Software Product Lines with Design Choices: Reasoning about Variability and Design Uncertainty

Michalis Famelis, Julia Rubin, Krzysztof Czarnecki, Rick Salay, Marsha Chechik

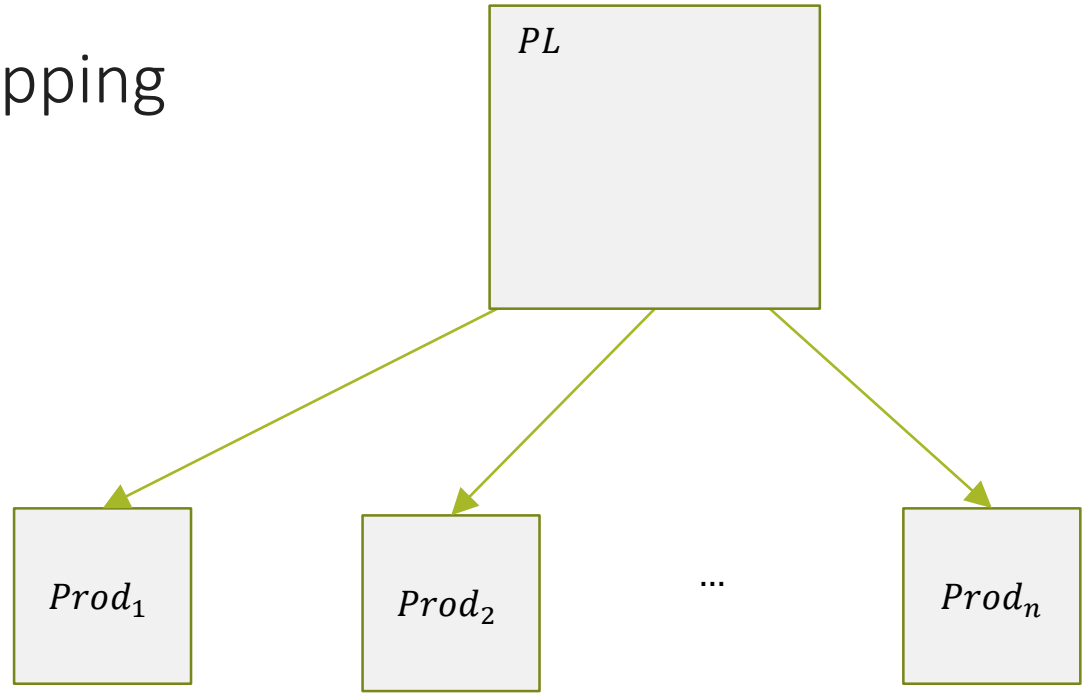
Université de Montréal
THE UNIVERSITY OF BRITISH COLUMBIA
UNIVERSITY OF WATERLOO
UNIVERSITY OF TORONTO

Software Product Line Basics

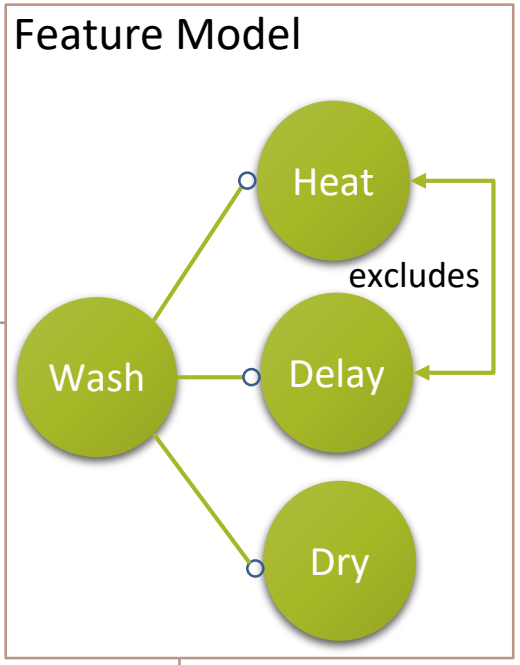
Feature model, Domain model, Feature mapping

Different feature **configurations** result in different variants

PL models a **set** of related, but different products



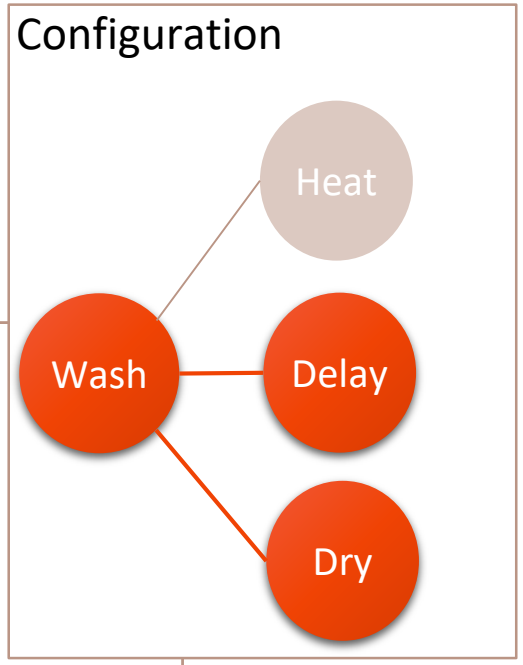
Washing Machine Product Line



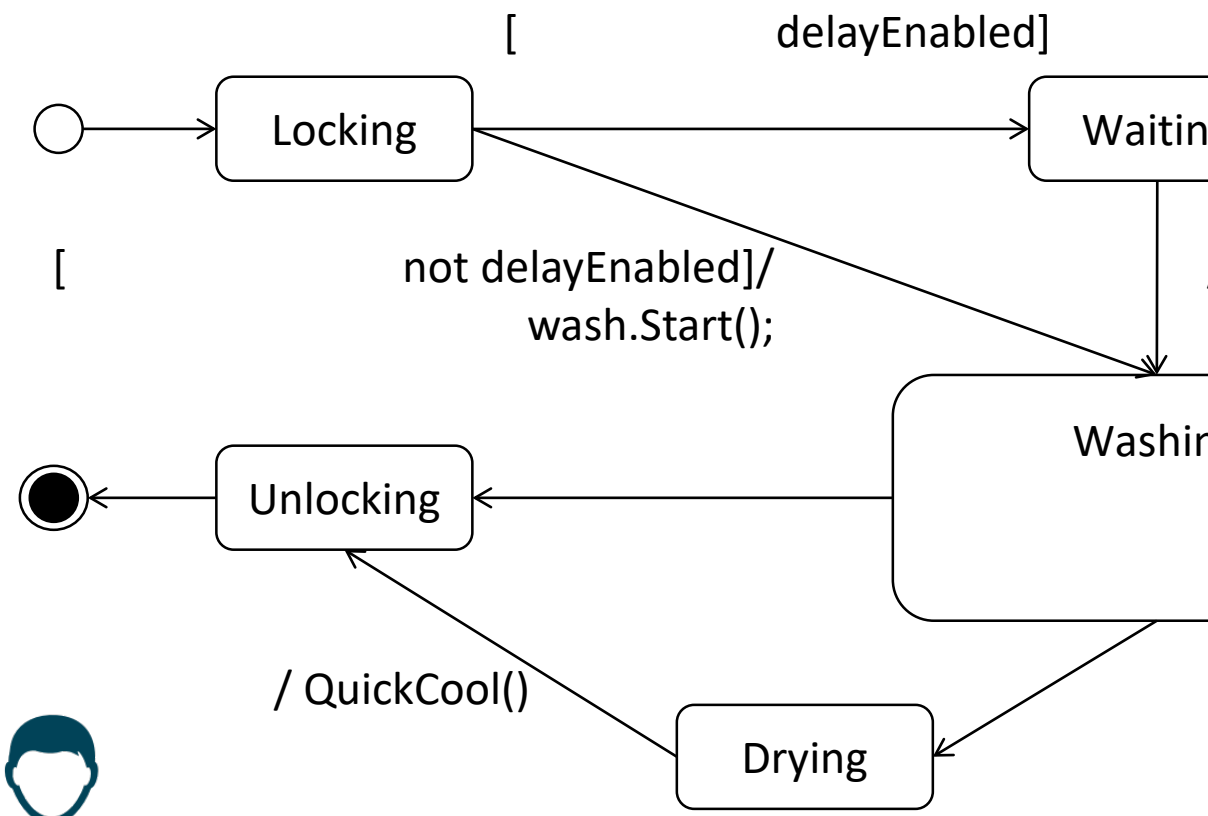
Domain Model

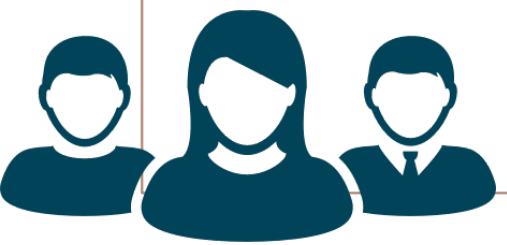
Heat Delay
[heatEnabled;delayEnabled]
/ HeaterOn() Heat
Heat V Delay
Heat V Delay
Heat Delay
[not heatEnabled;not delayEnabled]/
wash.Start();
Heat V Delay
/ HeaterOff(); Heat
wash.Start(); Heat V Delay
Heat
/ QuickCool()
¬Dry
Dry
Dry / QuickCool()
Dry
Dry
Locking
Waiting
Unlocking
Washing
entry/TempCheck() Heat
Drying

Variant with Dry and Delay

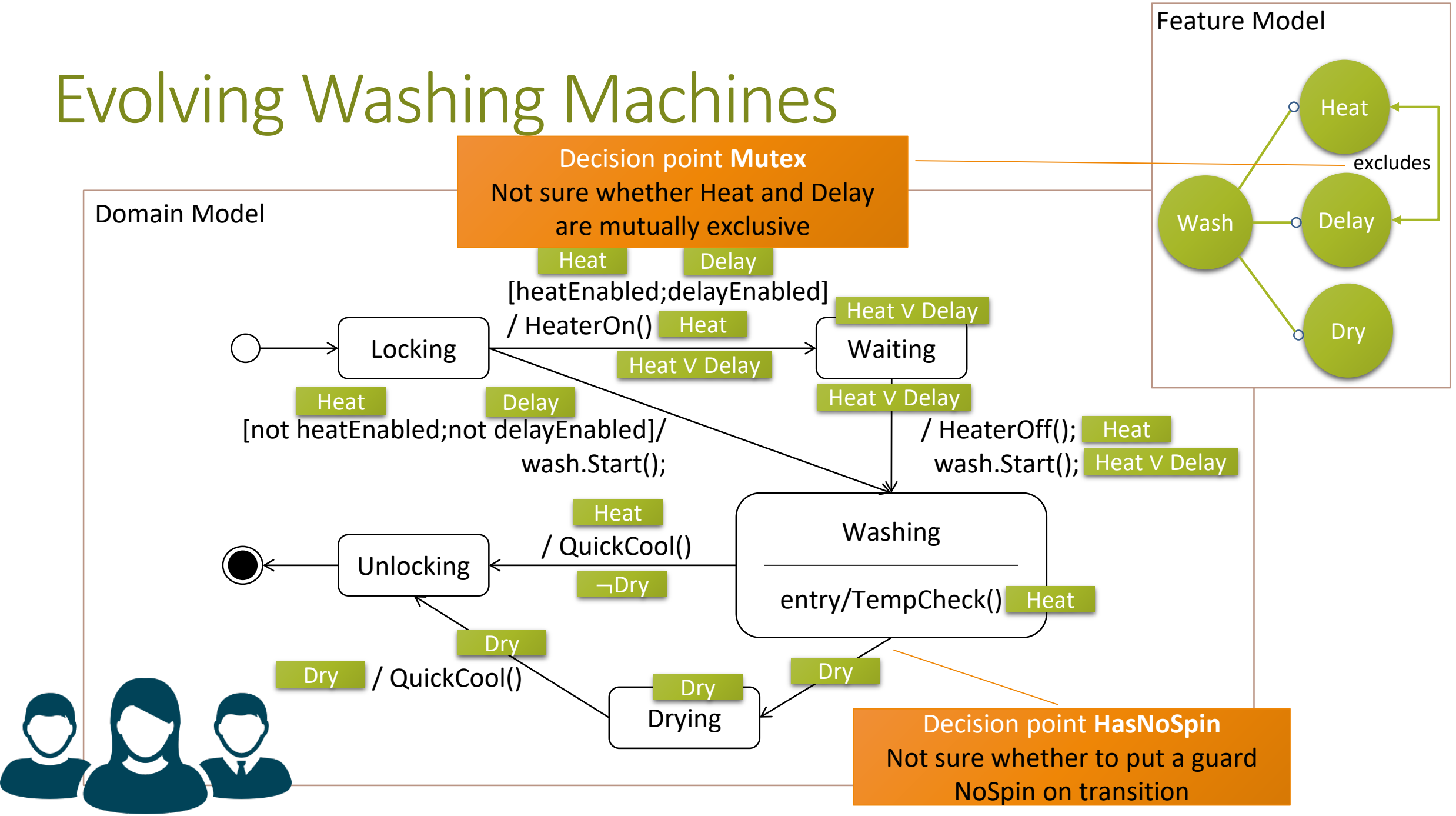


Model without variability

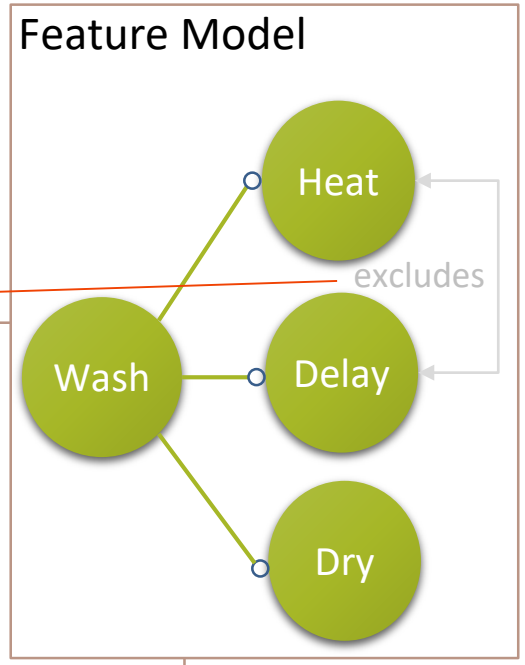




Evolving Washing Machines



Design 1: No Mutex or HasSpin

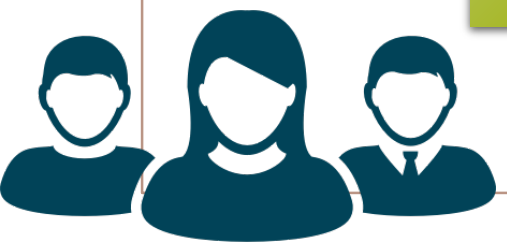


Heat and Delay not mutually exclusive

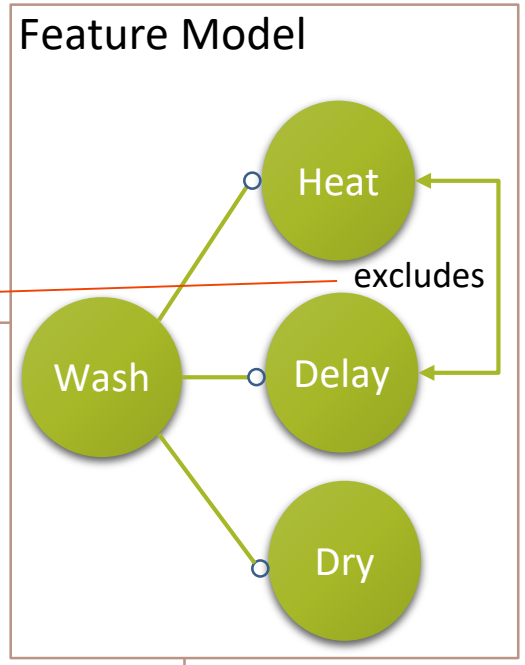
Domain Model

Heat
Delay
[heatEnabled;delayEnabled]
/ HeaterOn()
Heat
Heat V Delay
Locking
Waiting
Heat V Delay
Heat
Delay
[not heatEnabled;not delayEnabled]/
wash.Start();
Heat V Delay
/ HeaterOff();
Heat
wash.Start();
Heat V Delay
Heat
/ QuickCool()
¬Dry
Unlocking
Washing
entry/TempCheck()
Heat
Dry
Dry
/ QuickCool()
Dry
Dry
Drying
Dry

No guard on the transition



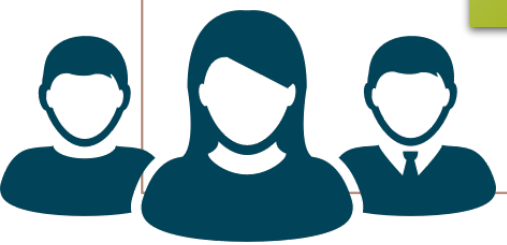
Design 2: Mutex and HasSpin



Heat and Delay mutually exclusive

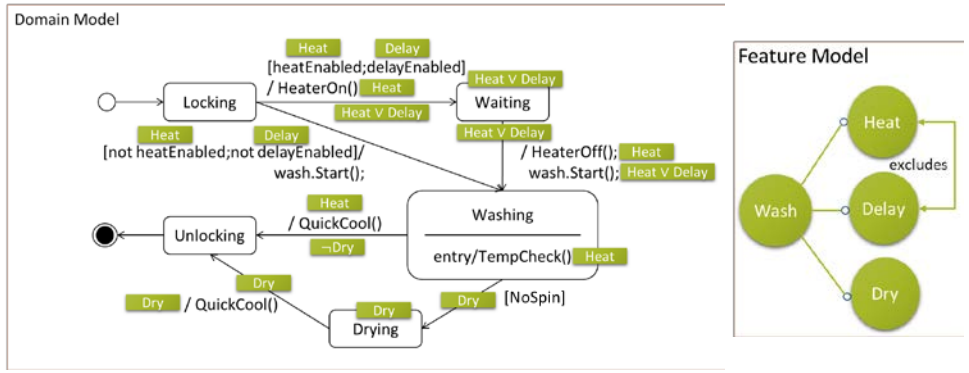
Domain Model

Heat Delay
[heatEnabled;delayEnabled]
/ HeaterOn() Heat
Heat V Delay
Locking
Waiting
Heat V Delay
Heat Delay
[not heatEnabled;not delayEnabled]/
wash.Start();
Heat V Delay
/ HeaterOff(); Heat
wash.Start(); Heat V Delay
Heat
/ QuickCool()
¬Dry
Unlocking
Washing
entry/TempCheck() Heat
Dry
Dry / QuickCool()
Dry
Drying
Dry
[NoSpin]

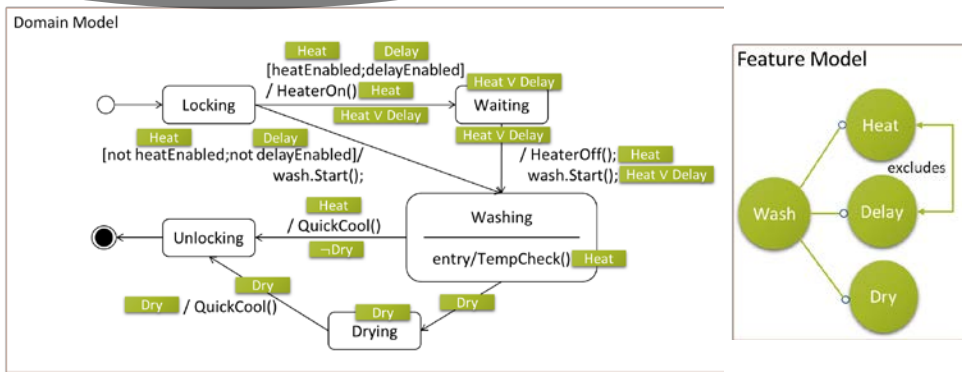


NoSpin guard on the transition

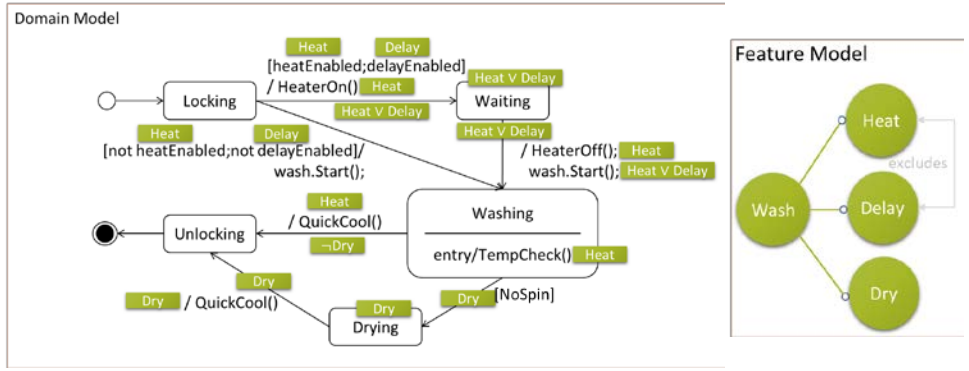
A Design Space of SPLs



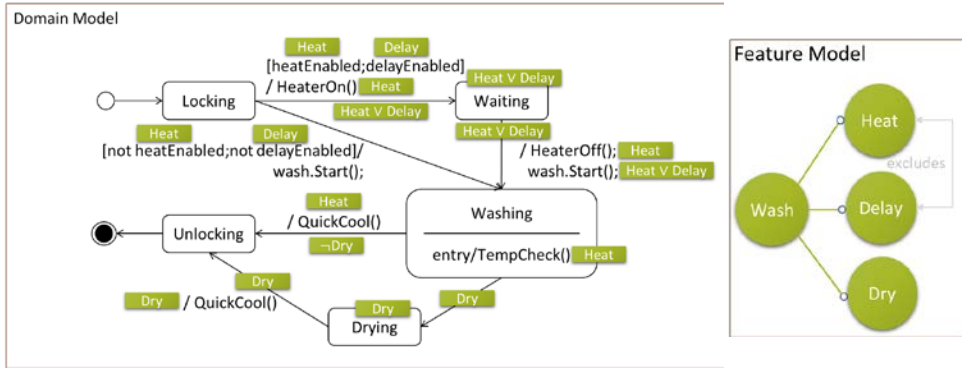
{Mutex, HasNoSpin}



{Mutex, ¬HasNoSpin}

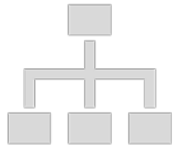


{¬Mutex, HasNoSpin}



{¬Mutex, ¬HasNoSpin}

Software Product Lines with Design Choices



Motivation

How to model this design space?

What are relevant properties for exploring it?
How to check them?

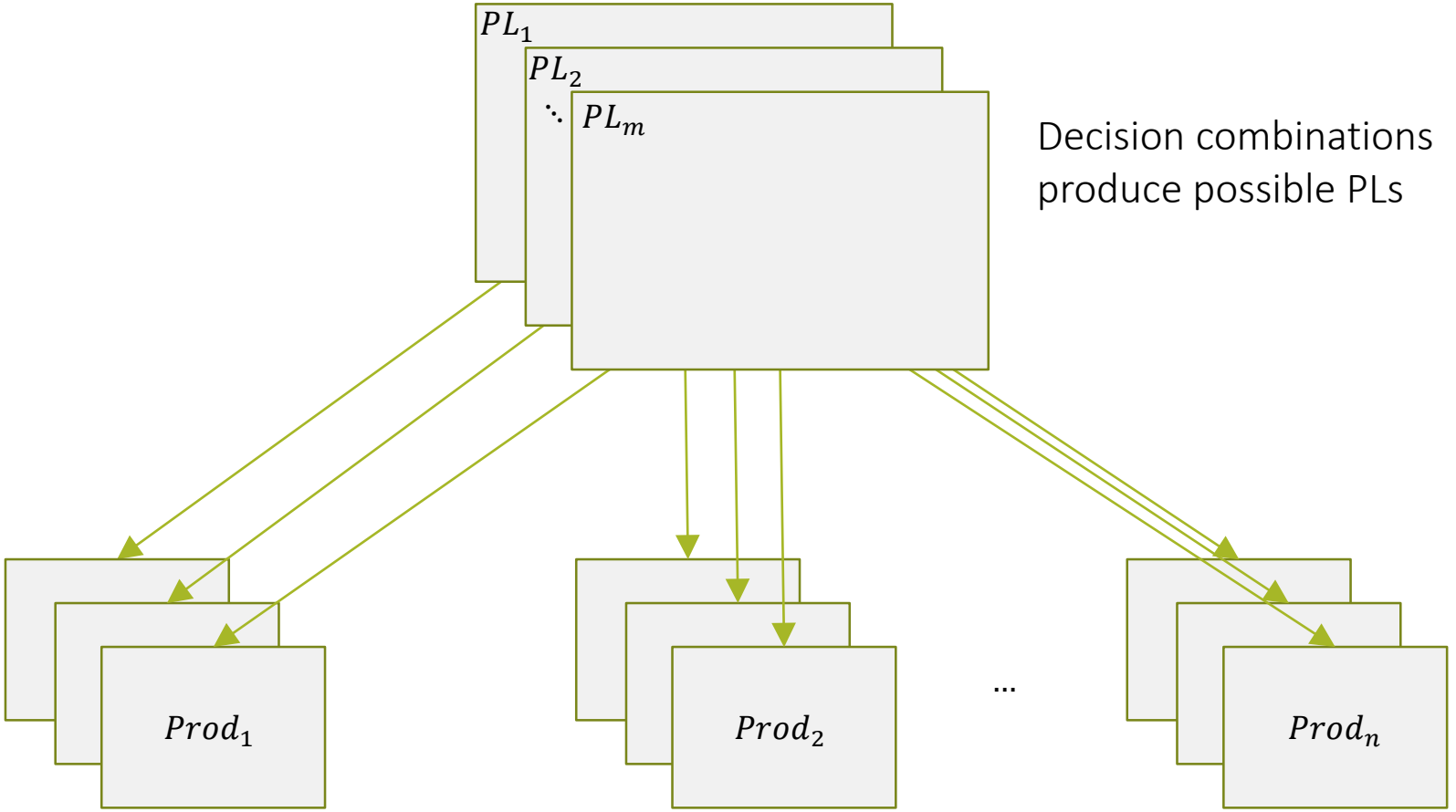
What to do when properties are violated?

Software Product Lines with Design Choices (SPLDCs)

Design choices can affect the Feature model, Domain model, Feature mapping

They define a **design space** of product lines

Given a space of product lines, which one should be selected (and why)?



Two Different Kinds of Choices

	Variability	Design Uncertainty
Reason	Market demands for product variants	Incomplete information, design alternatives, stakeholder conflicts, etc.
Granularity	Features	Decisions
Expression	Product line (PL) models	Partial models
Semantics	Set of artifacts produced by combinations of features	Set of artifacts produced by combinations of decisions
Horizon	Long term	Short term

Modelling SPLDCs

Potential candidates:

Clafer : textual structural modelling language with native variability abstractions

PEoPL: product line implementation in Jetbrains MPS

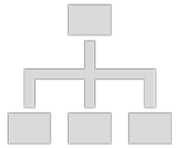
MU-MMINT: implementation of partial modelling in Eclipse

Main challenges:

Technological integration of tools

Identification of usable syntax for intuitively expressing the two concerns

Software Product Lines with Design Choices



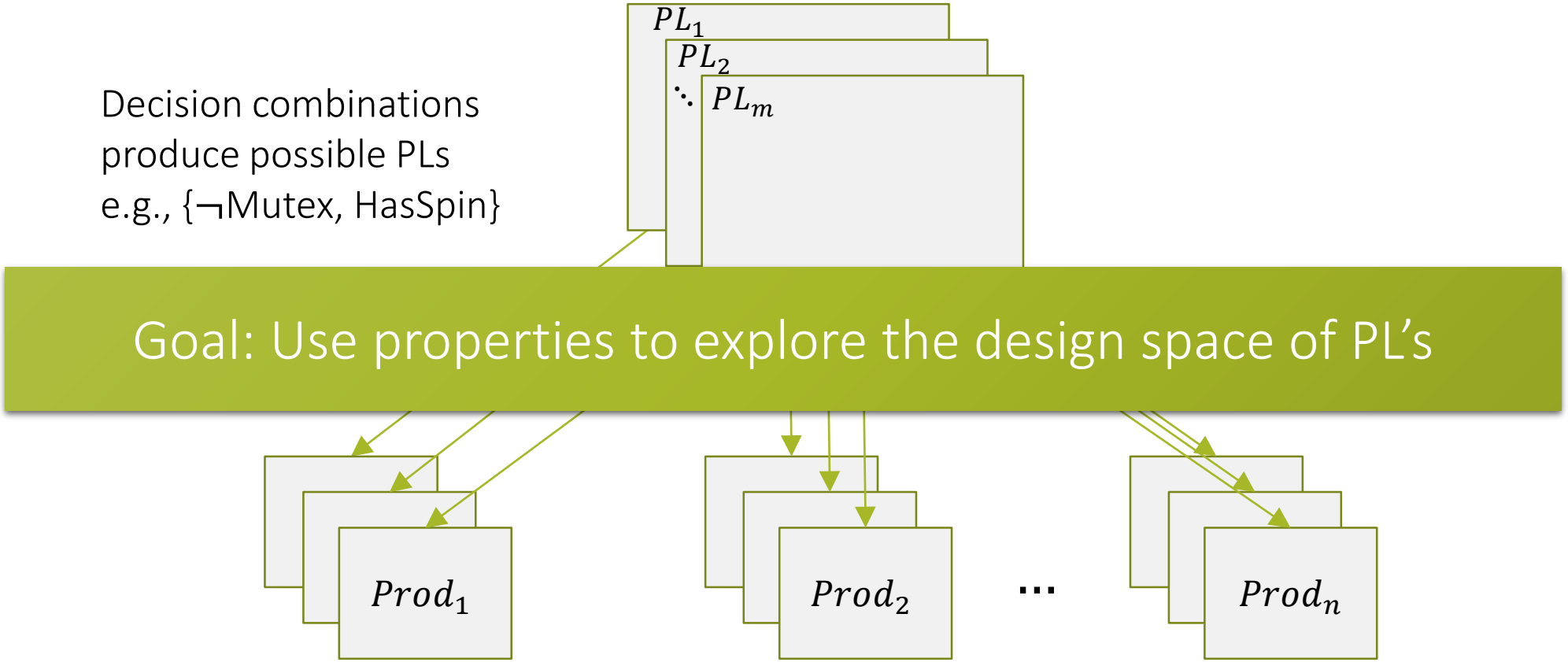
Motivation

How to model this design space?

What are relevant properties for exploring it?
How to check them?

What to do when properties are violated?

A Design Space of Product Lines



Constraining the Design Space using Properties

For a product-level property P, we define four SPLDC-level properties using the modalities:

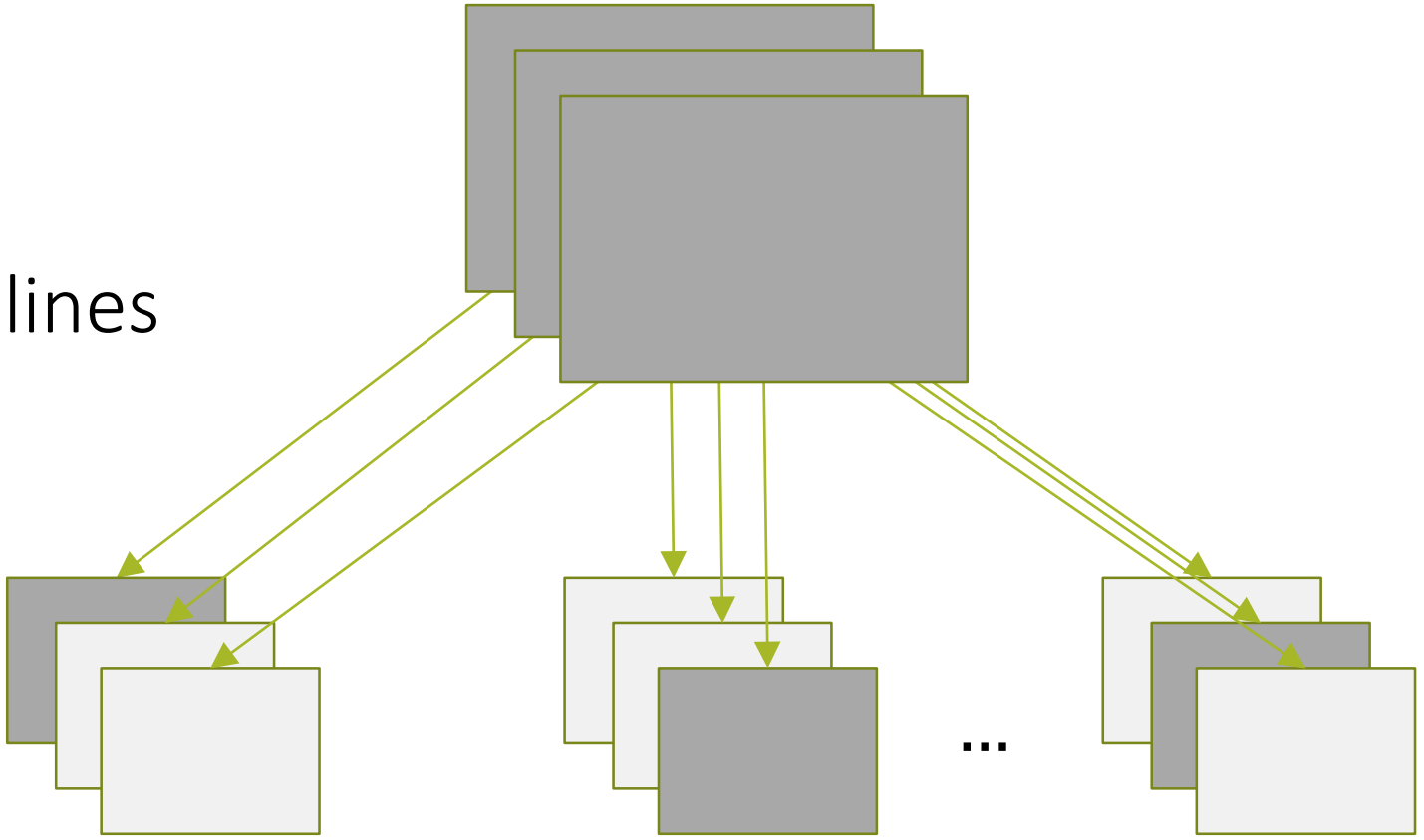
- Use All for critical properties and Some for desirable properties
- Use Necessary when you are sure it is needed and Possible when unsure but don't want to exclude the possibility

	Necessary for the product line	Possible for the product line
All products have P	All products in All product lines	All products in Some product line
Some products have P	Some product in All product lines	Some product in Some product line

Necessary-Some (*NS*)

All
product lines

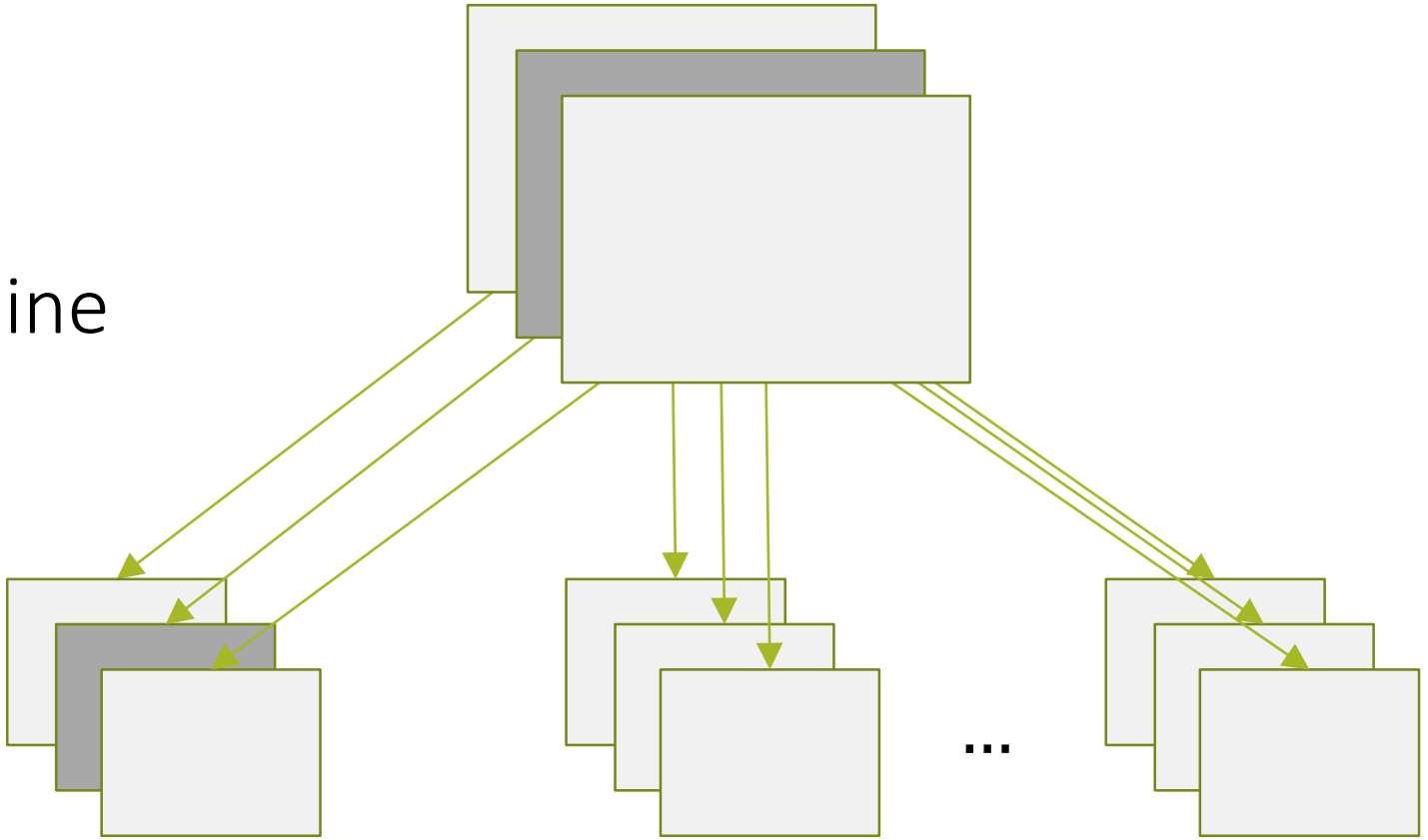
Some
product



Possible-Some (*PS*)

Some
product line

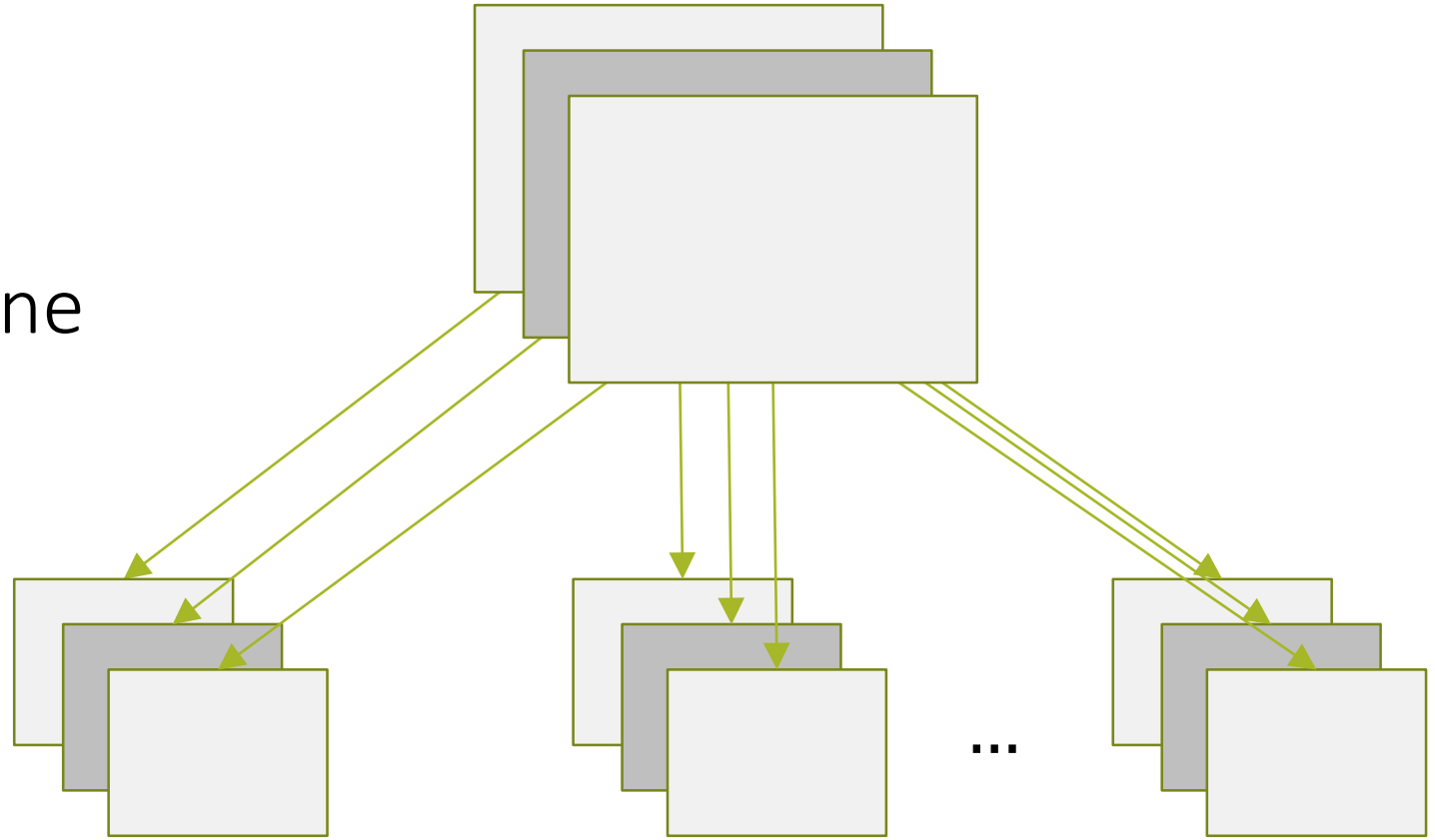
Some
product



Possible-All (*PA*)

Some
product line

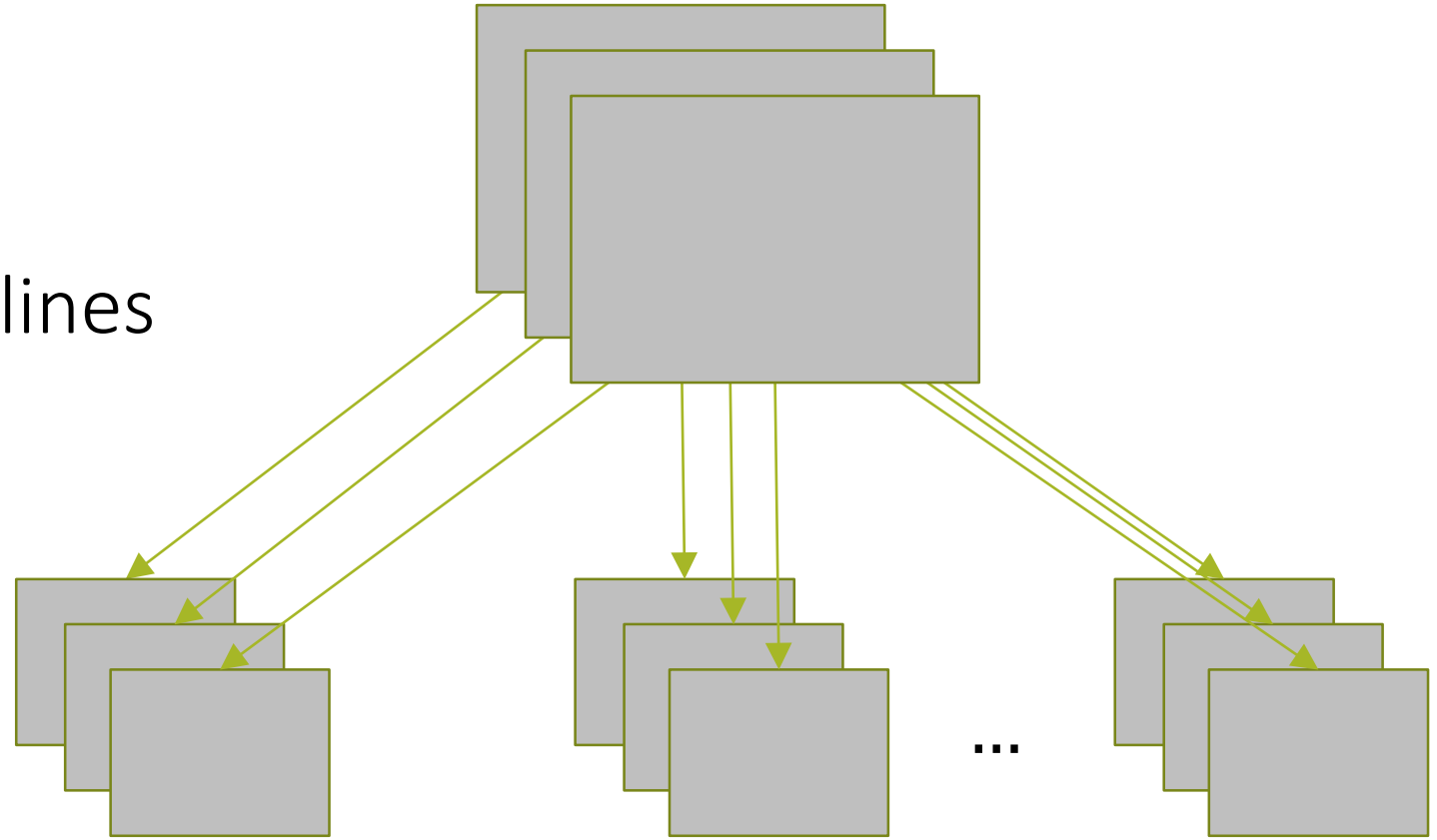
All
products



Necessary-All (*NA*)

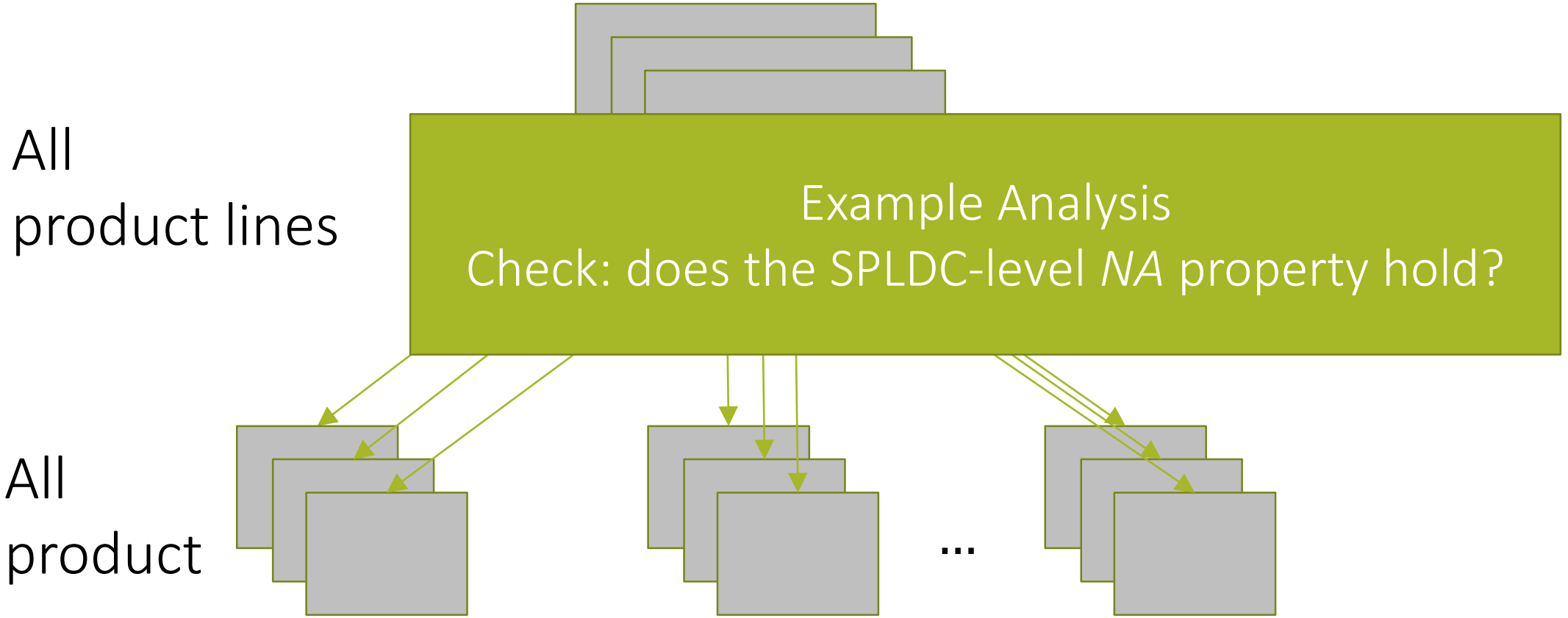
All
product lines

All
product

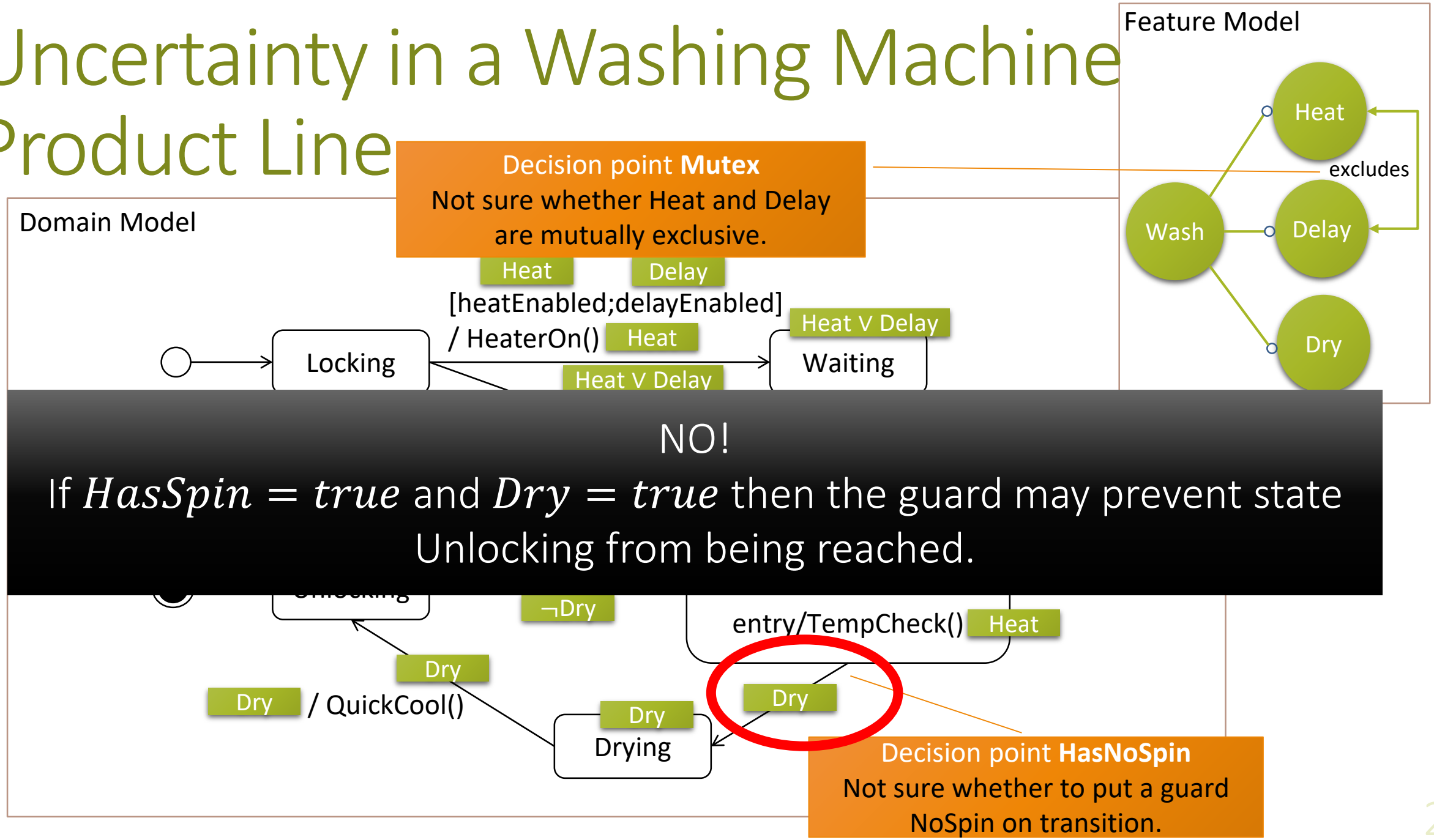


Necessary-All (*NA*)

Example product-level property: State **Unlocking** must always be reached
- a washing machine that violates this is unacceptable



Uncertainty in a Washing Machine Product Line



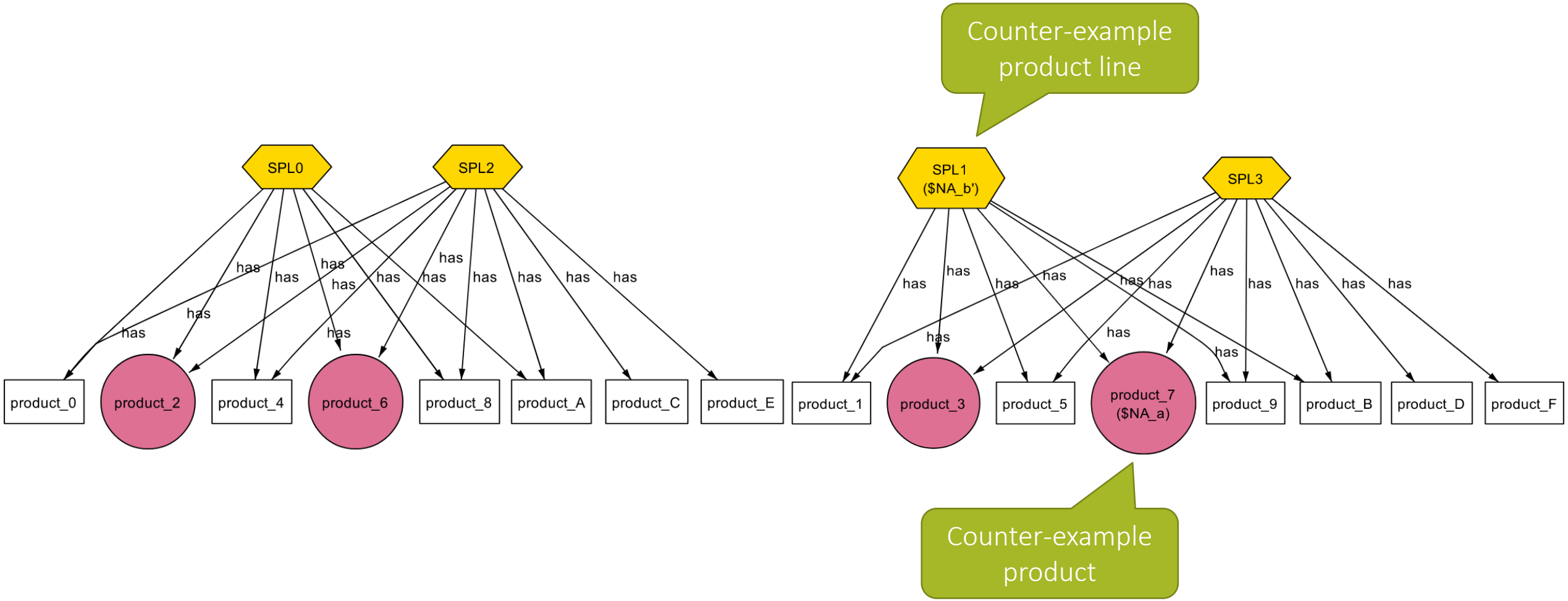
SPLDC-level Properties

SPLDC property modalities apply to any kind of product-level properties
- Behavioural (e.g., temporal logic)
- Structural (e.g., well-formedness constraints)
- Other (e.g., non-functional properties)

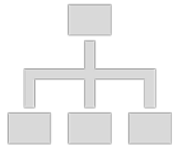
The main challenge is in implementing the lifted analysis
- such that it generates separate feedback for variability and design uncertainty

(Necessary-All)

Generating Feedback for *NA* Property



Software Product Lines with Design Choices



Motivation

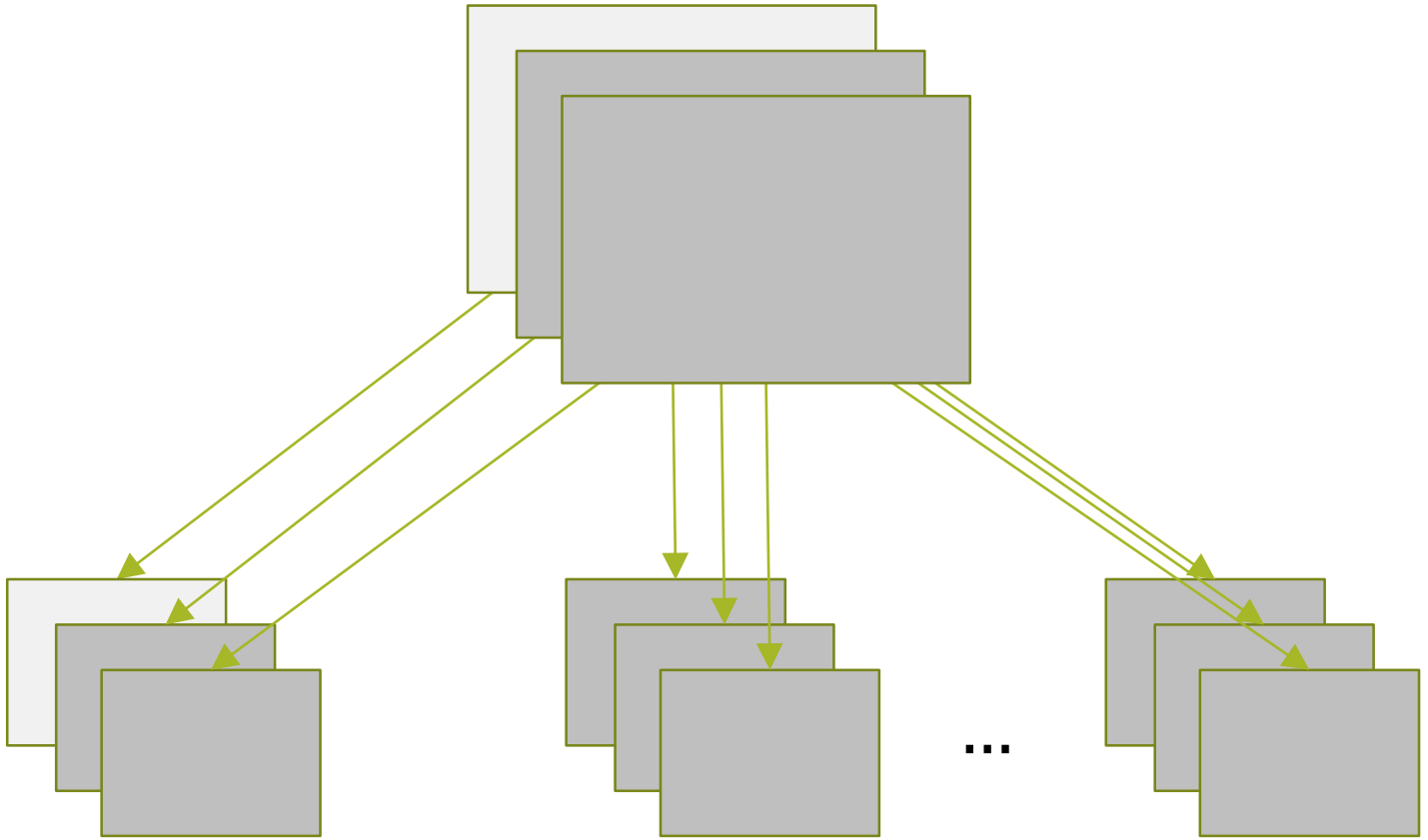
How to model this design space?

What are relevant properties for exploring it?
How to check them?

What to do when properties are violated?

Responses to Property Violation

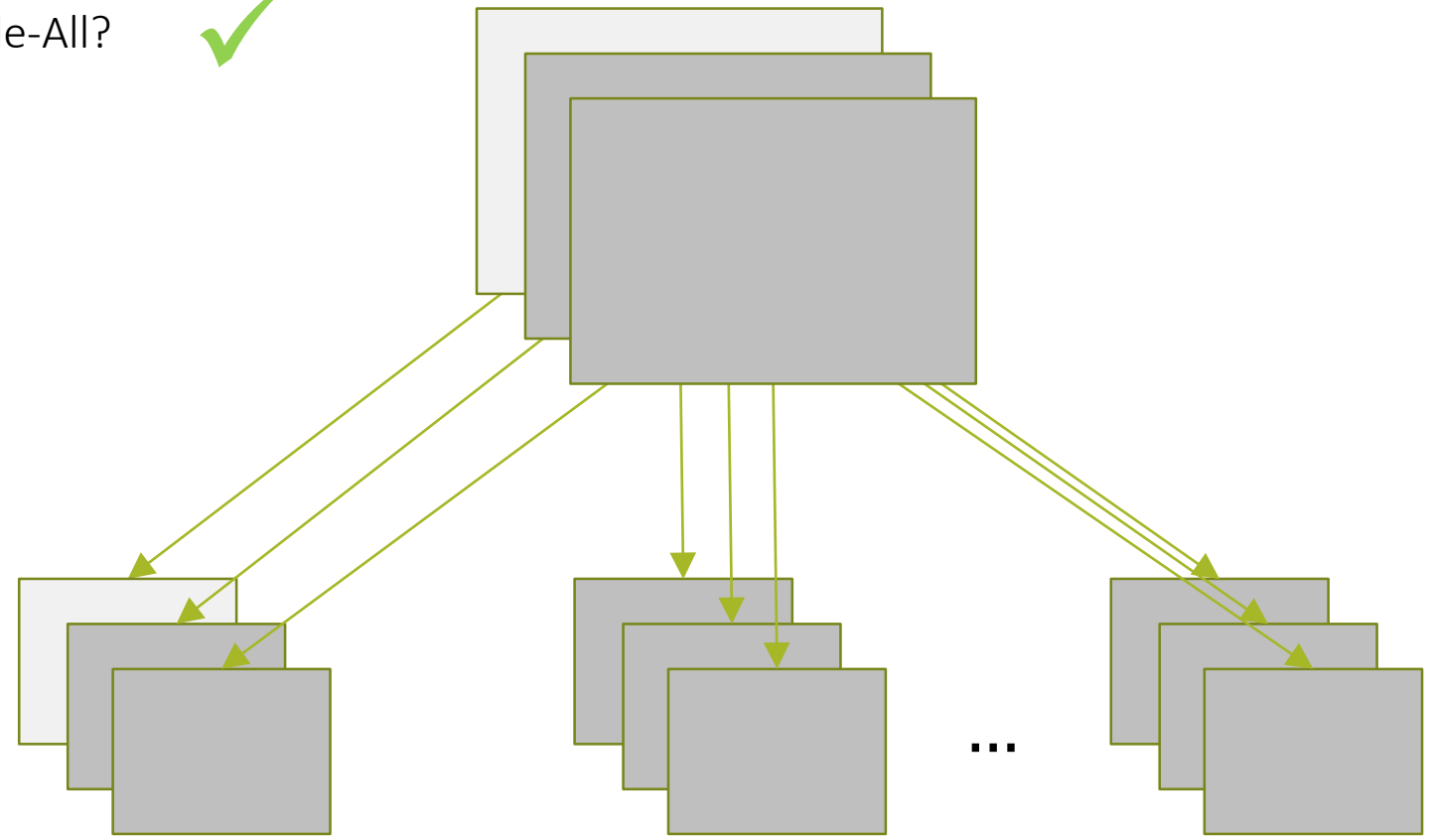
Necessary-All? ✗



Response 1: Relax Constraint

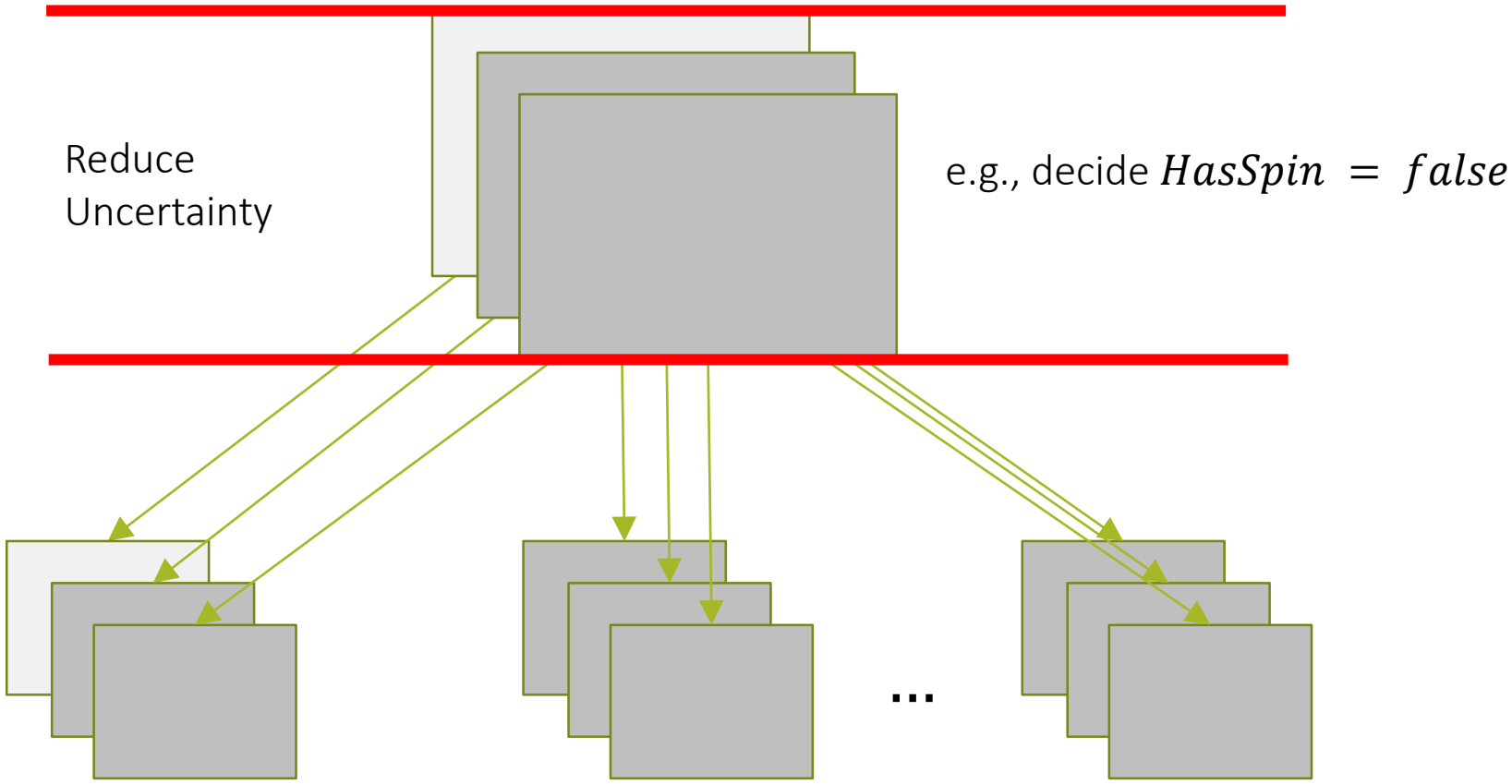
~~Necessary-All?~~ ✗

Possible-All? ✓



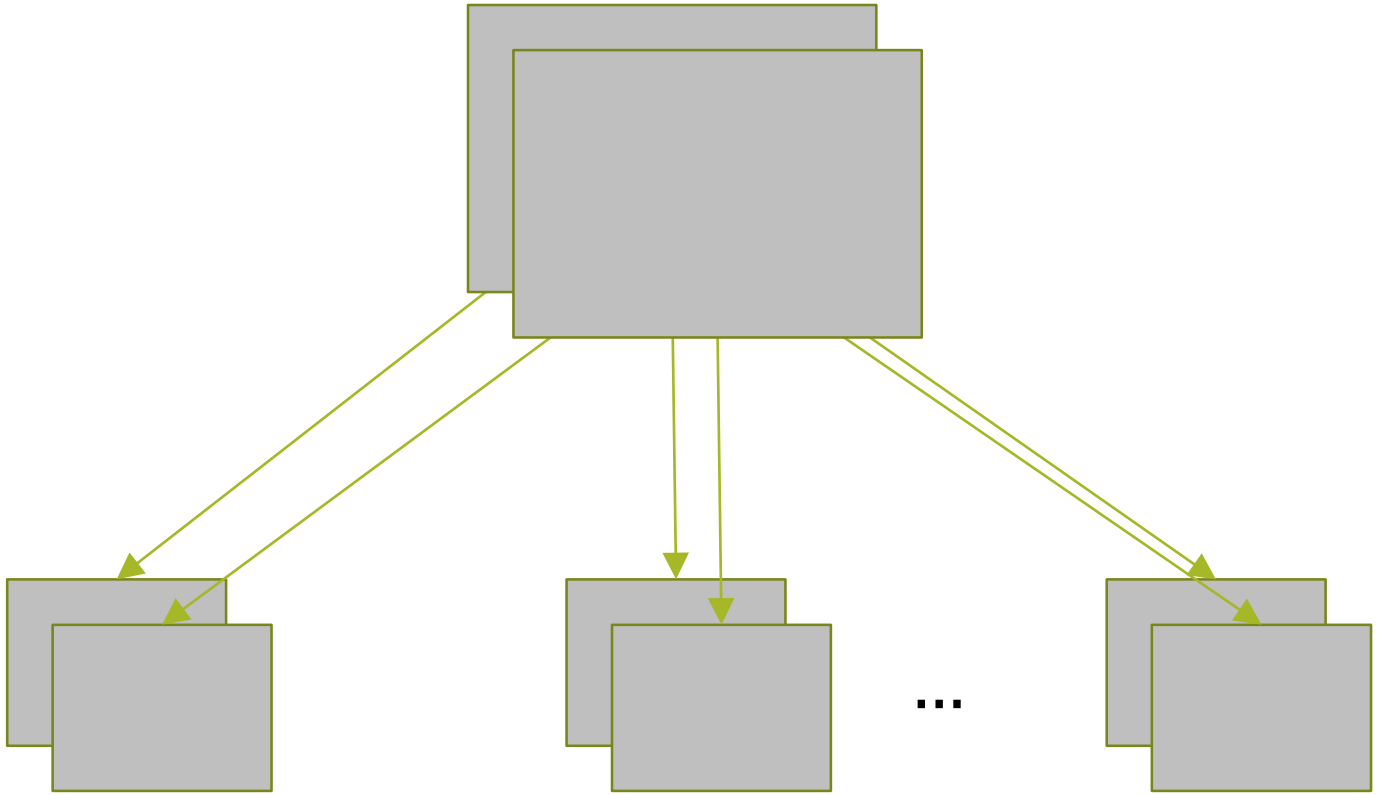
Response 2: Reduce Uncertainty

Necessary-All? ✗



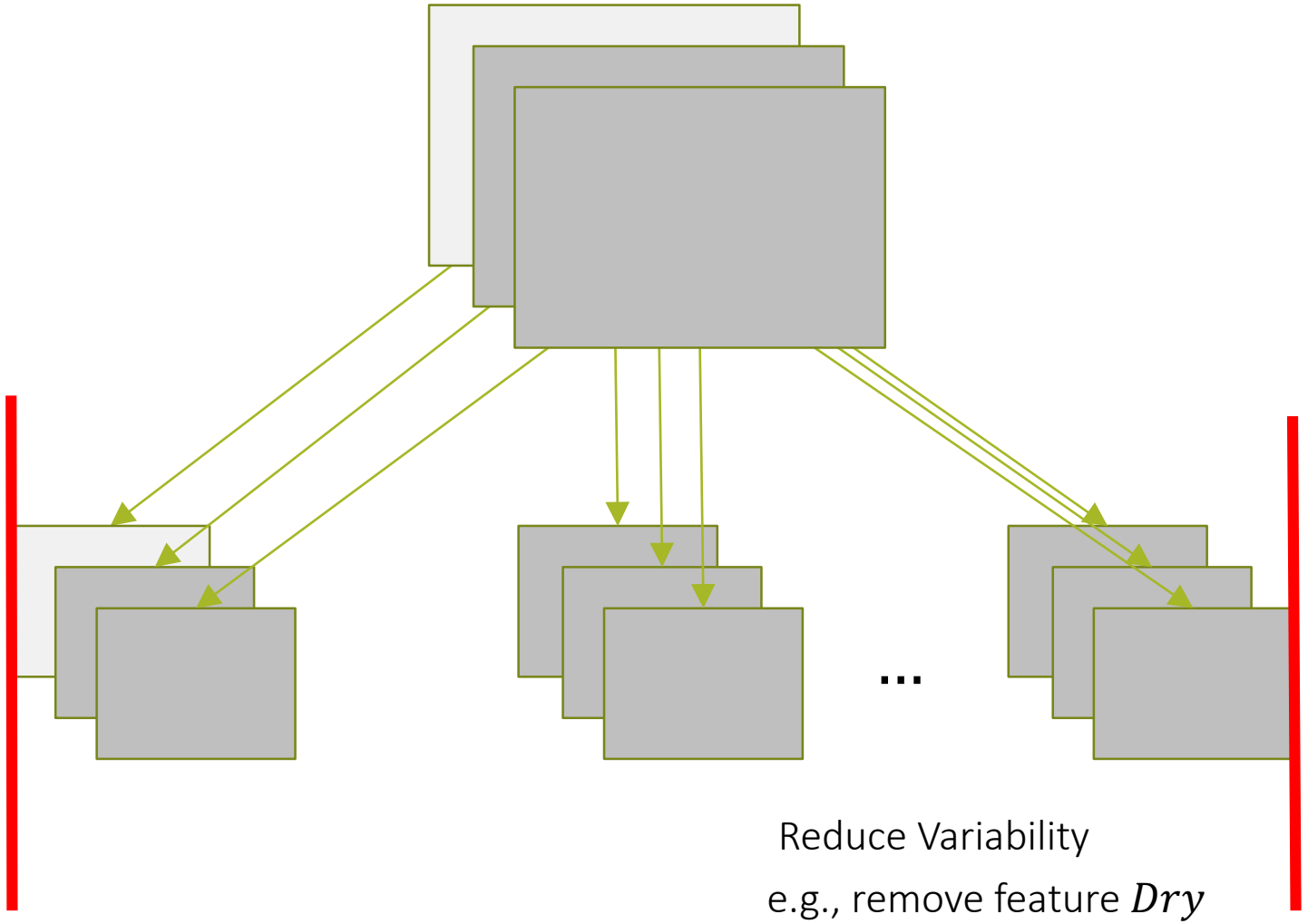
Response: Reduce Uncertainty

Necessary-All? ✓



Response: Reduce Variability

Necessary-All? ✗

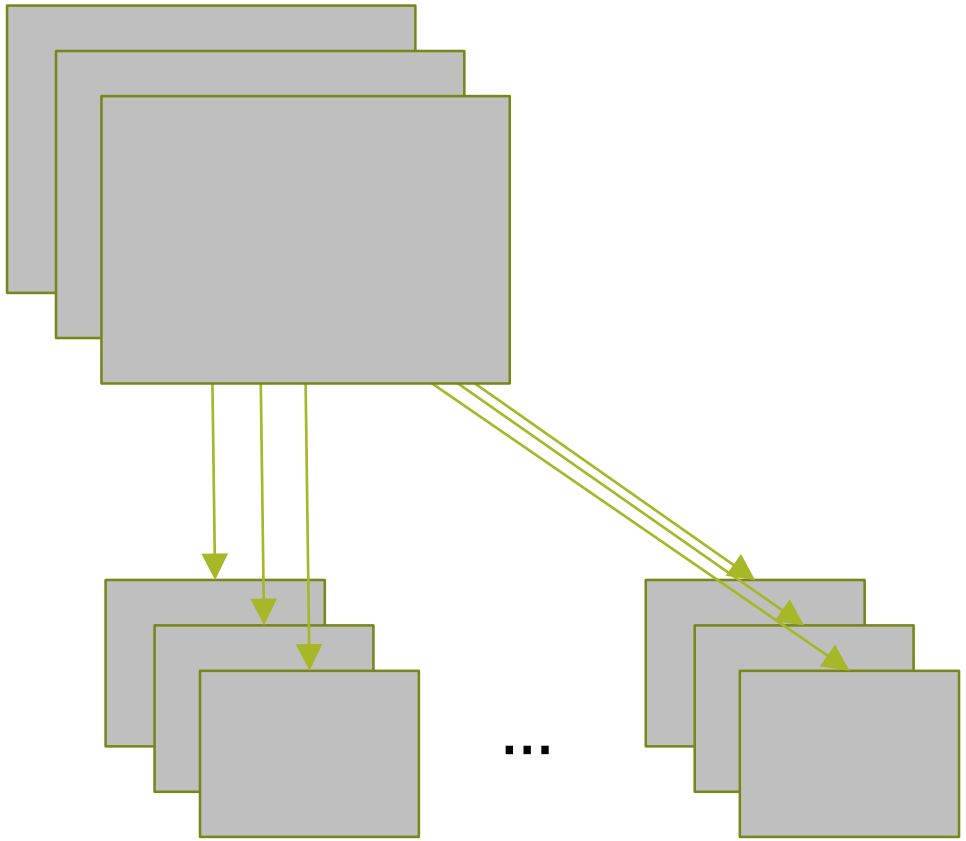


Reduce Variability

e.g., remove feature Dry

Response 3: Reduce Variability

Necessary-All? ✓



Responding to SPLDC Property Violations

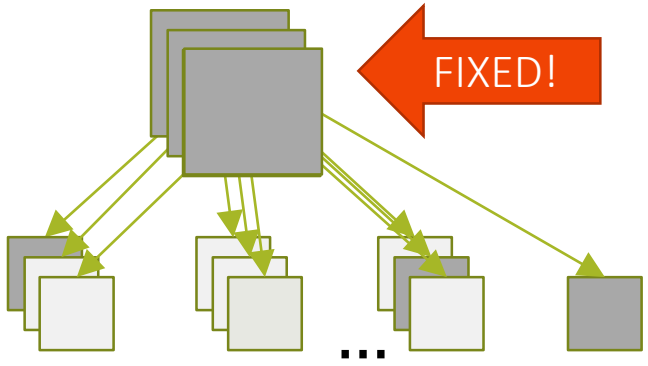
Guidance for leveraging generated feedback:

- Should the problem be addressed at the SPLDC level?
- If so, which strategy is most appropriate?
- If not, can we recommend repairs?

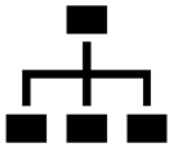
Sometimes we can address problems at the SPLDC level by **addition**

- E.g., adding previously forbidden configuration options
- Can we recommend such additions?

Necessary-Some (*NS*) violated:



Software Product Lines with Design Choices



Motivation

How to model this design space?

What are relevant properties for exploring it?
How to check them?

What to do when properties are violated?

Current Work

Focus on structural SPLDC properties
SPLDCs as first-order theories
Preliminary specification using Alloy
Investigating more sophisticated reasoning engines (e.g. QBF, Second Order Logic)

Next Steps

Evaluating scalability of the tools and effectiveness of the methodology
Better automation of response strategies
Case studies

Power Window, "Real" Washing Machine, GitHub clones of configurable projects

Design Uncertainty and Variability

Similar but different concerns

Can be used together to model and explore a space of SPL designs

We defined:

- Four natural classes of SPLDC-level properties that can be checked
- Strategies for responding to property violations