

# Partial Order Reduction for Scalable Testing of SystemC TLM Designs

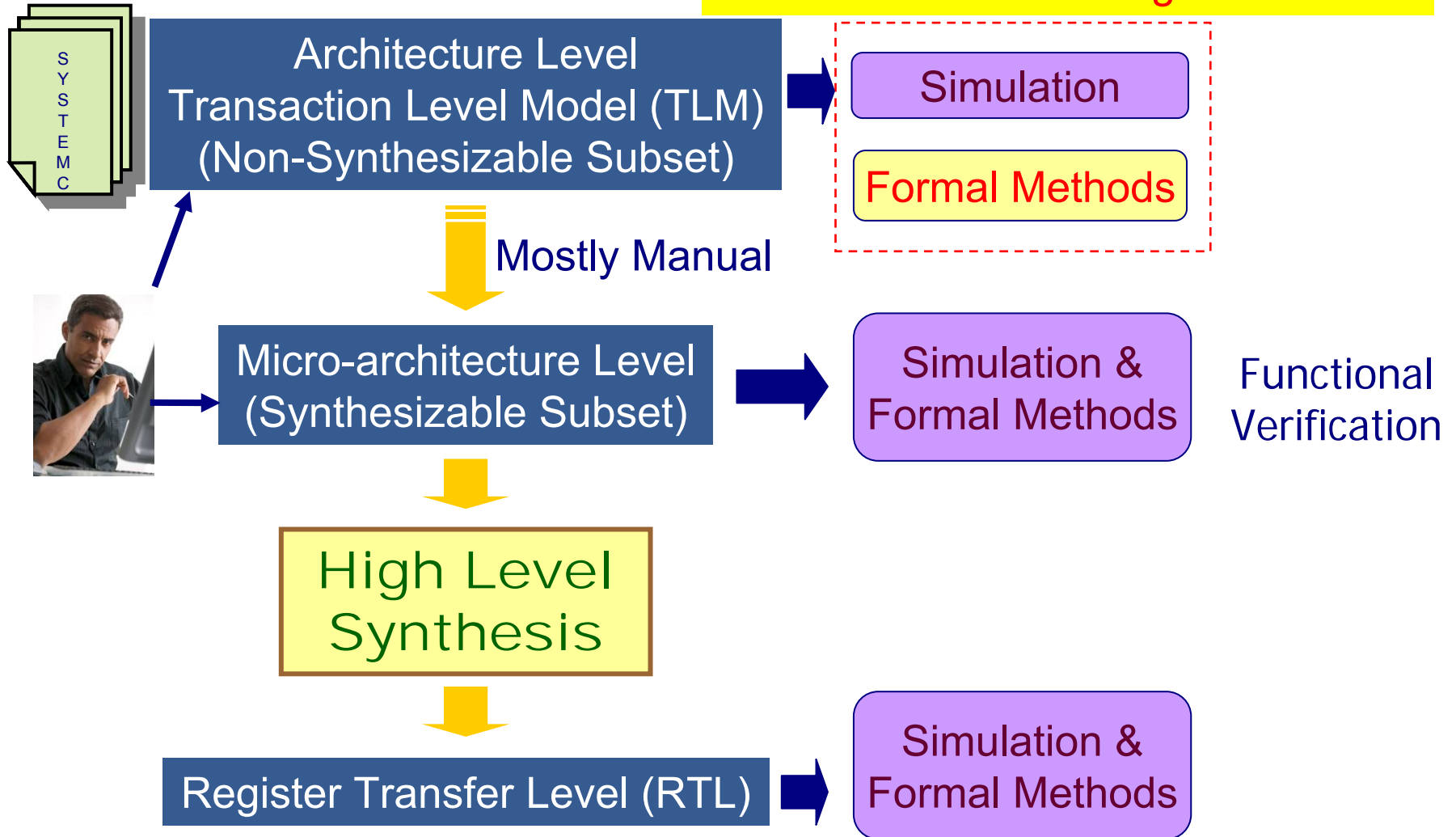
Sudipta Kundu, University of California, San Diego

Malay Ganai, NEC Laboratories America

Rajesh Gupta, University of California, San Diego

# Hardware Design Methodology

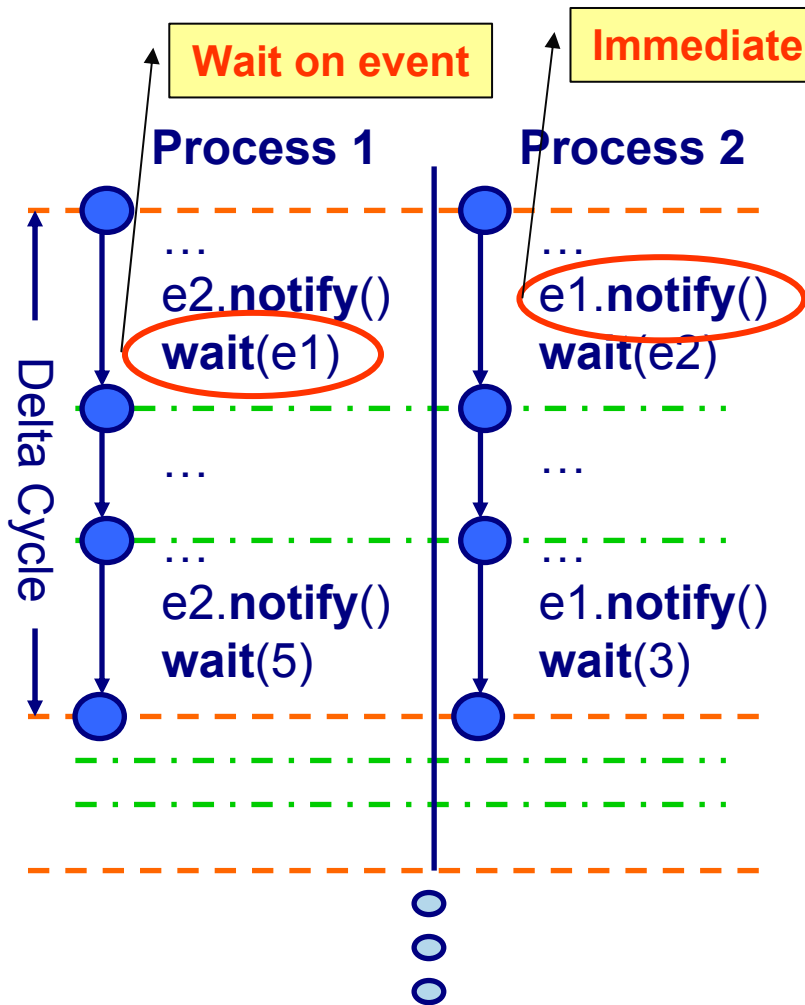
Can be 3 orders of magnitude faster



# Outline

- **Motivation**
- **Background**
  - **SystemC Semantics**
  - **Partial-order Reduction**
- **Our Approach**
  - **Static Analysis**
  - **Query-based Framework: Satya**
- **Experiments**
- **Conclusion**

# Semantics of SystemC

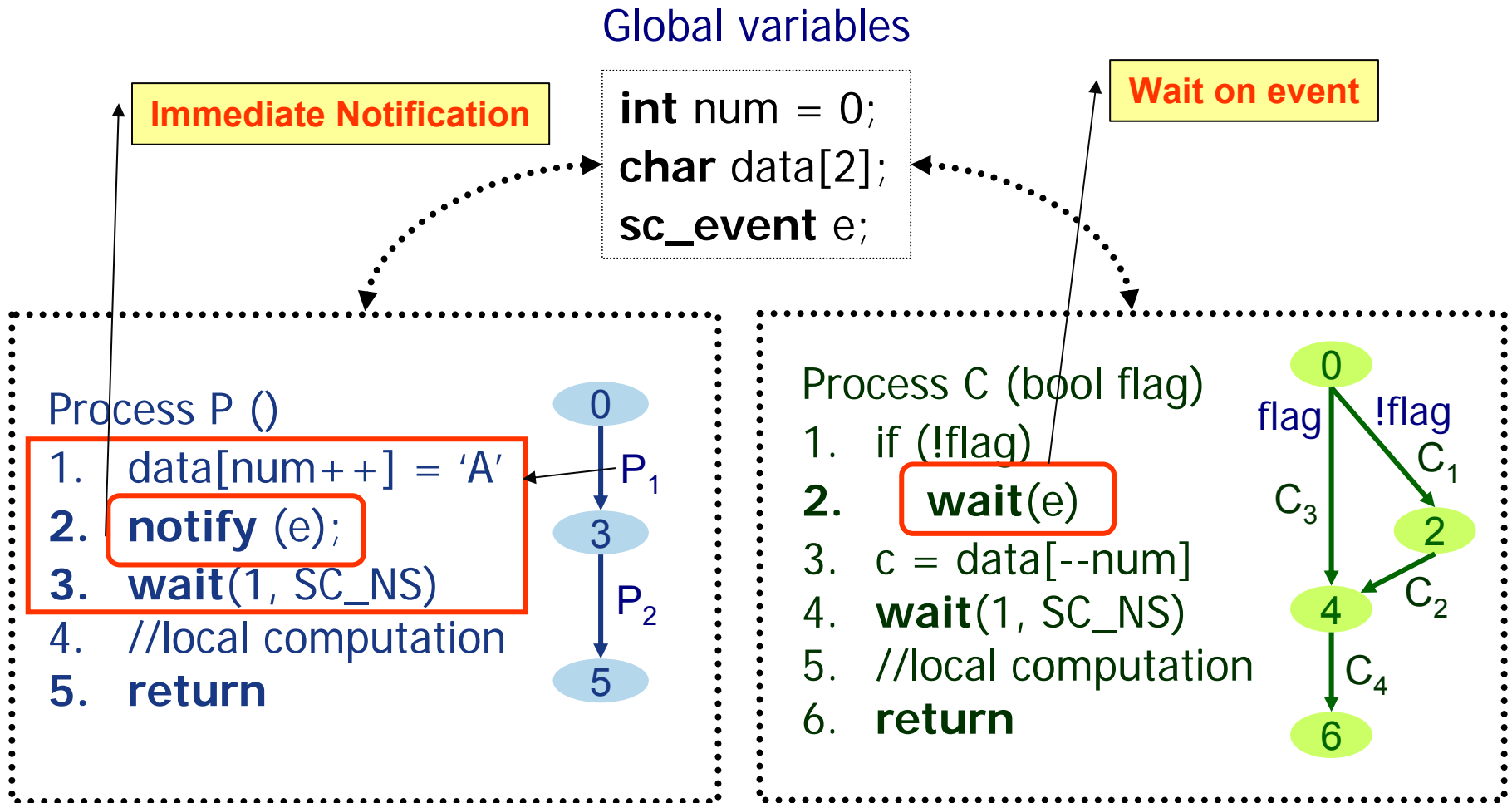


- C++ library
- Co-operatively Multitasking
- Asynchronous and Synchronous concurrency
- Variables
  - Signals : Blocking variables
  - Non-signals : Non-blocking variables

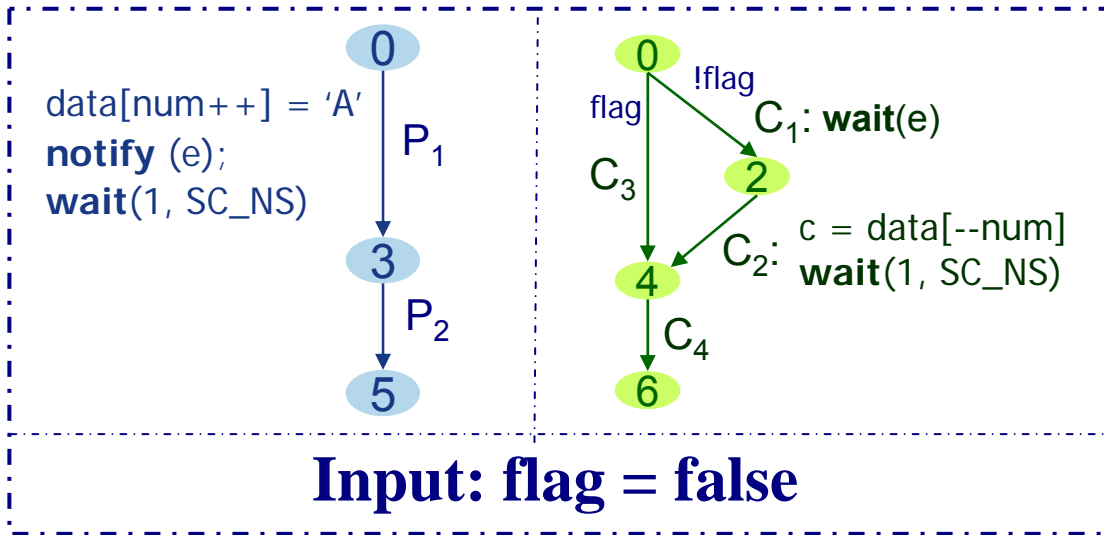
## Design Errors:

- Deadlock
- Write Conflicts
- Assertion Violations
- Data Races

# Example: Producer-Consumer



# Single Interleaving not enough ..



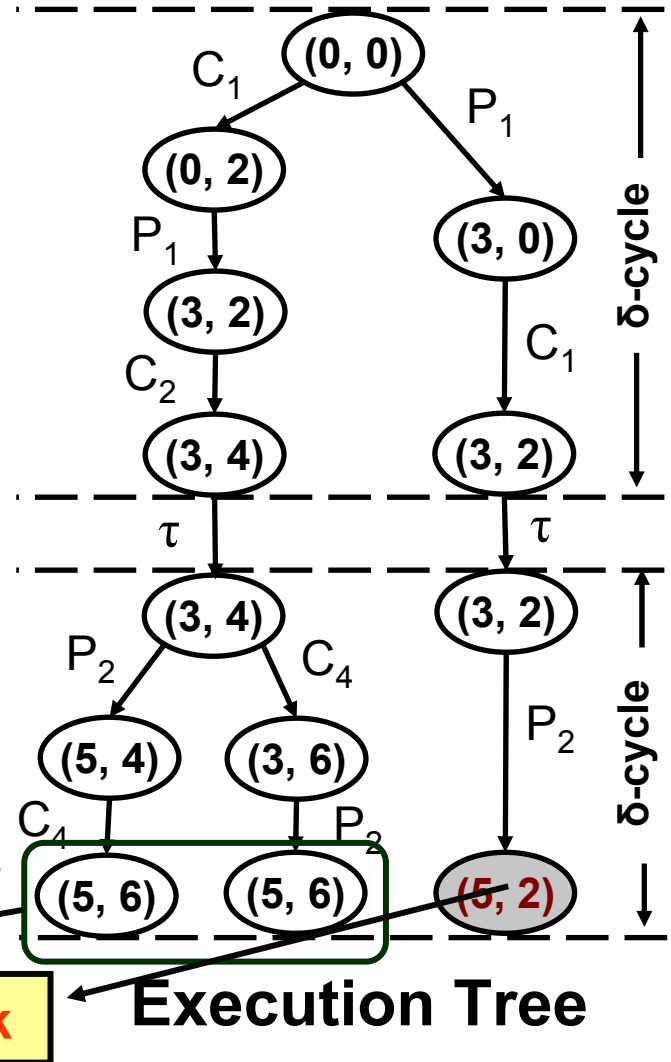
## Problem 1:

SystemC scheduler is deterministic

- For given input it explores only one interleaving

## Problem 2:

Exponential number of possible interleavings

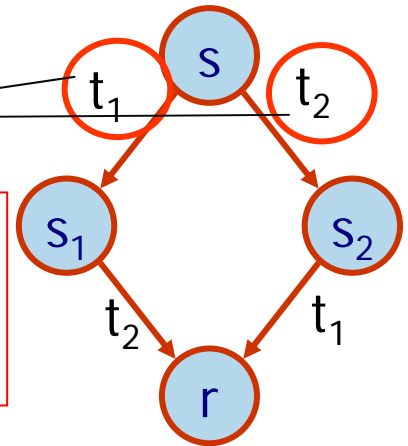


# Partial Order Reduction (POR)

- Reduces the interleaving that needs to be searched
- Exploits the **commutative** of concurrently executed transitions.

Enabled transitions

$t_1$  and  $t_2$  are commutative (independent)  
⇒ Explore interleaving  $t_1.t_2$  or  $t_2.t_1$  (not both)

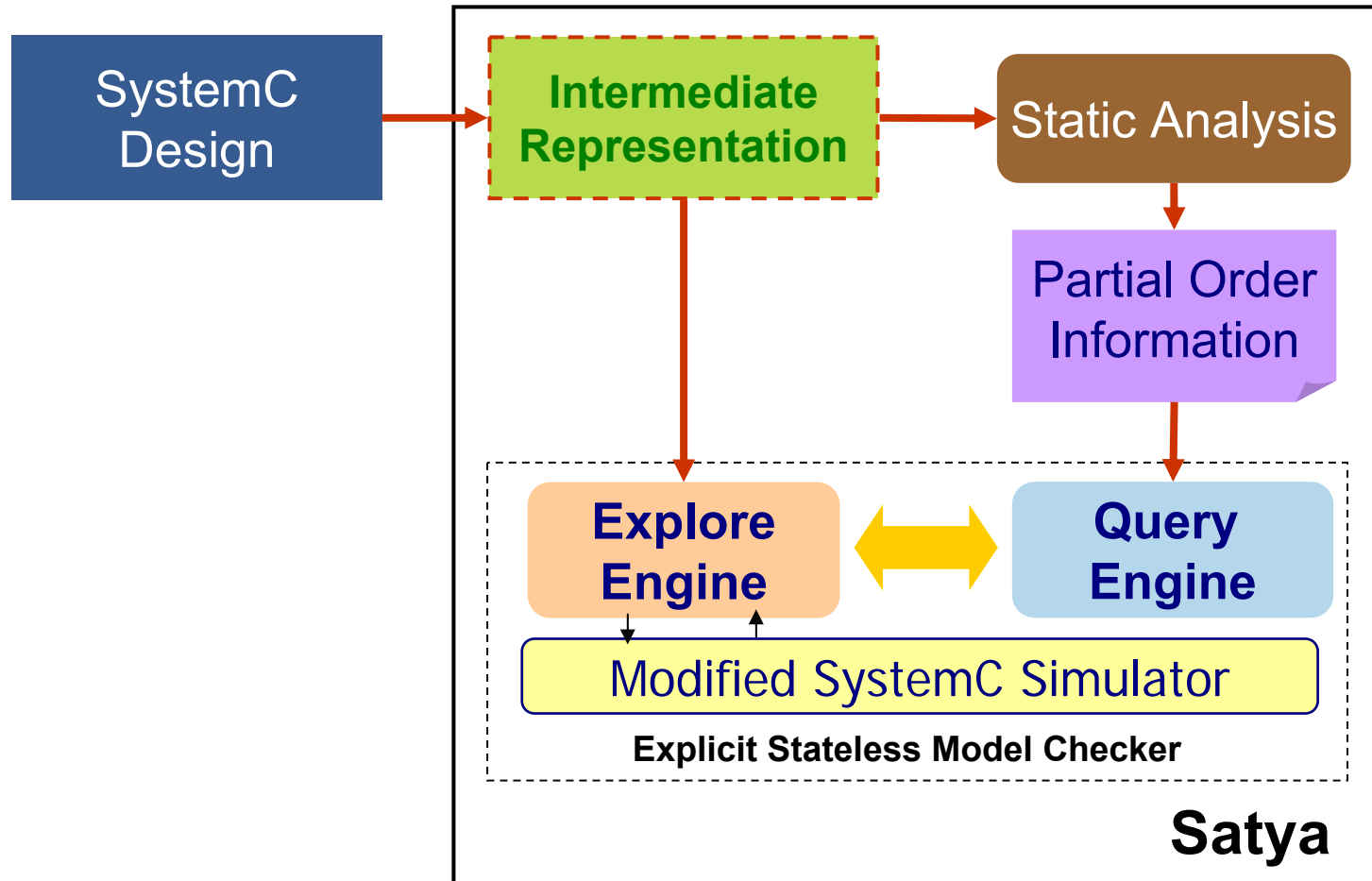


- Concurrent Software Verification
  - **Static POR** [Godefroid 95]
  - **Dynamic POR** [Flanagan 05]

# Our Approach: Overview

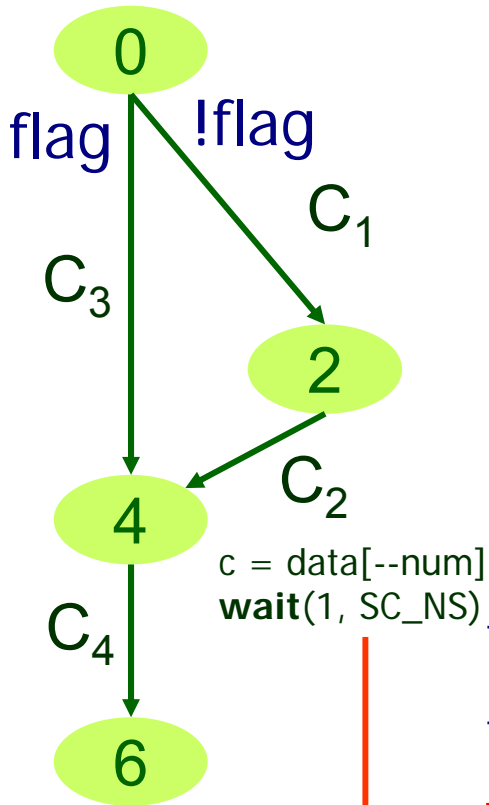
- Adapts POR techniques for SystemC TLM Designs
- Exploits SystemC specific semantics
  - Co-operatively multitasking
  - Wait to wait atomic block
  - Notion of  $\delta$ -cycle
  - Signal (blocking) variables
- We implemented a query-based framework
  - Combines static and dynamic POR techniques

# Our Framework: Satya



**Satya** is a Sanskrit word that translates into English as "**truth**" or "**correct**."

# Static Analysis: Basic Steps



1. Get a control skeleton.
2. Find out the wait boundaries (atomic blocks)
3. Summarize static informations ( $W_{ns}$ ,  $R_{ns}$ ,  $W_s$ ,  $R_s$ , Notify, Wait)
4. Compute the dependence relation between atomic blocks. (next slide)

ID	$W_{ns}$	$R_{ns}$	$W_s$	$R_s$	Notify	Wait
$C_1$	-	-	-	-	-	e
$C_2$	num	data, num	-	-	-	(1, SC_NS)
$C_3$	num	data, num	-	-	-	(1, SC_NS)
$C_4$	-	-	-	-	-	-

ns – non signal

s - signal

# Dependence Relation ( $\mathcal{D}$ )

Given two transitions (atomic blocks)  $t_1$  and  $t_2$ ,  $(t_1, t_2) \in \mathcal{D}$  if

- A write on shared **non-signal** variable  $v$  in  $t_1$  and a read or a write on the same variable  $v$  in  $t_2$ . (**data dependency**)

OR

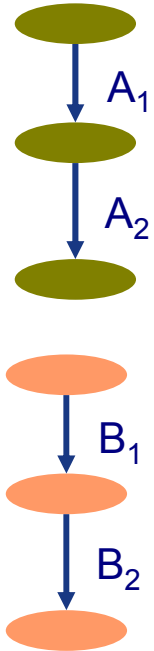
- A write on a shared **signal** variable  $s$  in  $t_1$  and a write on the same variable  $s$  in  $t_2$ . (**write-write conflict**)

OR

- A wait on an **event**  $e$  in  $t_1$  and an immediate notification on the same event  $e$  in  $t_2$ . (**causal dependency**)

Special Case: We consider **symmetric writes** (increment, decrement) on non-signals as independent.

# Dependence Relation: Example



ID	$W_{ns}$	$R_{ns}$	$W_s$	$R_s$	Notify	Wait
A <sub>1</sub>	i	-	-	-	-	e
A <sub>2</sub>	x	-	-	s	-	-

ID	$W_{ns}$	$R_{ns}$	$W_s$	$R_s$	Notify	Wait
B <sub>1</sub>	-	-	s	-	e	(1, SC_NS)
B <sub>2</sub>	i	x	-	-	-	-

Dependent?		
	A <sub>1</sub>	A <sub>2</sub>
B <sub>1</sub>	YES	NO
B <sub>2</sub>	NO	YES

Query Table

i++  
Symmetric write

Data Dependency

No conflict (global variable)

Causal Dependency

# Our Explore Algorithm

Runnable Runnable Sleep Sleep Todo

$\langle \{ P_1, P_2, C_1, C_2, C_3, C_4 \}, \{ P_1, C_1 \} \rangle$

Scheduler State =  $\langle \text{Runnable, Sleep, Todo} \rangle$

1. Randomly execute an execution path till some depth
2. Analyze the path bottom up considering each  $\delta$ -cycle separately.
3. If there exist a transition in  $(\text{Todo} \setminus \text{Sleep})$  then execute it from start (as our algorithm is stateless).

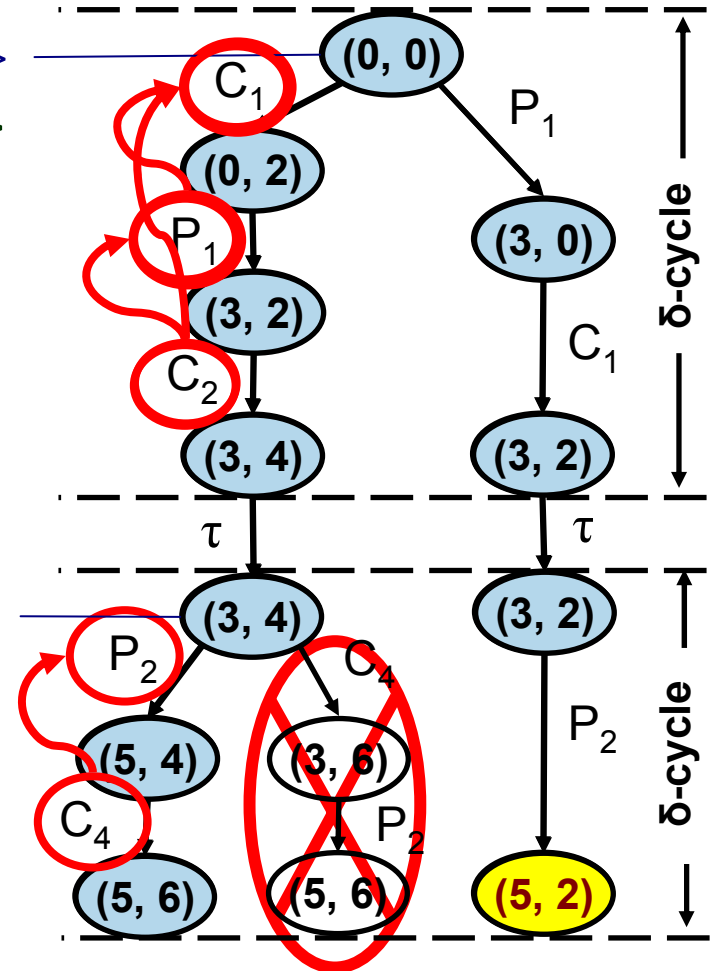
Runnable Runnable Sleep Sleep Todo

$\langle \{ P_2, P_4, C_4 \}, \{ P_2 \} \rangle$

Dependent?				
	C <sub>1</sub>	C <sub>2</sub>	C <sub>3</sub>	C <sub>4</sub>
P <sub>1</sub>	YES	YES	YES	NO
P <sub>2</sub>	NO	NO	NO	NO

Query Engine

Is (P<sub>1</sub>, C<sub>1</sub>)  
Dependent?



Execution Tree

# Our Contributions

- Commutative checks between the transitions are *not* done across  $\delta$ -cycles (not required)
- Low cost commutative checks
  - No book-keeping for dynamic reads and writes
  - Use pre-computed query table
- Conservative approach
  - Independent transitions are precise, but not the dependent ones
- Dependent transitions identified statically are *most likely* dependent
  - Large wait to wait atomic blocks
  - Signal variables are commonly used

# Experiments and Results 1/2

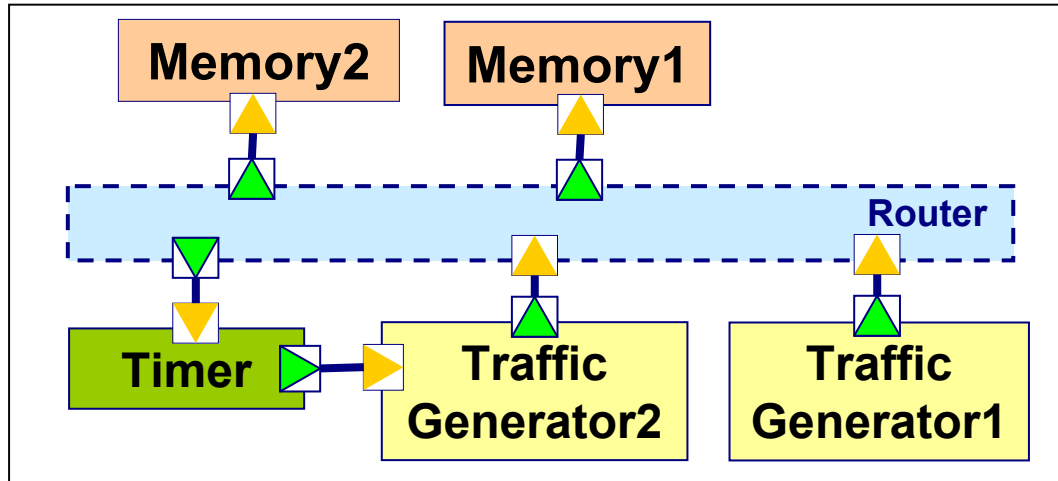
- **No POR** – Explore all execution paths
- **POR** – Our Approach using POR

## Fifo Benchmark

- Open SystemC Initiative (OSCI) Repository
- Array Bound Violation (2 producer, 1 consumer)

<b>Elements produced</b>	<b>Total #traces</b>	<b>Time (no POR) (sec:msec)</b>	<b>Reduced #traces</b>	<b>Time (POR) (sec:msec)</b>
14	8	00:046	6	00:032
28	80	00:469	42	00:265
44	992	06:344	318	02:313
62	13376	93:563	2514	19:031

# Experiments and Results 2/2



## Transaction Accurate Communication Benchmark (TAC)

- ST Microelectronics
- 6 modules – 2 traffic generators, 2 memories, 1 timer, 1 router
- Static slicing of the router while testing for deadlock

#Transactions	Total #traces	Time (no POR) (min:sec)	Reduced #traces	Time (POR) (min:sec)
80000	12032	89:47	1	00:13

# Conclusion and Future Work

- We presented Satya, a query-based approach build over SystemC Simulator
  - Compute and use static information efficiently
- We exploit SystemC specific semantics
  - Reduces interleaving that are needed to explore
- Improve previous explore algorithm
  - Avoids book-keeping cost
  - Avoid dynamic commutative checks
- In future,
  - We are working on intelligent test bench generation

**THANK YOU**