22c181:
Formal Methods in Software Engineering

The University of Iowa

Spring 2008

Overview of the KeY System

Copyright 2007-8 Reiner Hähnle and Cesare Tinelli.

Notes originally developed by Reiner Hähnle at Chalmers University and modified by Cesare Tinelli at the University of Iowa. These notes are copyrighted materials and may not be used in other course settings outside of the University of Iowa in their current form or modified form without the express written permission of one of the copyright holders.

Contents of Second Part of Course

- **Overview of KeY**
- **UML and its semantics**
- **Introduction to OCL**
- **Specifying requirements with OCL**
- **Modelling of Systems with Formal Semantics**
- **Propositional & First-order logic, sequent calculus**
- **OCL to Logic, horizontal proof obligations, using KeY**
- **Dynamic logic, proving program correctness**
- **Java Card DL**
- **Vertical proof obligations, using KeY**
- **Wrap-up, trends**

Philosophy of KeY Tool

Formal Methods must *and can* be integrated into commercial processes, tools and languages for software development

Philosophy of KeY Tool

Formal Methods must *and can* be integrated into commercial processes, tools and languages for software development

Integrated tool for

- **Modeling**
- **Development**
- **Formal specification**
- **Formal verification**

of object oriented programs

KeY Technologies Used in this Course

Standard language for Modeling of Software

- **Unified Modeling Language — UML** **(Borland Together)**

  Visual language for OO Modeling
  Standard of Object Management Group (OMG)

- **Object Constraint Language — OCL**

  Formal textual language for requirements specification
  UML sub-standard

Modern industrial programming language

Java (Card)

Logic, Automated Deduction

First-order logic, Dynamic Logic, Theorem proving

Other KeY Interfaces

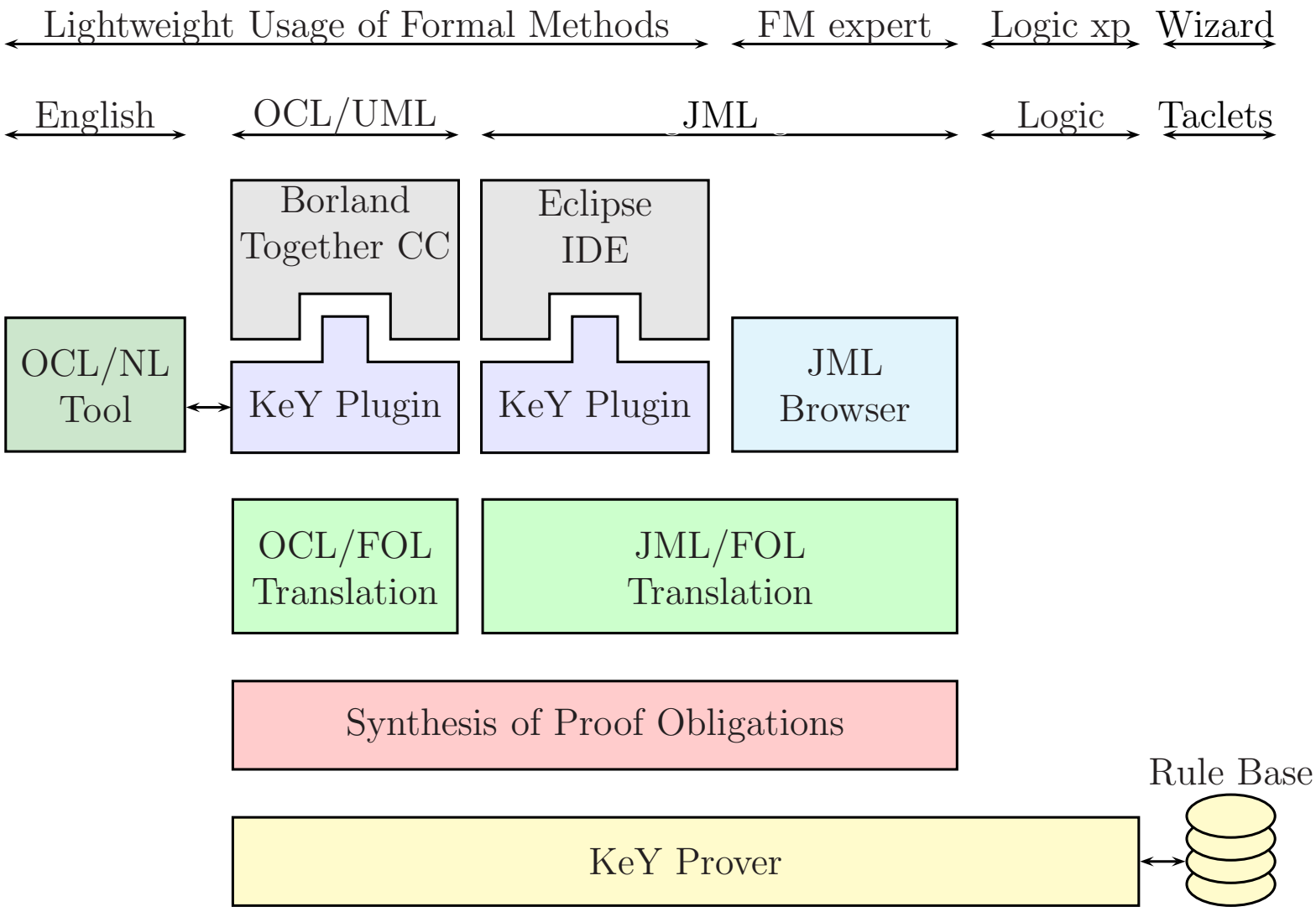
- **KeY plugin for Eclipse IDE**
- **Java Modeling Language — JML**

  Formal Interface Specification Language for Java
  Wide-spread in academic projects
- **Automatic Translation from OCL into English**

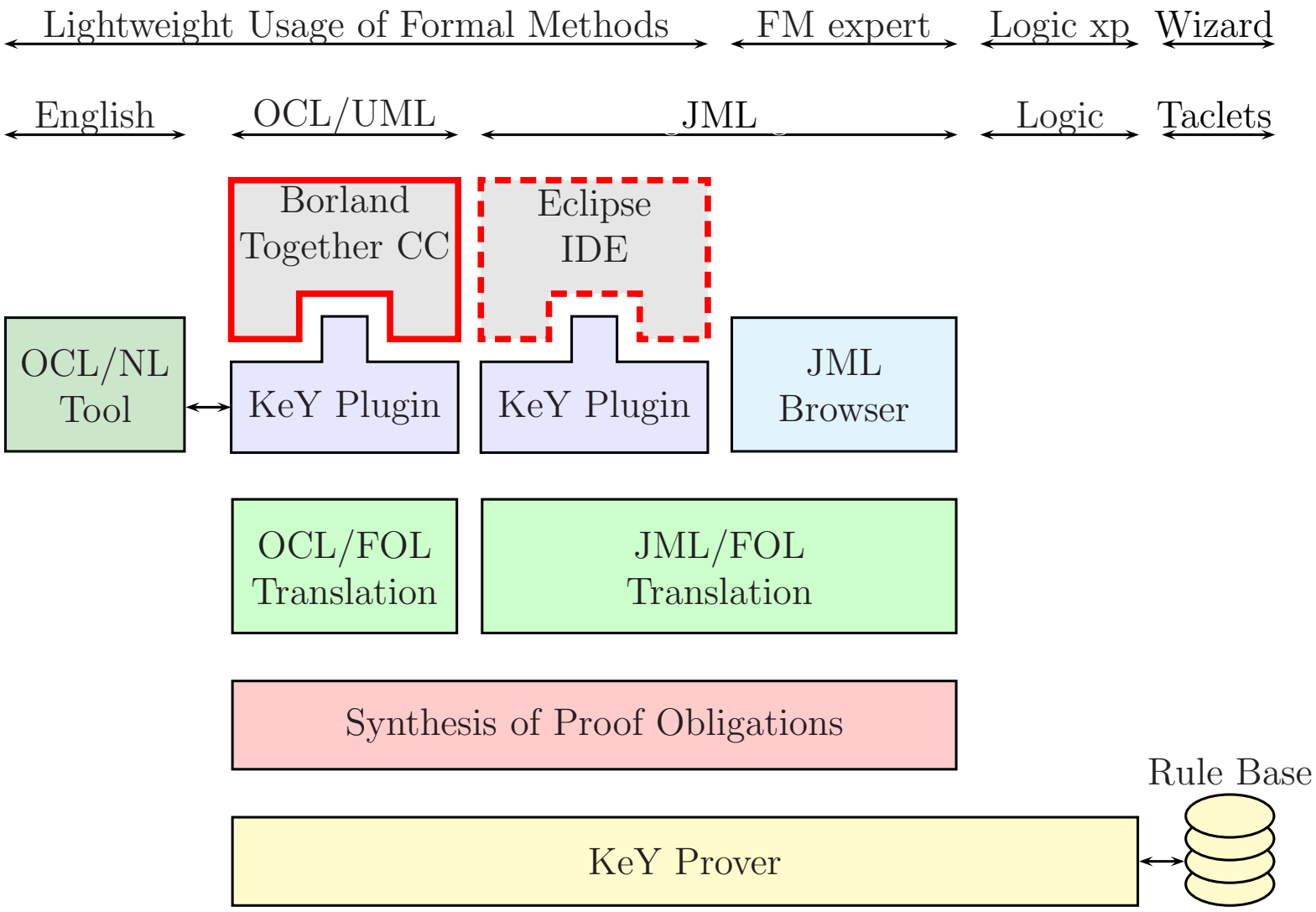
To come soon:

- **Visual symbolic debugger based on symbolic program execution**

The KeY System



The KeY System — UML Design, Java Coding



Target Language: Java Card

Target Language: Java Card

Java Card

- **Sun's Java dialect for smart cards and embedded systems**

Target Language: Java Card

Java Card

- Sun's Java dialect for smart cards and embedded systems

Java Card is relevant target language for verification:

- Restrictions admit complete coverage (more later)
- Applications smallish

Feasability

Target Language: Java Card

Java Card

- Sun's Java dialect for smart cards and embedded systems

Java Card is relevant target language for verification:

- Restrictions admit complete coverage (more later)
- Applications smallish

- Applications safety & security critical
- Often impossible to update smart cards/embedded systems

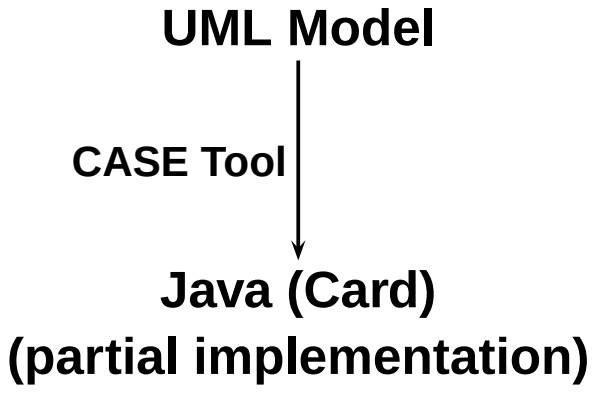
Significance Feasability

How KeY Works

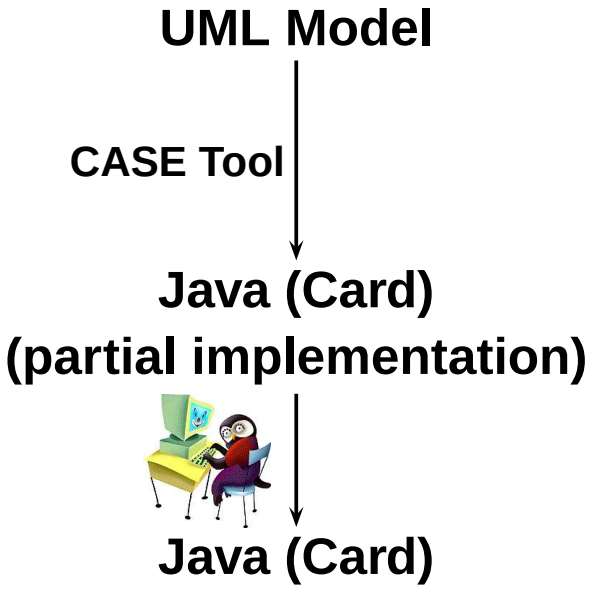
Conventional CASE-Based Development

UML Model

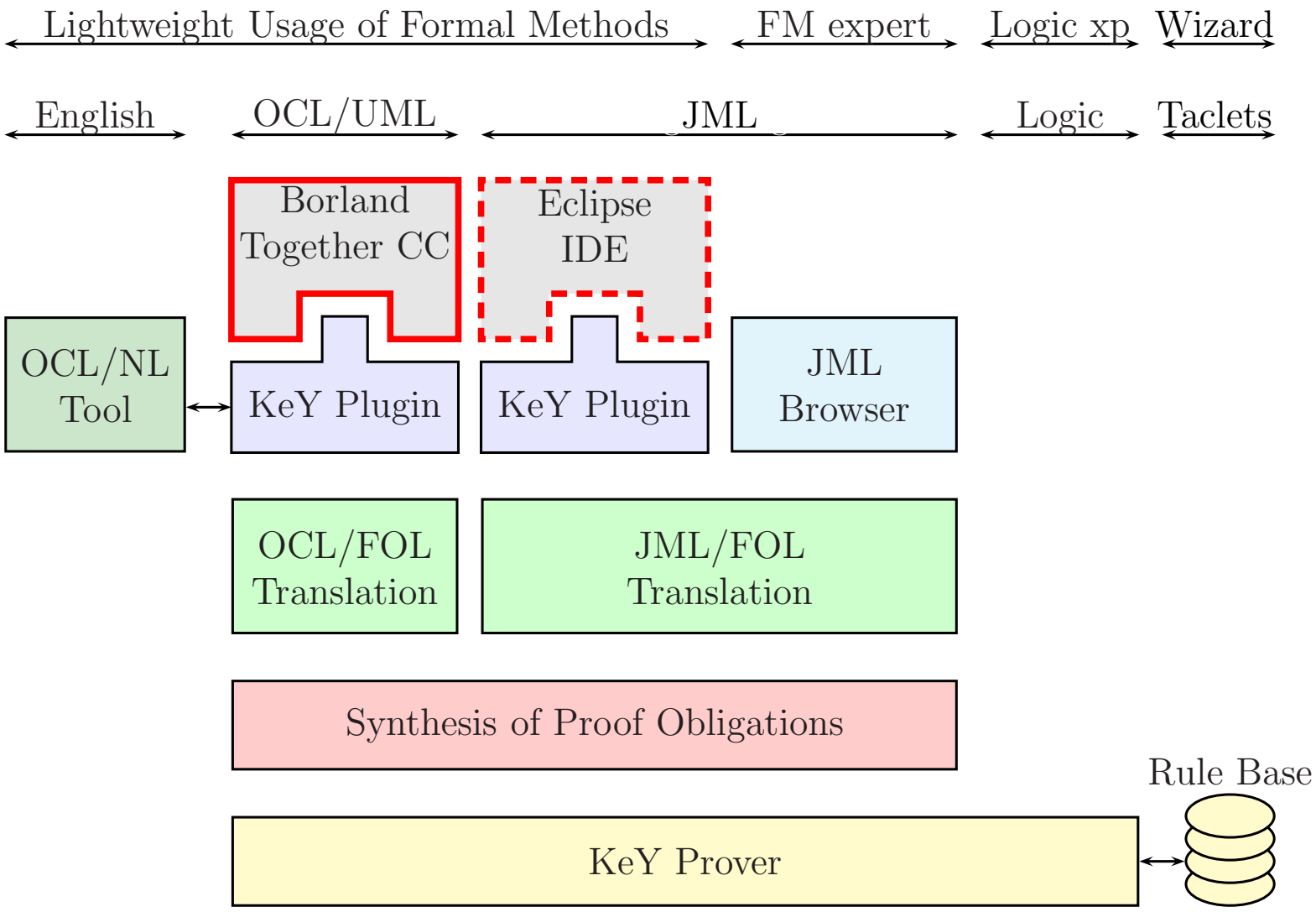
Conventional CASE-Based Development



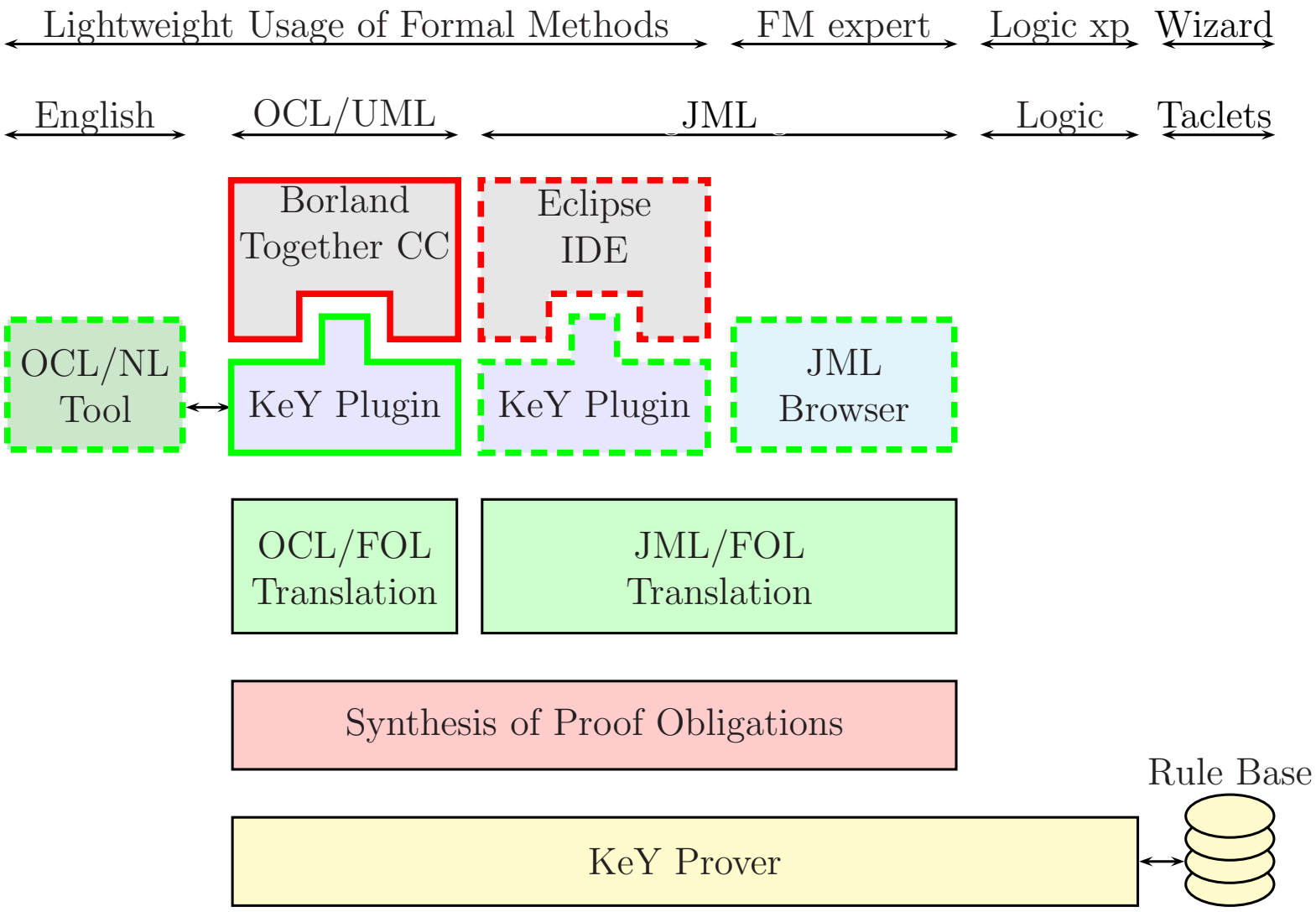
Conventional CASE-Based Development



The KeY System



The KeY System — Formal Specification



Specification Language: OCL

UML has textual specification language as sub-standard:

Object Constraint Language (OCL)

- **OOP-like syntax, ASCII**
- **designed for easy navigation within UML class diagrams, etc.**
- **strongly typed**
- **formal semantics: translation to typed FOL**

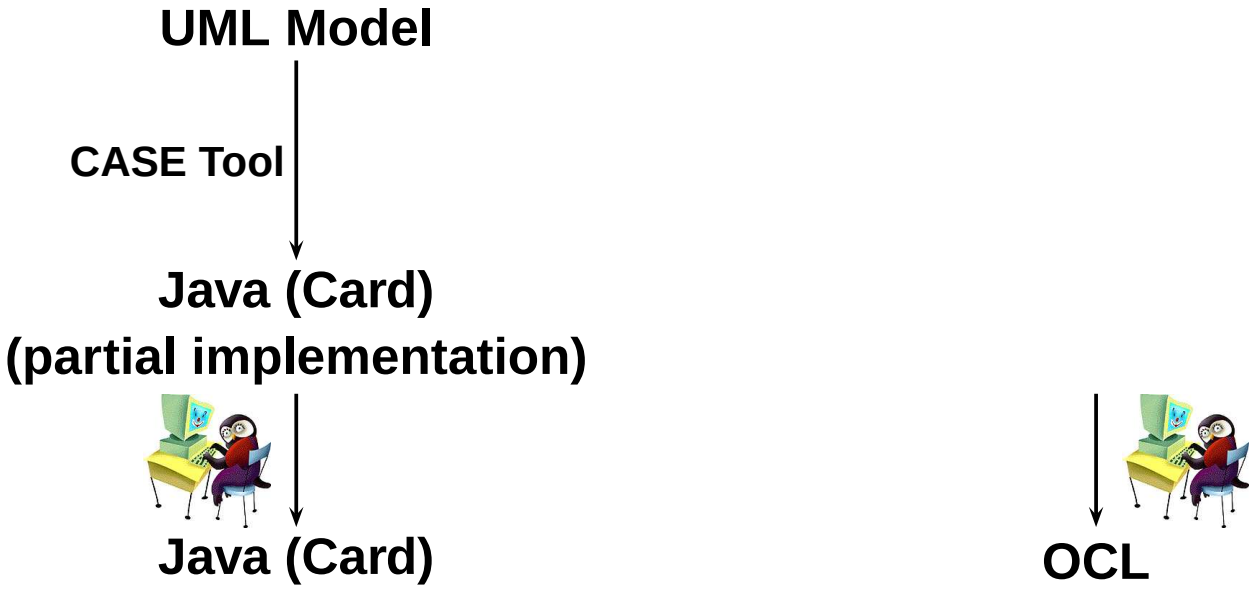
OCL expressions reduce legal instances of underlying UML diagram

- **Class invariants**
- **Pre-/postconditions of operations and methods**

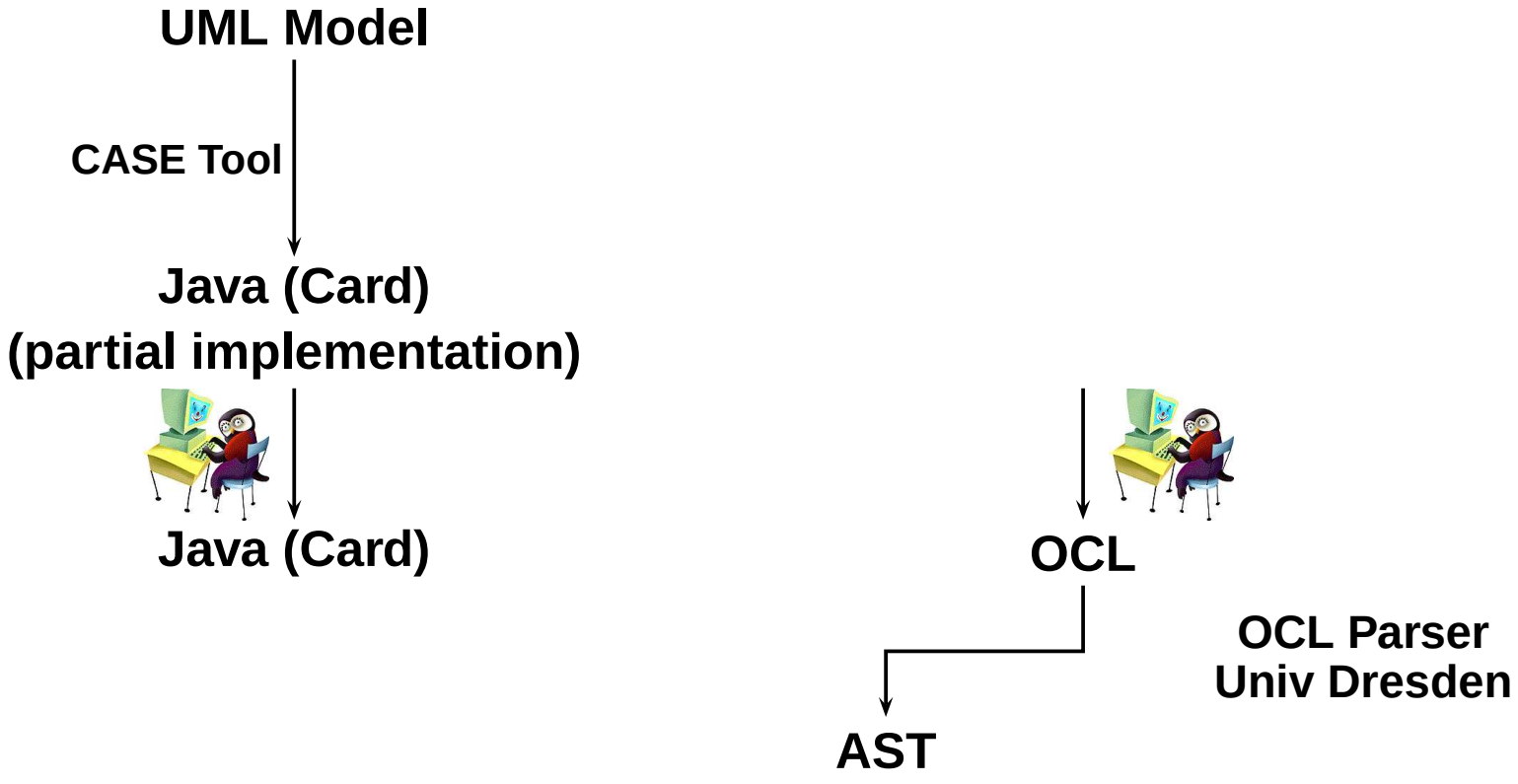
Permits formal specification of functional requirements

Other specification languages: *Alloy*, *JML*, *RSL*, . . .

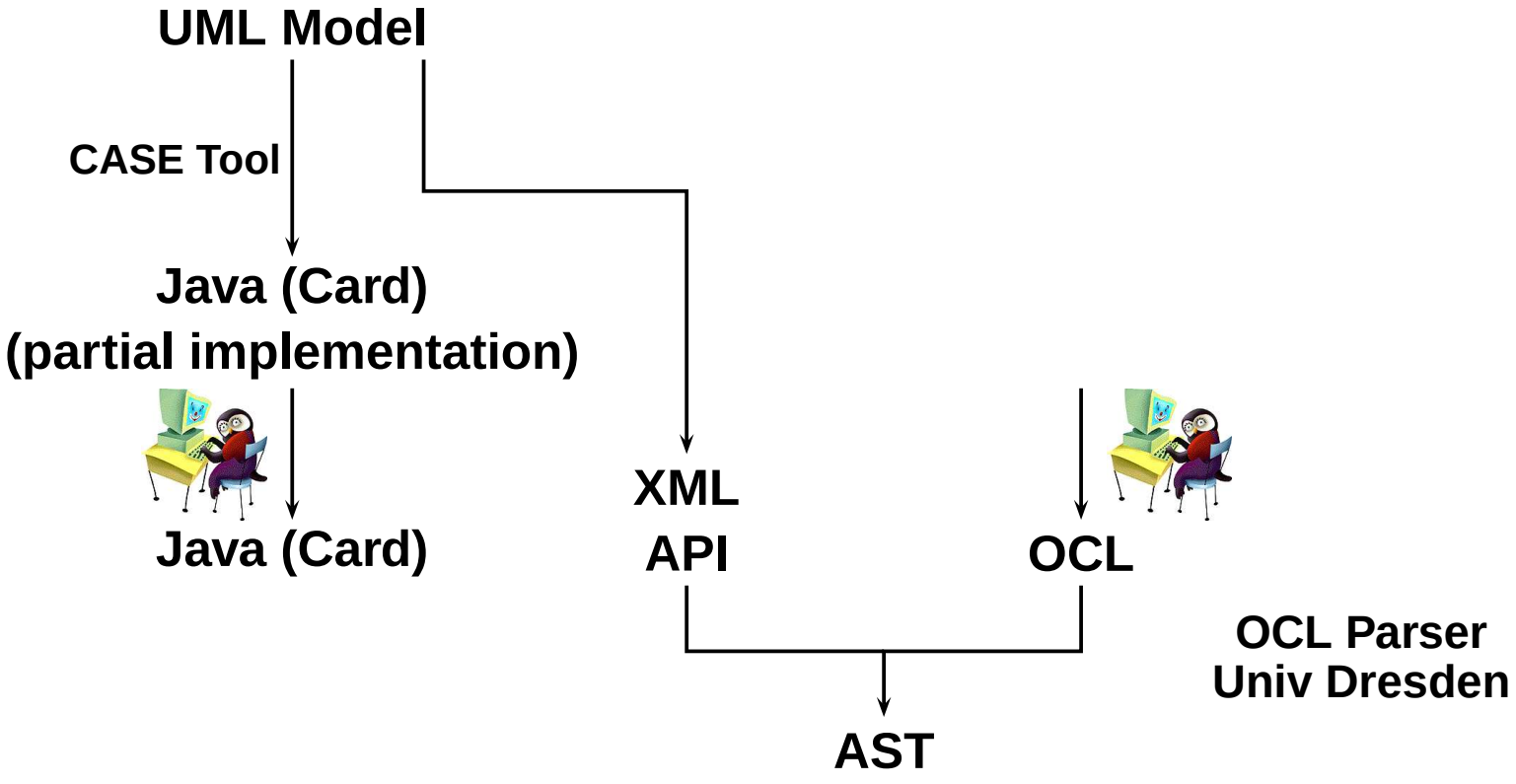
Formal Specification in OCL



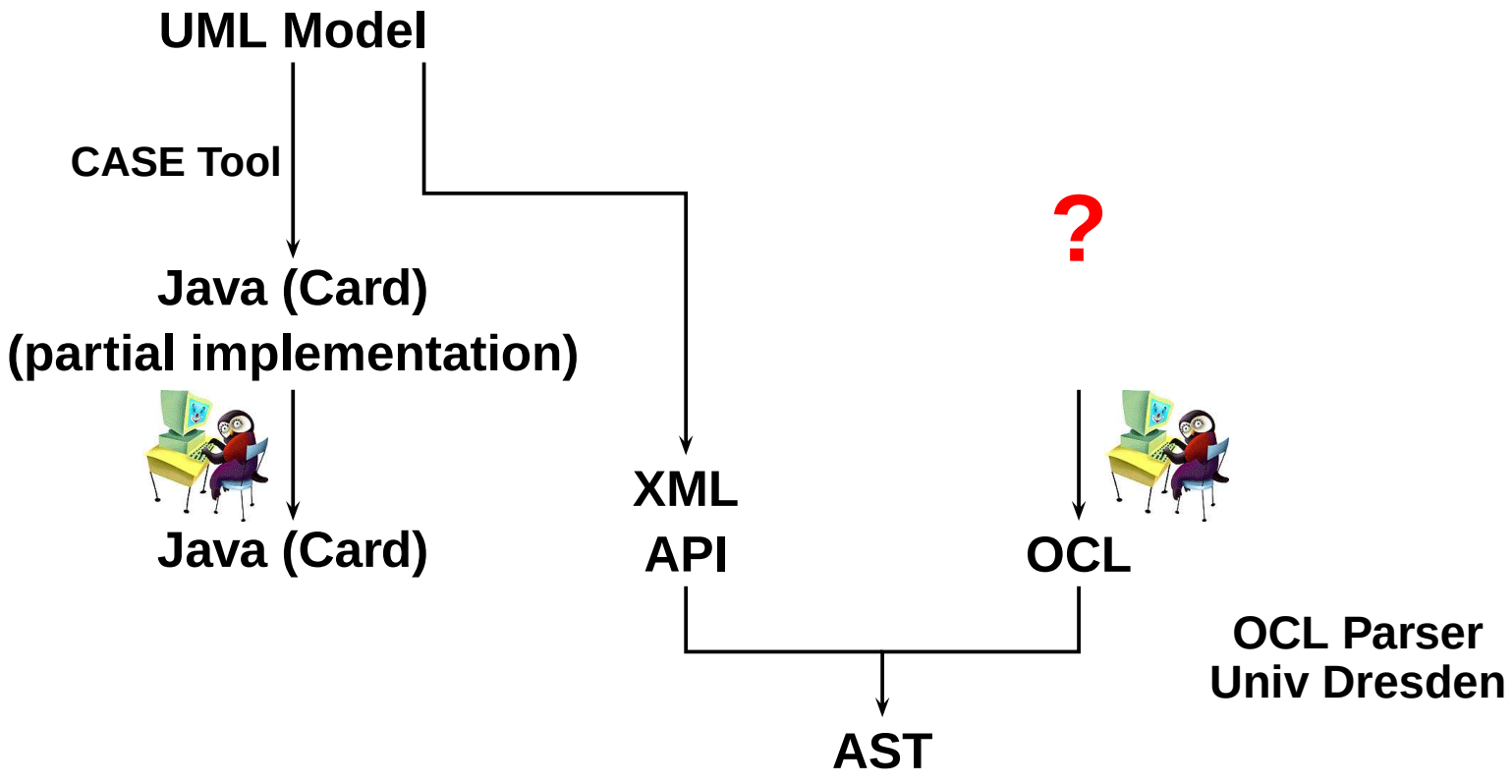
Parsing OCL


UML Model
CASE Tool
Java (Card)
(partial implementation)
Java (Card)
OCL
OCL Parser
Univ Dresden
AST

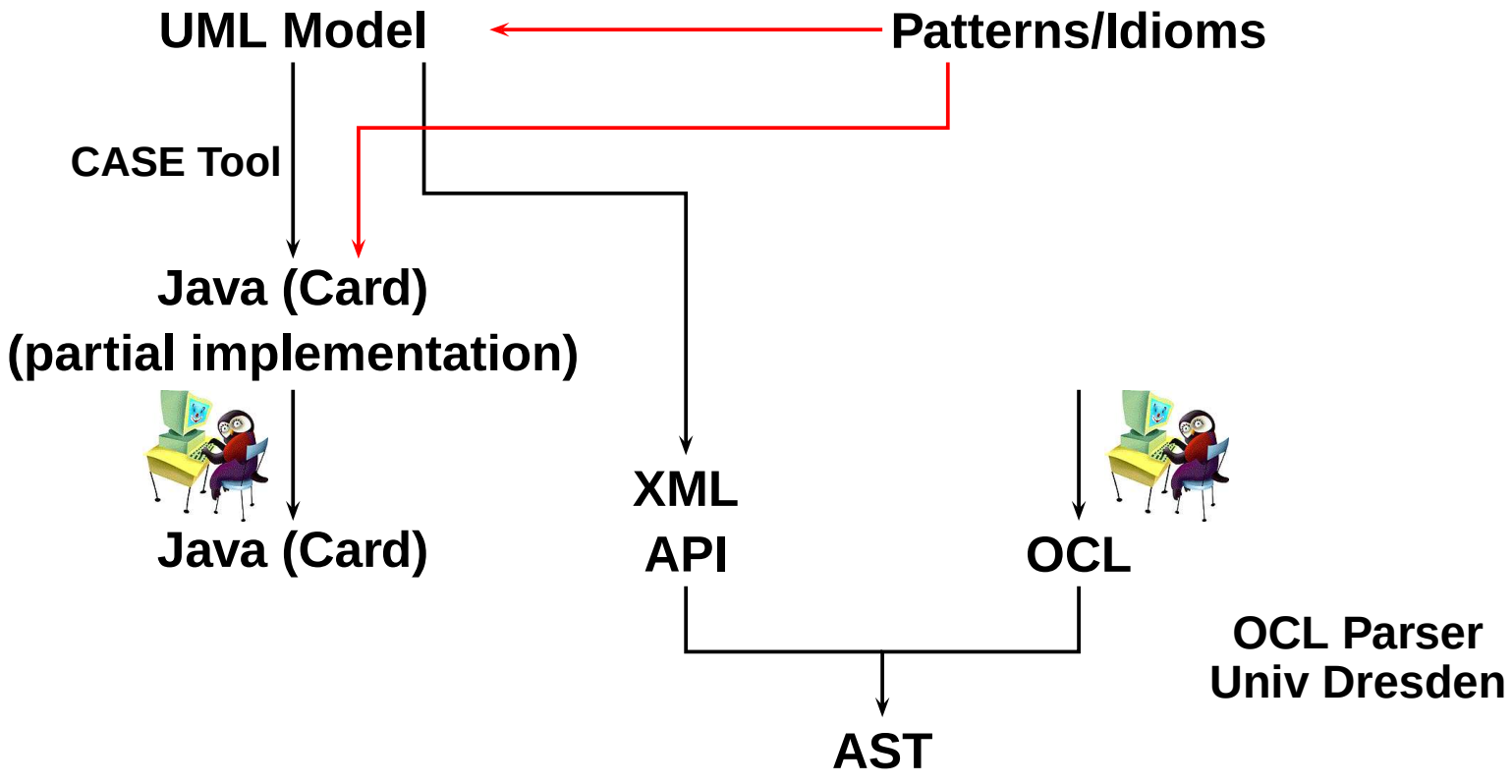
Parsing OCL


UML Model
CASE Tool
Java (Card)
(partial implementation)
Java (Card)
XML
API
OCL
AST
OCL Parser
Univ Dresden

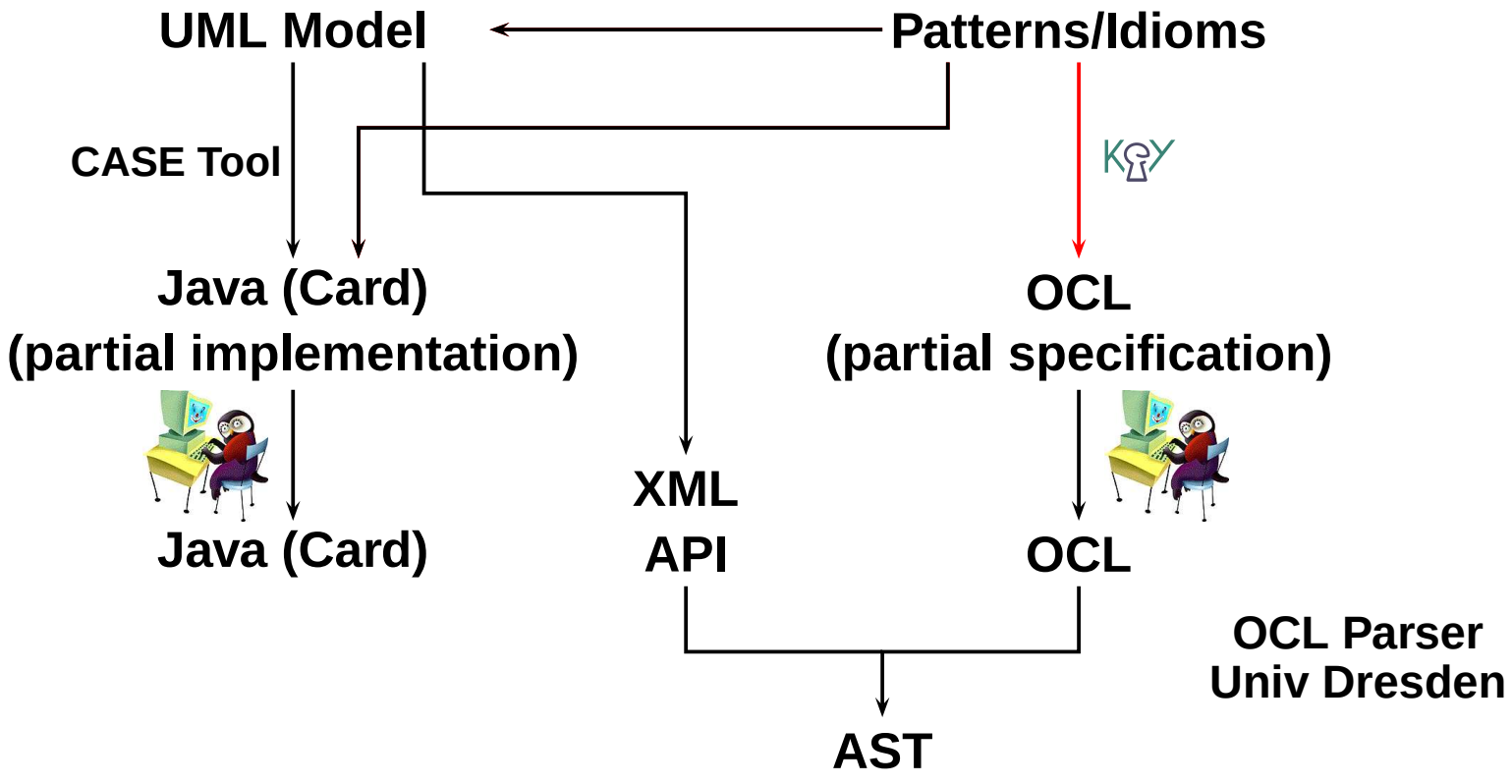
Where to Take Formal Specification from?



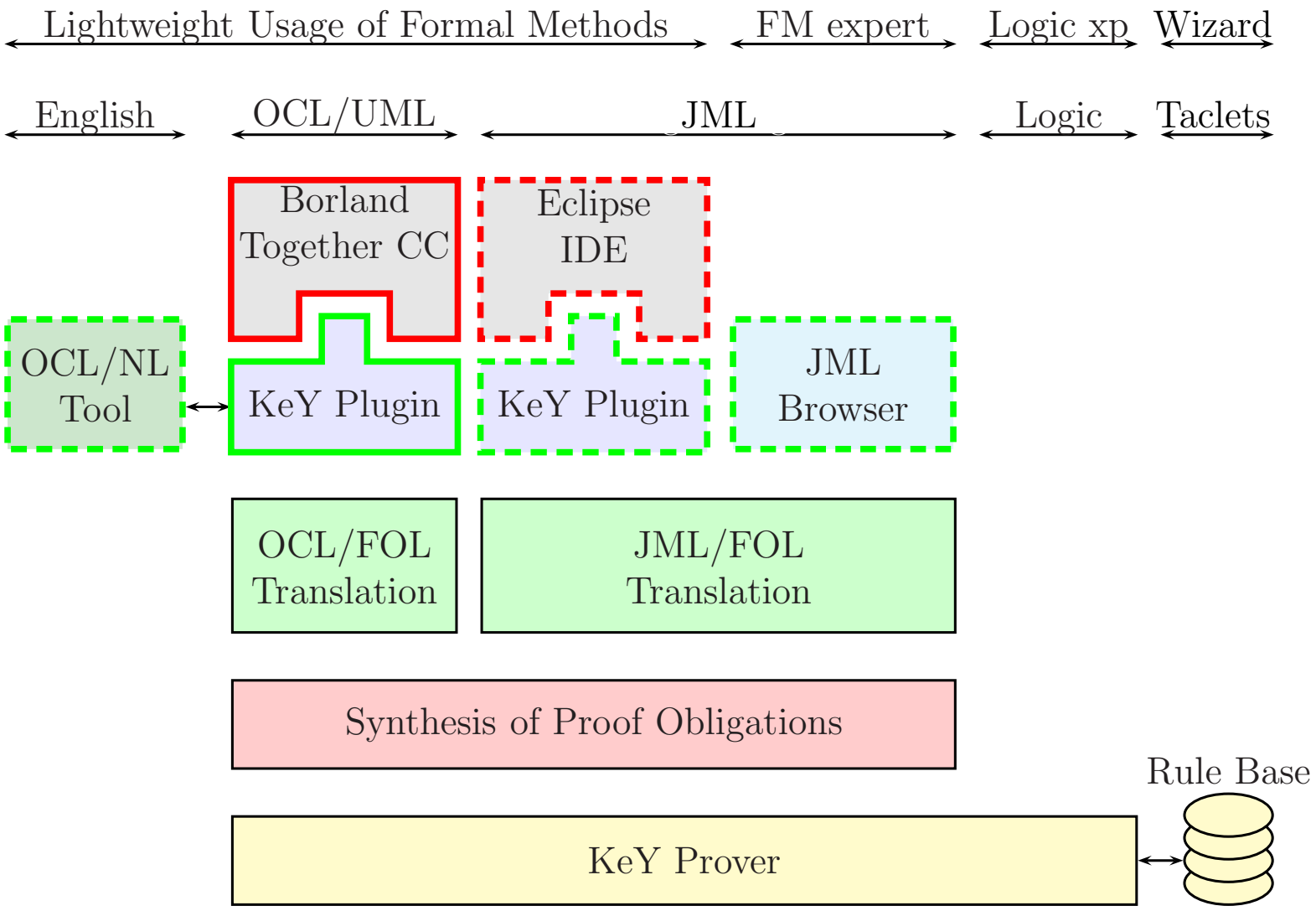
(Library of Patterns and Idioms)



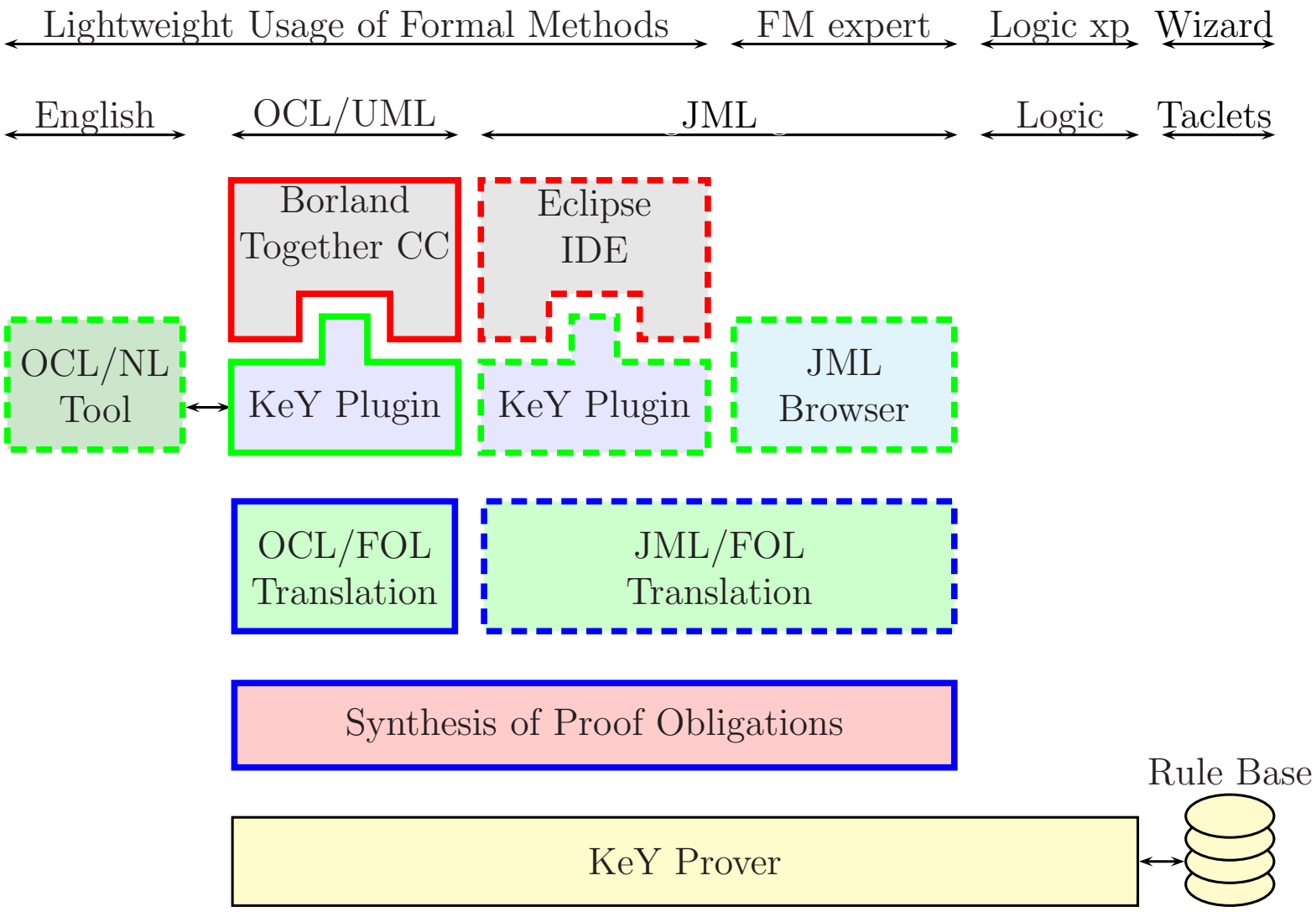
(KeY OCL Library)



The KeY System



The KeY System — Verification



Tasks of Verification Component

Tasks of Verification Component

- **Translation OCL $\Rightarrow$ logic**

Need to simplify resulting formulas

Tasks of Verification Component

- **Translation OCL $\Rightarrow$ logic**
- **Synthesize FOL formulas from horizontal verification tasks**

Eg, structural subtyping

Tasks of Verification Component

- **Translation OCL $\Rightarrow$ logic**
- **Synthesize FOL formulas from horizontal verification tasks**
- **Synthesize Java Card DL formulas from vertical verification tasks**

Eg, method invariants, total correctness

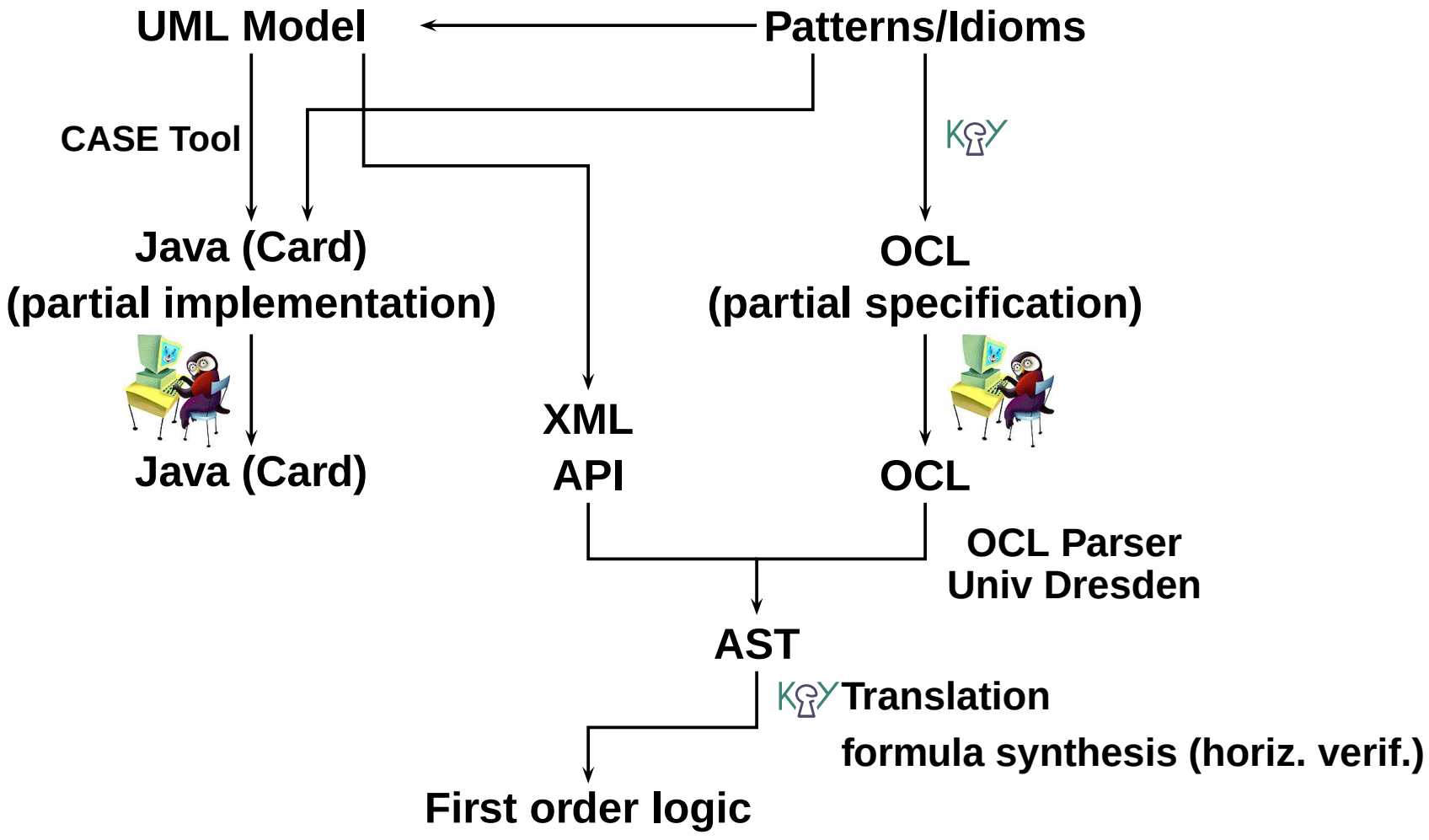
Tasks of Verification Component

- **Translation OCL $\Rightarrow$ logic**
- **Synthesize FOL formulas from horizontal verification tasks**
- **Synthesize Java Card DL formulas from vertical verification tasks**
- **Driving the deduction component**

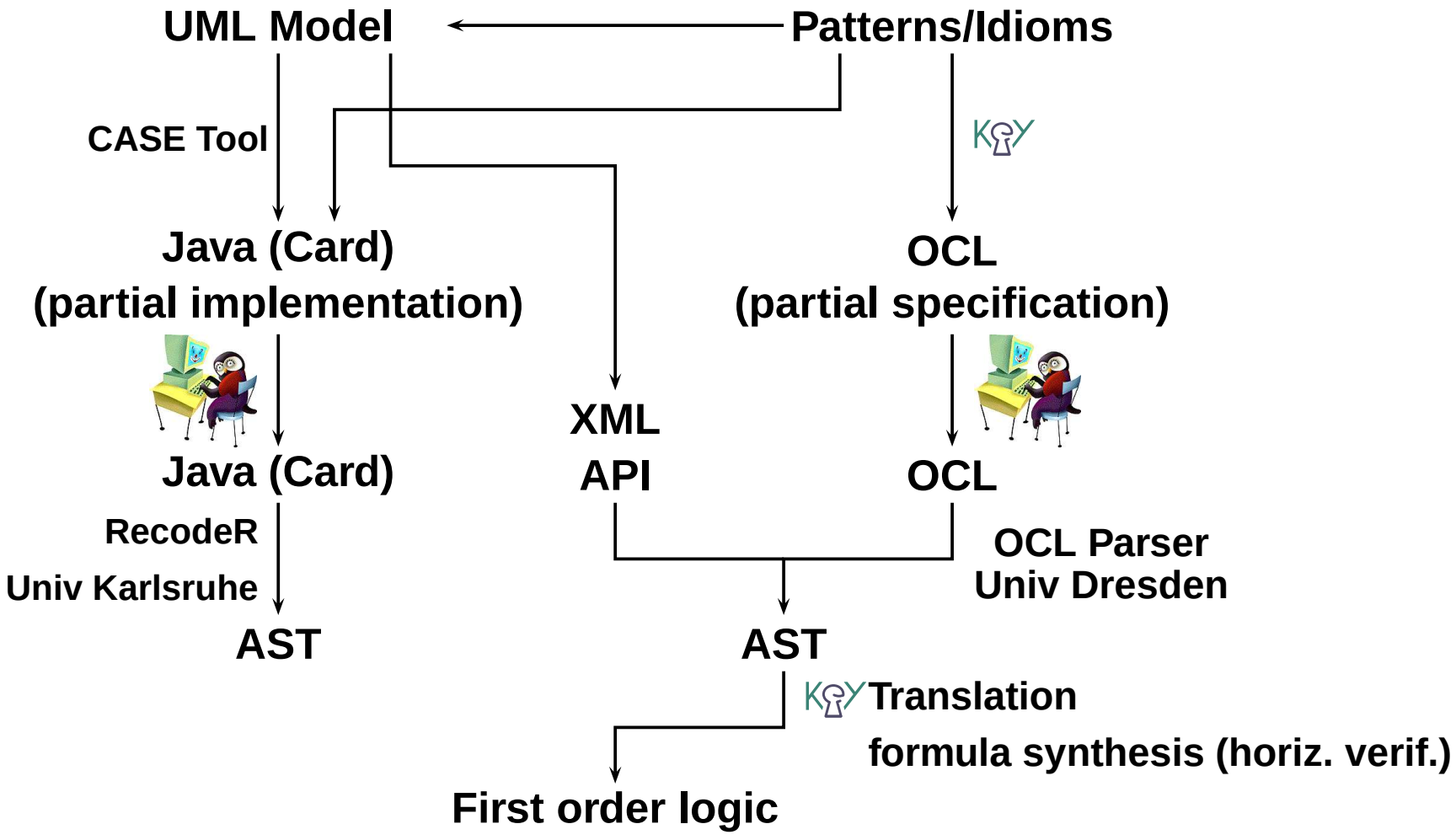
Tasks of Verification Component

- **Translation OCL $\Rightarrow$ logic**
- **Synthesize FOL formulas from horizontal verification tasks**
- **Synthesize Java Card DL formulas from vertical verification tasks**
- **Driving the deduction component**
- **Correctness management**

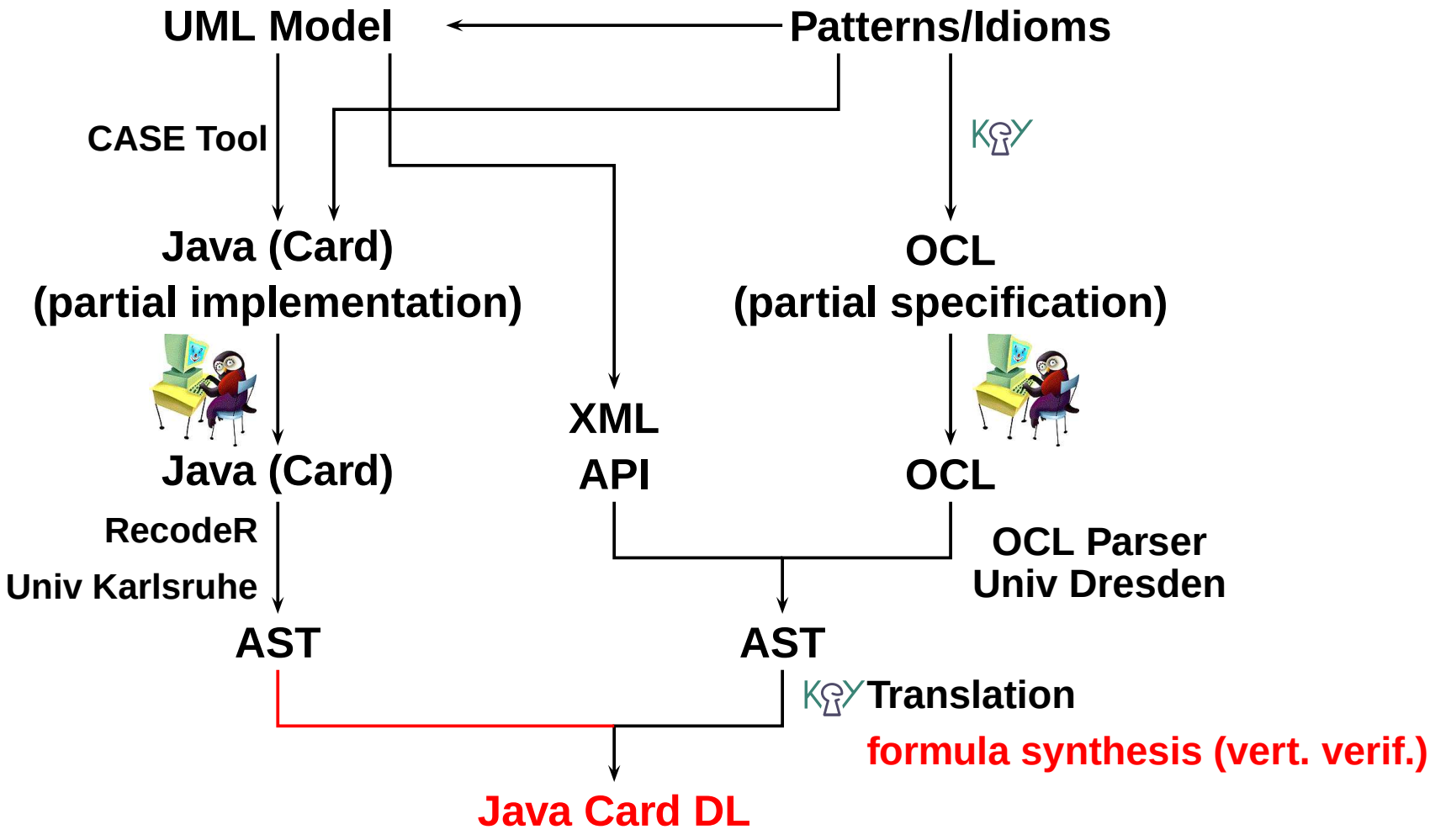
Translation, Horizontal Verification



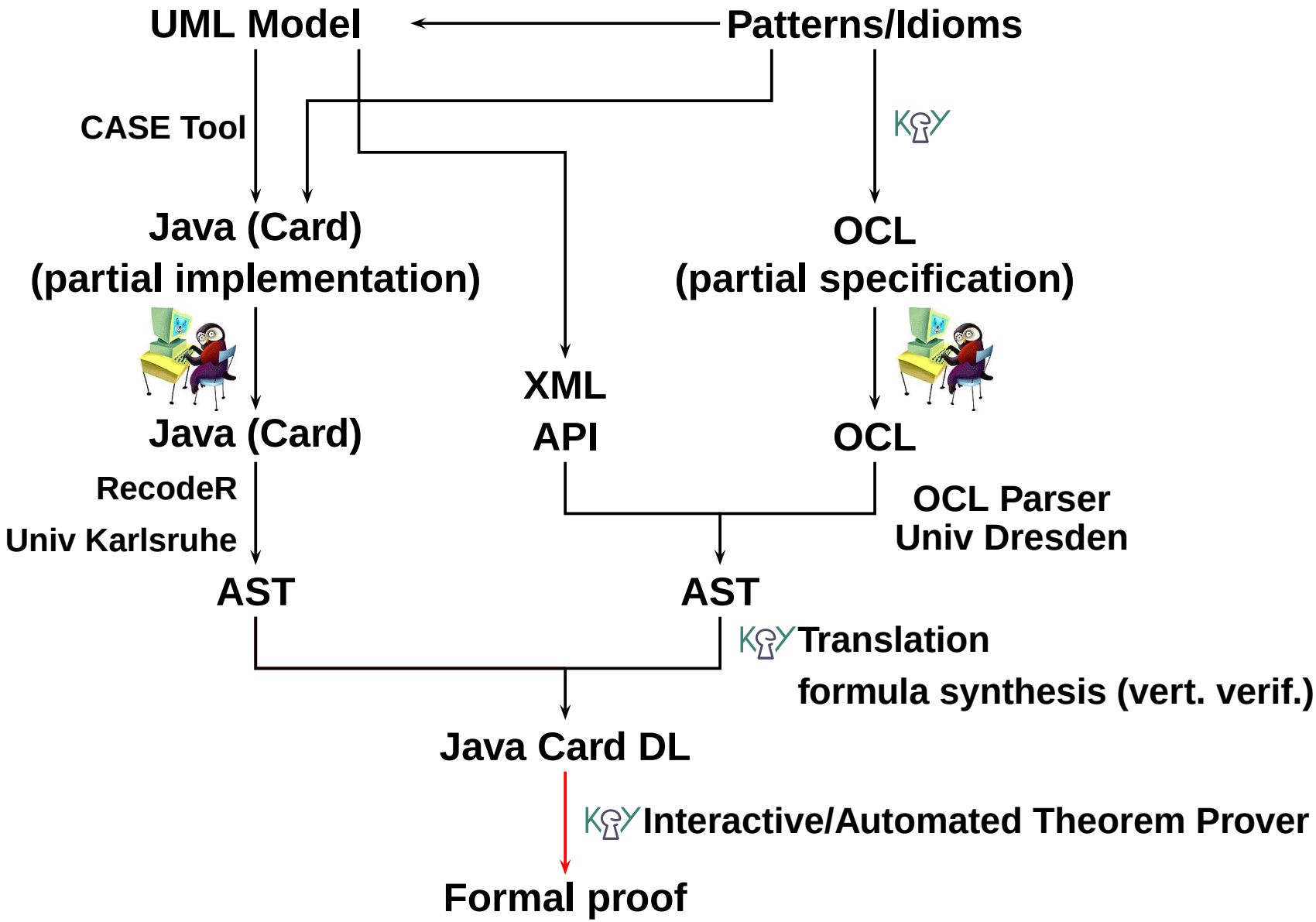
Parsing Java



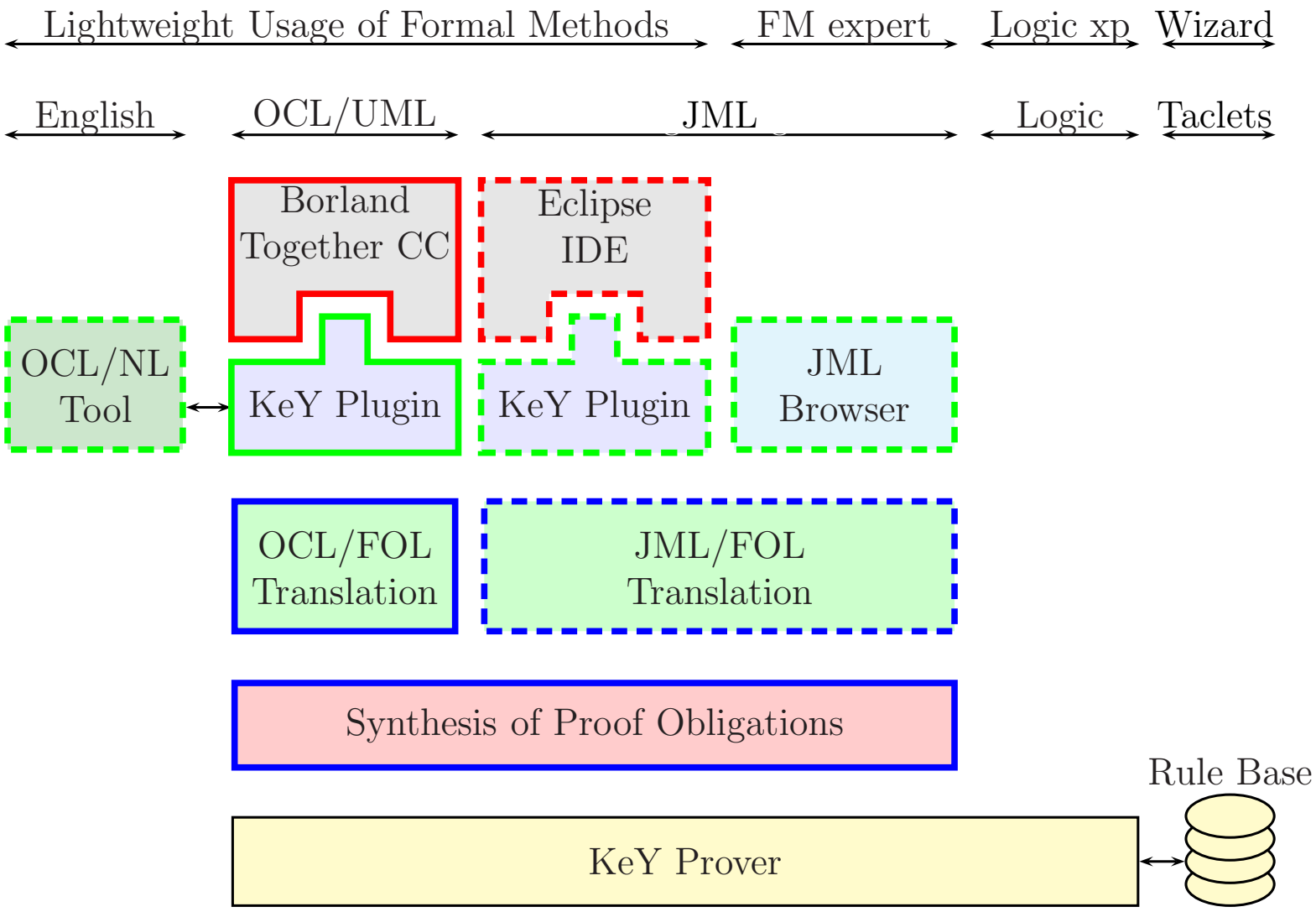
Translation, Vertical Verification



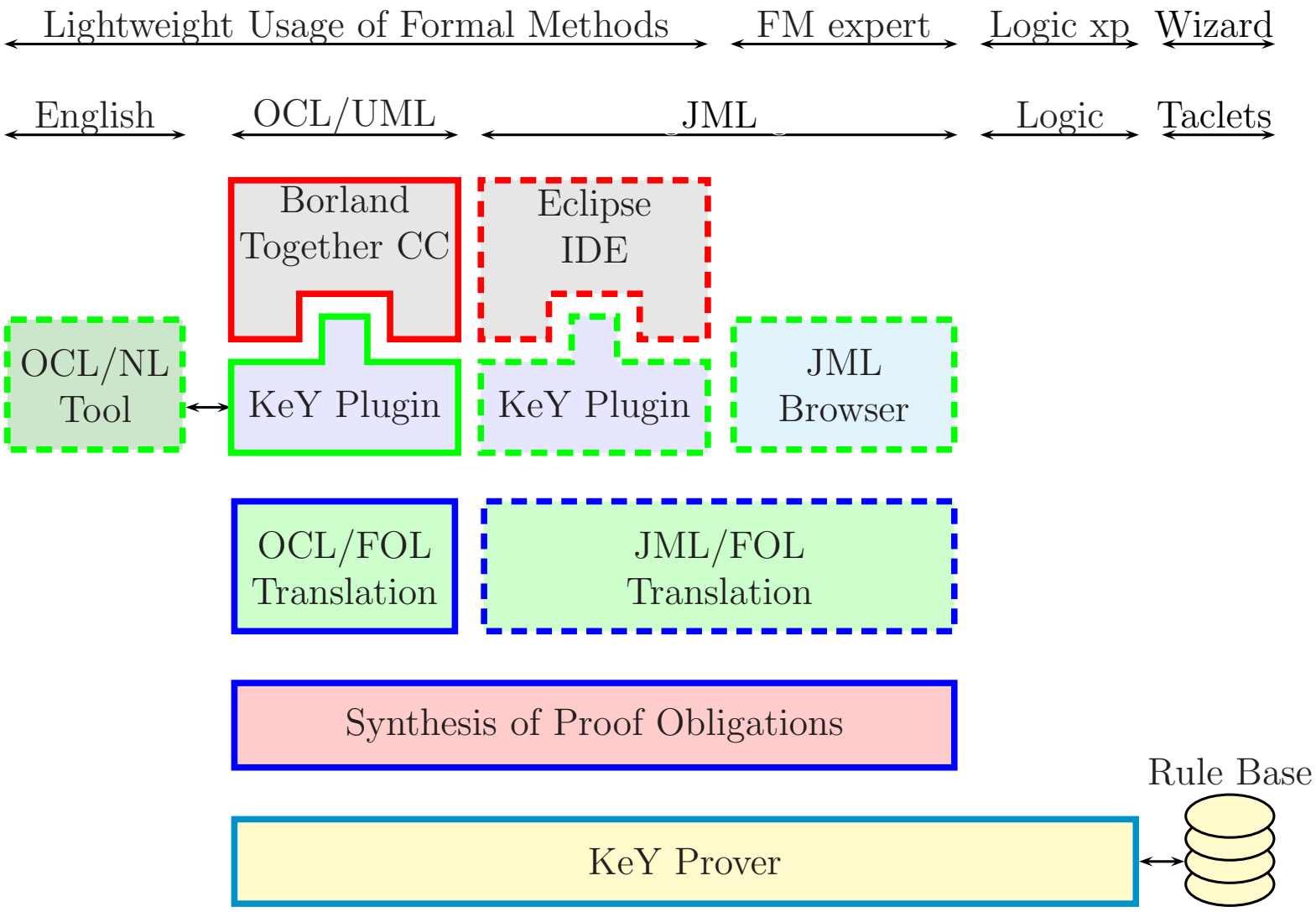
Deductive Verification



The KeY System



The KeY System — Proving



The Program Logic: Dynamic Logic

Syntax

- **Modal operators $[p]$ and $\langle p \rangle$ for each Java (Card) program p**
- **Statements about *final state* of p**

The Program Logic: Dynamic Logic

Syntax

- **Modal operators $[p]$ and $\langle p \rangle$ for each Java (Card) program p**
- **Statements about *final state* of p**

Semantics

- $[p]\ F$: **If p terminates normally, then F holds in the final state**
  (partial correctness)
- $\langle p \rangle\ F$: **p terminates normally *and* F holds in the final state**
  (total correctness)

Java vs. Java Card

Java features that are not present in Java Card

Java vs. Java Card

Java features that are not present in Java Card

- **Threads**

  unrestricted concurrency

Java vs. Java Card

Java features that are not present in Java Card

- **Threads**

  unrestricted concurrency

- **Floating point arithmetic**

  IEEE standard 754 is huge . . .

Java vs. Java Card

Java features that are not present in Java Card

- **Threads**

  unrestricted concurrency

- **Floating point arithmetic**

  IEEE standard 754 is huge . . .

- **Dynamic class loading**

  Implementation must be known before verification

Java vs. Java Card

Java features that are not present in Java Card

- **Threads**

  unrestricted concurrency

- **Floating point arithmetic**

  IEEE standard 754 is huge . . .

- **Dynamic class loading**

  Implementation must be known before verification

- **Graphical/buffered I/O**

  formal specification Swing classes?

Java vs. Java Card

Java features that are not present in Java Card

- **Threads**

  unrestricted concurrency

- **Floating point arithmetic**

  IEEE standard 754 is huge . . .

- **Dynamic class loading**

  Implementation must be known before verification

- **Graphical/buffered I/O**

  formal specification Swing classes?

□