

Managing Highly Configurable Design and Verification

Jeremy Ridgeway
Broadcom, Ltd.

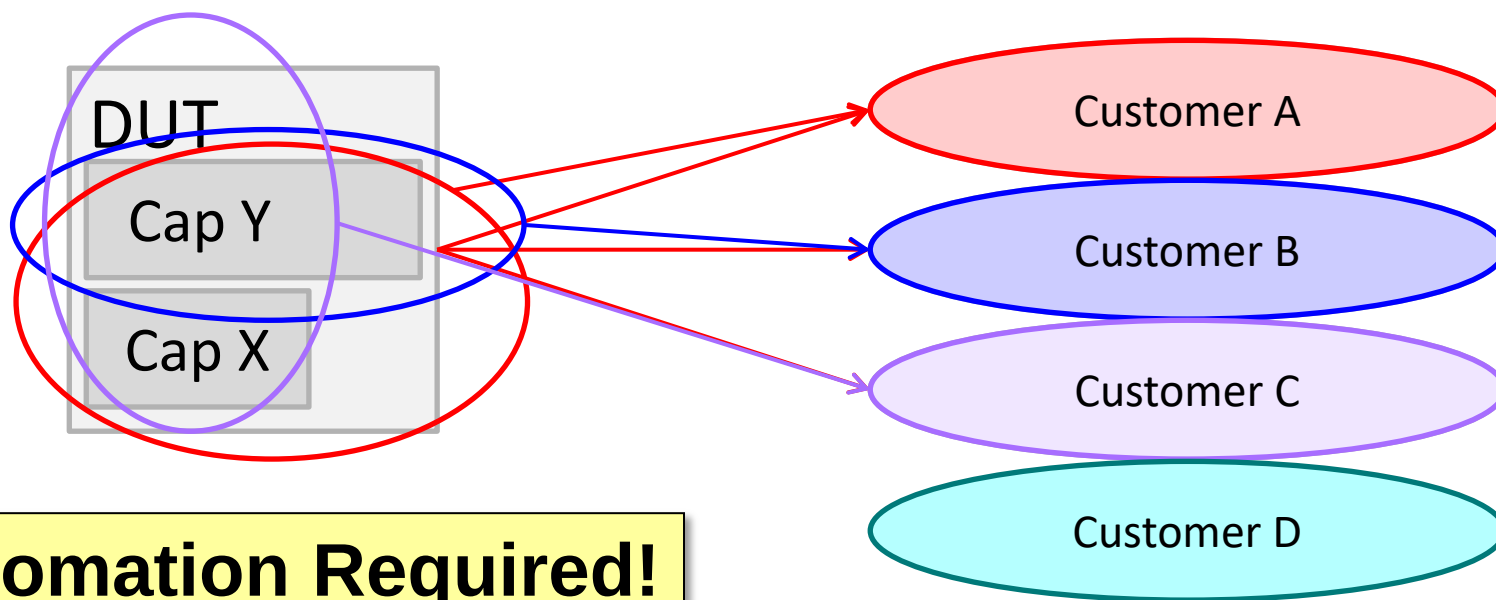


Agenda

- Root of the problem
- Automation required
 - DUT
 - Verification Planning
 - Test bench
 - Reporting Structures
- Conclusions

Hardware Configurations

- One RTL set to target multiple customers
- Configuration due to modify a common capability
- Configuration due to add/remove a specific capability

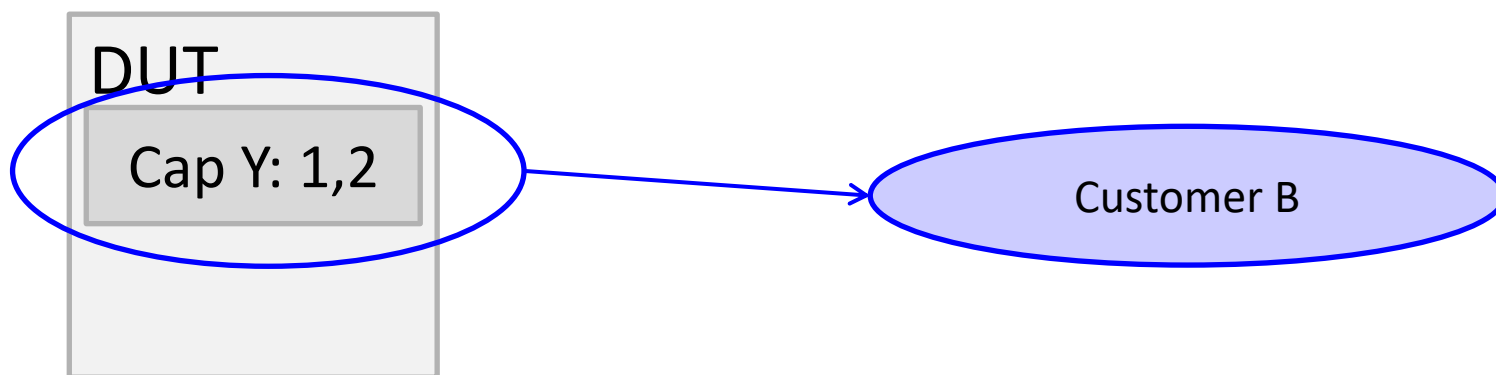


Automation Required!

Common Architectural Theme

- Super-set thing
- Customer-specific thing

Most important



Agenda

- Root of the problem
- Automation required
 - Single CONFIG structure
 - DUT
 - Verification Planning
 - Test bench
 - Reporting Structures
- Conclusions

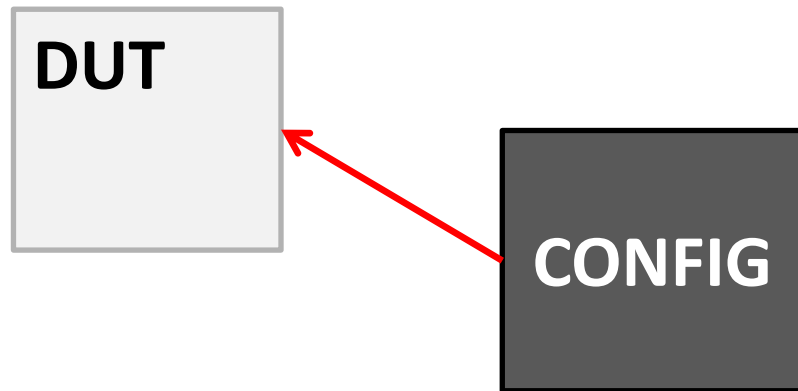
Configuration Matrix and Object

Configuration	Values	Customer Selection
CAP Y	1, 2	?
CAP X	Y / N	?
...		

Cust_A
CAP_Y => [1, 2]
CAP_x => 1
...

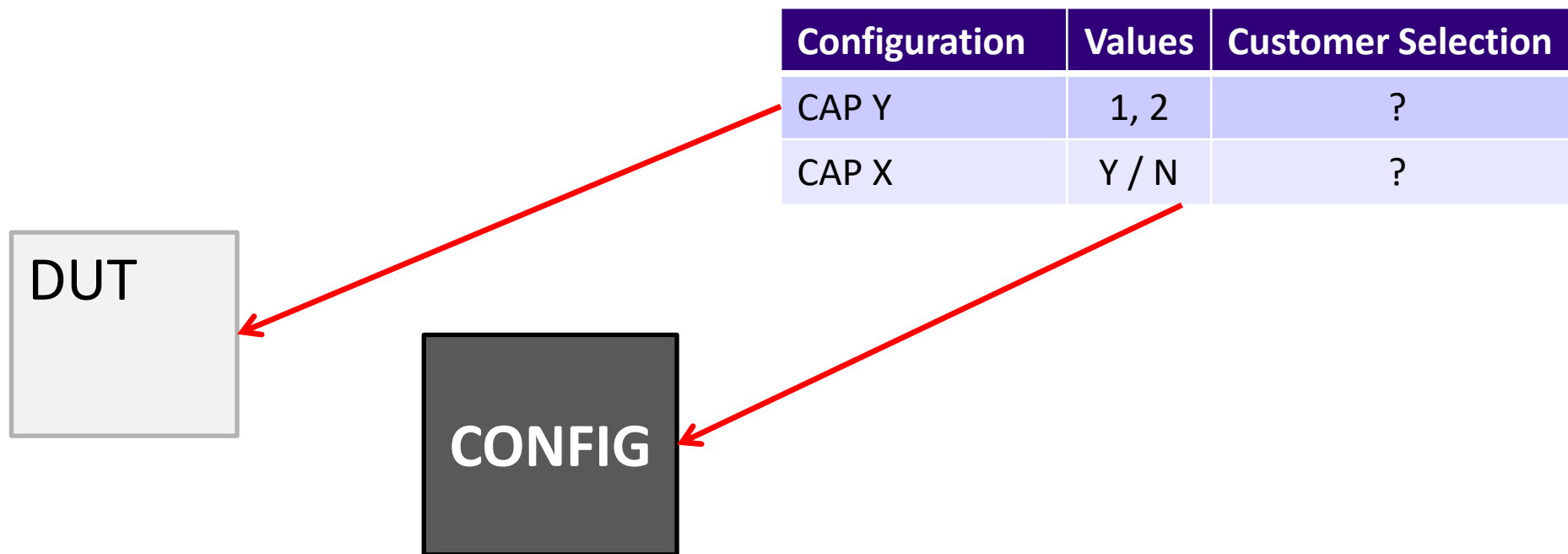
- Must match a configuration matrix agreed upon with the customer
 - We chose Perl5 (could be any OOP language)

Config: Center of Verification Universe



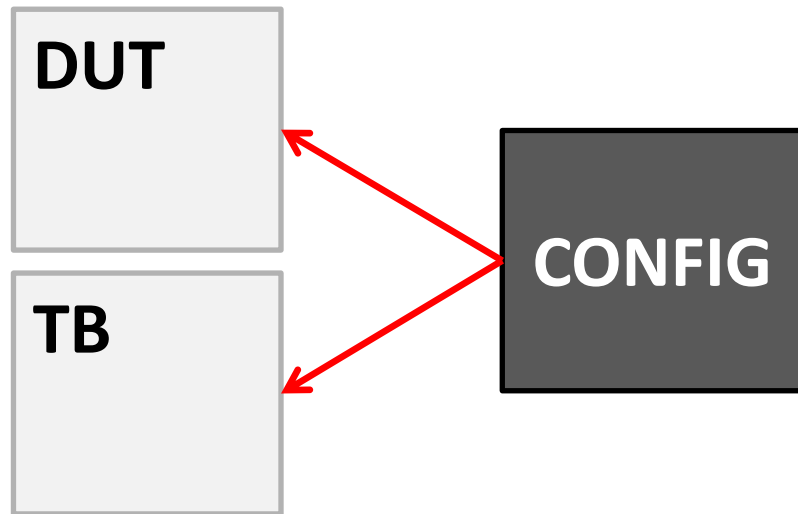
The CONFIG object drives the DUT transformation

Config: Center of Verification Universe

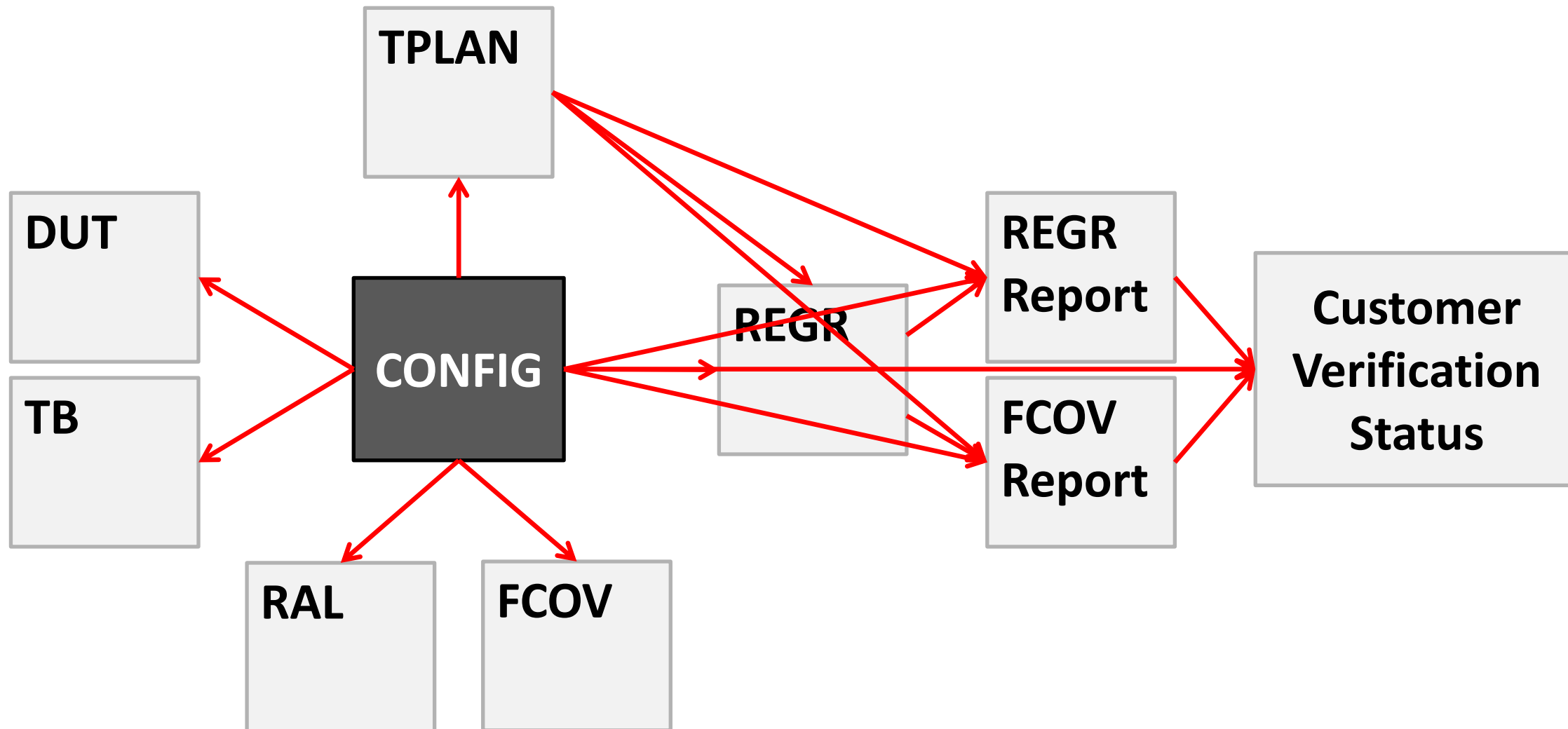


- The configuration matrix drives:
 - DUT Architecture & Implementation
 - CONFIG Structure & Value Selection

Config: Center of Verification Universe



Config: Center of Verification Universe



CONFIG Object Representation

CONFIG

CUST_C

Agenda

- Root of the problem
- Automation required
 - Single CONFIG structure
 - DUT
 - Verification Planning
 - Test bench
 - Reporting Structures
- Conclusions

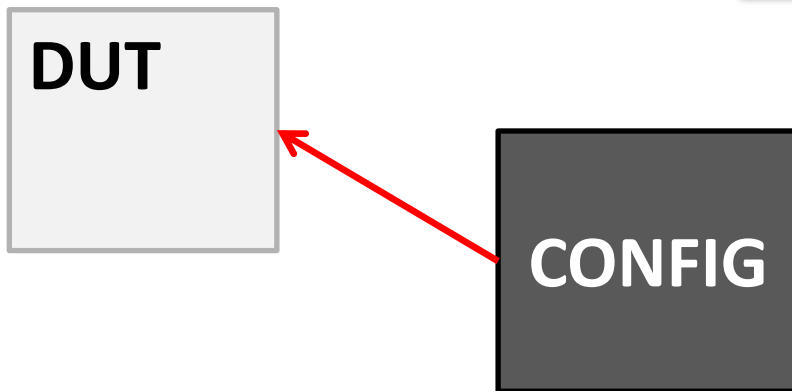
DUT Guidelines

- NO RTL should be simulated that is NOT delivered to customer

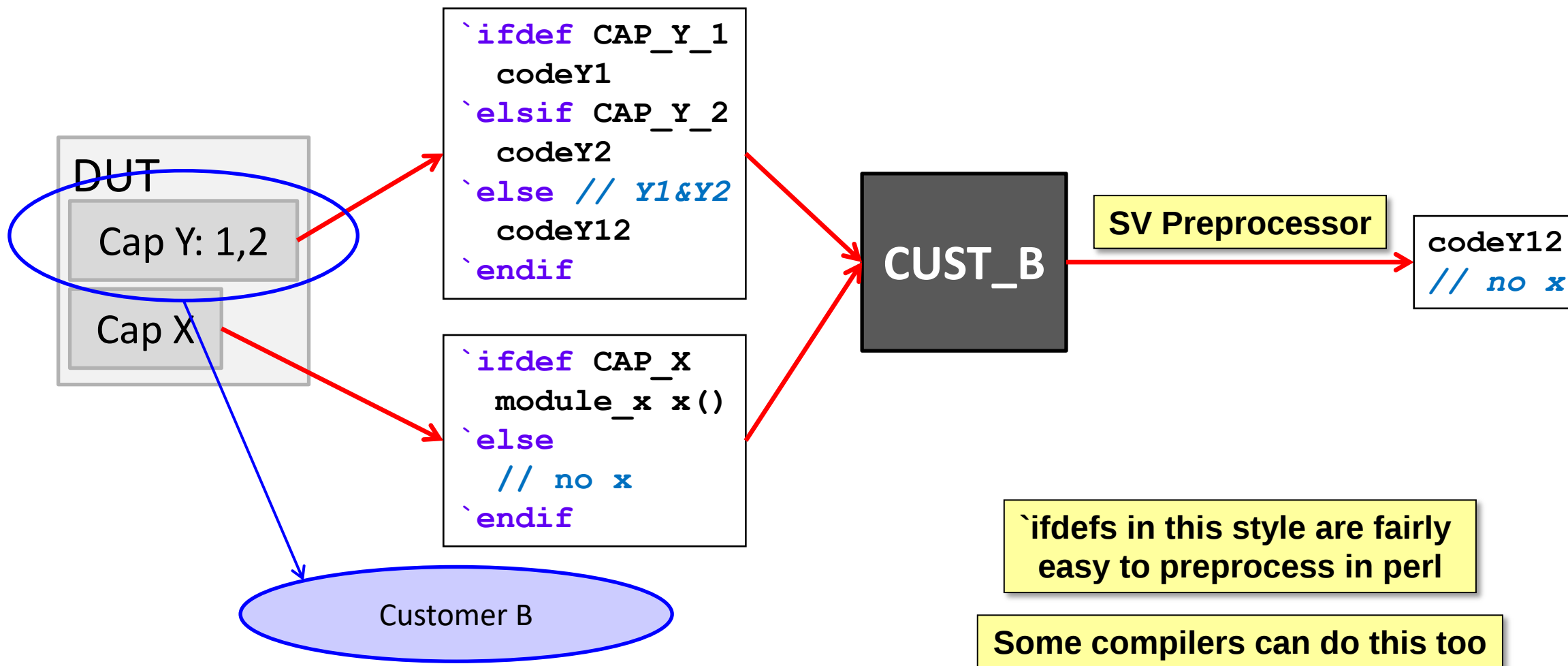
NO Superset Simulation / Verification

- Minimize the ability for the customer to make mistakes

Remove `ifdefs whenever possible



DUT Transformation



Agenda

- Root of the problem
- Automation required
 - Single CONFIG structure
 - DUT
 - Verification Planning
 - Test bench
 - Reporting Structures
- Conclusions

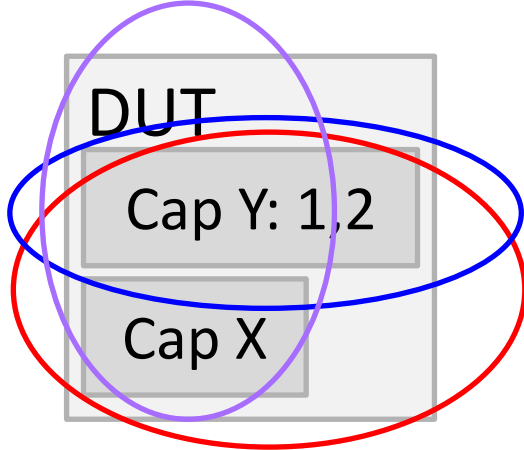
Testplan: A Live Document

Scenario ID	Feature	Subfeature	Generate	Expect
CAP_X.DATA.1	CAP X Data	Good	...stimulus...	...verification...
ERR.CAP_X.1	CAP X Data	Bad data	...	...
CAP_Y.MODE_1.1	CAP Y Mode 1	Select mode 1	...	...
CAP_Y.MODE_1.2	CAP Y Mode 1	Set FW mode 1	...	...
CAP_Y.MODE_2.1	CAP Y Mode 2	Select mode 2	...	...
ERR.CAP_Y.1	CAP Y Bad	Bad select	...	...

- ALL testing scenarios uniquely identified: the superset case

Testplan: A Live Document

Scenario ID	CUST_A	CUST_B	CUST_C
CAP_X.DATA.1	Y		Y
ERR.CAP_X.1	Y		Y
CAP_Y.MODE_1.1	Y	Y	Y
CAP_Y.MODE_1.2	Y	Y	Y
CAP_Y.MODE_2.1	Y	Y	
ERR.CAP_Y.1	Y	Y	Y

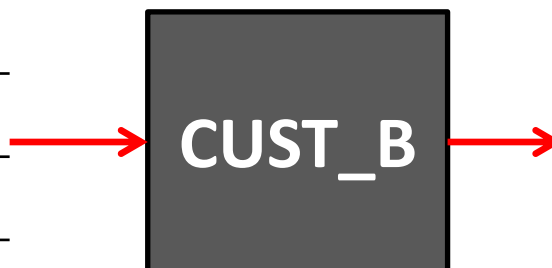


The diagram on the right illustrates the relationship between the testplan and the Device Under Test (DUT). It shows a large gray rectangle labeled 'DUT' containing two smaller gray rectangles: 'Cap Y: 1,2' and 'Cap X'. Three overlapping ovals represent customer-specific test scenarios: a red oval for CUST_A, a blue oval for CUST_B, and a purple oval for CUST_C. The red oval covers the 'CAP_X.DATA.1' and 'ERR.CAP_X.1' rows of the table. The blue oval covers the 'CAP_Y.MODE_1.1', 'CAP_Y.MODE_1.2', 'CAP_Y.MODE_2.1', and 'ERR.CAP_Y.1' rows. The purple oval covers the 'CAP_Y.MODE_1.1', 'CAP_Y.MODE_1.2', and 'ERR.CAP_Y.1' rows. The intersection of all three ovals (CUST_A, CUST_B, and CUST_C) corresponds to the 'CAP_Y.MODE_1.1' and 'CAP_Y.MODE_1.2' scenarios.

- Customer-specific scenarios are indicated in the testplan

Testplan: A Live Document

Scenario ID	CUST_A	CUST_B	CUST_C
CAP_X.DATA.1	Y		Y
ERR.CAP_X.1	Y		Y
CAP_Y.MODE_1.1	Y	Y	Y
CAP_Y.MODE_1.2	Y	Y	Y
CAP_Y.MODE_2.1	Y	Y	
ERR.CAP_Y.1	Y	Y	Y



Scenario ID	CUST_B
CAP_Y.MODE_1.1	Y
CAP_Y.MODE_1.2	Y
CAP_Y.MODE_2.1	Y
ERR.CAP_Y.1	Y

- Transform the testplan with customer CONFIG object
- Result indicates components required for customer-specific testbench

Agenda

- Root of the problem
- Automation required
 - Single CONFIG structure
 - DUT
 - Verification Planning
 - Test bench
 - Reporting Structures
- Conclusions

Testing Scenarios

- Directed tests
- Directed-random tests
- Constrained-random test bench

Directed Testing

- Scenario ID not tested when tests not run
- Scenario ID verified when tests run & pass
- Scenario ID fails when tests run & fail
- No functional coverage required
- Regression report sufficient for verification status
- No scenarios outside testplan expected

Scenario ID	Test	Regr
CAP_Y.MODE_1.1	Test1	← Y
CAP_Y.MODE_1.2	Test2	← Y
CAP_Y.MODE_2.1	Test3	←
ERR.CAP_Y.1	Test4	←

not run

**No Unknown
Scenario Coverage**

Directed-Random Testing

- Scenario ID not tested when tests not run
- Scenario ID verified when tests run & pass and functional coverage hit
- Scenario ID fails when tests run & fail
- Functional coverage required
- Regression report + functional coverage sufficient for verification status

Scenario ID	Test	Regr
CAP_Y.MODE_1.1	Test1	← Y
→ CAP_Y.MODE_1.2	Test1	Y
CAP_Y.MODE_2.1	Test1	
ERR.CAP_Y.1	Test2	←

**Some Unknown
Scenario Coverage**

Constrained-Random Testbench

- Scenario ID not tested when functional coverage not hit
- Scenario ID verified when functional coverage hit & tests pass
- Scenario ID fails when functional coverage hit & tests fail

Scenario ID	Coverage	Test	Regr
CAP_Y.MODE_1.1	group: a	Base	Y
CAP_Y.MODE_1.2	group: b	Base	Y
CAP_Y.MODE_2.1	group: c	Base	
ERR.CAP_Y.1	group: d	EnErrs	

**Unknown
Scenarios Likely**

- Functional coverage required and is focus
- Regression report + functional coverage required for verification status

Functional Coverage Model

- Developed independent of testbench
- Metrics per scenario indicated in the Testplan
- Hierarchical spreadsheet format employing configuration values

Scenario ID	Coverage	Test	Regr
CAP_Y.MODE_1.1	group: a	Base	Y
CAP_Y.MODE_1.2	group: b	Base	Y
CAP_Y.MODE_2.1	group: c	Base	
ERR.CAP_Y.1	group: d	EnErrs	

Functional Coverage XLSX

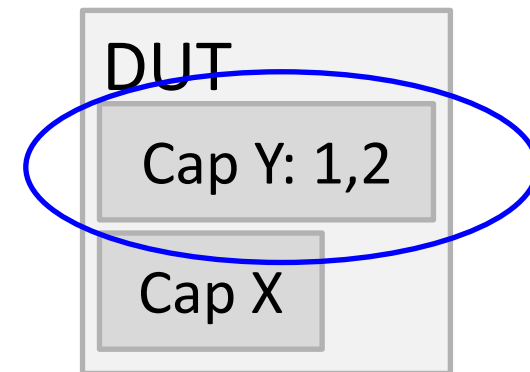
Scenario ID	Subfeature
CAP_X.DATA.1	Good
ERR.CAP_X.1	Bad data
CAP_Y.MODE_1.1	Select mode 1
CAP_Y.MODE_1.2	Set FW mode 1
CAP_Y.MODE_2.1	Select mode 2
ERR.CAP_Y.1	Bad select

Testplan Superset

Variable	Range	Description
C_CAP_X	0,1	Capability X support (>0)
C_CAP_Y	0,1,2	Capability Y support (>0) and range
M_CAP_Y	0,1,2	Capability runtime select
FW_SEL	0,1	Capability Y selected by firmware
SEL_INVALID	0,1	Capability Y selection failed

Coverpoint Superset

- Transform functional coverage model with CONFIG



Covergroup for CAP_Y.MODE_1.1

sel_mode_1 is valid for CUST_B

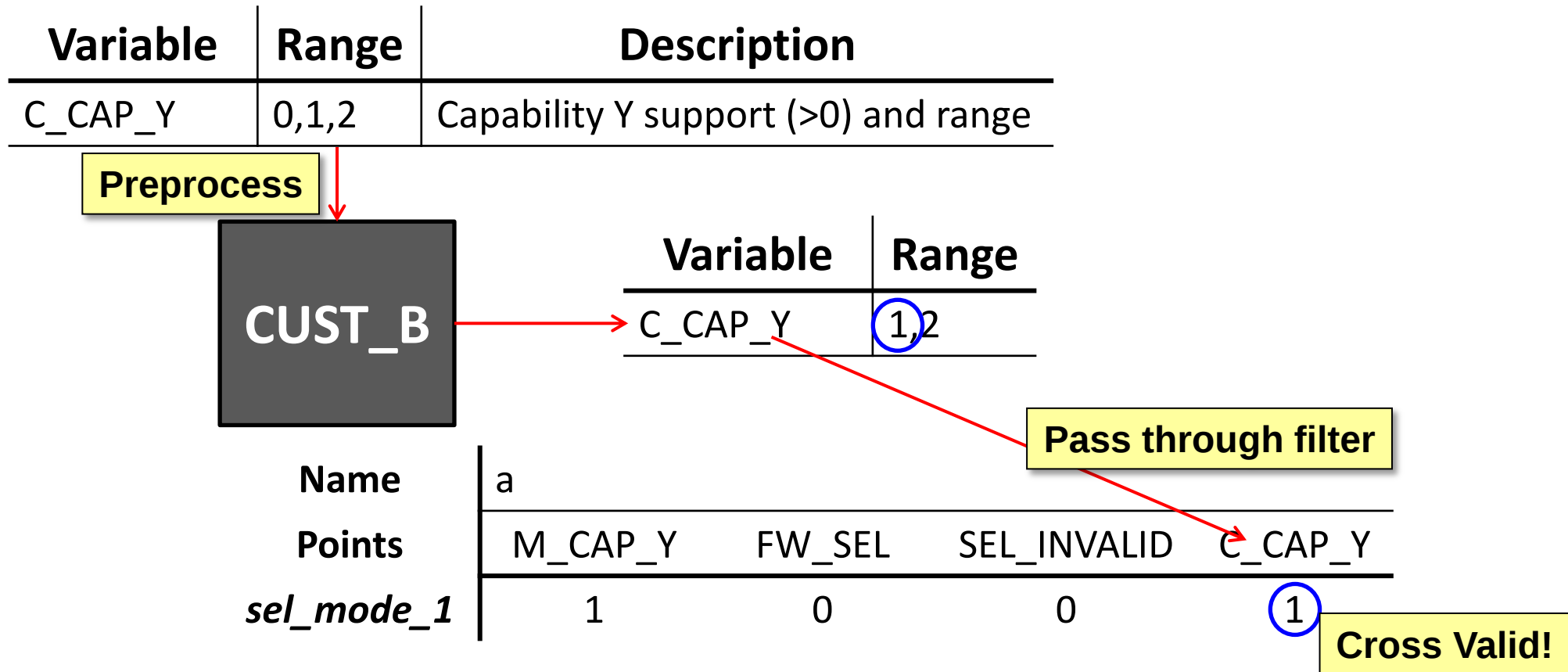
Name	a				
Points	M_CAP_Y	FW_SEL	SEL_INVALID	C_CAP_Y	
<i>sel_mode_1</i>	1	0	0	1	

covergroup a;

endgroup

Scenario ID	Subfeature	Coverage
CAP_Y.MODE_1.1	Sel mode 1	group: a
CAP_Y.MODE_1.2	FW mode 1	group: b
CAP_Y.MODE_2.1	Sel mode 2	group: c
ERR.CAP_Y.1	Bad sel	group: d

Configuration Filtering: Valid



- C_CAP_Y is defined in CUST_B CONFIG object as: CAP_Y: 1,2

Covergroup for CAP_Y.MODE_1.1

Name	a			
Points	M_CAP_Y	FW_SEL	SEL_INVALID	C_CAP_Y
<i>sel_mode_1</i>	1	0	0	1

covergroup a;

< >

endgroup

Scenario ID	Subfeature	Coverage
CAP_Y.MODE_1.1	Sel mode 1	group: a
CAP_Y.MODE_1.2	FW mode 1	group: b
CAP_Y.MODE_2.1	Sel mode 2	group: c
ERR.CAP_Y.1	Bad sel	group: d

Covergroup for CAP_Y.MODE_1.1

Name	a			
Points	M_CAP_Y	FW_SEL	SEL_INVALID	C_CAP_Y
<i>sel_mode_1</i>	1	0	0	1

```

covergroup a;
  coverpoint M_CAP_Y { bins B0 = { 1 };;}
  coverpoint FW_SELECT { bins B0 = { 0 };;}
  coverpoint SEL_INVALID { bins B0 = { 0 };;}

endgroup
  
```

Scenario ID	Subfeature	Coverage
CAP_Y.MODE_1.1	Sel mode 1	group: a
CAP_Y.MODE_1.2	FW mode 1	group: b
CAP_Y.MODE_1.3	Sel mode 2	group: c
ERR.CAP_Y		group: d

Cross sel_mode_1 actually covers the scenario

Configuration Filtering: Invalid

Variable	Range	Description
C_CAP_X	0,1	Capability X support (>0)

Preprocess

CUST_B

Variable	Range
C_CAP_0	0

Name	CapXData
Points	M_CAP_X ... C_CAP_X
<i>good_data</i>	1 ... 1

Cross Invalid!
Covergroup not applicable

- C_CAP_X is not part of CUST_B design, CUST_B CONFIG: CAP_X: 0

Covergroup for CAP_Y

Name	a			
Points	M_CAP_Y	FW_SEL	SEL_INVALID	C_CAP_Y
<i>sel_mode_1</i>	1	0	0	1
<i>fw_mode_1</i>	1	1	0	1
<i>sel_mode_2</i>	2	0	0	1
<i>sel_err</i>		*	1	1

Empty cells in a row indicate no dependency

Wildcard indicated all values are valid

```
sel_err0: FW_SEL==1 && SEL_INVALID==1
sel_err1: FW_SEL==0 && SEL_INVALID==1
```

Covergroup for CAP_Y

Name	a			
	M_CAP_Y	FW_SEL	SEL_INVALID	C_CAP_Y
<i>sel_mode_1</i>	1	0	0	1
<i>fw_mode_1</i>	1	1	0	1
<i>sel_mode_2</i>	2	0	0	1
<i>sel_err</i>		*	1	1

Scenario ID	Subfeature	Coverage	Test	Regr
CAP_Y.MODE_1.1	Sel mode 1	cross: tb::a.sel_mode_1	Base	Y
CAP_Y.MODE_1.2	Sel mode 1	cross: tb::a.sel_mode_1	Base	Y
CAP_Y.MODE_2.1	Sel mode 2	cross: tb::a.sel_mode_2	Base	Y
ERR.CAP_Y.1	Bad sel	cross: tb::a.sel_err	EnErrs	Y

Covered when both sel_err crosses are hit

Agenda

- Root of the problem
- Automation required
 - Single CONFIG structure
 - DUT
 - Verification Planning
 - Test bench
 - Reporting Structures
- Conclusions

Regression Report

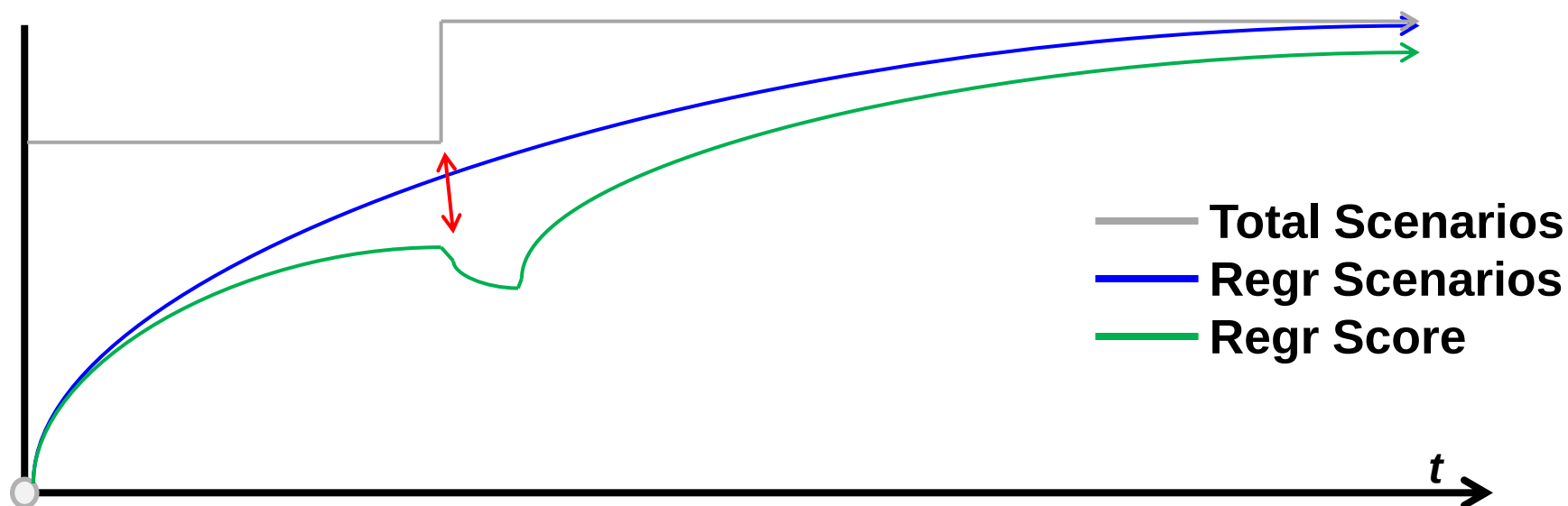
- In this regression, only Base test is run
 - Base test may be run 1000s of times
- Assume Base test passes 965 / 1000 times:

$$\text{Regression score} = (\text{pass_rate} * \text{covered_scenarios}) / \text{total_scenarios}$$

$$\text{Regression score} = (96.50\% * 3) / 4 = 72.375\%$$

Scenario ID	Subfeature	Coverage	Test	Regr
CAP_Y.MODE_1.1	Sel mode 1	cross: tb::a.sel_mode_1	Base	Y
CAP_Y.MODE_1.2	FW mode 1	cross: tb::a.fw_mode_1	Base	Y
CAP_Y.MODE_2.1	Sel mode 2	cross: tb::a.sel_mode_2	Base	Y
ERR.CAP_Y.1	Bad sel	cross: tb::a.sel_err	EnErrs	

Regression History



- Testplan is a live document:
 - Number of scenarios should (only) increase
 - Jumps will likely force the score down until regr scenarios catch up

Assuming 100% passing all time

Testplan Report

- Each scenario indicates pass/fail with simulation & coverage

Scenario ID/Metric		Instances	Rate	Result
CAP_Y_MODE_1.1				fail fcov_met
1	Base	1000	96.50%	fail
1	cross: tb::a.sel_mode_1		100%	fcov_met

Scenario ID	Subfeature	Coverage	Test	Regr
CAP_Y.MODE_1.1	Sel mode 1	cross: tb::a.sel_mode_1	Base	Y
CAP_Y.MODE_1.2	FW mode 1	cross: tb::a.fw_mode_1	Base	Y
CAP_Y.MODE_2.1	Sel mode 2	cross: tb::a.sel_mode_2	Base	Y
ERR.CAP_Y.1	Bad sel	cross: tb::a.sel_err	EnErrs	

Testplan Report

- Testplan score combines functional coverage with regression score
- Assume 96.50% pass rate and CAP_Y.MODE_1.2 not covered

Regression score = (pass_rate * covered_scenarios) / total_scenarios)

Testplan score = Regression score * functional_coverage_score

Testplan score = ((96.50% * 3) / 4) * (2 / 3) = 48.25%

Scenario ID	Subfeature	Coverage	Test	Regr
CAP_Y.MODE_1.1	Sel mode 1	cross: tb::a.sel_mode_1	Base	Y
CAP_Y.MODE_1.2	FW mode 1	cross: tb::a.fw_mode_1	Base	Y
CAP_Y.MODE_2.1	Sel mode 2	cross: tb::a.sel_mode_2	Base	Y
ERR.CAP_Y.1	Bad sel	cross: tb::a.sel_err	EnErrs	

Testplan Score

- Only includes functional coverage metrics *indicated* in the testplan
 - Functional coverage collected outside of testplan is ignored
 - Can include VIP or other external functional coverage
- Only includes run tests *indicated* in the testplan
 - Simulated tests run but not listed in the testplan is ignored
- Is this the best way to calculate testplan score?
 - I don't really know – let's discuss

Conclusions

- Holistic approach requires buy-in from whole team and management
- Our method can accurately pinpoint scenarios:
 - Verified
 - Failing
 - Missing
- Our method can manage multiple active programs simultaneously
- We have employed this approach on two major projects
 - >5 simultaneous customer programs

Agenda

- Root of the problem
- Automation required
 - Single CONFIG structure
 - DUT
 - Verification Planning
 - Test bench
 - Reporting Structures
- Conclusions

THANK YOU!

Questions Please

Extra Slides

Testbench Structure

- Common components / objects
- Extend to customer-specific
- Standard directory Structure
 - Common
 - Extended to customer-specific
- Class Inheritance Tree
 - Common
 - Extended to customer-specific
- Target the general case – common should be most heavily weighted

