

Hardware Functional Verification

Klaus-Dieter Schubert
eServer Verification
IBM Corp
Böblingen

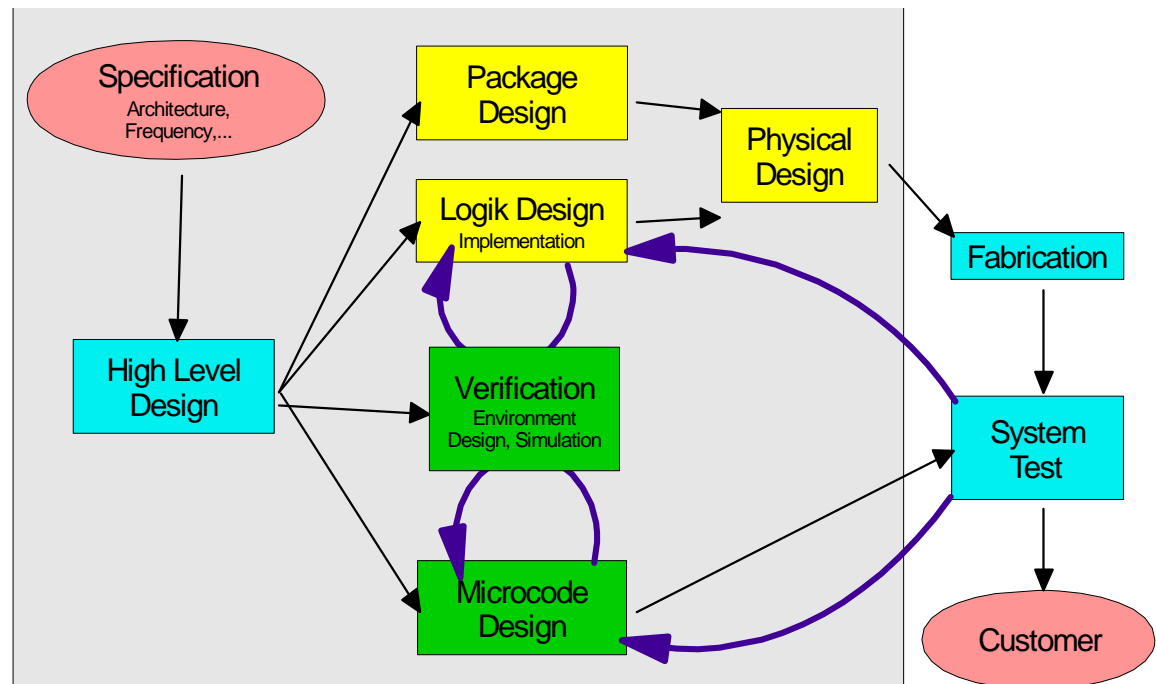
Contents

- Why Verification?
- What is Verification?
- Design
 - Example
 - Categories
- Verification
 - Testplan
 - Formal Verification
 - Simulation
 - HW / FW Co-Simulation
 - Escape Analysis
 - Guidelines
- Trends
 - Challenges
 - Evolution



Why verification?

- Time-to-market
 - Hardware turn-around time too slow
 - Debug of „bugs“ time consuming on real hardware
- Costs
 - Reduce „bugs“ to save test hardware
 - Justifies large investment into simulation

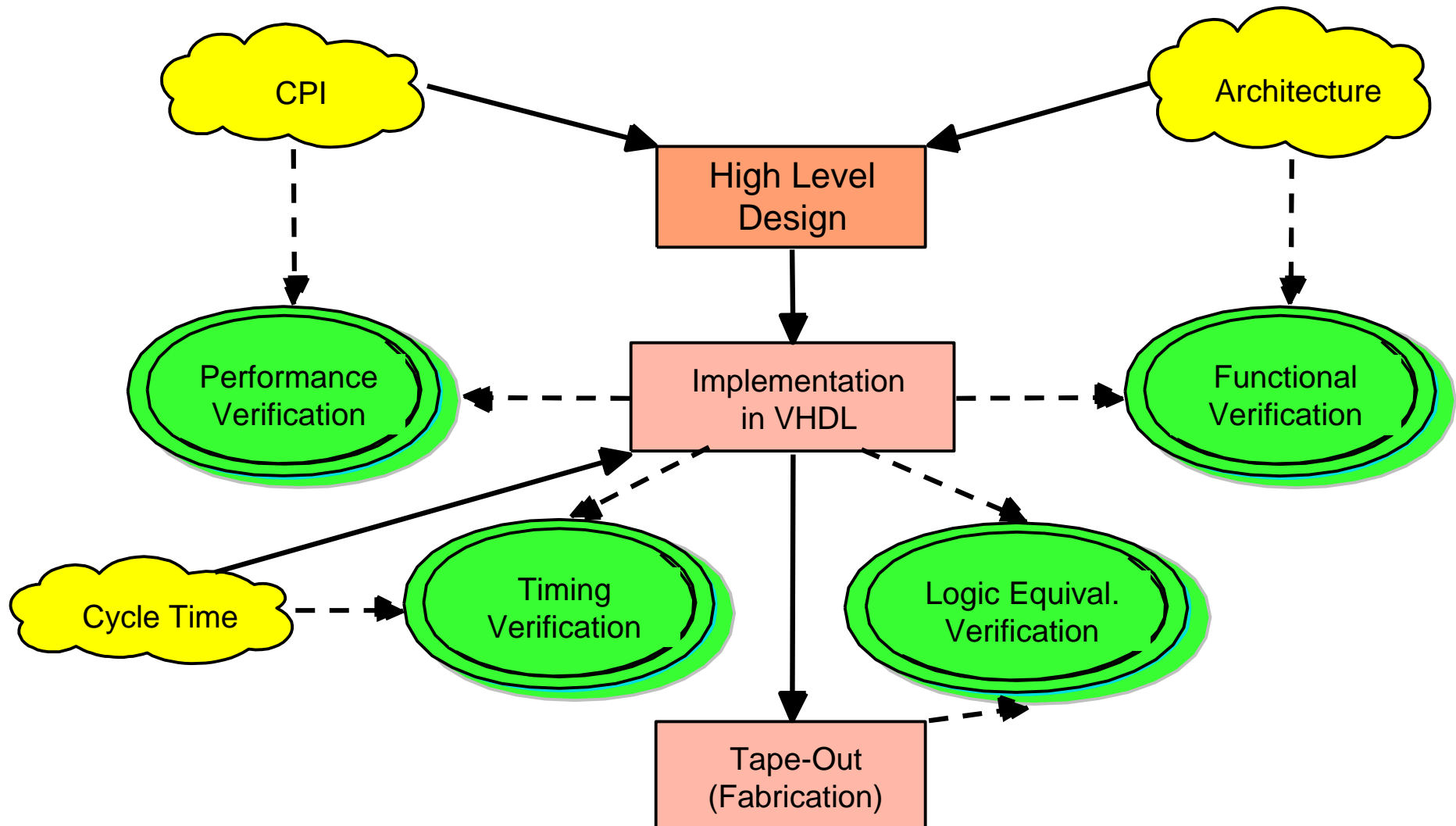


Just a few numbers to illustrate problem

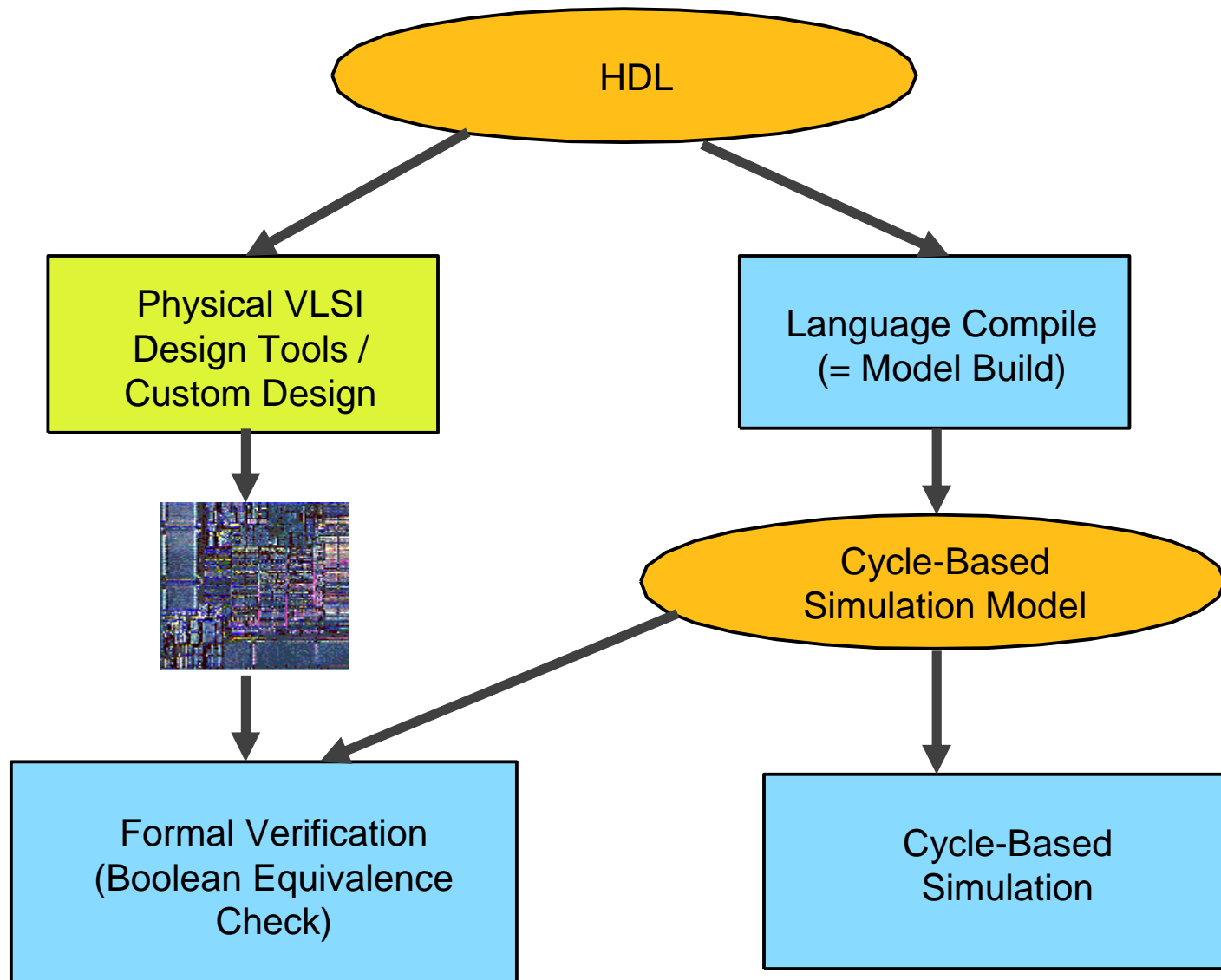
- Example PowerPC 4 (Typical Microprocessor)
 - The number of unique states are $10^{4200000}$
 - If each state were a golf ball,
the universe would be too small by a long shot to hold them.
- Available compute resources
 - Computing grid available, that sustained can simulate more than 7 billion cycles per week
 - Usage of 5-10 terabytes of data on a distributed file system
 - Peak computing power of the grid is approx 2+ TFLOPS
 - Would rank in the top 15 largest supercomputers in the 6/2002 www.top500.org list
- Batch processing software
 - Tools to efficiently use the hardware and submit hundreds of thousands of tests per day
 - Tools to manage the large amount of information generated by the simulations
- Resulting Simulation result
 - Usage of all available compute resources for a couple of month
 - Despite this effort overall simulation accounts for only about 5 min. of execution time on real hardware



What is Verification ?

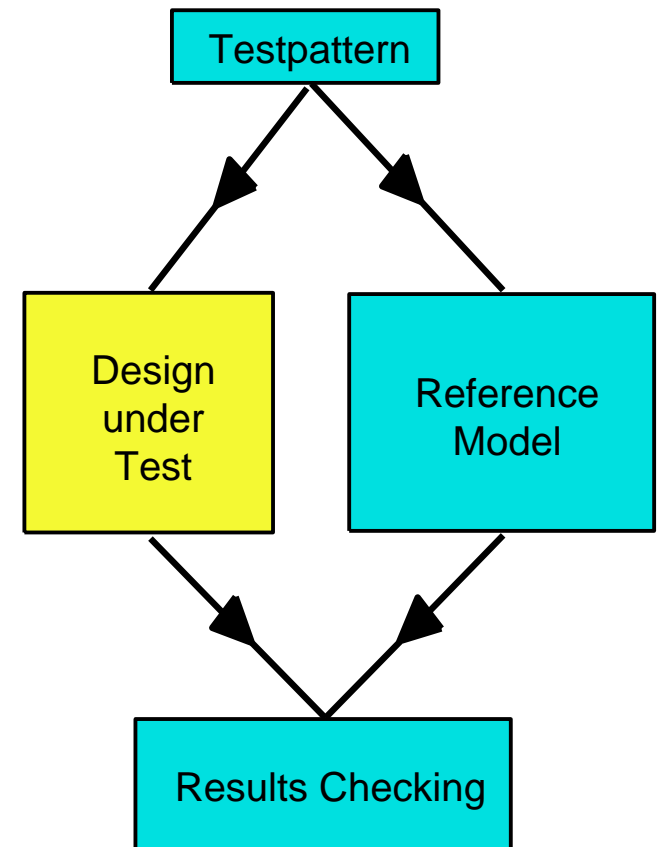


Methodology Flow



Functional Verification

- Also called:
 - Simulation
 - Logic Verification
- Verification is based on
 - Testpattern Generation
 - Reference Model Development
 - Result Checking



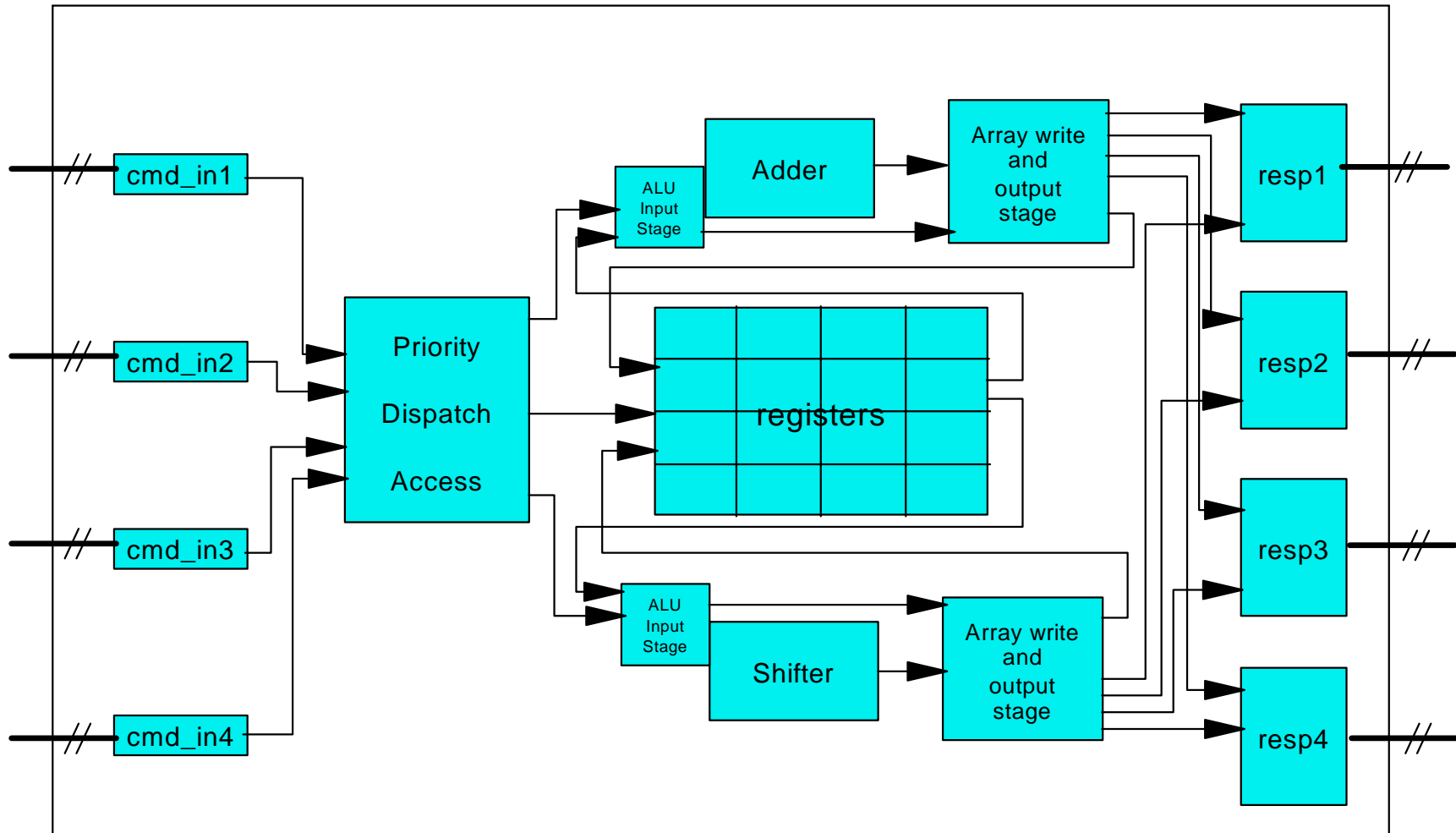
Is perfect verification possible?

- To *fully* verify a design it has to be shown that the logic works correctly for all combinations of inputs applied at every possible internal state.
- This entails:
 - Driving all permutations on the input lines
 - Checking for proper results in all cases
 - Ensure that all internal states have been reached
- Conclusion: Full verification is not practical on meaningful pieces of designs



Design

Design Example - Calculator



This design will be used throughout the course as an example



Specification: Calculator Design (1)

- Calculator has 4 functions:
 - Add
 - Subtract
 - Shift left
 - Shift right
- Calculator can handle 4 requests in parallel
 - All 4 requestors use separate input signals
 - All requestors have equal priority
- Calculator priority logic
 - Priority logic works on first come first serve algorithm
 - Priority logic allows for 1 add or subtract at a time and one shift operation at a time

Specification: Calculator Design (2)

- Input commands:
 - 0 - No-op
 - 1 - Add operand1 and operand2
 - 2 - Subtract operand2 from operand1
 - 5 - Shift left operand1 by operand2 places
 - 6 - Shift right operand1 by operand2 places
- Input Data
 - Operand1 data arrives with command
 - Operand2 data arrives on the following cycle
- Output response line definition
 - 0 - no response
 - 1 - successful operation completion
 - 2 - invalid command or overflow/underflow error
 - 3 - Internal error
- Output Data
 - Valid result data on output lines accompanies response (same cycle)



Design Content: Mainline Functions

- Functions, that are required to fulfill the expectations on a design
 - The reason why customers buy these things
- Typical functions are
 - Instruction execution in a microprocessor
 - Memory Access for a storage controller
 - Cache access
 - Transaction processing using
 - certain Protocols like PCI or other DMA operations

Design Content: Pervasive Functions

- Functions required to service and maintain the system
 - Mostly invisible for the customer
- Typical functions are
 - On chip tracing capabilities for chip debug
 - Chip initialization functions like
 - Power-on Reset
 - BIST (Built-In Self Test)
 - Error detection and error recovery
 - Synonym: Bad machine path
 - Power saving modes



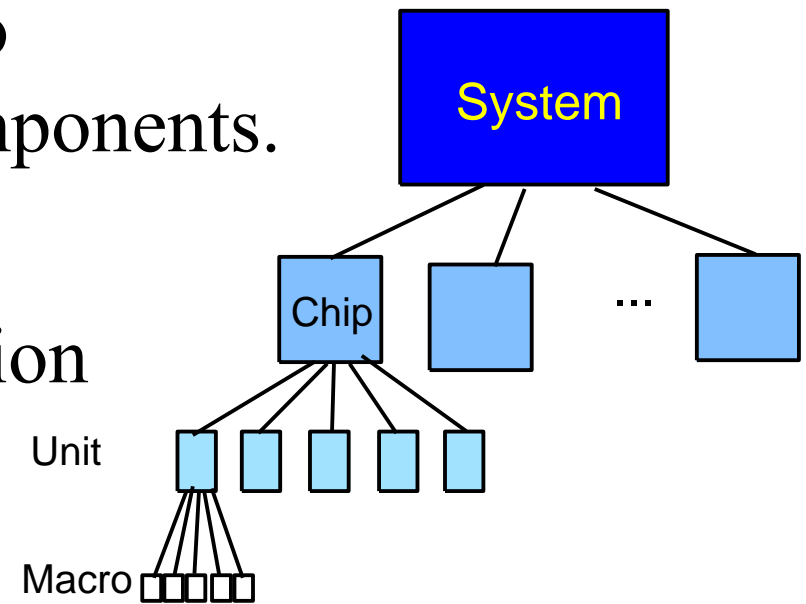
Design Content: Firmware

- A lot of functionality is not provided by hardware only but adding software to the final product
 - This software is also called firmware, as it is a firm part of the final product
 - This software is usually hidden from the user
 - The software has a lot of interaction with the hardware which needs to be verified as well
 - A typical real world example is the BIOS of every PC



Design Hierarchy

- Breaks system design down into logical and comprehensible components.
- Enable repeatable components
- Use Hierarchy also for verification
- Important especially for large and distributed teams

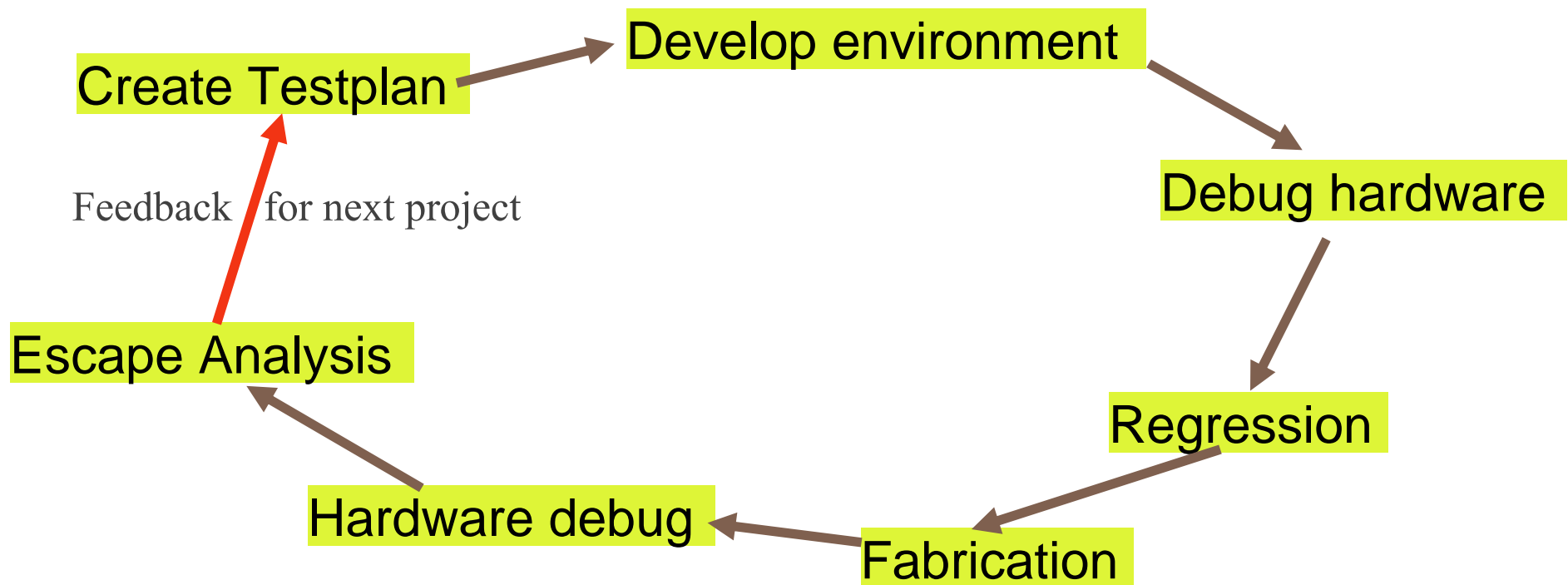


- IBM eServer developed in multiple sites with many design groups and a total of more than 100 people just in hardware verification



Verification

The Verification Cycle



Verification Testplan

- Verification leaders create a testplan with input from the design leaders
- The testplan includes (per hierarchy level)
 - Schedule
 - Input criteria
 - Methodology and required tools
 - Description of test procedures
 - Completion criteria
 - Discussion on expected verification coverage
 - What problem types can or cannot be found



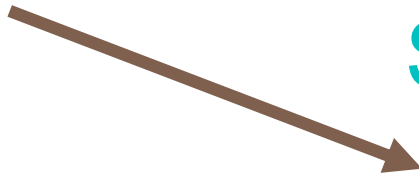
Formal Verification vs. Simulation

n If the overall **State space** of a design is the universe, then

Formal verification is like a bulb and



Simulation is like a laser beam



And both fail to lighten the whole universe !

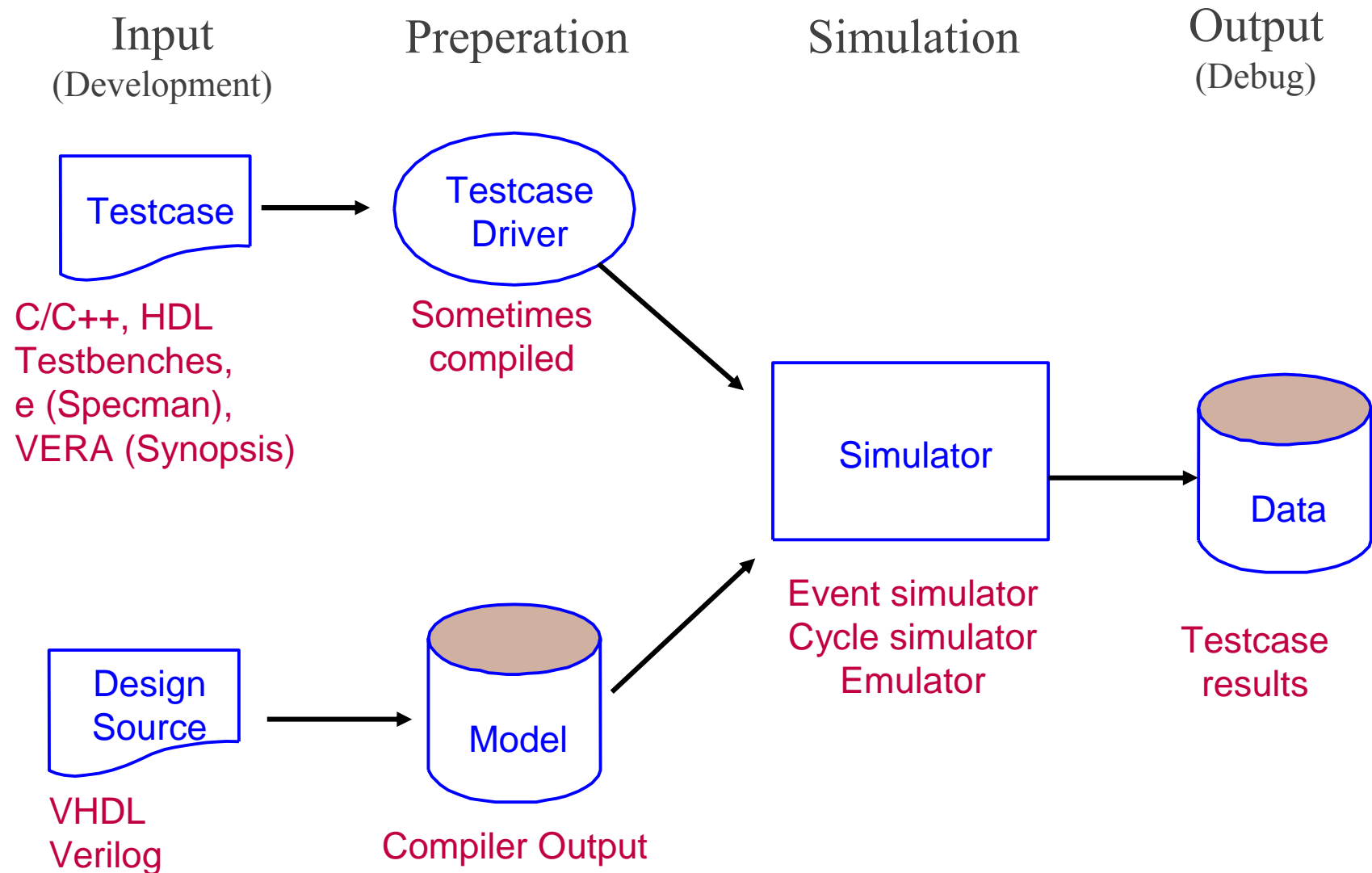


Formal Verification

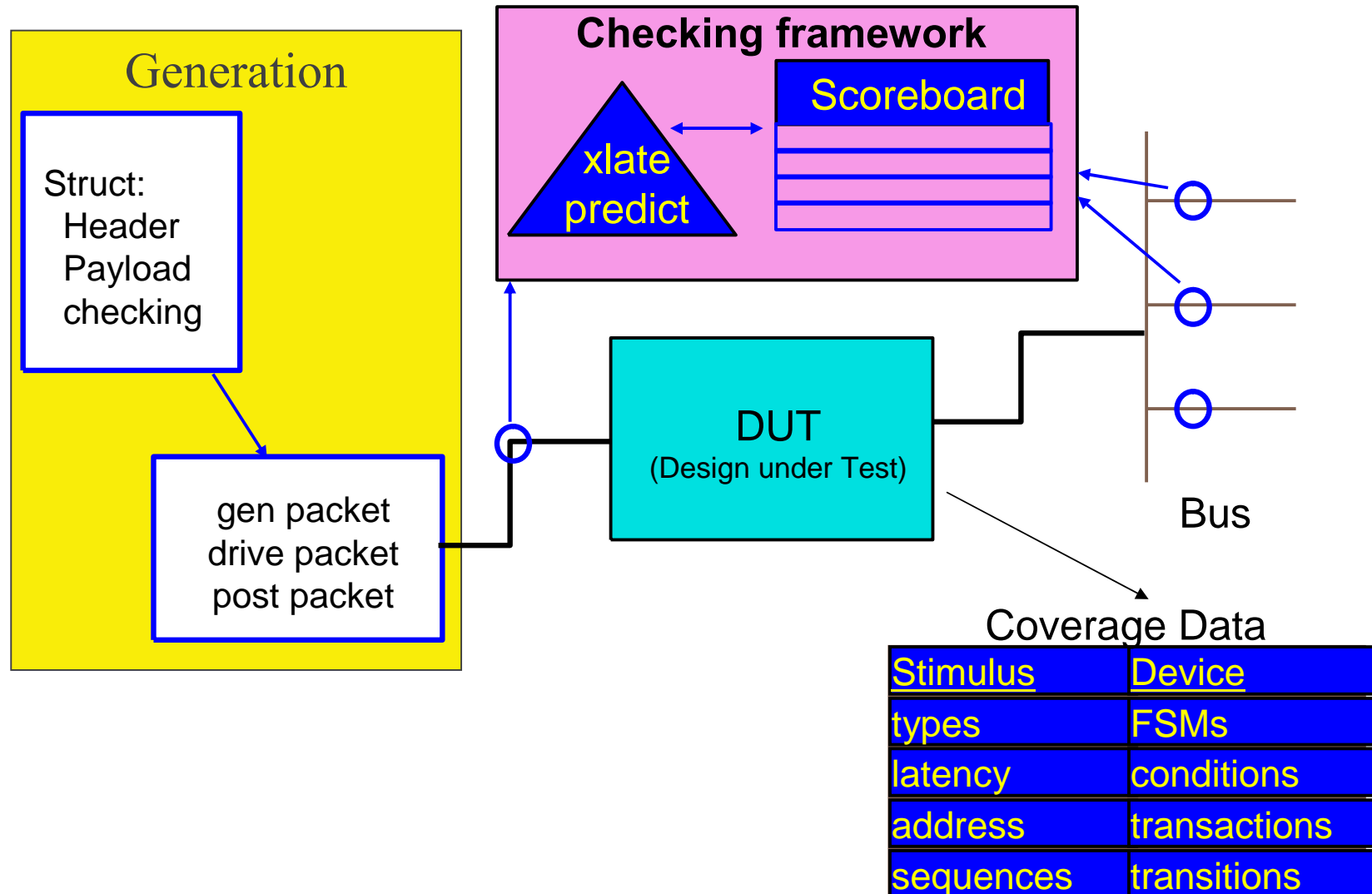
- Formal Verification employs mathematic algorithms to prove correctness or compliance
- Formal applications fall under the following:
 - Equivalence Checking
 - HDL vs. Synthesis output
 - Model Checking
 - Used for logic verification of units of limited size
 - Theorem Proving
 - Up to full chip verification, but no industrial usage today
 - Symbolic Trajectory Analysis (STE)
 - Often used for array designs



Simulation Process



Simulation Environment



Black Box Method



- The black box has inputs, outputs, and performs some function.
- The function may be well documented...or not.
- To verify a black box, you need to understand the function and be able to predict the outputs based on the inputs.
- The black box can be a full system, a chip, a unit of a chip, or a single macro.

White and Grey Box Method

- White box verification means that all internal facilities are visible and utilized by the testcase driver.
- Grey box verification means that a limited number of facilities are utilized in a mostly black-box environment.
 - Used in most environments!
 - Prediction of correct results on the output interface is occasionally impossible without viewing and internal signal.

Coverage Goals

- n Measure the "quality" of a set of tests
- n Supplement test specifications by pointing to untested areas
- n Help create regression suites
- n Provide a stopping criteria for unit testing



In addition:

Always leads to a better overall understanding of the design



Hardware Firmware Co-Simulation

- Usage of HW Emulator
 - Model parallelization
 - Large investment
- Environment
 - Performance critical
 - only minimal interactions between model in emulator and testcase environment allowed
 - Software reuse from other environments
 - Complex project management



Worldwide largest emulation system

Technology: ~ 250000 Processor
Speed: 200000 cycle/sec
Capacity: max. 60 Mio.Gates



Escape Analysis

- Escape analysis is a critical part of the verification process
- Important data:
 - Fully understand bug! Reproduce in sim if possible
 - Lack of repro means fix cannot be verified
 - Could misunderstand the bug
 - Why did the bug escape simulation?
 - Process update to avoid similar escapes in future (plug the hole!)

Verification Guidelines

- Verification requires deep design insight
- Verification needs to anticipate situations that have been missed by the design group
 - Focus on exotic scenarios and situations
 - e.g all queues full, while design is done in a way to avoid any buffer full conditions
 - Focus on multiple events at the same time
 - Stress functions that are not explicitly forbidden



Trends

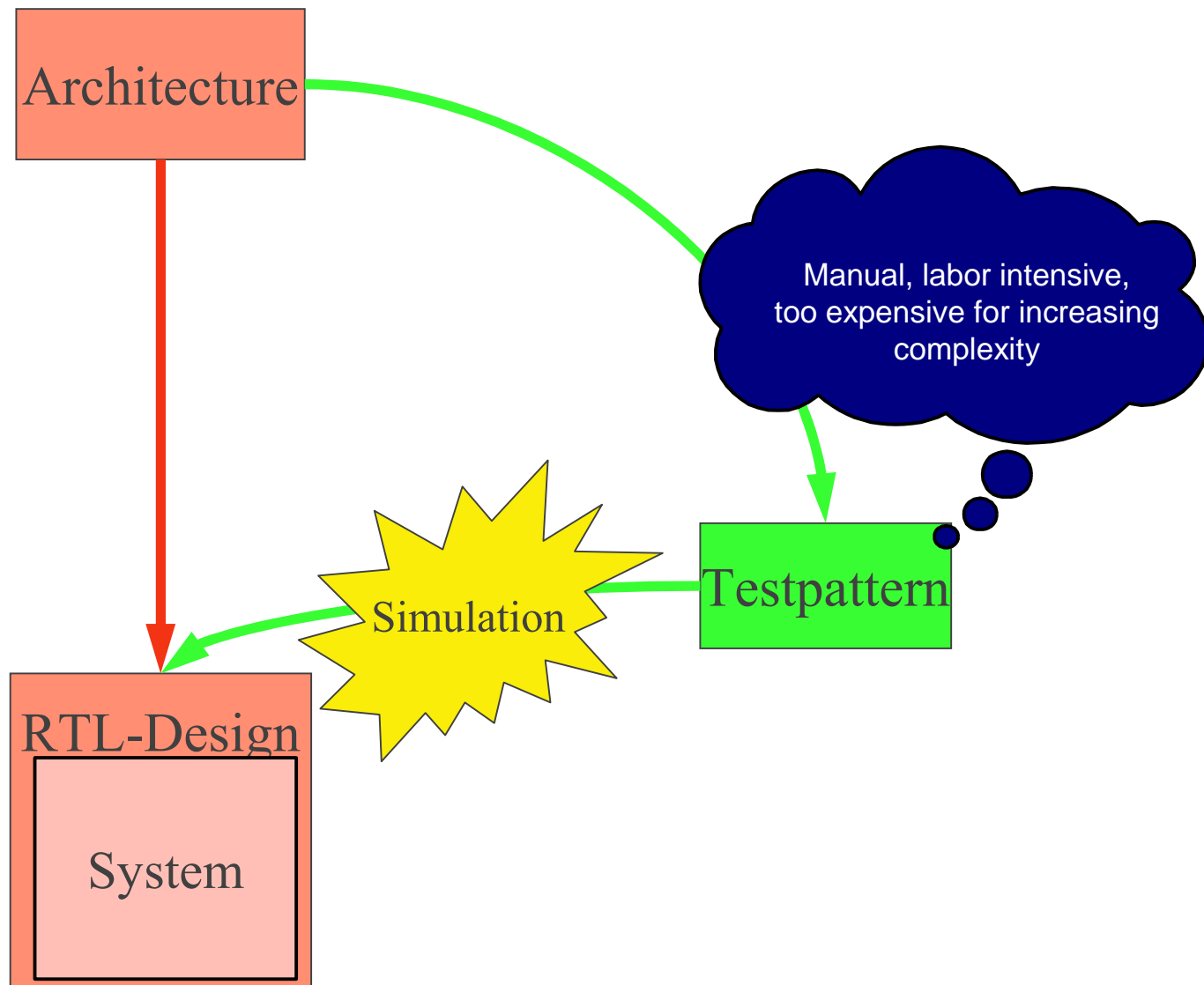
Challenges in Verification

- Increasing Complexity
- Increasing Design sizes
- Leading to exploding State spaces
- Increasing number of functions
- ... but ...
- Reduced timeframe
- Reduced development budget

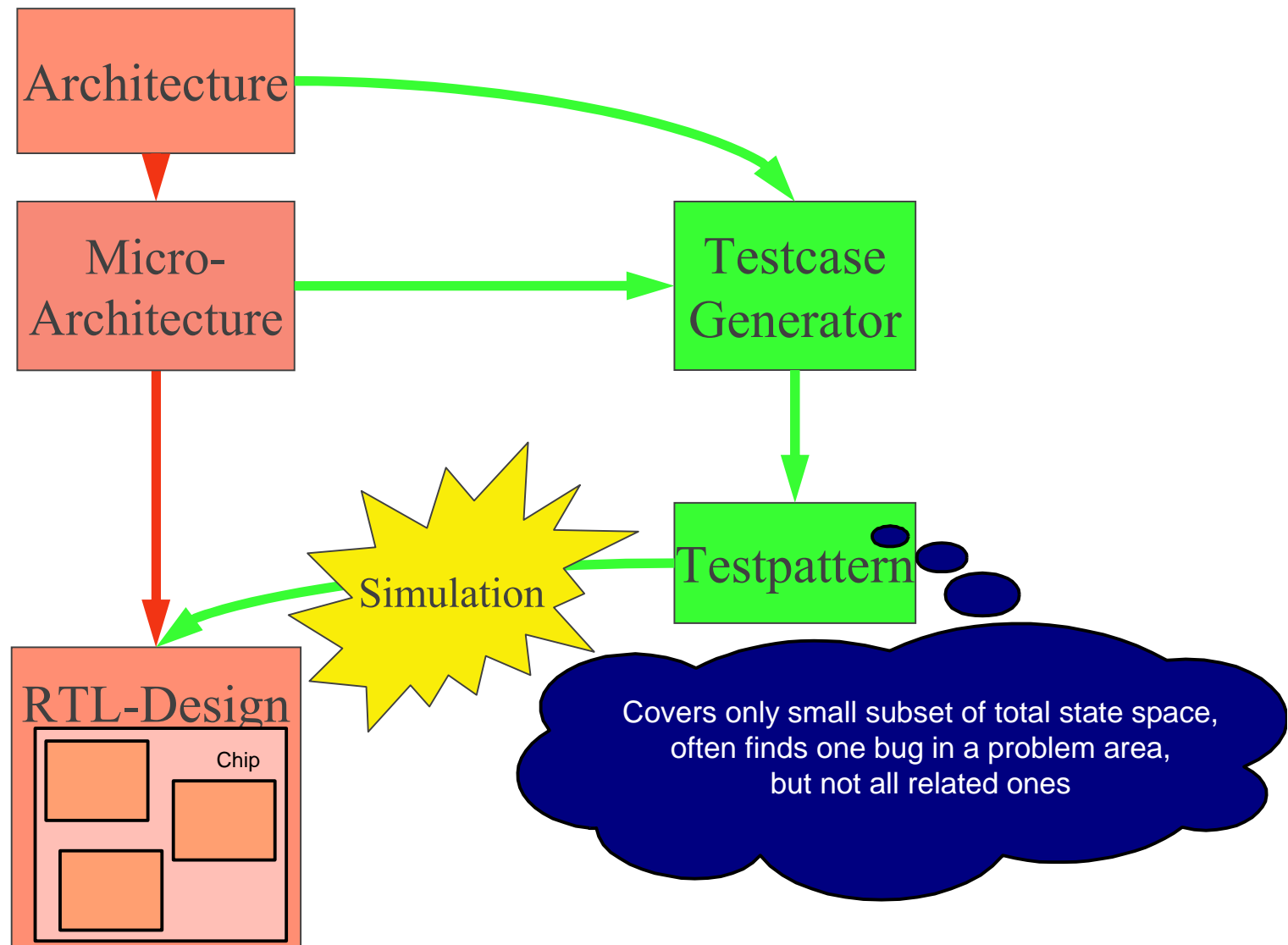
Explains the history and the future



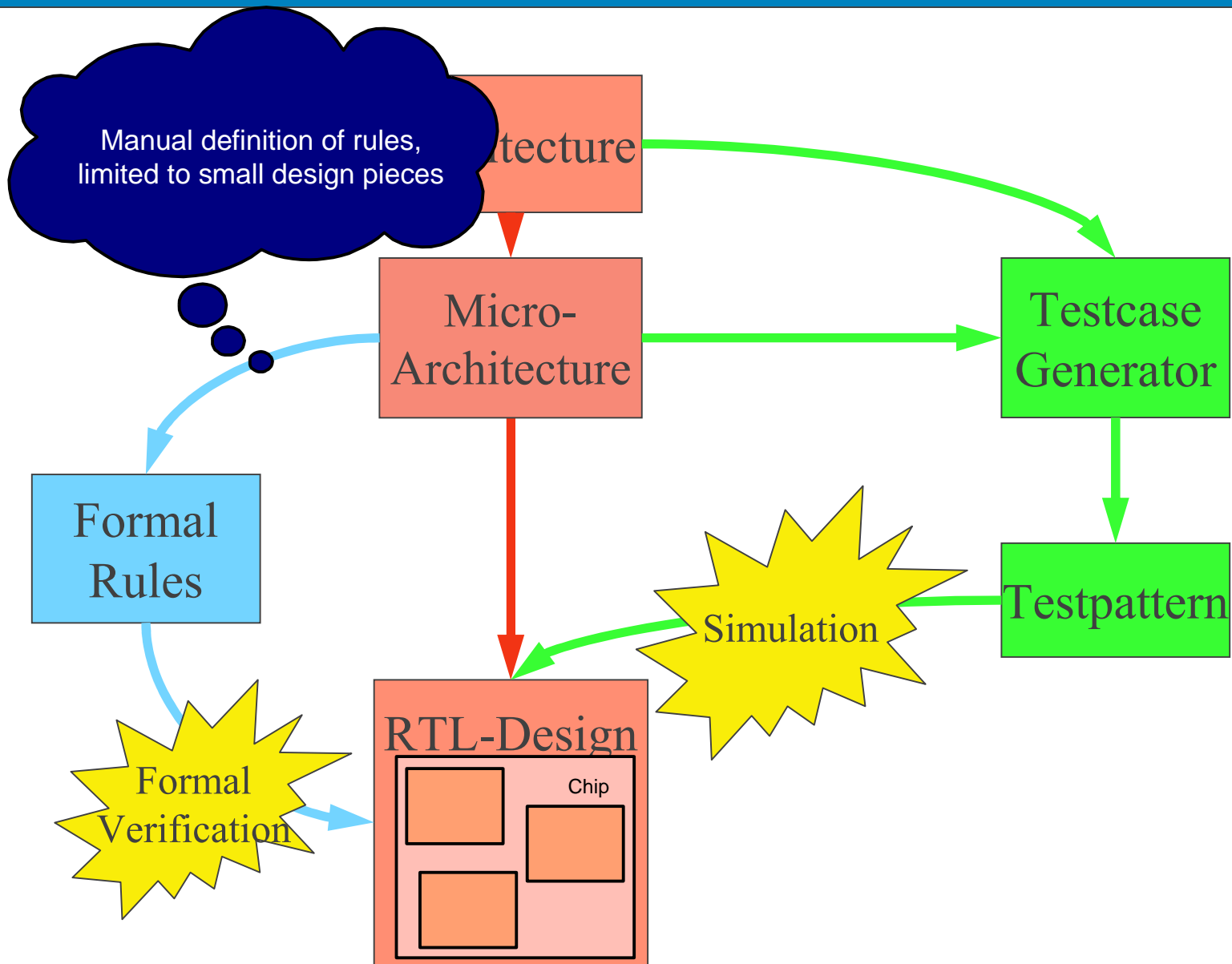
Evolution of Functional Verification



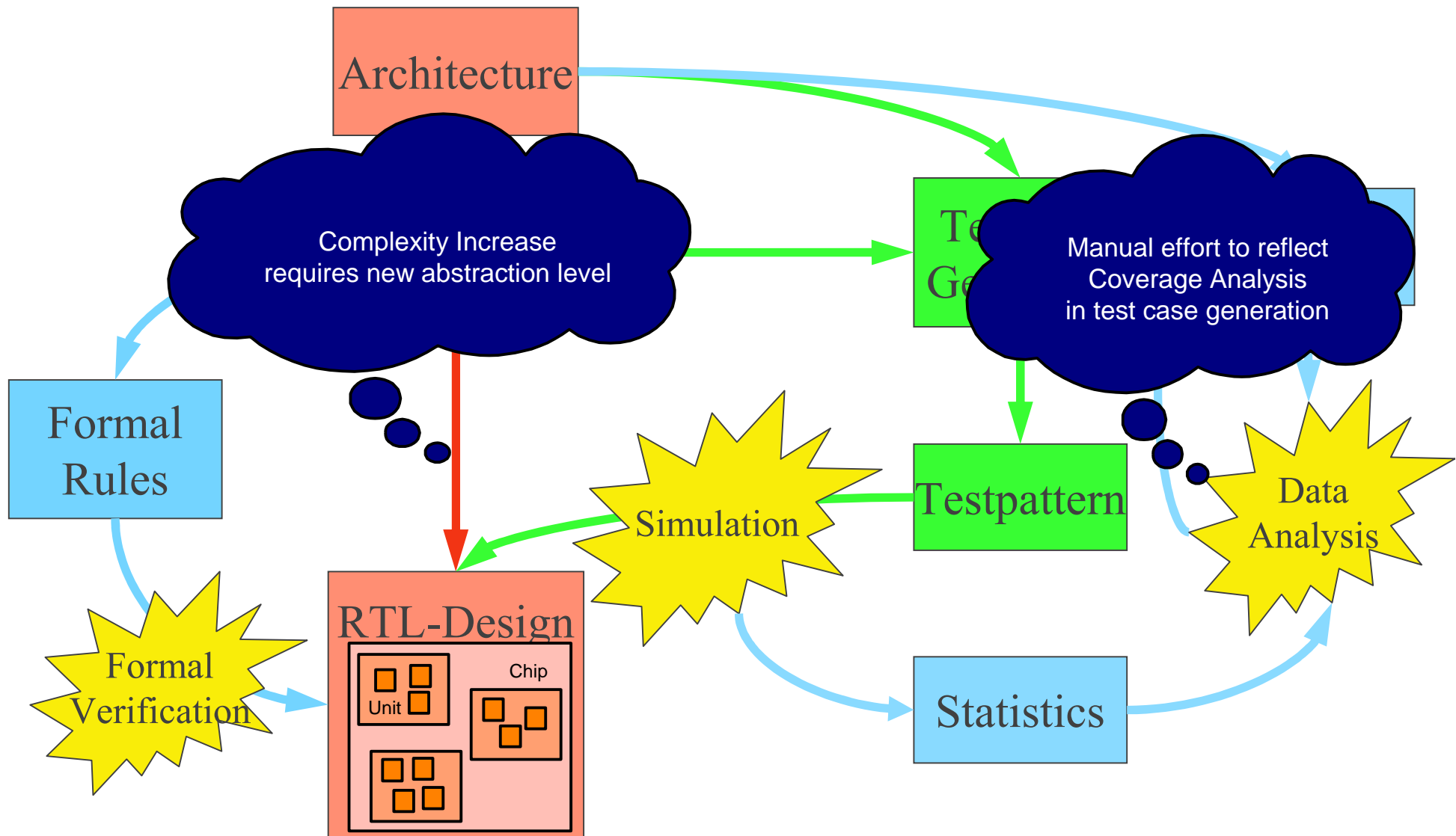
Evolution of Functional Verification



Evolution of Functional Verification



Evolution of Functional Verification



New Areas

- Coverage driven testcase generation
 - Automated feedback into simulation
 - Currently research topic
- Semi-Formal tools
 - Combination of formal verification and simulation
- Waiting for next design abstraction
 - Could enable next set of tools



Evolution of Problem Debug

Analysis of simulation results (no tools support)

Interactive observation of model facilities

Tracing of certain model facilities

Trace post processing to reduce amount of data

On the fly checking by writing programs

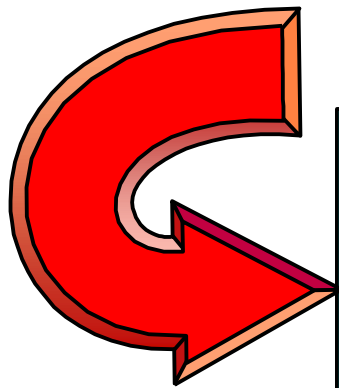
Intelligent agents, knowledge based systems

:

t



And at the end ...



At the end the verification engineer understands the design better than anybody else !



Next Sessions

- Session 1: 30.04.2003 Introduction
- **Session 2: 14.05.2003 Modelbuild, Simulators**
- Session 3: 21.05.2003 Testcase Generation
- Session 4: 04.06.2003 Random Simulation
- Session 5: 11.06.2003 Coverage, Formal Verification
- Session 6: 25.06.2003 Pervasive Function
- Session 7: 09.07.2003 Hardware/Software Co-Simulation
- Session 8: 23.07.2003 Infrastructure, Testplan, Project schedule

