Universität Bremen*

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

FZI

# Constraint-based Platform Variants Specification for Early System Verification

Andreas Burger[1], Alexander Viehl[1],
Andreas Braun[1], Finn Haedicke[2,4], Daniel Große[2],
Oliver Bringmann[1,3] and Wolfgang Rosenstiel[1,3]

[1]FZI Research Center for Information Technologies
[2]solvertec GmbH
[3]University of Tuebingen
[4]Institute of Computer Science, University of Bremen

Federal Ministry of Education and Research

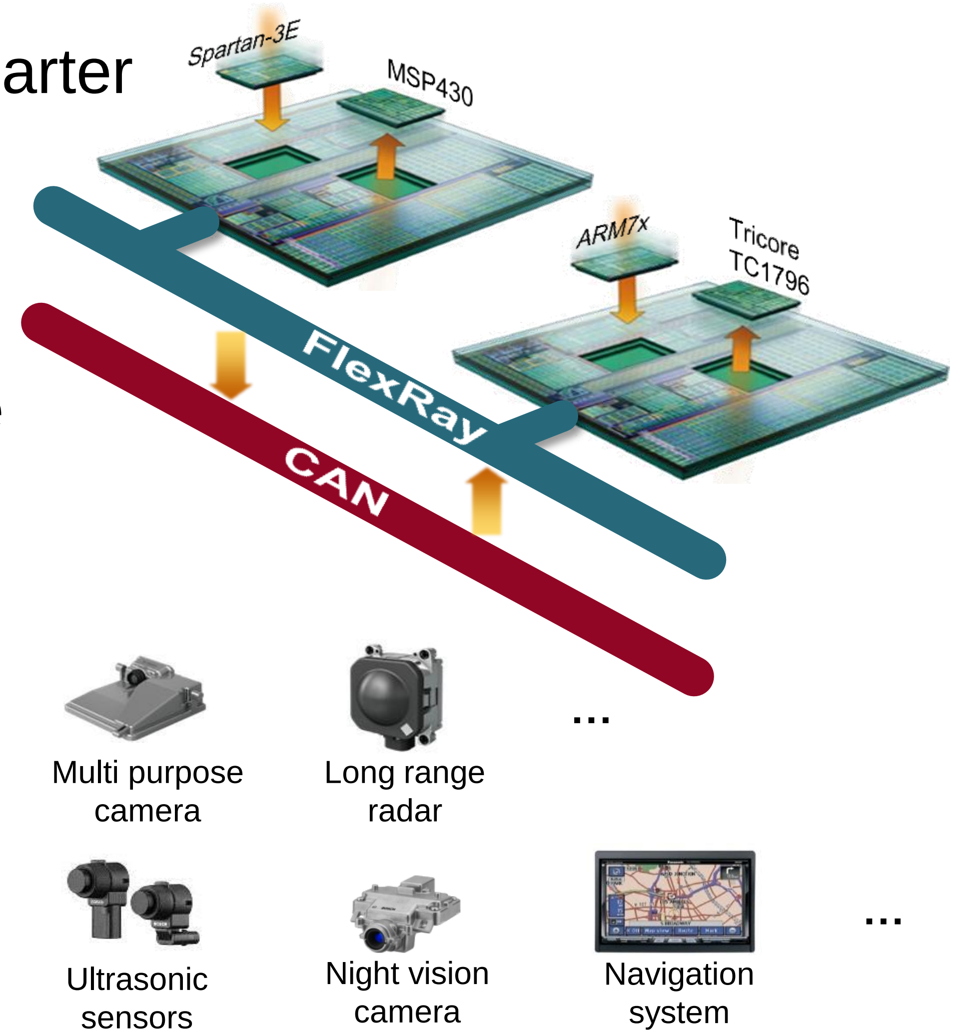
RES 2.0 CAR
Robustheit im Bereich Elektromobilität

16M3195E

FZI FORSCHUNGSZENTRUM INFORMATIK

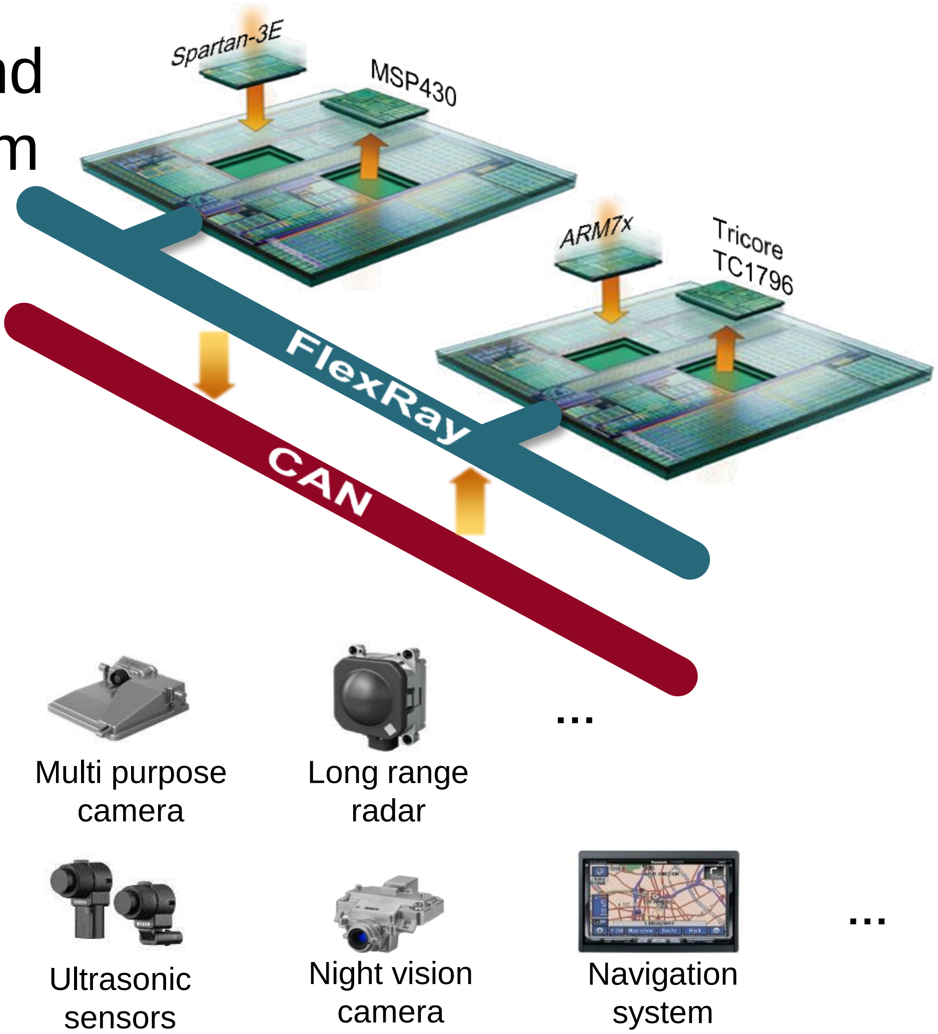
# Design Challenges for Automotive Electronics

- Vehicles have become smarter over the last years
- Significant increase of software in the automotive
  - Multi-sensor data fusion
  - Complex image recognition algorithms
  - Usage of background information (maps, GPS)
  - Situation perception, interpretation & reasoning



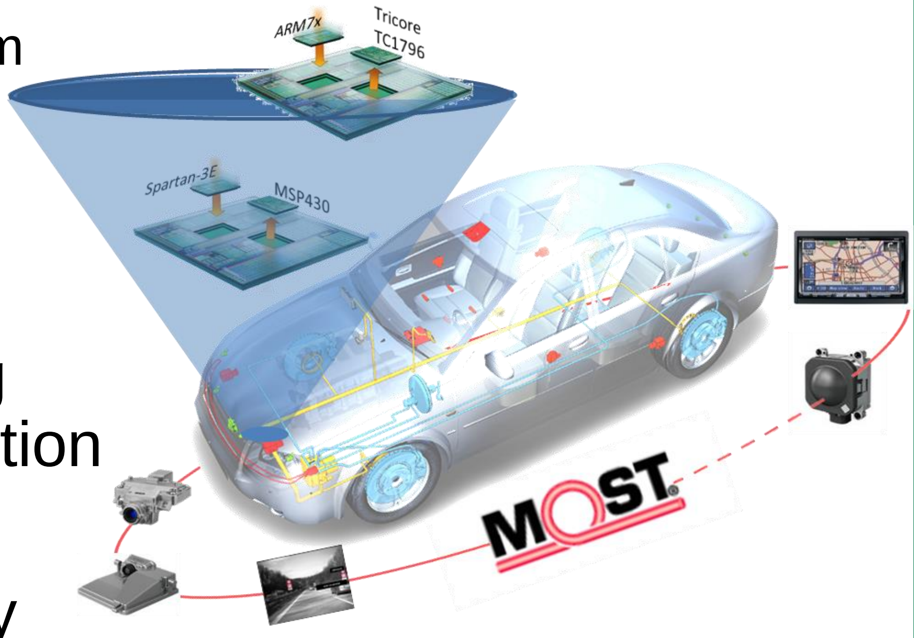
# Design Challenges for Automotive Electronics

- Increase of reused logic and IP integration in the platform design composition
- Significant increase of platform variant and configuration space, e.g.:
  - 6.4 million valid variants of an automatic gear shifting application in Daimler Trucks
  - $10^{21}$ valid MOST network variants



# Design Challenges for Automotive Electronics

- New challenges in verification, exploration and test:
  - Huge variant spaces
  - Verification of IP-Blocks in different platforms
  - Interaction of different IP-Block instances
  - Verification of different platform characteristics (e.g., software versions, component parameter, etc.)




- Therefore virtual prototyping can be used in early verification

- Hence focus is moving away from fixed virtual platforms to variable virtual platforms

# Constraint-based Platform Variant Specification

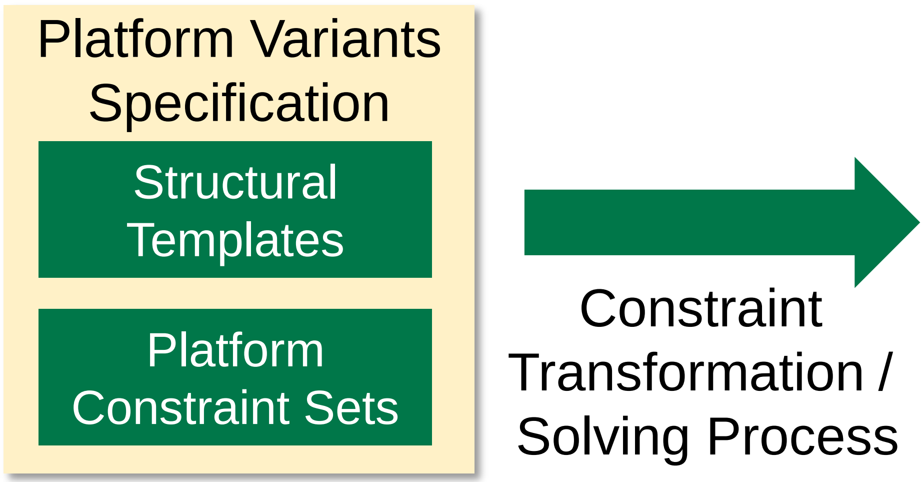
Platform Variants Specification

Structural Templates

Platform Constraint Sets

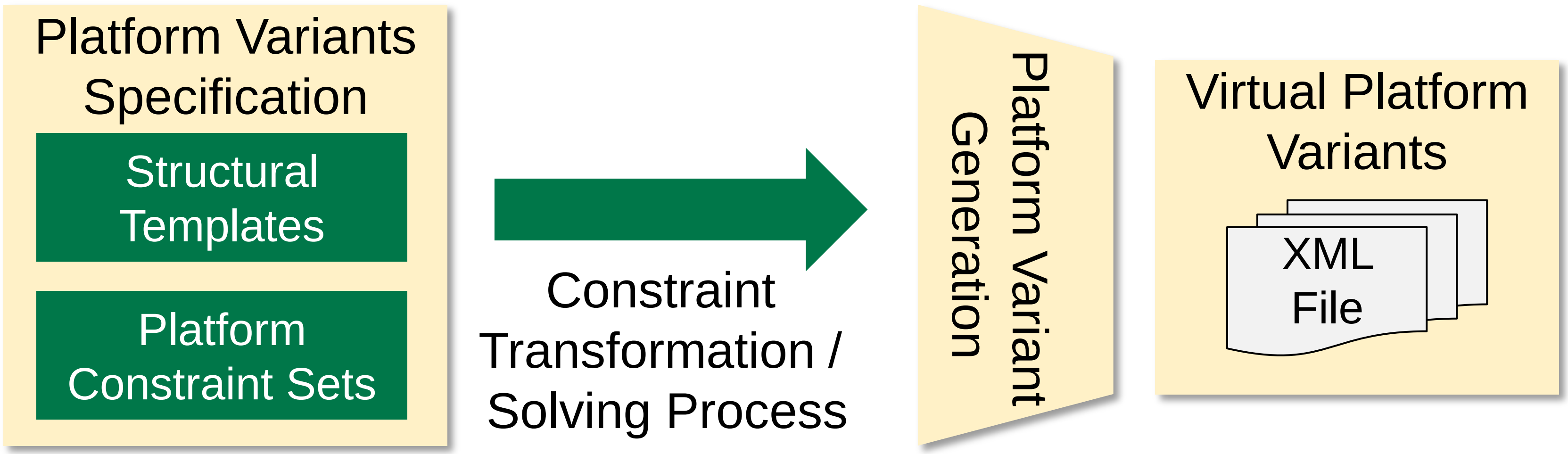
- Platform Variants Specification

# Constraint-based Platform Variant Specification



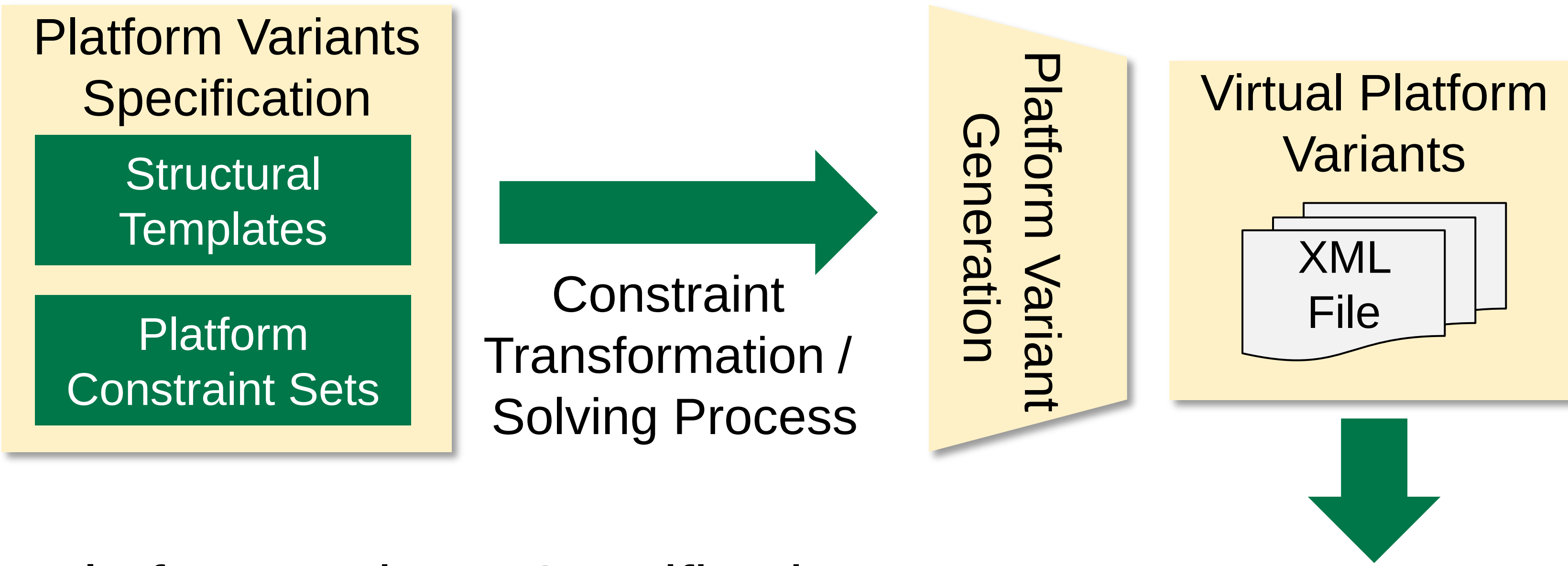

- Platform Variants Specification
- Platform Constraints

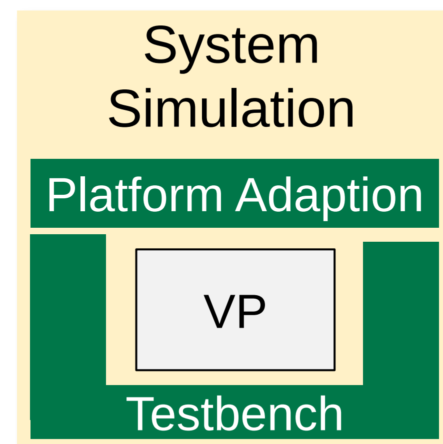
# Constraint-based Platform Variant Specification




- Platform Variants Specification
- Platform Constraints
- Platform Variant Generation

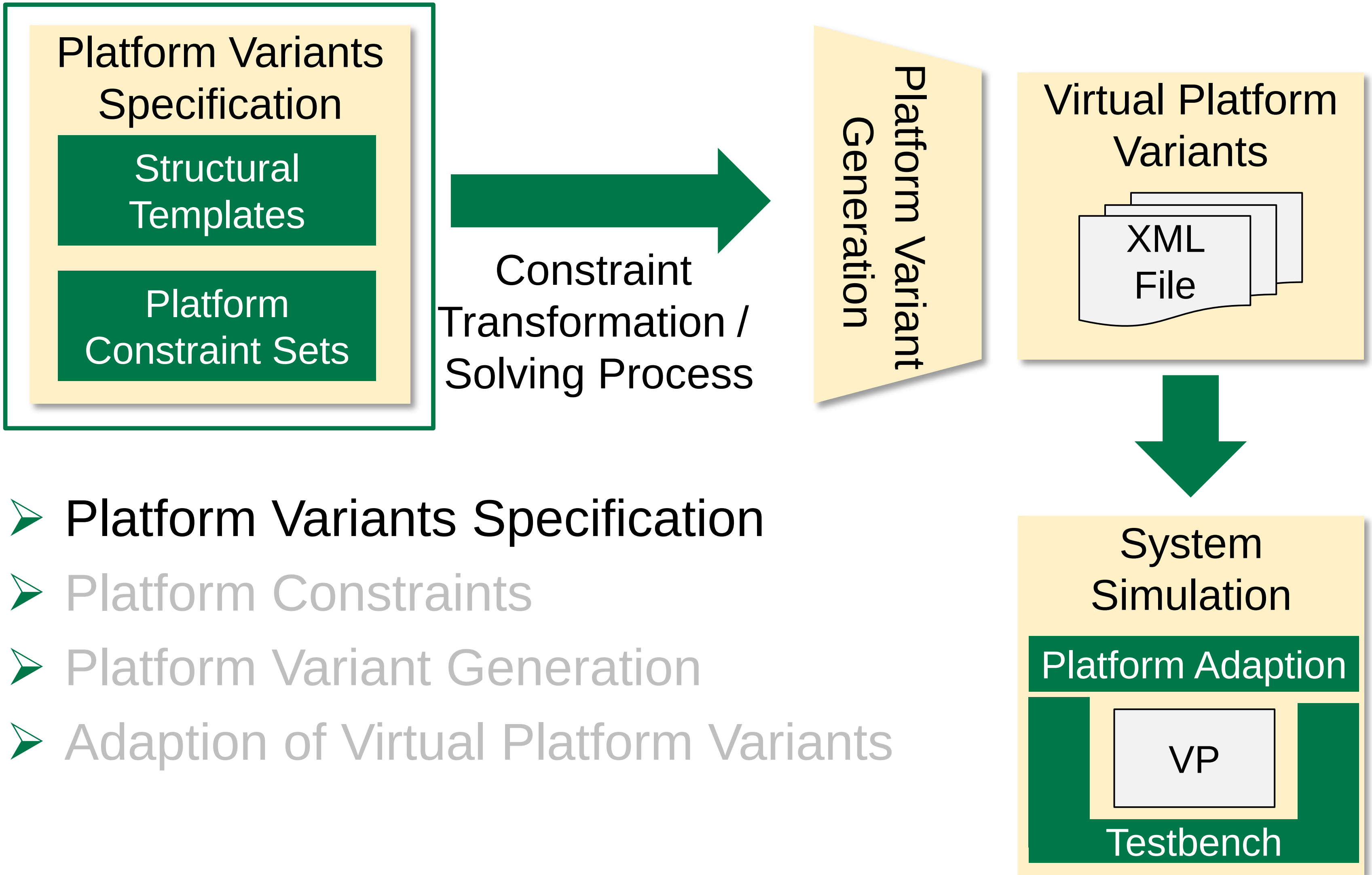
# Constraint-based Platform Variant Specification




- Platform Variants Specification
- Platform Constraints
- Platform Variant Generation
- Adaption of Virtual Platform Variants

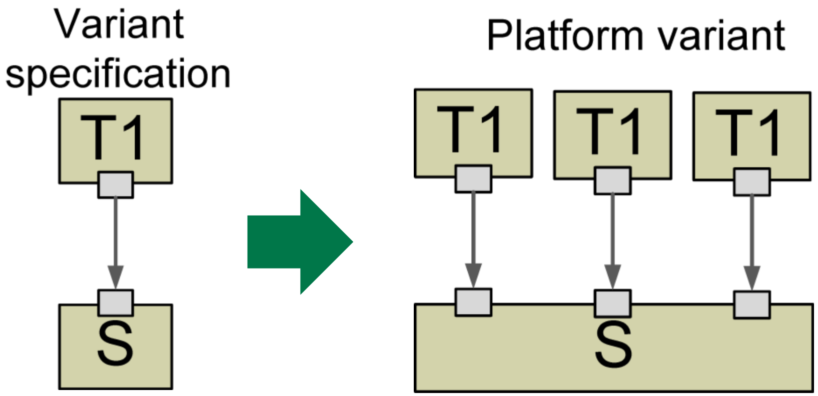
# Constraint-based Platform Variant Specification




- Platform Variants Specification
- Platform Constraints
- Platform Variant Generation
- Adaption of Virtual Platform Variants

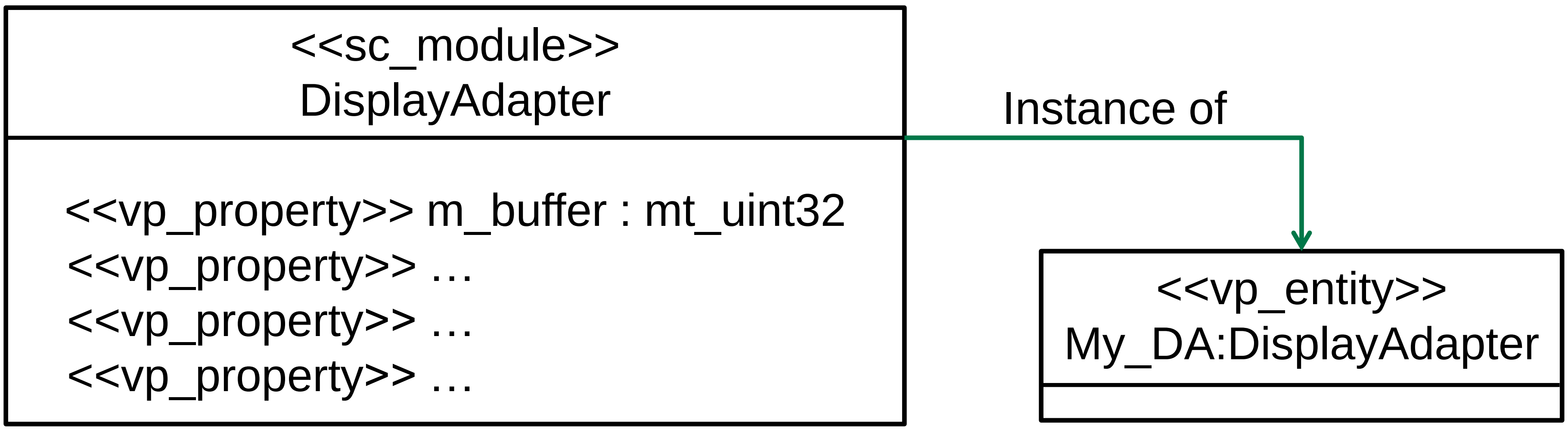
# Platform Variants Specification

- Model-based description approach
- Specification of platform variants structure
  - Hierarchical structured Templates based on UML
    - Platform UML Profile
    - UML Class Diagrams
    - UML Composite Structure Diagrams



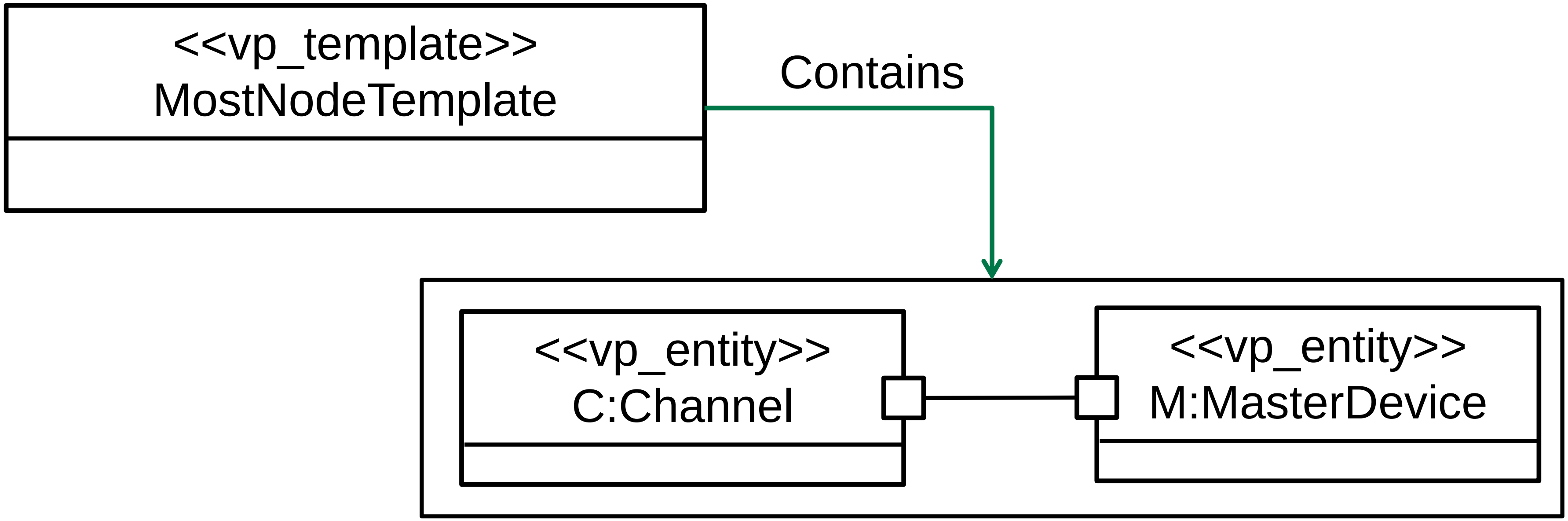

- Specification of different configuration possibilities
  - Attached constraint sets to specify feasible variants and configuration parameter

# Platform Variants Specification



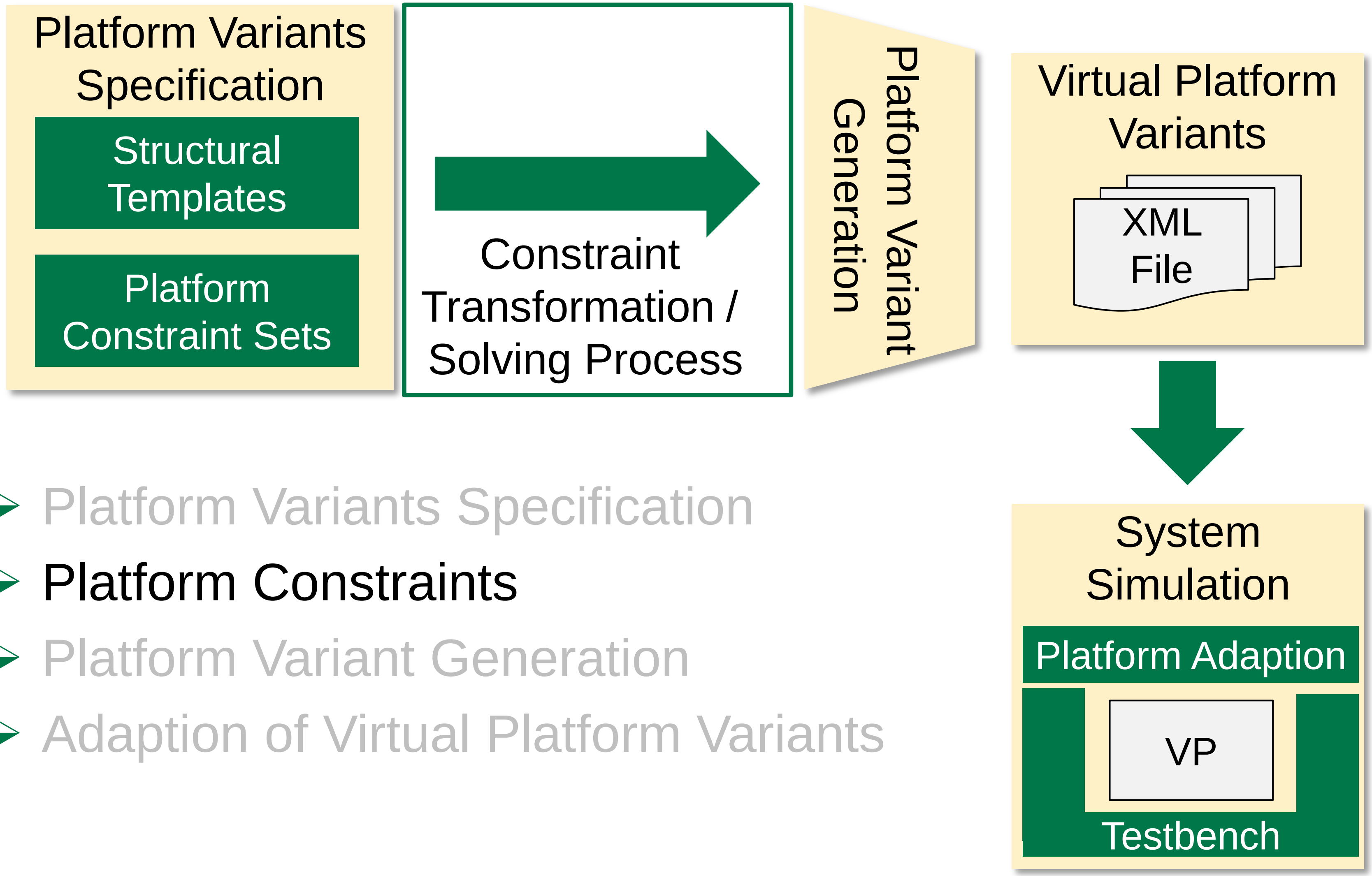

- UML Profile defines different platform types
- <<vp_property>>
  specifies parameter which can be configured
- <<sc_module>>
  virtual prototype modules  are defined as UML::Classes
- <<vp_entity>>
  specifies instance of a virtual prototype module

# Platform Variants Specification




- UML Profile defines different platform types
- <<vp_template>>
  abstracts a part of a system for simplicity, variability or structural reasons
- << vp_template >> contains entities and even other templates and can be used in variants specification

# Constraint-based Platform Variant Specification




- Platform Variants Specification
- Platform Constraints
- Platform Variant Generation
- Adaption of Virtual Platform Variants

# Platform Constraints

- Constraints are specified by an extended subset of the Object Constraint Language (OCL)
- OCL commonly defines constraints at the M1 layer of Meta Object Facility (MOF)

| | Standard MOF Layer | Platform Meta Layer |
|---|---|---|
| M3 | Meta Meta Model | UML Meta Model |
| M2 | UML Meta Model | Platform Templates / Profile |
| M1 | User-defined UML- / Object-models | Platform Variant Specification |
| M0 | Distinctive Data | Platform Variant Space |

# Platform Constraints – P-OCL

- OCL subset supports:
  - Boolean operators:

$$<, >, <=, <>, \\ and, or, if - then - else, \ldots$$

  - OCL Collection operators:

$$\mathbf{includes()}, \mathbf{size()}, \ldots$$

- OCL Extensions:
  - Probability distribution operators for OCL Collection-Type *Sequence:*

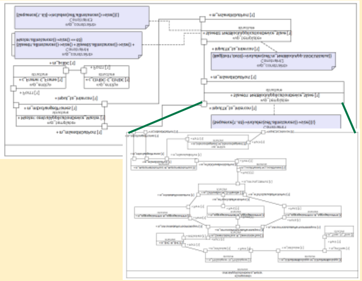
$$\mathbf{gaussian()}, \ldots$$

  - Special Template-Operators:

$$\mathbf{active()}, \ldots$$

# Platform Constraints – Transformation and Solving Process

Platform Variant Specification



Constraint Transformation

Python metaSMT

Dynamical insertion of constraints to avoid duplicate solutions

- Constraint transformation in Boolean formulas to use SMT/SAT Solver (metaSMT, Z3, PicoSAT, etc.)
- Automatically transformation in Quantifier-free bit-vector (QF-BV) logic

# Platform Constraints – Transformation

- Transformation of P-OCL constraints in Quantifier-free bit-vector (QF-BV) logic
- Numbers are converted in bit-vectors
- QF-BV logic is expressed in metaSMT Python code
- P-OCL Example:

# Platform Constraints – Transformation

- Transformation of P-OCL constraints in Quantifier-free bit-vector (QF-BV) logic
- Numbers are converted in bit-vectors
- QF-BV logic is expressed in metaSMT Python code
- P-OCL Example:

```
Sequence{2..16}->select(e | e/2 = 0)->
includes(self.allInstances()->size())
```

- Boolean formula:

$$y \geq \vec{a} \ \& \ y \leq \vec{b} \ \& \ (y - \vec{a}) \ \% \ \vec{s} == \vec{0}$$

- Whereby:

$$\vec{a} \equiv conv^{-1}(2), \vec{b} \equiv conv^{-1}(16) \ and \ \vec{s} \equiv conv^{-1}(2)$$

# Platform Constraints – Transformation

- Transformation of P-OCL constraints in Quantifier-free bit-vector (QF-BV) logic
- Numbers are converted in bit-vectors
- QF-BV logic is expressed in metaSMT Python code
- P-OCL Example:

```
Sequence{2..16}->select(e | e/2 = 0)->
includes(self.allInstances()->size())
```

- Boolean formula:

$$y \geq \vec{a} \;\&\; y \leq \vec{b} \;\&\; (y - \vec{a}) \;\%\; \vec{s} == \vec{0}$$

- Whereby:

$$\vec{a} \equiv conv^{-1}(2), \vec{b} \equiv conv^{-1}(16) \;\; and \;\; \vec{s} \equiv conv^{-1}(2)$$

# Platform Constraints – Transformation

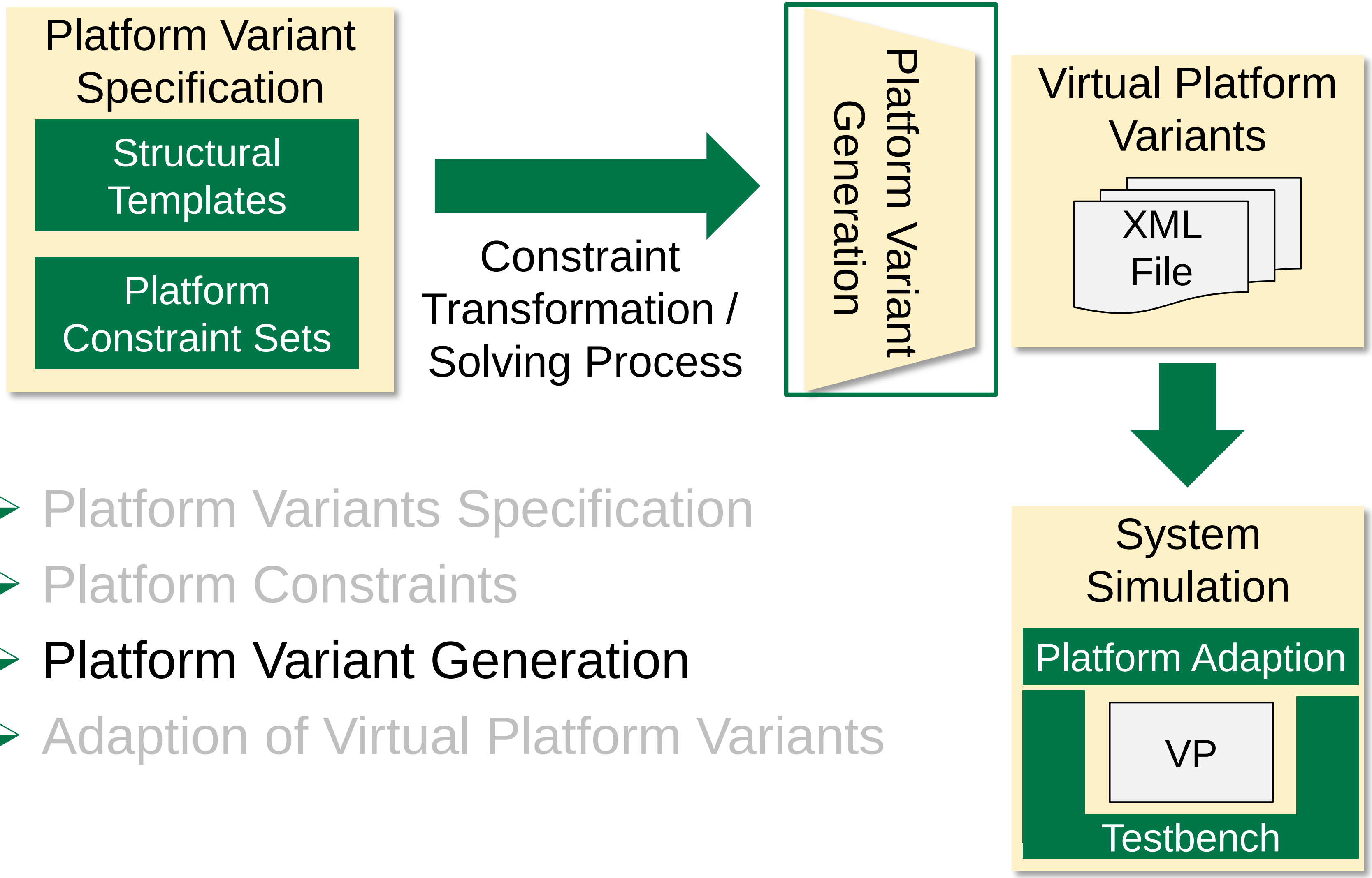
- Transformation of P-OCL constraints in Quantifier-free bit-vector (QF-BV) logic
- Numbers are converted in bit-vectors
- QF-BV logic is expressed in metaSMT Python code
- P-OCL Example:

```
Sequence{2..16}->select(e | e/2 = 0)->
includes(self.allInstances()->size())
```

- metaSMT Syntax:

```
y >= bv_uint(2)[bw]
y <= bv_uint(16)[bw]
(y - bv_uint(2)[bw]) %
    bv_uint(2)[bw] == bv_uint(0)[bw]
```

# Constraint-based Platform Variant Specification




- Platform Variants Specification
- Platform Constraints
- Platform Variant Generation
- Adaption of Virtual Platform Variants

# Platform Variant Generation

- Solver solutions are provided as matrix

$$\begin{pmatrix} x_0 & x_1 & x_2 & x_3 & \dots & x_n \\ 2 & 3 & 54 & 235 & \dots & i \end{pmatrix} n \in \mathbb{N}; i \in \mathbb{N}$$

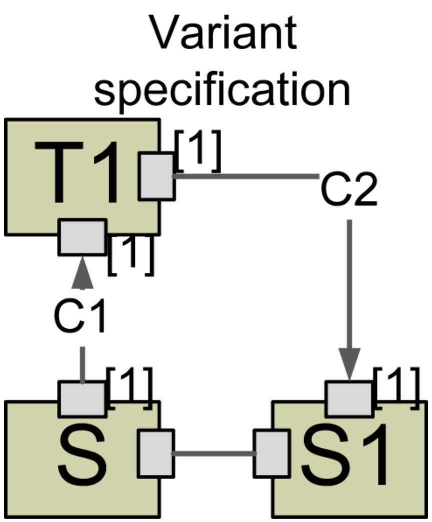
- Each variable represents a module, template or parameter specification

# Platform Variant Generation

- Variant generation example:
  - Ring-Topology
  - Template **T1** is specified variable by constraint:

$$\mathbf{Sequence}\{1..3\}->\mathbf{includes}(\mathbf{self.allInstances}()->\mathbf{size}())$$

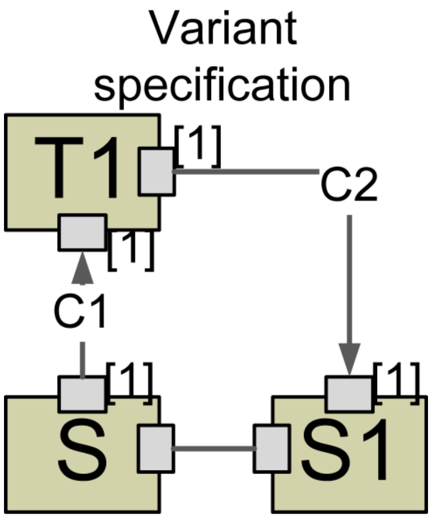
    - Whereby **`self`** refers to **T1**



# Platform Variant Generation

- Variant generation example:
  - Ring-Topology
  - Constraint is formalized in:

$$\begin{aligned} x0 &>= \text{bv\_uint}(1)[\text{bw}] \\ x0 &<= \text{bv\_uint}(3)[\text{bw}] \end{aligned}$$




- Solver solution:

$$\begin{pmatrix} x_0 & \cdots \\ 3 & \cdots \end{pmatrix}$$

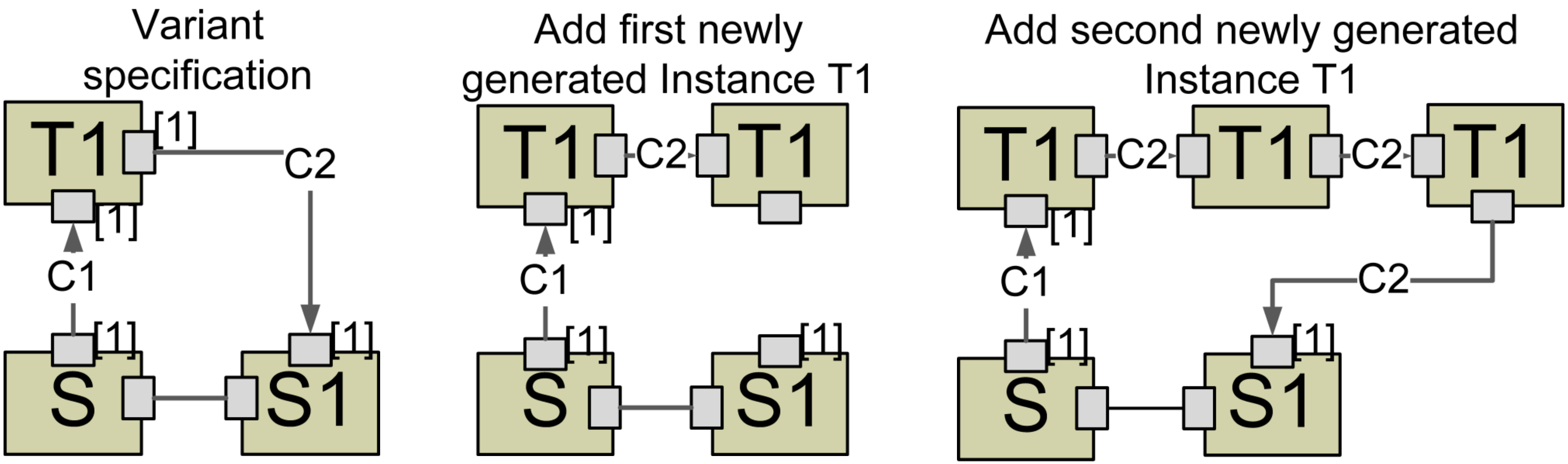
# Platform Variant Generation

- Variant generation example:
  - Ring-Topology
  - Constraint is formalized in:

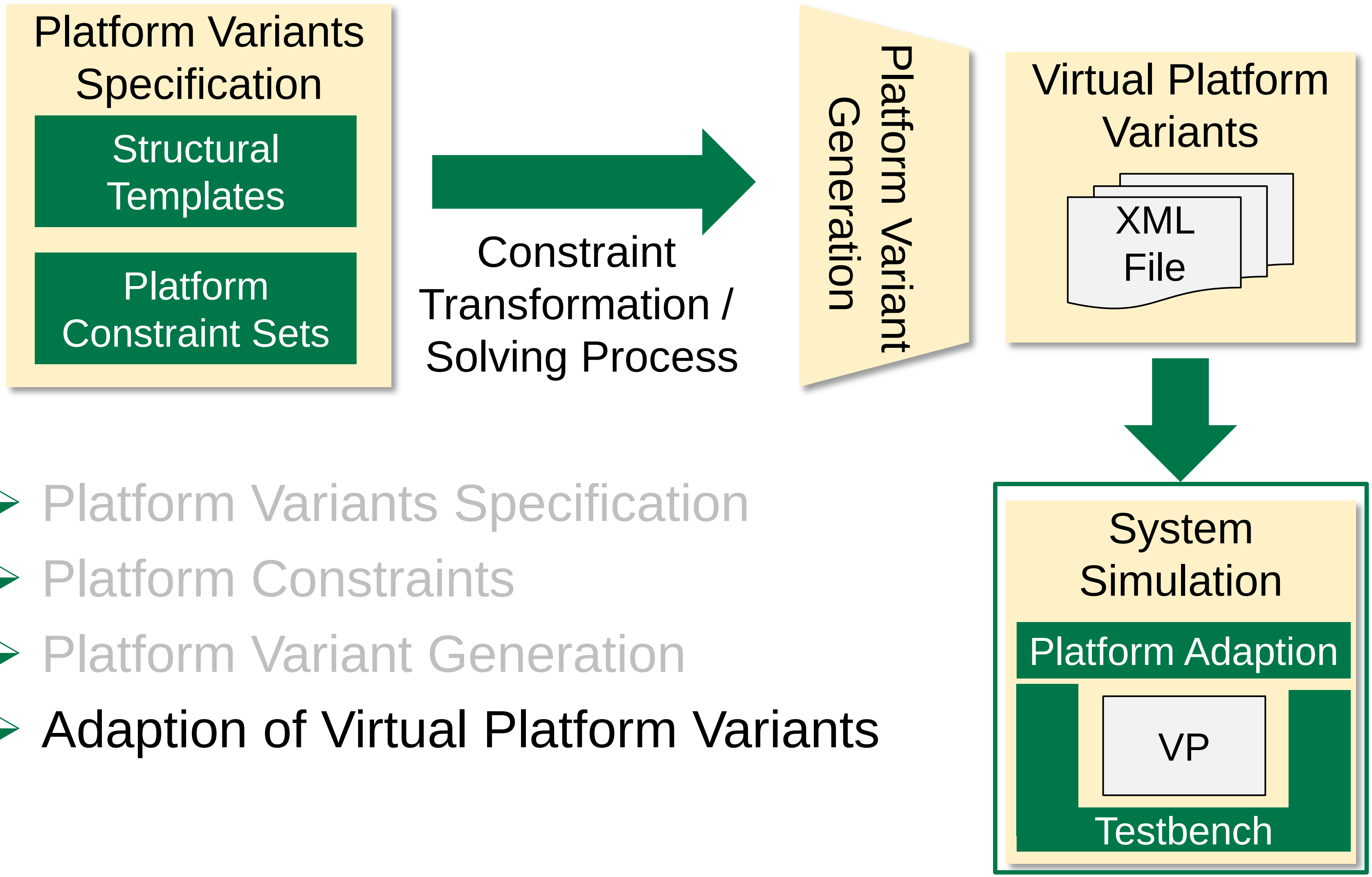
$$\begin{aligned} x0 &>= \mathrm{bv\_uint}(1)[bw] \\ x0 &<= \mathrm{bv\_uint}(3)[bw] \end{aligned}$$

- Solver solution:

$$\begin{pmatrix} x_0 & \dots \\ 3 & \dots \end{pmatrix}$$

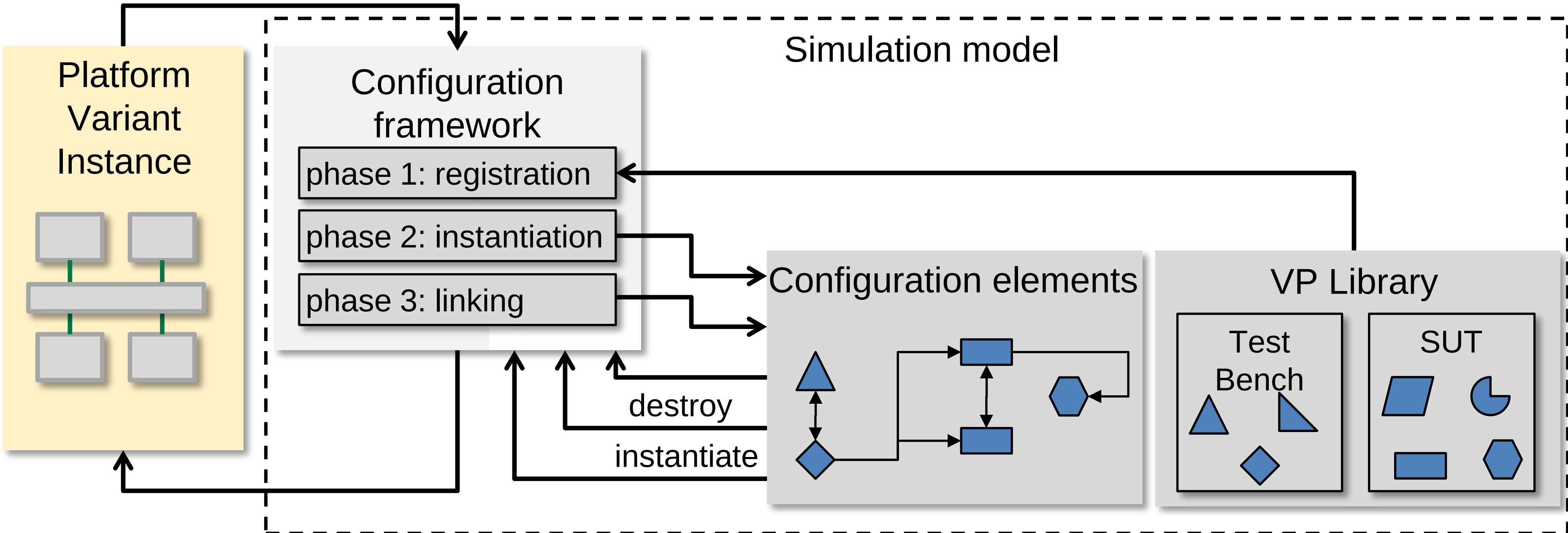


# Constraint-based Platform Variant Specification




- Platform Variants Specification
- Platform Constraints
- Platform Variant Generation
- Adaption of Virtual Platform Variants

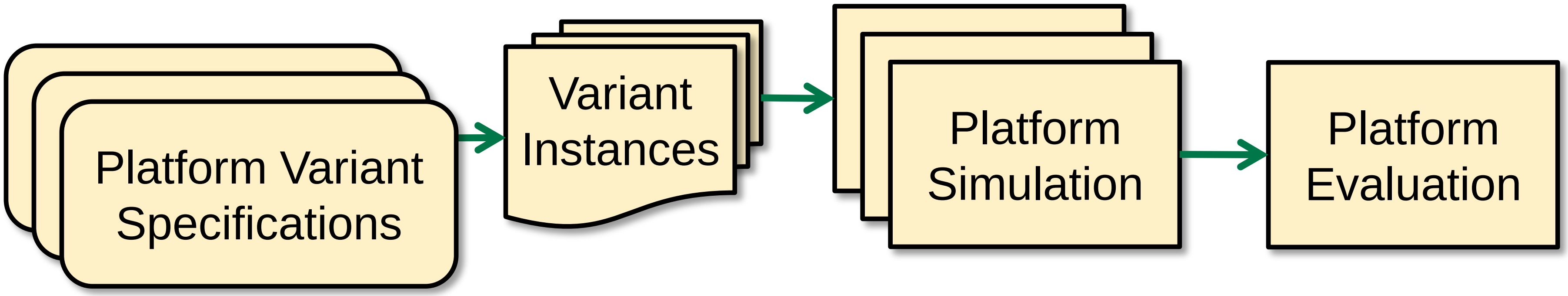
# Adaption of Platform Variants – Platform-Simulation Framework [1]




- Configuration of platform variants as virtual prototypes
  - Linking and instantiation of SystemC modules regarding the generated platform variant specification
- Dynamical reconfiguration during the simulation without recompilation

# Use Cases

- Media Oriented Systems Transport-Bus (MOST)
  - Simulation-based verification of implementation against specification:
    - Ring Break Diagnosis (RBD) Application
    - Central Component Application
- FlexRay
  - Exploration of a Camera and Recognize Module:
    - Traffic Sign Recognition (TSR)
- Verification Flow:

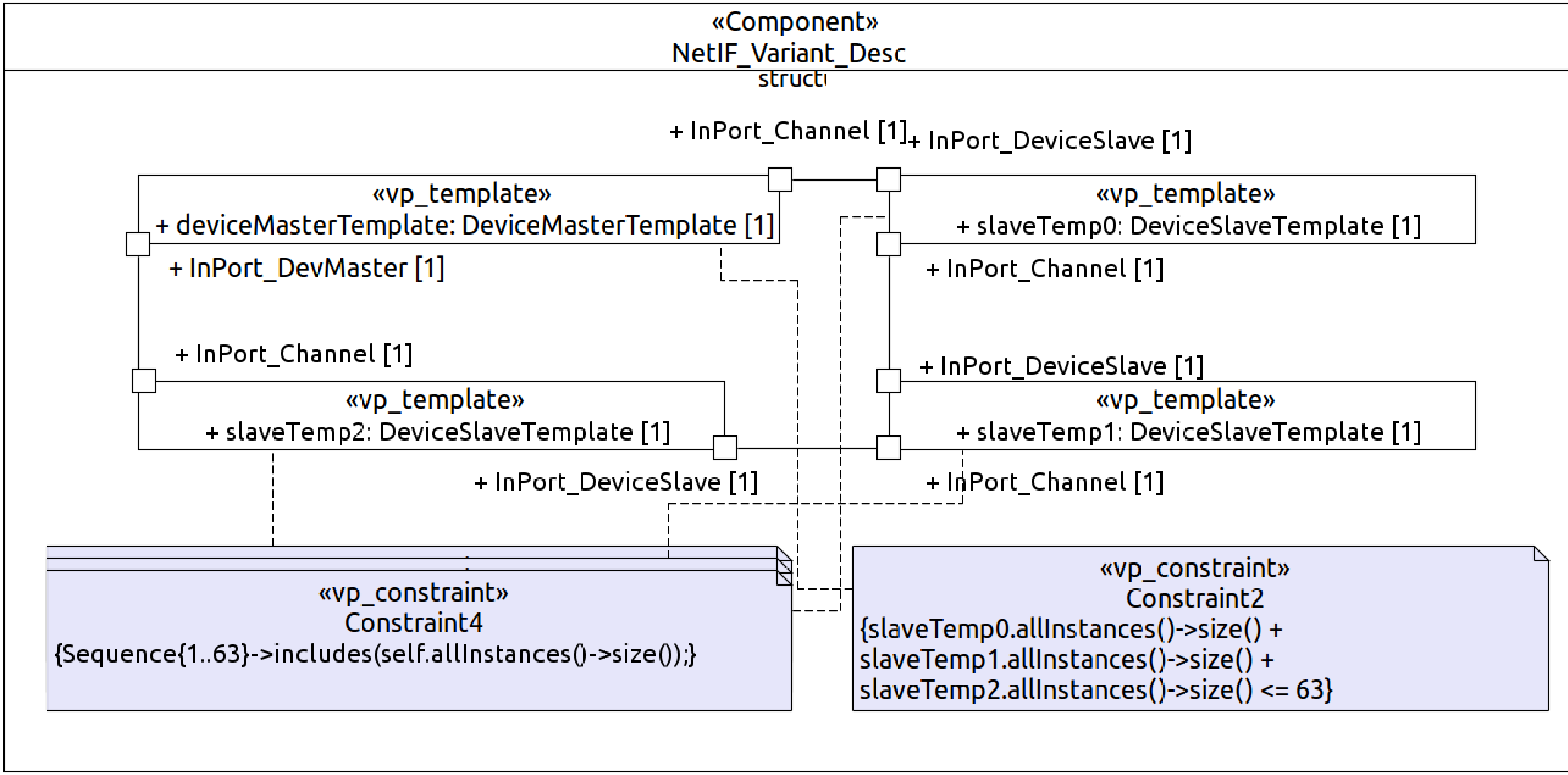


# Experimental Results – RBD Verification scenarios

- Six evaluation scenarios are turned out to be suggestive: *Error Free, Ring Break, Excessive Attenuation, Multi Master, All Slave, Combination* [1]

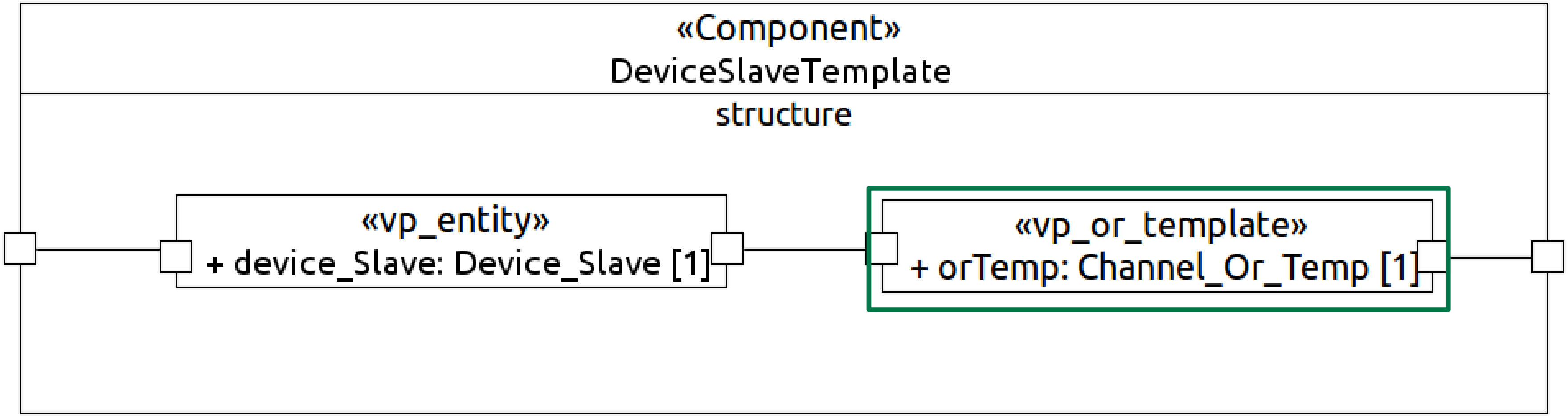
| Scenario | Variants | Templates | Constraints |
|---|---|---|---|
| Error Free | 25133 | 8 | 38 |
| Ring Break | 24478 | 8 | 37 |
| Excessive Attenuation | 24564 | 8 | 37 |
| Multi Master | 25231 | 8 | 37 |
| All Slave | 25117 | 7 | 29 |
| Combination | 24756 | 8 | 38 |

# Experimental Results – Ring Break Platform Variant Specification




- Top level of the variant specification for scenario „Ring Break“

# Experimental Results – Ring Break Platform Variant Specification




- Each SlaveTemplate contains Or-Template to inject Ring Break Channels
- Only one Ring Break can be diagnosed by RBD algorithm
  - Ensured by P-OCL If constraint

# Experimental Results –
# Ring Break Platform Variant Specification

```
«vp_constraint»
Constraint5
{if self.orTemp.ChannelRB.active() then
slaveTemp1.orTemp.allInstances-> forAll(e | !e.ChannelRB.active()) and
slaveTemp2.orTemp.allInstances-> forAll(e | !e.ChannelRB.active())
else ...
  endif
endif}
```

- Each SlaveTemplate contains Or-Template to inject Ring Break Channels
- Only one Ring Break can be diagnosed by RBD algorithm
  - Ensured by P-OCL If constraint

# Experimental Results – Central Component Verification Scenarios

- Eight evaluation scenarios turned out to be suggestive.

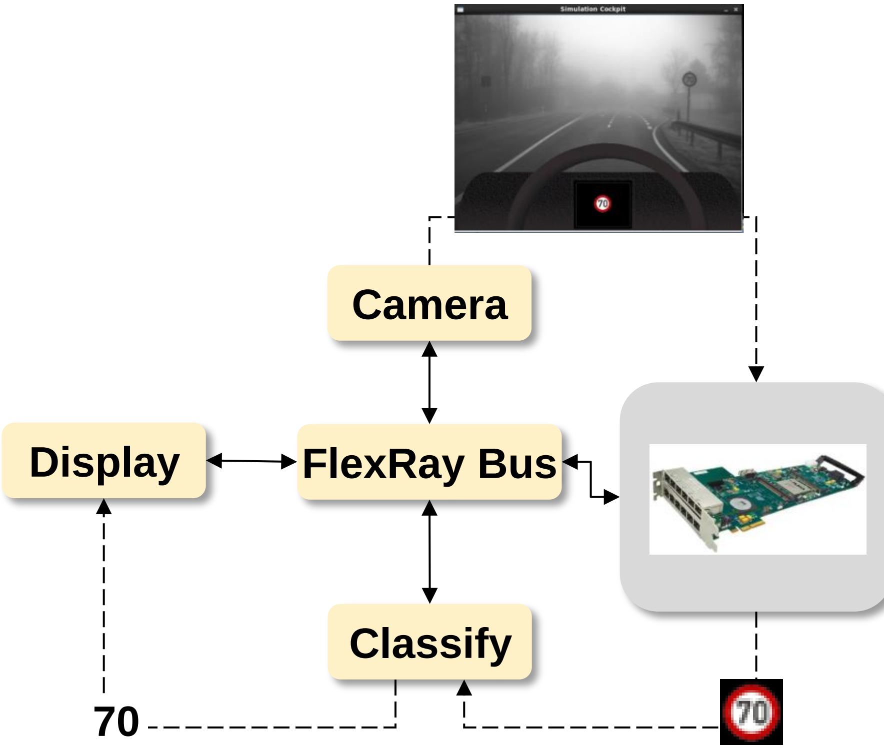
| Scenario | Variants | Templates | Constraints |
|---|---|---|---|
| No SSO | 15334 | 2 | 1 |
| One timing slave SSO | 119271 | 2 | 3 |
| Timing master SSO | 1625 | 2 | 3 |
| More than one timing slave reports SSO | 746616 | 3 | 4 |
| No CU | 20243 | 2 | 1 |
| One timing slave reports CU | 4744 | 2 | 3 |
| Timing master CU | 1465 | 2 | 3 |
| More than one timing slave reports CU | 56294 | 3 | 4 |

# FlexRay – Traffic Sign Recognition (TSR) Scenario

- Heterogeneous system, virtual prototypes and target code

- Virtual prototype modules

- Target code implementation for Tilera board



# Experimental Results

- Evaluated against frame rate and recognized traffic signs
- Exploration of the Camera Module regarding:
  - Display resolution
  - Greyscale- or colored-camera
  - Scale factor
- Exploration of different hardware parallelization options:
  - Number of used cores (up to 54 cores)
  - Range definition for circle detection
- 71150 valid Variants are generated

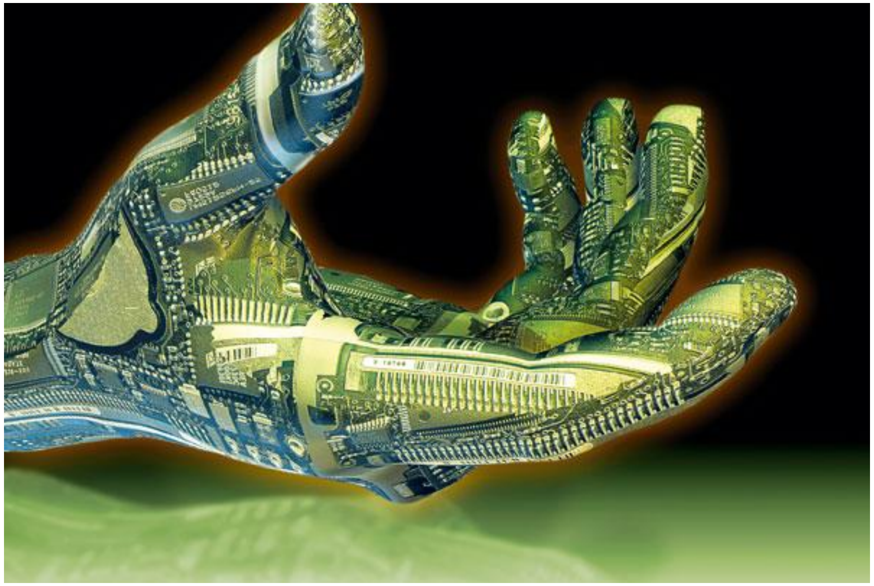
# Conclusion

- Constraint- and Model-based variants specification approach
  - High structural flexibility
  - Reuse of already modeled templates and variants
  - Enables to handle huge variants spaces
  - Precise, plausible and comprehensive specification of valid variants
- Automatically generation and simulation of platform variants

# Conclusion

- Constraint- and Model-based variants specification approach
  - High structural flexibility
  - Reuse of already modeled templates and variants
  - Enables to handle huge variants spaces
  - Precise, plausible and comprehensive specification of valid variants
- Automatically generation and simulation of platform variants
- Benefits are:
  - Reduction of manual effort in verification, exploration and test
  - Highly automatic generation of virtual prototype variants
  - Reusability of the variants specifications

# Thank you for your attention!




Contact person

Andreas Burger

FZI Research Center
for Information Technologies
Haid-und-Neu-Str. 10-14
76131 Karlsruhe, Germany
aburger@fzi.de
www.fzi.de/ispe

# References


- [1] A. Braun, O. Bringmann, D. Lettnin, and W. Rosenstiel, Simulation-based verification of the most netInterface specification revision 3.0," in Design, Automation Test in Europe Conference Exhibition (DATE), 2010,March 2010, pp. 538-543