Reusability of Software Transformations

Günter Kniesel

Dept. of Computer Science III, University of Bonn – Germany

gk@cs.uni-bonn.de • http://roots.iai.uni-bonn.de

How Reusable are Software Transformations?

- Composition Scenarios
- Challenges
- Reuse Scenarios
- Conclusions and Discussion Group Proposal

Composition Scenarios

– Two simple transformations –

– Three types of composition –

Transformation „CopyField“

- CopyField(*SC*,*F*,*TC*)
  - For each field *F* of each source class *SC*
  - ... that fulfills the property source(*SC*,*F*)
  - ... copy *F* to each target class *TC*
  - ... that fulfils property target(*SC*,*F*,*TC*)

CopyField(*SC*,*F*,*TC*) ≡

$\forall\theta$: (P $\vdash$

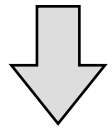
source(*SC*,*F*)θ $\wedge$ target(*SC*,*F*,*TC*)θ

)

→ copy(*SC*,*F*,*TC*)θ

Program P

```
class A {
  public int i;
  long d;
}

class B {
  public int j;
}
```



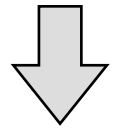
```
class A {
  public int i;
  long d;
}

class B {
  public int j;
  public int i;
  long d;
}
```

Transformation „EncapsulateField“

- Encapsulate(*F*, *C*)
  - For each public field *F* of each class *S*
  - ... that does not have a getter method
  - ... add a public getter method
  - ... add a public setter method
  - ... replace read accesses by getter calls
  - ... replace write accesses by setter calls
  - ... and make F private

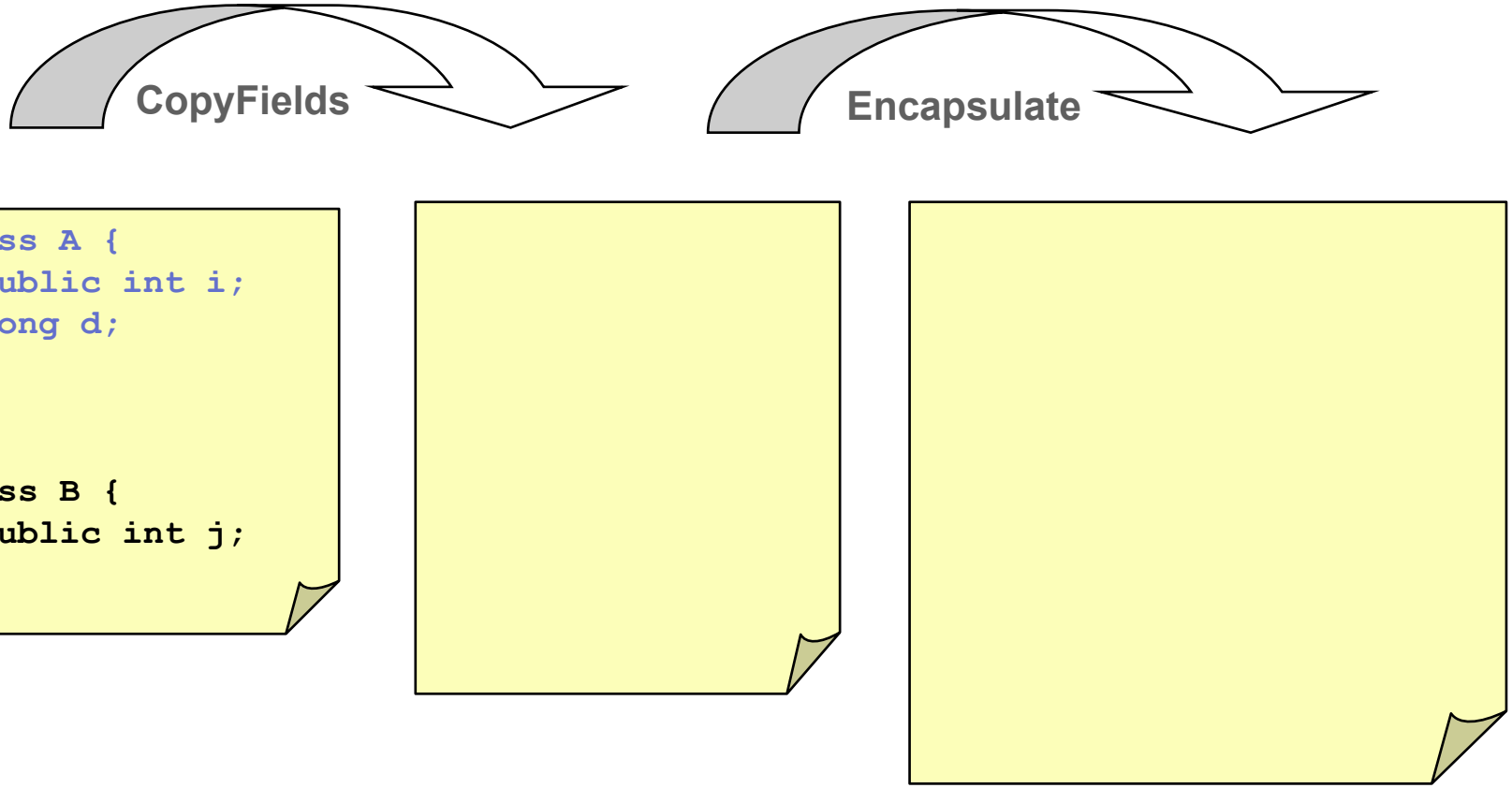
Program P

```
class A {
  public int i;
  long d;
}

class B {
  public int j;
}
```



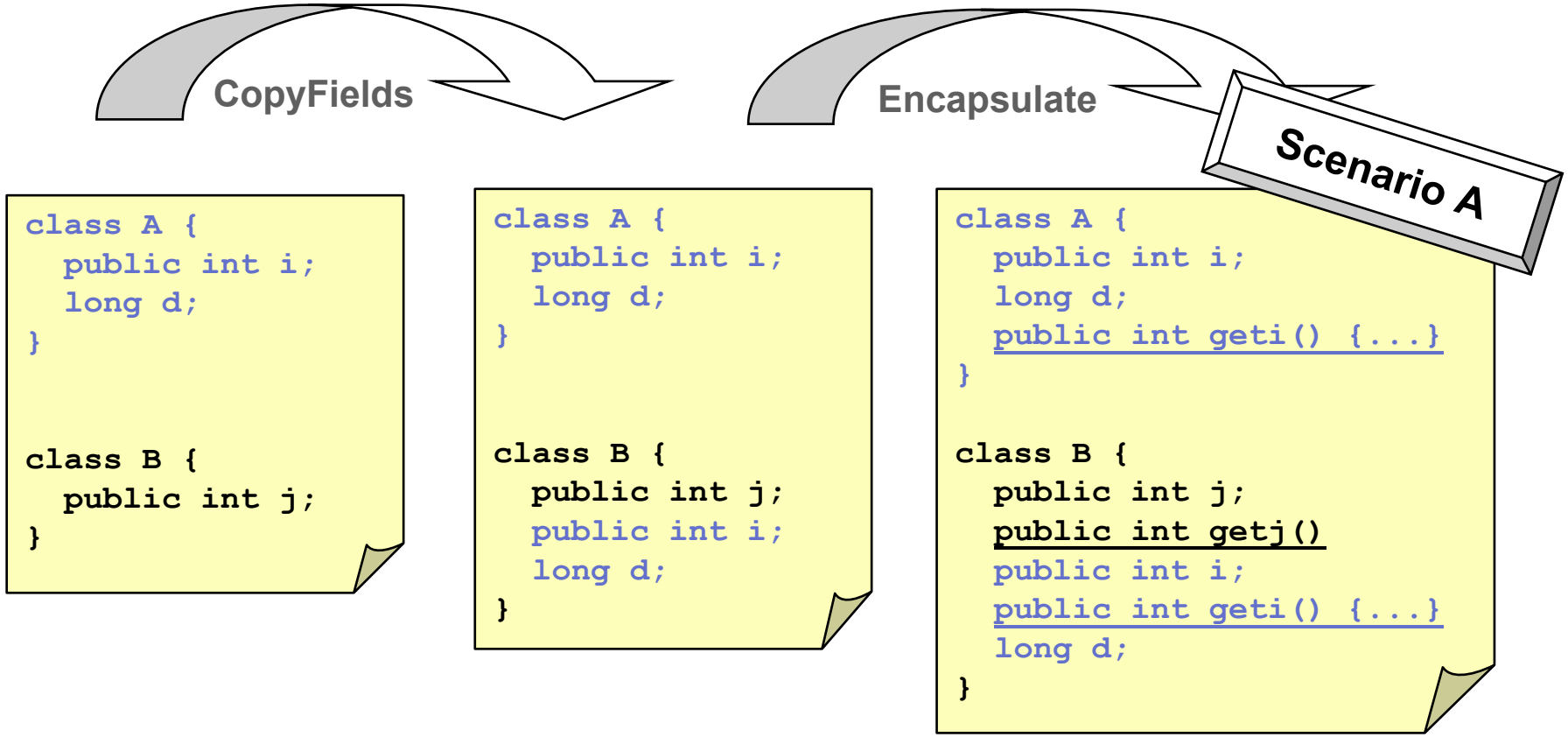
```
class A {
  public int i;
  long d;
  public int geti()
}

class B {
  public int j;
  public int getj()
}
```

Transformation „CopyAndEncapsulate“



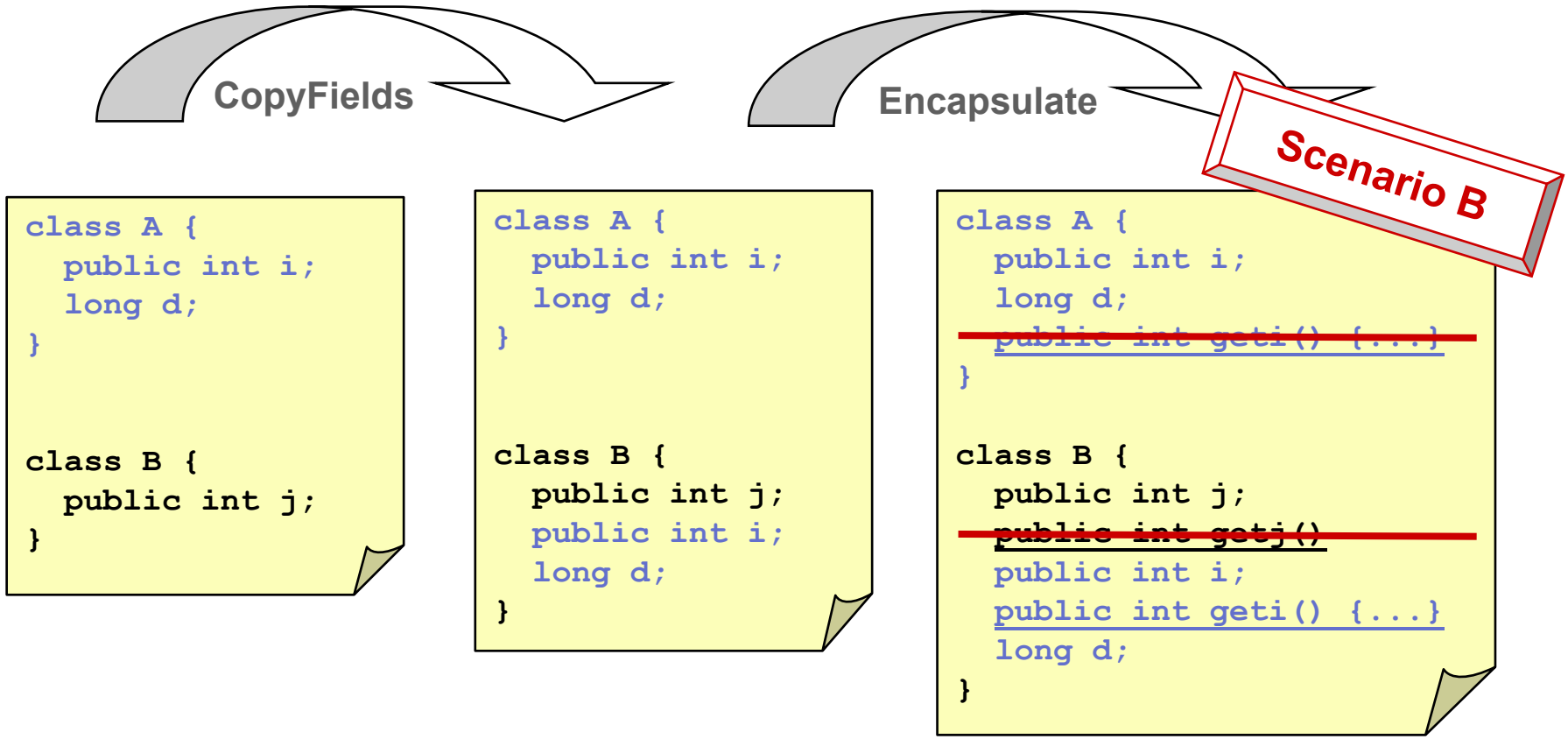
- But how should we compose the two transformations?

Transformation „CopyAndEncapsulate“



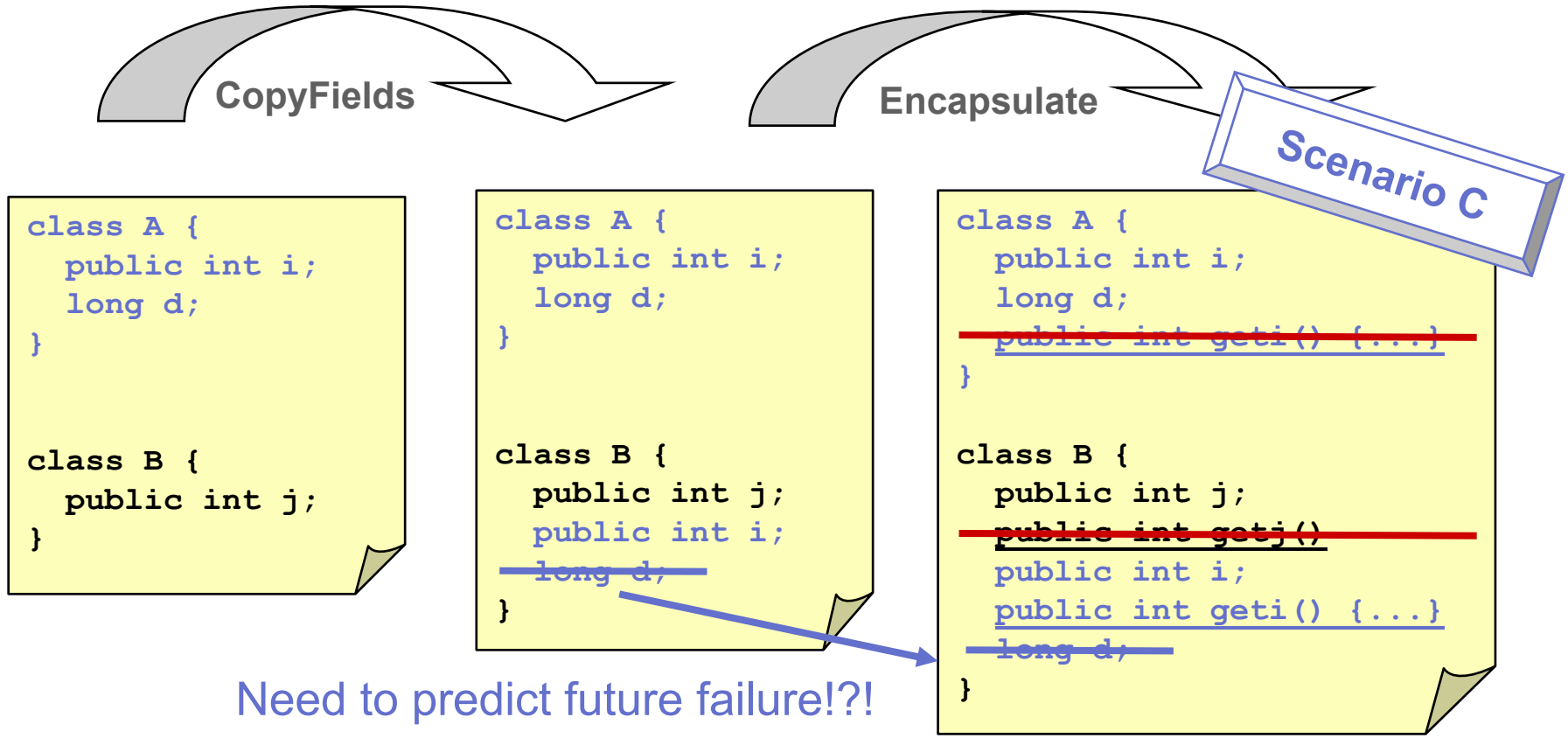
- Like this?
  - Copy from A to B and afterwards ...
  - ... encapsulate all fields (that can be encapsulated).

Transformation „CopyAndEncapsulate“



- Or like this?
  - Copy from A to B and afterwards ...
  - ... encapsulate only the copied fields (that can be encapsulated).

Transformation „CopyAndEncapsulate“



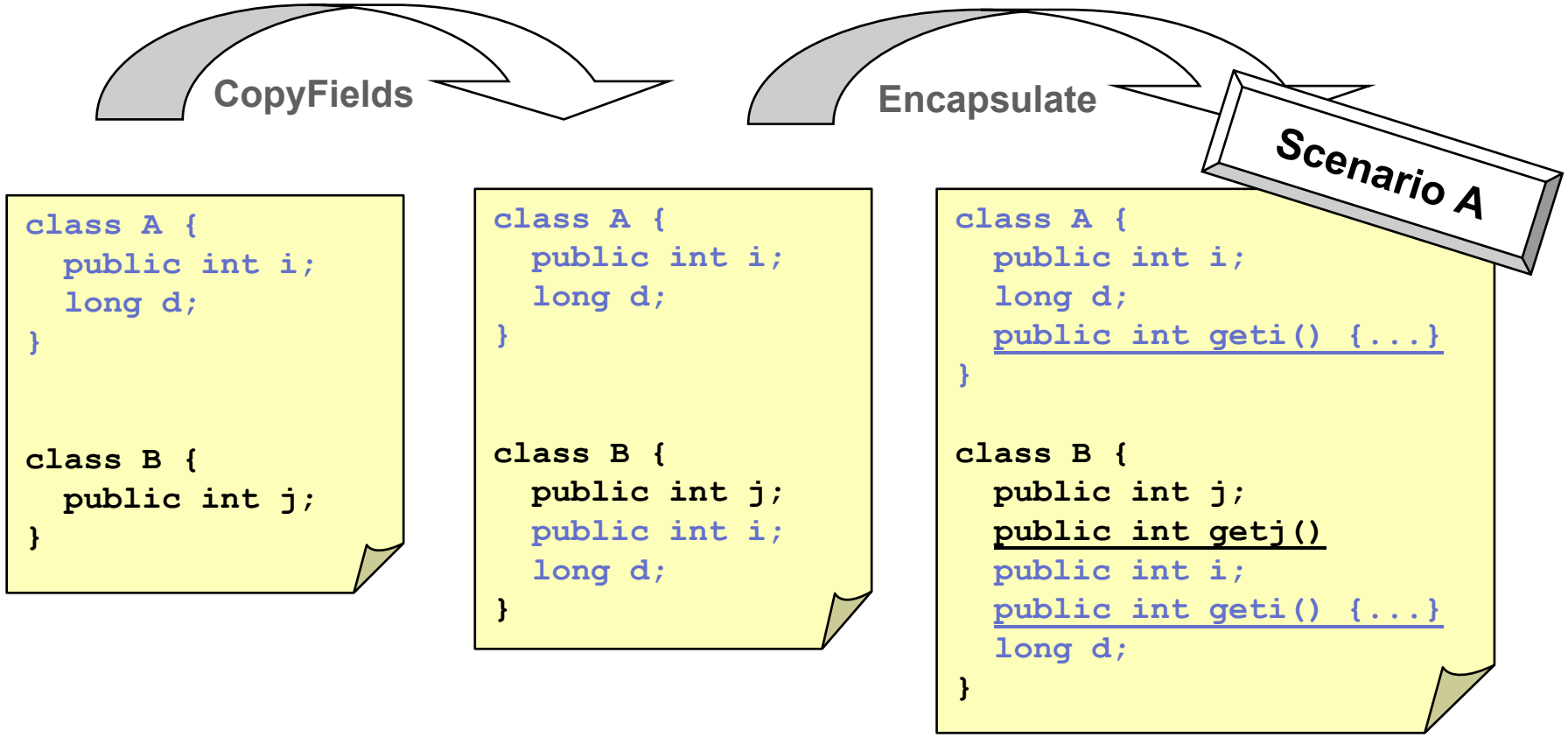
- Or like this?
  - Only copy fields that can be encapsulated and afterwards ...
  - ... encapsulate only the copied fields.

Analysis of Composition Scenarios

– Propagation of „trace information“

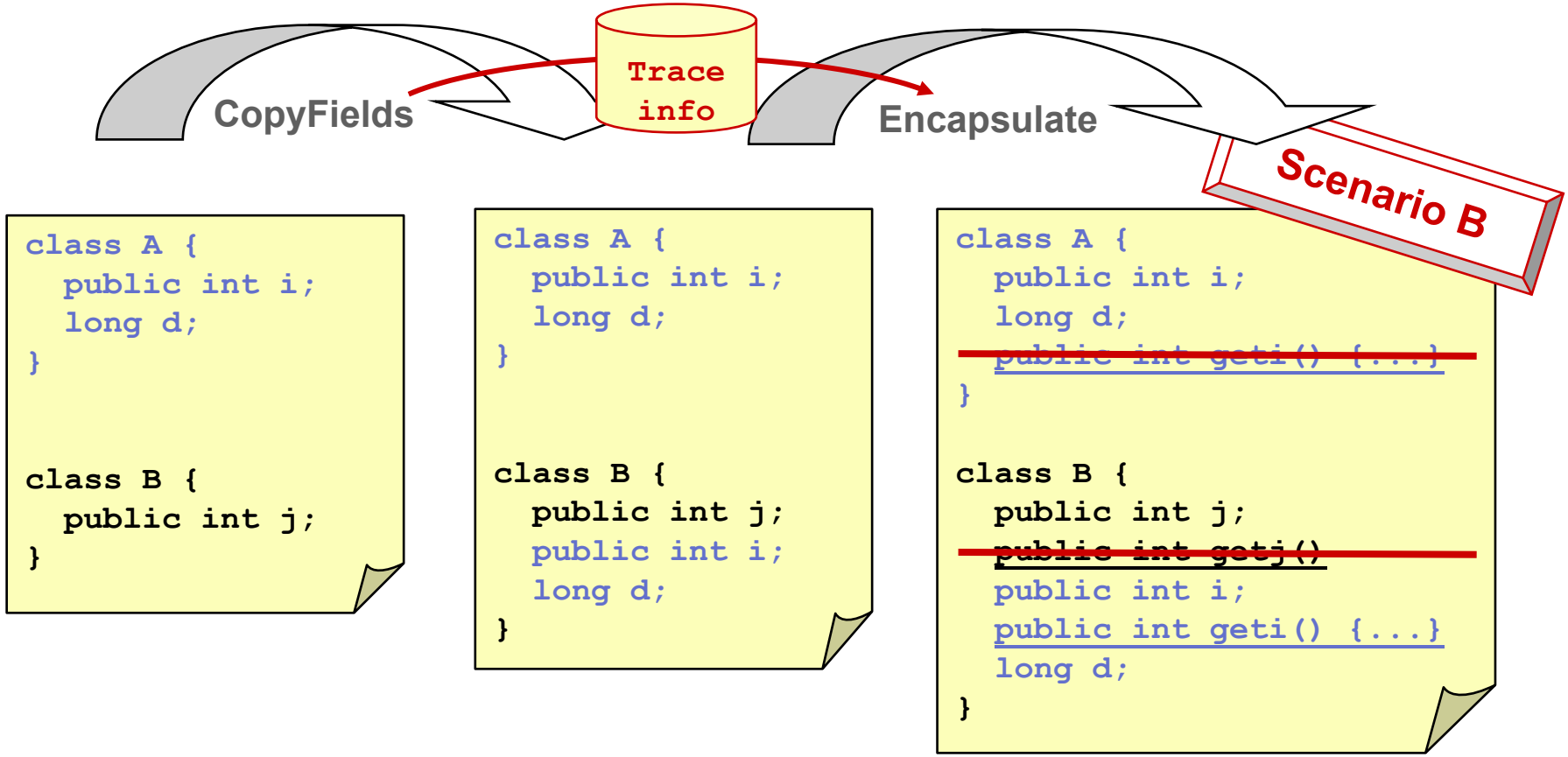
– Challenges

Transformation „CopyAndEncapsulate“



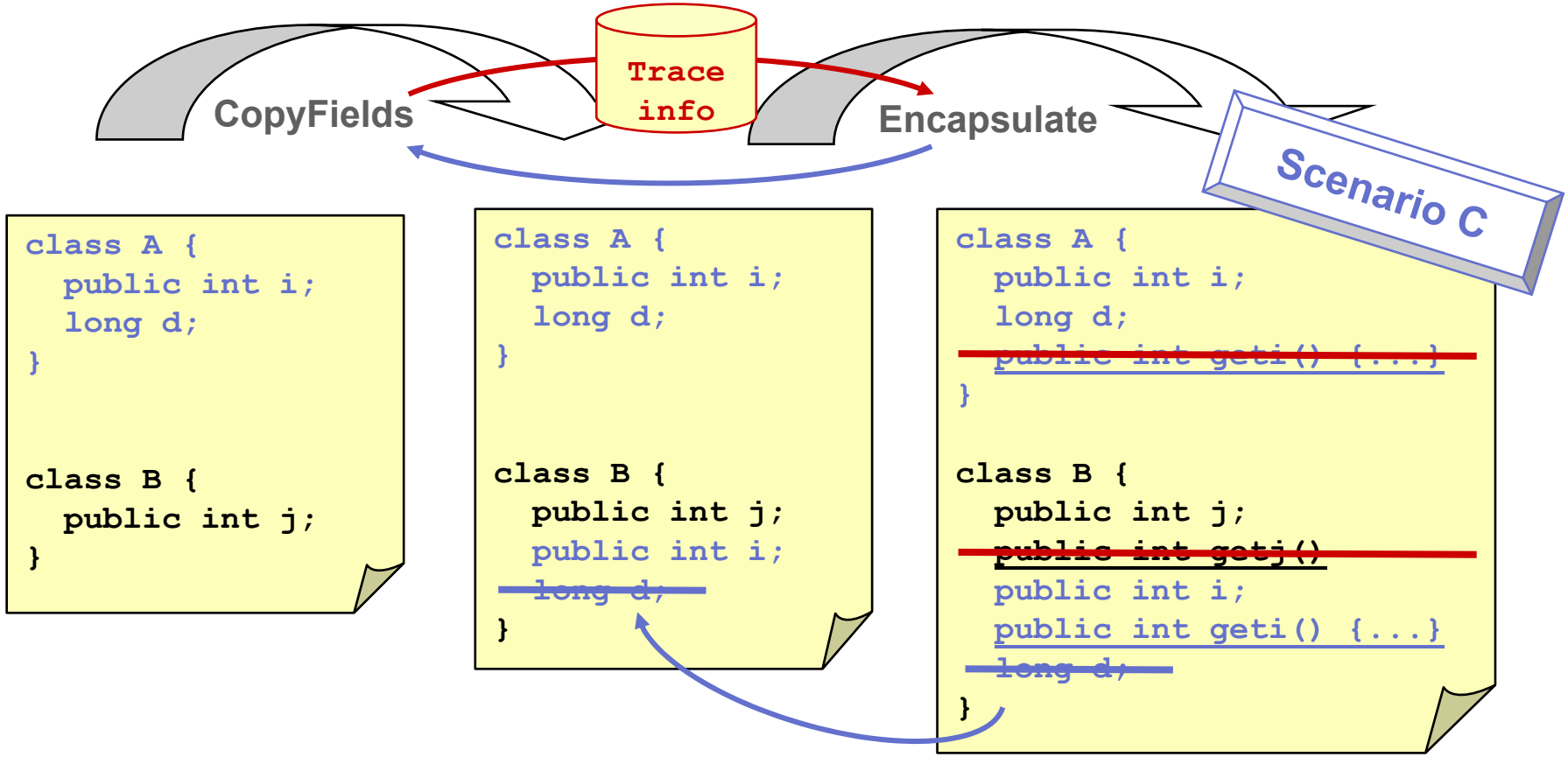
- Independent execution ⇔ „No propagation“
  - The second transformation needs just the current version of the program
  - No need to know what the previous one did

Transformation „CopyAndEncapsulate“



- Dependent execution ⇒ „Forward propagation“
  - The second transformation needs to know the current program...
  - ... and which fields had been copied by the previous one into which class

Transformation „CopyAndEncapsulate“



- Mutually dependent execution ⇔ Bidirectional propagation
  - The second transformation depends on the substitutions of the first ...
  - ... and the first depends on the substitutions of the second.

Challenges

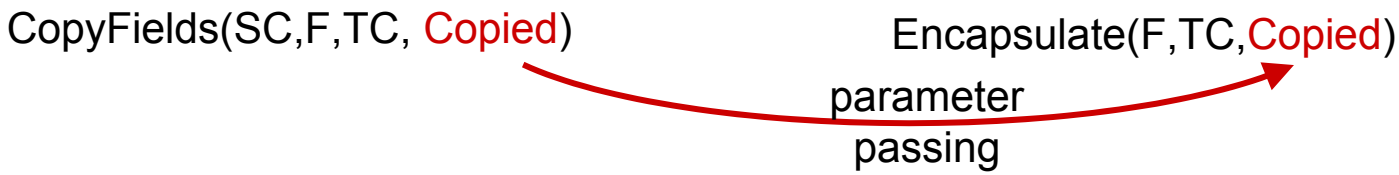
Forward Propagation

Bidirectional Propagation

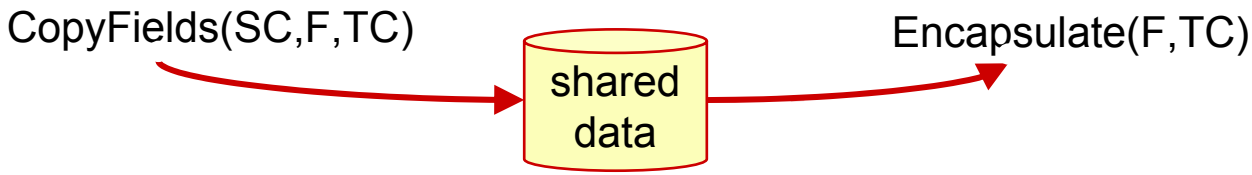
Reuse

Propagation Challenges: Forward

- Forward Propagation: Option 1
  - Add extra parameters for passing trace infos



- Forward Propagation: Option 2
  - Add trace information to the shared program database



- Common Problem
  - Explicit management of history propagation in each transformation module creates hard-wired dependencies → no reuse with other transformations that do not fullfill the assumptions / do not implement the same history passing scheme (for the same type of data!)

Propagation Challenges: Backward

- Additionally pass back information about failure
- Undo failed actions
- ... ??? ... (Proper implementation models???)

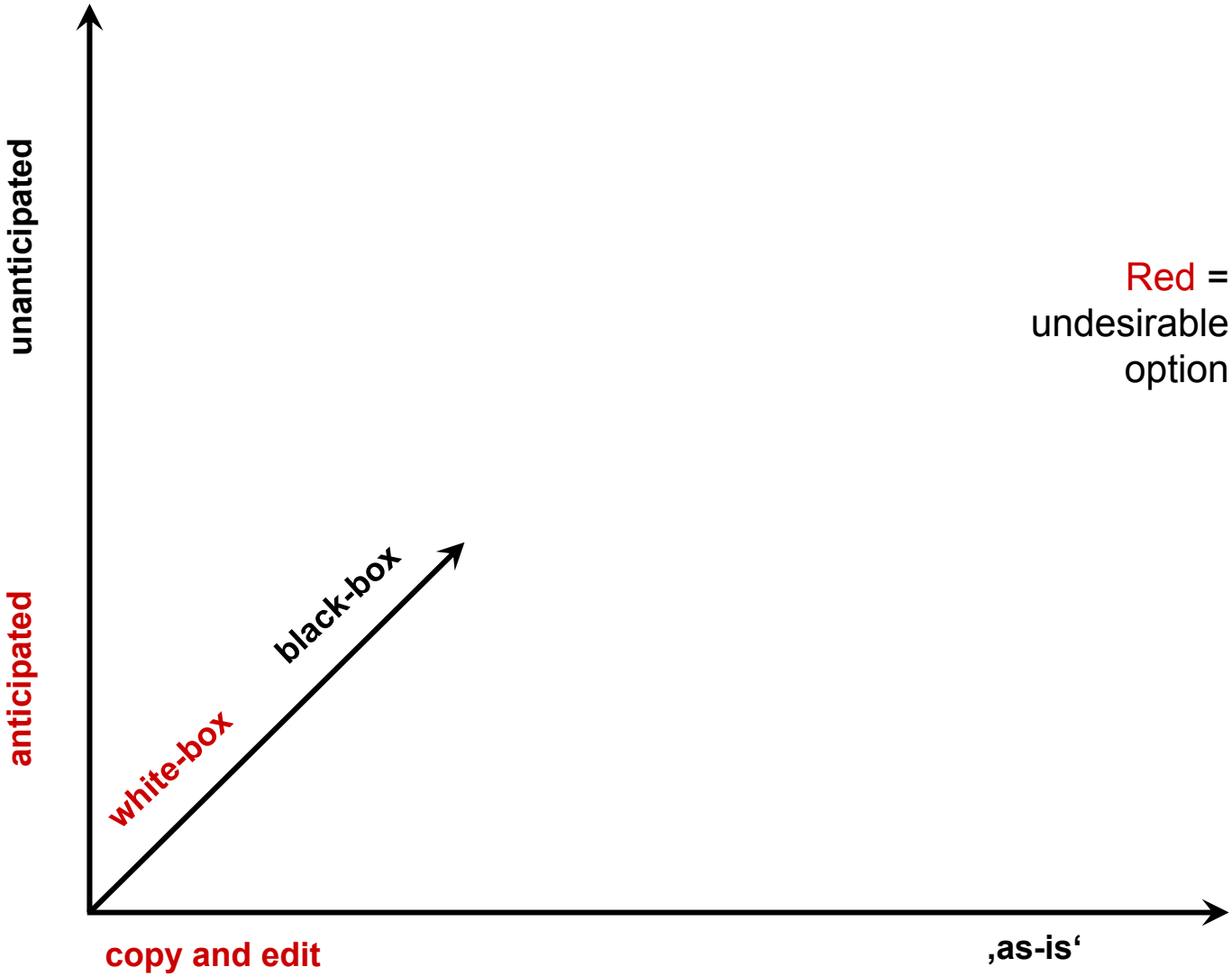
- → Even if there are implementation models, how reusable are the programs that use them?

Reuse Challenges

- How to propagate if we ...
- ... do not know enough about the collaborating transformations to built-in the proper interfaces for working with each other?
- ... do not know that we will need to combine specific transformations?
- ... do not want to change the existing transformations (‚as-is' reuse)?

Reuse Scenarios

Reuse Dimensions and Categories



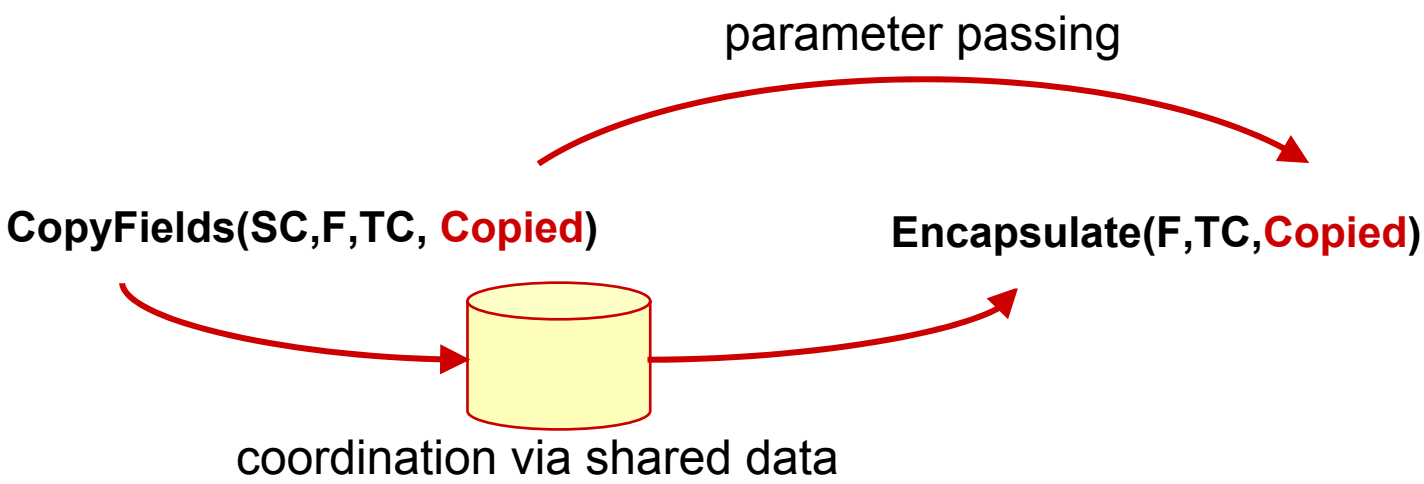
Red = undesirable option

Reuse Scenarios: Overview

- Edit → change the transformations
- Anticipated → prepare the transformations
- Unanticipated, as-is, white-box → add glue code
- Unanticipated, as-is, black-box → ???

Reuse Scenarios: Copy and Edit

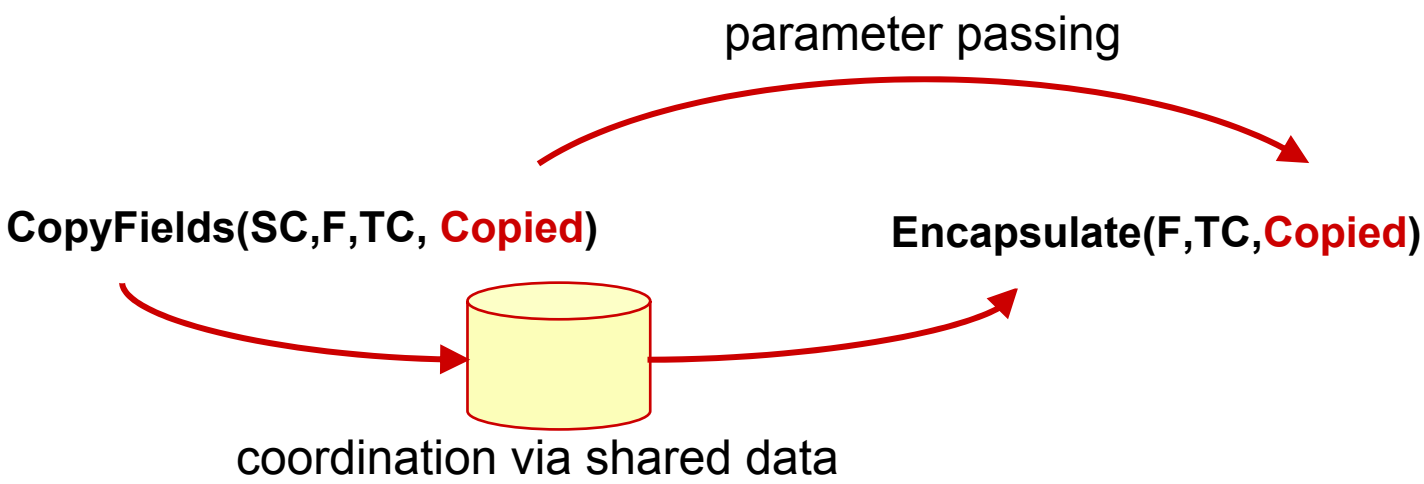
- Easy: Do whatever you need to
  - Add extra parameters for passing trace infos
  - Add trace information to the shared program database



- Problems
  - Redundancy, inconsistency, ...
  - Expensive
  - Sometimes not possible (third party code)

Reuse Scenarios: Anticipated Reuse

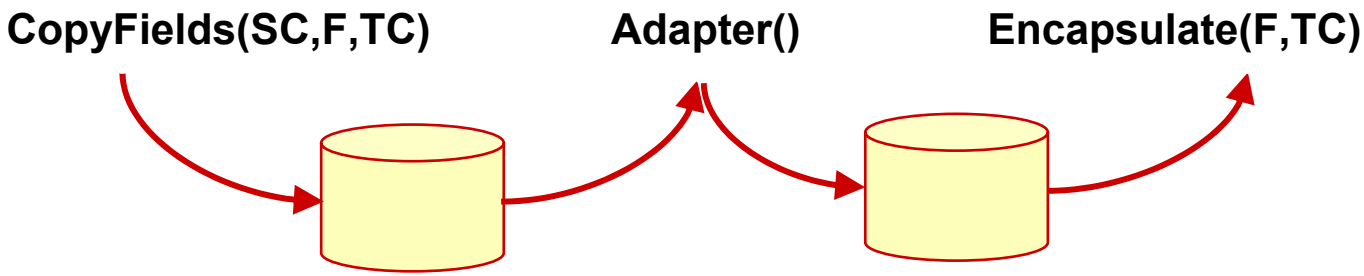
- Easy: Built-in hooks
  - Add extra parameters for passing trace infos
  - Add trace information to the shared program database



- Problems
  - We don‘t know the future
  - We cannot prepare for every possible future

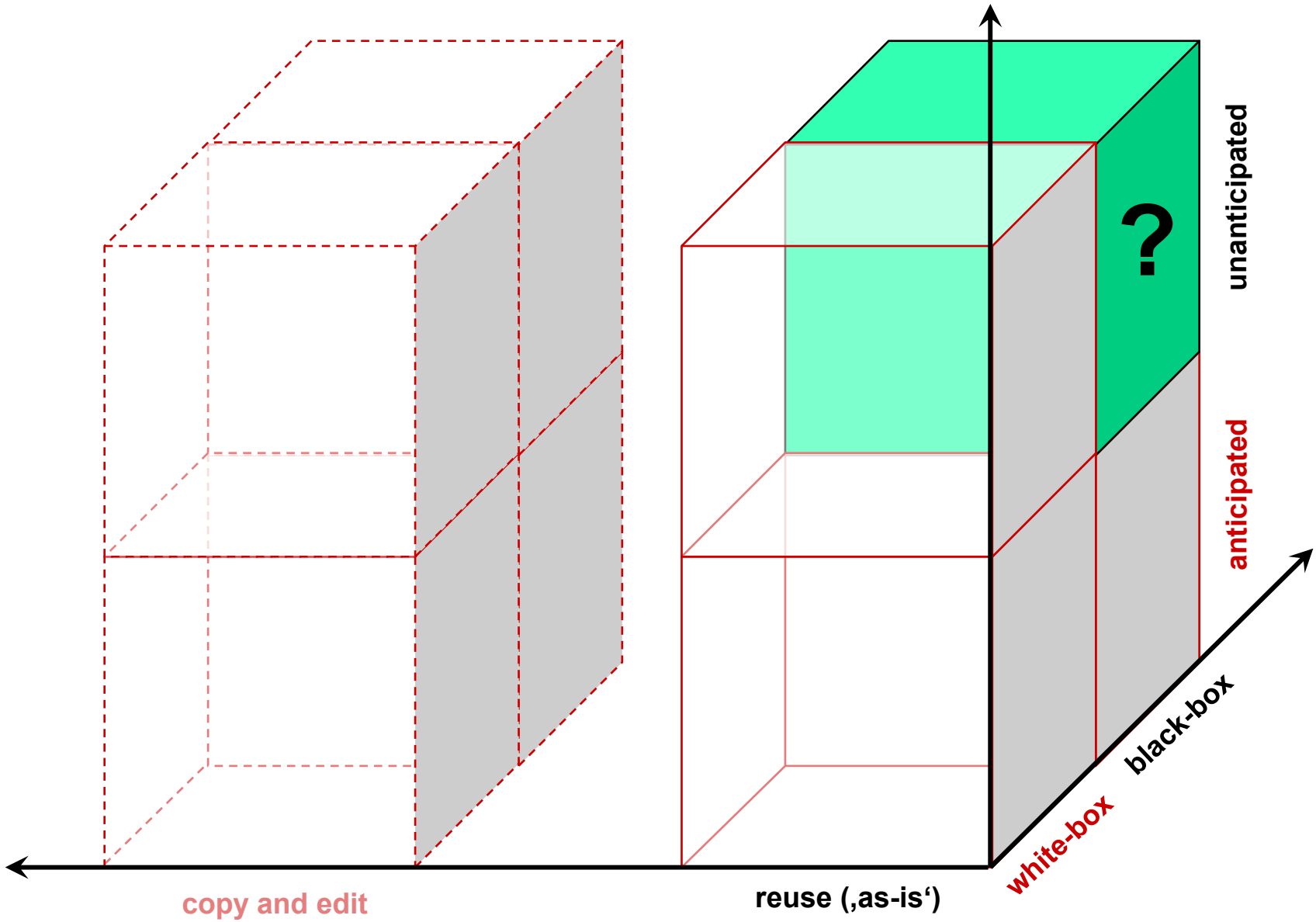
Reuse Scenarios: White-Box Reuse

- Add Glue Code
  - Use knowledge about internal details of participating transformations



- Problems
  - Dependency on internal details, evolution, maintainability, ...
  - Expensive (preferable not to write any glue code)

Reuse Dimensions and Categories



Conclusions

- Composed transformations often need „history information“
- MDA researchers identified this challenge but failed to provide a reuse-friendly solution
- Can we contribute a better one?

Discussion Group:
„Reusability of Transformation Modules“

„Reusability of Transformation Modules“

- Starting point
  - Need for history-aware composition operators for transformation modules
    - ➔ no, forward (&>) and bidirectional (<&>) propagation
  - Need to encapsulate history awareness into operators to achieve reusability

- Questions

1. Existing?

Is there any known formalization of these operators?

Are there any systems that do exactly this?

2. How to?

Are there existing concepts that can help formalize / implement these operators?

How?

3. Reuse?

What if we just use the concepts from 2. manually?

How reusable / composable will our programs be?

4. Wrap-up!

What‘s declared solved, what‘s still to do?

Discussion Group: „Reusability of Transformation Modules“

- Extend the categories
- Categorize know transformation systems

	Scenario		
	No propagation	Forward propagation	Bidirectional propagation
Destructive	[...]	[...]	[...]
Anticipated	[...]	[...]	[...]
White-box	[...]	[...]	[...]
Black-box	[...]	[...]	[...]