Universal Language Server Protocol and Debugger Adapter Protocol for Modular Language Workbenches

Federico Bruzzone

Università degli Studi di Milano
Computer Science Department
PhD Candidate in Computer Science

22/07/2024
Cyclus 40th

Problem Statement
Programming Language Implementation

The implementation of a programming language is a complex task that involves several implementation aspects, such as:

- Syntax and semantics definition
- Type system definition
- Code generation
- Error handling
- IDE support
- Documentation

Problem Statement
Programming Language Implementation

The implementation of a programming language is a complex task that involves several implementation aspects, such as:

- Syntax and semantics definition
- Type system definition
- Code generation
- Error handling
- IDE support
- Documentation

It is usually done in a monolithic way with a top-down approach, where all the aspects are tightly coupled.

Problem Statement
Programming Language Implementation

The implementation of a programming language is a complex task that involves several implementation aspects, such as:

- Syntax and semantics definition
- Type system definition
- Code generation
- Error handling
- IDE support
- Documentation

It is usually done in a monolithic way with a top-down approach, where all the aspects are tightly coupled.

This makes the maintainability, extensibility and reusability of the implementation difficult.

LSP and DAP
In a Nutshell

In 2016, Microsoft in collaboration with Red Hat introduced the Language Server Protocol (LSP) and the Debugger Adapter Protocol (DAP).

LSP and DAP
In a Nutshell

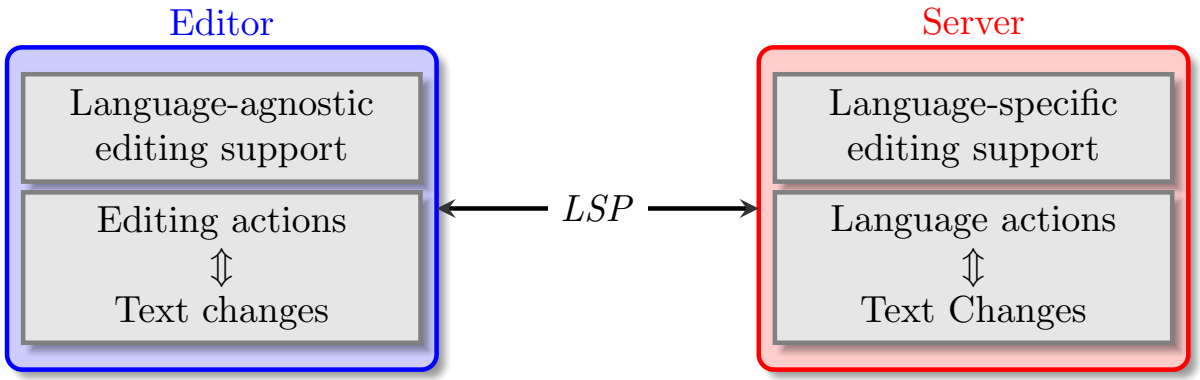
In 2016, Microsoft in collaboration with Red Hat introduced the Language Server Protocol (LSP) and the Debugger Adapter Protocol (DAP).

The LSP and DAP are JSON-RPC based protocols that allow the communication between a Language Server and an IDE.

LSP and DAP
In a Nutshell

In 2016, Microsoft in collaboration with Red Hat introduced the Language Server Protocol (LSP) and the Debugger Adapter Protocol (DAP).

The LSP and DAP are JSON-RPC based protocols that allow the communication between a Language Server and an IDE.



Intrinsic properties:

- Language-agnostic
- IDE-agnostic
- Asynchronous
- Text-based

Features:

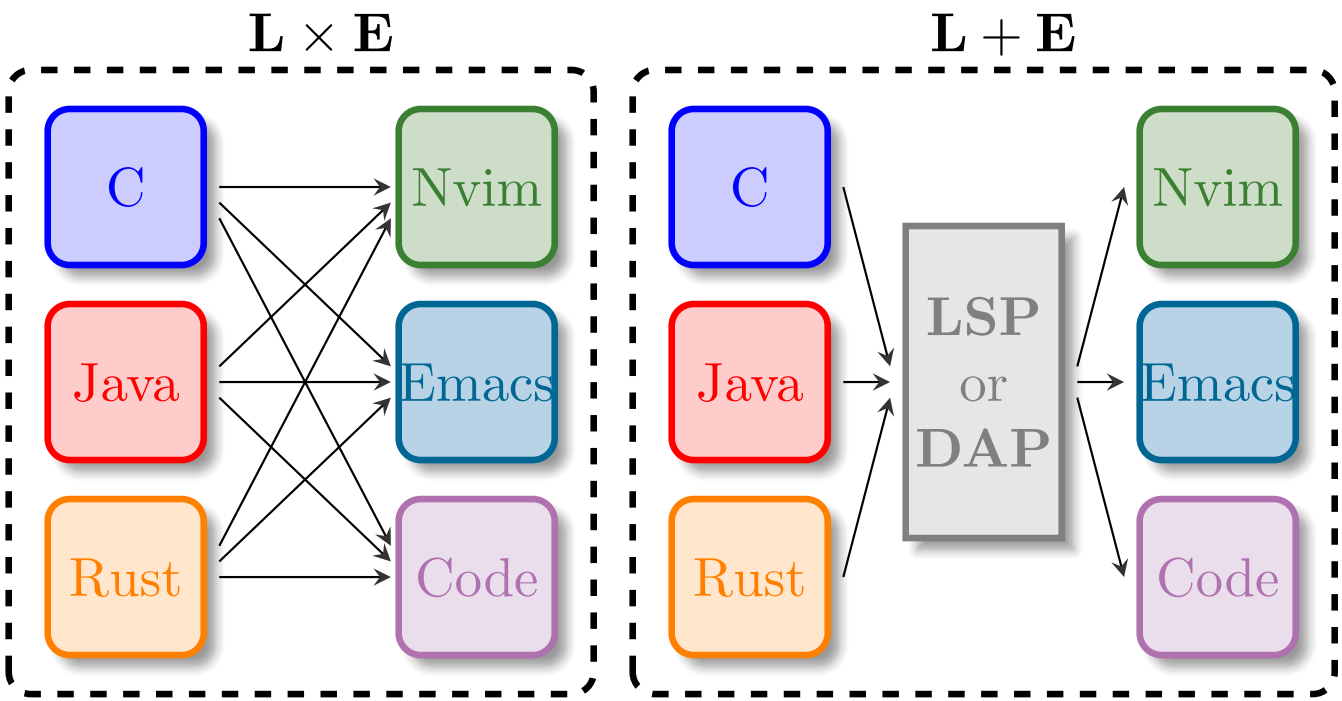
- Diagnostics
- Hover
- Go to definition
- Find references

LSP and DAP
The Reduction of Combinations

Initially implemented for Visual Studio Code, the LSP and DAP have been adopted by several IDEs and programming languages.

LSP and DAP
The Reduction of Combinations

Initially implemented for Visual Studio Code, the LSP and DAP have been adopted by several IDEs and programming languages.

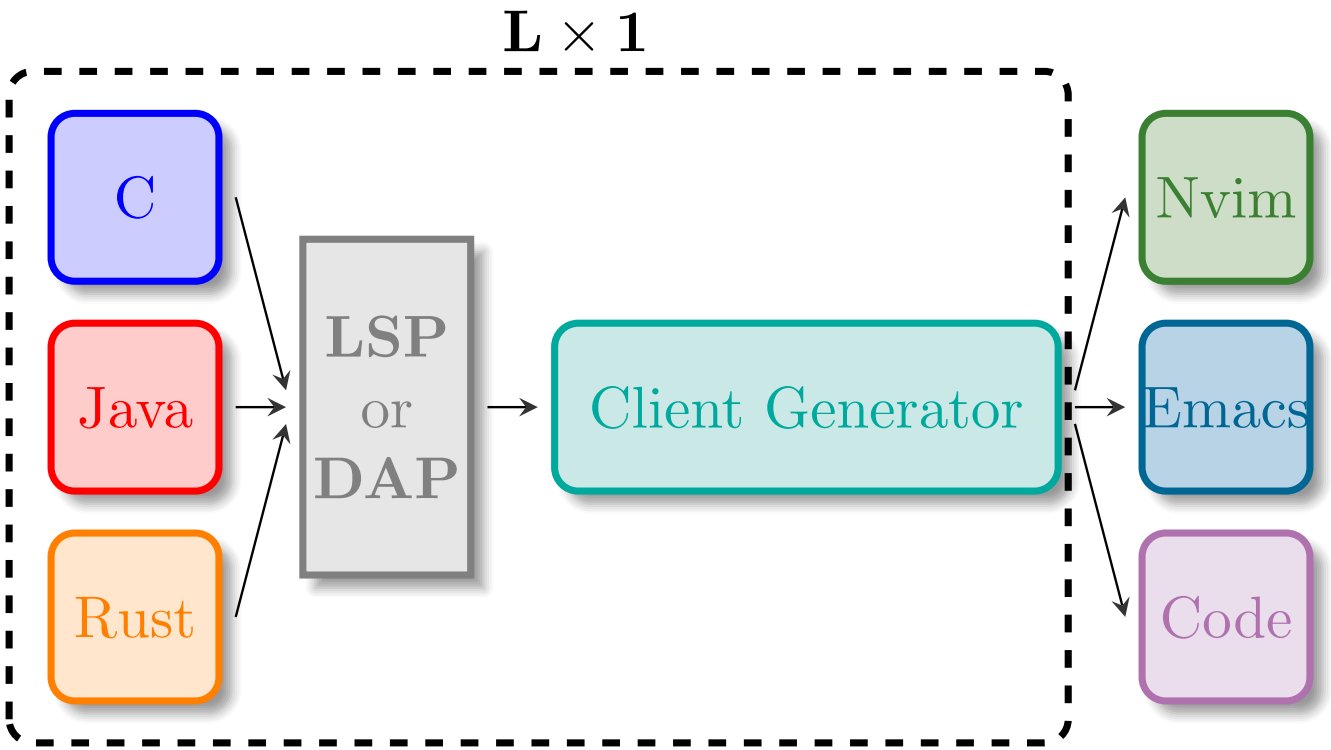


LSP and DAP
What would be an important achievement?

Reducing the number of combinations between Language Servers and IDEs.

LSP and DAP
What would be an important achievement?

Reducing the number of combinations between Language Servers and IDEs.



RO 3: Reduce to **L × 1** the number of combinations to support **L** languages

Feature-Oriented Programming

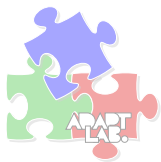
Feature-Oriented Programming (FOP) is a programming paradigm that allows the development of software product lines (SPLs).

Feature-Oriented Programming

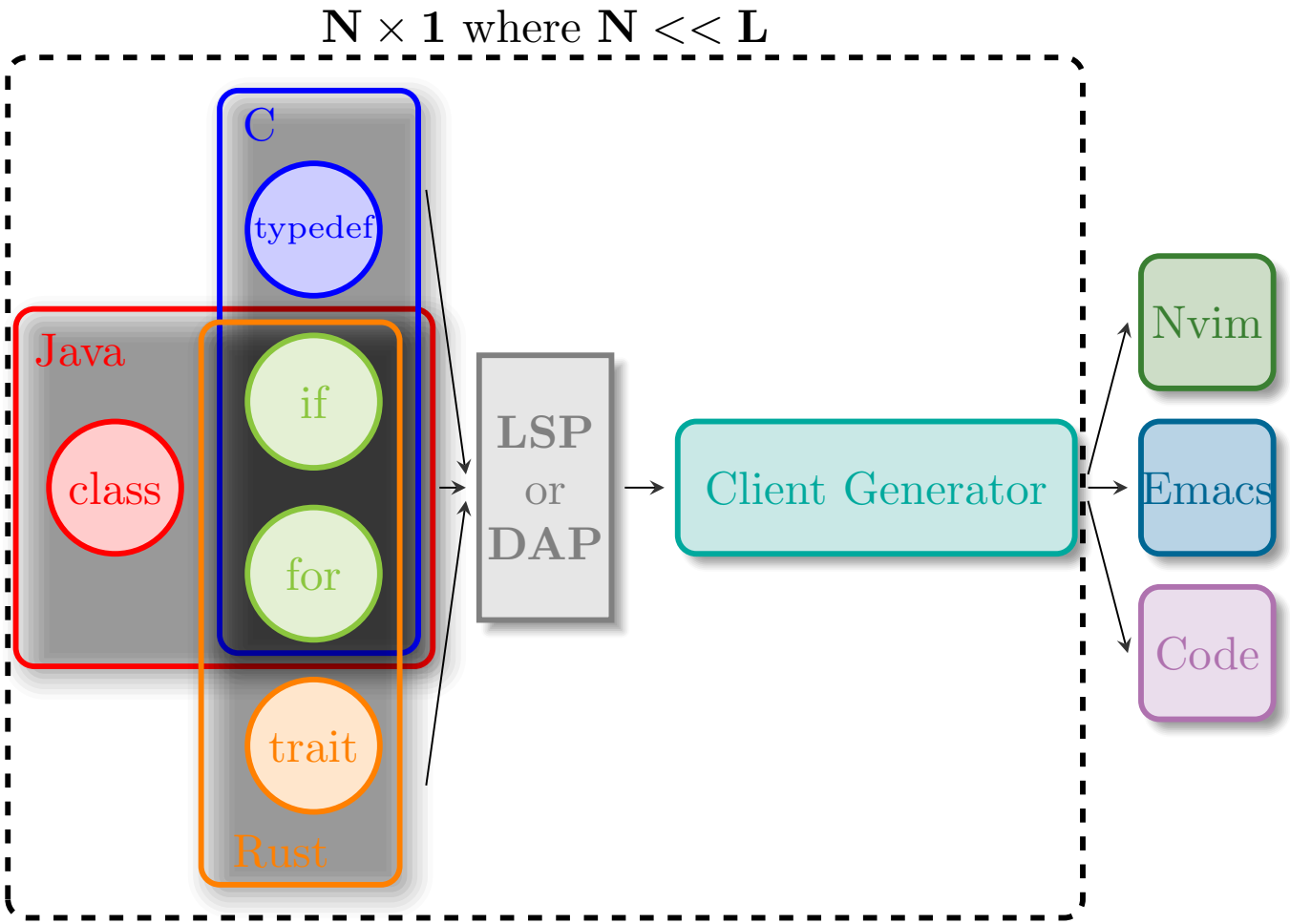
Feature-Oriented Programming (FOP) is a programming paradigm that allows the development of software product lines (SPLs).

- Feature is a unit of functionality that satisfies a requirement.
- Feature Model is a model that represents the variability of the SPL.
- Feature Configuration is a set of features that compose a product.

RO 2: Facilitate LSP and DAP Modularization



Feature-Oriented Programming



Language Workbenches

Language Workbenches (LWs) are tools that allow the development of programming languages, both GPLs and DSLs.

Language Workbench	Modularization Supp.	Precompiled Feature Supp.	Native IDE gen.	LSP & DAP Gen.	LSP & DAP Mod.
JustAdd	◐	○	○	○	○
Melange	◬	○	3rd p.	☆	☆
MontiCore	◐	◐	●	○	○
MPS	◬	○	●	☆	☆
Rascal	○	○	●	○	○
Spoofax	◬	◐	●	☆	☆
Xtext	○	◐	●	●	○
Neverlang	◊	●	○	★	★

- ● Full support
- ○ No support
- ◐ Limited support
- ◊ Fine-grained mod.
- ◬ Coarse-grained mod.
- ★ My expected contribution
- ☆ Extended contribution
- 3rd p. Third-party

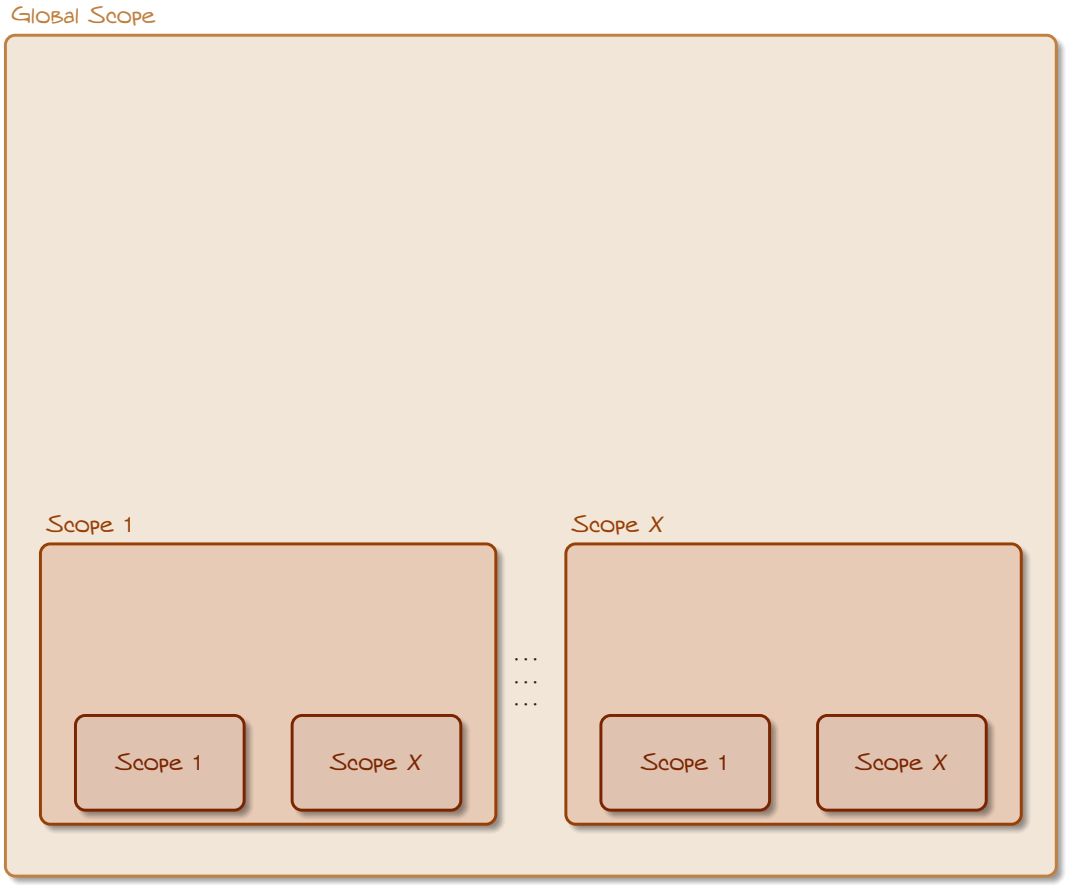
RO 1: Improve IDE and LSP Generation

Scientific Contribution

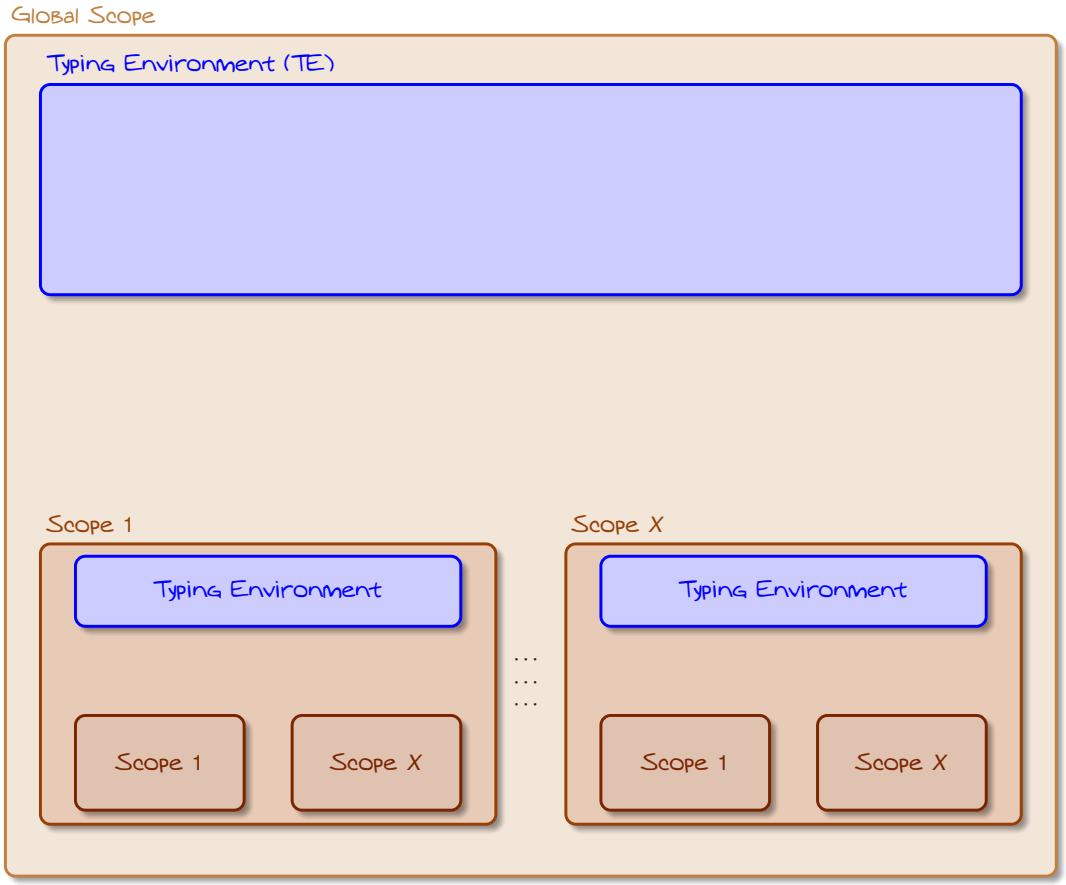
- Methodology for whole LWs that support at least component modularization.
- Type System, LSP and DAP Modularization.
- DSL for Type System definition.
- LSP and DAP generation for Neverlang languages.
- Clients and Syntax Highlighting generation reducing the number of combinations.
- Implementation of a Java Library for Neverlang to support the type system, LSP and DAP for every language developed with Neverlang.
- 3 use cases to show the effectiveness of the methodology.

RO 4: Leverage Neverlang for LSP and DAP in LPL Development

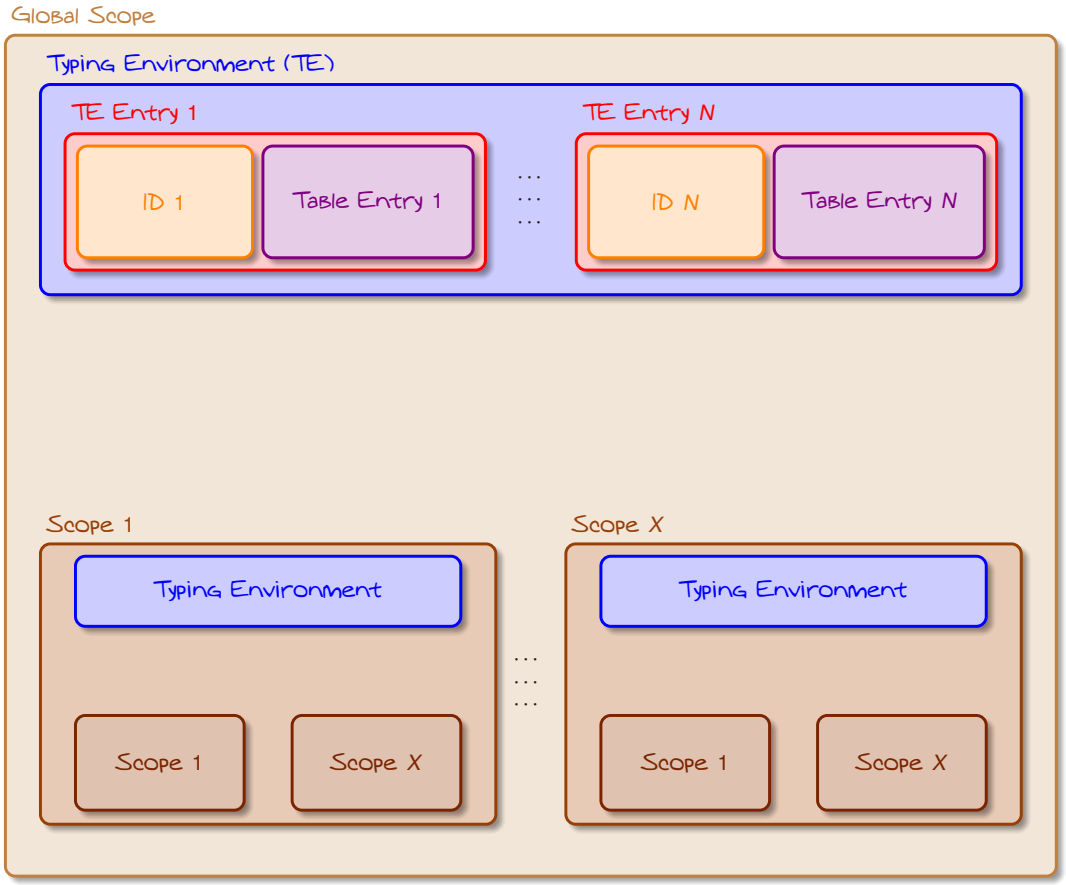
Scientific Contribution
Type System Components



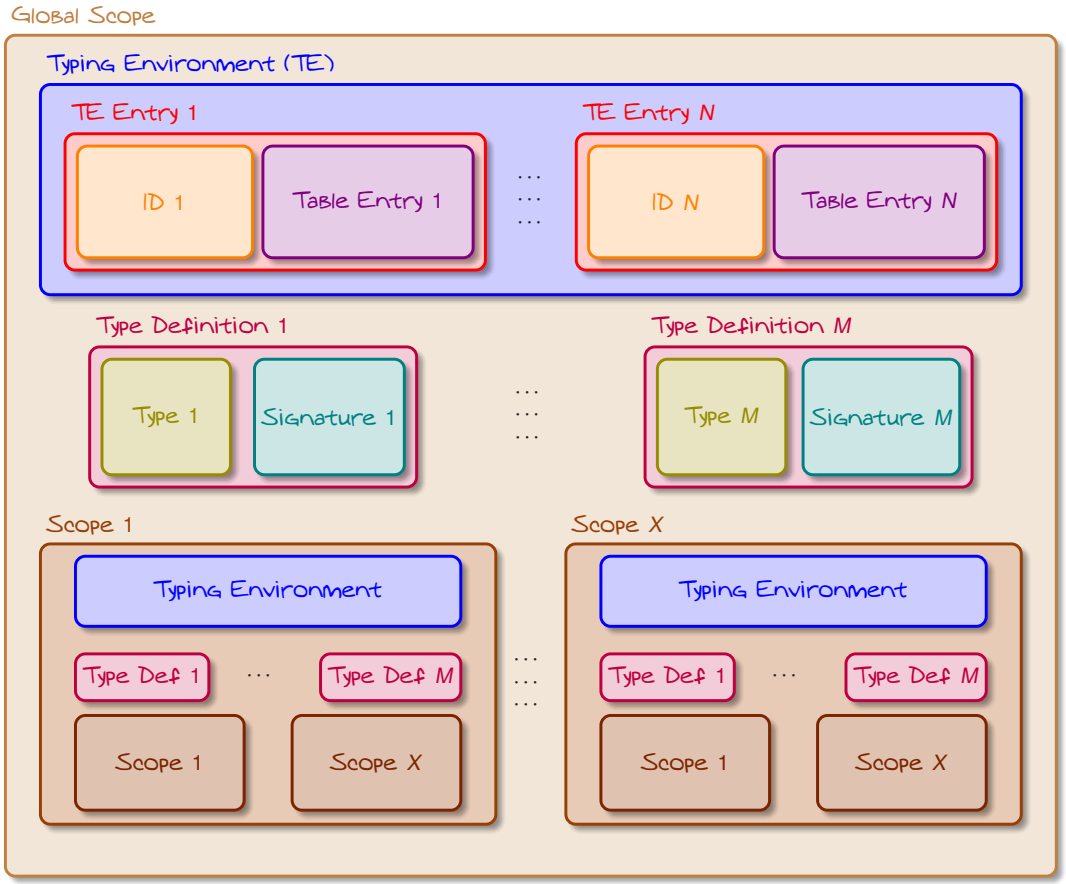
Scientific Contribution
Type System Components



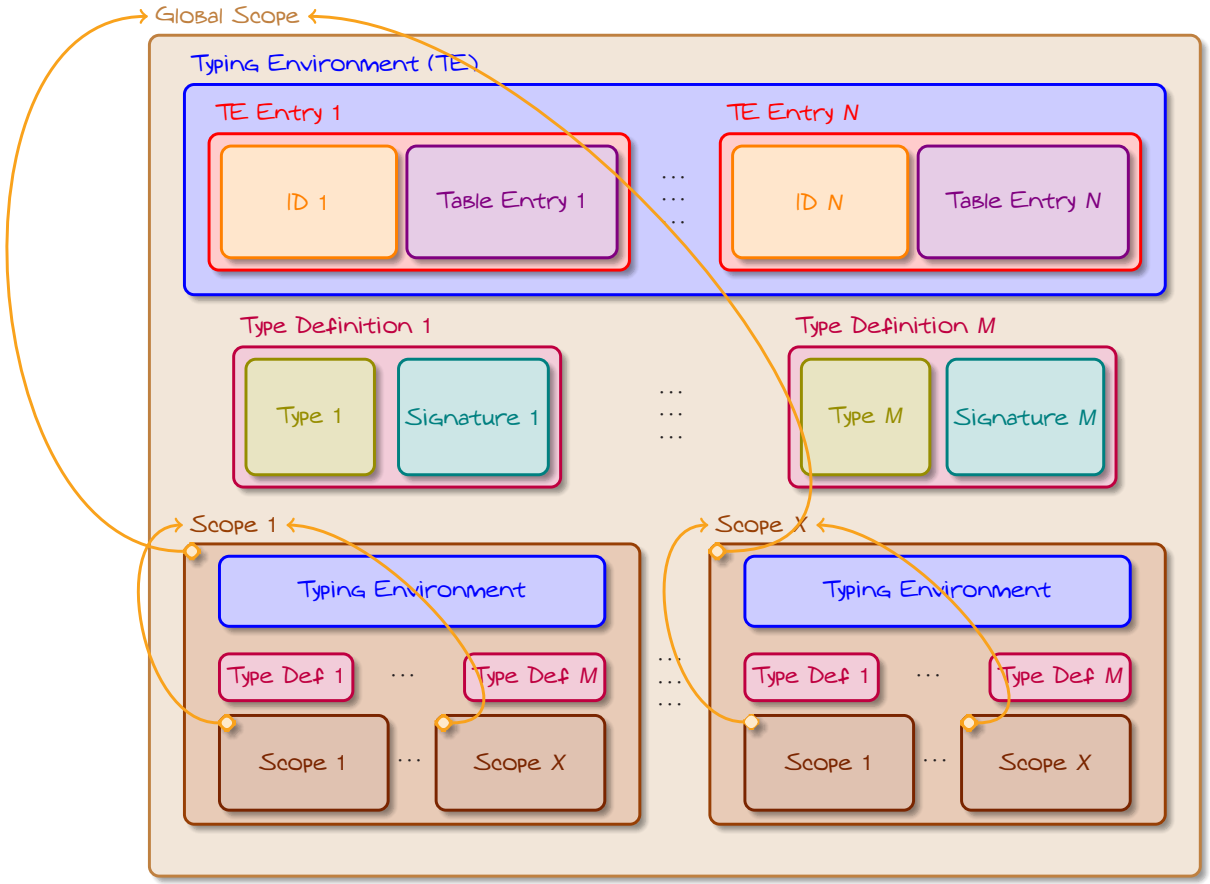
Scientific Contribution
Type System Components



Scientific Contribution
Type System Components

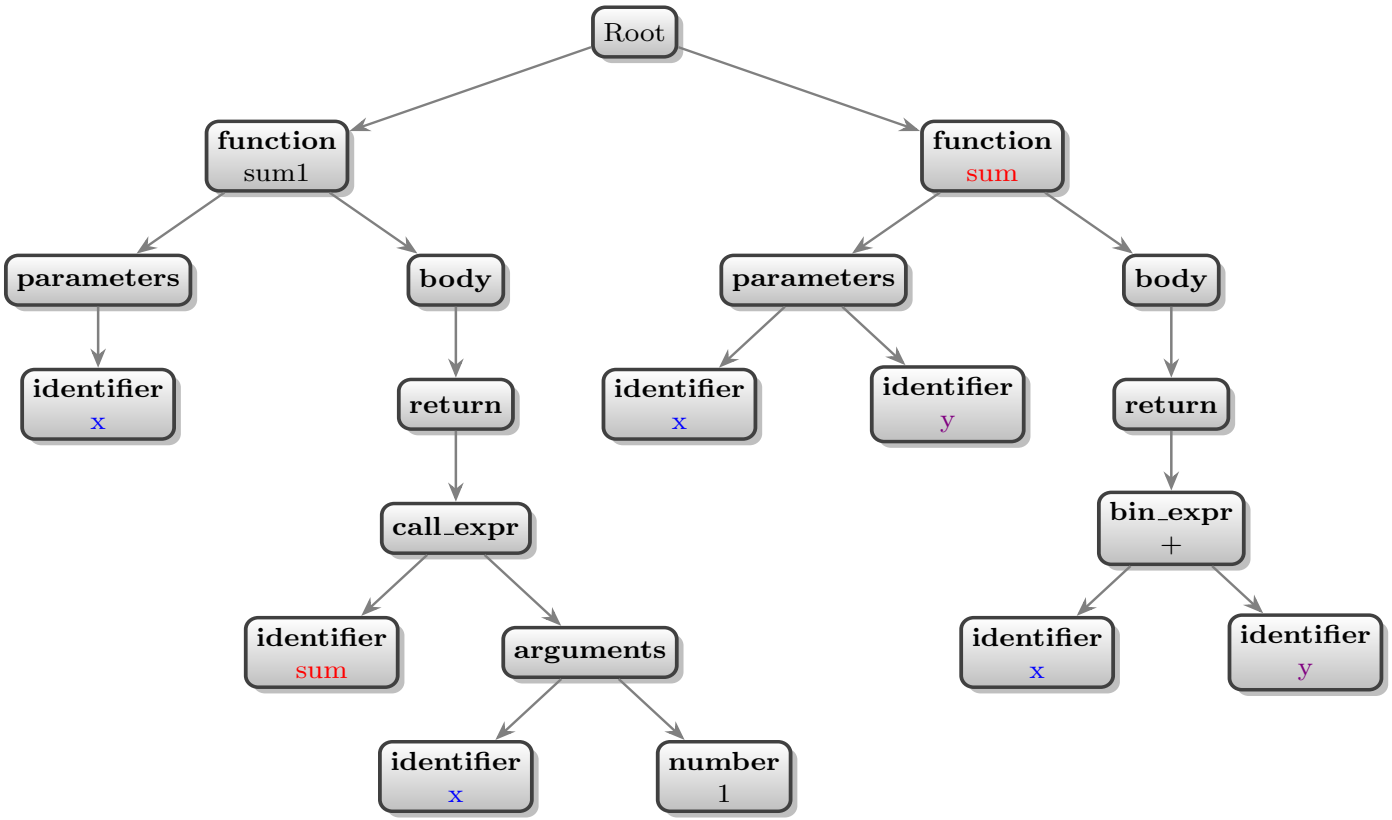


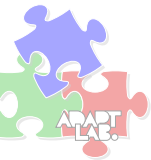
Scientific Contribution
Type System Components



Scientific Contribution
Type Checking and Type Inference


```
1  function sum1(x) {
2      return sum(x, 1);
3  }
5  function sum(x, y) {
6      return x + y;
7  }
```



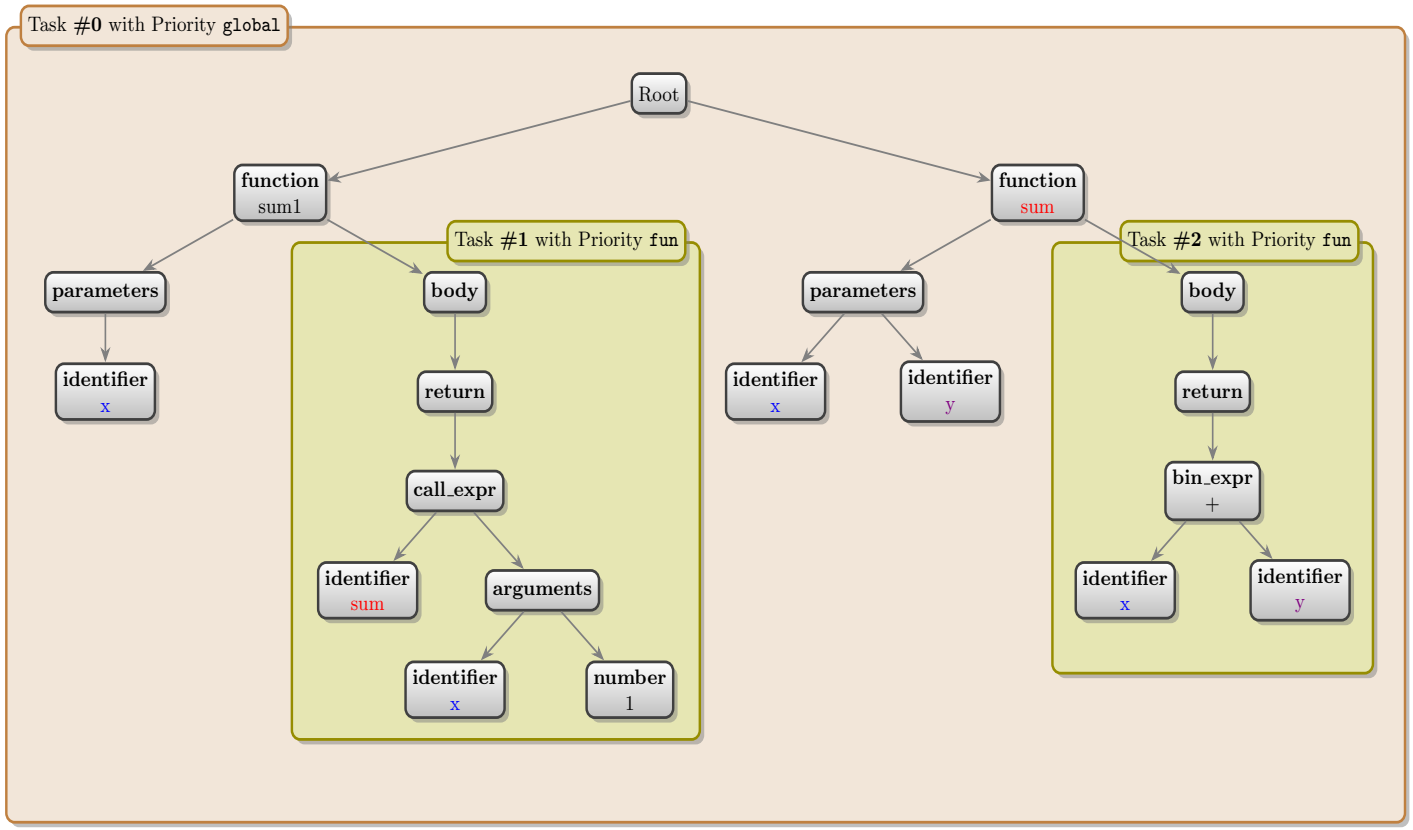


Scientific Contribution

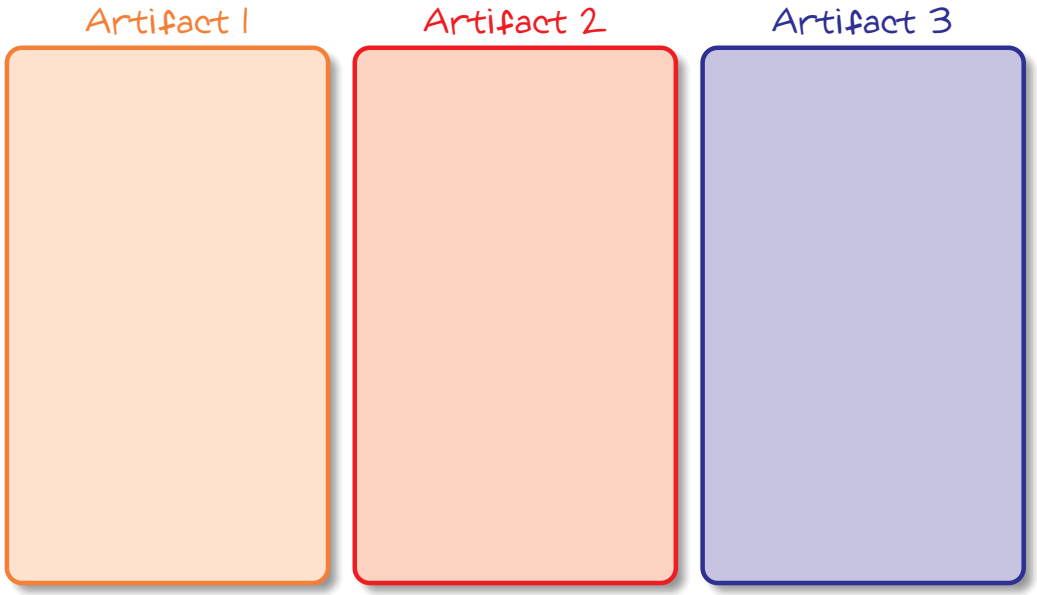
Type Checking and Type Inference

```
1  function sum1(x) {
2      return sum(x, 1);
3  }
5  function sum(x, y) {
6      return x + y;
7  }
```

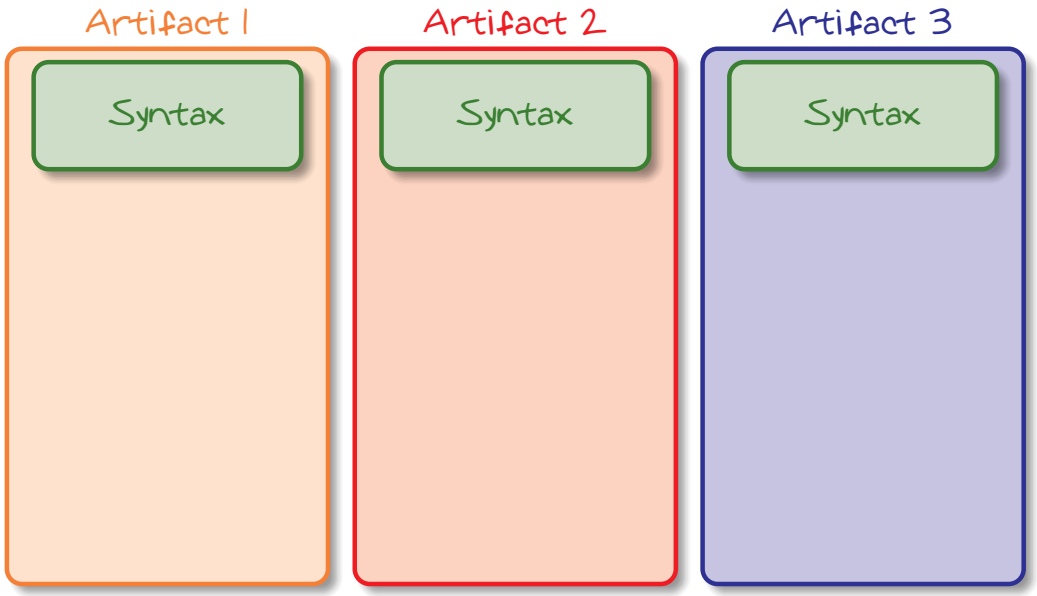
- Compilation Unit
- Compilation Unit Task
- Compilation Helper



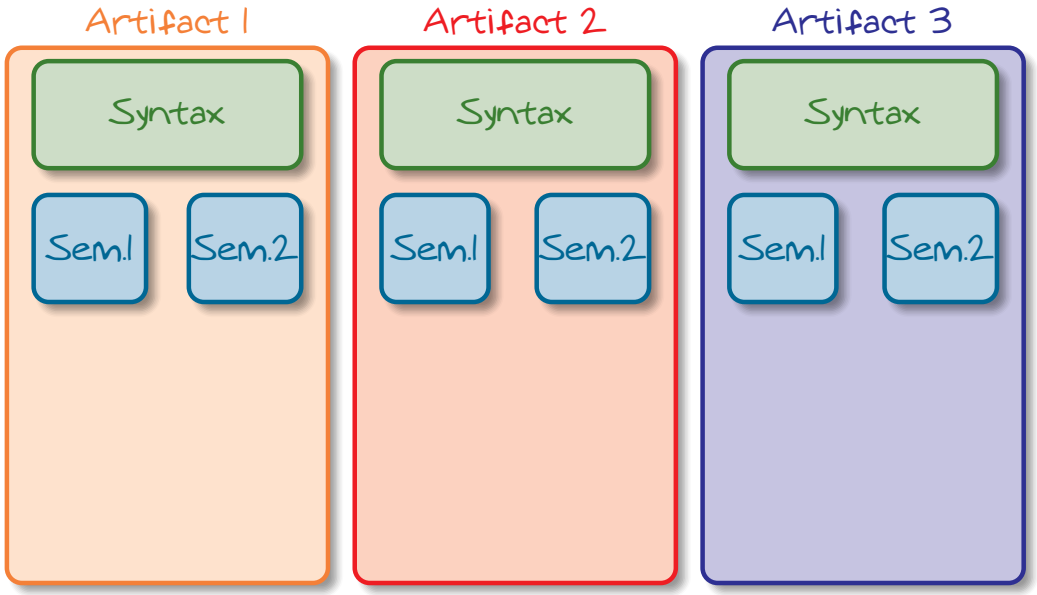
Scientific Contribution
TSs, LSP and DAP Modularization



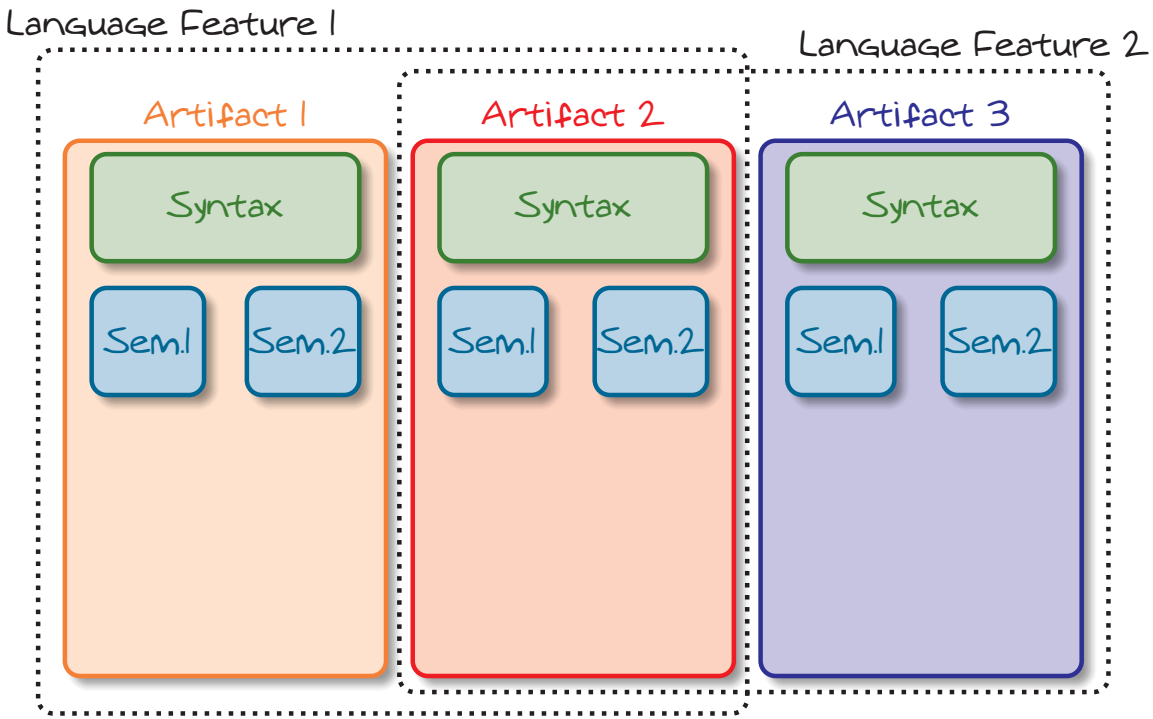
Scientific Contribution
TSs, LSP and DAP Modularization



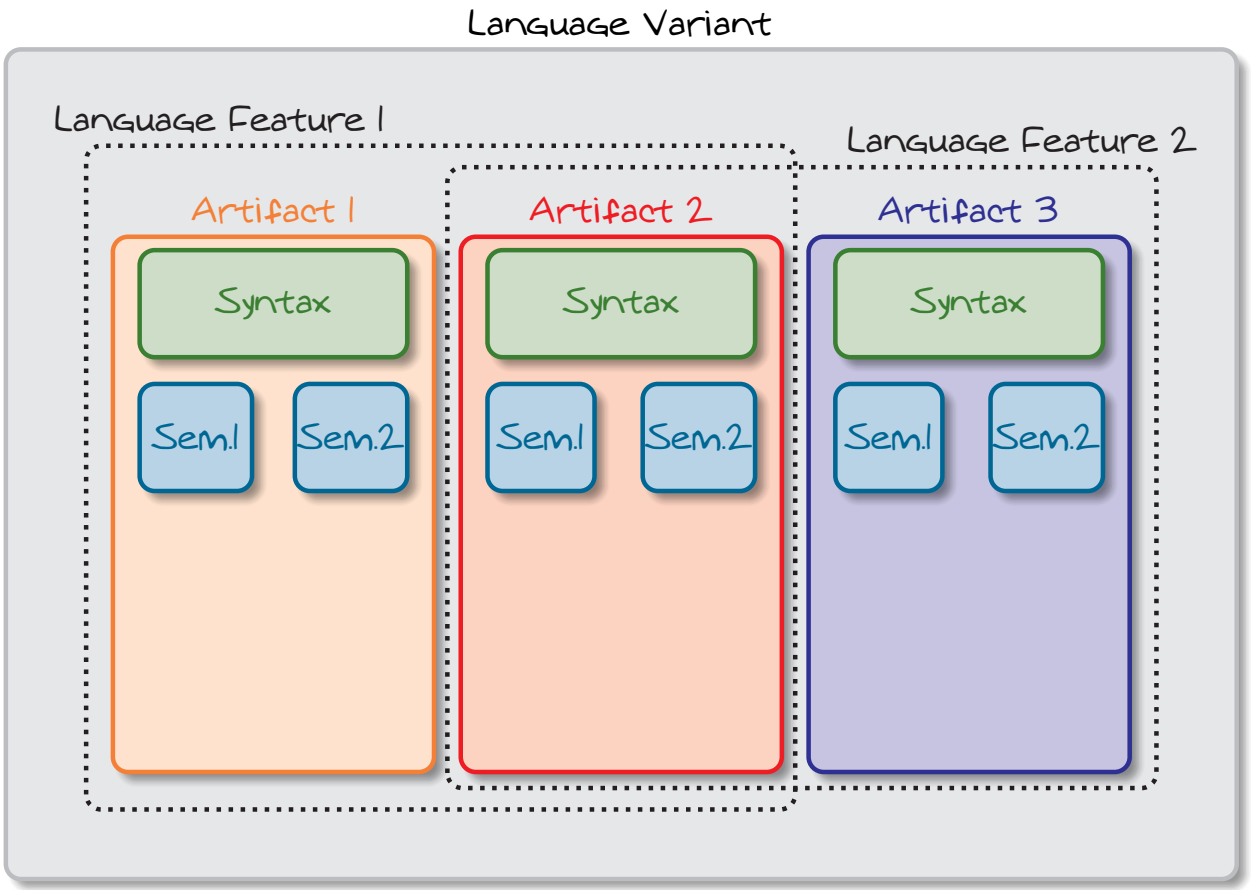
Scientific Contribution
TSs, LSP and DAP Modularization



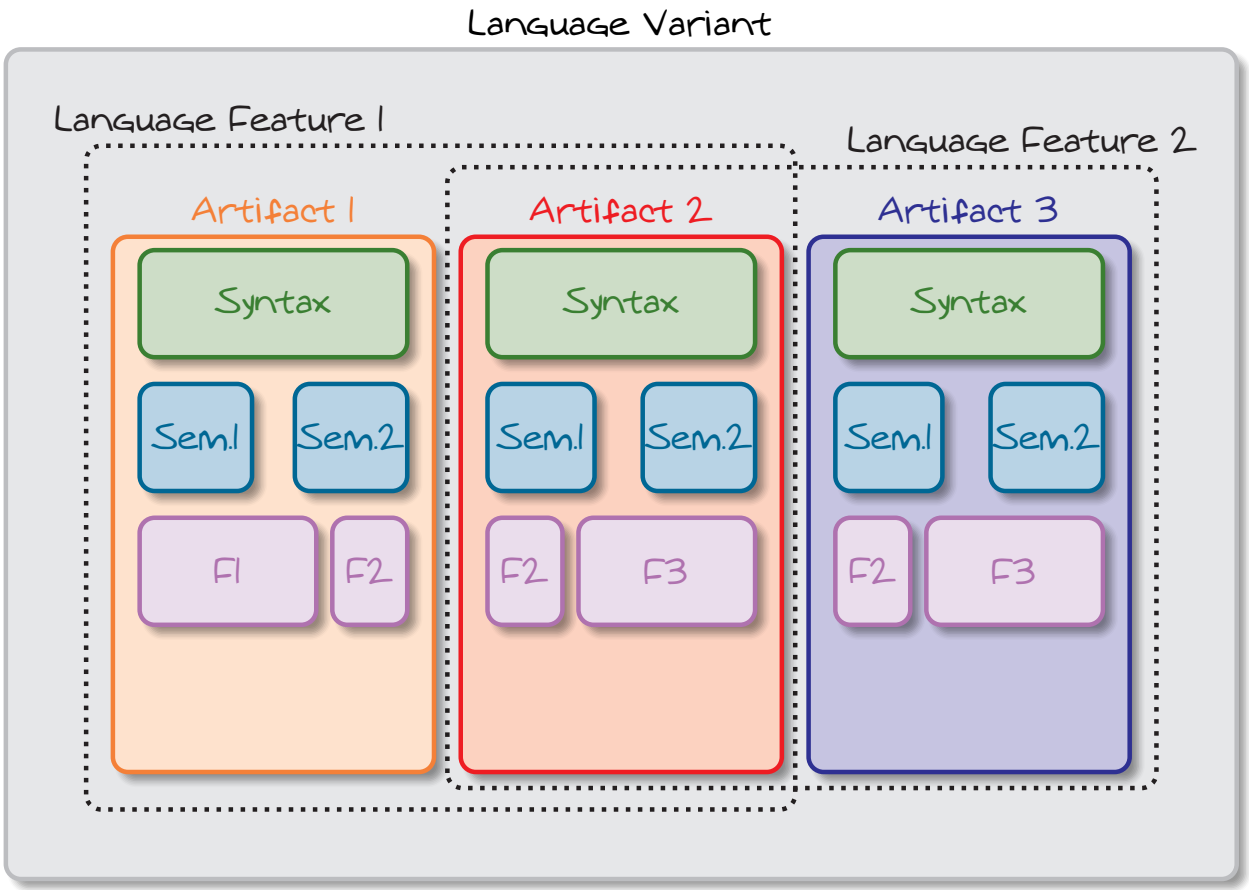
Scientific Contribution
TSs, LSP and DAP Modularization



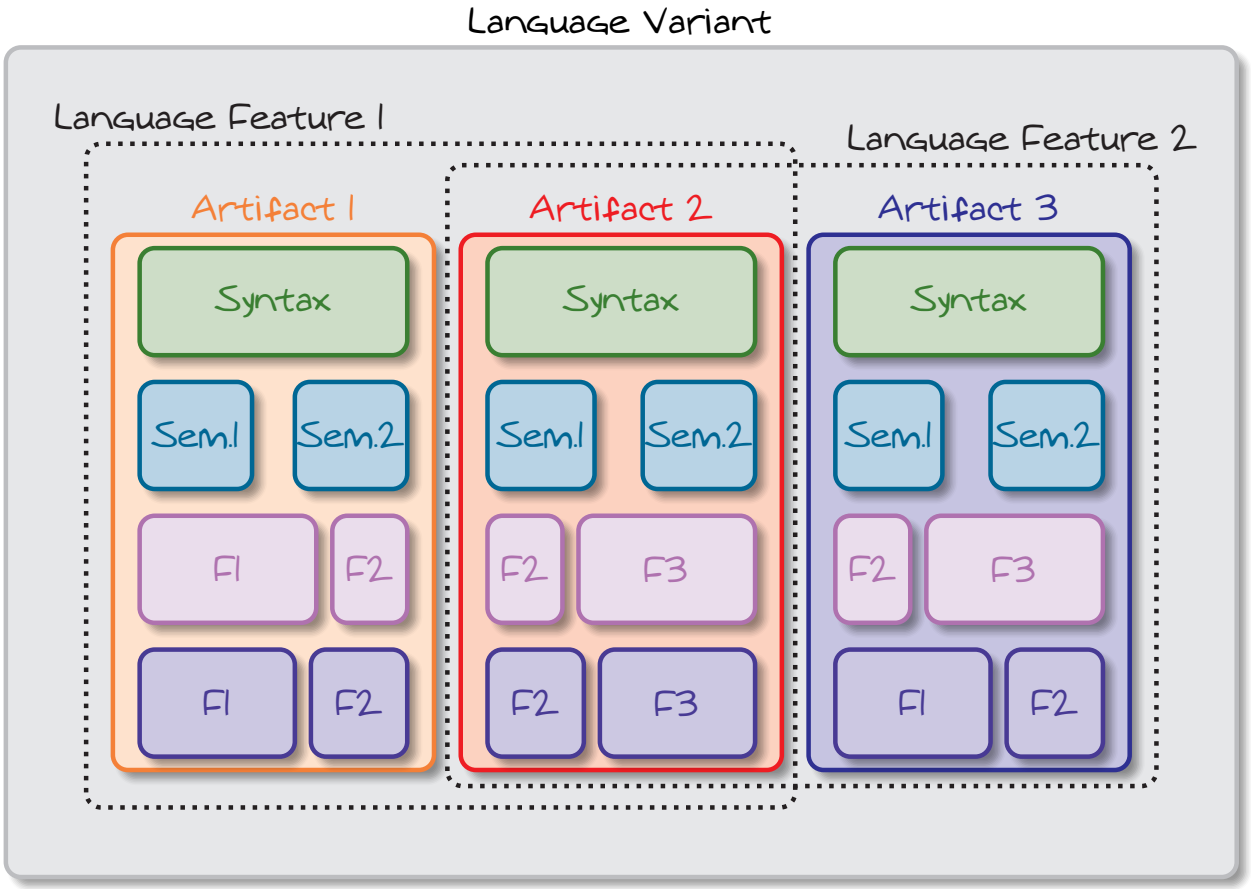
Scientific Contribution
TSs, LSP and DAP Modularization



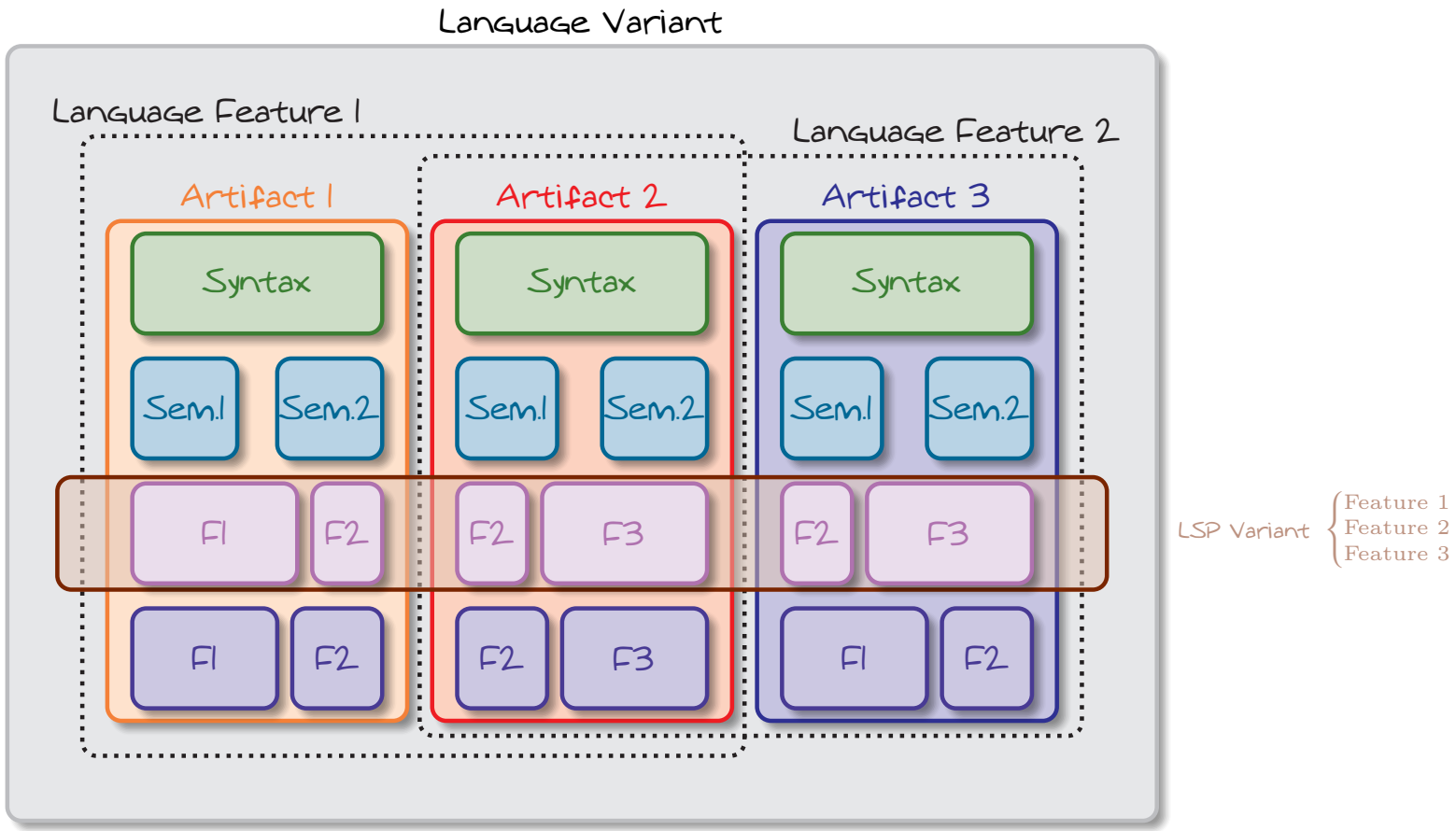
Scientific Contribution
TSs, LSP and DAP Modularization



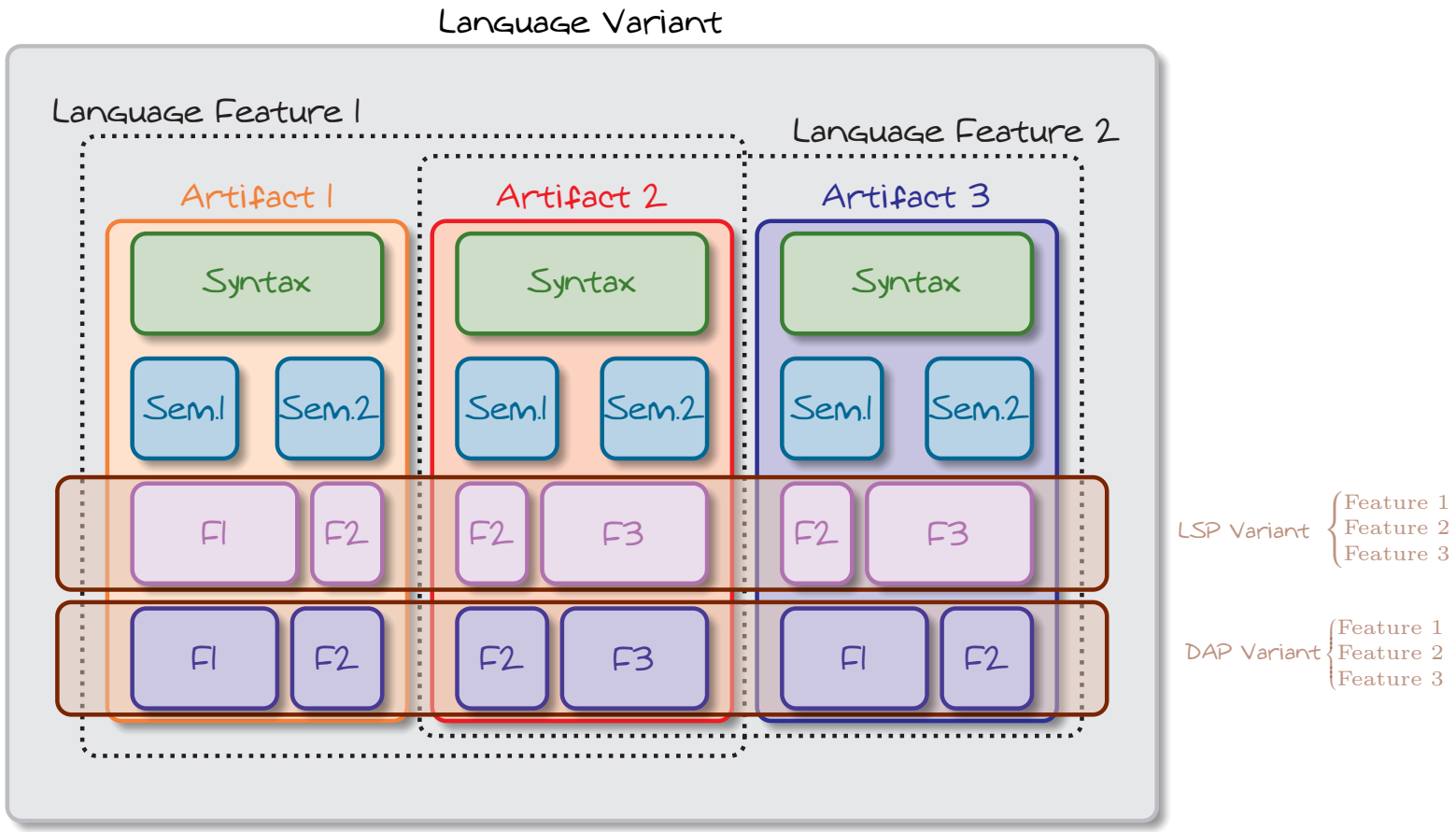
Scientific Contribution
TSs, LSP and DAP Modularization



Scientific Contribution
TSs, LSP and DAP Modularization



Scientific Contribution
TSs, LSP and DAP Modularization



Conclusions
Master's Thesis Results

Interesting results:

- We are writing an article (Code Less to Code More) to be submitted to JSS.
- Propose a feasibility study for the methodology.
- We prototyped the reduction of combinations.
- We prototyped the modularization of the type system.

Thanks for your attention!

Software Product Lines

Since 1990s, researchers have been working on the concept of Software Product Lines (SPLs) to move towards a more modular world.

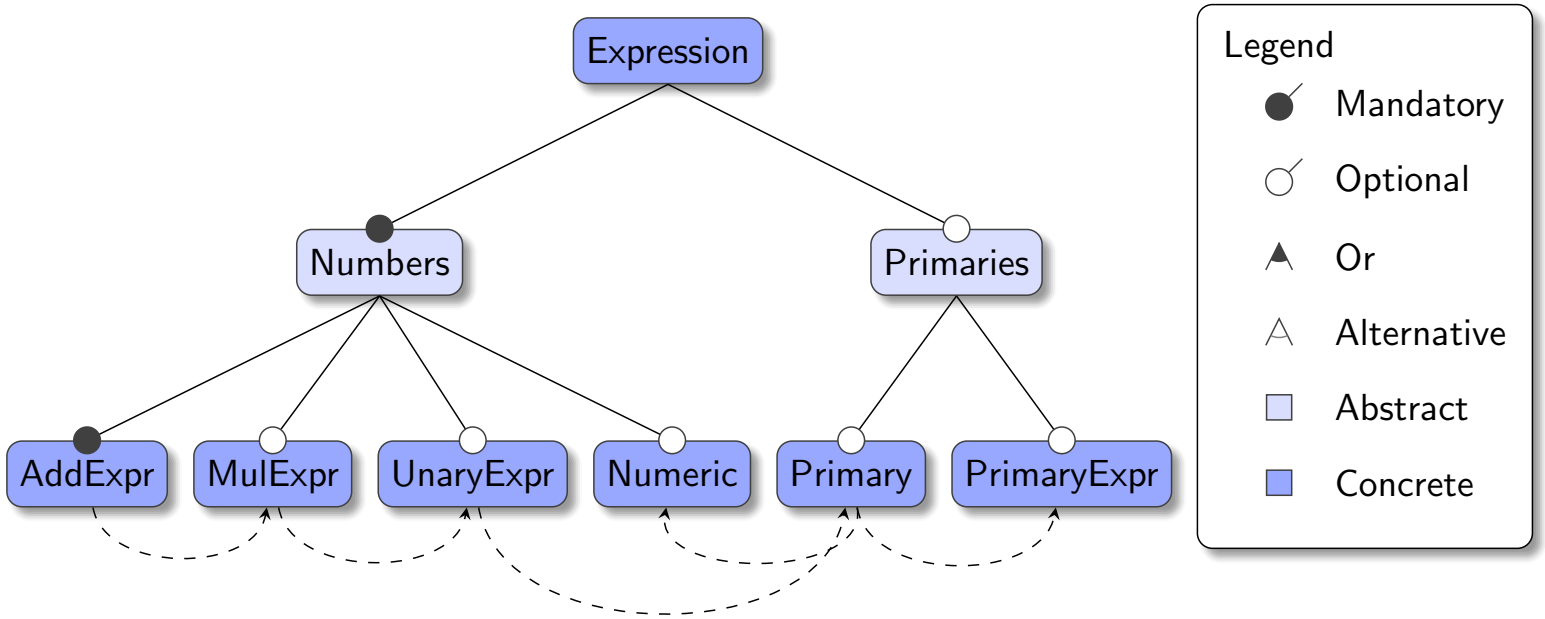
Software Product Lines

Since 1990s, researchers have been working on the concept of Software Product Lines (SPLs) to move towards a more modular world.

- SPLs defines a family of software products.
- SPLs is described by a Feature Model.
- A Feature Model describes the variability of the software.
- SPL variants are generated by selecting a set of features.
- A feature (or artifact) is a first-class entity in SPLs.

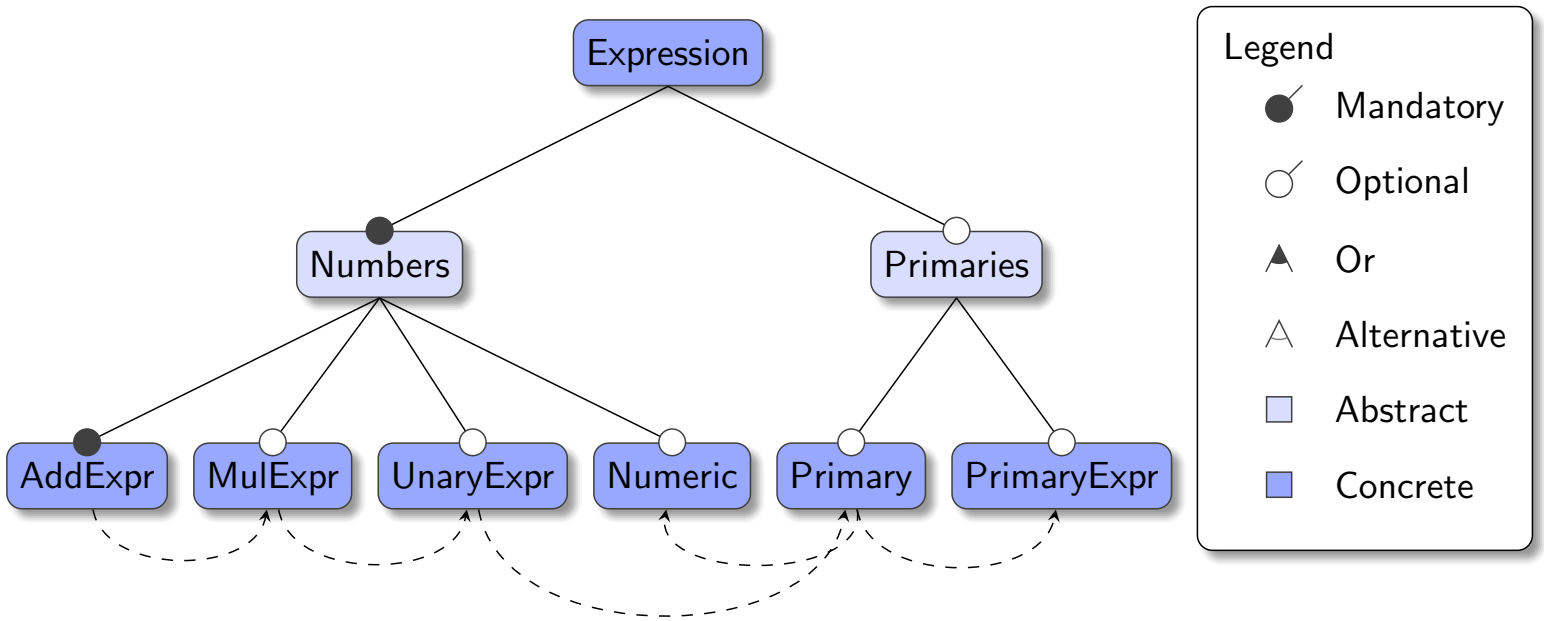
Language Product Lines

Applying the concept of SPLs to programming languages, we obtain the concept of Language Product Lines (LPLs).



Language Product Lines

Applying the concept of SPLs to programming languages, we obtain the concept of Language Product Lines (LPLs).



Some achievements:

- Bottom-up approach to language implementation
- Reusability of language artifacts
- Multiple variants of the same language
- Language Workbenches come to the rescue