Efficient Approaches for Functional Validation of SoC Designs Using High-Level Specifications

Mingsong Chen

Shanghai Key Laboratory of Trustworthy Computing

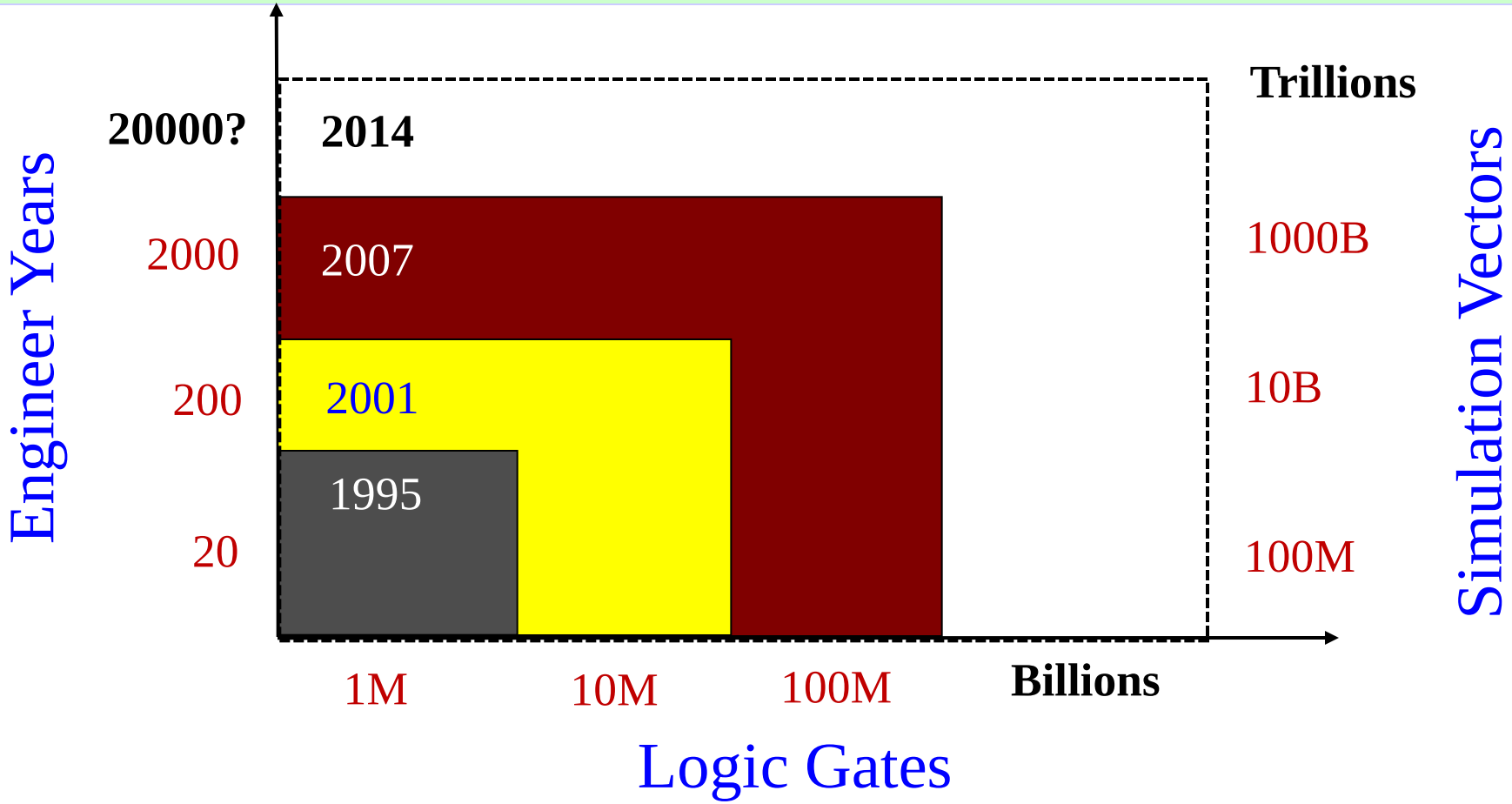
Software Engineering Institute, East China Normal University

October 14, 2013

Outline

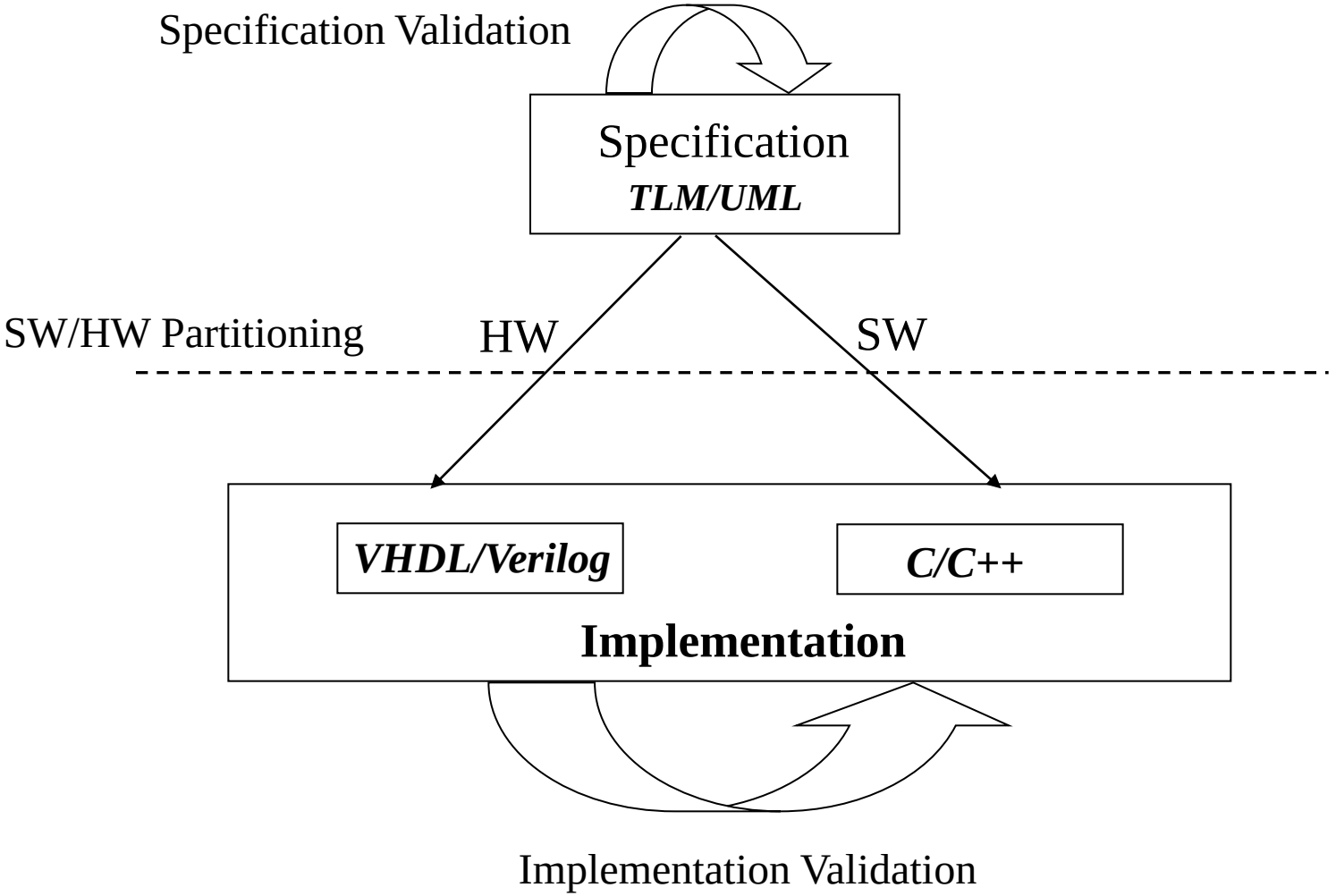
- **Motivation**
- **Research Work**
  - Automatic Validation of SoC Specifications
    - Modeling of SoC specifications
    - Basic Idea of Our Approach
  - Efficient Test Generation using Learning Methods
    - Novel property clustering approaches
    - Decision ordering based learning techniques
    - Property decomposition approaches
  - Automated Reuse using Validation Refinement Techniques
- **Conclusion**

Functional Validation of SOC Designs

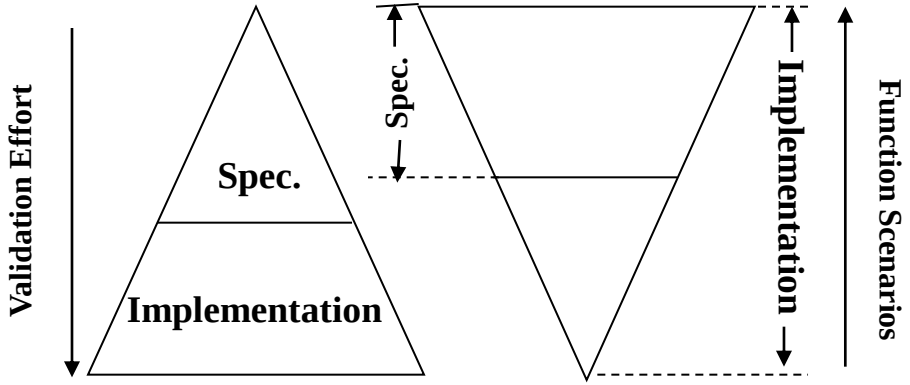


Functional validation is a major bottleneck during SoC development! (up to 70% of time and resources are used)

SoC Design and Validation Flow



Challenges and Opportunities



Potential Improvements

- **Current Approach**

$$F_{spec} * T_{spec} + (F_{spec} + F_{imp}) * T_{imp}$$

- **Our Approach**

$$F_{spec} * T_{spec} / \alpha + F_{spec} * T_{imp} / \beta + F_{imp} * T_{imp}$$

Minimize: $F_{spec} * T_{spec} + (F_{spec} + F_{imp}) * T_{imp}$

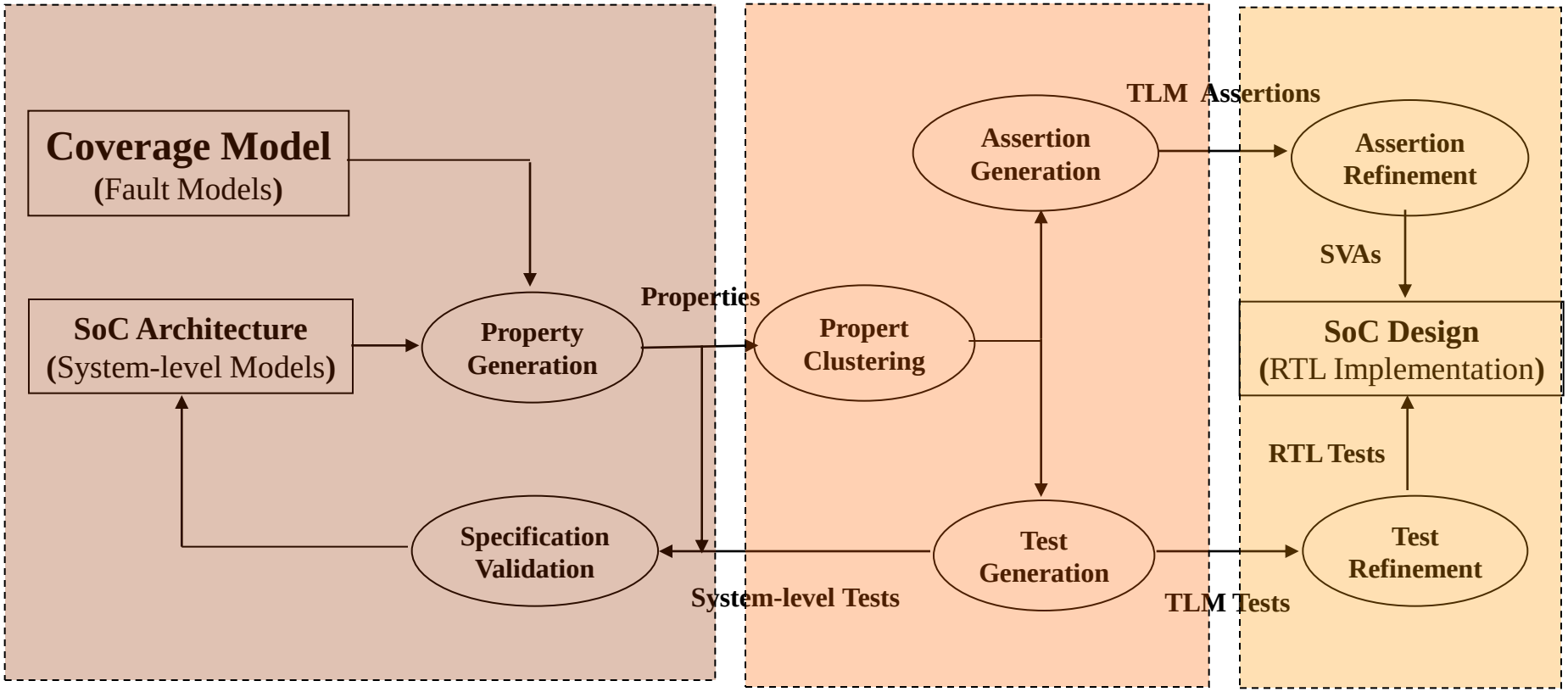
Subject to:

$$F_{spec} + F_{imp} = F_{Total}$$

$$T_{spec} << T_{imp}$$

$$F_{spec} > F_{imp}$$

Our Approach



Automatic Validation of SoC Specifications | **Efficient Test Generation** | **Validation Reuse**

Outline

- Motivation
- **Research Work**
  - Automatic Validation of SoC Specifications
    - Test Generation using Model Checking
    - Basic Idea of Our Approach
  - Efficient Test Generation using Learning Methods
    - Novel property clustering approaches
    - Decision ordering based learning techniques
    - Property decomposition approaches
  - Automated Reuse using Validation Refinement Techniques
- Conclusion

Test Generation using Model Checking

❑ Model Checking (MC)

- ❖ SoC formal models in temporal specification language, e.g., SMV
- ❖ Desired behaviors in temporal logic properties, e.g. LTL
- ❖ Property falsification leads to counterexamples (tests)

❑ Test Generation

- ❖ ***Generate a counterexample: sequence of variable assignments***

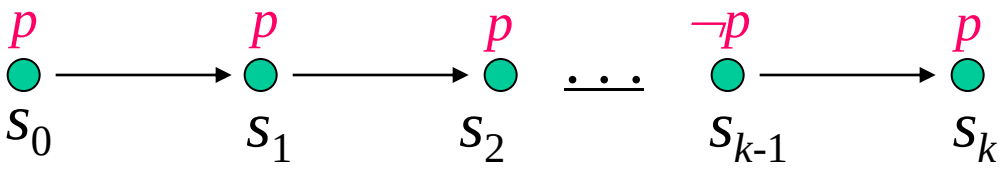
Problem: **Test generation is very costly or not possible in many scenarios -- in the presence of complex SoCs and/or complex properties.**

Approach: **Exploit learning to reduce validation complexity**
- Reduction of TG time & memory requirements
- Enables test generation in complex scenarios

SAT-based Bounded Model Checking

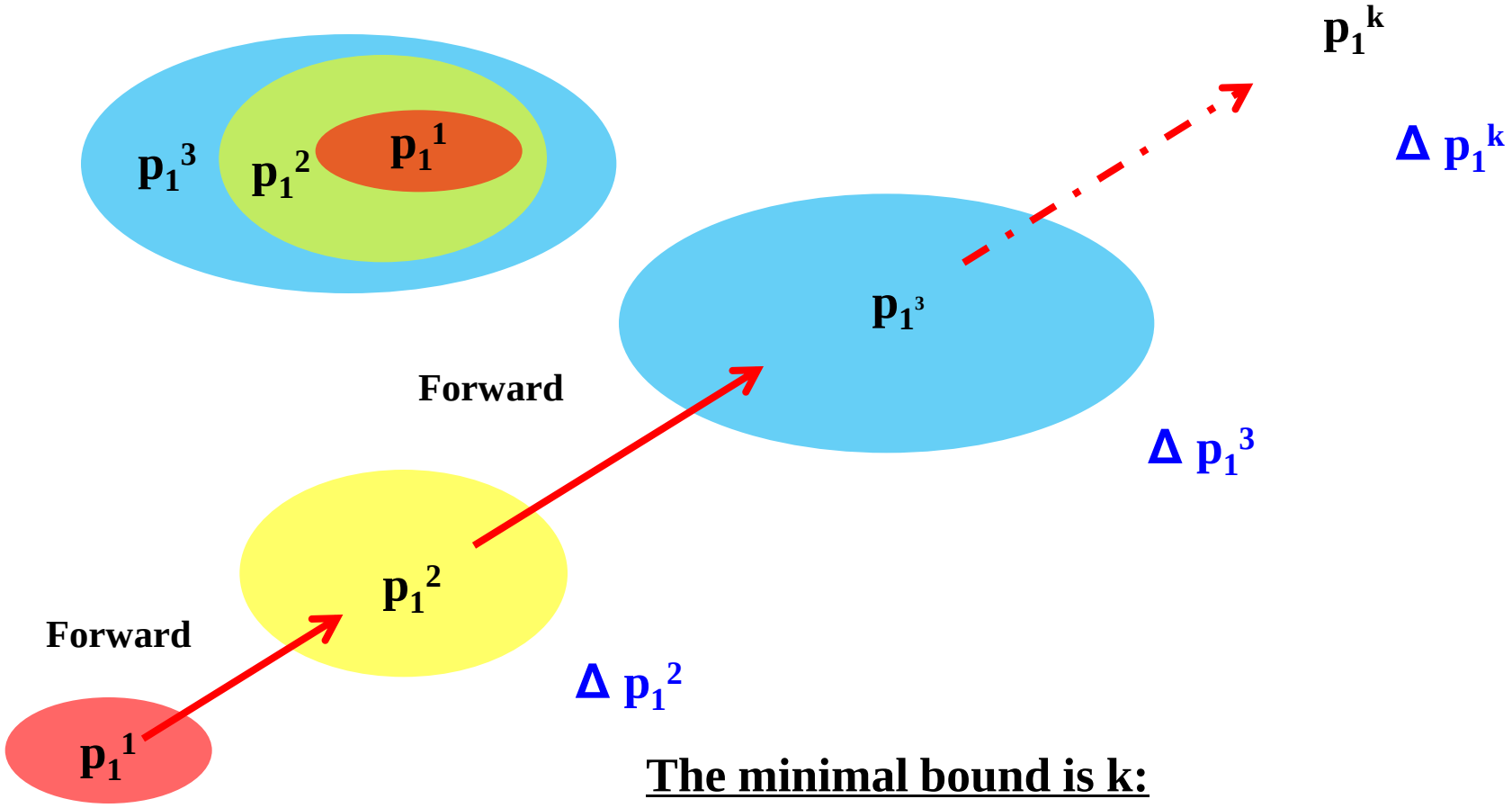
- The safety property ***P*** is valid up to cycle ***k*** iff $\Omega(k)$ is not satisfiable.

$$\Omega(k) = I(S_0) \wedge \bigwedge_{i=0}^{k-1} R(S_i, S_{i+1}) \wedge \bigvee_{i=0}^{k} \neg P(s_i)$$



- If $\Omega(k)$ is satisfiable, then we can get an assignment which can be translated to a test.

Same Property but Different Bounds

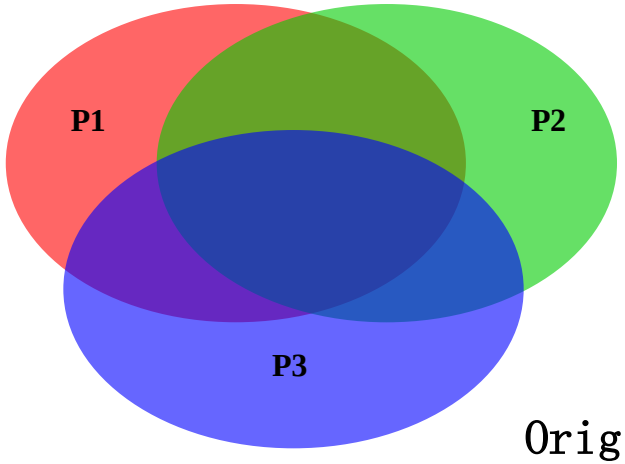


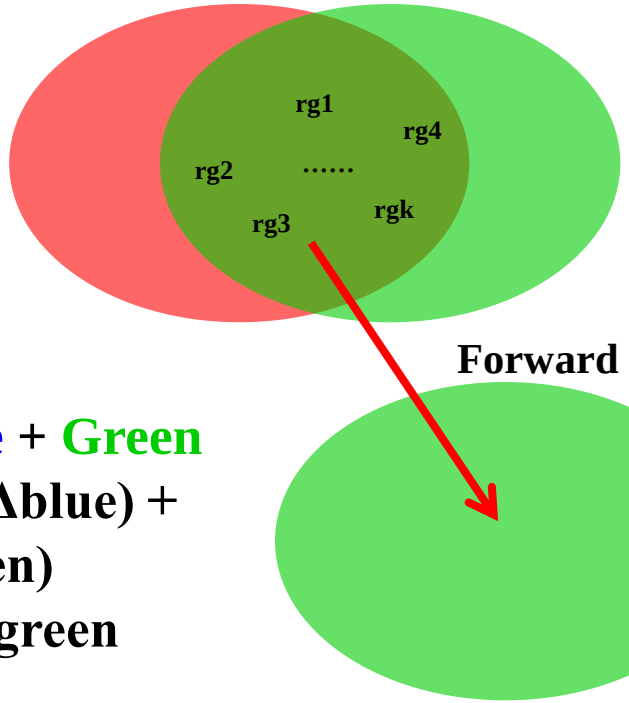
The minimal bound is k:

Save: $\Delta P_1^2 + \Delta p_1^3 + \ldots + \Delta p_1^{k-1} + \ldots + \Delta p_1^k$

O. Strichman. Pruning Techniques for the SAT-Based Bounded Model Checking Problems. ***CHARME , 2001***

Same Design, Different Properties



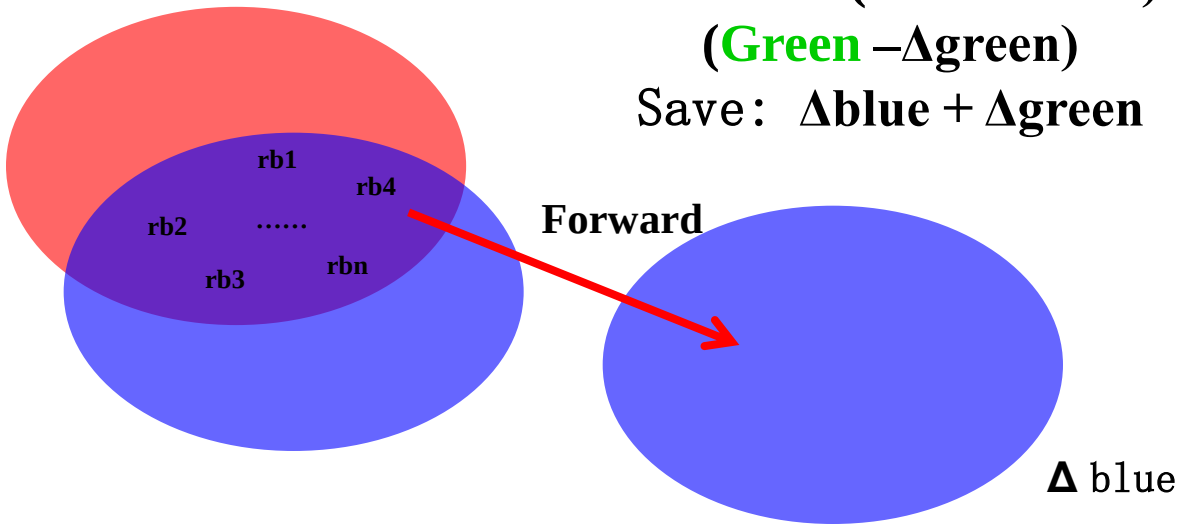


Benefit:

Original: **Red** + **Blue** + **Green**

Now: **Red** + (**Blue** –**Δblue**) + (**Green** –**Δgreen**)

Save: **Δblue** + **Δgreen**



Outline

- Motivation
- **Research Work**
  - Automatic Validation of SoC Specifications
    - Test Generation using Model Checking
    - Coverage-driven property generation
  - Efficient Test Generation using Learning Methods
    - Novel property clustering approaches
    - Decision ordering based learning techniques
    - Property decomposition approaches
  - Automated Reuse using Validation Refinement Techniques
- Conclusion

Property Clustering

- **Clustering properties is to exploit the structural and behavior similarity and maximize the validation reuse**
- **Property clustering methods:**
  - Based on structural similarity
  - Based on textual similarity
  - Based on Influence (Cone of Influence)
  - Based on CNF intersections

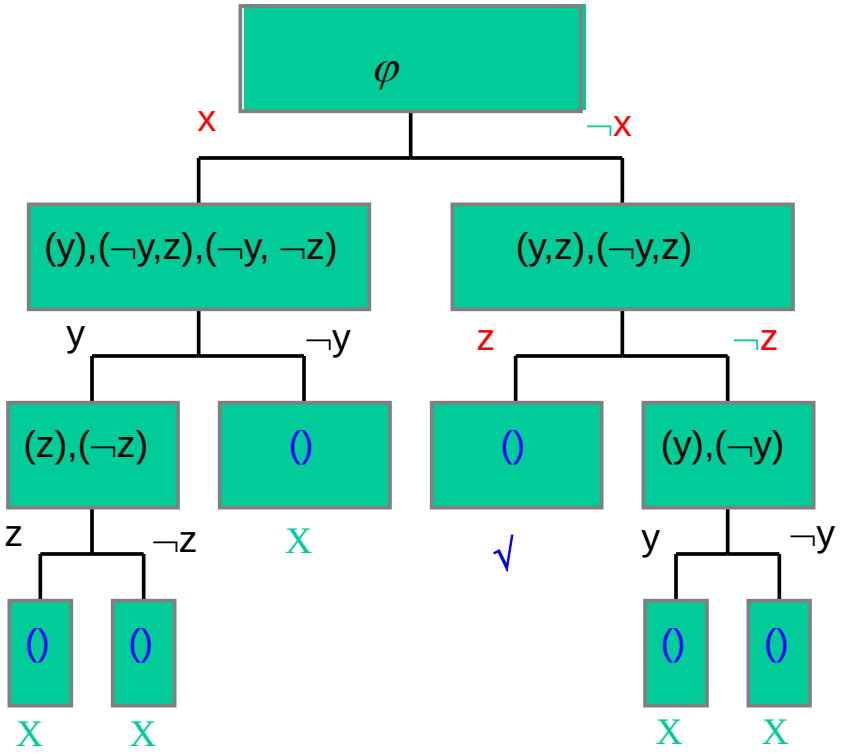
M. Chen and P. Mishra. Functional Test Generation using Efficient Property Clustering and Learning Techniques. IEEE *Transactions on CAD, 29(3), 2010*

Outline

- Motivation
- **Research Contributions**
  - Automatic Validation of SoC Specifications
    - Test Generation using Model Checking
    - Coverage-driven property generation
  - Efficient Test Generation using Learning Methods
    - Novel property clustering approaches
    - Decision ordering based learning techniques
    - Property decomposition approaches
  - Automated Reuse using Validation Refinement Techniques
- Conclusion

Decision Ordering Problem

Given a φ in CNF: (x+y+z)(¬x+y)(¬y+z)(¬x+¬ y+¬ z)

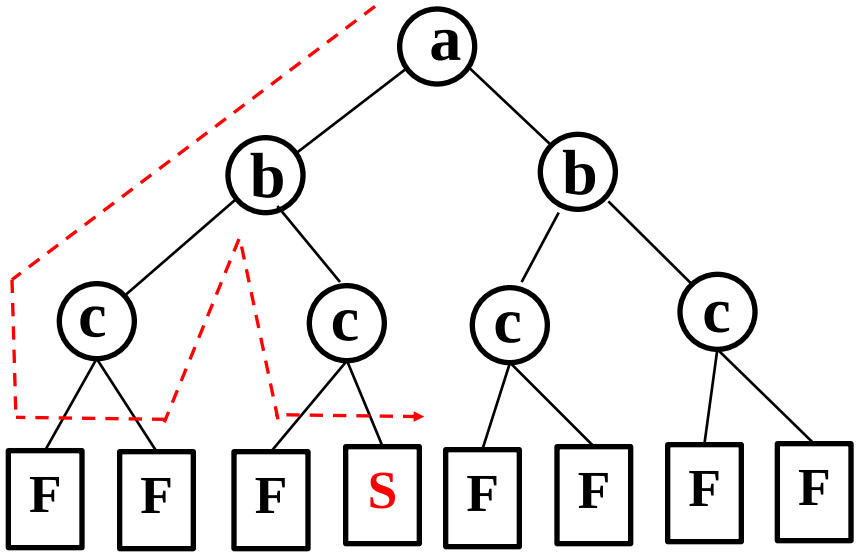


- **The essence of SAT problem is to find a satisfiable assignment for a Boolean formula.**
- **A wise decision ordering can quickly locate the true assignment.**
  - **Bit value ordering**
  - **Variable Orderinig**

Best decision: ¬ x, z

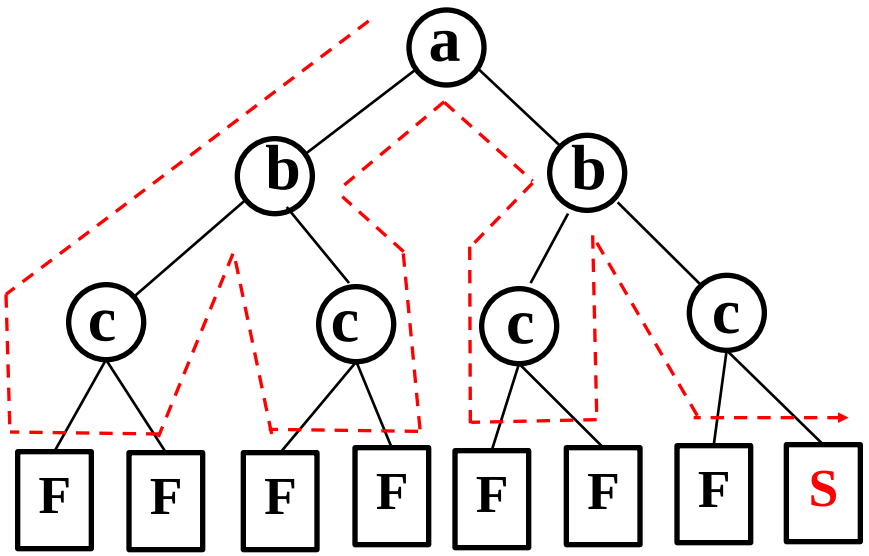
Two Similar SAT Problems

SAT 1



Ordering: a, a', b, b', c, c'

SAT 2



Ordering: a, a', b, b', c, c'

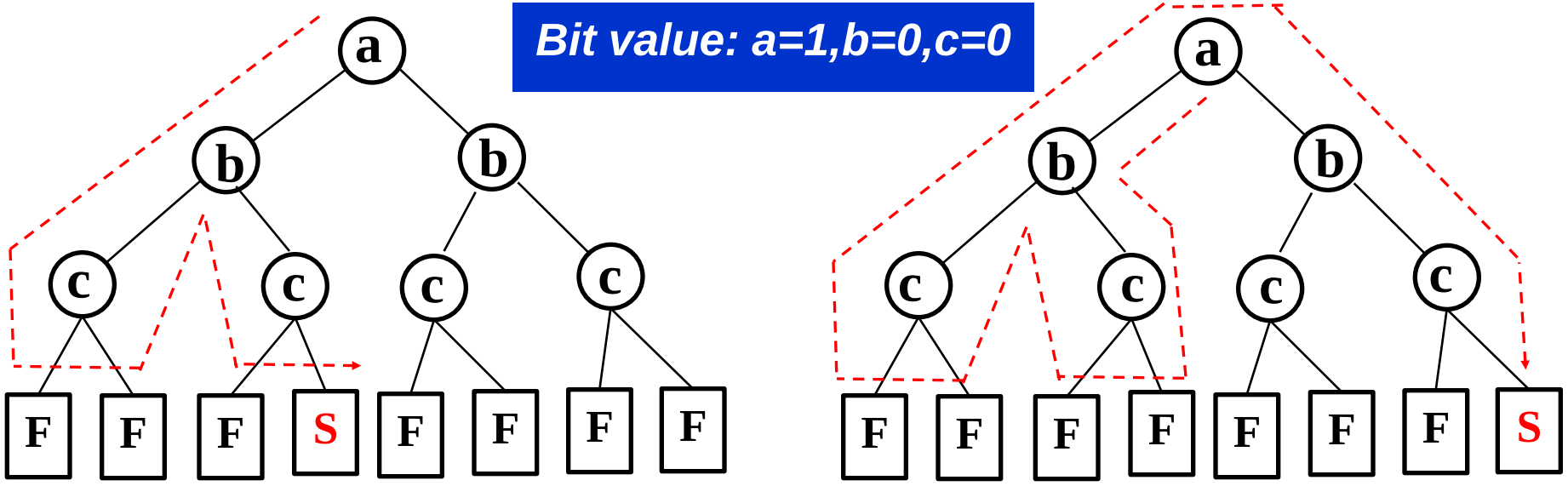
Without Learning, 7 conflicts in SAT2.

Learning: Bit Value Ordering

SAT 1

SAT 2

Bit value: a=1,b=0,c=0

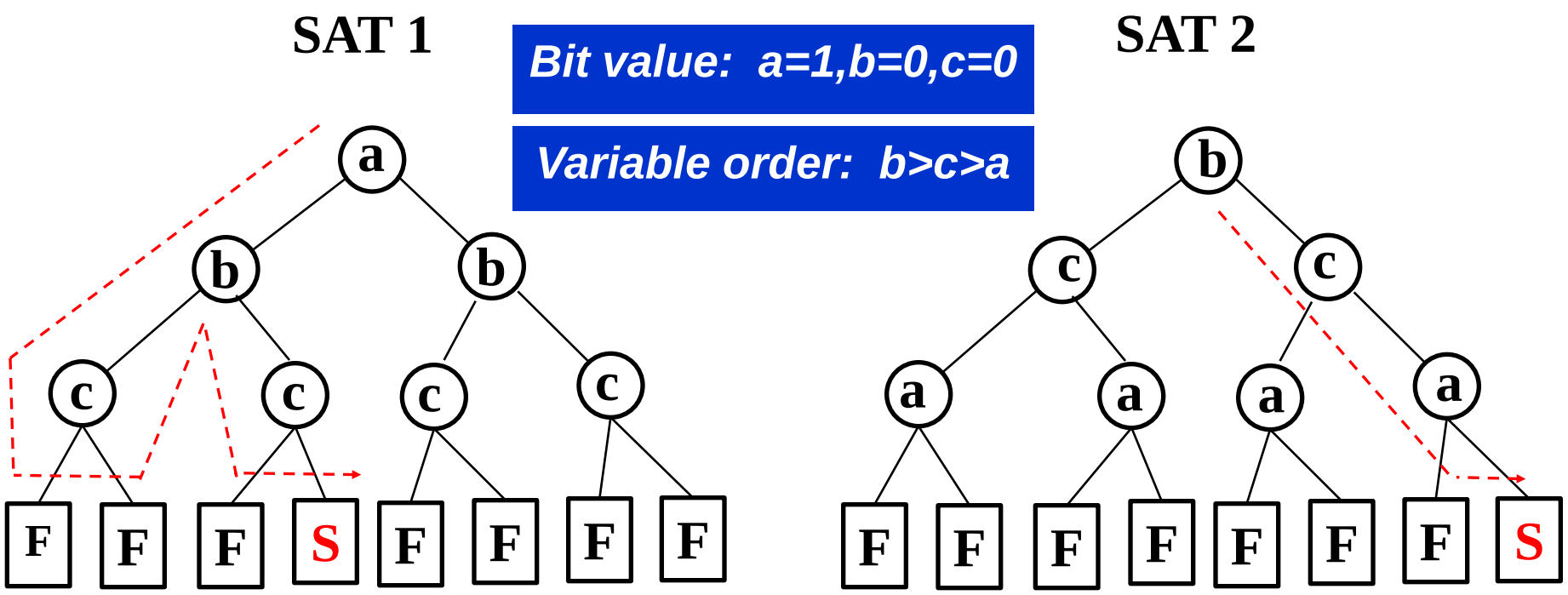


Ordering: a, a', b, b', c, c'

Ordering: a, a', b', b, c', c

With bit value learning, 4 conflicts in SAT2.

Learning: Bit Value + Variable Ordering



Ordering: a, a', b, b', c, c'

Ordering: b', b, c', c, a, a'

With bit value+ variable order learning, 1 conflict in SAT2.

Our method – An Example with 3 properties

VarStat	a	b	c	d	…
[0] V	0	0	0	0	…
[1] V	0	0	0	0	…

P1: a=0, b=0, c=1, d=1

VarStat	a	b	c	d	…
[0] V	**1**	**1**	0	0	…
[1] V	0	0	**1**	**1**	…

P2: a=0, b=0, c=1, d=0

VarStat	a	b	c	d	…
[0] V	**2**	**2**	0	**1**	…
[1] V	0	0	**2**	1	…

Predict ordering for P3

P3: a=0, b=0, c=1, d=?

score(a) ↑, score(a')↑

score(b) ↑, score(b')↑

score(c) ↑, score(c')↑

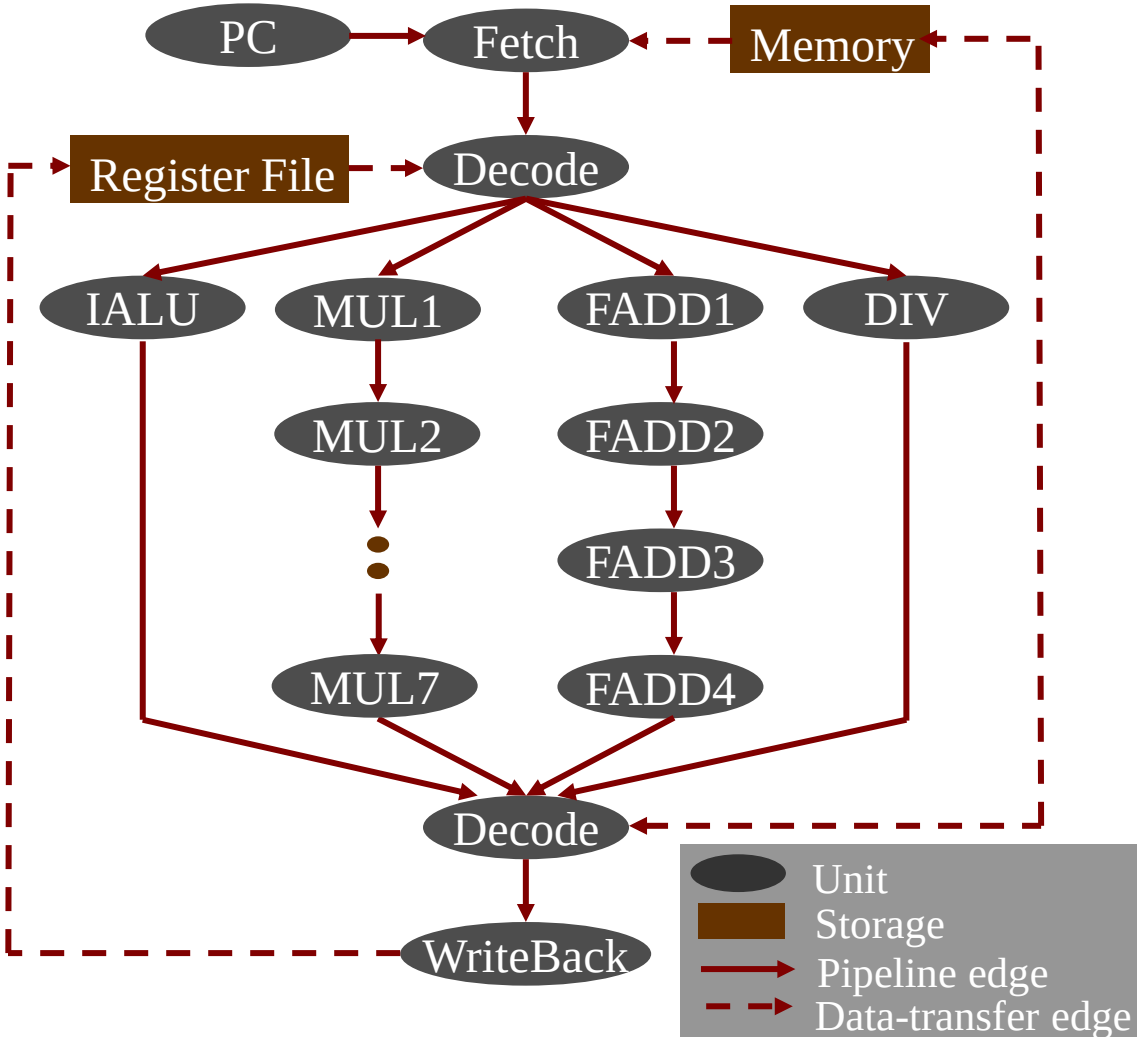
Approach: Using the statistics of the counterexamples when checking the properties in a cluster

- Count of values ➔ bit value ordering
- Variance of counts of two literals ➔ variable ordering

M. Chen, X. Qin and P. Mishra. Efficient Decision Ordering Techniques for SAT-based Test Generation. ACM/IEEE Design Automation and Test in Europe (*DATE), March 2010

M. Chen and P. Mishra. Property Learning Techniques for Efficient Generation of Directed Tests. *IEEE Transactions on Computers, 2011*

Case Study 1 : MIPS Processor



The Architecture

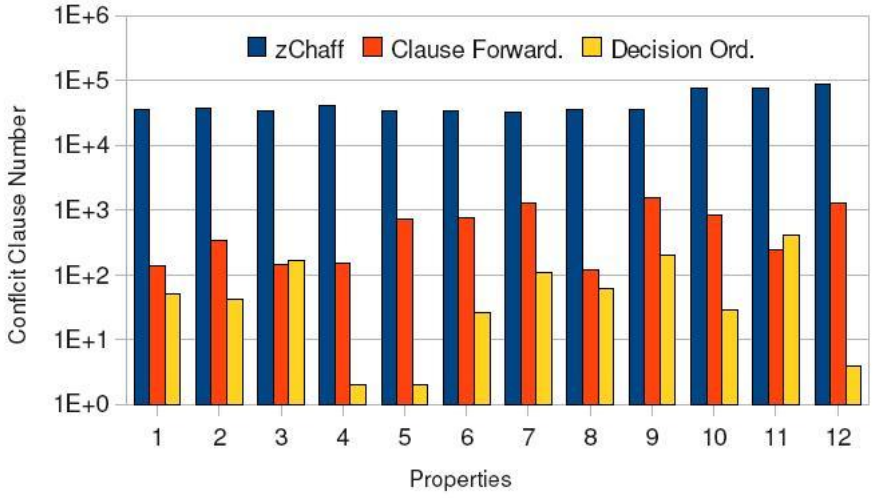
MIPS Processor
- 20 nodes
- 24 edges
- 91 instructions

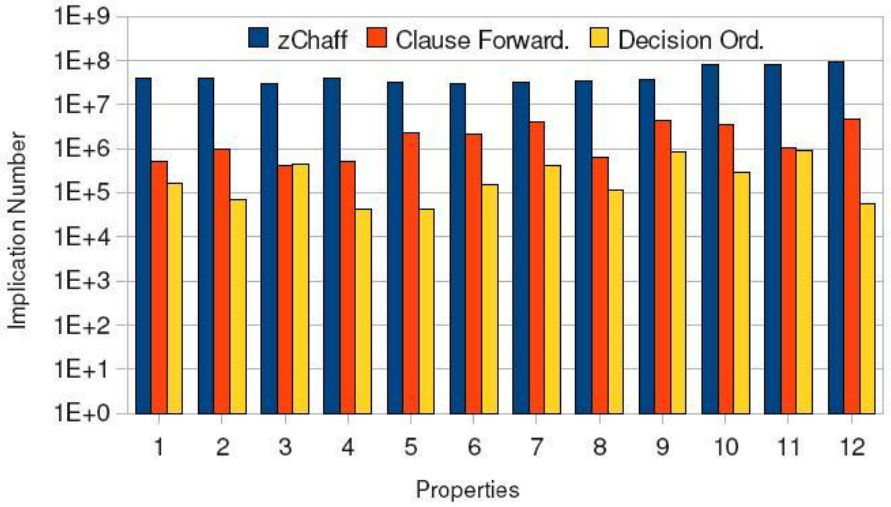
Case Study 1 : MIPS Processor

- For each function unit (ALU, DIV, FADD and MUL) in the pipelined processor. We generate 4 properties.

Property (test)	zChaff (sec)	Clustering	Speedup (over zChaff)	Decision Ordering	Speedup (over Clustering)
ALU	23.20	23.20	**1**	23.20	**1**
P1	20.73	2.74	**7.57**	0.18	**15.22**
P2	21.33	3.01	**7.09**	0.15	**20.07**
P3	18.03	2.70	**6.68**	0.29	**9.31**
DIV	18.78	18.78	**1**	18.78	**1**
P4	23.55	2.72	**8.66**	0.13	**20.92**
P5	18.31	3.60	**5.09**	0.14	**25.71**
P6	18.11	3.72	**4.87**	0.18	**20.67**
FADD	22.90	22.90	**1**	22.90	**1**
P7	16.95	4.46	**3.80**	0.23	**19.39**
P8	18.89	2.71	**6.97**	0.16	**16.94**
P9	19.80	4.70	**4.21**	0.39	**12.05**
MUL	64.21	64.21	**1**	64.21	**1**
P10	59.15	3.36	**17.60**	0.24	**14.00**
P11	59.65	3.85	**15.49**	0.45	**8.56**
P12	73.98	6.28	**11.78**	0.18	**34.89**

Case Study 1 : MIPS Processor

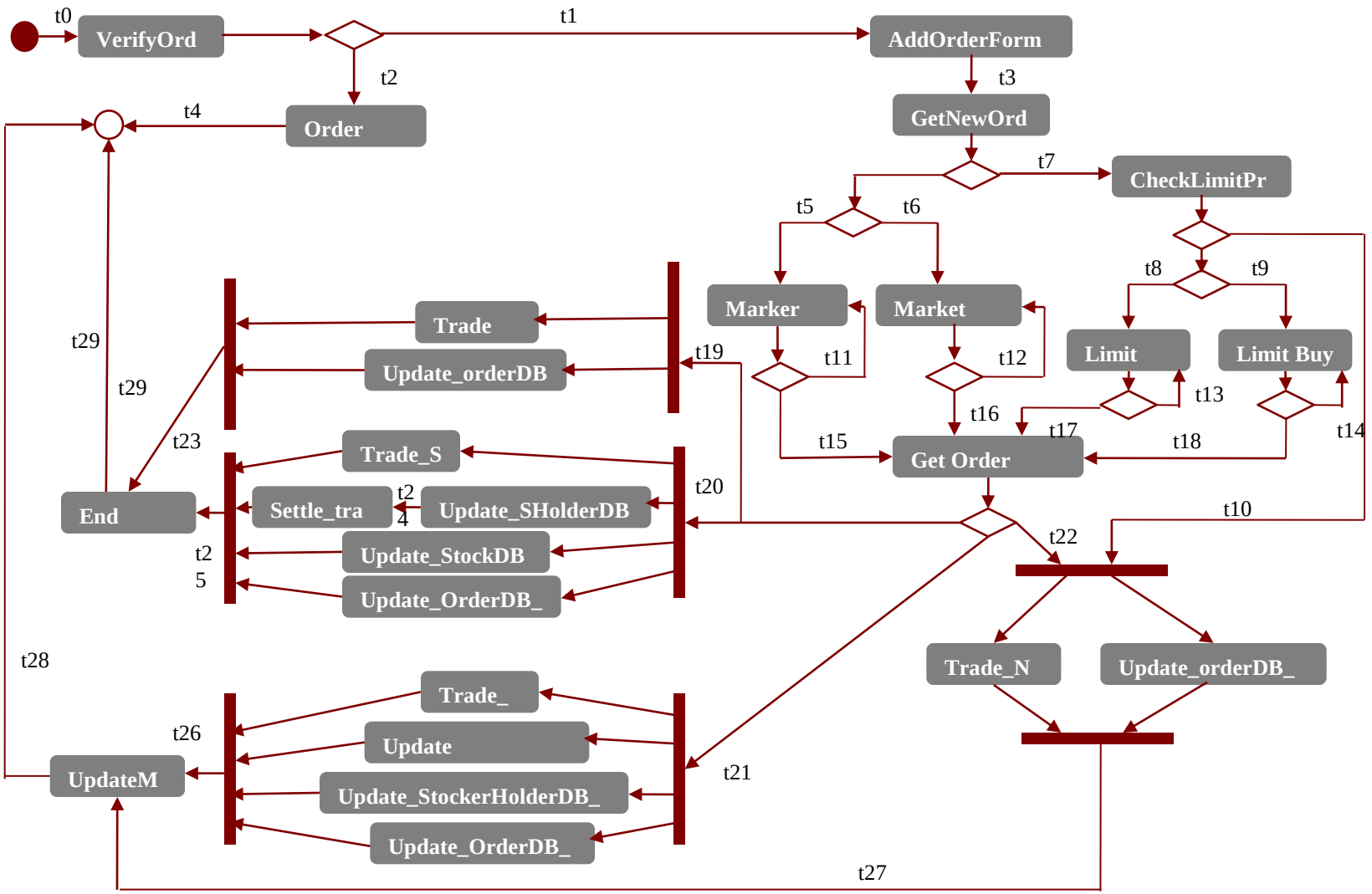




Test generation time is significantly improved

- Drastic reduction of conflict clauses
- Drastic reduction in number of implications

Case Study 2 : OSES



Case Study 2 : OSES

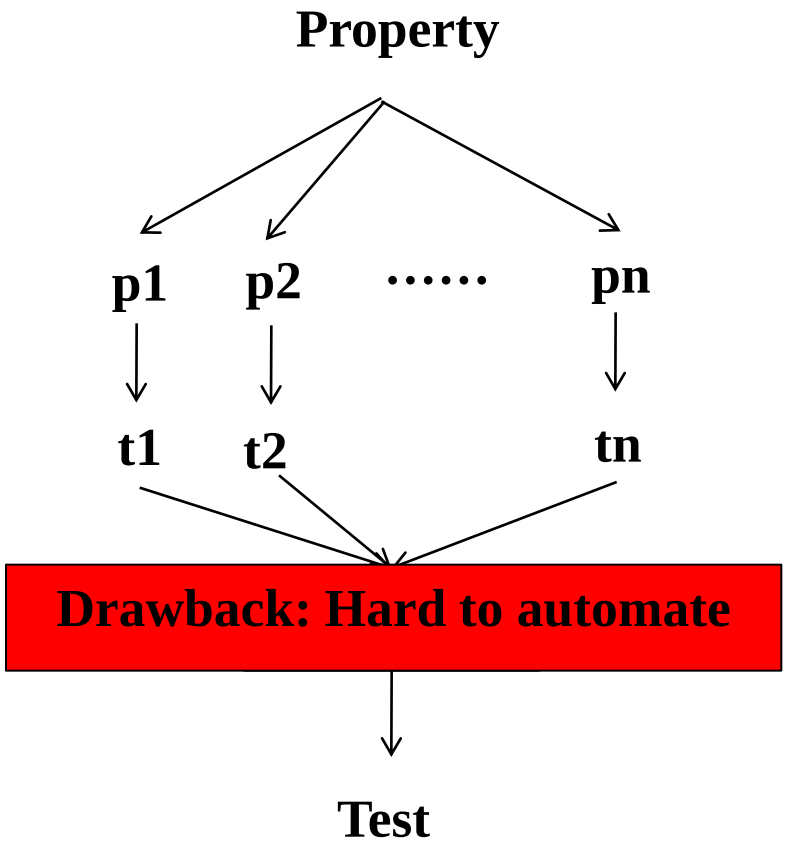
- **This case study is a on-line stock exchange system. The activity diagram consists of 27 activities, 29 transitions and 18 key paths.**

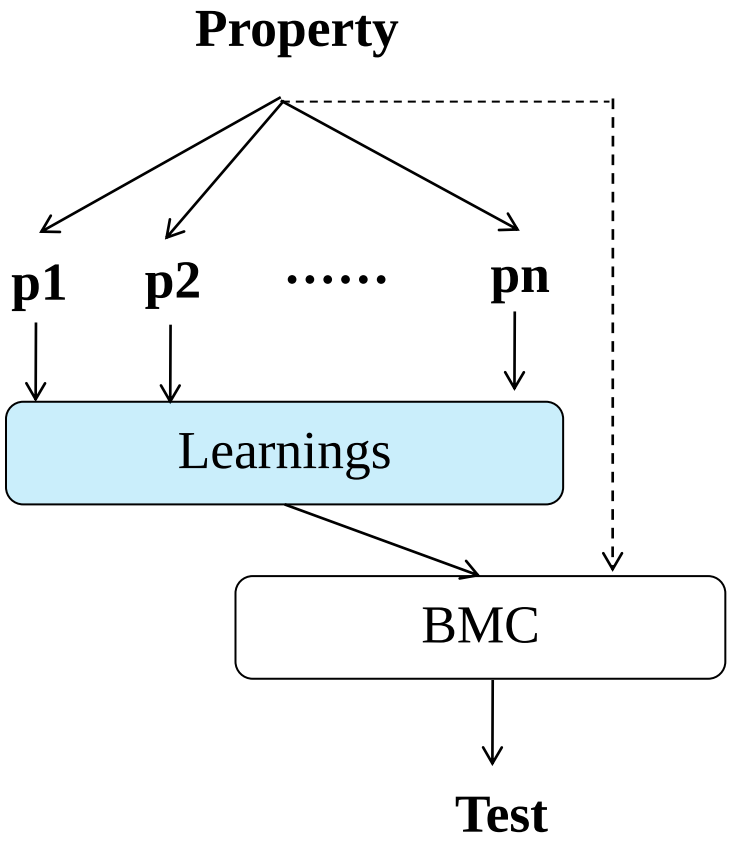
Cluster	Size	zChaff	Clustering	Speedup (over zChaff)	Decision Ordering	Speedup (over Clustering)
C1	3	1.18	2.18	**0.54**	0.70	**3.11**
C2	4	14.53	9.53	**1.52**	0.78	**12.22**
C3	8	375.91	170.06	**2.21**	36.19	**4.70**
C4	4	12.98	8.33	**1.56**	1.24	**6.72**
C5	4	7.13	16.88	**0.42**	1.02	**16.55**
C6	8	720.13	474.68	**1.52**	28.60	**16.60**
C7	4	10.80	24.55	**0.44**	1.95	**12.59**
C8	8	656.95	321.14	**2.05**	77.65	**4.14**
C9	8	248.17	82.42	**3.01**	37.93	**2.17**
Average	**-**	**227.53**	**123.21**	**1.85**	**20.67**	**5.97**

Outline

- Motivation
- **Research Contributions**
  - Automatic Validation of SoC Specifications
    - Test Generation using Model Checking
    - Basic Idea of Our Approach
  - Efficient Test Generation using Learning Methods
    - Novel property clustering approaches
    - Decision ordering based learning techniques
    - Property decomposition approaches
  - Automated Reuse using Validation Refinement Techniques
- Conclusion

Property Decomposition Techniques

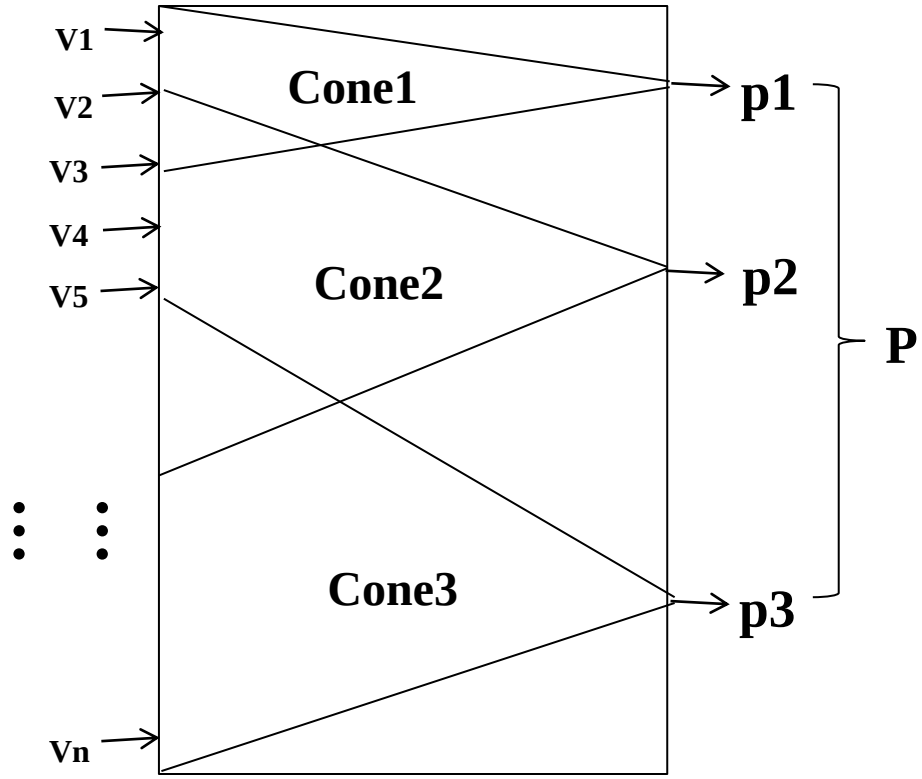




Koo et al. Functional Test Generation using Property Decomposition Techniques. ACM *TECS, 2009*

M. Chen and P. Mishra. Efficient Test Generation using Property Decomposition and Learning Techniques. *DATE, 2011*

Spatial Decomposition

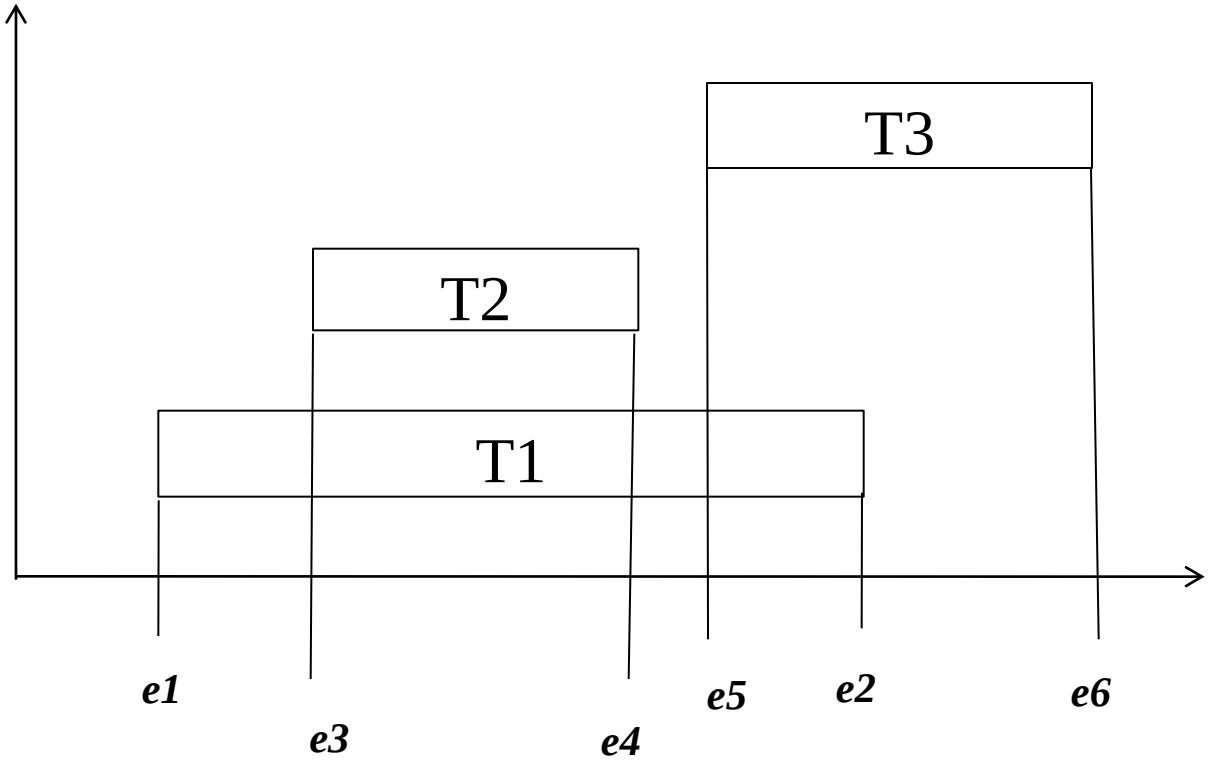


COI(p1) < COI(p2) < COI(p3) <COI(P)

Time(p1) < Time(p2) < Time(p3) <Time(P)

Learning from P1 can reduce the Time(P) ?

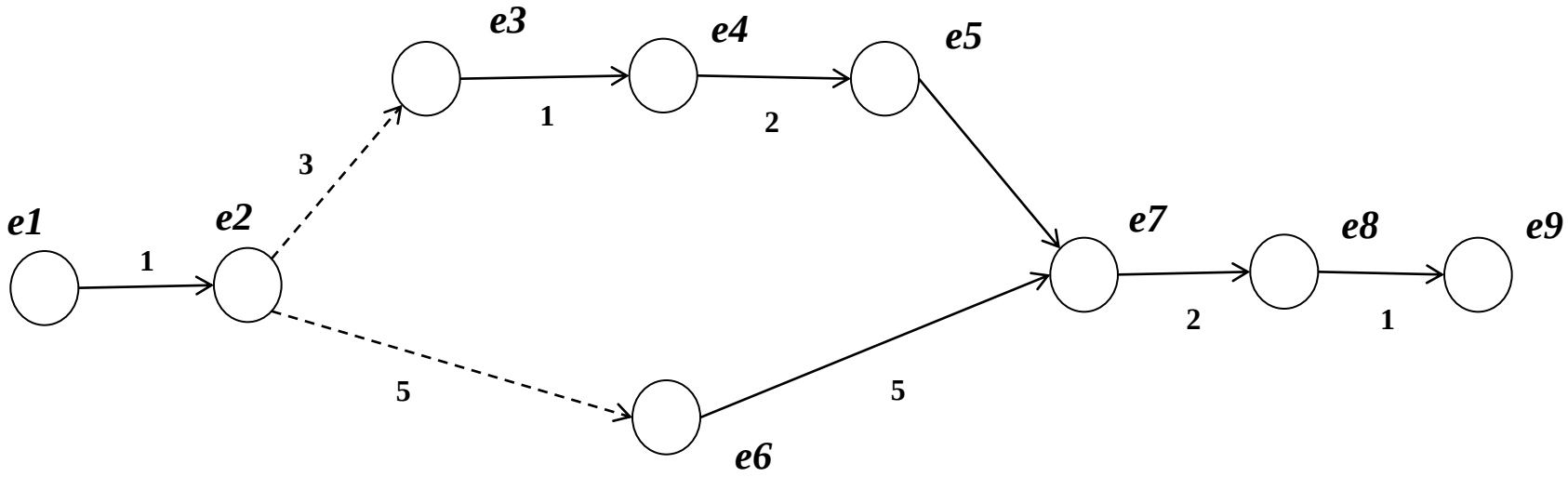
Temporal Decomposition



Cause effect relation: **e1→e2 e3→e4 e5→e6**

Happen before relation: **e1<e3<e4 <e5<e2<e6**

Temporal Decomposition



$$!F(e1) \rightarrow !F(e3) \rightarrow !F(e7) \rightarrow !F(e9)$$

Case Study 1: MIPS Processor

- We generated 6 properties based on interaction faults on various function unit (ALU, DIV, FADD and MUL), which cannot handled by temporal decomposition.

Property (test)	zChaff (sec)	Num. of Clusters	Num. of Sub-props	Spatial (sec)	Speedup
P1	127.52	3	**2**	49.41	**2.58**
P2	49.24	3	**2**	15.73	**3.13**
P3	9.18	2	**1**	4.99	**1.84**
P4	13.78	2	**1**	7.28	**1.89**
P5	31.63	3	**2**	12.74	**2.48**
P6	120.72	3	**2**	54.21	**2.23**

Speedup: 1.84-3.13 times

Case Study 2: OSES

- This case study is a on-line stock exchange system. The activity diagram consists of 27 activities, 29 transitions and 18 key paths.

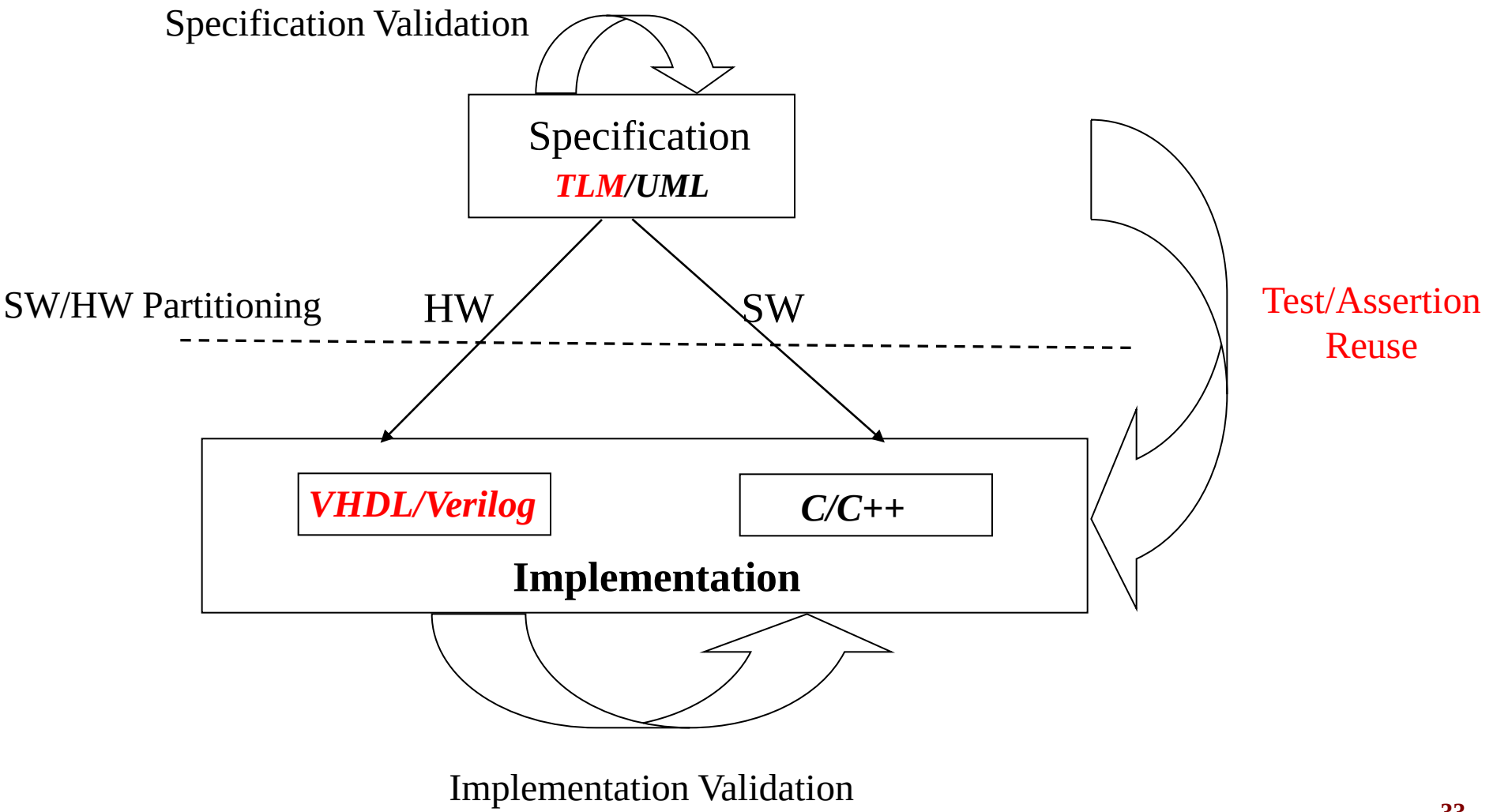
Property	zChaff (sec)	Bound	Num. of Sub-properties	Temporal (sec)	Speedup
P1	25.99	8	3	0.78	**33.32**
P2	48.99	10	4	2.69	**18.21**
P3	39.67	11	5	3.45	**11.50**
P4	247.26	11	5	22.46	**11.01**
P5	160.73	11	5	15.68	**10.25**
P6	97.54	11	4	1.56	**62.53**
P7	31.39	10	4	12.31	**2.55**
P8	161.74	11	4	12.62	**12.82**
P9	142.91	10	4	17.57	**8.13**
P10	33.77	10	4	1.76	**19.19**

Speedup: 3-63 times

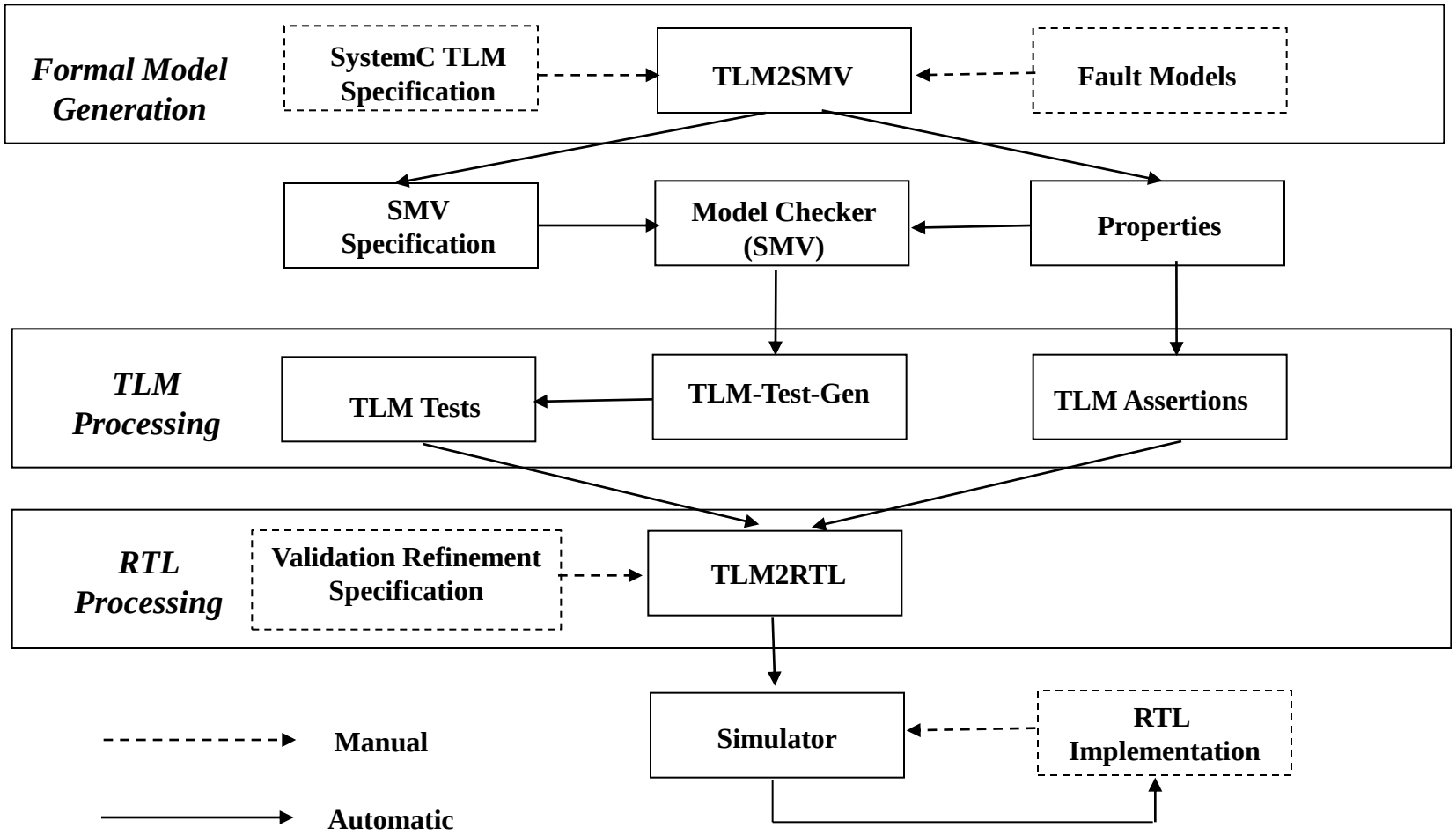
Outline

- Motivation
- **Research Contributions**
  - Automatic Validation of SoC Specifications
    - Test Generation using Model Checking
    - Basic Idea of Our Approach
  - Efficient Test Generation using Learning Methods
    - Novel property clustering approaches
    - Decision ordering based learning techniques
    - Property decomposition approaches
  - Automated Reuse using Validation Refinement Techniques
- Conclusion

SoC Design and Validation Flow



Overview of Our Prototype Tool



M. Chen, P. Mishra and D. Kalita. Automaitc RTL Test Generation from SystemC TLM Specifications. *ACM TECS, 2012*

Transformation of a TLM Test to RTL Test

Test Refiment Specification for Router

-- Port definition

input.data (10) [7:0] data;

-- TLM and RTL name binding

bit[7:0] head =
{packet_data.payload_size[7:2],
packet_data.to_chan[1:0]};

--Timing sequence

head => data;

TLM Testcase

```
p->to_chan=1;
p->payload_sz=2;
p->payload[0]=1;
p->payload[1]=2;
p->parity=10;
```

Data Composition

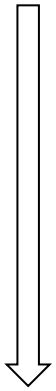
RTL Testcase

```
read_enb_0 = 0;
read_enb_1 = 0;
read_enb_2 = 0;                 Initialization
packet_valid = 0;

          reset = 0;
#5   reset = 1;                 Reset Sequence
#20 reset = 0;

#5   packet_valid = 1;          Use of half clock
     data = 8'b00001001;        Name Transformation
#10 data = 8'b00000001;
#10 data = 8'b00000010;
#10 packet_valid = 0;
    data= 8'b00001010;

#10  read_enb_1=1;
#40  read_enb_1=0;              Use of four clocks
$finish;
```

Transformation of a TLM Assertion to RTL Assertion

TLM Assertion :

Cover (tmp_packet.to_chan == 1);



Clock Expression

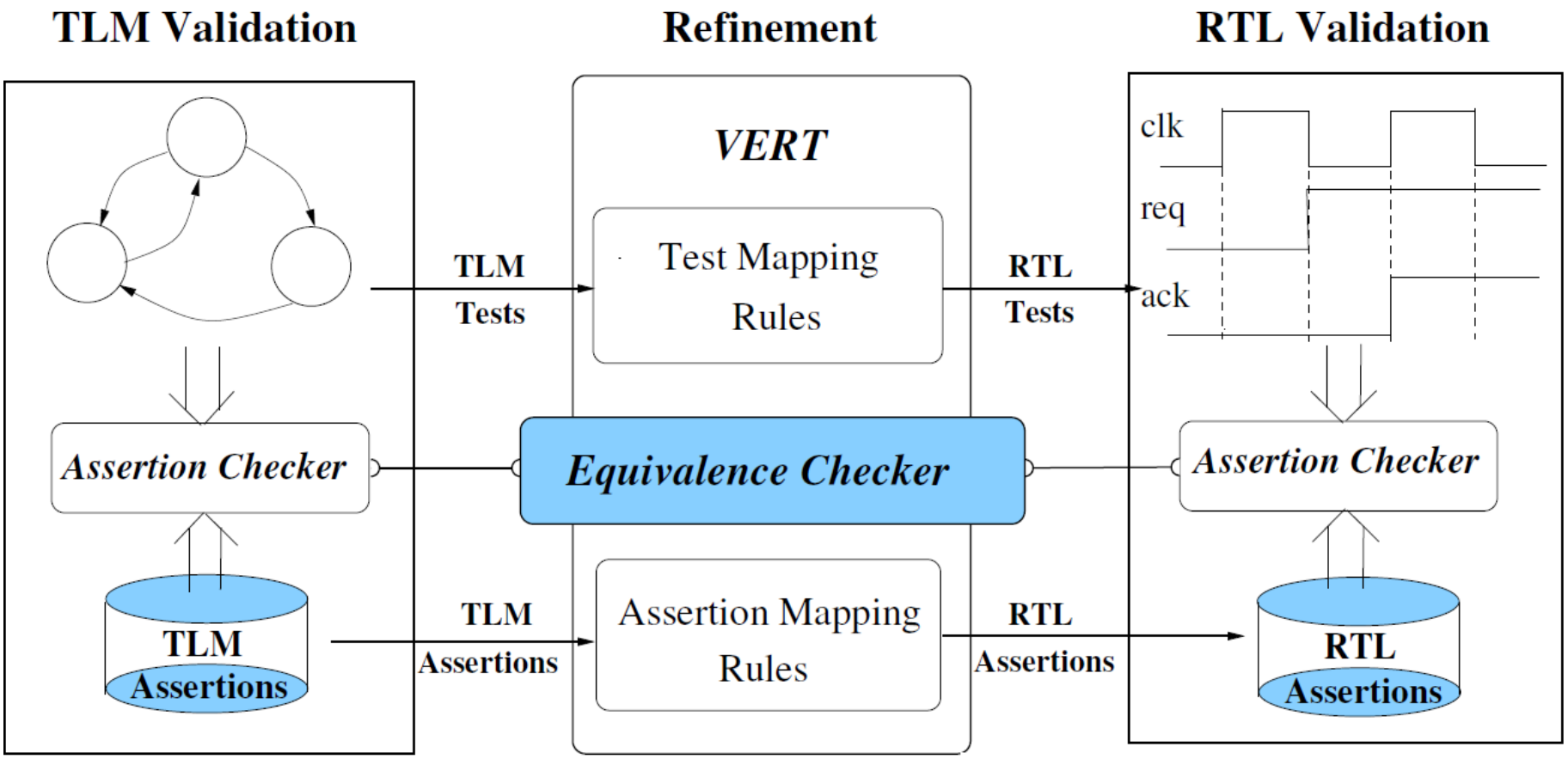
Control Signals

Name mapping

RTL Assertion : Cover Property

(@(posedge clock) ($rose(write_enb[1])) && (data_o_fsm == 2'd1));

Assertion-based Equivalence Checking



M. Chen and P. Mishra. Assertion-Based Functional Consistency Checking between TLM and RTL Models. *Intenational Conf. on LSI Design, 2013.*

Validation Refinement Case Studies

❑ An industry router example

- ❖ Generate 108 TLM tests, and 108 RTL corresponding tests
- ❖ RTL implementation Coverage

Source	Condition	FSM State(Transition)	Toggle	Path
99.5%	76.6%	100% (100%)	76.6%	73.6%

- ❖ Found 2 fatal errors in the RTL implementation.
- ❖ Assertion-based equivalent.

❑ Alpha AXP Processor

- ❖ Generate 212 TLM tests for transaction flow and data.
- ❖ After removing redundant tests: 112 TLM tests → 112 RTL tests
- ❖ The number of final required tests depend on the length of each test

Source	Condition	FSM	Toggle	Path
98.9%	97.0%	NA	70.2%	86.3%

- ❖ Assertion-based equivalent.

Conclusion

- Validation is a major bottleneck in SoC design methodology
- Our research focuses on three important challenges
  1. How tests can be automatically generated?
     - **Formal modeling of SoC specifications**
     - **Coverage-directed test generation techniques**
  2. How to reduce overall validation effort?
     - **Novel property clustering methods**
     - **Efficient decision ordering approaches**
     - **Design and Property decomposition techniques**
  3. How to reuse validation effort between abstraction levels?
     - **Rule-based test translation methods**
     - **Assertion-based equivalence checking**
- Successfully applied on both hardware and software designs
  - **Drastical reduction in overall validation effort (10X~100X)**

Thank you !