

Tutorial at 2021 SCS Annual Modeling And Simulation Conf. (ANNSIM'21)  
July 19-21, 2021, Virtual Event, (org. by George Mason Univ. Fairfax/VA, USA)

# Variability Modeling and Simulation Using Multiple Simulators

Faculty of Engineering / Research Group CEA

Thorsten Pawletta  
Hendrik Folkerts  
Christina Deatcu

E-Mail:  
{thorsten.pawletta, christina.deatcu}@hs-wismar.de  
hendrik.folkerts@cea-wismar.de

Web:  
[www.hs-wismar.de](http://www.hs-wismar.de) / [www.cea-wismar.de](http://www.cea-wismar.de)





## Outline

1. Introduction
2. The case study
3. Basics of SES/MB based modeling
4. Practical modeling: implementation of an SES
5. Model selection and model generation
6. Organization of a simulator-independent MB
7. Full automation of simulation experiments
8. Conclusion



# Outline

- 1. Introduction** (T. Pawletta)
2. The case study
3. Basics of SES/MB based modeling
4. Practical modeling: implementation of an SES
5. Model selection and model generation
6. Organization of a simulator-independent MB
7. Full automation of simulation experiments
8. Conclusion



# Introduction



# Introduction

- Systems, such as IOT, CPS or Industry 4.0, have a **high degree of variability**
  - Modern cars have more than 50 electronic control units (ECU), each may be instantiated in at least 10.000 different ways.<sup>(1)</sup>
    - **Millions of configurations (variants)**

<sup>1)</sup> Oster S. (2011) „Feature Model-based Software Product Line Testing“. PhD thesis, TU Darmstadt.



# Introduction

- Systems, such as IOT, CPS or Industry 4.0, have a **high degree of variability**
  - Modern cars have more than 50 electronic control units (ECU), each may be instantiated in at least 10.000 different ways.<sup>(1)</sup>
    - **Millions of configurations (variants)**
- **Variability** is defined in Software Engineering as:  
the ability of a system to be configurable, extendable or adaptable depending on its purpose and objective. → **SPL engineering** (SPLE)

<sup>1)</sup> Oster S. (2011) „Feature Model-based Software Product Line Testing“. PhD thesis, TU Darmstadt.



# Introduction

- Systems, such as IOT, CPS or Industry 4.0, have a **high degree of variability**
  - Modern cars have more than 50 electronic control units (ECU), each may be instantiated in at least 10.000 different ways.<sup>(1)</sup>
    - **Millions of configurations (variants)**
- **Variability** is defined in Software Engineering as:  
the ability of a system to be configurable, extendable or adaptable depending on its purpose and objective. → **SPL engineering** (SPLE)
- **M&S**: Management of model families (set of models with common features)

<sup>1)</sup> Oster S. (2011) „Feature Model-based Software Product Line Testing“. PhD thesis, TU Darmstadt.



# Introduction



- Systems, such as IOT, CPS or Industry 4.0, have a **high degree of variability**
  - Modern cars have more than 50 electronic control units (ECU), each may be instantiated in at least 10.000 different ways.<sup>(1)</sup>
    - **Millions of configurations (variants)**
- **Variability** is defined in Software Engineering as:  
the ability of a system to be configurable, extendable or adaptable depending on its purpose and objective. → **SPL engineering** (SPLE)
- **M&S:** Management of model families (set of models with common features)
  - Today methods often transferred 1:1 from SPLE to M&S domain 🙄

<sup>1)</sup> Oster S. (2011) „Feature Model-based Software Product Line Testing“. PhD thesis, TU Darmstadt.





# Introduction

- Systems, such as IOT, CPS or Industry 4.0, have a **high degree of variability**
  - Modern cars have more than 50 electronic control units (ECU), each may be instantiated in at least 10.000 different ways.<sup>(1)</sup>
    - **Millions of configurations (variants)**
- **Variability** is defined in Software Engineering as:  
the ability of a system to be configurable, extendable or adaptable depending on its purpose and objective. → **SPL engineering** (SPLE)
- **M&S**: Management of model families (set of models with common features)
  - Today methods often transferred 1:1 from SPLE to M&S domain 
  - More suitable are methods from **model based systems engineering** (→ **SES/MB approach**) 

<sup>1)</sup> Oster S. (2011) „Feature Model-based Software Product Line Testing“. PhD thesis, TU Darmstadt.



## Introduction (2)



## Introduction (2)

- **Why should SPLE not be transferred 1:1 to the M&S domain?**



## Introduction (2)

- **Why should SPLE not be transferred 1:1 to the M&S domain?**
  - **Specific paradigms apply to M&S:**



## Introduction (2)

- **Why should SPLE not be transferred 1:1 to the M&S domain?**
  - **Specific paradigms apply to M&S:**
    - Separation of model structures & model dynamics  
(violation in 150% modeling approach)



## Introduction (2)

- **Why should SPLE not be transferred 1:1 to the M&S domain?**
  - **Specific paradigms apply to M&S:**
    - Separation of model structures & model dynamics (violation in 150% modeling approach)
    - Separation of model and experiment (use of a model in different context)



## Introduction (2)

- **Why should SPLE not be transferred 1:1 to the M&S domain?**
  - **Specific paradigms apply to M&S:**
    - Separation of model structures & model dynamics (violation in 150% modeling approach)
    - Separation of model and experiment (use of a model in different context)
    - Modular-hierarchical modeling of configurations (variants of structure & parameter settings)



## Introduction (2)

- **Why should SPLE not be transferred 1:1 to the M&S domain?**
  - **Specific paradigms apply to M&S:**
    - Separation of model structures & model dynamics (violation in 150% modeling approach)
    - Separation of model and experiment (use of a model in different context)
    - Modular-hierarchical modeling of configurations (variants of structure & parameter settings)
- **Why should a model be used with different simulators?**





## Introduction (2)

- **Why should SPLE not be transferred 1:1 to the M&S domain?**
  - **Specific paradigms apply to M&S:**
    - Separation of model structures & model dynamics (violation in 150% modeling approach)
    - Separation of model and experiment (use of a model in different context)
    - Modular-hierarchical modeling of configurations (variants of structure & parameter settings)
- **Why should a model be used with different simulators?**
  - domain dependency, (?) simulator correctness




## Introduction (2)

- **Why should SPLE not be transferred 1:1 to the M&S domain?**
  - **Specific paradigms apply to M&S:**
    - Separation of model structures & model dynamics (violation in 150% modeling approach)
    - Separation of model and experiment (use of a model in different context)
    - Modular-hierarchical modeling of configurations (variants of structure & parameter settings)
- **Why should a model be used with different simulators?**
  - domain dependency, (?) simulator correctness
- **Why should studies of variants be automated?**



## Introduction (2)

- **Why should SPLE not be transferred 1:1 to the M&S domain?**
  - **Specific paradigms apply to M&S:**
    - Separation of model structures & model dynamics (violation in 150% modeling approach)
    - Separation of model and experiment (use of a model in different context)
    - Modular-hierarchical modeling of configurations (variants of structure & parameter settings)
- **Why should a model be used with different simulators?**
  - domain dependency, (?) simulator correctness
- **Why should studies of variants be automated?** 
  - Simulation studies are time-consuming (selection of a configuration, model generation & simulation, analysis of simulation results)



## Outline

1. Introduction
- 2. The case study** (T. Pawletta)
3. Basics of SES/MB based modeling
4. Practical modeling: implementation of an SES
5. Model selection and model generation
6. Organization of a simulator-independent MB
7. Full automation of simulation experiments
8. Conclusion

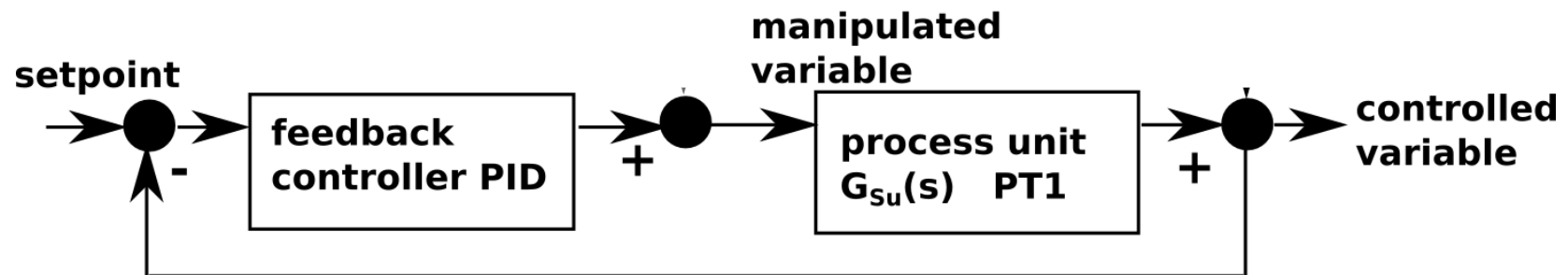


## Case Study



## Case Study

- Feedback control system

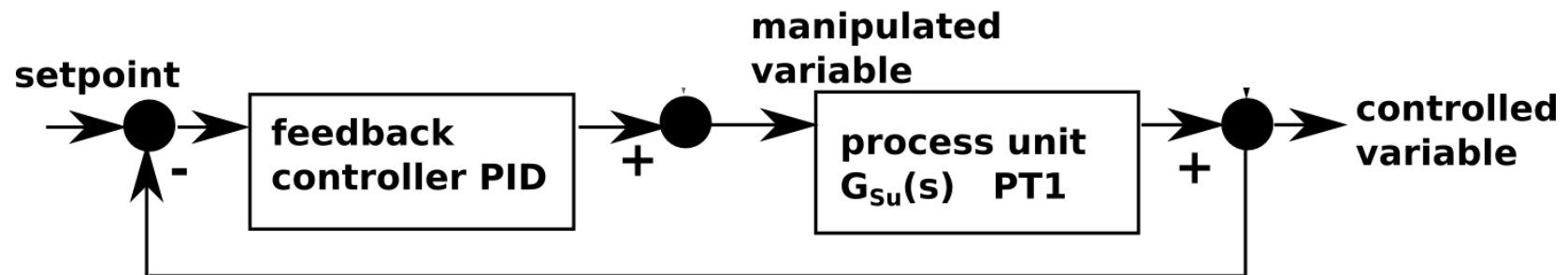




## Case Study

- Feedback control system
- Described by transfer functions

$$G_{Su}(s) = \frac{1}{20 \cdot s + 1}$$



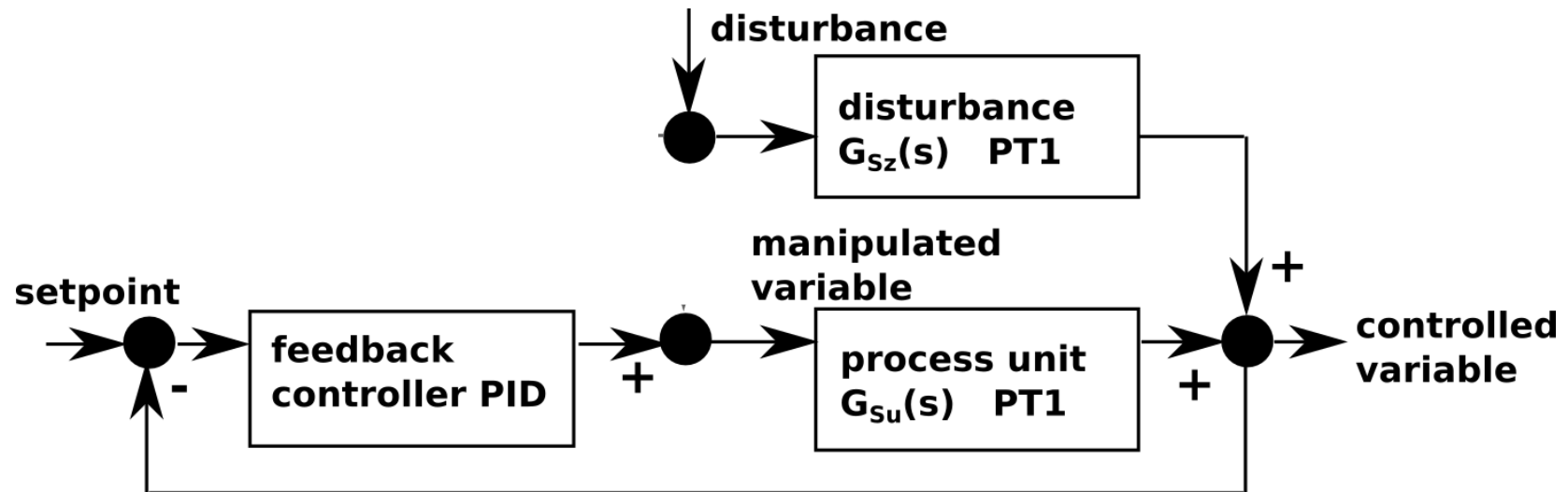


## Case Study

- Feedback control system
- Described by transfer functions
- Influenced by disturbances

$$G_{Su}(s) = \frac{1}{20 \cdot s + 1}$$

$$G_{Sz}(s) = \frac{1}{10 \cdot s + 1}$$







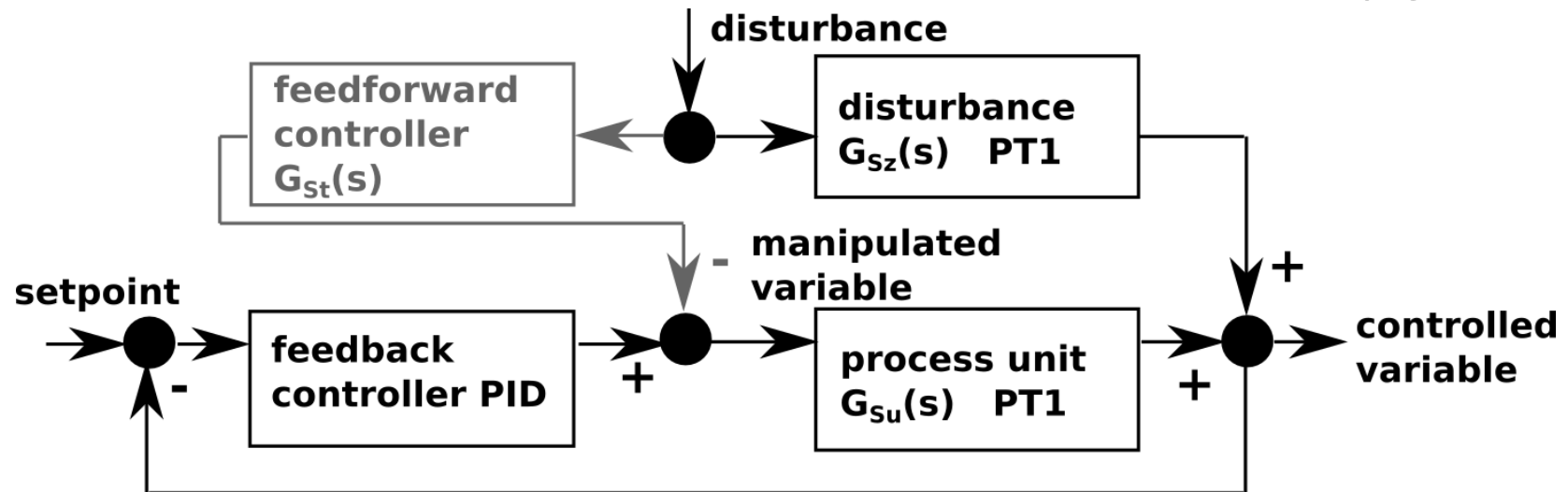
## Case Study

- Feedback control system
- Described by transfer functions
- Influenced by disturbances
- Measurable disturbances
  - Compensated with feedforward control

$$G_{Su}(s) = \frac{1}{20 \cdot s + 1}$$

$$G_{Sz}(s) = \frac{1}{10 \cdot s + 1}$$

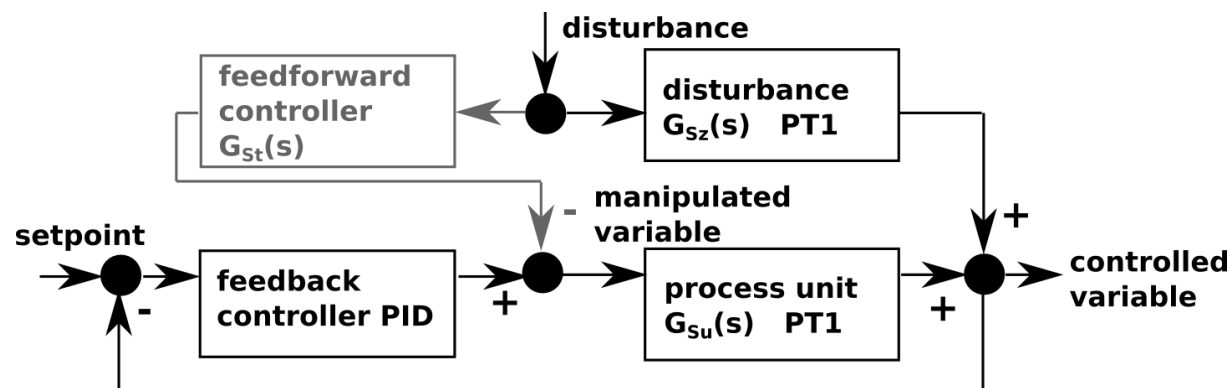
$$G_{St}(s) = \frac{20 \cdot s + 1}{10 \cdot s + 1}$$





## Case Study (2)

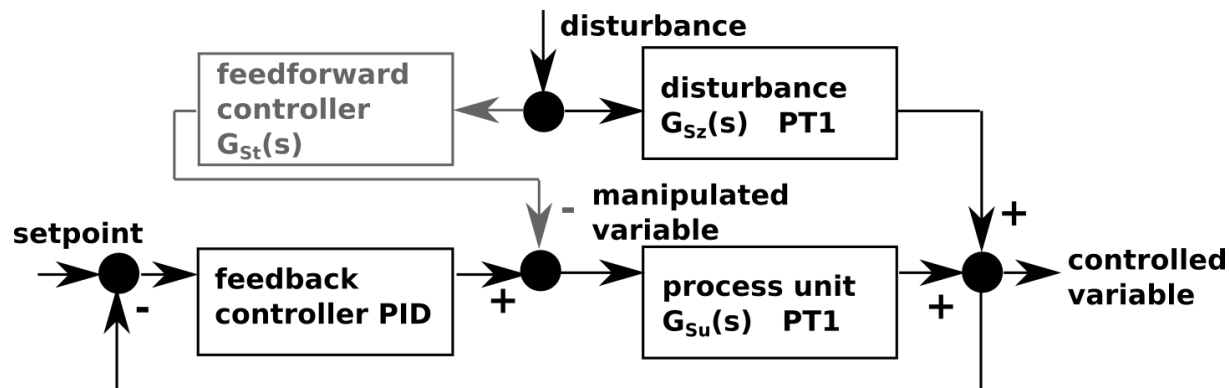
- **Two system structure variants**
  - Without feedforward control: `feedforward=0`
  - With feedforward control: `feedforward=1`





## Case Study (2)

- **Two system structure variants**
  - Without feedforward control: `feedforward=0`
  - With feedforward control: `feedforward=1`
- For every structure variant
  - **Different parameter configurations of PID controller**  
(we consider two)

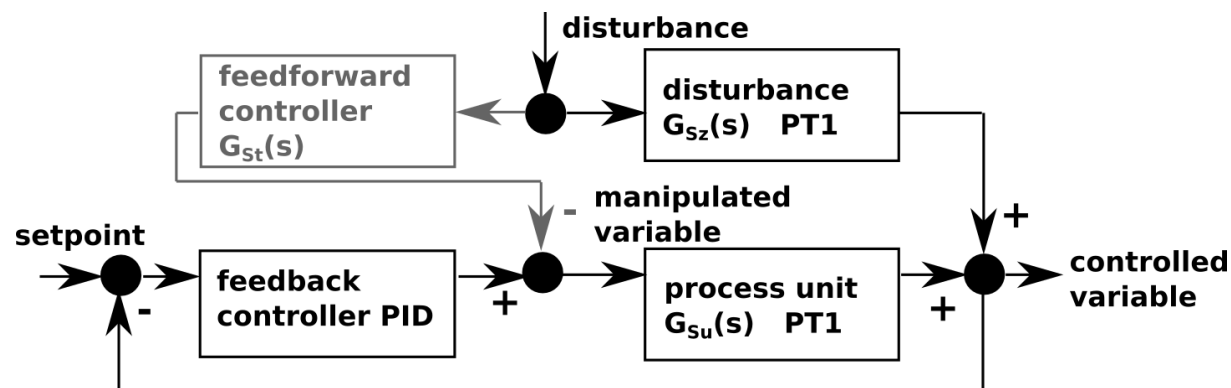




## Case Study (2)

**Design objective:  
Find best control configuration.**

- **Two system structure variants**
  - Without feedforward control: `feedforward=0`
  - With feedforward control: `feedforward=1`
- For every structure variant
  - **Different parameter configurations of PID controller**  
(we consider two)



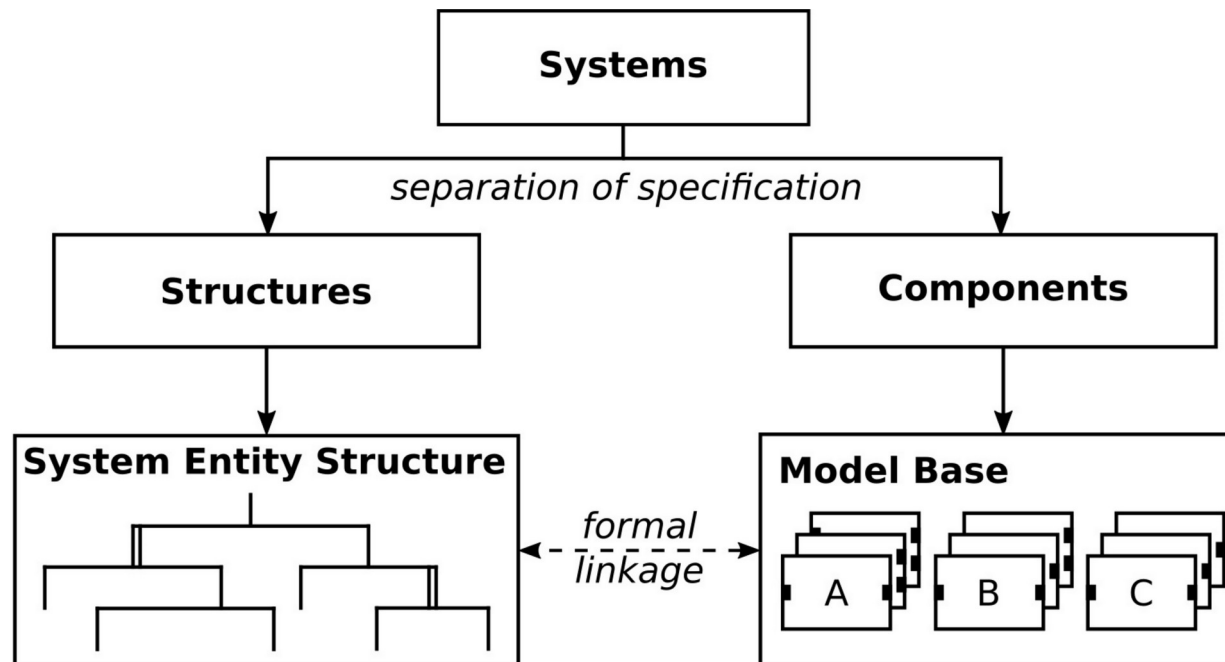


## Outline

1. Introduction
2. The case study
- 3. Basics of SES/MB based modeling** (T. Pawletta)
4. Practical modeling: implementation of an SES
5. Model selection and model generation
6. Organization of a simulator-independent MB
7. Full automation of simulation experiments
8. Conclusion

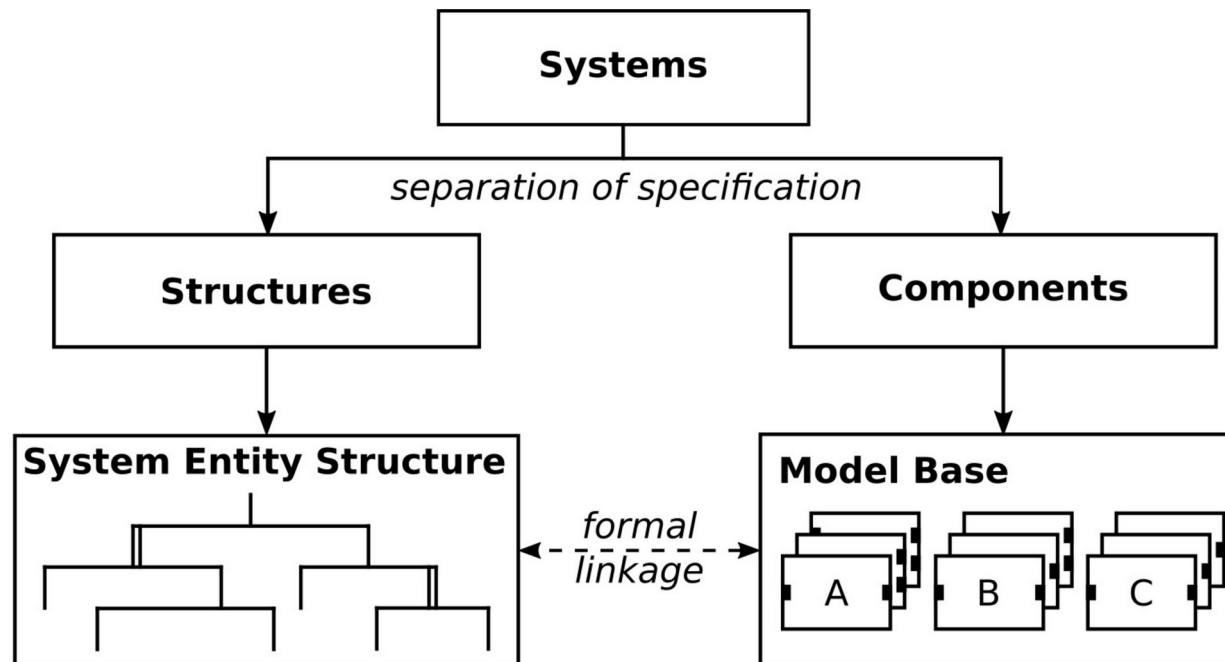


## SES/MB Modeling Approach





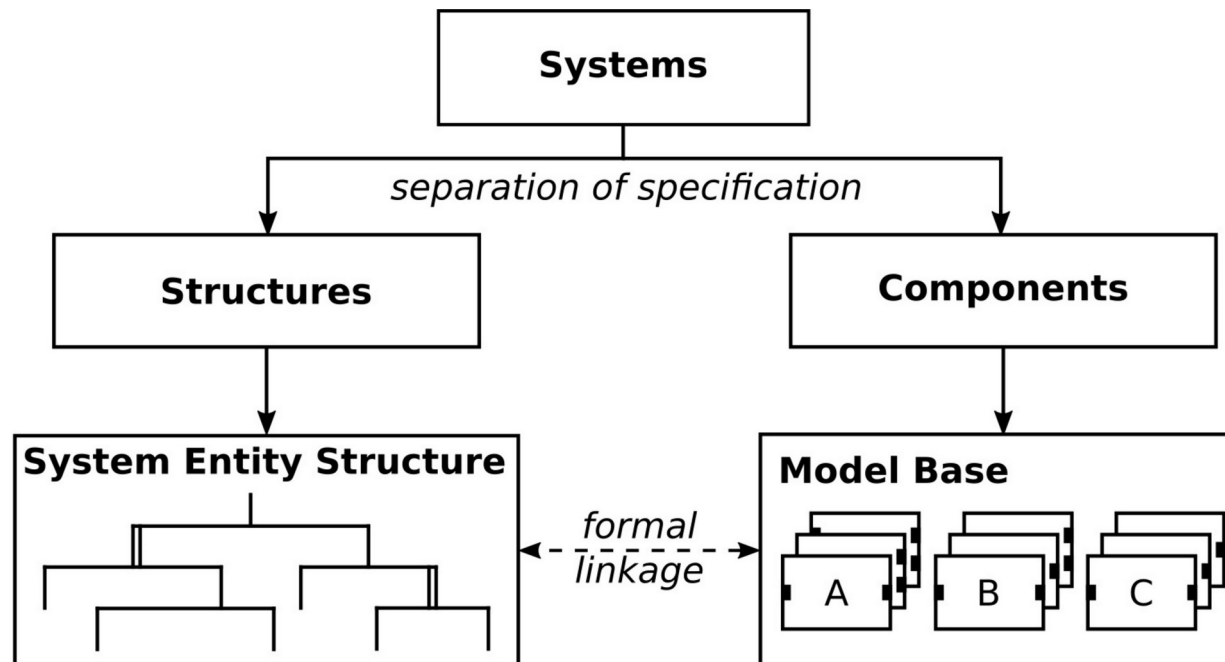
## SES/MB Modeling Approach



- **SES** describes permissible structure & parameter variants (simulator-independent)



## SES/MB Modeling Approach



- **SES** describes permissible structure & parameter variants (simulator-independent)
- **MB** defines basic dynamic models (usually simulator dependent)





## Basics of the System Entity Structure (SES)



## Basics of the System Entity Structure (SES)

- SES introduced by B.P. Zeigler and J. Rozenblit



## Basics of the System Entity Structure (SES)

- SES introduced by B.P. Zeigler and J. Rozenblit
- Amongst others extended by research group CEA (Wismar)



## Basics of the System Entity Structure (SES)

- SES introduced by B.P. Zeigler and J. Rozenblit
- Amongst others extended by research group CEA (Wismar)
- **SES is a tree structure**



## Basics of the System Entity Structure (SES)

- SES introduced by B.P. Zeigler and J. Rozenblit
- Amongst others extended by research group CEA (Wismar)
- **SES is a tree structure**
  - Well defined by axioms



## Basics of the System Entity Structure (SES)

- SES introduced by B.P. Zeigler and J. Rozenblit
- Amongst others extended by research group CEA (Wismar)
- **SES is a tree structure**
  - Well defined by axioms
  - Two types of nodes
    - Entity nodes
    - Descriptive nodes



## Basics of the System Entity Structure (SES)

- SES introduced by B.P. Zeigler and J. Rozenblit
- Amongst others extended by research group CEA (Wismar)
- **SES is a tree structure**
  - Well defined by axioms
  - Two types of nodes
    - Entity nodes
    - Descriptive nodes

**Entity nodes**  
real or imaginary  
objects

**Descriptive nodes**  
Aspect  
(Multi-aspect)  
Specialization



## Basics of the System Entity Structure (SES)

- SES introduced by B.P. Zeigler and J. Rozenblit
- Amongst others extended by research group CEA (Wismar)
- **SES is a tree structure**
  - Well defined by axioms
  - Two types of nodes
    - Entity nodes
    - Descriptive nodes
  - Three types of edges (relations between nodes)

**Entity nodes**  
real or imaginary  
objects

**Descriptive nodes**  
Aspect  
(Multi-aspect)  
Specialization





## Basics of the System Entity Structure (SES)

- SES introduced by B.P. Zeigler and J. Rozenblit
- Amongst others extended by research group CEA (Wismar)
- **SES is a tree structure**
  - Well defined by axioms
  - Two types of nodes
    - Entity nodes
    - Descriptive nodes
  - Three types of edges (relations between nodes)
  - Node/Edge specific attributes

**Entity nodes**  
real or imaginary  
objects

**Descriptive nodes**  
Aspect  
(Multi-aspect)  
Specialization



## Basics of the System Entity Structure (SES)

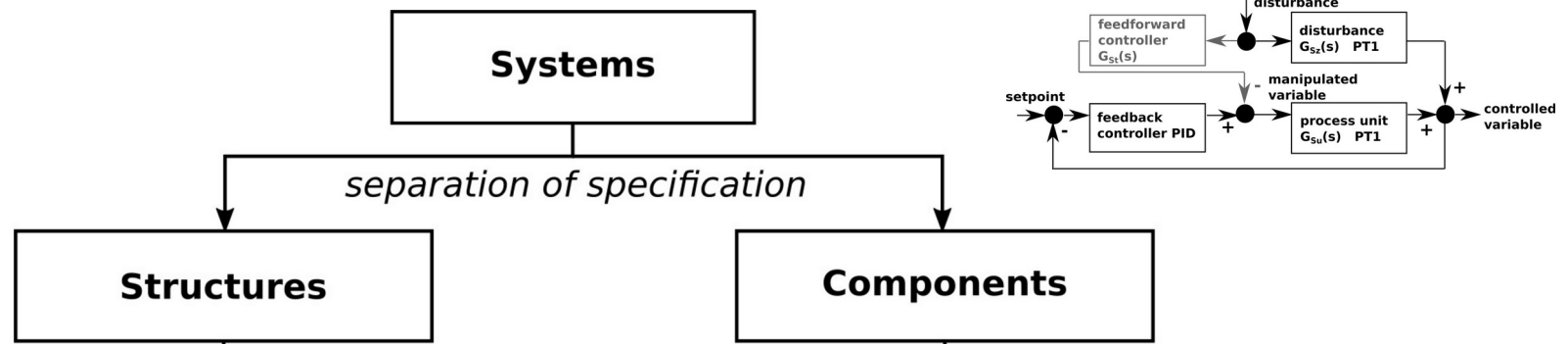
- SES introduced by B.P. Zeigler and J. Rozenblit
- Amongst others extended by research group CEA (Wismar)
- **SES is a tree structure**
  - Well defined by axioms
  - Two types of nodes
    - Entity nodes
    - Descriptive nodes
  - Three types of edges (relations between nodes)
  - Node/Edge specific attributes
  - Global variables, functions, constraints, ...

**Entity nodes**  
real or imaginary  
objects

**Descriptive nodes**  
Aspect  
(Multi-aspect)  
Specialization

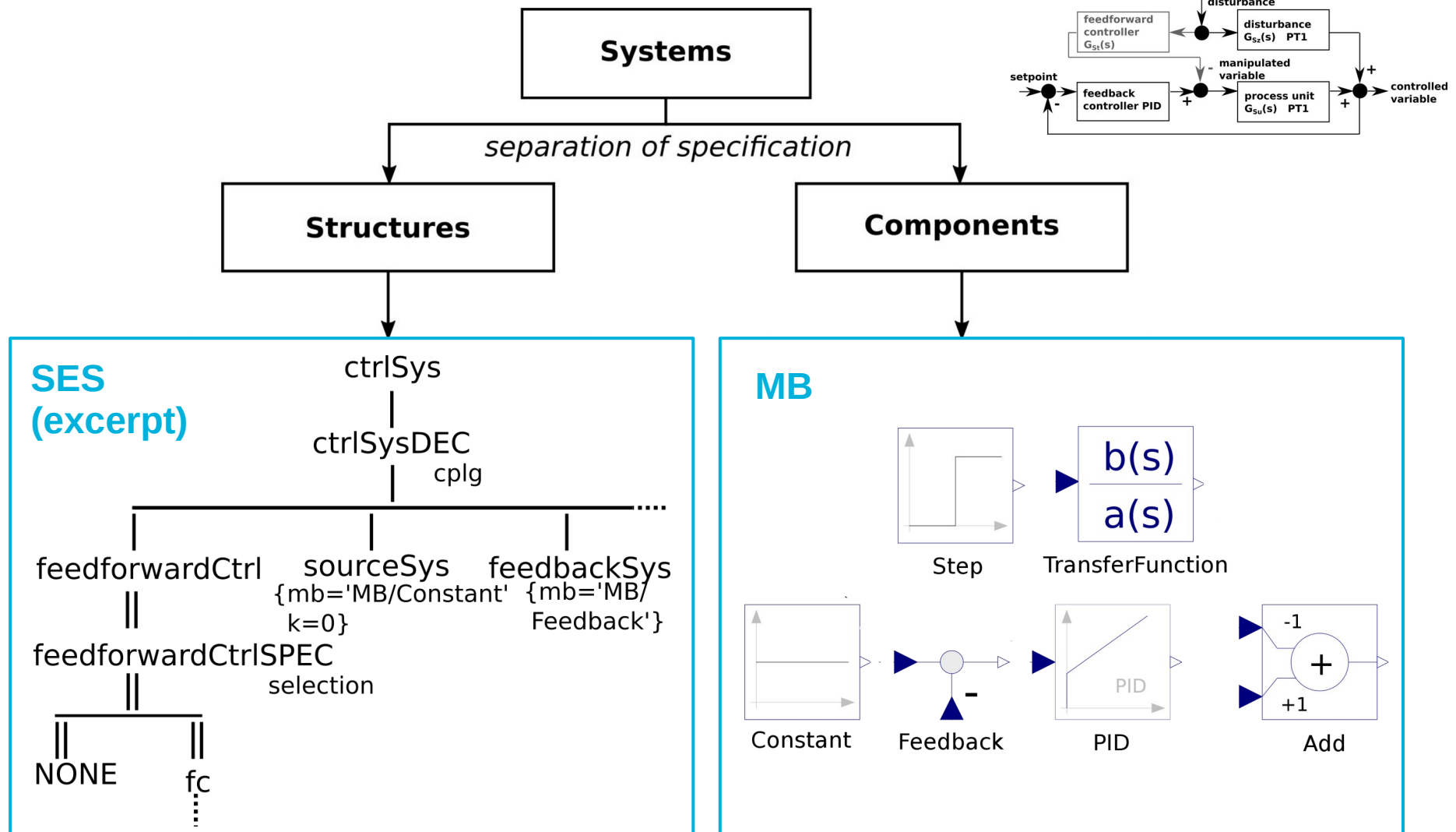


## SES/MB-based Modeling of the Case Study



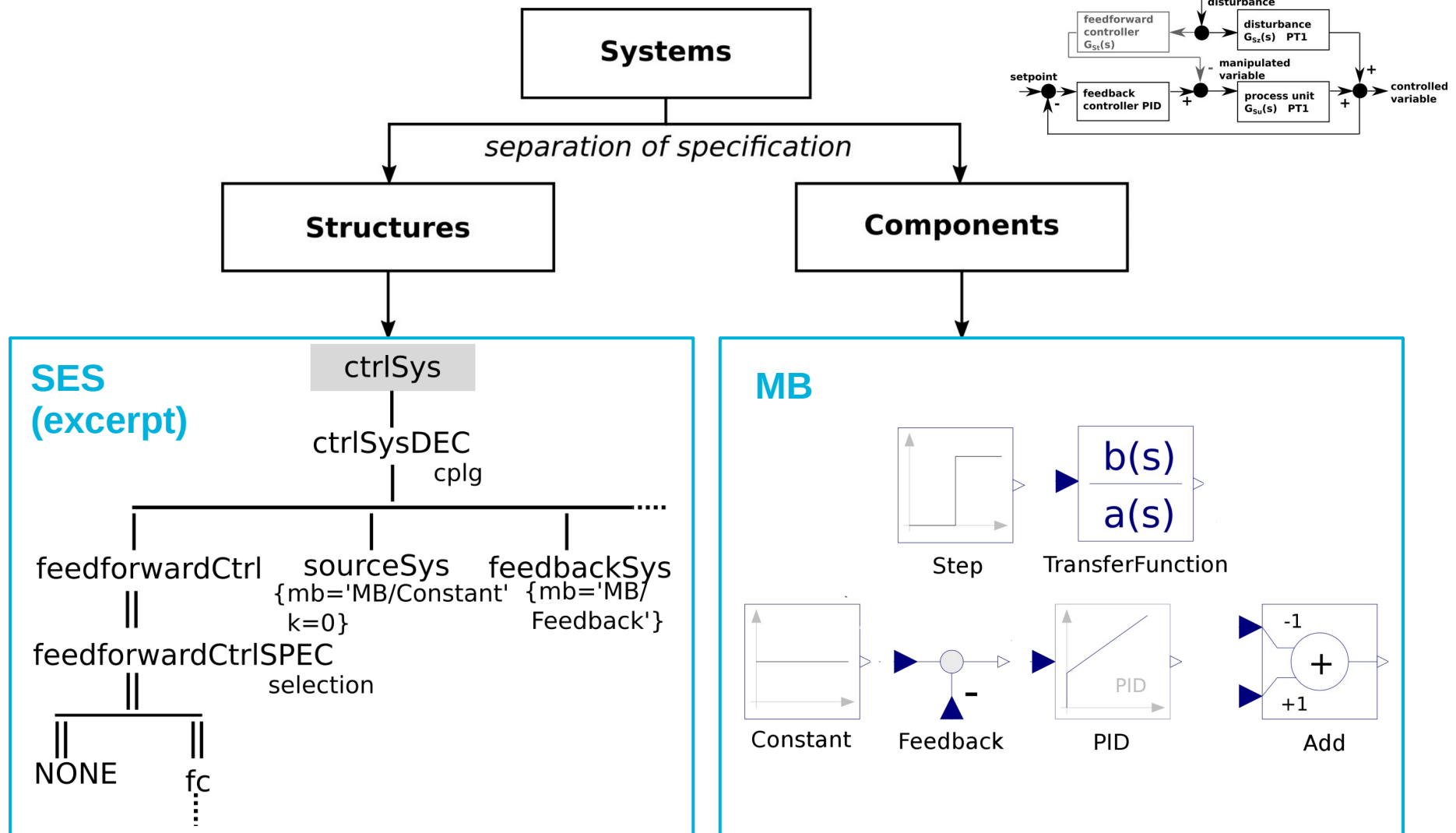


# SES/MB-based Modeling of the Case Study



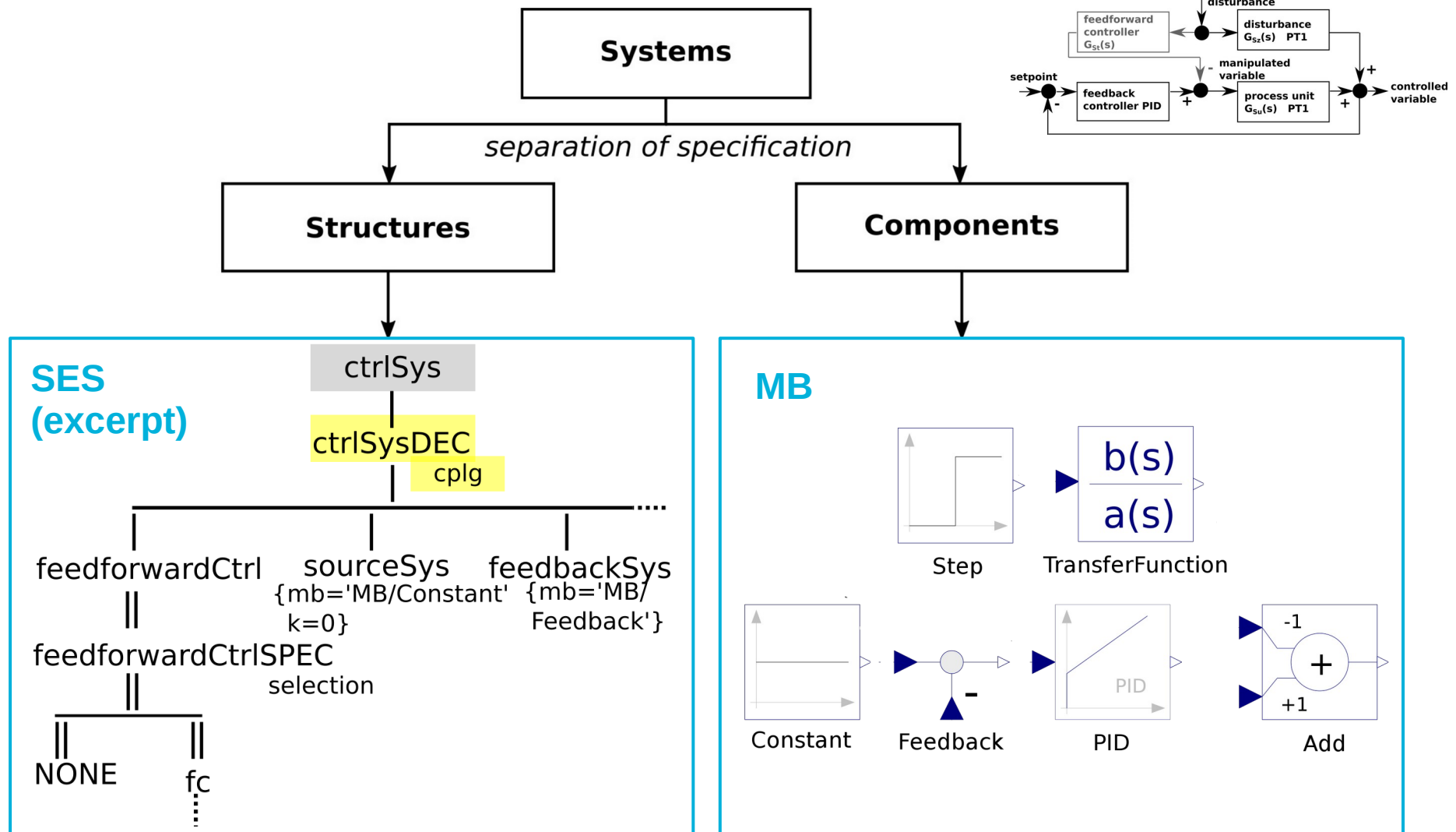


# SES/MB-based Modeling of the Case Study



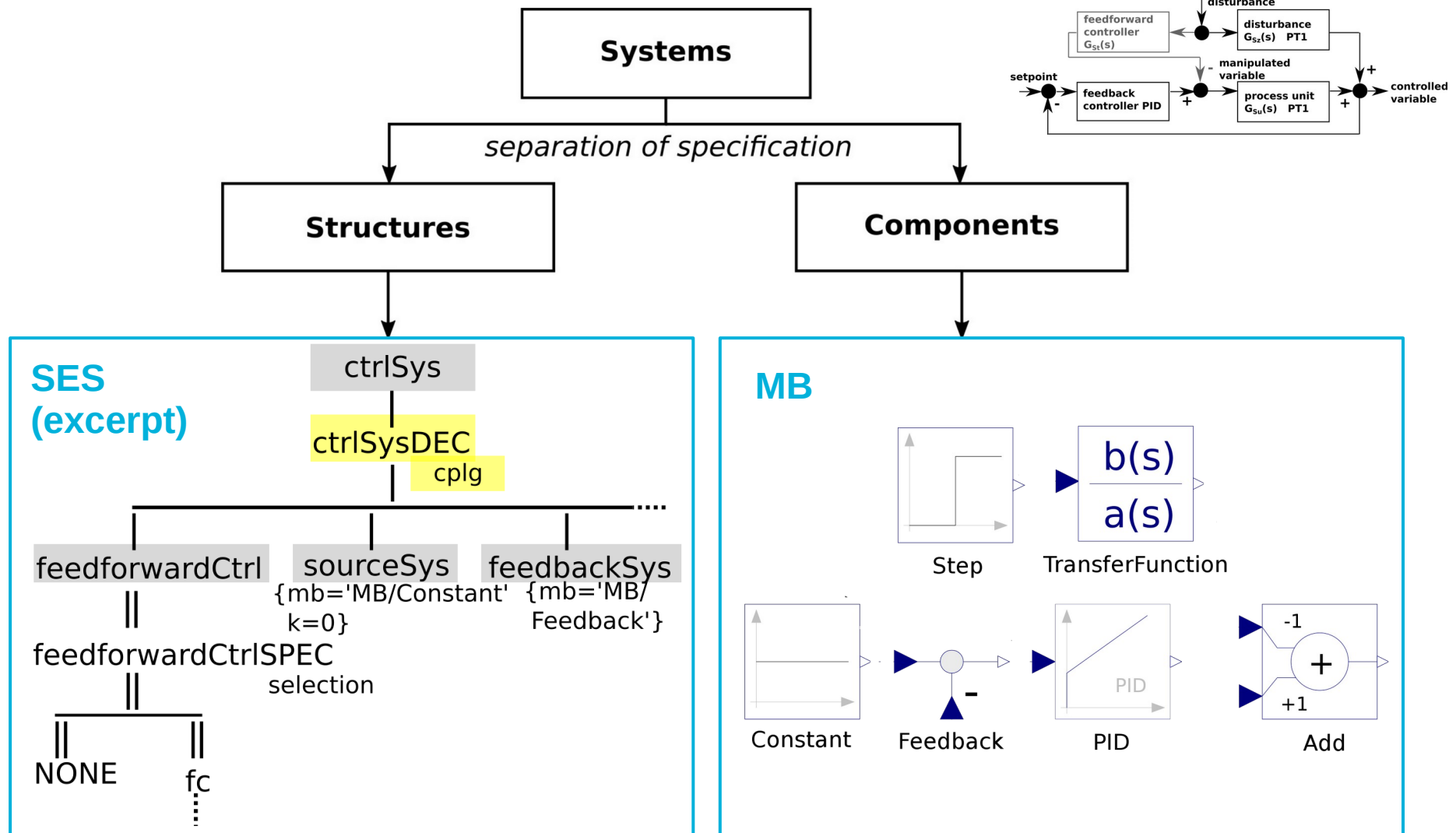


# SES/MB-based Modeling of the Case Study



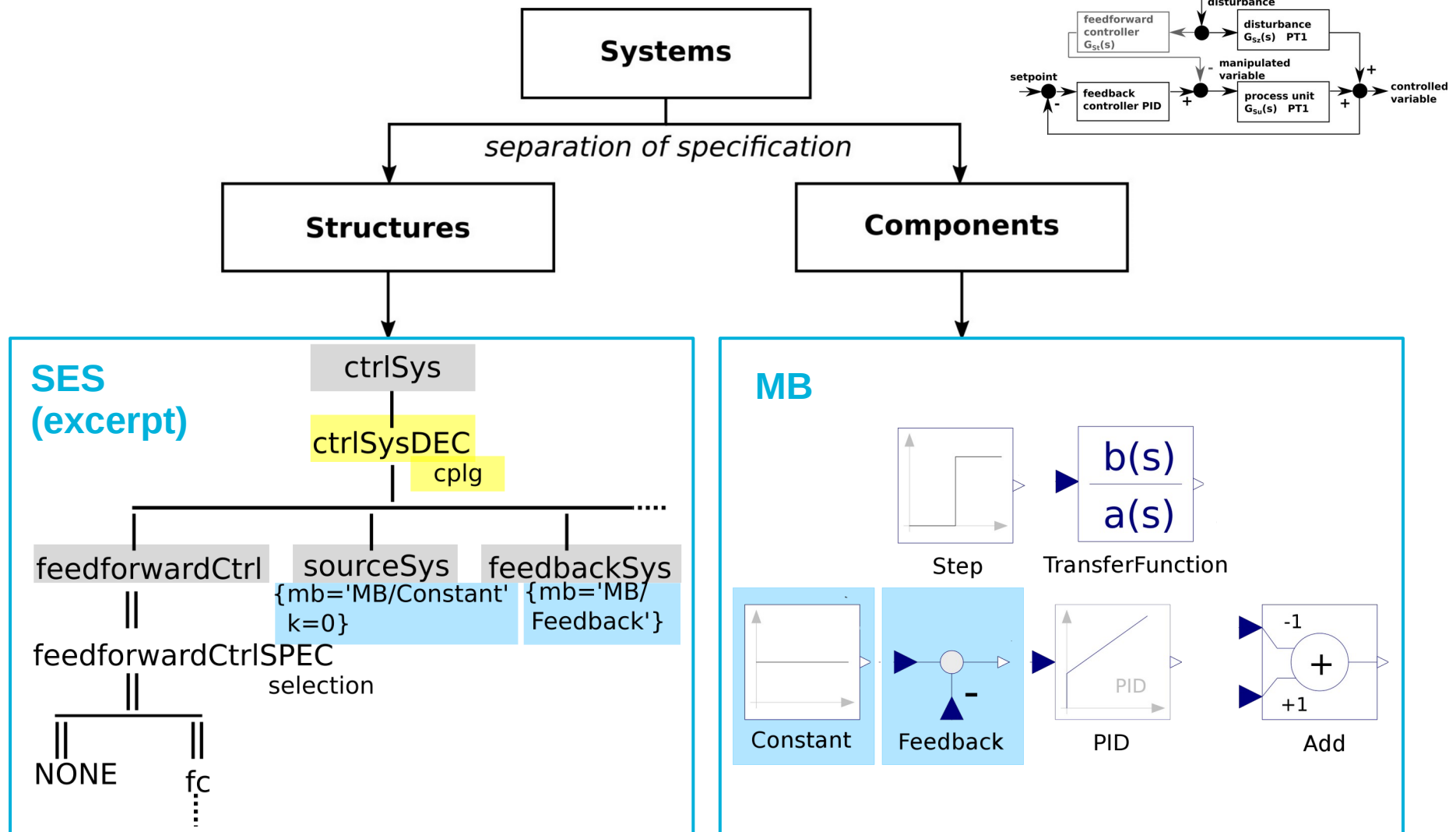


# SES/MB-based Modeling of the Case Study





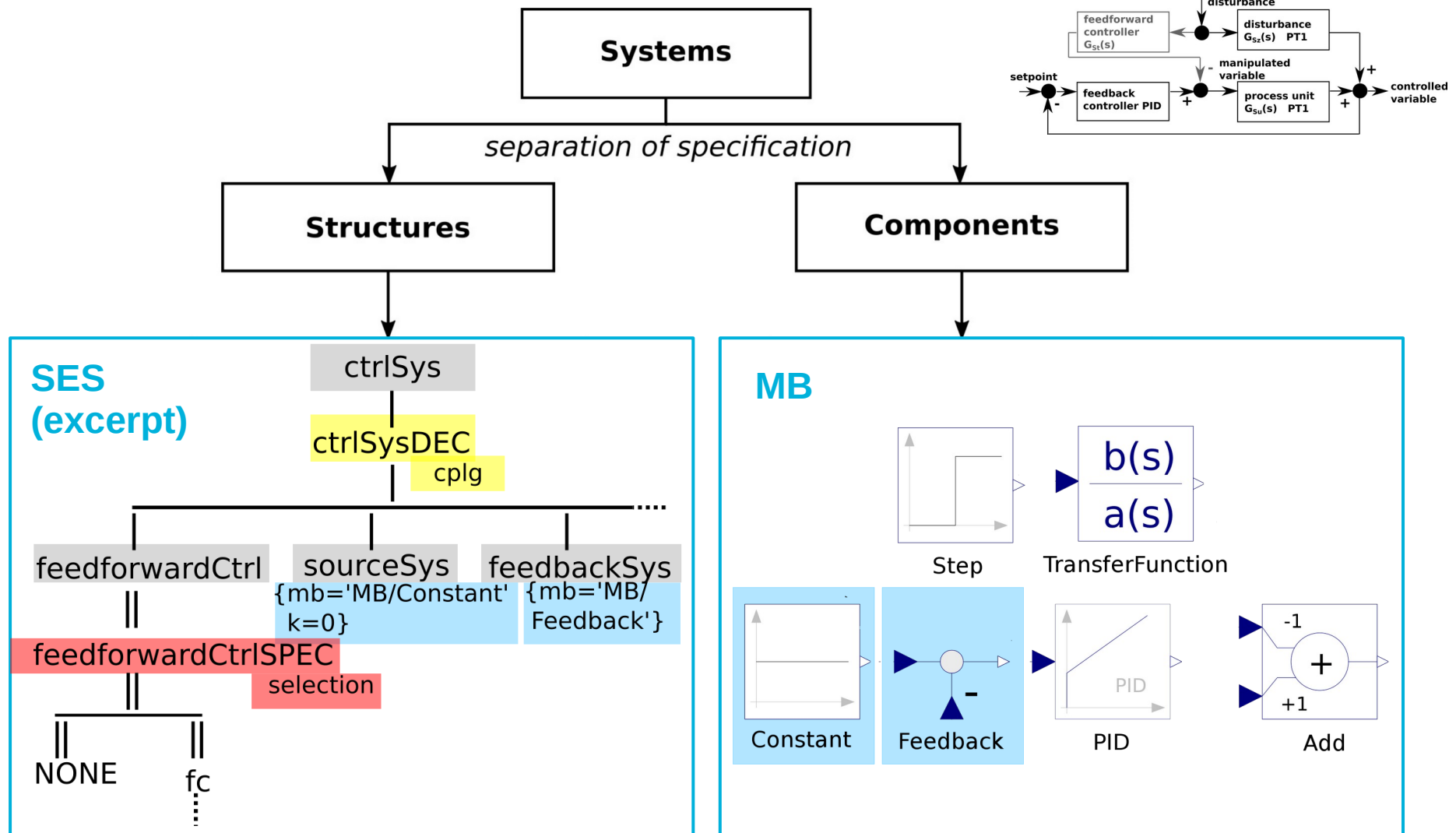
# SES/MB-based Modeling of the Case Study





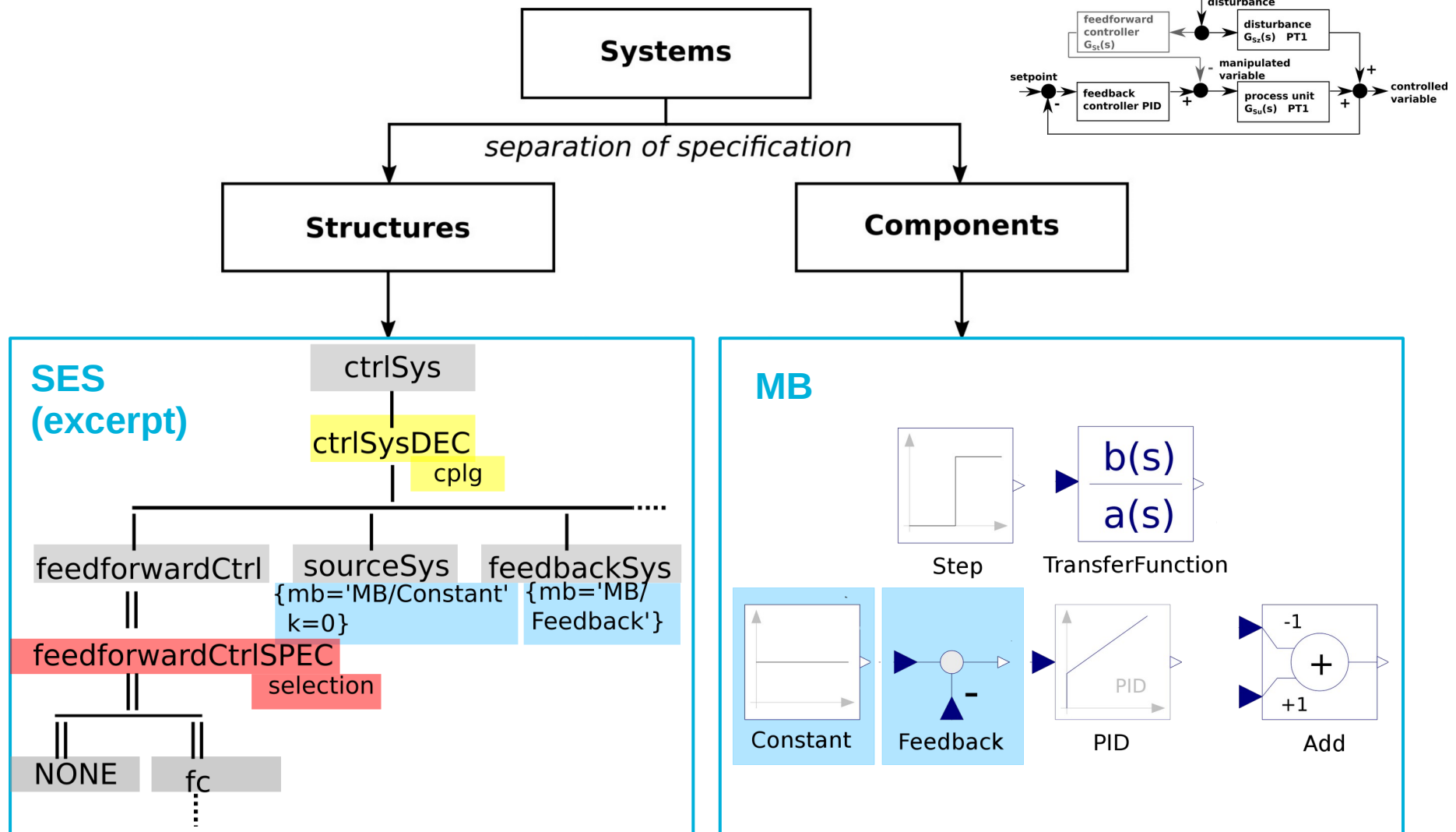


# SES/MB-based Modeling of the Case Study





# SES/MB-based Modeling of the Case Study

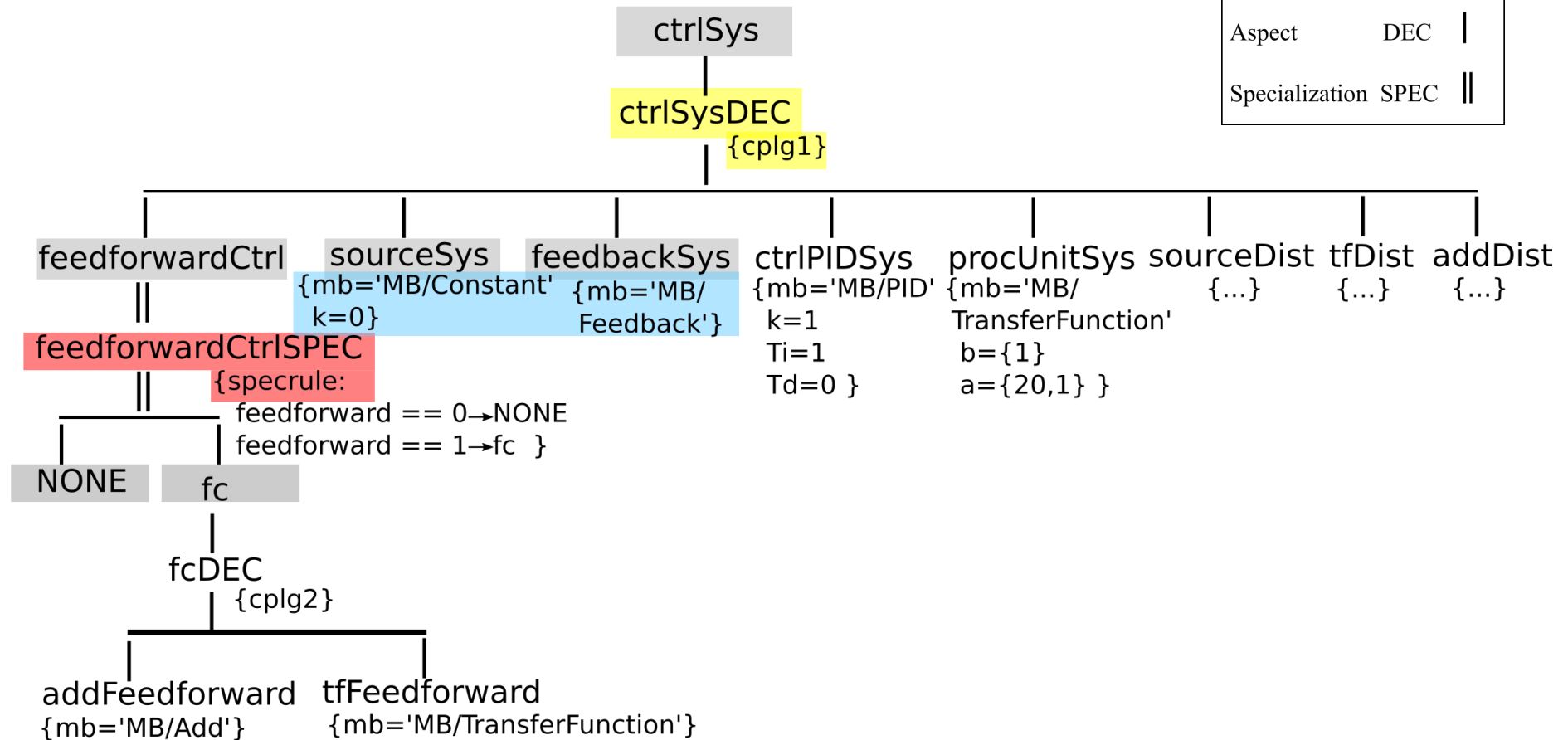




## More Detailed Extract of the SES

**SES** SESVAR={feedforward}  
SemanticCondition={feedforward in [0,1]}

Type	Key	Suffix	Edge
Aspect		DEC	
Specialization		SPEC	

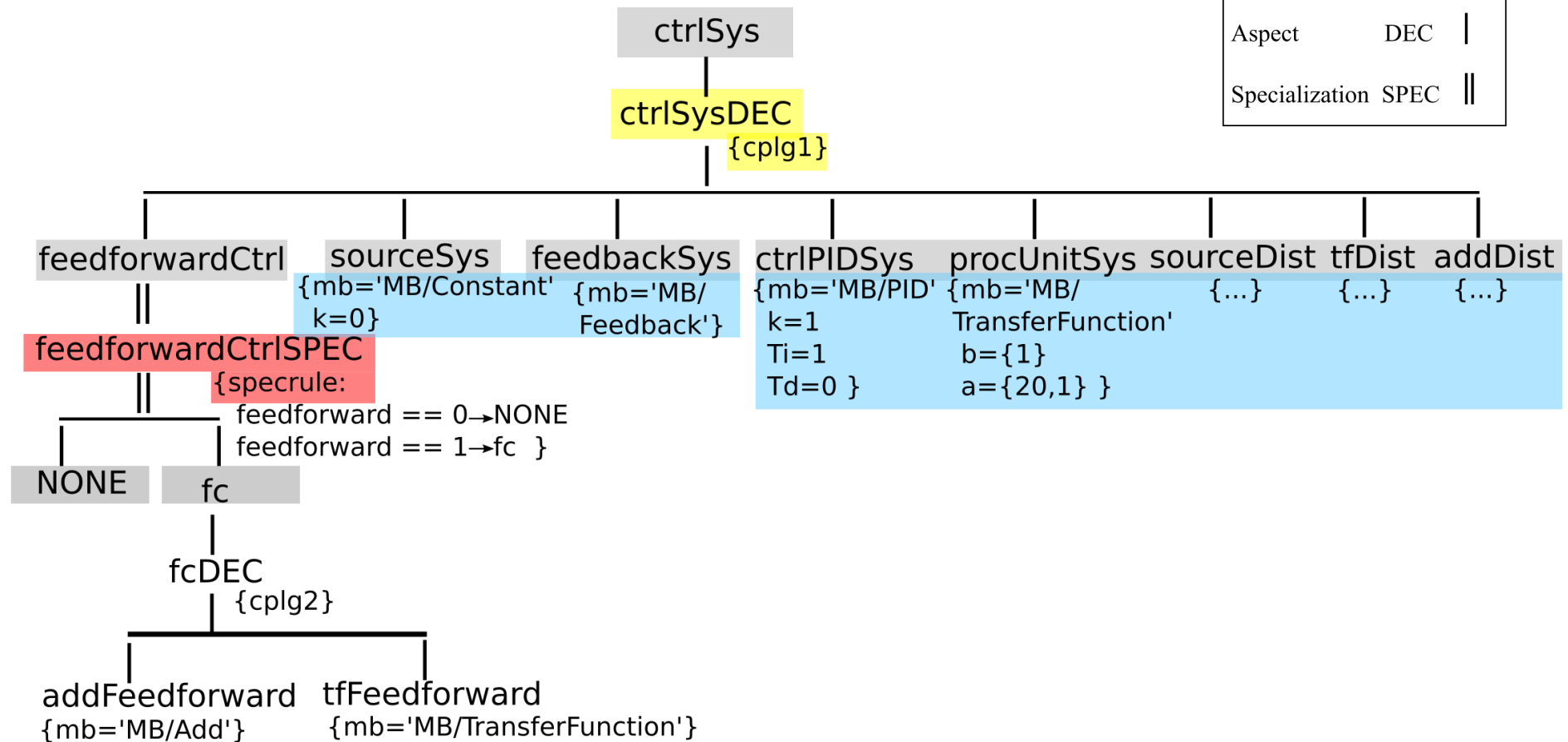




## More Detailed Extract of the SES

**SES** SESVAR={feedforward}  
SemanticCondition={feedforward in [0,1]}

Type	Key	Suffix	Edge
Aspect		DEC	
Specialization		SPEC	

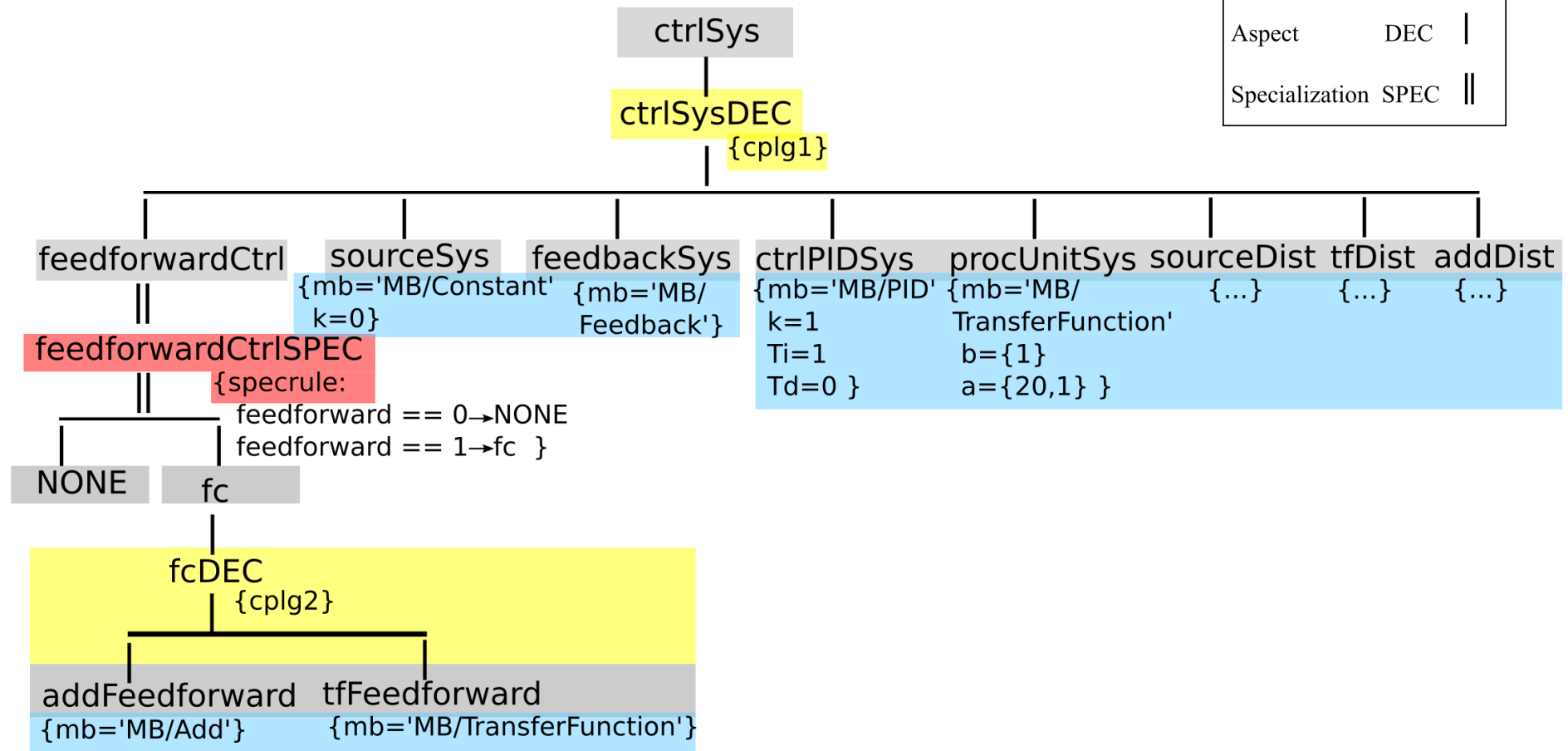




## More Detailed Extract of the SES

**SES** SESVAR={feedforward}  
SemanticCondition={feedforward in [0,1]}

Type	Key	Suffix	Edge
Aspect		DEC	
Specialization		SPEC	





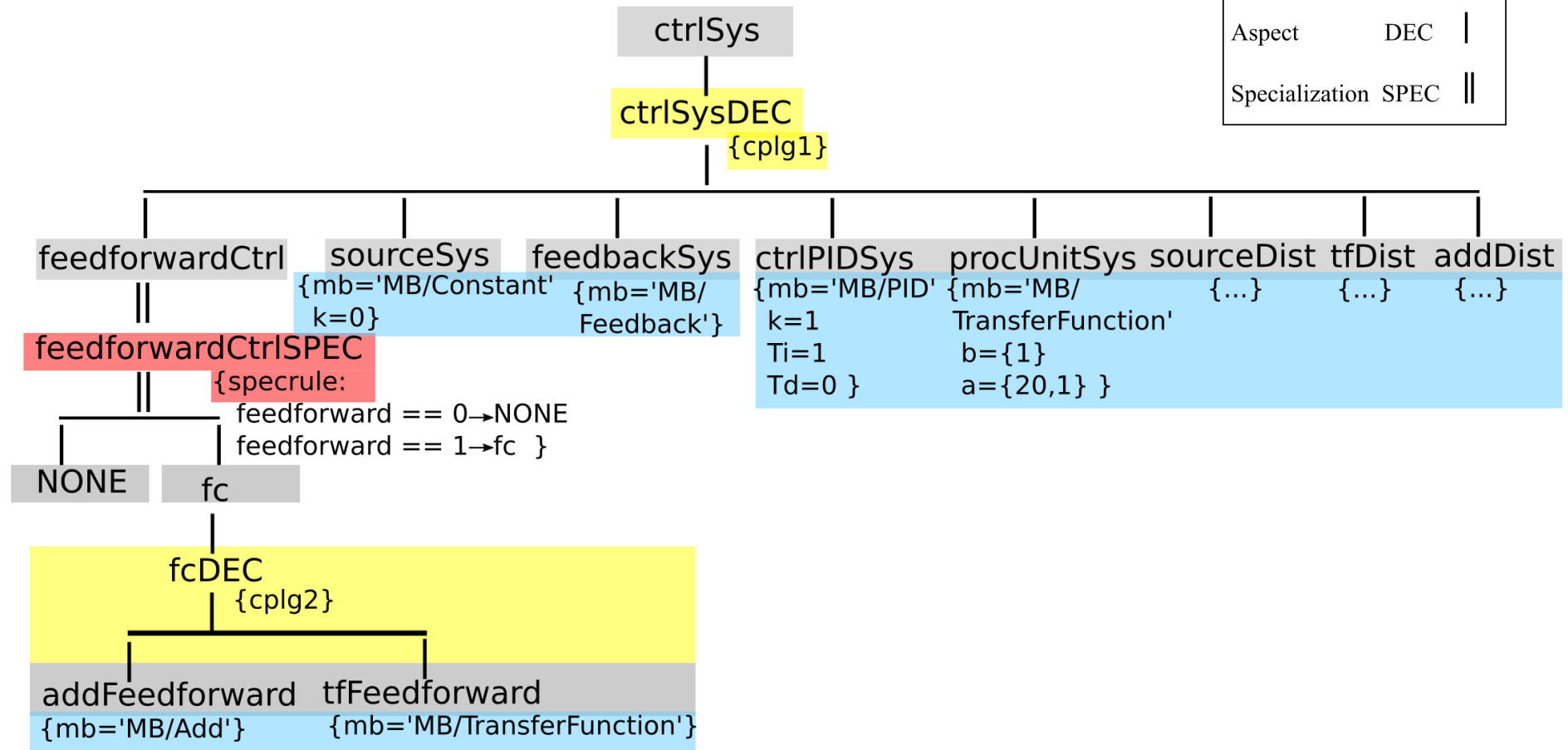
## More Detailed Extract of the SES

**SES**

SESVAR={feedforward}

SemanticCondition={feedforward in [0,1]}

Type	Key	Suffix	Edge
Aspect		DEC	
Specialization		SPEC	





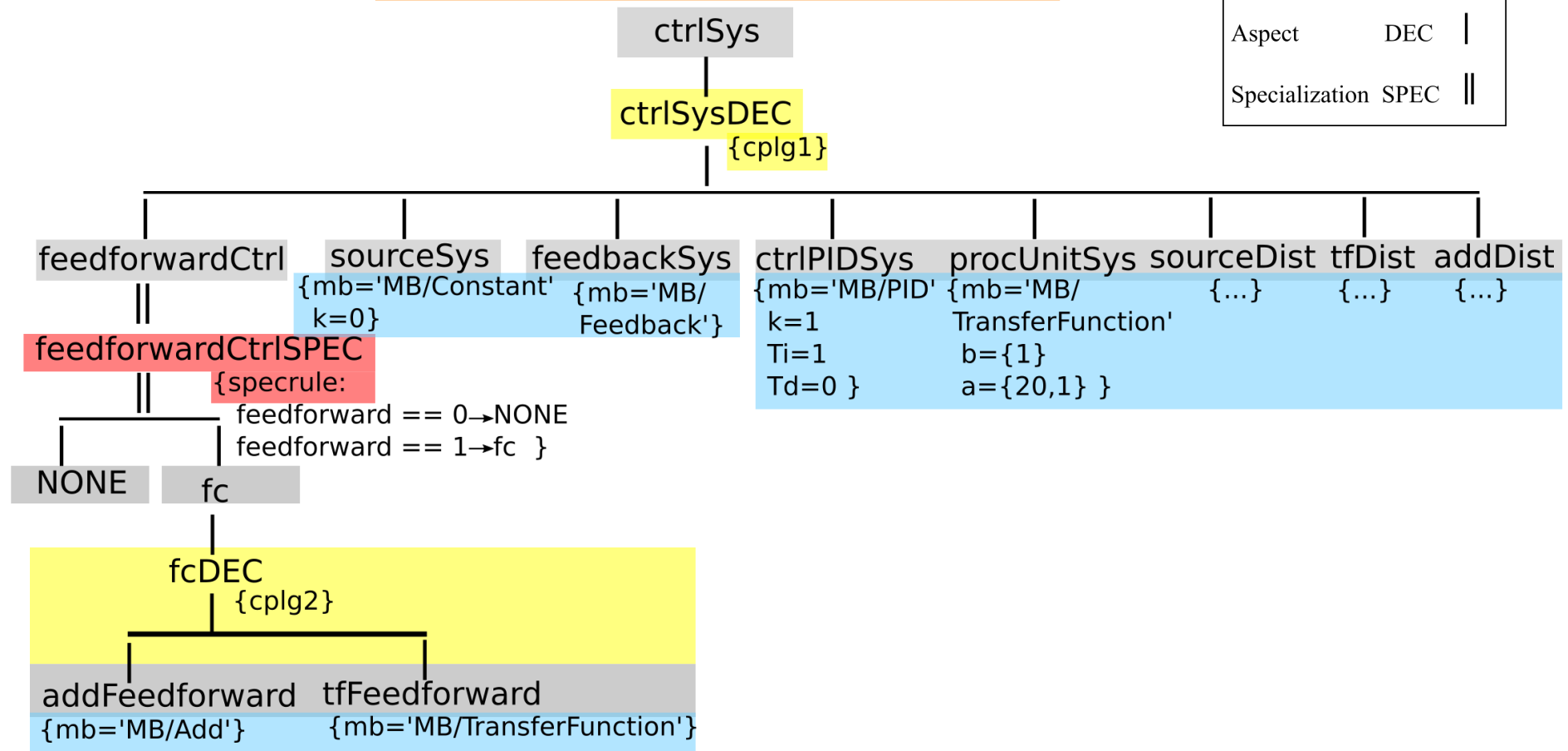
## More Detailed Extract of the SES

**SES**

SESVAR={feedforward}

SemanticCondition={feedforward in [0,1]}

Type	Key	Suffix	Edge
Aspect		DEC	
Specialization		SPEC	





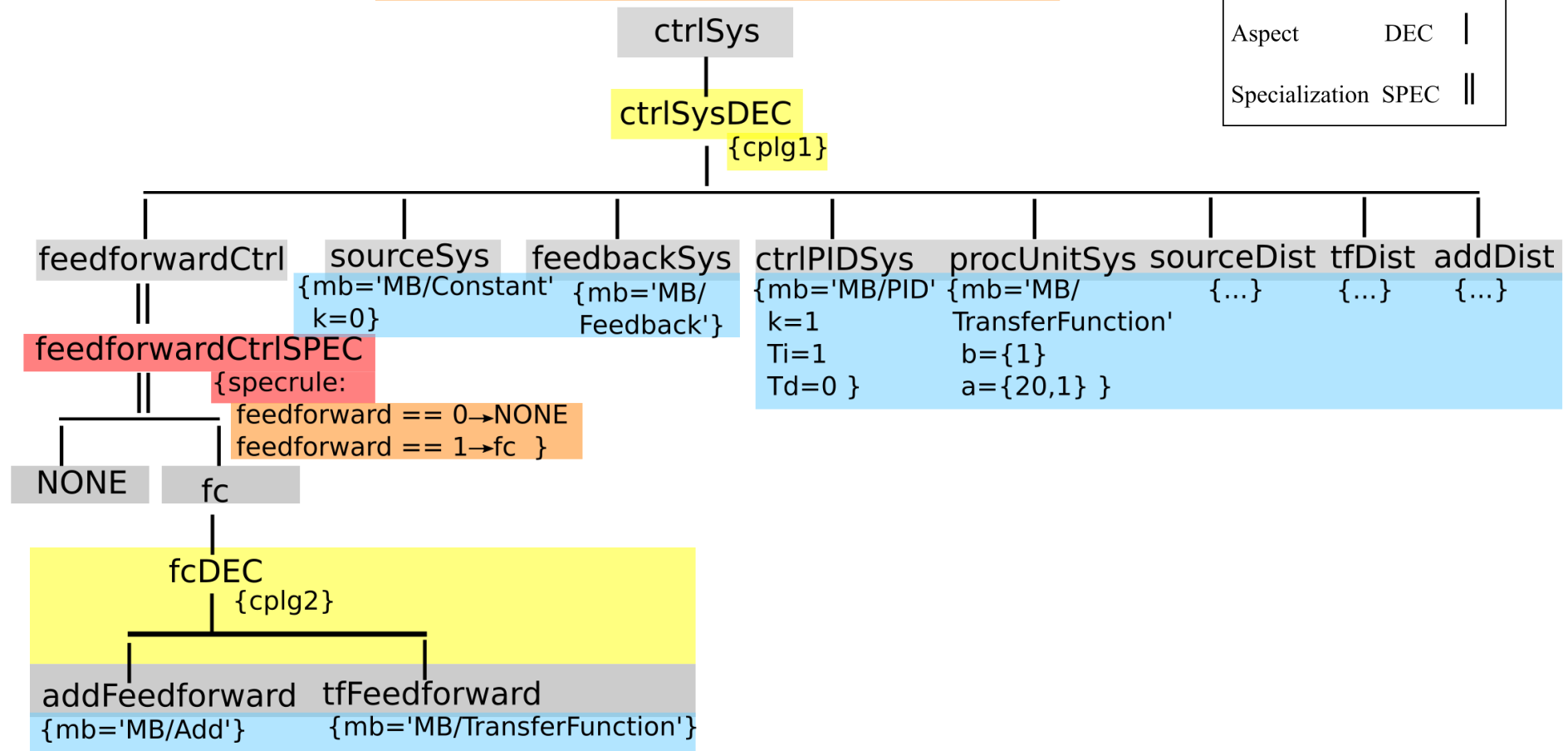
## More Detailed Extract of the SES

**SES**

SESVAR={feedforward}

SemanticCondition={feedforward in [0,1]}

Type	Key	Suffix	Edge
Aspect		DEC	
Specialization		SPEC	







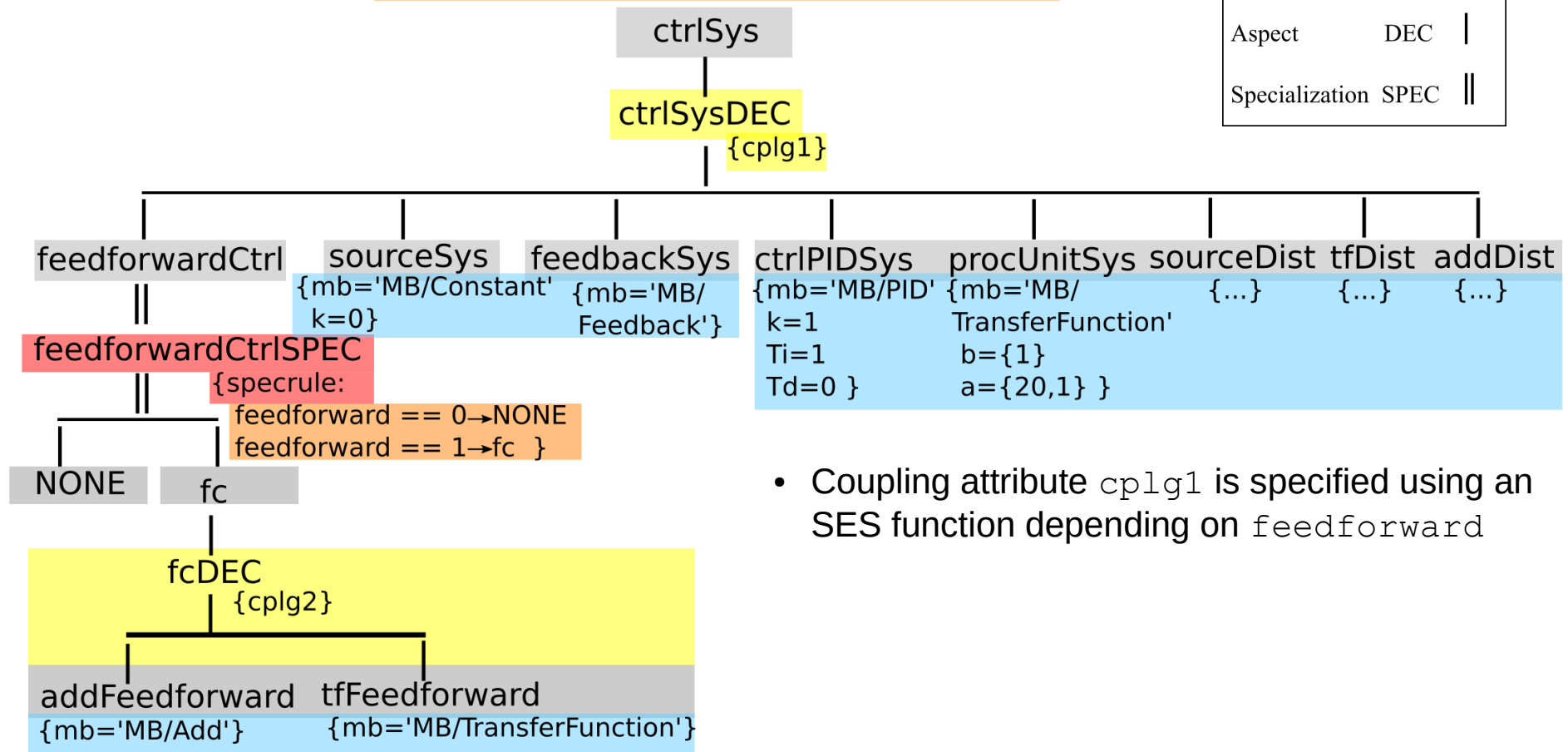
## More Detailed Extract of the SES

**SES**

SESVAR={feedforward}

SemanticCondition={feedforward in [0,1]}

Type	Key	Suffix	Edge
Aspect		DEC	
Specialization		SPEC	





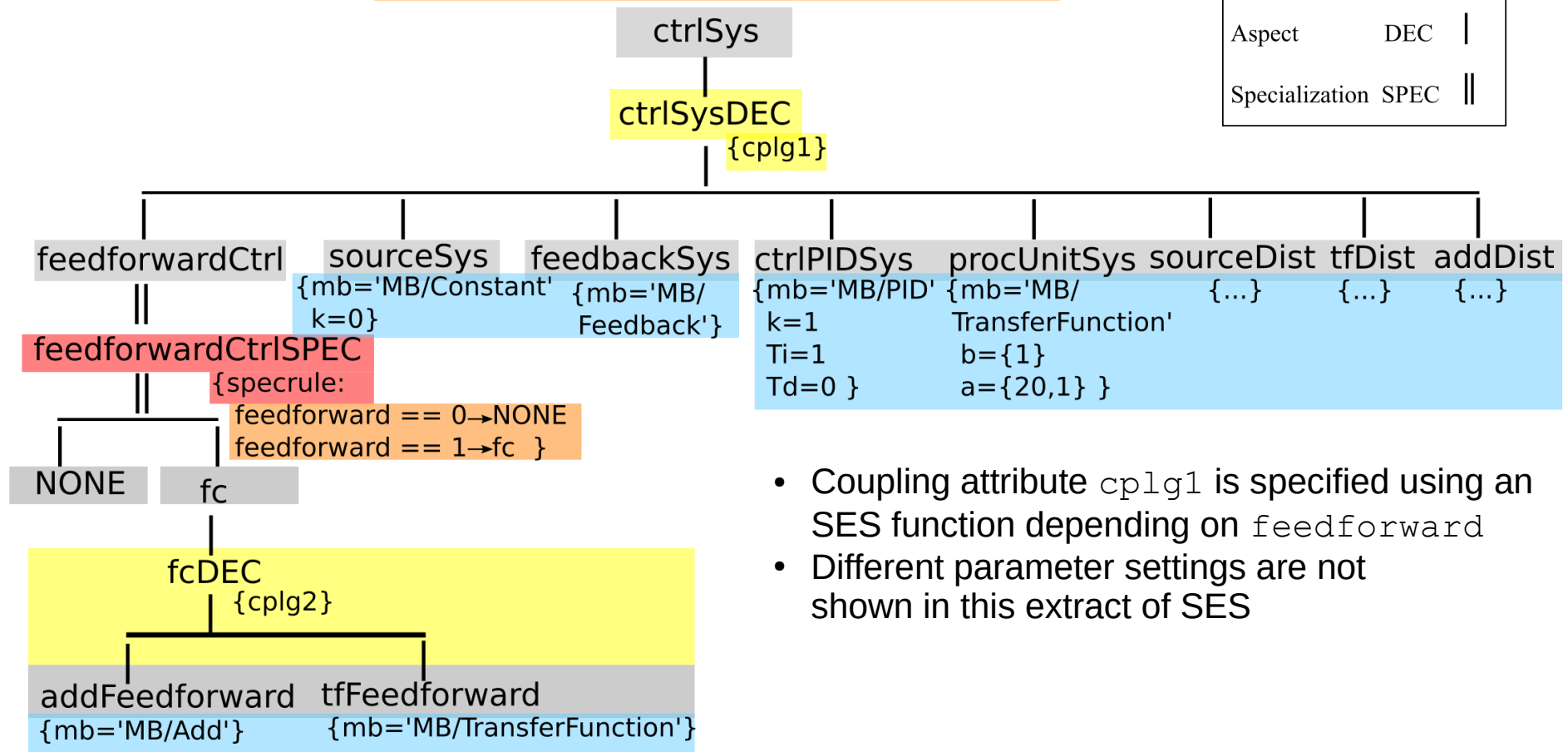
## More Detailed Extract of the SES

**SES**

SESVAR={feedforward}

SemanticCondition={feedforward in [0,1]}

Type	Key	Suffix	Edge
Aspect		DEC	
Specialization		SPEC	



- Coupling attribute **cplg1** is specified using an SES function depending on **feedforward**
- Different parameter settings are not shown in this extract of SES



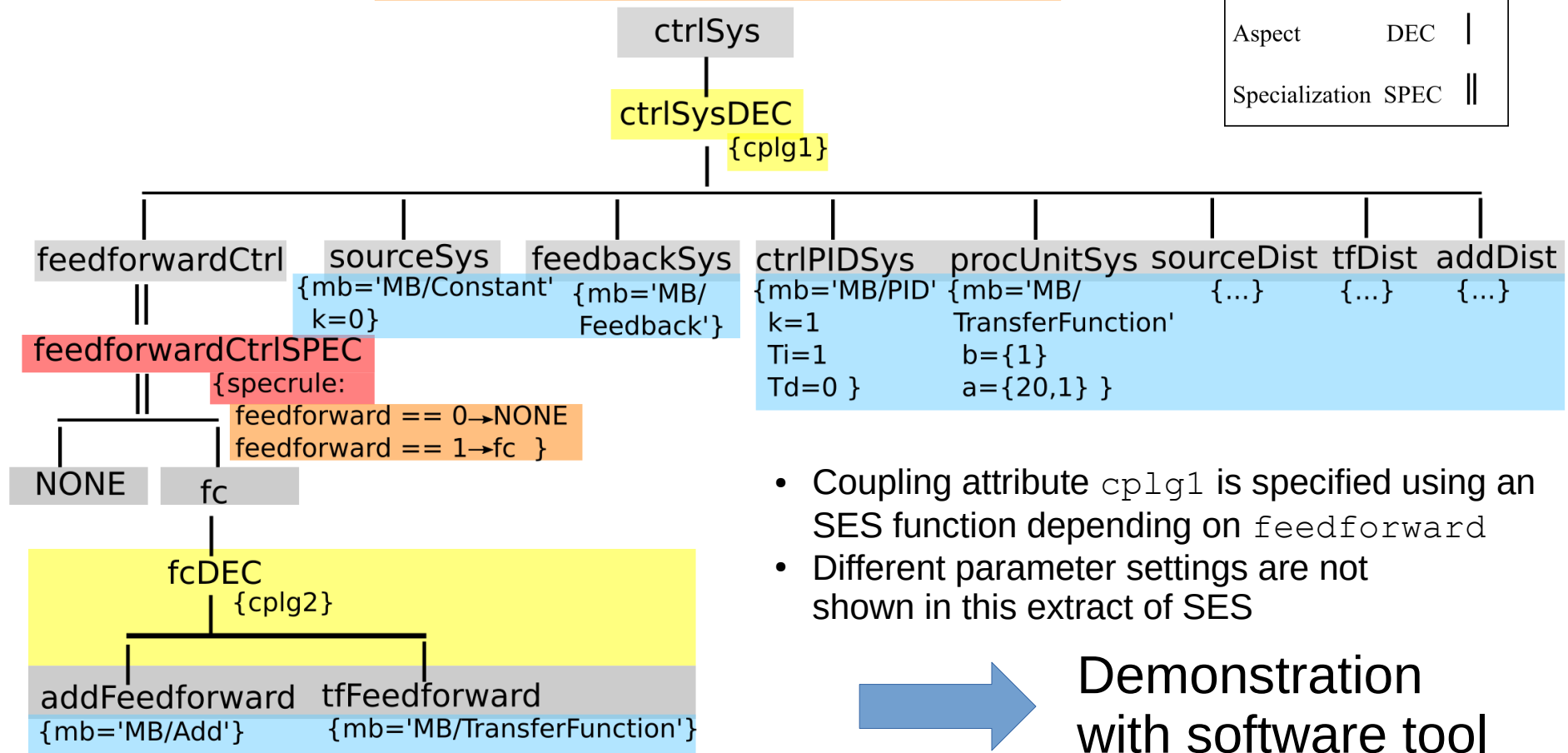
## More Detailed Extract of the SES

**SES**

SESVAR={feedforward}

SemanticCondition={feedforward in [0,1]}

Type	Key	Suffix	Edge
Aspect		DEC	
Specialization		SPEC	





## Outline

1. Introduction
2. The case study
3. Basics of SES/MB based modeling
- 4. Practical modeling: implementation of an SES**  
(H. Folkerts)
5. Model selection and model generation
6. Organization of a simulator-independent MB
7. Full automation of simulation experiments
8. Conclusion



## Python Toolset

- Available: <https://github.com/hendrikfolkerts>
- Tools
  - **SESToPy** → SES editor and IDE
  - **SESViewEI** → SES tree viewer
  - SESMoPy
  - SESEuPy
  - SESEcPy



## Demonstration of SESToPy with SESViewEI (case study)

- Connect SESToPy with SESViewEI (show SESToPy and SESViewEI next to each other)
- Add sub node, add sibling node, change type of node, rename node, delete node, inflate tree, deflate tree
- Edit entity node, descriptive node (aspect, specialization)
- Empty current model
- Save/Load model (JSON) → load `Feedback.jsonsestree` example
- (Export/Import model (XML))
- Maximize SESToPy
- Use the feedback example to show:
  - SES Variables, Semantic Conditions
  - Selection rules → here: `specrule`
  - NONE node
  - Attributes, `mb-attribute` (decouple name of node and name of basic model)
  - Coupling list (composition of basic models)
  - SES function to set couplings (dynamic coupling) → procedural knowledge
- Mention merging

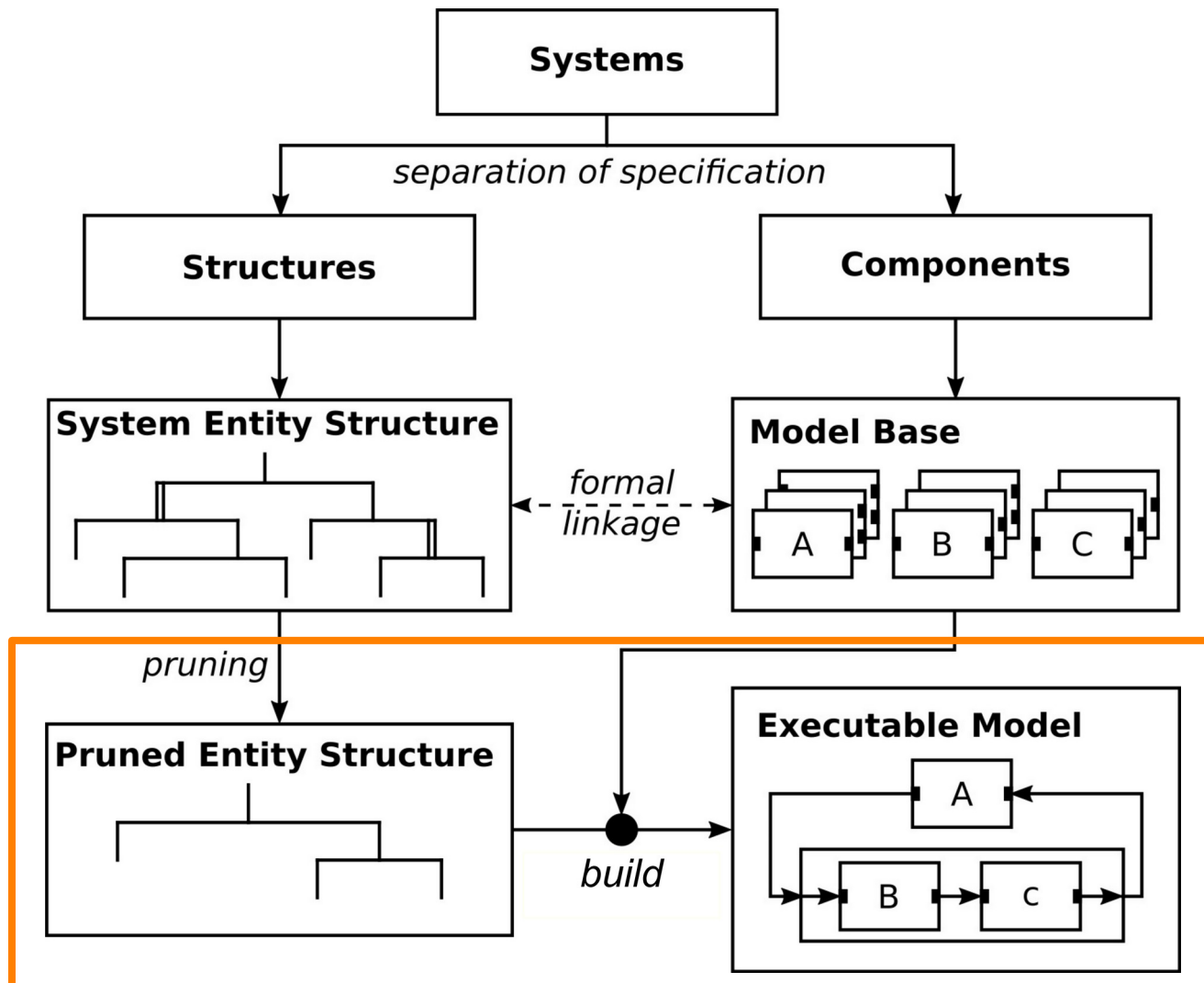


## Outline

1. Introduction
2. The case study
3. Basics of SES/MB based modeling
4. Practical modeling: implementation of an SES
- 5. Model selection and model generation** (H. Folkerts)
6. Organization of a simulator-independent MB
7. Full automation of simulation experiments
8. Conclusion



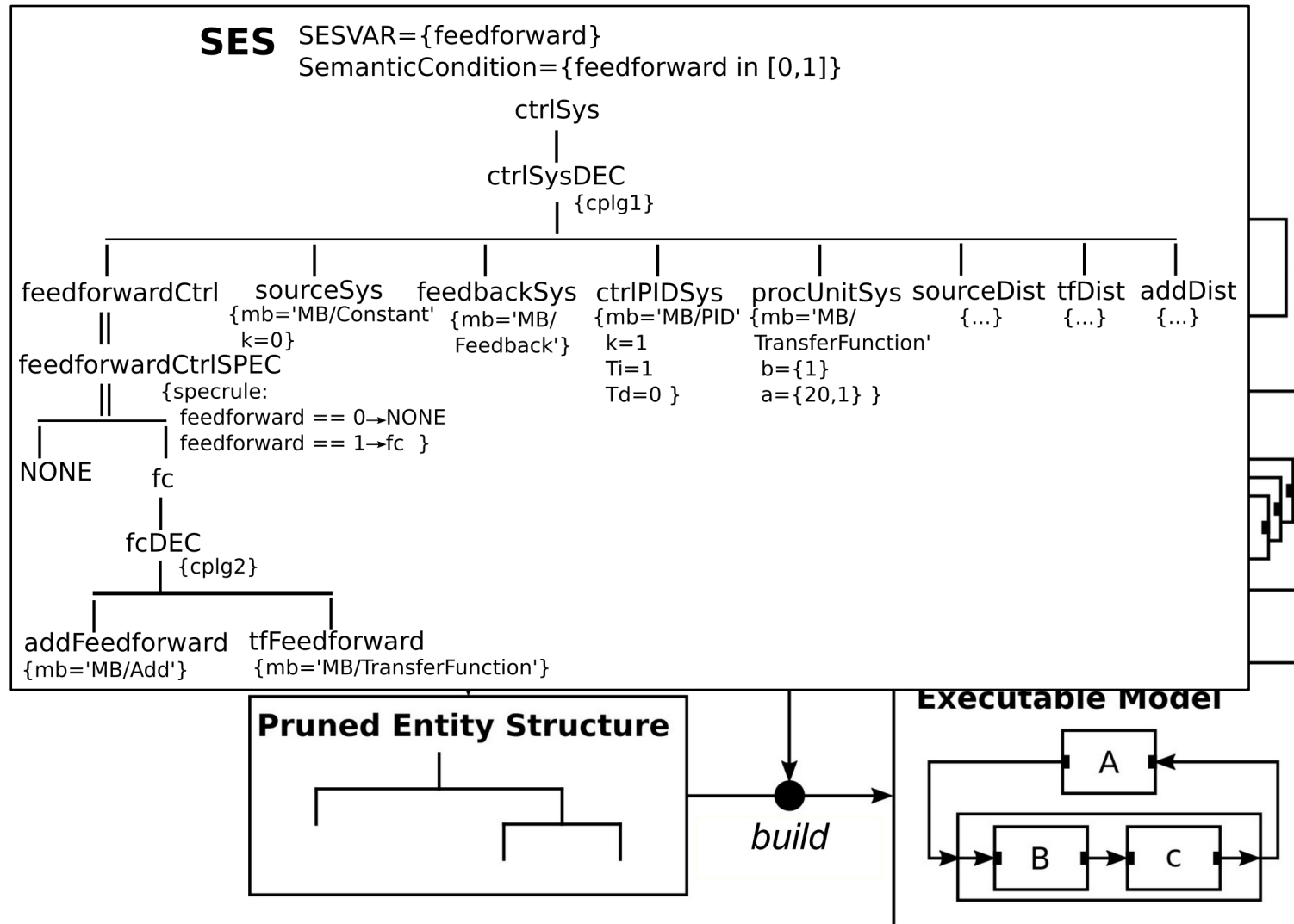
## Model Selection and Generation





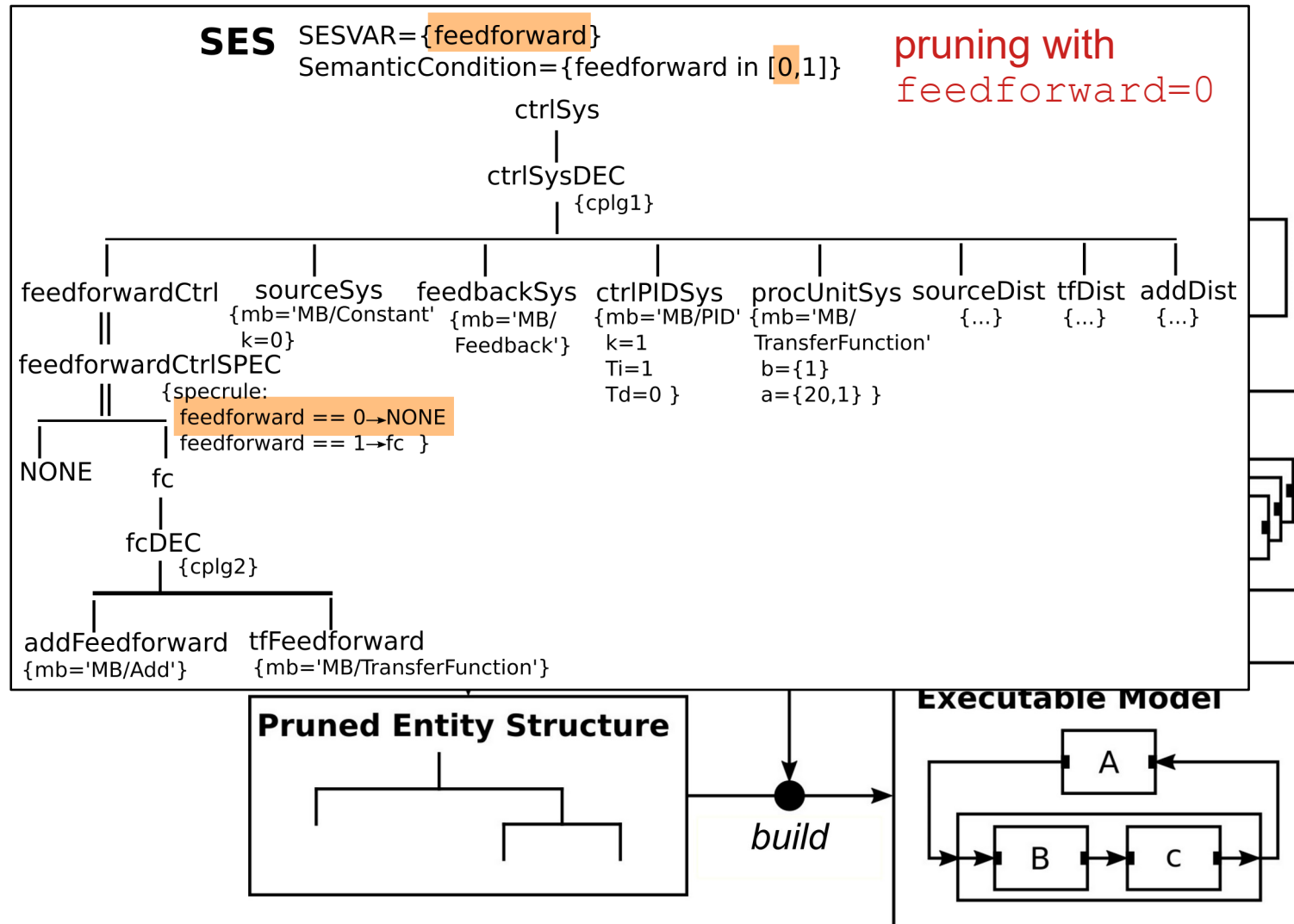


## Model Selection and Generation of Variant #1



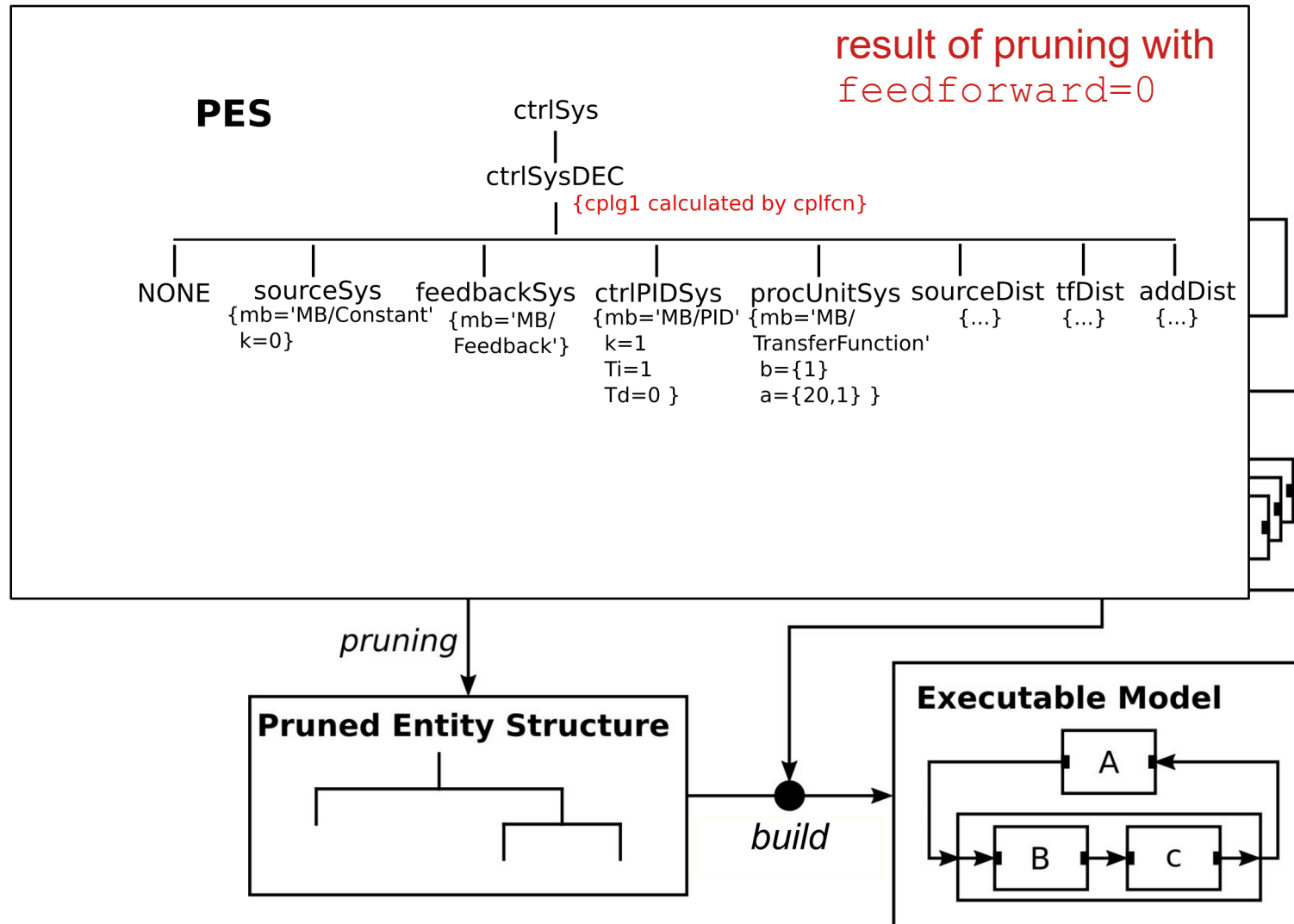


## Model Selection and Generation of Variant #1



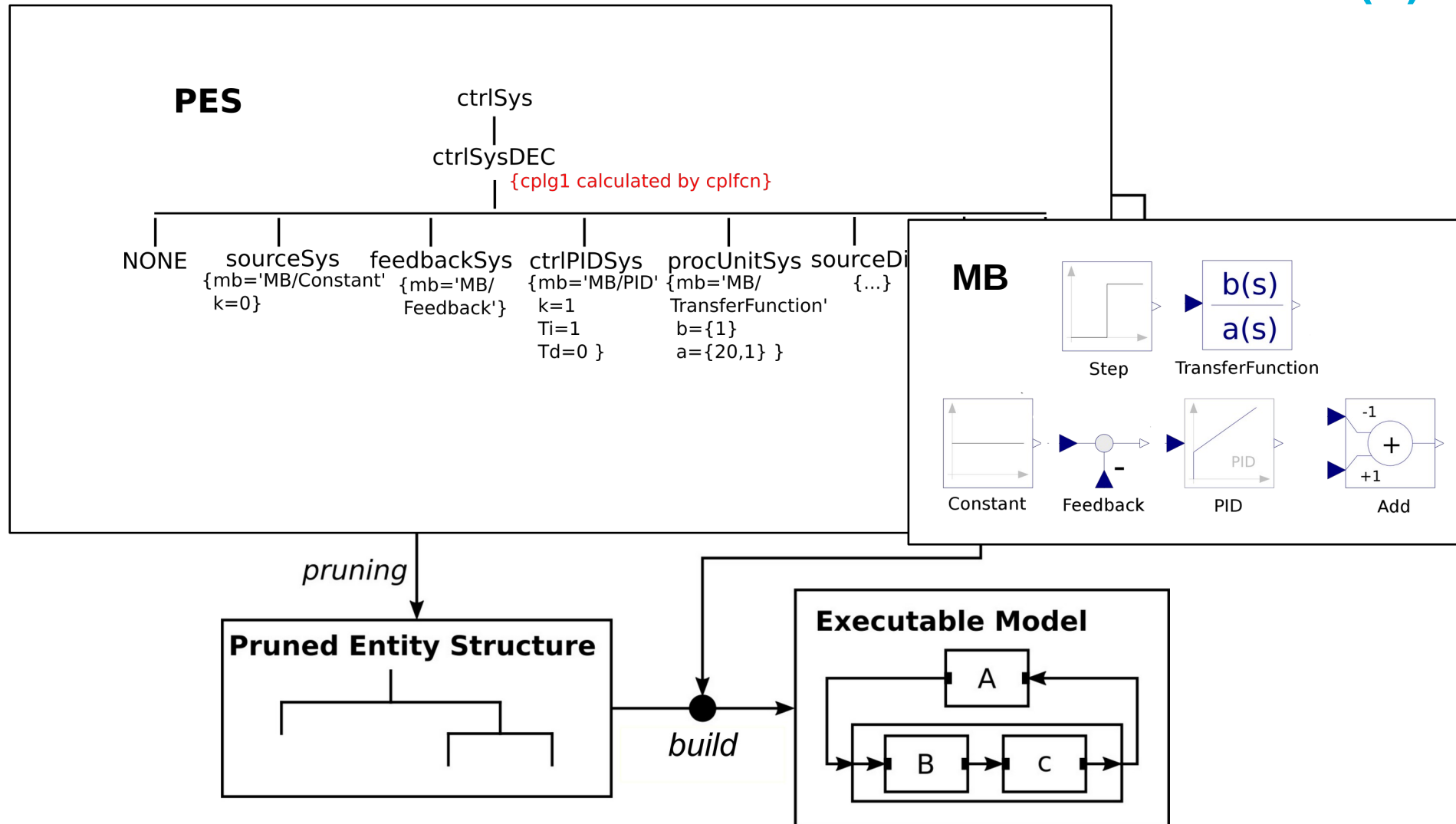


## Model Selection and Generation of Variant #1 (2)



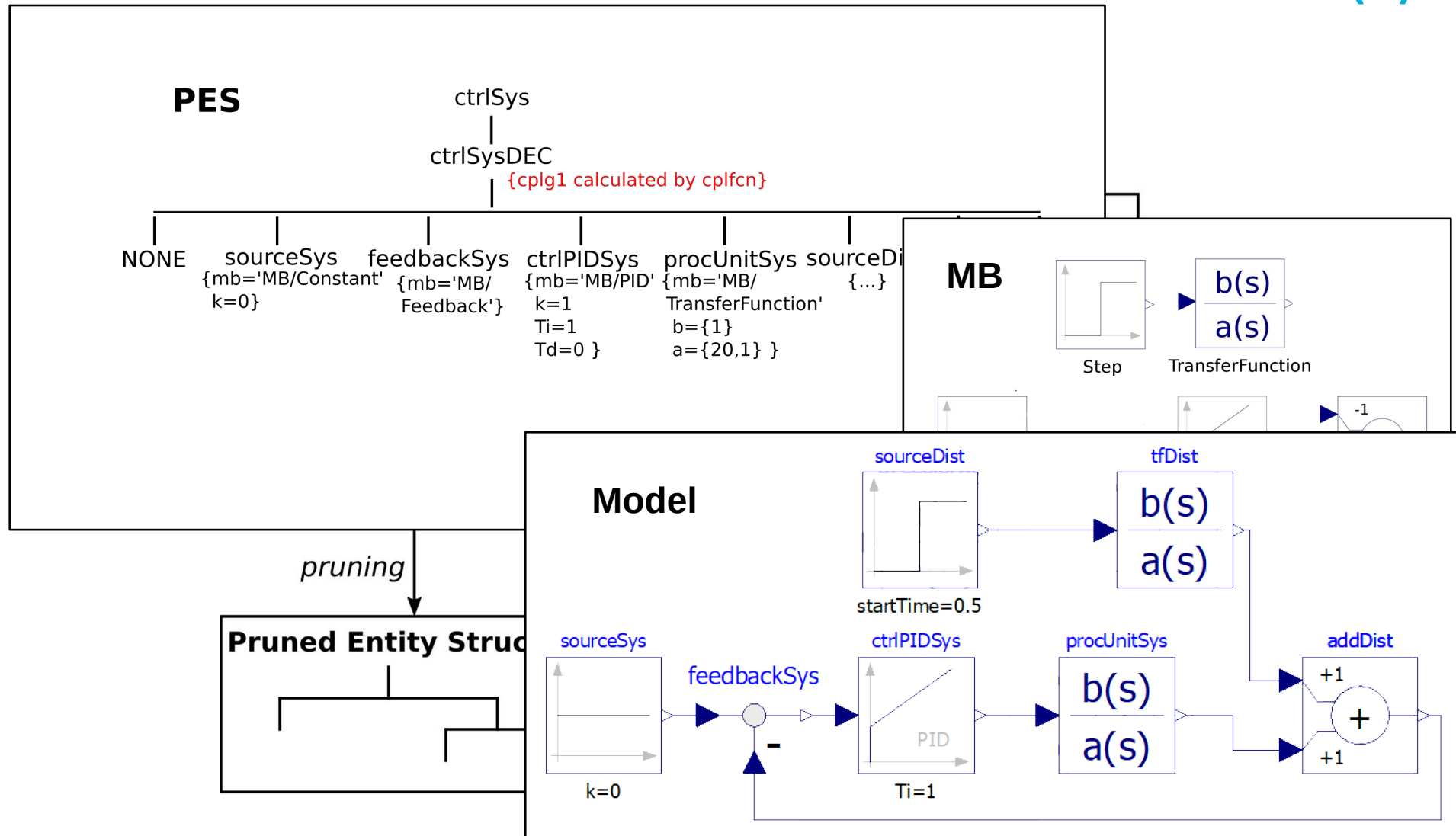


## Model Selection and Generation of Variant #1 (3)



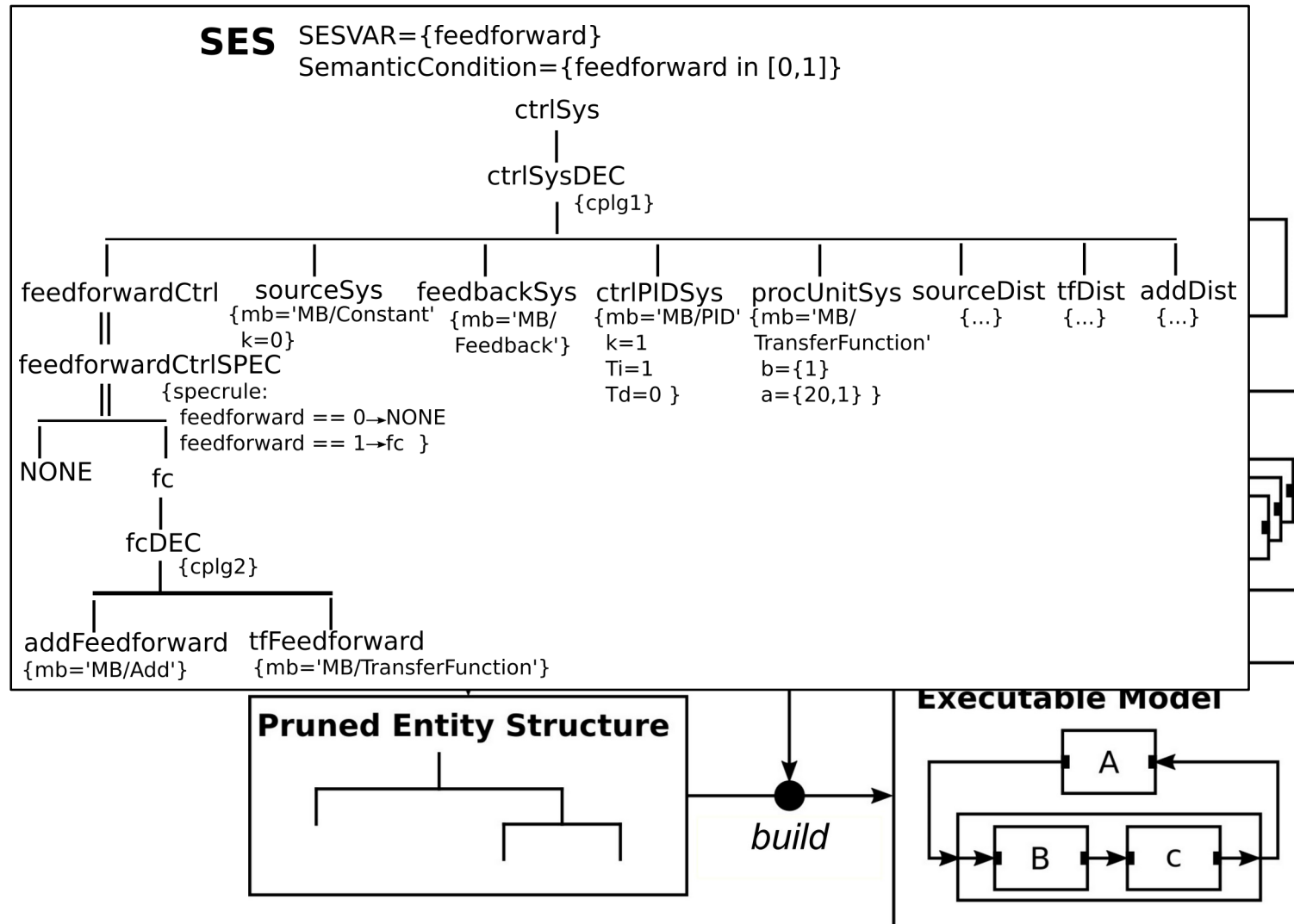


## Model Selection and Generation of Variant #1 (4)



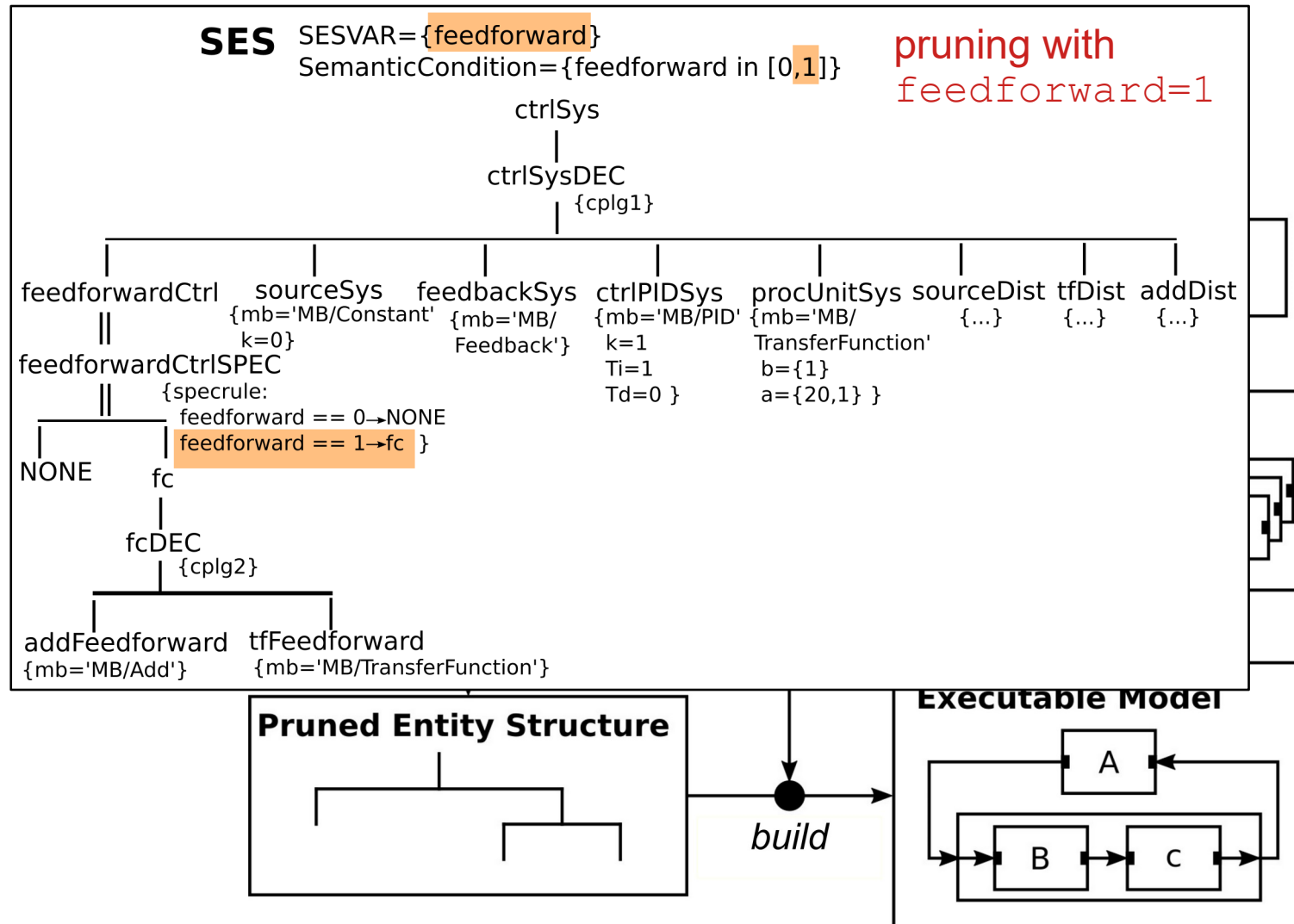


## Model Selection and Generation of Variant #2



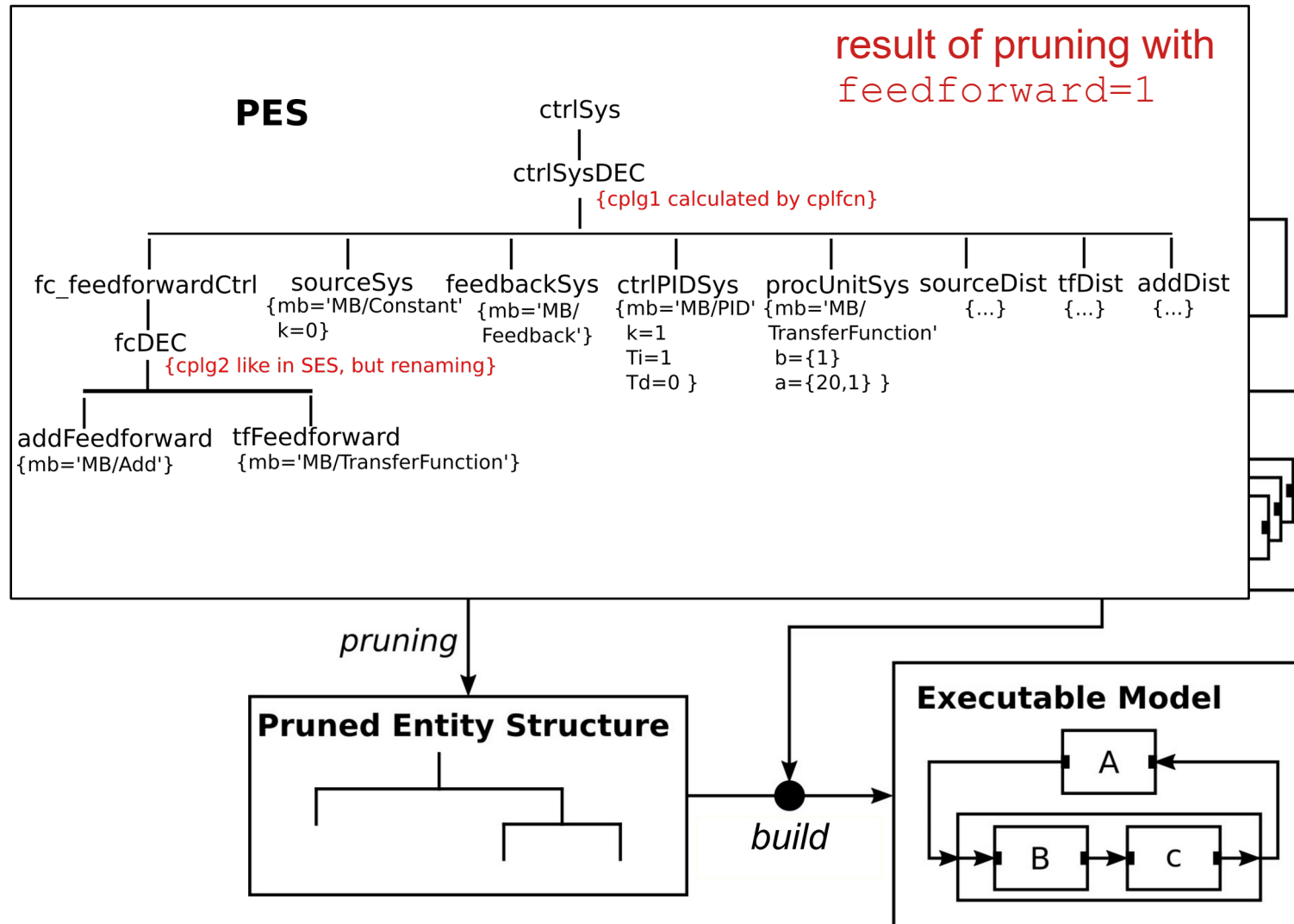


## Model Selection and Generation of Variant #2





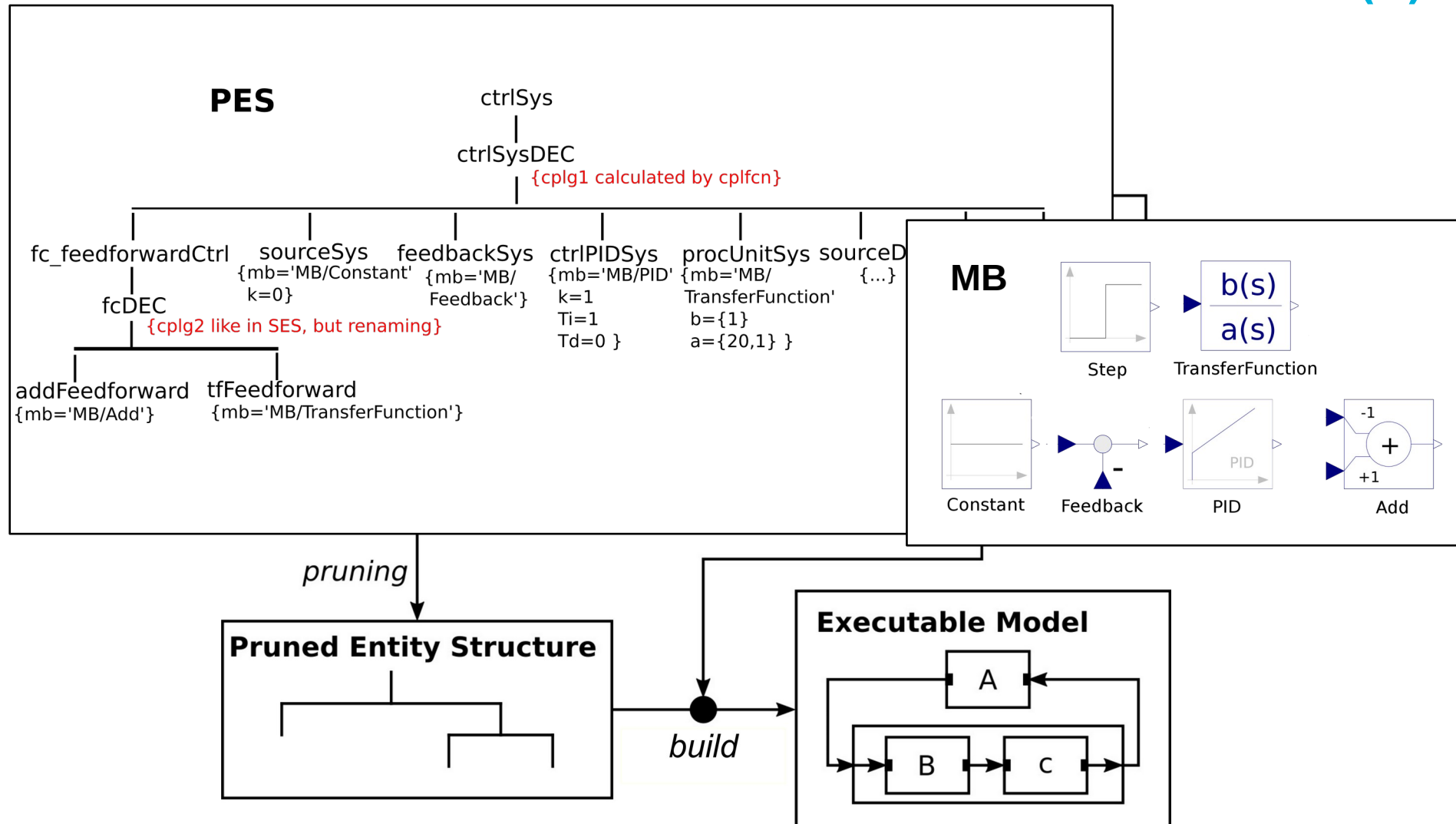
## Model Selection and Generation of Variant #2 (2)





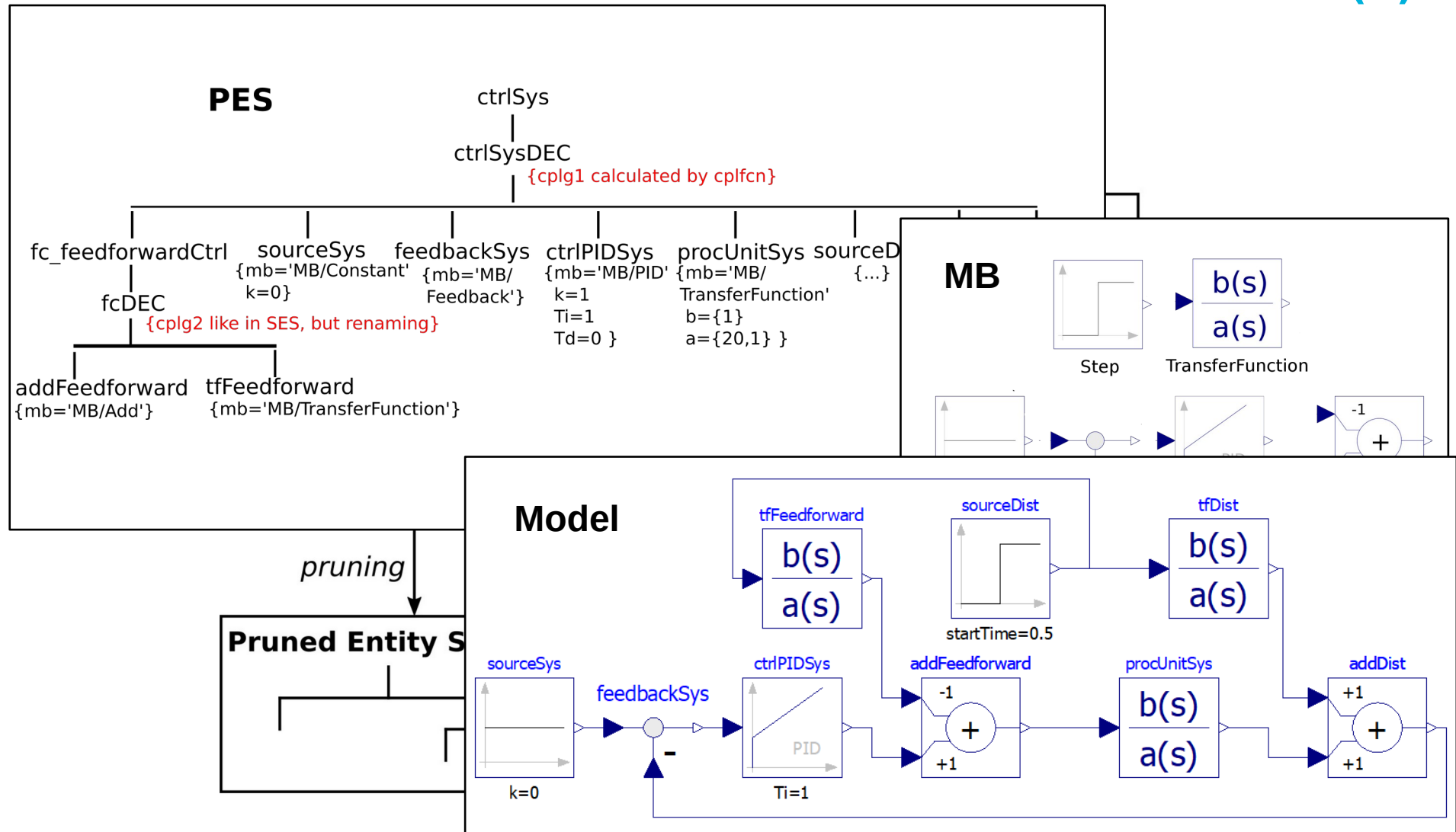


## Model Selection and Generation of Variant #2 (3)



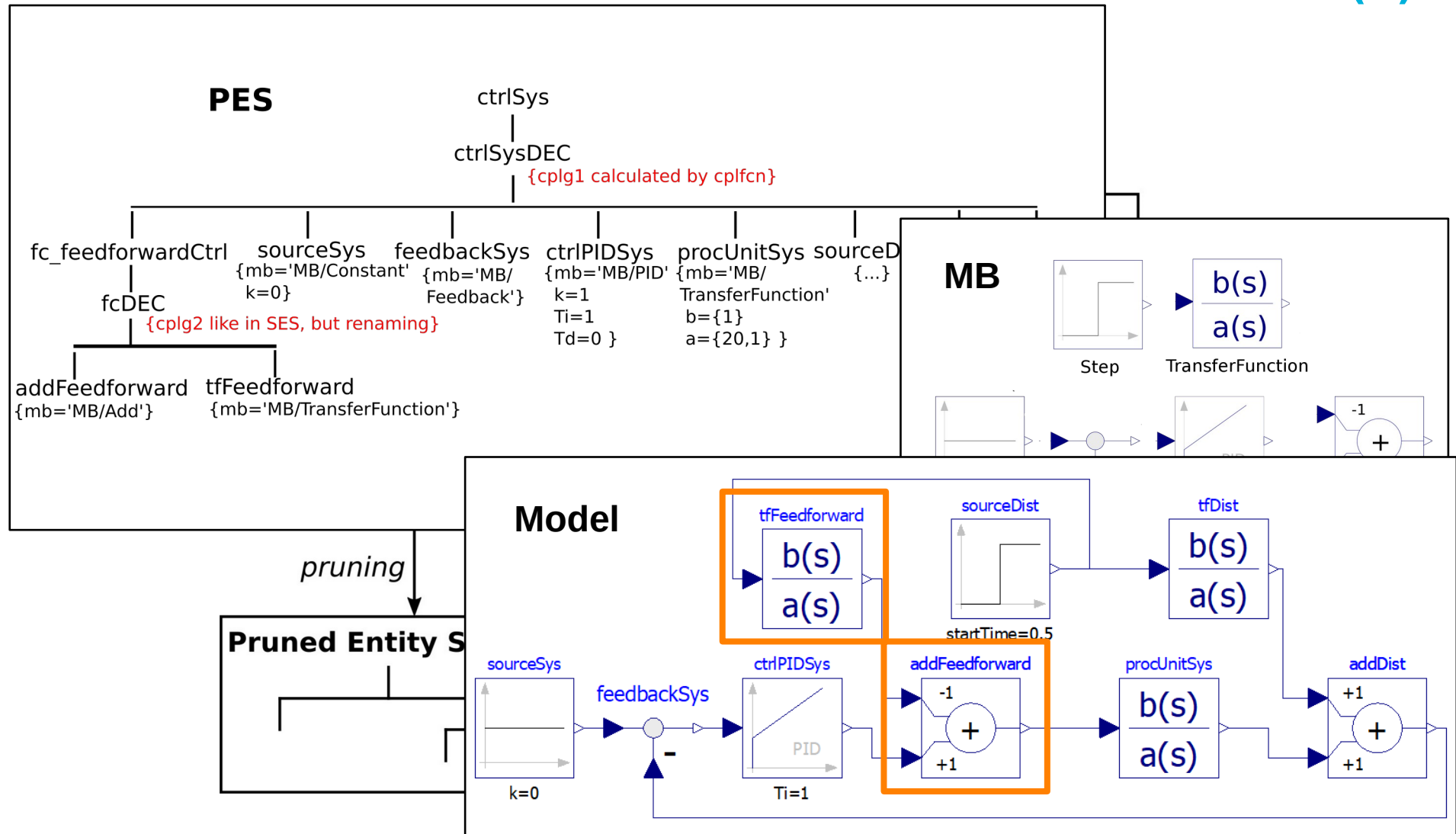


## Model Selection and Generation of Variant #2 (4)





## Model Selection and Generation of Variant #2 (4)





## Python Toolset

- Available: <https://github.com/hendrikfolkerts>

- Tools

- SESToPy → SES editor and IDE

- SESViewEl → SES tree viewer

- **SESMoPy** → Model builder
    - OpenModelica
    - Dymola
    - Simulink

- SESEuPy

- SESEcPy



## Demonstration of SESMoPy (case study)

- Show provisional Experimental Frame from SESMoPy examples → `Template_for_SESMoPy`
- Show that different simulators can be set → here `OpenModelica`
- Show that two interfaces can be set → here `native`
- Merge Feedback SES from SESToPy examples to `simModel` → rename `simModel` to `ctrlSys` for merging.
- Show that configurations can be set in `expMethod`
- Prune for `feedforward=0` → dynamic couplings to static couplings
- Flatten for `feedforward=0` and save the FPES as file → explanation flattening: remove inner, coupled components → root node and leaves stay in tree → couplings recalculated
- (Prune for `feedforward=1` to show)
- Show the OpenModelica MB `MB.mo` and copy it in the same directory to the FPES file
- Open SESMoPy GUI → set FPES → create model → models for both configurations created
- Open one created model in OpenModelica and load MB file
- In OpenModelica open the model by double clicking
- Execute simulation → set simulation time to 50 seconds
  - Signals of interest: `sourceSys.y` `sourceDist.y` `addDist.y`
  - If the signals do not show up in plot: Click `Auto Scale` and `Fit in View` in plot
- If design objectives are not met with this structure and parameterization → later how to simulate automatically to find fitting structure and parameterization



## Outline

1. Introduction
2. The case study
3. Basics of SES/MB based modeling
4. Practical modeling: implementation of an SES
5. Model selection and model generation
- 6. Organization of a simulator-independent MB**  
(H. Folkerts)
7. Full automation of simulation experiments
8. Conclusion



# Model Building – Support Different Simulators Facts / Challenges



## Model Building – Support Different Simulators Facts / Challenges

- System models should be executable with different simulators
  - Simulators are domain specific
  - Verify simulator correctness





## Model Building – Support Different Simulators Facts / Challenges

- System models should be executable with different simulators
  - Simulators are domain specific
  - Verify simulator correctness
- **SES is independent of simulator**



## Model Building – Support Different Simulators Facts / Challenges

- System models should be executable with different simulators
  - Simulators are domain specific
  - Verify simulator correctness
- **SES is independent of simulator**
- **Native model building using a simulator dependent MB**
  - Needs **one** MB for **each** simulator (error prone and costly to maintain)
  - Needs **specific model builders**, because simulators are different (syntax and semantics such as port names, block parameters, ...)



## Model Building – Support Different Simulators Facts / Challenges

- System models should be executable with different simulators
  - Simulators are domain specific
  - Verify simulator correctness
- **SES is independent of simulator**
- **Native model building using a simulator dependent MB**
  - Needs **one** MB for **each** simulator (error prone and costly to maintain)
  - Needs **specific model builders**, because simulators are different (syntax and semantics such as port names, block parameters, ...)
- **Goal: One** (simple) MB and model builder for **all** simulators



## Functional Mock-up Interface (FMI) <sup>1,2</sup>

<sup>1</sup>Blochwitz et al. (2011) „The Functional Mockup Interface for Tool independent Exchange of Simulation Models“. Proc. of the 8th Modelica Conference, Dresden.

<sup>2</sup>Blochwitz et al. (2012) „Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models“. Proc. of the 9th Modelica Conference, Munich.



## Functional Mock-up Interface (FMI) <sup>1,2</sup>

- FMI defines a standardized interface of components (models, blocks)

<sup>1</sup>Blochwitz et al. (2011) „The Functional Mockup Interface for Tool independent Exchange of Simulation Models“. Proc. of the 8th Modelica Conference, Dresden.

<sup>2</sup>Blochwitz et al. (2012) „Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models“. Proc. of the 9th Modelica Conference, Munich.



## Functional Mock-up Interface (FMI) <sup>1,2</sup>

- FMI defines a standardized interface of components (models, blocks)
  - Reuse of components
    - (i) For **model exchange**
    - (ii) For co-simulation

<sup>1</sup>Blochwitz et al. (2011) „The Functional Mockup Interface for Tool independent Exchange of Simulation Models“. Proc. of the 8th Modelica Conference, Dresden.

<sup>2</sup>Blochwitz et al. (2012) „Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models“. Proc. of the 9th Modelica Conference, Munich.



## Functional Mock-up Interface (FMI) <sup>1,2</sup>

- FMI defines a standardized interface of components (models, blocks)
  - Reuse of components
    - (i) For **model exchange**
    - (ii) For co-simulation
  - Based on C code or binaries

<sup>1</sup>Blochwitz et al. (2011) „The Functional Mockup Interface for Tool independent Exchange of Simulation Models“. Proc. of the 8th Modelica Conference, Dresden.

<sup>2</sup>Blochwitz et al. (2012) „Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models“. Proc. of the 9th Modelica Conference, Munich.



## Functional Mock-up Interface (FMI) <sup>1,2</sup>

- FMI defines a standardized interface of components (models, blocks)
  - Reuse of components
    - (i) For **model exchange**
    - (ii) For co-simulation
  - Based on C code or binaries
  - Many simulators support FMI

<sup>1</sup>Blochwitz et al. (2011) „The Functional Mockup Interface for Tool independent Exchange of Simulation Models“. Proc. of the 8th Modelica Conference, Dresden.

<sup>2</sup>Blochwitz et al. (2012) „Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models“. Proc. of the 9th Modelica Conference, Munich.





## Functional Mock-up Interface (FMI) <sup>1,2</sup>

- FMI defines a standardized interface of components (models, blocks)
  - Reuse of components
    - (i) For **model exchange**
    - (ii) For co-simulation
  - Based on C code or binaries
  - Many simulators support FMI
  - Still problems for discrete event models

<sup>1</sup>Blochwitz et al. (2011) „The Functional Mockup Interface for Tool independent Exchange of Simulation Models“. Proc. of the 8th Modelica Conference, Dresden.

<sup>2</sup>Blochwitz et al. (2012) „Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models“. Proc. of the 9th Modelica Conference, Munich.



## Functional Mock-up Unit (FMU)



## Functional Mock-up Unit (FMU)

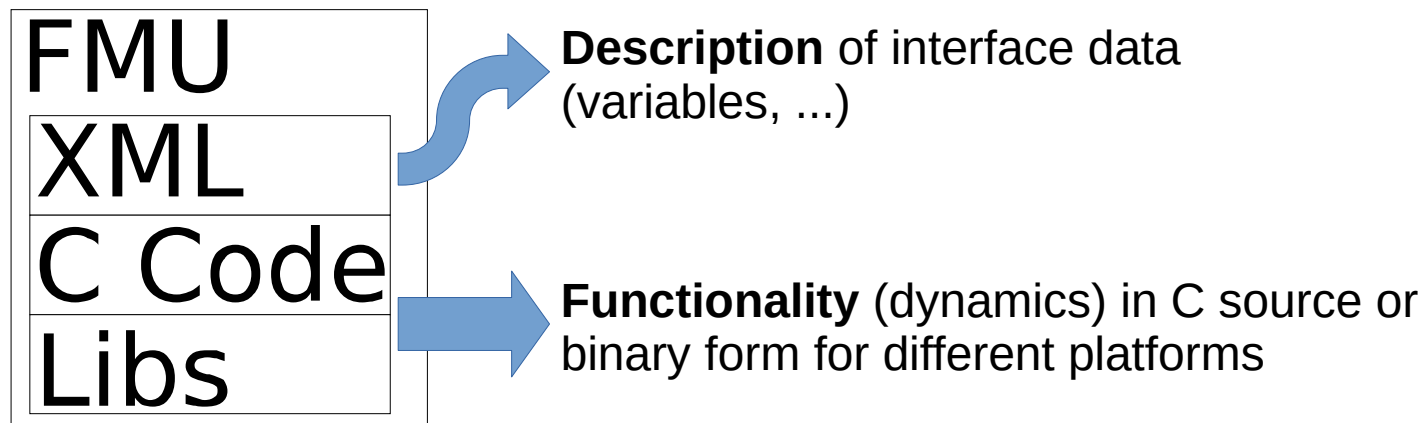
Component implementing FMI = Functional Mock-up Unit (FMU)  
a zipped file with fileextension .fmu



## Functional Mock-up Unit (FMU)

Component implementing FMI = Functional Mock-up Unit (FMU)

a zipped file with fileextension .fmu

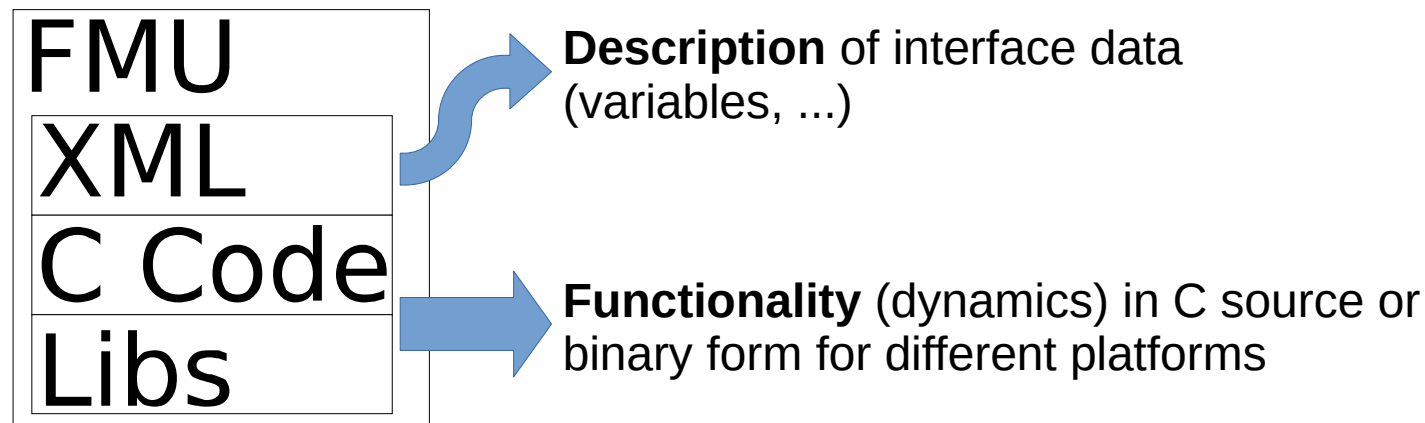




## Functional Mock-up Unit (FMU)

Component implementing FMI = Functional Mock-up Unit (FMU)

a zipped file with fileextension .fmu



Model Exchange

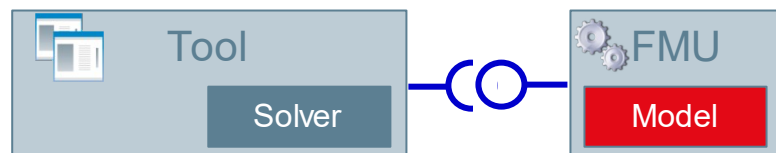


Figure taken from the FMI presentation on the website [www.fmi-standard.org](http://www.fmi-standard.org).



# Model Building – Support Different Simulators Using FMI



## Model Building – Support Different Simulators Using FMI

- **Idea:** Using FMI for model exchange



## Model Building – Support Different Simulators Using FMI

- **Idea:** Using FMI for model exchange
  - Export basic models as FMUs from any simulator to create simulator-independent MB





# Model Building – Support Different Simulators Using FMI

- **Idea:** Using FMI for model exchange
  - Export basic models as FMUs from any simulator to create simulator-independent MB
  - Model generation (build):
    - Import and configure FMUs from MB and create couplings



# Model Building – Support Different Simulators Using FMI

- **Idea:** Using FMI for model exchange
  - Export basic models as FMUs from any simulator to create simulator-independent MB
  - Model generation (build):
    - Import and configure FMUs from MB and create couplings
  - **Problems:**
    - Some simulators do not support configuration of FMUs (basic models) and creation of couplings
    - FMU import is time consuming → slow model building



# Model Building – Support Different Simulators Using FMI

- **Idea:** Using FMI for model exchange
  - Export basic models as FMUs from any simulator to create simulator-independent MB
  - Model generation (build):
    - Import and configure FMUs from MB and create couplings
  - **Problems:**
    - Some simulators do not support configuration of FMUs (basic models) and creation of couplings
    - FMU import is time consuming → slow model building
- **Workaround:** Using FMI for model exchange and OpenModelica



# Model Building – Support Different Simulators Using FMI

- **Idea:** Using FMI for model exchange
  - Export basic models as FMUs from any simulator to create simulator-independent MB
  - Model generation (build):
    - Import and configure FMUs from MB and create couplings
  - **Problems:**
    - Some simulators do not support configuration of FMUs (basic models) and creation of couplings
    - FMU import is time consuming → slow model building
- **Workaround:** Using FMI for model exchange and OpenModelica
  - Export basic models as FMUs from any simulator to create simulator-independent MB

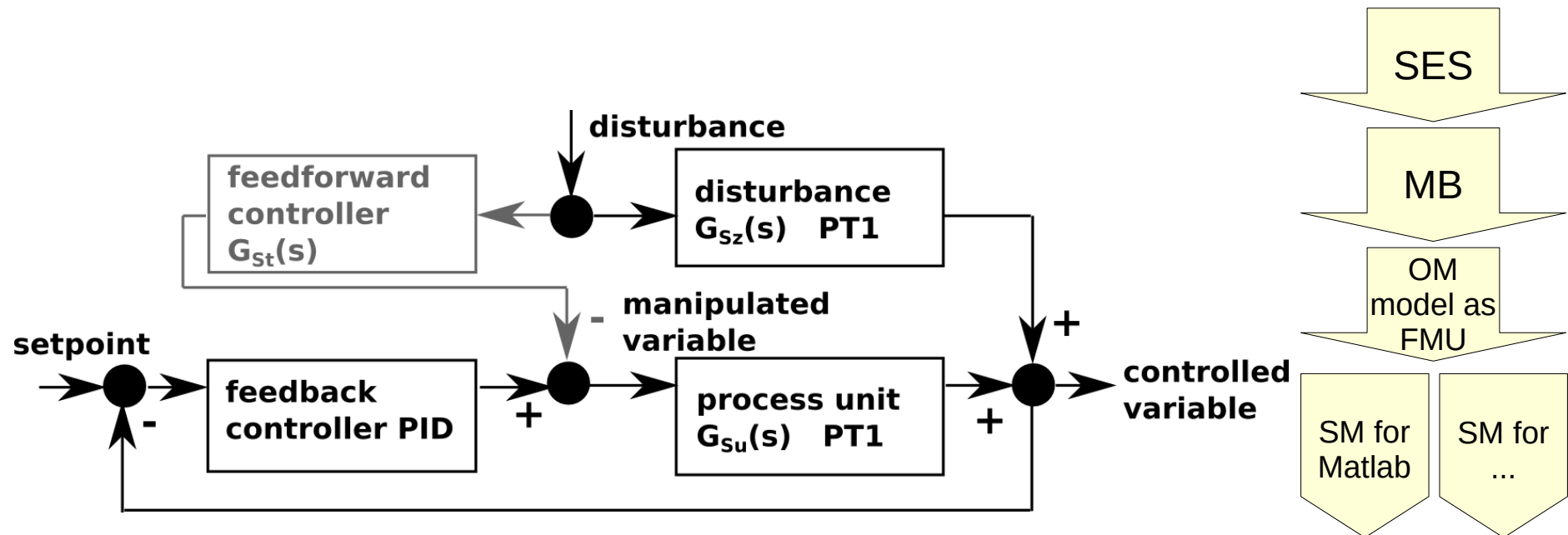


# Model Building – Support Different Simulators Using FMI

- **Idea:** Using FMI for model exchange
  - Export basic models as FMUs from any simulator to create simulator-independent MB
  - Model generation (build):
    - Import and configure FMUs from MB and create couplings
  - **Problems:**
    - Some simulators do not support configuration of FMUs (basic models) and creation of couplings
    - FMU import is time consuming → slow model building
- **Workaround:** Using FMI for model exchange and OpenModelica
  - Export basic models as FMUs from any simulator to create simulator-independent MB
  - Model generation (build):
    - Import and configure FMUs from MB and create couplings in OpenModelica
    - Export the configured model as one FMU
    - Import model FMU in the target simulator

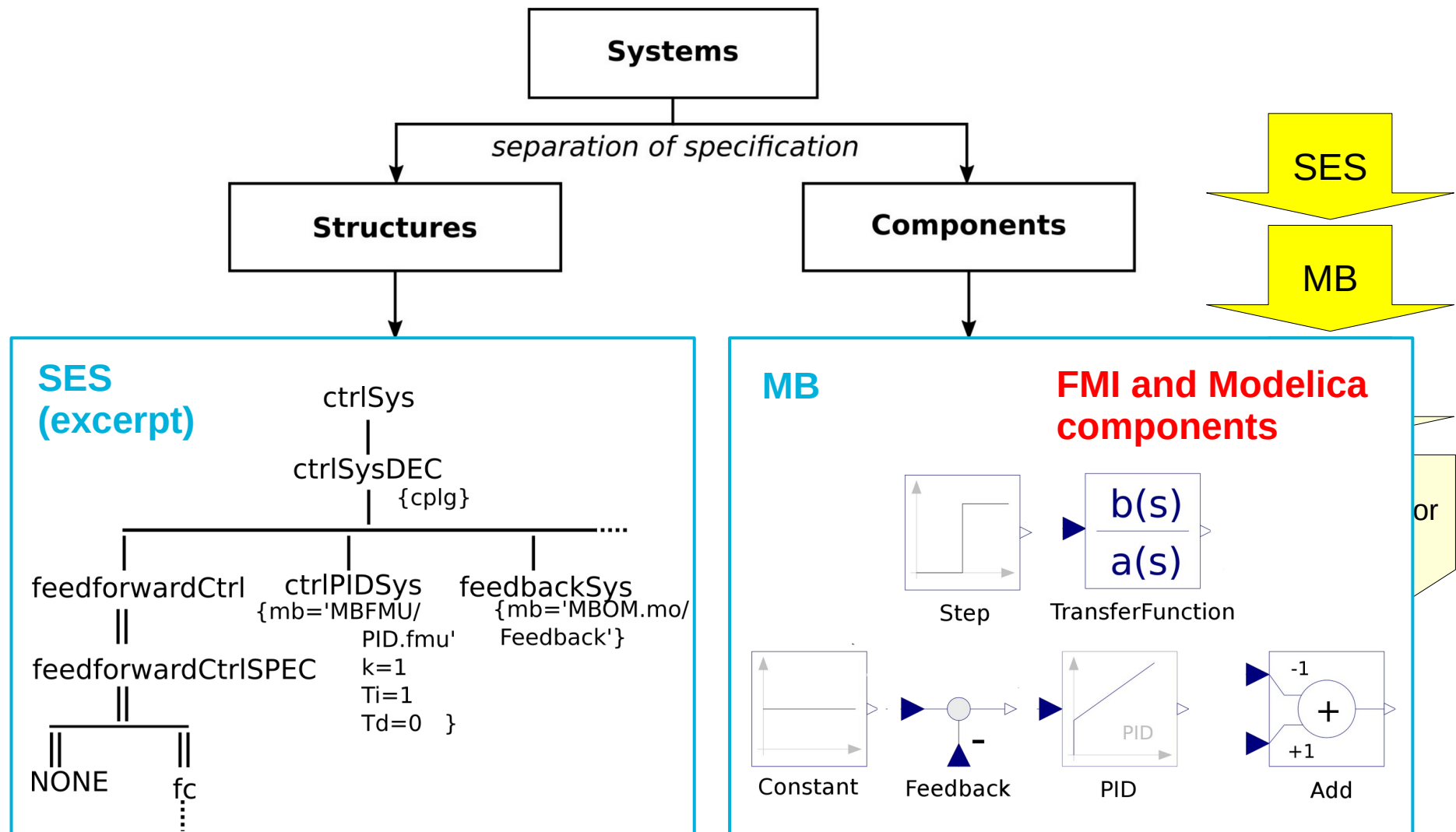


## Case Study with FMI



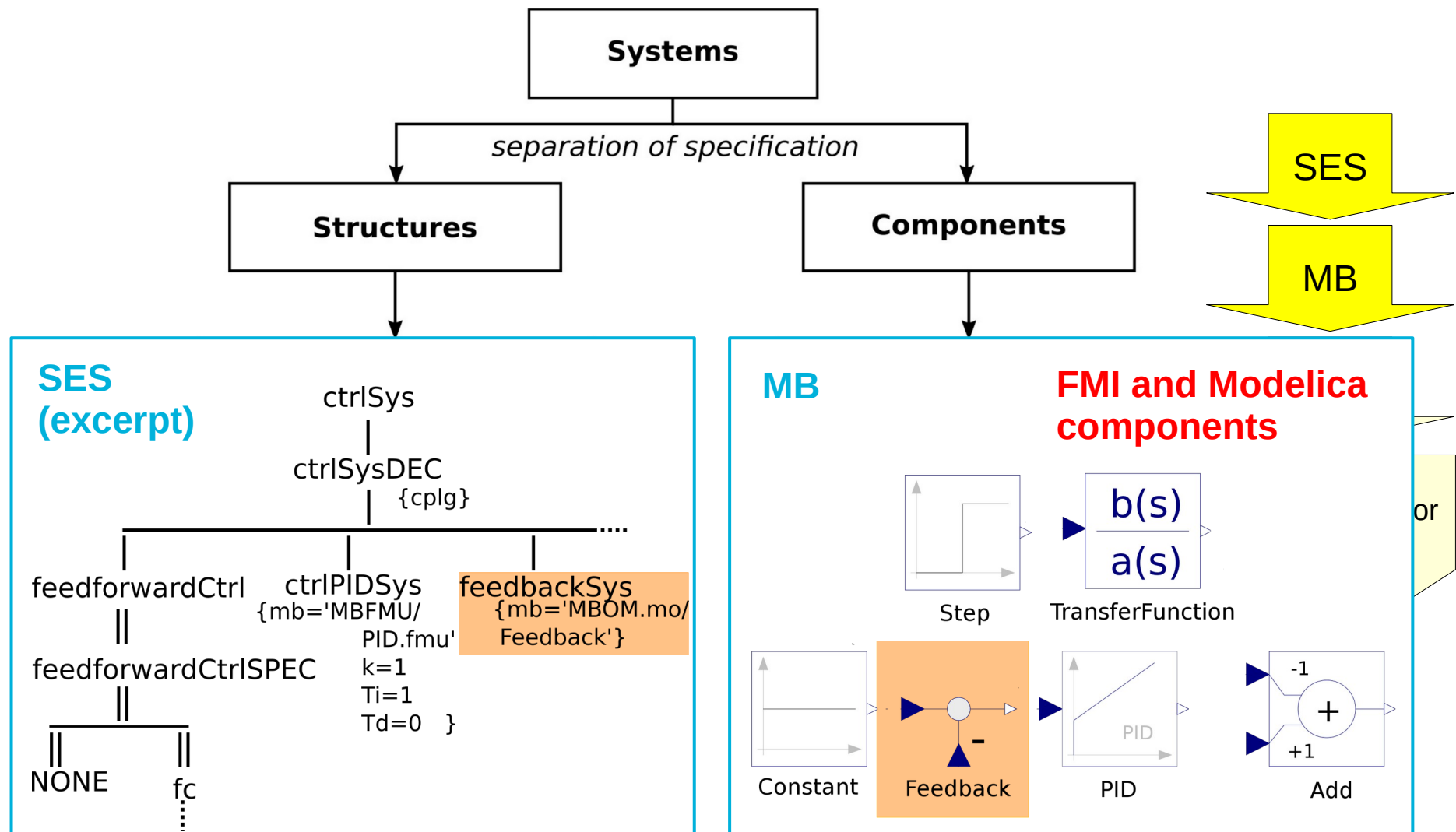


# SES/MB Approach with FMI Components





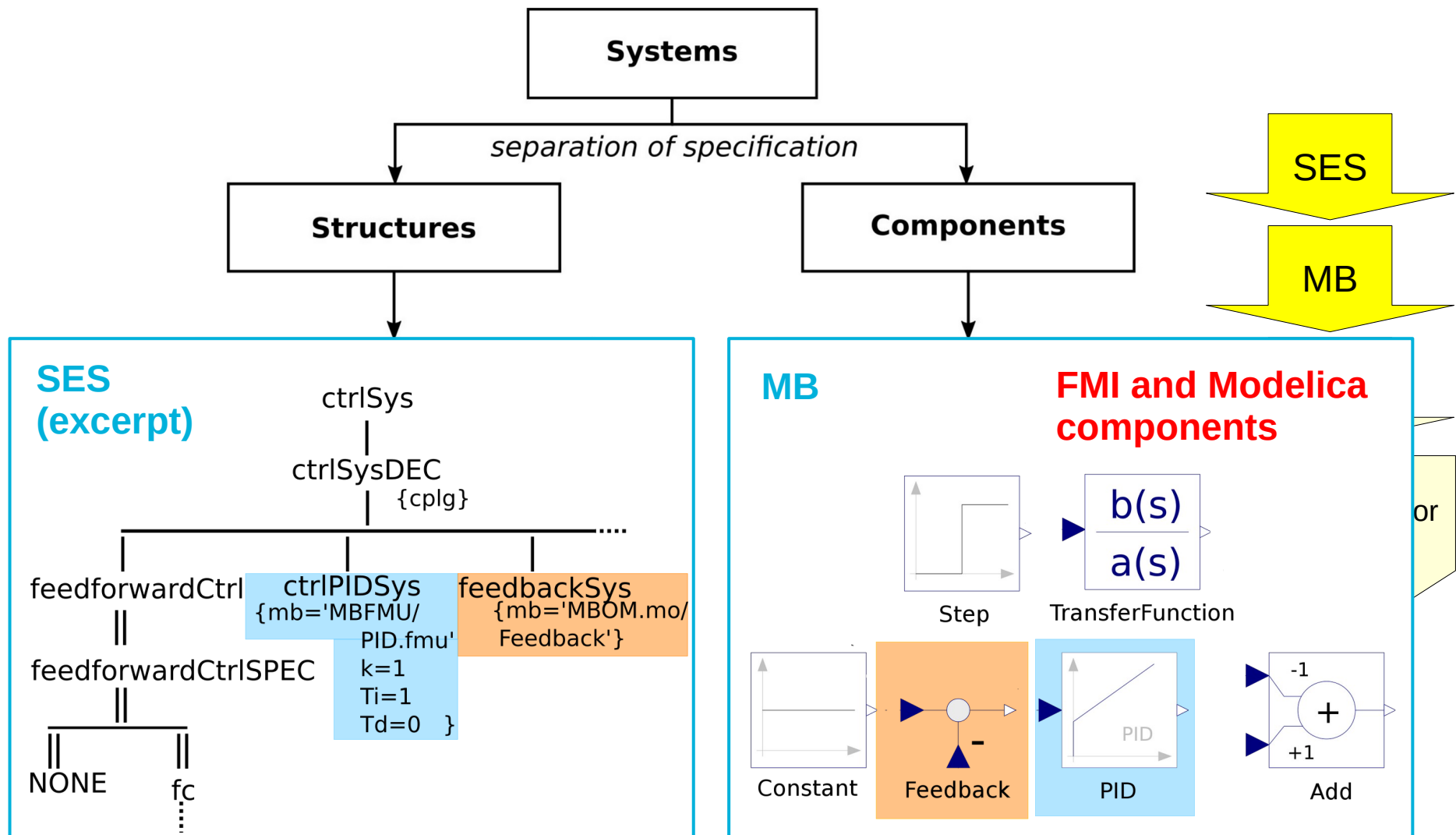
# SES/MB Approach with FMI Components





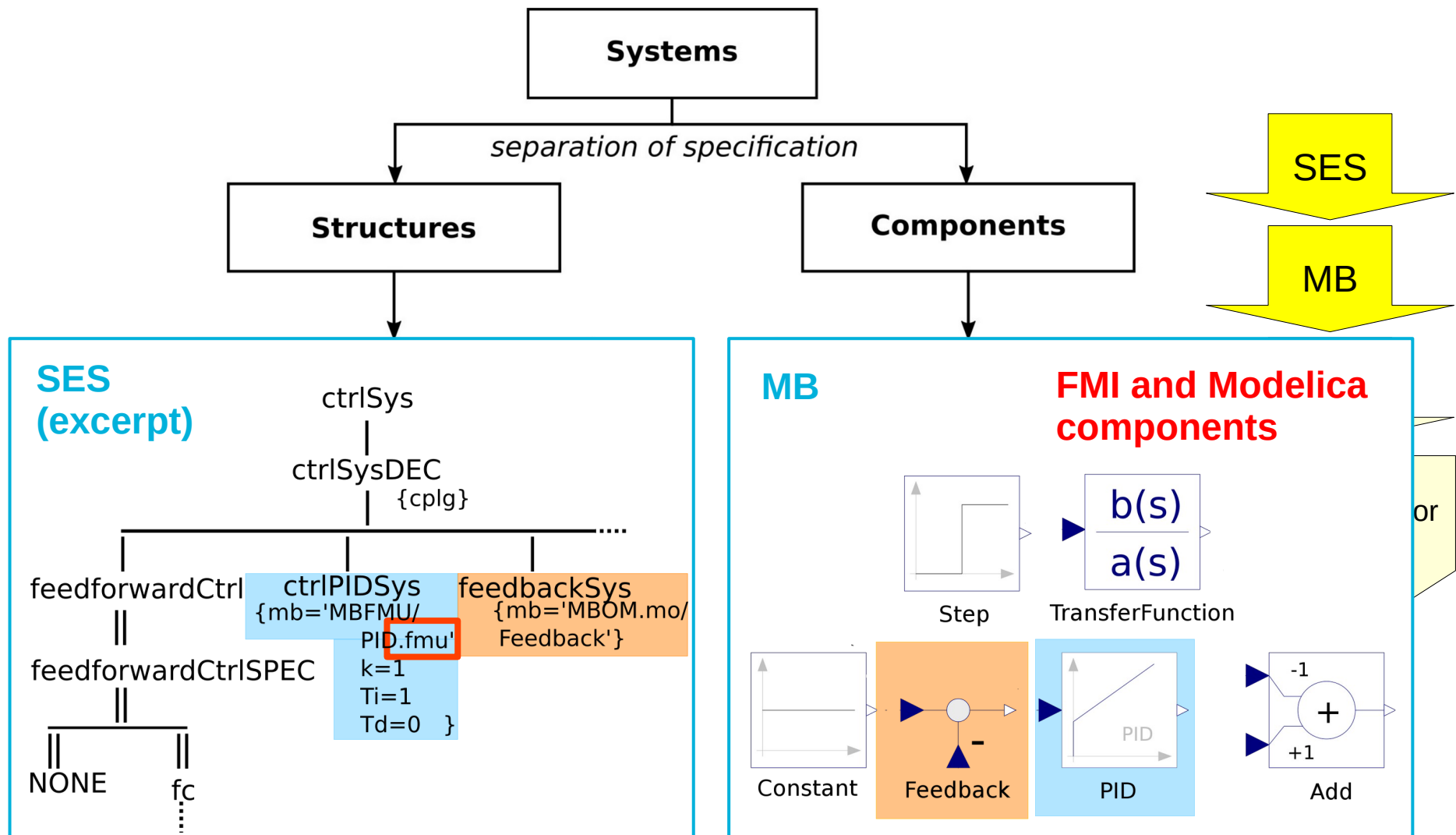


# SES/MB Approach with FMI Components





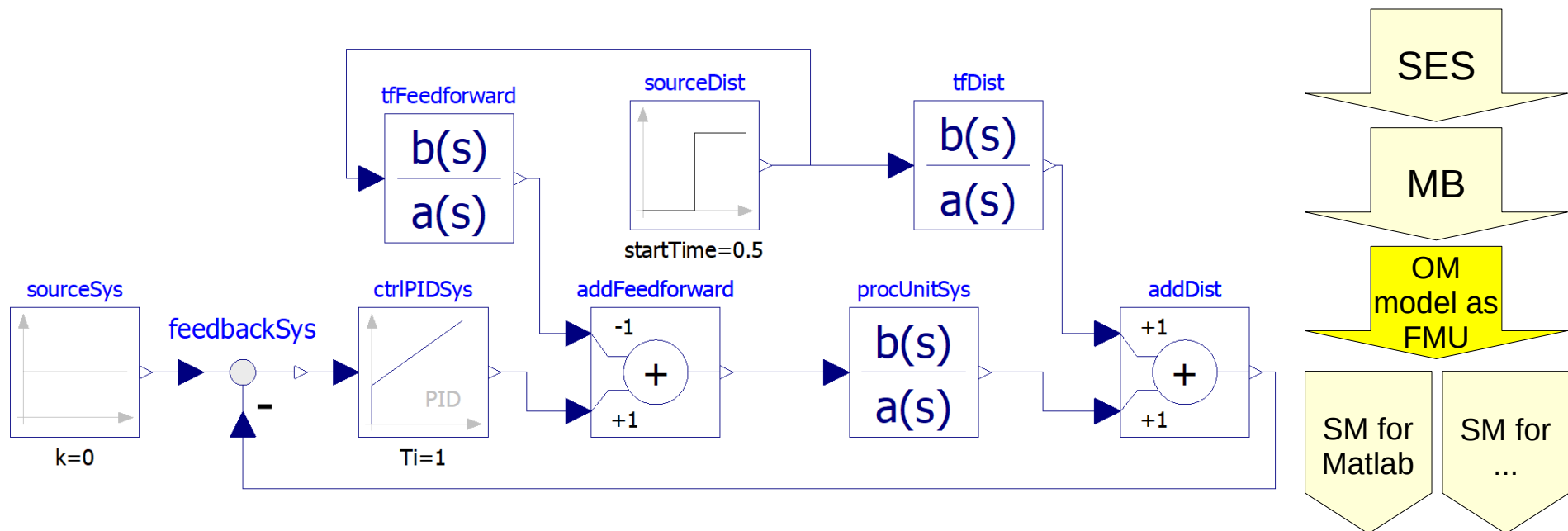
# SES/MB Approach with FMI Components





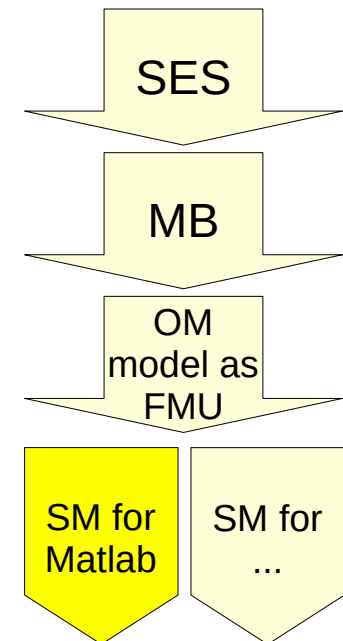
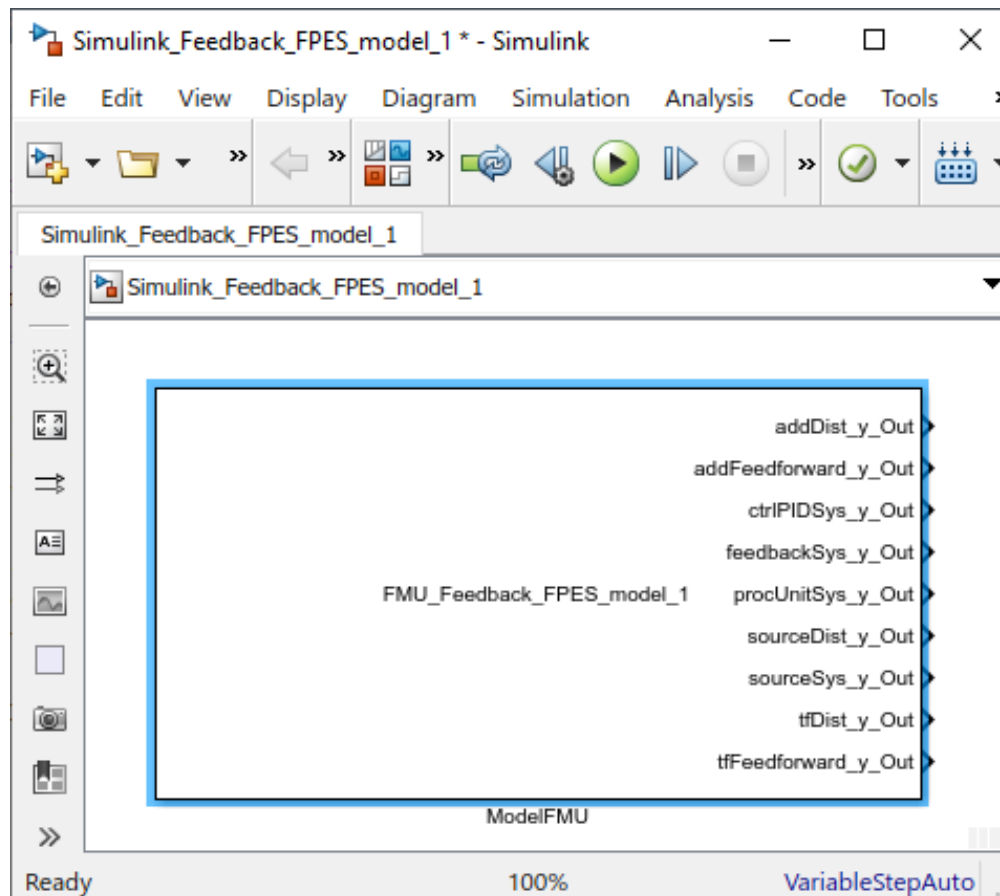
# Generated Model in OpenModelica

## → Export as one Model FMU





## Case Study: Model FMU Imported in MATLAB/Simulink



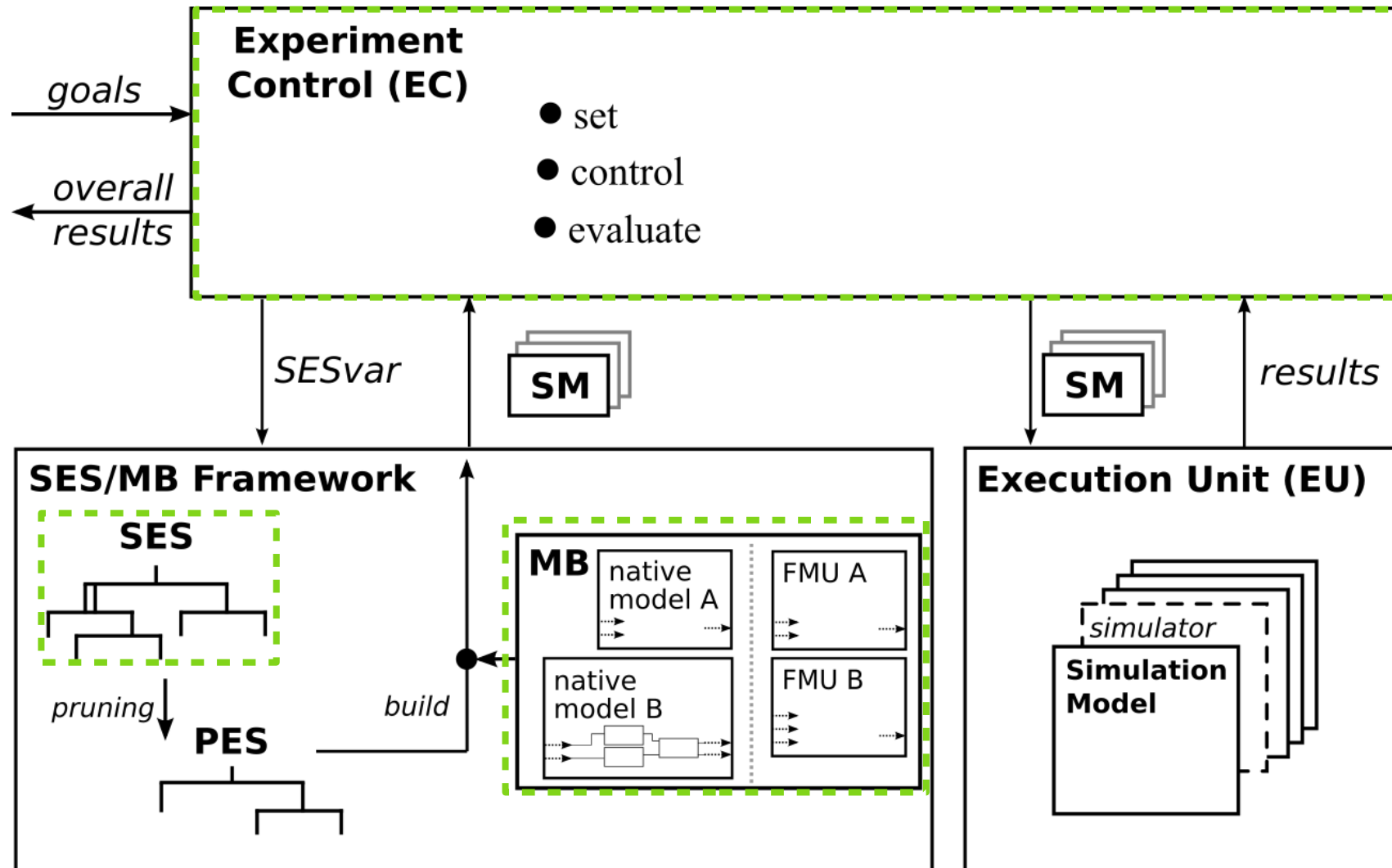


## Outline

1. Introduction
2. The case study
3. Basics of SES/MB based modeling
4. Practical modeling: implementation of an SES
5. Model selection and model generation
6. Organization of a simulator-independent MB
- 7. Full automation of simulation experiments**  
(H. Folkerts)
8. Conclusion

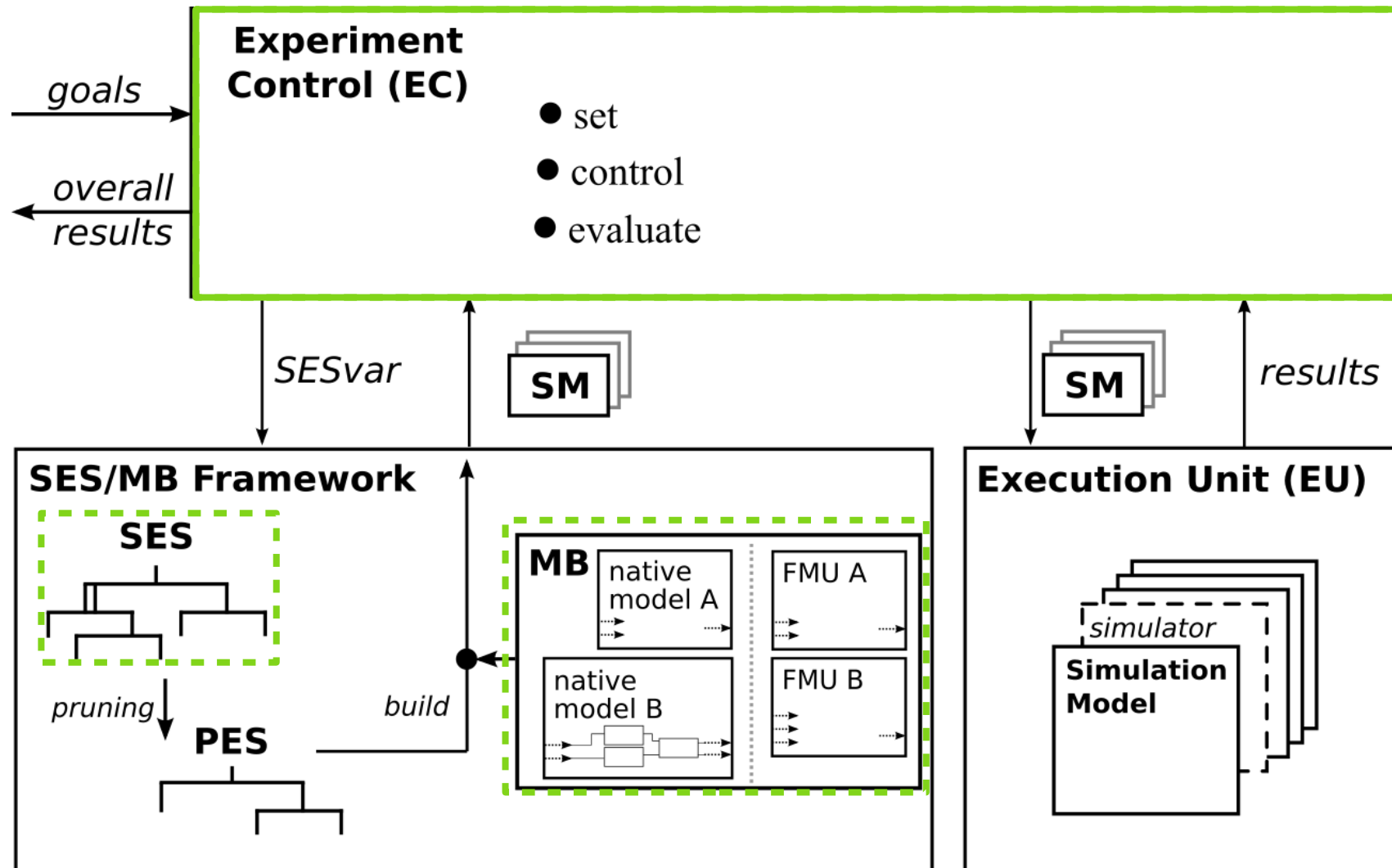


## Extended SES/MB Architecture (Native and FMI)



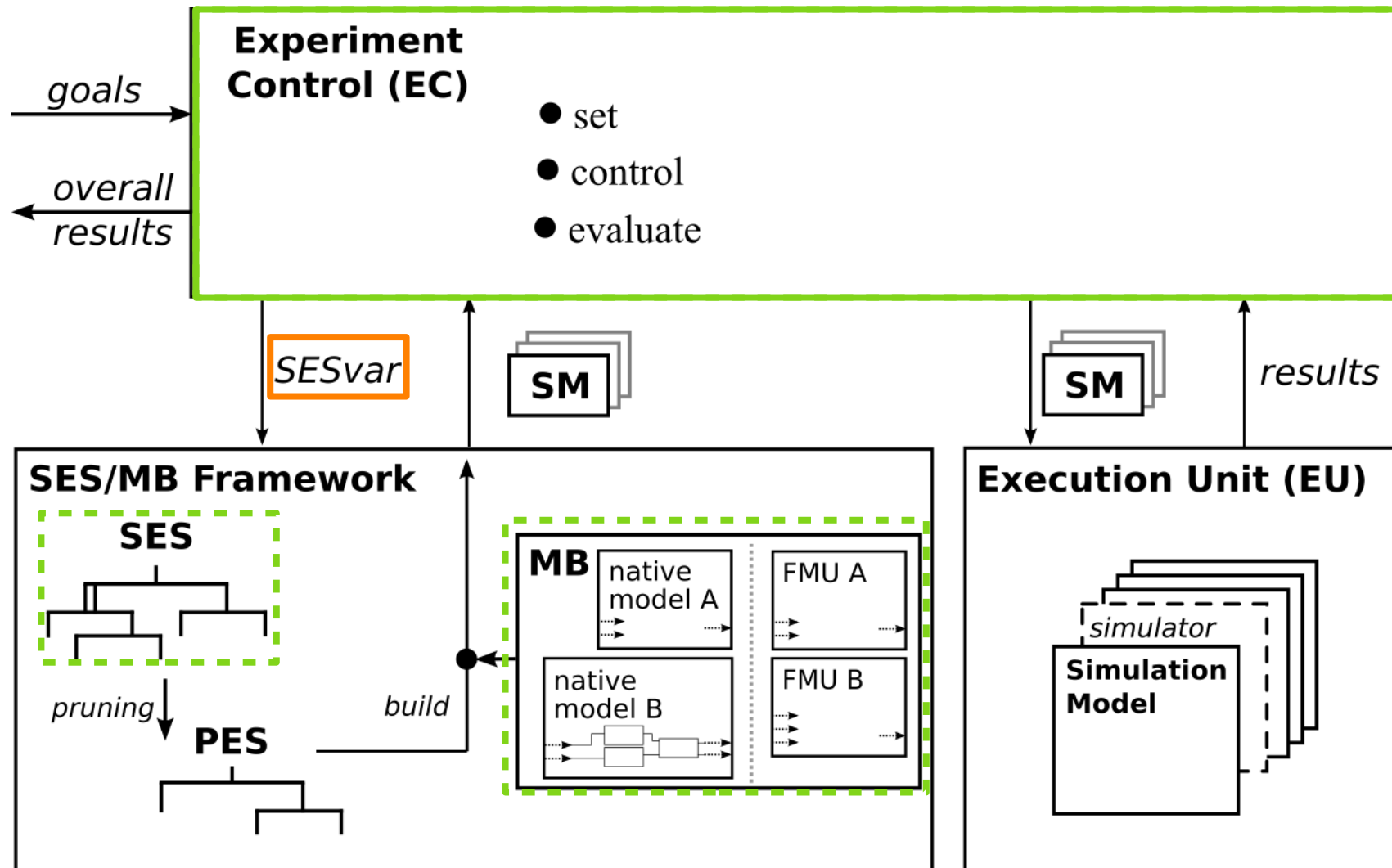


## Extended SES/MB Architecture (Native and FMI)





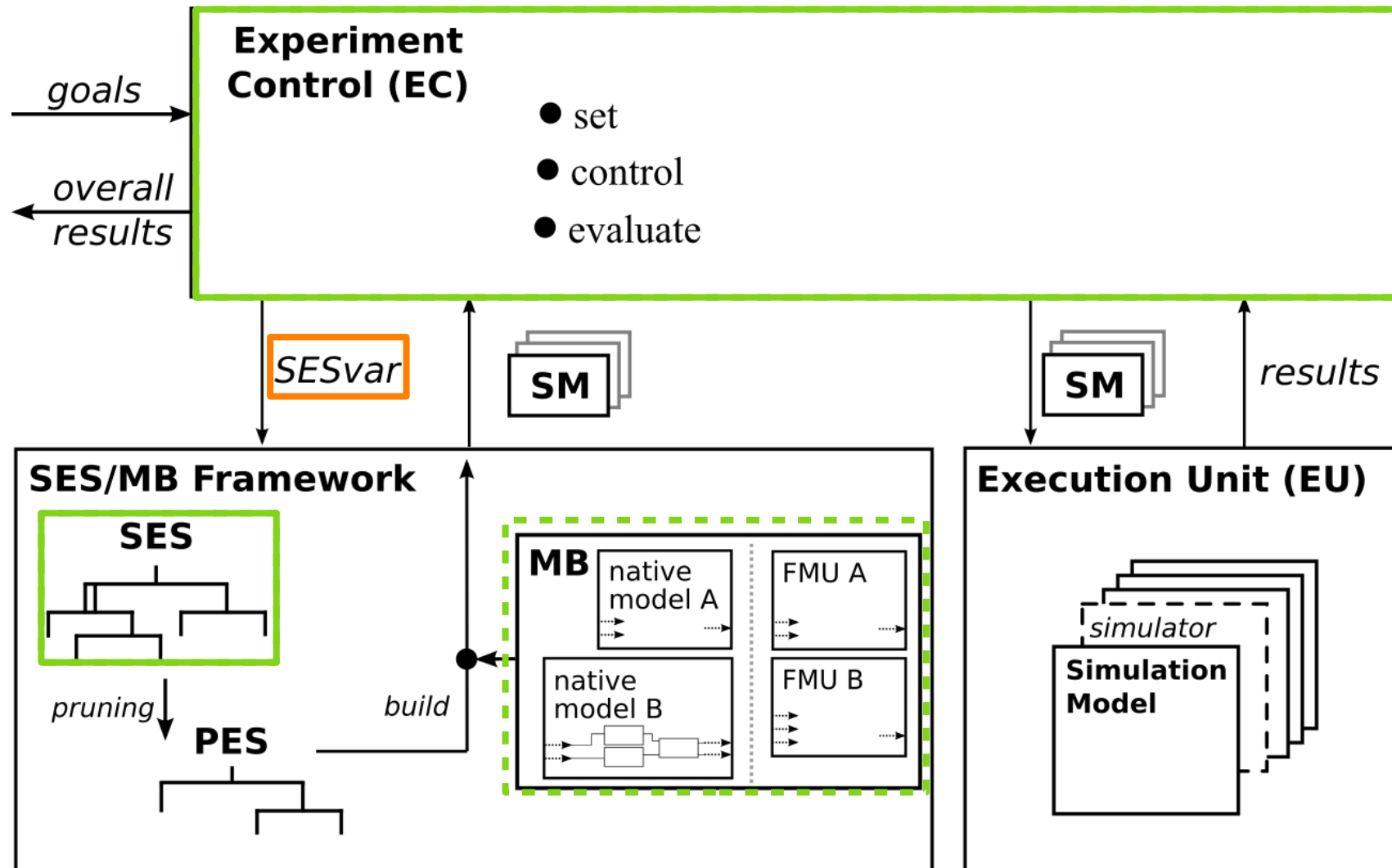
## Extended SES/MB Architecture (Native and FMI)





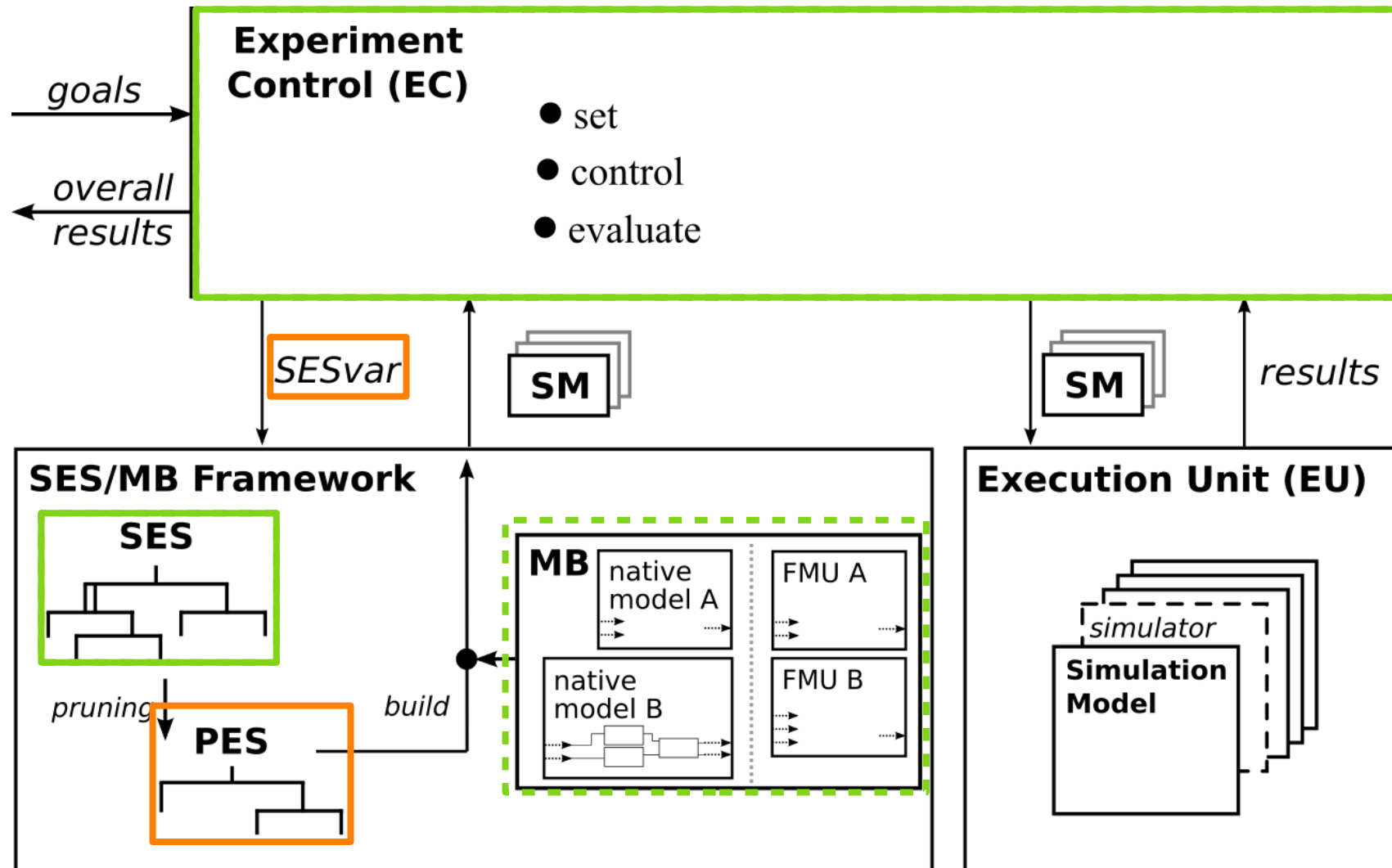


## Extended SES/MB Architecture (Native and FMI)



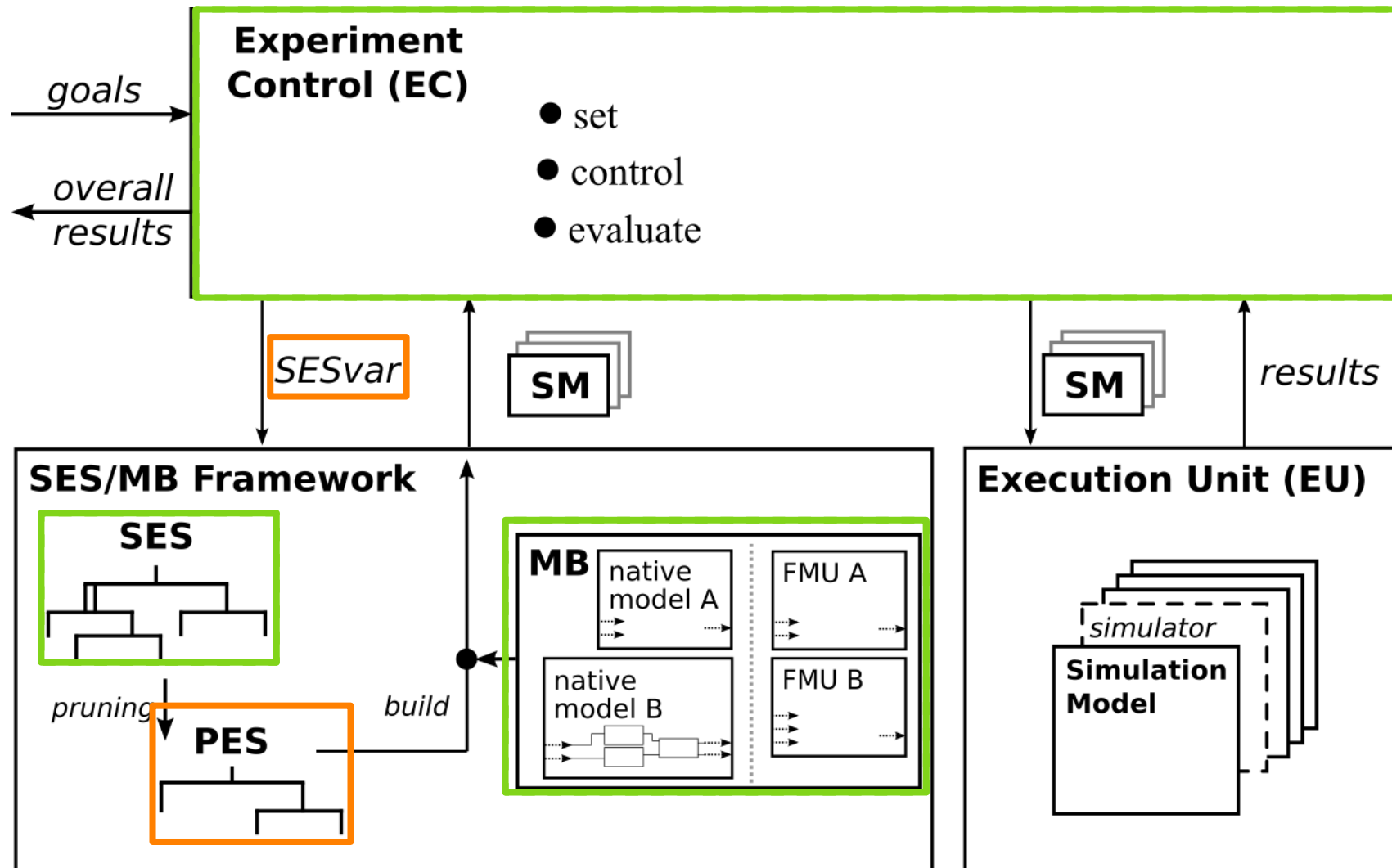


## Extended SES/MB Architecture (Native and FMI)



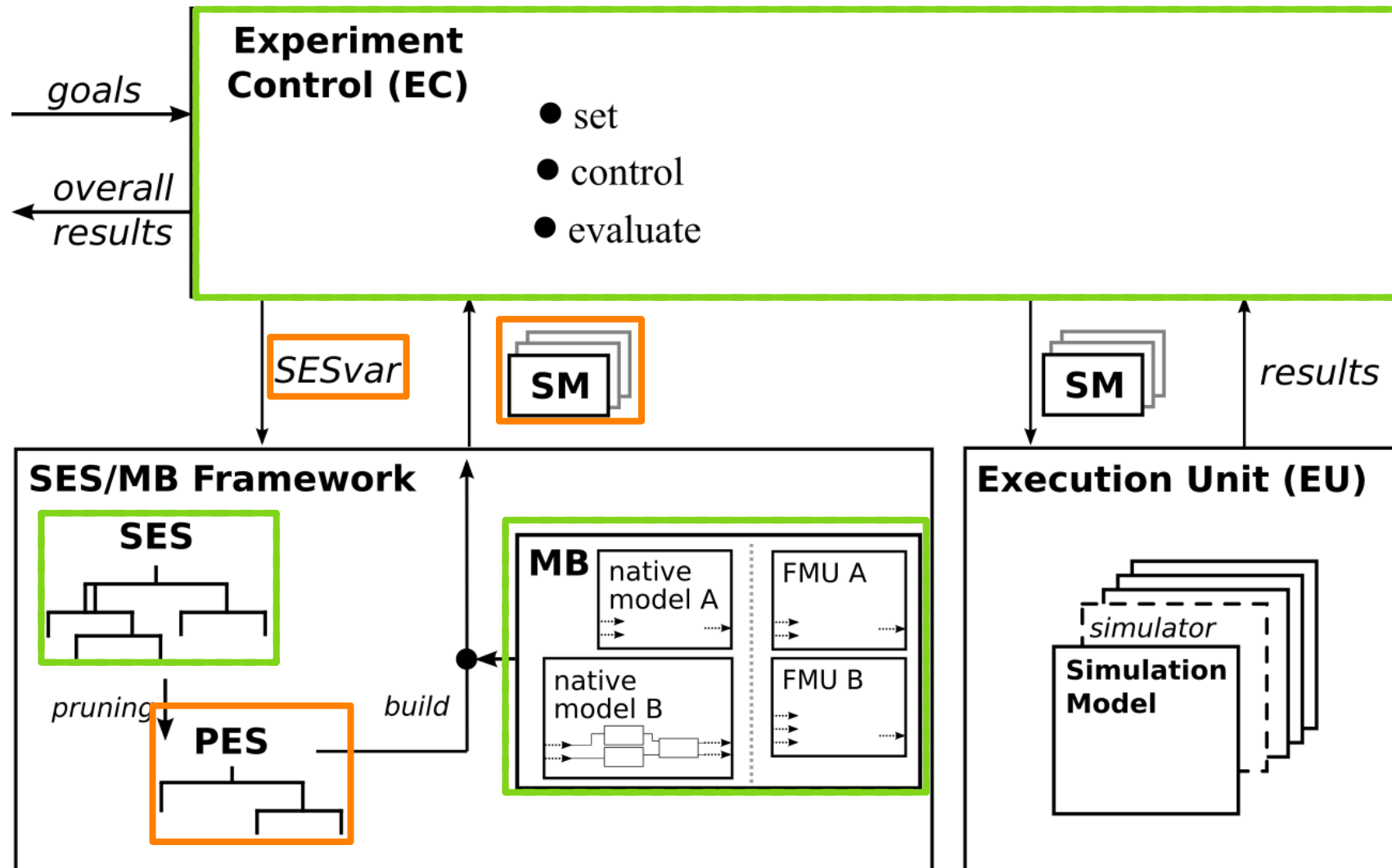


## Extended SES/MB Architecture (Native and FMI)



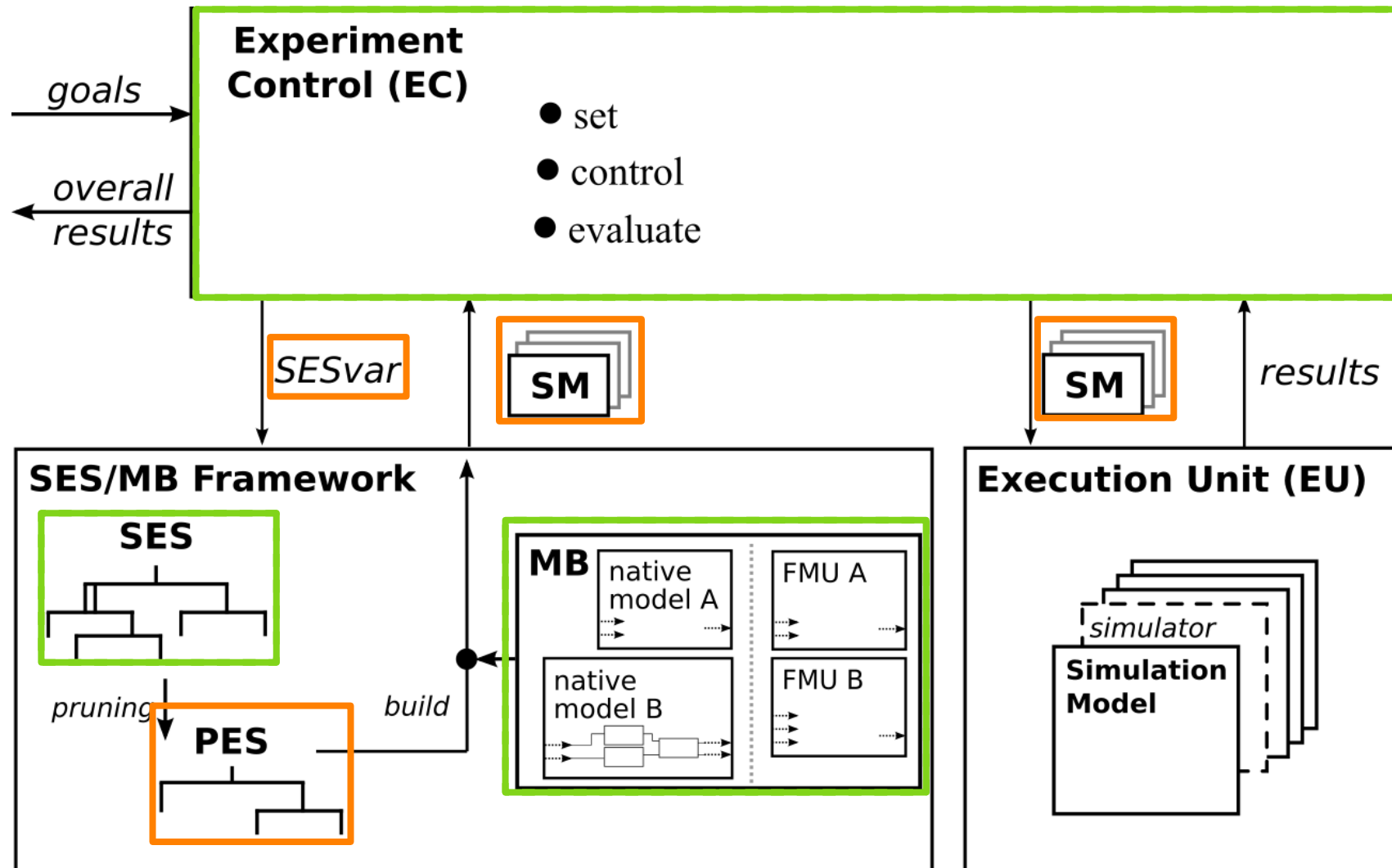


## Extended SES/MB Architecture (Native and FMI)



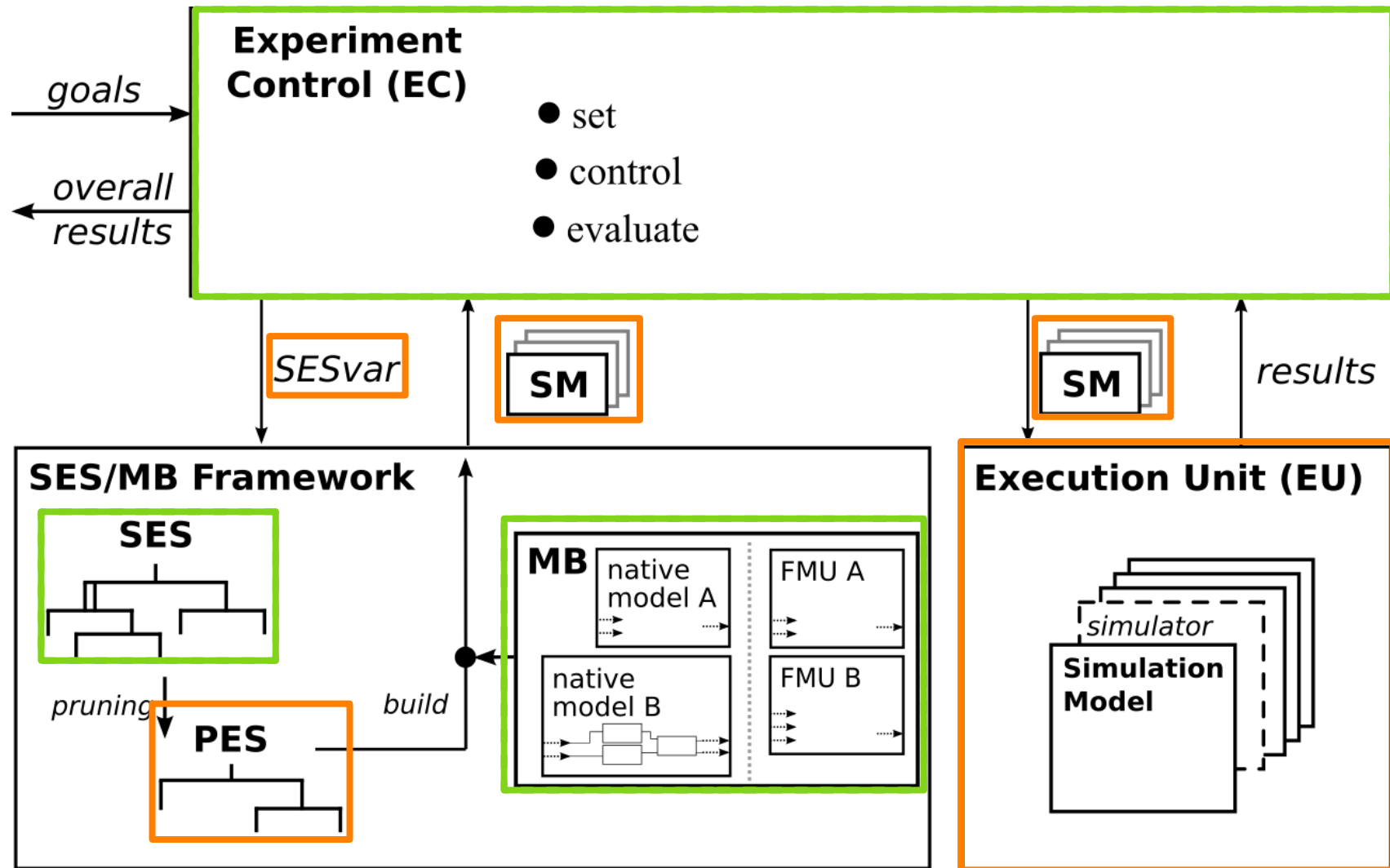


## Extended SES/MB Architecture (Native and FMI)



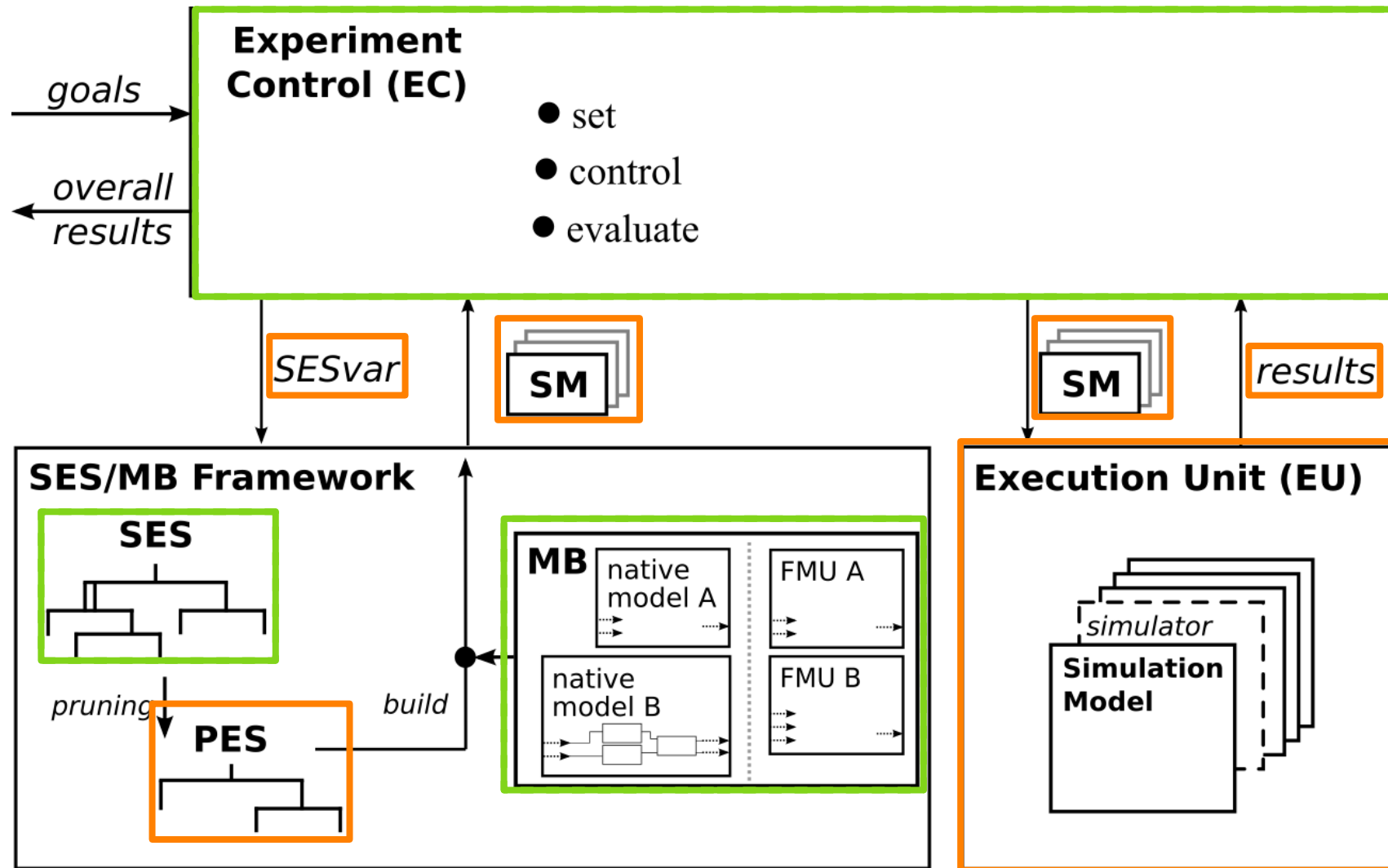


## Extended SES/MB Architecture (Native and FMI)



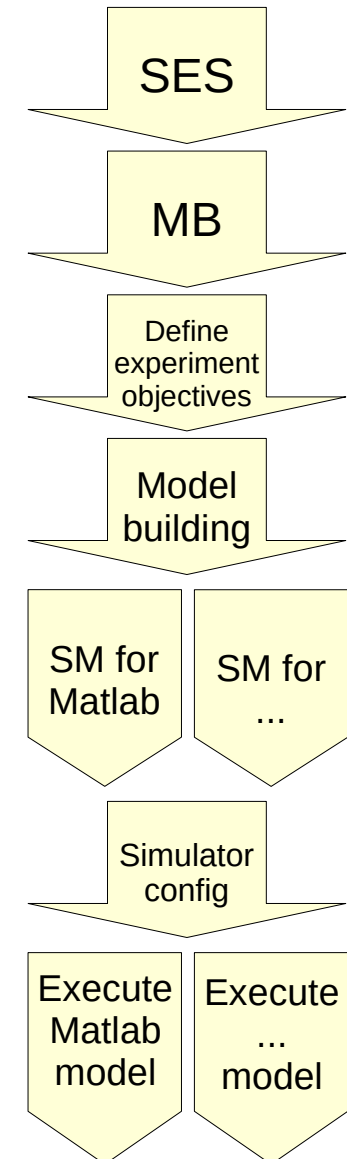
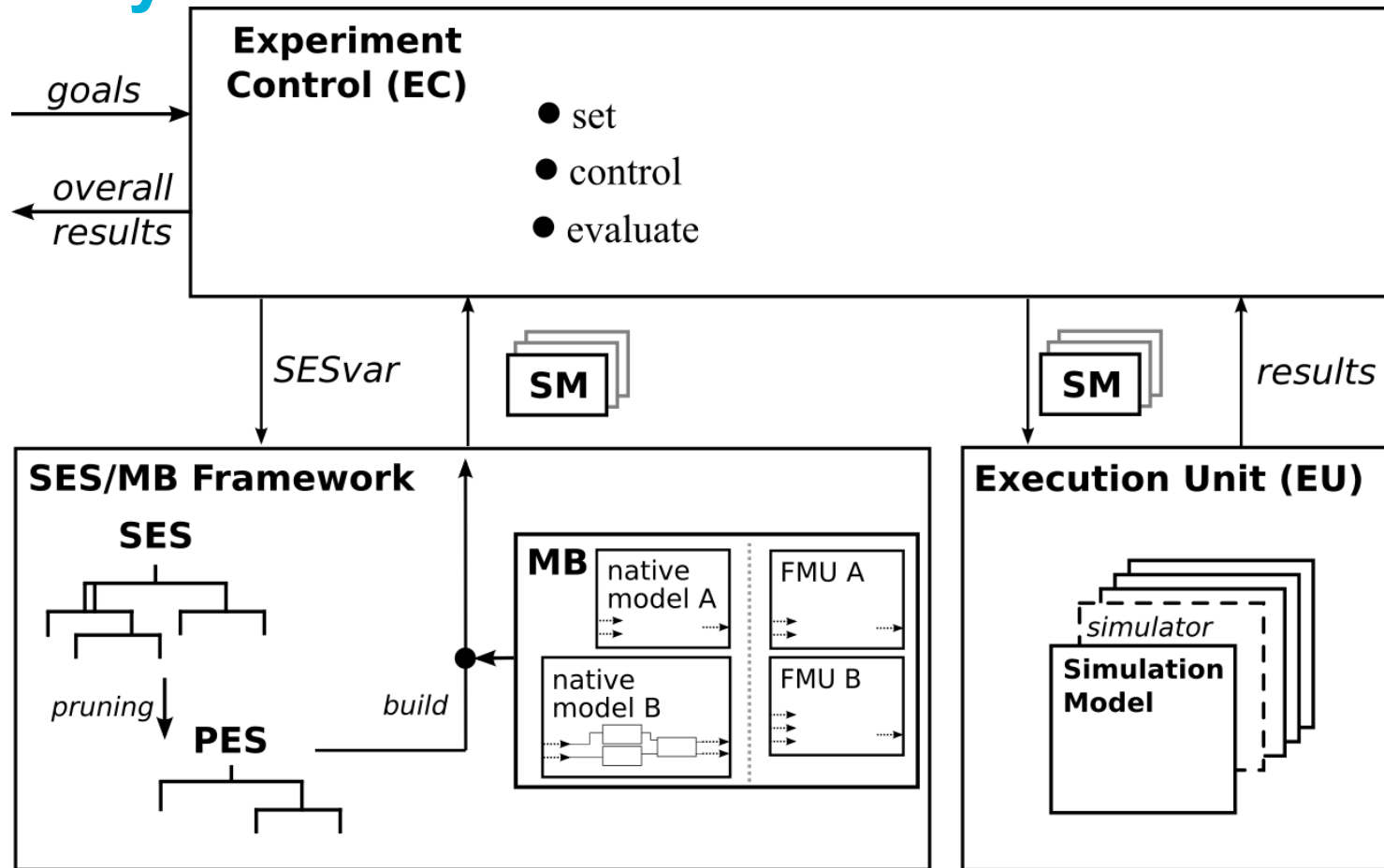


## Extended SES/MB Architecture (Native and FMI)





# Extended SES/MB Architecture Python Toolset

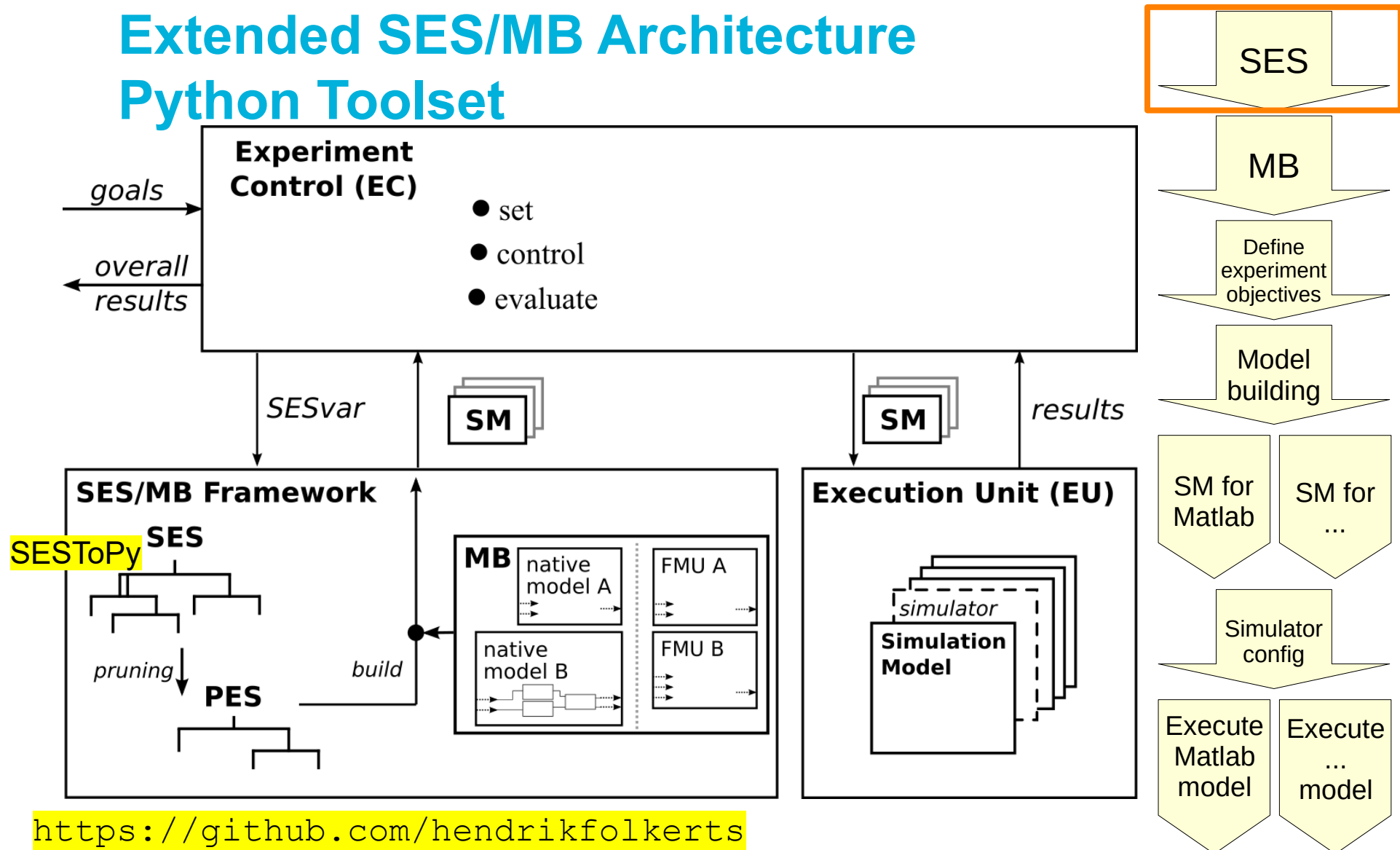


<https://github.com/hendrikfolkerts>



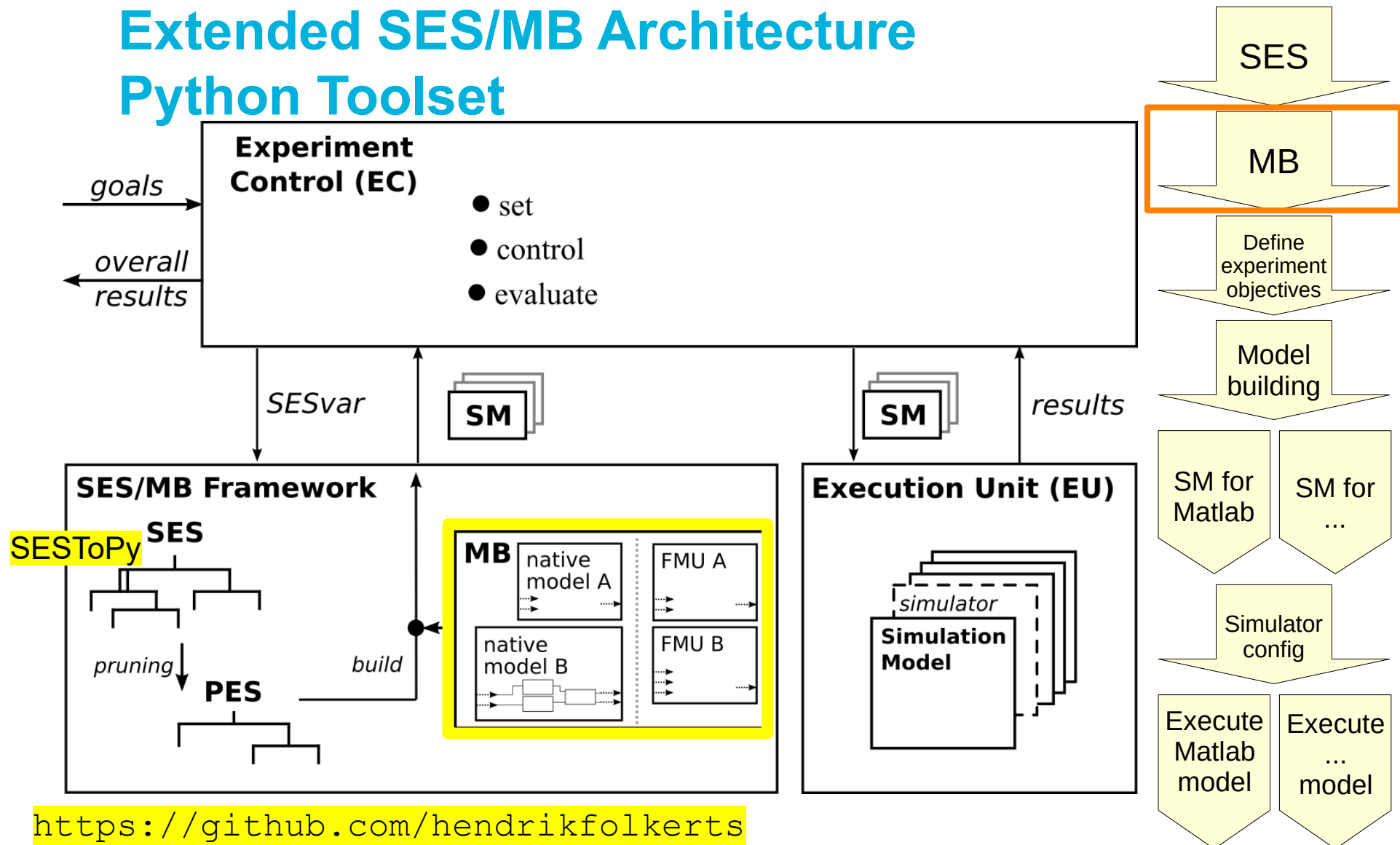


# Extended SES/MB Architecture Python Toolset



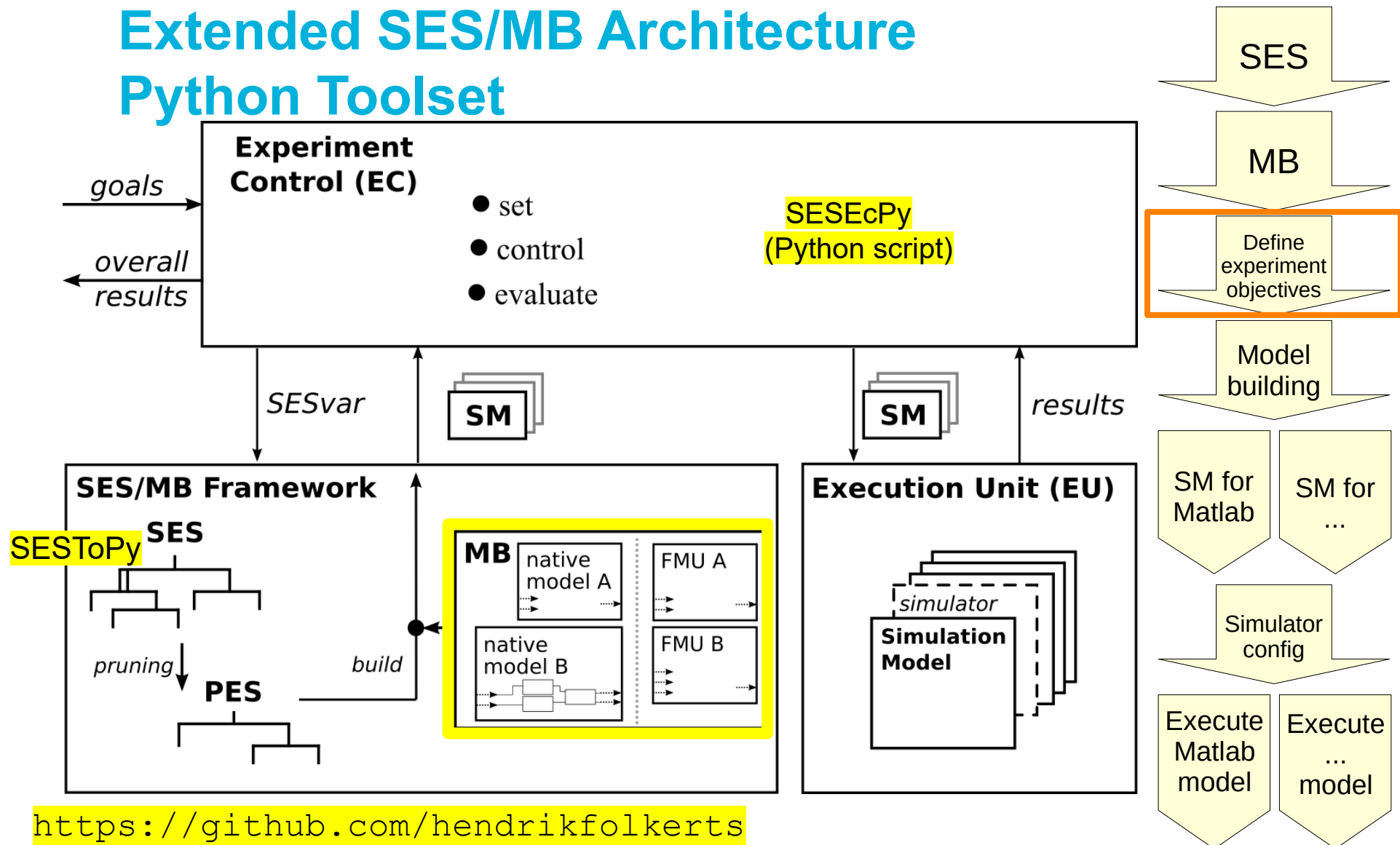


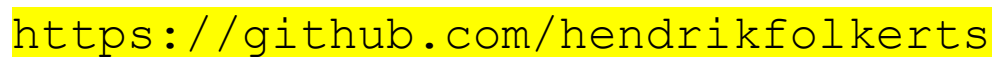
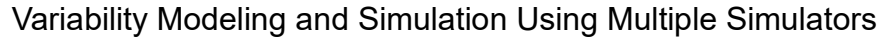
# Extended SES/MB Architecture Python Toolset





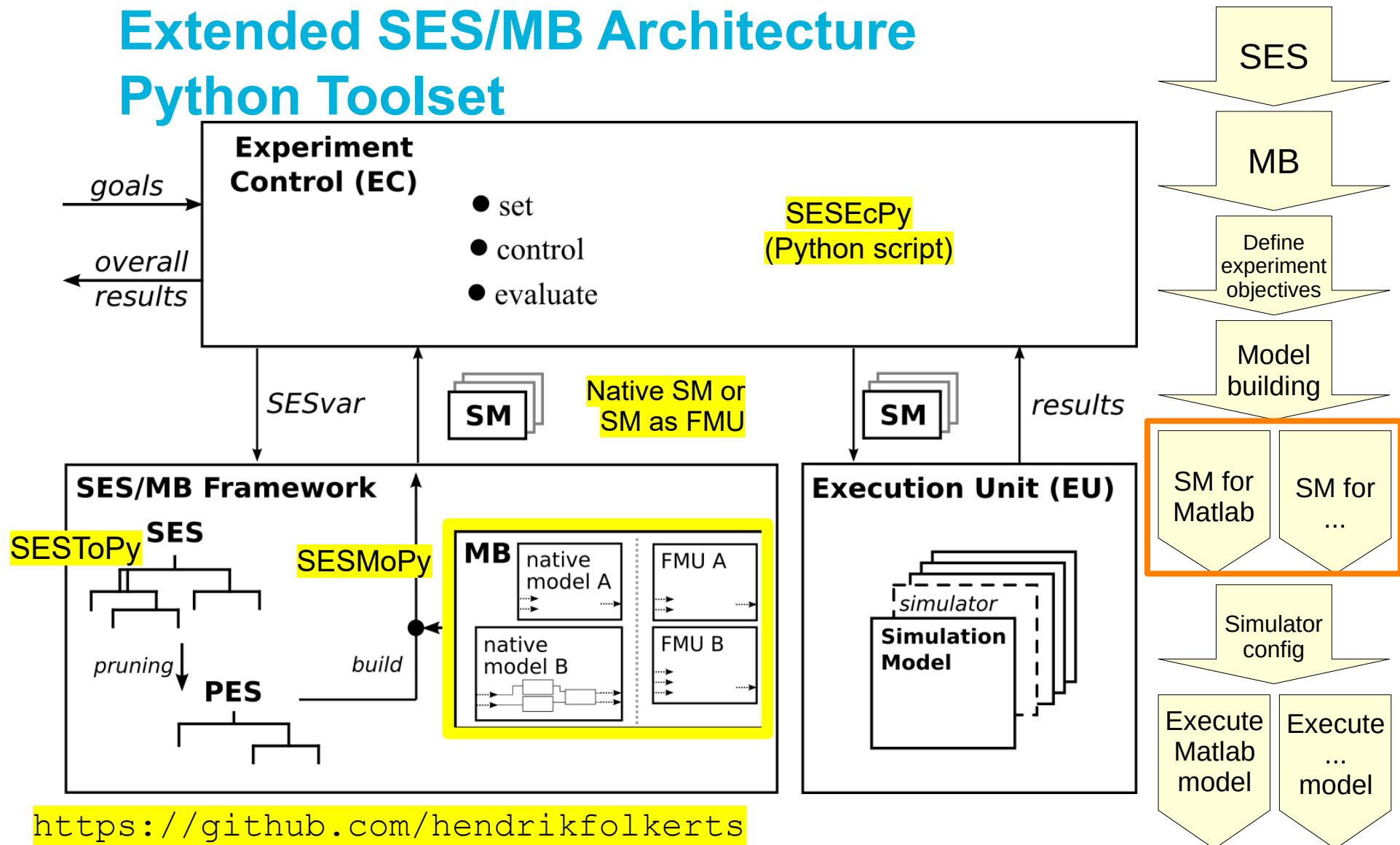
# Extended SES/MB Architecture Python Toolset





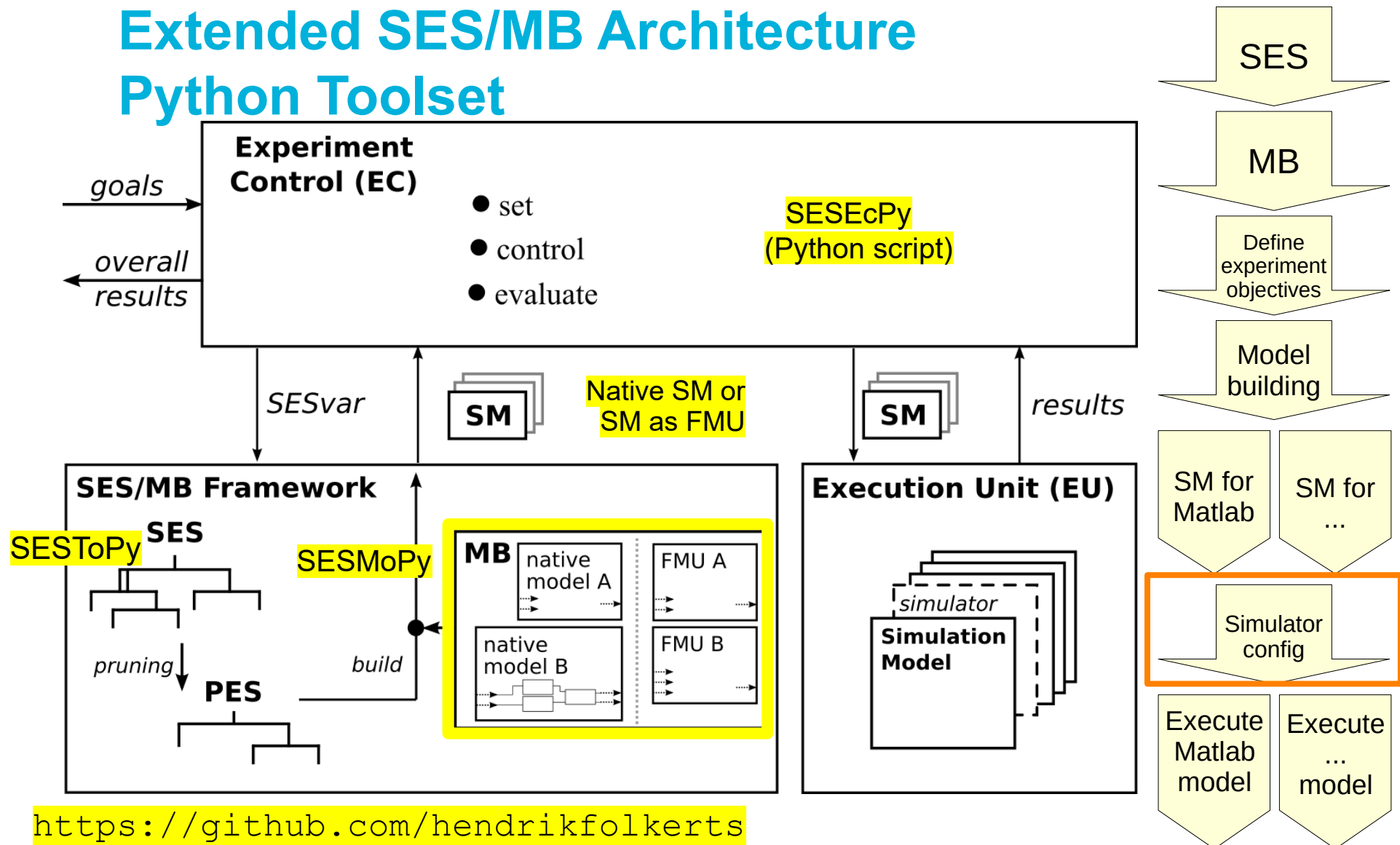


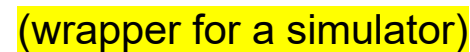
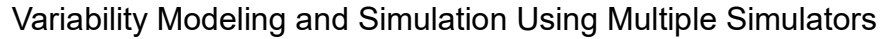
# Extended SES/MB Architecture Python Toolset





# Extended SES/MB Architecture Python Toolset

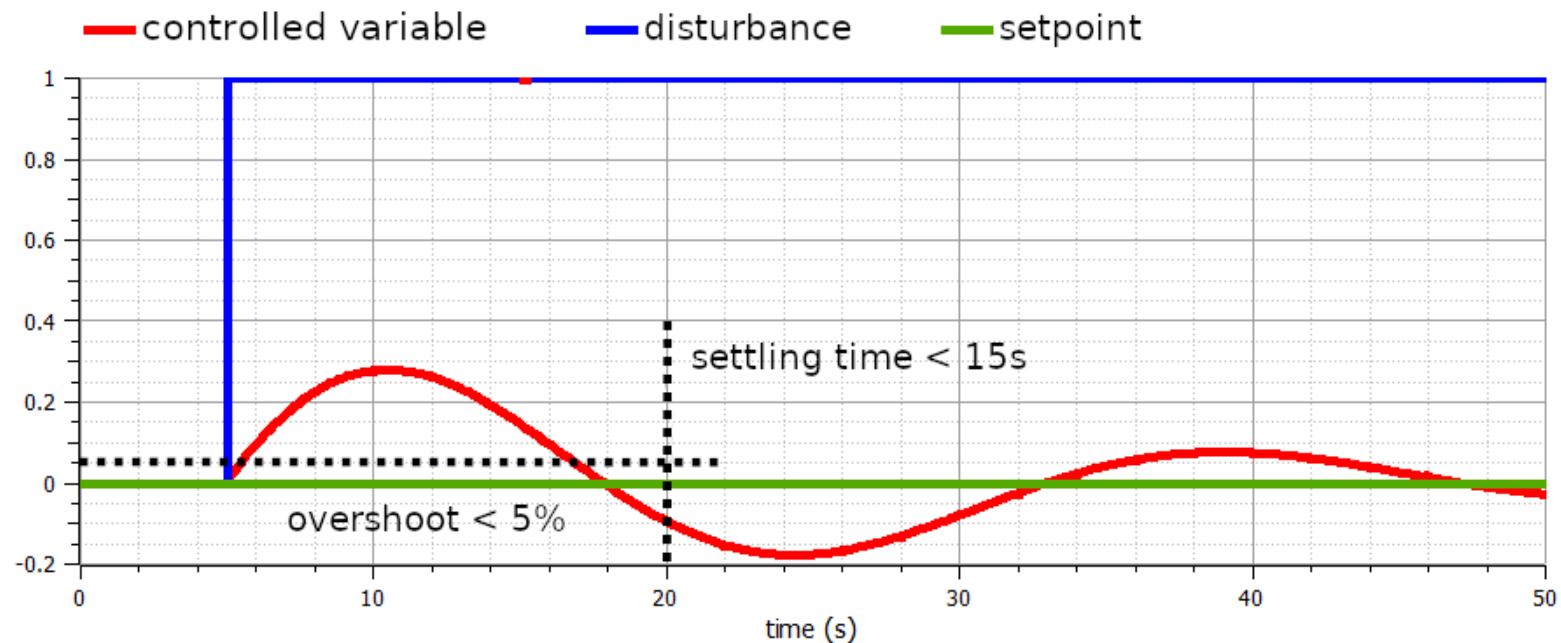
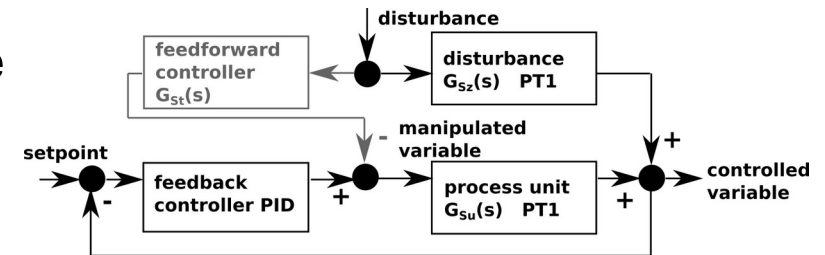






## Automation of Case Study: Control Goals

- Goal for the control after a disturbance
  - Overshoot < 5%
  - Settling time < 15s







## Case Study: Experimentation Steps



## Case Study: Experimentation Steps

- **Code in Experiment Control**



## Case Study: Experimentation Steps

- **Code in Experiment Control**
  - **Try without a feedforward control:**
    - `feedforward=0` simulate with PID:  $k=1$ ,  $T_i=1$ ,  $T_d=0$
    - `feedforward=0` simulate with PID:  $k=5$ ,  $T_i=0.5$ ,  $T_d=0$



## Case Study: Experimentation Steps

- **Code in Experiment Control**
  - **Try without a feedforward control:**
    - `feedforward=0` simulate with PID:  $k=1$ ,  $T_i=1$ ,  $T_d=0$
    - `feedforward=0` simulate with PID:  $k=5$ ,  $T_i=0.5$ ,  $T_d=0$
  - **If the goals are reached with one of these configurations:**
    - Return PID configuration as overall result



## Case Study: Experimentation Steps

- **Code in Experiment Control**
  - **Try without a feedforward control:**
    - `feedforward=0` simulate with PID:  $k=1$ ,  $T_i=1$ ,  $T_d=0$
    - `feedforward=0` simulate with PID:  $k=5$ ,  $T_i=0.5$ ,  $T_d=0$
  - **If the goals are reached with one of these configurations:**
    - Return PID configuration as overall result
  - **Else try with a feedforward control:**
    - `feedforward=1` simulate with both PID configurations



## Case Study: Experimentation Steps

- **Code in Experiment Control**

- **Try without a feedforward control:**
  - `feedforward=0` simulate with PID:  $k=1$ ,  $T_i=1$ ,  $T_d=0$
  - `feedforward=0` simulate with PID:  $k=5$ ,  $T_i=0.5$ ,  $T_d=0$
- **If the goals are reached with one of these configurations:**
  - Return PID configuration as overall result
- **Else try with a feedforward control:**
  - `feedforward=1` simulate with both PID configurations
- **If the goals are reached with one of these configurations:**
  - Return PID configuration as overall result



## Case Study: Experimentation Steps

- **Code in Experiment Control**

- **Try without a feedforward control:**
  - `feedforward=0` simulate with PID:  $k=1$ ,  $T_i=1$ ,  $T_d=0$
  - `feedforward=0` simulate with PID:  $k=5$ ,  $T_i=0.5$ ,  $T_d=0$
- **If the goals are reached with one of these configurations:**
  - Return PID configuration as overall result
- **Else try with a feedforward control:**
  - `feedforward=1` simulate with both PID configurations
- **If the goals are reached with one of these configurations:**
  - Return PID configuration as overall result
- **Else:**
  - Return goals cannot be reached with these configurations / parameters



## Case Study: Experimentation Steps

- **Code in Experiment Control**

- **Try without a feedforward control:**
  - `feedforward=0` simulate with PID: `k=1, Ti=1, Td=0`
  - `feedforward=0` simulate with PID: `k=5, Ti=0.5, Td=0`
- **If the goals are reached with one of these configurations:**
  - Return PID configuration as overall result
- **Else try with a feedforward control:**
  - `feedforward=1` simulate with both PID configurations
- **If the goals are reached with one of these configurations:**
  - Return PID configuration as overall result
- **Else:**
  - Return goals cannot be reached with these configurations / parameters

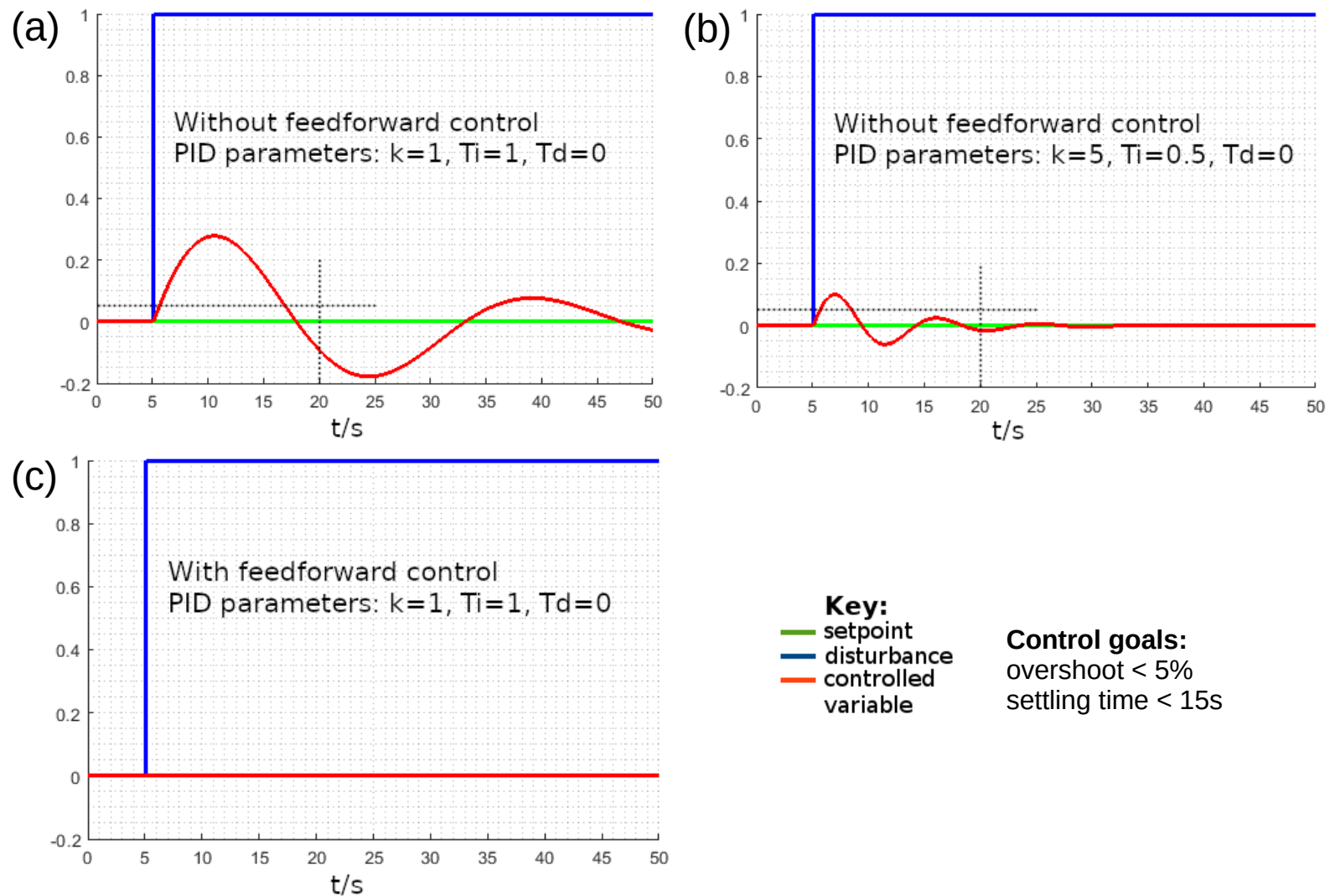
---

**Starting over with another simulator possible (model by model validation)**



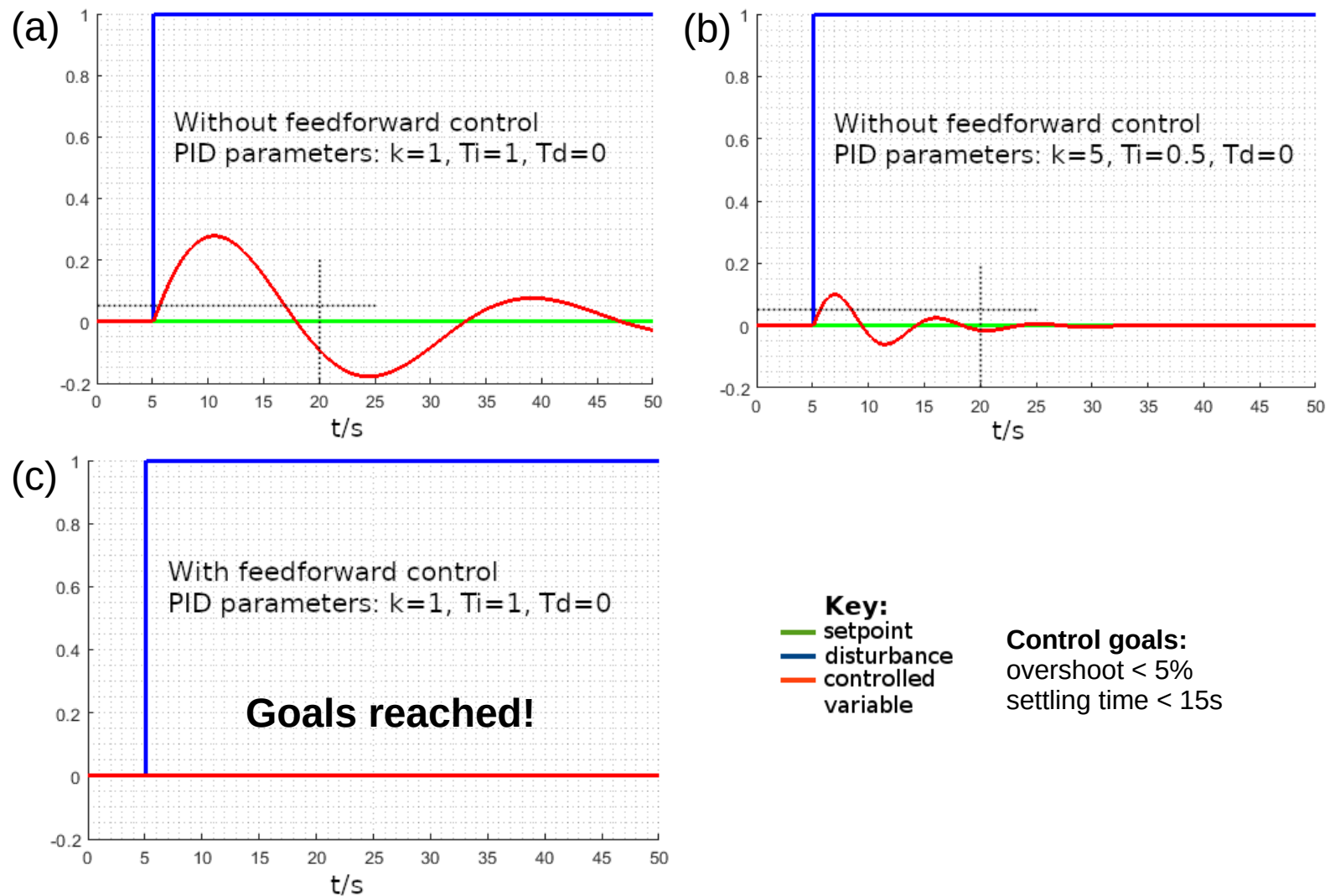


## Case Study: Simulation Results





## Case Study: Simulation Results





## Outline

1. Introduction
2. The case study
3. Basics of SES/MB based modeling
4. Practical modeling: implementation of an SES
5. Model selection and model generation
6. Organization of a simulator-independent MB
7. Full automation of simulation experiments
- 8. Conclusion** (T. Pawletta)



## Conclusion



## Conclusion

- **SES** supports **simulator-independent modeling** of model configurations regarding model structures and parameter settings



## Conclusion

- **SES** supports **simulator-independent modeling** of model configurations regarding model structures and parameter settings
- **MBs** are usually **simulator-specific**
  - No problem, if working in only one M&S environment
  - Difficult maintenance, if working with multiple simulators



## Conclusion

- **SES** supports **simulator-independent modeling** of model configurations regarding model structures and parameter settings
- **MBs** are usually **simulator-specific**
  - No problem, if working in only one M&S environment
  - Difficult maintenance, if working with multiple simulators
- Using FMI a **simulator-independent MB** is possible
  - Support for efficient model building for multiple simulators
  - But still problems for discrete event models



## Conclusion

- **SES** supports **simulator-independent modeling** of model configurations regarding model structures and parameter settings
- **MBs** are usually **simulator-specific**
  - No problem, if working in only one M&S environment
  - Difficult maintenance, if working with multiple simulators
- Using FMI a **simulator-independent MB** is possible
  - Support for efficient model building for multiple simulators
  - But still problems for discrete event models
- The Extended SES/MB Architecture supports a **full experiment automation** regarding defined design objectives (using multiple simulators)





## Remark: The Free MATLAB SES Toolbox

- Available: <https://github.com/cea-wismar>
- **SES Tbx for MATLAB**
  - Graphical SES editor
  - Function suite to work on SES (merge, check, prune, flatten,...)
  - Model builder (Simulink, Open-Modelica, Dymola, next MATLAB DEVS)
- **SESViewEI**
  - SES tree viewer can be connected



# Thank you!

Questions?

Sources for presented tools:

- <https://github.com/hendrikfolkerts>
- <https://github.com/cea-wismar>