Plugins for the Isabelle Platform:

A Perspective for Logically Safe, Extensible, Powerful and Interactive Formal Method Tools

Burkhart Wolff

Université Paris-Sud

(Technical Advice by: Makarius Wenzel, Université Paris-Sud)

What I am not Talking About

What I am not Talking About

Isabelle as:

"Proof - Assistent"

or

"Theorem Prover"

What I will Talk About

Isabelle as:

Formal Methods Tool Framework

What I will Talk About

Isabelle as:

Formal Methods Tool
Framework

"The ECLIPSE of FM - Tools"

Overview

- Three Histories

Overview

- Three Histories
  - Evolution of the ITP Programme and
    Evolution of the Isabelle - Architecture
  - Evolution of Isabelle - LCF - Kernels
  - Evolution of Tools built upon Isabelle

The ITP Research Programme
and
The Evolution of the
Isabelle/Architecture

The "Interactive Proof" Research Programme

- 1968 : Automath
- 1975 : Stanford LCF
  LISP based Goal-Stack, orientation vs. functional Programming, Invention: Parametric Polymorphism
- 1979 : Edinburgh LCF
- 1984/5 : Cambridge LCF: core LCF principles (1) an abstract type of theorems a (2) tactics that deliver a validation in the form of a function from a theorem list to a theorem.

Historic Overviews:

http://www.cambridge.org/catalogue/catalogue.asp?ISBN=9780521395601
http://www.cl.cam.ac.uk/~mjcg/papers/HolHistory.pdf

The "Interactive Proof" Research Programme

- 1986-88 : HOL88, Isabelle, Coq

  Further search to more foundational and logically safe systems lead to abandon of LCF; HOL became replacement.

  Invention: Basic Embedding Techniques

  Invention: Coq: Dependent types, proofobjects

  Invention: HOL: recursion embeddable,
  datatype packages,
  semantics & conservativity

  Invention: Isabelle: Meta-Logic,
  tactics as relations over thm's,
  Meta-Variables, HO Unification,
  explicit global context (thy's) in thm's and goal's ...

The “Interactive Proof” Research Programme

- 1990-95 : HOL88, HOL4, Isabelle, Coq,
  Maturing of “classic style”,
  search for more auomation

  Invention: Coq: Powerful Module Systems

  Invention: HOL: executable “formulas”
  meson-tac,
  embedding CSP with FP

  Invention: Isabelle: LF, Cube, FOL, ZF, (HOL)
  higher-order rewriter,
  tableaux prover

The "Interactive Proof" Research Programme

- 1995-00 : HOL4, Isabelle, Coq, HOL-light
  Back to more basics again ...
  and more power and framework, too

  Invention: Isabelle:
  Class-type System,
  proof objects (Isabelle 96
  Workshop !!!)
  auto (combined reasoners)

  Invention: Isabelle:
  embedding HOLCF, HOL definitively
  superseded LCF. ProofGeneral.

The "Interactive Proof" Research Programme

- 2000-05 : Isabelle, HOL-light
  Back to more basics again ...
  and more power and framework, too

  Invention: HOL-Light
  Real-number theories & IEEE754,
  Groebner Basis tactics, ...

  Invention: Isabelle:
  ISAR-engine, Proof Documents
  context (state) replaces "theory"
  integration of ATP via
  Proof Objects

The "Interactive Proof" Research Programme

- 2005-10 : Isabelle, HOL-light

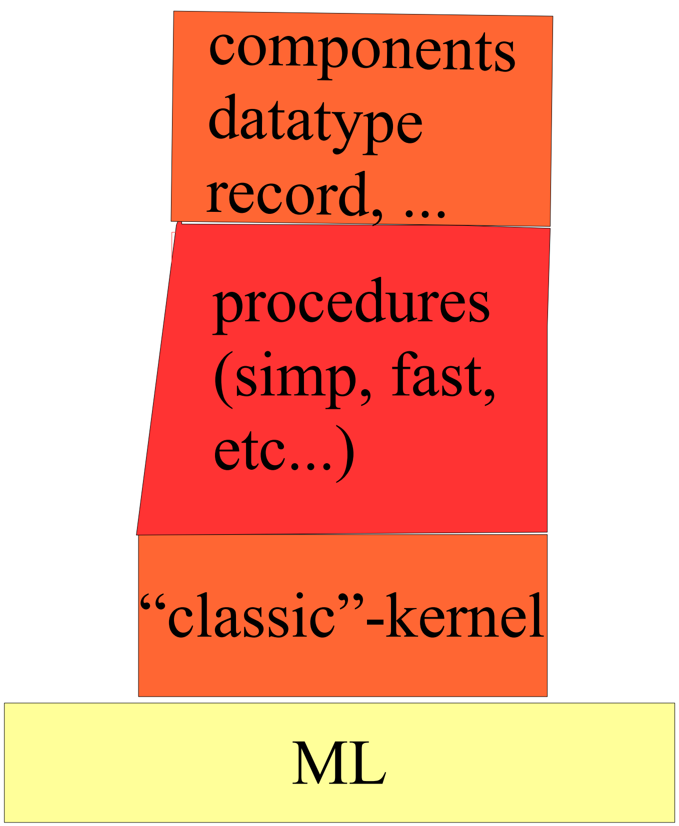
  Back to more basics again ...
  and more power and framework, too

  Invention: HOL-Light
  - Formal Verification of Kernel (without Conservativity)

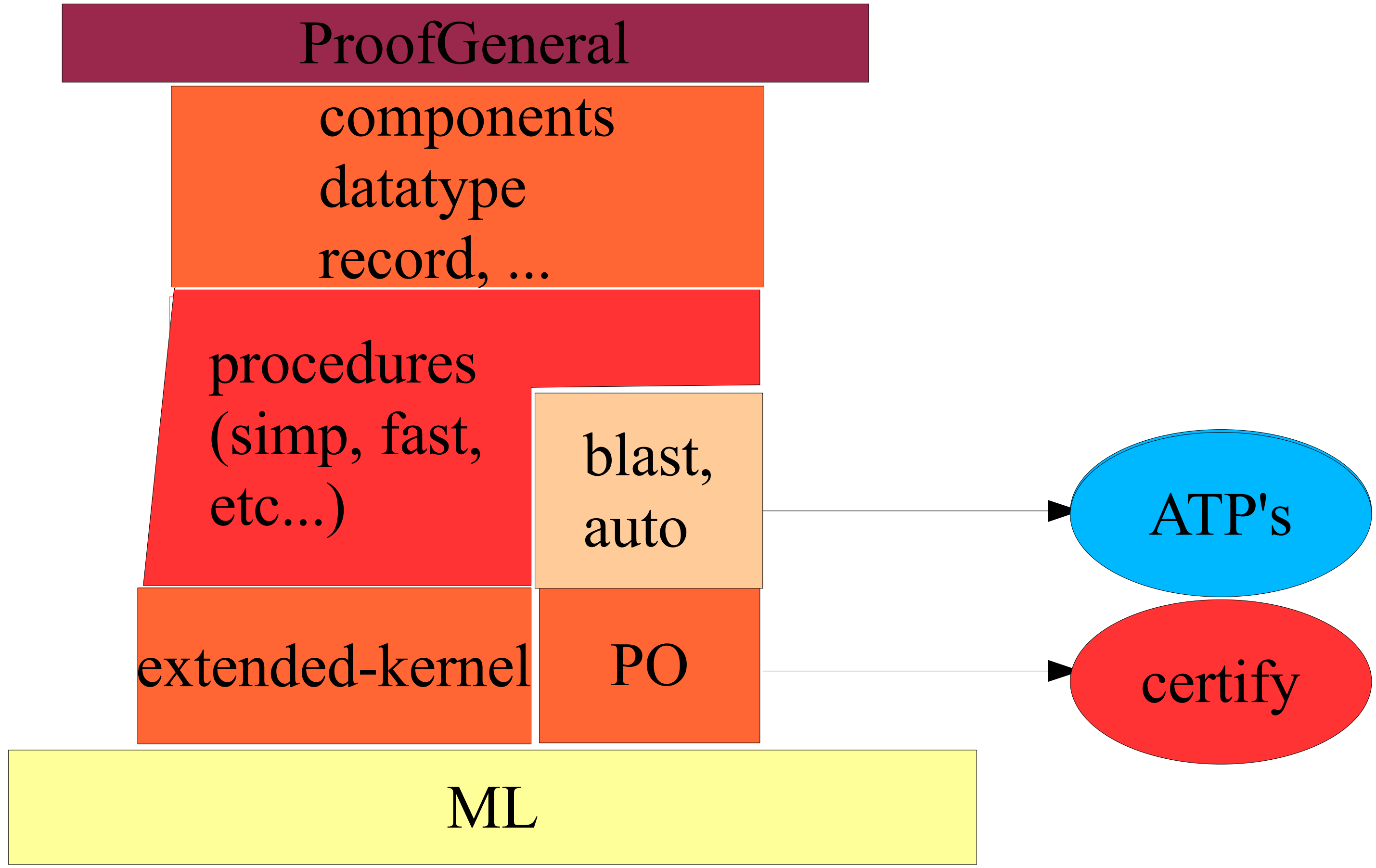
  Invention: Isabelle:
  - Tools: C0, Simpl,
    TestGen, HOL-Z, HOL-OCL,
    HOL-Boogie,

Evolving Isabelle Architecture (86)

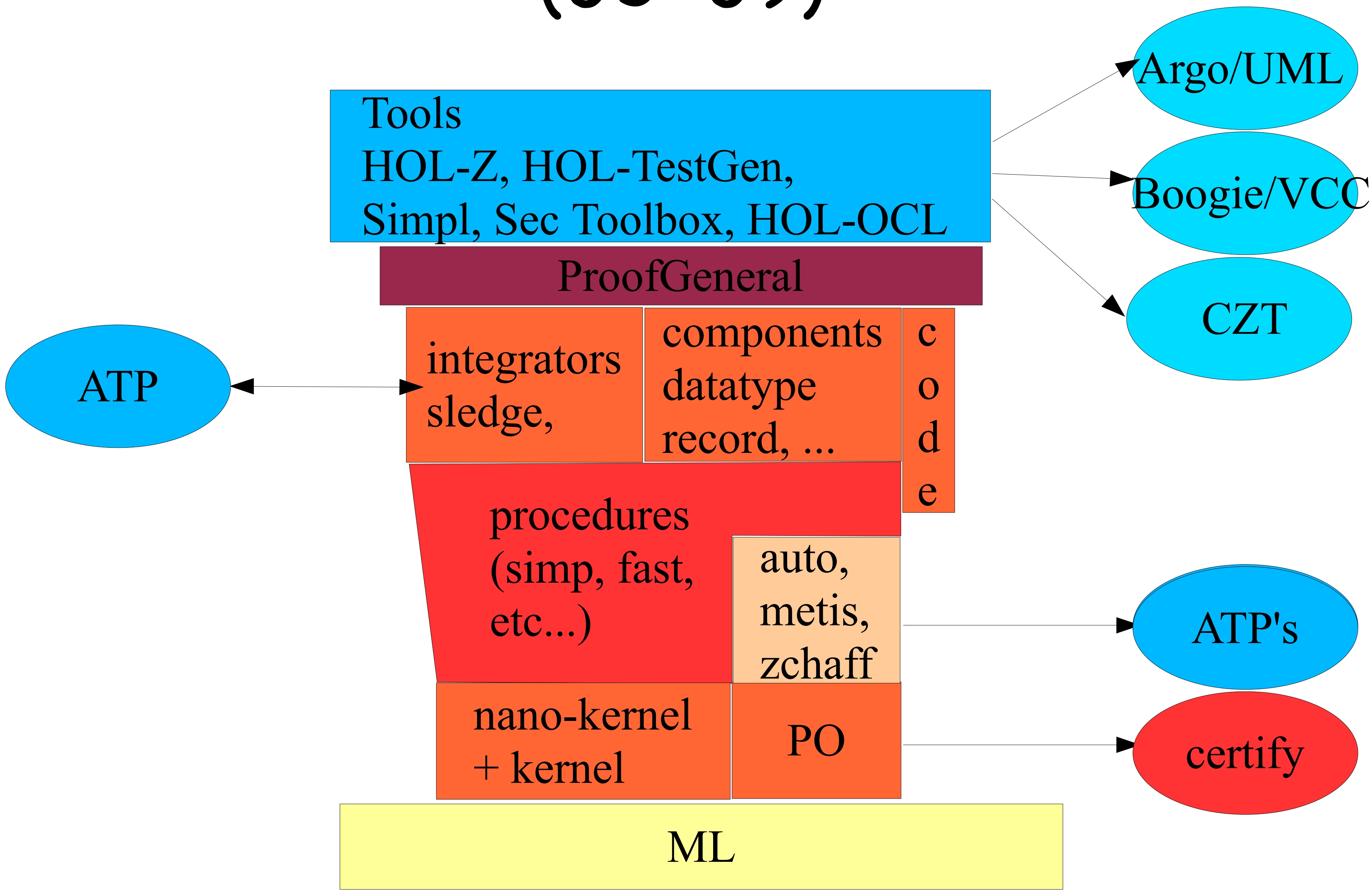
Evolving Isabelle Architecture (89)



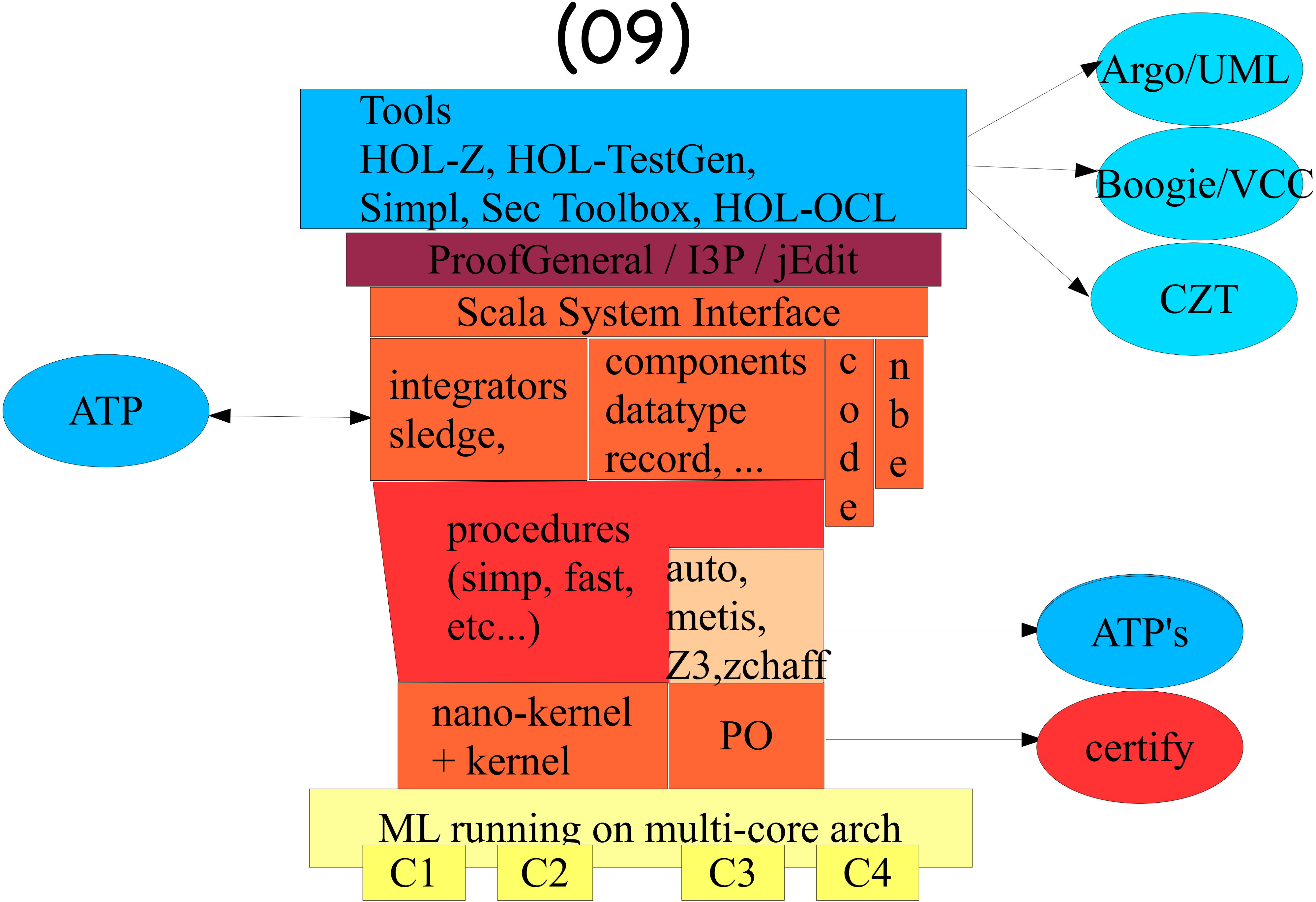
Evolving Isabelle Architecture (98-05)



Evolving Isabelle Architecture (05-09)



Evolving Isabelle Architecture (09)



The Evolution of

Isabelle - Kernels

The Classical LCF Kernel:

Coarse grained global context transition with branch and merge
(Edinburg LCF, HOL88?, Isabelle 89 ... 94-4, ...)

$$\Gamma \vdash_{\Theta} \varphi$$

Meaning: φ can be derived from Γ in the global context Θ

where:

Γ : local context, assumptions, premisses, ...
φ : conclusion
Θ: global context, the „theory“ (Σ,A)consisting
of the „signature Σ“ and the „Axioms A“

The Classical LCF Kernel:

Coarse grained global context transition with branch and merge
(Edinburgh LCF, HOL88?, Isabelle 89 ... 94-4, ...)

„Θ“ thy = { ancestors : thy list ,
sign : Signature ,
axms : thm list}

„$\Gamma \vdash_{\Theta} \varphi$“ thm = {context : thy,
hyps : term list,
prop : term}

$\_ \subseteq \_$ subthy : thy $*$ thy => bool

Invariant: $\subseteq$ is a partial ordering (no cycles)

The inclusion ordering $\subseteq$ is critically used for the transfer of judgements („thm“s):

$$\Gamma \vdash_{\Theta 1} \varphi \quad \text{implies} \quad \Gamma \vdash_{\Theta 2} \varphi \qquad \text{if } \Theta_1 \subseteq \Theta_2$$

The Classical LCF Kernel:

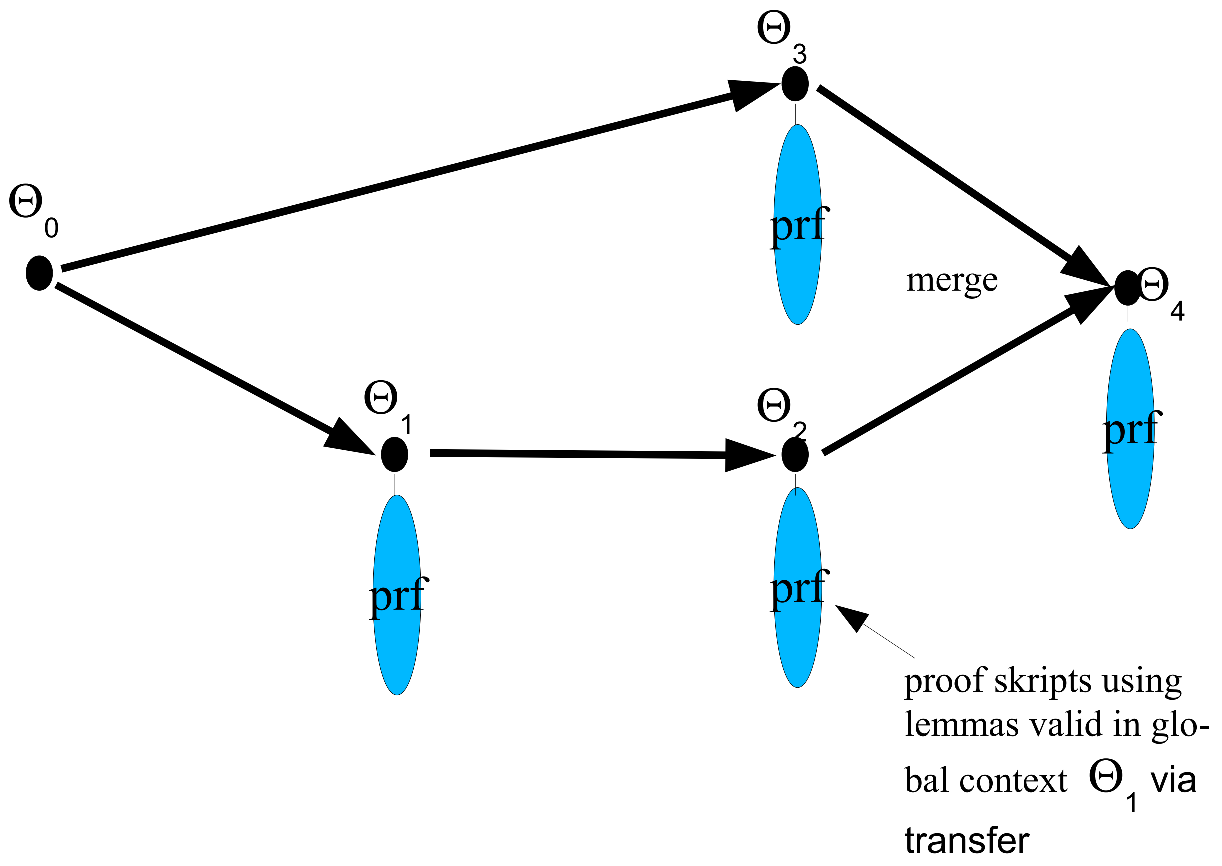
Typical Programming Interface

„$\varphi \vdash_{\Theta} \varphi$“	trivial Θ „φ“ :: thm
„$\Gamma \vdash_{\Theta} \varphi \ \{\xi \mapsto E\}$“	instantiate:: ... => thm => thm
„forward-chaining“	implies_elim :: thm => thm => thm
„backward-chaining“	type tactic = thm => seq thm
	rtac , etac, dtac, ...

In Cambridge LCF: elementary rules of the HOL-logic as basic operators on thm's, in Isabelle the elementary rules of an intuitionistic fragment of HOL called „Pure“

The Classical LCF Kernel:

Coarse grained global context transition with branch and merge
(Isabelle 89 ... 94-4, ...)



The Classical LCF Kernel:

Coarse grained global context transition with branch and merge
(Isabelle 89 ... 94-4, ...)

Explicit proof contexts turn the Kernel into a “transaction machine” where the proofs can be executed interleaved (The following was essentially already possible in 98):

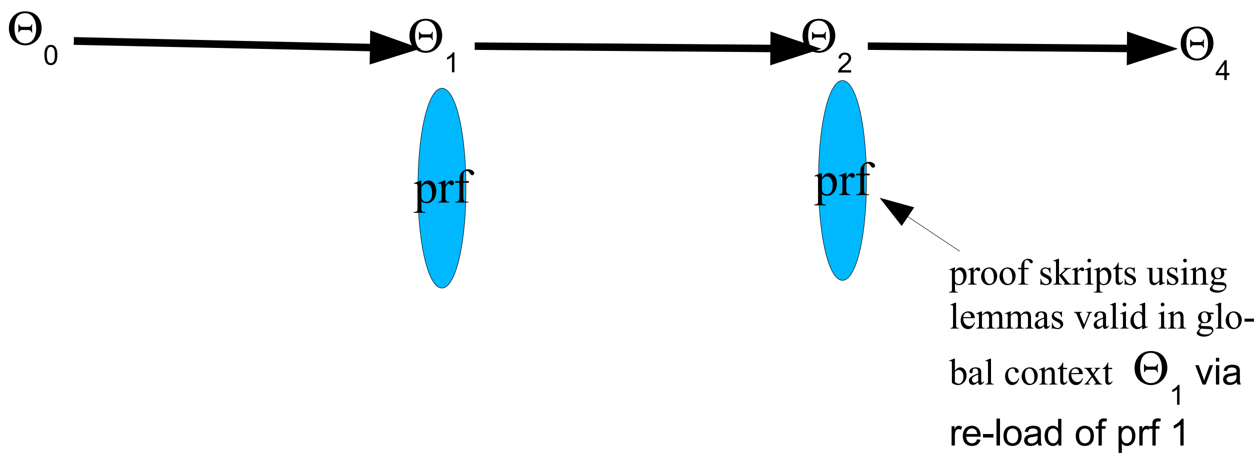
```
goal A.thy “<lemma1>”
by(rtac …) by(dtac … )
val P1 = push_proof ()

goal B.thy “<lemma1>”
by(dtac … )
val P2 = push_proof ()

pop_proof(P1)
by(simp_tac …)
val thm1 = result()

pop_proof(P2)
by(simp_tac …)
val thm2 = result()
```

Comparison: The "Minimal" LCF Kernel:

Fine grained global context transition without branch and merge

Global Contexts implicit in the top-level ML shell

no transfer - import by reproving (HOL-light, HOL-88, HOL4)



The Extended LCF Kernel:

Internalising again the Name-Management and the plug-in Data into the Kernel
(ca. Isabelle 98, ...)

„Θ“ thy = {id:Id,
ancestors : thy list ,
sign: Signature,
axms: thm list,
...}

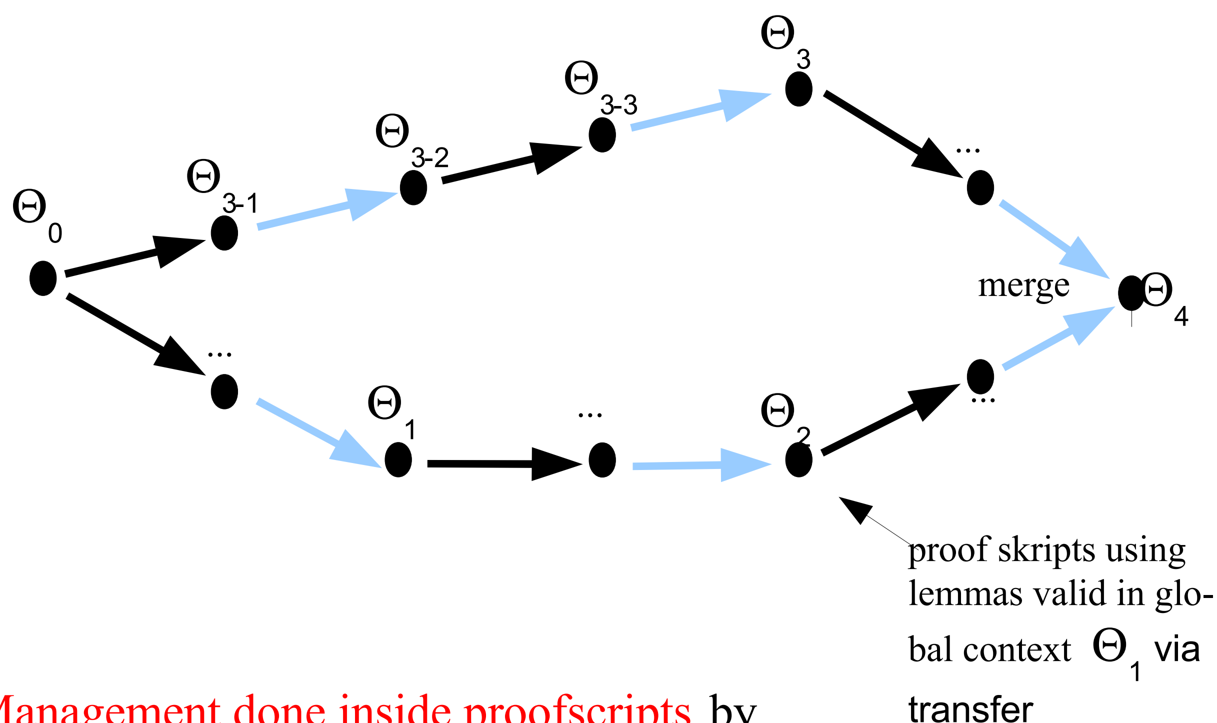
„$\Gamma \vdash_{\Theta} \varphi$“ thm = {context:thy,
hyps:term list,
prop:term}

„_⊆_“ subthy: thy × thy → bool

The Global Context becomes an „Extensible Record“ where Plugins can register their local state. (Used for configuration data of automated provers (simpset, claset, etc.), but rapidly for other stuff like a global Thm-Database, oracles, and proof-terms. Consequence: Plugin-Infrastructure with merge, provided that plugins were consequently parameterized wrt. Θ.

The Extended LCF Kernel:

fine-grained global context transition with branch and merge
proofs are global transitions, mixed with other extensions
(Isabelle 98, ..., but also Nano-Kernels Isabelle2005)



Name-Management done inside proofscripts by Global Context-Management, NOT by SML.
Requires get_thm(the_context(), „add_commute"),
later antiquotation „{@thm add_commute}" in proof scripts.
Mixture between Signature extensions and proofs facilitated programming of packages and automated provers.

The Extended LCF Kernel:

An Example at the Isar level:

```
theory AVL_def
imports Testing Main
begin
```

$\Theta_{3\text{-}0}$

```
datatype 'a tree = ET |  MKT 'a "'a tree" "'a tree"
```

$\Theta_{3\text{-}1}$

```
fun height :: "'a tree ⇒ nat"
where
  "height ET = 0"
| "height (MKT n l r) = 1 + max (height l) (height r)"
```

$\Theta_{3\text{-}2}$

```
fun is_in  :: " 'a ⇒ 'a tree ⇒ bool"
where
  "is_in k ET = False"
| "is_in k (MKT n l r) = (k=n ∨ is_in k l ∨ is_in k r)"
```

$\Theta_{3\text{-}3}$

The Nano-Kernel LCF - Architecture:

Putting the Classical Kernel actually into Plugins ...
(used since Isabelle2005)

Classical Kernel: Naming (and therefore referencing to thm's) left to the SML-toplevel, Kernel talks of logic-specific items (terms, hyps,...)

Nano-Kernel: Naming and Referencing is at the heart of the design; keeping $\_\subseteq\_$ acyclic is the key invariant. From the perspective of the Nano-Kernel, thm's and sign's are just “data”.

The Nano-Kernel LCF - Architecture:

Putting the Classical Kernel actually into Plugins ...
(used since Isabelle2005)

context = {id : Id,
ancestors : Id list,
...}

„Θ" thycontext = context + {
sign : Signature,
thm_db : name $\rightharpoonup$ thm,
...}

„$\Gamma \vdash_\Theta \varphi$" thm = {certificate : CertId,
hyps : term,
prop : term}

CertificateTable : CertId $\rightharpoonup$ thycontext

„_$\subseteq$_" subthy: thycontext × thycontext → bool

The Nano-Kernel LCF - Architecture:

Putting the Classical Kernel actually into Plugins ...
(used since Isabelle2005)

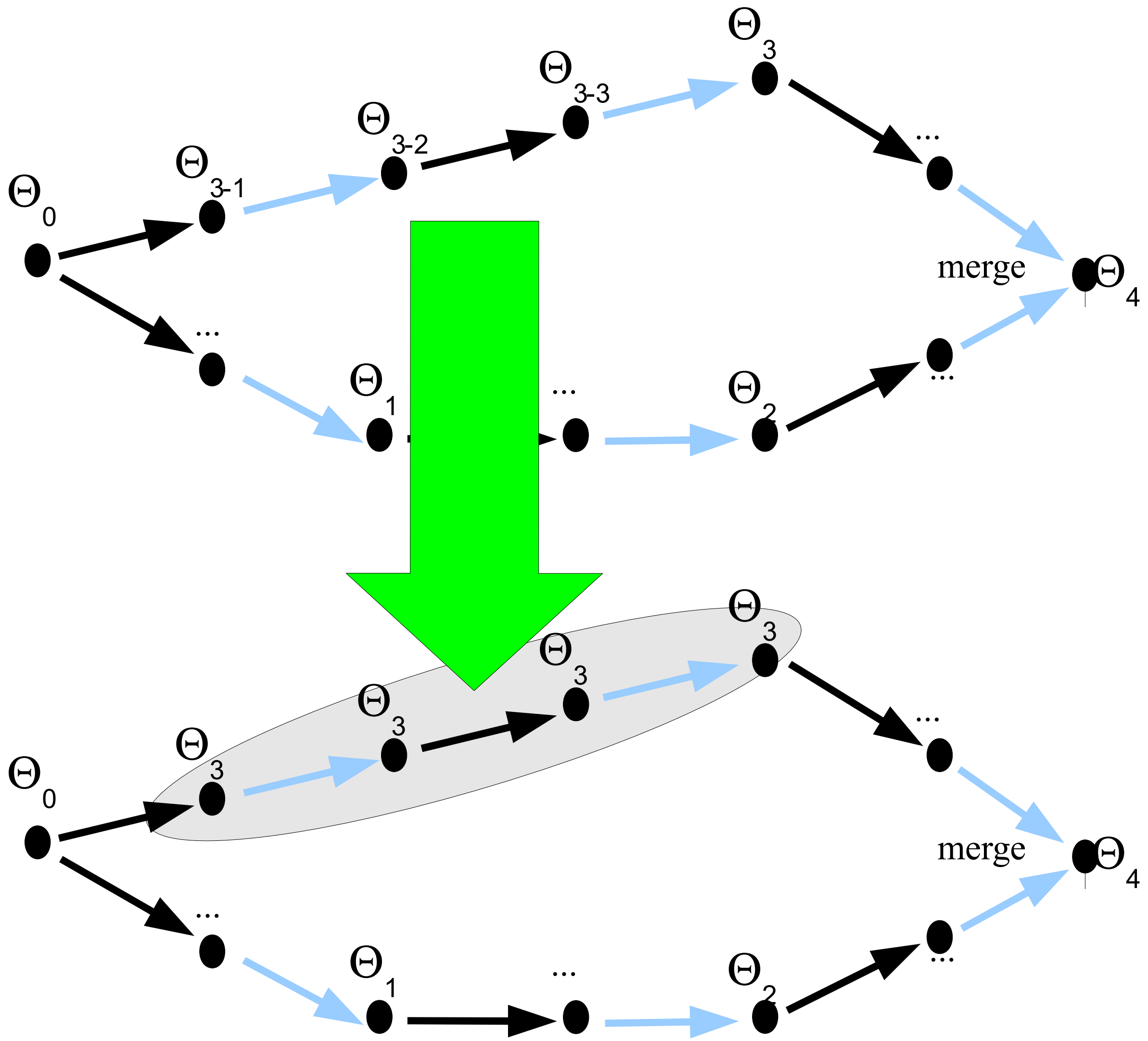
```
proofcontext = context + {
                   theory_of_proof : CertId,
                   fixes : string list,
                   assumes : term list,
                   ...}
```

Proof-Contexts are data-structures to capture local information like fixes, assumptions, abbreviations etc., their names and their prover-configuration ...

In particular all local data relevant for the interfacing between sub-proofcontexts to their supercontexts...

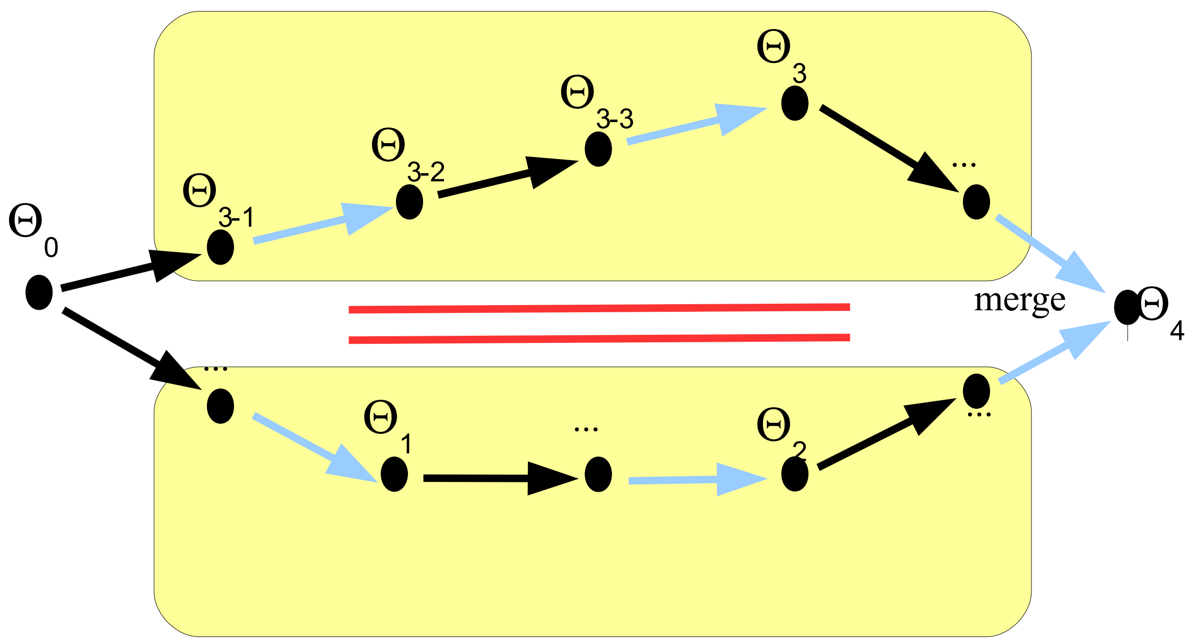
Nano-Kernel LCF-Architecture:

fine-grained global context transition with branch and merge
proofs are global transitions, mixed with other extensions
grouping of context transitions via Kernel re-certication
(but also Nano-Kernels Isabelle2005)



Parallel Nano-Kernel LCF-Architecture:

coarse-grained parallelism
(Isabelle2008 in batch-mode, Isabelle2010 also in interactive mode)



Parallel Nano-Kernel LCF - Architecture:

Putting the Classical Kernel actually into Plugins ...

Isabelle2009 - 10 (!)

...

„Θ“ thycontexts = contexts + {
sign : Signature,
thm_db : name ⇸ thm,
...}

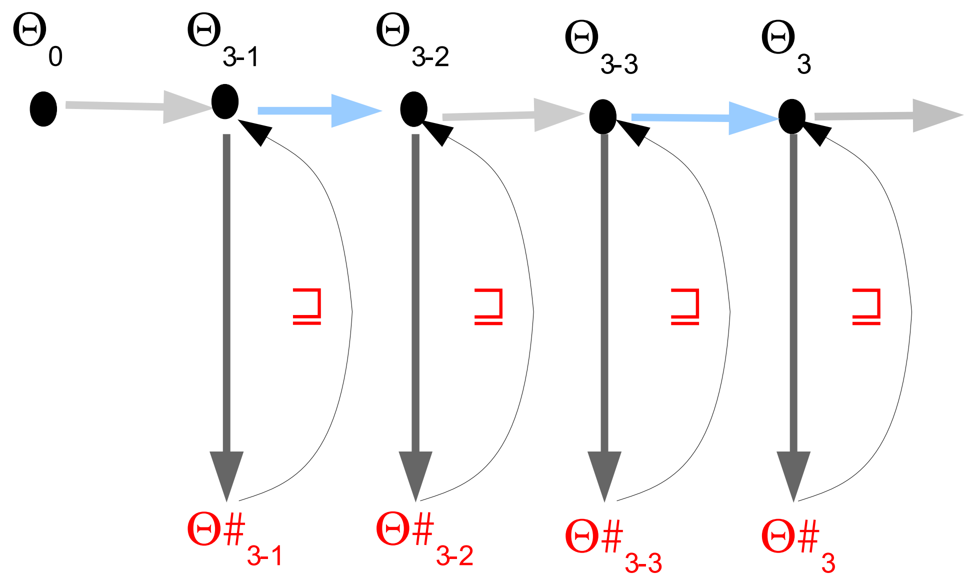
„$\Gamma \vdash_{\Theta} \varphi$“ thm = {context : CertId,
promises: name ⇸ thm future,
hyps : term,
prop : term}

status :: thm => { failed : bool,
oracle: bool,
unfinished: bool}

...

Parallel Nano-Kernel LCF-Architecture:

fine-grained, asynchronous parallelism
(Isabelle2009)



All thm's may contain sub-thm's (promises) used in their proof whose validation is actually left to an asynchronous thread managed in a data-stucture future. Successful validation leads to a fulfil-ment of a promise. Merges were postponed till fulfillment of all promises in a thm_db of a global context.

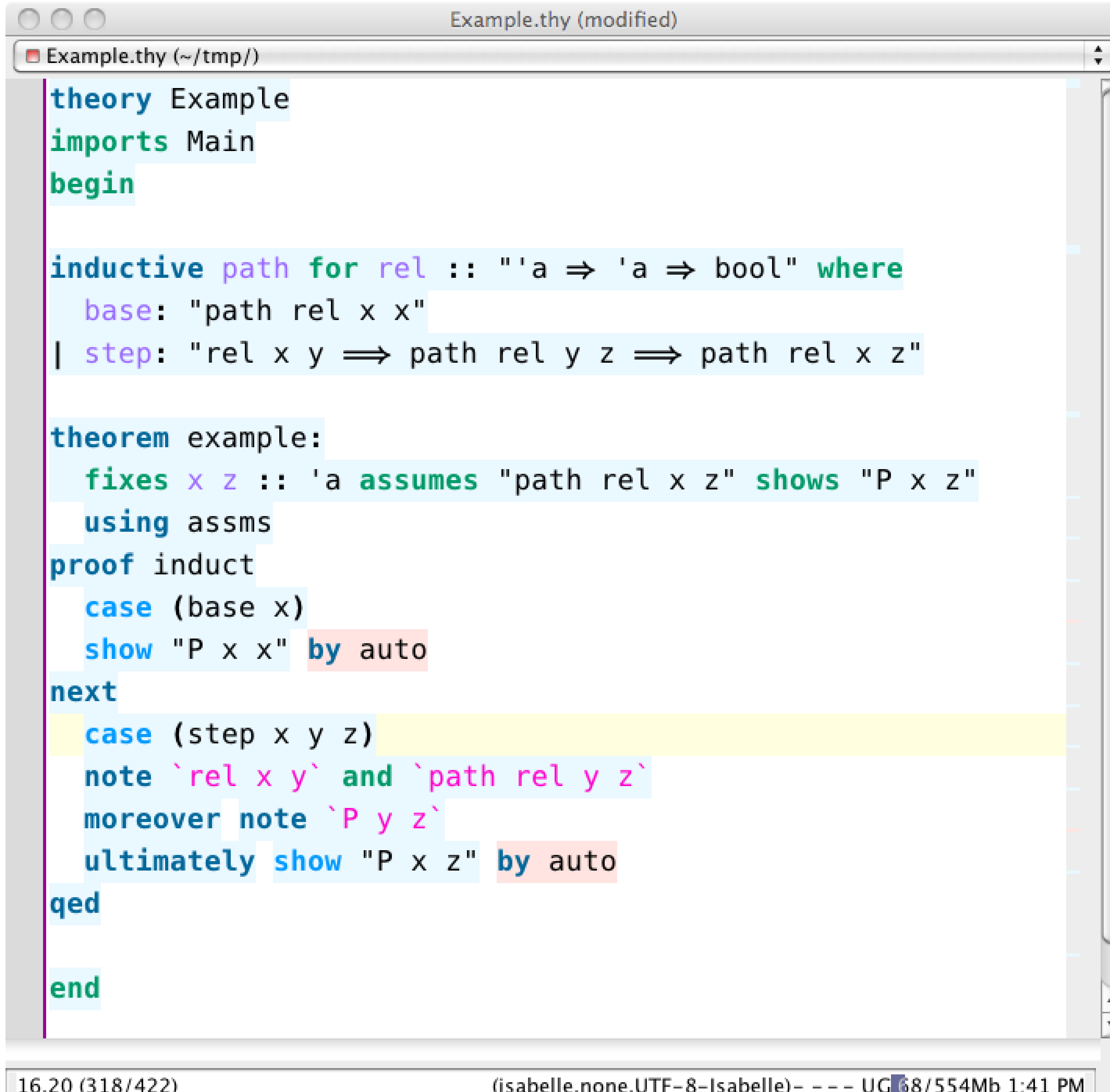
(Futures are actually grouped, can emit/receive events and can be killed).

Parallel Nano-Kernel LCF-Archi-tecture

in the

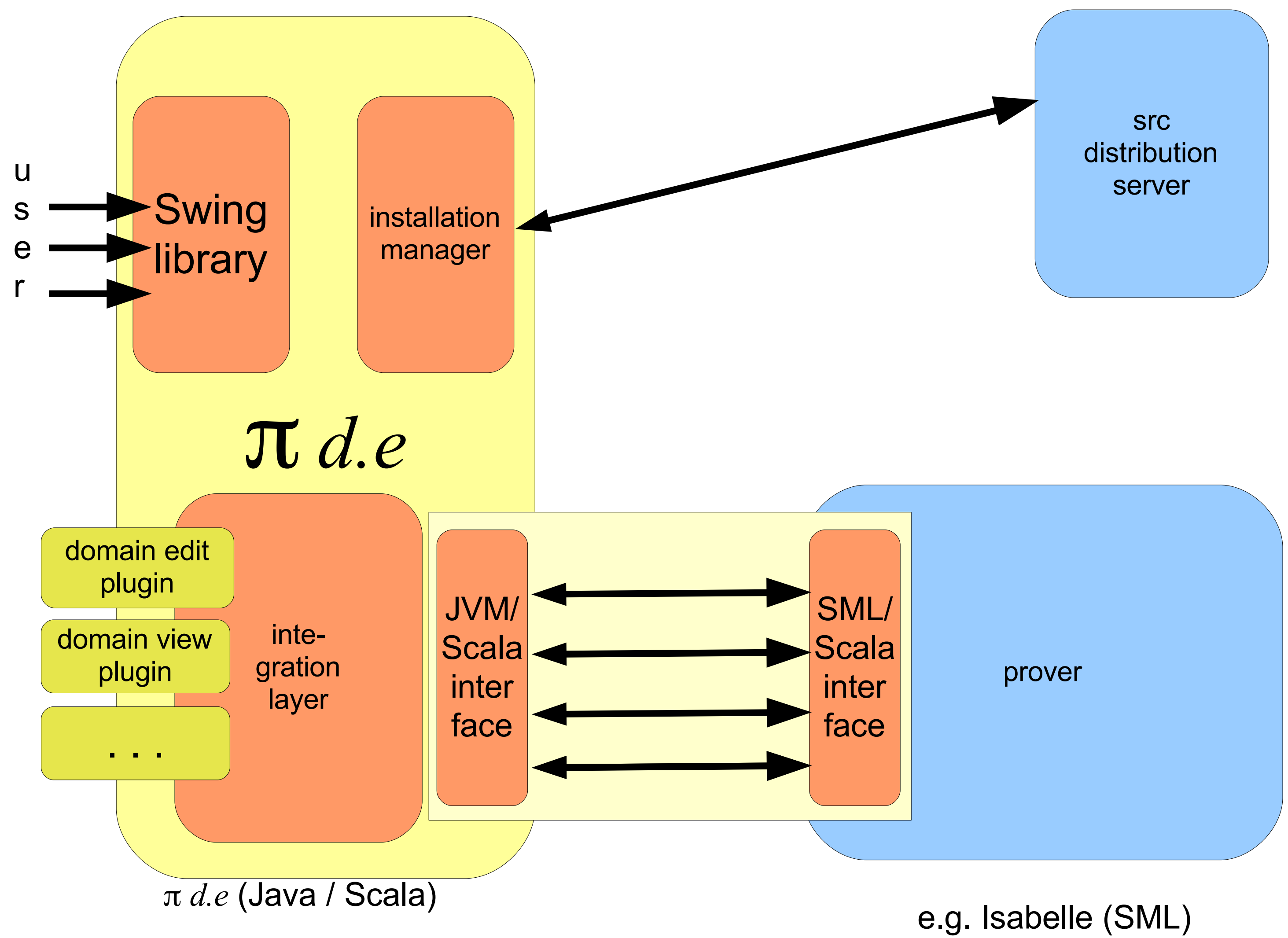
jEdit - GUI

fine-grained, asynchronous parallelism (Isabelle2009-2)



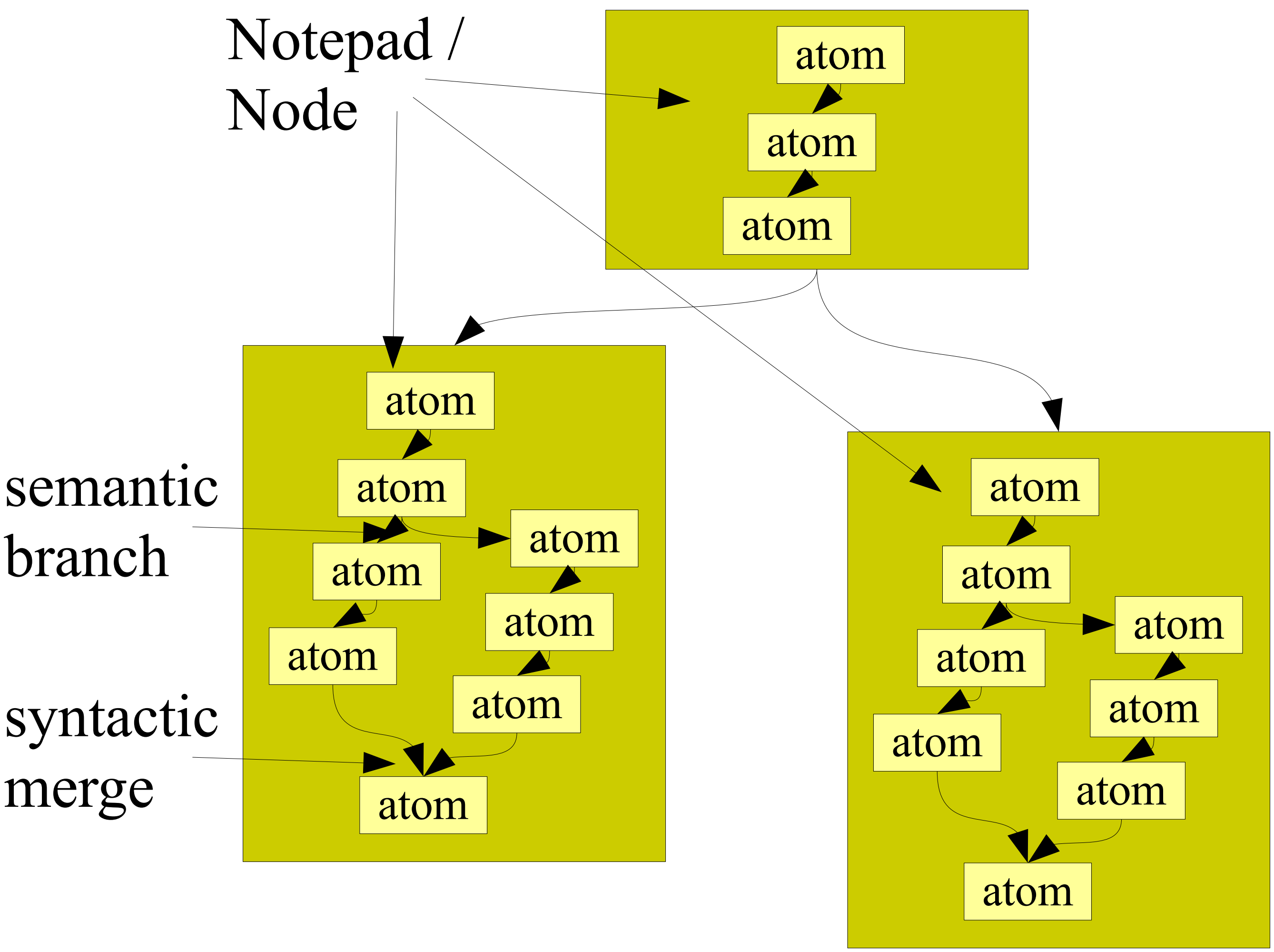
PIDE - GUI - Architecture

(see PIDE - Project: http://bitbucket.org/pide/pide/wiki/Manifesto)



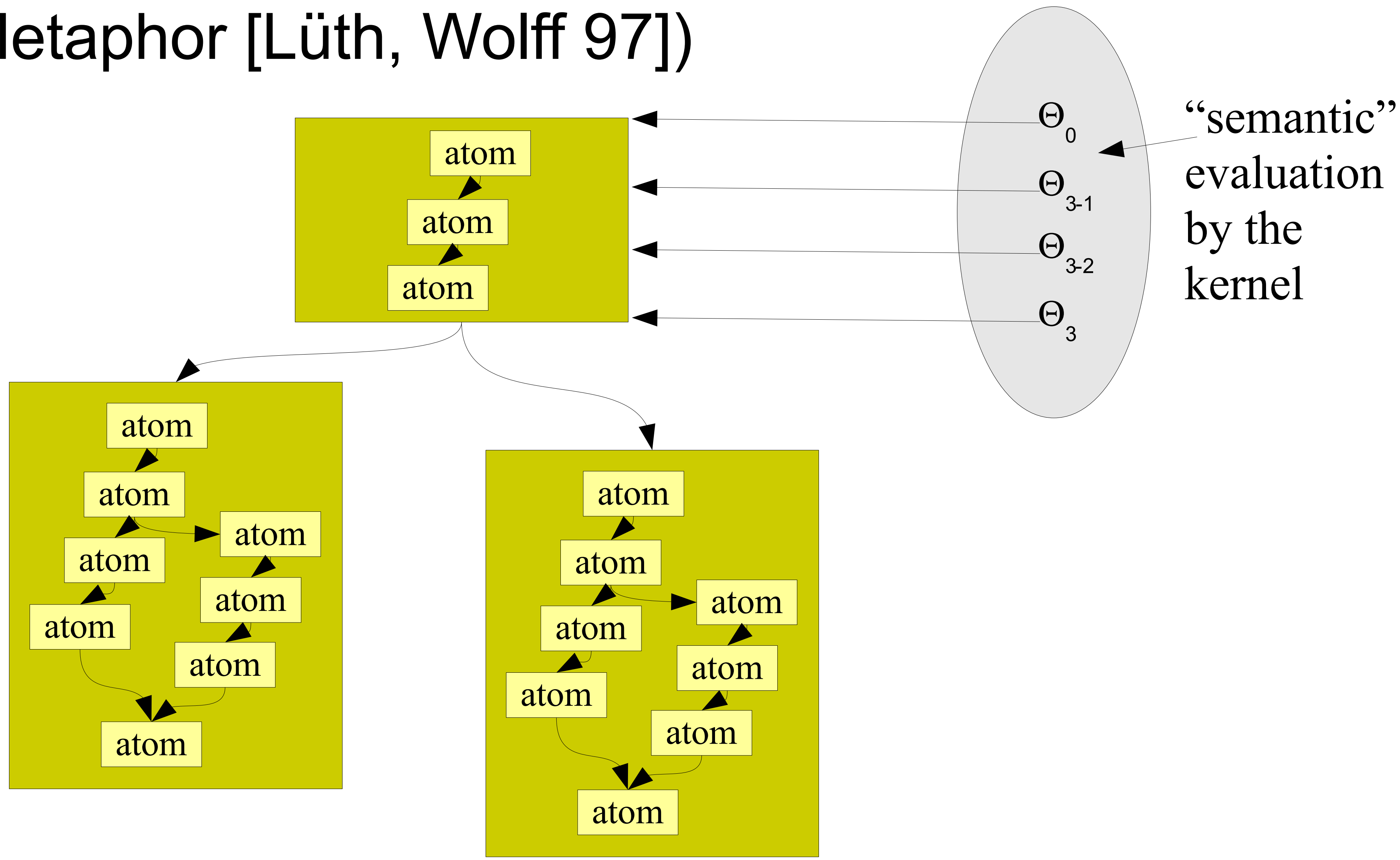
Context-Management and Document Model

- Document Model (following the Notepad-Metaphor [Lüth, Wolff 97])

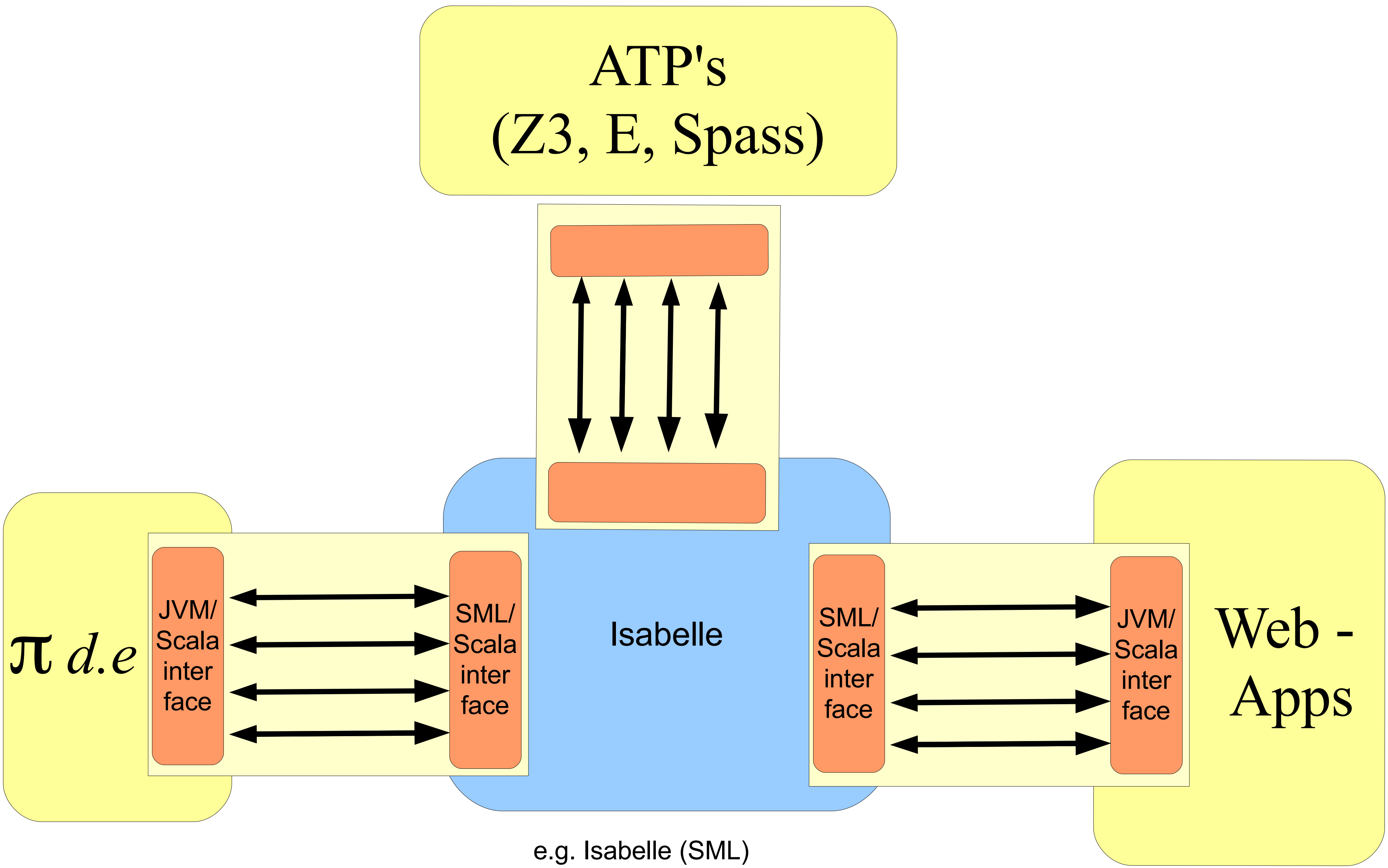


Context-Management and Document Model

- Document Model (following the Notepad-Metaphor [Lüth, Wolff 97])



Architecture in the Future



FM Tool-Development built upon the Isabelle Framework

Tools as Plug-Ins (I)

- Simpl [Schirmer]
  - conservatively derived PO-generator for an imperative core-language
  - front-ends: C0 (Leinenbach), C0-VAMOS (Daum) C?? (Norrish, NICTA)
  - classical library development
- Security Toolbox [Sprenger]
  - conservatively derived PO-generator for an interleaved transition systems
  - classical library development for Crypt-Engines

Tools as Plug-Ins (II)

- HOL-Z [Brucker, Rittinger, Wenzel, Wolff]
  - conservative, shallow Embedding for Z and Schema-Calculus,
  - integrated in a TOOL-chain
    (loader for external TC ZETA and format .holz)
    - Plug-In with
    - own state (ZEnv capturing “schema signatures”
      and proof-obligations)
    - own Isar commands
      - for loading “load_holz”,
      - for support of refinement methodology “refine A B [functional]”
      - for proving “zallintro, zexelim” …
    - reuse of: GUI, Prover, Libraries, ...

Tools as Plug-Ins (II')

- HOL-Z (cont)

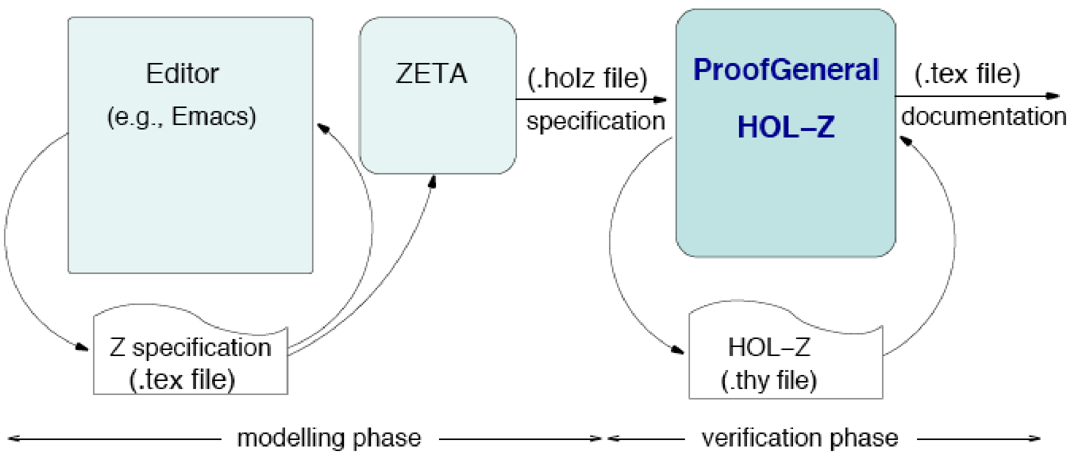
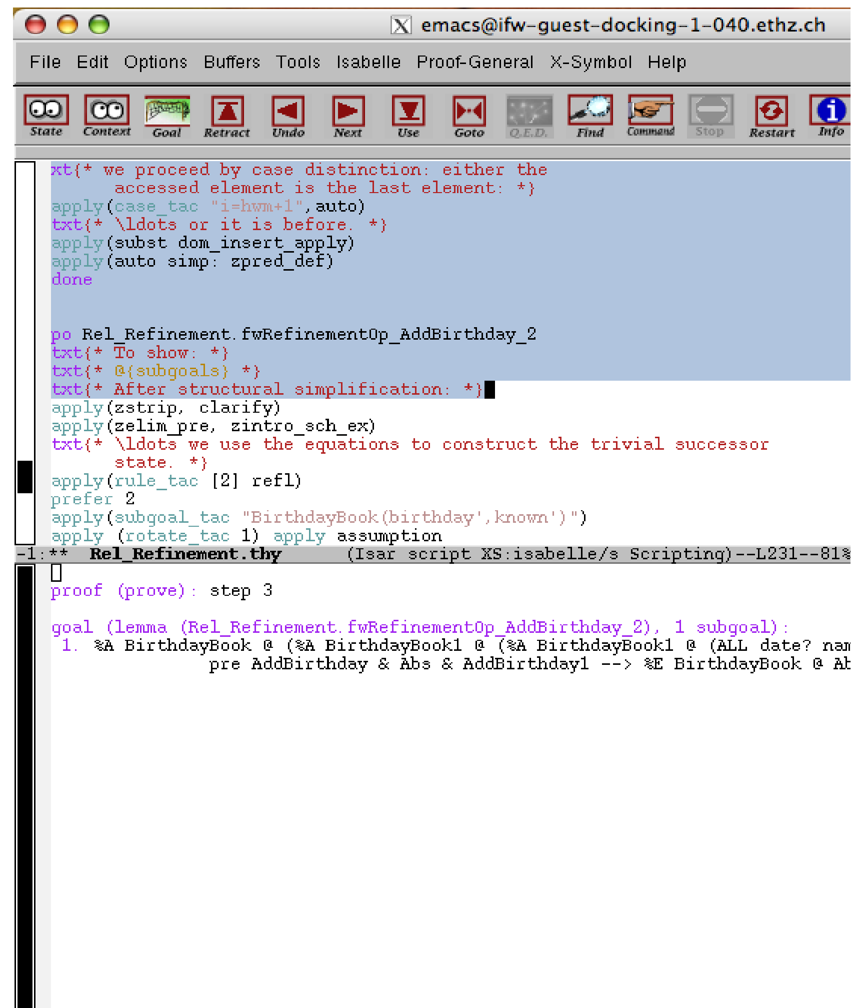


Figure 1. The HOL-Z system architecture perspective

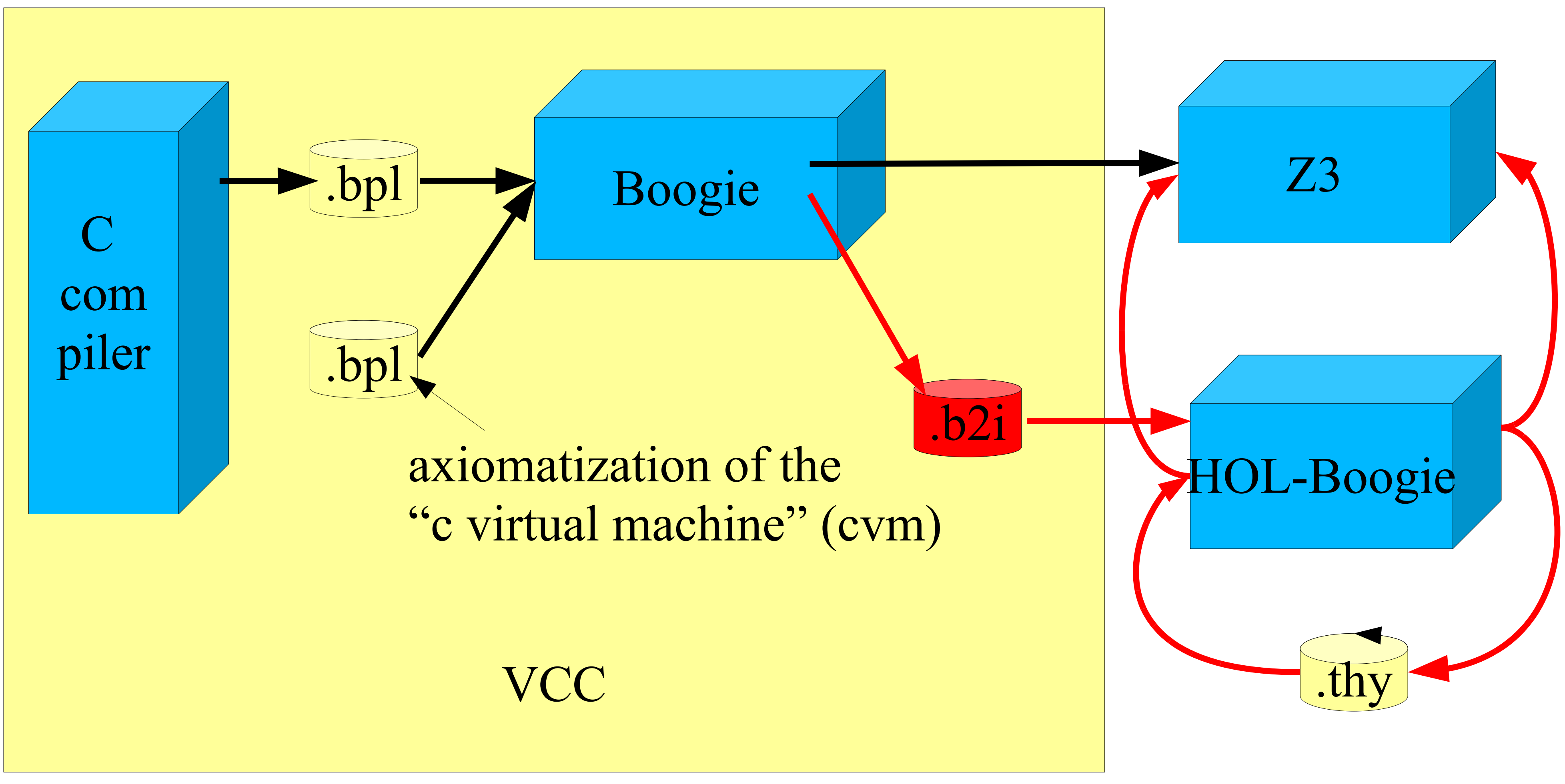


Tools as Plug-Ins (III)

- HOL-Boogie [Böhme, Wolff]
  - Proof-Environment for non-conservative PO-generator Boogie and the VCC - FrontEnd (Concurrent, X86 C)
  - Intended to Debug Z3 - Proofs (Z3 integrated)
  - Plug-In Managed State: PO-Management
  - Integration of Z3 + Proof-Reconstruction [Böhme]
  - own integrative (SMT) Proof-Methods
  - own (native) Proof-tactics for Decomposition and Memory-Model-Handling for VCC1 and VCC2
  - Tracking of Assertions

Tools as Plug-Ins (III')

- HOL-Boogie [Böhme, Wolff]



Tools as Plug-Ins (IV)

- HOL-OCL [Brucker, Wolff]
  - conservative, shallow Embedding for UML/OCL class diagrams and object-oriented specifications
  - Support for Refinement-Methodology
  - Plug-In in Tool-Chain (Loader for Argo/UML …)
  - Plug-in State: PO-Management, OO-DM Management
  - Own Proof-Commands
  - Own Proof Methods

Tools as Plug-Ins (IV')

- HOL-OCL [Brucker, Wolff]

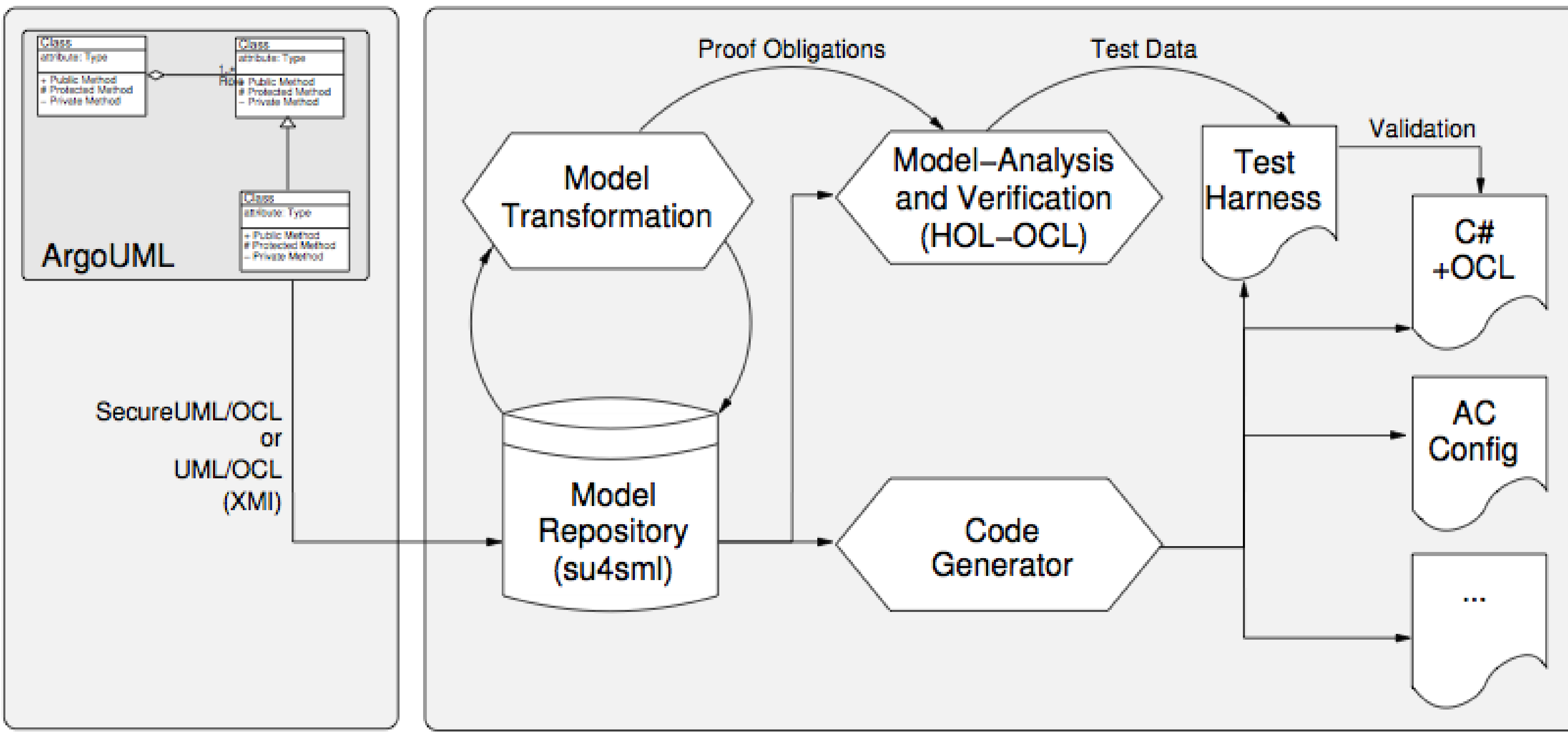
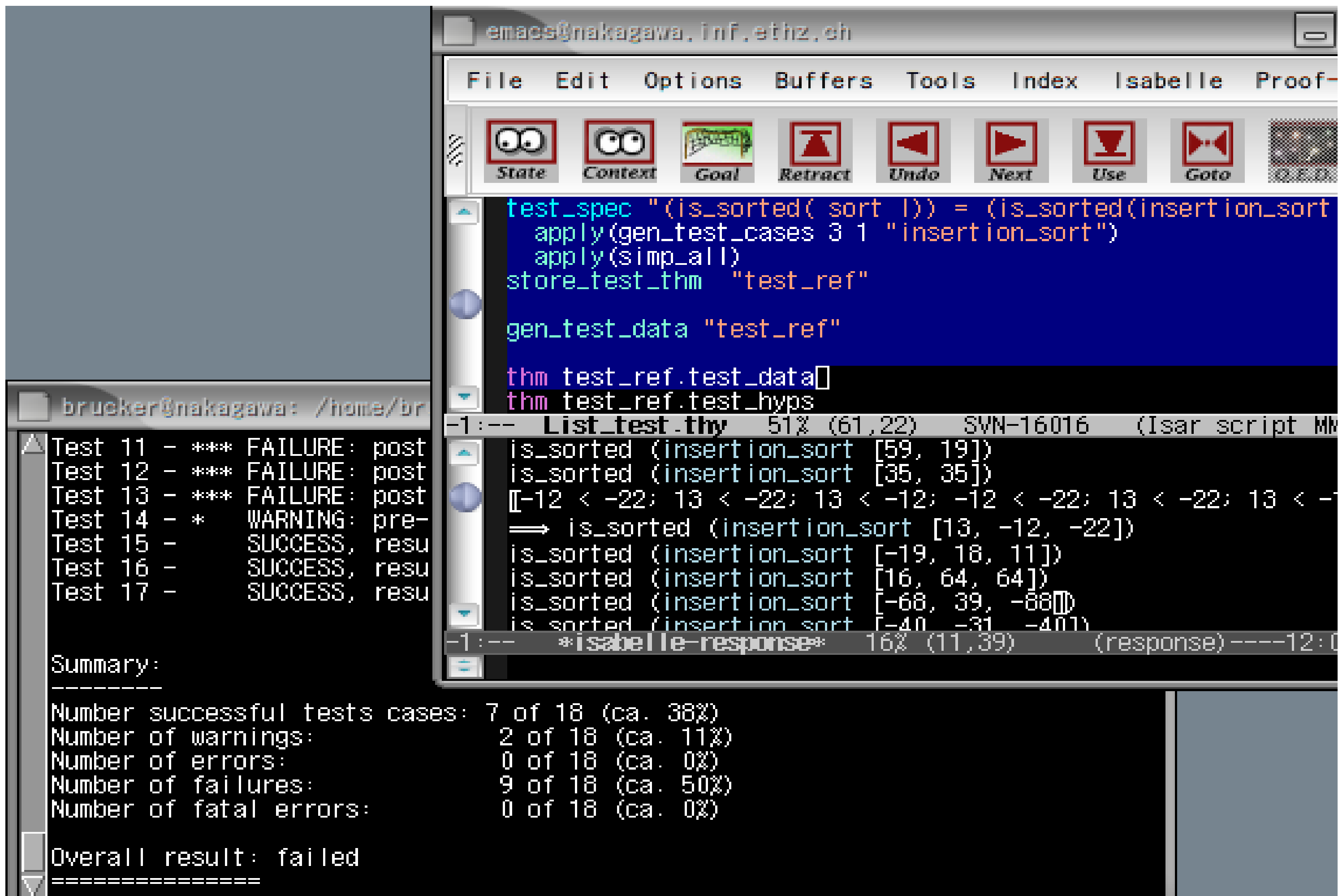


Figure 1: MDA Framework and Toolchain Overview

Tools as Plug-Ins (III)

- HOL-TestGen [Brucker, Brügger, Krieger, Wolff]
  - Proof-Environment for Conservative Test-Data-Generation and Test-Driver Generation
  - Used for Security Test Scenarios ...



Conclusion

Conclusion

- The ITP Programme (and Isabelle in particular) allowed:
  - reconciliation of foundational with pragmatic technology issues
  - reconciliation specification & programming
  - reconciliation with ATP (via Oracles, Proof-Object certification, Tactic Proof Reconstruction)
    - parallel evaluation of proofs &
    - parallel (distributed) documents

Conclusion

- Reusing Isabelle as FM tool foundation offers:
  - substantial conservative libraries
  - standardized interfaces to tactic and automatic proof
  - proof documentation
  - code generation
  - a programming interface and genericity in design

... a lot of machinery not worth to reinvent.

Conclusion

- Larry Paulson,
  “How to write a theorem prover”:
  - One final advice:
    Don't write a theorem prover,
    try to reuse someone else's.
- Harald Ganzinger, confronted with a Java-From-Scratch Tableaux Prover:
  - “Das ist doch wieder der naive Ansatz.”