The ANR Project
Paral-ITP

Background,
Goals & Scientific Challenges,
First Results

Burkhart Wolff
Project Coordinator
Université Paris-Sud, LRI, Nov 2011
ANR-11-INSE-001

Overview

- Motivation
- Background
- The Research Challenges
- First Results

The Consortium:

- U-PSud/ ForTesSE (M. Wenzel, B. Wolff / Isabelle)
- INRIA Roquencourt (Hugo Herbelin, Damien Dogliez)
- INRIA Saclay (Bruno Barras, Enrico Tassi)

Motivation

- Boosting ITP Technology
  - Profiting from more Computer-Power
  - New IDE's for Theory Development

- Transforming ITP's into Frameworks
  for Domain-Specific Formal Languages

  ("The Eclipse of Formal Methods Tools")

Background

- ITP vs. ATP Design
- The LCF Paradigm and its Development
  - prover architecture (example: Isabelle)
  - kernel architecture (example: Isabelle)
- The Document Model Challenge
- The Parallelization Challenge
  - logically safe, programmable kernel
  - asynchronous computation at kernel level

The "Automated Theorem Prover" Research Programme

- 1960 : Davis / Putnam Procedure (Resolution-based)
- 1962 : **Davis-Putnam-Logemann-Loveland (DPLL) algorithm** i.e. for solving the CNF-SAT problem.
- 1965 : Robinson: Unification & Resolution
- 1980 : McCune: Otter
- 2004 : Ganzinger, Hagen, Nieuwenhuis: DPLL(X) Concept
- 2006-7: Z3 (Microsoft Research Development of a DPLL(X) prover for static analysis, test and program verification

The "Interactive Theorem Prover" Research Programme

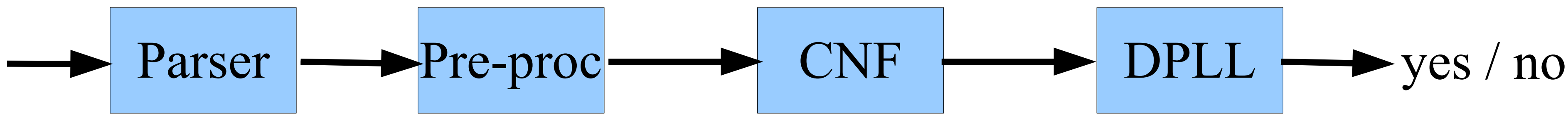
- 1968 : Automath
- 1975 : Stanford LCF
  LISP based Goal-Stack, orientation vs. functional Programming, Invention: Parametric Polymorphism
- 1984/5 : Cambridge LCF
- 1986 : Isabelle
- 1986-90 : HOL-88, Coq

Historic Overviews:

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

ITP vs. ATP's

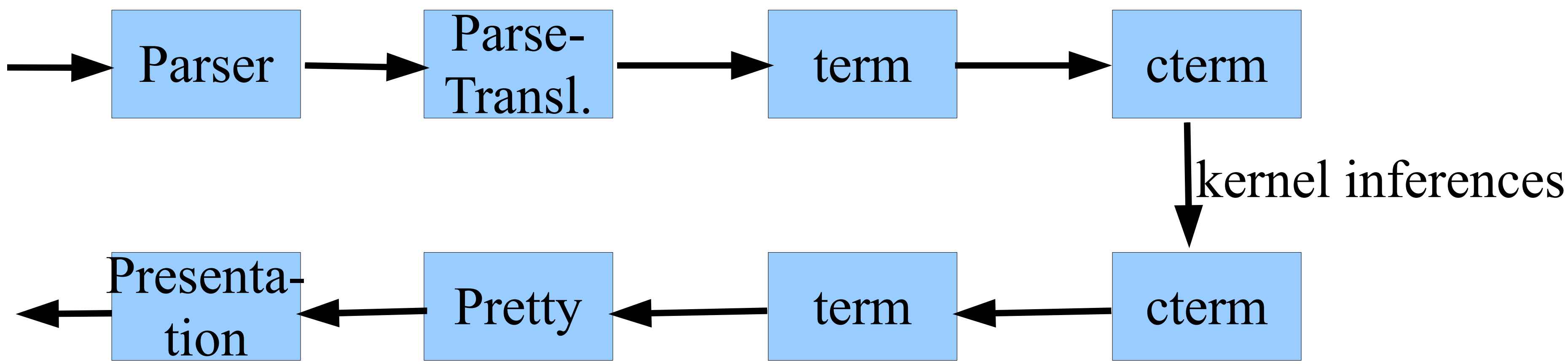
- ATP Design one-way-compilation of “input”:



- Core implemented in complex, highly efficient data-structures (usually C)
- several million inferences per second
- untyped formula representation, originally without binding, presentation unimportant
- logical theories (“background”) unique and small
- after source modification: simply reprove from scratch

ITP vs. ATP's

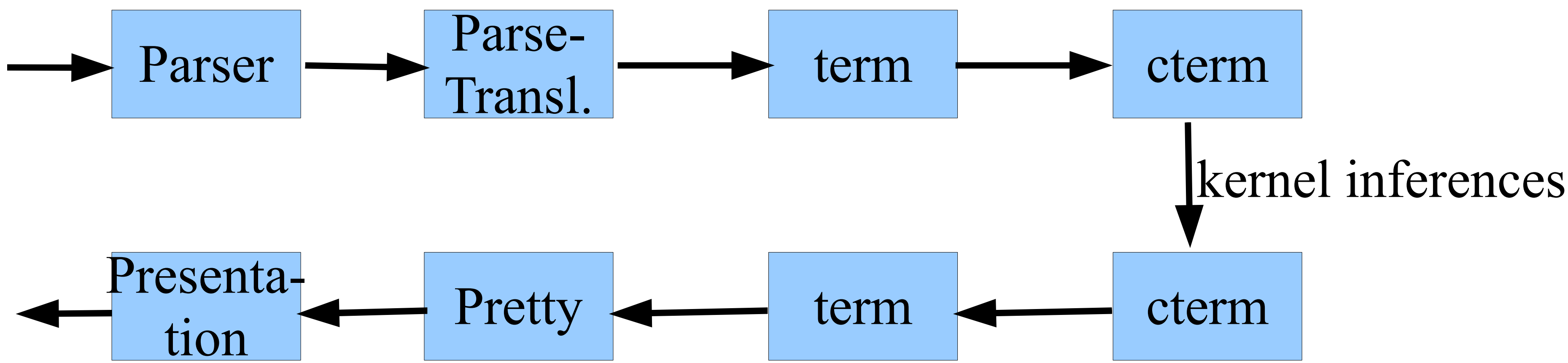
- ITP Design two-ways: INTERACTION



- Core implemented in simple, universal typed data-structures (usually ML)
- several thousand* inferences per second

ITP vs. ATP's

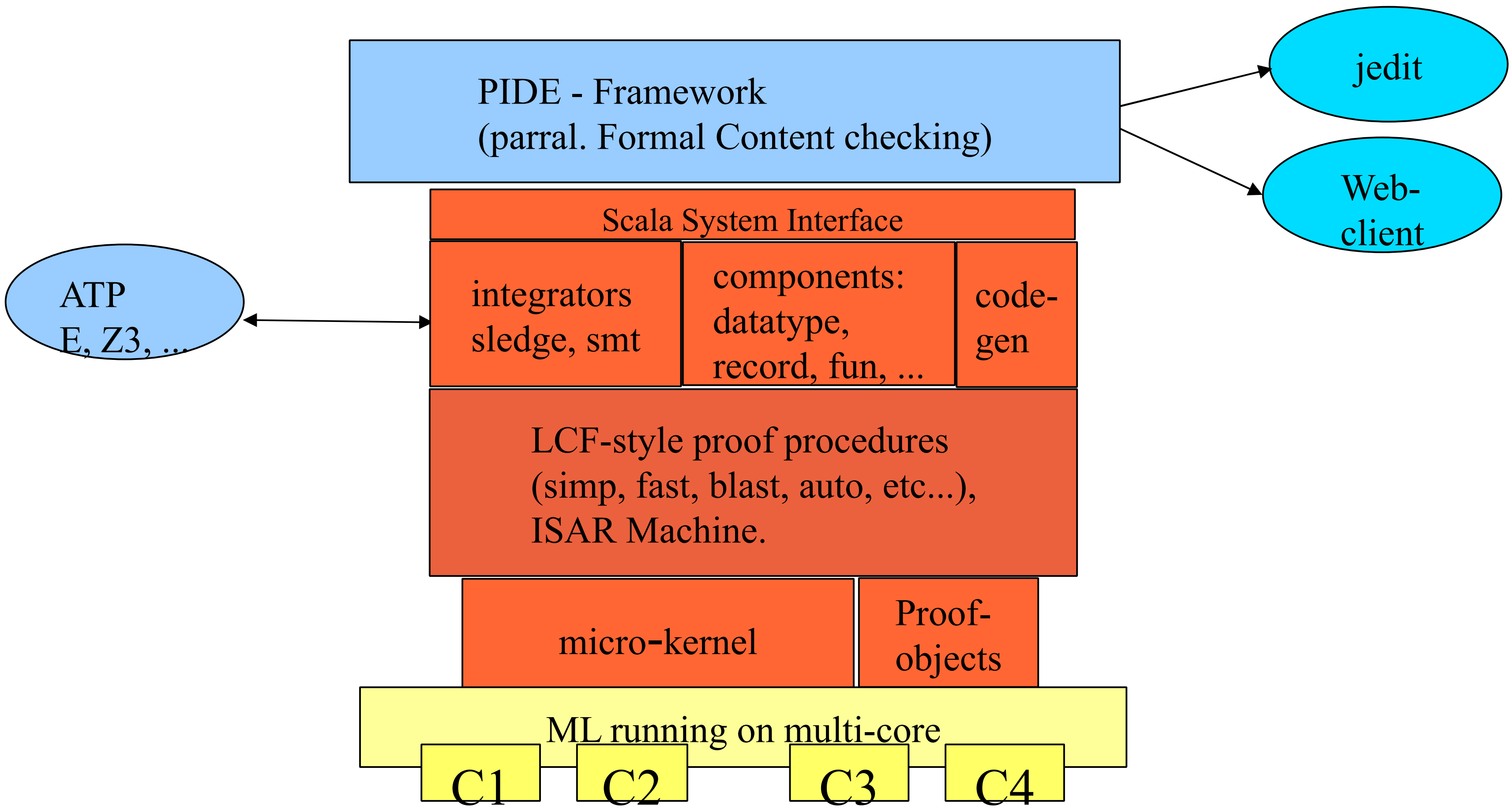
- ITP Design two-ways: INTERACTION



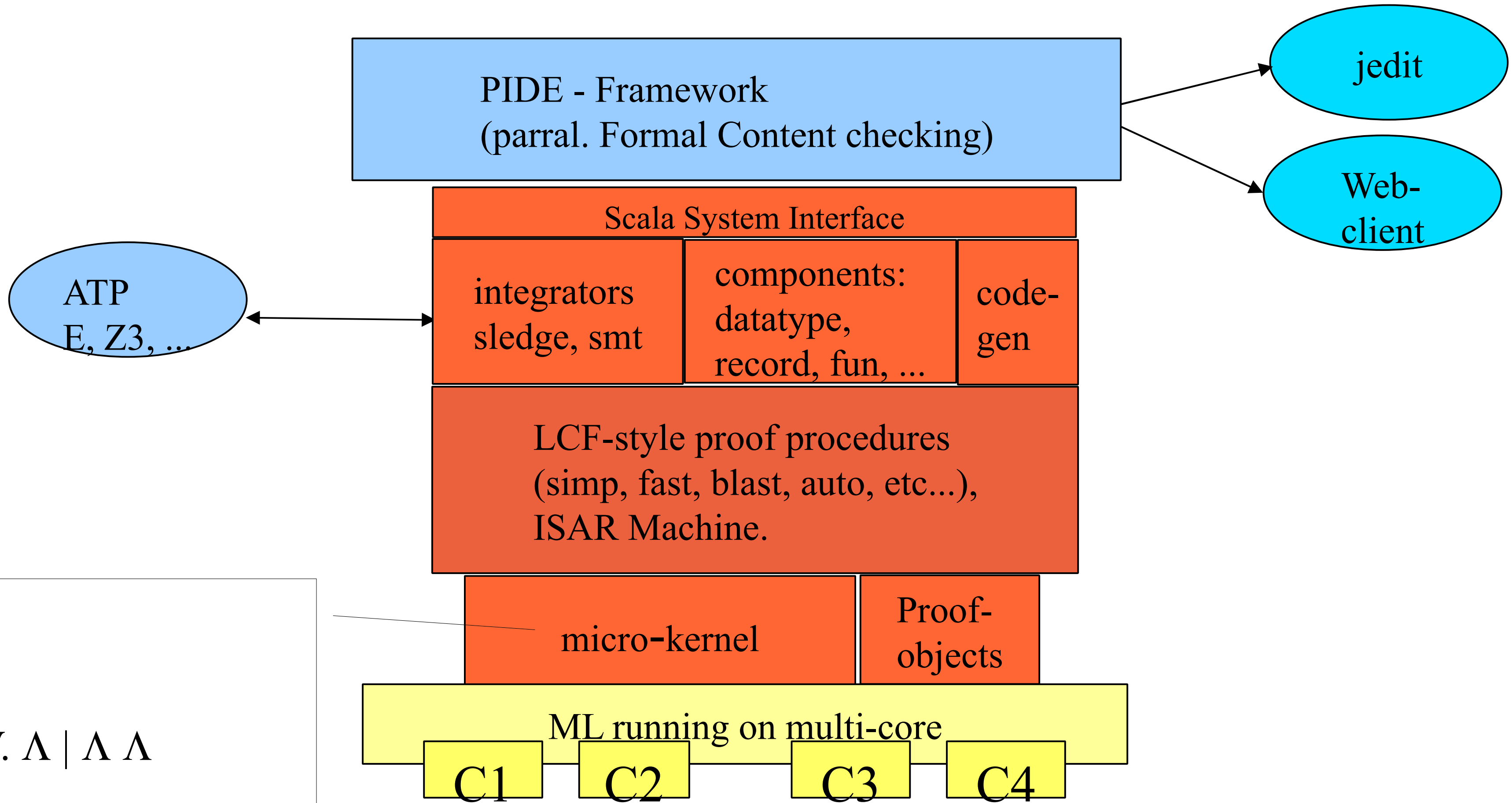
- logical theories very large
- source modification: UNDO, incremental algorithms, functional design
- the document and proofs become important.

The ITP Research Programme
and
The Evolution of the
Isabelle Architecture

Isabelle Architecture (2012)

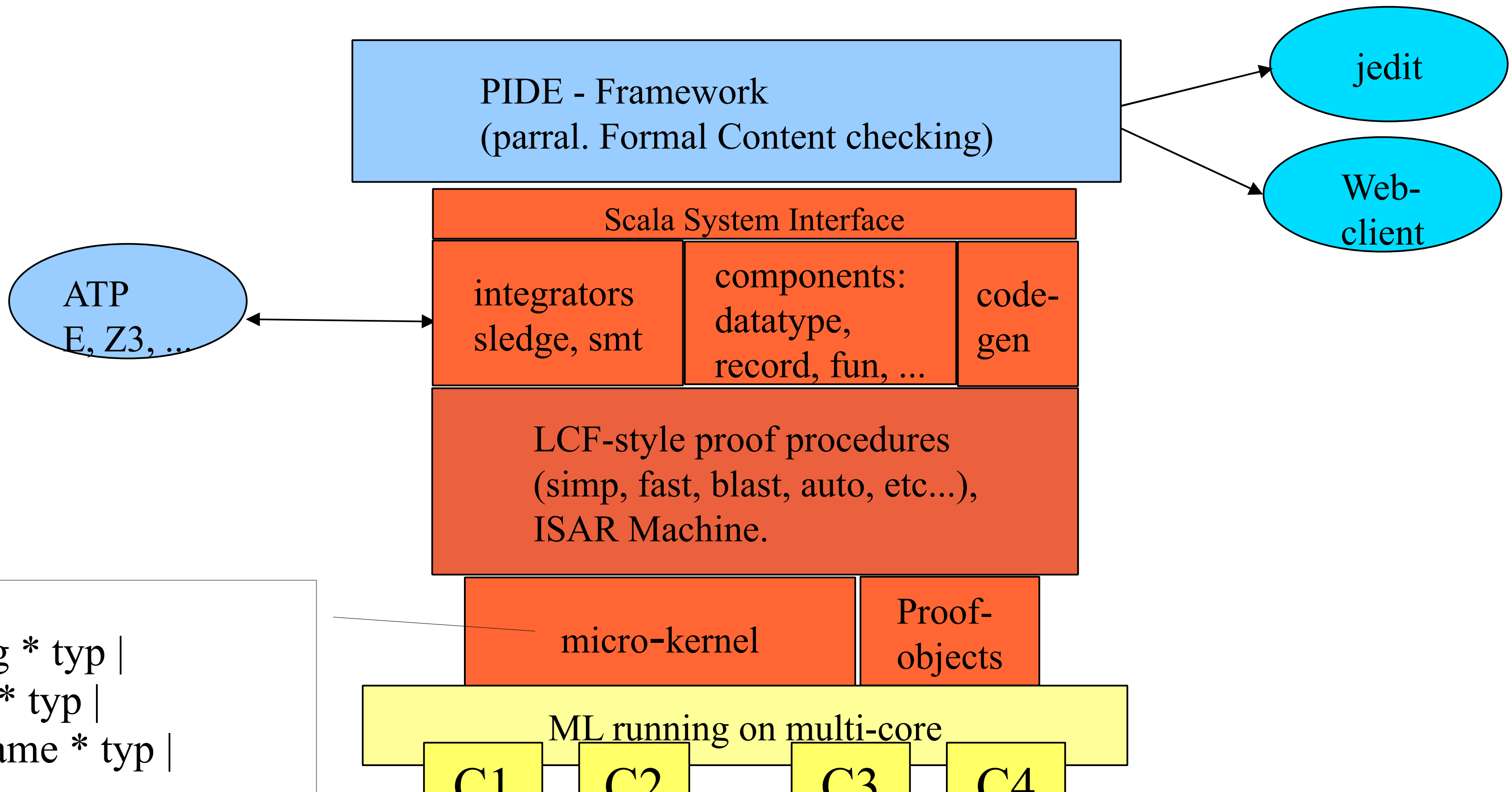


Isabelle Architecture (2012)



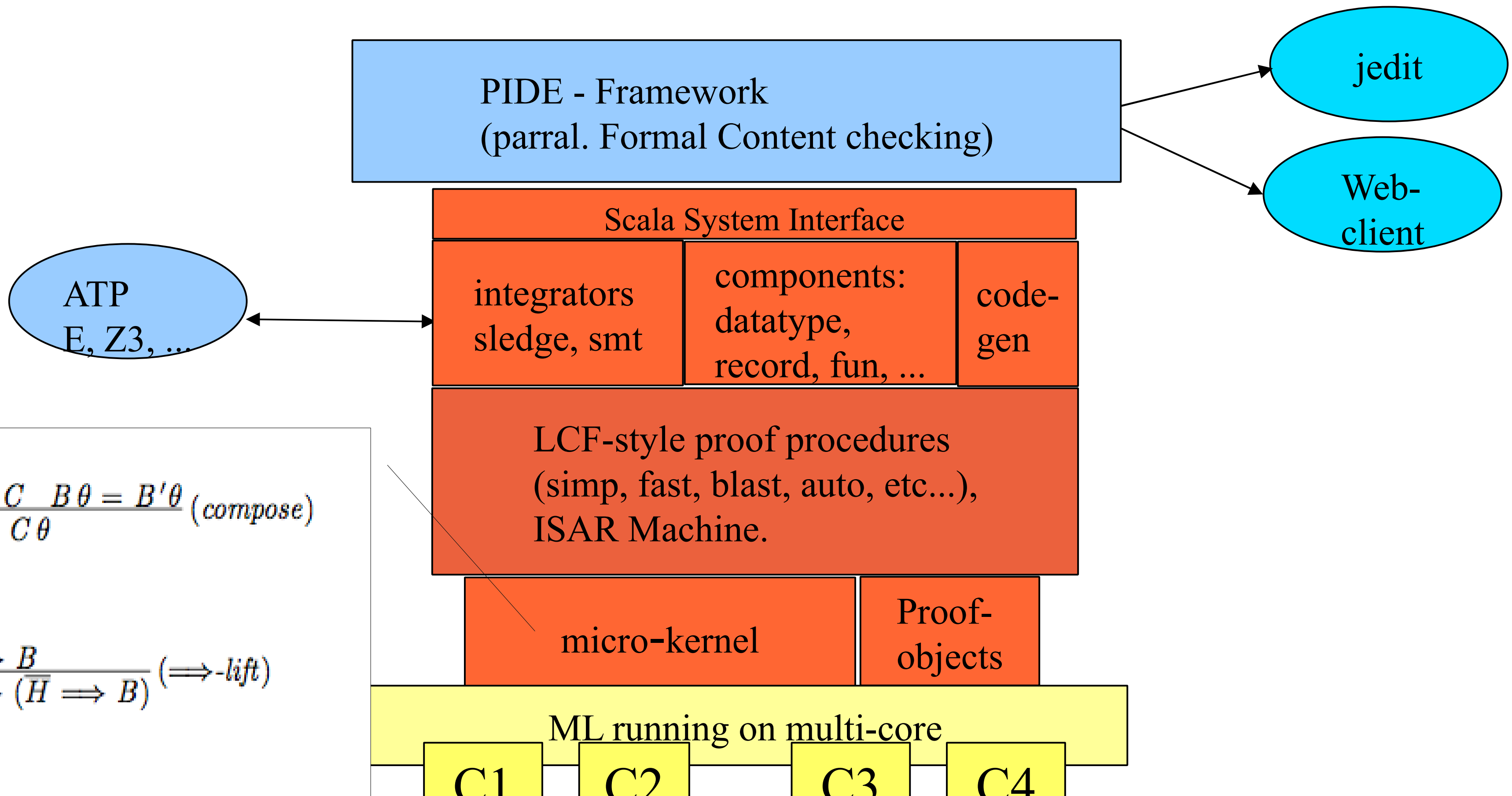
$\Lambda = C \mid V \mid \lambda V.\ \Lambda \mid \Lambda\ \Lambda$

Isabelle Architecture (2012)




```
datatype term =
   Const of string * typ |
   Free of string * typ |
   Var of indexname * typ |
   Bound of int |
   Abs of string * typ * term |
   $ of term * term
```

Isabelle Architecture (2012)

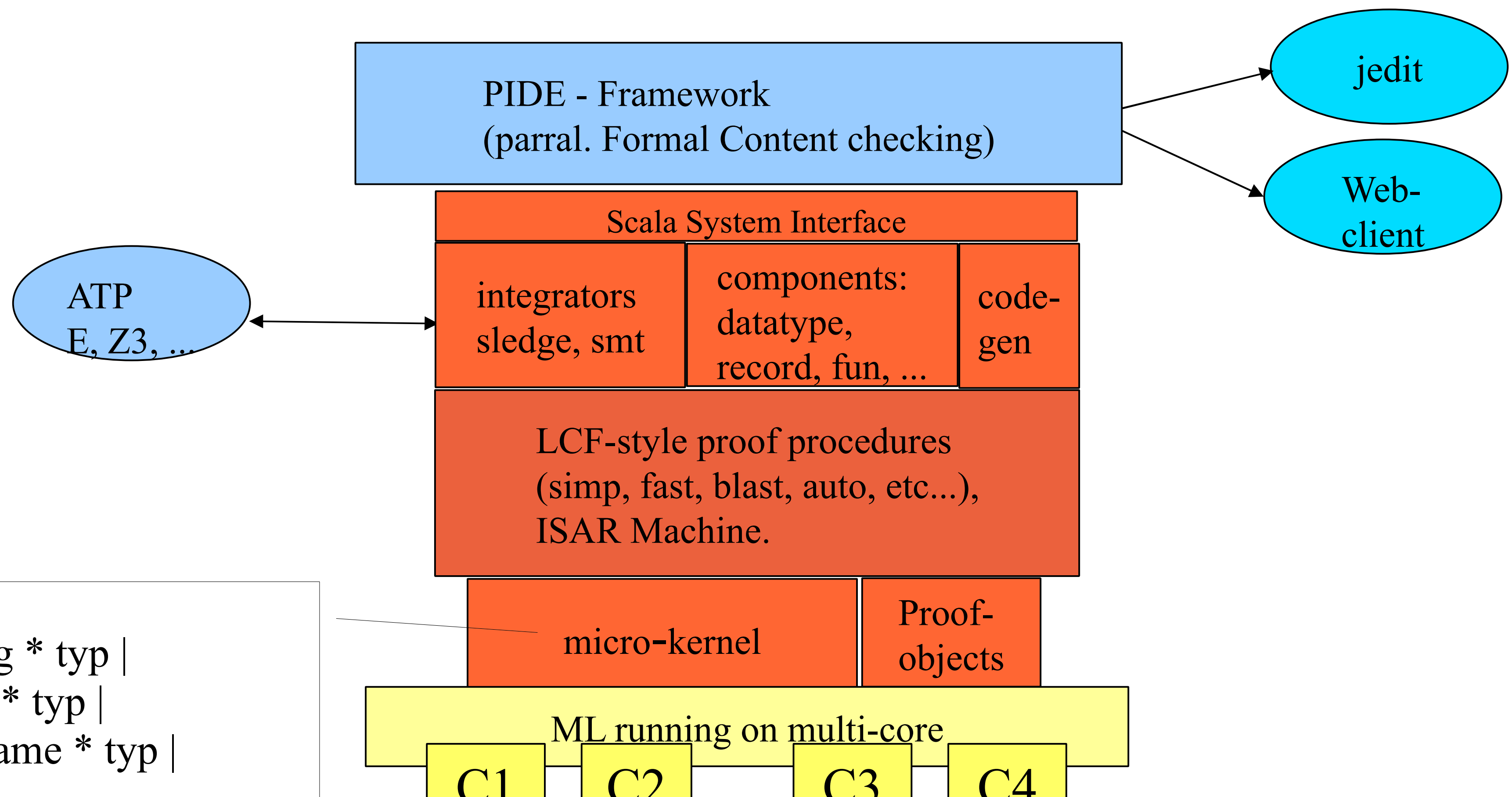


$$\frac{\overline{A} \Longrightarrow B \quad B' \Longrightarrow C \quad B\,\theta = B'\theta}{\overline{A}\,\theta \Longrightarrow C\,\theta}\,(compose)$$

$$\frac{\overline{A} \Longrightarrow B}{(\overline{H} \Longrightarrow \overline{A}) \Longrightarrow (\overline{H} \Longrightarrow B)}\,(\Longrightarrow\text{-}lift)$$

$$\frac{\overline{A}\;\overline{a} \Longrightarrow B\;\overline{a}}{(\bigwedge\overline{x}.\;\overline{A}\;(\overline{a}\;\overline{x})) \Longrightarrow (\bigwedge\overline{x}.\;B\;(\overline{a}\;\overline{x}))}\,(\bigwedge\text{-}lift)$$

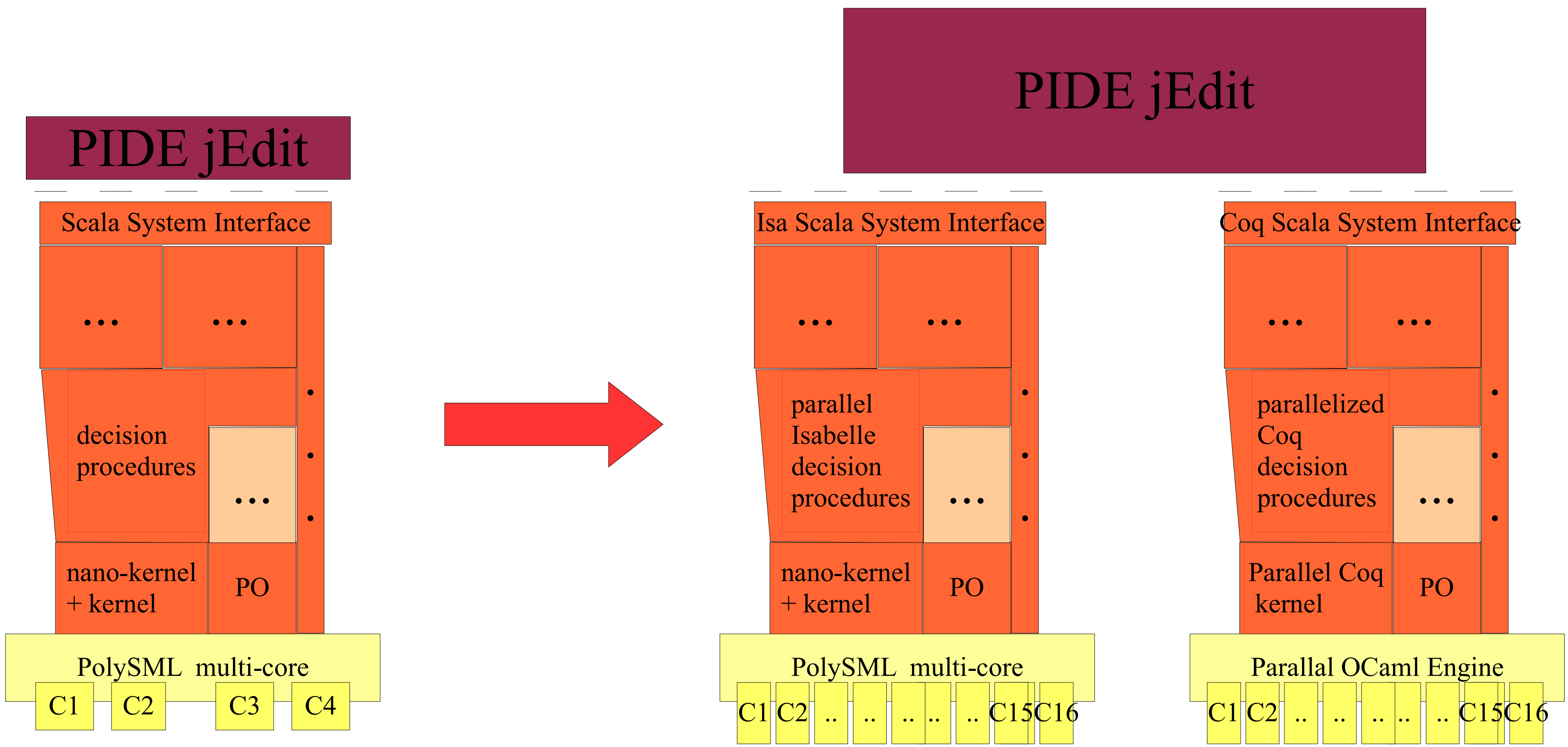
Isabelle Architecture (2012)




```
datatype term =
   Const of string * typ |
   Free of string * typ |
   Var of indexname * typ |
   Bound of int |
   Abs of string * typ * term |
   $ of term * term
```

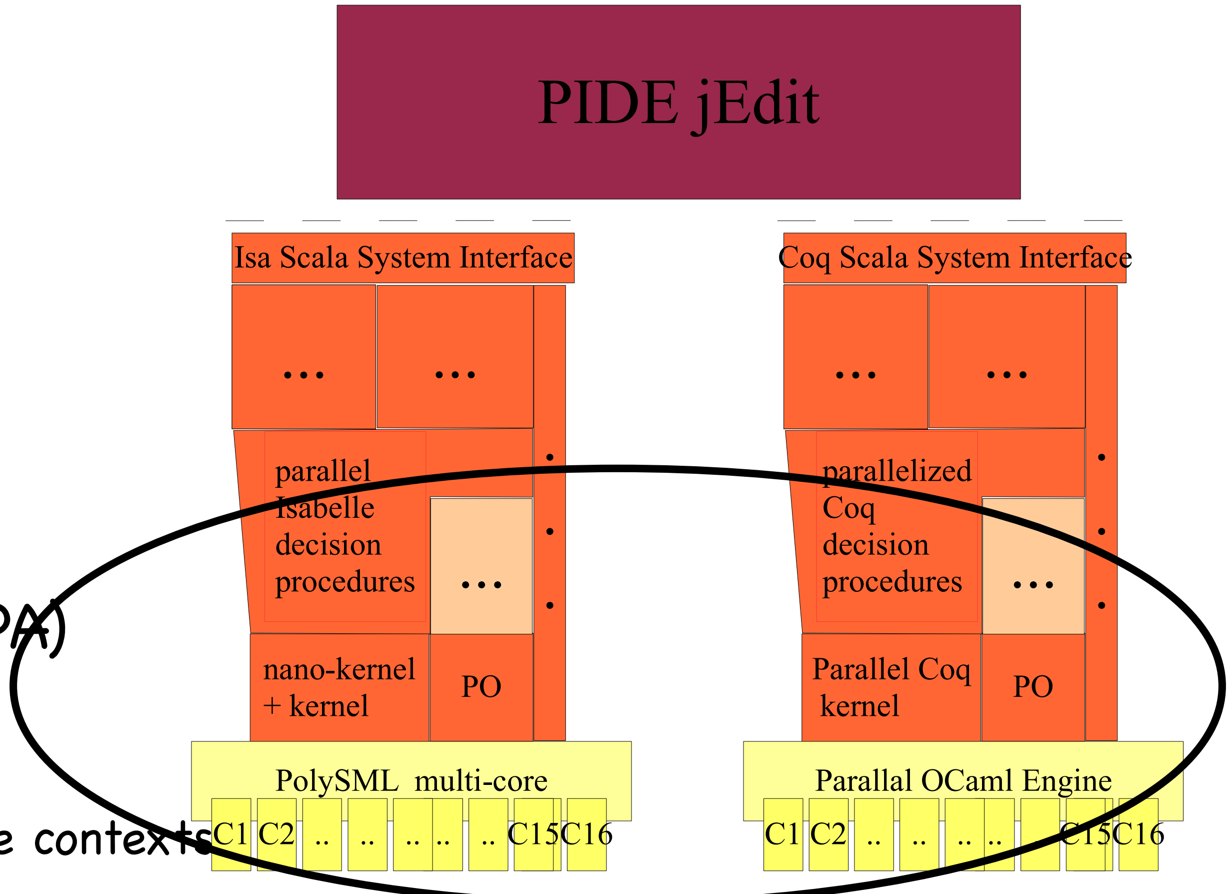
The Project Goals at a Glance

- Paral ITP:



The Project Goals at a Glance

- Paral ITP:



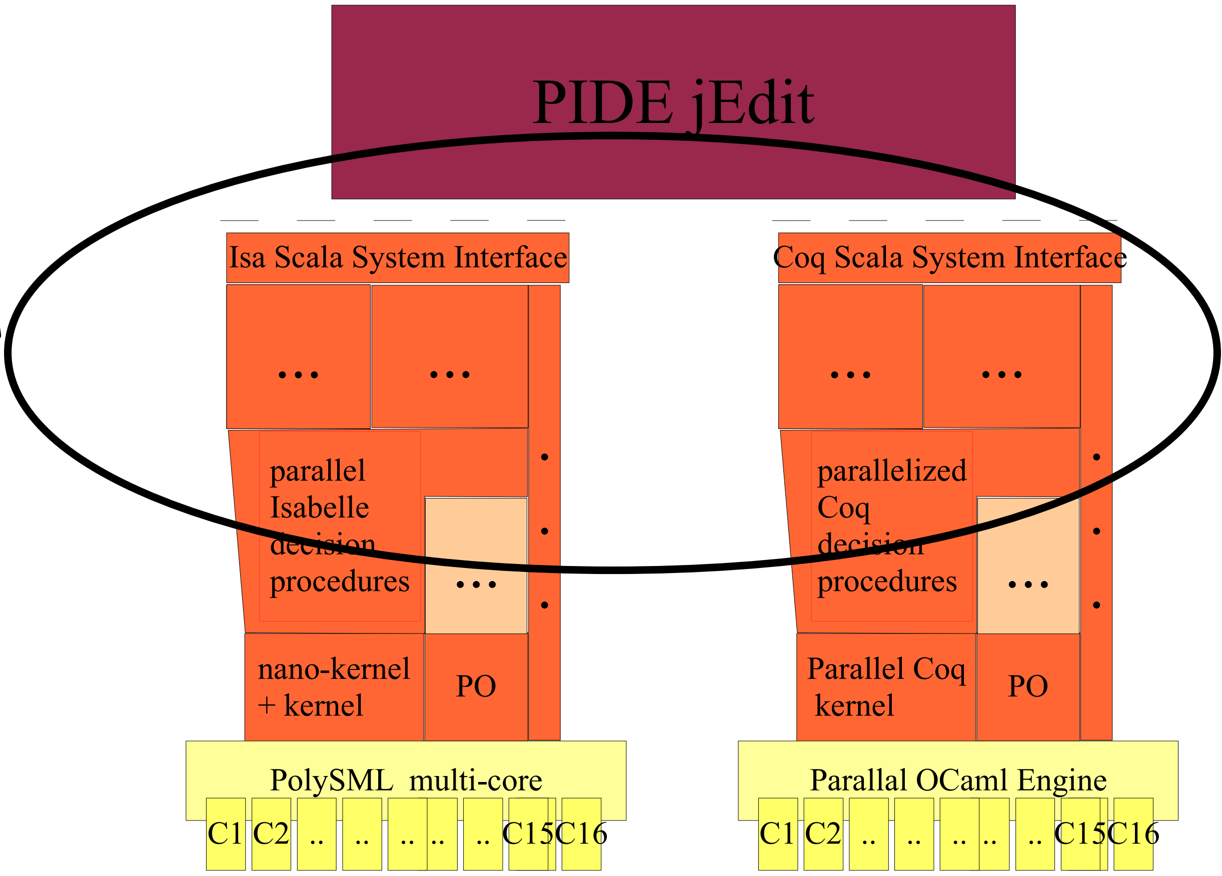
Topic Prover Architecture (PA)
- parallel ML
- parallel inference kernel
- parallel transactions over immutable contexts

The Project Goals at a Glance

- Paral ITP:

Topic Document Model (DM)

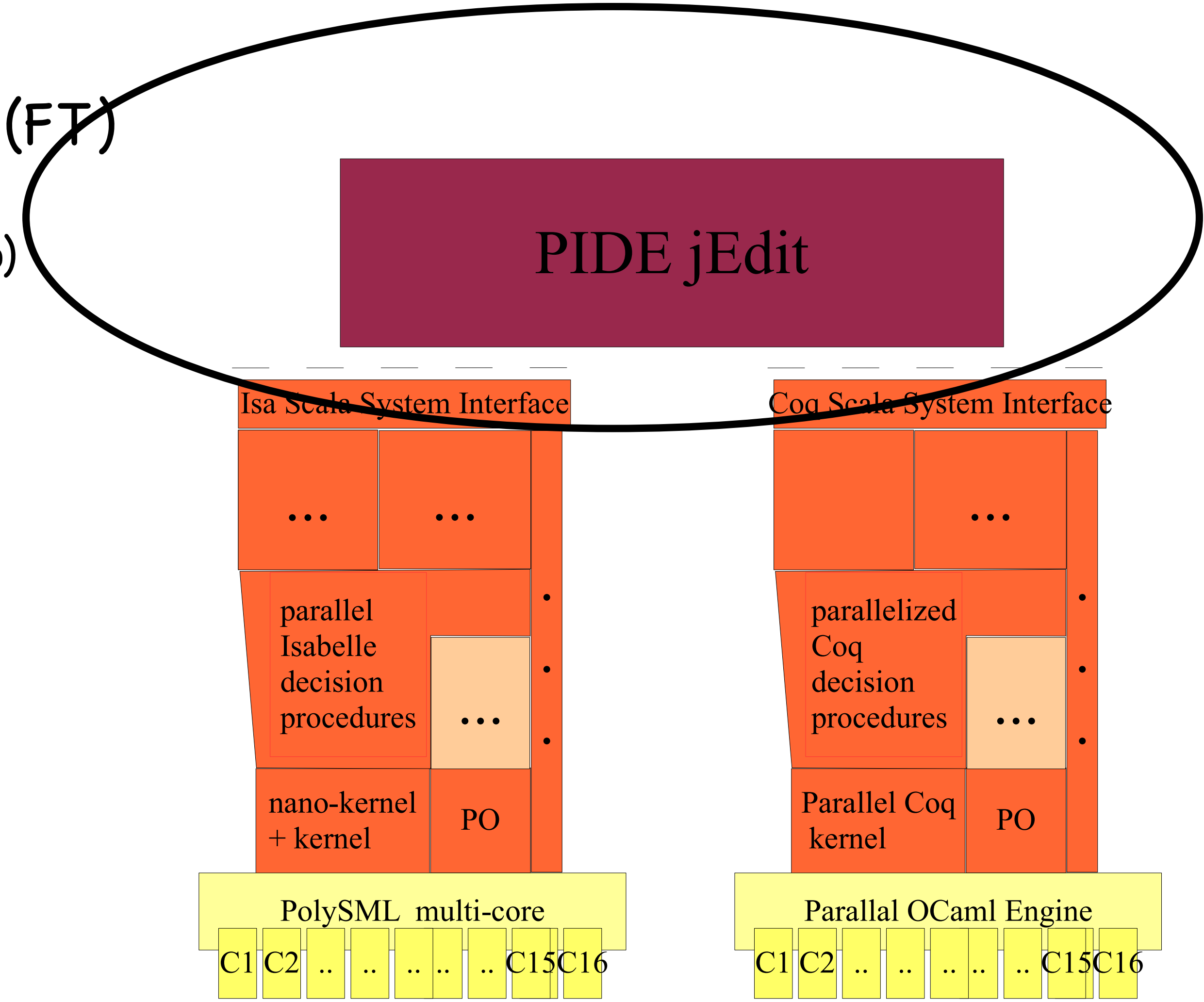
- Development of Formal, Generic DM
- Concurrent Changes
- Evaluation Strategies
- Persistent History
- Formal Content



The Project Goals at a Glance

- Paral ITP:

Topic Front-end technology (FT)
- Generic GUI Technology
- User Interactions (Rich Client, Web)
- Asynchronous Agents

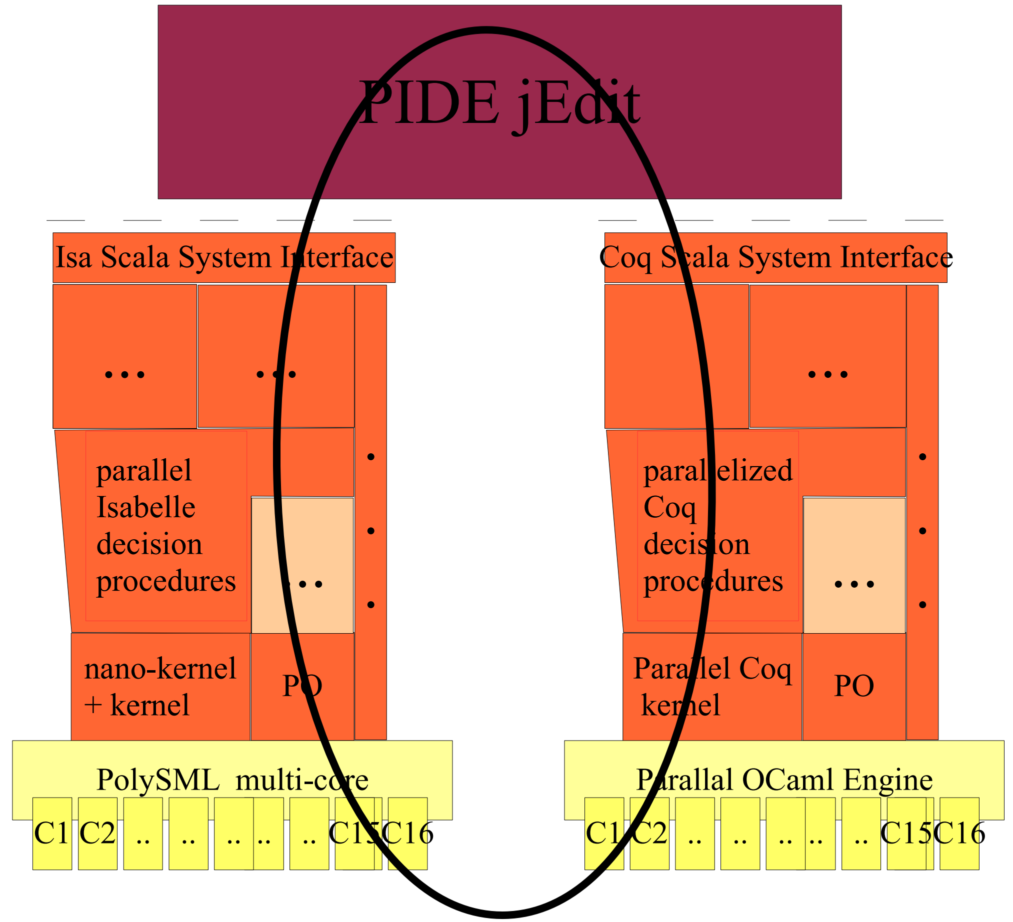


The Project Goals at a Glance

- Paral ITP:

Transversal activity.

Topic Formal Analysis (FA)
- Key Algorithms in Kernel (Context Transfer Check)
- Analysis of Persistent DM Algorithms
- Test-Generation of GUI Elements



First Results

Front-End Technologies

- Isabelle: PIDE / jedit is meanwhile robust and stable and part of the Isabelle Distribution.

  In Version 2013: probably the default interface.

- Support for advanced (nested) tool-tipping and hypertexting in the entire session.

- experiments with JAVA-Browsers.

- Coq: First Proof-of-Technologies to replace CoqIde available.

Prover Architecture

Application: AFP

Isabelle/AFP:

- $\approx$ 122 sessions with diversity of single-core run-time (3s . . . 1h)
- parameters of fully pervasive parallelism:

8 hardware cores / 16 CPU threads (Intel Xeon with hyperthreading)
4 parallel build jobs (Unix processes)
4 parallel ML worker threads (Isabelle/ML) 4 parallel GC threads (Poly/ML)
parallel theory and proof checking (Isabelle/Isar)

- timing results:

```
Finished LatticeProperties (0:00:15 elapsed time, 0:00:22 cpu time, factor 1.46)
...
Finished JinjaThreads (0:32:59 elapsed time, 1:56:55 cpu time, factor 3.54)
0:36:01 elapsed time, 5:17:18 cpu time, factor 8.80
```

Prover Architecture

- Isabelle: Substantial Performance Boost

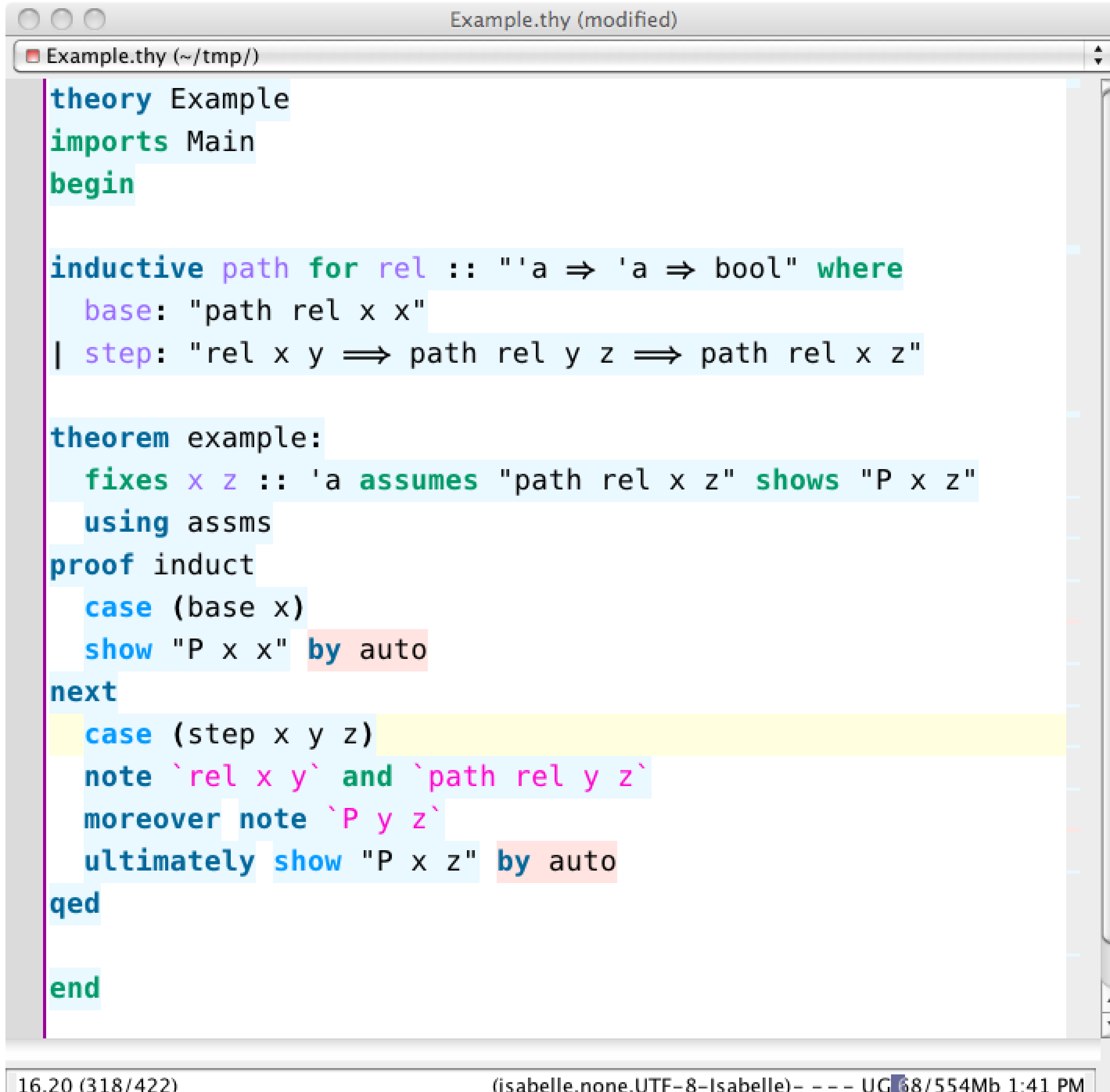
  The only parallel symbolic computing environment that scales to 8 cores (as far as we know).

- Coq: First Kernel renovation. Controlled side-effects, more elements in structured proof language, first experiments in concurrent validation of sub-proofs.

Parallel fine-grained validation of structured proofs in

in the

jEdit - PIDE



(Isabelle2012-D)

Prover Architecture

- Isabelle: Local Subproof-Parallelization works in current developper release reliably.

Prover Architecture

- Isabelle: Substantial Performance Boost

  The only parallel symbolic computing environment that scales to 8 cores (as far as we know).

- Isabelle:

Document Model

- Isabelle: Implementation in Scala supports entire "sessions" as DM's.
- Own experience: I will never ever will use Proof-General again !!! An IDE-like approch brings (at least for me) a sensible boost in productivity.
- Coq: First Formal Document Models on basis of HOAS under consideration.

The ISAR Document Model

- hierarchy of "documents" (theory files)
- atoms (units of text)
- syntax reconfigurable
- can be combined with SML code referring to kernel operations

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

Θ_{3-0} →

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

Θ_{3-1} →

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

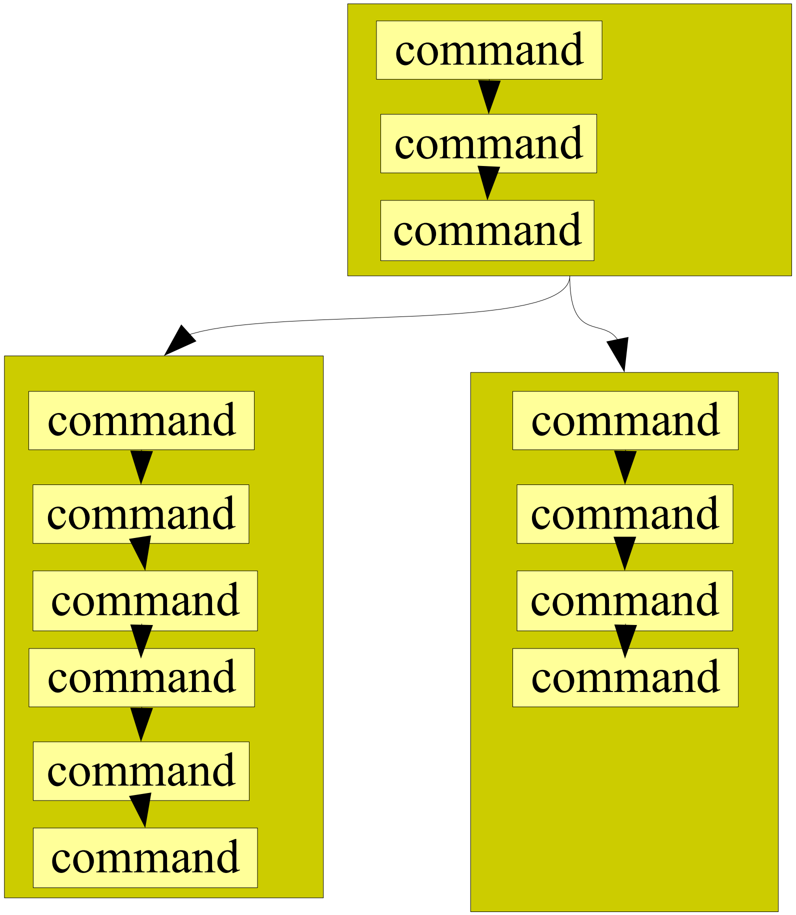
Θ_{3-2} →

```
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)"
```

Θ_{3-3} →

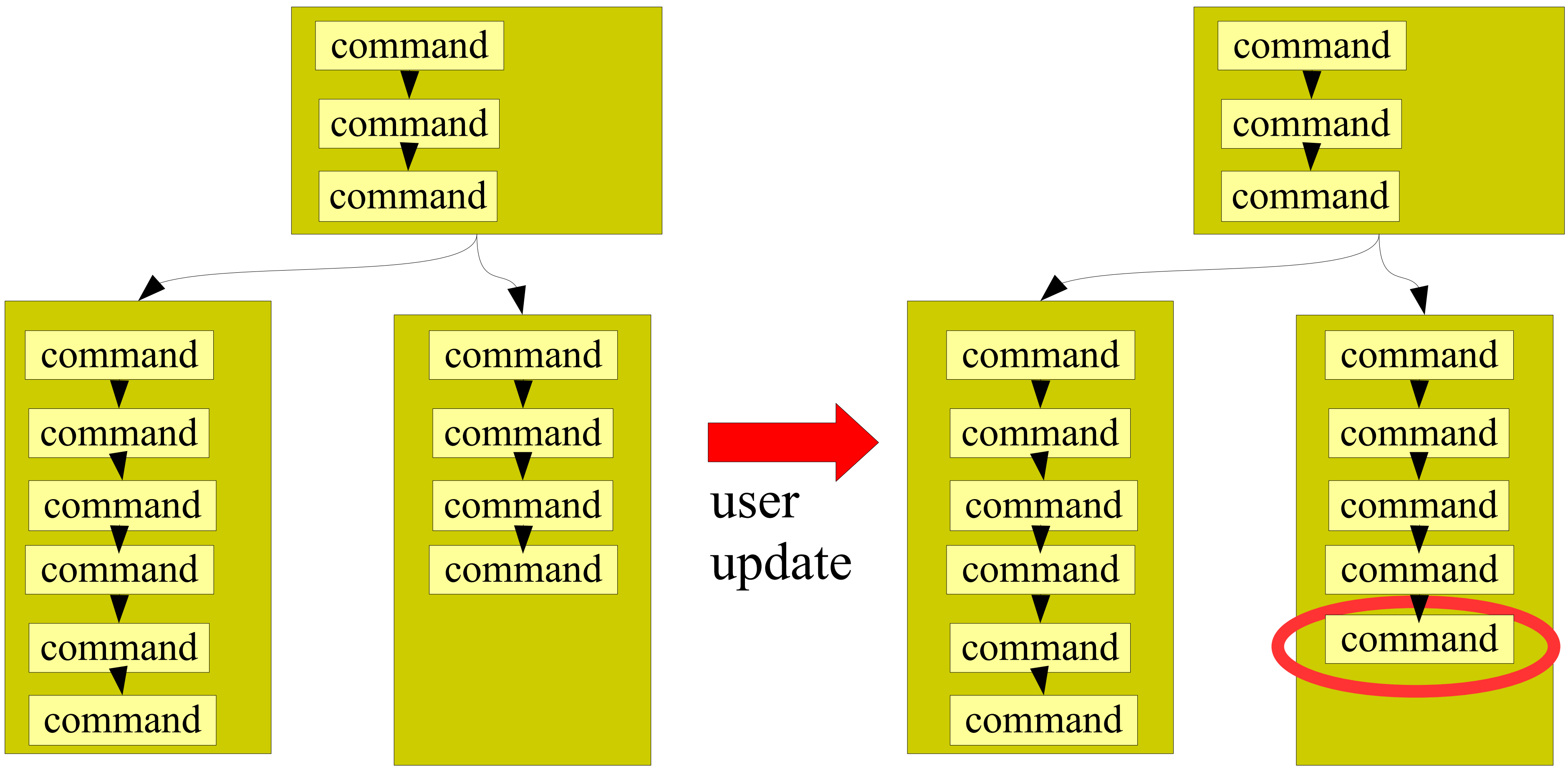
The ISAR Document Model

- document hierarchies,
- updates, and versions ...



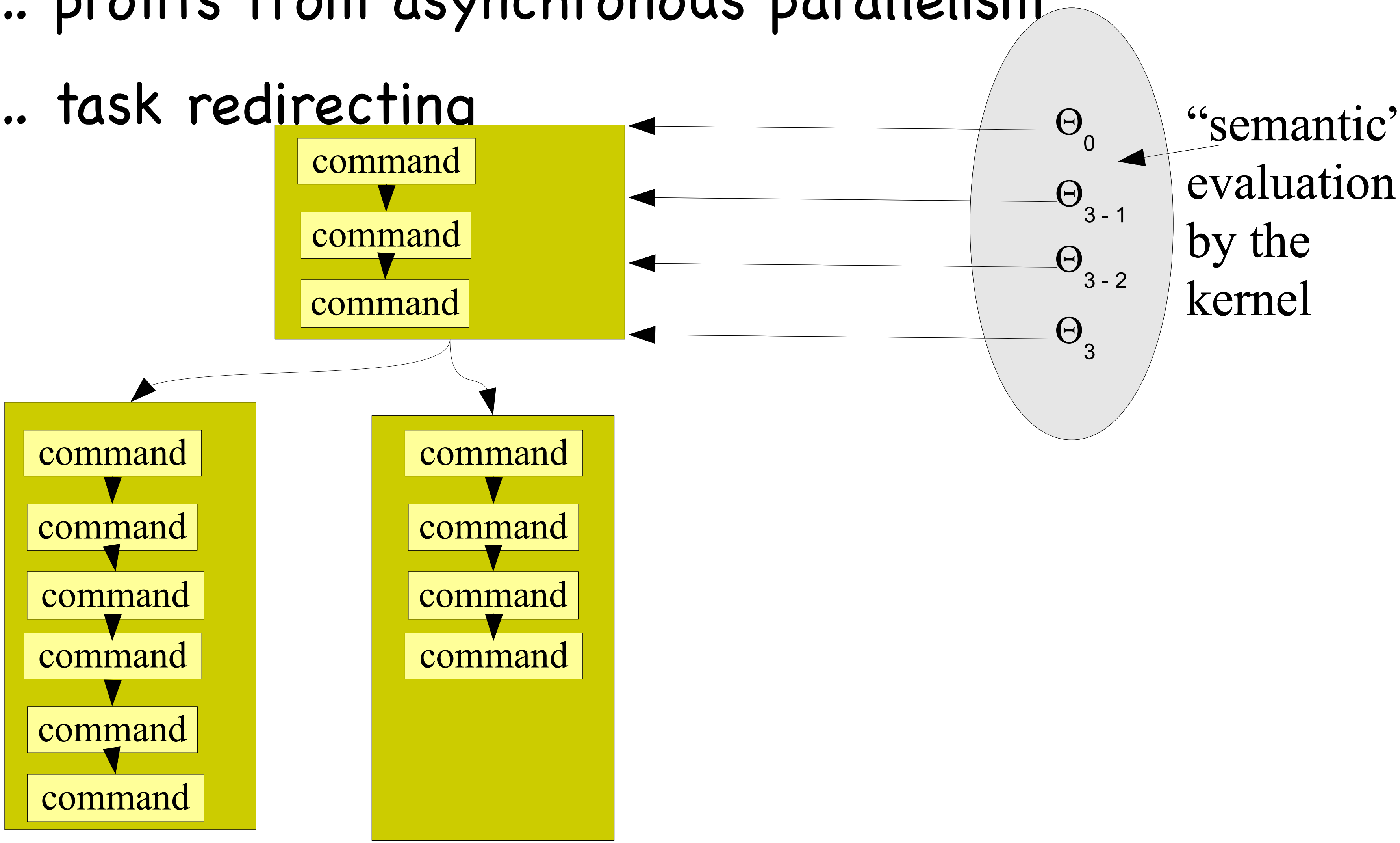
The ISAR Document Model

- document hierarchies,
- updates, and versions …



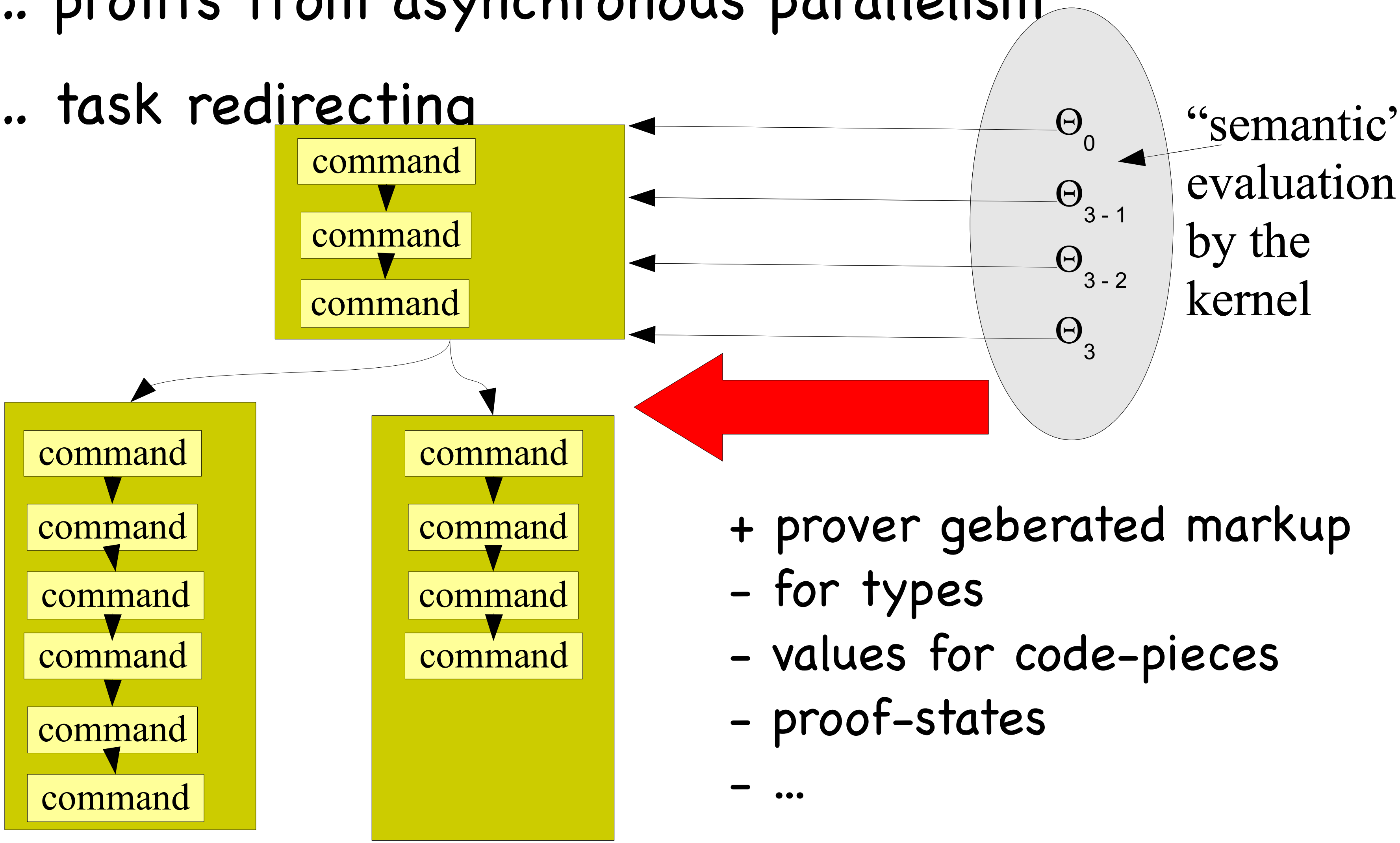
The ISAR Document Model

- … and its validation by the Isabelle Kernel
- … profits from asynchronous parallelism
- … task redirecting



The ISAR Document Model

- … and its validation by the Isabelle Kernel
- … profits from asynchronous parallelism
- … task redirecting



+ prover geberated markup
- for types
- values for code-pieces
- proof-states
- …

Formal Analysis

- First Formal Kernel Model under Development. Achieved: Formal Theory term, typ, and cterm (including type inference with Type Constructors)

  Goal: Relative Correctness Proof of the asynchonous Kernel wrt. to synchronous one: Whenever an parallelized proof (with all "promises" "fulfilled") exists, it corresponds to a conventional "non-parallel" proof.

Conclusion

Conclusion

- The research challenges:
  - Parallelized Prover Kernels
  - Parallelized API's for Symbolic Computing
  - Prover IDE's for Formal Mathematics and Large Program Verifications
  - Generic Prover IDE's for Domain-Specific Formal Languages

have been attacked an various levels, and at least on the Isabelle-side there is visible impact for end-users.

Conclusion

- Isabelle is at the moment slightly advanced in parallelization issues, ...
- ... on the other hand, the project has 2 years to go !
- Beyond practical evidence, theoretical evidence has to be provided that the logically safe, LCF-Kernel-based reliability of these systems is maintained ...

The Project Goals at a Glance

- Paral ITP:

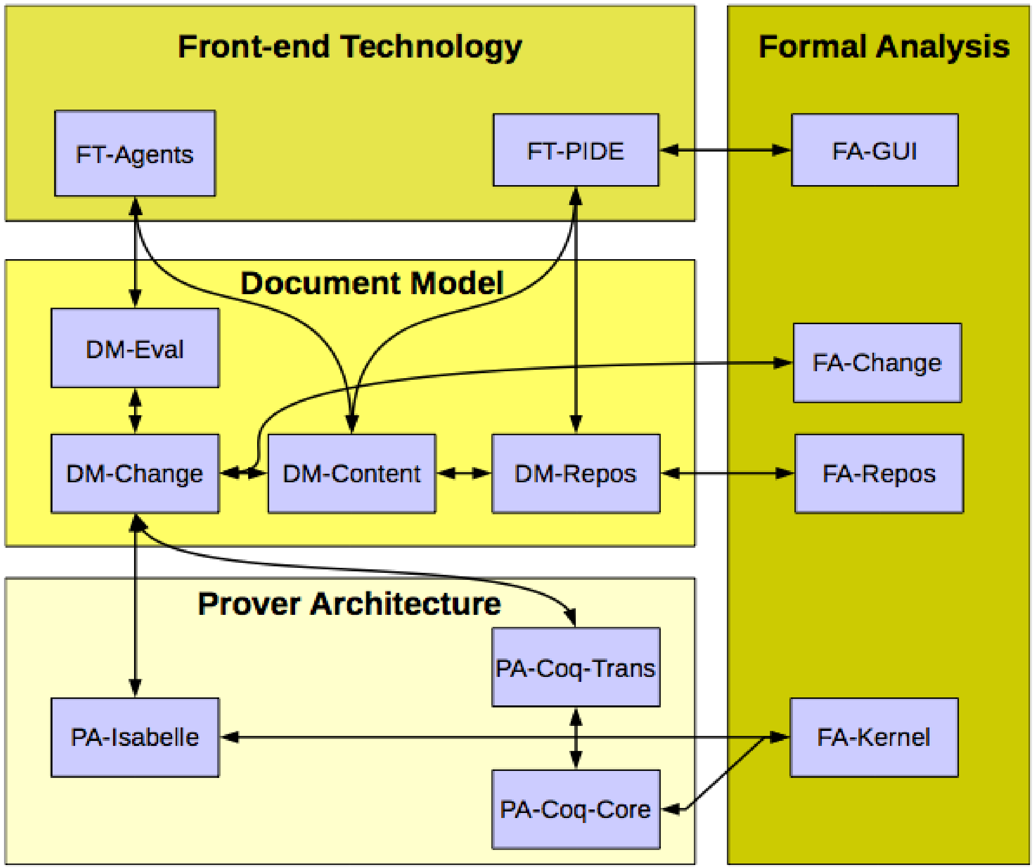


Figure 5: Tasks with topics and dependencies

The Classical LCF Kernel:

Coarse grained global context transition with branch and merge
(From the Beginning Specific forIsabelle 88)

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

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

„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:

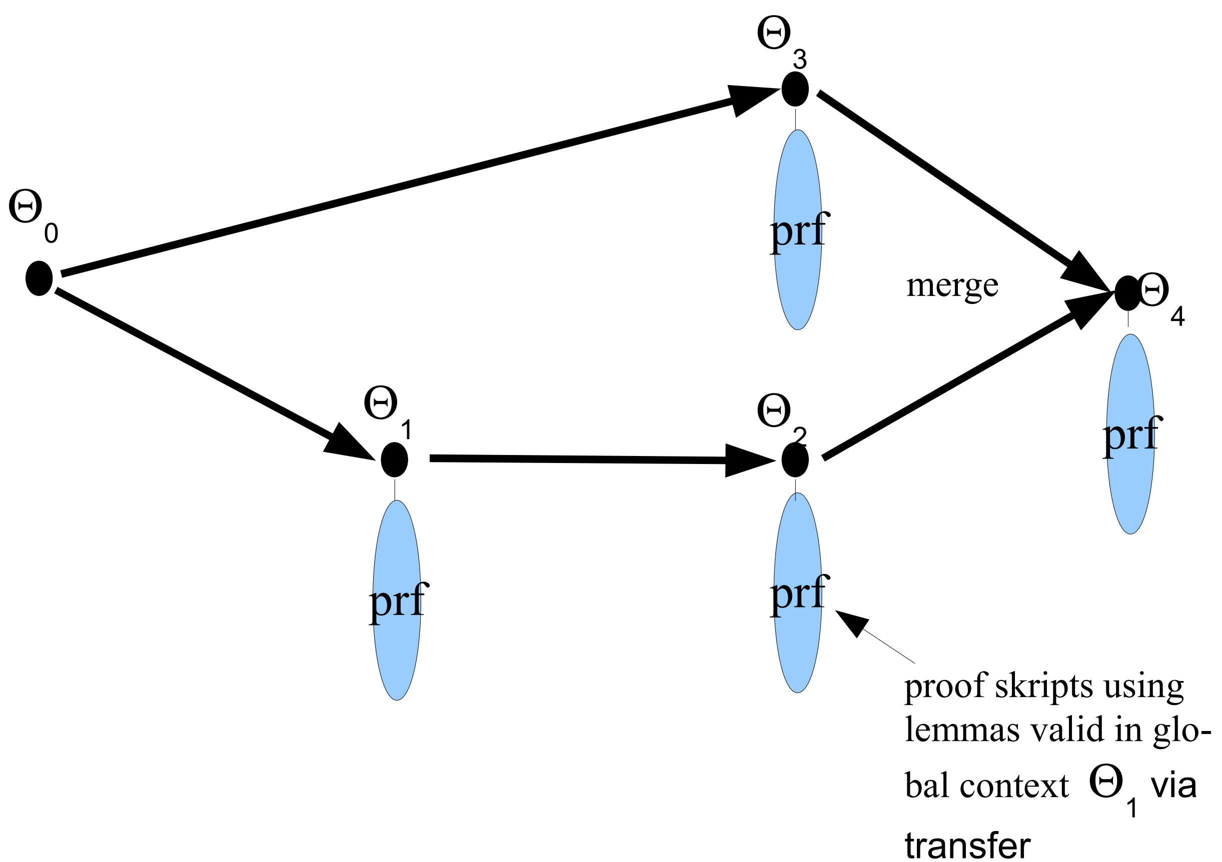
Typical Programming Interface

„$\varphi \vdash_{\Theta} \varphi$“	trivial Θ „φ“ :: thm
„$\Gamma \vdash_{\Theta} \varphi \; \{\xi \mapsto E$“	instantiate:: ... => thm => thm
„forward-chaining“	bi_compose :: 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:

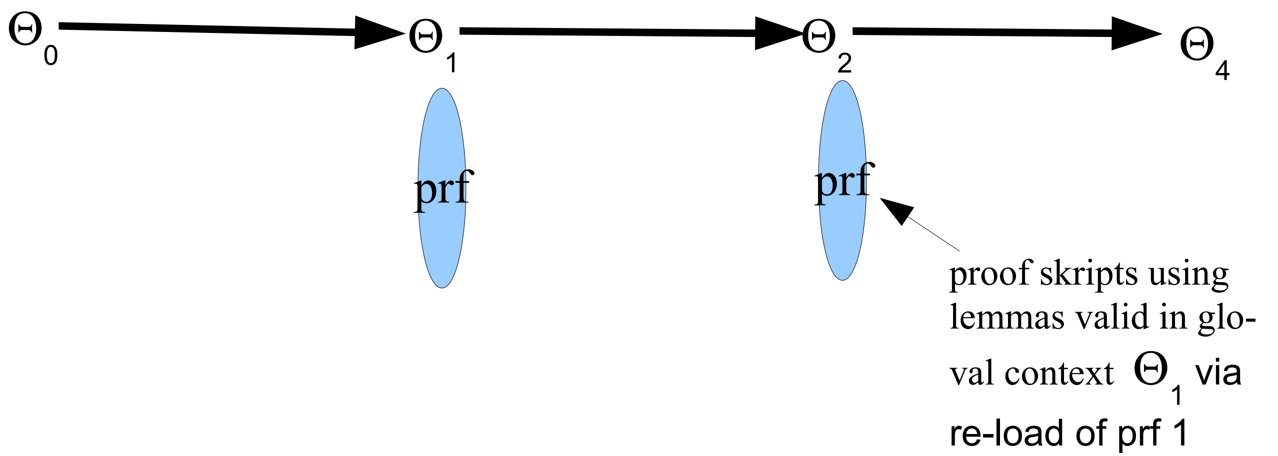
Coarse grained global context transition with branch and merge
(Isabelle 89 ... 94-4, ...)

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}

„$\_ \subseteq \_$“ 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:

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}

„$\_\subseteq\_$“ 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:

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}

„$\_ \subseteq \_$“ subthy: thy $\times$ thy $\rightarrow$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:

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
...}

record extensions for Isabelle Components (rewriter, datatype package...) which must be functional

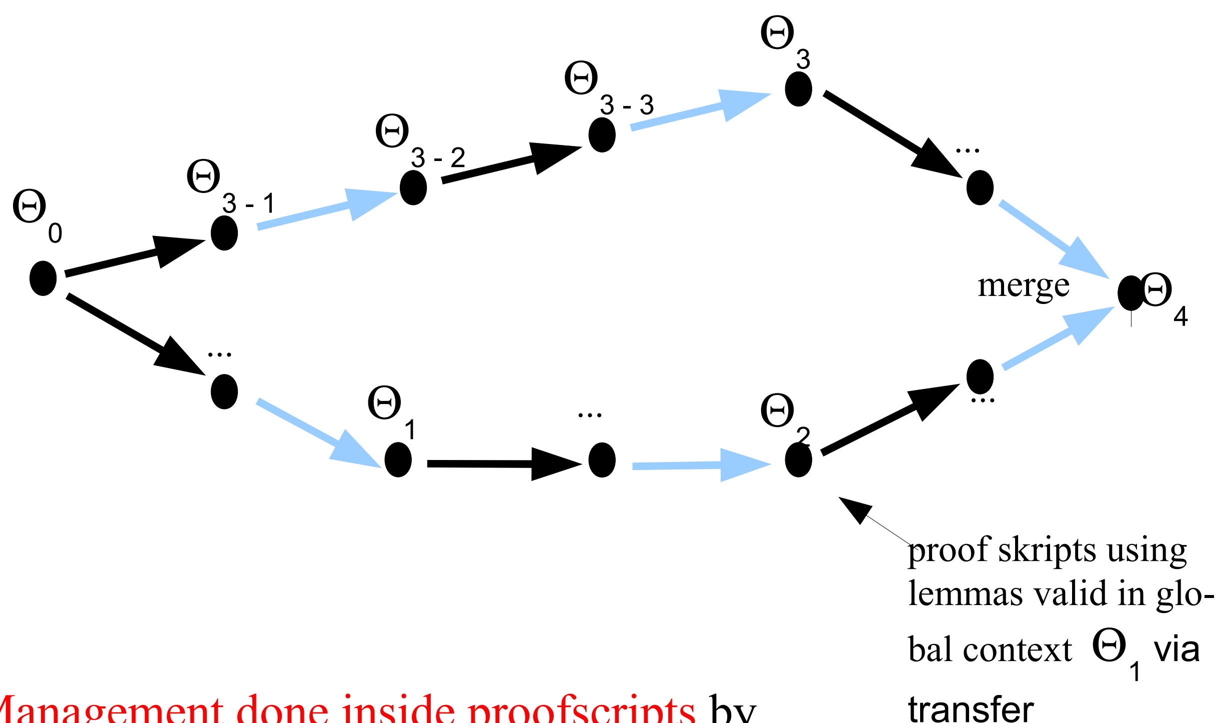
„$\Gamma \vdash_{\Theta} \varphi$“ thm = {context:thy,
hyps:term list,
prop:term}

„_⊆_“ 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 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 ⇸ thm,
                              ...}
„Γ ⊢_Θ φ“       thm = {certificate : CertId,
                       hyps : term,
                       prop : term}

                CertificateTable :   CertId ⇸ thycontext

„_⊆_“           subthy: thycontext × thycontext →bool
```

The Nano-Kernel LCF - Architecture:

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

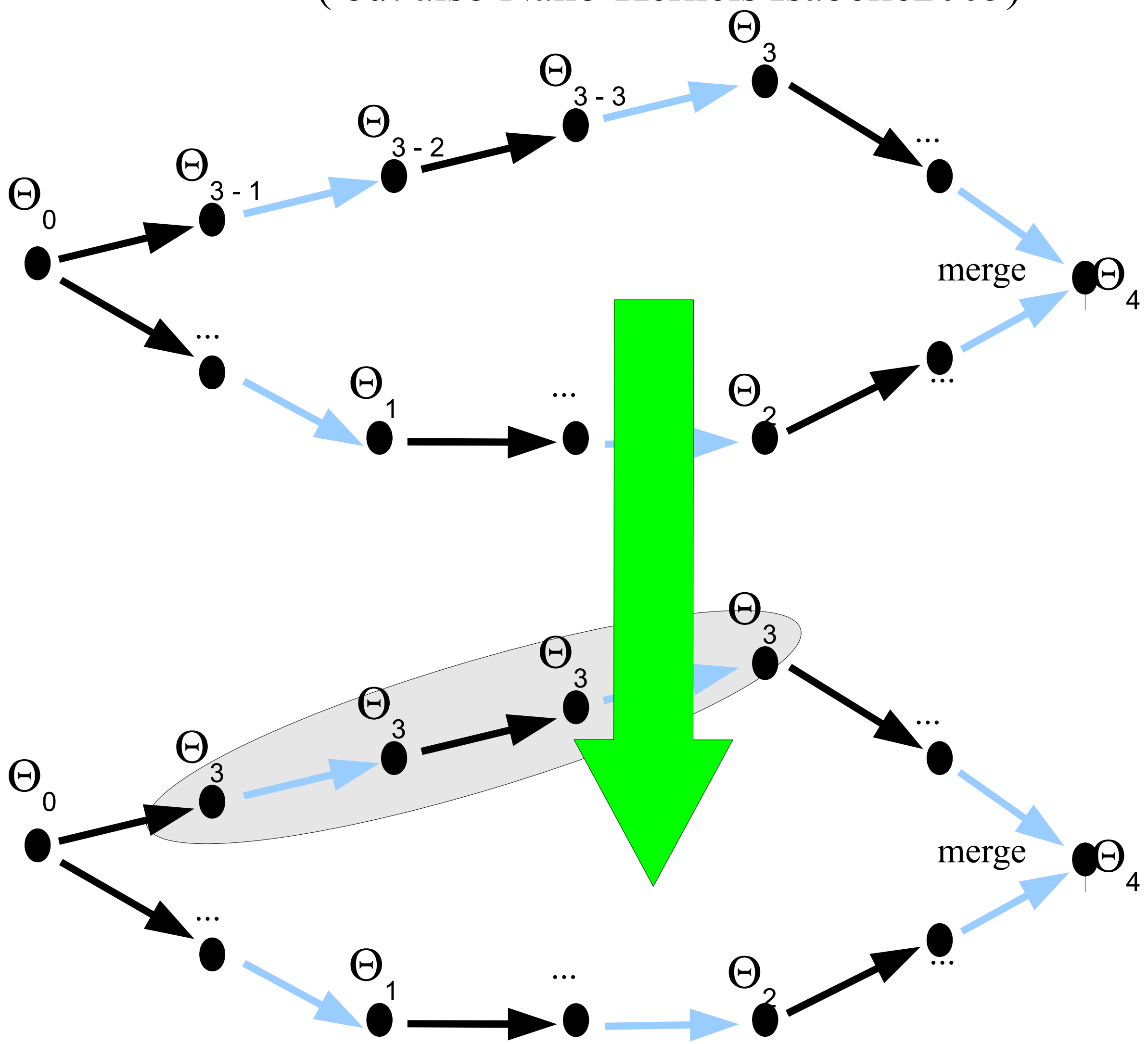
```
proofcontext = context + {
                           theory_of_proof : CertId,
                           fixes : string list,
                           assumes : term list,
                           ...}
```

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...

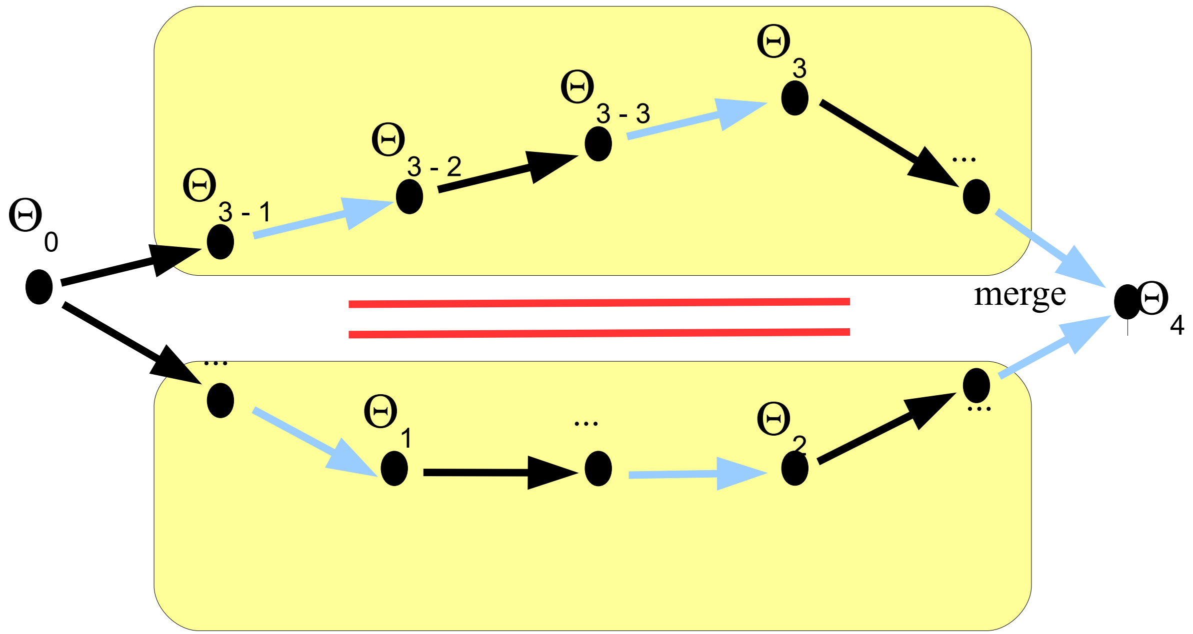
Nano-Kernel LCF-Architecture:

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 $\rightarrow\!\!\!\!\!\!\rightarrow$ thm,
...}

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

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

...

Parallel Nano-Kernel LCF - Architecture:

Putting the Classical Kernel actually into Plugins ...

Isabelle2009 - 10 (!)

...

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

“holes” in proofs to be filled in asyn-chronously later

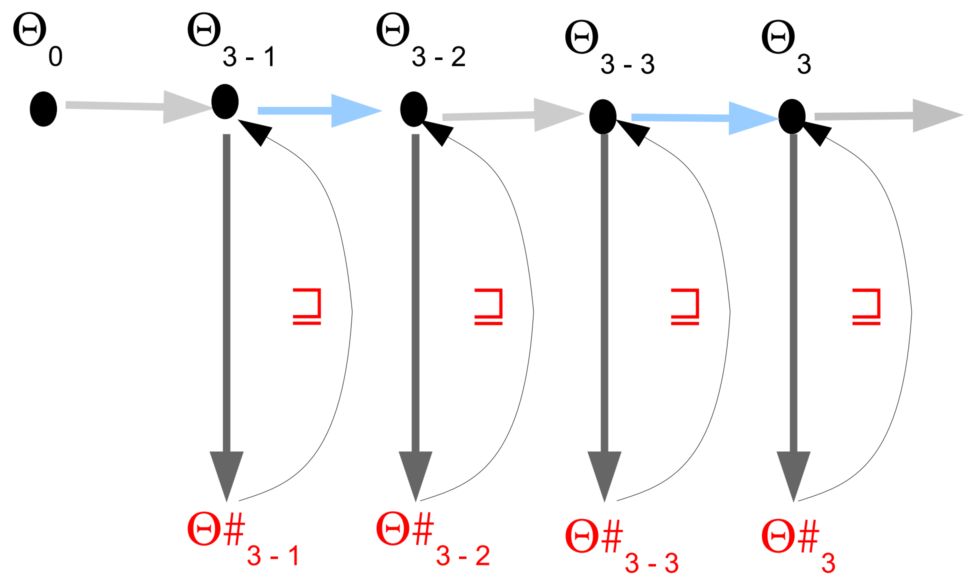
„$\Gamma \vdash_\Theta \varphi$“ thm = {context : CertId,
promises: name ⇸ thm future,
hyps : term,
prop : term}

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).

The Evolution of Document Models

The Role of Document Models in the ITP Programme

- Presentation is Key in ITP Design
- The notion of document becomes the center of ITP; theory development is document-centric!
- for common user-interfaces (like ProofGeneral) generic document models had been developed.
- what is the document -
  - a "bunch of emacs-buffers!" (David Aspinall, 03)
  - a data-structure (tree - dag - graph) of code/definitions/proofs/text/documentation (= formal content) ?
  - ... textual presentation is actually accidental.

The Role of Document Models

- An early, abstract, visual document model: the IsaWin System [Wolff, Lüth 97]
  - notepad metaphor
  - … and explicit, generic document model (objects, types, operations, presentations)

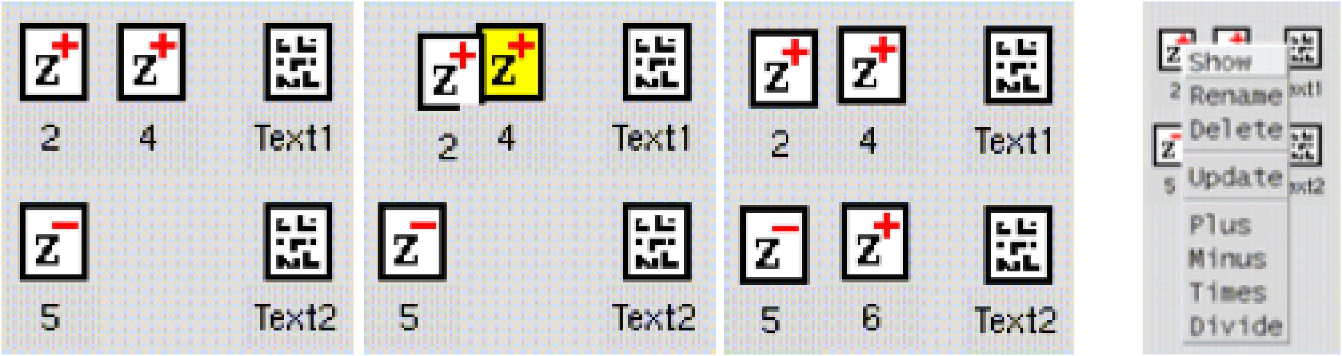


Fig. 1. Introducing the notepad metaphor and manipulation by drag&drop.

The Role of Document Models

- An early, abstract, visual document model: the IsaWin System [Wolff, Lüth 97]
  - notepad metaphor
  - … and explicit, generic document model (objects, types, operations, presentations)

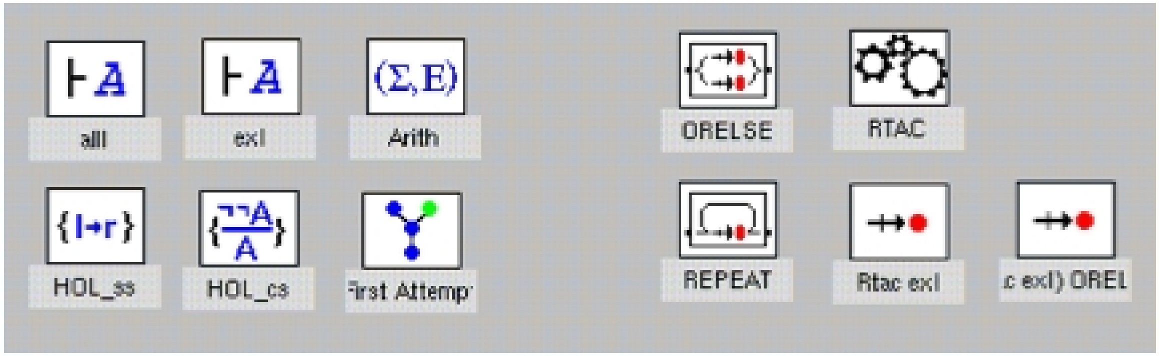


Fig. 3. The Objects of IsaWin: to the left, basic objects; to the right, tactical objects

The Role of Document Models

- An early, abstract, visual document model: the IsaWin System [Lüth,Wolff 97]
  - notepad metaphor
  - … and explicit, generic document model (objects, types, operations, presentations)
  - … implemented by an SML functor mapping the "DM" of an application to its notepad …

4.2 The Generic Graphical User Interface GenGUI

The module GenGUI uses the interface description facilities provided by sml_tk to provide a generic graphical user interface. It is independent of Isabelle, and given as a functor

```
functor GenGUI(structure appl: APPL_SIG) : GEN_GUI = ...
```

The Role of Document Models

- An early, abstract, visual document model: the IsaWin System [Wolff, Lüth 97]
  - a "DM" was:

```
signature APPL_SIG =
sig
  type object             (* The type of all objects *)
  eqtype objtype          (* The type of object types *)
  eqtype mode             (* The type of modes *)
  val obj_type            : object -> objtype
  val modes               : objtype-> mode list
  val mode_name           : mode-> string
  val initial_mode        : object-> mode
  val construction_obj    : objtype
  datatype object_result = OK of object | Error of string
  type opn
  val apply   : opn* object list-> object_result
  val mon_ops : objtype-> ((object* (opn->unit)-> unit)* string) list
  val bin_ops : (objtype* mode)* (objtype* mode)-> opn option
```

The Role of Document Models

- The IsaWin System - why didn't it work out?
  - development divergences in the presentation layer (code wars in the SyntaxEngine)
  - too visual; textual representation needed
  - no states, but versioning; ok. BUT: naive functional evaluation model.
    - no interrupts
    - no asynchronous communication
    - not dynamic - extensions had to be recompiled.

The Role of Document Models

- Current Isabelle/ISAR [Wenzel 98 - 11]
  - textual (perhaps even a bit too much)

    (but everything you can do with Unicode)
  - Prover IDE oriented:

    tooltips, hovering, continuous check & build
  - asynchronous, parallel
  - highly dynamic and reconfigurable
    (the "ISAR-language" is actually just a
    config of the Isabelle/ISAR machine)
  - programming: PURELY FUNCTIONAL

Position of the Consortium

- Coq Core Developers
  - DR Dr Hugo Herbelin
    Coq Development Coordinator, INRIA Roquencourt
  - CR Dr Bruno Barras

    Coq CTO, Inria Saclay
  - Dr Damien Dogliez
    OCaml Core Developper,
    INRIA Roquencourt

Position of the Consortium

- Isabelle Core Developers
  - Dr M. Wenzel: CTO of Isabelle since 99,
    Initiator of Parallelization in Isabelle
  - Prof. Dr Burkhart Wolff
    Developper of Tools on Isabelle-Kernel
    Expert in Formal Analysis

Working Organization

- Major Working Axes
  - DR Dr Hugo Herbelin   Leader DM
    (coll. M. Wenzel, B. Barras, Yann Régis-Gianas, B. Wolff)
  - CR Dr Bruno Barras   Leader PA
    (coll. M. Wenzel, Damien Dogliez)
  - Dr Makarius Wenzel   Leader FT
    (coll. B. Barras, Yann Régis-Gianas, B. Wolff)
  - Pr Dr Burkhart Wolff   Leader FA
    (coll. M. Wenzel, Yann Régis-Gianas)

Working Organization

- Working Axes + Smaller Work-Packages
  - DR Dr Hugo Herbelin Leader DM
    (coll. M. Wenzel, B. Barras, Yann Régis-Gianas, B. Wolff)
    +Pierre Courtieu, Olivier Ponsm Matthieu Sozeau, Assia Mahboubi.
  - CR Dr Bruno Barras Leader PA
    (coll. M. Wenzel, Damien Dogliez)
    +Pierre Courtieu, Olivier Pons, Germain Faure, Assia Mahboubi.
  - Dr Makarius Wenzel Leader FT
    (coll. B. Barras, Yann Régis-Gianas, B. Wolff)
    +Delphine Longuet, Frédéric Voisin, Pierre Courtieu, Olivier Pons.
  - Pr Dr Burkhart Wolff Leader FA
    (coll. M. Wenzel, Yann Régis-Gianas)
    +Delphine Longuet, Frédéric Voisin, Olivier Pons,Assia Mahboubi.

The Project Organization

- project infra-structure
  - repositories
    - Common (archiv, pub, reports, presentations) :
      https://www.lri.fr.svn.fortesse/anr-paral-itp
      access already distributed
    - INRIA - git for Coq - Contributions
    - Munich hg for Isabelle - Contributions
  - web-page (http://paral-itp.lri.fr/)
  - wiki (not yet)

The Project Organization

- reporting & project output
  - we are in bus distance to each other !
  - regular meetings in each “Topic”
  - 6 month meeting, 6 month reports
  - annual software publications (Coq&Isabelle)

The Project Organization

- IPR Issues
  - Longstanding Open-source tradition for all three, independent components:
    - Isabelle: TUM+UCam+"Collaborators" (us)
    - Coq: INRIA
    - ProofGeneral: UEdin / Replaced by Isabelle
  - Open Source Licences:

    "All software-components produced in the project will be published with Open Source Licenses that are compatible with the respective prover distributions (Coq: LGPL, Isabelle: BSD, contributing tools: BSD, LGPL, GPL). This is achieved either by using sufficient liberal licensing from the start (BSD) and implicitly strengthen towards GPL, or by dual-licensing of certain components. Thus the integrated systems will be usable by academic and industrial users alike, according to established practice both in the Coq and Isabelle communities."

Conclusion

- To advance the ITP Programme
  - more specific asynchronous computation models were needed to use modern parallel hardware
  - more advanced generic document models were needed
  - advanced API's for using ITP's as "Eclipse of FM Tools"
- Still, the LCF-Kernel Character needs to be maintained ...