Generating Models using Metaheuristic Search

James R. Williams, Simon Poulding
University of York, UK

James Williams

An apology.

MDE and SBSE

Search Based Model Generation

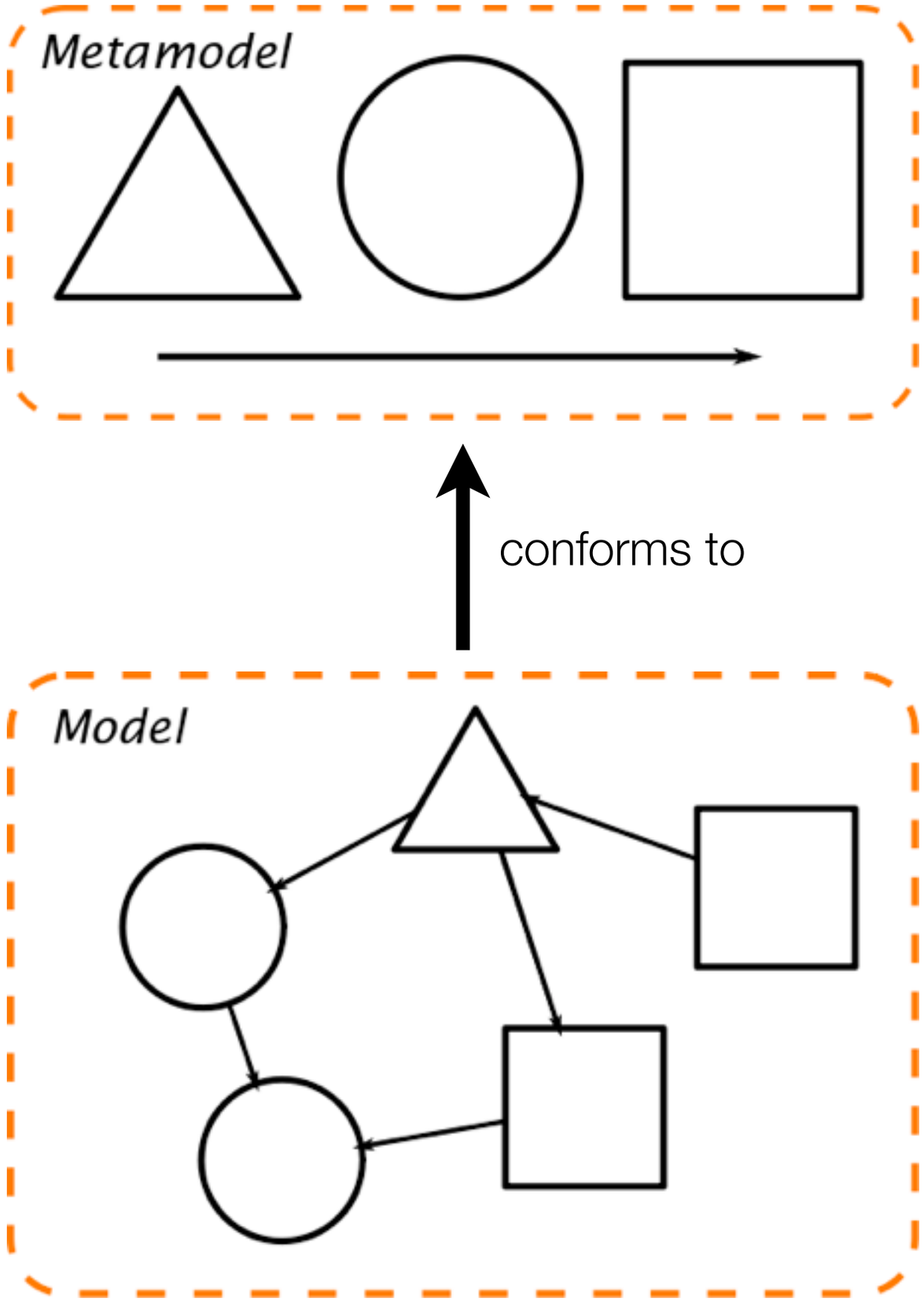
Future Work

The Problem

To automatically derive effective models that test MDE processes

Model Driven Engineering

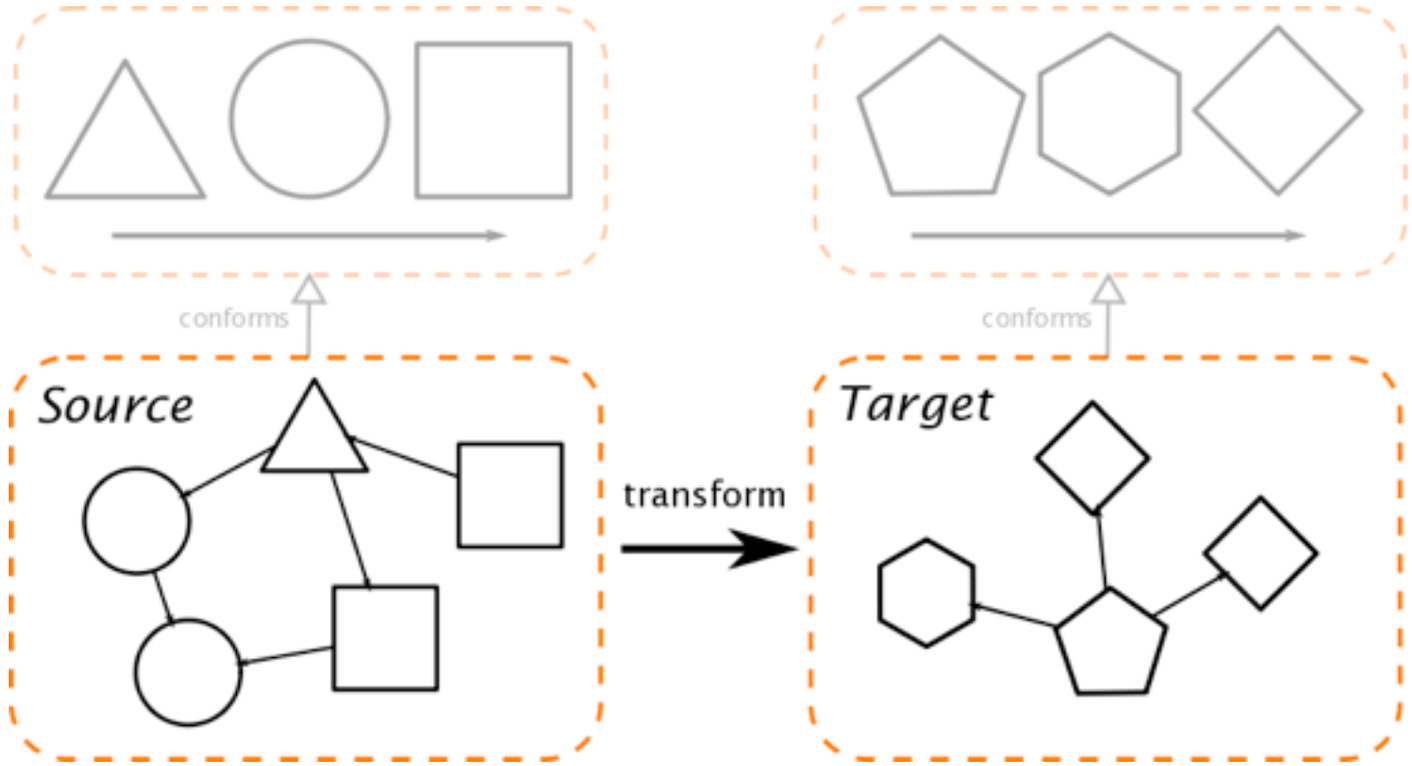
Treats models as first-class artefacts in the software development lifecycle.



Model Management

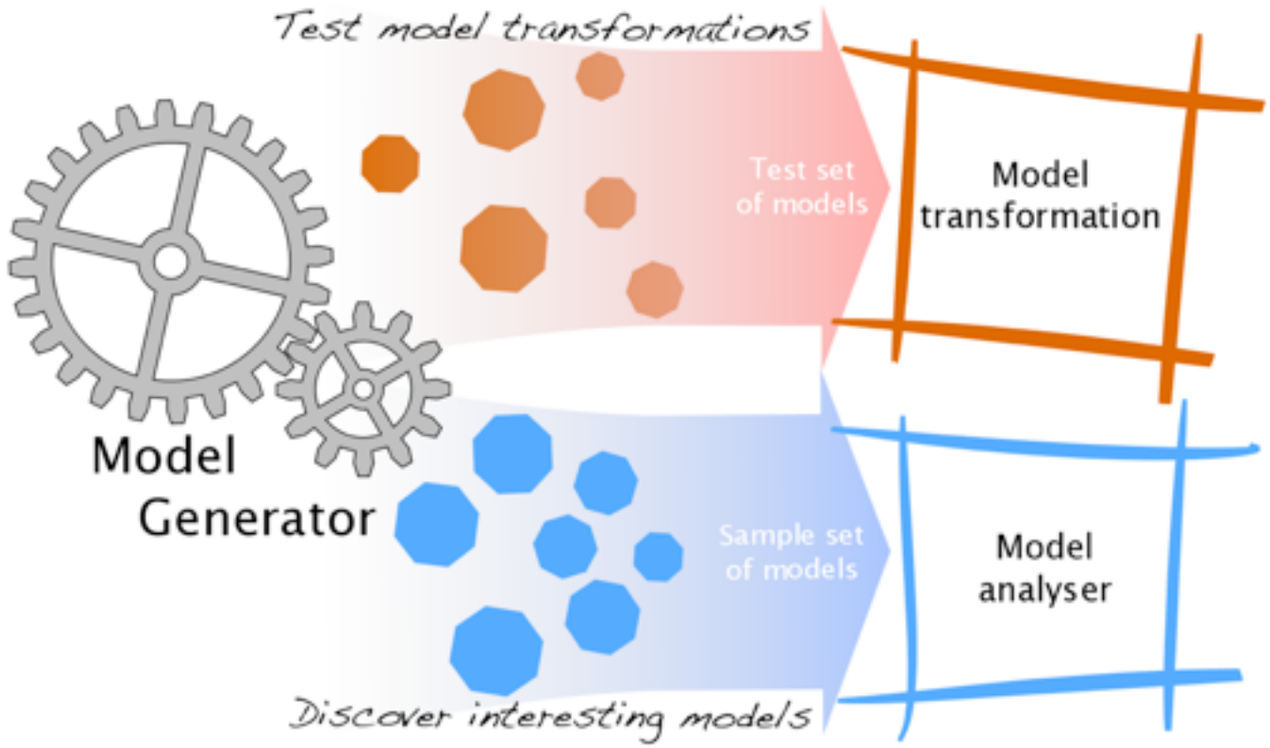
Transformation
Migration
Merging
Comparison
Validation
Refactoring

Heterogeneous
model-to-model transformation



Model Generation

To test model management tasks, we execute the tasks on models.

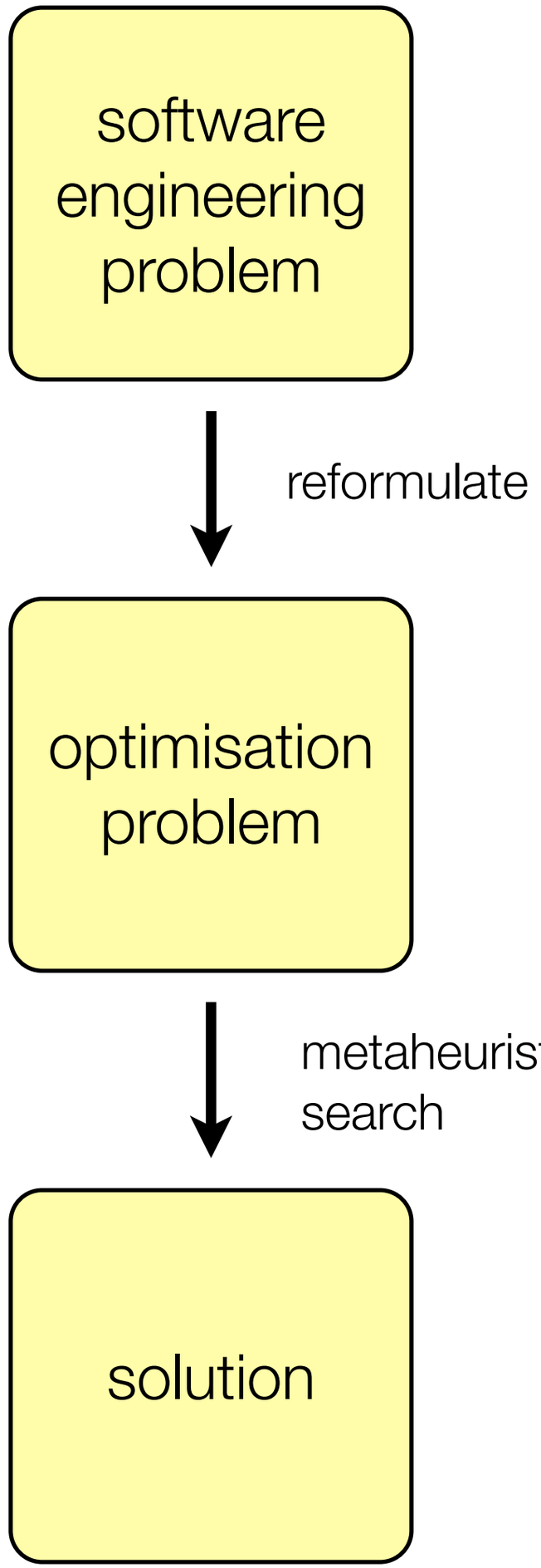


Proposed Approach

Search-Based Software Engineering

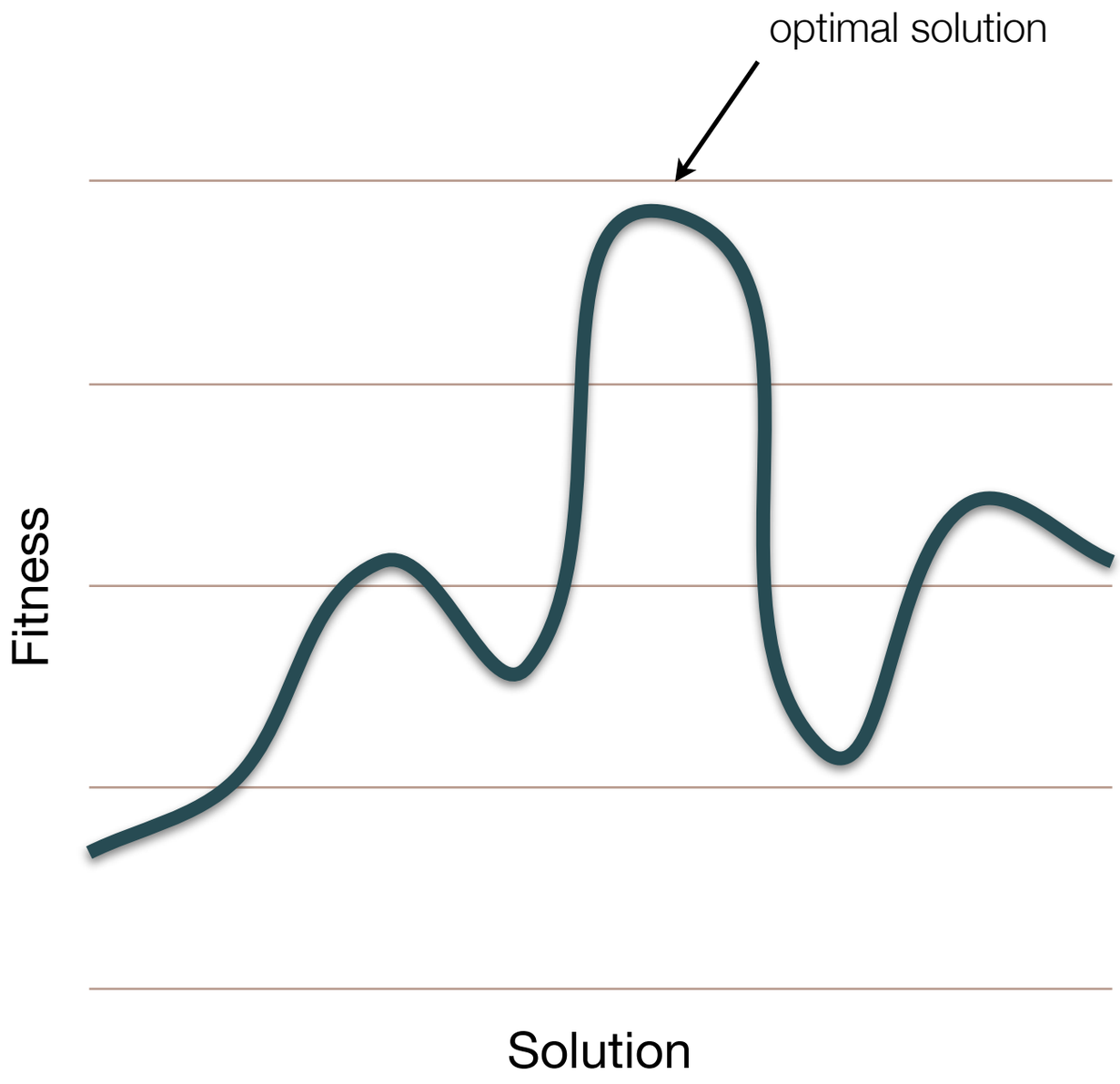
Search-Based Software Engineering

Consider engineering task as an optimisation problem, and apply metaheuristic search algorithm to solve it.



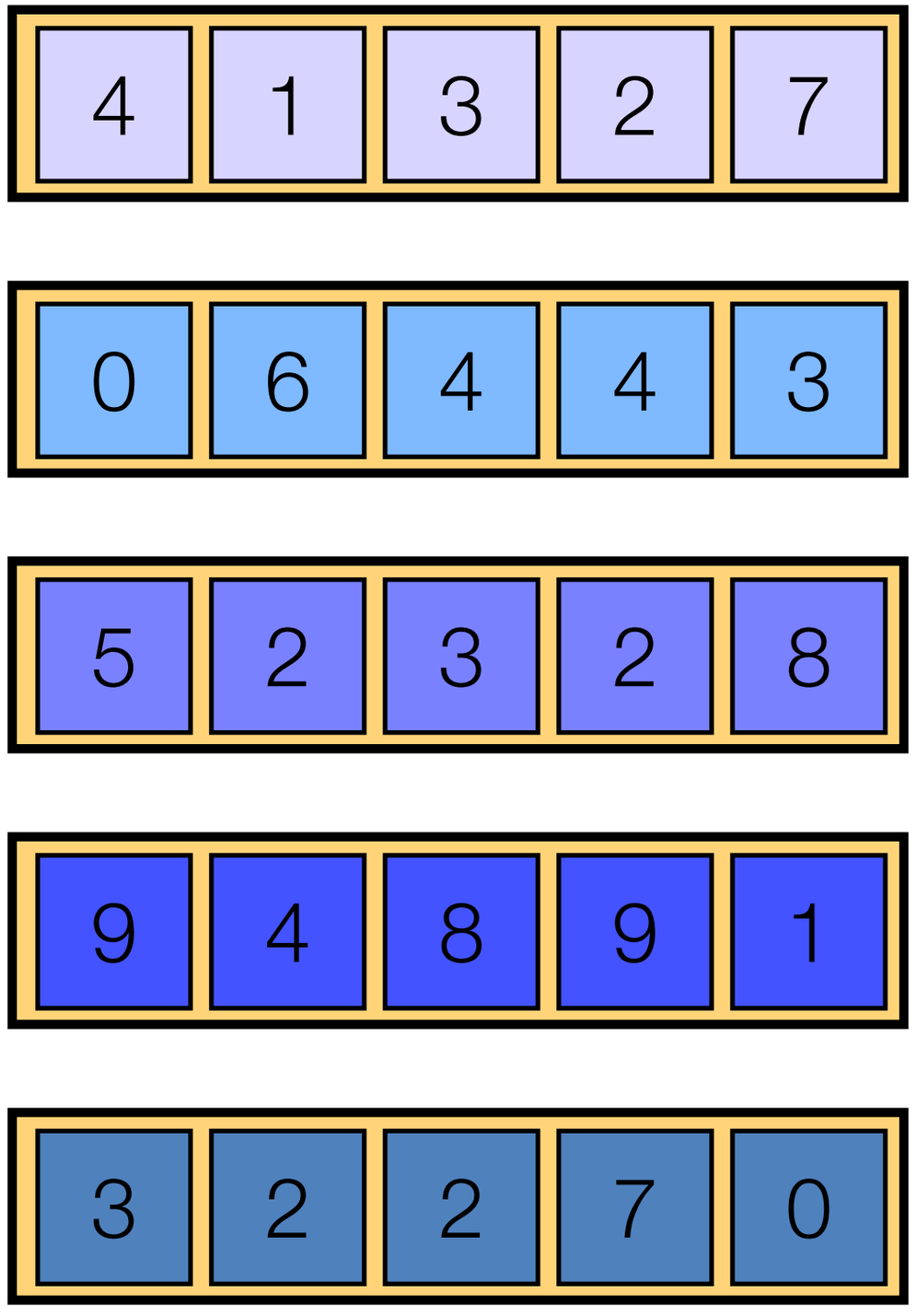
Metaheuristic Search

"it is easier to determine whether one solution is better than another, than to construct an optimal solution"



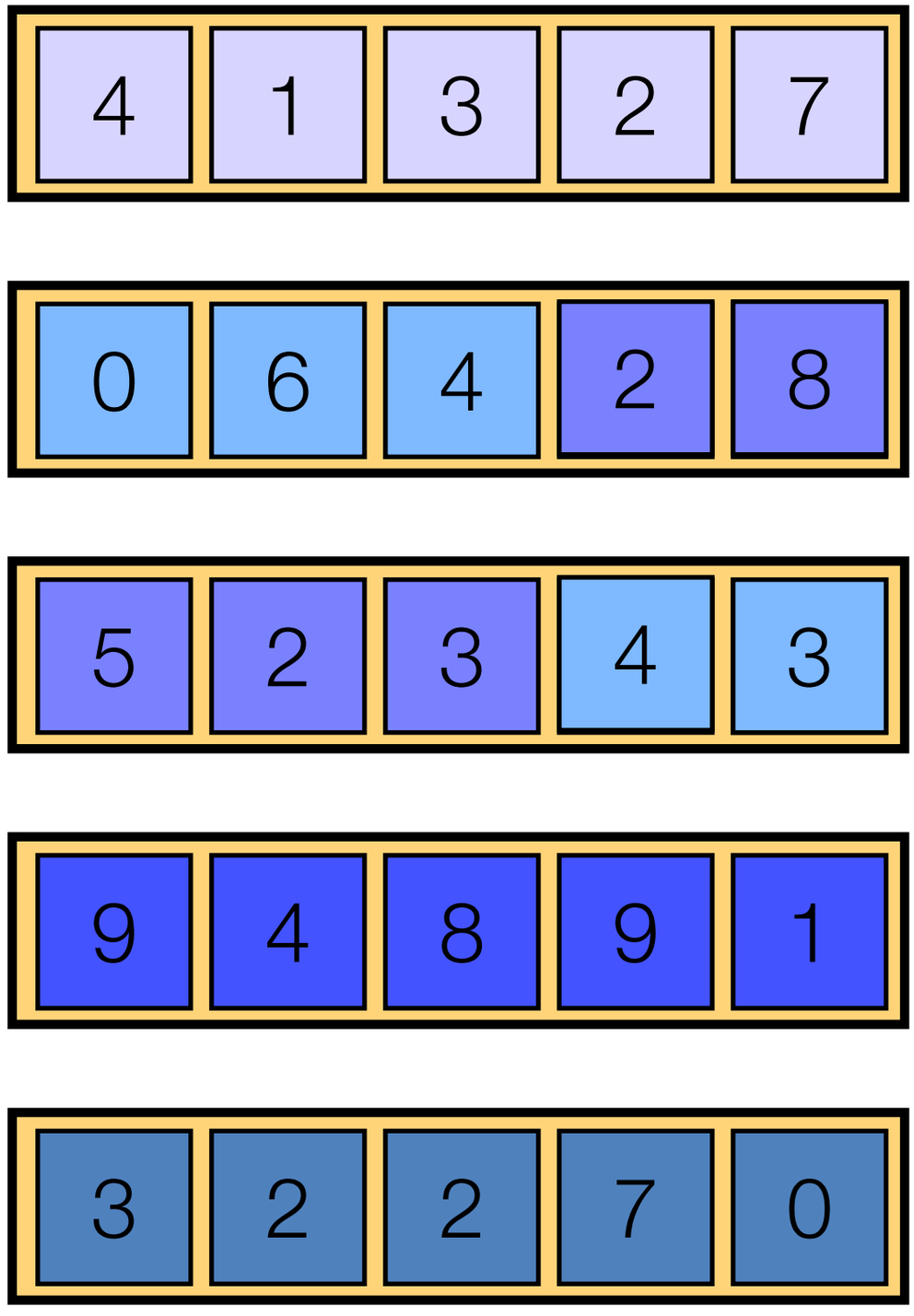
Genetic Algorithm

Inspired by basic principles of natural selection; applied to population of solutions.



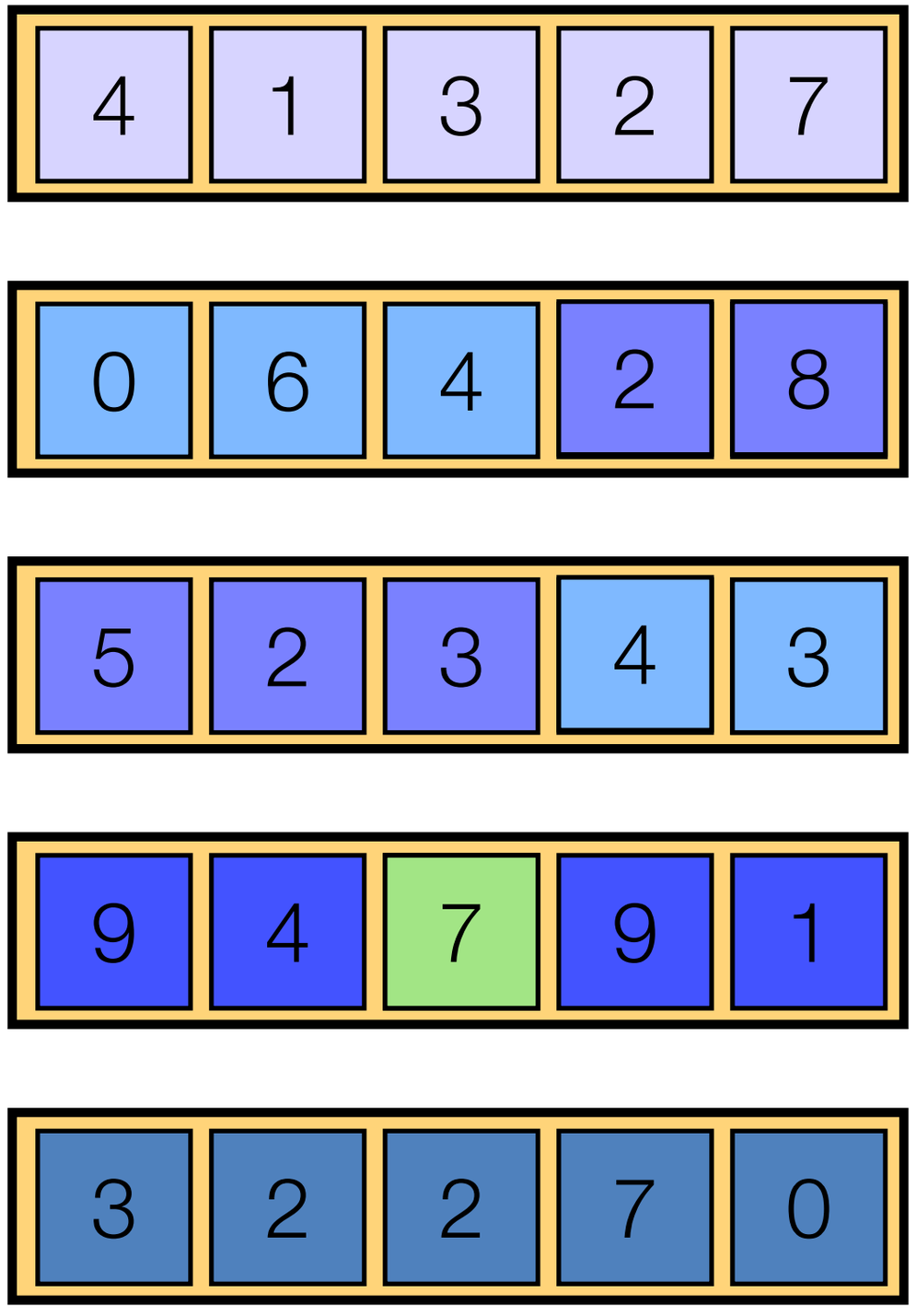
Genetic Algorithm

Inspired by basic principles of natural selection; applied to population of solutions.



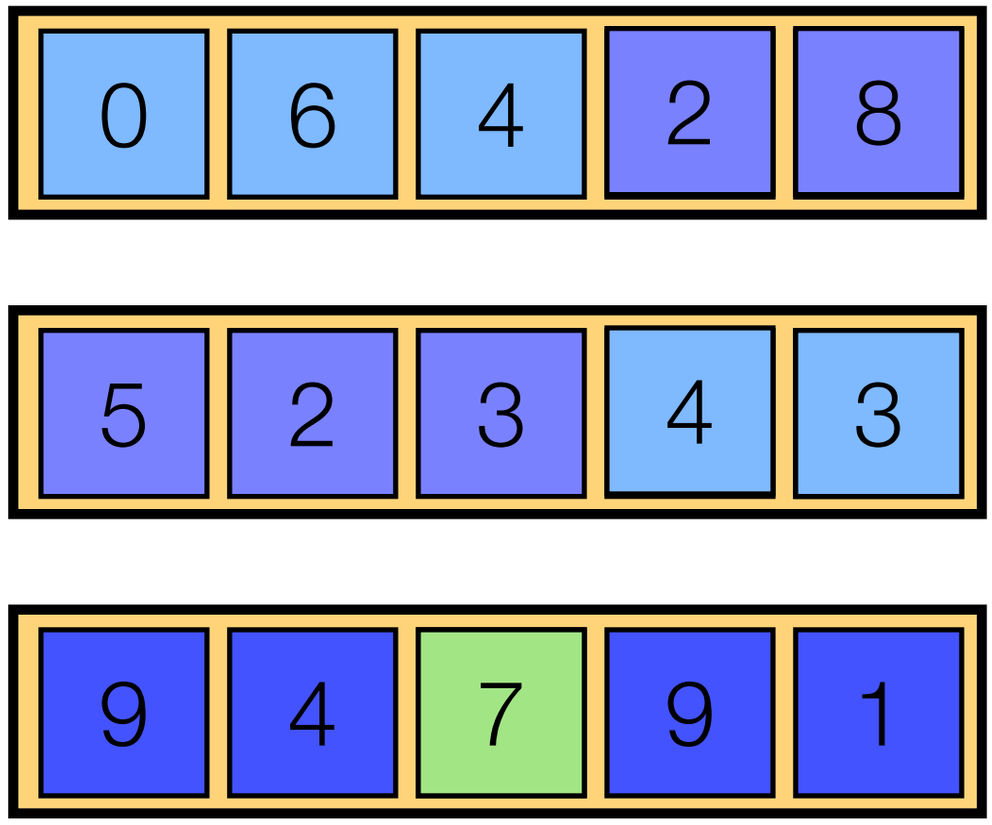
Genetic Algorithm

Inspired by basic principles of natural selection; applied to population of solutions.



Genetic Algorithm

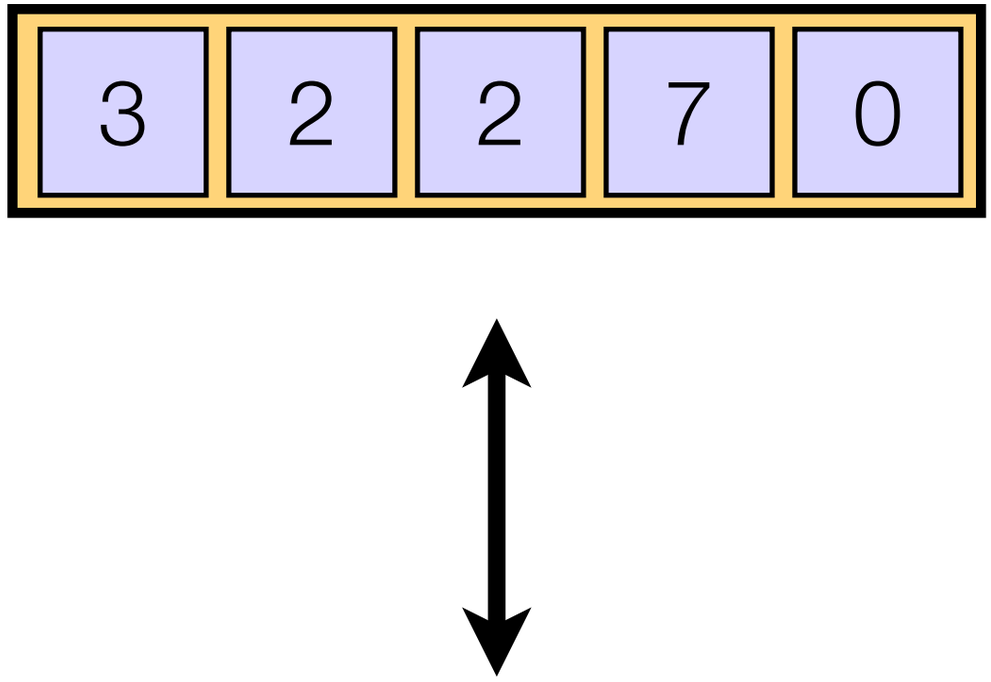
Inspired by basic principles of natural selection; applied to population of solutions.



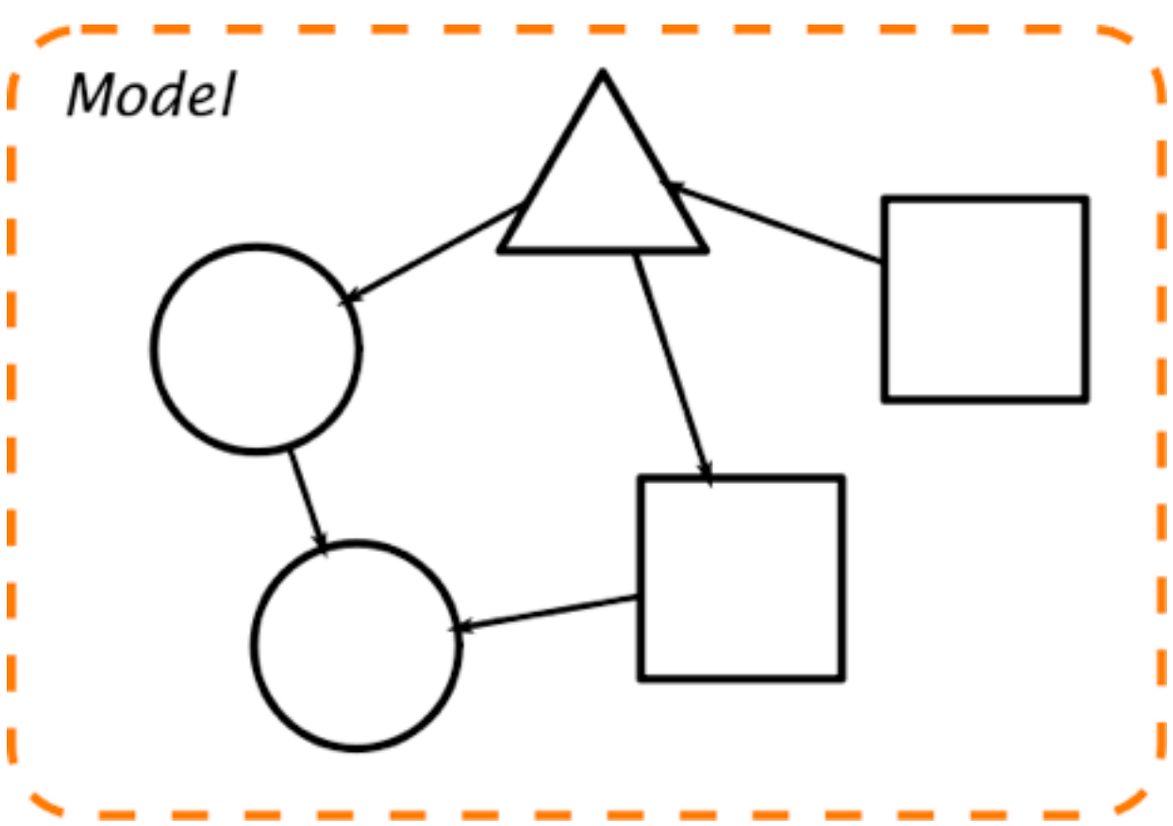
Representation

An abstract genotype is mapped to a phenotype specific to the problem

genotype

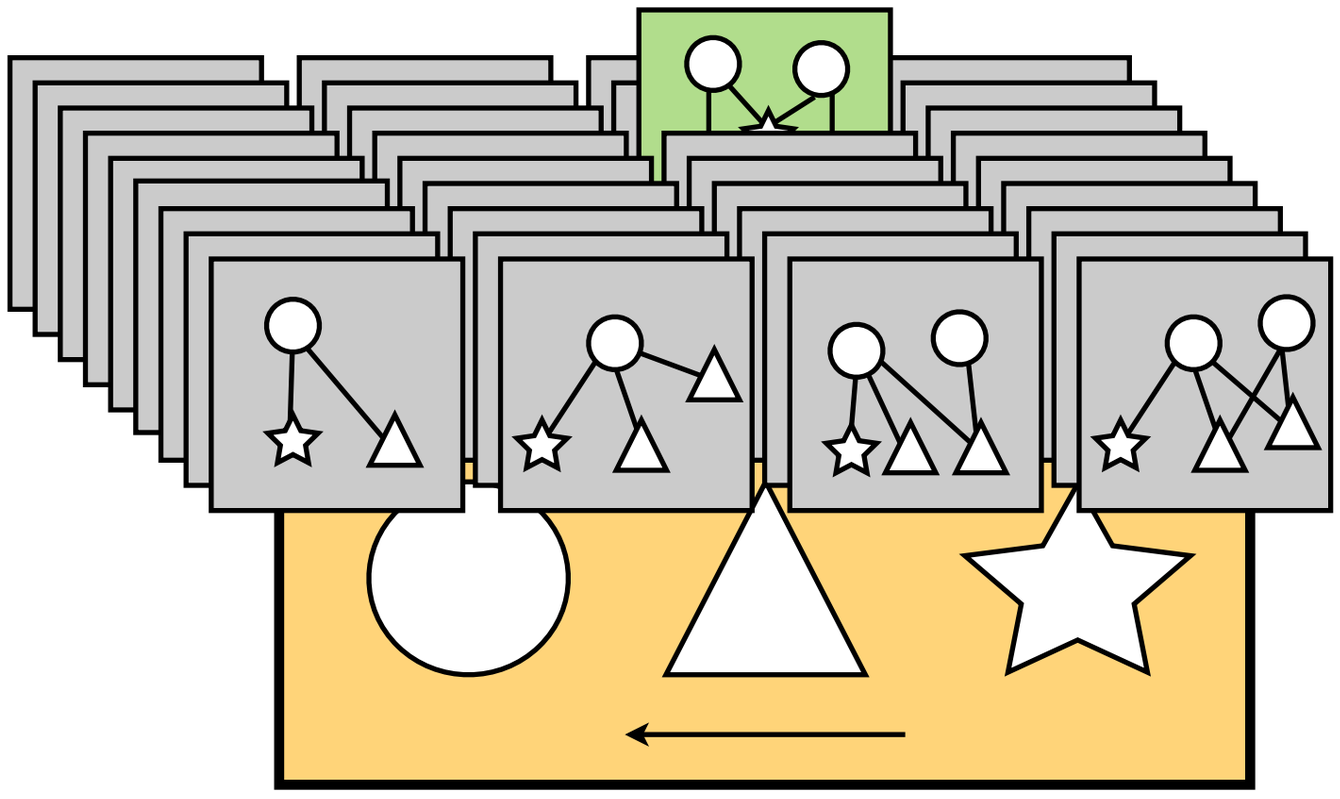


phenotype



MDE + Search

Metamodels define a space of valid models over which we can search.



MDE and SBSE

Search Based Model Generation

Future Work

Previously ...

Williams, Poulding, Rose, Paige, and Polack
Identifying Desirable Game Character Behaviours through the Application of Evolutionary Algorithms to Model-Driven Engineering Metamodels
International Symposium on Search Based Software Engineering
2011

Super Awesome Fighter 4000

Human player FDL

```
fighter human {
  punchPower=7
  kickReach=4
  near[crouch punch_low]
  always[walk_away kick_high]
}
```

Super Awesome Fighter 4000

versus

Non-player FDL

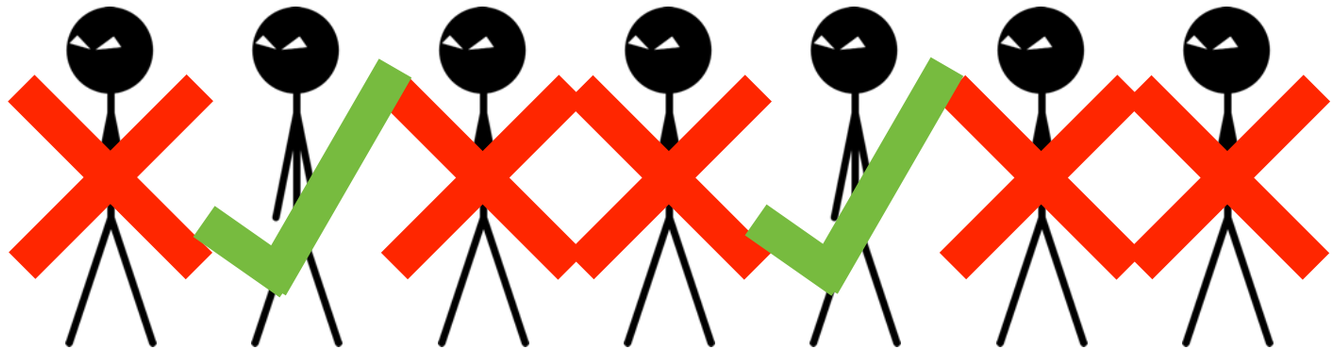
```
fighter nonPlayer {
  punchReach=8
  near[jump kick_high]
  far[run_towards block_low]
  always[run_away block_high]
}
```

Engineering Objective

Searching for non-player fighters with desirable properties.

Non-player Fighter

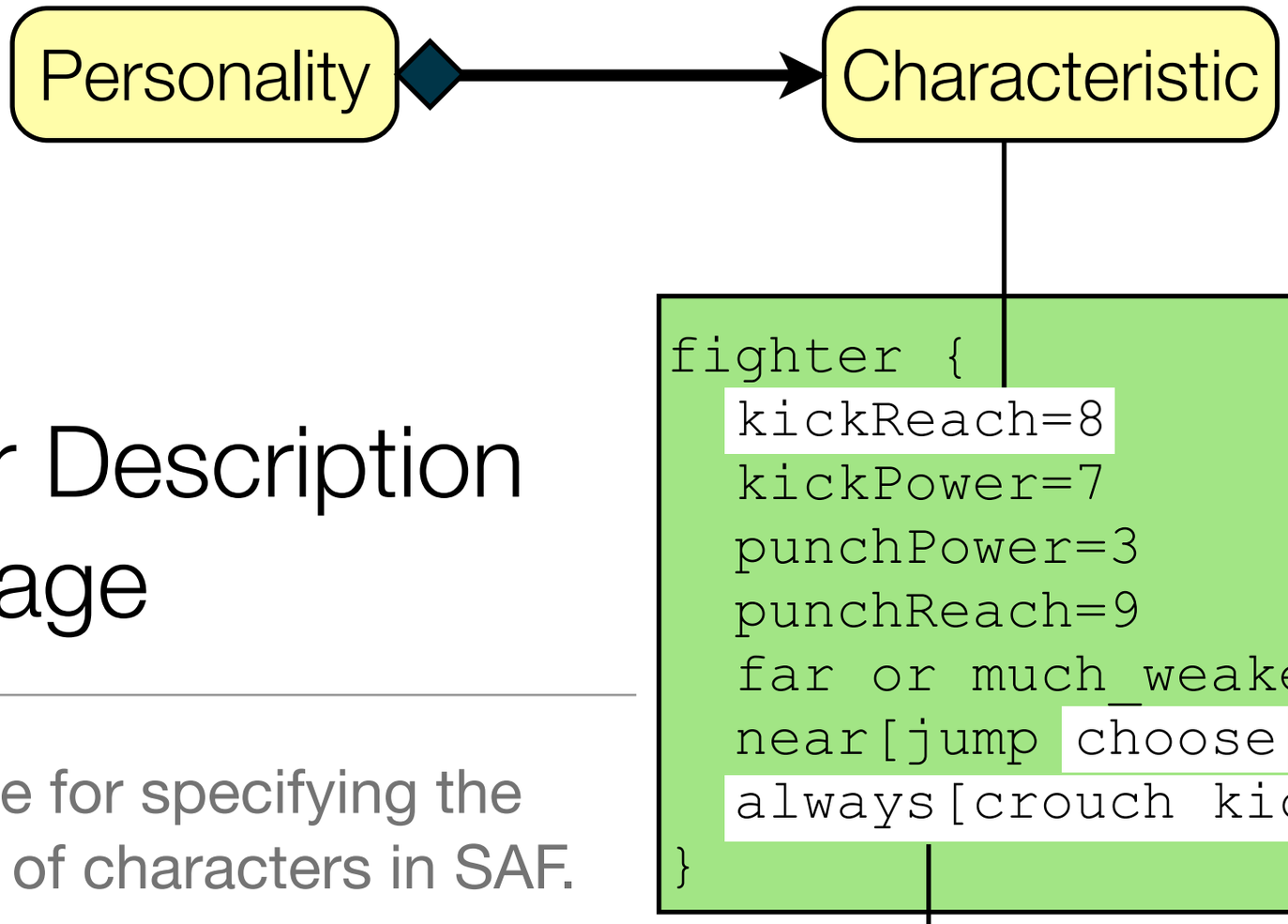
```
fighter{
  punchReach=8
  near[jump ...
```



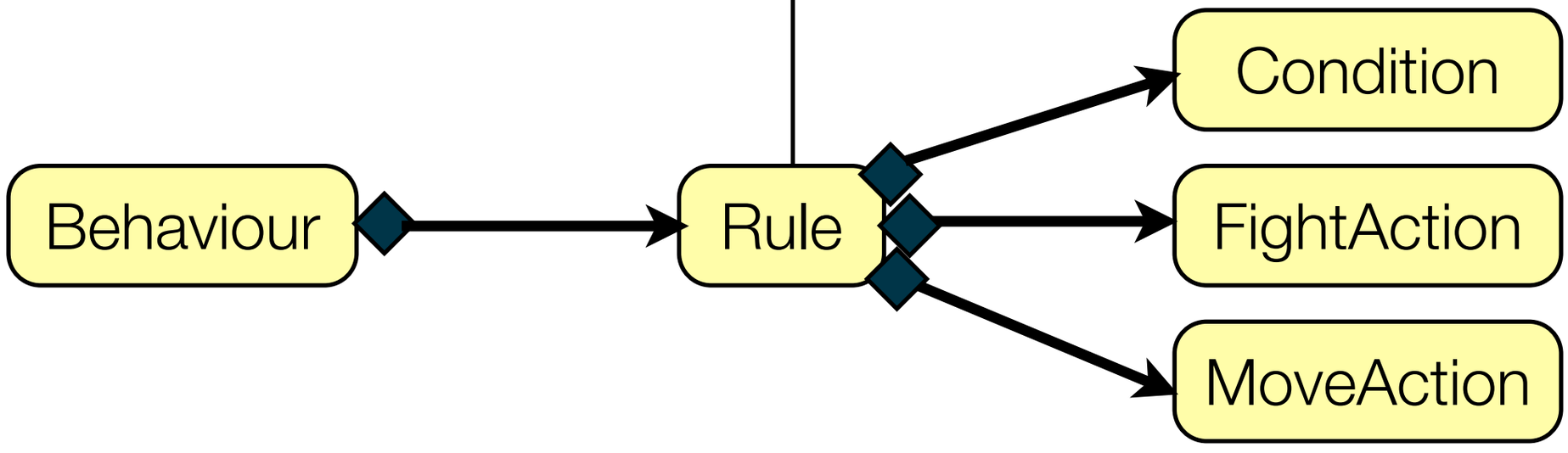
Human Opponents

Fighter Description Language

A language for specifying the behaviour of characters in SAF.

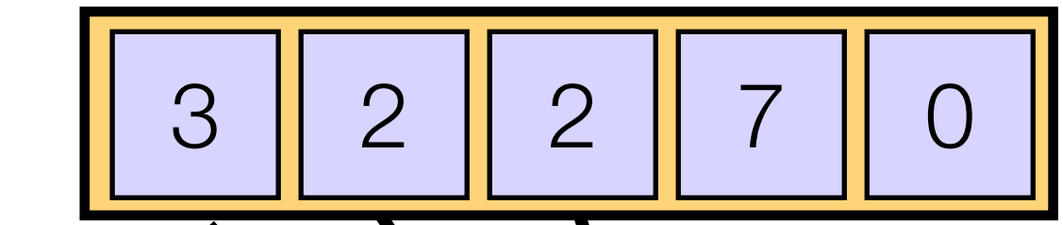



```
fighter {
  kickReach=8
  kickPower=7
  punchPower=3
  punchReach=9
  far or much_weaker[crouch punch_high]
  near[jump choose(kick_low block_low)]
  always[crouch kick_high]
}
```



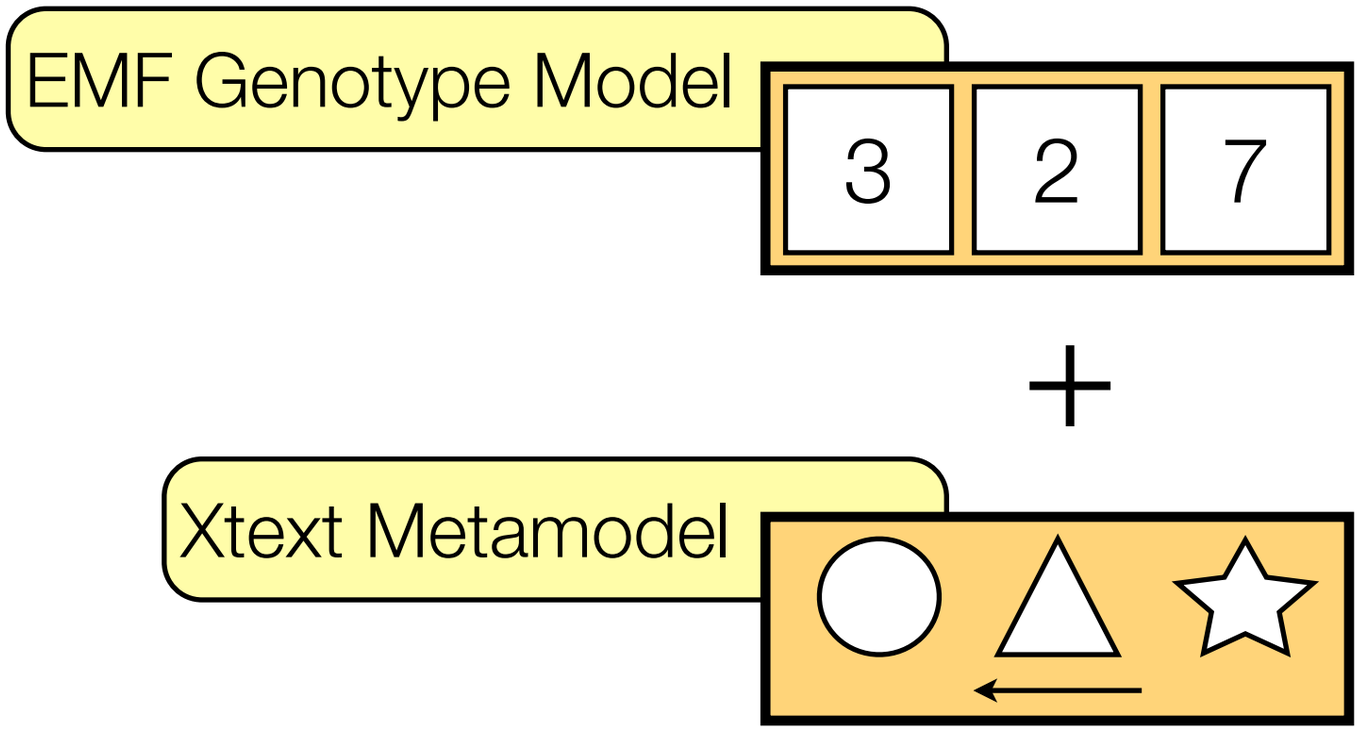
Mapping

Grammatical evolution-like mapping from genotype to phenotype.



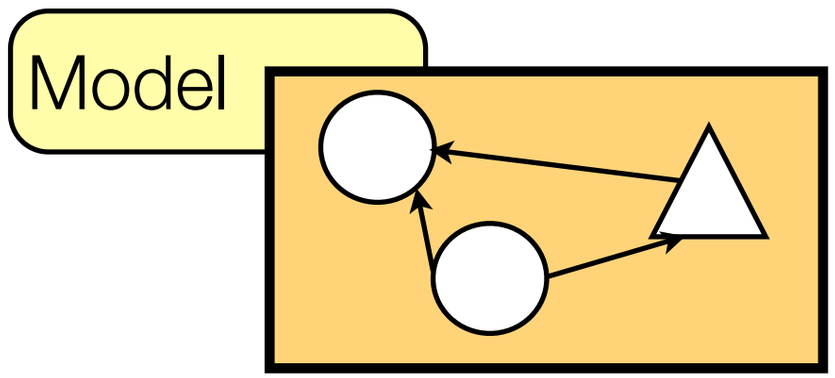

```
Bot: Personality Behaviour
Personality: Characteristic
     | Characteristic Personality
Characteristic: KP | KR | PP | PR
KP: ‘kickPower =’ PowerValue
KR: ‘kickReach =’ ReachValue
PP: ‘punchPower =’ PowerValue
PR: ‘punchReach =’ ReachValue
PowerValue: ‘0’ | ‘1’ | ‘2’ ...
ReachValue: ‘0’ | ‘1’ | ‘2’ ...
Behaviour: Rule | Rule Behaviour
Rule: Condition ‘[‘ MoveAction
FightAction ‘]’
...
```

Implementation



Eclipse Modeling Framework
Xtext
Epsilon

model to text transformation
using Epsilon



www.eclipse.org/xtext

www.eclipse.org/emf

www.eclipse.org/gmt/epsilon/

Results

Some unexpected – and very useful – outcomes.

```
fighter{
   punchPower=9
   punchPower=7
   punchPower=2
   kickPower=7
   punchPower=2
   kickPower=2
   near[crouch punch_low]
   stronger or far[choose(run_towards
      run_towards) kick_high]
   much_weaker and weaker[walk_away
      block_low]
   always[crouch kick_high]
}
```

Features

Desirable mapping features

Explicitly model-based

Handles model references

Amenable to search

Represents entire space

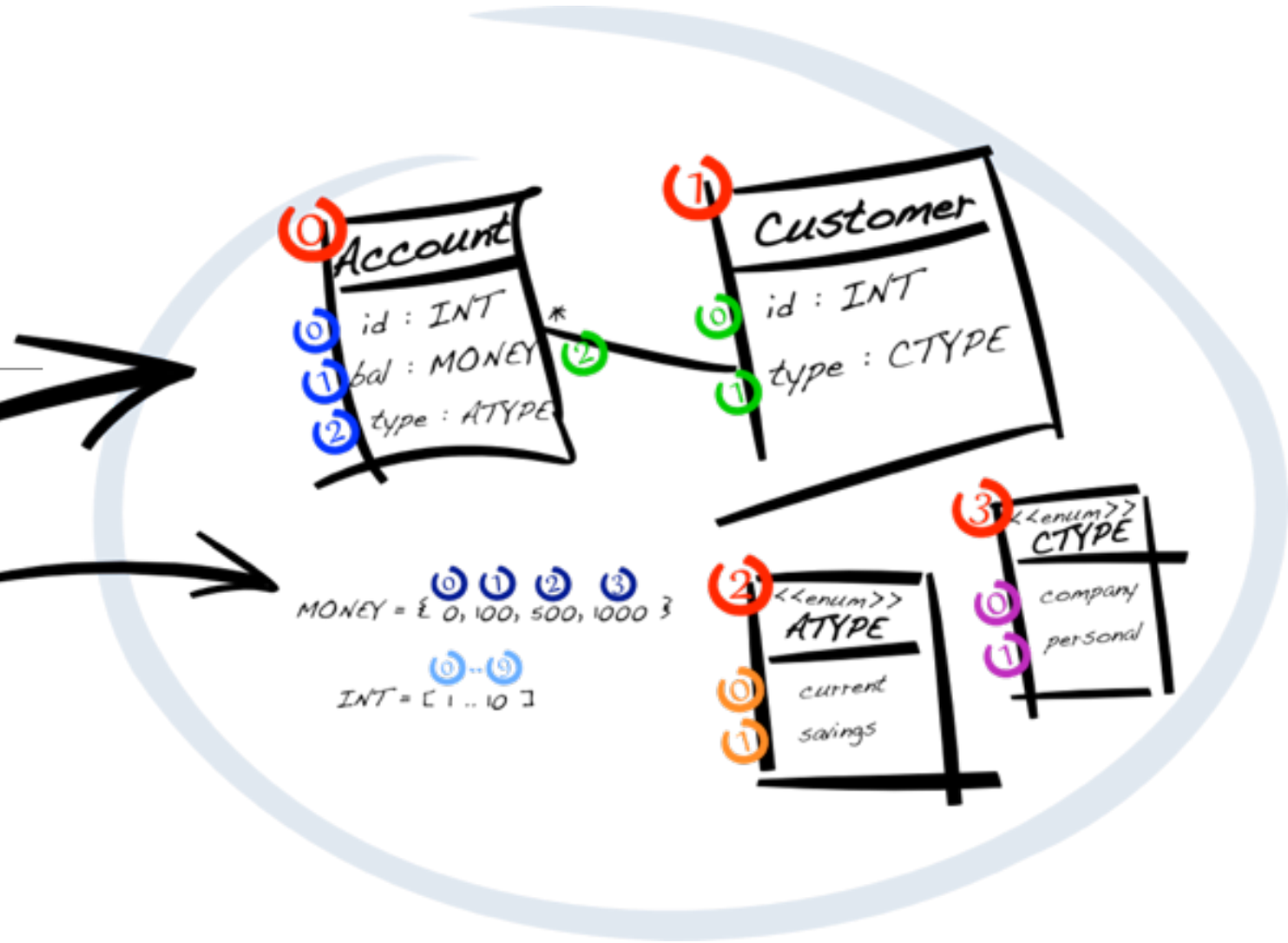
Proposed Encoding

Simple representations of complex structures.

1. Assign IDs to all features of the metamodel

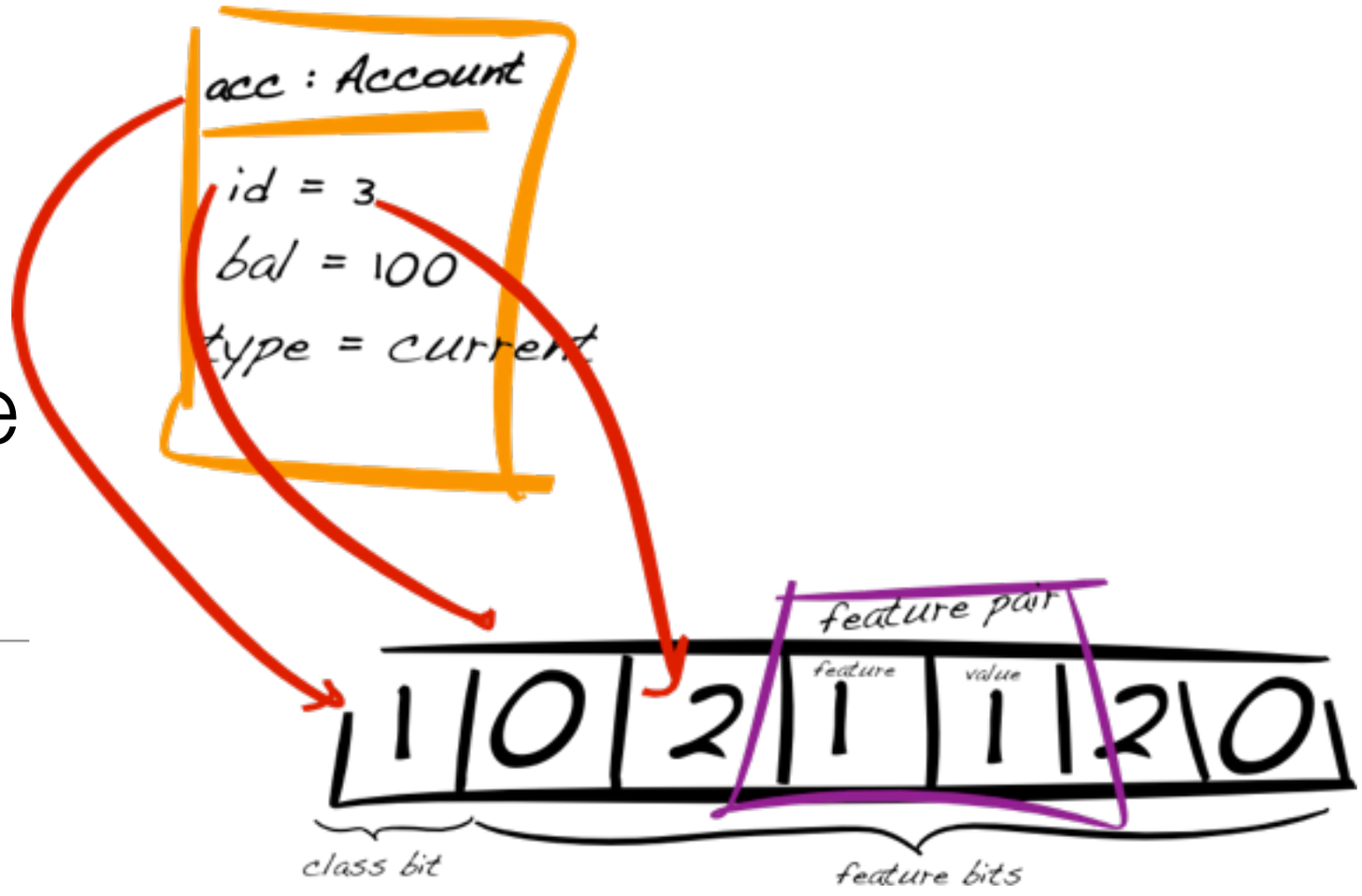
IDs relative to parents.

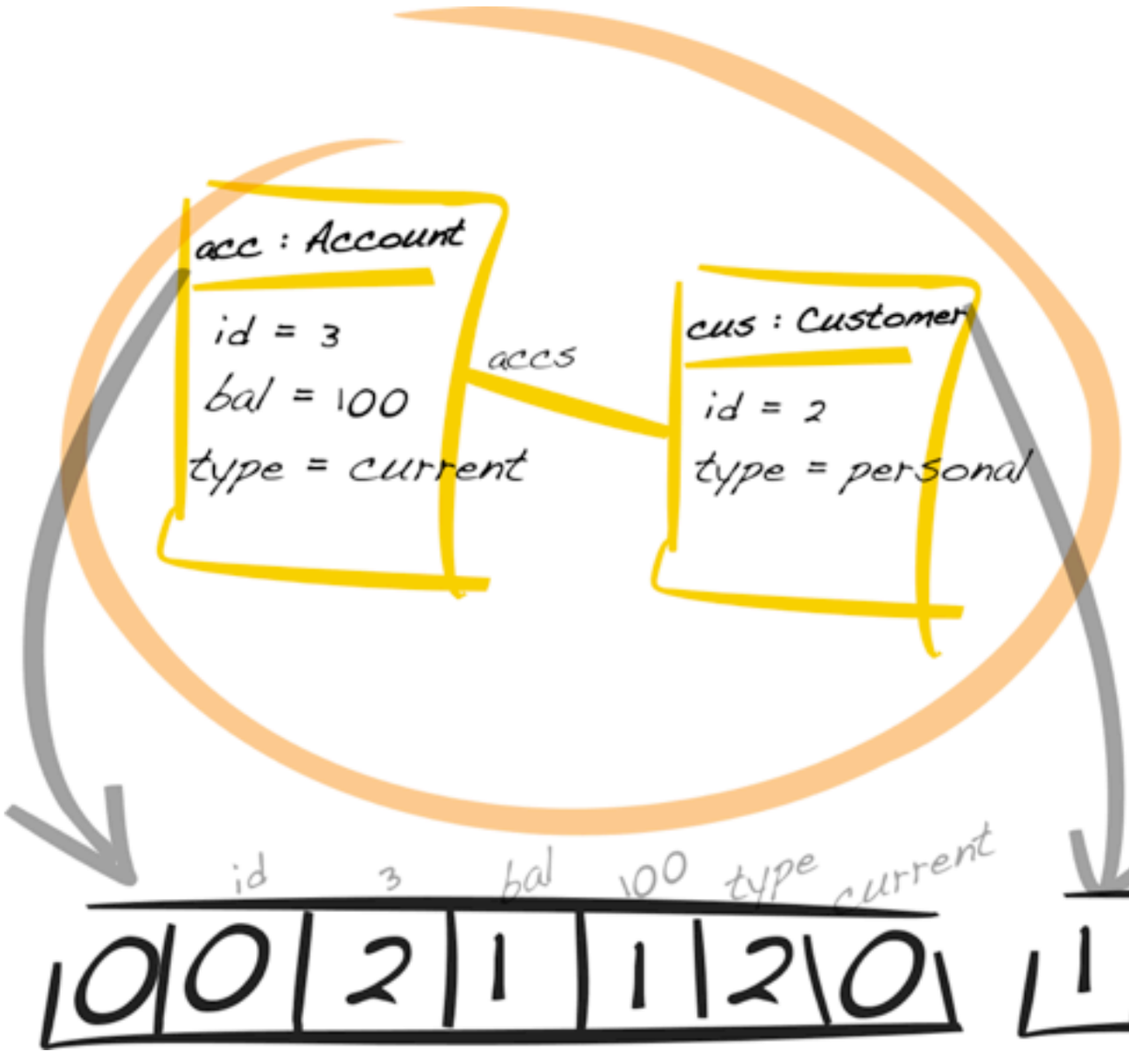
Bound infinite types / specify exemplar values.



2. Map model elements using these IDS

Each class maps to one integer substring.



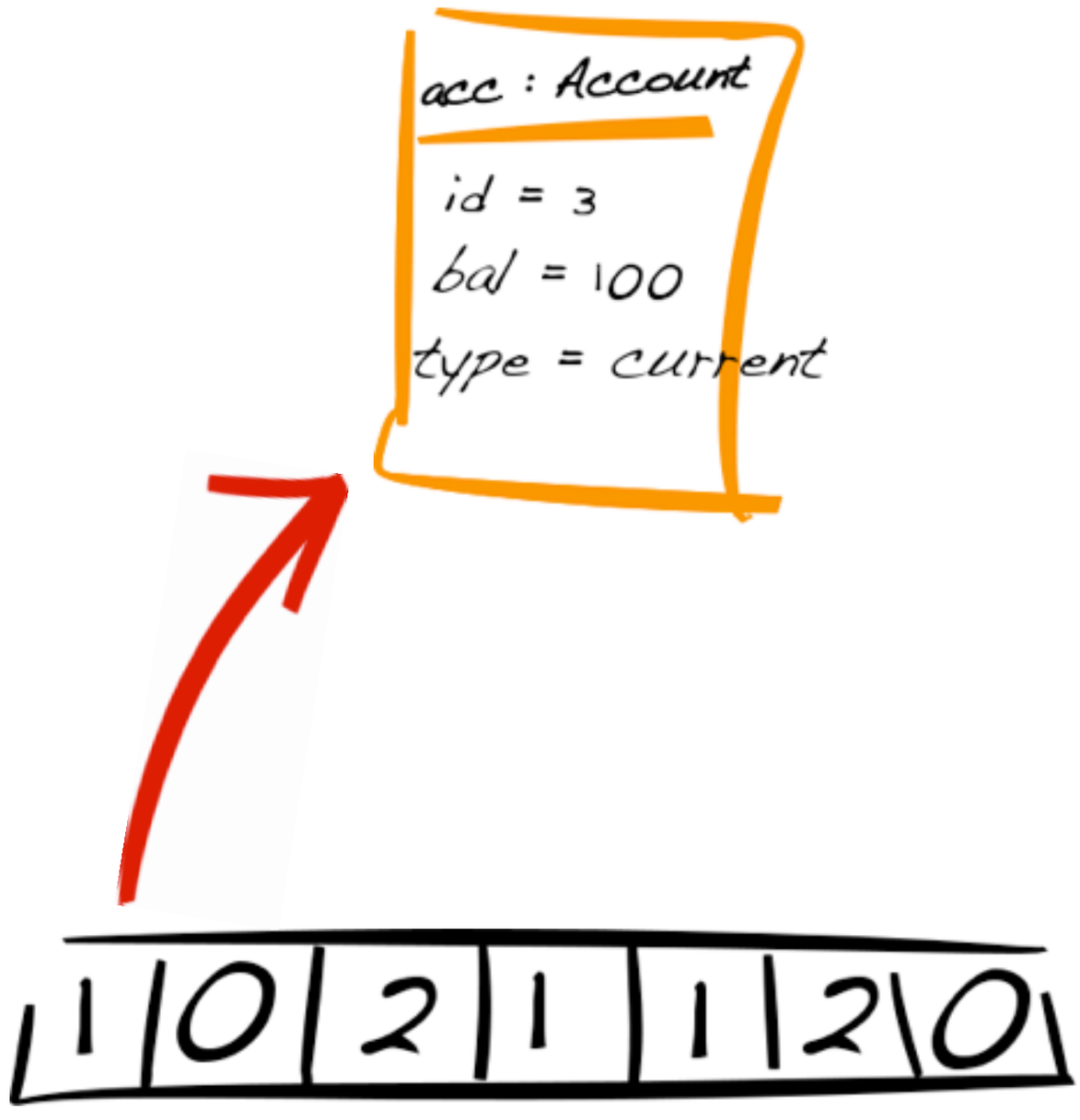


3. Resolve references

Resulting in an integer string representation.

4. Reverse

Generate models!



Features

Desirable mapping features

Explicitly model-based ✓

Handles model references ✓

Amenable to search ?

Represents entire space ?

MDE and SBSE

Search Based Model Generation

Future Work

Fitness Function

What makes a good model?

Random

Constrained

Distinct

Example-driven

Future Work

Taking this further...

IDE Integration

Empirical Work

Cross-model referencing

Practitioner input + evaluation

MDE and Search

Search Based Model Generation

Future Work

Thank you!