

Deploying formal in a simulation world

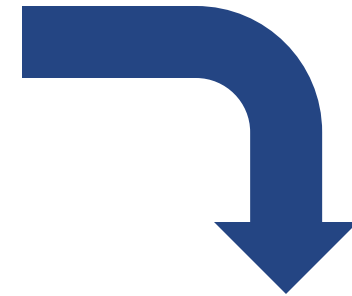
VIGYAN SINGHAL
OSKI TECHNOLOGY



Formal from different vantage points

**University Researcher
(4 yrs 1991-1995)
(UC Berkeley)**

**Goal: advance state-
of-the-art**



**Semiconductor tool user
(6 yrs 2005-2011)
(Oski)**

**Goal: optimize \$ and
time-to-market**



**EDA tool developer
(10 yrs 1995-2005)
(Cadence, Jasper)**

**Goal: build
competitive tools**

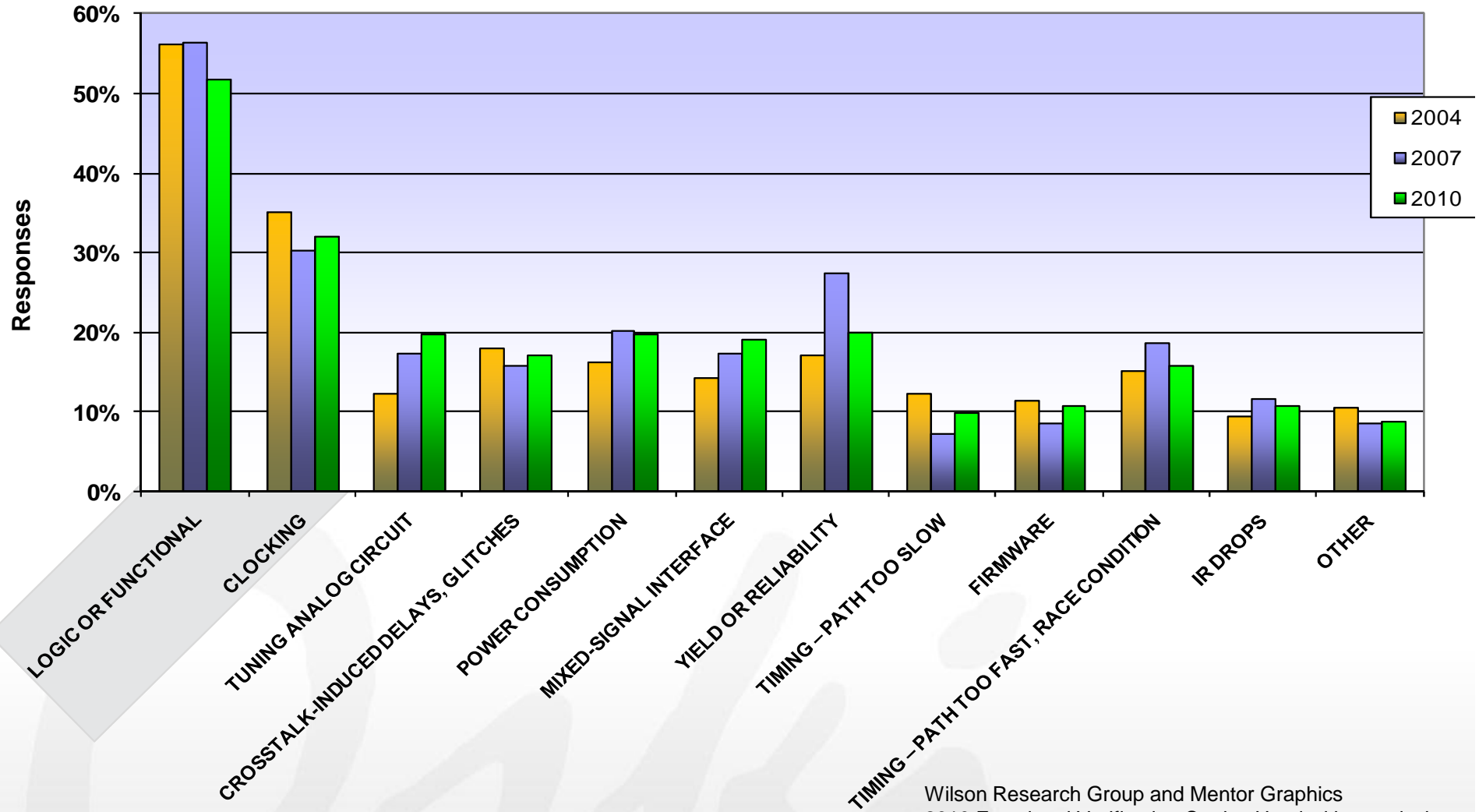
- Goal: advance state-of-the-art
- Areas of concern
 - Temporal logics (CTL, CTL*, PLTL)
 - Fairness and ω -automata
 - Complexity
 - Known: CTL Model checking linear time complexity (size of FSM)
 - Proved: CTL model checking PSPACE-complete (size of design)
- Time to returns: almost infinite
- Anecdote: “model check a 9-state FSM at Motorola”

- Goal: build competitive tools
- Areas of concern
 - Verilog/SystemVerilog parsing
 - User interface and GUI
 - Property synthesis: PSL and SVA
- Time to returns: 4-5 years
- Anecdote: “lost an eval at Intel because tool ran for 28 days”

- Goal: optimize \$ and time-to-market
- Areas of concern
 - Verification planning
 - Metrics to measure progress, and when we are done
 - Integrate simulation and formal planning and results
 - Abstraction (and reductions) are key to making formal productive
- Time to returns: 3-6 months
- Anecdote: “how did you miss a bug on a formally verified block?”

Types of post-silicon flaws

Verification is still the largest problem

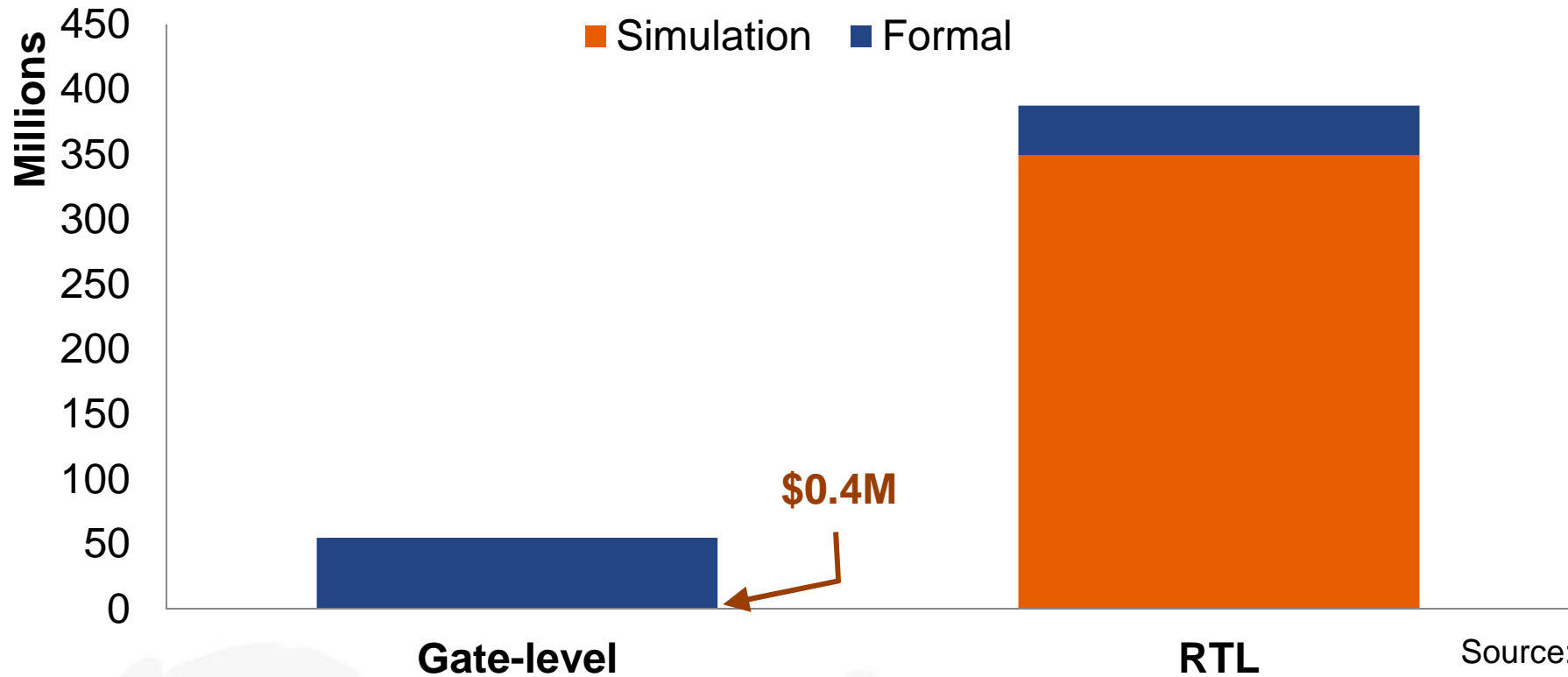


Wilson Research Group and Mentor Graphics
2010 Functional Verification Study. Used with permission.

Verification market size (2009)*



* excluding analog

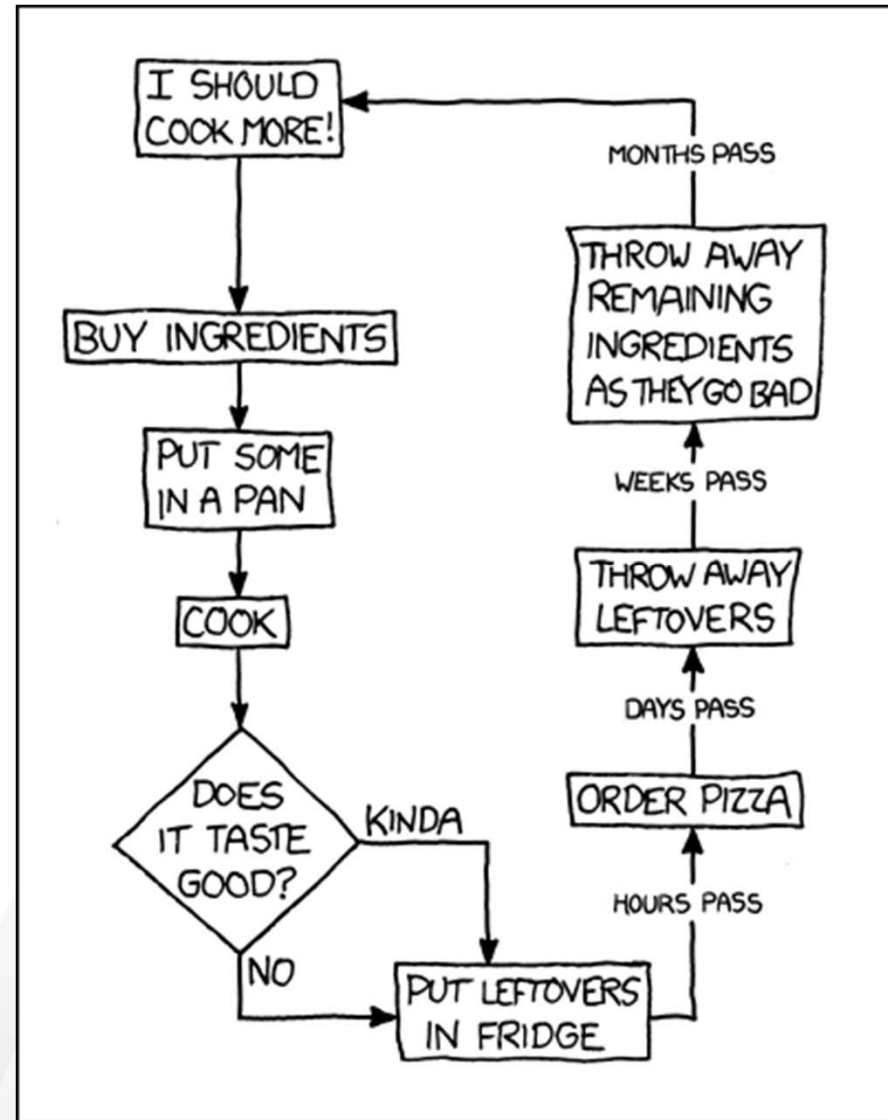


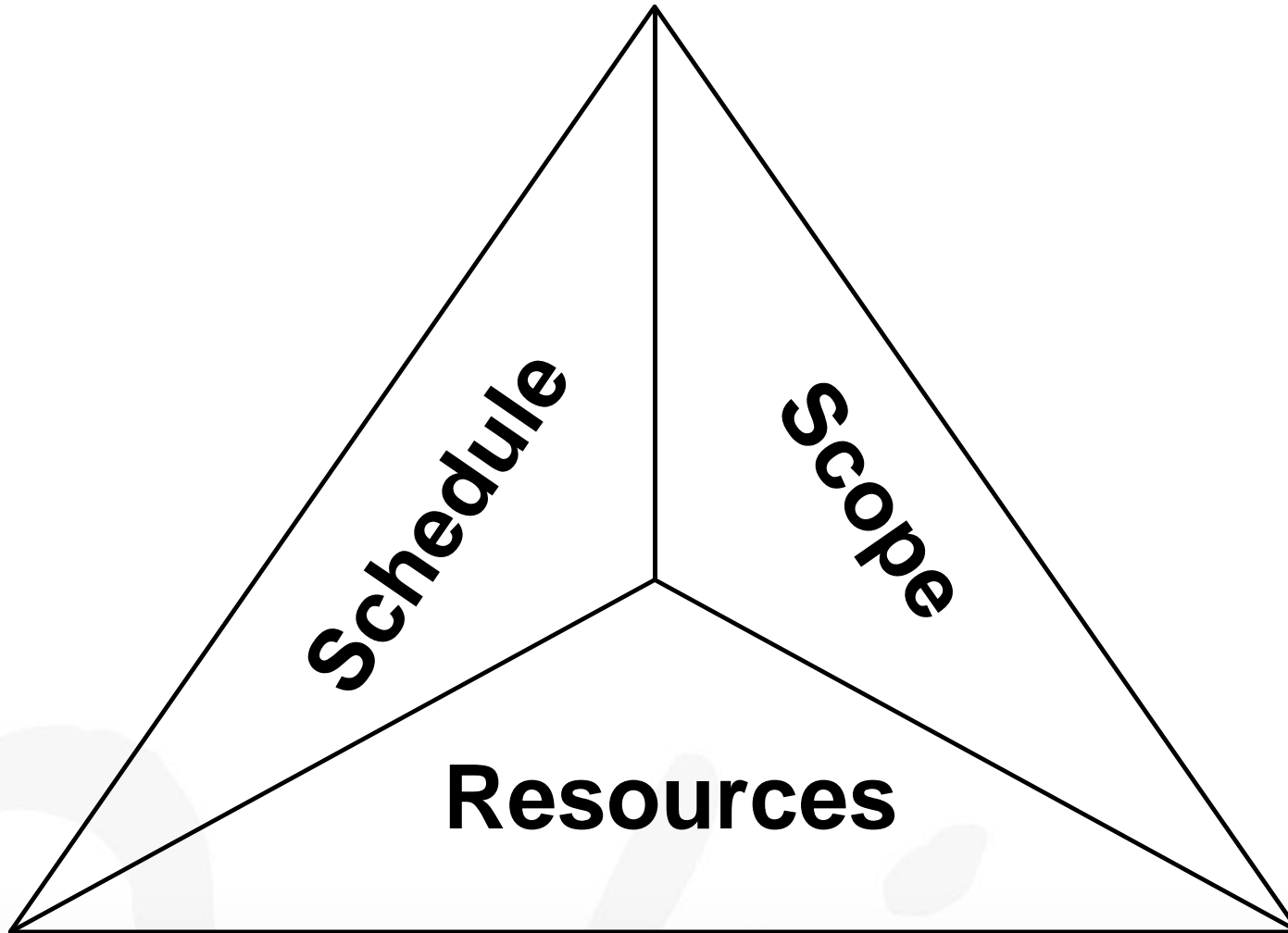
Source:
Gary Smith EDA,
October 2010

- **Gate-level formal (equivalence checking)**
 - Then (1993): Chrysalis; Now: Cadence (Verplex), Synopsys
- **RTL formal (model checking)**
 - Then (1994): Averant, IBM; Now: Jasper, Mentor (0-In)

Formal tool usage in industry

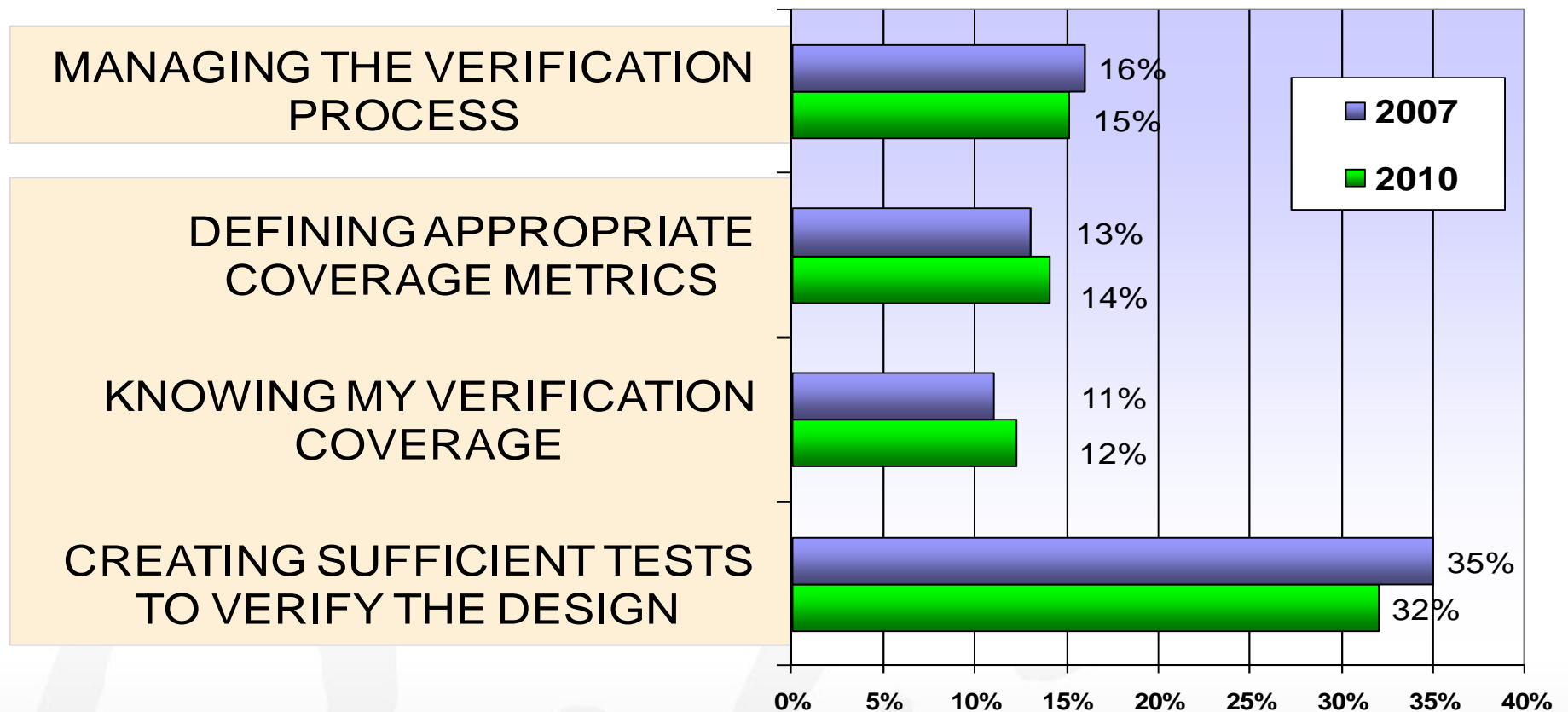
- Around for 20 years
- Expectations has been set high
 - Low efforts for constraints
 - Tools run fast enough
- Expectations have been set low
 - Only verify local assertions
 - No End-to-End proofs
- Perception: low !/\$
- Training and staffing
 - Few places to learn formal application
 - Single user should not do both formal and simulation





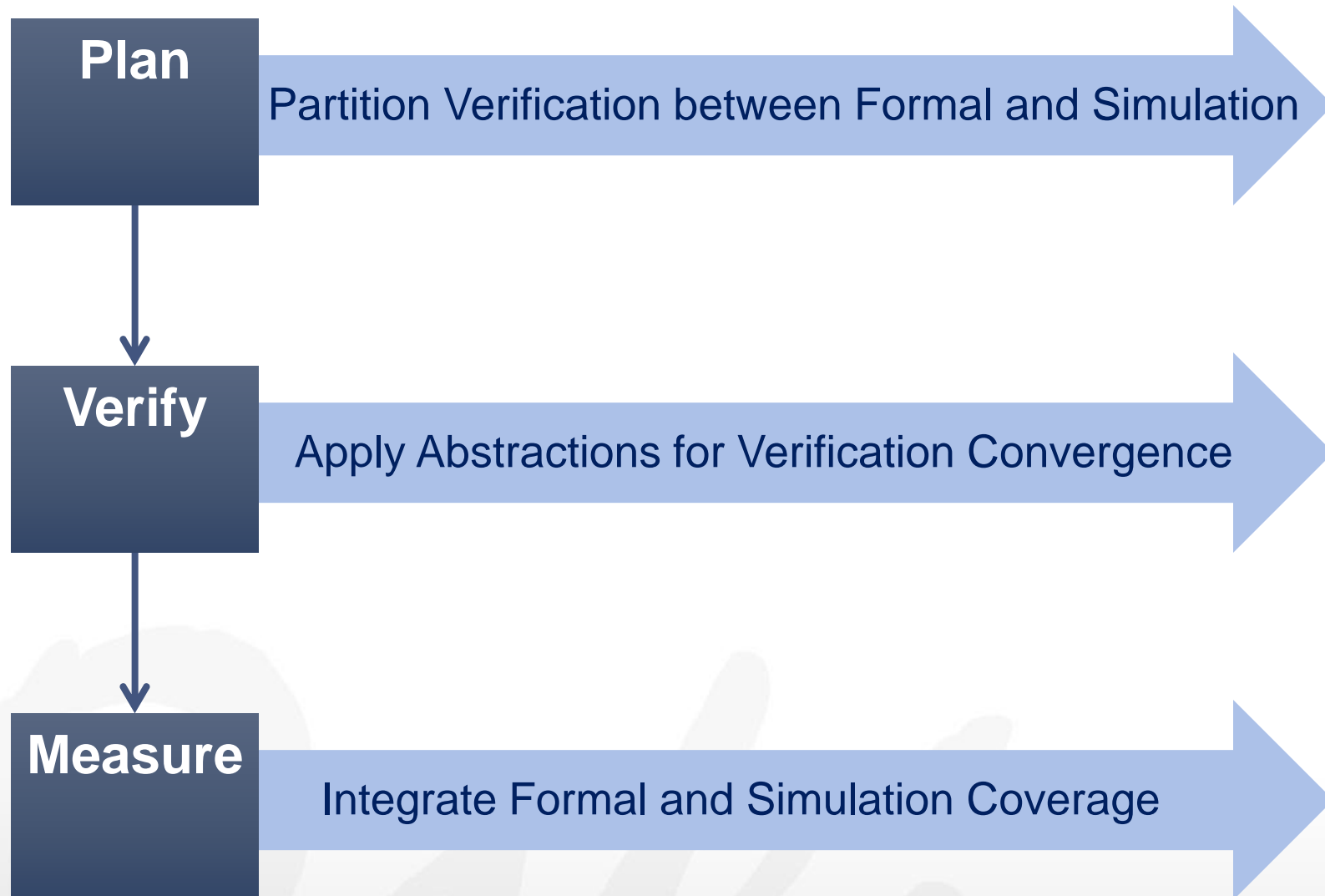
Biggest challenges needing solutions

- Verification management and coverage



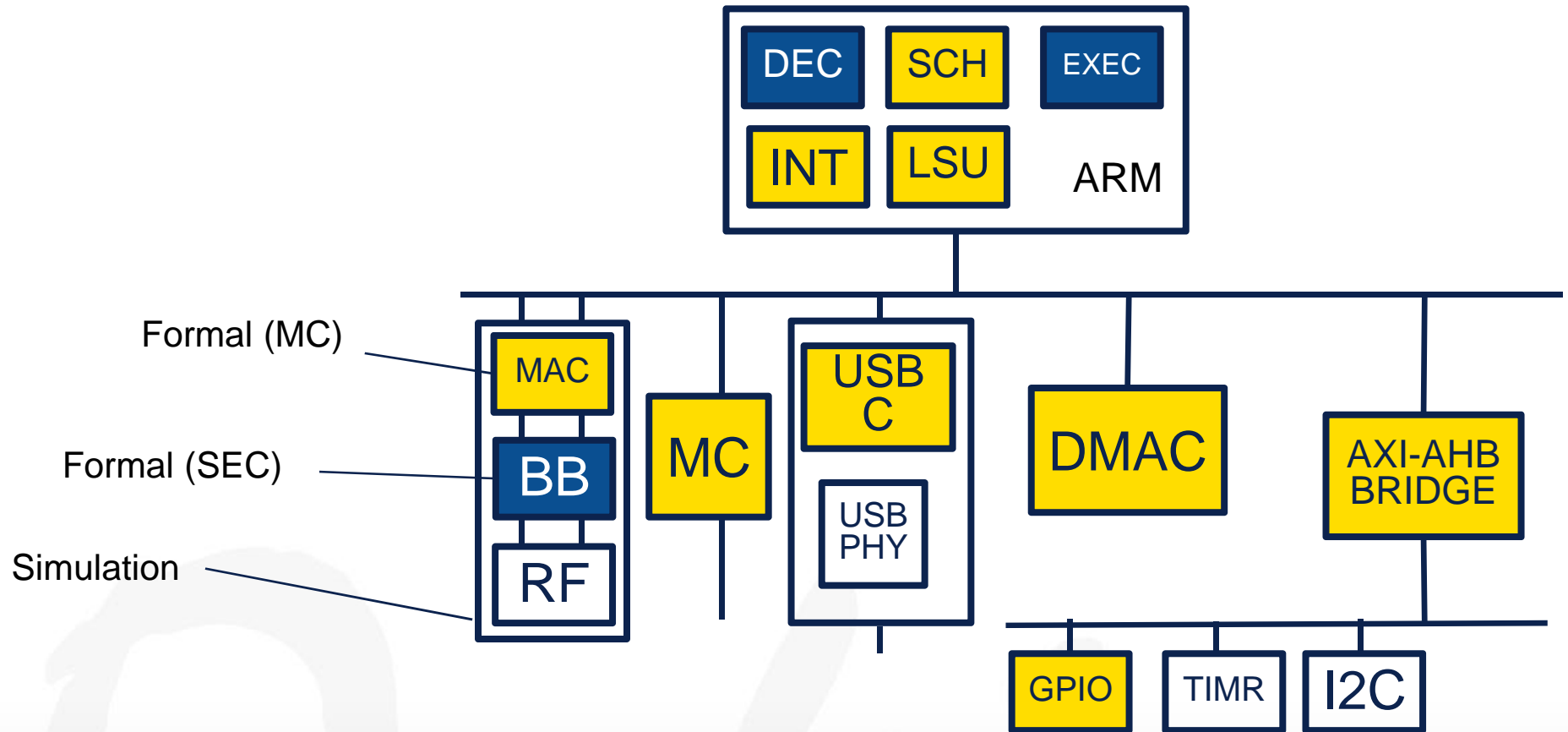
Wilson Research Group and Mentor Graphics
2010 Functional Verification Study, Used with permission.

Achieving verification closure



Formal (MC, SEC*) and simulation strengths

* SEC = Sequential Equivalence Checking (RTL vs C model)



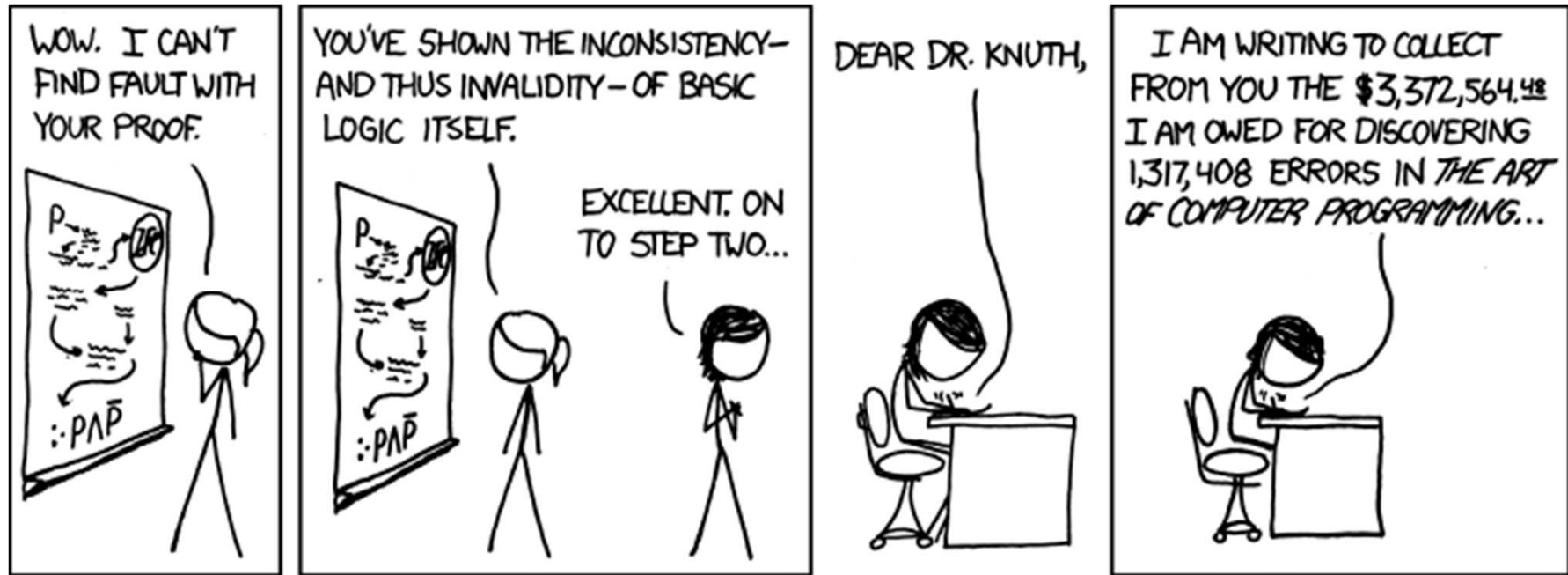
How perfect does formal have to be?



Graphic: MacGregor
Marketing

- Not all bugs need to be found/fixed
- Formal does not need to find the last bug
- Usually bounded proofs are good enough
(if bound is good enough!)
- Formal has to be more cost-effective than the alternative

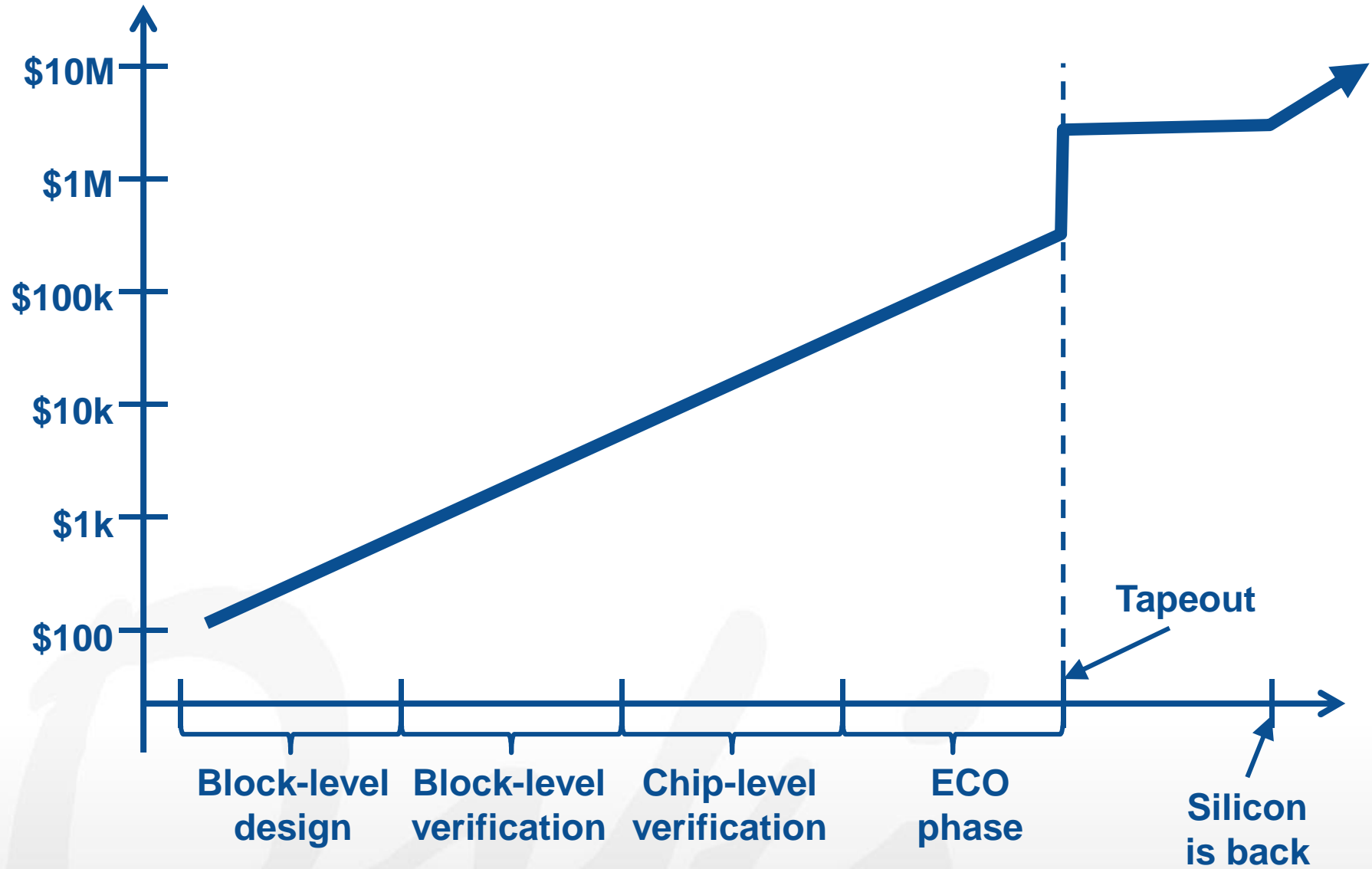
“Pay-per-bug” shows formal ROI

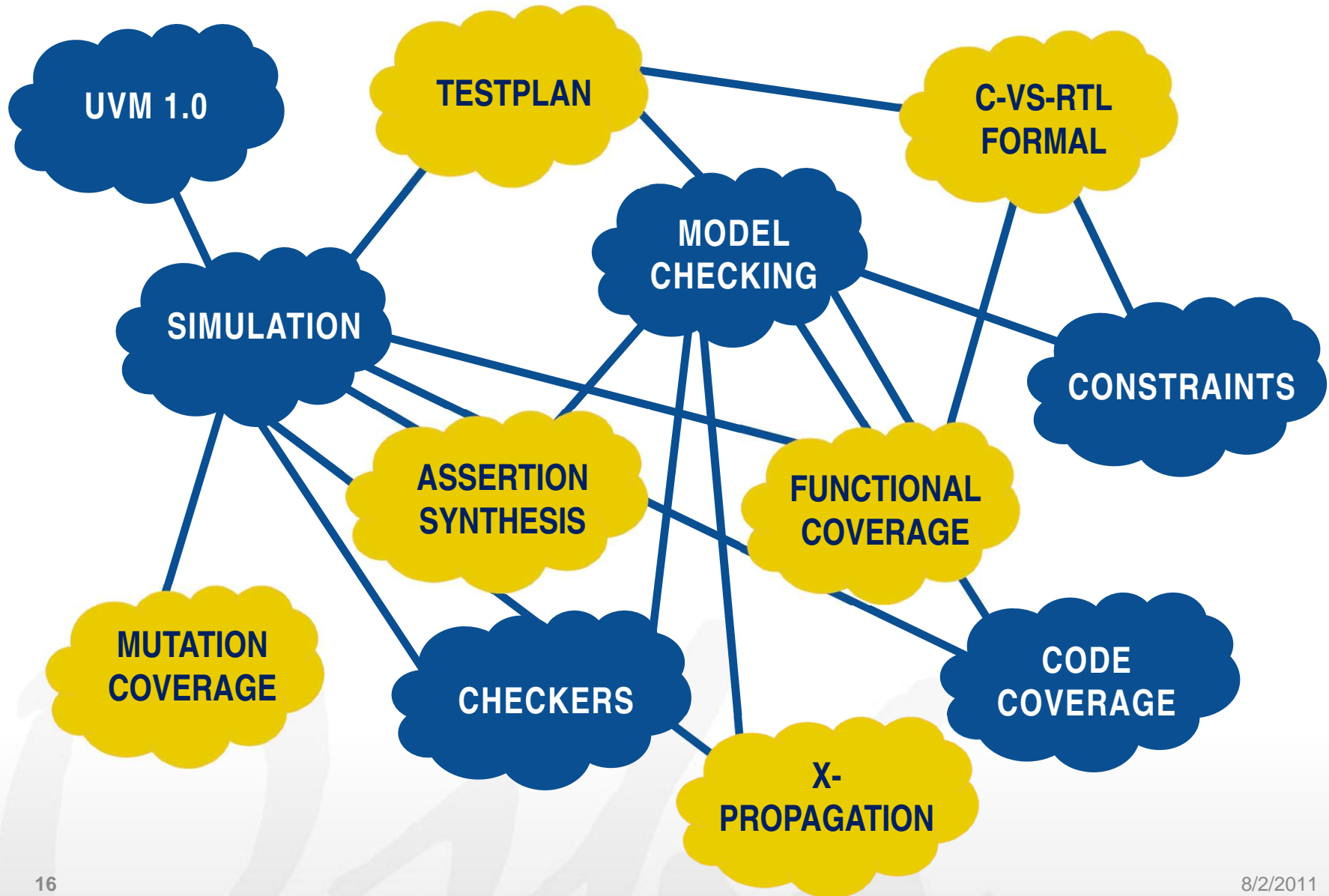


Source: xkcd.com

- First successful business model
 - Pay-per-bug!
- “Bug” is defined as something that violates the specification AND customer decides to fix it
- \$/bug varies, depending on design stage

Bug-fix cost rises exponentially





When do we know we are done?



*“There are **known knowns**; there are things we know we know.*

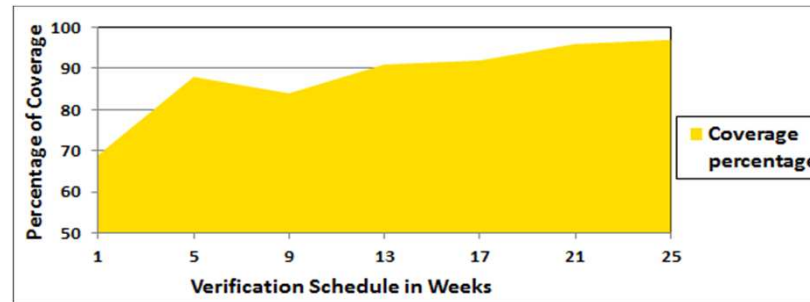
*We also know there are **known unknowns**; [...] we know there are some things we do not know.*

*But there are also **unknown unknowns** – the ones we don't know we don't know.”*

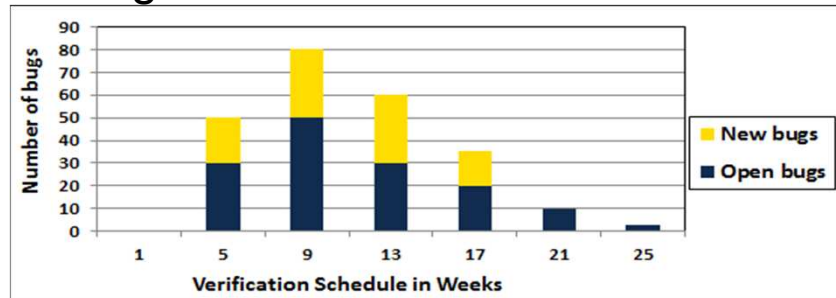
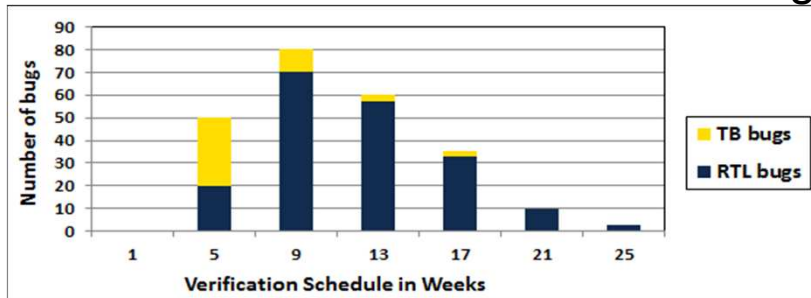
- *Donald Rumsfeld (2002)*
U.S. Secretary of Defense
(on the war in Iraq)

Verification manager's dashboard

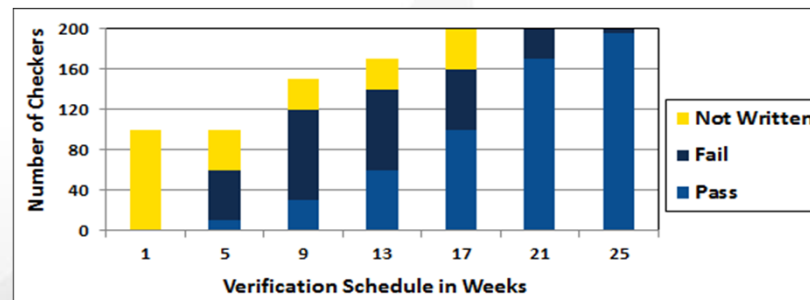
Coverage tracking



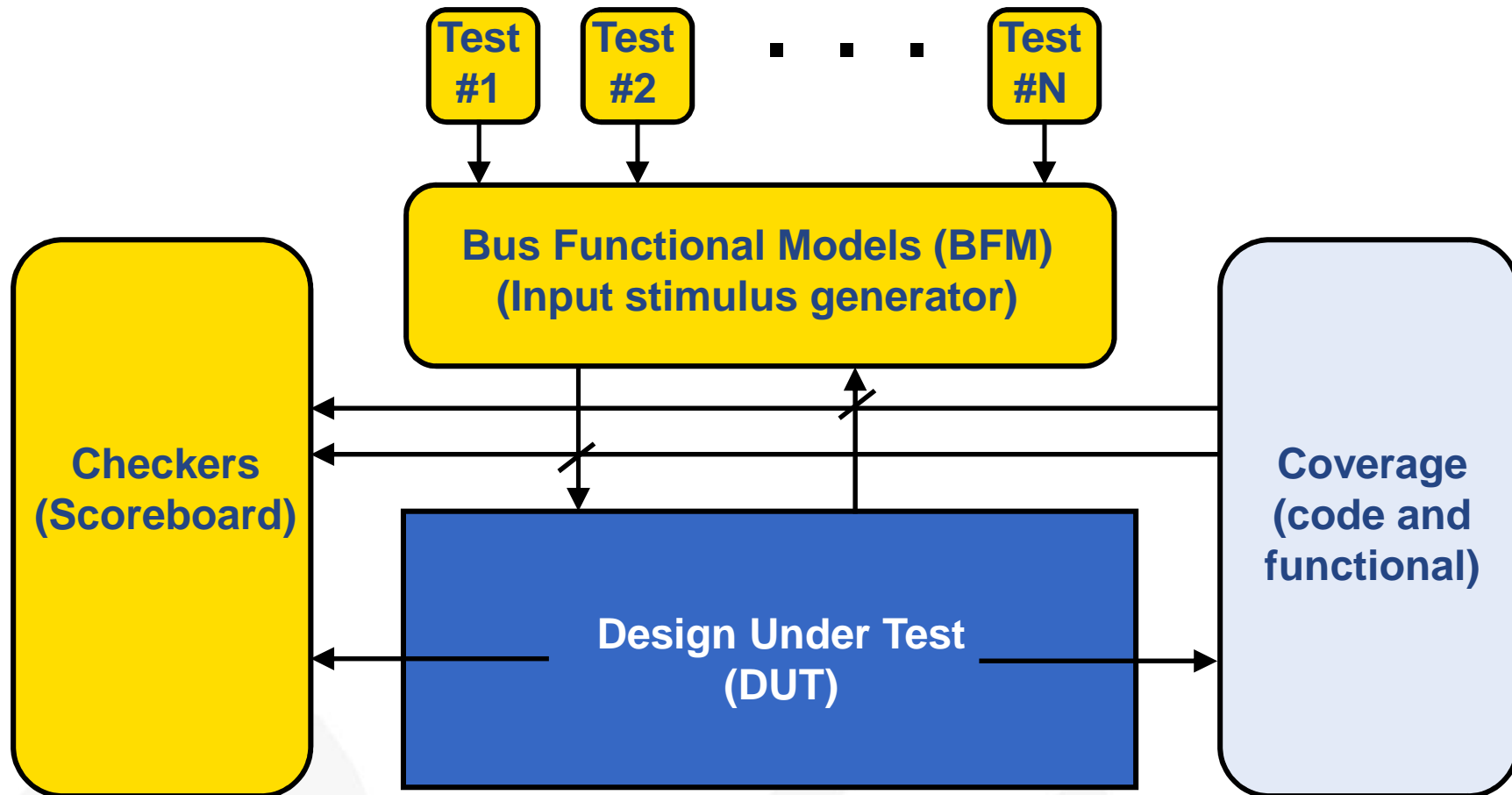
Bug tracking



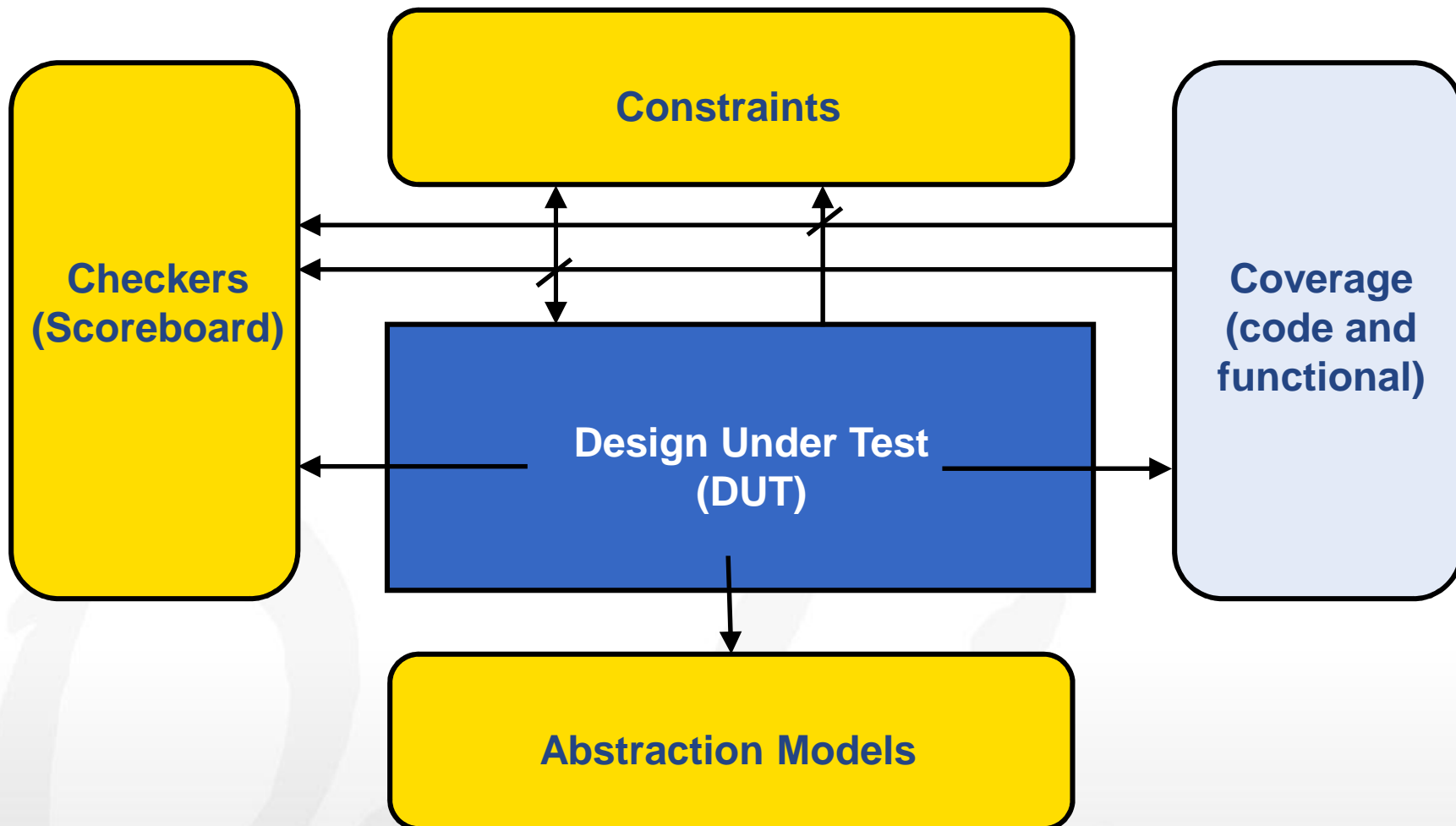
Runtime status



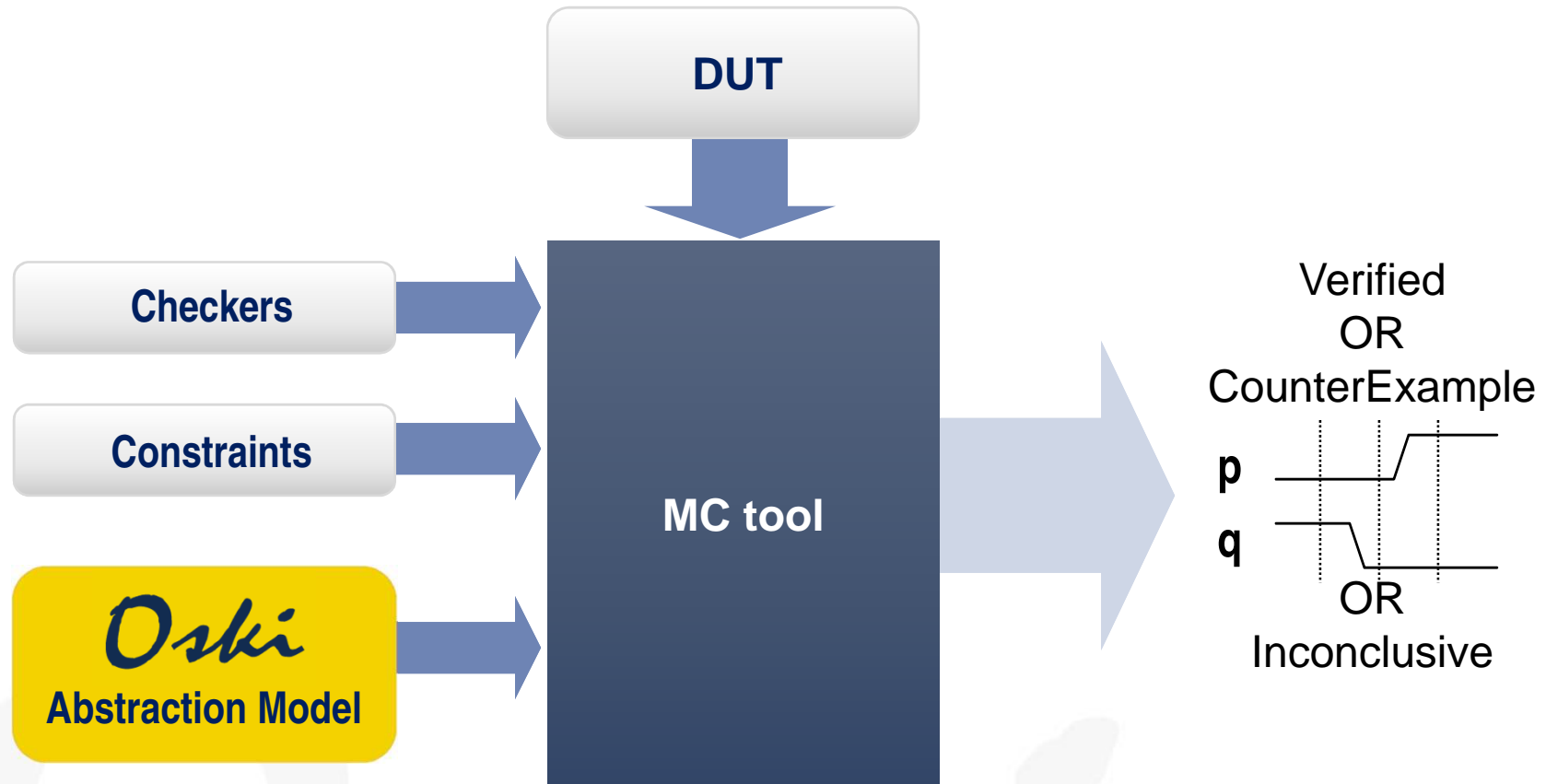
A simulation testbench



A formal testbench

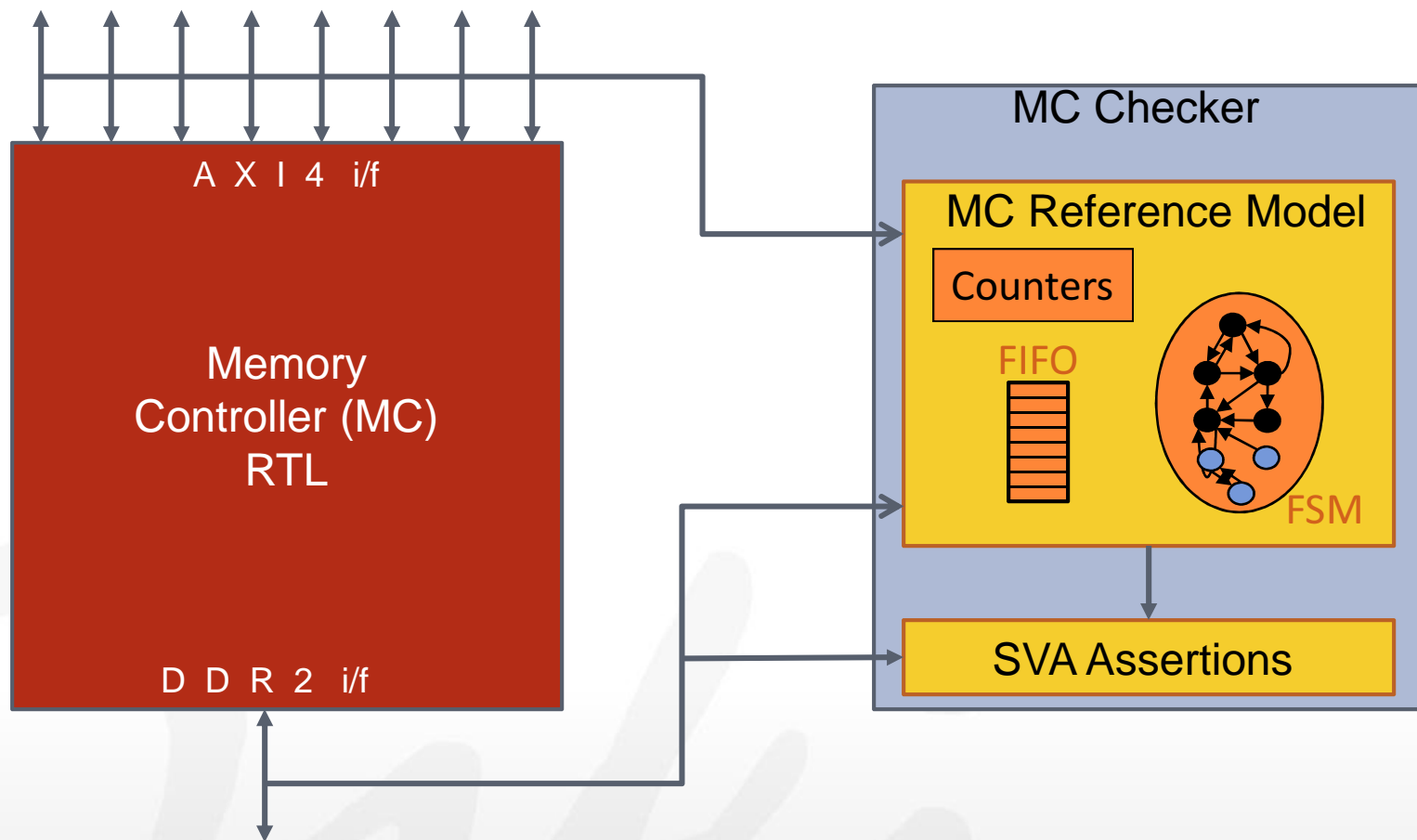


Abstraction Models: accelerate proofs



Checkers (End-to-End)

- For End-to-End formal verification, less than 5% of Checker code is SVA; rest is SV or Verilog
 - (Synthesizable) Reference model is typically as big an effort as the RTL



- Example SVA check:

```
AXI4_AWREADY_stable_a : assert property (
```

```
  @(posedge ACLK) disable iff (ARESETn == 1'b0)  
  (AWVALID && (!AWREADY)) |-> $stable(AWADDR));
```

clocks + reset

property

Antecedent

- Almost all checks (RTL asserts and End-to-End checkers):

```
AXI4_AWREADY_err : assert property (
```

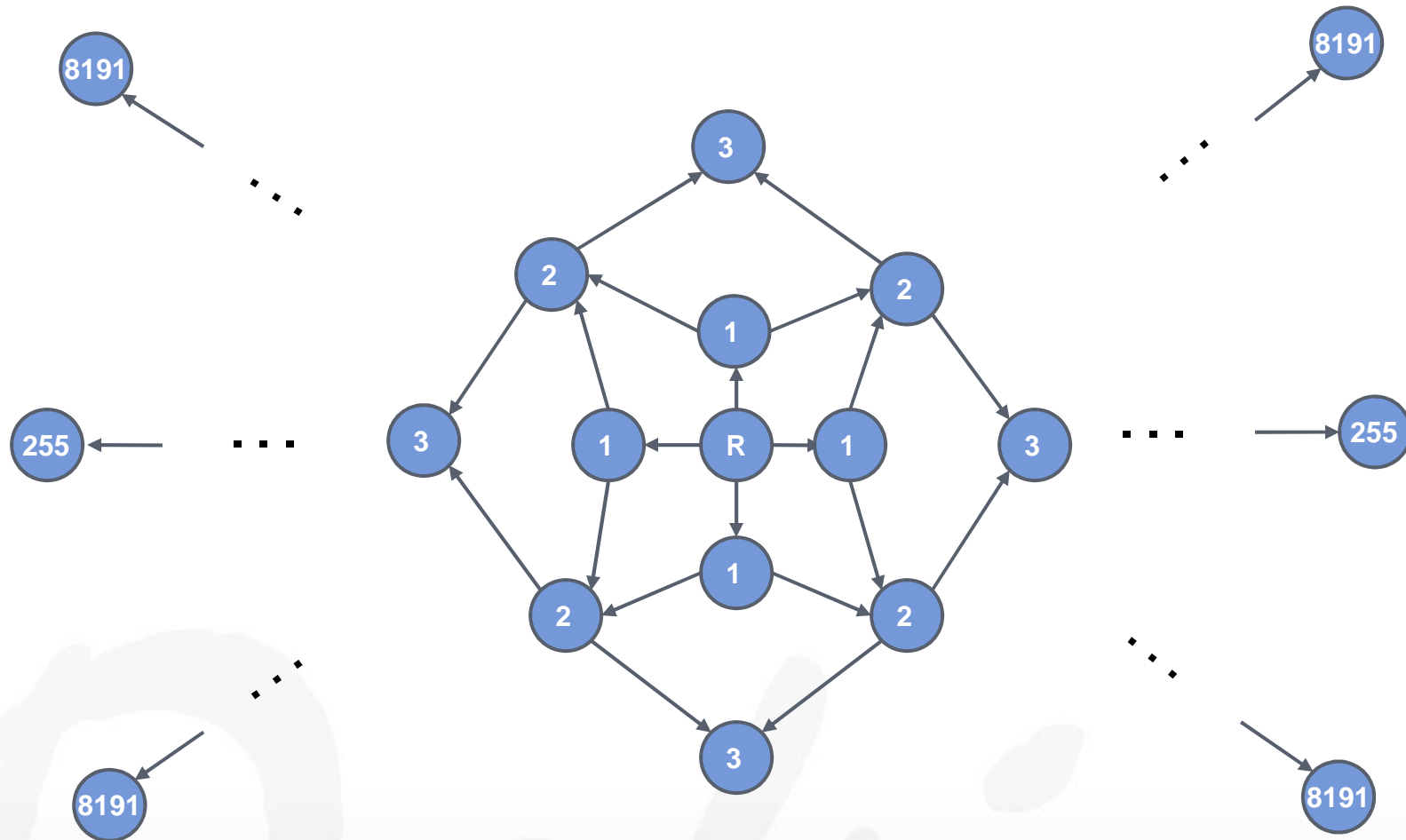
```
  @(posedge ACLK) disable iff (ARESETn == 1'b0)  
  (!AWVALID_err));
```

- Trivial properties, built using complex modeling (Verilog)

NOTE

Property analysis (e.g. vacuity checks) can detect few defects in checkers

State space complexity

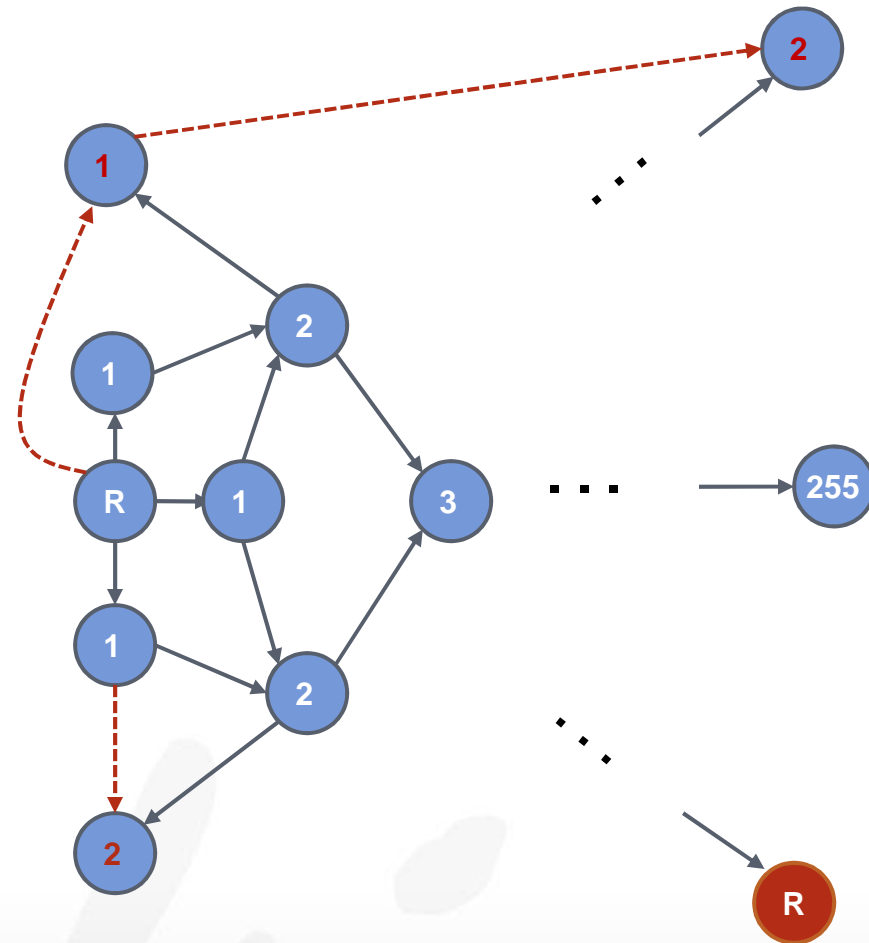


Abstractions (to manage complexity)

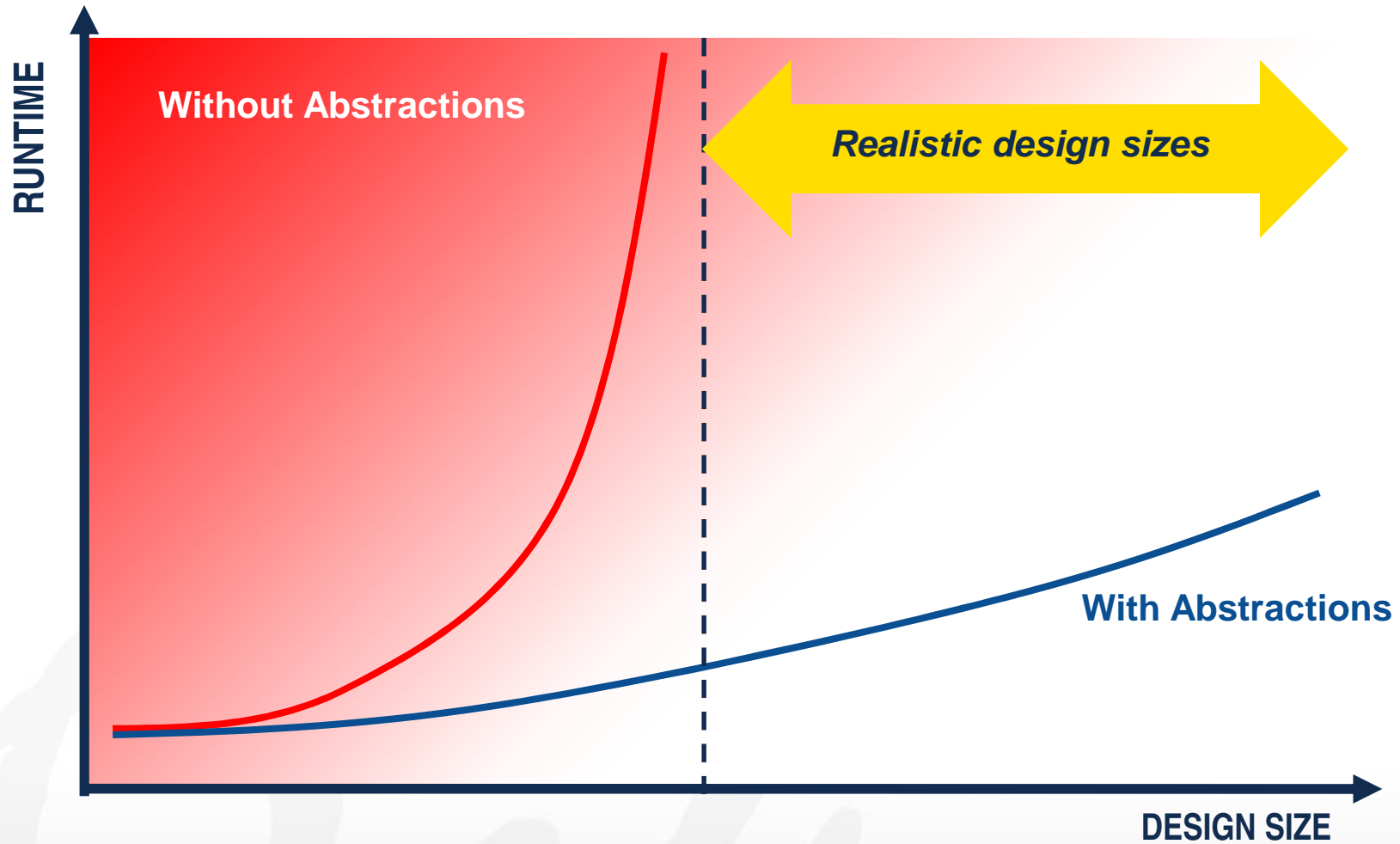
- An “Abstraction” of a design is a design that has a superset of the design behavior
- Useful to overcome complexity barriers
 - Smaller Cone-of-Influence
 - Shallower search space
 - Ability to skip long initialization sequences
- Cannot give a false positive
- Can give a false negative (Fail), but...
 - You get a trace to determine the reason for the negative

Complexity (and Abstractions)

- Effect of abstractions:
 - Reduces state space
 - Adds state transitions
 - Adds Reset states



Overcoming complexity with Abstractions

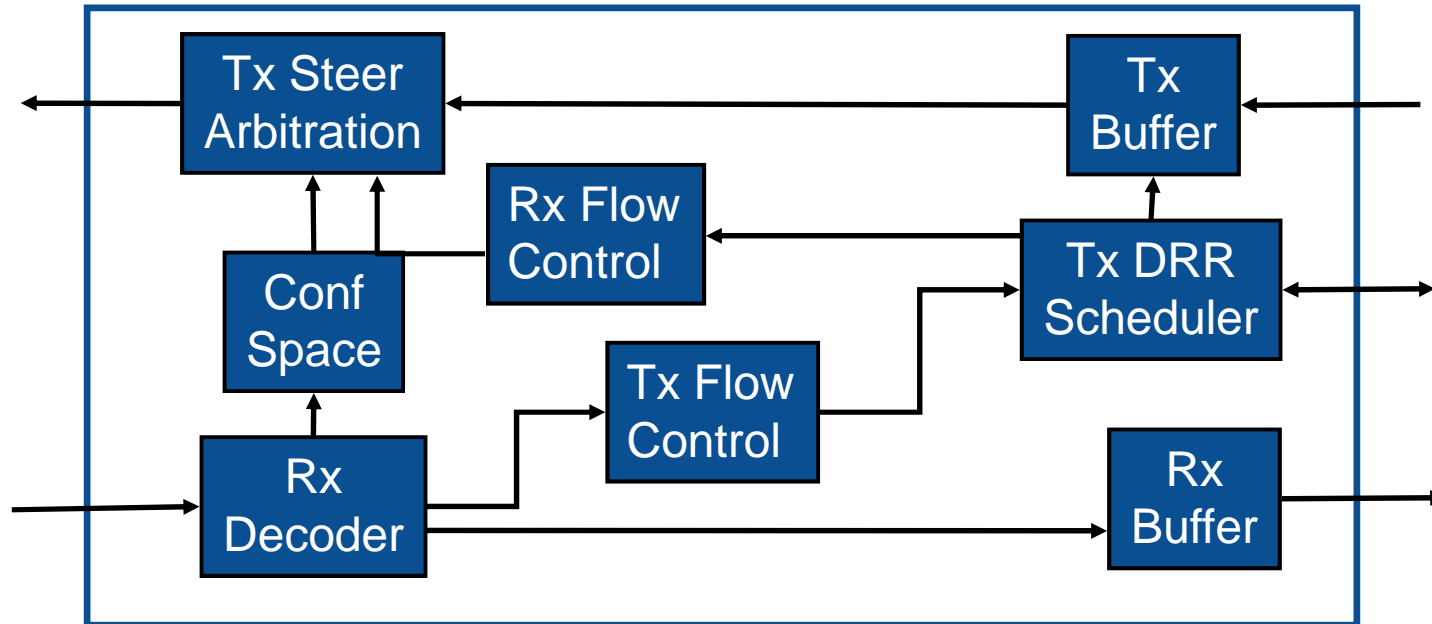


Examples of Abstractions

- Allow DUT to reset to a deep state
 - BANKS_IDLE state for a memory controllers (skips thousands of clocks of initialization)
- Replace memory by a memory model tracking a specific Byte of a specific Beat of a specific Transaction
 - 13th transaction, 243rd beat, 1st Byte
- Replace a Tag Generator by an abstract model
 - Reduce sequential depth by tracking specific value
 - Example in the next few slides...

- Other example of Abstraction Models:
 - Localization
 - Datapath
 - Memory
 - Sequence
 - Counter
 - Floating pulse
- Without Abstraction Models:
 - On most interesting designs, formal tools do not search far enough

Example: PCIe Transaction Layer



- 128-bit datapath
- 8 VCs
- History, policy-dependent DRR scheduler
- Conf responses arbitrated with TLPs and FC DLLPs
- Tx, Rx Buffers can store multiple TLPs (upto 32kb each)

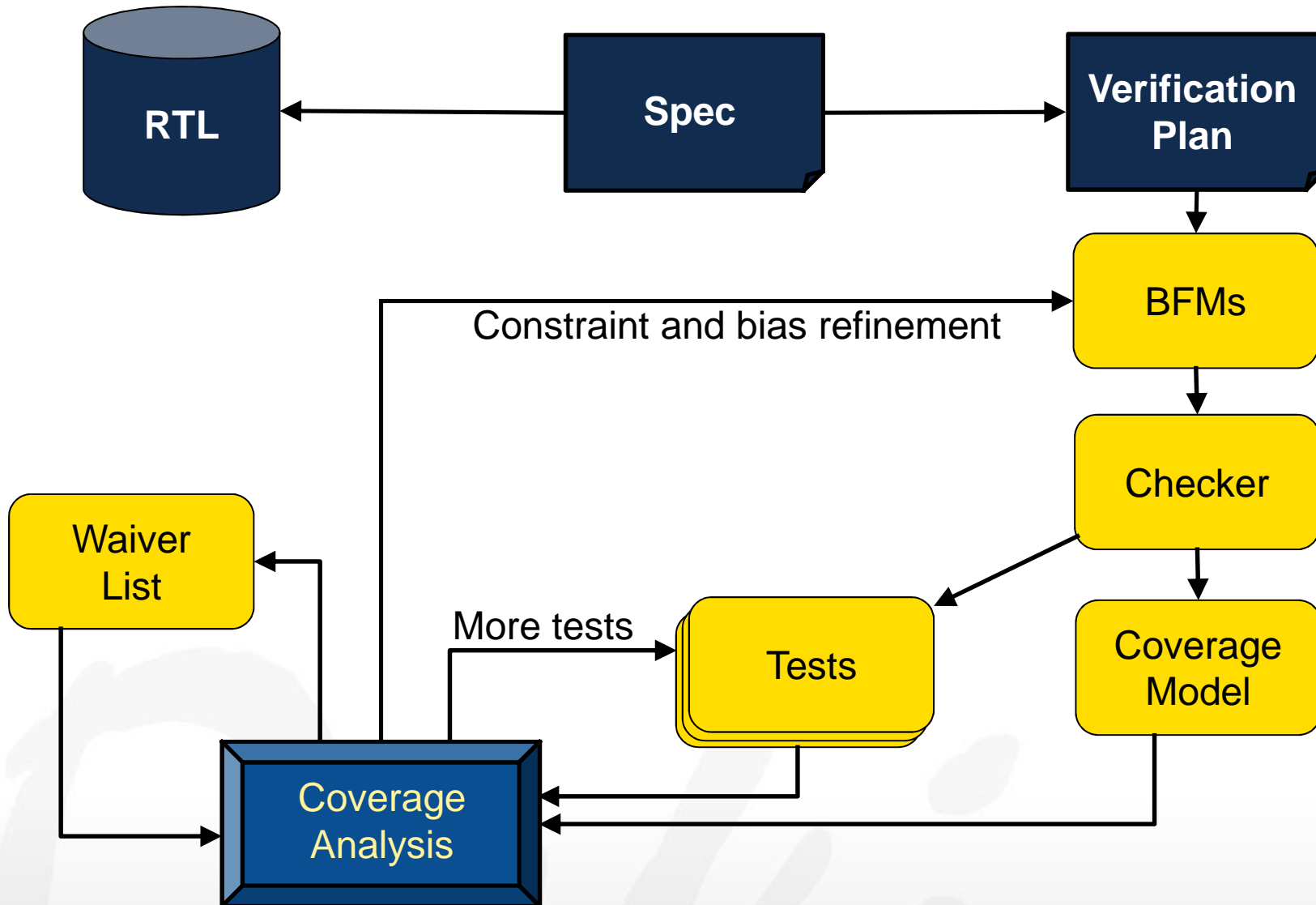
*Verification closure with formal
and simulation*

OSKI TECHNOLOGY, INC.

Unique Methodology. Highest Coverage. Fastest Time to Market.



Coverage-driven simulation methodology



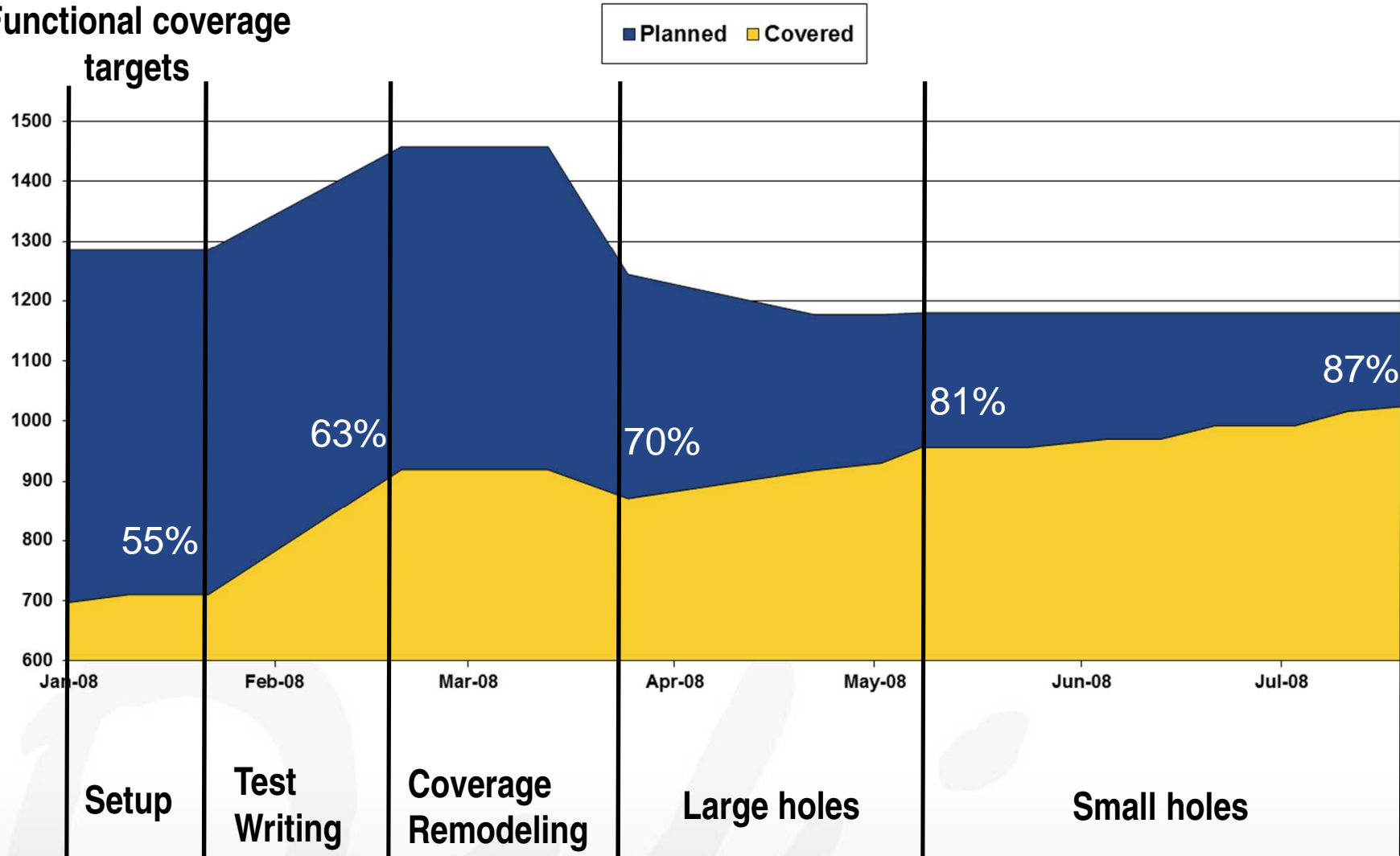
Coverage for hardware designs

- Trivial to get to 60-70% code coverage
- 100% line/expression coverage often required for tapeouts
 - Manual waivers are allowed
- NVIDIA SNUG 2011 paper
 - 270 man weeks to do waiver analysis for one design
 - 180 man weeks to write missing tests

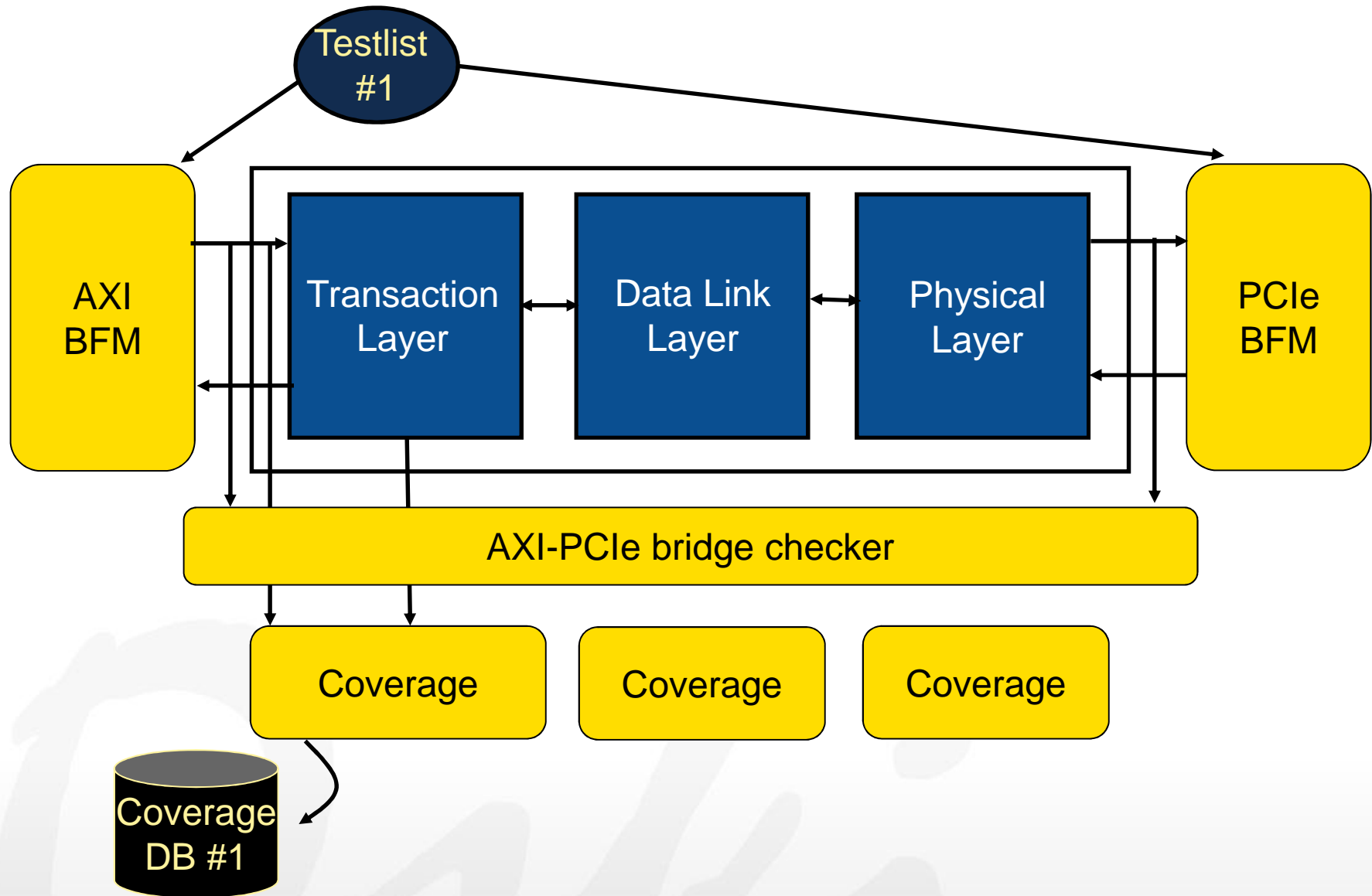
Coverage closure phases



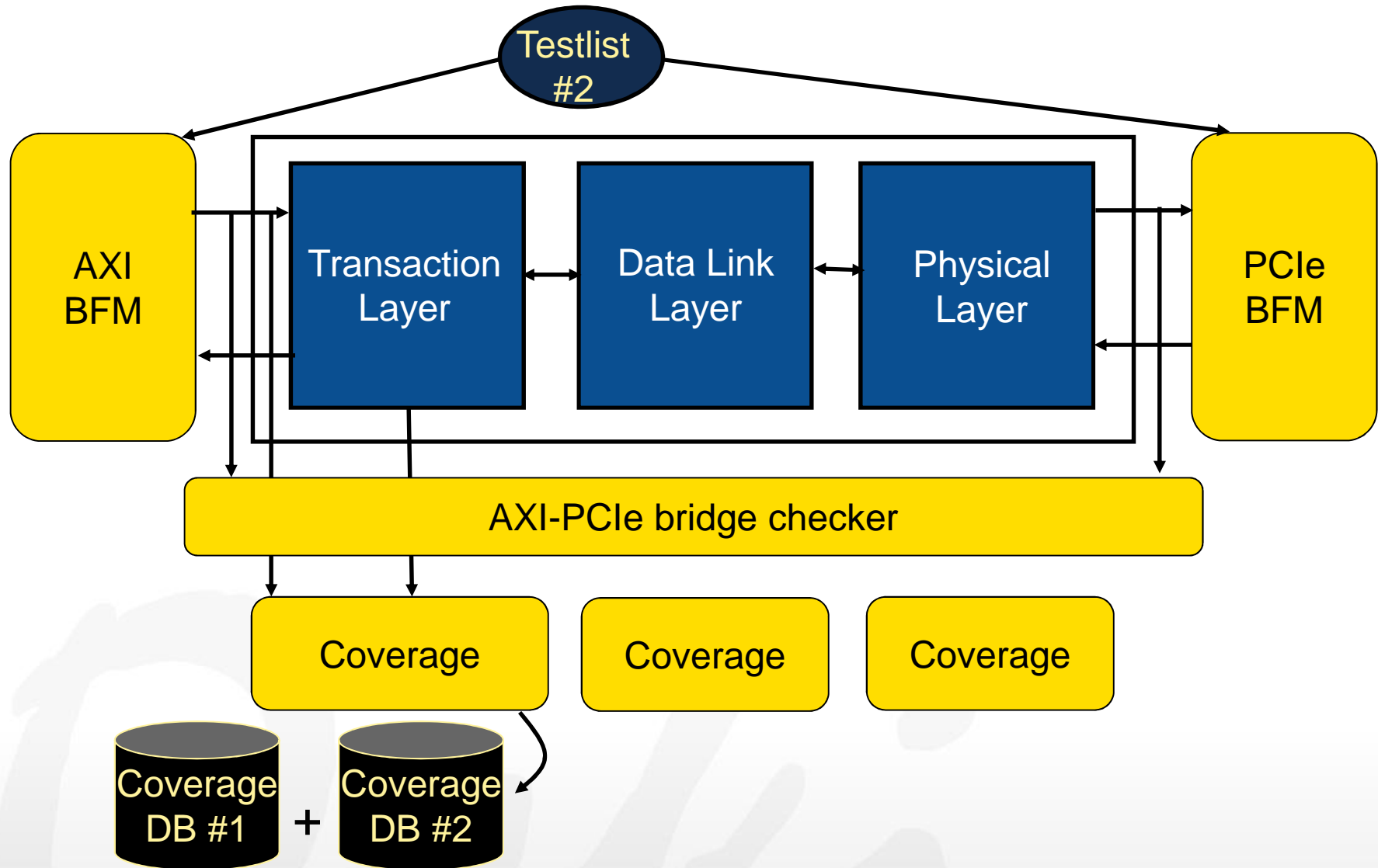
Functional coverage targets



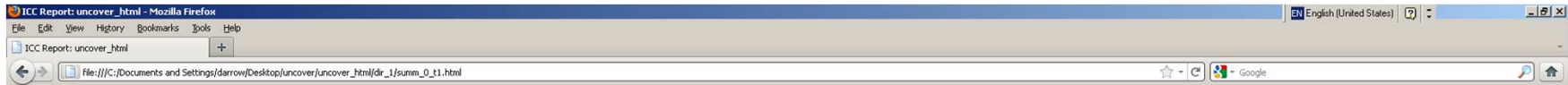
Coverage database collection



Coverage database collection



Coverage reporting



Coverage Summary Report, Instance-Based

Top Level Summary

[Legend and Help](#)

Instance name: mic
Module/Entity name: mic

Total	Block	Expression	Toggle	FSM	Assertion	Name
82%	95% (172/180)	100% (3/3)	96% (2929/3056)	100% (24/24)	20% (1/5)	Cumulative
97%	No Items	No Items	97% (412/424)	No Items	No Items	Self

Coverage of immediate sub-instances:

Total	Block	Expression	Toggle	FSM	Assertion	Name
98%	100% (73/73)	No Items	96% (2032/2121)	No Items	No Items	mic_fifo_0
58%	91% (29/32)	No Items	82% (52/63)	No Items	0% (0/2)	mic_arb_0
75%	96% (50/52)	100% (3/3)	79% (19/24)	100% (24/24)	0% (0/2)	fifo_state_0
85%	70% (7/10)	No Items	100% (264/264)	No Items	No Items	mux8_0
97%	100% (8/8)	No Items	92% (100/109)	No Items	100% (1/1)	memctl_0
99%	100% (5/5)	No Items	98% (50/51)	No Items	No Items	sram_0





“The perfect is the enemy of good”

-Voltaire (1772)

- Coverage is not perfect
 - Bugs are missed even with 100% coverage
- But...
 - Helps measure progress
 - Helps identify blind-spots

Input vs Observable coverage

- “Have I verified enough input sequences” (Input coverage)
- “Is my set of checkers complete enough” (Observable coverage)
- Same two notions apply for both simulation AND formal
 - Bounded model checking (BMC) is the most used formal technique

NOTE

Formal does not verify all possible input sequences

Coverage on RTL designs

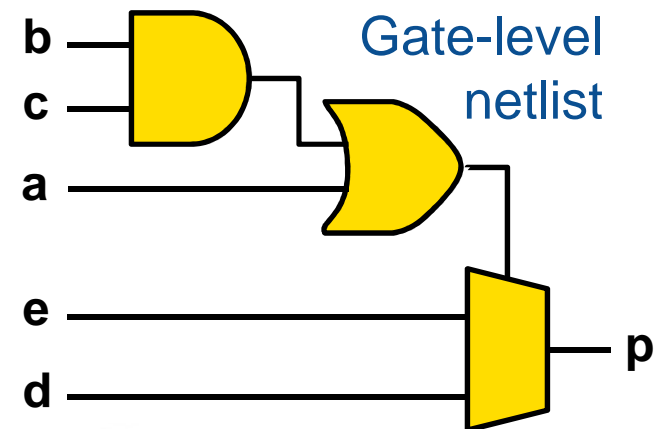
RTL (Verilog)

```
1. reg p;  
2. always @(*) begin  
3.   if (a || (b && c))  
4.     p = d;  
5.   else  
6.     p = e;  
7. end
```

Equivalent RTL

```
1. reg w, p;  
2. always @(*) begin  
3.   w = a || (b && c);  
4. end  
5. always @ (*) begin  
6.   p = (w && d) || ((!w) && e);  
7. end
```

Synthesis



Input Coverage: line/expression coverage



```
1. reg p;  
2. always @(*) begin  
3.   if (a || (b && c))  
4.     p = d;  
5.   else  
6.     p = e;  
7. end
```

Line coverage

a	b	c	p
0	0	0	e
0	0	1	e
0	1	0	e
0	1	1	d
1	0	0	d
1	0	1	d
1	1	0	d
1	1	1	d

target #1

target #2

Expression coverage

a	b	c	p
0	0	0	e
0	0	1	e
0	1	0	e
0	1	1	d
1	0	0	d
1	0	1	d
1	1	0	d
1	1	1	d

#1

#2

#3

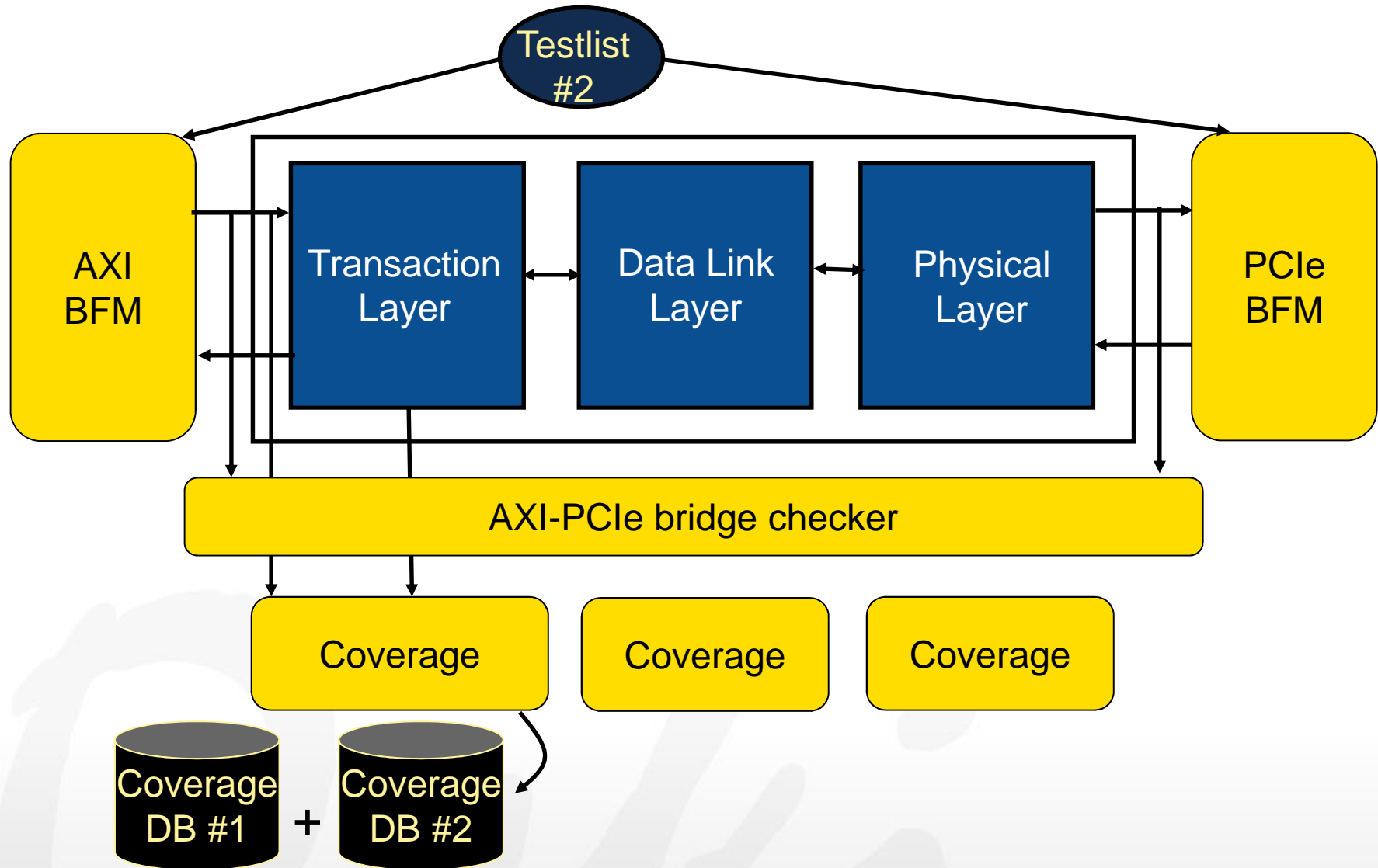
#4

Is my formal complete?

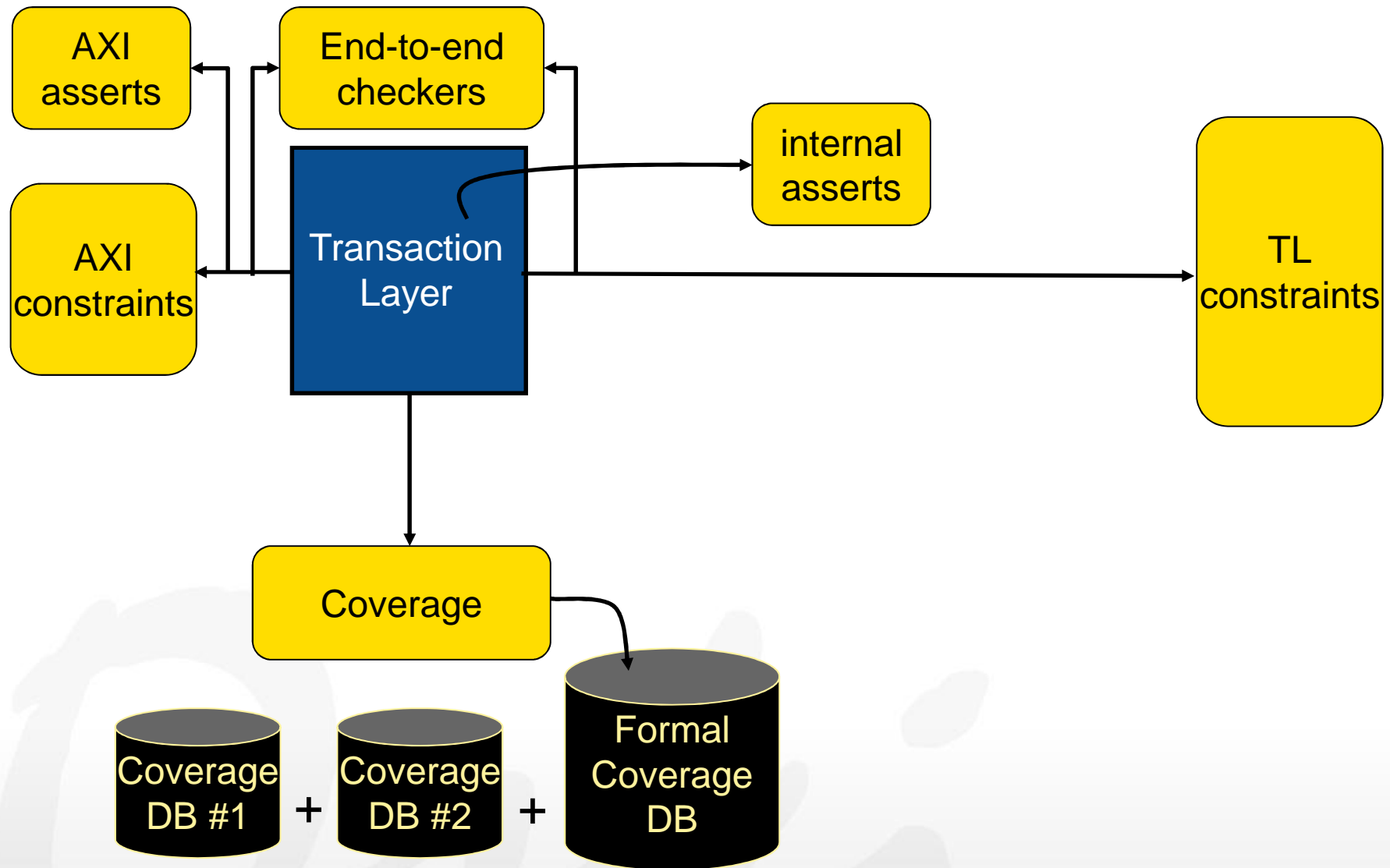
- Are my Checkers complete?
- Are my Constraints complete?
- Is my Complexity strategy complete?

- Constraints: Environment may be over-constrained
 - Intentional: avoided some hard to model or verify input combinations
 - Unintentional: bugs in constraints; forgot to remove intentional over-constraints
- Complexity: All checkers are verified up to proof depth N
 - Any target, not reachable in N clocks, is not covered
- Checkers: does not verify completeness of Checkers
 - No different than simulation!

Coverage database collection



Formal coverage integrated with simulation



Observable Coverage (using mutations)

```
1. reg p;  
2. always @(*) begin  
3.   if (a || (b && c))  
4.     p = d;  
5.   else  
6.     p = e;  
7. end
```

```
1. reg p;  
2. always @(*) begin  
3.   if (a || (b && c))  
4.     p = 1'bX;  
5.   else  
6.     p = e;  
7. end
```

Mutant for line#4

```
1. reg p;  
2. always @(*) begin  
3.   if (a || (b && c))  
4.     p = d;  
5.   else  
6.     p = 1'bX;  
7. end
```

Mutant for line#6

Observable Coverage for formal

```
1. reg p;  
2. always @(*) begin  
3.   if (a || (b && c))  
4.     p = d;  
5.   else  
6.     p = e;  
7. end
```

```
1. reg p;  
2. always @(*) begin  
3.   if (a || (b && c))  
4.     p = <primary_input>;  
5.   else  
6.     p = e;  
7. end
```

Mutant for line#4

```
1. reg p;  
2. always @(*) begin  
3.   if (a || (b && c))  
4.     p = d;  
5.   else  
6.     p = <primary_input>;  
7. end
```

Mutant for line#6

- Formal Coverage must fit with Simulation Coverage
 - Same metrics, same meaning
- Formal verification in practice:
 - BMC is the primary workhorse of practical formal verification
 - Checkers are complex Verilog, simple SVA
 - Abstraction Models are key to increasing formal coverage

Thanks



- Prashant Aggarwal
- Adnan Aziz
- Philippa Slayton