USING CONSISTENCY PRESERVATION FOR MULTI-DOMAIN VARIABILITY MODELING

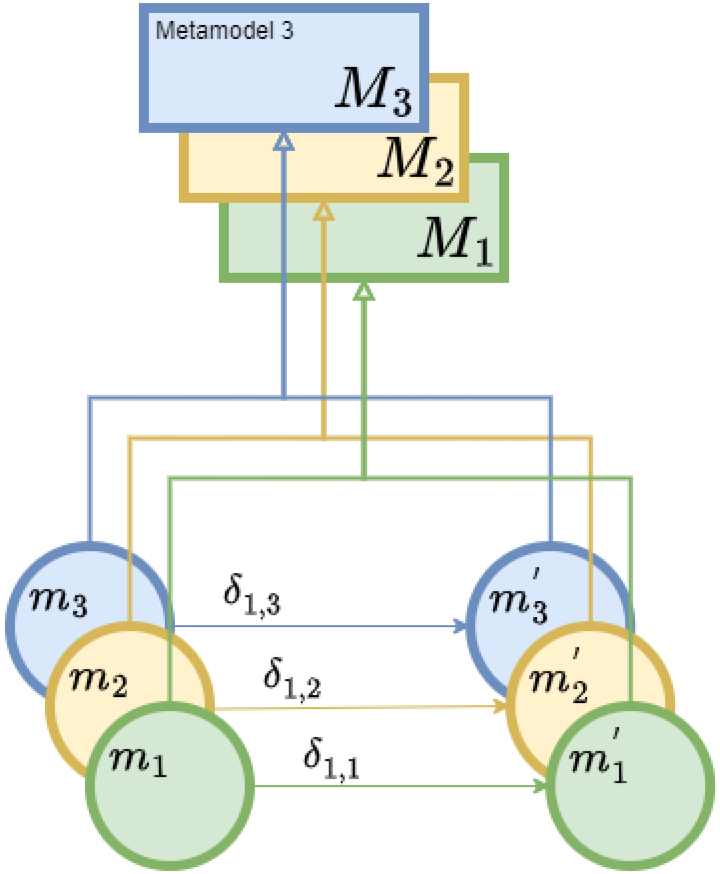
Dirk Neumann

KIT – KASTEL – TVA

dirk.neumann@kit.edu

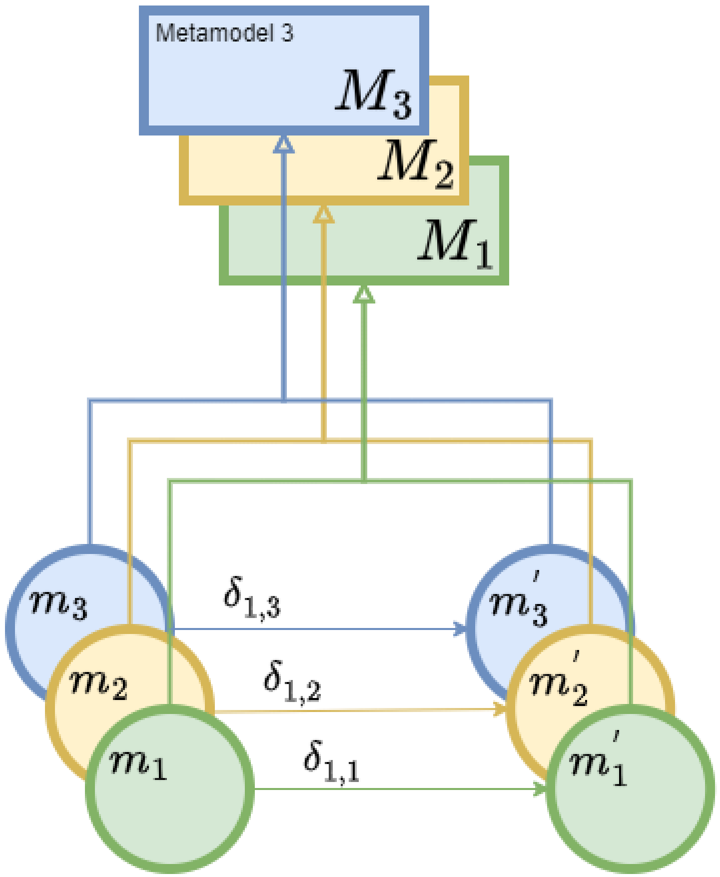
supervised by Ina Schaefer

CONTEXT – DELTA-ORIENTED VARIABILITY MODELING

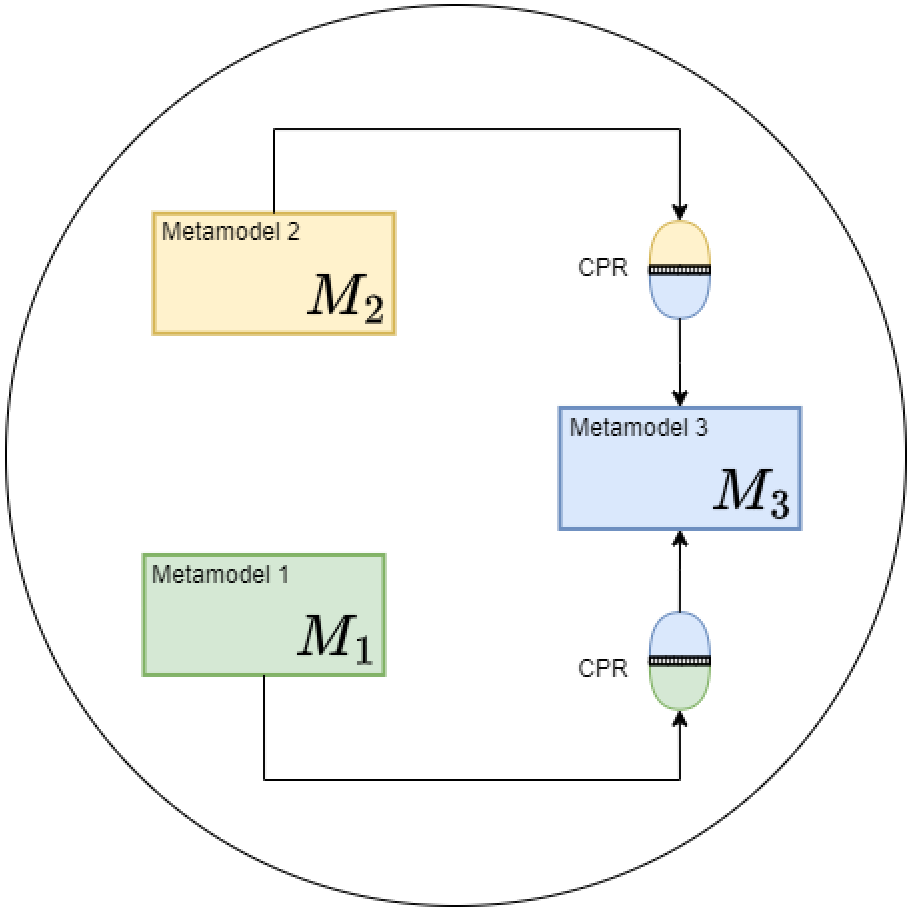


Delta-oriented variability
in a model-driven world

CONTEXT – CONSISTENCY PRESERVATION

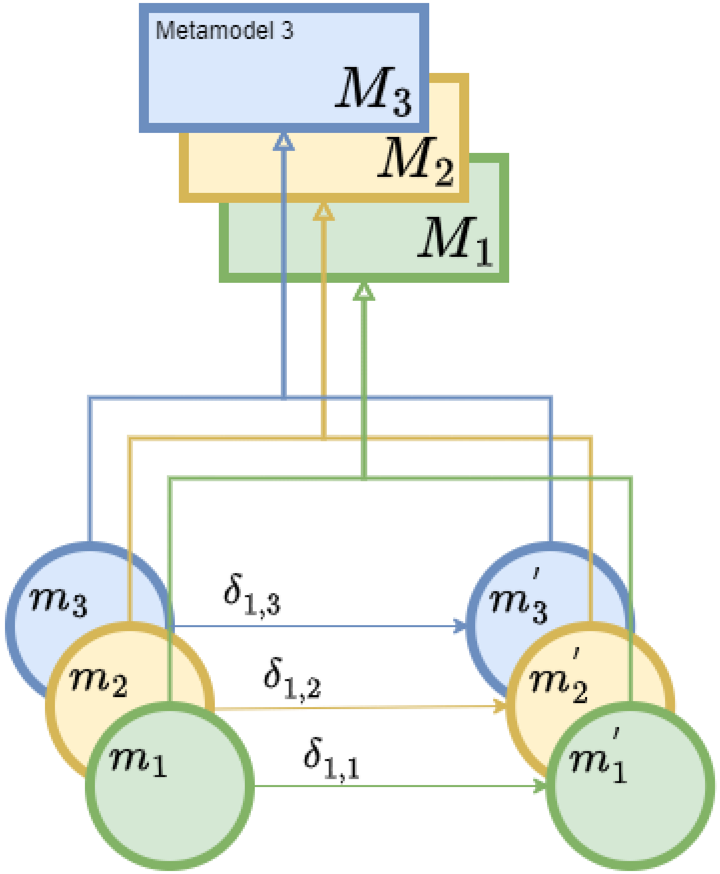


Delta-oriented variability
in a model-driven world



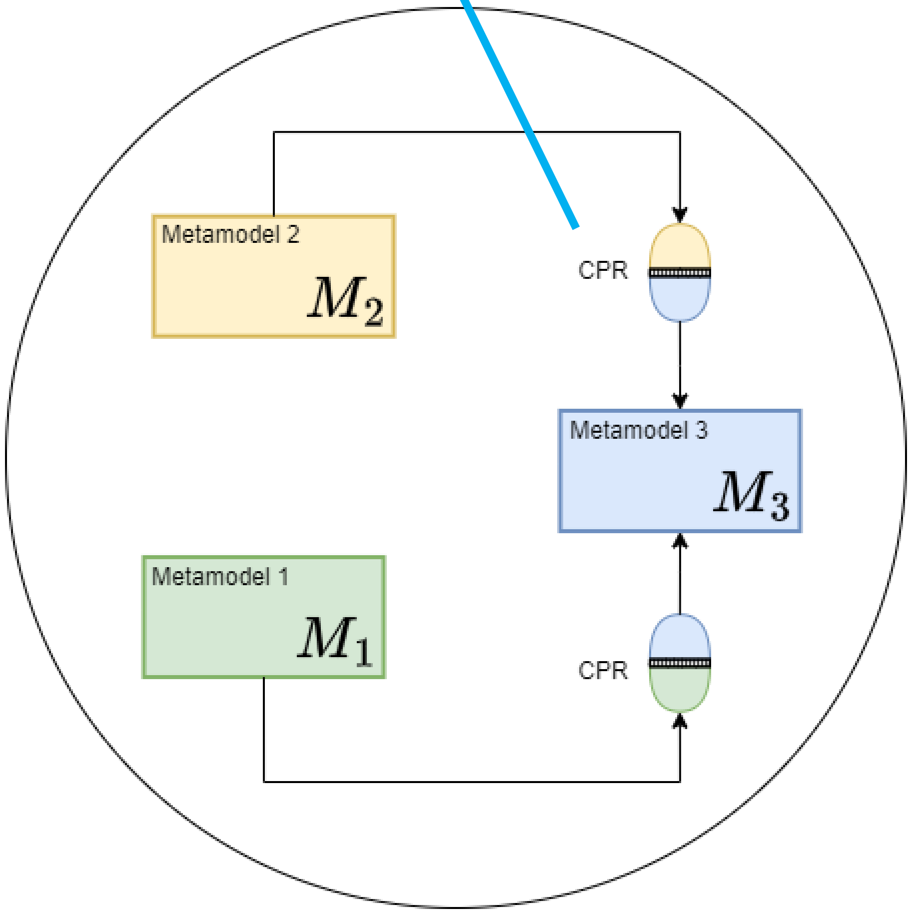
Consistency preservation
in a model-driven world

CONTEXT – CONSISTENCY PRESERVATION



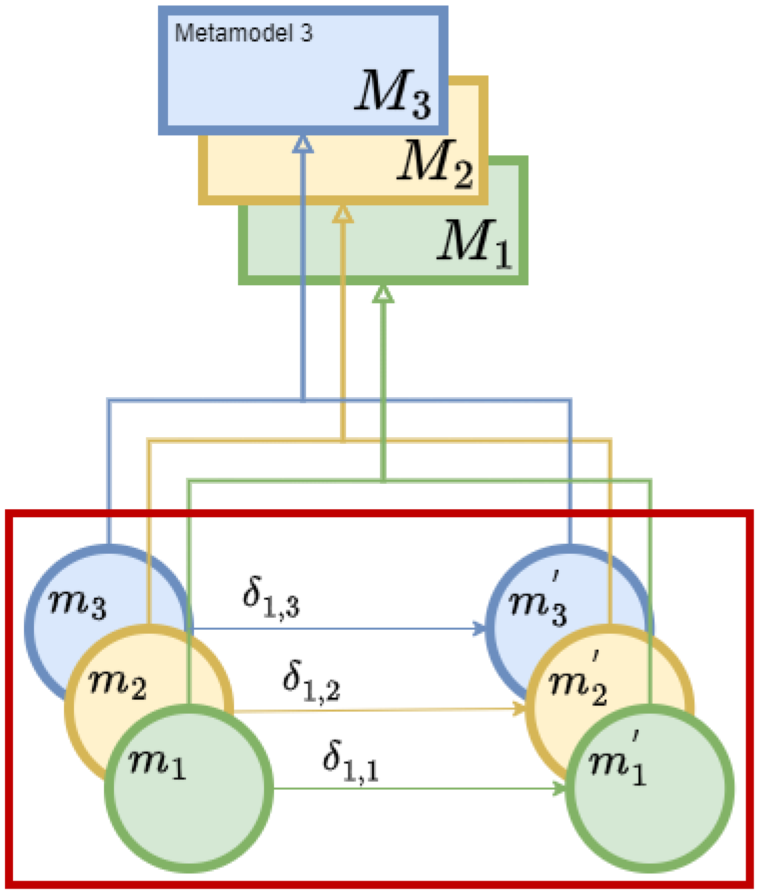
Delta-oriented variability
in a model-driven world

CPR = Consistency Preservation Rule

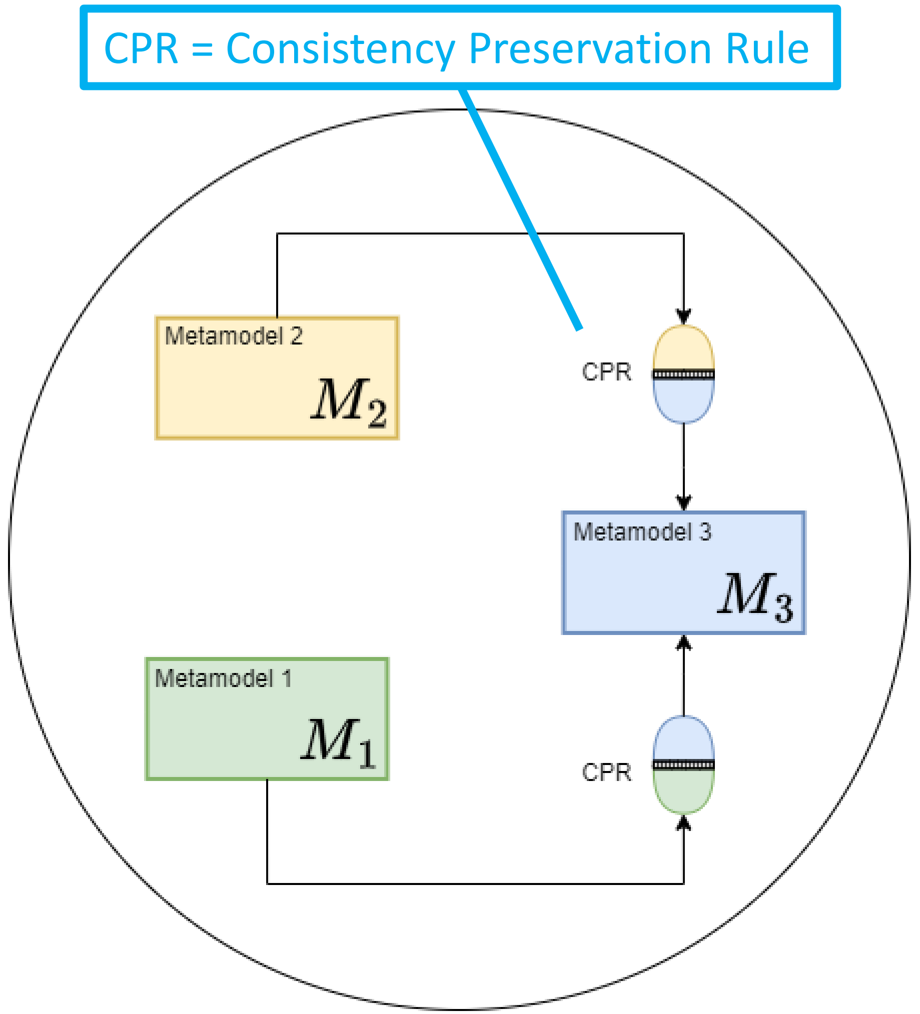


Consistency preservation
in a model-driven world

CONTEXT – CONSISTENCY PRESERVATION

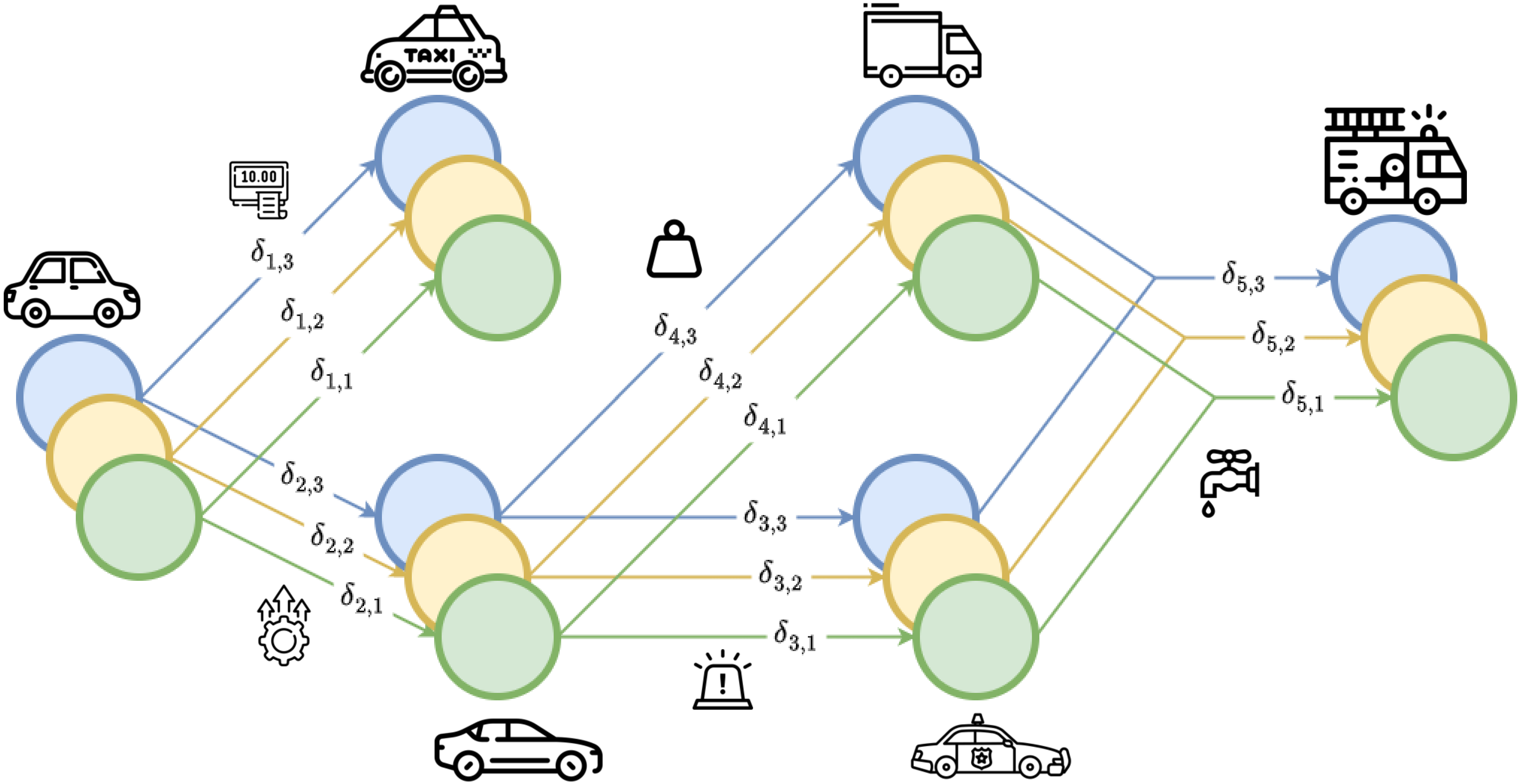


Delta-oriented variability
in a model-driven world

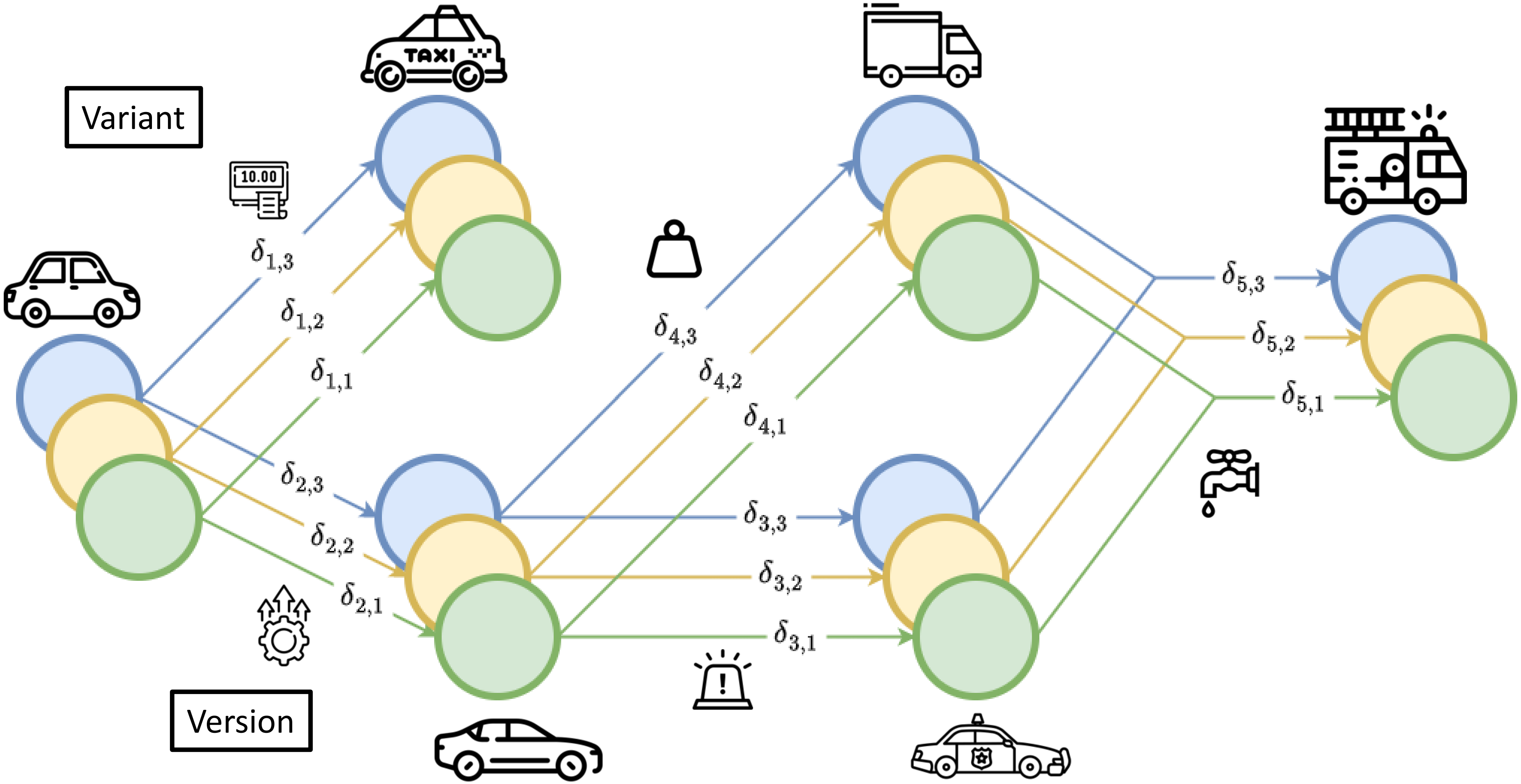


Consistency preservation
in a model-driven world

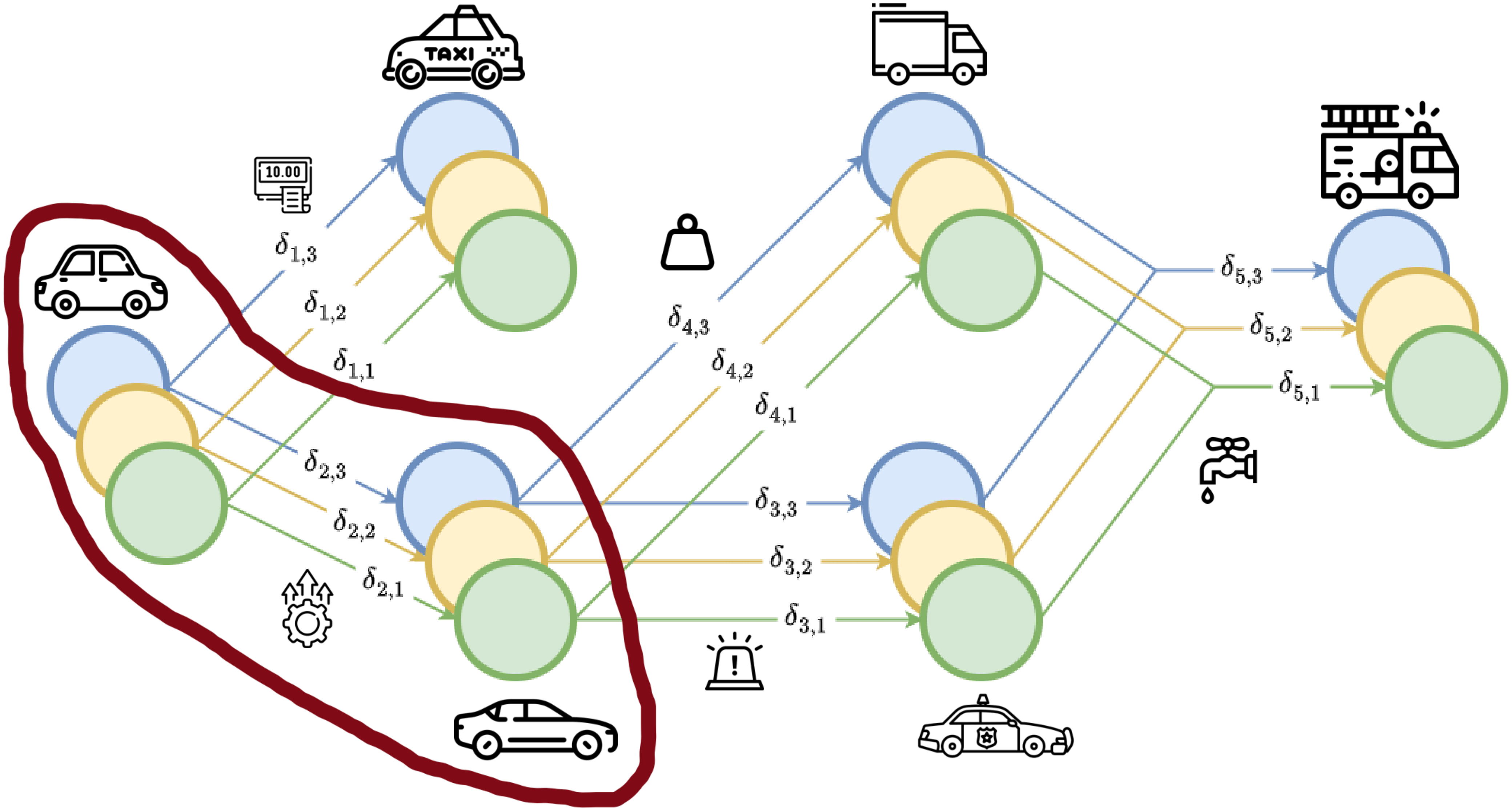
CONTEXT – DELTA-ORIENTED PRODUCT LINE



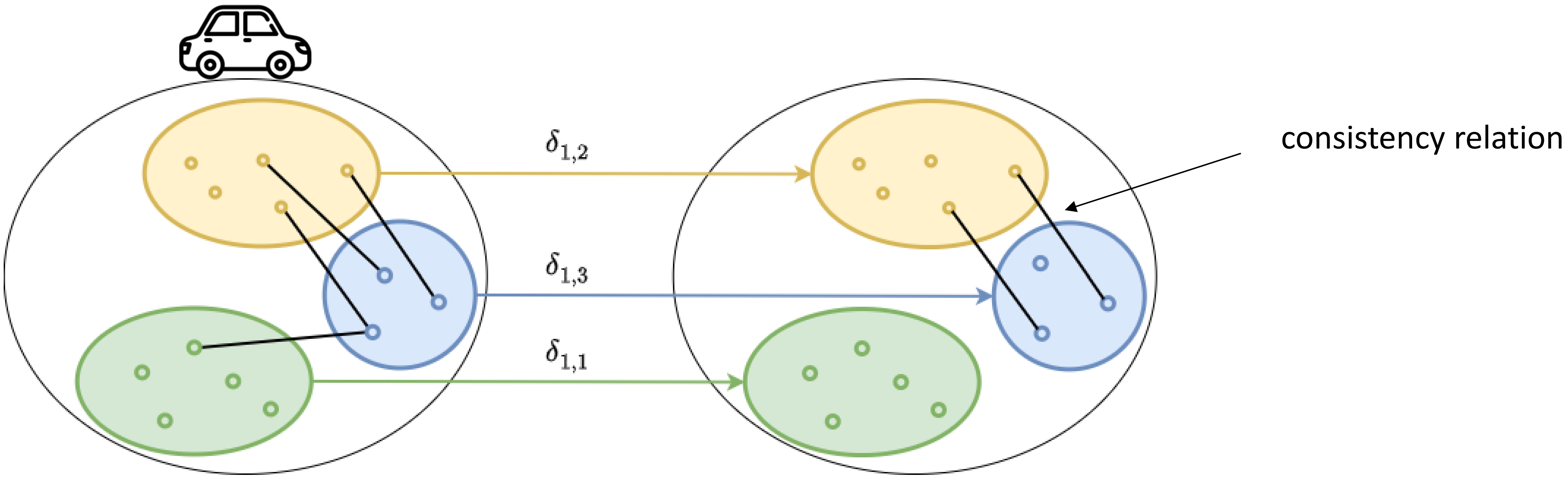
CONTEXT – DELTA-ORIENTED PRODUCT LINE



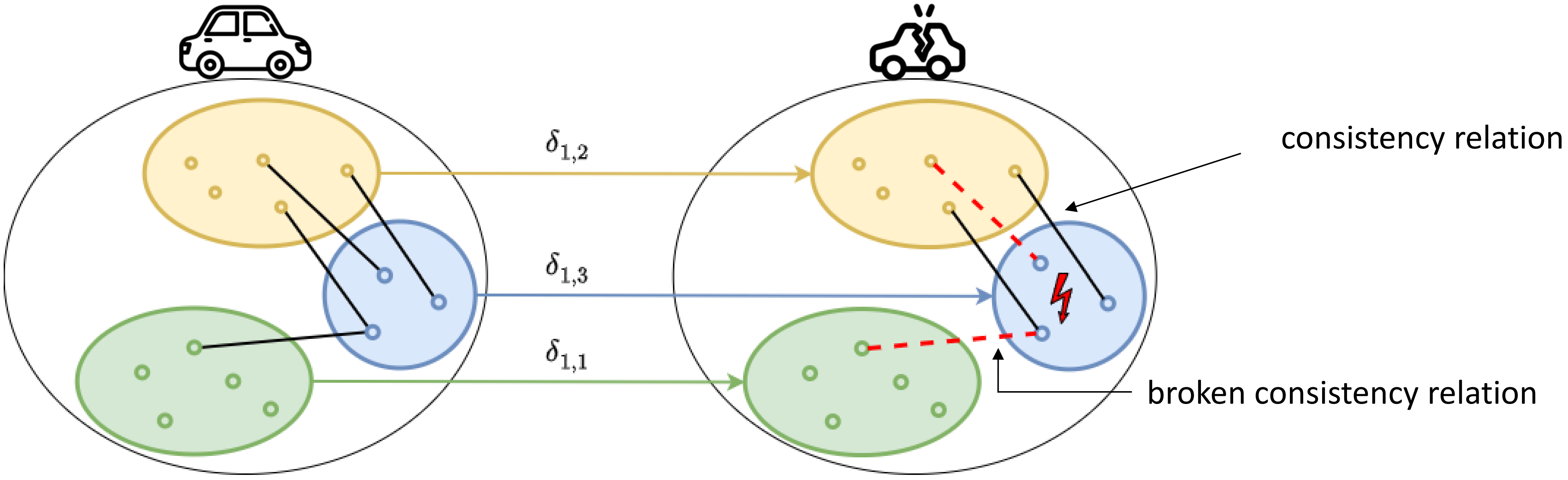
CONTEXT – DELTA-ORIENTED PRODUCT LINE



PROBLEM

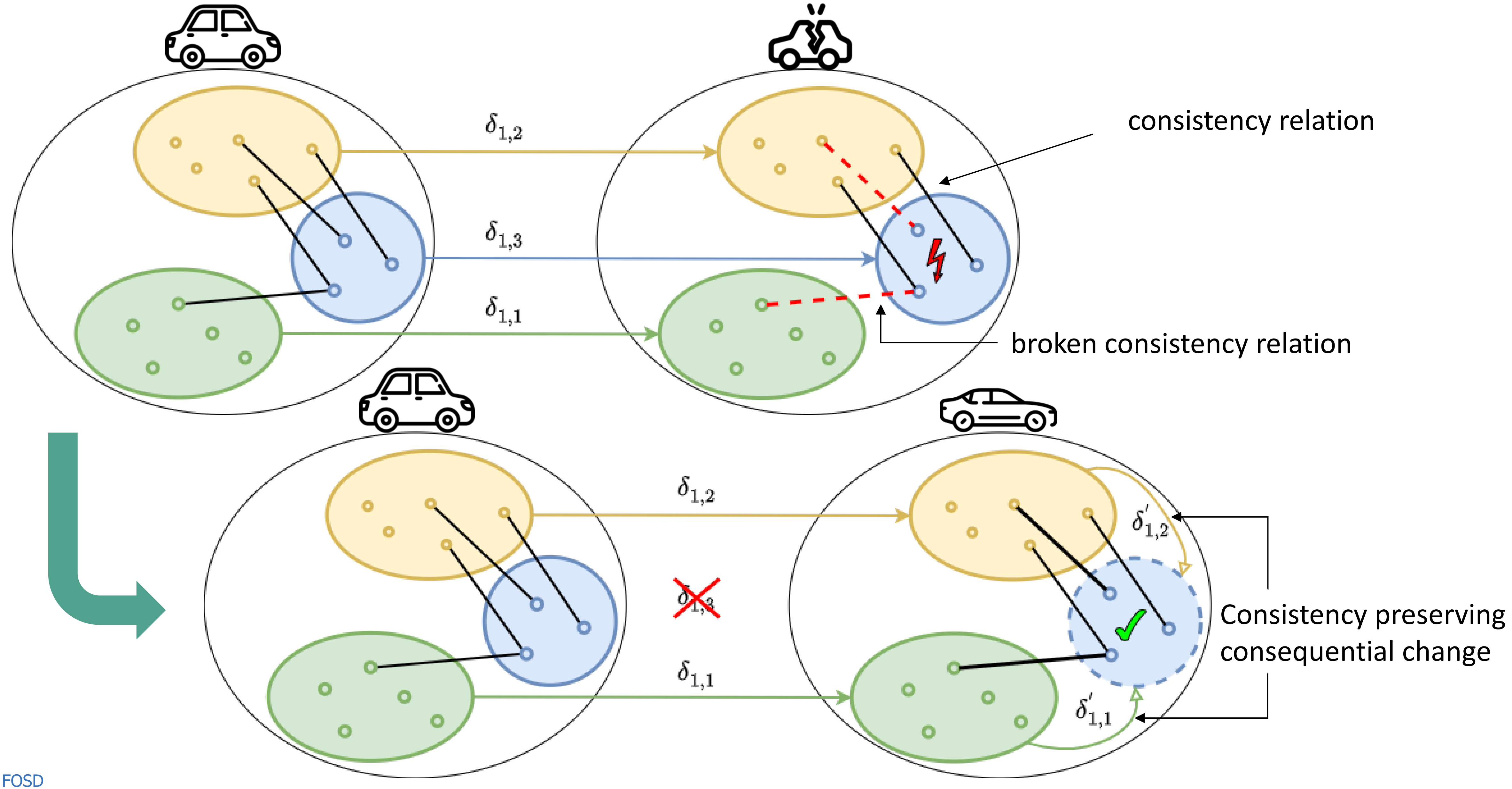


PROBLEM

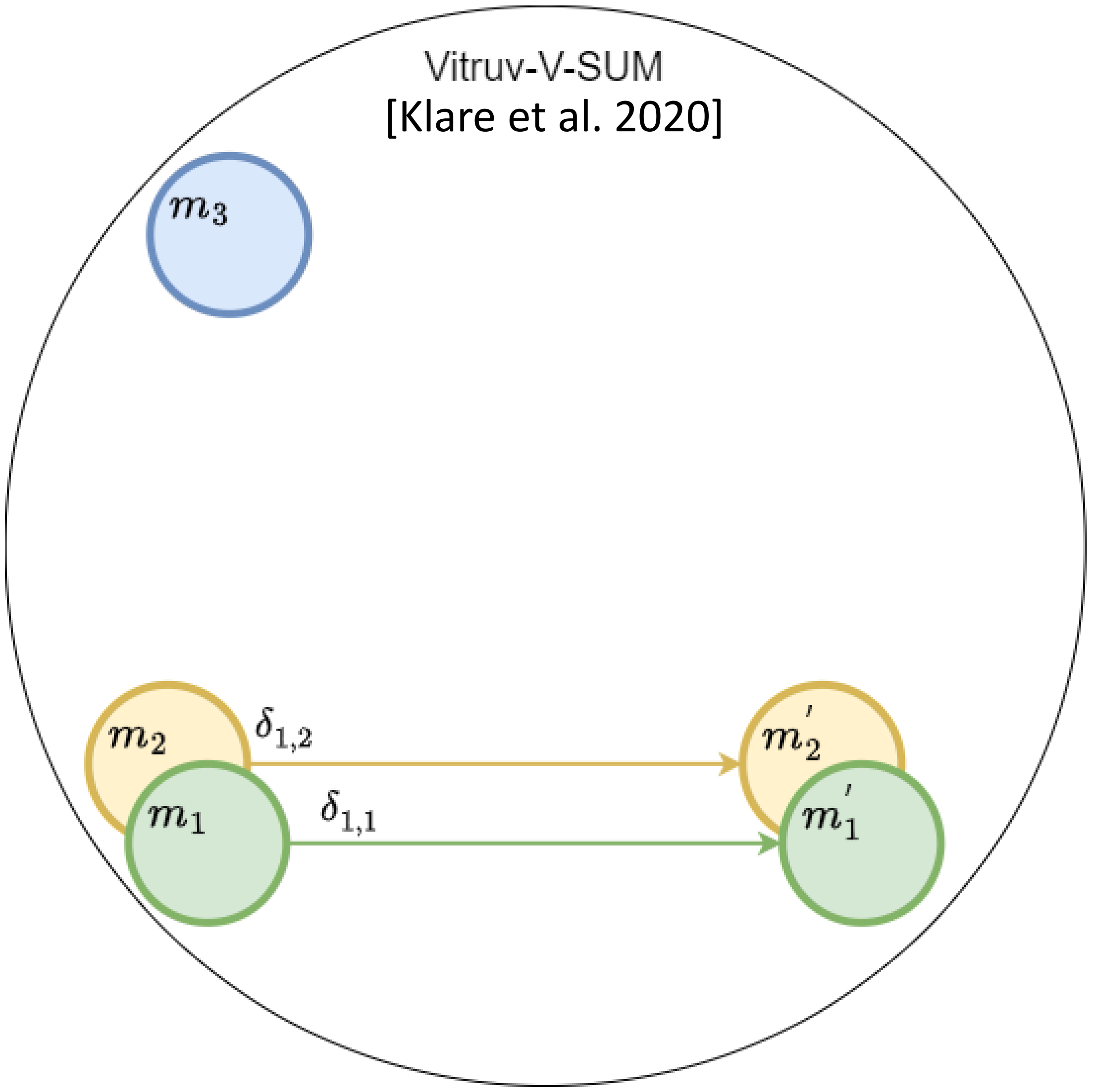


$\delta_{1,2}$
$\delta_{1,3}$
$\delta_{1,1}$
consistency relation
broken consistency relation

PROBLEM


$\delta_{1,2}$
$\delta_{1,3}$
$\delta_{1,1}$
consistency relation
broken consistency relation
$\delta_{1,2}$
$\delta_{1,3}$
$\delta_{1,1}$
$\delta'_{1,2}$
$\delta'_{1,1}$
Consistency preserving
consequential change

IDEA

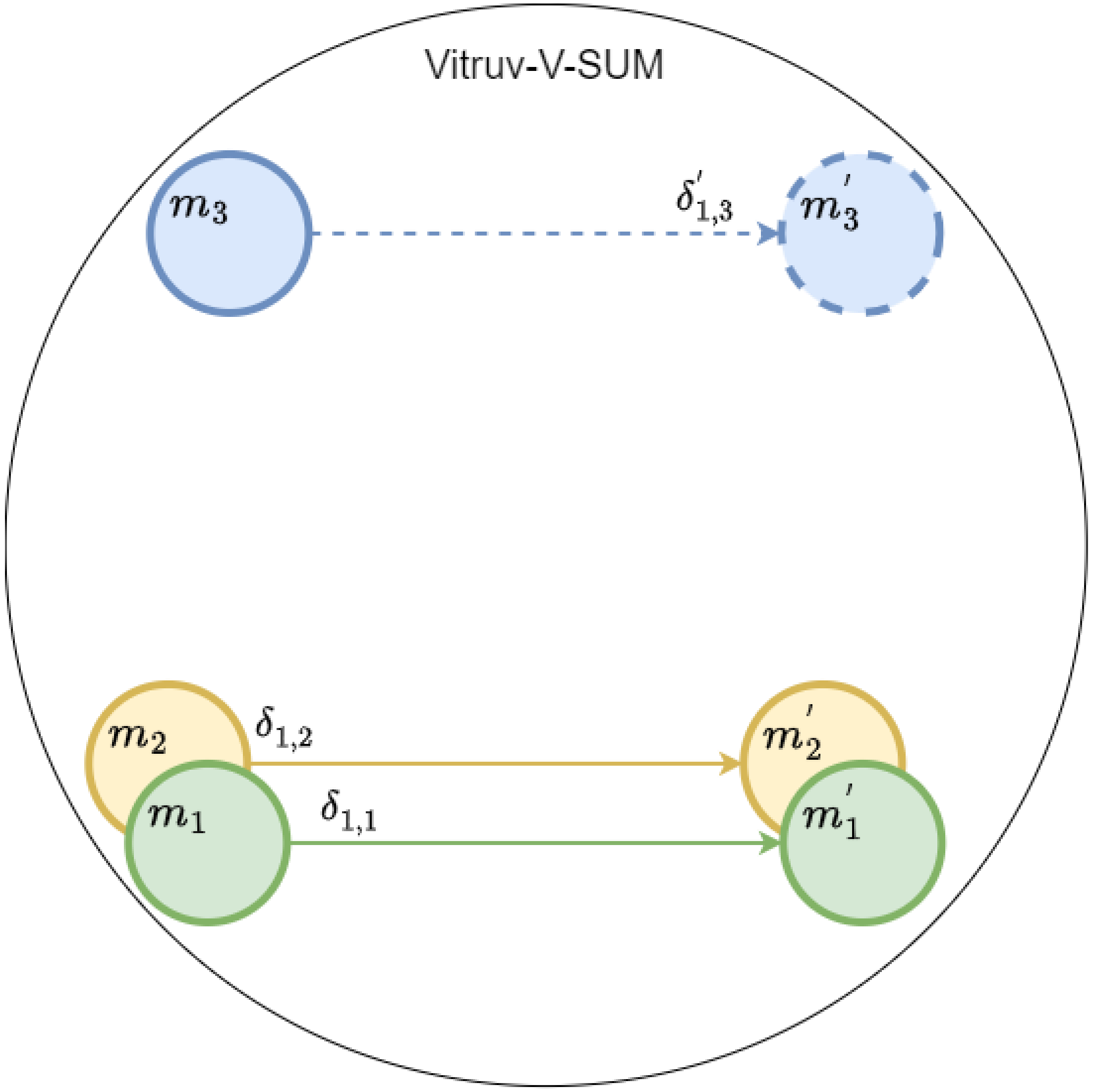


Idea:
Using consistency preservation as a mechanism for model-driven development

Advantage:

- Getting a consistent version of model 3
- Getting a delta for further SPLE development

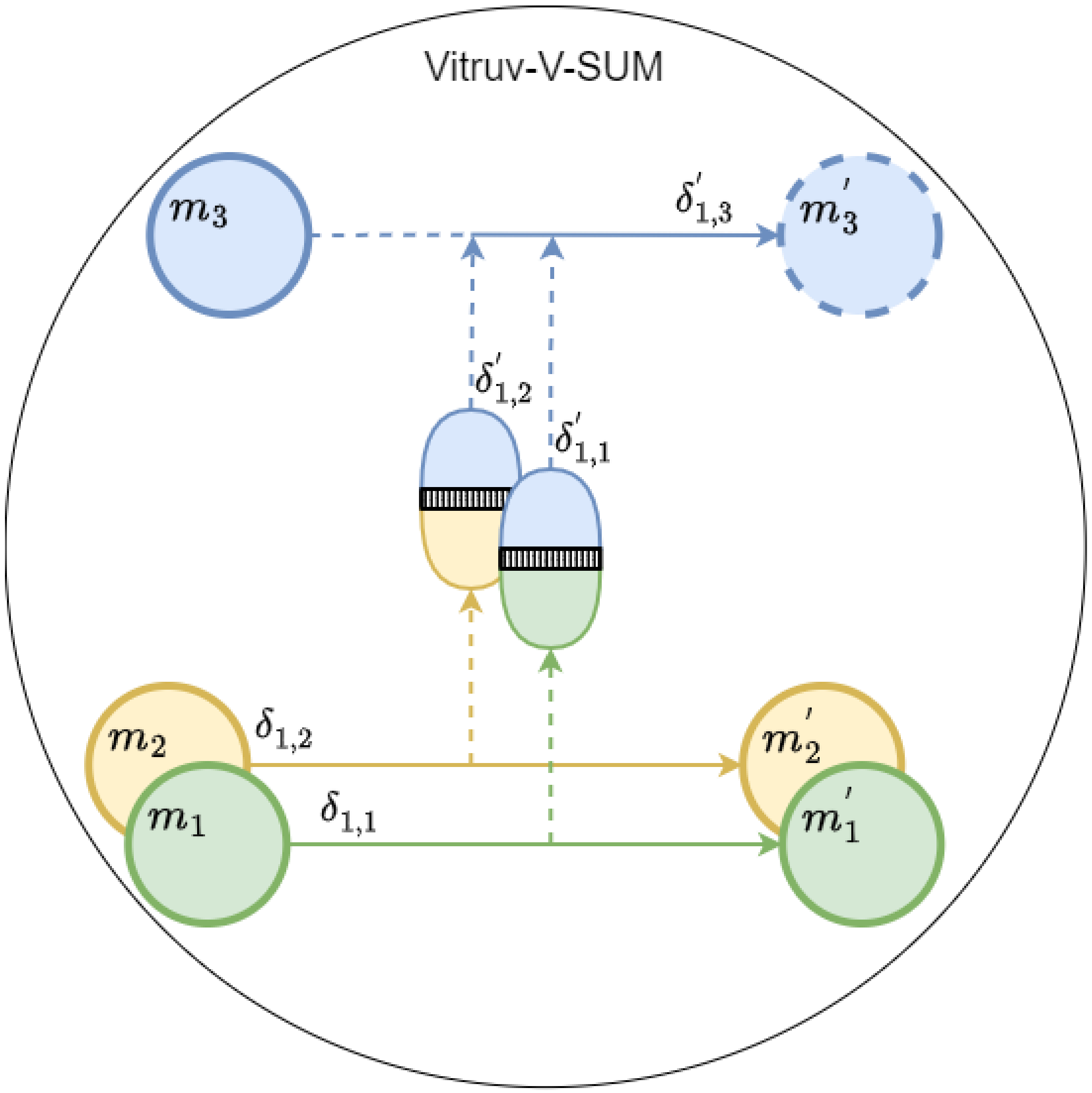
IDEA



Idea:
Using consistency preservation as a mechanism for model-driven development

Advantage:
- Getting a consistent version of model 3
- Getting a delta for further SPLE development

IDEA

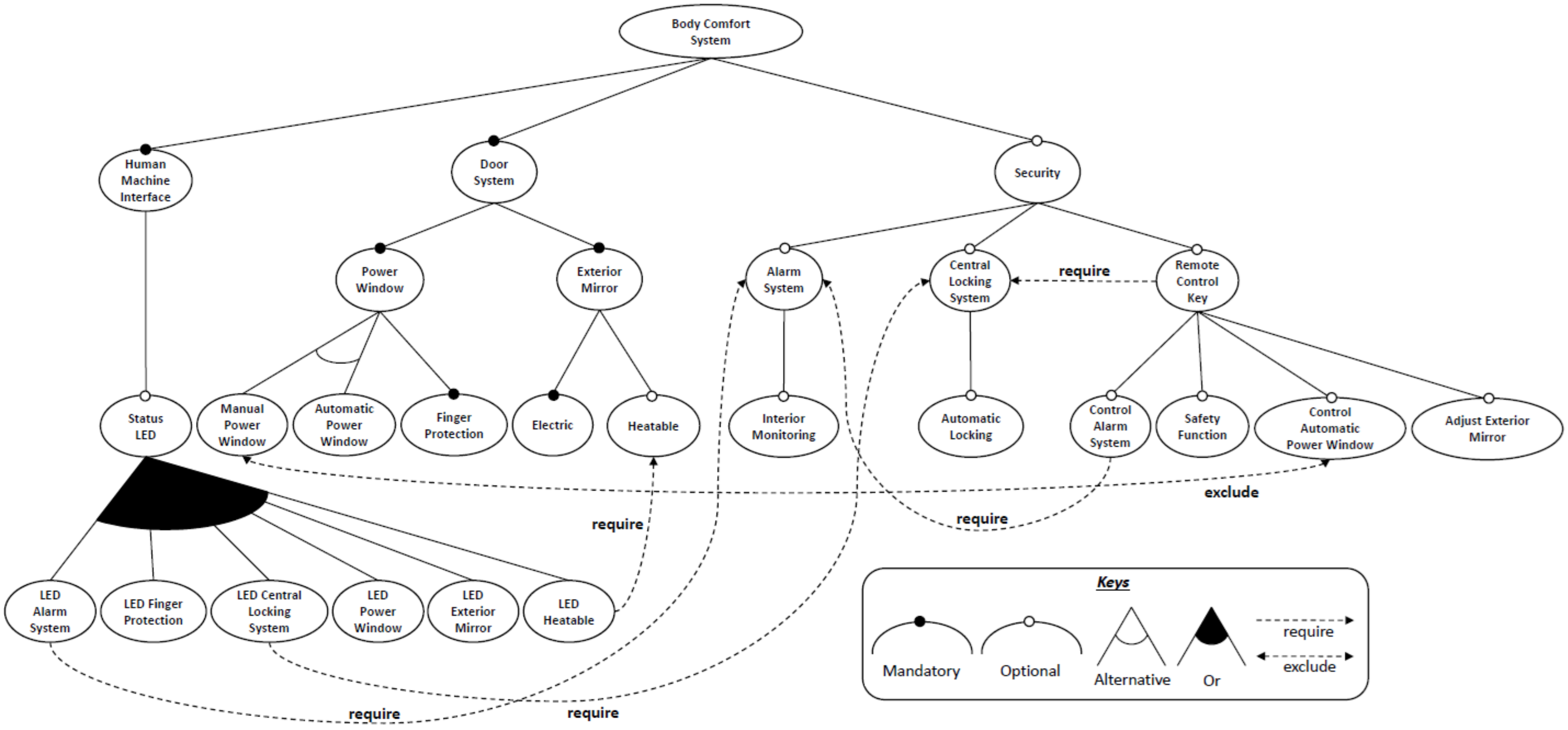


Idea:
Using consistency preservation as a mechanism for model-driven development

Advantage:

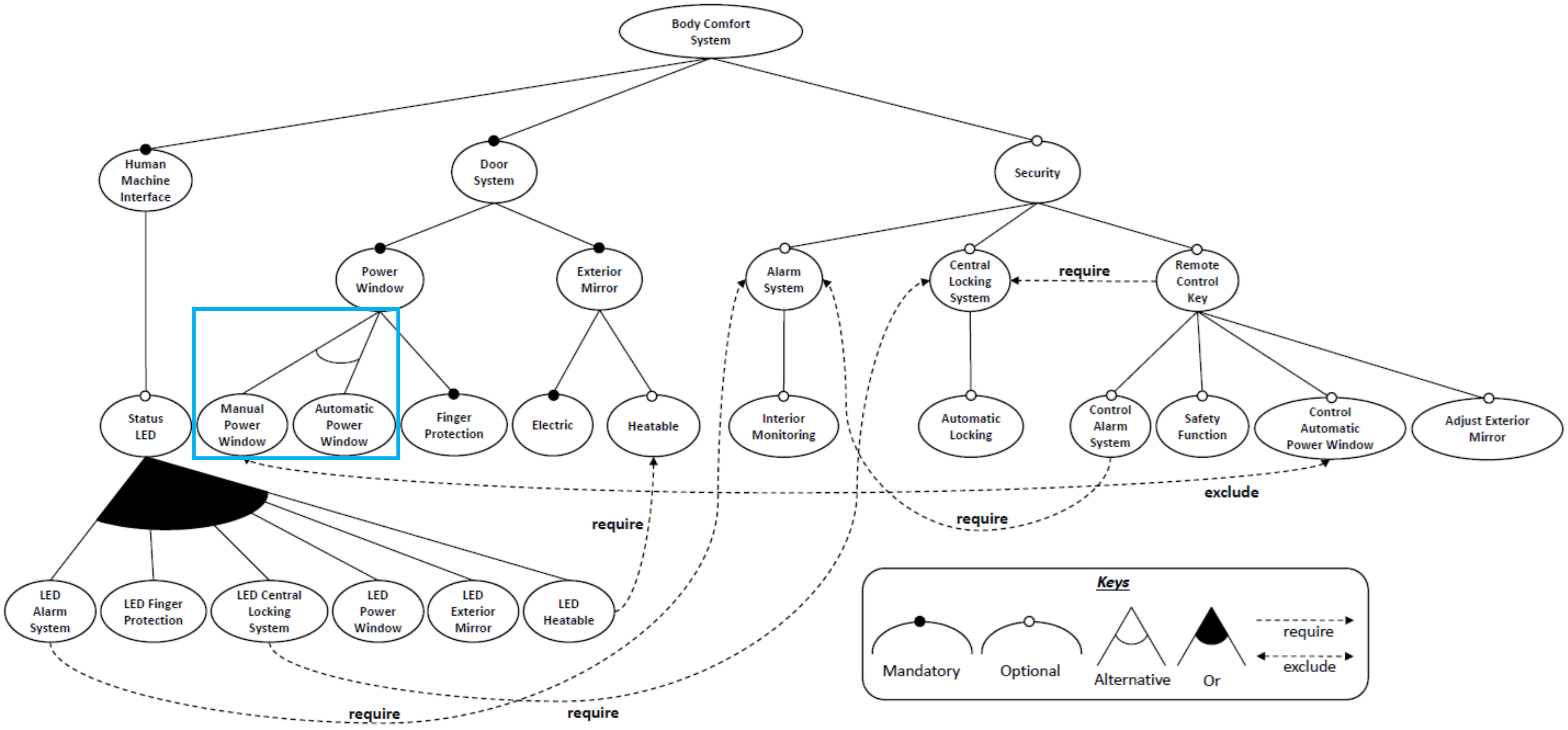
- Getting a consistent version of model 3
- Getting a delta for further SPLE development

EXAMPLE – THE BODY COMFORT SYSTEM (BCS)



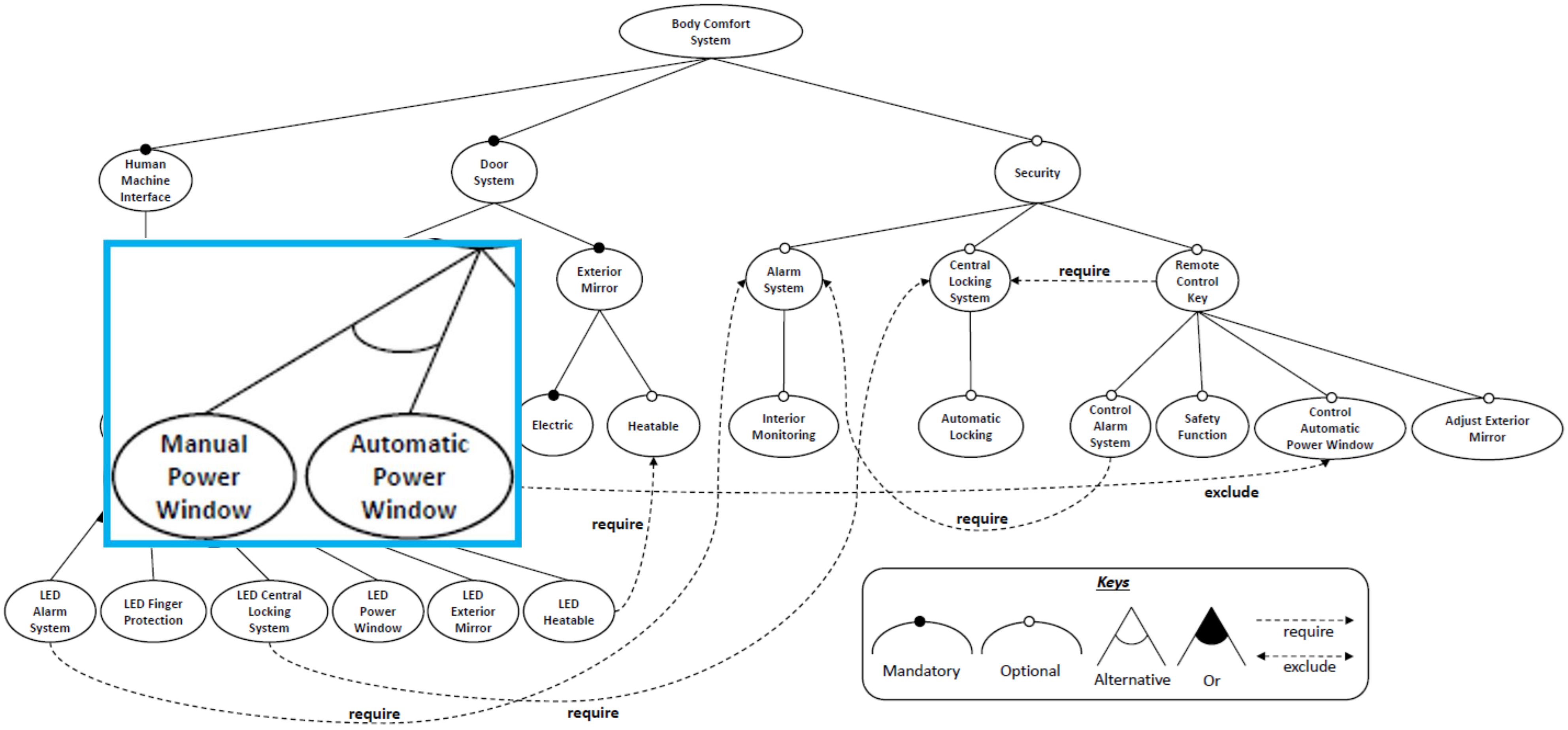
11‘616 valid configurations

EXAMPLE – THE BODY COMFORT SYSTEM (BCS)



11‘616 valid configurations

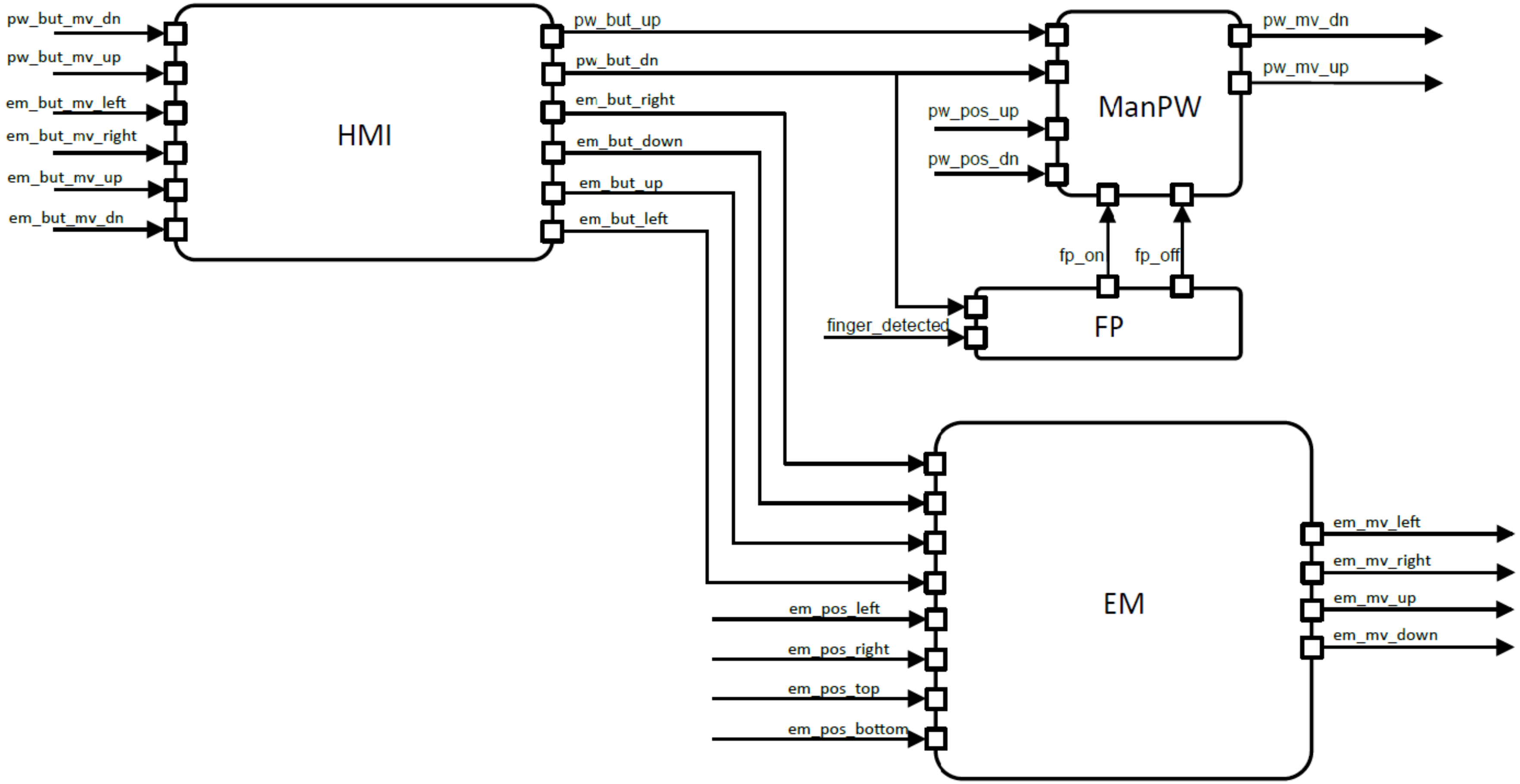
EXAMPLE – THE BODY COMFORT SYSTEM (BCS)



11‘616 valid configurations

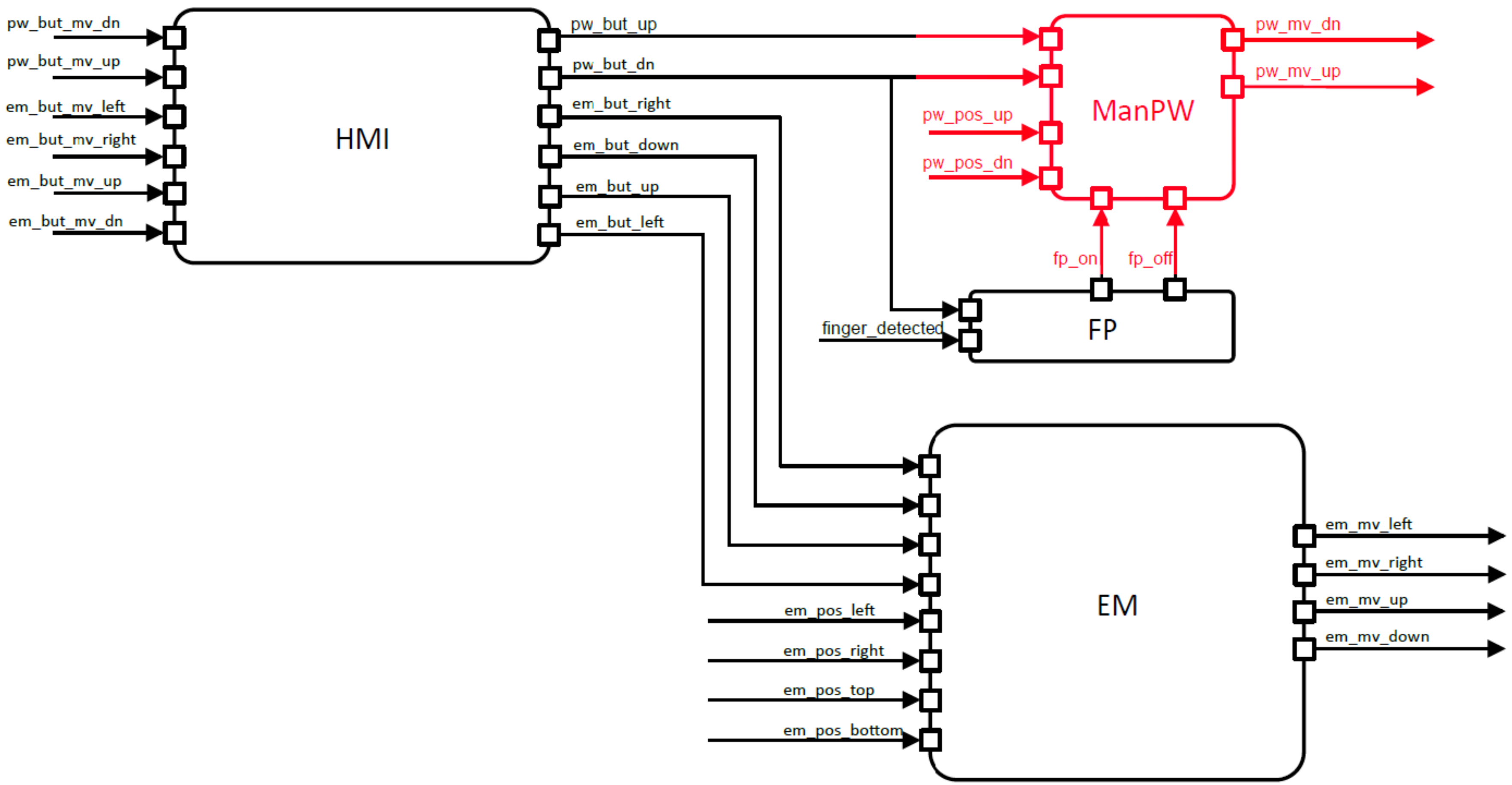
BCS – CORE PRODUCT (P0)

Structural:
architecture component



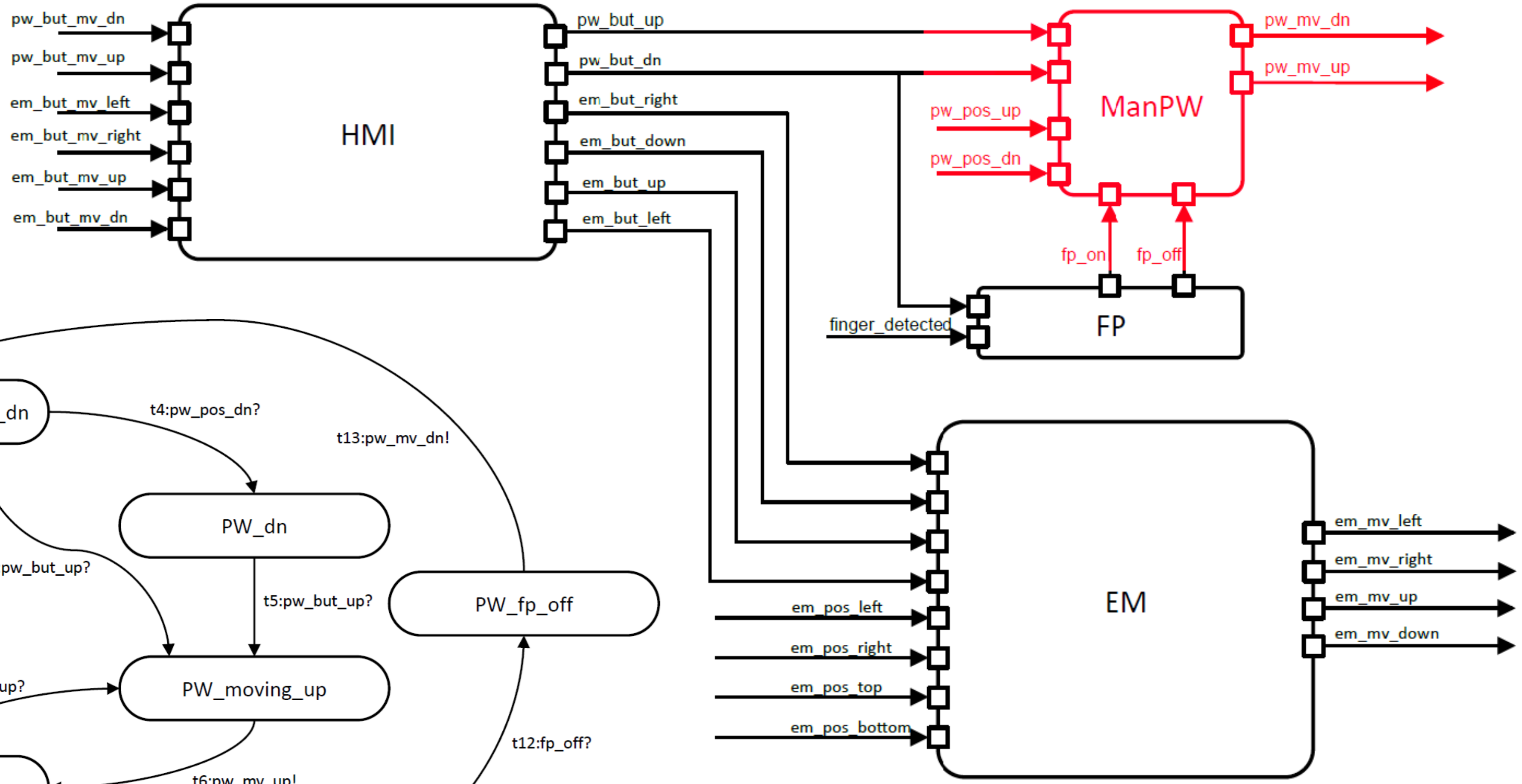
BCS – CORE PRODUCT (P0)

Structural:
architecture component

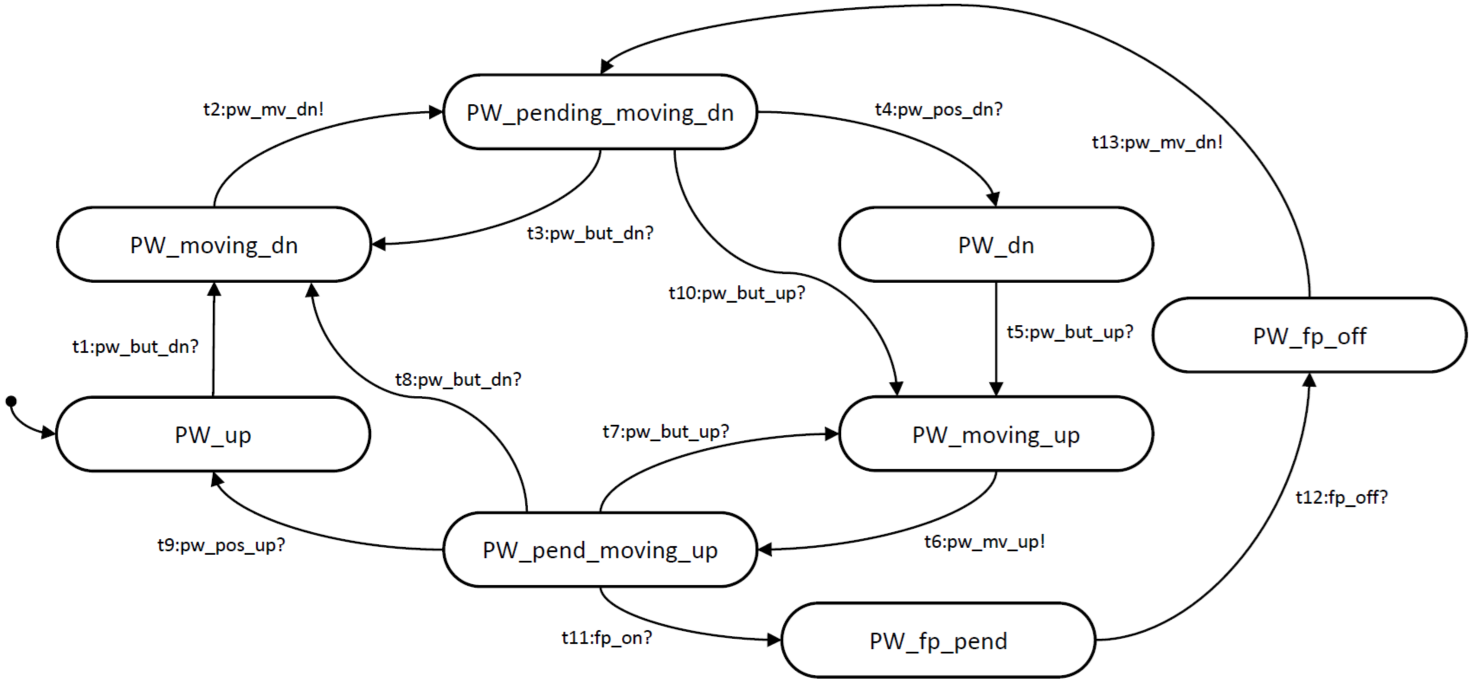


BCS – CORE PRODUCT (P0)

Structural:
architecture component

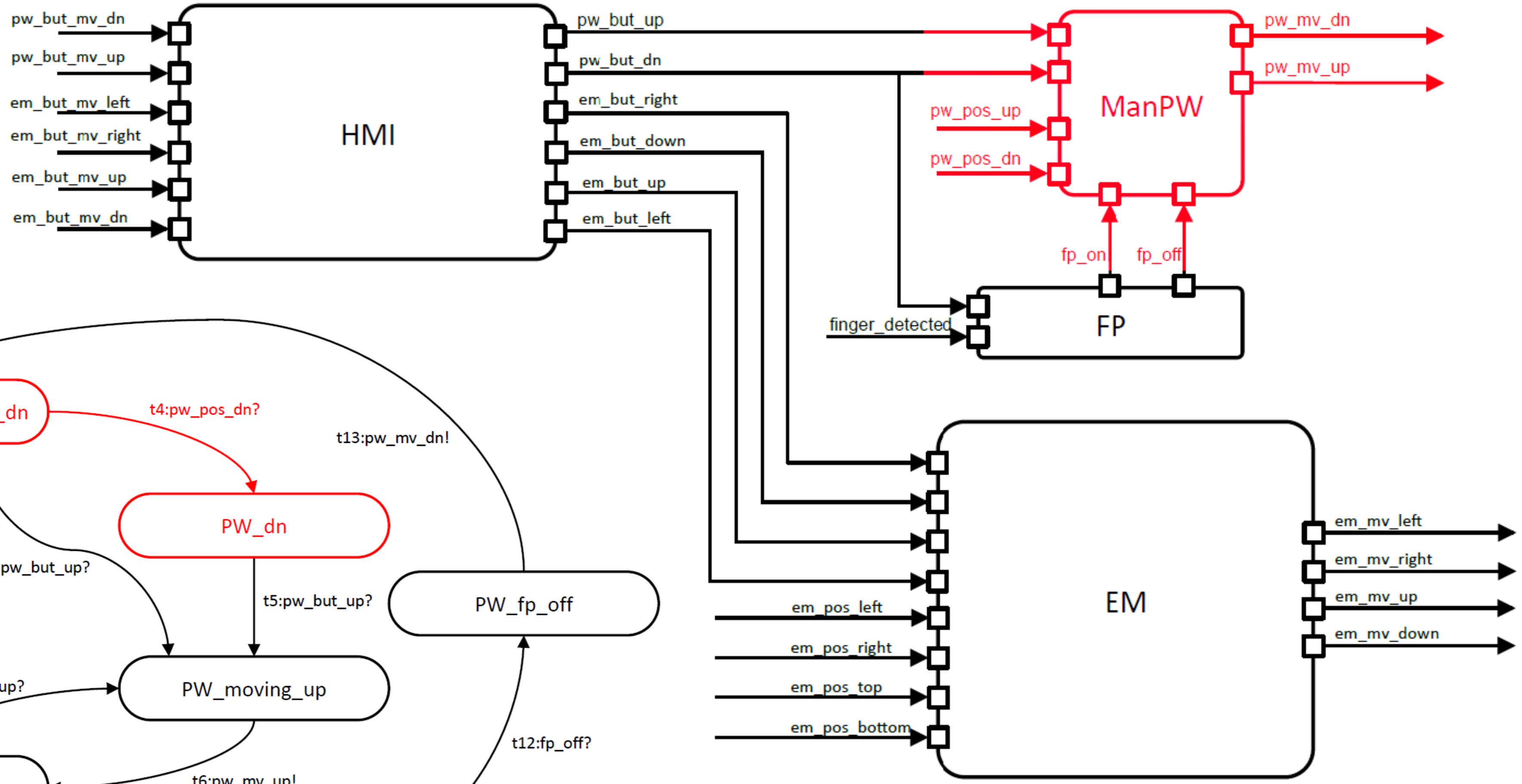


Behavioral:

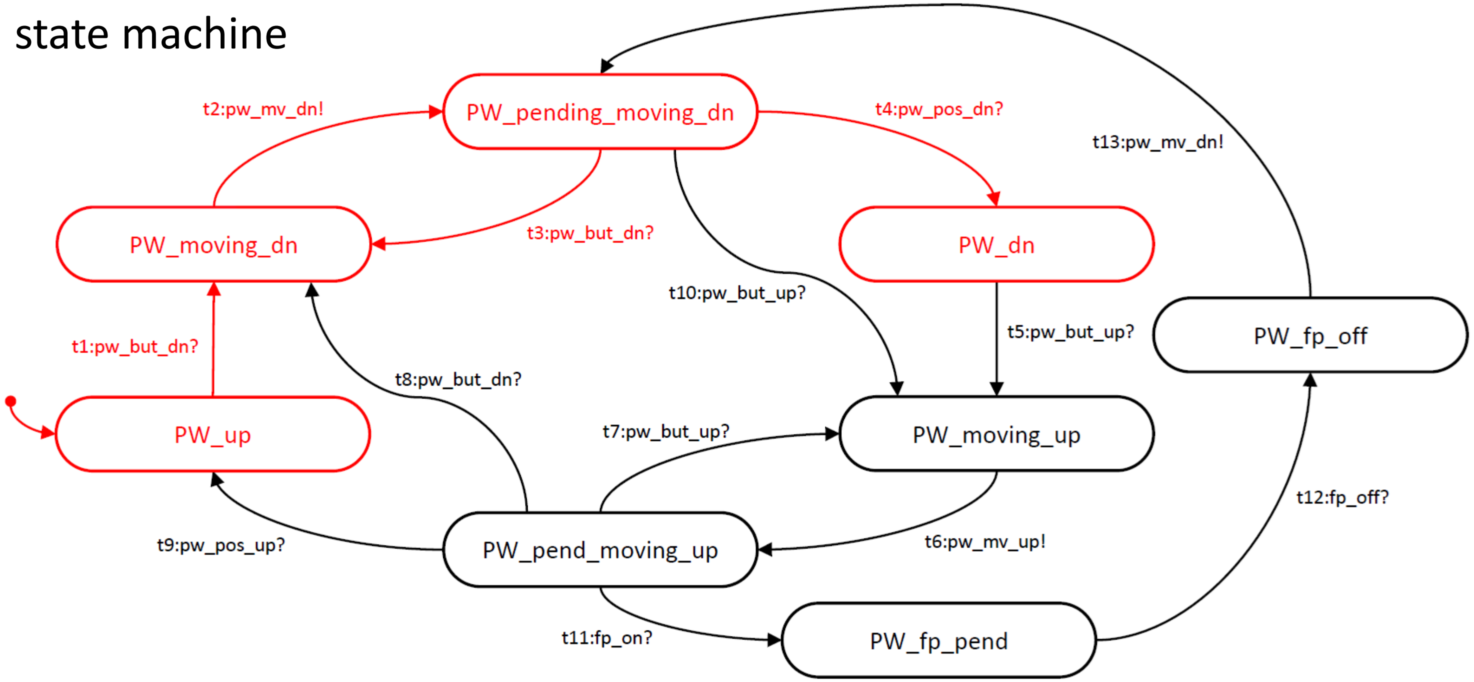


BCS – CORE PRODUCT (P0)

Structural:
architecture component

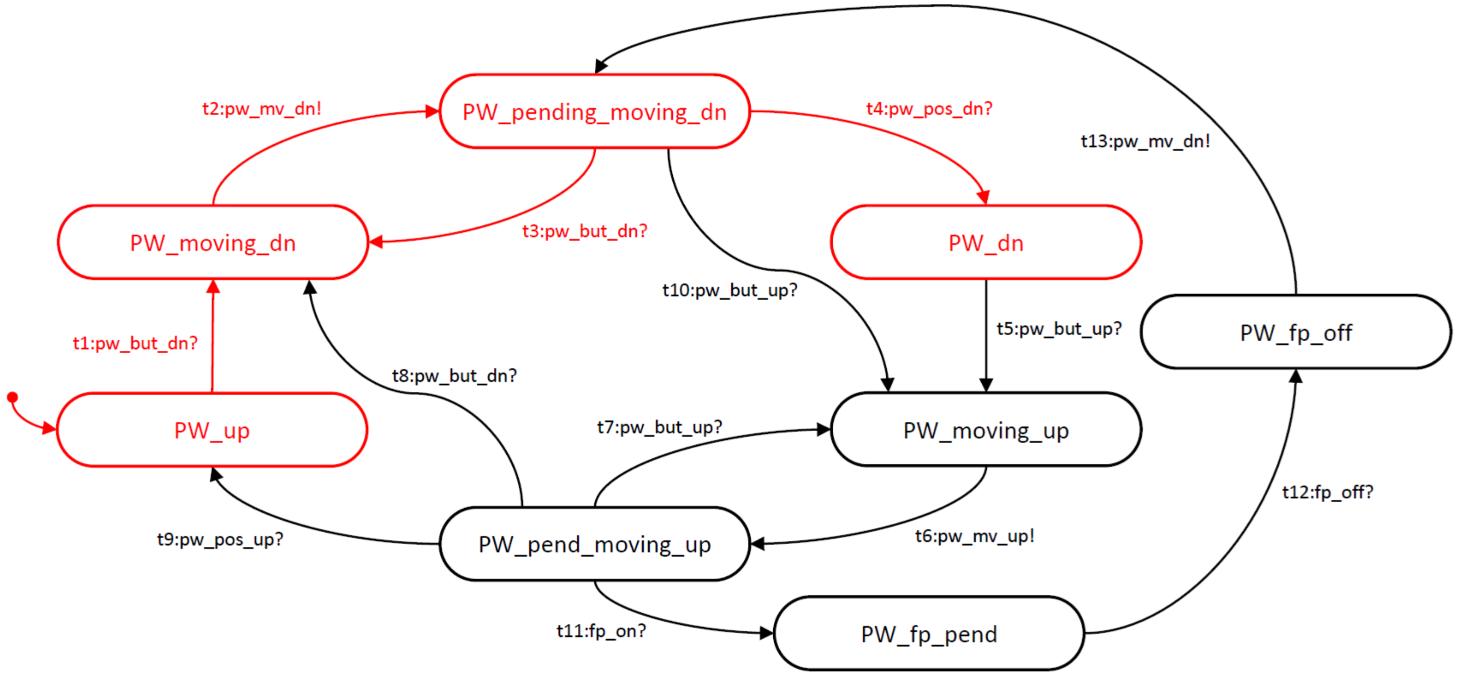


Behavioral:
state machine

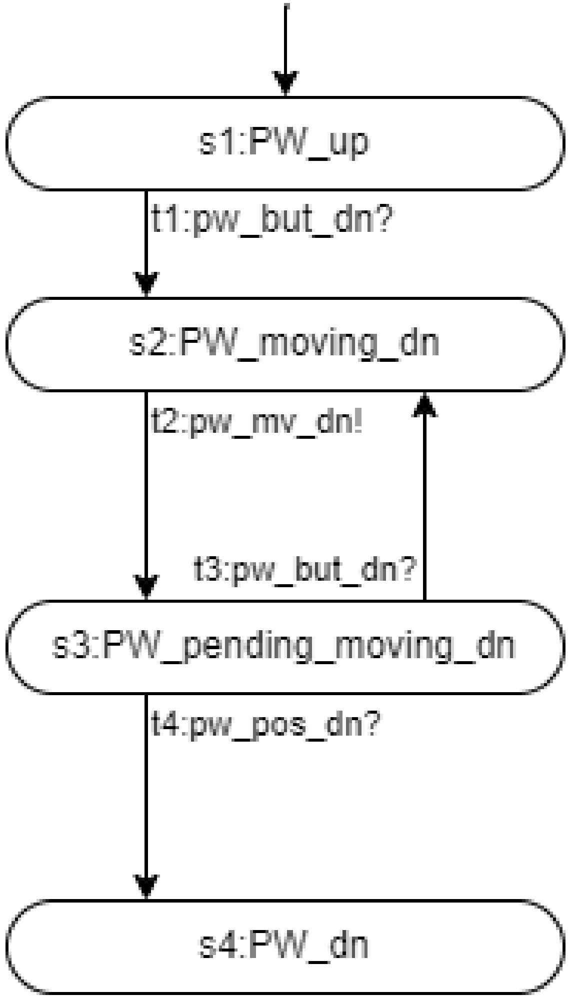


DELTA-ORIENTED VARIABILITY

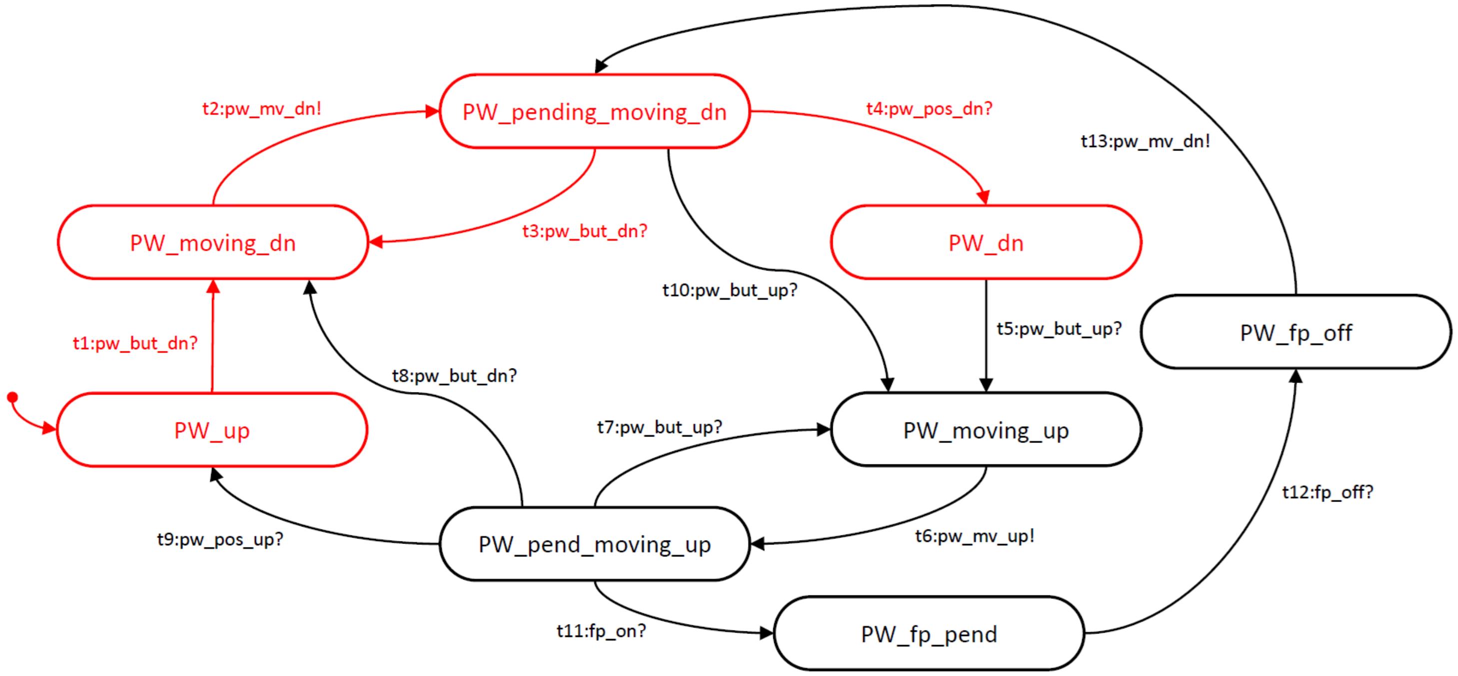
P0.statemachine → region:ManualPowerWindow



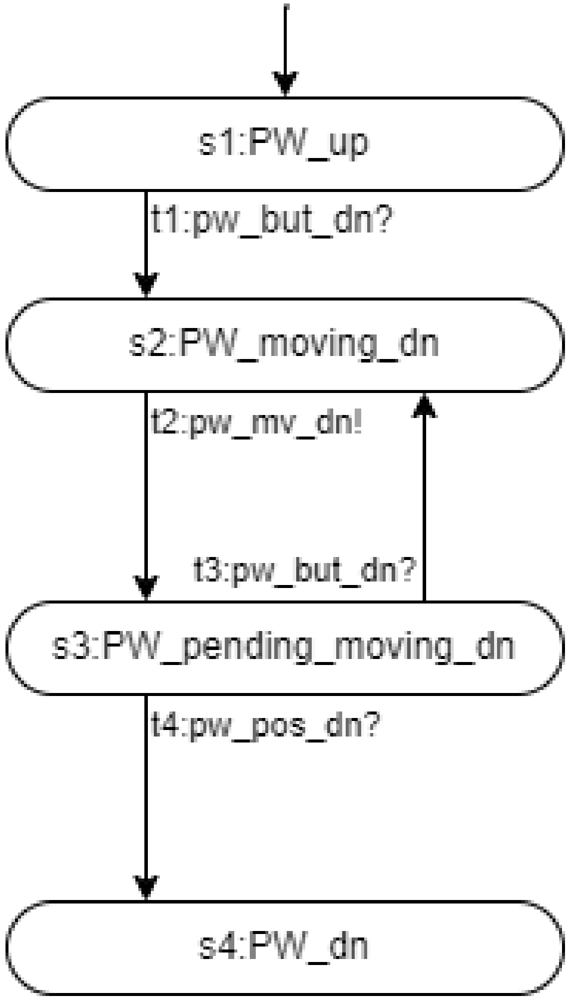
DELTA-ORIENTED VARIABILITY



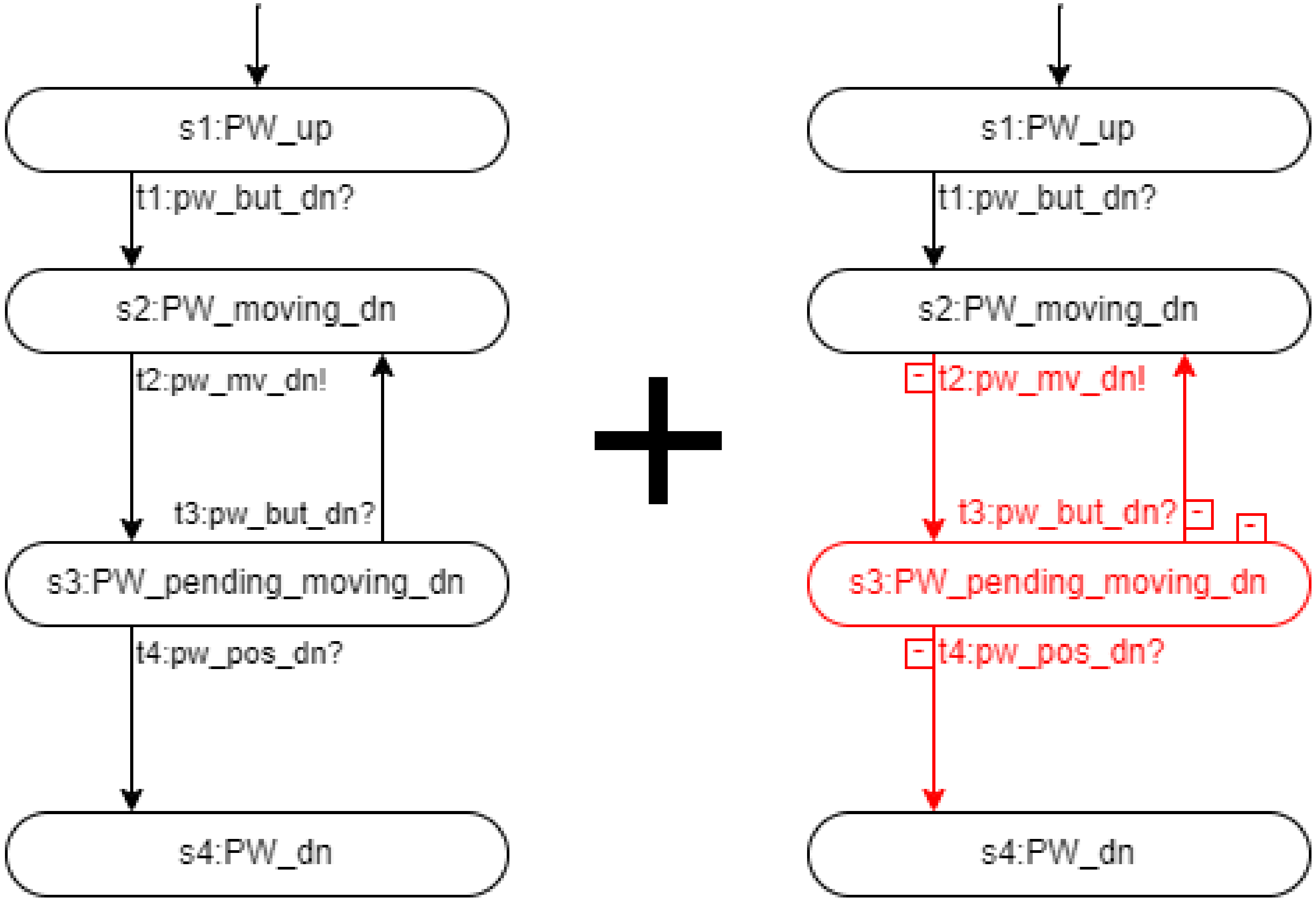
P0.statemachine → region:ManualPowerWindow



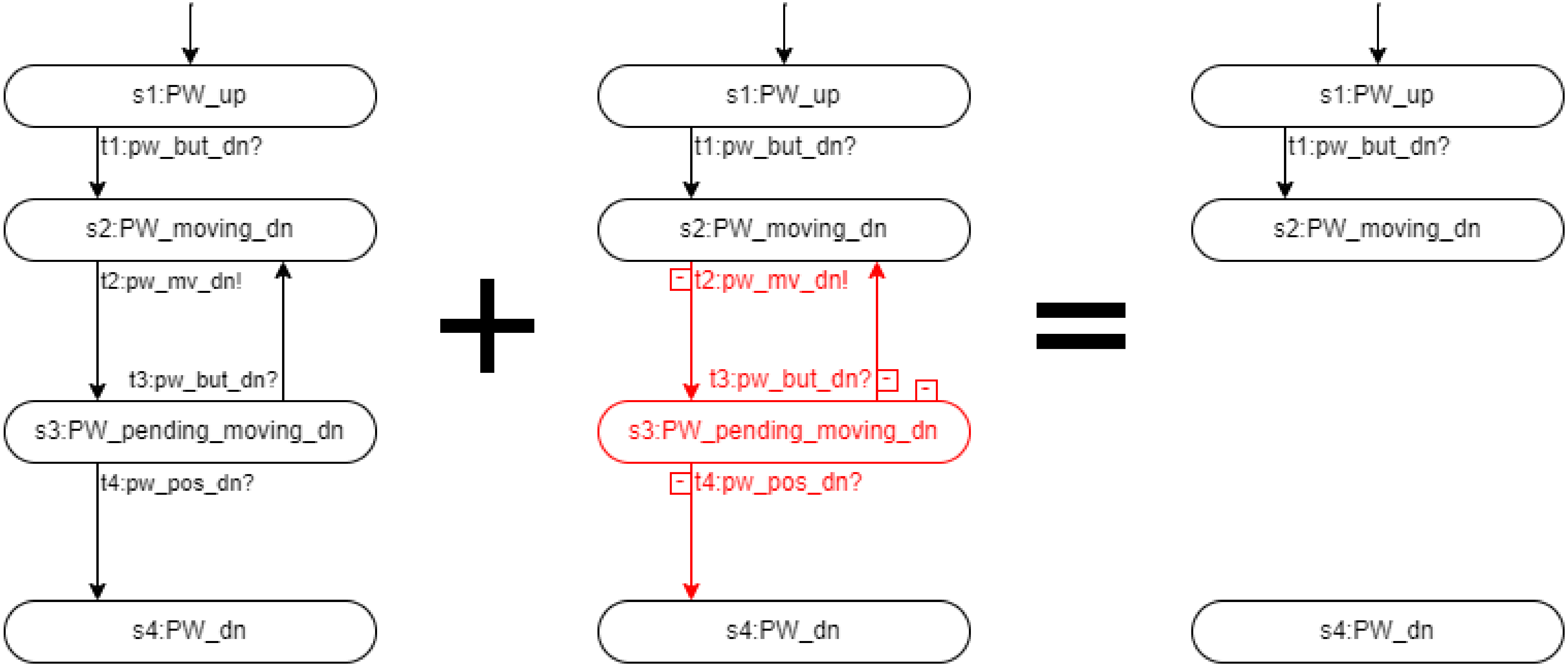
DELTA-ORIENTED VARIABILITY



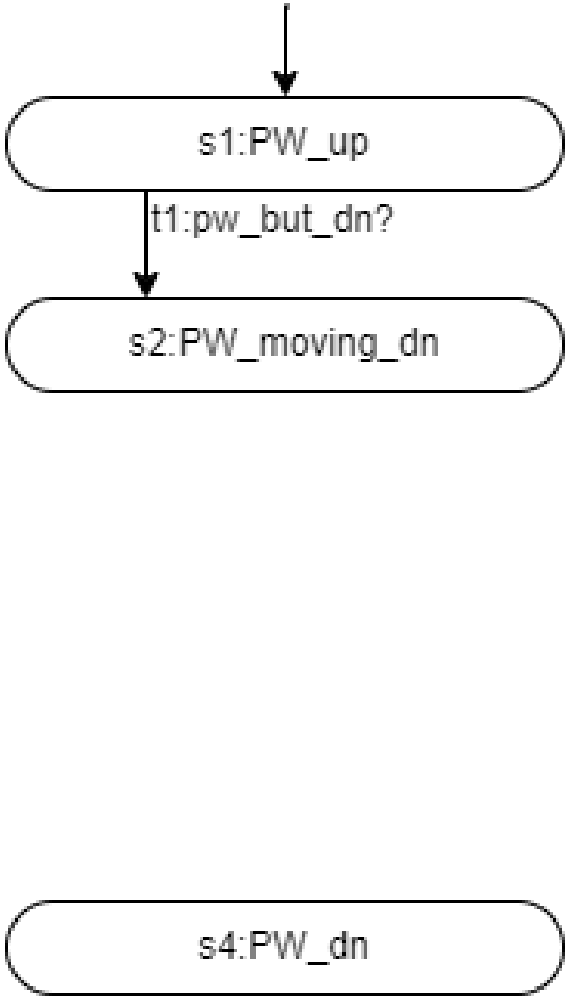
DELTA-ORIENTED VARIABILITY



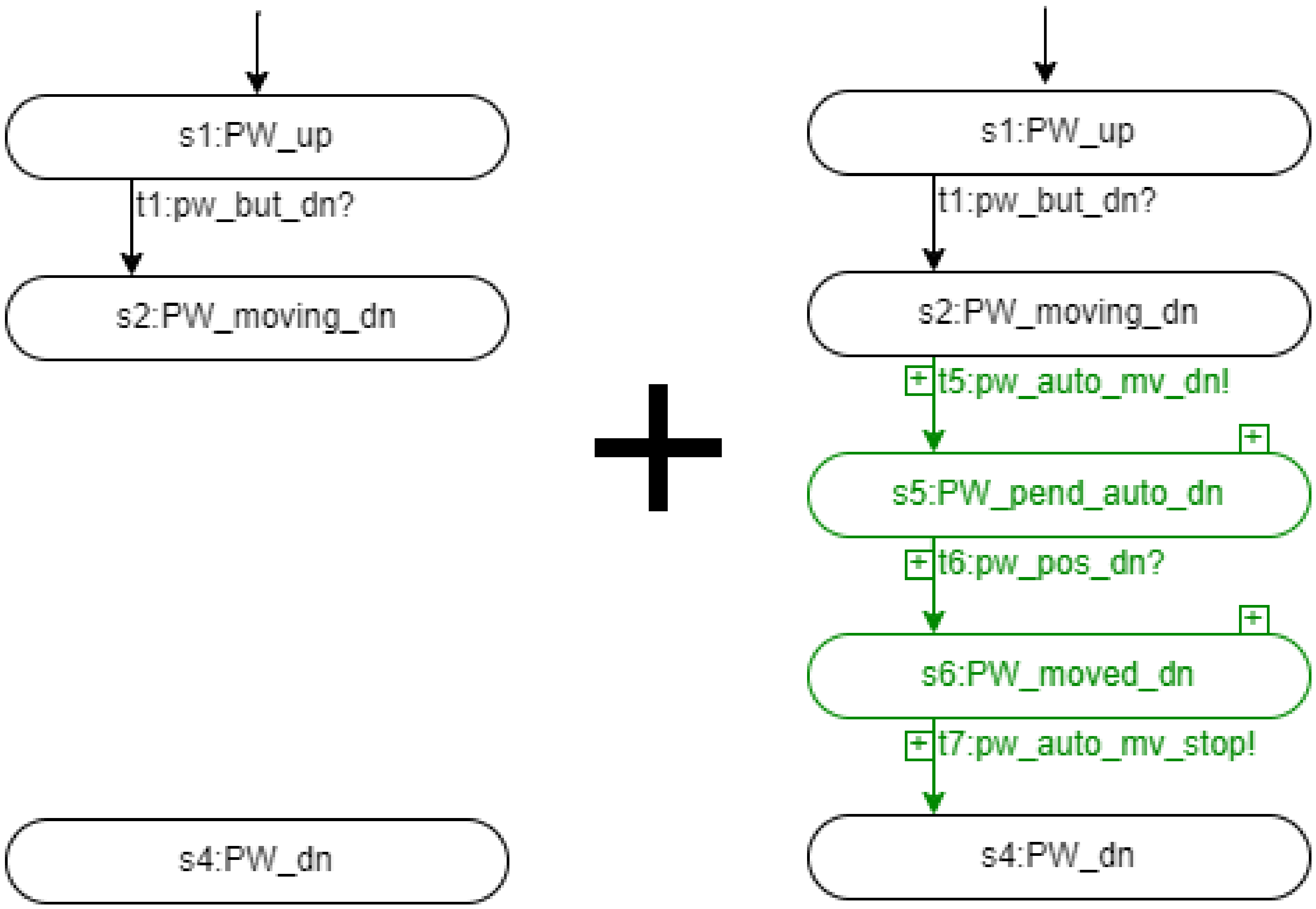
DELTA-ORIENTED VARIABILITY



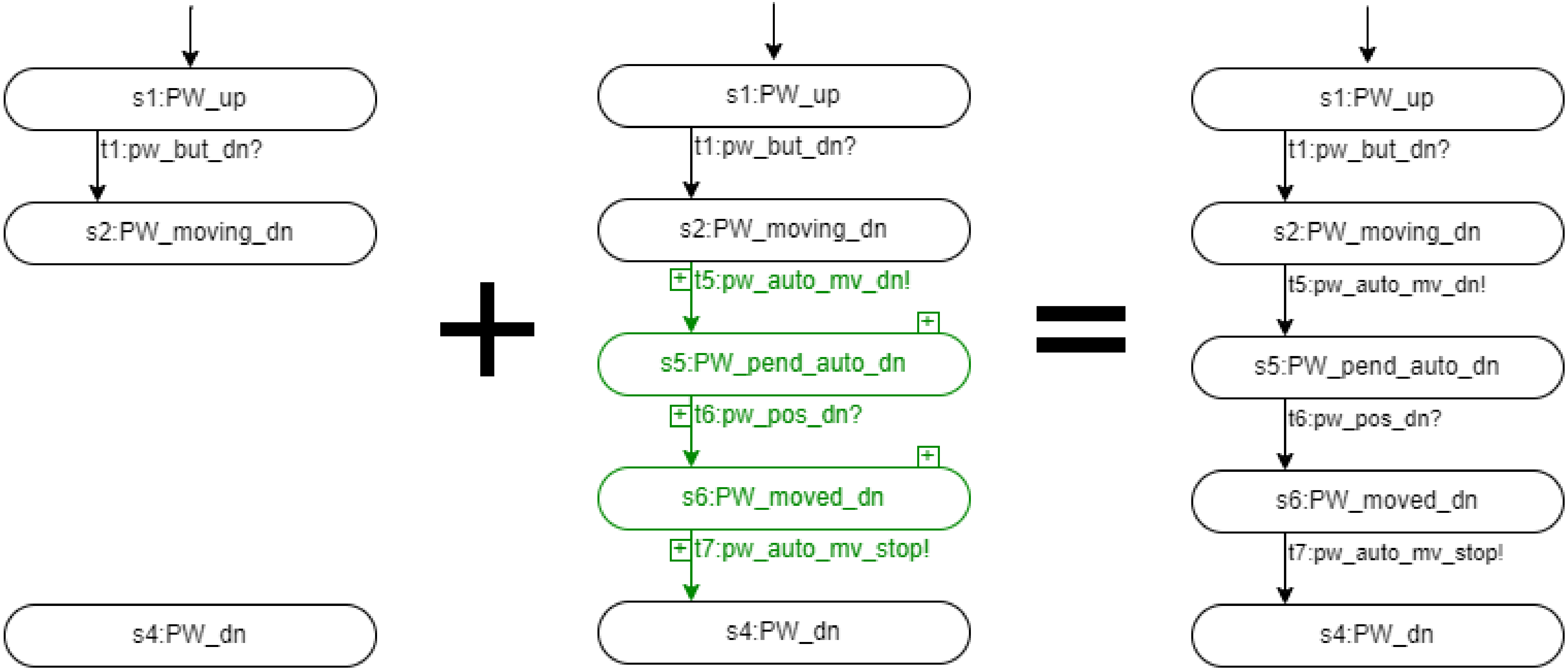
DELTA-ORIENTED VARIABILITY



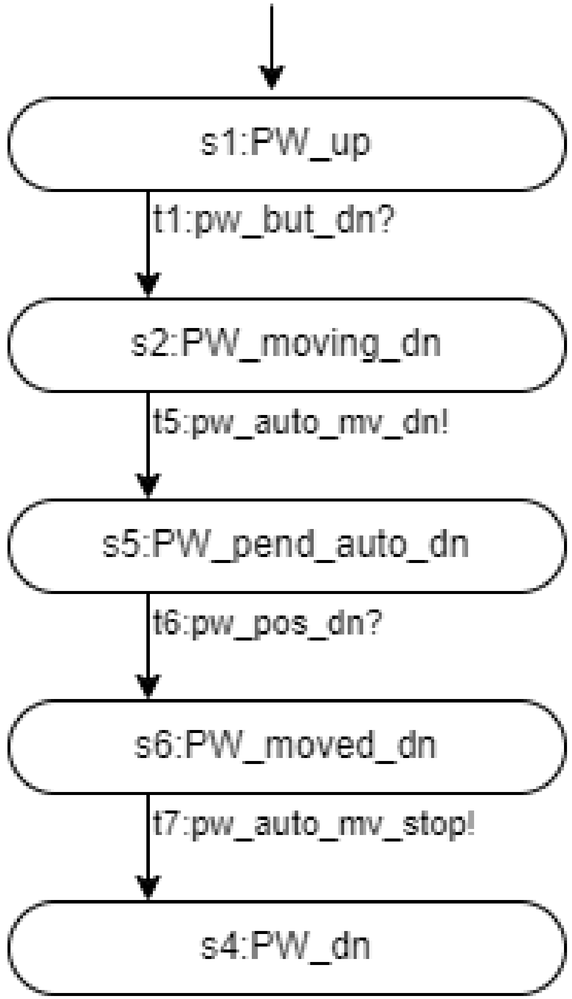
DELTA-ORIENTED VARIABILITY



DELTA-ORIENTED VARIABILITY

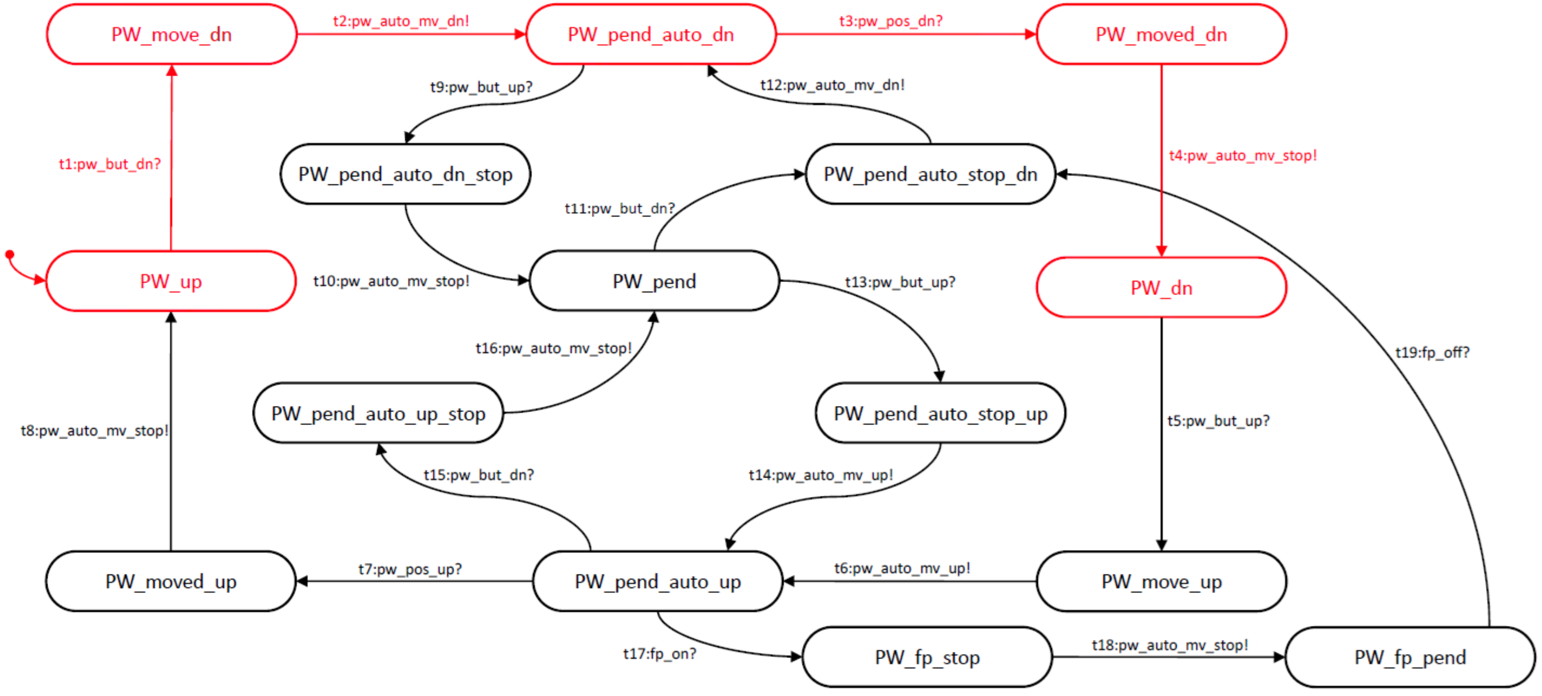


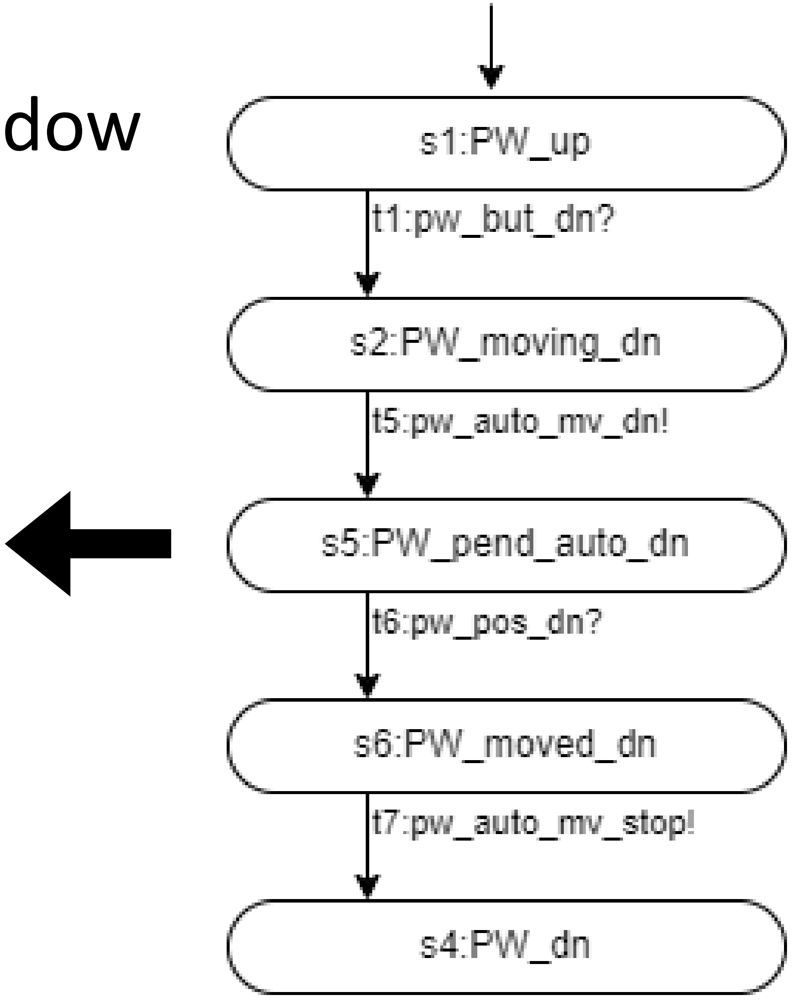
DELTA-ORIENTED VARIABILITY



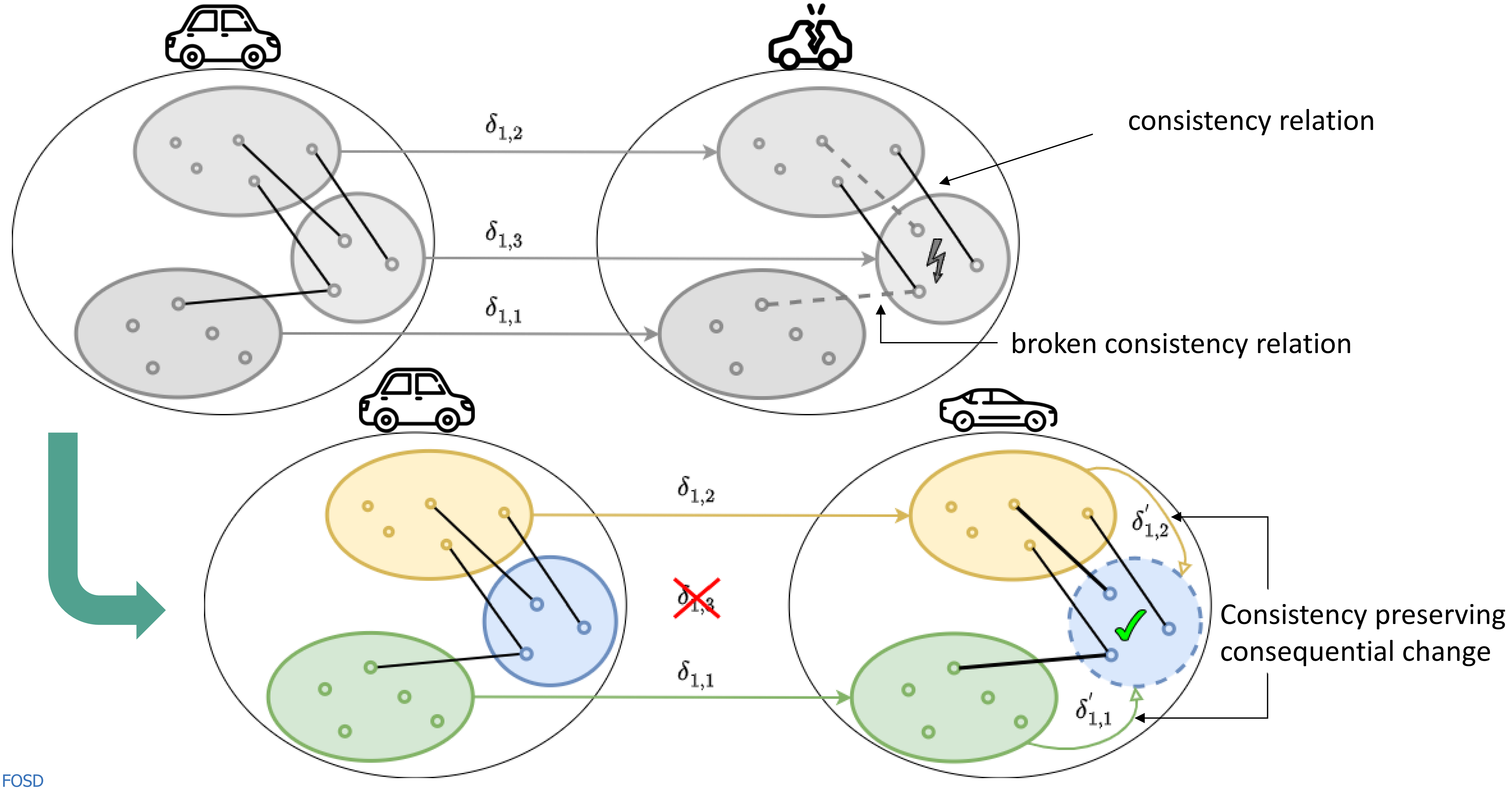
DELTA-ORIENTED VARIABILITY

P1.statemachine → region:AutomaticPowerWindow

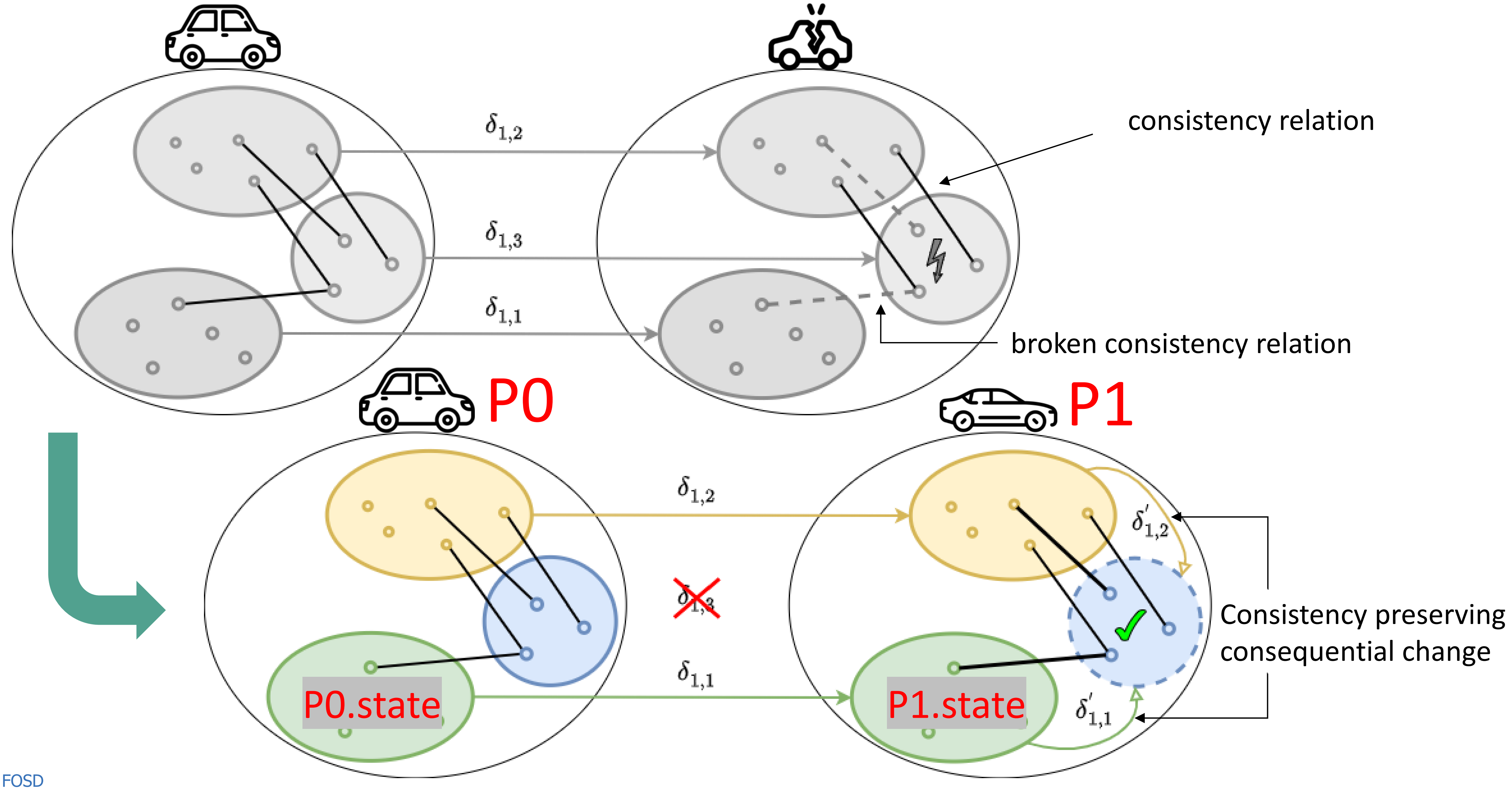




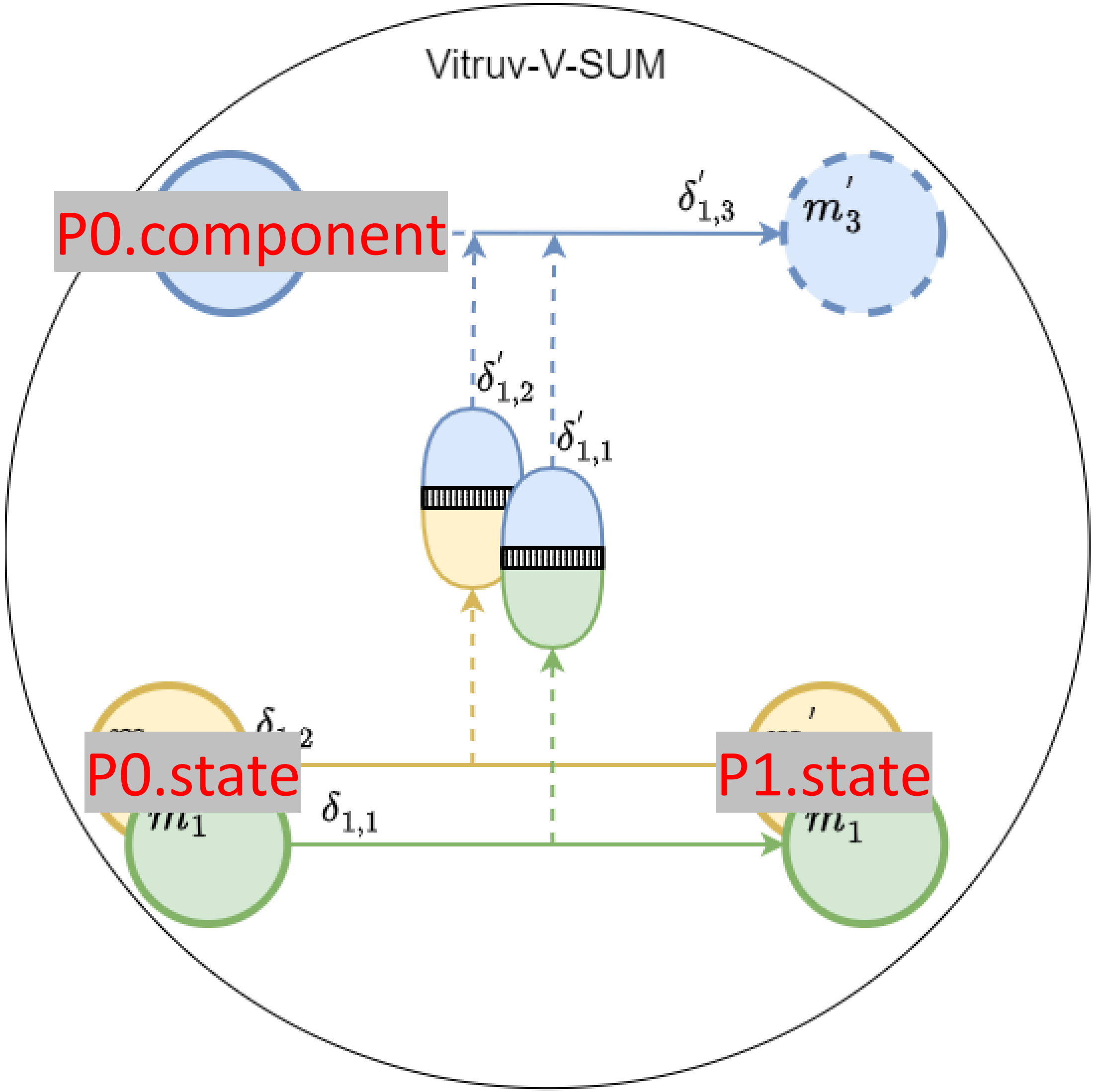
REMINDER – WE WANTED TO DO THIS



REMINDER – WE WANTED TO DO THIS



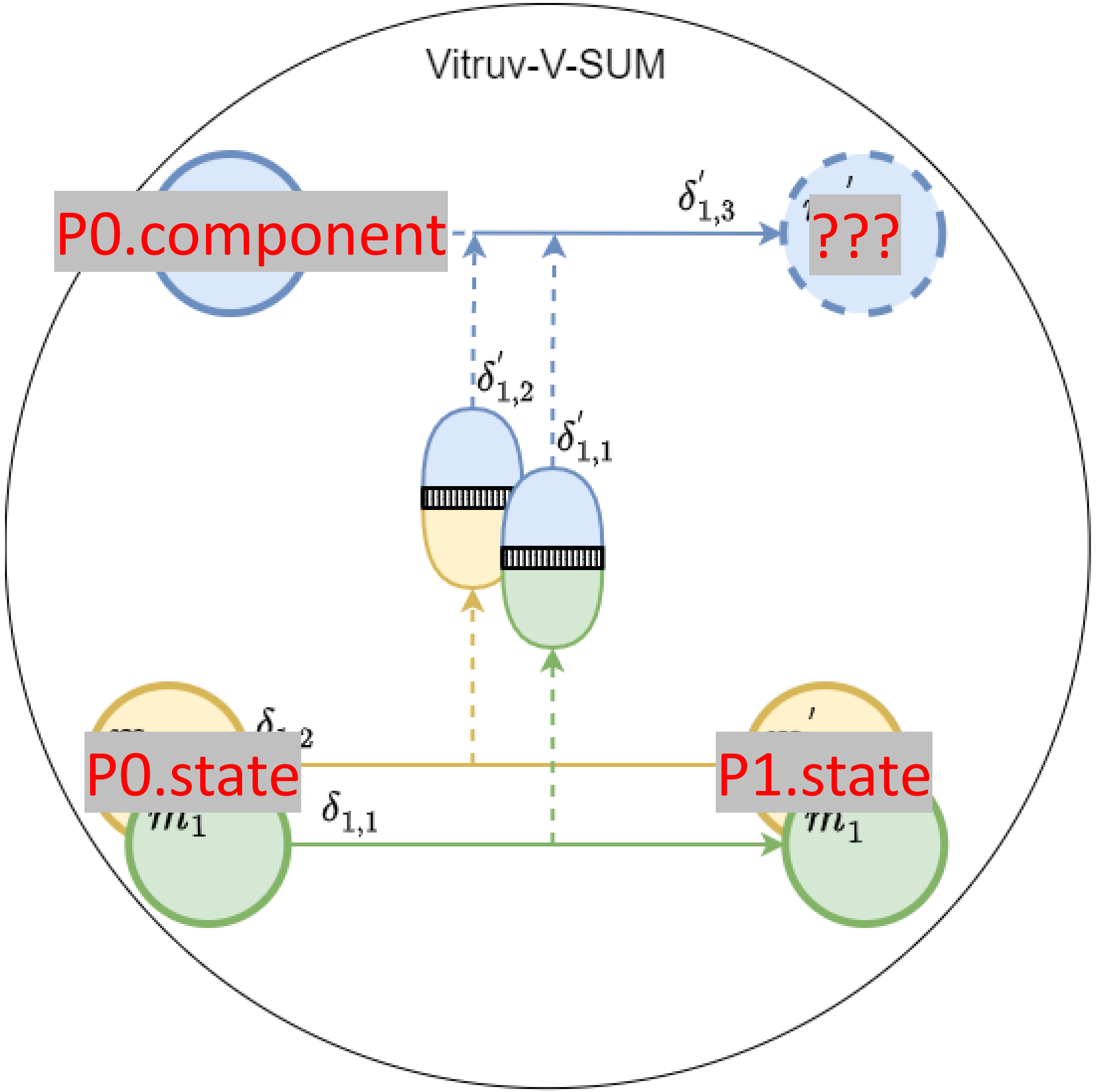
IDEA



Idea:
Using consistency preservation as a mechanism for model-driven development

Advantage:
- Getting a consistent version of model 3
- Getting a delta for further SPLE development

IDEA

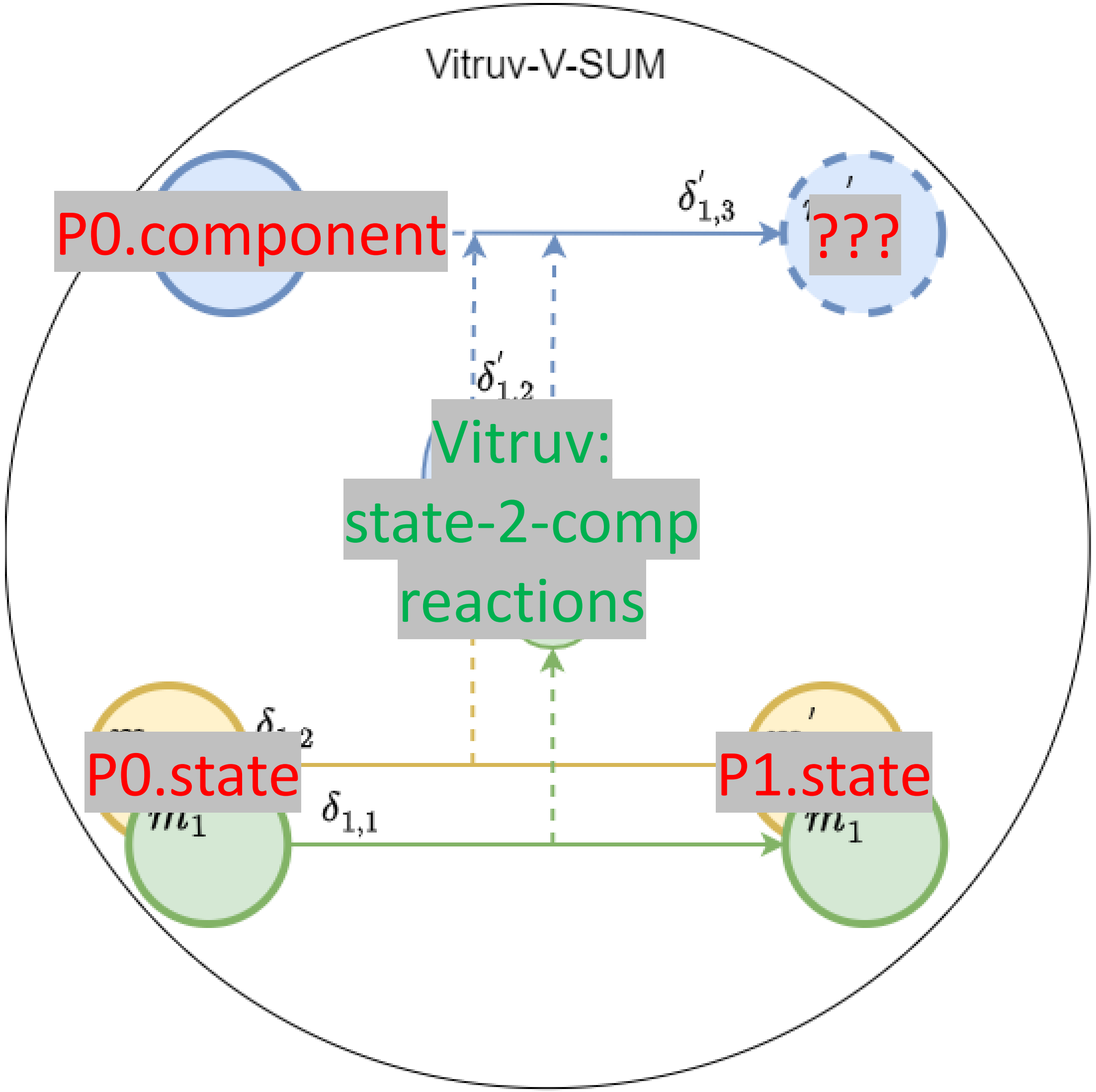


Idea:
Using consistency preservation as a mechanism for model-driven development

Advantage:

- Getting a consistent version of model 3
- Getting a delta for further SPLE development

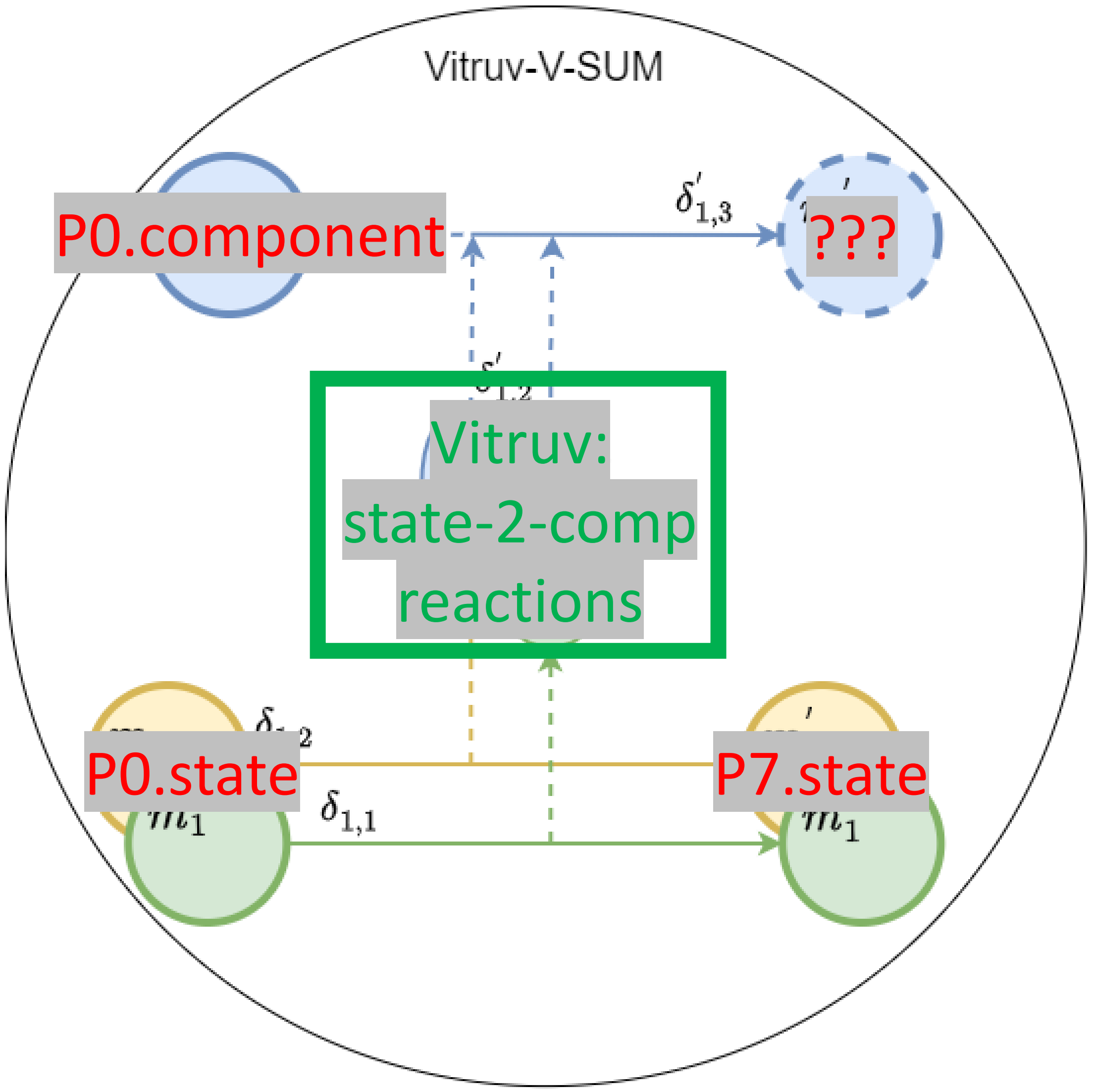
IDEA



Idea:
Using consistency preservation as a mechanism for model-driven development

Advantage:
- Getting a consistent version of model 3
- Getting a delta for further SPLE development

IDEA



Idea:
Using consistency preservation as a mechanism for model-driven development

Advantage:

- Getting a consistent version of model 3
- Getting a delta for further SPLE development

INCREMENTAL CONSISTENCY PRESERVATION

```
state-2-component.reactions
import state-metamodel as STATE;
import component-metamodel as COMP;
```

Reference Domains

INCREMENTAL CONSISTENCY PRESERVATION

state-2-component.reactions

```
import state-metamodel as STATE;
import component-metamodel as COMP;
reaction AddedTransition {
    after attribute added STATE:Region[transitions]

}
```

Define Trigger

INCREMENTAL CONSISTENCY PRESERVATION

state-2-component.reactions

```
import state-metamodel as STATE;
import component-metamodel as COMP;
reaction AddedTransition {
    after attribute added STATE:Region[transitions]
    call updatePortsOfComponent(affectedEObject, newValue)
}
routine updatePortsOfComponent(STATE::Region smRegion, String newTransition) {
    match {


    }
    update {





    }
}
```

Call Repair Routine

Define Repair Routine

INCREMENTAL CONSISTENCY PRESERVATION

state-2-component.reactions

```
import state-metamodel as STATE;
import component-metamodel as COMP;
reaction AddedTransition {
    after attribute added STATE:Region[transitions]
    call updatePortsOfComponent(affectedEObject, newValue)
}
routine updatePortsOfComponent(STATE::Region smRegion, String newTransition) {
    match {
                Ensure Preconditions
    }
    update {
                Execute Repair
    }
}
```

INCREMENTAL CONSISTENCY PRESERVATION

state-2-component.reactions

```
import state-metamodel as STATE;
import component-metamodel as COMP;
reaction AddedTransition {
    after attribute added STATE:Region[transitions]
    call updatePortsOfComponent(affectedEObject, newValue)
}
routine updatePortsOfComponent(STATE::Region smRegion, String newTransition) {
    match {
        check { newTransition.signal != null }

    }
    update {





    }
}
```

INCREMENTAL CONSISTENCY PRESERVATION

state-2-component.reactions

```
import state-metamodel as STATE;
import component-metamodel as COMP;
reaction AddedTransition {
    after attribute added STATE:Region[transitions]
    call updatePortsOfComponent(affectedEObject, newValue)
}
routine updatePortsOfComponent(STATE::Region smRegion, String newTransition) {
    match {
        check { newTransition.signal != null }
        val component = retrieve COMP:Component corresponding to smRegion
    }
    update {




    }
}
```

Management of Connected Entities

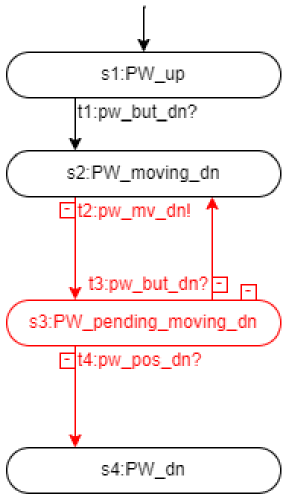
INCREMENTAL CONSISTENCY PRESERVATION

state-2-component.reactions

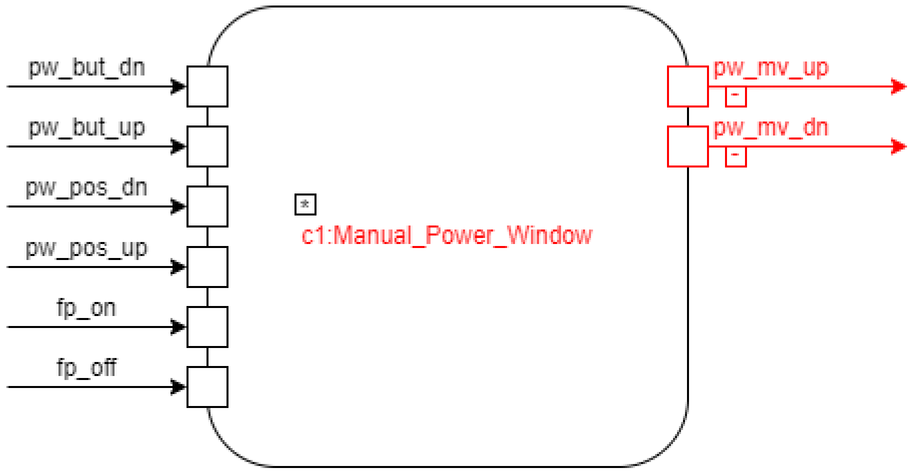
```
import state-metamodel as STATE;
import component-metamodel as COMP;
reaction AddedTransition {
    after attribute added STATE:Region[transitions]
    call updatePortsOfComponent(affectedEObject, newValue)
}
routine updatePortsOfComponent(STATE::Region smRegion, String newTransition) {
    match {
        check { newTransition.signal != null }
        val component = retrieve COMP:Component corresponding to smRegion
    }
    update {
        if (newTransition.sendsSignal()) {
            components.outputPorts.add(new Port(newTransition.signal))
        } else if (newTransition.expectsSignal()) {
            components.inputPorts.add(new Port(newTransition.signal))
        }
    }
}
```

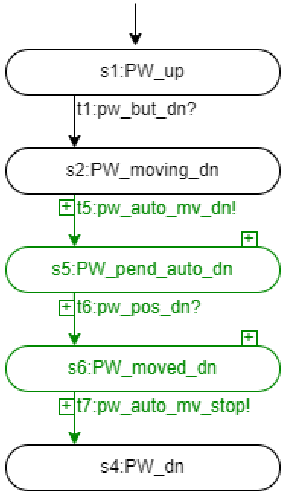
Repair

SYNTHESIS OF COMPONENT DELTAS

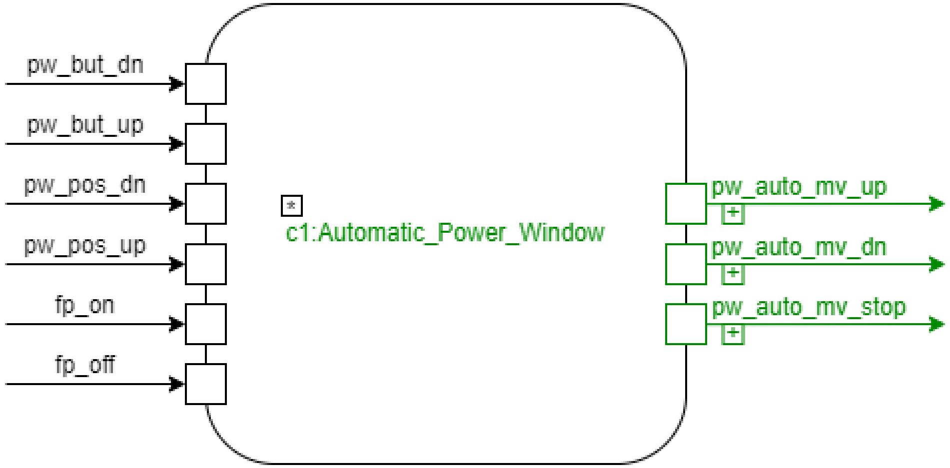


state-2-component
.reactions

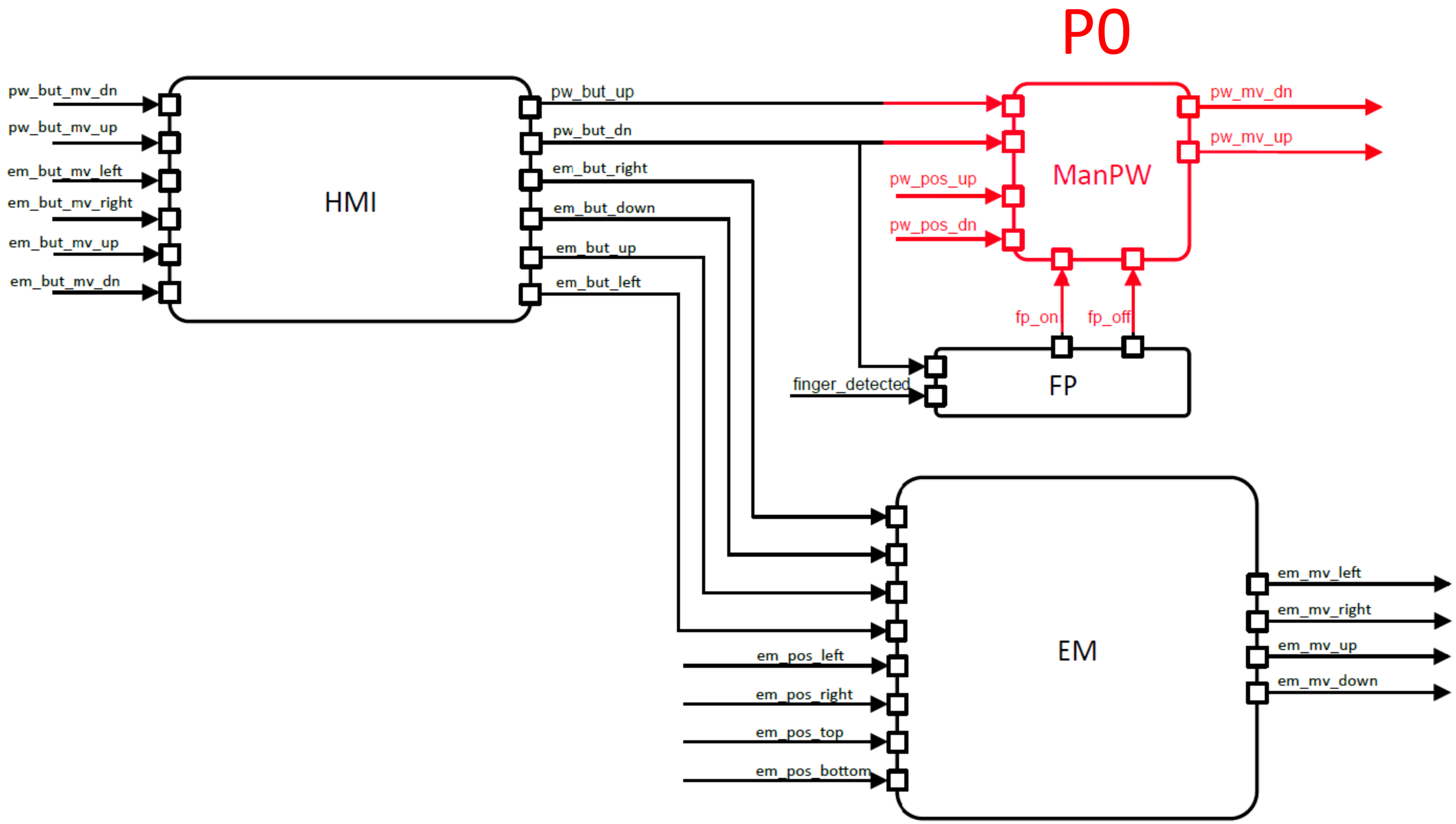




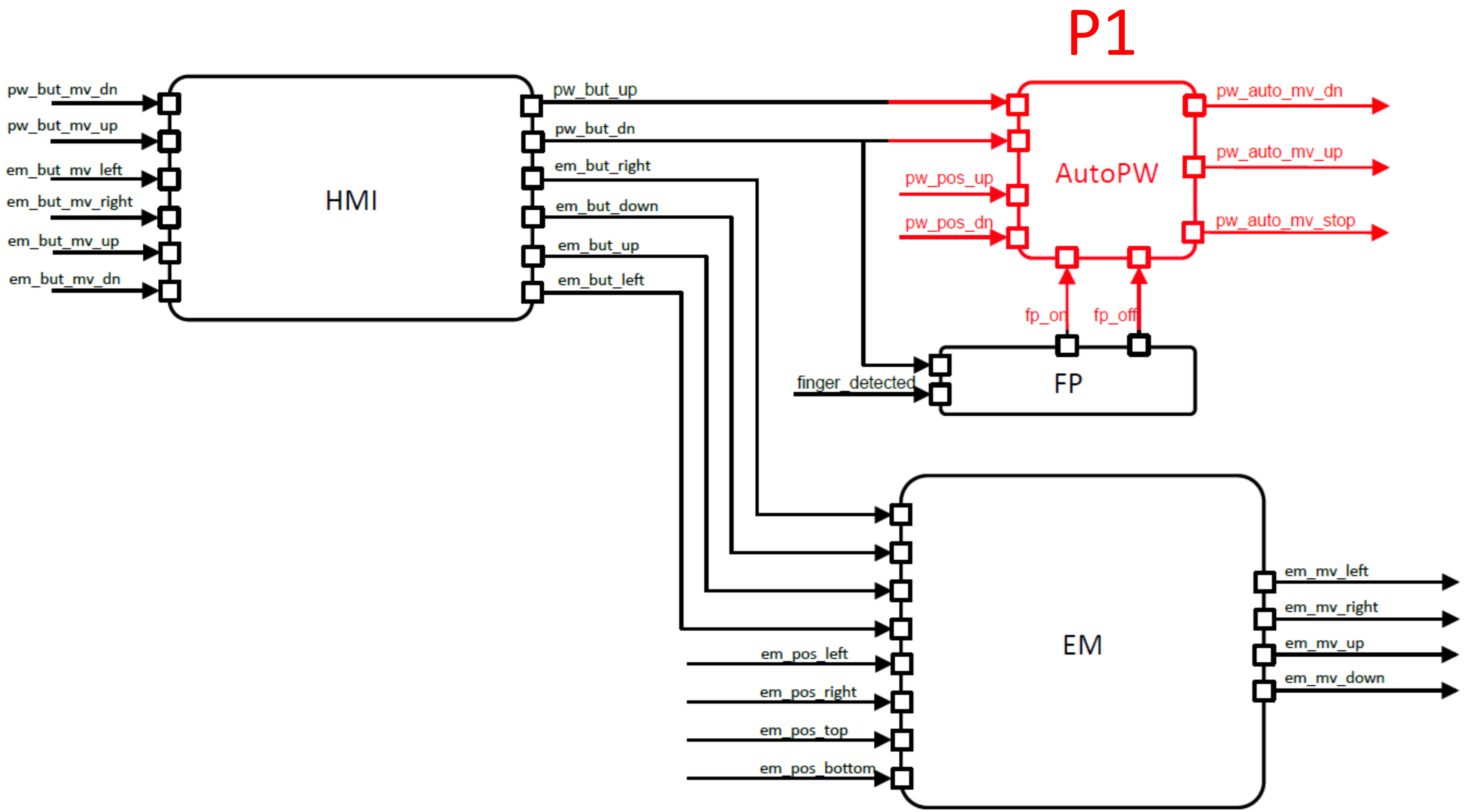
state-2-component
.reactions



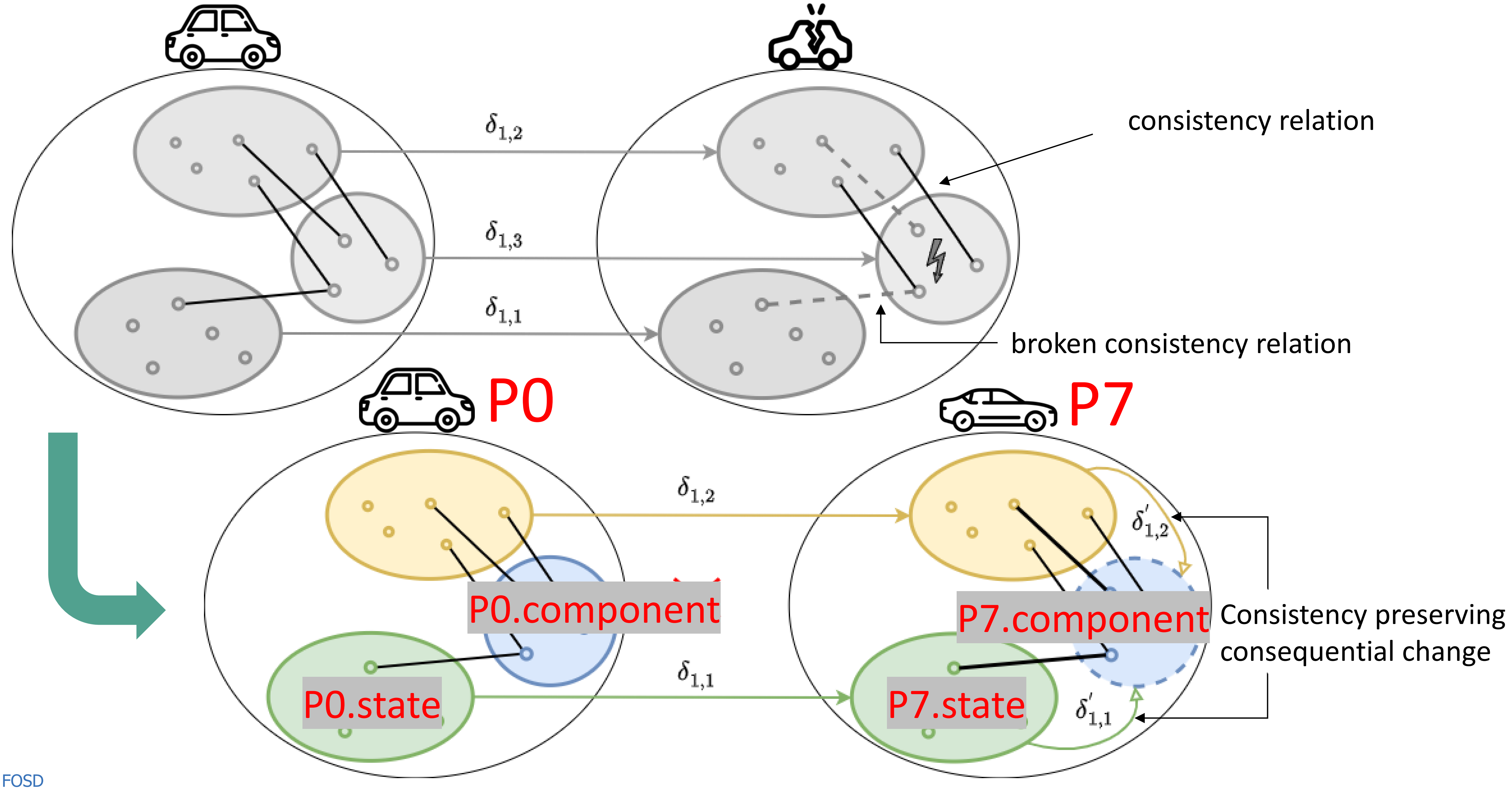
BACK TO THE BIG PICTURE



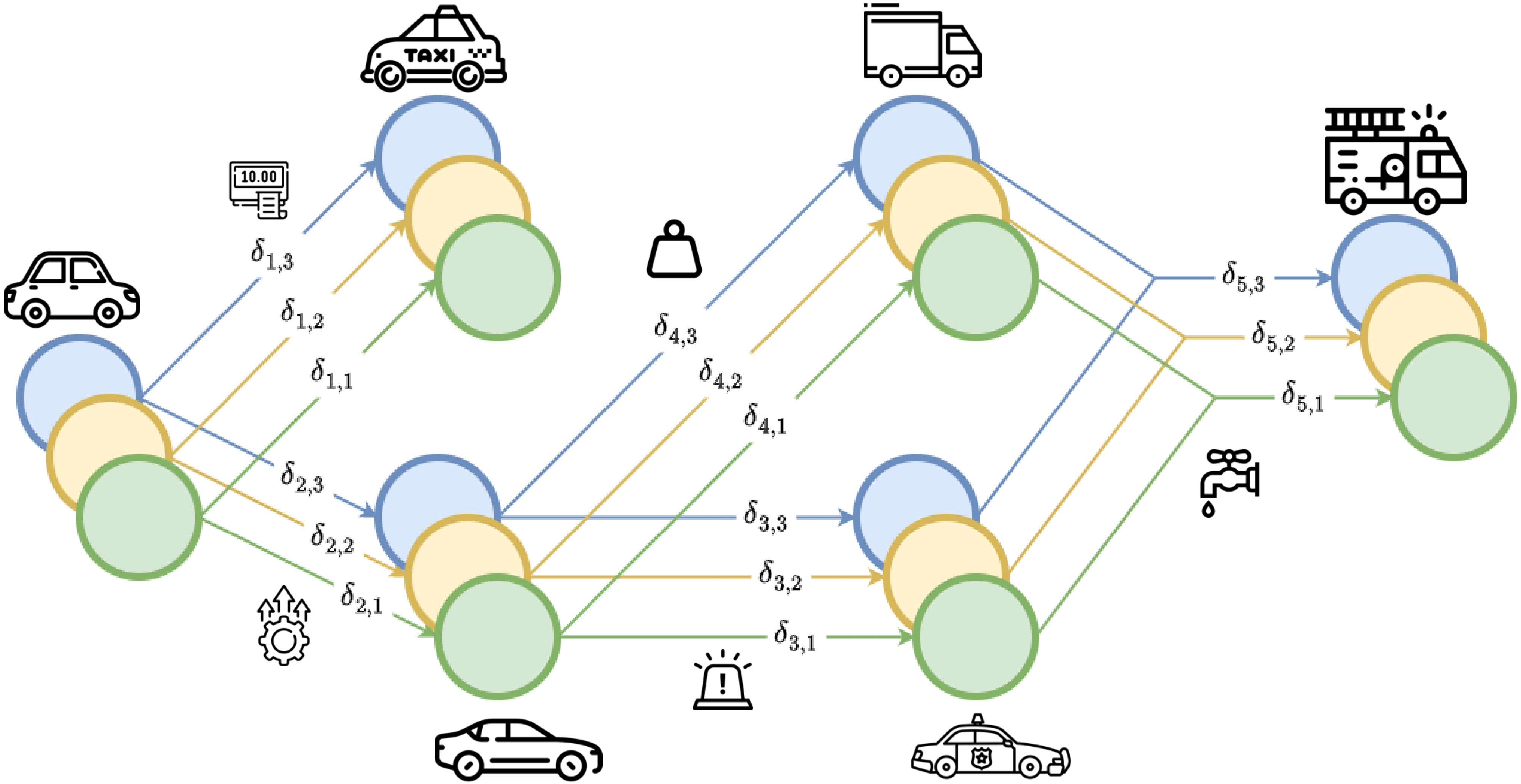
BACK TO THE BIG PICTURE



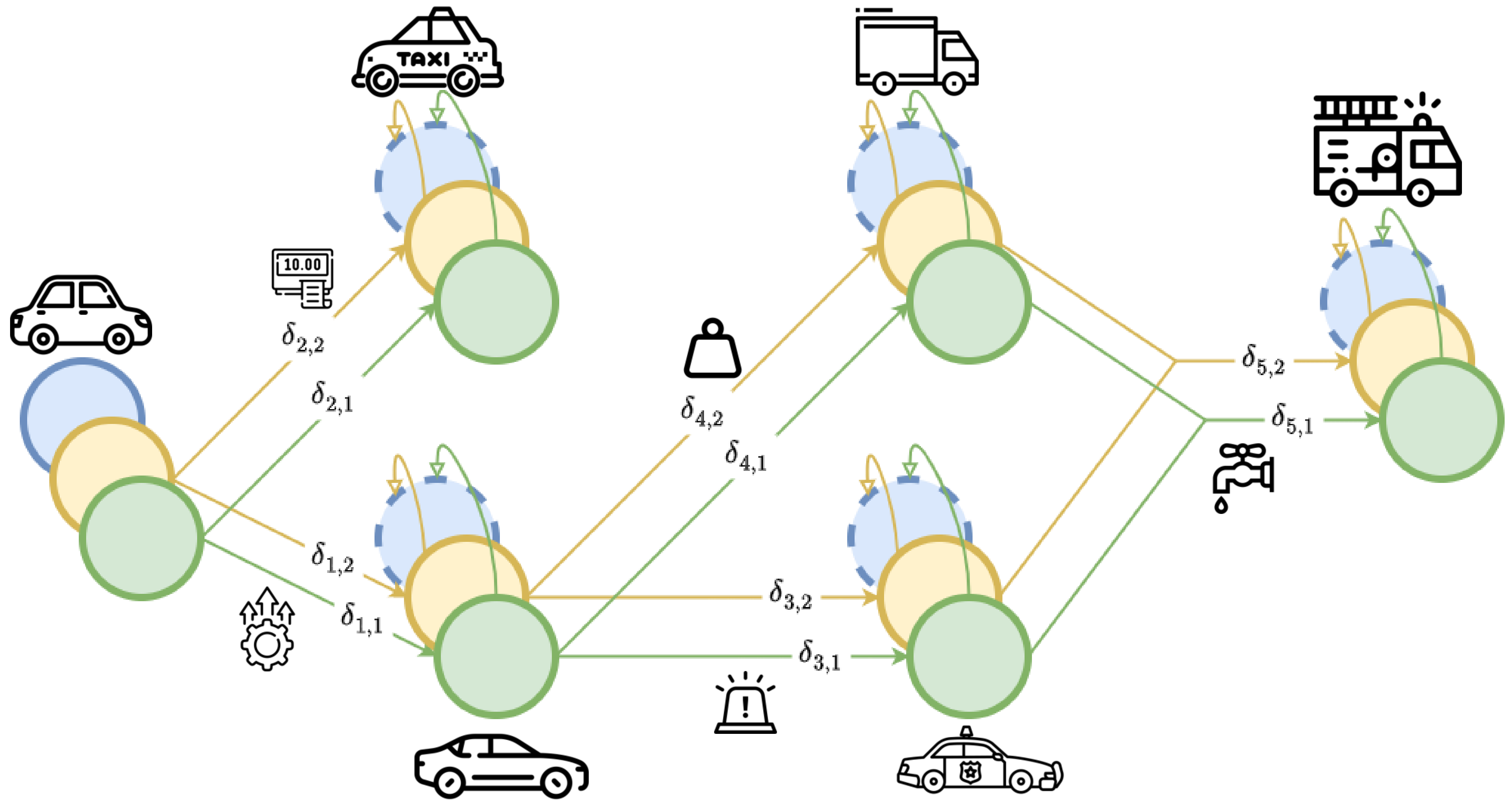
REMINDER – WE WANTED TO DO THIS



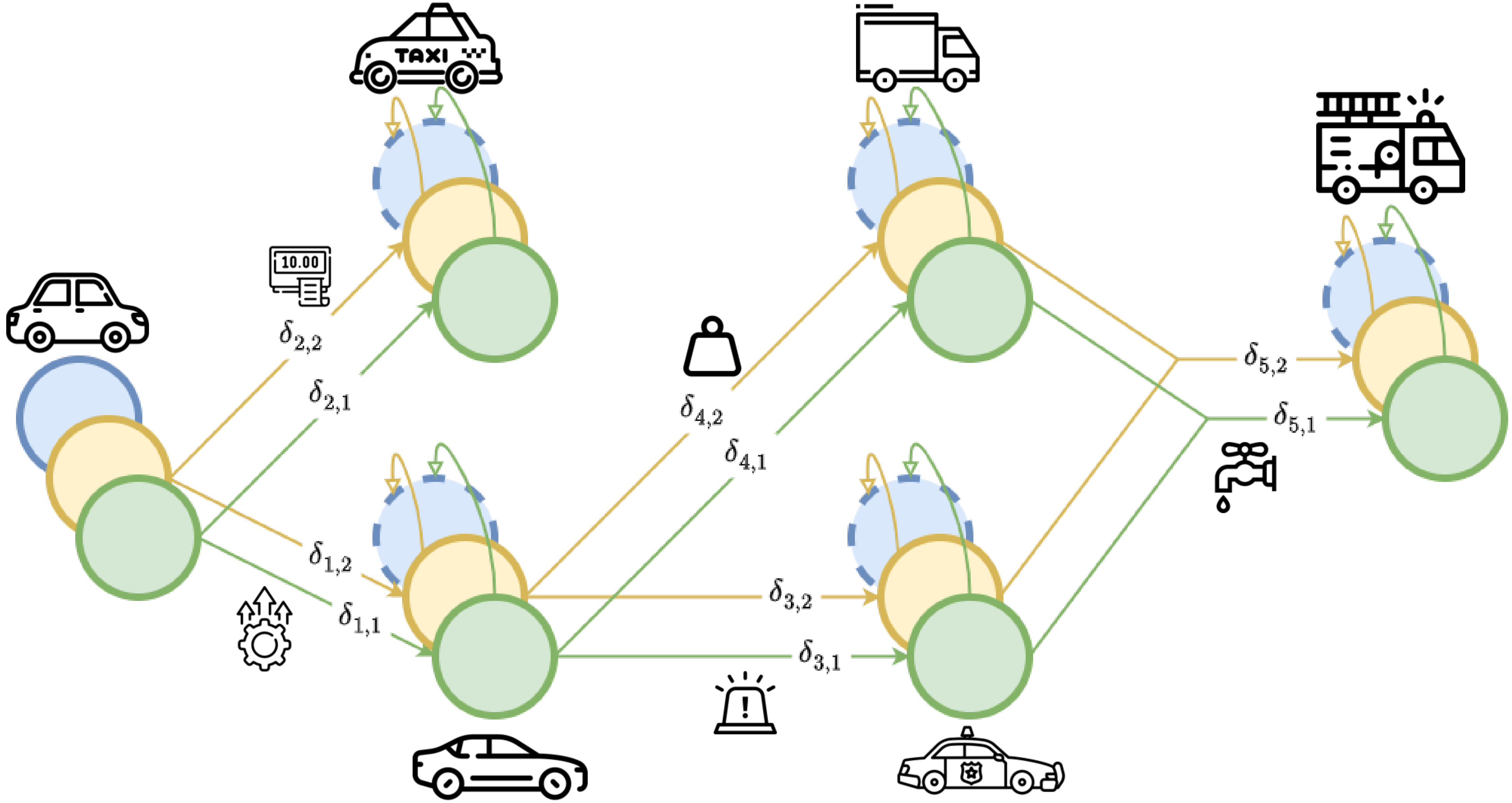
MANUAL DEVELOPMENT



USING CONSISTENCY PRESERVATION



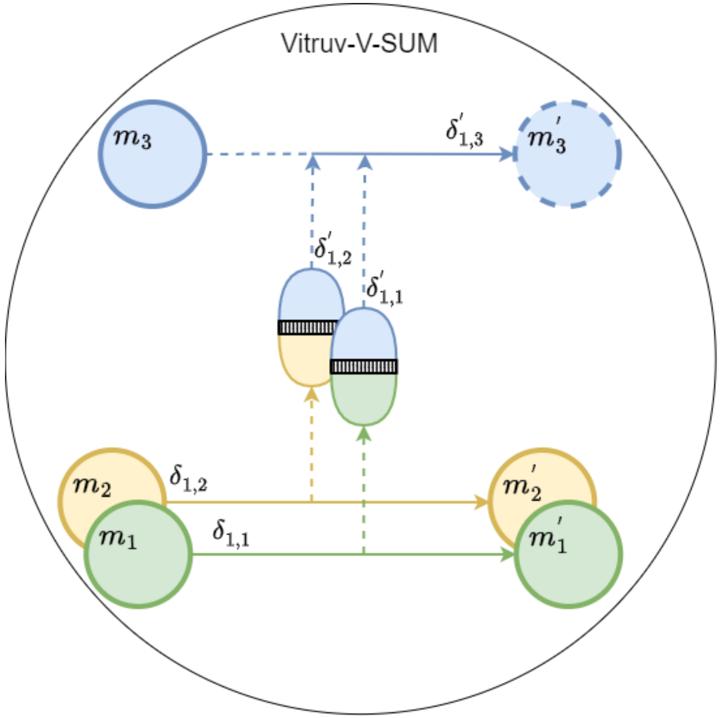
USING CONSISTENCY PRESERVATION



Consistency preservation simplifies delta-oriented software product line engineering.

SUMMARY

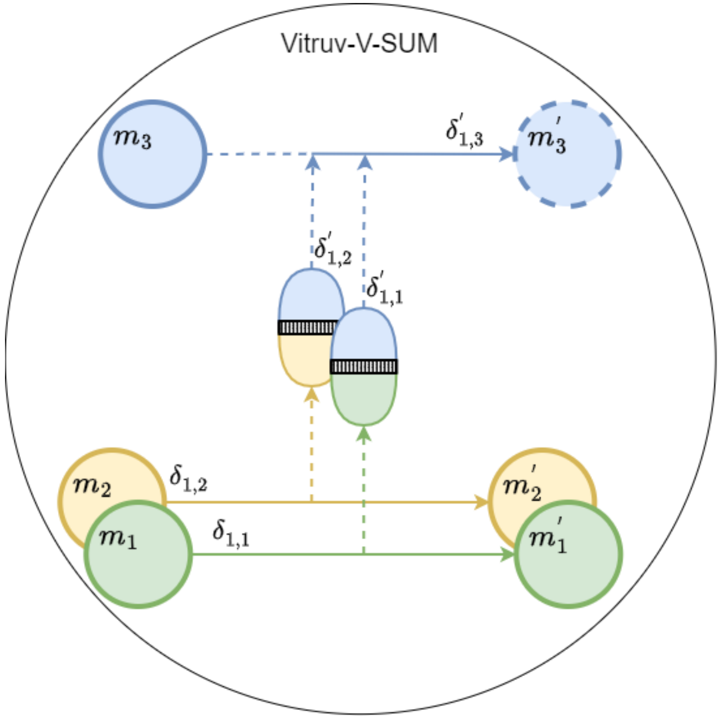
Using Consistency Preservation

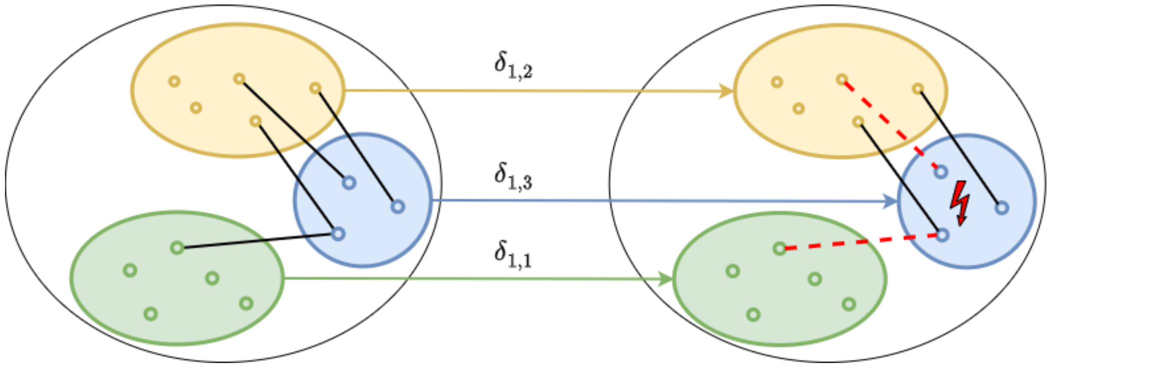



```
state-2-component.reactions
import STATE, COMP;
reaction AddedTransition {
    after attribute added STATE:Region[transitions]
    call updatePortsOfComponent(affectedEObject, newValue)
}
routine updatePortsOfComponent(STATE::Region smRegion, String newTransition) {
    match {
        check { newTransition.signal != null }
        val component = retrieve COMP:Component corresponding to smRegion
    }
    update { … }
}
```

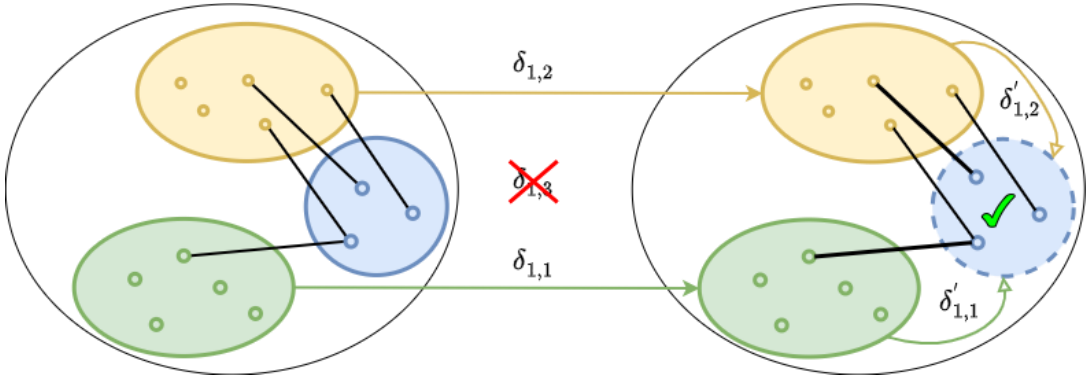
SUMMARY

Using Consistency Preservation



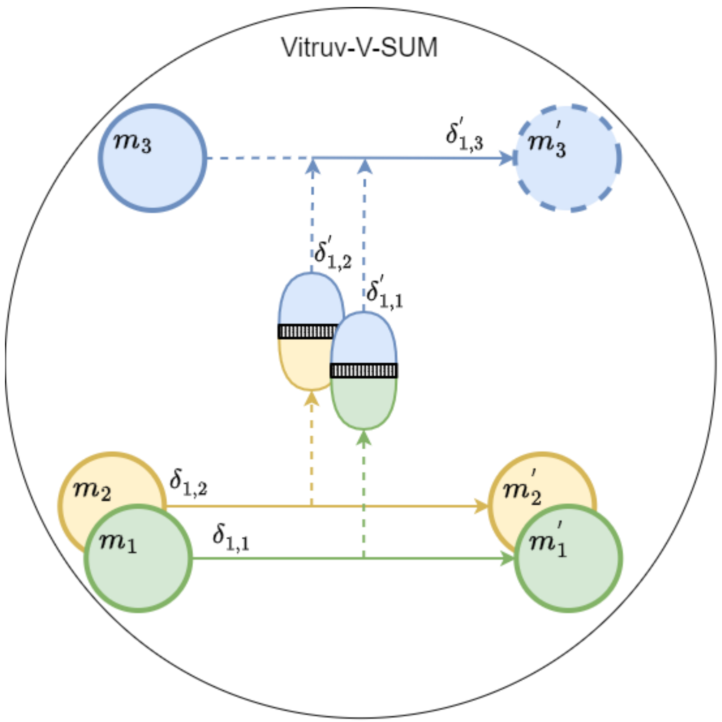


for Delta-Oriented

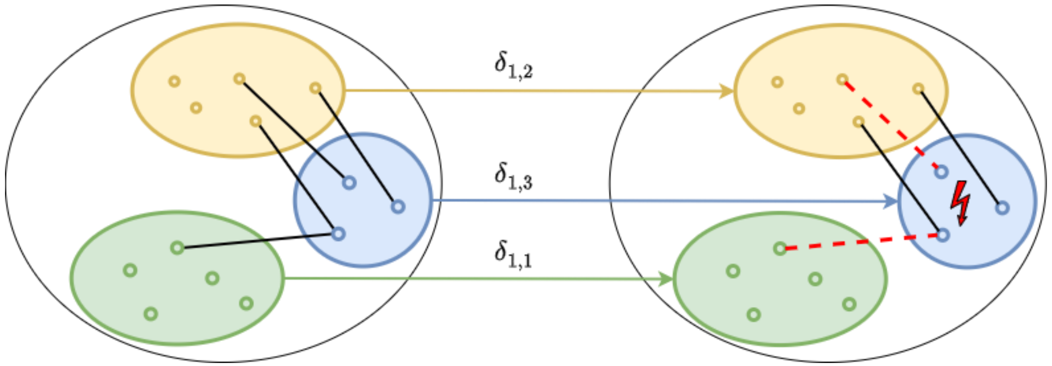



```
state-2-component.reactions
import STATE, COMP;
reaction AddedTransition {
    after attribute added STATE:Region[transitions]
    call updatePortsOfComponent(affectedEObject, newValue)
}
routine updatePortsOfComponent(STATE::Region smRegion, String newTransition) {
    match {
        check { newTransition.signal != null }
        val component = retrieve COMP:Component corresponding to smRegion
    }
    update { … }
}
```

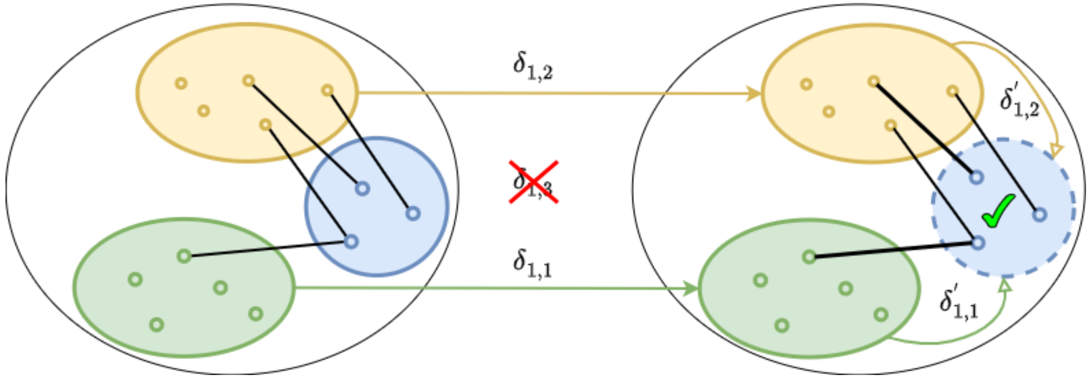
SUMMARY



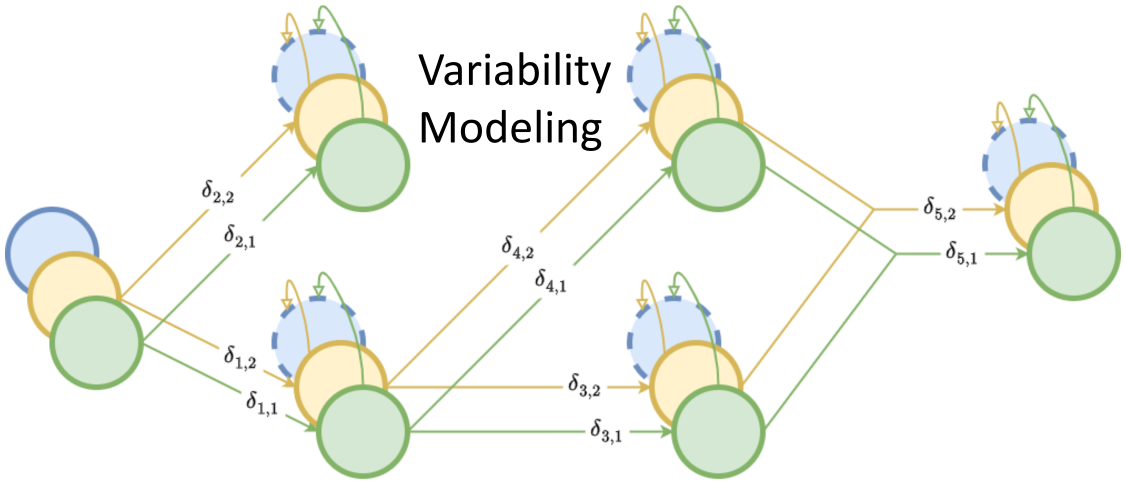
Using Consistency Preservation



for Delta-Oriented



Variability Modeling




```
state-2-component.reactions
import STATE, COMP;
reaction AddedTransition {
    after attribute added STATE:Region[transitions]
    call updatePortsOfComponent(affectedEObject, newValue)
}
routine updatePortsOfComponent(STATE::Region smRegion, String newTransition) {
    match {
        check { newTransition.signal != null }
        val component = retrieve COMP:Component corresponding to smRegion
    }
    update { … }
}
```