Designing Reliable, High-Performance Networks

...with the NUPRL **L**ogical **P**rogramming **E**nvironment

Christoph Kreitz

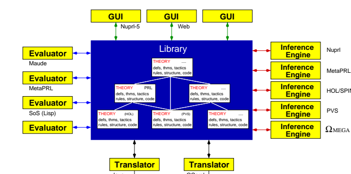
Department of Computer Science, Cornell University

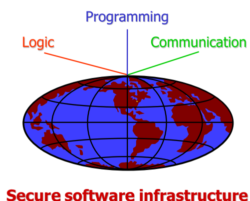
Ithaca, NY 14853

Formal methods tools are most sucessful when engaged at early stages of system design

- **Great potential**
  - Clarifying critical design concepts
  - Linking abstract and concrete specifications
  - Detecting subtle errors in design and prototype code
  - Generating code of components from specifications
  - Improving system performance
- **Requires**
  - Expressive formal language
  - Knowledge base of formalized facts about systems concepts
  - Proof environment capable of integrating different reasoning techniques
  - Collaboration between systems and formal methods experts in real applications

Type Theory





The Nuprl LPE meets these requirements

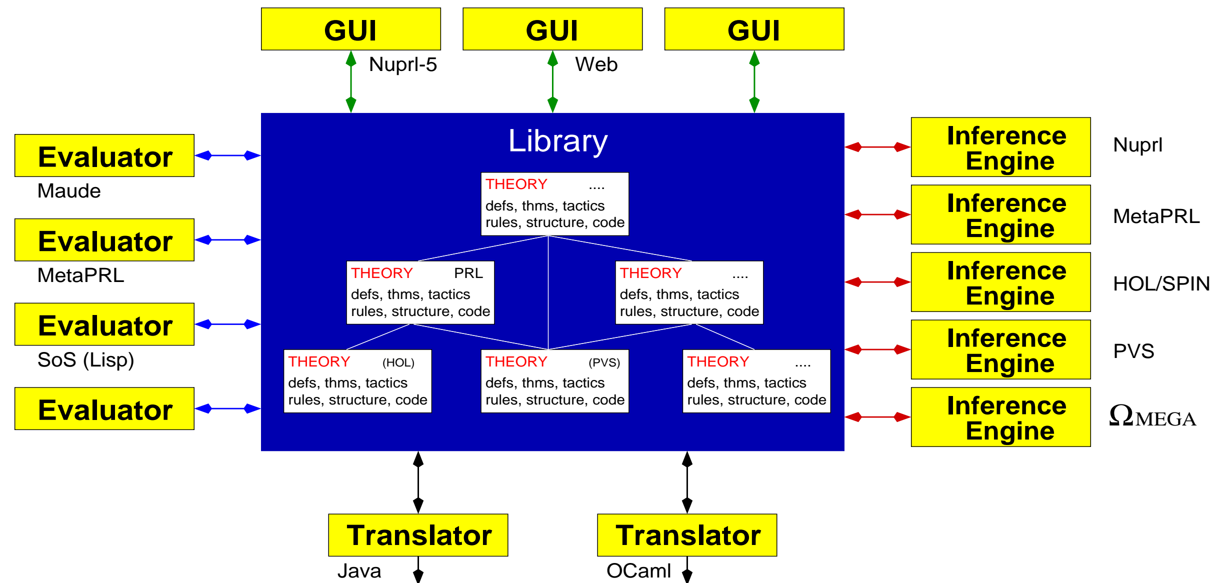
Nuprl's Formal Logic: Computational Type Theory

- **Logic for constructive reasoning**
- **Open-ended, expressive type system**
  - Function, product, disjoint union, Π- & Σ-types, atoms ⇝ programming
  - Integers, lists, inductive types ⇝ inductive definition
  - Propositions as types, equality type, void, top, universes ⇝ logic
  - Subsets, subtyping, quotient types ⇝ mathematics
  - (Dependent) intersection, union, records ⇝ modules, program composition

  New types can/will be added as needed
- **Top-down refinement calculus** ⇝ interactive proof development
  - Sequent calculus + computation rules + extract terms ⇝ program development
- **Expressions separate from their types** ⇝full λ-calculus

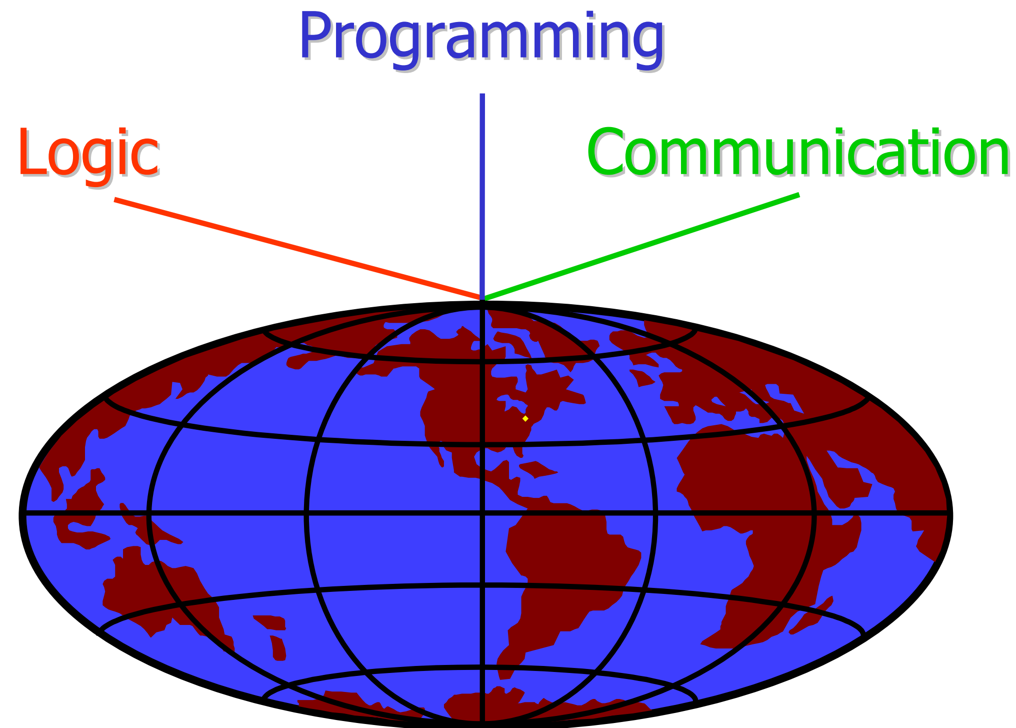
  ... but must be typeable *in proofs* ⇝ "total" functions
- **Uniform internal notation + display forms** ⇝ "free syntax"
- **User-defined extensions possible**

Nuprl's Automated Reasoning Environment



- **Interactive proof development**
  - Supports program extraction (synthesis) and evaluation
  - Proof automation through tactics & decision procedures
  - Highly customizable: language extensions, term display, system structure,. . .
- **Cooperating processes centered around knowledge base** (CADE 2000)
  - Large library of formal algorithmic knowledge
  - Asynchronous, distributed & collaborative theorem proving
  - Multiple user interfaces: proof editor, structured term editor, web browser
  - External proof engines: MetaPRL, JProver (TPHOLs 2000, IJCAR 2001)

Application: Reliable, High-Performance Networks



Secure software infrastructure

- **Ensemble Group Communication Toolkit**
  – System optimization and verification, formal component design
- **MediaNet Stream Computation Network** (ongoing)
  – Validation of real-time schedules wrt. resource limitations

The Ensemble Group Communication Toolkit

Modular group communication system

– Developed by Cornell's System Group (Ken Birman)

– Used commercially (BBN, JPL, Segasoft, Alier, Nortel Networks)

Architecture: stack of micro-protocols

– Select from more than 60 micro-protocols for specific tasks

– Modules can be stacked arbitrarily

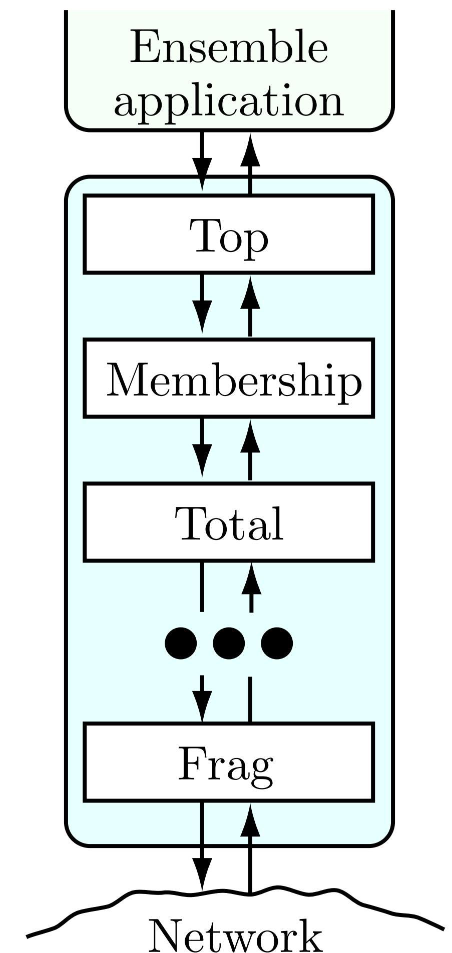
– Modeled as state/event machines

Implementation in Objective Caml (INRIA)

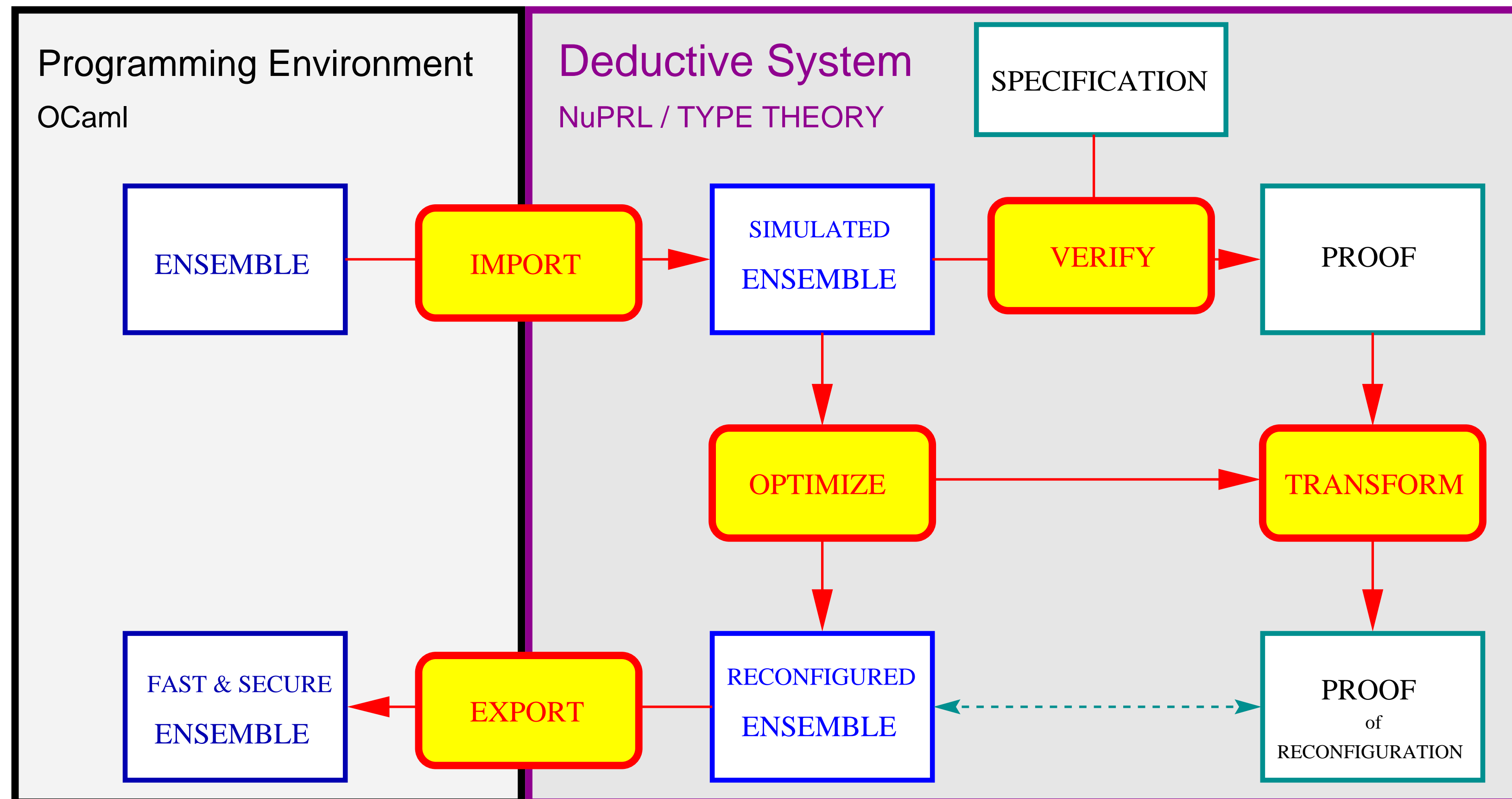
– Easy maintenance (small code, good data structures)

– Mathematical semantics, strict data type concepts

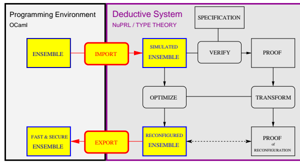
– Efficient compilers and type checkers



Formal Reasoning about Ensemble in Nuprl

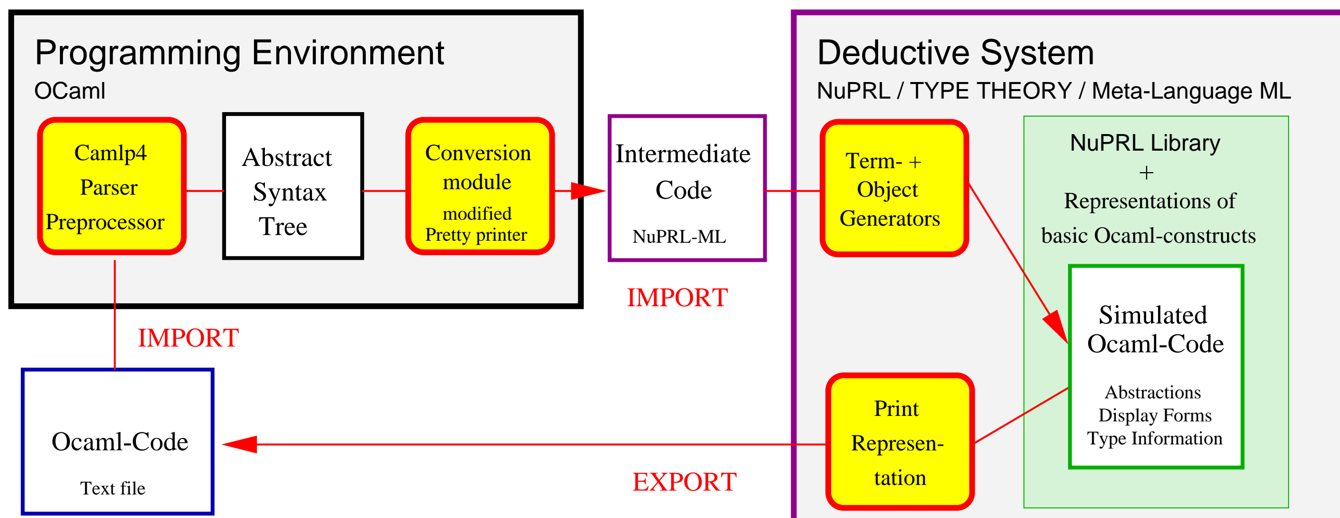


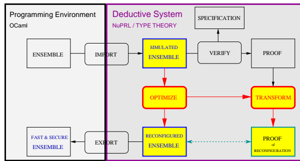
- Formalize semantics of OCaml (CADE 1998, ...)
- Optimize performance of configured systems (TACAS 1999, SOSP 1999)
- Verify protocol components and system configurations (TACAS 1999)
- Formally design and verify new protocols (DISCEX 2001, TPHOLS 2001)



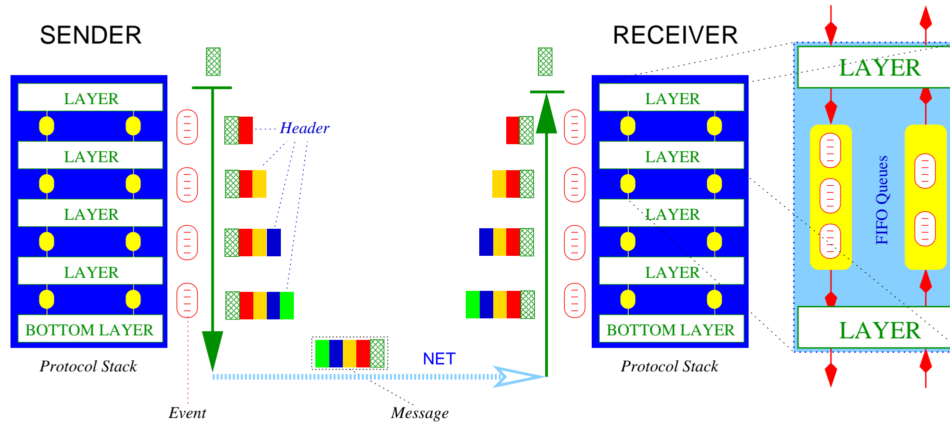
EMBEDDING ENSEMBLE'S CODE INTO NUPRL
ENABLE FORMAL REASONING ON OCAML LEVEL

- **Type-theoretical semantics of OCAML**
  - Pattern matching, exceptions, references, modules, ... $\mapsto$ type theory
- **Implementation in NUPRL**
  - OCAML semantics $\mapsto$ abstractions   OCAML syntax $\mapsto$ display forms
- **Programming logic for OCaml**
  - Derived inference rules for reasoning about OCAML code
- **Import and Export mechanisms**
  - Actual system code available for formal reasoning in NUPRL





Optimization of Protocol Stacks

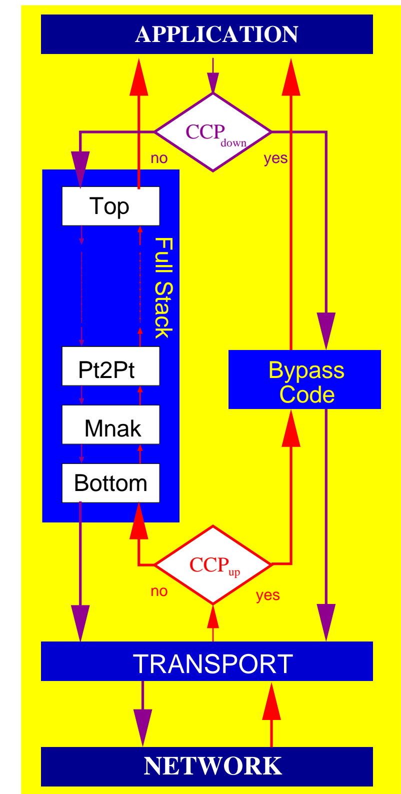


Protocol stacking creates performance loss

– redundancy, internal communication, large message headers

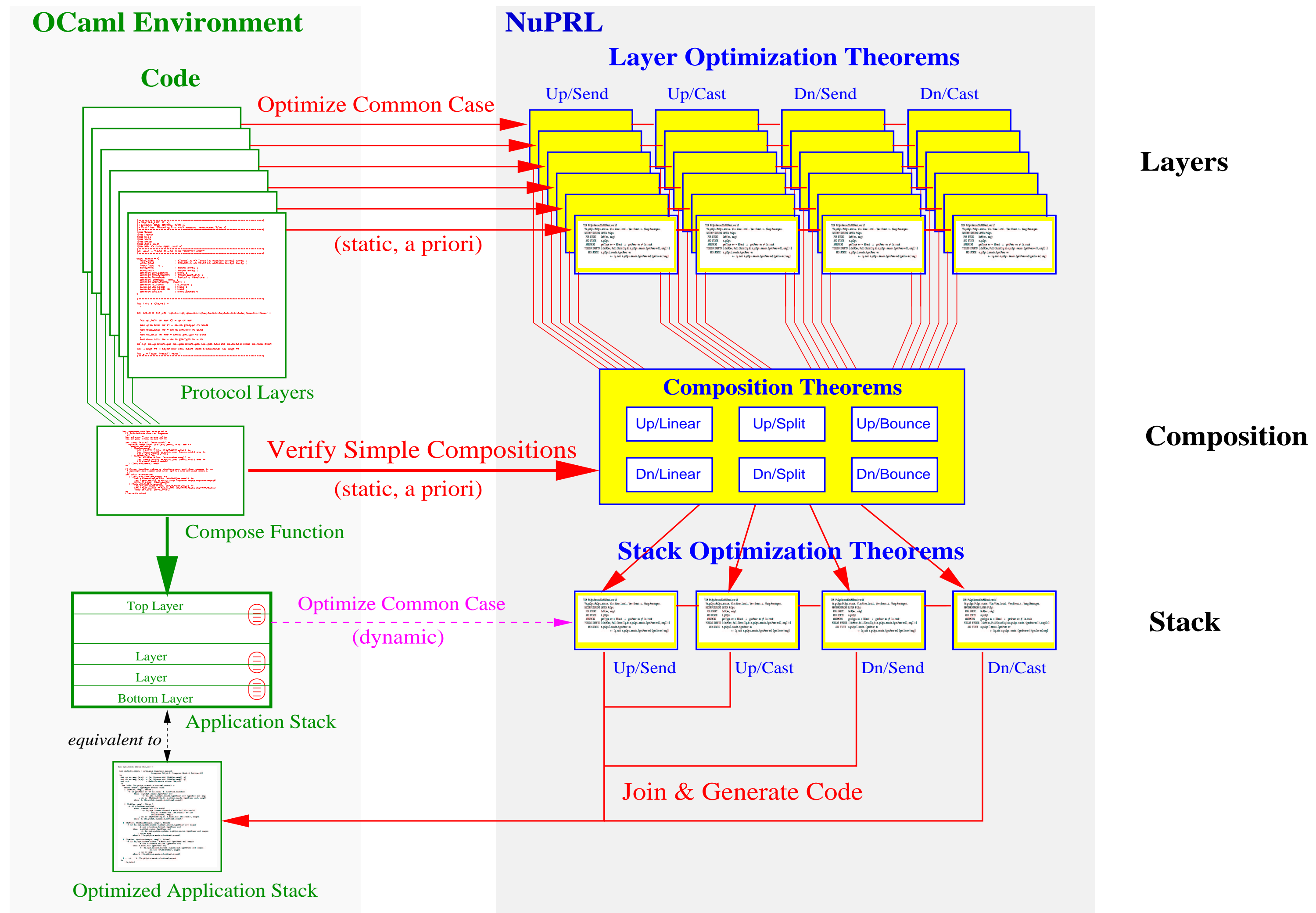
Possible optimizations

- **Fast-path for common execution sequences**
  - Identify Common Case as Predicate
  - Analyze path of events through stack
  - Isolate code for fast-path and generate bypass
  - Insert CCP as runtime switch
- **Header compression for common messages**



Need formal reasoning tools to do this correctly

Methodology: Compose Optimization Theorems

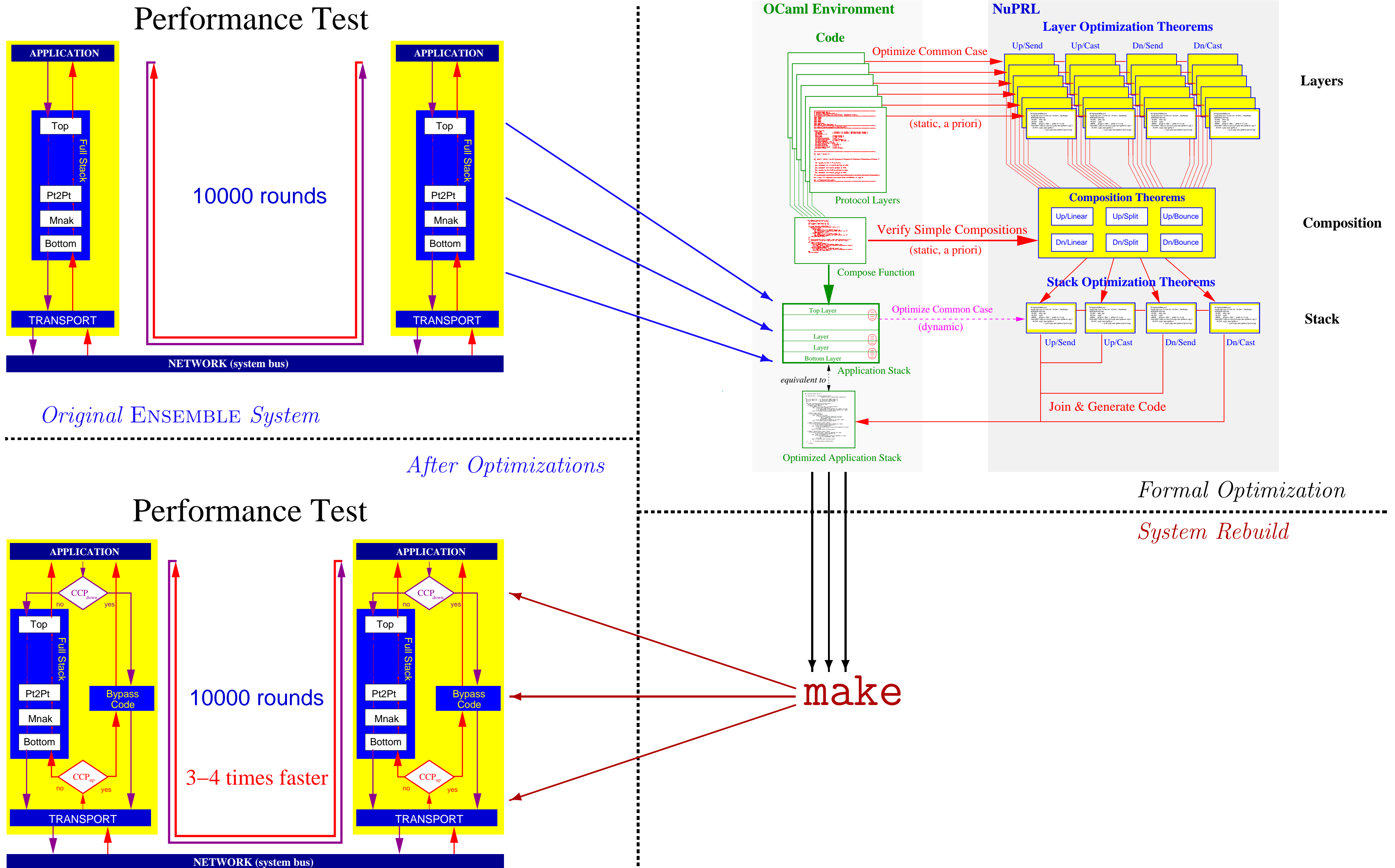


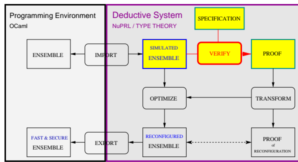
1. Use known optimizations of micro-protocols — A priori: Ensemble + Nuprl experts
2. Compose into optimizations of protocol stacks — automatic: application designer
3. Integrate message header compression — automatic: ⋮
4. Generate code from optimization theorems and reconfigure system — automatic: ⋮

Fast, error-free, independent of programming language — **speedup factor 3-10**

DEMO: Optimizing a 24-layer Protocol Stack

Top :: Heal :: Switch :: Migrate :: Leave :: Inter :: Intra :: Elect :: Merge :: Slander :: Sync :: Suspect :: Stable :: Vsync :: Partial_appl :: Total :: Collect :: Local :: Frag :: Pt2ptw :: Mflow :: Pt2pt :: Mnak :: Bottom





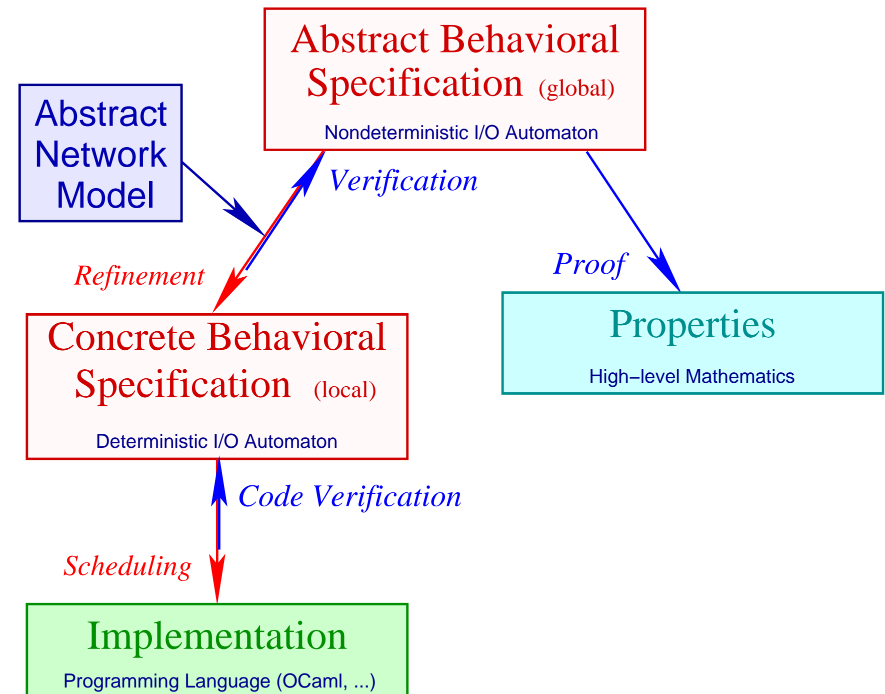
Verification and Synthesis
Link four levels of abstraction

Formalize system specification and code

e.g. *"Messages are received in the same order in which they were sent"*

- *"Messages may be appended to global event queue and removed from its beginning"*
- *"Messages whose sequence number is too big will be buffered"*
- Ensemble *module* `Pt2pt.ml`*: 250 lines of* OCaml *code*

All levels represented in type theory



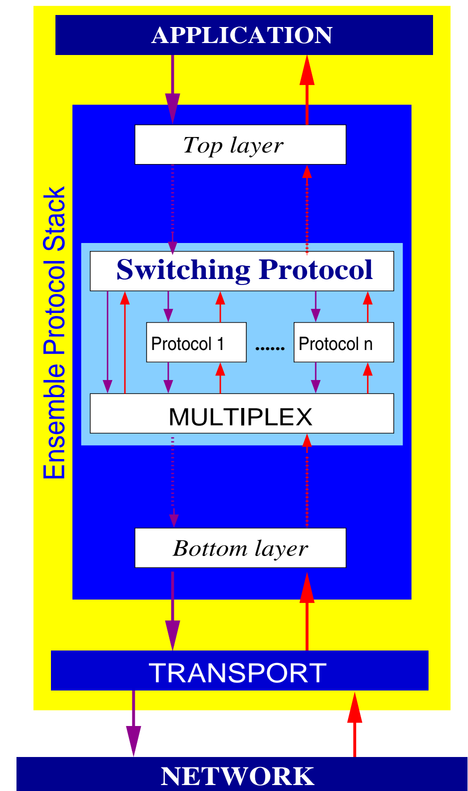
Verification methodology

- Verify component specifications
  (benign assumptions)
- Verify systems by composition
  (IOA-composition preserves safety properties)
- Weave aspects
- Verify code

Reasoning direction can be reversed into network synthesis

Formal Design of Adaptive Systems

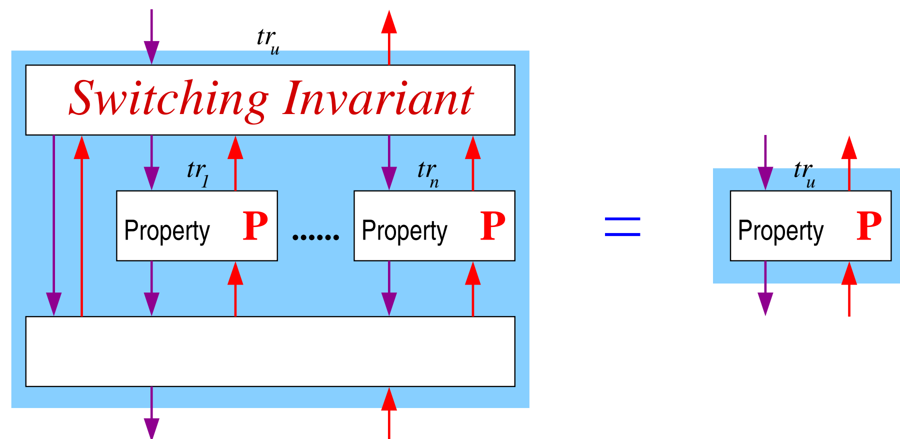
- **Make systems adapt safely to run-time dynamics**
  - On-line upgrading, security, performance
  - Difficult to design correctly *(distributed migration?)*
- **Generic switching protocol**
  - Construct hybrid protocols from simpler ones
  - Normal mode: interact with one protocol
  - Switching mode: deliver old messages, buffer new ones
- **Correctness issues**
  - *What kind of protocols are switchable at all?*
    - Reliability? Integrity? Confidentiality? Total Order? ...
  - *What code invariant guarantees that switchable properties are preserved?*



LPE verification answers both questions

Verifying the Correctness of Switching

- **Develop Formal Model of Communication**
  - Communication property P: predicate on traces (lists of Send/Deliver events)
  - Meta-property: relation between traces that preserves properties
- **Characterize switchable properties by meta-properties**




```
switchable(P)
  ≡   P refines Causality
    ∧ P refines No-replay
    ∧ P safety property
    ∧ P asynchrony property
    ∧ P delayable property
    ∧ P send-enabled property
    ∧ P memoryless property
    ∧ P composable property₃
```

- **Characterize switch invariant between tr_u and tr_1,..,tr_n**
  - tr_u results from joint trace by swapping events with different origin
  - Messages sent by different protocols must be delivered in same order
- **Prove that all switchable properties will be preserved**

Correct implementation and use of switch

Lessons learned

- **Results**
  - Type theory expressive enough to formalize today's software systems
  - Formal optimization can significantly improve practical performance
  - Formal verification reveals errors even in well-explored designs
  - Formal design reveals hidden assumptions and limitations for use of protocols

  $\Rightarrow$ **Nuprl is capable of supporting real design at reasonable pace**

- **Ingredients for success ...**
  - Implementation language with precise semantics
  - Employing formal methods at every design stage
  - Collaboration between systems and formal reasoning groups
  - Formal models of: communication, I/O-automata, programming language
  - Knowledge-based approach: large library of algorithmic knowledge
  - Great colleagues! Stuart Allen, Mark Bickford, Ken Birman, Robert Constable, Richard Eaton, Xiaming Liu, Lori Lorigo, Robbert van Renesse

Future Challenges

- **Advanced reasoning environment**
  - Interactive library of formal algorithmic knowledge
    - Archival capacities (documentation & certification, version control)
    - A variety of justifications (levels of trust)
    - Creation of formal and textual documents
    - Meta-reasoning and reflection
  - Embed external library contents
  - Connect additional proof engines: PVS, HOL, MinLog, ...

  Improve cooperation between research groups

- **Learn more from applications**
  - Reasoning about real-time, embedded systems and stream computations
    - verified self-adaptation to changing resource constraints
  - Support programming languages with less clean semantics
  - Invert reasoning direction: from verification towards network synthesis