2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Smart Formal for Scalable Verification

Ashish Darbari

Axiomise

accellera SYSTEMS INITIATIVE

PACKET-BASED DESIGNS

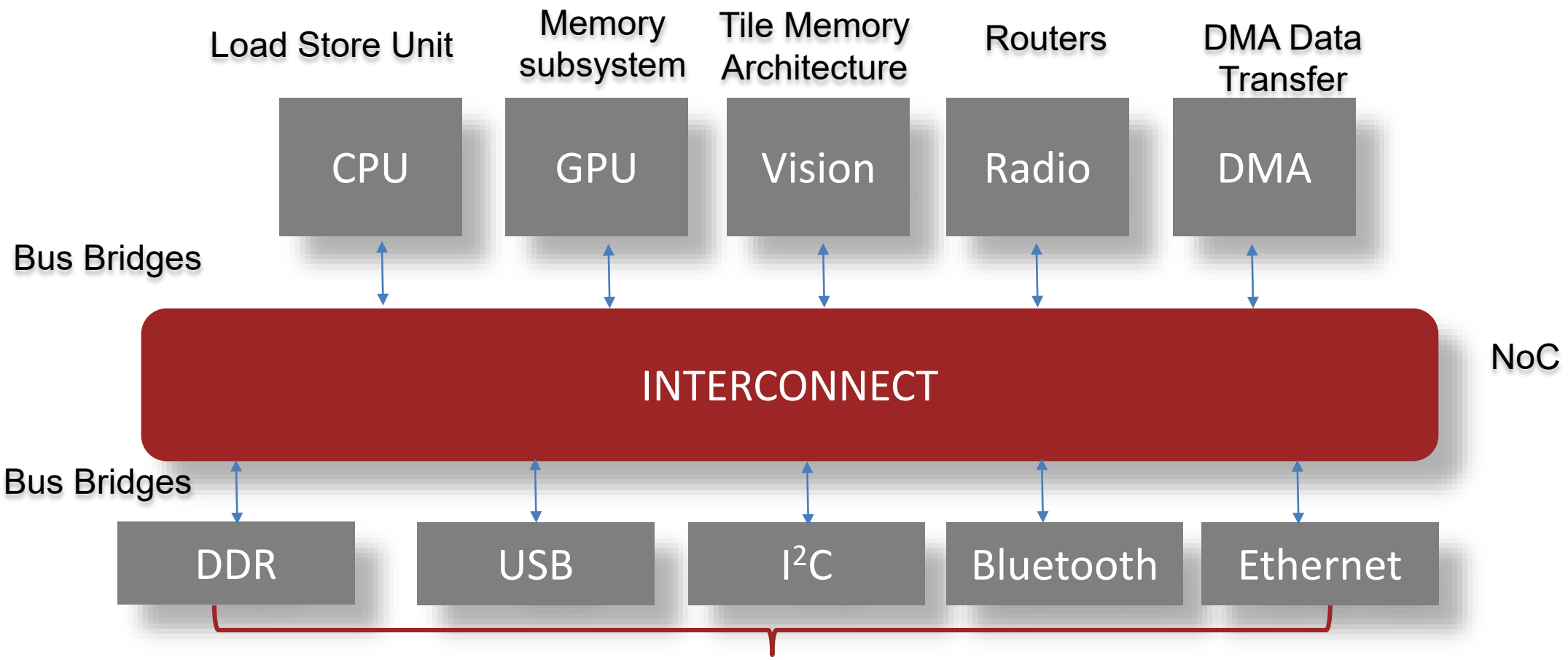
What's the game here?

Sequential Multi-packet Designs

- Units of data flow together in what we call as packets
  - Fixed number of packets
  - Variable number of packets
- Usual requirement is that packets must not be:
  - Reordered
  - Dropped
  - Modified in flight
- State of each packet depends on the state of all others ahead of it

VERIFICATION NIGHTMARE!

SoC Verification

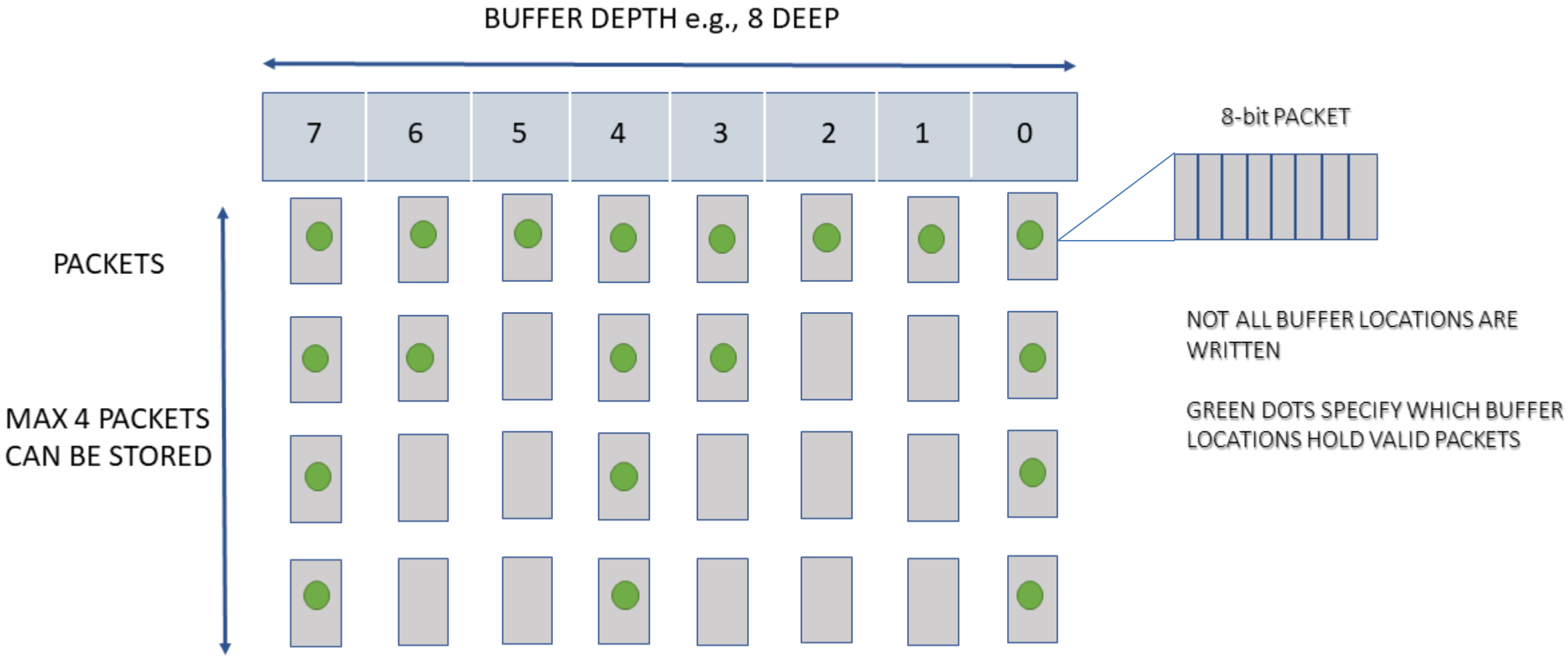


PACKET-BASED DESIGNS ARE A COMMON THEME

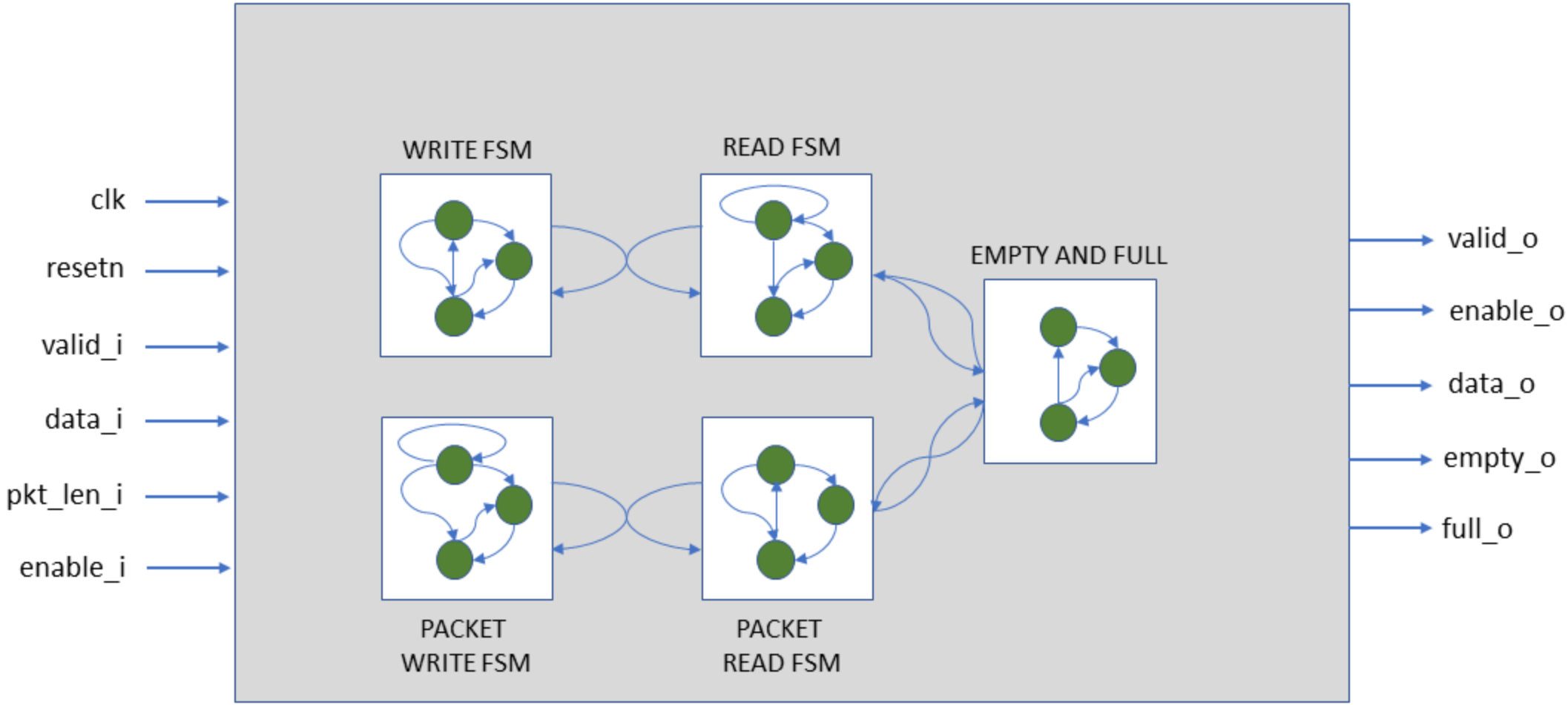
Verification Challenge

- Directed testing is too directed
- Constrained random inherently incomplete – FSMs challenging
- Formal verification capable of
  - Hunting deep corner case bugs
  - Build exhaustive proofs
- But formal without a good methodology doesn't scale

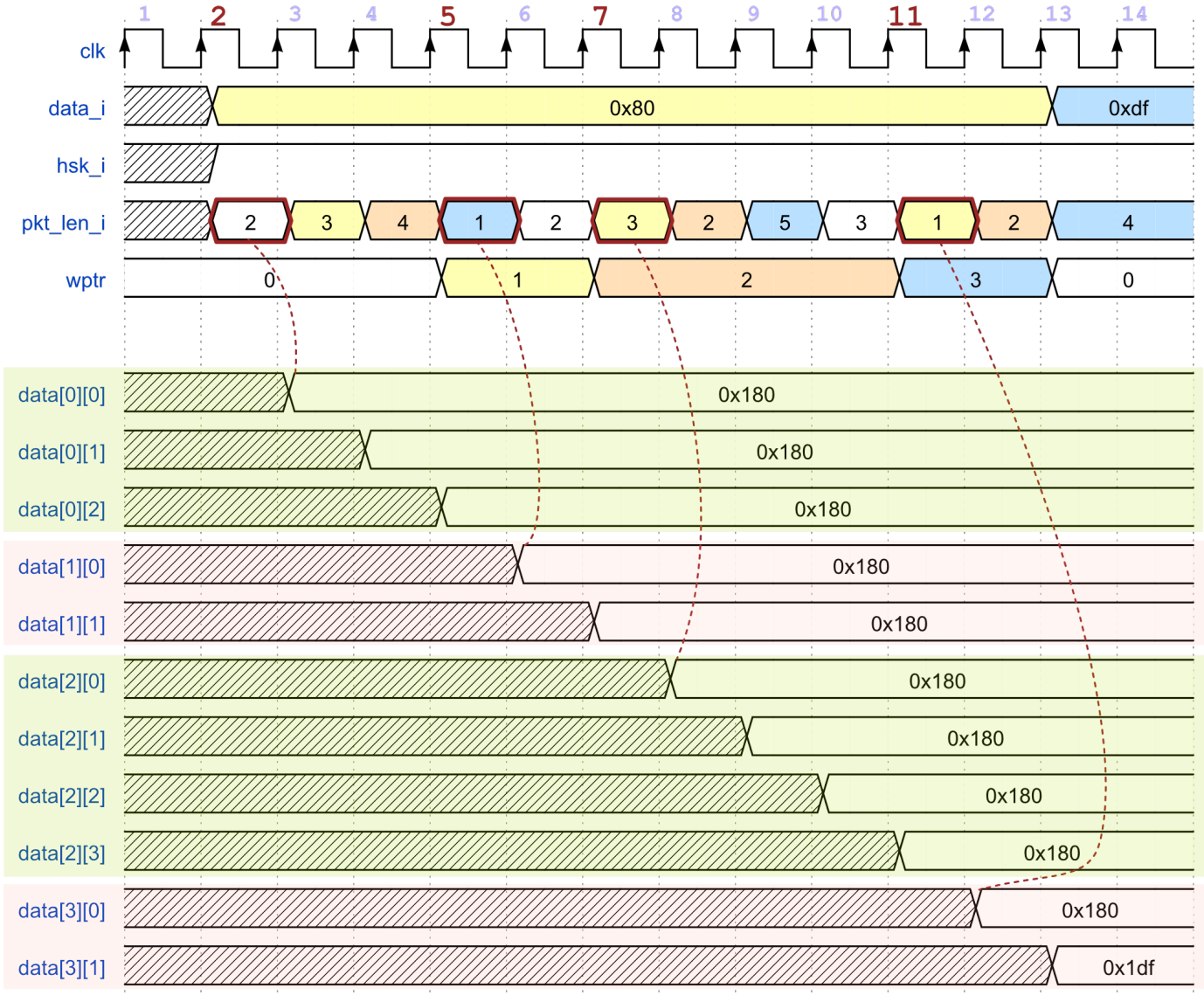
Variable Packet Design



Design Description



Example Behaviour



Buffer depth: 4

No of packets: up to 8

PACKET SIZE: 8-bits

9th valid bit set to 1

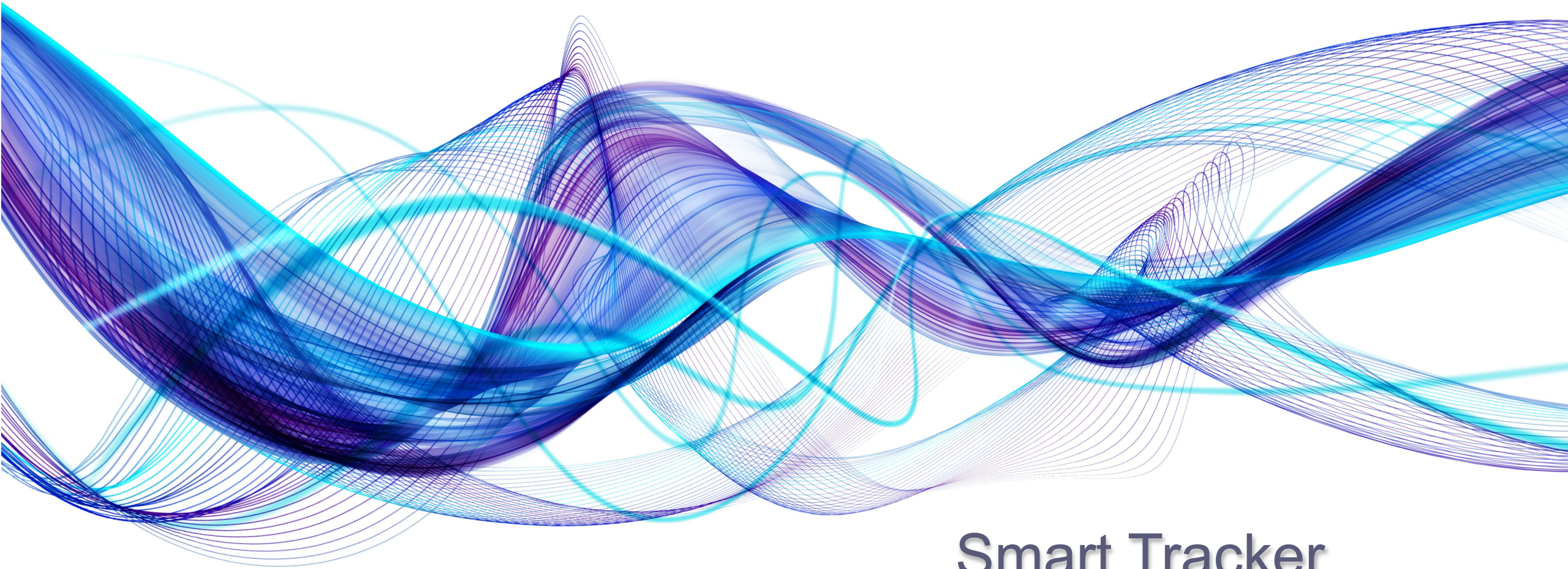
Packet length=pkt_len_i+1

Verification Strategy

- Build mechanisms to track data
- Provide any constraints or assumptions
- Write checks/assertions to establish “correctness always holds”
- Write cover properties to check certain behaviours can occur
- Ensure that you have not missed any bug in your test bench

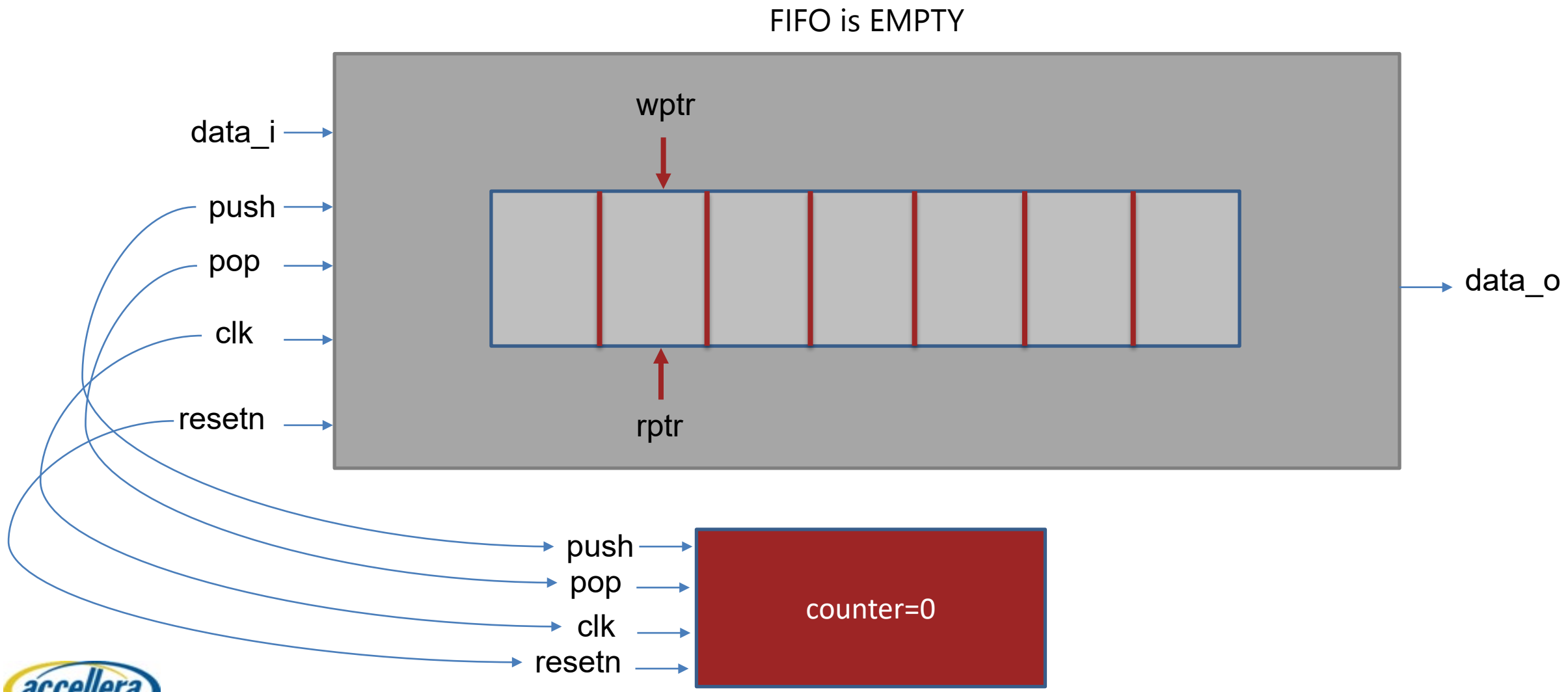
Formal Verification Strategy

- No sequence generation, stimulus is free
- Just constrain out the illegal stimulus
- Checking by formal tools is symbolic – covering all combinations of 0s and 1s
- Track inputs going into the DUT and check if the expected ones come out
  - But, tracking is not done for every incoming value explicitly
  - Track one symbolic value – you track all the values… **yeah seems like magic!**
  - **Designated values that are tracked are called "watched values"**

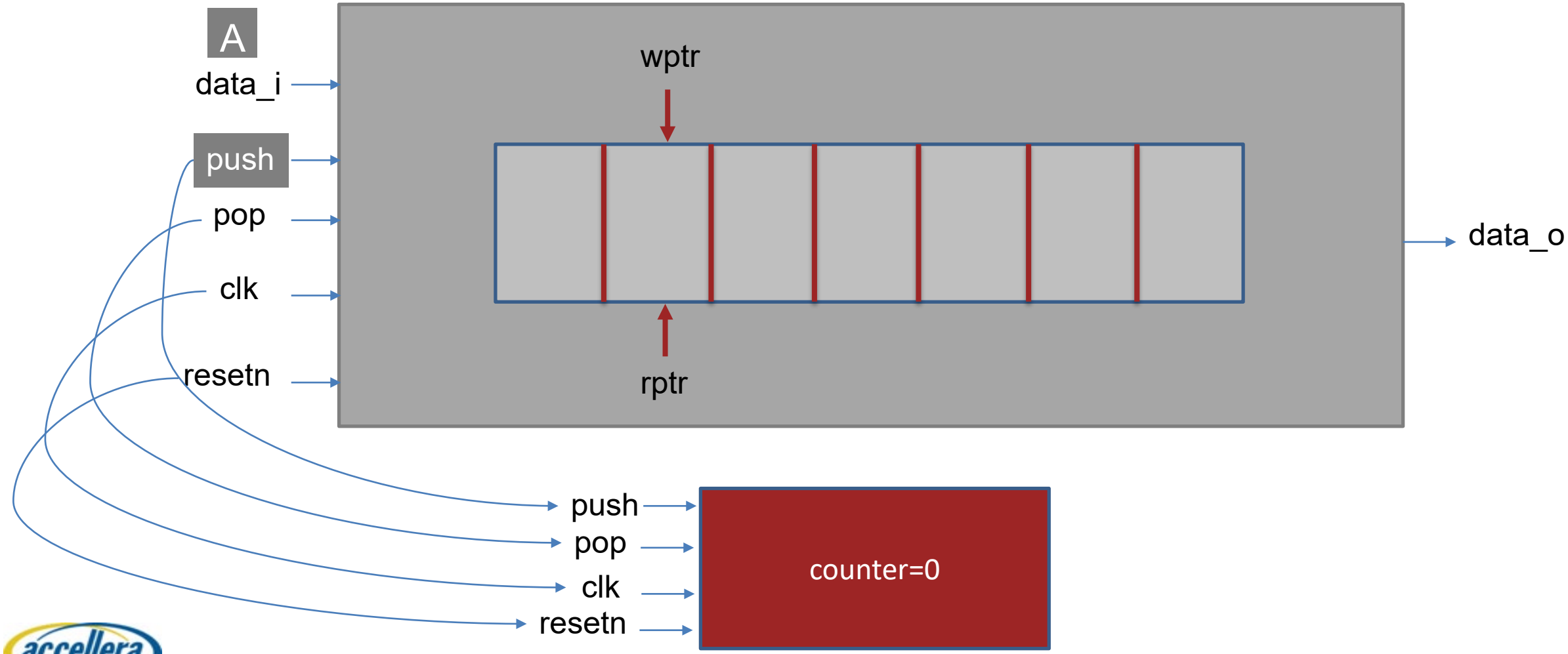


Smart Tracker

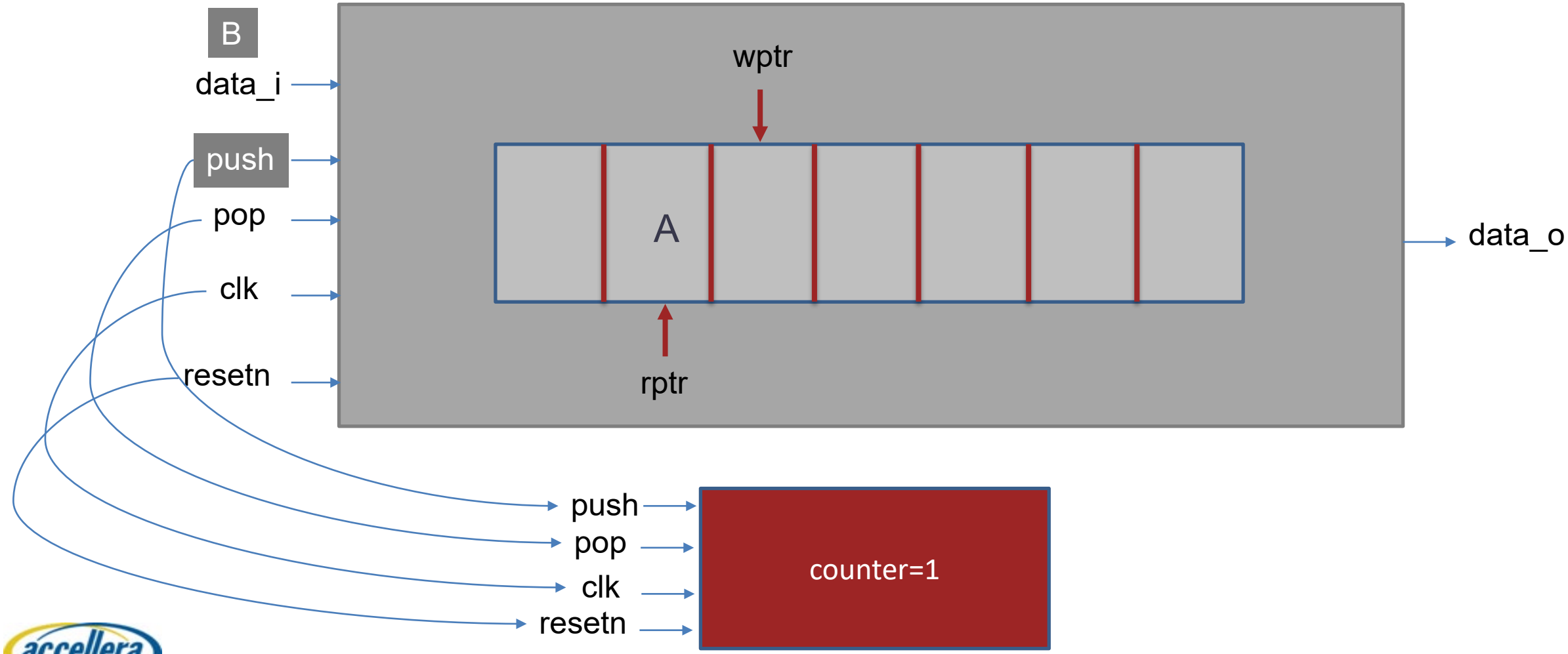
Smart Tracker in Action



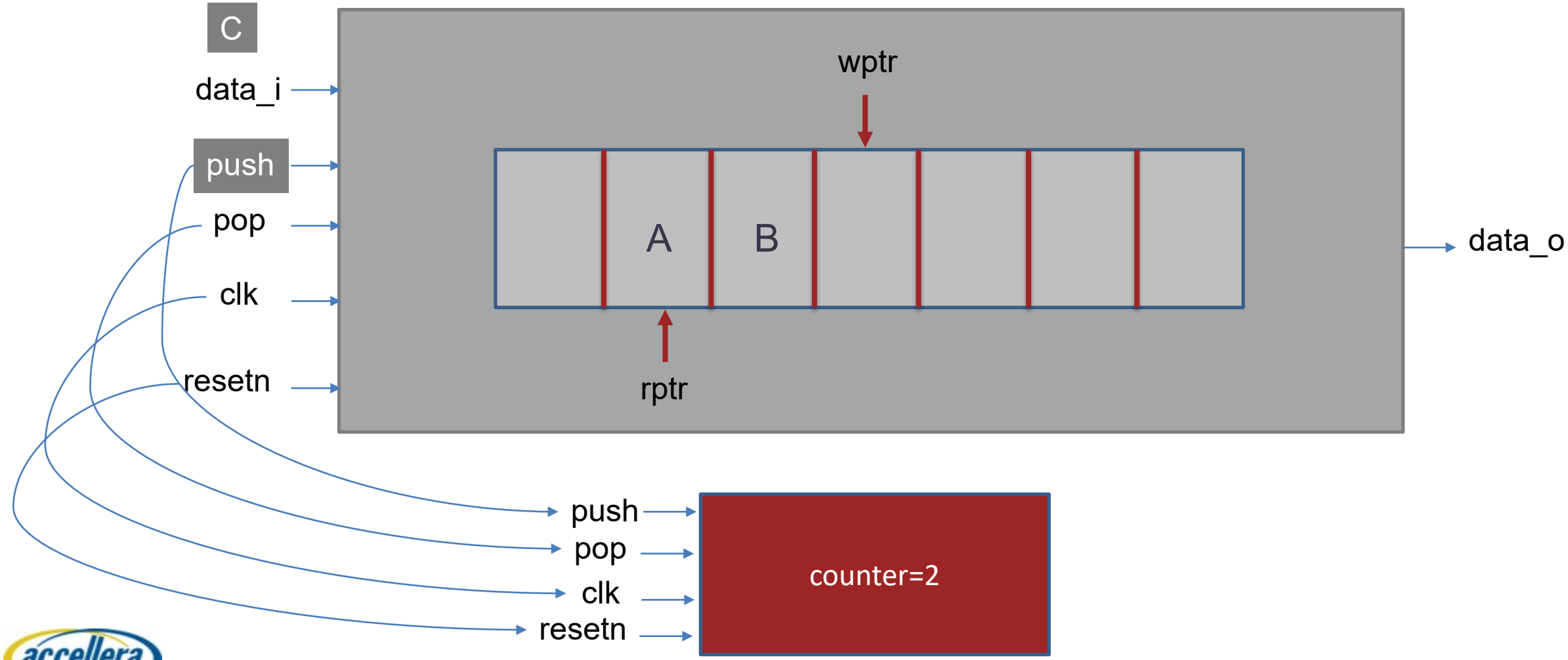
First Write


A
data_i
push
pop
clk
resetn
wptr
rptr
data_o
push
pop
clk
resetn
counter=0

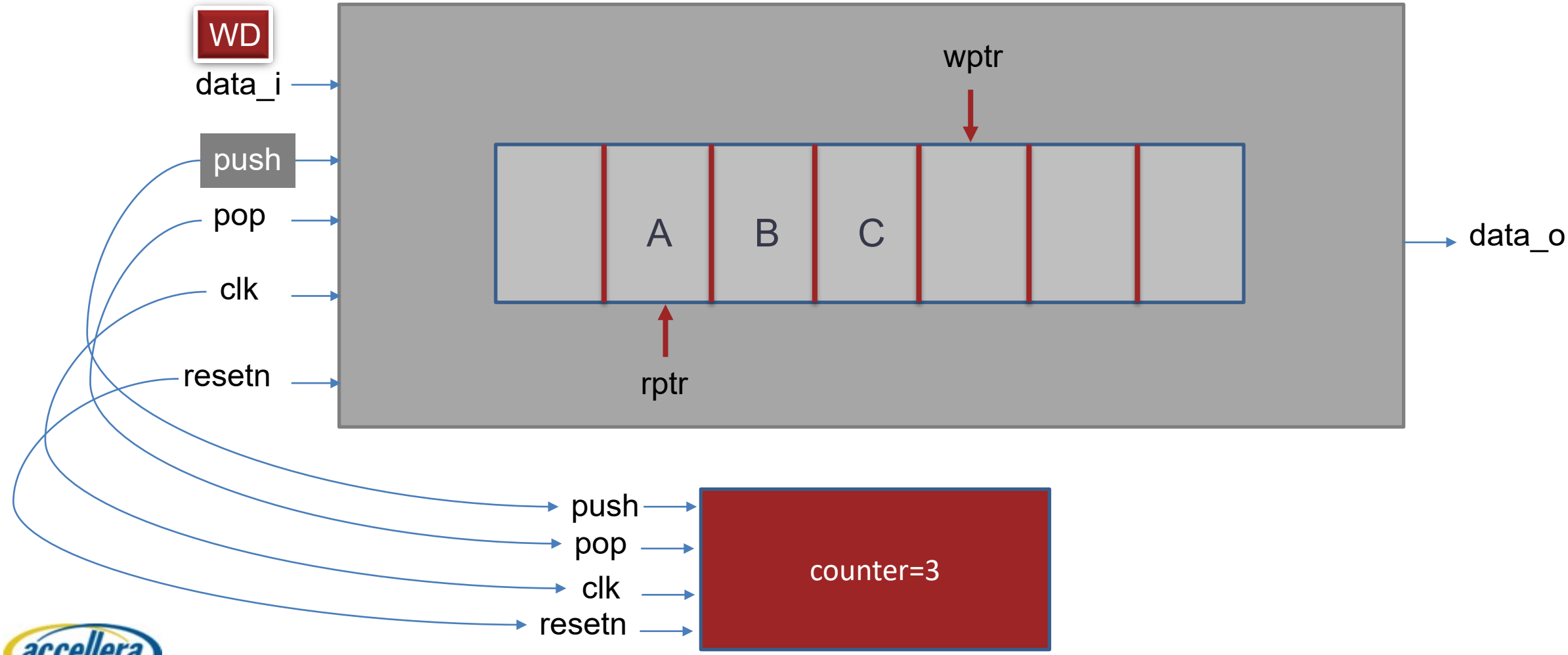
Second Write



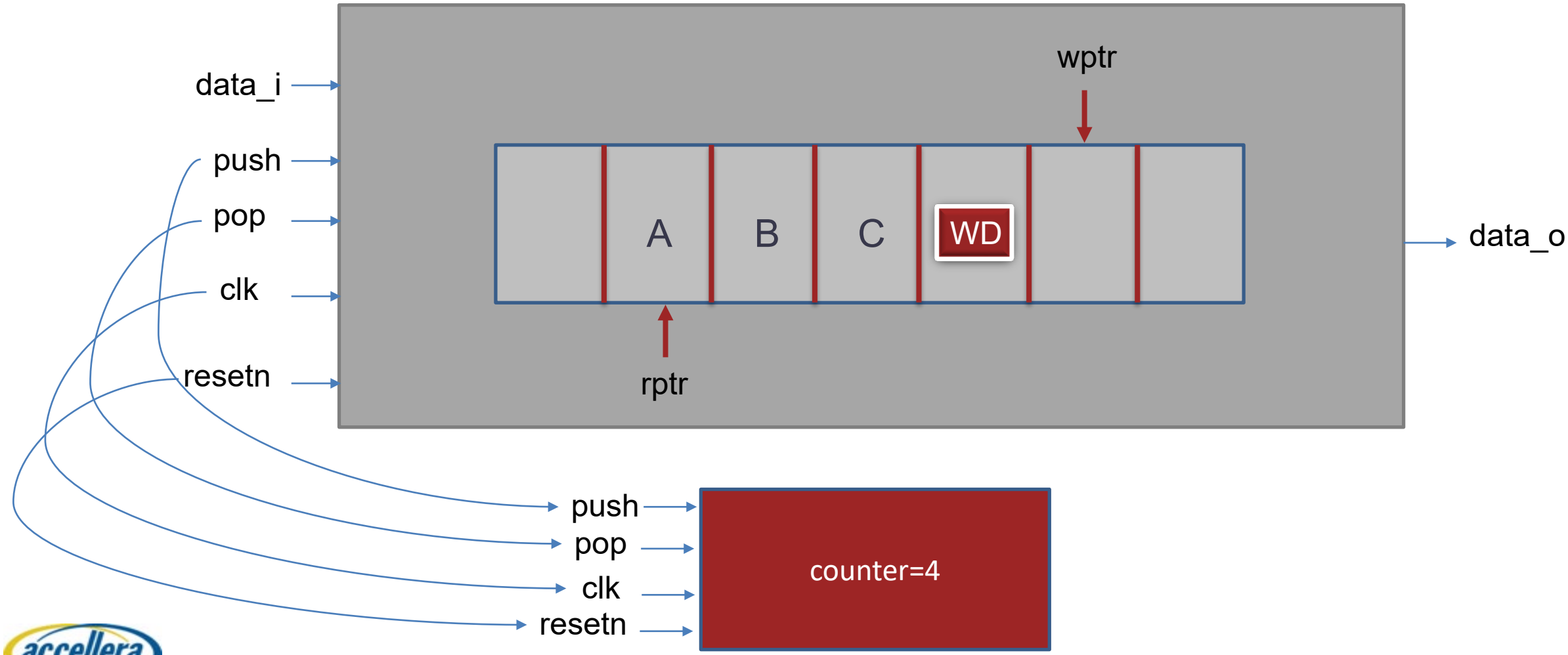
Third Write


C
data_i
push
pop
clk
resetn
wptr
A
B
rptr
data_o
push
pop
clk
resetn
counter=2

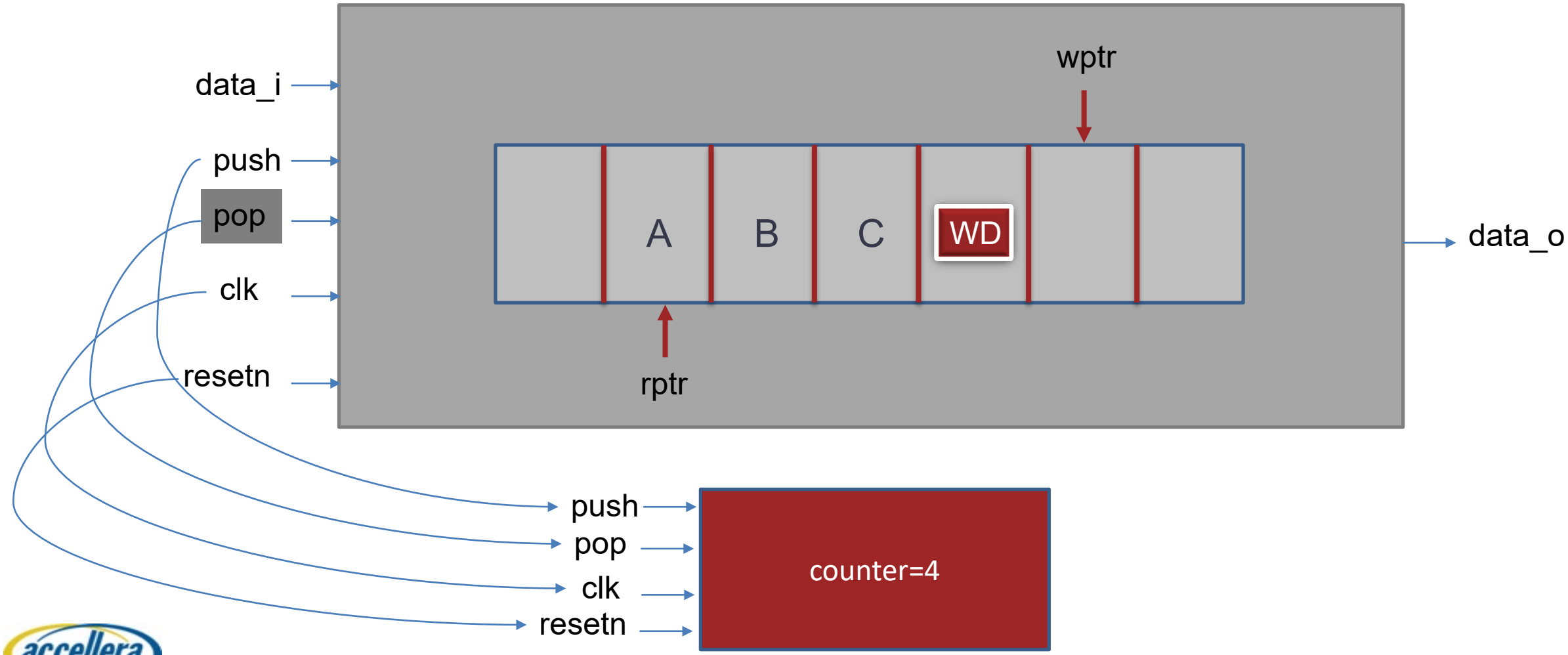
Watched Data Appears on Input



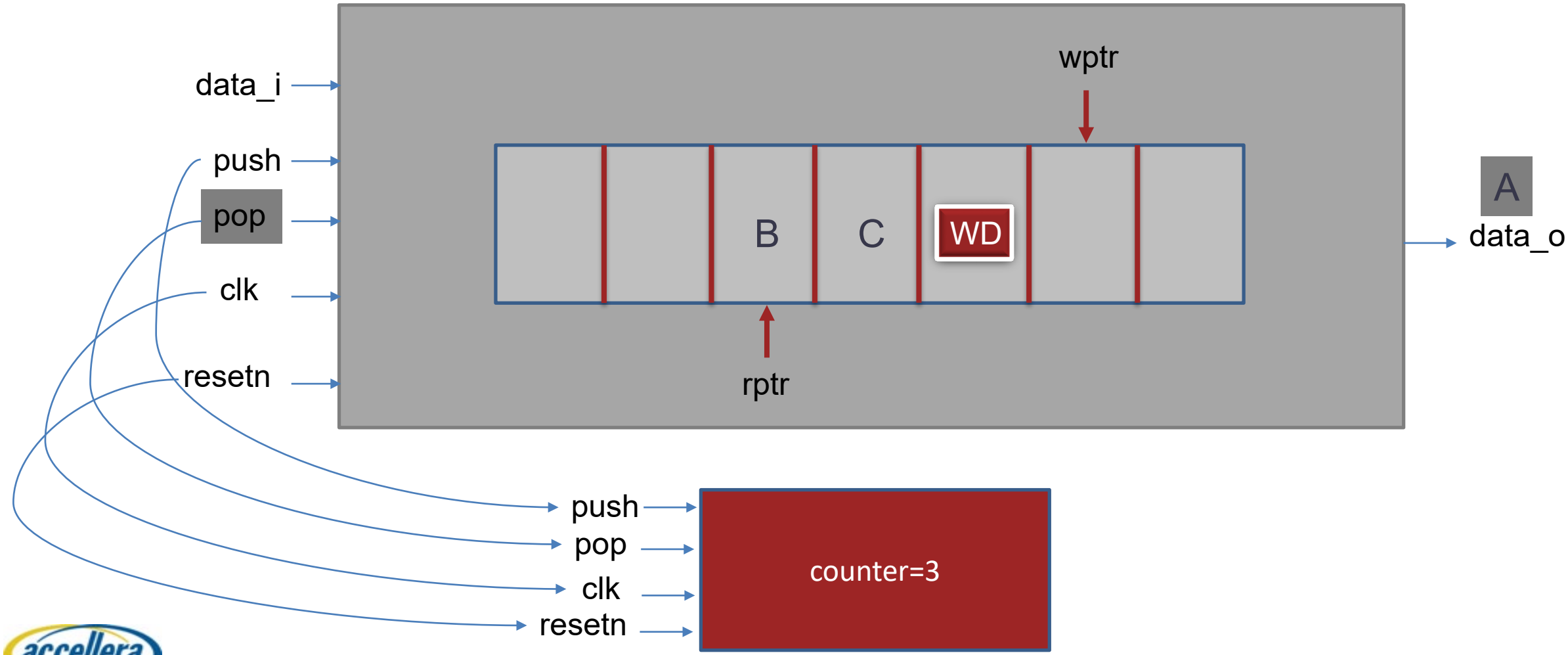
Three Elements Ahead of Watched Data



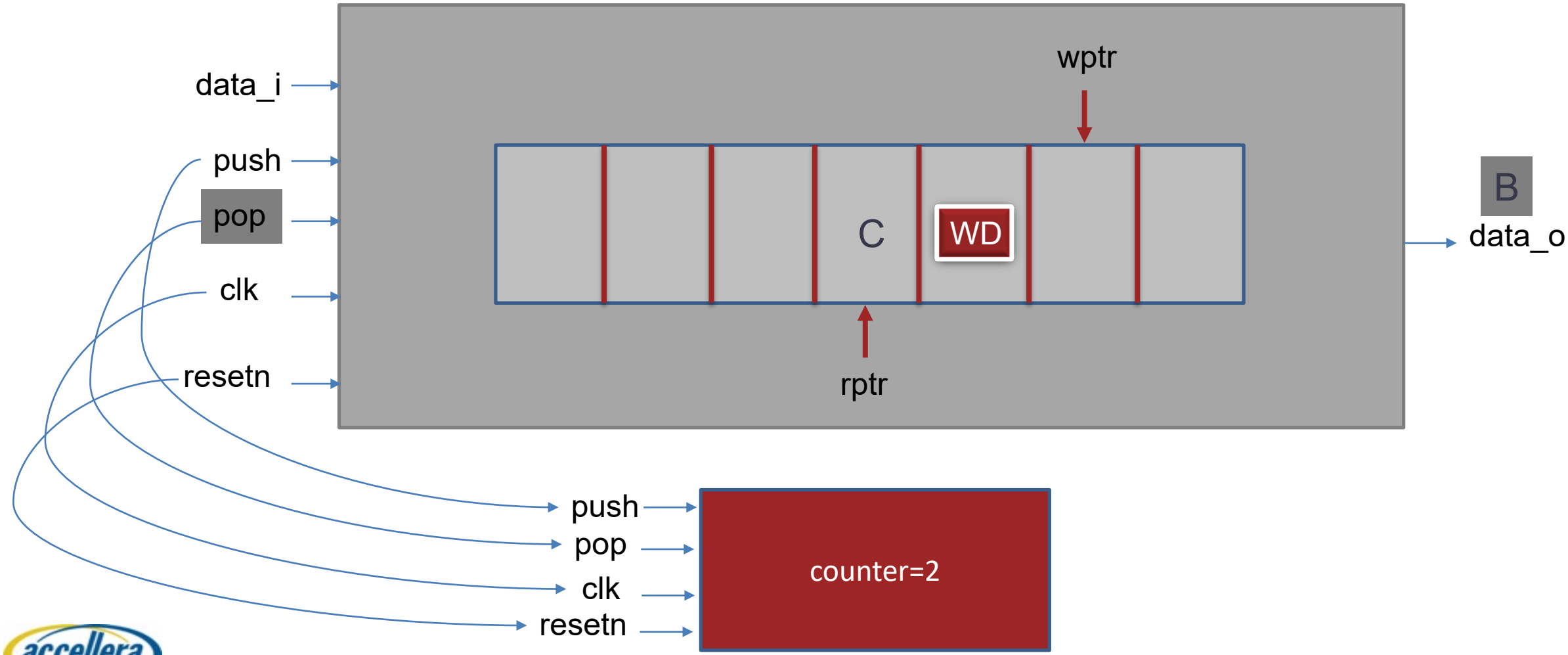
Reading The Data



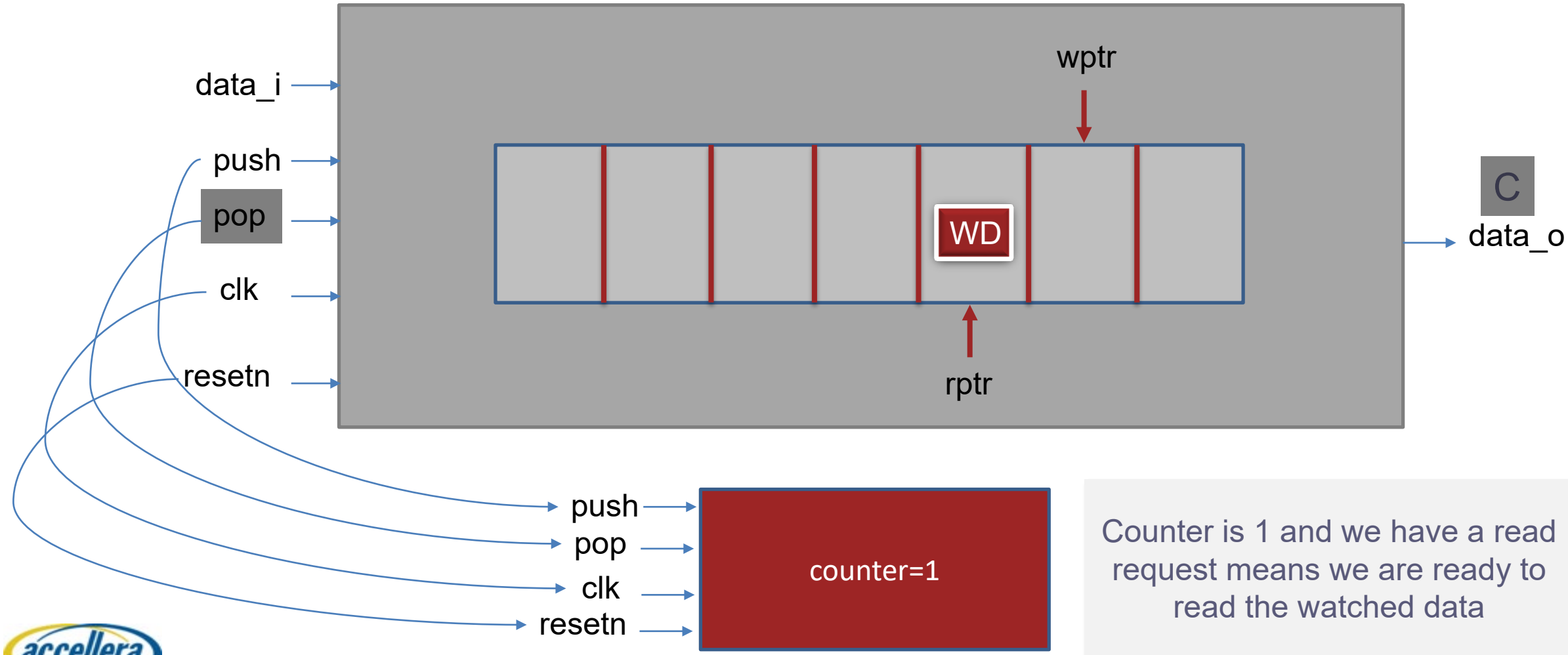
Reading The Data



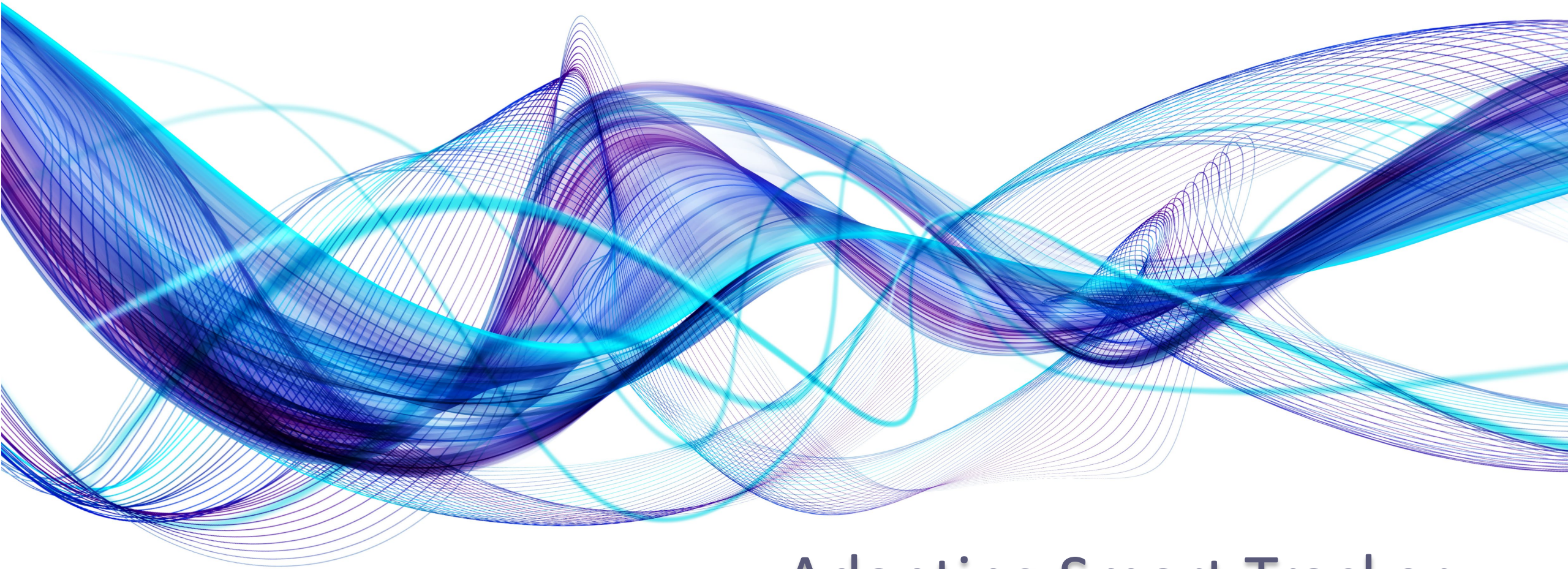
Reading The Data



Reading The Data



Counter is 1 and we have a read request means we are ready to read the watched data



Adapting Smart Tracker

Adapting Smart Tracker

- Tracking would have to be done for all packets, not just one!
- Challenge is that we have variable number of packets, not just fixed size!
- Watched data can be any packet between 0 and P-1 for a max of P packets!
- Checkers verify that the design works for all watched packets not just one!

Observing Sampling In

```
always @(posedge clk or negedge resetn)
if (!resetn)
        sample_in_started <= 1'b0;
else
        sample_in_started <= ready_to_start_sampling_in;

always @(posedge clk or negedge resetn)
if (!resetn)
        sample_in_finished <= 1'b0;
else
        sample_in_finished <= ready_to_finish_sampling_in;
```

Observing Sampling Out

```
always @(posedge clk or negedge resetn)
if (!resetn)
        sample_out_started <= 1'b0;
else
        sample_out_started <= ready_to_start_sampling_out;

always @(posedge clk or negedge resetn)
if (!resetn)
        sample_out_finished <= 1'b0;
else
        sample_out_finished <= ready_to_finish_sampling_out;
```

Ready to Sample?

```
assign ready_to_start_sampling_in   = sample_in_started  || sample_in_started_c;
assign ready_to_finish_sampling_in  = sample_in_finished || sample_in_finished_c;
assign ready_to_start_sampling_out  = sample_out_started   ||
                                      sample_out_started_c;
assign ready_to_finish_sampling_out = sample_out_finished  ||
                                      sample_out_finished_c;
```

`sample_in_started_c:`	on an input handshake when the very first input data matches wd[0]
`sample_in_finish_c:`	if we already started and are going to finish and have an input handshake
`sample_out_started_c:`	start sampling out when tracking counter is 1
`sample_out_finish_c:`	stop sampling out if all packets would be be read out and we started to sample out

Tracking Counter

```
always @(posedge clk or negedge resetn)
if (!resetn)
        tracking_counter <= 'h0;
else if
        tracking_counter <= tracking_counter + incr - decr;

assign incr = hsk_i && (pkt_wptr[wptr]==cpkt_len[wptr]) && !input_sampled_completely;

assign decr = hsk_o && (pkt_rptr[rptr]==0) && !sample_out_started;
```

```
sample_in_started && sample_in_finished;
```

Counting Output Packets

```
always @(posedge clk or negedge resetn)
if (!resetn)
        counter_out <= 'h0;
else if (input_sampled_completely && not_sampled_out_completely)
        counter_out <= counter_out + 1'b1;
```

Master Ordering Checks

```
generate
  for (i=0;i<PKT_LEN-1;i=i+1) begin: as_all_packets
            but_last:
                assert property (input_sampled_completely    &&
                                 not_sampled_out_completely &&
                                (counter_out==i)
                                 |=>
                                 data_o==wd[i]);
                              end
endgenerate
```

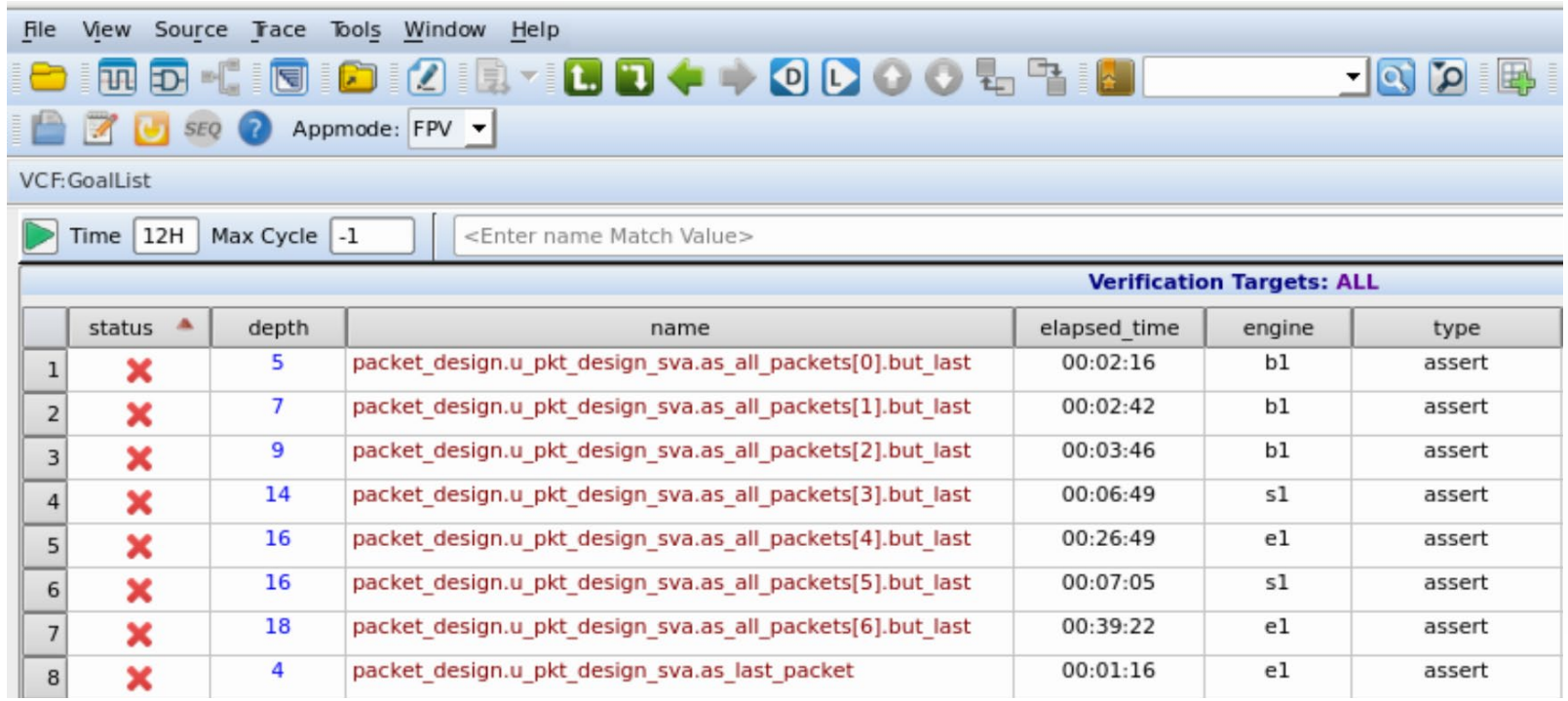
```
as_last_packet:
              assert property (input_sampled_completely && sample_out_started &&
                               $rose(sample_out_finished)
                                   |->
                               (data_o==wd[PKT_LEN-1] || data_o==wd[0]));
```

Packets of length one

Catching Bugs

Catching Reordering Bug

Carried out on an Intel i5 7300 with 6 GB RAM



	status	depth	name	elapsed_time	engine	type
1	✗	5	packet_design.u_pkt_design_sva.as_all_packets[0].but_last	00:02:16	b1	assert
2	✗	7	packet_design.u_pkt_design_sva.as_all_packets[1].but_last	00:02:42	b1	assert
3	✗	9	packet_design.u_pkt_design_sva.as_all_packets[2].but_last	00:03:46	b1	assert
4	✗	14	packet_design.u_pkt_design_sva.as_all_packets[3].but_last	00:06:49	s1	assert
5	✗	16	packet_design.u_pkt_design_sva.as_all_packets[4].but_last	00:26:49	e1	assert
6	✗	16	packet_design.u_pkt_design_sva.as_all_packets[5].but_last	00:07:05	s1	assert
7	✗	18	packet_design.u_pkt_design_sva.as_all_packets[6].but_last	00:39:22	e1	assert
8	✗	4	packet_design.u_pkt_design_sva.as_last_packet	00:01:16	e1	assert

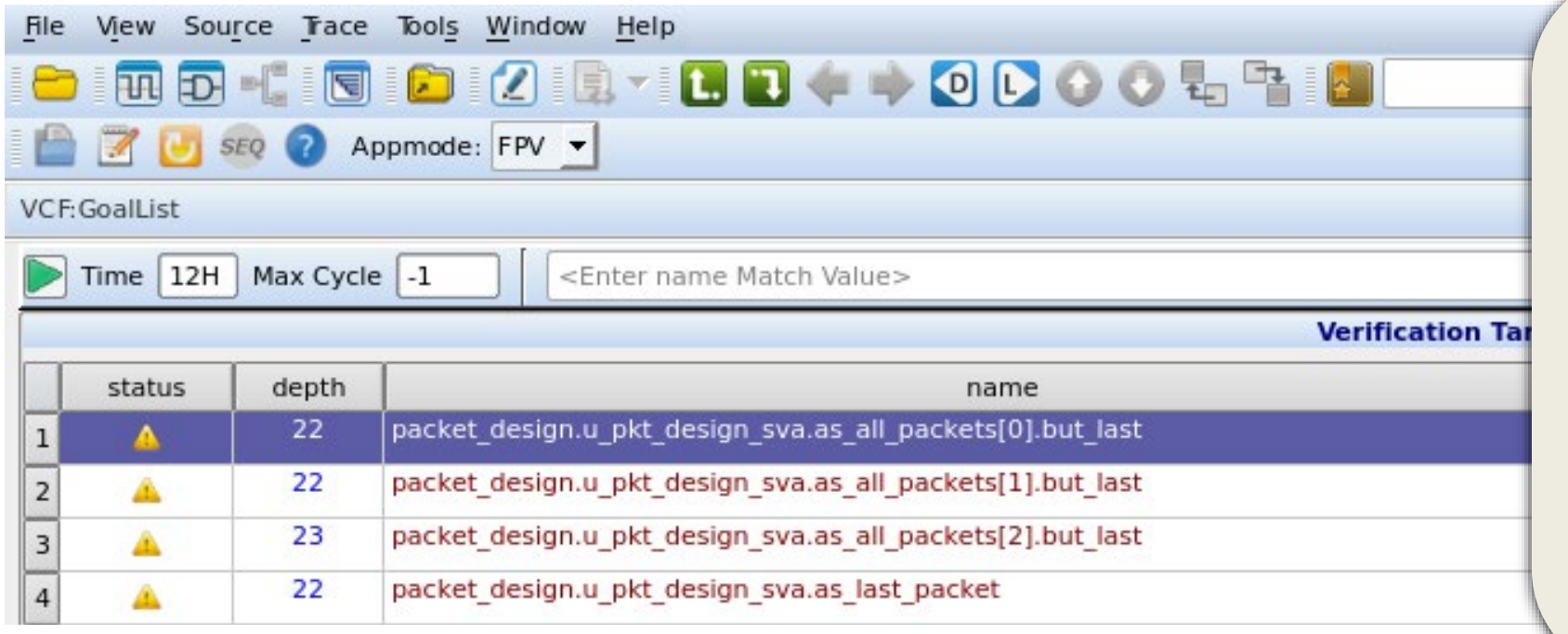
$10^{42,248}$ states
~141K flops

512-deep buffer
Up to 8 packets
Packet size: 32 bit

Bug found in less than 40 minutes

Tool shown: Synopsys VC Formal

Inconclusive Proofs



10^{38} states!
~128 flops

8-deep buffer
Up to 4-packets
Packet size: 4 bit

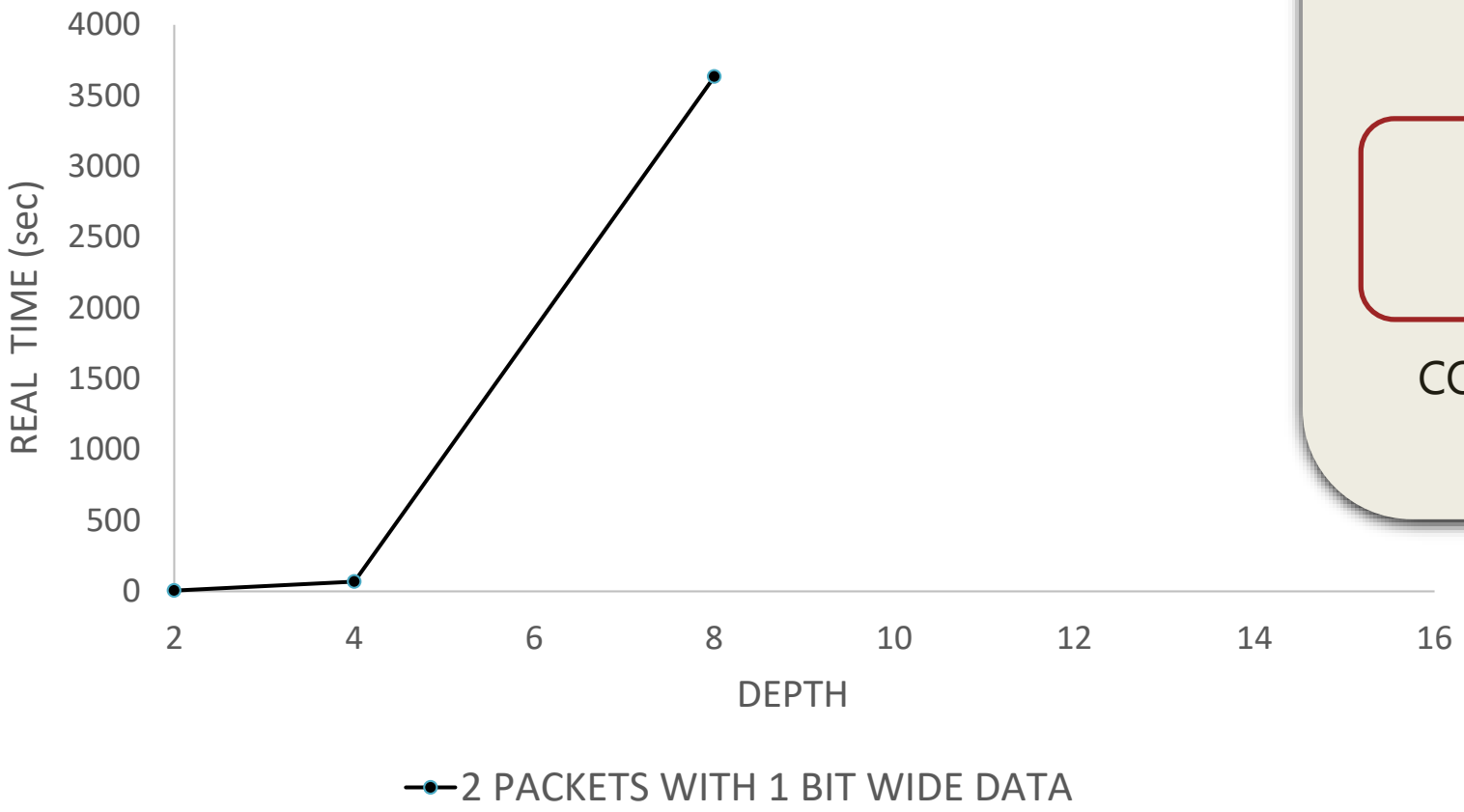
INCONCLUSIVE PROOF

Carried out on an Intel i5 7300 with 6 GB RAM

What's the Biggest Design Provable in an Hour?

When rubber hits the ground

Increasing Buffer Depth

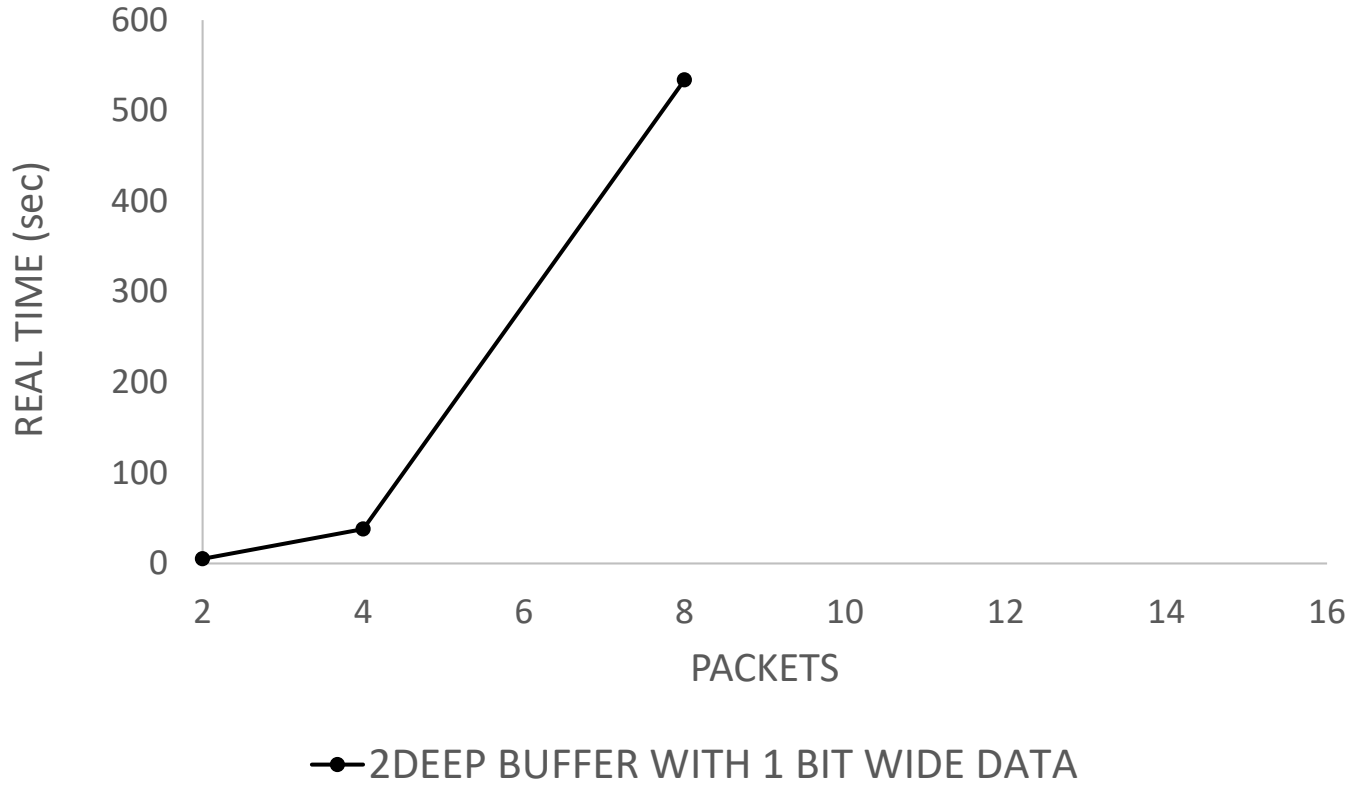


10^5 states!
~16 flops

8-deep buffer
Up to 2-packets
Packet size: 1 bit

CONCLUSIVE PROOF

Increasing Number of Packets

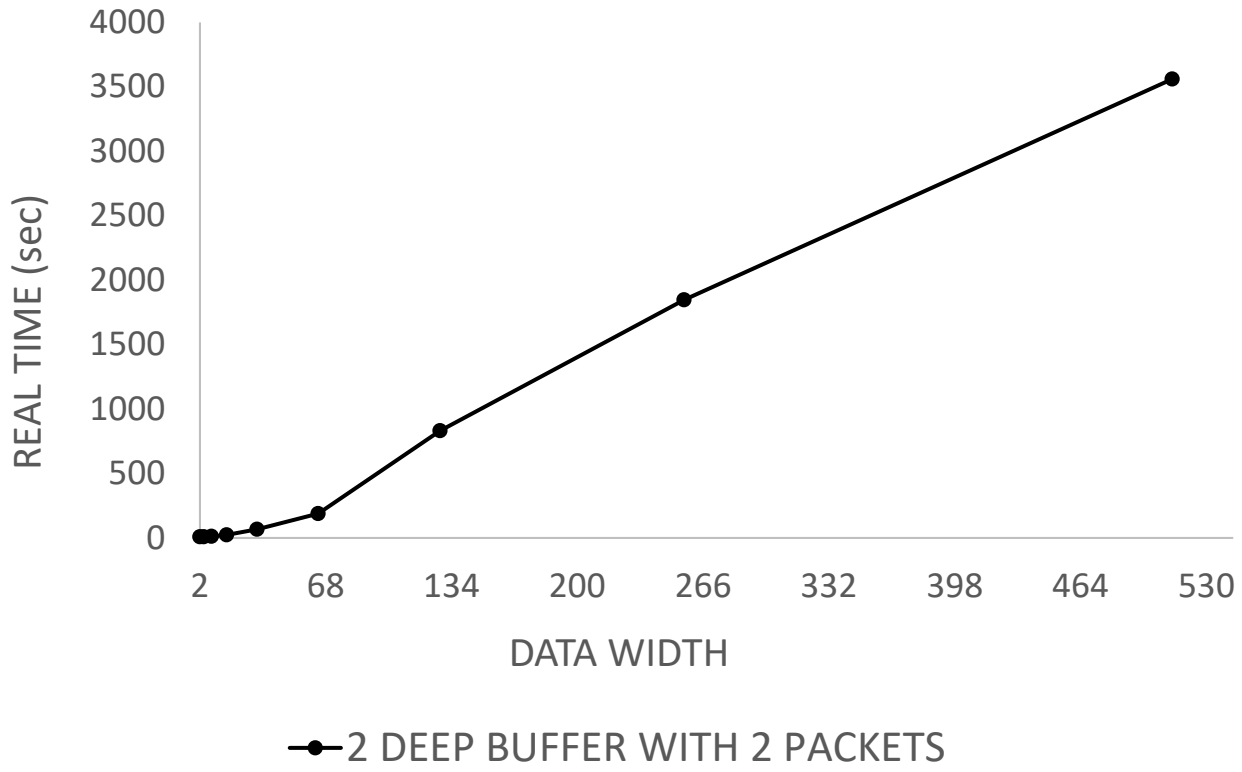


10^5 states!
~16 flops

2-deep buffer
Up to 8-packets
Packet size: 1 bit

CONCLUSIVE PROOF

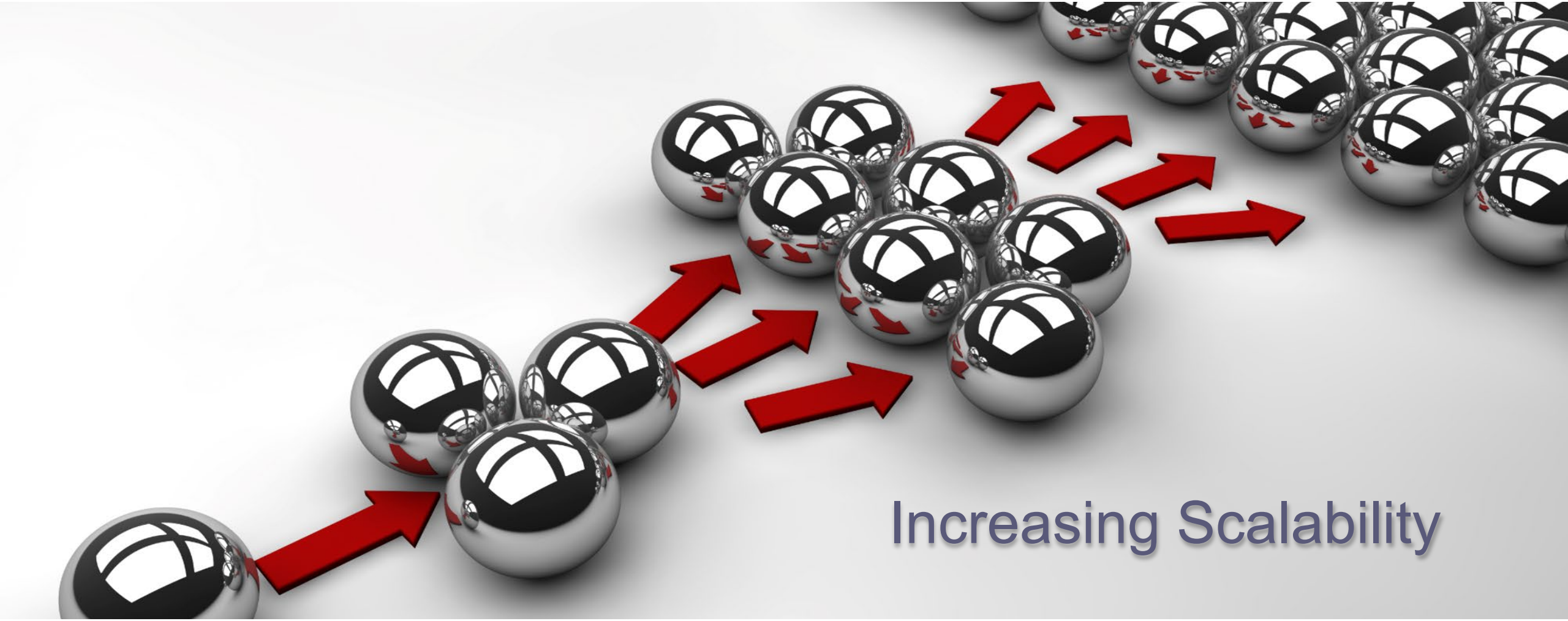
Increasing Packet Size



10^{616} states!
~2048 flops

2-deep buffer
Up to 2-packets
Packet size: 512 bit

CONCLUSIVE PROOF



Increasing Scalability

Proof Engineering

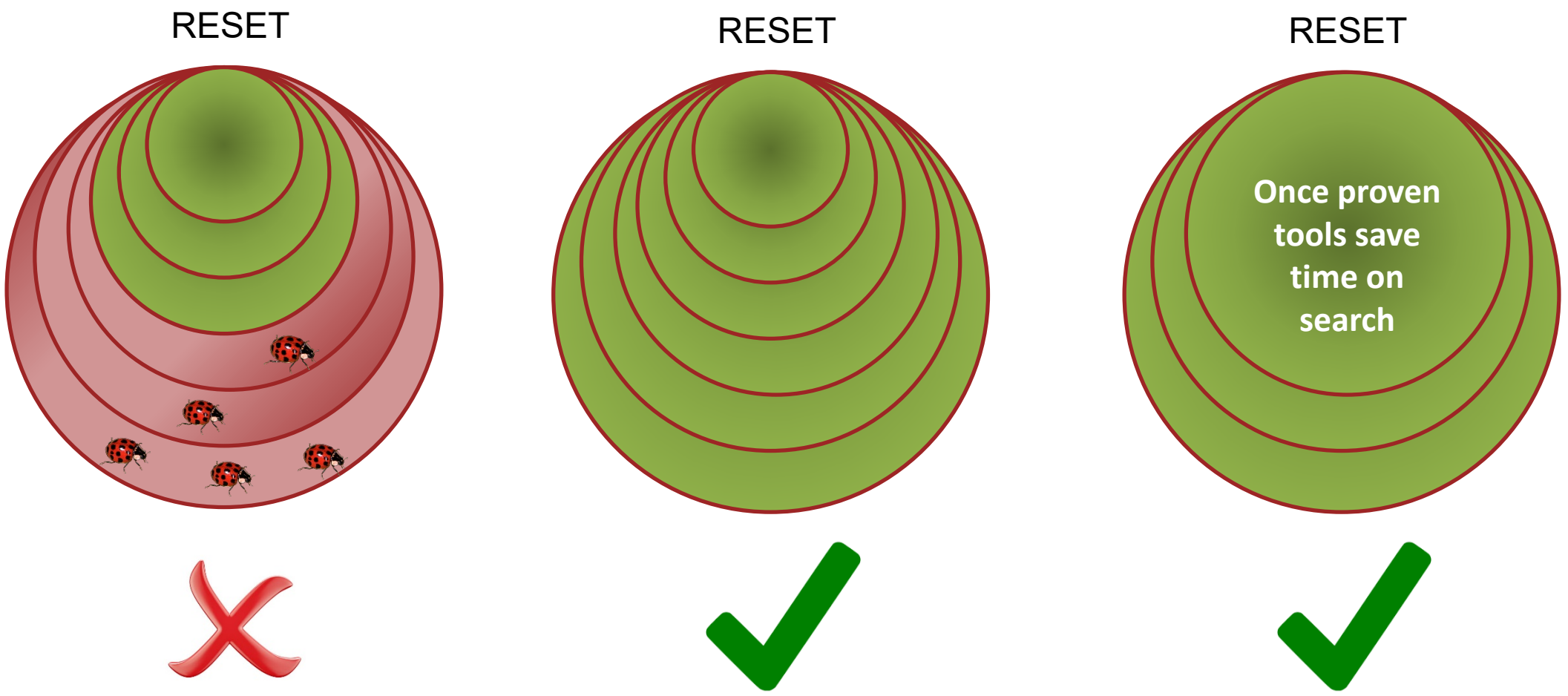
Scalable formal verification
=
“Proof Engineering”

Invariants
Assume Guarantee

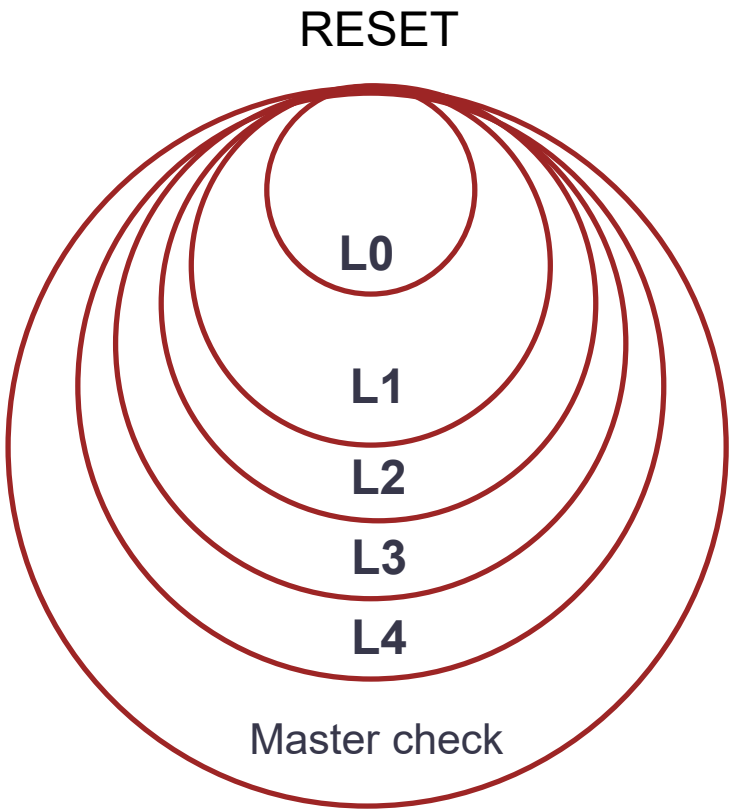
Invariants and Assume Guarantee

- Break the whole puzzle into smaller jigsaws
- Identify helper lemmas as individual components of jigsaw
- Identify how they fit together to complete the full puzzle
- PROVE helper lemmas then ASSUME them to prove other lemmas

Invariants Help in Proof Convergence

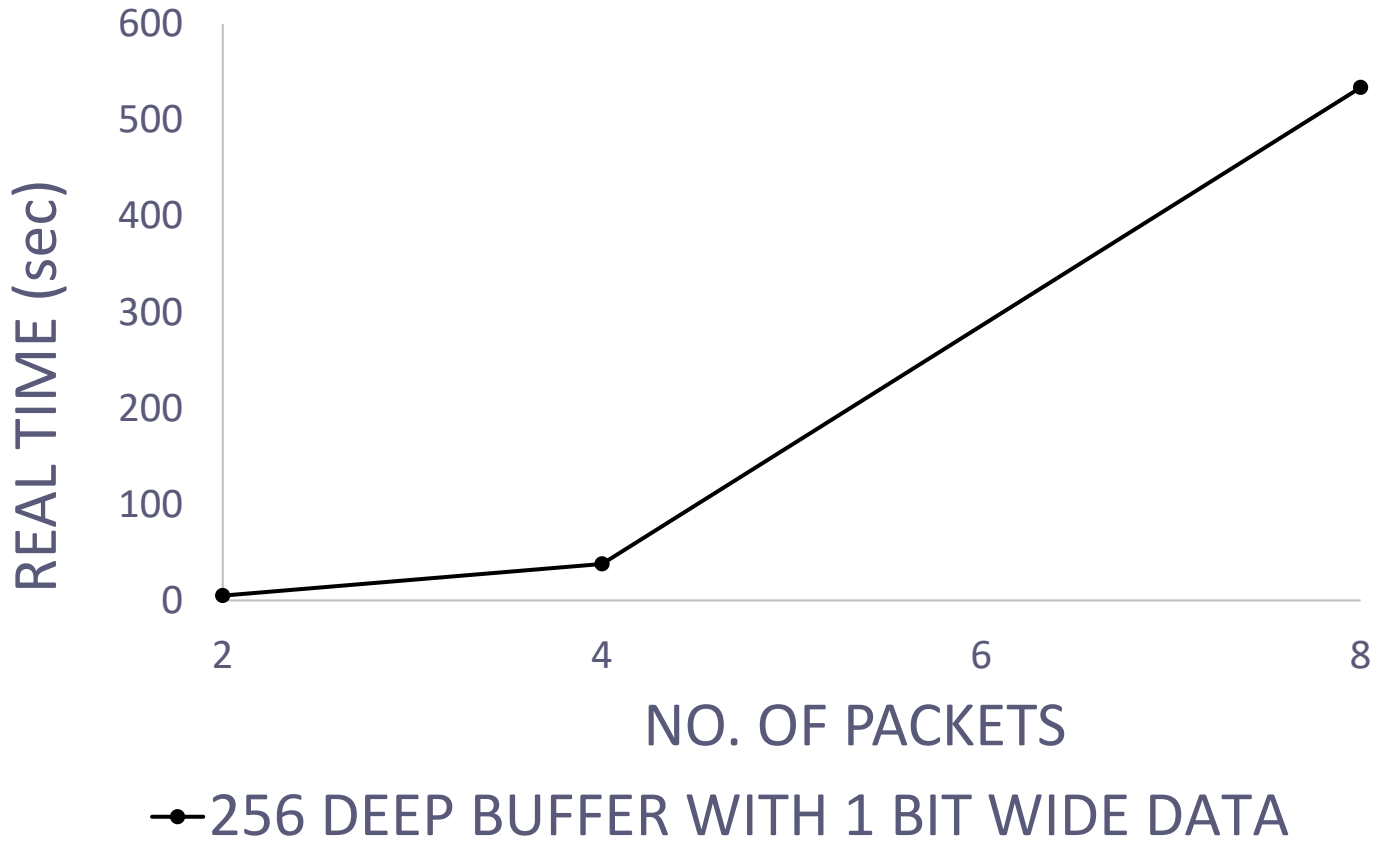


Invariants and Assume Guarantee



L0: If **sample in started and sample out finished** then the tracking counter is zero.
L1: If watched data input **has not** been sampled in then *tracking counter* is *less than or equal to* the difference between the write and the read pointers.
L2: If watched data input **has been** sampled in but sampling out has not started then *the tracking counter is less than or equal to* the difference between the write and the read pointers.
L3: If the **sampling out has not started** then the *very first watched data* that was sampled in the design must be residing at the location given by the function of read pointer and tracking counter.
L4: If the **very first watched data value** has been seen at the output port then *the next data value* to be seen on the output will be the next watched data value sampled into the design.

Increasing Packets

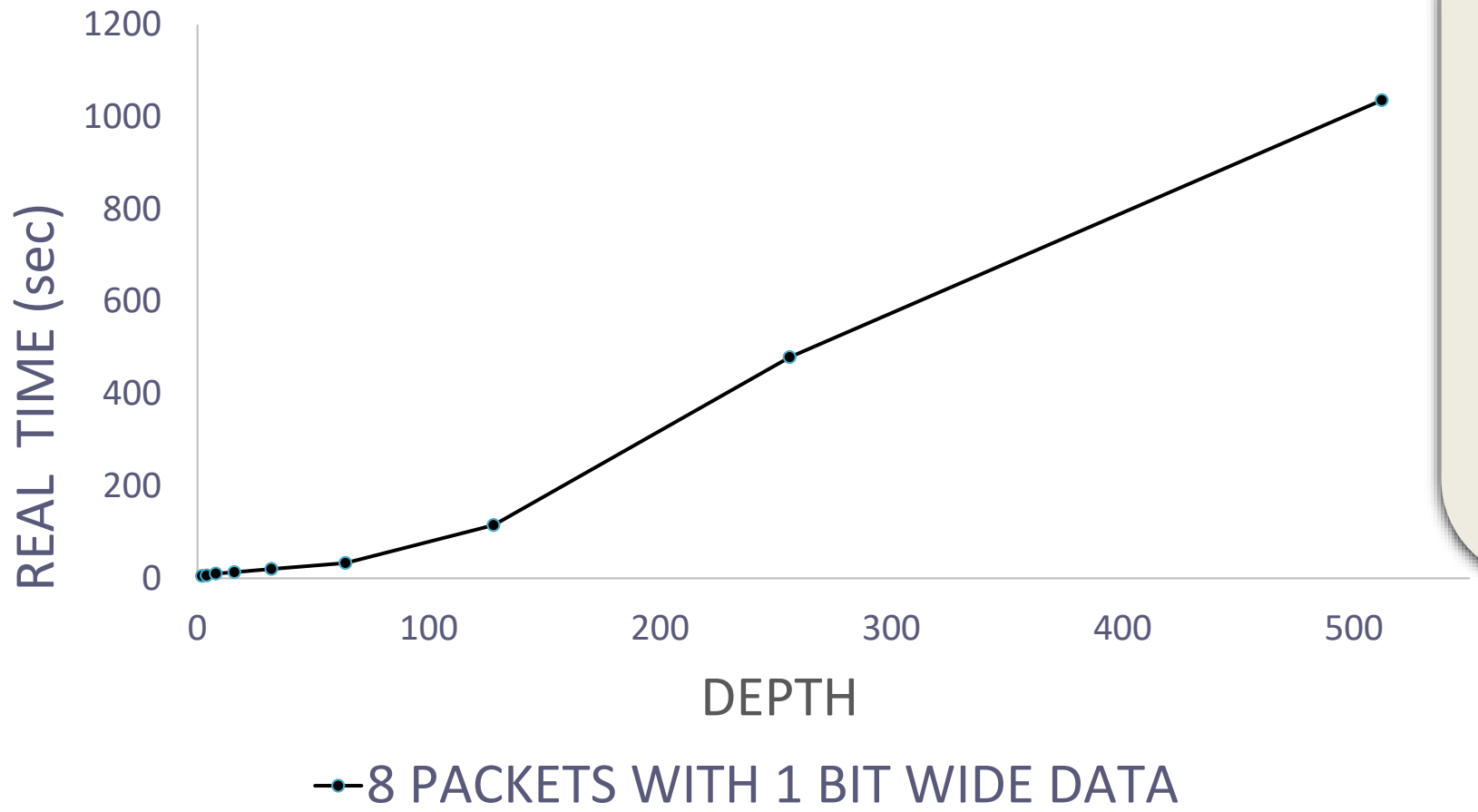


10^{616} states!
~2048 flops

256-deep buffer
Up to 8-packets
Packet size: 1 bit

PROOF in 7 min 59 sec

Increasing Buffer Depth

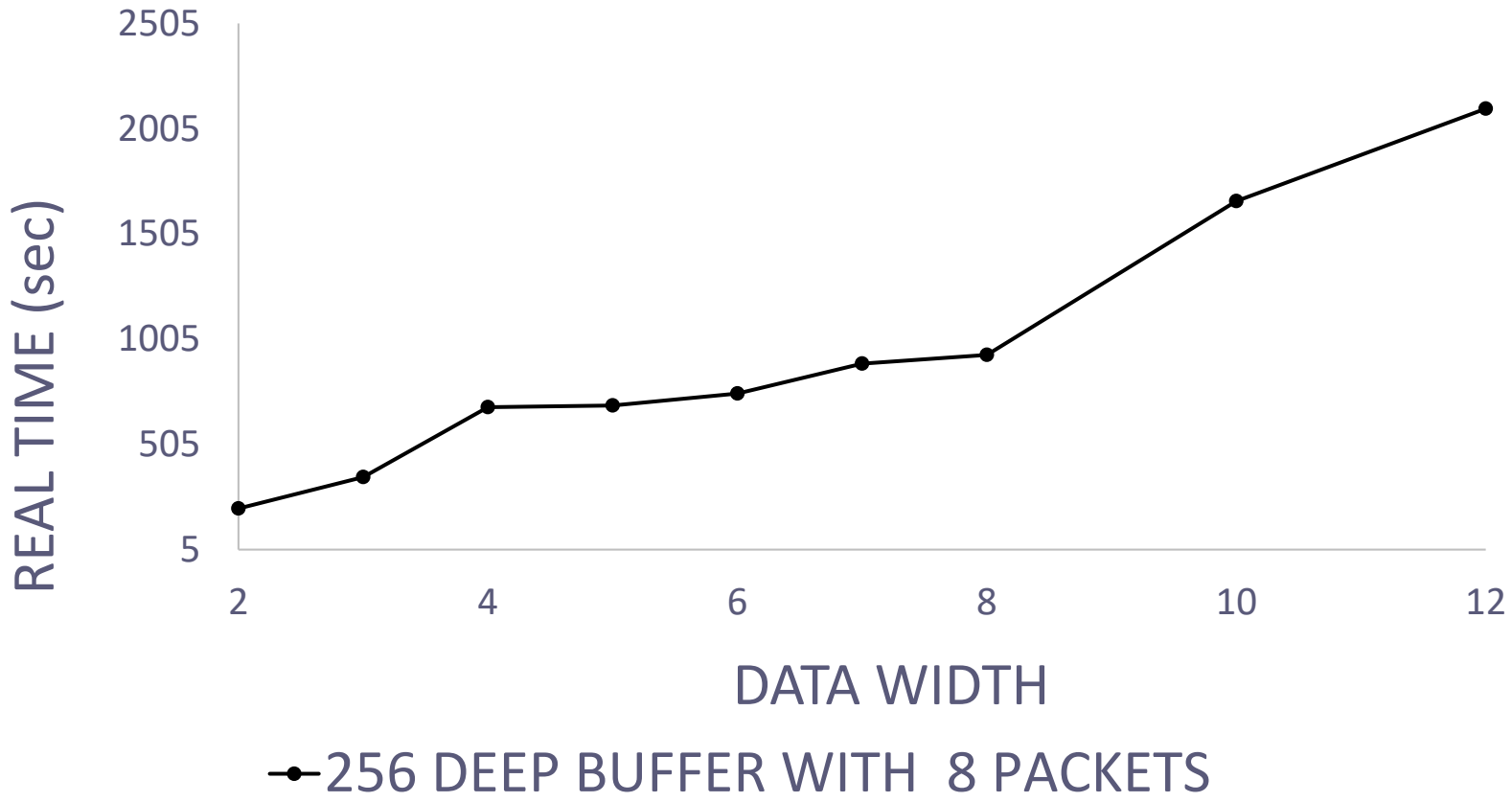


10^{1232} states!
~4096 flops

512-deep buffer
Up to 8-packets
Packet size: 1 bit

PROOF in 17 min

Increasing Packet Size



10^{7397} states!
~24,576 flops

256-deep buffer
Up to 8-packets
Packet size: 12 bit

PROOF in 35 min

Proof Engineering

Scalable formal verification
=
“Proof Engineering”

Invariants
Assume Guarantee

Decomposition

Structural Decomposition

Split wide vectors into groups of 8-bits
Creates more properties to prove
But each property itself is tractable

	status	depth	name (C)	engine	elapsed_time
4			packet_design.u_pkt_design_sva.all_packets_bits[0].index_7_0		
5	✔		packet_design.u_pkt_design_sva.all_packets_bits[1].index_15_8	e1	00:16:29
6	✔		packet_design.u_pkt_design_sva.all_packets_bits[1].index_23_16	e1	00:30:33
7	✔		packet_design.u_pkt_design_sva.all_packets_bits[1].index_31_24	e3	05:26:31
8			packet_design.u_pkt_design_sva.all_packets_bits[1].index_7_0		
9	✔		packet_design.u_pkt_design_sva.all_packets_bits[2].index_15_8	e3	01:55:23
10	✔		packet_design.u_pkt_design_sva.all_packets_bits[2].index_23_16	e3	05:39:03
11	✔		packet_design.u_pkt_design_sva.all_packets_bits[2].index_31_24	e3	05:56:47
12			packet_design.u_pkt_design_sva.all_packets_bits[2].index_7_0		

	status	depth	name (C)	engine	elapsed_time	type
1	⚠	46	packet_design.u_pkt_design_sva.all_packets_bits[2].index_7_0	b1	02:02:09	assert
2	✔		packet_design.u_pkt_design_sva.all_packets_bits[1].index_7_0	e1	00:48:24	assert
3	✔		packet_design.u_pkt_design_sva.all_packets_bits[0].index_7_0	e1	00:40:18	assert

	status	depth	name (C)	engine	elapsed_time	type
1	✔		packet_design.u_pkt_design_sva.all_packets_bits[2].index_7_0	e1	00:14:21	assert

$10^{10,000}$ states!
~32,768 flops

256-deep buffer
Up to 4-packets
Packet size: 32 bit

PROOF in 7 hours 54 min

Running out of compute power at this point!!

Scaling Compute Power

- On a 128 GB, 18-core machine
- For a design with **500K+ flops** ($10^{164,228}$ states)
  - 256-deep, up to 64 packets, each packet being 32-bit
- Bug hunting
  - Random reordering bug can be found in under 20 min of run time
- Exhaustive Proof obtained in under 3 hours

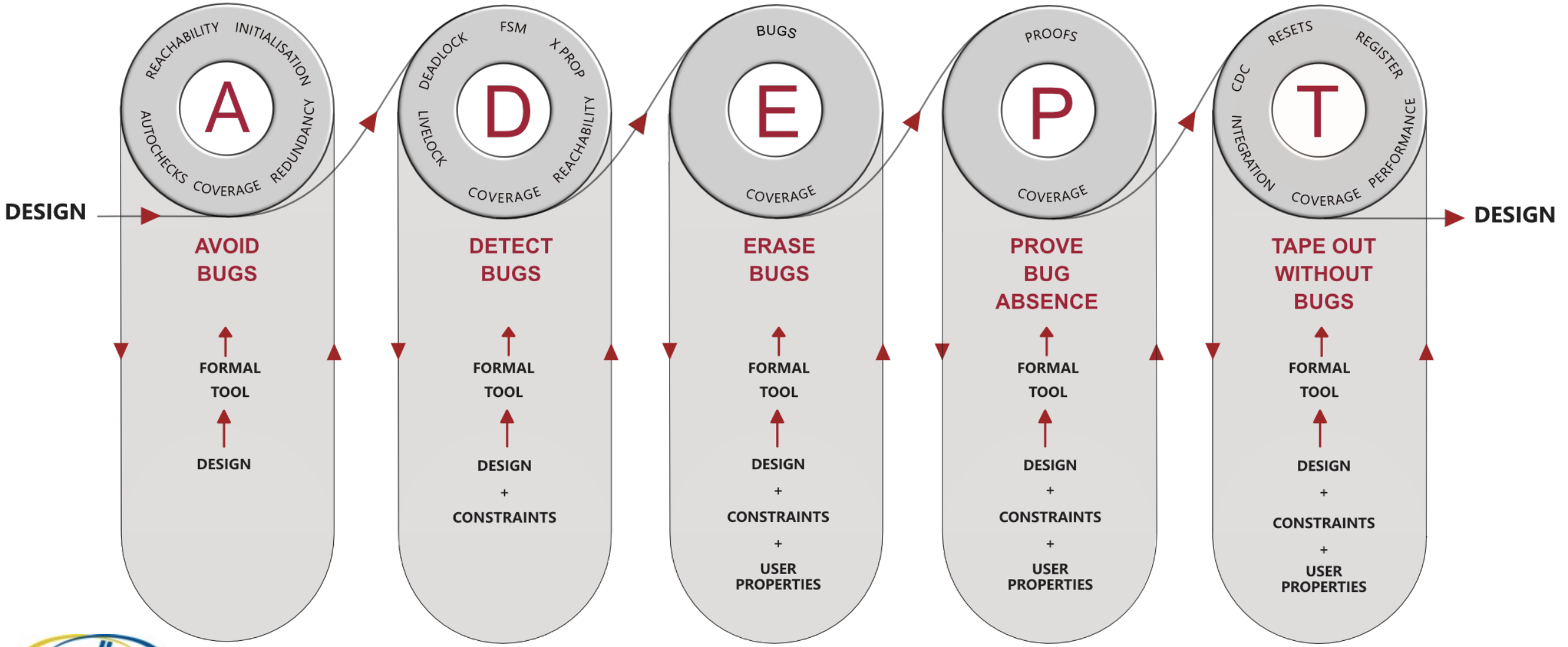
formal

92 sec to find corner case bugs!

Design flops	126+ million
Gates	449+ million
Property flops	7.9+ million
Check type	end-to-end
Compile time	35 minutes!
Cut-pointing	No
Black-boxing:	No

338 million flops, 1.1 billion gates, 100.2 seconds to find bugs!

ADEPT FV®


DESIGN
A
REACHABILITY
INITIALISATION
REDUNDANCY
COVERAGE
AUTOCHECKS
AVOID BUGS
FORMAL TOOL
DESIGN
D
FSM
X PROP
REACHABILITY
COVERAGE
LIVELOCK
DEADLOCK
DETECT BUGS
FORMAL TOOL
DESIGN + CONSTRAINTS
E
BUGS
COVERAGE
ERASE BUGS
FORMAL TOOL
DESIGN + CONSTRAINTS + USER PROPERTIES
P
PROOFS
COVERAGE
PROVE BUG ABSENCE
FORMAL TOOL
DESIGN + CONSTRAINTS + USER PROPERTIES
T
RESETS
REGISTER
PERFORMANCE
COVERAGE
INTEGRATION
CDC
TAPE OUT WITHOUT BUGS
FORMAL TOOL
DESIGN + CONSTRAINTS + USER PROPERTIES
DESIGN

Summary

- Verification of serial designs is a challenge
- What we need is: Methodology + Technology
- On a laptop we find:
  - bugs in designs with 141K flops in less than 40 min
  - proof in designs with 24K flops in less than 35 min
- On a server with 128GB RAM we find:
  - catch bugs in designs with 500k+ flops in less than 20 min
  - proof in 3 hours (256 deep, 64 packets, 32-bit packet size)
- There is no other verification paradigm quite like formal!