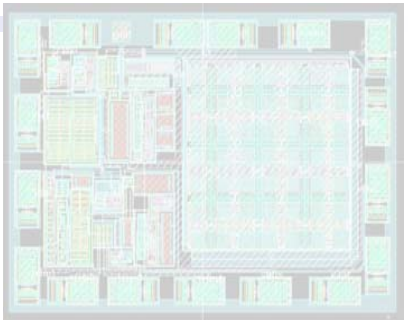
Introduction to Formal Verification



Presented by:

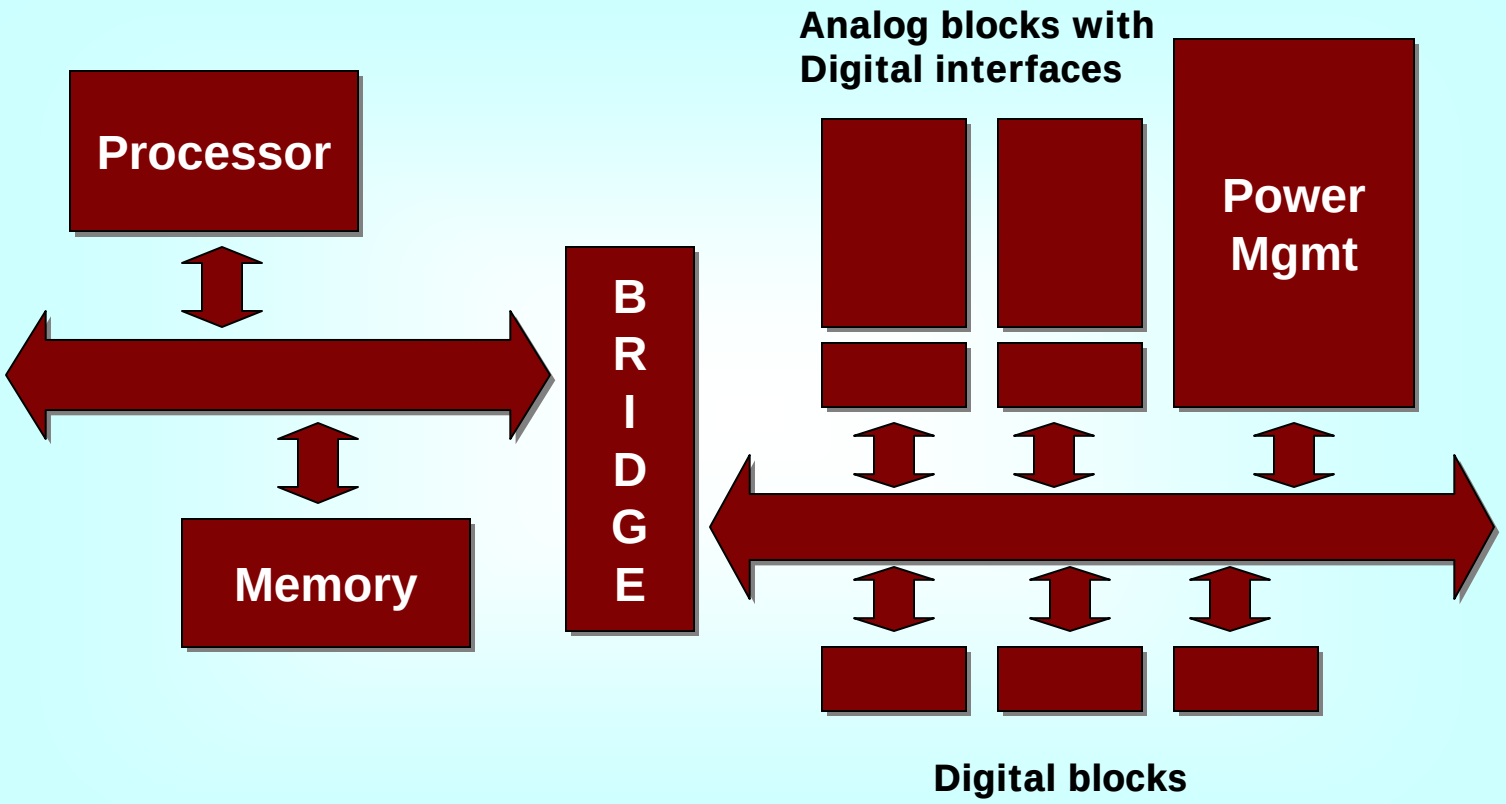
P. P. Chakrabarti

Dept. of Computer Sc. & Engg.,

& Advanced VLSI Design Laboratory

Indian Institute of Technology Kharagpur

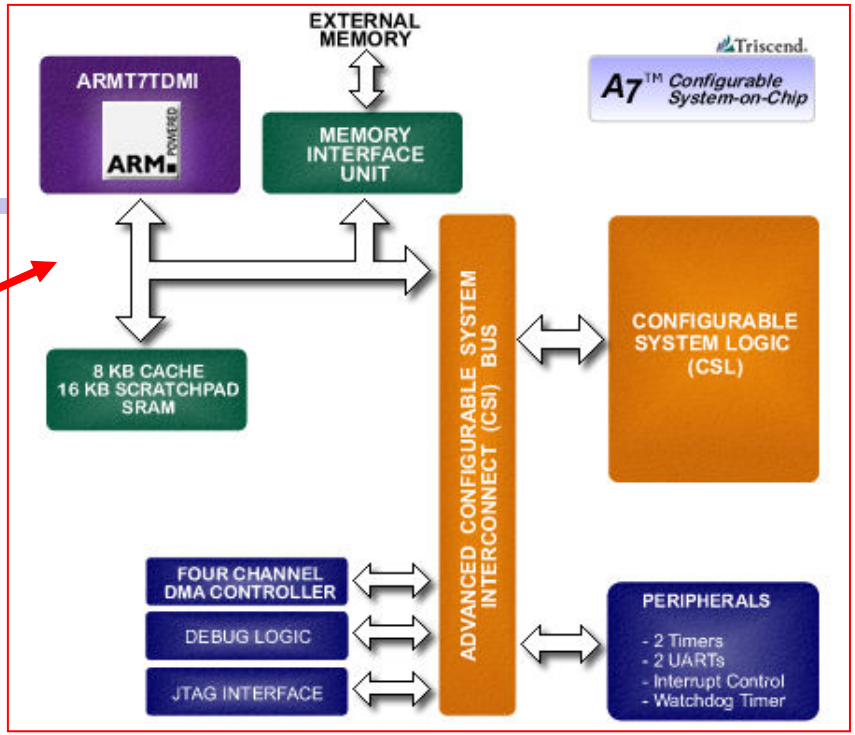
Typical SOC Architecture

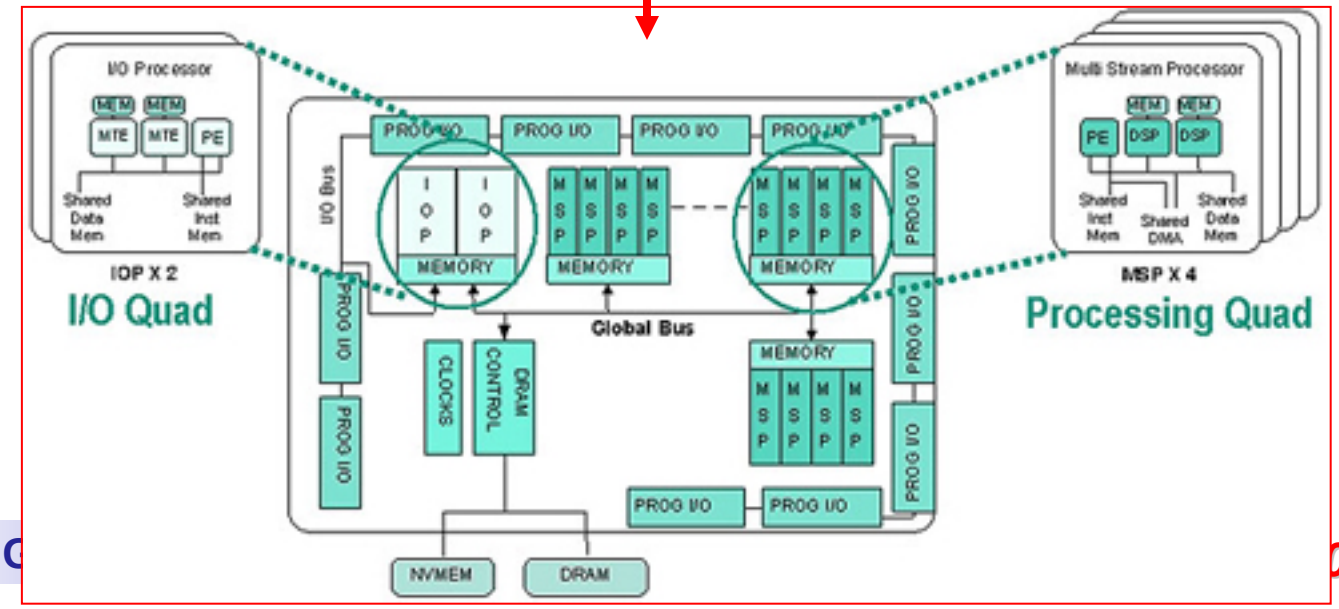


Two Examples

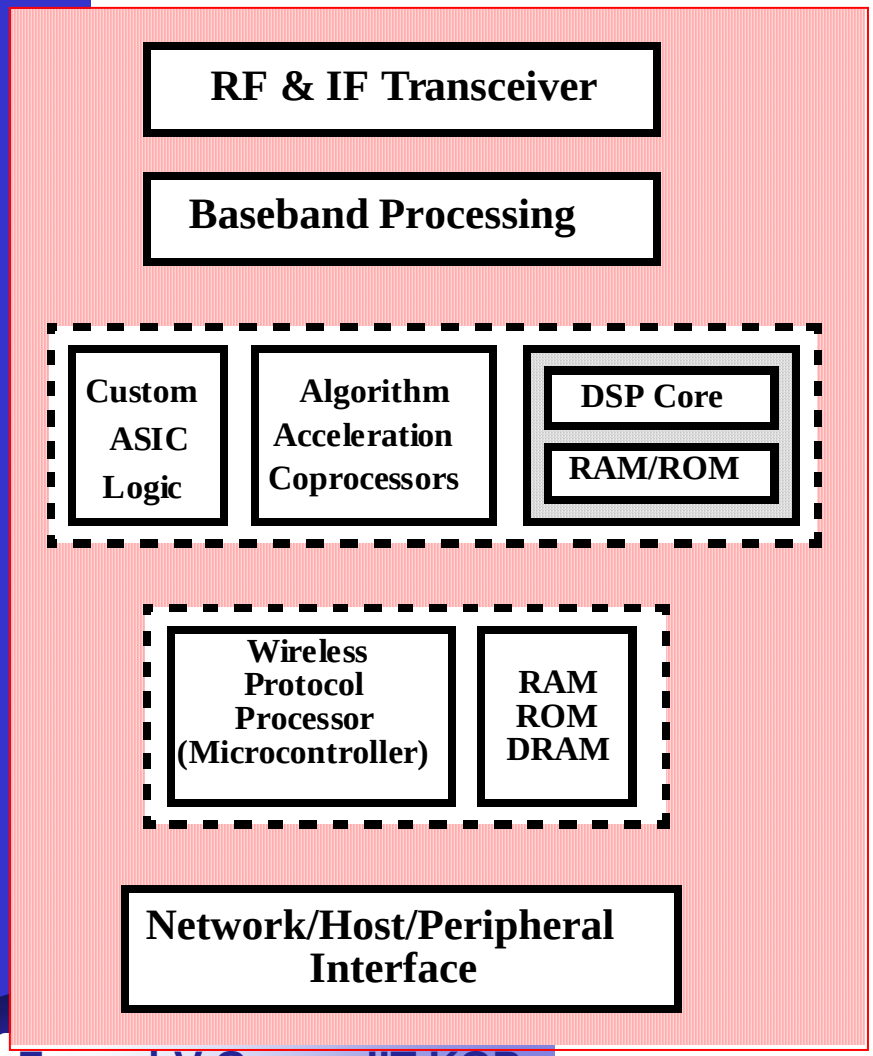
Low-power, low-cost embedded systems

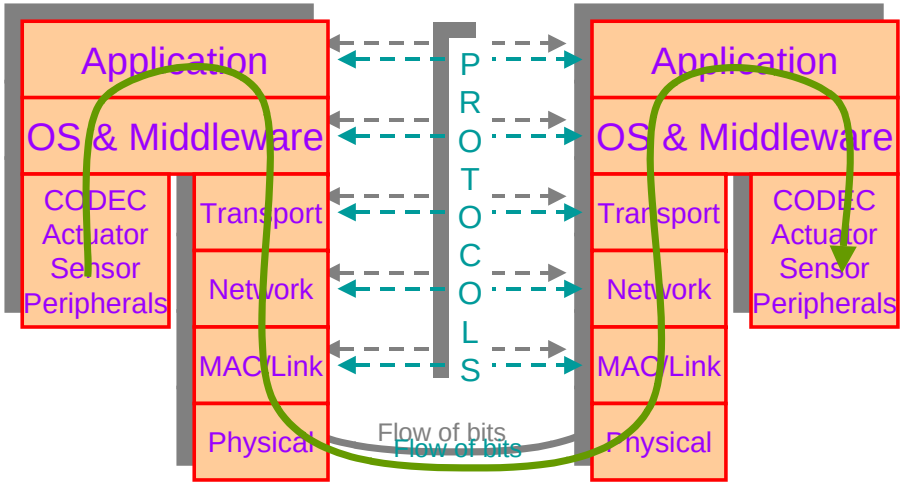
- **High-performance multiprocessors for complete applications**





Networked Embedded Appliance

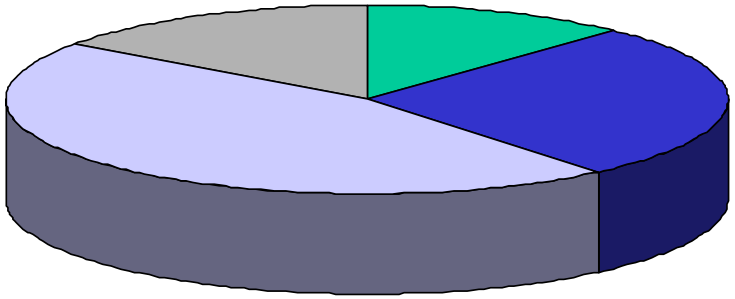




The challenge is to make a Computer and a Radio (Transmitter& Receiver on a Single Chip)

We need to bridge the Analog & Digital Design Divide at Nanometer scale

Verification Dominates Design

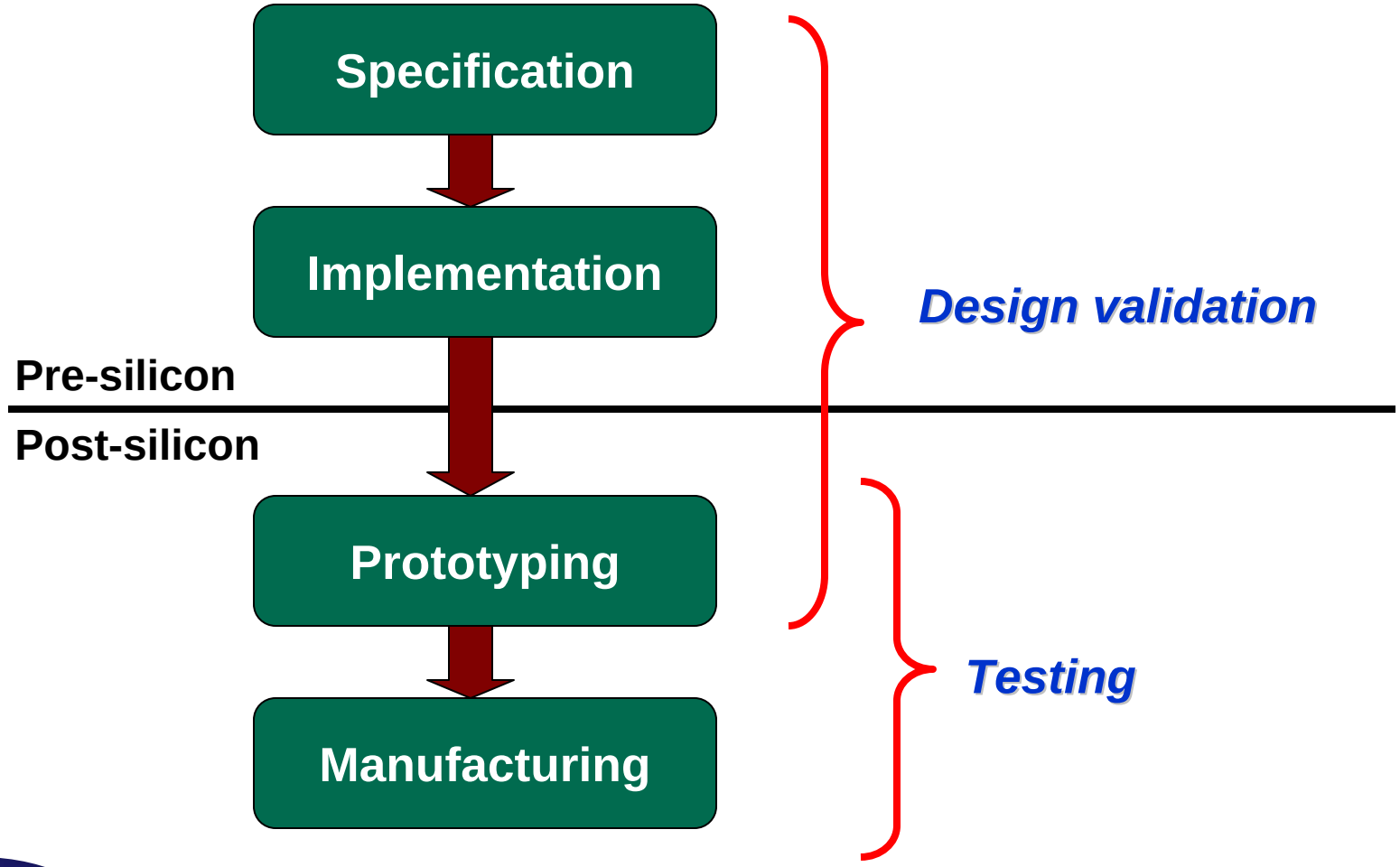


- Synthesis
- Timing analysis
- Equivalence checking
- DFT

- High-level design
- RTL coding
- Block-level simulation

- Behavioral modeling
- Architecture level simulation
- System level simulation

Chip: *From concept to market*



Pre-silicon Validation Tasks

- Specification verification
- System verification
- Performance verification
- Testability verification
- Timing verification
- Silicon verification

Design Levels

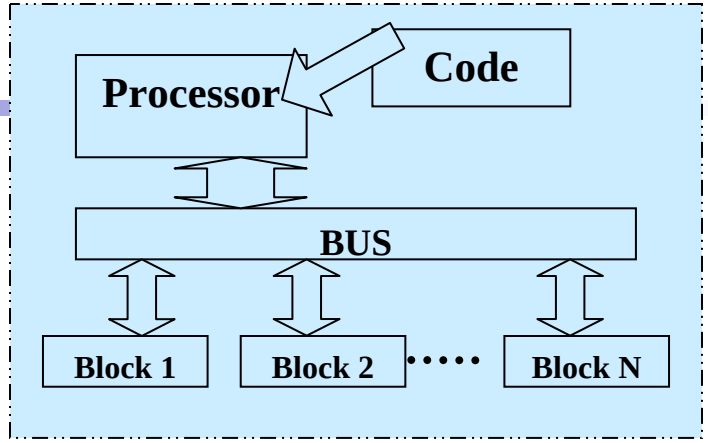
Application Spec

System Architecture

Processor Design

Software Design

Blocks



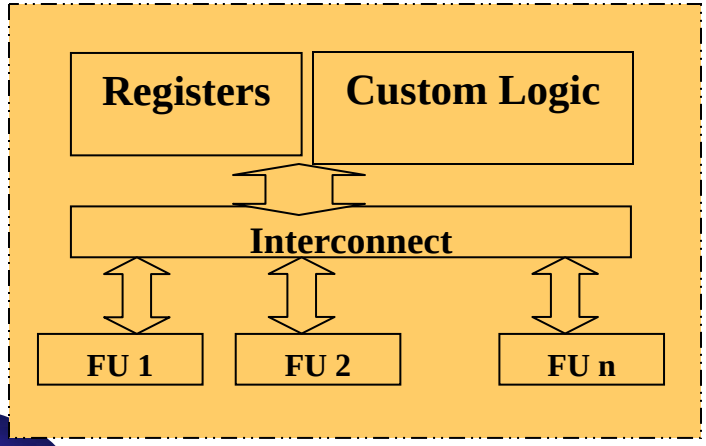
Process/Behavior

Register Transfer

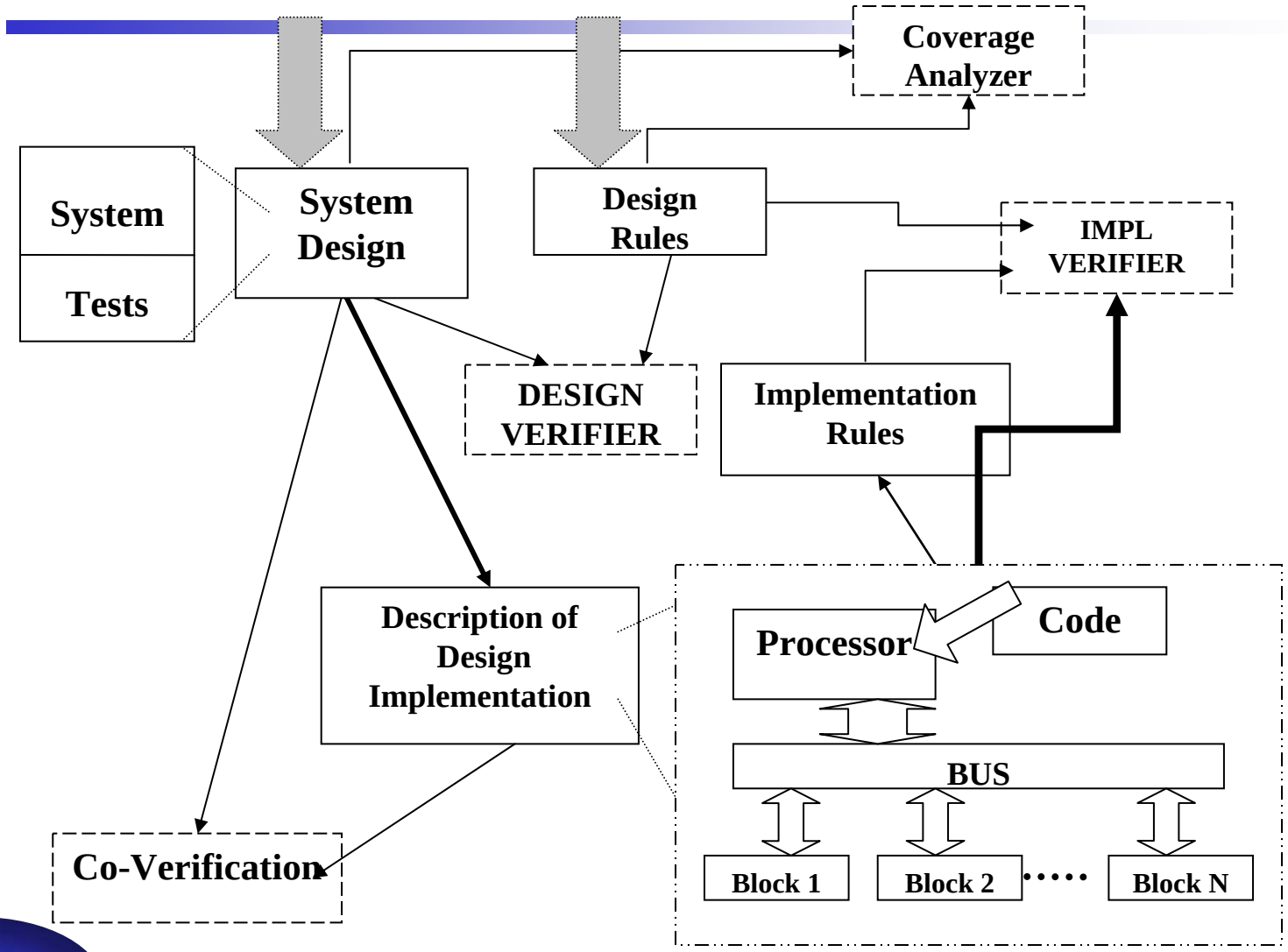
Gate

Transistor/RLC

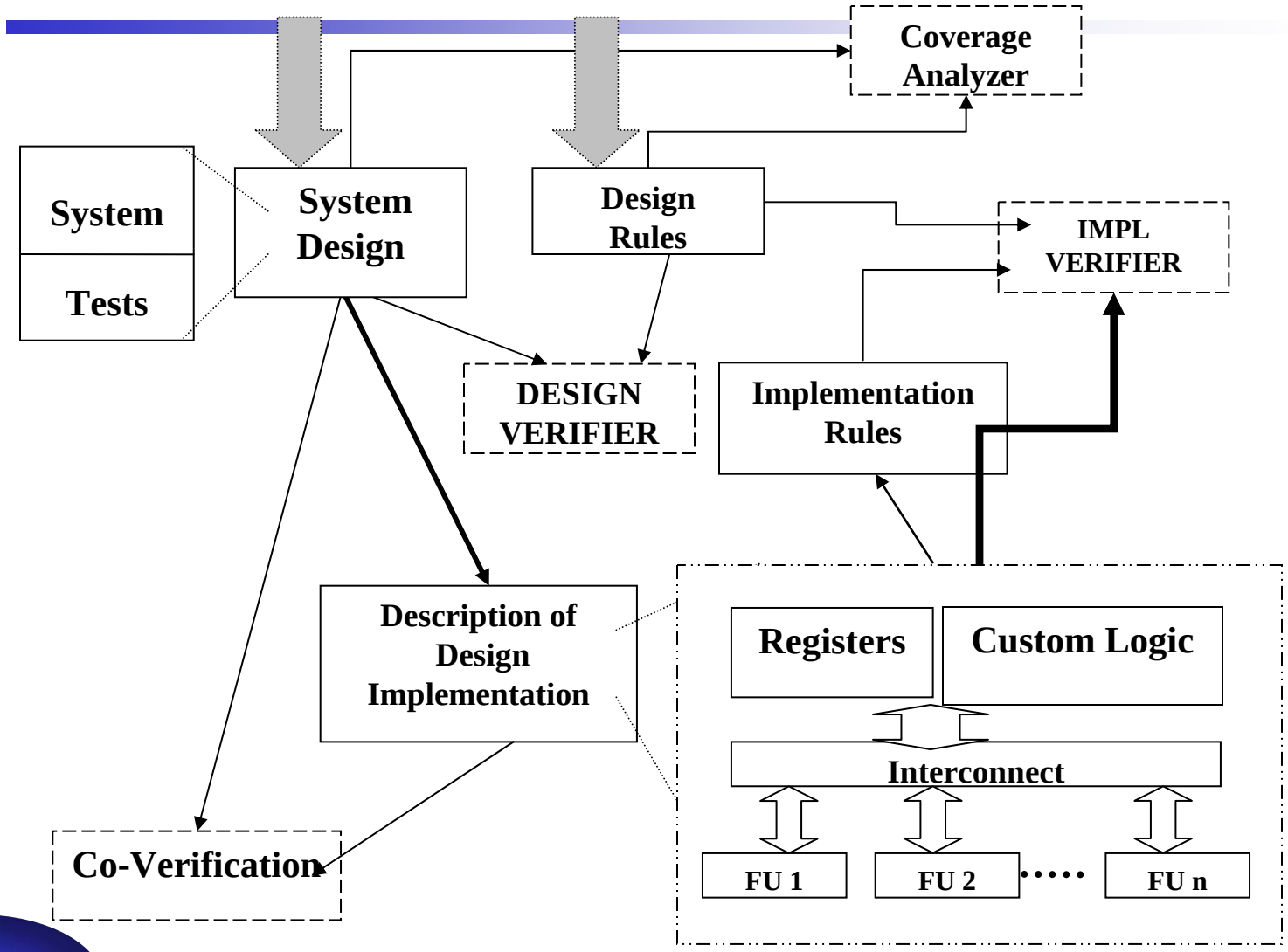
Layout



A Design & Verification Flow



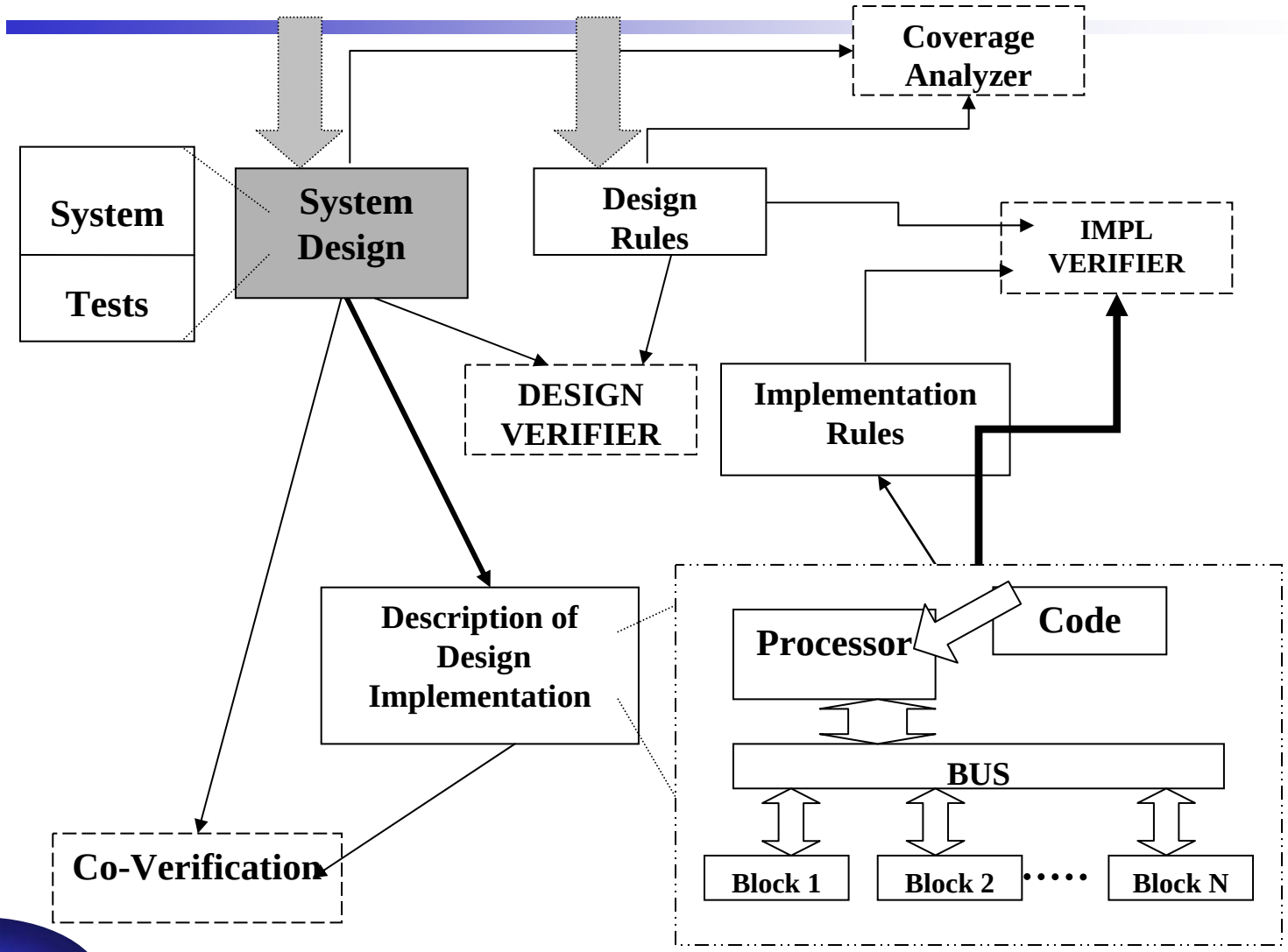
Hardware Verification



Important Tasks

- System-level design validation
  - High-level modelling of CCS (SDL, UML, SystemC)
  - Verification by simulation and assertion checking
- Processor and Toolset Design
  - Flexible Architecture Design & DSE
  - Systems Software Generation
- Designing & Verifying Custom Blocks
  - Architecture Spec and RTL
  - Coverage Analysis
- RTL Validation
  - Functional & Timing Verification
  - Modular Verification

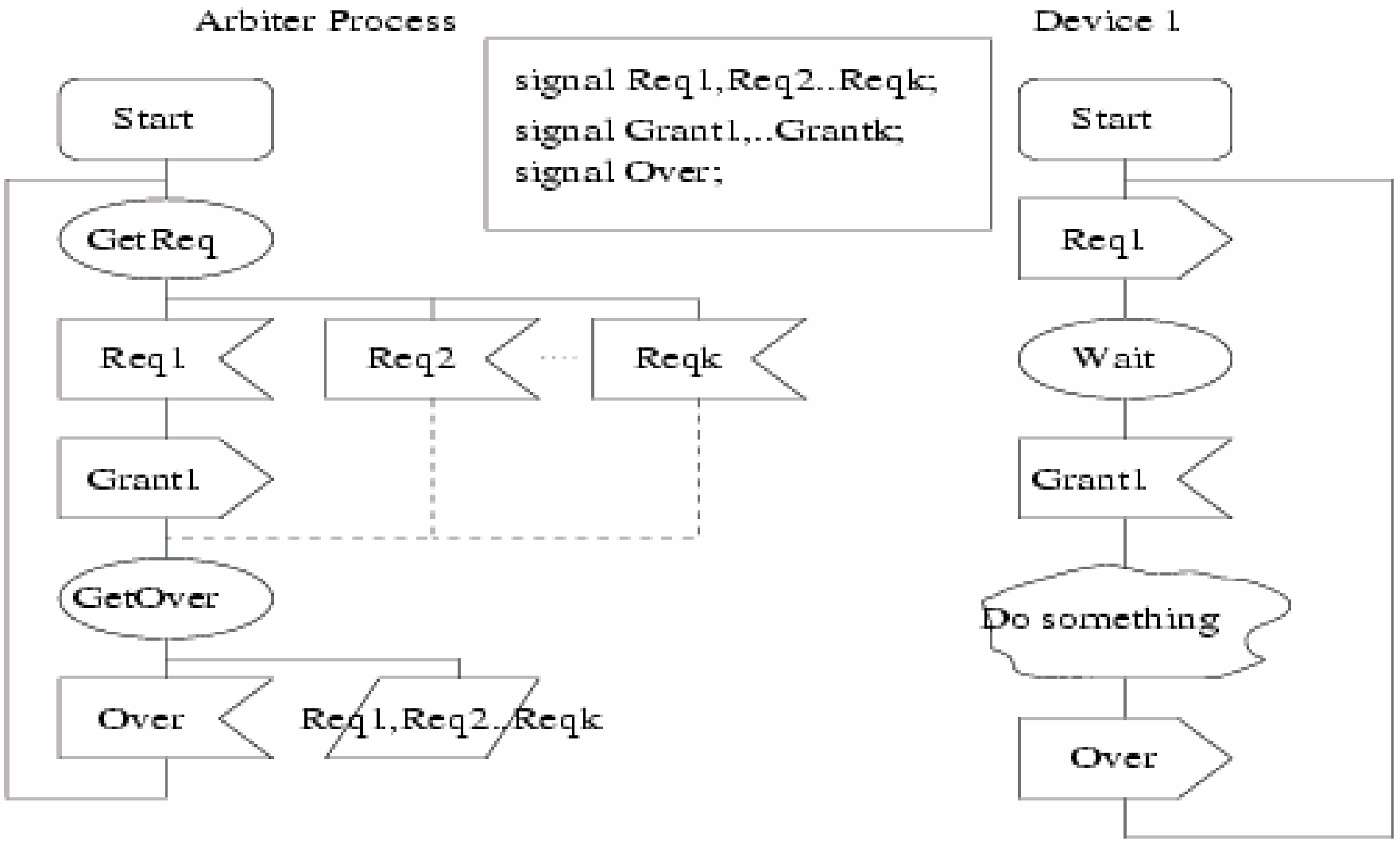
High Level Design



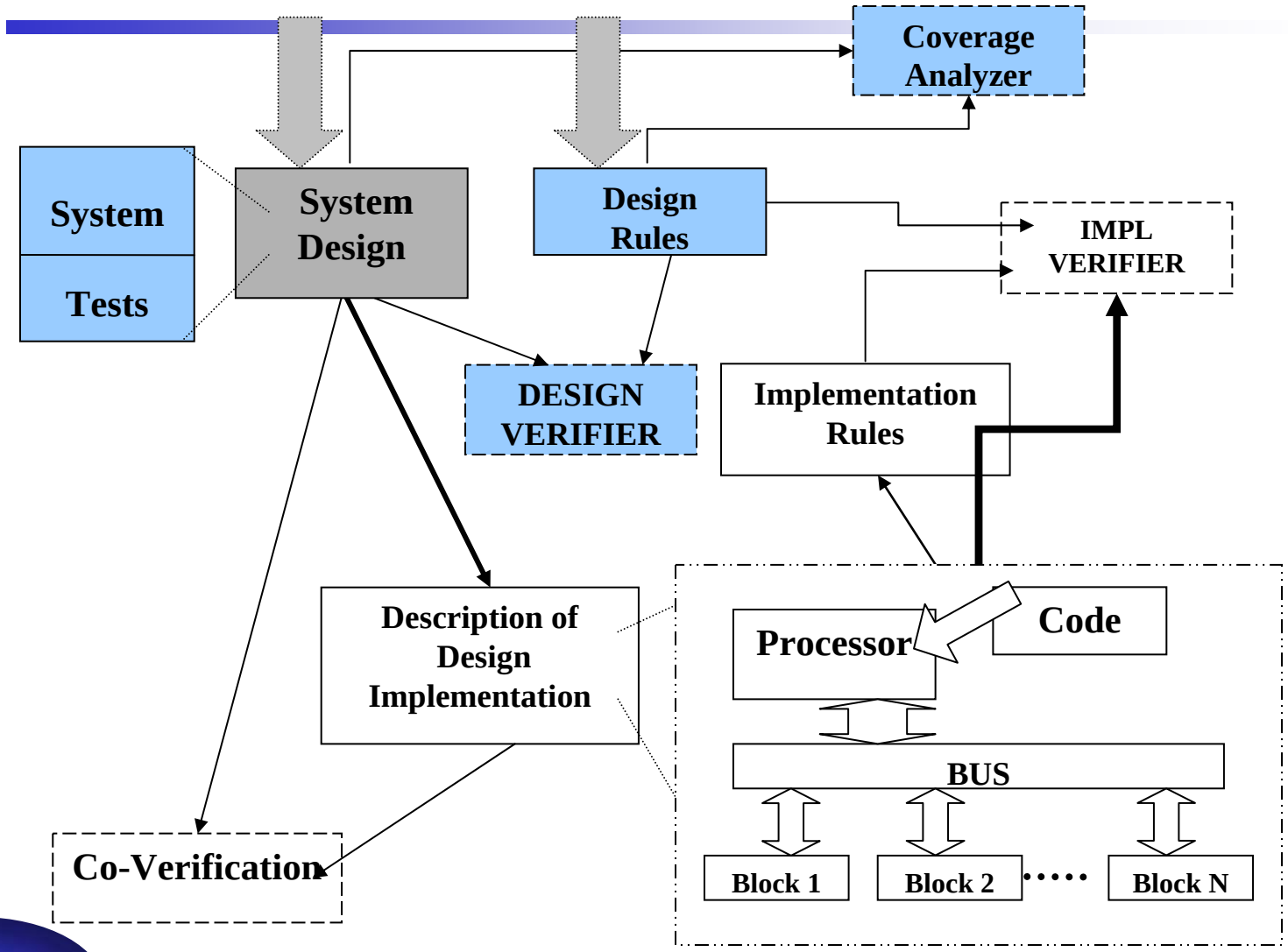
System-Level Design

- Modelling Communicating Concurrent Processes
- Typical Interest ranges from:
  - Interface Verification (handshaking, data-transfer)
  - Complete Cycle Accurate Simulation with processor cores, memories, etc
- Synthesis from High-level specs like SDL to cycle accurate models like SystemC
- Verification by a method of of controlled simulation traces and trace analysis by temporal logic.

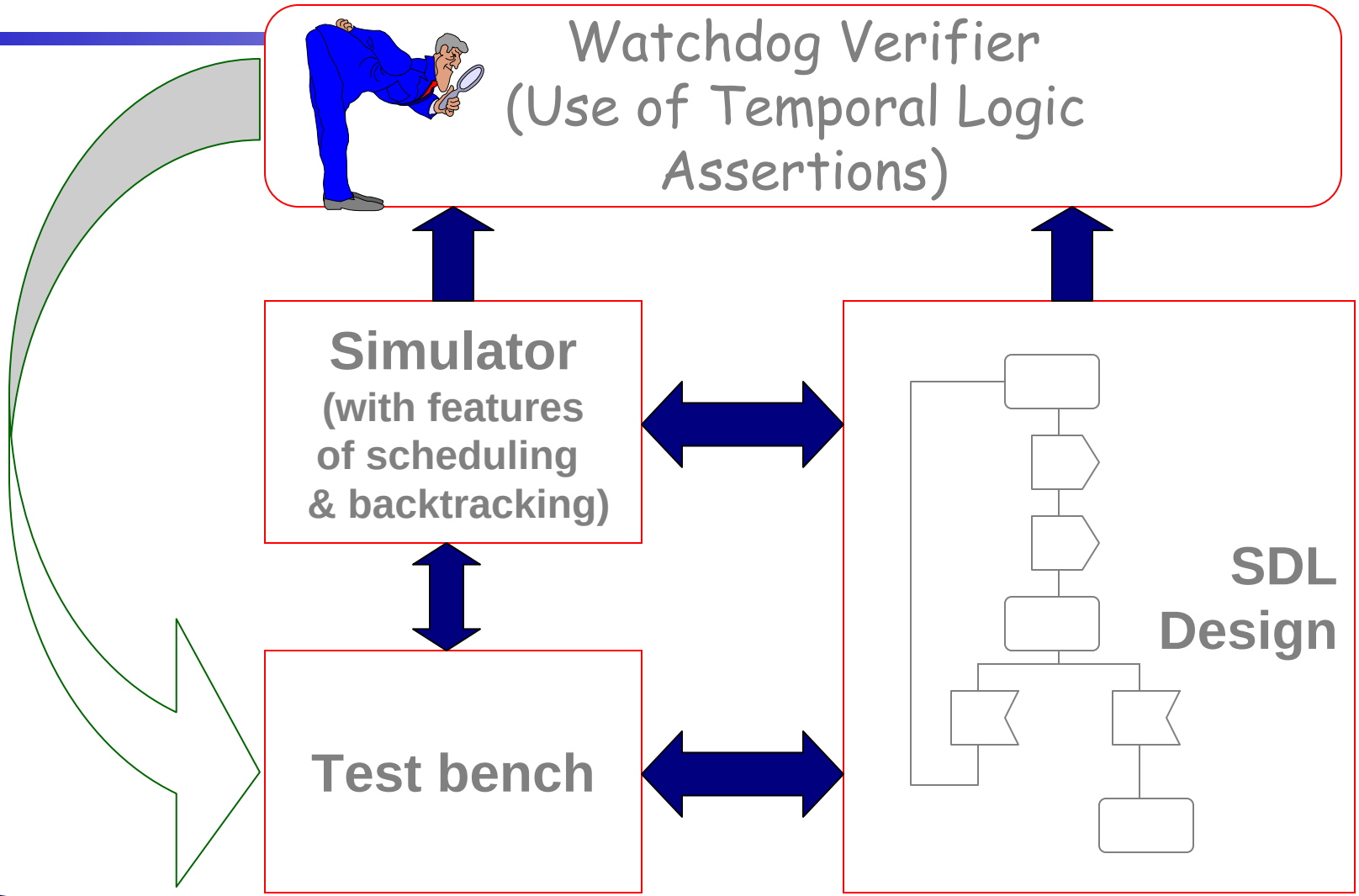
Specification in SDL



Design Intent Verification

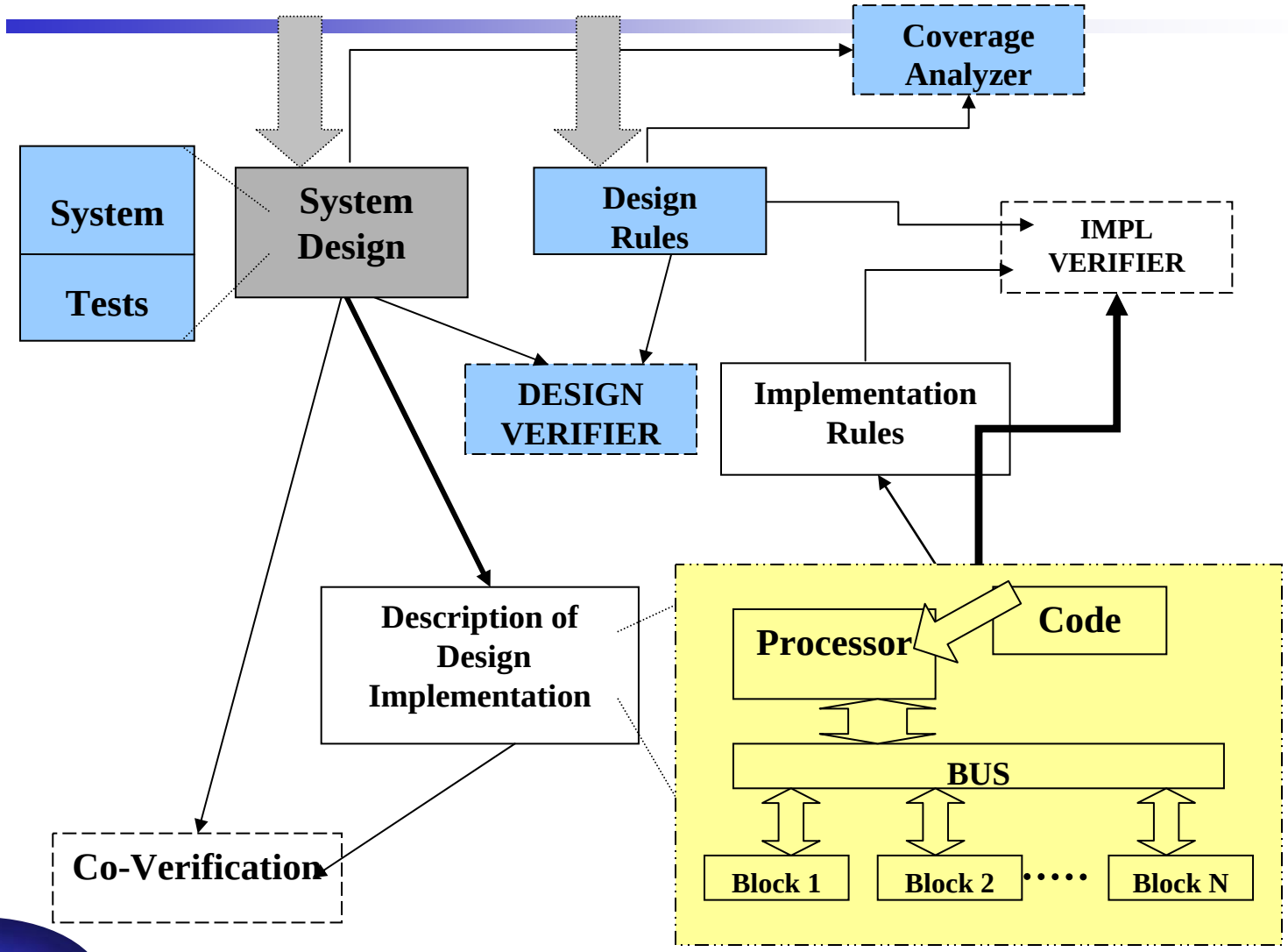


Verification using Controlled Simulation

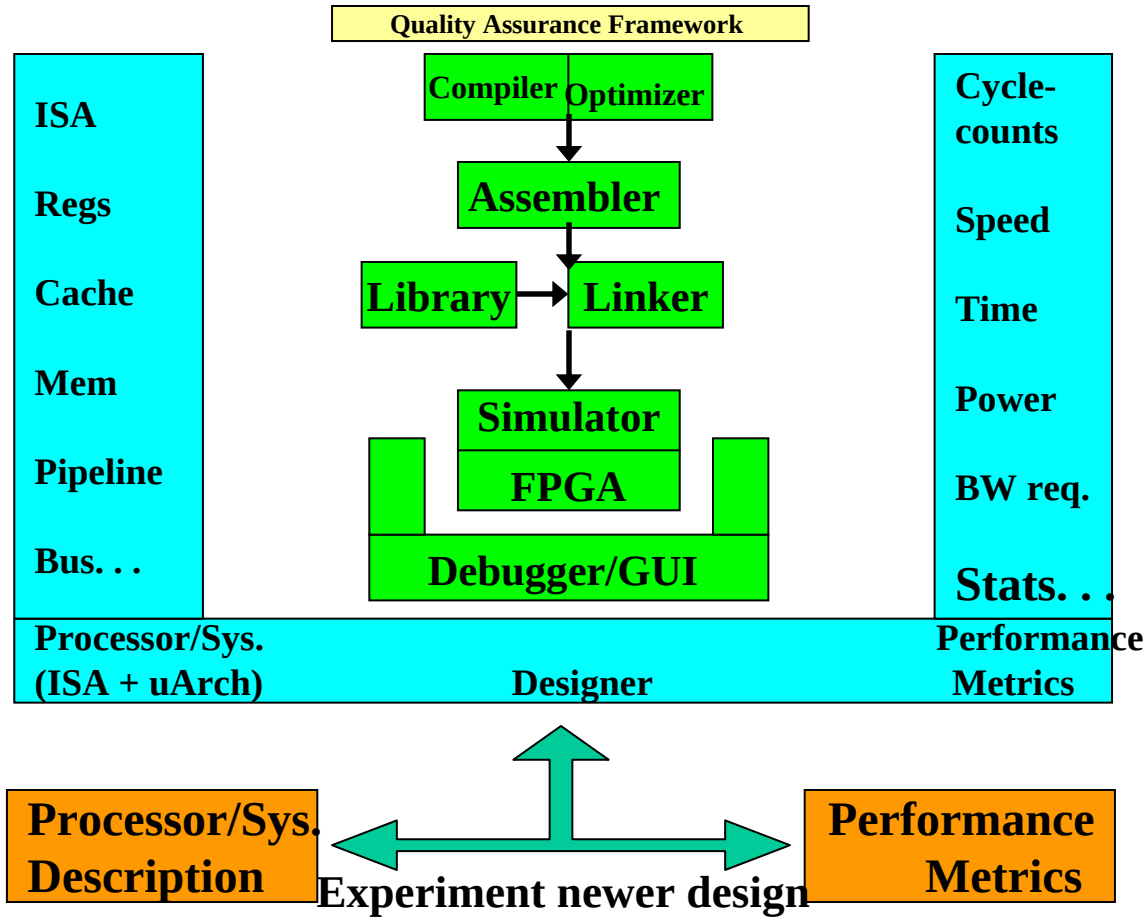


Design Implementation:

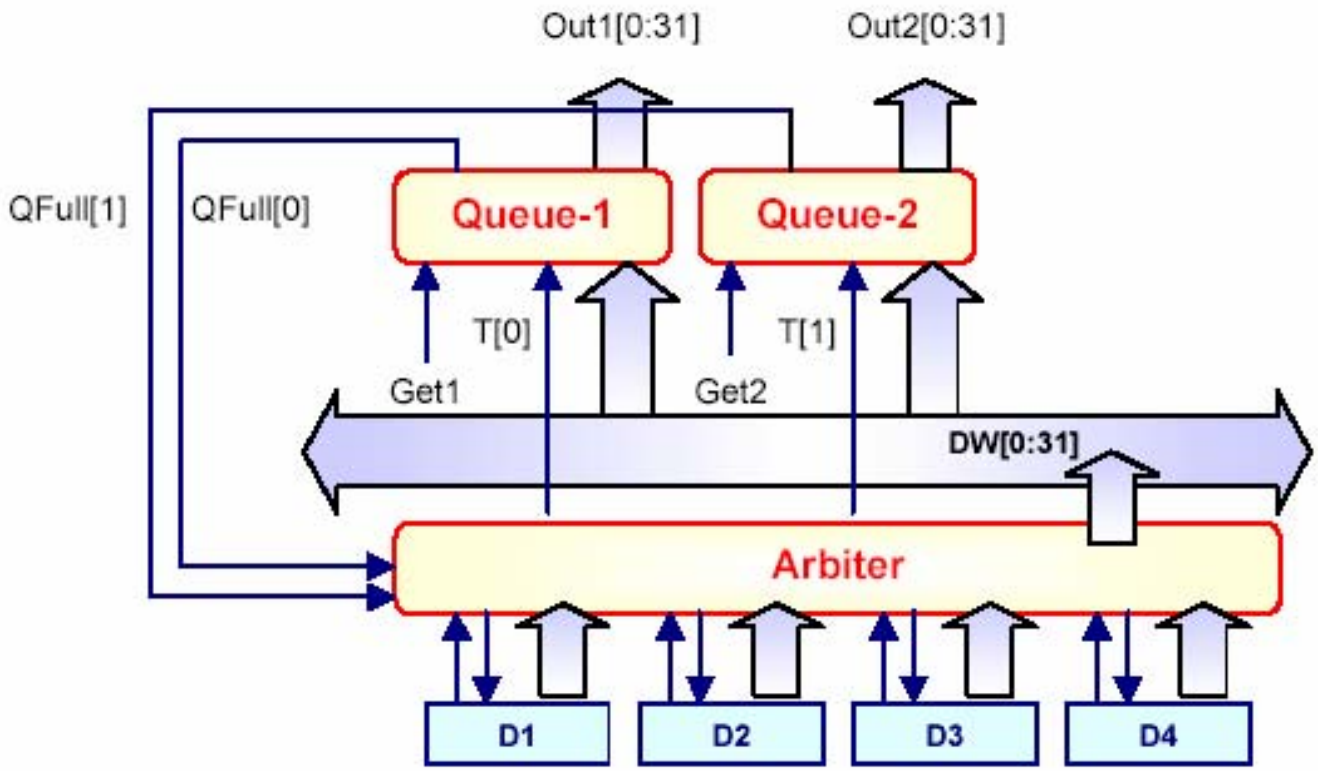
Partitioning, Processor Design, Software Tools, Custom Blocks



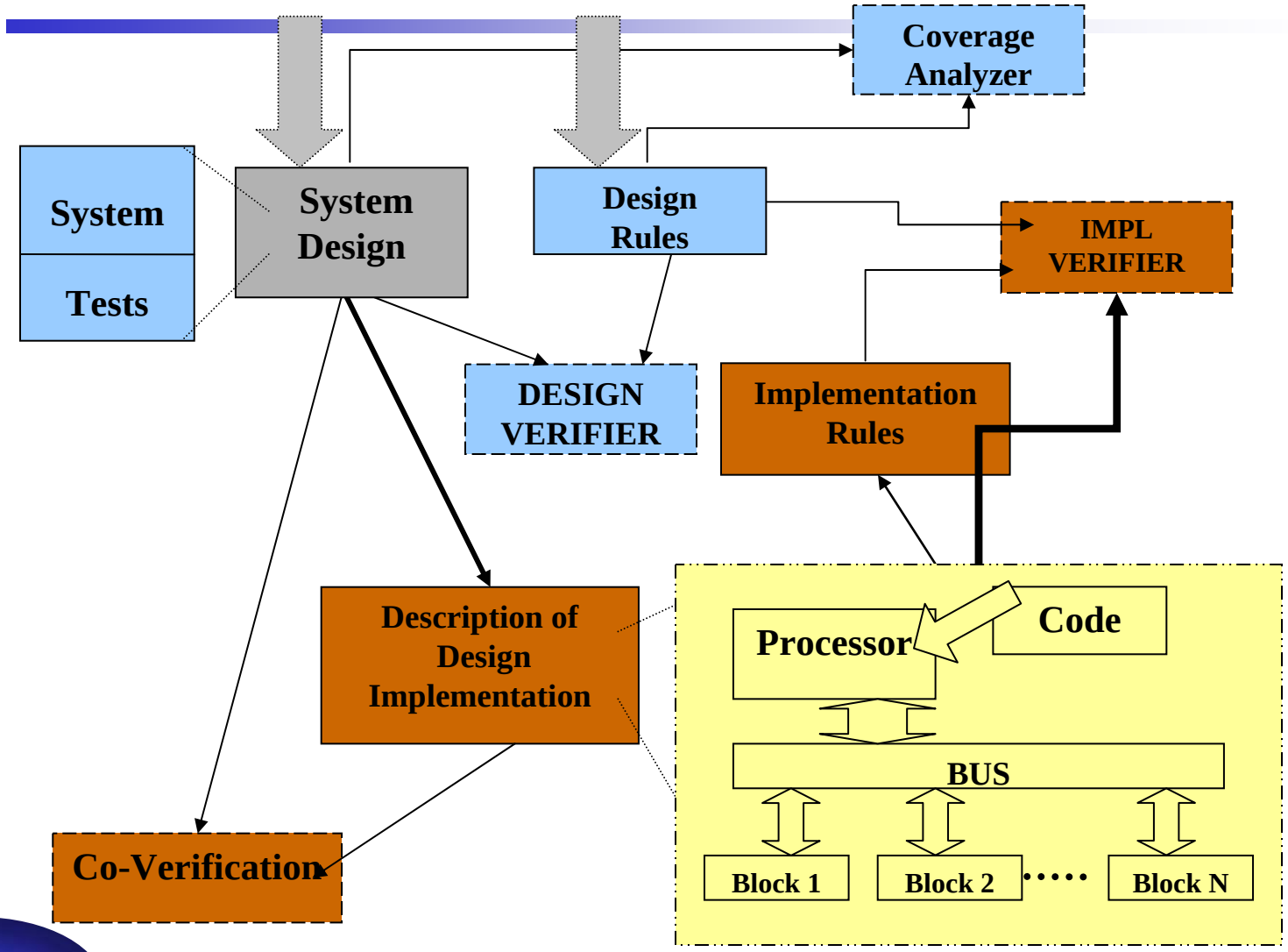
Processor Design Cycle



Custom Blocks



Implementation Verification



RTL Design

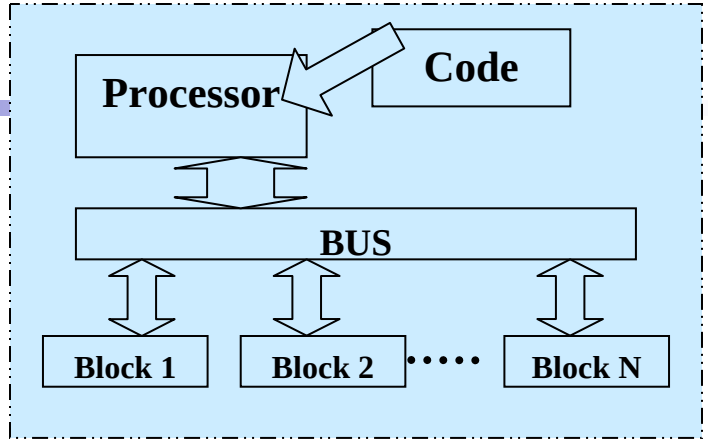
Application Spec

System Architecture

Processor Design

Software Design

Blocks



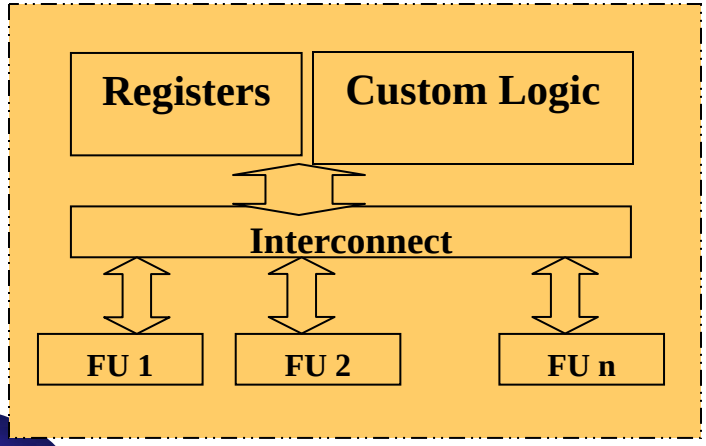
Process/Behavior

Register Transfer

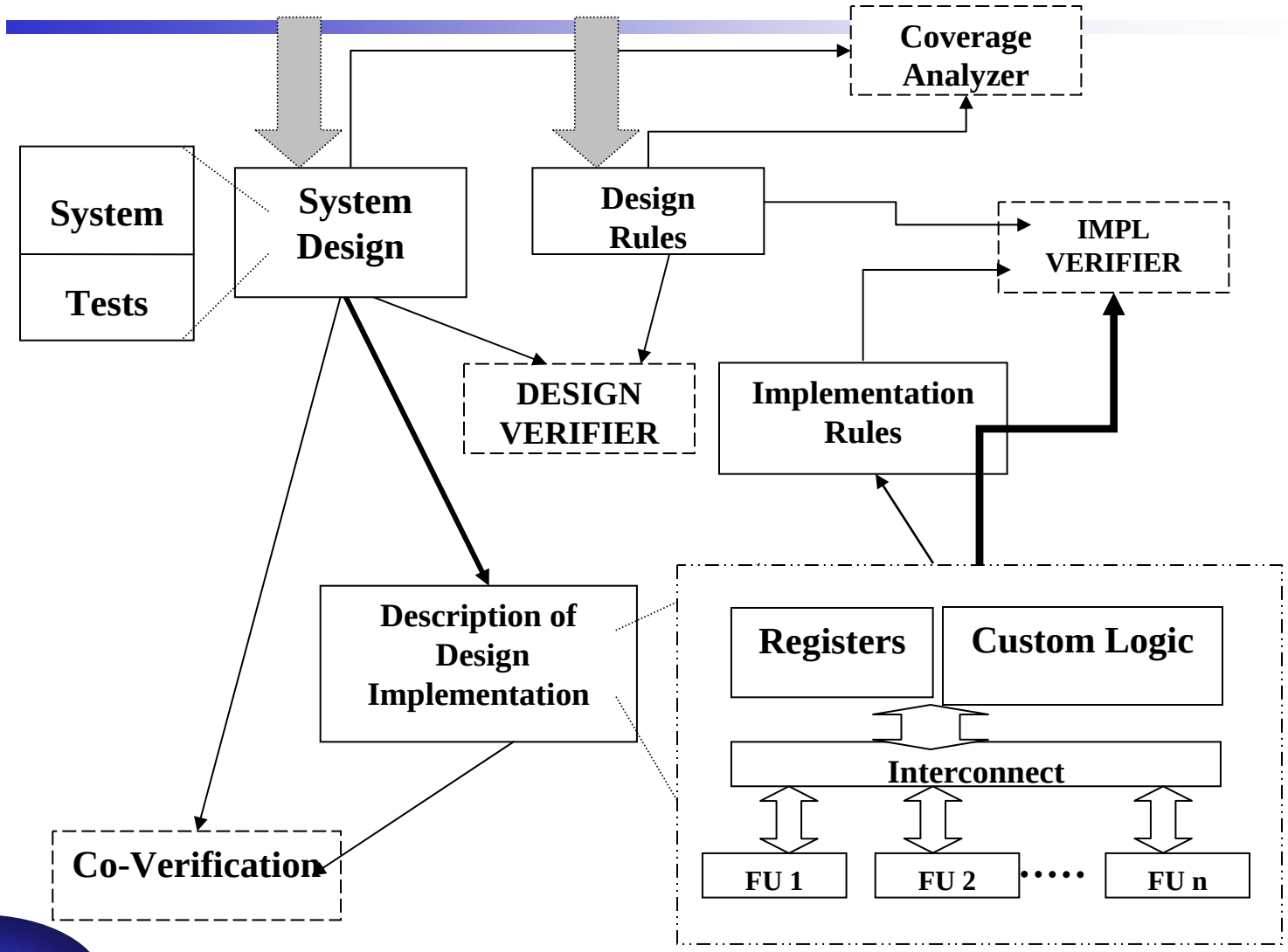
Gate

Transistor/RLC

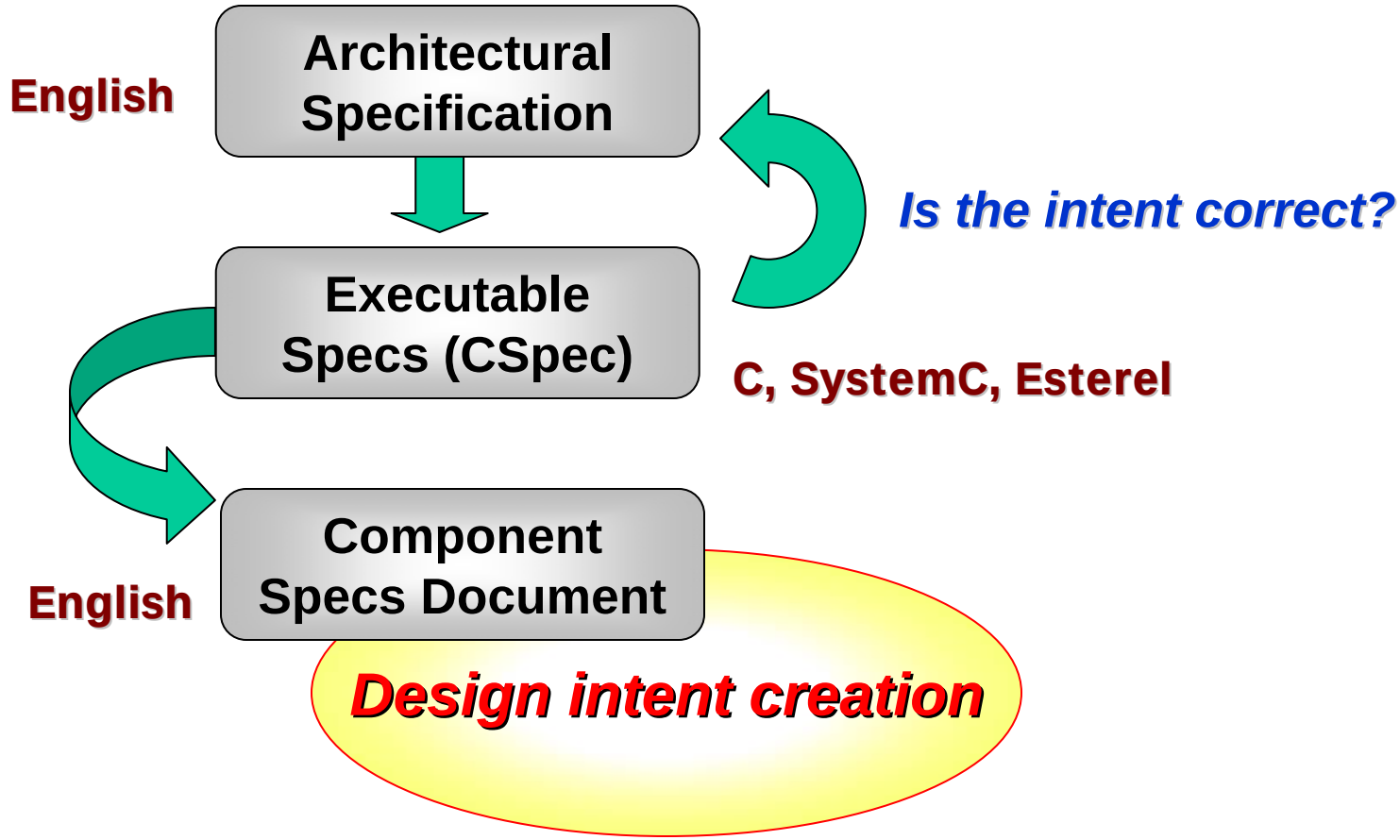
Layout



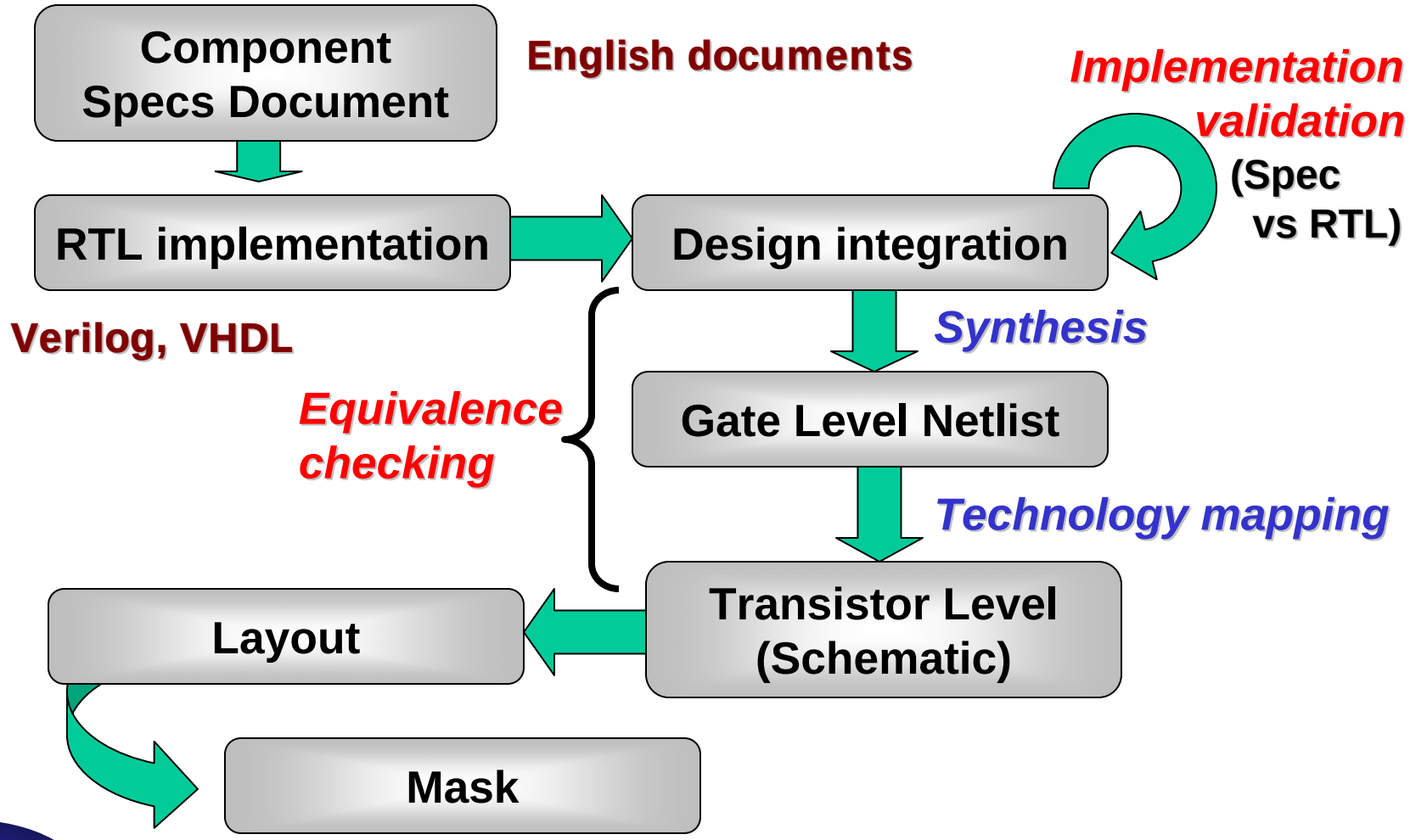
Hardware Verification



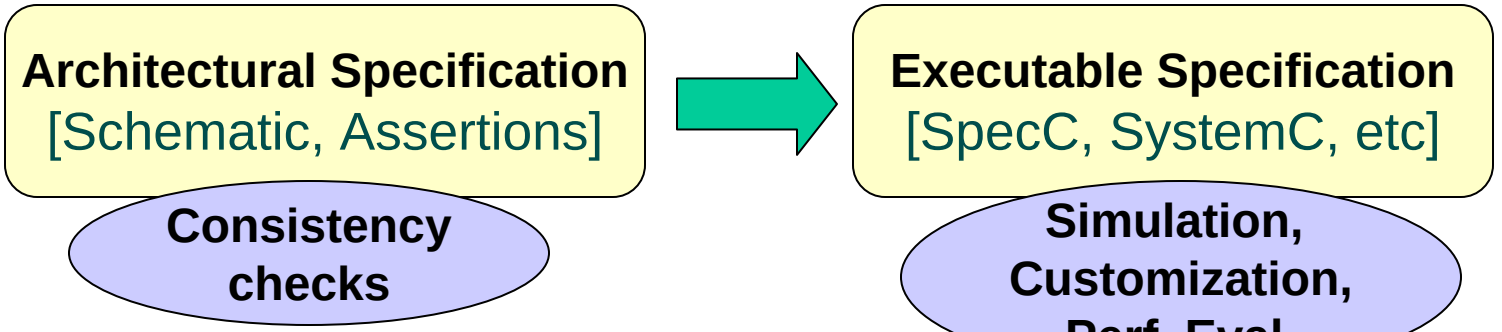
Design Cycle: *Intent Creation*



Design Cycle: *Implementation*

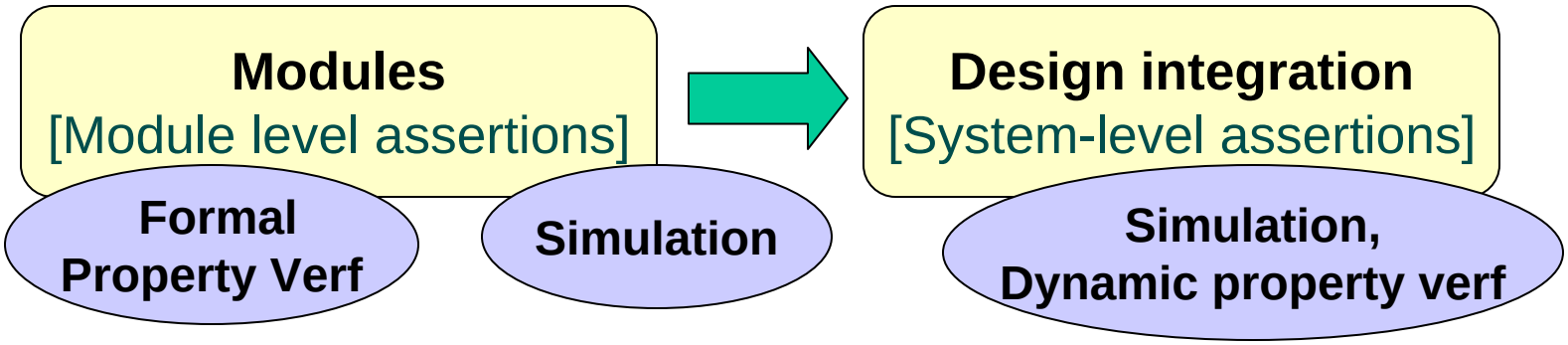


Main validation tasks

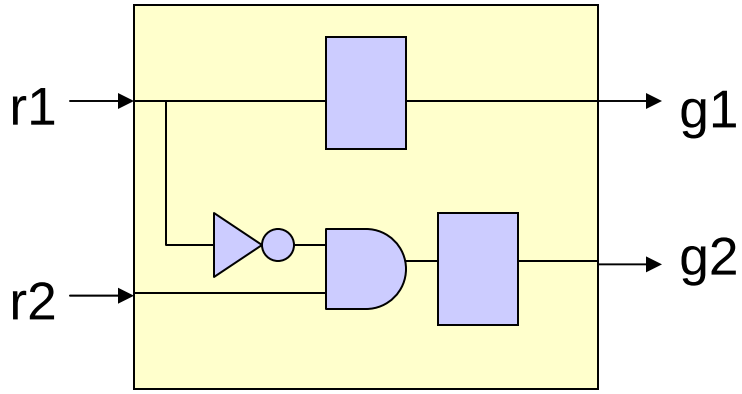


Design intent verification

Implementation verification



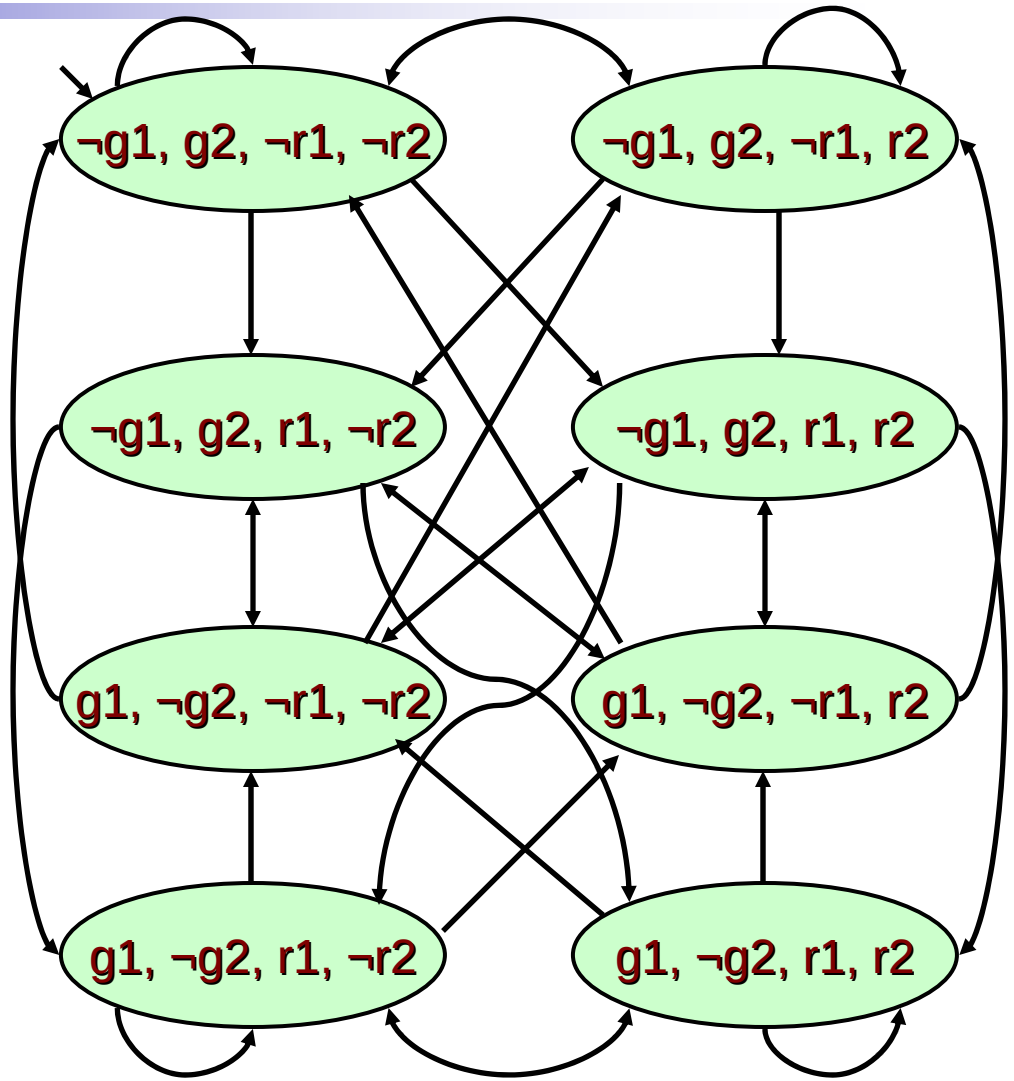
Example: Priority Arbiter [Schematic and High-Level Spec]



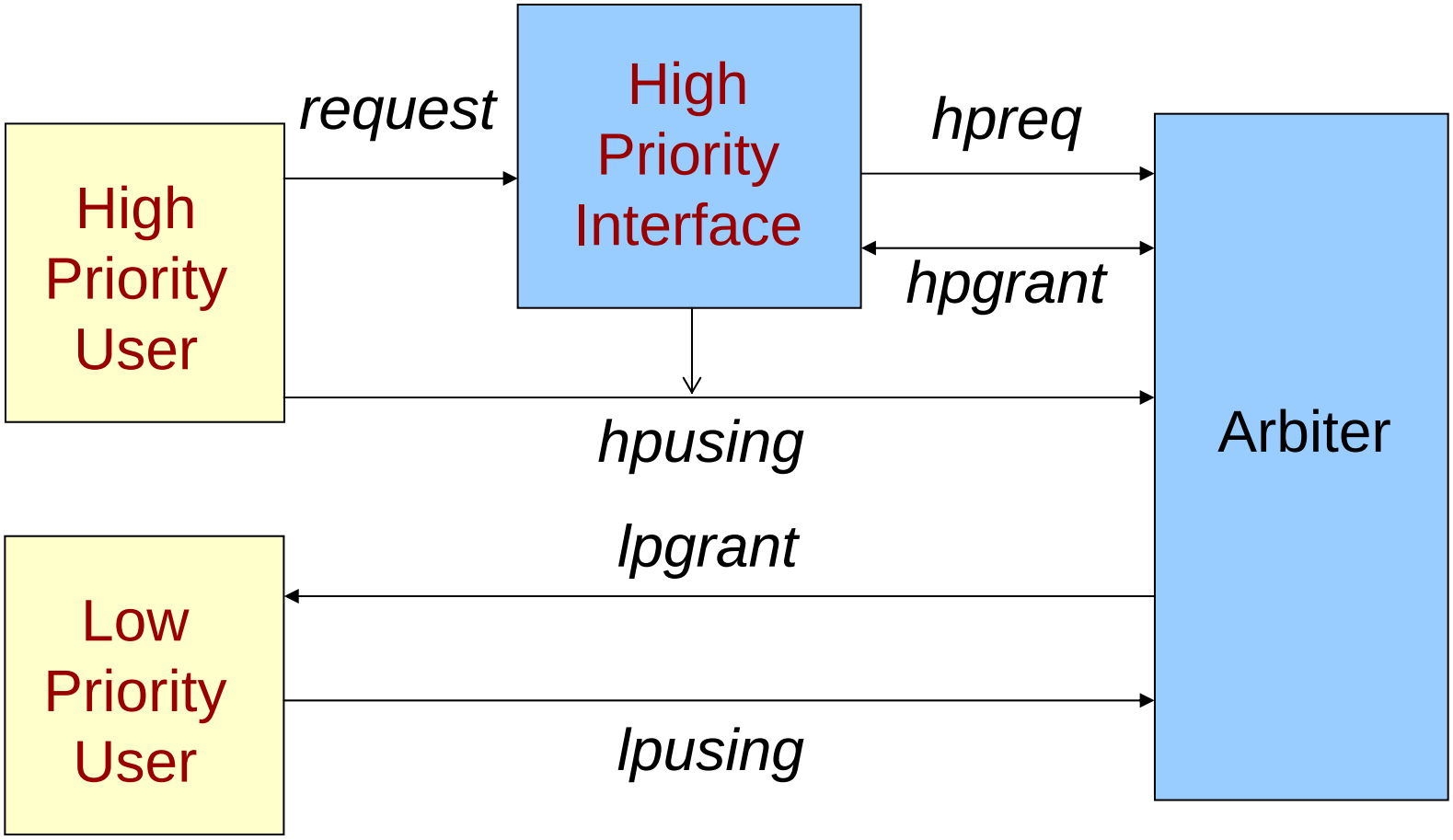
The system requires to arbitrate between requests r1 and r2 and provide grants g1 and g2 in such a way that r2 is default but r1 is given higher priority over r2. Mutual exclusion must be guaranteed.

Example: Priority Arbiter [Verilog Code and Formal Model]

```
always
begin  @(posedge clk)
         begin
         if(r1 == 1)
            @(posedge clk)
            begin  g1 = 1;  g2 = 0; end
         if(r2 == 1 && r1 == 0)
           @(posedge clk)
            begin  g2 = 1;  g1 = 0; end
         if(r2 == 0 && r1 == 0)
           @(posedge clk)
           begin  g2 = 1;  g1 = 0;  end
         end
end
```

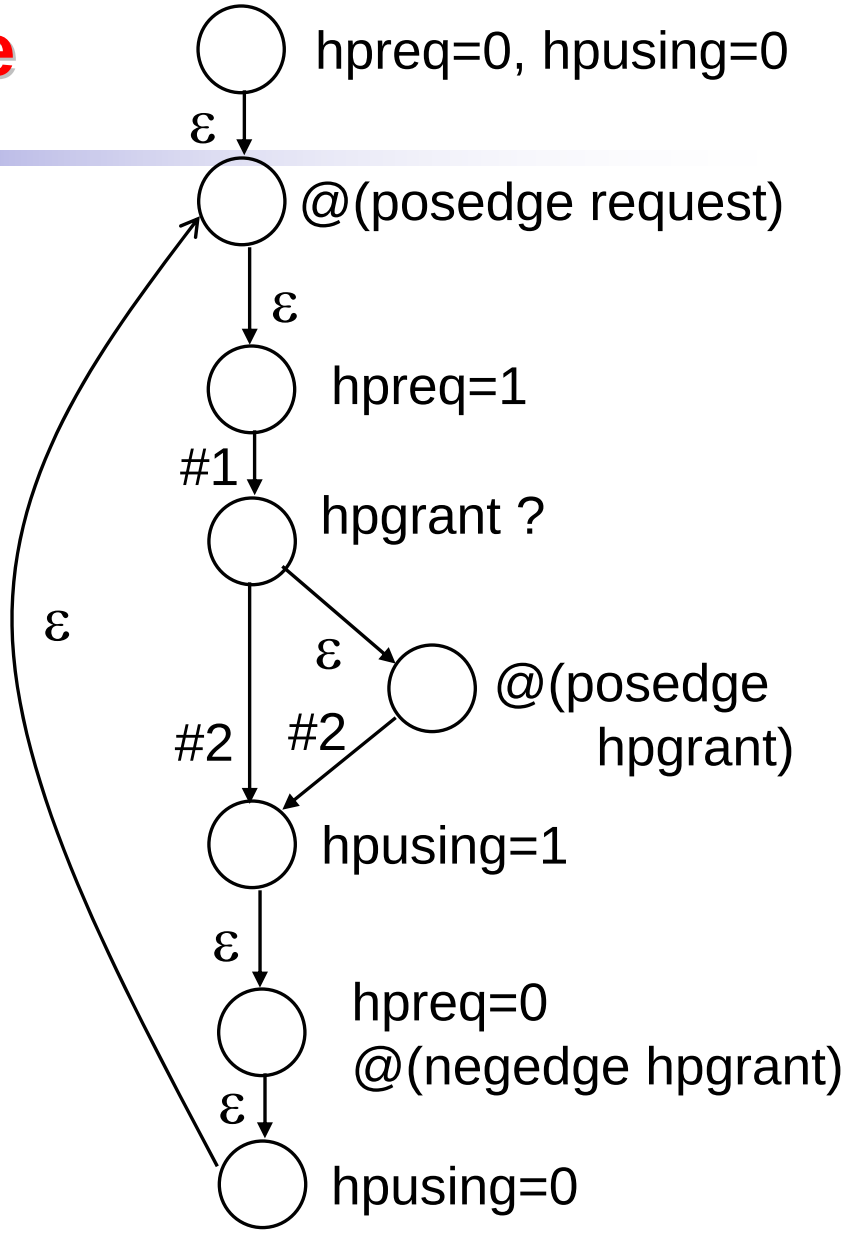


Resource Arbiter System (Schematic)



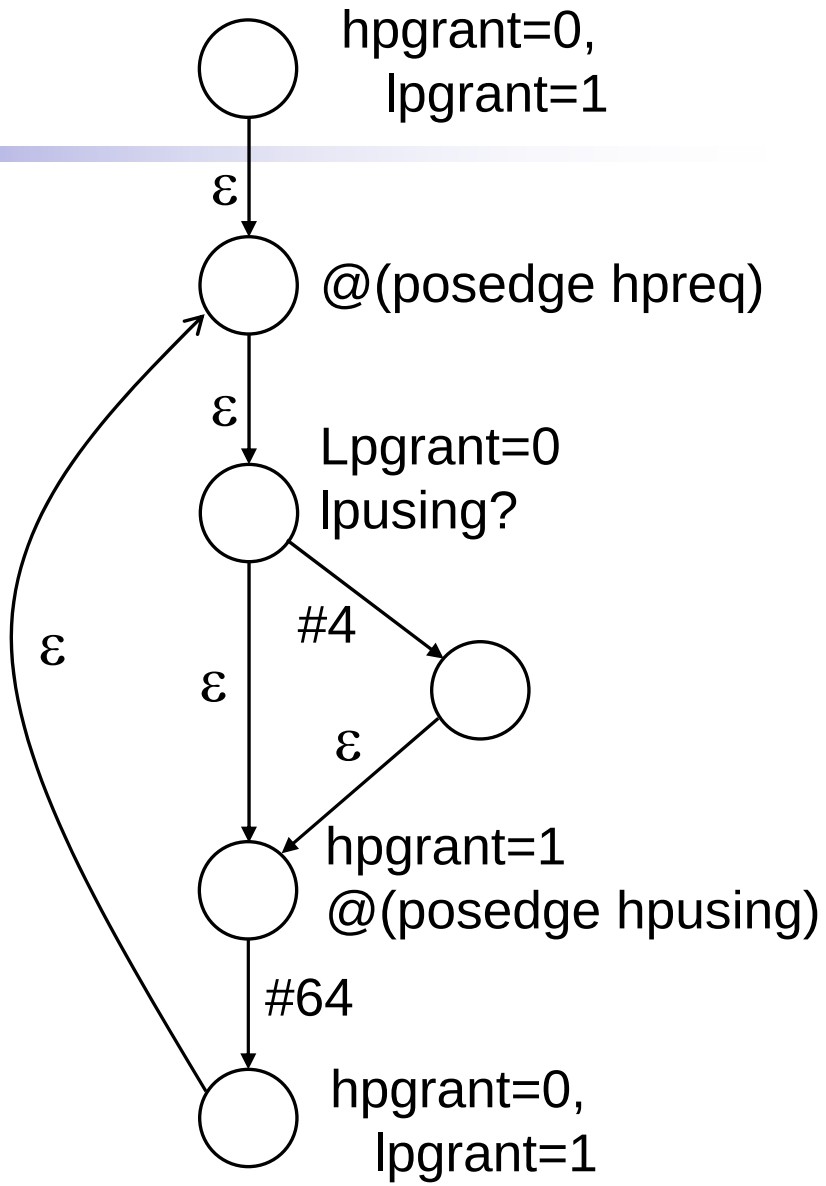
High Priority Interface

```
initial begin
 hpreq=0; hpusing=0;
end
always @(posedge request)
begin
 hpreq=1; #1;
 if (hpgrant==1) #2;
 else begin
   @(posedge hpgrant); #2;
 end
 hpusing=1; hpreq=0;
 @(negedge hpgrant);
 hpusing=0;
end
```

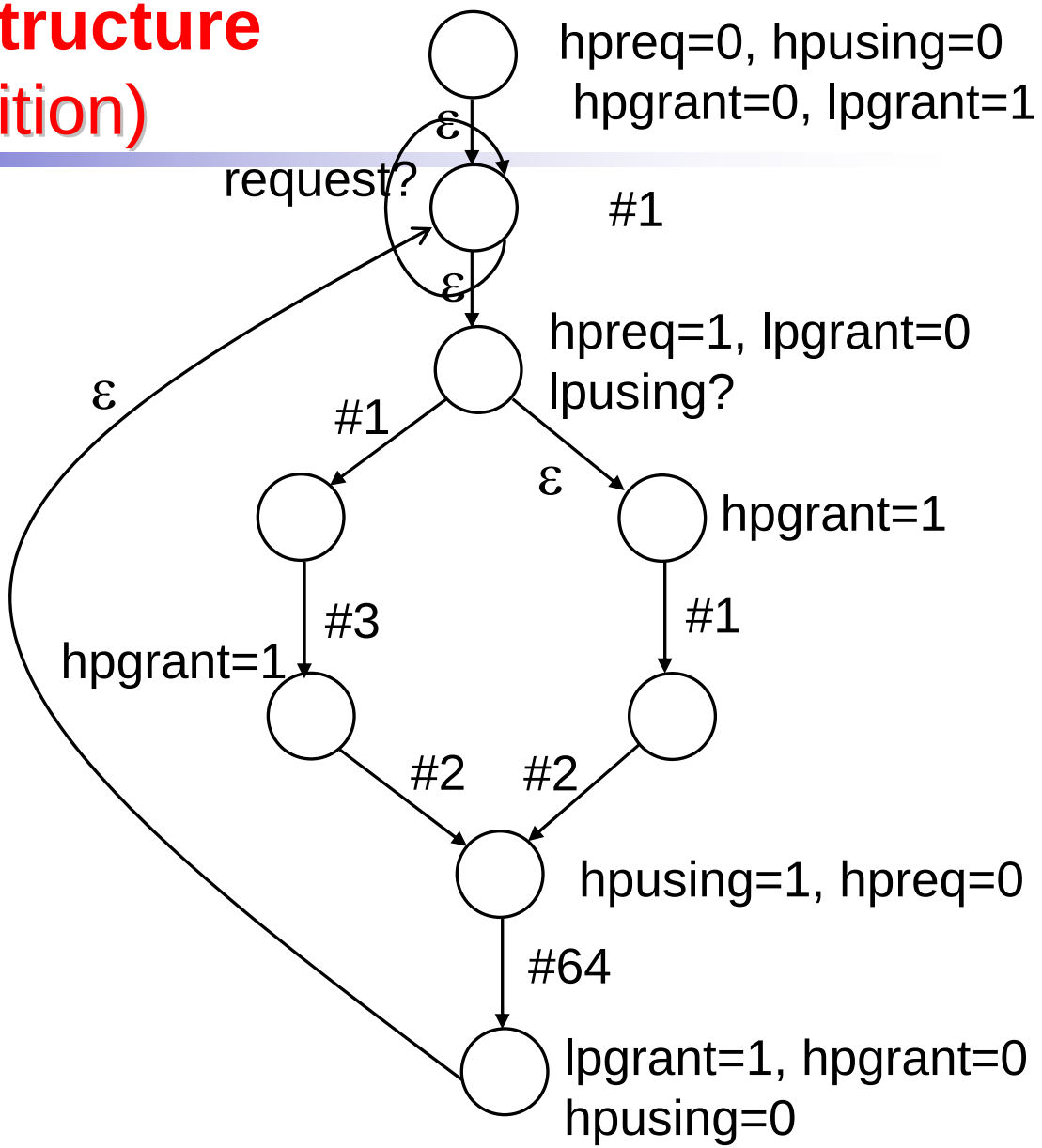


Arbiter

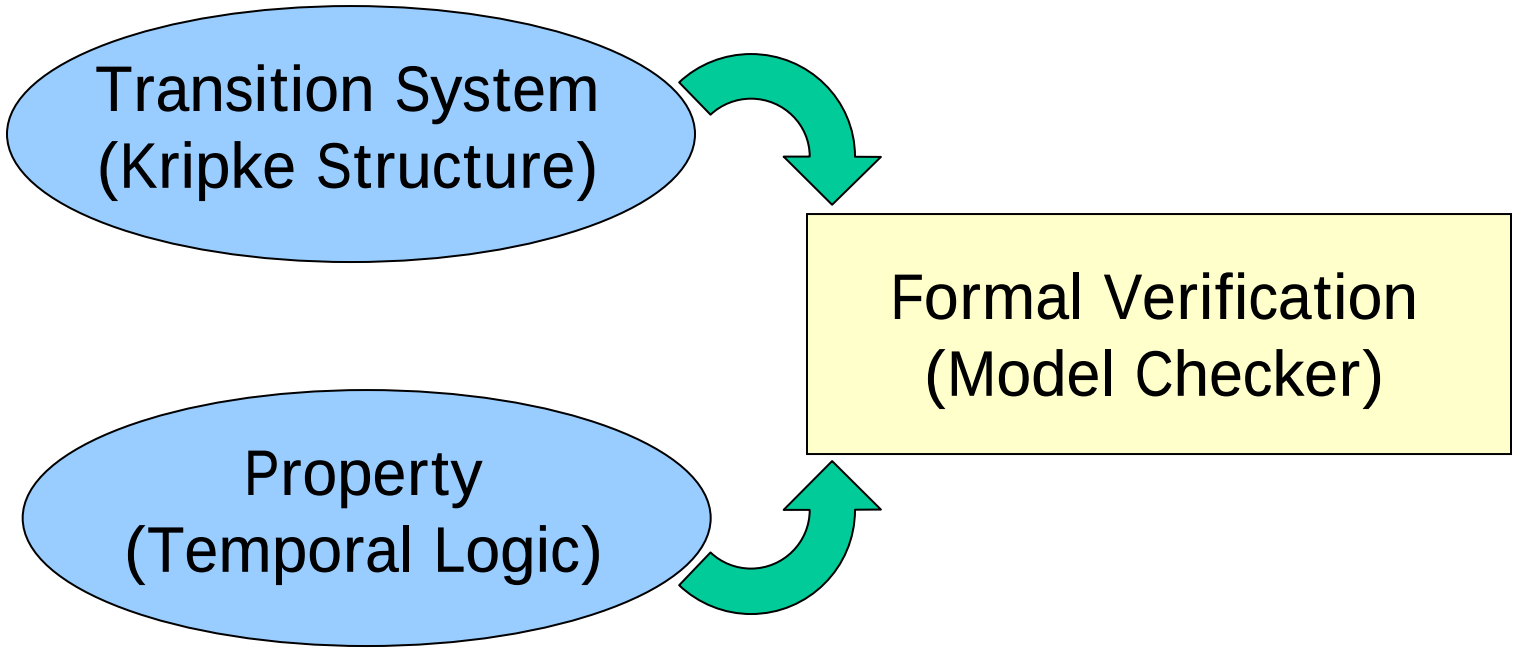
```
initial begin
  hpgrant=0; lpgrant=1;
end

always @(posedge hpreq)
begin
  lpgrant=0;
  if (lpusing !=0)
     #4;
  hpgrant=1;
  @(posedge hpusing);
  #64;
  hpgrant=0; lpgrant=1;
end
```



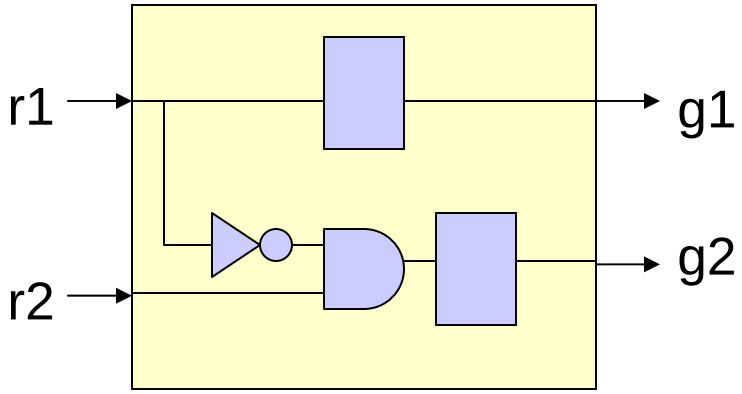
Timed event structure (after composition)



The Design Verification Problem

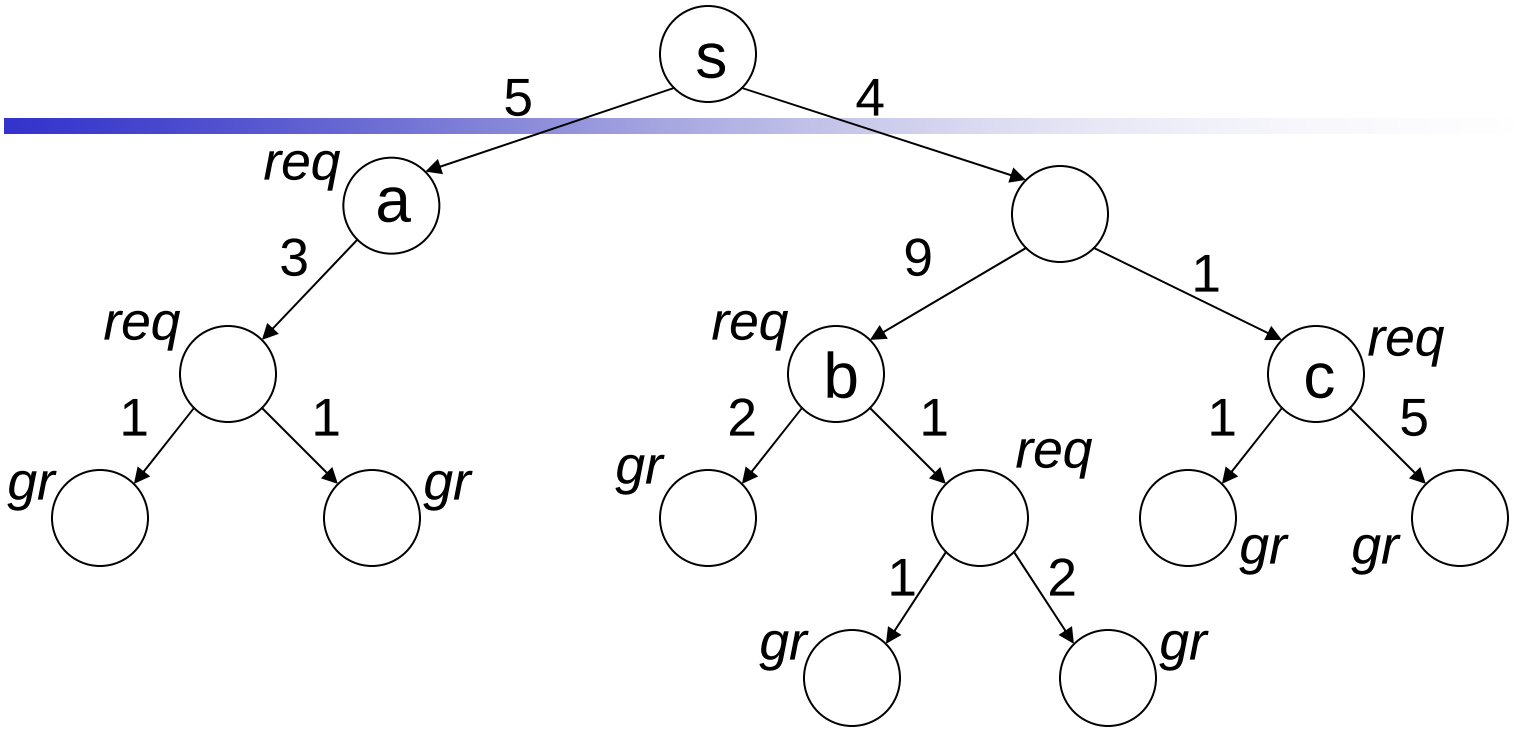


Priority Arbiter: Properties



- Whenever r1 is asserted, g1 is given in the next cycle
- When r2 is the sole request, g2 comes in the next cycle
- When none of them are requesting, the arbiter parks the grant on g2
- g1 and g2 can not be true at the same time (mutual exclusion)

Analyzing Request and Grants



- From s the system always makes a request in future
- All requests are eventually granted
- Sometimes requests are immediately granted
- Requests are not always immediately granted
- Requests are held till grant is received

Timing Properties

- Whenever a hpreq is recorded, the hpgrant should take place within 4 units of time.

- The arbiter will provide exactly 64 units of time to high-priority users in each grant.

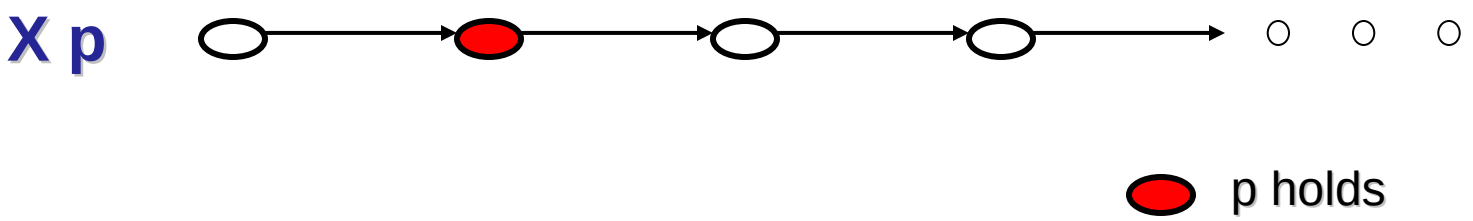
What is temporal logic?

- Logic with *temporal* operators (operators that talk about time)
  - Eg. Tense Logic (A. N. Prior, 1957)
    - P "It has at some time been the case that …"
    - F "It will at some time be the case that …"
    - H "It has always been the case that …"
    - G "It will always be the case that …"

Temporal Logic for Validation

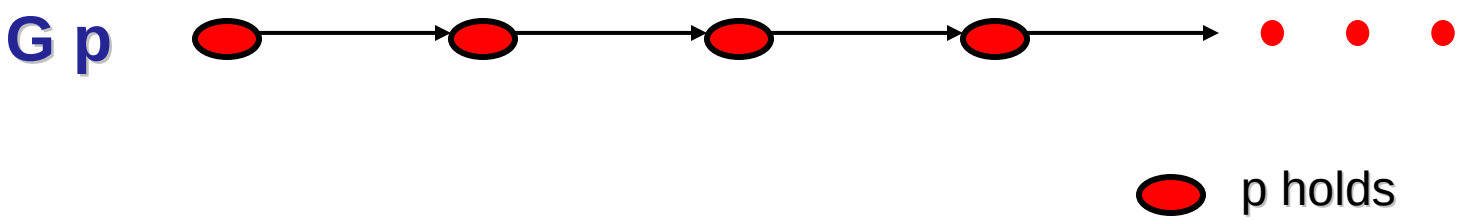
- Formalism for describing sequences of transitions between states in a reactive system
- Time is not mentioned explicitly
  - *eventually* some designated state is reached
  - an error state is *never* entered
  - *eventually* or *never* are specified using special temporal operators
  - temporal operators can also be combined with Boolean connectives or nested arbitrarily

Informal Semantics



- p holds in the next state

Informal Semantics

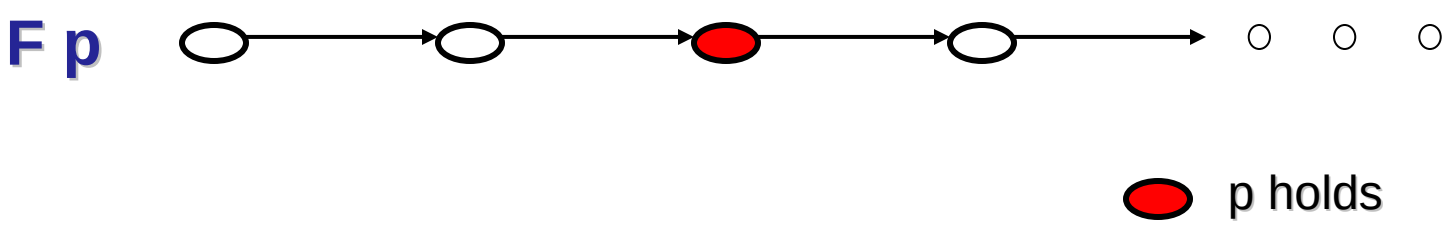


- p holds always (globally)

  alternatively

- ¬p does not hold eventually

Informal Semantics

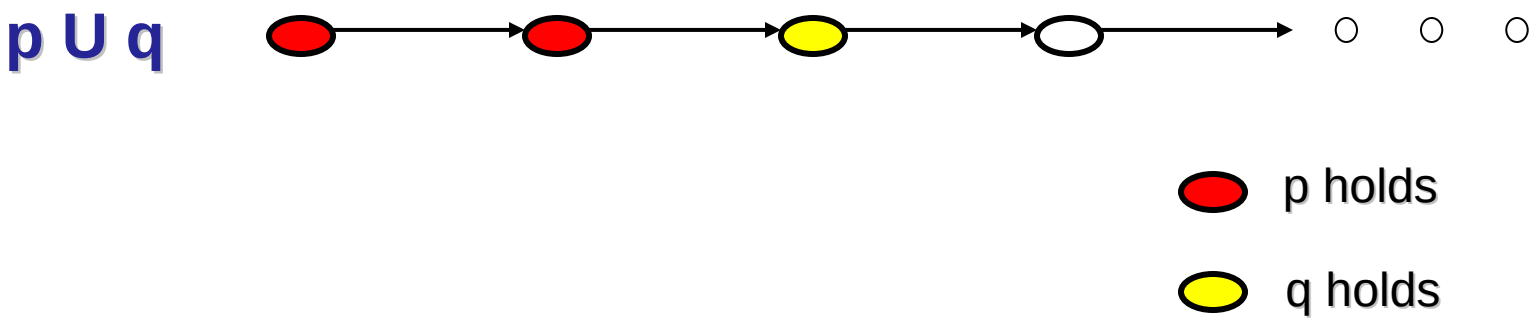


- p holds eventually (in future)

  alternatively

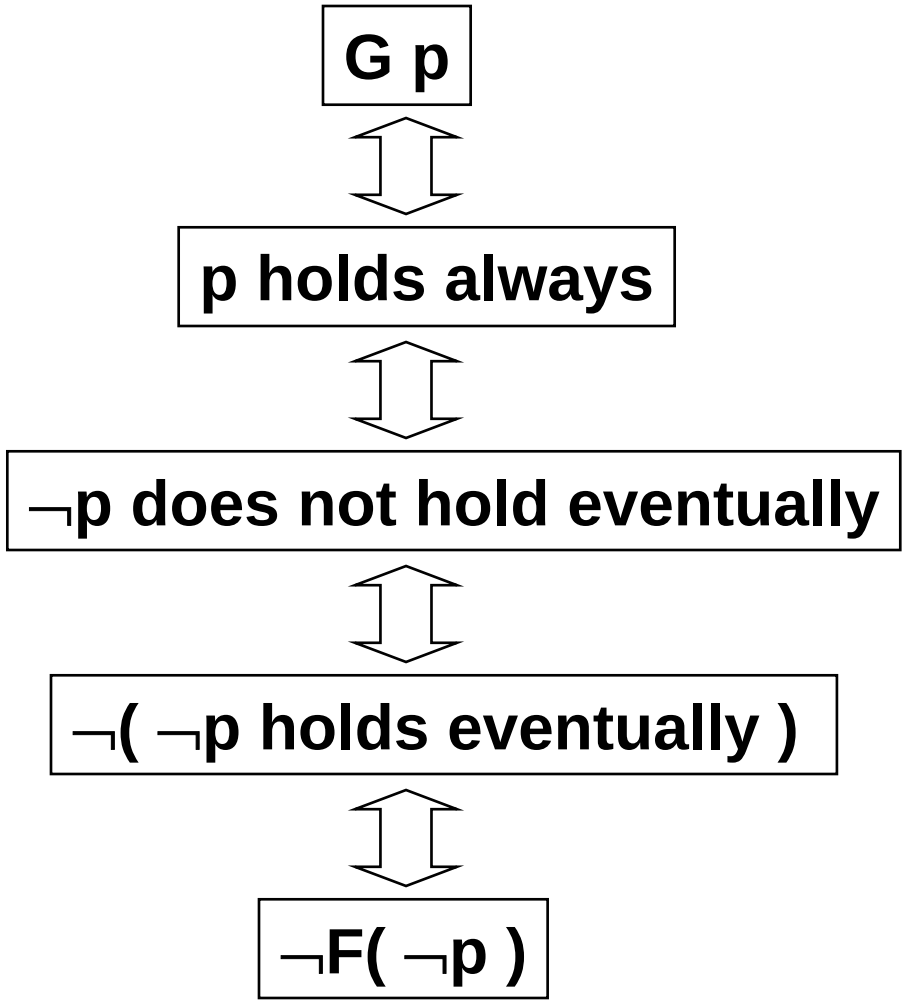
- ¬p does not hold always

Informal Semantics



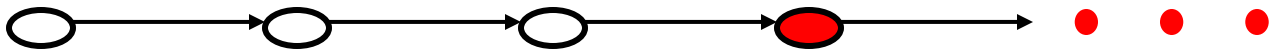
- q holds eventually **and** p holds until q holds

Duality between Temporal Operators



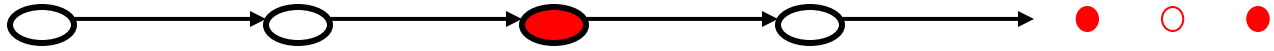
Nesting of Temporal Operators

F G p



Along the path there exists a state from which *p* will hold forever

G F p

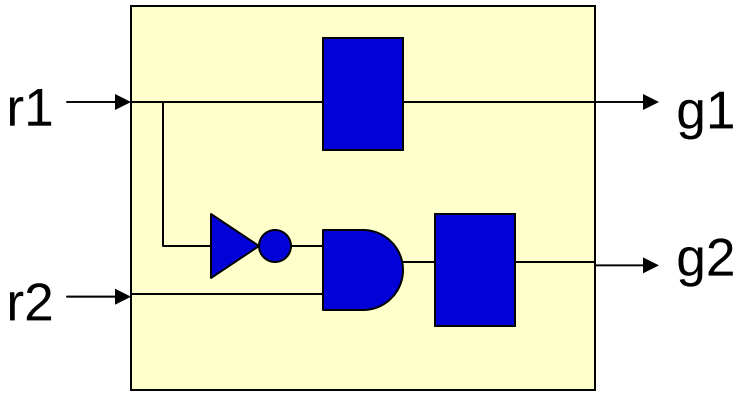


Along the path for all states there will be eventually some state where *p* holds

alternatively

Along the path p will hold ***infinitely often***

Example



- Either g1 or g2 is always false (mutual exclusion)

$$G[\neg g1 \vee \neg g2]$$

- Whenever r1 is asserted, g1 is given in the next cycle

$$G[\, r1 \Rightarrow Xg1 \,]$$

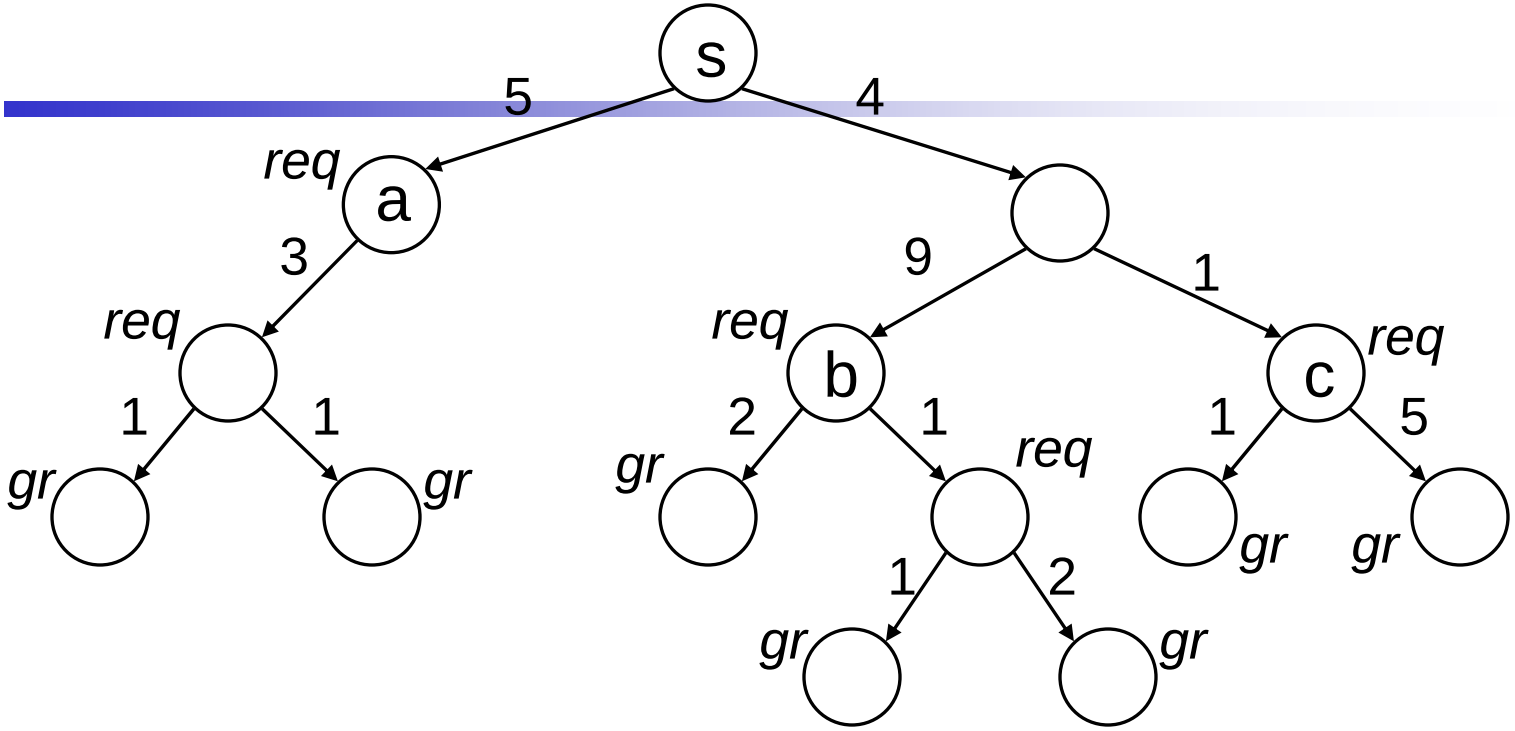
- When r2 is the sole request, g2 comes in the next cycle

$$G[\, (\neg r1 \wedge r2) \Rightarrow Xg2 \,]$$

- When none are requesting, the arbiter parks the grant on g2

$$G[\, (\neg r1 \wedge \neg r2) \Rightarrow Xg2 \,]$$

Analyzing Request and Grants



From s the system always makes a request in future: $AFreq$

All requests are eventually granted: $AG(req \rightarrow AFgr)$

Sometimes requests are immediately granted: $EF(req \rightarrow EXgr)$

Requests are not always immediately granted: $\neg AG(req \rightarrow AXgr)$

Requests are held till grant is received: $AG(req \rightarrow AF(req\ U\ gr))$

Timing Properties

- Whenever a hpreq is recorded, the hpgrant should take place within 4 units of time.

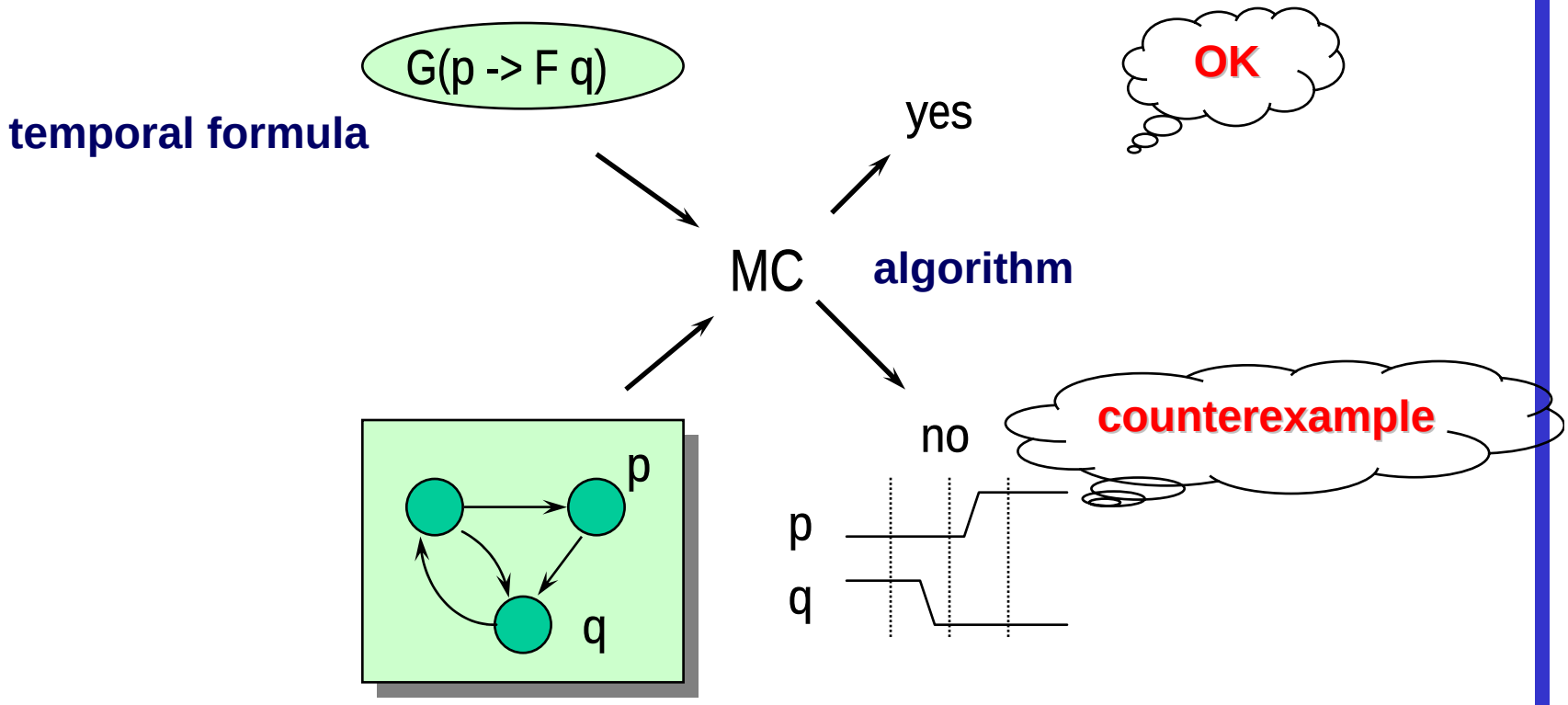
  $AG(posedge(hpreq) \rightarrow A(true\ U_{[0,4]}\ posedge(hpgrant)))$

- The arbiter will provide exactly 64 units of time to high-priority users in each grant.

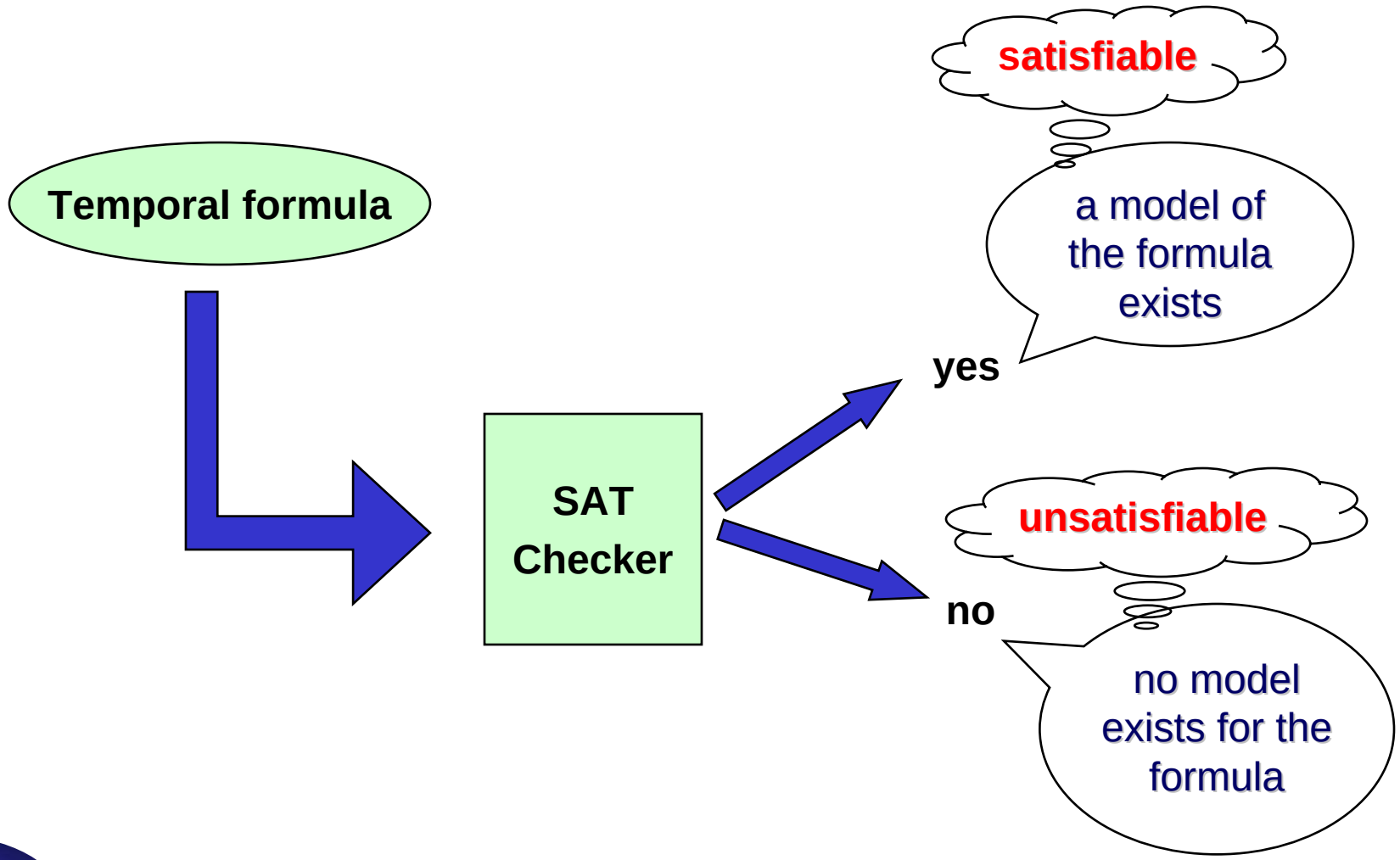
  $AG(posedge(hpusing) \rightarrow$

  $A(\neg negedge(hpusing)\ U_{[64,64]}\ negedge(hpusing)))$

Model Checking of a Temporal Logic Formula

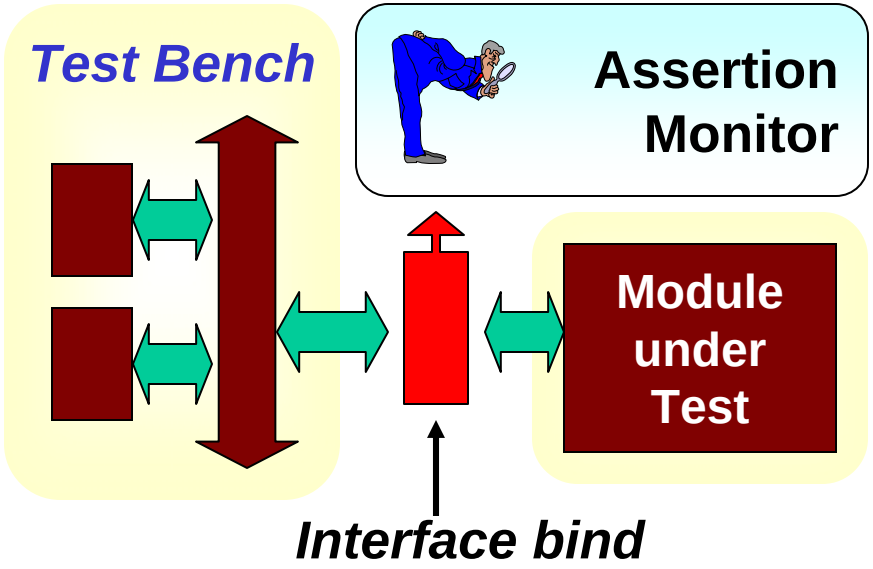


Satisfiability Checking of a Temporal Logic Formula

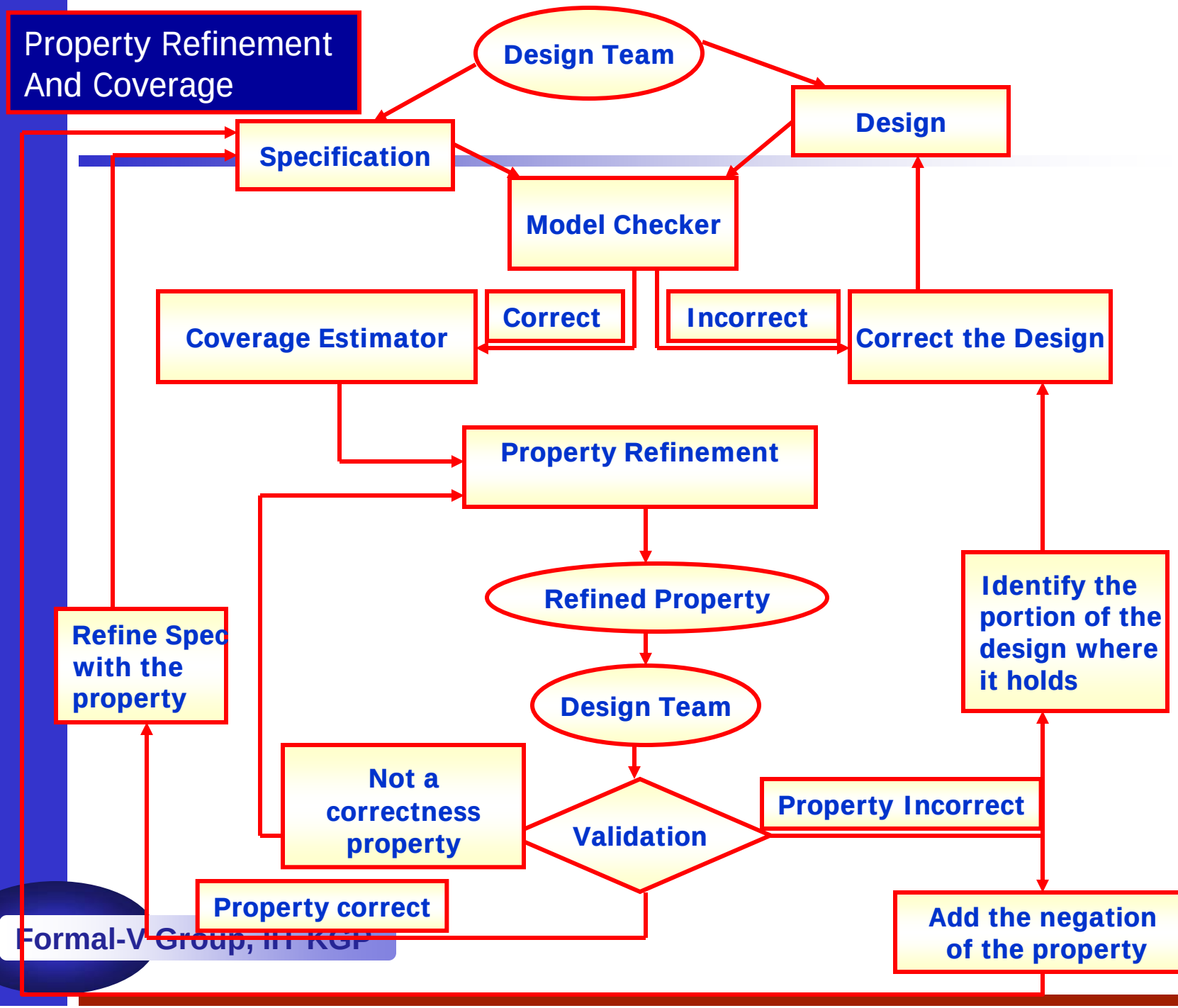


Assertion-Based Verification (ABV)

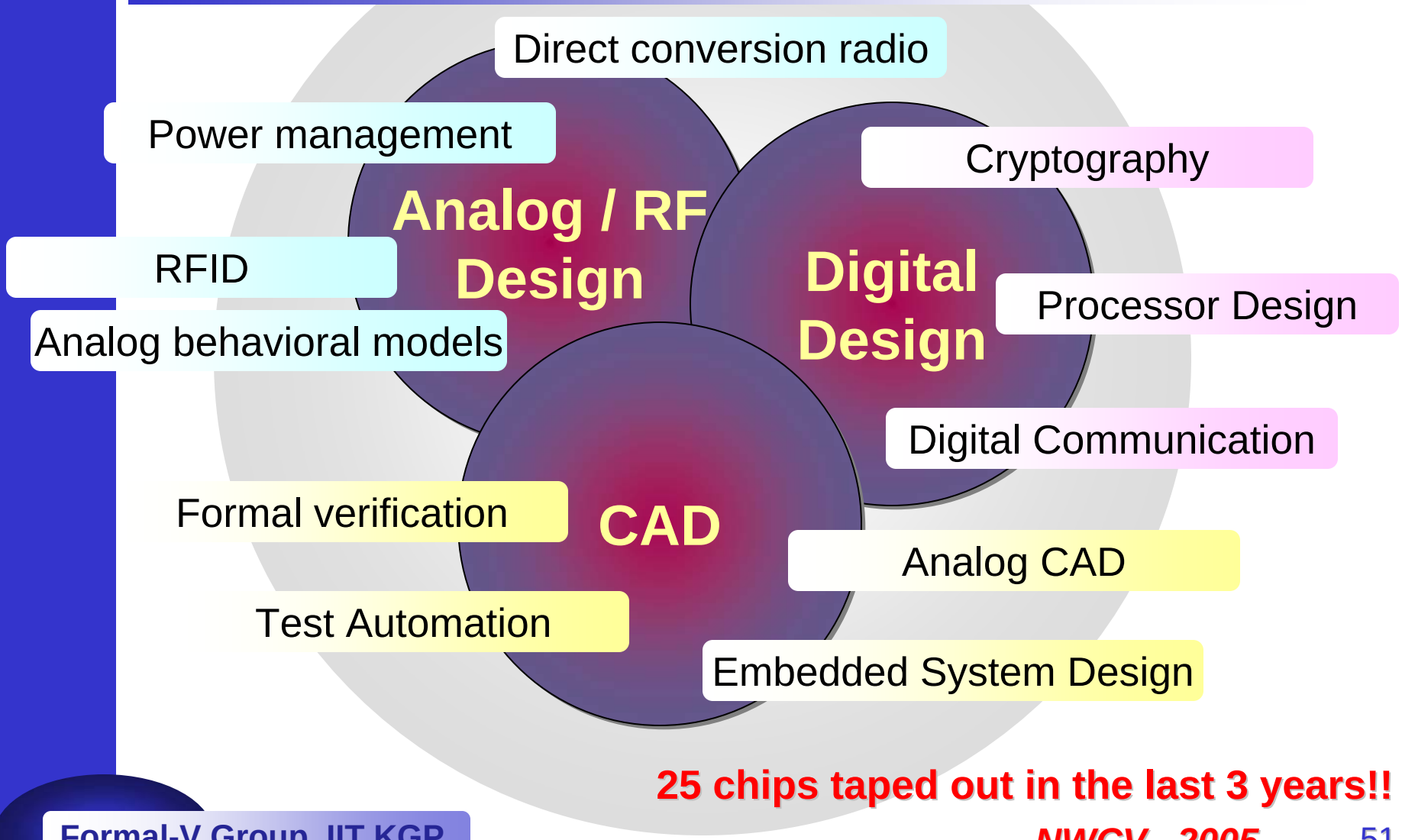
- Design intent is expressed using assertions
- Simulation is done as usual
  - Assertions find more bugs faster
  - Assertions isolate the source of the problem
- Use formal methods to guide simulation



Property Refinement And Coverage



Advanced VLSI Design Lab



25 chips taped out in the last 3 years!!