DACOTA: Post-silicon validation of the memory subsystem in multi-core designs

Andrew DeOrio
Ilya Wagner
Valeria Bertacco

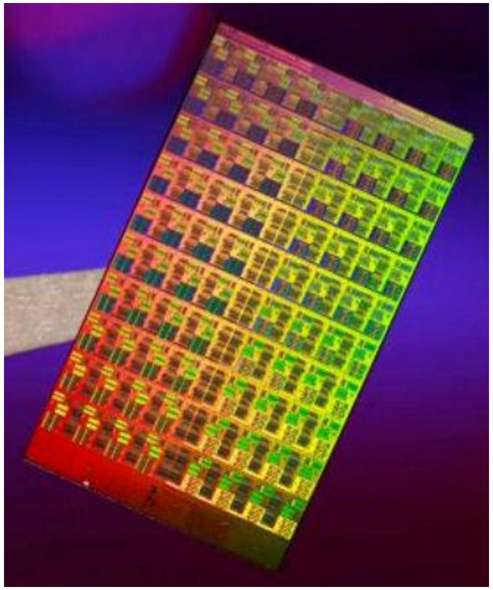
Advanced Computer
Architecture Laboratory
University of Michigan, Ann Arbor

HPCA 2009

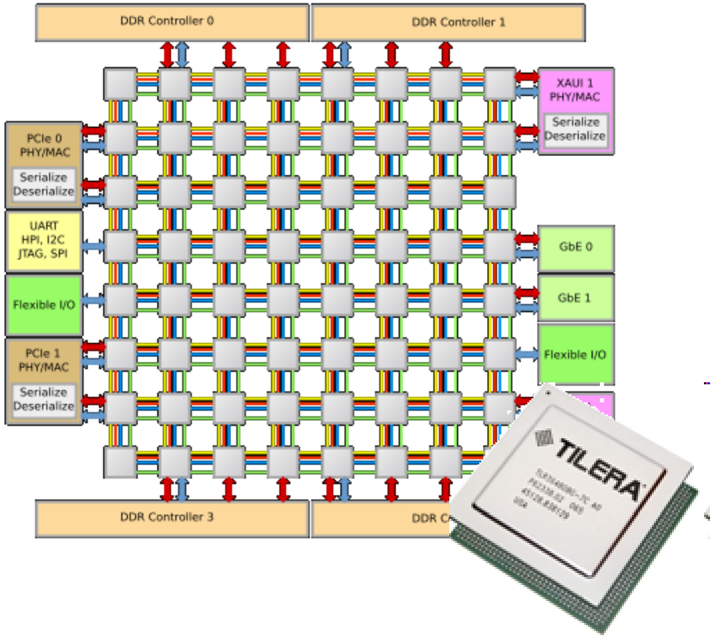
Multi-core Designs

- Many simple processors
- Communicate through interconnect network

Intel Polaris

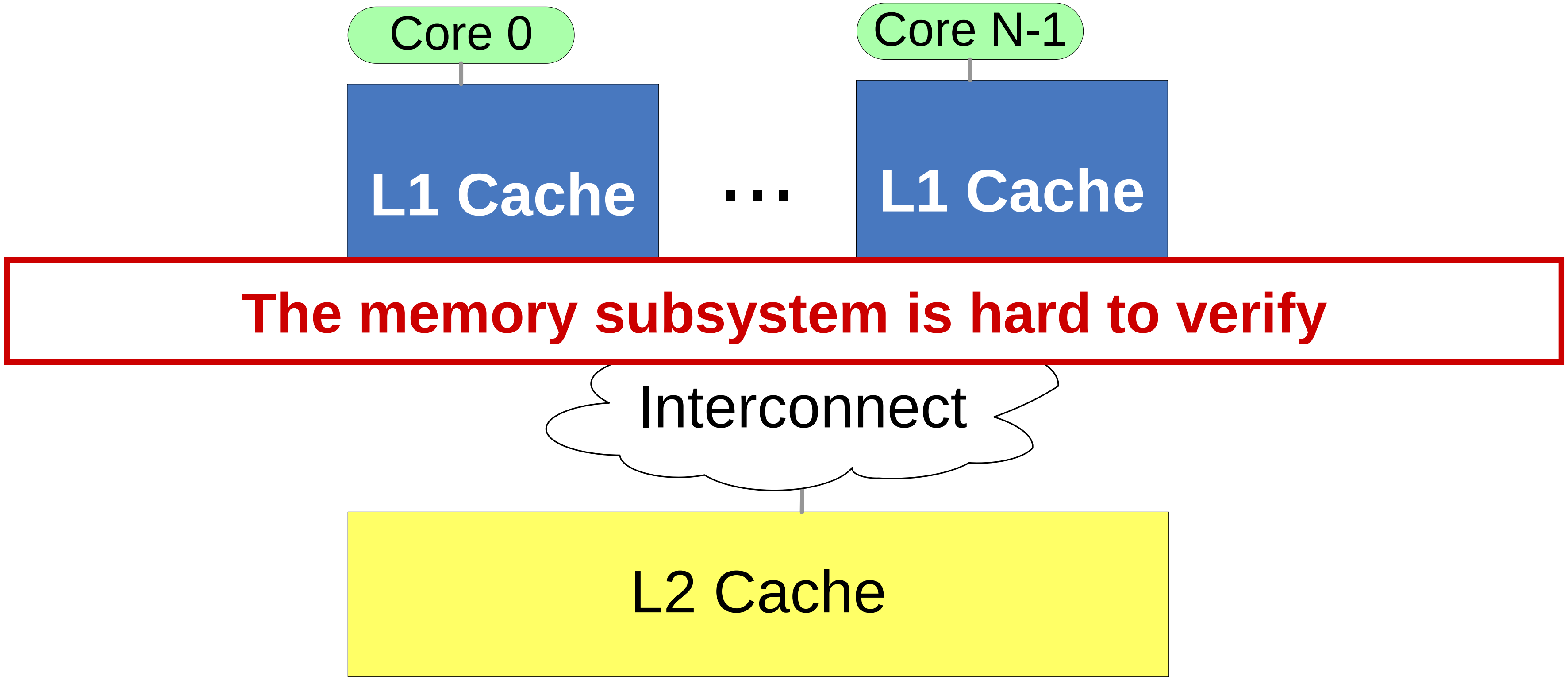


Tilera TILE64



Complex Multi-core: Memory Subsystem

- **Cache coherence:** the ordering of operations to a single cache line
- **Memory consistency:** controls the ordering of operations among different memory addresses



Core 0

Core N-1

L1 Cache

…

L1 Cache

The memory subsystem is hard to verify

Interconnect

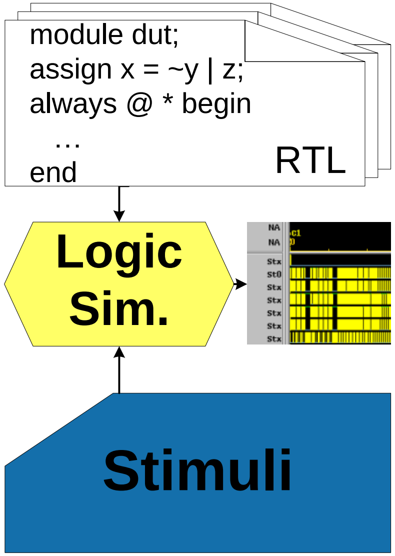
L2 Cache

The Verification Landscape

Pre-Silicon

bugs exposed: 98%

effort: 70%



```
module dut;
assign x = ~y | z;
always @ * begin
   …
end
```

RTL

Logic Sim.

Stimuli

- Slow: ~Hz
- Stimuli generators
- Random testers
- Formal verification

Post-Silicon

bugs exposed: 2%

effort: 30%

- Fast: at-speed
- Early HW prototypes
- Hard-to-find bugs
- Relatively new technology
  - Ad-hoc

Runtime

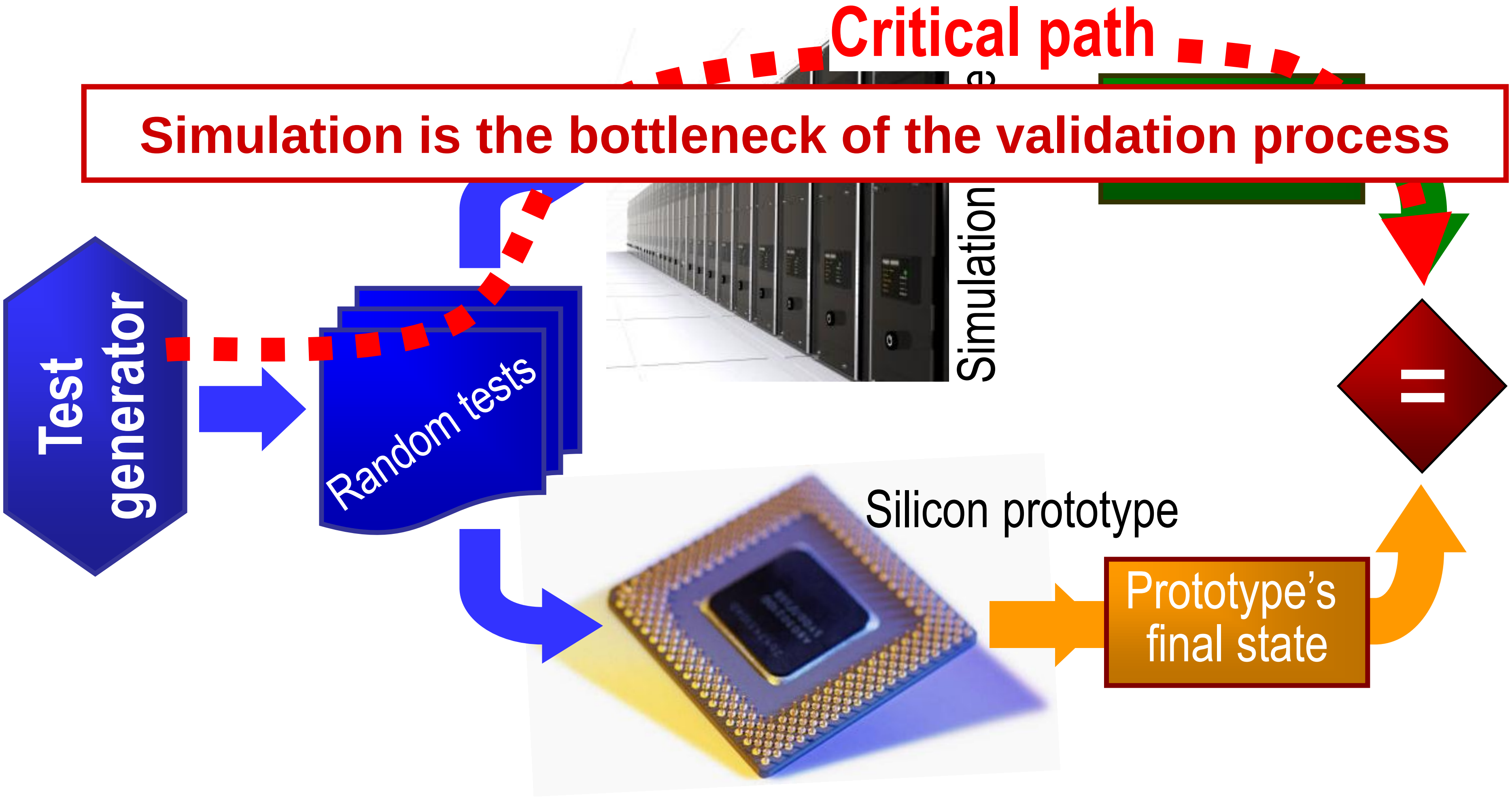
bugs exposed: <1%

effort: 0%

bug detected

- Fast: at-speed
- Research ideas
  - Austin, Malik, Sorin
- Microcode patching
  - Intel, AMD

Post-Silicon Validation Today



Critical path

Simulation is the bottleneck of the validation process

Test generator

Random tests

Simulation

Silicon prototype

Prototype's final state

=

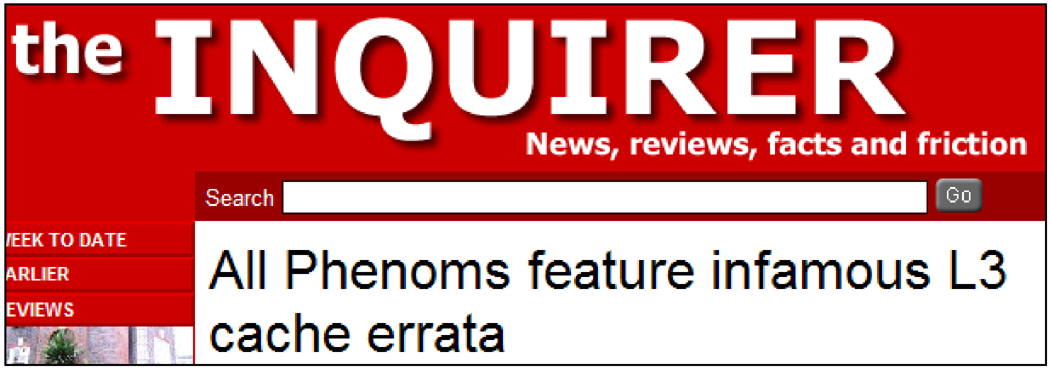
Escaped Bugs in the Memory Subsystem

- 10% of the bugs that made it to product are related to the **memory subsystem**

Intel® Core™2 Duo Processor E8000Δ and E7000Δ Series

Excerpt from Specification Update

bug AW38	**No Fix**	Instruction fetch may cause a livelock during snoops of the L1 data cache



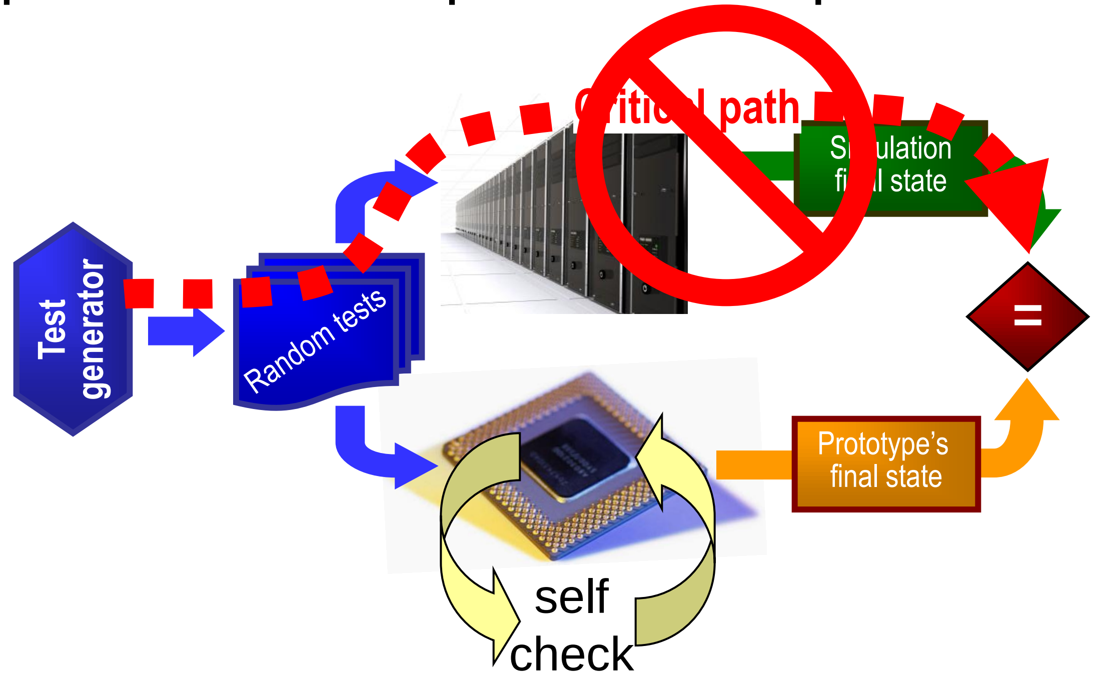
All Phenoms feature infamous L3 cache errata

[Nov. 2007]

Memory related bugs are hard to find

Post-Silicon Design Goals

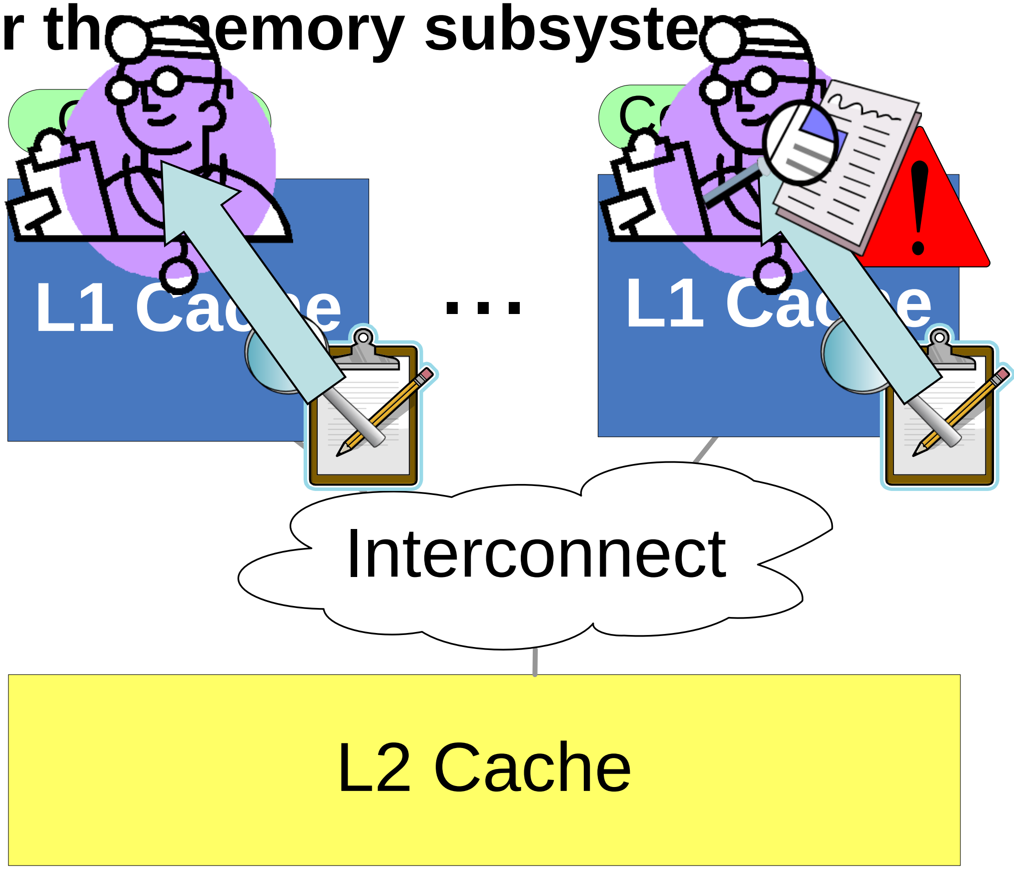
- High coverage
  - Enable self-detection of memory ordering errors
  - Coherence and consistency errors
- Low area impact
- No performance impact after shipment

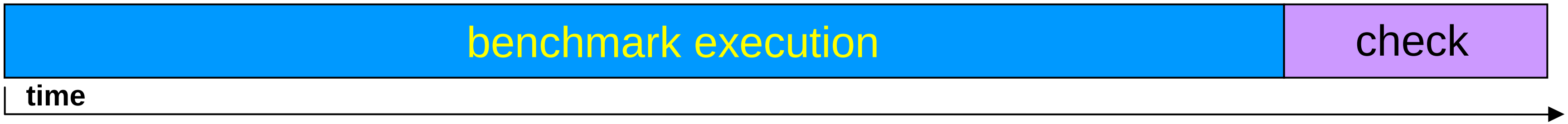


DACOTA: Data Coloring for Consistency Testing and Analysis

Post-silicon validation for the memory subsystem

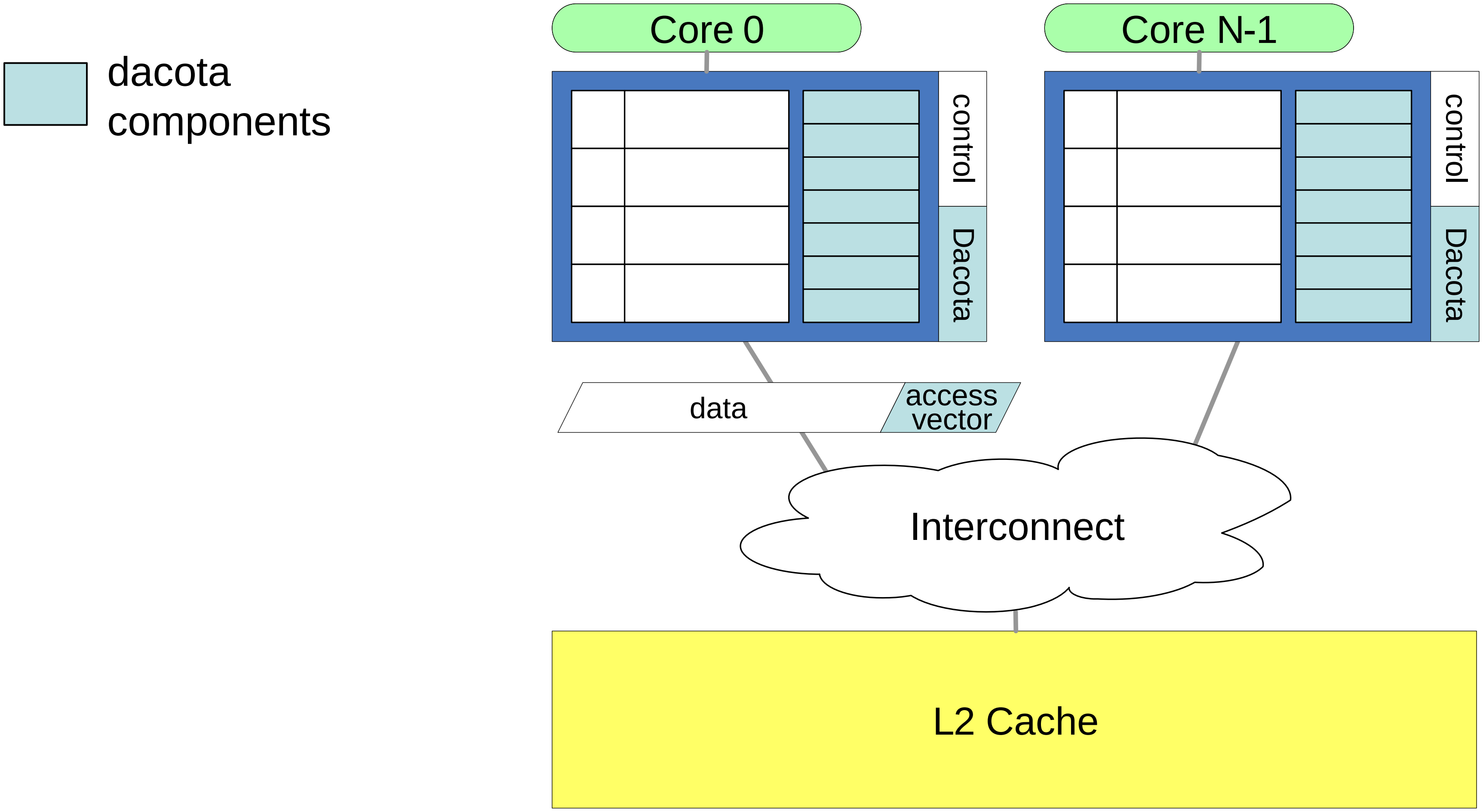
- **Logging**
  - stores ordering info
  - uses cache storage temporarily
- **Checking**
  - starts when storage fills
  - distributed algorithm on individual cores





Low Overhead Logging Architecture

- DACOTA controller augments cache controller logic
- Reconfigures a portion of cache for activity log



dacota components

Core 0

Core N-1

control

Dacota

data

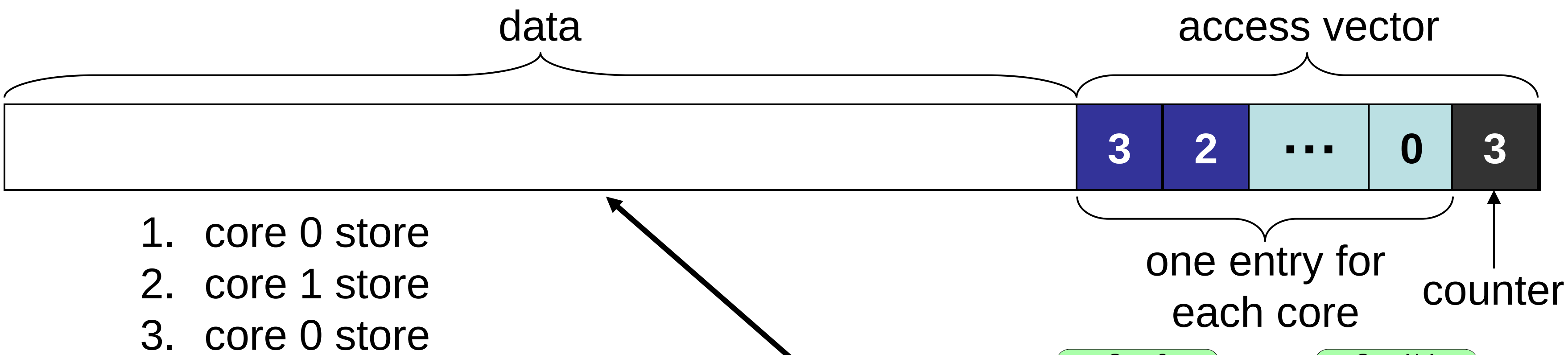
access vector

Interconnect

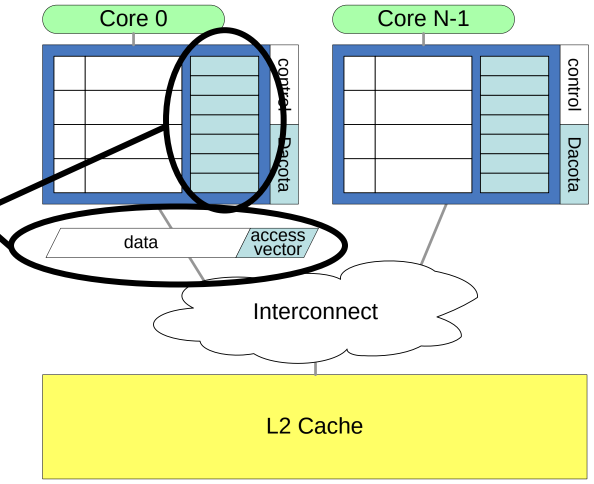
L2 Cache

Low Overhead Logging Architecture

- Attach **access vector** to each cache line
  - Tracks the order of memory accesses to one line
  - Entry for each core stores a sequence ID



1. core 0 store
2. core 1 store
3. core 0 store

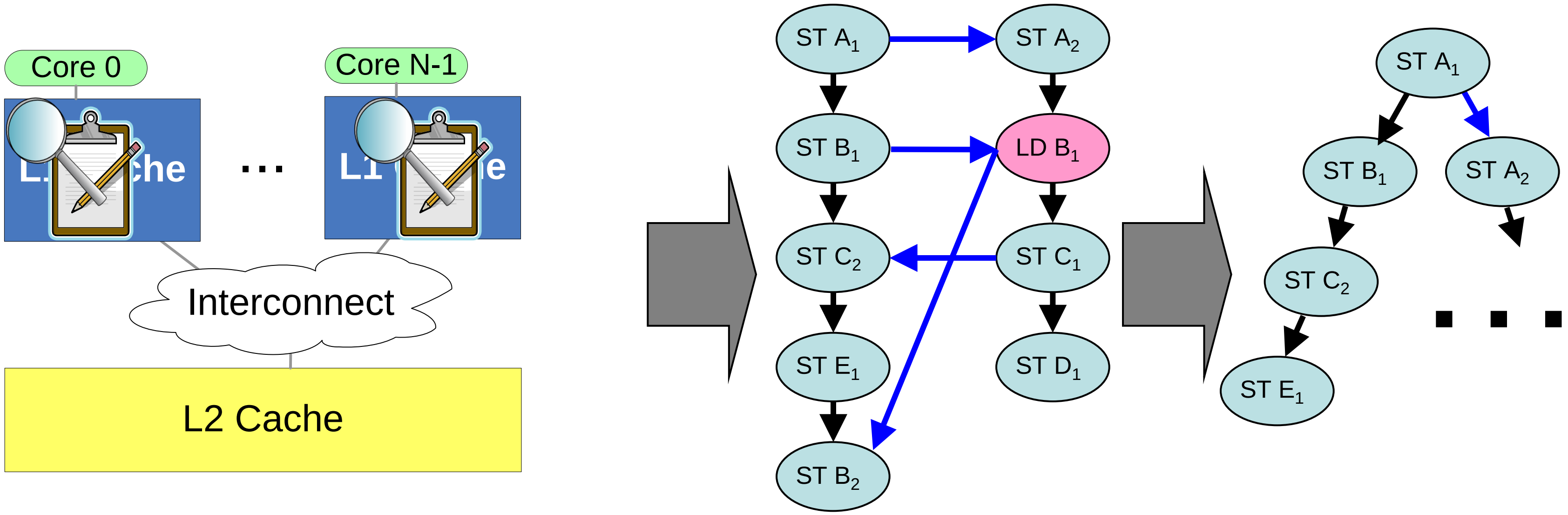


- Allocate space for **activity log**
  - Stores a sequence of access vectors in program order

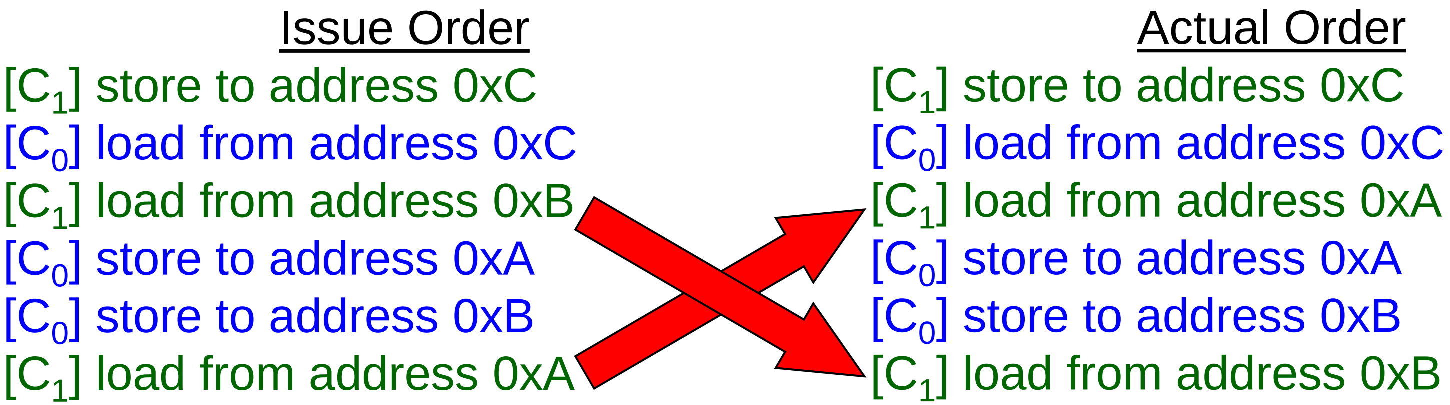
Checking Algorithm – On Site

- Compares activity logs from L1 caches
- Distributed algorithm runs on cores

1. Aggregate logs
2. Construct graph (protocol specific)
    - many protocol supported: SC, TSO, processor C., weak C.
3. Search graph for cycles, indicating ordering violation



Example – Sequential Consistency

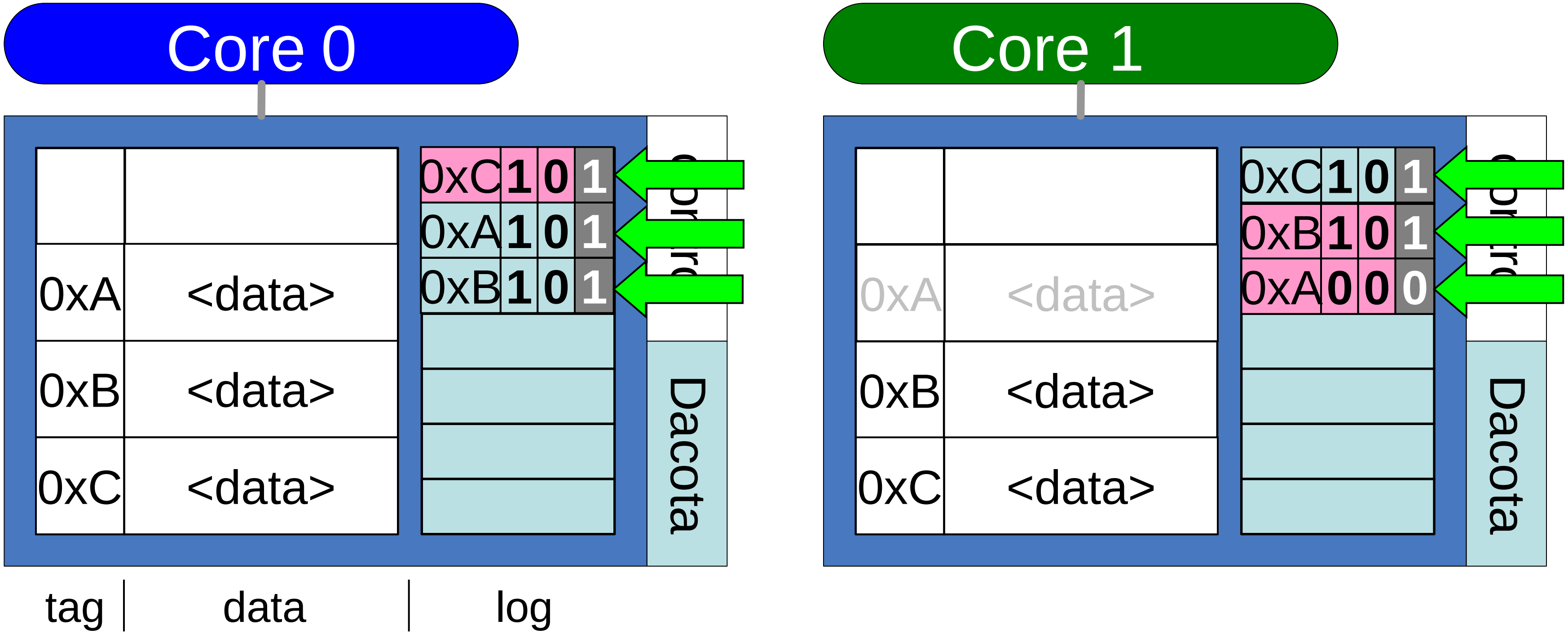


Issue Order

[C_1] store to address 0xC
[C_0] load from address 0xC
[C_1] load from address 0xB
[C_0] store to address 0xA
[C_0] store to address 0xB
[C_1] load from address 0xA

Actual Order

[C_1] store to address 0xC
[C_0] load from address 0xC
[C_1] load from address 0xA
[C_0] store to address 0xA
[C_0] store to address 0xB
[C_1] load from address 0xB



Core 0

tag	data
0xA	<data>
0xB	<data>
0xC	<data>

log			
0xC	1	0	1
0xA	1	0	1
0xB	1	0	1

Core 1

tag	data
0xA	<data>
0xB	<data>
0xC	<data>

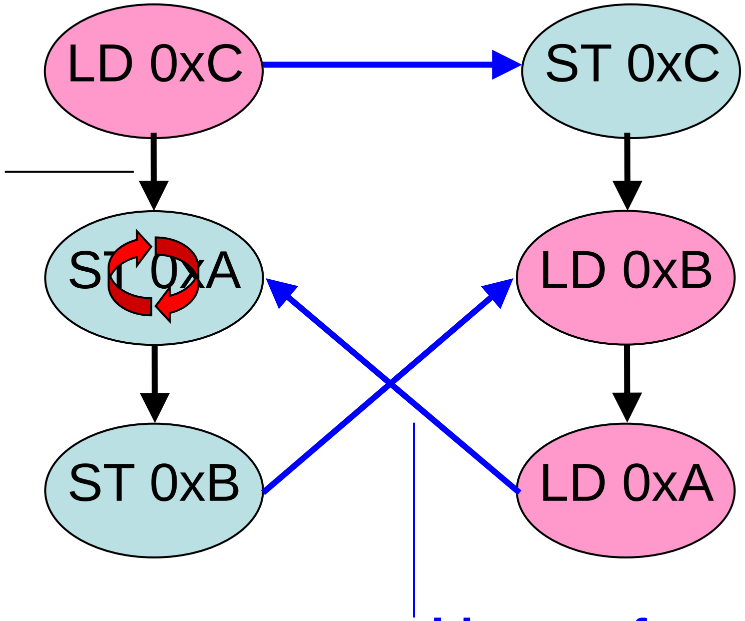
log			
0xC	1	0	1
0xB	1	0	1
0xA	0	0	0

Example – Sequential Consistency

Activity Logs

0xC	**1**	**0**	1
0xA	**1**	**0**	1
0xB	**1**	**0**	1

0xC	**1**	**0**	1
0xB	**1**	**0**	1
0xA	**0**	**0**	0



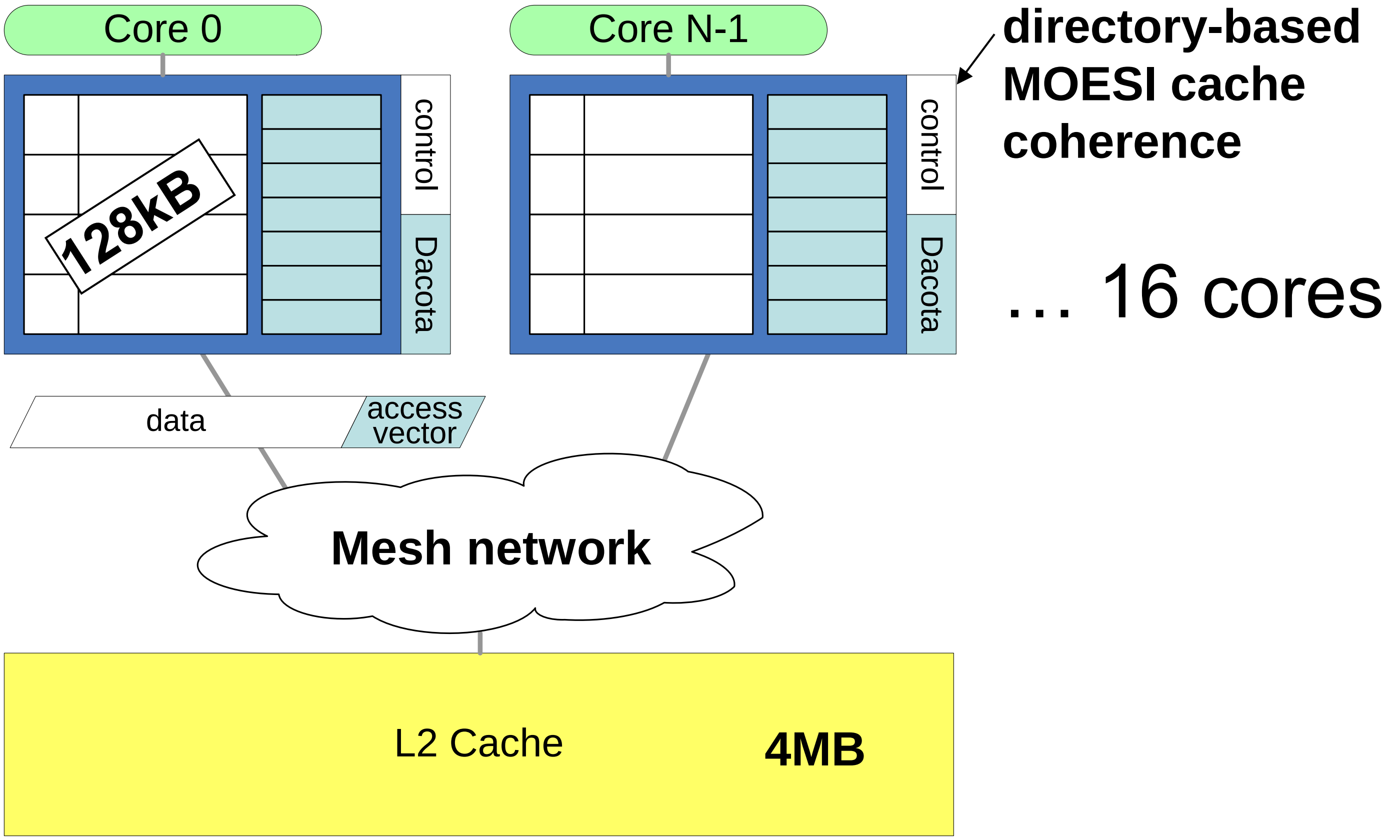
program order edges

cycle indicates violation

address reference edges

Experimental Setup

- Implemented checkers in GEMs simulator
- Created buggy versions of cache controllers
- TSO consistency model



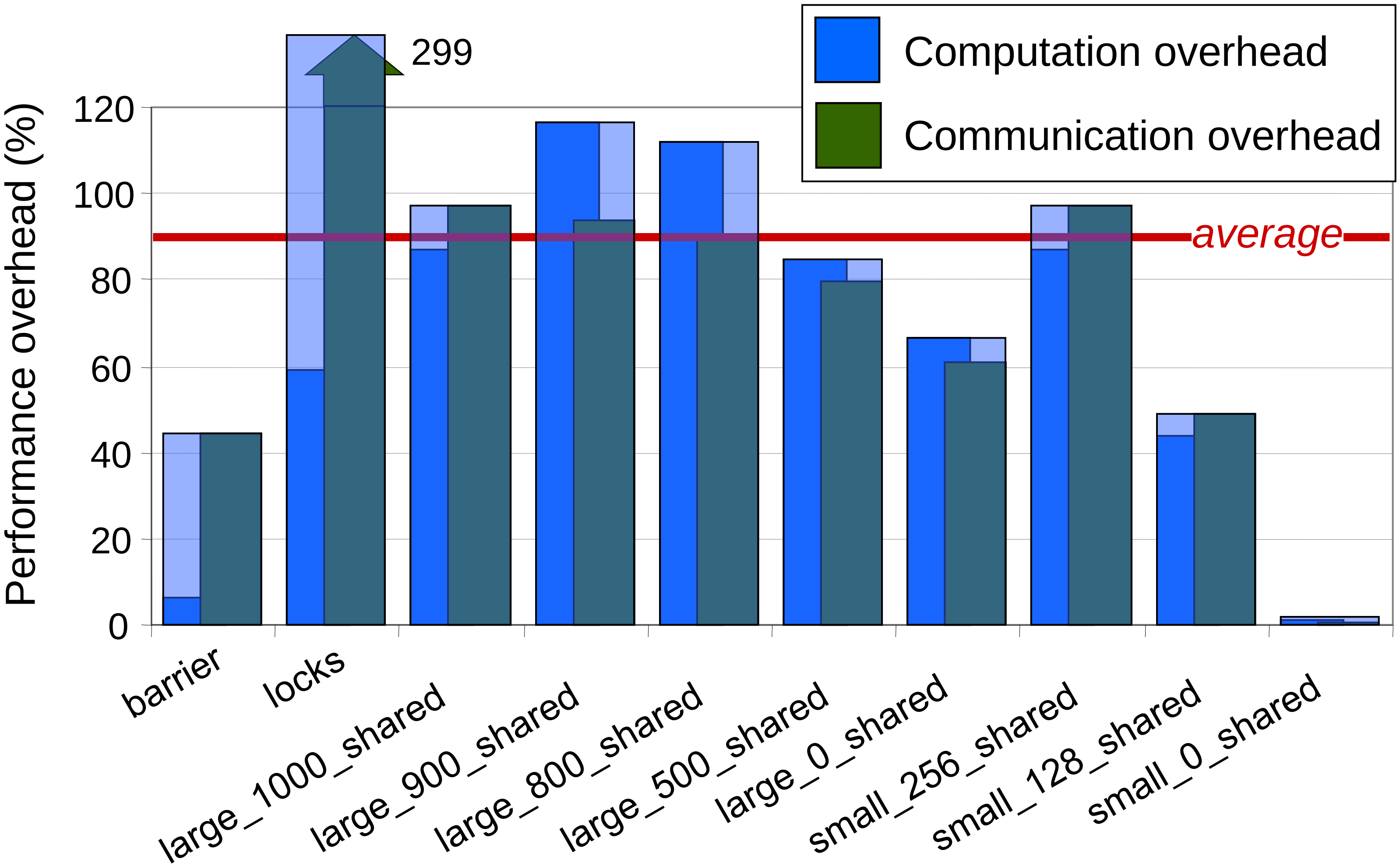
Experimental Setup

- Testbenches
  - Directed random stimulus: memory intensive
  - SPLASH2 Benchmarks

		Cycles to Expose Bug
shared-store	store to a shared line may not invalidate other caches	0.3M
invisible-store	store message may not reach all cores	1.3M
store-alloc1	store allocation in any core may not occur properly	1.9M
store-alloc2	store allocation in one core may not occur properly	2.3M
reorder1	invalid store reordering (all cores)	1.4M
reorder2	invalid store reordering (one core)	2.8M
reorder3	invalid store reordering (single address, all cores)	2.9M
reorder4	invalid store reordering (single address, one core)	5.6M

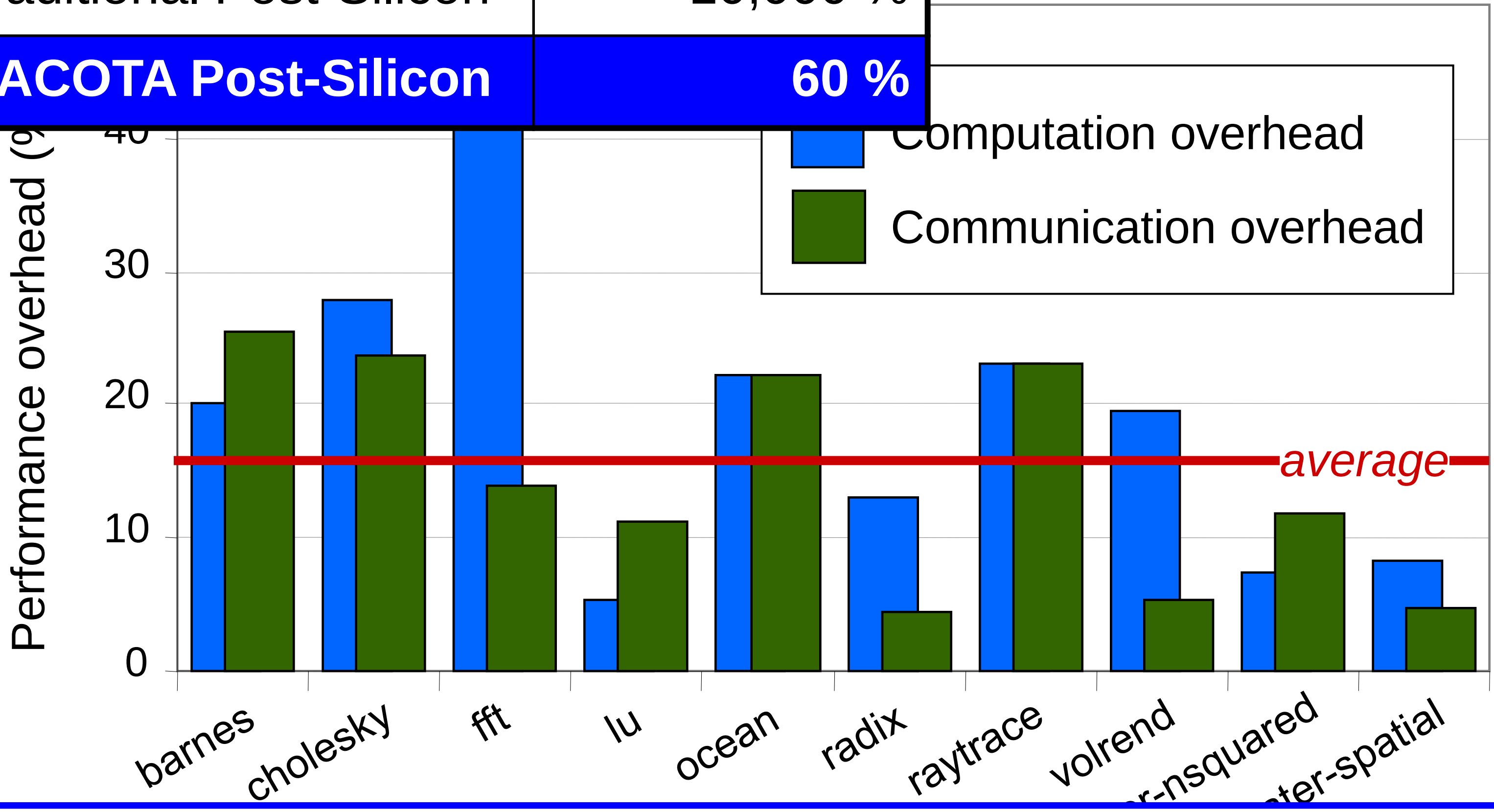
- Bugs inspired by bugs found in processor errata
- Injected one at a time

Performance Impact - Random



Performance Impact – SPLASH2

Pre-Silicon	100,000,000 %
Traditional Post-Silicon	10,000 %
DACOTA Post-Silicon	**60 %**



100x more tests!

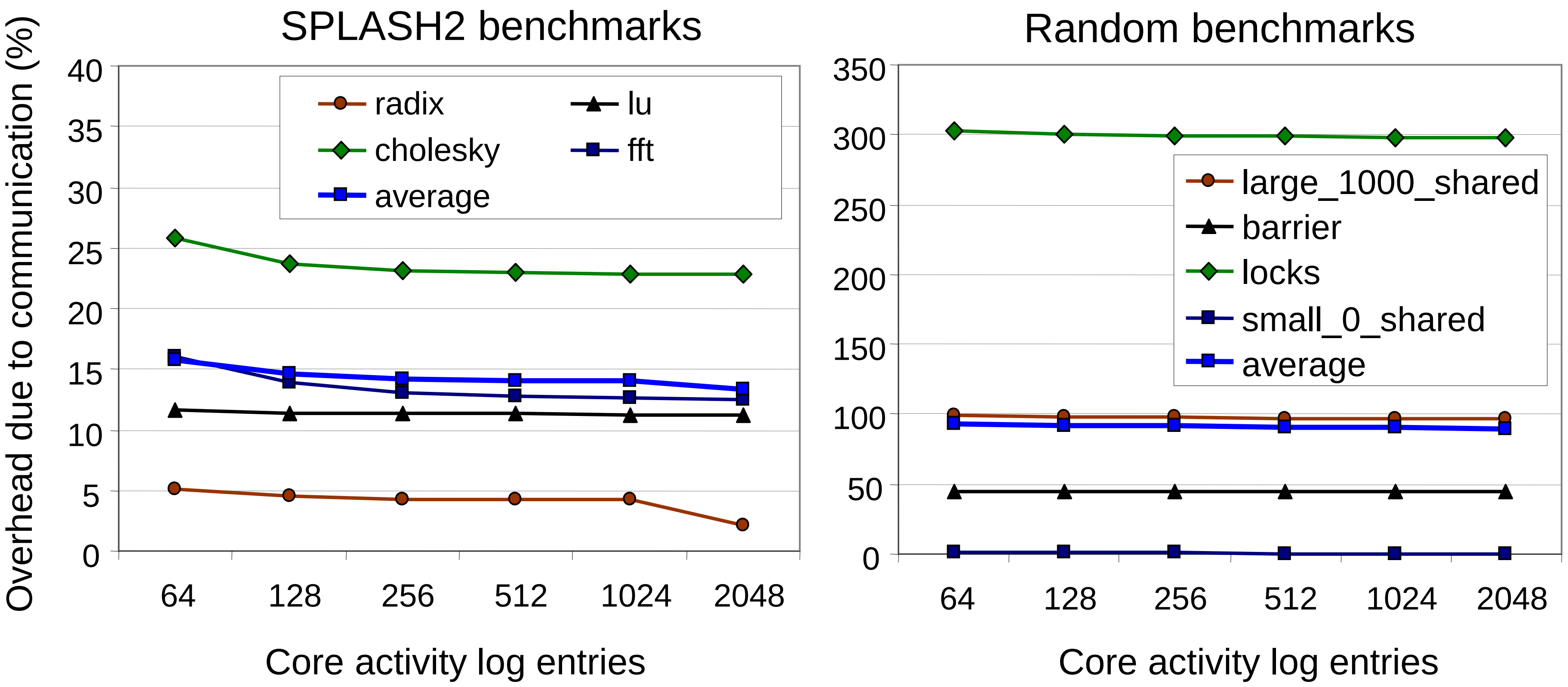
Area Impact

Pre-Silicon	100,000,000 %
Traditional Post-Silicon	10,000 %
DACOTA Post-Silicon	**60 %**
Runtime	0 %

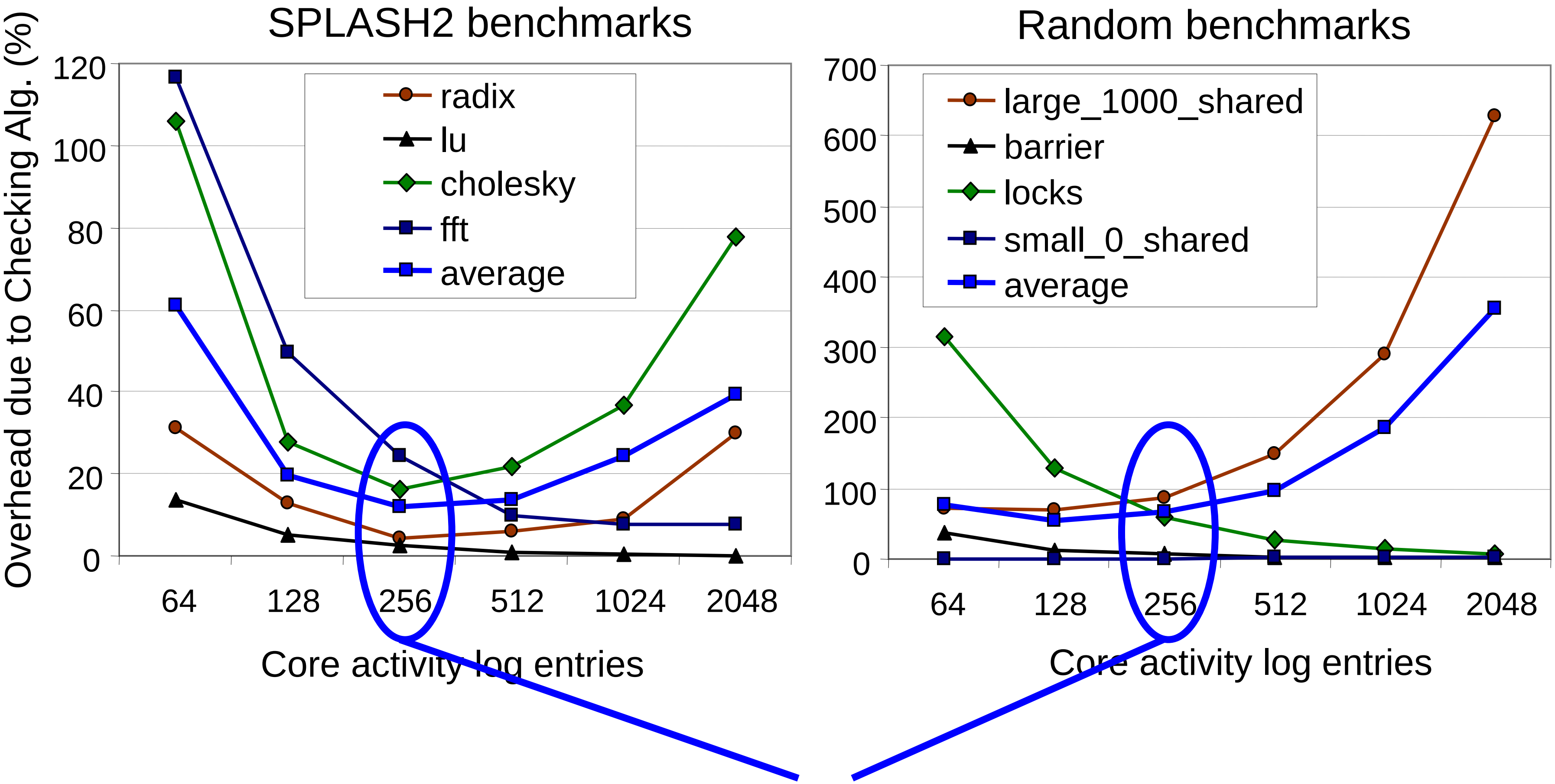
Area Overhead - Storage	
DACOTA	**544 B**
Chen, *et al., 2008*	617,472 B
Meixner, *et al., 2006*	940,032 B

- Implemented DACOTA in Verilog
- **0.01%** overhead in OpenSPARC T1

Communication Overhead



Checking Algorithm Overhead



ideal trade-off

Related Work

Pre-Silicon

Dill, *et al*., 1992;

Abts, *et al*., 1993;

Pong, *et al*., 1997;

German, *et al*., 2003

- Formal verification possible for abstract protocol
- Insufficient for implementation

Post-Silicon

Josephson, *et al.,* 2006

Paniccia, *et al.,* 1998

Whetsel, *et al.,* 1991

Tsang, *et al.,* 2000

- Post-Si testing

DeOrio, *et al.,* 2008

- Post-Si verification
- Verifies coherence, but not consistency

Runtime

Meixner, *et al.,* 2006;

Chen, *et al.,* 2008

- Effective for protection against transient faults
- Problematic for functional errors
- High area overhead

Conclusions

- DACOTA is an on-chip post-silicon debugging solution for detecting errors in memory ordering
  - Enables **self-detection** of memory ordering errors
- Effective at **catching bugs**
  - **100x more coverage** than traditional post-silicon
- Very low area overhead
  - **0.01% area overhead** on OpenSPARC T1
- **No performance impact** to end user
  - Disable on shipment