Formal Hardware Verification @ IBM

Pradeep Kumar Nalla
Srobona Mitra
Sudhakar R A

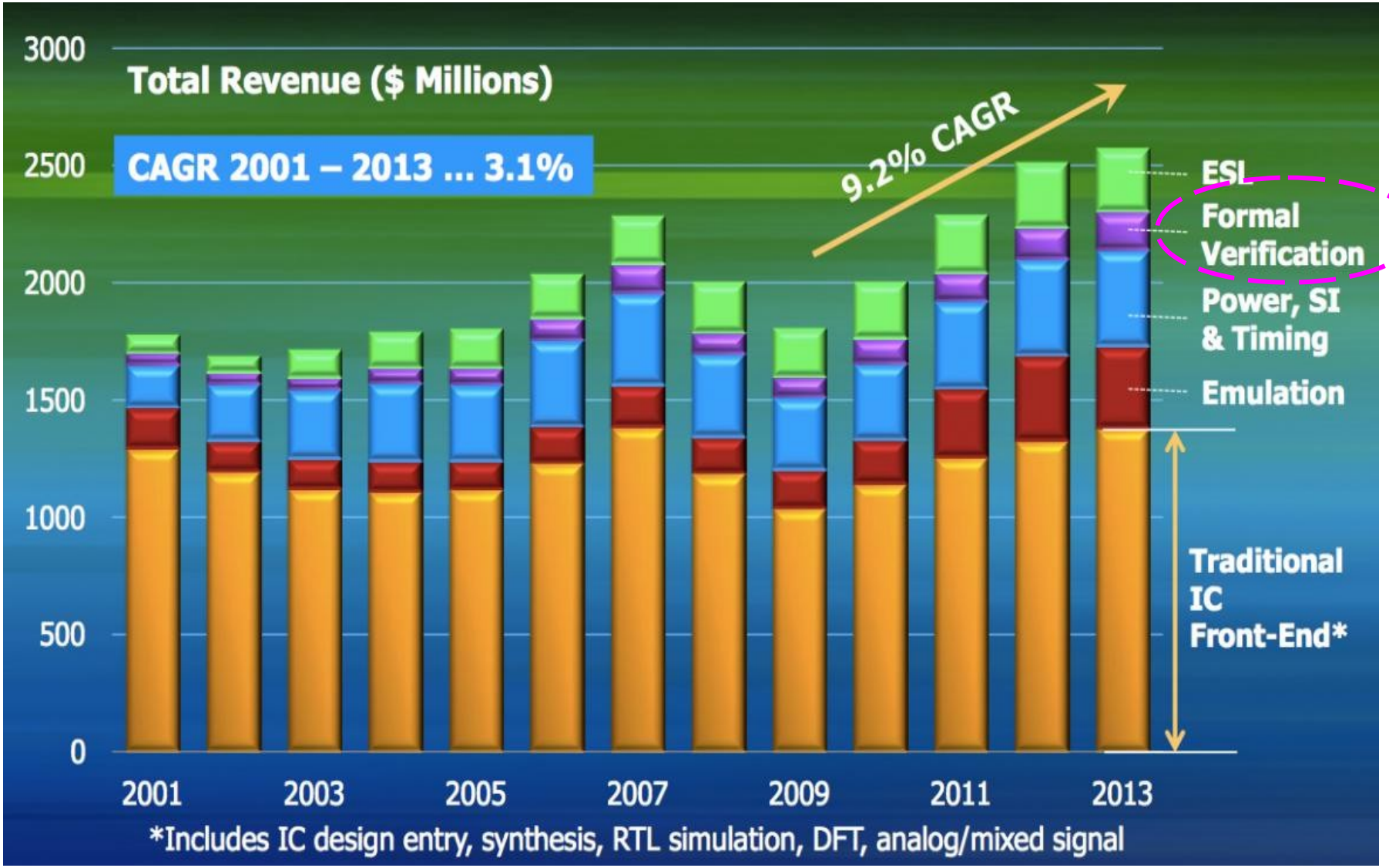
DVClub Bangalore

3rd July 2014, Bangalore, India

Agenda

- **Verification at IBM (Background and History)**
- Formal Verification using RuleBase SixthSense
- Formal Verification: Execution/Adoption to IBM designs

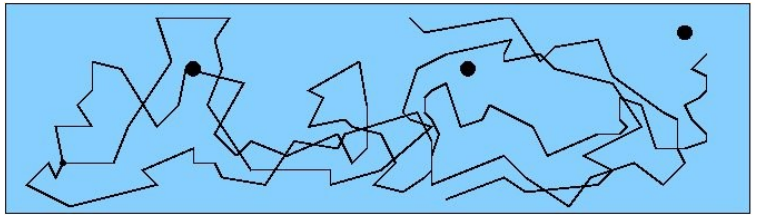
Putting things in perspective



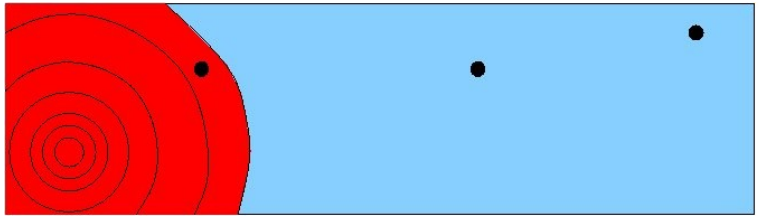
Courtesy : DAC'14 article/Mentor Graphics

Verification Techniques

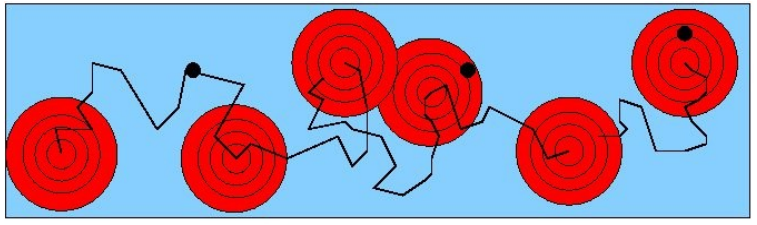
- **Simulation and Acceleration**
  - Explicit-state guided random walk
  - ✔ Scalable to **HUGE** designs
  - ✔ **Mature** methodology + tools for high coverage
  - ✘ %Coverage inherently ***very limited***
    - ✘ Misses bugs; never *complete*



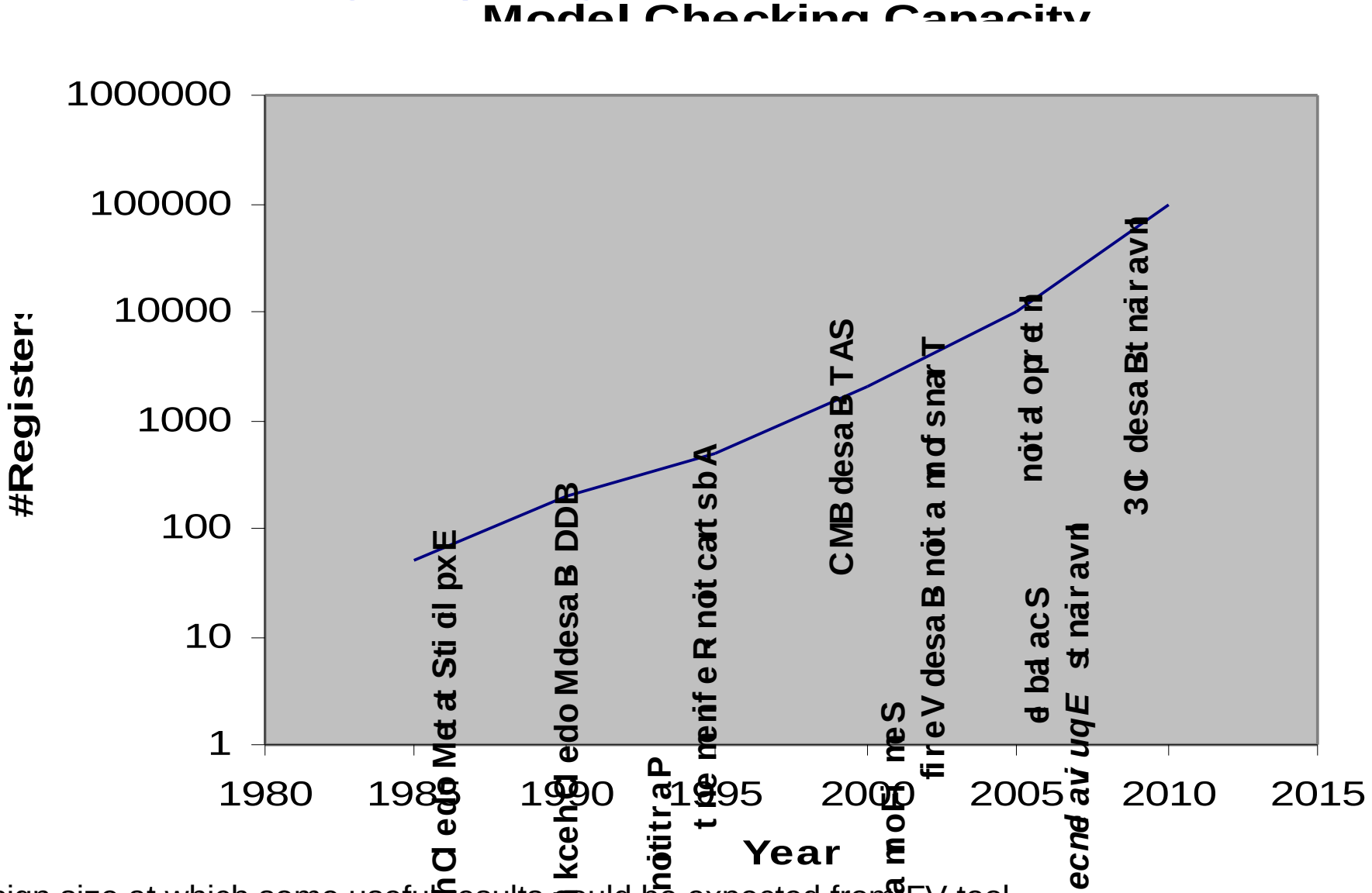
- **Formal Verification (FV)**
  - *Exhaustive* state coverage via symbolic algos
  - ✔ Yields (corner-case) bugs *or* proofs
  - ✘ Capacity-limited to moderately-sized designs



- **Semi-formal Verification (SFV)**
  - *Combine* symbolic + explicit search
  - ✔ Exposes corner-case bugs on large designs
  - ✘ Only yields ***bounded*** proofs



Model Checking Capacity vs Time



Design size at which some useful results could be expected from FV tool

Caveat: not *guaranteed* capacity; 1) some tiny problems are unsolvable! 2) includes *bounded* proofs

Very incomplete list; cumulative capacity trend leverages earlier innovations + SW engineering

Formal Verification Evolution @ IBM

Early Times

Middle Ages

Modern Era

The Future…

2000 2002 2006 2014

Applied to *small logics* (~100s of registers)
Manual Intensive w/ *dedicated resources*
Required setting up of *complex drivers*

Advent of *SFV*, Parallel, SEC
Larger logics verified; higher coverage
Same "look and feel" as simulation
SEC key to many newer methodologies

Large scale FV application
Integrated approach / DLV
Out-of-the-box methodologies
High speed, capacity toolsets

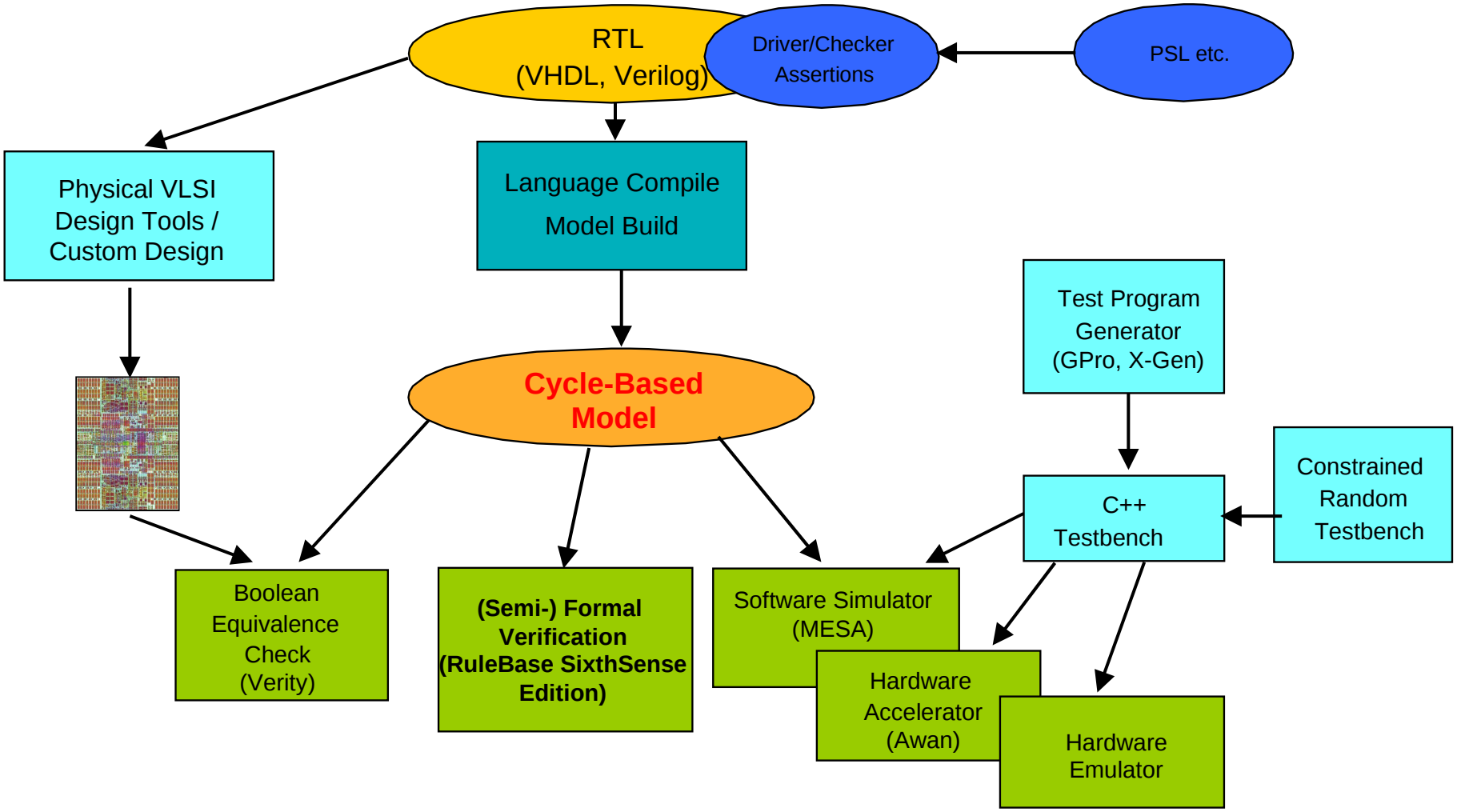
Avoid duplicate work
Reusable methodologies / IP
Automation, automation…
Stay tuned!

SFV: Semi-formal verification
SEC: Sequential Equivalence Checking
DLV: Designer-level Verification

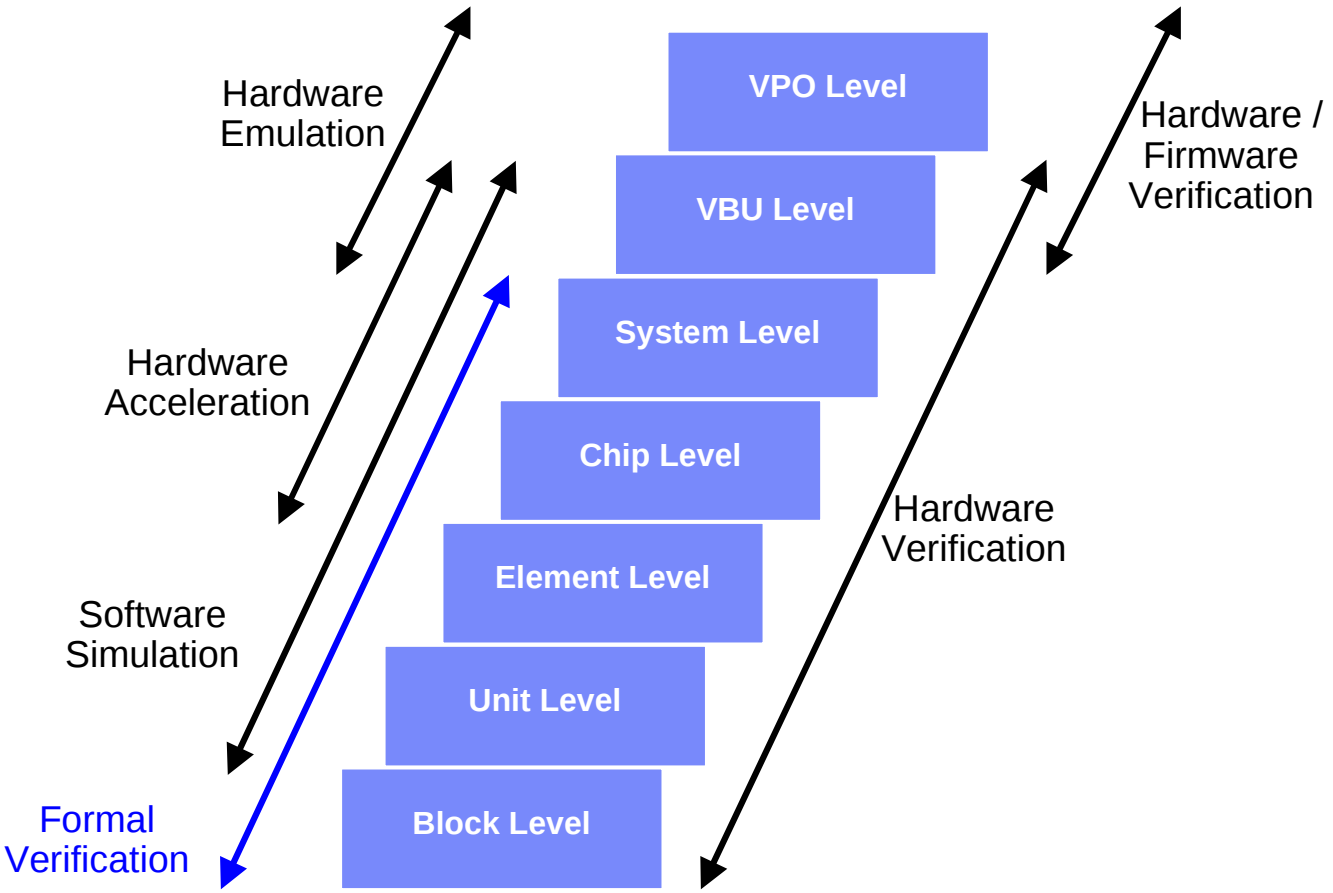
High tool capacity has enabled profound methodology impact

FV Capacity = Usability

Verification Flow at IBM



Hierarchical Verification Progression



VBU = Virtual Bring-Up (chip)
VPO = Virtual Power-On (system)

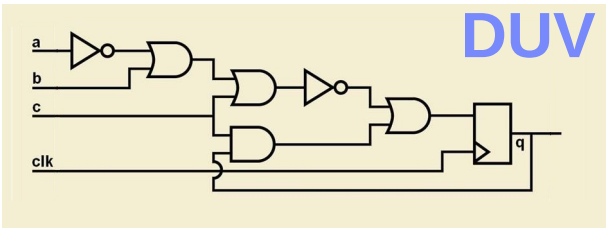
Agenda

- Verification at IBM (Background and History)
- **Formal Verification using RuleBase SixthSense**
- Formal Verification: Execution/Adoption to IBM designs

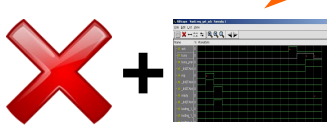
Model checking with RuleBase SixthSense Edition

Environment, Driver

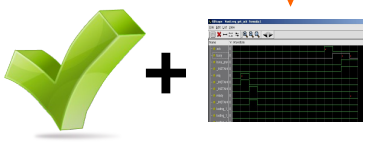
Assertions, Properties



RuleBase SixthSense



Fail
+
Counter example



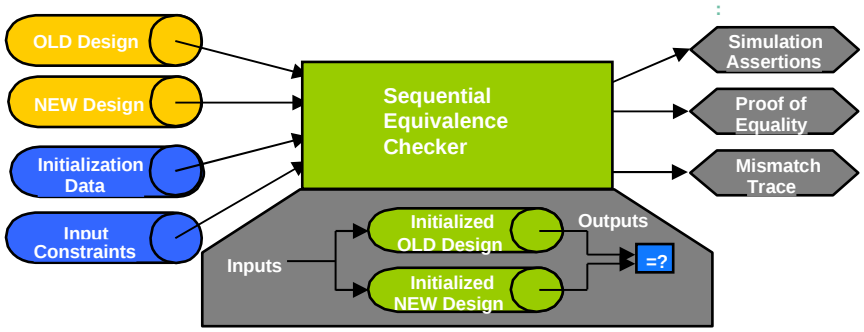
Pass
+
Witness

Pass vacuously

Bounded pass

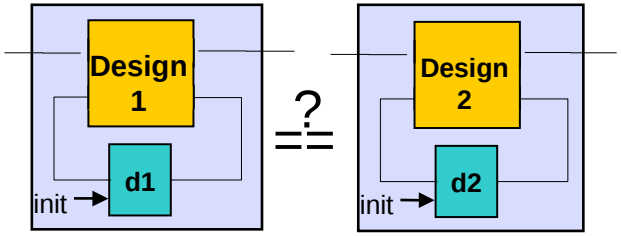
Sequential Equivalence Checking

Sequential Equivalence Checking (SEC)

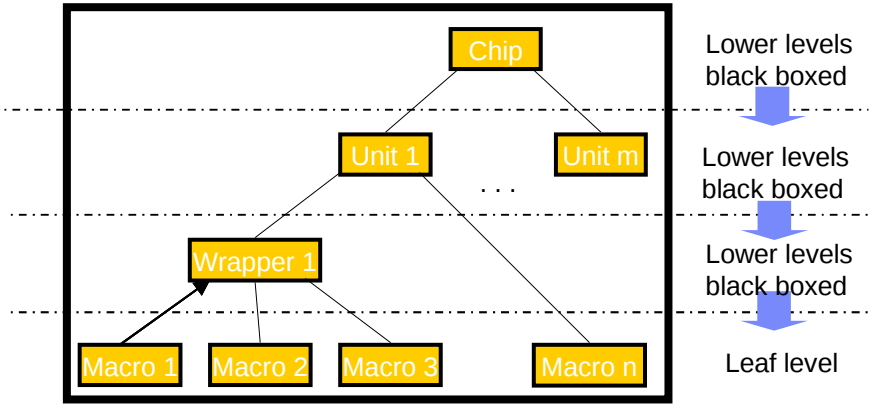


Supports arbitrary changes that preserve IO behavior
E.g., does Design1 behave identically to Design2?
1. RTL vs RTL (non-functional changes)
2. RTL vs netlist
3. Netlist vs Nestlist

Retiming, power optimization, logic minimization, …



Hierarchical application enables high scalability



Design hierarchy

Game changing application of FV

End-to-end verification of entire chips

Invaluable productivity advantage, resource savings

Unbounded proofs are critical in SEC!

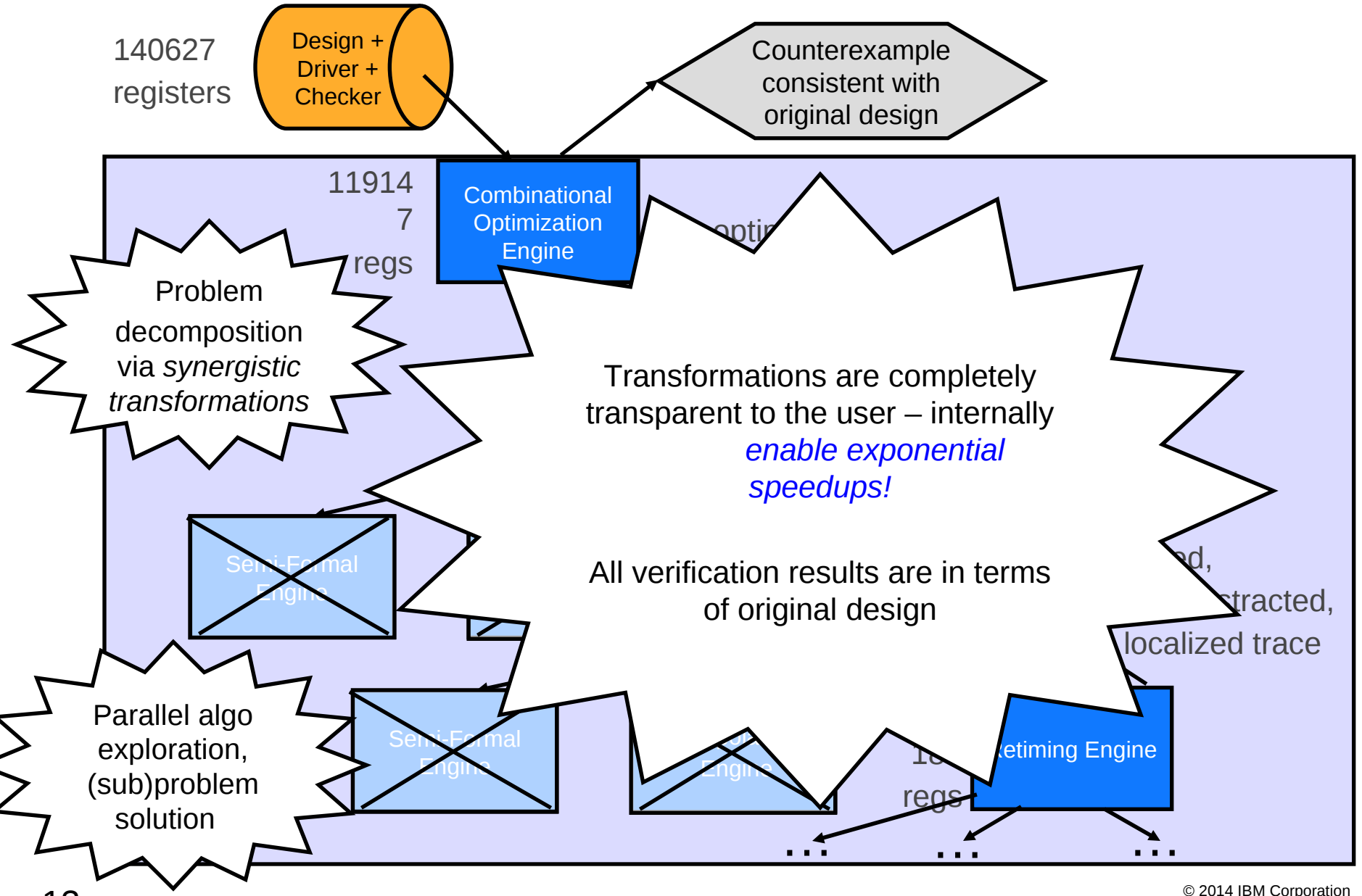
Example Engines for internal designs

- Combinational rewriting
- Sequential redundancy removal
- Min-area retiming
- Sequential rewriting
- Input reparameterization
- Localization
- Target enlargement
- State-transition folding
- Circuit quantification
- Temporal shifting + decomposition
- Isomorphic property decomposition

- Unfolding
- Speculative reduction
- Symbolic sim: SAT+BDDs
- Semi-formal search
- Random simulation
- Bit-parallel simulation
- Symbolic reachability
- Property-directed reachability
- Induction
- Interpolation
- Invariant generation
- Array abstraction

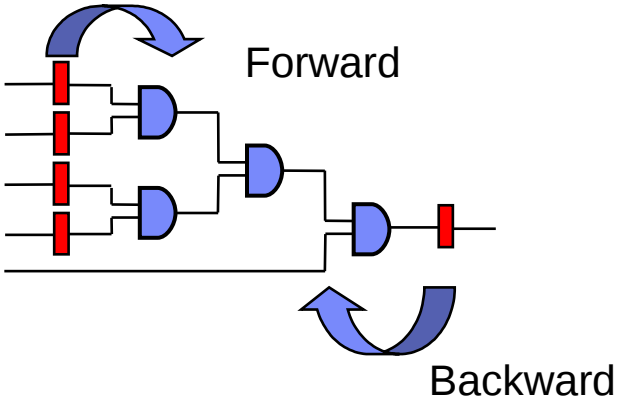
- Expert System Engine orchestrates parallel optimal engine selection
- If there is a useful verification algorithm, RuleBase SixthSense Edition likely has it!
  - *Much* innovation: necessity is the mother of invention; IBM has deep verification needs!
  - Also *much* collaboration!

Transformation-Based Verification

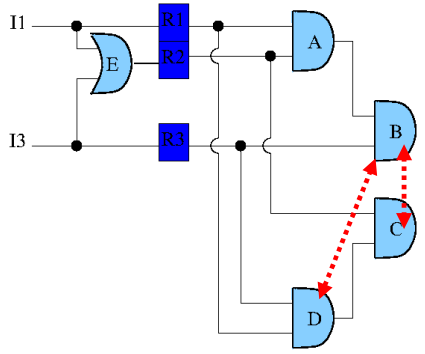


Example Transformations

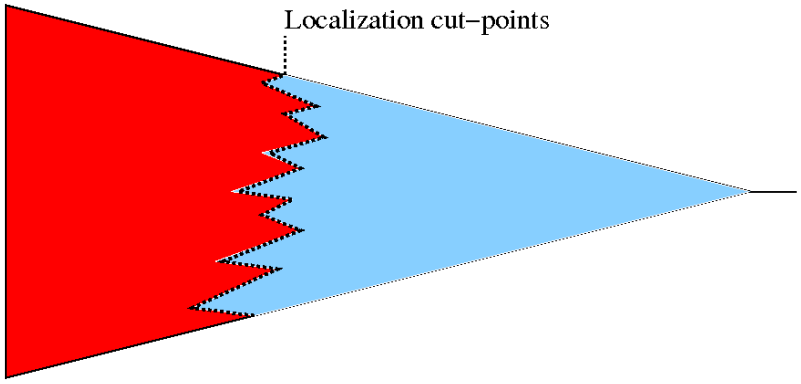
- Retiming



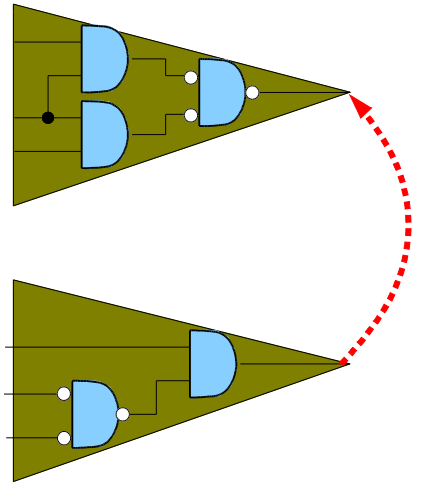
- Redundancy removal



- Localization



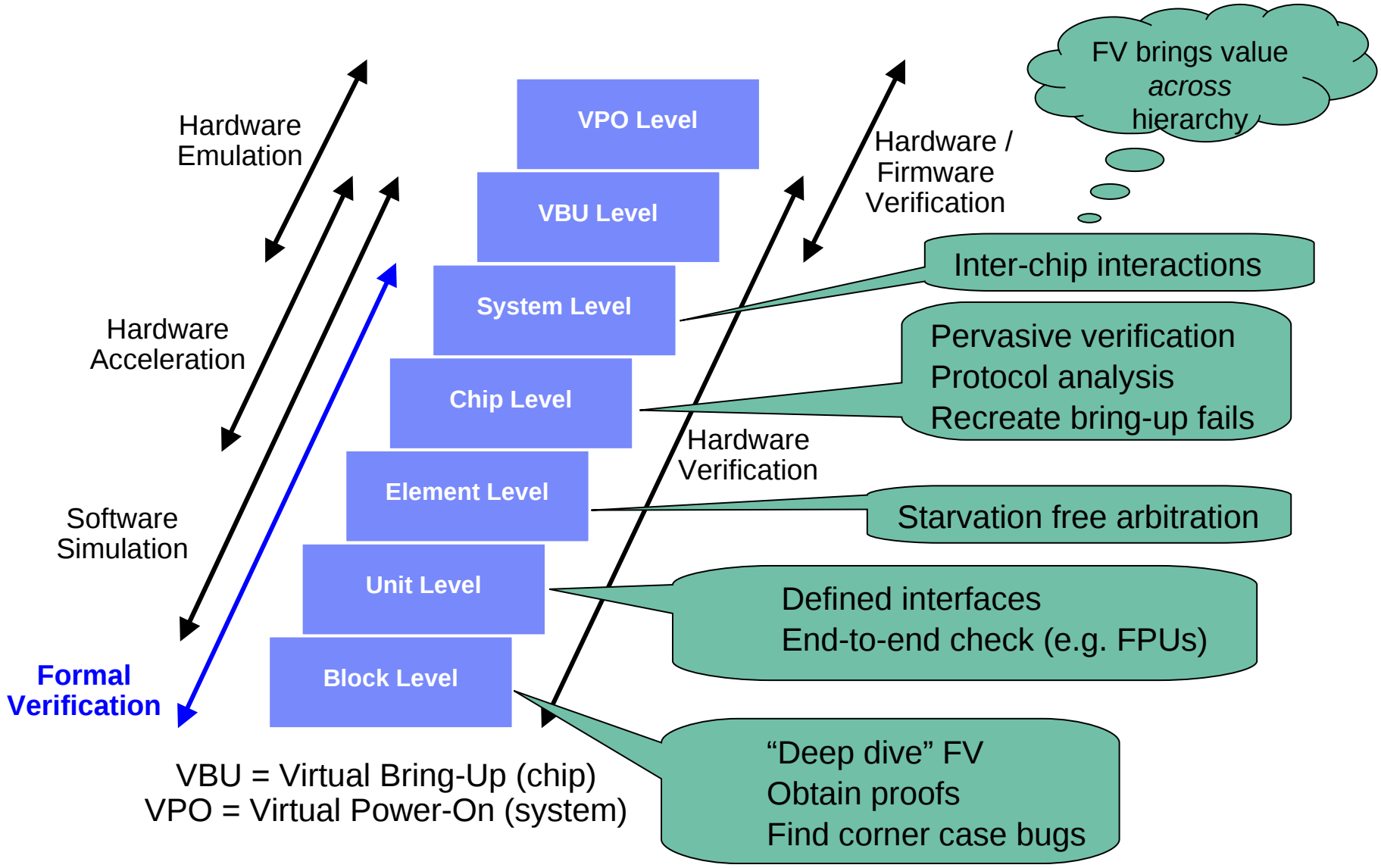
- Logic Rewriting



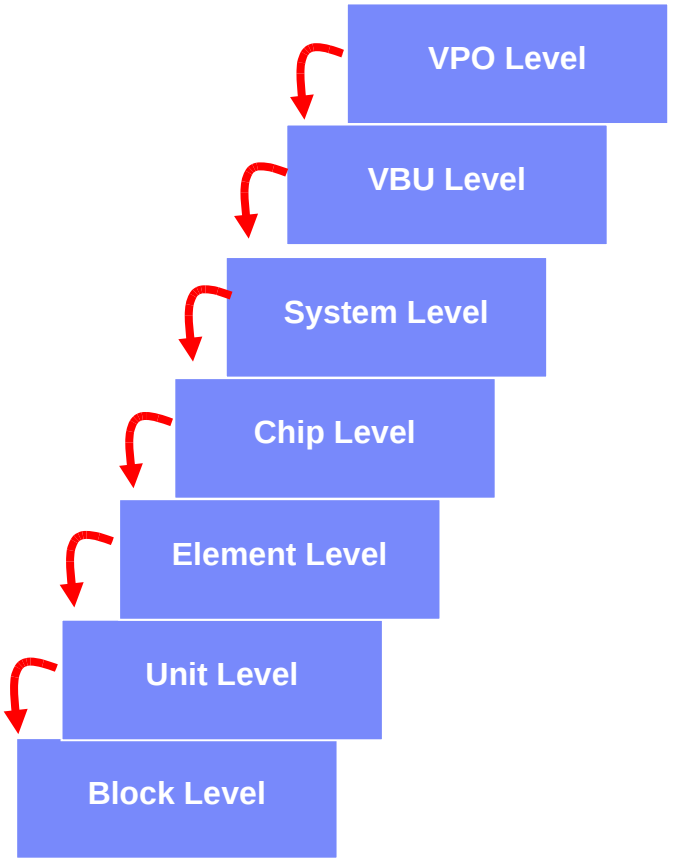
Agenda

- Verification at IBM (Background and History)
- Formal Verification using RuleBase SixthSense
- **Formal Verification: Execution/Adoption to IBM designs**

Hierarchical Verification Progression



Quality Refinement Process

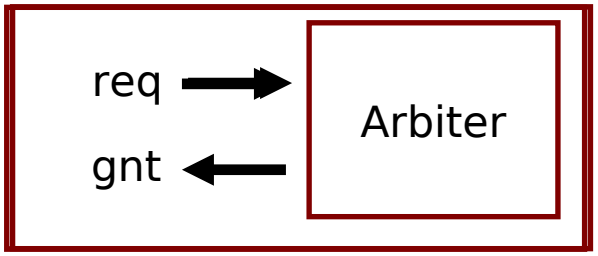


Because *controllability*, *state coverage* is higher, and *cost* of a bug is lower, at lower levels :

- Every major bug find at higher level is treated as *escape* of lower level
- Lower level team gets feedback to reproduce problems
  - Harden lower level environments
  - Reproduce with targeted block-level checkers
    - Prove fixes with formal verification

Liveness Checks

- Liveness property asserts that something good eventually will hold
- Example: request eventually should get a grant
- Trace consists of infinite length
- Simulation inherently incapable of proving liveness:
  - Checking is ad-hoc
  - Keeps no record of visited states



- Formal suitable for both proof and bug – but added complexity
- Liveness is a desirable property to verify off a variety of logics
  - Arbitration – check requests are eventually granted
  - FSM – a final state is eventually reached /there is no hang in the FSM
  - LRUs – every entry has a path to LRU
- In practice may be approximated by bounded safety checking
  - Check for the event occurring within a bound (time steps)
  - If we get a proof, we are done; counterexample may be spurious

Things to solve in formal

- In design/verification
  - Absence of detailed design documentation
  - Getting designers bandwidth
  - FV team not involved during early phases of design/verification
  - PACKs
- In tools
  - Improve bit-level verification, falsification algorithms !
  - Improve bit-level synthesis algorithms !
  - Improve high-level verification algorithms (e.g. SMT)

References

- Project homepage
  - http://www.haifa.il.ibm.com/projects/verification/RB_Homepage
- Technical publications
  - https://www.research.ibm.com/haifa/projects/verification/SixthSense
  - https://www.research.ibm.com/haifa/projects/verification/RB_Homepage/publications.html
- Contact:
  - Jason Baumgartner baumgarj@us.ibm.com
  - Viresh Paruthi vparuthi@us.ibm.com
  - Sudhakar R Amireddy samiredd@in.ibm.com
  - Pradeep Nalla pranalla@in.ibm.com
  - Eli Arbel arbel@il.ibm.com
  - Srobona Mitra sromitra@in.ibm.com