Isabelle Tutorial:
System, HOL and Proofs

Burkhart Wolff

Université Paris-Sud

What we will talk about

What we will talk about Isabelle with:

- Brief Revision
- Advanced Automated Proof Techniques
- Structured Proofs
  (“declarative style”)

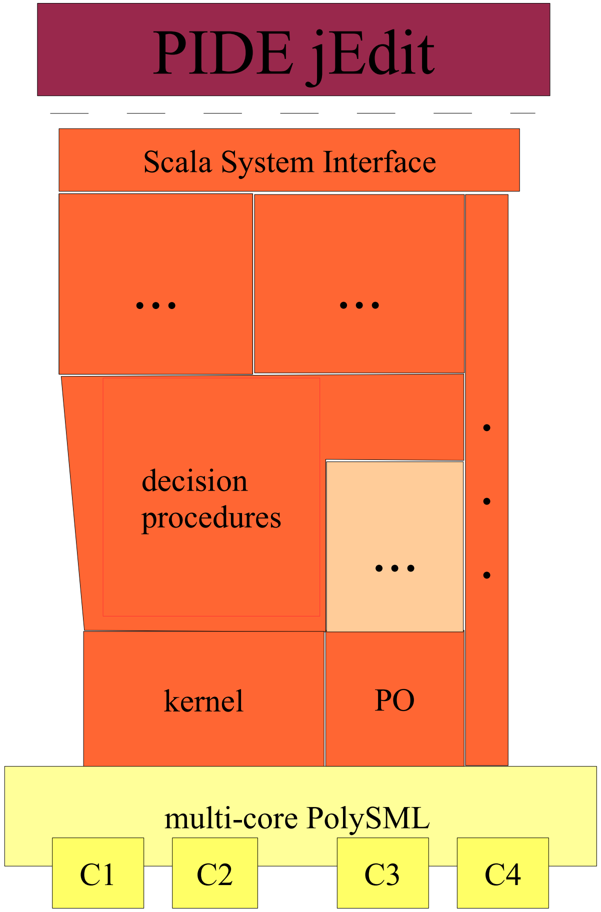
Isabelle

Isabelle is

- A Kernel-based Interactive Modeling, Programming and Theorem Proving Environment
- … in the Tradition of LCF style Provers
- … purely functional, highly parallel execution environment

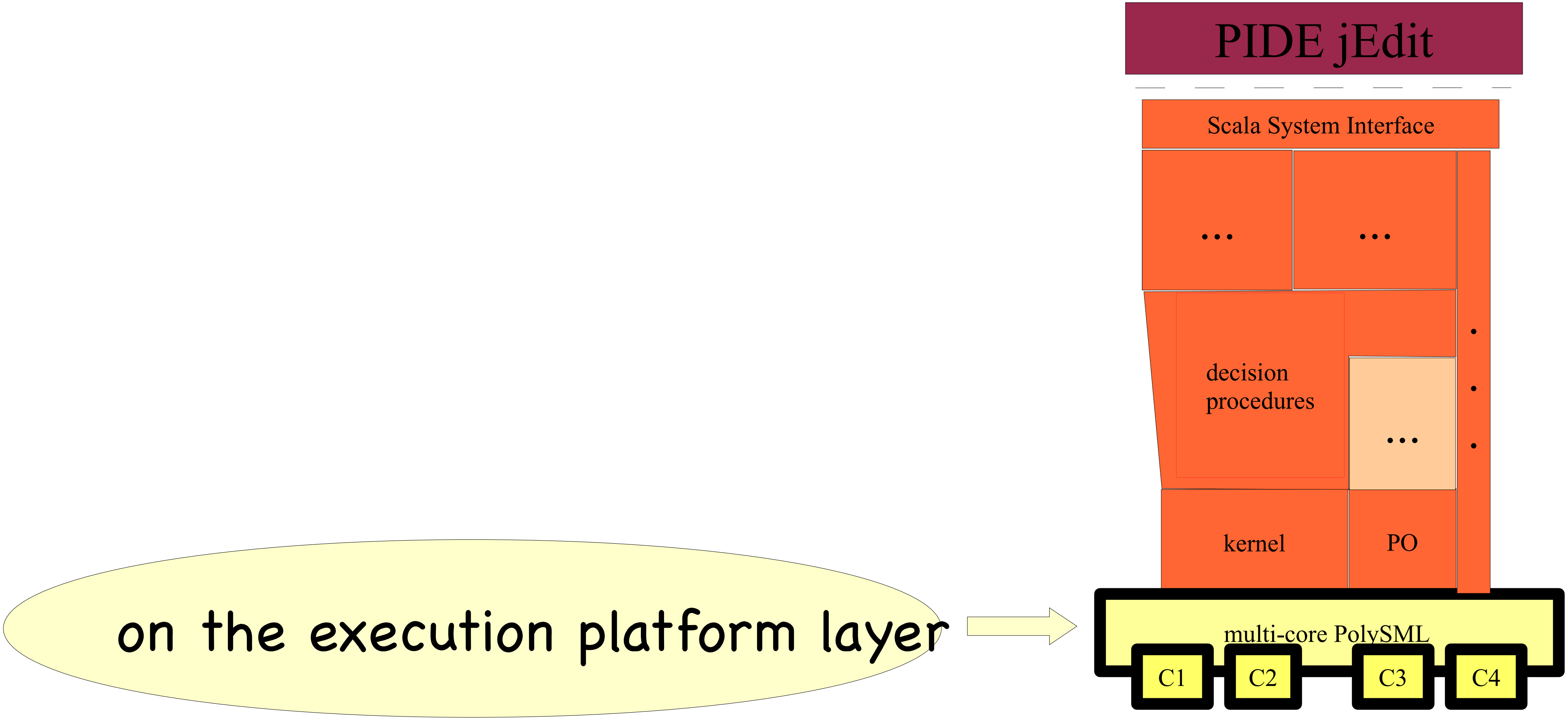
Isabelle Architecture

- Observation:
  Effective parallelization is a PERVASIVE PROBLEM, that must be addressed



Isabelle Architecture

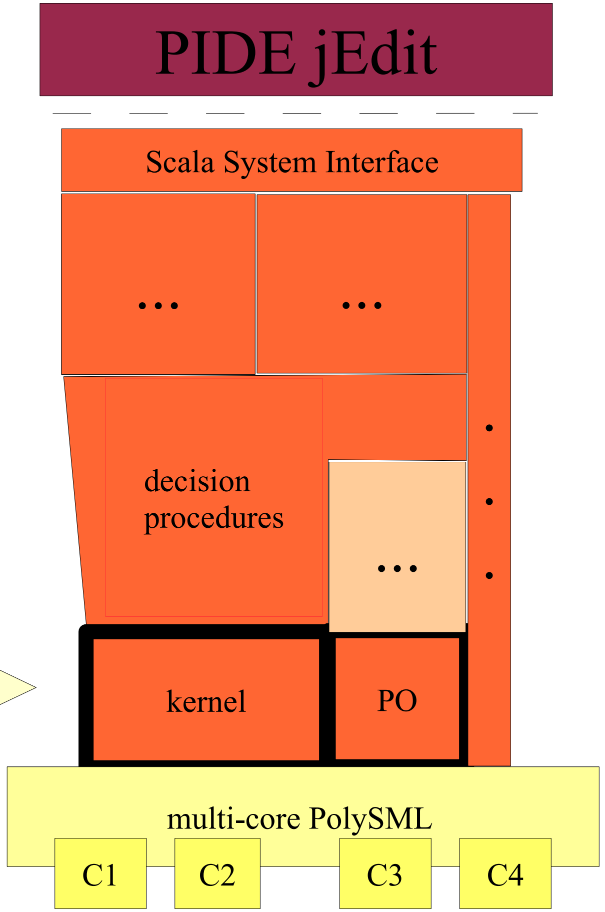
- In detail:



Isabelle Architecture

- In detail:

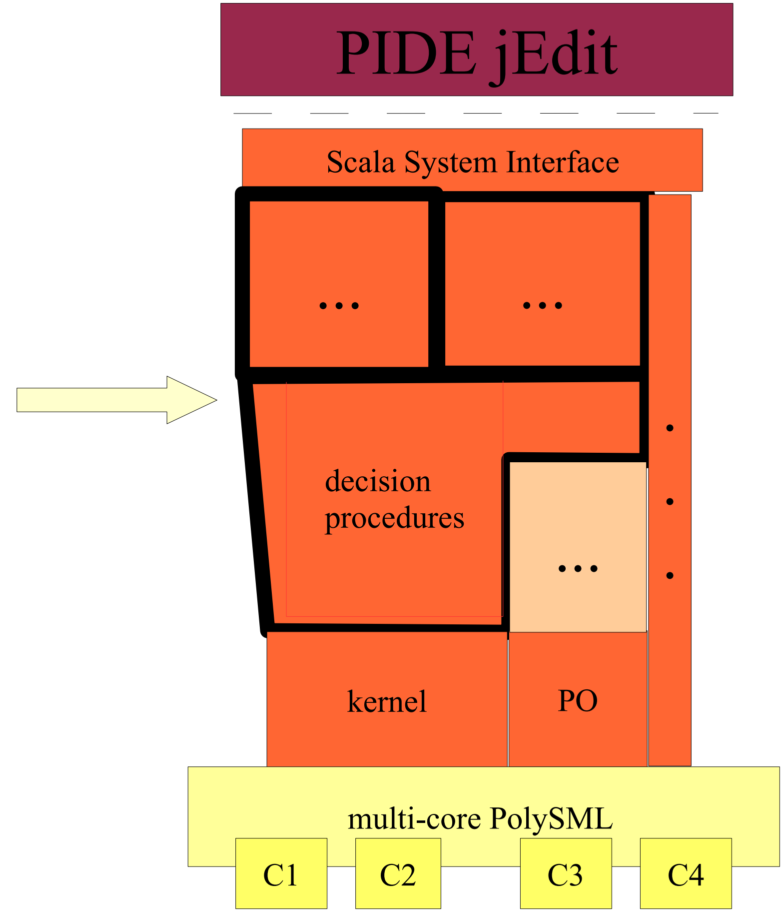
on the kernel layer



Isabelle Architecture

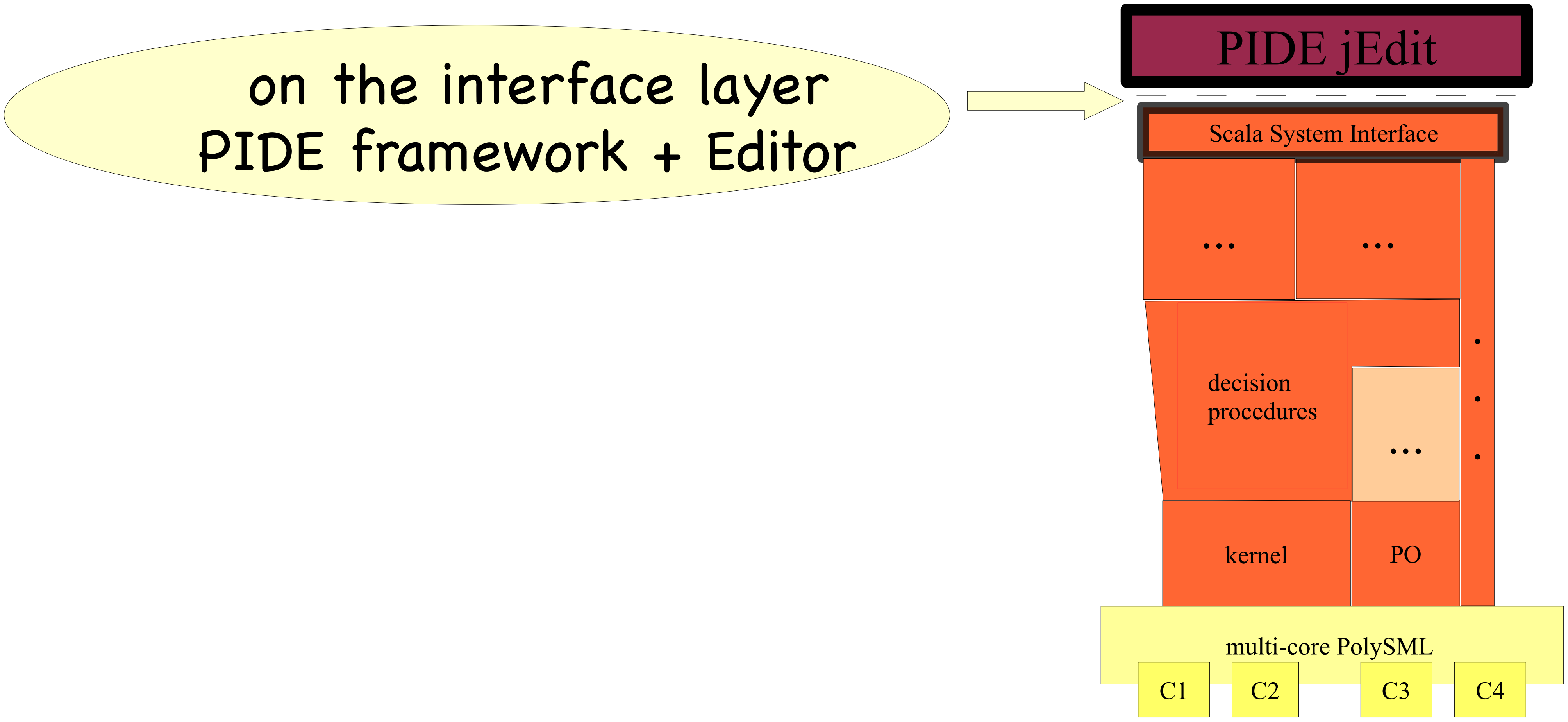
- In detail:

on layer of procedures and packages



Isabelle Architecture

- In detail:



External Provers

Provers in Isabelle

Commonly used internal procedure:

- fast, formerly: fast_tac (via auto)
  higher-order tableaux prover.
  As tactic implemented in the decision
  procedure level.
- requires (f|d|e)rule - instrumentation
- HO-Logics, Quantifier-reasoning, Sets,
  not very strong with large rule sets.

Provers in Isabelle

Commonly used internal procedure:

- simp, formerly: simp_tac (via auto)
  higher-order rewriting prover.
  As tactic implemented, fully internal.
- requires simp-cong-split - instrumentation
- Quantifier-reasoning, Sets;
  pretty strong even with large rule sets.
- Supports HO-order pattern ordered
  context rewriting with splitting.
  Debugging cycles in large rewrite sets
  can be tedious.

(External) Provers Isabelle

Commonly used internal externals :

- blast, formerly: blast_tac (via auto)
  first-order tableaux prover.
  In SML implemented, semi-external, reconstruction via PO's.
- requires (f|d|e)rule - instrumentation
- Quantifier-reasoning, Sets, transitivity;
  but not not very strong with large rule sets.
- Limited wrt. Quantifier alternations, usually faster than fast though.

Provers in Isabelle

Commonly used internal procedure:

- auto, combination tactic consisting essentially of
  - simp
  - blast

  Nowadays no longer the strongest prover, but interactively highly useable, highly configurable
  (requires simp and blast instrumentation)

Provers in Isabelle

Commonly used internal procedure:

- arith, a tactic solving linear arithmetic. Implemented as tactic decision procedure.

  Powerful, but relatively slow.

(External) in Isabelle

Commonly used semi-internal procedure:

- metis, an SML implementation for a first-order prover with equality based on ordered paramodulation. Proofs integrated in Isabelle by tactic reconstruction.

  NO INSTRUMENTATION NECESSARY.

  Working with it incrementally is impossible.
  Nowadays usually backend of sledgehammer ;-)

External in Isabelle

Commonly used external procedure:

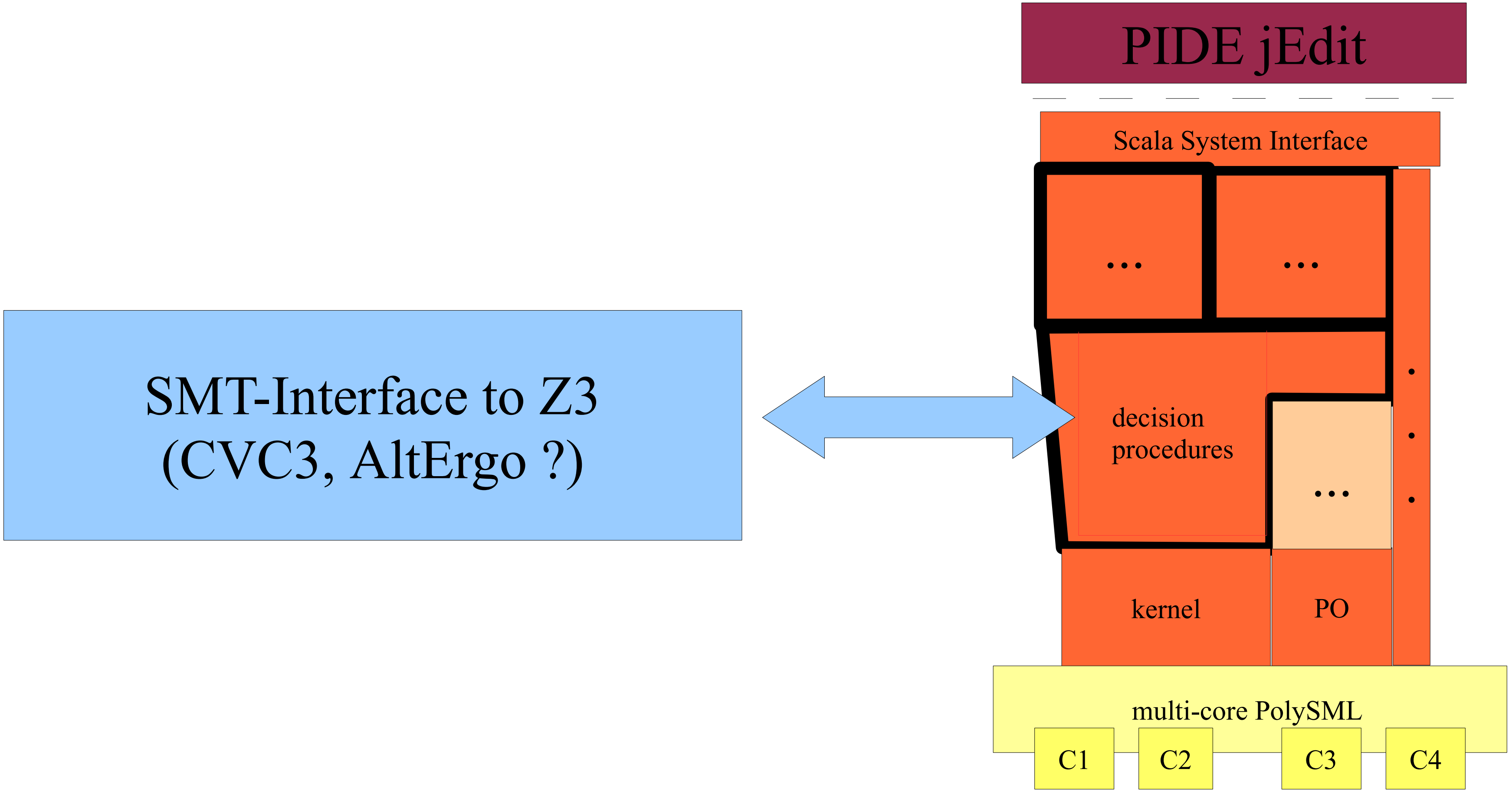
- smt, an SML interface for SMT solvers supporting the SMT-lib Format.

  Tuned for Z3 (which must be ticked for "non-commercial use"), for which a tactic reconstruction of the proofs has been developed. Quantifiers need instrumentation(Triggers).

  VERY POWERFUL for First-Order proofs with Built-In- Z3 Theories, but discouraged for use in final proof documents. Needs instrumentation.

Isabelle Architecture

- In detail:



External in Isabelle

Commonly used external prover interface

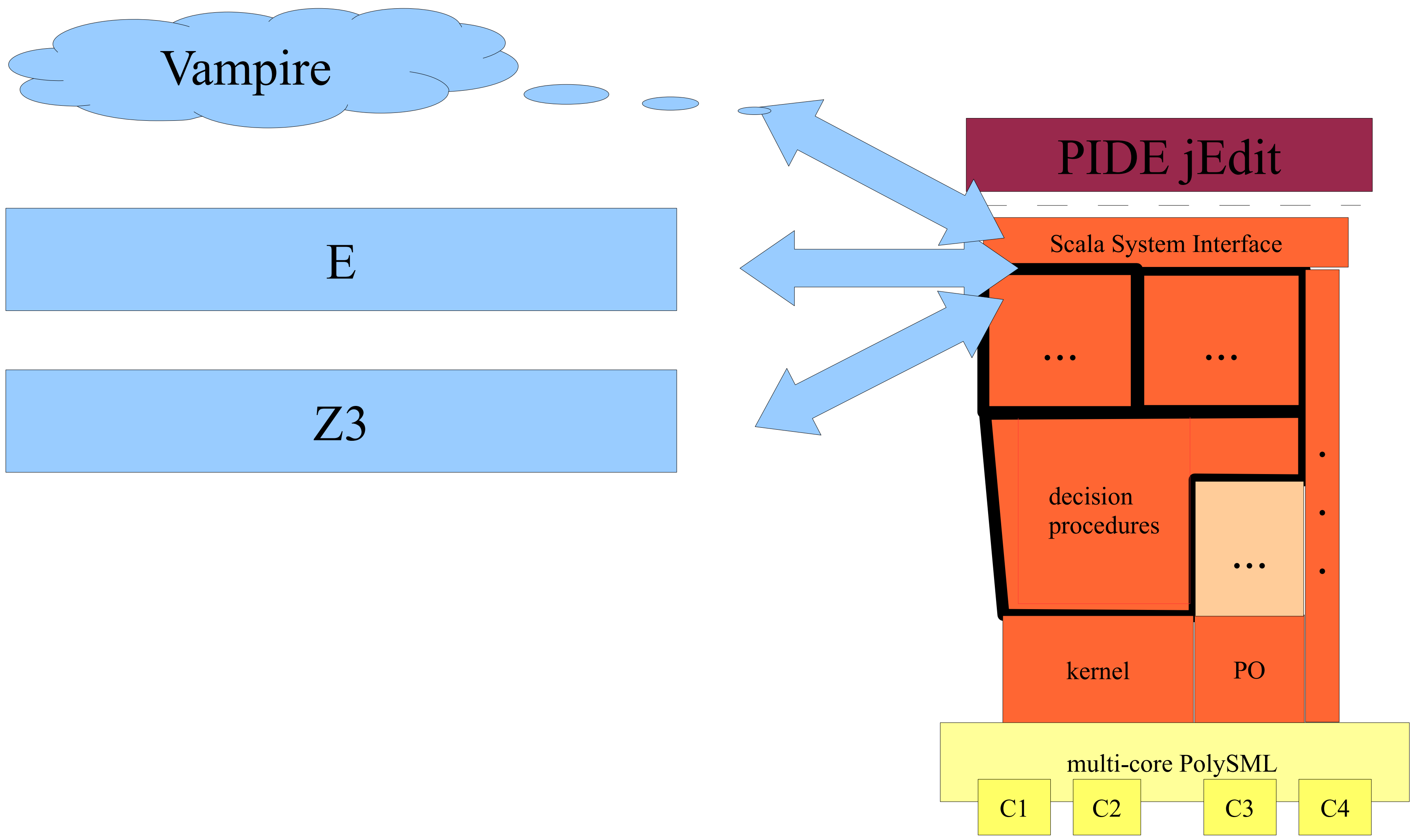
- sledgehammer, an interface to external provers, which can be server applications.
  - local provers: E (first order with equality), Z3 as smt interface
  - server applications: Vampire,

  NO INSTRUMENTATION NECESSARY.

  Produces (structured) tactical proofs skripts; works as filter to large rule sets ...

Isabelle Architecture

- In detail:



Prover Instrumentations (simp-blast-auto)

Generalities

Do we need still proof development ?

- sledgehammer makes advances of ATP technology palpable for Isabelle Users
- However, one should not overestimate them
  - induction,
  - quantifier instantiations, in particular HO instances
  - deep arithmetic reasoning
  - instantiations with non-ground, non-trivial intermediate steps, and
  - strategical case-splits

  remain key decisions in interactive development, where metis, Vampire, smt sometimes gloriously fail.

Generalities

Do we need still proof development ?

- In contrast to most ATP's, which follow an

  „all - or - nothing“ behaviour,

  simp and auto lend themselves to INTERACTIVE development, producing a result following

  „the best that I can“

  (which allows for gradually improving the rewrite-sets or adding intermediate lemmas that were not found automatically).

A Summary of Advanced Proof Methods

- advanced procedures:
  - insert <thmname>

    inserts local and global facts into assumptions
  - induct "φ", induct_tac "φ"

    searches for appropriate induction scheme using type information and instantiates it
  - cases "φ", case_tac "φ"

    searches for appropriate induction scheme using type information and instantiates it

A Summary of Advanced Proof Methods

- advanced automated procedures:
  - simp [add: <thmname>+] [del: <thmname>+]
    [split: <thmname>+] [cong: <thmname>+]
  - auto [simp: <thmname>+] [del ... split ... cong]
    [intro: <thmname>+] [intro [!]: <thmname>+]
    [dest: <thmname>+] [dest [!]: <thmname>+]
    [elim: <thmname>+] [elim[!]: <thmname>+]
  - metis <thmname>+
  - arith <thmname>+

The Simplifier

Supports Rewriting, in particular:

- Rewriting of HO-Patterns,
- Ordered Rewriting
- Conditional Rewriting
- Context - Rewriting
- Automatic Case-Splitting

INSTRUMENTATION NECESSARY, so it is necessary to tell which rule should be used HOW.

The Simplifier

What is a higher-Order Pattern ?

It is a λ-term of form that is:

- constant head, i.e. of the form $c\ t_1 \dots t_n$
- linear in free variables
- All HO Variables occur only in the form:
  $F(x_1 \dots x_n)$ for distinct x_i

Seems very limited ? Well, you can have λ ...

Consider the rule:

$$\forall(\lambda\ x.\ P(x) \wedge Q(x)) = \forall(\lambda\ x.\ P(x)) \wedge (\forall(\lambda\ x.Q(x))$$

The Simplifier

Supports Rewriting, in particular:

- Rewriting of HO-Patterns, i.e. rules of the form:

  <lhs> = <rhs>

  where lhs is a HO-Pattern, where
  lhs is linear in the free variables and
  free variables in rhs occur also in lhs

The Simplifier

Supports Rewriting, in particular:

- Ordered Rewriting:
  There is an implicit wf-ordering on terms.
  Rewriting is only done if the re-written term is smaller.
  Commutativity: $a+b = b+a$

  With a little trickery, one can have ACI rewriting:

disj_comms(2):	$(P \vee Q \vee R) = (Q \vee P \vee R)$
disj_comms(1):	$(P \vee Q) = (Q \vee P)$
disj_ac(3):	$((P \vee Q) \vee R) = (P \vee Q \vee R)$
disj_ac(2):	$(P \vee Q \vee R) = (Q \vee P \vee R)$
disj_ac(1):	$(P \vee Q) = (Q \vee P)$
disj_absorb:	$(A \vee A) = A$
disj_left_absorb:	$(A \vee A \vee B) = (A \vee B)$

The Simplifier

Supports Rewriting, in particular:

- Conditional Rewriting

if_P:	$P \Longrightarrow (\text{if } P \text{ then } x \text{ else } y) = x$
if_not_P:	$\neg P \Longrightarrow (\text{if } P \text{ then } x \text{ else } y) = y$

```
apply(simp cong: if_cong)
```

The Simplifier

Supports Rewriting, in particular:

- Context - Rewriting

HOL.if_cong:

$$b = c \Longrightarrow$$
$$(c \Longrightarrow x = u) \Longrightarrow$$
$$(\neg\, c \Longrightarrow y = v) \Longrightarrow$$
$$(\text{if } b \text{ then } x \text{ else } y) = (\text{if } c \text{ then } u \text{ else } v)$$

HOL.conj_cong:

$$P = P' \Longrightarrow (P' \Longrightarrow Q = Q') \Longrightarrow (P \wedge Q) = (P' \wedge Q')$$

```
apply(simp  cong: if_cong)
```

The Simplifier

Supports Rewriting, in particular:

- Automatic Case-Splitting
  (by a new type of rule which is NOT constant head)

  split_if_asm: $P\ (\text{if } Q \text{ then } x \text{ else } y) = (\neg\ (Q \wedge \neg\ P\ x \vee \neg\ Q \wedge \neg\ P\ y))$

  split_if: $P\ (\text{if } Q \text{ then } x \text{ else } y) = ((Q \longrightarrow P\ x) \wedge (\neg\ Q \longrightarrow P\ y))$

  For any data type (example: Option):

  Option.option.split_asm:

  $P\ (\text{case } x \text{ of None} \Rightarrow f1 \mid \text{Some } x \Rightarrow f2\ x) =$

  $(\neg\ (x = \text{None} \wedge \neg\ P\ f1 \vee (\exists a.\ x = \text{Some } a \wedge \neg\ P\ (f2\ a))))$

  Option.option.split:

  $P\ (\text{case } x \text{ of None} \Rightarrow f1 \mid \text{Some } x \Rightarrow f2\ x) =$

  $((x = \text{None} \longrightarrow P\ f1) \wedge (\forall a.\ x = \text{Some } a \longrightarrow P\ (f2\ a)))$

```
apply(simp  split: split_if_asm split_if)
```

blast and auto

Tableaux Provers

- For Logic and Set theory

- Necessary classification as
    - rule
    - erule
    - drule
    - frule

    - REVISION ELEMENTARY PROOFS

Demo VII

- Some some examples of automatic proof.