End-to-End Formal using Abstractions to Maximize Coverage

PRASHANT AGGARWAL
OSKI TECHNOLOGY
DARROW CHU
CADENCE DESIGN SYSTEMS
VIJAY KADAMBY
CISCO
VIGYAN SINGHAL
OSKI TECHNOLOGY

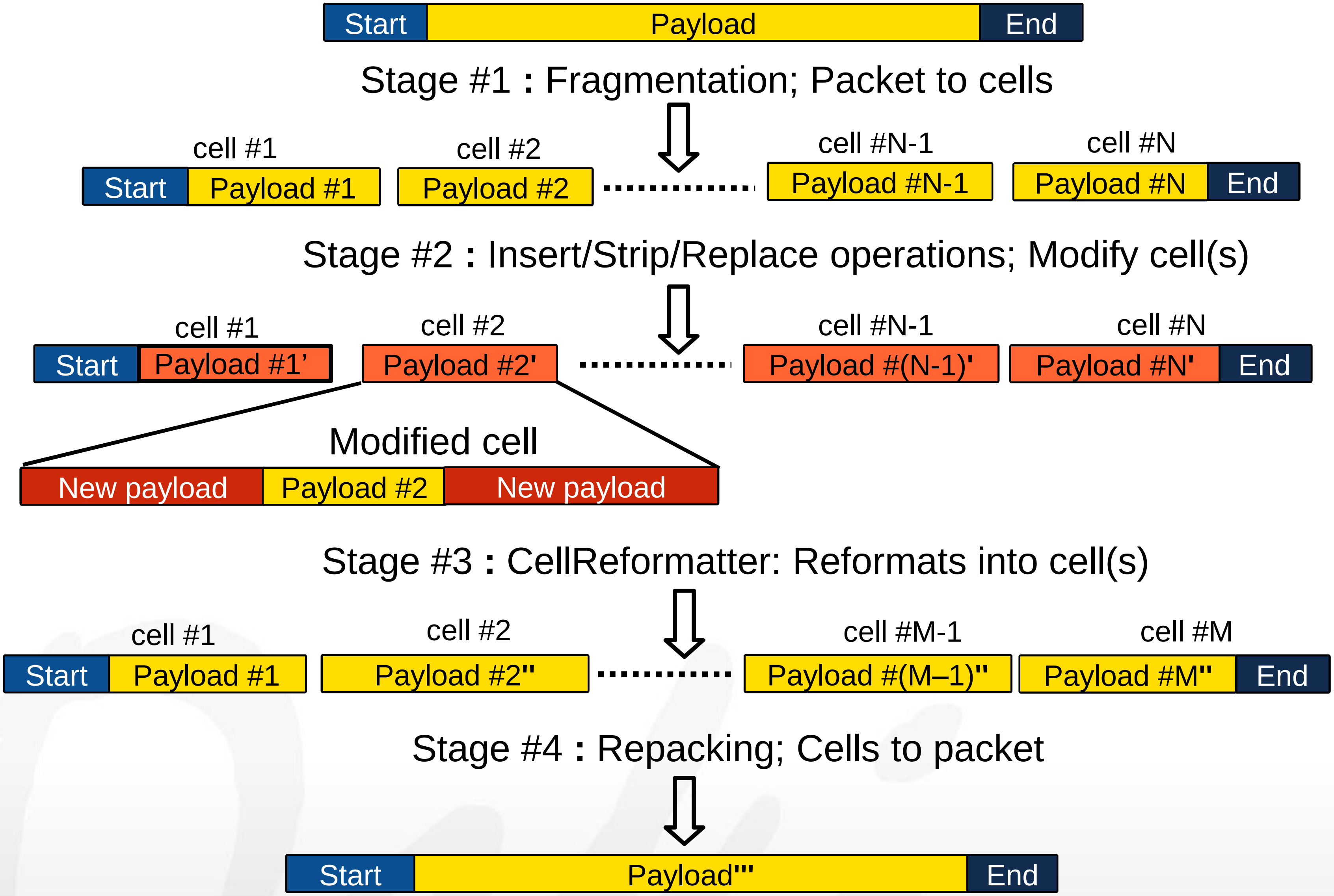
Agenda

Cover three topics using a real design in a simulation setting

- End-to-end formal
- Abstractions to achieve convergence
- Coverage to measure completeness

The design

The design: Packet rewrite module (PRM)



Start | Payload | End

Stage #1 : Fragmentation; Packet to cells

cell #1: Start | Payload #1 — cell #2: Payload #2 .... cell #N-1: Payload #N-1 — cell #N: Payload #N | End

Stage #2 : Insert/Strip/Replace operations; Modify cell(s)

cell #1: Start | Payload #1' — cell #2: Payload #2' .... cell #N-1: Payload #(N-1)' — cell #N: Payload #N' | End

Modified cell

New payload | Payload #2 | New payload

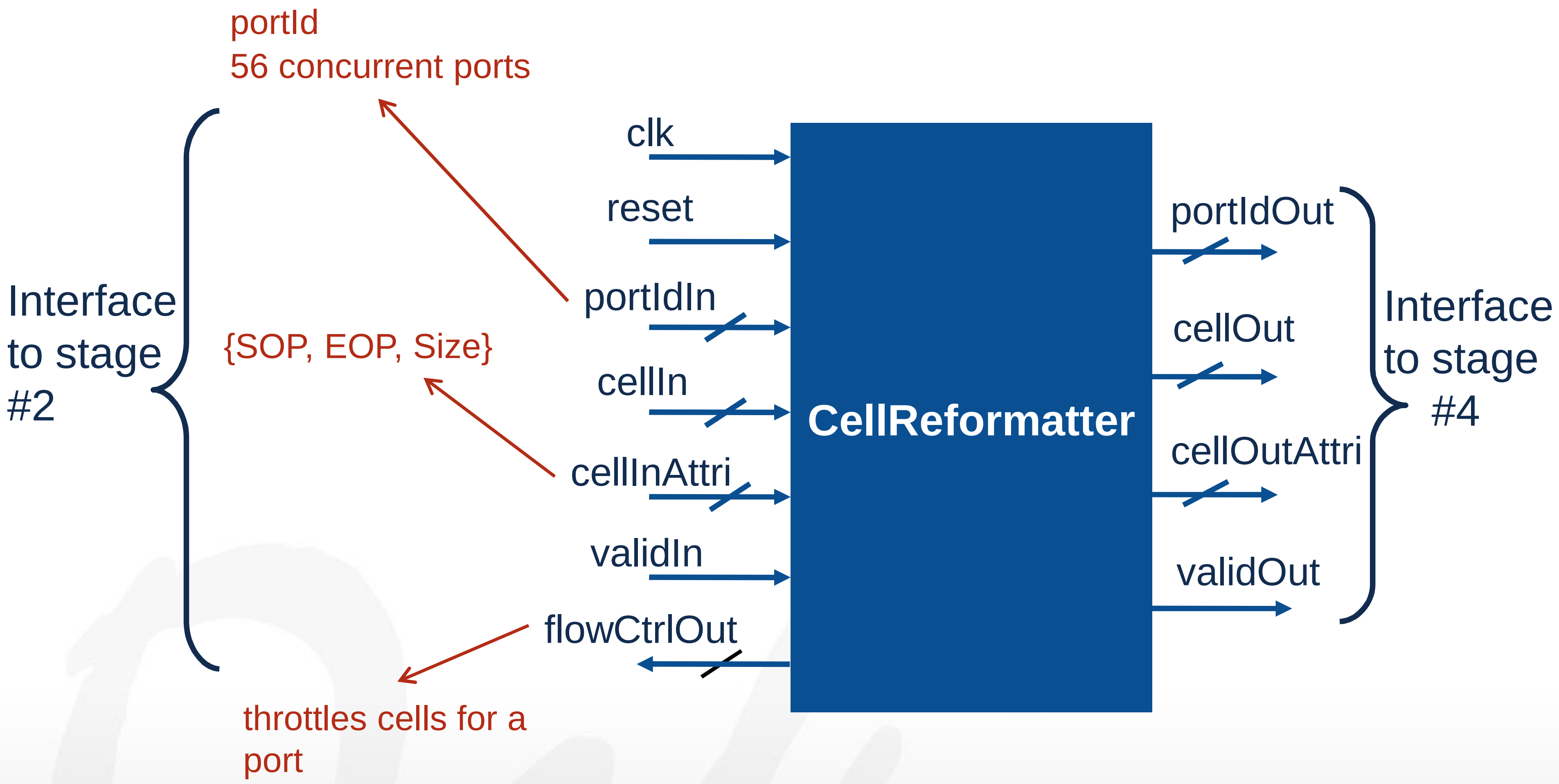
Stage #3 : CellReformatter: Reformats into cell(s)

cell #1: Start | Payload #1 — cell #2: Payload #2" .... cell #M-1: Payload #(M–1)" — cell #M: Payload #M" | End

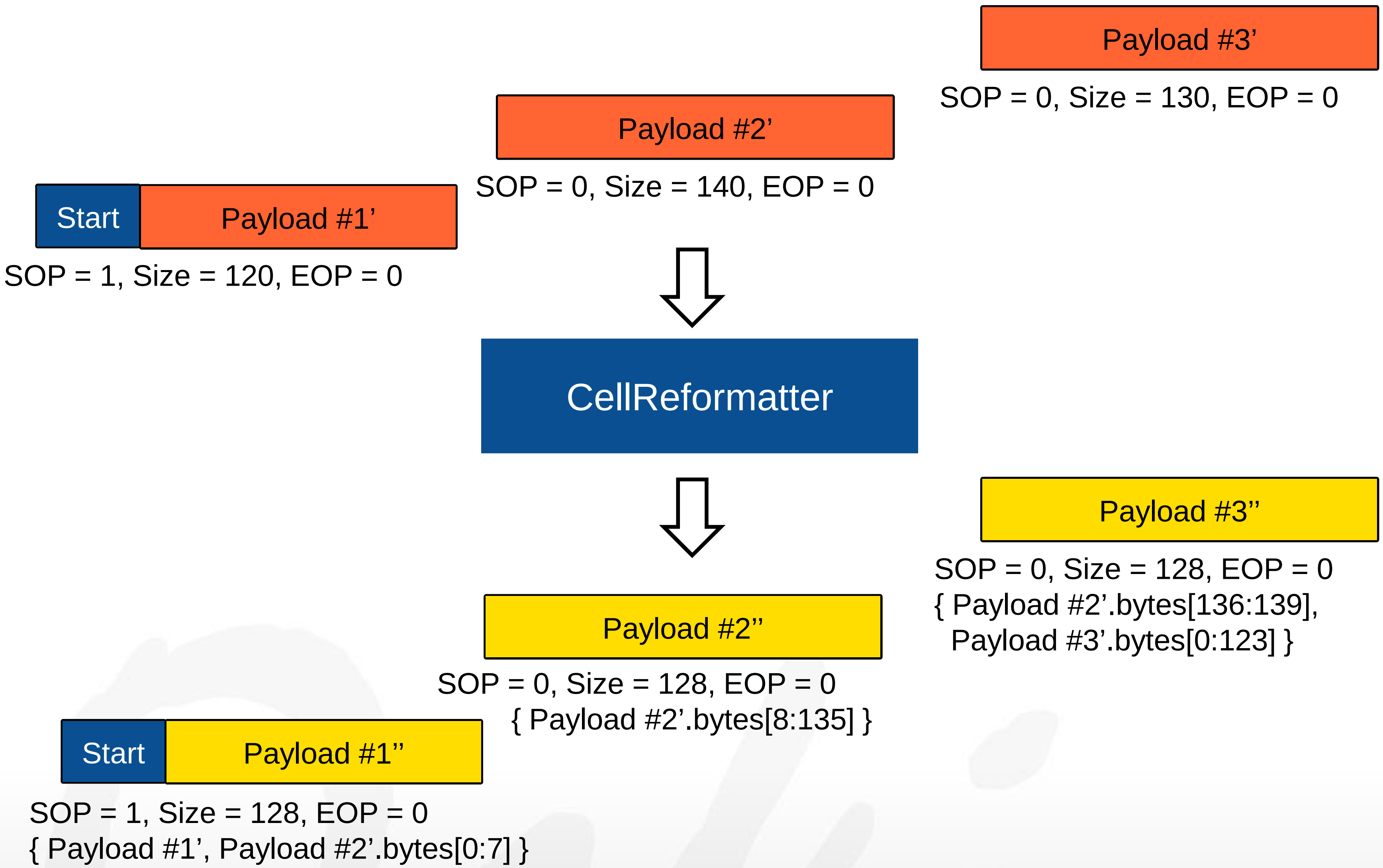
Stage #4 : Repacking; Cells to packet

Start | Payload''' | End

CellReformatter inputs/outputs



CellReformatter in action



Start | Payload #1’

SOP = 1, Size = 120, EOP = 0

Payload #2’

SOP = 0, Size = 140, EOP = 0

Payload #3’

SOP = 0, Size = 130, EOP = 0

CellReformatter

Start | Payload #1’’

SOP = 1, Size = 128, EOP = 0
{ Payload #1’, Payload #2’.bytes[0:7] }

Payload #2’’

SOP = 0, Size = 128, EOP = 0
{ Payload #2’.bytes[8:135] }

Payload #3’’

SOP = 0, Size = 128, EOP = 0
{ Payload #2’.bytes[136:139],
Payload #3’.bytes[0:123] }

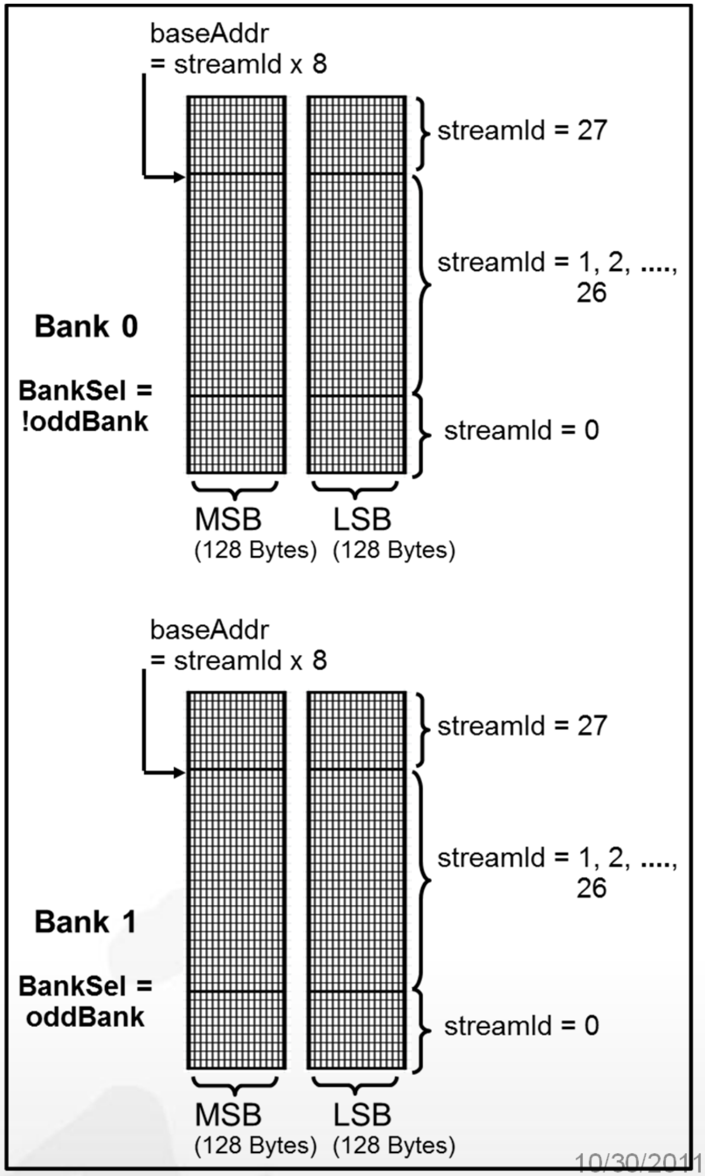
Design summary

Attribute	Value
Inputs	4,425
Outputs	3,488
Memory bits	948,636
Total flops	1,048,481

- Large data path
    - 256 Bytes in one cycle
- 56 concurrent ports
    - Interleave data for a given packet
    - Multiple partial packets can be outstanding for different ports
- RTL stores up to 16 cells
- QOS requirements depending on register programming
- Design has to deal with input errors

Memory architecture

- 3 FIFOs in the design
  - dataFifo: stores the reformatted cells
  - statusFifo: stores the attributes of cells
  - stateFifo: stores the read and write address pointers of the port
- Bank architecture
  - oddBank: determines the memory bank and toggles every cycle to avoid bank contention
  - streamId: determines the memory address in a bank
  - Each bank is divided into 2 single port RAMs: MSB & LSB

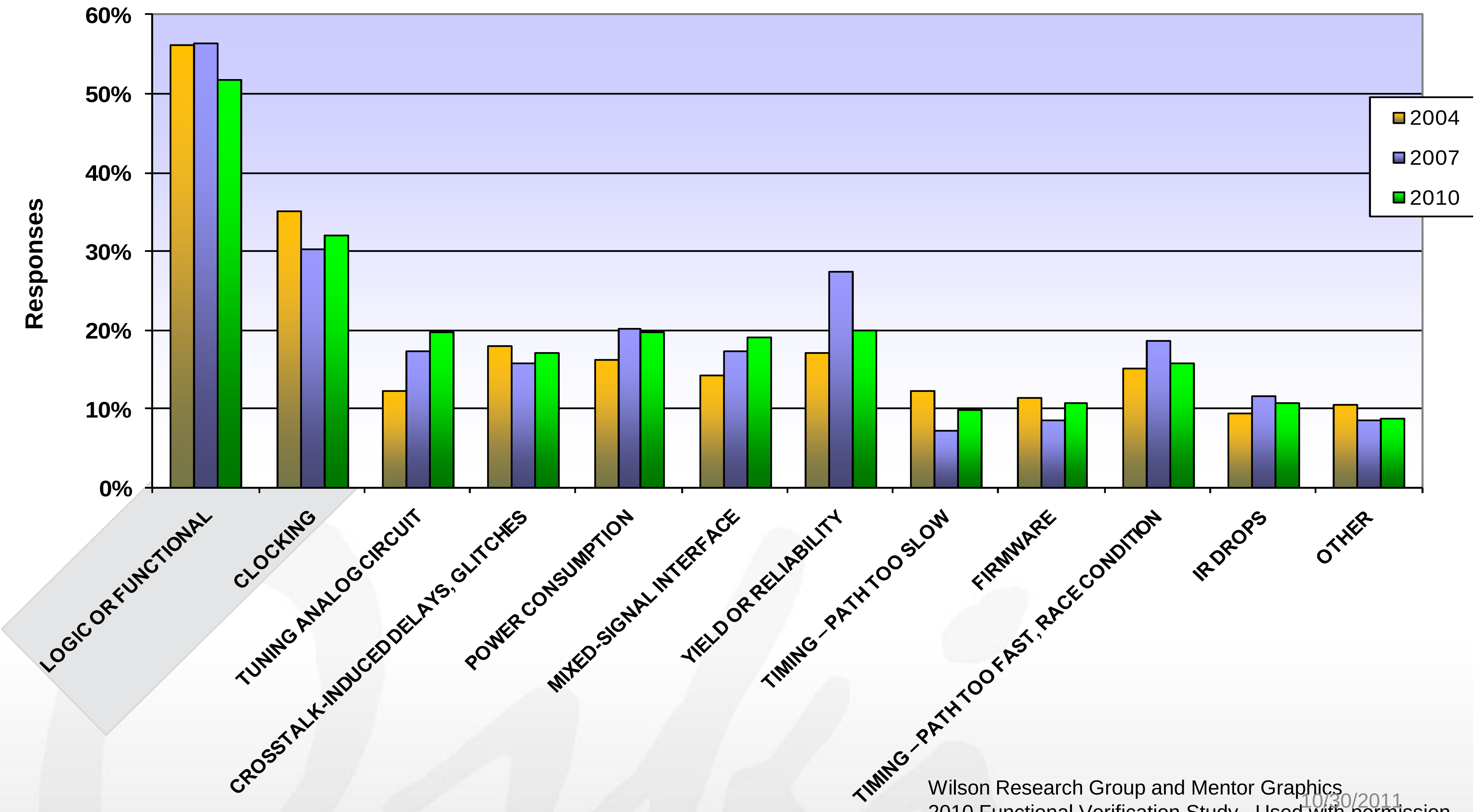


Formal in a simulation world

Oski TECHNOLOGY

Types of post-silicon flaws

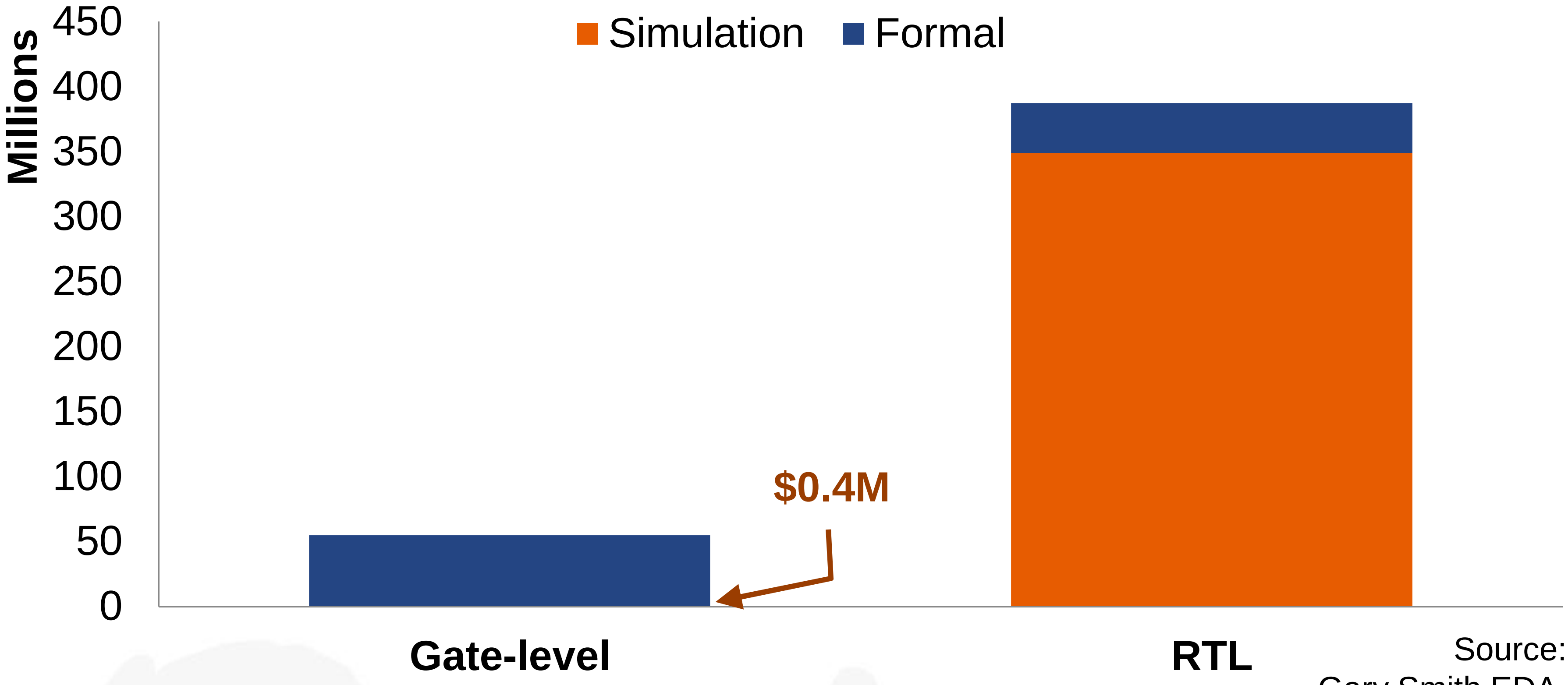
Verification is the still the largest problem



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

*Verification market size (2009)**

* excluding analog



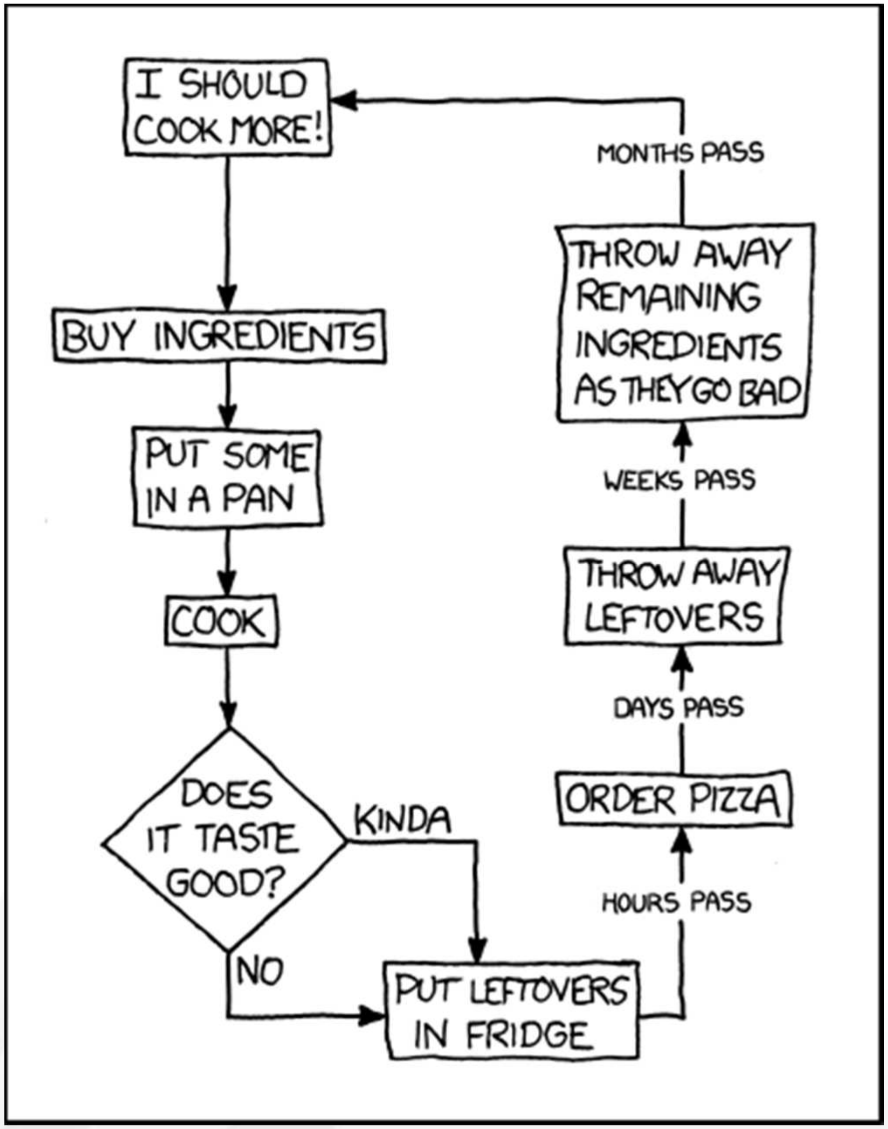
Source:
Gary Smith EDA,
October 2010

- **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

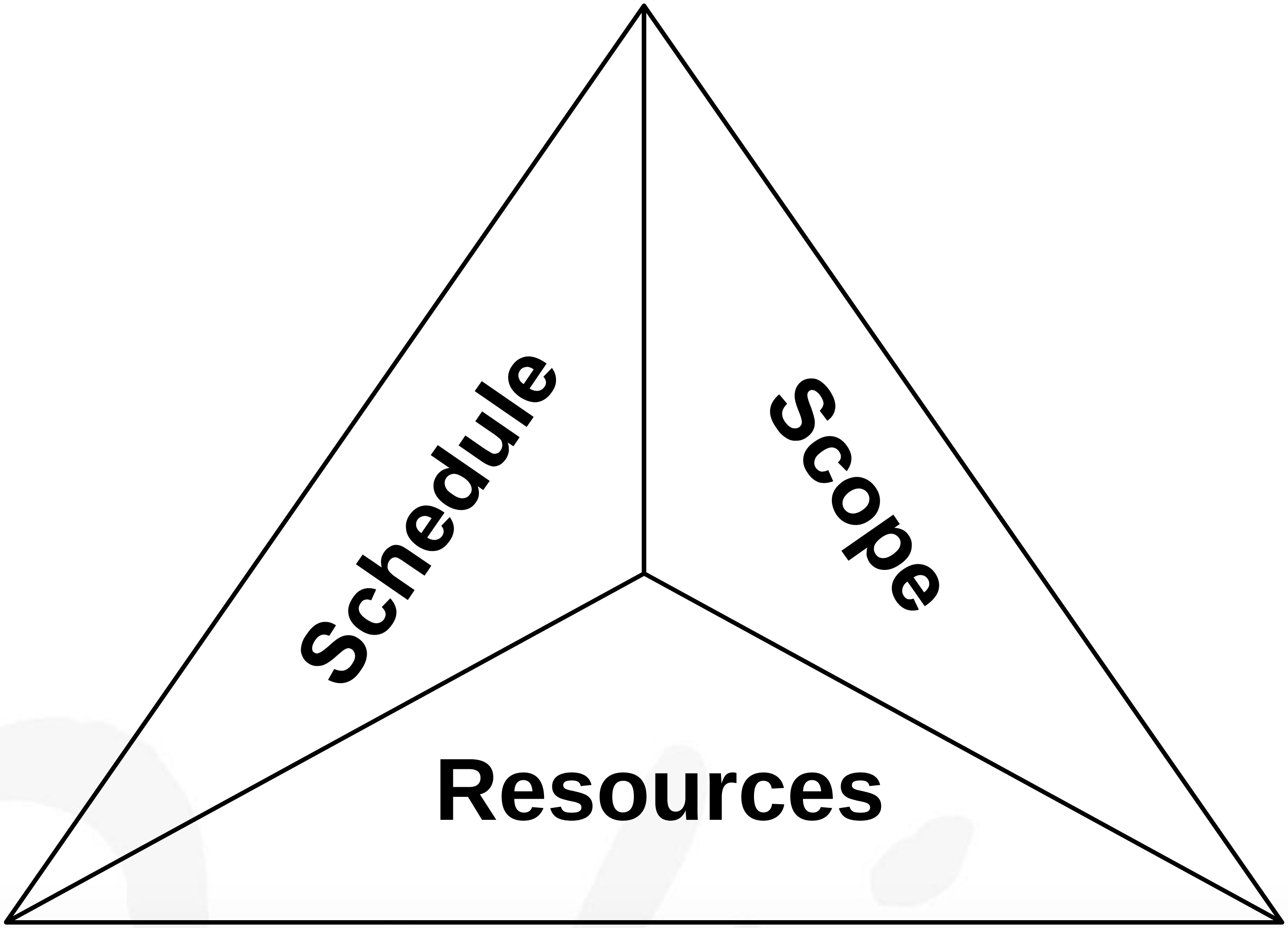
- Around for 20 years
- Expectations has been set high
  - Low effort for constraints
  - Tools run fast enough
- Expectations have been set low
  - Only verify local assertions
  - No End-to-End proofs
- Perception: low !/$

- Training and staffing
  - Few places to learn formal application
  - Single user should not do both formal and simulation



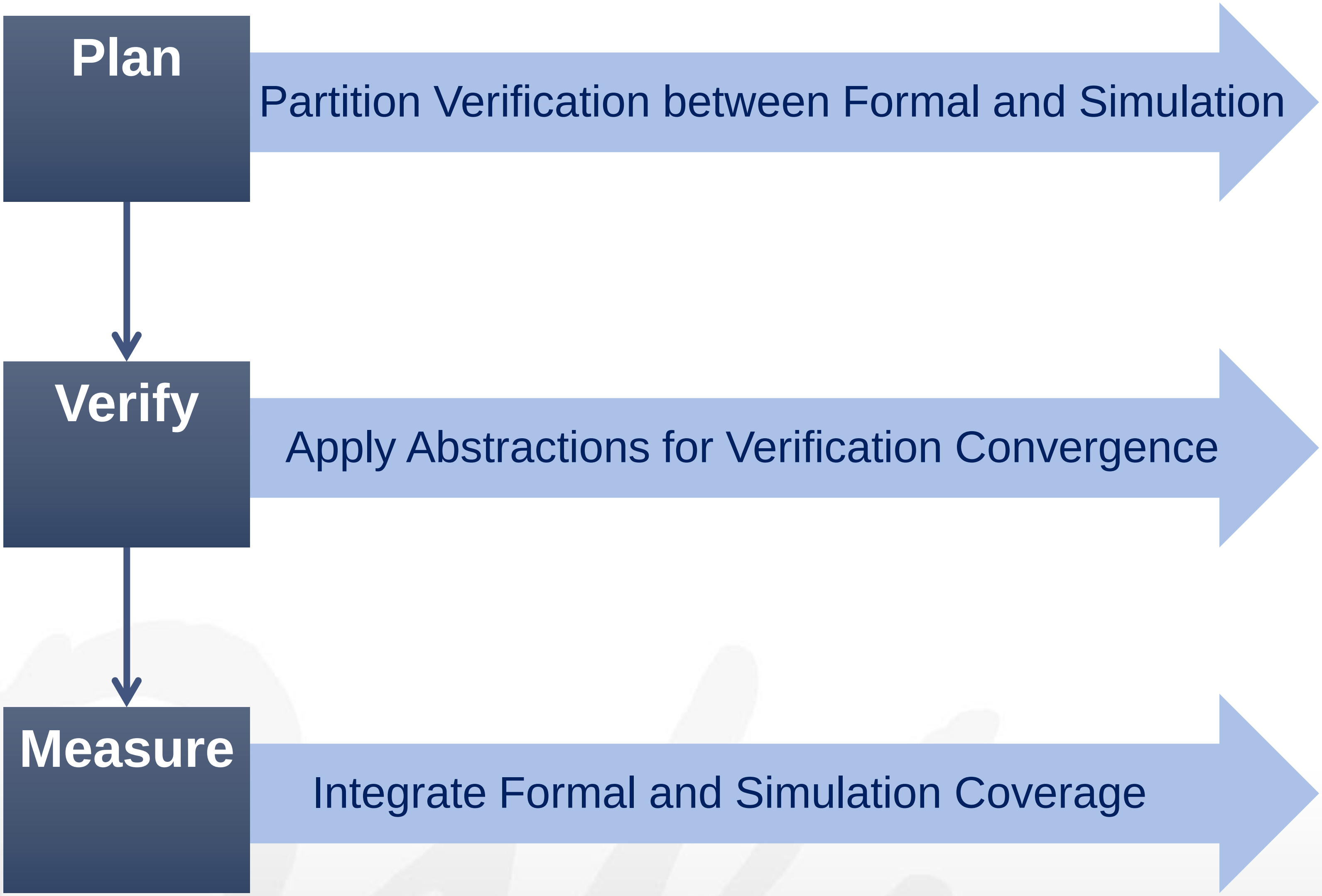
Source: xkcd.com

Tradeoffs in design flow



Source:
Stuart Oberman, NVIDIA

Achieving verification closure



Plan

Partition Verification between Formal and Simulation

Verify

Apply Abstractions for Verification Convergence

Measure

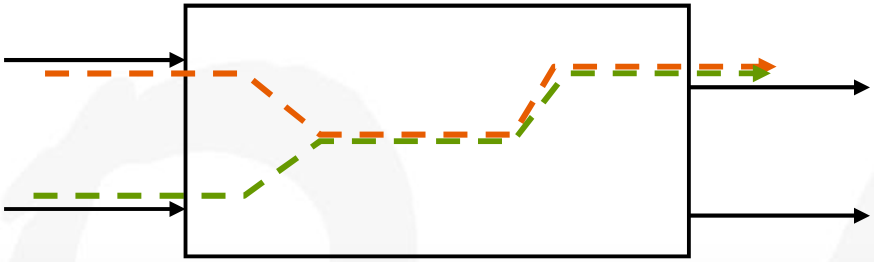
Integrate Formal and Simulation Coverage

Where to apply model checking

"Control", "Data Transport" designs

- Arbiters of many kinds
- Interrupt controller
- Power management unit
- Credit manager block
- Tag generator
- Schedulers

- Bus bridge
- Memory Controller
- DMA controller
- Host bus interface
- Standard interfaces (PCI Express, USB)
- Clock disable unit



Multiple, concurrent streams

Hard to completely verify using simulation

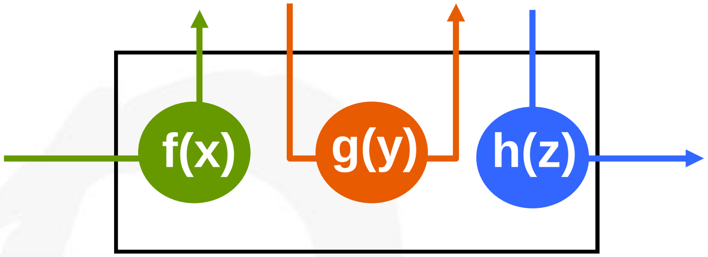
"10 bugs per 1000 gates"

-Ted Scardamalia, IBM

Where not to apply model checking

"Data transform" designs

- Floating point unit
- Graphics shading unit
- Inverse quantization
- Convolution unit in a DSP chip
- MPEG decoder
- Classification search algorithm
- Instruction decode



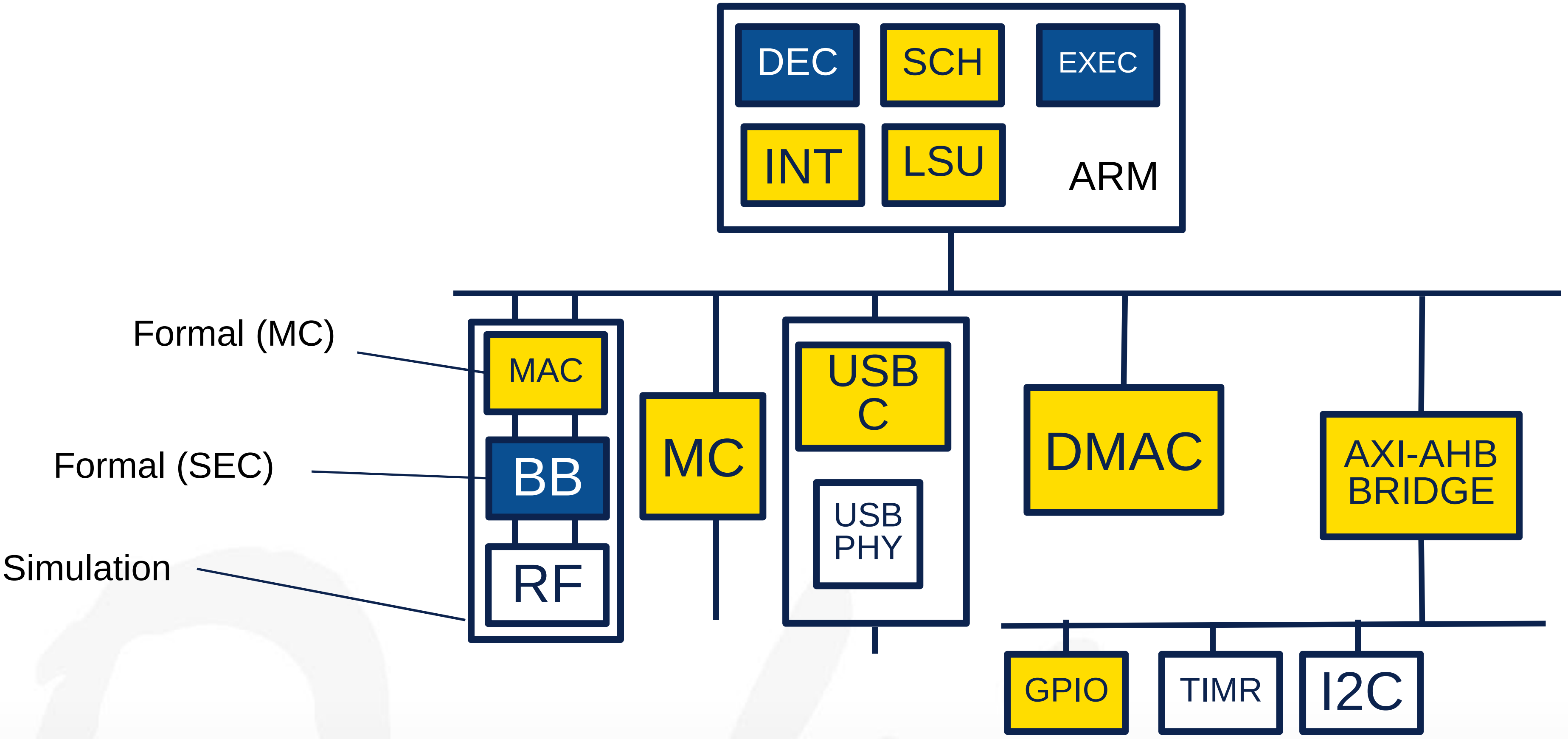
Single, sequential functional streams

"2 bugs per 1000 gates"

-Ted Scardamalia, IBM

Formal (MC, SEC) and simulation strengths*

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



How perfect does formal have to be?

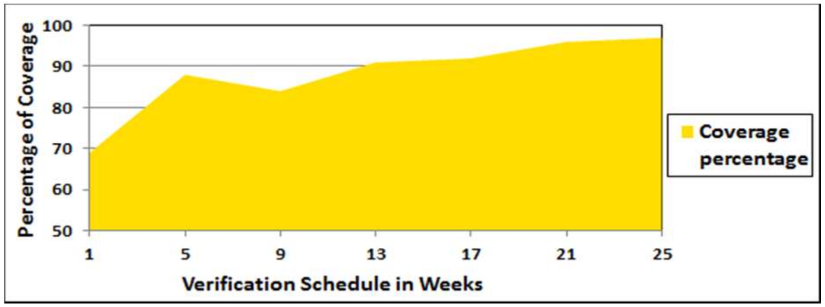
Graphic: MacGregor Marketing

- Not all bugs need to found/fixed
- Formal does not need to find the last bug
- Usually bounded proofs are good enough

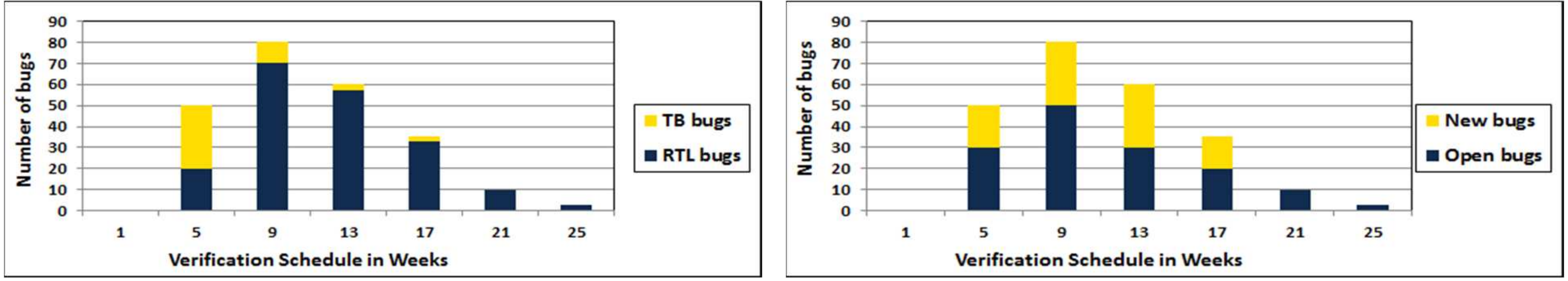
  (if bound is good enough!)
- Formal has to be more cost-effective than the alternative

Verification manager’s dashboard

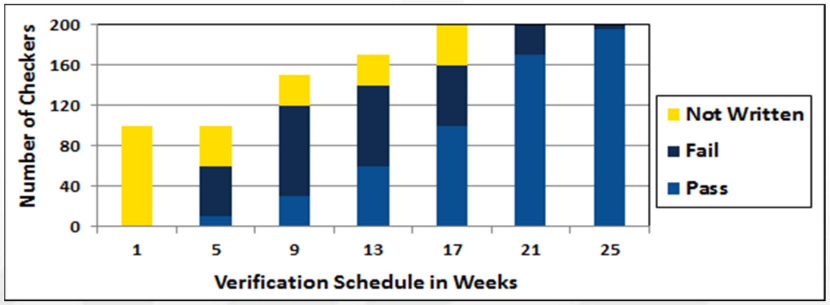
Coverage tracking



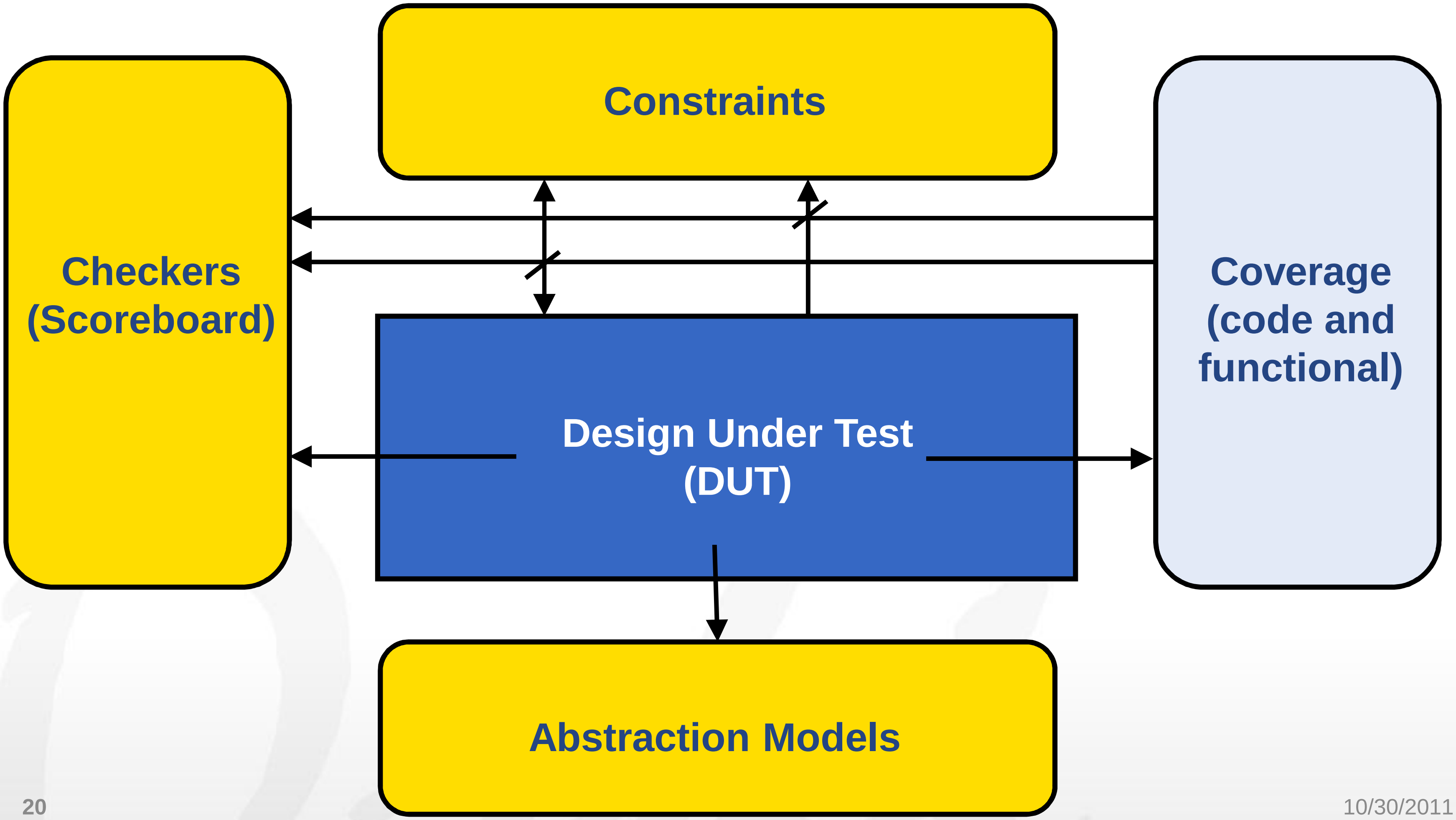
Bug tracking



Runtime status



A formal testbench



Three Cs of Formal

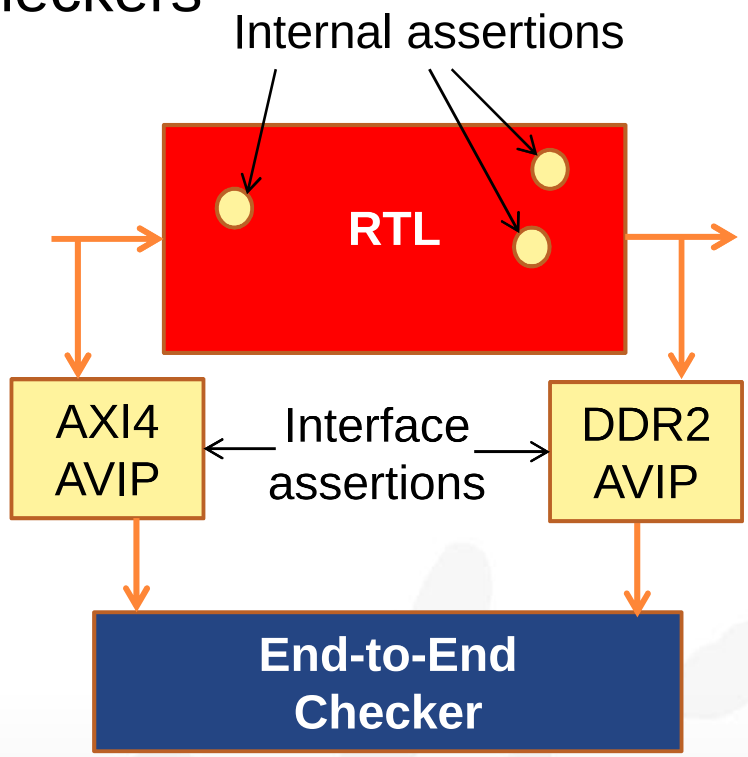
- Checkers
- Constraints
- Complexity
  - (using Abstraction Models)

- … and Coverage (to measure completeness of formal)

End-to-End formal

Different kinds of Checkers

- Internal assertions
- Interface assertions
- End-to-end checkers



Internal assertions

- Relate a few design signals
- Can be written completely in SVA
- Usually embedded in RTL, and written by designers
- e.g. state machine “sm[7:0]” is one-hot encoded

- Useful for bug hunting
  - Not for finding all/most bugs, or as replacement for simulation effort
- Complexity
  - Can be small, if proof core is small

Interface assertions

- Relate input and output signals on a given interface
- May require a small amount of modeling code
- E.g. valid-ack protocol

```
(validOut && (!ackIn)) |-> ##1 (dataOut == $past(dataOut));
```

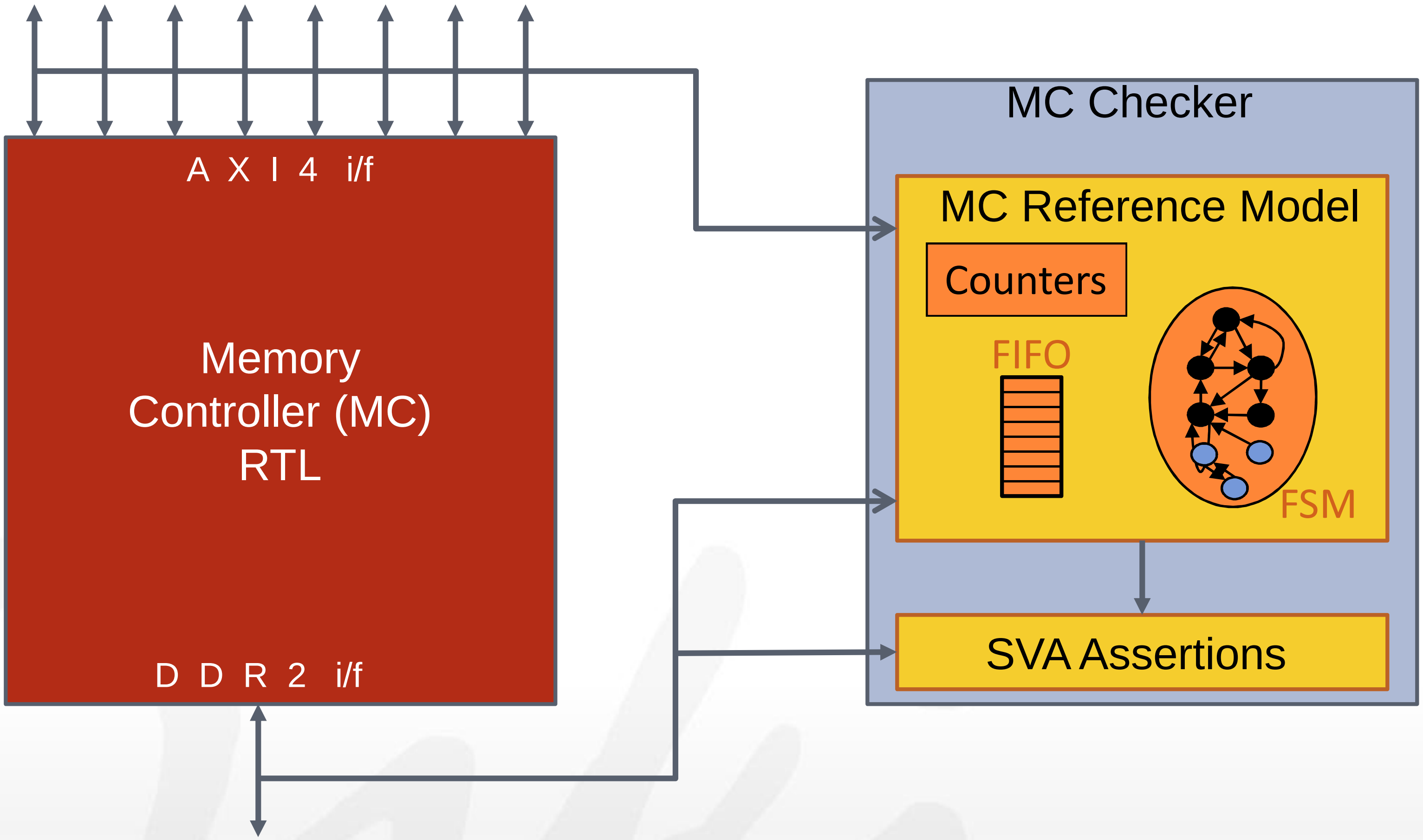
- Protocol interfaces kits, e.g. AMBA AHB/AXI3, DDR/DDR2
- Useful for bug hunting
  - Not for finding all/most bugs, or as replacement for simulation effort
- Complexity
  - Often harder to prove than internal assertions

End-to-End Checkers

- Require a reference model to implement Checker
- Can replace simulation effort for that design, mostly or completely
- Usually needs a plan to avoid complexity barrier
  - Often abstractions are necessary to overcome complexity
    - For each search step
    - Reduce the diameter of search
- Example of end-to-end checkers
  - Number of bytes coming out equals number of bytes going in
  - Output cell sizes and SOP/EOP corresponds to input data
  - Output data values match predicted values

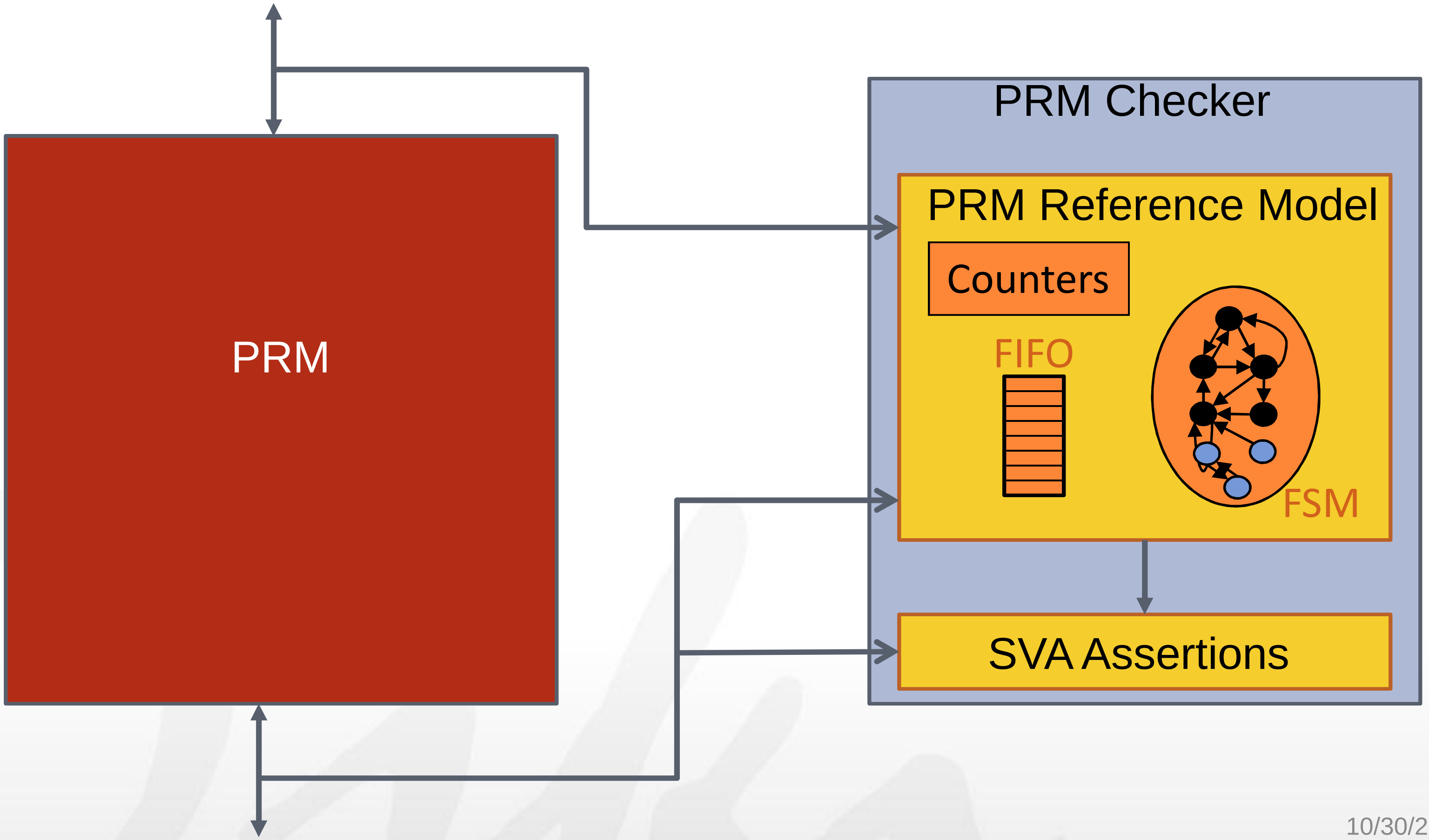
End-to-End Checkers

- 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



PRM Checkers

- Model reformatting function
- Model sizes and data of cells in flight
- Predict output cell sizes and data value



PRM Checkers

- Interface checkers
  - For a port, between 2 cells of SOP as 1 there should be cell with EOP as 1
  - For a port, between 2 cells of EOP as 1 there should be cell with SOP as 1
  - For a port, the next valid cell after an EOP as 1 must have SOP as 1
  - Output cell should have Size > 0
  - Output cell with EOP as 0 should have Size =128

- End-to-end checkers
  - For a port, the valid output (validOut) can be 1 only if there are outstanding cells in flight that have not been sent out
  - For a port, payload of a cell at the output should correspond to payload of expected cell in the reference model

Abstractions to overcome complexity

Source of Complexity

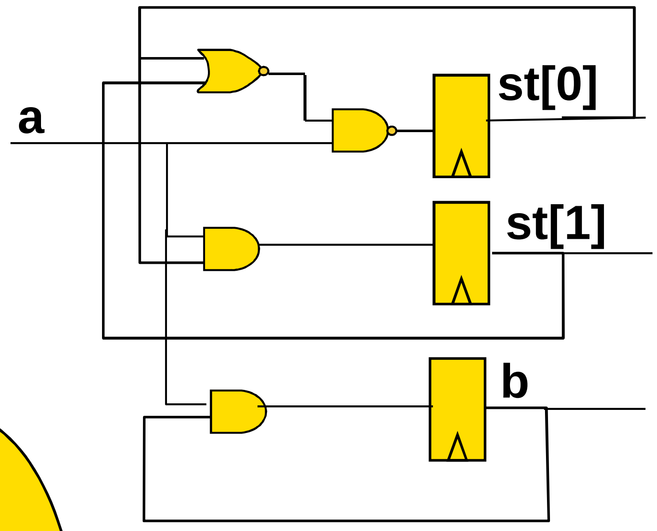
RTL

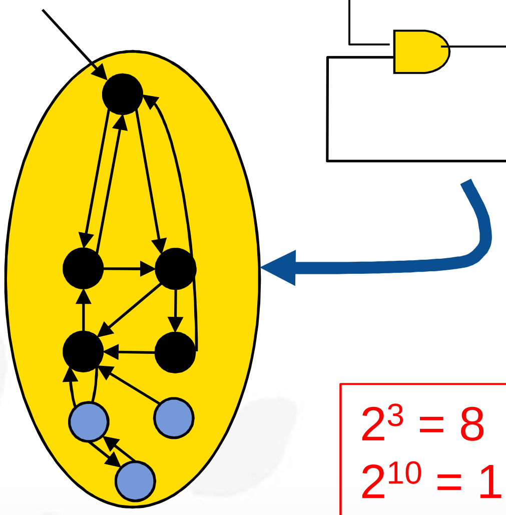
```
input a;
reg b;
reg [1:0] st;

always @(posedge clk or negedge rst)
  if (~rst) st <= 2’b00;
  else case( st )
      2’b00:  if (~a) st <= 2’b01;
      2’b01:  st <= 2’b10;
      2’b10:  if (a) st <= 2’b00;
      endcase

always @(posedge clk or negedge rst)
  if (~rst) b <= 1’b0;
  else if (~a | b) b <= 1’b0;
  else b <= 1’b1;
```

Internal netlist





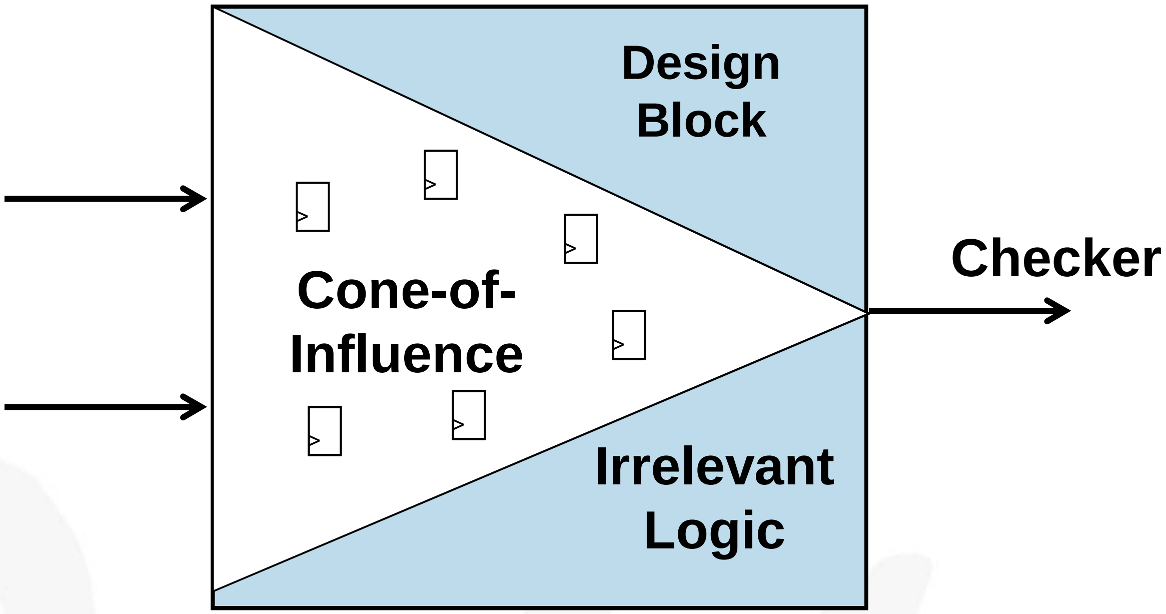
State Transition Graph (STG)

$2^3 = 8$
$2^{10} = 1{,}024$
$2^{20} = 1{,}048{,}576$
$2^{30} = 1{,}073{,}741{,}824$

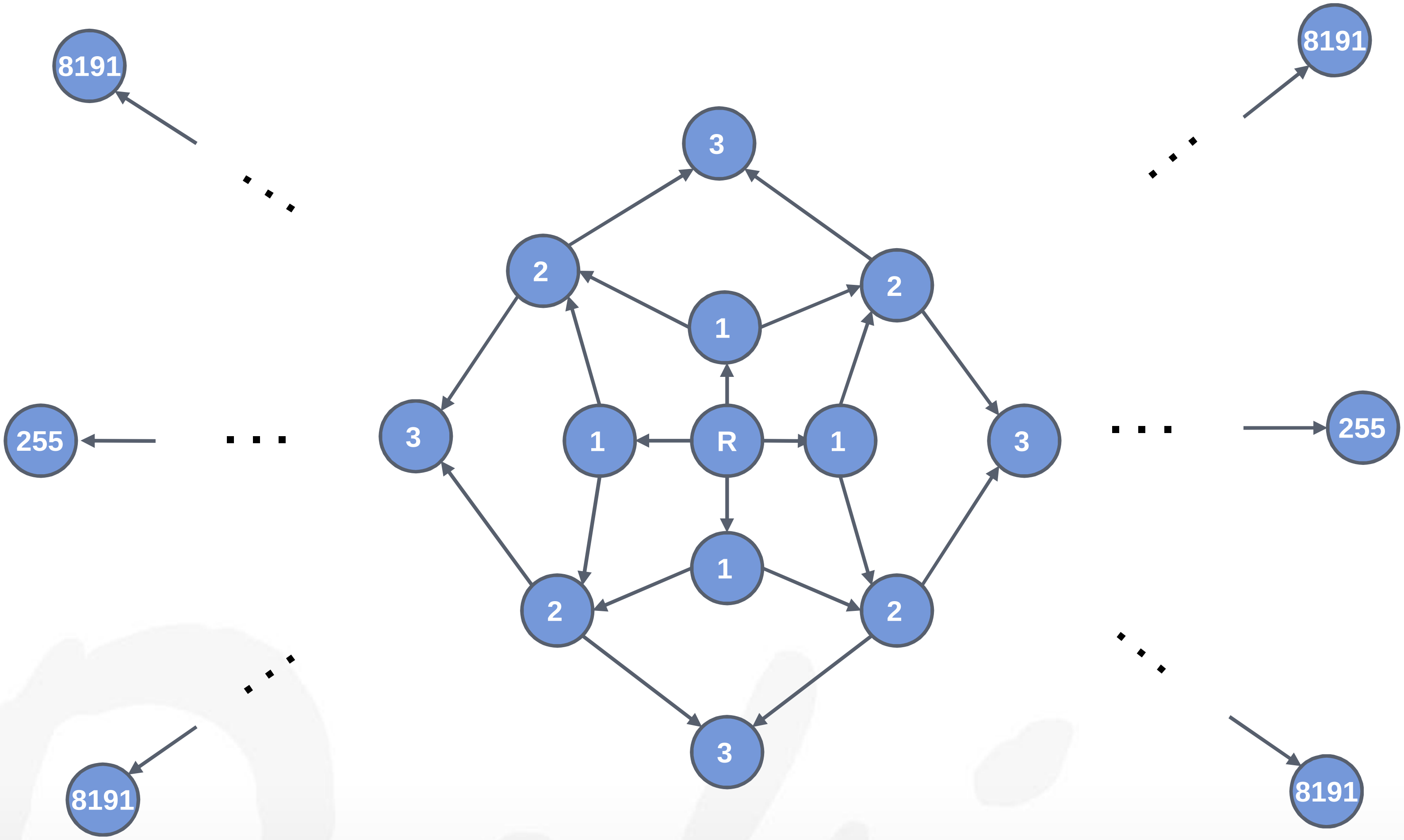
Checker: (st == 2’b01) => ~b

Complexity – function of Cone-of-Influence

- One coarse measure of Complexity
  - number of flops/memory bits in the Cone-of-Influence of the Checker



State space complexity

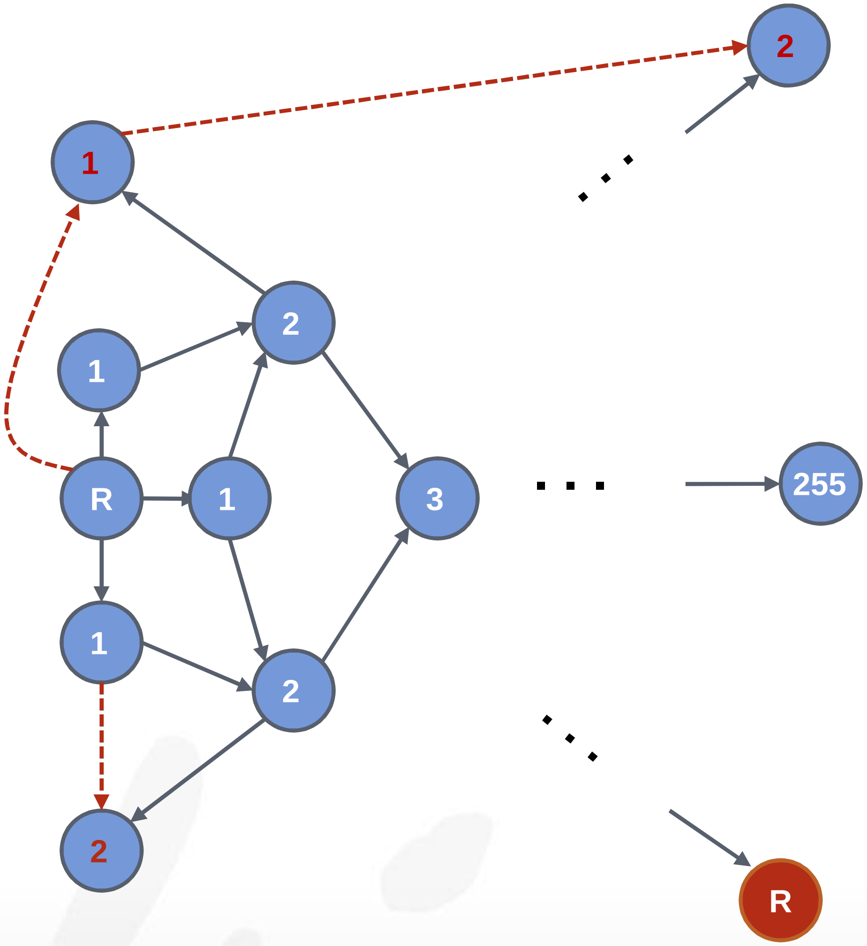


Abstractions (to manage complexity)

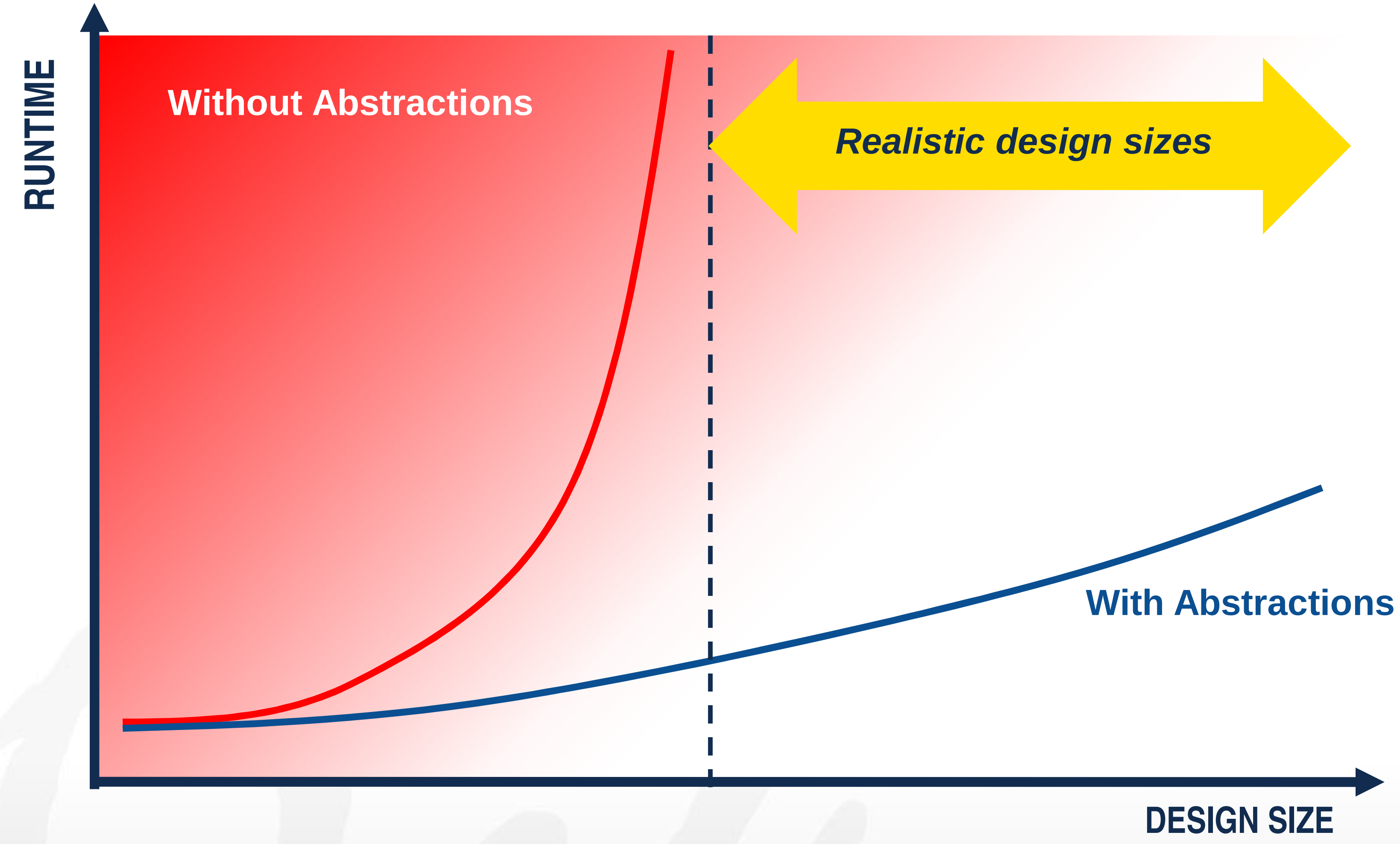
- An “Abstraction” of a design models a superset of the design behavior
- Useful to overcome complexity barriers
  - Smaller Cone-of-Influence
  - Shallower search space
  - Ability to skip long initialization sequences
- Cannot give a false positive
- Can give a false negative (Fail), but…
  - you get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

Complexity (and Abstractions)

- Effect of abstractions:
  - Reduce per-cycle search time
  - Reduces state space
  - Adds state transitions
  - Adds Reset states

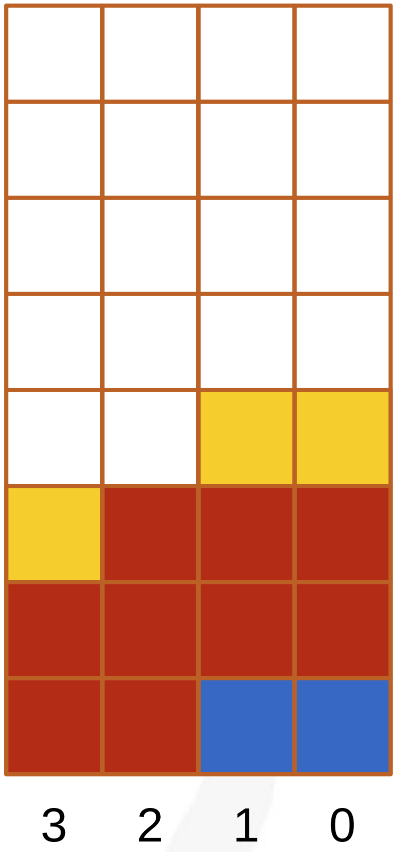


Overcoming complexity with Abstractions



CellReformatter memory abstraction

3	2	4	3	1	1	Size
1	0	0	1	0	1	SOP
1	1	0	0	1	0	EOP



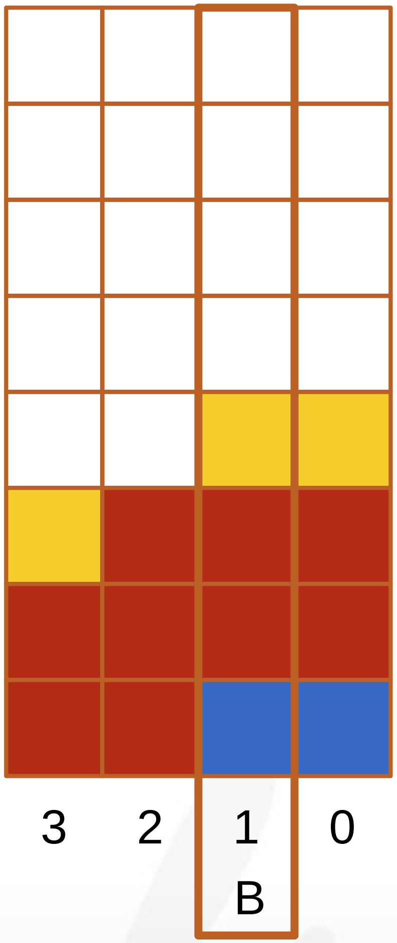
3	1	4	4	2	Size
1	0	0	1	1	SOP
1	1	0	0	1	EOP

Checker:

(rtl.validOut |->
(rtl.cellOut = ref.cellOut))

CellReformatter memory abstraction

3	2	4	3	1	1	Size
1	0	0	1	0	1	SOP
1	1	0	0	1	0	EOP



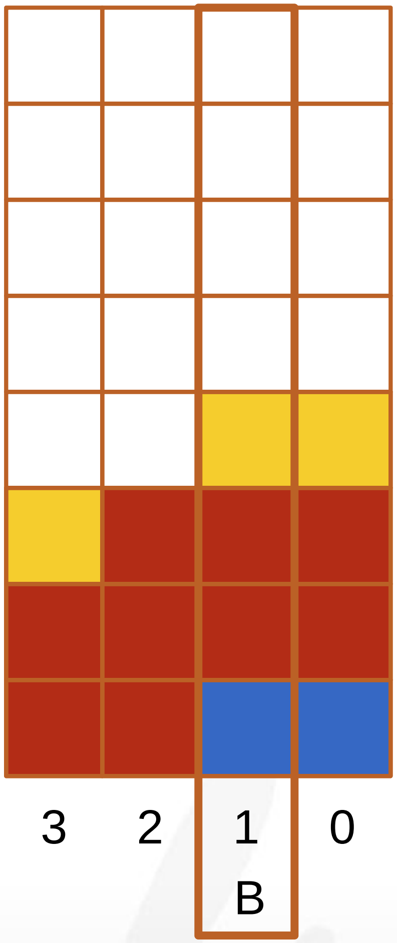
3	1	4	4	2	Size
1	0	0	1	1	SOP
1	1	0	0	1	EOP
					C

Abstraction:
Watch index B (variable)

CellReformatter memory abstraction

3	2	4	3	1	1	Size
1	0	0	1	0	1	SOP
1	1	0	0	1	0	EOP

Abstraction:
Watch index B (variable)



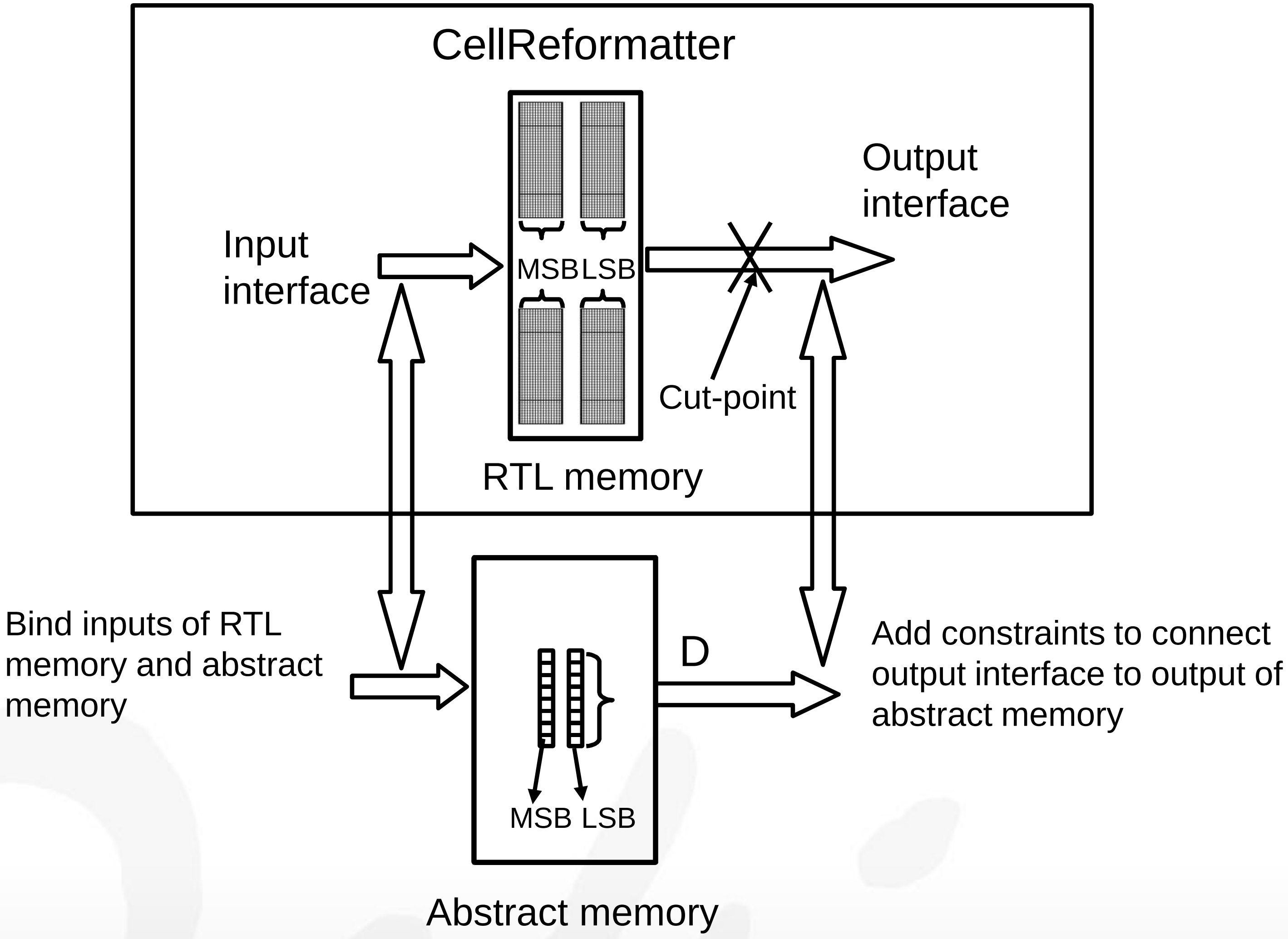
3	1	4	4	2	Size
1	0	0	1	1	SOP
1	1	0	0	1	EOP
2	–	3	3	1	C

```
initial(C) <= B
if (rtl.validOut)
  next(C) <= (C – rtl.cellOutAttri.Size) % 4;
```

Checker:

```
(rtl.validOut &&
 (rtl.cellOutAttri.Size > C)) |->
 (rtl.cellOut[C] = ref.cellOut[C]))
```

Deploying memory abstraction



Bind inputs of RTL memory and abstract memory

Add constraints to connect output interface to output of abstract memory

Abstraction for RTL memory

RTL memory

```
module rtl_memory (
  input clk, input we,
  input [3:0] addr,
  input [127:0] wd,
  output [127:0] rd);

  reg [127:0] mem [15:0];

  always @(posedge clk)
    if (we) mem[addr] <= wd;

  wire rd = mem[addr];

endmodule
```

Abstract memory

```
module abs_memory (
  input clk, input we,
  input [3:0] addr,
  input [127:0] wd,
  input [6:0] B);

  reg mem [15:0];

  always @(posedge clk)
    if (we) mem[addr] <= wd[B];

  wire rd = mem[addr];

endmodule
```

- Bind abs_memory to same inputs as rtl_memory
- Bind input B to variable B
- Blackbox rtl_memory
- Assume (rtl_memory.rd[B] == abs_memory.rd)

Abstraction for RTL memory

- Without the abstraction:
  - Entire memory (128 * 16 = 2,048 bits) is in the COI for the checker:
    - ***Checker: (rtl.validOut |-> (rtl.cellOut[C] = ref.cellOut[C]))***
  - Run-time: 0 cycles in 20min
- With the memory abstraction:
  - Only one bit per line; total of 16 bits
  - Run-time: 30 cycles in 20min
- Can implement a more aggressive abstraction:
  - Check only one symbolic bit per run
    - Use random input that becomes one for exactly one bit
    - Modeling C is a bit more complex

Other abstractions for PRM

- Port number is a fixed variable
  - $0 <= P <= 55$
- Byte number is a fixed variable
  - $0 <= I <= 7$
- Wolper’s data independence abstraction is used to verify data corruption
  - Replace input sequence by 0^*110^ω
  - Verify that the output sequence equals 0^*110^ω

Abstractions

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

Coverage to measure completeness

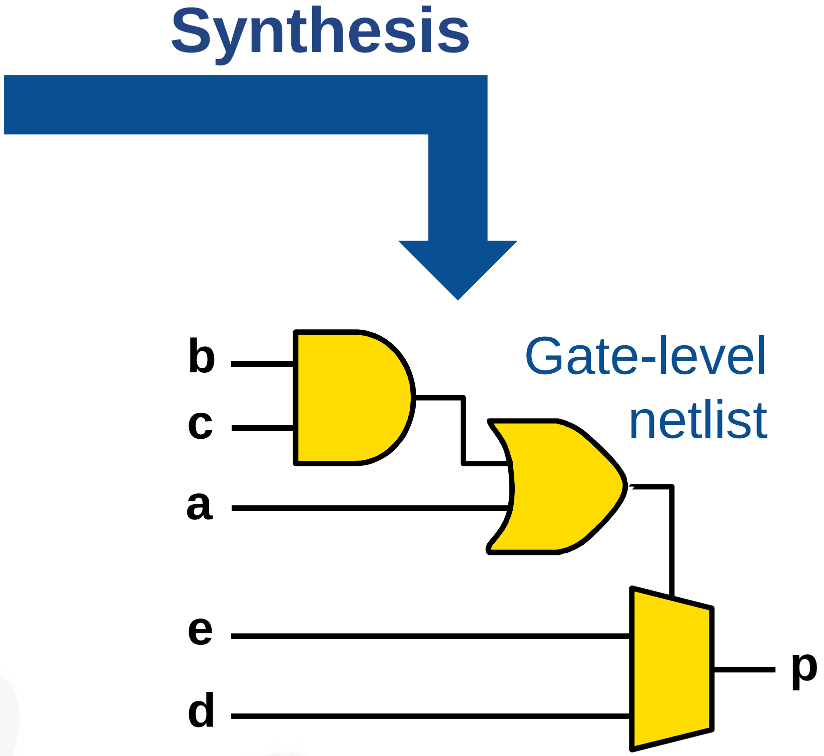
Coverage on RTL designs

RTL (Verilog)

```
reg p;
always @(*) begin
  if (a || (b && c))
    p = d;
  else
    p = e;
end
```

Equivalent RTL

```
reg w, p;
always @(*) begin
  w = a || (b && c);
end
always @ (*) begin
  p = (w && d) || ((!w) && e);
end
```


Synthesis

Gate-level netlist

Input Coverage: line/expression coverage

```
reg p;
always @(*) begin
  if (a || (b && c))
    p = d;
  else
    p = e;
end
```

Line coverage

a	b	c	p	
0	0	0	e	target #1
0	0	1	e	
0	1	0	e	
0	1	1	d	target #2
1	0	0	d	
1	0	1	d	
1	1	0	d	
1	1	1	d	

Expression coverage

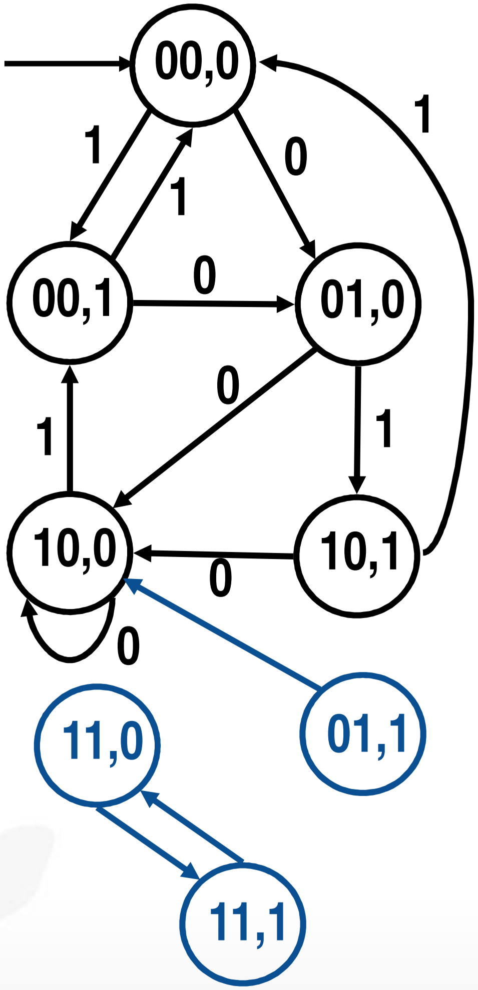
a	b	c	p	
0	0	0	e	
0	0	1	e	#1
0	1	0	e	#2
0	1	1	d	#3
1	0	0	d	#4
1	0	1	d	
1	1	0	d	
1	1	1	d	

Code coverage vs STG coverage

```
input a;
reg b;
reg [1:0] st;

always @(posedge clk or negedge rst)
  if (~rst) st <= 2'b00;
  else case( st )
       2'b00:  if (~a) st <= 2'b01;
       2'b01:  st <= 2'b10;
       2'b10:  if (a) st <= 2'b00;
       endcase

always @(posedge clk or negedge rst)
  if (~rst) b <= 1'b0;
  else if (~a | b) b <= 1'b0;
  else b <= 1'b1;
```

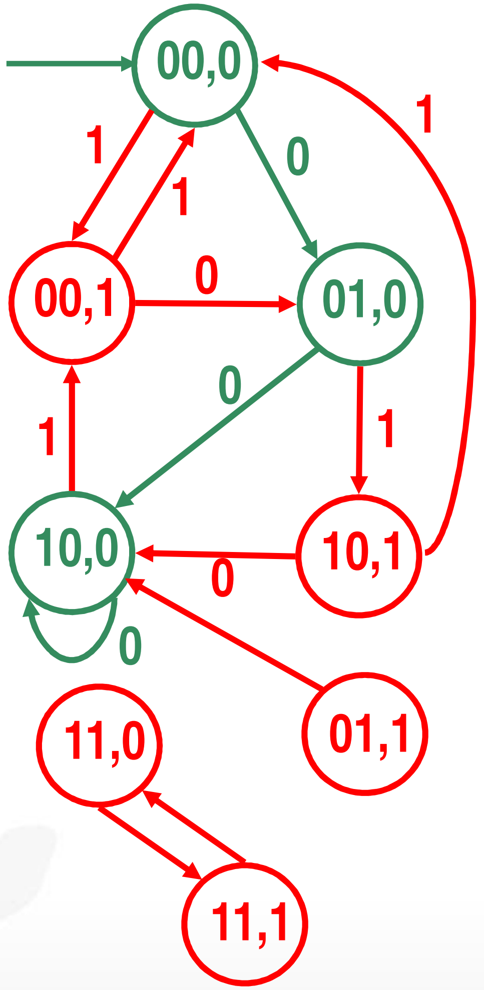


Simulation coverage (a = 0)

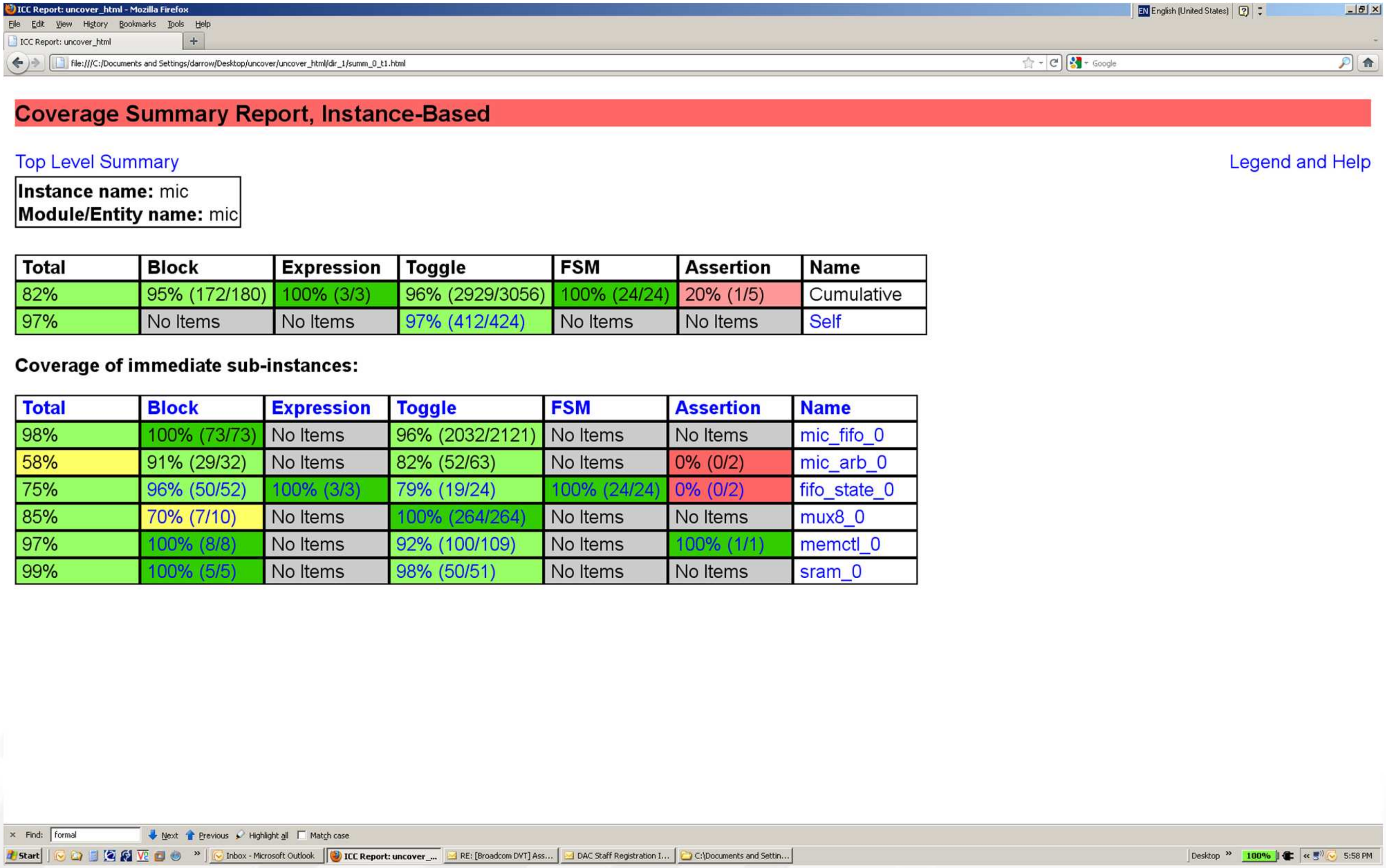
```
input a;
reg b;
reg [1:0] st;

always @(posedge clk or negedge rst)
  if (~rst) st <= 2’b00;
  else case( st )
       2’b00:  if (~a) st <= 2’b01;
       2’b01:  st <= 2’b10;
       2’b10:  if (a) st <= 2’b00;
       endcase

always @(posedge clk or negedge rst)
  if (~rst) b <= 1’b0;
  else if (~a | b) b <= 1’b0;
  else b <= 1’b1;
```



Coverage reporting



ICC Report: uncover_html - Mozilla Firefox
English (United States)
File Edit View History Bookmarks Tools Help
ICC Report: uncover_html
file:///C:/Documents and Settings/darrow/Desktop/uncover/uncover_html/dir_1/summ_0_t1.html
Google
Find: formal
Next Previous Highlight all Match case
Start
Inbox - Microsoft Outlook
ICC Report: uncover_...
RE: [Broadcom DVT] Ass...
DAC Staff Registration I...
C:\Documents and Settin...
Desktop
100%
5:58 PM

Coverage Summary Report, Instance-Based

Top Level Summary

Legend and Help

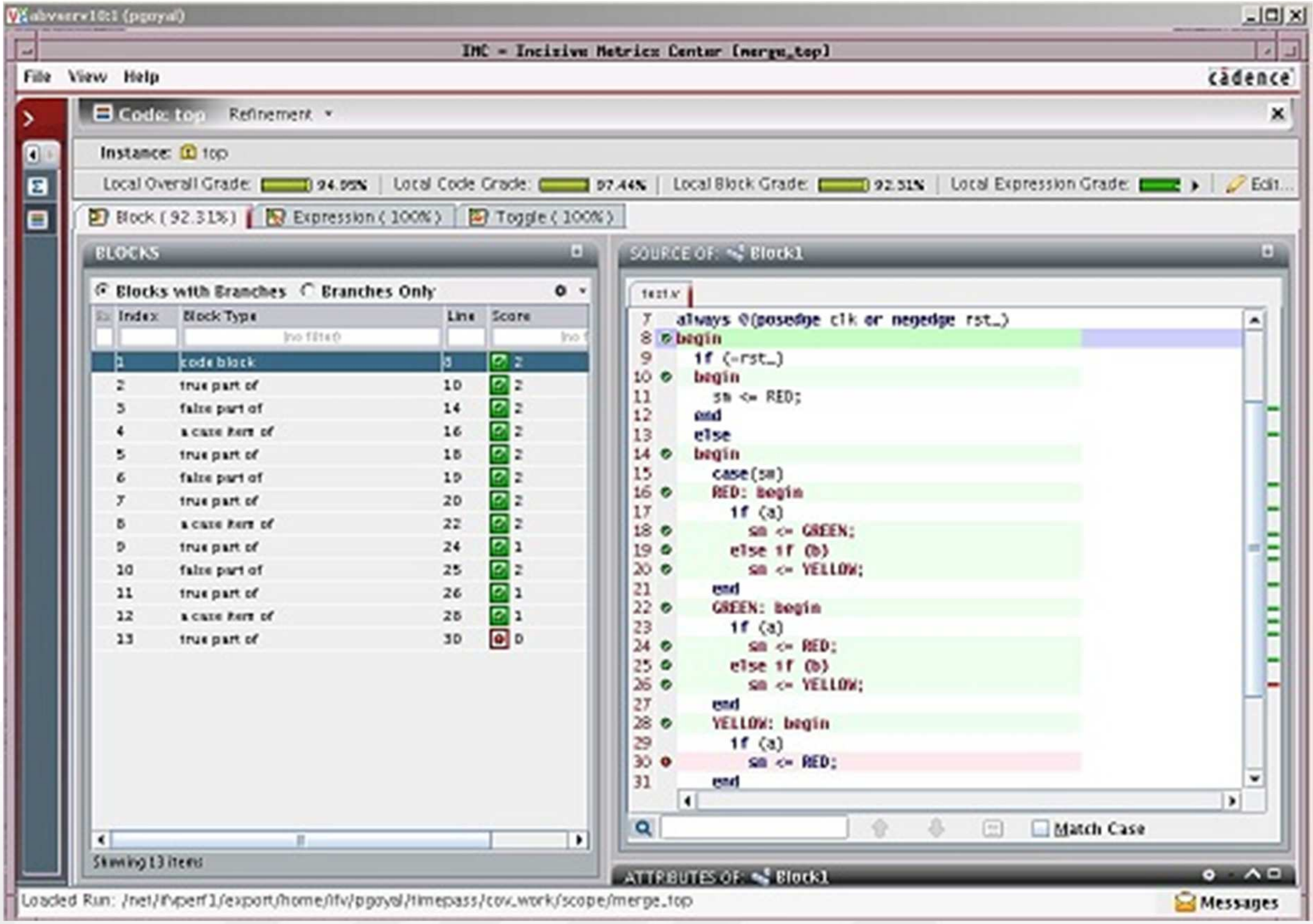
Instance name: mic
Module/Entity name: mic

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

Coverage reporting



● Blocks with Branches ○ Branches Only

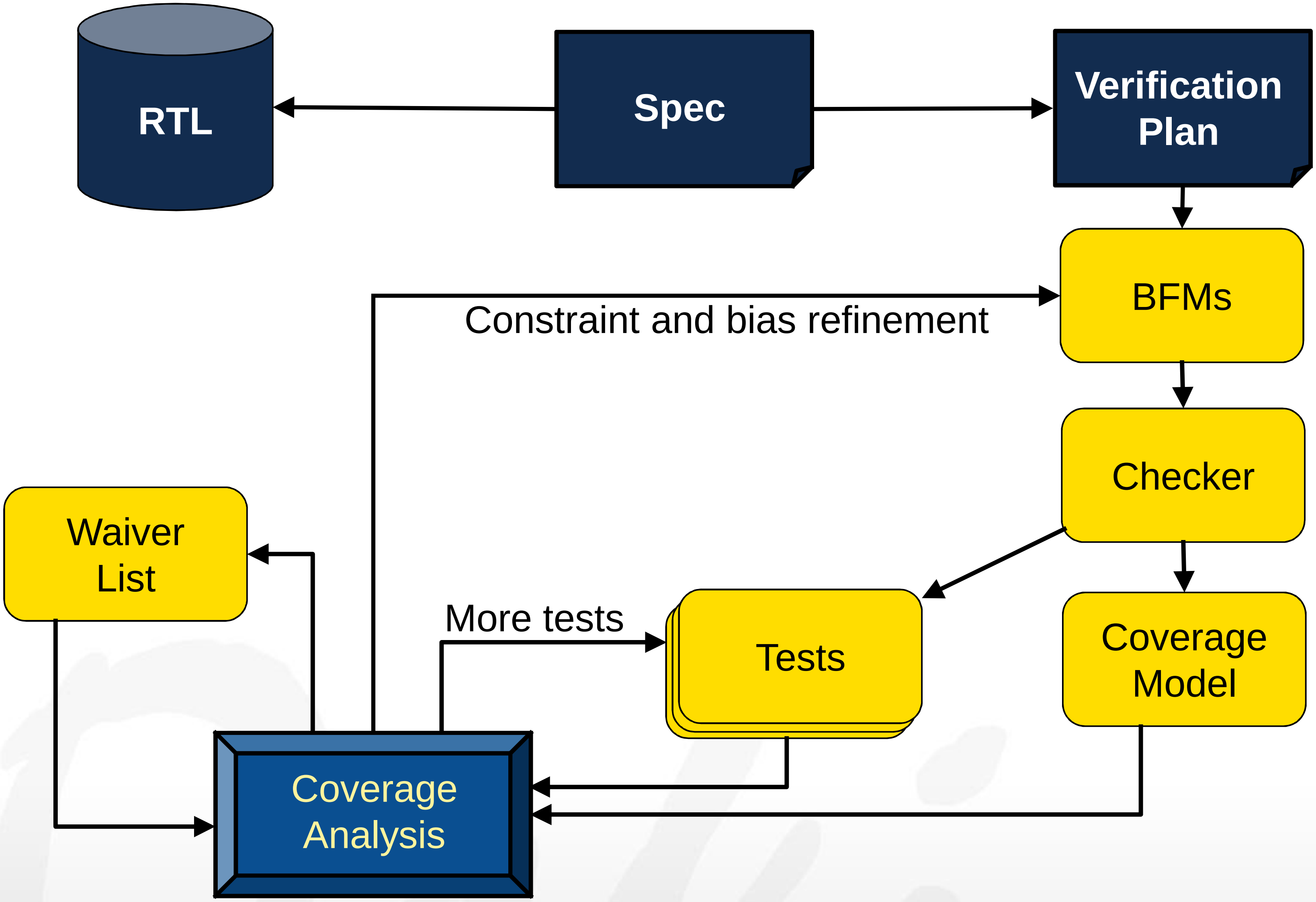
Index	Block Type	Line	Score
1	code block	8	🟩 2
2	true part of	10	🟩 2
3	false part of	14	🟩 2
4	a case item of	16	🟩 2
5	true part of	18	🟩 2
6	false part of	19	🟩 2
7	true part of	20	🟩 2
8	a case item of	22	🟩 2
9	true part of	24	🟩 1
10	false part of	25	🟩 2
11	true part of	26	🟩 1
12	a case item of	28	🟩 1
13	true part of	30	🟥 0

Showing 13 items

```
7      always @(posedge clk or negedge rst_)
8   ●  begin
9        if (~rst_)
10  ●    begin
11         sm <= RED;
12       end
13       else
14  ●    begin
15         case(sm)
16  ●      RED: begin
17           if (a)
18  ●          sm <= GREEN;
19  ●        else if (b)
20  ●          sm <= YELLOW;
21         end
22  ●      GREEN: begin
23           if (a)
24  ●          sm <= RED;
25  ●        else if (b)
26  ●          sm <= YELLOW;
27         end
28  ●      YELLOW: begin
29           if (a)
30  ●          sm <= RED;
31         end
```

Loaded Run: /net/ifvperf1/export/home/ifv/pgoyal/timepass/cov_work/scope/merge_top

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 (input vs observable)

- Two questions determine completeness:
  - “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 is a practical formal technique
- 100% coverage does not mean design is bug-free
- But, coverage is useful to
  - manage verification progress
  - highlight missed verification holes

Is my formal complete?

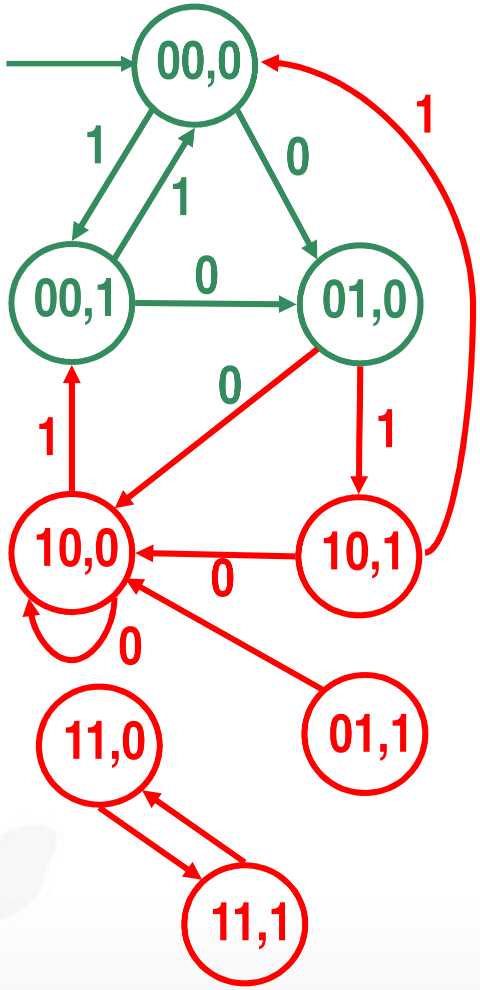
- Are my Checkers complete?
- Are my Constraints weak enough?
- Is my Complexity strategy complete?

Formal coverage (depth = 1)

```
input a;
reg b;
reg [1:0] st;

always @(posedge clk or negedge rst)
  if (~rst) st <= 2’b00;
  else case( st )
      2’b00:  if (~a) st <= 2’b01;
      2’b01:  st <= 2’b10;
      2’b10:  if (a) st <= 2’b00;
      endcase

always @(posedge clk or negedge rst)
  if (~rst) b <= 1’b0;
  else if (~a | b) b <= 1’b0;
  else b <= 1’b1;
```



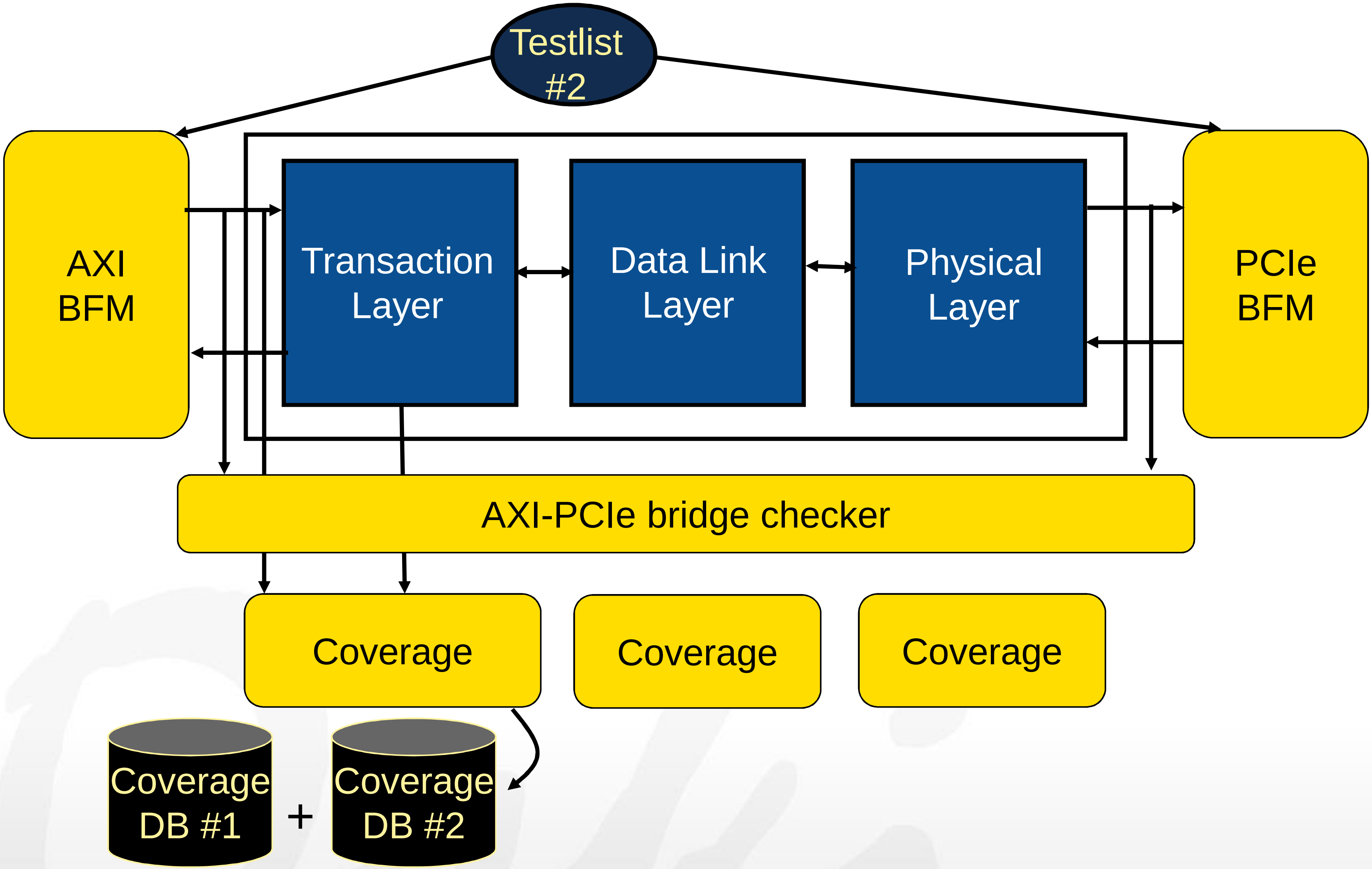
Formal (input) coverage

- 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

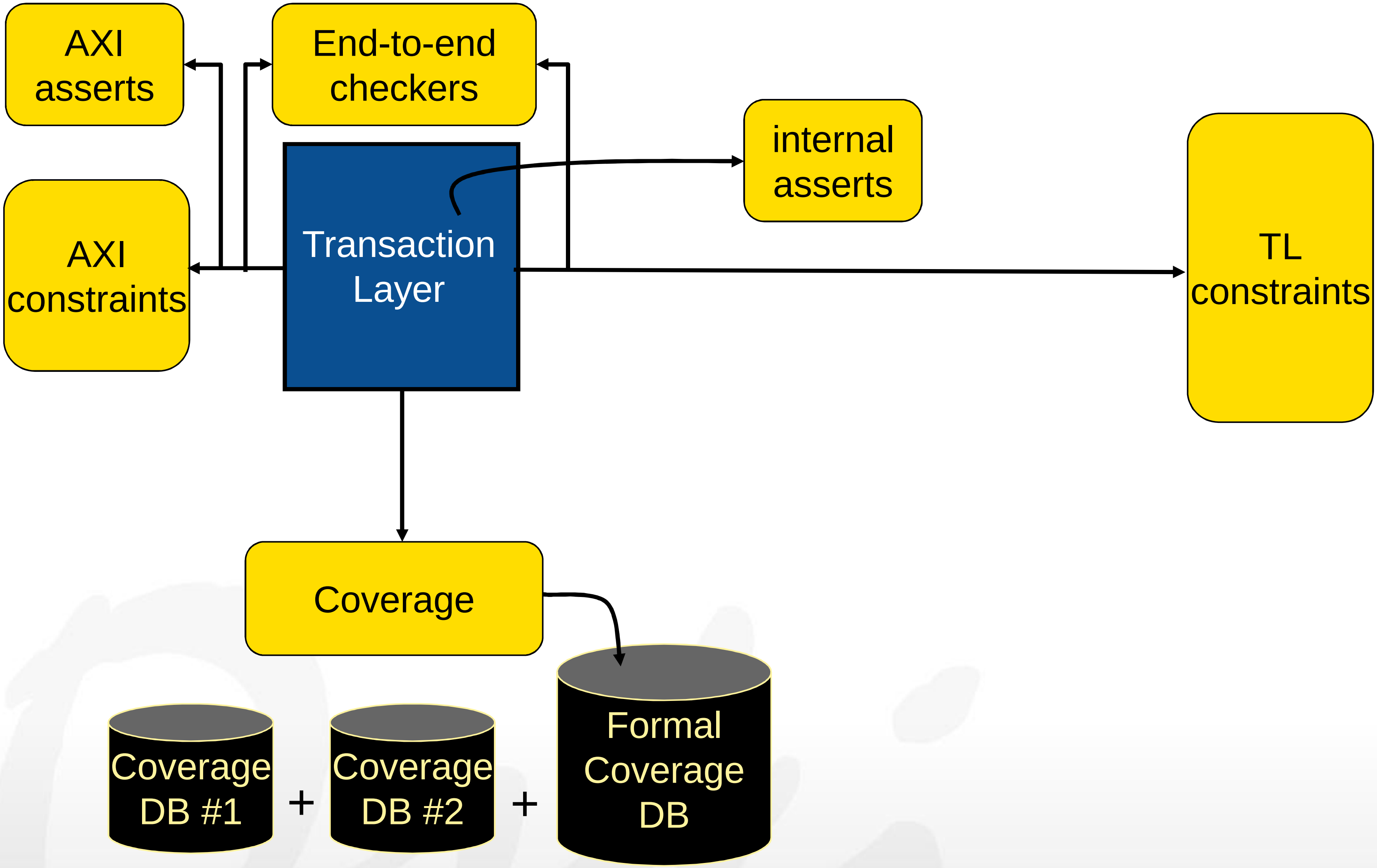
- Complexity: All checkers are verified up to proof depth N
  - Any target, not reachable in N clocks, is not covered

- Checkers: does not verify completeness of Checkers
  - No different than simulation!

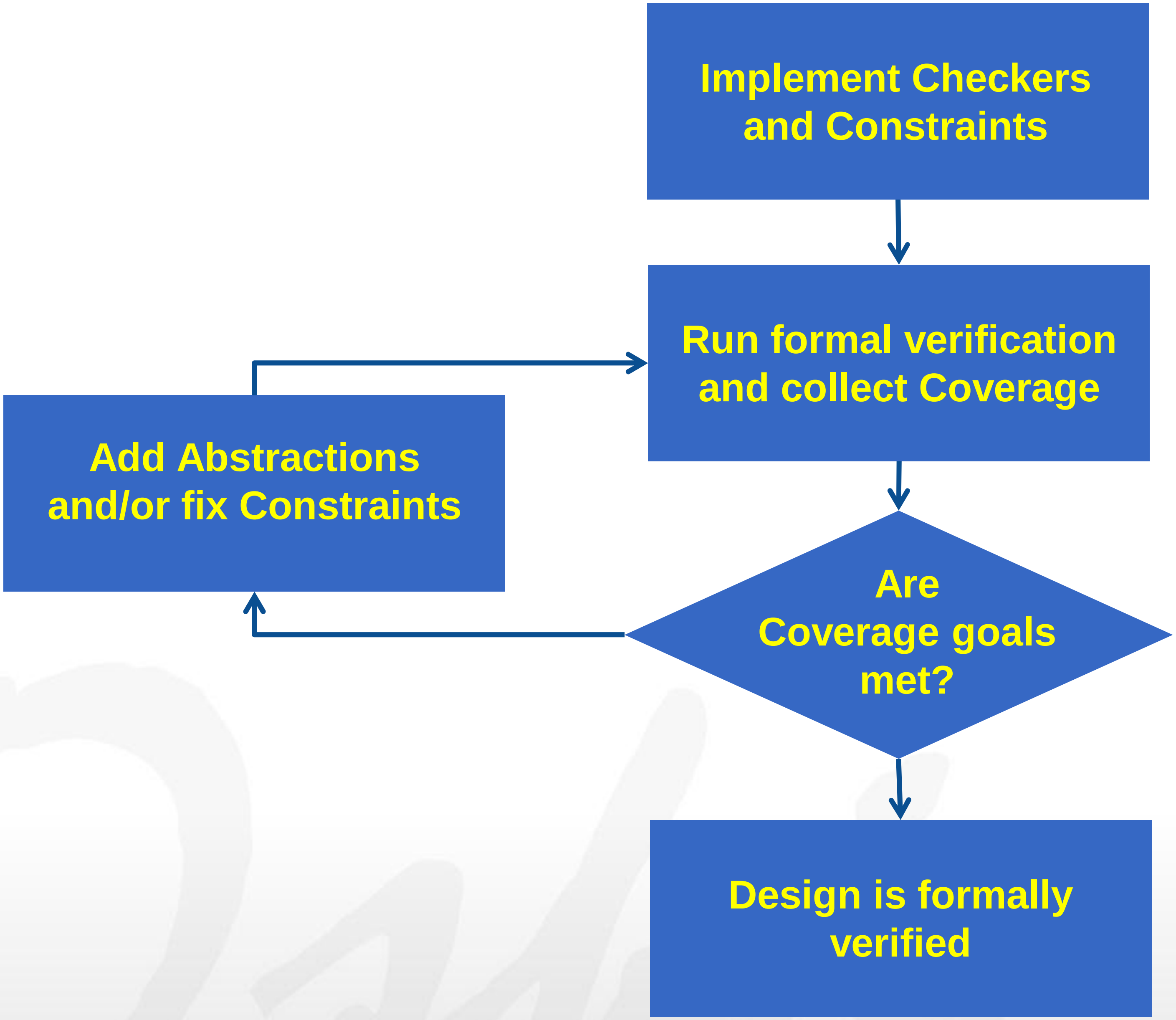
Coverage database collection



Formal coverage integrated with simulation



Formal code coverage methodology


Implement Checkers and Constraints
Run formal verification and collect Coverage
Add Abstractions and/or fix Constraints
Are Coverage goals met?
Design is formally verified

PRM coverage (with abstractions)

- Using Cadence IEV (Incisive Enterprise Verifier)

Proof depth	Line coverage	Expression coverage
7	96.5%	100.0%
15	99.5%	100.0%
63	99.7%	100.0%

- Without abstractions, with 20m run-time, Proof depth reached was still 0 (0% coverage)

Conclusions

- End-to-End formal is what replaces simulation
- Abstractions are necessary to achieve convergence
- Coverage helps measure completeness

Thanks

- Adnan Aziz
- Sandesh Borgaonkar
- Choon Chng
- Harry Foster
- Vineet Gupta
- Anton Lopatinsky
- Deepak Pant
- Philippa Slayton
- Shashidhar Thakur