

# **Cedalion: A Language Oriented Programming Language**

**Boaz Rosenan**

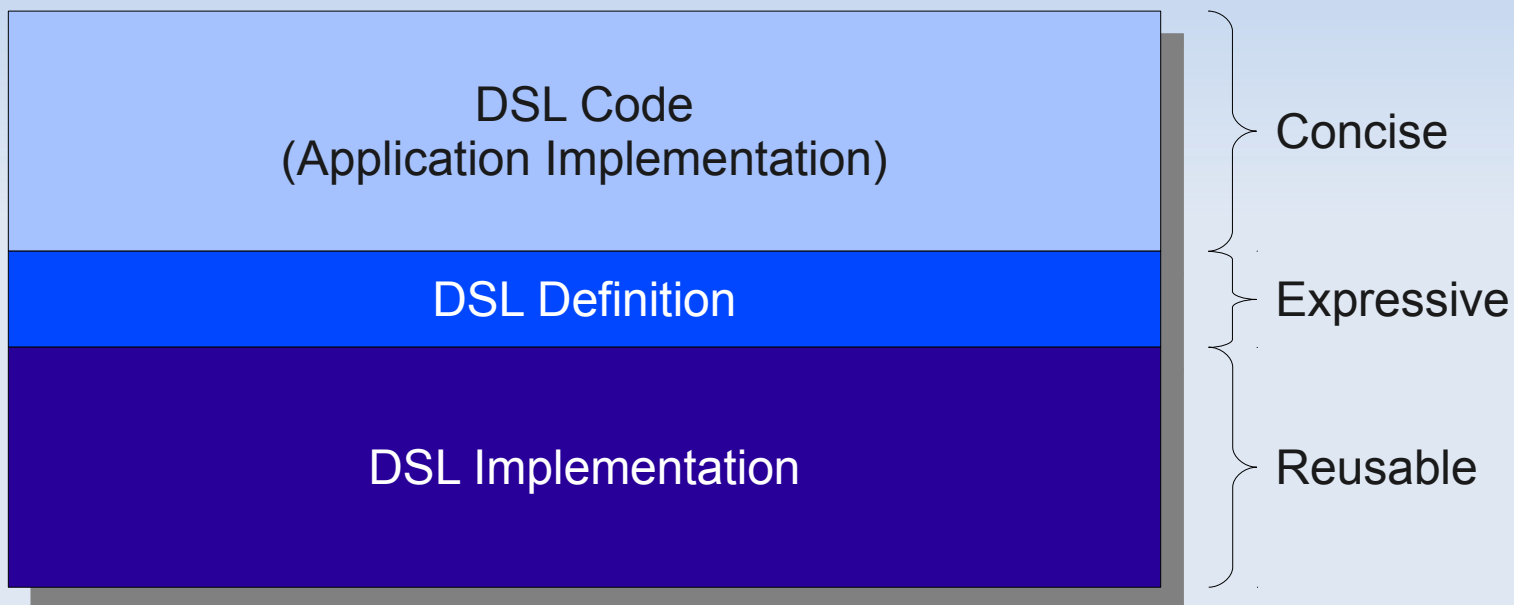
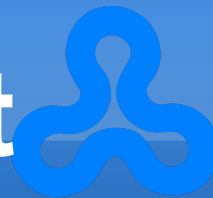
**Dept. of Computer Science**

 **The Open University of Israel**

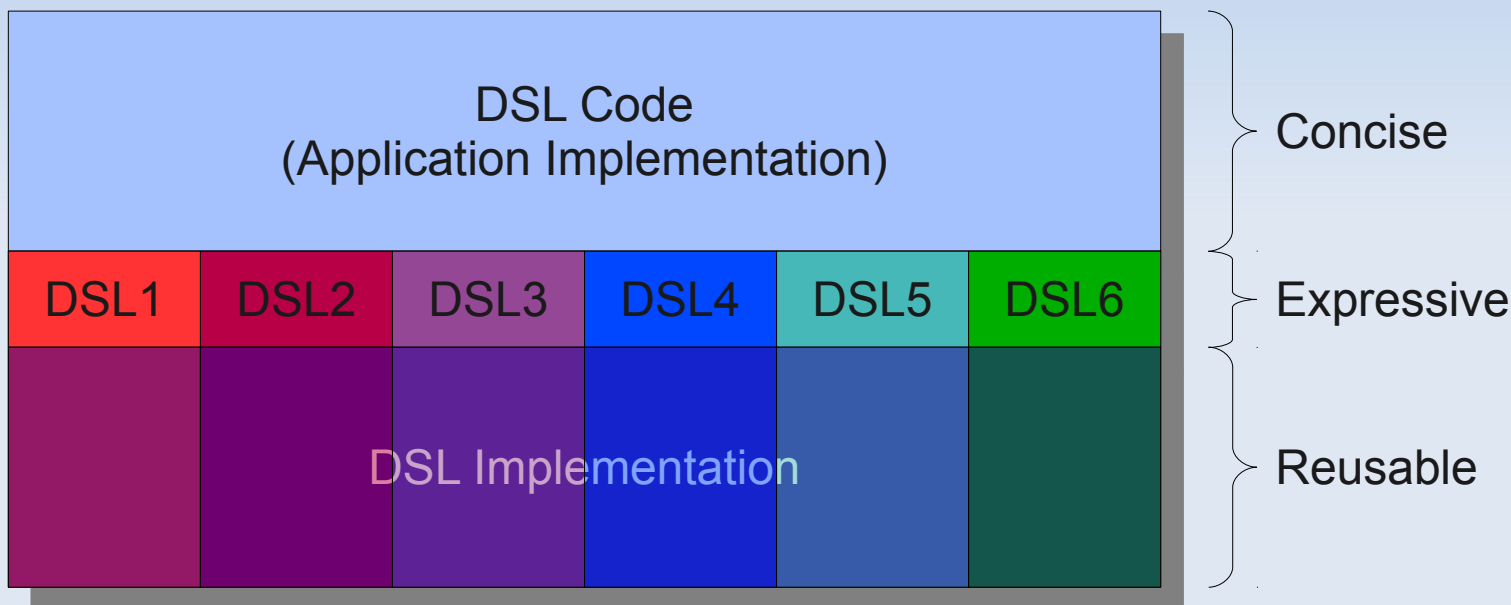
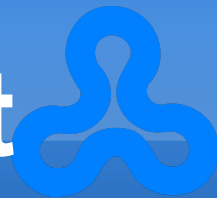
**Joint Work With:**

**David H. Lorenz**

# LOP: Middle-Out Development



# LOP: Middle-Out Development

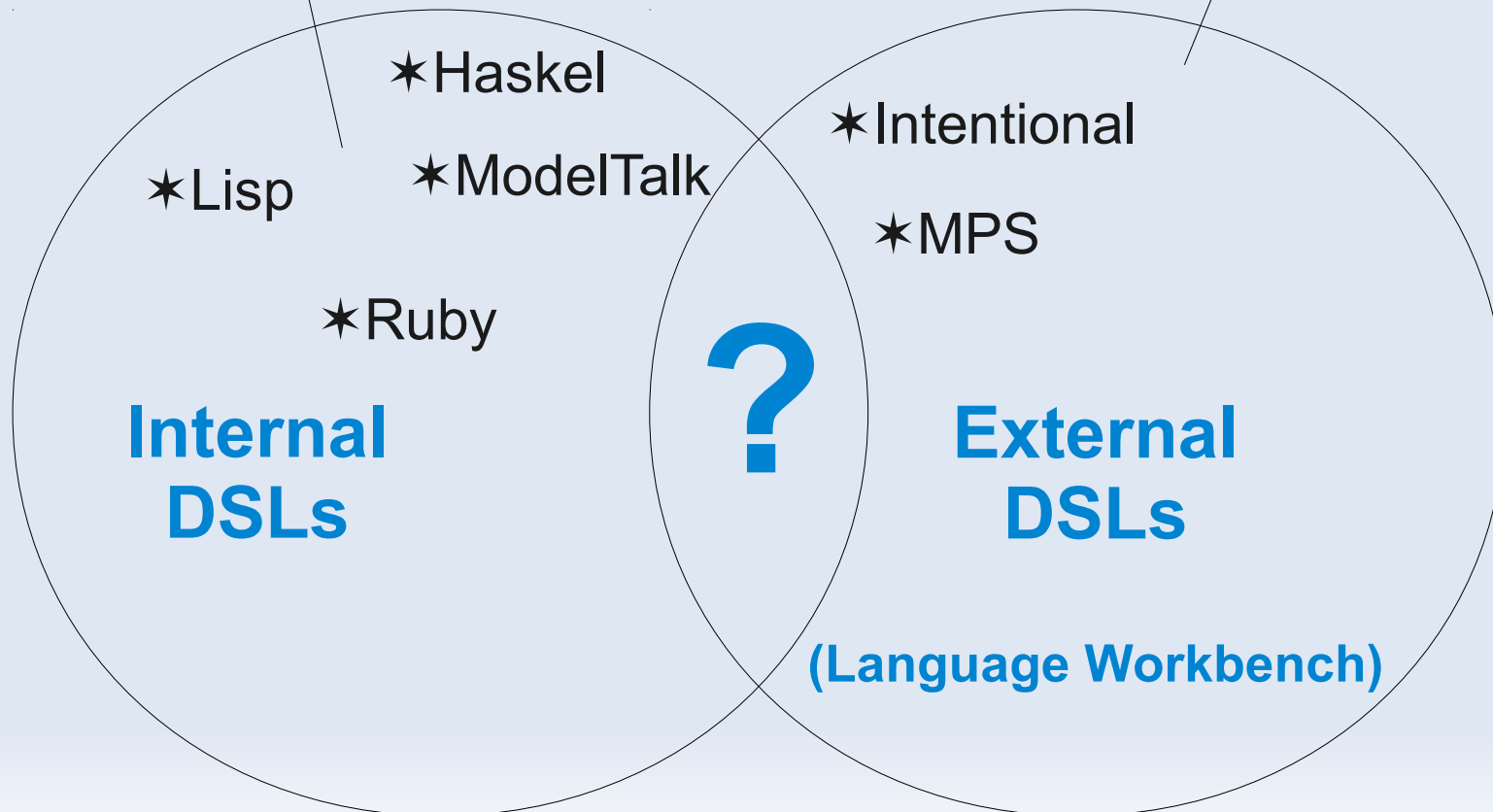


# LOP Platforms



Easy to implement

Freedom in defining syntax and semantics



**Internal  
DSLs**

\*Lisp

\*Haskell

\*ModelTalk

\*Ruby

**External  
DSLs**

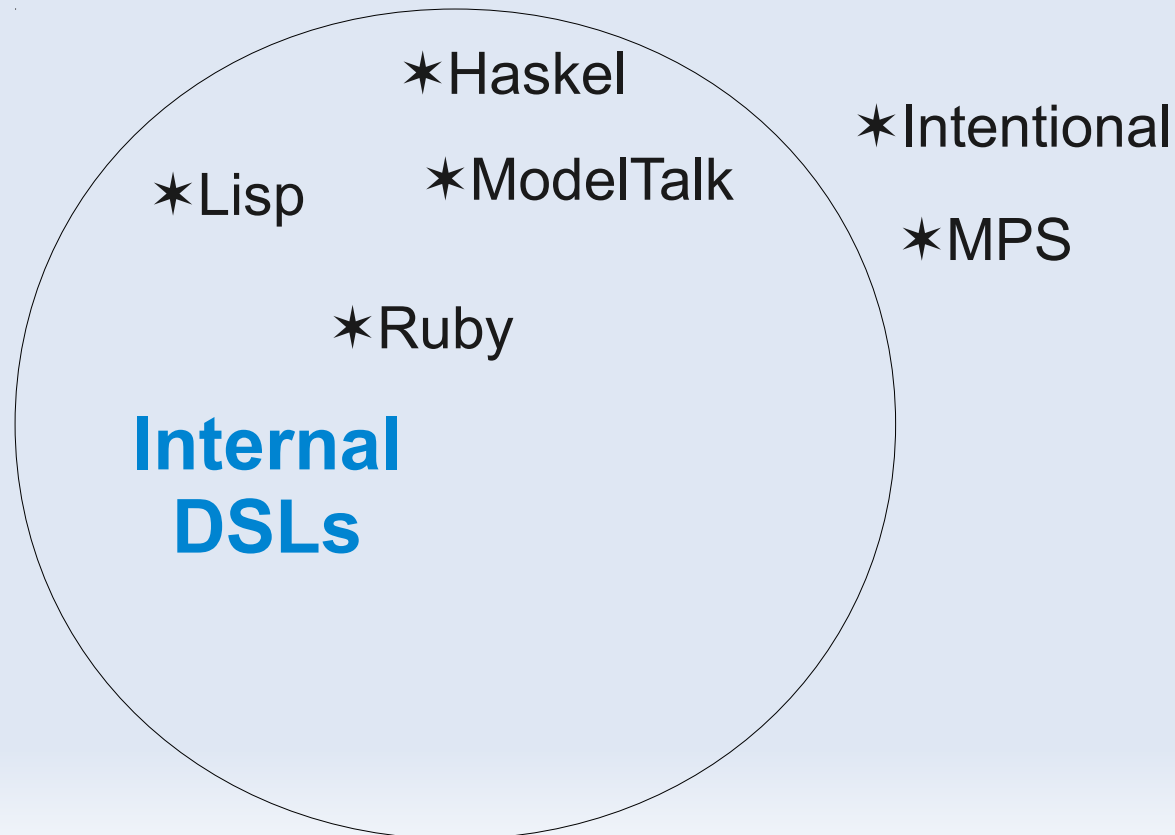
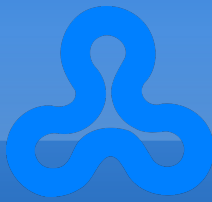
\*Intentional

\*MPS

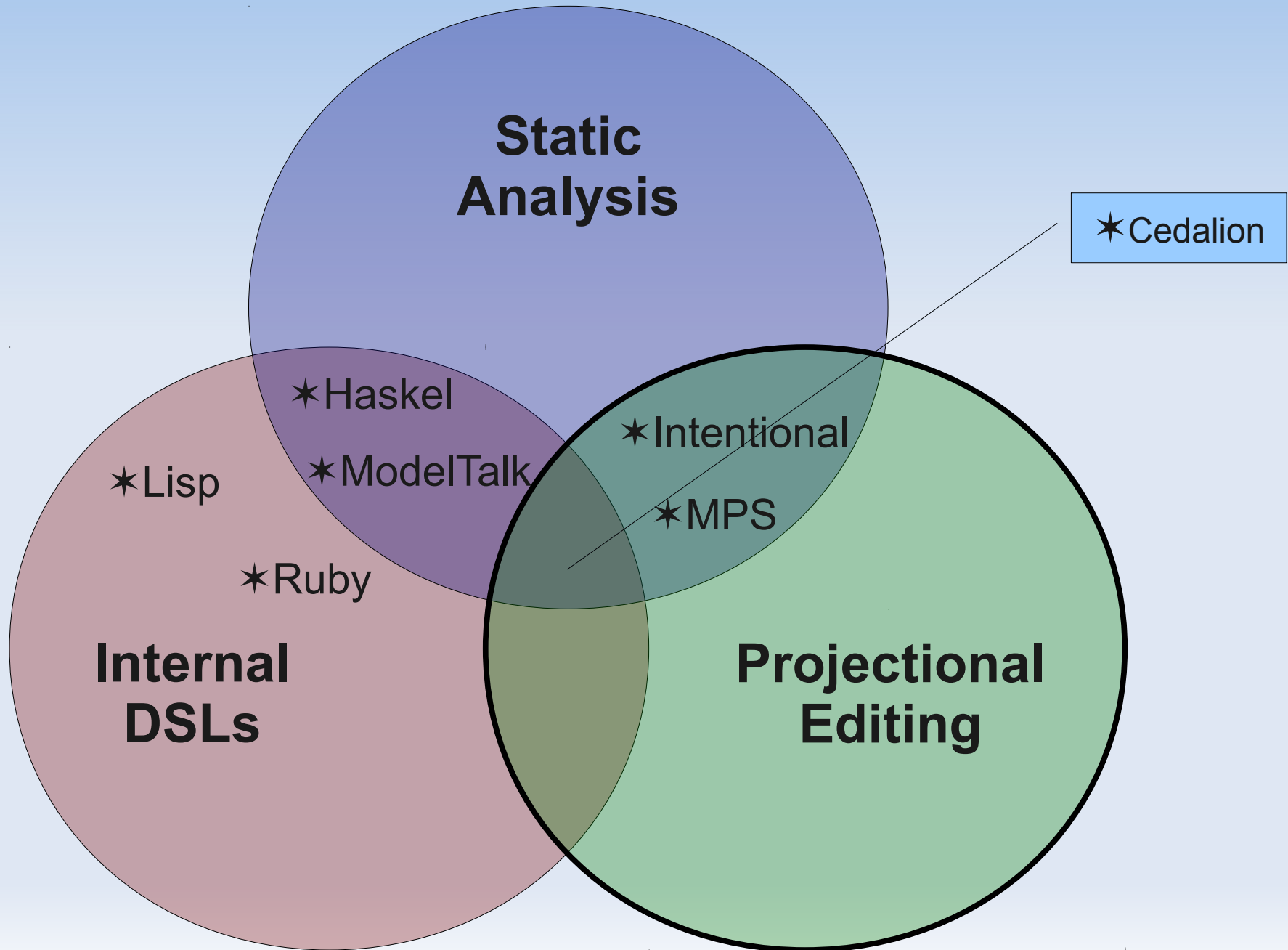
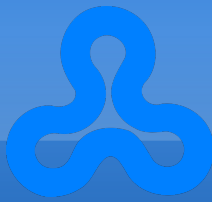
?

(Language Workbench)

# Language Design Perspective



# Language Design Perspective

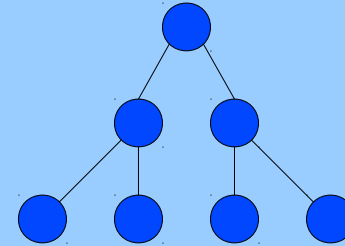


# Projectional Editing

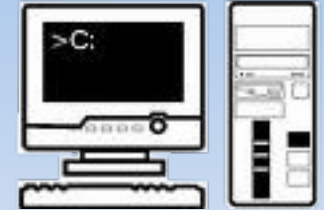


 Text File

Parsing



AST



Traditional

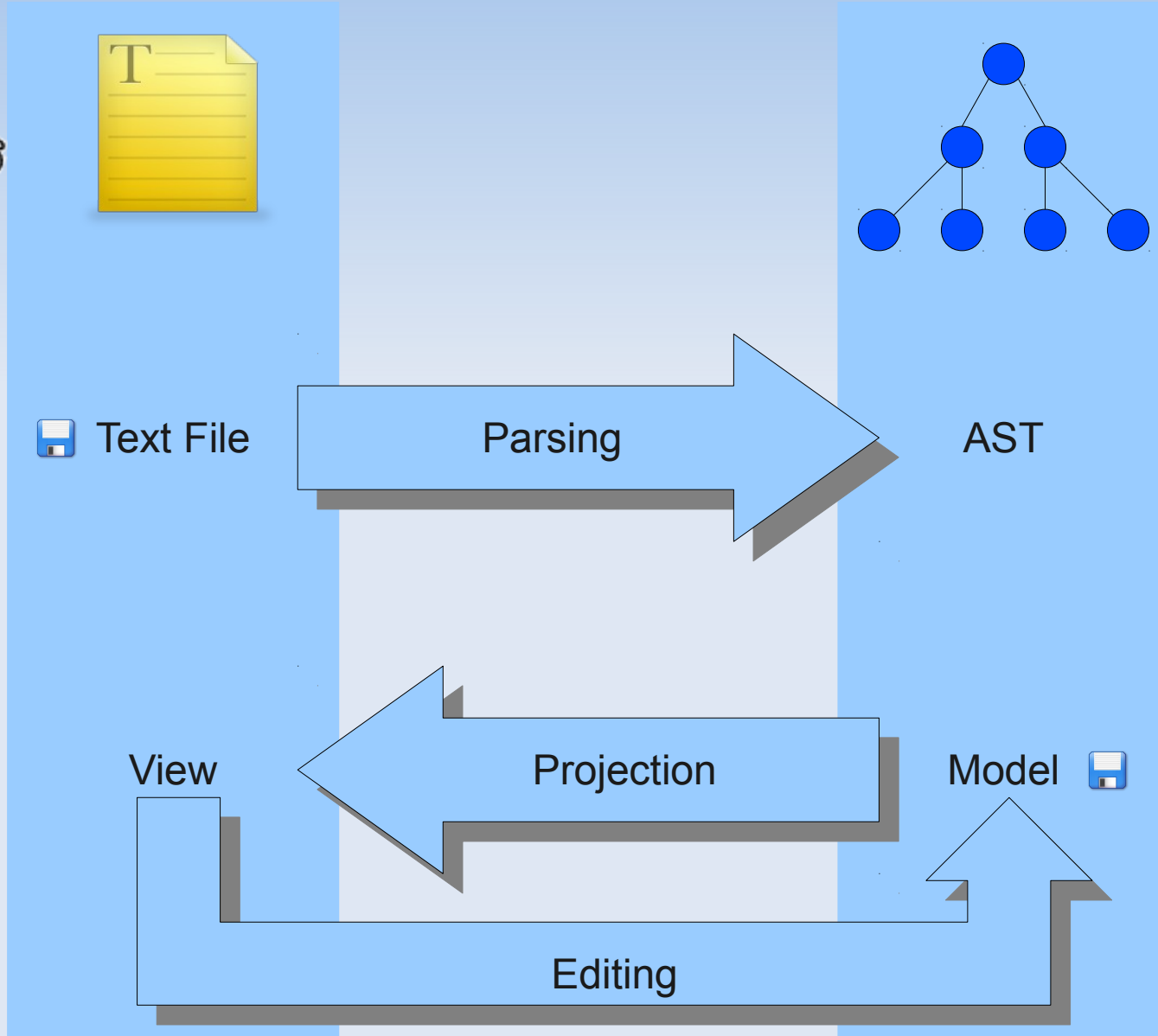
Projectional  
Editing

View

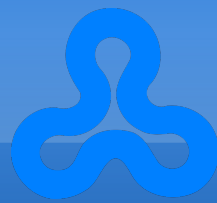
Projection

Model 

Editing



# A Peek at Cedalion



```
File Edit Navigate Search Project Run Window Help
[Icons] [Java]
procedure.ced file.ced edit.ced projection.ced unittest.ced grammer-example.ced »6
[Icons] [Icons] [Icons]
• pattern :: type ↔ []
• ε :: pattern ↔ []
• ε :: pattern → symbol ( 949 )
• ' C ' :: pattern → [ C :: string ]
• ' C ' :: pattern →h [ label ( ' ) , italic ( « C :: string » ) , label ( ' ) ]
• P1 . P2 :: pattern ↔ [ P1 :: pattern , P2 :: pattern ]
• P1 . P2 :: pattern →h [ « P1 :: pattern » , label ( . ) , « P2 :: pattern » ]
• Symbol ::= Pattern :: statement ↔ [ Symbol :: pattern , Pattern :: pattern ]
• Symbol ::= Pattern :: statement →h [ « Symbol :: pattern » , label ( := ) , « Pattern :: pattern » ]
• Pattern ⇒ Text / Residue :: pred ↔ [ Pattern :: pattern , Text :: list ( string ) , Residue :: list ( string ) ]
• Pattern ⇒ Text / Residue :: pred →h [ « Pattern :: pattern » , symbol ( 8658 ) , « Text :: list ( string ) @ horiz » , label ( / ) , « Residue :: list ( string ) @ horiz » ]
• ε ⇒ X / X :-
  true
• ' A ' ⇒ [ A B ] / B :-
  true
• P1 . P2 ⇒ Text / Residue :-
  P1 ⇒ Text / R1 ,
  P2 ⇒ R1 / Residue
• Symbol ::= Pattern → Symbol ⇒ Text / Residue :-
  Pattern ⇒ Text / Residue
• s :: pattern ↔ []
• s ::= ' a ' . s . ' b '
• s ::= ε
• Unit Test: s ⇒ [ a , a , b , b ] / []
• Unit Test: → s ⇒ [ a , a , a , b , b ] / []
```

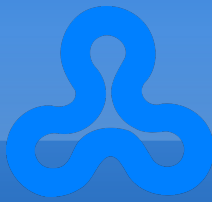
DSL Definition

DSL Implementation

DSL Code (Application Implementation)



# Outline



- Introduction
- **LOP Example**
- Designing an LOP Language
- Demo
- Conclusion

# Example: LOP with Internal DSLs

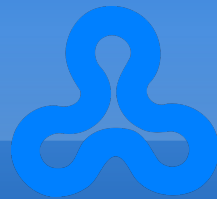


- Task: Implement a parser for a given BNF grammar

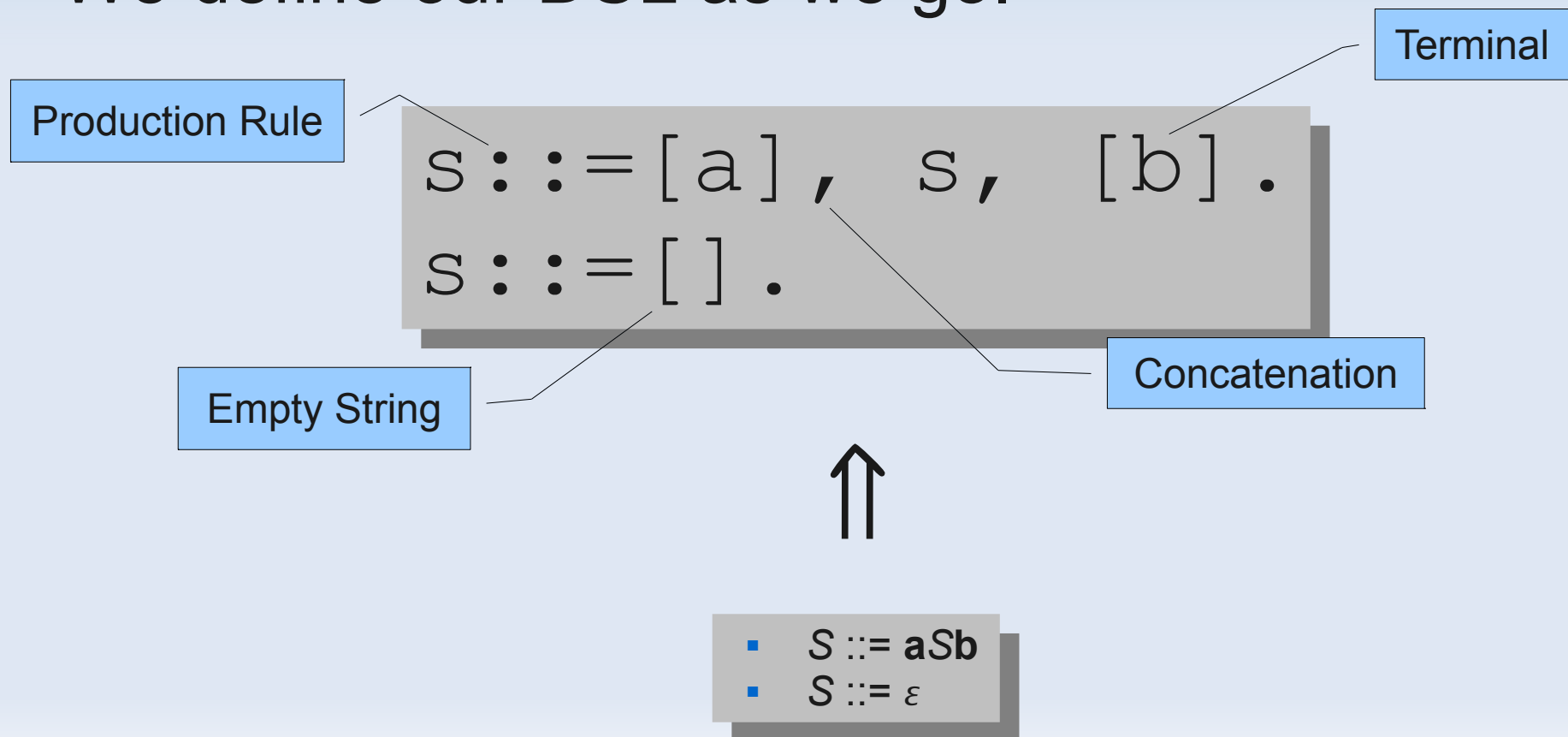
```
■ S ::= aSb  
■ S ::= ε
```

- Steps:
  1. Translating the specification into a “parsable” form in our target language, while implicitly defining the DSL.
  2. Making the code executable by implementing the DSL constructs.
- We use Prolog as a host language.

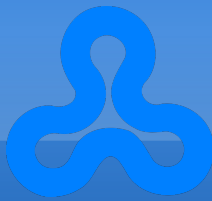
# Step 1: Formalizing the Spec



- We use Prolog's (meta) syntax to express the grammar (our specifications).
- We define our DSL as we go.



# Step 2: Making It Executable



```
:- op(1100, xfx, ' ::= ').
parse([], X, X).
parse([T], [T|L], L).
parse((P1,P2), X, Y) :-
    parse(P1, X, X1),
    parse(P2, X1, Y).
parse(P, X, Y) :-
    (P ::= Q),
    parse(Q, X, Y).
```

Syntactic definition of ' ::= '

Empty String

Terminal

Concatenation

Production Rule

***parse(P, T, R)***: succeeds if a prefix of *T* is derivable from *P*, leaving residue *R*.

# Limitations of Internal DSLs in Prolog

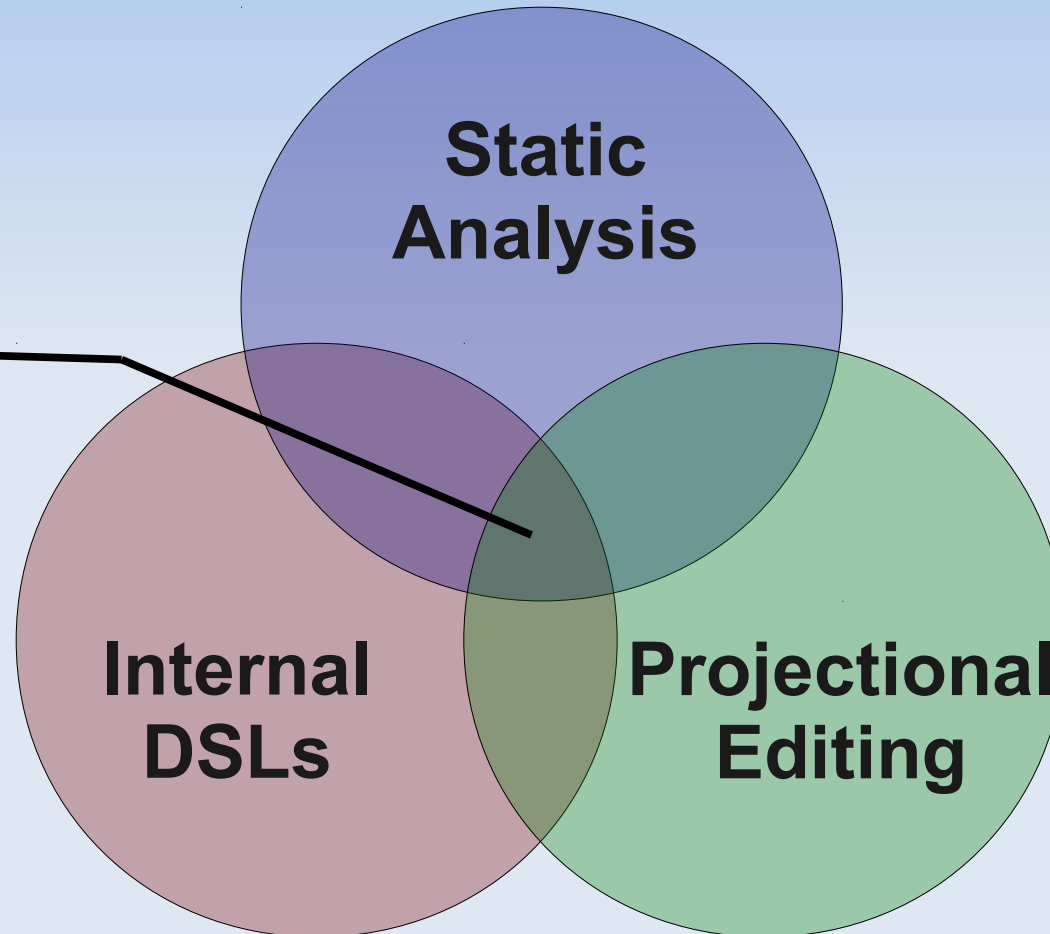


- Syntactic Limitations
  - When fusing several DSLs, operator collisions may occur.
- Semantic Limitations
  - No conformance checking.
  - Mistakenly writing `s::={a},s,{b}` instead of `s::=[a],s,[b]` will result in a bug, not an error.

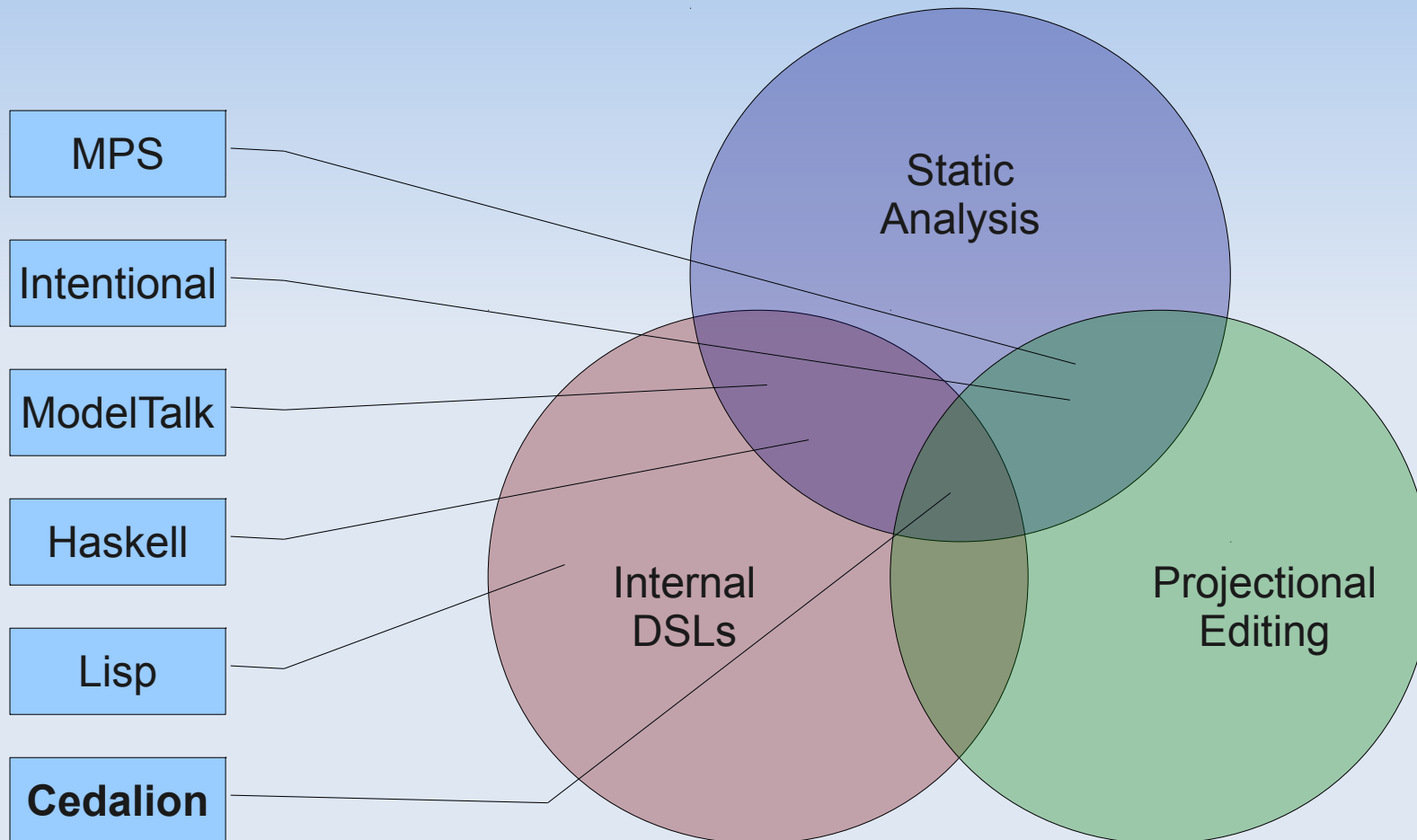
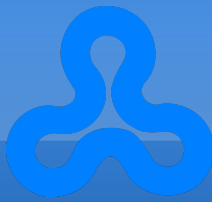
# What is Cedalion



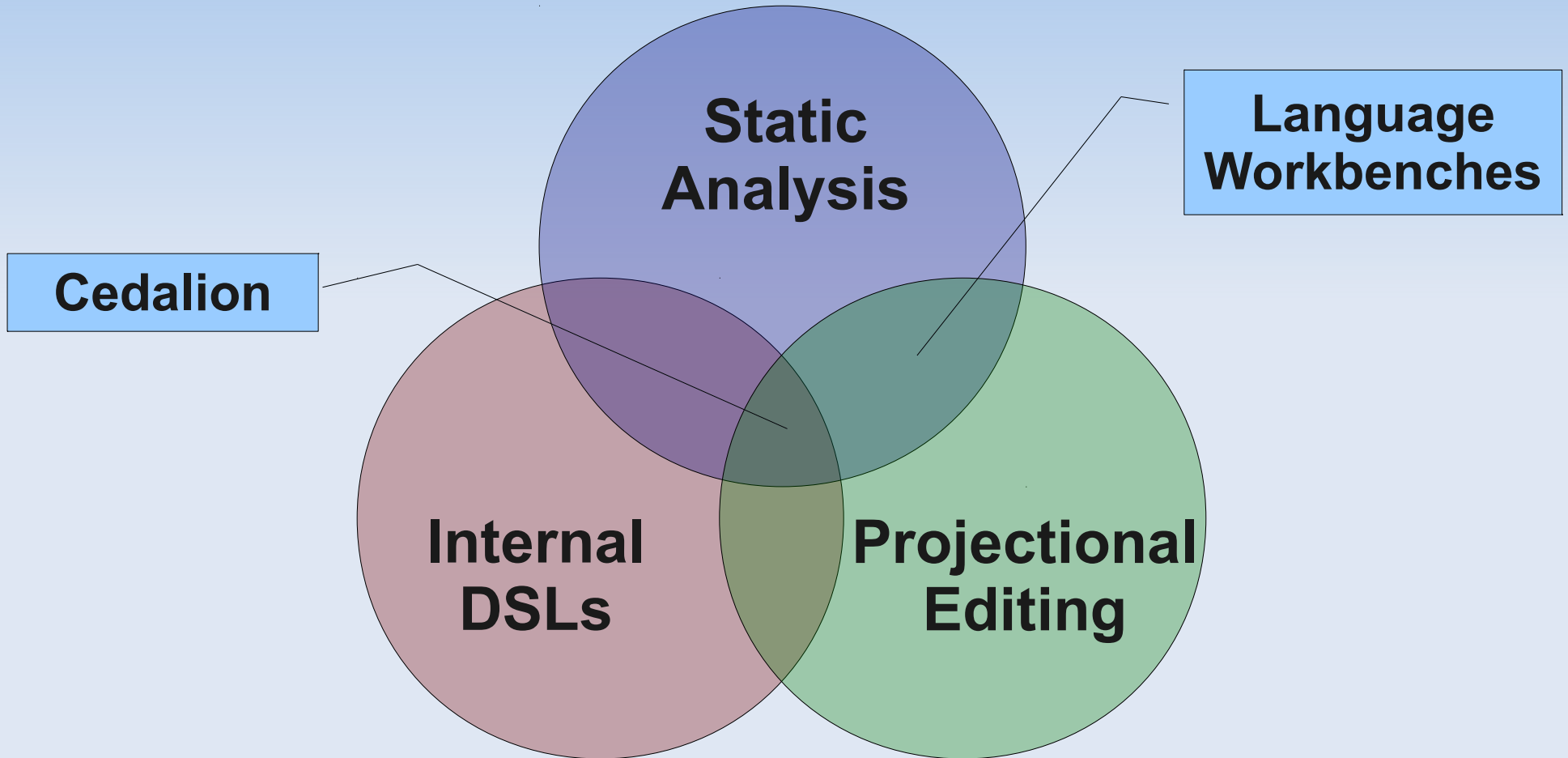
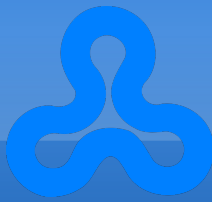
**Cedalion is an  
LOP  
Programming  
Language**



# The Best of All Worlds

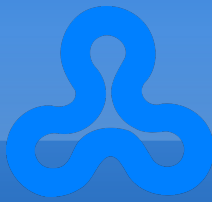


# Our Approach

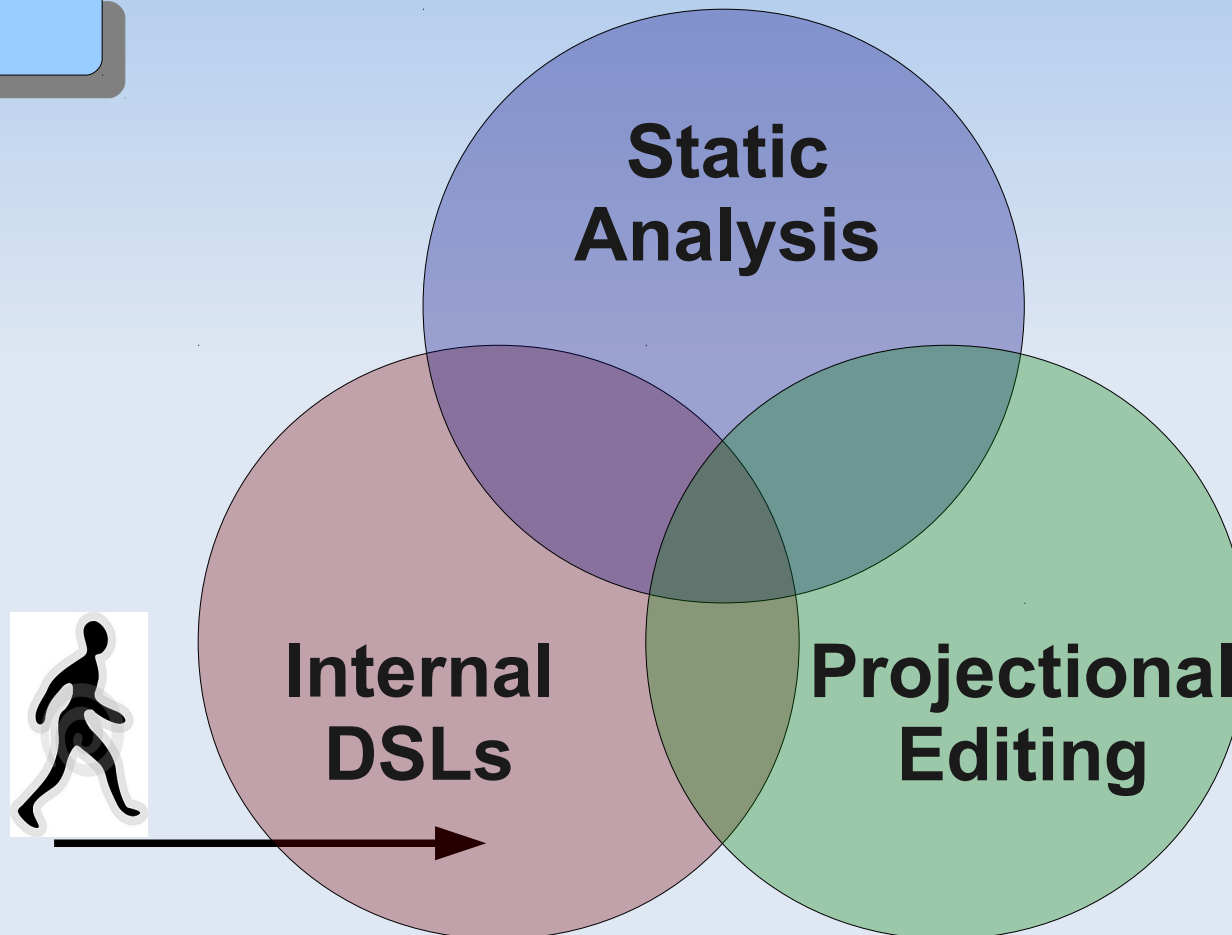




# Designing an LOP Language

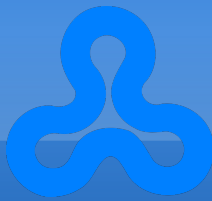


Extensible



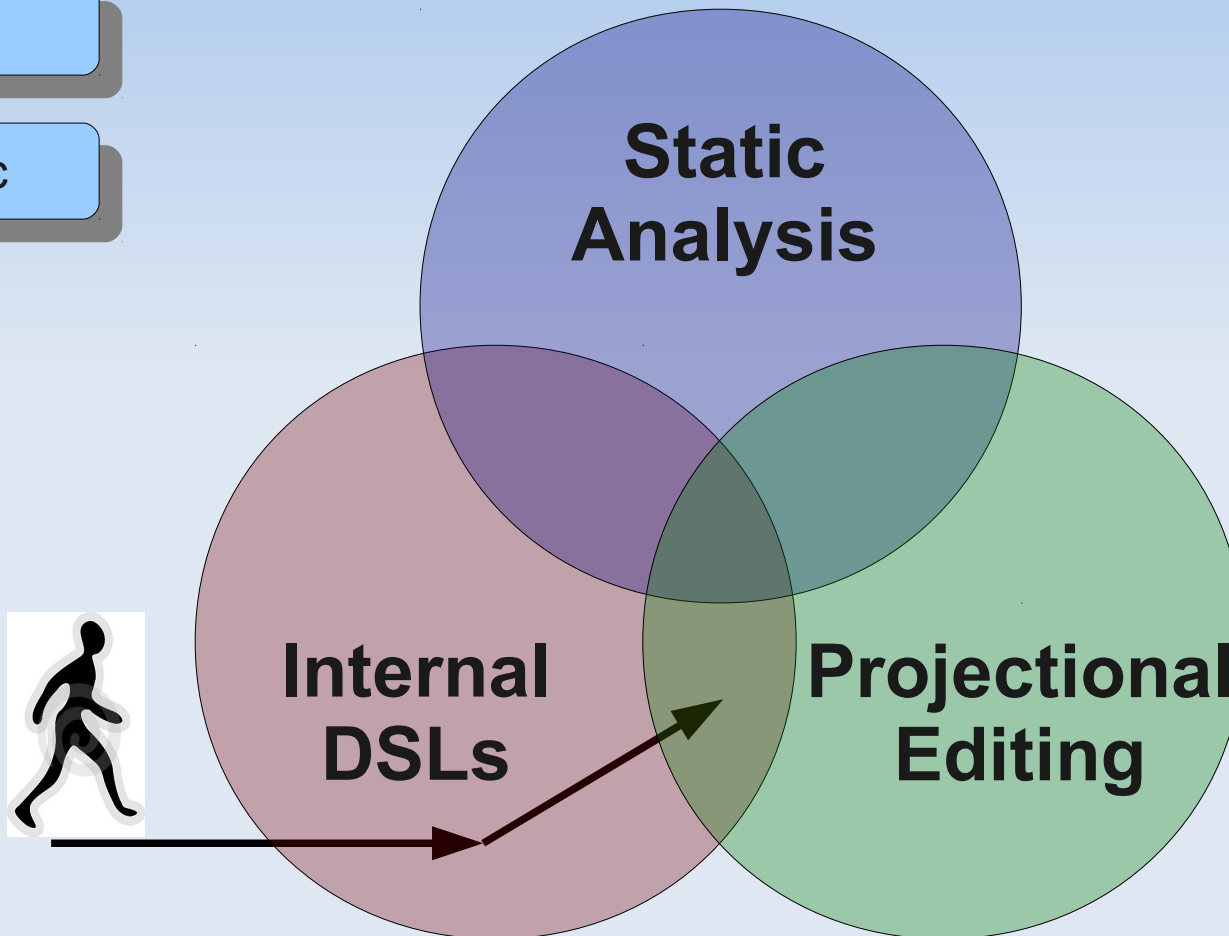
- ~~■ C/C++~~
- ~~■ C#~~
- Haskell
- ~~■ Java~~
- Lisp
- Nemerle
- Prolog
- Ruby
- Smalltalk

# Designing an LOP Language



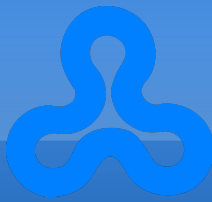
Extensible

Homoiconic



- ~~■ C/C++~~
- ~~■ C#~~
- ~~■ Haskell~~
- ~~■ Java~~
- Lisp
- Nemerle
- Prolog
- ~~■ Ruby~~
- ~~■ Smalltalk~~

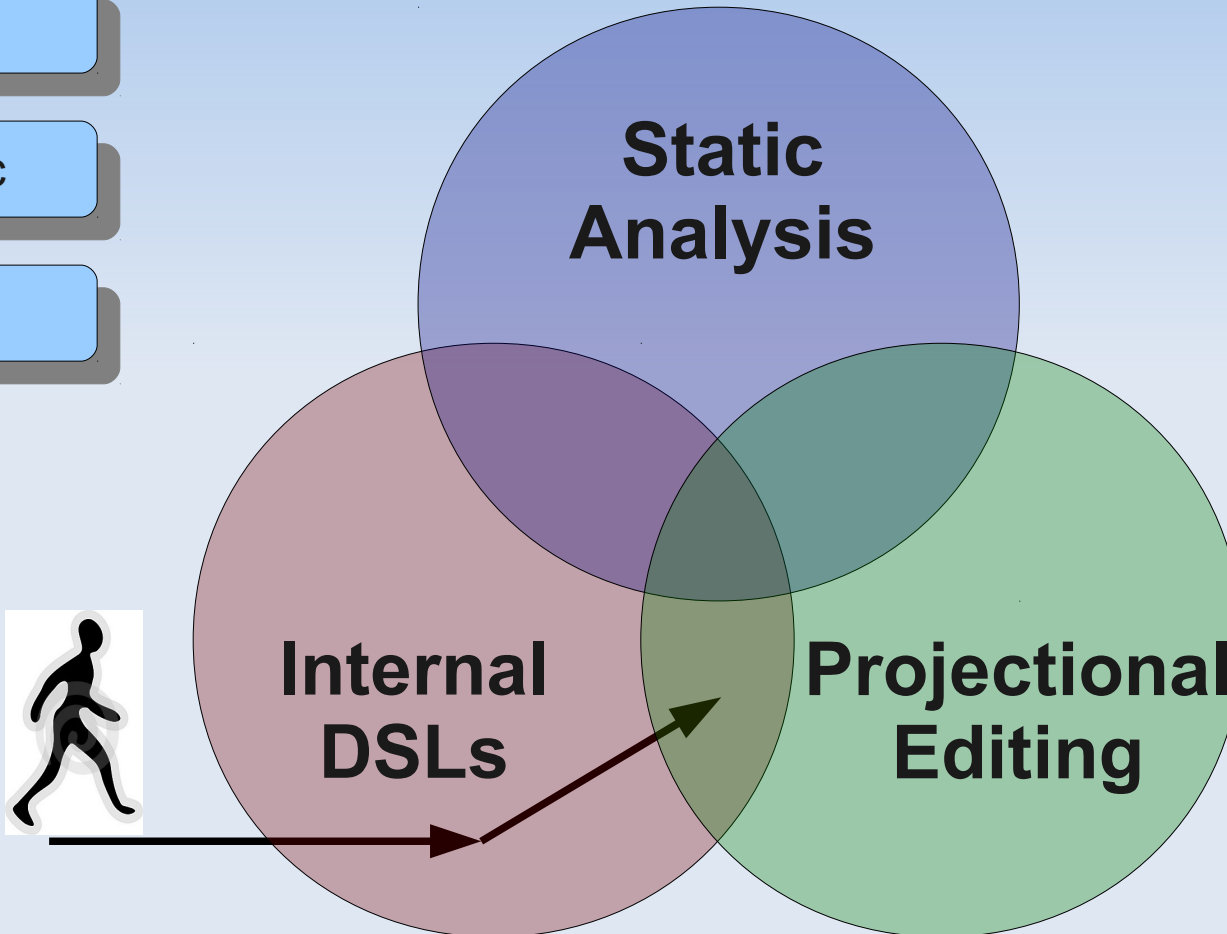
# Designing an LOP Language



Extensible

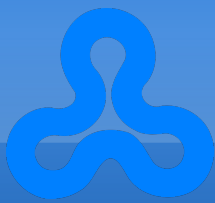
Homoiconic

Dynamic



- ~~■ C/C++~~
- ~~■ C#~~
- ~~■ Haskell~~
- ~~■ Java~~
- Lisp
- ~~■ Nemerle~~
- Prolog
- ~~■ Ruby~~
- ~~■ Smalltalk~~

# Designing an LOP Language

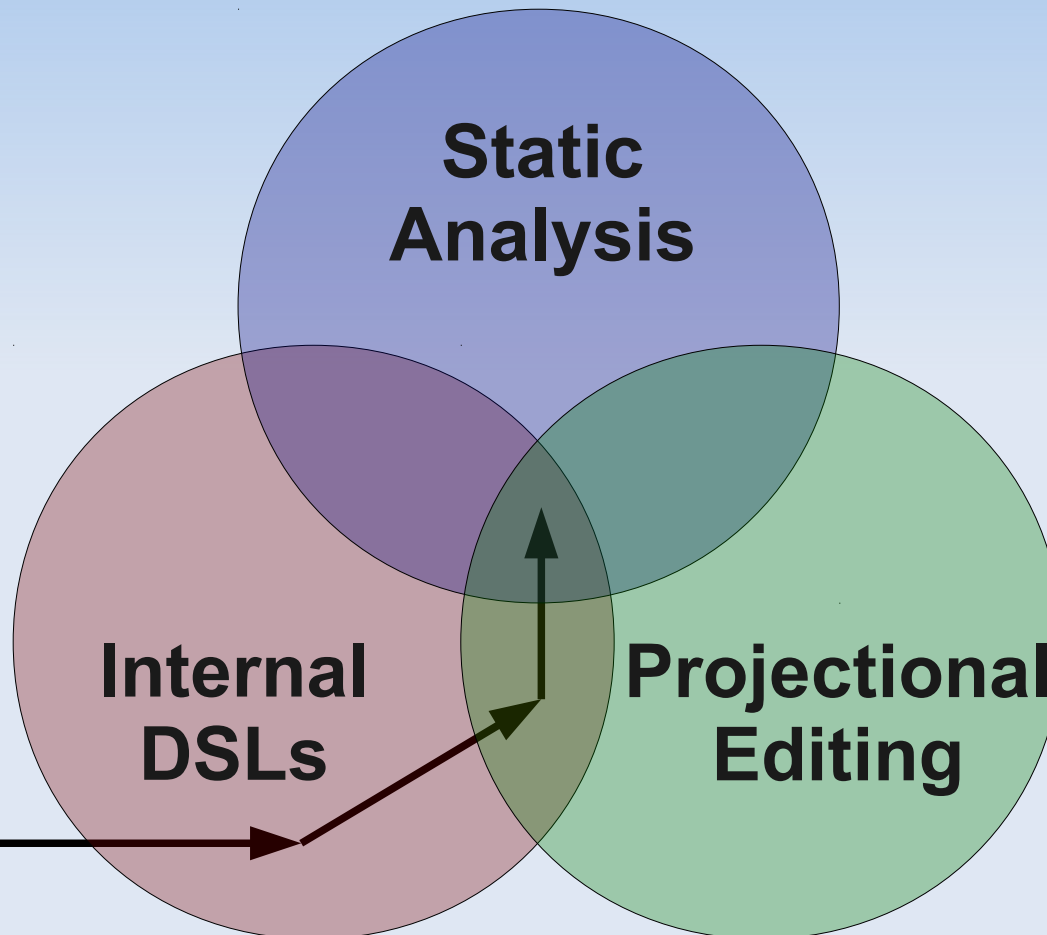


Extensible

Homoiconic

Dynamic

Validators



~~■ C/C++~~

~~■ C#~~

~~■ Haskell~~

~~■ Java~~

■ Lisp

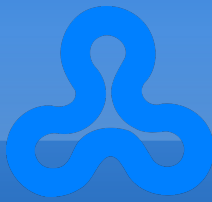
~~■ Nemerle~~

■ Prolog

~~■ Ruby~~

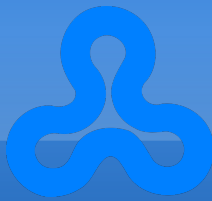
~~■ Smalltalk~~

# Cedalion



- An LOP language
  - Extensible, homoiconic, and dynamic
  - Statically typed, using Hindley-Milner type inference, implemented using validators.
  - Uses projectional editing (cannot be edited as text)
- Based on logic programming
  - As its “Domain 0”
- Comes with a Language Workbench
  - Provides projectional editing for Cedalion

# The Cedalion Workbench



- Based on Eclipse (3.5).
- Queries the Cedalion program for information on how to project language constructs.

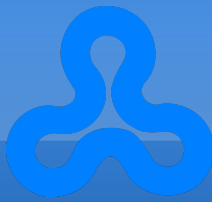
```
File Edit Navigate Search Project Run Window Help
A procedure.ced A file.ced A edit.ced A grammar-example.ced 33 A projection.ced A sets.ced %x

• pattern :: type → []
• Pattern ⇒ Text / Residue :: pred ↔ [ Pattern :: pattern , Text :: list ( string ) , Residue :: list ( string ) ]
• Pattern ⇒ Text / Residue :: pred ↔ ^ [ « Pattern :: pattern » , symbol ( 8658 ) , « Text :: list ( string ) @ horiz » ]
• ε :: pattern → []
• ε :: pattern → symbol ( 949 )
• ε ⇒ X / X :-
  true
• ' C ' :: pattern → [ C :: string ]
• ' C ' :: pattern → ^ [ label ( ' ) , italic ( « C :: string » ) , label ( ' ) ]
• ' A ' ⇒ [ A B ] / B :-
  true
• P1 . P2 :: pattern → [ P1 :: pattern , P2 :: pattern ]
• P1 . P2 :: pattern → ^ [ « P1 :: pattern » , label ( . ) , « P2 :: pattern » ]
• P1 . P2 ⇒ Text / Residue :-
  P1 ⇒ Text / R1 ,
  P2 ⇒ R1 / Residue
• Symbol ::= Pattern :: statement ↔ [ Symbol :: pattern , Pattern :: pattern ]
• Symbol ::= Pattern :: statement ↔ ^ [ « Symbol :: pattern » , label ( ::= ) , « Pattern :: pattern » ]
• Symbol ::= Pattern → Symbol ⇒ Text / Residue :-
  Pattern ⇒ Text / Residue

s :: pattern → []
s ::= ' a ' . s . ' b '
s ::= ε
Unit Test: s = [ a , a , b , b ] / []
Unit Test: → s = [ a , a , a , b , b ] / []
```

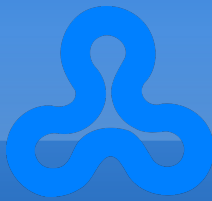


# Outline



- Introduction
- LOP Example
- Designing an LOP Language
- **Demo**
- Conclusion

# Related Work

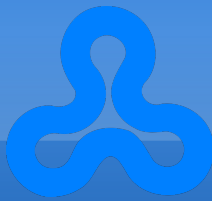


- Meta Programming System
  - A language workbench for external DSLs.
  - Developed by S. Dmitriev at JetBrains, released 2009.
- Intentional Domain Workbench
  - A language workbench for external DSLs.
  - Developed by C. Simonyi at Intentional Software, not released yet.
- ModelTalk/Ink
  - A language workbench for XML-based internal DSLs. Used internally at Pontis.
  - [A. Hen-Tov, D. H. Lorenz and L. Shachter, 2009].



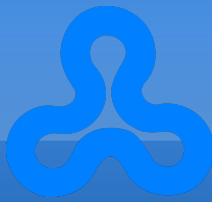


# Conclusion



- Cedalion is a proof of concept
  - Provides the ease of defining internal DSLs.
  - Provides full control in defining syntax and semantics.
- Future Work
  - Making Cedalion ready for “prime time”.
  - Case studies: Solve real-life problems with Cedalion.
  - Evaluating other base semantics.

# Thank You



- Cedalion
  - <http://aop.cslab.openu.ac.il/research/cedalion/>
- Boaz Rosenan
  - [boaz@nansore.net](mailto:boaz@nansore.net)
  - [http://wiki-openu.openu.ac.il/research/index.php/Boaz\\_Rosenan](http://wiki-openu.openu.ac.il/research/index.php/Boaz_Rosenan)
- David H. Lorenz
  - [lorenz@openu.ac.il](mailto:lorenz@openu.ac.il)
  - <http://www.openu.ac.il/home/lorenz/>