2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Data-Driven Verification: Driving the next wave of productivity improvements

Cadence Presenters

Larry Melling, Director Product Management

Chris Komar, Group Director Product Engineering

Sharon Rosenberg, Solutions Architect

Michael Young, Group Director Product Management

UltraSoC Presenter

Hanan Moller, Systems Architect

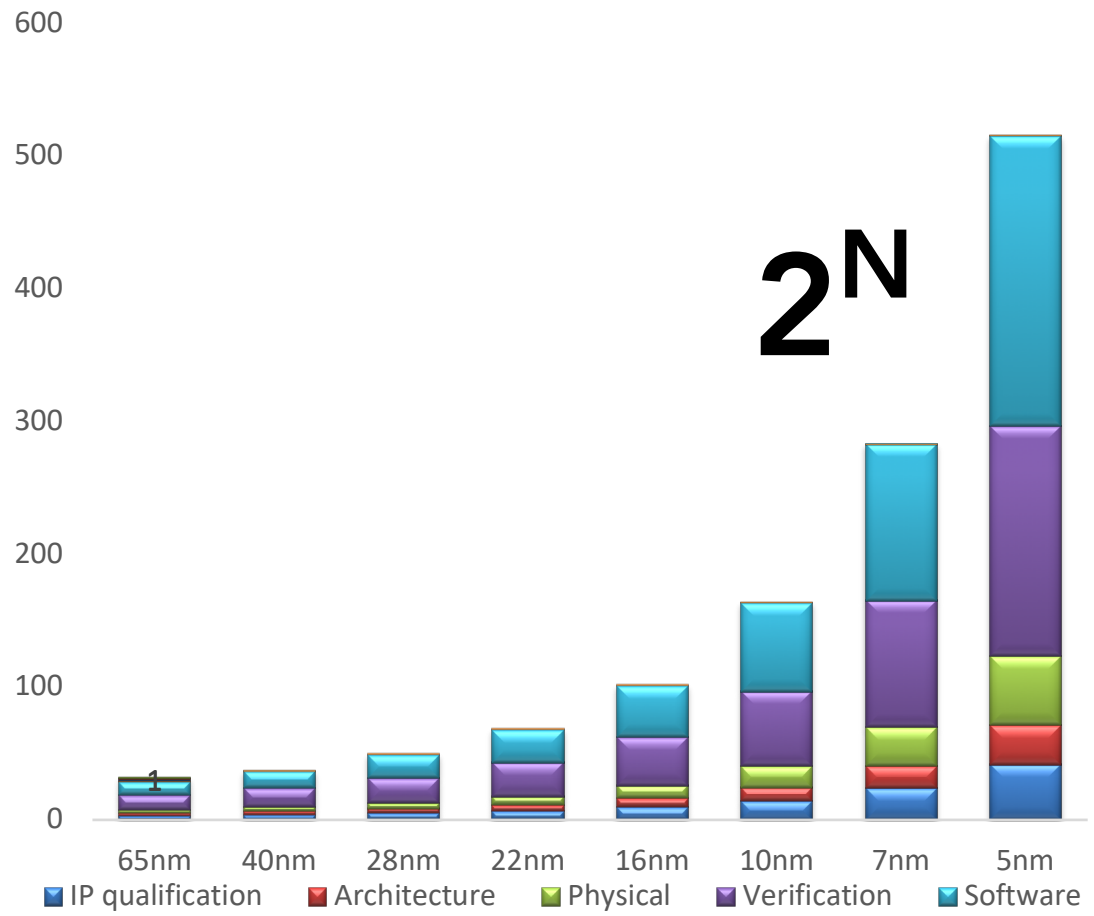
accellera SYSTEMS INITIATIVE

cadence

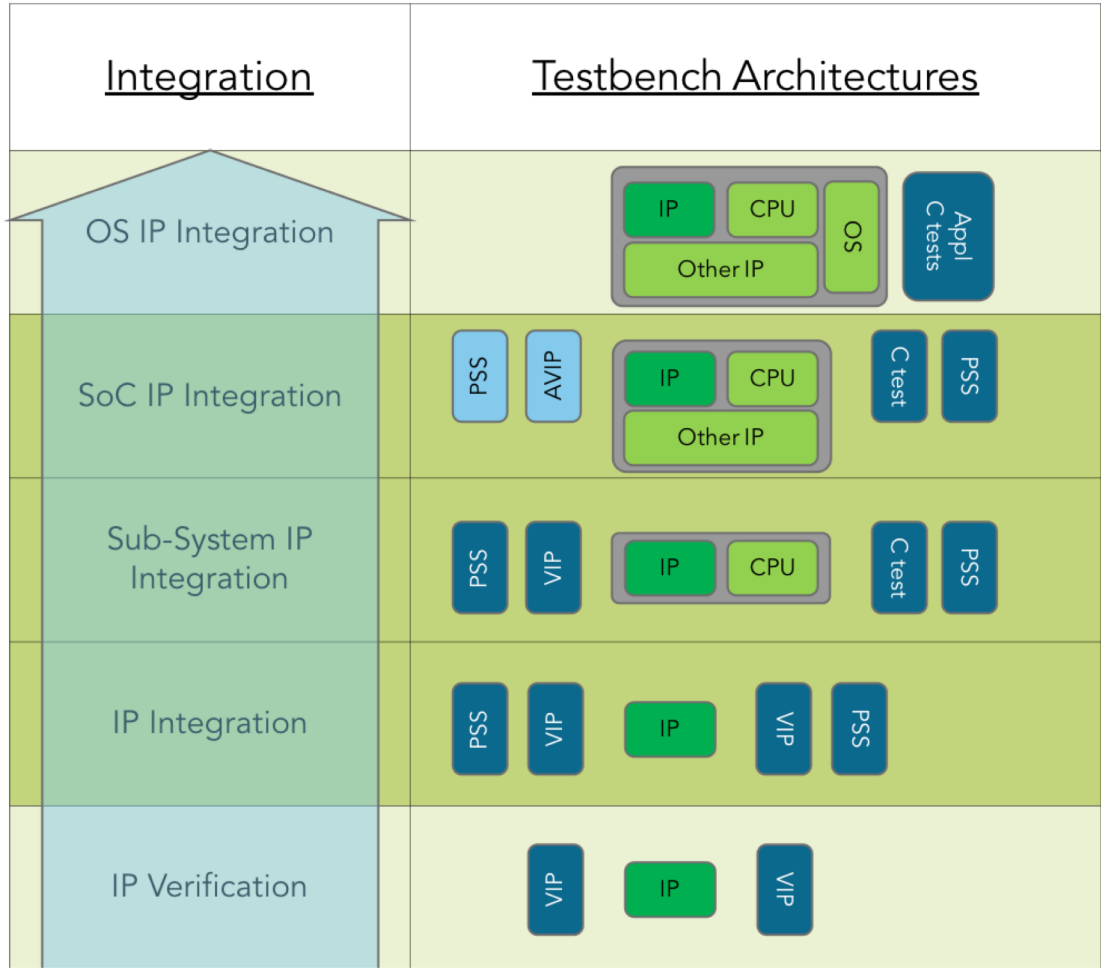
ultrasoc

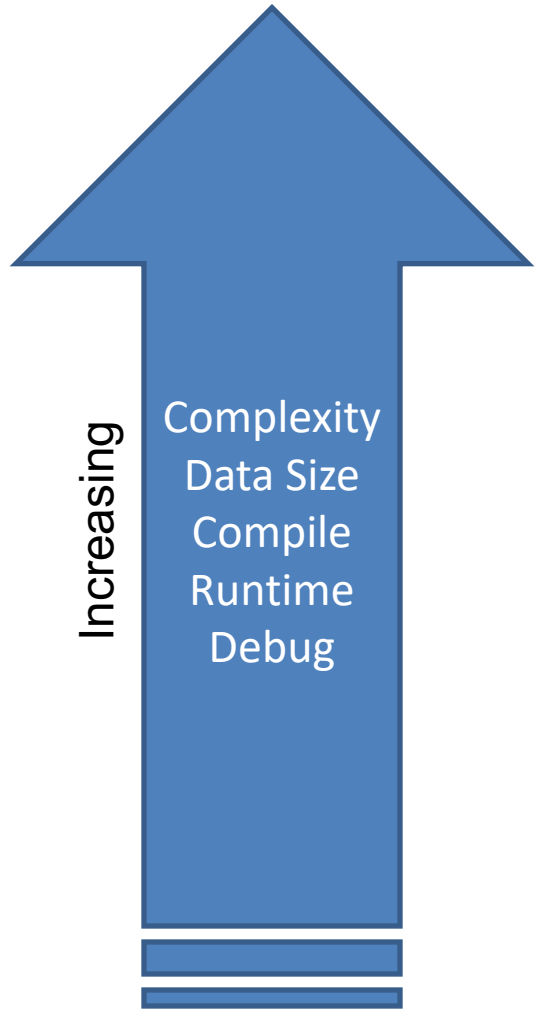
The Problem

- Verification cost growing exponentially with complexity
- Finite budget
- Finite resources
- Compromise quality/Increase risk



The Problem


Integration
Testbench Architectures
OS IP Integration
SoC IP Integration
Sub-System IP Integration
IP Integration
IP Verification
IP
CPU
OS
Other IP
Appl C tests
PSS
AVIP
C test
VIP


Increasing
Complexity
Data Size
Compile
Runtime
Debug

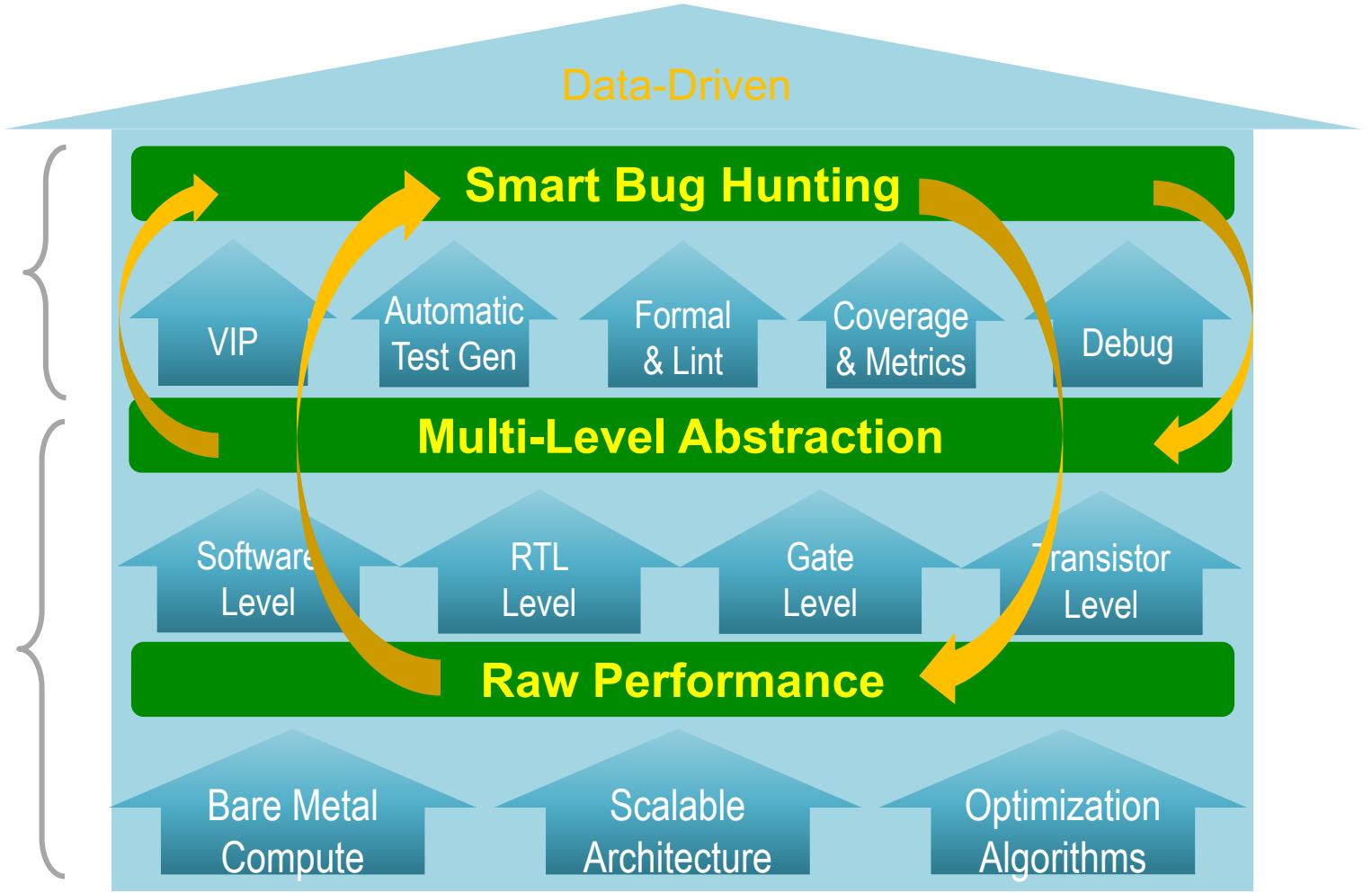
What is Data-Driven Verification?

Use-case-based	• Define legal operations • Workload matters: must represent real operation
Data Collection	• Non-intrusive data collection • Use the right execution platform
Analysis	• Correlate, filter, learn, predict • Anomaly Detection
Goal-based	• Verification throughput • Smarter bug hunting

VERIFICATION THROUGHPUT

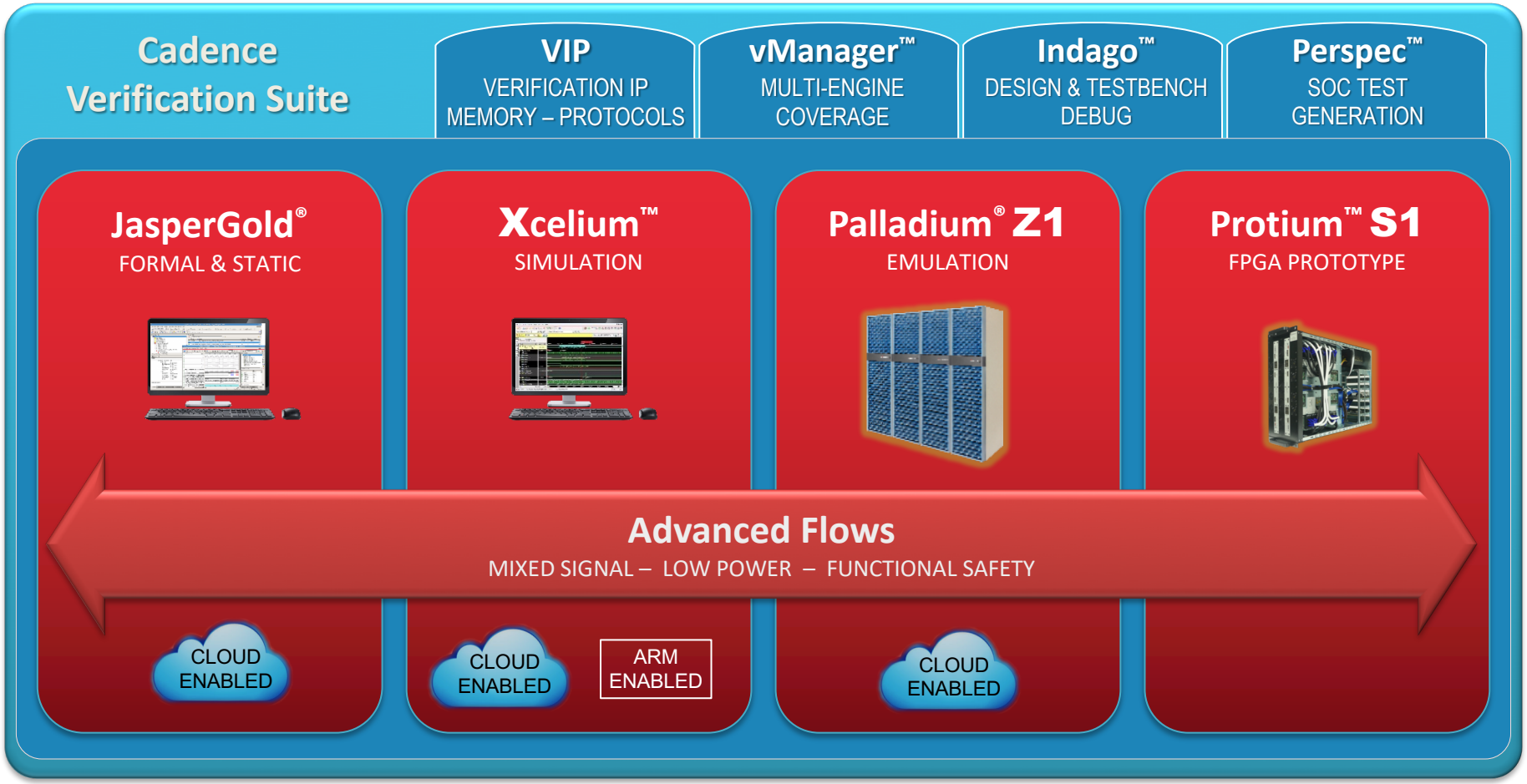
Bugs per $ per day

Cycles per $ per day



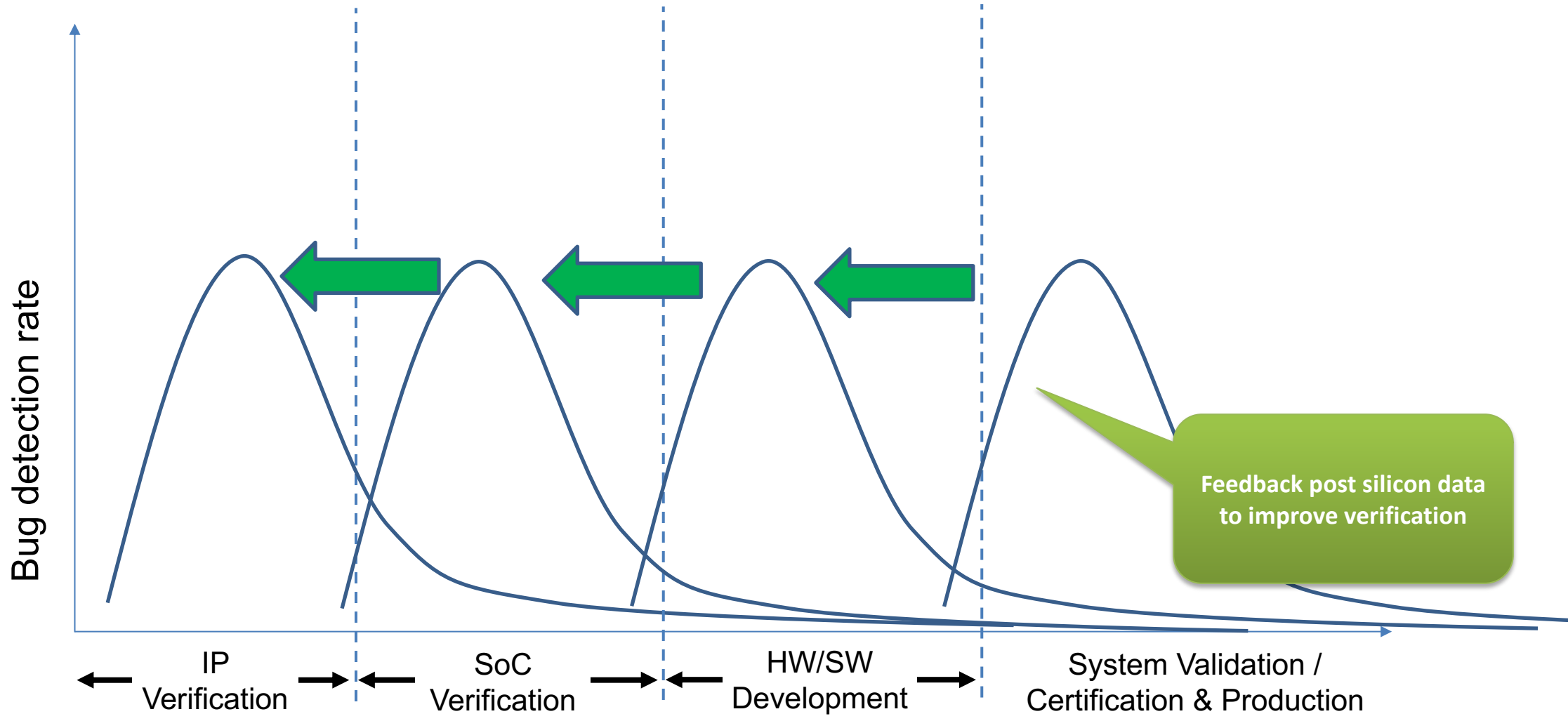
2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Cadence **Verification Suite**



accellera SYSTEMS INITIATIVE

Bug detection still not as early as possible



2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

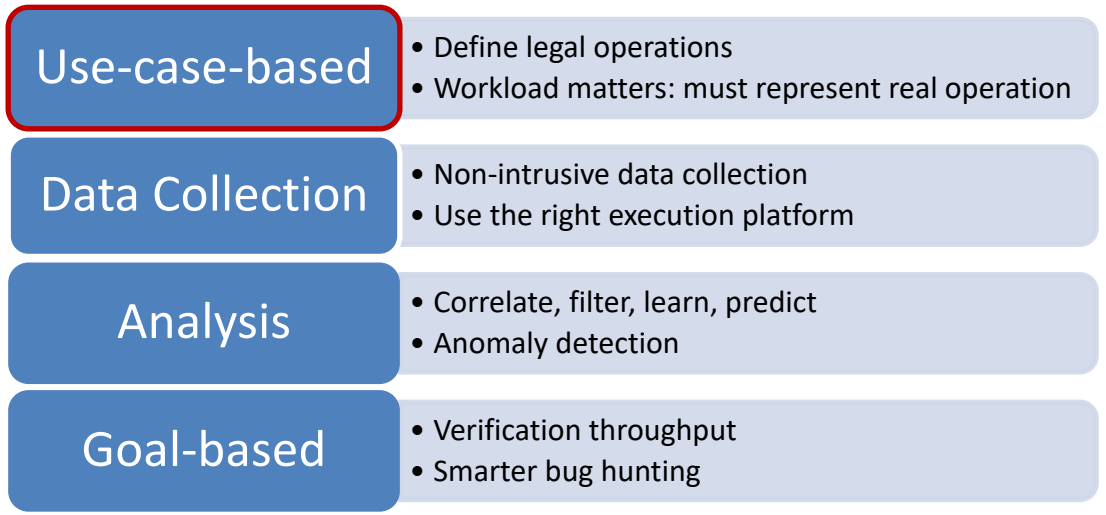
Sharon Rosenberg, Solutions Solutions Architect, Cadence

PORTABLE STIMULUS: USE-CASE-BASED VERIFICATION

accellera SYSTEMS INITIATIVE

Portable Stimuli Standard (PSS)

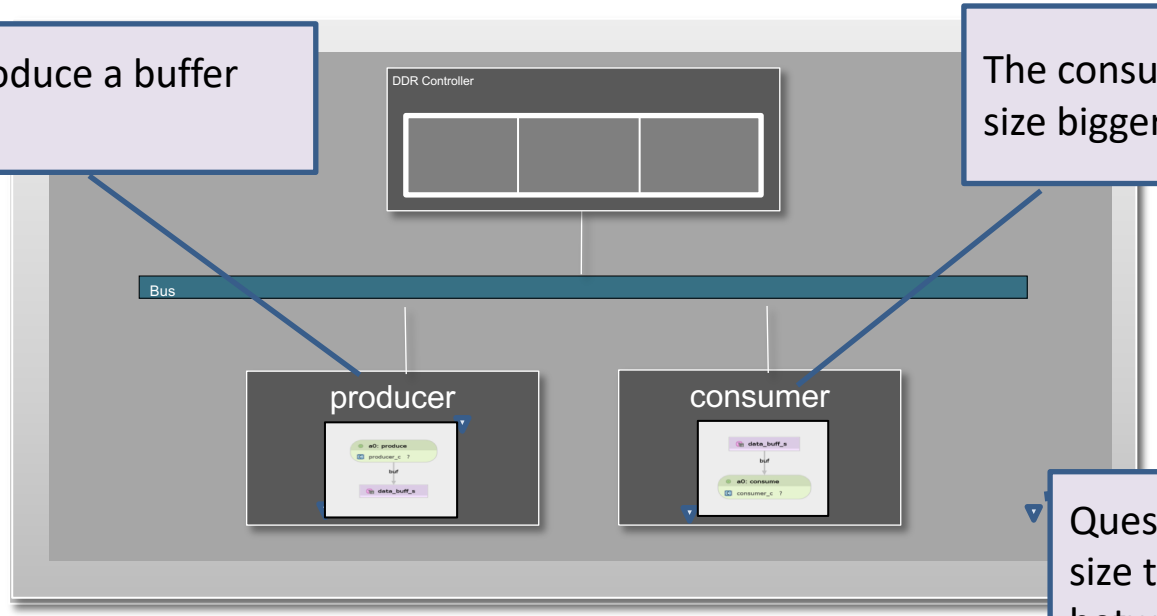
- Behavioral standard language to express scenarios
  - Parallelism with fork and join
  - Control flow with loops, conditionals
  - Data path via memory buffers and streams
- Powerful built-in system-specific semantic for
  - Resource availability and distribution
  - Configuration, and operation modes
- Codified in two equally powerful input formats:
  - C++ library – appeals to C++ users
  - PSS – a Domain Specific Language (DSL) – easier to read and better error messages
- Standard is defined by PSWG in Accellera



Capturing Legal Behaviors

Capture test intent, analyze legal paths, generate tests randomizing options

The producer can produce a buffer size smaller than 15



The consumer can consume buffer of size bigger than 10 bytes

Question: What will be a proper data size to enable communication between them?

Answer: 11..14

System assumptions – IPs can be programmed to communicate if they connected to the same memory, and no restriction prevents that communication (e.g. data size or data kind mismatch)

Capturing Legality Rules

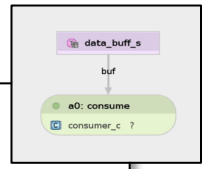
Capture test intent, analyze legal paths, generate tests randomizing options

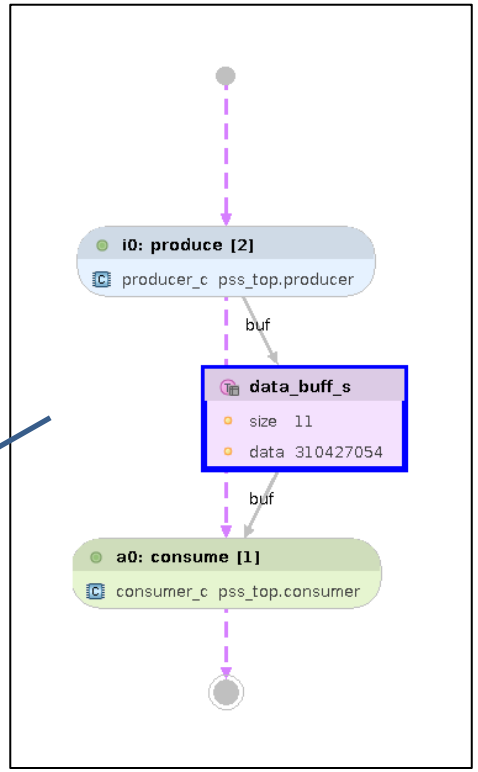
```
buffer data_buff_s {
  rand uint[1..20] size;
  rand uint data;
};
```

PSS allows capturing the dependencies in a special flow-object struct

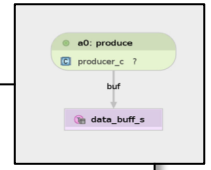
This association of activity to legality rule is a revolution!

```
component consumer_c {
    action consume {
        input data_buff_s buf;
        constraint buf.size > 10;
    };
};
```






```
component producer_c {
    action produce {
        output data_buff_s buf;
        constraint buf.size < 15;
    };
};
```



A constraint solver solved the scenarios to achieve a legal programming

Each sub-system model captures it's own dependencies according to it's specifications

Capturing Legality Rules

Capture test intent, analyze legal paths, generate tests randomizing options

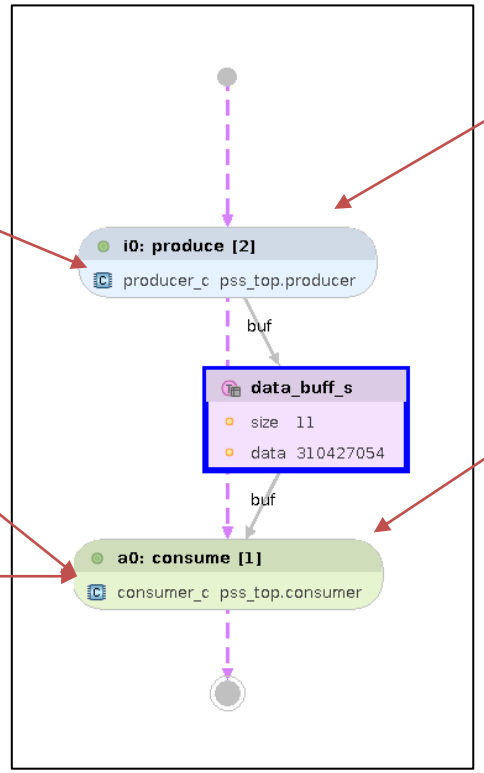
```
buffer data_buff_s {
  rand uint[1..20] size;
  rand uint data;
};
```

config

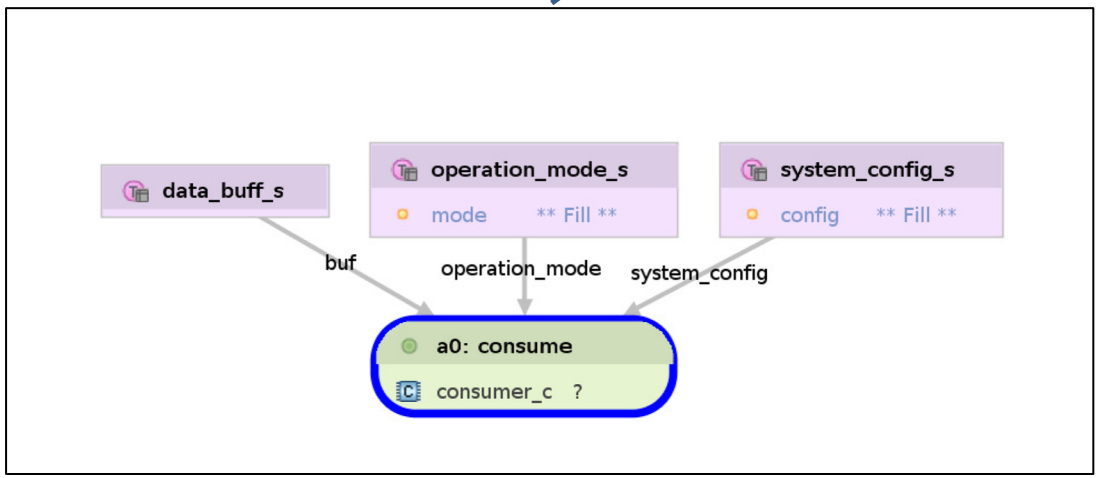
resource pool

Power-domain B

Power-domain A

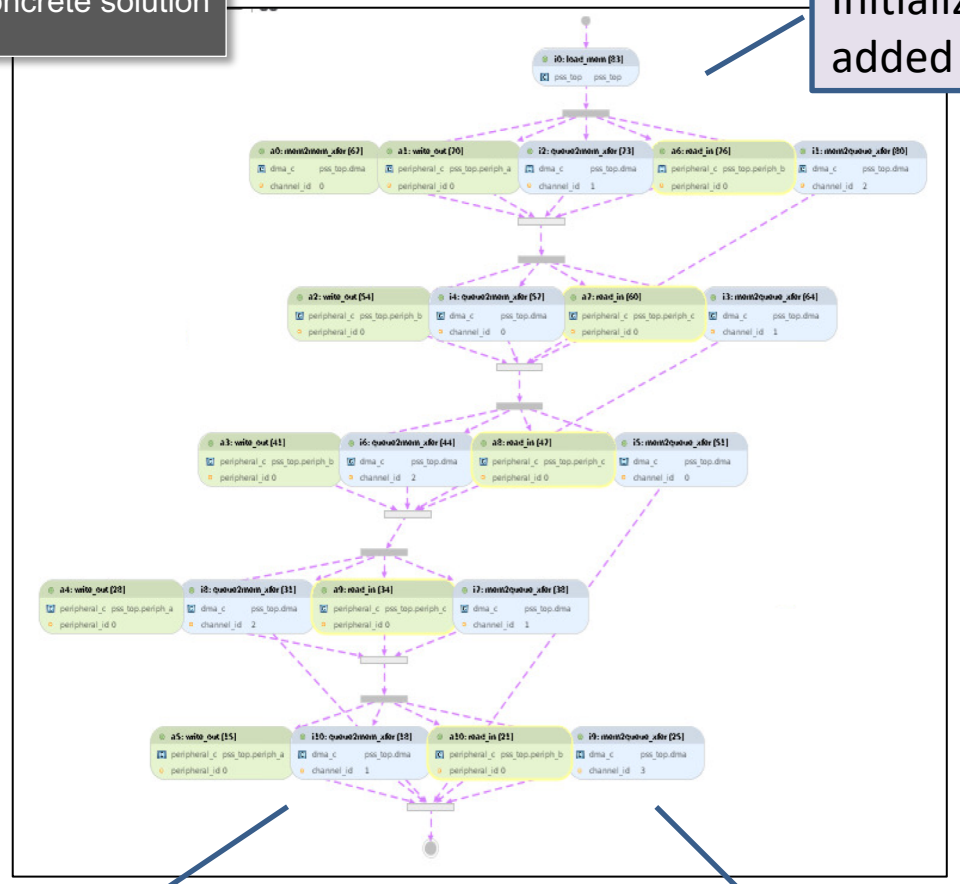


Consume in all legal configuration and operation modes to automatically fill your coverage goals!



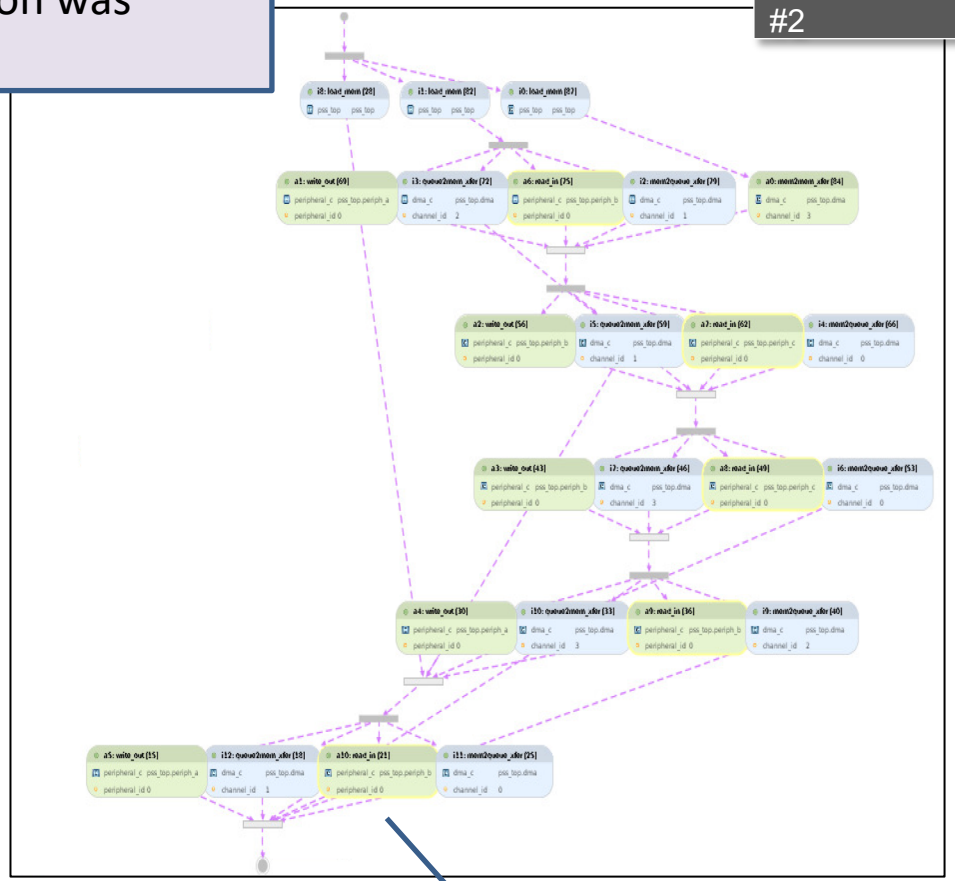
Generating Test Scenarios

Concrete solution #1



Initialization action was added

Concrete solution #2



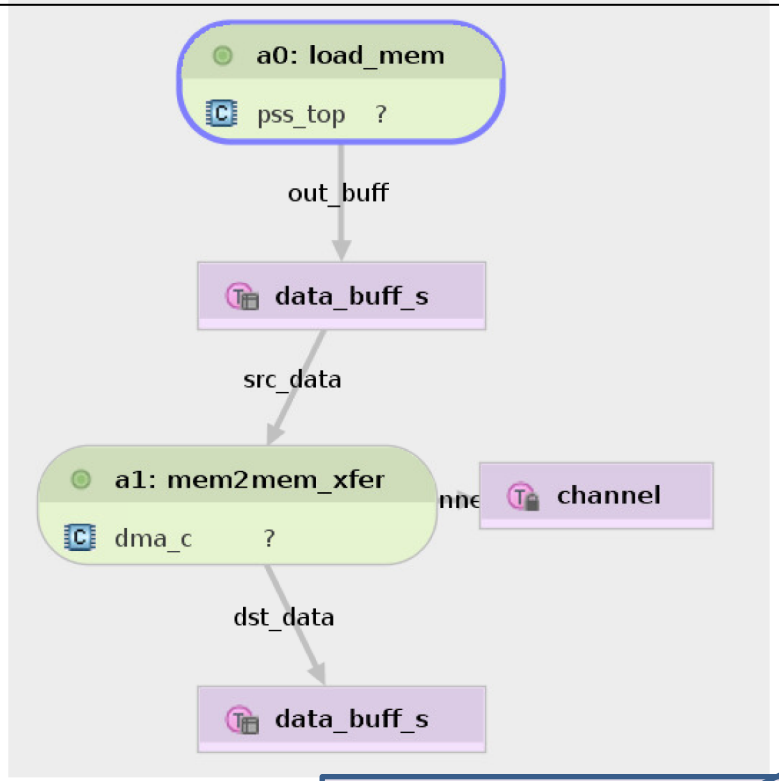
DMA actions were added to serve the read and write requests

Legal peripherals and DMA channels were assigned

As many as desired solutions with different timing can be randomized to serve the same original request

Test can be Generated to Run on Any Platform

My first test: load the memory with data and use the DMA to copy it to a different location




```
// my first test
int main_core3()
{
    tb_initial_mem(0x5000,my_data);
    signal_core(1);
    done(1);
}

int main_core1()
{
    wait_for_core(1);
    dma_program(2, 0x5000, 0x700, 20);
    dma_start(2);
    dma_wait_for_done(2);
    done(1);
}
```

This might be a UVM virtual sequence creating the same test

User firmware code

- Tool generated code
- Synchronizations, loops, fork and joins, all are done by the PSS tool

PSS Impact on Stimulus

Existing stimulus

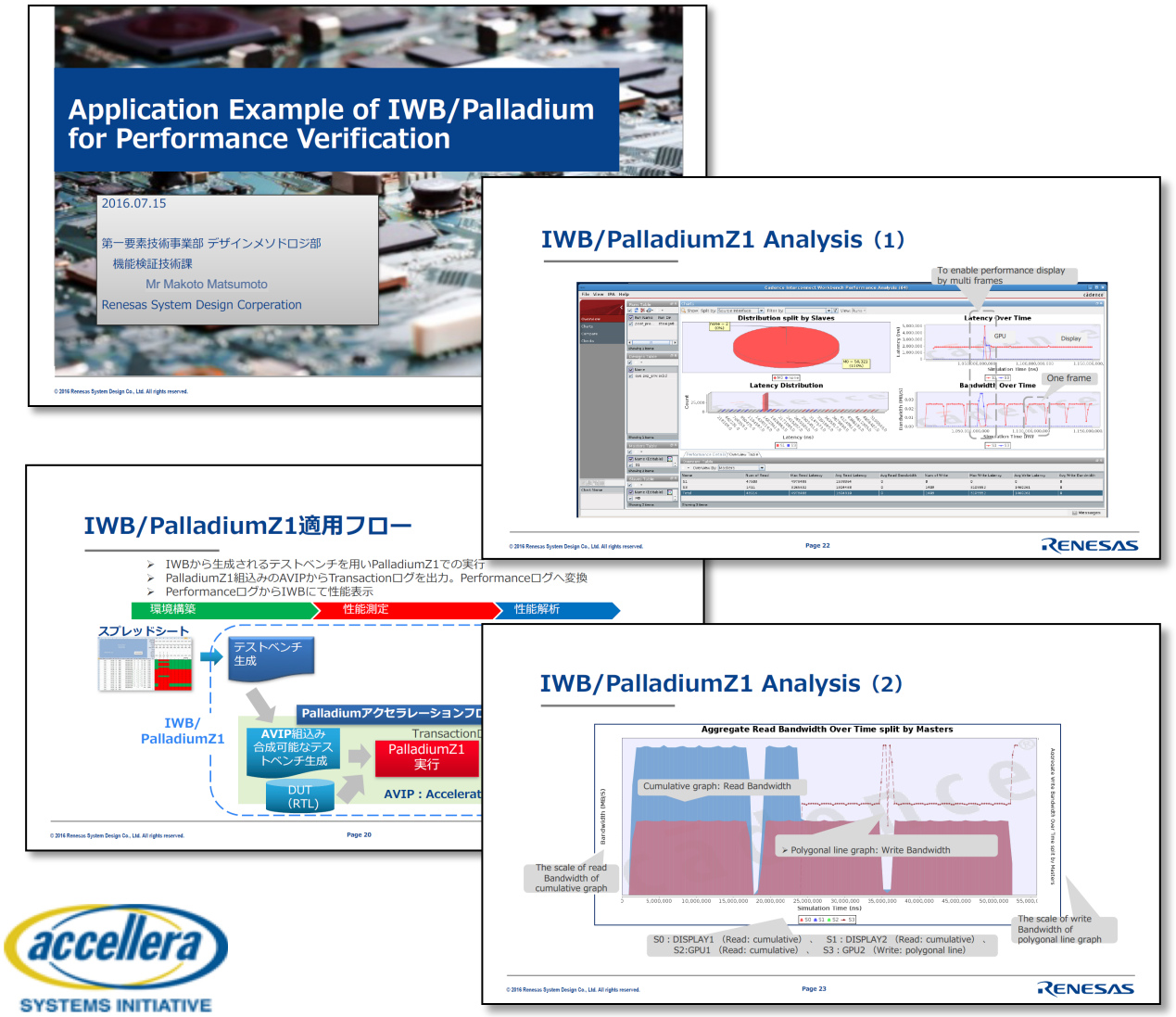
Post-silicon	• Generally OS based tests. Long test consume valuable r • Longer debug time • Failures are difficult to bring to emulation or simulation fo
Pre-silicon SoC	• Simple directed feature tests • Difficult to manually create complex scenarios • Long run time for complex scenarios
Pre-silicon IP	• Excellent UVM based constrained random testbench • IP initialization sequences not easily portable to FW or p • IP level tests lack system context

Excerpt from AMD DVCON presentation

Stimulus with PSS

Post-silicon	• Smaller deterministic bare-metal tests • Compose complex scenarios • Easily bring debug to Emulation • Generate large set of tests for regression
Pre-silicon SoC	• Describe test intent with PSS • Automation helps with complex scenario composition • Reuse tests post-silicon
Pre-silicon IP	• Reuse SoC scenarios • Export initialization sequences to firmware and post-silicon • Export IP specific scenarios to SoC

Renesas Performance Verification with Pespec Generated Use-cases

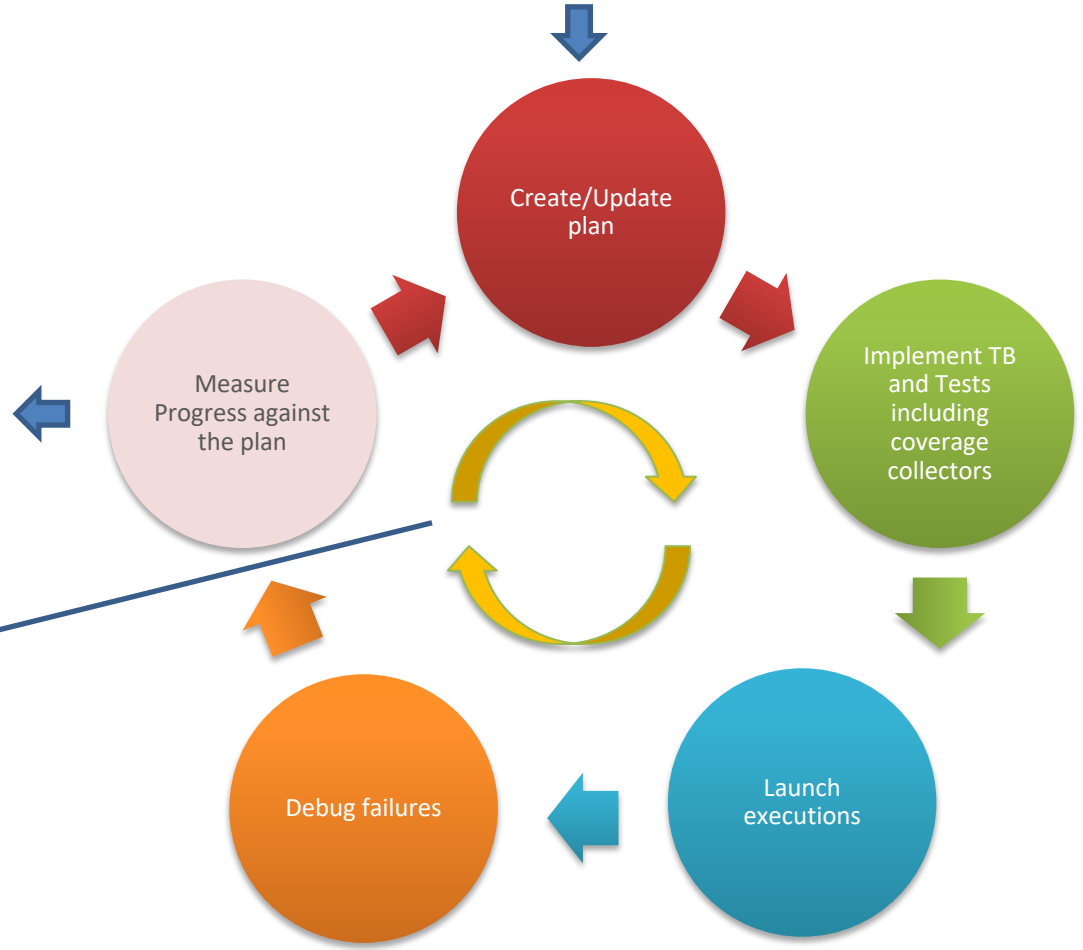


- Leading industrial and automotive MCUs
  - Number of integrated IPs is increasing
  - Switched interconnect
  - Configuration has big impact on performance
- Interconnect Workbench performance analysis
  - Early performance characterization
  - Interconnect tuning to optimize performance
  - Use case performance validation
- Palladium Z1 with Perspec use cases
  - Bring-up the entire design and software
  - Perspec generating use case tests
  - Reduce from 50 hour simulations to 12 minutes

Why PSS is Great for Data-Driven?

- Captures the verification flows
  - Allows focusing on intent
  - Abstracts away implementation details
- Automated traffic to close the coverage gap
  - UVM gives a fresh stream per seed but virtual sequences are highly directed
  - Accomplishing a goal may require coming up with different timing or test topology
  - The power of the PSS random schedule capability
- PSS captures the legality rules
  - “Don’t move the furniture and don’t clean the dog”
- Portability
  - Coverage filling task may cross platform borders
    - May not have enough cycles to be closed in a single platform
- PSS solve the entire scenario in one time
  - Can leverage data before running the simulation

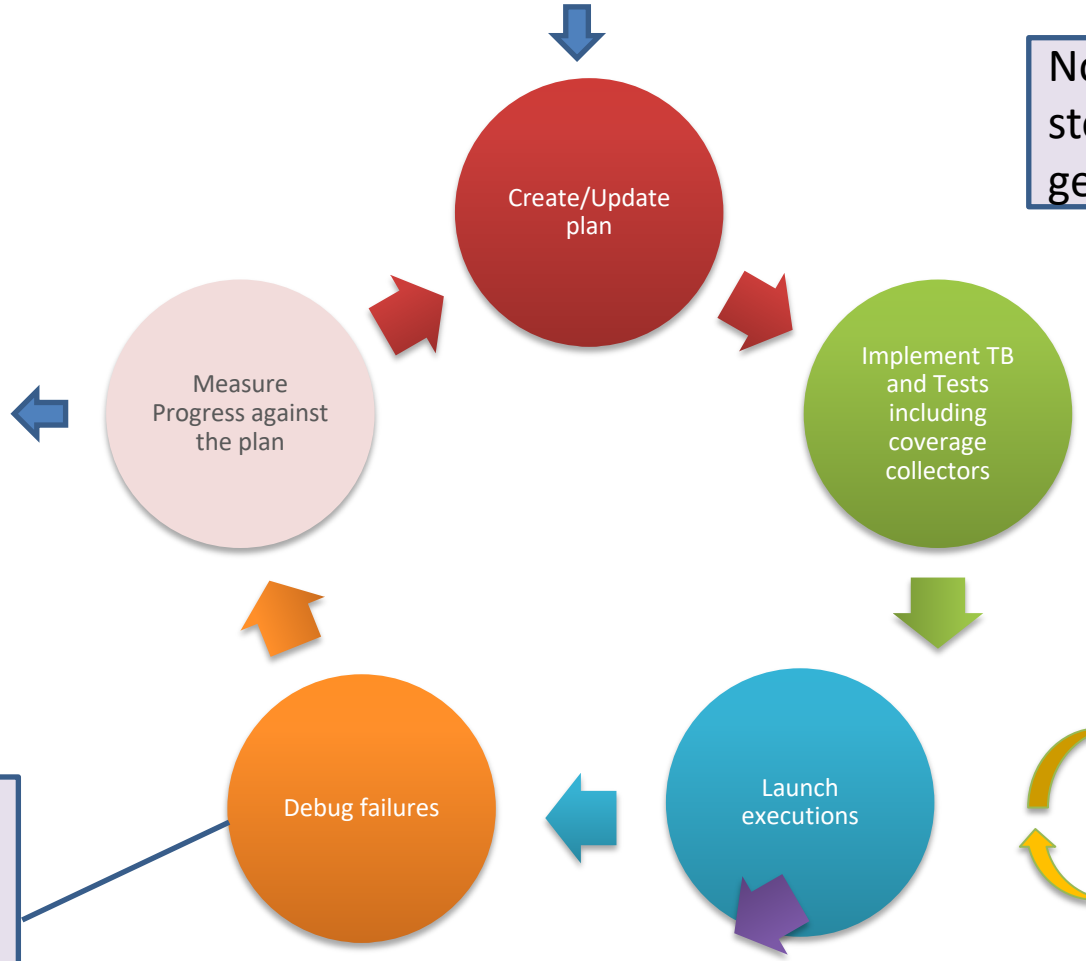
Traditional MDV Flow



Need to re-do the work in case of TB changes or a different simulator release

Coverage can be measured only after simulation is done
Manual work to achieve 100% coverage or discovering a value to be unreachable

Data-Driven with with vManager and Perspec



Noticeably, Perspec adds a step of automatic test generation

Generate tests and Review gen-time coverage

Implications:

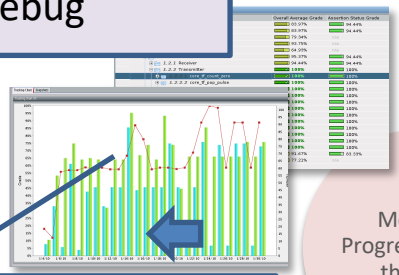
- Reduced number of machines and farm size
- Less human efforts for coverage review and test creation
- Shorter cycles to meet coverage goals and project deadlines

But there is much more that is added on all MDV steps!

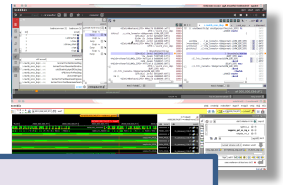
New test topologies are automatically produced Coverage maximization is done before running the regression per user criteria vManager decides if a test is contributing for a plan or redundant

Data Driven with Perspec and vManager

Coverage on use-cases
Reachability analysis and debug



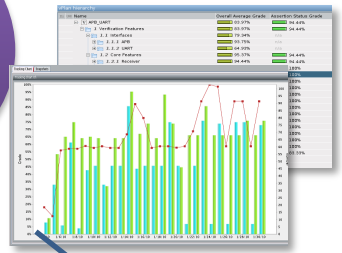
Runtime collection from all platforms
Combine HW and SW coverage items



Abstract debug and use case review

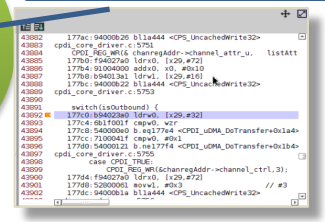
Progress management in terms of meaningful test properties

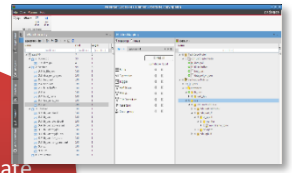
Extra step of test generation and gen-time coverage review
Use case visualization to approve the generated scenario



Regression level maximization

Use case coverage on the abstracted model
- Easy to implement
- Direct mapping to high level plan
- Contains both SW and HW
- Can be exported to all platforms





Create/Update plan

Implement TB And coverage model

Generate tests and Review gen-time coverage

Launch executions

Debug failures

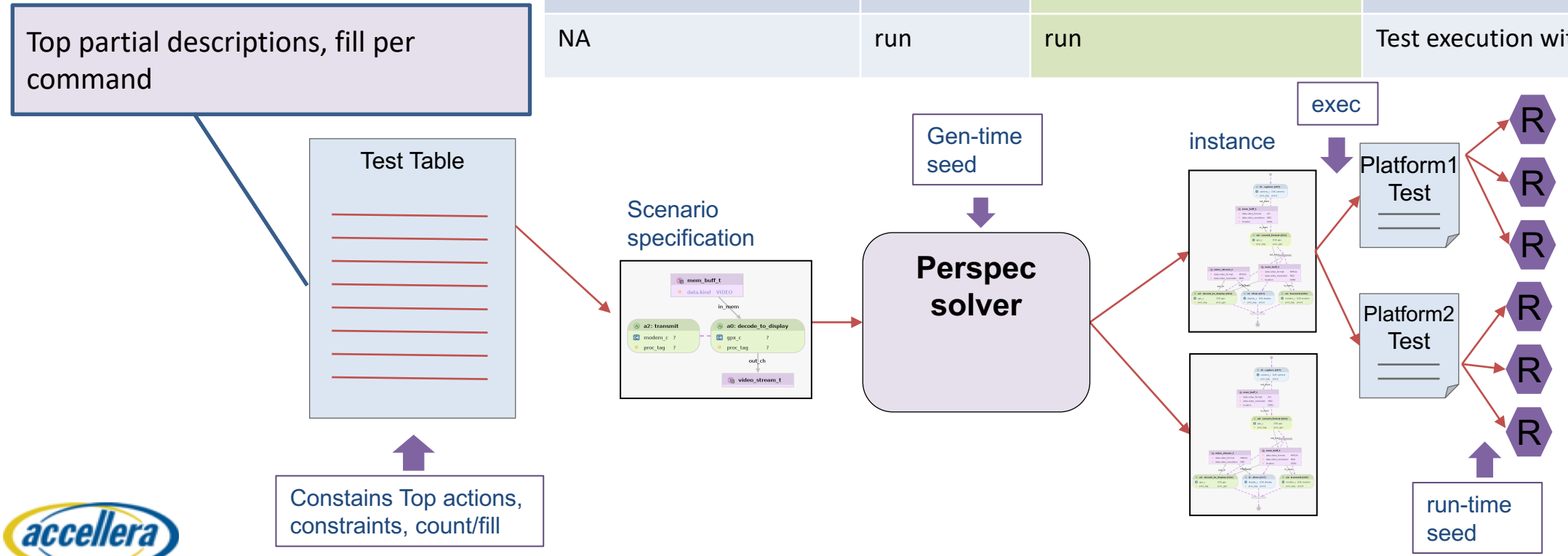
Measure Progress against the plan

Perspec and vManager Revolution

- Builds on the vManager flows capability
  - Allows running a multi-steps session
  - Steps can run in parallel to each other (e.g. start launching tests as soon as they are ready)
- Simplified integration scripts using the following user-defined scripts
  - ps_gen_script – a script for generating a full perspec regression
  - ps_exec_script – a script for running a single test
  - config file – lists step names, top-directories and the two scripts above
- Enhanced regression control with test tables
  - The tests to be generated and executed are coming from Perspec test tables (and not VSIF)
    - Include multiple top actions, constraint settings, counts and fill capabilities for each
  - The execution scripts are the gen and exec scripts provided above
- The MDV flow does not force usage of test tables
  - Users can use home grown scripts
  - More automation can be provided on top of test tables

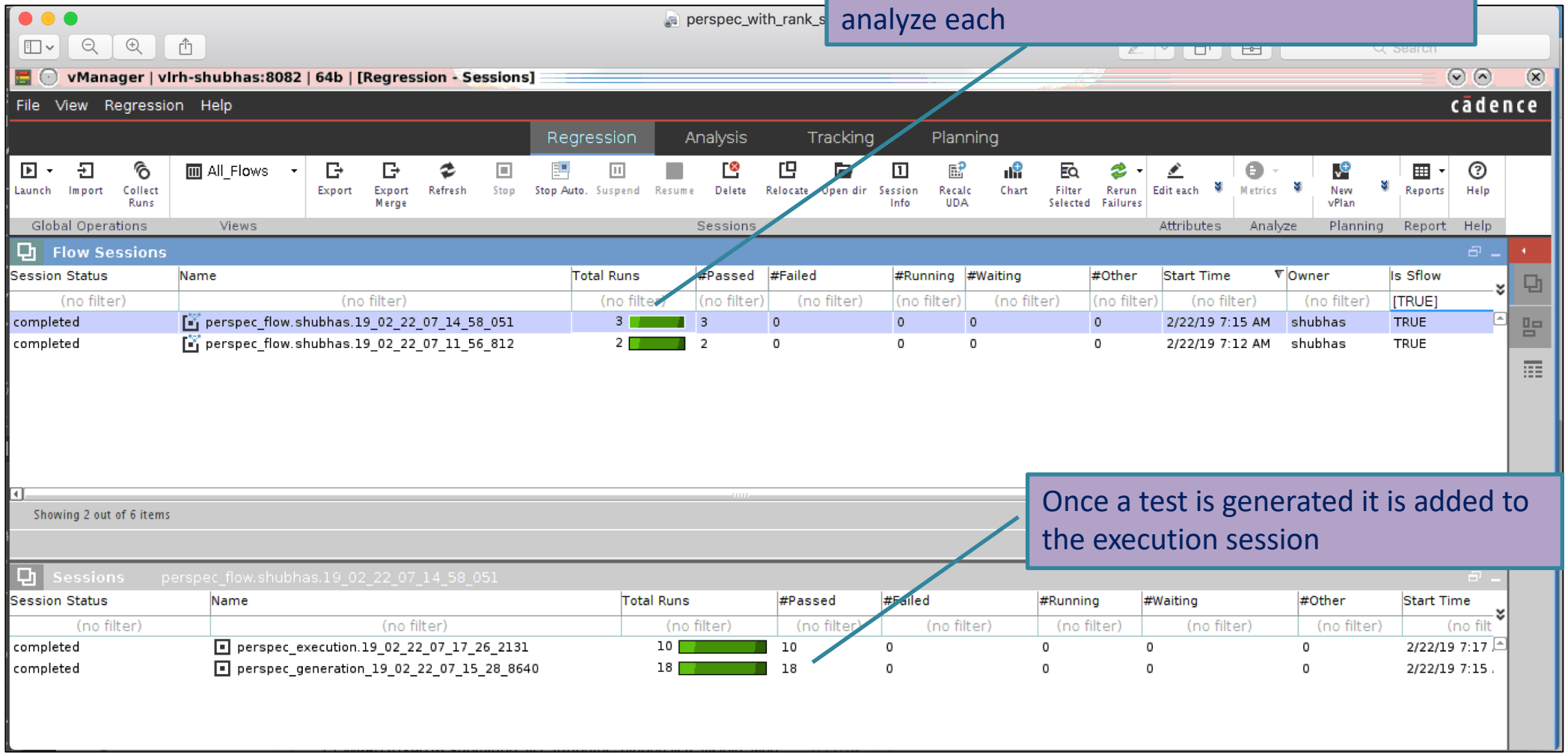
Resolving vManager/Perspec terminology review

Perspec	Vmanager	Integrated terminology	Definition
Scenario specification	NA	Scenario specification	Pure intent, partial specification
Scenario instance	NA	Scenario instance	Fully statically solved scenario
test	test	test	Code representation of scenario instance
NA	run	run	Test execution with a seed



Perspec-vManager Solution

Perspec flow consists of two steps:
Test generation & test execution with ability to analyze each



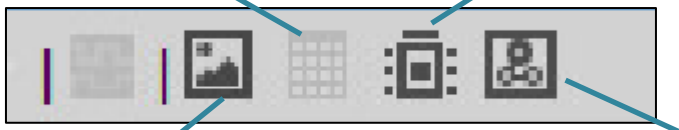
Once a test is generated it is added to the execution session

Perspec-vManager Solution

eration Steps

Regression recipe

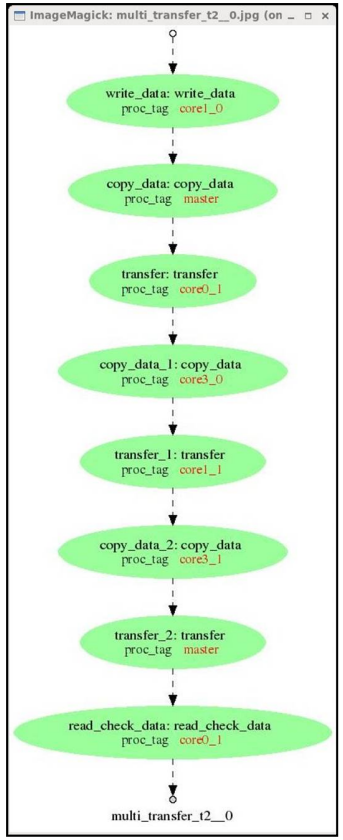
Debug contradictions



Link to the UML diagram of the test

View the UML of the entire regression

Test generation results (by top action)



Runs | TESTS

Index	Name	Generation Rank	Status	Duration (sec.)	Top Files	Start Time
(no filte	(no filter)	(no filter)	(no filter)	(no filter)	(no filter)	(no filter)
1	/multi_transfer_t1	contribute	passed	2	n/a	2/22/19 7:15 AM
2	/multi_transfer_t1	contribute	passed	1	n/a	2/22/19 7:15 AM
3	/multi_transfer_t1	redundant	passed	2	n/a	2/22/19 7:15 AM
4	/multi_transfer_t1	contribute	passed	1	n/a	
5	/multi_transfer_t1	contribute	passed	1	n/a	
6	/multi_transfer_t1	contribute	passed	1	n/a	
7	/multi_transfer_t2	redundant	passed	2	n/a	
8	/multi_transfer_t2	contribute	passed	1	n/a	
9	/multi_transfer_t2	contribute	passed	2	n/a	
10	/multi_transfer_t2	redundant	passed	2	n/a	2/22/19 7:16 AM
11	/multi_transfer_t2	contribute	passed	1	n/a	2/22/19 7:16 AM
12	/multi_transfer_t2	redundant	passed	1	n/a	2/22/19 7:16 AM
13	/multi_transfer_t3	redundant	passed		n/a	2/22/19 7:16 AM
14	/multi_transfer_t3	redundant	passed	2	n/a	2/22/19 7:16 AM
15	/multi_transfer_t3	redundant	passed	1	n/a	2/22/19 7:16 AM
16	/multi_transfer_t3	redundant	passed	1	n/a	2/22/19 7:16 AM
17	/multi_transfer_t3	contribute	passed	1	n/a	2/22/19 7:16 AM
18	/multi_transfer_t3	contribute	passed	2	n/a	2/22/19 7:16 AM

Status: **passed**

Showing 18 items

Upon generation the scenario contributed coverage is evaluated against the verification plan or specific plan section

Perspec-vManager Solution

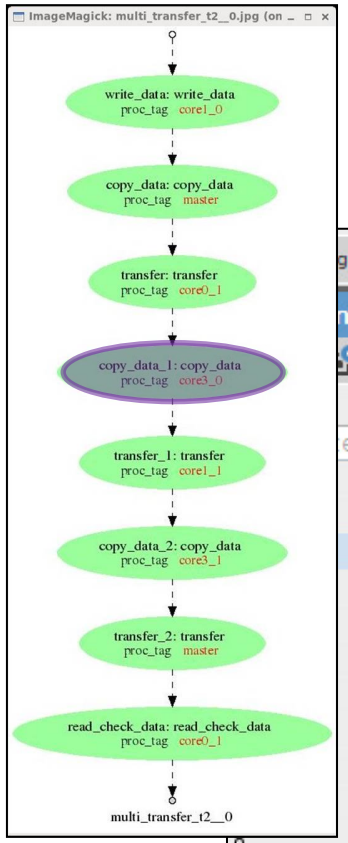
Execution Step

Regression recipe (test table)

Debug execution with waveform, smart log, and activity diagram

View the UML of the entire regression

Link to the UML diagram of the test. Reflects the execution progress



	Name	Status	Duration (sec.)	Top Files	Start Time
	(no filter)	(no filter)	(no filter)	(no filter)	(no filter)
	/cdn_uart_apbe_tests/data_poll_vir_seq	passed	18	/grid/avs/install/incisiv/10.2/lates...	1/4/11 11:22 PM
	/cdn_uart_apbe_tests/data_poll_vir_seq	passed	18	/grid/avs/install/incisiv/10.2/lates...	1/4/11 11:21 PM
	/cdn_uart_apbe_tests/test_uart	failed	17	/grid/avs/install/incisiv/10.2/lates...	1/4/11 11:21 PM
	/cdn_uart_apbe_tests/test_uart	failed	19	/grid/avs/install/incisiv/10.2/lates...	1/4/11 11:21 PM
	/cdn_uart_apbe_tests/fill_tx_buffer	passed	18	/grid/avs/install/incisiv/10.2/lates...	1/4/11 11:20 PM
	/cdn_uart_apbe_tests/fill_tx_buffer	passed	25	/grid/avs/install/incisiv/10.2/lates...	1/4/11 11:20 PM
	/cdn_uart_apbe_tests/fill_tx_buffer	passed	18	/grid/avs/install/incisiv/10.2/lates...	1/4/11 11:19 PM
	/cdn_uart_apbe_tests/fill_tx_buffer	passed	22	/grid/avs/install/incisiv/10.2/lates...	1/4/11 11:19 PM
	/cdn_uart_apbe_tests/data_poll	passed	22	/grid/avs/install/incisiv/10.2/lates...	1/4/11 11:18 PM
	/cdn_uart_apbe_tests/data_poll	passed	24	/grid/avs/install/incisiv/10.2/lates...	1/4/11 11:18 PM
	/cdn_uart_apbe_tests/data_poll	passed	23	/grid/avs/install/incisiv/10.2/lates...	1/4/11 11:17 PM
7	/cdn_uart_apbe_tests/data_poll	passed	18	/grid/avs/install/incisiv/10.2/lates...	1/4/11 11:17 PM
6	/cdn_uart_apbe_tests/data_poll	passed	20	/grid/avs/install/incisiv/10.2/lates...	1/4/11 11:17 PM
5	/cdn_uart_apbe_tests/data_poll	passed	19	/grid/avs/install/incisiv/10.2/lates...	1/4/11 11:16 PM

Execution runs

PSS and Data Driven

- Use-case capture essential for Data driven flows
  - Capturing information and legality rules
  - Revolution in test generation automation
  - Can be applied to any execution platform
- Capabilities exist today
  - Used by multiple users world-wide for both sub and full systems
  - Applications include workloads for performance, power, and coherency testing
- Can feed a data-driven cognitive machine for further analysis

Thank you!

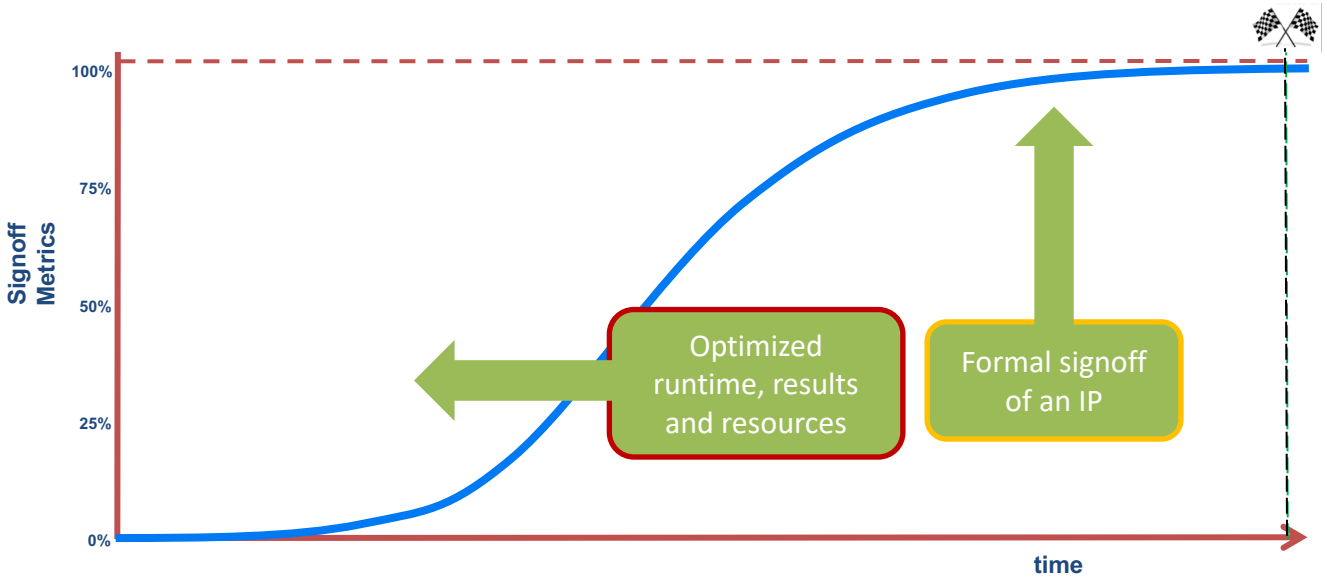
2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Chris Komar, Product Engineering Group Director, Cadence

DATA-DRIVEN FORMAL VERIFICATION

accellera SYSTEMS INITIATIVE

Data to Drive…



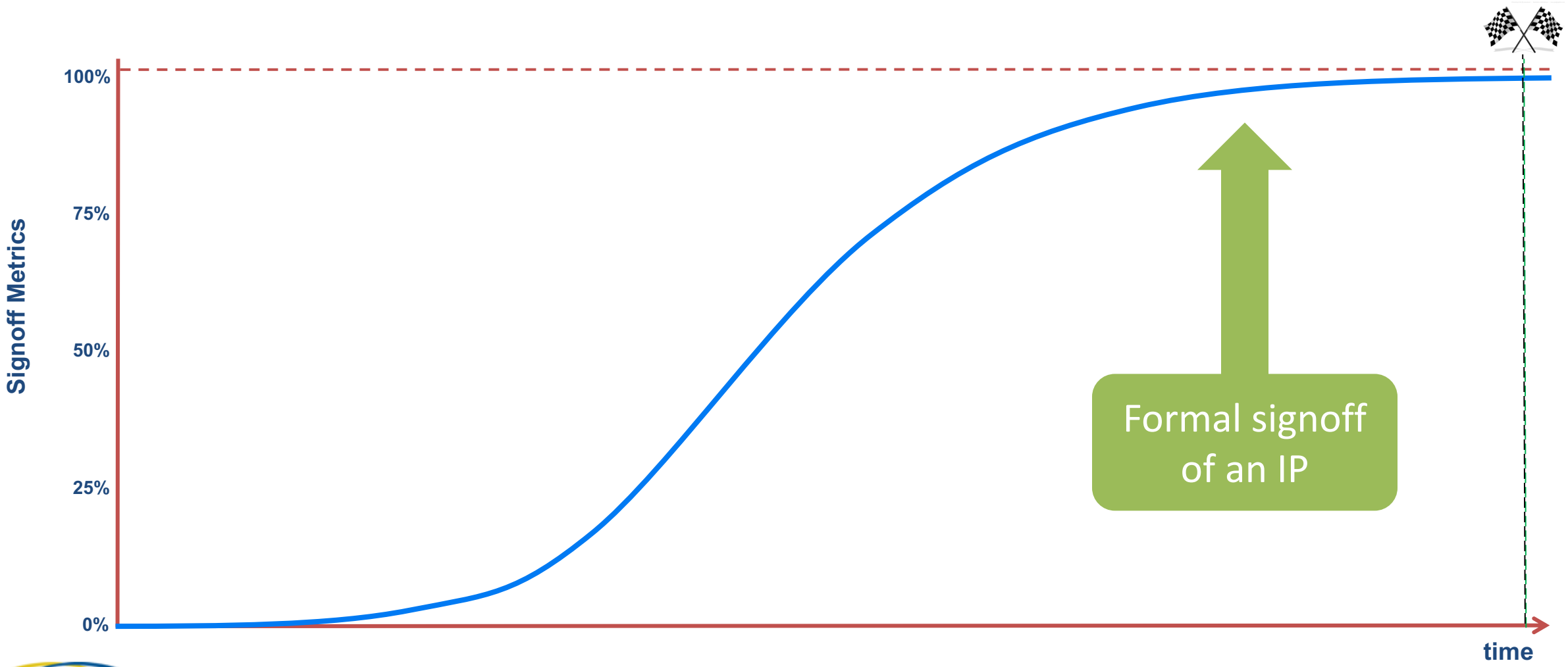
Use-case-based
- Define legal operations
- Workload matters: must represent real operation

Data Collection
- Non-intrusive data collection
- Use the right execution platform

Analysis
- Correlate, filter, learn, predict
- Anomaly detection

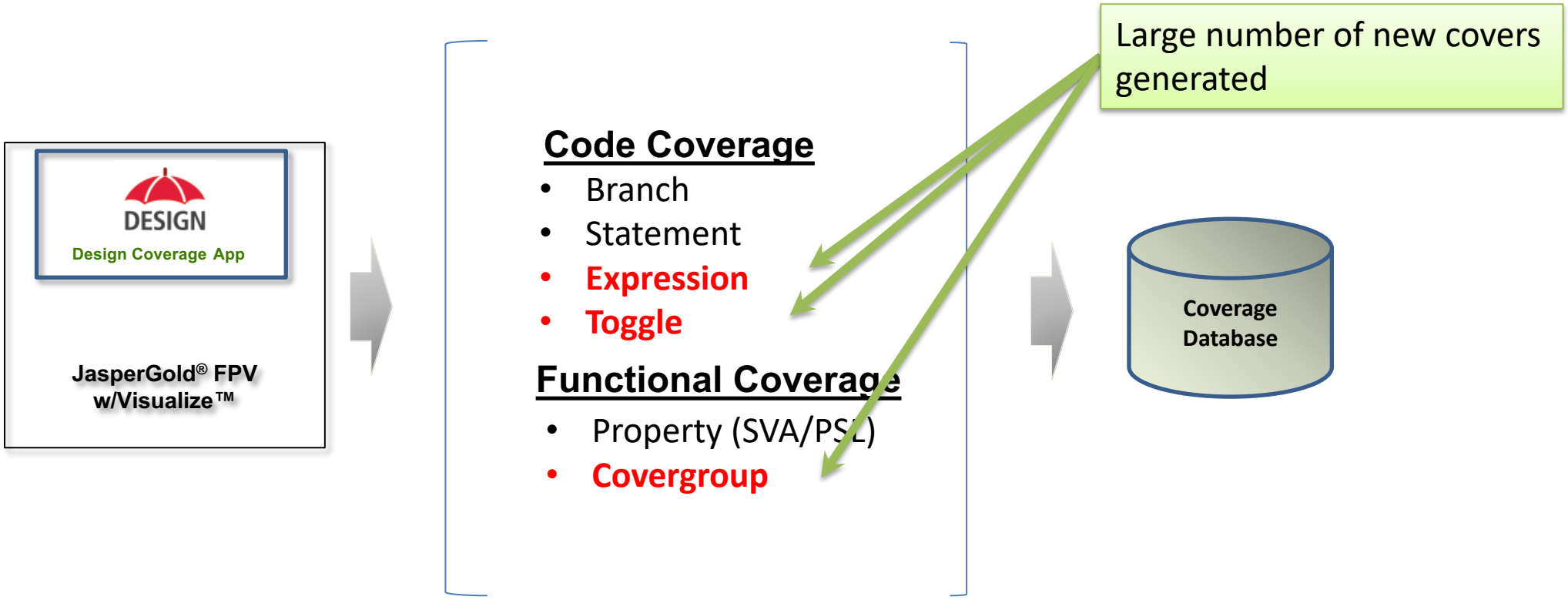
Goal-based
- Verification throughput
- Smarter bug hunting

Data to Drive…



Ever-Increasing Amount of Formal Coverage Data

- Formal Coverage models and types continue to expand



Formal-specific Coverage Types

Stimuli Coverage

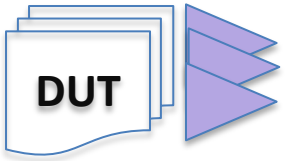
Formal Setup

How restrictive is the design behavior under the formal setup?

Is the design over-constrained?

Cone-of-Influence (COI) Coverage

Formal Setup

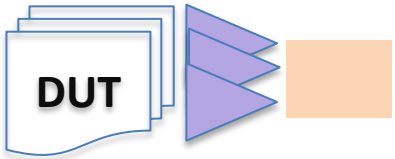


Properties (Structural COI analysis)

How complete is my property set?
Do I cover all design behaviors ?

Proof Coverage

Formal Setup



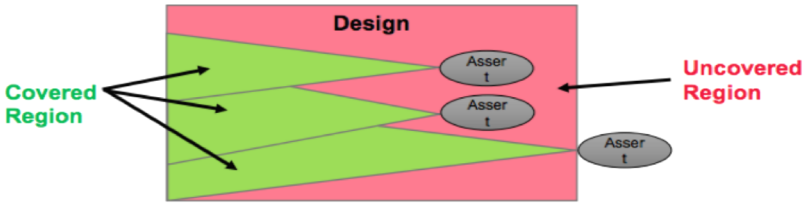
Proven Properties (Proof Core)

What coverage is achieved by the proven properties?

COI / Proof Core Coverage

Cone-Of-Influence Measurement

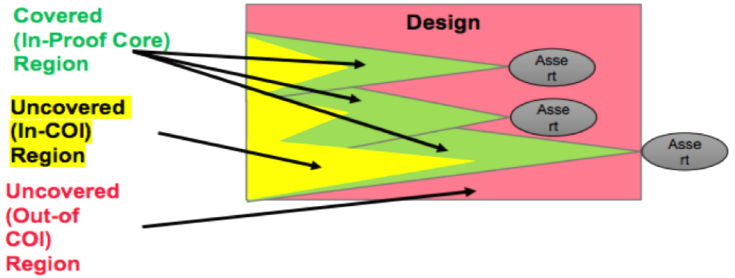
- Design fan-in is computed starting from assertion, traversing back to inputs
- The union of COIs from all assertions is reported
- Anything outside the COI region *cannot* influence assertion status
- Anything inside the COI region *may* influence assertion status
- Fast measurement – no formal engines are run



cadence

Proof Core Measurement

- The union of proof cores from all asserts is reported
- Anything outside the Proof Core region *cannot* influence assertion status
- Anything inside the Proof Core region *may* influence assertion status
- Slower measurement than COI – requires running formal engines



cadence

From COV App Rapid Adoption Kit on http://support.cadence.com

Formal-specific Coverage Types

Stimuli Coverage

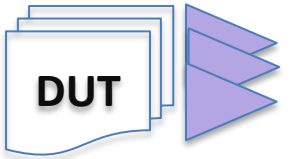
Formal Setup

How restrictive is the design behavior under the formal setup?

Is the design over-constrained?

Cone-of-Influence (COI) Coverage

Formal Setup

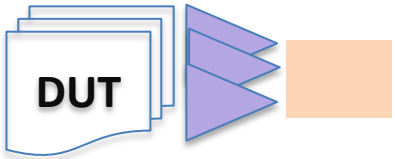


Properties (Structural COI analysis)

How complete is my property set?
Do I cover all design behaviors ?

Proof Coverage

Formal Setup

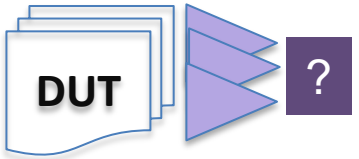


Proven Properties (Proof Core)

What coverage is achieved by the proven properties?

Bounded Proof Coverage

Formal Setup



Bounded Proof Analysis

What coverage is achieved by bounded proofs?
Is the bound enough? How to do better?

Multi-Dimensional Coverage Data

- Coverage data is multiplied by the unique coverage types offered by formal

Coverage Models

Coverage Types

	Branch	Statement	Expression	Toggle	Property	Covergroup
Reachability	✓	✓	✓	✓	✓	✓
Deadcode	✓	✓	✓	✓	✓	✓
COI	✓	✓	✓	✓		
Proof Core	✓	✓	✓	✓		
Bound	✓	✓	✓	✓	✓	✓

How to make sense of the data?

1) Abstract to more meaningful metrics
2) Provide an intuitive GUI to analyze results
3) Intelligent exclusions

Meaningful Metrics

Coverage Models

Coverage Types

	Branch	Statement	Expression	Toggle	Property	Covergroup
Reachability	✓	✓	✓	✓	✓	✓
Deadcode	✓	✓	✓	✓	✓	✓
COI	✓	✓	✓	✓		
Proof Core	✓	✓	✓	✓		
Bound	✓	✓	✓	✓	✓	✓

Stimuli Coverage

Checker Coverage

Bound Analysis

Stimuli Coverage

Checker Coverage

Bound Analysis

Stimulus exists that explores all code

Sufficient assertions exist that checks all code

In the case of an undetermined property, what is/is not covered

Meaningful Metrics

Coverage Models

Coverage Types

	Branch	Statement	Expression	Toggle	Property	Covergroup
Reachability	✓	✓	✓	✓	✓	✓
Deadcode	✓	✓	✓	✓	✓	✓
COI	✓	✓	✓	✓		
Proof Core	✓	✓	✓	✓		
Bound	✓	✓	✓	✓	✓	✓

Stimuli Coverage

Checker Coverage

Bound Analysis

Formal Coverage

Formal Coverage

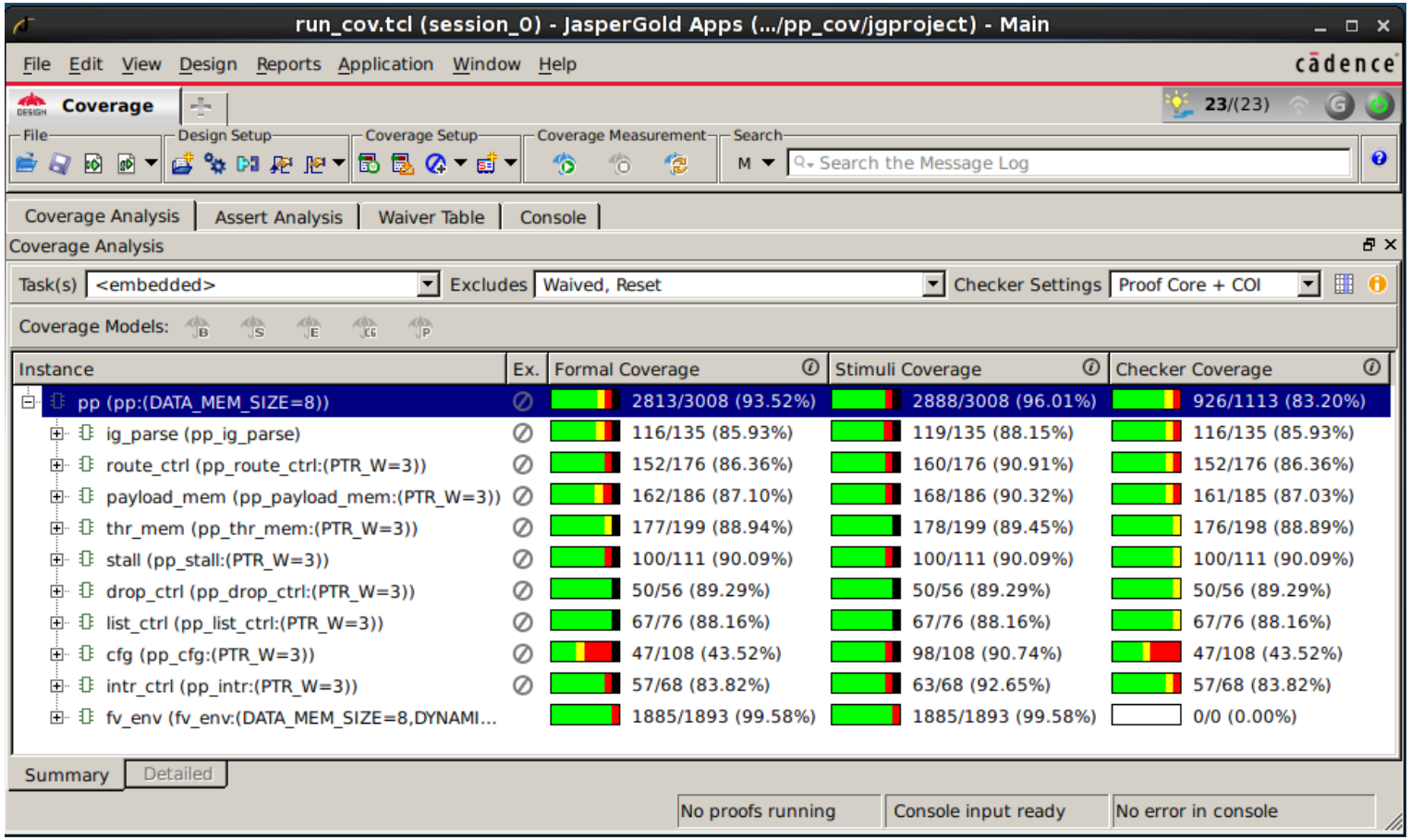
Code cover item can be exercised by the environment/inputs
AND
Has been checked by the assertions

Intuitive GUI

Formal Coverage

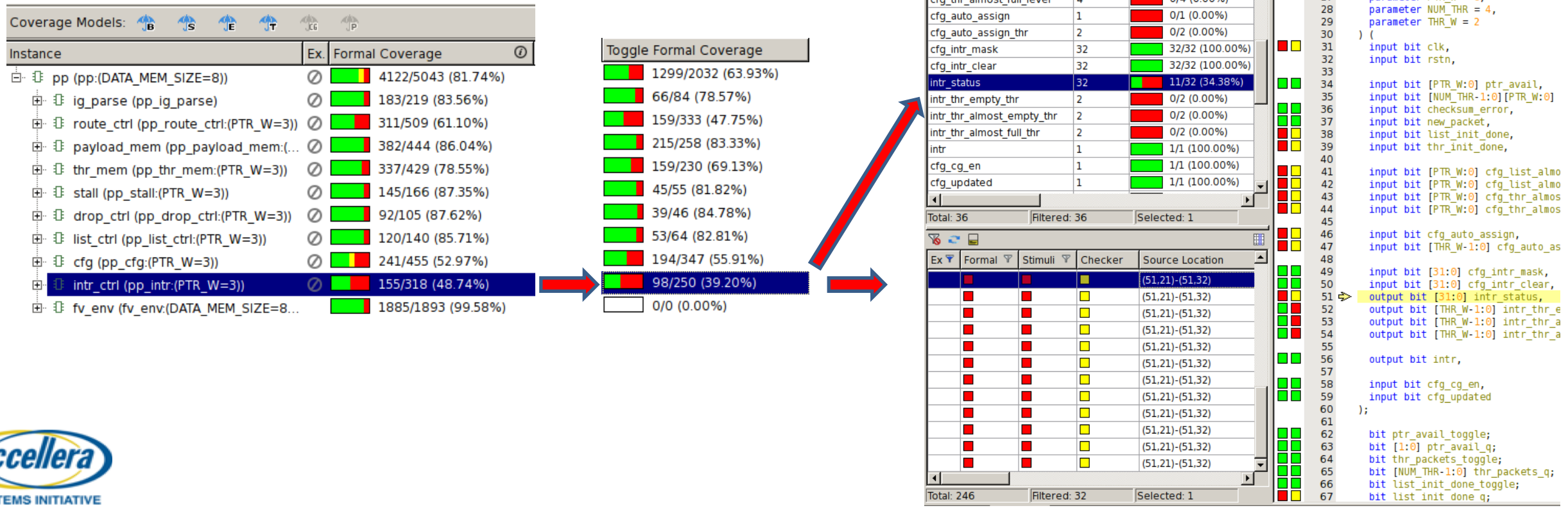
Stimuli Coverage

Checker Coverage



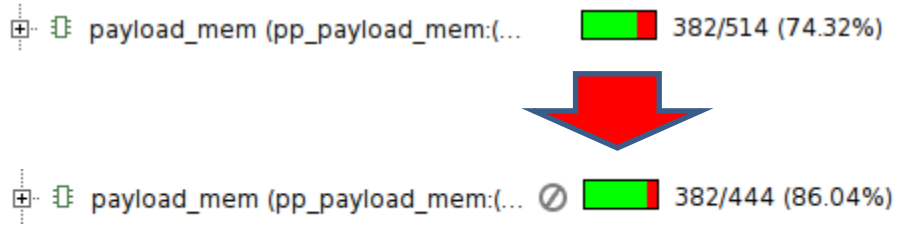
Intuitive Analysis

- Top-down navigation
  - Summary views reflect the progress of bug-hunting or signoff efforts
  - Quickly analyze the source of remaining gaps



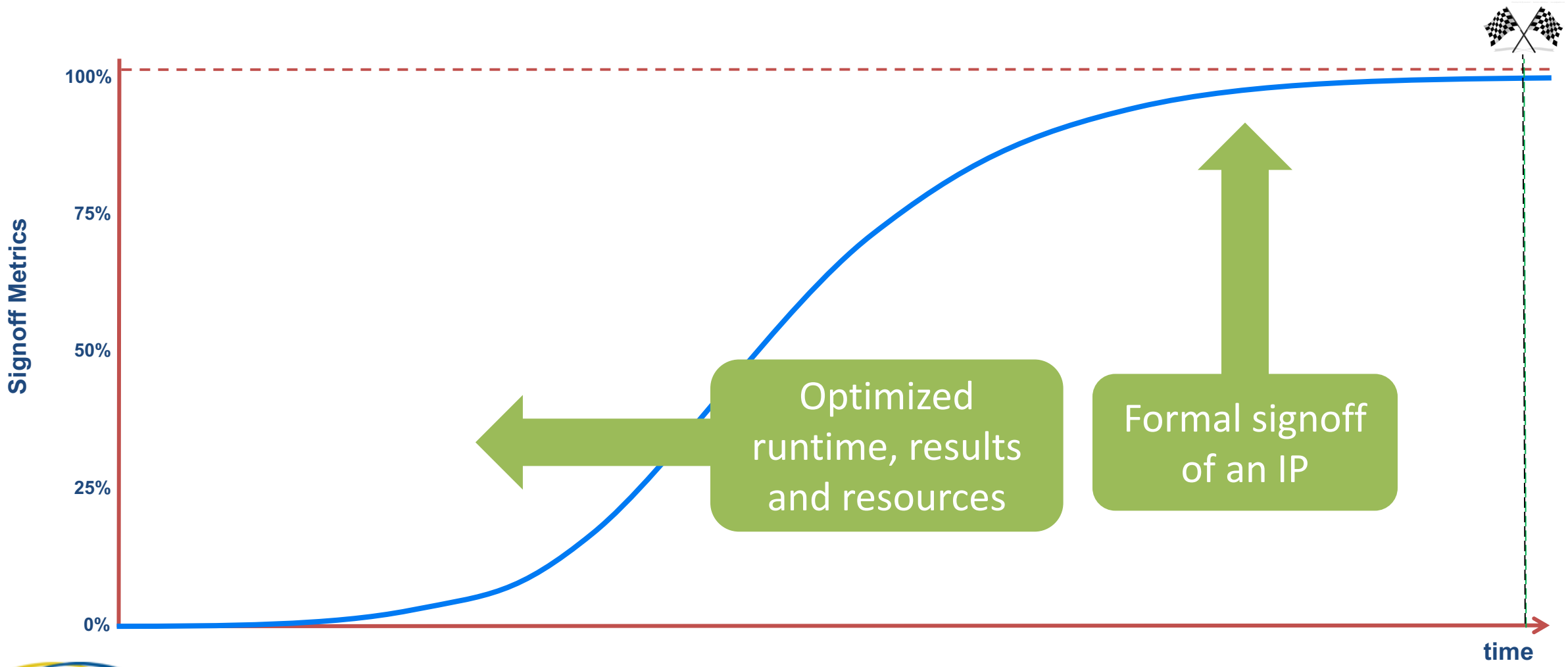
Intelligent Exclusions Save Effort

- Auto-exclude certain covers to reduce noise
  - Reset-related unreachable covers
  - Constant-related unreachable covers
  - Deadcode



- Advanced Waiver Capability
  - Persistent waivers tolerant of design changes
    - Avoids re-analyzing previously waived items
  - Waive-multiple by expression greatly reduces the number of user actions

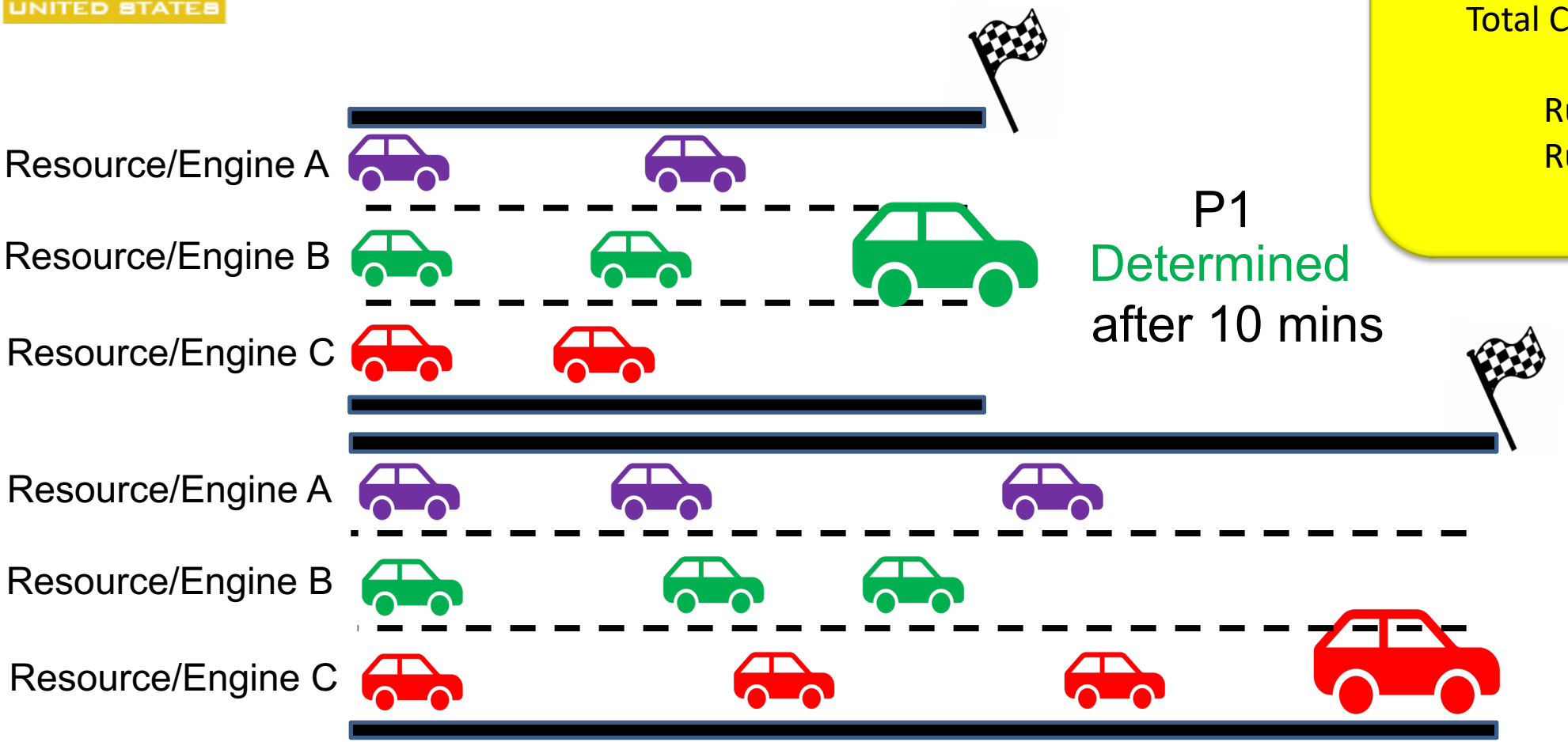
Data to Drive…



Problem Statement

3 CPUs x 10 mins = 30 mins
3 CPUs x 20 mins = 60 mins
Total CPU time = 90 mins

Run 2: 90 mins
Run 3: 90 mins
…

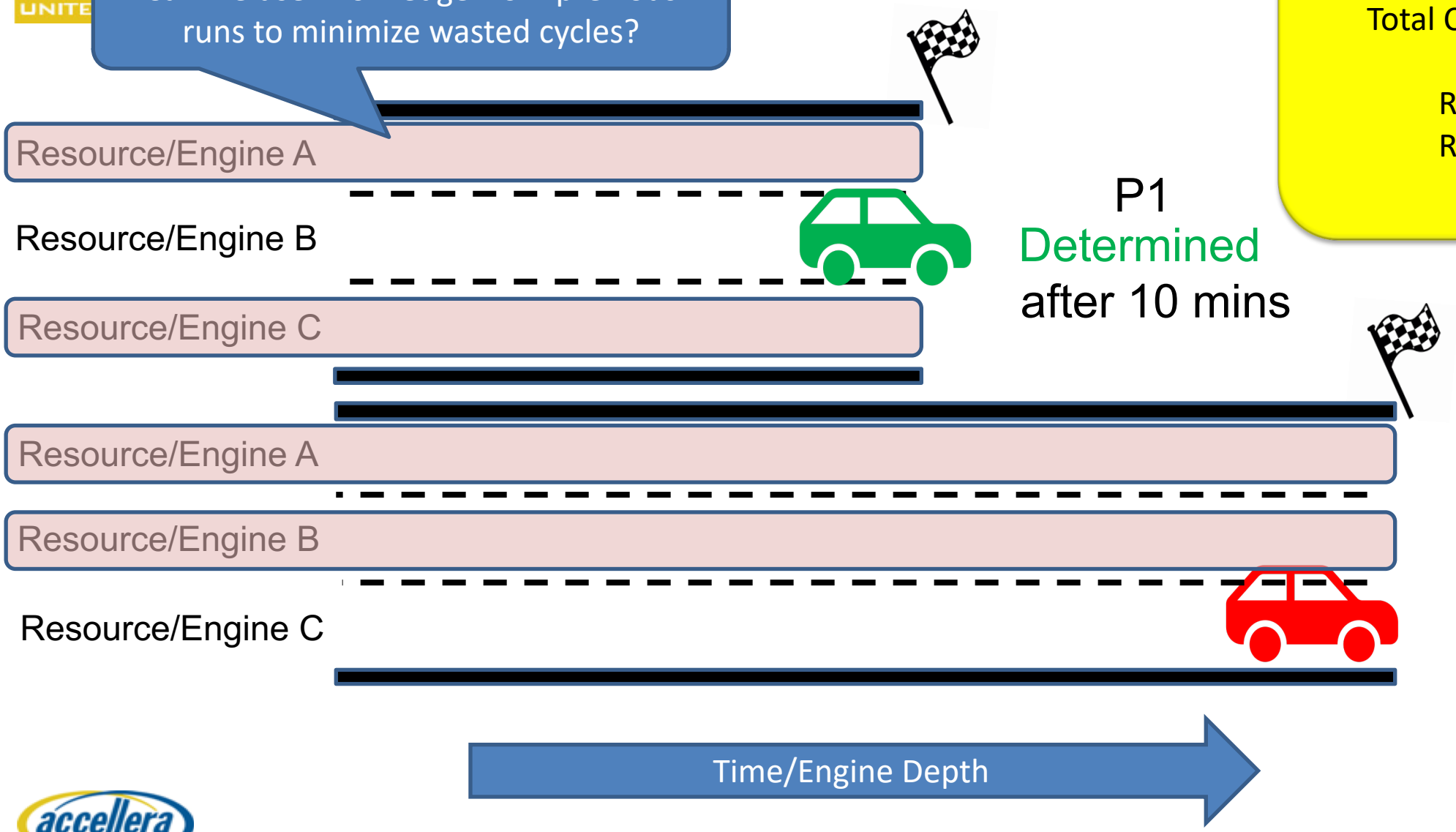


Problem Statement

Can we use knowledge from previous runs to minimize wasted cycles?

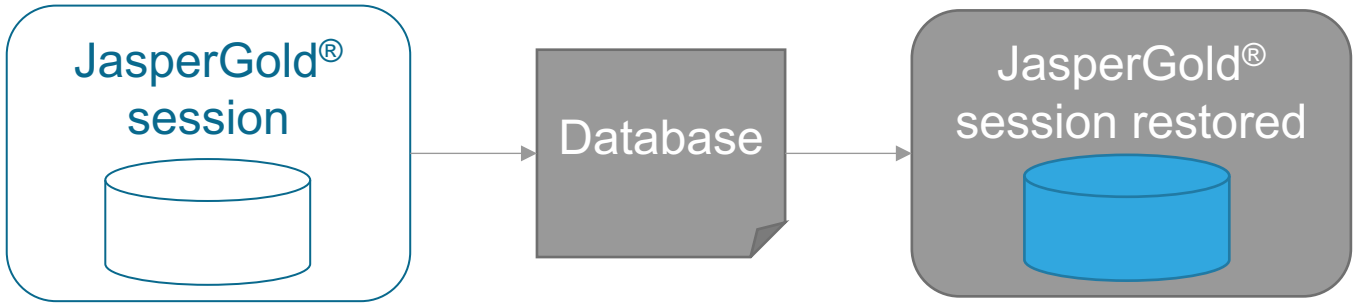
3 CPUs x 10 mins = 30 mins
3 CPUs x 20 mins = 60 mins
Total CPU time = 90 mins

Run 2: 90 mins
Run 3: 90 mins
…



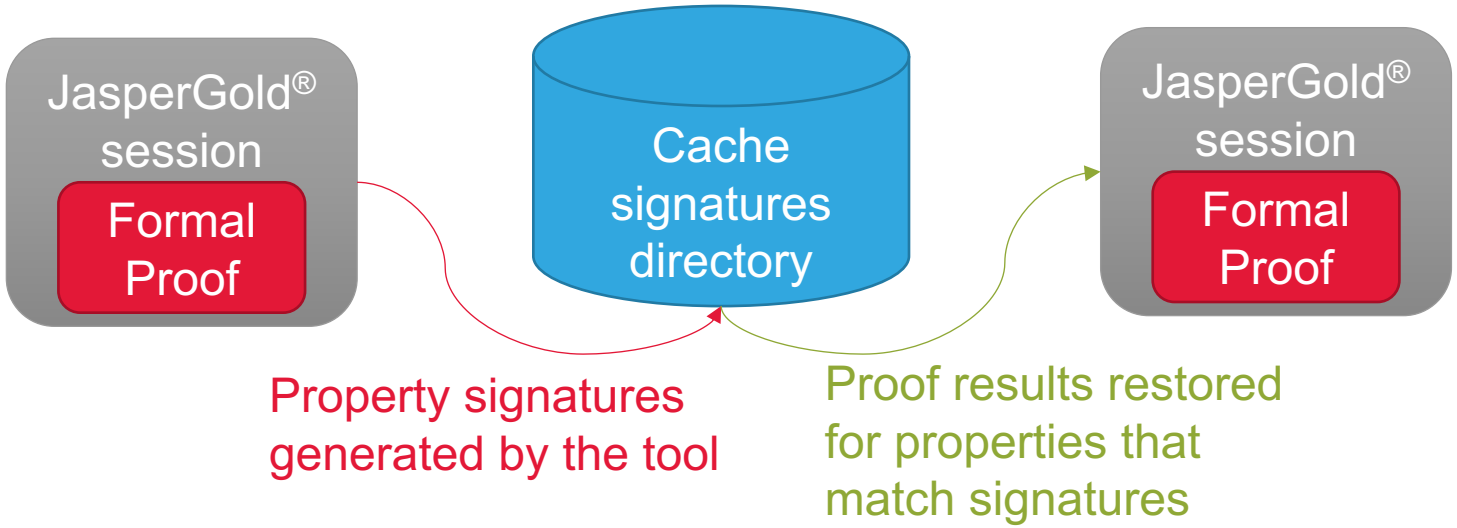
Simple Solutions

Save and restore flow



If design changes in any way, no proof result is restored

Proof cache



Proof starts from scratch on unmatched properties

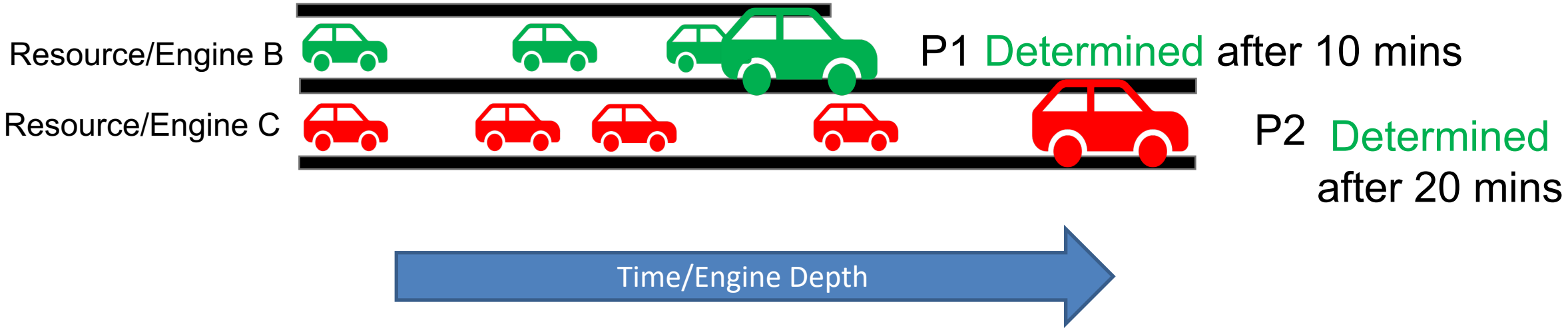
Need ability to learn from previous runs, to optimize subsequent proofs and smartly react to changes introduced to the design/environment

Challenge

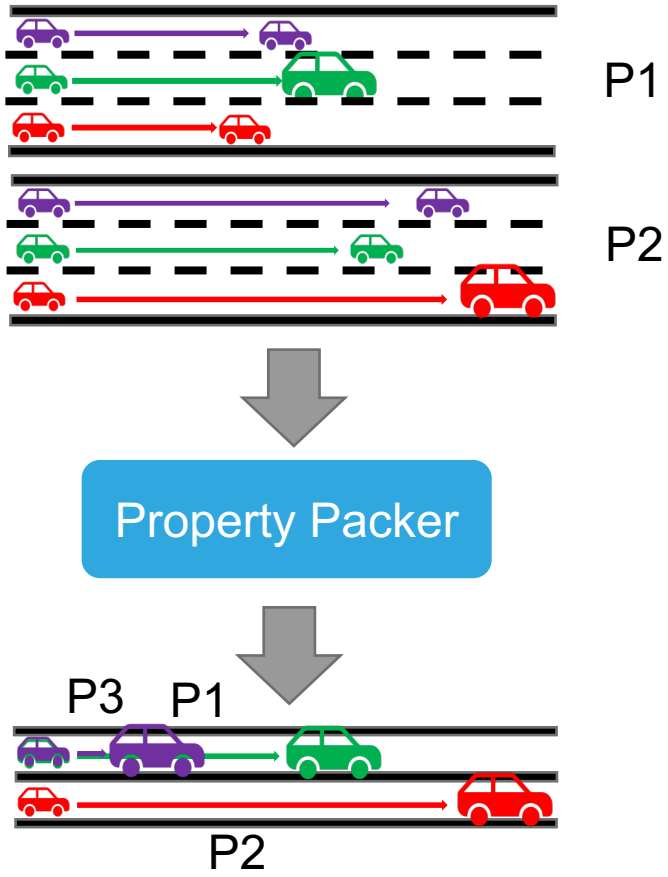
1 CPUs x 10 mins = 10 mins
1 CPUs x 20 mins = 20 mins
Total CPU time = 30 mins

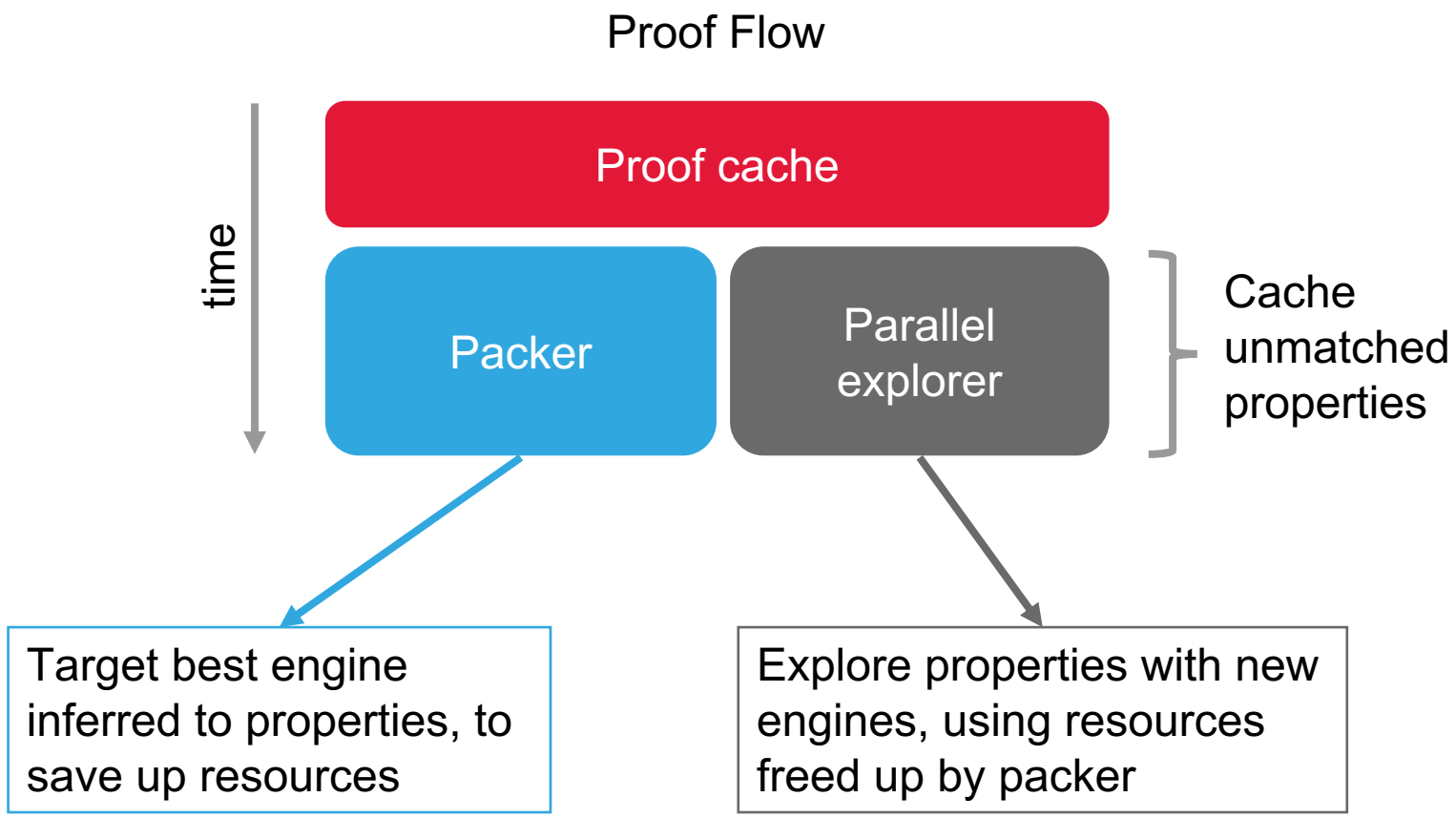
Run 2: 30 mins
Run 3: 30 mins
…

Use knowledge from previous run as a hint for subsequent run

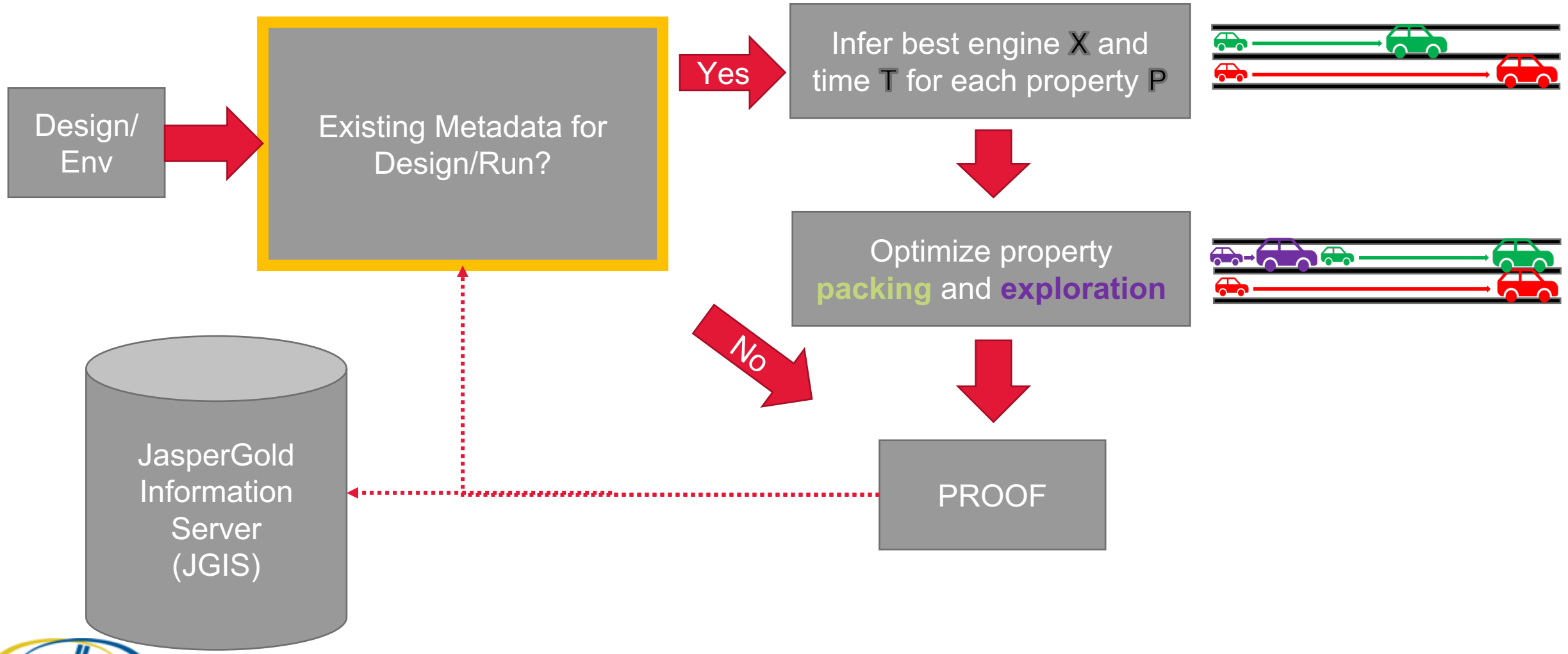


Property Packer/Proof Flow





Adaptive Regression (Cache "Miss")

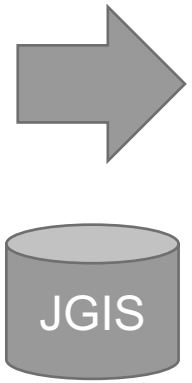


Adaptive Regression Example

- Learn best configuration on future runs, optimize continuously according to outcome

Prop	Status	Engine	Time
p1	determined	A,B,C...	35
p2	determined	A,B,C...	10
p3	undetermined	A,B,C...	60
p4	determined	A,B,C...	30
p5	determined	A,B,C...	10
p6	undetermined	A,B,C...	60

Run X



Prop	Status	AR inferred engine	Time	Parallel exploration with other engines	Time
p1	determined	C	50	A,B,D,E…	60
p2	determined	C	15	A,B,D,E…	15
p3	determined	B	20	A,C,D,E…	20
p4	determined	B	30	A,C,D,E…	30
p5	determined	A	10	B,C,D,E…	10
p6	undetermined	A	60	B,C,D,E…	60

Run Y

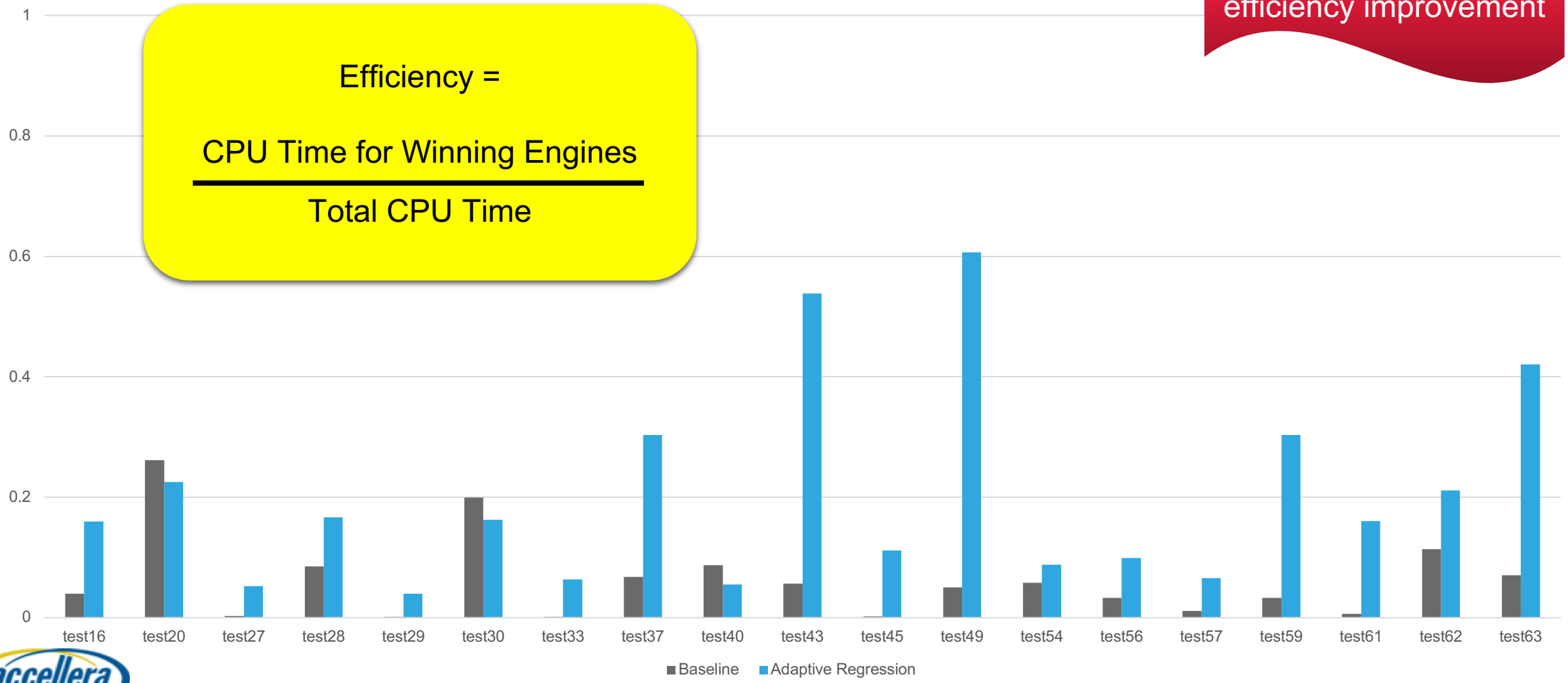
Design changes

- Speedup: select best engine and proof time per property based on previous runs
- Convergence: use saved up time to explore properties with additional engines

Results

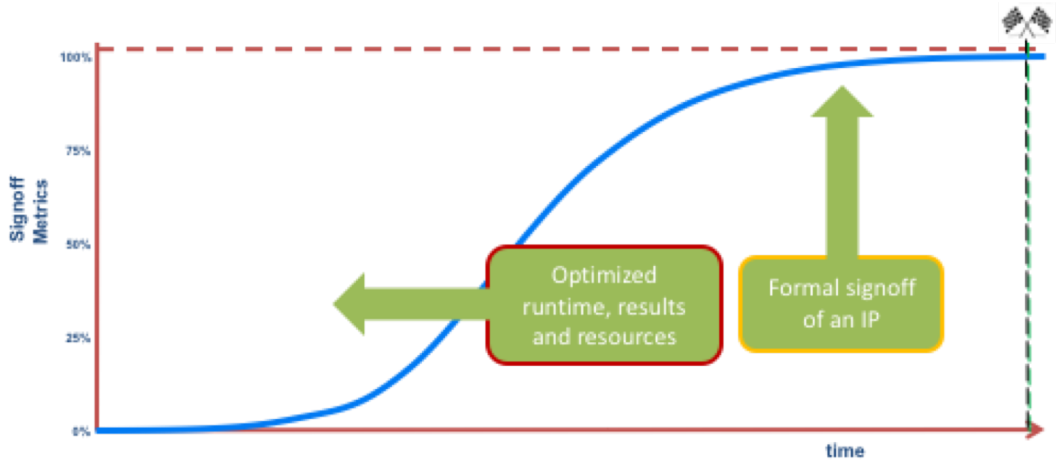
6x computational efficiency improvement

$$\text{Efficiency} = \frac{\text{CPU Time for Winning Engines}}{\text{Total CPU Time}}$$



Data-driven Formal Verification Summary

- Data to enable
  - User productivity
    - Analyze issues, measure formal verification progress/signoff when complemented with JasperGold COV GUI
  - Tool efficiency
    - Improve throughput and overall verification productivity with smart ML-based regression capability



Use-case-based	• Don't verify what won't happen • Workload matters (garbage-in/garbage-out)
Data Collection	• Non-intrusive data collection • Use the right execution platform
Analysis	• Correlate, filter, learn, predict
Goal-based	• Verification throughput • Smarter bug hunting

Michael Young, Director Product Management, Cadence

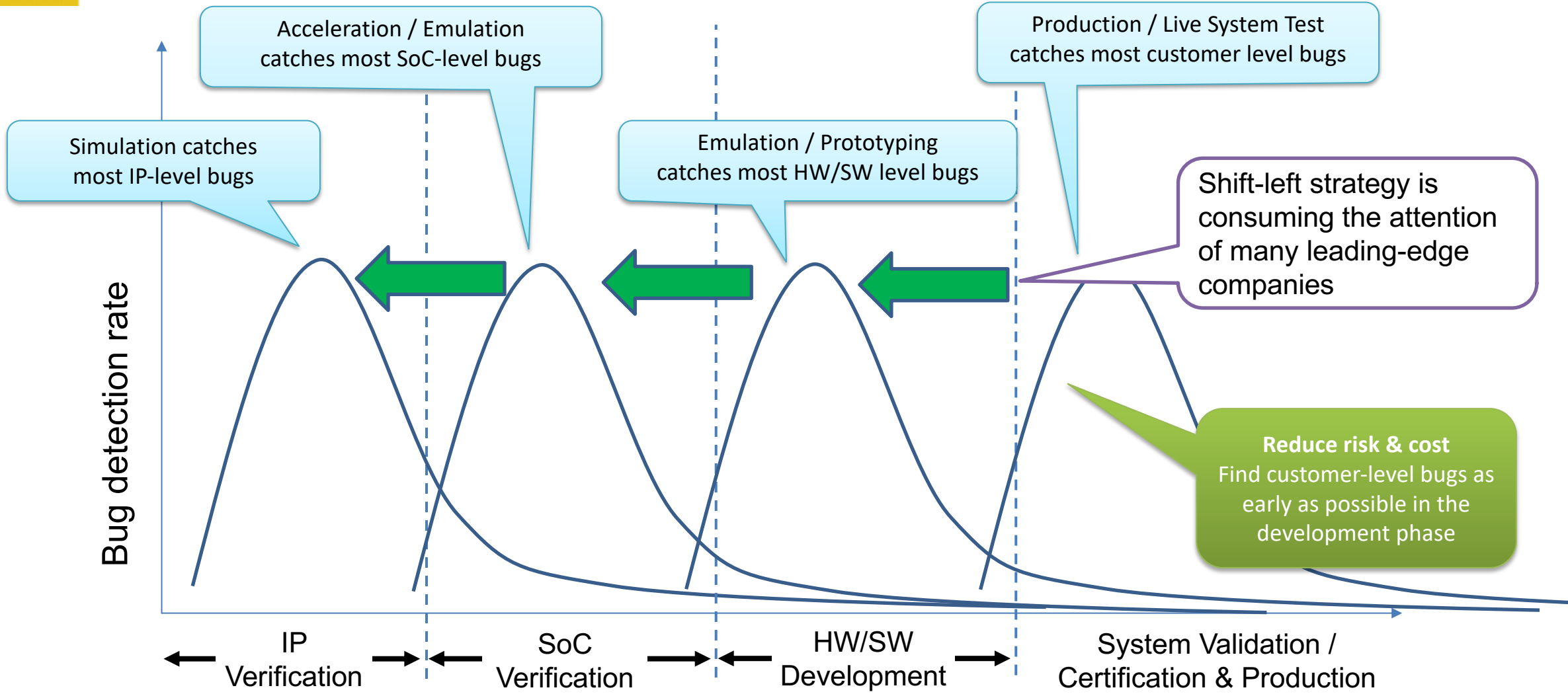
DATA-DRIVEN EMULATION

Data-Driven Emulation with Palladium

Use-case-based	• Define legal operations • Workload matters: must represent real operation
Data Collection	• Non-intrusive data collection • Use the right execution platform
Analysis	• Correlate, filter, learn, predict • Anomaly detection
Goal-based	• Verification throughput • Smarter bug hunting

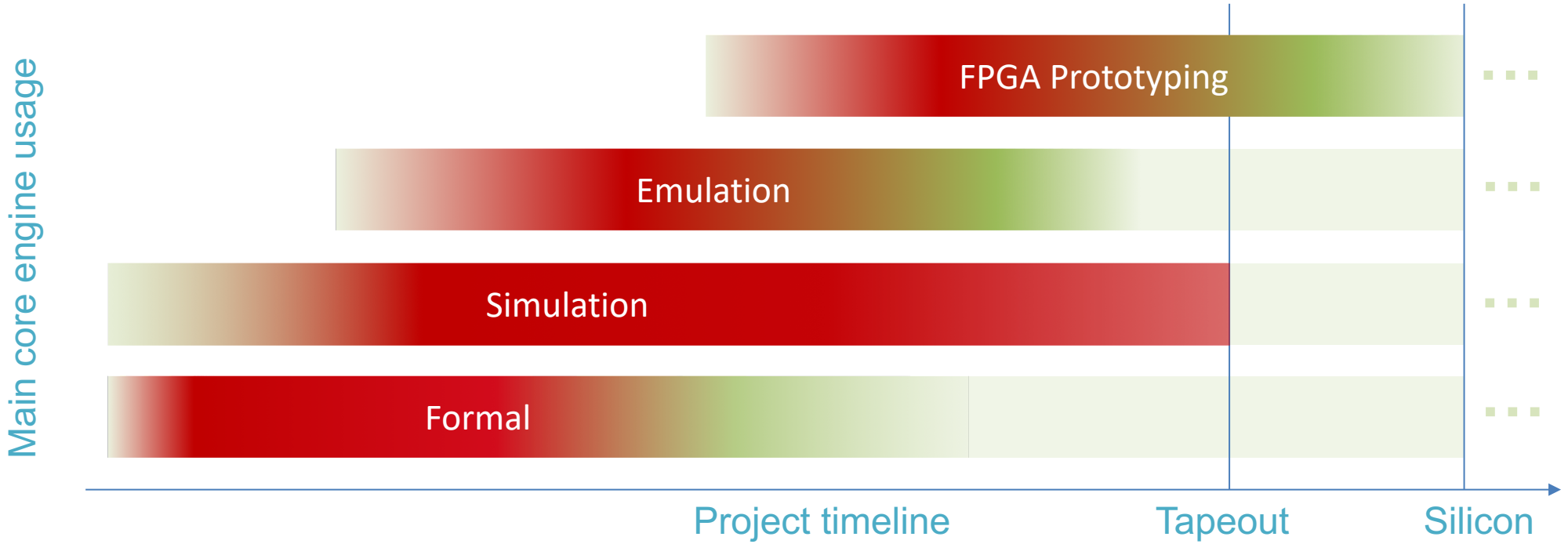
- Why emulate?
  - Palladium enables users to verify and test with *directed*, *pseudo-random*, *random*, lab-based, real-case scenarios that are typically not practical with other verification platforms especially during heavy HW/SW integration and co-debugging stages
- Emulation trends
  - Scalable models: IP to billion-class design
  - Ease of migration: simulation, prototype, etc.
  - Multi-chip and benchmark

Bug detection still not as early as possible



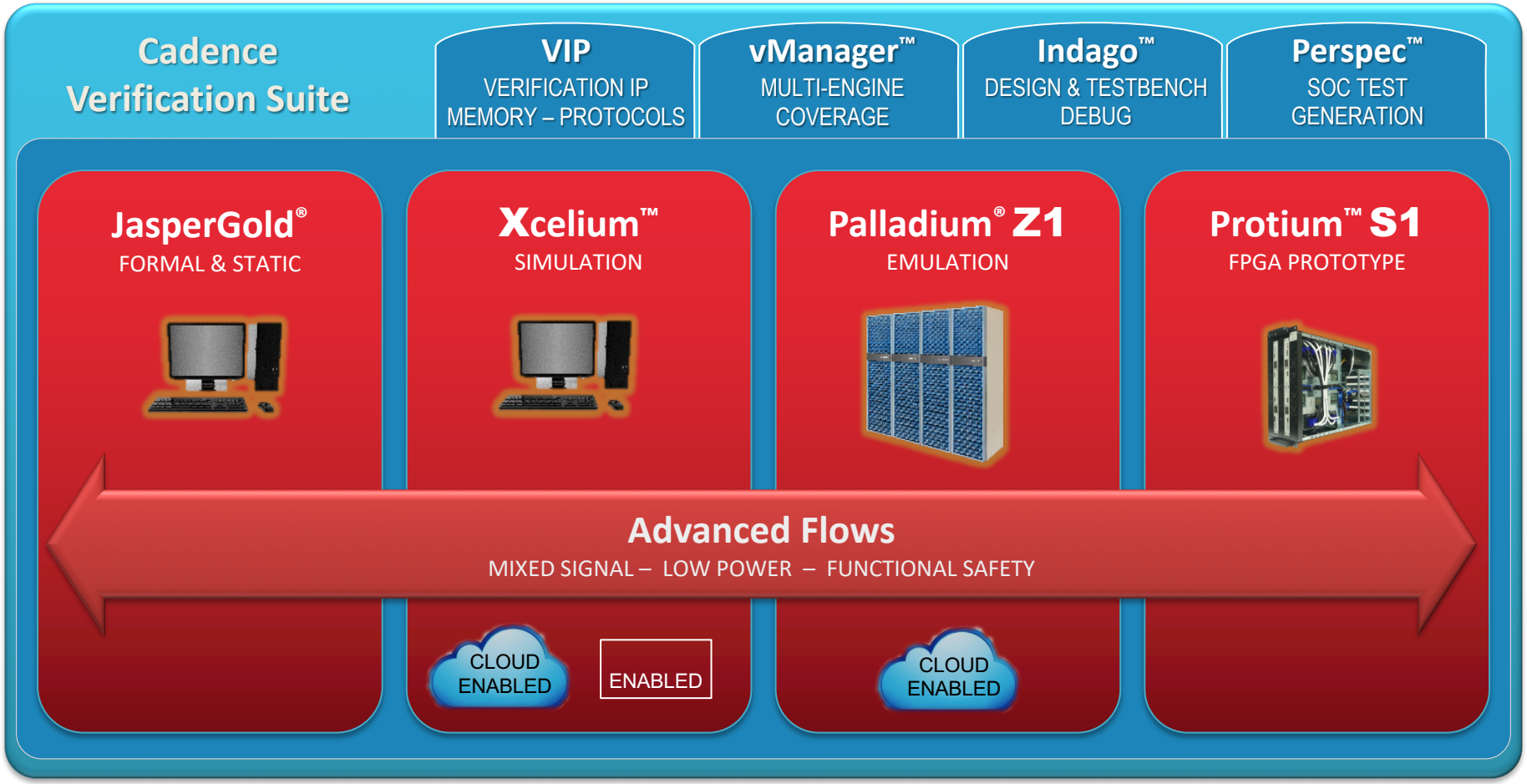
Customers Need the Fastest Engines

- **Ever-increasing verification** requirements driven by growing hardware and **software complexity**
- **Fast time to results** is essential to ensure projects can **meet schedules**
- **Right tools for the right job**: Combination of formal, simulation, emulation, and FPGA prototyping



2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

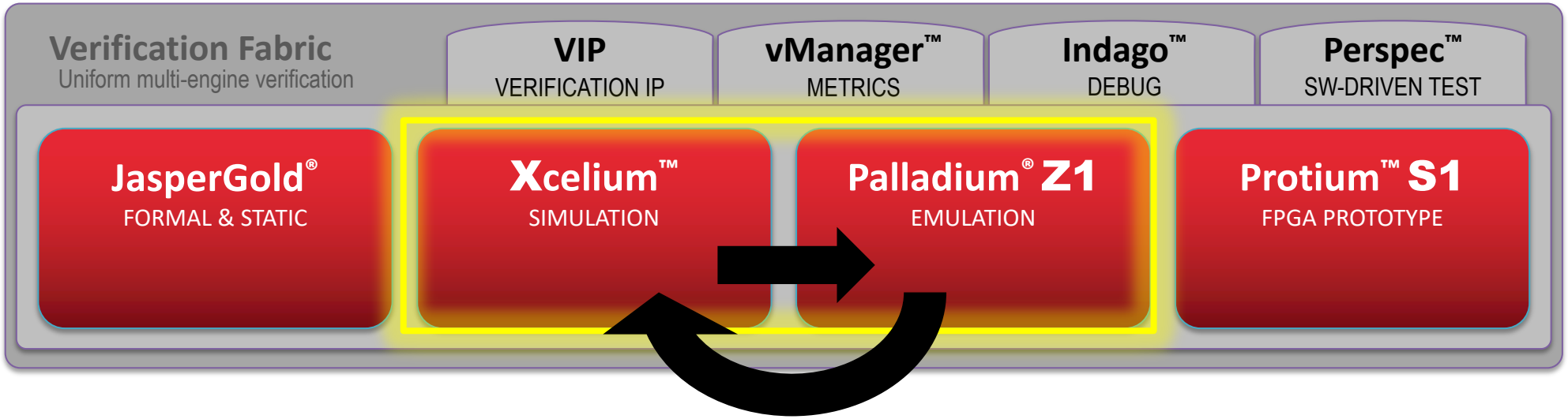
Cadence Verification solution



accellera SYSTEMS INITIATIVE

Verification Acceleration

Congruency between core engines

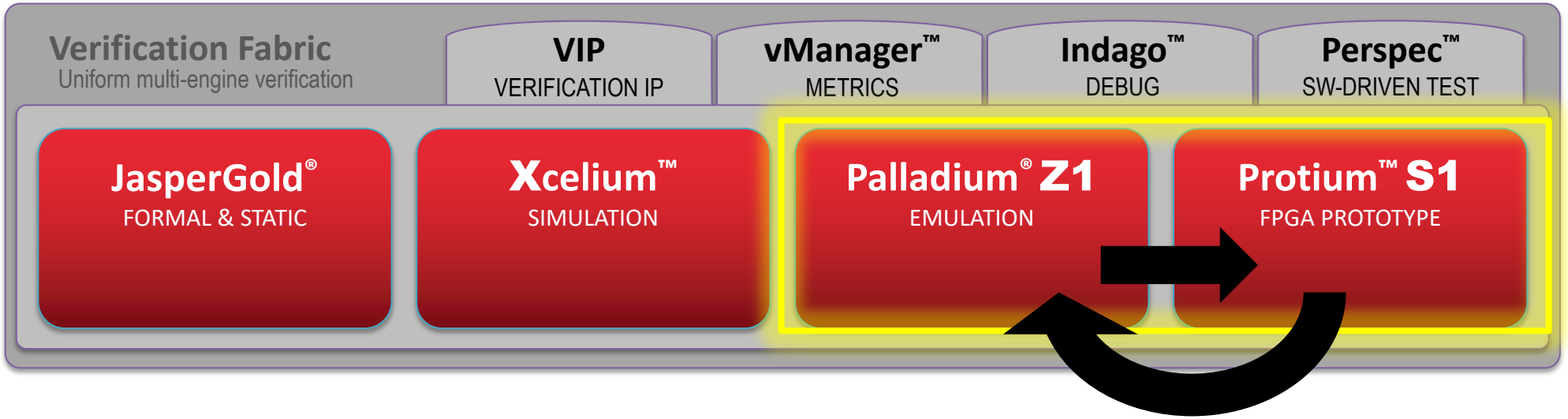


Xcelium-Palladium congruency

- Hybrid: Accelerate software bring-up
- UVM acceleration / hot swap
- Software driven verification and debug

Platform Congruency: Game Changer

Reducing bring-up time with Multi-fabric Compiler



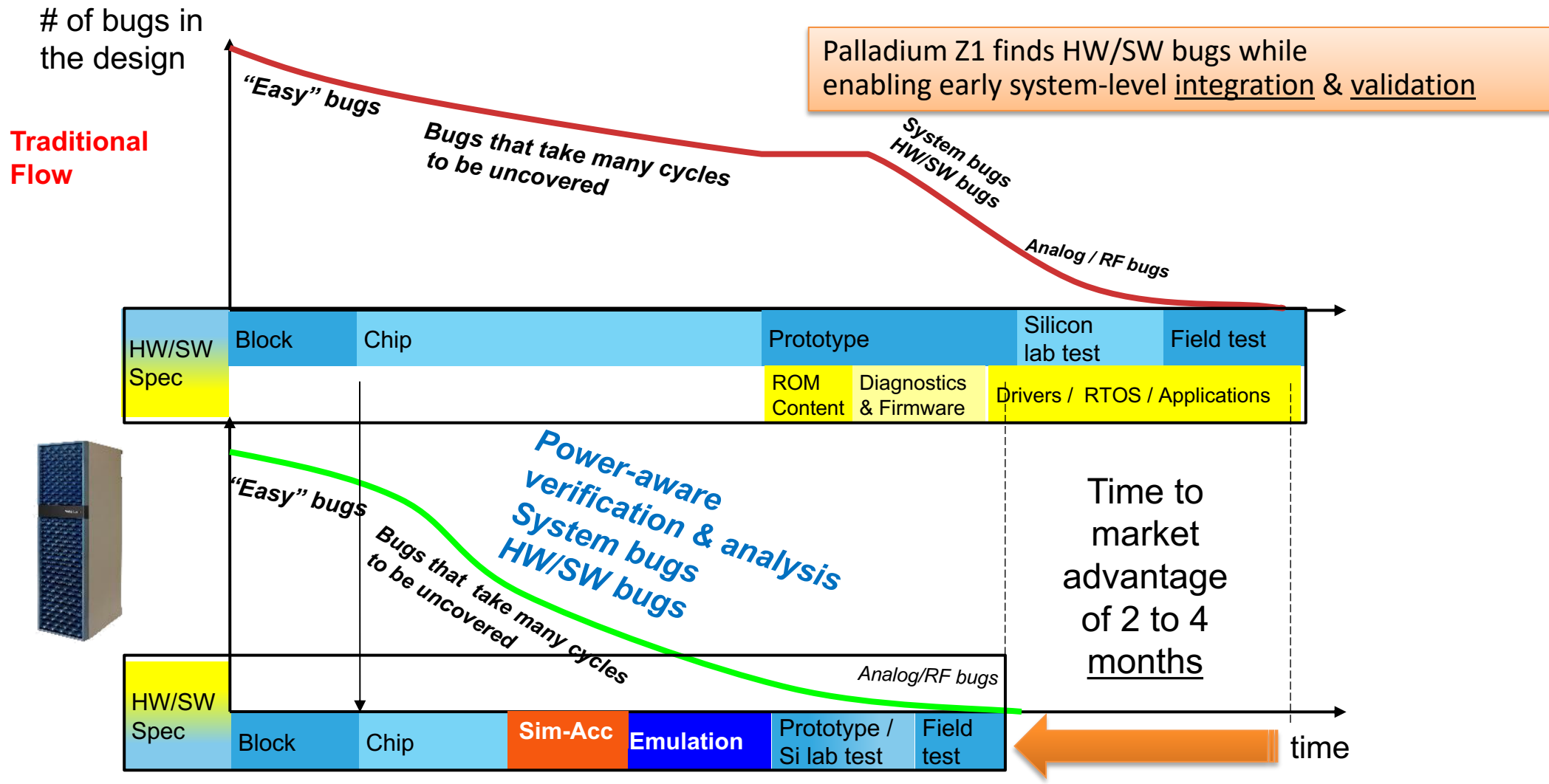
Palladium-Protium congruency

- Common Front-end
- Multi-fabric Compiler
- Combination enables debug and speed

Palladium Z1: core value proposition

Bridging the Productivity GAP

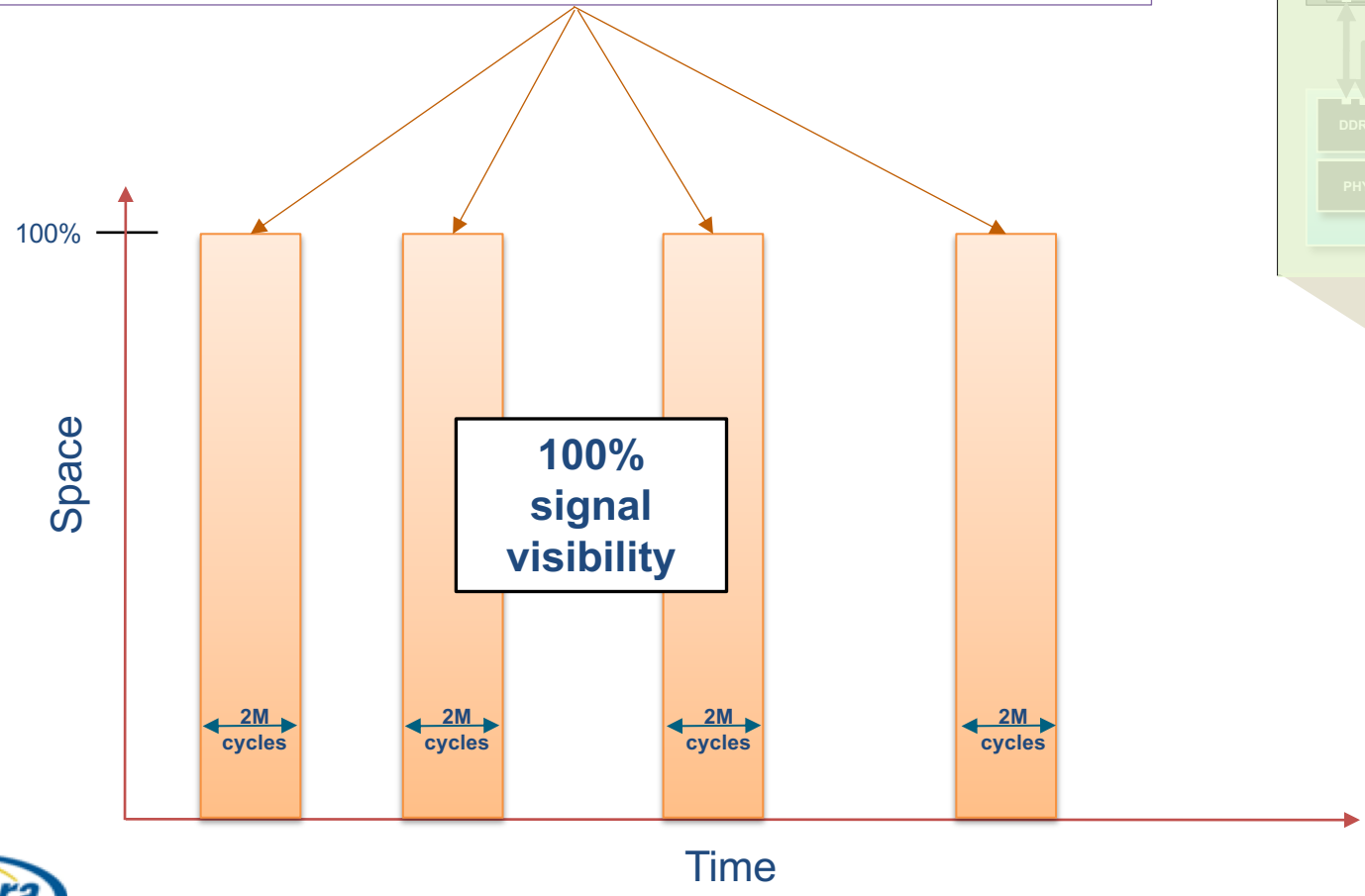
Palladium Z1 finds HW/SW bugs while enabling early system-level integration & validation

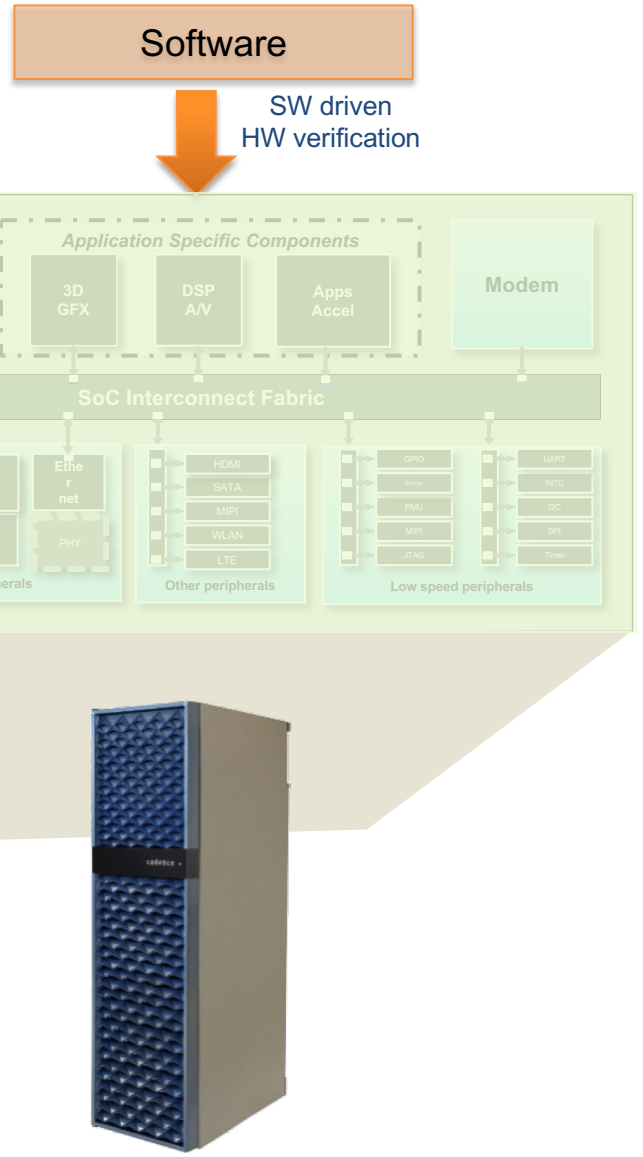


Debug with Palladium

Using FullVision (FV)

- Specify time window & capture up to 2M samples (typical)
- Trigger at points of interest
- Full signal depth captured

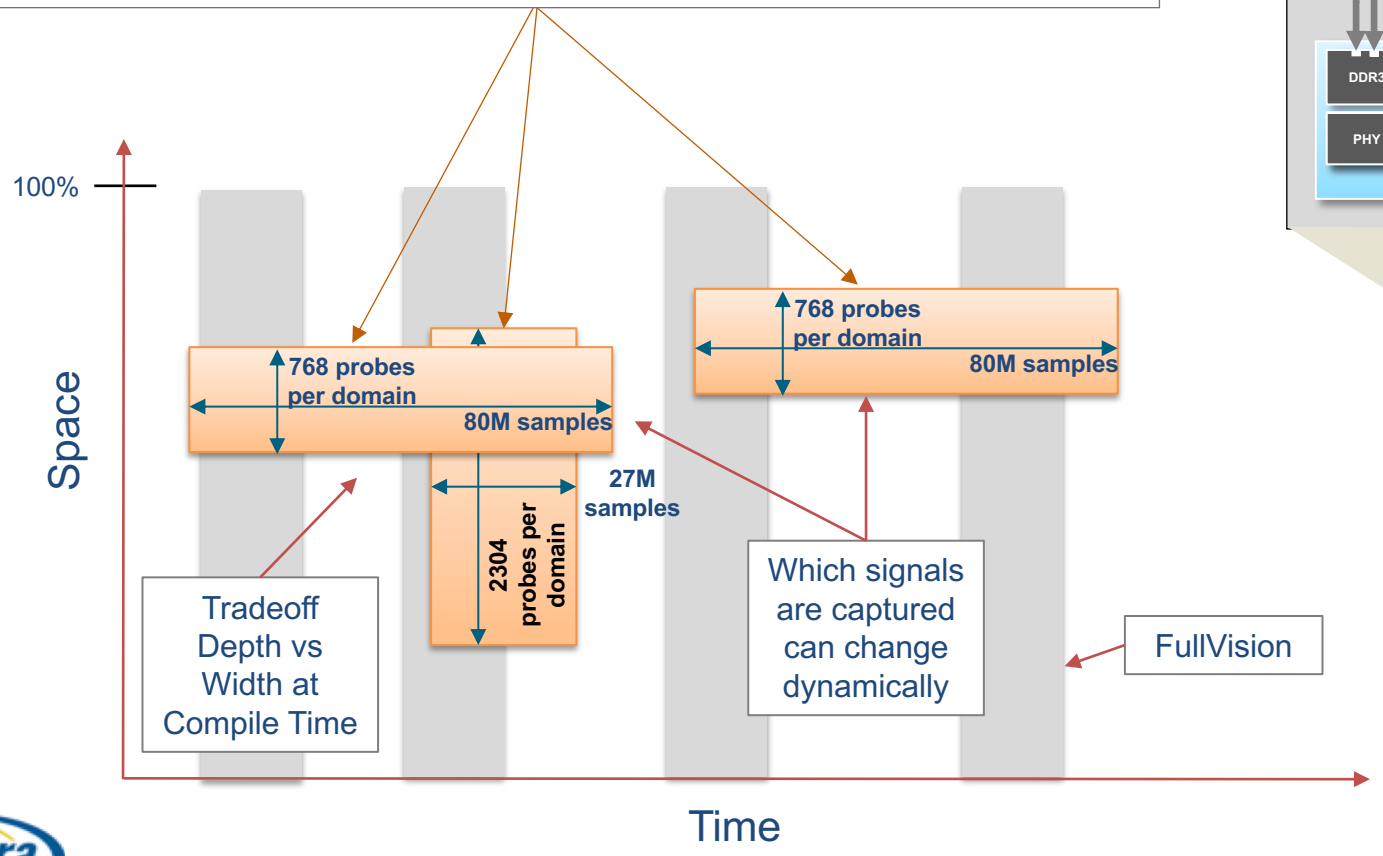


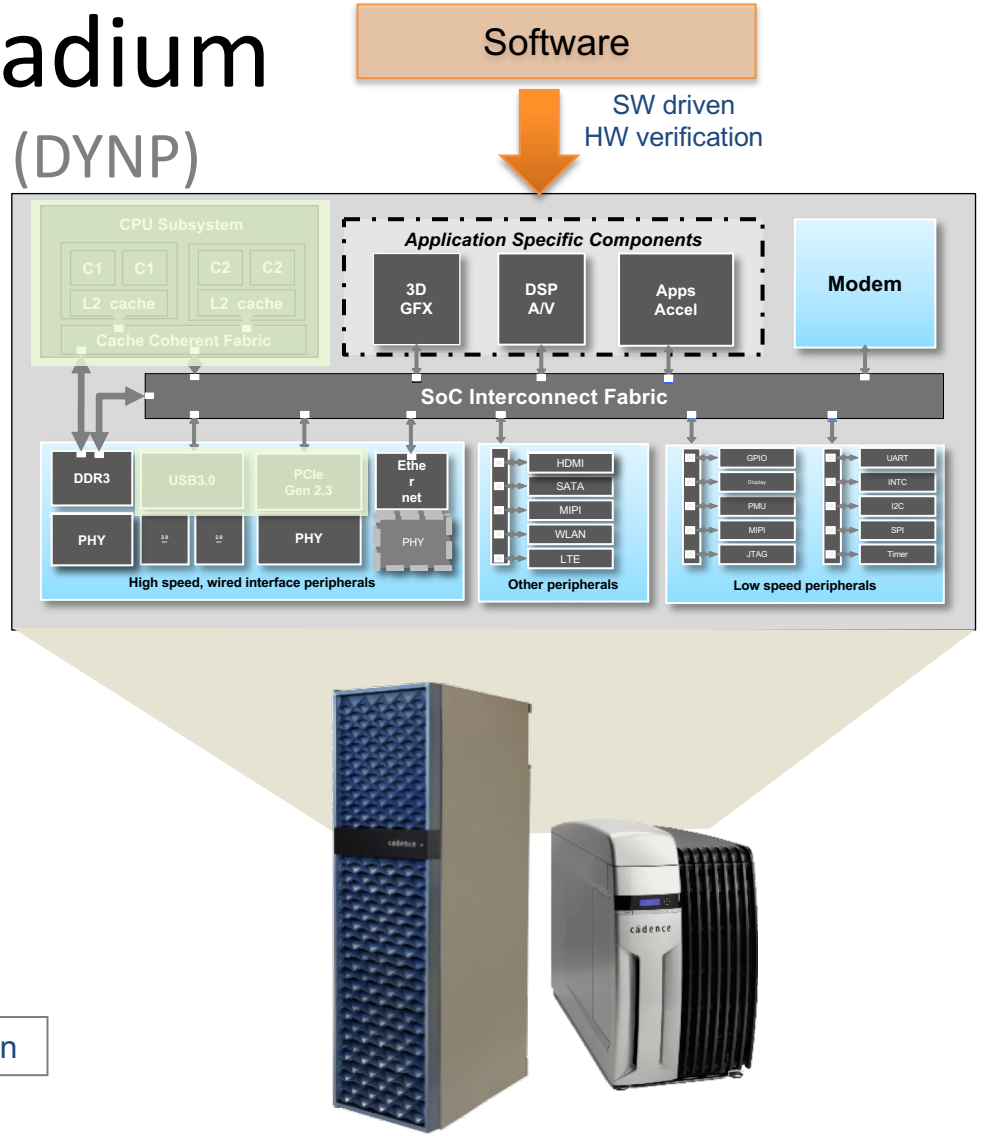


Debug with Palladium

Using Dynamic Probes (DYNP)

- Specify time window & capture up to 80M samples
- Vary sample size & probe depth
- Dynamically (at run time) choose the signals to capture
- Recompile design to change depth versus width

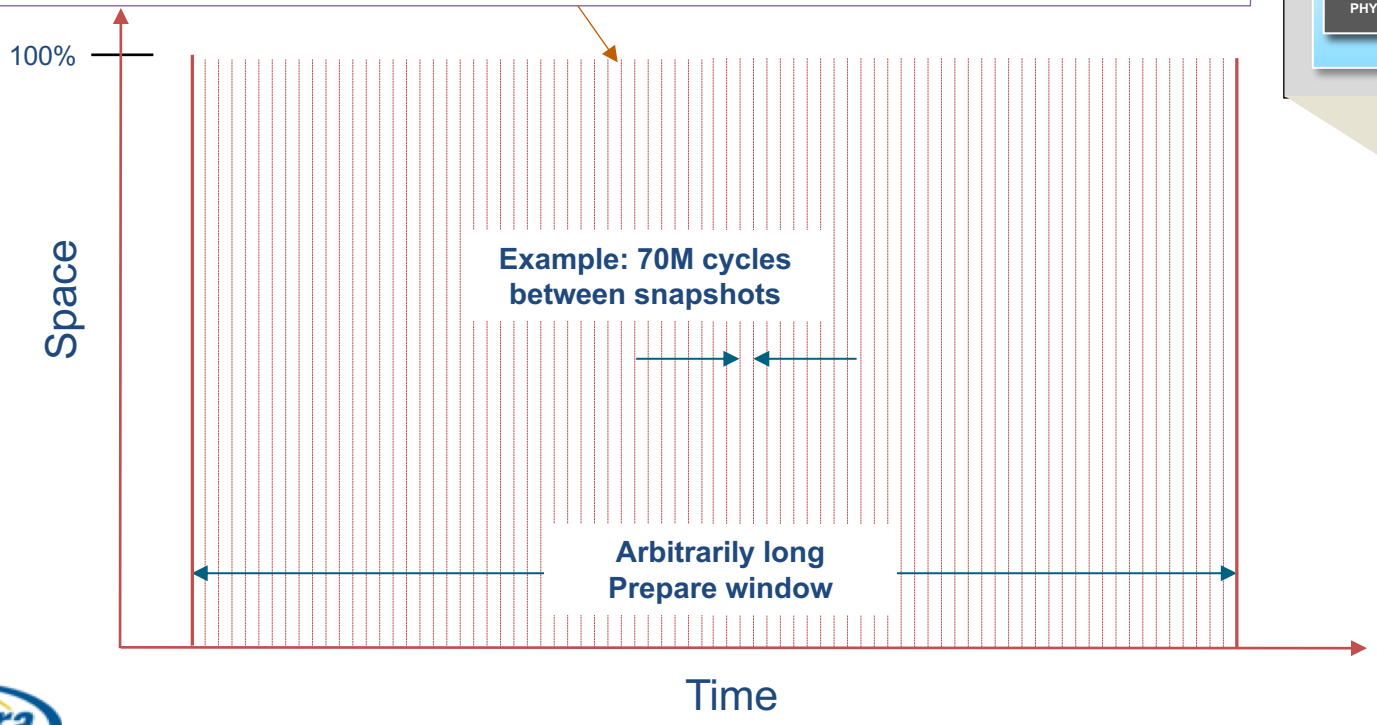


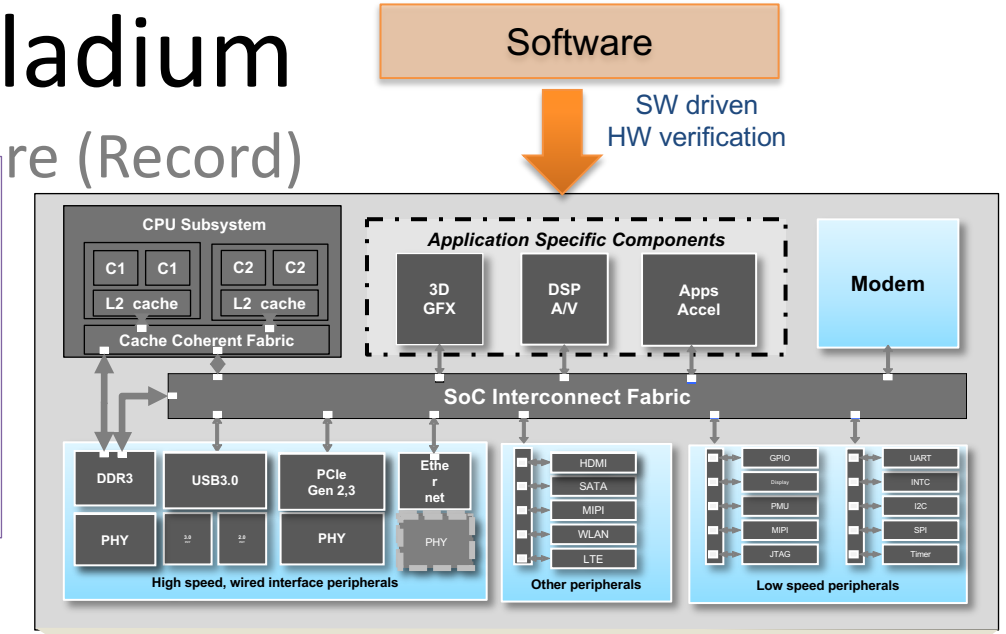


Debug with Palladium

Using InfiniTrace – Prepare (Record)

- During the Prepare session
  - Use all the normal commands for the run
  - Snapshots captured at user specified intervals automatically
  - Primary inputs / memory outputs are continuously captured
- Support included by default, just user enabled at run time
- Use in either Fullvision or Dynamic Probes mode
- Supported in all modes except with dynamic targets

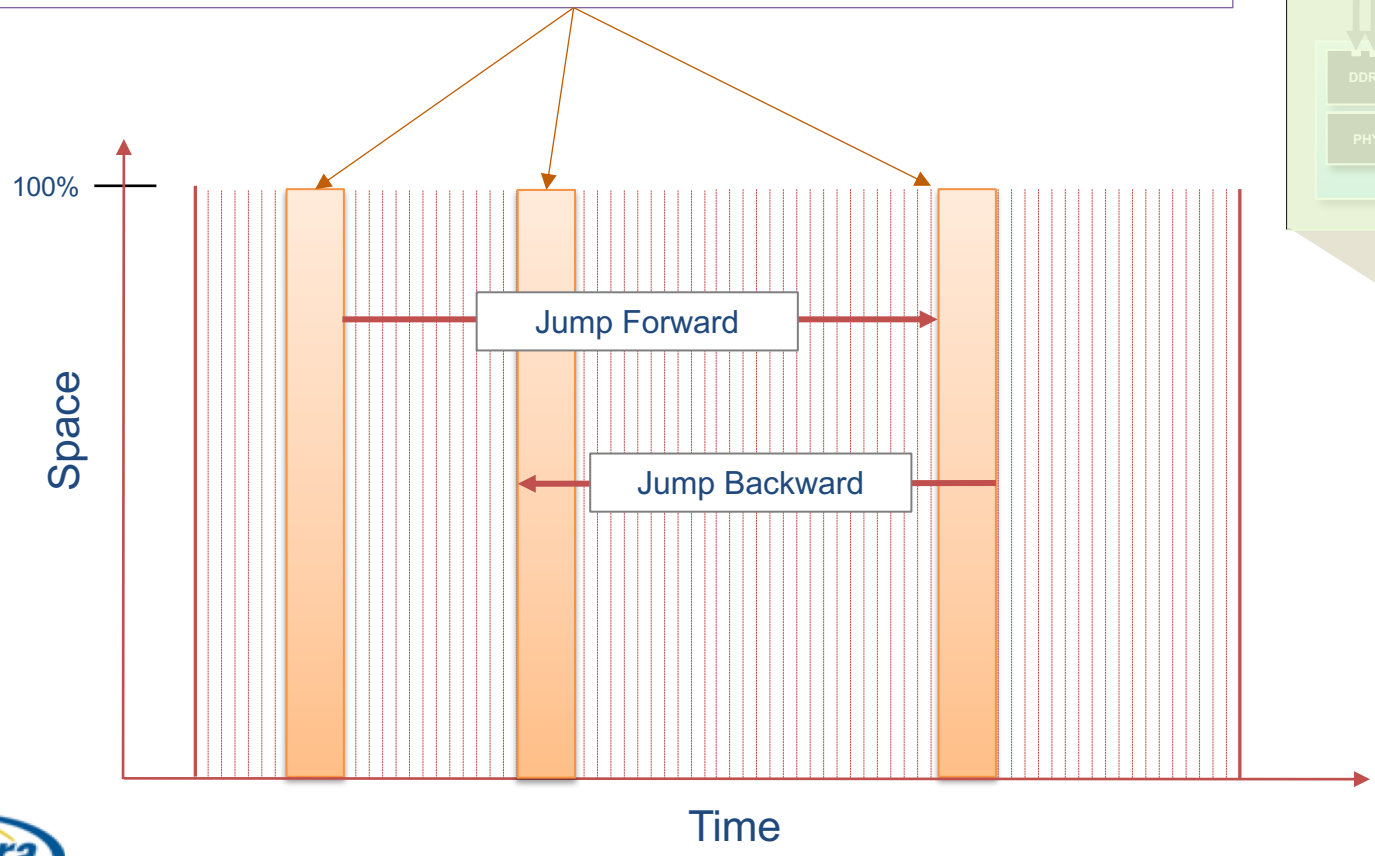


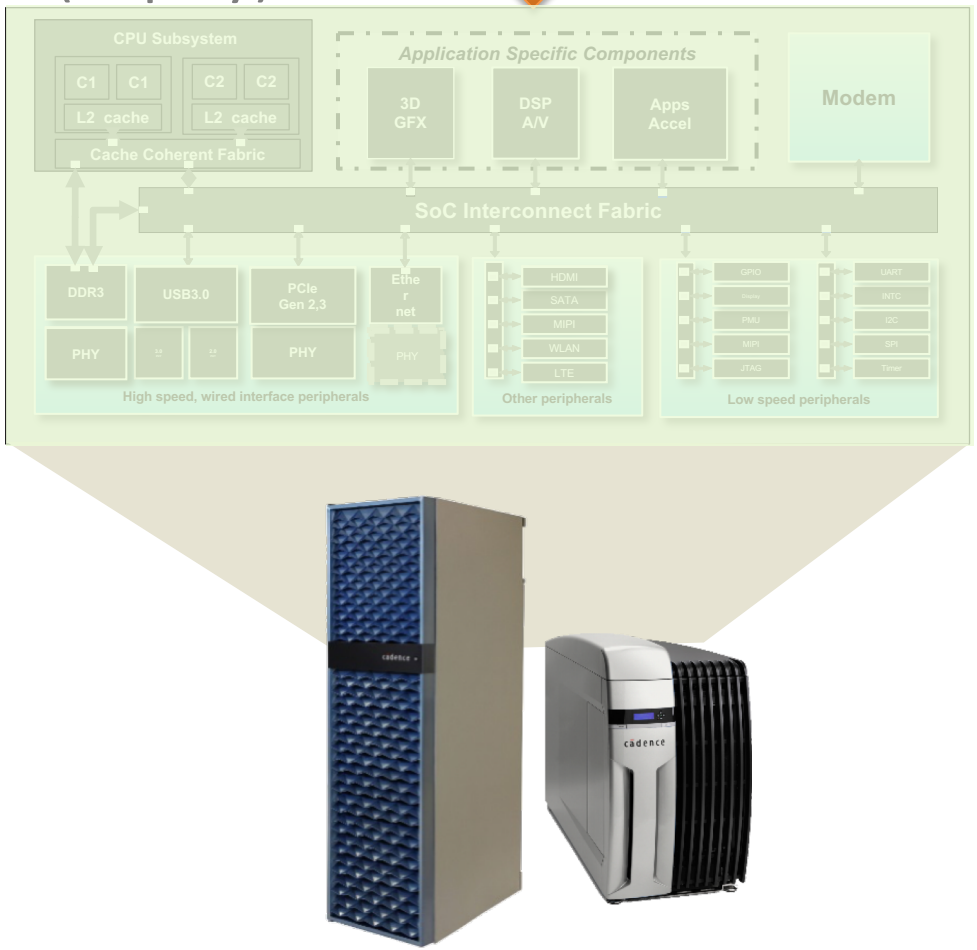


Debug with Palladium

Using Infinitrace – Observe (Replay)

- Jump to time window of interest using a specific time or a trigger
- Move forward and backward in time to capture window of interest
- Targets and testbench not used during the Observe session, just their recorded inputs are needed





State Description Language (SDL) - Intro

- SDL is the language you use to define a Trigger State machine. It has all the capabilities of commercial logic analyzers – plus more.
- When user-defined logic conditions are met, logic analyzer will “trigger”
  - In all modes, stop collection of trace data
  - In all modes except Logic Analyzer (LA) mode, stop the running design
    - In LA mode, trace data collection stops but design keeps running
- Trigger is like a simulation breakpoint
  - But can be more powerful, because triggering can be determined by a state machine that you define during debug
- Trigger state machine can be changed dynamically during a debug session, all signals available to SDL without recompiling the design

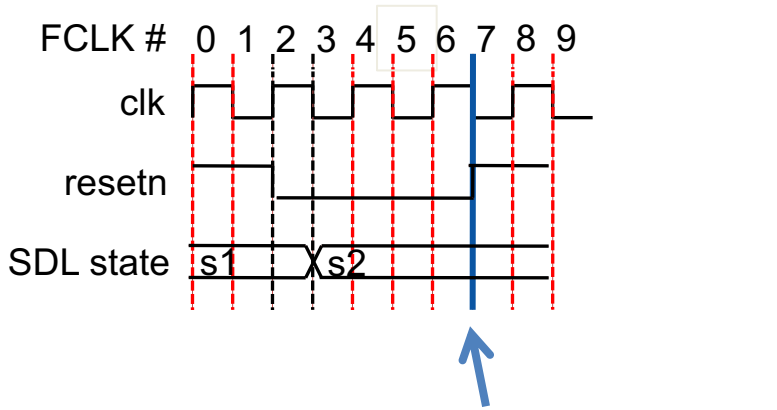
SDL – Basic Properties

- SDL tracks sequences of events by monitoring design objects such as signals, assertions, CPF/UPF objects using a state machine description
- Multiple instances of SDL can be used to track multiple independent sequences of events
- Each SDL instance has its own hardware resources:
  - One state machine
  - Expression evaluators (can be used inside state machines, or independently)
  - 2 general purpose counters (for counting events)
- Each SDL instance can perform, on a cycle by cycle basis, any of the following actions:
  - ACQUIRE: decide whether an individual probe sample should be acquired or rejected
  - TRIGGER: stop design clocks and/or waveform acquisition (depends on settings)
  - EXEC: Execute a TCL/XEL command/proc
  - DISPLAY: print out a formatted message, including time and signal values
  - Control internal SDL resources (go to a different state, increment/decrement/load counters, etc.)

SDL – Execution Model

- At the beginning of the run we are in the first state of the SDL program
- At each FCLK, SDL program can only be in one state
- If in a certain FCLK we are in state S1 and we execute “Goto S2”, then in the next FCLK we will be in state S2
- At each FCLK,
  - First, all signals in the design are updated
  - Then, all the tests in the SDL for current state are evaluated concurrently
  - Then, (depending on the test results) 0 or more actions are executed concurrently

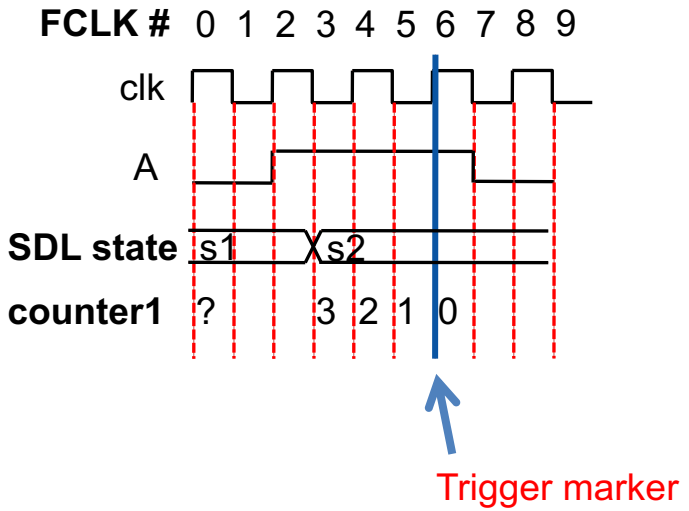
```
State  s1
{
    if ( resetn == ‘b0 ) {
       goto s2;
    }
}
State  s2
{
    if ( resetn == ‘b1 ) {
       Trigger;
    }
}
```



SDL – Extended Example

```
State  s1
{
     if ( A == 'b1 ) {
       load counter1 4;
       Goto s2;
     }
}
State  s2
{
    if ( A == 'b0) {
       goto s1;
    } else if (counter1<=0) {
       trigger;
    } else {
       decrement counter1;
   }
}
```

Trigger the first time signal A remains high for at least 5 consecutive FCLK cycles.



2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Dynamic RTL – DRTL

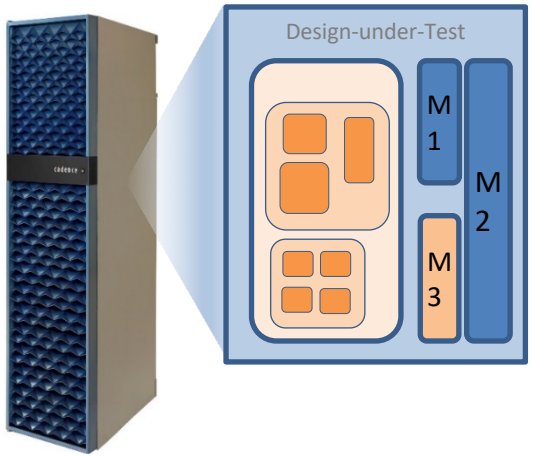
Alternative and Complement to SDL

- New runtime monitor functionality
  - Constructed using standard Verilog/VHDL RTL design
  - Loaded and instantiated at runtime. Fully dynamic and independent of compile.
  - Can monitor, display, trigger and provide runtime control
- Advantages of DRTL
  - Code complex monitors with state machines in a standard RTL language (Verilog or VHDL)
  - Able to Save and Load DRTL from precompiled files
    - This allows the creation of standard libraries of DRTL monitors
  - Flexible, single module can be instantiated multiple times
    - User only needs to instantiate the DRTL module and connect to the signals of interest
- Complements SDL
  - Easier to write complex logic and state machines
  - Optionally interacts with SDL to provide control of the runtime session

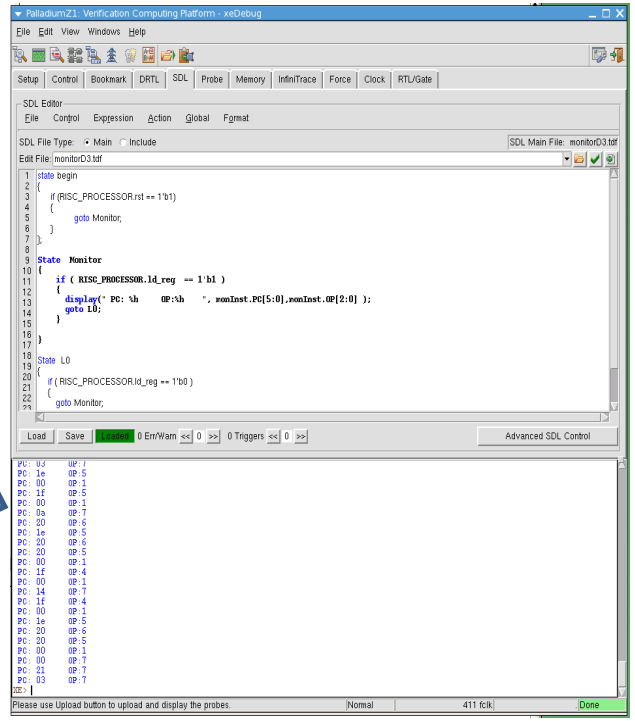
accellera SYSTEMS INITIATIVE

DRTL Independently Controlling and Monitoring

$display and $qel used within the DRTL module




```
module riscMon(clk, rst, data, ld);
  input clk, rst;
  input [8:0] data;
  input       ld;
…..
…..

always @(posedge clk or negedge rst)
 begin
  if (rst == 1'b0 )
    currentState <= STATE_INIT;
  else begin
   currentState <= nextState;
   $display (" PC: %h     OP:%h     ", PC[5:0], OP[2:0] );
   if (nextState == STATE_FINISH) begin
          $qel("trigger");
   end
  end
end

always @( * )
begin
  case (currentState)
   STATE_INIT: begin
     if ( ld == 1'b1) begin
        nextState =  STATE_LOAD;
     end
   end
  ……..
  ……..
```

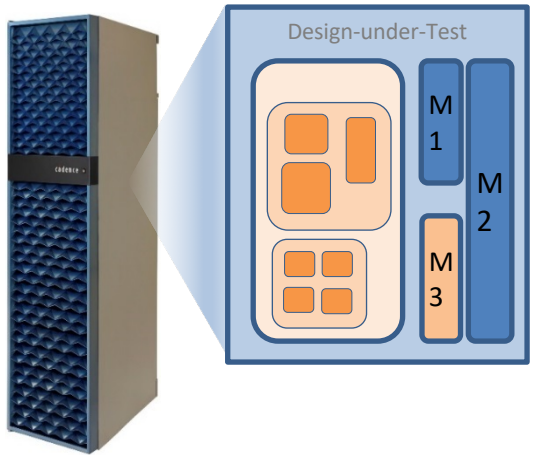


DRTL code

- Complex state machine monitoring
- Read-in / compiled at runtime (similar to SDL)
- $display used to print monitoring messages
- $qel used for control such as triggering

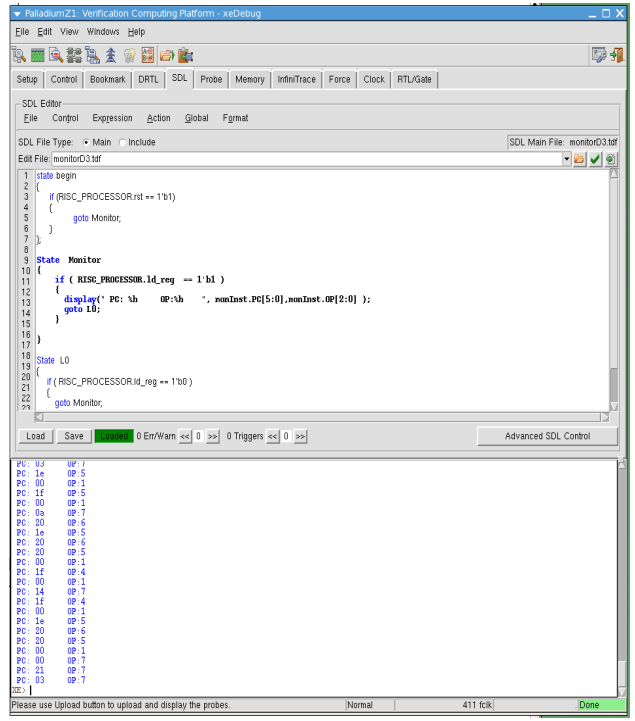
Dynamic RTL Complements SDL

Output ports available to SDL




```
module riscMon(clk, rst, data, ld, PC,OP);
  input clk, rst;
  input [8:0] data;
  input       ld;
  output [5:0] PC;
  output [2:0] OP;
 .....
 .....
  assign PC = rPC;
  assign OP = rOP;

always @(posedge clk or negedge rst)
  begin
    if (rst == 1'b0 )
      currentState <= STATE_INIT;
    else
      currentState <= nextState;
  end

always @( * )
begin
   case (currentState)
     STATE_INIT: begin
       if ( ld == 1'b1) begin
          nextState =  STATE_LOAD;
       end
     end
  ........
  ........
   endcase // case (currentState)
end
endmodule
```

```
state begin
{
    if (RISC_PROCESSOR.rst == 1'b1)
    {
       goto Monitor;
    }
};

State  Monitor
{
   if ( RISC_PROCESSOR.ld_reg  == 1'b1
)
{
    display(" PC: %h     OP:%h    ",
 monInst.PC[5:0],monInst.OP[2:0] );
    goto L0;
  }

}
```



DRTL code

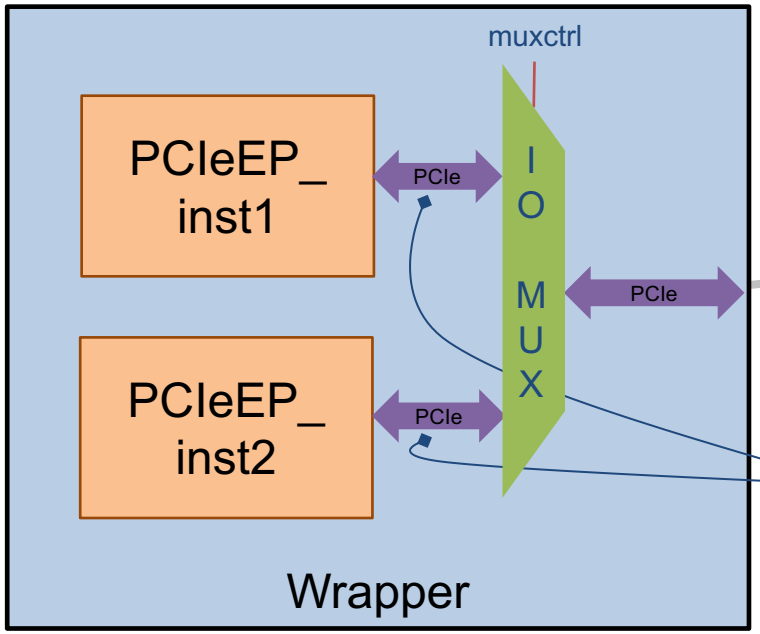
- Complex state machine monitoring
- Outputs available to SDL

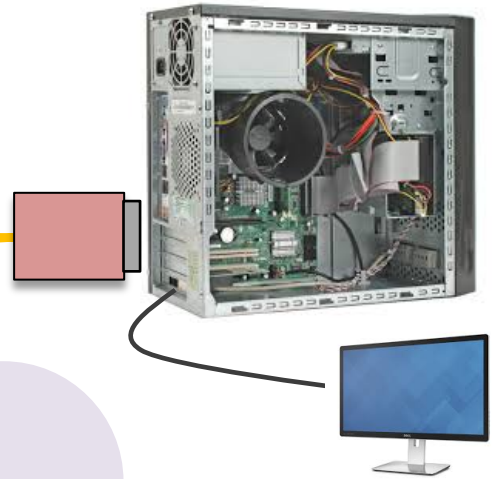
SDL code

- Existing control mechanism for emulator
- Accesses outputs from DRTL state machine

DRTL Usage Example

Monitoring a standard interface



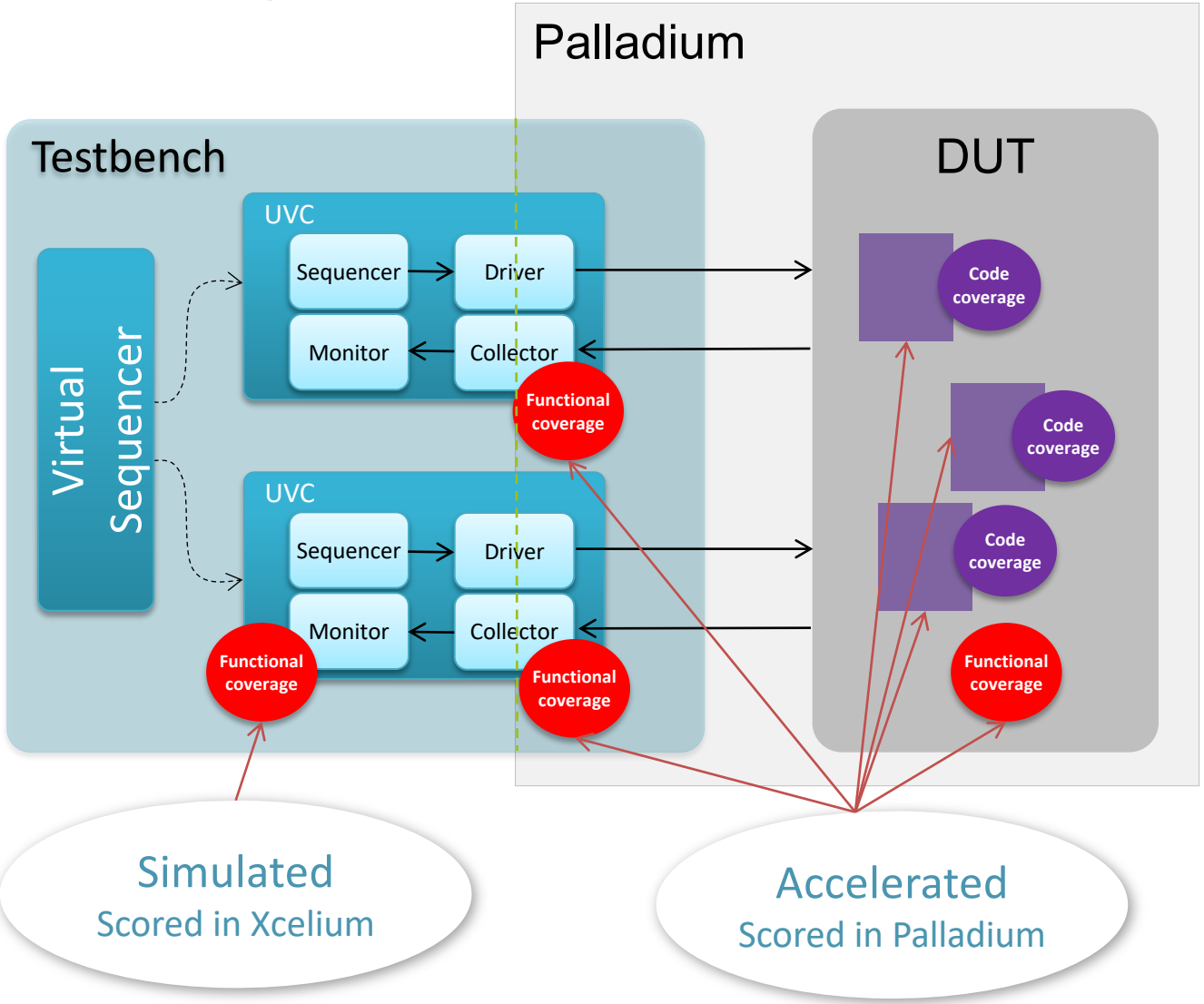


```
module PCIeMon (state_signal, signal_width, reset,clk);
input [7:0] state_signal;
input reset;
input [2:0] signal_width;
reg [7:0] state_signal_prev;
reg [3:0] state, next_state;
 input clk;

always@(posedge clk)
begin
state <= next_state;
state_signal_prev <= state_signal;
end

always@(*)
begin
   case(state)
       BEGIN: begin
                   $display("state = BEGIN");

                   next_state <= UPDATE;
                end
```

Two Instances of the DRTL PCIe Monitor

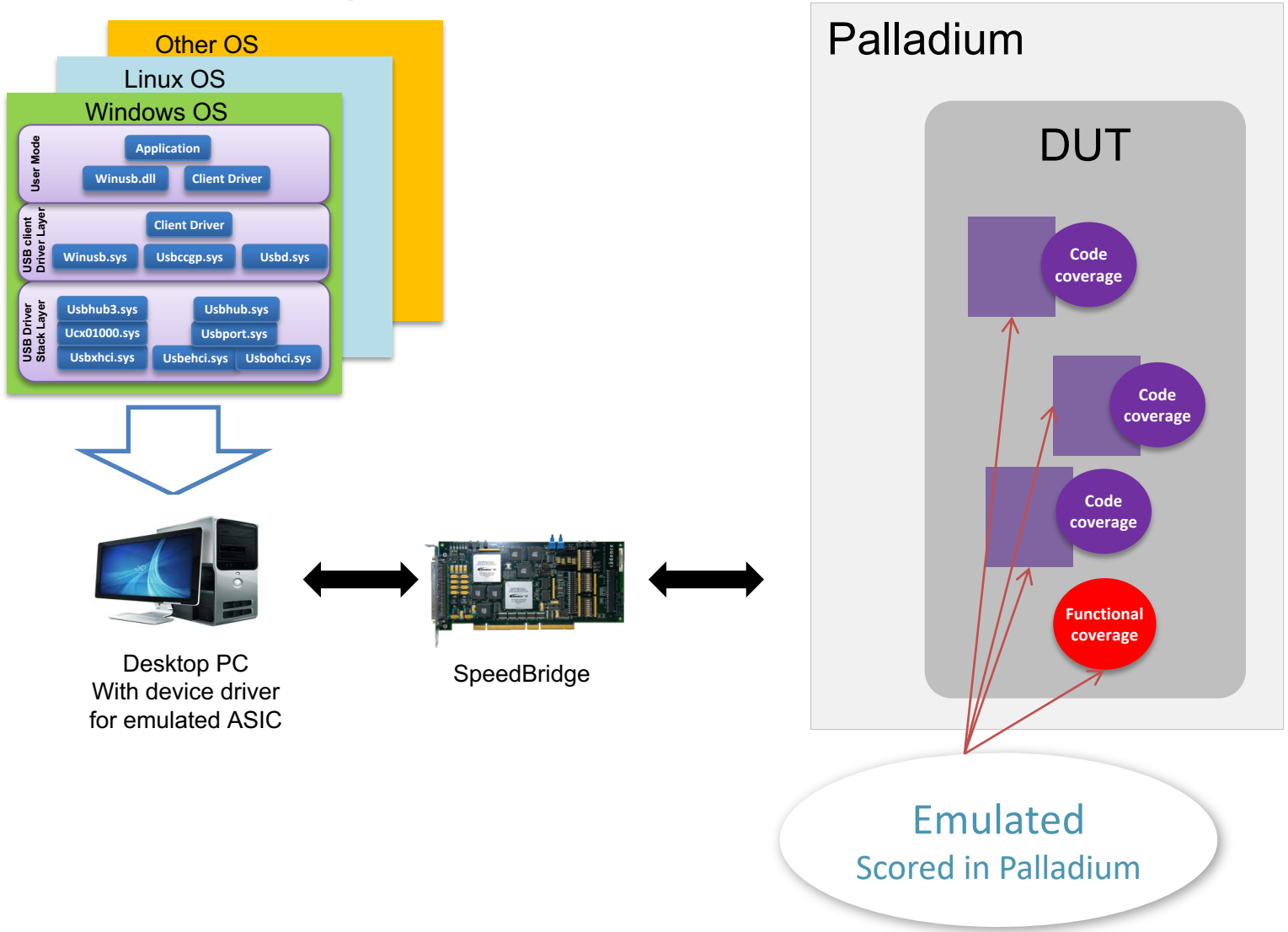
Data type: coverage example with Palladium

All coverage in simulator and Palladium is scored



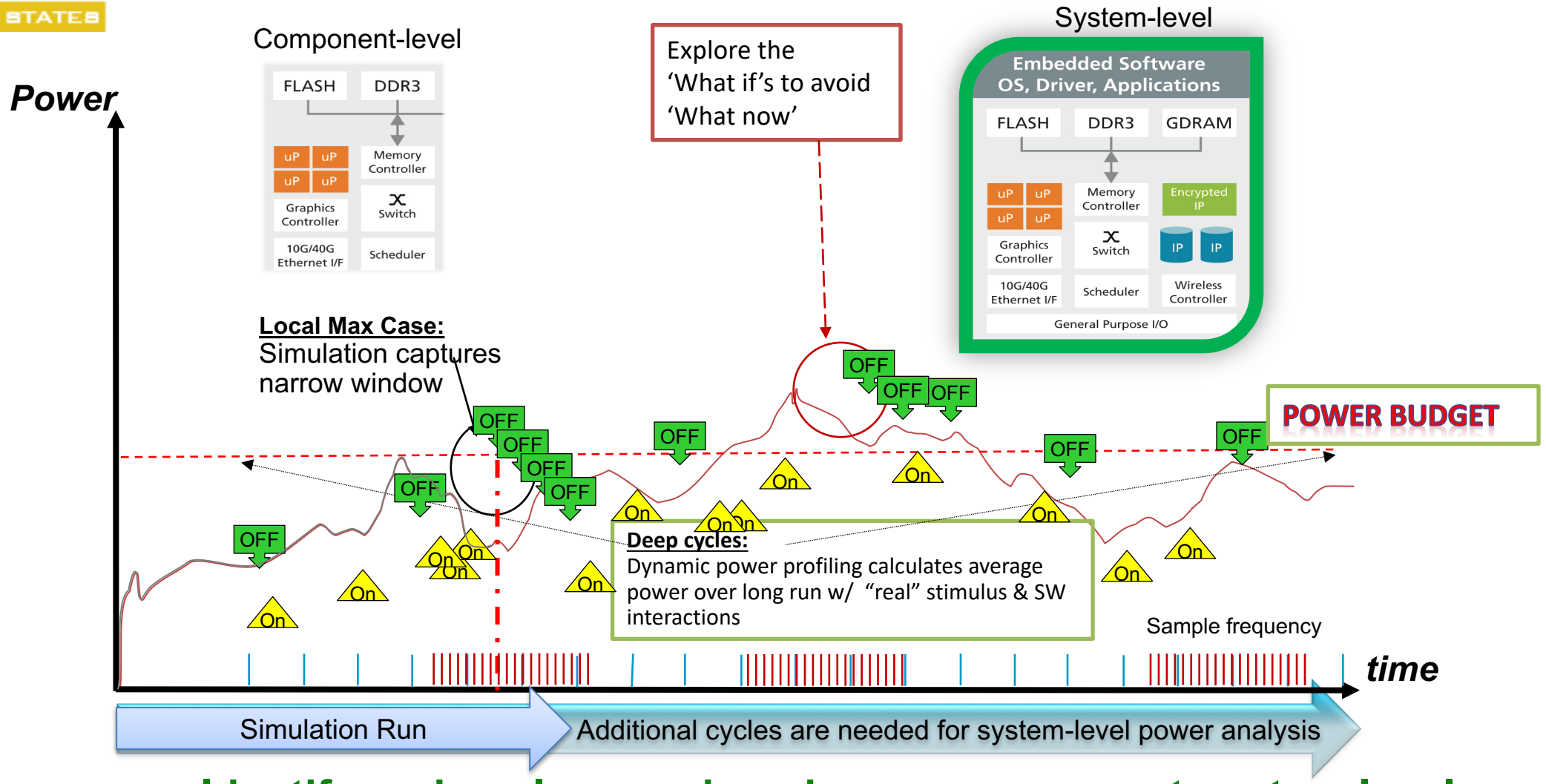
Data type: coverage example with Palladium

All coverage in Palladium is scored in emulation as well



SoC Power Analysis Requires "*Deep*" Cycles

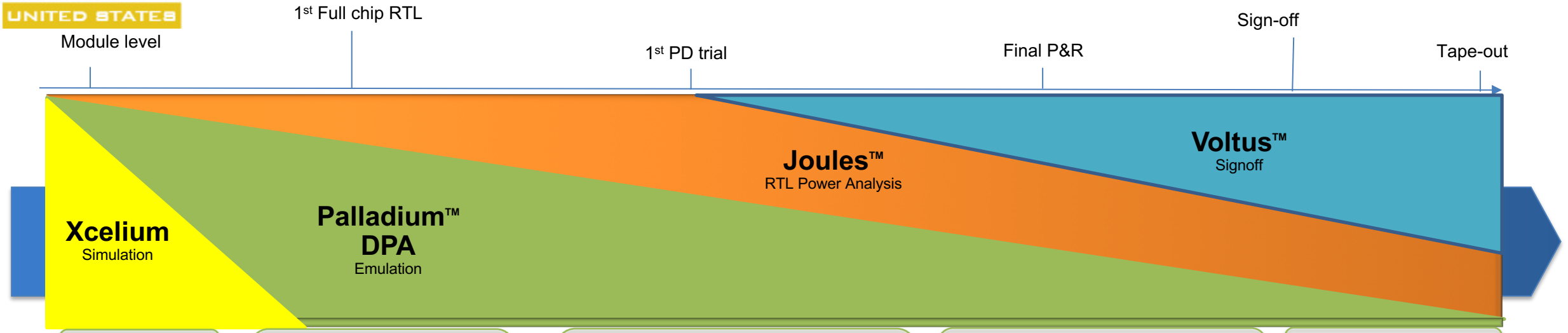
@100MHz for 10 secs → 1 Billion cycles



Identify and analyze peak and average power at system level

OFF On **Indicates Block turns on/off**

Data-driven emulation example: Power analysis



Power reduction

Power reduction ASIC-level
Start ave power estimation
Start peak estimation
High correlation

Add placement information to power estimation.
Power reduction over RTL and GTL
better peak estimation
Initial testing of signoff flow

Accurate placement information to power estimation. Higher accuracy.
Power reduction over GTL.
Pattern selection for signoff

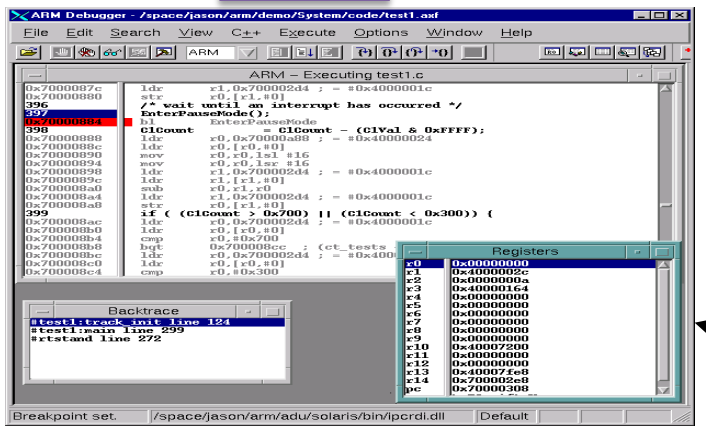
Final power estimation.
Signoff

Data-driven workload can be leveraged to extract power profile: average and peak power

	Power Info	Inputs	Work model
Palladium DPA	Toggles	RTL or Gates	• Power Analysis (per hierarchy / time) • Peak detection • Find window of interest for other tool
Joules	Watts	RTL or Gates	• RTL Power Analysis and Optimization • Power estimation
Voltus	Watts	Gates	• Power estimation and power Integrity • IR-drop and final Signoff

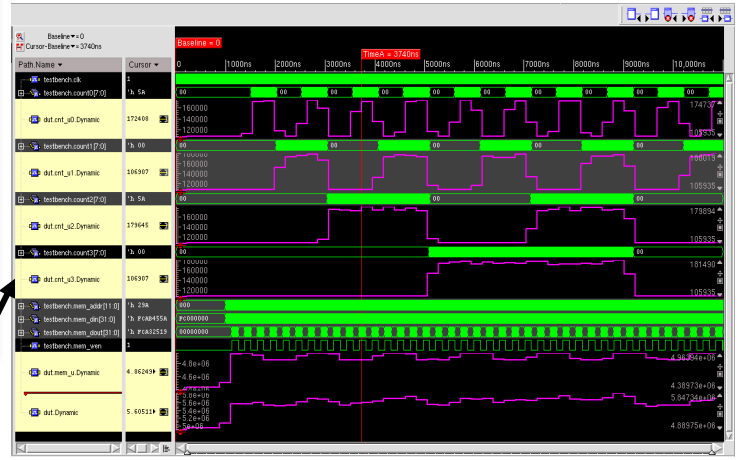
Summary: Data driven emulation enables system-level analysis

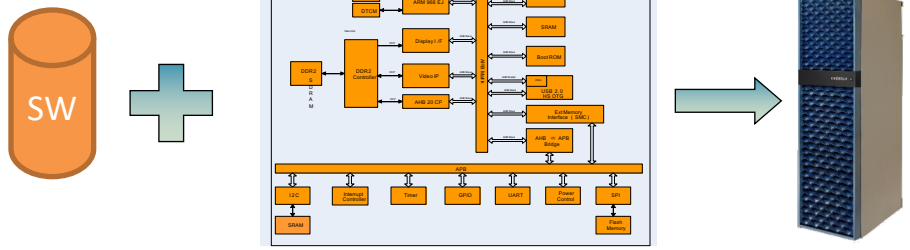
Software Debugger



Hardware Debugger

DPA option Power Profiler





Palladium Series
High-performance verification platform from RTL acceleration to system emulation

Palladium enterprise emulation platform excels with early HW/SW integration and co-verification with power analysis at the system-level

Hanan Moller, Systems Architect, UltraSoC

POST-SILICON AND IN-LIFE ANALYTICS IN HETEROGENOUS SOCS

Problem statements

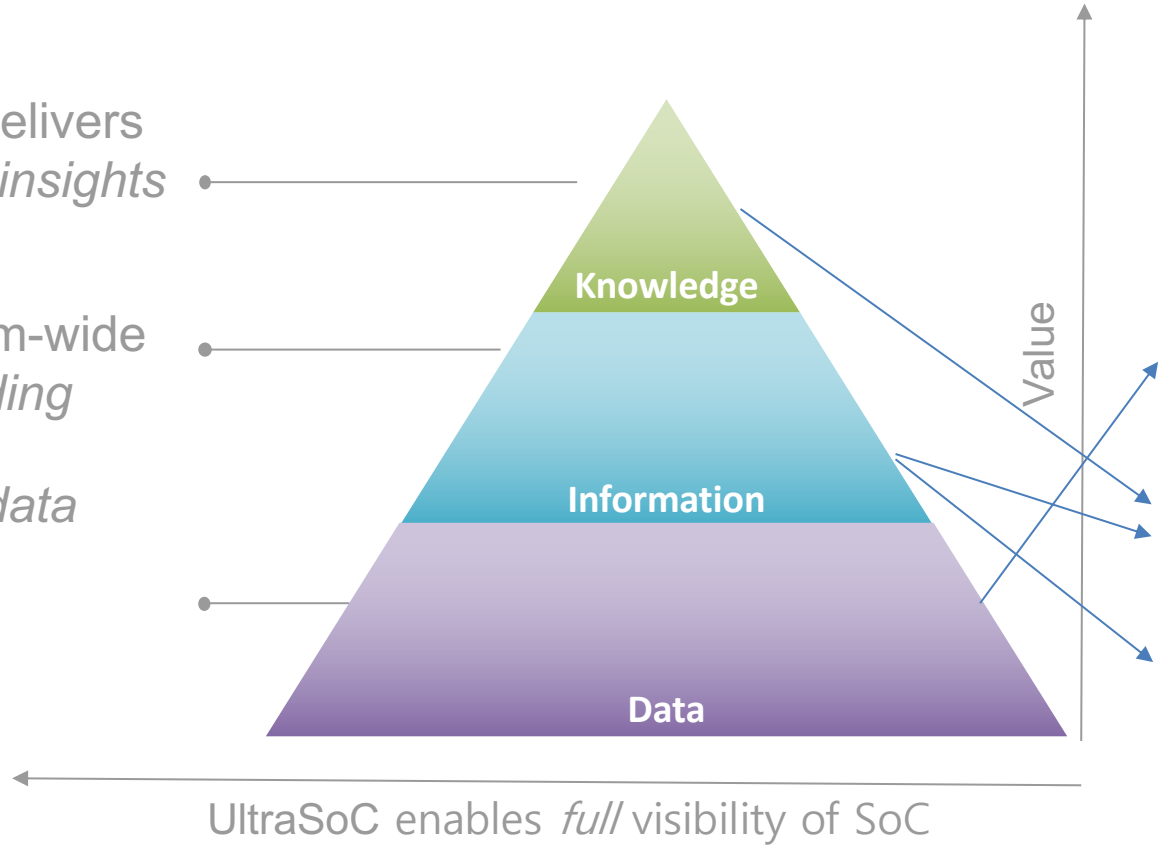
- It is not about the ISA(s)
- It is not about the core(s)
  - Compute is largely ‘solved’
- The challenge today is systemic complexity, for example:
  - Ad-hoc programming paradigms
  - Processor-processor interactions
  - HW/SW interactions
  - Interconnect, NoC & deadlock
  - System are informally architected
  - Workload details unknown in advance
  - Massive data

UltraSoC Distills Insights from Data

UltraSoC delivers actionable *insights*

With system-wide *understanding*

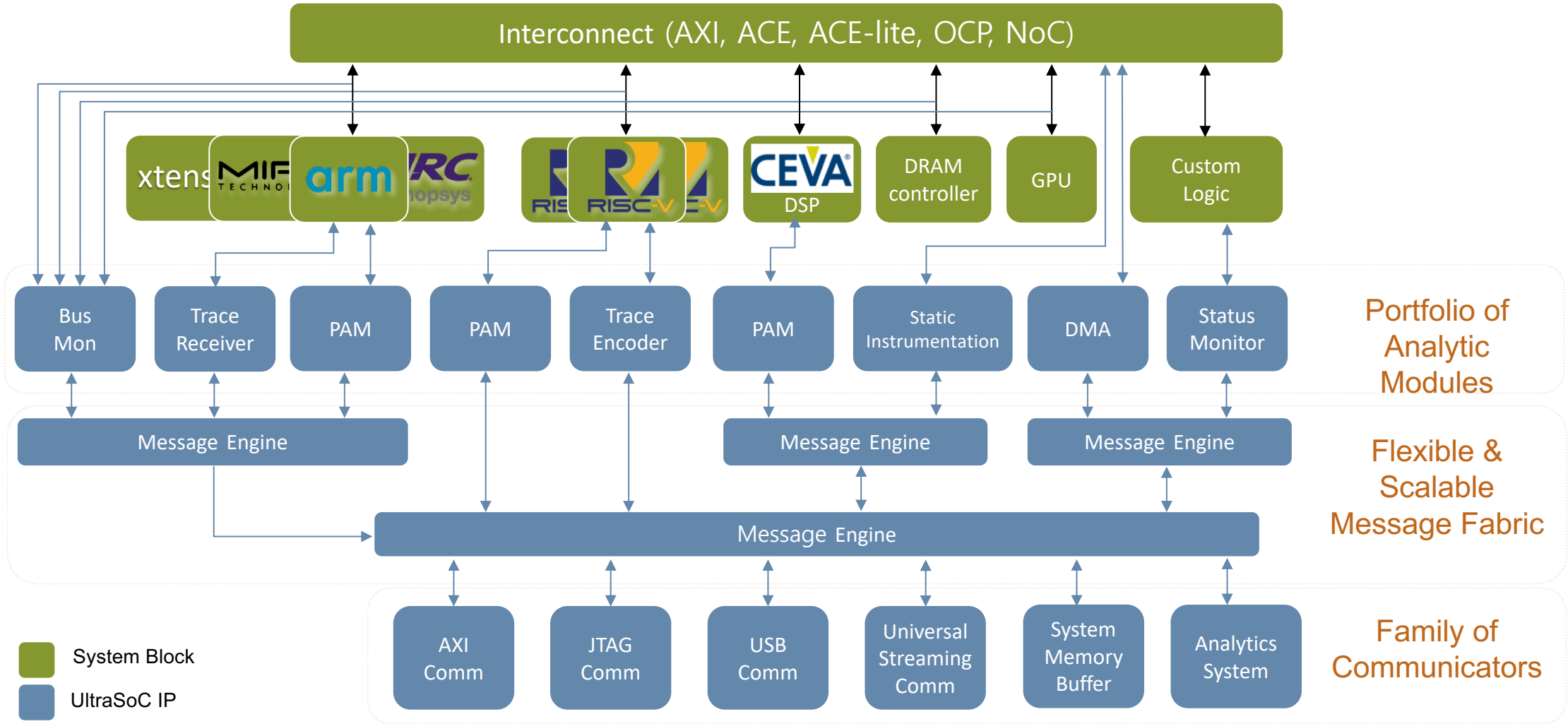
From rich *data* across the whole SoC



Use-case-based	•Define legal operations •Workload matters: must represent real operation
Data Collection	•Non-intrusive data collection •Use the right execution platform
Analysis	•Correlate, filter, learn, predict •Anomaly detection
Goal-based	•Verification throughput •Smarter bug hunting

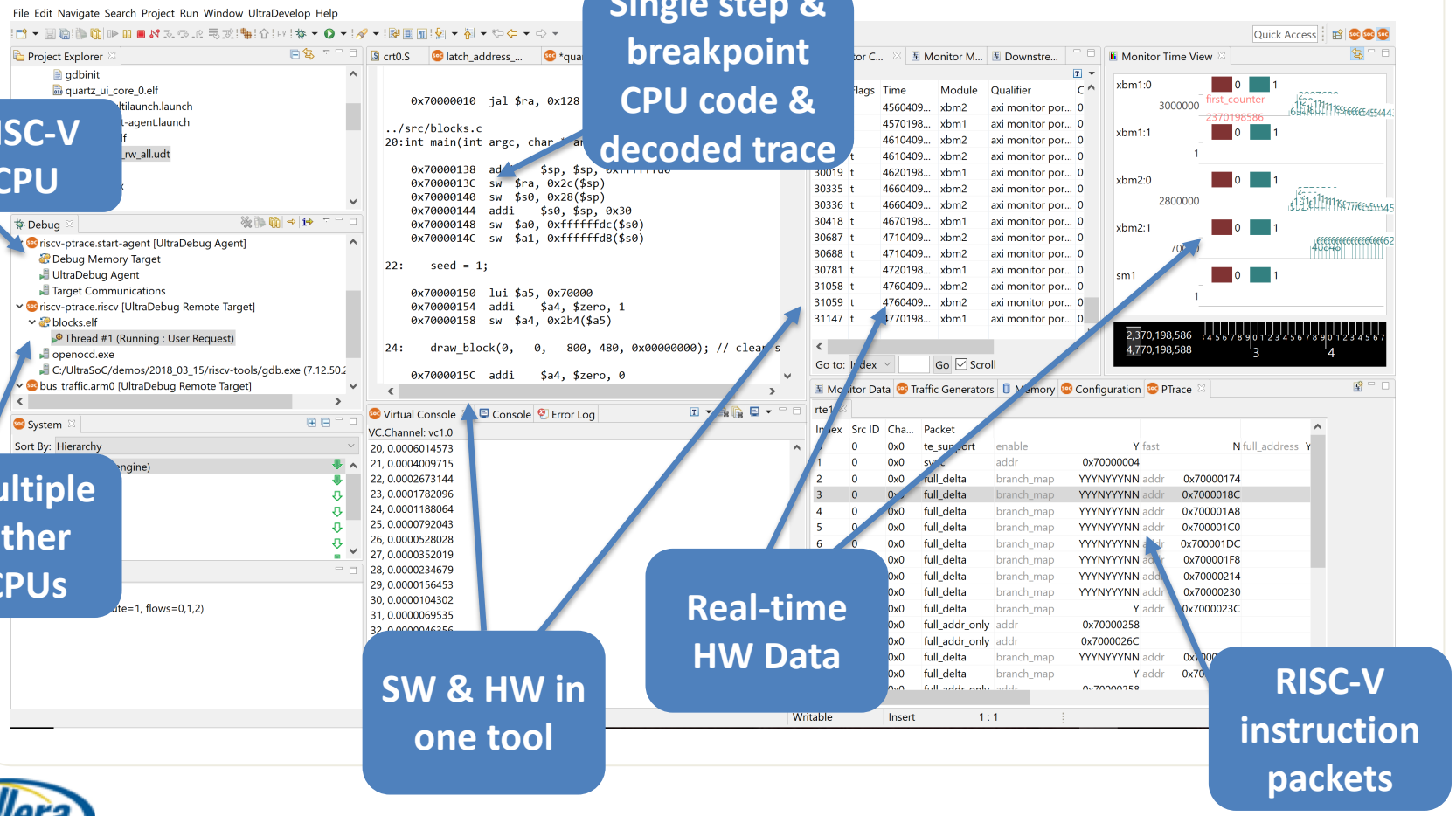
Actionable Analytics from any Chip for performance, safety, cyber-security

Advanced Debug/Monitoring for the Whole SoC

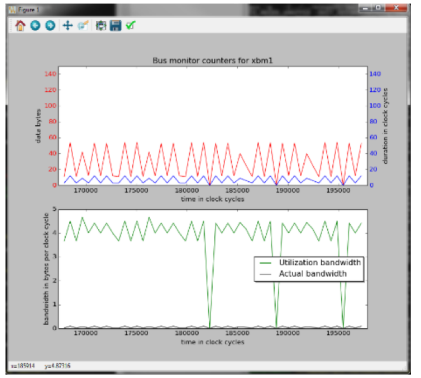


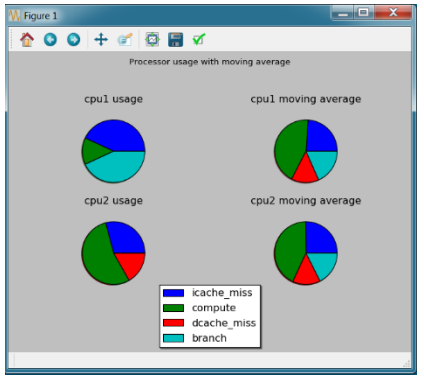
Software tools for data-driven insights

Eclipse based UltraDevelop IDE



Script based

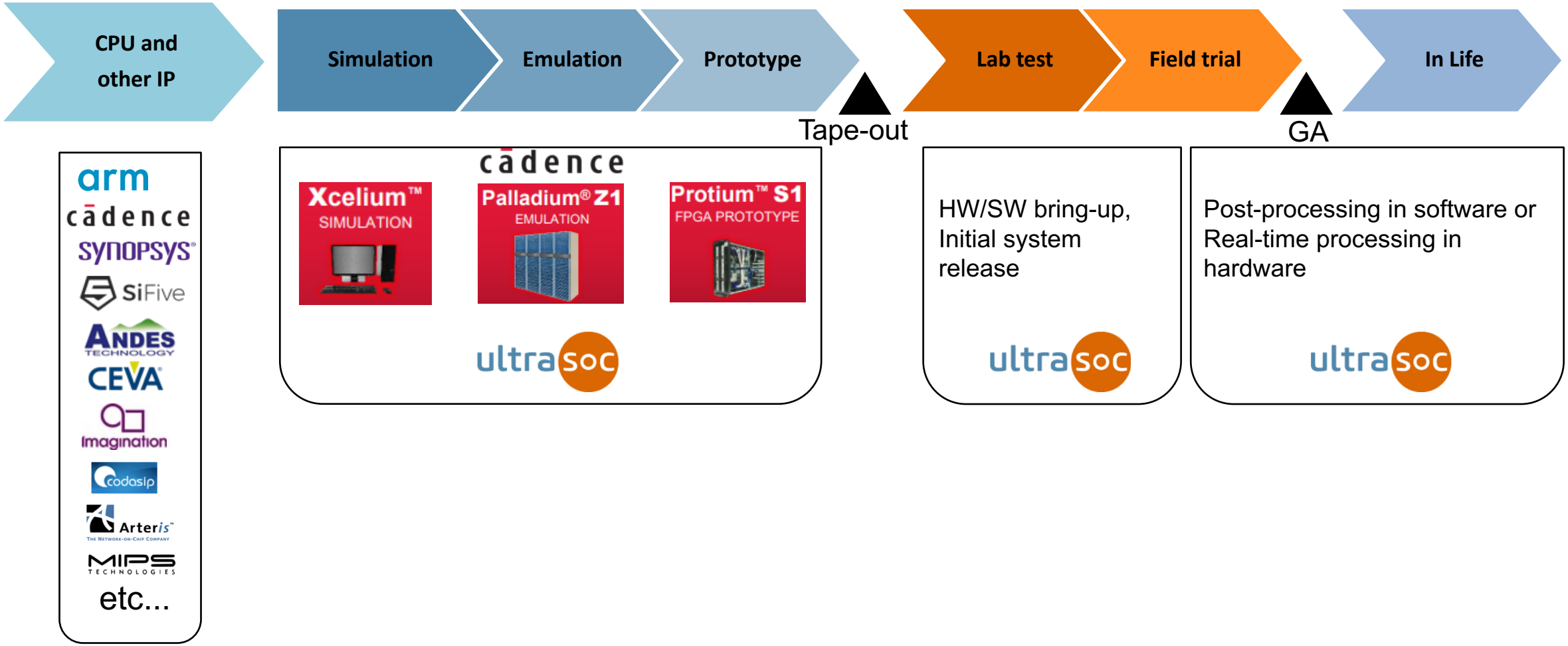




UltraSoC

- A coherent architecture to debug, monitor and provide rich data for run-time analytics
  - RTL IP is highly parameterizable - allows customers to trade hardware resources and thus silicon area
  - Hardware resources are configurable at runtime
  - Allows reuse of hardware resources for different scenarios and different algorithms
  - Help with security and safety of systems
  - Hardware provides rich data so CPU load for analysis is small

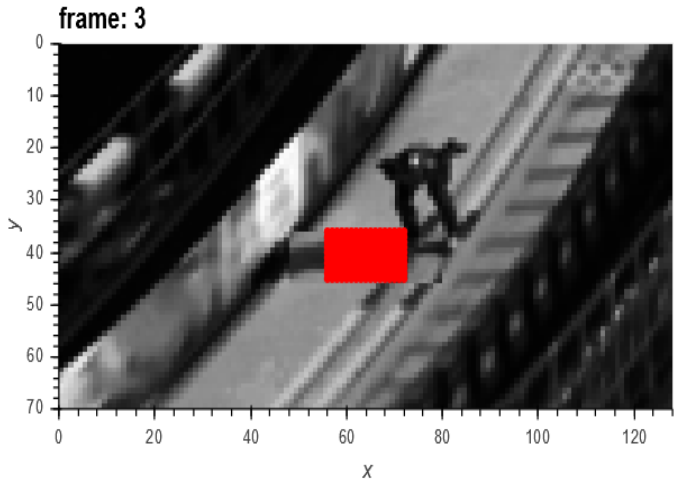
Analytics throughout Simulation→Emulation→In-Life



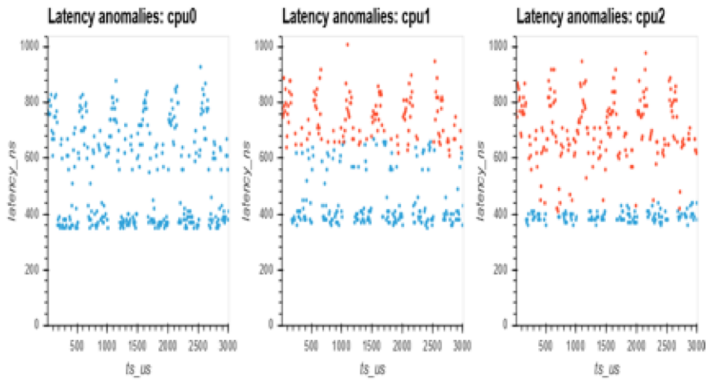
In-life Detection

Lab test → Field trial → In Life

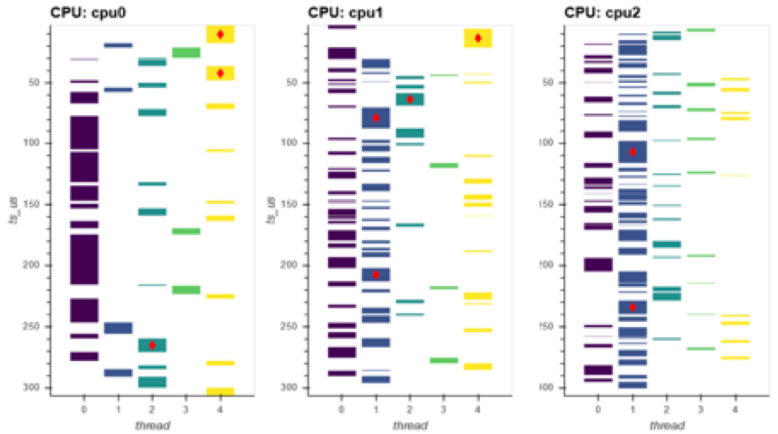
Safety
HW "stuck pixel" detection



Security
HW-based attack detection



Performance Optimization
Run-time server SW tuning / security

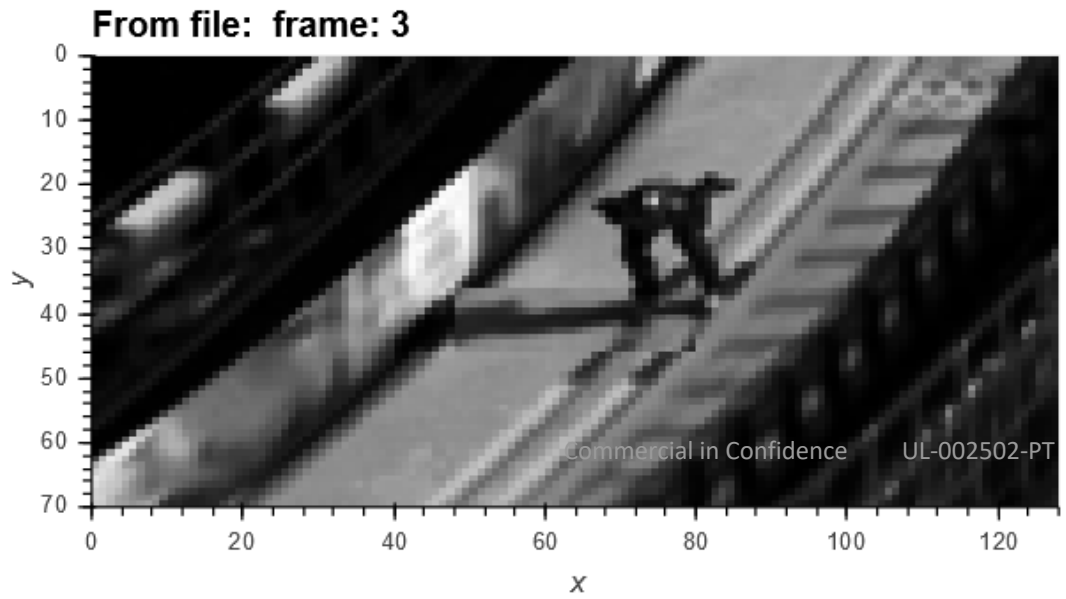


- **Non-intrusive:** No performance impact
- **Hardware:** Fast, react at HW timescale; invisible to software
- **Visibility:** Analyze software and system everywhere in SoC

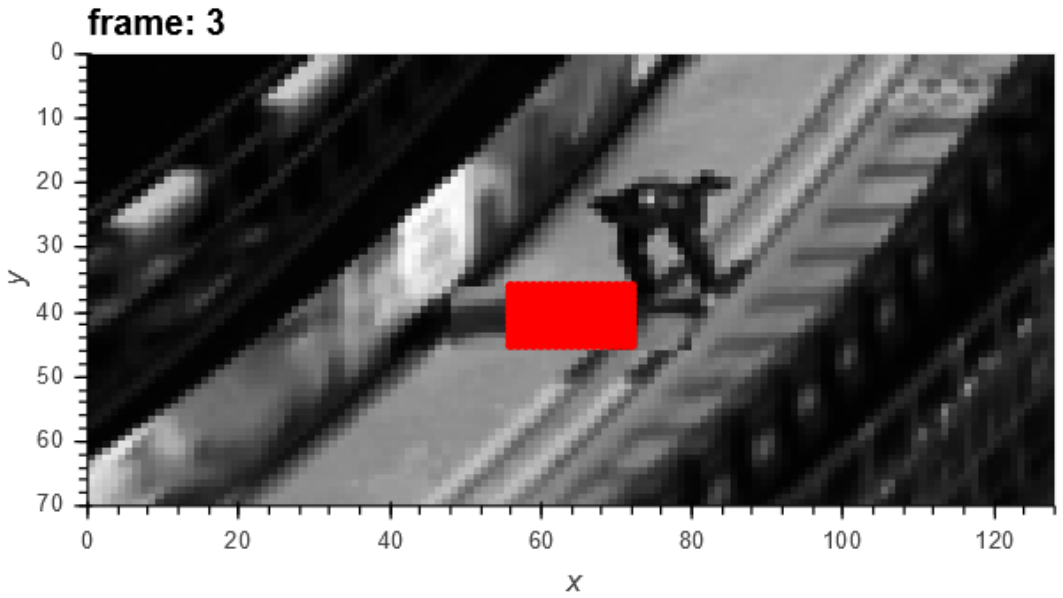
Non-intrusive stuck pixels detection

Fastest time to detection

Incoming image

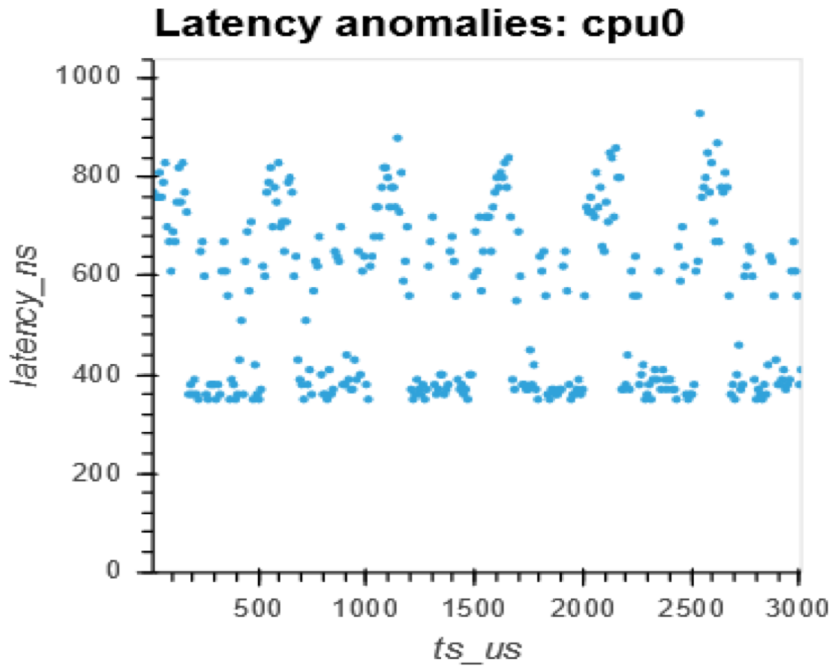


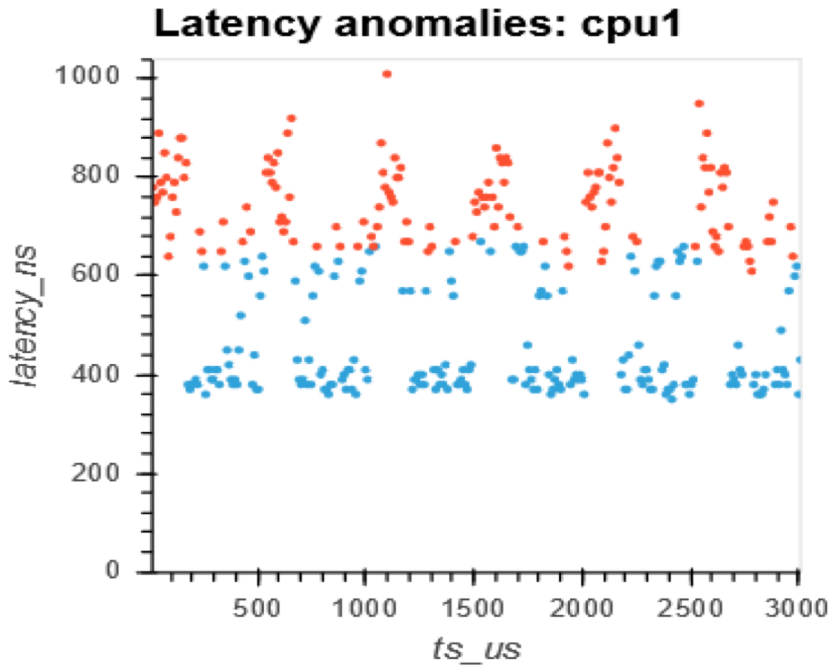
Detected stuck pixels

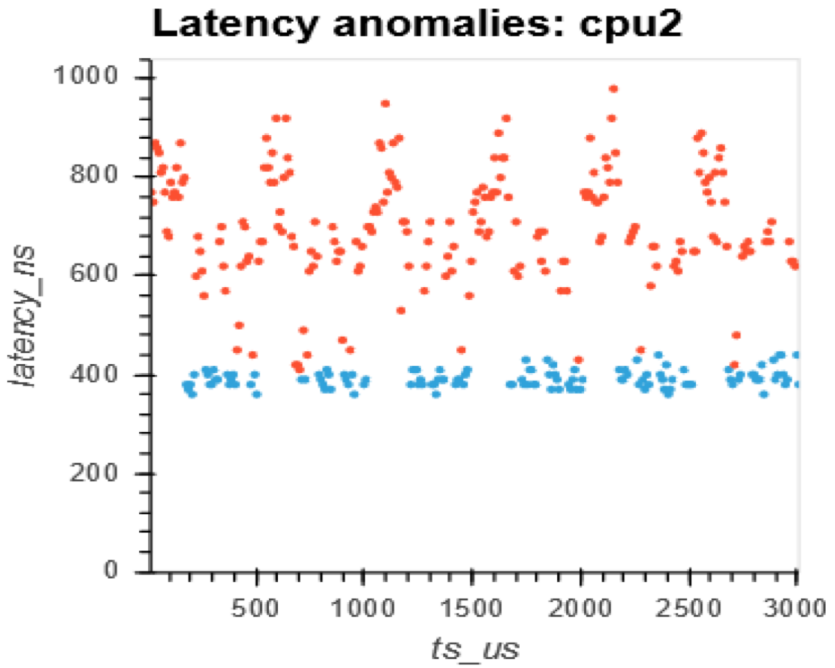


Non intrusive anomaly detection

- Three CPU plots below show CPU cache-like traffic for 3 CPUs configured with different miss rates
- Excessive (anomalous) latencies are shown in red

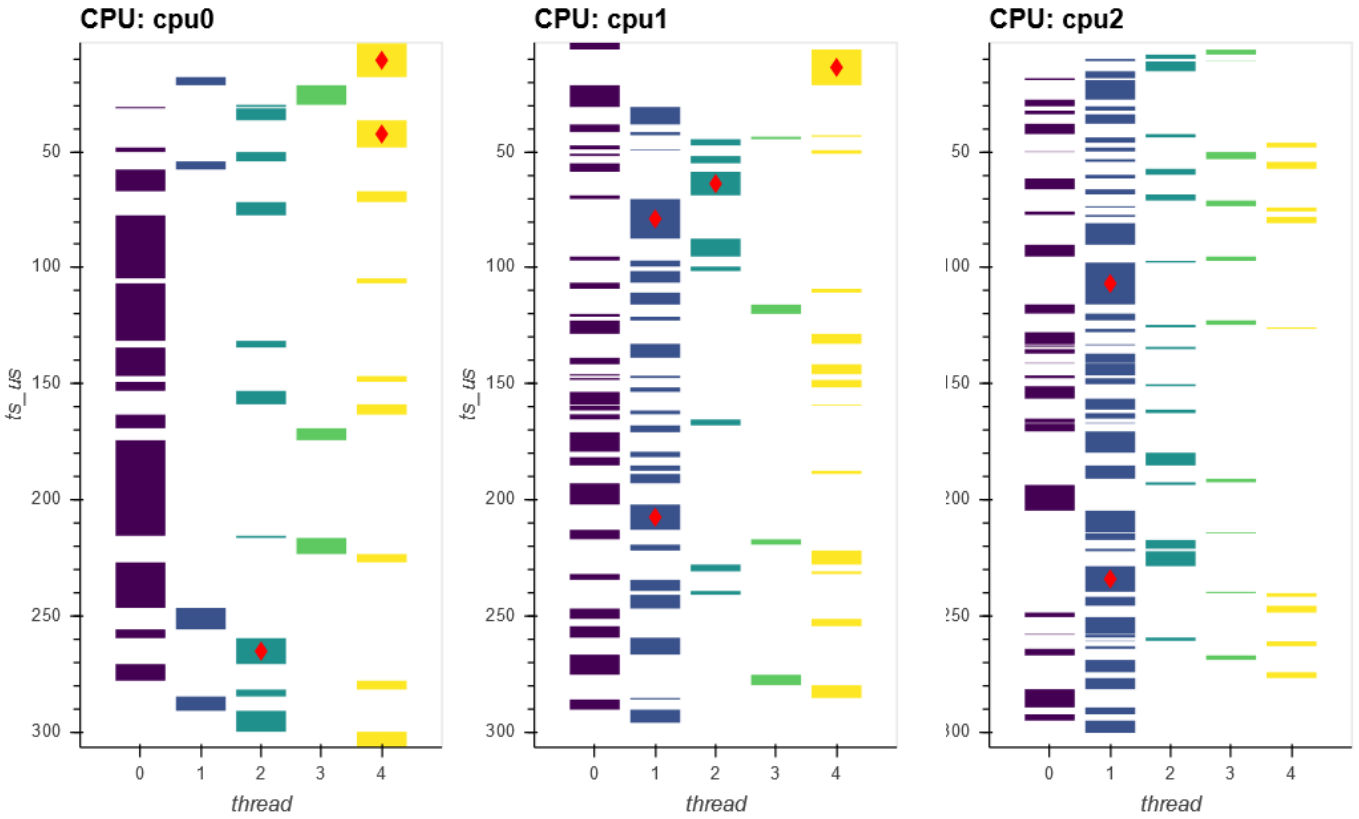






Non-intrusive profiling with anomaly detection

- Traditional profilers are inadequate:
  - Sampling = miss subtle or fast events (Nyquist)
  - Performance impact/intrusive
  - “Heisenbugs”
- UltraSoC is non-intrusive
- UltraSoC is wirespeed (100% coverage)
- Analytics and automated anomaly detection to make engineer more efficient



Summary

- The challenge today is systemic complexity
  - Architectural and modelling is needed but not enough
- Data analysis critical throughout product life-cycle
  - Focused, non-intrusive data collection
- Need tools that support heterogenous systems
- Complex systems may require autonomous analytics and causality detection in real-time

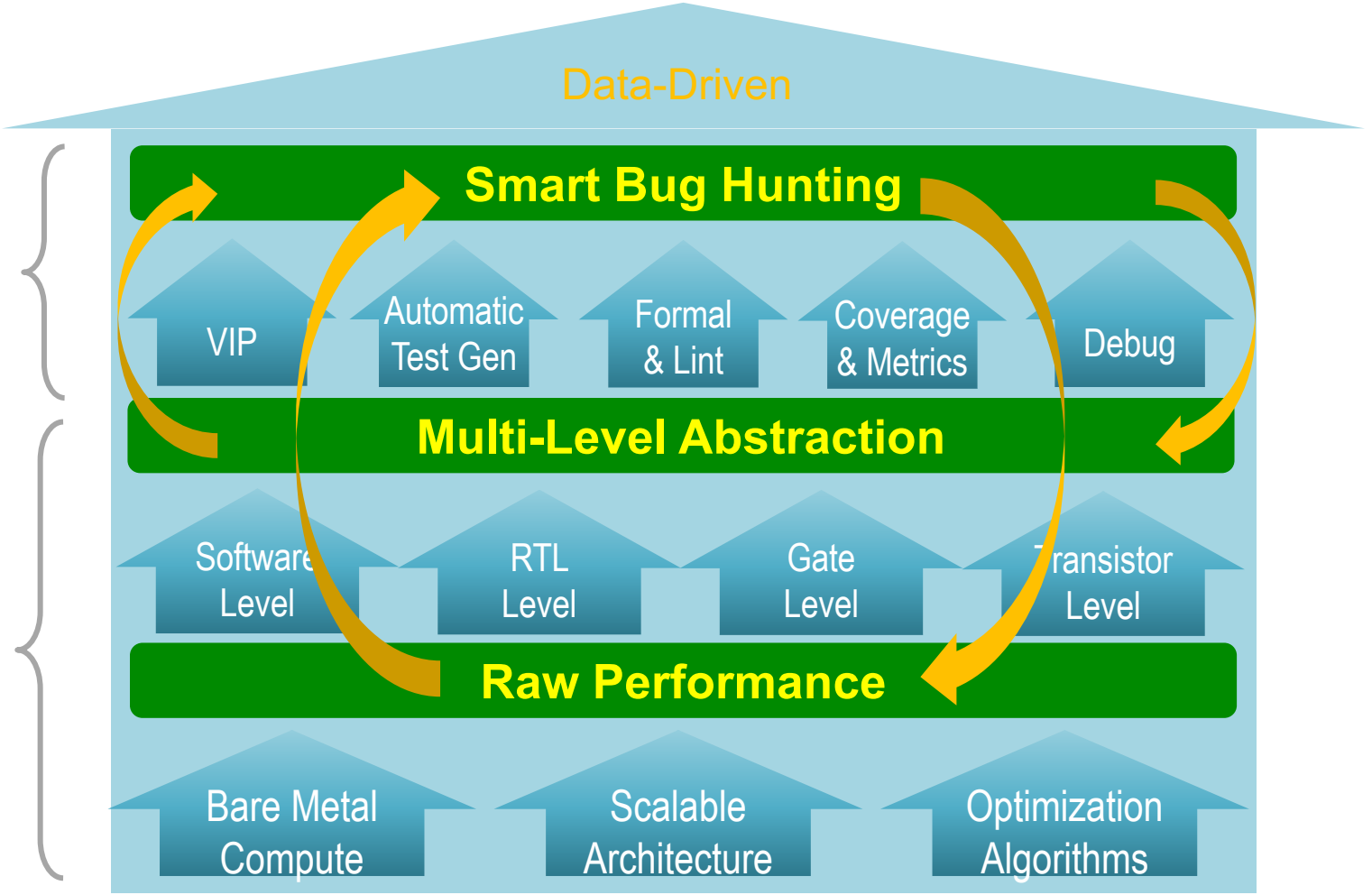
Data-Driven Verification

Use-case-based	• Define legal operations • Workload matters: must represent real operation
Data Collection	• Non-intrusive data collection • Use the right execution platform
Analysis	• Correlate, filter, learn, predict • Anomaly detection
Goal-based	• Verification throughput • Smarter bug hunting

VERIFICATION THROUGHPUT

Bugs per $ per day

Cycles per $ per day



2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Thank You!

- Q&A

accellera SYSTEMS INITIATIVE