Using a Formal Model to Improve Verification of a Cache-Coherent System-on-Chip

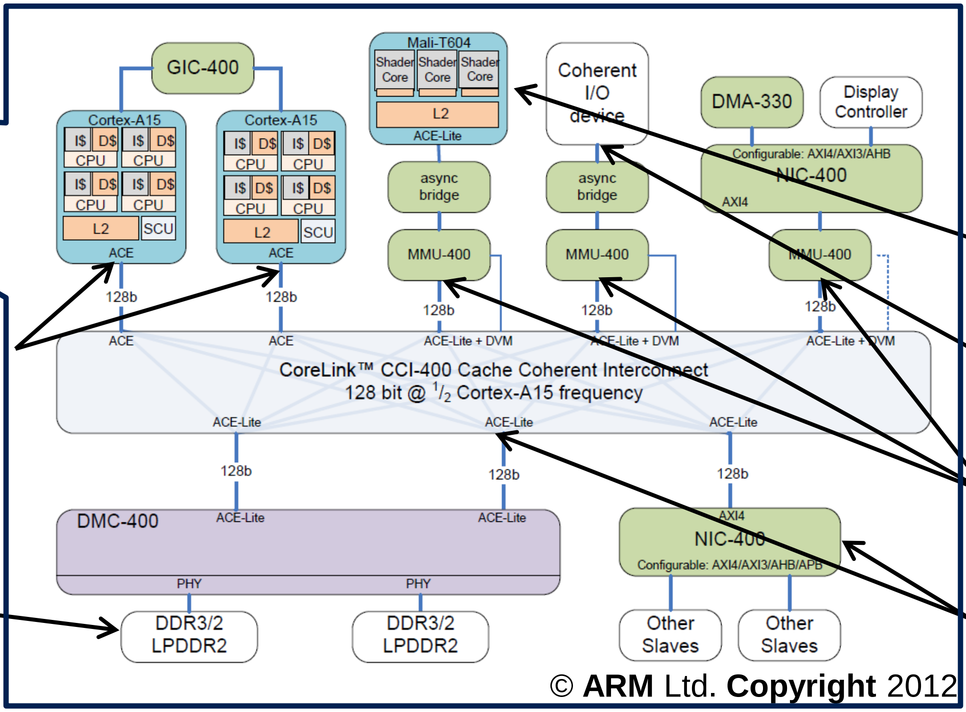
Abderahman KRIOUILE [1,2]

Wendelin SERWE [2]

[1] STMicroelectronics
[2] Inria Univ. Grenoble Alpes - LIG

TACAS'2015
London, UK, Apr 13th-17th 2015

Heterogeneous System-on-Chip (SoC)



Processors (with caches)

Memory

Components without caches:

Graphical co-processor

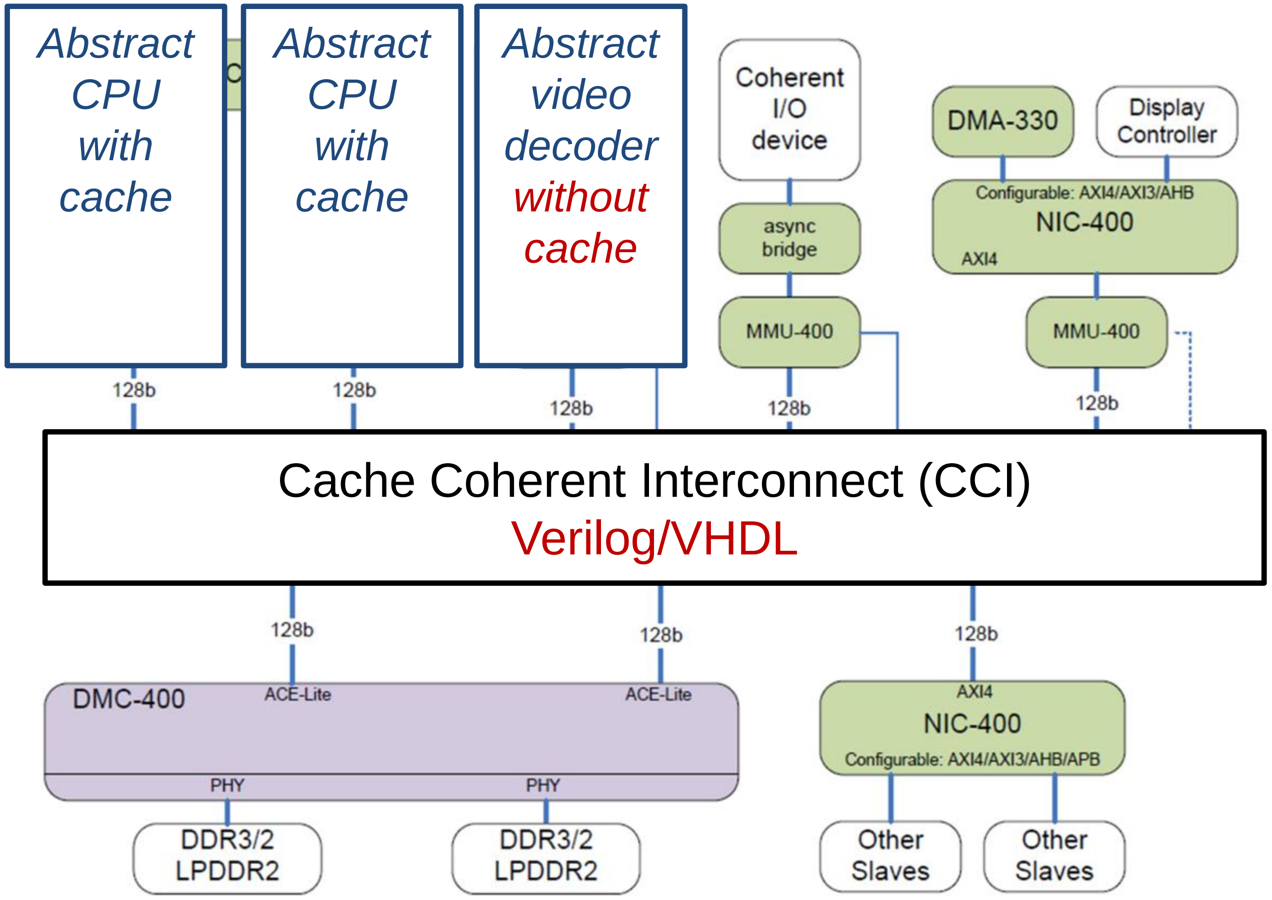
I/O Blocks

Specialized Blocks

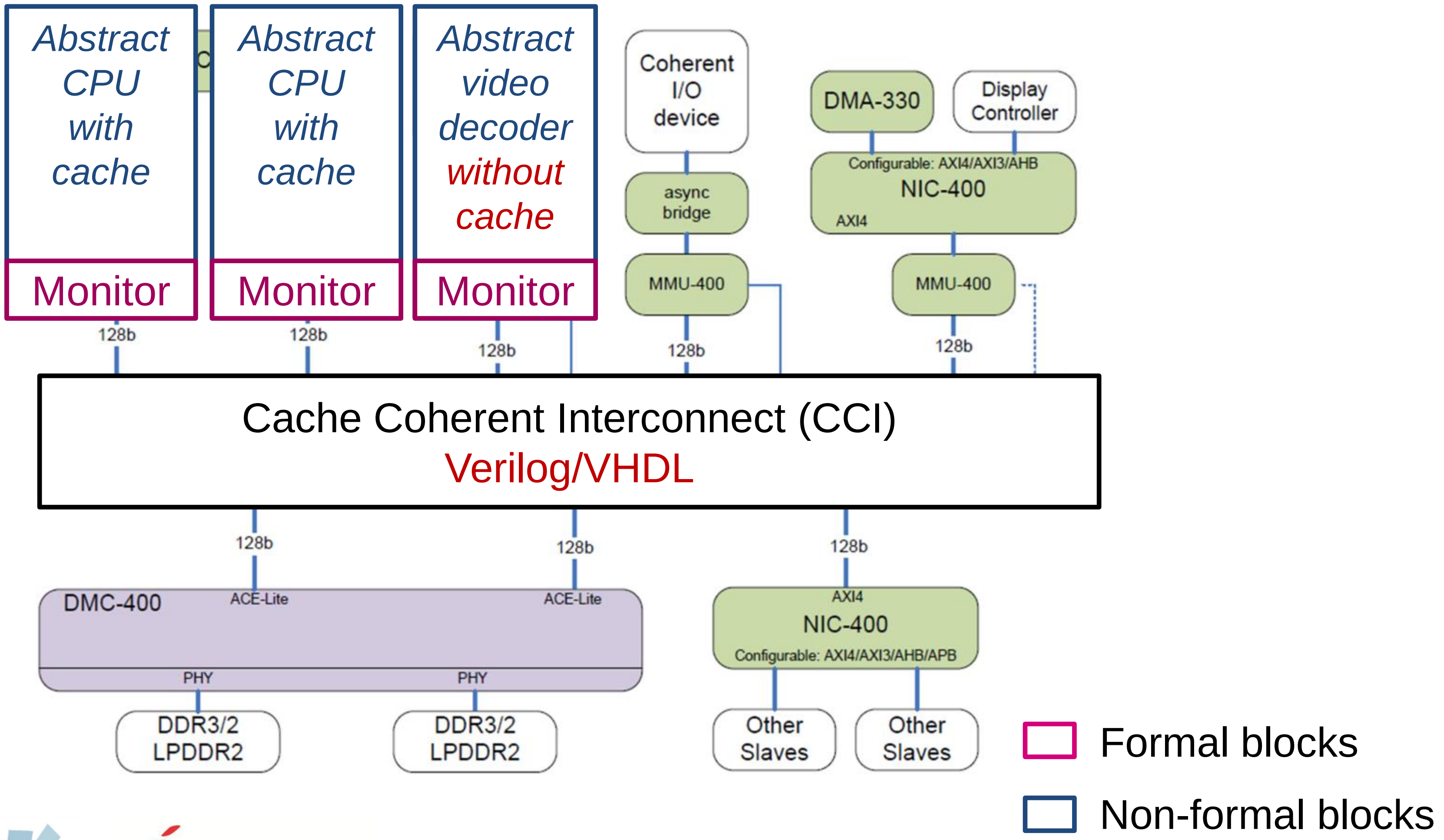
Interconnects

- Need for **System-Level Cache Coherency**
- **ARM** proposed ACE specification: standard for system level cache coherency

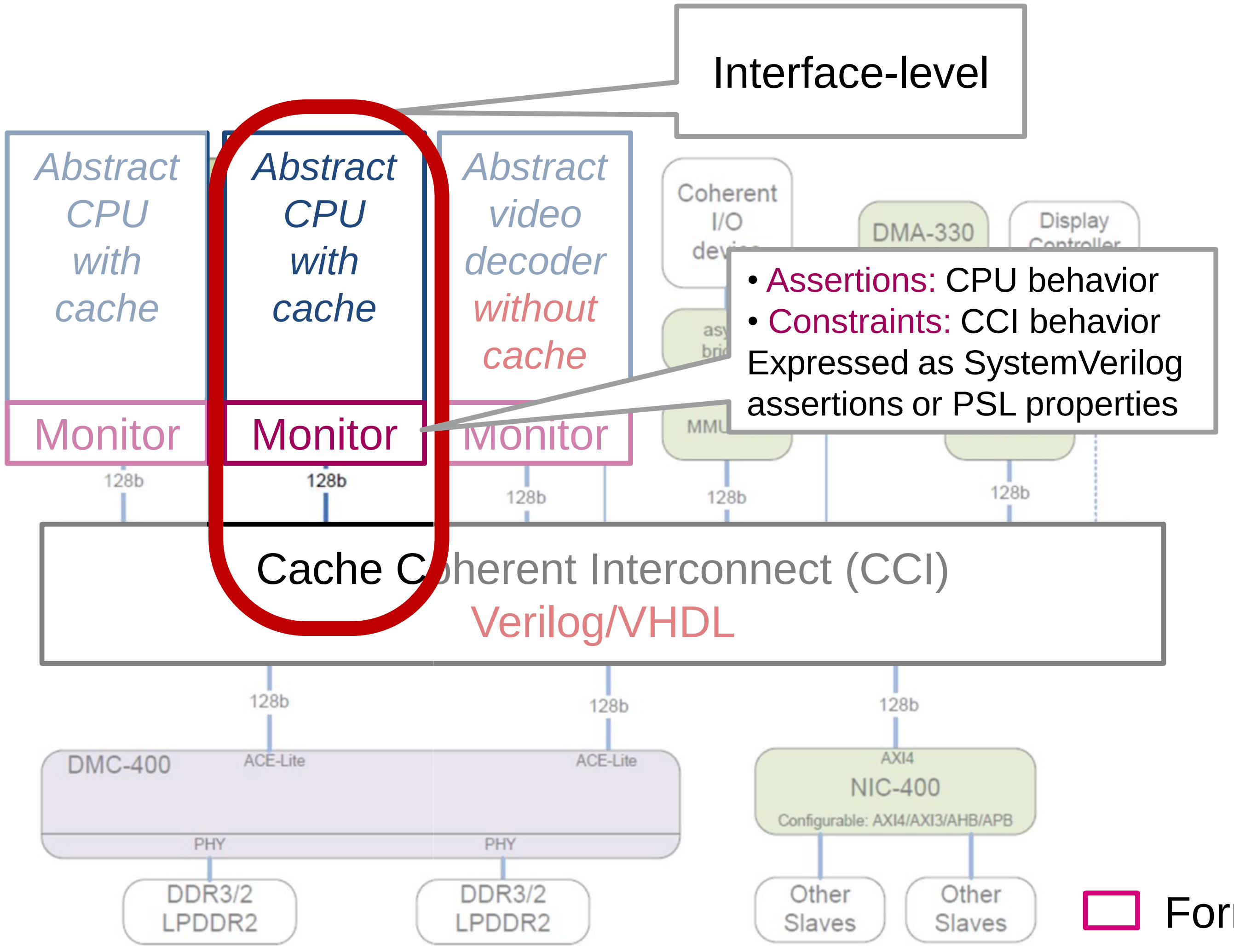
Simulation-Based Testing



Simulation-Based Testing



Simulation-Based Testing



- Assertions: CPU behavior
- Constraints: CCI behavior

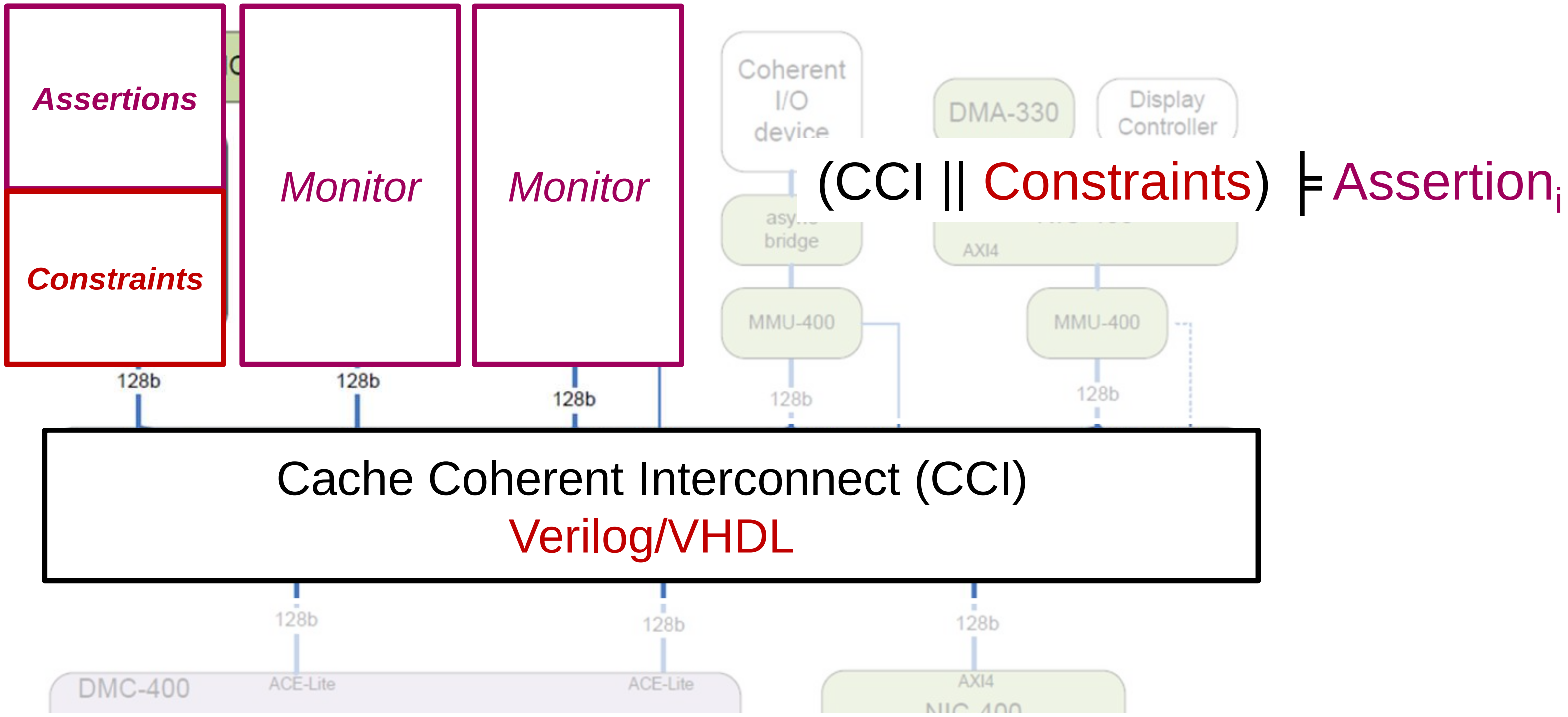
Expressed as SystemVerilog assertions or PSL properties

🟥 Formal blocks

🟦 Non-formal blocks

Model Checking

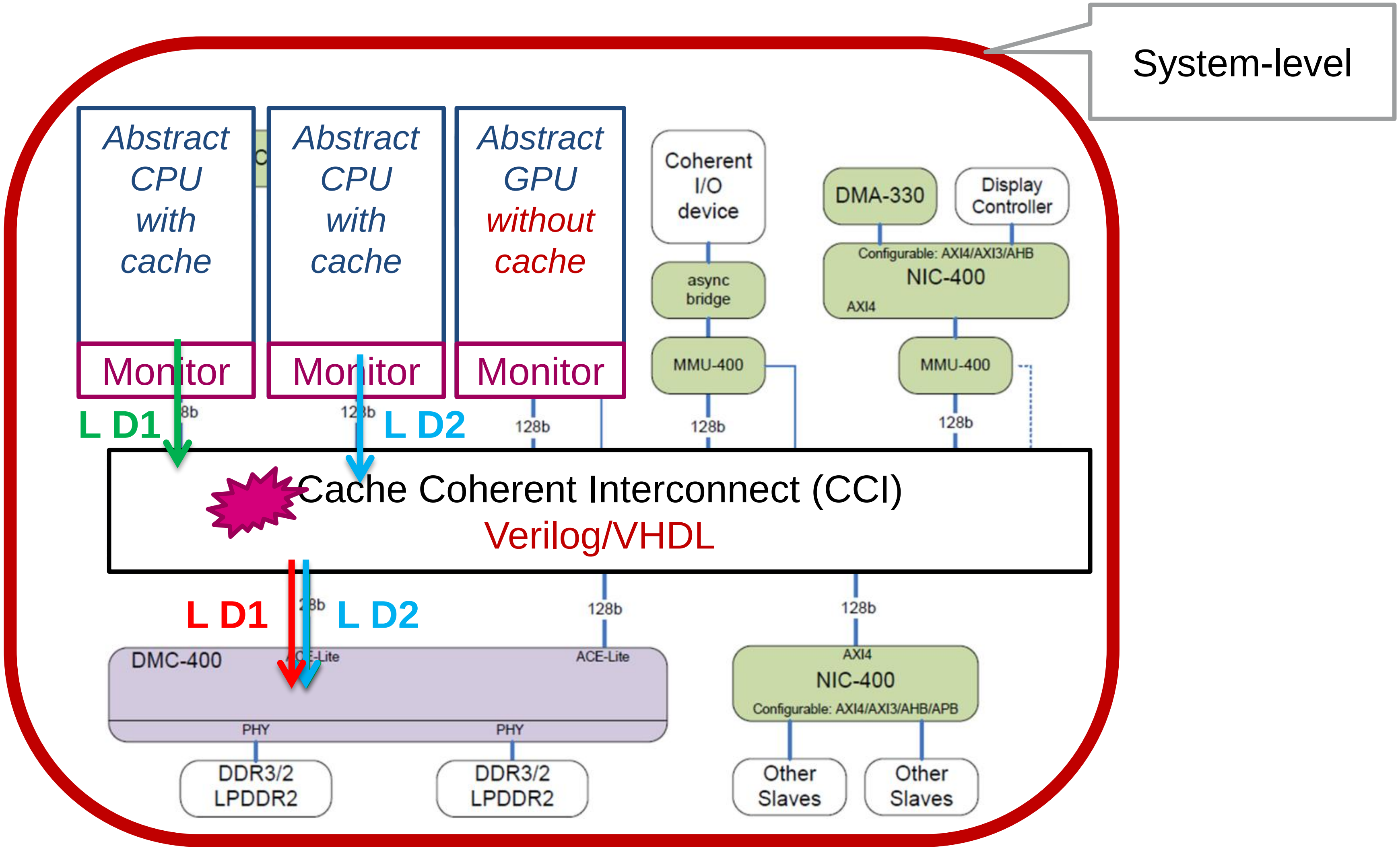
No ran test



$(CCI \parallel Constraints) \models Assertion_i$

- Applying restrictions for more exploration
- Limitation due to state-space explosion problem

Need for System-Level Verification



Functional Verification Current Limitations

- Achievements
  - Interface-level verification
  - Formal monitoring behavior of hardware interfaces
- Issues
  - Black-box commercial interface-level monitors
  - Distributed Systems on-chip: System-level verification
    - Complete formal verification: intractable!
    - System-level testing and monitoring
    - Scenarios to test system-level properties

Contributions to Improve Validation

- Goal: use a parametric system-level formal model to improve validation

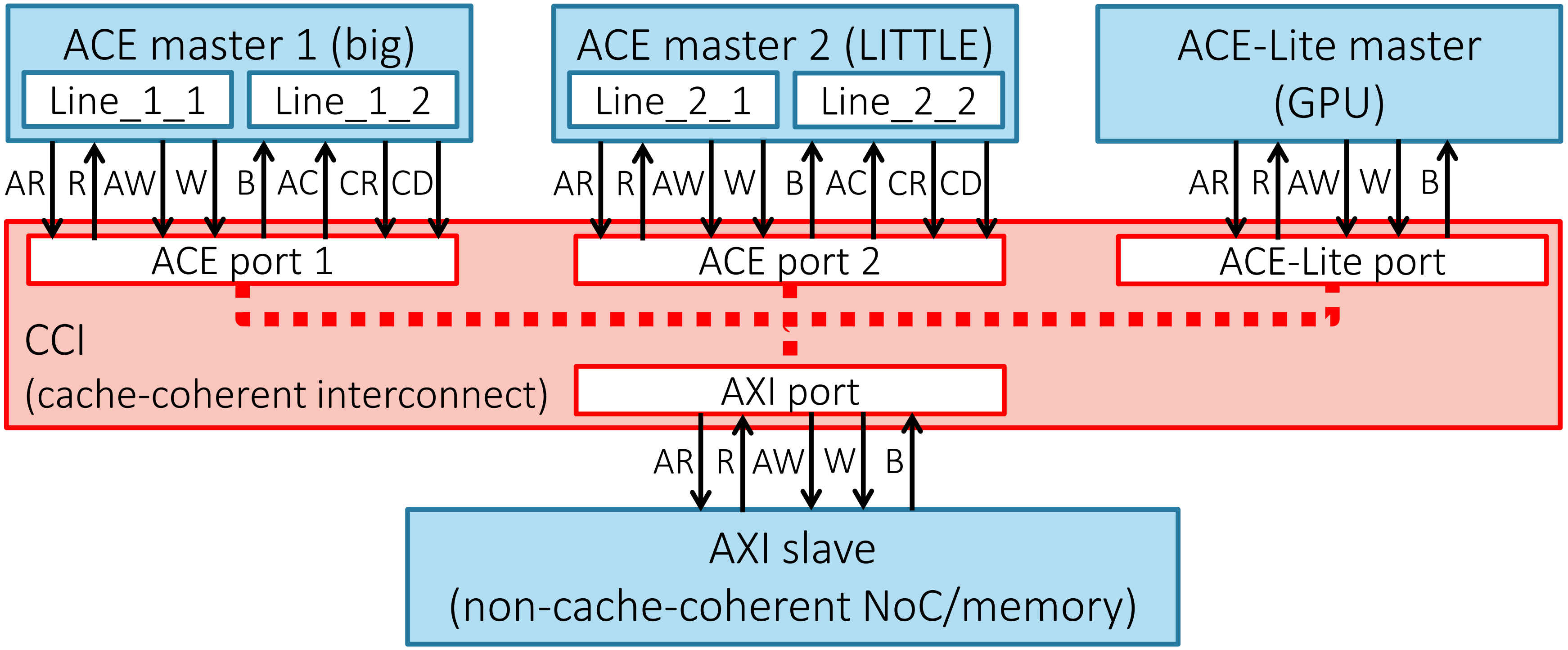
- Approach
  - **Formal Model of an ACE-Based System-on-Chip**
  - **Sanity of Interface-Level Properties**
  - **From System-Level Properties to Clever Test Cases**
    - **System-level formal model**
    - **Unconstrained the formal model to allow possible faults**
    - **Counterexamples to get interesting configurations**
    - **Refine scenarios using existing commercial verification**

CADP verification toolbox

- Modular toolbox for formal modeling and enumerative verification of asynchronous systems (about 50 tools)
- Based on concurrency theory (process calculi)
- Semantic model: Labeled Transition System (LTS)
- User-friendly input language (LNT) integrating features of
  - Imperative programming: loops, variables, …
  - Concurrency theory: parallel composition, formal semantics
- Several verification paradigms and techniques: model and equivalence checking, compositional, on-the-fly, …
- Model-based testing
- Free of charge for academic use

More information: http://cadp.inria.fr

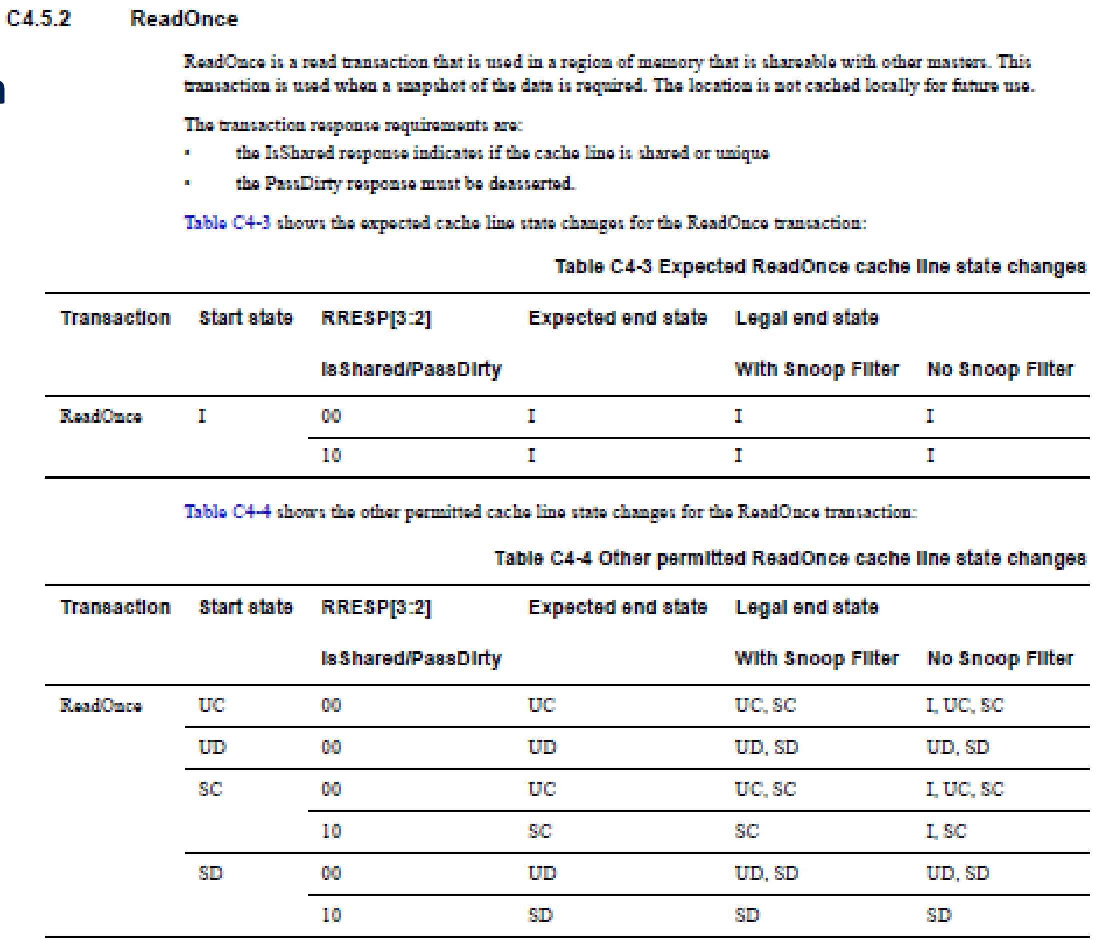
Formal model of an ACE-based SoC



- Interface transfers modeled by rendezvous
- 3400 lines of LNT code derived from ACE specification
- Parametric: #masters, forbidden ACE transactions, …
- [Kriouile-Serwe-13] Formal Analysis of the ACE Specification for Cache Coherent Systems-on-Chip, FMICS, LNCS 8187, 2013

From ACE specification to LNT model

```
select
  only if Line.state == ACE_UC then
    R (ReadOnce, …)
    -- Line.state does not change
  end if
[]
  only if Line.state == ACE_SC then
    select
      R (ReadOnce, …);
      Line.state := ACE_UC
    []
      R (ReadOnce, …);
      Line.state := ACE_SC
    end select
  end if
[]
  …
end select
```



C4.5.2 ReadOnce

ReadOnce is a read transaction that is used in a region of memory that is shareable with other masters. This transaction is used when a snapshot of the data is required. The location is not cached locally for future use.

The transaction response requirements are:

- the IsShared response indicates if the cache line is shared or unique
- the PassDirty response must be deasserted.

Table C4-3 shows the expected cache line state changes for the ReadOnce transaction:

Table C4-3 Expected ReadOnce cache line state changes

Transaction	Start state	RRESP[3:2]	Expected end state	Legal end state	
		IsShared/PassDirty		With Snoop Filter	No Snoop Filter
ReadOnce	I	00	I	I	I
		10	I	I	I

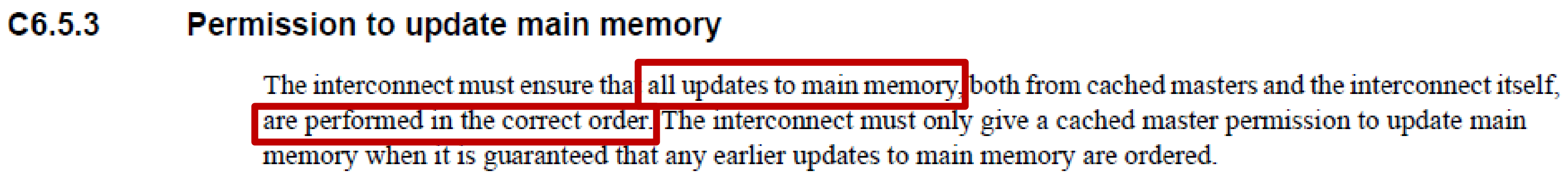
Table C4-4 shows the other permitted cache line state changes for the ReadOnce transaction:

Table C4-4 Other permitted ReadOnce cache line state changes

Transaction	Start state	RRESP[3:2]	Expected end state	Legal end state	
		IsShared/PassDirty		With Snoop Filter	No Snoop Filter
ReadOnce	UC	00	UC	UC, SC	I, UC, SC
	UD	00	UD	UD, SD	UD, SD
	SC	00	UC	UC, SC	I, UC, SC
		10	SC	SC	I, SC
	SD	00	UD	UD, SD	UD, SD
		10	SD	SD	SD

Modeling Global Requirements

- How to model

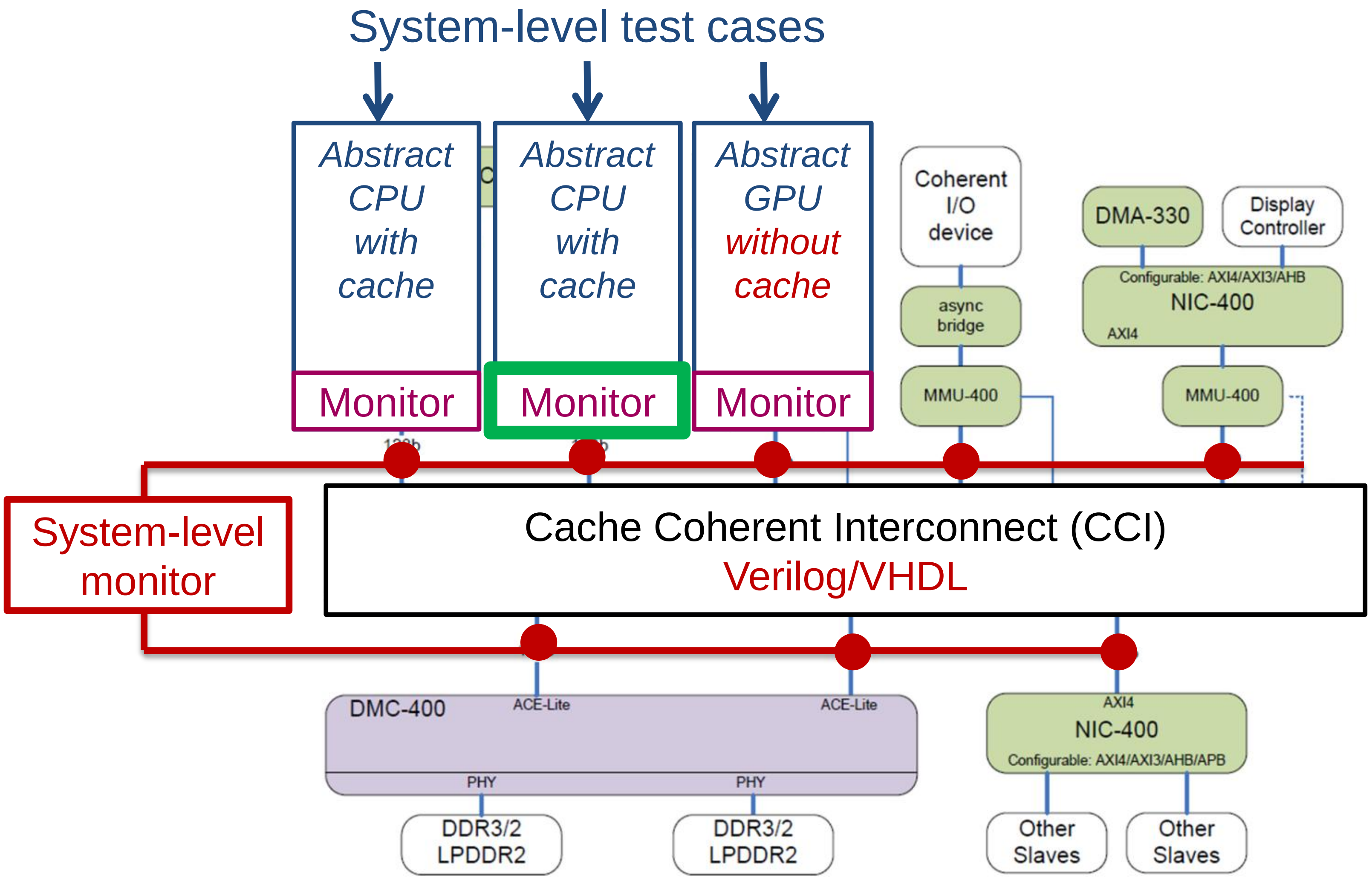


C6.5.3 Permission to update main memory

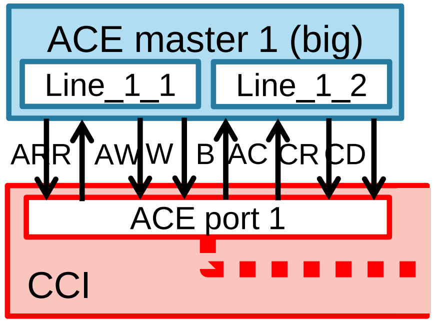
The interconnect must ensure that all updates to main memory, both from cached masters and the interconnect itself, are performed in the correct order. The interconnect must only give a cached master permission to update main memory when it is guaranteed that any earlier updates to main memory are ordered.

- Is this the definition of data integrity?
- Constraint-oriented specification style:
  Observer processes constrain the behavior
- Generate the LTS with or without global constraints

Improving Simulation-Based Testing



Sanity of Interface-Level Properties



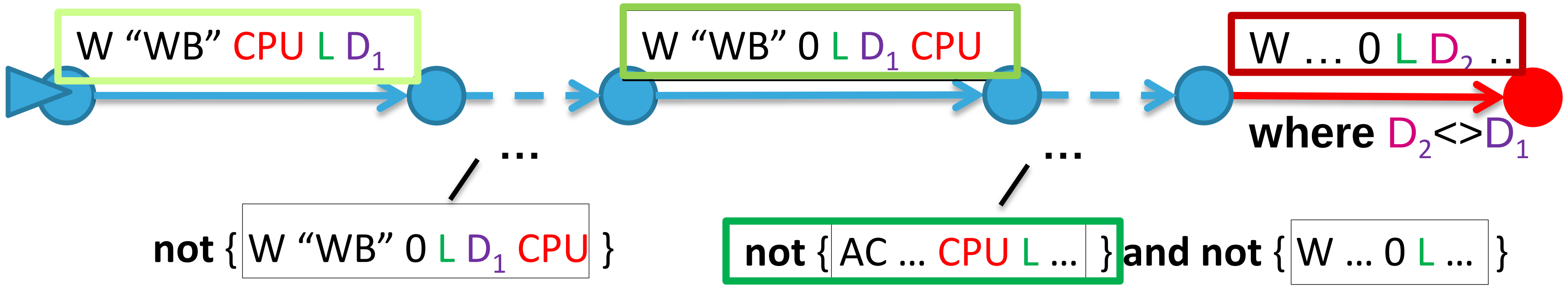
- Project the model on the interface:
  - Hide unrelated transitions
  - Minimize (for divergence-sensitive branching bisimulation)
- Represent each formal property as an LNT process
- Two facets of sanity:
  - Local: each property includes the (sub-)interface
  - Global: composition of all properties includes the interface
- Discovery of a missing property in the list:
  same number of AW and W transactions

Data Integrity: A System-Level Property Example

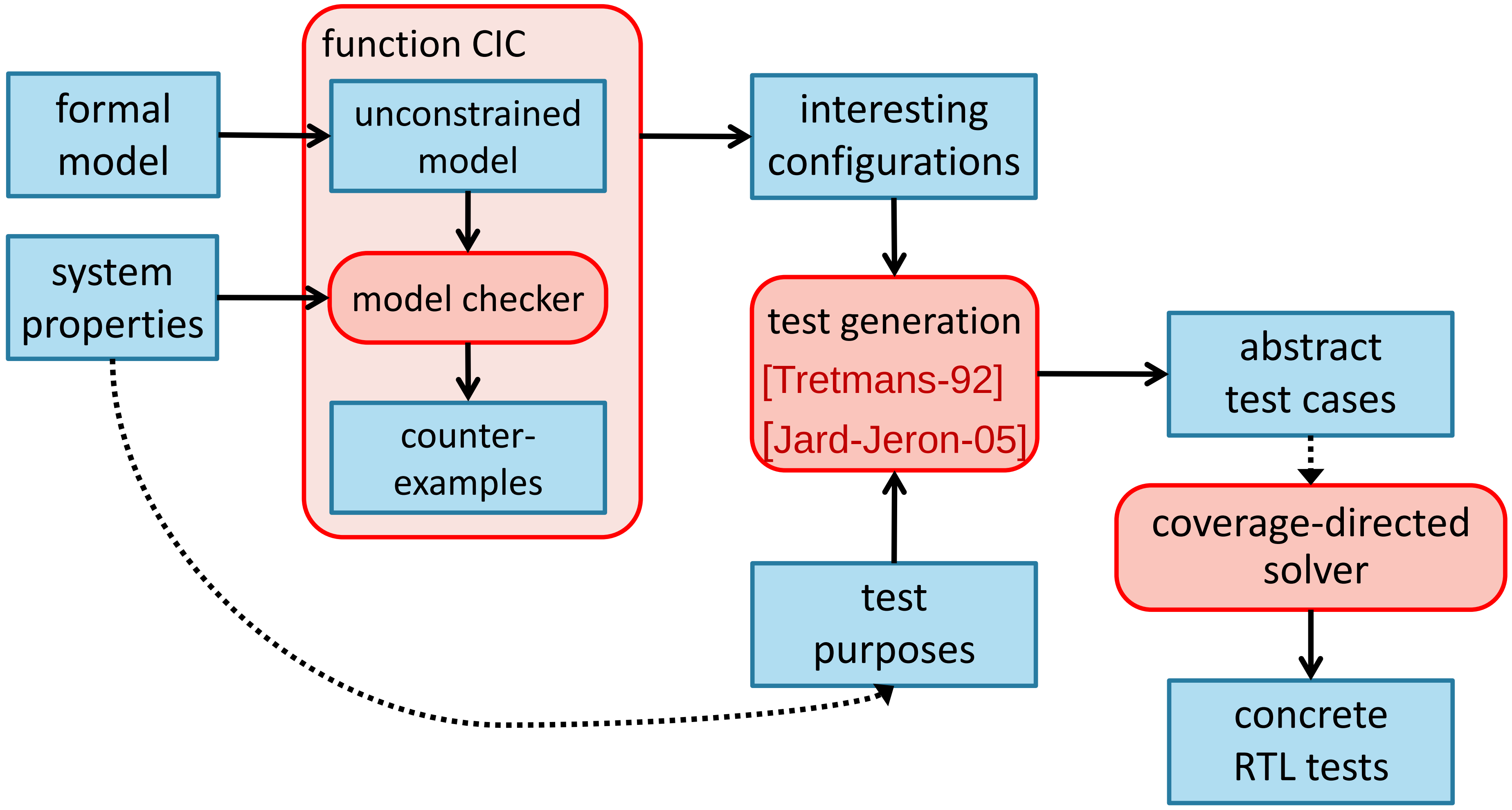
correct order of write operations ("WRITEBACK") to the shared memory (with index "0")

MCL (Model Checking Language) [Mateescu-Thivolle-08]

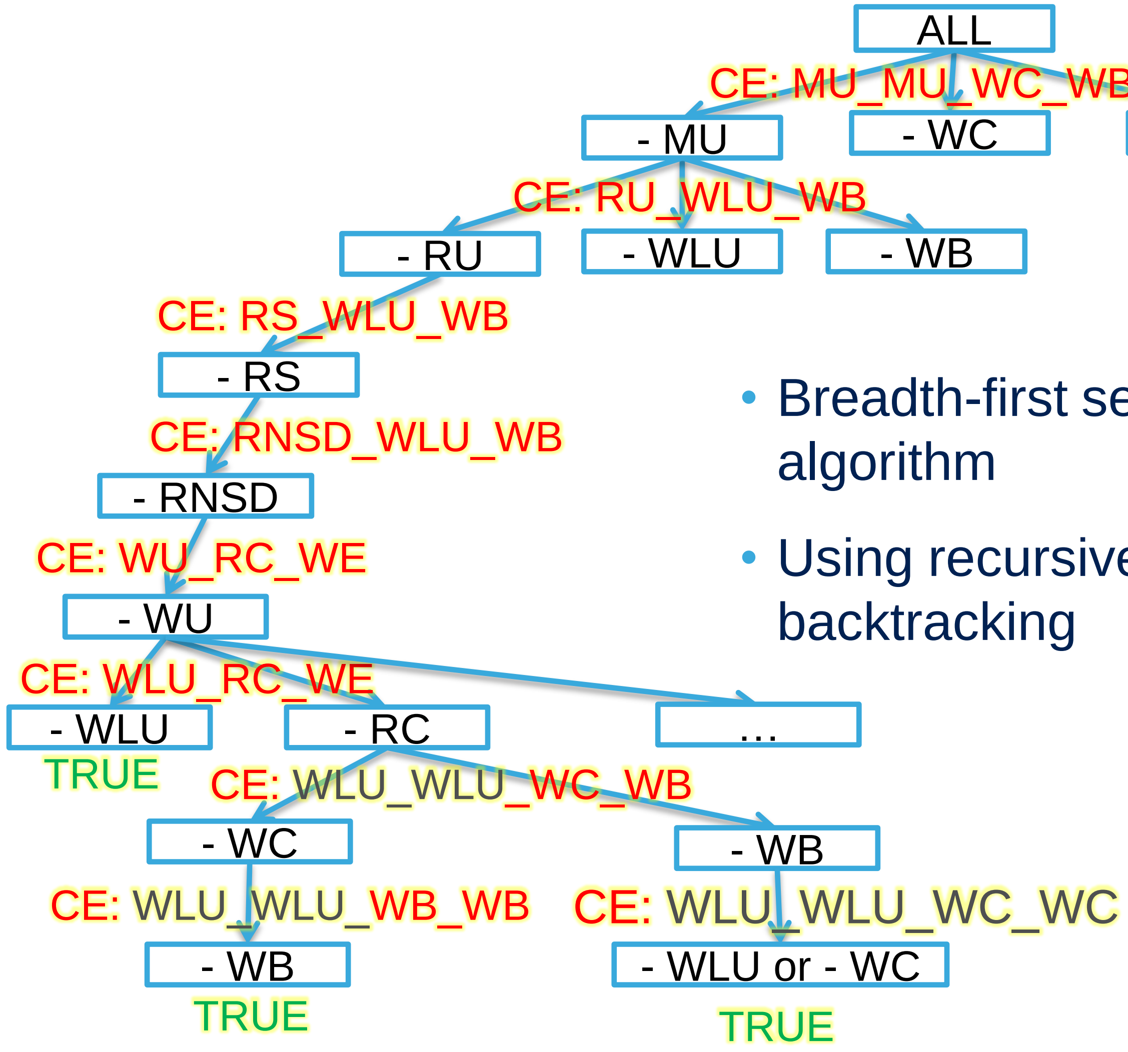
```
[ true * .
  { W !"WRITEBACK" ?CPU:Nat ?L:Nat ?D1:Nat } .
  ( not { W !"WRITEBACK" !"0" !L !D1 !CPU } ) * .
  { W !"WRITEBACK" !"0" !L !D1 !CPU } .
  (
     ( not { AC ... !CPU ?any of Nat !L ... } ) and
     ( not { W ?any of String !"0" !L ... } )
  ) * .
  { W ?any of String !"0" !L ?D2:Nat ... where D2<>D1 }
] false
```


W "WB" CPU L D_1
W "WB" 0 L D_1 CPU
W … 0 L D_2 ..
where D_2<>D_1
...
not { W "WB" 0 L D_1 CPU }
not { AC ... CPU L ... } and not { W ... 0 L ... }

Generation of System-Level Test Cases



Computation of Interesting Configurations (CIC)



- Breadth-first search algorithm
- Using recursive backtracking

RC	ReadClean
RNSD	ReadNotSharedDirty
RS	ReadShared
CS	CleanShared
CI	CleanInvalid
MI	MakeInvalid
RO	ReadOnce
WU	Writeunique
WLU	WriteLineUnique
RU	ReadUnique
CU	CleanUnique
MU	MakeUnique
WB	WriteBack
WC	WriteClean
WE	WriteEvict

Several Kinds of Derived Tests

- 39 + 3 generated CTGs (Complete Test Graphs)

prop.	masters	global CTG		extr.	nb. of	largest CTG		smallest CTG		extr.
		states	trans.	time	CTGs	states	trans.	states	trans.	time
φ_3	2ACE	6,402	14,323	$>\frac{1}{2}$ y	18	903	1,957	274	543	$\simeq$7h
φ_5	2ACE	23,032	48,543	$>\frac{1}{2}$ y	14	462	888	59	107	$<$1h
	1ACE/1Lite	2,815	7,071	$>\frac{1}{2}$ y	7	193	394	59	107	$<$1h

- 296 simple system-level tests
  - for each correct initial state with two masters possibly sharing a memory line, initiate all permitted transitions
  - check correct behavior of the Cache Coherent Interconnect (e.g., generation of corresponding snoops)
- 10 sequence tests to recreate counter-examples
  - concurrency between transactions
  - conditioned by response of the Cache Coherent Interconnect

Industrial Results

- Increase of test coverage
  - 100% coverage for system-level monitor
- Discovery of ten bugs in Abstract CPU blocks and Monitors
- Major Result: Discovery of a limitation of the Cache Coherent Interconnect
  - Triggered 16 times by the derived tests, but never by the standard tests used by STMicroelectronics
  - Discovered by the architects more than a year later
  - Property to check whether tests encounter limitation

Concluding Remarks

- Formal model useful to
  - Gain better understanding (via interactive simulation)
  - Check sanity of interface-level properties
  - Derive tests for corner cases of system-level properties
- Contribute to the maturation of the test bench
- Relies on industrial tools to refine abstract tests
- Requirements for application of the approach:
  - Configurable formal model
  - Properties generating counterexamples