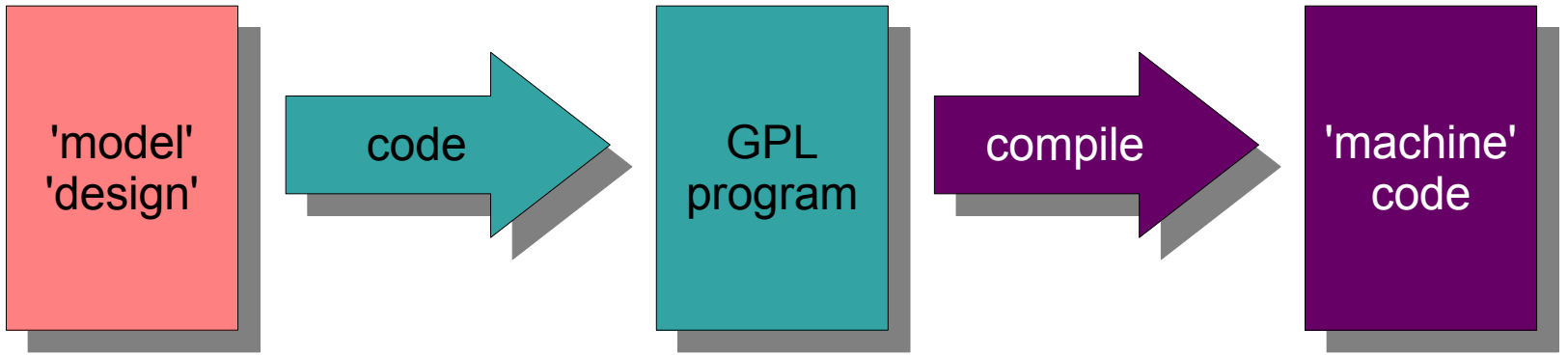
Model-Driven Software Evolution

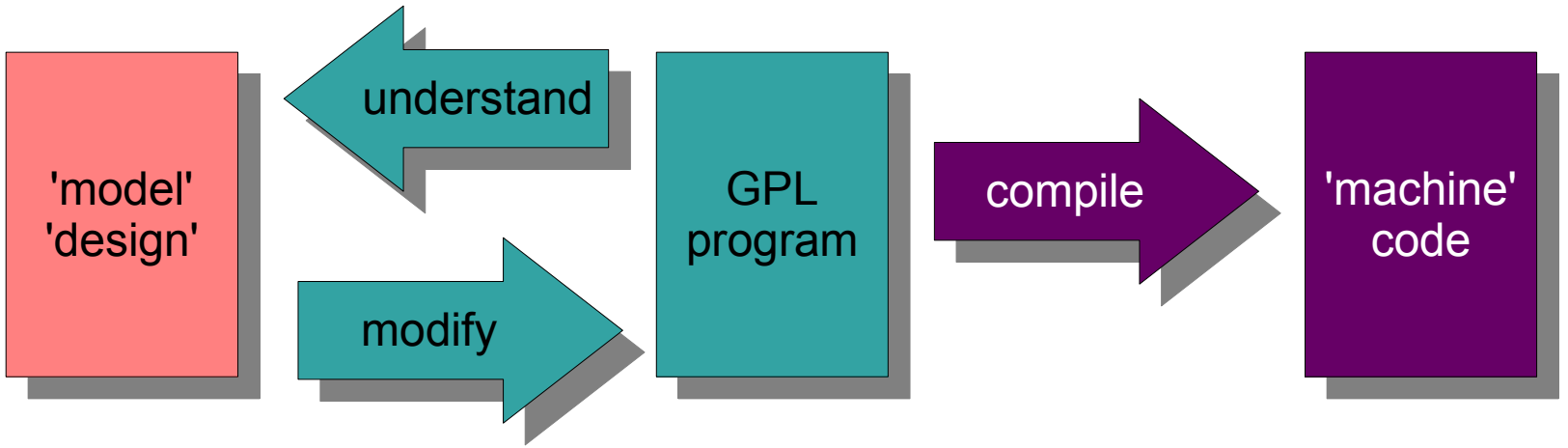
A Research Agenda

Arie van Deursen (TUD)
Eelco Visser (TUD)
Jos Warmer (Ordina)

conventional software development

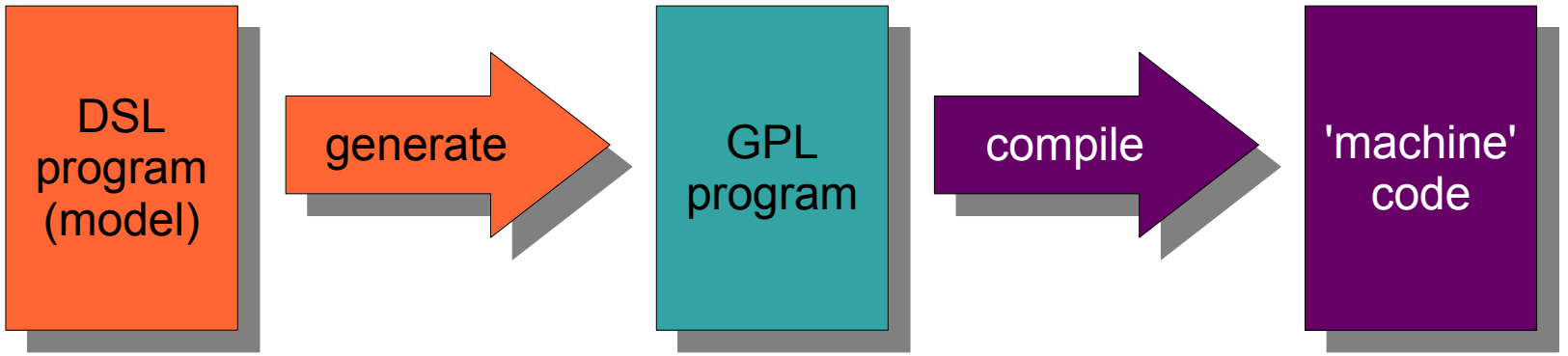


conventional software maintenance



abstractions encoded in program
maintenance at low level of abstraction

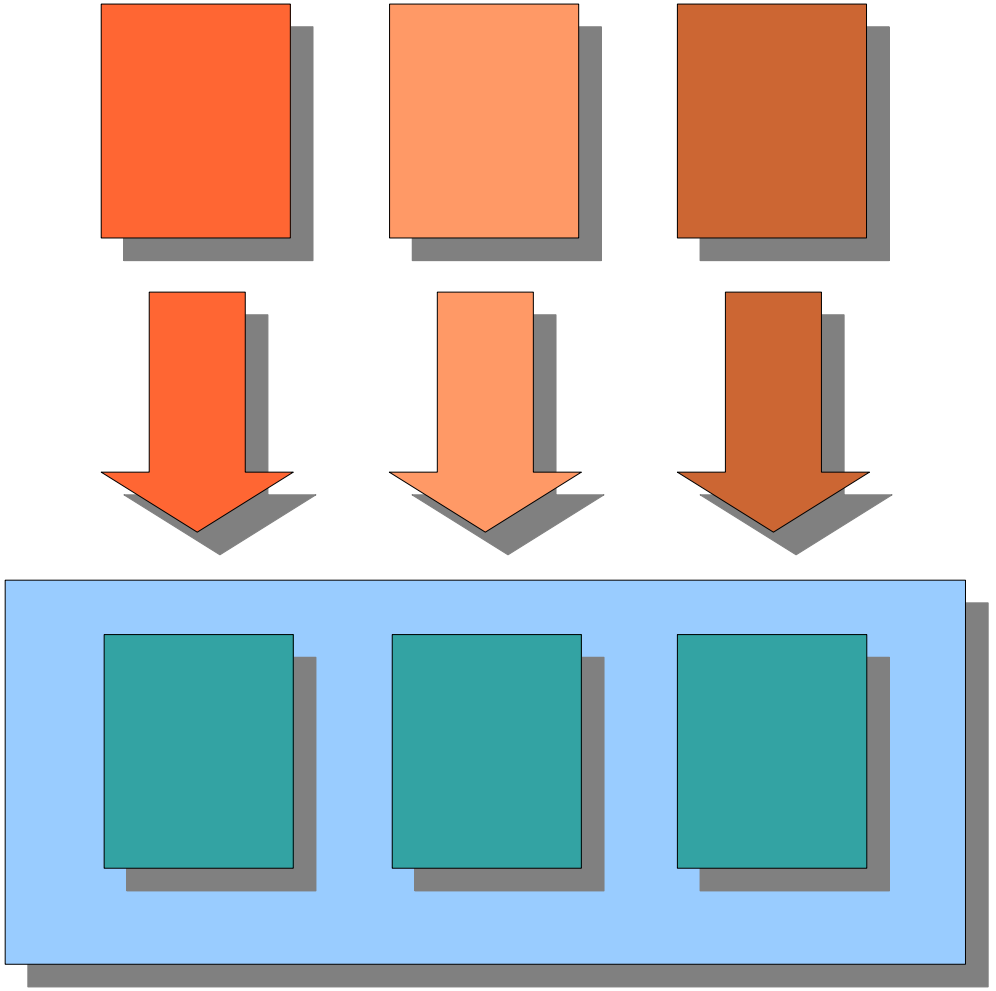
domain-specific languages
model-driven engineering



raise the level of abstraction to a technical or application domain
automatically generate implementation code from model

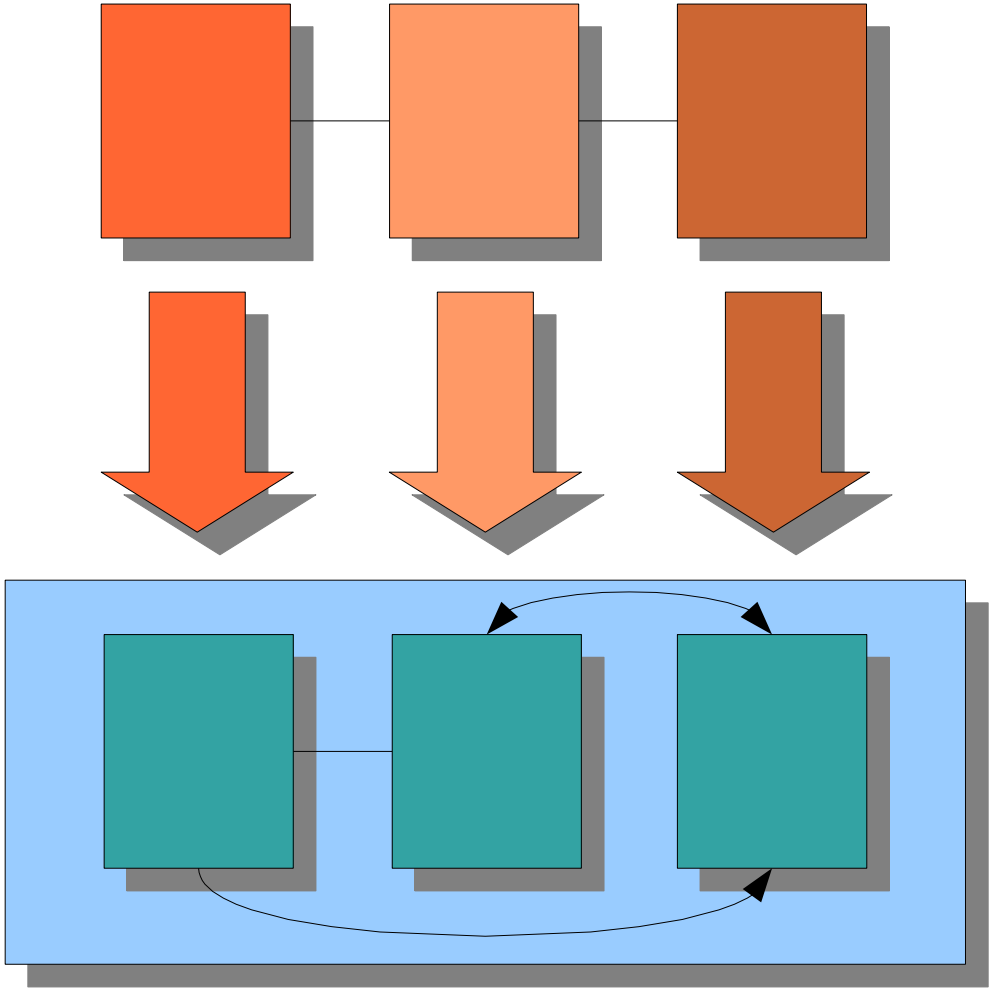
problem1: interaction

multiple models / multiple dsls



generate software from combinations of domain-specific languages

model/model interaction

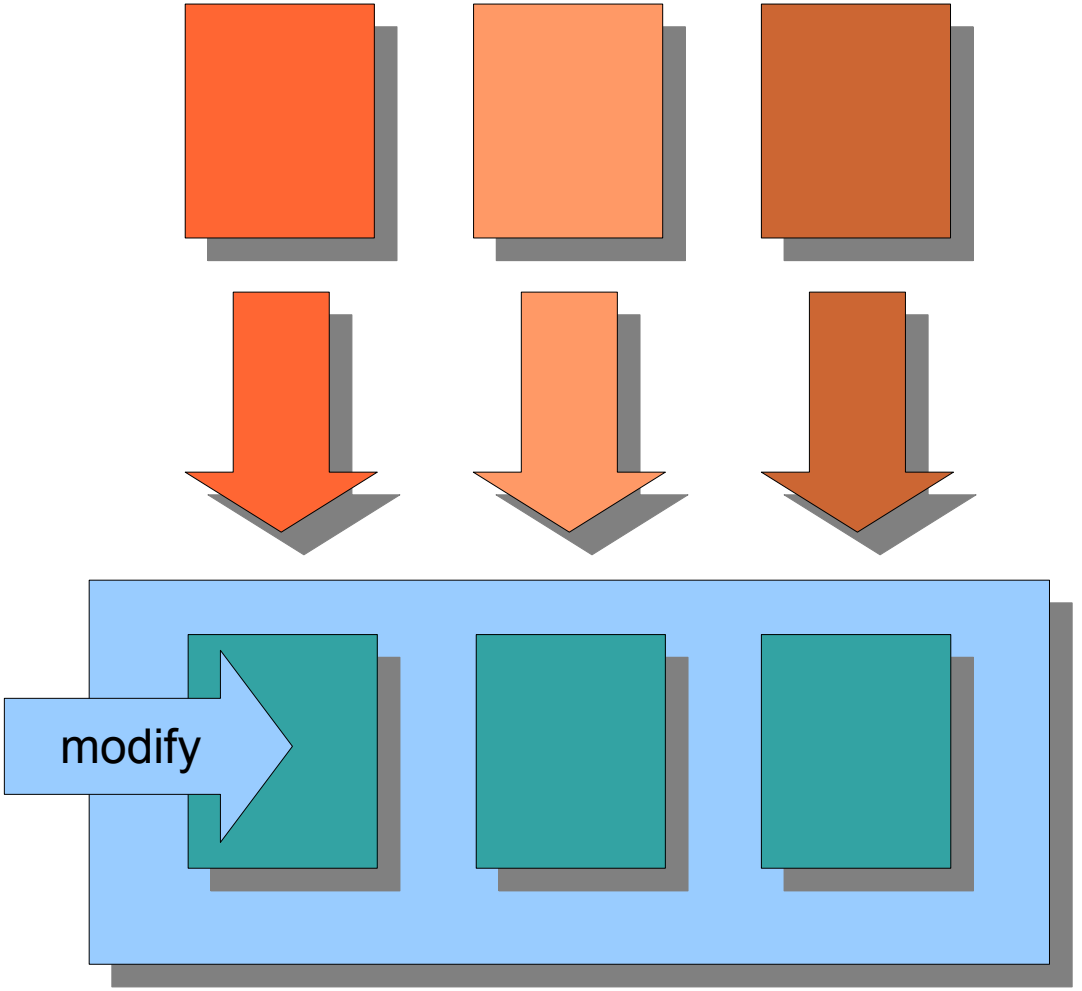


consider models as components / modules

what is interface of a model? what is the scope of model elements

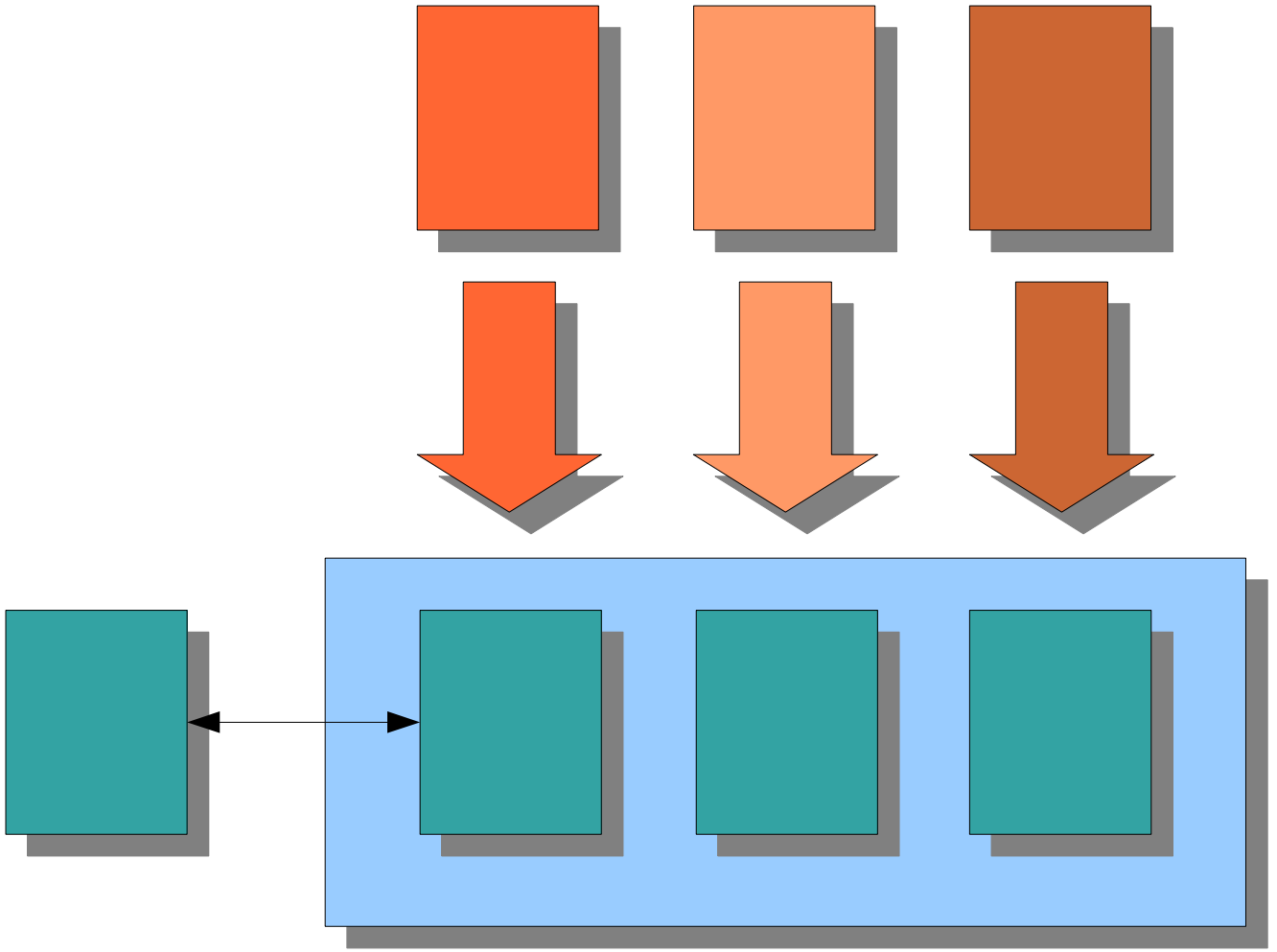
model encapsulation; separate compilation

customization of generated code



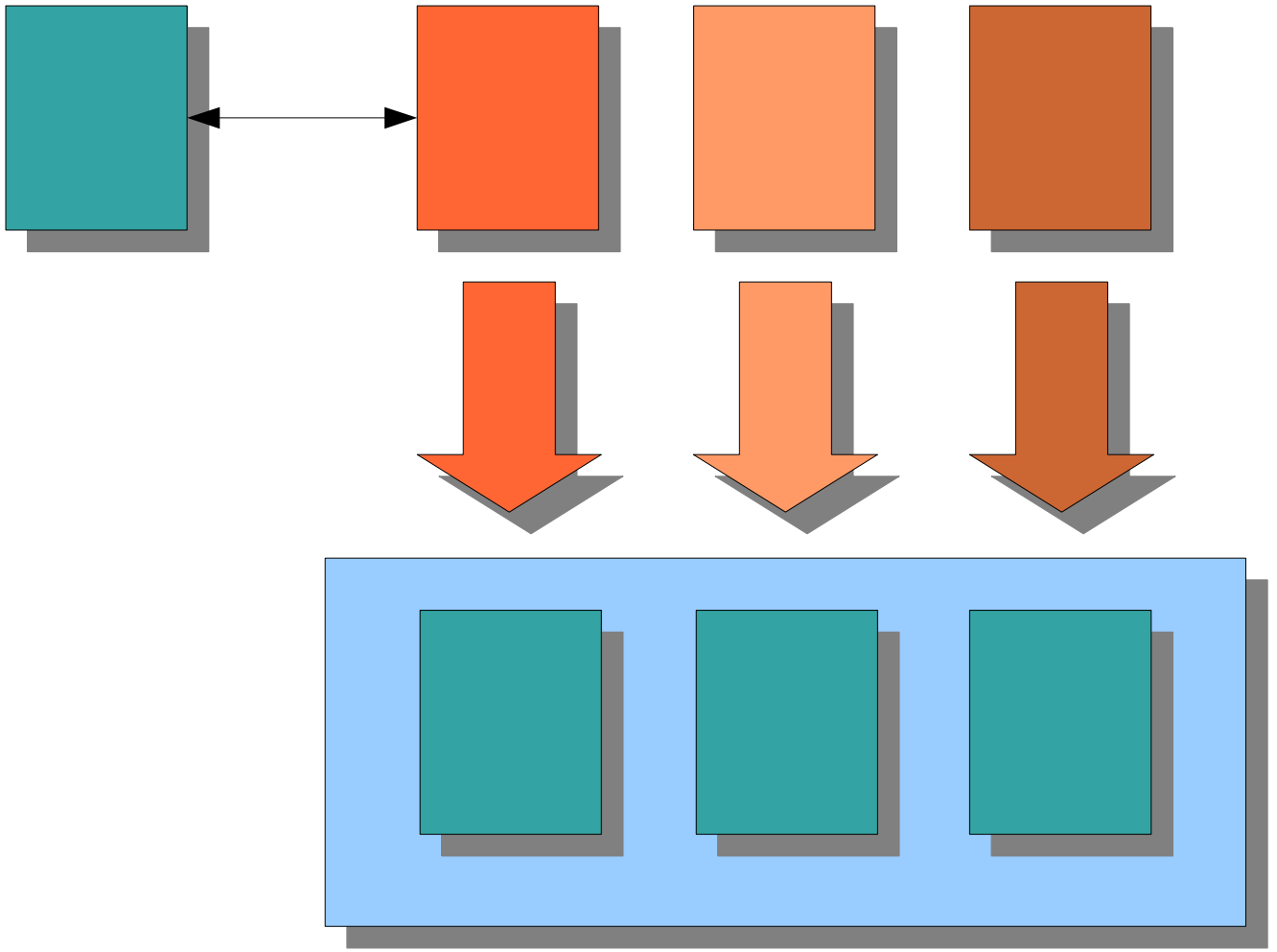
not all customizations can be realized in models
generated code may need to be adapted

customize 'from the outside'



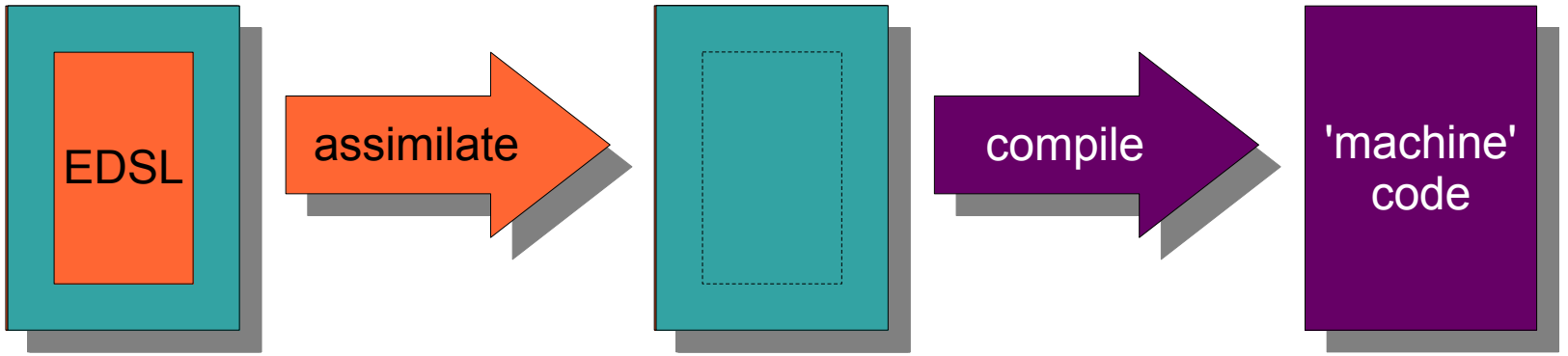
customization should **never** require direct modification of generated code
customization code must modify/interact with generated code
what is the interface? avoid exposing generation scheme

model/code interaction



customization code should be considered as part of the generator input
should interact with (interface of) models, not with generated code

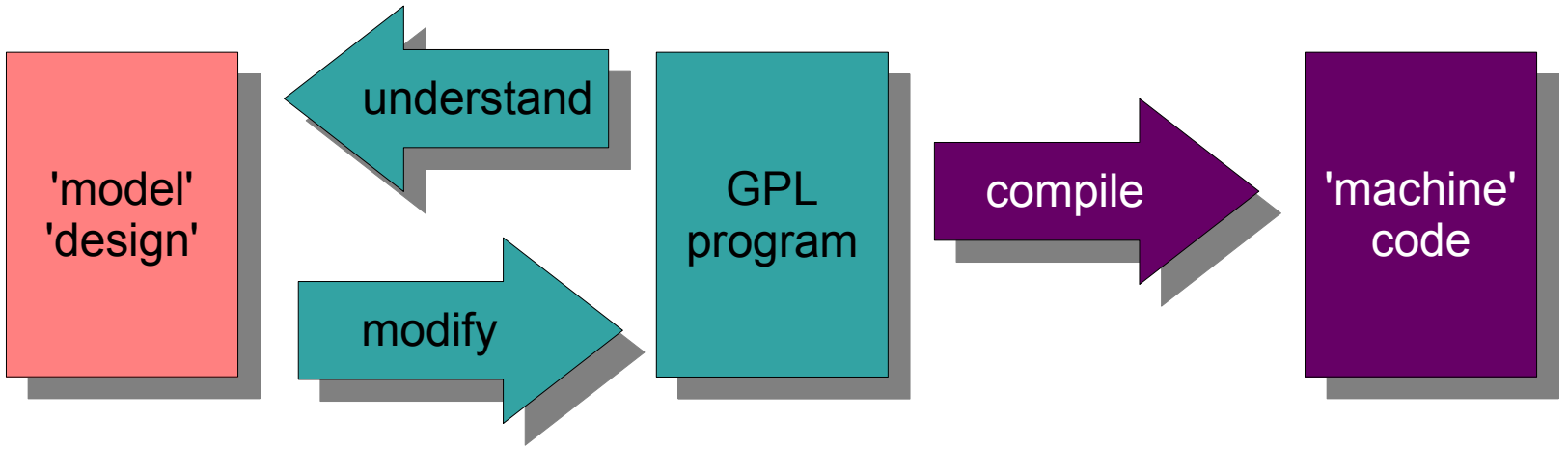
embedded domain-specific languages



MetaBorg (OOPSLA'04)
DSLs for abstraction over libraries/frameworks
fine-grained interaction with 'host' code
language conglomerates mix DSL and GPL code

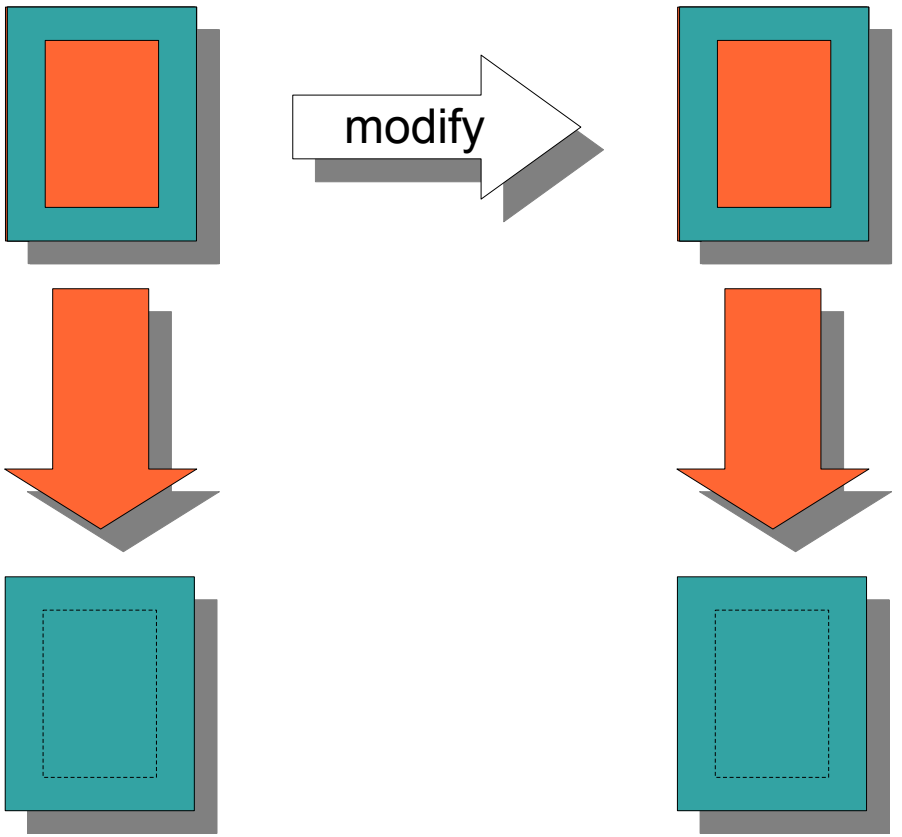
problem 2: evolution

dimensions of evolution



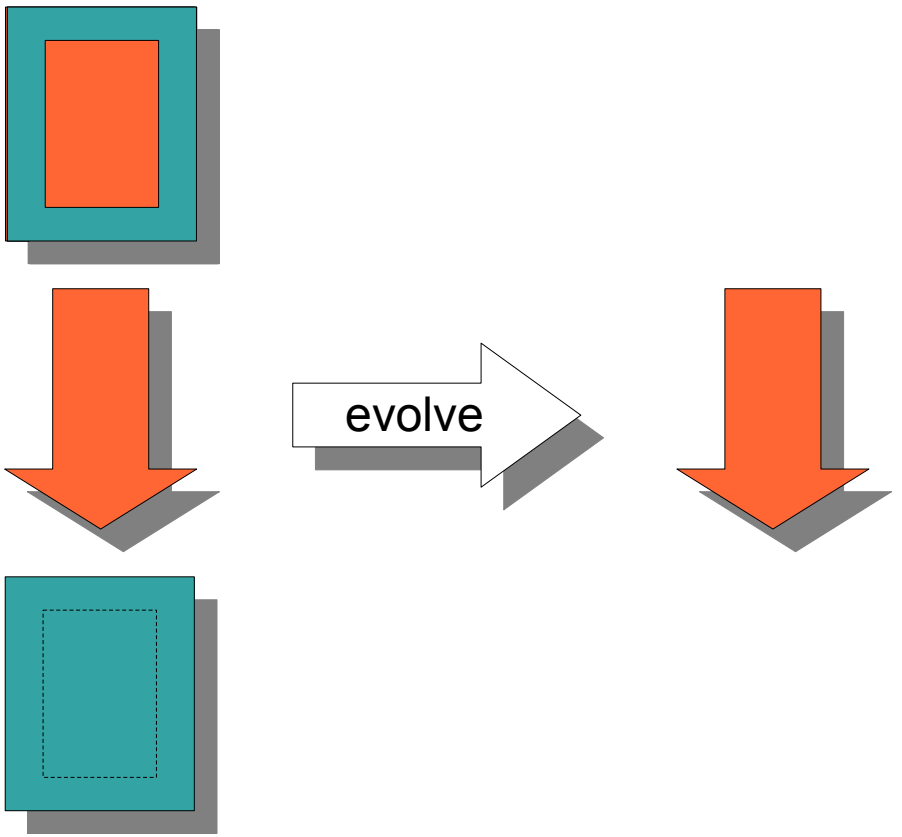
traditional evolution is one-dimensional
(only one artifact (gpl code) to maintain)

regular evolution



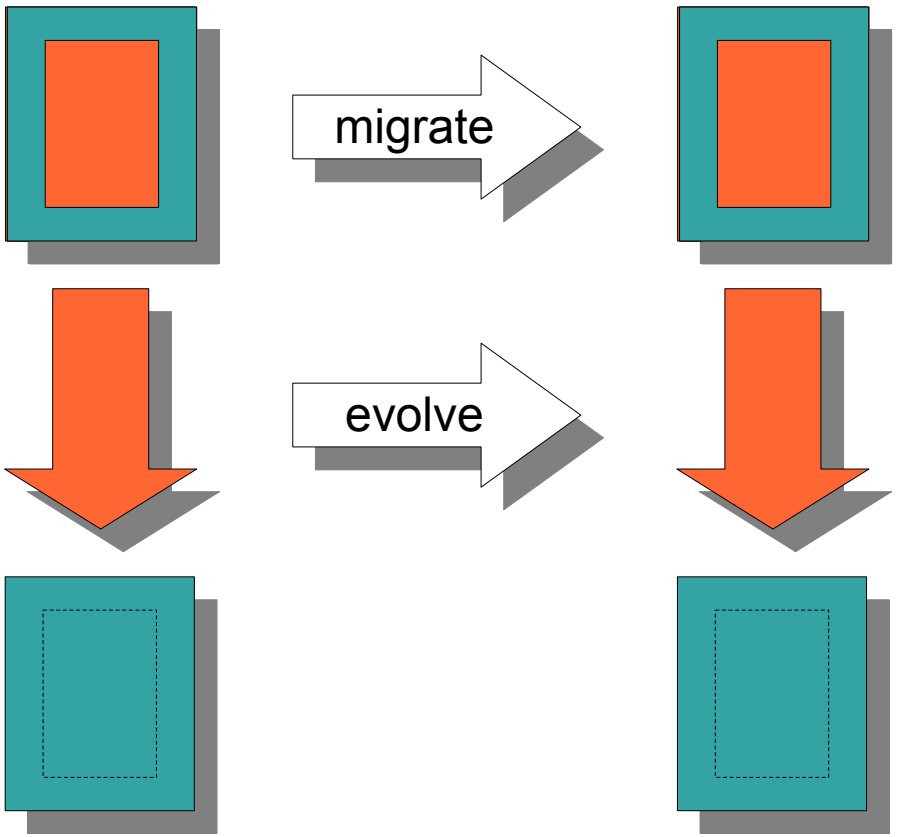
regular evolution: adapt software to new requirements
implementation simply regenerated after modification of models

meta-model evolution



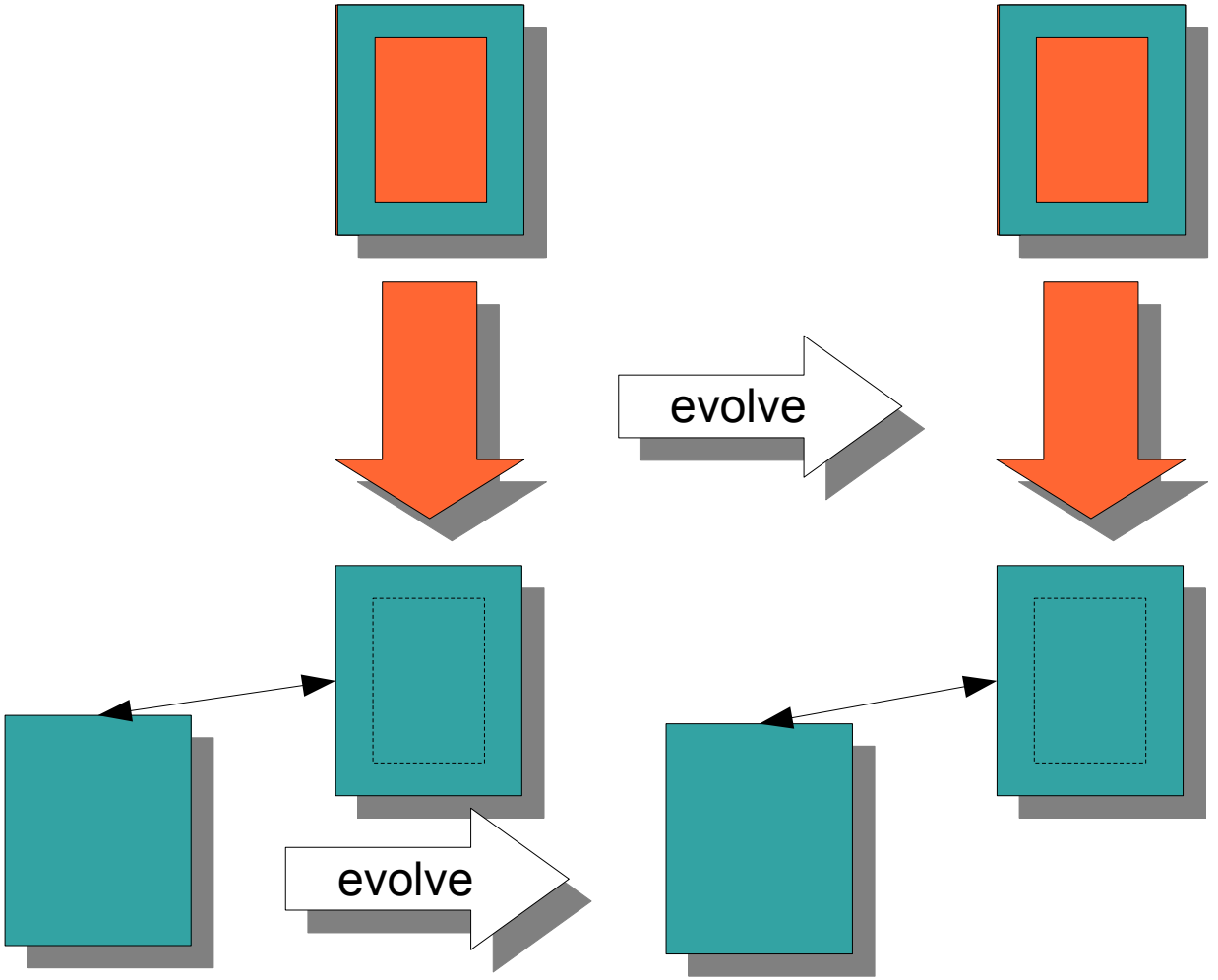
language (syntax and/or transformations) evolve

model migration



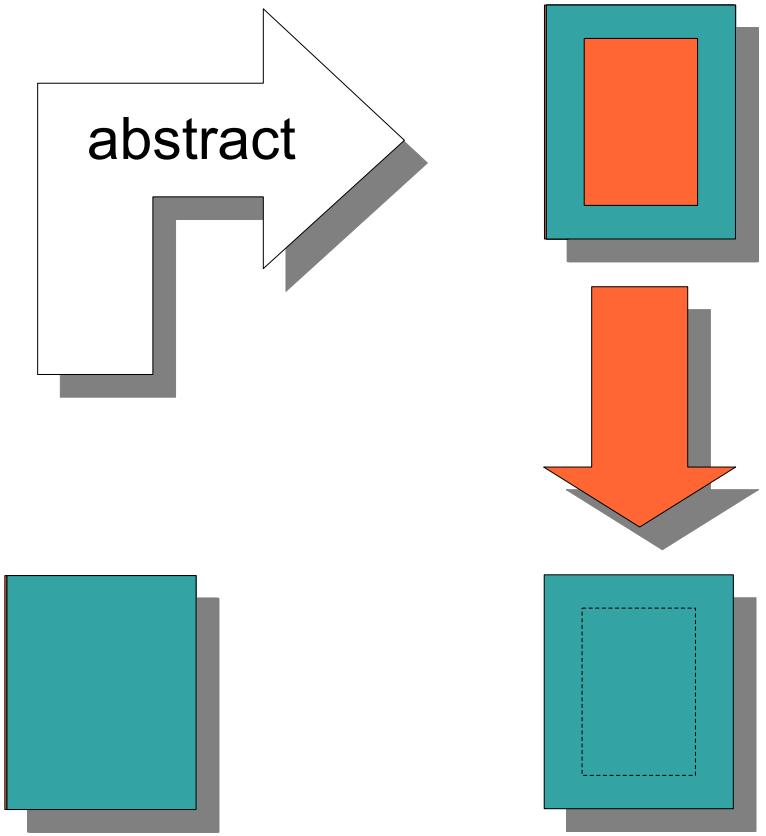
language evolution requires migration of models

platform evolution



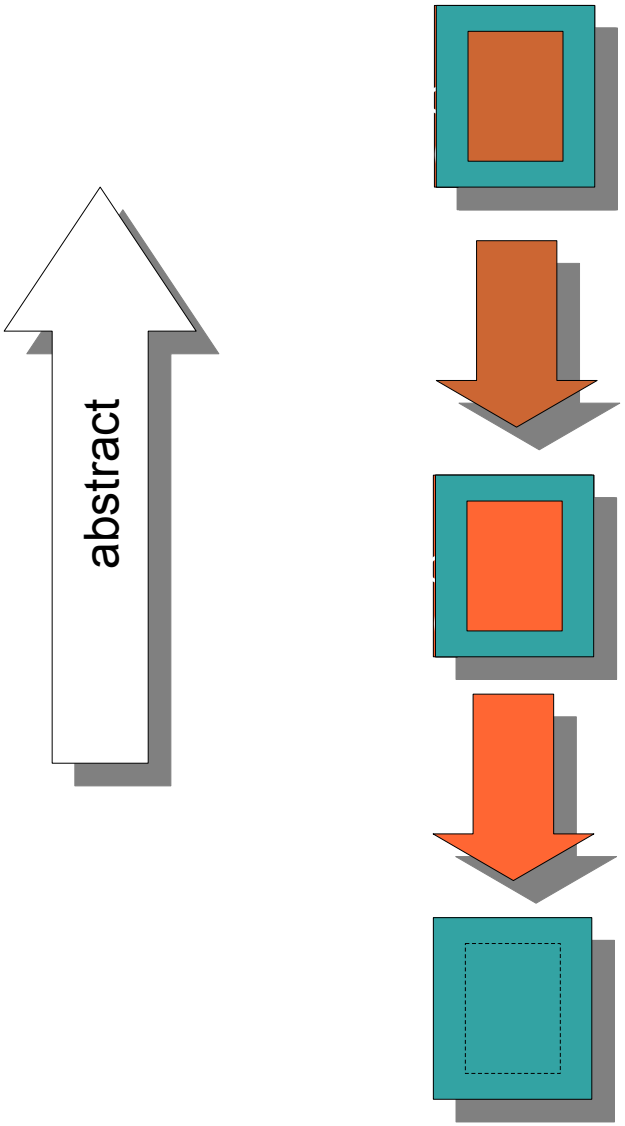
changes in the platform requires evolution of transformations
maintain generators for multiple platforms

model extraction



derive models from (legacy) GPL programs

abstraction evolution



develop higher-level abstractions

themes for a research agenda

- technology
  - model development environment
- generation
  - from model to code
- evolution
  - from code to model
- evaluation
  - how

model development environment

- connecting technological spaces
  - modelware (uml), grammarware (sdf), xmlware, ...
  - grammars for language combinations
- unifying model and code transformation
  - model extraction from code
  - code generation
- language definitions in development env.
  - making a new dsl should be as easy as making a new class

generation – from model to code

- modeling business logic
  - scope and expressivity of DSLs
  - balance between generality and dom. specificity
- model interaction
  - separation of concerns -> dependencies
  - modularity: encapsulation, interfaces
  - how to refer to elements in other languages?
- model composition
  - composition of whole systems from models

evolution – from code to model

- incremental model introduction
  - migrate part of legacy code base to models
  - models and code co-exist
- model reconstruction
  - harvest models from existing (legacy) code
  - agnostic: search for recurring patterns
  - reconstruct models for known DSLs
- model-based testing
  - validation of migration to models

evaluation

- risk/benefit analysis
  - return on investment: when does effort of dsl design and implementation pay off?
  - goal of MDE is to lower the treshhold
  - factors for success and counter indicators
- methodological embedding
  - decision making process for adopting MDE
  - guidelines based on case studies and literature

our contribution

- funding for several research projects
  - model-driven software evolution (MoDSE)
    - 2 phd students, 2 postdocs (we are still hiring!)
    - NWO/JACQUARD program (software engineering)
  - transformations for abstractions (TFA)
    - 1 postdoc
    - how to deal with combinations of languages
  - single page computer interaction (SPCI)
    - 1 phd student
    - reverse engineering & modeling rich user interfaces
- in collaboration with industrial partners