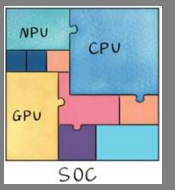


Topic 8 (Bonus Lecture[1])

SoC Verification – A Brief Overview

Xinfei Guo
xinfei.guo@sjtu.edu.cn

December 4th, 2024

[1] Materials in Bonus Lecture will not appear in the final exam.

T8 learning goals

- How to verify a SoC?
  - Motivation & Concepts

This lecture will be full of concepts…

Why SoC Verification?

What is SoC Design Verification?

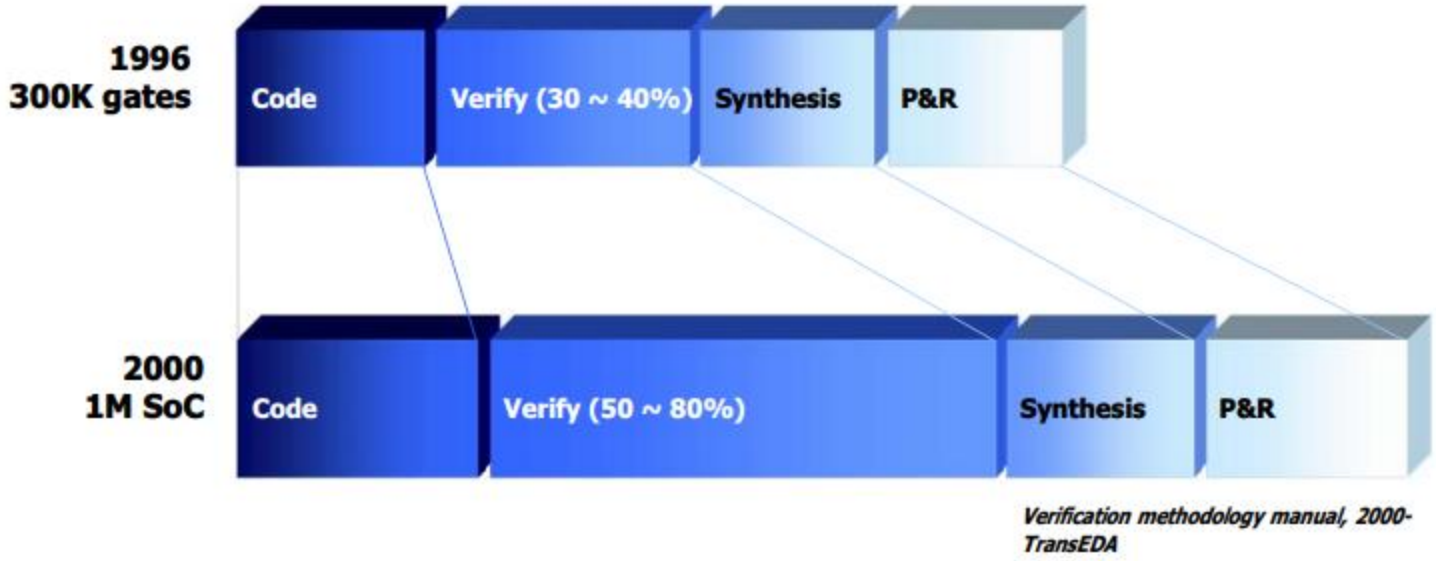
- Design Verification is the process of checking that a given design correctly implements the specification.
- The largest task in SoC development that has the biggest impact on the key business drivers:
  - quality, schedule, design to market and cost
- **Verification team to Design team ratio ranges from 2:1 to 3:1**
- > 70% development cycle is dedicated to verification
- A diverse domain/field with endless strategies/techniques to sign-off (finalize) the chip
- Every design needs verification - need for highly skilled design verification engineers to meet the challenges of the state-of-the-art technological innovations

Industry Always Look for Verification Engineers

source: Dr. Gul N. Khan

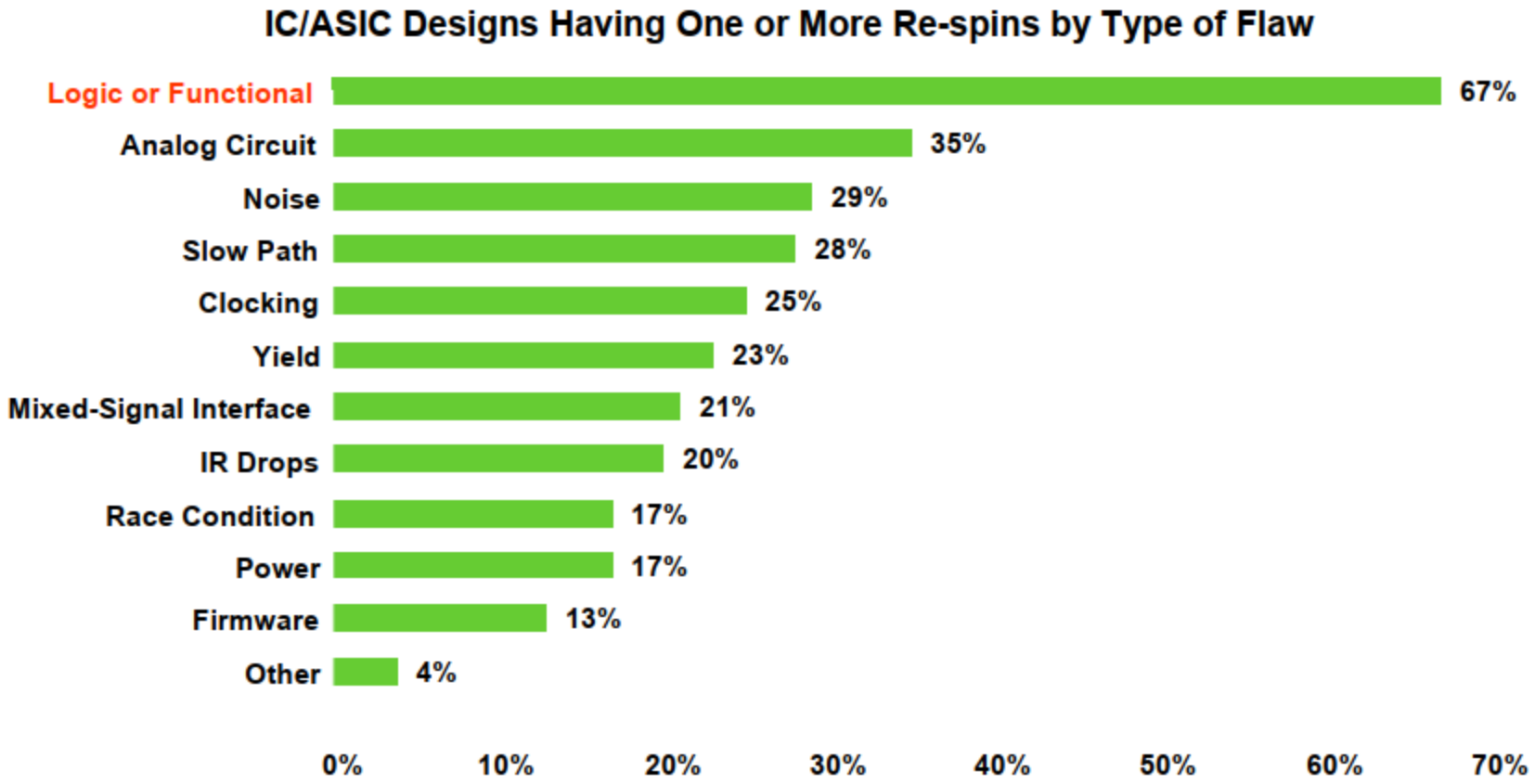
Verification Effort

- Verification portion of design increases to anywhere from 50 to 80% of total development effort for the design.



source: J. Abraham, UT Austin

Verification Challeges



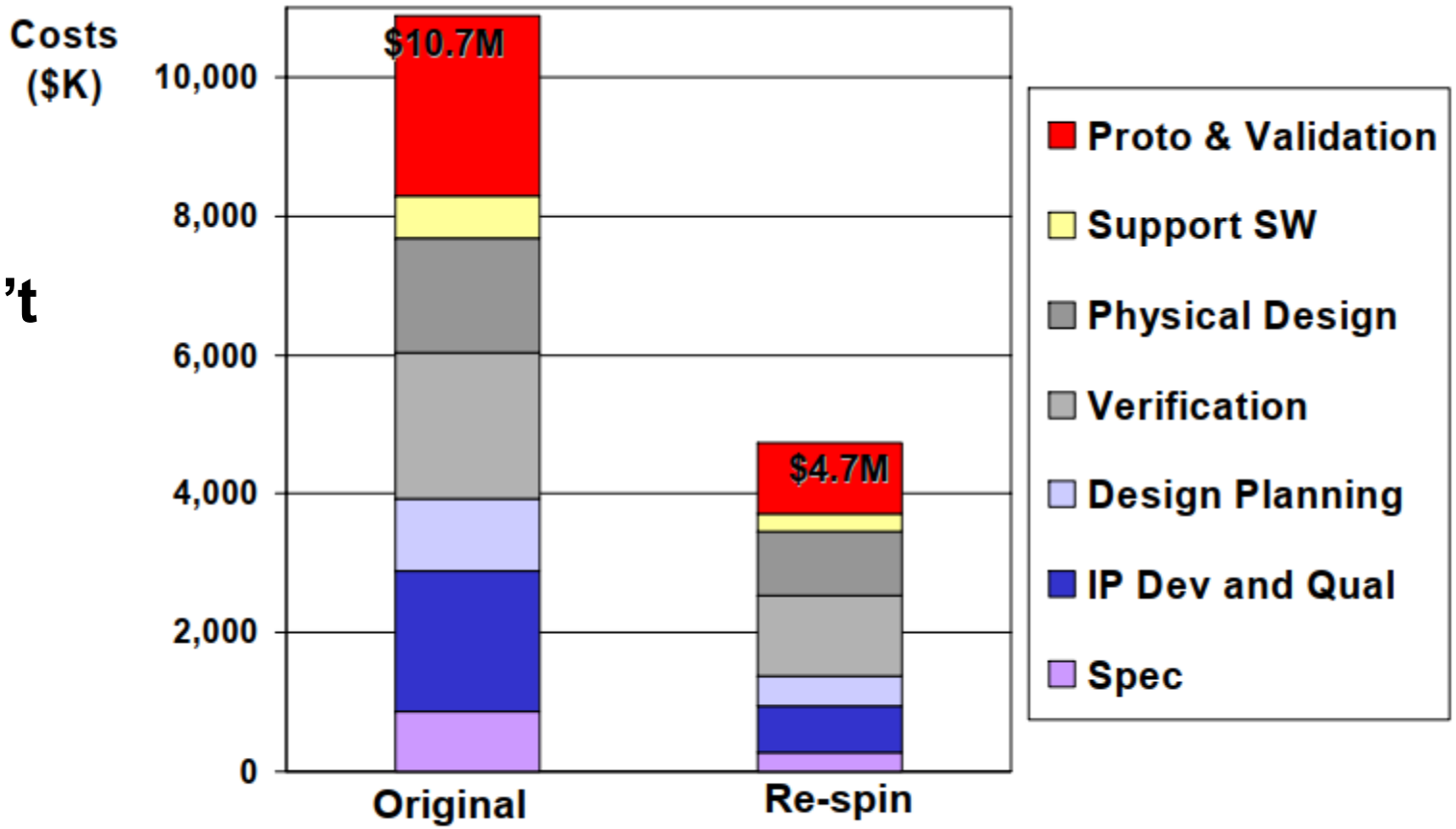
Verification Goal is 100% correct, but it is mission impossible…

Source: Collett International Research (Apr02)

Re-spins are expensive!

What if things don't work?

Answer: Respin!!!

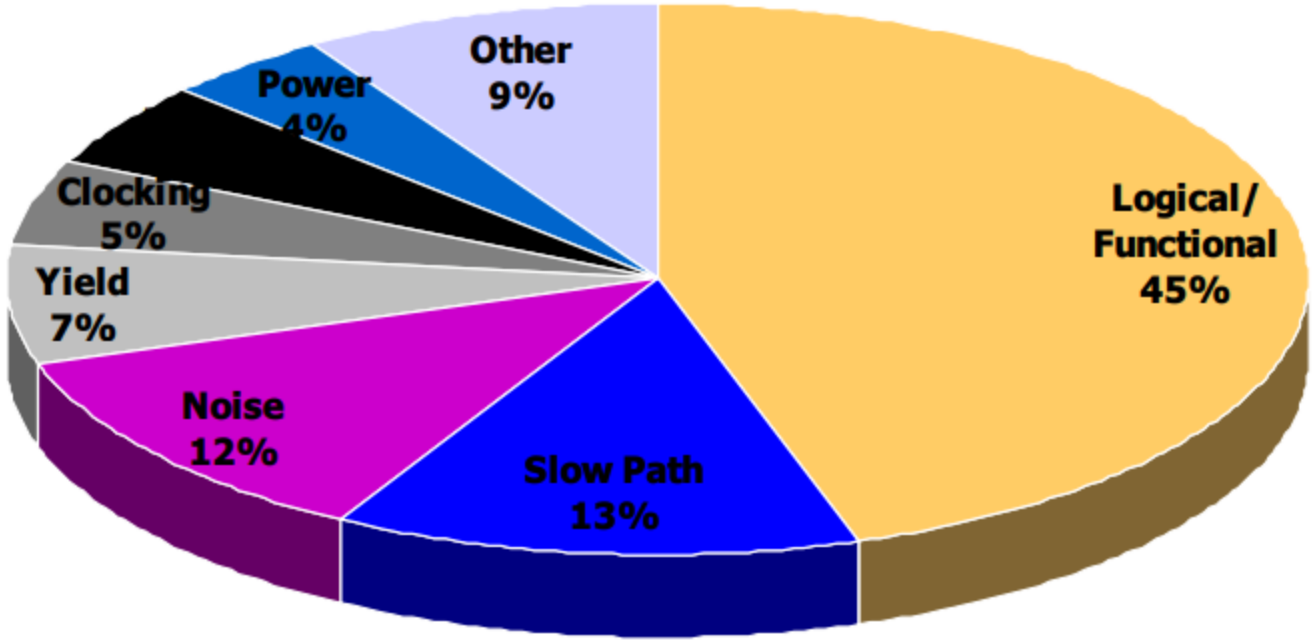


Plus a) lost revenue, b) opportunity costs

Source: International Business Strategies

Percentage of Total Flaws

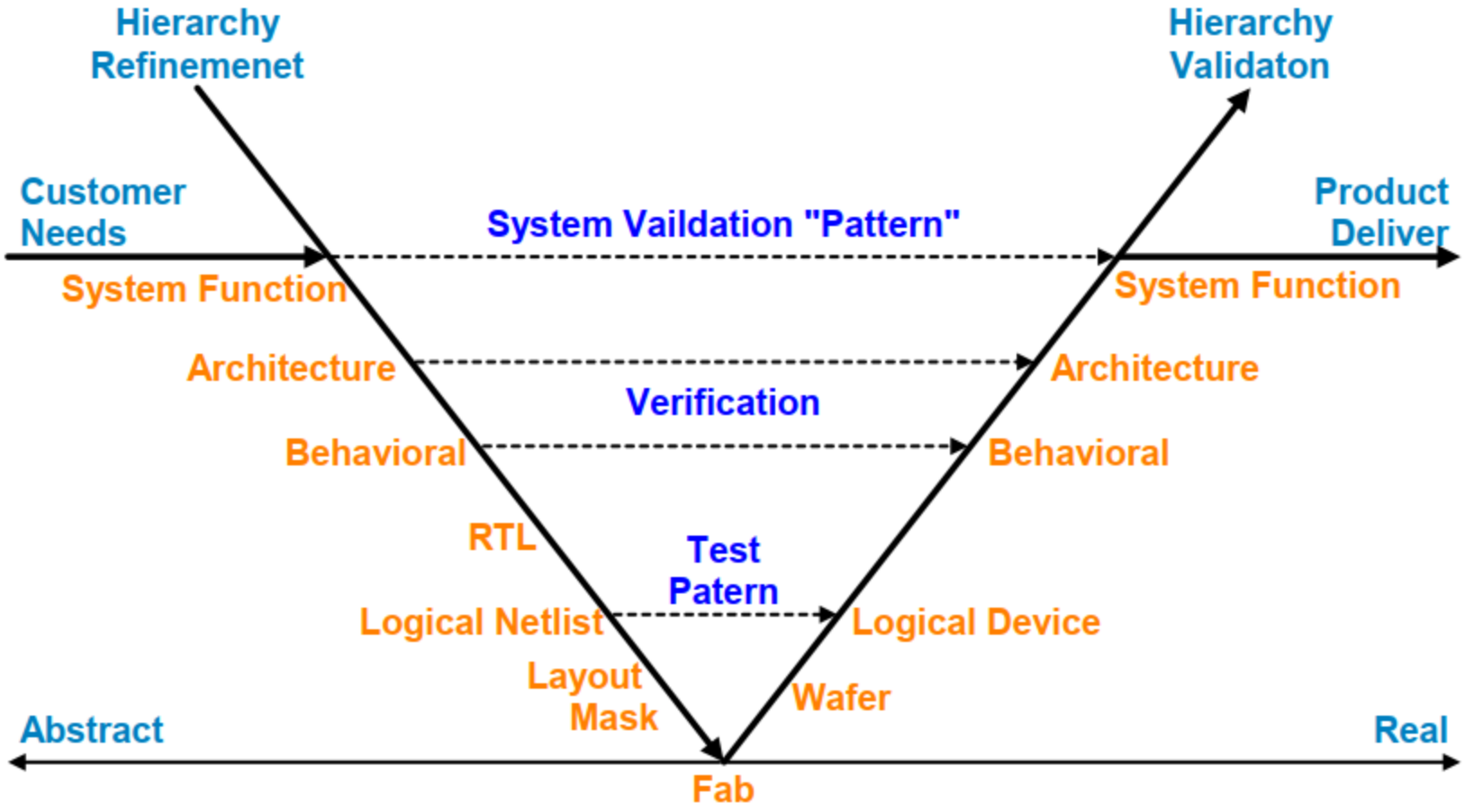
- About 50% of flaws are functional flaws
  - Need verification method to fix logical & functional flaws



source: J. Abraham, UT Austin

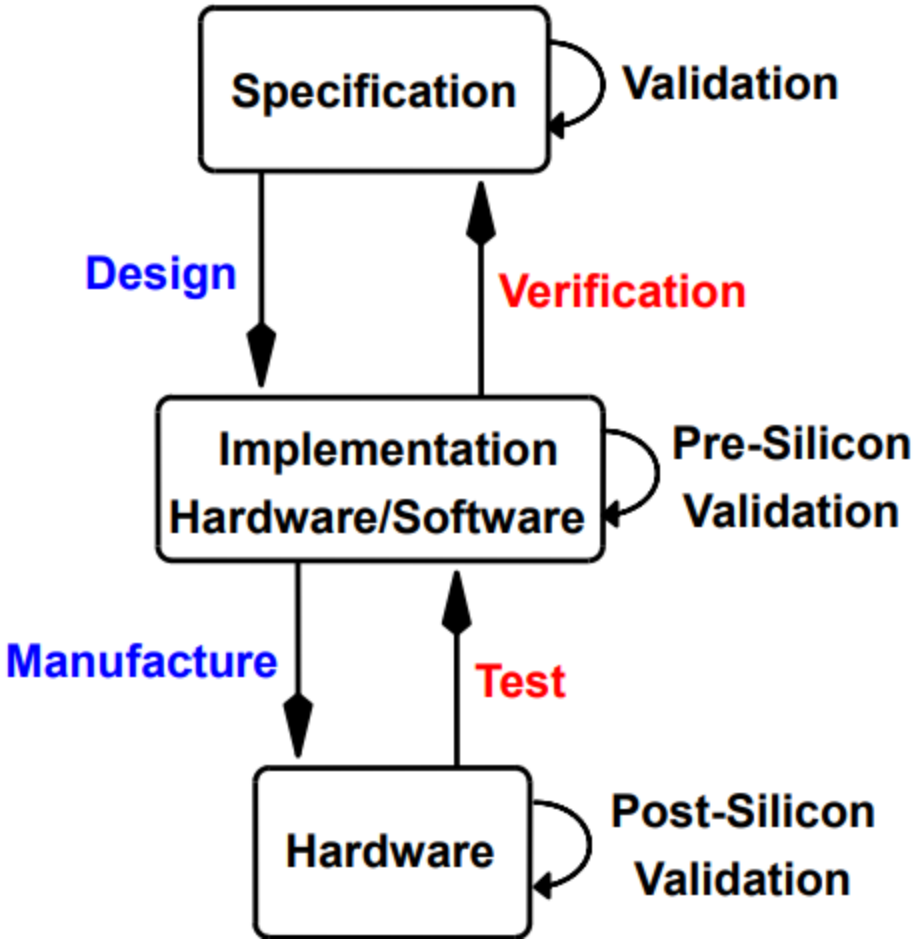
Verification Basics

From Requirement to Delivery (V curve)



source: Dr. Gul N. Khan

Verification vs. Test



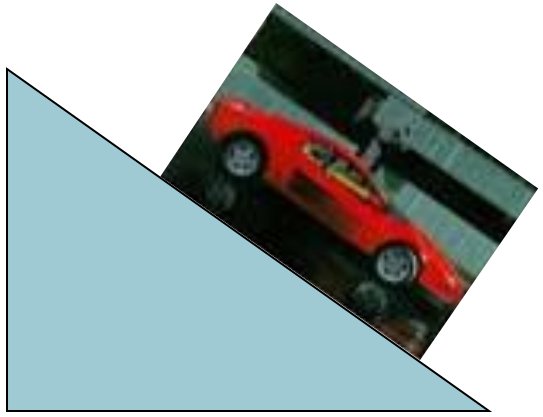
source: J. Abraham, UT Austin

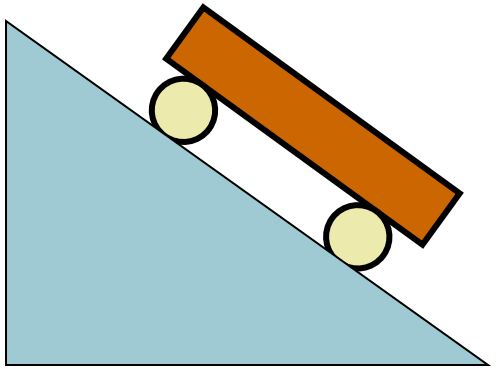
System-on-Chip Verification Challenges

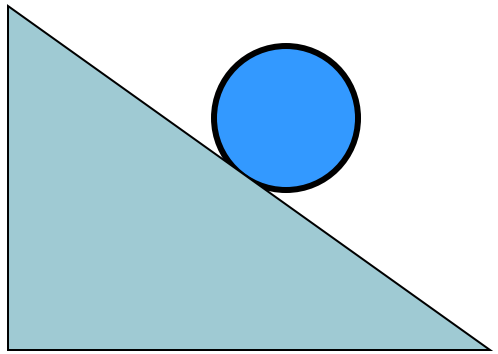
- Verification goals
  - functionality, timing, performance, power, physical and more
- Design complexity
  - MPUs, MCUs, DSPs, AMS IPs, ESW, clock/power distribution, test structures, interface, telecom, multimedia

Physical issues verification Barriers

- Increasing complexity $\Rightarrow$ bigger and more complex models
  - E.g. supply grid, parasitics…
- Need to find a “reduced” model so that
  - Still good representation
  - Manageable size



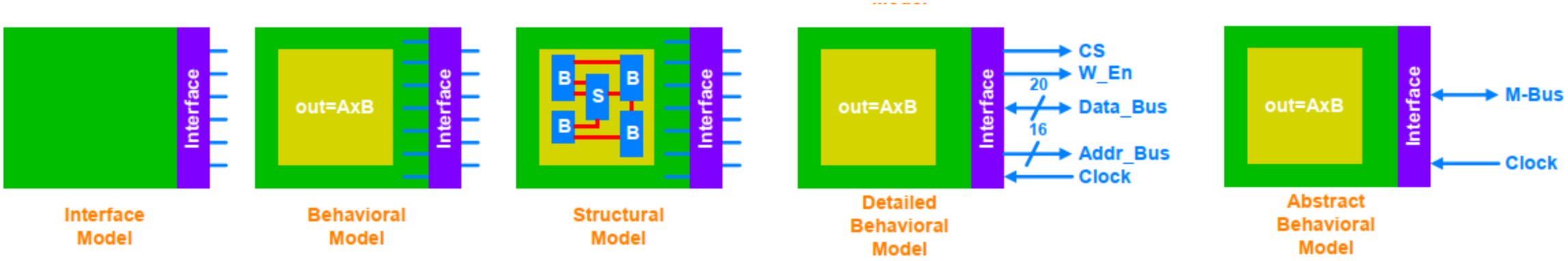




source: Dr. Pao-Ann Hsiun, CCD

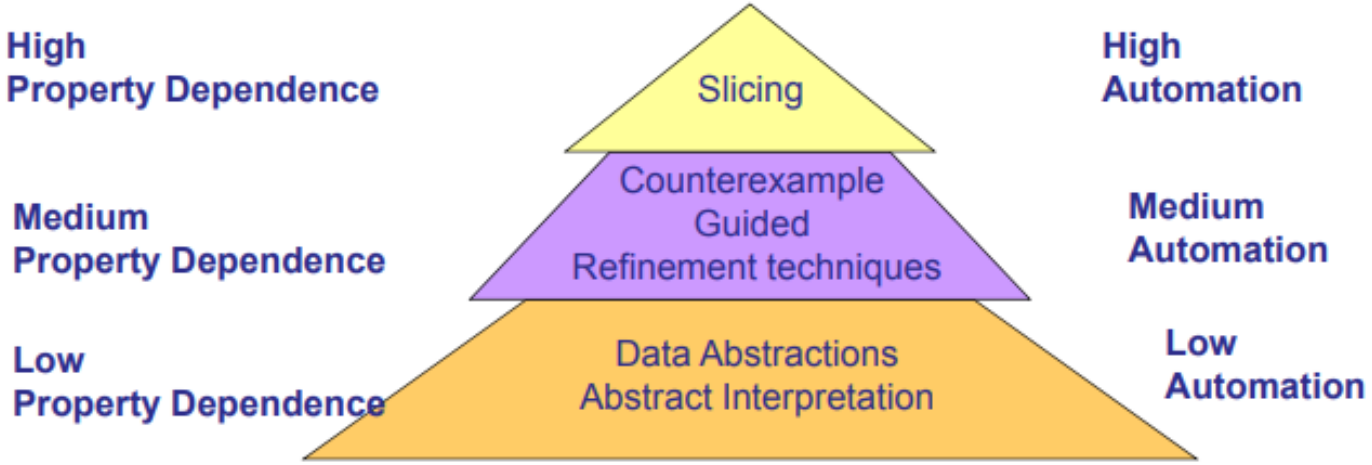
IP Modeling

- Design exploration at higher level
  - Import of top-level constraint and block architecture
  - Hierarchical, complete system refinement
  - Less time for validating system requirement
  - More design space of algorithm and system architecture



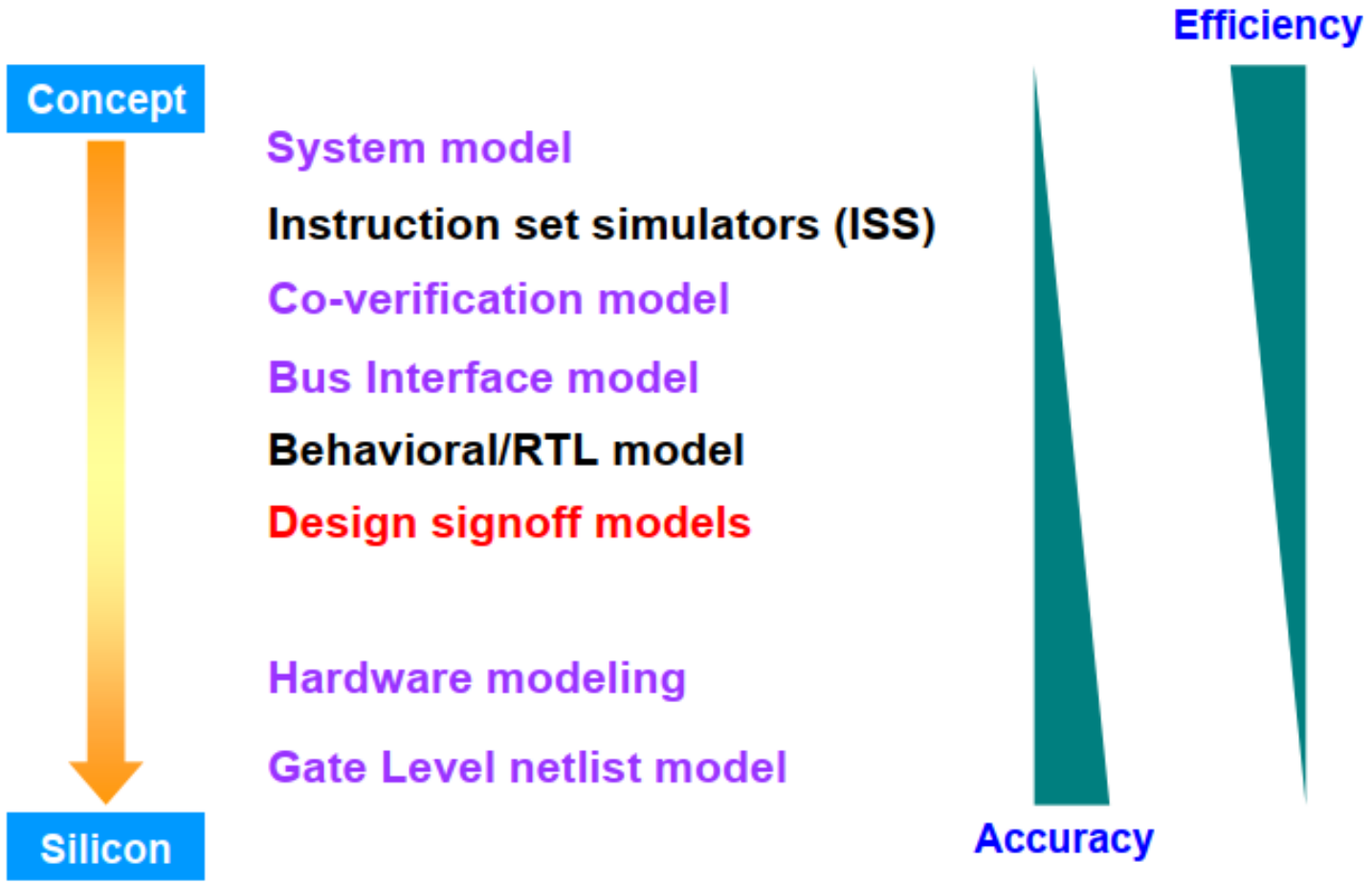
source: Dr. Gul N. Khan

Abstraction Landscape



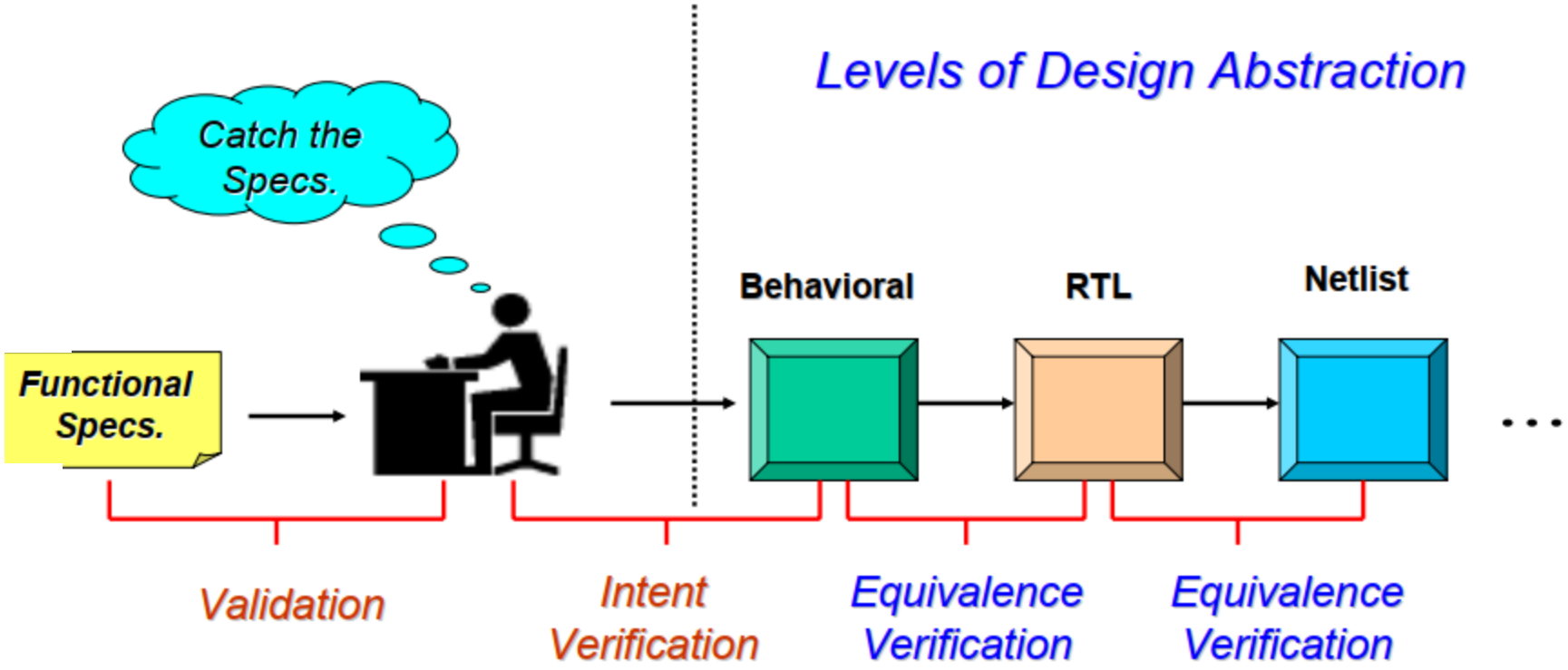
source: A. Gerstlauer, UT Austin

ARM Modeling



source: Dr. Gul N. Khan

Classification of Verification



source: Dr. Gul N. Khan

Verification Methods

- Simulation Technologies
- Static Technologies
- Formal Technologies
- Physical Verification and Analysis

SIMULATION TECHNOLOGIES

Simulation Technologies

- Event-driven Simulators
- Cycle-based Simulators
- Rapid Prototyping Systems
- Emulation Systems
- Speeding up Simulators (C, BFM, ISS,…)
- Testing & Coverage-driven Verification
- Assertion-based Verification
- HW/SW Co-simulation
- AMS Modeling and Simulation

Hardware Simulation

- Event-driven
  - compiled code
  - native compiled code (directly producing optimized object code)

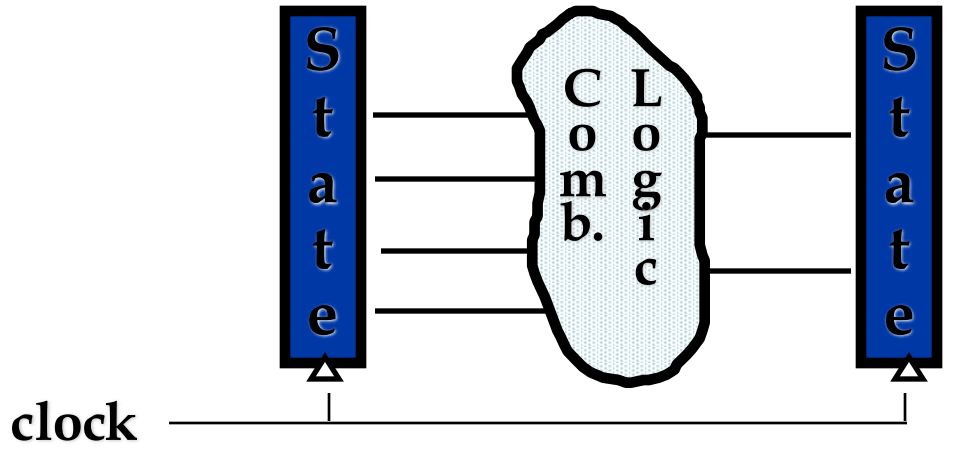
  – **very slow**

  + asynchronous circuits, timing verification, initialize to known state

- Cycle-based

  + faster (3-10x than NCC)

  – synchronous design, no timing verification, cannot handle x,z states



source: Dr. Pao-Ann Hsiun, CCD

Simulation: Performance vs Abstraction

Cycle-based Simulator

Event-driven Simulator

SPICE

Abstraction

.001x

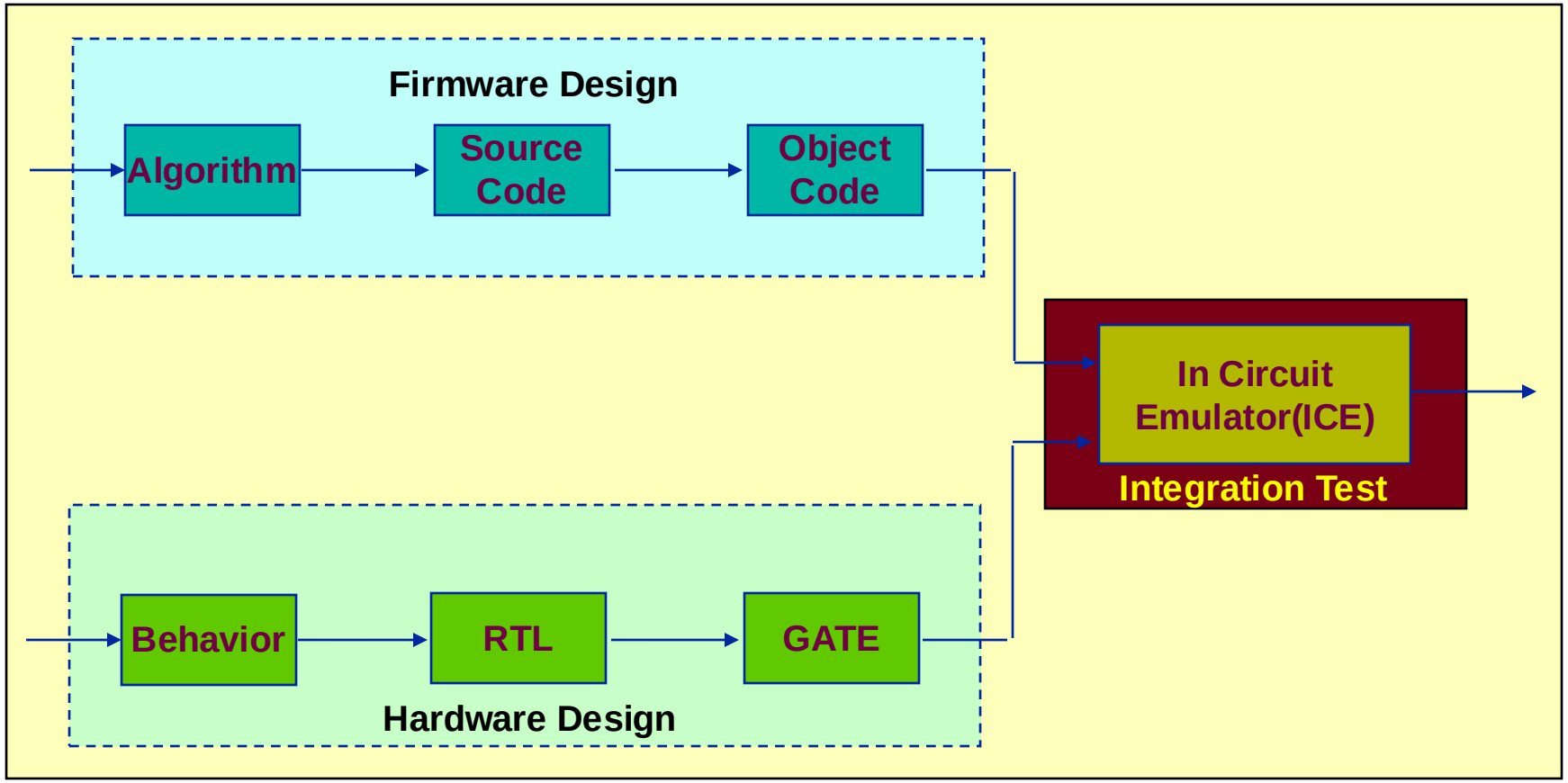
1x

10x

Performance and Capacity

source: Dr. Pao-Ann Hsiun, CCD

Rapid Prototyping Systems



source: Dr. Pao-Ann Hsiun, CCD

Prototyping with FPGAs

- Hardware prototyping
  - Based on FPGA boards
  - Low visibility of internals but high speed and characteristics close to real IC
  - Used for software development and testing
- Virtual prototyping
  - Performed by software in early stages of design
  - Used for early software development

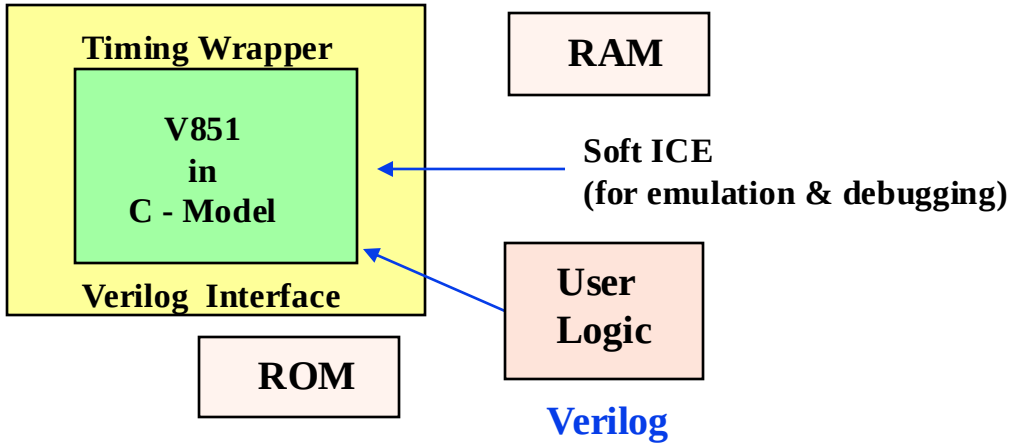
Enhancing Simulation Speed Using Simulation Models

- Hardware model
- Behavioral model in *C*
- Bus-functional model (BFM)
- Instruction-Set simulation (ISS) model
  - instruction accurate
  - cycle accurate
- Full-timing gate-level model
  - encrypted to protect IP

source: Dr. Pao-Ann Hsiun, CCD

Example of Simulation Models

- NEC provides the following simulation models:
  - **Behavioral C model**: used in early design stage, for functional verification, fastest execution
  - **RTL model with timing wrapper**: for accurate timing and function verification
  - **Verilog gate-level model**: for final design verification, very slow



source: Dr. Pao-Ann Hsiun, CCD

Coverage

Concept of Coverage

- A metric identifies important:
  - Structures in a design representation
  - e.g. HDL lines, FSM states, paths in netlist
- Classes of behavior
  - Transactions, event sequences
- Maximize the probability of simulating and detecting bugs, at minimum cost (in time, labor, and computation)
- Difficult to formally prove that a coverage metric provides a good proxy for bugs
- Goal
  - Comprehensive validation without redundant effort

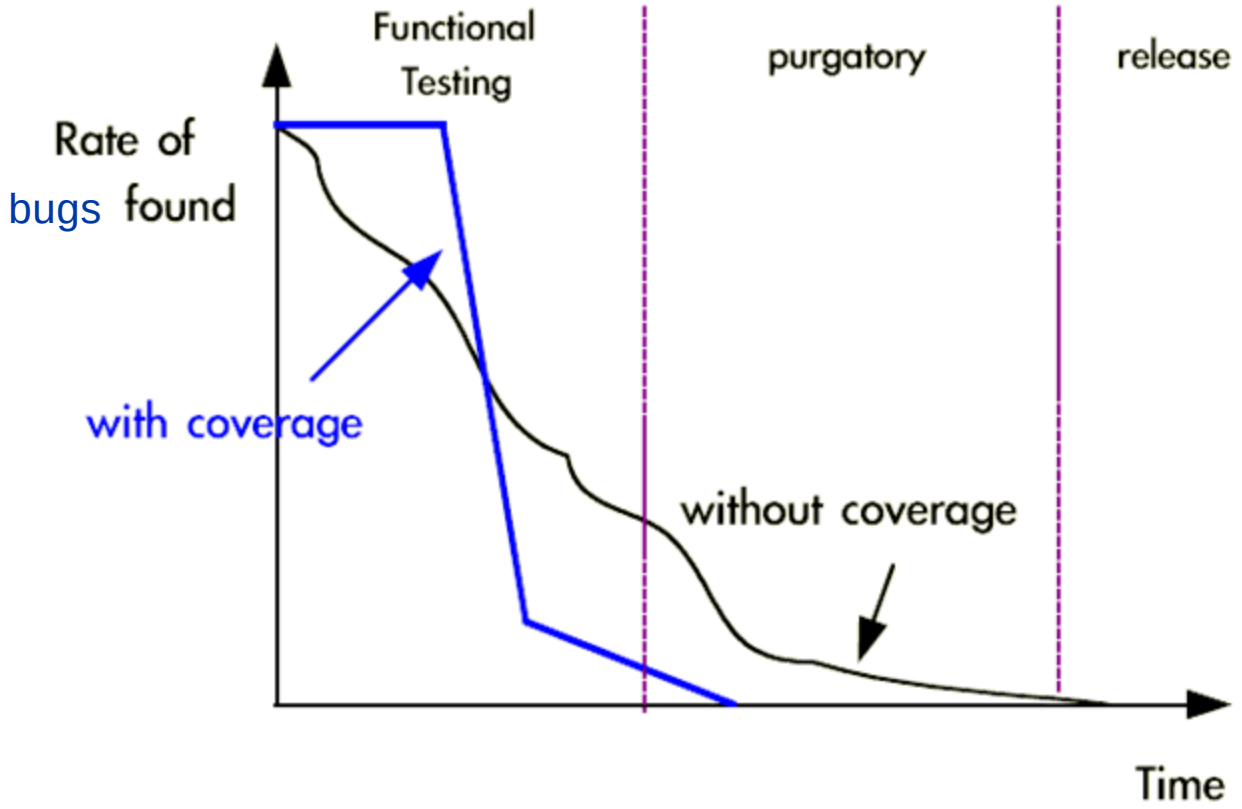
source: Dr. Gul N. Khan

Coverage-driven Verification

- Coverage reports can indicate how much of the design has been exercised
  - Point out what areas need additional verification
- Optimize regression suite runs
  - Redundancy removal (to minimize the test suites)
  - Minimizes the use of simulation resources
- Quantitative sign-off (the end of verification process) criterion
- **Verify more but simulate less**

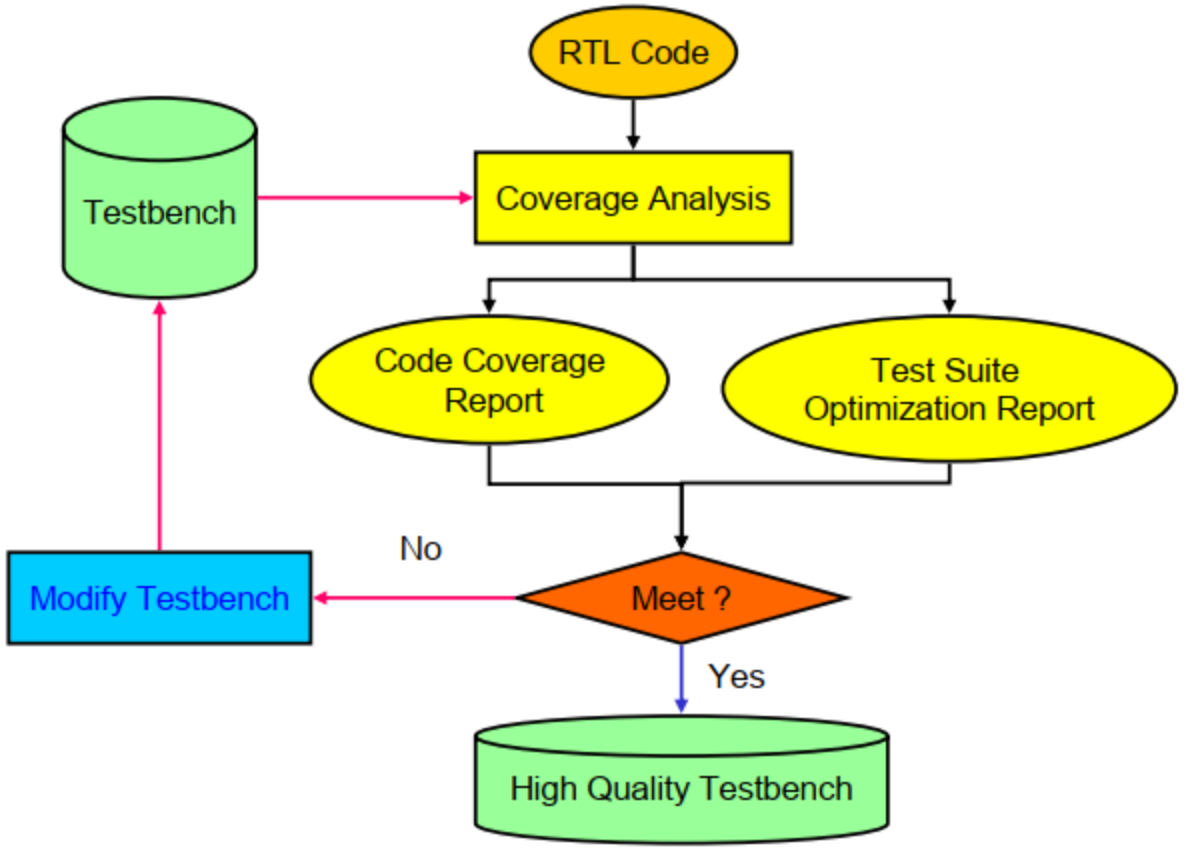
source: Dr. Pao-Ann Hsiun, CCD

The rate of bug detection



source : "Verification Methodology Manual For Code Coverage In HDL Designs" by Dempster and Stuart

Code Coverage Flow

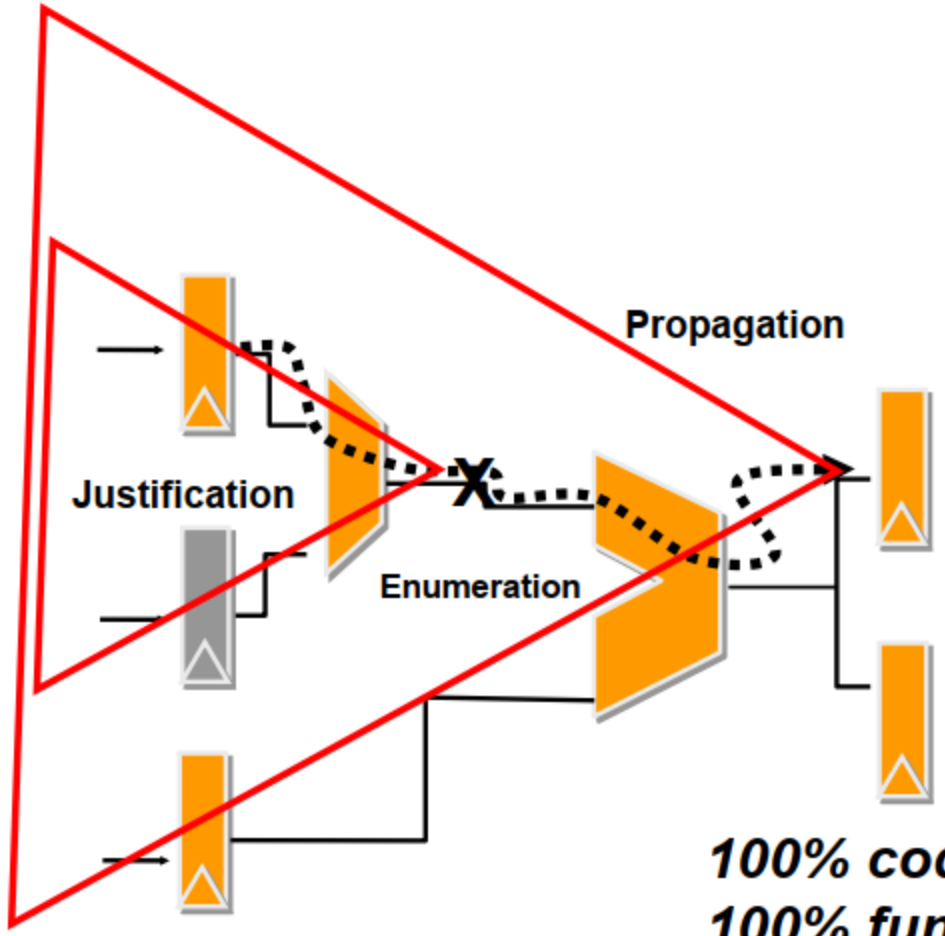


source: Dr. Gul N. Khan

Drawbacks of Code Coverage

- No **qualitative** insight into functional correctness
- Limited to measure what is controllable
- Activating an erroneous statement does not mean the bug will manifest itself to an observable output
  - Like testing problems
  - Cases found where **90%** line coverage only achieved **54%** observability coverage [source: Devadas et al. ICCAD 96]

Controllability vs. Observability



Controllability

vs.

Observability

100% code coverage does not imply
100% functional coverage !

source: Dr. Gul N. Khan

Increase Observability

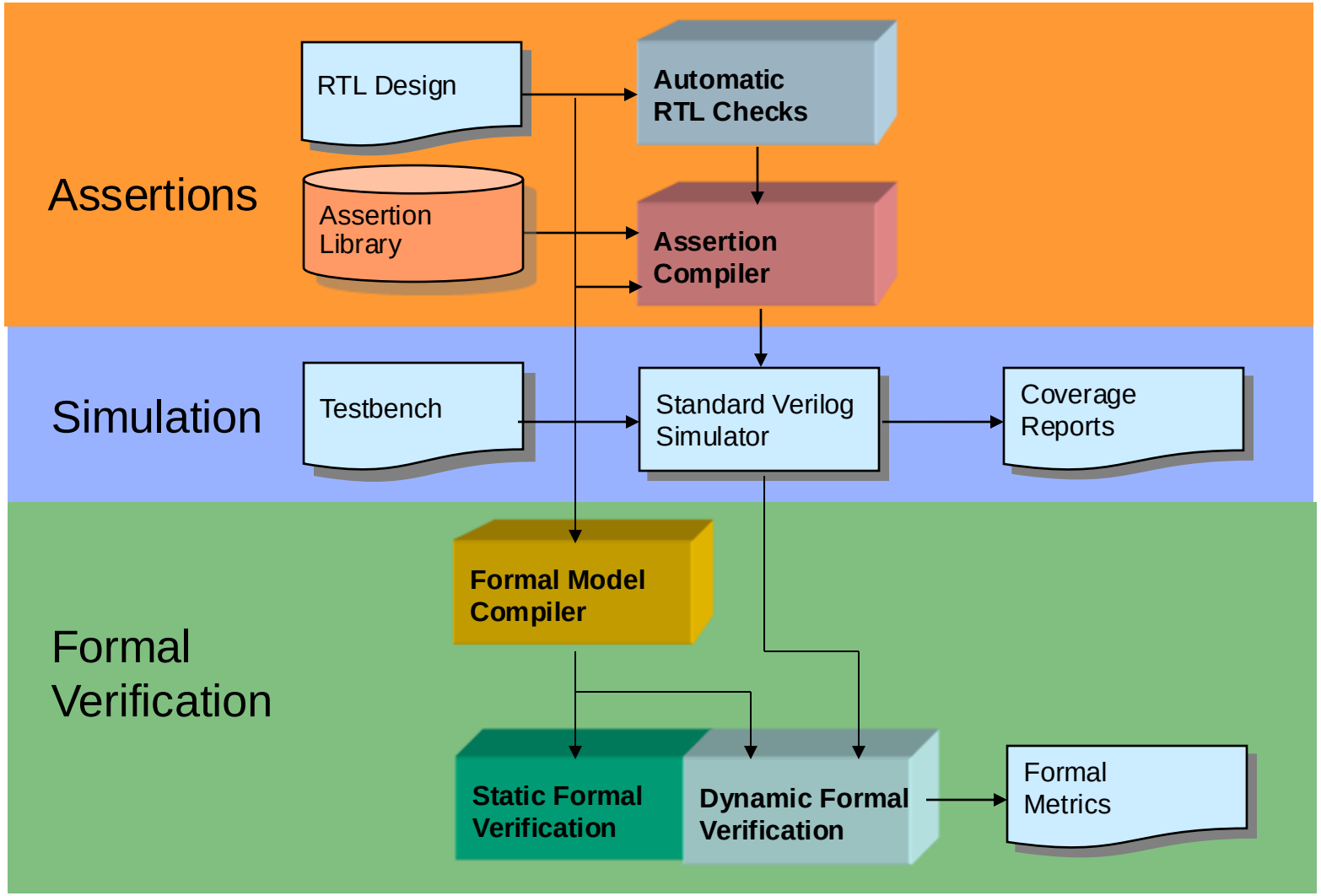
- Event-Monitors and Assertion Checkers
  - Halt simulation (if desired)
  - Simplifies Debugging
  - Increases test stimuli observability
  - Measure functional coverage (using a line cover tool)
  - Enables formal and semi-formal techniques
  - Capture and validate design assumptions and constraints

ABV

- Assertion-based verification (ABV) solutions are gaining in popularity
  - Assertions are statements of designer assumptions or design intent
  - Assertions should be inherently reusable
- These supplement, not replace, traditional simulation tests
  - Design and verification engineer knowledge is leveraged
  - Both design observability and design controllability can be improved
  - Assertions enable formal verification

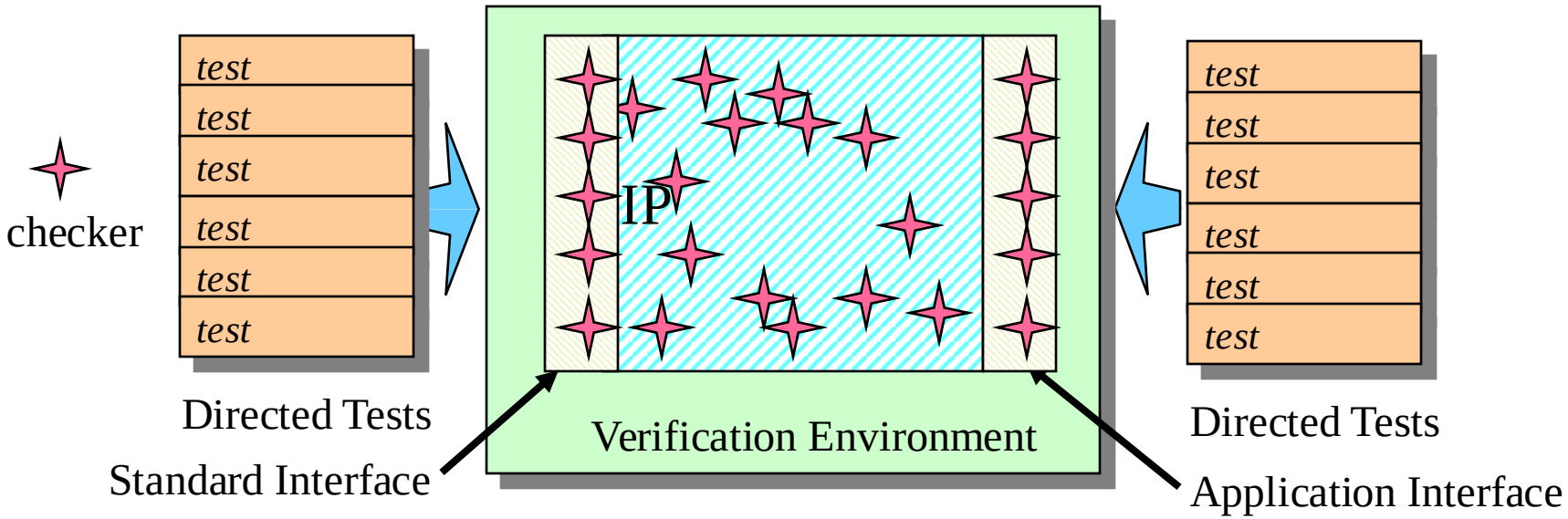
source: Dr. Pao-Ann Hsiun, CCD

ABV Flow



source: Dr. Pao-Ann Hsiun, CCD

IP Verification with Checkers



- Use checkers during interface IP creation
  - Saturate RTL code with checkers
  - Use checkers on interfaces as trip-wires
  - Report illegal inputs and scenarios not handled
- Deliver IP with assertion checkers included

source: Dr. Pao-Ann Hsiun, CCD

Static Technologies

Static Technologies

- "Lint" Checking
  - Syntactic correctness
  - Identifies simple errors
- Static Timing Verification
  - Setup, hold, delay timing requirements
  - Challenges: multiple sources

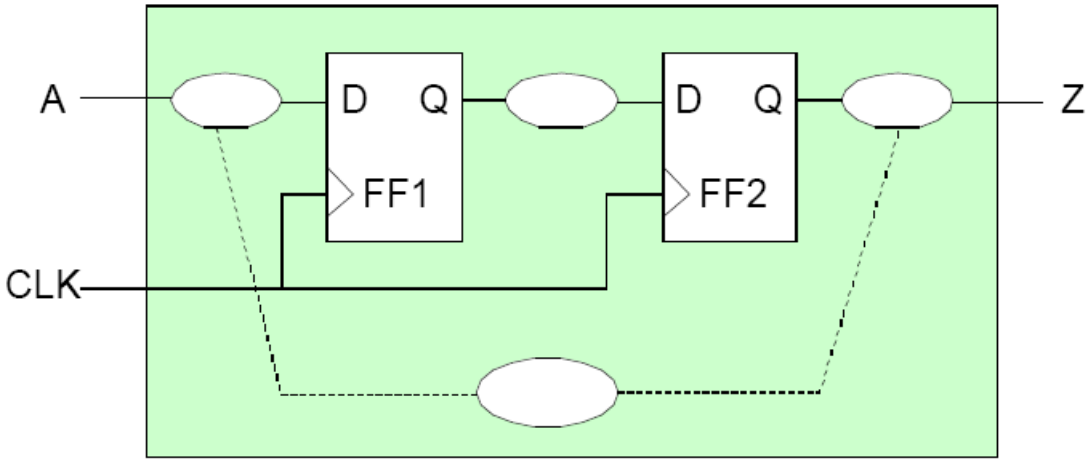
HDL Linter

- Fast static RTL code checker
  - Preprocessor of the synthesizer
  - **RTL purification (RTL DRC)**
    - Syntax, semantics, simulation
  - Check for built-in or user-specified rules
    - Testability checks
    - Reusability checks
    - ……
  - Shorten design cycle
    - Avoid error code that increases design iterations

source: Dr. Pao-Ann Hsiun, CCD

Static Timing Analysis (STA)

- STA is a method for determining if a circuit meets timing constraints (setup, hold, delay) without having to simulate
- No input patterns are required
  - 100% coverage if applicable
- Challenging: multiple sources



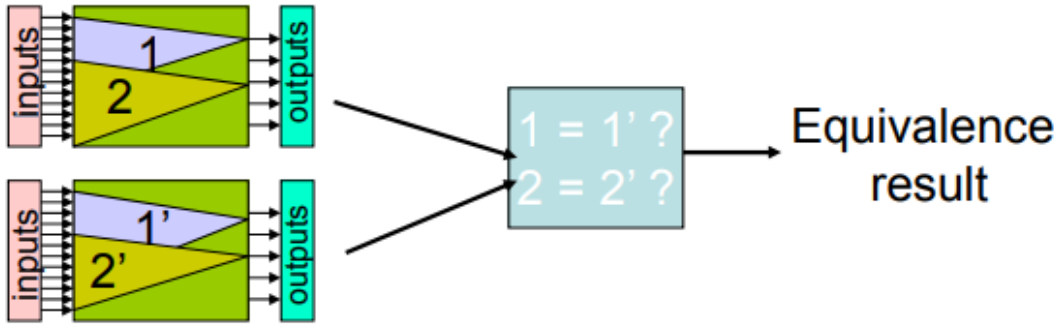
Reference :Synopsys

source: Dr. Pao-Ann Hsiun, CCD

Formal Verification

Formal Verification

- Formal Verification: An analytic way of proving a system correct
  - no simulation triggers, stimuli, inputs
  - no test-benches, test-vectors, test-cases
- Logic Equivalence Checking (LEC) uses boolean algebra to check for logic equivalence



source: A. Gerstlauer, UT Austin

Why Formal Verification Now?

- SoC has a high system complexity
- Simulation and test are taking unacceptable amounts of time
- More time and efforts devoted to verification (40% ~ 70%) than design
- Need automated verification methods for integration into design process

source: Dr. Pao-Ann Hsiun, CCD

Why Formal Verification Now?

Examples of undetected errors

- Ariane 5 rocket explosion, 1996
  - Exception occurred when converting 64-bit floating number to a 16-bit integer!
- Pentium FDIV bug
  - Multiplier table not fully verified!

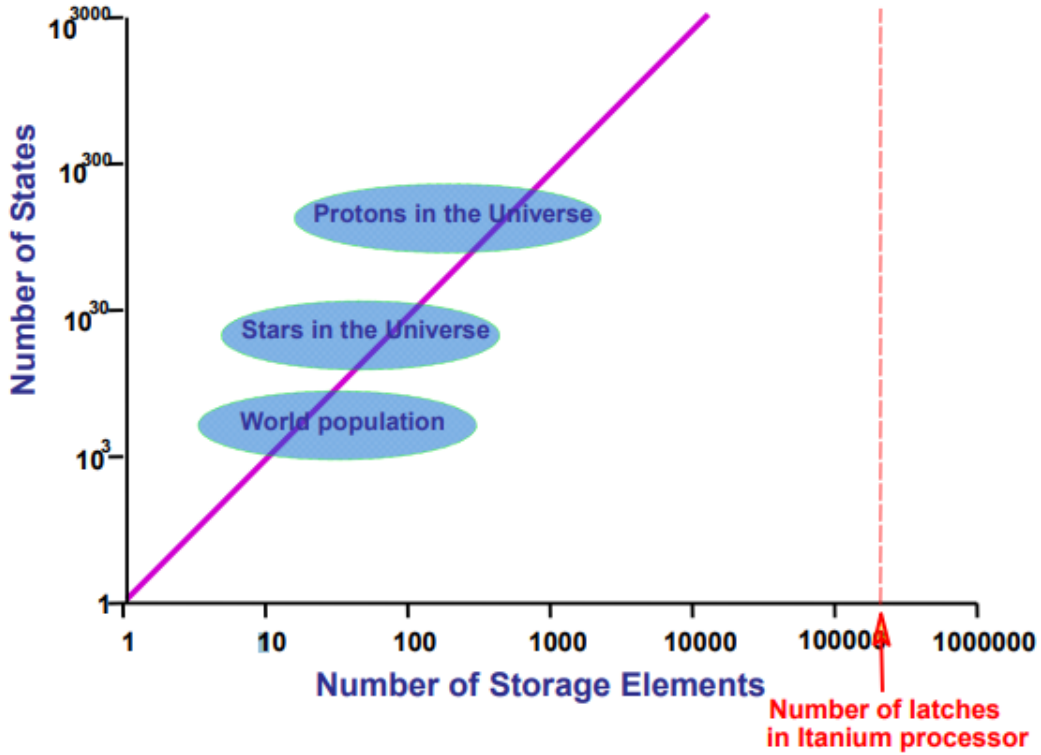
source: Dr. Pao-Ann Hsiun, CCD

“The Worst Computer Bugs in History”

source: Dr. Pao-Ann Hsiun, CCD

Formal Verification Issues

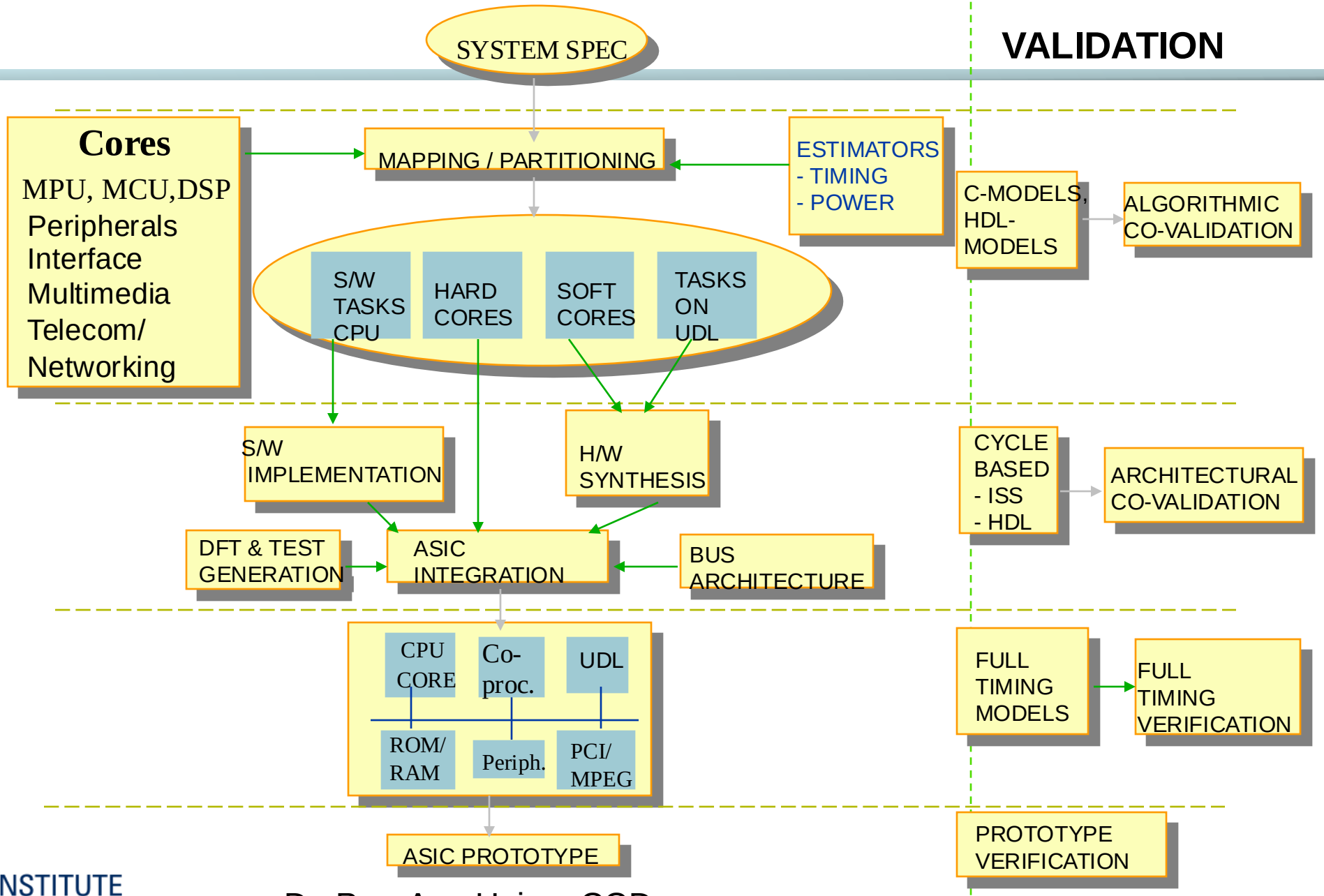
- State-space explosion!!!
- Cannot handle large systems!
  - For control-oriented behavior of small modules
  - For interface-centric verification
  - Constrained for feasible verification
- Supplementary to simulation
  - Counterexample → simulation trace



Quick Sum: Which is the fastest option?

- Event-based simulation
  - Best for asynchronous small designs
- Cycle-based simulation
  - Best for medium-sized designs
- Formal verification
  - Best for control-oriented designs
- Emulation
  - Best for large capacity designs
- Rapid Prototype
  - Best for software development

Putting it all together: SoC Design and Validation Flow



source: Dr. Pao-Ann Hsiun, CCD

Summary

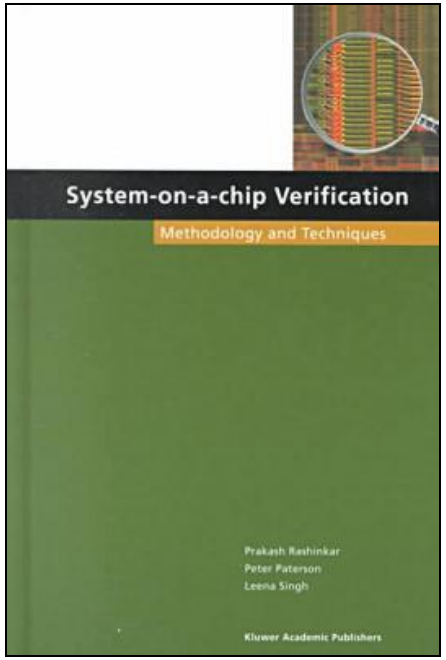
- Verification is tedious yet critical
- The concept of abstraction also applies in verification field
- SoC verification requires increasing efforts
  - Scaling this effort to chiplet scale now?
- Different types of verifications

Where are we Heading?

- Course Review

Action Items

- Final projects – Notice the deadline
- Reading Assignment
  - Slides
  - Reference
    - System-on-a-Chip Verification Methodology and Techniques by Prakash Rashinkar, Peter Paterson, Leena Singh, Kluwer Academic Publishers, 2001



Acknowledgement

Slides in this topic are inspired in part by material developed and copyright by:

- Dr. Jacob A. Abraham (UT Austin)
- Dr. Andreas Gerstlauer (UT Austin)
- Dr. Tian Sheuan Chang (NTU)
- Dr. Gul N. Khan (Ryerson University)
- Dr. Pao-Ann Hsiung (CCU)