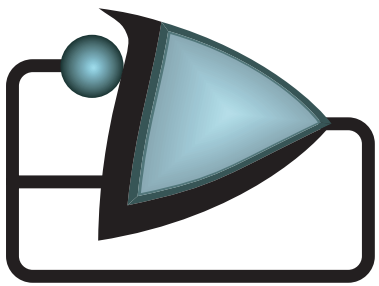
MetaPRL – A Modular Logical Environment

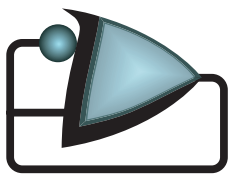
Jason Hickey, Aleksey Nogin, Robert Constable, Brian Aydemir, Eli Barzilay, Yegor Bryukhof, Richard Eaton, Adam Granicz, Alexei Kopylov, Christoph Kreitz, Vladimir Krupski, Lori Lorigo, Stephan Schmitt, Carl Witty, Xin Yu

Caltech, Cornell, CUNY, Moscow State, Newton Labs, …



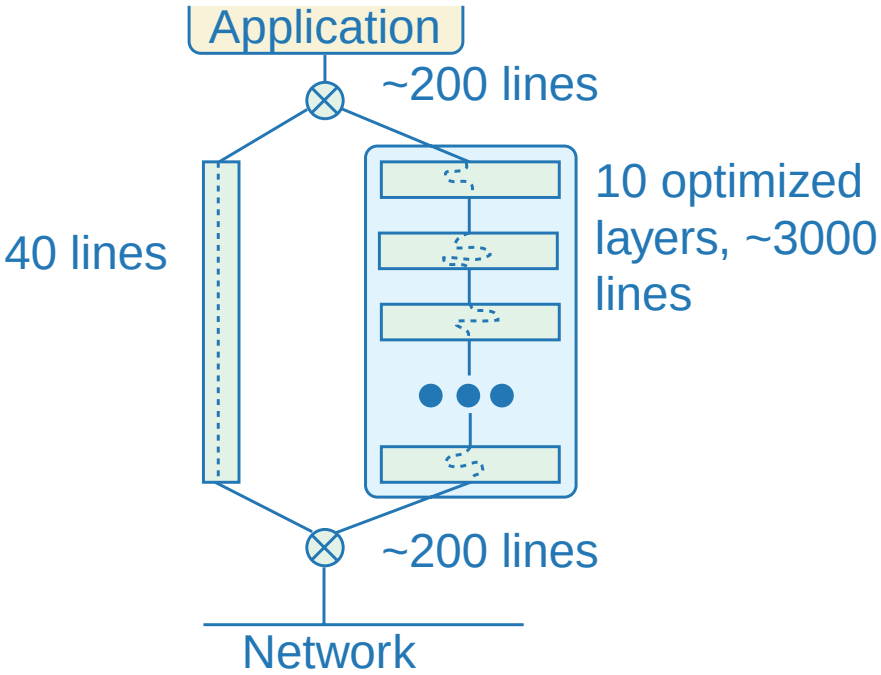
PRL (Proof/Program Refinement Logic) History

- 1970-80s: PRL, Lambda-PRL, NuPRL-3 (Constable et.al.)
- 1992: NuPRL-4
  - *Structured term editing and proof development environment*
  - *Extensive tactic collection and rewrite package (Paul Jackson)*
  - *Logic is the NuPRL type theory, a Martin-LÖf style type theory with many extensions (quotients, recursive types, …)*
- 1998: NuPRL-5
  - *Multi-prover environment*
  - *Extended term/proof editor*
  - *Formal Digital Library for sharing formal knowledge among provers (currently supports PVS,Isabelle, …)*
- Architecture: Common Lisp, Classic ML



1990s: focus on scalability

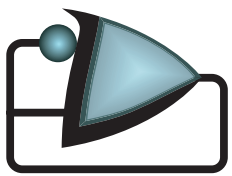
- 1995: Application to the Ensemble Group Communication System (Kreitz, Hayden, Hickey)



Protocols are composed of about 10-30 layers

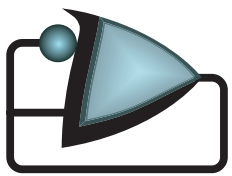
Each layer is about 100-300 lines of OCaml code

NuPRL is used both for verification and optimization (order of magnitude speedup)

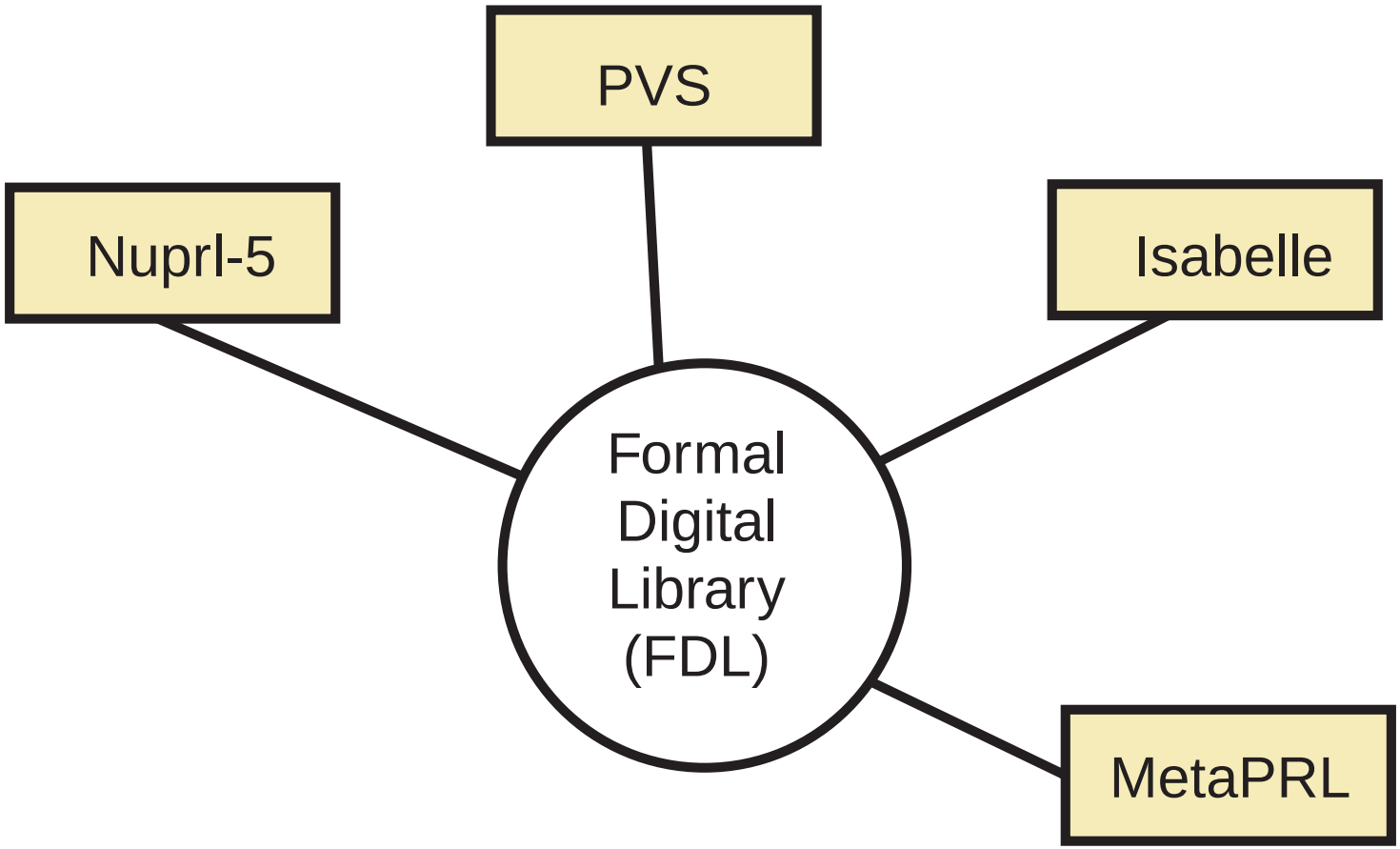


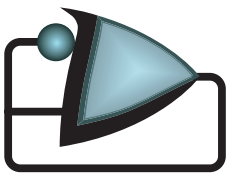
MetaPRL history

- 1995: developed to address problems of scalability (Hickey, NuPRL-Light)
  - *Modular architecture*
    - *Clean separation of trusted logic core from logical* theories
  - *Logical framework and programming environment implemented as an extension to the OCaml compiler (Leroy et.al.)*
  - *Preserves*
    - *NuPRL term structure*
    - *Includes NuPRL type theory as one of the logics*
  - *Adds*
    - *Second-order term matching, rewriting, and refinement*

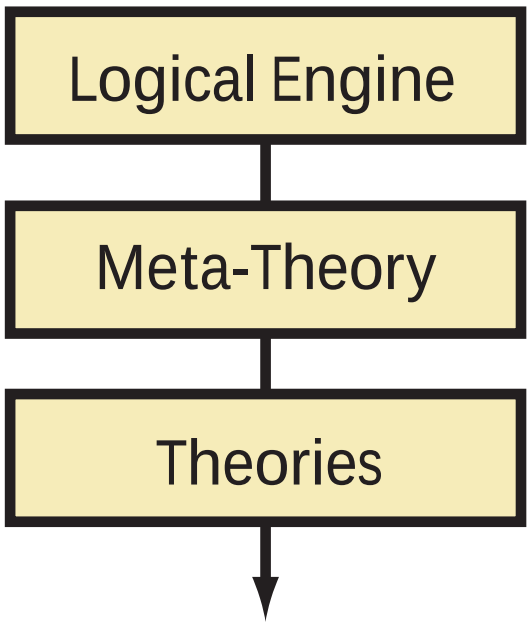


Formal Digital Library

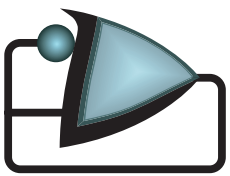




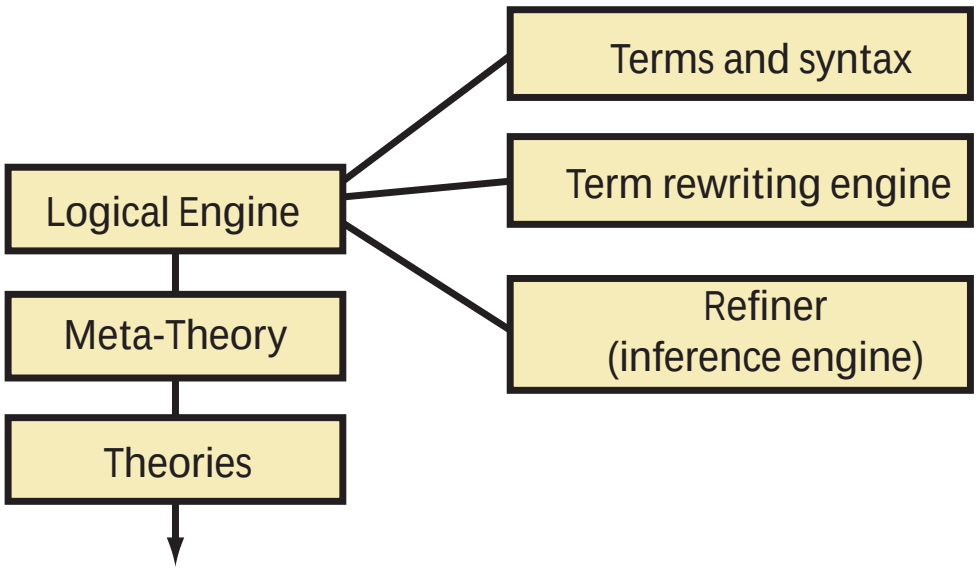
Architecture



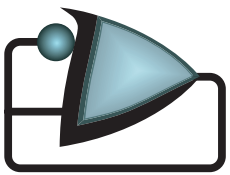
- Logical Engine
  - *defines syntax, rewriting, inference, accounting, checking*
- Meta-theory
  - *Defines a language (OCaml) for directing proofs with tactics and conversions*
- Theories
  - *Logics, type theories, set theories, etc.*
- Note: the user-interface is not an explicit part (though the system provides one)



Architecture II



- Term module
  - *Defines syntax, with binding*
- Rewriting
  - *Defines rewriting with second-order variables and patterns*
- Refiner
  - *Defines inference, accounting, proof management*



An example inference rule

$$\frac{\Gamma, \forall x{:}A.B[x] \vdash a \in A \quad \Gamma, \forall x{:}A.B[x], B[a] \vdash C}{\Gamma, \forall x{:}A.B[x] \vdash C}\ \text{all\_elim}(a)$$

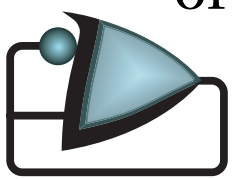
- The rule declaration defines a tactic:

  val all_elim : term → tactic

- The low-level refiner defines an application:

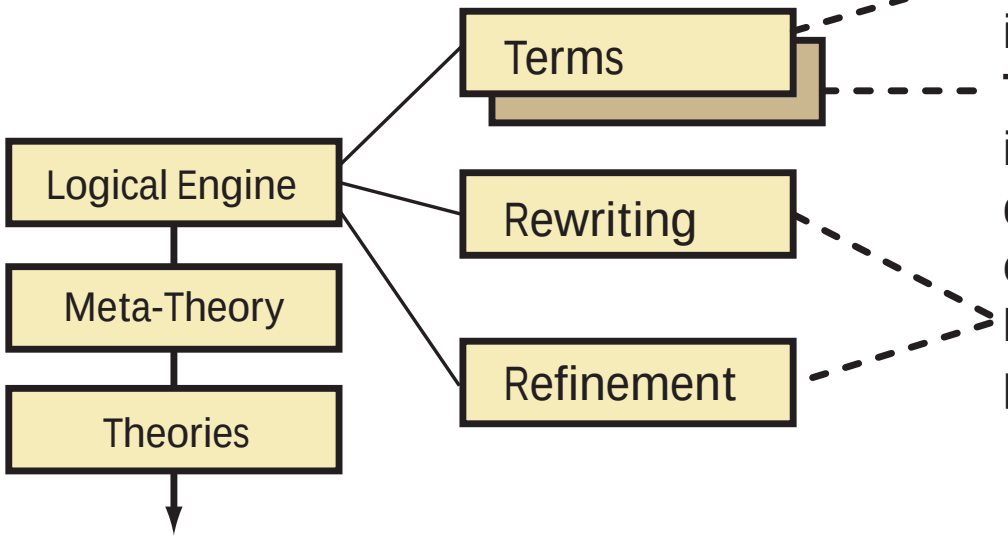
  val refine : tactic → goal → goal list * justification

- The meta-logic uses second-order Horn clauses with second-order meta-variables

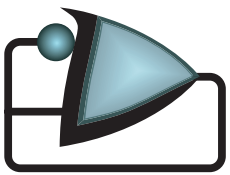


Speed and optimization

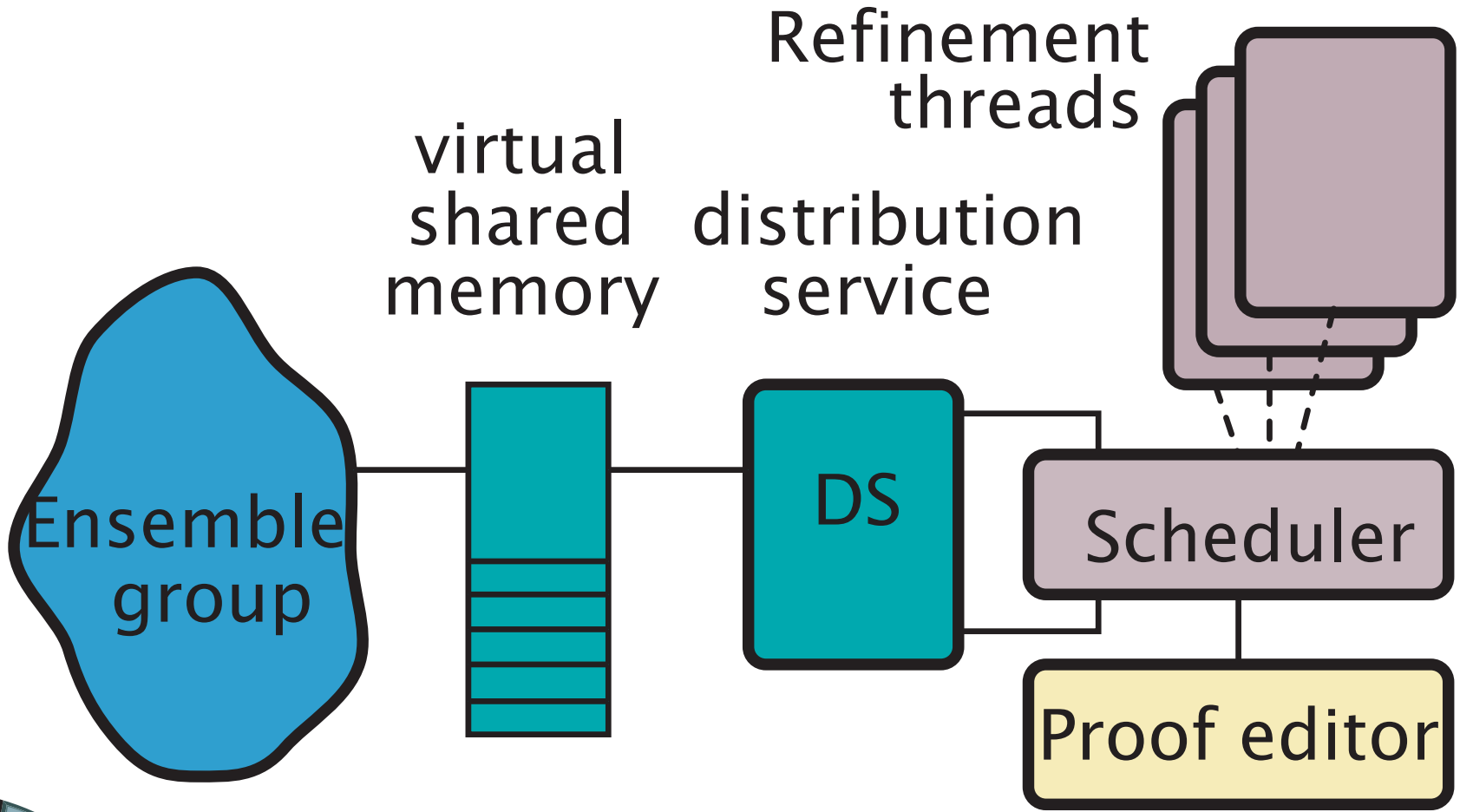
Speed comparison: MetaPRL is consistently x100 faster than NuPRL
-- OCaml vs.Common Lisp/Classic ML (about x5 speed improvement)
-- Highly optimized algorithms and data structures (asymptotic improvement)



Term_std: standard, reference implementation of term
Term_ds: highly optimized implementation of terms with delayed substitution and cached free vars computation
Rewriter: rules and rewrites are precompiled into bytecode programs

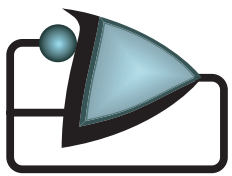


Transparent, fault-tolerant distributed proving



Two main logical concepts

Syntax	`declare` $A \Rightarrow B$ `declare` $\lambda x.b[x]$ `declare` $apply\{f;a\}$
Rewrites	`prim_rw beta:` $apply\{\lambda x.b[x];a\} \longleftrightarrow b[a]$
Rules	`prim imp_intro :` $b[x] : (\Gamma, x{:}A \vdash B) \longrightarrow$ $\Gamma \vdash A \Rightarrow B$ $= \lambda x.b[x]$

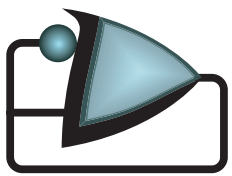


Derived rules (and abstraction)

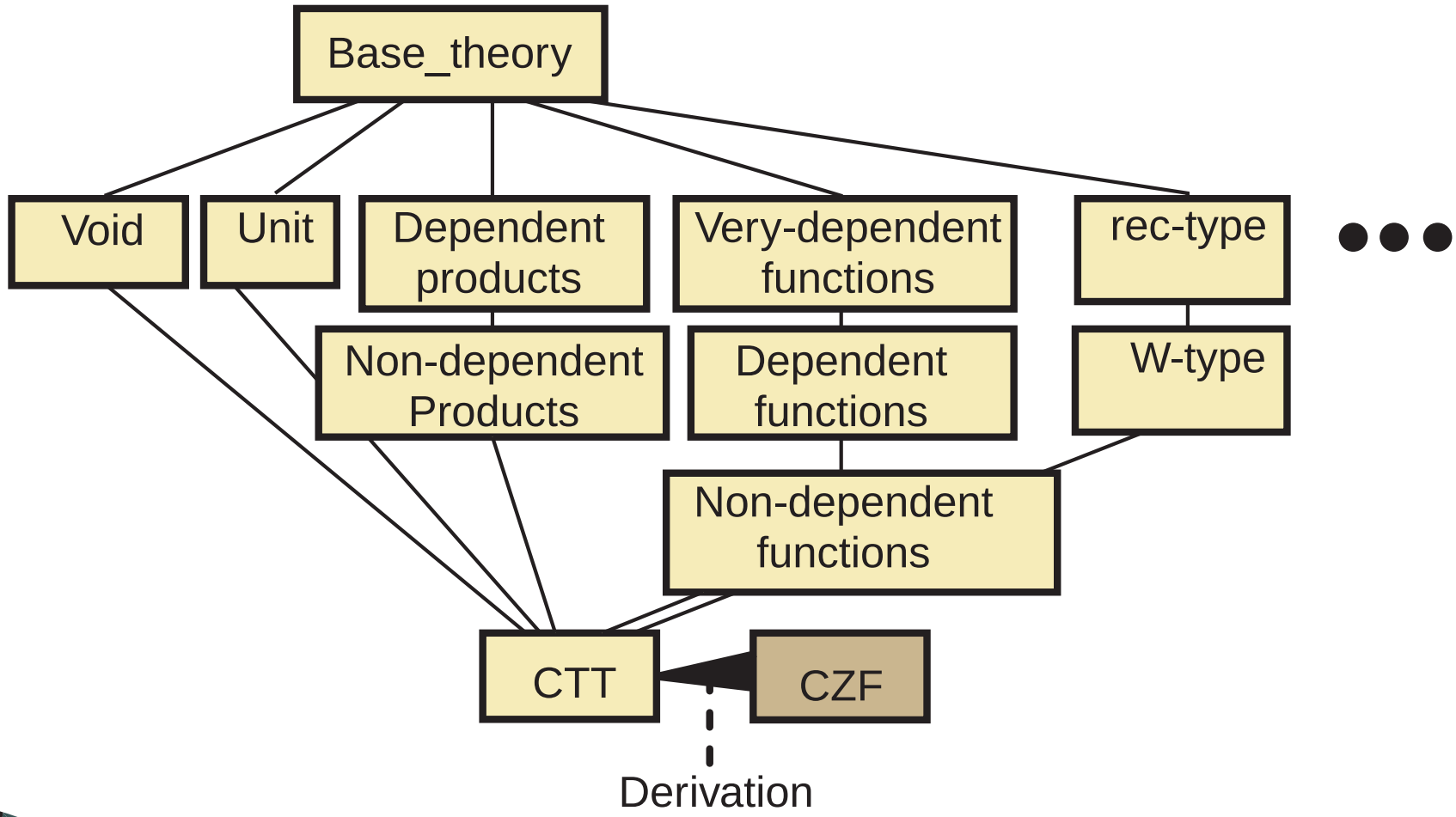
define btree = · · · (μ-type, W-type, etc.)
define leaf = · · ·
define node{a; b} = · · ·

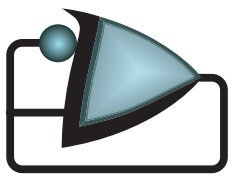
interactive btree_ind :

$$\frac{\Gamma \vdash P[leaf] \quad \Gamma, a: btree, P[a], b: btree, P[b] \vdash P[node(a, b)]}{\Gamma, x: btree \vdash P[x]}$$



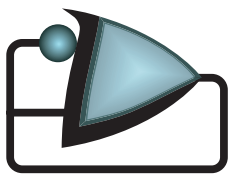
Theory hierarchy





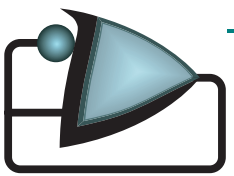
Resources

- Context-sensitive proof automation
- Generic tactics
  - *dT : generic application of introduction and elimination reasoning*
  - *autoT : generic automated reasoning*
- The behavior of the tactic depends on the context
- Resources collect components for generic, context-sensitive, data structures (including dT, autoT)



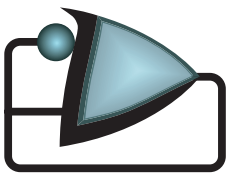
Theories in MetaPRL

- CTT (Computational Type Theory)
  - *Based on the NuPRL type theory (a Martin-Löf style type theory)*
  - *Very-dependent function types (Hickey)*
  - *New,modular account of quotient types (Nogin)*
  - *Extensible dependent records (Kopylov)*
- CZF (Aczel's Constructive ZF set theory)
  - *Fully axiomatized and derived from CTT*
- Programming language logics
  - *A theory of imperative programs with operational semantics*
  - *Theories of compilers and intermediate languages*
  - *…*



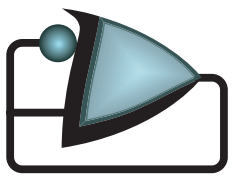
Tactics and decision procedures

- NuPRL implemented arithmetic as a *trusted* decision procedure
  - *MetaPRL axiomatizes arithmetic*
  - *The arith decision procedure is a tactic that produces a proof derived from the axioms (Brukhoff, Nogin, TPHOLs 2003)*
    - *Note Norish's work on adding decision procedures to HOL (TPHOLs 2003)*
- JProver
  - *A highly-optimized first-order prover that produces a proof to be checked in MetaPRL (Kreitz, Schmitt)*



Other ongoing work

- Compilers (e.g. Hickey et.al. Merlin 2003)
  - *A complete compiler for a small ML-like language*
  - *CPS, closure conversion, optimization, code generation*
  - *The compiler is implemented as a set of rewrites (using HOAS)*
  - *The trusted core is about 200 rewrites*
  - *Generates functional(!) assembly code for the Intel x86 platform*
  - *Extremely easy to implement because of all the usual things: built-in substitution, generic rewriting strategies, perservation of scoping, etc…*



Summary

- MetaPRL is designed to be fast, flexible, modular
  - *In part, MetaPRL is a logical toolkit for formal methods (terms, rewriting, decision procedures, etc)*
  - *It serves as part of the FDL, sharing proofs with systems including Nuprl, PVS, Isabelle, with planned support for others like Coq, HOL, Larch, etc*
- It is a logical framework
  - *Emphasizing organization and relation of logics*
- It is designed as a logical programming environment
  - *Clean integration with the OCaml programming language*
- It is open-source, freely available
  - *See metaprl.org*