vitesco TECHNOLOGIES

DATA CONSISTENCY TEST

TOWARDS SYSTEMATIC REQUIREMENTS ELICITATION IN AUTOMOTIVE MULTI-CORE APPLICATIONS

ERTS 2020 Toulouse, 30.01.2020

Ralph Mader Vitesco Technologies GmbH
Wolfgang Pree University of Salzburg and Chrona.com

Public

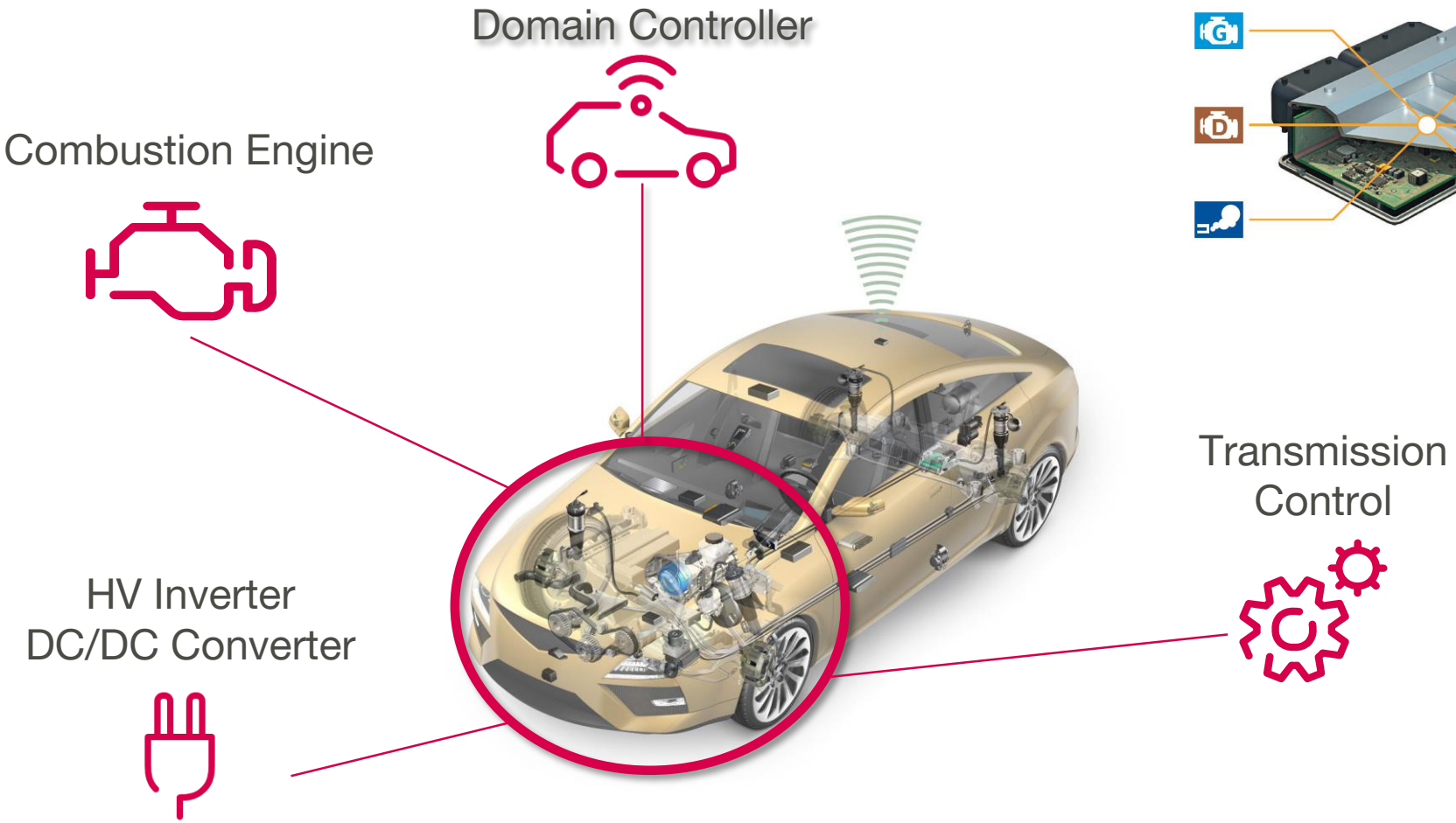
+chrona.com

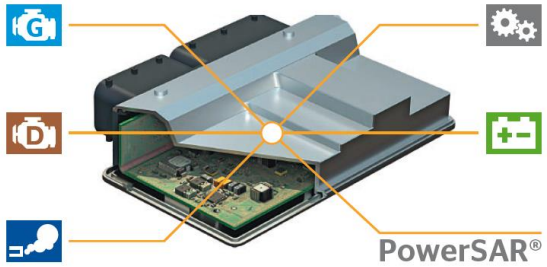
DATA CONSISTENCY TEST – TOWARDS SYSTEMATIC REQUIREMENTS ELICITATION

vitesco TECHNOLOGIES

MULTI-CORE SOFTWARE FOR POWERTRAIN

WHERE IS MULTI-CORE IN USE?

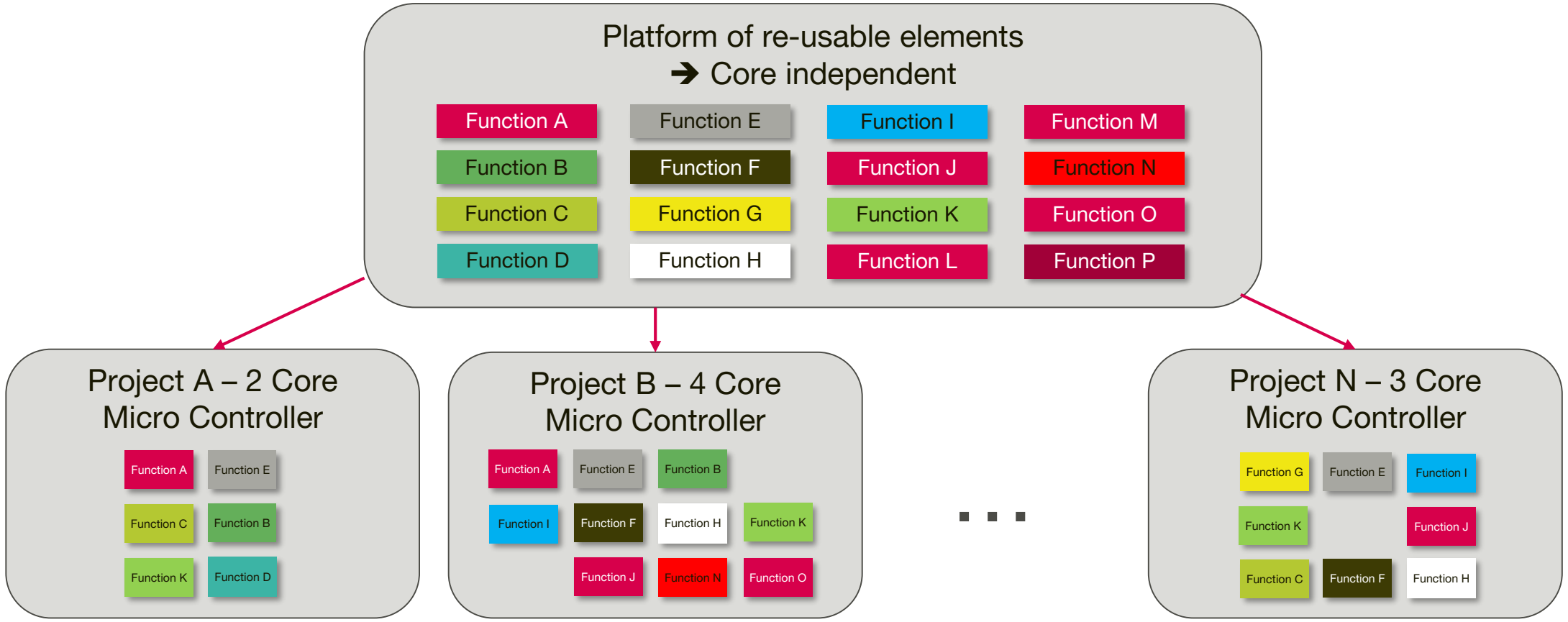




vitesco TECHNOLOGIES

MULTI-CORE SOFTWARE FOR POWERTRAIN

PROJECT VERSUS PLATFORM DEVELOPMENT

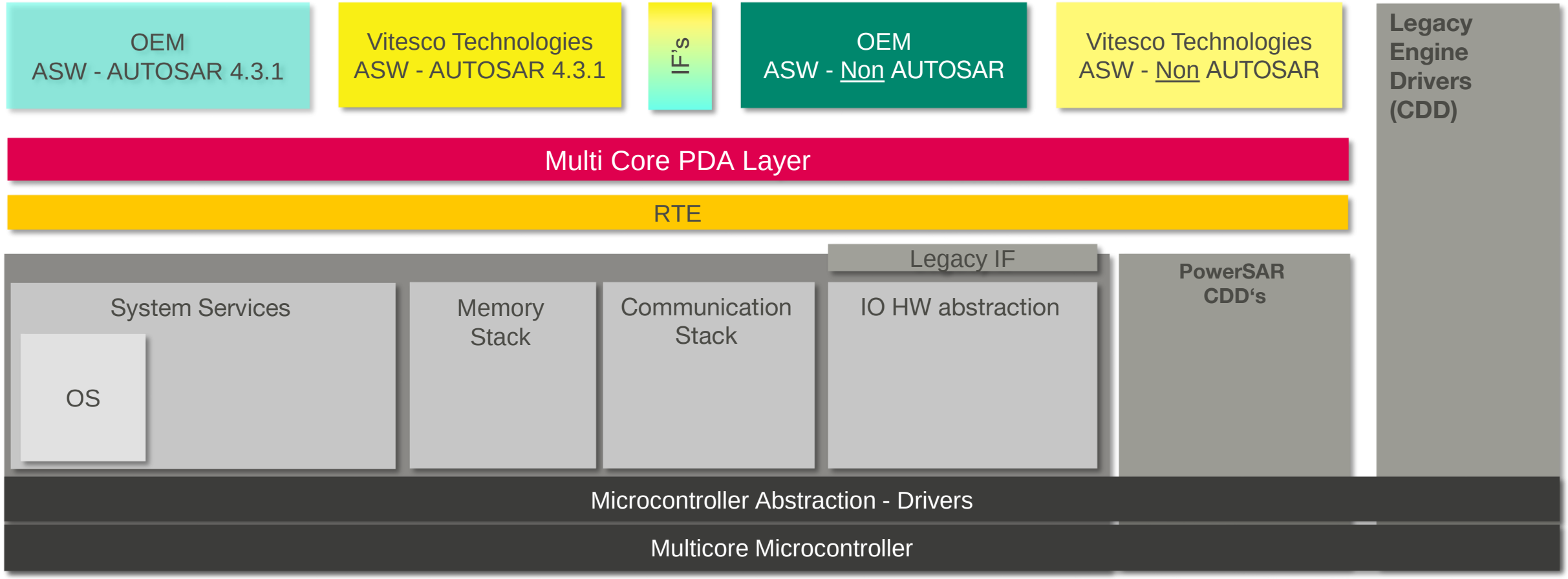


> Platform solution must be independent from core partitioning in project

vitesco TECHNOLOGIES

MULTI-CORE SOFTWARE FOR POWERTRAIN

TYPES OF SOFTWARE USED



> Multi Core Layer is generated based on data protection needs in project

vitesco TECHNOLOGIES

DATA CONSISTENC TEST – TOWARDS SYSTEMATIC REQUIREMENTS ELICITATION

vitesco TECHNOLOGIES

WHAT IS DATA CONSISTENCY?

DATA CONSISTENCY = DATA STABILITY & DATA COHERENCY

Stability

Coherency

> For proper functional behavior, both stability and coherency have to be ensured

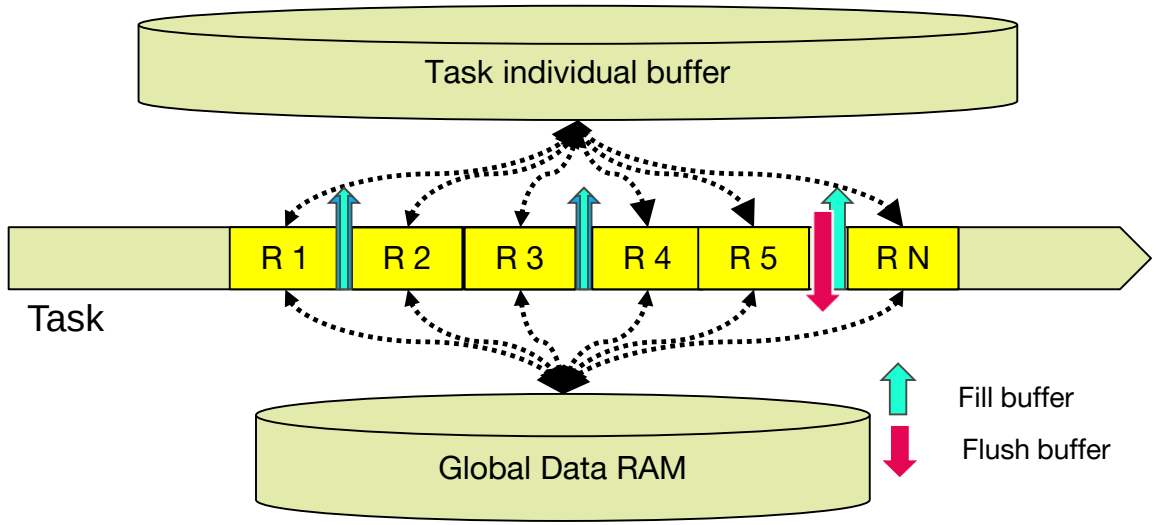
vitesco TECHNOLOGIES

MEANS TO ENSURE DATA CONSISTENCY

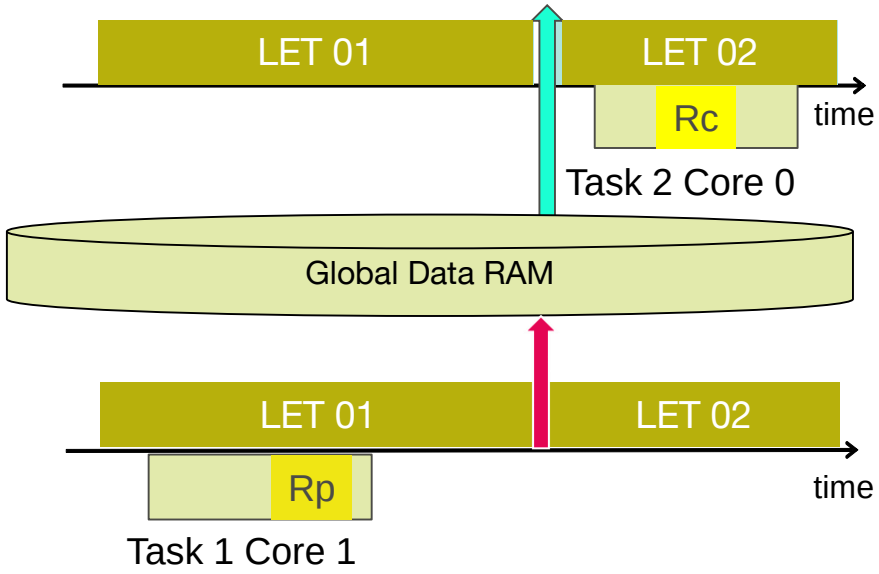
BUFFERING OR LOGICAL EXECUTION TIME (LET)

> Below you find two means how to ensure data consistency in Multi-Core Systems

Buffering of Data



Logical Execution Time (LET)



> Ensuring data consistency generates overhead

vitesco TECHNOLOGIES

DATA CONSISTENCY WITH MINIMAL OVERHEAD

SHORTCOMINGS OF REQUIREMENTS ELICITATION

Status Quo

> Functions are designed mostly by mechanical engineers

> Design object reviews are used today for identifying consistency requirements

> Quality of requirements is based on the multi-core background of the reviewers

Consequences

> Missing Requirements
could generate sporadic functional issues (sleeping issues)

> Non-maintained Requirements
could lead to miss data protection

> Useless Requirements
consume resources and add validation & maintenance effort

> Ensuring data consistency should minimize the overhead

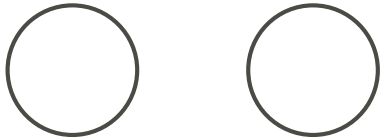
vitesco TECHNOLOGIES

DATA CONSISTENCY TESTING – TOWARDS SYSTEMATIC REQUIREMENTS ELICITATION

SAMPLE STABILITY VIOLATION

TIMING IS EVERYTHING …

milliseconds



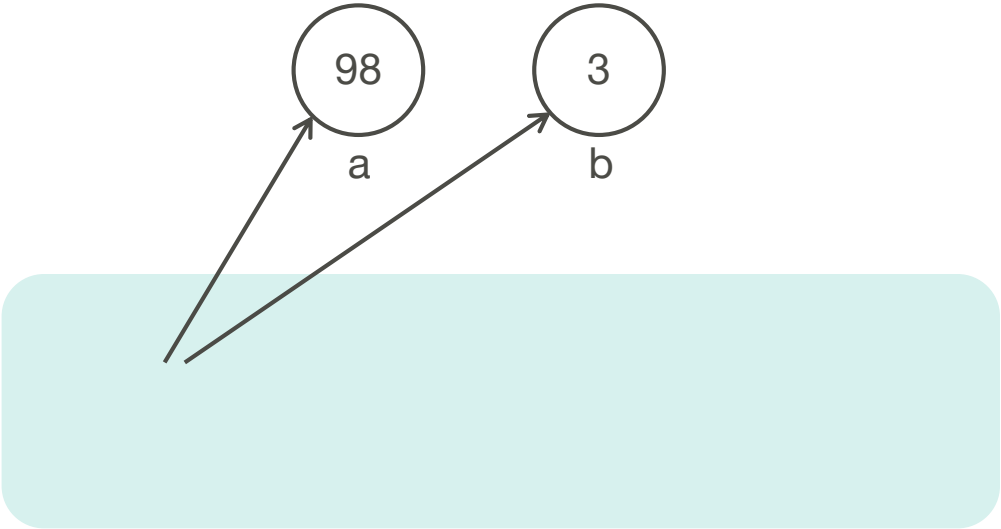
a b

global variables a, b

SAMPLE STABILITY VIOLATION

TIMING IS EVERYTHING …

milliseconds



global variables a, b

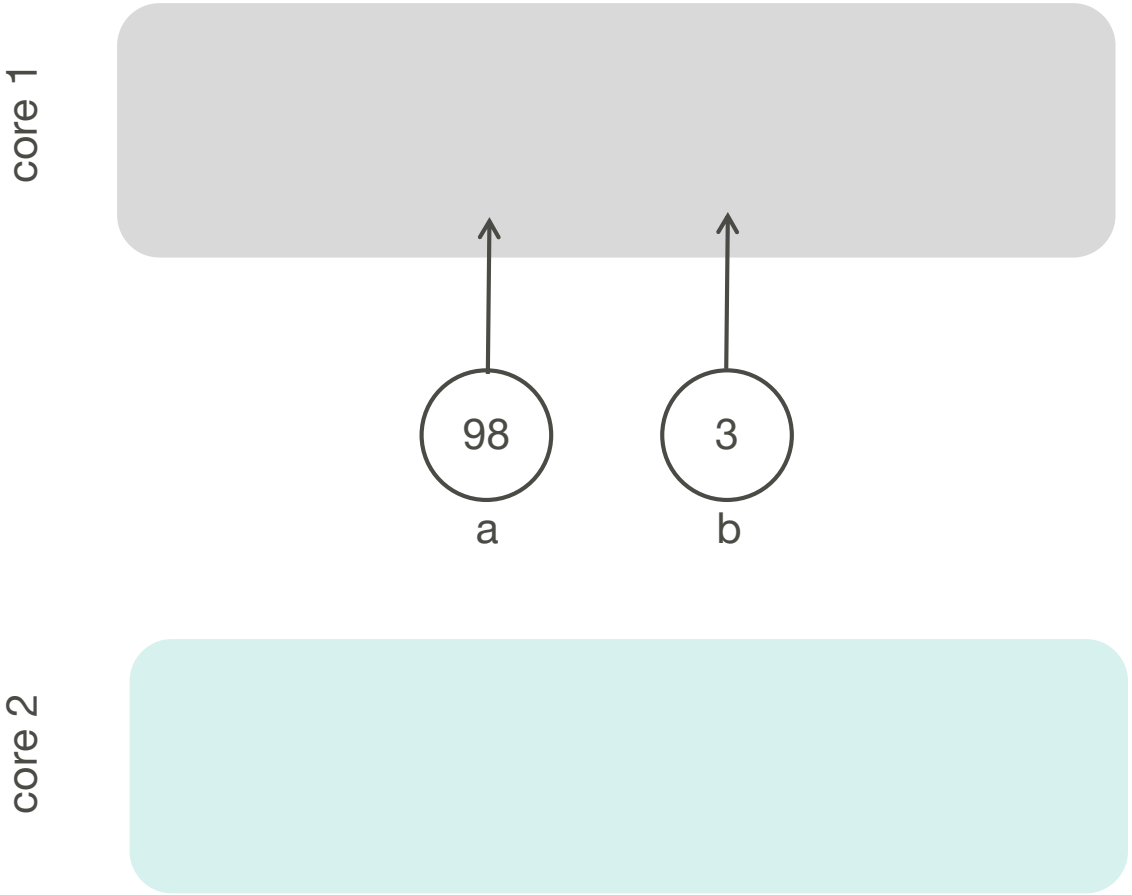
Provider component (PRV) writes values to a, b

SAMPLE STABILITY VIOLATION

TIMING IS EVERYTHING …

milliseconds

Module under Test (MUT) reads variables a, b as inputs



Provider component (PRV) writes values to a, b

```
void f(void) {
    ...
    if (a > b) {
        ...
        ...
        // use a, b for calc.
        ...
    }
}
```

if (98 > 3)

global variables a, b

SAMPLE STABILITY VIOLATION

TIMING IS EVERYTHING ...

milliseconds

Module under Test (MUT) reads variables a, b as inputs

core 1

98 a

3 b

core 2

Provider component (PRV) writes values to a, b

```
void f(void) {
    ...
    if (a > b) {
        ...
        ...
        // use a, b for calc.
        ...
    }
}
```

if (98 > 3)

global variables a, b

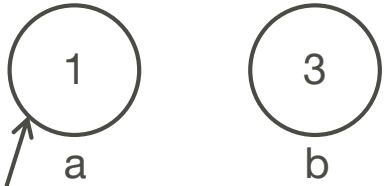
SAMPLE STABILITY VIOLATION

TIMING IS EVERYTHING …

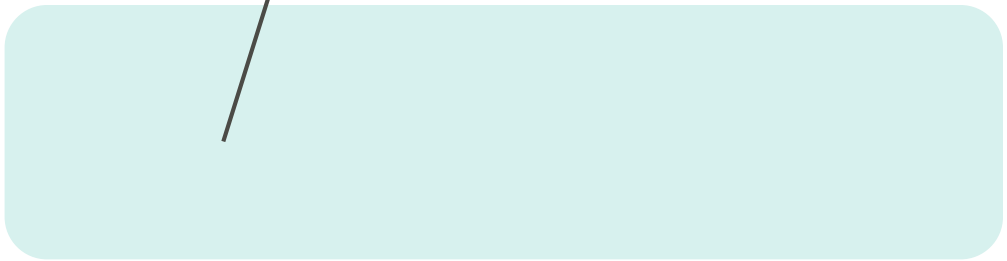
milliseconds

Module under Test (MUT) reads variables a, b as inputs

core 1



core 2



Provider component (PRV) writes values to a, b

```
void f(void) {
    ...
    if (a > b) {
        ...
        ...
        // use a, b for calc.
        ...
    }
}
```

if (98 > 3)

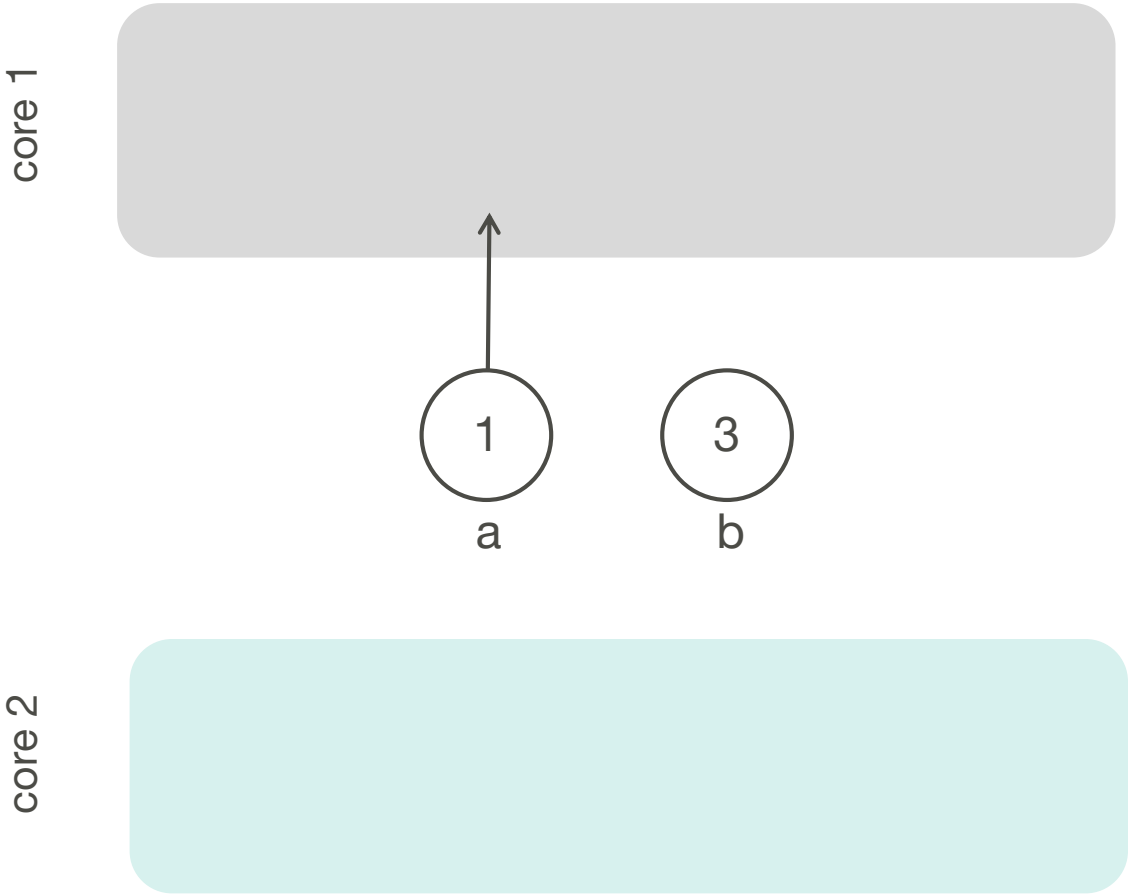
global variables a, b

SAMPLE STABILITY VIOLATION

TIMING IS EVERYTHING …

milliseconds

Module under Test (MUT) reads variables a, b as inputs



Provider component (PRV) writes values to a, b

```
void f(void) {
    ...
    if (a > b) {
        ...
        ...
        // use a, b for calc.
        ...
    }
}
```

if (98 > 3)

a= 1, b= 3

global variables a, b

SAMPLE STABILITY VIOLATION

TIMING IS EVERYTHING …

All would have been fine if:
> MUT would have executed a bit faster (eg, shorter waiting time for bus communication resource), or
> PRV would have executed a bit slower (eg, longer interrupt by another task function on core 2)

```
void f(void) {
    ...
    if (a > b) {
        ...
        ...
        // use a, b for calc.
        ...
    }
}
```

(98 > 3)

a= 98, b= 3

```
void f(void) {
    ...
    if (a > b) {
        ...
        ...
        // use a, b for calc.
        ...
    }
}
```

if (98 > 3)

a= 1, b= 3

we call this a
Problematic Access Pattern (PAP)

CORE CONCEPT: ADVERSARIAL TESTING

BY VARYING THE EXECUTION TIMES OF TASK FUNCTIONS WITHIN WCET LIMITS

- maximize occurrences of violations by manipulating execution times of code fragments to achieve ”bad” interleaving of MUT and PRV executions
  - PAP coverage (as many different PAPs as possible)
  - filter by assessing the effect of certain PAPs on the outputs

- basis for consistency testing: Validator simulator: a platform-aware Software-in-the-Loop (SiL) simulation
  - execution of application software is interleaved with simulation of a virtual platform model

RESULTS OF CONSISTENCY TESTING

ADEQUATE SET OF VARIABLES THAT NEED TO BE BUFFERED

>a (typically a reduced) set of data protection requirements

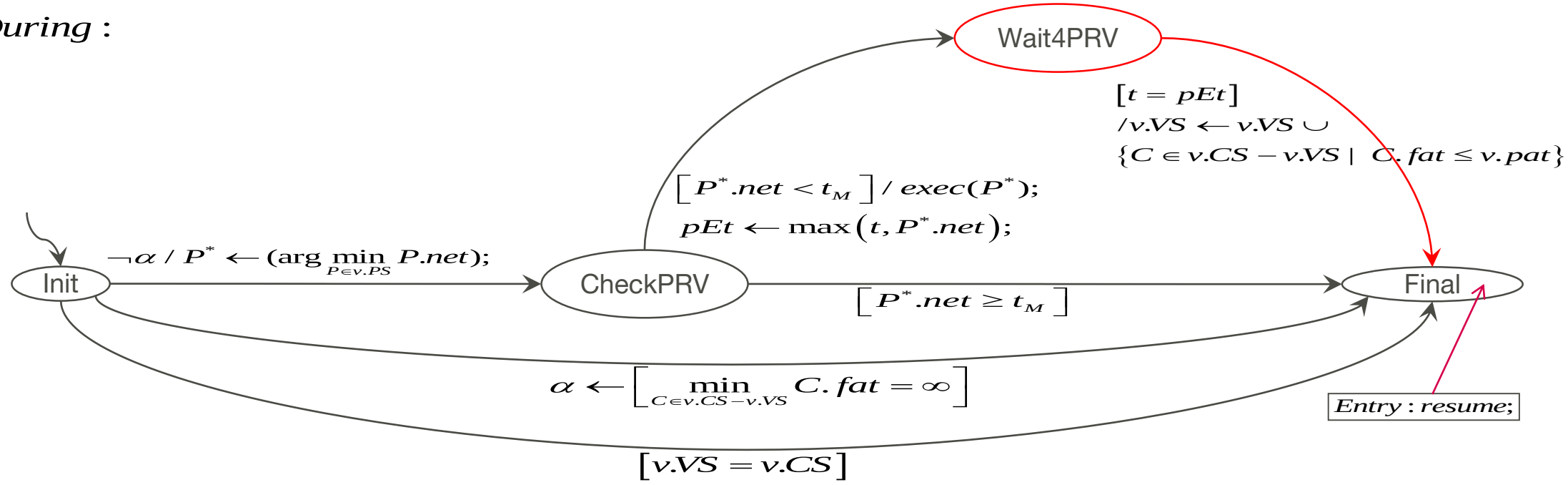
>documented exceptions with reproducible tests

SOLID FORMAL BASIS

FINITE STATE MACHINES

$$W.Entry : v.VS \leftarrow v.VS \cup \{C \in v.CS - v.VS \mid C.fat \leq v.pat\}$$

$W.During$:



$$W.Exit : \forall C \in v.CS,\ C.fat \leftarrow \min(C.fat, t);$$

> THUS, CONSISTENCY/COHERENCY TESTING CAN BE FORMALLY VERIFIED

TOOL USAGE

IMPROVES SOFTWARE QUALITY AND REDUCES RESOURCE CONSUMPTION

> batch mode as part of a daily build (continuous integration)
> interactively with UI seamlessly integrated in Matlab/Simulink and Eclipse

ConsistestReport.ctxml

Module	Set	Var	E: 0	E: 1	E: 2	E: 3	E: 4	E: 5	E: 6	E: 7
fmsp_ispclbas0__systrig_sege1										
	fmsp_ispclbas0_ispcl_bas__10ms									
		fac_afu_afs								
		fac_afu_ratio								
		inh_iv_cyl_deac								
	fmsp_ispclbas0_ispcl_bas__5ms									
		m_fg_inv_clp_cyl_inj								
	fmsp_ispclbas0_ispcl_bas__seg									
		lf_mfl_inj_upd								
fmsp_ispclmplinj__systrig_sege1										
	fmsp_isp_mpl_inj__100ms									
		tco_1_sys								
		tco_st_cur								
	fmsp_isp_mpl_inj__10ms									
		lv_st_end								
		state_var_mkt								
		t_ast								
	fmsp_isp_mpl_inj__5ms									
		m_air_cyl_pred								

DATA CONSISTENC TEST - TOWARDS SYSTEMATIC REQUIREMENT ELICITATION

1 MULTI CORE SOFTWARE FOR POWERTRAIN

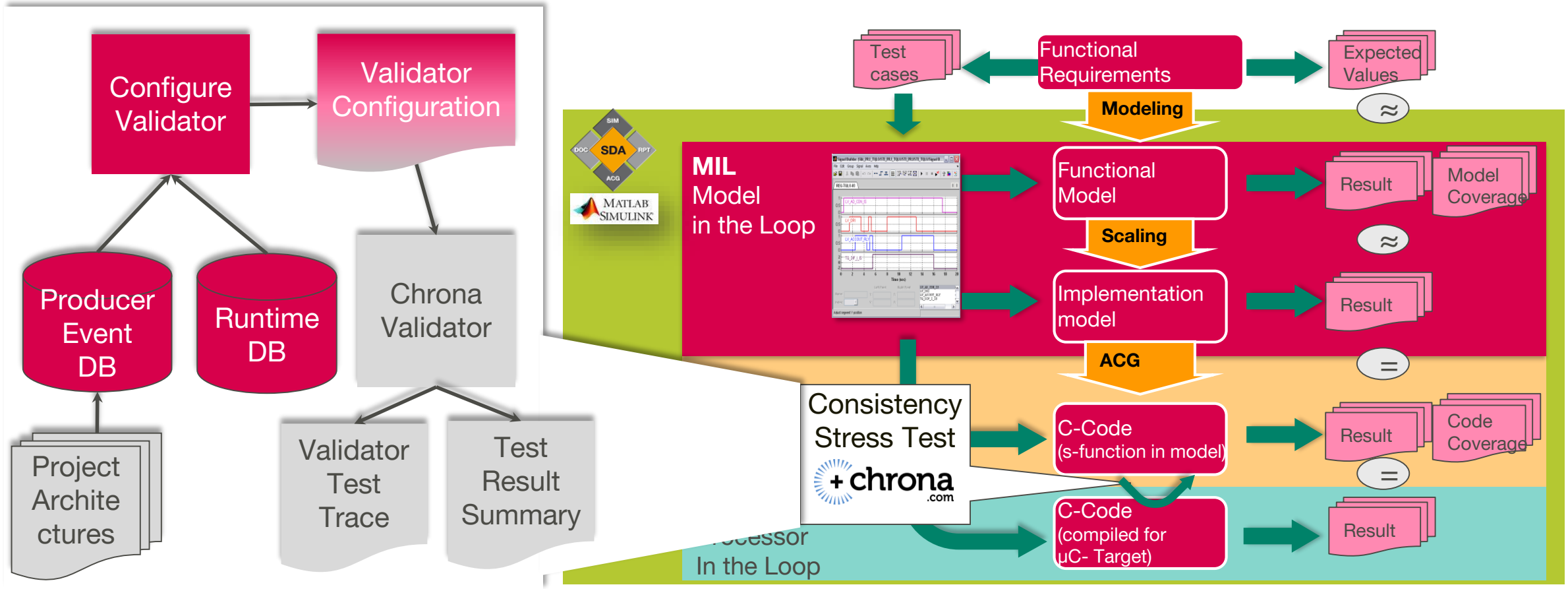
2 WHAT IS DATA CONSISTENCY

3 IDENTIFICATION OF CONSISTENCY REQUIREMENTS

4 SUMMARY AND OUTLOOK

SUMMARY AND OUTLOOK

WHEN TO PERFORM THE CONSISTENCY TEST?



Consistency stress test will complement the SIL test as a formal way to prove data consistency

vitesco TECHNOLOGIES

SUMMARY AND OUTLOOK

> Test is based on a formal method to identify consistency requirements

> It works in context of a project

> Extension to platform approach is possible by batch processing of different scenarios

> Piloting Phase within Vitesco Technologies is started

vitesco
TECHNOLOGIES

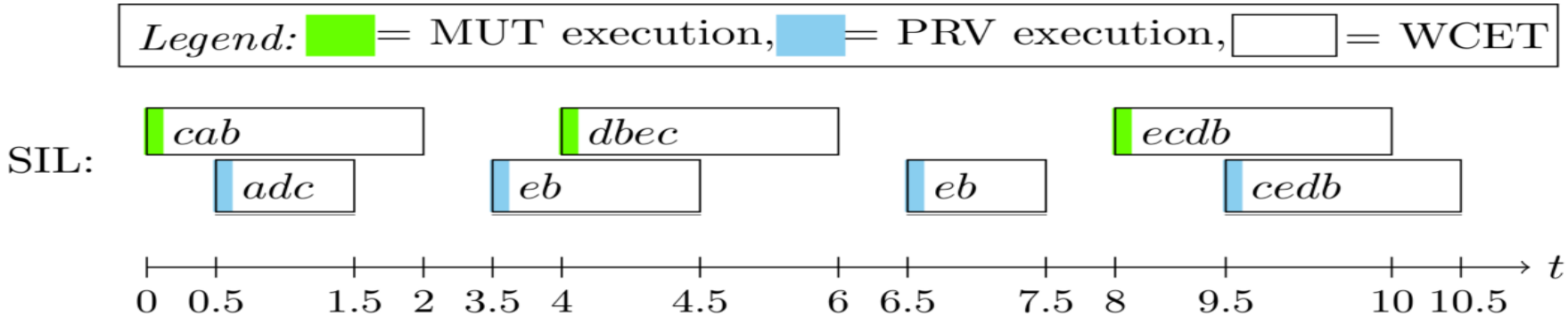
QUESTIONS?

ANNEX

EXAMPLE AND CASE STUDY ENGINE CONTROL FUNCTION – FOR SELF STUDY

EXAMPLE

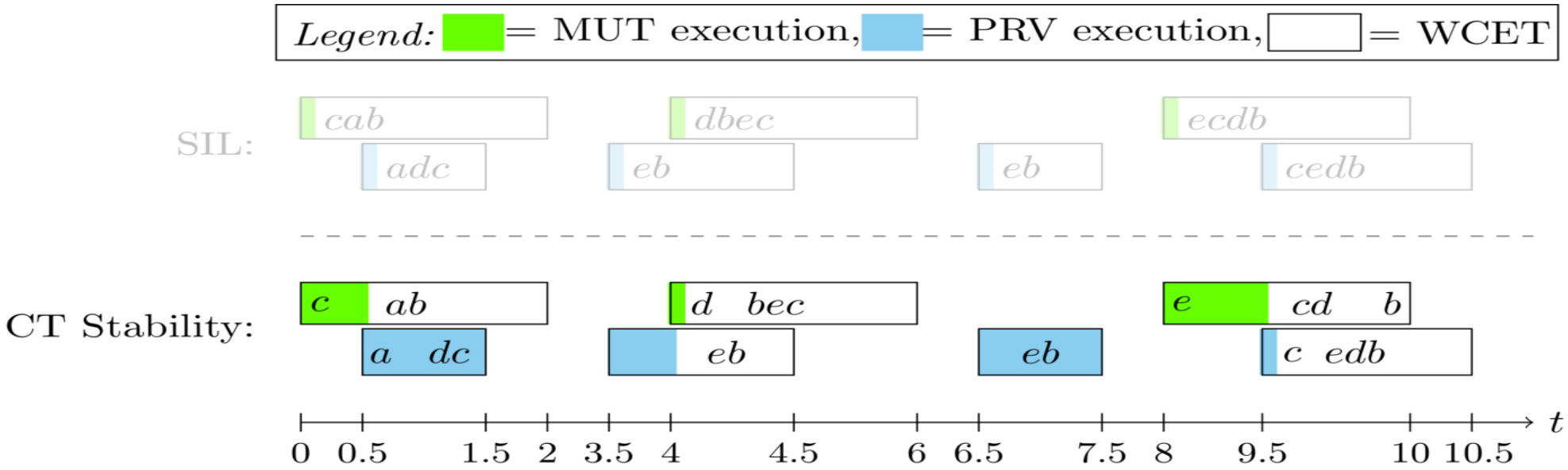
> Consistency sets: C_0={a,c,e}, C_1={b,d}



vitesco TECHNOLOGIES

EXAMPLE

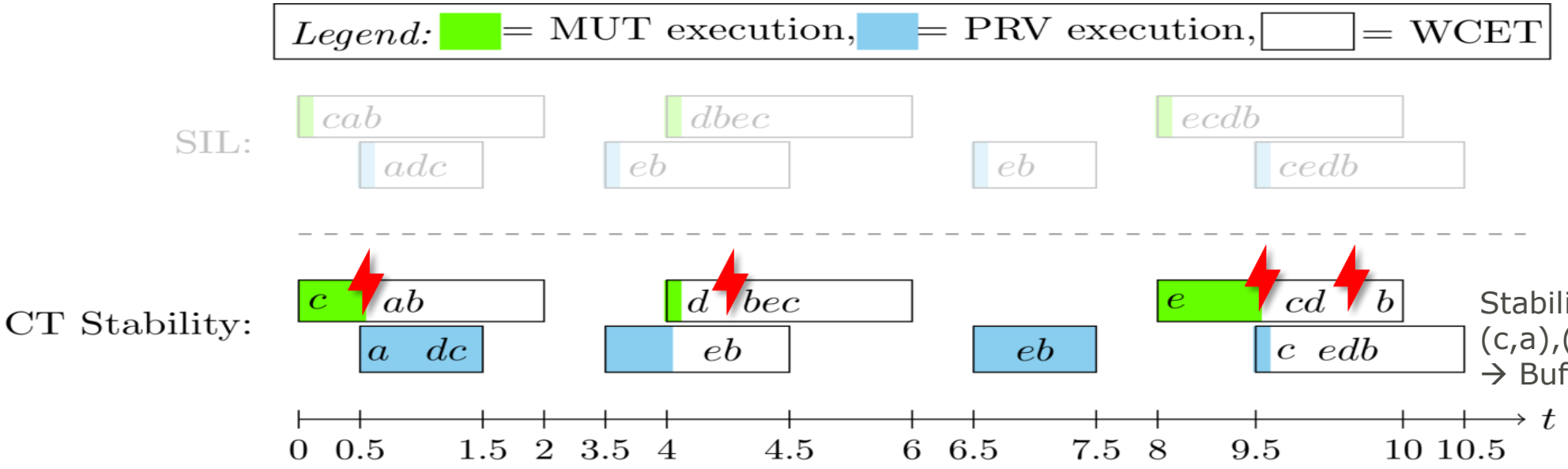
> Consistency sets: C_0={a,c,e}, C_1={b,d}



vitesco TECHNOLOGIES

EXAMPLE

> Consistency sets: C_0={a,c,e}, C_1={b,d}

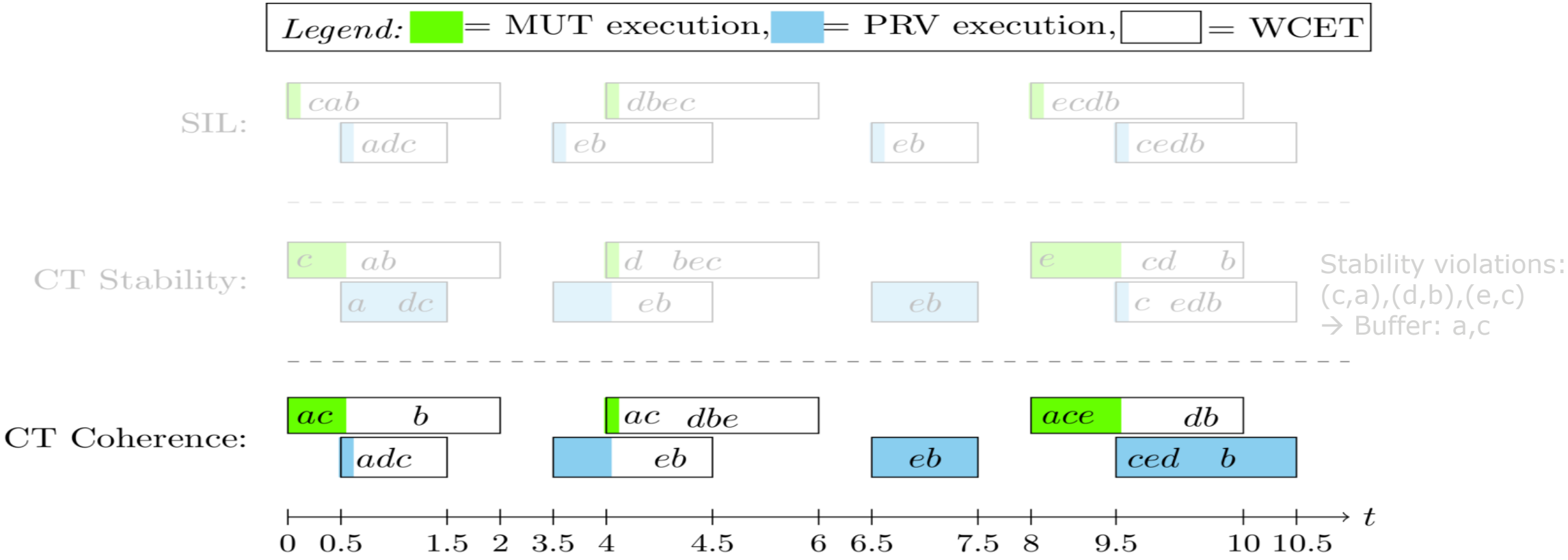


Stability violations:
(c,a),(d,b),(e,c)
→ Buffer: a,c

vitesco TECHNOLOGIES

EXAMPLE

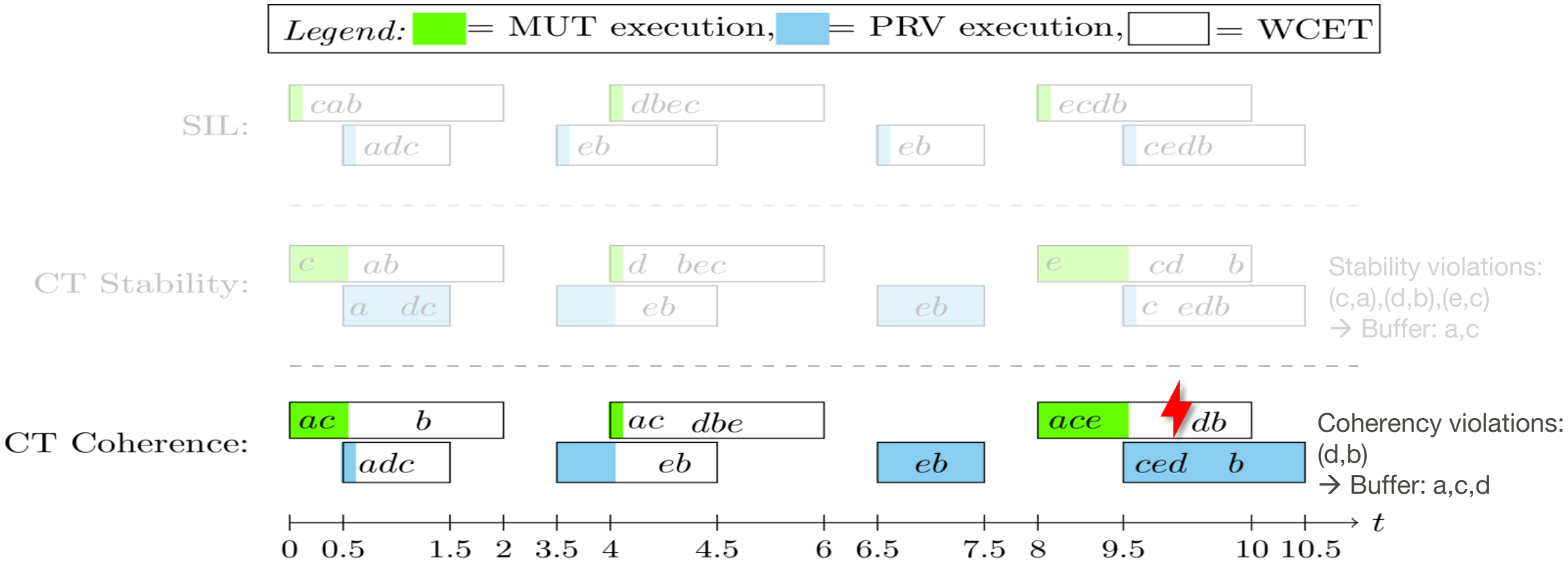
> Consistency sets: C_0={a,c,e}, C_1={b,d}



vitesco TECHNOLOGIES

EXAMPLE

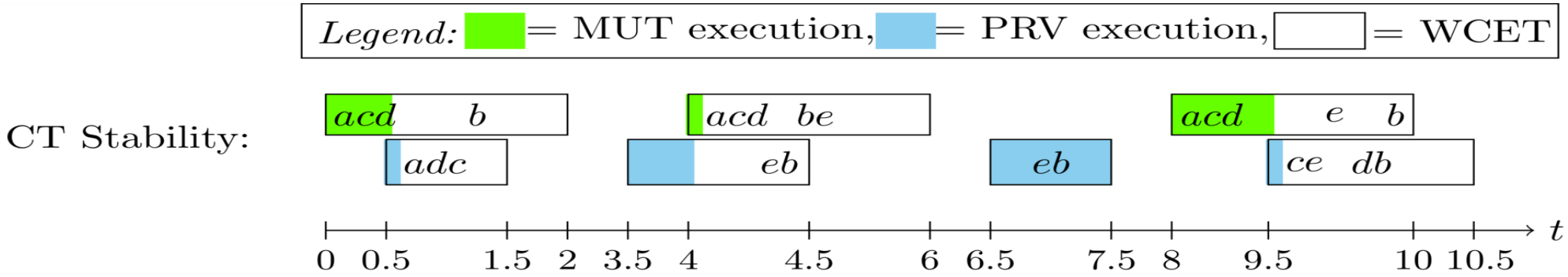
> Consistency sets: C_0={a,c,e}, C_1={b,d}



vitesco TECHNOLOGIES

EXAMPLE

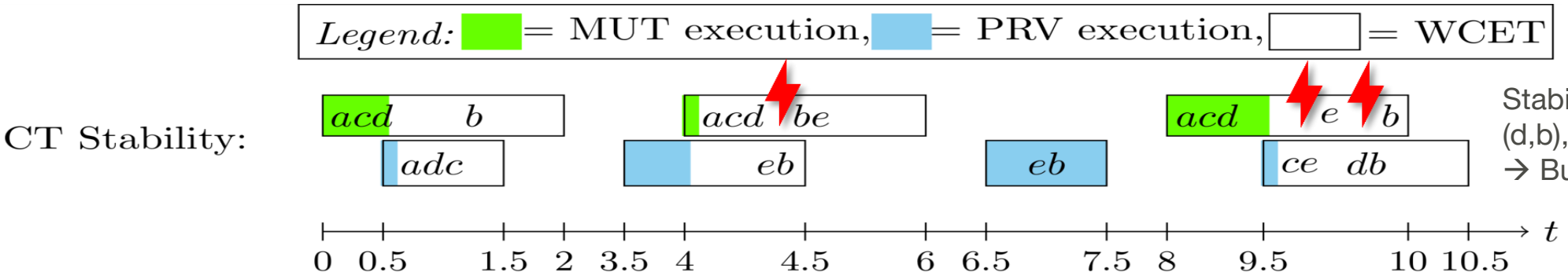
> Consistency sets: C_0={a,c,e}, C_1={b,d}



vitesco TECHNOLOGIES

EXAMPLE

> Consistency sets: C_0={a,c,e}, C_1={b,d}

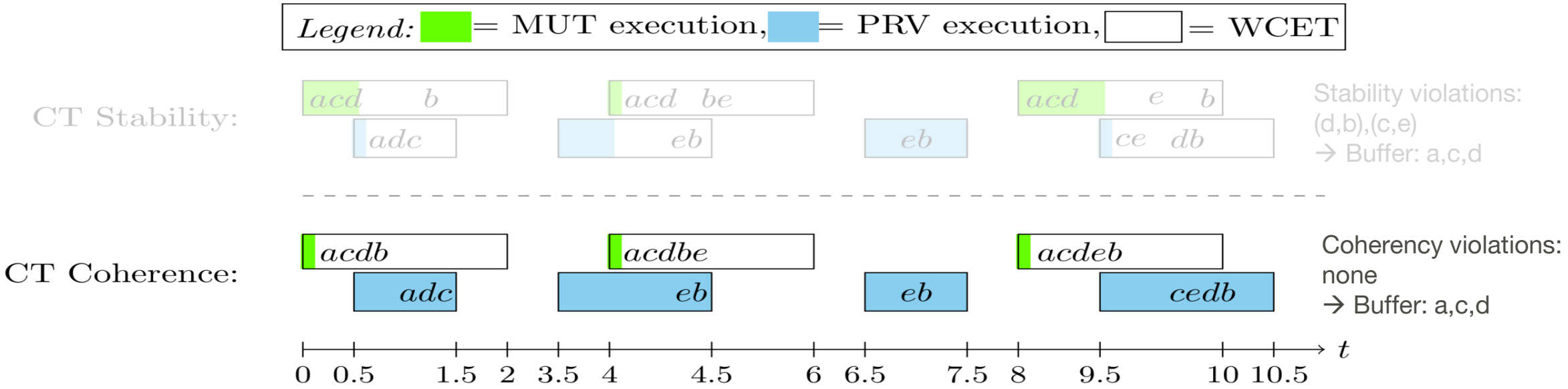


Stability violations:
(d,b),(c,e)
→ Buffer: a,c,d

vitesco TECHNOLOGIES

EXAMPLE

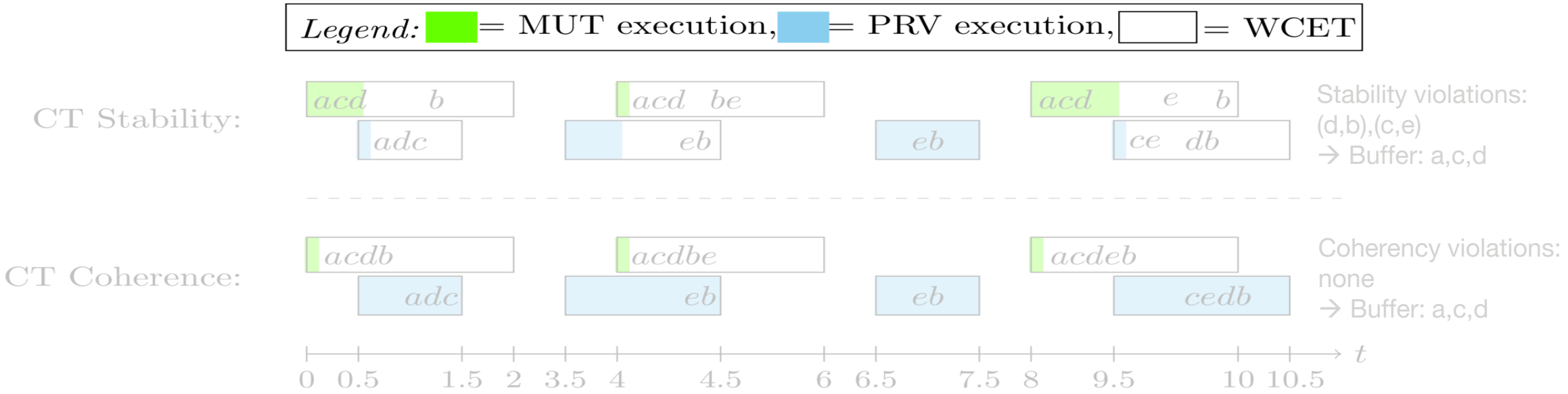
> Consistency sets: C_0={a,c,e}, C_1={b,d}



vitesco TECHNOLOGIES

EXAMPLE

> Consistency sets: C_0={a,c,e}, C_1={b,d}



> → Buffering requirements for **a,c,d**

vitesco TECHNOLOGIES

IDENTIFICATION OF CONSISTENCY REQUIREMENTS

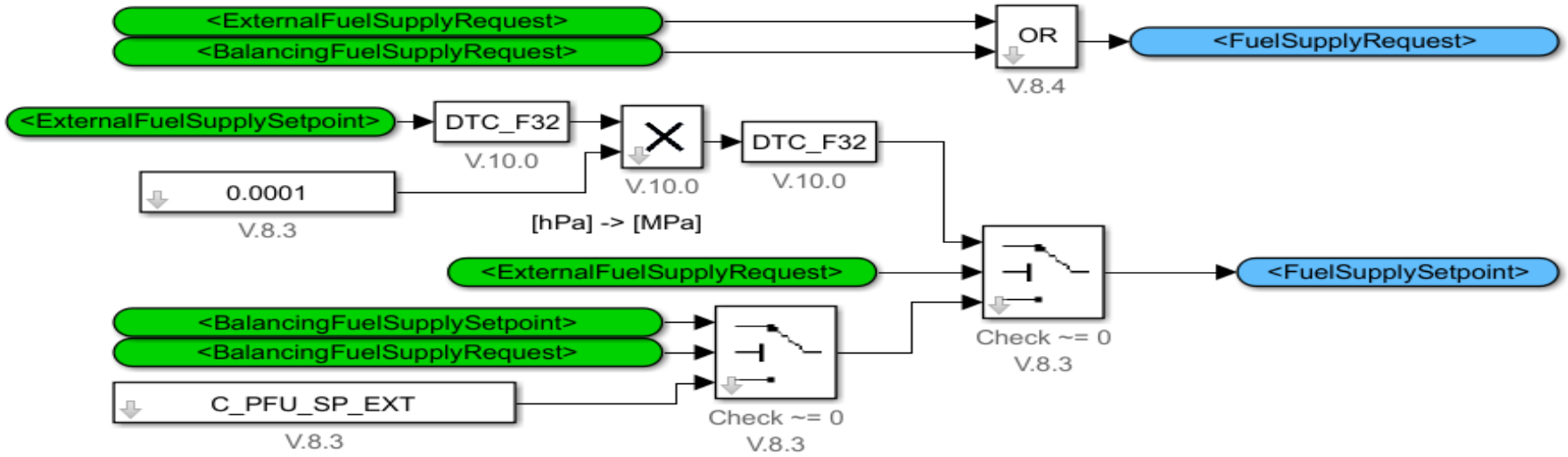
TESTING WORKFLOW

Starting point	ConsisTest model generation	Testing
› Standard Simulink SIL model › MUT and PRV software is built into an S-function › Testcases	› Replace original S-function with Validator S-function in the Simulink model › Instrument the MUT and PRV software at the access points › Generate Validator glue code, set parameters of virtual execution platform › Build generated and instrumented software together with Validator library into a separate executable	› Set runtime configuration: test case, alternative CSs and WCETs, protection levels for inputs › Run test group › Evaluate results: view report on PAPs and output comparison › Decide on protection levels of variables and repeat tests, if necessary

Seamless Workflow in existing Simulink Models

USE CASE: POWERTRAIN CONTROL FUNCTION

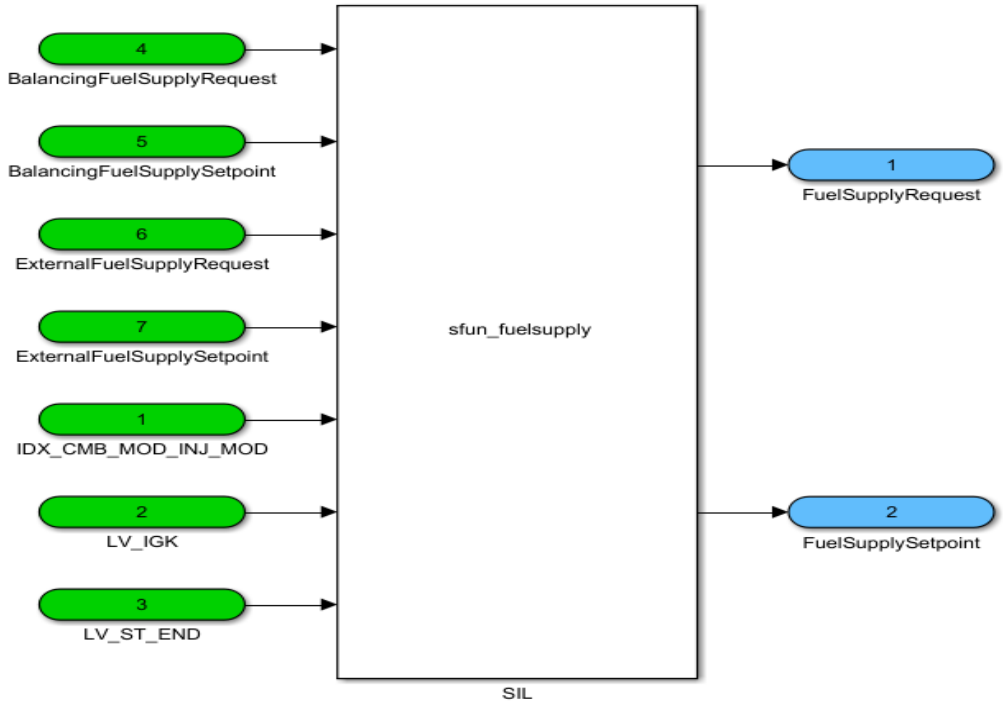
>MUT: periodic (10ms)
>PRV: event-triggered (crank-angle event)



vitesco TECHNOLOGIES

USE CASE: POWERTRAIN CONTROL FUNCTION

>SIL model with test configuration



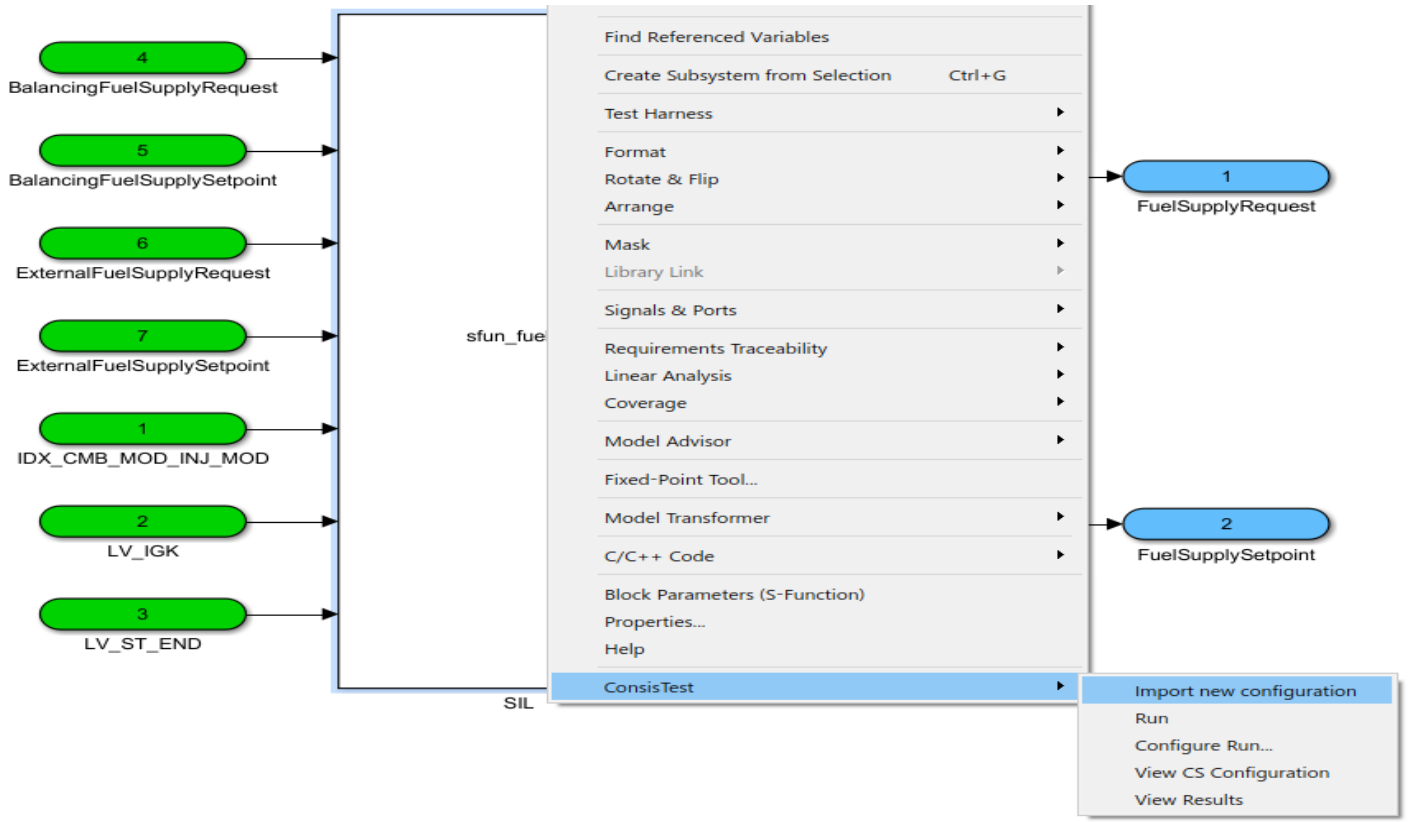

```
<MUT name="fuelsupply_10ms" index="0" func="fuelsupply_10ms">
    <Input var="ExternalFuelSupplyRequest"/>
    <Input var="ExternalFuelSupplySetpoint"/>
    <Input var="BalancingFuelSupplyRequest"/>
    <Input var="BalancingFuelSupplySetpoint"/>
    <Input var="lv_igk"/>
    <Input var="lv_st_end"/>
    <Input var="idx_cmb_mod_inj_mod"/>
</MUT>
<CSConfig id="0" name="ID0">
    <MUTConf mutRef="fuelsupply_10ms">
        <CS name="CS_seg">
            <Var ref="ExternalFuelSupplyRequest"/>
            <Var ref="ExternalFuelSupplySetpoint"/>
            <Var ref="BalancingFuelSupplyRequest"/>
            <Var ref="BalancingFuelSupplySetpoint"/>
            <Var ref="lv_igk"/>
        </CS>
        <CS name="CS_10ms">
            <Var ref="lv_st_end"/>
            <Var ref="idx_cmb_mod_inj_mod"/>
        </CS>
    </MUTConf>
</CSConfig>
<ETConfig id="0" name="HM6M1">
    <ExecTimePrv ref="PRV_ttrig_0_seg" minET="0us" WCET="100us"/>
    <ExecTimeMUT ref="fuelsupply_10ms" minET="11.62us" WCET="65.26us"/>
</ETConfig>
```


vitesco TECHNOLOGIES

USE CASE: POWERTRAIN CONTROL FUNCTION

>SIL model with test configuration

>Apply test configuration on SIL model

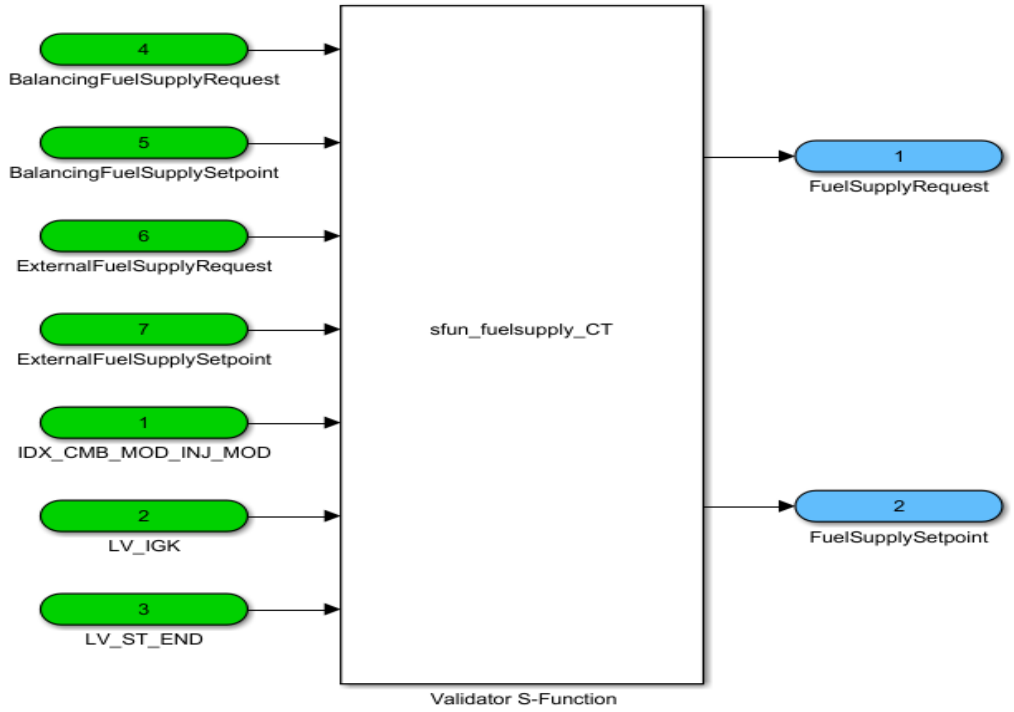



```
ly_10ms" index="0" func="fuelsupply_10ms">
ernalFuelSupplyRequest"/>
ernalFuelSupplySetpoint"/>
ancingFuelSupplyRequest"/>
ancingFuelSupplySetpoint"/>
igk"/>
st_end"/>
_cmb_mod_inj_mod"/>

me="ID0">
="fuelsupply_10ms">
s_seg">
f="ExternalFuelSupplyRequest"/>
f="ExternalFuelSupplySetpoint"/>
f="BalancingFuelSupplyRequest"/>
f="BalancingFuelSupplySetpoint"/>
f="lv_igk"/>

s_10ms">
f="lv_st_end"/>
f="idx_cmb_mod_inj_mod"/>

me="HM6M1">
f="PRV_ttrig_0_seg" minET="0us" WCET="100us"/>
f="fuelsupply_10ms" minET="11.62us" WCET="65.26us"/>
```


vitesco TECHNOLOGIES

USE CASE: POWERTRAIN CONTROL FUNCTION

>SIL model with replaced S-Function

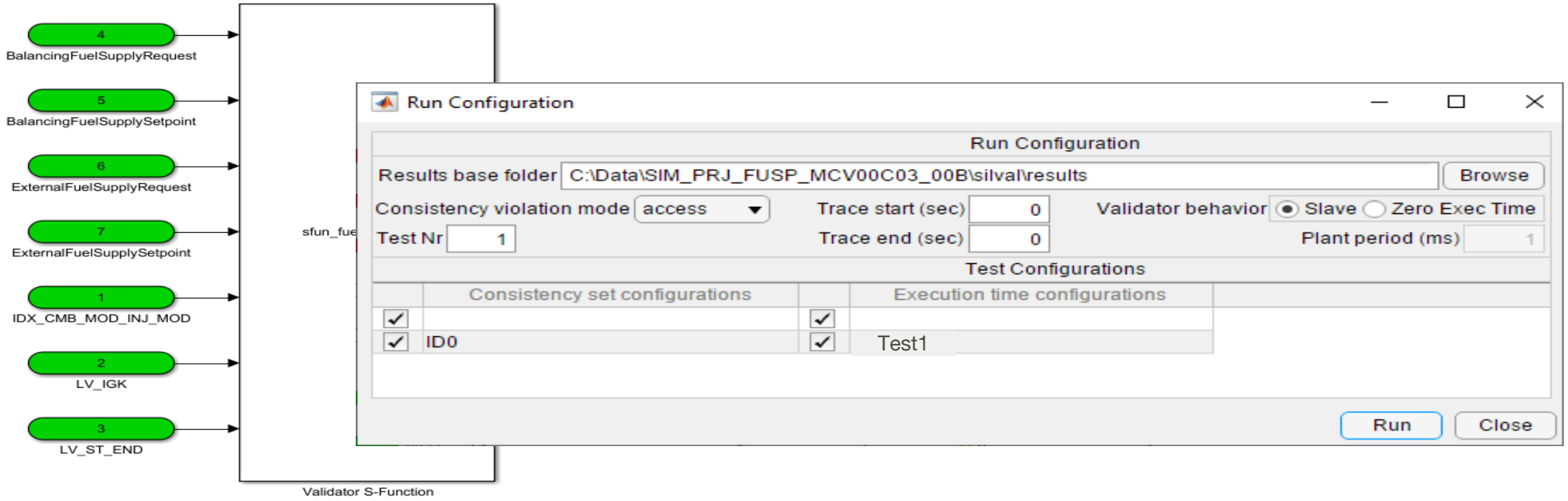


vitesco TECHNOLOGIES

USE CASE: POWERTRAIN CONTROL FUNCTION

>SIL model with replaced S-Function

>Execute test runs



vitesco TECHNOLOGIES

TEST RESULTS

silval - sfun/results/MonitoringResults_19_10_14_111807/MUTC/CTReport.csxml - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Chrona Window Help

Project Ex... | Connections

- fuelsupply
  - Binaries
  - Archives
  - Includes
  - bin
  - results
    - MonitoringResults_19_10_14_111807
  - src
    - application
      - application
        - CTConfig_default.xml
        - model_data.xml
    - include
    - valgen
  - cosim.h
  - CTConfig_sfun.xml
  - inputs.vin
  - sfun.ovd
  - val_build_conf.bat
  - val_run_conf.xml

CTReport.csxml

Consistency Mode: MUT Consistency **Violation Mode:** Access-based

Module > CS-Config > CSet

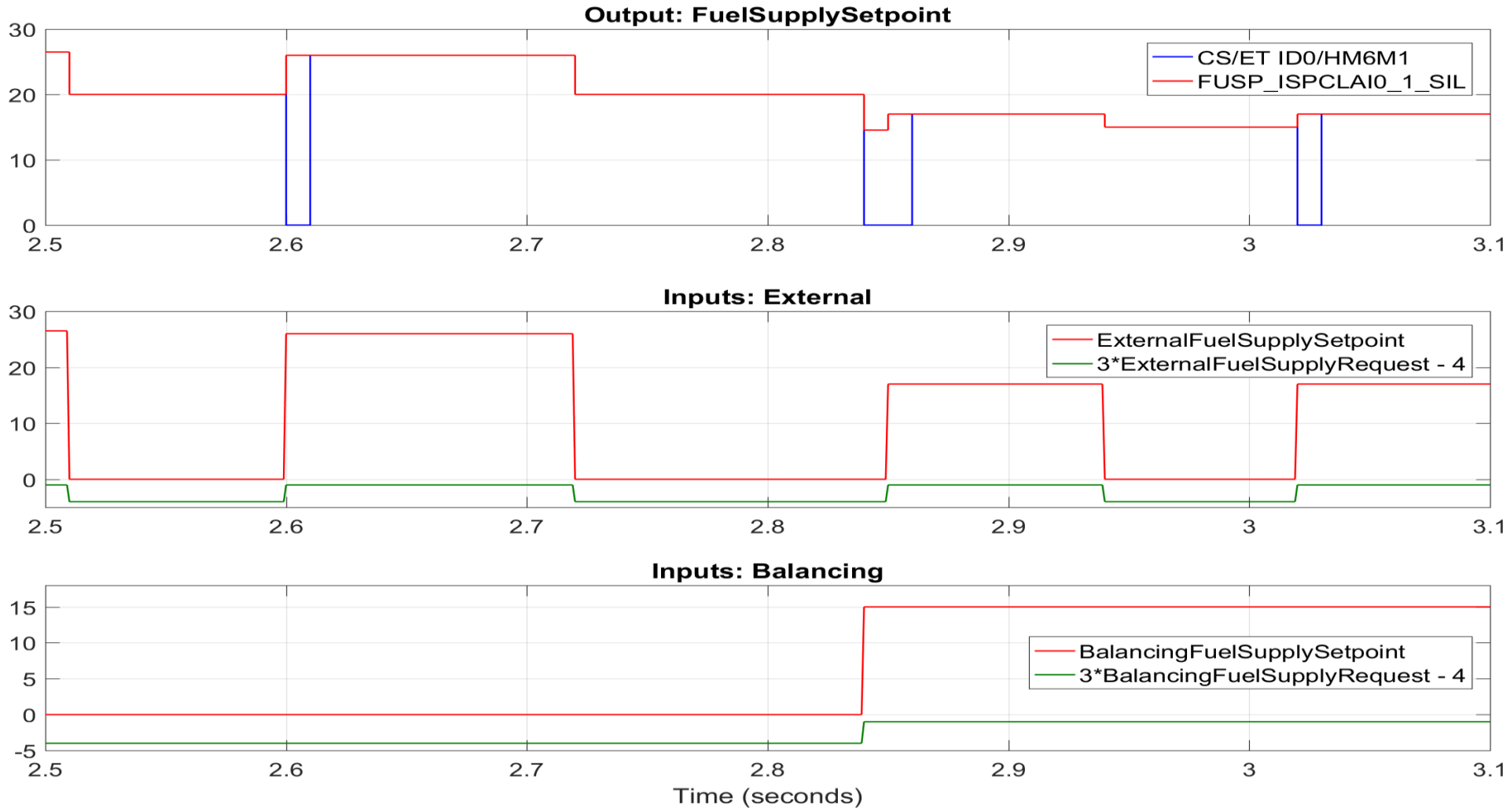
Module	CS-Config	CSet	Variable	Protection Used	Protection Revised	Test1
fusp_ispclai0__systrig_10mst2						
ID0						
CS_10ms						
fusp_ispclai0__systrig_10mst2	ID0	CS_10ms	idx_cmb_mod_inj_mod	test	test	
fusp_ispclai0__systrig_10mst2	ID0	CS_10ms	lv_st_end	test	test	
CS_seg						
fusp_ispclai0__systrig_10mst2	ID0	CS_seg	BalancingFuelSupplyRequest	test	test	
fusp_ispclai0__systrig_10mst2	ID0	CS_seg	BalancingFuelSupplySetpoint	test	test	
fusp_ispclai0__systrig_10mst2	ID0	CS_seg	ExternalFuelSupplyRequest	test	test	
fusp_ispclai0__systrig_10mst2	ID0	CS_seg	ExternalFuelSupplySetpoint	test	test	
fusp_ispclai0__systrig_10mst2	ID0	CS_seg	lv_igk	test	test	

0 items selected

115M of 256M

vitesco TECHNOLOGIES

INPUT AND OUTPUT SIGNAL TRACES



vitesco TECHNOLOGIES

TESTING EFFORT CONSIDERATION

> Run on an INTEL i7 with 2.7GHz and 32GB RAM
> SIL environment setup: ~60min (one time effort)
> One test case execution: 12secs–3.5min (1min avg.)
> Evaluation of test results: ~30min
> On average 5 test cases per module
> One module is on average reused in 10 projects

> Additional testing overhead introduced per module: **140min**