University of Namur

University of Duisburg-Essen

Cross-checking Disambiguated Product Line Variability Models

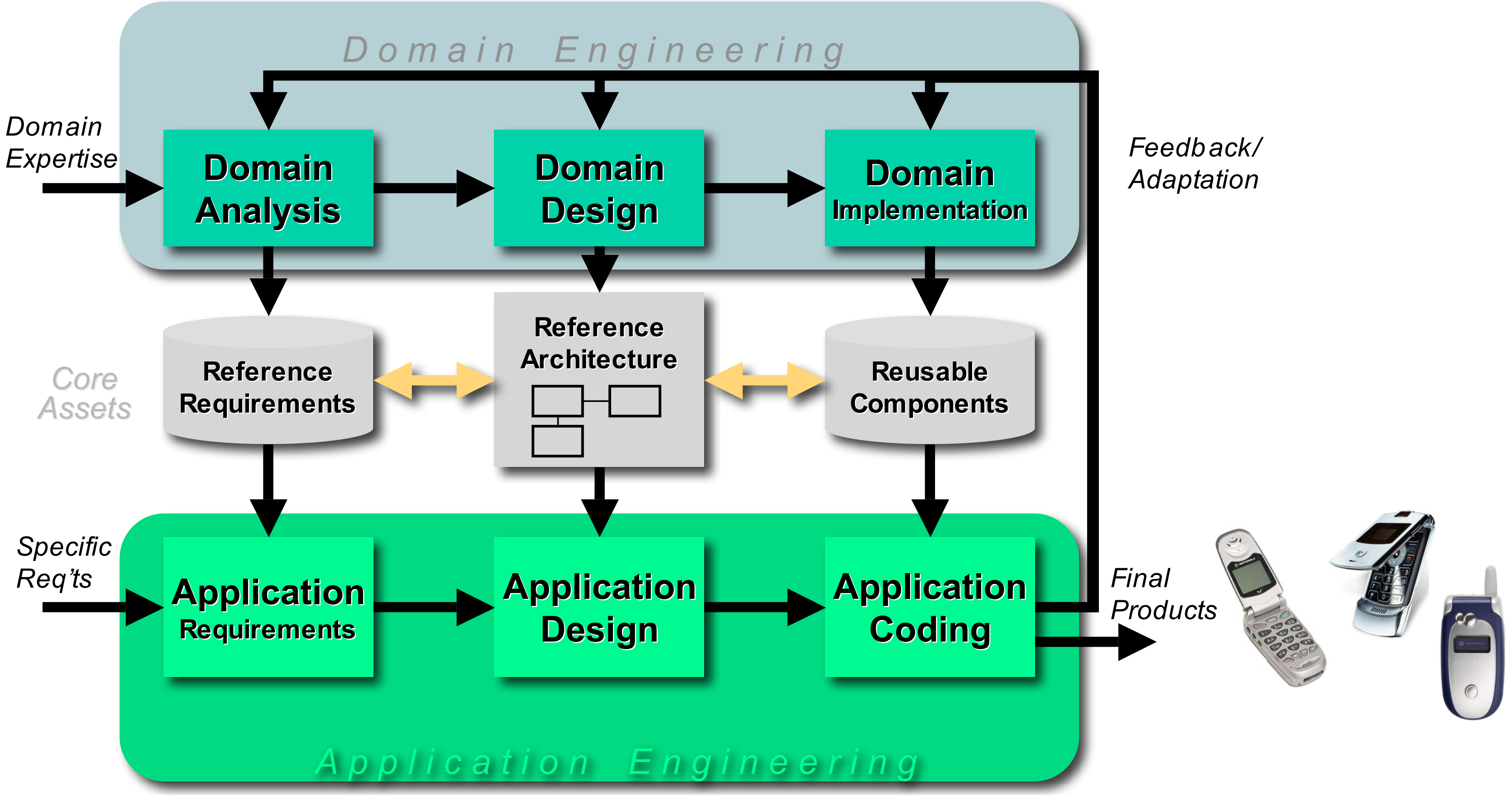
P. Heymans, A. Metzger, P-Y. Schobbens,
K. Pohl, G. Saval, A. Hubaux

svpp 08, VUB 8-9/8/2008

Presentation outline

1. **Software Product Lines Engineering**
2. Two kinds of variability
3. Objectives and approach overview
4. Internal model verification
5. Cross-model verification
6. Summary of contributions
7. Current & future work

Software Product Line Engineering (SPLE)



www.sei.cmu.edu/productlines

Key SPLE benefits & challenges

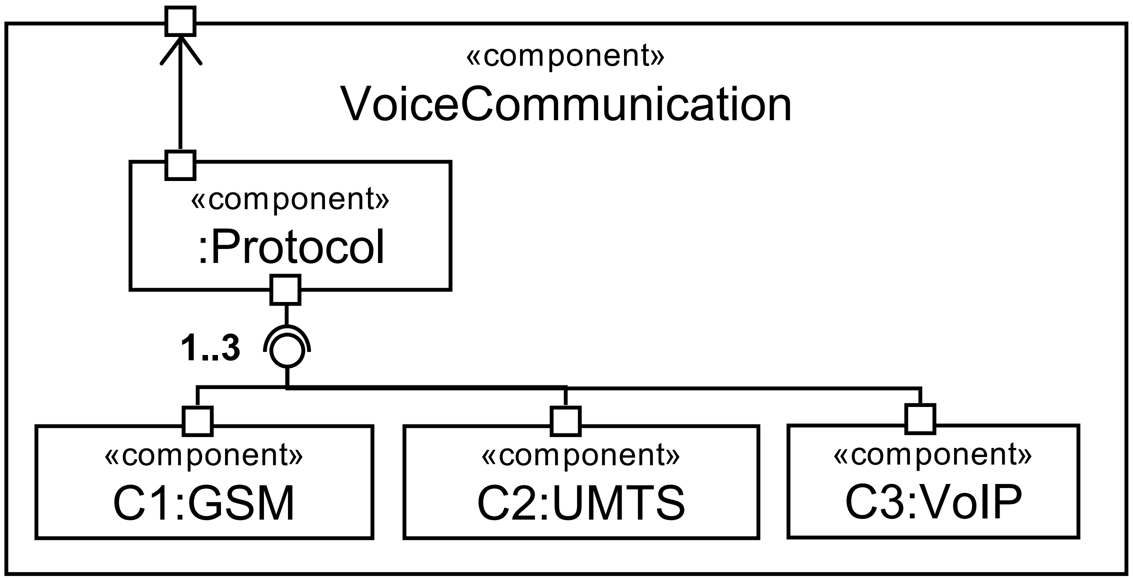
- Benefits
  - Scale economies
  - Shorter time to maket
  - Less risky development
- Challenges
  - High upfront adoption costs
  - Requirements are even more crucial
    - they determine the success of the whole family
  - Manage the variation between products

1. Software Product Lines Engineering
2. **Two kinds of variability**
3. Objectives and approach overview
4. Internal model verification
5. Cross-model verification
6. Summary of contributions
7. Current & future work

Two kinds of variability

> ☛ **Software Variability** refers to the **ability of a software system or artefact** to be efficiently extended, changed, customized or configured for use in a particular context. *[Svahnberg et al. 2005]*

- *descriptive* statements about the existing software assets
- relevant to both SPLE and single product development
- example:



Two kinds of variability

☛ **Product Line Variability** describes the **variation (differences) between the systems** that belong to a product line in terms of properties and qualities (like features that are provided or requirements that are fulfilled).

[Coplien et al., 1998] [Kang et al., 2002] [Pohl et al., 2005]

- *prescriptive* statements about the products to be built
- explicit **decisions** made by product management
- specific to SPLE
- example :
  - *“ Every mobile phone in the PL shall support the GSM protocol, the UMTS protocol, or both (but not VoIP or other protocols) ”*

Relationship between the two variabilities

☛ **Product Line Variability** describes the **variation (differences) between the systems** that belong to a product line in terms of properties and qualities (like features that are provided or requirements that are fulfilled).

[Coplien et al., 1998] [Kang et al., 2002] [Pohl et al., 2005]

is realized through

☛ **Software Variability** refers to the **ability of a software system or artefact** to be efficiently extended, changed, customized or configured for use in a particular context. *[Svahnberg et al. 2005]*

1. Software Product Lines Engineering
2. Two kinds of variability
3. **Objectives and approach overview**
4. Internal model verification
5. Cross-model verification
6. Summary of contributions
7. Current & future work

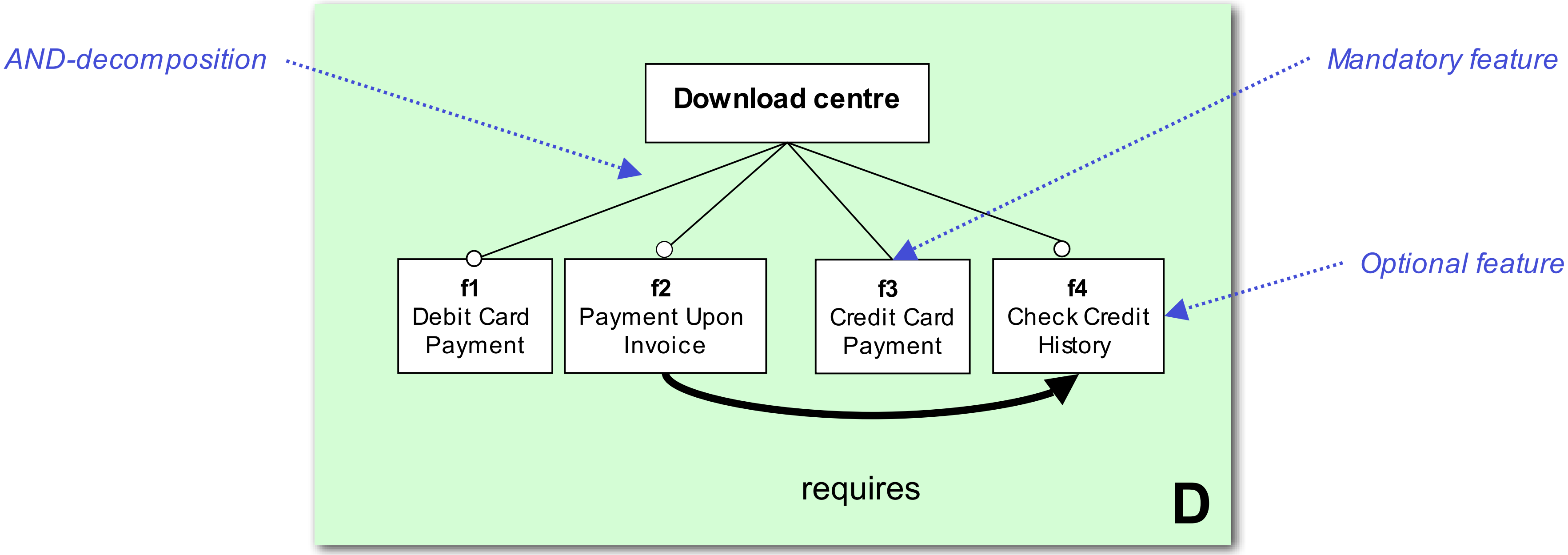
Our objective

- Support PL business and software engineers in
  - Making PL variability decisions that are aware of
    - the software asset's capabilities
    - the software adaptation costs
  - Developing software assets that allow
    - to realize all PL variability
    - but not *too much* more

- Current practice
  - information is not documented
  - or documented informally
  - or software & PL variability are not distinguished

[Metzger & Heymans, TR, 2006]

Use formal variability models *[RE'06]*



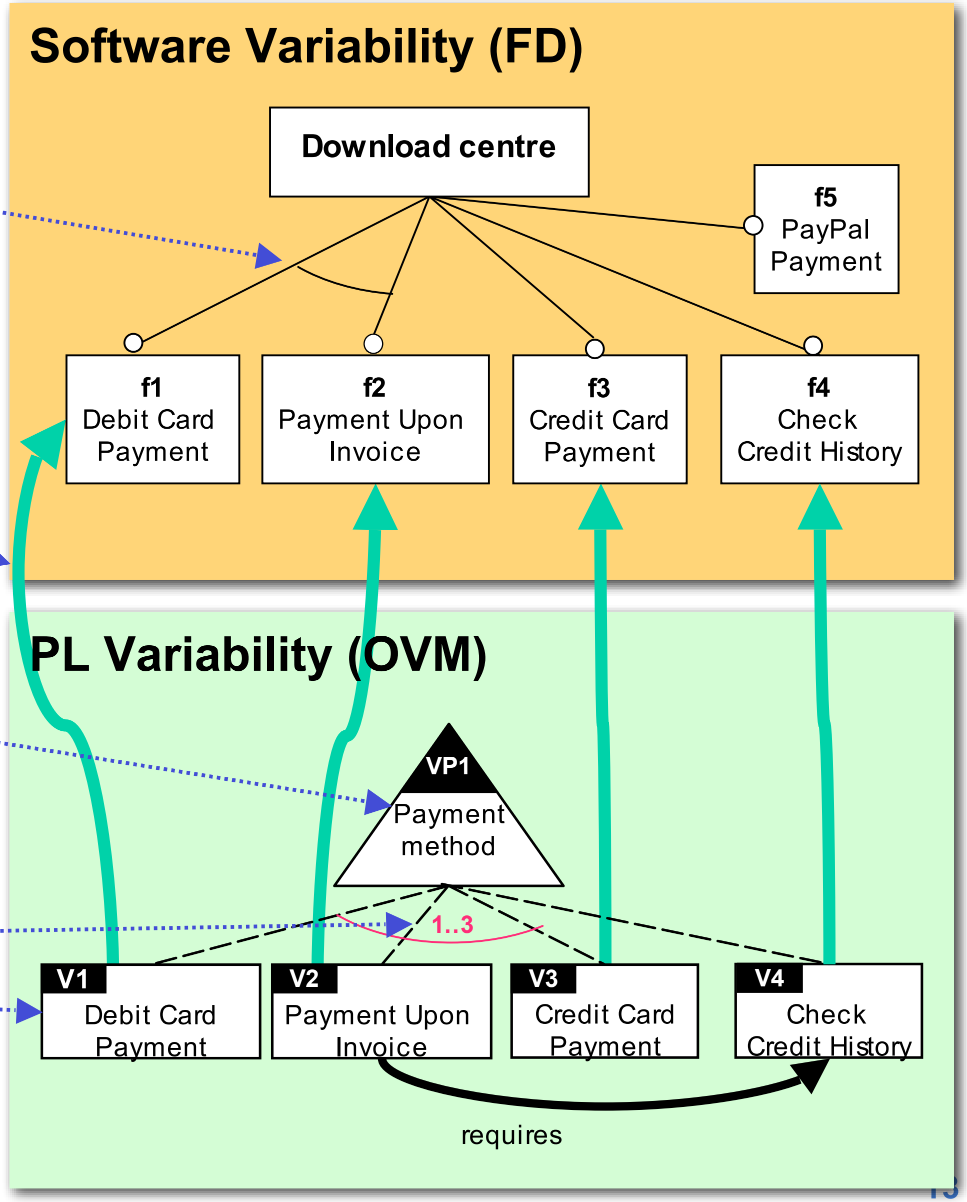
$$[\![D]\!] = \{ \ \{ f3 \}, \{ f1,f3 \}, \{ f3,f4 \}, \{ f2,f3,f4 \}, \{ f1,f3,f4 \}, \{ f1,f2,f3,f4 \} \ \}$$

Formal, but still **ambiguous**:

- are these the realizable software products?
- are these the PL members to be offered to customers?
- or an entangled mixture of each?

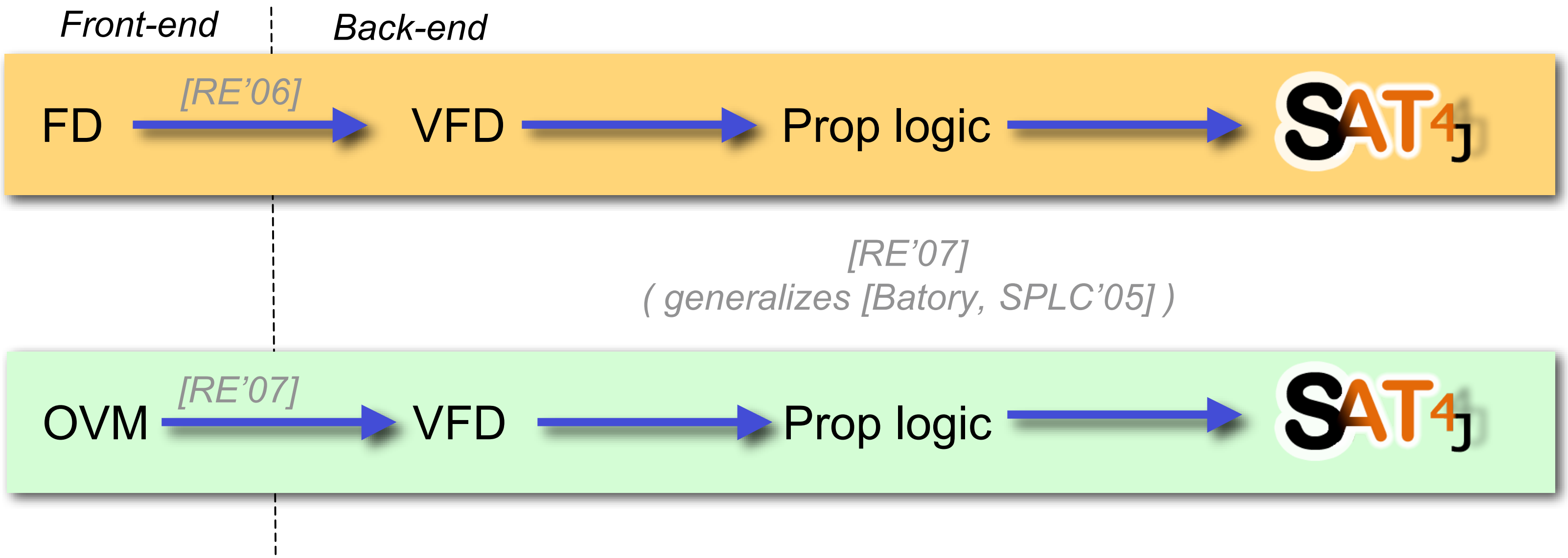
The proposed approach

- The 2 variabilities are
  - Documented **separately**
  - **Related** through X-links
- Using popular concrete syntaxes in the front-end
  - OVM *[Pohl et al., 2005]*
  - any FD dialect
- Using formal semantics in the back-end
  - less ambiguity
  - automated reasoning support



1. Software Product Lines Engineering
2. Two kinds of variability
3. Objectives and approach overview
4. **Internal model verification**
5. Cross-model verification
6. Summary of contributions
7. Current & future work

Internal model verification



Basic semantic checks *[Benavides et al. 2006] [RE'06]*, **e.g.**

- **Satisfiability**: $[\![D]\!]_{VFD} \neq? \emptyset$
- **Product (resp. PL member) enumeration**: list all p_i s.t. $p_i \in [\![D]\!]_{VFD}$
- **Product (resp. PL member) checking**: $p_i=\{f_{i,1},\ldots,f_{i,n}\} \in? [\![D]\!]_{VFD}$
- **Dead features (resp. variants)**: $\{f_1,\ldots,f_m\} \setminus \cap [\![D]\!]_{VFD}$
- **Commonality**: $\cup [\![D]\!]_{VFD}$

1. Software Product Lines Engineering
2. Two kinds of variability
3. Objectives and approach overview
4. Internal model verification
5. **Cross-model verification**
6. Summary of contributions
7. Current & future work

X-links and their semantics

- *"Whenever a variant is chosen,*
  all its X-linked features must be in"
  e.g.

  $\{\boldsymbol{V1}, V3, f3, V4, f4\} \notin [\![G]\!]$

  $\{V1, \boldsymbol{f1}, V3, f3, V4, f4\} \in [\![G]\!]$

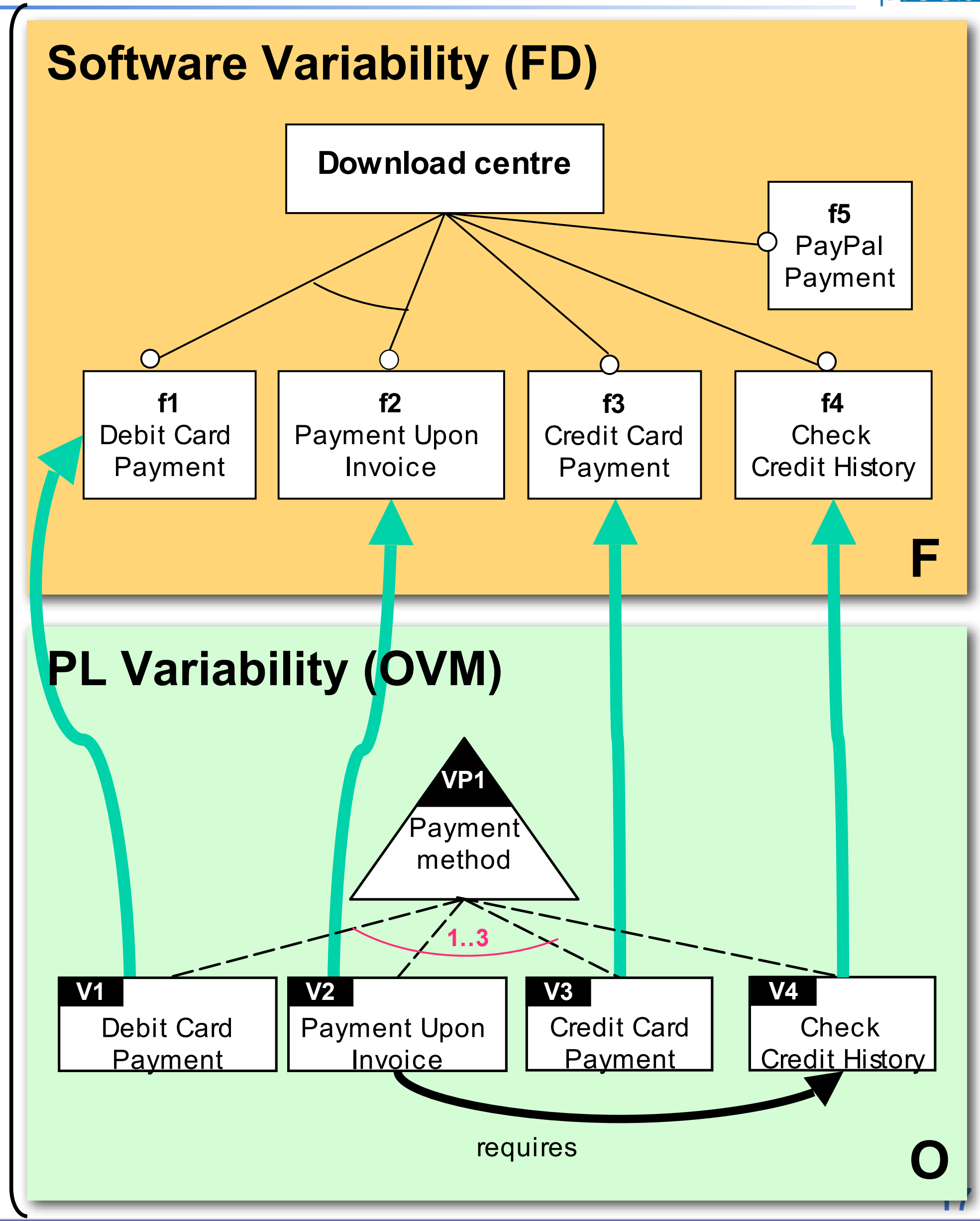
- *"An X-linked feature requires at least one*
  X-linked variant (justification) to be chosen"
  e.g.

  $\{\boldsymbol{f1}, V3, f3, V4, f4\} \notin [\![G]\!]$

  $\{\boldsymbol{V1}, f1, V3, f3, V4, f4\} \in [\![G]\!]$

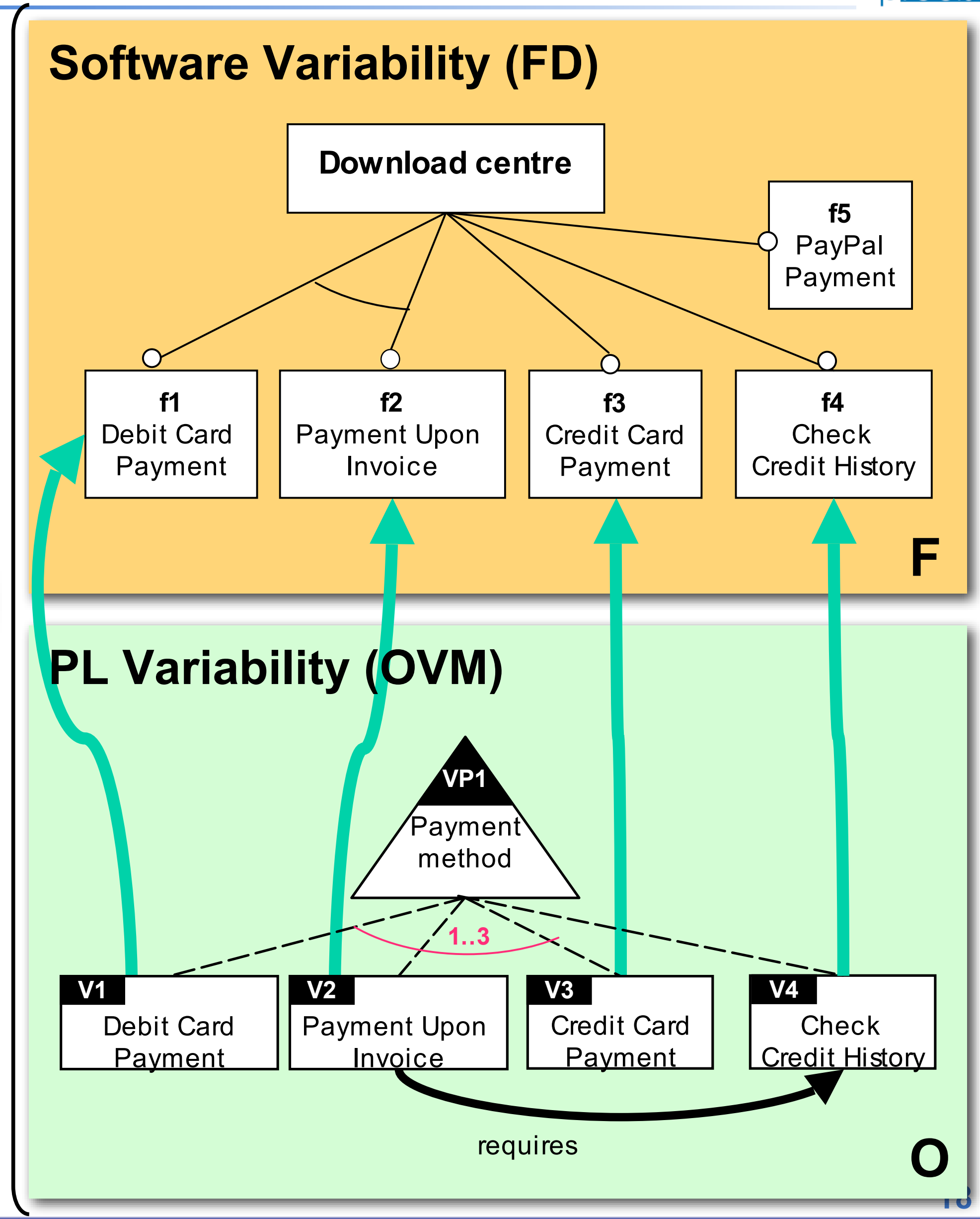
  $\{V1, f1, V3, f3, V4, f4, \boldsymbol{f5}\} \in [\![G]\!]$

G =



Global model's semantics

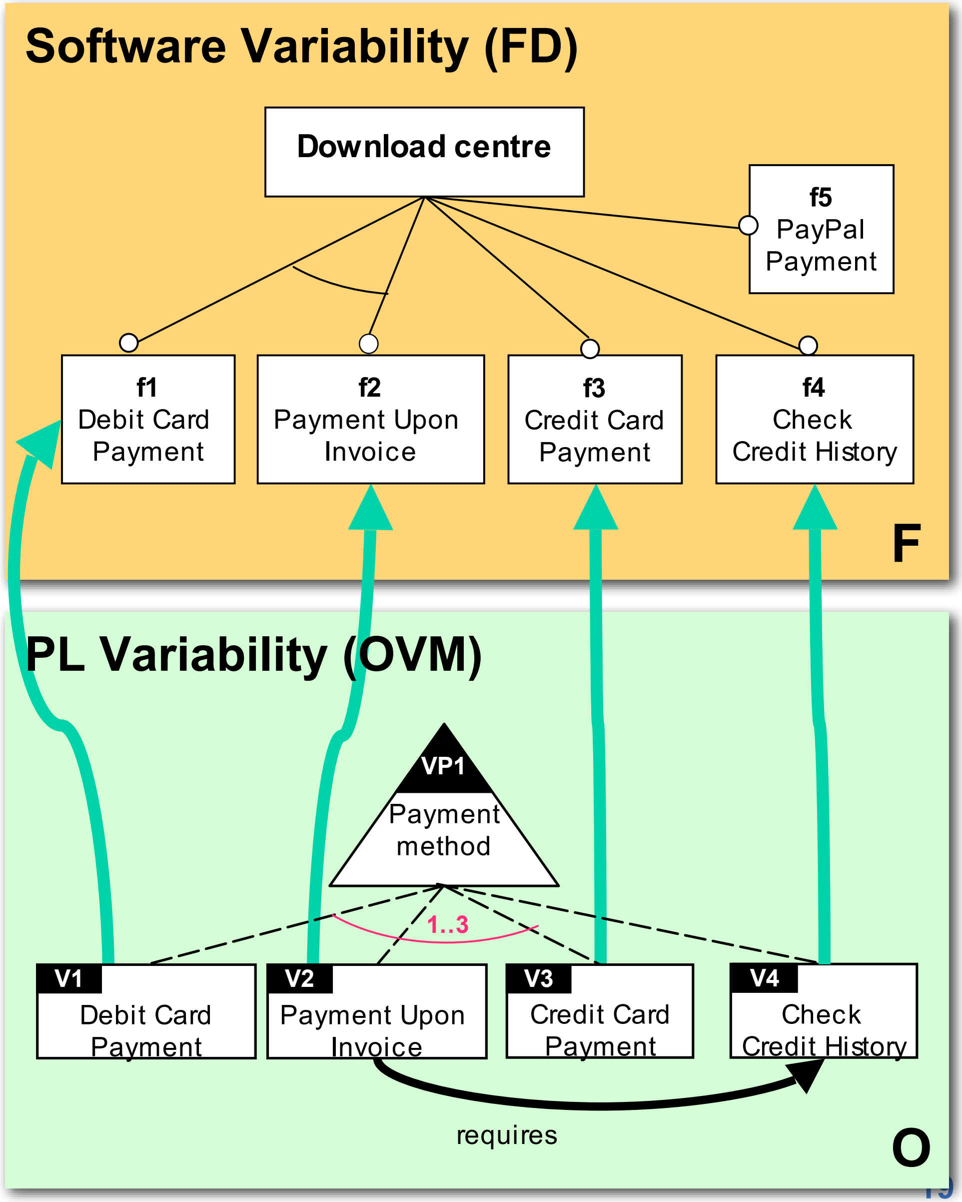
G =



⟦G⟧ is the set of realizable PL members (incl. their features)

Simple syntactic X-checks (warnings)

- Suspect cases
  - *Features not hit by an X-link*
  - *Variants with no departing X-link*

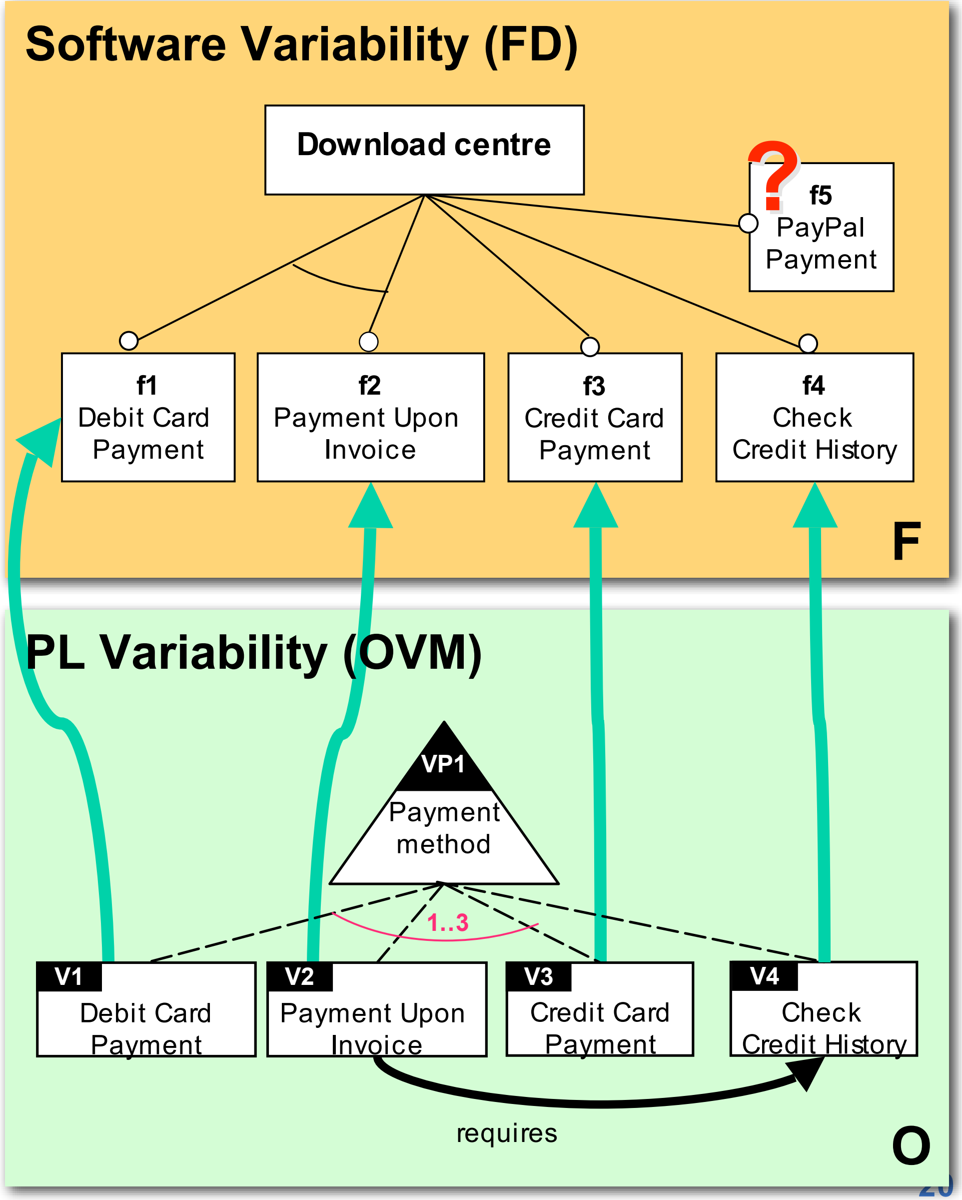


Simple syntactic X-checks (warnings)

- Suspect cases

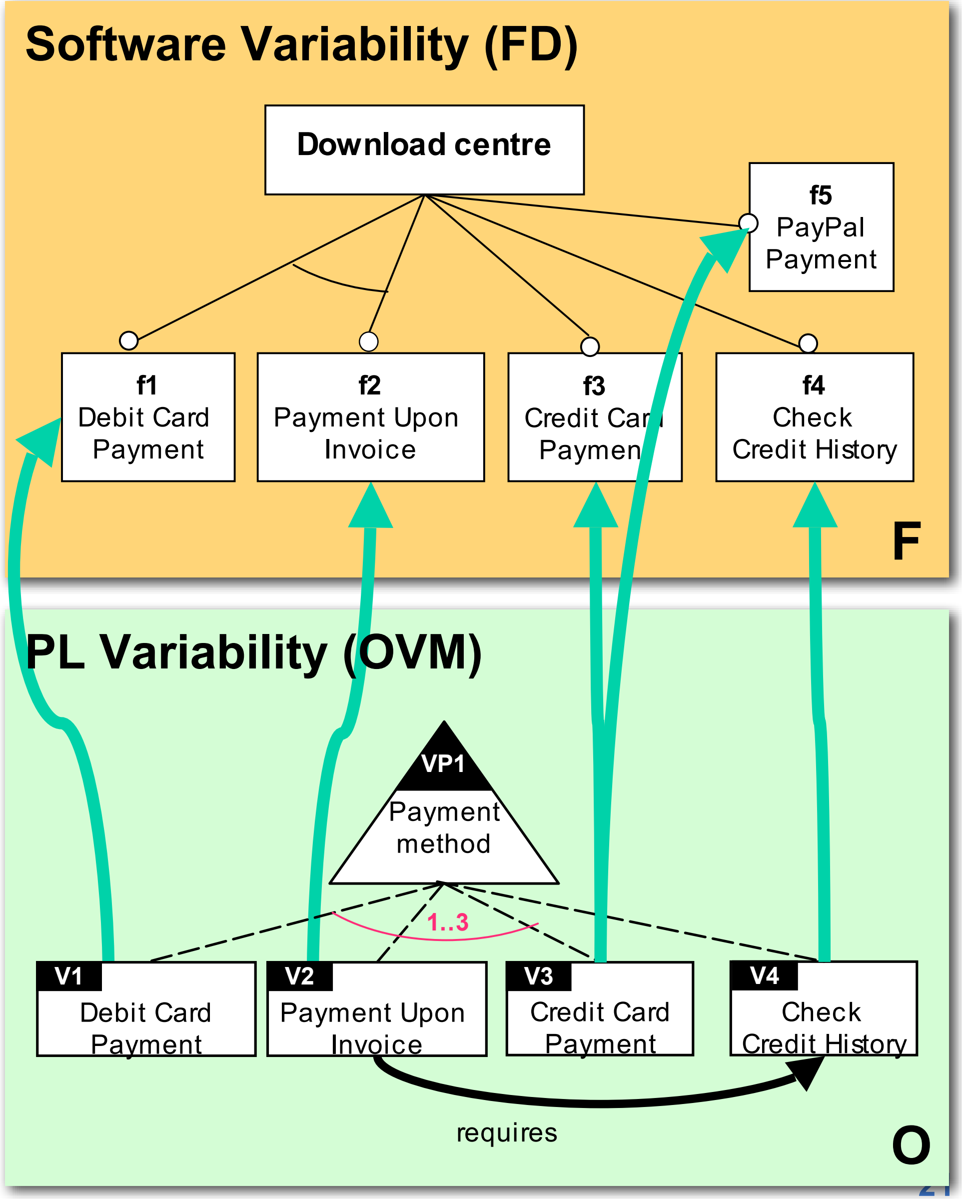
  ☒ *Features not hit by an X-link*

  ☑ *Variants with no departing X-link*



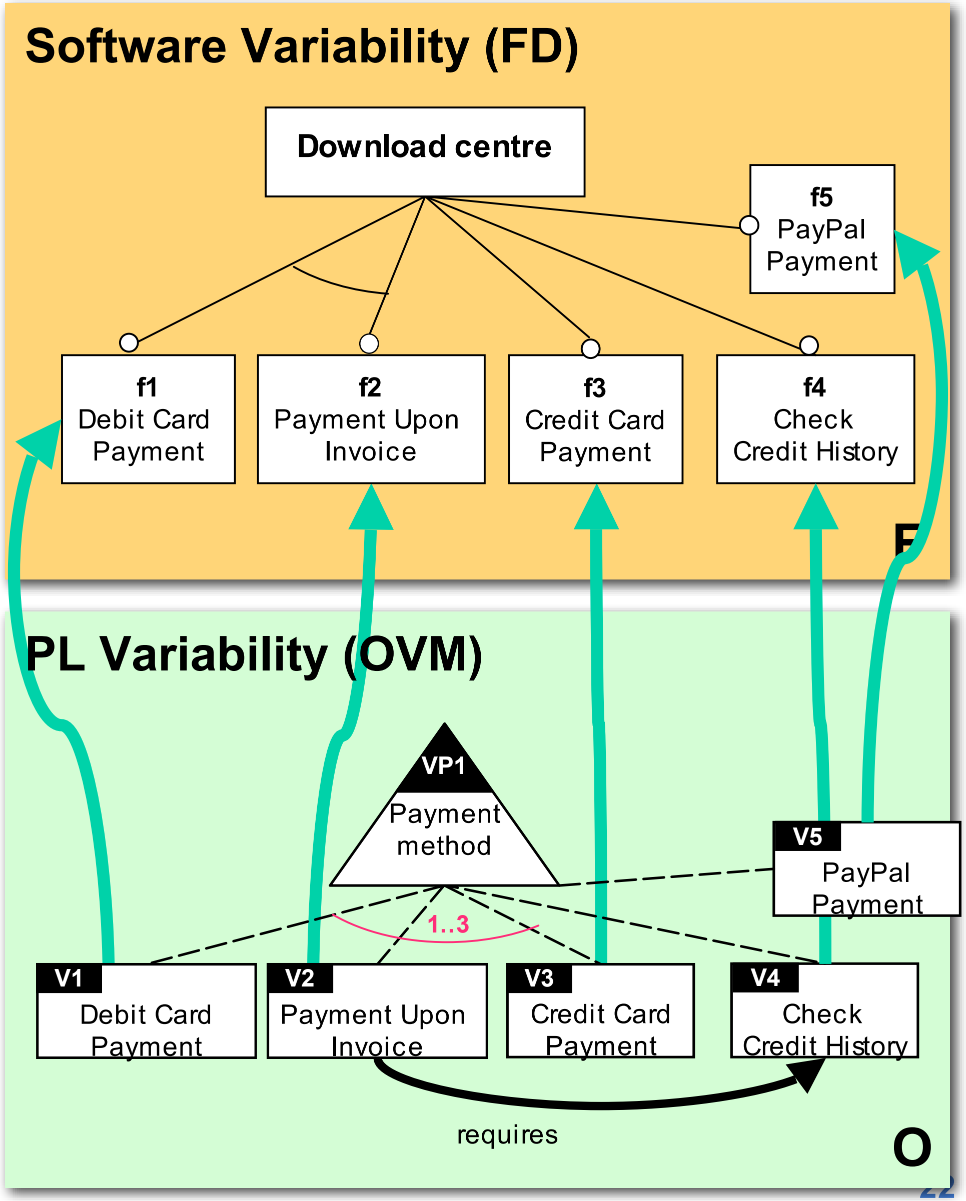
Simple syntactic X-checks (warnings)

- Suspect cases
  - ✓ *Features not hit by an X-link*
  - ✓ *Variants with no departing X-link*



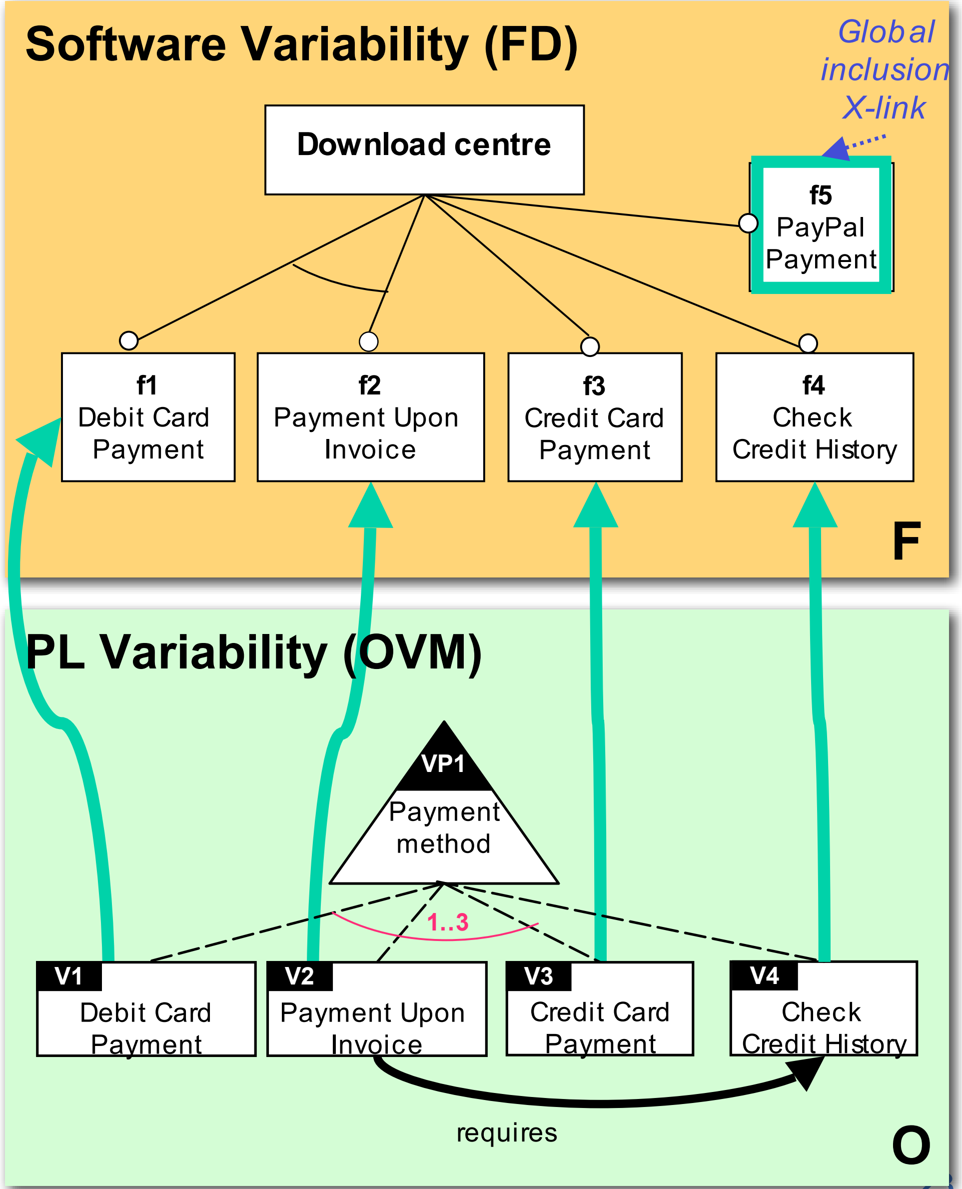
Simple syntactic X-checks (warnings)

- Suspect cases
  - *Features not hit by an X-link*
  - *Variants with no departing X-link*



Simple syntactic X-checks (warnings)

- Suspect cases
  - ☑ *Features not hit by an X-link*
  - ☑ *Variants with no departing X-link*

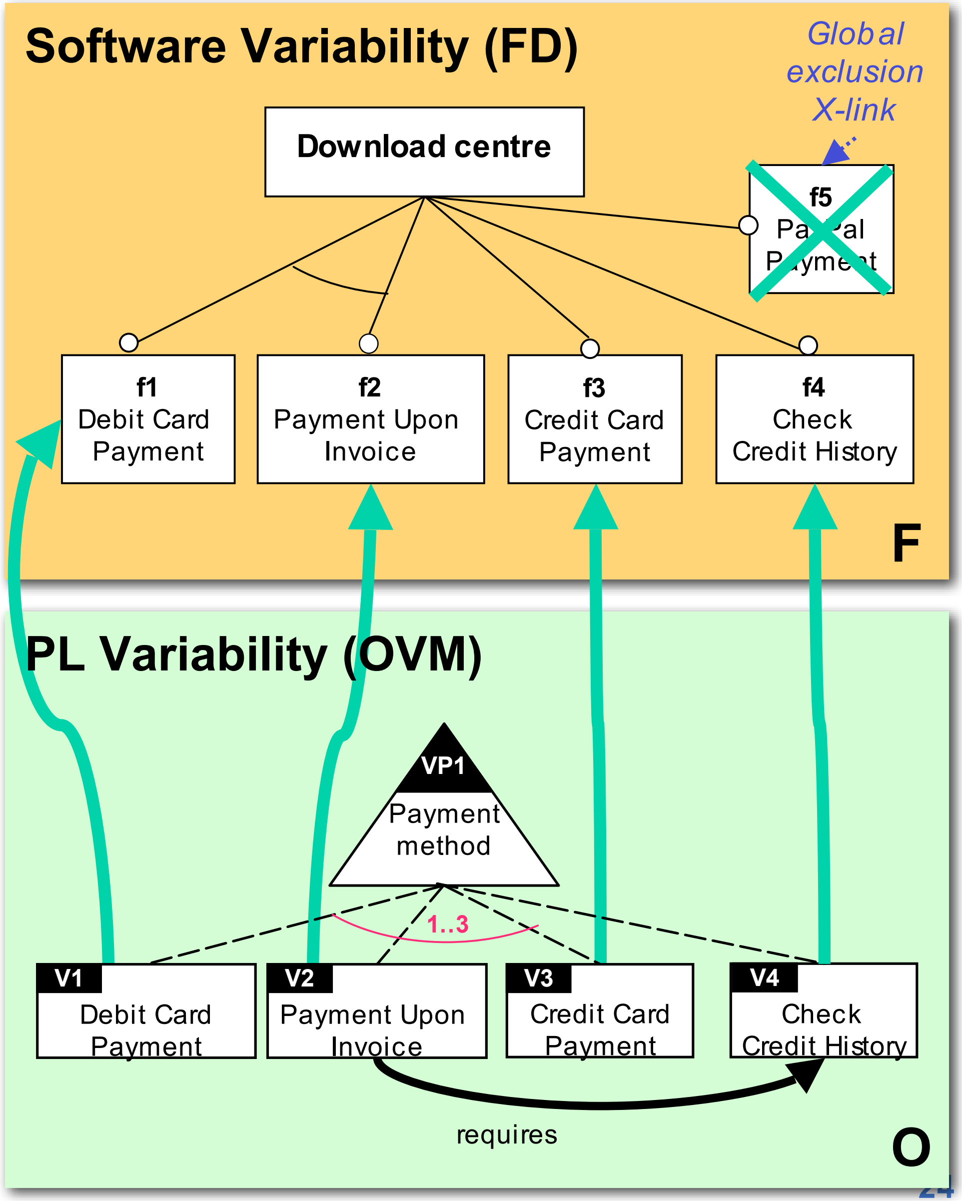


Simple syntactic X-checks (warnings)

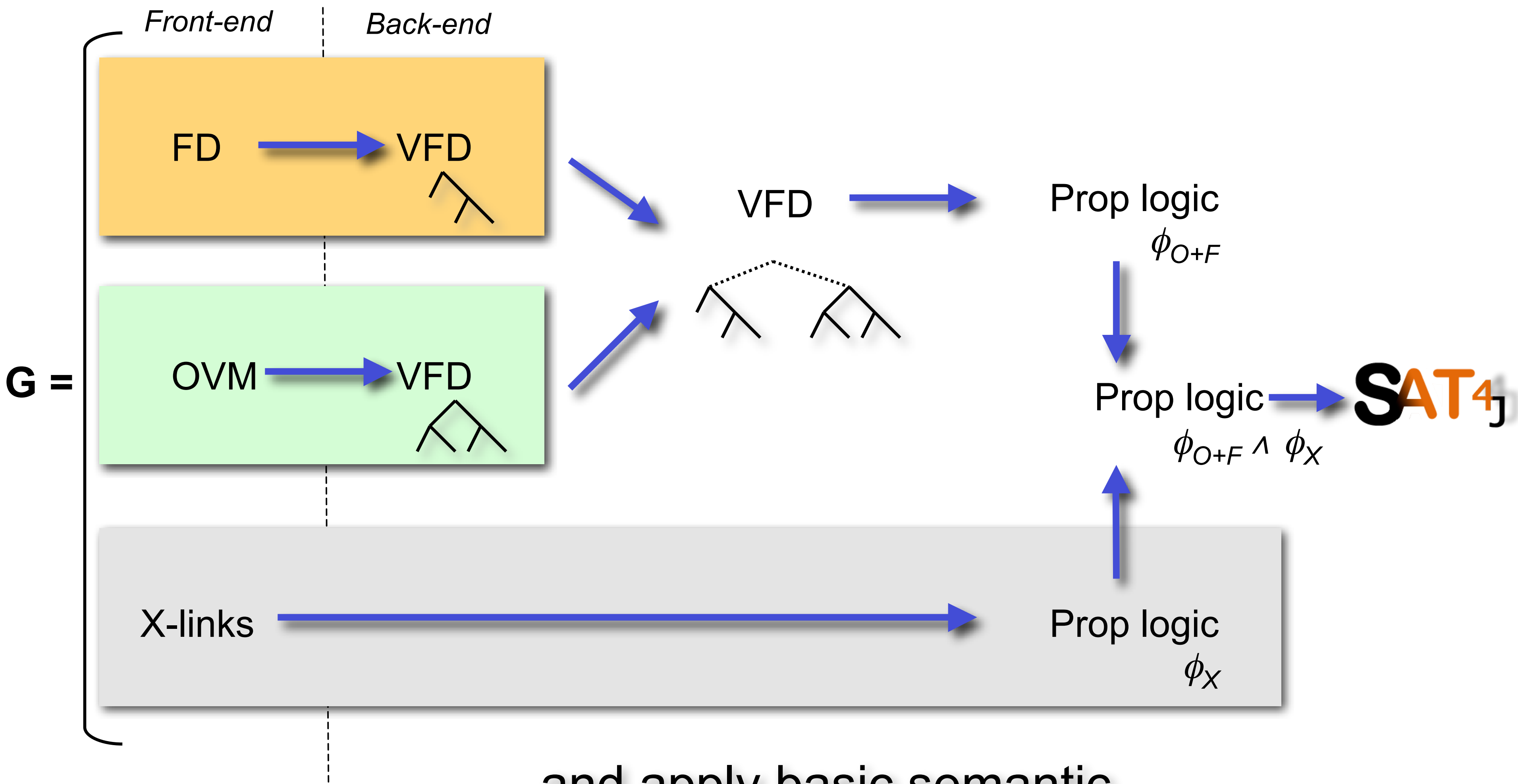
- Suspect cases

  ✓ *Features not hit by an X-link*

  ✓ *Variants with no departing X-link*



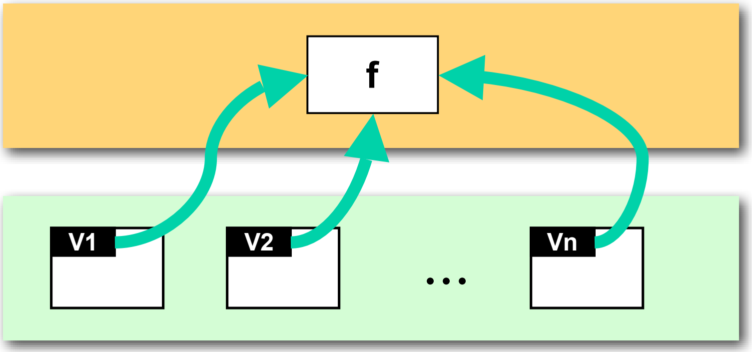
Towards semantic X-checks



...and apply basic semantic checks (satisfiability, product checking...) for a start

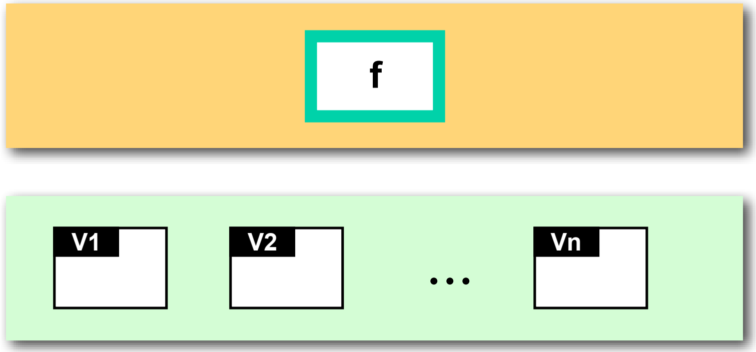
X-links → prop logic

***Pattern 1** — one-to-one inclusion X-link*



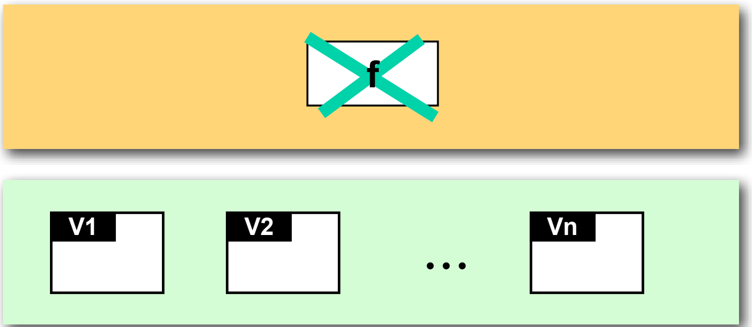
$\rightarrow f \Leftrightarrow v_1 \vee v_2 \vee \ldots \vee v_n$

***Pattern 2** — global **in**clusion X-link*



$\rightarrow f$

***Pattern 3** — global **ex**clusion X-link*



$\rightarrow \neg f$

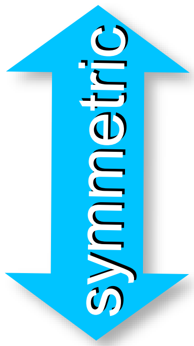
No specific pattern

Use any formula in

$\mathbb{B}$ ($\{f1,\ldots,fm\} \cup \{V1,\ldots,Vn\}$)

Advanced semantic checks

- **Check1. Realizability** — are there non-realizable PL members?
  - PL member $po \in [\![O]\!]$ is **realizable** if $po \in [\![G]\!]|_O$
  - non-realizable PL members are given by $[\![O]\!] \setminus [\![G]\!]|_O$

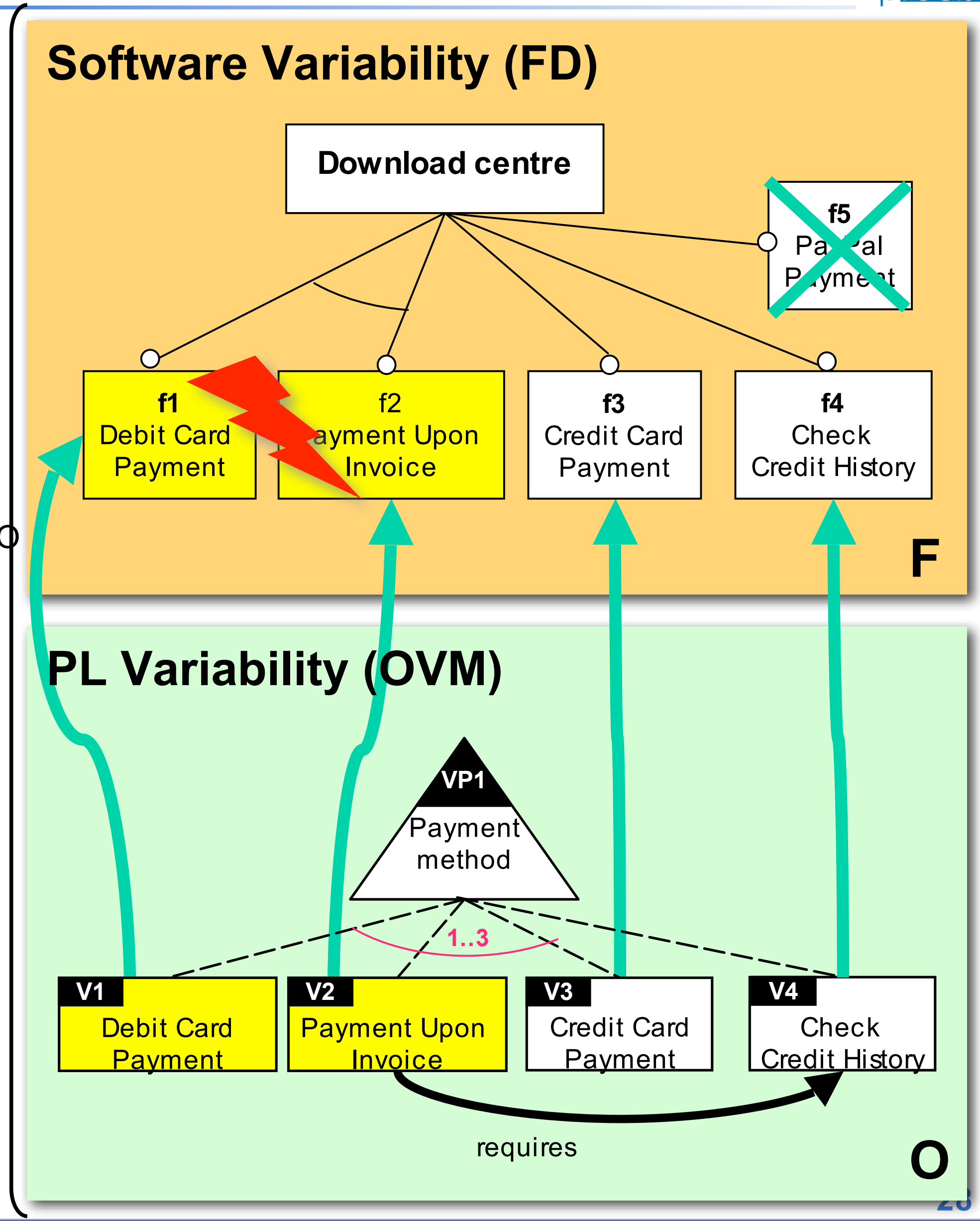


- **Check1'. Usefulness** — are there useless products?
  - product $pf \in [\![F]\!]$ **useful** if $pf \in [\![G]\!]|_F$
  - useless products are given by $[\![F]\!] \setminus [\![G]\!]|_F$

Realizability — example

- Detected issue
  - $\{V1, V2, V3, V4\} \in [\![O]\!]$
  - $\{V1, V2, V3, V4\} \notin [\![G]\!]|_O$
  - Hence, $\{V1, V2, V3, V4\} \in [\![O]\!] \backslash [\![G]\!]|_O$
- Solution
  - *either* restrict the PL scope
  - *or* increase the flexibility of the software platform

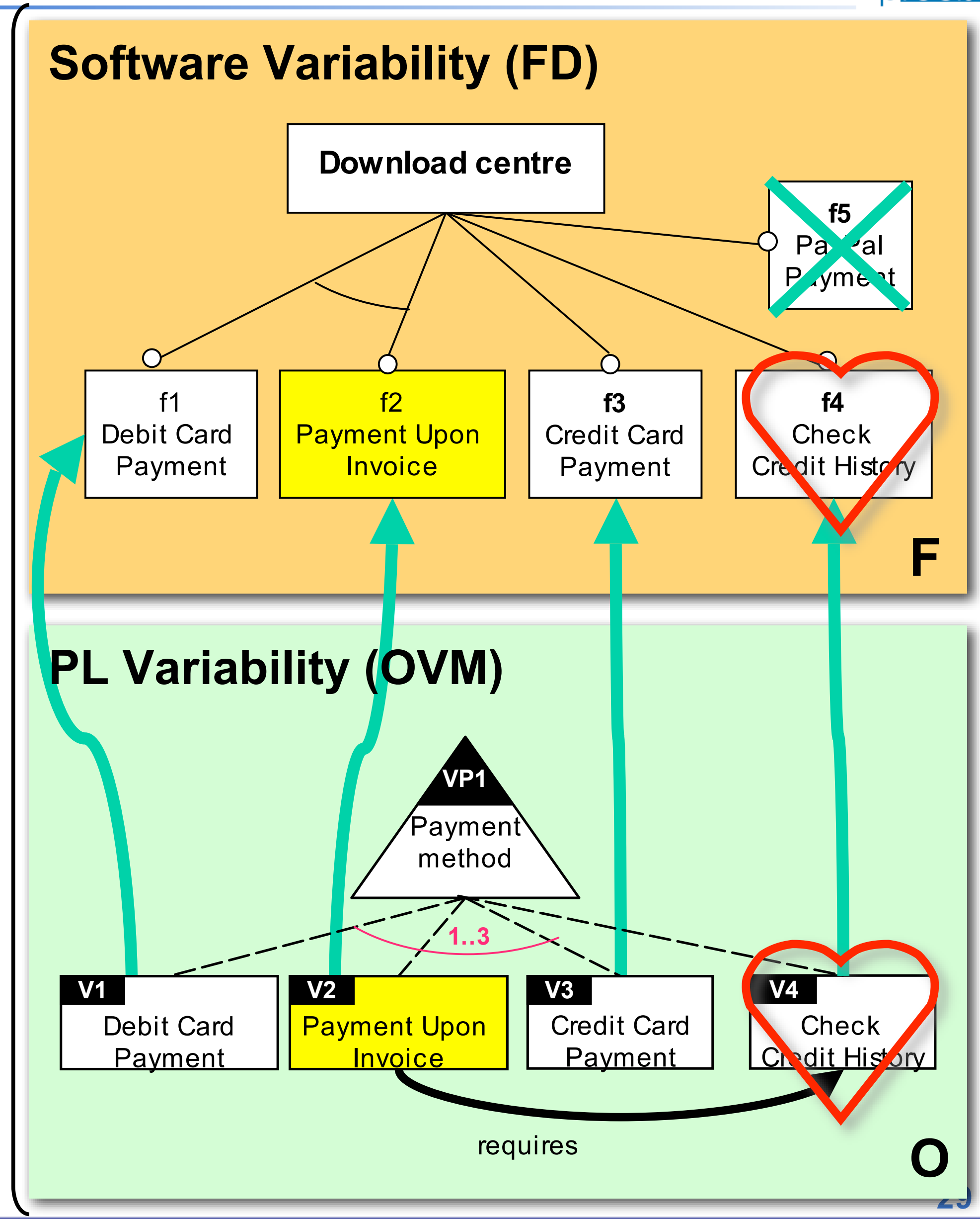
G =



Usefulness — example

- Detected issue
  - $\{f2\} \in [\![F]\!]$
  - $\{f2\} \notin [\![G]\!]|_F$
  - Hence, $\{f2\} \in [\![F]\!] \setminus [\![G]\!]|_F$
- Possible optimization
  - *either* expand the PL scope (for free)
  - *or* remove the flexibility of the software platform

G =



More advanced semantic checks

- **Check2. Internal competition** — 2 distinct PL members realized by 1 product?
  - $(po_1 \cup pf) \in [\![G]\!] \wedge (po_2 \cup pf) \in [\![G]\!] \wedge (po_1 \neq po_2)$
  - i.e. several ways for the customer to get the same features, maybe at different prices...

symmetric

- **Check2’. Unloyalty to customer** — 2 distinct products realizing the same PL member?
  - $(po \cup pf_1) \in [\![G]\!] \wedge (po \cup pf_2) \in [\![G]\!] \wedge (pf_1 \neq pf_2)$
  - i.e. two customers could choose the same PL member, and get different features

1. Software Product Lines Engineering
2. Two kinds of variability
3. Objectives and approach overview
4. Internal model verification
5. Cross-model verification
6. **Summary of contributions**
7. Current & future work

Summary of contributions

- Disambiguation of variability models
  - Separation of concerns — software *vs.* PL variability
  - Formal models — FD (not new), OVM, X-links
- Automated verification
  - internal model consistency
    - not new, but now (more) meaningful !
  - cross-model consistency
- Proof-of-concept prototype using SAT4J
- Application to non-toy (but not real-size) exemplar
  - Private Branch eXchange *[Lee, Kang et al., 2006]*

1. Software Product Lines Engineering
2. Two kinds of variability
3. Objectives and approach overview
4. Internal model verification
5. Cross-model verification
6. Summary of contributions
7. **Current & future work**

Current & future work

- Apply approach to real-size project
  - transitioning OSS PloneGov.org into a SPL

    [Delannay et al., OSSPL'07] [Hubaux et al., SPLC'08]

- Validate and improve notations
  - modularize variability models *[Classen et al., VaMoS'07]*
  - more X-link patterns needed?
  - further formalization and separation of concerns

Current & future work

- Validate and improve tools
  - optimize verifications
    - less naive use of SAT solver
  - identify more checks
  - towards an integrated tool chain for SPLE