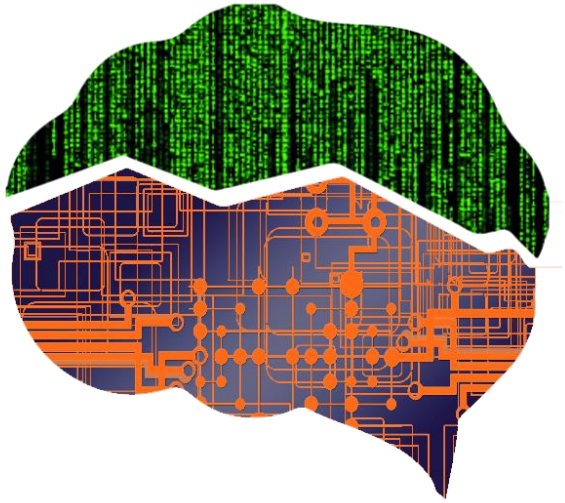
Software Product Line Engineering

Feature Interactions

Christian Kästner (Carnegie Mellon University)
Sven Apel (Universität Passau)
Norbert Siegmund (Bauhaus-Universität Weimar)
Gunter Saake (Universität Magdeburg)



Bauhaus-Universität Weimar

Introduction

- Not considered so far:
  - What if features are not independent?
  - How do features interact?
  - How to keep variability despite of dependencies?
  - How much variability is meaningful?

Agenda

- Feature interactions and their problems
- Solutions to feature interactions
- Discussion: Variability in praxis

Feature Interactions

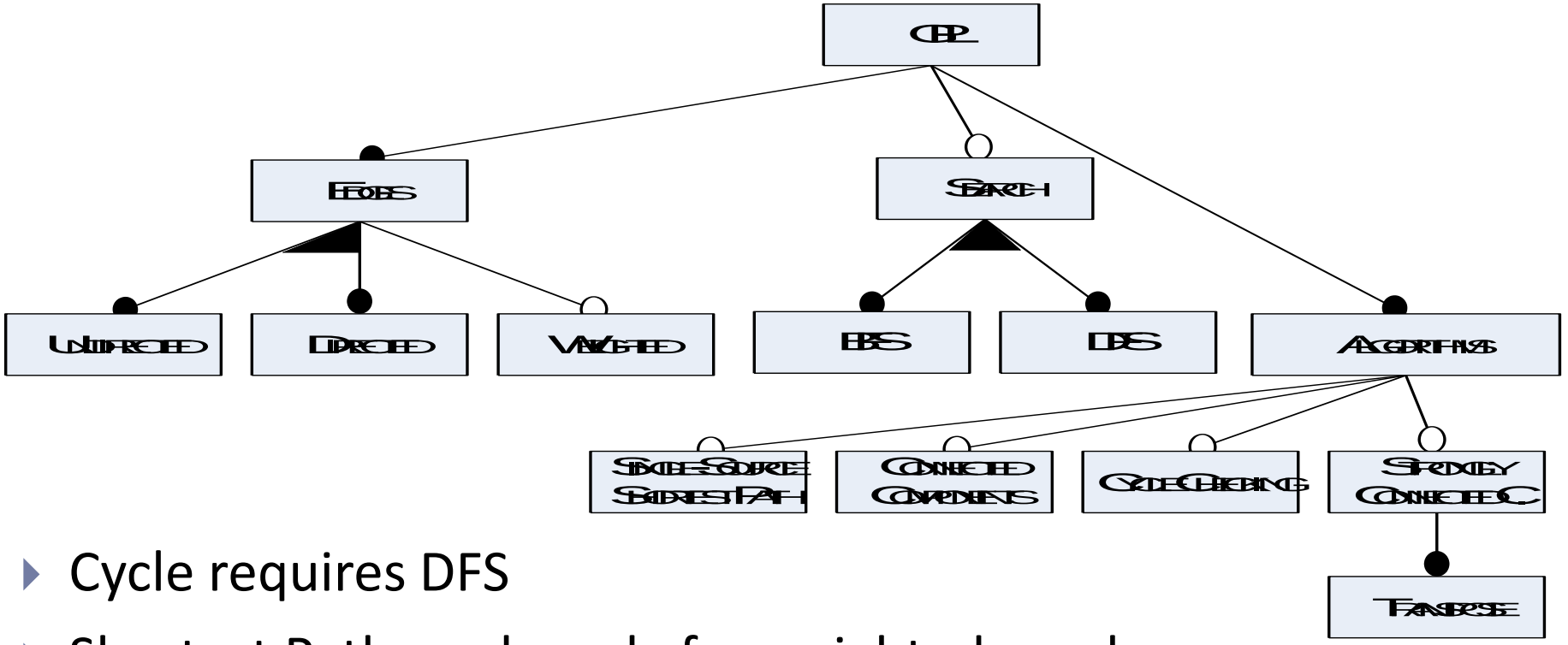
Feature Interactions 1

- Telephone product line: Some phones support „Knocking“, others „delegation when busy“
- What happens if both features are activated?
  - Fee line: no problem
  - Busy line: knocking or delegation?
- Can we (automatically) spot such problems?

Feature Interactions 2

- Flood Control
  - Prevents from flooding by turning off water
- Fire Control
  - Fights fire with water sprinkler

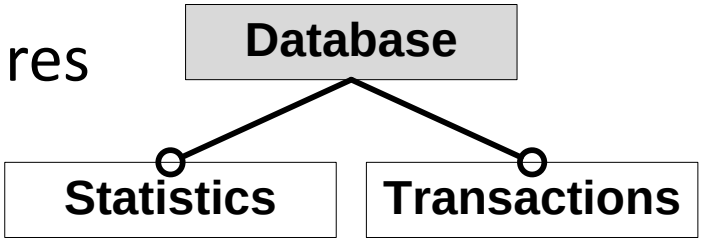
Feature Interactions 3



- Cycle requires DFS
- Shortest Path works only for weighted graphs
- Connected works only for undirected graphs
- Strongly Connected works only for directed graphs and requires DFS

Feature Interactions 4

- Database product line with 2 features
  - Statistic: collects statistics about buffer hit Ratio, transactions, etc.
  - Transactions: Ensures ACID properties



- Both features are optional
  - But: Statistic collects information about transactions, transaction might be used to collect statistical information

- How to implement this such that all variants are possible?

Feature Interactions 5

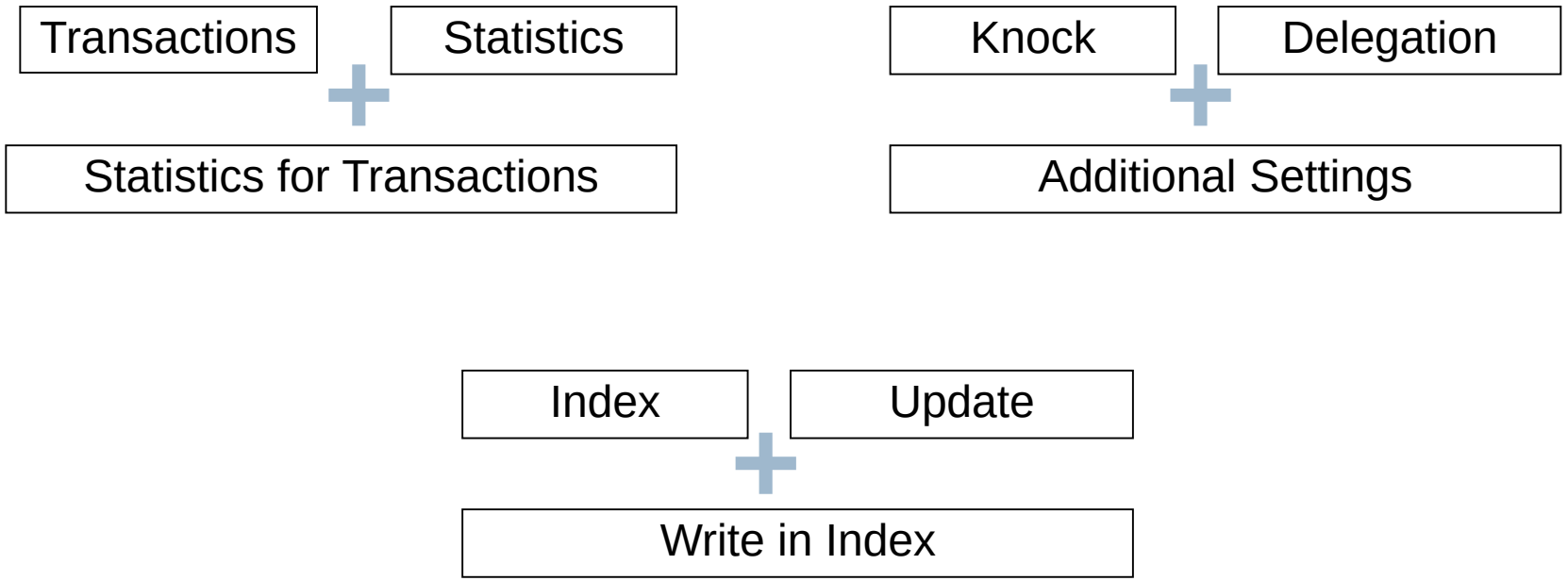
- Database product line with 2 features
  - Index: faster access via B-tree
  - Update: Enables updates to the database
- Both features are optional
  - Efficient read index
  - Writing to the database without index
  - But: if both features selected → write with index
- How to implement such that all variants are possible?

Interactions cause Dependencies

- Features use methods from other features
  - Cycle uses search functionality Graph.search, which was introduced by DFS
  - Shorted Path expects that the method Edge.getWeight is present
- Features extend other features
  - Feature Weighted implements weights by overrding method addEdge from Base
- Features expect a certain behaviour that is specified by another feature
  - Connected expects that edges point always to both directions

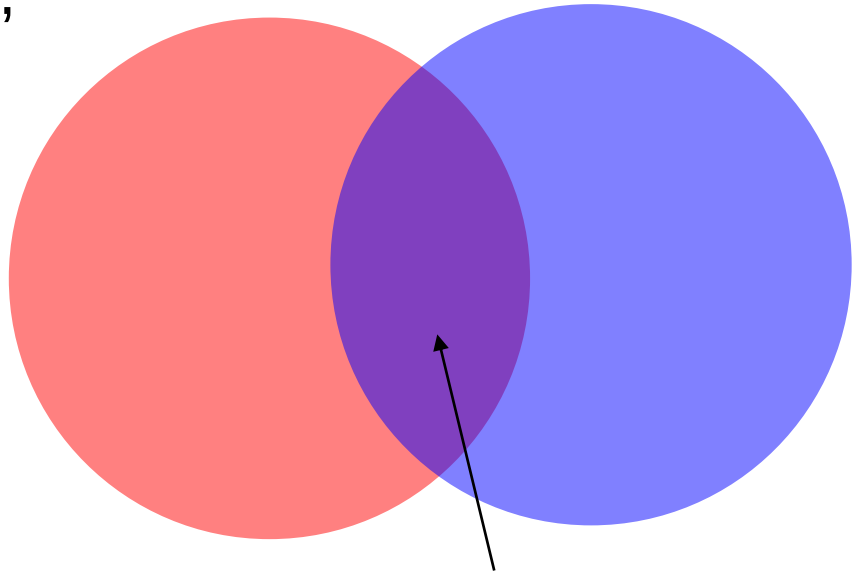
Feature Optionality Problem

- Optional feature behaves correct in isolation
- Problem in combination with other features
- Additional source code coordinates correct behavior



Feature Optionality Problem: Transactions and Statistics

Statistics
(buffer hit ratio, table size, and cardinality, …)

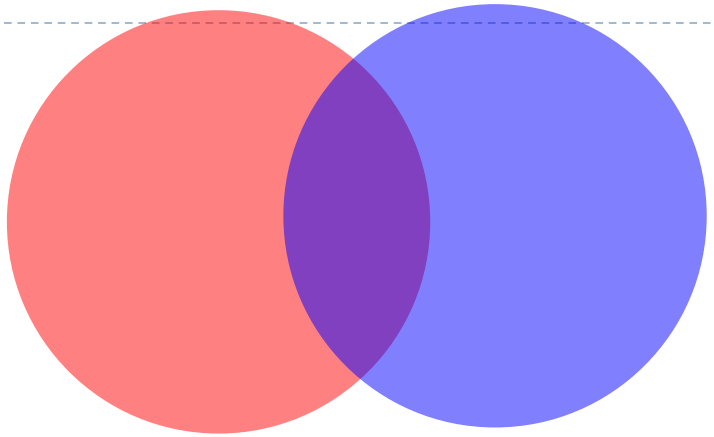


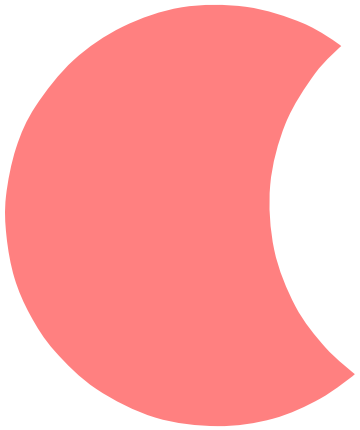
Transactions
(locks, commit, rollback, …)

Throughput measurement
(“Transactions per second”)

Desired Products

Database with statistics and transactions



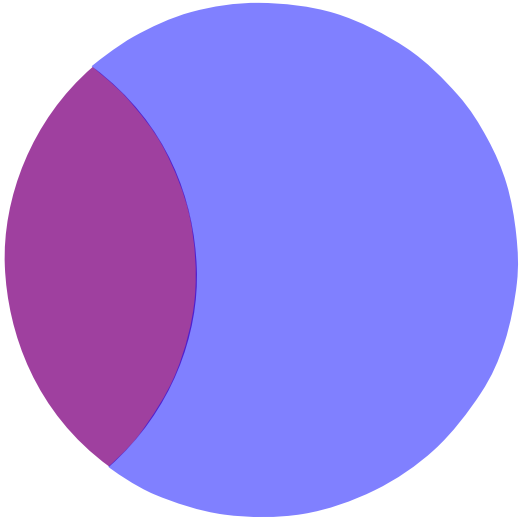


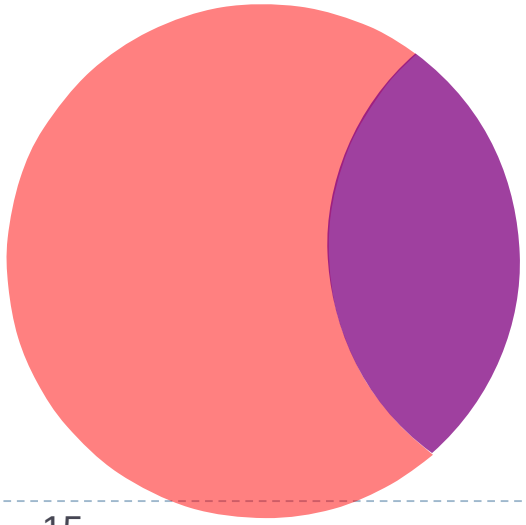
Database with statistics and without transactions

Database with transactions, but without statistics

Unwanted/Impossible Products

Database with transactions and without statistics, which, however, measure throughput





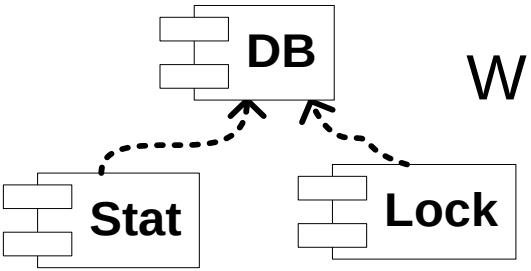
Database with statistics, but without transactions, which, however, measure throughput of transactions (?)

Implementation Example

```
class Database {
List locks;
void lock() { /*...*/}
void unlock() { /*...*/ }
void put(Object key,Object data){
     lock();
     /*...*/
     unlock();
}
Object get(Object key) {
     lock();
     /*...*/
     unlock();
}
int getOpenLocks() {
     return locks.size();
}
int getDbSize() {
     return calculateDbSize();
}
static int calculateDbSize() {
     lock();
     /*...*/
     unlock();
}
}
```

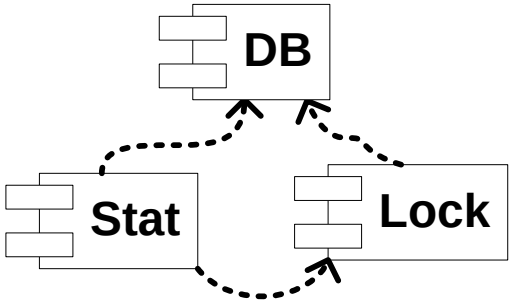
- Locks (blue)
- Statistics (red)
- Features overlap at 2 positions (violet)
  - Statistics over Locks
  - Synchronisation of statistic method

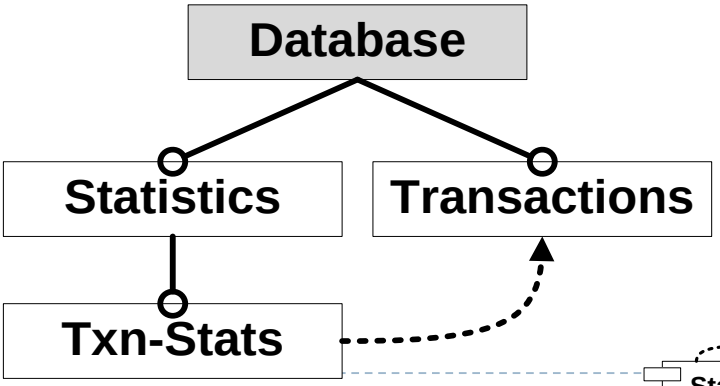
Separation in Modules?



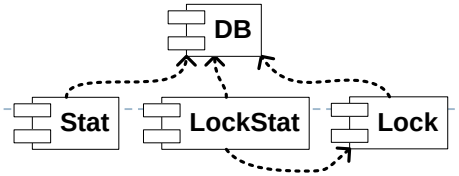
Where to implement the throughput measurement?

How to create a product with statistics but without transactions?





Is the throughput measurement really a feature?

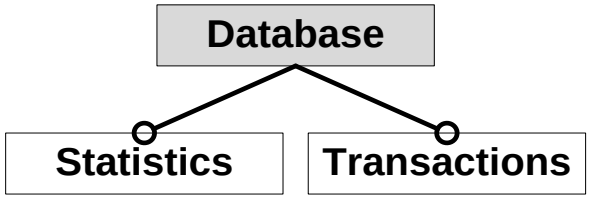


Variability

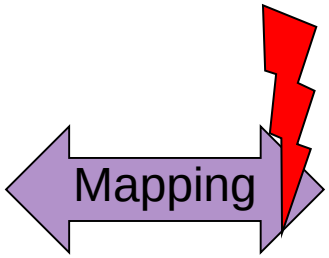
- Describes how many products can be created from a product line
- A product line with n independent, optional features gives rise to 2^n products
- Dependencies and constraints among features limit the variability of a product line
- A single constraint „Feature A requires B“ reduces the number of possible product by 25%

Limited Variability by unsuitable Modularization

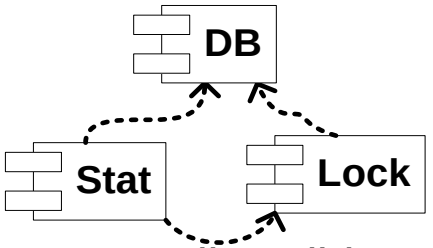
Feature Model



Desired:
4 Products

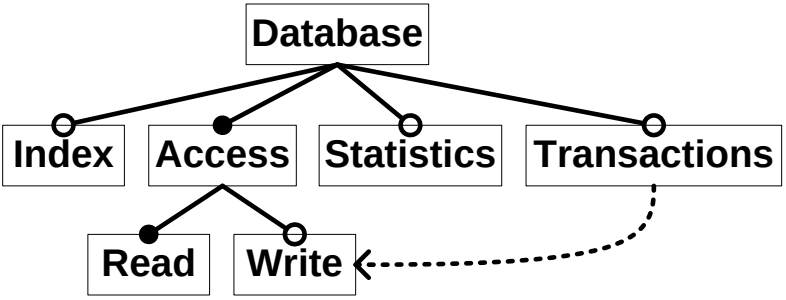


Implementation

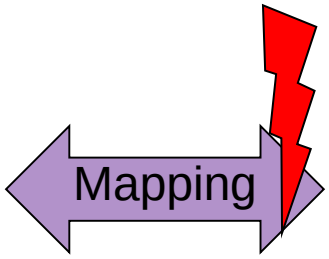


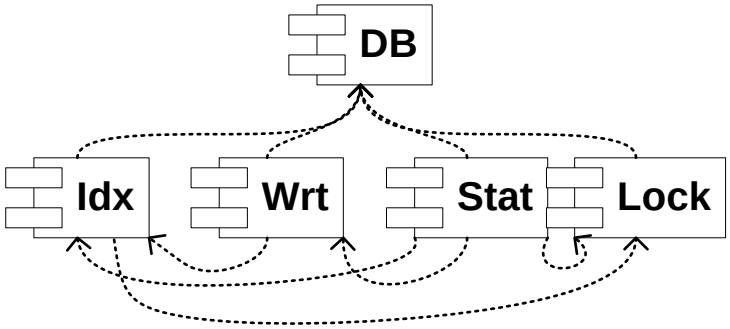
Actually valid:
3 products

Limited Variability



Desired:
12 products





Actually valid:
5 products

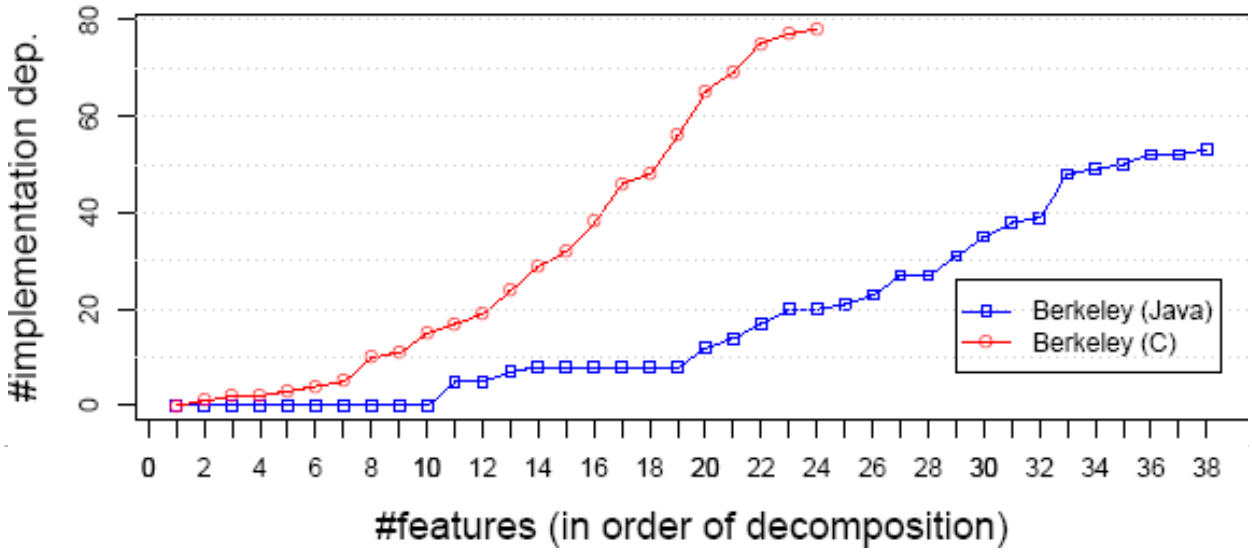
Experience: Berkeley DB

Java version

- Feature model
  - 38 features
  - 16 constraints
  - 3.6 Billion variants
- Implementation
  - 53 implementation constraints (limited variability!)

C version

- Feature model
  - 24 features
  - 8 constraints
  - 1 Millionen variants
- Implementation
  - 78 implementation constraints (limited variability!)



Dependencies are transitive

- „A requires B“, „B requires C“ → „A requires B and C“
- Result: Individual features can require the selection of many additional features and, thus, substantially limit the variability
- Example: Berkeley DB
  - The statstic feature collects statistics of different areas of the program, such as memory consumption, transactions, write accesses, buffer hit ratio, etc.
  - The selection of the statistic feature requires the selection of 14 (from 37) additional features, e.g., transactions, caches

Resolution of Feature Interactions

Classification of Feature Interactions

- Dynamic vs. static
  - Dynamic: At runtime of a variant; unexpected program behavior, crashes, race conditions
  - Static: At generation or compile time; e.g., calling method that is not defined
- Domain vs. implementation
  - Domain constraints: Constraint originates from the conceptual level; alternative implementations have the same constraints
  - Implementation dependencies: Dependencies that are caused due to the chosen implementation; alternative implementation is possible

Dynamic Feature Interactions

- Hard to detect
- Much research in telecommunication systems
  - M. Calder, M. Kolberg, E.H. Magill, S. Reiff-Marganiec. Feature interaction: A critical review and considered forecast. *Computer Networks*, Volume 41, Issue 1, 2003, pp. 115-141
- At requirements analysis, specific focus on identifying interactions using special modeling
- Formal specification, model checking, …
- Testing, testing, testing ...

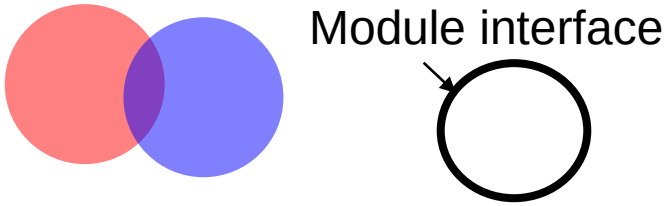
- If found→ Feature Optionality Problem

Focus: Implementation Dependencies

- Implementation dependencies are unpleasant
- Reduce variability, although variants in the domain might be possible
- Example: Transactions vs. Statistics
  - Solution 1: In feature model constraint with Statistic Transactions ➔ Reduced variability
  - Solution 2: No statistic about transactions➔ bad implementation
- Try to find: Possibilities to resolve implementation dependencies

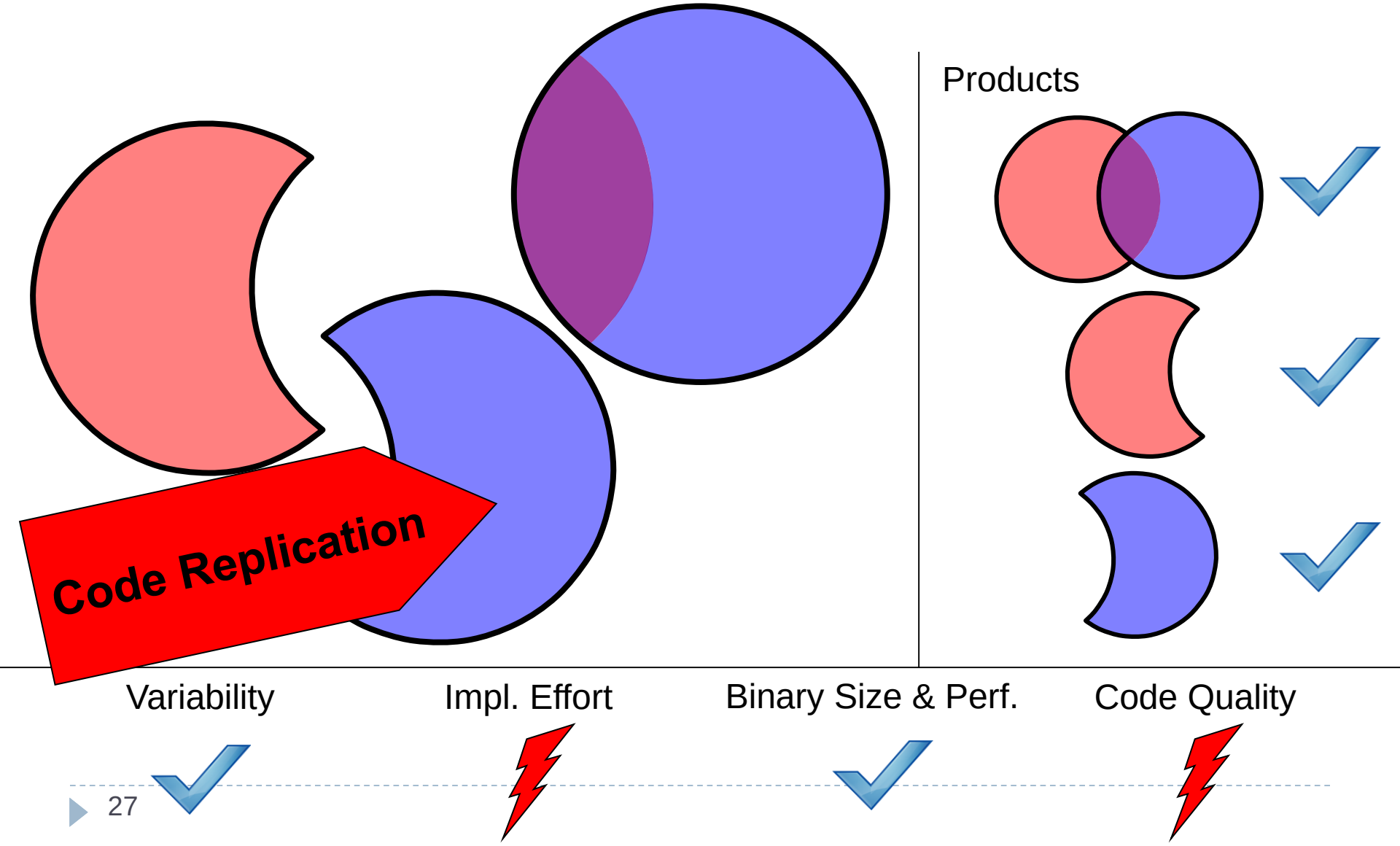
Solutions to the Feature Optionality Problem

- How to modularize two interaction features?

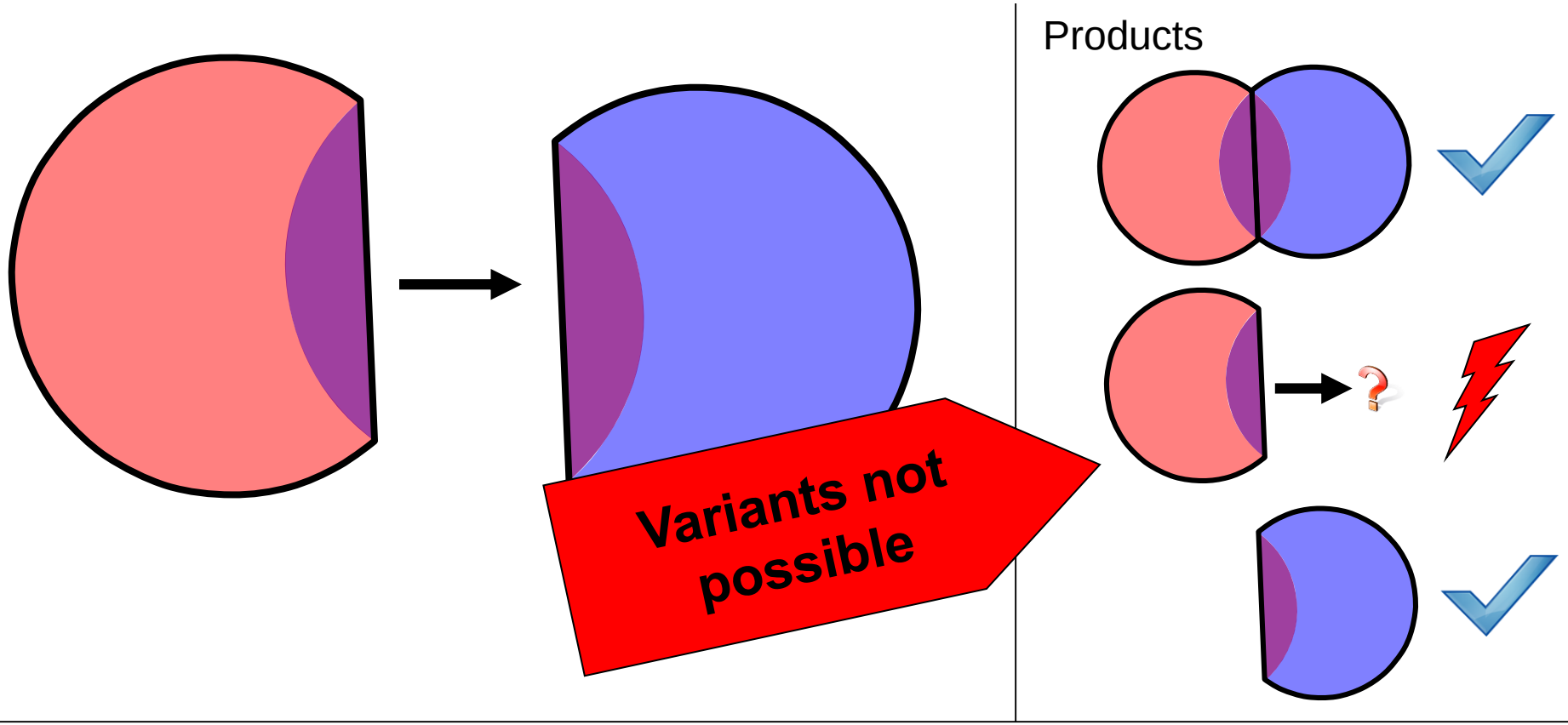


- Goals:
  - Variability as defined in the feature model
  - Small implementation effort
  - Efficient implementation (code size, performance)
  - Code quality (separation of concerns, modularity)

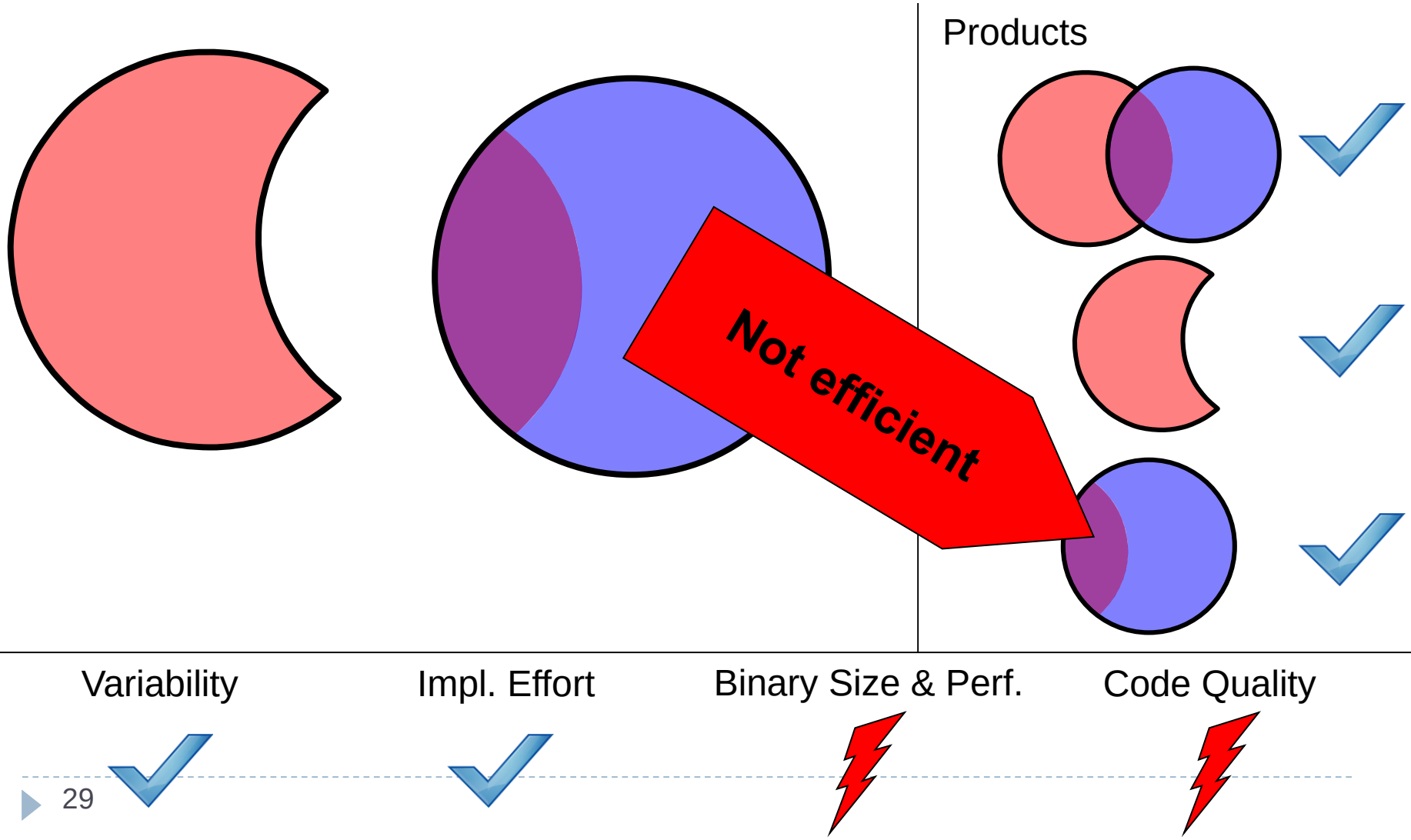
Solution 1: Multiple Implementations



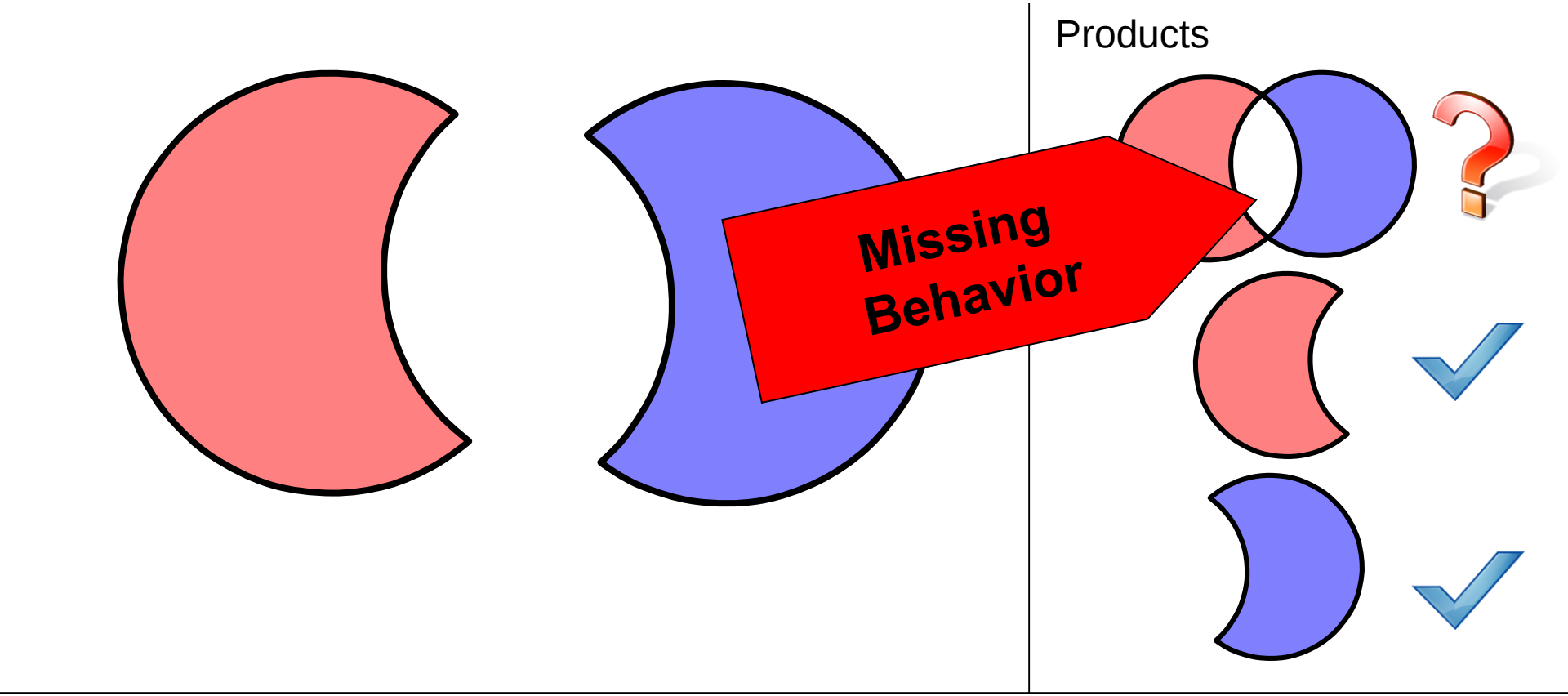
Solution 2: Keep Dependencies (document in the feature model)



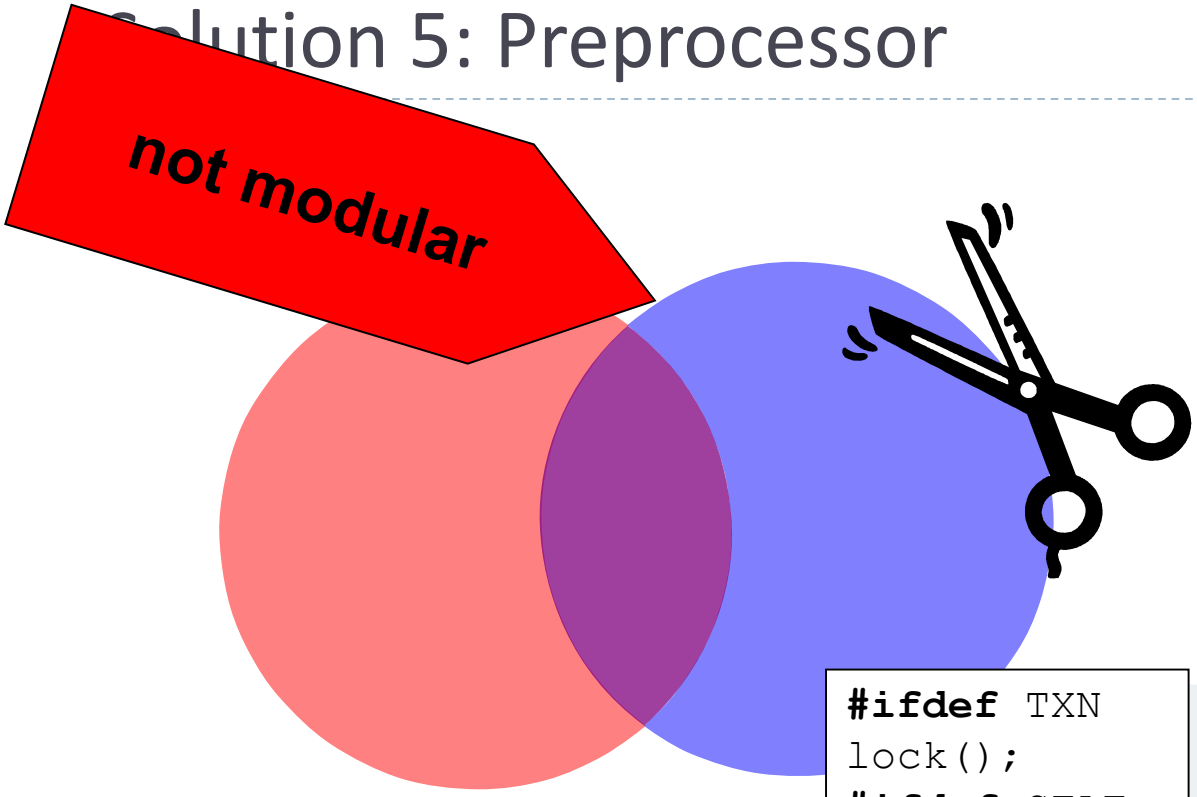
Solution 3: Move Source Code (until there are no dependencies)



Solution 4: Change Behavior (orthogonal Implementations)

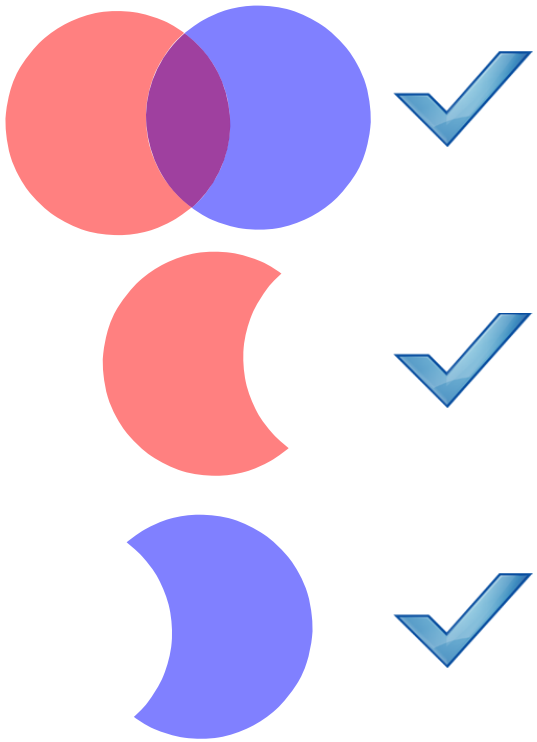


Solution 5: Preprocessor



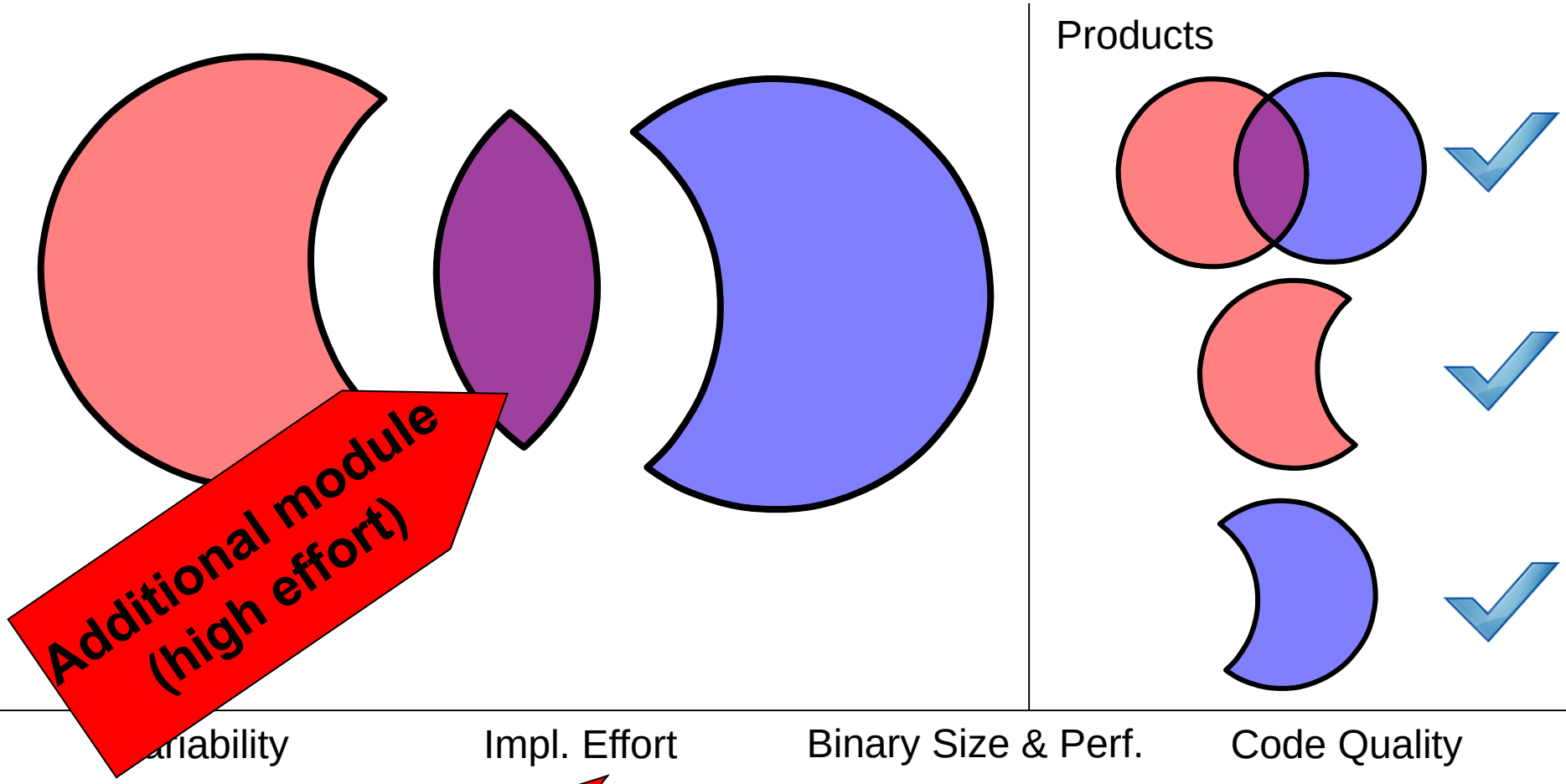

```
#ifdef TXN
lock();
#ifdef STAT
lockCount++;
#endif
#endif
```

Products



Variability	Impl. Effort	Binary Size & Perf.	Code Quality
✓	✓	✓	⚡

Solution 6: Extraction of Interaction (Glue-Code-Module)



Overview of Solutions

Solution	Variability	Effort	Size & Performance	Quality
Multiple Implementations				
Keep Dependencies				
Move Source Code				
Change Behavior				
Preprocessor				
Extract Interaction				

No single best solution

Example in Detail: Extraction of Interaction

- Feature modules contains only its own code, no interaction code
- New module (A#B) will be automatically select when A and B are selected
- A#B extend A or B if both are selected
- With this, all 4 products are possible
  - without A, without B
  - with A, without B
  - without A, with B
  - with A, with B (and with A#B)
- „Optimal“ implementation for all variants

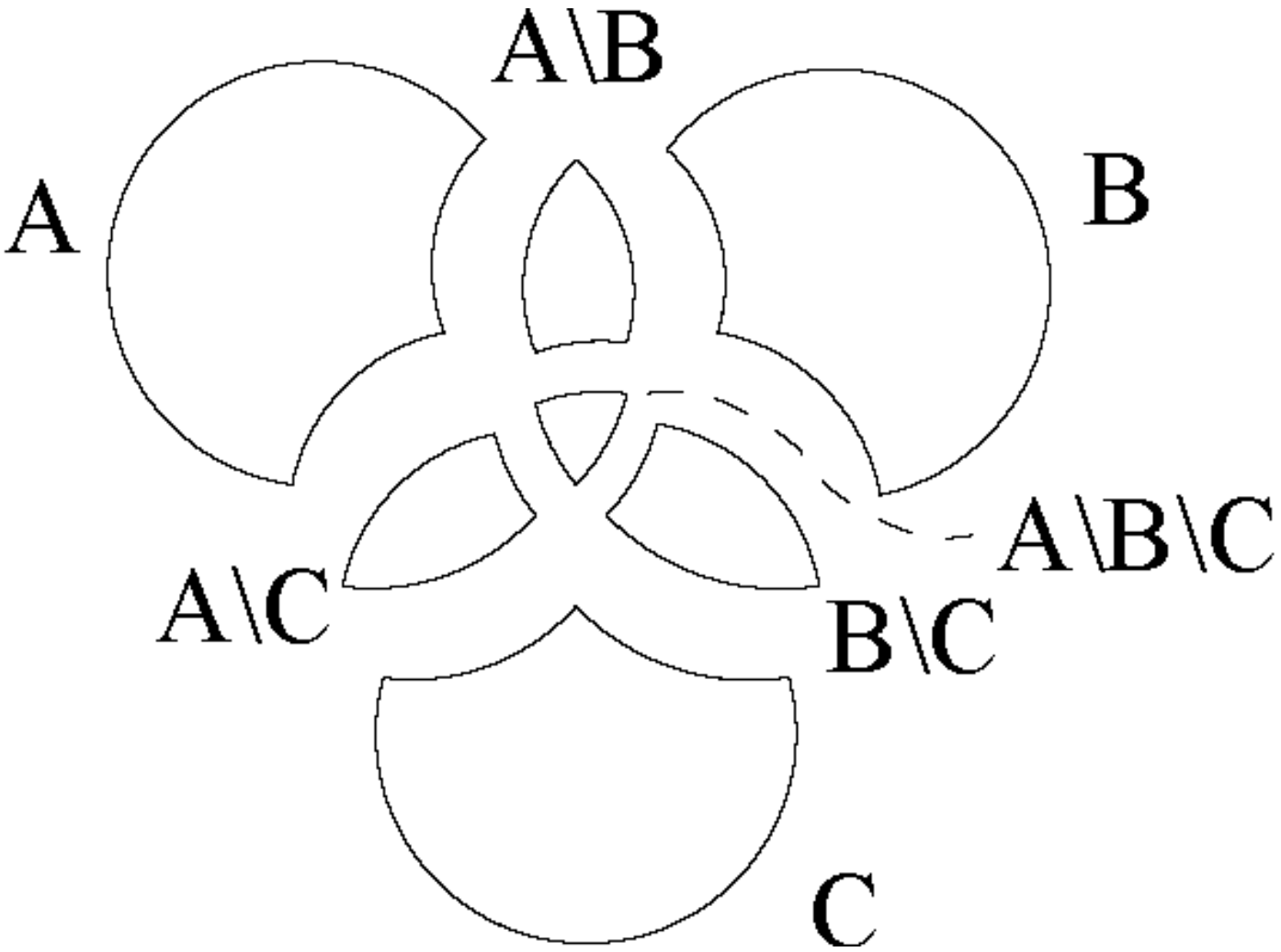
Problems

- High effort for Manuel extraction
- Additional module ➔ higher complexity
- Interactions are often heavy distributed and heterogeneous extensions
- Theoretically many interactions possible

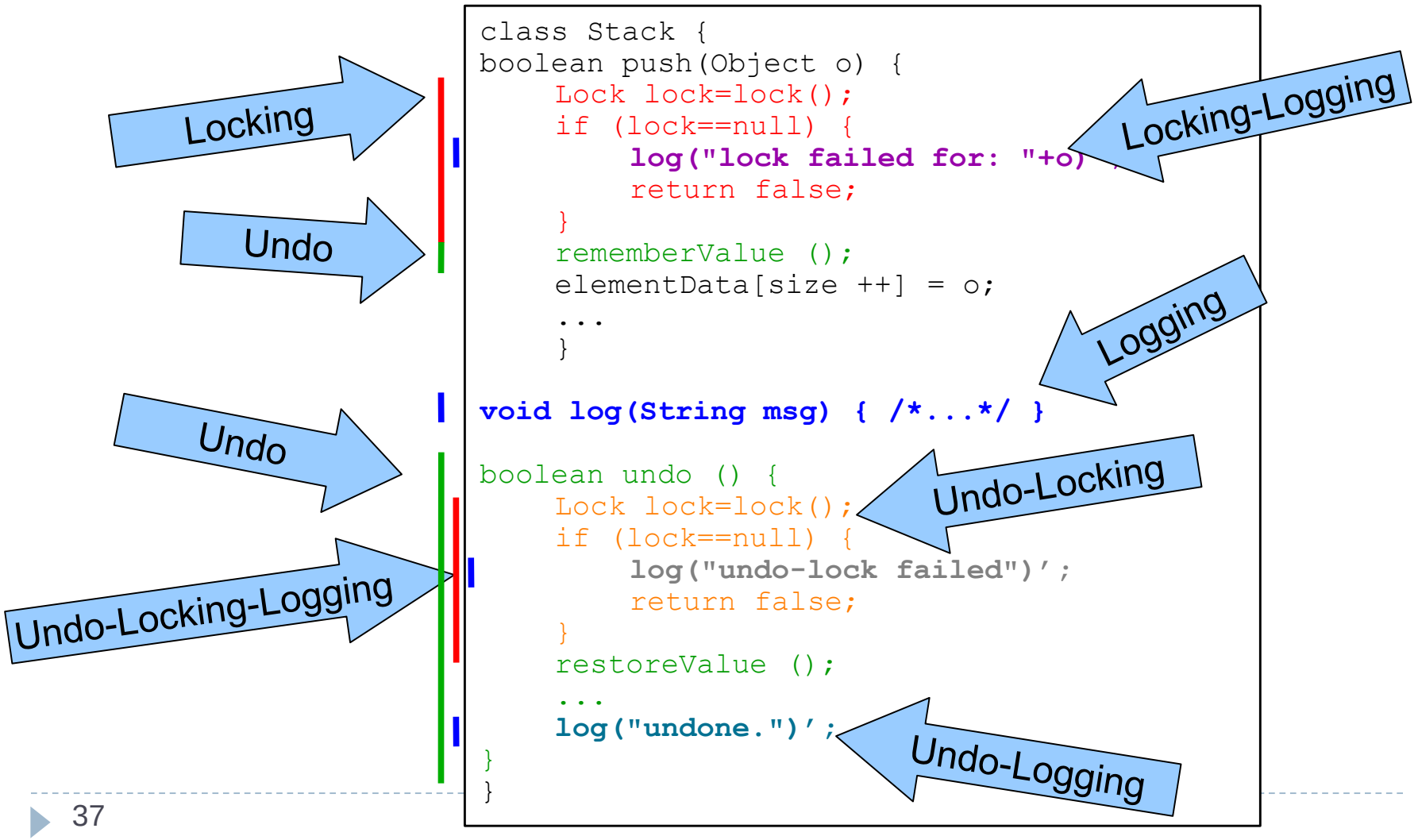
$$i_{max} = \binom{n}{2} = \frac{n^2 - n}{2}$$

- Also interactions between more than 2 features possible

Higher Order Interactions

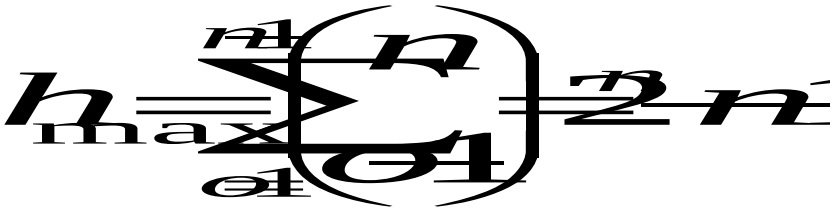


Example of Higher Order Interactions



How many Interactions?

- Theoretical upper bound:



- In practice
  - Substantially less
  - But, still more than features

Experience

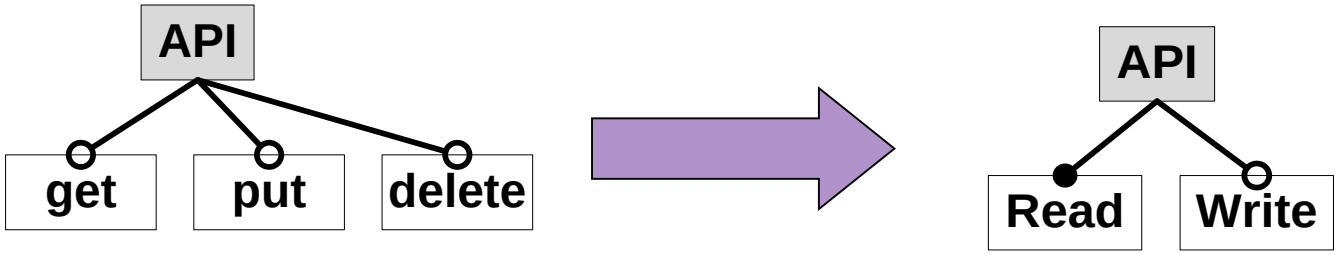
Experience with Berkeley DB

- Keep dependencies?
  - Important features were de-facto obligatory (statistics, transactions, memory management, …)
- Change behavior?
  - Wanted to keep existing behavior
- Extraction of interactions?
  - 76% of statistic code extracted into 9 modules
  - → possible, but high effort
- Preprocessor?
  - Faster, easier
  - Substantially scattered and tangle concerns




Experience with FAME-DBMS

- Change of feature model
  - Avoided 14 dependencies and lost ¾ of possible products



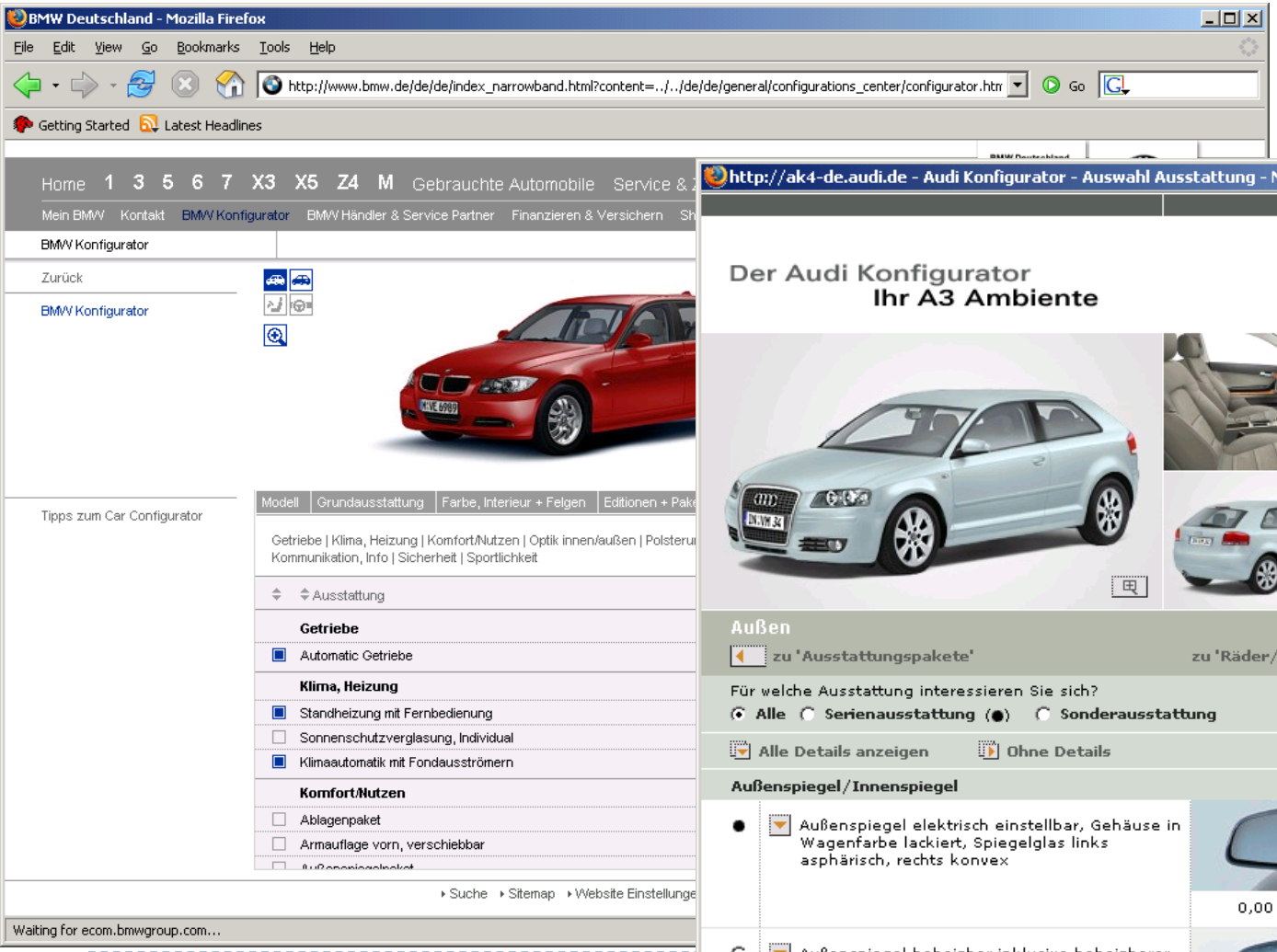
- Logging with preprocessor
  - Avoided 11 dependencies, but scattered source code
- B-tree does not always support write operation
  - Increased binary size by 5—13%
- 10 remaining interactions extracted

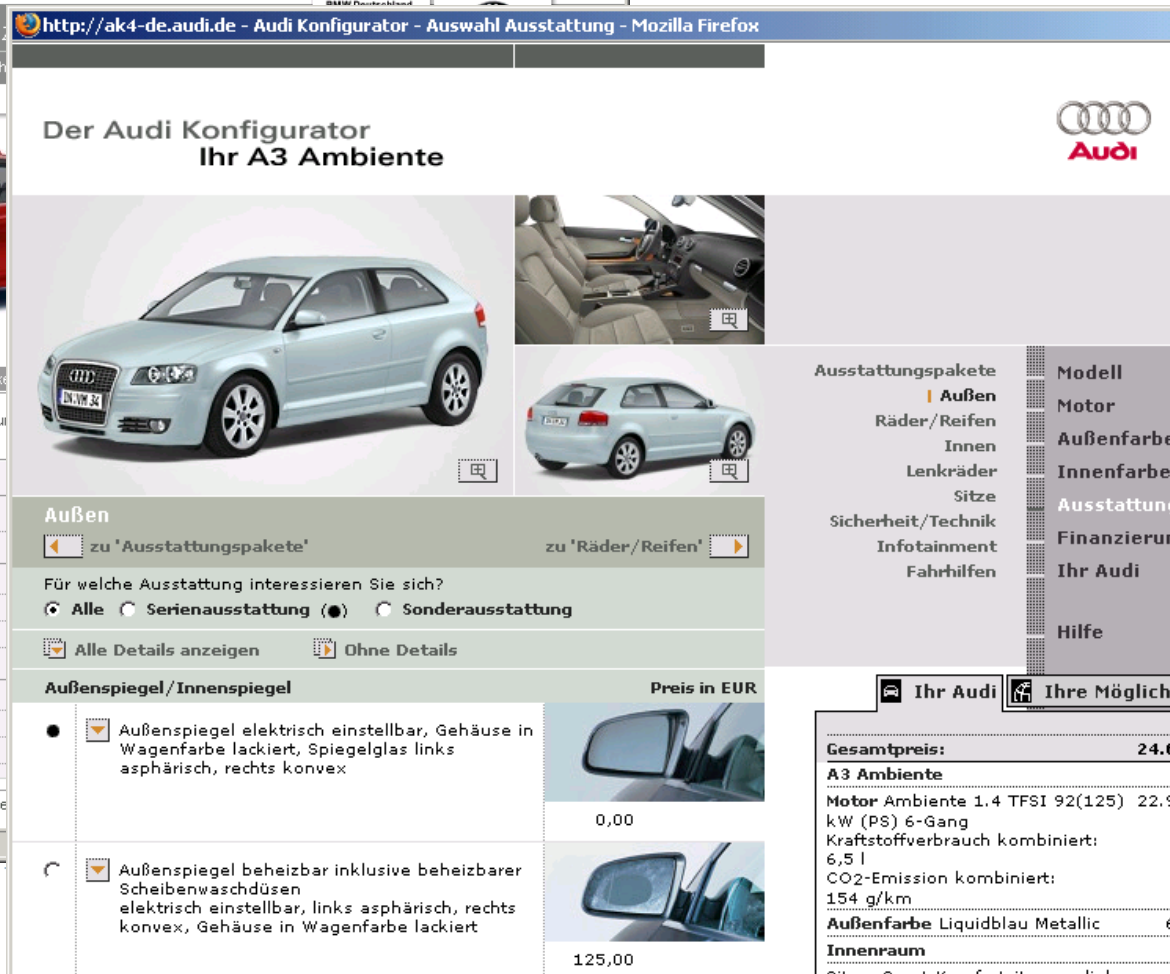
Discussion: Variability in the Wild

Which Interactions to Extract?

- Variability is not an end in itself
  - With 33 optional, independent features, there is one variant for each human on the planet
  - With 320 features, there is one variant for each atom in the universe
  - Nobody can test all variants; nobody needs all these variants
- So
  - Focus on actually required variants
  - Variability management, variability at the right spot, domain analysis

Recap: Automobile Product Line (BMW, Audi)





Automobile Product Lines 20 Years Ago

- Choice of car was limited to the type and some small extras, such as cassette deck
- One single variant (Audi 80, 1.3l, 55PS) was responsible for 40% of the sales

Automobile Product Lines Now

- 10^{20} variants of a single Audi;
  10^{32} variants of a single BMW
- Nearly no identical car leaves production
- Just the base platform has 100 different variants for a single model depending on engine and extras
- There are 50 different steering wheels (3 vs. 4 spokes, wood vs. plastic vs. leather, heating, colors, etc.)

Problems of Automobile Product Lines

- High number of variants caused huge costs and complexity
  - High logistic costs
  - Development effort
  - Investments in tools and construction capabilities
  - High construction costs due to low number of items, especially for bought items
- „However, we need enough variants to satisfy our customer needs.“
- Idea: Variant management as strategic project; incorporates developers, logistics, marketing

Variant Management at Automobile Product Lines

- First analyse what combination is actually asked for
  - Exotic variants will be removed
  - „Do not develop and construct items that will never be used “
- Apply variant management early at product development
  - Audi could save 5 Millionen Euro by reducing the variants of the ceiling module via the usage of a common control element that fits for all variants
  - BMW reduced the number of ground elements from 100 to 4 : Right-/Left wheel, with/out sun roof;

Variability in Software Product Lines

- Provide required variability
- Avoid unnecessary variability
  - Reduced development effort
  - Reduced testing
  - Reduced maintenance
- E.g., decouple Shortest Path from Weighted, of there is need for it, otherwise keep the constraint
- E.g., split statistics and transactions with glue code only if there is need for it

Summary

- Dependencies between features due to feature interactions
- Resolve implementation dependencies with additional modules
- Variant management is meaningful

Outlook

- Feature interactions as an open research field
- Interactions between features are an important variability problem of software product lines
- Dynamic interactions are hard to identify
- Do formal specifications or tool support help?

Literature

- J. Liu, D. Batory, and C. Lengauer. Feature Oriented Refactoring of Legacy Applications. In Proc. Int'l Conf. on Software Engineering, 2006.
  [Resolution of interactions with additional modules]
- C. Kästner, S. Apel, S. S. ur Rahman, M. Rosenmüller, D. Batory, and G. Saake. On the Impact of the Optional Feature Problem: Analysis and Case Studies. In Proc. Int'l Software Product Line Conference (SPLC), 2009.
  [Feature Optionality Problem with solutions]
- M.-S. Andres. Die Optimale Varianz. brand eins, 2006(1)
  [Variants in automobile industry]

Quiz

- Illustrate all possible interactions among four features using a Venn-Diagram

- What is the connection of feature interactions and the feature optionality problem?