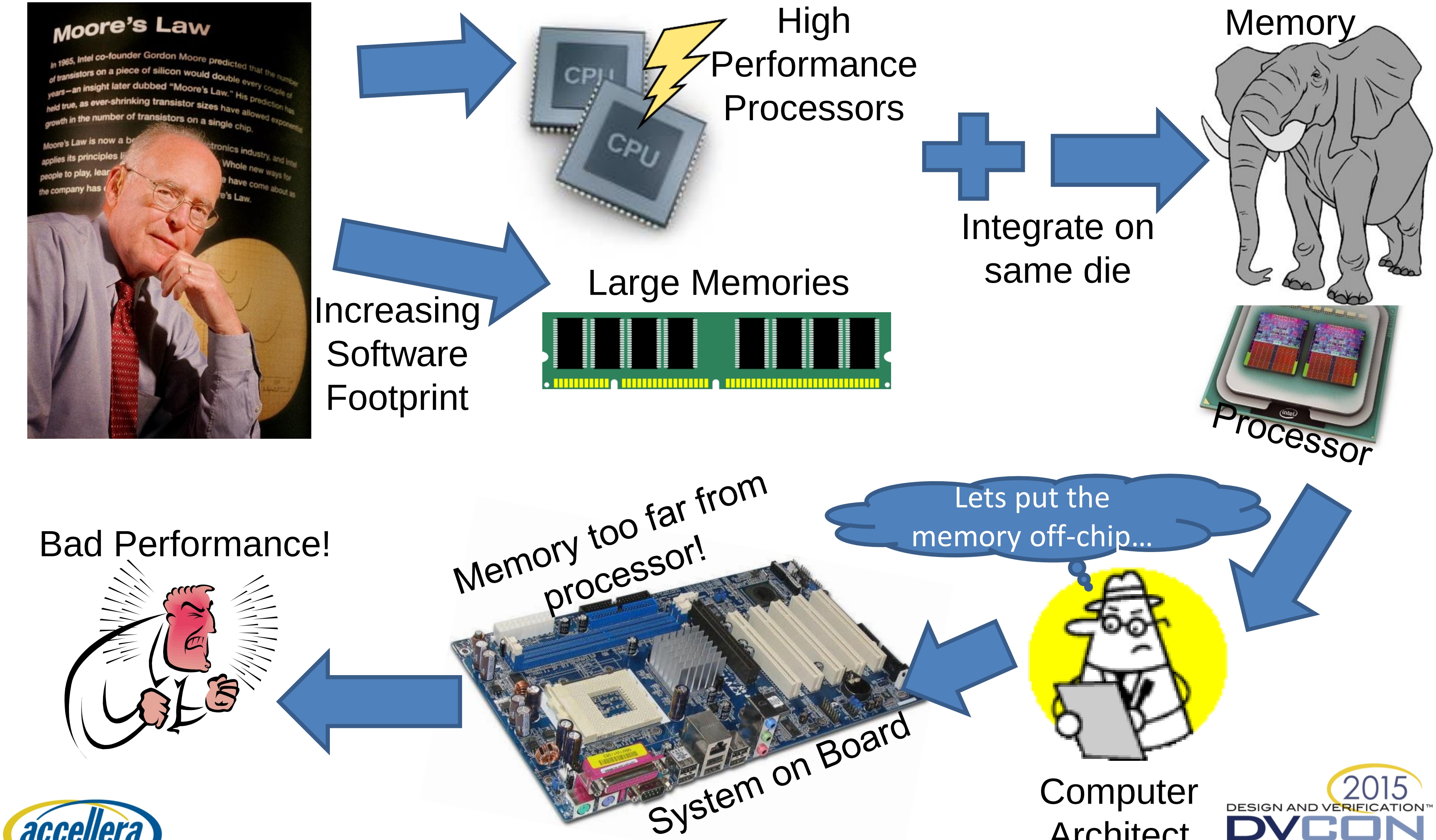
Sushil Menon

NVIDIA

Agenda

- Motivation (“Why build this simulator?”)
- TLM to the Rescue – Mapping Simulator Design Requirements to TLM Guidelines
- Modeling a Generic Memory-System at the Transaction-Level using TLM
- Simulator Validation
- Current Limitations – Scope for Future Enhancements
- Conclusion
- Appendix

Why Memory Hierarchies? – Problem


Moore's Law
In 1965, Intel co-founder Gordon Moore predicted that the number of transistors on a piece of silicon would double every couple of years—an insight later dubbed "Moore's Law." His prediction has held true, as ever-shrinking transistor sizes have allowed exponential growth in the number of transistors on a single chip.
Moore's Law is now a b
tronics industry, and Intel applies its principles li
Whole new ways for people to play, lear
e have come about as the company has
e's Law.
CPU
CPU

High Performance Processors

Increasing Software Footprint

Large Memories

Integrate on same die

Memory

Processor

Lets put the memory off-chip…

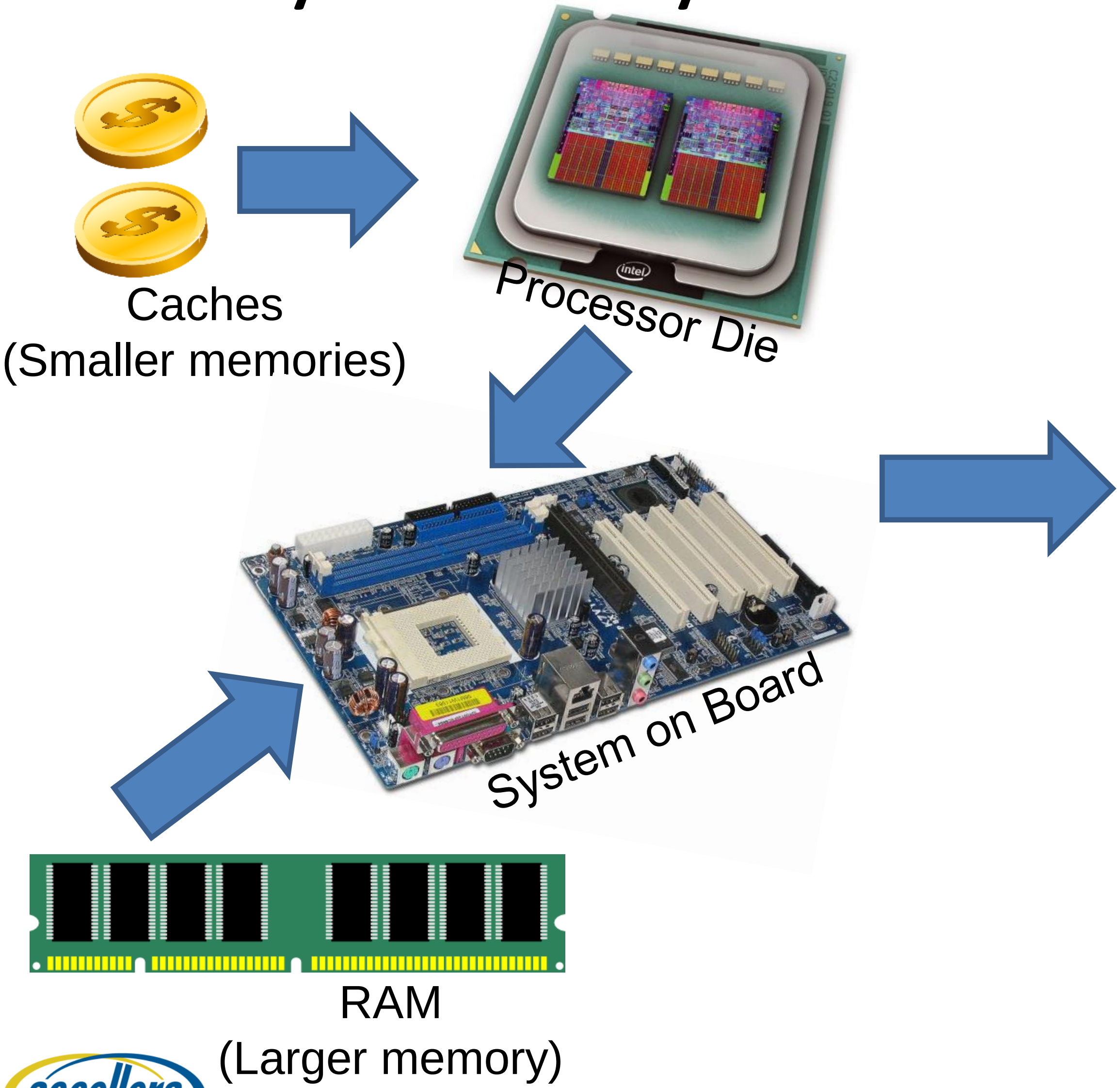
Computer Architect

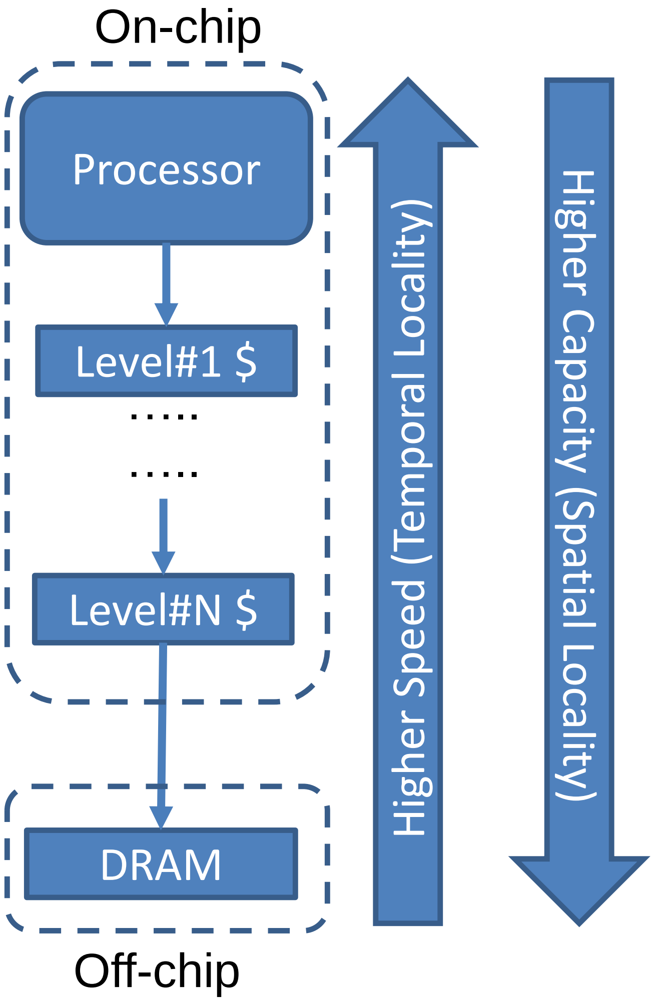
Memory too far from processor!

System on Board

Bad Performance!

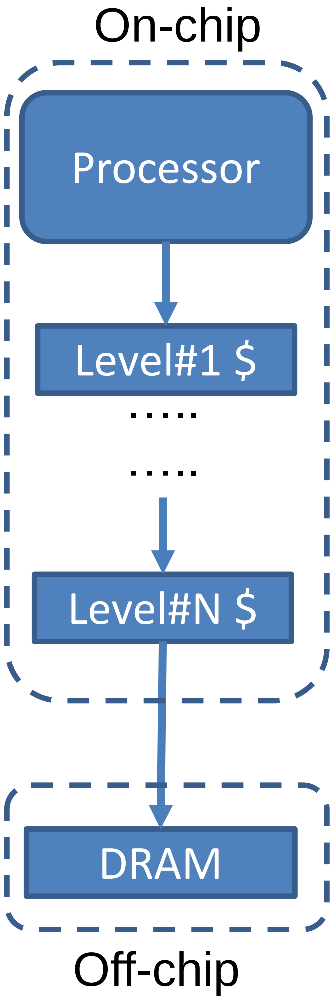
Why Memory Hierarchies? – Solution




On-chip
Processor
Level#1 $
.....
.....
Level#N $
DRAM
Off-chip
Higher Speed (Temporal Locality)
Higher Capacity (Spatial Locality)

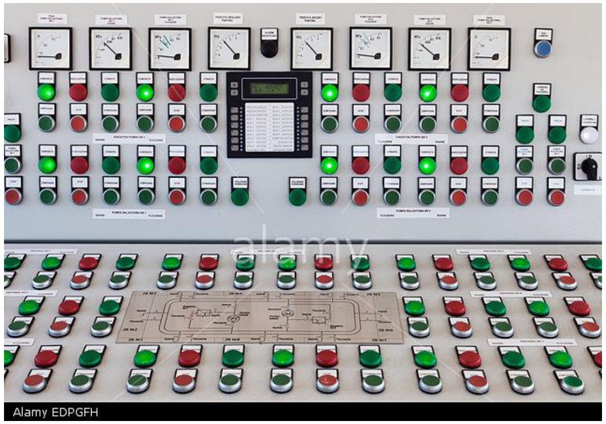
Hierarchy of Memories

Problematic Solution?! – Need for Simulators



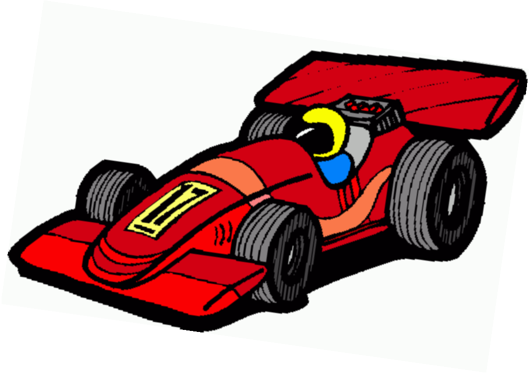
Hierarchy of Memories

Too many configuration options…best one???


Alamy EDPGFH

Computer Architect

Time-to-market pressure



Fast Simulators – Trade-off result accuracy for simulation-speed by raising level of abstraction!
➔ Enable faster design-space exploration!

TLM to the Rescue – Mapping Simulator Design Requirements to TLM Guidelines

Design Requirement	TLM Guideline
Functionally Generic No implementation-specific details, easy to model and explore	**Don’t model functionality of μ-architectural features** Capture effects through timing information
Estimate System Performance Only Coarse-grained result accuracy, fast simulation	**Model data exchange at Transaction-Level** Timing-accuracy via lumped delays
Easily Extensible Plug-n-play style architectural exploration	**Separate Computation from Communication** Model computation details inside the process and communication details inside the channel

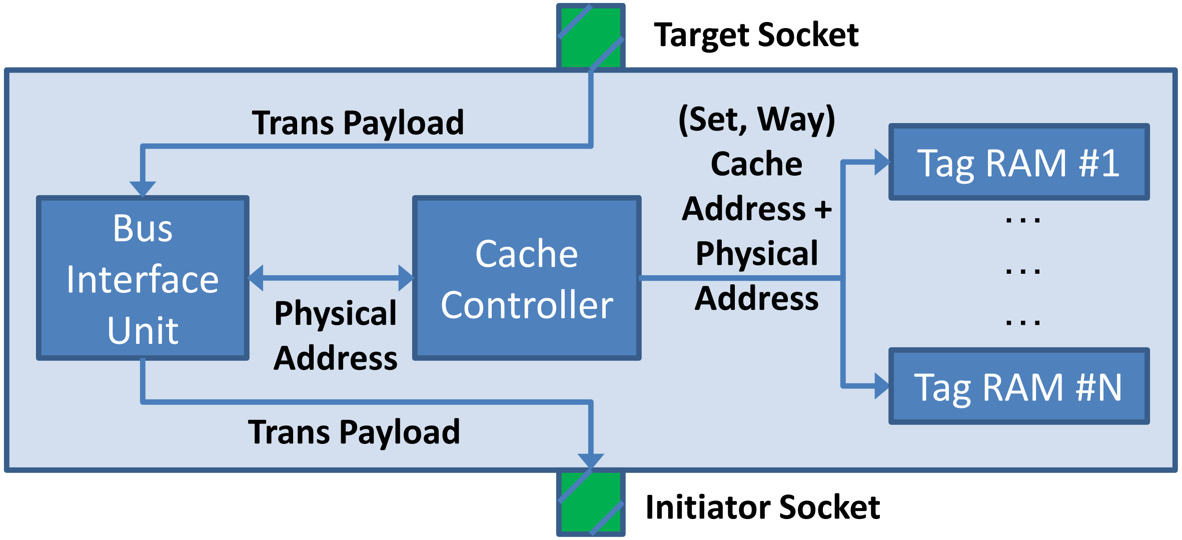
Modeling a Generic Memory-System at the Transaction-Level using TLM

- Memory-System Components:
  - Generic Cache (full-flow covered)
  - Memory Controller (only structure)*
  - Serializing Interconnect (only structure)*
  - Hardware-Thread (only structure)*
- Construct Memory-Hierarchy for a Hardware Multithreaded Architecture
- Coarse-grained accuracy → All components modeled as loosely-timed
  - Implement b_transport( ) only

***Refer the Appendix Section and the Paper for More Details**

Generic Cache – Transaction-Level Model

- Tag RAM – stores tag-address, dirty-bit, valid-bit and age-counters of a way
- Cache Controller – implements state-machines that capture functionality of cache
- Bus Interface Unit – implements interface for inter-module communication



Generic Cache – Configuration Parameters

Parameter	Unit/ Options	Description
Cache Size	Kilobytes	Size of the cache
Cache Line-Size	Bytes	Size of a cache-line
Associativity	<NA>	# of ways in a set
# of Comparators	<NA>	# of comparators used during a lookup
Write-Allocate	Yes/No	Allocation of a way-entry on a cache-miss
Write-Through	Yes/No	Generate write-transaction to lower-level on write-hit
Way Prediction	Yes/No	Predict way of current access to reduce lookup-time
Clock Period	Nanosecs	Time-period of a clock-cycle

Green entries indicate micro-architectural features

Generic Cache - µArch Features

- # of comparators
  - Used in parallel tag-lookup – don’t do parallel lookup!
  - Perform sequential tag-lookup and then divide the time by # of comparators
  - Equation: $\# \; of \; clock \; cycles = \left\lceil \frac{Associativity}{\# \; of \; Comparators} \right\rceil$
- Way Prediction
  - Don’t model setting of multiplexor to channel data, etc.
  - Perform normal lookup and conditionally adjust time based on way accessed (prediction: LRU-way)
  - Equation:

$$if \; (WP \; enabled \; \& \; HitWay == LRUWay) \; then \qquad \# \; of \; clock \; cycles = 1$$

Generic Cache – Transaction-Level Data Exchange

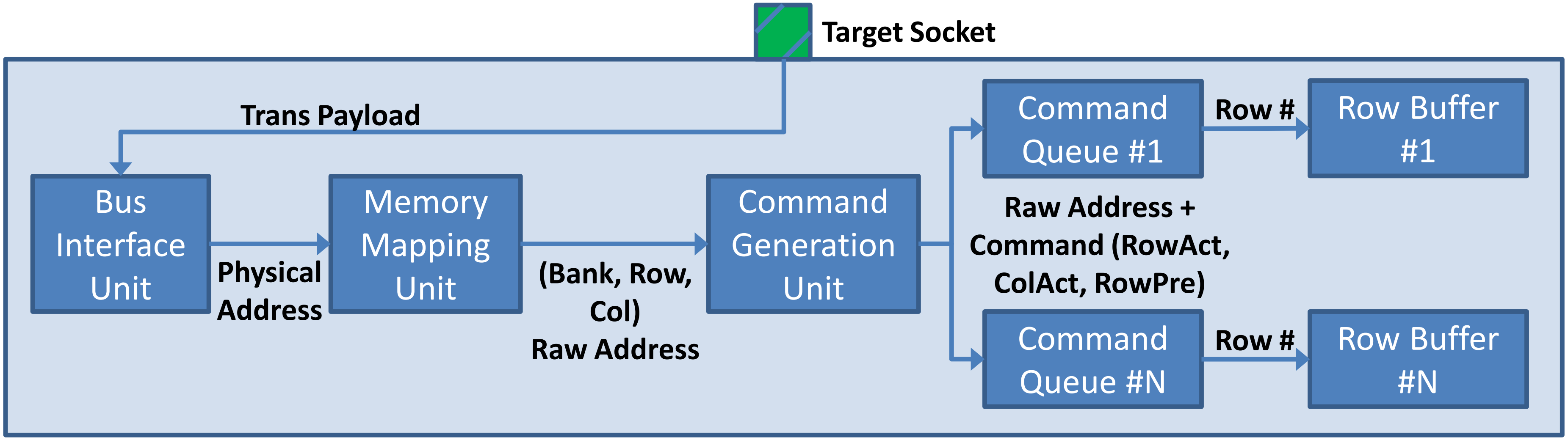
- Track cumulative # of clock-cycles during execution of state-machines at current-level without context-switching
- Conditionally forward transaction to lower-level – lumped time-delay received on return-path

$$\begin{aligned}&Total\ Transaction\ Time\ at\ Current\ Level\\&= (\#\ of\ cycles\ at\ current\ level\ x\ cycle\ time)\\&+ lower\ level\ lumped\ time\ delay\end{aligned}$$

- Return Total Transaction Time upstream as lumped time-delay

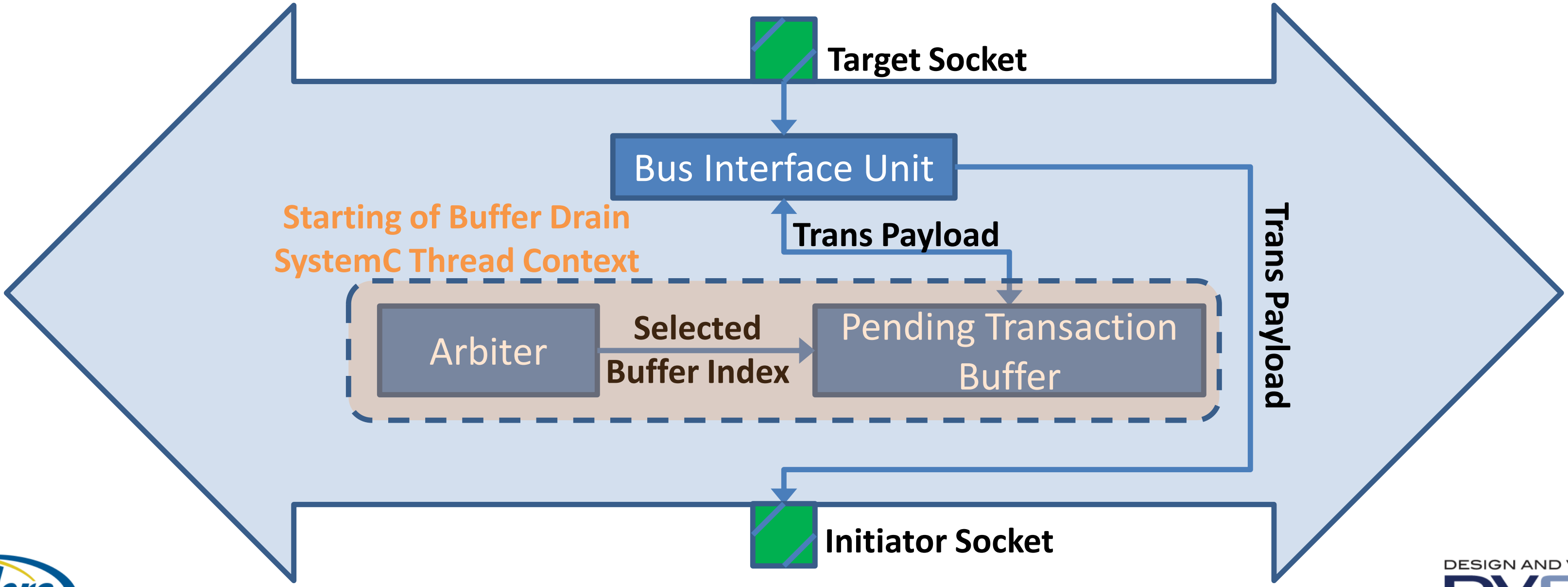
Memory Controller – Transaction-Level Model

- Memory Mapping Unit – physical to raw-address translation
- Command Generation Unit – generates commands to be performed on DRAM for data-access (stored in command queues)
- Row-Buffer – stores row-address of currently opened row
- Bus Interface Unit – implements interface for inter-module communication



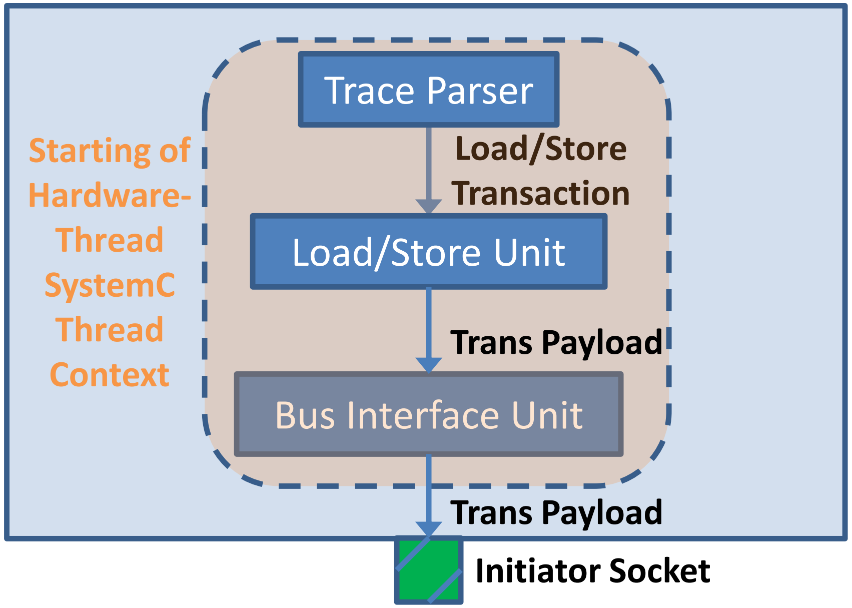
Serializing Interconnect – Transaction-Level Model

- Pending Transaction Buffer (PTB) – stores incoming transaction payload
- Arbiter – implements algorithms to select transaction from PTB for injection downstream
- Bus Interface Unit – implements interface for inter-module communication

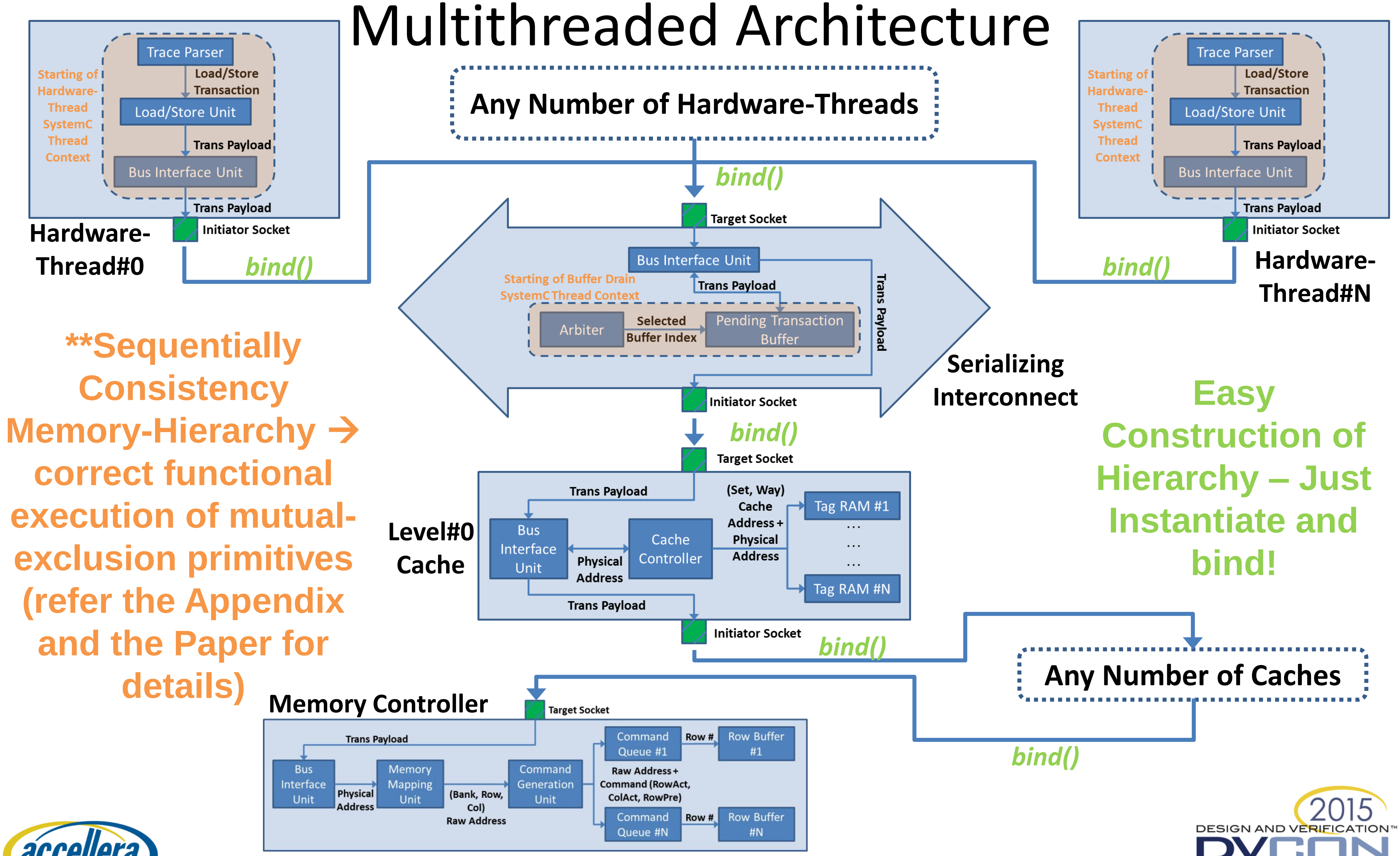


Hardware-Thread – Transaction-Level Model

- Load/Store Unit – single-entry depth FIFO which when given a load/store transaction, generates the transaction payload
- Trace Parser – infrastructure to read and parse benchmark file
- Bus Interface Unit – implements interface for inter-module communication



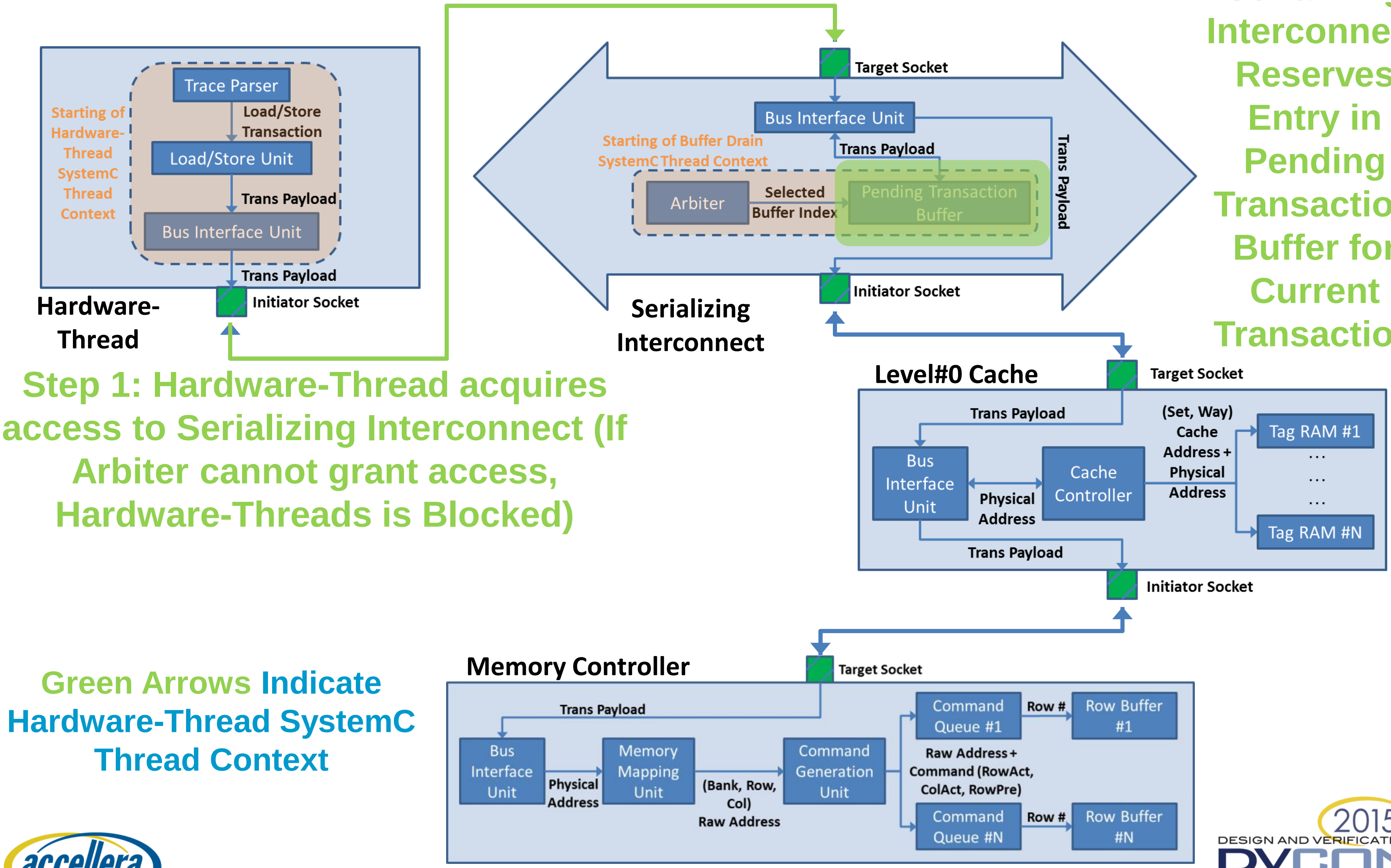
Constructed Memory-Hierarchy for a Hardware Multithreaded Architecture



****Sequentially Consistency Memory-Hierarchy → correct functional execution of mutual-exclusion primitives (refer the Appendix and the Paper for details)**

Easy Construction of Hierarchy – Just Instantiate and bind!

Tracing a Memory Transaction

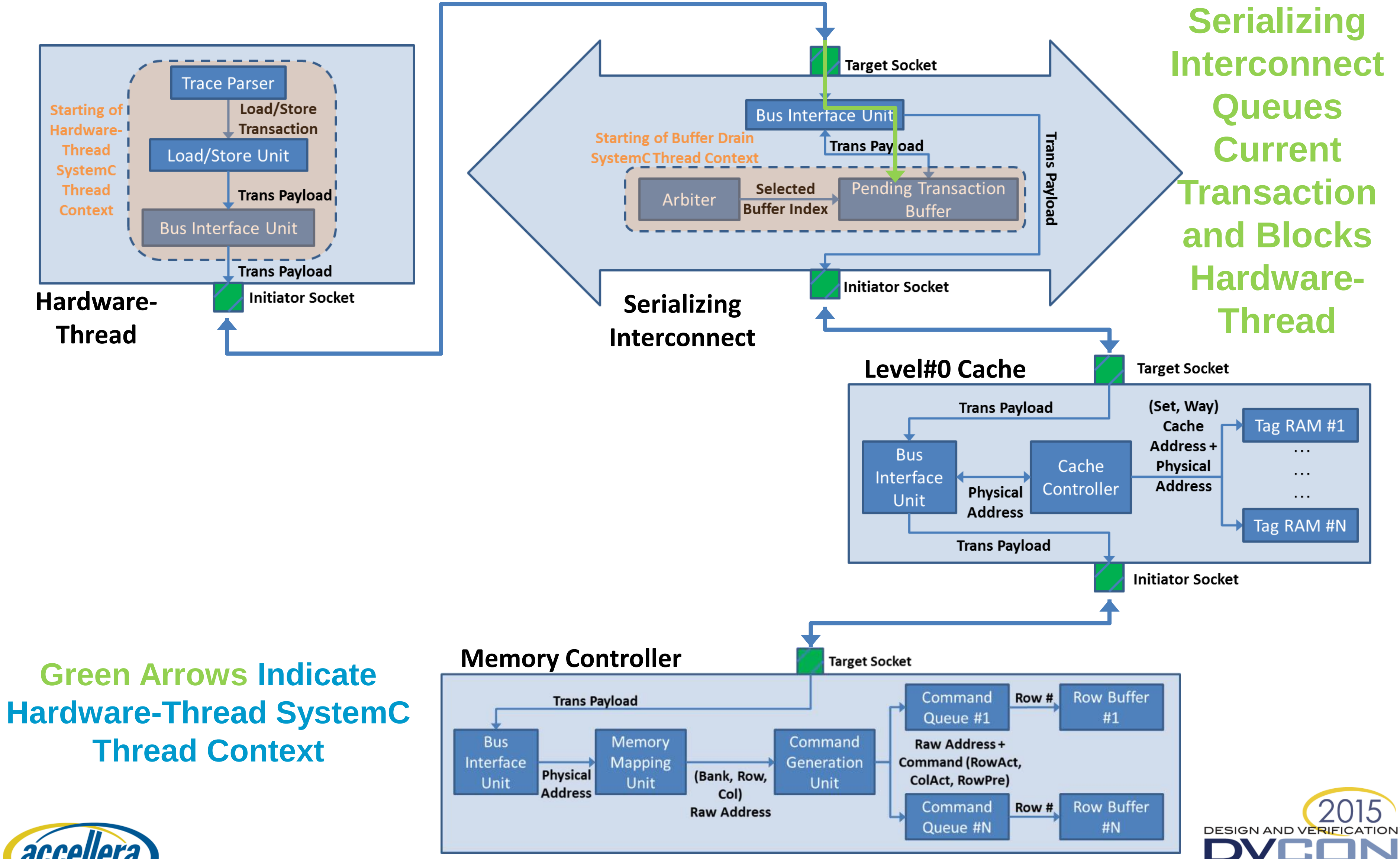


Serializing Interconnect Reserves Entry in Pending Transaction Buffer for Current Transaction

Step 1: Hardware-Thread acquires access to Serializing Interconnect (If Arbiter cannot grant access, Hardware-Threads is Blocked)

Green Arrows Indicate Hardware-Thread SystemC Thread Context

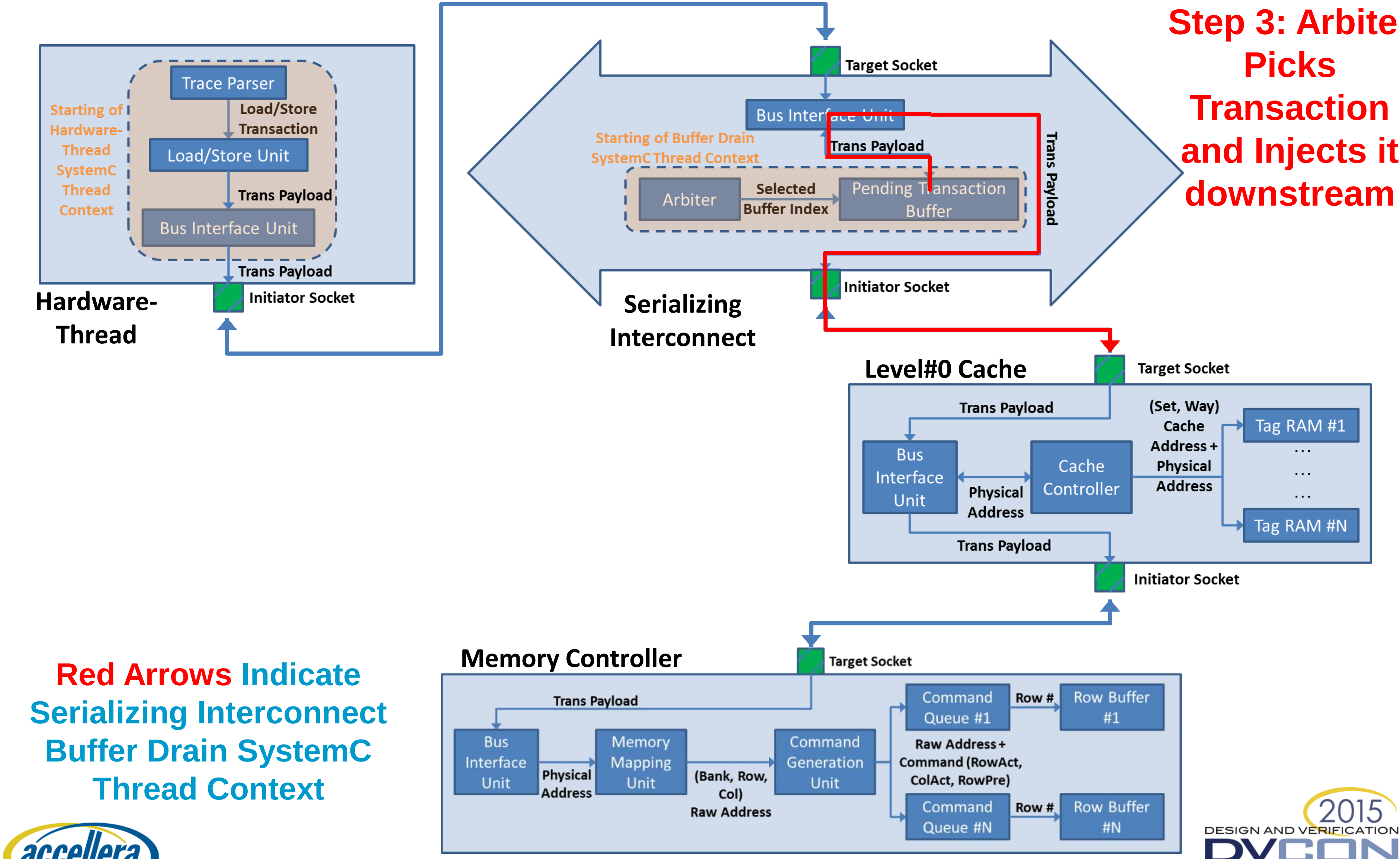
Tracing a Memory Transaction



Step 2: Serializing Interconnect Queues Current Transaction and Blocks Hardware-Thread

Green Arrows Indicate Hardware-Thread SystemC Thread Context

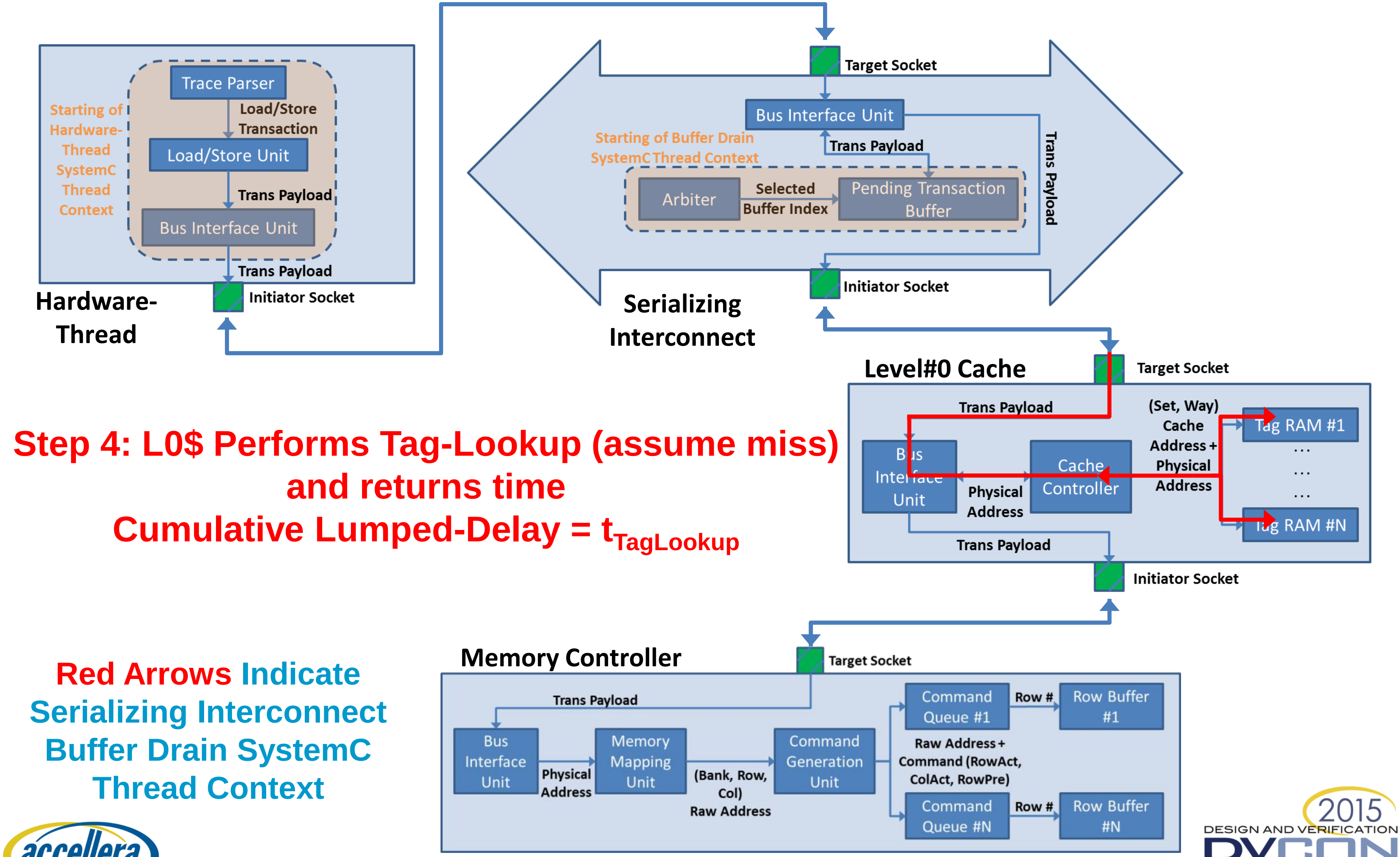
Tracing a Memory Transaction



Step 3: Arbiter Picks Transaction and Injects it downstream

Red Arrows Indicate Serializing Interconnect Buffer Drain SystemC Thread Context

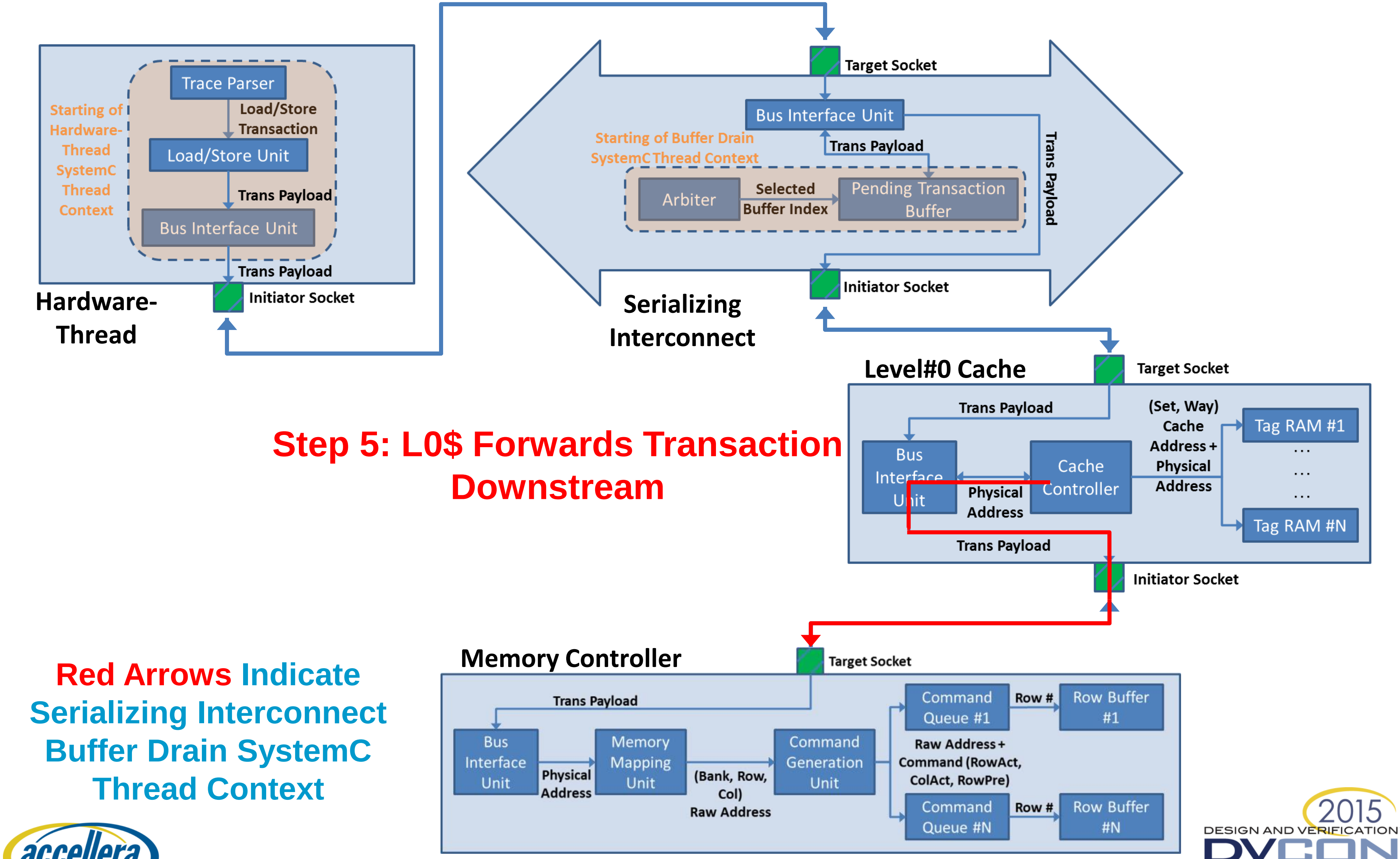
Tracing a Memory Transaction



Step 4: L0$ Performs Tag-Lookup (assume miss) and returns time
Cumulative Lumped-Delay = $t_{TagLookup}$

Red Arrows Indicate Serializing Interconnect Buffer Drain SystemC Thread Context

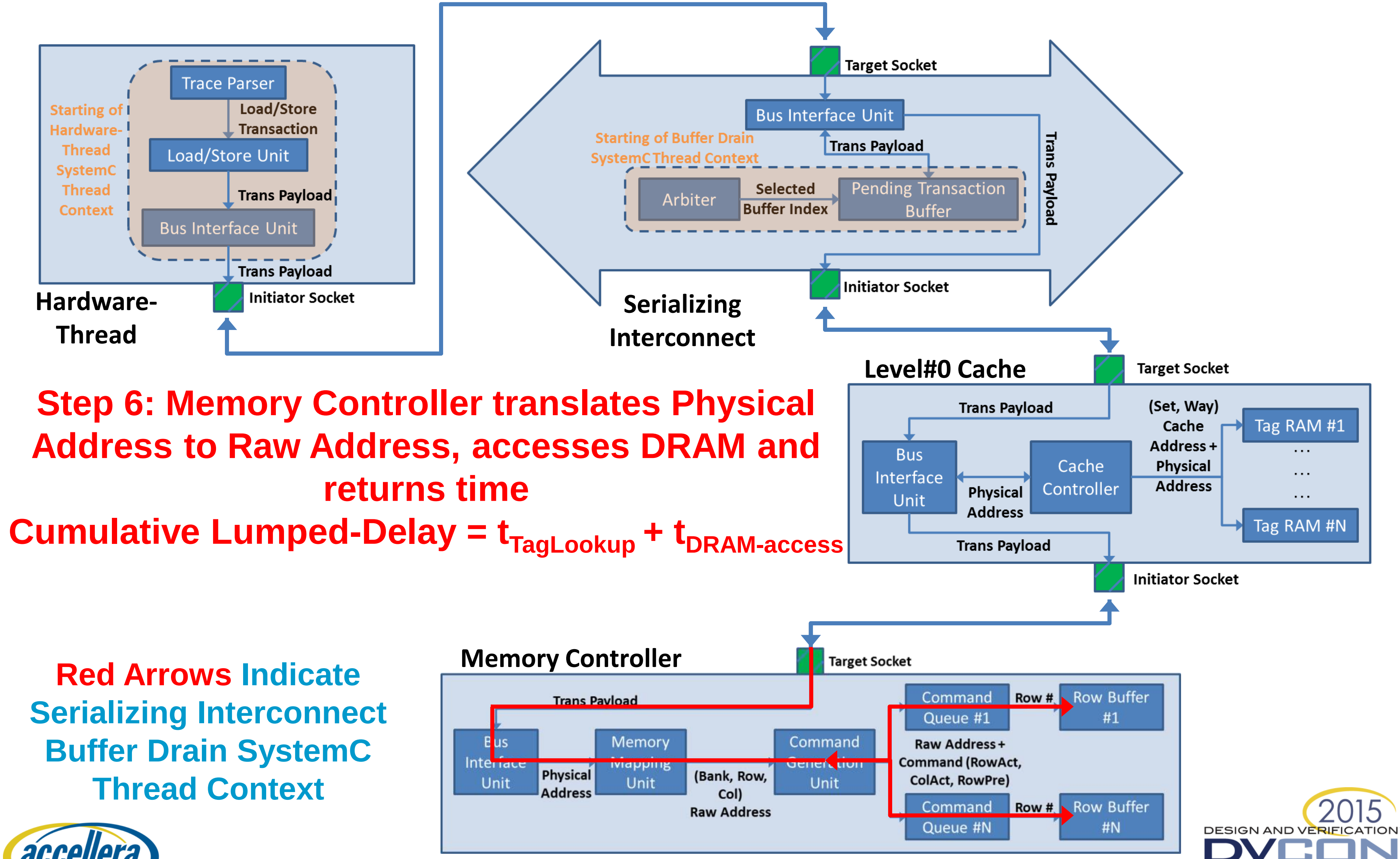
Tracing a Memory Transaction



Step 5: L0$ Forwards Transaction Downstream

Red Arrows Indicate Serializing Interconnect Buffer Drain SystemC Thread Context

Tracing a Memory Transaction

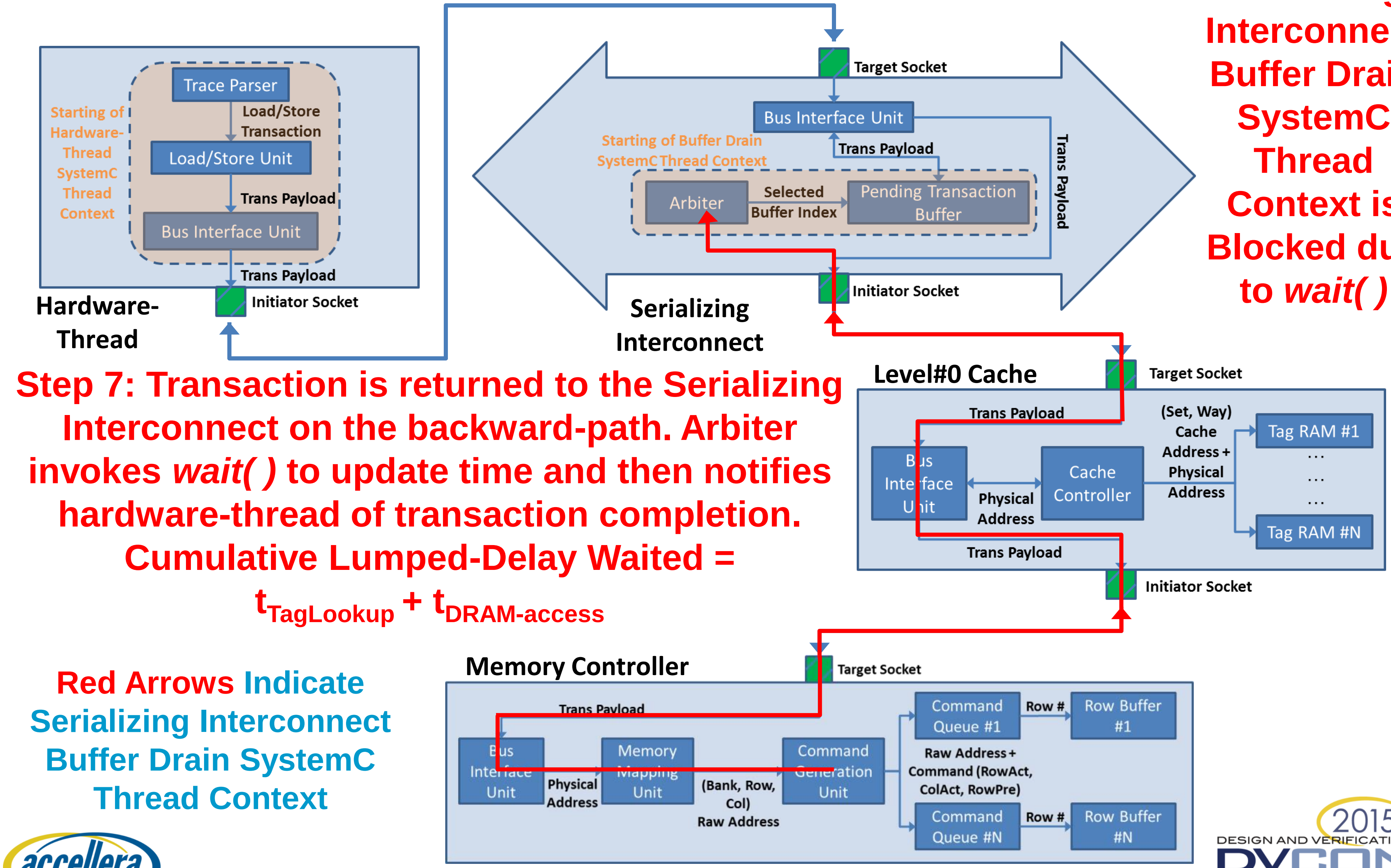


Step 6: Memory Controller translates Physical Address to Raw Address, accesses DRAM and returns time

Cumulative Lumped-Delay = $t_{TagLookup}$ + $t_{DRAM\text{-}access}$

Red Arrows Indicate Serializing Interconnect Buffer Drain SystemC Thread Context

Tracing a Memory Transaction



Serializing Interconnect Buffer Drain SystemC Thread Context is Blocked due to *wait( )*

Step 7: Transaction is returned to the Serializing Interconnect on the backward-path. Arbiter invokes *wait( )* to update time and then notifies hardware-thread of transaction completion.

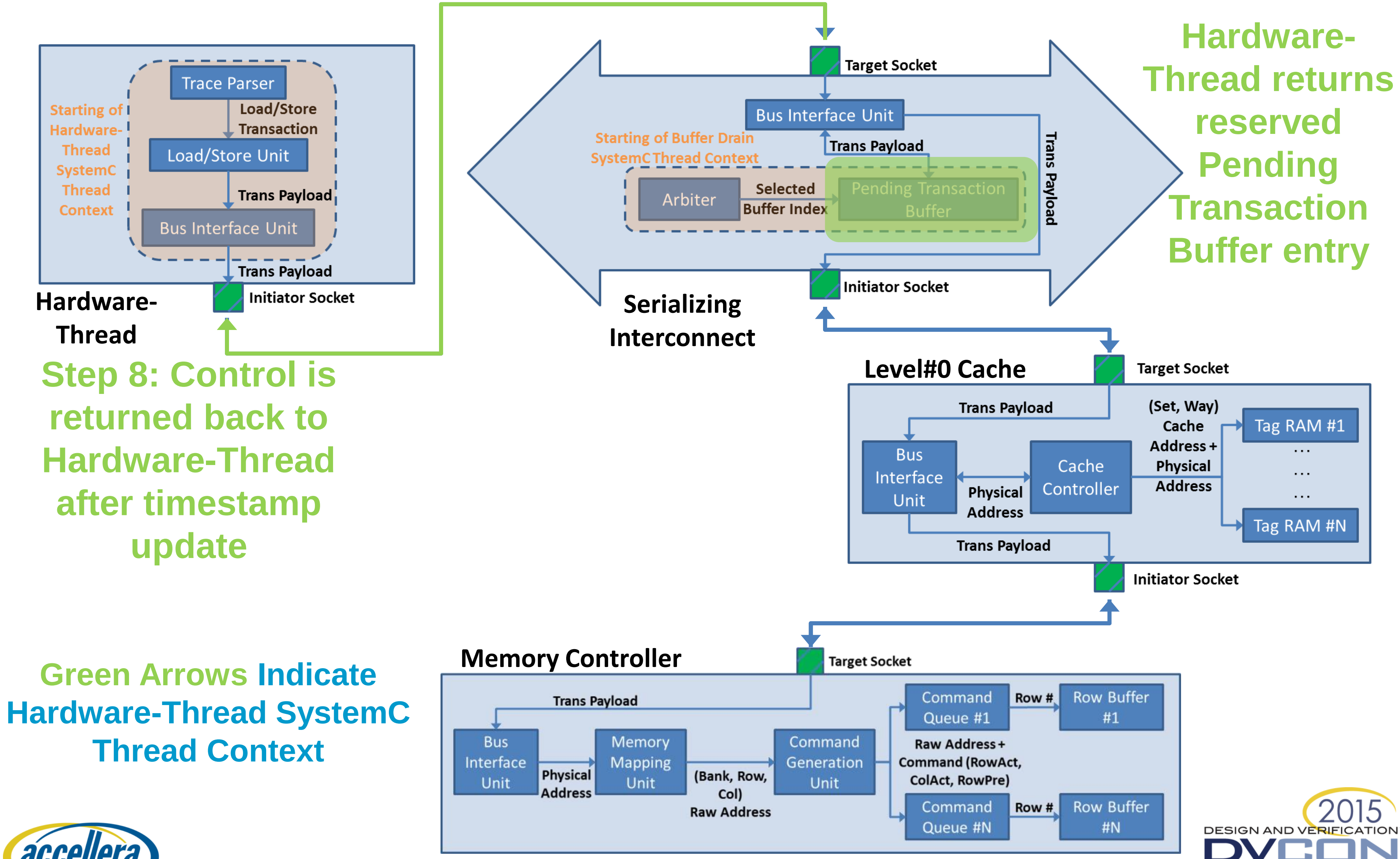
Cumulative Lumped-Delay Waited =

$t_{TagLookup} + t_{DRAM\text{-}access}$

Red Arrows Indicate Serializing Interconnect Buffer Drain SystemC Thread Context

2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Tracing a Memory Transaction



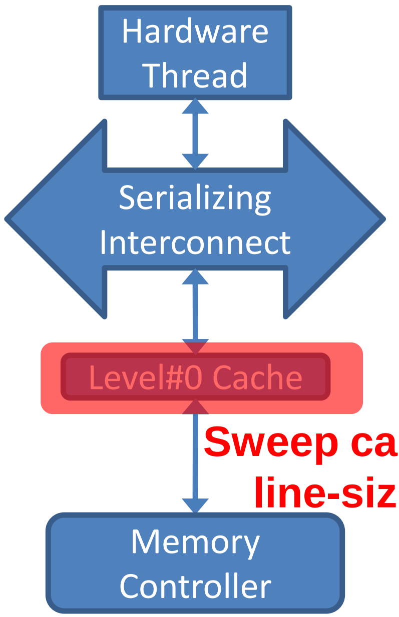
Hardware-Thread returns reserved Pending Transaction Buffer entry

Step 8: Control is returned back to Hardware-Thread after timestamp update

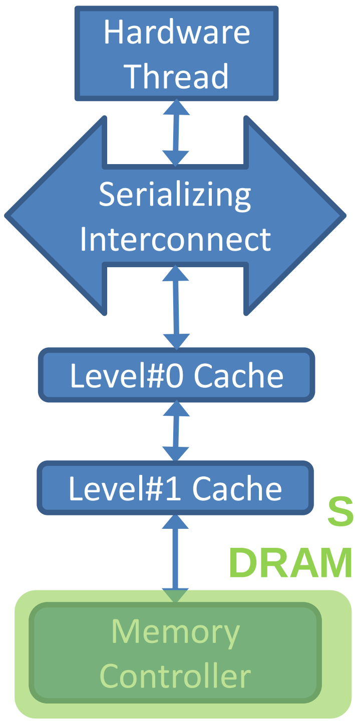
Green Arrows Indicate Hardware-Thread SystemC Thread Context

Simulator Validation

- Verify use-cases that signify fundamental tenets of memory-hierarchy

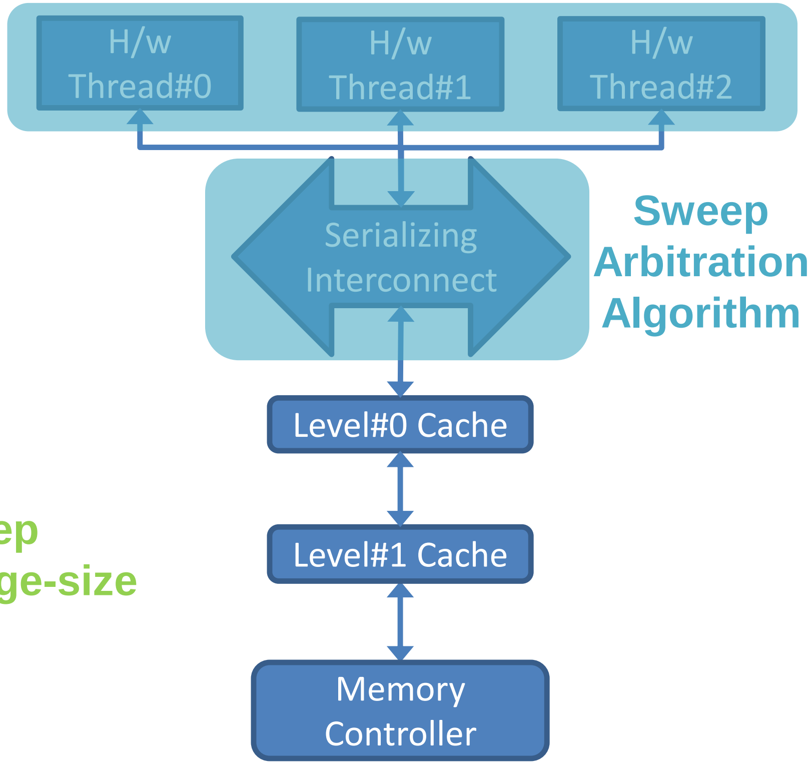


Sweep cache line-size



Sweep DRAM page-size

Multiple Hardware Threads



Test Scenario#1: Cache Use-Case

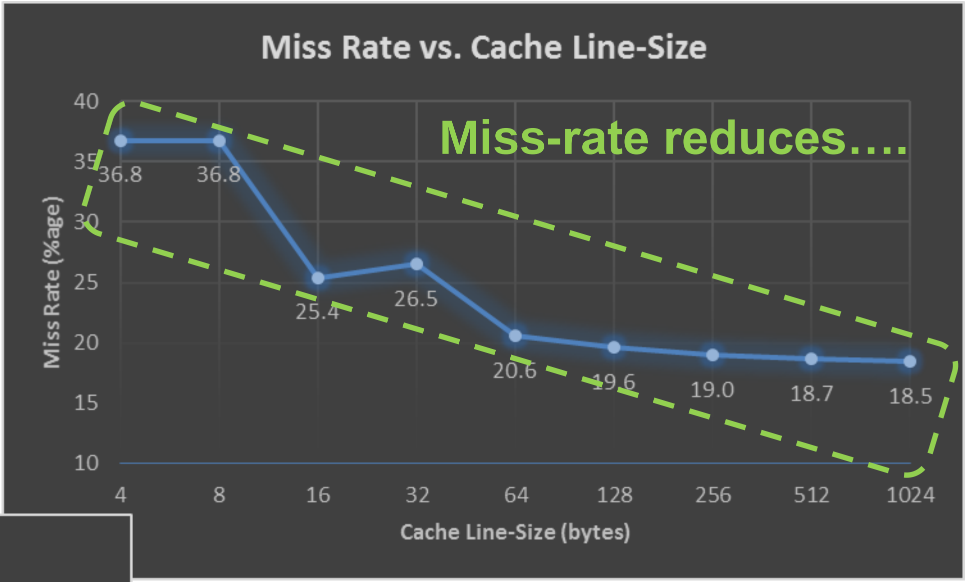
Test Scenario#2: Memory-Controller Use-Case

Test Scenario#3: Multithreading Use-Case

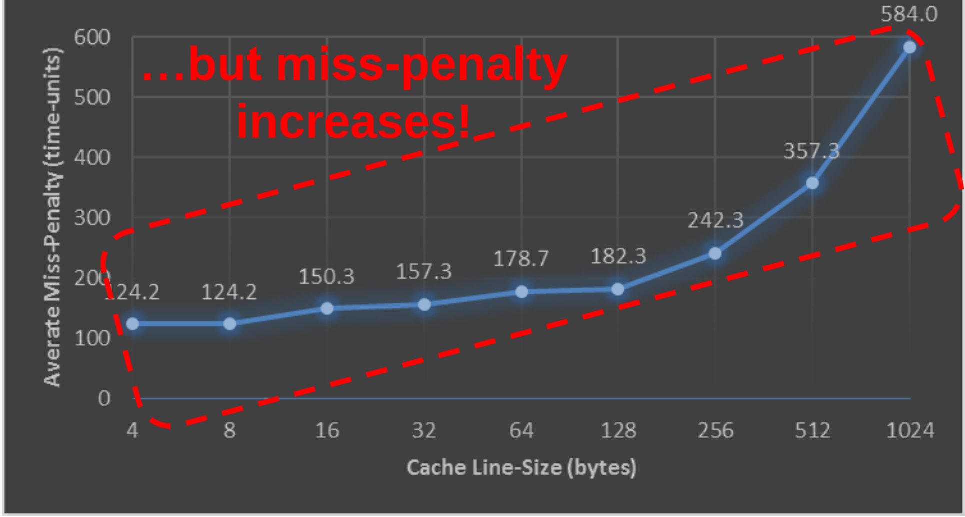
****Refer Appendix for Information about Benchmarks used and Paper for more Use-Cases**

Test Scenario 1: Sweep Cache Line-Size

Larger Cache-Line → Higher probability of Cache-hit (Spatial Locality) → Reduced Miss-Rate


Miss Rate vs. Cache Line-Size

Miss-rate reduces….

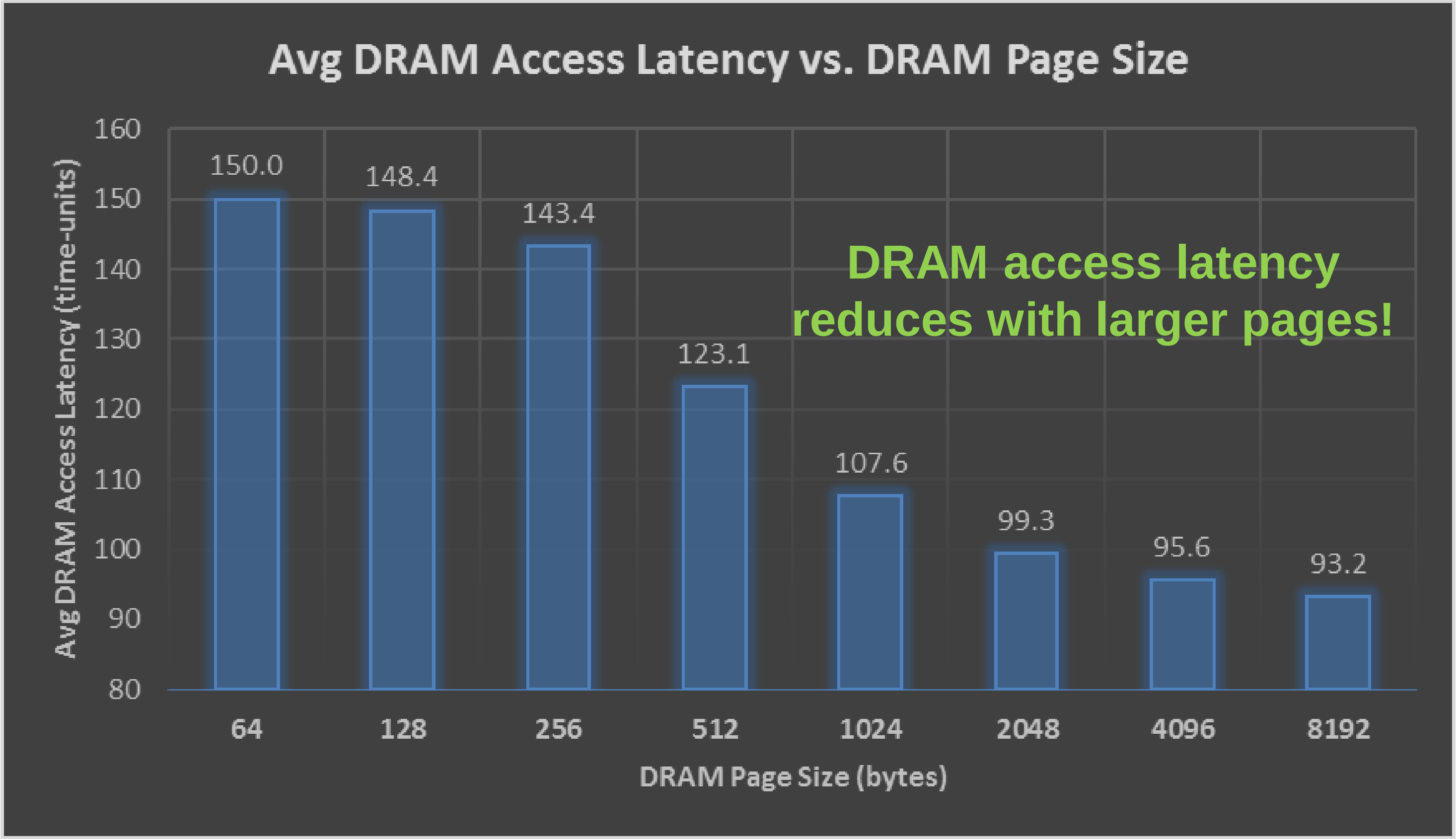

Average Miss Penalty vs. Cache Line-Size

…but miss-penalty increases!

Larger Cache-Line → More Data Fetched during Cache-miss → Increased Miss-Penalty

Test-Scenario 2: Larger DRAM pages reduce the average DRAM access latency

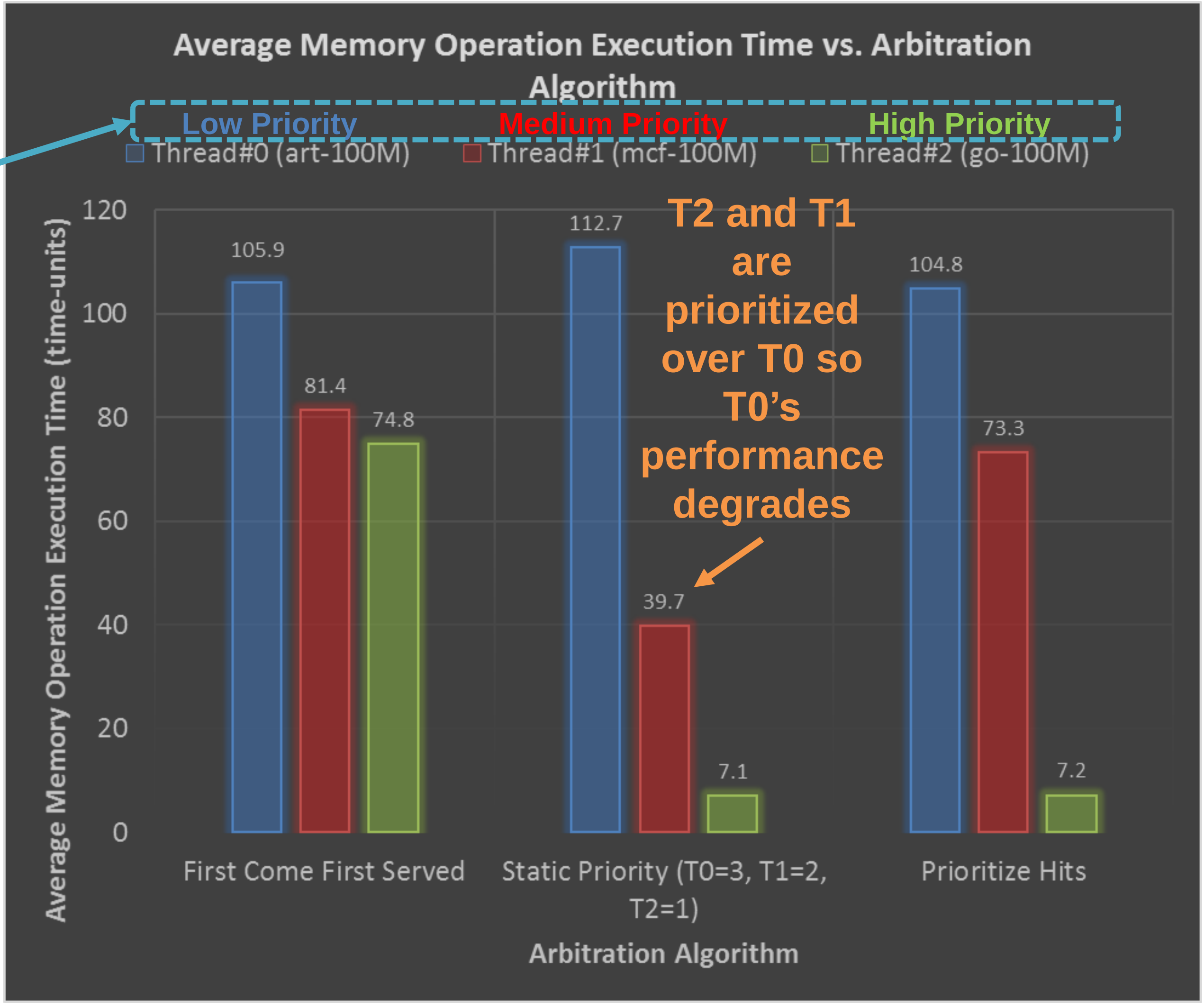
Larger DRAM page → Higher Probability of Row-Buffer hit (Spatial Locality) → Lower DRAM access-latency



DRAM access latency reduces with larger pages!

Test-Scenario 3: Prioritization of a thread is achieved at the cost of performance of other threads

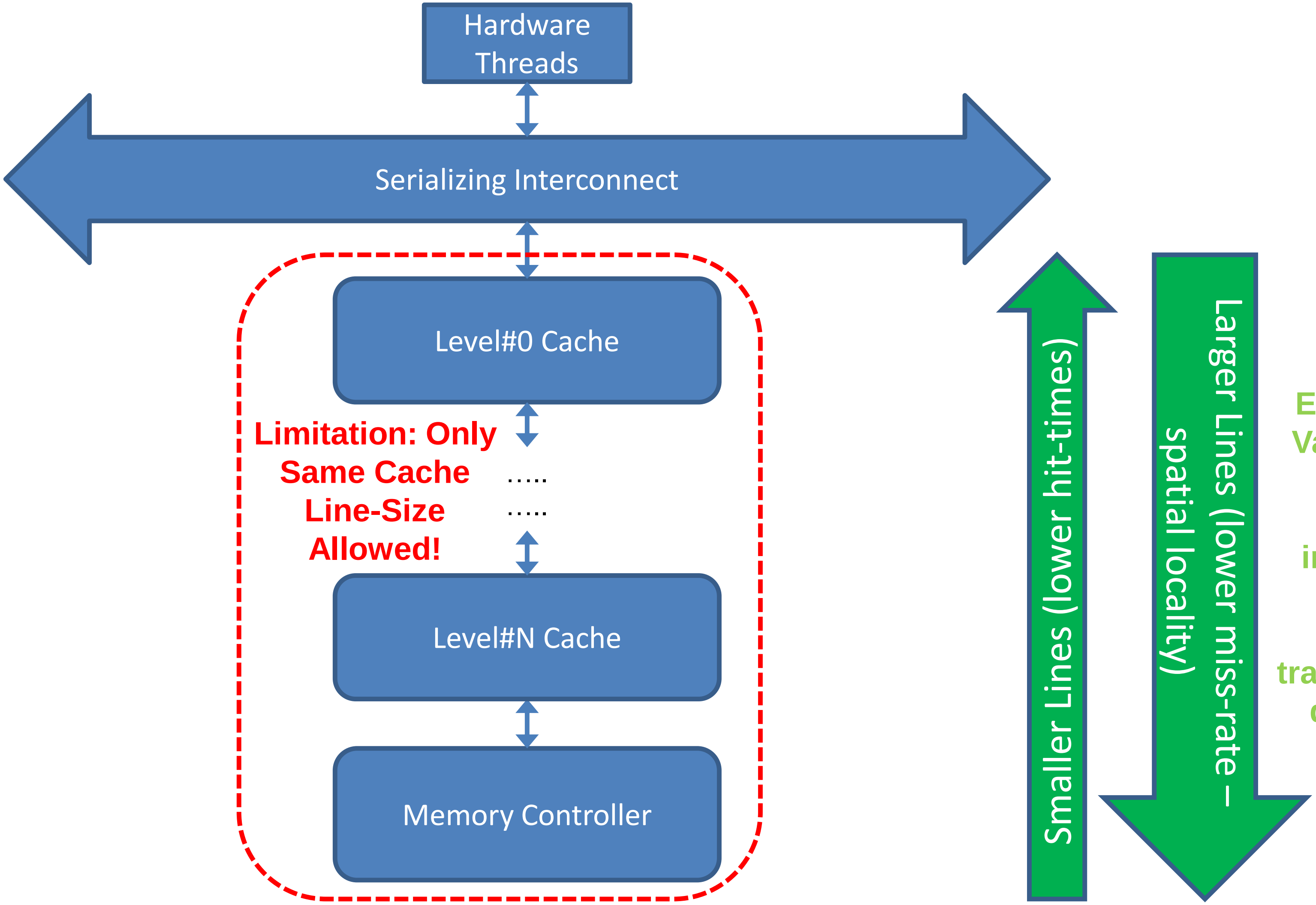
Priorities for Static Priority case



Low Priority Medium Priority High Priority

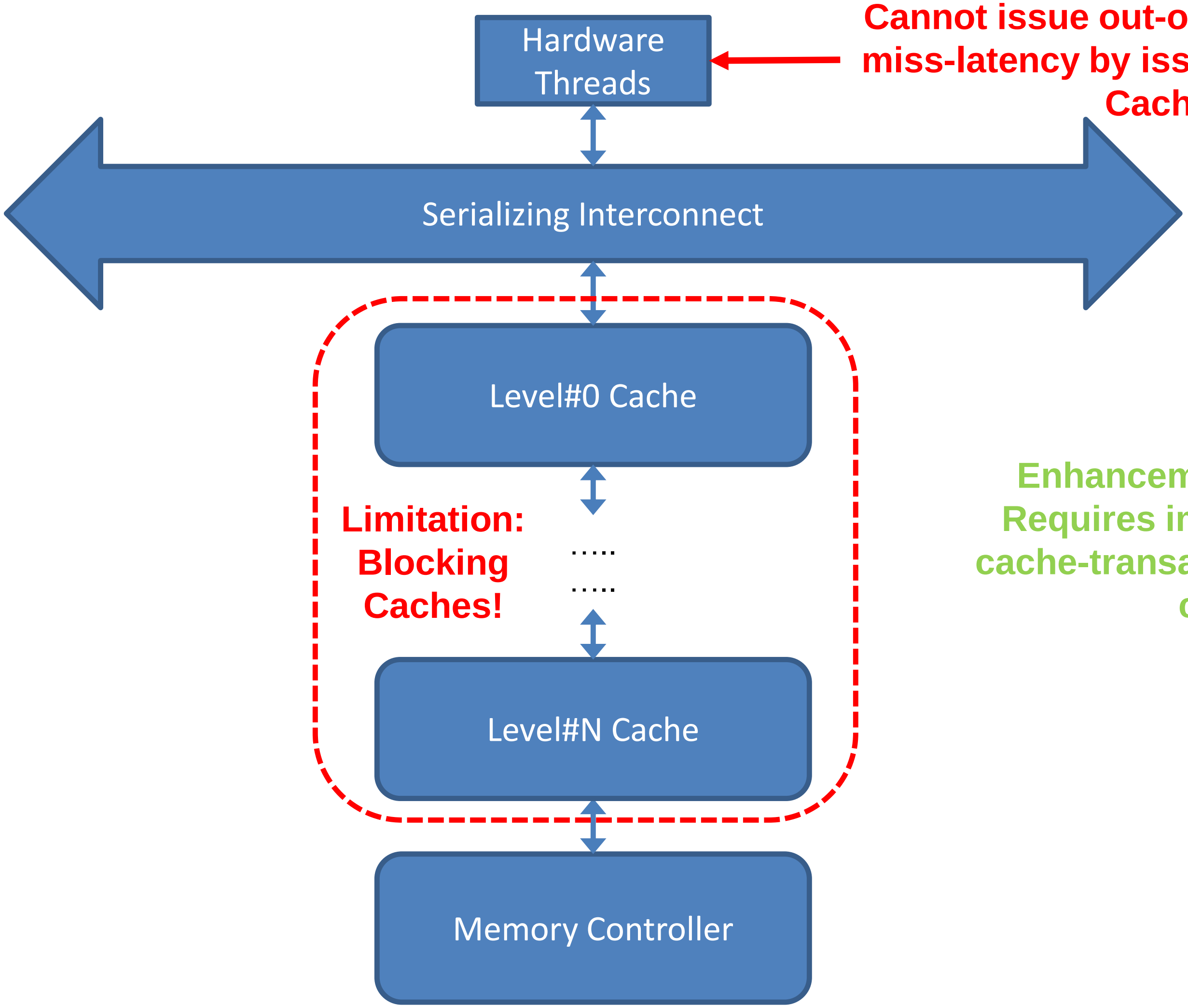
T2 and T1 are prioritized over T0 so T0's performance degrades

Current Limitations – Future Enhancement #1


Hardware Threads
Serializing Interconnect
Level#0 Cache
Limitation: Only Same Cache Line-Size Allowed!
.....
.....
Level#N Cache
Memory Controller
Smaller Lines (lower hit-times)
Larger Lines (lower miss-rate – spatial locality)

Enhancement: Variable Cache Line-Size -- Requires implementing logic to split/merge transactions that differ in line-sizes

Current Limitations – Future Enhancement #2

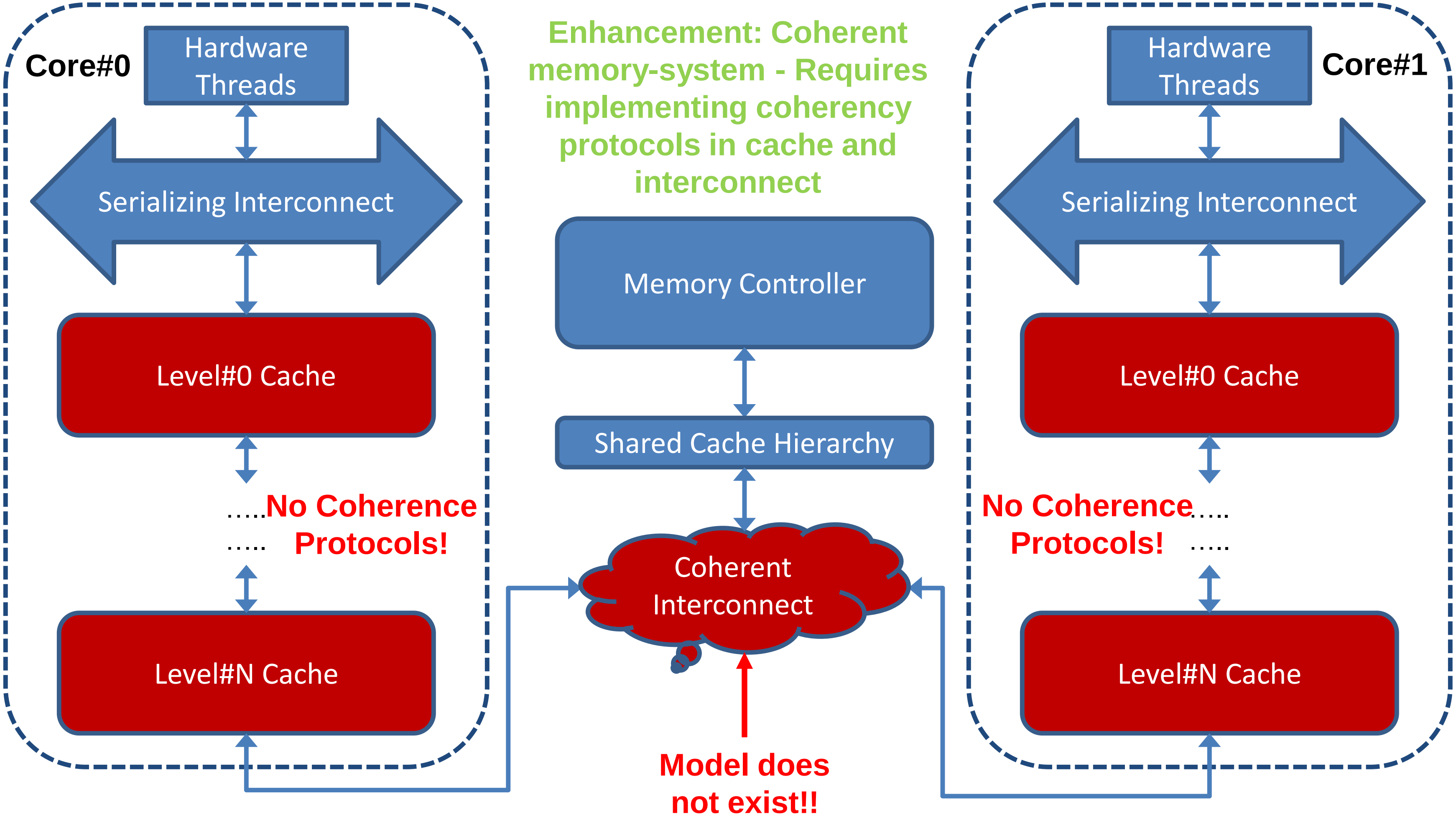


Cannot issue out-of-order transactions to hide miss-latency by issuing hit-after-miss because Cache is Blocked!

Limitation: Blocking Caches!

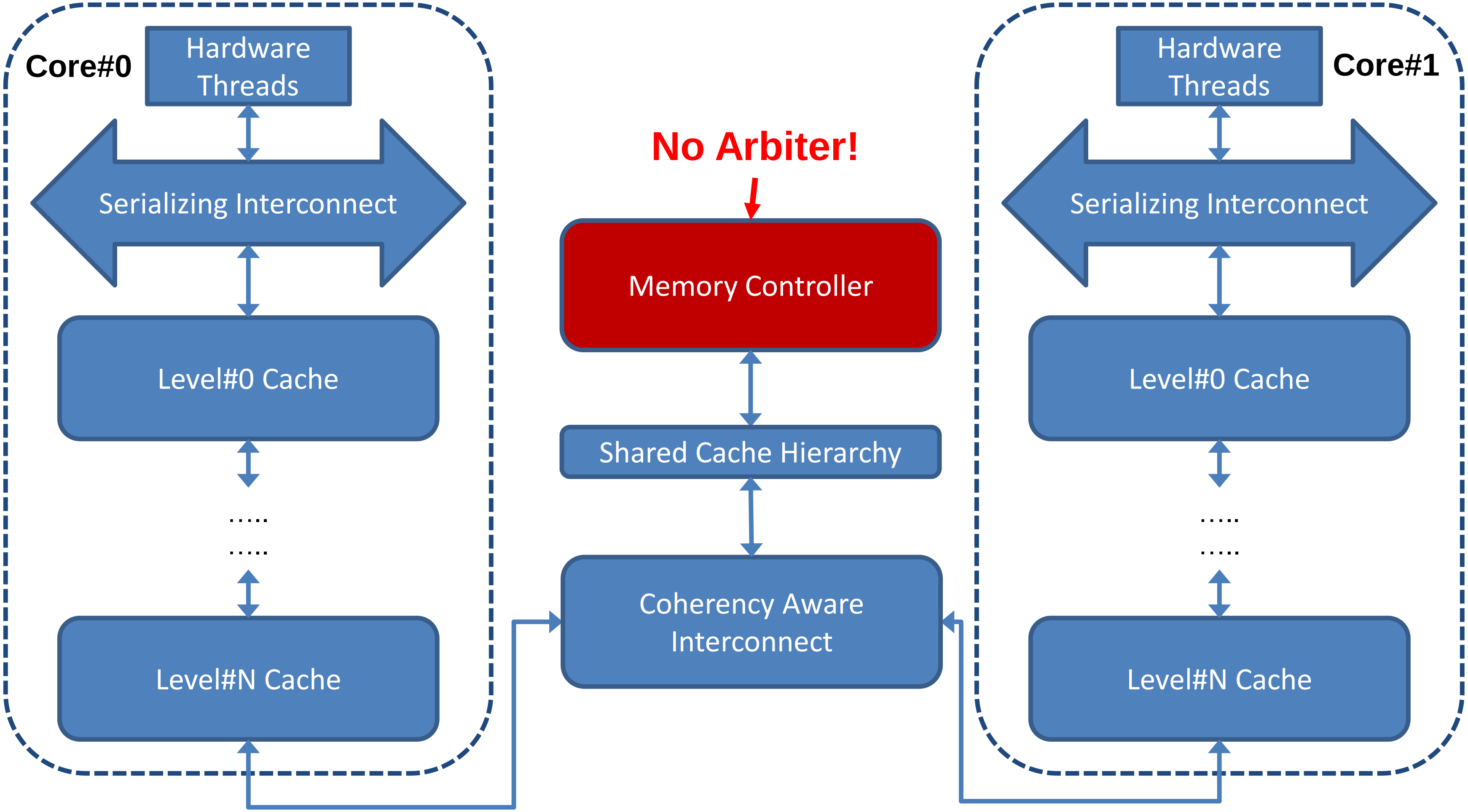
Enhancement: Non-Blocking Caches - Requires implementing logic to pipeline cache-transactions and allow reordering of cache-transactions

Current Limitations – Future Enhancement #3



Limitation: Cannot Explore Typical "Multicore" Architectures!

Current Limitations – Future Enhancement #4

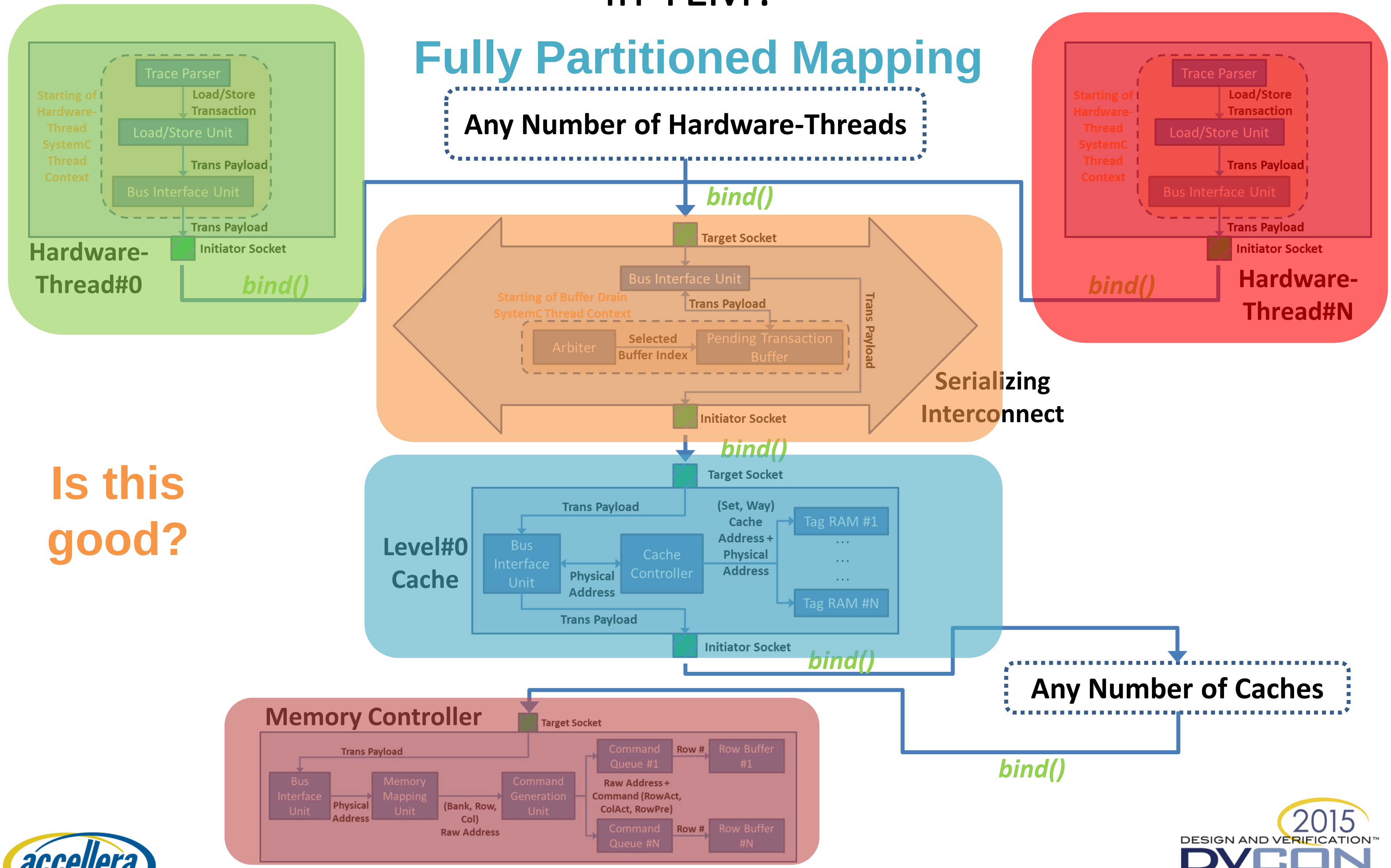


Enhancement: Arbitrated Coherency memory-system - Requires implementing an Arbiter and Arbitration Algorithms in Memory Controller

Limitation: Cannot Model Core Unfairness by re-ordering transactions from multiple cores! [17]

Conclusion – Challenge – No Thread/Process Scope Guidelines in TLM?

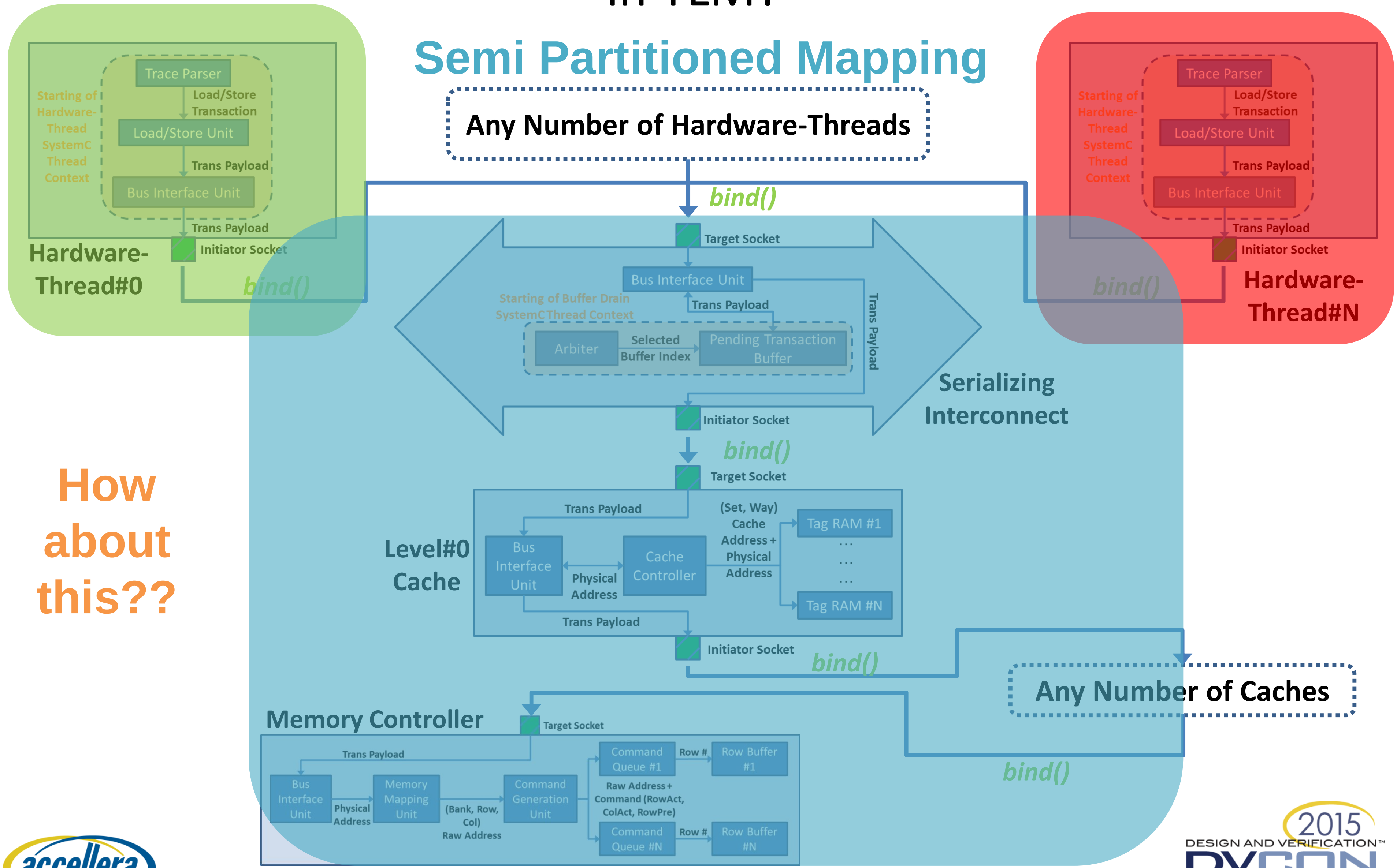
Fully Partitioned Mapping



Is this good?

Conclusion – Challenge – No Thread/Process Scope Guidelines in TLM?

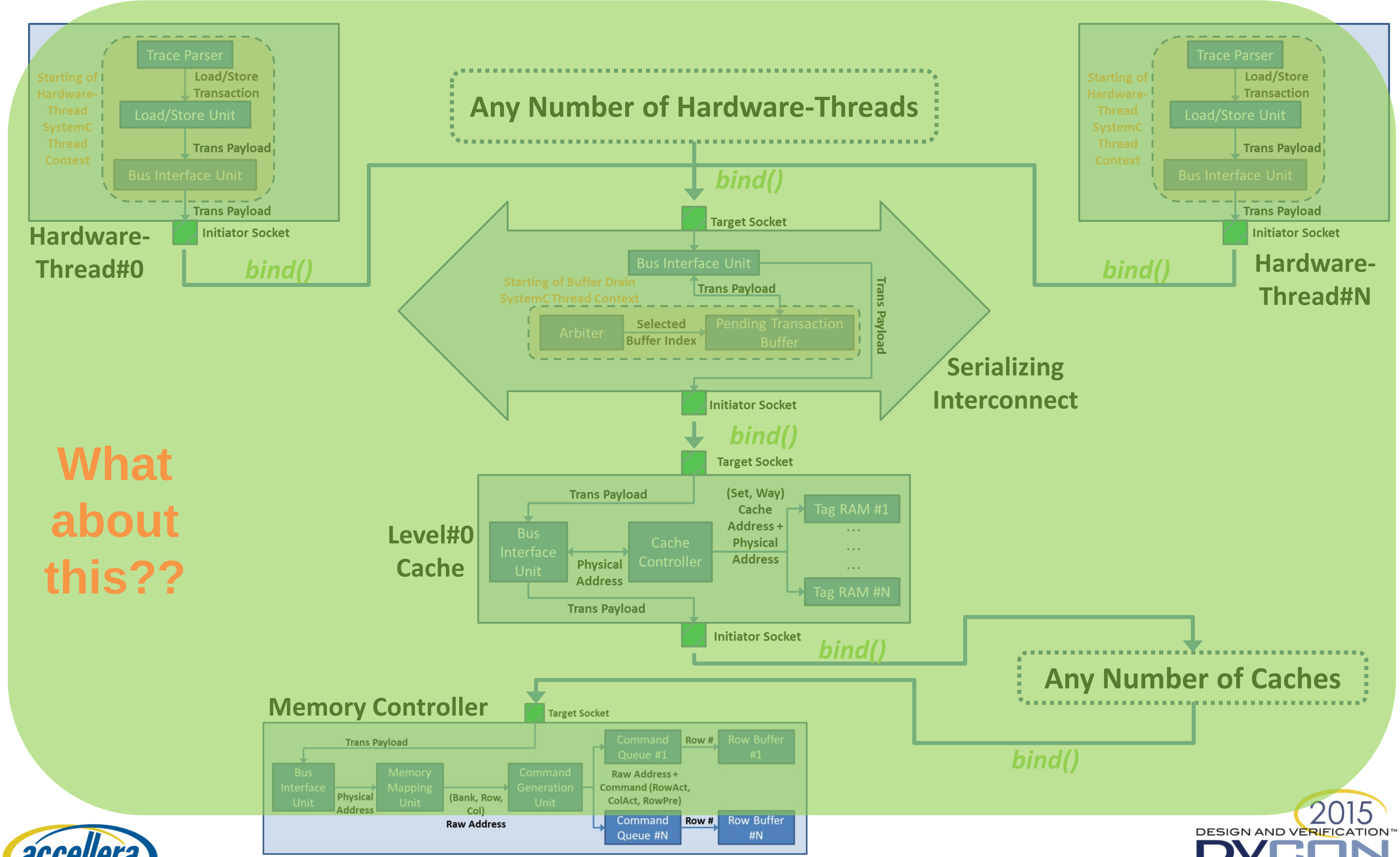
Semi Partitioned Mapping



How about this??

Conclusion – Challenge – No Thread/Process Scope Guidelines in TLM?

No Partitioning



What about this??

Conclusion – Our Experience using TLM

- Easy to transform an architecture-specification to an executable-model
- Separation of computation from communication enables flexible simulator design and architectural-exploration
- Modeling at Transaction-Level enables fast simulation with reasonable accuracy for exploration
- But, need a guideline to define thread/process scope!
- And, if I ever get down to improving it:
  - MeSSMArch v2.0 – A Memory System Simulator for **Multicore** Hardware Architectures ? ☺

References

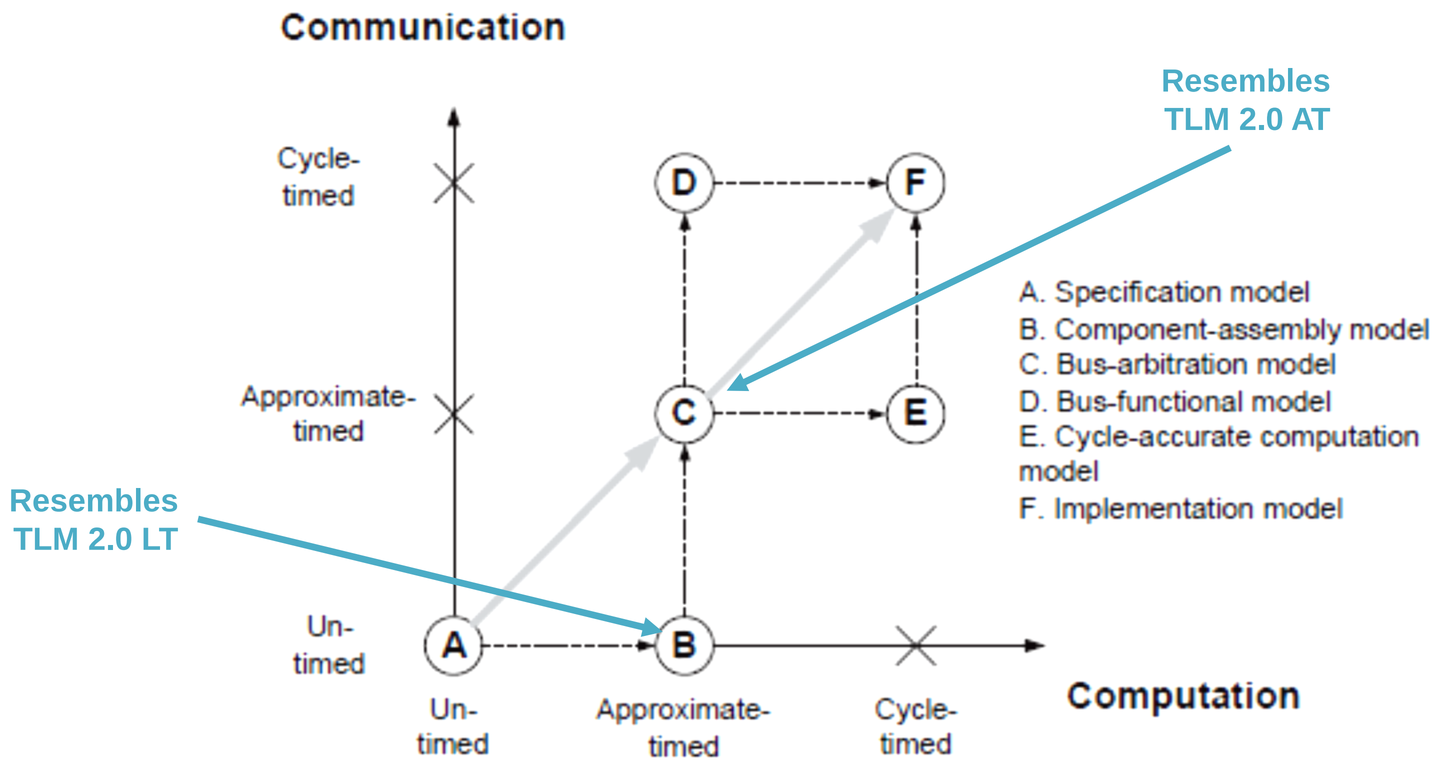
1. John L. Hennessy and David A. Patterson. 2011. Computer Architecture, Fifth Edition: A Quantitative Approach (5th ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA
2. Lukai Cai and Daniel Gajski. 2003. Transaction level modeling: an overview. In Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (CODES+ISSS '03). ACM, New York, NY, USA, 19-24
3. Frank Ghenassia. 2006. Transaction-Level Modeling with Systemc: Tlm Concepts and Applications for Embedded Systems. Springer-Verlag New York, Inc., Secaucus, NJ, USA.
4. "IEEE Standard for Standard SystemC Language Reference Manual," IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005) , vol., no., pp.1,638, Jan. 9 2012
5. Ye Lu; Sezer, S.; McCanny, J., "TLM2.0 based timing accurate modeling method for complex NoC systems," Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on , vol., no., pp.2900,2903, May 30 2010-June 2 2010
6. Menon, S.; Suryaprasad, J., "A pattern based methodology for the design and implementation of multiplexed Master-Slave devices at the system-level use-case: Modeling a Level-2 Cache IP module at transaction level," Networked Embedded Systems for Enterprise Applications (NESEA), 2010 IEEE International Conference on , vol., no., pp.1,6, 25-26 Nov. 2010
7. Benny Akesson. An introduction to SDRAM and memory controllers. URL: http://www.es.ele.tue.nl/premadona/files/akesson01.pdf
8. Onur Mutlu. Computer Architecture, Spring 2015, Lecture 21: Main Memory, Carnegie Mellon University. URL: http://www.ece.cmu.edu/~ece447/s15/lib/exe/fetch.php?media=onur-447-spring15-lecture21-main-memory-afterlecture.pdf
9. Milo M. K. Martin. Introduction to Computer Architecture, Fall 2010, Unit 10: Hardware Multithreading, University of Pennsylvania. URL: https://www.cis.upenn.edu/~milom/cis501-Fall10/lectures/10_multithreading.pdf
10. Amir Roth, "A High-Bandwidth Load-Store Unit for Single- and Multi-Threaded Processors", . January 2004
11. Intel® Hyper-Threading Technology: Technical User’s Guide. January 2003. URL: http://cache-www.intel.com/cd/00/00/01/77/17705_htt_user_guide.pdf
12. Daniel J. Sorin, Mark D. Hill, and David A. Wood. 2011. A Primer on Memory Consistency and Cache Coherence (1st ed.). Morgan & Claypool Publishers.
13. L. Lamport. 1979. How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. IEEE Trans. Comput. 28, 9 (September 1979), 690-691. DOI=10.1109/TC.1979.1675439 http://dx.doi.org/10.1109/TC.1979.1675439
14. Adaptive Resonance Theory 2 (ART 2) Benchmark. URL: https://www.spec.org/cpu2000/CFP2000/179.art/docs/179.art.html
15. MCF Benchmark. URL: https://www.spec.org/cpu2006/Docs/429.mcf.html
16. Game of Go (Go) Benchmark. URL: https://www.spec.org/cpu2006/Docs/445.gobmk.html
17. Mutlu, O.; Moscibroda, T., "Parallelism-Aware Batch Scheduling: Enabling High-Performance and Fair Shared Memory Controllers," Micro, IEEE , vol.29, no.1, pp.22,32, Jan.-Feb. 2009, doi: 10.1109/MM.2009.12

Thank You! ☺

Questions/Thoughts/Comments?

APPENDIX

Design-space of Widely used System Models

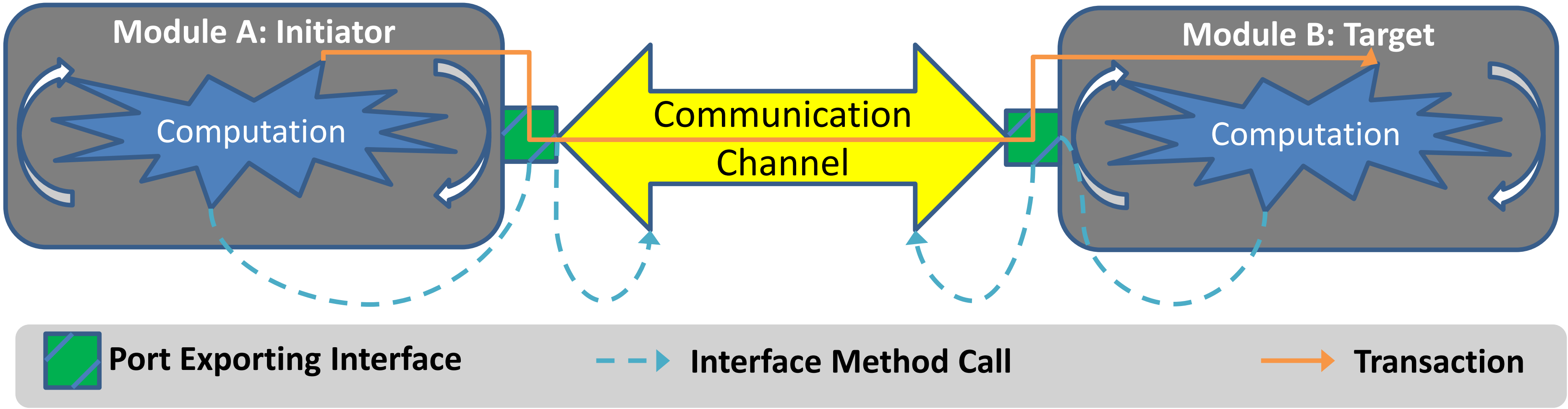


A. Specification model
B. Component-assembly model
C. Bus-arbitration model
D. Bus-functional model
E. Cycle-accurate computation model
F. Implementation model

Graph Showing the Design-space of Widely used System Models [2]

TLM - Overview

- Separate Computation from Communication
- TLM 2.0 LT – {Timed Computation} + {Untimed Communication}
- TLM 2.0 AT – Timed {Computation + Communication}



Advantages of the TLM Methodology

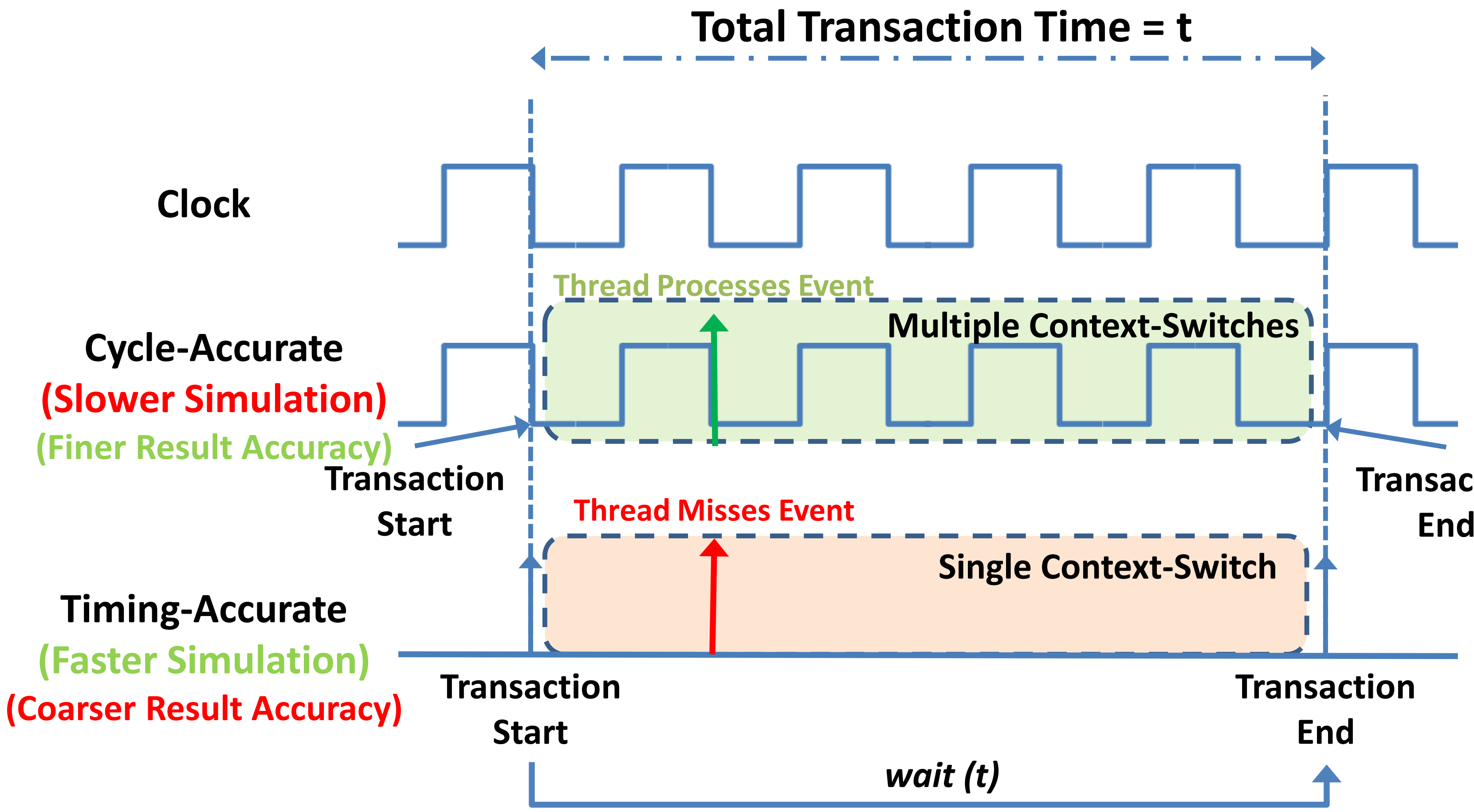
- Early Software Development
  - Functional TLM platform can be constructed from system-architecture specification – aids pre-silicon software development
- Architectural Analysis
  - Timed TLM platforms comprising of parameterized components can be used for swift architectural-exploration
- Functional Verification
  - TLM platforms represent an executable specification, functional o/p can be compared with RTL for verification

This Information is Borrowed from [3]

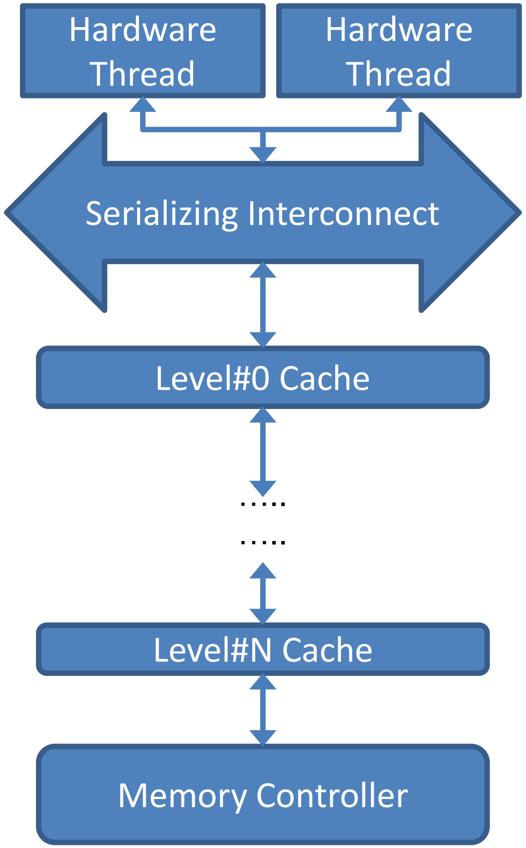
TLM – Salient Features/Guidelines

1. Separate Module Computation from Inter-module Communication
   - Model computation details inside the process
   - Model communication details inside the channel
2. Avoid Modeling Functionality of Micro-architectural Features
   - Capture their effects through timing information
3. Simulate data exchange at Transaction-Level
   - Raise level of timing-abstraction from cycle-accuracy to timing-accuracy via lumped delays

Timing-Accuracy vs. Cycle-Accuracy

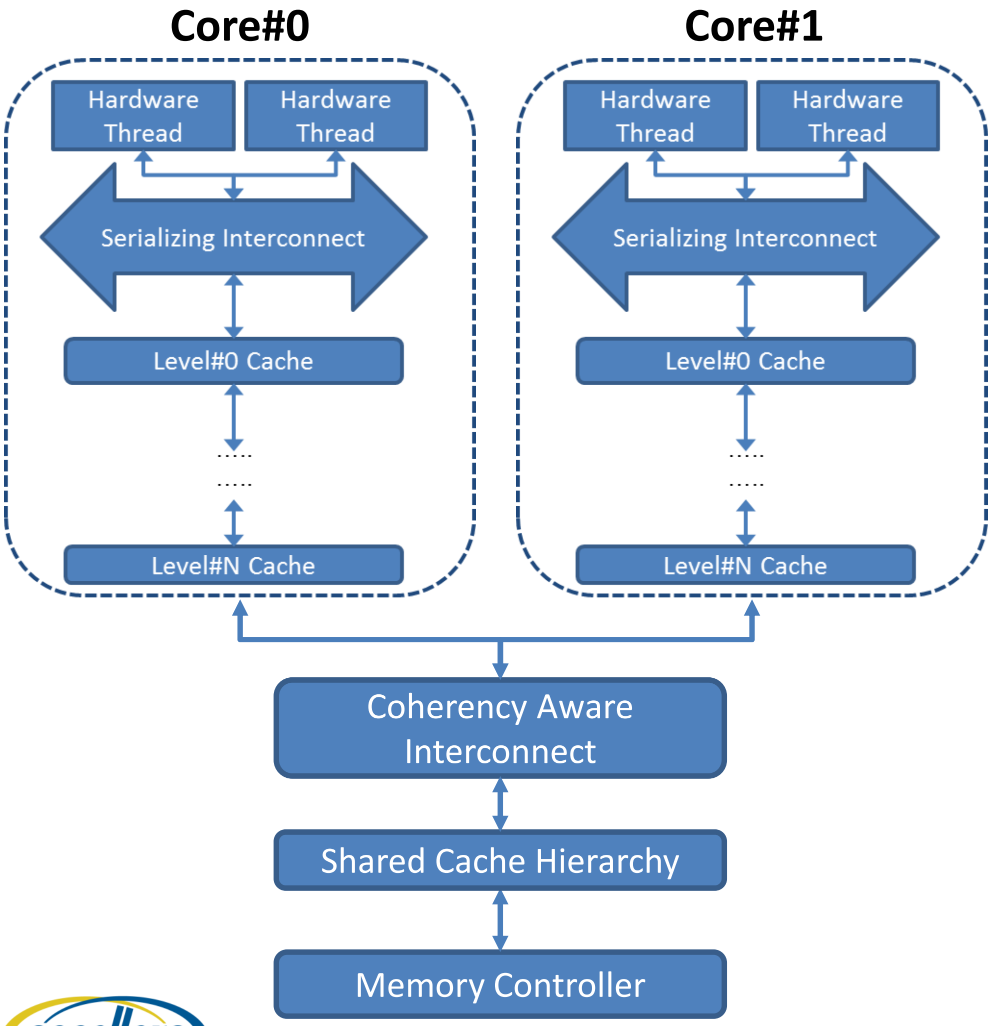


Hardware Multithreading Architectures



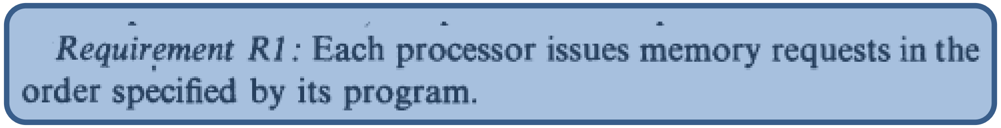
- Multiple Threads share a Unified Memory-Hierarchy
  - Thread scheduling may be *coarse-grained, fine-grained or simultaneous-multithreaded (SMT)*
- Replicate "software state" for each thread (PC, registers)
- Share "hardware state" (caches, branch predictors etc.)
- Reason: improve exploitation of ILP
  - Hardware may provide many execution resources
  - Single instruction stream cannot fully utilize those resources
  - Share resources between multiple threads to increase utilization
- No memory-coherence issues since hierarchy is shared!

Multicore Architectures



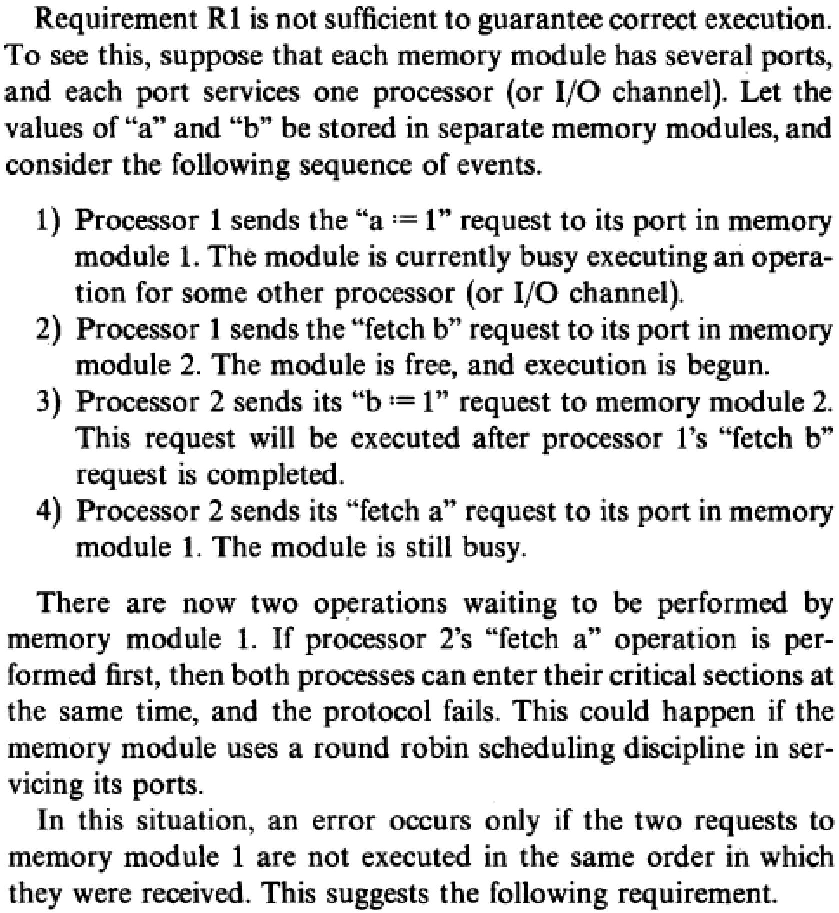
- Multiple "cores" share a memory hierarchy
- "Cores" contain memory hierarchies
  - May also contain multiple threads
- Reason: improve exploitation of TLP
  - Replicate hardware "cores" to enable true parallelism by providing a "private" memory-hierarchy
- Memory-coherence issues arise when private hierarchies do not present the same view of memory
  - Need coherence protocols

Sequential Consistency - Theory



Requirement R1: Each processor issues memory requests in the order specified by its program.

1. Transactions from single-processor are in-order

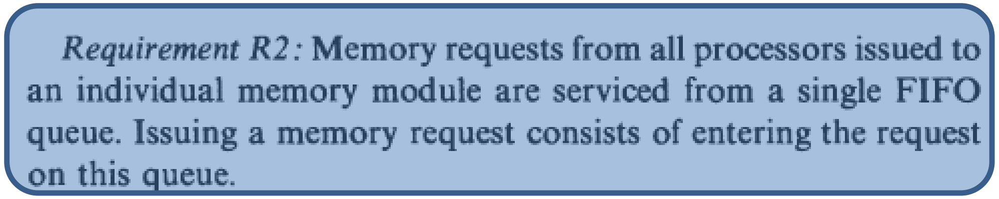


Requirement R1 is not sufficient to guarantee correct execution. To see this, suppose that each memory module has several ports, and each port services one processor (or I/O channel). Let the values of "a" and "b" be stored in separate memory modules, and consider the following sequence of events.

1) Processor 1 sends the "a := 1" request to its port in memory module 1. The module is currently busy executing an operation for some other processor (or I/O channel).
2) Processor 1 sends the "fetch b" request to its port in memory module 2. The module is free, and execution is begun.
3) Processor 2 sends its "b := 1" request to memory module 2. This request will be executed after processor 1's "fetch b" request is completed.
4) Processor 2 sends its "fetch a" request to its port in memory module 1. The module is still busy.

There are now two operations waiting to be performed by memory module 1. If processor 2's "fetch a" operation is performed first, then both processes can enter their critical sections at the same time, and the protocol fails. This could happen if the memory module uses a round robin scheduling discipline in servicing its ports.

In this situation, an error occurs only if the two requests to memory module 1 are not executed in the same order in which they were received. This suggests the following requirement.



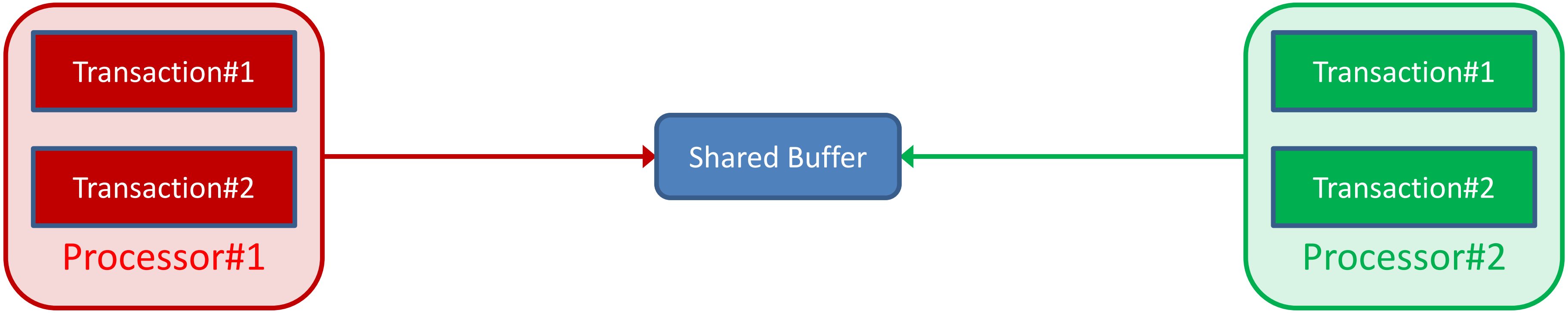
Requirement R2: Memory requests from all processors issued to an individual memory module are serviced from a single FIFO queue. Issuing a memory request consists of entering the request on this queue.

2. Transactions from different processors may be interleaved

Snapshots of actual text in paper borrowed from [13]

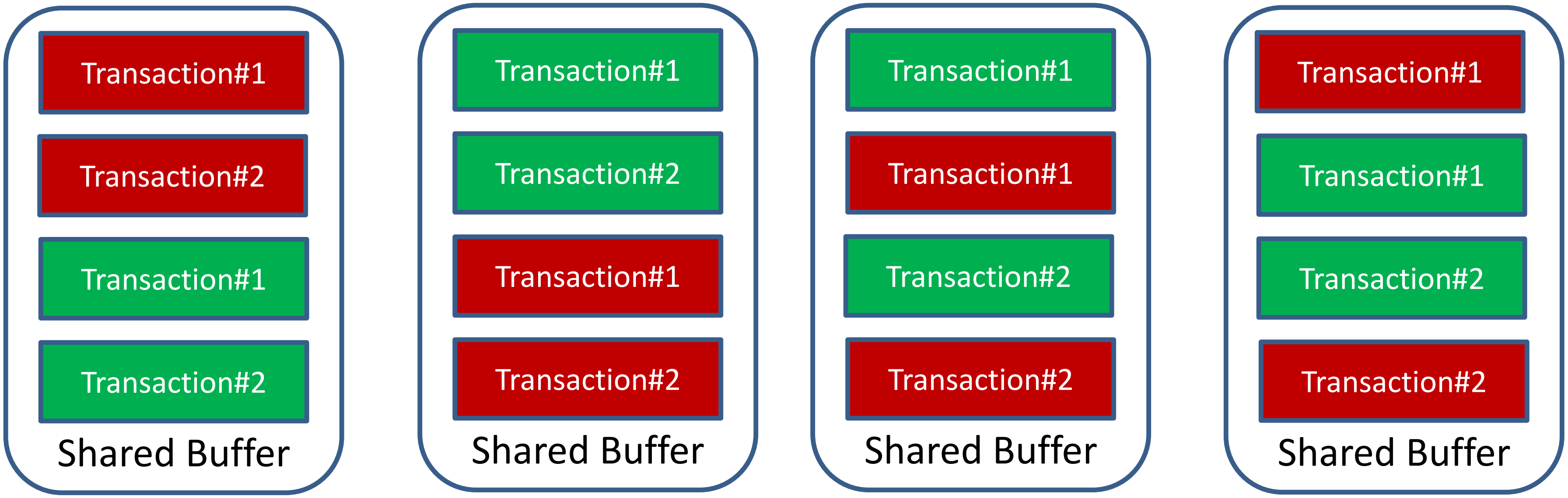
Sequential Consistency – Illustration

Assume that each Processor needs to send it's transactions in program-order to the Shared Buffer



What combinations of transaction ordering are possible in the Shared Buffer?

Sequential Consistency – Illustration



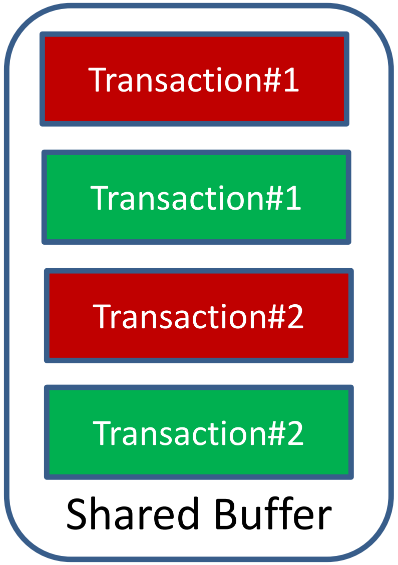
🟥 Transaction#1
🟥 Transaction#2
🟩 Transaction#1
🟩 Transaction#2
Shared Buffer

🟩 Transaction#1
🟩 Transaction#2
🟥 Transaction#1
🟥 Transaction#2
Shared Buffer

🟩 Transaction#1
🟥 Transaction#1
🟩 Transaction#2
🟥 Transaction#2
Shared Buffer

🟥 Transaction#1
🟩 Transaction#1
🟩 Transaction#2
🟥 Transaction#2
Shared Buffer

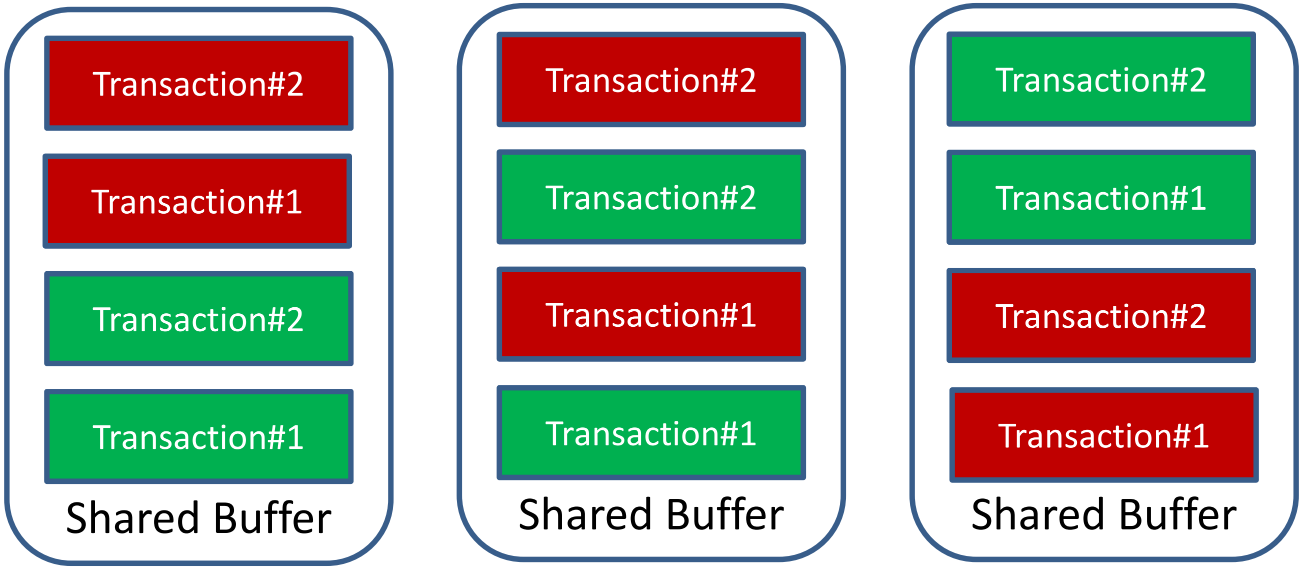
All of these transaction orderings are sequentially consistent!



🟥 Transaction#1
🟩 Transaction#1
🟥 Transaction#2
🟩 Transaction#2
Shared Buffer

1. **Transactions from each processor are in program-order**
2. **Transactions between processors can be interleaved**

Sequential Consistency – Illustration



🟥 Transaction#2
🟥 Transaction#1
🟩 Transaction#2
🟩 Transaction#1
Shared Buffer

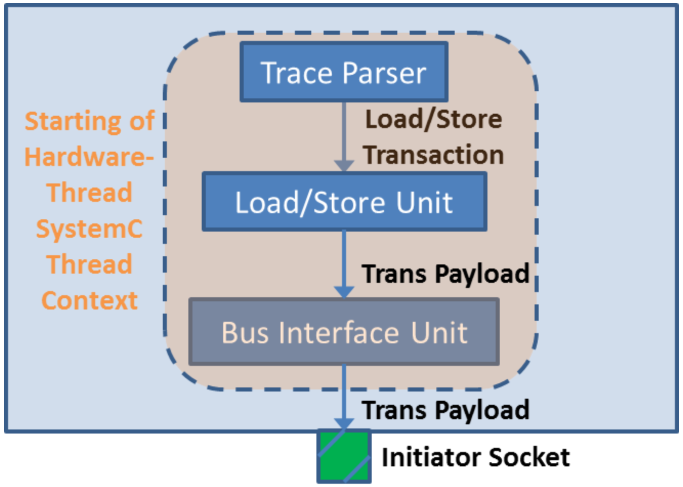
🟥 Transaction#2
🟩 Transaction#2
🟥 Transaction#1
🟩 Transaction#1
Shared Buffer

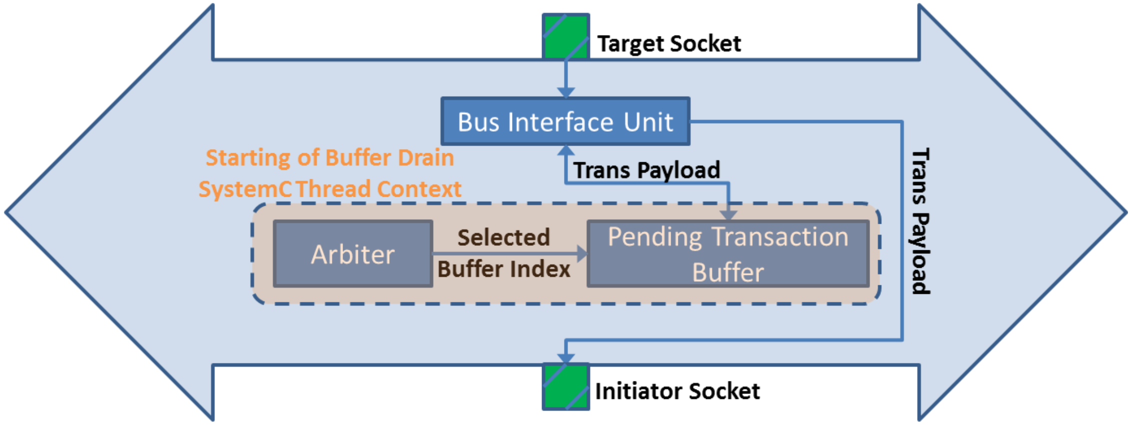
🟩 Transaction#2
🟩 Transaction#1
🟥 Transaction#2
🟥 Transaction#1
Shared Buffer

…. and the other remaining combinations

Not Sequentially Consistent! – From the perspective of each processor, it's transactions have been re-ordered!

Why is MeSSMArch Sequentially Consistent?





Load/Store Unit of Hardware-Thread has FIFO of single-entry depth only → Since Parser reads benchmark in-order, impossible for Load/Store unit to re-order them!

This satisfies requirement 1

Arbiter in Serializing Interconnect may re-order transactions between hardware-threads, but cannot re-order transactions from the same hardware-thread, because the hardware-thread always issues them in-order!

This satisfies requirement 2

Memory Controller – Configuration Parameters

Parameter	Unit/ Options	Description
DRAM Page-size	Bytes	Size of a DRAM page – effectively Row-buffer size
Cache Line-Size	Bytes	Size of a cache-line
# of DRAM Banks	<NA>	# of banks in a multi-banked DRAM
Memory Data-bus Size	Bits	Size of the data-bus connecting memory controller and DRAM
Memory Timing Parameters	Cycles	tRCD, tCL, tRP
DRAM Memory Type	Sync/Async	Affects derived tRAS memory-timing parameter
Physical Address to Raw Address Mapping	Byte Interleaved/Bank Seq/Row Seq	Affects decoding of Physical Address to Raw Address
Clock Period	Nanosecs	Time-period of a clock-cycle

Green entries indicate micro-architectural features

Serializing Interconnect – Configuration Parameters

Parameter	Unit/ Options	Description
# of Outstanding Transactions	Transactions	Denotes the size of the Pending Transaction Buffer
Arbitration Algorithm	First-pending/FCFS/Static Priority/Prioritize Hits	Algorithm determining the transaction picked from the Pending Transaction Buffer for injection downstream

Hardware-Thread – Configuration Parameters

Parameter	Unit/ Options	Description
Benchmark File	Path to file	Benchmark file to read, parse and execute
Master Priority	Integer (lower number implies higher priority)	Static-priority for the thread (used only for the static-prioritization arbitration algorithm)

Generic Cache – Performance Counters

Statistic	Unit
# of caches-hits and cache-misses (further classified into reads/writes)	# of transactions
# of overhead-writes generated by write-policy (write-through/write-back)	# of transactions
Miss-classification into capacity/compulsory/conflict	# of transactions and %ages
Way-prediction accuracy and inaccuracy	# of transactions and %ages
Cache-bandwidth (effective and wasted)	Bytes per time unit

Memory Controller – Performance Counters

Statistic	Unit
Row-buffer hit-rate and miss-rate (per-bank and average)	%ages and # of transactions
Average memory-transaction latency	time units
Average memory-bandwidth	Bytes per time unit
# of row-activates, col-activates, row-precharges per bank	<NA>

Hardware-Thread – Performance Counters

Statistic	Unit
# of memory-transactions issued	<NA>
Total Thread Execution Time	time-units
Bus-contention time, Bus-queuing time, Effective-execution time	time-units and %ages
Average memory-transaction execution time	time-units

Benchmarks used for Use-Case Verification

- Collect dynamic execution-trace for each SPEC CPU benchmark
- Pick first 100-million instructions
- Simulate all memory-transactions present in the first 100-million instructions

SPEC CPU Benchmark	# of Memory Transactions	Brief Description
art-100M	19888117	Adaptive Resonance Theory – Image Recognition/Neural Networks [14]
mcf-100M	32362081	Single-Depot Vehicle Scheduling [15]
go-100M	35497321	Artificial Intelligence: Game of Go [16]

The End