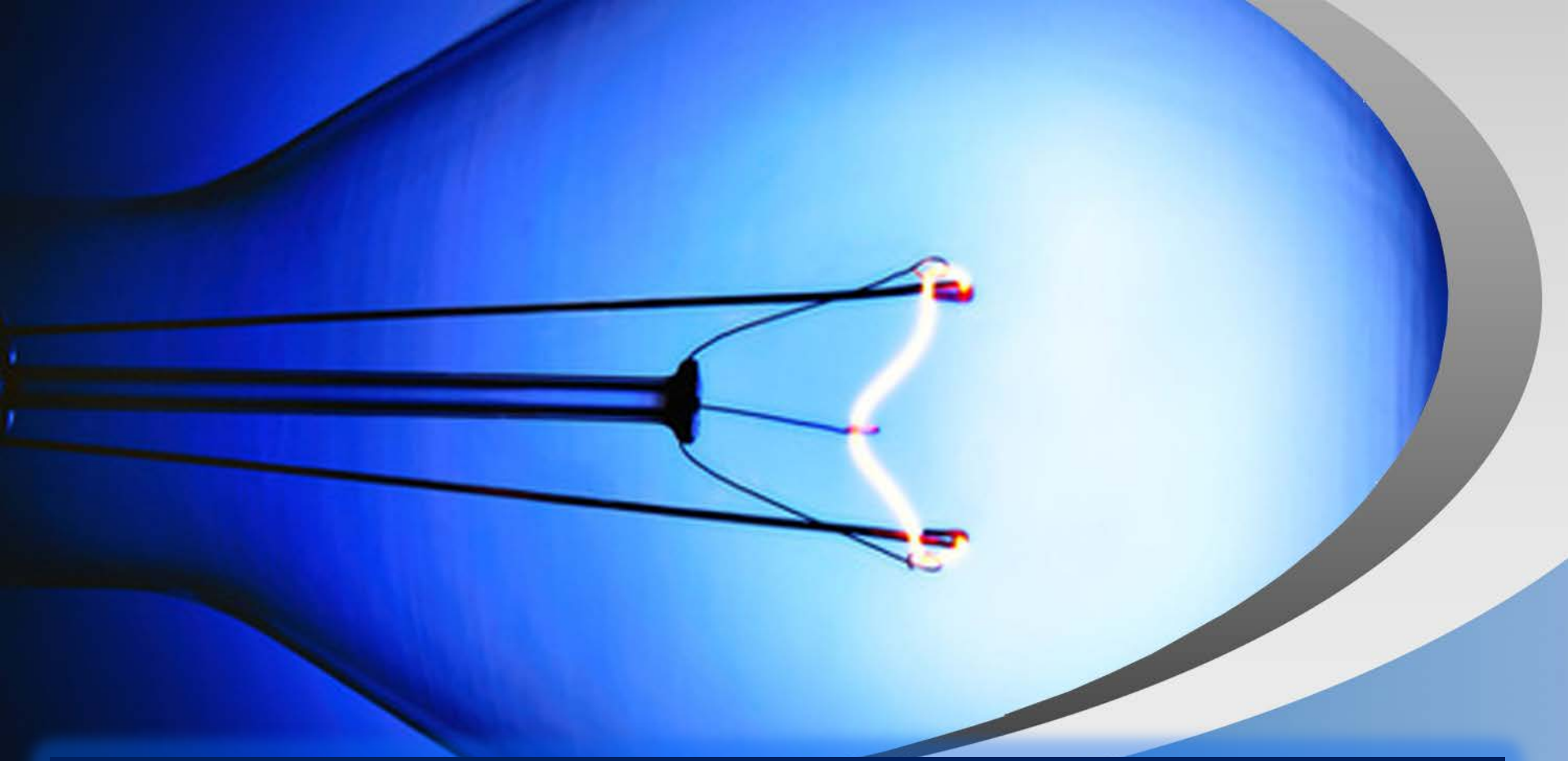


Next Generation Design and Verification Today

Requirements-driven Verification Methodology (for Standards Compliance)

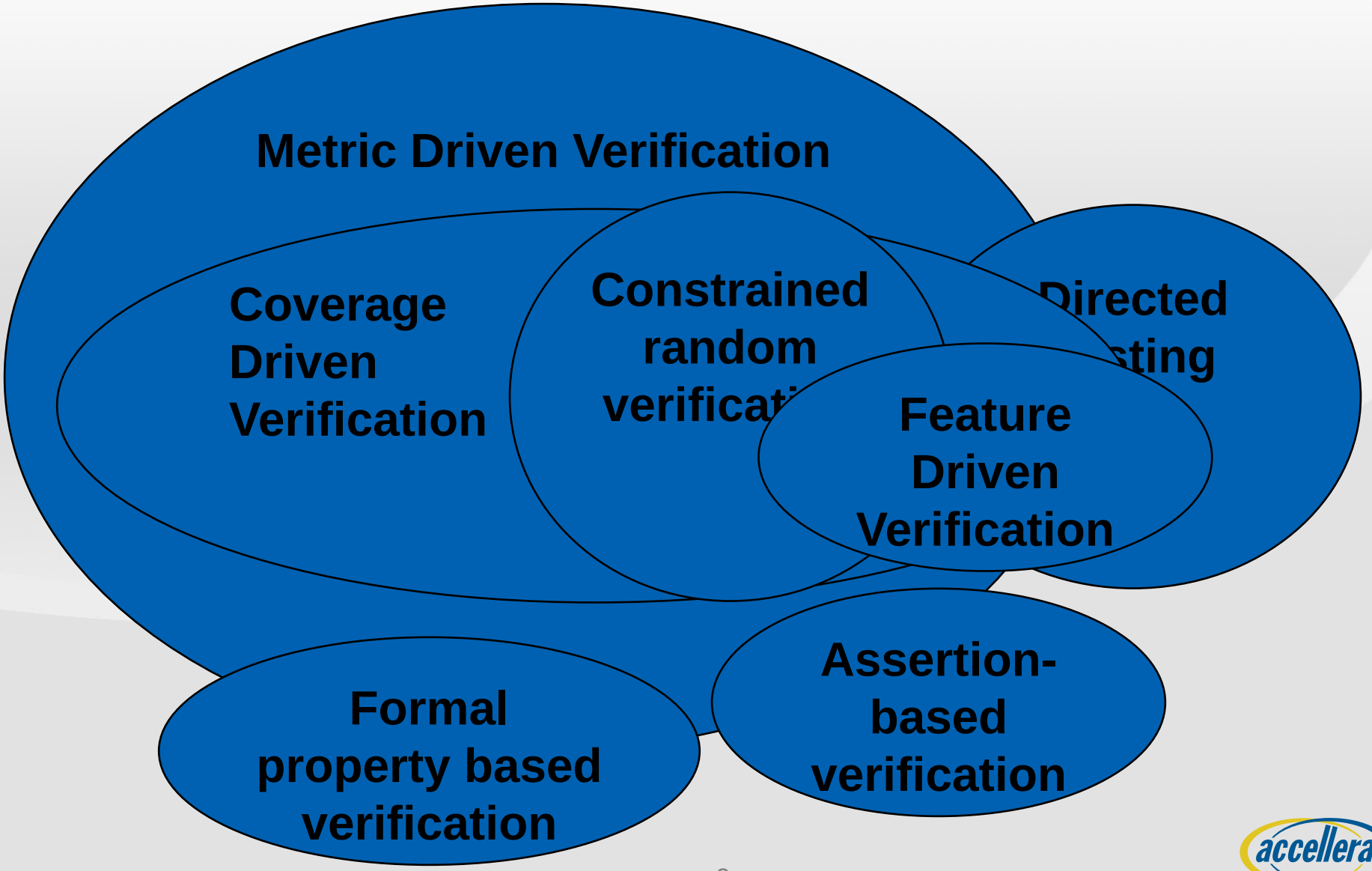
Mike Bartley, TVS

accellera SYSTEMS INITIATIVE

Agenda

- **Motivation**
  - Why Requirements Driven Verification?
- **Introduction to Safety**
  - The Safety Standards
  - What do we need to do? And deliver?
- **Supporting Requirements Driven Verification with Advanced Verification Techniques**
- **Tool Support**
- **Advantages of Requirements Driven Verification**

An Overview of Verification Approaches



accellera SYSTEMS INITIATIVE

Why Requirements Driven Verification?

- **Metric Driven Verification**
  - Allows us to define targets
  - And monitor progress

The metrics can become the end rather than the means to the end

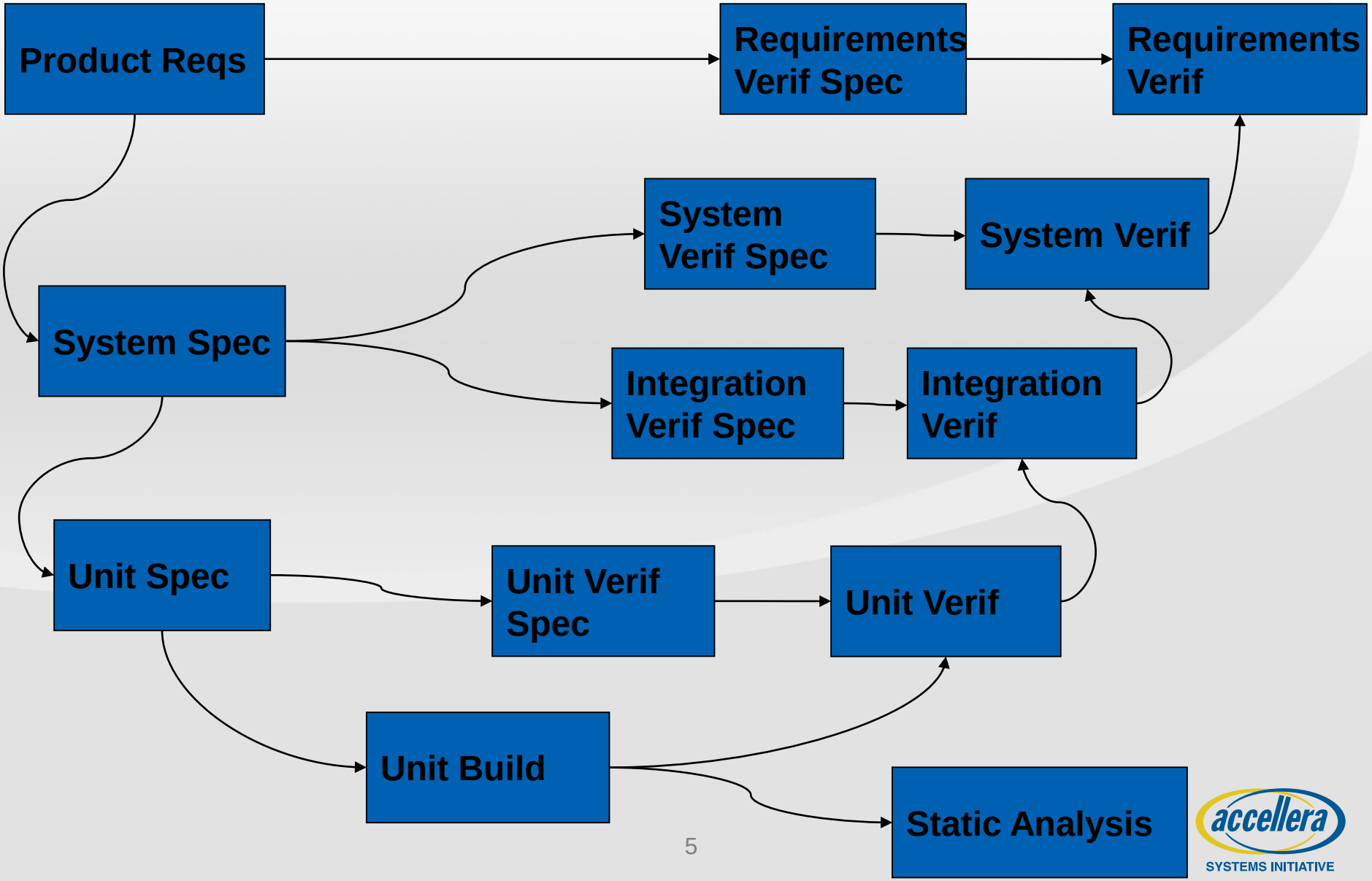
- **Coverage Driven Verification**
  - Most common metric driven verification approach
  - Code Coverage
  - Functional coverage
    - Might be related to features

How often have do you chase a coverage goal with limited ROI?

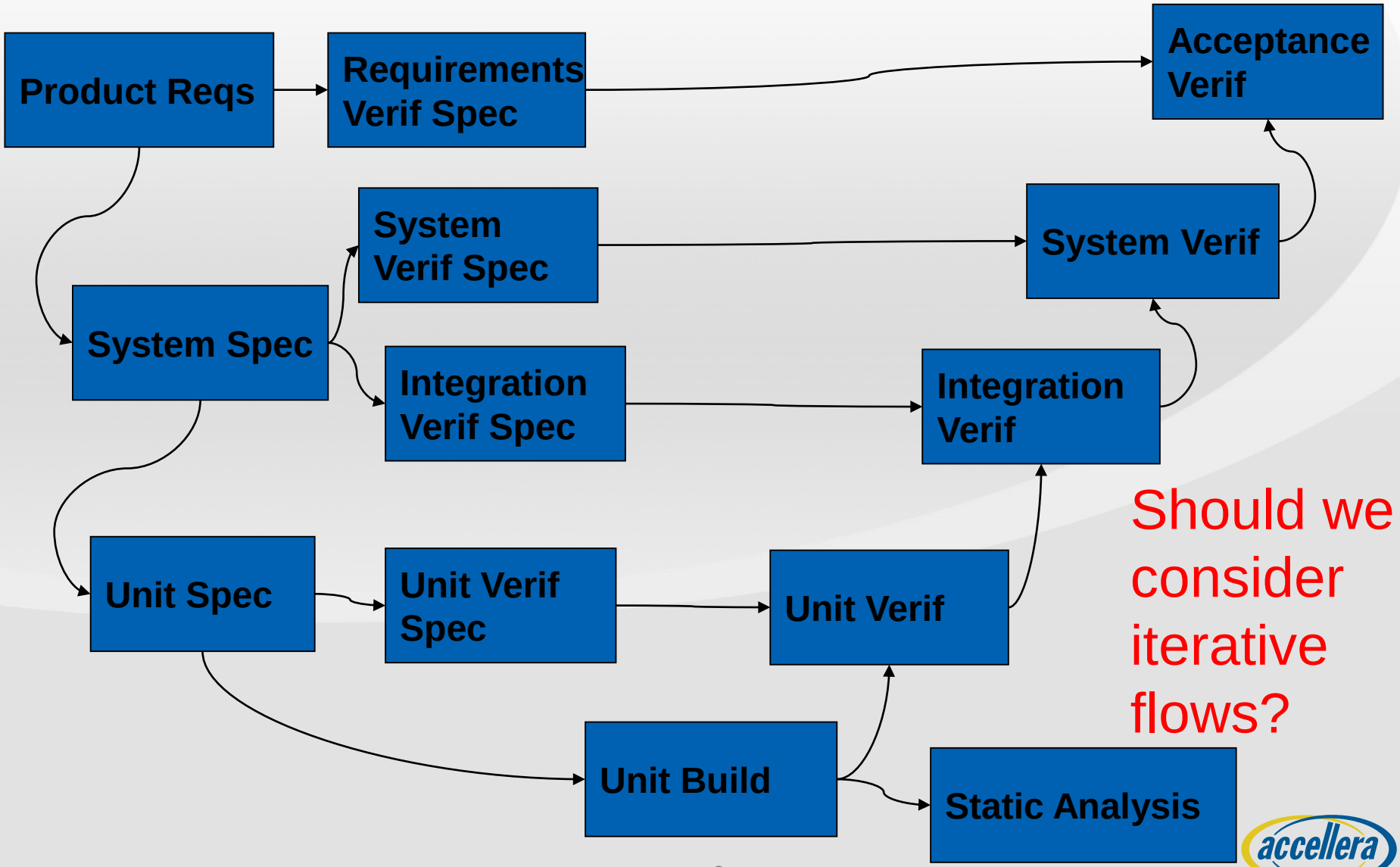
- **Feature Driven Verification**
  - Features MIGHT be related to spec
    - Is that relationship captured?
  - Are features related to requirements?

Shouldn't everything we do be related to a requirement?

Sequential Development Flow



Shift-Left “Sequential” Development Flow



accellera SYSTEMS INITIATIVE

Safety Standards

- **IEC61508:** Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems
- **DO254/DO178:** Hardware/Software considerations in airborne systems and equipment certification
- **EN50128:** Software for railway control and protection systems
- **IEC60880:** Software aspects for computer-based systems performing category A functions
- **IEC62304:** Medical device software -- Software life cycle processes
- **ISO26262:** Road vehicles – Functional safety

Introduction to Safety

- **The life cycle processes are identified**
- **Objectives and outputs for each process are described**
  - Objectives are mandatory
  - But vary by Integrity Level
  - For higher Integrity Levels, some Objectives require Independence

Key Elements

- **Plans & Standards**
- **Requirements**
- **Design Specifications**
- **Reviews and Analyses**
- **Testing (against specifications)**
  - At different levels of hierarchy
  - Test Coverage Criteria
  - Requirements Traceability
  - Independence

Key Deliverables

- Hardware Verification Plan
- Validation and Verification Standards
- Hardware Traceability Data
- Hardware Review and Analysis Procedures
- Hardware Review and Analysis Results
- Hardware Test Procedures
- Hardware Test Results
- Hardware Acceptance Test Criteria
- Problem Reports
- Hardware Configuration Management Records
- Hardware Process Assurance Records

Requirements Engineering Definitions

Requirement:

1. A condition or capability needed by a user to solve a problem or achieve an objective
2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed documents
3. A documented representation of a condition or capability as in (1) or (2)

[IEEE Std.610.12-1990]

Stakeholder*:

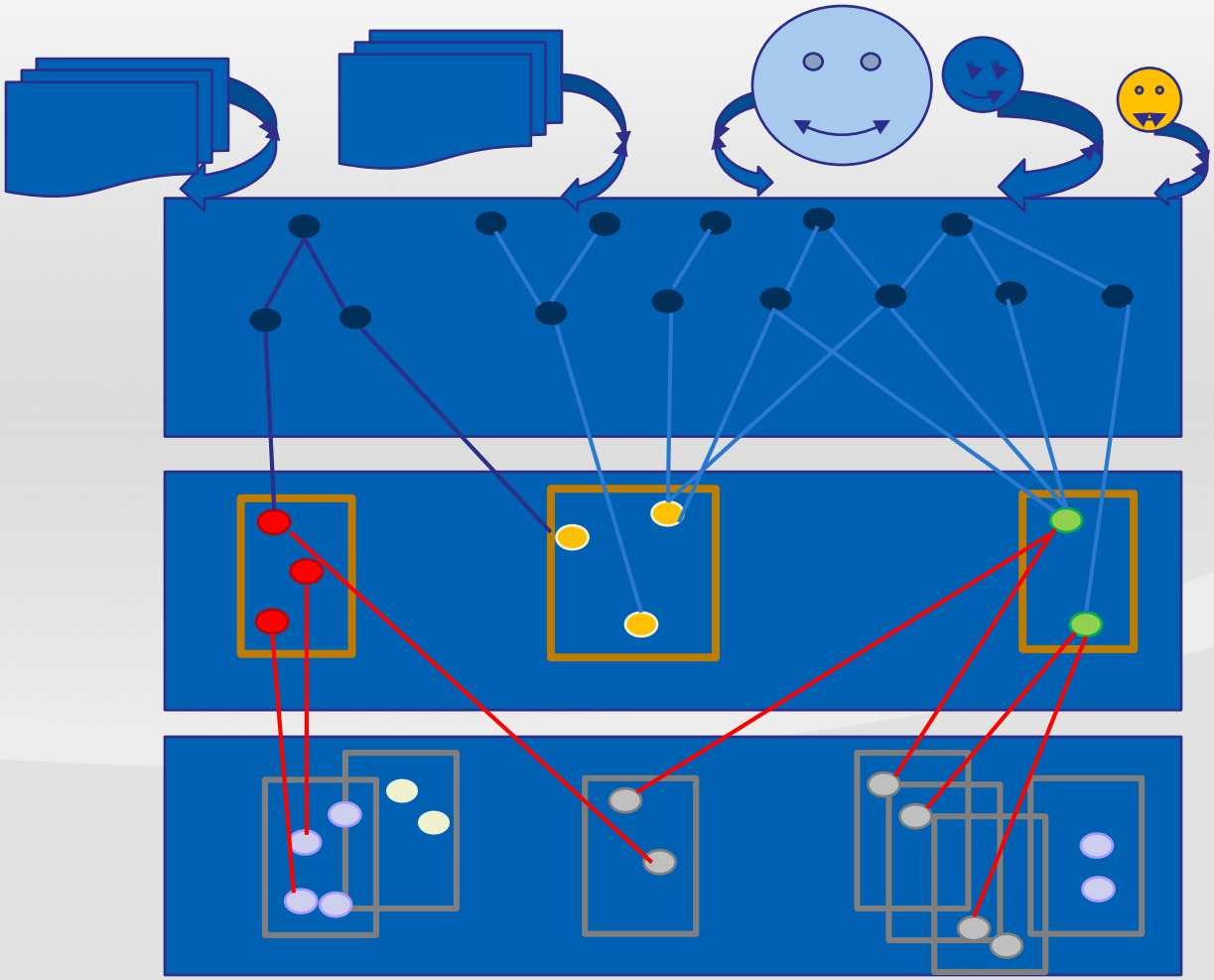
- A stakeholder of a system is a person or an organization that has an (direct or indirect) influence on the requirements of the system

Requirements Engineering:

- Requirements engineering is a systematic and disciplined approach to the specification and management of requirements with the following goals:
  1. Knowing the relevant requirements, achieving a consensus among the Stakeholders about these requirements, documenting them according to given standards, and managing them systematically
  2. Understanding and documenting the stakeholders' desires and needs, then specifying and managing requirements to minimize the risk of delivering a system that does not meet the stakeholders' desires and needs

*** All Definitions taken from IREB**

Requirements Engineering



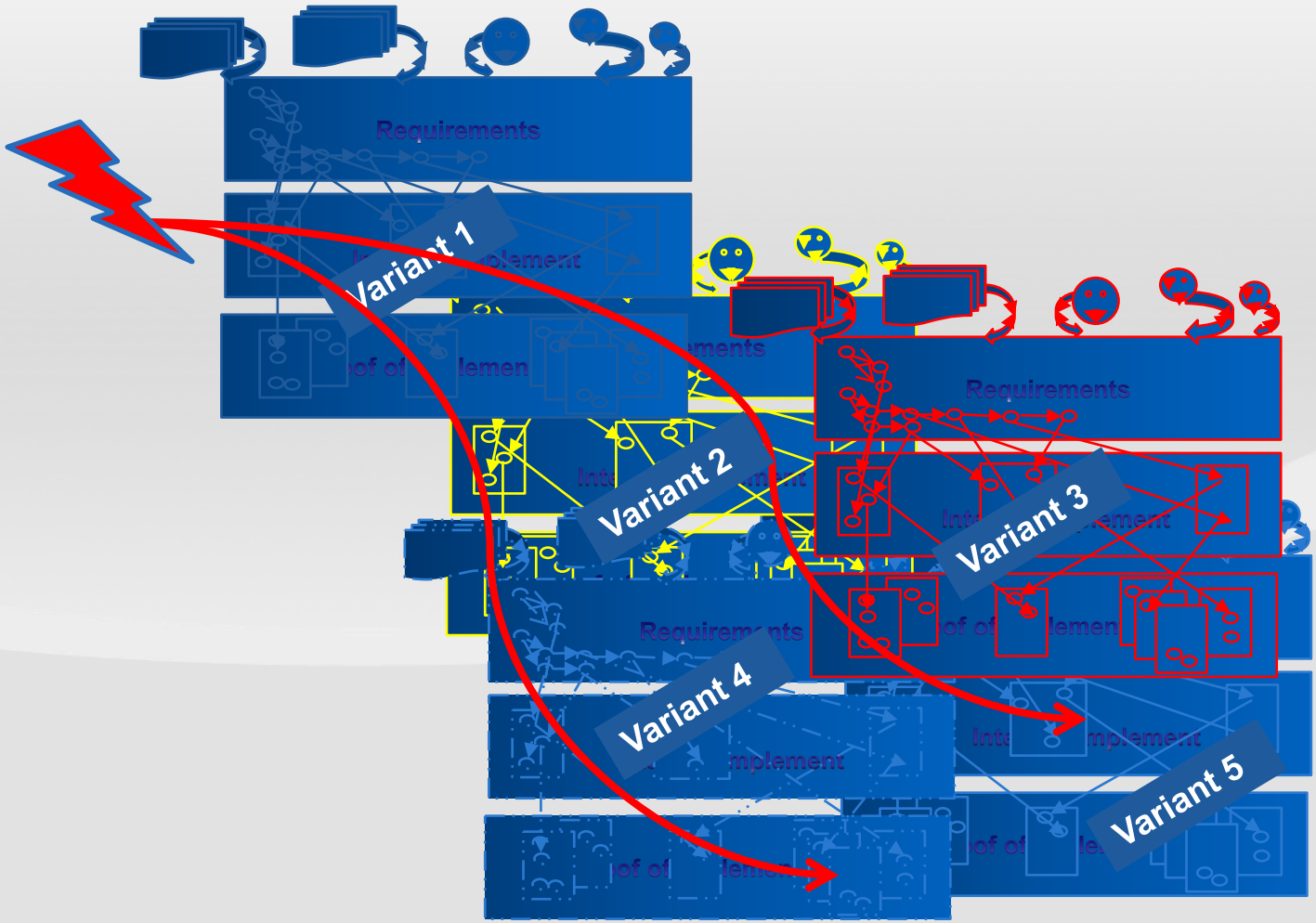
Requirements

Intent to Implement

Proof of implementation

accellera SYSTEMS INITIATIVE

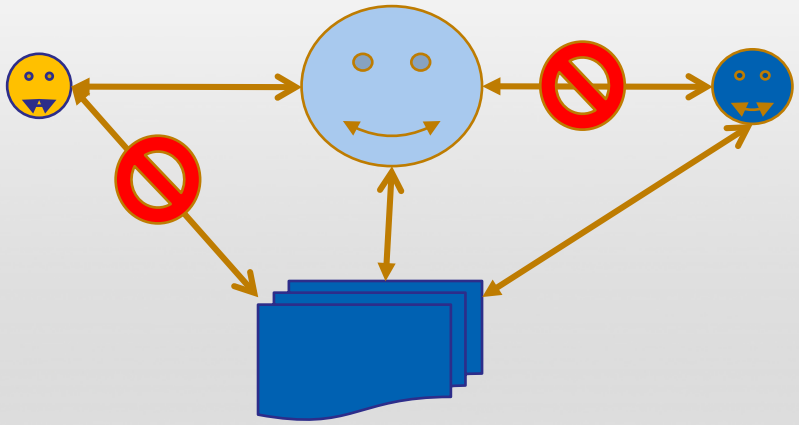
Variants, Reuse & Communication



accellera SYSTEMS INITIATIVE

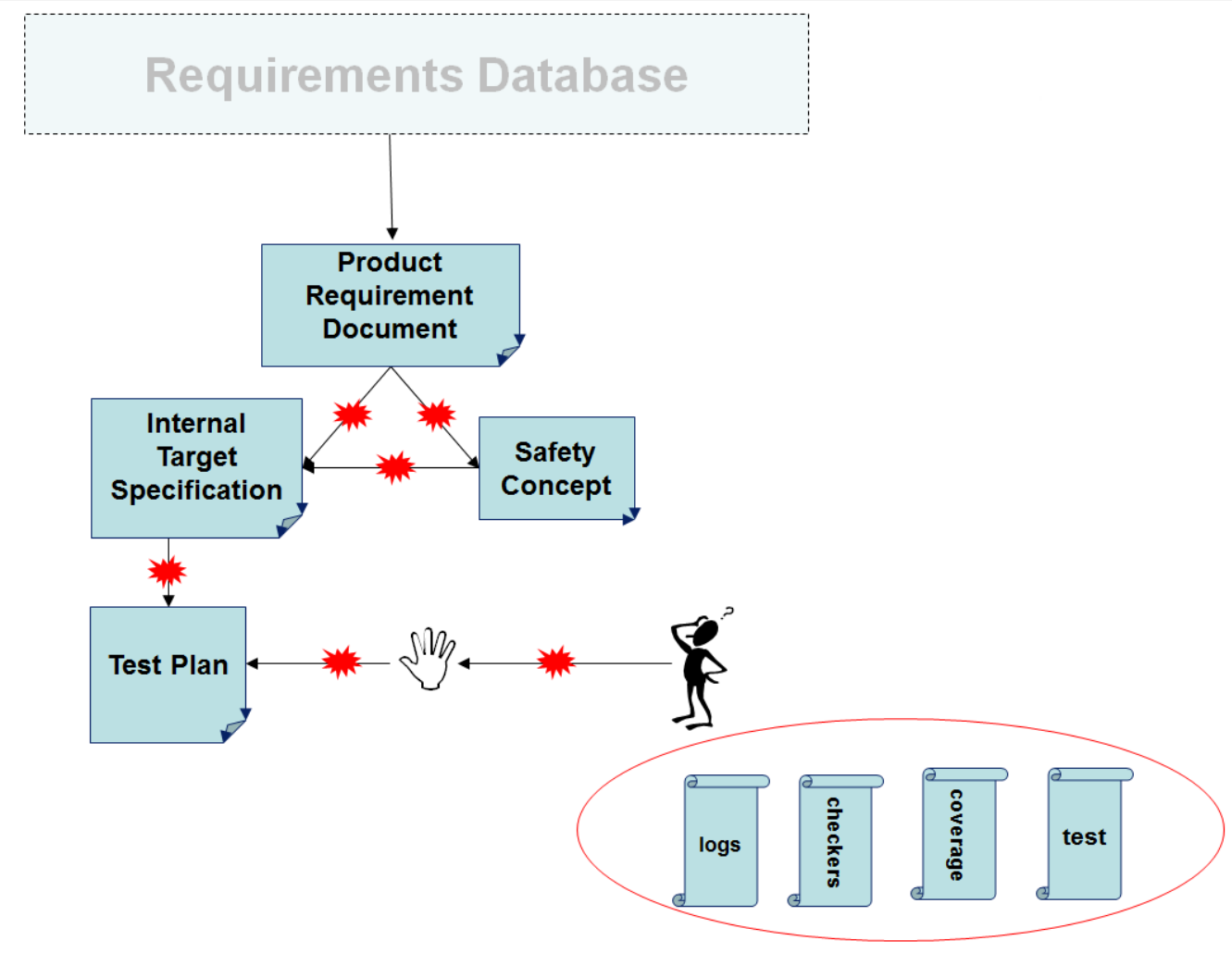
Issues

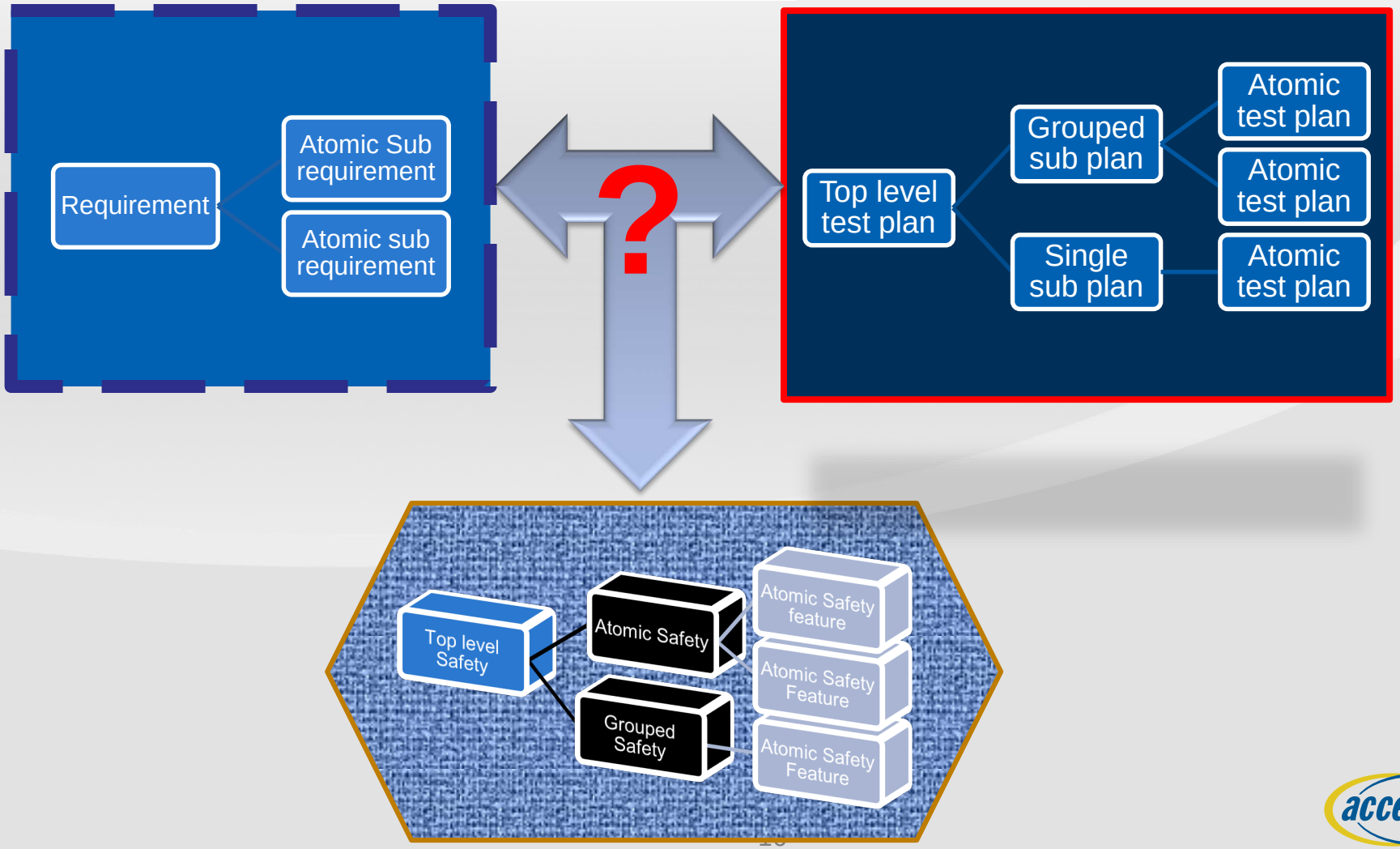
Conflicts



Comprehension

Data Integrity




CRYSTAL
Requirement
Atomic Sub requirement
Atomic sub requirement
?
Top level test plan
Grouped sub plan
Single sub plan
Atomic test plan
Atomic test plan
Atomic test plan
Top level Safety
Atomic Safety
Grouped Safety
Atomic Safety feature
Atomic Safety Feature
Atomic Safety Feature
accellera
SYSTEMS INITIATIVE

Functional Hazard

Function

- What function ensures requirement is achieved

Functional Failures

- No Function
  - **HAZARD** : Doesn't do what its designed to
- Incorrect Function
  - **HAZARD** : Incorrectly does an incorrect function

Situational Analysis

- Usage situation - when is it likely to happen
- People at risk – who can be hurt by a failure

Hazard Level Analysis

Lane Keeping assistant example

Identify hazards

Hazard	:	Doesn’t stay in lane
Situation	:	Unintended lane change
UID	:	123
Severity	:	S3
Rationale	:	Unintended change due to speed at which the system is active or required may be life threatening to multiple parties
Exposure	:	E4
Rationale	:	Possibility of occurrence over any frequency or duration of travel in car
Control	:	C3
Rationale	:	May be required override for danger situation - short time scale to consider appropriate other actions and system not reacting to request
ASIL	:	ASIL D

Safety Requirements

Safety goal

The Drivers and other road users shall not be exposed to unreasonable risk due to unintended lane change

Safe State

The Vehicle shall remain in the lane in which they intended

Functional goal

Avoid Undemanded Steering

Functional Safety Requirement

System shall detect excessive motor torque

Requirement Quality Gateway

- Requirements are expensive
  - ROI
  - Quality Criteria :
    - Unambiguous
    - Testable (verifiable)
    - Clear (concise, terse, simple, precise)
    - Correct
    - Understandable
    - Feasible (realistic, possible)
    - Independent
    - Atomic
    - Necessary
    - Implementation-free (abstract)

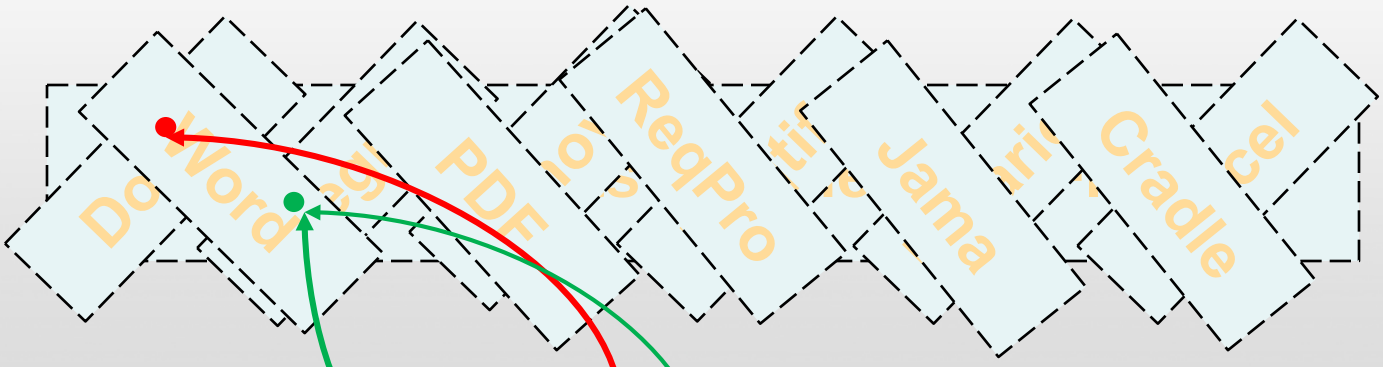
- How do we check for quality
  - Boilerplates
  - Manual inspection (review)
  - model rule checker (if model based)

Shift left

Considerations

- Requirements stages
- Data management
- Where to store/communicate
- Change management
- Visualisation
- Process/Flow
- Communication
- How to prove

Requirements Driven Verification And Test



Test scripts

Integration testing

Manual review

Assertions

Where?

Pass?

Metadata?

SOC Simulations

IP2 Simulations

IP1 Simulations

Parameteric testing

Directed testing

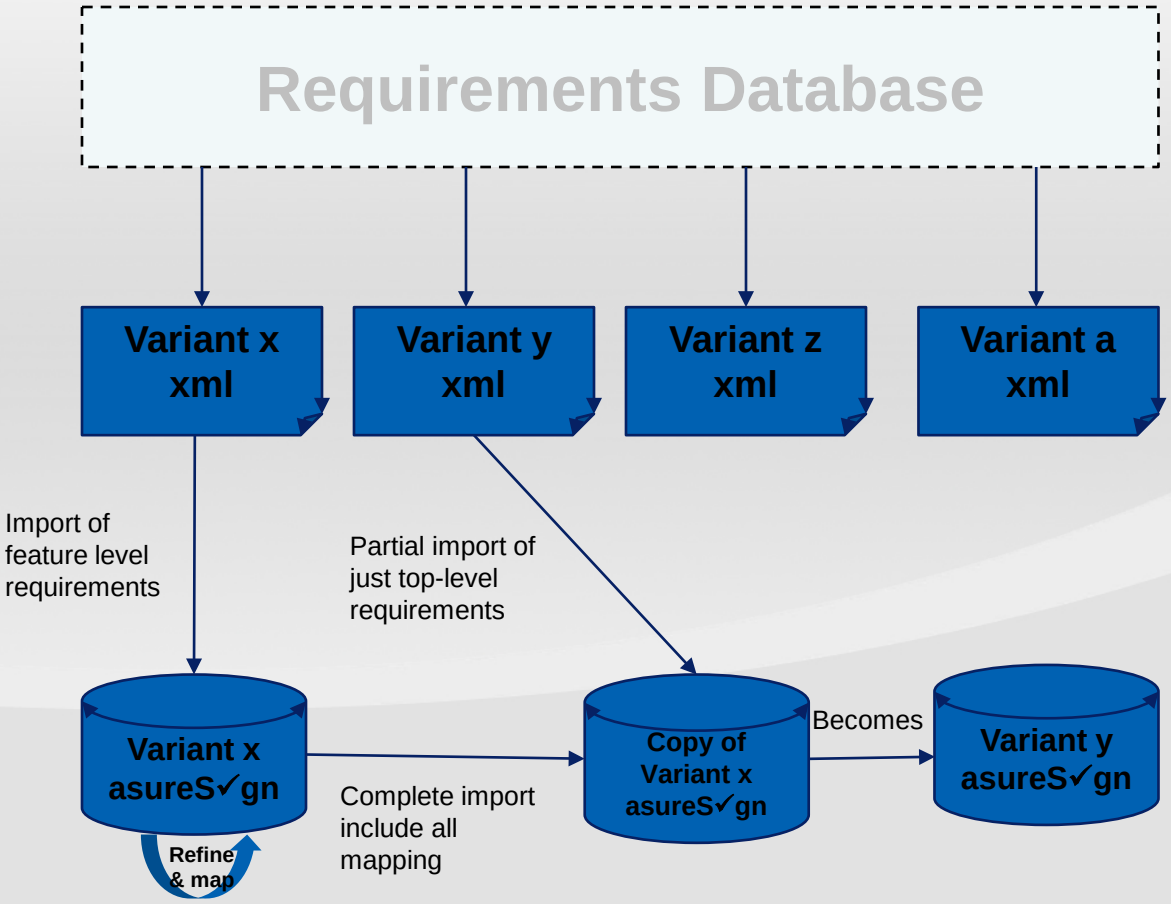
Structural coverage

Formal testing

Functional coverage

SYSTEMS INITIATIVE

Variant Management

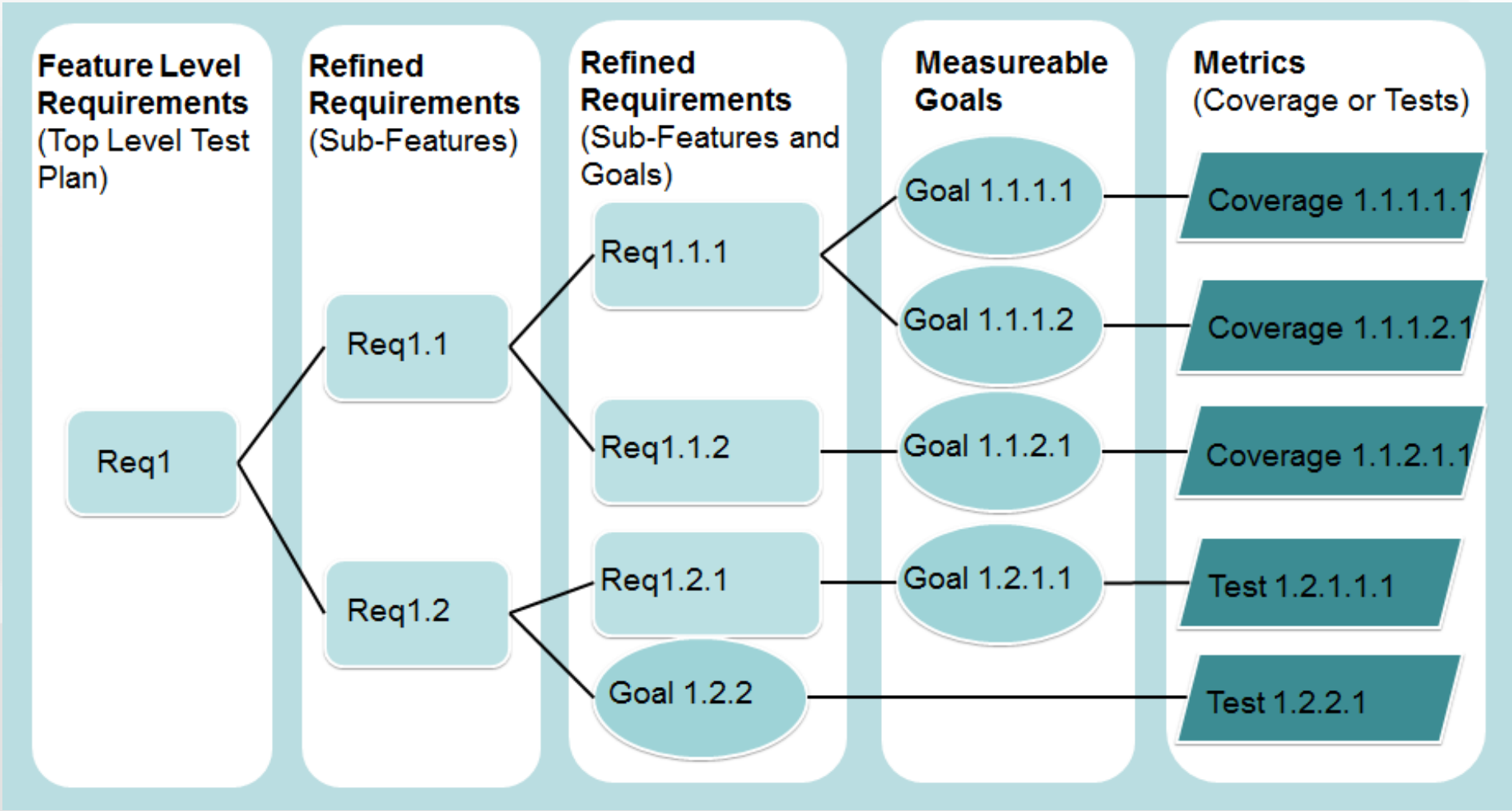


accellera SYSTEMS INITIATIVE

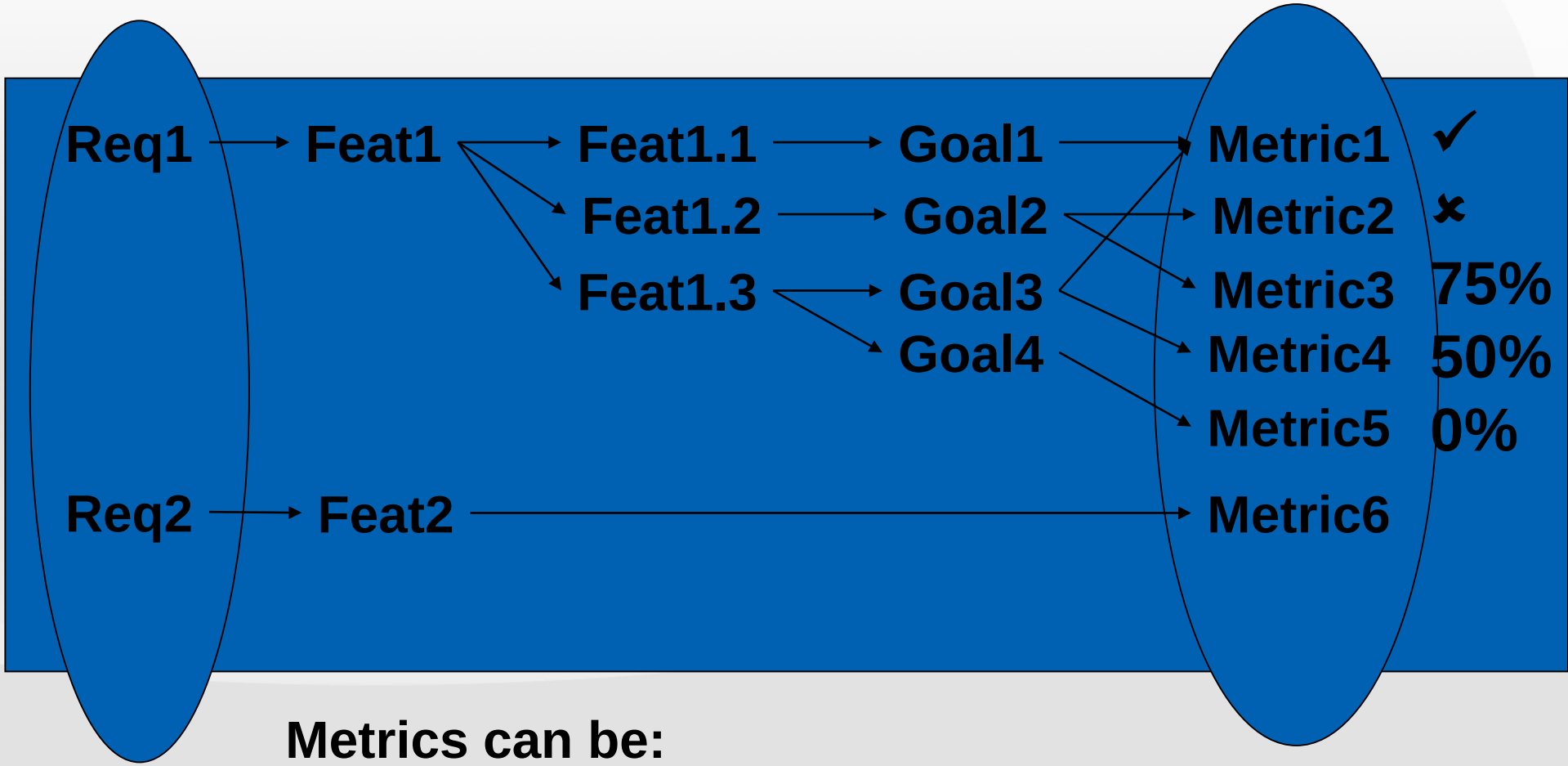
Supporting Advanced Verification

- Constrained random verification with automated checks based on models or scoreboards, etc.
- Coverage driven verification based on functional coverage models and code coverage metrics.
- Assertion-based verification.
- Formal property based verification.

Supporting Advanced Verification



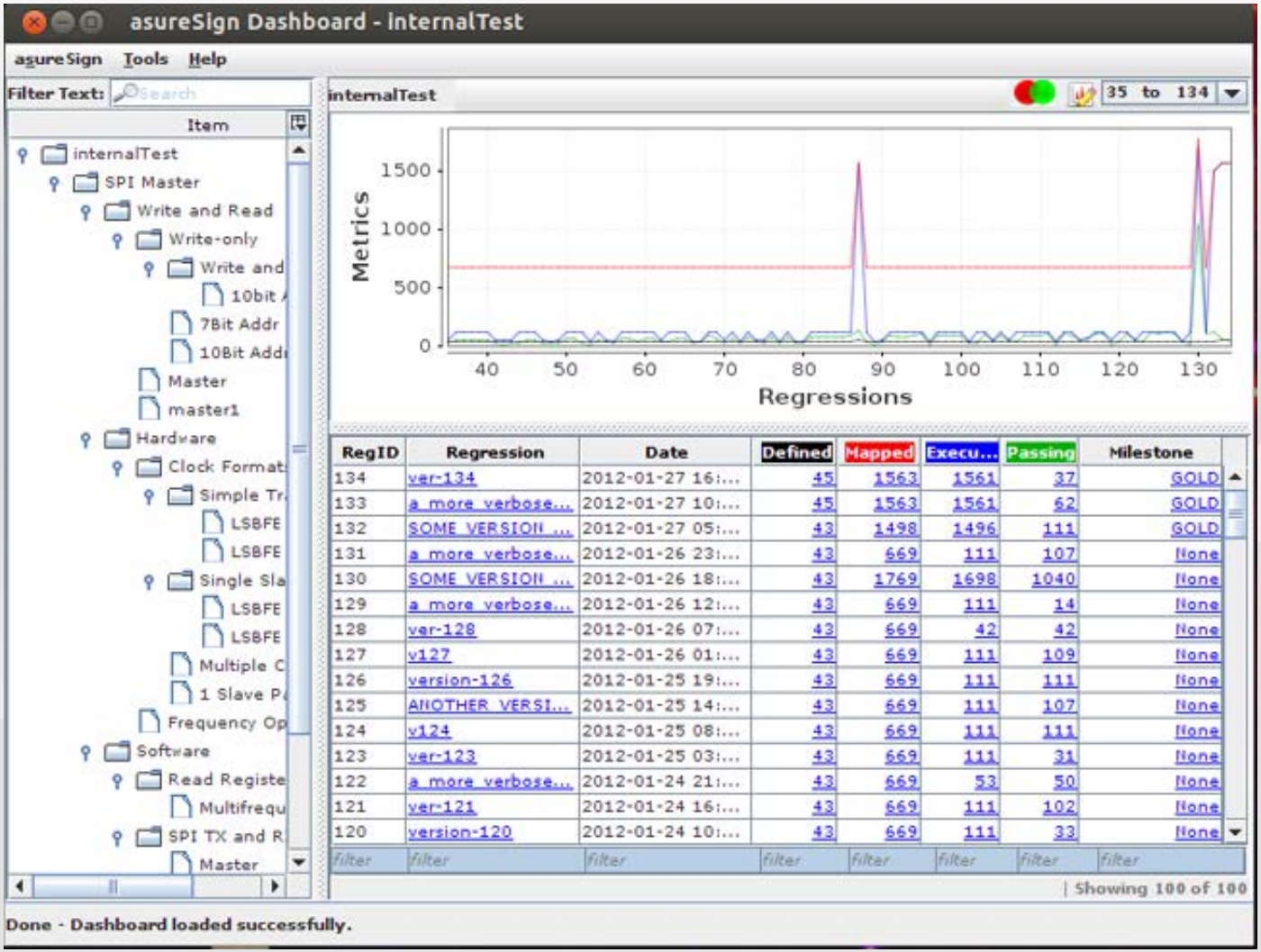
Tracking



Metrics can be:

- **From HW verification**
- **From Silicon validation**
- **From SW testing**

Track Progress on Requirements Signoff



accellera SYSTEMS INITIATIVE

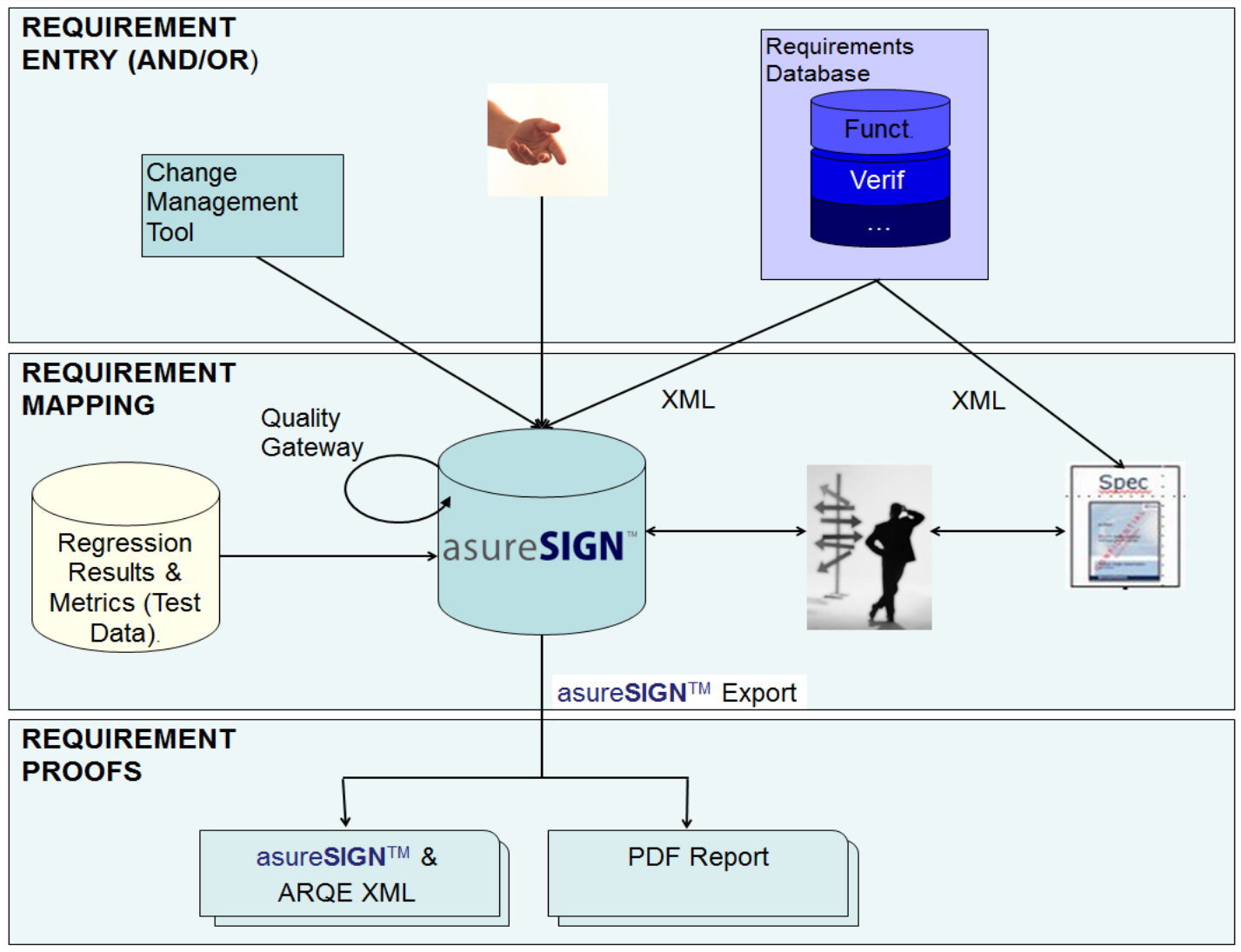
Supporting Hierarchical Verification

- A requirement might be signed off at multiple levels of hierarchy during the hardware development
  - Block
  - Subsystem
  - SoC
  - System
    - Including Software
  - Post Silicon

Tool Support Requirements

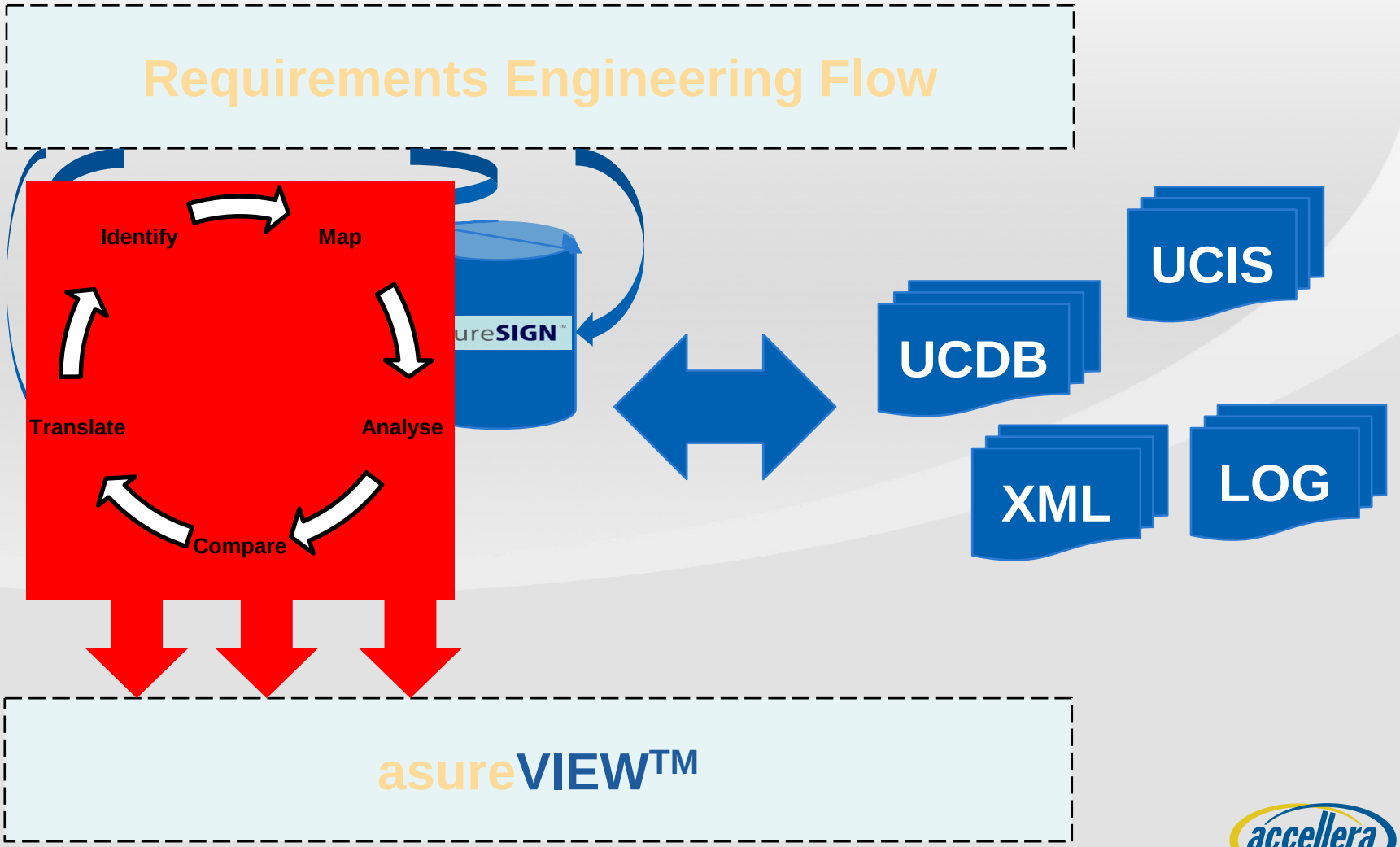
- Requirements -> test plan
- Data Integrity, hierarchy, data translation
- Change management – instant update
- Live database
- Tailored Documented proof
- Allows reviews of implementation document against test plan
- Mapping
- Test management
- Compliance / Audit Management

asureSIGN Dataflow

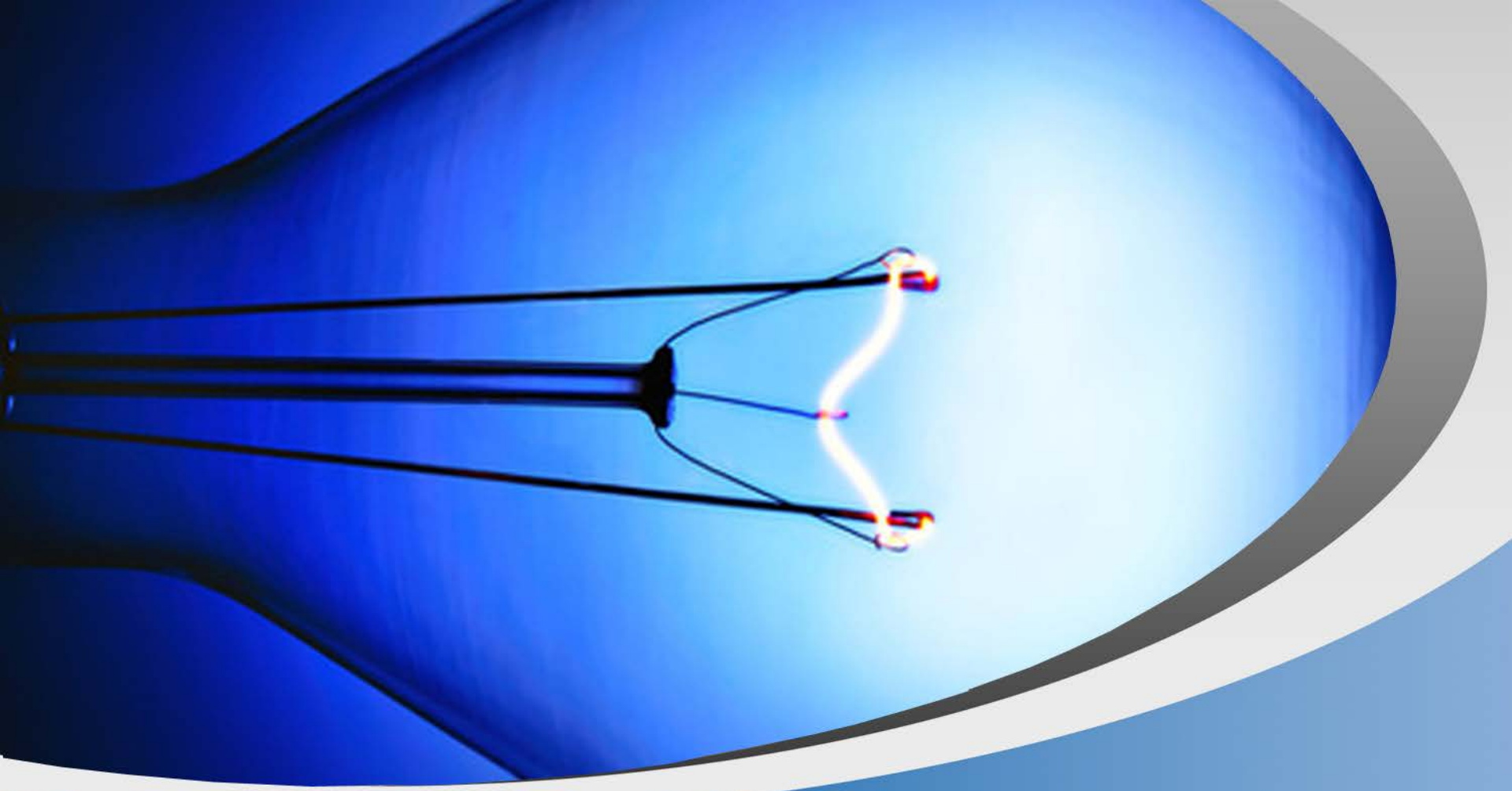


accellera SYSTEMS INITIATIVE

asureSIGN™ Solution Built on UCIS

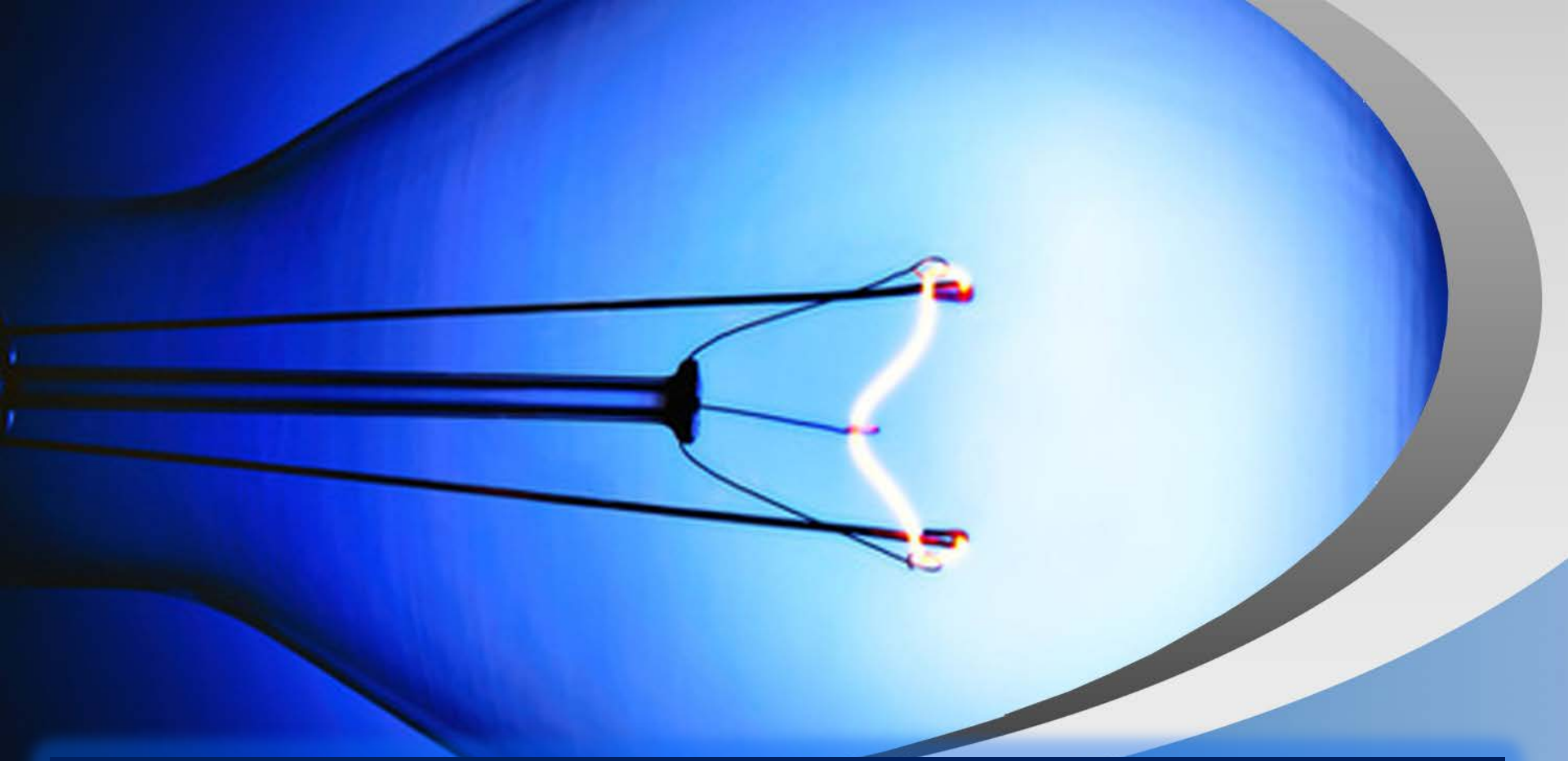


accellera SYSTEMS INITIATIVE



Thank you!

accellera
SYSTEMS INITIATIVE



Next Generation Design and Verification Today

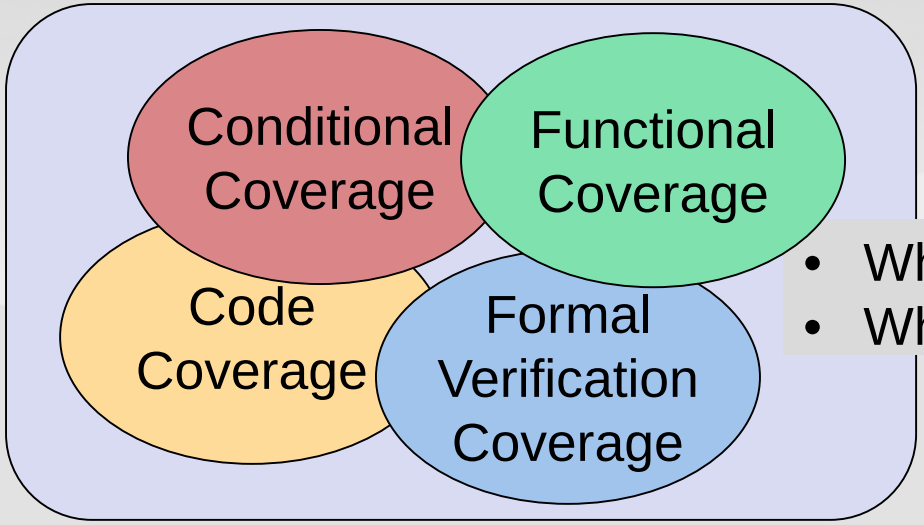
Using UCIS to Combine Verification Data from Multiple Tools

Mike Bartley, TVS

accellera SYSTEMS INITIATIVE

Motivation for UCIS

- **Verification is hard**
  - <insert standard slide: 70+%, increasing complexity, yadda, yadda, yadda>
- **Variety of verification techniques and methods**
  - Directed and constrained-random simulation
  - Formal verification
  - Testbench methodologies



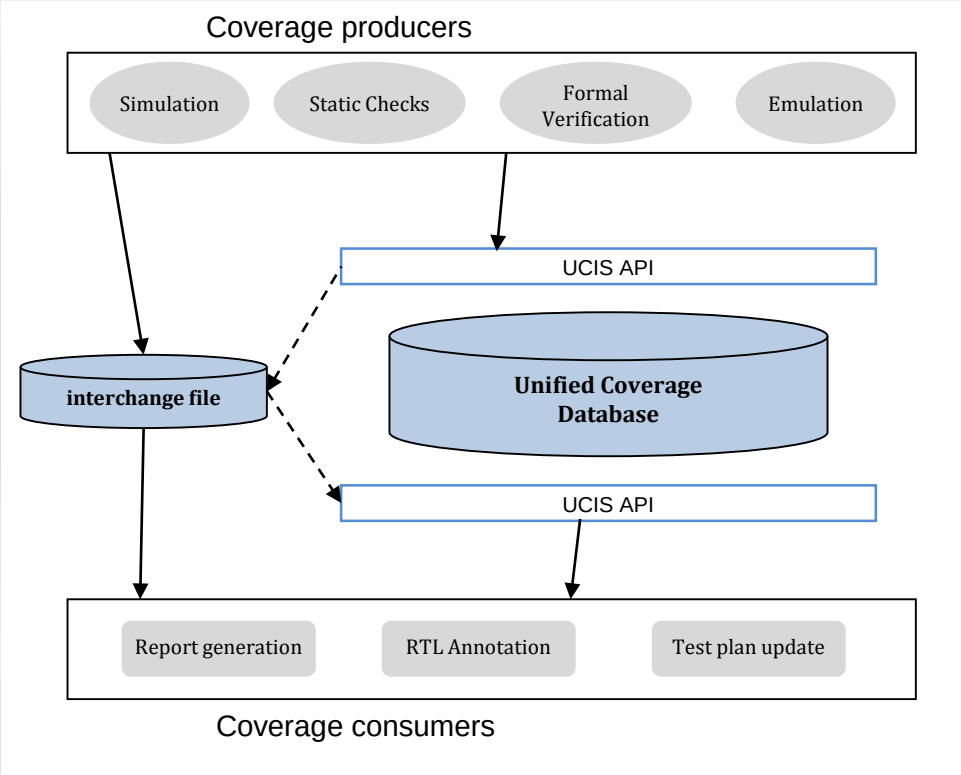
- What coverage overlaps?
- What coverage is missing?

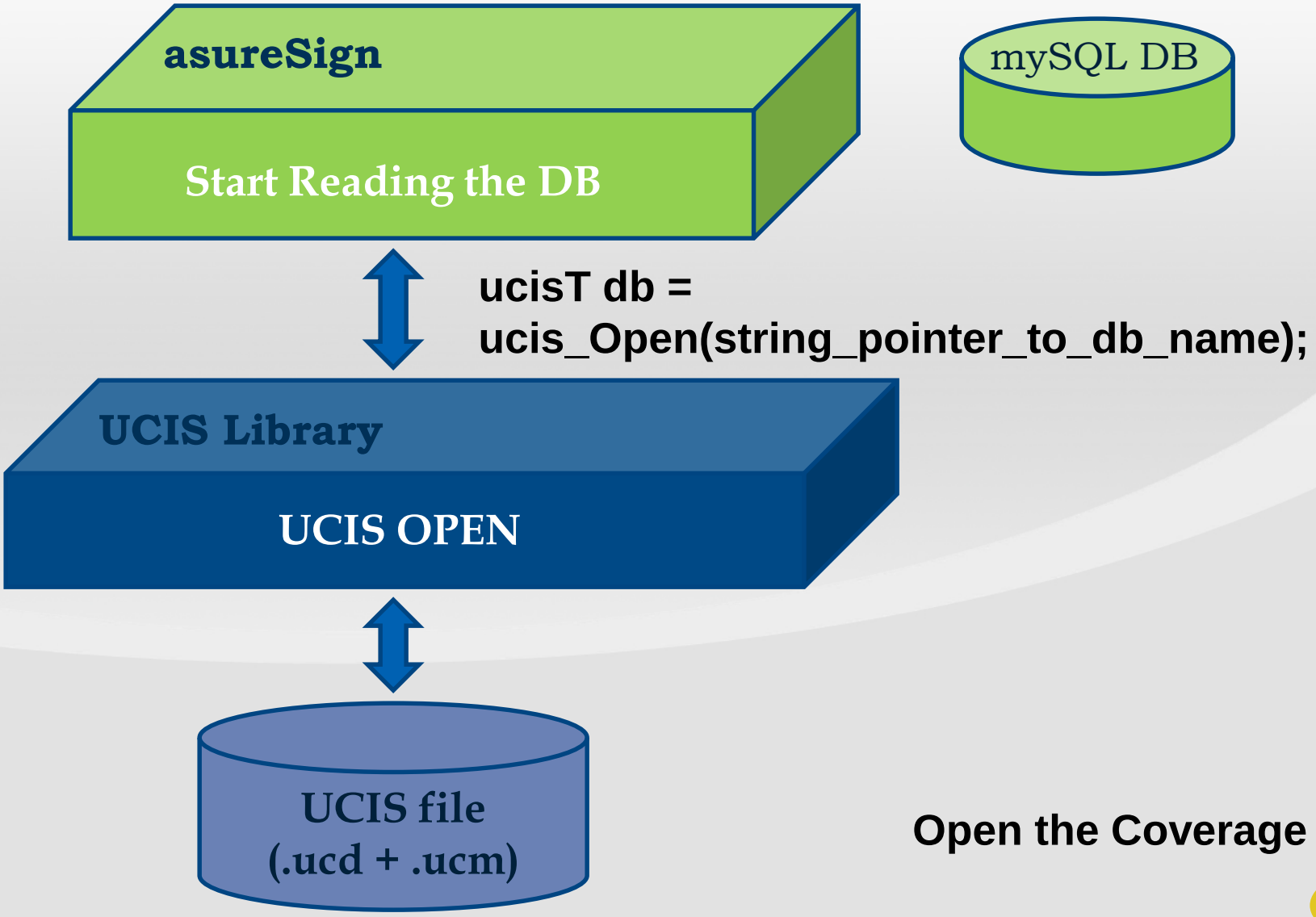
Motivation for UCIS

- **Verification is hard**
  - <insert standard slide: 70+%, increasing complexity, yadda, yadda, yadda>
- **Variety of verification techniques and methods**
  - Directed and constrained-random simulation
  - Formal verification
  - Testbench methodologies
- **Design and verification engineers need coverage metrics:**
  - What has been checked, what remains to be checked?
  - How many engineers do we need?
  - How much time do we need?
  - Where best to direct verification resources?
  - What is the best tool or method to efficiently cover problem areas?

Unified Cases and Data Flow

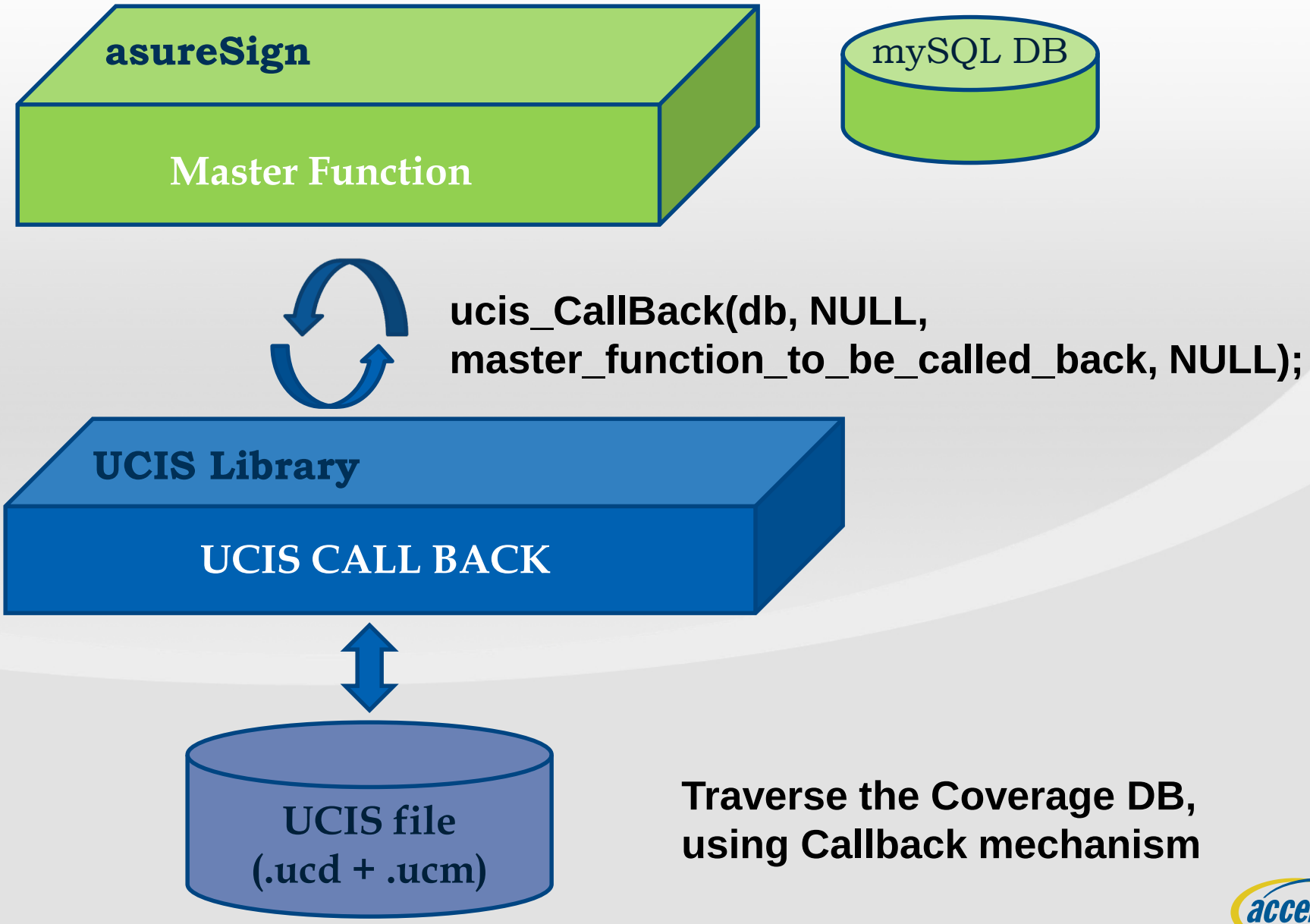
- **Generate**
  - Single verification run, single/multiple coverage types
  - Multiple verification runs
- **Access**
  - Using UCIS Application Programming Interface (API)
  - Using Interchange Format (XML Interchange Format)
- **Analyze**
  - Report unhit coverage points
  - Track progress of coverage over time
- **Merge**
  - Across runs, components, tools

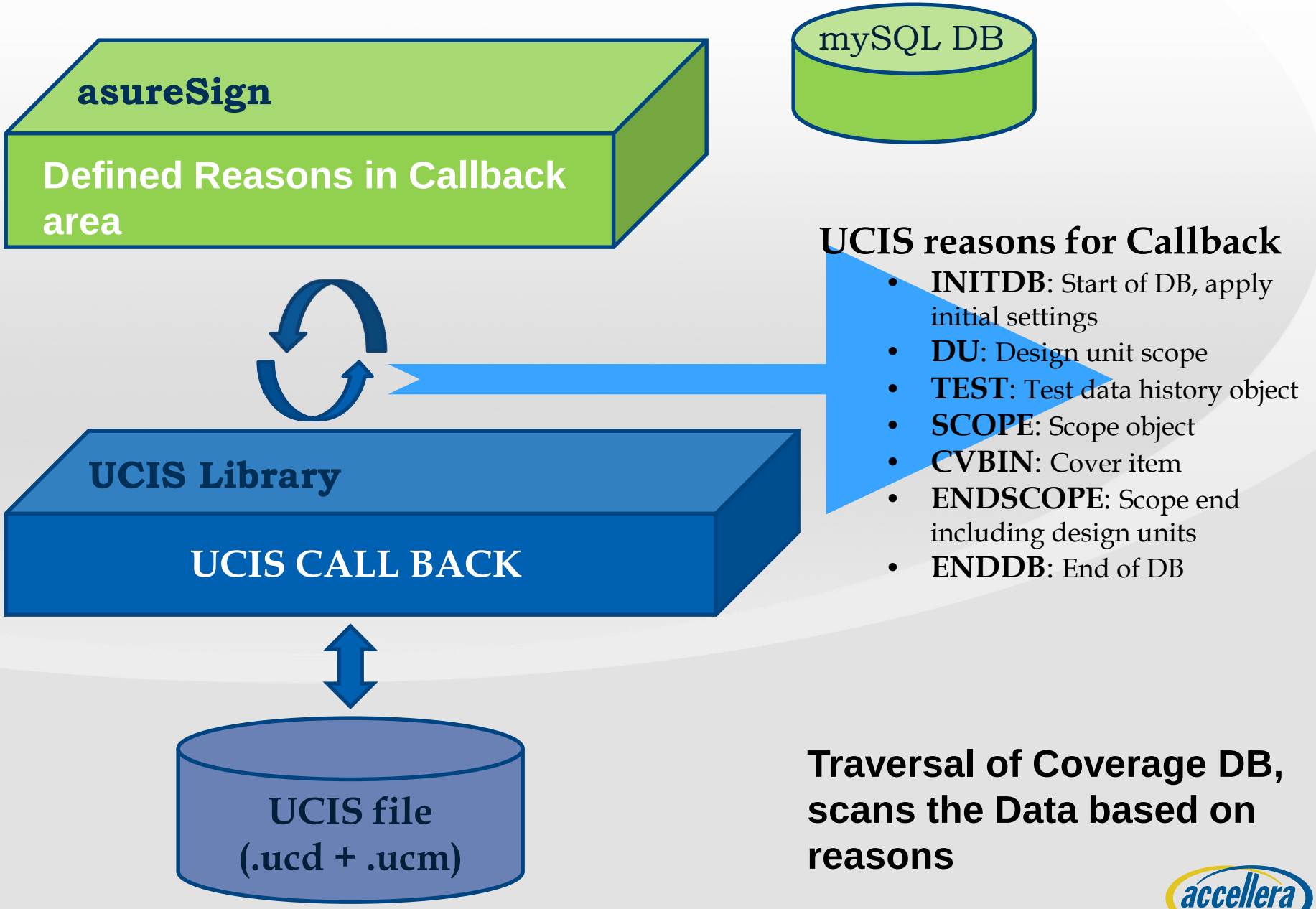




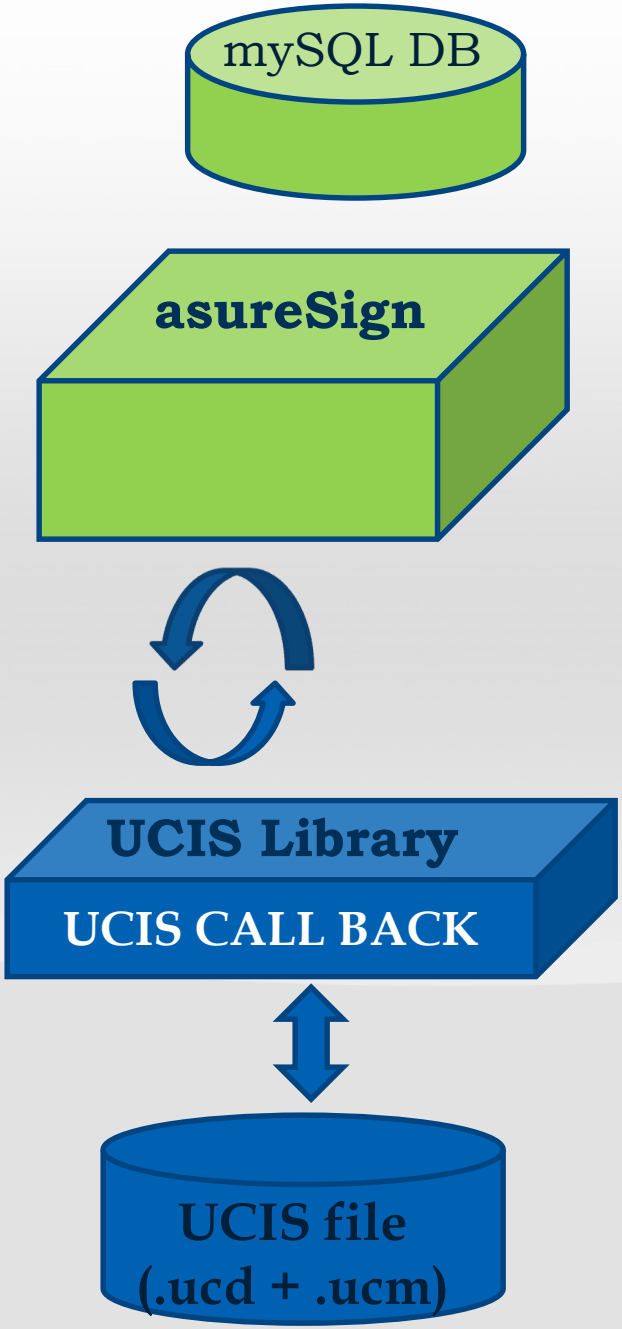
Open the Coverage DB

accellera SYSTEMS INITIATIVE


asureSign
Master Function
mySQL DB
ucis_CallBack(db, NULL,
master_function_to_be_called_back, NULL);
UCIS Library
UCIS CALL BACK
UCIS file
(.ucd + .ucm)
Traverse the Coverage DB,
using Callback mechanism

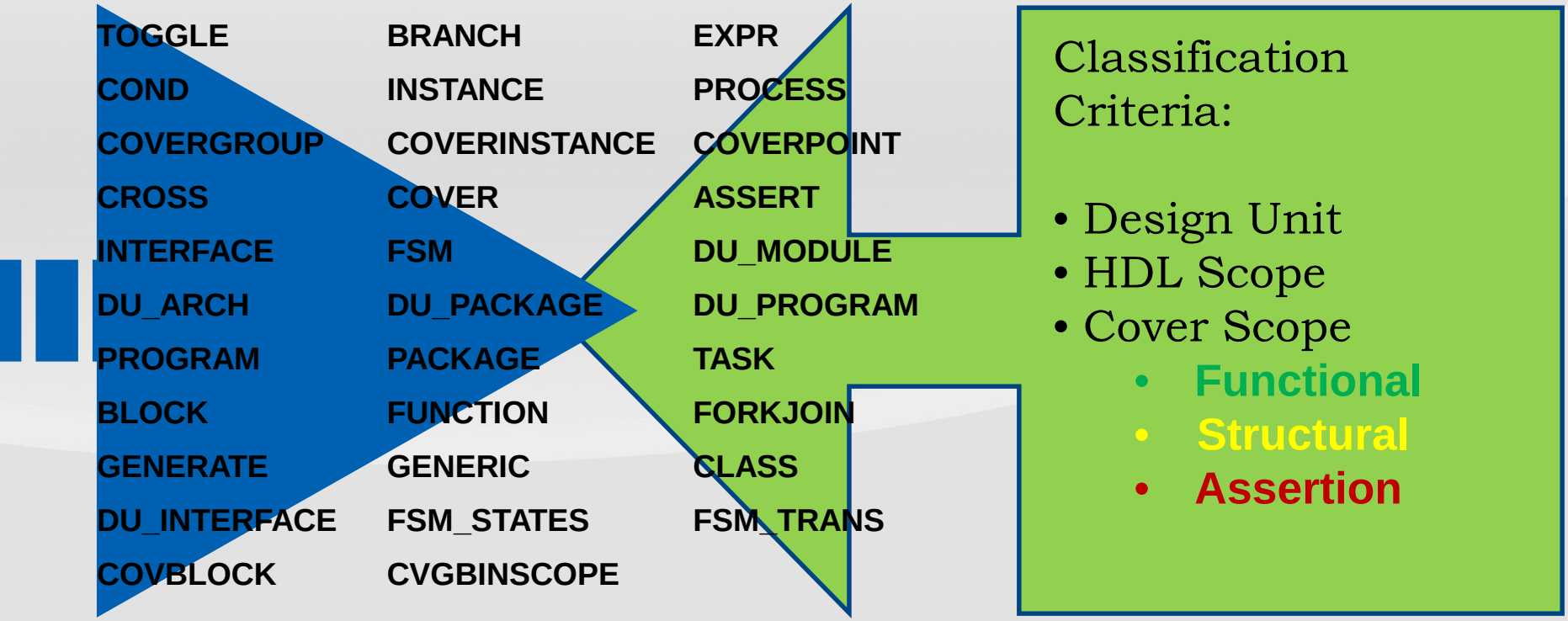

asureSign
Defined Reasons in Callback area
mySQL DB
UCIS reasons for Callback
INITDB: Start of DB, apply initial settings
DU: Design unit scope
TEST: Test data history object
SCOPE: Scope object
CVBIN: Cover item
ENDSCOPE: Scope end including design units
ENDDB: End of DB
UCIS Library
UCIS CALL BACK
UCIS file
(.ucd + .ucm)
Traversal of Coverage DB, scans the Data based on reasons
accellera
SYSTEMS INITIATIVE

Cover Items captured on the basis of Design and Scope



TOGGLE	BRANCH	EXPR
COND	INSTANCE	PROCESS
COVERGROUP	COVERINSTANCE	COVERPOINT
CROSS	COVER	ASSERT
INTERFACE	FSM	DU_MODULE
DU_ARCH	DU_PACKAGE	DU_PROGRAM
PROGRAM	PACKAGE	TASK
BLOCK	FUNCTION	FORKJOIN
GENERATE	GENERIC	CLASS
DU_INTERFACE	FSM_STATES	FSM_TRANS
COVBLOCK	CVGBINSCOPE	

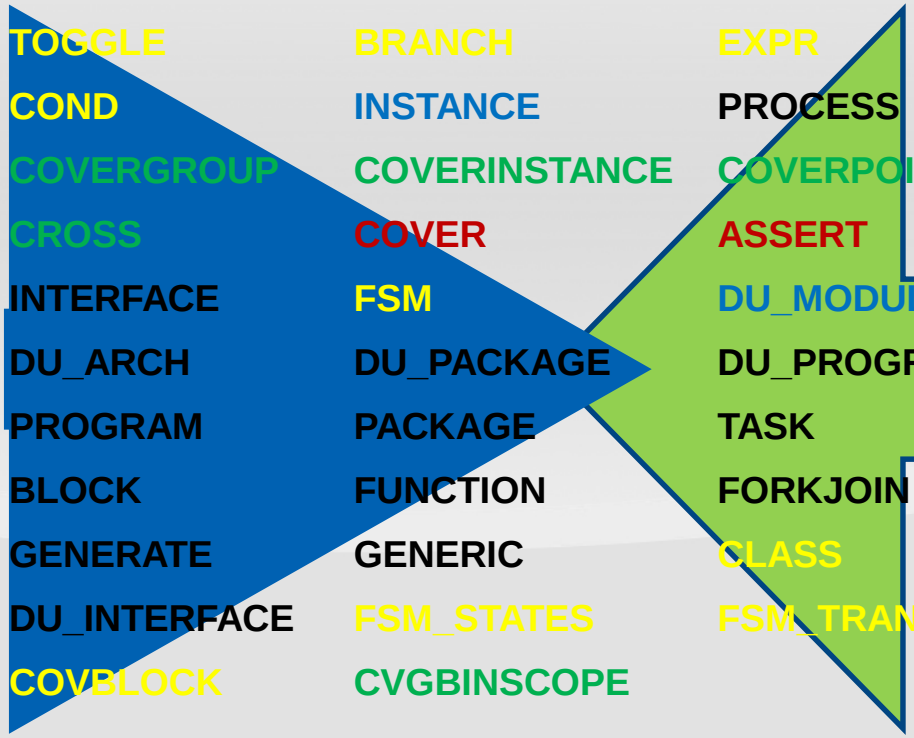
accellera SYSTEMS INITIATIVE



Cover Items captured based on Classification Criteria

accellera SYSTEMS INITIATIVE

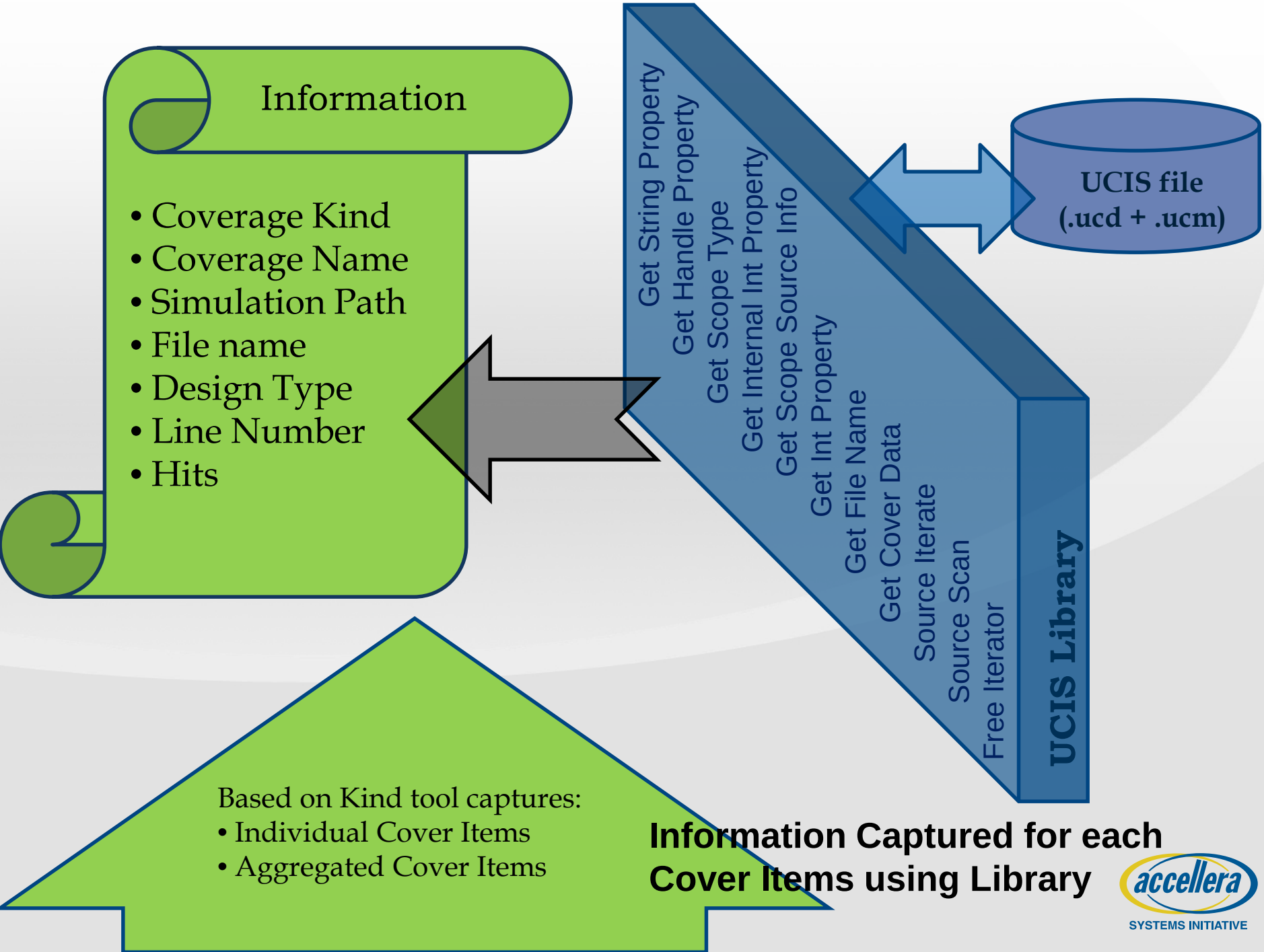
Based on Kind tool captures:
- Individual Cover Items
- Aggregated Cover Items



Classification Criteria:
- **Design Unit**
  - **asureSign only uses instance and module coverage**
- HDL Scope
- Cover Scope
  - **Functional**
  - **Structural**
  - **Assertion**

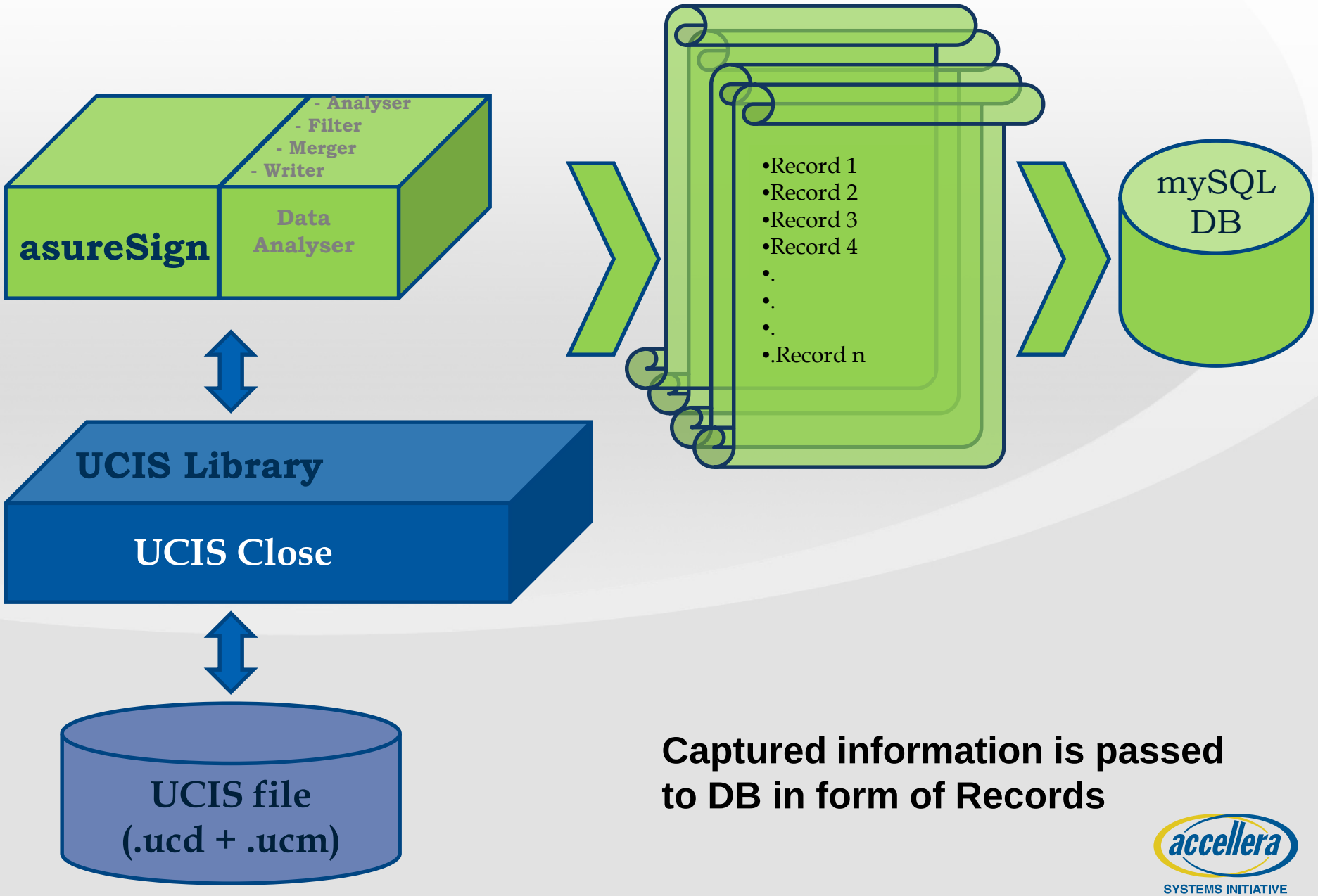
Cover Items captured based on Classification Criteria

accellera SYSTEMS INITIATIVE



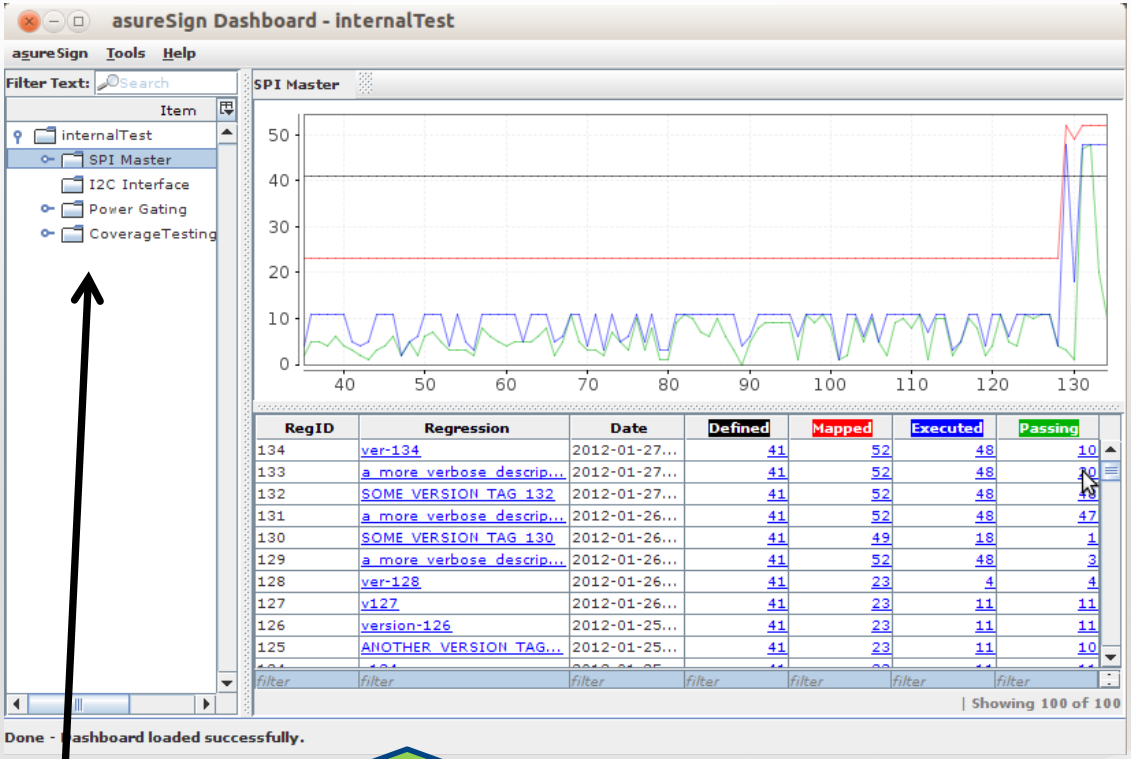
Information Captured for each Cover Items using Library

accellera SYSTEMS INITIATIVE

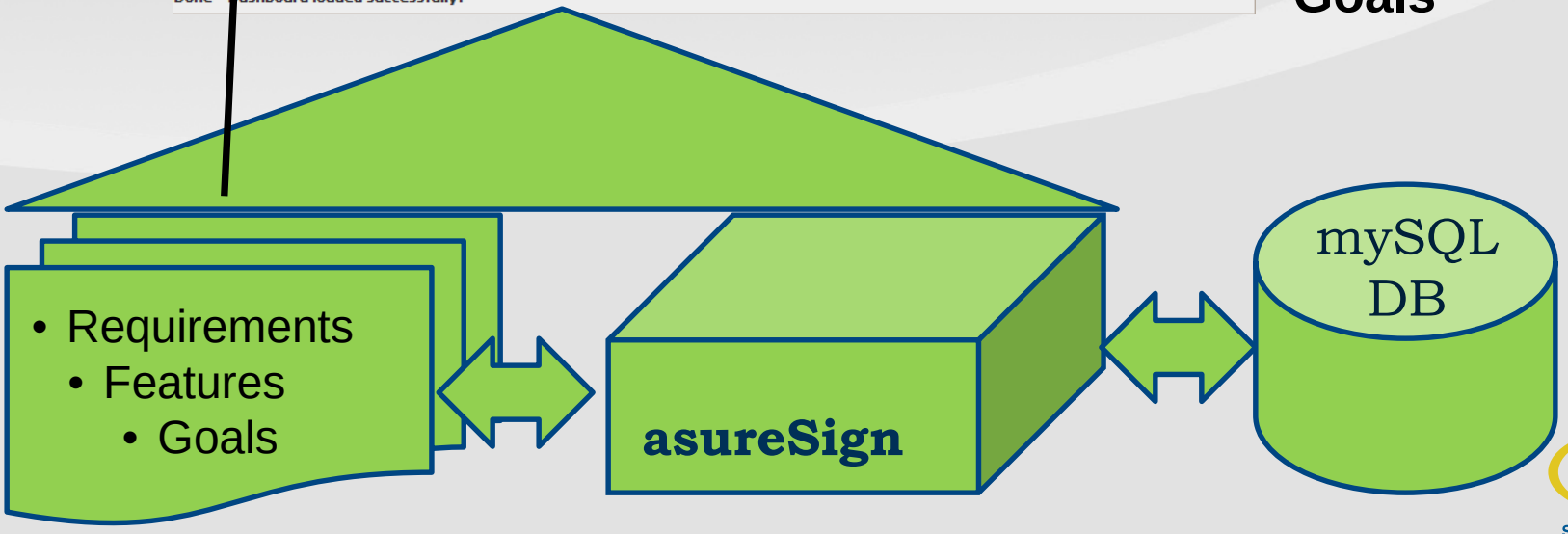


Captured information is passed to DB in form of Records

accellera SYSTEMS INITIATIVE



asureSign uses Captured Data from all sources, and relates it to Requirements via Features and Goals



accellera SYSTEMS INITIATIVE

Advantages of Requirements Driven Verif

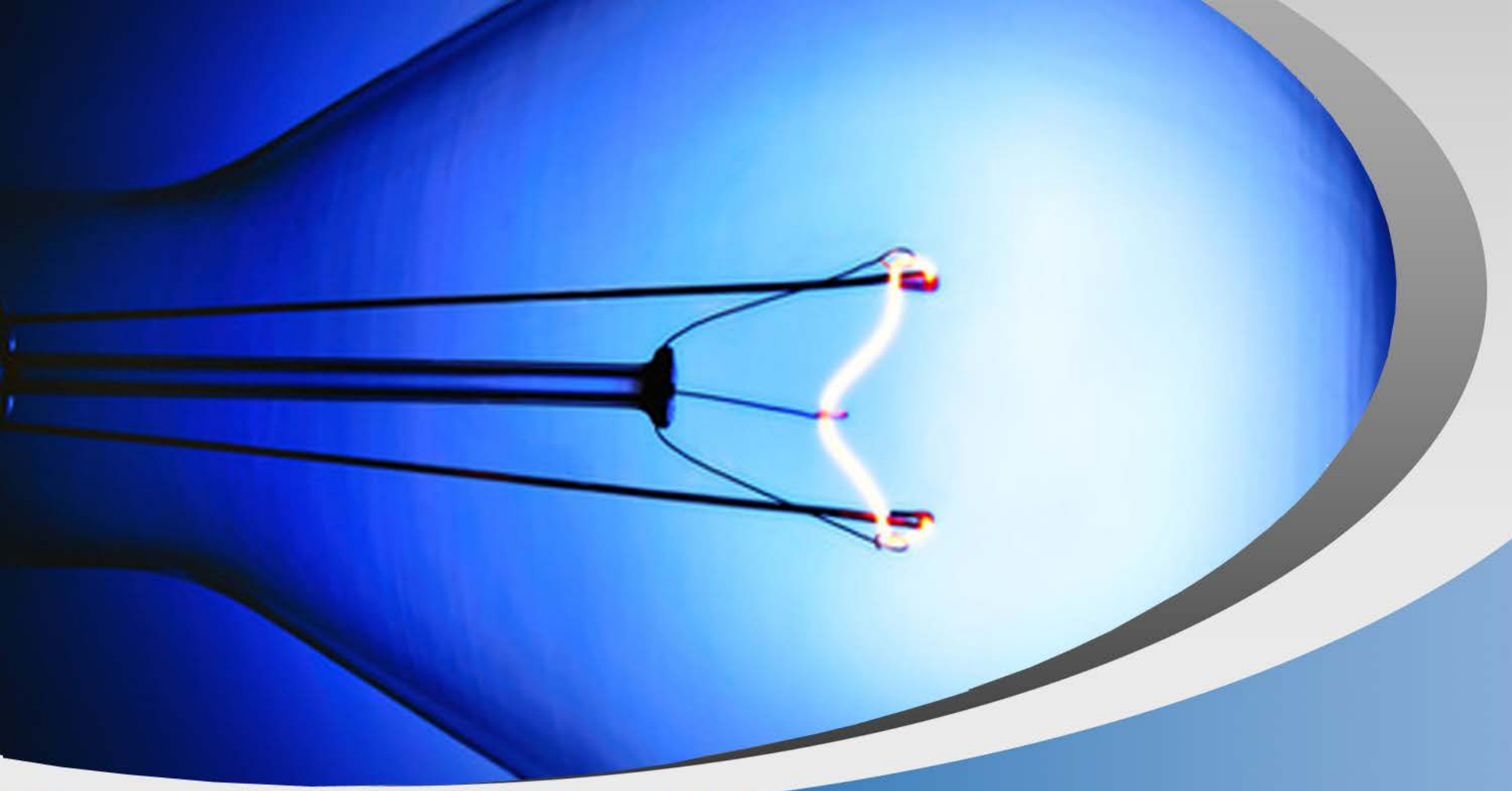
- Requirements Management
- Verification Management
- Project Management
- Impact Analysis
- Product Line Engineering
- Variant Management
- Improved Product Sign-Off

Conclusions #1

- **Requirements Driven Verification**
  - Compliance to various hardware (and software) safety standards
    - IEC61508: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems
    - DO254/DO178: Hardware/Software considerations in airborne systems and equipment certification
    - EN50128: Software for railway control and protection systems
    - IEC60880: Software aspects for computer-based systems performing category A functions
    - IEC62304: Medical device software -- Software life cycle processes
    - ISO26262: Road vehicles – Functional safety
  - And ….
    - Identify test holes and test orphans
    - Track the status of the whole verification effort (planning, writing, execution)
    - Build historical perspective for more accurate predictions
    - Better reporting of requirements status
    - Risk-based testing
    - Prioritisation and Risk Analysis
    - Filtering Requirements based on Customers and releases
    - Impact and conflict analysis

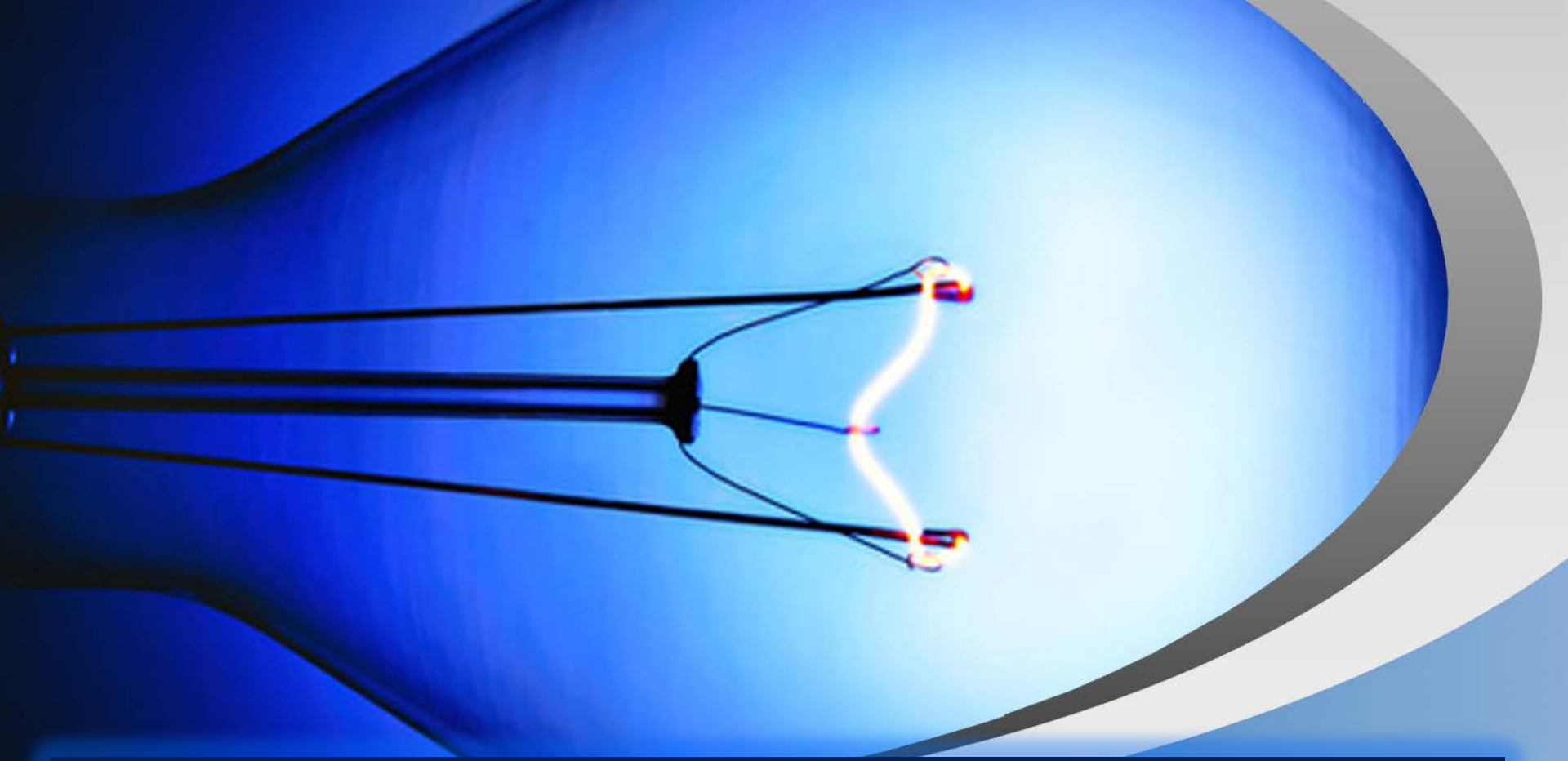
Conclusions #2

- **Advanced verification techniques can be deployed in Requirements Driven Verification**
  - Requirements engineering tools to capture the verification plan & mapping
  - Verification management tools to automate collection of results
- **More info**
  - CRYSTAL http://www.crystal-artemis.eu/
  - White Paper http://www.testandverification.com/wp-content/uploads/tvs-white-paper-asureSIGN.pdf



Thank you!

accellera
SYSTEMS INITIATIVE



Next Generation Design and Verification Today

UVM REG: Path Towards Coverage Automation in AMS Simulations

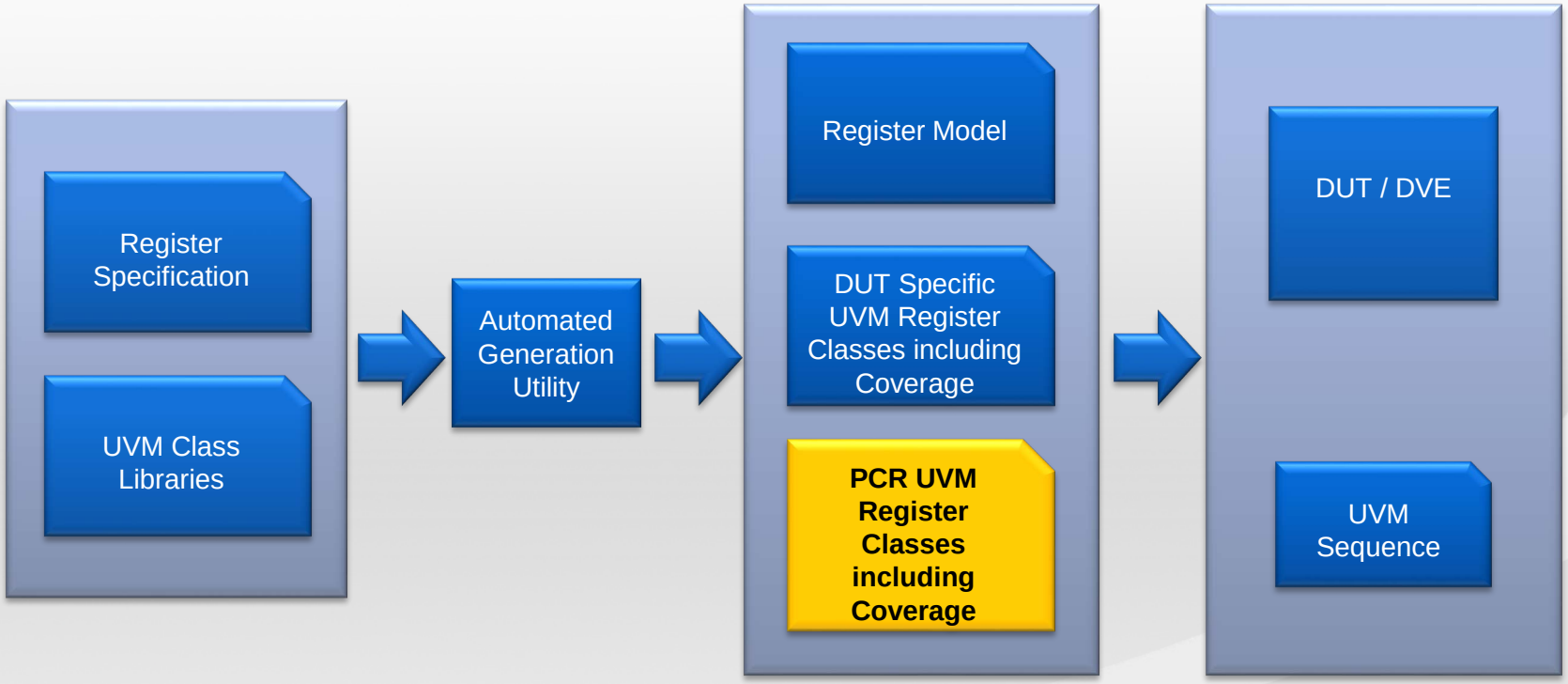
Kyle Newman, Texas Instruments

accellera SYSTEMS INITIATIVE

Agenda

- UVM REG Overview
- Automated UVM REG Generation
- UVM REG Support Tasks
- What are Phantom Coverage Registers (PCRs)?
- PCR Testbench Architecture
- PCR Usage in Mixed Signal Simulation Environment
- Simulation and Coverage Results Examples
- Conclusion
- Discussion and Feedback

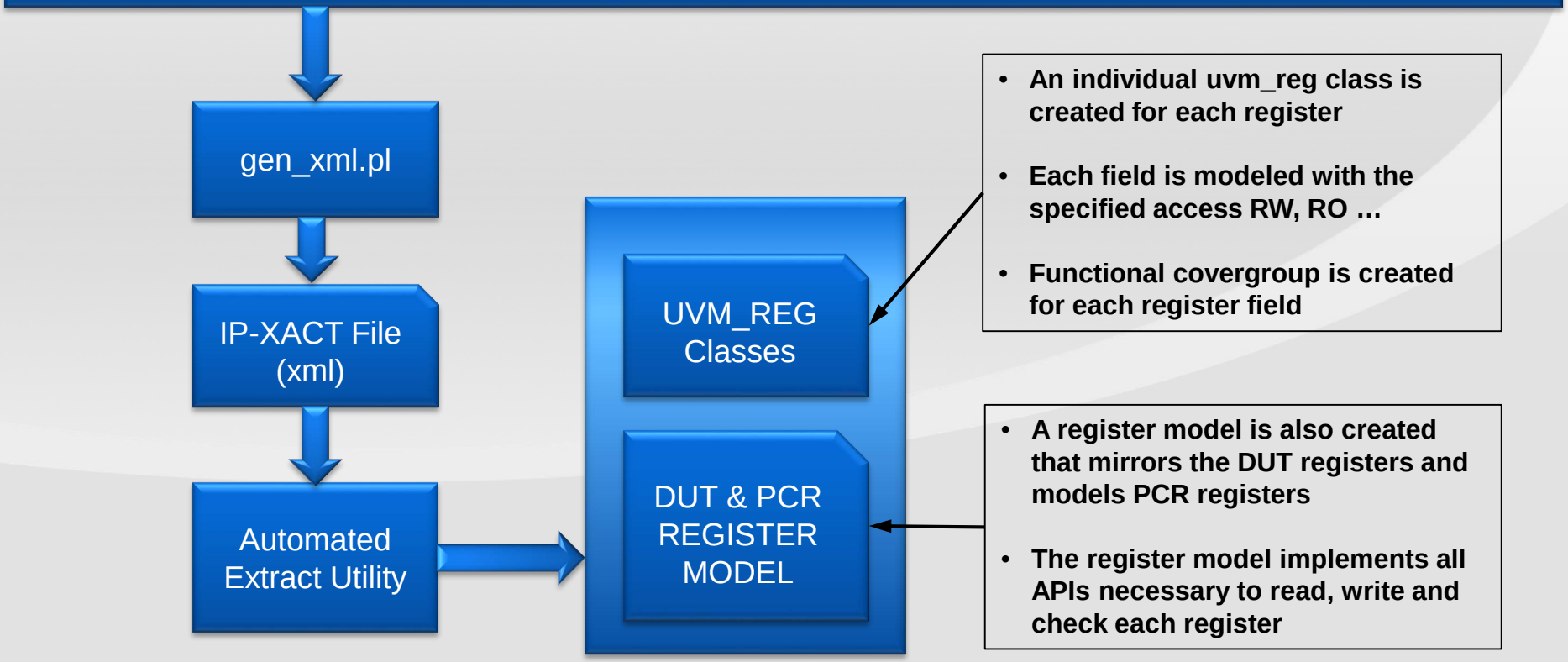
UVM_REG Components



- Set of UVM (System Verilog) register class library
- Register specification spreadsheet
- Automatic generation utilities for creation of UVM register environment
- Provides models for all registers including functional coverage
- Provides all APIs needed to access register model and easily interface to the DVE
- Includes a set of built-in sequences to do basic tests on all registers

Automated UVM REG Generation

NAME	DESCRIPTION	ADDRESS	SIZE	ACCESS	RESET(VALUE)	RESET(MASK)	FIELDNAME	FIELDOFFSET	WIDTH	FIELD ACCESS	FIELD NAME	FIELD OFFSET	WIDTH	FIELDACCESS
REG_1	GLOBAL	0x01	8	RW	0x00	0xFF	CMD	0	8	RW				
PCR_FSM	FSM	0x10	8	RW	0x00	0xFF	STATE	0	8	RW				
PCR_BG_ASSERT	**BANDGAP_ASSERT**	**0x20**	**5**	**RW**	**0x00**	**0x1F**	**VALID**	**4**	**1**	**RO**	**TRIM**	**0**	**4**	**RW**



- **An individual uvm_reg class is created for each register**
- **Each field is modeled with the specified access RW, RO …**
- **Functional covergroup is created for each register field**

- **A register model is also created that mirrors the DUT registers and models PCR registers**
- **The register model implements all APIs necessary to read, write and check each register**

accellera SYSTEMS INITIATIVE

UVM REG Support Tasks

Task	Purpose
write() / read()	Write or read value to DUT through register interface BFM
set() / get()	Zero time access to set or get desired value from the register model
peek() / poke()	Zero time backdoor access to get or set DUT register value using specified hdl path

accellera SYSTEMS INITIATIVE

What are Phantom Coverage Registers (PCRs)?

1. Phantom Coverage Registers (PCRs) are design verification "only" registers for coverage collection and dynamic stimulus generation

2. PCR registers are not HW registers but require an hdl_path to be defined to each bit in the PCR as design verification registers

3. Using peek()/poke() accesses on PCRs in zero time, important DUT (analog & digital) signals can be monitored for DV

4. Coverage is automatically collected when peek()/poke() accesses are done on PCRs

What are Phantom Coverage Registers (PCRs)? (Cont.)

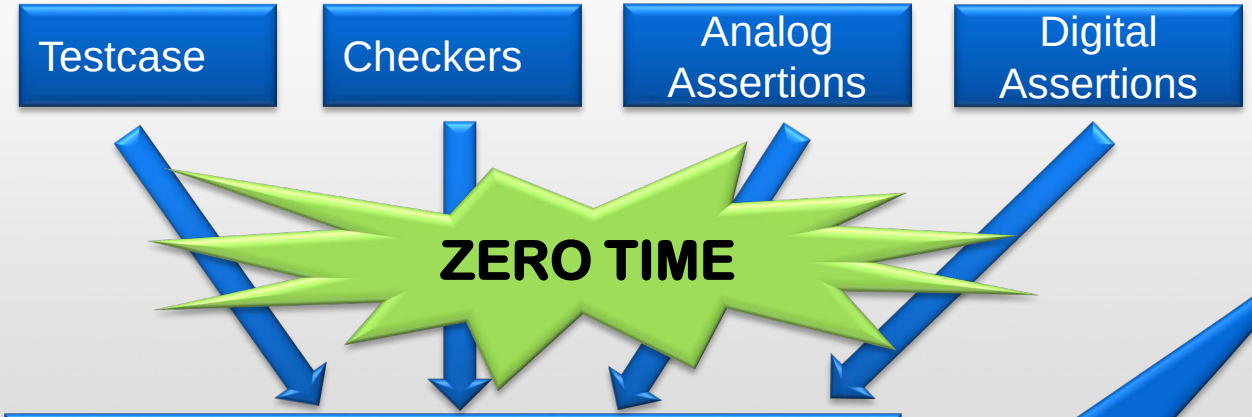
5. Unique hdl_paths for each PCR bit provides an extremely flexible yet simple methodology for collecting coverage data

6. PCRs are defined in Excel spreadsheet which allows for easy management and quick automatic regeneration of the SV code

7. PCR bits can represent state of a particular electrical/digital node in the DUT or even a Pass/Fail status from a testbench checker or assertion (PSL or SV)

8. Status bits in PCRs can be accessed or polled by testcases for automatic stimulus adjustment based on DUT state

PCR Testbench Architecture



Coverage Data

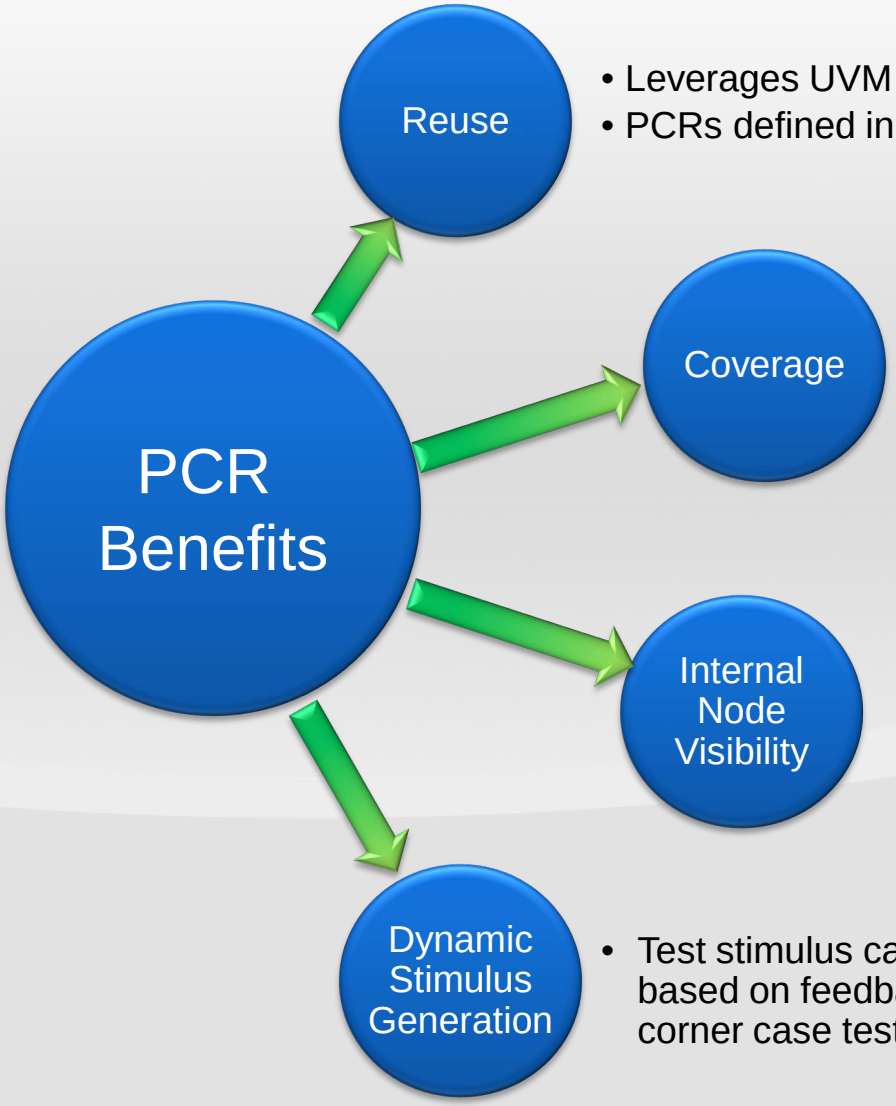
- PCR_BG_ASSERT
- PCR_*block1*
- PCR_*block2*
- PCR_*blockn*

```
// Register definition
class PCR_BG_ASSERT_type extends uvm_reg;
  rand uvm_reg_field VALID;

covergroup wr_cg;
    option.per_instance=1;
    VALID : coverpoint VALID value[0:0];
endgroup

  virtual function void sample(uvm_reg_data_t  data, byte_en,

    bit is  _read, uvm_reg_map map);
    super.sample(data, byte_en, is_read, map);
    if(!is_read) wr_cg.sample();
  endfunction

endclass : PCR_BG_ASSERT_type

class rf_type extends uvm_reg_block;
   // Automatic SV code from generator

endclass : rf_type
```

Auto Generated PCR Access and Coverage Code

accellera SYSTEMS INITIATIVE

Benefits of PCR Methodology



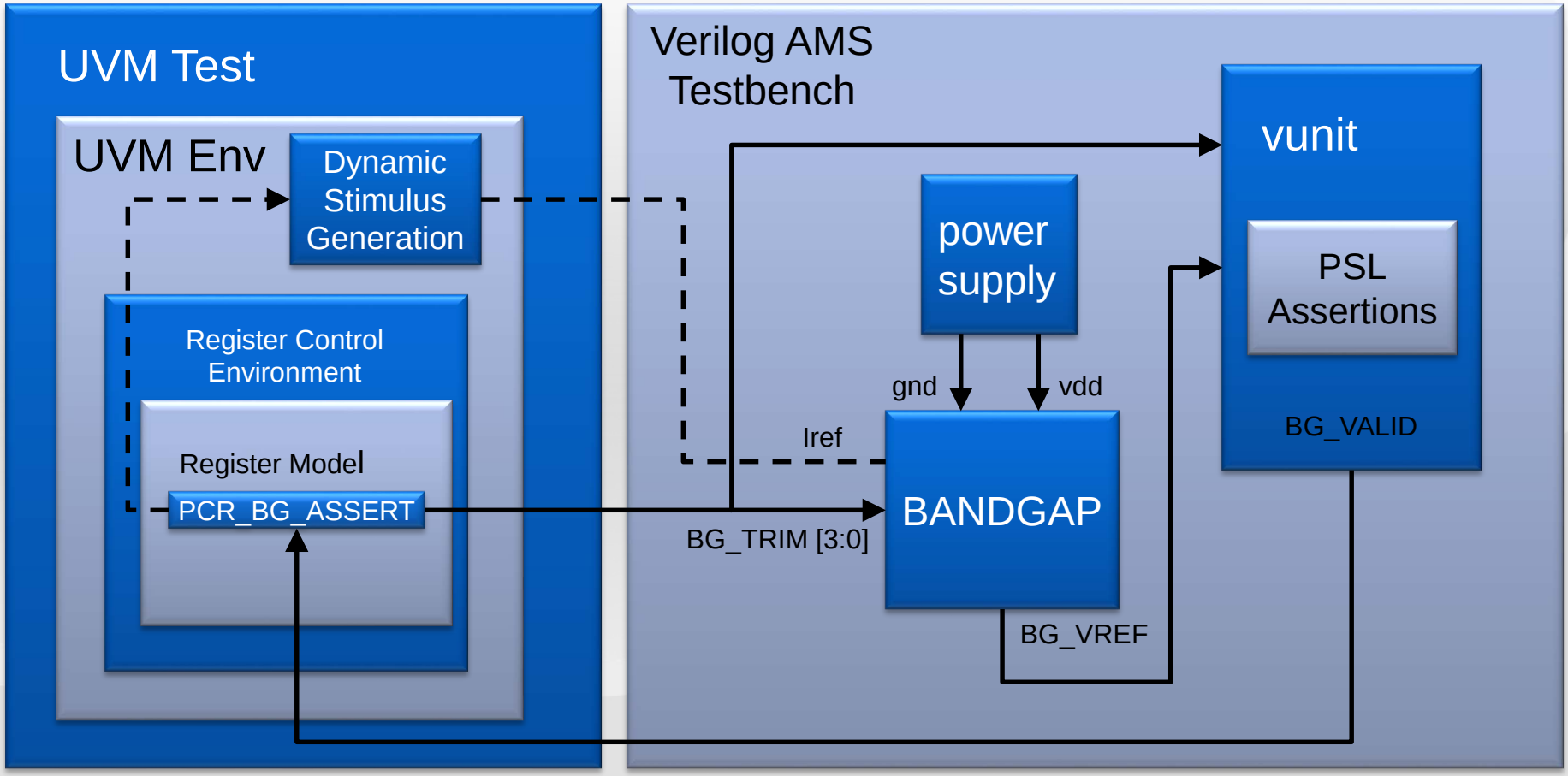
- Leverages UVM REG Methodologies
- PCRs defined in excel spreadsheets

- Early coverage metrics available
- Secondary Cross Coverage (e.g. A/D, D/A)
- Grouping of related assertions

- HDL paths can be maintained in spreadsheets matching design hierarchy
- Test suite independent of HDL paths

- Test stimulus can be modified realtime based on feedback from the PCRs for corner case testing

Band Gap Testbench Example



This band gap reference has a 4 input trim that with a 10mV range to vary VREF from 1.25V to 1.40V.

This example illustrates how to use the PCR methodology to verify a simple band gap voltage reference.

Stimulus and Coverage Test Cases Example

```
class bg_test extends uvm_test;
  uvm_status_e status;
  uvm_reg regs[$];
  bit [7:0] rval; bit [7:0] data;

  top_env env;

  …. Standard UVM overhead …
  task run_phase(uvm_phase phase);
    super.run_phase(phase);

    phase.raise_objection(this);
    // Set HDL path for backdoor register access
    env.reg_model.rf.set_hdl_path_root("tb_top.dut");

    // Set PCR_BG_ASSERT HDL Path
    env.reg_model.rf.PCR_BG_ASSERT.add_hdl_path('{ '{"BG_TRIM",  0, 4},
                                                   '{"BG_VALID", 4, 1} } );

    env.reg_model.rf.PCR_BG_ASSERT.set_coverage(UVM_CVR_REG_BITS);

    for(int trim=0;trim<=15;trim++) begin
      env.reg_model.rf.PCR_BG_ASSERT.poke(status, trim);
      #10ns; // Allow time for BG output to stablize
      env.reg_model.rf.PCR_BG_ASSERT.peek(status, rval);
      env.reg_model.rf.sample_values();
      #1us;
    end

    phase.drop_objection(this);
  endtask:run_phase
endclass:bg_test
```

Requires uvm_status and data declaration

hdl_path defined for PCR data

This code represents the basic test case methodology for sampling or depositing data in the PCR for coverage collection and stimulus generation using UVM REG zero time access tasks.

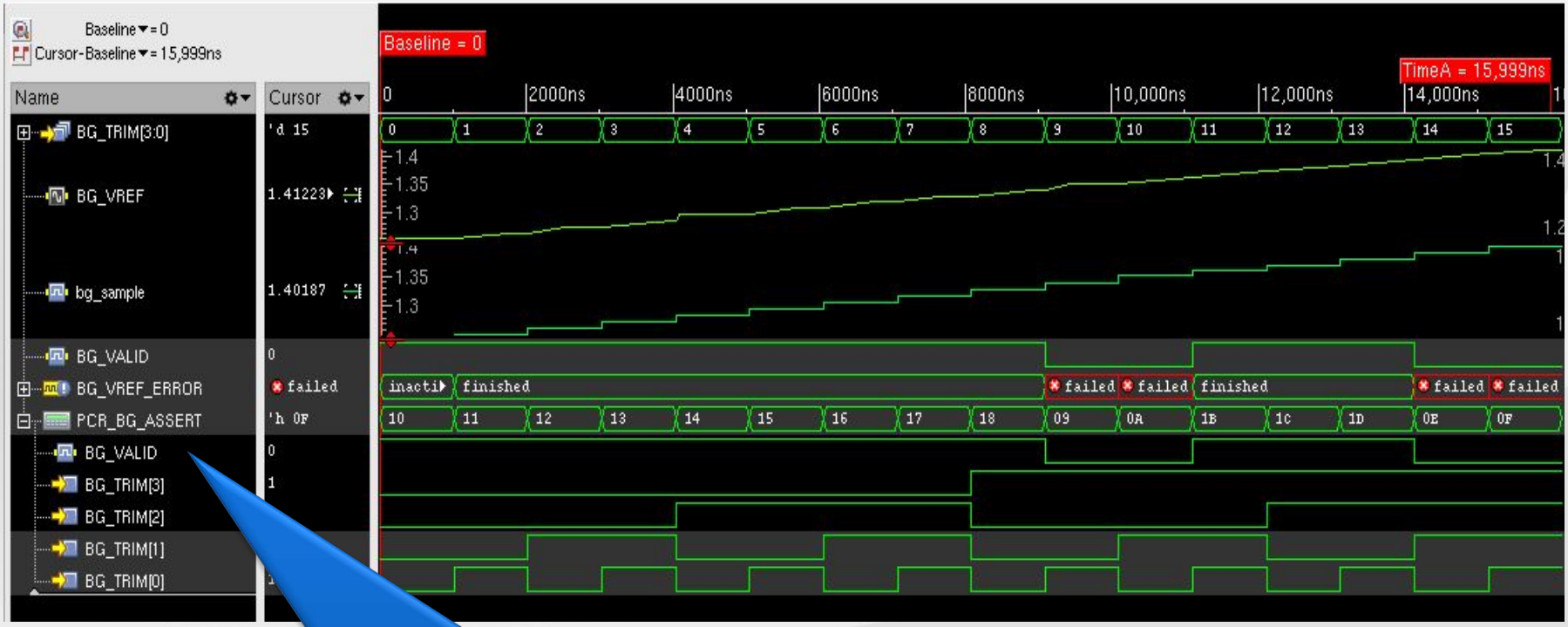
Stimulus generation performed via “poke()” and coverage collected via “peek()” and “sample_data()” PCR tasks

Band Gap Under Test

Notice that assertion failed for BG_TRIM = 9, 10, 14 and 15

accellera SYSTEMS INITIATIVE

Band Gap Under Test with PCR



PCR also captures assertion failure,
but now allows secondary
cross coverage to be collected between
BG_TRIM and BG_VALID signals!

accellera SYSTEMS INITIATIVE

Simulation Results Log File

BG_TRIM value poked into bits 0-3 of PCR

"Peek" reads back 1'b1 in bit 4 indicating BG output is valid

UVM_INFO@10100000: reporter [RegModel]	Poked register "reg_model.rf.PCR_BG_[illegible]": 'h0000000000000001
UVM_INFO@10200000: reporter [RegModel]	Peeked register "reg_model.rf.PCR_BG_ASSERT": 'h0000000000000011
UVM_INFO@20200000: reporter [RegModel]	Poked register "reg_model.rf.PCR_BG_ASSERT": 'h0000000000000002
UVM_INFO@20200000: reporter [RegModel]	Peeked register "reg_model.rf.PCR_BG_ASSERT": 'h0000000000000012
UVM_INFO@101000000: reporter [RegModel]	Poked register "reg_model.rf.PCR_BG_ASSERT": 'h000000000000000a
ncsim: *E,ASRTST (../tb/vunit.pslvlog,11): (time 10100 NS) Assertion tb_top.dut.BG_VREF_ERROR has failed BG VREF FAILURE	
UVM_INFO@101100000: reporter [RegModel]	Peeked register "reg_model.rf.PCR_BG_ASSERT": 'h000000000000000a

Assertion Fails and "Peek" reads back 1'b0 in bit 4 indicating BG output is invalid

Dynamic Stimulus Generation

BG_VALID signal is used to adjust current reference when failure occurs

BG_VREF now passes where it previously failed without Dynamic Adjustment!

accellera SYSTEMS INITIATIVE

Coverage Collection

By creating a single PCR that contains both BG_TRIM and BG_VALID, cross coverage can be collected by adding a cross statement to the auto generated coverage code.

Auto Generated Covergroup

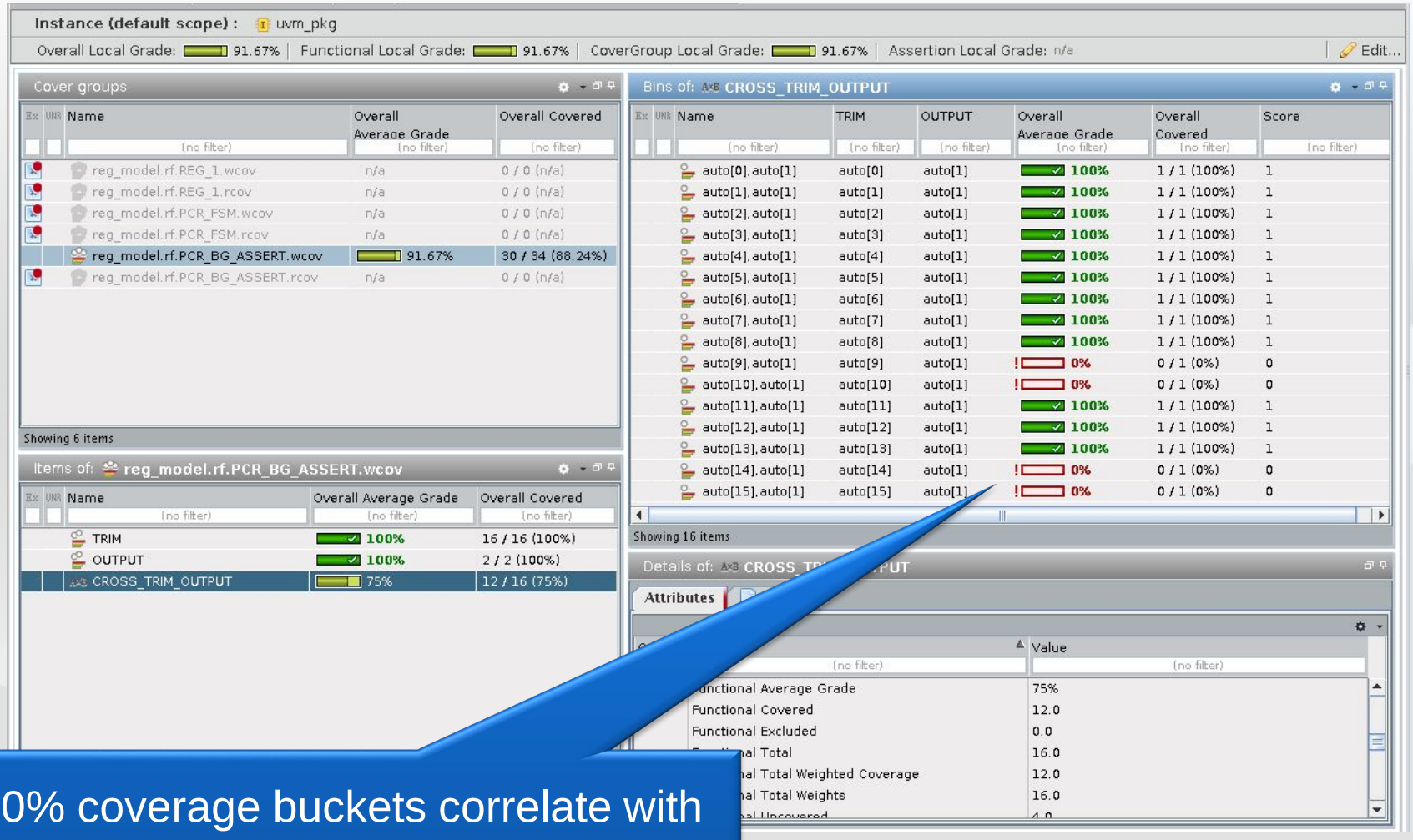
PCR_BG_ASSERT				
BG_VALID	BG_TRIM			
Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

BG_VALID coverage is updated when sample_data() is called in PCR_BG_ASSERT PCR

```
covergroup wr_cg;
   TRIM : coverpoint TRIM.value[3:0];
   BG_VALID : coverpoint BG_VALID.value[0:0];
   CROSS_TRIM_BG_VALID: cross TRIM, BG_VALID;
endgroup
```

Cross coverage statement added to auto generated coverage code.

PCR Cross Coverage Results



0% coverage buckets correlate with Simulation Failures!

accellera SYSTEMS INITIATIVE

Conclusion

PCRs can be created in the design planning stage which provides early and accurate coverage metrics

PCRs allow for easier management of assertion based coverage collection

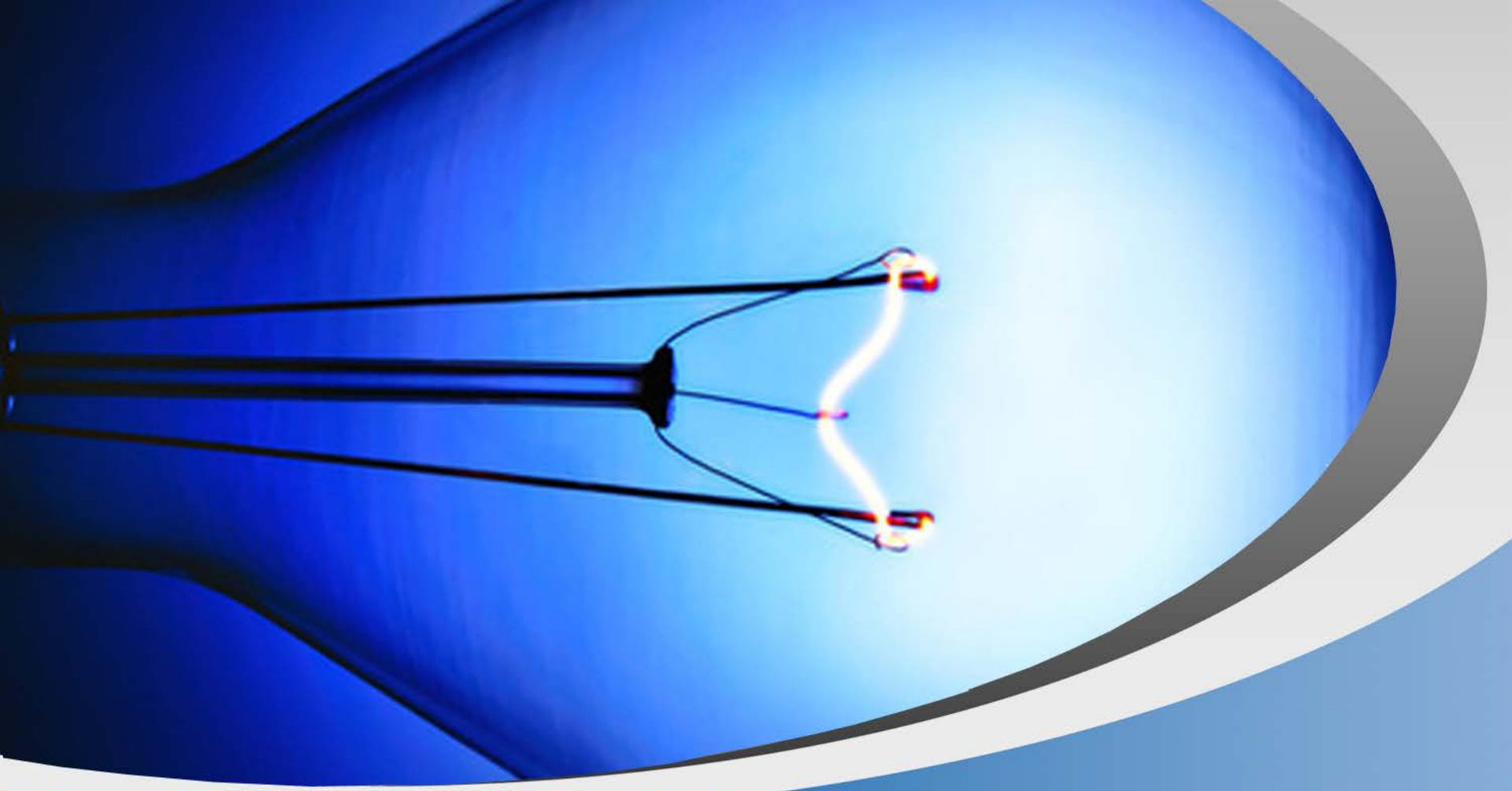
PCRs allows Dynamic Stimulus Generation for critical corner case generation

PCRs leverage existing DV methodologies to make them more efficient and reusable

Contributors

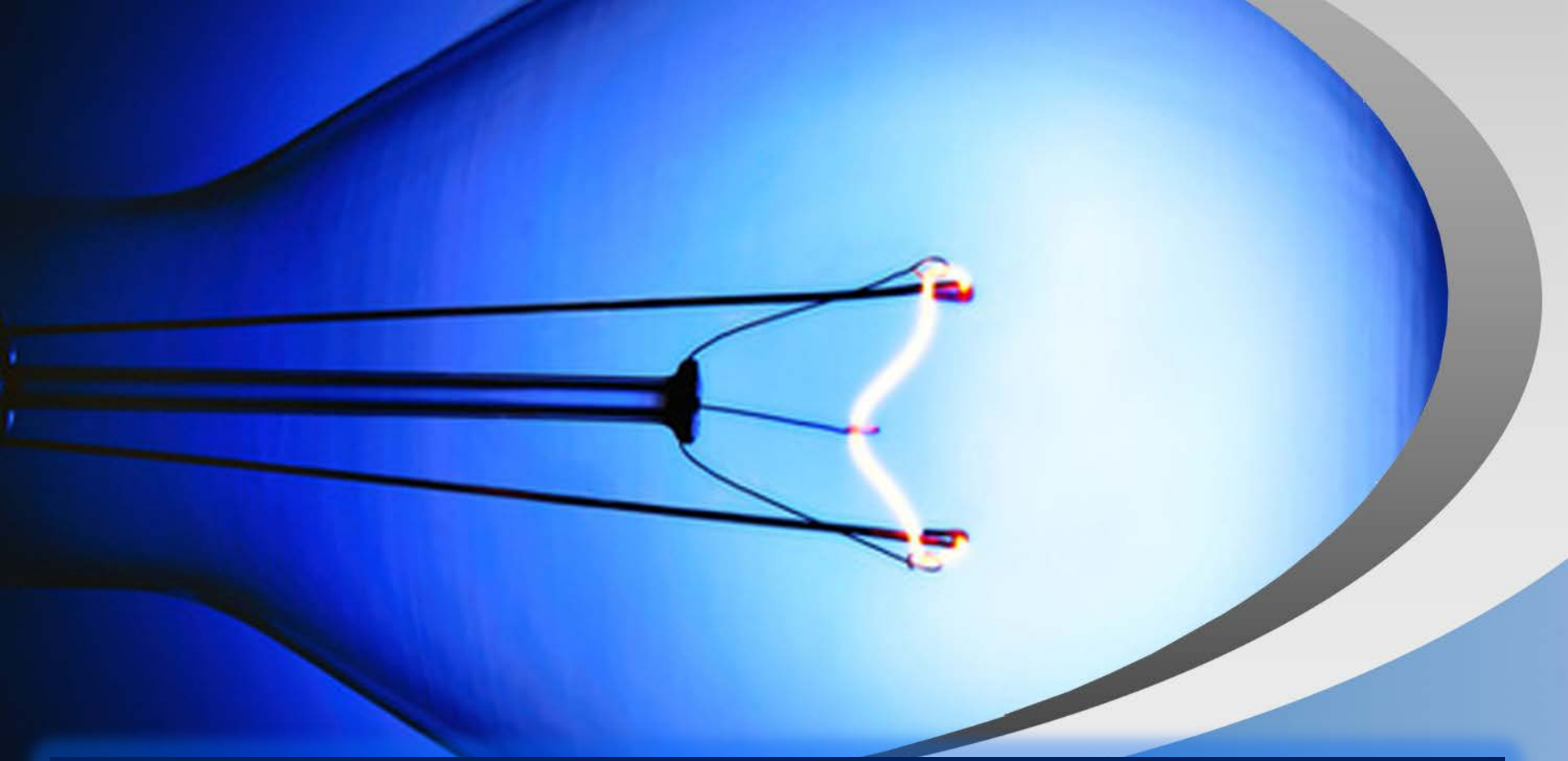
- Asad Khan (MGTS)
- Ravi Makam
- Zhipeng Ye
- Jonathan King
- Paul Howard

All the Business Unit folks who got us to this point!



Thank you!

accellera
SYSTEMS INITIATIVE



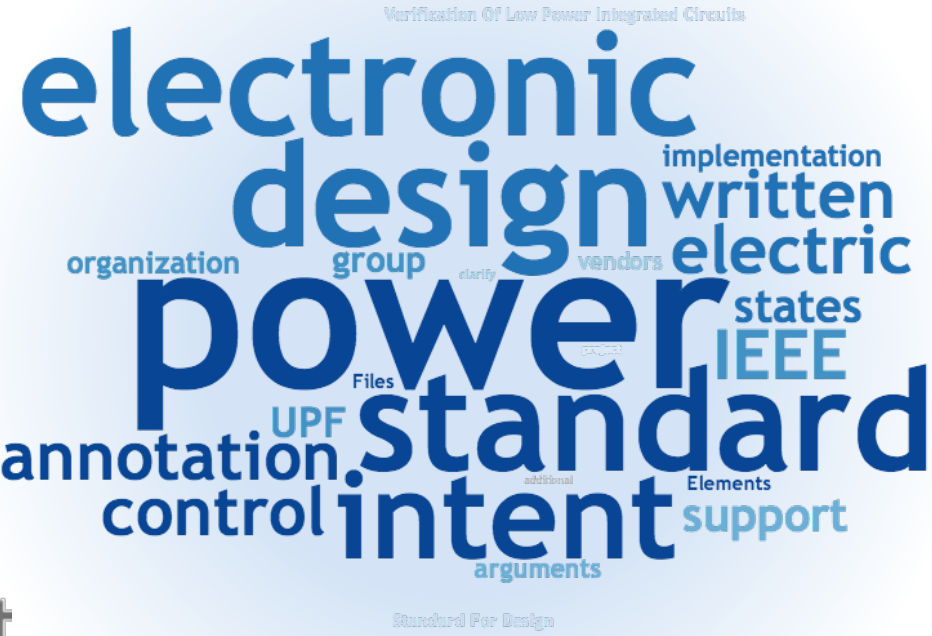
Next Generation Design and Verification Today

New Developments in UPF 3.0

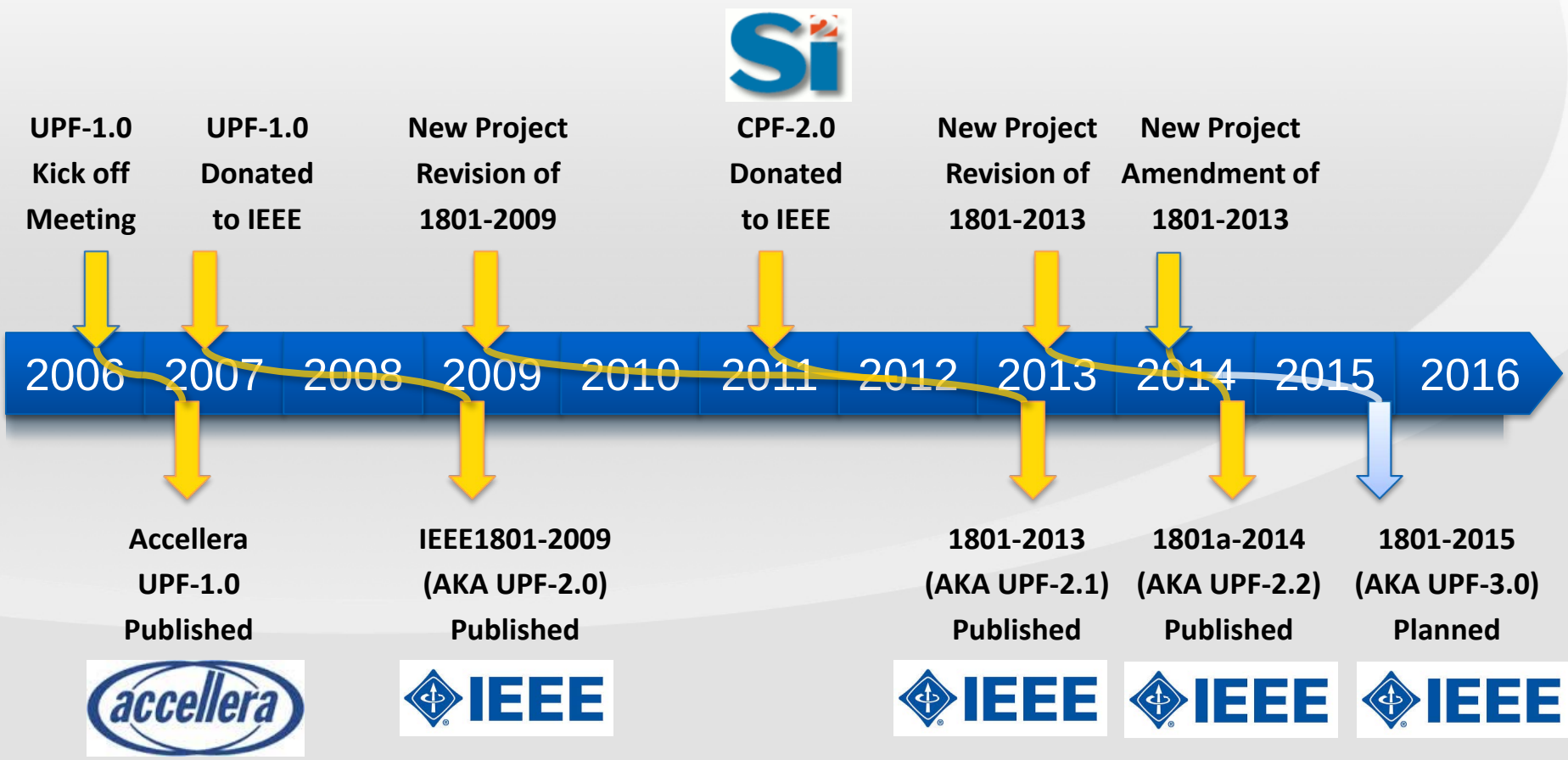
Erich Marschner, Vice-Chair, IEEE P1801 WG

accellera
SYSTEMS INITIATIVE

P1801: IEEE-SA Entity Based Work Group



IEEE 1801 (UPF) timeline

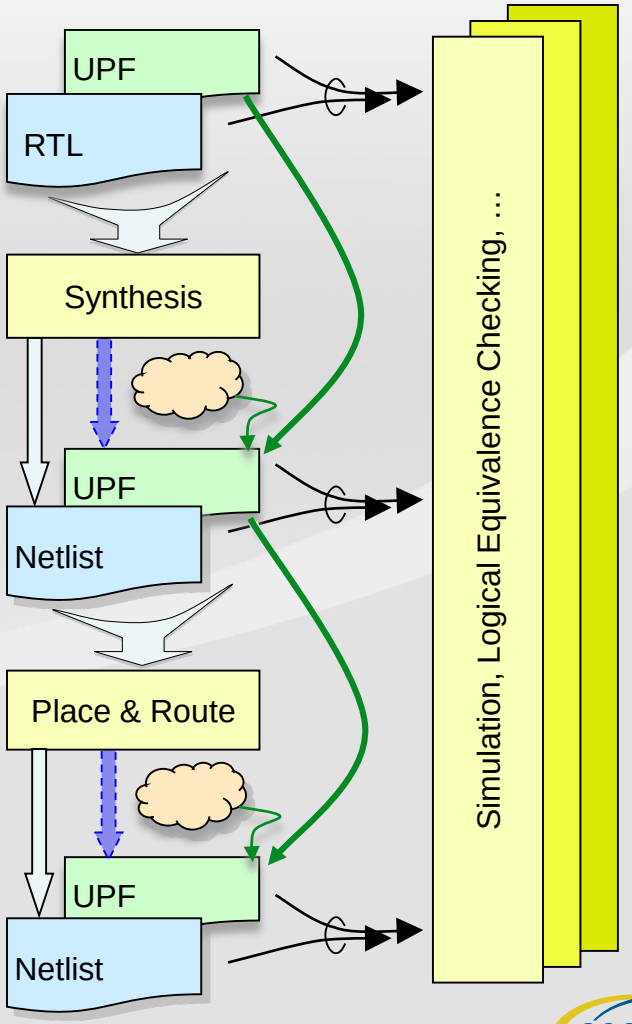


Agenda

- **Successive Refinement**
  - Elaborating the UPF 2.0 Concept
- **Power State Definition and Refinement**
  - Power State Definition with add_power_state
  - Power State Composition
- **Component Level Power Modeling**
  - Power States and Power Consumption Functions

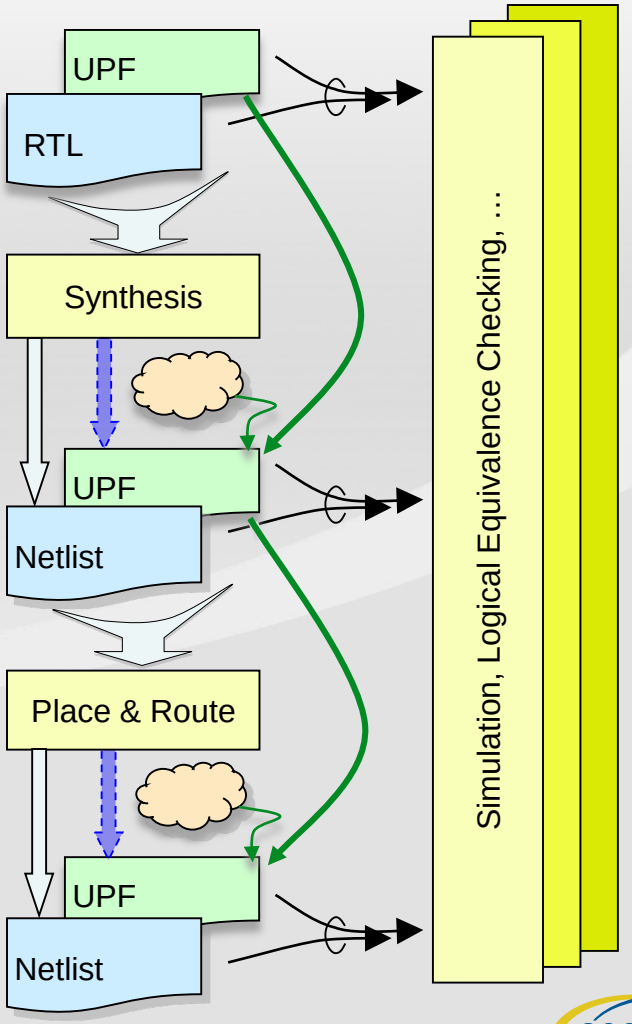
UPF 1.0 Design Flow

- **RTL is augmented with UPF**
  - To define power management architecture
- **RTL + UPF verification**
  - To ensure that power architecture completely supports planned power states of design
  - To ensure that design works correctly under power management
- **RTL + UPF implementation**
  - Synthesis, test insertion, place & route, etc.
  - UPF may be updated by user or tool
- **NL + UPF verification**
  - Power aware equivalence checking, static analysis, simulation, emulation, etc.



UPF 1.0 Flow Issues

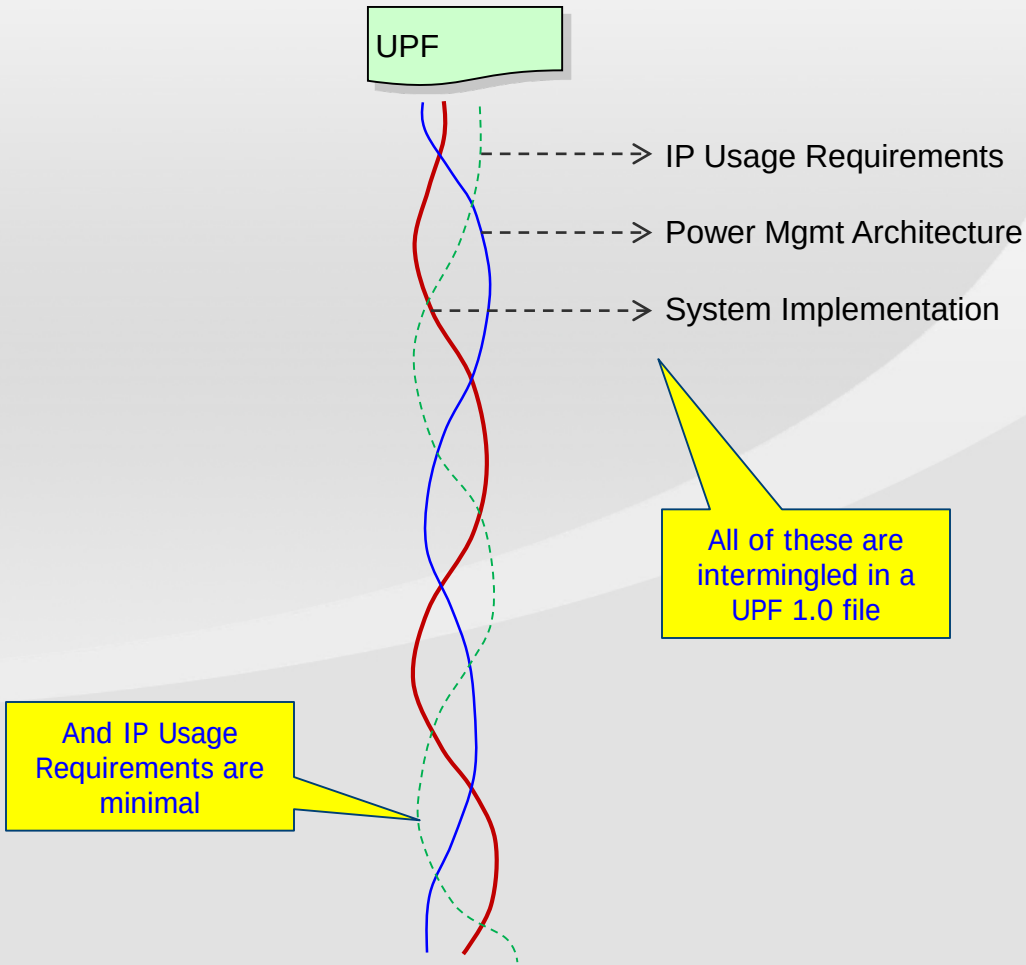
- **Power Aware Verification requires complete supply distribution network**
  - Supplies determine when each power domain is on (normal) or off (corrupted)
- **Supply networks are not defined until system implementation**
  - Part of integrating the whole system together
- **So power aware verification cannot begin until implementation is specified**
  - Limits how much the schedule can be shortened by parallel development
  - Must be redone entirely if the design is retargetted to a different technology
- **And debugging power management issues becomes more difficult**
  - Is a failure due to
    - Incorrect implementation?
    - A power management architecture flaw?
    - Misuse of an IP block?
    - Some combination of the above?



accellera SYSTEMS INITIATIVE

UPF 1.0 Power Intent Specification

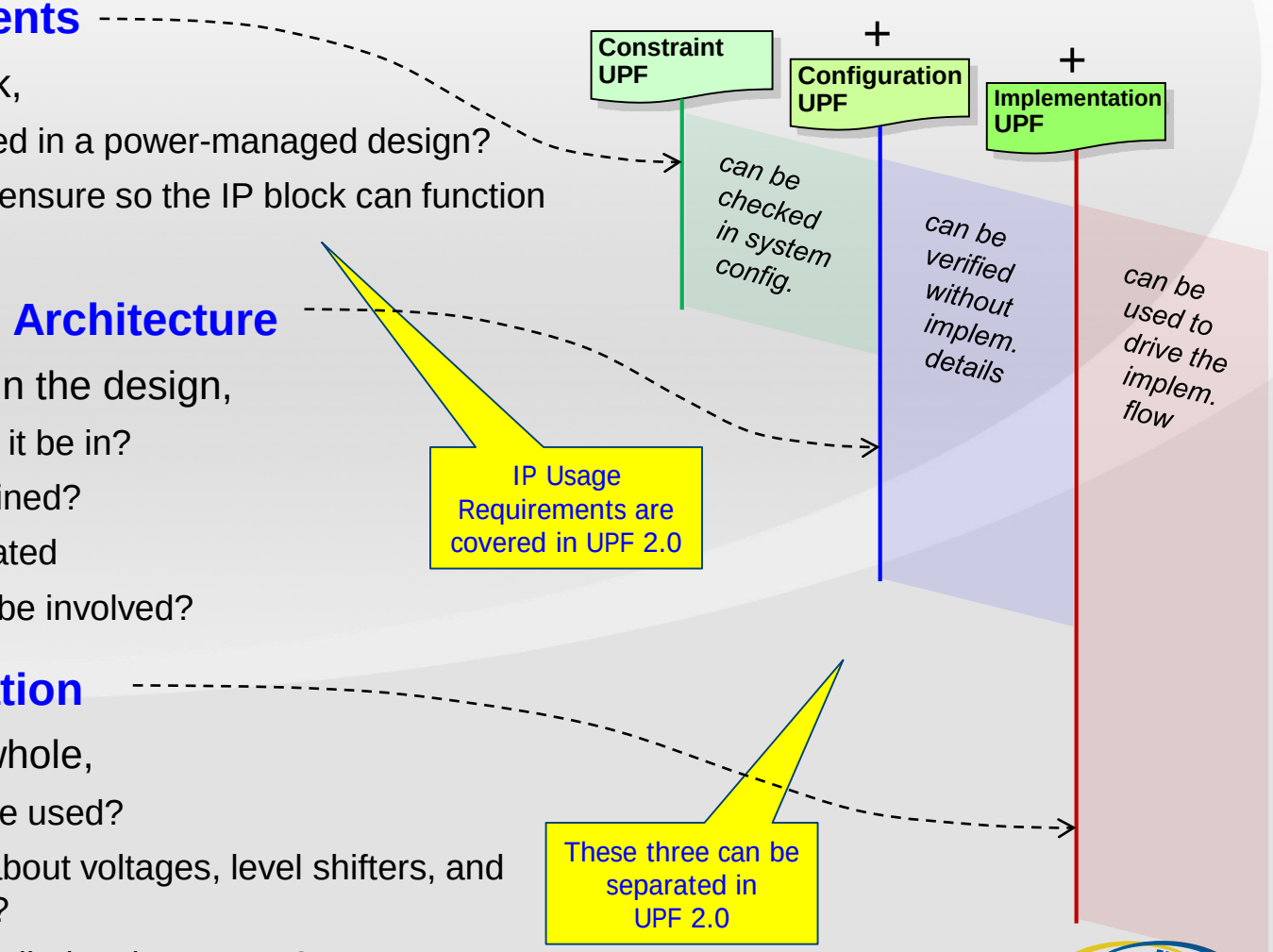
- **Power Domain definitions**
  - elements
  - supply connections
- **Supply Ports and Supply Nets**
  - and their connections
- **Power Switches**
  - supply connections
  - control inputs
- **Isolation Strategies**
  - clamp values
  - supply connections
  - control inputs
- **Level Shifting Strategies**
  - supply connections
- **Retention Strategies**
  - supply connections
  - control inputs
- **Port States**
  - states
  - voltages
- **Power State Tables (PSTs)**
  - combinations of port states



accellera SYSTEMS INITIATIVE

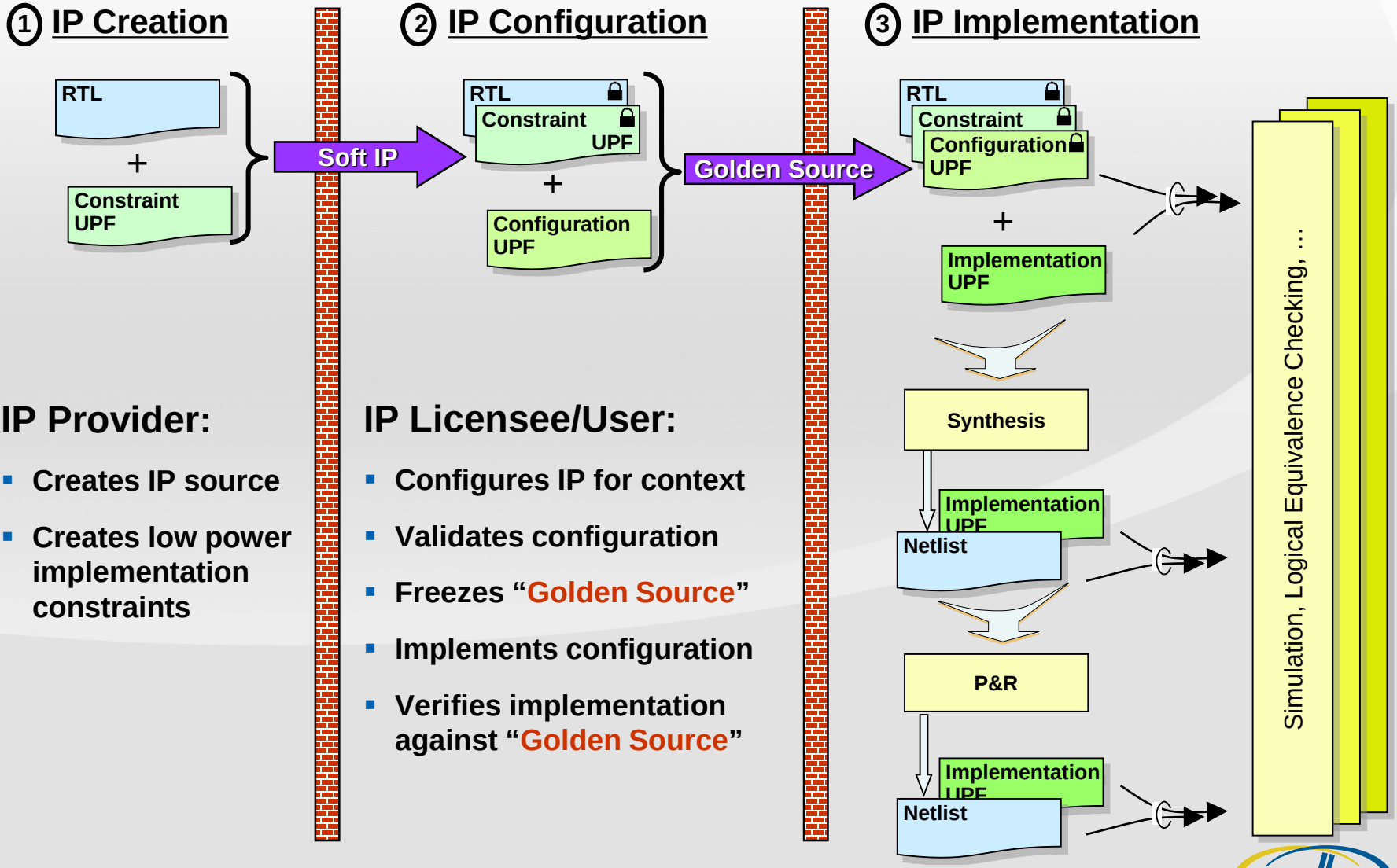
Solution: Partition UPF into Layers

- **IP Usage Requirements**
  - For any given IP block,
    - How can this IP be used in a power-managed design?
    - What must the design ensure so the IP block can function correctly?
- **Power Management Architecture**
  - For each IP instance in the design,
    - What power states will it be in?
    - What state will be retained?
    - What ports will be isolated
    - What control logic will be involved?
- **System Implementation**
  - For the system as a whole,
    - What technology will be used?
    - What does this imply about voltages, level shifters, and isolation cell locations?
    - How will power be supplied to the system?



accellera SYSTEMS INITIATIVE

Successive Refinement of Power Intent



IP Provider:

- Creates IP source
- Creates low power implementation constraints

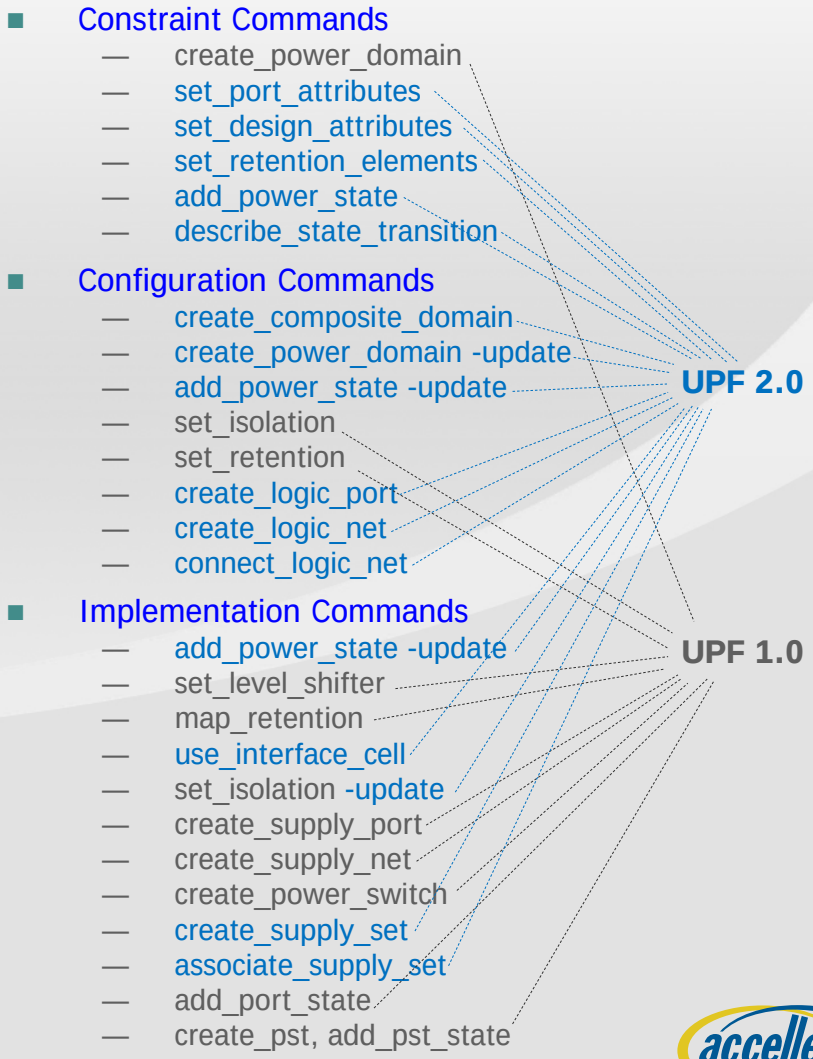
IP Licensee/User:

- Configures IP for context
- Validates configuration
- Freezes "Golden Source"
- Implements configuration
- Verifies implementation against "Golden Source"

accellera SYSTEMS INITIATIVE

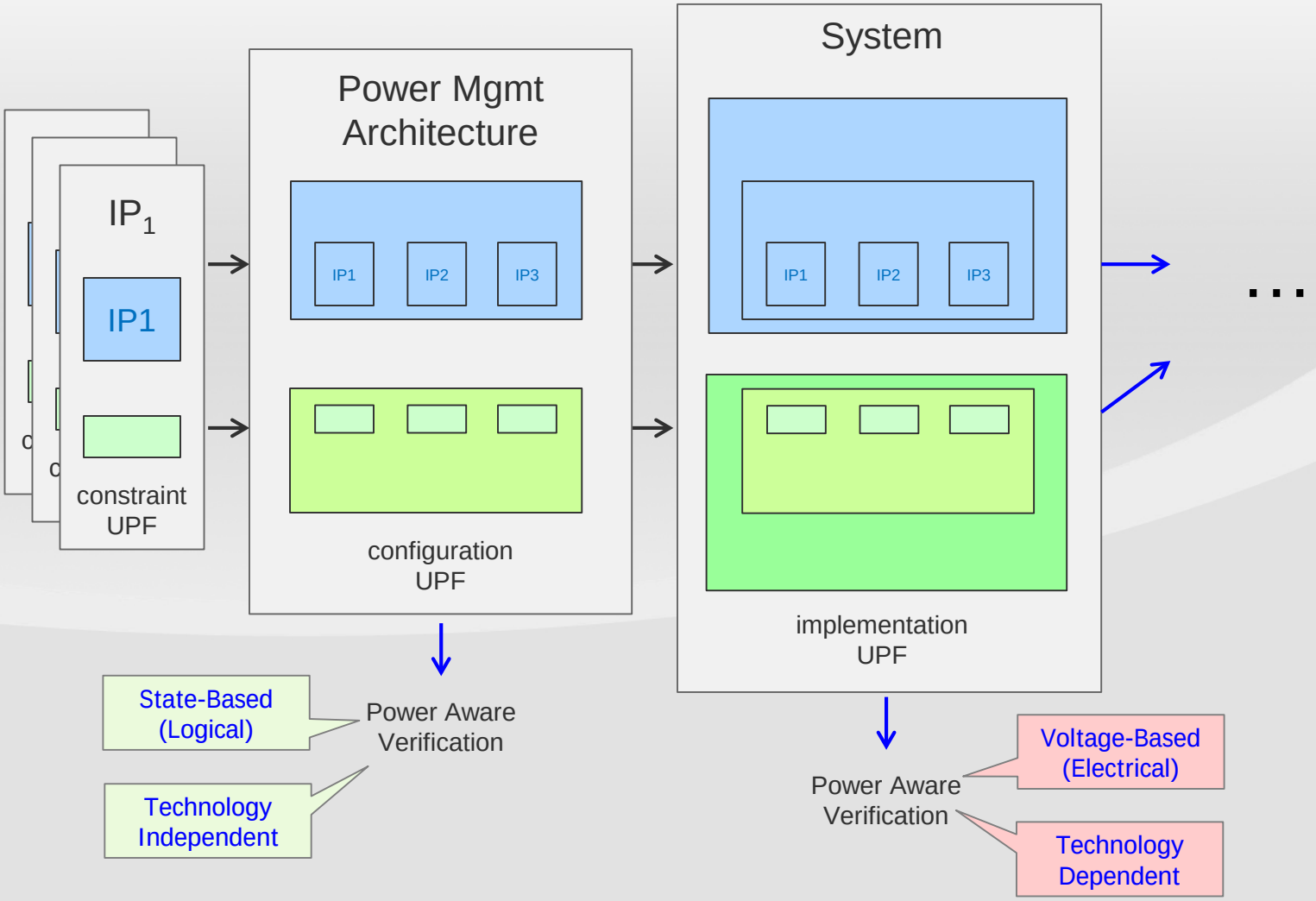
UPF Command Layers

- Constraint UPF
  - Atomic power domains
  - Clamp value requirements
  - Retention requirements
  - Fundamental power states
  - Legal/illegal states/transitions
- Configuration UPF
  - Actual power domains
  - Additional domain supplies
  - Additional power states
  - Isolation and Retention strategies
  - Control signals for power mgmt
- Implementation UPF
  - Voltage updates for power states
  - Level Shifter strategies
  - Mapping to Library power mgmt cells
  - Location updates for Isolation
  - Supply ports, nets, switches, and sets
  - Port states and Power state tables



accellera SYSTEMS INITIATIVE

Incremental Verification



accellera SYSTEMS INITIATIVE

Agenda

- **Successive Refinement**
  - Elaborating the UPF 2.0 Concept
- **Power State Definition and Refinement**
  - Power State Definition with add_power_state
  - Power State Composition
- **Component Level Power Modeling**
  - Power States and Power Consumption Functions

What is a “Power State” ?

A named set of object states

- Each state has a “defining expression”
- It refers to values of the object’s “characteristic elements”
- Some characteristic elements may be don’t cares for a given state
- Multiple object states may satisfy the defining expression

S1: A==1’b0 && B==1’b0

S2: (A xor B) ==1’b1

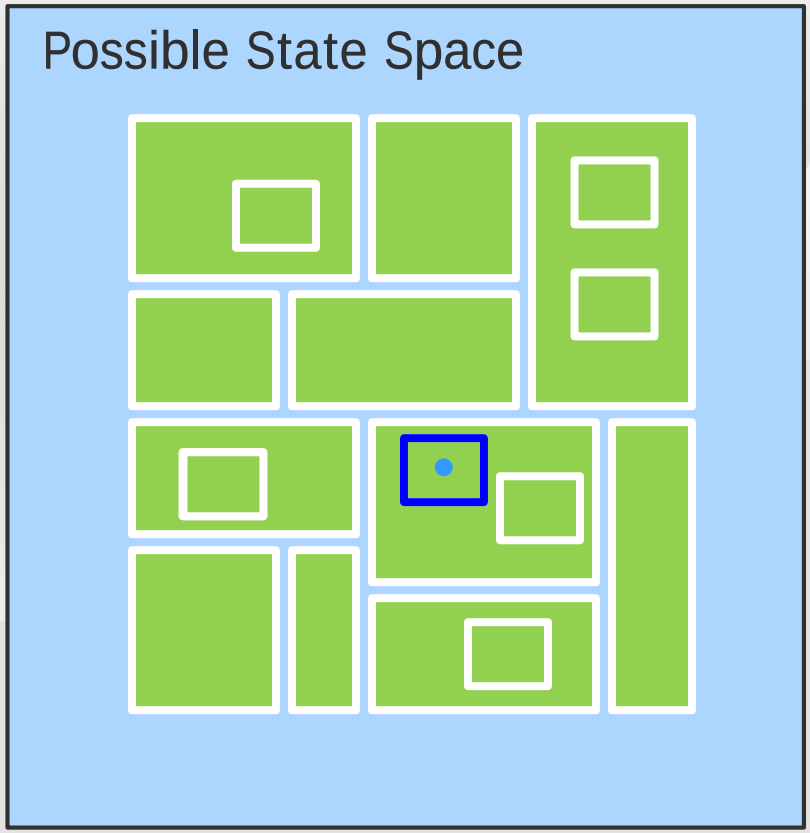
S3: A==1’b1 && B==1’b1

A	B	C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

don’t cares

Power States as Sets

- Largest set = all possible object states
- Some of these states are legal states
- Subsets represent “more specific” (or more refined) power states
  - Refinement creates subsets by adding more conditions to satisfy
  - The innermost subset containing a given object state represents the most specific power state of that object
- Supersets represent “more general” (or more abstract) power states
- Non-overlapping subsets represent mutually exclusive power states
- Subset containment implies non-mutex power states (subset => superset)



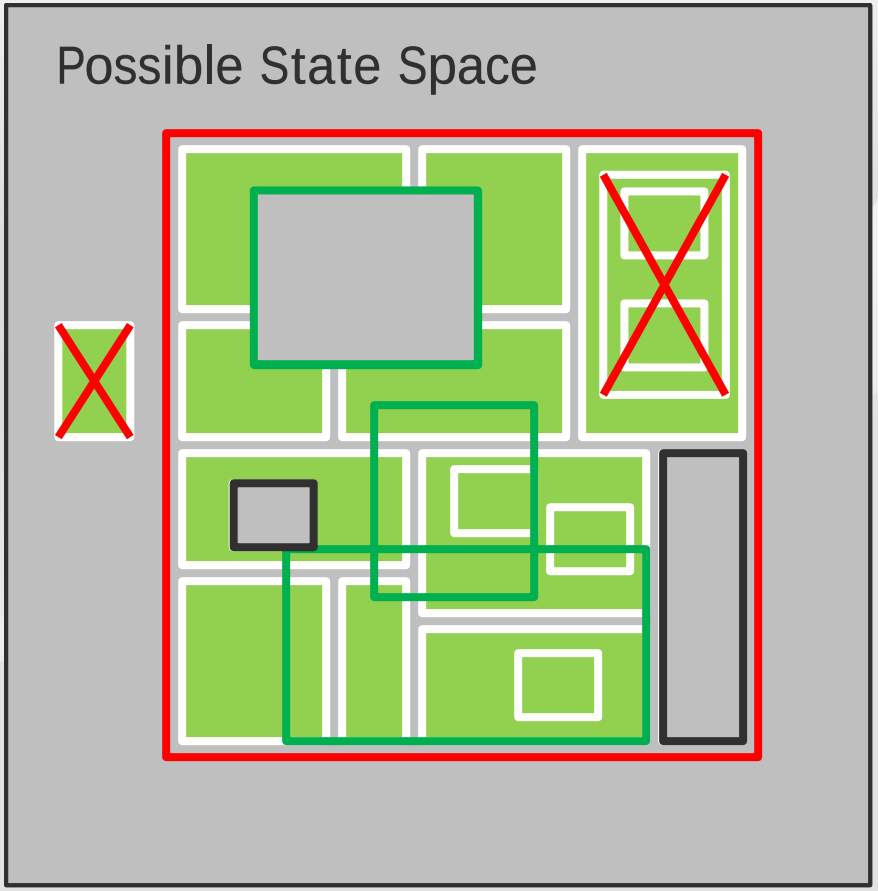
Power State Definition Rules

You can:

- Define (legal) states
- Define explicitly illegal states
- Specify -complete to make undefined states illegal
- Define Definite subset states (existing state AND new condition)
- Define Indefinite superstates ([X]OR of existing states)
- Mark existing legal states illegal

You cannot:

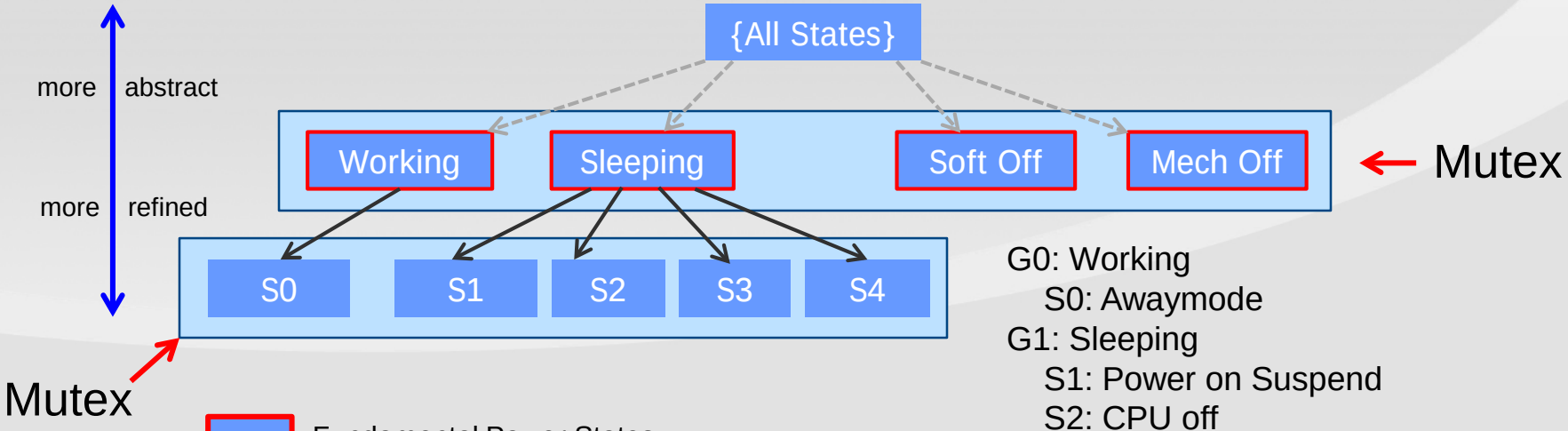
- Create legal states in illegal state space
- Define superstates that are the AND of two or more existing states



accellera SYSTEMS INITIATIVE

Applying These Concepts

- Same level states must be mutually exclusive
- Superstates contain (overlap) substates - non-mutex
- These principles allow state partitioning, hierarchical refinement

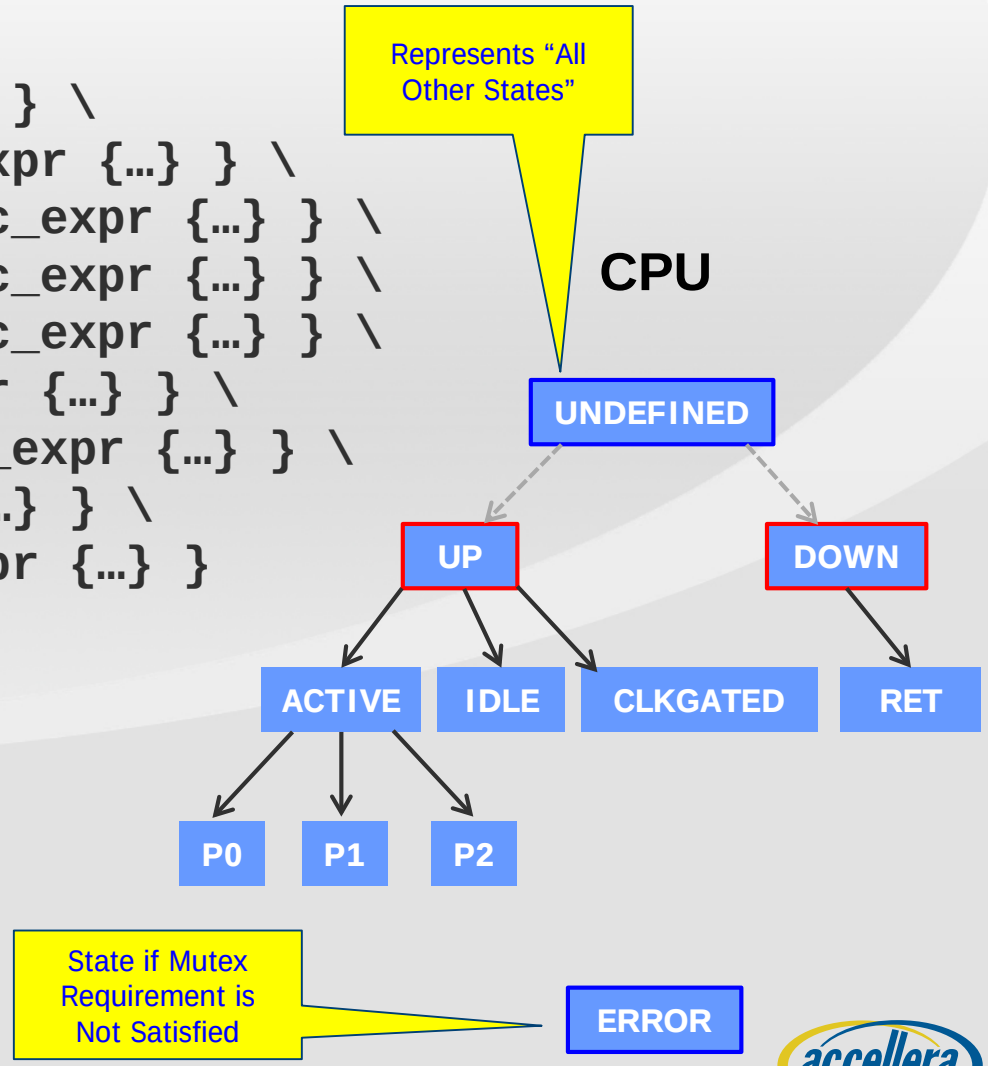


{All States} represents the set of all possible states;
the fundamental states are subsets of {All States}

G0: Working
  S0: Awaymode
G1: Sleeping
  S1: Power on Suspend
  S2: CPU off
  S3: Standby
  S4: Hibernation
G2: Soft Off
G3: Mech. Off

Defining Hierarchical Power States

```
add_power_state -model CPU
  -state {UP -logic_expr {…} } \
  -state {UP.ACTIVE -logic_expr {…} } \
  -state {UP.ACTIVE.P0 -logic_expr {…} } \
  -state {UP.ACTIVE.P1 -logic_expr {…} } \
  -state {UP.ACTIVE.P2 -logic_expr {…} } \
  -state {UP.IDLE -logic_expr {…} } \
  -state {UP.CLKGATED -logic_expr {…} } \
  -state {DOWN -logic_expr {…} } \
  -state {DOWN.RET -logic_expr {…} }
```



Agenda

- **Successive Refinement**
  - Elaborating the UPF 2.0 Concept
- **Power State Definition and Refinement**
  - Power State Definition with add_power_state
  - Power State Composition
- **Component Level Power Modeling**
  - Power States and Power Consumption Functions

Power State Dependencies

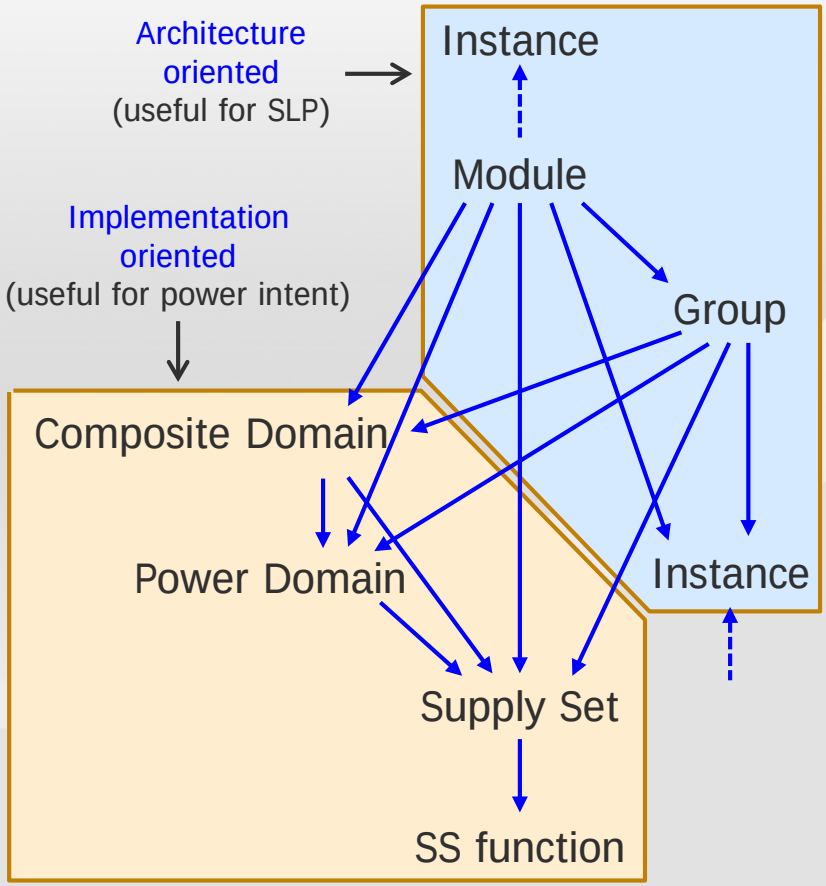
- Instance
  - Functional modes as power states
  - Based on module states
- Module
  - Functional modes as power states
  - Based on component states, control inputs
- Composite Domain
  - Functional modes as power states
  - Based on subdomain states, control inputs
- Power Domain
  - Operational modes as power states
  - Based on supply set states, control inputs
- Supply Set
  - Supply function combinations as power states
  - Based on individual supply function electrical states (and voltages), clock frequency, control inputs

- Supply Function
  - Electrical states/voltages as power states
  - Based upon supply net/port states/voltages
  - Determined also by supply_on/off calls from testbench (for unassociated supply sets)
- Supply Net
  - Electrical states/voltages as power states
  - Based upon supply net/port states/voltages
  - Determined also by supply net resolution (for resolved supply nets)
- Supply Port
  - Electrical states/voltages as power states
  - Determined by supply_on/off calls from testbench (for primary supply inputs)
  - Determined also by power switches (for switch output ports)
  - NOT based on port state definitions
    - no way to refer to them today

Named power states (add_power_state)
Supply states (supply_net_type values)

Power State References

- Supply Set power states
  - can refer to SS function supply states
- Power Domain power states
  - can refer to supply set power states
- Composite Domain power states
  - can refer to subdomain power states and/or supply set power states
- Group power states
  - can refer to power states of any object at or below the same scope
- Module power states
  - can refer to power states of any object at or below the module scope
- Instance power states
  - inherit (upwards) power states of the instantiated module
  - can override legality of a power state for a given instance (make a legal state illegal)

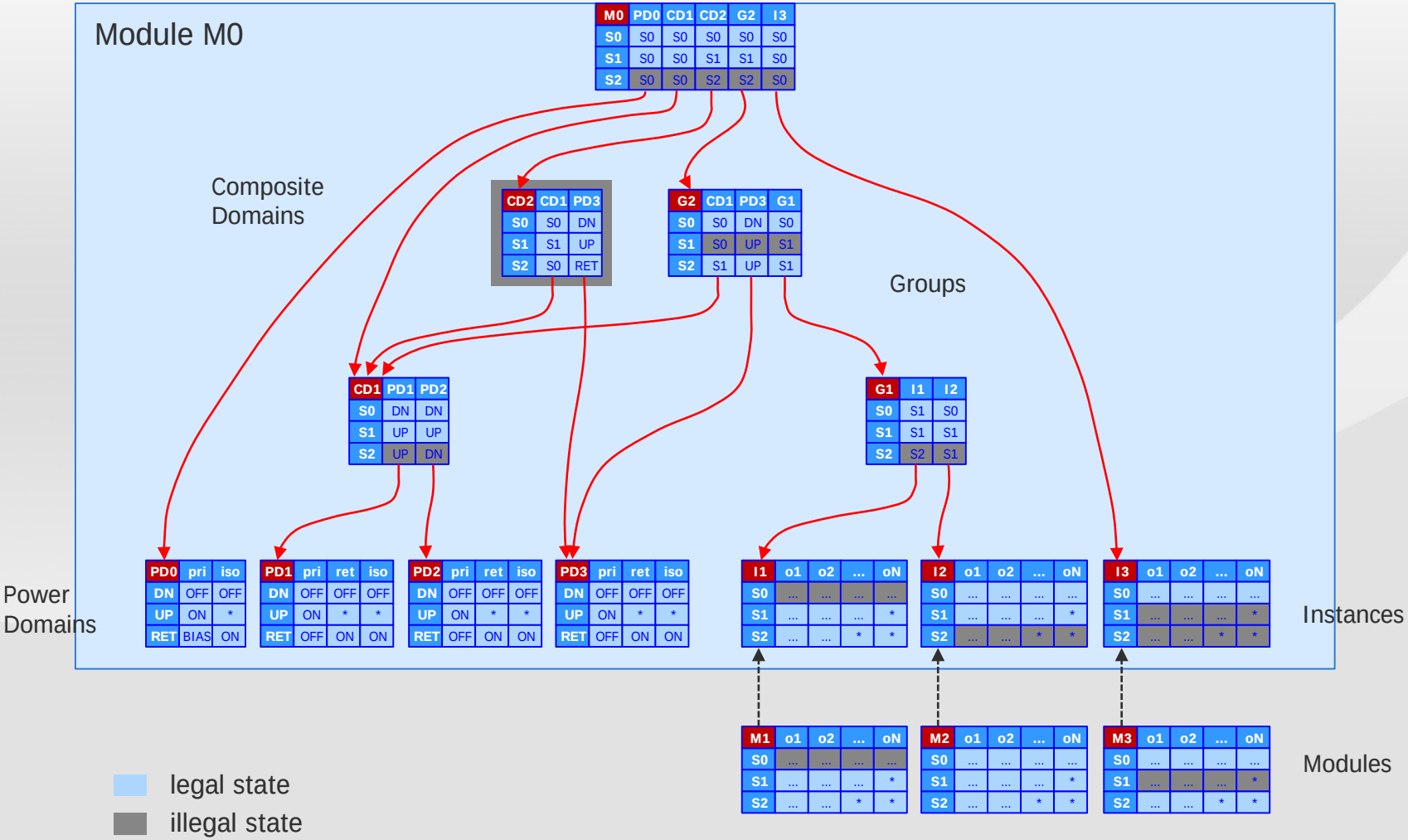


* not showing supply refs to ports/nets or control conditions

Power State Composition

- Fundamental power states of a given object are mutually exclusive
- Power states of two different objects are by default independent
  - All combinations of the legal states of each are legal
- An object that consists of other objects can
  - Define named combinations of the states of its component objects
    - Some of these are fundamental power states and therefore must be mutex
  - Mark a named combination of component objects states as illegal
  - Mark the set of named combinations as complete - which makes all others illegal
  - In particular:
    - supply set states define named combinations of supply set function (supply) states
    - domain states define named combinations of the domain's supply set states
    - composite domain states define named combinations of the subdomain states
- An object that contains other objects can do the same (UPF 3.0)
  - In particular:
    - group power states name combinations of states of objects at/below the group scope
    - module power states name combinations of states of objects in/below the module scope
      - module states become instance states when the module is instantiated
- A legal module state can be marked illegal for a given instance

Example



accellera SYSTEMS INITIATIVE

Agenda

- **Successive Refinement**
  - Elaborating the UPF 2.0 Concept
- **Power State Definition and Refinement**
  - Power State Definition with add_power_state
  - Power State Composition
- **Component Level Power Modeling**
  - Power States and Power Consumption Functions

Energy Consumption Varies w/ Usage

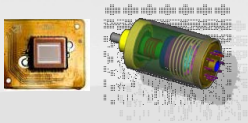
Highly dynamic operation of multiple interacting hardware and software components

Usage Scenarios
Idle
Walking
Driving
Navigating
Talking
Browsing
VideoPlay
AudioPlay

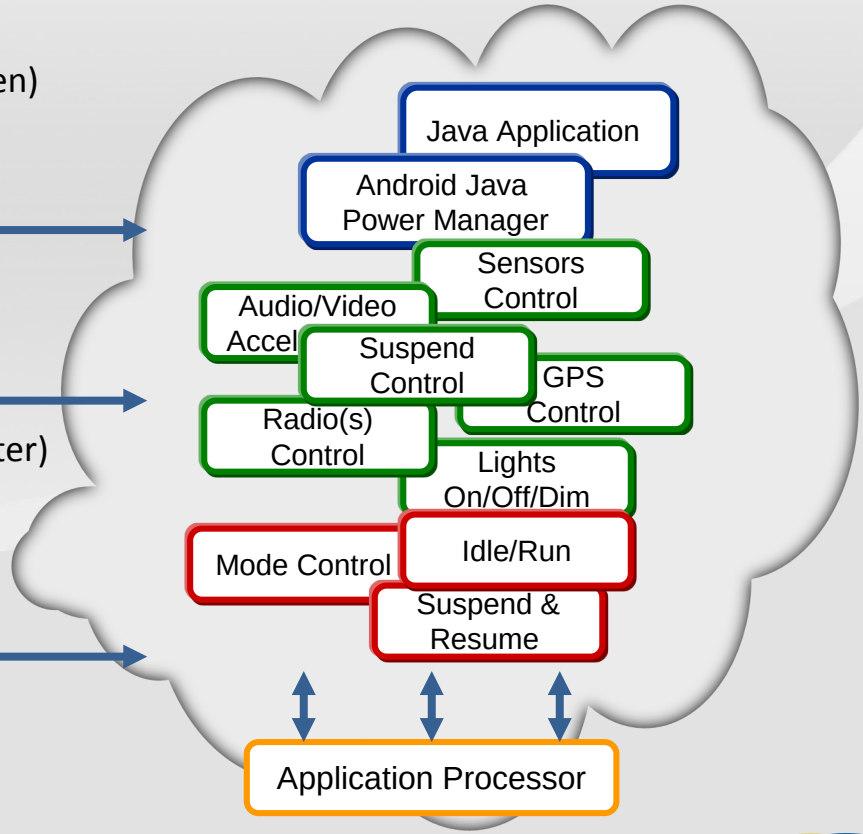
Combined Scenarios:
Navigating/Talking
Browsing/3G
VideoPlay/3G

User Activity
(Keys, Touchscreen)

Sensors
(CCD, Accelerometer)

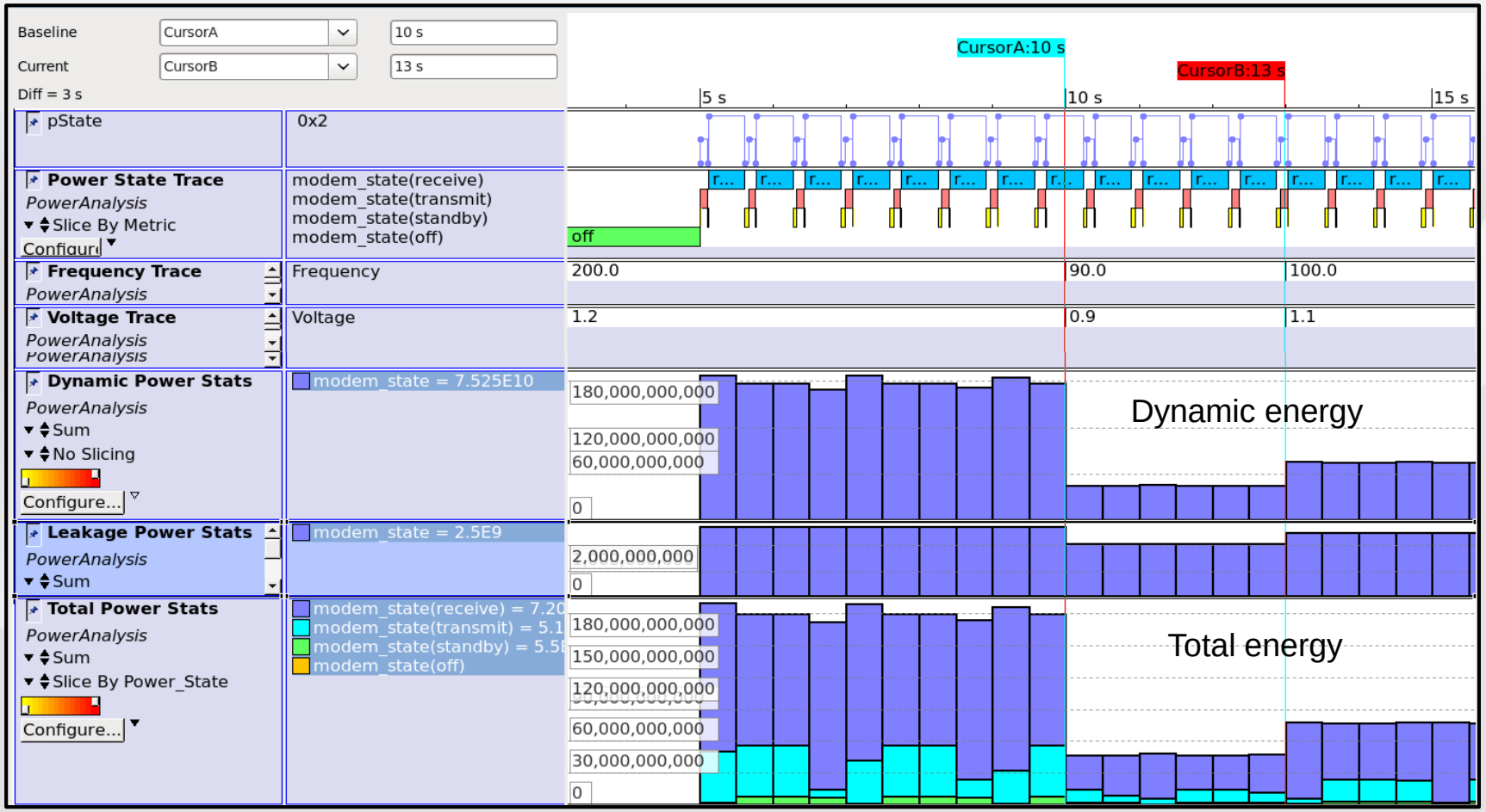


Radios
(GPS, WiFi, GSM)



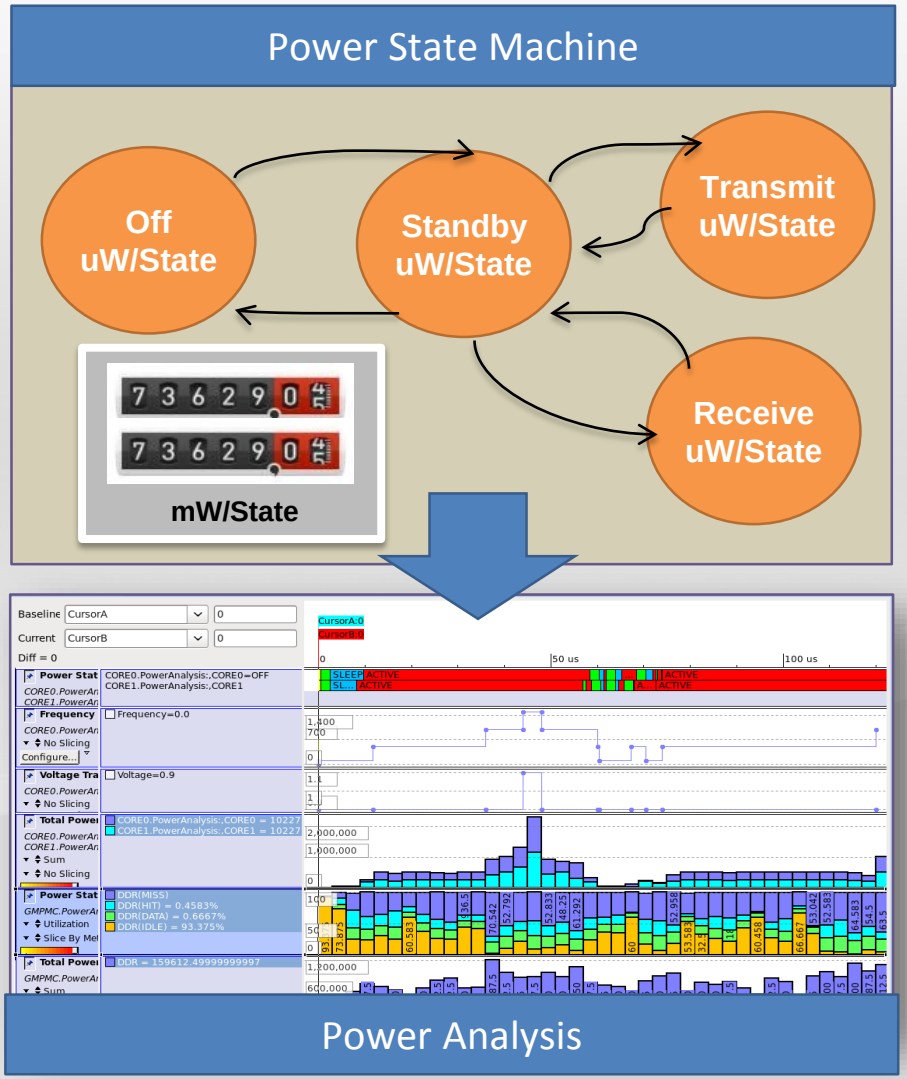
accellera SYSTEMS INITIATIVE

Need to Model Energy Usage



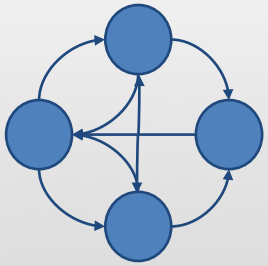
accellera SYSTEMS INITIATIVE

Each State Has Different Power Reqs.



accellera SYSTEMS INITIATIVE

Power Model Components



Power state enumeration

- Steady states
- Transient states (transitions)
- Power dissipation function per state
  - With relevant parameters
    - voltage, frequency, event rates, …
  - Returns Static + Dynamic power
- PVT independent

Addressed in UPF 3.0

Power consumption data

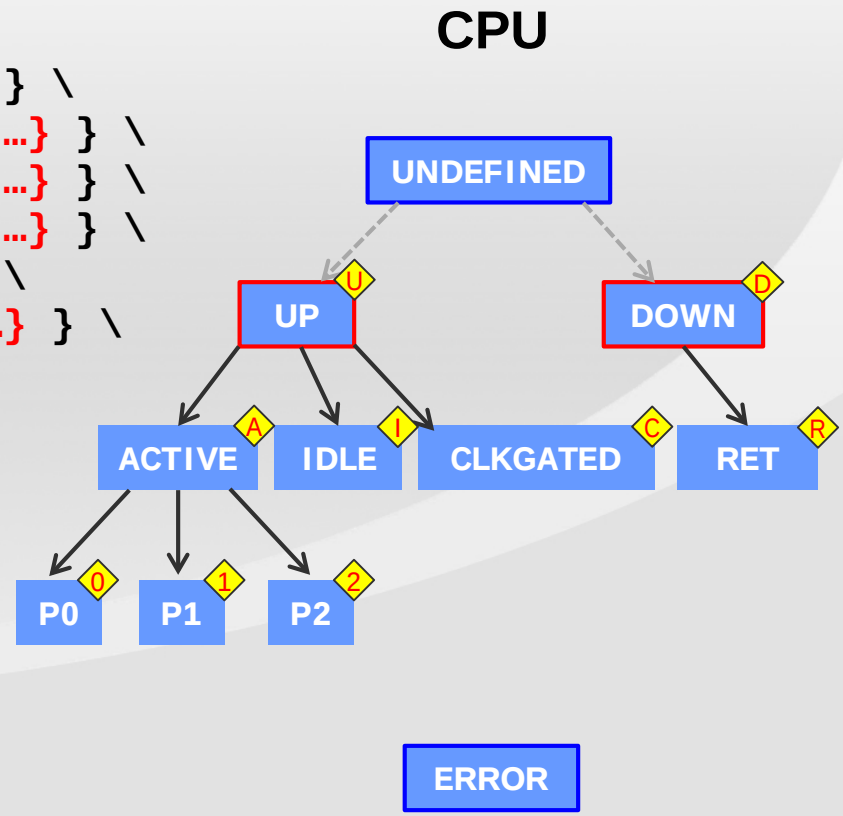
- PVT specific parameters
- Characterized or estimated

Power state activation

- Scenario-based or functional simulation based
- Resolution limits overall accuracy of power model

Modeling Power Consumption in UPF

```
add_power_state -model CPU -update
  -state {UP -power_expr {fU …} } \
  -state {UP.ACTIVE -power_expr {fA …} } \
  -state {UP.ACTIVE.P0 -power_expr {f0 …} } \
  -state {UP.ACTIVE.P1 -power_expr {f1 …} } \
  -state {UP.ACTIVE.P2 -power_expr {f2 …} } \
  -state {UP.IDLE -power_expr {fI …} } \
  -state {UP.CLKGATED -power_expr {fC …} } \
  -state {DOWN -power_expr {fD …} } \
  -state {DOWN.RET -power_expr {fR …} }
```

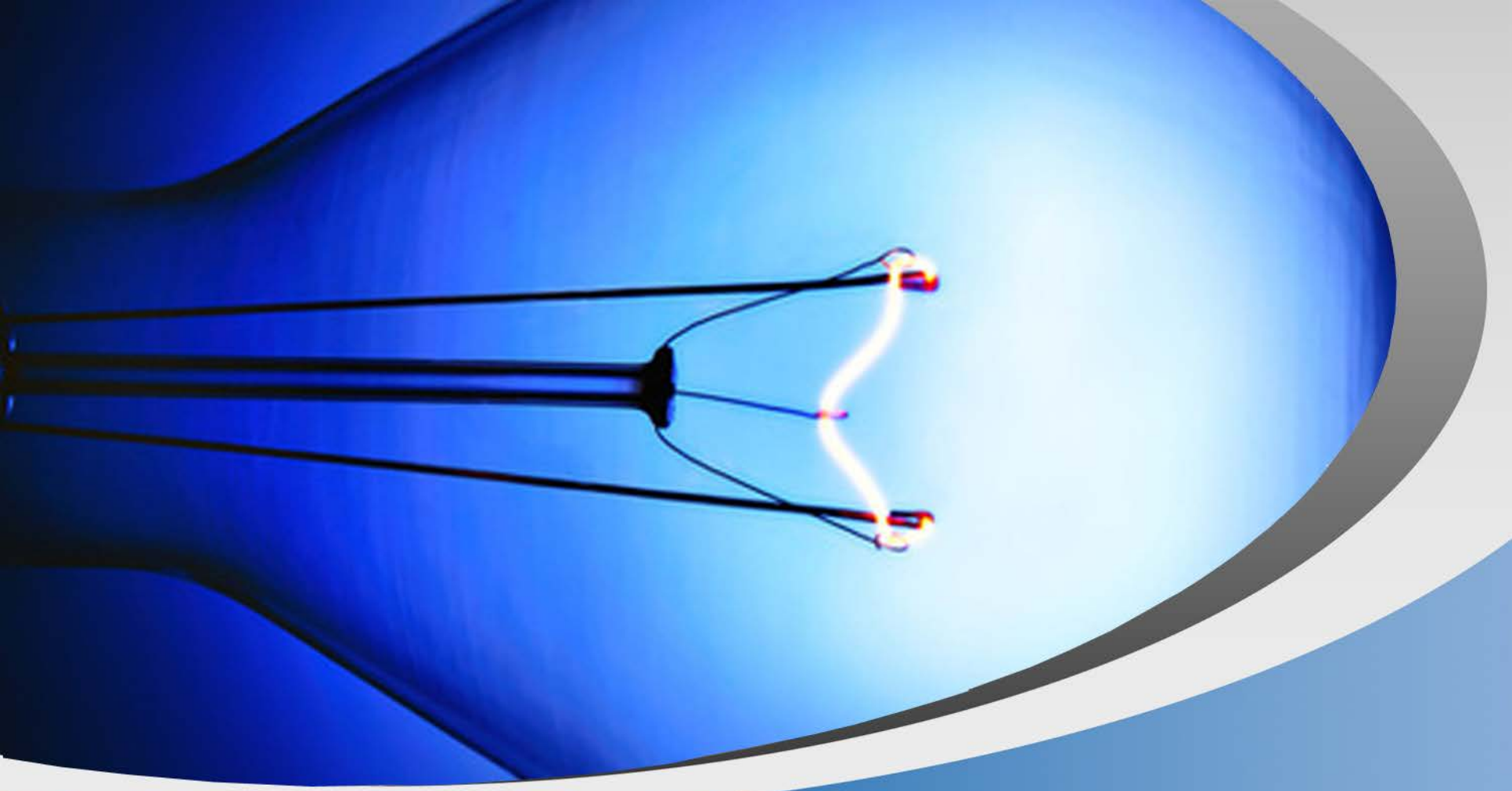


Power expression of the “current” power state would be the natural one to use for power computations

More refined power states would have more detailed power functions

For More Information On …

- **Successive Refinement of UPF Power Intent**
  - See paper/presentation/poster
    - **Successive Refinement: A Methodology for Incremental Specification of Power Intent**
      - by A. Khan, E. Quiggley, J. Biggs (ARM); E. Marschner (Mentor Graphics)
    - Session 8: Low Power Verification (Weds 10:00-11:30am; Oak)
- **Power State Definition and Refinement**
  - See paper/presentation
    - **Unleashing the Full Power of UPF Power States**
      - by E. Marschner (Mentor Graphics), J. Biggs (ARM)
    - Session 3: Design (Tues 9:00-10:30am; Monterey/Carmel)
- **Component Power Modeling**
  - Join the P1801 Working Group and the System Level Power (SLP) subgroup
    - Visit the web page at http://standards.ieee.org/develop/project/1801.html
    - Or send a request for information to admin@p1801.org



Thank you!

accellera
SYSTEMS INITIATIVE