

Tutorial on Test Model-checking

Ganesh Gopalakrishnan

Ratan Nalumasu

Rajnish Ghughal

Mike Jones

Ritwik Bhattacharya

Ali Sezgin

Prosenjit Chatterjee

October 31, 2000

School of Computing, University of Utah

Overview of talk

- **Advantages of the approach**
- **An example**
- **Avenues of work**
 - **Formal specification of memory models**
 - **Characterization of violations**
 - **Obtaining finite-state abstractions**
 - **Obtaining finite-state models of the memory system**
 - **Combating state explosion and reporting errors**
- **Conjectures**
- **Conclusions**

Advantages of the approach

- Test automata inspired by proven architectural testing methods (lifting architectural testing techniques to the realm of model-checking)
- Can formally study properties of memory models as well as their sub-ordering rules (e.g. read orderings w/o write ordering)
- Can use test model-checking without knowing the internal details of the memory system (rapid verification regressions)
- Can use pure tests as putative queries to help understand the memory model
- Can define a variety of memory models in a uniform framework (weak as well as strong models) – hence not “hard wired” for particular models

An example: Deriving a pure test for (CMP,RO)

- Obtain execution scenario capturing ordering rule
- Create finite-state abstraction of violating executions

P1

A:=1

A:=2

..

A:=k

P2

X[1]:=A

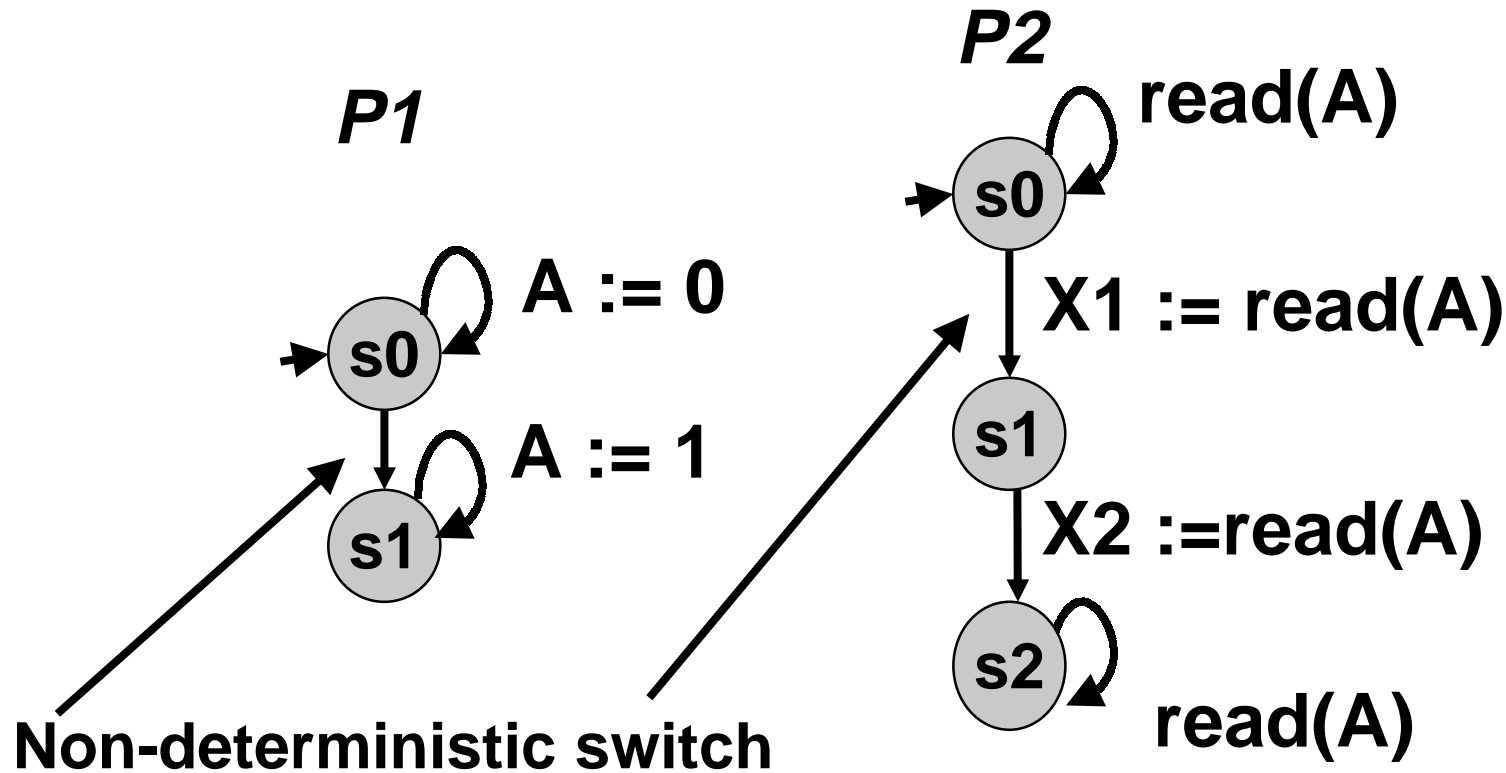
X[2]:=A

..

X[k]:=A

Condition : for all $p, q, r : p < q < r :$
 $X[p] = X[r] \Rightarrow X[p] = X[q] = X[r]$

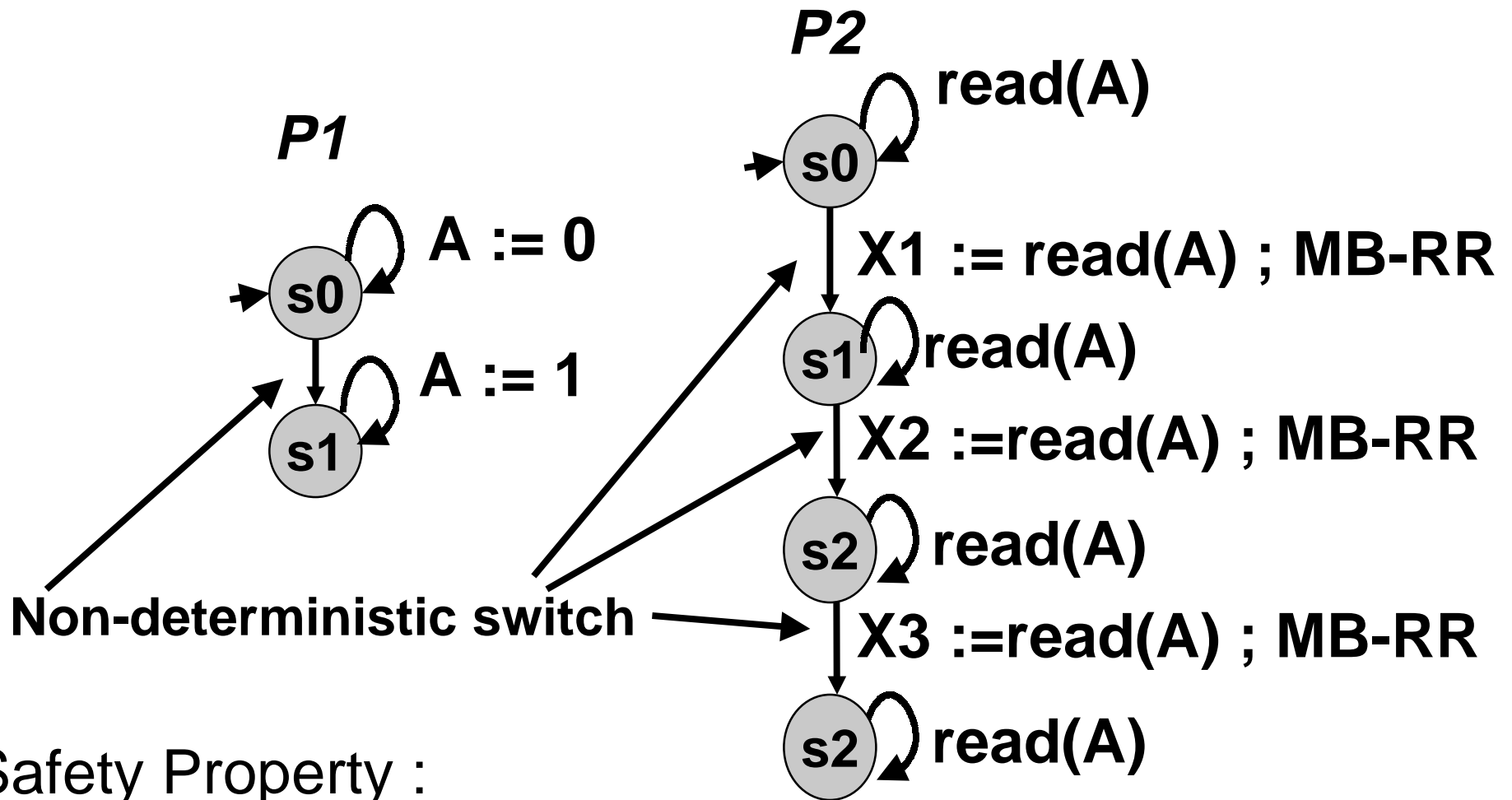
Example test for (CMP,RO)



Safety Property to be established:

$$\mathbf{P2 \text{ in final state} \Rightarrow X2 \geq X1}$$

**Test Automata for (CMP, MB-RR)
assuming that the # of MB-RRs don't matter**



Finally, $X_1 = X_3 \Rightarrow X_1 = X_2 = X_3$

Specification of memory models via architectural rules

- **CMP (Rule of Computation) :**
 - **A Basic rule specifying how operand values are computed from other operand values.**
- **RO (Read Order) :**
 - **Reads in a single process are completed in program order**
- **WO (Write Order) / PO (Program Order):**
 - **Writes / reads and writes in a single process are completed in program order**
- **WOS, POS:**
 - **Writes posted in program order in remote stores also**

Specifying Weaker memory models relaxations of orderings

- **Partial-PO Relaxation :**
 - **Relaxes PO partially - WR is always relaxed**
 - **May relax WA in various orders**
 - **examples : SPARC V9 TSO, PSO, Intel Pentium Pro, Processor consistency etc.**
- **Complete-PO relaxation :**
 - **Relaxes PO completely**
 - **typically does not relax WA**
 - **examples : SPARC V9 RMO, Alpha, PowerPC, Release Consistency**

WA relaxation via rule WA-S

- **WA :**
 - a write becomes visible to all processors
“instantly”
 - atomic set of events - all write events
- **WA-S :**
 - a write becomes visible to all other processors
“instantly”
 - atomic set of events - all write events in stores
of other processors

Some examples (informal proofs in Ghughal's MS)

- TSO = A (CMP, UPO, RO, WO, RW, WA-S, MB-WR)
- PSO = A (CMP, UPO, RO, RW, WA-S, MB-WR, MB-WW)
- IBM 370 = A(CMP, UPO, RO, WO, RW, WA, MB-WR)
- Alpha subset = A(CMP, UPO, ROO, WA-S, MB, MB-WW)

Identifying Violation Classes: some of the POS violations (1-3 of 5)

(1)	(2)	(3)																																
<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%;">P_i</td> <td style="width: 50%;">P_j</td> </tr> <tr> <td>...</td> <td>...</td> </tr> <tr> <td>rd(a, v)</td> <td>...</td> </tr> <tr> <td>...</td> <td>...</td> </tr> </table> <p>v is not the initial value T of a, and a is not written anywhere</p>	P_i	P_j	rd(a, v)	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%;">P_i</td> <td style="width: 50%;">P_j</td> </tr> <tr> <td>...</td> <td>...</td> </tr> <tr> <td>rd(a,v1)</td> <td>wr(a,v2)</td> </tr> <tr> <td>...</td> <td>...</td> </tr> <tr> <td>rd(a,v2)</td> <td>wr(a,v1)</td> </tr> <tr> <td>...</td> <td>...</td> </tr> </table> <p>P_i and P_j could be the same process</p>	P_i	P_j	rd(a,v1)	wr(a,v2)	rd(a,v2)	wr(a,v1)	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%;">P_i</td> <td style="width: 50%;">P_j</td> </tr> <tr> <td>...</td> <td>...</td> </tr> <tr> <td>rd(a,v)</td> <td>wr(a,v)</td> </tr> <tr> <td>...</td> <td>...</td> </tr> <tr> <td>rd(a,T)</td> <td></td> </tr> <tr> <td>...</td> <td></td> </tr> </table> <p>P_i and P_j could be the same process</p>	P_i	P_j	rd(a,v)	wr(a,v)	rd(a,T)		...	
P_i	P_j																																	
...	...																																	
rd(a, v)	...																																	
...	...																																	
P_i	P_j																																	
...	...																																	
rd(a,v1)	wr(a,v2)																																	
...	...																																	
rd(a,v2)	wr(a,v1)																																	
...	...																																	
P_i	P_j																																	
...	...																																	
rd(a,v)	wr(a,v)																																	
...	...																																	
rd(a,T)																																		
...																																		

...All one-address PO violations (4-5 of 5)

(4) P_i

...

rd(*a*,*v*)

...

wr(*a*,*v*)

...

v is not the initial
value *T* of *a*, and
a is not written
before being read

(5) P_i

...

wr(*a*,*v*)

...

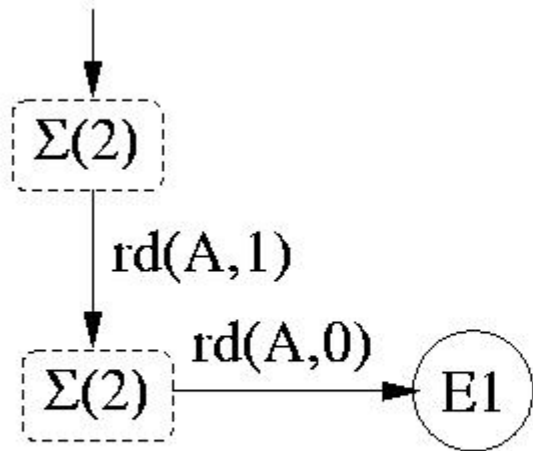
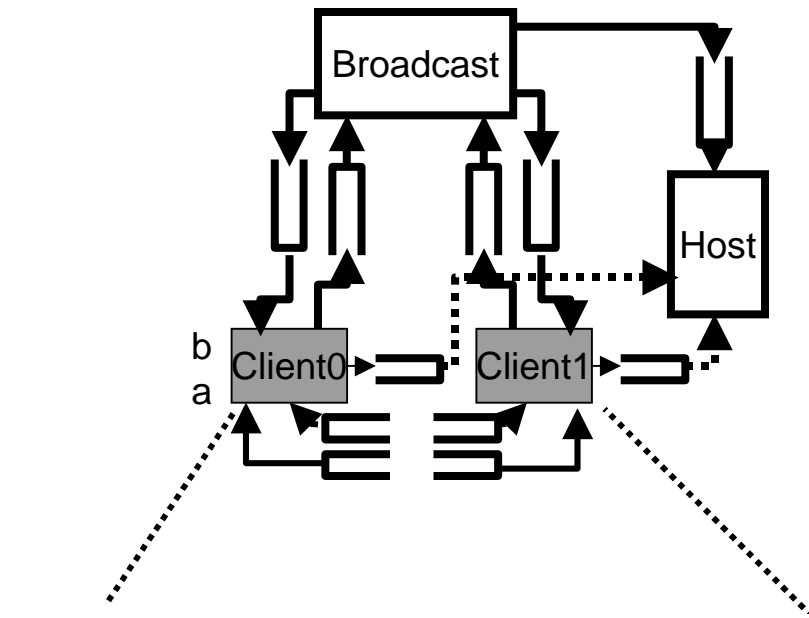
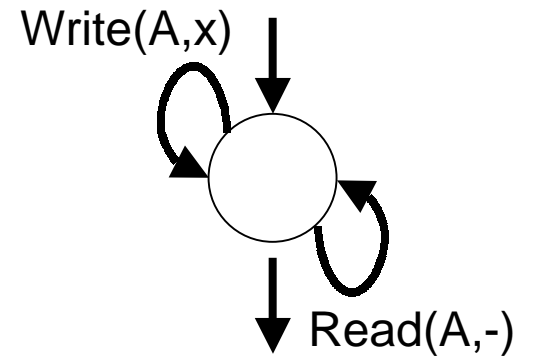
rd(*a*,*T*)

...

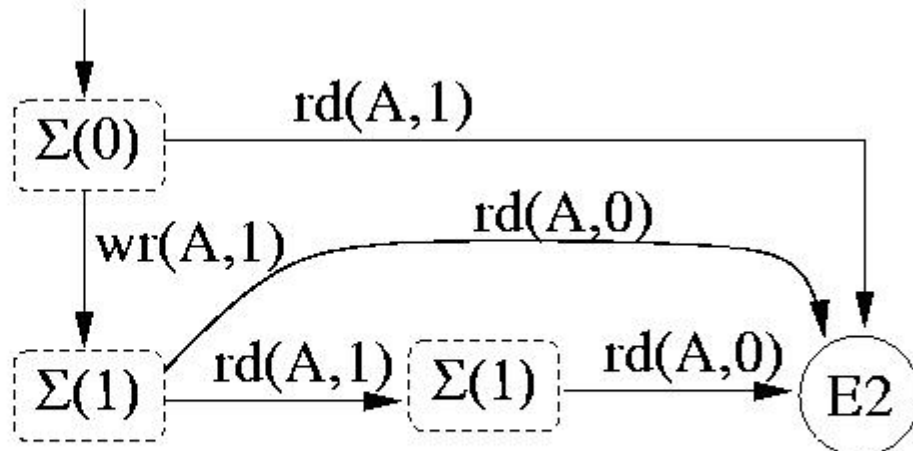
**The formal status of these violations is under study
(believed to be exhaustive for one address)**

Verification of Program Ordering for all one-address programs

$\Sigma(x)$ means



P



Q

Error states: E1, E2

Experimental result

- A model of the MIT HCN protocol was created in Murphi by Mike Jones (details in the next presentation)
- He applied his parameterized verification technique to identify network classes in which to verify
- The PRAM test on the previous page was applied
- Subtle coding error revealed pretty quickly

*Message: Often cheaper to verify sub-ordering rules than the whole
(in this case SC)*

Creating a test automaton based on violation scenarios

Pure Test for RO

- different operands
- WO not assumed

<i>P1</i>	<i>P2</i>	<i>P3</i>
B:=1	X := A;	U := C;
	Y := B;	A := U;
	C := Y;	

- Initially all vars == 0
 - Finally all vars == 1
- => In P2, B must have been read before A

Pure Test for RO

- different operands
- WO not assumed

P1

B:=1
B:=2
..
B:=k

P2

Y[0] := 0;
X[1] := A;
Y[1] := B;
C := Y[i];
X[2] := A;
Y[2] := B;
C := Y[2];
..
X[k] := A;
Y[k] := B;
C := Y[k];

P3

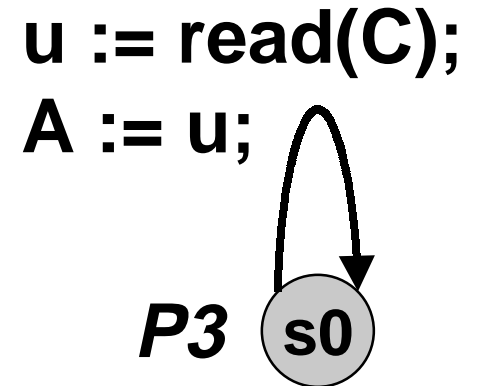
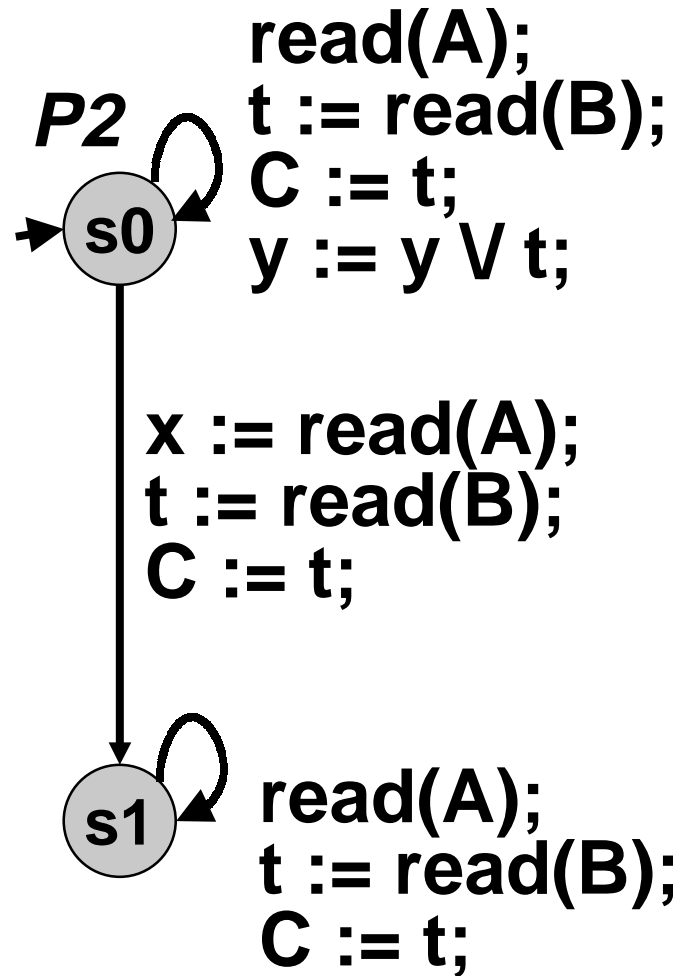
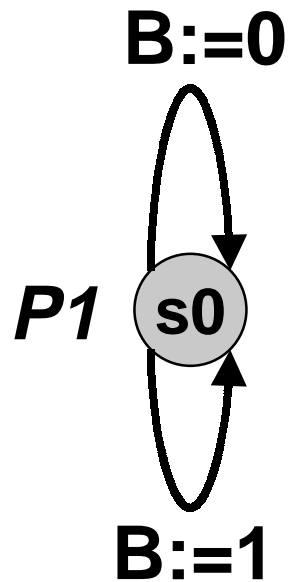
U[1] := C;
A := U[1];
U[2] := C;
A := U[2];
..
U[k] := C;
A := U[k];

Condition : Exists $i:1 \leq i \leq k$
Forall $j:0 \leq i: X[i] \neq Y[j]$

“X is getting ahead of all the Y’s so far” -- need to examine a history of values...

Turn into OR accumulator via data-independence!

Test Automata for RO (diff opnds)



Safety Property :
(P2 in S1 \wedge y==0)

\Rightarrow x==0

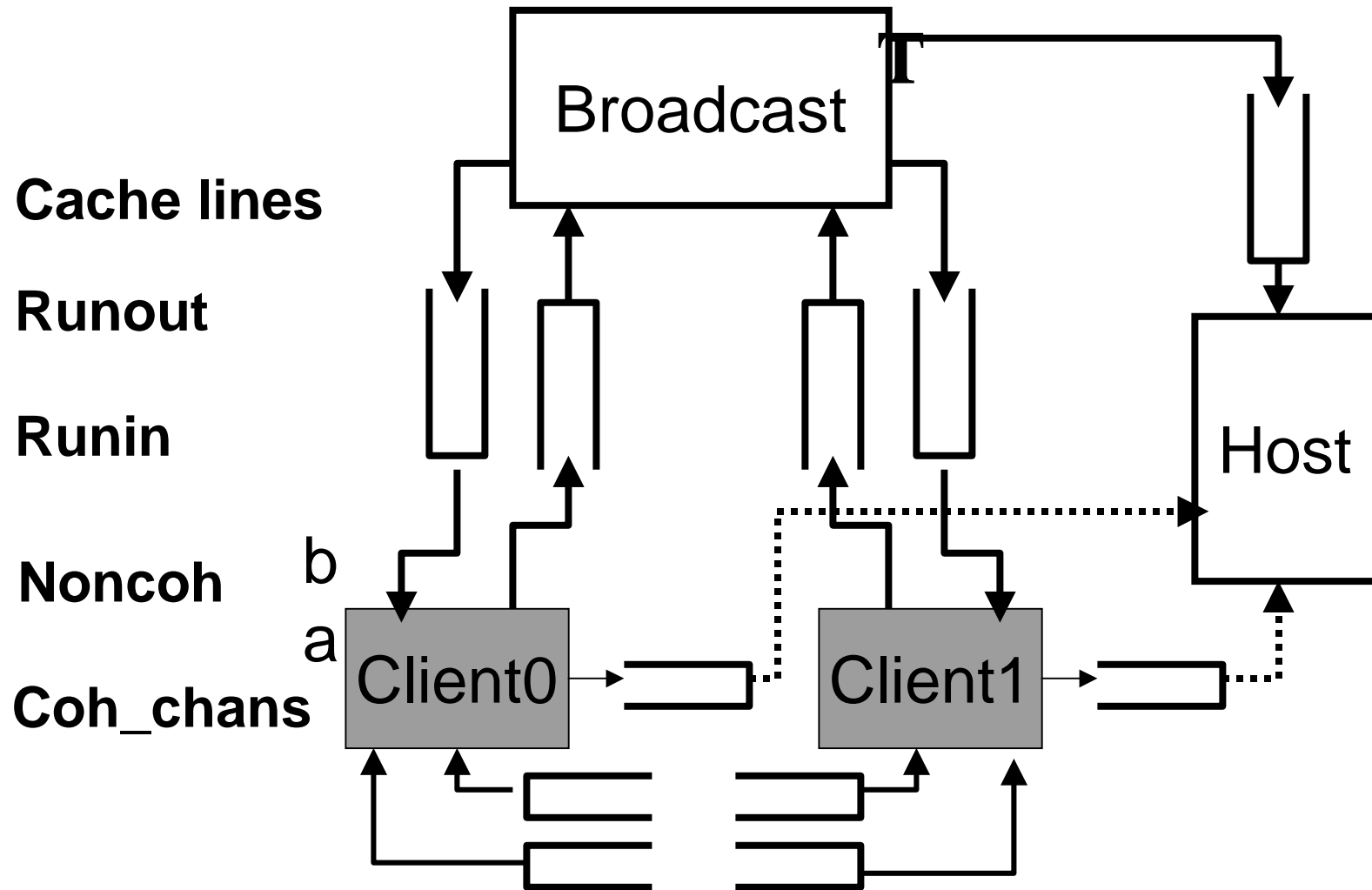
Characterization of violations: a summary

- Test model-checking allows different violation classes to be studied and reused
- Abstract interpretation techniques seem essential to use in order to derive sound abstractions

Modeling memory systems

- Good modeling languages will allow succinct specifications to be written
- They may even help meet certain assumptions automatically for verification (e.g. address projectability and data independence)
- Explicit or implicit state? None seem to have an edge (see next slides)
- Reductions appear to be quintessential (tried Partial Order Reductions so far)

Formal Specification of Memory Systems



Experimental Results (VIS)

PO	States	Bdds	Time
Serial Memory	7k	7k	9 sec
Lazy Caching	7.8M	306k	36 min
PA8000	953k	1.6M	27 hrs



WA	States	Bdds	Time
Serial Memory	212k	10k	34 sec
Lazy Caching	1.9M	513k	59 min
PA8000	985k	1.7M	40 hrs

SC verification of the HP/Runway model Promela, with SPIN and PV (#states)

	Spin	PV
PO-1	56K	2794
PO-2	> 5M/DNF	11M
SC-1	499K	7880
SC-2a	> 5M/DNF	5.9M
SC-2b	> 4M/DNF	574K

Some results as well as conjectures

- $(\text{CMP}, \text{UPO}, \text{WO}, \text{RO}, \text{RW})$ has been argued to be the same as $(\text{CMP}, \text{POS}_c)$ where POS_c is as in RAPA
- Assuming CON to be part of WA-S, Ghughal has shown that $(\text{CMP}, \text{WOS}_c, \text{WA-S}) = (\text{CMP}, \text{WO}, \text{WA-S})$
- It has been shown that $(\text{CMP}, \text{UPO}, \text{CON}) \Rightarrow (\text{CMP}, \text{UPO}, \text{CON}, \text{WA-S}_i)$ where WA-S_i is the “intra” component of WA-S
- It appears that we don't need SRW if we have RW and RO
- Is $(\text{CMP}, \text{UPO}, \text{WO}, \text{RO}, \text{RW})$ the same as PRAM or (CMP, POS) ?
 - Reason to think so: PRAM tests don't fail on TSO
 - We define TSO as $(\text{CMP}, \text{UPO}, \text{RO}, \text{WO}, \text{RW}, \text{WA-S}, \text{MB-WB})$

Concluding Remarks

- We've studied test model-checking for over 3 years
- Many formal proofs remain tightened (tedious and error-prone)
- An experimental system based on sound (but incomplete) automata will get us going, prove useful for others, and may help debug our proofs quicker
- We anticipate building such a framework as our logical next step
- Some questions:
 - How to verify those memory systems that really get designed against memory models that really get used?
 - How to cut needless variety (without clear performance payoffs)?
 - Can test model-checking be used in other situations (e.g. JVM validation)?