

IBM

# RuleBase Parallel Edition: A Massively Parallel Platform for Formal Verification

Rachel Tzoref  
November, 21<sup>st</sup> 2004





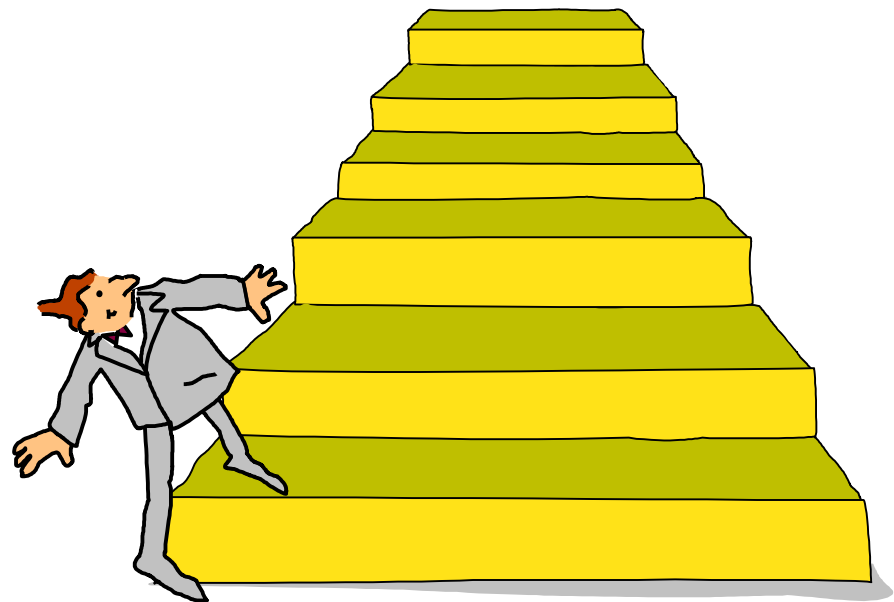
## Agenda

- ◆ Motivation
- ◆ RuleBase PE Architecture
  - ◆ GUI
  - ◆ Dispatcher
  - ◆ Backend
  - ◆ Verification Engines
- ◆ Experience
- ◆ Concluding Remarks



## Motivation: Challenges in Formal Verification

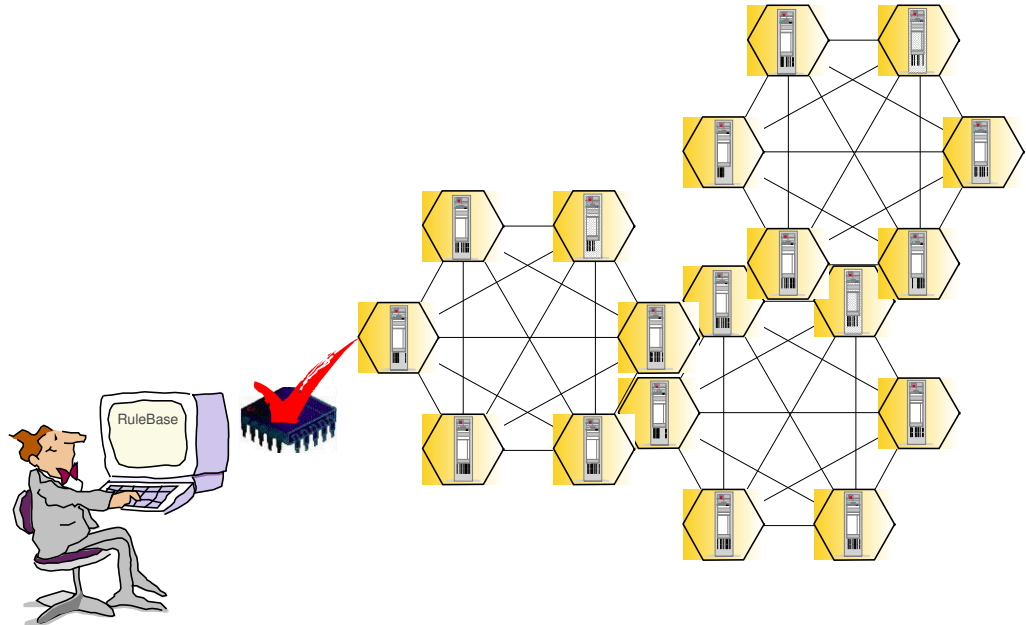
- ◆ Current formal verification techniques are lagging behind growing design complexity
- ◆ State-space explosion limits the size of the units which formal verification can check
- ◆ Implications
  - ◆ Specification effort
  - ◆ Verification engineers time is mostly spent waiting for answers.
  - ◆ Uncertainty in selecting verification algorithms for a given assertion





## RuleBase Parallel Edition – Marry FV with Parallel Computing

- ❖ Reduce runtime by distributing verification tasks over a large number of CPUs
- ❖ CPUs are relatively cheap comparing to verification engineers time
- ❖ Bonus: larger verification tasks can be handled





## The RuleBase Parallel Edition Platform: Two Tiers of Parallel FV

### Coarse-Grain Parallel FV

- ◆ Checking multiple verification units simultaneously
- ◆ Engines share runtime information

### Fine-Grain Parallel FV

- ◆ Decomposition of a large verification task into smaller, tractable subtasks, which can be run simultaneously on different CPUs



## User's Perspective:

3. Click Run...

The screenshot shows the IBM Odyssey software interface. The main window is titled "Odyssey - andand (Inx-taz) - [Rule andand\_issue\_2266\_1]". The interface includes a menu bar (File, Batch, Rule, Configurations, Windows, Help), a toolbar with icons for file operations and execution, and a status bar at the bottom.

The "Rules List" pane on the left displays a table of rules:

Stat	Rule Name	Status
	test_RST	started via web
	test_RST1	started via web
	test_RST2	unknown
	test_RST3	unknown
	test_RST4	started via web
	test_RST5	unknown
	test_RST6	unknown
	one_and	
	two_and	
	three_and	
	no1_twodots_no2_1	
	no1_twodots_no2_2	done
	no1_twodots_no2_3	done
	lt_seg_fault_2099_1	unknown
	lt_seg_fault_2099_2	unknown
	Seg_fault_2265	unknown
	andand_issue_2266_1	started via web
	andand_issue_2266_2	unknown
	shoham	started via web
	issue_2157	unknown
	dana	unknown
	OR_AND	started via web
	minimized_or_and	unknown

A yellow cloud with the text "PSL" is overlaid on the "andand\_issue\_2266\_1" rule. A red arrow points from the "Run" button in the toolbar to the "andand\_issue\_2266\_1" rule.

The "Rule andand\_issue\_2266\_1" pane in the center shows a tree view of the rule's components: Run progre, Load log, Rule details, Formulas, Traces, Errors/Warr, General set, Debugging, and Engines. The "Engines" component is selected, and a table of engine configurations is displayed:

	1	2	3	4	5	6	7	8
Discovery *								
Classical *								
Unfolding *								
SAT								
SmartLoc								
Beelzebub								

The "Configuration Window" on the right shows a tree view of the configuration settings: All engines, General, Engines, Discovery, Classical, Unfolding, SAT, SmartLoc, and Beelzebub. The "Engines" section is expanded, and the "Beelzebub" engine is selected. Below the tree view, there are sections for "Reductions", "Model Checking Algorithm", and "Trace production".

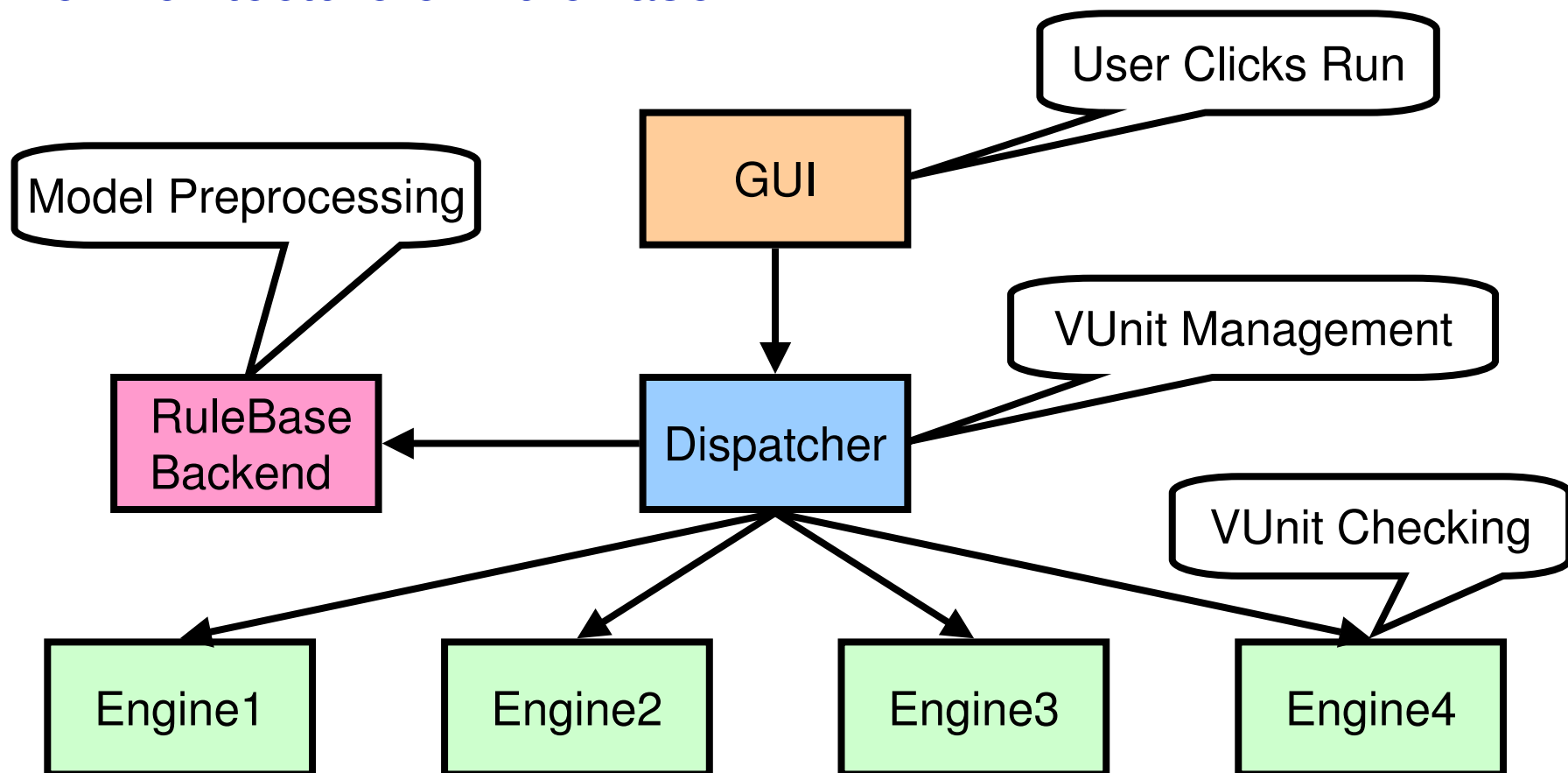
The "Error log" pane at the bottom shows the following text:

```
Refresh ended successfully
IBM Odyssey (based on RuleBase Technology) Version 1.10, build 190. Built on Oct 15 2003, 10:52:22
(C) Copyright IBM 2003
```

Red arrows indicate the user's perspective: "1. Choose vunit" points to the "andand\_issue\_2266\_1" rule, "2. Choose engines" points to the "Engines" section in the Rule details pane, and "3. Click Run..." points to the "Run" button in the toolbar.



## The Architecture of RuleBase PE





## Dispatcher – The Verification Unit Manager

Dispatcher





## The Dispatcher: Job Description of VUnit Manager

- ◇ Manages the checking of a specific verification unit
- ◇ Starts RuleBase backend (the model preprocessor)
- ◇ Starts engines when backend is done
- ◇ Communicates with engines and GUI
- ◇ Continues running until vunit verification is finished
- ◇ Can distribute engines across remote machines (possibly by using Load-Balancers)




## RuleBase Backend – Preprocessing the Model

RuleBase  
Backend



## The RuleBase Backend

- ◆ Parses the PSL assertions and input constraints
- ◆ Transforms PSL expressions to invariants with state machines
  - ◆ Highly optimized algorithms, based on IBM's PSL Technology 
- ◆ Merges design and input constraints into a unified state machine
- ◆ Executes numerous **reductions** and abstractions on the merged model



## Reductions – Reduce Model Size While Retaining Behavior

### A Key Role of the RuleBase PE Backend

- ◆ Remove logic that does not affect the assertions
  - ◆ E.g. cone of influence
- ◆ Reduce signals to constants
  - ◆ E.g. constant propagation
- ◆ Exploit similarity of parts of logic
  - ◆ E.g. identify equivalences
- ◆ Other reductions
  
- ◆ All reductions are safe
  - ◆ Preserve assertions validity
  - ◆ Restore reduced signals values to show a complete trace



## The Verification Engines

Engine1

Engine2

Engine3

Engine4



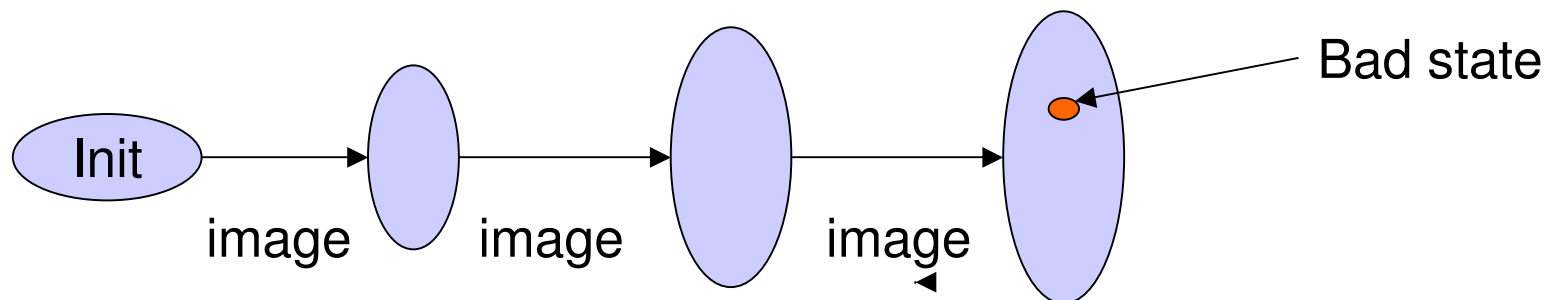
## Verification Engines – Multiple Types, Parallelization-Driven

- ◆ Model Checking (MC) vs. Bounded Model Checking (BMC)
  - ◆ In MC, assertions are evaluated on infinite paths (the entire model)
  - ◆ In BMC assertions are evaluated on finite paths (a truncated model)
- ◆ Verification vs. Falsification
  - ◆ Some engines are better for proving assertions
  - ◆ Some are better for falsifying assertions
  - ◆ Some equally handle true and false assertions
- ◆ Safety vs. Liveness
  - ◆ Safety – something bad never happens
  - ◆ Liveness – something good will finally happen
  - ◆ Some engines can only handle safety assertions
- ◆ Checking multiple assertions vs. checking a single assertion



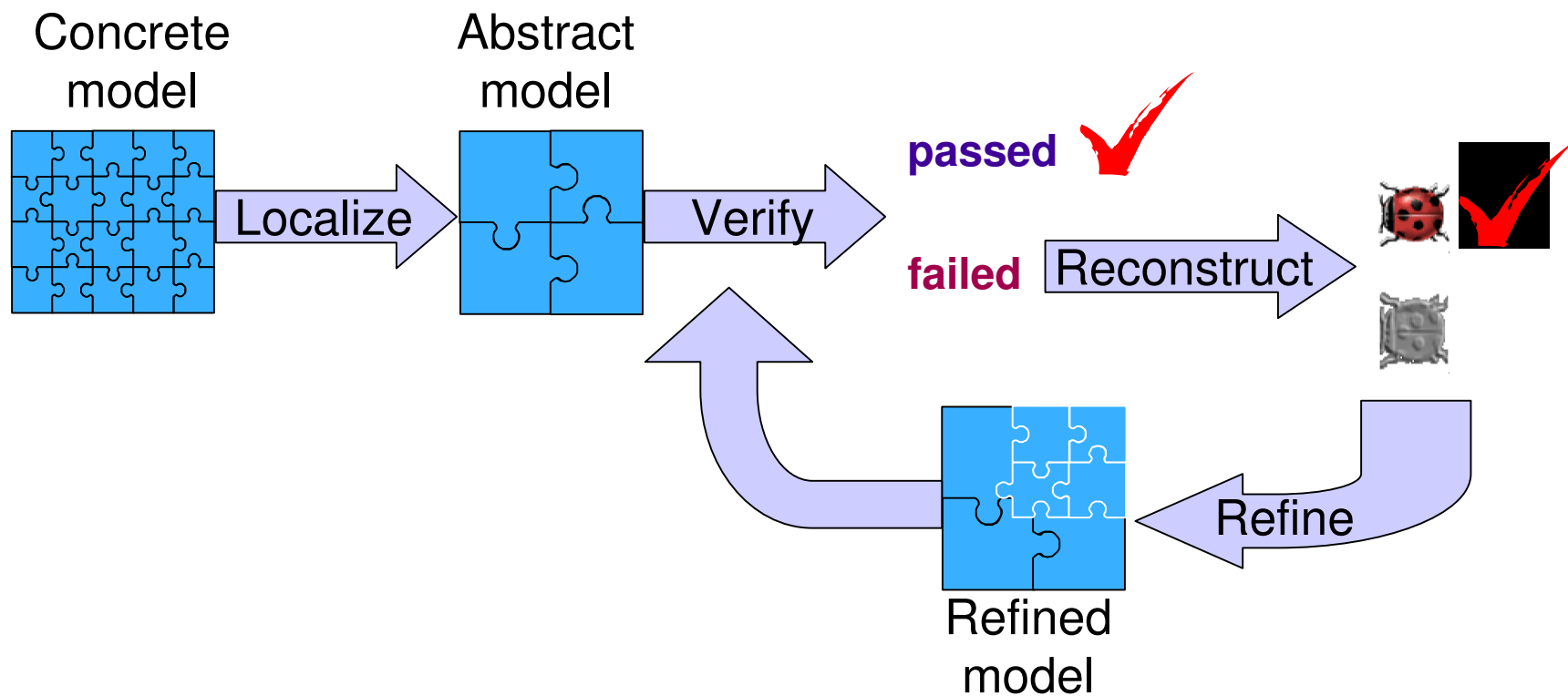
## Symbolic Engines in RuleBase PE: Partial Preview

- ◆ **Classical** – performs “text-book” model checking using BDDs.
  - ◆ All fixpoint calculations are done by performing pre-image calculations (backwards steps)
- ◆ The IBM “Discovery” symbolic engine – first performs **reachability analysis** and then uses it to simplify classical model-checking.
  - ◆ Reachability analysis – BFS search from initial states until fixpoint – done by using image calculations (forward steps)
  - ◆ Reachability is also used to check safety assertions “on-the-fly”





## Abstraction Refinement in RuleBase PE: The SmartLoc Engine

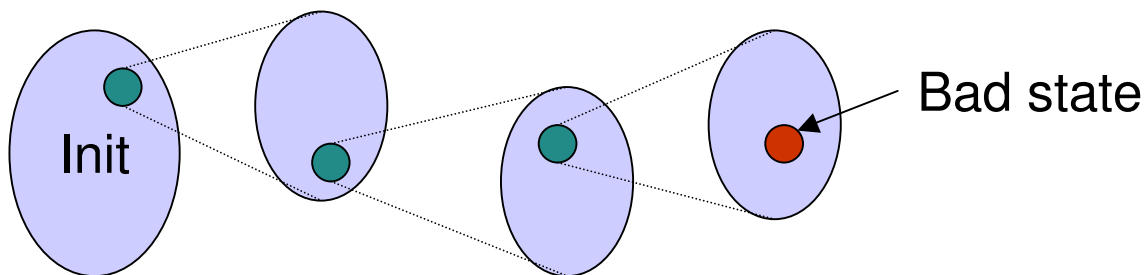






## Adaptive Search in RuleBase PE: The Beelzebub Engine

- ◆ Performs **partial guided forward search** towards “bad” states
- ◆ Computes image of small sets – unlikely to explode
- ◆ Clever heuristics carefully determine which small sets to focus upon

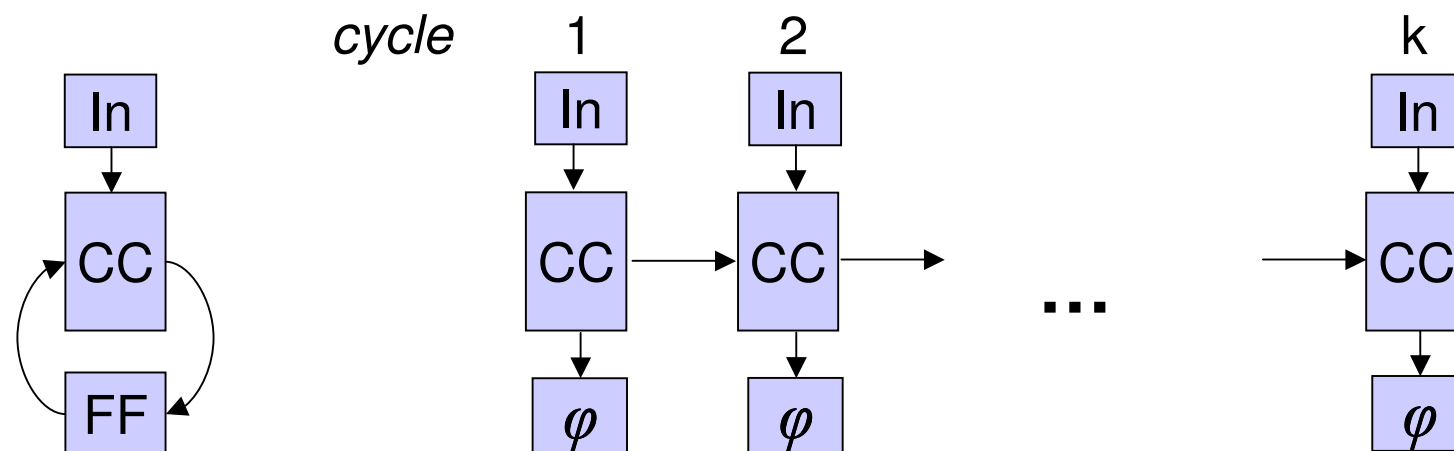




## Circuit Unfolding – Parallel Symbolic Simulation Engine

### Parallelized Version of CHARME03 Algorithm

- ◇ Unfolds the circuit into  $k$  combinational circuits, one for each cycle
- ◇ FFs become wires – FF inputs of cycle  $i$  are connected to FF outputs of cycle  $i+1$
- ◇ The assertions are formulated as part of the combinational circuit
- ◇ Use **BDD-Based** decision procedures to prove the assertions
- ◇ Non-deterministic inputs are the variables of the BDDs





## Parallel SAT

- ◆ Performs bounded model checking by transforming the model into a propositional formula that describes all reachable paths of the model that contain a bug in cycle k

$$I(s_0) \wedge TR(s_0, s_1) \wedge TR(s_1, s_2) \wedge \dots \wedge TR(s_{k-1}, s_k) \wedge \neg \phi(s_k)$$

- ◆ The propositional formula is translated into conjunctive normal form (CNF) and is given to a SAT solver

$$(x \vee y \vee z) \wedge (\neg x \vee y) \wedge (\neg y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)$$

- ◆ If the SAT solver finds a satisfying assignment to the CNF, it is a counter example. Otherwise, the model doesn't contain a bug in cycle k

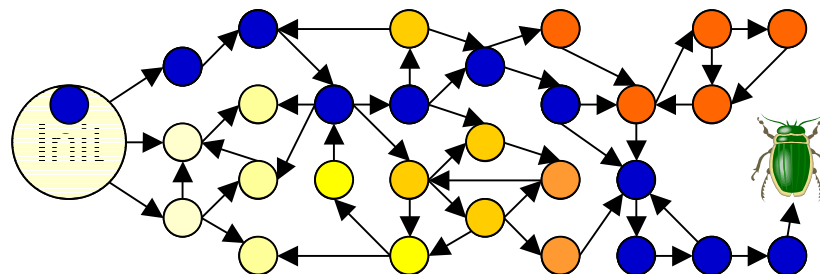
$$X = \text{False} \ \& \ Z = \text{True}$$

- ◆ Parallelized version of DPLL algorithm



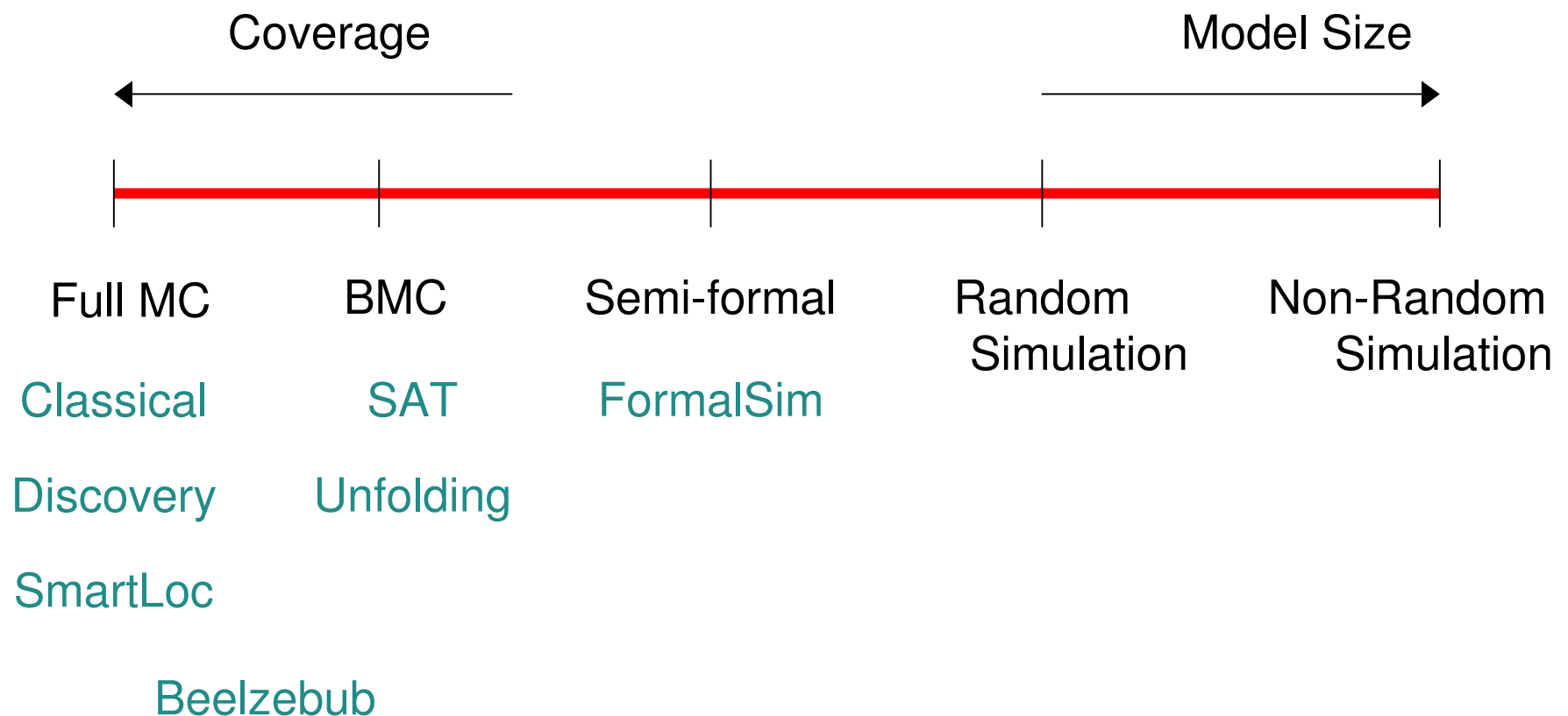
## FormalSim – Semi Formal Engine

- ◇ An explicit model checker that is oriented to find bugs
- ◇ Does not attempt to achieve full coverage of the state space
- ◇ Uses clever heuristics that guide towards the bad states
- ◇ Can address very large designs





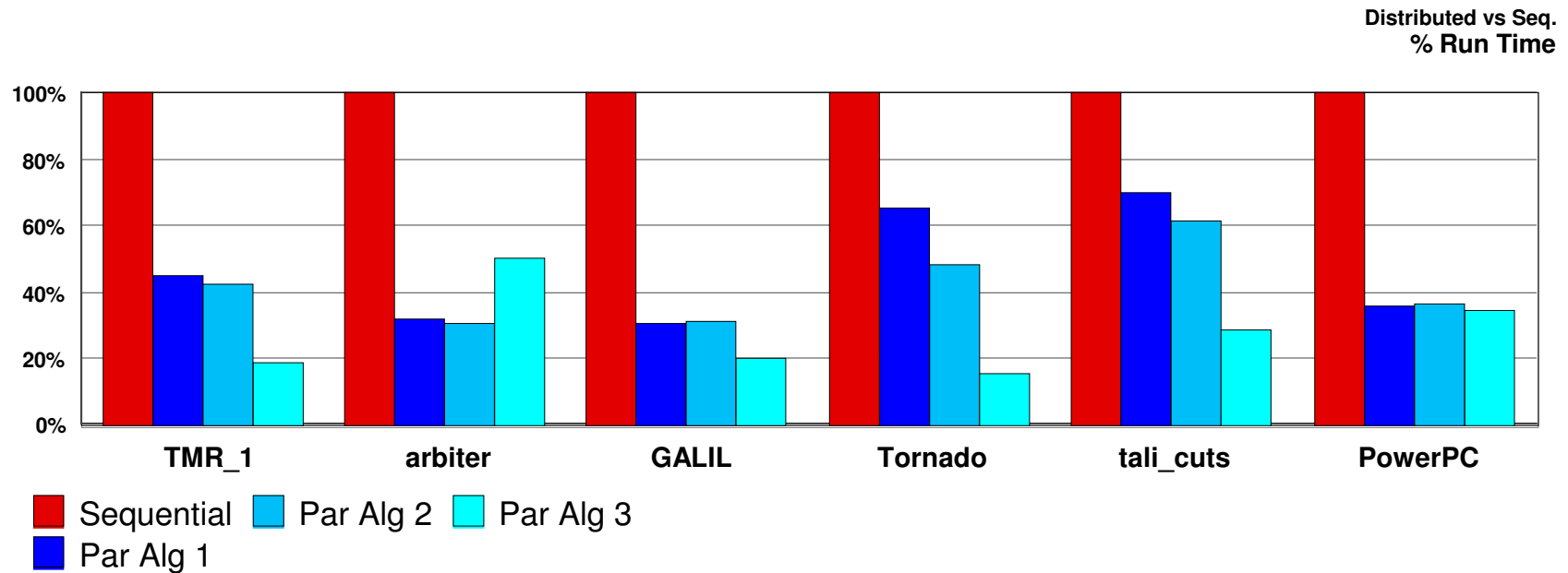
## Coverage vs. Model Size





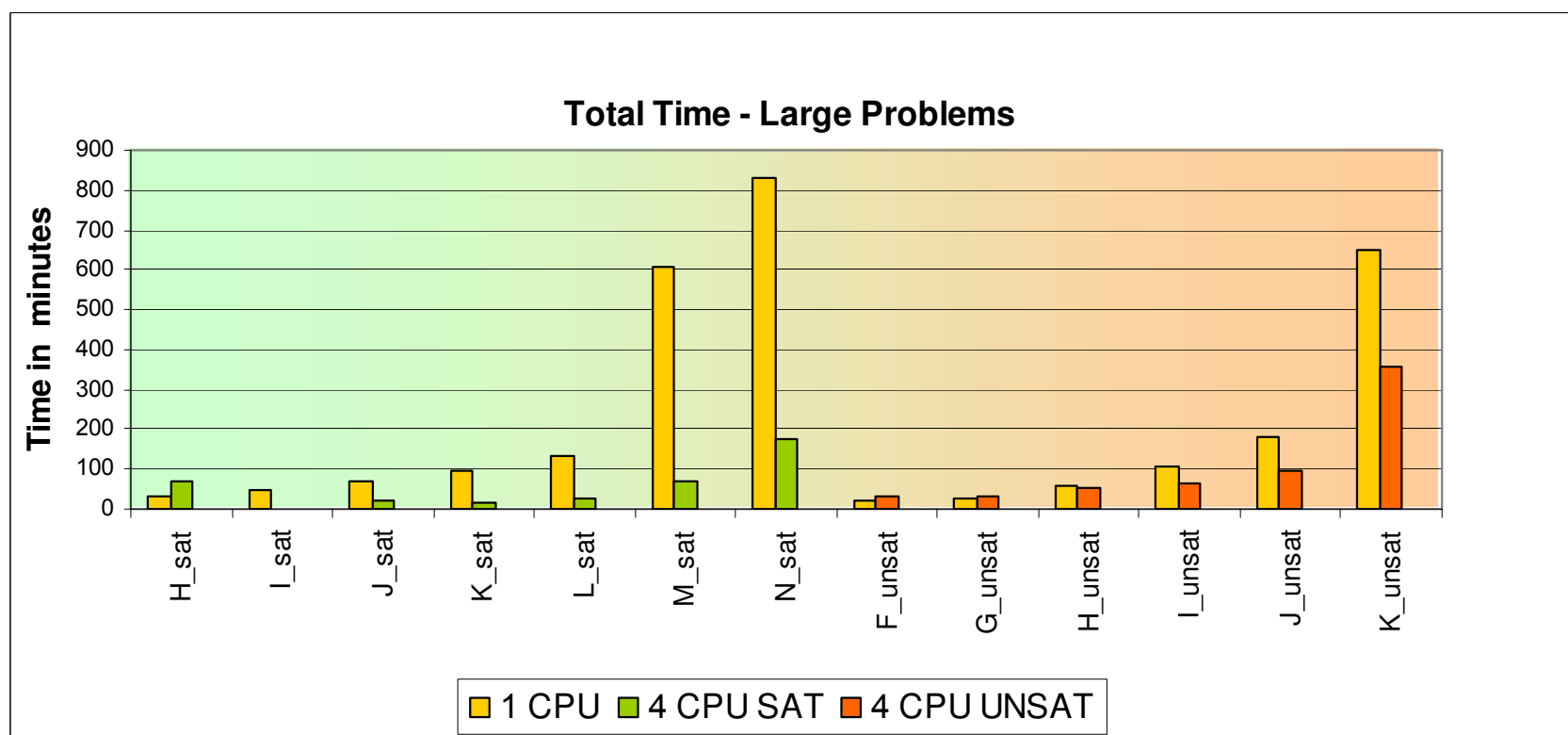
## Parallel BDD Algorithms - Results

Average improvement using 3 slaves: **27.85%**





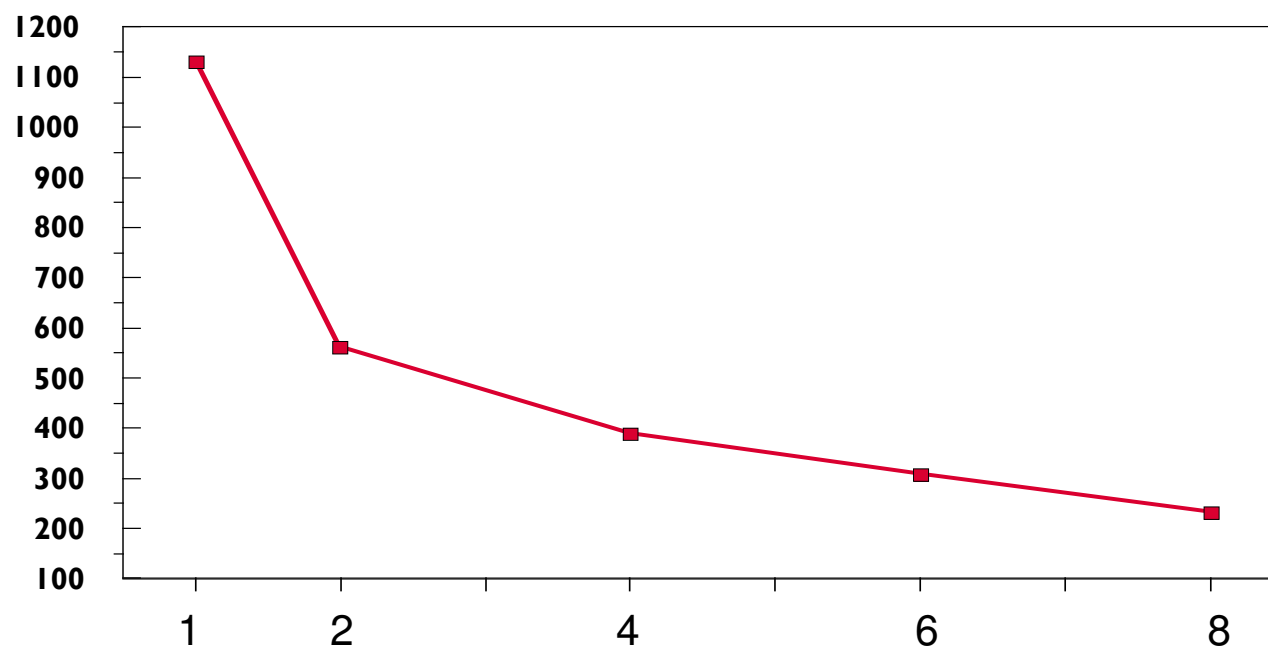
## Parallel SAT - Results





## Parallel Circuit Unfolding - Results

◇ Twice faster with 3 slaves







## Customer Feedback

- ◆ "We are very satisfied with **RuleBase Parallel Edition**; the coarse grain parallelism seems to substantially increase the verification productivity"

FV Engineer, Multinational Semiconductor Company

- ◆ "It is our experience that non formal method with three times the resources will yield lower verification quality than the one we achieve with **Rule Base PE**. Furthermore the multi-engine parallelization, gave our engineers 2x the productiveness in Formal Verification deployment. Furthermore the new algorithms added to RuleBase PE were very significant in our ability to do the verification so effectively as we did."

IBM Haifa Development Lab



# **RuleBase PE**