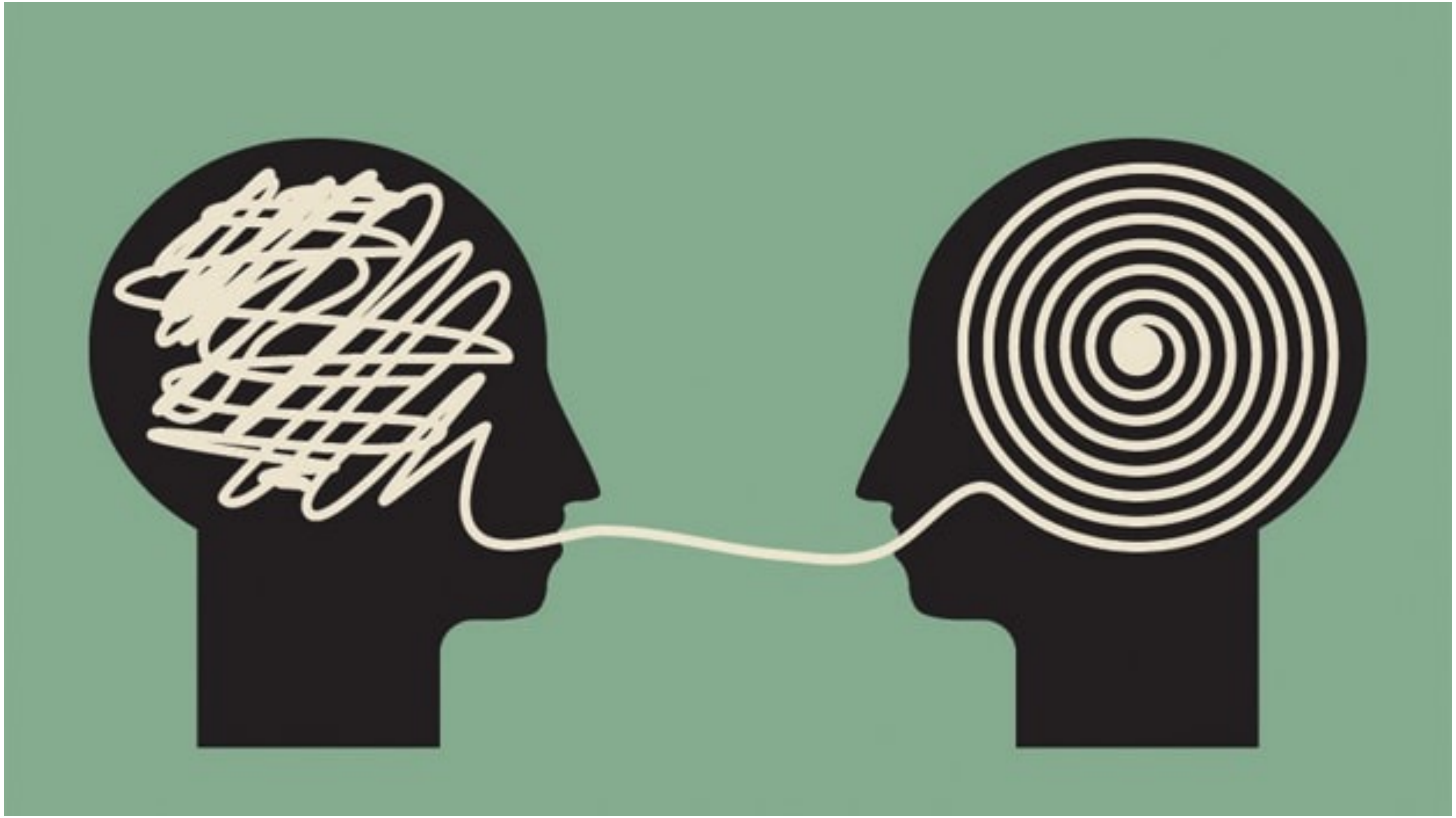
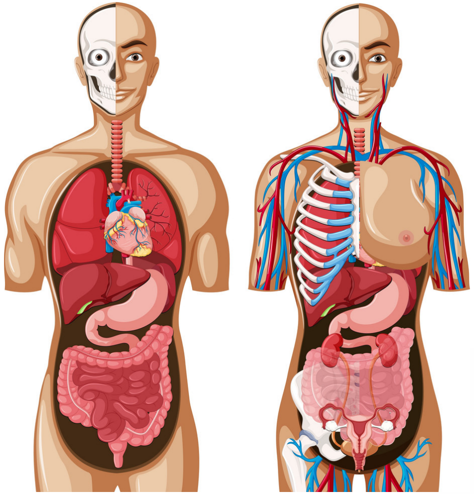
Model Execution: From a Retrospective on Code Generation to a Perspective on Model Compilation

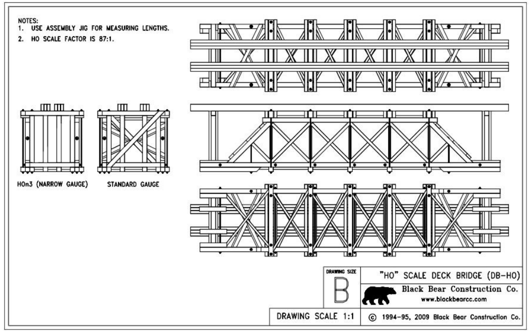
Federico Ciccozzi

federico.ciccozzi@mdu.se

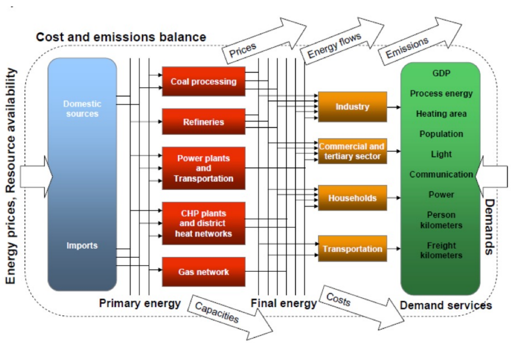


Models

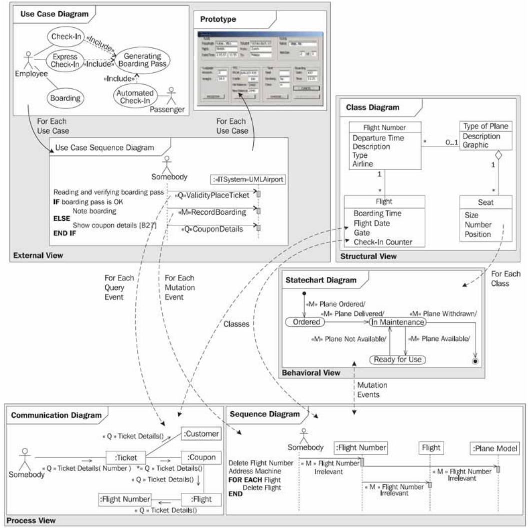




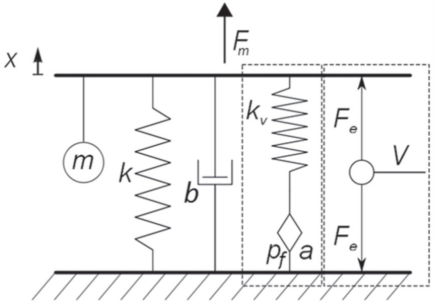
NOTES:
1. USE ASSEMBLY JIG FOR MEASURING LENGTHS.
2. HO SCALE FACTOR IS 87:1.
HOn3 (NARROW GAUGE)
STANDARD GAUGE
DRAWING SIZE
B
"HO" SCALE DECK BRIDGE (DB-HO)
Black Bear Construction Co.
www.blackbearcc.com
DRAWING SCALE 1:1
© 1994-95, 2009 Black Bear Construction Co.



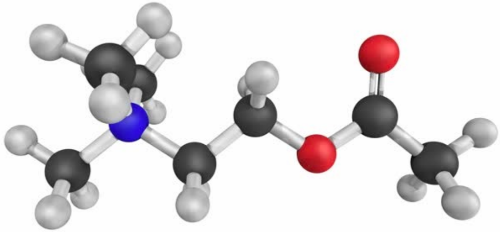
Cost and emissions balance
Prices
Energy flows
Emissions
Energy prices, Resource availability
Domestic sources
Imports
Coal processing
Refineries
Power plants and Transportation
CHP plants and district heat networks
Gas network
Industry
Commercial and tertiary sector
Households
Transportation
GDP
Process energy
Heating area
Population
Light
Communication
Power
Person kilometers
Freight kilometers
Demands
Primary energy
Capacities
Final energy
Costs
Demand services



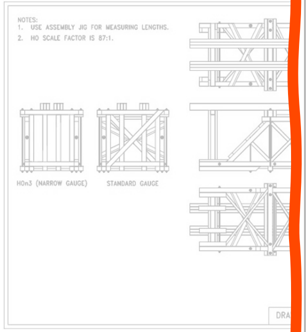
Use Case Diagram
Prototype
Check-In
Express Check-In
Generating Boarding Pass
Employee
Boarding
Automated Check-In
Passenger
For Each Use Case
For Each Use Case
Use Case Sequence Diagram
Somebody
External View
Class Diagram
Flight Number
Departure Time
Description
Type
Airline
Type of Plane
Description
Graphic
Flight
Boarding Time
Flight Date
Gate
Check-In Counter
Seat
Size
Number
Position
Structural View
For Each Query Event
For Each Mutation Event
Classes
For Each Class
Statechart Diagram
Ordered
In Maintenance
Ready for Use
Behavioral View
Mutation Events
Communication Diagram
:Customer
:Ticket
:Coupon
Somebody
:Flight Number
:Flight
Sequence Diagram
Somebody
:Flight Number
Flight
:Plane Model
Process View



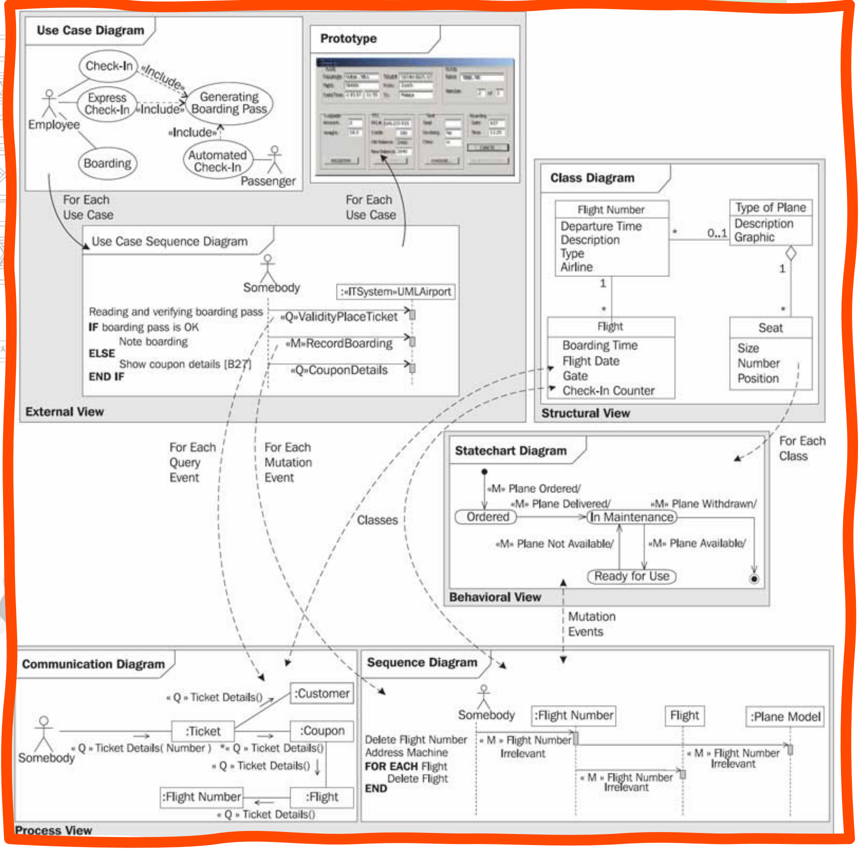
F_m
x
m
k
b
k_v
F_e
V
F_e
p_f
a

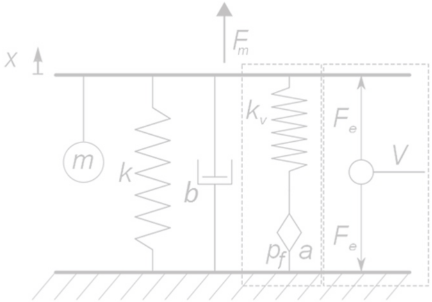


Models


NOTES:
1. USE ASSEMBLY JIG FOR MEASURING LENGTHS.
2. HO SCALE FACTOR IS 87:1.
HOn3 (NARROW GAUGE)
STANDARD GAUGE


Cost and emissions balance
Prices
Energy flows
Emissions
Coal processing
Industry
GDP
Process energy
Heating area


Use Case Diagram
Check-In
Express Check-In
«Include»
Generating Boarding Pass
Employee
Boarding
Automated Check-In
Passenger
Prototype
For Each Use Case
For Each Use Case
Use Case Sequence Diagram
Somebody
:«ITSystem»UMLAirport
Reading and verifying boarding pass
IF boarding pass is OK
Note boarding
ELSE
Show coupon details [B27]
END IF
«Q»ValidityPlaceTicket
«M»RecordBoarding
«Q»CouponDetails
External View
Class Diagram
Flight Number
Departure Time
Description
Type
Airline
Type of Plane
Description
Graphic
Flight
Boarding Time
Flight Date
Gate
Check-In Counter
Seat
Size
Number
Position
Structural View
For Each Query Event
For Each Mutation Event
Classes
For Each Class
Statechart Diagram
«M» Plane Ordered/
«M» Plane Delivered/
«M» Plane Withdrawn/
Ordered
In Maintenance
«M» Plane Not Available/
«M» Plane Available/
Ready for Use
Behavioral View
Mutation Events
Communication Diagram
Somebody
:Customer
:Ticket
:Coupon
:Flight Number
:Flight
« Q » Ticket Details()
« Q » Ticket Details(Number)
Sequence Diagram
Somebody
:Flight Number
Flight
:Plane Model
Delete Flight Number
Address Machine
FOR EACH Flight
Delete Flight
END
« M » Flight Number Irrelevant
Process View


x
F_m
m
k
b
k_v
F_e
V
p_f
a
F_e

Models

- Models can be found in any scientific discipline
- We need to be precise and specific on ***WHAT a model is***

Models

- Models can be found in any scientific discipline
- We need to be precise and specific on ***WHAT a model is***

"A model is an abstract representation of a specific part, problem, solution, or feature of a specific domain"

Software models (some definitions)

“Software models are ways of expressing a software design”

“Software models are representations of software systems made to understand, analyze, and design such systems”

“A software model is a collection of representations whose contents depend on the languages and tools used”

“Software models are formal methods for handling the process of creating software”

Software models in our context

A (software) model

- is a ***blue-print*** of a software application,

Software models in our context

A (software) model

- is a ***blue-print*** of a software application,
- can be itself ***executable***, and

Software models in our context

A (software) model

- is a ***blue-print*** of a software application,
- can be itself ***executable***, and
- is directly usable for ***automating the development*** process

Model execution

- Process of partly or fully running a computational model in a software environment

Model execution

- Process of partly or fully running a computational model in a software environment
- Simulation for prediction/monitoring based on specific inputs and configurations

Model execution

- Process of partly or fully running a computational model in a software environment
- Simulation for prediction/monitoring based on specific inputs and configurations
- Pivotal to understand complex systems

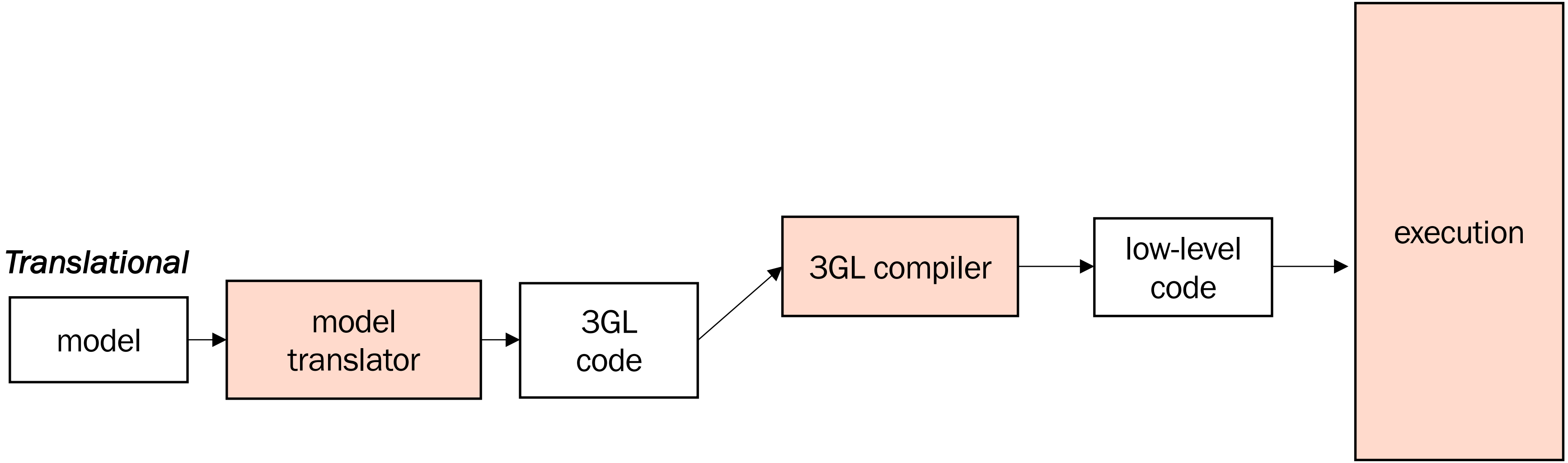
Model execution

- Process of partly or fully running a computational model in a software environment
- Simulation for prediction/monitoring based on specific inputs and configurations
- Pivotal to understand complex systems
- Pivotal to forecast outcomes related to criticality aspects (e.g., time, safety, security)

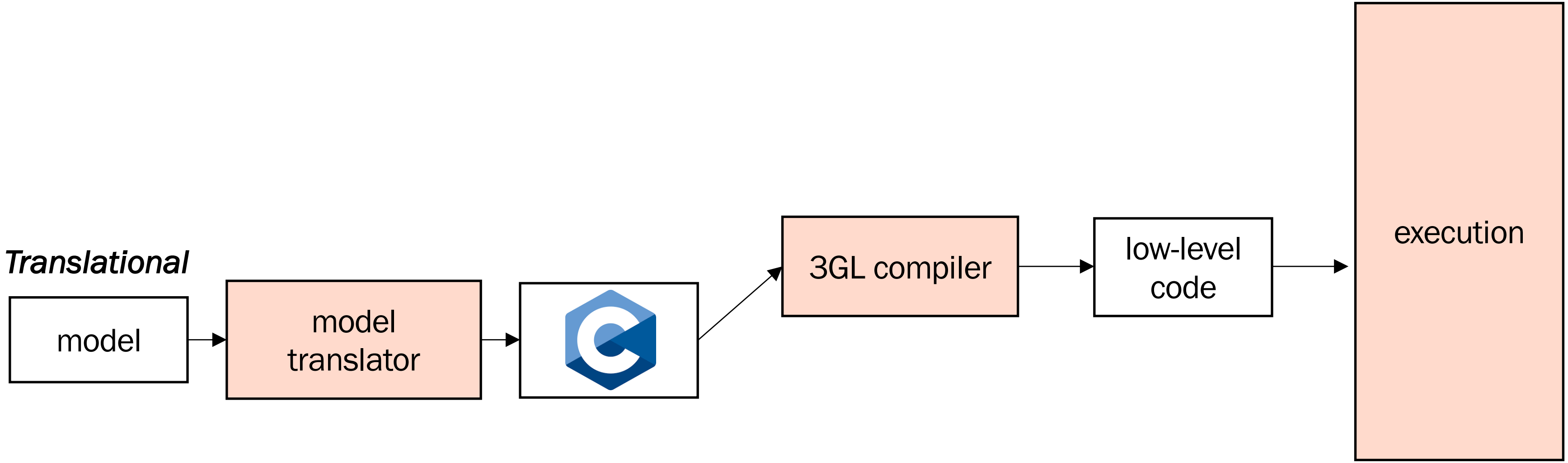
Model execution

- Process of partly or fully running a computational model in a software environment
- Simulation for prediction/monitoring based on specific inputs and configurations
- Pivotal to understand complex systems
- Pivotal to forecast outcomes related to criticality aspects (e.g., time, safety, security)
- Essential in domains like data science, AI, machine learning

Model execution strategies

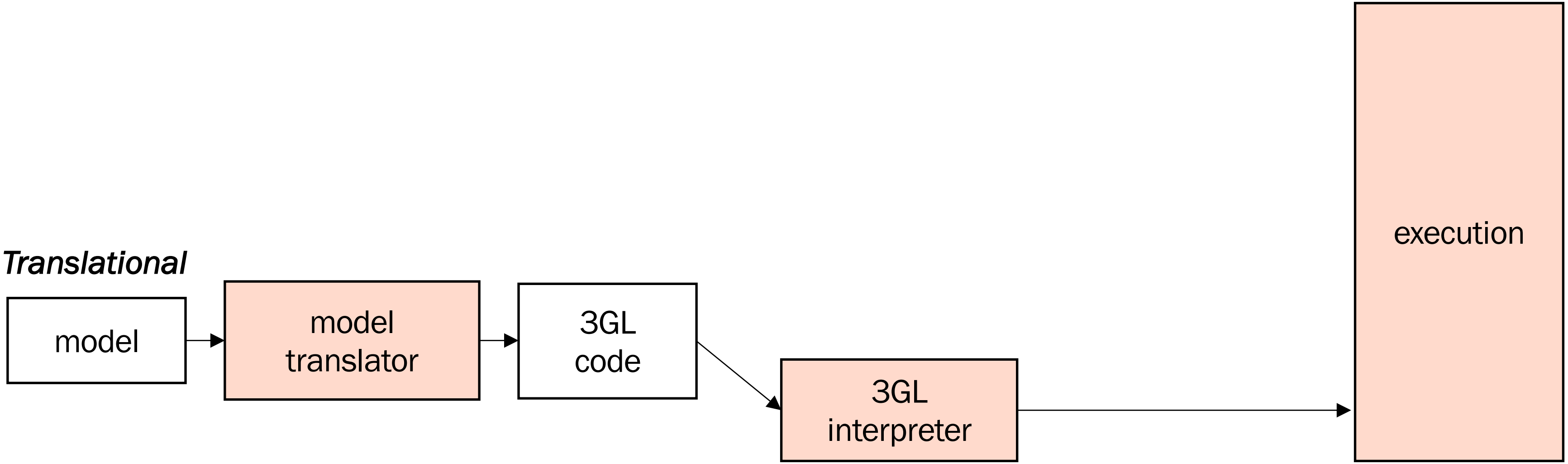


Model execution strategies

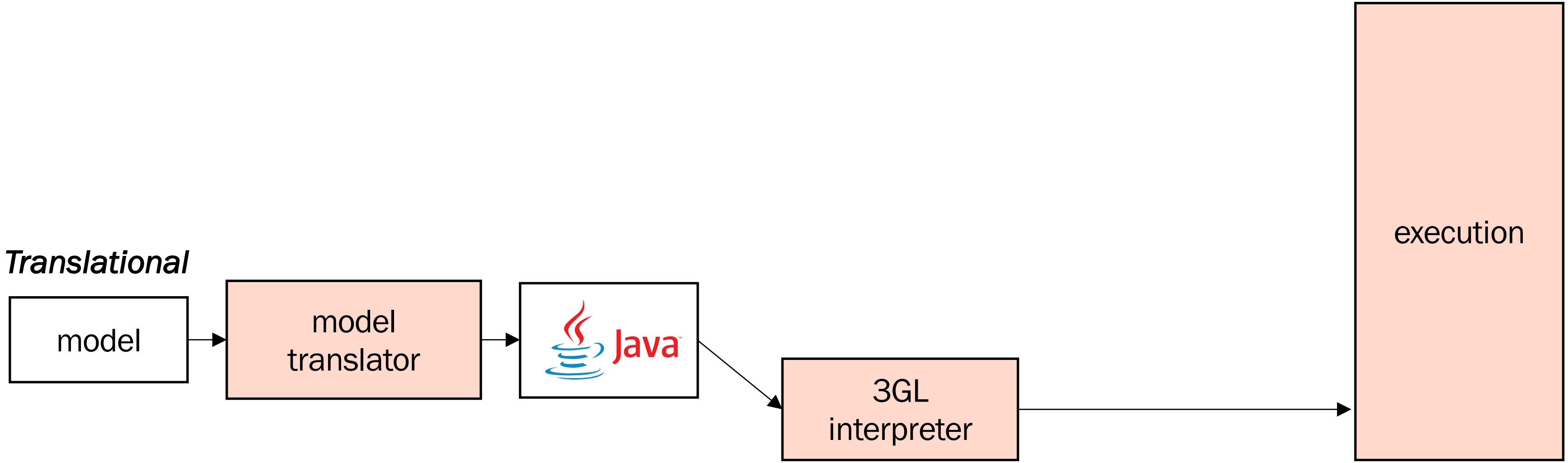


Model execution strategies

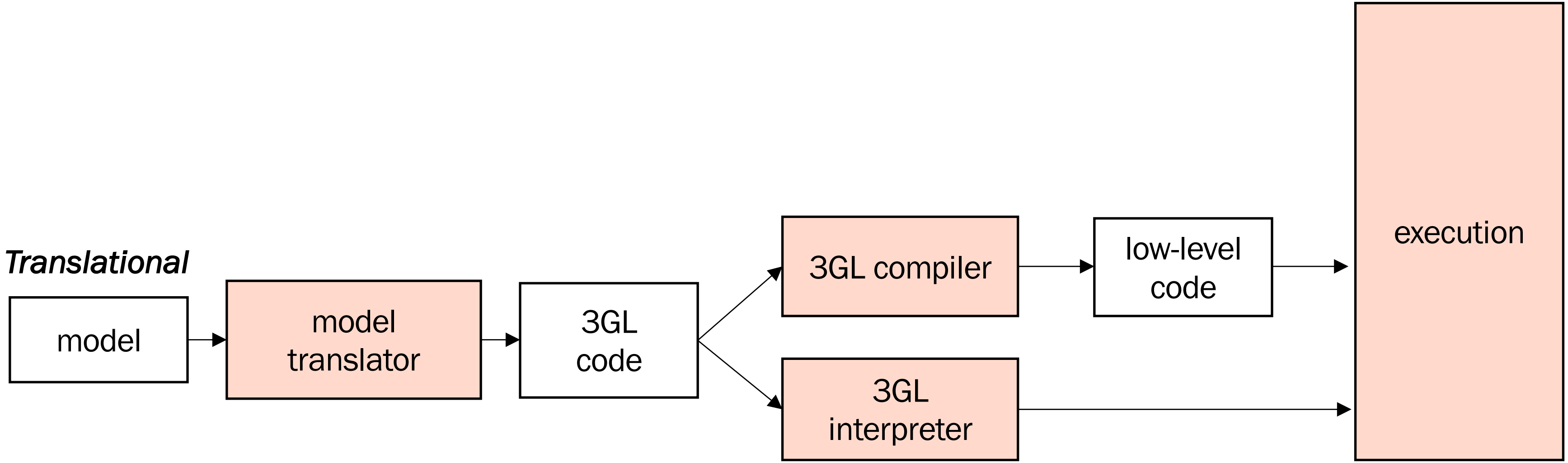
Translational



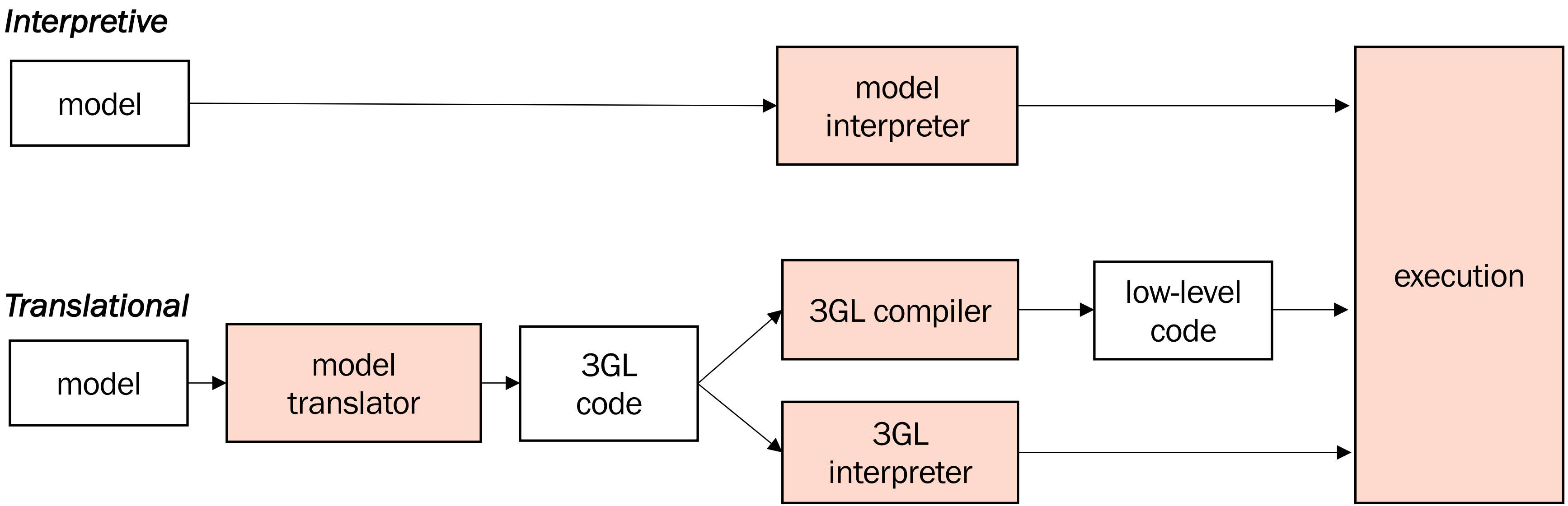
Model execution strategies



Model execution strategies



Model execution strategies

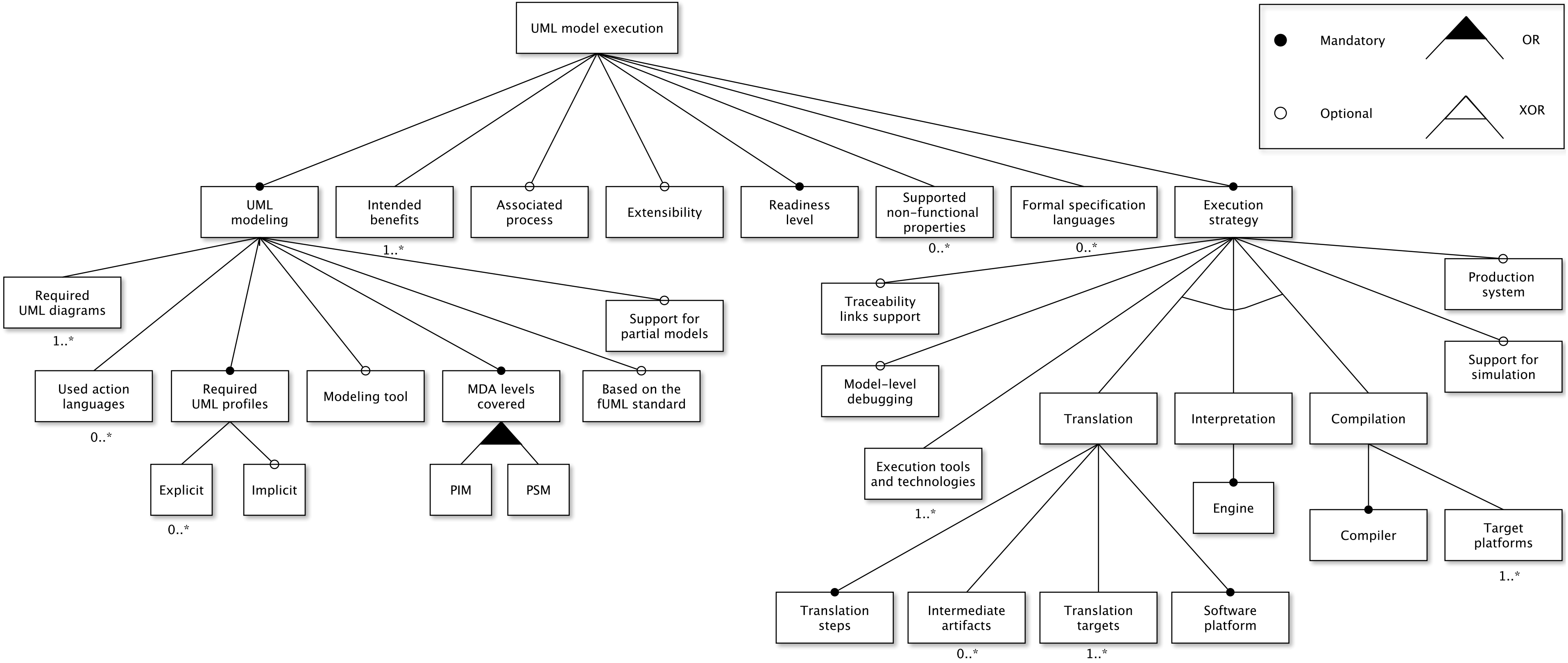


UML execution

- Systematic review of research and practice
    - Research articles
    - Tools
- Investigated >5400 items
- Included 63 articles and 19 tools
- Systematic search and data extraction

Execution of UML models: a systematic review of research and practice. Federico Ciccozzi, Ivano Malavolta, Bran Selic. Software & Systems Modeling (2019) 18:2313–2360

Classification of UML execution solutions



About executability..

- Execution strategy
  - 85% based on translation to 3GLs (mostly Java and C++)

Execution of UML models: a systematic review of research and practice. Federico Ciccozzi, Ivano Malavolta, Bran Selic. Software & Systems Modeling (2019) 18:2313–2360

About executability..

- Execution strategy
  - 85% based on translation to 3GLs (mostly Java and C++)
    - *50% use Java as model transformation/translation language*

Execution of UML models: a systematic review of research and practice. Federico Ciccozzi, Ivano Malavolta, Bran Selic. Software & Systems Modeling (2019) 18:2313–2360

About executability..

- Execution strategy
  - 85% based on translation to 3GLs (mostly Java and C++)
    - *50% use Java as model transformation/translation language*
  - 17% based on interpretation, only for simulation/analysis purposes

Execution of UML models: a systematic review of research and practice. Federico Ciccozzi, Ivano Malavolta, Bran Selic. Software & Systems Modeling (2019) 18:2313–2360

About executability..

- Execution strategy
  - 85% based on translation to 3GLs (mostly Java and C++)
    - *50% use Java as model transformation/translation language*
  - 17% based on interpretation, only for simulation/analysis purposes
- Execution semantics
  - 15% based on fUML

Execution of UML models: a systematic review of research and practice. Federico Ciccozzi, Ivano Malavolta, Bran Selic. Software & Systems Modeling (2019) 18:2313–2360

About executability..

- Execution strategy
  - 85% based on translation to 3GLs (mostly Java and C++)
    - *50% use Java as model transformation/translation language*
  - 17% based on interpretation, only for simulation/analysis purposes
- Execution semantics
  - 15% based on fUML
- Action languages
  - >90% use 3GLs
  - Only 10% based on Alf

Execution of UML models: a systematic review of research and practice. Federico Ciccozzi, Ivano Malavolta, Bran Selic. Software & Systems Modeling (2019) 18:2313–2360

About executability..

- Execution strategy
  - 85% based on translation to 3GLs (mostly Java and C++)
    - *50% use Java as model transformation/translation language*
  - 17% based on interpretation, only for simulation/analysis purposes
- Execution semantics
  - 15% based on fUML
- Action languages
  - >90% use 3GLs
  - Only 10% based on Alf
- Support for simulation
  - 59% provide some simulation feature

Execution of UML models: a systematic review of research and practice. Federico Ciccozzi, Ivano Malavolta, Bran Selic. Software & Systems Modeling (2019) 18:2313–2360

Other interesting aspects..

- Extensibility
  - 21% provide some sort of extension mechanism

Execution of UML models: a systematic review of research and practice. Federico Ciccozzi, Ivano Malavolta, Bran Selic. Software & Systems Modeling (2019) 18:2313–2360

Other interesting aspects..

- Extensibility
  - 21% provide some sort of extension mechanism
- Traceability (model-code)
  - 18% provide some support for trace links

Execution of UML models: a systematic review of research and practice. Federico Ciccozzi, Ivano Malavolta, Bran Selic. Software & Systems Modeling (2019) 18:2313–2360

Other interesting aspects..

- Extensibility
  - 21% provide some sort of extension mechanism
- Traceability (model-code)
  - 18% provide some support for trace links
- Interactive debuggability
  - 25% provide debugging features at model-level

Execution of UML models: a systematic review of research and practice. Federico Ciccozzi, Ivano Malavolta, Bran Selic. Software & Systems Modeling (2019) 18:2313–2360

In summary

- ***Translation*** outnumbers interpretation
- Interpretation is used for higher-level execution (e.g., simulation)
- Execution semantics from ***fUML is neglected*** in most cases
- Most solutions employ ***3GLs as action languages***
- Almost no model-level interactive debugging
- Little extensibility and customizability

Execution of UML models: a systematic review of research and practice. Federico Ciccozzi, Ivano Malavolta, Bran Selic. Software & Systems Modeling (2019) 18:2313–2360

What about ”code generation” approach

- Convenient, reuse of existing (trusted) 3GL compilers

What will it take? A view on adoption of model-based methods in practice. Bran Selic. Software & Systems Modeling 11.4 (2012): 513-526.

What about ”code generation” approach

- Convenient, reuse of existing (trusted) 3GL compilers
- Creates discontinuity between model and executable
  - Model debugging can become very difficult
  - Co-debugging and co-simulation nearly impossible

**What will it take? A view on adoption of model-based methods in practice.* Bran Selic. Software & Systems Modeling 11.4 (2012): 513-526.

What about ”code generation” approach

- Convenient, reuse of existing (trusted) 3GL compilers
- Creates discontinuity between model and executable
  - Model debugging can become very difficult
  - Co-debugging and co-simulation nearly impossible
- Lack of trust from developers
  - Generated 3GL “inspected” and **modified** by hand

**What will it take? A view on adoption of model-based methods in practice.* Bran Selic. Software & Systems Modeling 11.4 (2012): 513-526.

What about ”code generation” approach

- Convenient, reuse of existing (trusted) 3GL compilers
- Creates discontinuity between model and executable
  - Model debugging can become very difficult
  - Co-debugging and co-simulation nearly impossible
- Lack of trust from developers
  - Generated 3GL “inspected” and **modified** by hand

What will it take? A view on adoption of model-based methods in practice. Bran Selic. Software & Systems Modeling 11.4 (2012): 513-526.

What about ”code generation” approach

- Convenient, reuse of existing (trusted) 3GL compilers
- Creates discontinuity between model and executable
  - Model debugging can become very difficult
  - Co-debugging and co-simulation nearly impossible
- Lack of trust from developers
  - Generated 3GL “inspected” and **modified** by hand
    - *Violate source models*
    - *Violate model-based analysis, optimisation and V&V*
    - *Get lost if source models change*

**What will it take? A view on adoption of model-based methods in practice.* Bran Selic. Software & Systems Modeling 11.4 (2012): 513-526.

What about "code generation" approach[1]

- Convenient, reuse of existing (trusted) 3GL compilers
- Creates discontinuity between model and executable
  - Model debugging can become very difficult
  - Co-debugging and co-simulation nearly impossible
- Lack of trust from developers
  - Generated 3GL "inspected" and **modified** by hand
- 3GL compilers do not understand model semantics
  - Program optimisations may be missed
  - Generated executables may be semantically different from models

**What will it take? A view on adoption of model-based methods in practice.* Bran Selic. Software & Systems Modeling 11.4 (2012): 513-526.

What about ”code generation” approach

- Convenient, reuse of existing (trusted) 3GL compilers
- Creates discontinuity between model and executable
    - Model debugging can become very difficult
    - Co-debugging and co-simulation nearly impossible
- Lack of trust from developers
    - Generated 3GL “inspected” and **modified** by hand
- 3GL compilers do not understand model semantics
    - Program optimisations may be missed
    - Generated executables may be semantically different from models
- Not suitable for heterogeneous platforms (multiple 3GLs needed)

**What will it take? A view on adoption of model-based methods in practice.* Bran Selic. Software & Systems Modeling 11.4 (2012): 513-526.

Back to code generation for UML

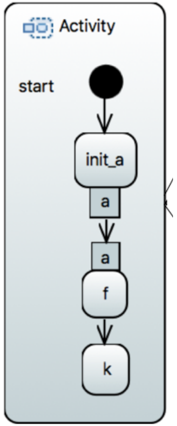
- 26 different code generators from UML to Java

Back to code generation for UML

- 26 different code generators from UML to Java
- No reference semantics (e.g. fUML)
  - Different code generators (even commercial!) produce different codes from same models

Back to code generation for UML

- 26 different code generators from UML to Java
- No reference semantics (e.g. fUML)
  - Different code generators (even commercial!) produce different codes from same models

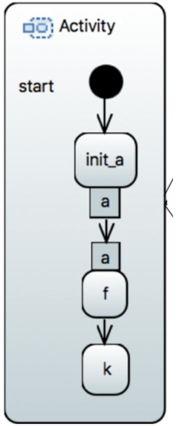


Back to code generation for UML

- 26 different code generators from UML to Java
- No reference semantics (e.g. fUML)
  - Different code generators (even commercial!) produce different codes from same models

fUML model in textual format

```
let a : A = null;
f(a); execution stops
k();
```

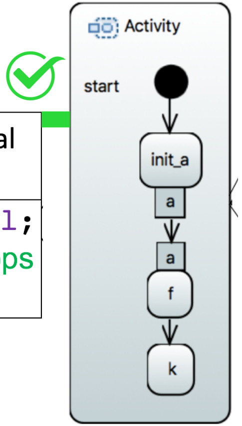


Back to code generation for UML

- 26 different code generators from UML to Java
- No reference semantics (e.g. fUML)
    - Different code generators (even commercial!) produce different codes from same models

fUML model in textual format

```
let a : A = null;
f(a); execution stops
k();
```

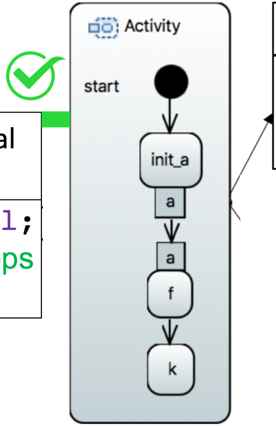


Back to code generation for UML

- 26 different code generators from UML to Java
- No reference semantics (e.g. fUML)
  - Different code generators (even commercial!) produce different codes from same models

fUML model in textual format

```
let a : A = null;
f(a); execution stops
k();
```

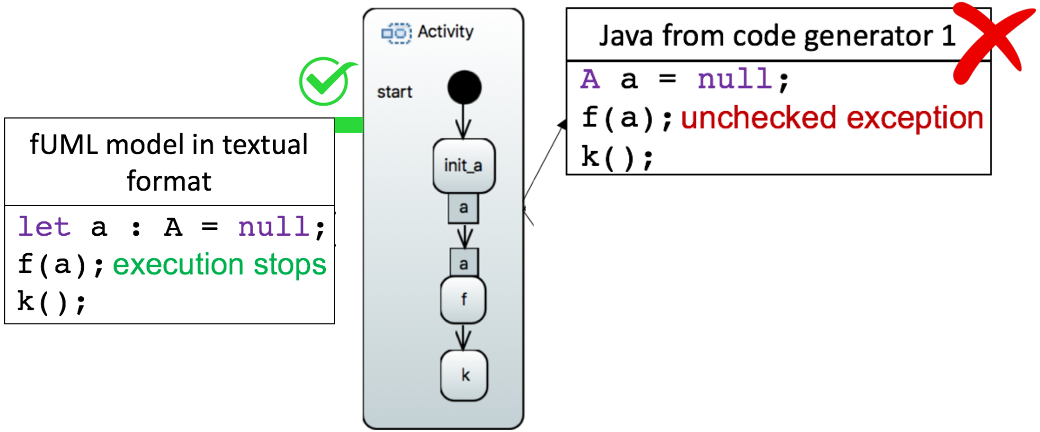


Java from code generator 1

```
A a = null;
f(a); unchecked exception
k();
```

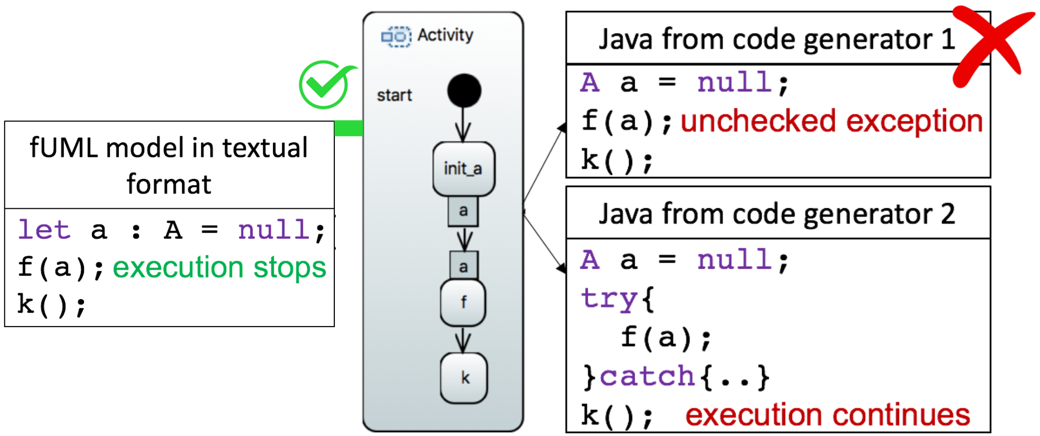
Back to code generation for UML

- 26 different code generators from UML to Java
- No reference semantics (e.g. fUML)
  - Different code generators (even commercial!) produce different codes from same models



Back to code generation for UML

- 26 different code generators from UML to Java
- No reference semantics (e.g. fUML)
  - Different code generators (even commercial!) produce different codes from same models

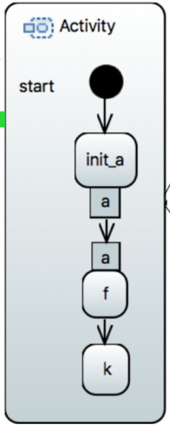


Back to code generation for UML

- 26 different code generators from UML to Java
- No reference semantics (e.g. fUML)
  - Different code generators (even commercial!) produce different codes from same models

fUML model in textual format

```
let a : A = null;
f(a); execution stops
k();
```



Java from code generator 1

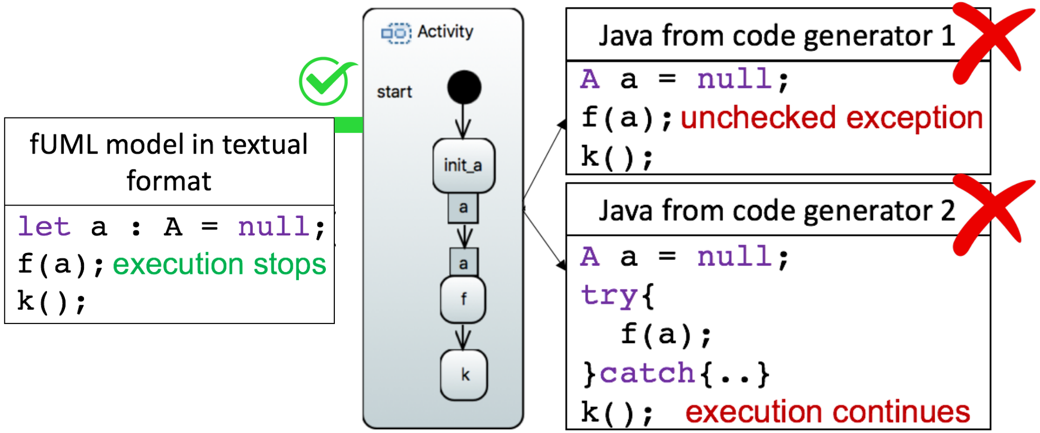
```
A a = null;
f(a); unchecked exception
k();
```

Java from code generator 2

```
A a = null;
try{
  f(a);
}catch{..}
k();   execution continues
```

Back to code generation for UML

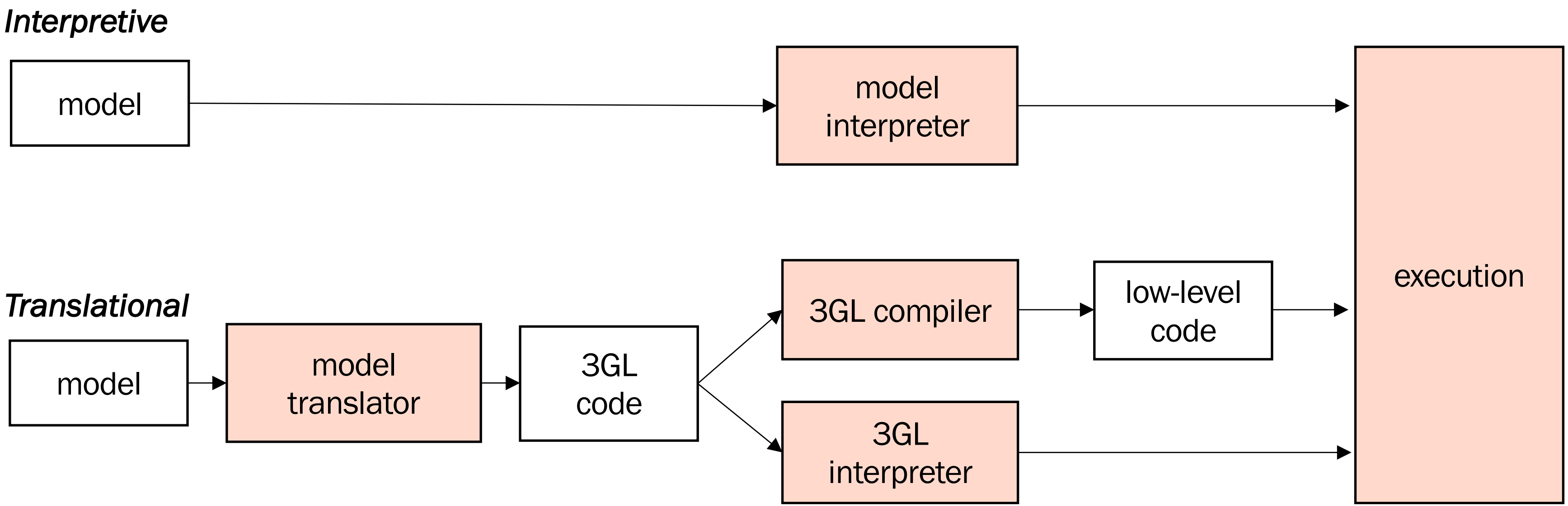
- 26 different code generators from UML to Java
- No reference semantics (e.g. fUML)
  - Different code generators (even commercial!) produce different codes from same models



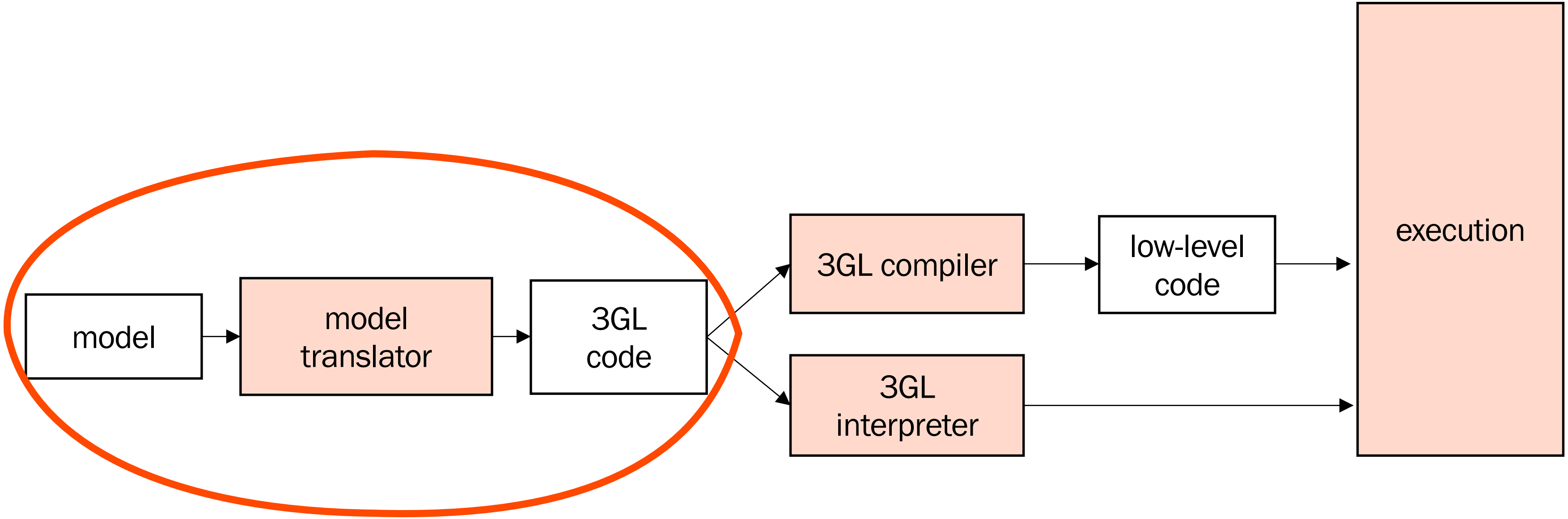
Some of the issues

- *Predictability*
- *Validity of MB-analysis*
- *Consistency model-code*
- *Bidirectional traceability*
- *Co-simulation*
- *Co-debugging*

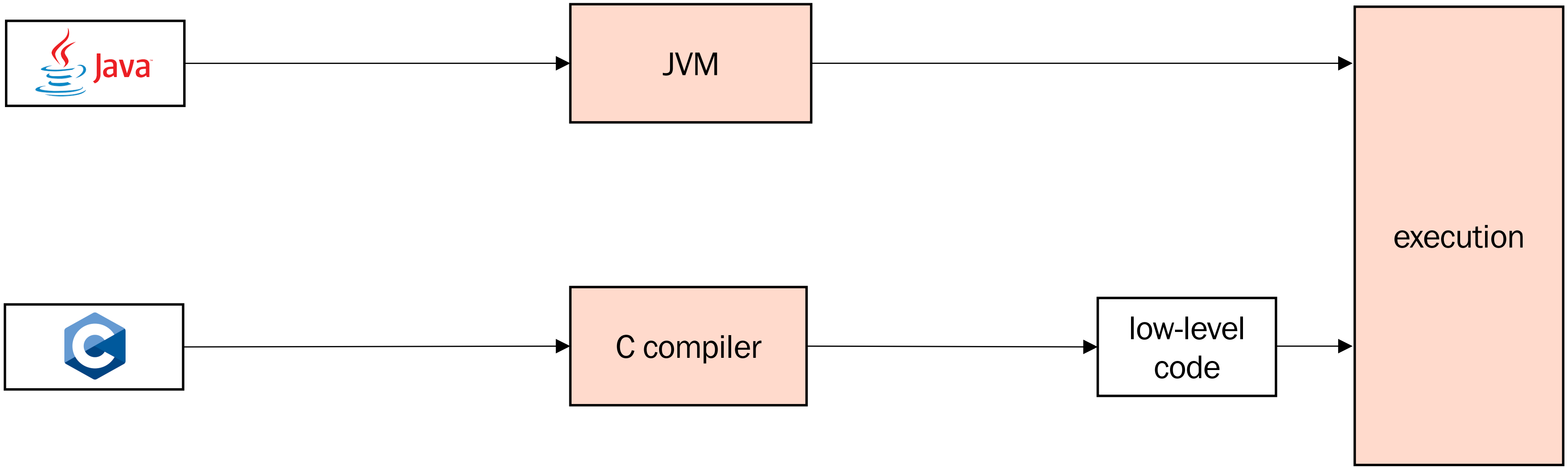
Model execution strategies



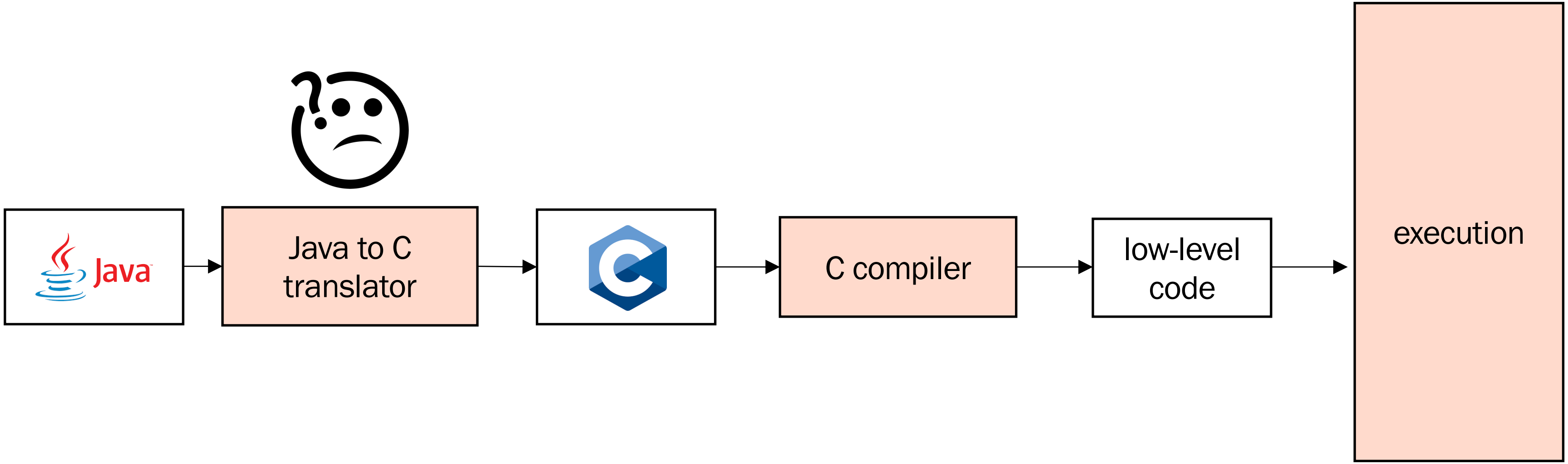
Why are we generating 3GL code?!



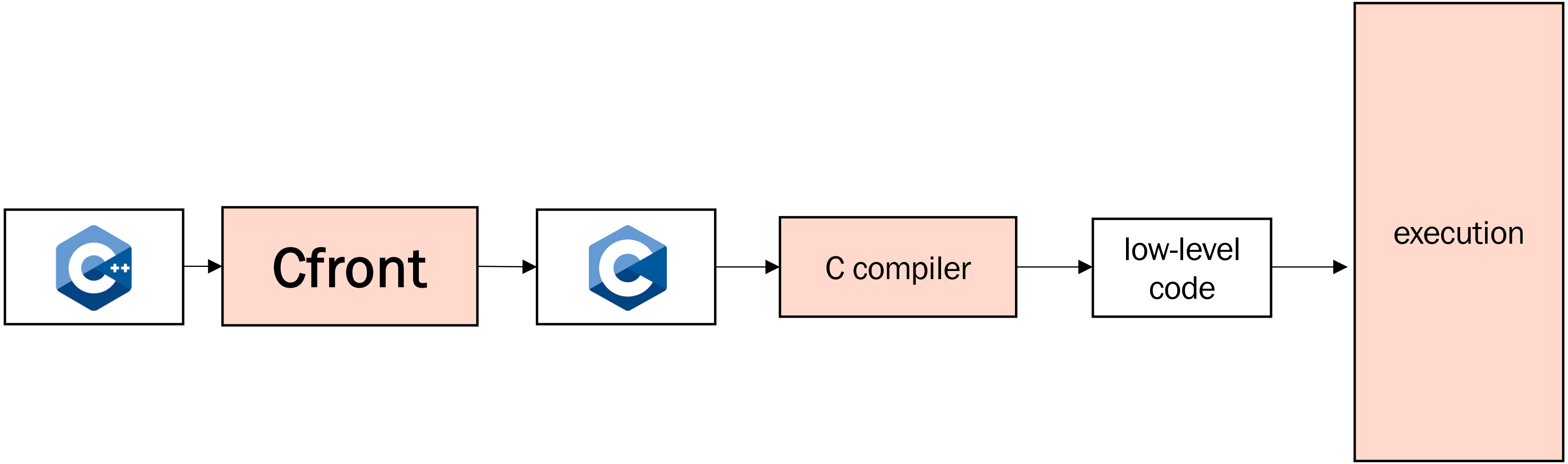
Why are we generating 3GL code?!



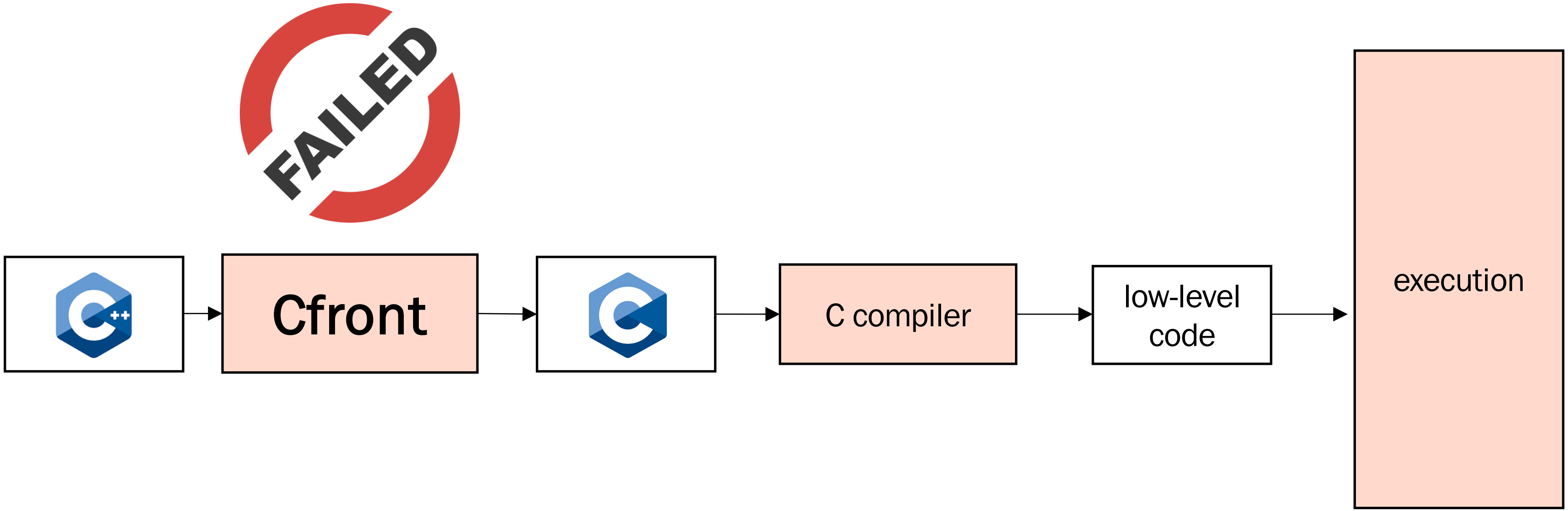
Why are we generating 3GL code?!



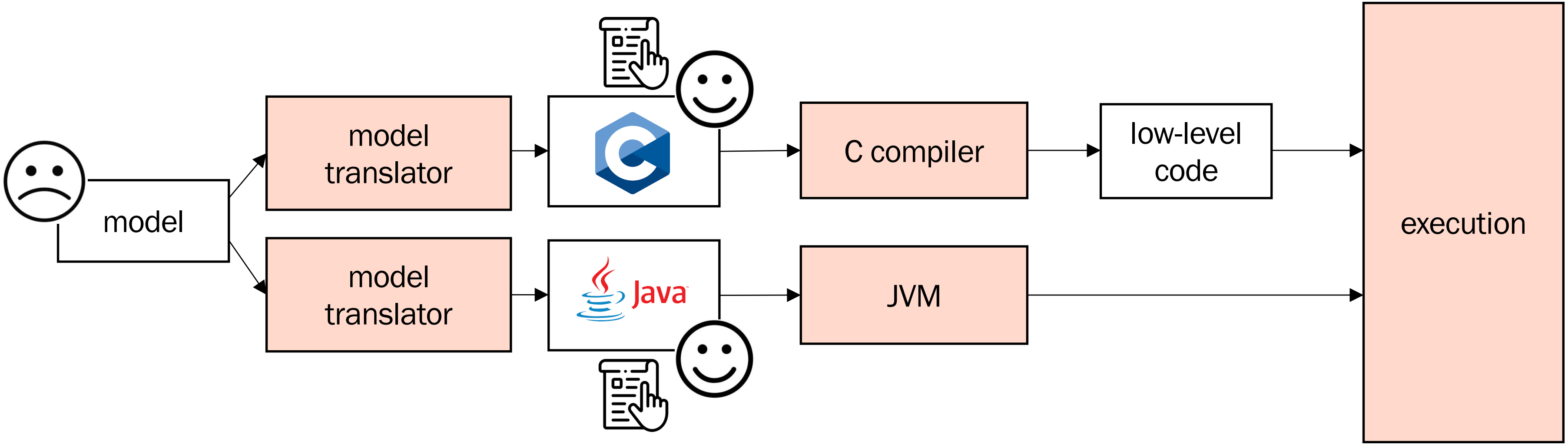
We have tried before..

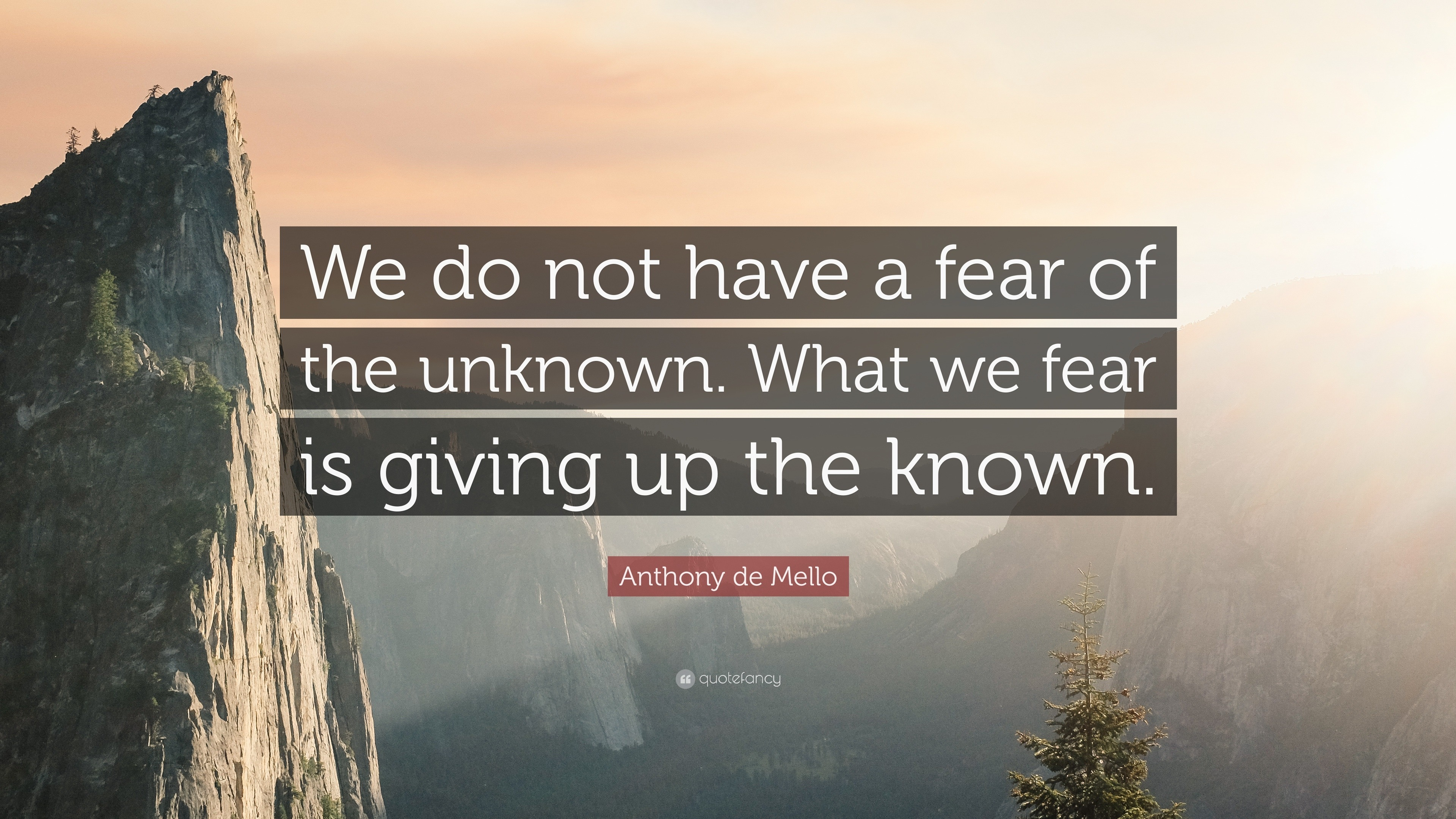


.. And not succeeded

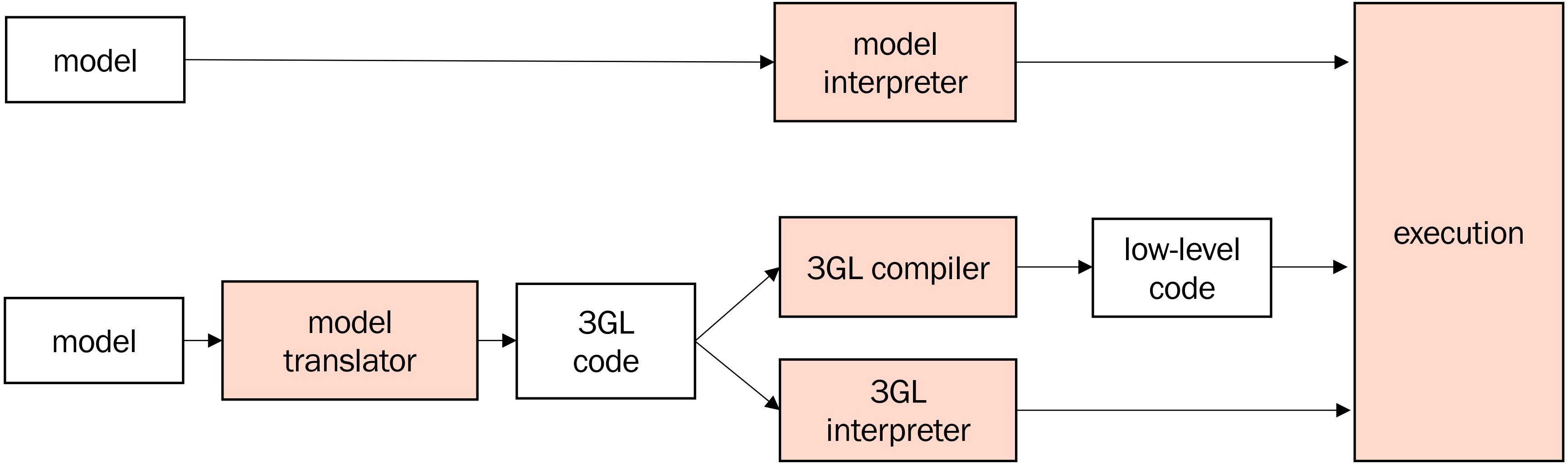


Code generation.. why



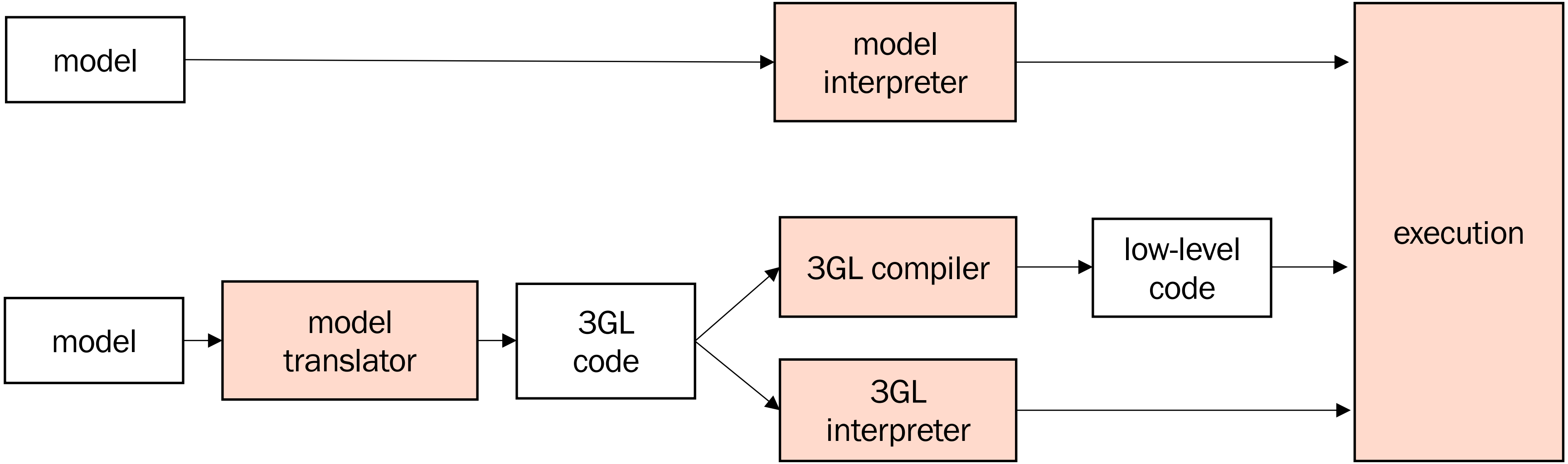

We do not have a fear of the unknown. What we fear is giving up the known.
Anthony de Mello
quotefancy

Model execution strategies

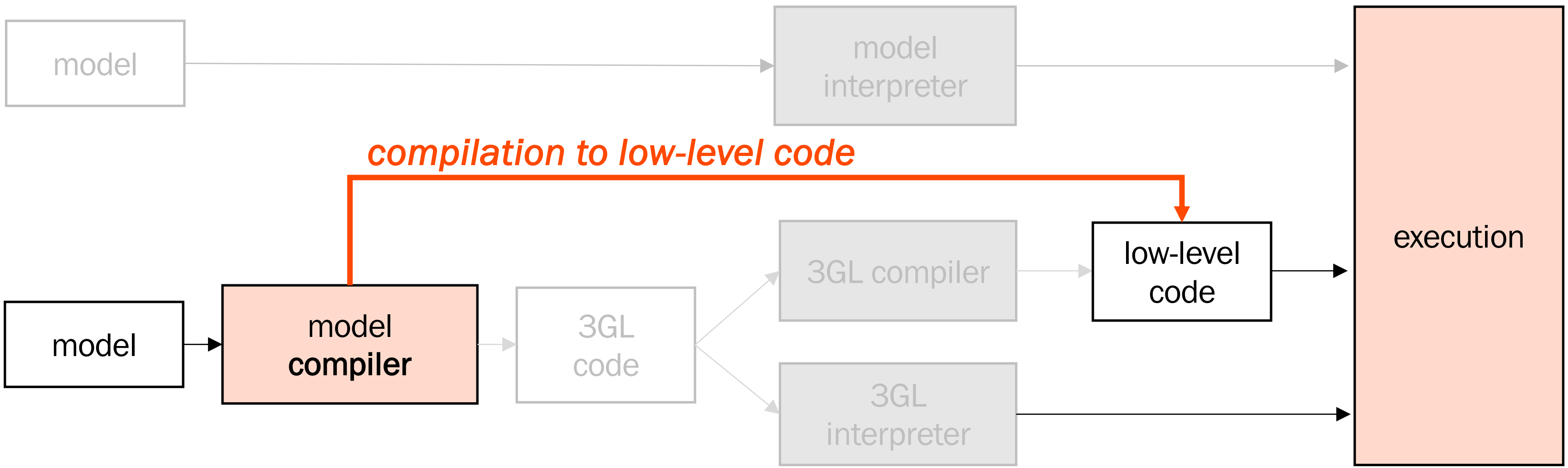


Model execution strategies

In H2020 SPACE-10-TEC-2020, ***new model execution*** *approaches regarded as the* ***most urgent software engineering*** *need in the* ***space*** *industry*

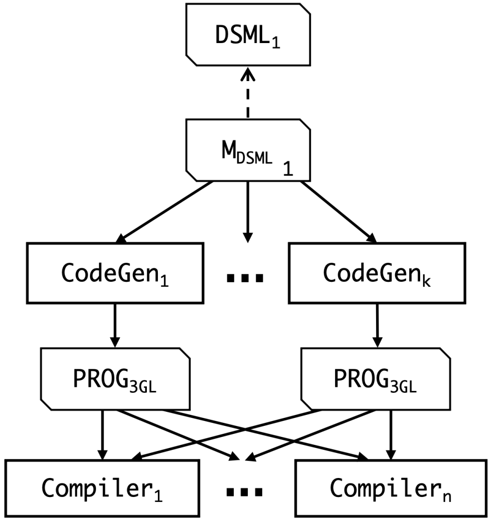


Model compilation



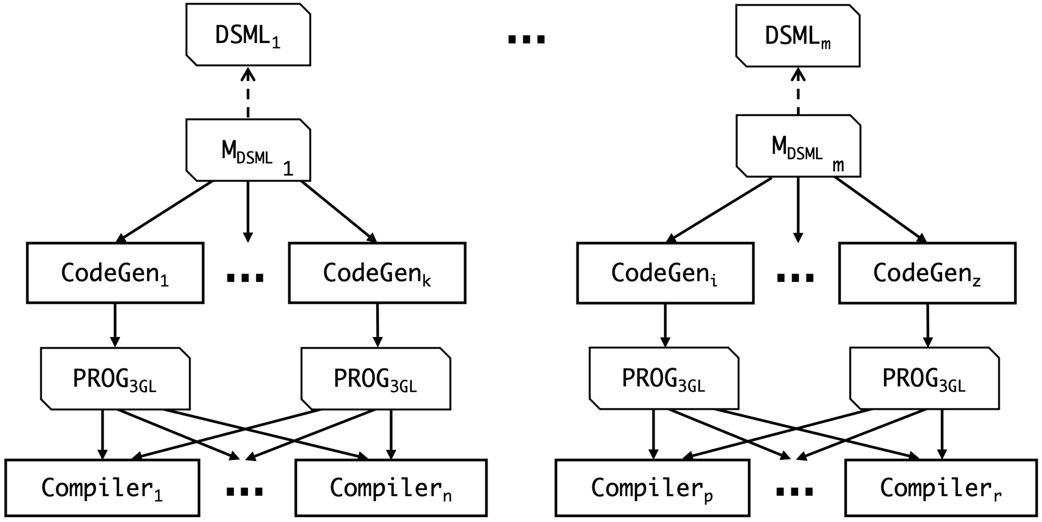
Model compilation

- 3GL code generators are specific to source DSML and target 3GL



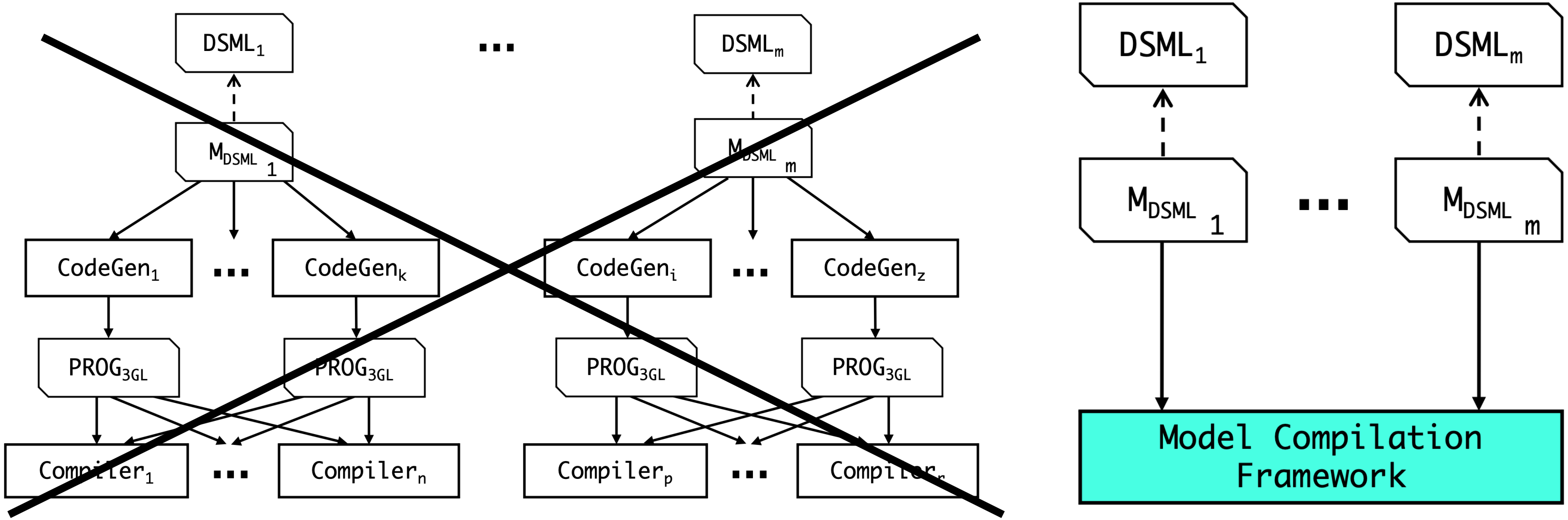
Model compilation

- 3GL code generators are specific to source DSML and target 3GL



Model compilation

- 3GL code generators are specific to source DSML and target 3GL



Model compilation

- Bypass 3GLs and compile models by a flexible compiler framework

Model compilation

- Bypass 3GLs and compile models by a flexible compiler framework
- Semantic anchoring of DSML at hand to compiler IR language

Model compilation

- Bypass 3GLs and compile models by a flexible compiler framework
- Semantic anchoring of DSML at hand to compiler IR language
- Preservation of model semantics in generated executables

Model compilation

- Bypass 3GLs and compile models by a flexible compiler framework
- Semantic anchoring of DSML at hand to compiler IR language
- Preservation of model semantics in generated executables
- Coherent model semantics-based analysis and optimisations

Model compilation

- Bypass 3GLs and compile models by a flexible compiler framework
- Semantic anchoring of DSML at hand to compiler IR language
- Preservation of model semantics in generated executables
- Coherent model semantics-based analysis and optimisations
- Reusability of compiler “lowerings” for same front-ends and back-ends

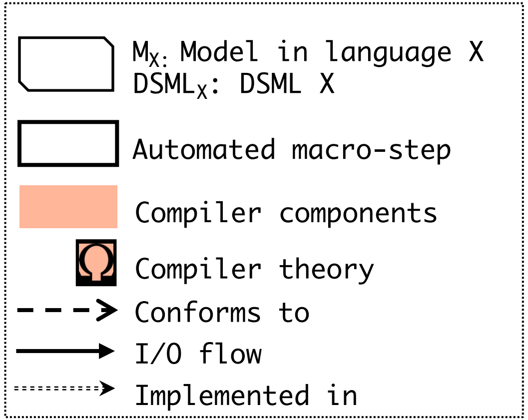
Model compilation

- Bypass 3GLs and compile models by a flexible compiler framework
- Semantic anchoring of DSML at hand to compiler IR language
- Preservation of model semantics in generated executables
- Coherent model semantics-based analysis and optimisations
- Reusability of compiler “lowerings” for same front-ends and back-ends
- Use of AI/ML for compilation purposes (e.g. semantic anchoring)

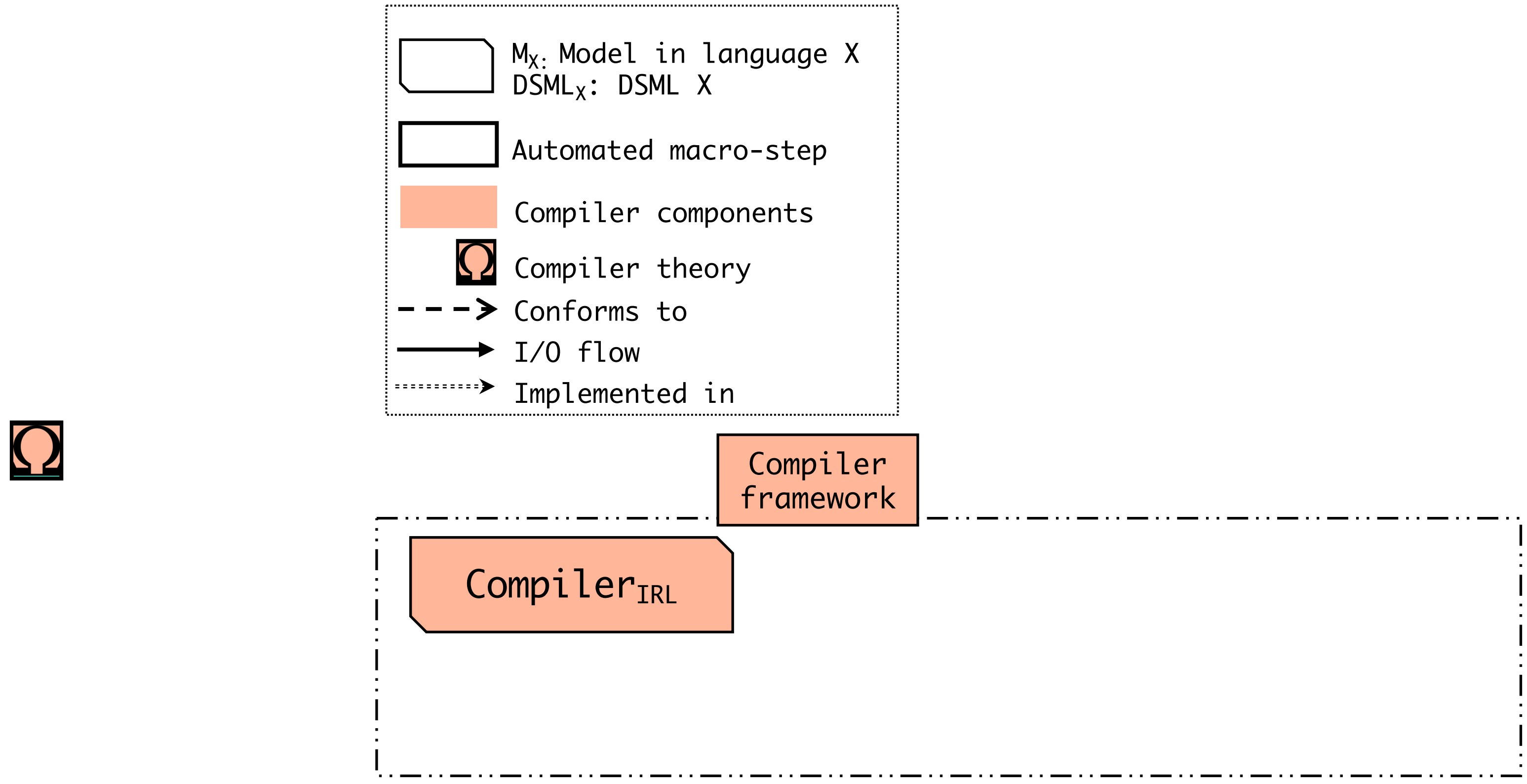
Model compilation

- Bypass 3GLs and compile models by a flexible compiler framework
- Semantic anchoring of DSML at hand to compiler IR language
- Preservation of model semantics in generated executables
- Coherent model semantics-based analysis and optimisations
- Reusability of compiler “lowerings” for same front-ends and back-ends
- Use of AI/ML for compilation purposes (e.g. semantic anchoring)
- Use of model compilation for AI/ML purposes

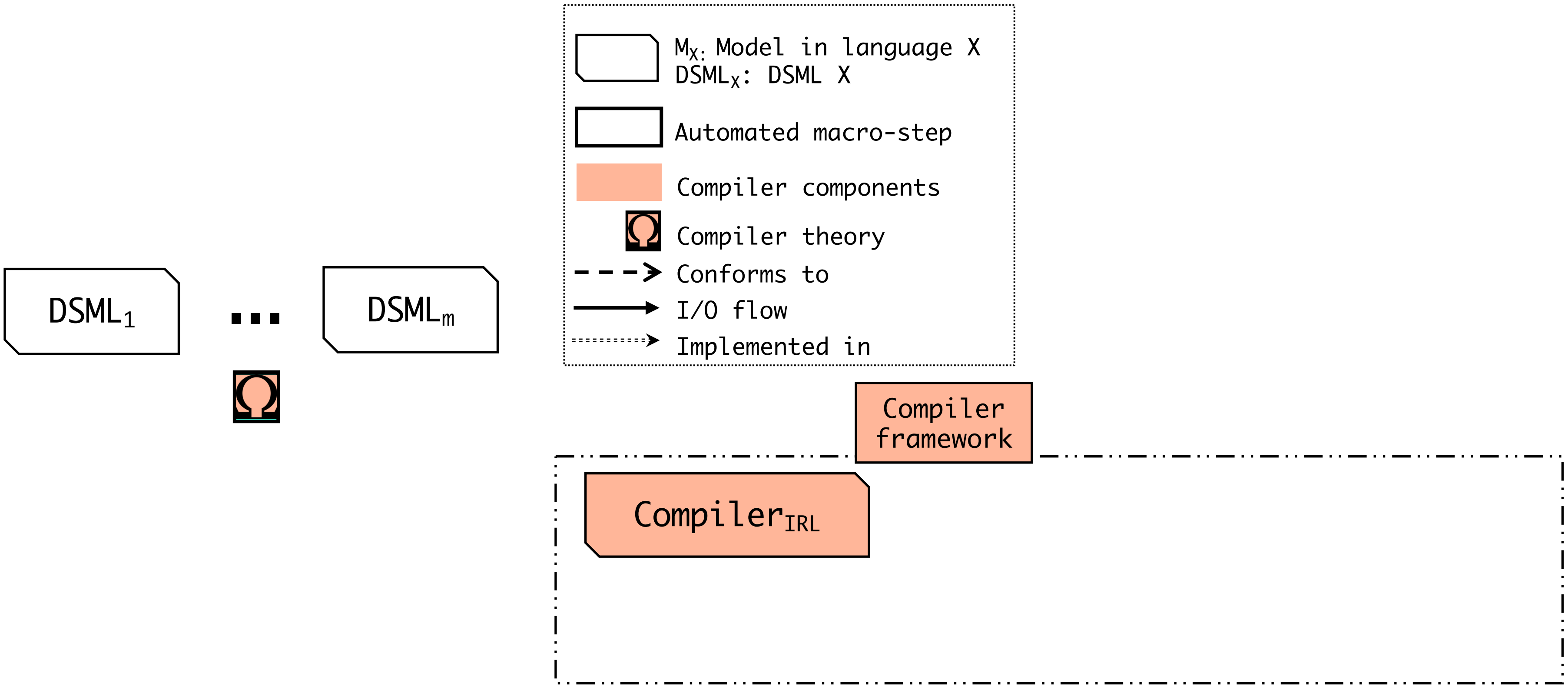
Envisioned model compilation approach



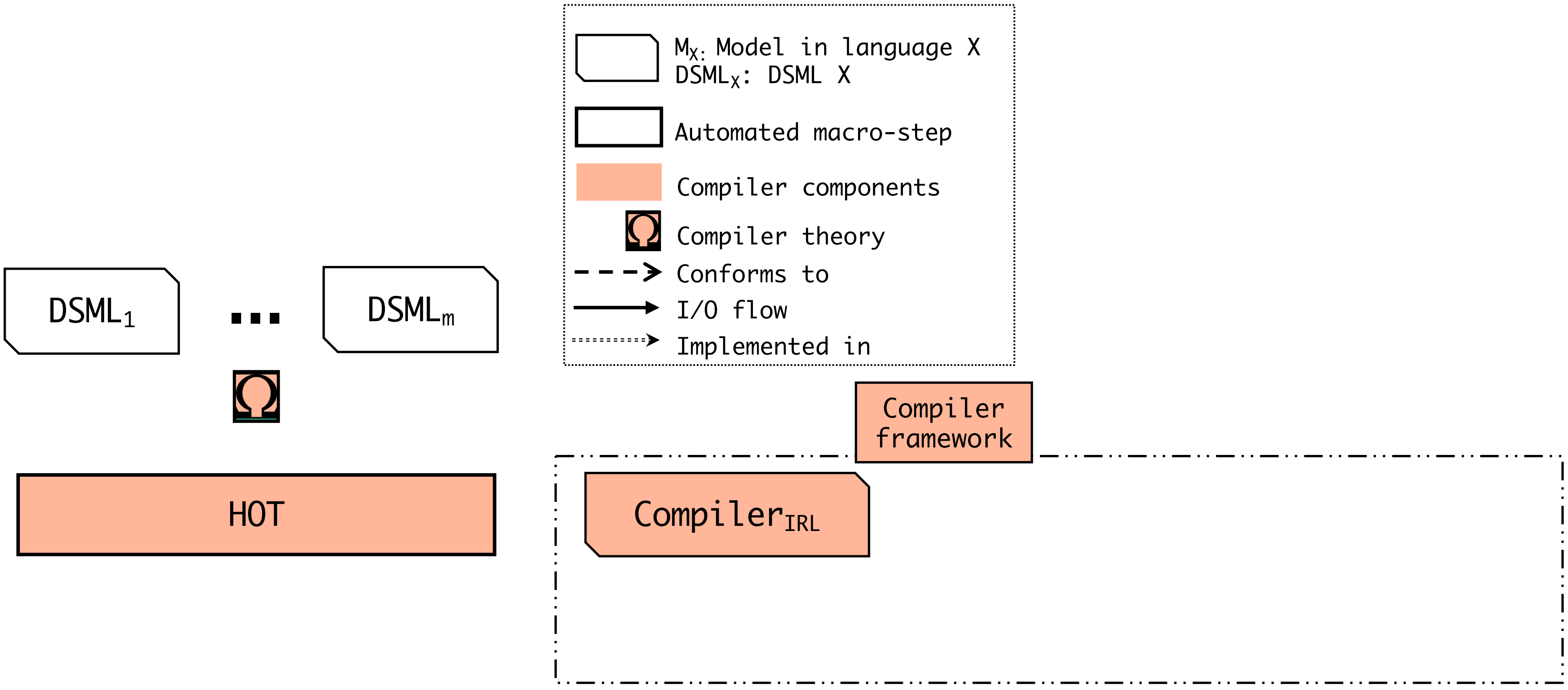
Envisioned model compilation approach



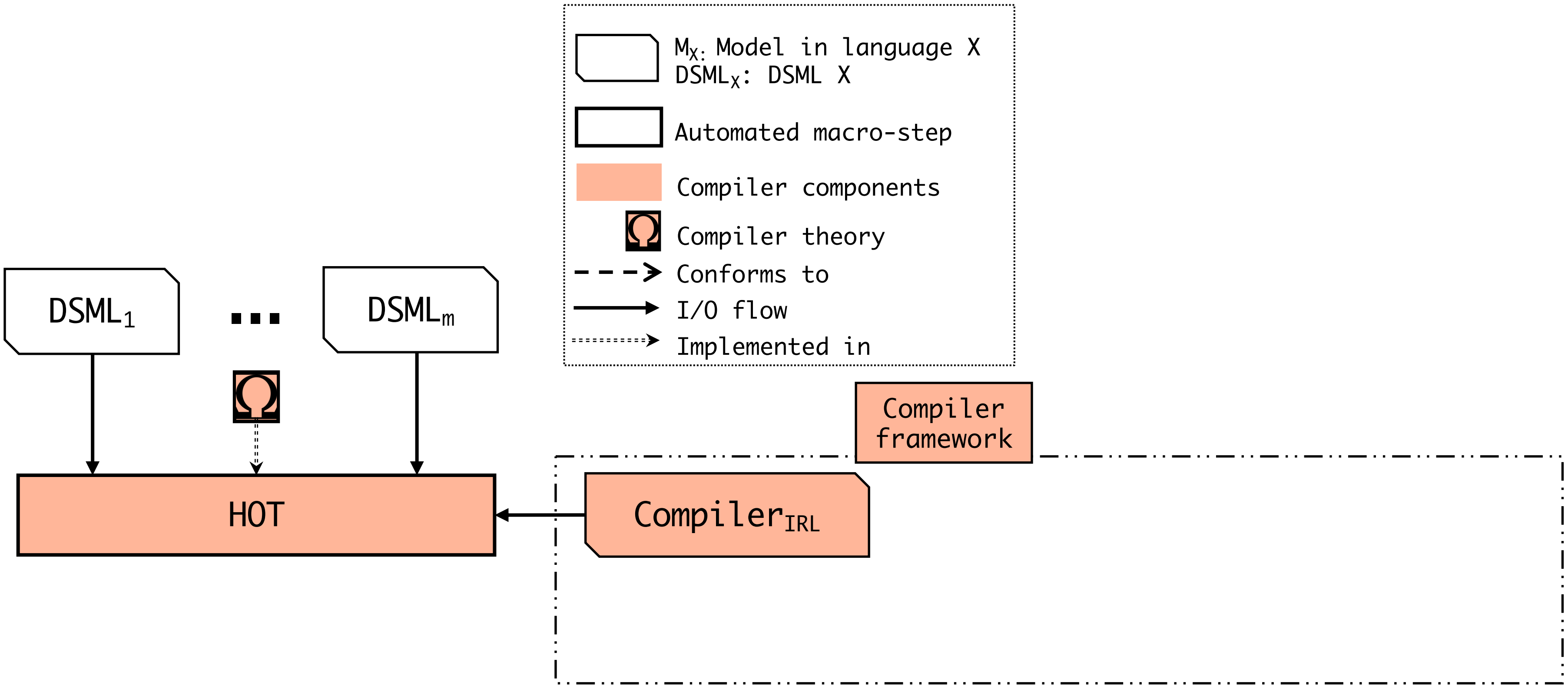
Envisioned model compilation approach



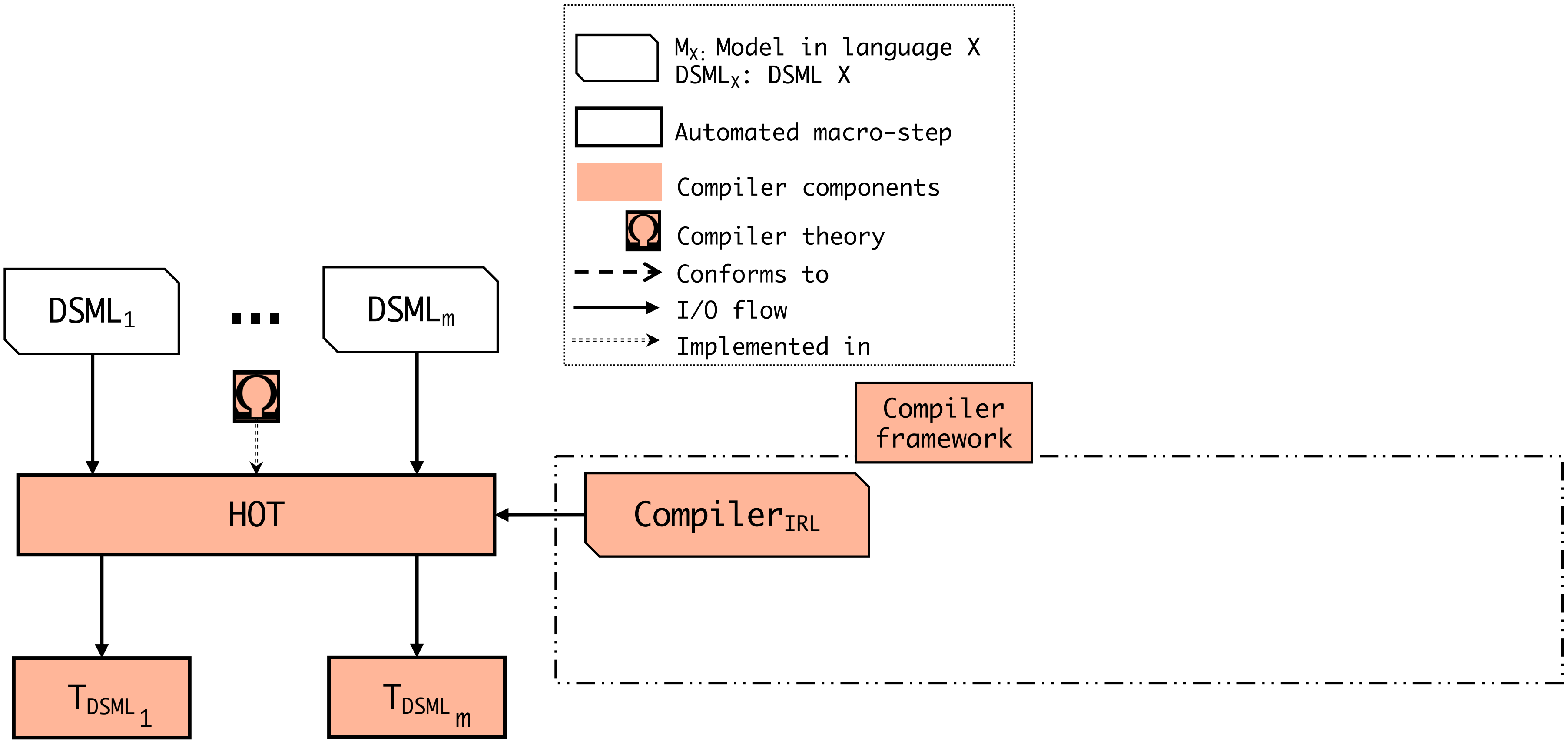
Envisioned model compilation approach



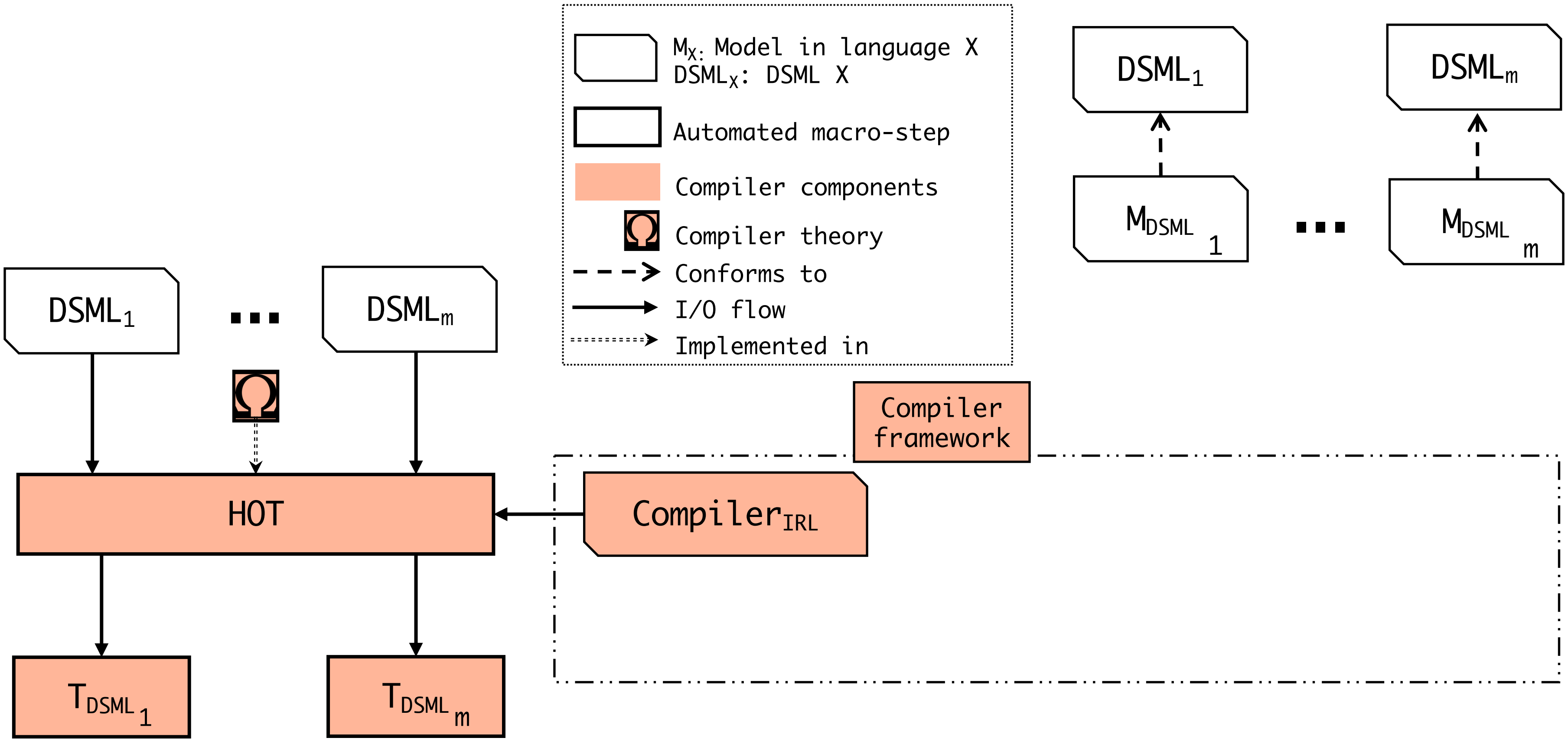
Envisioned model compilation approach



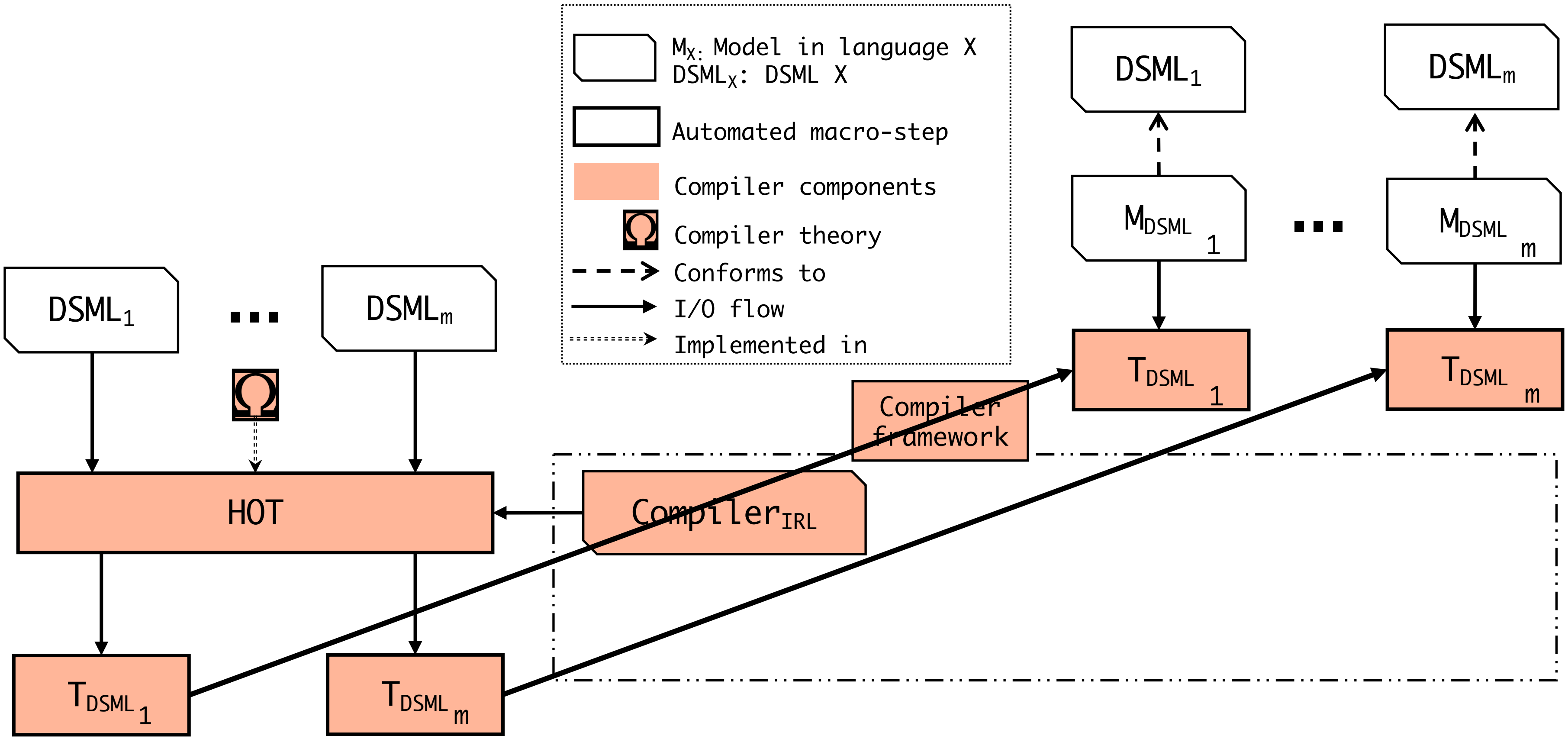
Envisioned model compilation approach



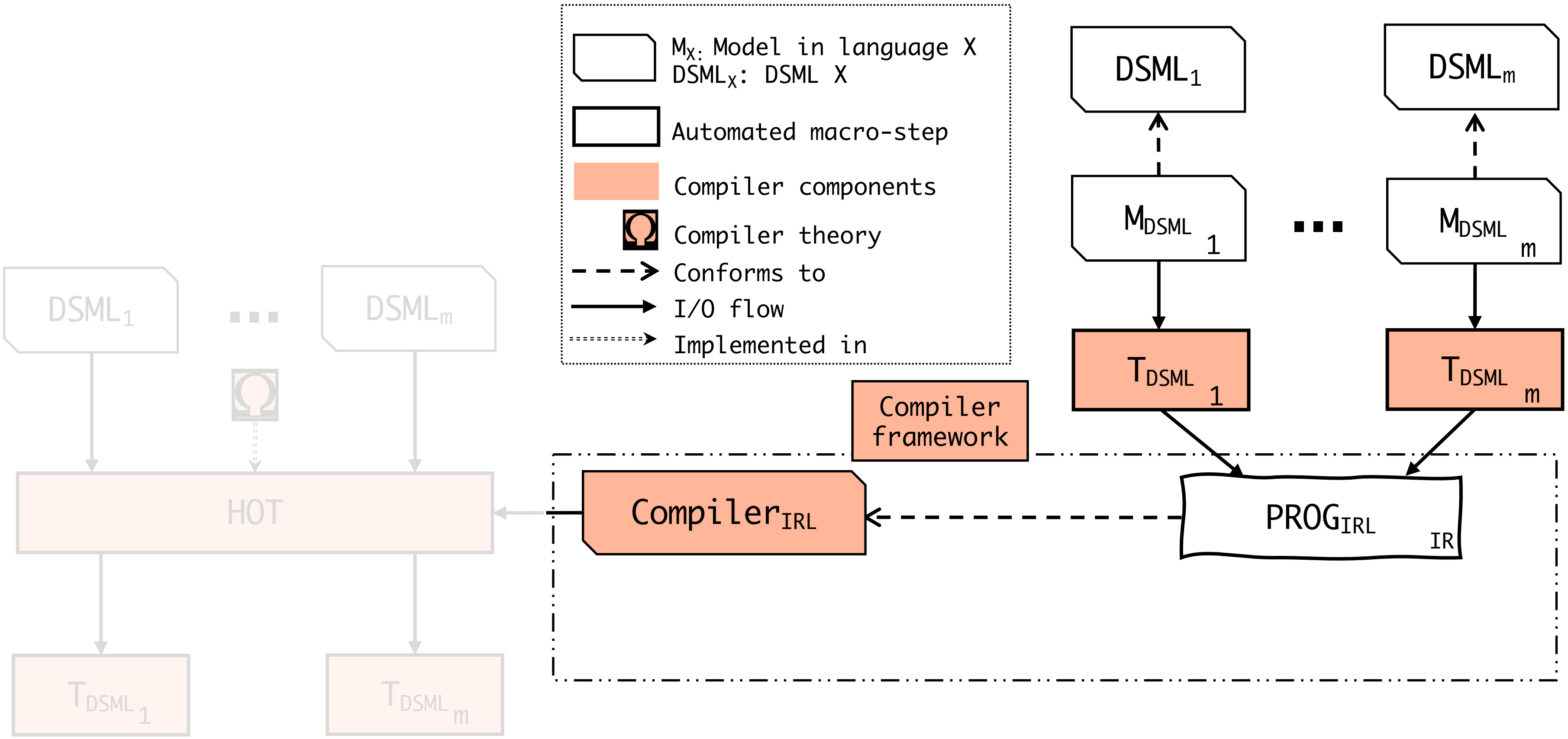
Envisioned model compilation approach



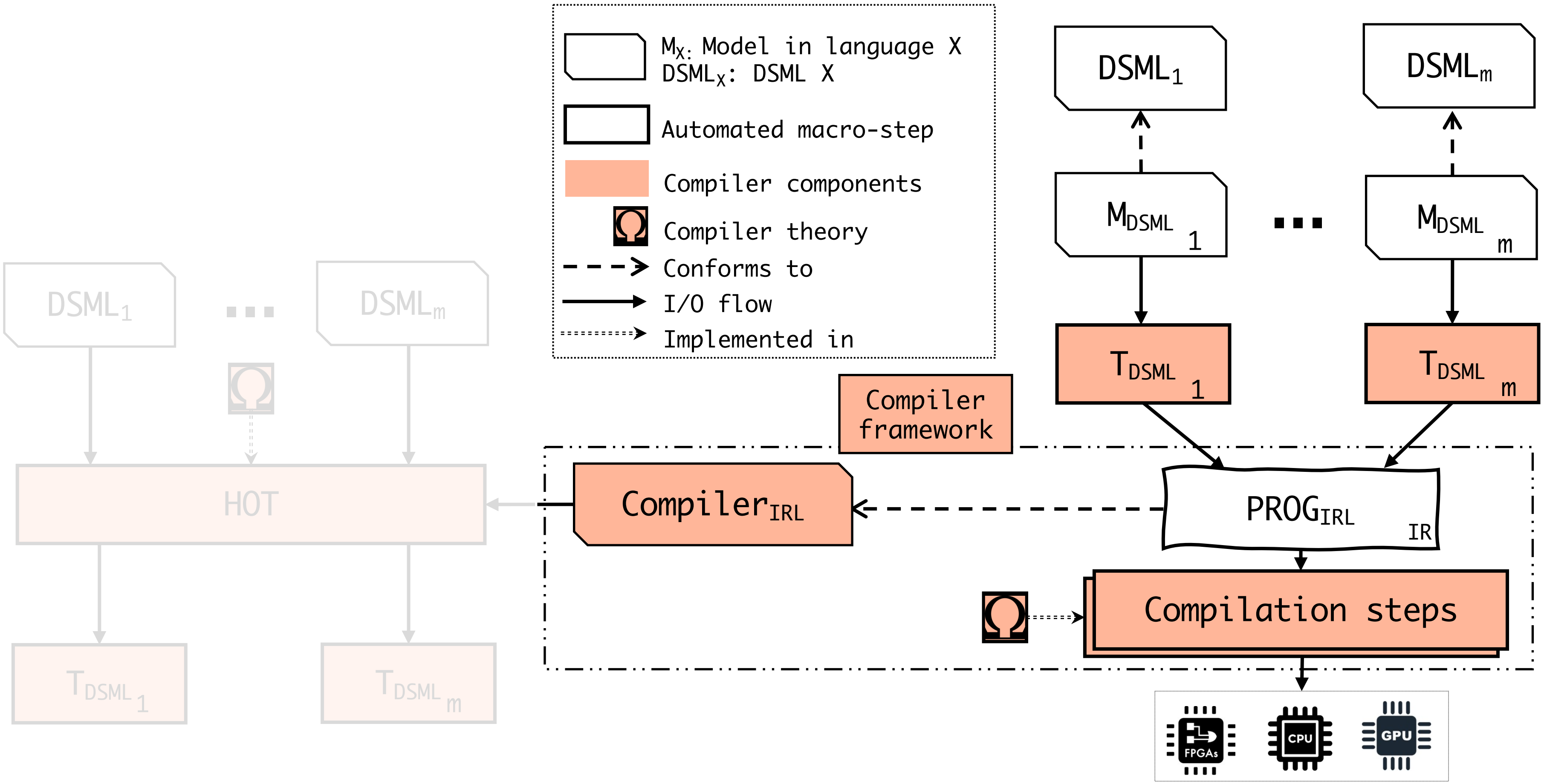
Envisioned model compilation approach



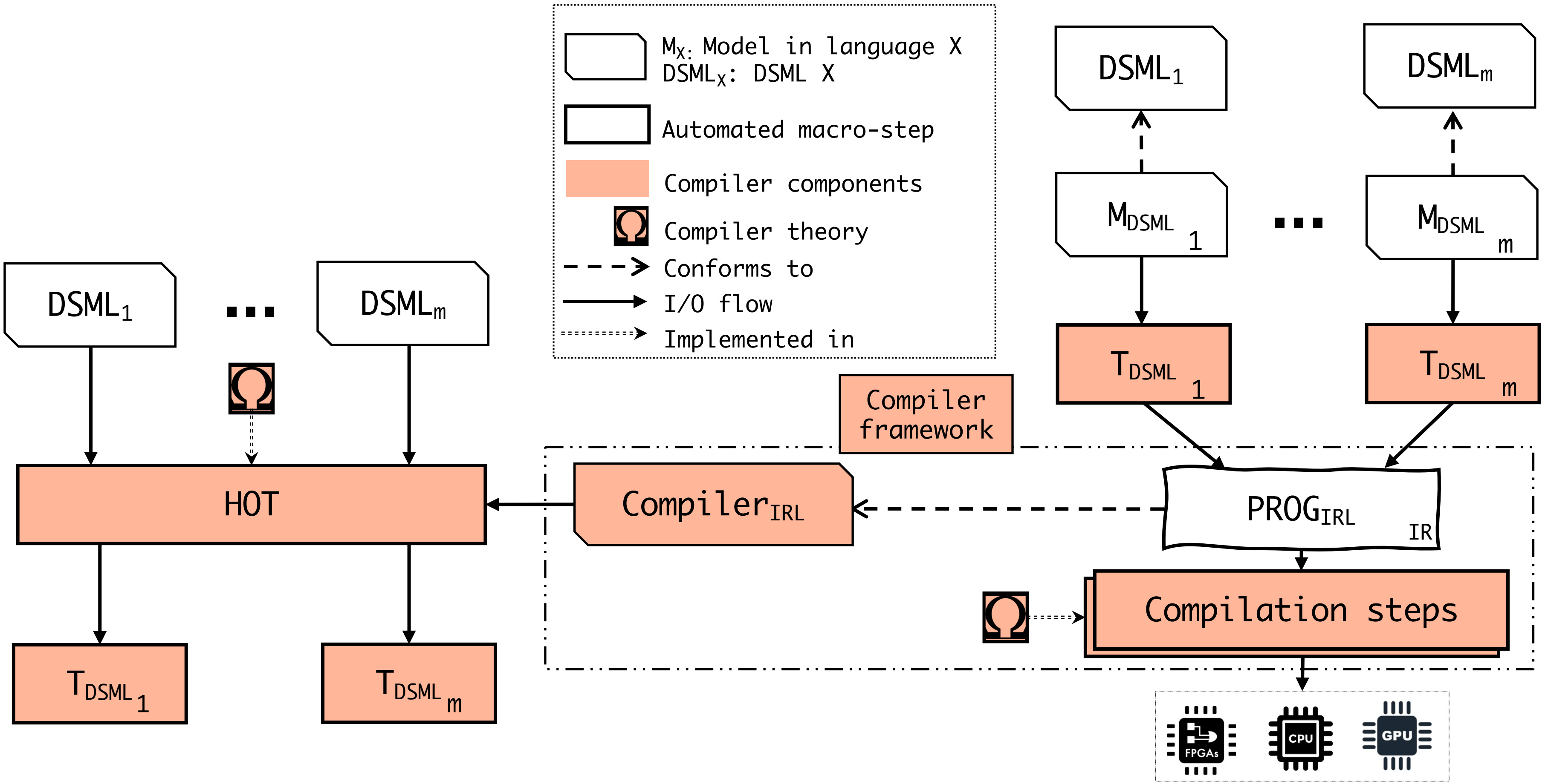
Envisioned model compilation approach



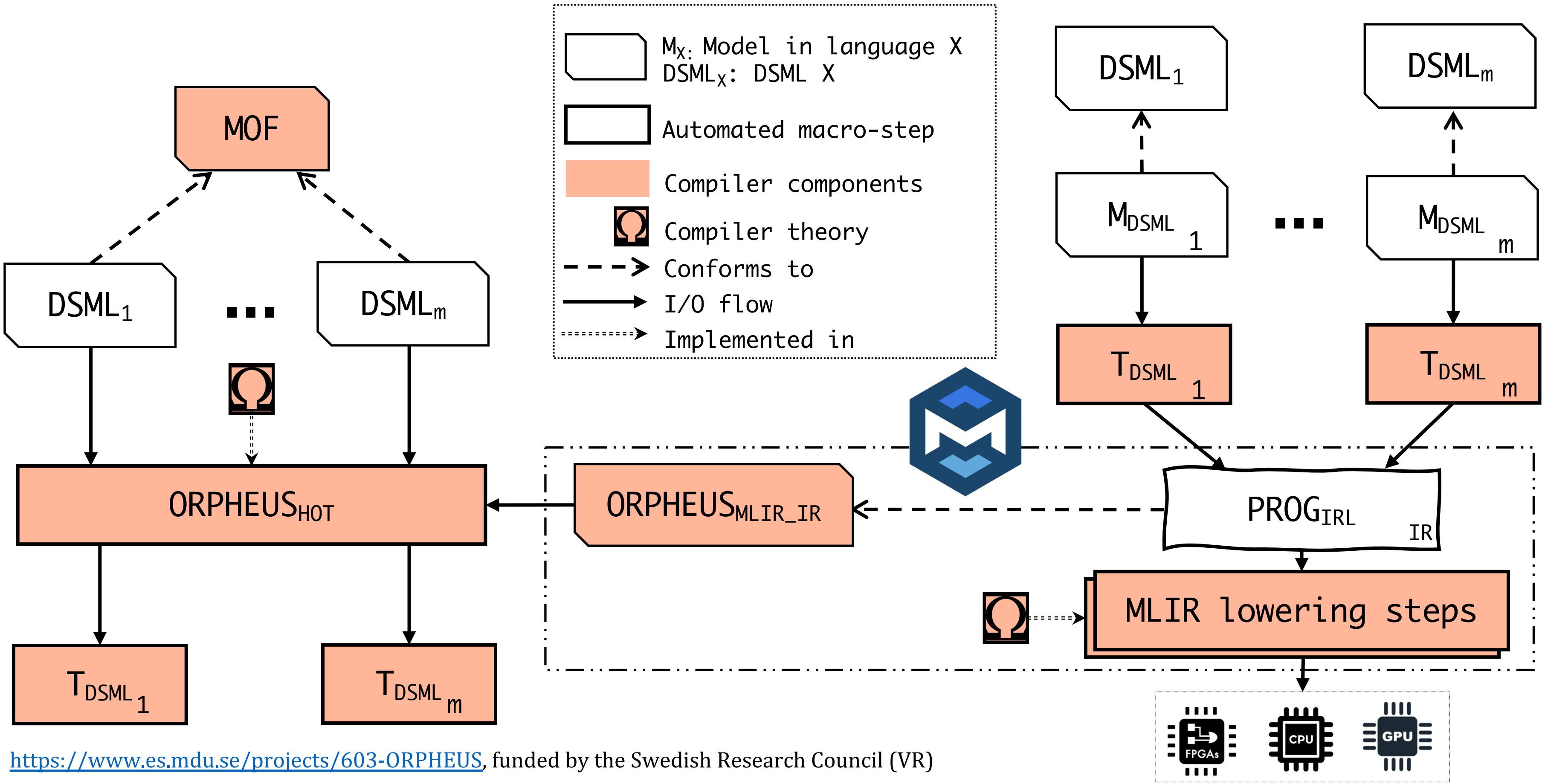
Envisioned model compilation approach



Envisioned model compilation approach



ORPHEUS model compilation approach



https://www.es.mdu.se/projects/603-ORPHEUS, funded by the Swedish Research Council (VR)

A lot to do..

- Ability to execute abstract (high-level) and incomplete models

A lot to do..

- Ability to execute abstract (high-level) and incomplete models
- Observability of executing models

A lot to do..

- Ability to execute abstract (high-level) and incomplete models
- Observability of executing models
- Control of model execution

A lot to do..

- Ability to execute abstract (high-level) and incomplete models
- Observability of executing models
- Control of model execution
- Compilation of DSMLs

A lot to do..

- Ability to execute abstract (high-level) and incomplete models
- Observability of executing models
- Control of model execution
- Compilation of DSMLs
- Integration of model simulation into heterogeneous multi-paradigm simulation systems

A lot to do..

- Ability to execute abstract (high-level) and incomplete models
- Observability of executing models
- Control of model execution
- Compilation of DSMLs
- Integration of model simulation into heterogeneous multi-paradigm simulation systems
- UML model execution
  - Compliance to fUML execution semantics
  - Support for UML-compliant action languages
  - Support for executing models based on UML profiles

.. and..

.. and..

Many of us are *engineers* and..

we tend to bring all back to “*programs*” and “*programming*”,

which is what we know (and often value) the most

.. and..

Many of us are *engineers* and..

we tend to bring all back to “*programs*” and “*programming*”,

which is what we know (and often value) the most

Let’s not forget that **modelling**, for sketching, communication, and brainstorming purposes, is..

.. *fun, useful, and very valuable!*


Design adds value
faster than it adds cost.
Joel Spolsky
quotefancy

Thank you!